



MOTOROLA

M6809EXOR(D1)

M6809  
**EXORciser**

User's Guide



**MICROSYSTEMS**

M6809EX0R(D1)

SEPTEMBER 1979

M6809

EX0Rciser

USER'S GUIDE

The information in this document has been carefully checked and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies. Furthermore, Motorola reserves the right to make changes to any products herein to improve reliability, function, or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others.

EX0Rciser®, EX0Rdisk, and EXbug are trademarks of Motorola Inc.

First Edition

©Copyright 1979 by Motorola Inc.



## TABLE OF CONTENTS

	<u>Page</u>	
<b>CHAPTER 1</b>	<b>GENERAL INFORMATION</b>	
1.1	INTRODUCTION	1-1
1.2	FEATURES	1-1
1.3	SPECIFICATIONS	1-2
1.4	EQUIPMENT SUPPLIED	1-2
1.5	GENERAL DESCRIPTION	1-3
1.5.1	EXORciser Memory Parity	1-3
1.5.2	Dual Map Concepts	1-5
1.5.3	Second Level Interrupt Feature	1-7
1.5.4	Dynamic System Bus	1-10
<b>CHAPTER 2</b>	<b>INSTALLATION INSTRUCTIONS AND HARDWARE PREPARATION</b>	
2.1	INTRODUCTION	2-1
2.2	UNPACKING INSTRUCTIONS	2-1
2.3	INSPECTION	2-1
2.4	INSTALLATION INSTRUCTIONS	2-1
2.5	DATA TERMINAL SELECTION AND CONNECTIONS	2-2
2.5.1	RS-232C Interconnections	2-2
2.5.2	20mA Current Loop Interconnections	2-2
2.6	PREPARATION OF SYSTEM MODULES	2-2
<b>CHAPTER 3</b>	<b>OPERATING INSTRUCTIONS</b>	
3.1	INTRODUCTION	3-1
3.2	SWITCHES AND INDICATORS	3-1
3.2.1	Front Panel Switches and Indicators	3-1
3.2.2	Switches on the DDebug Module	3-2
3.3	INITIALIZATION	3-3
3.3.1	Power ON/OFF Procedures	3-3
3.3.2	Baud Rate Selection	3-4
3.3.3	Start-Up Procedures	3-4
3.4	USING THE DUAL MEMORY MAP	3-5
3.5	ADDRESS SELECTION	3-5
3.6	EXbug COMMANDS	3-6
3.6.1	Four-Character Commands	3-8
3.6.2	Single Character Commands	3-14
3.6.2.1	Register Display and Change	3-14
3.6.2.2	Program Execution Control	3-16
3.6.2.3	Program Execution	3-19
3.6.2.4	Memory Parity Control	3-20
3.6.2.5	I/O Control	3-22
3.6.2.6	Memory Search	3-23
3.6.2.7	Miscellaneous	3-26
3.6.3	Memory Change	3-27
3.6.3.1	Adding EXbug Commands	3-28
3.7	EXbug SUBROUTINES AND ENTRY POINTS	3-29
<b>CHAPTER 4</b>	<b>SYSTEM DEVELOPMENT USING EXORciser</b>	
4.1	INTRODUCTION	4-1
4.2	THE EXORciser IN SYSTEM DEVELOPMENT	4-1
4.3	PERIPHERAL INTERFACING	4-1
4.4	PROCEDURE FOR DESIGN	4-3
4.5	EXORciser CONFIGURATION	4-5
4.6	SYSTEM ADDRESS SELECTION	4-5
4.7	SECOND LEVEL INTERRUPT	4-5

	<u>Page</u>	
CHAPTER 4 (cont'd)		
4.8      MEMORY ASSIGNMENTS	4-6	
4.9      EXORciser CONFIGURATION FOR SYSTEM EMULATION	4-6	
4.10     TESTING PROTOTYPE OR PRODUCTION SYSTEMS	4-7	
4.11     PRECAUTIONS WHEN USING THE USER SYSTEM EVALUATOR	4-9	
4.12     SYSTEM EVALUATION AND DEBUG PROCEDURES	4-9	
4.12.1    Memory Loader	4-9	
4.12.2    Abort Function	4-9	
4.12.3    Default Debug Offset	4-9	
4.12.4    Memory Change Function	4-10	
4.12.5    Breakpoint, Trace, and Halt-on-Address/ Scope Sync Functions	4-10	
4.12.5.1    Breakpoints	4-11	
4.12.5.2    Trace	4-11	
4.12.5.3    Halt-on-Address	4-11	
4.12.5.4    Scope Sync	4-12	
4.12.6    Error Correction	4-12	
4.13     SOFTWARE DEVELOPMENT USING EXORciser	4-12	
CHAPTER 5      THEORY OF OPERATION		
5.1     INTRODUCTION	5-1	
5.2     BASIC EXORciser BLOCK DIAGRAM DESCRIPTION	5-1	
APPENDIX A     EXORciser BUS DESCRIPTION AND SPECIFICATIONS		A-1
APPENDIX B     EXbug 2.1 PROGRAM FOR EXORciser	B-1	
APPENDIX C     PERIPHERAL REQUIREMENTS FOR EXORciser OPERATION	C-1	
APPENDIX D     USE OF THE ASR33 TELETYPEWRITER WITH EXORciser	D-1	
APPENDIX E     THE RS-232C STANDARD	E-1	
APPENDIX F     TI TERMINAL DESCRIPTION	F-1	
APPENDIX G     EXECUTIVE MAP -- USER MAP INTERFACE	G-1	
APPENDIX H     COMPATIBILITY AMONG M6809 EXORciser, EXORciser II, and M6800 EXORciser I MODULES	H-1	
APPENDIX I     DIRECT MEMORY ACCESS ON THE DEVELOPMENT SYSTEM BUS	I-1	
LIST OF ILLUSTRATIONS		
FIGURE 1-1.    EXORciser Simplified Block Diagram	1-4	
1-2.    EXORciser Dual Memory Map	1-6	
1-3.    Interrupt Vectors in User Map	1-7	
1-4.    Interrupt Vectors for Executive Map and Single Map Mode	1-8	
3-1.    EXORciser Front Panel Switches and Indicators	3-1	
3-2.    Toggle Switches on DEbug Module	3-2	
3-3.    PRNT Example	3-9	
3-4.    Tape Format	3-10	
3-5.    PNCH Example	3-11	
3-6.    LOAD Example	3-12	
3-7.    VERF Example	3-13	
3-8.    SRCH Example	3-13	
3-9.    MDOS Line Printer Driver	3-24	
4-1.    System Designing and Verifying Procedure	4-2	
4-2.    EXORciser, the Development Tool	4-4	
4-3.    Using Memory Change to Calculate a Relocatable Address	4-10	
5-1.    EXORciser Simplified Block Diagram	5-3	

LIST OF TABLES

TABLE 1-1.	EXORciser Specifications	1-2
1-2.	Second Level Interrupt Rules	1-9
1-3.	Second Level NMI Rules	1-9
1-4.	Second Level SWI Rules	1-10
1-5.	IRQ, FIRQ, SWI2, and SWI3 Rules	1-10
3-1.	EXbug Commands	3-6
3-2.	Dual Map Mode Second Level SWI Options	3-25
3-3.	EXbug Routines	3-30



## CHAPTER 1

### GENERAL INFORMATION

#### 1.1 INTRODUCTION

The M6809 EXORciser Development System is the basic tool for designing and developing M6809 microprocessor-based systems. It is an extremely powerful and easy-to-use development system that has been designed to be highly user-oriented, reducing system development time and cost. The M6809 EXORciser incorporates several advanced features, including Dual Memory Map mode of operation and the ability to develop higher performance systems using the MC68A and MC68B series parts (1.5 MHz and 2.0 MHz, respectively).

Documentation relevant to the M6809 EXORciser Development System is separated into two major categories: the M6809 EXORciser User's Guide and various supplemental manuals. This manual (the M6809 EXORciser User's Guide) is the primary guide for using the entire M6809 EXORciser system, and provides general information, installation and set-up instructions, operating instructions, guidelines to follow for system development, and theory of operation for the entire system. It also provides a thorough description of each M6809 EXbug command, as well as all of the entry points for user access to the various subroutines. Specifically excluded from this manual are operating instructions for development system modules requiring special operating procedures (such as the PROM Programmer III Module or System Performance Monitor). Operating instructions for these modules are provided in their accompanying User's Guide.

Supplemental manuals are provided with this User's Guide. These supplements offer detailed information on each of the standard and optional modules used with the M6809 EXORciser, including schematic diagrams, detailed theory of operation, and a complete parts list.

#### NOTE

THROUGHOUT THIS USER'S GUIDE, WHEREVER  
EXORciser, EXbug, MPU MODULE, OR DDebug  
MODULE TERMS ARE USED, THE TYPE NUMBER  
M6809 IS IMPLIED, UNLESS OTHERWISE SPECIFIED.

#### 1.2 FEATURES

The features of the EXORciser include:

- Versatile and easily expandable modular assembly tool used to evaluate and debug the user's system hardware and software.
- Dual Memory Map mode of operation.
- Selectable clock speeds of up to 2.0 MHz.
- 32K of Development System Random Access Memory (RAM).
- 8 selectable baud rates from 110 to 9600 baud.
- A single RS-232C compatible serial communications interface.
- A chassis containing a 14-card motherboard and the necessary +5 Vdc and +12 Vdc power supplies.

### 1.3 SPECIFICATIONS

Table 1-1 identifies the basic EXORciser specifications.

TABLE 1-1. EXORciser Specifications

CHARACTERISTICS	SPECIFICATIONS
Power Requirements	95-135/205-250 VAC 47-420 Hz 250 W
Word Size	
Data	8 bits
Address	16 bits
Instruction	Up to 5 bytes
Memory Capability	65,536 bytes (maximum) in each map
Instructions	59 variable length
System Speed	Not over 2.0 MHz
Interrupts	3 hardware and 3 software
Data Terminal Interface Characteristics	
Baud Rates (Jumper Selectable)	110, 150, 300, 600, 1200, 2400, 4800, and 9600
Signal Characteristics	TTY (20 mA neutral current loop) or EIA RS-232C compatible
Reader Control signal	Control signal for TTY devices modified for external control
Operating Temperatures	0 to 55°C

### 1.4 EQUIPMENT SUPPLIED

The EXORciser is shipped with the chassis (including power supply, motherboard, and cover), MPU Module, DEbug Module, and 32K bytes of either static or dynamic RAM (depending on customer preference). The Macro Assembler/Linking Loader and Text Editor are also included in a media specified by the customer (either paper tape, cassette, or diskette). This manual, together with the appropriate User's Guides, is also included.

## 1.5 GENERAL DESCRIPTION

The EXORciser consists of a chassis with power supply and two essential hardware modules -- the MPU Module and the DEbug Module. Optional memory modules and peripheral modules may be configured in the system, as required by the user. The MPU Module provides the central time base for the system. The 2 MHz M6809 microprocessor chip is located on this module. The DEbug Module includes the system terminal interface and 3K bytes of firmware referred to as EXbug. This firmware is organized into a 2K ROM and a 1K ROM so that all I/O utility routines reside in the 1K part. These routines are available to the user, and the single 1K ROM may be replaced by the user in situations where special I/O routines are desirable. EXbug represents the basic system monitor for the EXORciser, through which the user may enter commands to examine and to change memory, to trace program execution, or to load additional systems software. All EXbug commands are described in Chapter 3 of this manual.

Figure 1-1 shows a block diagram of the EXORciser configuration, with each of the large boxes representing a plug-in module. Along the left of the diagram, it can be seen that the configuration includes at least two systems that timeshare the same MPU Module -- the debug system and the user system (the one under development). In fact, if you include the Software Development System to be discussed later, there are really three systems encompassed by the EXORciser. The debug system is a complete operating M6809 system, which includes a Microprocessor, RAM, ROM, and I/O. It is a specialized microcomputer with unique hardware and software features to aid the designer.

MDOS is the optional software package that provides a complete disk operating and file management system to the EXORciser. MDOS is described in detail in the MDOS Reference Manual, and requires at least 16K bytes of RAM.

In order for the user to be able to take full advantage of the system development aids provided, the following system concepts must be fully understood:

- Memory Parity
- Dual Map
- Second Level Interrupt Vectors
- Dynamic System Bus

### 1.5.1 EXORciser Memory Parity

Both the Static RAM and Dynamic RAM Modules, available for 2 MHz operation with the EXORciser, provide parity detection that interfaces with the system bus. The DEbug Module may be enabled via an EXbug command to monitor this parity error signal. If enabled, the occurrence of a parity error will cause a non-maskable interrupt (NMI) to be generated, and EXbug will print a parity error message to warn the system user of the fault. While writing to the disk, a memory parity error will generate a disk time-out error instead of a parity error message. A parity error generated while examining User Map memory from EXbug will result in EXbug switching to the Executive mode.

Because of the extensive flexibility for configuring hardware in an EXORciser, the system cannot automatically initialize all memories to the proper parity at power start-up without some direction from the user. As a result, EXbug starts up following POWER-ON or RESTART by refusing to accept any parity errors from memory modules. (Enabling and disabling parity does not involve modifying hardware; i.e., the parity detection logic is always active on the memory modules.)

In order to properly initialize the system to take advantage of memory parity, the user must first initialize all locations in the memory modules (EXbug command "I") and then must turn on the enabling indicator in the EXbug firmware (command ";;"). Chapter 3 contains a complete explanation of each of these commands. If a parity error occurs and the user has enabled this feature, the message "PARITY" will be printed by EXbug.

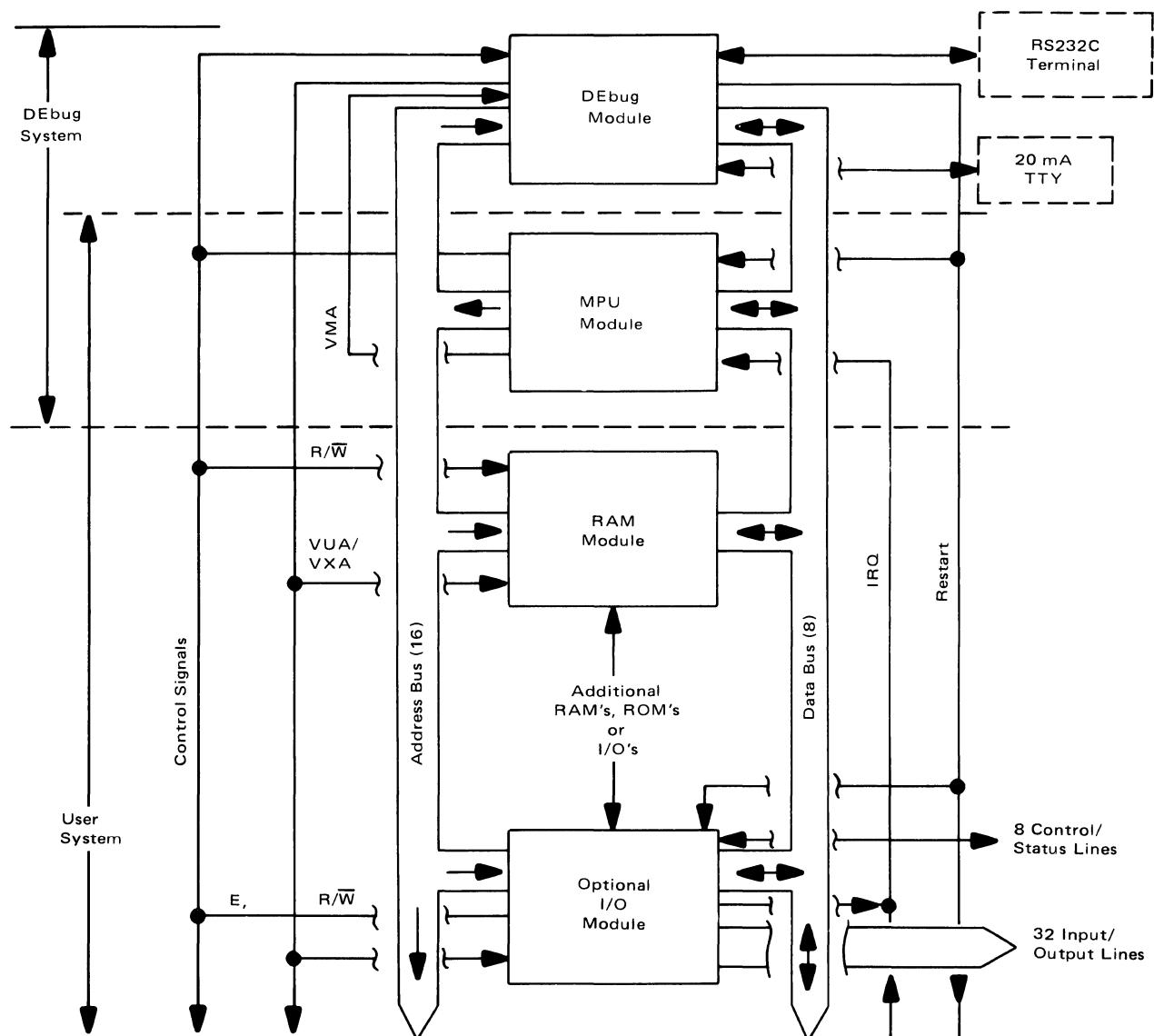


FIGURE 1-1. EXORciser Simplified Block Diagram

### 1.5.2 Dual Map Concepts

The DDebug Module provides the EXORciser with the ability to address two separate 64K memory maps, as illustrated in Figure 1-2. To accomplish this, the DDebug Module takes the Valid Memory Address (VMA) signal from the MPU Module and converts it to two other signals: Valid User Address (VUA) and Valid Executive Address (VXA). All EXORciser hardware modules may be configured to respond to one of these enabling signals. As a result, two complete maps of 64K bytes are addressable for either random access data storage or for data I/O.

EXbug is always assigned to the Executive map, regardless of Single or Dual Map mode. The user selects Dual or Single Map mode via a switch on the DDebug Module. In the Dual Map mode, the generation of the VUA and VXA signals is under EXbug control. In this mode, the user can specify which map he wishes to use via EXbug commands. However, when using the User map, the VUA light on the front panel will not be on until a program is actually being run in the User map. This occurs because EXbug, in the Executive map, is still in control of the system until control is given to a program in the User map.

MDOS must be run in the Executive map when the system is in the Dual Map mode because MDOS must access EXbug and its I/O facilities. This leaves the full 64K User map available without restrictions. The Executive map has full access to the User map. However, the User map has only limited access to the Executive map. Control can be passed from the User map to the Executive map only by use of software interrupts (SWI's) when properly enabled by the user.

The DDebug Module also provides for a Single Map mode of operation, as illustrated in Figure 1-2. Here, all addresses of \$F000 and larger are assigned to VXA (EXbug), and all addresses smaller than \$F000 are assigned to VUA. (This assignment is controlled by hardware - not by EXbug firmware.) In this mode, all hardware modules must be configured for VUA operation. Thus, MDOS is effectively run in VUA when the EXORciser is in the Single Map mode. Modules used in this mode must not use addresses \$F000 or larger. Programs run in this mode can access EXbug, but cannot reside in the EXbug address range. The user must also account for the presence of the MDOS ROM and I/O devices when laying out his development memory map, if MDOS is configured in the system. For more information on MDOS, see EXORDisk User's Guide.

The debug system memory is set up so that the very top of memory is RAM, in contrast to ROM or PROM which would normally be used. This is done so that the interrupt vectors can be changed to suit any user requirements. A PROM address is switched in during restart to get the EXORciser started (in EXbug). The EXbug initialization routine sets the RAM interrupt vector locations to values used by EXbug. IRQ is always masked and is not used by EXbug.

The DDebug Module contains hardware that controls the switching of the maps. This hardware may be accessed only via software running under VXA. This normally means that the map control is handled completely by the EXbug firmware. The user may, however, set up service routines in the Executive Map that can control the map selector. By setting the proper trigger value, then by executing the proper instruction that will cause the map to change on the appropriate subcycle of an instruction execution, the User map dynamically transfers program control from one map to another. The Software Interrupt (SWI) servicing routines in EXbug play a major role in accomplishing this control. Normal system development using the EXORciser does not require explicit map control by the user. EXbug provides complete access and control to the User map for system development requiring an unobstructed 64K bytes of address space (User map).

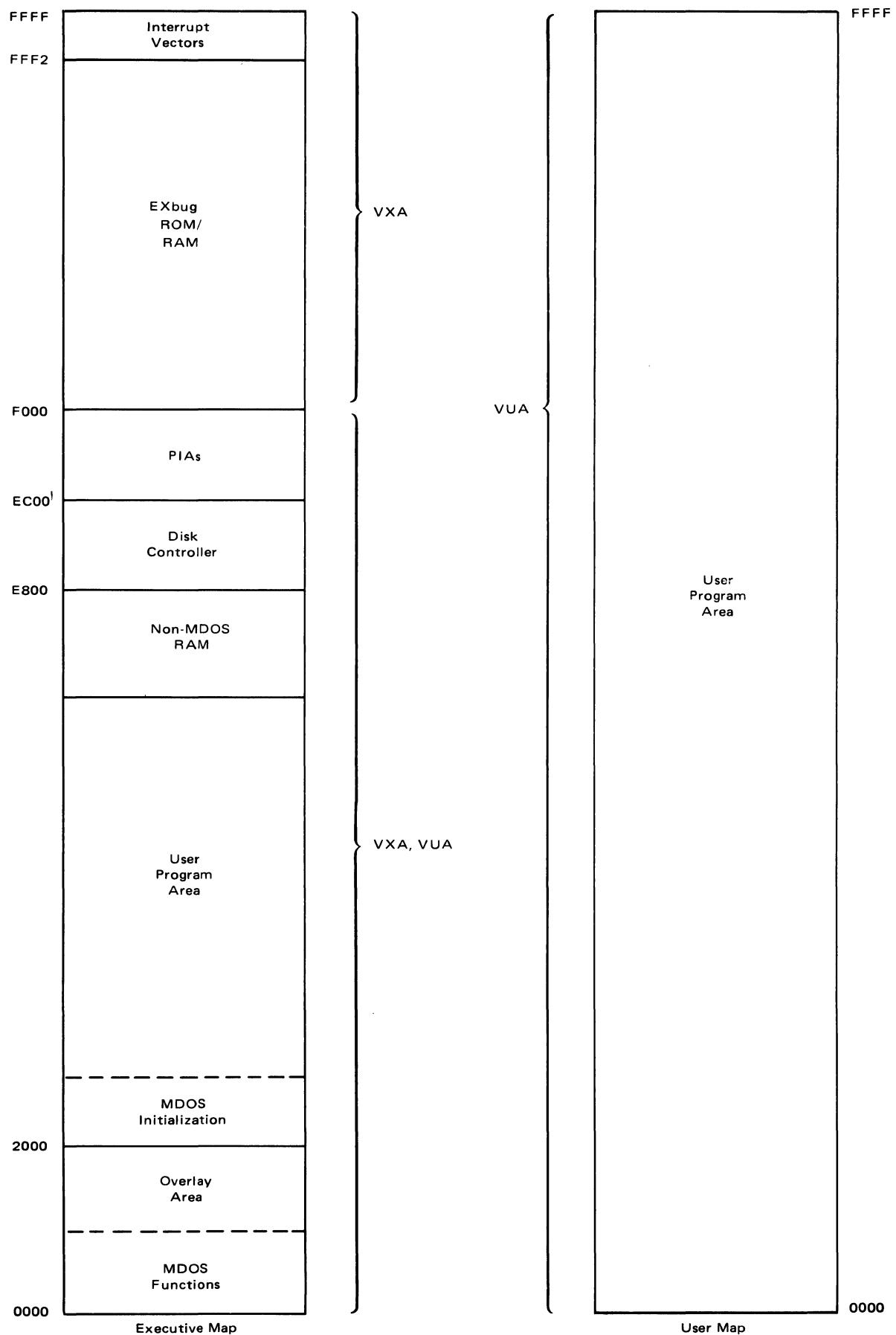


FIGURE 1-2. EXORciser Dual Memory Map

### 1.5.3 Second Level Interrupt Feature

EXbug provides for the SWI and NMI features of the M6809 to be available to the user without restricting EXbug use of them. This capability is referred to as the second level interrupt feature. The EXbug use of the interrupt is the first level and has the highest priority. EXbug gains control first, then decides if user processing at the second level is specified. If the interrupt is not an EXbug interrupt, EXbug passes control on to the second level service routine. Using this technique, breakpoints can be set in a program that uses SWI's for its own use (EXbug uses SWI's to mark breakpoints). The second level service routines are reached via an address vector, just as normal interrupt service routines. However, the locations where the vectors are kept vary, depending on the map in use and whether the system is in the Single or Dual Map mode.

In the User map, the second level vectors are at the same locations as the normal vectors -- i.e., \$FFF2 through \$FFFF. Since EXbug vectors must be at these locations in the Executive map, the second level Executive map vectors are located indirectly; that is, the address contained in locations \$FF00 and \$FF01 in the Executive map points to the last byte of the second level reset vector. In this manual, this location is referred to symbolically as "ATOP". The other vectors are in the normal order, as illustrated in Figure 1-3.

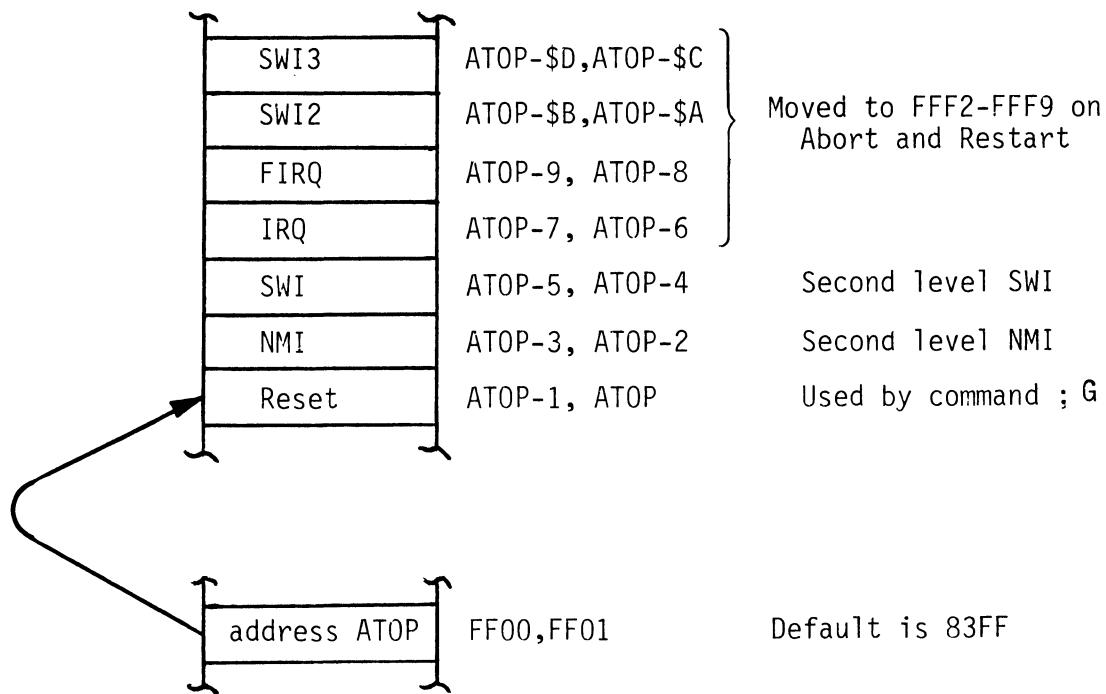
Reset	FFFE, FFFF (used by user restart and ;g command)
NMI	FFFC, FFFD
SWI	FFFA, FFFB (2nd level if breakpoints set)
IRQ	FFF8, FFF9
FIRQ	FFF6, FFF7
SWI2	FFF4, FFF5
SWI3	FFF2, FFF3

FIGURE 1-3. Interrupt Vectors in User Map

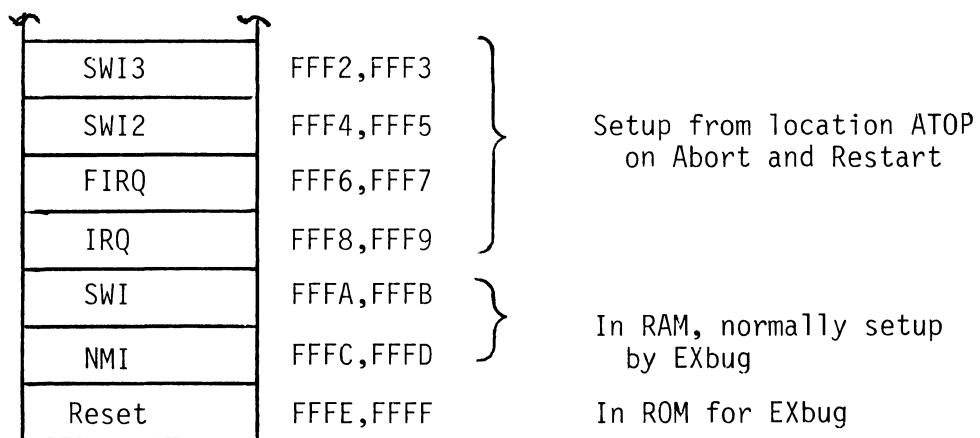
EXbug only gains control of non-EXbug NMI's if the system is in the Single Map mode or if instructions are being executed from the Executive map of the Dual Map mode when the NMI occurs. In both cases, EXbug uses the indirect approach through the address in \$FF00, \$FF01 to determine the address of the second level service routine. Figure 1-4 illustrates this process.

The action taken due to a nonbreakpoint SWI is determined by the value of EXbug E parameter (see the E command description, paragraph 3.6.1). A non-breakpoint SWI can generate an error message, or be serviced by the second level routine. The address of the second level SWI routine in the User map is obtained from \$FFFA, \$FFFB. For the Executive map and the Single Map mode, the address of the second level SWI routine is obtained indirectly through \$FF00, \$FF01. EXbug does not handle IRQ's, so they can be handled directly by the user in whichever map the interrupt is recognized.

The user must put the IRQ, FIRQ, SWI2, and SWI3 vectors at \$FFF2 through \$FFF9 in the User map. In the Executive map and the Single Map mode, the indirect pointer in \$FF00, \$FF01 is used to establish these vectors. During abort and restart, EXbug moves the IRQ, FIRQ, SWI2, and SWI3 vectors specified by the address in \$FF00, \$FF01 to the normal \$FFF2 through \$FFF9 address range. On restart, EXbug puts an address of \$83FF into locations \$FF00, \$FF01 as the default second level vector top of memory in the Executive Map.



a. Executive map (Dual mode) or User map if in Single mode.



b. User map of the Dual Map mode.

FIGURE 1-4. 2nd Level Interrupt Vector Locations for Executive Map and User Map

Tables 1-2 through 1-5 summarize the rules used by EXbug for handling interrupts, including these second level interrupts.

The response time for second level service of interrupts is slower than if the interrupts were serviced directly. This is because EXbug gains control and must determine that the interrupt is not an EXbug function. EXbug must then determine what action to take, and then set up the processor registers and stack accordingly. On entry to a second level service routine, the processor registers and stack are configured as if control had been given directly to the routine without intervention by EXbug.

Response time of second level SWI's serviced in the User map can be speeded up by not setting breakpoints. Executive map response can be speeded up by directly taking over the interrupt vector. This is done by placing the address of the service routines at the normal vector positions, \$FFFA through \$FFFD. However, this eliminates use of EXbug functions which require these vectors: SWI - breakpoints; NMI - abort, trace, program continuation at a breakpoint. Also, the EXbug vectors are restored to the locations on restart.

TABLE 1-2. Second Level Interrupt Rules

- I. Second level interrupts are interrupts which are not related to EXbug functions. They are generated by the user.
- II. Exbug gets control from a second level interrupt but passes it on to the user's service routine.
- III. Possible second level interrupts are NMI and SWI.
- IV. Addresses for second level interrupt service routines in the Executive map are located indirectly through ATOP. Second level interrupt service routine addresses in the User map are in the normal vector locations.

TABLE 1-3. Second Level NMI Rules

- I. Second level NMI is only possible in the Single Map mode or when the system is running in the Executive map of the Dual Map mode.
- II. Second level NMI is always enabled when the system is in either of the two modes described in item I.
- III. The address of the second level NMI service routine for both of the modes in item I is determined indirectly through ATOP.
- IV. User NMI's generated while the system is in the User map of the Dual Map mode will not return the system to the Executive map.

TABLE 1-4. Second Level SWI Rules

- I. Second level SWI's can occur in any map and in any map mode.
- II. Second level SWI's must be enabled by the EXbug E command. User SWI's in the Single Map mode or the Executive map of the Dual Map mode will cause an "SWI" error message if the second level SWI is not enabled. User SWI's in the User map of the Dual Map mode, without the second level enabled, will be serviced directly in the User map if no breakpoints are set, or will generate an "SWI" error message if breakpoints are set.
- III. Second level SWI's from the User map of the Dual Map mode may be serviced in either the User map or the Executive map, depending on the E command value entered.
- IV. The address of the second level SWI service routine for the Executive map is determined indirectly through ATOP. The address of the second level SWI service routine for the User map is in \$FFFA, \$FFFB of the User map.
- V. User map SWI's being handled in the Executive map require the S stack to be in memory common to both maps (i.e., enabled by VMA).

TABLE 1-5. IRQ, FIRQ, SWI2, and SWI3 Rules

- I. EXbug does not use these interrupts and, therefore, does not have service routines for them. They must be serviced directly by the user.
- II. They are serviced in the map in which they are recognized. They do not cause a map change.
- III. The vectors always come from \$FFF2-\$FFF9 in the current map when the interrupt is recognized.
- IV. The vectors for the Single Map mode and the Executive map of the Dual Map mode are initialized by EXbug during abort and restart indirectly from ATOP. On abort and restart, ATOP is initialized to \$83FF.

#### 1.5.4 Dynamic System Bus

In the upper left-hand corner of all EXORciser modules (except the MPU Module) is a 20-pin (4-pin on Static RAM) header known as the Dynamic System Bus (DSB). As implied in its description, this input/output port gives the designer a means for dynamically expanding the hardware capabilities of the EXORciser. The DSB facilitates the implementation of such system features as:

- Priority Interrupts
- Extended Memory Systems
- Advanced Parity Error Control
- Multiprocessor Applications

None of the connections on the DSB is used by EXbug or any of the standard EXORciser modules (although the pins are connected to their appropriate logic on the modules). No cables are provided for the DSB. Instead, the DSB provides a unified approach for system designers to incorporate EXORciser modules into more sophisticated end products of their own.

Some of these possible applications are described below. Additional ideas will be recognized by imaginative designers.

PARITY-ERROR: For memory systems that demand a more sophisticated parity error control than the standard EXORciser parity (see par. 1.5.1), the PAR-ERR signal could be used as an input to a custom module that could perform such functions as retry, restart, or fault address identification. If more than one memory module needed to be controlled, the open header used for the DSB could be used for nonbusied connections. For example, an individual pair of twisted wires with a 2-pin female connector could tie multiple memory modules to a single hardware module designed by the user.

PAGE ENABLE: The Dual Map concept has been extended by making each EXORciser module addressable in one of three modes:

VUA - Valid User Address

VXA - Valid Executive Address

PAGE-ENABLE - for multiple "pages" of 64K bytes

Each module provides a jumper arrangement that allows the user to assign the module to one of these addressing modes. If a user builds a controller that can convert the VMA signal from the MPU into one of several pages, an unlimited number of "pages" of 64K bytes can be realized. These "pages" could contain any combinations of peripherals and memories. Once again, individual twisted pairs would need to be connected from unique modules to a central control module. This addressing capability will be useful in multiple-terminal, multi-disk, and extended memory systems.



## CHAPTER 2

### INSTALLATION INSTRUCTIONS AND HARDWARE PREPARATION

#### 2.1 INTRODUCTION

This chapter provides the unpacking, inspection, installation, and preparation-for-use instructions for the EXORciser Development System. Information is also provided on the data terminal selection and connection to the EXORciser.

#### 2.2 UNPACKING INSTRUCTIONS

##### NOTE

IF THE SHIPPING CARTON IS DAMAGED UPON RECEIPT,  
REQUEST THAT THE CARRIER'S AGENT BE PRESENT  
DURING UNPACKING AND INSPECTION OF THE SYSTEM.

Unpack the EXORciser from its shipping carton. Refer to the packing list and verify that all of the items are present. Save the packing materials for storing or reshipping the system.

#### 2.3 INSPECTION

The EXORciser should be inspected upon receipt for broken, damaged, or missing parts, and for any physical damage to the chassis and/or internal modules.

#### 2.4 INSTALLATION INSTRUCTIONS

As delivered, the EXORciser can be mounted on a table top, bench, or any other flat surface having sufficient room to allow easy access to the front and rear panels. After a location has been selected, proceed with the following steps.

- a. Connect the selected data terminal to the EXORciser (refer to par. 2.5 for data terminal selection and connection information).
- b. With the EXORciser POWER switch positioned OFF, connect the system to the selected power source.

##### CAUTION

INSERTING MODULE WHILE POWER IS APPLIED MAY  
RESULT IN DAMAGE TO COMPONENTS ON THE MODULE.

- c. Ensure that all of the necessary modules are installed prior to application of power. (NOTE: Since each module is offset, as well as keyed, there is no chance of installing the modules backward. However, wire-wrapped modules that require more than normal card separation should be installed into the slots provided for this purpose.)
- d. Connect interface cables to controller modules (if installed). If the EXORciser top is to be installed, all cables must exit the chassis from the rear.
- e. Depress the POWER switch and observe that the indicator lamp on the switch illuminates.

## 2.5 DATA TERMINAL SELECTION AND CONNECTIONS

The type of data terminal chosen for use with the EXORciser depends on the other peripherals that are used, and on the total role of the Development System. The EXORciser is compatible with a wide range of terminals because of its capability to communicate via either the RS-232C or 20 mA current loop interface. The terminal interface should be full duplex and transfer data at rates between 110 and 9600 baud. The standard ASCII communications protocol has been implemented and is further described in Appendix C.

### 2.5.1 RS-232C Interconnections

The RS-232C interface is commonly used with terminals and modems. Most terminals come equipped with a cable which will plug into the EXORciser and operate correctly. However, in some cases, it is necessary to assemble or modify a cable to make certain the proper signals are supplied to the terminal. The EXORciser only needs to be connected to the data send and receive lines (and ground), but most terminals are designed to work with modems and require the RS-232C handshake signals to work properly. For this reason, logic is included to sense the Data Terminal Ready (DTR) signal and to turn it around to supply signals on the Clear-to-Send (CTS), Data Set Ready (DSR), and Data Carrier Detected (DCD) lines. These signals are required by some terminals before they will operate. It is not advisable to connect more than one RS-232C device to the EXORciser (in parallel), since this is not permitted by the standard, and voltages may be out of limits. (See Appendix E for RS-232C signal descriptions and pin assignments.) Appendix F describes the use of the TI ASR733 terminal.

### 2.5.2 20 mA Current Loop Interconnections

Although originated for teletypewriters, the 20 mA current loop interface is used by many other terminals because of its simplicity. When the 20 mA interface is used for terminals other than a TTY, a baud rate faster than 110 baud can be used, but not faster than 1200 baud because of the bypassing used for noise reduction.

While the EXORciser directly outputs only RS-232C interface signals, these signals can be easily converted to the 20 mA neutral current loop protocol through the use of Micromodule 11 (M68MM11). The interconnections required for this current loop interface, along with the necessary connections at the ASR33 teletype, are described in detail in Appendix D and the Micromodule 11 User's Guide.

## 2.6 PREPARATION OF SYSTEM MODULES

The MPU and DEbug Modules represent the minimum configuration for an EXORciser system. The Floppy Disk Controller Module must be added in order to use the EXORDisk and MDOS. Various memory modules may be configured, as required. The minimum module configuration to operate an MDOS-based EXORciser consists of the MPU, DEbug, and Floppy Disk Controller Modules, plus at least 16K of memory. (The Macro Assembler requires at least 24K of memory.) Additional I/O modules may be required for specific system development.

The reader should refer to the applicable User's Guide for complete details on each module.

## CHAPTER 3

### OPERATING INSTRUCTIONS

#### 3.1 INTRODUCTION

Information in this chapter is intended to familiarize the user with the basic operating procedure needed to initialize the EXORciser, use the dual memory map, and select the addresses of the various modules. This chapter also provides a description of the EXbug commands, subroutines, and entry points that are used to perform system evaluation and debug procedures.

#### 3.2 SWITCHES AND INDICATORS

The EXORciser switches and controls are divided into three categories: (1) those on the EXORciser chassis, (2) those on the DDebug Module, and (3) those on the MPU Module and the optional modules. The controls available on the MPU and optional modules (memories and peripherals) are described in their respective supplements.

##### 3.2.1 Front Panel Switches and Indicators

Figure 3-1 illustrates the arrangement of the front panel switches and indicators.

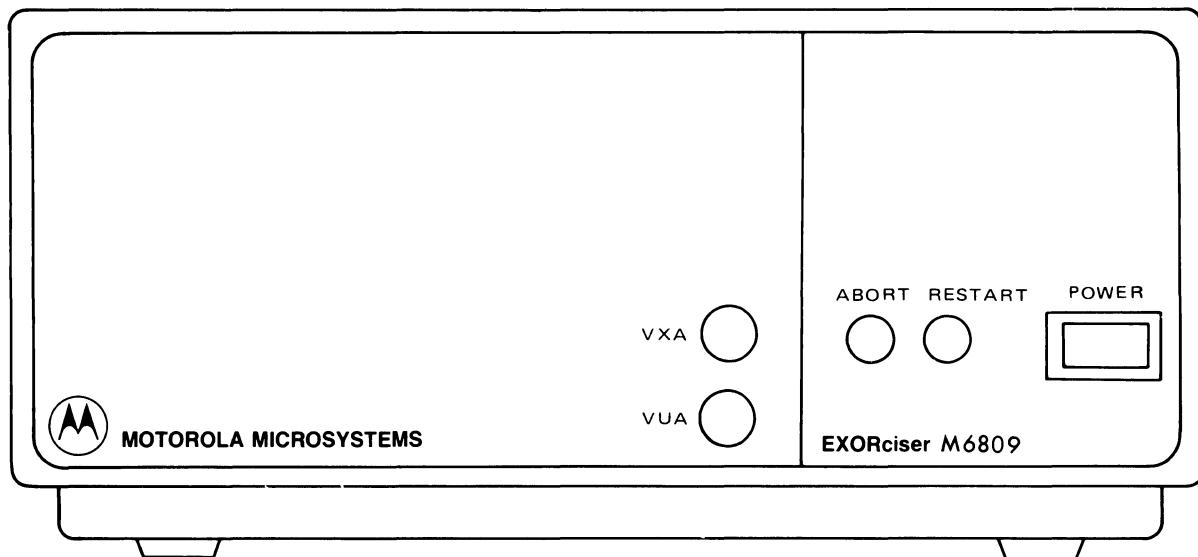


FIGURE 3-1. EXORciser Front Panel Switches and Indicators

**POWER** - The power switch is used to turn the EXORciser ON and OFF. It is an alternate action pushbutton switch with a built-in indicator which is illuminated when A/C power is ON.

**ABORT** - The momentary pushbutton switch labeled ABORT causes an NMI (Non-Maskable Interrupt) to be generated. Program control is returned to the address indicated in the Executive map locations FFFC (MSB) and FFFD (LSB) of the NMI vector. Except when a program is running in the User map, an abort operation always forces the system into the Executive map, even though the Dual Map mode is selected. If locations FFFC/FFFD of the Executive map have not been explicitly changed by the user, pressing the ABORT button will cause control to be returned to the EXbug routine.

**RESTART** - This momentary pushbutton switch generates a low level RESTART signal throughout the system. As in the case of power ON, control is determined by the address contained in the Restart vector locations FFFE (MSB) and FFFF (LSB). A switch on the DEbug Module indicates in which map (User or Executive) the Restart vector is to be found. The RESTART signal is also supplied to the bus to reset all hardware that recognizes the Restart. If EXbug is in the map indicated by the RESTART switch on the DEbug Module, pressing the front panel RESTART button causes the EXORciser to initialize itself through the EXbug initialization firmware.

**VUA/VXA** - The lights labeled VUA and VXA indicate when and in which map the EXORciser is executing. Both lamps will be OFF when any of the three bus signals -- BA (Bus Available), HALT, or BUSREQ (Bus Request) --is active. For example, if the processor is in a CWAI (Clear and Wait for Interrupt) state, neither light will be on. When the processor is executing, the lights will indicate which map is currently active. The VUA indicator (Valid User Address) is on when the system is executing in the User map. This is true when Dual Map mode is selected, the user has entered "USER", and the EXbug firmware is not being accessed. It is also true when in Single Map mode and the EXbug firmware is not being accessed. The VXA light is on when EXbug firmware is being executed under either single or dual map selection, or when execution is from the Executive map under Dual map configuration.

### 3.2.2 Switches on the DEbug Module

The three switches that will be used during normal EXORciser use are described here. They are accessible when the chassis cover is removed. Figure 3-2 summarizes their orientation, when viewed from the component side of the DEbug Module.

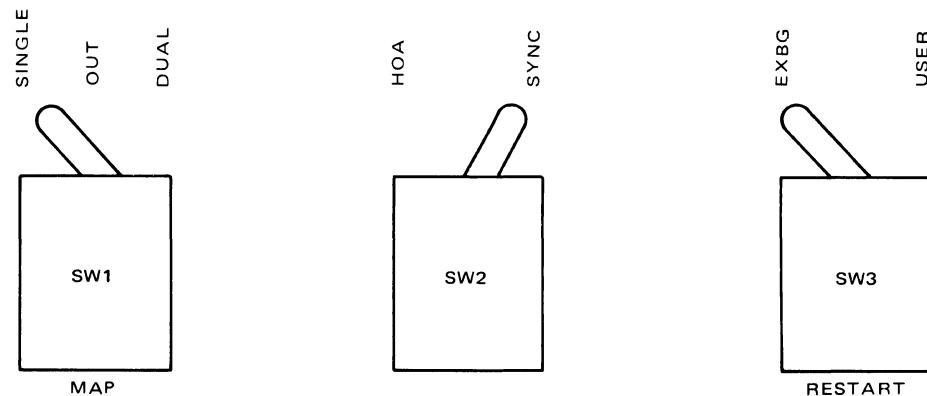


FIGURE 3-2. Toggle Switches on DEbug Module

**SW1 (MAP MODE)** - This is a three-position switch which selects which map the EXbug firmware is to reside in. In the SINGLE position, only 64K of memory addressing space is available. EXbug occupies the high order portion of this map. In the OUT position, the EXbug firmware is essentially disabled and the DEbug Module becomes inactive. In the DUAL position, EXbug resides in the executive portion of memory. The dual 64K memory map capability is enabled by this switch position. When in this position, the user controls which map is accessed via commands entered into EXbug (see USER and EXEC commands).

**SW2 (HALT-ON-ADDRESS)** - This switch is used in conjunction with the EXbug command H (Halt-on-Address). When the user has enabled the Halt-on-Address feature by entering the H command, this switch selects either the Trigger (SYNC) mode or the Halt-on-Address (HOA) mode. When set in its HALT-ON-ADDRESS position (toggle left), the EXORciser hardware will stop program execution if the address bus compares with the address entered by the user for the H command. When execution is halted, the MPU registers are displayed and control is returned to the EXbug firmware. When the switch is set to the right, a scope trigger pulse is generated when the program accesses the specified address. This trigger is physically available to the user at the sync test point in the upper left of the DEbug Module. During Trigger mode, program execution continues at full speed.

**SW3 (RESTART MAP)** - This switch allows the user to specify which map the EXORciser is to access during power-up or RESTART. EXbug Restart vectors will always be used when in Single Map mode. If in Dual Map mode, and the Restart switch is set to its USER position, the EXORciser will use the User map for its restart vector and subsequent execution. This feature allows the user to fully evaluate the power startup portion of his final system design.

### 3.3 INITIALIZATION

System initialization consists of selecting the system baud rate, module addresses, turning power on, and performing any required start-up procedures. Before the power is applied, it is necessary to select the baud rate to match the terminal, and set the switches on the DEbug Module for the desired configuration. Normally, the baud rate is selected when the system is first configured, and does not require further adjustments unless the system terminal is changed. Memory, I/O, or Disk Controller Module must be properly jumpered for VUA or VXA to match the mode selected, as well as being set for the proper addresses. Module addresses are selected as required for software development or for the target system being emulated (see paragraph 3.5).

#### 3.3.1 Power ON/OFF Procedures

The recommended sequence for turning on the equipment in an EXORciser system is:

- a. System Console (Terminal)
- b. EXORciser
- c. EXORDisk Unit
- d. Printer Unit

#### CAUTION

DISKETTES SHOULD NOT BE MOUNTED  
IN EXORDISK WHEN APPLYING POWER.

The recommended sequence for turning power OFF is:

- a. Printer Unit
- b. EXORDisk Unit
- c. EXORciser
- d. System Console (Terminal)

CAUTION

DISKETTES SHOULD BE REMOVED FROM  
EXORDISK BEFORE TURNING POWER OFF.

### 3.3.2 Baud Rate Selection

Typically, the EXORciser will be operated at the maximum baud rate permitted by the system console. Refer to DDebug User's Guide for instructions on configuring the DDebug Module for the selected baud rate.

### 3.3.3 Start-Up Procedures

The procedures that are required at power-up and following a restart are described below. Here again, the required procedures depend on the system configuration.

If the system console requires null padding after a carriage return (see paragraph 3.6.2.5 and Appendix F), the "K" command should be used to specify the padding. This ensures that the first few characters on new lines are accurately displayed on various terminals.

If the system is being operated in Dual Map mode, the MAP switch on the DDebug Module must be set to DUAL, and the memory or I/O modules must be properly jumpered for VUA or VXA. If the user wishes to debug programs in the User map, the EXbug command USER must be entered.

If parity error detection is desired, the user must enable the detection logic after the memory has been initialized. The EXbug semicolon command is used to enable parity. If parity is to be monitored in the User map memory (RAM only), the memory initialization (command I) should be entered after the USER command is specified.

In the Single Map mode or Executive map of the Dual Map mode, EXbug assumes that the user program may be contained within half of the maximum range of the EXORciser memory. For this reason, EXbug provides for moving the user IRQ, FIRQ, SWI2, and SWI3 vectors up to the EXbug memory at \$FFF2 through \$FFF9. Two locations in the EXbug memory are initialized to \$83FF on the assumption that that is the top of user memory. The IRQ vector would be at \$83F8. The vector is moved up to the EXbug memory at \$FFF8 and \$FFF9 simply by pressing the ABORT button. This causes the user normal IRQ, FIRQ, SWI2, and SWI3 vectors to be moved to the appropriate area. In the event a different top of memory address is desired, it is only necessary to change locations \$FF00 and \$FF01 to the correct value by means of the EXbug Memory Change feature (described in par. 3.6.3) and press the ABORT button. Remember that pressing the RESTART button, or turning the EXORciser power OFF and ON, will restore \$83FF, and it will be necessary to re-enter the revised address. EXbug uses the top of memory address to find the user restart vector and second level SWI and NMI vectors.

NOTE

If the IRQ, FIRQ, SWI2, and SWI3 vectors are to be used, the ABORT button should always be pressed after loading a program into the Executive map or the Single Map mode, even though the top of memory address is \$83FF.

### 3.4 USING THE DUAL MEMORY MAP

The DEbug Module provides the EXORciser with the capability of addressing two separate 64K blocks of memory. These two blocks of memory are referred to as the Dual Memory map. One of these, the Executive map, contains EXbug and, if configured for Dual map, its peripheral devices and RAM, the EXORDisk ROM and I/O devices, and the Printer I/O device. The other, the User map, is completely available to the user for emulation of his target system. This gives the user complete freedom in assigning addresses to his memory and I/O devices without worrying about addressing conflicts with the system monitor and I/O devices, yet EXbug still provides the user with full debug capabilities in the User map. Optionally, in the Single Map mode, the DEbug Module can merge the two maps. In this mode, all addresses less than \$F000 come from the User map. The DEbug Module preparation section describes how to select the Single or Dual Map mode.

In the Dual Map mode, all of the EXbug debug commands are available in either map. The EXbug USER and EXEC commands control which map will be accessed by the debug commands. The command USER causes the EXbug debug commands to operate in the User map. In this mode, EXbug's prompt is \*. The command EXEC causes the EXbug debug commands to operate in the Executive map. EXbug's prompt is \*E in this mode. On power-up, EXbug comes up in the EXEC mode.

In the Single Map mode, the EXEC and USER commands are not usually required since the maps have been merged. However, if the Halt-on-Address or Scope Sync functions are to be used at an address less than \$F000, the USER command must be entered so that the address compare circuitry will detect the appropriate map.

### 3.5 ADDRESS SELECTION

The address selection of the various modules will depend on whether the system is being used for software development or to emulate the target system. Included in the address selection is the determination of which map the module will respond in. The user assigns a module to one of the two maps by installing either the VUA or VXA addressing jumper that is found on all EXORciser modules.

During software development (edits, compilations, assemblies, etc.), the user will probably want as much contiguous RAM as possible, starting at address 0000 (minimum of 16K). In the Dual Map mode, programs in the User map cannot reference EXbug or the system terminal without special techniques (see Appendix G). Therefore, MDOS memory or disk controllers must be in the Executive map with EXbug. In the Single Map mode, where the distinction between Executive and User maps is simply the address boundary F000, EXbug can be accessed from the User map. This requires that the EXORDisk Interface, Printer Interface (if they are used), and RAM be configured for the VXA, and the DEbug Module be configured for the Dual Map mode, or that the RAM, EXORDisk Interface, and Printer Interface be configured for VUA, and the DEbug Module be configured for the Single Map mode. These same requirements also apply for any programs that use the system terminal or EXbug routines.

During target system emulation, the module addresses will be selected as required for the target system. The target system may be emulated in the User map of the Dual Map mode or, if it does not require any addresses equal to or greater than F000, it may be emulated in the Single Map mode. In the Single Map mode, all modules should be configured to respond to VUA.

The various module User's Guides contain instructions on setting the address and map of the various modules. Modules in the Executive map should be addressed only at values less than F000 to avoid conflicts with EXbug.

### 3.6 EXbug COMMANDS

There are three groups of EXbug commands:

- a. Four-character commands followed by a carriage return.
- b. Single-character commands following a semicolon, period, or dollar sign.
- c. Memory change commands.

Four-character commands specify the map to be used, control the operation of the console tape (cassette or paper tape), load the disk operating system, and print memory. The user may add four-character commands to EXbug. Single-character commands control program debug functions. Memory change commands allow memory locations to be examined and changed. Any command may be entered when EXbug is displaying its prompt. A list of the EXbug commands is in Table 3-1.

TABLE 3-1. EXbug Commands

COMMAND	EXPLANATION	Page Ref.
EXEC return	Debug in the Executive map (default).	3-8
USER return	Debug in the User map.	3-9
PRNT return	Print memory in both hexadecimal and ASCII format.	3-9
LOAD return	Load an object tape from the terminal to memory.	3-11
VERF return	Verify an object tape from the terminal against memory.	3-12
SRCH return	Search an object tape on the terminal.	3-13
PNCH return	Punch an object tape on the terminal from memory.	3-10
MDOS return	Set the EXEC mode, then jump to E800.	3-9
.A nn [byte] return	Display and change the A accumulator.	3-15
.B nn [byte] return	Display and change the B accumulator.	3-15
.C nn [byte] return	Display and change the condition code register.	3-15
.D nn [byte] return	Display and change the DPR register.	3-16
;E nn [byte] return	Display and change the second level SWI enable.	3-27
;G	Go (jump) to the target program at its restart address.	3-19
addr;G	Go (jump) to the target program at the specified address.	3-19
\$H nnnn [addr] return	Enable and change the halt on address or scope sync.	3-18
;H	Disable the halt on address and scope sync.	3-18
byte; I	Initialize memory with a specified byte.	3-21

TABLE 3-1. EXbug Commands (cont'd)

COMMAND	EXPLANATION	Page Ref.
;K nnnn [value] return addr;L	Display and change the terminal null pad value. Calculate long relative offset from currently open location to the specified location.	3-22 3-28
\$M or ;M	Display and change the memory search beginning and ending addresses and search mask.	3-23
;N value;N addr;0	Trace the next instruction. Trace the next specified number of instructions. Calculate short relative offset from currently open location.	3-20 3-20 3-28
.P nnnn [addr] return ;P value;P	Display and change the program counter. Proceed with program execution. Proceed with program execution from breakpoint; value specifies the number of times the breakpoint location is to be passed before returning control to EXbug and providing a register printout.	3-15 3-20 3-20
;Q nnnn [value] return \$R or ;R	Display and change the default debug offset. Display the target program registers.	3-26 3-14
.S nnnn [addr] return	Display and change the stack pointer.	3-16
\$T nnnn [addr] return ;T	Enable and change the trace to ending address. Disable the trace to ending address.	3-18 3-19
.U nnnn [addr] return ;U addr;U	Display and change the U register. Remove all breakpoints.	3-16 3-17
\$V or ;V addr;V byte;W	Remove a specified breakpoint. Display the breakpoint addresses. Set a breakpoint at the specified address. Search memory for the specified byte (word). See the M command.	3-17 3-17 3-17 3-16 3-26
.X nnnn [addr] return .Y nnnn [addr] return	Display and change the X index register. Display and change the Y index register.	3-15 3-15
;Z nn [byte] return ;: ;;	Copy terminal output to printer option. Display the memory parity error interrupt. Enable the memory parity error interrupt.	3-23 3-21 3-21
Control-X	Abort the current command or entry.	3-9
Control-W	Wait until some other character is entered.	3-9

TABLE 3-1. EXbug Commands (cont'd)

COMMAND	EXPLANATION	Page Ref.
addr/nn cmnd	The memory change function is invoked by entering addr/. Cmnd is one of the following memory change function commands. These commands are accepted as long as EXbug remains in the memory change function.	3-27
[byte] LF	Change memory if byte entered, and display the next sequential location.	3-27
[byte] space	Change memory if byte entered, and display the previous sequential location.	3-27
[byte]/	Change memory if byte entered, and redisplay the current location.	3-27
[byte] return	Change memory if byte entered, and exit the memory change function.	3-27

- NOTE:
- a. Hexadecimal numbers may be preceded by a minus sign to obtain the two's complement of the value entered.
  - b. Values shown in brackets ([ ]) in above explanations indicate optional user inputs.
  - c. When addr is a single number (e.g., 142), the debug offset is added to the number to determine the value used by the command. If two comma-separated values are entered, the sum of the values is used by the command.

Some features are common to most commands. All parameters entered are assumed to be hexadecimal values. A minus sign preceding a value will cause the two's complement of the value to be used. Control-X can be used to delete the current entry or command and cause EXbug to print another prompt. All address and 16-bit register parameters will automatically be adjusted by the addition of a debug offset (see the Q command) unless a second parameter, separated from the first by a comma, is entered. On power-up and restart, the debug offset is reset to a value of 0. When a second parameter is entered, it (instead of the debug offset) is added to the first parameter.

### 3.6.1 FOUR-CHARACTER COMMANDS

The four-character commands are activated by entering the appropriate four characters, followed by a carriage return. If more than four characters are entered before the carriage return, only the first four will be used to determine the command. If an invalid command is entered, EXbug responds by typing a question mark, ringing the bell in the system terminal, and then issuing another prompt. The four-character commands are described as follows.

EXEC - When the system is in the Dual Map mode, this command causes all debug commands entered after it to operate on the Executive map. This mode remains in effect until the USER command is entered. This is the default mode following power-up or restart. The prompt in this mode is \*E. In the Single Map mode, this command does not affect what memory is accessed.

USER - When the system is in the Dual Map mode, this command causes all debug commands entered after it to operate on the User map. This mode remains in effect until the EXEC command is entered. The prompt in this mode is \*. In the Single Map mode, this command does not affect what memory is accessed. However, it should be entered when a Halt-on-Address or Scope Trigger is active in the Single Map mode.

MDOS - This command causes the EXEC command to be executed and then jumps to the disk boot loader routine at \$E800.

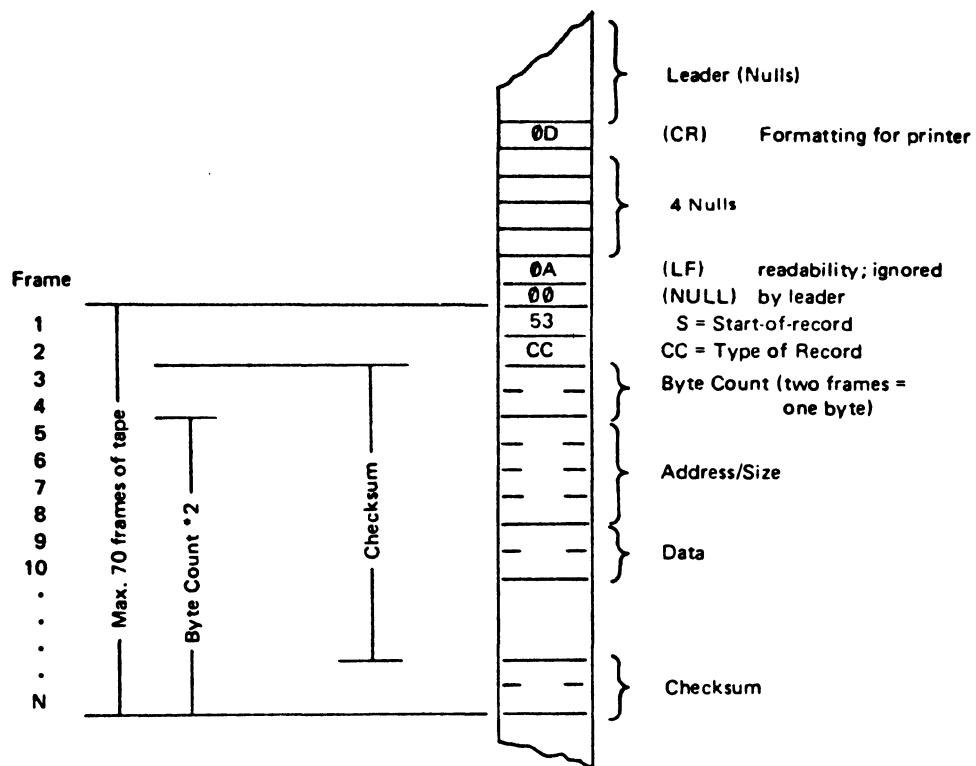
PRNT - This command prints the specified portion of the current memory map in both hexadecimal and ASCII forms. After the user has entered the carriage return, EXbug responds by printing BEG nnnn (where nnnn is the last beginning address entered). If a beginning address had not been entered before, nnnn is whatever value was in the beginning address memory location when the system was turned on. If the beginning address is correct, the user should enter a carriage return. To change the beginning address, the user may enter an address followed by a carriage return, or an address followed by a comma, followed by a second address followed by a carriage return. (The command can also be aborted at this point by entering a Control-X.) If a single address is entered, the beginning address is determined by adding the debug offset to it; if two comma-separated addresses are entered, the beginning address becomes the sum of two addresses entered. If an invalid character is entered in one of the addresses, the command is aborted. If an incorrect address is entered, the correct address may be entered on the same line before the comma or carriage return is entered. Up to 19 hexadecimal characters may be entered before the comma or carriage return. Only the last four characters will be used as the address. If less than four hexadecimal characters have been entered, the unspecified most significant bits are assumed to be zero. For example, entering E CR gives an address of \$000E if the debug offset is 0.

After the beginning address has been successfully entered, EXbug prints END nnnn (where nnnn is the current ending address). Here, the user has the same options for entering an ending address as described for the beginning address. If the ending address to be used is less than the beginning address, EXbug will request the beginning and ending addresses again. If the ending address is equal to or greater than the beginning address, EXbug will print the requested portion of memory. An example of the PRNT command is shown in Figure 3-3. While memory is being displayed, entering Control-W will cause the display to wait at the end of the current line until some other character is entered. Entering control-X will abort the PRNT command at the end of the current line.

```
♦E PRNT
BEG F000
END F050
F000 7E F2 AE 7E F0 45 7E F0 6F 7E F0 D5 7E F0 D9 7E  ^R.^PE^PO^PU^PY^
F010 F0 88 7E F1 18 7E F2 2A 7E F0 B3 7E F0 AF 7E F0  F.^Q.^R^P3^P/^P
F020 AD 7E F0 3A 7E F0 2D 7E F0 2F 7E F0 B1 8D F2 A6  -^P:^P-^P/^P1.R&
F030 00 81 04 27 37 8D E1 08 20 F5 86 0D 8D DA 86 0A  ...^T.A. U...Z..
F040 8D D6 4F 20 D3 CE FB 99 8D E3 CE FF 0A BD F6 04  .VO SNK..cN..=v.
F050 25 F3 CE FB 9E 8D D6 CE FF 0C BD F6 04 25 F3 CE %sNC..VN..=v.%sN
♦E
```

FIGURE 3-3. PRNT Example

PNCH - This command punches the specified portion of the current memory map on the console punch device in an ASCII hexadecimal format. The tape format is described in Figure 3-4.



Frames 3 through N are hexadecimal digits represented by a 7-bit ASCII character.  
Two hexadecimal digits are combined to make one 8-bit byte.

The checksum is the one's complement of the summation of 8-bit bytes.

Frame	CC = 30 Header Record	CC = 31 Data Record	CC = 39 End-of-File Record
1. Start-of-Record	53	S	
2. Type of Record	30	0	
3. Byte Count	31	12	
4.	32		
5.	30		
6. Address/Size	30	0000	
7.	30		
8.	30		
9.	34		
10. Data	38	48-H	
.	34		
.	34	44-D	
.	35	52-R	
.	32		
N. Checksum	39 45	38 41	8A (Checksum)
			(Checksum)

FIGURE 3-4. Tape Format

After the command has been entered, EXbug requests the beginning and ending addresses as described under the PRNT command. After the beginning and ending addresses have been entered, EXbug requests the information to be put in the header record. Zero through 17 characters, terminated by a carriage return, may be entered. If 18 characters, not including a carriage return, are entered, the characters will be ignored and the header information will be requested again. ASCII control characters should not be entered in the header.

When the carriage return terminating the header information is entered, EXbug will begin the punch sequence. Therefore, if the console punch device does not have automatic control using the ASCII DC2 and DC4 characters, it should be turned on after the last header character has been entered but before the carriage return terminating the header is entered. The non-automatic punch should be turned off after EXbug has completed the punch operation and has printed a prompt. If the console punch device does have automatic control, EXbug will control the punch, and no operator intervention is required.

If the system terminal is a TI Silent 700 with the Remote Device Control option, and EXbug has been informed of this in the K command, then the data sent to the console punch will not be printed on the terminal. On other terminals, the data will be printed as it is being punched. If the line printer option is set at zero (see the Z command), the punch data will not be listed on the line printer as it is being punched. Figure 3-5 shows an example of the PNCH command.

```
♦E PNCH
BEG F000
END F050
HDR=XBG2A
$00800005842473241A3
$11BF0007EF2AE7EF0457EF06F7EF0D57EF0D97EF0887EF1187EF22AA5
$11BF0187EF0B37EF0AF7EF0AD7EF03A7EF02D7EF02F7EF0B18DF2A65F
$11BF03000810427378DE10820F5860D8DDA860H8ID164F20D3CEFB99C5
$10CF04880E3CEFF0ABDF6042598
$9030000FC

♦E
```

FIGURE 3-5. PNCH Example

LOAD - The LOAD command reads an ASCII hexadecimal tape from the system console reader into the current memory map. The required tape format is specified in Figure 3-4. The system console reader should respond to either the ASCII device control codes (DC1 through DC4) or to the TI Silent 700 remote device control codes so that EXbug can control tape motion.

After the user has entered the command, EXbug responds with "S/C". Entering Control-X will abort the Load function. Entering S will cause EXbug to load a single object file -- that is, EXbug will stop the LOAD command and issue a prompt when it reads an end-of-file (S9) record. Entering C will cause EXbug to load continuously. All end-of-file records will be ignored. To stop the continuous load, the ABORT button or the RESTART button must be pressed. Entering any other character will cause the message to be repeated. During either the single or continuous load, EXbug will print at the system terminal the data in each header (S0) record it reads.

If a checksum error is encountered while loading, EXbug will print CKSM nnnn (where nnnn is the starting load address of the record in error). Entering Control-X will abort the LOAD command. Entering C will cause the LOAD command to continue and load the record in error. If the checksum error is due to an error in the record load address, there is no way to determine where data will be loaded in memory. The user can reposition the tape to the beginning of the record in error, enter R, and EXbug will re-read the record. Entering any other character will cause the message to be repeated.

As the LOAD command writes each byte into memory, it reads it back to verify that memory changed. If memory does not change properly, the error message

```
ADDR/MM/TP  
nnnn mm tt
```

is printed, and the LOAD command is aborted. nnnn is the address of the memory location that did not change correctly. mm is the hexadecimal value that the memory location changed to. tt is the hexadecimal value for that location from the tape.

Figure 3-6 shows an example of the LOAD command.

```
*E LOAD  
S/C S  
XBG2A  
*E
```

FIGURE 3-6. LOAD Example

VERF - The VERF command verifies the current memory map with an ASCII hexadecimal tape in the system console reader. The required tape format is specified in Figure 3-4. The system console reader should respond to either the ASCII device control codes (DC1 through DC4) or to the TI Silent 700 remote device control codes so that EXbug can control the tape motion.

After the user has entered the command, EXbug responds with "S/C". The user should now enter S, C, or Control-X, as described for the LOAD command, to obtain the desired operation.

The VERF command checks for header (S0) records and checksum errors, the same as the LOAD command. The data in each header record read will be printed at the system terminal. Checksum errors during VERF result in the same error message and permit the same error response options as checksum errors during LOAD. If the checksum error is due to an address error, and the continue option is selected, the verification will be meaningless.

When a mismatch between the tape and memory is detected, the error message

```
ADDR/MM/TP  
nnnn mm tt
```

is printed. nnnn is the address of the error. mm is the memory contents. tt is the tape contents. The heading ADDR/MM/TP is printed only for the first error detected;. Only the address and data portions of the message are printed for any subsequent errors. While verify errors are being printed, entering Control-W will cause the printout to stop until another character is entered. Also, while verify errors are being printed, entering Control-X will cause the VERF command to be aborted.

An example of the VERF command is shown in Figure 3-7.

```
*E VERF  
S/C S  
XBG2A  
*E
```

FIGURE 3-7. VERF Example

SRCH - The SRCH command searches the tape in the system console reader for header (S0) records. All other record types will be skipped over. Figure 3-4 shows the required tape format. The system console reader should respond to either the ASCII device control codes (DC1 through DC4) or to the TI Silent 700 remote device control codes so that EXbug can control the tape motion.

On reading a header record, SRCH will stop the tape and print the data in the record. SRCH will then print "C/L/V". Entering C will cause the search to continue. Entering L will cause the LOAD command to be entered. Entering V will cause the VERF command to be entered. Entering Control-X will cause the SRCH command to be aborted and an EXbug prompt to be issued. Entering any other character will cause the message to be repeated. Refer to the LOAD and VERF command descriptions for the messages printed by these commands and the appropriate responses.

If SRCH detects a checksum error in the header record, it will print the checksum error message described in the LOAD command. An example of the SRCH command is shown in Figure 3-8.

```
*E SRCH  
XBG2A  
C/L/V L  
S/C s
```

FIGURE 3-8. SRCH Example

### 3.6.2 Single Character Commands

The single character commands control debug functions. These commands are preceded by a semicolon, period, or dollar sign. Single character commands fall into various groups: register display and change, program execution control, memory parity control, I/O control, memory search, and miscellaneous functions.

The following conventions are applied in the command format representations:

n a hexadecimal digit of 0 through F displayed by the command. Thus, nnnn is a 4-digit hexadecimal number, and nn is a 2-digit hexadecimal number.

[ ] items contained within brackets indicate an optional parameter.

addr a 16-bit value entered by the user and expressed in hexadecimal. A single hexadecimal number may be entered, or two hexadecimal numbers separated by a comma may be entered. For example, 12AB and 56DC,123 are valid entries. Leading zeros may be omitted. If a single number is entered, the debug offset is added to the number to determine the value used by the command (see Q command). If two comma-separated values are entered, the sum of the values is used by the command.

byte an 8-bit value entered by the user and expressed in hexadecimal. Leading zeros may be omitted.

value a 16-bit value entered by the user and expressed in hexadecimal. This value is not relocated by the debug offset. Leading zeros may be omitted.

return a carriage return.

If an error is made in entering a value, the value may be re-entered so that the last four digits entered is the desired value. Up to 19 digits may be entered for a single value. However, only the last four digits entered will be used. A command may be aborted while a value is being entered by entering Control-X. If an invalid character is entered in a value, the item being displayed will not be changed when the command is terminated. If an error is made in a command, EXbug will print a question mark and ring the bell in the system terminal.

**3.6.2.1 Register Display and Change.** These commands allow the user to display and change the M6809 register values that will be used while executing the program under test. There is one command that displays all the register values, while individual commands are used to display and change each register.

Function: Display all registers

Format: ;R or \$R

Description: This command displays the target register values in the following format:

P-nnnn X-nnnn Y-nnnn A-nn B-nn C-nn DP-nn U-nnnn S-nnnn

where n is a hexadecimal digit. P, X, Y, A, B, C, DP, U, and S designate the program counter, X index register, Y index register, A accumulator, B accumulator, condition code register, direct page register, user stack pointer, and stack pointer, respectively.

- Function: Display and change the program (location) counter  
Format: .P nnnn [addr] return
- Description: This command displays the target program counter value (nnnn). To change the program counter value, enter a new value as described in the definition of addr. A carriage return terminates the command.
- Function: Display and change the X index register  
Format: .X nnnn [addr] return
- Description: This command displays the target X index register value (nnnn). The value may be changed by entering a new value. A carriage return terminates the command.
- Function: Display and change the Y index register  
Format: .Y nnnn [addr] return
- Description: This command displays the target Y index register value (nnnn). The value may be changed by entering a new value. A carriage return terminates the command.
- Function: Display and change the A accumulator  
Format: .A nn [byte] return
- Description: This command displays the target A accumulator value (nn). The value may be changed by entering a new value. A carriage return terminates the command.
- Function: Display and change the B accumulator  
Format: .B nn [byte] return
- Description: This command displays the target B accumulator value (nn). The value may be changed by entering a new value. A carriage return terminates the command.
- Function: Display and change the condition code register  
Format: .C nn [byte] return
- Description: This command displays the target condition code register (nn). The value may be changed by entering a new value. A carriage return terminates the command.

Function: Display and change the direct page register

Format: .D nn [byte] return

Description: This command displays the target direct page register value (nn). The value may be changed by entering a new value. A carriage return terminates the command.

Function: Display and change the user stack pointer

Format: .U nnnn [addr] return

Description: This command displays the target user stack pointer (nnnn). The value may be changed by entering a new value. A carriage return terminates the command.

Function: Display and change the stack pointer

Format: .S nnnn [addr] return

Description: This command displays the target stack pointer (nnnn). The value may be changed by entering a new value. A carriage return terminates the command.

**3.6.2.2 Program Execution Control.** These commands control the execution of the target program. They permit the user to set and remove breakpoints, halt the program or generate a scope sync pulse when a specified address appears on the bus, and specify an ending address for a program trace.

Function: Set a breakpoint

Format: addr;V

Description: This command permits the user to specify a breakpoint in the breakpoint table of EXbug. A maximum of eight breakpoints can be entered. During an EXbug execute function, the breakpoints are inserted into the target program. When a breakpoint location is encountered, the program is halted to permit visual check printing out, or other performance analysis of the processor program registers. The breakpoint sequence is:

- User designates the breakpoint locations. A breakpoint cannot be set at an absolute address of 0000. Also, since an SWI instruction is inserted into a breakpoint location, breakpoints should only be set on the first byte of an instruction, and only be used in portions of the target program where the stack pointer is pointing to a valid stack area.
- User initiates the target program through the use of the program execute command (;G, addr;G, and ;P). Breakpoints are not inserted in memory during trace operations.

- When a breakpoint is encountered, control is returned to EXbug, and the contents of the processor registers are printed. Breakpoints are inserted in the map where the target program execution begins, regardless of what map was in use when the breakpoint addresses were entered.

#### NOTE

When an abort occurs, all breakpoints in memory are removed. However, breakpoints are not removed from memory during the restart sequence. In both cases, abort and restart, the table of breakpoint addresses is cleared. Following a restart during which breakpoints are active, the user will have to manually restore the original instructions, using the memory change function. Also, when EXbug encounters a memory location where a breakpoint cannot be inserted (e.g., read only memory), it will print a question mark, sound the terminal bell, and issue another prompt as an error indication. If this occurs, all instructions that are normally saved for re-insertion into the program are lost. These instructions may be restored, using the memory change function. It is also necessary to keep the U stack out of the S stack when using breakpoints if the U stack is to maintain its integrity.

Function: Display the breakpoint addresses

Format: ;V or \$V

Description: This command displays the table of absolute addresses at which breakpoints are set. An address of 0000 indicates that the associated location in the table does not contain a breakpoint address.

Function: Remove a specified breakpoint

Format: addr;U

Description: Regardless of the map that is in use, the command removes the specified address from the breakpoint table. If a breakpoint is not set at the specified address, a question mark is printed and the terminal bell is sounded.

Function: Remove all breakpoints

Format: ;U

Description: Regardless of the map that is in use, the command clears all addresses from the breakpoint table.

**Function:** Enable and change the Halt on Address or Scope Sync

**Format:** \$H nnnn [addr] return

**Description:** This command enables, displays, and allows the user to change the Halt on Address/Scope Sync address. The selection between Halt on Address or Scope Sync is determined by the position of the appropriate switch on the DEbug Module (refer to par. 3.2.2). In Halt on Address operation, control will be returned to EXbug when the specified absolute address appears on the system bus as a valid memory address. Since the address match causes an NMI, program execution will be stopped after the instruction that accessed the halt address has been executed.

In the Scope Sync mode, a pulse will be generated at the scope trigger pin on the DEbug Module each time the specified address appears as a valid address on the bus. The Halt/Sync address is set in the map in which program execution begins, regardless of the map in which it was enabled. Thus, in the Single Map mode, the command USER should be entered before starting a program with Halt/Sync enabled, since all addresses below F000 come from the User map. Once enabled, the Halt/Sync will remain active until it is disabled or an abort or restart is performed. When enabled, the Halt/Sync is active only while executing the target program (following a program execution command).

**Function:** Disable the Halt on Address or Scope Sync

**Format:** ;H

**Description:** Regardless of the map that is in use, the command disables the Halt-on-Address/Scope Sync function.

**Function:** Enable and change the Trace to Ending Address

**Format:** \$T nnnn [addr] return

**Description:** This command enables the Trace to Ending Address function, displays the address, and allows the user to change the ending address. Once enabled, the Trace to Ending Address is initiated by starting program execution with the PROCEED command (;P). EXbug will continue tracing until the trace program counter is equal to the ending address. Therefore, the ending address should be the first byte of an instruction. During the trace, entering Control-W will cause the trace to pause until some other character is entered. Entering Control-X will abort the trace and return control to EXbug. Once enabled, the Trace to Ending Address remains enabled until it is disabled or an abort or restart is performed. Since the trace operation uses an NMI and the stack, tracing should not be used unless the stack pointer is pointing to a valid stack area. Also, SWI instructions should not be traced since some SWI instructions are serviced by EXbug. CWAI instructions cannot be traced because the trace-NMI would cause the CWAI to continue and not wait for the user interrupt.

Function: Disable the Trace to Ending Address

Format: ;T

Description: This command disables the Trace to Ending Address function.

**3.6.2.3 Program Execution.** These commands permit the user to execute the target program. The various program execution commands permit starting the target program through its restart vector or at a specified address, proceeding with program execution, and tracing one or more instructions.

Function: Start the target program through its restart vector

Format: ;G

Description: This command starts the target program through its restart vector. In the USER mode, the restart vector is obtained from locations FFFE and FFFF; while in the EXEC mode, the restart vector is obtained from the target program top of memory as specified at FF00 and FF01. (See the Start-up Procedures section.) Therefore, when using ;G in the Single Map mode, EXbug should be in the EXEC mode and the top of memory address should be set up appropriately.

This command cannot be used to initiate a Trace to Ending address. If Trace to Ending address is enabled when this command is entered, EXbug will print a question mark, sound the terminal bell, and issue another prompt.

On entering the target program, the stack pointer, condition code register, and direct page register will contain the last target value obtained by EXbug. The contents of the A and B accumulators and the X, Y, and U registers are indeterminate. The user should ensure that the stack pointer is pointing to a valid stack area before any debug functions, such as breakpoints or a Halt on Address, are encountered in the target program. This can be accomplished by specifying the stack pointer value, using the .S command before entering the ;G command, or by executing an LDS immediate instruction as the first instruction of the target program.

Function: Start the target program at a specified address

Format: addr;G

Description: This command starts the target program at the specified address. A Trace to Ending Address function cannot be initiated with this command. If Trace to Ending Address is enabled when this command is entered, EXbug will print a question mark, sound the terminal bell, and issue another prompt.

On entering the target program, the stack pointer, condition code register, and direct page register will contain the last target value obtained by EXbug. The contents of the A and B accumulators and the X, Y, and U registers are indeterminate. The user should ensure that the stack pointer is pointing to a valid stack area before any debug functions such as breakpoints or a Halt on Address are encountered in the target program. This can be accomplished by specifying the stack pointer value, using the .S command before entering the ;G command, or by executing an LDS immediate instruction as the first instruction of the target program.

Function: Proceed with target program

Format: [value];P

Description: This command resumes target program execution using the target register values. If a value is entered, then the point of program continuation must be at a breakpoint location. The value specifies the number of times the breakpoint location is to be passed before the breakpoint returns control from other breakpoints while the pass count is in effect, unless they also have a non-zero pass count. A pass value will not be accepted if the Trace to Ending Address is active. If a pass value is entered while the Trace to Ending address is active, EXbug will print a question mark, sound the bell in the terminal, and issue another prompt.

This command should not be used to resume program execution at an SWI or CWAI instruction if a breakpoint is set at that instruction. Since continuing at a breakpoint causes an NMI, the CWAI instruction will not wait for the user interrupt. The breakpoint at the SWI will prevent it from being serviced as a user SWI, but as a breakpoint.

This command can be used to initiate a Trace to Ending address if a pass value is not entered. Breakpoints are not active during a Trace to Ending address.

Function: Trace the next instruction

Format: [value];N

Description: This command traces the next instruction. If a value is entered, it specifies the number of instructions to trace. After each instruction is executed, the contents of the registers are displayed. If multiple instructions are traced, entering Control-W will cause the trace to stop until some other character is entered. Entering Control-X will cause the trace to abort.

Since the trace function uses NMI, CWAI instructions should not be traced because CWAI instructions will not wait for the user interrupt, but will continue due to the NMI. Also, SWI instructions cannot be traced due to the servicing of some SWI's by EXbug and the fact that the stack has a high chance of being destroyed. Because the trace NMI uses the S stack pointer, tracing should only be done in portions of the program where the stack pointer is pointing to a valid stack area. In all cases, if the User stack pointer is being used, it should not be placed in the S stack during a trace. It will be overwritten if it is.

**3.6.2.4 Memory Parity Control.** These commands provide the user with control over memory parity functions. Included are commands to initialize memory with a specific pattern and to enable and disable the memory parity error interrupt function.

Function: Initialize memory to a specific pattern  
Format: byte;I  
Description: This command initializes random access memory to the byte value entered. After the command is entered, EXbug requests the beginning and ending addresses of the memory region to be initialized. The beginning and ending addresses are entered as described in the PRNT command. After valid beginning and ending addresses have been entered, the memory is initialized. The byte value entered is put in each memory location, starting at the beginning address through the ending address.

Since the state of the memory is indeterminate when power is first turned on, individual byte parity may be in error. Therefore, the memory with parity should be initialized by writing to it before it is read with the parity error interrupt enabled. Writing to the memory can be accomplished by using this command, or by loading a program.

Function: Enable memory parity error interrupt  
Format: ;;  
Description: This command enables the memory parity error interrupt. When a memory parity error interrupt is enabled, an NMI will be generated, which returns system control to EXbug. EXbug will then print PARITY, followed by a printout of the interrupted registers. Note that the program counter value displayed will not be pointing to the instruction being executed while the parity error occurred, but will be pointing to the next instruction to be executed after the parity error occurred.

NOTE

While writing to the disk, a memory parity error will generate a disk time-out error instead of a parity error message.

In order to prevent any pending interrupts from occurring when the error interrupt is enabled, the memory initialization command I should be used immediately before the parity error interrupt enable command. Using the memory change function to write a location will also clear any pending parity error interrupts.

Function: Disable memory parity error interrupt  
Format: ;:  
Description: This command disables the memory parity error interrupt. Following this command, a memory parity error will not generate an NMI. This is the default mode in EXbug.

**3.6.2.5. I/O Control.** These commands provide the user with control over EXbug I/O functions. Included are commands to specify the number of nulls to be padded after a carriage return or other characters, and to direct the EXbug output to a line printer.

Function: Display and change the terminal null pad values

Format: ;K nnnn [value] return

Description: This command specifies the control codes used to control the console reader and punch, the number of nulls to be padded after a carriage return, and the number of nulls to be padded after all other characters. The null pad is required for terminals that have a mechanical carriage , and it cannot turn in a single character time. The null pad value is a 16-bit value. When a value is entered, leading zeros are assumed.

The most significant bit, bit 15, of the null pad value controls which console reader control codes are used. When this bit is zero, the normal ASCII DC1 and DC3 codes are used to turn the reader on and off. When this bit is one, the TI Silent 700 RDC codes are used to read a block of tape from the console. Also, when bit 15 is a one, the TI Silent 700 RDC codes are used to turn the terminal printer off before sending data to the terminal punch, and to turn the printer back on when punching is completed.

The eight least-significant bits, 0 through 7, specify a binary number which is the number of nulls sent to the terminal after a carriage return is sent. This number is the last two hexadecimal digits printed and entered.

The remaining seven bits, 8 through 14, specify a binary number which is the number of nulls sent to the terminal after any character other than a carriage return is sent.

The following values of the null pad parameter are used for TI Silent 700 terminals at the baud rate listed:

<u>BAUD RATE</u>	<u>K VALUE</u>
300	4
1200	8317
2400	872F

Since the null pad value is initialized to zero on power-up and restart, a terminal that requires null pads will not print properly until the the appropriate null pad value is entered. Even though the terminal may not correctly print the current value, the appropriate value can be entered and will be echoed to the terminal.

Function: Display and copy the terminal output to line printer option

Format: ;Z nn [byte] return

Description: This command displays the status of the line printer interface. When the nn value is 0, the line printer interface is not initialized and data is only displayed at the terminal. The 0 indication is a default value following a power-up, restart, or abort. When the nn value is non-zero (1), the line printer interface is initialized and the terminal output is sent to the line printer. The printer output is not paged, but is continuous. To initialize the line printer, a [byte] value of 1 must be entered.

EXbug uses the line printer routines in the ROM on the Floppy Disk Controller Module to initialize the printer interface and send characters to the printer. A listing of these routines is provided in Figure 3-9. EXbug calls the LPINIT entry when the line printer interface is enabled. The LIST entry point is used by EXbug to send characters to the line printer. If EXbug detects a printer error by the carry bit being set on return from LIST, EXbug disables the line printer interface. If the ROM is not in the system, then equivalent line printer routines must be provided for EXbug if the line printer output feature is to be used.

User program output, directed through various EXbug entry points, will also be directed to the line printer under control of the line printer interface (see par. 3-7). The line printer routine is contained in the EXbug I/O listing of Appendix B, and can be controlled by the user program. However, the user program must guarantee that the printer interface has been initialized before setting the switch to non-zero.

**3.6.2.6 Memory Search.** These commands control the memory search function. Commands are included to establish the search address range and comparison mask and to initiate the memory search.

Function: Establish search address range and comparison mask

Format: ;M or \$M

Description: This command first requests the search address range as described in the PRNT command. Memory will be searched from the beginning address specified through the ending address. After a valid address range is entered, the command requests the search comparison mask in the following manner:

MASK = nn [byte] return

nn is the hexadecimal representation of the current mask. If it is to be changed, a new value can be entered. A carriage return terminates the command. The mask specifies which bits in each byte are to be checked against the search value. Only those bit locations set to a one in the mask will be compared. For example, a mask of FF would compare each bit in the byte during the search, while a value of 01 would compare only bit 0, the last significant bit.

00761

TTL LINE PRINTER DRIVER

00763 \*  
 00764 \* LINE PRINTER DRIVER FOR CENTRONICS TYPE  
 00765 \* INTERFACE THROUGH A PIA WITH OUTPUT  
 00766 \* CHARACTER ON A SIDE, INPUT STATUS ON B SIDE  
 00767 \* 6809 VERSION  
 00768 \*  
 00769 \* VERSION 1.2 19 FEB 1979  
 00770 \* COPYRIGHT 1979 BY MOTOROLA INC  
 00771 \*

00773 \* PIA ADDRESSES  
 00774 EC10 A DATA EQU \$EC10  
 00775 EC11 A CNTRL1 EQU \$EC11  
 00776 EC12 A STAT EQU \$EC12  
 00777 EC13 A CNTRL2 EQU \$EC13

00779A EBB4 ORG RMSTRT+\$3B4  
 00780 \* STROBE PRINTER  
 00781 EBB4 A LIST5 EQU \*  
 00782A EBB4 8D 2A EBE0 BSR LERROR CLEAR ACKNOWLEDGE  
 00783A EBB6 86 34 A LDA #\$34  
 00784A EBB8 8D 02 EBBC BSR LIST7  
 00785A EBBA 86 3C A LDA #\$3C  
 00786A EBBC B7 EC11 A LIST7 STA CNTRL1  
 00787A EBBF 39 RTG

00789 \* SUBROUTINE TO INITIALIZE PIA  
 00790 EBC0 A LPINIT EQU \*  
 00791A EBC0 8E FF3C A LDX #\$FF3C A DATA OUTPUT  
 00792A EBC3 BF EC10 A STX DATA  
 00793A EBC6 86 3C A LDA #\$3C B STATUS INPUT  
 00794A EBC8 B7 EC13 A STA CNTRL2  
 00795A EBCB 39 RTG

00797 \* SUBROUTINE TO PRINT CHARACTER FROM A ACC  
 00798 \* AND CHECK FOR PRINTER ERROR  
 00799 \* IF ERROR CARRY IS SET ON RETURN  
 00800 EBCC A LIST EQU \*  
 00801A EBCC B7 EC10 A STA DATA SEND DATA  
 00802A EBCF 43 COMA SET ERROR STATUS  
 00803A EBD0 8D E2 EBB4 BSR LIST5 SEND STROBE  
 00804A EBD2 B6 EC12 A LIST3 LDA STAT CHECK STATUS  
 00805A EBD5 84 03 A ANDA #3 BIT 0=SELECT, BIT 1=PA  
 00806A EBD7 4A DECA A SHOULD HAVE BEEN 01  
 00807A EBD8 26 06 EBE0 BNE LERROR NO PAPER OR NOT SELECT  
 00808A EBDA 7D EC11 A TST CNTRL1 ACKNOWLEDGE?

FIGURE 3-9. MDOS Line Printer Driver

00809A	EBDD	2A	F3	E8D2	BPL	LIST3	NO
00810A	EBDF	4F			CLRA		YES, CLEAR ERROR STATUS
00811A	E8E0	B6	EC10	A LERROR	LDA	DATA	RESTORE A
00812A	E8E3	39			RTS		

00814	* SUBROUTINES TO PRINT STRING AND STRING, CR, LF							
00815	E8E4	A	LDATA	EQU	*			
00816A	E8E4	86	OD	A	LDA	#\$D	SEND CR	
00817A	E8E6	8D	E4	EBCC	LDATA7	BSR	LIST	
00818A	E8E8	25	FC	E8E6		BCS	LDATA7	HANG UP ON ERROR
00819A	E8EA	86	0A	A	LDA	#\$A	SEND LF	
00820A	E8EC	20	OO	EBEE		BRA	LDATA3	HOLD LDATA1 ENTRY POINT
00821A	E8EE	8D	DC	EBCC	LDATA3	BSR	LIST	
00822A	E8F0	25	FC	EBEE		BCS	LDATA3	HANG UP ON ERROR
00823A	E8F2	A6	80	A	LDATA1	LDA	O, X+	
00824A	E8F4	81	04	A		CMPA	#4	EOT?
00825A	E8F6	26	F6	EBEE		BNE	LDATA3	NO
00826A	E8F8	30	1F	A		LEAX	-1, X	YES, CORRECT X
00827A	E8FA	39				RTS		

FIGURE 3-9. MDOS Line Printer Driver (cont'd)

TABLE 3-2. Dual Map Mode Second Level SWI Options

MAP SWI IS IN	MAP SWI it SERVICED IN	REQUIRED E VALUE
user	user	01
user	executive	FF
executive	executive	FF
executive	user	not permitted

Function: Search memory for a byte (word)

Format: byte;W

Description: This command searches memory over the last beginning-ending address range specified for a match with the value entered. Only those bit positions set to one in the last comparison mask entered are compared during the search. The same beginning and ending address parameters are used for the PRNT, the PNCH, the I, and the M commands. Therefore, if one of these commands is entered after the M command and before the W command, the beginning and ending addresses specified for the last such command entered will be used for the W command.

When the memory search finds a match, it prints the memory address of the match and the contents of memory. Entering Control-W while this printout is occurring causes the search command to wait until some other character is entered. Entering Control-X during the printout causes the search to abort and return to the EXbug command level.

**3.6.2.7 Miscellaneous.** These commands control various EXbug functions. They permit the user to specify a default debug offset to be used with address parameters that are entered, and also control EXbug responses to SWI's that are not breakpoints.

Function: Display and change the default debug offset

Format: ;Q nnnn [value] return

Description: This command displays, and permits the user to change, the default debug offset. The debug offset is added to a single parameter entered as an address value in an EXbug command to determine the absolute address. When two parameters are entered as an address value, the sum of the two parameters is the absolute address used. To change the debug offset, enter a new value. A carriage return terminates the command.

Use of the debug offset permits the user to easily test programs assembled with the relocatable option. Once the debug offset is set to the main program section, all references to the main program section made in EXbug can be accomplished simply by entering the relative address given in the assembly listing. References can also be made to other load sections by entering both the relative address and the section starting address, separated by a comma, in place of the single address parameter. EXbug then uses the sum of these two values as the absolute address.

**Function:** This command permits the user to display and change the option in which second level SWI's are serviced.

**Format:** ;E nn [byte] return

**Description:** The second level SWI enable controls the operation of second level SWI instructions that are not breakpoints. The location of the vector depends upon the map that is in use (User or Executive) and whether the system is in the Single or Dual Map mode. The value of nn indicates the map where the second level SWI's are to be serviced. To change the map where second level SWI's are serviced, a new value, [byte], can be entered.

To use the second level SWI feature in the Single Map mode, the value of nn should be set to FF. Table 3-2 lists the various second level SWI options supported in the Dual Map mode. When the value of nn is zero (which is the default value following a power-up, restart, or abort sequence), non-breakpoint SWI's in the Single Map mode or in the Executive map of the Dual Map mode return control to EXbug which, in turn, prints a breakpoint error message. EXbug prints "SWI" followed by the register values when the SWI is encountered. EXbug then issues a prompt. A zero value causes SWI's in the User map of the Dual Map mode to return control to EXbug and print the SWI message only if breakpoints have been set in the User map. However, if the nn value is zero, and no breakpoints are set in the User map, then SWI's in the User map of the Dual Map mode cause program control to be given to the location pointed to by the SWI vector (contained in addresses \$FFFA and \$FFFFB of the User map).

### 3.6.3 Memory Change

The Memory Change function permits the user to examine and change individual memory locations. To invoke the Memory Change function, the user enters:

addr/

Here again, addr is either a single parameter that is added to the debug offset to determine the absolute address, or two parameters separated by a comma that are added together to determine the absolute address. After the user enters the slash, EXbug prints a space, the contents of the specified location in hexadecimal, and then another space. If the memory contents are to be changed, the user may enter a new hexadecimal value. Next, the user enters one of the following characters to close the current memory location:

Carriage return This ends the memory change function and returns control to the EXbug command level. EXbug then prompts the user.

Line Feed This causes the next sequential memory location to be opened for memory change and its contents displayed.

Space This causes the previous sequential memory location to be opened for memory change and its contents displayed.

Slash This causes the current memory location to be reopened for memory change and its contents displayed.

If memory is being changed, but it does not change properly, an error indication will be displayed. A space will be printed, then a question mark, the terminal bell will be sounded, and another space will be printed. The memory contents after the attempted memory change are then printed. The Memory Change function then continues as requested by the terminating character (carriage return, line feed, space, or slash).

Memory locations displayed by the Memory Change function, after it has been invoked, are in the form -- absolute address, space, contents, space.

Also, the Memory Change function will calculate the required offset for a long or short relative addressing mode instruction (addr;L or addr;0 command). To calculate a relative address offset, first open the memory location that is to contain the offset (e.g., the second byte of a branch instruction). Next, the destination address is entered, followed by a semicolon and the capital letter L for long relative, or 0 for short. If a single parameter is entered for the destination address, the debug offset will be added to it to determine the absolute destination address. If two parameters separated by commas are entered for the destination address, they will be added together to determine the absolute destination address.

The Memory Change function will indicate that the destination address is out of range by printing a space, a question mark, and sounding the terminal bell. If the destination address is in range, the correct offset will be printed. In both cases, the address and contents of the currently open location will be redisplayed on the next line, permitting the user to easily modify it or request another relative offset calculation.

**3.6.3.1 Adding EXbug Commands.** The user has the ability to add as many four-character commands as desired. The only limiting factor is memory size. In order to implement this feature, the user must have a table of his commands and the actual commands stored in the Executive memory map (if the Dual Map mode is in use), and must have told EXbug where his command table resides. The user command table format must be as follows:

Example:

```
CTBEG EQU*      Command table beginning
FCC/CMD1/    Four character command
FDB CMD1E    Entry address of command
FCC/CMD2/    Four character command
FDB CMD2E    Entry address of command
        .
        .
        .
        .
        .
FCC/CMDN/    Four character command
FDB CMDNE    Entry address of command
CTBENDEQU*   Command table end
```

Once the user command table is stored in memory, EXbug must be informed of its location by having the beginning address of the table (the value of CTBEG in the above example) put at locations \$FF0E and \$FF0F, while the ending address of the table (the value of CTBEND in the above example) is put at locations \$FF10 and

\$FF11. In both of these cases, the addresses are loaded into memory in the order of most significant byte first, followed by the least significant byte. If the command table and commands are loaded from a tape, the tape may contain an object code that will properly initialize these locations. This object code may be generated by the ORG and FDB statements in the source. For the above example, the source code required to generate the proper object code to initialize these locations would be:

Example: ORG \$FFOE  
FDB CTBEG, CTBEND

NOTE: An ORG statement or END statement would be required after the two source lines shown above, so that the object code would not be produced at location \$FF12 or beyond.

If the command table and commands are loaded from an MDOS file, a short program can be included in the file that would initialize these locations and then give control to EXbug when it is executed. Programs cannot be loaded from the disk at these locations.

Pressing the ABORT pushbutton will not modify locations \$FFOE through \$FF11. However, pressing the RESTART pushbutton will cause these locations to be restored to the EXbug values. These locations will also be restored to the EXbug values when power is initially applied. Thus, following a restart, the user must restore the beginning and ending addresses of his table (if required) in memory locations \$FFOE through \$FF11. If the user does not wish to add commands, no operation is needed.

On entry to the user command, the stack pointer will be pointing at two locations below the top of the EXbug stack area; the A accumulator will contain \$20; and the contents of the B accumulator and X, Y, U, CC, and DP registers will be unspecified. None of these values need be restored before returning to EXbug. However, IRQ and FIRQ are normally made while EXbug is running. User commands that are intended to return to EXbug without affecting the current states of EXbug variables, should return by jumping to location \$F5C2. User programs that are intended to return to EXbug and initialize EXbug variables, should return by jumping to location \$F564.

### 3.7 EXbug SUBROUTINES AND ENTRY POINTS

This paragraph lists and describes the subroutines in EXbug that are available to run programs in the EXORciser. Since EXbug is in the Executive map, any program that uses EXbug routines must also be in the Executive map if the EXORciser is being operated in the Dual Map mode. Also, programs run in the Single Map mode can use EXbug routines. Table 3-3 lists the available routines. A listing of the first 1K of EXbug, which contains most of these routines, is provided in Appendix B. This is the 1K ROM on the DEbug Module. This ROM can be replaced by a user-provided ROM if he wishes to modify the I/O or restart sequence. However, the other 2K of EXbug makes references to the first 1K, as given in cross reference symbol table of Appendix B. These references must be provided for in a user-installed ROM for proper operation of EXbug. In order for programs that use EXbug to be compatible with past and future versions of EXbug, they should only use the routines listed in Table 3-3, and only at the addresses given in that table.

TABLE 3-3. EXbug Routines

ENTRY ADDR	MNEMONIC	FUNCTION	Page Ref.
F000	PWRUP	ENTER EXBUG FROM RESTART	3-30
F003	XBEGEN	INPUT START & END ADDRESSES	3-31
F006	XCBCDH	CONVERT HEX TO BCD	3-31
F009	XCHEXL	CONVERT MS BCD TO HEX (ASCII)	3-32
F00C	XCHEXR	CONVERT LS BCD TO HEX (ASCII)	3-32
F00F	XINADD	INPUT HEX ADDR INDIRECT (X)	3-32
F012	XINCH	INPUT ONE CHARACTER	3-33
F015	XINCHN	INPUT ONE CHAR NO PARITY	3-33
F018	XOUTCH	OUTPUT CHAR (WITH SPEED FILL)	3-33
F01B	XOUT2H	PRINT 2 HEX CHAR (X)	3-34
F01E	XOUT4H	PRINT 4 HEX CHAR (X)	3-34
F021	XPCRLF	PRINT C/R L/F (Uses A)	3-34
F024	XPDATA	PRINT C/R L/F + DATA STRING	3-35
F027	XPDAT1	PRINT DATA STRING (Enter with X)	3-35
F02A	XPSPAC	PRINT SPACE	3-35
F0F3	F0F3	REENTER EXBUG COMMAND LEVEL	3-36
F564	F564	REENTER EXBUG COMMAND LEVEL	3-36
F5C2	F5C2	REENTER EXBUG COMMAND LEVEL	3-36
F8A4	F8A4	READ OBJECT RECORD	3-36
F9CF	F9CF	OUTPUT CHAR (NO SPEED FILL)	3-37
FBFB	RTNUSR	RETURN TO USER MAP FROM SWI	3-37

Except as stated in the descriptions, all of these are subroutines, end with an RTS, and should be called with a JSR. Control will be returned to the next instruction following the JSR, providing the stack pointer and stack memory are properly implemented. Most of these routines involve input or output of data on the terminal that is connected to the EXORciser. The routines that involve input from the keyboard (or tape reader) will sit and wait (in a loop) until the character is input; then it will return. Unless indicated otherwise, routines that output to the terminal are affected by the Z option. That is, output through these routines will be sent to the line printer, as well as the terminal, if the Z option is on (non-zero). The Z flag is kept in location ZFLAG (\$FF32) and can be modified by the program. (Refer to the Z command description.)

Name: PWRUP -- Power-up and restart entry

Function: Configure EXbug and its peripherals from a restart or power-up condition

Call: JMP PWRUP

Input: None

Output: EXbug parameters are initialized along with the EXbug peripheral devices. the EXbug start-up message is sent to the terminal.  
NOTE: Control is not returned to the calling program, but is given to the EXbug command input routine.

Name: XBEGEN -- Input Start and End Addresses

Function: Request Input of Beginning and Ending Addresses as defined in the PRNT command. Verify inputs are hexadecimal characters. Verify ending address is larger than beginning address.

Call: JSR XBEGEN

Subroutine Input: None

Subroutine Output: \$FF0A BEGA 16 Bit Beginning Address  
\$FF0C ENDA 16 Bit Ending Address

NOTE

Acc A and B and the X and Y Index Registers are used by this subroutine. If their contents are meaningful, they must be saved prior to calling this subroutine. If single parameters are entered for BEG or END, the default debug offset (Q in locations \$FFE6, \$FFE7) will be added to them to determine BEGA and ENDA. If two parameters separated by a comma are entered, they will be added together to determine the address being entered. The calling program may modify Q to specify the default debug address. However, if it does this, the new value of Q will be used by EXbug as the default debug offset.

Name: XCBCDH -- Convert a hexadecimal character to a binary number

Function: Verify input is a hexadecimal digit character. Convert character to a 4-bit binary number with HI order 4 bits equal zero. Set N (negative) condition code for non-hexadecimal characters.

Call: JSR XCBCDH

Subroutine Input: Character to convert must be in Acc A

Subroutine Output: If hexadecimal character input, Acc A contains the 4-bit binary number represented by the input character, and the N (negative) condition code is cleared. If non-hexadecimal character input, Acc A contains the character input, and the N condition code is set. the B, X, Y, and U Registers are preserved.

Name: XCHEXL -- Convert most significant binary value to Hex  
Function: Convert the most significant 4 bits of Acc A to an ASCII coded hexadecimal digit character  
Call: JSR XCHEXL  
Subroutine Input: Contents of Acc A  
Subroutine Output: An ASCII coded hexadecimal digit character in Acc A. The B, X, Y, and U Registers are preserved.

Name: XCHEXR -- Convert least significant binary value to Hex  
Function: Convert the least significant 4 bits of Acc A to an ASCII coded hexadecimal digit character  
Call: JSR XCHEXR  
Subroutine Input: Contents of Acc A  
Subroutine Output: An ASCII coded hexadecimal digit character in Acc A. The B, X, Y, and U Registers are preserved.

Name: XINADD -- Input a hexadecimal address  
Function: Convert up to 4 input hexadecimal characters to a 16-bit binary address.  
Call: JSR XINADD  
Subroutine Input: X Index Register contains address to store result  
Subroutine Output: Most significant 8 bits of resultant 16-bit address will be stored into the memory location specified by the X Register. The least significant 8 bits will be stored into the next higher memory location. Acc A will contain last character input. Acc B will contain number of input hexadecimal characters. The X Register is unchanged. The subroutine returns to the calling program when an invalid character, or the fifth hexadecimal digit, is entered. The Y and U Registers are preserved.

NOTE

This address is not modified by the default debug offset.

Name: XINCH -- Input one character

Function: Wait for and accept input of one character from debug terminal and echo character back to terminal, if required.

Call: JSR XINCH

Subroutine Input: There is a no echo flag (AECHO) at \$FF58. It must be set non-zero before each call to XINCH for each character that is not to be echoed to the terminal (and line printer, if the Z option is on).

Subroutine Output: Acc A contains 8-bit input character as received from the debug terminal. XINCH clears AECHO if it was non-zero. The B, X, Y, and U Registers are preserved.

Name: XINCHN -- Input one character with no parity

Function: Wait for and accept input of one character from debug terminal and echo character back to terminal, if required. Clear HI order bit of input character.

Call: JSR XINCHN

Subroutine Input: There is a no echo flag (AECHO) at \$FF58. It must be set non-zero before each call to XINCHN for each character that is not to be echoed to the terminal (and line printer, if the Z option is on).

Subroutine Output: Acc A contains input character as received from the debug terminal with the HI order bit cleared. XINCHN clears AECHO if it was non-zero. The B, X, Y, and U Registers are preserved.

Name: XOUTCH -- Output Character

Function: Output one character with required speed fill

Call: JSR XOUTCH

Subroutine Input: Acc A contains character to output to the debug terminal (and to the line printer, if the Z option is on).

Subroutine Output: Acc a contains character output. The B, X, Y, and U Registers are preserved.

Name: XOUT2H -- Output two hexadecimal characters and a space

Function: Convert the contents of an 8-bit binary byte to two hexadecimal characters and output them, followed by a space character, to the debug terminal.

Call: JSR XOUT2H

Subroutine  
Input: The X Register contains address of the byte to be converted and output.

Subroutine  
Output: Acc A contains last character output. The X Register is incremented by one. The B, Y, and U Registers are preserved.

Name: XOUT4H -- Output four hexadecimal characters and a space.

Function: Convert the contents of two consecutive 8-bit binary bytes to four hexadecimal characters and output them, followed by a space character, to the debug terminal.

Call: JSR XOUT4H

Subroutine  
Input: The X Register contains address of the first byte to be converted and output.

Subroutine  
Output: Acc A contains last character output. The X Register contains the input address plus 2. The B, Y, and U Registers are preserved.

Name: XPCRLF -- Print CR/LF/Null

Function: Output a carriage return, a line feed, and a null character to the debug terminal with required speed fill.

Call: JSR XPCRLF

Subroutine  
Input: None

Subroutine  
Output: Acc A contains a null character (0). The B, X, Y, and U Registers are preserved.

Name: XPDATA -- Print CR/LF/Data string

Function: Output a carriage return, a line feed, and the user-specified string of data characters to the debug terminal.

Call: JSR XPDATA

Subroutine Input: The X Register will contain the starting address of user data string to output. Output string is terminated by an EOT (04) character.

Subroutine Output: The X Register will contain the address of the EOT character. Acc A will contain the EOT character. The B, Y, and U Registers are preserved.

Name: XPDAT1 -- Print Data String

Function: Output a user-specified string of data characters to the debug terminal.

Call: JSR XPDAT1

Subroutine Input: The X Register will contain the starting address of user data string to output. Output string is terminated by an EOT (04) character.

Subroutine Output: The X Register will contain the address of the EOT character. Acc A will contain the EOT character. The B, Y, and U Registers are preserved.

Name: XPSPAC -- Print space

Function: Output a space character to the debug terminal

Call: JSR XPSPAC

Subroutine Input: None

Subroutine Output: Acc A will contain a space character. The B, X, Y, and U Registers are preserved.

Name: F0F3 -- Reenter EXbug

Function: Entry point for programs to reenter EXbug. In 6800 EXbug 1, this is the MAID re-entry point. Reentering EXbug at this point with breakpoints active can cause unexpected results.

Call: JMP F0F3

Subroutine Input: None

Subroutine Output: Control is not returned to the calling program.

Name: F564 -- Reenter EXbug

Function: Entry point for programs to reenter EXbug. This entry point initializes most of the EXbug parameters. This is the recommended EXbug reentry address.

Call: JMP F564

Subroutine Input: None

Subroutine Output: Control is not returned to the calling program

Name: F5C2 -- Reenter EXbug

Function: Entry point for user-added, four-character commands to reenter EXbug

Call: JMP F5C2

Subroutine Input: None

Subroutine Output: Control is not returned to the calling program

Name: F8A4 -- Read object record

Function: Read an object record from the terminal tape reader to a memory buffer and convert the data from ASCII to binary. This routine continues to read records until an object record is read. The object format is described in Figure 3-4.

Call: JSR F8A4

Subroutine Input: None

**Subroutine Output:** The record type, ASCII 0, 1, or 9, is in BCONT (location \$FF91). The byte count is in location \$FF92. The rest of the record in hexadecimal, up to the checksum, begins in location \$FF93. An indication of the validity of the checksum is in BCKSM (location \$FF90). If this location contains zero, the checksum was correct. If it contains a non-zero value, the checksum was in error. The register values are indeterminate on return.

**Name:** F9CF -- Output character

**Function:** Output one character without speed fill

**Call:** JSR F9CF

**Subroutine Input:** Acc A contains the character to output to the debug terminal. The character is not sent to the line printer if the Z option is zero.

**Subroutine Output:** Acc A contains character output. The B, X, Y, and U Registers are preserved.

**Name:** RTNUSR -- Return to the user map

**Function:** Returns control to the user map following a user map SWI that was serviced in the Executive map. See E command description for further information. This entry point is in EXbug 2 only.

**Call:** JMP RTNUSR

**Subroutine Input:** The processor registers to be restored from the SWI must be on the stack as the next items that can be pulled off the stack.

**Subroutine Output:** Control is not returned to the calling program.



## CHAPTER 4

### SYSTEM DEVELOPMENT USING THE EXORciser

#### 4.1 INTRODUCTION

The EXORciser is a system development tool used in the design and development of M6809 Microprocessor Systems. This chapter contains an overview of the tasks of developing hardware and software, and an explanation of the role of the EXORciser in the development of the user system. The use of the EXORciser to emulate (functionally duplicate) the user system, or to connect to an existing microprocessor system, is described, and methods to debug the user system are shown.

The optional EXORciser modules are discussed in general terms. Refer to the User's Guide for the optional modules, and to this User's Guide for details in preparing the EXORciser to emulate your system, or to connect it to another system. It is assumed in this chapter that the M6809 Data Sheet is being used, as well as the M6809 Programming Manual. These are the manuals referenced by the expression "M6809 Manuals" used throughout this chapter.

#### 4.2 THE EXORciser IN SYSTEM DEVELOPMENT

The EXORciser reduces the time required for an engineer to construct a working model of a prototype system. This is accomplished by the ability of the EXORciser to emulate a user system hardware, and to debug the interfaces of external devices and user software. Rather than immediately designing and building a prototype of a system, the engineer sets up the EXORciser to functionally represent the system, using the plug-in optional modules mentioned here and described in detail in the separate User's Guides.

If the designer has already built a prototype or production model of his system before acquiring the EXORciser, he can proceed to the later sections of this chapter for descriptions of the use of the EXORciser with external microcomputer hardware.

#### 4.3 PERIPHERAL INTERFACING

When a system is being designed to use the MC6809, MC68A09, or MC68B09 Microprocessor, it must be determined how to control the hardware, and how much of the system logic can be done in software. The hardware typically has functions that need to be controlled, and sensors that need to be monitored. The EXORciser is an operating microcomputer in modular form, which can be interfaced to the user hardware. Since most external devices can be interfaced with 6820 or 6821 Peripheral Interface Adapters (PIA's), and use TTL compatible signals, an I/O Module would, typically, be plugged into the EXORciser, and the PIA I/O pins wired to the user peripheral device via a flat ribbon cable. (It is assumed that the user circuits are TTL compatible.) Once this is done, the user can test the hardware interface by using the EXbug Display/Change Memory command, as described in Chapter 3 (i.e., since PIA registers are like memory locations in 6809 systems, storing to the PIA data register, after it has been programmed as an output port, will put signal levels on the I/O lines in accordance with the data word stored). For example, depending on the system hardware, a data word can be selected to cause a motor to run, a solenoid valve to open, or a relay to close. When such a word is entered, it will frequently be found that the correct action did not happen, in which case a scope or meter

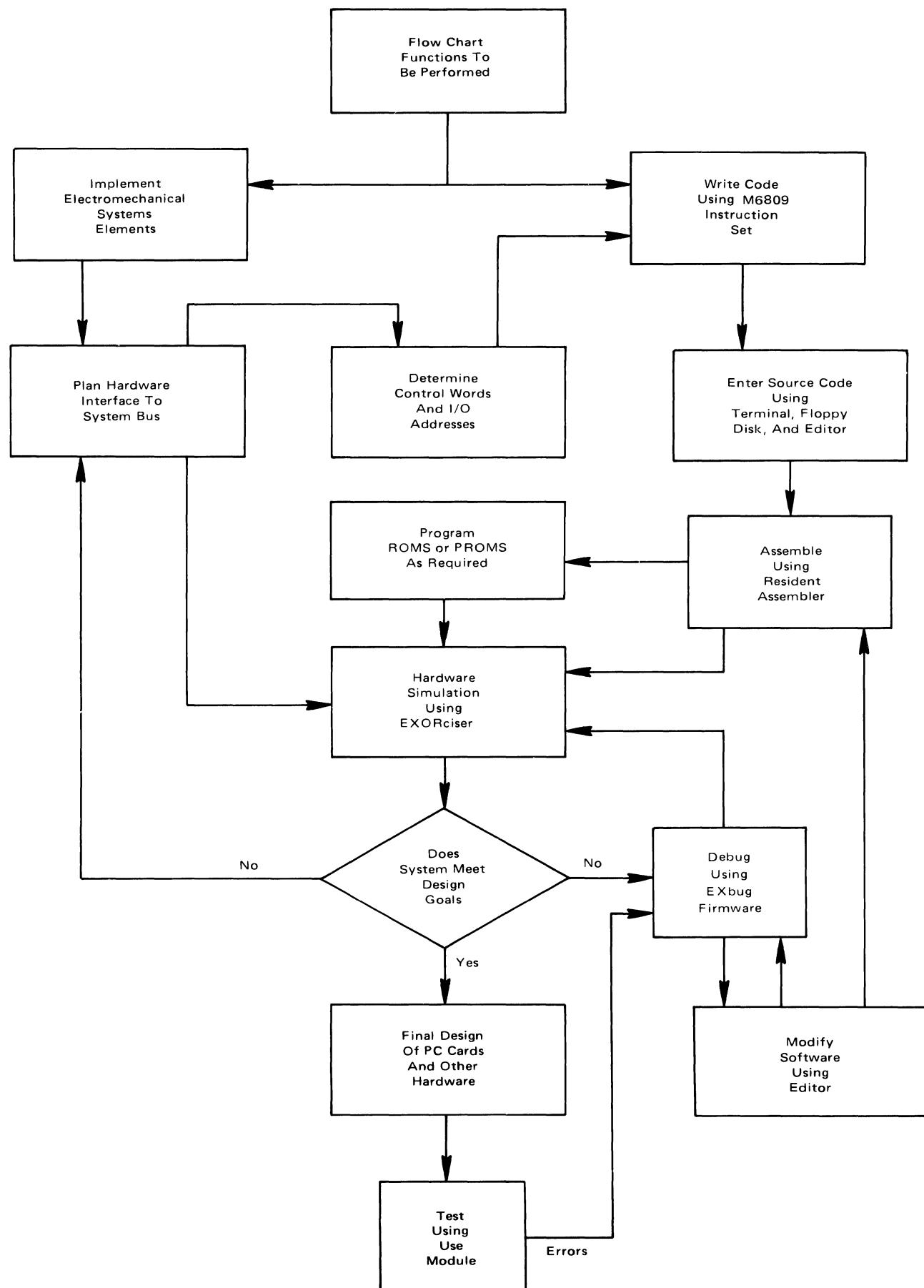


FIGURE 4-1. System Designing and Verifying Procedure

can then be used to observe changes on the output lines and/or peripheral devices, and the wires moved or the control word changed to correct the problem. Once the proper control words have been determined, a similar analysis is made of the data word read from the PIA input data register. The bits of this word are the result of the data levels on the lines from the sensor hardware. When the control and status words are known, a routine can be entered into memory, in machine language, using EXbug, to exercise the external hardware. When everything has been tried and it is fully understood, a complete program can be written and assembled on the EXORciser, using the optional resident Editor/Assembler programs. The program for this subsystem is then loaded into memory, and tested by using breakpoints or "run-one-instruction" methods (see Chapter 3). When finalized, the same technique is repeated for other subsystems and their additional peripheral units until a whole system is assembled. This entire process is depicted in Figure 4-1. It is seen in that figure that each step has feedback paths so the hardware or software can be improved any number of times until the desired performance is reached.

The engineer now has an operating model of his system using EXORciser hardware, with a minimum amount of time spent on prototype construction. The programmer can now finalize his routines.

The advantages of the EXORciser to the programmer are that he can verify his I/O programming steps by testing them before he writes a source program, and it permits him to debug the resulting object code on a real time basis.

#### 4.4 PROCEDURE FOR DESIGN

To better understand using the EXORciser in the design and development of an M6809 Microprocessor system, let us review the procedure followed by a typical engineer in developing a microprocessor system, and how the EXORciser will simplify the design (see Figure 4-2). The engineer:

- a. Defines his system, using flow charts (or other means). In this definition, he determines the software and hardware functions to be performed.
- b. Sets up the EXORciser to emulate his system hardware (to be explained). If required, he also builds any special hardware interface circuitry to his peripherals.
- c. Prepares his software programs on the EXORciser, after testing the hardware required to accomplish the intended function, and determining the addresses and control words.
- d. Loads software into the EXORciser and, using the EXbug Firmware, debugs both the hardware and the software until he has a working system.
- e. Designs and builds a preproduction model of his system.
- f. Combines the preproduction hardware with the user system software and, using the EXORciser EXbug Firmware and the User System Evaluator (USE\*) module, debugs and makes any hardware and software adjustments.
- g. Extensively tests and evaluates his system in an actual working environment. At this point, the program may be stored in PROM's, using the optional PROM Programmer.

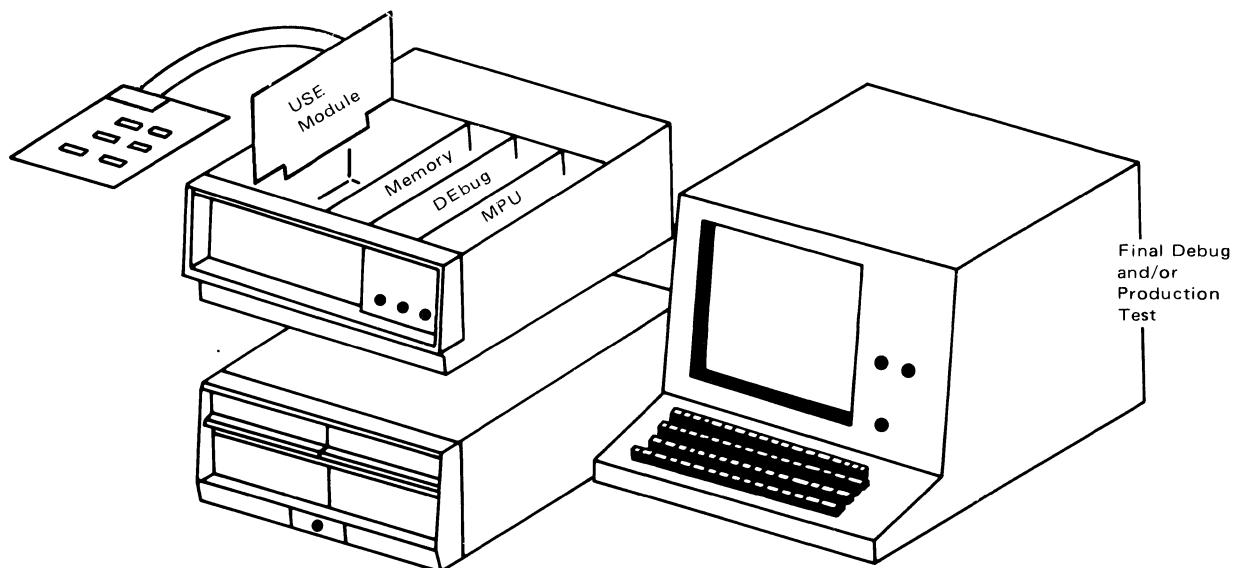
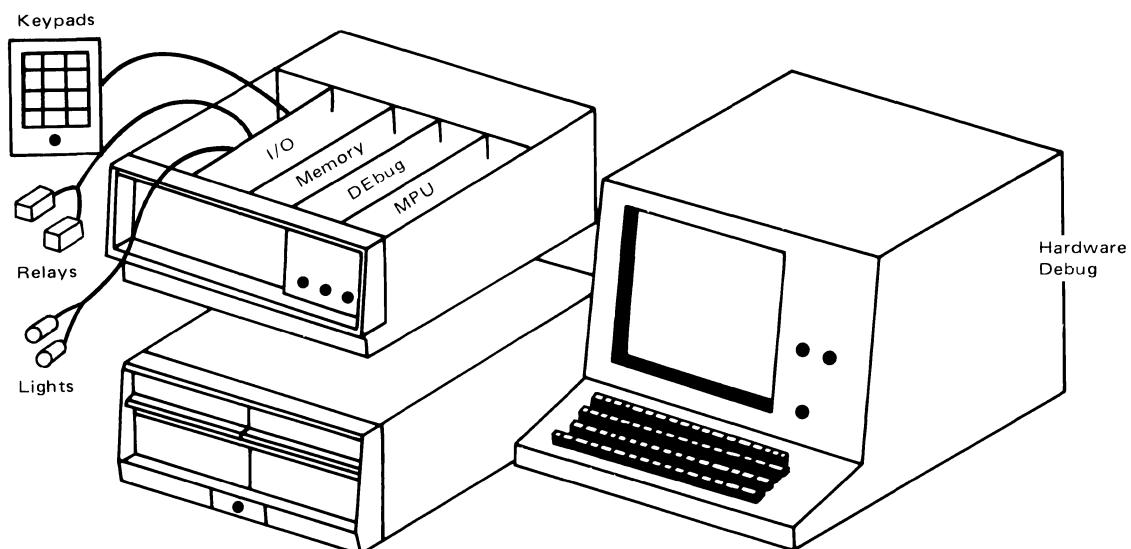
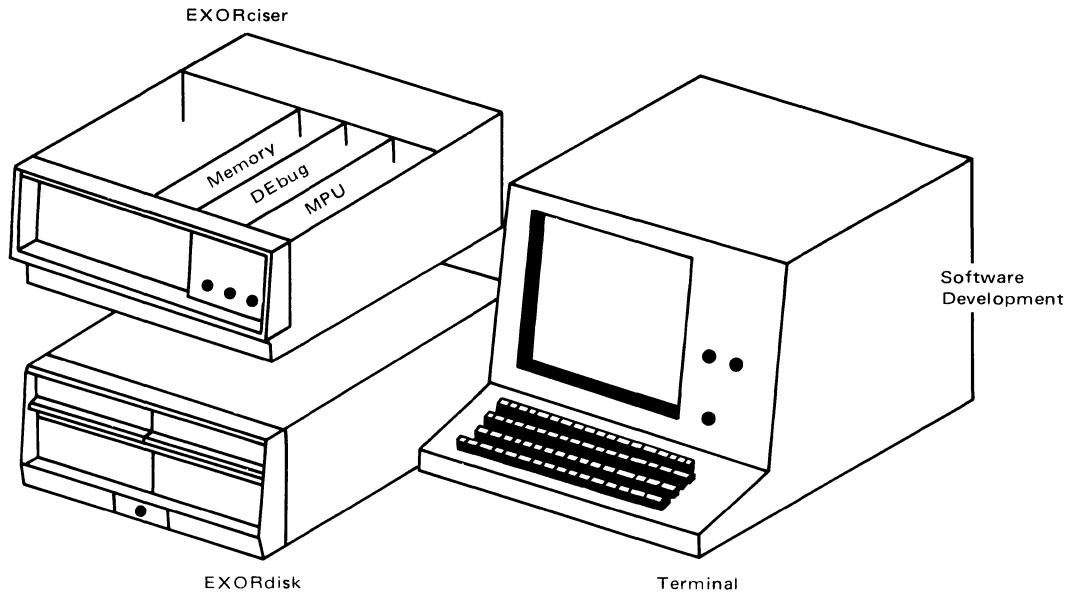


FIGURE 4-2. EXORciser, the Development Tool

- h. Builds the production hardware of his system and has his ROM's made. The USE\* is used in testing and debugging the production systems.
  - i. Combines the production hardware with the system software and makes the final adjustments to his system. He again may use the EXORciser in evaluating his system by means of the USE\*.
  - j. Releases his system to production.
  - k. Analyzes problems in production hardware with the EXORciser and USE\*.
- \* The User System Evaluator is an optional module with an interconnecting cable (and buffers) which plugs into the M6809 socket of the user prototype or production unit. It connects the two systems together in such a way that all the debug capabilities of the EXORciser are usable in the user hardware. (See the USE User's Guide for details.)

#### 4.5 EXORciser CONFIGURATION

The EXORciser configuration is described in detail in paragraph 1.5. Memory parity is described in paragraph 1.5.1. Dual map concepts are discussed in paragraph 1.5.2.

#### 4.6 SYSTEM ADDRESS SELECTION

The address selection of the various modules will depend on whether the system is being used for software development or to emulate the target system. Included in the address selection is the determination of which map the module will respond in. The user assigns a module to one of the two maps by installing either the VUA or VXA addressing jumper that is found on all EXORciser modules.

During software development (edits, compilations, assemblies, etc.), the user will probably want as much continuous RAM as possible, starting at address 0000. This RAM must also be in the same map as EXbug so that the software development programs can communicate with the system terminal. This requires that the RAM, EXORDisk interface, and Printer interface be configured for the VXA, and the DEbug Module be configured for the Dual Map mode, or that the RAM, EXORDisk interface, and Printer interface be configured for VUA, and the DEbug Module be configured for the Single Map mode. These same requirements also apply for any programs that use the system terminal or EXbug routines.

During target system emulation, the module addresses will be selected as required for the target system. The target system may be emulated in the User map of the Dual Map mode or, if it does not require any addresses equal to or greater than F000, it may be emulated in the Single Map mode. In the Single Map mode, the modules should be configured to respond to VUA.

Refer to the applicable User's Guide for instructions on setting the address and map of the various modules. To avoid conflicts with EXbug, Modules in the Executive map should be addressed only at values less than \$F000.

#### 4.7 SECOND LEVEL INTERRUPT

The second level interrupt feature of the EXORciser is described in detail in paragraph 1.5.3.

#### 4.8 MEMORY ASSIGNMENTS

The DEbug Module provides the EXORciser with the capability of addressing two separate 64K blocks of memory. These two blocks of memory are referred to as the Dual Memory map. One of these, the Executive map, contains EXbug (if configured for VXA), its peripheral devices and RAM, the EXORDisk ROM and I/O devices, and the Printer I/O device. The other, the User map, is completely available to the user for emulation of his target system. This gives the user complete freedom in assigning addresses to his memory and I/O devices without worrying about addressing conflicts with the system monitor and I/O devices, yet EXbug provides the user with full debug capabilities in the User map. Optionally, in the Single Map mode, the DEbug Module can merge the two maps. In this mode, all addresses less than \$F000 come from the User map.

In the Dual Map mode, all of the EXbug debug commands are available in either map. The EXbug USER and EXEC commands control which map will be accessed by the debug commands. The command USER causes the EXbug debug commands to operate in the User map. In this mode, EXbug prompt is \*. The command EXEC causes the EXbug debug commands to operate in the Executive map. EXbug prompt is \*E in this mode. On power-up, EXbug comes up in the EXEC mode.

In the Single Map mode, the EXEC and USER commands are not usually required, since the maps have been merged. However, if the Halt-on-Address or Scope Sync function is to be used at an address less than \$F000, the USER command must be entered so that the address compare circuitry will detect the appropriate map.

Executive map/User map interface is described in detail in Appendix G. It is suggested that the user read Appendix G, and then prepare the User Map as follows:

- a. Construct a basic memory map of the User System.
- b. Assign the memory location for ROM's or PROM's at the top of memory. The top of memory of the ROM's must appear to the MPU as address FFFF in the planned decoding scheme. (Some address lines will not be used, and the ROM, therefore, will respond to more than one range of addresses.)
- c. Assign the memory addresses for RAM's. It is recommended that the RAM's be placed below address 100 in memory, to realize the advantage of the direct addressing mode to save memory.
- d. If the User System is using Peripheral Interface Adapters (PIA's), assign four addresses for each PIA, as described in the M6809 Manuals.
- e. If the User System is using Asynchronous Communication Interface Adapters (ACIA's), assign two addresses for each ACIA, as described in the M6809 Manuals.

#### 4.9 EXORciser CONFIGURATION FOR SYSTEM EMULATION

This paragraph discusses preparing the EXORciser to emulate (functionally represent) the User System. Refer to the optional module User's Guides, as required, for details. Prepare the EXORciser hardware as follows:

- a. Install the EXORciser modules in the EXORciser card slots.

- b. Determine whether the EXORciser clock or an external clock is to be used in the User System. Install jumpers on the MPU Module accordingly. (If using the User System Evaluator, see applicable User's Guide.)
- c. If you are using an external clock, determine the clock frequency to be used in the User System.
- d. Connect the external clock to the MPU Module.
- e. Connect the EXORciser to the user process or peripheral device, using the I/O Module for parallel interface, or the ACIA Module for serial interface.
- f. Construct any required special circuitry for interface.
- g. Set the base memory addresses on the EXORciser memory and peripheral interface modules by means of the address switches, as described in the module User's Guides.
- h. Assign memory map address enable jumpers (VUA, VXA, or PAGE ENABLE) on each module.

#### 4.10 TESTING PROTOTYPE OR PRODUCTION SYSTEMS

Once the designer has emulated his system in the EXORciser chassis and it is operating properly, he can begin construction of a prototype as a next step in the development. By his emulation, he now knows exactly how much memory he needs, how many output ports or lines will be required, and even what his clock circuit and decoding scheme must be. This information allows him to design a prototype that will be reasonably close to the final production units. He has not eliminated the need for a prototype altogether, but probably has bypassed several iterations, at least.

When construction is completed, he must test his prototype and determine whether it performs as well as the emulation system did. For this purpose, the EXORciser is augmented by the addition of the User System Evaluator (USE). This subsystem consists of a USE processor module connected by cables to the MPU Module, and a buffer and cable assembly.

The USE-EXORciser can be used with the Motorola M6809 microprocessor system. The purpose of this arrangement is to provide the same debug capabilities in the user system as previously used in the emulation. The interconnection of the two systems in this way creates one bigger system which operates in real time with one MPU. With this arrangement, it is now possible to operate and test the system with all or part of the I/O, or memory, in either system. The memory can be RAM or ROM (or PROM), and the I/O can be the original emulated version using EXORciser modules, or can be the newly constructed circuits on the prototype.

Since it is better to take little steps, rather than one big plunge, the procedure for the development of a typical system might be as follows:

- a. One or more of the I/O chips are installed, and the associated external peripheral is tested by using the EXbug memory display/change routine, just as was done in the original emulation (see par. 4-3).

- b. After all of the I/O circuits have been activated and are working correctly (assuming that the system previously shown in Figure 4-2 is being prototyped), resume testing by loading the program that was used in the emulation into RAM module.

NOTE

Many engineers may choose to assemble their I/O circuitry on the prototype board, or boards, and get it working with their peripherals before the programming is finalized. In this case, it is desirable to be able to edit and re-assemble the program without dismantling the hardware setup. This is possible by installing one or more of the dynamic RAM memory modules.

- c. If using a tape terminal and the Resident Editor/Assembler programs, a minimum of 8K of RAM will be required with its address switches set for 0 and 1 (so as to have memory from 0000 to 1FFF). The Editor/Assembler reside in this range with 744 bytes left over for the edit buffer or assembler symbol table.
- d. If the EXORdisk Floppy Disk System is chosen with the standard Editor and Assembler programs, a minimum of 16K of RAM will be required, with its address switch set for 0 to provide memory from 0000 to 3FFF.

NOTE

With either of these arrangements, the editing and assembling process can be carried on in the EXORciser without disturbing the hardware or the program in memory. The USE feature, whereby the memory in the EXORciser has priority over any memory (or I/O) in the user system, allows this to work.

- e. Once the user program has been edited and re-assembled until it operates properly, it can be copied into EROM, or bi-polar PROM, by means of the PROM Programmer Module (MEX68PP3). The PROM can then be installed in the appropriate socket of the user prototype for extended testing.

NOTE

Any of the features of the EXbug program can be used at any time. For example, the Trace and Breakpoint functions are fully usable in either system. Trace works with PROM (or ROM) and, although breakpoints will not work in ROM, the Stop-on-Address will, and does essentially the same job. These allow testing in real time (i.e., no wait states are required, and the instructions are executed at the user clock rate, including all I/O).

When the program has been put into PROM's, the EXORciser will start up in the map specified by the switch settings on the DBUG. EXbug Restart vectors will always be used when in Single Map mode. If in Dual Map mode, and the Restart switch is set to its USER position, the EXORciser will use the USER map for its RESTART vector and subsequent execution. This feature allows the user to fully evaluate the power startup portion of his final system design.

## 4.11 PRECAUTIONS WHEN USING THE USER SYSTEM EVALUATOR

The User System Evaluator is designed to interface with the user M6809 system, but certain non-obvious precautions must always be taken. When the USE module is first installed, many users have difficulty getting their EXORciser to run properly. Refer to the USE User's Guide.

## 4.12 SYSTEM EVALUATION AND DEBUG PROCEDURES

Once a program has been written, entered, and assembled or compiled, it must be tested to determine if there are any errors or bugs in it. The program must also be tested with a prototype version of the system hardware to determine if the complete system functions as required. The EXORciser, with the EXbug program, provides a ready tool for system testing. For preliminary system testing, the various optional EXORciser modules, such as the PIA and ACIA modules, can be used to emulate the I/O functions of the final system. The EXORciser RAM modules can be configured, as required, to contain the program under test. Paragraph 4.9 discusses how to configure the EXORciser for system testing. After the system is properly configured, load the program.

### 4.12.1 Memory Loader

The program may be loaded to memory from tape, using the terminal tape reader and EXbug LOAD command. If the program is on an MDOS diskette, the MDOS LOAD command can be used to bring it into memory. At this point, the program can be run without any EXbug debug features active. This will give an indication if the program operates correctly. The program can be started using the ;G or addr;G command.

### 4.12.2 Abort Function

If the program does not operate properly, the ABORT button should return control to EXbug. The abort will give a register printout of the processor state when the abort occurred. This printout can be useful in further debugging by indicating a likely place to start debugging. If the program were run in the Single Map mode or in the Executive map, it might have destroyed EXbug NMI vector, in which case the abort will not function. Also, execution of an invalid op code might put the processor in a state where it will not respond to the abort NMI. If either of these two conditions occurs, the RESTART button will have to be used to return control to EXbug. In the Dual Map mode, the restart switch S3 on the DEbug Module must be in the EXBG position for the restart to start EXbug.

### 4.12.3 Default Debug Offset

Before continuing further with debugging, the default debug offset, or Q value, should be specified if a relocatable program is under test. Specifying the main base address of the program as the Q value speeds debugging by permitting direct entry of all addresses relative to it. EXbug automatically adds the Q value to each single parameter address value entered. The Q value can be overridden by entering two parameters separated by a comma as an address parameter. In this case, the two parameters are added together to determine the absolute address. Refer to par. 3.6, EXbug commands, for further information about the default debug offset. All addresses displayed by EXbug are absolute values. The Memory Change function can be used to determine the corresponding relative value. To determine the relative address, enter the absolute address, a comma, a minus

sign, the base address, and then a slash. EXbug will then display the contents of the relative address used as an absolute address. Entering another slash will cause EXbug to display the relative address and the memory contents again. A carriage return ends the Memory Change function. Figure 4-3 shows an example of this use of the Memory Change function.

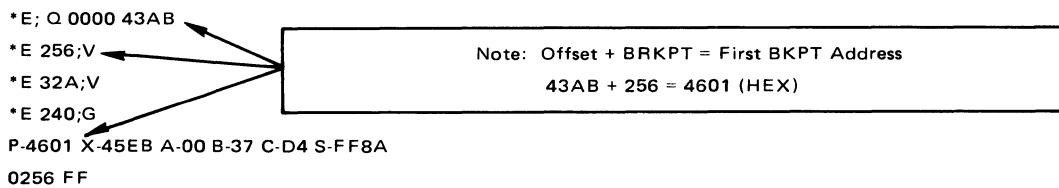


FIGURE 4-3. Using Memory Change to Calculate a Relocatable Address

#### 4.12.4 Memory Change Function

Since the I/O devices in an M6809 system are in the memory map, the Memory Change function can be used to read and change their contents. Certain features of the memory change command facilitate its use to read and modify I/O devices. The slash (/) memory change terminator updates the open location to the new value entered, if one was, and then re-opens the same location. This capability is handy in toggling a bit in a parallel output device, or repetitively viewing an input device. If a location does not change correctly, the Memory Change function is not automatically exited. Instead, an error message, consisting of a question mark, bell character, and the contents of the location, is sent to the system terminal. After the error message, the Memory Change function proceeds according to the command used to close the last location. This feature is useful when writing to write only registers or PIA control registers. The EXbug Memory Change function reads a location only once when its contents are displayed. Also, a location is read once after it is written to verify that it changed properly. When a location is written, only one store instruction is used.

#### 4.12.5 Breakpoint, Trace, and Halt-on-Address/Scope Sync Functions

EXbug provides three different methods of controlled program execution to assist in locating hardware and software problems. The methods are:

- Breakpoints
- Halt-on-Address/Scope Sync
- Trace

Each method has its advantages and restrictions. Combined, they provide very powerful and flexible techniques for testing and debugging M6809 systems. There is one restriction that is common to all three controlled program execution methods, except for the scope sync. Breakpoints, trace, and halt-on-address all use an interrupt to stop program execution. Since an interrupt pushes the contents of the processor register on the stack, the stack pointer must be pointing at a valid stack area when the interrupt occurs. Therefore, if the

stack pointer is not pointing at RAM when the interrupt occurs, the register contents will be lost. If the stack pointer is pointing to a data area in RAM when the interrupt occurs, the data will be overwritten by the register contents.

**4.12.5.1 Breakpoints.** Breakpoints are probably the most useful of the three techniques. A maximum of eight breakpoints can be active at any given time. Therefore, they can be used to check program flow. Breakpoints can be set at various locations throughout the program; then, when program execution reaches the breakpoint, execution is stopped and the processor registers are displayed. Breakpoints are also useful in testing the execution of program loops. This is provided by the n;P command, which continues execution at a breakpoint but does not stop and display the registers at that breakpoint until the nth time it occurs. Except for the use of the n;P in loop testing, the program will execute in real time while breakpoints are active. During the n;P operation, the breakpoint remains in the program and gives control to EXbug each time it is encountered. EXbug returns control to the program as long as the n value is non-zero. This operation will slow loop execution. The restrictions on the use of breakpoints are that they can only be used in programs running in RAM, and that they must only be set on the first byte of an instruction. Both of these restrictions result because EXbug uses the SWI instruction for breakpoints.

**4.12.5.2 Trace.** The trace feature comes in two options: trace a given number of instructions or trace to a given address. Both are useful in watching the processor registers during program execution. Care must be exercised while using trace, however, since it is easy to quickly generate more register printout than one can reasonably digest. Since the trace feature uses the NMI, programs in ROM as well as RAM can be traced. The restriction on trace is that the program execution is not real time. EXbug interrupts the program after each instruction so that it can provide a register printout.

**4.12.5.3 Halt-on-Address.** The Halt-on-Address function is useful in finding how a location is being changed when it is not expected to be. Halt-on-Address generates an interrupt when the specified address appears on the address bus. It is not necessarily associated with the execution of a specific instruction, as are the two previous techniques. Since it is a function of address, the Halt-on-Address can also be used as a single breakpoint for a program in ROM. The restrictions on the Halt-on-Address are that the processor is not interrupted until after the completion of the instruction which accessed the specified location, and that unexpected halts may occur due to the operation of certain instructions. The first restriction causes the program counter to indicate the next instruction to be executed after the instruction that accessed the specified location. In most cases, this will not cause a problem. However, if the instruction which caused the halt also caused a change in program flow (such as a branch, JMP, JSR, RTS, or RTI), it may be difficult to determine which instruction caused the halt. Unexpected halts may be caused by certain instructions. For example, all single-byte instructions cause the next address to be read while the instruction is being executed. Therefore, if a halt is set on an instruction following a single-byte instruction, the halt will occur twice -- once when the single-byte instruction is executed and, again, when the instruction following it is executed. One instance of this type of operation that is not always obvious is when a halt is set on an instruction where the preceding location contains an RTS. The halt will occur when the RTS is executed, but the associated register printout will show no relationship to where the halt was set. A cycle-by-cycle description of all M6809 instructions is contained in the M6809 Data Sheets.

4.12.5.4 Scope Sync. Associated with the Halt-on-Address function is a Scope Sync function, which is enabled and disabled by the same EXbug commands as the Halt-on-Address. The selection between the two functions is made by switch S2 on the DEbug Module. When the switch is in the Halt-on-Address position, an NMI is generated by the address match. In the Scope Sync position, a pulse is generated at the scope trigger point on the DEbug Module by the address match. The Scope Sync feature is used to trigger an oscilloscope to monitor the waveforms associated with a particular point in the program. The restriction on the Scope Sync is that unexpected trigger pulses may result. This happens for the same reason as the unexpected halts -- the specified address appearing on the bus because of a preceding single-byte instruction.

#### 4.12.6 Error Correction

Once program errors, or bugs, have been found using the above procedures, the next step is to correct the bug and continue testing. There are two ways of correcting bugs. If the bug is relatively minor, such as loading the wrong register or the wrong value, it may be patched -- that is, corrected in memory. A written record should be made of the patch so that the program source may be corrected at a later time. It is also a good idea to dump the patched program to disk or tape so that memory can be restored. After several patches have been collected, the program source should be edited and the bugs corrected. Then the program can be recompiled or assembled, and testing can continue with the new version of the program. If the program bug is major, such as an error in program organization, it may be necessary to go directly to the edit, compile, assemble procedure.

The point at which a bug becomes too large to patch depends on the time required to edit, compile, assemble the program. If the program can be edited and compiled or assembled and a new object program obtained quickly, then lengthy patches may not be appropriate.

### 4.13 SOFTWARE DEVELOPMENT USING THE EXORciser

With the addition of the EXORDisk and Microsystems Printer, the EXORciser becomes a very powerful software development tool. Software is developed as follows.

Program Requirements - The first step in software development is to define the program requirements. This specification should be as complete and detailed as possible. Without this road map of what is required of the software, the design of it can easily become misdirected into areas that are really unimportant.

Language Selection - Select the language the program will be written in. The choice is between assembly language and the higher level languages. Along with resident absolute and macro/relocatable assemblers, Motorola currently supports the following resident higher level languages for the M6809: FORTRAN, BASIC, MPL, and PASCAL. The decision of which language to use depends on the application, the proposed production volume of the final system, and what languages the user has experience with. Typically, as the production volume of a product is increased, the reduced memory requirements of assembly language over higher level languages makes assembly language a better choice. However, when the production volume will be low, the reduced development time for higher level languages, as compared to assembly language, will make them the better choice. System response time requirements may also play a part in determining what language is used, since higher level languages execute more slowly than

assembly language. Some higher level languages overcome this by permitting portions of the program to be written in assembly language. The user should also give preference to languages he has had experience with, in order to avoid a possibly lengthy learning period which would affect the product schedule.

Program Design - After a language has been selected and program requirements defined, design the program. This step is very important. The completed design of the program is equivalent to the schematic for the hardware. This design may be done in flow chart, metacode, a higher level language, or some other technique with which the designer is familiar. The language selected will have a bearing on the design. A more detailed design is required if the program is to be written in assembly language instead of a higher level language. This is required so that memory and the processor registers are properly allocated. These are tasks that a higher level language will take care of for the programmer. Some higher level languages such as MPL are structured so that the program could be designed using the higher level language. In these cases, the design and coding, or programming, would occur concurrently.

Writing the Program - Once the design is completed, the next step is to write the program. As the program is being written, it can be entered onto disk or tape, using the resident editor. This is referred to as the source. If the program is written in modular blocks, they may be compiled or assembled and tested as they are completed. Otherwise, the complete program will have to be entered before it can be compiled/assembled and tested.

Testing - The procedure described in the preceding section may be used for testing the program. However, if the equivalent I/O devices for the final system are not duplicated in the EXORciser, the routines that handle these devices and the corresponding I/O cannot be tested easily. If these routines are to be tested at all without the corresponding hardware, then software routines must be written to simulate the I/O. Generally, though, routines that do not require I/O can be tested without much difficulty once any required parameters are set up. As testing continues and program errors are found, they may be corrected by patches, as described in the preceding section. When several patches have been made to the program, it is wise to incorporate the corrections in the source by editing it. After the corrections have been made, the program can be compiled/assembled again, and testing can continue.



## CHAPTER 5

### THEORY OF OPERATION

#### 5.1 INTRODUCTION

This chapter provides a block diagram description of the EXORciser. As a system development tool, the EXORciser may be configured in a variety of applications and with a variety of options. This chapter, rather than discussing each possible configuration, discusses the basic EXORciser, which comes equipped with an MPU Module, a DEbug Module, 32K of RAM Modules, and the power supply.

The basic EXORciser and its optional modules provide the user with a system development tool for the M6809 Microcomputer Family of Parts. The user, through his configuration of the optional modules in the basic EXORciser unit, has the capability of emulating (functionally creating) a hardware prototype of his system.

#### 5.2 BASIC EXORciser BLOCK DIAGRAM DESCRIPTION

The basic EXORciser, as illustrated in Figure 5-1, consists of the MPU Module, DEbug Module, power supply, chassis, EXORciser bus, and the EXbug Firmware. Each of these modules is built around the M6809 Microcomputer Family of Parts, MC68B09 Microprocessing Unit (MPU), MC68B21 Peripheral Interface Adapter (PIA), MC68B10 Random Access Memory (RAM), and MC68B50 Asynchronous Communications Interface Adapter (ACIA).

Through its system debug and program control features, the EXORciser EXbug Firmware minimizes the time required to develop a user system. The EXbug Firmware provides the EXORciser with the capability to:

- Display the contents of the MPU registers at any time.
- Step through the user program one instruction at a time.
- Trace through a user program to locate problem areas.
- Stop the program on a selected program step.
- Abort from the user program and return to the EXbug control program on command.
- Re-initialize the EXORciser on command.

The user communicates with the EXORciser in one of two ways:

- Through an RS-232C or TTY data terminal.
- Through the EXORciser front panel controls and indicators.

The data terminal permits the user to communicate directly with the EXbug Firmware. The EXORciser front panel permits the user to apply power to the EXORciser to abort (exit) the EXORciser from a routine, and to initialize and reset the EXORciser.

The MPU Module incorporates the MC68B09 Microprocessing Unit (MPU) and the system clock. This module provides the MPU and the clock signals for both the EXORciser and the user prototype system. The MC68B09 Microprocessing Unit is an

8-bit parallel processing unit capable of addressing 64K bytes of memory. In addition, the MPU addresses its input and output devices as memory. The MPU also provides the EXORciser with 59 variable length instructions and the capability of responding to real time interrupt signals.

The MPU Module controls the flow of commands, data, and addresses on the EXORciser bus. During an MPU memory read or write operation, the MPU Module controls the transfer of command, status, addresses, and data to the selected module by controlling the EXORciser bus.

The DEbug Module provides the EXORciser with the capability to evaluate and debug the user prototype hardware and software in an actual application. The EXbug Firmware, contained in ROM, provides the EXORciser with program control capabilities. The RAM is used as a scratchpad memory for the EXbug Firmware. The EXbug Firmware enables the EXORciser to:

- Load data into the EXORciser.
- Verify that the data in the EXORciser is valid.
- Search a tape for a specific file.
- Print the contents of the memory.
- Punch (or record) the contents of the memory.
- Perform the MAID (Motorola Active Interface Debug) functions.

The MAID function enables the EXORciser to:

- Examine and, if required, change the contents in a memory location.
- Examine and, if required, change the contents of the MPU registers.
- Calculate the offset in the relative addressing mode.
- Insert, display, and remove breakpoints in the user program.
- Freerun or trace through a user program under MAID control.
- Search memory for a specific bit pattern.
- Perform decimal-octal-hexadecimal conversions.
- Stop the EXORciser on a selected memory address in the user program.
- Provide an oscilloscope trigger pulse at a selected memory address.

The DEbug Module also incorporates the level converter circuits required to interface the EXORciser with a TTY or RS-232C data terminal, and also provides the EXORciser with eight standard baud rates (110, 150, 300, 600, 1200, 2400, 4800, and 9600). This module also interfaces the EXORciser with a TTY or RS-232C compatible data terminal.

The Power Supply provides the EXORciser with the +5, +12, and -12 Vdc power sources that are required by the EXORciser and related modules. This power supply will support a full rack of modules.

The chassis is capable of holding 14 plug-in modules. These 14 modules connect directly into the EXORciser bus. The Power Supply is not a plug-in module, and is mounted directly to the EXORciser chassis.

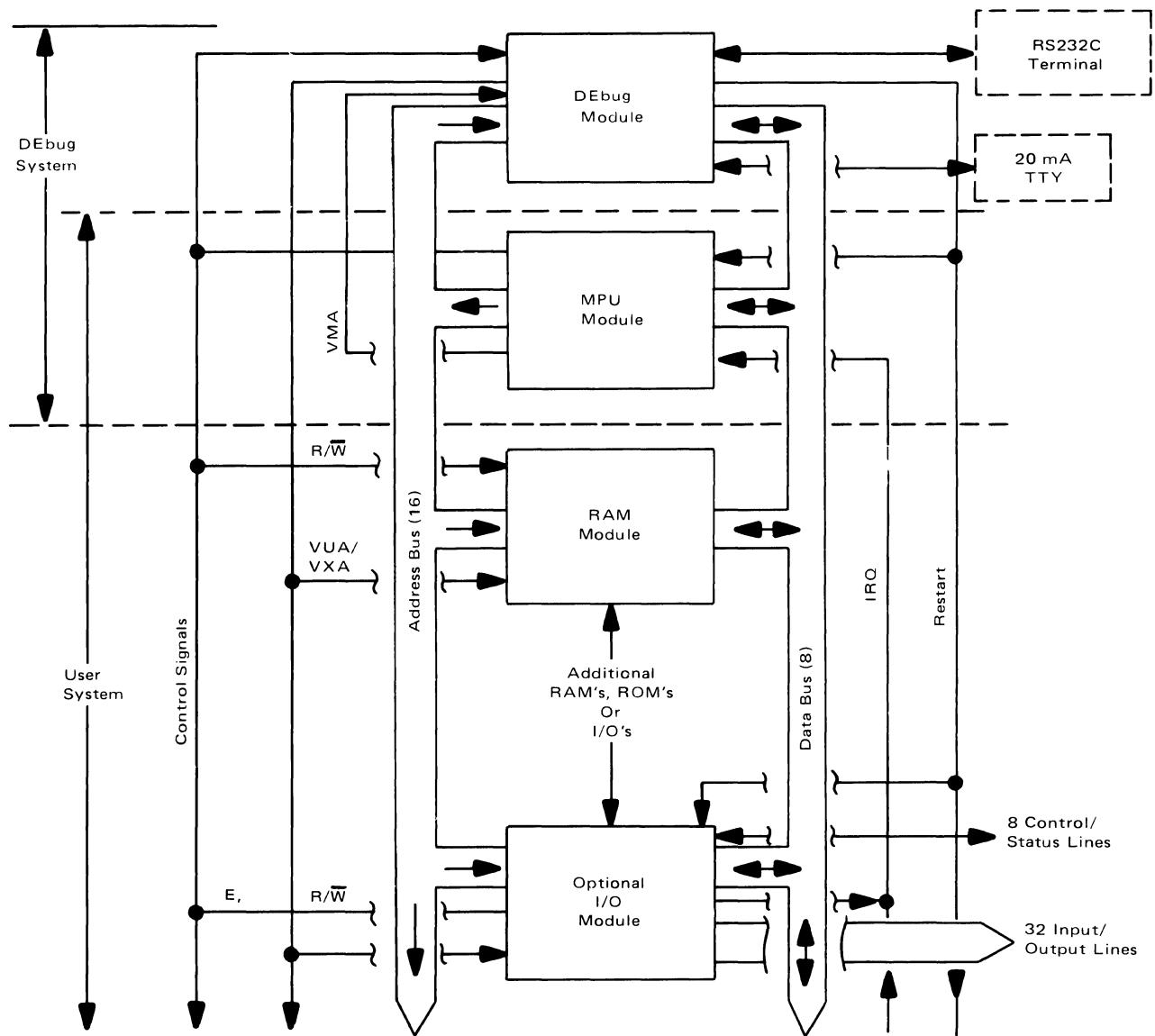


FIGURE 5-1. EXORciser Simplified Block Diagram



APPENDIX A  
EXORciser BUS DESCRIPTION AND SPECIFICATIONS

## INTRODUCTION

The EXORciser Development System incorporates a multilayer motherboard to interconnect the address, data, and control buses for up to 14 separate system modules. The motherboard is made with an embedded groundplane to provide the high noise immunity requirements of a high speed microprocessor development system. This appendix identifies all of the motherboard interconnections, and describes the function of each interconnect signal. In addition, complete timing specifications are provided for the bus interconnections.

## DESCRIPTION OF BUS SIGNALS AND PIN ASSIGNMENTS

Table 1 summarizes the pin assignments for the EXORciser motherboard, while Table 2 lists the signal mnemonic, name, and functional description. Eight motherboard interconnection lines have been specifically reserved for user applications. Other unused EXORciser motherboard interconnects are reserved for future EXORciser expansions.

TABLE 1. EXORciser Bus Connections

PIN NUMBER (COMPONENT SIDE)	FUNCTION	PIN NUMBER (CIRCUIT SIDE)	FUNCTION
A,B,C	+5VDC	1,2,3	+5VDC
D	IRQ	4	HALT
E	NMI	5	RESET
F	VMA	6	R/W
H	Not used	7	Q
J	E	8	GND
K	GND	9	GND
L	MEMCLK	10	VUA
M	-12VDC	11	-12VDC
N	BUSREQ	12	REF REQ
P	BA	13	REF GNT
R	MNRDY	14	DEBUG
S	LIC*	15	BUSGNT
T	+12VDC	16	+12VDC
U	STANDBY	17	STANDBY
V	PWR FAIL	18	CLK
W	PARITY-ERR	19	VXA
X,Y,Z	GND	20,21,22	GND

TABLE 1. EXORciser Bus Connections (cont'd)

PIN NUMBER (COMPONENT SIDE)	FUNCTION	PIN NUMBER (CIRCUIT SIDE)	FUNCTION
<u>A</u>	<u>FIRQ</u>	23	BS
<u>B</u>	GND(REF)	24	GND(REF)
<u>C,D,E,F</u>	Reserved for Bus Expansion	25,26,27,28	Reserved for Bus Expansion
<u>H</u>	<u>D3</u>	29	<u>D1</u>
<u>J</u>	<u>D7</u>	30	<u>D5</u>
<u>K</u>	<u>D2</u>	31	<u>D0</u>
<u>L</u>	<u>D6</u>	32	<u>D4</u>
<u>M</u>	A14	33	A15
<u>N</u>	A13	34	A12
<u>P</u>	A10	35	A11
<u>R</u>	A9	36	A8
<u>S</u>	A6	37	A7
<u>T</u>	A5	38	A4
<u>U</u>	A2	39	A3
<u>V</u>	A1	40	A0
<u>W,X,Y</u>	GND	41,42,43	GND

\*Denotes signal not used with M6809 EXORciser. For M6809E EXORcisers, LIC signal is used.

TABLE 2. EXORciser Bus Signals

PIN NUMBER	SIGNAL MNEMONICS	SIGNAL NAME AND DESCRIPTION
A,B,C	+5VDC	+5 Vdc Power - Used by the system module logic circuits and available to the user for prototype module requirements. (15 Amps max.)
D	<u>IRQ</u>	INTERRUPT REQUEST - An active low signal used to request generation of an MPU interrupt sequence. The MPU will wait until it completes the instruction being executed before it recognizes the request. At that time, if the interrupt mask bit in the MPU condition code register is not set, the MPU will begin executing the interrupt sequence.
E	<u>NMI</u>	NON-MASKABLE INTERRUPT - A low going, edge sensitive signal used to request generation of an MPU non-maskable interrupt sequence. The MPU will wait until it completes the instruction being executed before it recognizes the request. At that time, regardless of the logic state of the interrupt mask bit in the MPU condition code register, the MPU will begin executing the non-maskable interrupt.

TABLE 2. EXORciser Bus Signals (cont'd)

PIN NUMBER	SIGNAL MNEMONICS	SIGNAL NAME AND DESCRIPTION															
F	VMA	VALID MEMORY ADDRESS - A high level, TTL compatible signal produced by the MPU Module and used to indicate to the DEbug Module that a valid memory address is present on the address bus.															
H	GND	GROUND.															
J	E	E - A bi-phase clock signal generated by the MPU Module. Data from the MPU is guaranteed good with the falling edge of E. This signal is held high during MNRDY.															
K	GND	GROUND															
L	MEMCLK	MEMORY CLOCK - A TTL level clock signal, in phase with E, used to refresh all dynamic memory modules within the system. This signal is stretched high during 2 MHz use. It is held high during MNRDY (pin R low).															
M	-12VDC	-12 Vdc Power - Used by the system module logic circuits and available to the user for prototype module requirements. (1.5 Amps max.)															
N	<u>BUSREQ</u>	BUS REQUEST - An active low signal used to request access to the system bus. A low on this line will cause the MPU Module to three-state (off or high impedance state) the data, address, and R/W lines. A BUSGNT signal (pin 15) will also be generated at this time.															
P	BA	BUS AVAILABLE - A normally low level signal generated by the system MPU. This signal along with the BS signal (pin 23) indicates the MPU state. <table> <thead> <tr> <th><u>BA</u></th> <th><u>BS</u></th> <th><u>MPU STATE</u></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>Normal (Running)</td> </tr> <tr> <td>0</td> <td>1</td> <td>Interrupt Acknowledge</td> </tr> <tr> <td>1</td> <td>0</td> <td>Sync Acknowledge</td> </tr> <tr> <td>1</td> <td>1</td> <td>Halt or Bus Grant</td> </tr> </tbody> </table>	<u>BA</u>	<u>BS</u>	<u>MPU STATE</u>	0	0	Normal (Running)	0	1	Interrupt Acknowledge	1	0	Sync Acknowledge	1	1	Halt or Bus Grant
<u>BA</u>	<u>BS</u>	<u>MPU STATE</u>															
0	0	Normal (Running)															
0	1	Interrupt Acknowledge															
1	0	Sync Acknowledge															
1	1	Halt or Bus Grant															
R	<u>MNRDY</u>	MEMORY NOT READY - A signal generated by the user that permits the EXORciser to work with slow memory modules. When this signal is low (set-up time before the falling edge of E), the clocks will be stretched with E high and Q low. Furthermore, the memory ready function actually changes the system E, MEMCLK (pin L), and CLK (pin 18) signals. Devices which require a real-time clock must use a different clock source.															
S		Not used (M6809 only).															
S	LIC	LAST INSTRUCTION CYCLE (M6809E only). - An active high signal produced only by the MC68B09E MPU during the last cycle of each instruction. This line will be a high during the halt and sync states.															
T	+12VDC	+12 Vdc Power - Available to the user for prototype module requirements. (2.5 Amps max.)															

TABLE 2. EXORciser Bus Signals (cont'd)

PIN NUMBER	SIGNAL MNEMONICS	SIGNAL NAME AND DESCRIPTION
U	STANDBY	STANDBY Power - This line is reserved for use with battery backup memory modules. If battery backup is not required, the STANDBY line is not used.
V	<u>PWR FAIL</u>	POWER FAIL - This signal line is reserved for use with memory modules requiring battery backup. When used, this low level signal would disable the protected memory module. (This feature is not supplied as part of the EXORciser.)
W	<u>PARITY-ERR</u>	PARITY ERROR - This signal line is normally held high by the DEbug Module. If a memory module that incorporates a parity check circuit is used within the EXORciser, and a parity error is detected, this signal will be forced low for one clock cycle.
X,Y,Z	GND	GROUND
<u>A</u>	<u>FIRQ</u>	FAST INTERRUPT REQUEST - An active low signal used to request the generation of an MPU fast interrupt sequence. The MPU will wait until the instruction being executed is completed before recognizing the request. At that time, if the interrupt mask bit in the MPU condition code register is not set, the MPU will begin the interrupt sequence. This sequence is fast in the sense that it only stacks the return address and condition codes.
<u>B</u>	GND(REF)	GROUND (REFERENCE) - Available to the user for prototype modules that require an isolated ground. This ground line is not connected to the normal EXORciser ground connection.
<u>C,D,E,F</u>		USER DEFINED - These signal lines, along with their counterpart pin numbers (25,26,27,28) are reserved by Motorola for possible expansion of the data bus to 16 bits. Since compatibility of 16-bit data bus modules with existing modules is unlikely, these eight lines may be used for custom modules.
<u>H</u>	D3	DATA bus (bit 3) - One of 8 bi-directional data lines used to provide a two-way data transfer between the MPU Module and all other plug-in modules within the system. The data bus drivers on the other modules are in their off or high impedance state except when selected during a memory read or write operation.
<u>J</u>	<u>D7</u>	DATA bus (bit 7) - Same as <u>D3</u> on Pin <u>H</u> .
<u>K</u>	<u>D2</u>	DATA bus (bit 2) - Same as <u>D3</u> on Pin <u>H</u> .
<u>L</u>	<u>D6</u>	DATA bus (bit 6) - Same as <u>D3</u> on Pin <u>H</u> .
M	A14	ADDRESS bus (bit 14) - One of 16 address lines from the MPU Module that permits the MPU to select any addressable memory location within the EXORciser.
<u>N</u>	A13	ADDRESS bus (bit 13) - Same as A14 on Pin <u>M</u> .
<u>P</u>	A10	ADDRESS bus (bit 10) - Same as A14 on Pin <u>M</u> .

TABLE 2. EXORciser Bus Signals (cont'd)

PIN NUMBER	SIGNAL MNEMONICS	SIGNAL NAME AND DESCRIPTION
<u>R</u>	A9	ADDRESS bus (bit 9) - Same as A14 on Pin <u>M</u> .
<u>S</u>	A6	ADDRESS bus (bit 6) - Same as A14 on Pin <u>M</u> .
<u>T</u>	A5	ADDRESS bus (bit 5) - Same as A14 on Pin <u>M</u> .
<u>U</u>	A2	ADDRESS bus (bit 2) - Same as A14 on Pin <u>M</u> .
<u>V</u>	A1	ADDRESS bus (bit 1) - Same as A14 on Pin <u>M</u> .
<u>W,X,Y</u>	GND	GROUND
1,2,3	+5VDC	+5 Vdc Power - Used by the system module logic circuits and available to the user for prototype module requirements. (15 Amps Total max.)
4	<u>HALT</u>	HALT - A normally <u>high</u> level signal used to halt the MPU. A low level on the HALT input causes the MPU to halt at the end of the present instruction, and <u>remain</u> halted indefinitely without loss of data, until the HALT pin is driven high.
5	<u>RESET</u>	RESET - An active low signal used to reset the MPU as well as other peripheral devices and system modules. This signal also restarts the EXORciser when power is initially applied. Depressing the RESTART pushbutton switch located on the front panel of the EXORciser while the system is operating will generate a RESET signal and cause the MPU Module to execute the EXbug restart routine or the restart routine indicated by the user.
6	R/W	READ/WRITE - This signal is generated by the MPU Module, and indicates to the other modules contained within the system that the MPU is performing a memory read (high) or write (low) operation. The normal standby state of this signal is read (high).
7	Q	Q - A quadrature clock signal which leads E. Addresses from the MPU will be guaranteed good with the leading edge of Q. This signal is held low during <u>MNRDY</u> (pin R low).
8,9	GND	GROUND
10	VUA	VALID USER ADDRESS - This signal is produced by the DDebug Module. When high, this signal indicates that the address on the address bus is valid and the MPU Module is not addressing the EXbug program.
11	-12VDC	-12 Vdc Power - Available to the user for prototype module requirements. (1.5 Amps max.)
12	<u>REF REQ</u>	REFRESH REQUEST - When low, this input signal to the MPU Module initiates a memory refresh cycle of the dynamic memory modules. The memory clock signal will continue to run to allow memory refreshing.

TABLE 2. EXORciser Bus Signals (cont'd)

PIN NUMBER	SIGNAL MNEMONICS	SIGNAL NAME AND DESCRIPTION
13	REF GRANT	REFRESH GRANT - When high, this output signal from the MPU Module instructs the dynamic memory modules to refresh their memories.
14	<u>DEBUG</u>	DEBUG - This low level signal from the DEbug Module indicates that the DEbug Module is installed in the EXORciser. This is used to determine whether the VUA signal is controlled by the DEbug Module or the MPU Module.
15	BUSGNT	BUS GRANT - This signal is generated by the MPU Module in response to a low level <u>BUSREQ</u> . When high, this signal indicates that the MPU is not in control of the bus.
16	+12VDC	+12 Vdc Power - Available to the user for prototype module requirements. (2.5 Amps max.)
17	STANDBY	STANDBY Power - Same as STANDBY on Pin U.
18	CLK	CLOCK - A symmetrical clock signal generated by the MPU Module in phase with E. It is free running except when <u>used</u> with slow memories. During memory ready (low level MNRDY), the CLK signal is stretched high.
19	VXA	VALID EXECUTIVE ADDRESS - A high level signal generated by the DEbug Module in place of the VUA signal (refer to description of VUA on Pin 10) when the EXORciser is operating in the dual map mode and the EXbug program is addressing the executive portion of the memory map. Additionally, all peripheral modules (such as memories) must be set to respond to VXA signal if the user wants to operate those modules in the executive portion of the map.
20,21,22	GND	GROUND
23	BS	BUS STATUS - This signal is generated by the MPU Module. When high, this signal, in conjunction with the BA signal (pin P), determines the MPU halt, interrupt, and sync states.
24	GND(REF)	GROUND (REFERENCE) - Available to the user for prototype modules that require an isolated ground. This ground line is not connected to the normal EXORciser ground connection.
25,26 27,28		USER DEFINED - These signal lines, along with their counterpart pin numbers ( <u>C,D,E,F</u> ) are reserved by Motorola for possible expansion of the data bus to 16 bits. Since compatibility of 16-bit data bus modules with existing modules is unlikely, these eight lines may be used for custom modules.
29	<u>D1</u>	DATA bus (bit 1) - Same as <u>D3</u> on Pin <u>H</u> .
30	<u>D5</u>	DATA bus (bit 5) - Same as <u>D3</u> on Pin <u>H</u> .

TABLE 2. EXORciser Bus Signals (cont'd)

PIN NUMBER	SIGNAL MNEMONICS	SIGNAL NAME AND DESCRIPTION
31	$\overline{D_0}$	DATA bus (bit 0) - Same as $\overline{D_3}$ on Pin $\overline{H}$ .
32	$\overline{D_4}$	DATA bus (bit 4) - Same as $\overline{D_3}$ on Pin $\overline{H}$ .
33	A15	ADDRESS bus (bit 15) - Same as A14 on Pin $\overline{M}$ .
34	A12	ADDRESS bus (bit 12) - Same as A14 on Pin $\overline{M}$ .
35	A11	ADDRESS bus (bit 11) - Same as A14 on Pin $\overline{M}$ .
36	A8	ADDRESS bus (bit 8) - Same as A14 on Pin $\overline{M}$ .
37	A7	ADDRESS bus (bit 7) - Same as A14 on Pin $\overline{M}$ .
38	A4	ADDRESS bus (bit 4) - Same as A14 on Pin $\overline{M}$ .
39	A3	ADDRESS bus (bit 3) - Same as A14 on Pin $\overline{M}$ .
40	A0	ADDRESS bus (bit 0) - Same as A14 on Pin $\overline{M}$ .
41,42,43	GND	GROUND

## BUS SIGNAL TIMING SPECIFICATIONS

The second portion of Appendix A is intended to provide the user with detailed timing data and general application notes. It is written from the perspective of the design engineer who wants to design a new peripheral and/or memory module that will interface with the EXORciser bus. Therefore, timing data is presented with respect to the bus (internal timing on specific modules is not discussed). Figure 1 presents a block diagram illustrating this point of view (using the bus as the reference). Figure 2 provides a general timing diagram for the most important signals on the bus. Frequent reference will be made to these two figures throughout the following discussion.

## Assumptions

All timing data presented in this appendix is based upon the EXORciser system hardware consisting of the multilayer motherboard, the MPU Module (M6809MPU), and the DEbug Module (M6809DB). In addition, it is assumed that all interfacing to the bus is buffered with MC8T97 (MC6887) or equivalent devices. Timing data on each signal is based upon a capacitive load ( $C_L$ ) of 50 picofarads and a maximum device loading of 10 MC8T97 or equivalent devices (-4.0 mA @ 0.5V and 400 uA @ 2.4V). Vcc is always assumed to be 5.0 volts. Ground is 0 volts and is logic zero (low). Positive current is defined as into the terminal referenced.

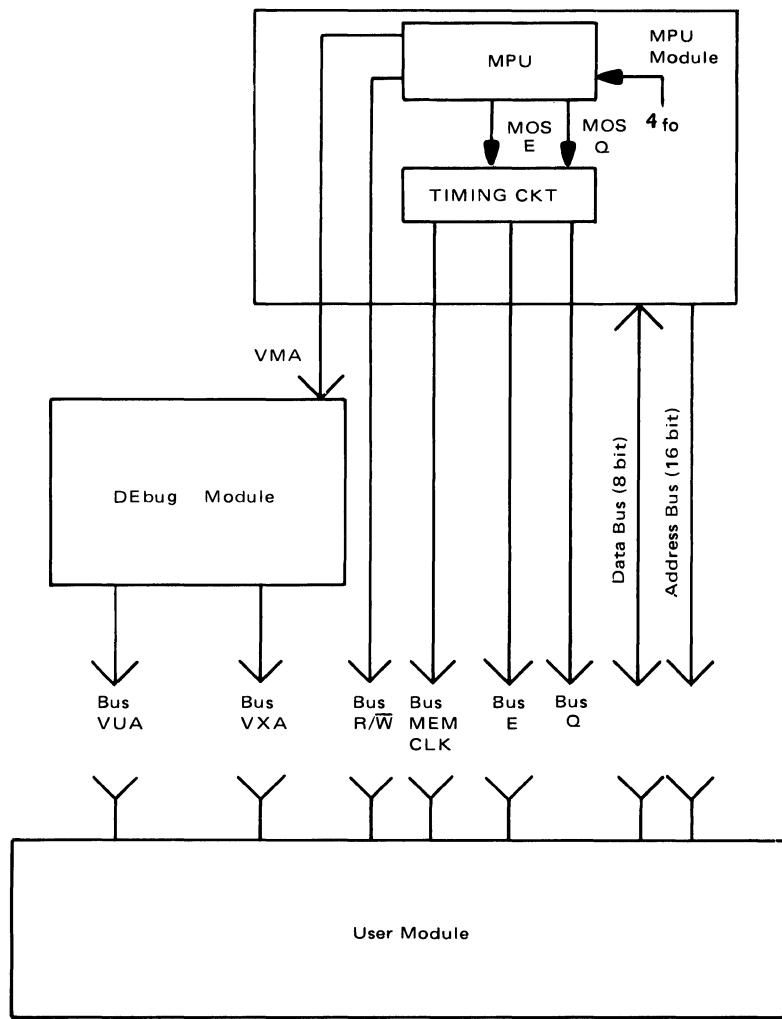


FIGURE 1. EXORciser Timing Signals Diagram (Bus View)

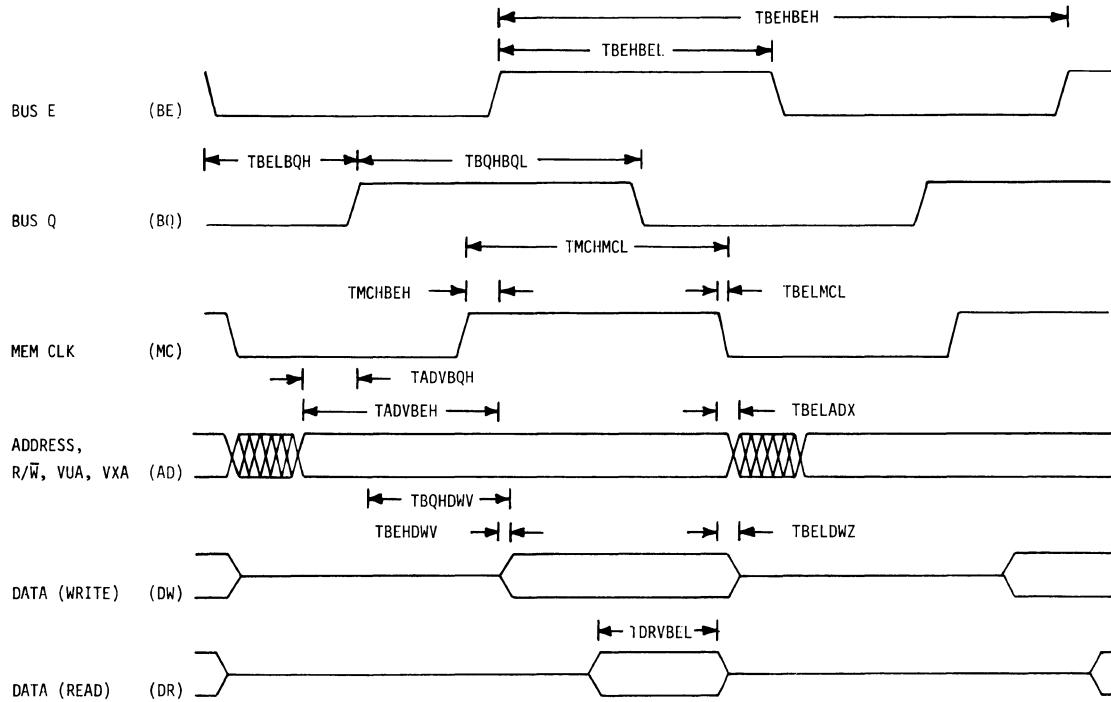


FIGURE 2. EXORciser Bus Specification Timing Diagram

#### Nomenclature and Abbreviations

All abbreviations shown in Figure 2 and Table 5 (as well as in the examples) use upper case characters with no subscripts. The initial character is always the letter T followed by a six-character descriptor. This descriptor is used to identify the to/from measurement points. The first three letters of the descriptor identify the signal name and transition level for the measurement starting point, while the last three letters identify the signal name and transition level for the measurement ending point. The descriptor format is illustrated in Table 3, while Table 4 lists the measurement abbreviations.

#### Description of Bus Timing and Examples

The bus timing and control signals are generated by the MPU and DEbug Modules. These signals will be described with reference to Figures 1, 2, and Table 5.

During instruction execution, the MPU uses both the address bus and data bus, regardless of whether an internal or external MPU operation is being performed. In order to prevent the system from using erroneous address and data information, the MPU generates the VMA signal. When VMA is high, the MPU is

addressing an external addressable location. During DMA and refresh operations, VMA is appropriately three-stated or pulled low to prevent operational errors. In the EXORciser system, the DDebug Module uses the high level VMA signal to generate either the VUA or the VXA signal. The high level VUA signal selects address locations within the User map, while the high level VXA signal selects locations within the Executive map. These signals will never be simultaneously high.

TABLE 3. Descriptor Format

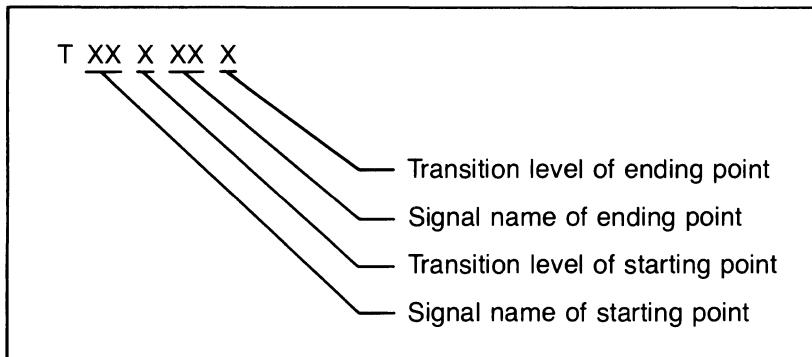


TABLE 4. Measurement Abbreviations

ABBREVIATIONS (Signal Names)		
AD	= Bus Address, Bus R/W, Bus VUA, or Bus VXA	
DW	= Write data from MPU to peripheral module	
DR	= Read data from peripheral module to MPU	
MC	= Bus Memory Clock	
BE	= Bus E clock signal	
BQ	= Bus Q clock signal	
ABBREVIATIONS (Transition Levels)		
H	= Low-To-High Transition	
L	= High-To-Low Transition	
V	= Transition to Valid State	
X	= Transition to Invalid or Don't Care State	
Z	= Transition to Off (High Impedance State)	
WAVEFORMS		
WAVEFORM SYMBOL	INPUT	OUTPUT
—	Must be valid	Will be valid
	Change from High-To-Low	Will change from High-To-Low
	Change from Low-To-High	Will change from Low-To-High
	Don't Care (Any change)	Changing state
	High impedance (Off)	

The time relationship between the bus signals is shown in Figure 2, while the minimum, typical, and maximum time values for the signals are listed in Table 5 (for 1.0 MHz, 1.5 MHz, and 2.0 MHz operation). All of these time values are referenced to either the leading edge or trailing edge of BUS E (BE) or BUS Q (BQ). Time relationships not specified within this table can be readily calculated from the information provided. Examples 1 and 2 present two typical problems, along with the calculations required to determine the solution to each.

From the analysis, it has been determined that the user module must place valid data on the data bus within 463 nanoseconds from the time that BUS ADDRESS, BUS R/W, and BUS VUA or BUS VXA become valid.

TABLE 5. EXORciser Bus Specifications

SIGNAL NAME	1.0 MHz			1.5 MHz			2.0 MHz		
	MIN	TYP	MAX	MIN	TYP	MAX	MIN	TYP	MAX
TBEHBEH	980	1000		647			480		
TBEHBEL	435	510		265			205		
TMCHBEH	-17	-4	8	-17	-4	8	30		
TMCHMCL	430	495		260			255		
TBELMCL	-7	4	16	-7	4	16	-7	4	16
TADVBEH	248	410		153			98		
TADVQBHQH	18	175		18			8		
TBELADX	9	50		9			9		
TBEHDWV		10	28			53			63
TBQHDWV		260	280			188			153
TBELDWZ	13	20		13			13		
TDRVBEL	123	220		103			83		
TBELBQH		250	260			175			135
TBQHBQL	435	500		265			205		

NOTE: All times are shown in nanoseconds. For information on values not shown in table, contact (800) 528-1908.

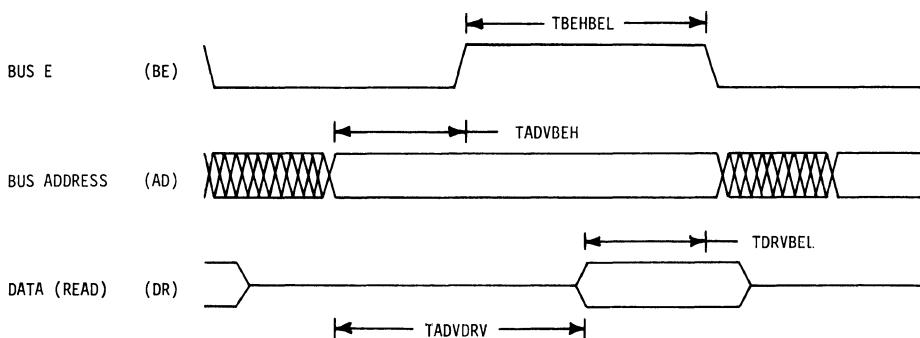
## EXAMPLE 1. Determining Data Ready Time (Read Operation)

### PROBLEM:

To determine, during a memory read operation, the minimum (worst case) time interval between the time that the BUS ADDRESS, BUS R/W, and BUS VUA or VXA become valid and the time that valid data must be on the data bus. (This time is TADVDRV).

### ANALYSIS:

From Figure 2, we obtain the BUS E, BUS ADDRESS, and DATA (READ) signals.



$$TADVDRV_{(MIN)} = TADVBEH_{(MIN)} + TBEHBEL_{(MIN)} - TDRVBEL_{(WORST\ CASE)}$$

$$TADVDRV_{(MIN)} = 248 \text{ nSEC} + 435 \text{ nSEC} - 220 \text{ nSEC}$$

$$TADVDRV_{(MIN)} = 463 \text{ nSEC}$$

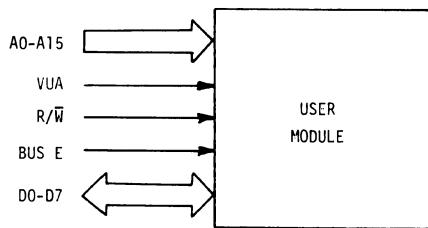
### CONCLUSION:

From the analysis, it has been determined that the user module must place valid data on the data bus within 463 nanoseconds from the time that BUS ADDRESS, BUS R/W, and BUS VUA or BUS VXA become valid.

## EXAMPLE 2. Determining the Data Margin During Read Operation

### PROBLEM:

To determine whether the memory module shown will function satisfactorily during a read operation. This example also works equally well for I/O devices such as the PIA or ACIA.



### ASSUMPTIONS:

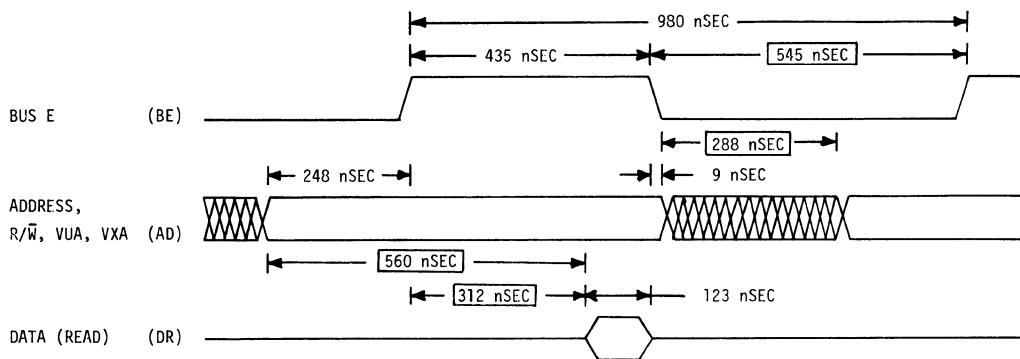
- (1) User module data access time from a valid address location:  
 $ta(A) = 340$  nanoseconds
- (2) User module data access time from BE going high:  
 $ta(BE) = 100$  nanoseconds
- (3) In order to occur during the read cycle, the last data access time to become valid must be selected:  $ta(A)$  or  $ta(BE)$ , whichever occurs later.
- (4) BUS ADDRESS, BUS R/ $\bar{W}$ , and BUS VUA or BUS VXA must be valid 30 nanoseconds prior to the leading edge of BUS BE.

### ANALYSIS:

- (1) From the bus specifications, determine the worst case conditions for a memory read operation and plot the timing diagram for a read cycle.
  - (a) TBEHBEH (min) = 980 nanoseconds
  - (b) TBEHBEL (min) = 435 nanoseconds
  - (c) TADVBEH (min) = 248 nanoseconds
  - (d) TBELADX (min) = 9 nanoseconds
  - (e) TDRVEBL (worst case) = 123 nanoseconds

Plot: Bus Read Specifications (Worst Case)

EXAMPLE 2. Determining the Data Margin During Read Operation (con't)



(2) From the timing plot produced in step 1, calculate the timing relationships required to evaluate the module read cycle. (These values are shown within boxes in the plot produced in the previous step.)

$$\begin{aligned}
 \text{(a) } \text{TADVDRV} &= \text{TADVBEH} + \text{TBEHBEL} - \text{TDRVBEL} \\
 \text{TADVDRV} &= 248 \text{ nsec} + 435 \text{ nsec} - 123 \text{ nsec} \\
 \text{TADVDRV} &= 560 \text{ nanoseconds}
 \end{aligned}$$

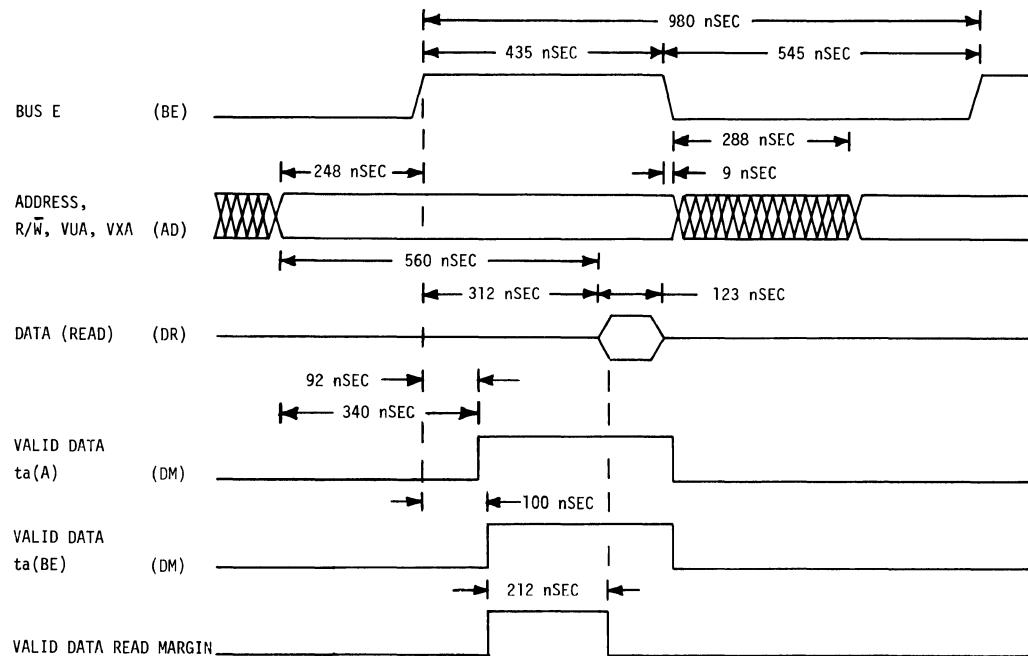
$$\begin{aligned}
 \text{(b) } \text{TBEHDRV} &= \text{TBEHBEL} - \text{TDRVBEL} \\
 \text{TBEHDRV} &= 435 \text{ nsec} - 123 \text{ nsec} \\
 \text{TBEHDRV} &= 312 \text{ nanoseconds}
 \end{aligned}$$

$$\begin{aligned}
 \text{(c) } \text{TBELBEH} &= \text{TBEHBEL} - \text{TBEHBEL} \\
 \text{TBELBEH} &= 980 \text{ nsec} - 435 \text{ nsec} \\
 \text{TBELBEH} &= 545 \text{ nanoseconds (not required for read cycle)}
 \end{aligned}$$

$$\begin{aligned}
 \text{(d) } \text{TADXADV} &= \text{TBELBEH} - \text{TBELADX} - \text{TADVBEH} \\
 \text{TADXADV} &= 545 \text{ nsec} - 9 \text{ nsec} - 248 \text{ nsec} \\
 \text{TADXADV} &= 288 \text{ nanoseconds}
 \end{aligned}$$

(3) From the timing plot produced in step 1 and the worst case bus read specifications, determine when valid data (DM) will be available from the module. Calculate and plot the data margin by referencing the times to TBEH.

EXAMPLE 2. Determining the Data Margin During Read Operation (cont'd)



(a) Using  $ta(A)$ :

$$\begin{aligned} TBEHDMV &= ta(A) - TADVBEH \\ TBEHDMV &= 340 \text{ nsec} - 248 \text{ nsec} \\ TBEHDMV &= 92 \text{ nanoseconds} \end{aligned}$$

(b) Using  $ta(BE)$ :

$$\begin{aligned} TBEHDMV &= ta(BE) \\ TBEHDMV &= 100 \text{ nanoseconds} \end{aligned}$$

(4) Valid data  $ta(A)$  occurs 92 nanoseconds after TBEH. Valid data  $ta(BE)$  occurs 100 nanoseconds after TBEH and is the limiting time factor. Therefore, data is valid 100 nanoseconds after TBEH. Since the bus specification requires that data be valid within 312 nanoseconds of TBEH or TBEHDRV, the valid data read margin is calculated using the following formula (illustrated in preceding step).

$$\begin{aligned} \text{Valid Data Read Margin} &= TBEHDRV - TBEHDMV \\ \text{Valid Data Read Margin} &= 312 \text{ nsec} - 100 \text{ nsec} \\ \text{Valid Data Read Margin} &= 212 \text{ nanoseconds} \end{aligned}$$

CONCLUSION:

The user module will operate satisfactorily for a read operation on the EXORciser bus.



**APPENDIX B**  
**EXBUG 2.1 PROGRAM FOR EXORciser**

00001 NAM EXBUG2  
 00002 TTL VERSION 2.1,6809 15 MAR 1979  
 00003 IDNT VERSION 2.1,6809 15 MAR 1979  
 00004 OPT NOCLIST DON'T LIST CONDITIONAL ASSEMBLY STATEMENTS  
  
 00006 \* EXBUG (TM) PROGRAM FOR EXORCISER (TM) SERIES-II  
 00007 \* COPYRIGHT 1979 BY MOTOROLA INC  
  
 00009 \* EXBUG COMMANDS  
 00010 \* LOAD(CR) LOADER  
 00011 \* VERF(CR) VERIFY  
 00012 \* PNCH(CR) PUNCH  
 00013 \* PRNT(CR) PRINT  
 00014 \* SRCH(CR) SEARCH  
 00015 \* MDOS(CR) LOAD AND START MDOS  
 00016 \* USER(CR) DEBUG IN THE USER MAP  
 00017 \* EXEC(CR) DEBUG IN THE EXECUTIVE MAP  
 00018 \* / DISPLAY/CHANGE MEMORY  
 00019 \* (LF) NEXT LOCATION  
 00020 \* (SPACE) PREVIOUS LOCATION  
 00021 \* / CLOSE/REOPEN CURRENT LOCATION  
 00022 \* (CR) CLOSE  
 00023 \* ;O CALCULATE SHORT RELATIVE OFFSET  
 00024 \* ;L CALCULATE LONG RELATIVE ADDRESS  
 00025 \* .A DISPLAY/CHANGE THE A ACCUMULATOR  
 00026 \* .B DISPLAY/CHANGE THE B ACCUMULATOR  
 00027 \* .C DISPLAY/CHANGE THE CONDITION CODE REGISTER  
 00028 \* .D DISPLAY/CHANGE THE DPR REGISTER  
 00029 \* ;E DISPLAY/CHANGE THE SWI RETURN SWITCH  
 00030 \* ;G GO (JUMP TO ADDRESS)  
 00031 \* \$H SET HALT ON ADDRESS  
 00032 \* ;H RESET HALT ON ADDRESS  
 00033 \* ;I INITIALIZE MEMORY  
 00034 \* ;K DISPLAY/CHANGE CONSOLE PAD VALUE  
 00035 \* \$M SET MEMORY SEARCH ADDR AND MASK  
 00036 \* ;M SET MEMORY SEARCH ADDR AND MASK  
 00037 \* ;N TRACE INSTRUCTION  
 00038 \* .P DISPLAY/CHANGE THE PROGRAM COUNTER  
 00039 \* ;P CONTINUE EXECUTION  
 00040 \* ;Q DISPLAY/CHANGE DEBUG OFFSET  
 00041 \* \$R DISPLAY REGISTERS  
 00042 \* ;R DISPLAY REGISTERS  
 00043 \* .S DISPLAY/CHANGE STACK POINTER  
 00044 \* \$T SET TRACE TO ENDING ADDRESS  
 00045 \* ;T RESET TRACE TO ENDING ADDRESS  
 00046 \* .U DISPLAY/CHANGE THE U REGISTER  
 00047 \* ;U REMOVE BREAKPOINTS  
 00048 \* \$V DISPLAY/SET BREAKPOINTS  
 00049 \* ;V DISPLAY/SET BREAKPOINTS  
 00050 \* ;W 8-BIT MEMORY SEARCH  
 00051 \* .X DISPLAY/CHANGE THE X REGISTER  
 00052 \* .Y DISPLAY/CHANGE THE Y REGISTER  
 00053 \* ;Z DISPLAY/CHANGE THE LINE PRINTER SWITCH  
 00054 \* ;I DISABLE PARITY ERROR INTERRUPT  
 00055 \* ;II ENABLE PARITY ERROR INTERRUPT  
 \* THE USER CAN ADD COMMANDS TERMINATED BY A CARRIAGE RETURN  
 \* BY SUPPLYING EXBUG WITH THE BEGINNING AND ENDING ADDRESSES  
 \* OF A COMMAND TABLE

00060  
00061

\* EXBUG COMMANDS RECOGNIZE THE FOLLOWING PARAMETER PREFIX:  
 \* - USE TWO'S COMPLEMENT

00063 \* THIS FILE GENERATES THE LISTING OF THE FIRST 1K OF  
 00064 \* EXBUG, WHICH IS SUPPLIED WITH THE SYSTEM, AS WELL AS  
 00065 \* THE ENTIRE EXBUG LISTING. THIS IS CONTROLLED BY THE  
 00066 \* CONDITIONAL ASSEMBLY FLAG LISTNG. LISTNG IS EQUATED  
 00067 \* TO ONE OF THE FOLLOWING LABELS AS DESIRED.  
 00068 0000 A ONEK EQU 0 ASSEMBLY LISTING OF FIRST 1K  
 00069 0001 A ALL EQU 1 ASSEMBLY LISTING OF ALL  
 00070 \* HERE'S THE CONDITIONAL ASSEMBLY FLAG  
 00071 0000 A LISTNG EQU ONEK

00073 \* EXBUG EQUATES  
 00074 FCF4 A ACIASC EQU \$FCF4 ACIA STATUS/CONTROL REGISTER  
 00075 FCF5 A ACIADT EQU \$FCF5 ACIA DATA REGISTER  
 00076 83FF A TOPTGT EQU \$83FF DEFAULT TOP OF VECTORS (EXEC MAP)  
 00077 FCF8 A HPIAAD EQU \$FCF8 HALT ON ADDRESS PIA - A DATA  
 00078 FCF9 A HPIAAC EQU HPIAAD+1 HALT ON ADDRESS PIA - A CNTL  
 00079 FCFA A HPIABD EQU HPIAAD+2 HALT ON ADDRESS PIA - B DATA  
 00080 FCFB A HPIABC EQU HPIAAD+3 HALT ON ADDRESS PIA - B CNTL  
 00081 FCFC A MPIAAD EQU HPIAAD+4 MAP CONTROL PIA - A DATA  
 00082 FCFD A MPIABD EQU HPIAAD+5 MAP CONTROL PIA - B DATA  
 00083 FCFE A MPIAAC EQU HPIAAD+6 MAP CONTROL PIA - A CNTL  
 00084 FCFF A MPIABC EQU HPIAAD+7 MAP CONTROL PIA - B CNTL  
 00085 FCFD A SBIT EQU MPIABD STOP BIT INDICATION  
 00086 003F A SWI EQU \$3F SWI INSTRUCTION  
 00087 008C A SKIP2 EQU \$8C CMPX IMMEDIATE INSTRUCTION

00089 \* MDOS LINE PRINTER DRIVER EQUATES  
 00090 EBC0 A LPINIT EQU \$EBC0 INITIALIZE LINE PRINTER INTERFACE  
 00091 EBCC A LIST EQU \$EBCC SEND CHARACTER TO LINE PRINTER

00093 \* MDOS BOOT LOAD ENTRY EQUATE  
 00094 E800 A MDOSE EQU \$E800

00097A F000		ORG	\$F000	
00102		* THE FOLLOWING JUMP TABLE IS COMMON TO EXBUG 1 AND 2		
00103A F000 16	02A7 F2AA	PWRUP LBRA	START	GET HERE FROM RESET
00104A F003 20	41	F046 XBEGEN BRA	BEGEND	GET BEGINNING AND ENDING ADDRESSES
00105A F005 12		NOP		HOLD ENTRY POINT
00106A F006 20	64	F06C XCBCDH BRA	CBCDHX	CONVERT ASCII HEX TO BINARY
00107A F008 12		NOP		HOLD ENTRY POOLINT
00108A F009 16	00EA F0F6 XCHEXL LBRA	CHEXL		CONVERT 4MSB TO ASCII HEX
00109A F00C 16	00EB F0FA XCHEXR LBRA	CHEXR		CONVERT 4LSB TO ASCII HEX
00110A F00F 20	73	F084 XINADD BRA	INADDR	GET HEX ADDRESS INDIRECT (X)
00111A F011 12		NOP		HOLD ENTRY POINT
00112A F012 16	00BD F0D2 XINCH LBRA	INCH		INPUT ONE CHARACTER
00113A F015 16	020D F225 XINCHN LBRA	INCHNP		INPUT ONE CHARACTER NO PARITY
00114A F018 16	0092 F0AD XOUTCH LBRA	CMNDE		OUTPUT CHARACTER (WITH SPEED PAD)
00115A F01B 16	008B F0A9 XOUT2H LBRA	OUT2HS		PRINT 2 HEX CHARS, SPACE (X)
00116A F01E 16	0086 F0A7 XOUT4H LBRA	OUT4HS		PRINT 4 HEX CHARS, SPACE (X)
00117A F021 20	18	F03B XPCRLF BRA	PCRLF	PRINT CR, LF
00118A F023 12		NOP		HOLD ENTRY POINT
00119A F024 20	07	F02D XPDATA BRA	PDATA	PRINT CR, LF, DATA STRING
00120A F026 12		NOP		HOLD ENTRY POINT
00121A F027 20	06	F02F XPDAT1 BRA	PDATA1	PRINT DATA STRING
00122A F029 12		NOP		HOLD ENTRY POINT
00123A F02A 20	7F	FOAB XPSpac BRA	PSPACE	PRINT SPACE
00124A F02C 12		NOP		HOLD ENTRY POINT

00126	*
00127	* I/O ROUTINES START HERE
00128	* THESE ROUTINES ARE ACCESSED FROM THE JUMP TABLE
00129	* AND ARE THE EXBUG 1 FUNCTIONAL EQUIVALENTS WITH
00130	* OPTIONAL LINE PRINTER OUTPUT
00131	*

00133	*
00134	* PRINT CR, LF, DATA STRING TERMINATED BY EOT
00135	*
00136	F02D A PDATA EQU *
00137A F02D 8D	F2 F021 BSR XPCRLF PRINT CR, LF
00138	* FALL INTO PDATA1 ROUTINE
00139	*
00140	* PRINT DATA STRING TERMINATED BY EOT
00141	*
00142	F02F A PDATA1 EQU *
00143A F02F A6	84 A LDA 0,X GET CHAR
00144A F031 81	04 A CMPA #4 EOT?
00145A F033 27	34 F069 BEQ END1 YES, RETURN
00146A F035 8D	E1 F018 BSR XOUTCH NO, SEND CHAR
00147A F037 30	01 A LEAX 1,X INC POINTER
00148A F039 20	F4 F02F BRA PDATA1 CONTINUE

```

00150          *
00151          * PRINT CR, LF, NULL
00152          *
00153      F03B   A PCRLF EQU  *
00154A F03B 86  0D   A     LDA   #$D    SEND CR
00155A F03D 8D  D9   F018  BSR   XOUTCH
00156A F03F 86  0A   A     LDA   #$A    SEND LF
00157A F041 8D  D5   F018  BSR   XOUTCH
00158A F043 4F          CLRA
00159A F044 20  D2   F018  BRA   XOUTCH    SEND NULL

```

```

00161          *
00162          * INPUT BEGINNING AND ENDING ADDRESSES
00163          *
00164      F046   A BEGEND EQU  *
00165A F046 8E  FB91   A     LDX   #MBEG  PRINT CR, LF, BEG
00166A F049 8D  D9   F024  BSR   XPDATA
00167A F04B 8E  FF0A   A     LDX   #BEGA  PRINT/CHANGE BEGA
00168A F04E 17  0557 F5A8  LBSR   PCMNID1
00169A F051 25  F3   F046  BCS   BEGEND  ERROR
00170A F053 8E  FB96   A ENDIN LDX   #MEND  PRINT CR, LF, END
00171A F056 8D  CC   F024  BSR   XPDATA
00172A F058 8E  FF0C   A     LDX   #ENDA  PRINT/CHANGE ENDA
00173A F05B 17  054A F5A8  LBSR   PCMNID1
00174A F05E 25  F3   F053  BCS   ENDIN  ERROR
00175A F060 8E  FF0A   A     LDX   #BEGA  POINT TO BEGA
00176A F063 EC  02   A     LDD   2,X    INSURE ENDA >= BEGA
00177A F065 A3  84   A     SUBD  0,X
00178A F067 25  DD   F046  BCS   BEGEND  BEGA LARGER
00179A F069 39          END1  RTS

```

```

00181          *
00182          * INPUT ONE ASCII HEX CHARACTER AND CONVERT TO BINARY
00183          *
00184      F06A   A IN1H EQU  *
00185A F06A 8D  A9   F015  BSR   XINCHN  INPUT CHARR
00186          * FALL INTO CBCDHX ROUTINE

```

```

00188          *
00189          * CONVERT ASCII HEX TO BINARY
00190          * IF NOT HEX, CHAR NOT CONVERTED AND N=1
00191          * IF HEX, CHAR CONVERTED AND N=0
00192          *
00193      F06C   A CBCDHX EQU  *
00194A F06C 81  46   A     CMPA  #'F
00195A F06E 22  11   F081  BHI   CBCDH1  NOT HEX

```

00196A F070 81 41 A CMPA #`A  
 00197A F072 25 05 F079 BCS CDEC CA  
 00198A F074 80 07 A SUBA #7 CORRECT A-F  
 00199A F076 84 0F A CBCDH3 ANDA #\$F MASK TO 4 LSB  
 00200A F078 39 RTS

00202 \* CONVERT ASCII DECIMAL TO BINARY  
 00203A F079 81 39 A CDEC CMPA #`9  
 00204A F07B 22 04 F081 BHI CBCDH1 ERROR, >9  
 00205A F07D 81 30 A CMPA #`0  
 00206A F07F 24 F5 F076 BCC CBCDH3 0-9  
 00207A F081 1A 08 A CBCDH1 ORCC #\$8 SET N  
 00208A F083 39 RTS

00210 \*  
 00211 \* INPUT ADDRESS, RETURN WITH:  
 00212 \* (A) LAST CHAR INPUT  
 00213 \* (B) NUMBER OF HEX CHARS INPUT  
 00214 \*  
 00215 F084 A INADDR EQU \*  
 00216A F084 4F CLRA INIT INPUT STORAGE  
 00217A F085 5F CLR B  
 00218A F086 ED 84 A STD 0, X  
 00219A F088 8D E0 F06A INADD1 BSR IN1H GET HEX CHAR  
 00220A F08A 2B 07 F093 BMI INADD3 NOT HEX  
 00221A F08C 8D 06 F094 BSR MERGEH MERGE INPUT WITH NUMBER  
 00222A F08E 5C INC B INC HEX CHAR COUNT  
 00223A F08F C1 05 A CMPB #5 HAVE MORE THAN 4 CHARS?  
 00224A F091 26 F5 F088 BNE INADD1 NO  
 00225A F093 39 INADD3 RTS

00227 \*  
 00228 \* MERGE HEX CHAR IN A (4 LSB) WITH 16 BIT HEX NUMBER (X)  
 00229 \*  
 00230 F094 A MERGEH EQU \*  
 00231A F094 34 06 A PSHS A,B SAVE ACC A,B  
 00232A F096 C6 04 A LDB #4 NUMBER\*16  
 00233A F098 68 01 A MERGE1 ASL 1,X  
 00234A F09A 69 84 A ROL 0,X  
 00235A F09C 5A DECB  
 00236A F09D 26 F9 F098 BNE MERGE1  
 00237A F09F AB 01 A ADDA 1,X INPUT+NUMBER  
 00238A F0A1 A7 01 A STA 1,X  
 00239A F0A3 35 86 A PULS A,B,PC RESTORE ACC B & RTS

00241 \*

00242 \* PRINT TOP OF DATA STACK  
 00243 \*  
 00244 F0A5 A OUTHEX EQU \*  
 00245A F0A5 1F 31 A TFR U,X  
 00246 \* FALL INTO OUT4HS  
 00247 \*  
 00248 \* PRINT 4 HEX CHARACTERS, SPACE (X)  
 00249 \*  
 00250 F0A7 A OUT4HS EQU \*  
 00251A F0A7 8D 19 F0C2 BSR OUT2H PRINT FIRST 2 HEX CHARS  
 00252 \* FALL INTO OUT2HS FOR LAST 2 HEX CHARS, SPACE  
 00253 \*  
 00254 \* PRINT 2 HEX CHARACTERS, SPACE (X)  
 00255 \*  
 00256 F0A9 A OUT2HS EQU \*  
 00257A F0A9 8D 17 F0C2 BSR OUT2H PRINT 2 HEX CHARS  
 00258 \* FALL INTO PSPACE  
 00259 \*  
 00260 \* PRINT SPACE  
 00261 \*  
 00262 FOAB A PSPACE EQU \*  
 00263A FOAB 86 20 A LDA #\$20 SPACE  
 00264 \* FALL INTO COMMAND OUTPUT CHARACTER

00266 \*  
 00267 \* COMMAND ECHO/OUTPUT ROUTINE  
 00268 \*  
 00269 FOAD A CMNDE EQU \*  
 00270A FOAD 7D FF67 A TST CASSET PUNCHING?  
 00271A F0B0 26 76 F128 BNE OUTCH YES  
 00272A F0B2 8D 74 F128 BSR OUTCH ALWAYS OUTPUT TO CONSOLE  
 00273A F0B4 7D FF37 A TST ZFLAG OUTPUT TO PRINTER?  
 00274A F0B7 27 DA F093 BEQ INADD3 NO  
 00275A F0B9 BD EBCC A CMNDE1 JSR LIST YES  
 00276A F0BC 24 D5 F093 BCC INADD3 OK  
 00277A F0BE 7F FF37 A CLR ZFLAG PRINTER ERROR, CLEAR Z FLAG  
 00278A F0C1 39 RTS

00280 \*  
 00281 \* OUTPUT 2 HEX CHARS (X)  
 00282 \*  
 00283 F0C2 A OUT2H EQU \*  
 00284A F0C2 A6 84 A LDA 0,X GET CHAR TO PRINT  
 00285A F0C4 34 02 OUT2H1 PSHA SAVE IT  
 00286A F0C6 8D 2E F0F6 BSR CHEXL CONVERT 4 MSB  
 00287A F0C8 8D E3 FOAD BSR CMNDE PRINT ASCII  
 00288A FOCA 35 02 PULA RESTORE CHAR  
 00289A FOCC 8D 2C FOFA BSR CHEXR CONVERT 4 LSB  
 00290A FOCE 30 01 A LEAX 1,X INC PONTR  
 00291A F0D0 20 DB FOAD BRA CMNDE

00293 \*  
 00294 \* INPUT ONE CHARACTER FROM CONSOLE  
 00295 \* ECHO CHARACTER IF AECHO CLEAR  
 00296 \*  
 00297 F0D2 A INCH EQU \*  
 00298A F0D2 B6 FCF4 A LDA ACIASC RECEIVE REG FULL?  
 00299A F0D5 47 ASRA  
 00300A F0D6 24 FA F0D2 BCC INCH NO  
 00301A F0D8 B6 FCF5 A LDA ACIADT YES, GET CHAR  
 00302A F0DB 7D FF58 A TST AECHO ECHO?  
 00303A F0DE 27 CD FOAD BEQ CMNDE YES  
 00304A F0E0 7F FF58 A CLR AECHO NO, RESET AECHO  
 00305A F0E3 39 RTS

00307 \*  
 00308 \* EXBUG COMMAND INPUT ROUTINE  
 00309 \*  
 00310 F0E4 A CMNDI EQU \*  
 00311A F0E4 17 013B F222 LBSR INNPNE ASSUME NO ECHO  
 00312A F0E7 81 0A A CMPA #\$A LF?  
 00313A F0E9 27 19 F104 BEQ CHEX1 YES, DON'T ECHO  
 00314A F0EB 81 0D A CMPA #\$D CR?  
 00315A F0ED 27 15 F104 BEQ CHEX1 YES, DON'T ECHO  
 00316A F0EF 81 18 A CMPA #\$18 CTL-X?  
 00317A F0F1 26 BA FOAD BNE CMNDE NO,ECHO CHAR  
 00318 \* YES, RETURN TO EXBUG

00320 \*  
 00321 \* EXBUG MAIN ENTRY ADDRESS - MUST BE AT \$F0F3  
 00322 \*  
 00323A F0F3 16 0250 F346 F0F3 LBRA RENTR2

00325 \*  
 00326 \* CONVERT 4 MSB TO ASCII HEX  
 00327 \*  
 00328 F0F6 A CHEXL EQU \*  
 00329A F0F6 44 LSRA SHIFT 4 MSB TO 4 LSB  
 00330A F0F7 44 LSRA  
 00331A F0F8 44 LSRA  
 00332A F0F9 44 LSRA  
 00333 \* FALL INTO CHEXR FOR CONVERSION  
 00334 \*  
 00335 \* CONVERT 4 LSB TO ASCII HEX  
 00336 \*  
 00337 FOFA A CHEXR EQU \*  
 00338A FOFA 84 OF A ANDA #\$F MASK TO 4 LSB

00339A	F0FC	8B	30	A	ADDA	#'0	ADD ASCII 0
00340A	F0FE	81	39	A	CMPA	#'9	
00341A	F100	23	02	F104	BLS	CHEX1	0-9
00342A	F102	8B	07	A	ADDA	#7	A-F
00343A	F104	39			CHEX1	RTS	

00345			*				
00346			*	Z COMMAND			
00347			*				
00348		F105	A	ZCMND EQU	*		
00349A	F105	8E	FF37	A	LDX	#ZFLAG	SAVE OLD Z FLAG
00350A	F108	A6	84	A	LDA	0,X	
00351A	F10A	34	12	A	PSHS	A,X	
00352A	F10C	17	043E	F54D	LBSR	ACMND1	
00353A	F10F	35	14	A	PULS	B,X	GET OLD Z FLAG
00354A	F111	25	F1	F104	BCS	CHEX1	ERROR
00355A	F113	A6	84	A	LDA	0,X	NEW ZFLAG
00356A	F115	26	0B	F122	BNE	ZCMND1	PRINTER NOW ON
00357A	F117	5D			TSTB		PRINTER NOW OFF
00358A	F118	27	EA	F104	BEQ	CHEX1	PRINTER WAS OFF
00359A	F11A	86	0D	A	LDA	#\$D	SEND CR, LF
00360A	F11C	8D	9B	F0B9	BSR	CMNDE1	
00361A	F11E	86	0A	A	LDA	#\$A	
00362A	F120	20	97	F0B9	BRA	CMNDE1	
00363			*				
00364A	F122	5D		ZCMND1	TSTB		
00365A	F123	26	DF	F104	BNE	CHEX1	PRINTER WAS ON
00366A	F125	7E	EBC0	A	JMP	LPINIT	INIT LP, PRINTER WAS OFF

00368			*				
00369			*	OUTPUT CHARACTER FROM ACC A TO CONSOLE			
00370			*	WITH SPEED PAD IF PUNCH FLAG OFF			
00371			*				
00372		F128	A	OUTCH EQU	*		
00373A	F128	17	00AC	F1D7	LBSR	OCHAR	SEND CHAR
00374A	F12B	34	06	A	PSHS	A,B	SAVE ACC A,B
00375A	F12D	84	7F	A	ANDA	#\$7F	RESET CHAR PARITY
00376A	F12F	81	0D	A	CMPA	#\$D	CR?
00377A	F131	26	15	F148	BNE	OUTCH5	NO
00378A	F133	C6	04	A	LDB	#4	4 NULLS IF CR AND PUNCHING
00379A	F135	7D	FF67	A	TST	CASSET	
00380A	F138	26	03	F13D	BNE	OUTCH1	PUNCHING
00381A	F13A	F6	FF03	A	LDB	CRNL	CR AND NOT PUNCHING
00382A	F13D	C4	7F	A	OUTCH1	ANDB	#\$7F RESET SIGN BIT (120,240 TI FLAG)
00383A	F13F	5A			DEC8		DONE PADDING?
00384A	F140	2B	0E	F150	BMI	OUTCH3	YES
00385A	F142	4F			CLRA		SEND NULL
00386A	F143	17	0091	F1D7	LBSR	OCHAR	
00387A	F146	20	F5	F13D	BRA	OUTCH1	

00389A	F148	F6	FF02	A	OUTCH5	LDB	CHARNL
00390A	F14B	7D	FF67	A	TST	CASSET	
00391A	F14E	27	ED	F13D	BEQ	OUTCH1	NOT CR AND NOT PUNCHING
00392A	F150	4F			OUTCH3	CLRA	TO CLEAR CARRY
00393A	F151	35	86	A	PULS	A,B,PC	NOT CR AND PUNCHING ,RTS

00395

\*

00396 \* PNCH COMMAND

00397

\*

00398		F153	A	PNCH	EQU	*	
00399A	F153	17	FEAD	F003	LBSR	XBEGEN	GET ADDRS
00400A	F156	8E	FBE2	A	PNCH1	LDX	#MHDR GET SO DATA
00401A	F159	17	FEC8	F024	LBSR	XPIDATA	
00402A	F15C	8E	FF94	A	LDX	#BUF+2	INIT BUFFER POINTER
00403A	F15F	C6	02	A	LDB	#2	INIT BYTE COUNT
00404A	F161	17	0283	F3E7	PNCH3	LBSR	INPUT1
00405A	F164	C1	15	A	CMPB	#21	
00406A	F166	24	EE	F156	BCC	PNCH1	BUFFER OVERFLOW
00407A	F168	81	0D	A	CMPA	#\$D	
00408A	F16A	26	F5	F161	BNE	PNCH3	NOT CR
00409A	F16C	8D	77	F1E5	BSR	SETUP	SET UP FOR SO
00410A	F16E	7C	FF67	A	INC	CASSET	PUNCH FLAG ON
00411A	F171	7D	FF02	A	TST	SPEED	
00412A	F174	2A	04	F17A	BPL	PNON1	NOT TI CONTROL
00413A	F176	86	30	A	LDA	#'0	TI PRINTER OFF
00414A	F178	8D	55	F1CF	BSR	DLE	
00415A	F17A	86	12	A	PNON1	LDA	DC2 - PUNCH ON
00416A	F17C	8D	59	F1D7	BSR	OCHAR	
00417A	F17E	17	0096	F217	LBSR	LDR	PUNCH LEADER
00418A	F181	C6	30	A	LDB	#'0	PUNCH RECORD TYPE, BYTE COUNT, ADDR
00419A	F183	8D	69	F1EE	BSR	TPUN	
00420A	F185	27	04	F18B	BEQ	PNCH15	NO MSG
00421A	F187	8D	77	F200	PNCH5	BSR	PUNCH MESSAGE
00422A	F189	26	FC	F187	BNE	PNCH5	NOT DONE
00423A	F18B	FE	FF0A	A	PNCH15	LDU	BEGA
00424A	F18E	FC	FF0C	A	PNCH7	LDD	ENDA CALC BYTE COUNT
00425A	F191	34	40	A	PSHS	U	
00426A	F193	A3	E1	A	SUBD	0,S++	
00427A	F195	1083	0018	A	CMPD	#24	
00428A	F199	25	02	F19D	BCS	PNCH11	
00429A	F19B	C6	17	A	PNCH9	LDB	#23
00430A	F19D	CB	04	A	PNCH11	ADD8	#4
00431A	F19F	F7	FF91	A	STB	BCONT	SAVE BYTE COUNT
00432A	F1A2	C6	31	A	LDB	#'1	PUNCH DATA RECORD
00433A	F1A4	8D	48	F1EE	BSR	TPUN	
00434A	F1A6	1F	31	A	TFR	U,X	
00435A	F1A8	17	08F6	FAA1	PNCH13	LBSR	FETCH
00436A	F1AB	8D	55	F202	BSR	PUNT1	
00437A	F1AD	26	F9	F1A8	BNE	PNCH13	
00438A	F1AF	33	1F	A	LEAU	-1,X	CORRECT AND UPDATE ADDRESS
00439A	F1B1	11B3	FF0C	A	CMPU	ENDA	
00440A	F1B5	25	D7	F18E	BCS	PNCH7	NO
00441A	F1B7	C6	03	A	LDB	#3	S9 RECORD BYTE COUNT
00442A	F1B9	8D	2A	F1E5	BSR	SETUP	SET UP S9 RECORD

00443A F1BB C6 39 A LDB #'9 PUNCH S9 RECORD  
 00444A F1BD 8D 2F F1EE BSR TPUN  
 00445A F1BF 8D 56 F217 BSR LDR PUNCH TRAILER  
 00446A F1C1 7F FF67 A CLR CASSET PUNCH FLAG OFF  
 00447A F1C4 86 14 A LDA #\$14 DC4 - PUNCH OFF  
 00448A F1C6 8D 0F F1D7 BSR OCHAR  
 00449A F1C8 7D FF02 A TST SPEED  
 00450A F1CB 2A 49 F216 BPL PUNT5 NOT TI CONTROL  
 00451 \* TI PRINTER ON ENTRY - CALLED FROM POWER-ON INITIALIZATION  
 00452A F1CD 86 39 A PN0FF3 LDA #'9 TI PRINTER ON  
 00453 \* FALL INTO DLE ROUTINE

00455 \*  
 00456 \* SEND DLE AND ACC A TO CONSOLE  
 00457 \*  
 00458 F1CF A DLE EQU \*  
 00459A F1CF 34 02 PSHA SAVE ACC A  
 00460A F1D1 86 10 A LDA #\$10 SEND DLE  
 00461A F1D3 8D 02 F1D7 BSR OCHAR  
 00462A F1D5 35 02 PULA RESTORE ACC A  
 00463 \* FALL INTO OCHAR

00465 \*  
 00466 \* OUTPUT CHARACTER FROM ACC A TO CONSOLE, NO SPEED PAD  
 00467 \* GET HERE FROM ENTERING AT \$F9CF  
 00468 \*  
 00469 F1D7 A OCHAR EQU \*  
 00470A F1D7 34 04 PSHB SAVE ACC B  
 00471A F1D9 F6 FCF4 A OCHAR1 LDB ACIASC TRANSMIT REG EMPTY?  
 00472A F1DC C5 02 A BITB #2  
 00473A F1DE 27 F9 F1D9 BEQ OCHAR1 NO  
 00474A F1E0 B7 FCF5 A STA ACIADT YES, SEND CHAR  
 00475A F1E3 35 84 A PULS B,PC RESTORE ACC B AND RTS

00477 \*  
 00478 \* SET UP BYTE COUNT, ADDR=0000  
 00479 F1E5 A SETUP EQU \*  
 00480A F1E5 8E FF91 A LDX #BCONT  
 00481A F1E8 CE 0000 A LDU #0 SET ADDR = 0  
 00482A F1EB E7 84 A STB 0,X SAVE COUNT  
 00483A F1ED 39 RTS  
 00484 \*  
 00485 \* PUNCH S, RECORD TYPE, BYTE COUNT, ADDR  
 00486 F1EE A TPUN EQU \*  
 00487A F1EE 86 53 A LDA #'S PUNCH S  
 00488A F1F0 8D E5 F1D7 BSR OCHAR

00489A	F1F2	1F	98	A	TFR	B,A	PUNCH RECORD TYPE
00490A	F1F4	8D	E1	F1D7	BSR	OCHAR	
00491A	F1F6	8E	FF91	A	LDX	#BCONT	
00492A	F1F9	5F			CLRB		INIT CKSM
00493A	F1FA	8D	04	F200	BSR	TPUN1	PUNCH BYTE COUNT
00494A	F1FC	EF	84	A	STU	0,X	GET ADDR
00495A	F1FE	8D	00	F200	BSR	TPUN1	PUNCH ADDR MSB
00496A	F200	A6	84	A	TPUN1	LDA	PUNCH ADDR LSB
00497			*				
00498			*				* PUNCH DATA
00499A	F202	34	02		PUNT1	PSHA	UPDATE CKSM
00500A	F204	EB	E0	A	ADD8	0,S+	
00501A	F206	17	FEBB	F0C4	PUNT3	LBSR	OUT2H1
00502A	F209	7A	FF91	A	DEC	BCONT	CHECK BYTE COUNT
00503A	F20C	26	08	F216	BNE	PUNT5	NOT DONE
00504A	F20E	53			COMB		PUNCH CKSM
00505A	F20F	1F	98	A	TFR	B,A	
00506A	F211	8D	F3	F206	BSR	PUNT3	
00507A	F213	8D	0A	F21F	BSR	XCRLF	PUNCH CR, LF
00508A	F215	4F			CLRA		SET Z FLAG (DONE)
00509A	F216	39			PUNT5	RTS	

00511		*					
00512		*					* PUNCH LEADER
00513		F217	A	LDR	EQU	*	
00514A	F217	C6	37	A	LDB	#55	COUNT
00515A	F219	4F			CLRA		NULL
00516A	F21A	8D	BB	F1D7	LDR1	BSR	OCHAR
00517A	F21C	5A			DEC8		
00518A	F21D	26	FB	F21A	BNE	LDR1	
00519A	F21F	16	FDFF	F021	XCRLF	LBRA	XPCRLF
00520			*				
00521			*				* INPUT ONE CHARACTER, NO PARITY OR ECHO
00522			*				
00523		F222	A	INNPNE	EQU	*	
00524A	F222	7C	FF58	A	INC	AECNO	NO ECHO
00525			*				* FALL INTO INCHNP
00526			*				
00527			*				* INPUT ONE CHARACTER, STRIP PARITY
00528			*				
00529		F225	A	INCHNP	EQU	*	
00530A	F225	17	FEAA	F0D2	LBSR	INCH	INPUT CHARACTER
00531A	F228	84	7F	A	ANDA	#\$7F	STRIP PARITY
00532A	F22A	39			RTS		

00534		*					* READER INPUT - IGNORE RUBOUTS
00535		F22B	A	RDIN	EQU	*	
00536A	F22B	8D	F5	F222	BSR	INNPNE	
00537A	F22D	81	7F	A	CMPA	#\$7F	
00538A	F22F	27	FA	F22B	BEQ	RDIN	RUBOUT, TRY AGAIN

00539A F231 39

RTS

00541A	F232	B4	FCFD	A	RDON3	ANDA	SBIT
00542A	F235	B7	FCF4	A		STA	ACIASC
00543A	F238	39				RTS	

00545			*				
00546			*	READ RECORD ROUTINE			
00547			*	GET HERE FROM ENTERING AT \$F8A4			
00548			*				
00549		F239	A	READR	EQU	*	
00550			*	READER ON			
00551A	F239	F6	FF02	A	LDB	SPEED	
00552A	F23C	2A	06	F244	BPL	RDON1	NOT TI CONTROL
00553A	F23E	86	37	A	READR9	LDA	#'7
							BLOCK FORWARD
00554A	F240	8D	8D	F1CF	BSR	DLE	
00555A	F242	20	08	F24C	BRA	READR1	
00556			*				
00557A	F244	86	11	A	RDON1	LDA	#\$11
							DC1 - READER ON
00558A	F246	8D	8F	F1D7	BSR	OCHAR	
00559A	F248	86	5F	A	LDA	#\$5F	READER RELAY ON
00560A	F24A	8D	E6	F232	BSR	RDON3	
00561A	F24C	8D	DD	F22B	READR1	BSR	RDIN GET CHAR
00562				*	IF TI AND CR READ ANOTHER RECORD		
00563A	F24E	81	0D	A	CMPA	#\$D	
00564A	F250	26	03	F255	BNE	READR3	NOT CR
00565A	F252	5D			TSTB		
00566A	F253	2B	E9	F23E	BMI	READR9	IS TI
00567A	F255	81	53	A	READR3	CMPA	#'S
00568A	F257	26	F3	F24C	BNE	READR1	FIRST CHAR NOT S
00569A	F259	8E	FF8F	A	LDX	#BUF-3	INIT BUFFER POINTER
00570A	F25C	8D	CD	F22B	BSR	RDIN	GET RECORD TYPE
00571A	F25E	A7	02	A	STA	2,X	SAVE IT IN BCNT
00572A	F260	81	39	A	CMPA	#'9	
00573A	F262	27	05	F269	BEQ	READR5	EOF
00574A	F264	80	30	A	SUBA	#'0	
00575A	F266	44			LSRA		
00576A	F267	26	E3	F24C	BNE	READR1	NOT HEADER OR DATA
00577A	F269	6F	01	A	READR5	CLR	1,X INIT CKSM
00578A	F26B	8D	1C	F289	BSR	RDBYTE	READ BYTE COUNT
00579A	F26D	6C	84	A	INC	0,X	CORRECT CKSM
00580A	F26F	1F	89	A	TFR	A,B	SAVE BYTE COUNT
00581A	F271	8D	16	F289	READR7	BSR	GET DATA
00582A	F273	5A			DEC B		CHECK BYTE COUNT
00583A	F274	26	FB	F271	BNE	READR7	NOT DONE
00584A	F276	86	04	A	LDA	#4	STORE ETX FOR HEADER
00585A	F278	A7	02	A	STA	2,X	
00586				*	READER OFF - CALLED FROM POWER ON INITIALIZATION		
00587A	F27A	86	3F	A	RDROFF	LDA	#\$3F
							READER RELAY OFF
00588A	F27C	8D	B4	F232	BSR	RDON3	

00589A	F27E	86	13	A	READR6	LDA	#\$13	DC3 - READER OFF
00590A	F280	17	FEA5	F128		LBSR	OUTCH	
00591A	F283	8D	00	F285		BSR	READR8	CLEAR ACIA
00592A	F285	B6	FCF5	A	READR8	LDA	ACIADT	
00593A	F288	39				RTS		

00595					* READ BYTE			
00596				F289	A	RDBYTE	EQU	*
00597A	F289	8D	1A	F2A5		BSR	RDINHX	GET CHAR
00598A	F28B	48				ASLA		MOVE TO 4 MSB
00599A	F28C	48				ASLA		
00600A	F28D	48				ASLA		
00601A	F28E	48				ASLA		
00602A	F28F	34	02	A	PSHS	A		AND SAVE IT
00603A	F291	8D	12	F2A5		BSR	RDINHX	GET CHAR
00604A	F293	AB	E0	A	ADDA	0,S+		FORM BYTE
00605A	F295	A7	03	A	STA	3,X		STORE BYTE
00606A	F297	BB	FF90	A	ADDA	BCKSM		UPDATE CKSM
00607A	F29A	B7	FF90	A	STA	BCKSM		
00608A	F29D	8C	FFD8	A	CPX	#BUF+70		
00609A	F2A0	27	02	F2A4	BEQ	RDBYT1		BUFFER OVERFLOW
00610A	F2A2	30	01		INX			
00611A	F2A4	39			RDBYT1	RTS		

00613					* READER INPUT - HEX CHARACTER			
00614				F2A5	A	RDINHX	EQU	*
00615A	F2A5	8D	84	F22B		BSR	RDIN	
00616A	F2A7	16	FDC2	F06C		LBRA	CBCDHX	

```

00618          *
00619          * EXBUG POWER-UP INITIALIZATION
00620          *
00621      F2AA A START EQU   *
00622A F2AA 10CE FF05 A LDS #XSTACK INIT SP
00623          * INITIALIZE RAM
00624A F2AE CE FF02 A LDU #ATOP+2
00625A F2B1 8E 83FF A LDX #TOPTGT
00626A F2B4 AF 5E A STX -2,U ATOP,U
00627A F2B6 8E FB61 A LDX #CRCMD
00628A F2B9 AF 4C A STX $C,U CMDBEG-ATOP,U
00629A F2BB 30 06 A LEAX 6,X
00630A F2BD AF 4E A STX $E,U CMDEND-ATOP,U
00631A F2BF 4F     CLRA
00632A F2C0 5F     CLR B
00633A F2C1 FD FFE6 A STD Q
00634A F2C4 ED C4   A STD 0,U
00635          * INITIALIZE HALT ON ADDRESS PIA
00636A F2C6 CE FCF4 A LDU #ACIASC PIA BASE ADDRESS
00637A F2C9 CC FF3C A LDD $$FF3C A DATA TO OUTPUT
00638A F2CC ED 44   A STD 4,U A DIR+CNTL
00639A F2CE C6 04   A LDB #$4 B DATA TO OUTPUT
00640A F2D0 ED 46   A STD 6,U B DIR+CNTL
00641          * INITIALIZE MAP CONTROL PIA
00642A F2D2 CC 7F7F A LDD $$7F7F A+B DATA OUTPUT
00643A F2D5 ED 48   A STD 8,U A+B DIR
00644A F2D7 86 2C   A LDA #$2C A CNTL
00645A F2D9 ED 4A   A STD 10,U A+B CNTL
00646          * COME UP IN EXEC MAP
00647A F2DB 17 0824 FB02 LBSR EXEC
00648          * INITIALIZE ACIA
00649A F2DE C6 03   A LDB #$3 RESET ACIA
00650A F2E0 E7 C4   A STB 0,U
00651A F2E2 C6 31   A LDB #$31 ASSUME 2 STOP BITS
00652A F2E4 E7 C4   A STB 0,U
00653A F2E6 8D 96   F27E BSR READR6 SEND 2 CHARACTERS
00654A F2E8 8D 94   F27E BSR READR6
00655A F2EA 1F 31   A TFR U,X
00656A F2EC 30 1F   A INACIA LEAX -1,X COUNT CHAR SEND TIME
00657A F2EE A6 C4   A LDA 0,U ACIASC
00658A F2F0 85 02   A BITA #2 CHAR SENT?
00659A F2F2 27 F8   F2EC BEQ INACIA NO
00660A F2F4 8C E700 A CPX $$E700 REALLY 2 STOP BITS?
00661A F2F7 2B 04   F2FD BMI IACIA1 YES
00662A F2F9 C6 35   A LDB #$35
00663A F2FB E7 C4   A STB 0,U ACIASC
00664A F2FD CA 40   A IACIA1 ORAB #$40 SAVE IN SBIT
00665A F2FF E7 49   A STB 9,U SBIT
00666A F301 7F FF1E A CLR PRDPR SET PSEUDO DPR=0
00667A F304 86 50   A LDA #$50 SET PSEUDO I, F MASKS
00668A F306 B7 FF21 A STA PRCC

00670          * GET HERE FROM ENTERING AT $F564
00671      F309 A RENTER EQU   *
00672A F309 10CE FF05 A LDS #XSTACK INIT SP
00673A F30D 8E FF90 A LDX #STACK INIT PSEUDO SP
00674A F310 BF FF22 A STX SPSAVE

```

00676		* INITIALIZE VECTORS - ABORT REENTRY POINT			
00677	F313	A RENTR1	EQU *		
00678		*INIT EXEC MAP IRQ VECTORS			
00679A	F313 CE	0000	A LDU	#0	U=TOP OF EXBUG'S POINTER
00680A	F316 BE	FF00	A LDX	ATOP	
00681A	F319 30	01	A LEAX	1,X	INX
00682A	F31B C6	18	A LDB	#24	
00683A	F31D A6	82	A RENTR4	LDA	0,-X
00684A	F31F A7	C2	A STA	0,-U	
00685A	F321 5A			DEC B	
00686A	F322 26	F9 F31D	BNE	RENTR4	
00687A	F324 8E	F8C3	A LDX	#NMISRV	INIT NMI VECTOR
00688A	F327 BF	FFFC	A STX	\$FFFC	
00689A	F32A 8E	F827	A LDX	#SWISRV	INIT SWI VECTOR
00690A	F32D BF	FFFA	A STX	\$FFFA	
00692		* CLEAR EXBUG RAM			
00693A	F330 C6	44	A LDB	#ZEND-BKADDR	
00694A	F332 17	0355 F68A	LBSR	UCMND1	
00696A	F335 86	3A	A LDA	#'	TI RDC ON
00697A	F337 17	FE95 F1CF	LBSR	DLE	
00698A	F33A 17	FE90 F1CD	LBSR	PNOFF3	PRINTER ON
00699A	F33D 17	FF3A F27A	LBSR	RDROFF	READER OFF
00701A	F340 8E	FBE7	A LDX	#HDNG	PRINT HEADING
00702A	F343 17	FCDE F024	LBSR	XPDATA	
00704		* GET HERE FROM ENTERING AT \$F0F3 OR \$F5C2			
00705	F346	A RENTR2	EQU *		
00706A	F346 10CE	FFE5	A LDS	#XSTACK	INIT SP
00707		* ISSUE PROMPT			
00708A	F34A 8E	FBD7	A LDX	#PRM1	PRINT *
00709A	F34D 17	FCD4 F024	LBSR	XPDATA	
00710A	F350 B6	FFE5	A LDA	MODE	
00711A	F353 26	05 F35A	BNE	CLP1	IS USER MODE
00712A	F355 30	01	A LEAX	1,X	
00713A	F357 17	FCCD F027	LBSR	XPDAT1	
00714A	F35A 4F		CLP1 CLRA		NO PARAMETERS YET
00715A	F35B 17	021E F57C	LBSR	INDSP	INIT DATA STACK POINTER
00716A	F35E 17	0083 F3E4 CLP9	LBSR	INPUT	GET FIELD
00717A	F361 25	4E F3B1	BCS	CLP3	ERROR
00718A	F363 81	0D	A CMPA	#\$D	CR TERMINATOR?
00719A	F365 27	32 F399	BEQ	CLP5	YES, CR COMMAND
00720A	F367 17	00A9 F413	LBSR	CNVRT	CONVERT INPUT FIELD
00721A	F36A 8E	FB3F	A LDX	#PCMDT-6	INIT PERIOD CMD TABLE POINTER
00722A	F36D 81	2E	A CMPA	#'	PERIOD?
00723A	F36F 27	0F F380	BEQ	CLP7	YES
00724A	F371 8E	FB08	A LDX	#SCMD-6	INIT TABLE POINTER
00725A	F374 81	2F	A CMPA	#'/	SLASH?
00726A	F376 27	0D F385	BEQ	CLP13	YES
00727A	F378 81	24	A CMPA	#'\$	\$?
00728A	F37A 27	04 F380	BEQ	CLP7	YES
00729A	F37C 81	3B	A CMPA	#';	SEMICOLON?
00730A	F37E 26	DE F35E	BNE	CLP9	NO, GET NEXT PARAMETER
00731		* PERIOD, SEMICOLON, OR DOLLAR SIGN			
00732	F380	A CLP7	EQU *		
00733A	F380 17	FD61 F0E4	LBSR	CMND1	GET COMMAND CHAR

00734A	F383	1F	89	A	TFR	A,B	SAVE COMMAND CHAR IN B	
00735A	F385	30	03	A	CLP13	LEAX	3,X	UPDATE POINTER
00736A	F387	6D	03	A	TST	3,X	DONE?	
00737A	F389	27	26	F3B1	BEQ	CLP3	YES, ERROR	
00738A	F38B	A1	03	A	CMPA	3,X	MATCH?	
00739A	F38D	26	F6	F385	BNE	CLP13	NO	
00740A	F38F	17	FC98	F02A	CLP15	LBSR	XPSpac	PRINT SPACE
00741A	F392	AD	98	04	A	JSR	[4,X]	GET COMMAND ADDRESS
00742A	F395	25	1A	F3B1	BCS	CLP3	ERROR	
00743A	F397	20	AD	F346	BRA	RENTR2	NEXT COMMAND	

00745	* CR COMMAND							
00746A	F399	8E	FB61	A	CLP5	LDX	#CRCMD	CHECK EXBUG COMMANDS FIRST
00747A	F39C	8D	18	F3B6	CLP17	BSR	CMDCHK	LOOK FOR MATCH
00748A	F39E	24	EF	F38F	BCC	CLP15	FOUND IT	
00749A	F3A0	8C	FB91	A	CPX	#CRCMDE	CECKED ALL OF TABLE?	
00750A	F3A3	26	F7	F39C	BNE	CLP17	NO	
00751A	F3A5	BE	FF0E	A	LDX	CMDBEG	CHECK USER TABLE	
00752A	F3A8	8D	0C	F3B6	CLP19	BSR	CMDCHK	LOOK FOR MATCH
00753A	F3AA	24	E3	F38F	BCC	CLP15	FOUND IT	
00754A	F3AC	BC	FF10	A	CPX	CMDEND	CHECKED ALL OF TABLE?	
00755A	F3AF	26	F7	F3A8	BNE	CLP19	NO	
00757A	F3B1	17	0129	F4DD	CLP3	LBSR	PERR	PRINT ERROR MESSAGE
00758A	F3B4	20	90	F346	BRA	RENTR2		

00760	* LOOK FOR CR COMAND TABLE MATCH							
00761		F3B6	A	CMDCHK	EQU	*		
00762A	F3B6	FC	FF92	A	LDD	BUF	CHECK FIRST & SECOND CHAR	
00763A	F3B9	81	0D	A	CMPA	#\$D		
00764A	F3BB	27	89	F346	BEQ	RENTR2	REPROMPT IF CR	
00765A	F3BD	A3	84	A	SUBD	0,X		
00766A	F3BF	26	07	F3C8	BNE	INX6		
00767A	F3C1	FC	FF94	A	LDD	BUF+2	CHECK THIRD AND FOURTH CHAR	
00768A	F3C4	A3	02	A	SUBD	2,X		
00769A	F3C6	27	15	F3DD	BEQ	CKBRK1	FOUND IT	
00771A	F3C8	30	06	A	INX6	LEAX	6,X	
00772A	F3CA	1A	01		SEC	SEC		NOT FOUND FLAG
00773A	F3CC	39				RTS		

00775	*							
00776	* CHECK FOR BREAK/WAIT							
00777	*							
00778		F3CD	A	CKBRK	EQU	*		
00779A	F3CD	B6	FCF4	A	LDA	ACIASC	CHECK ACIA	
00780A	F3D0	47			ASRA			
00781A	F3D1	24	0A	F3DD	BCC	CKBRK1	NO CHAR	
00782A	F3D3	17	FE4C	F222	CKBRK7	LBSR	INNPNE	
00783A	F3D6	80	17	A	SUBA	#\$17	CTL-W?	
00784A	F3D8	27	F9	F3D3	BEQ	CKBRK7	YES	
00785A	F3DA	4A			DECA		CTL-X?	
00786A	F3DB	27	D4	F3B1	BEQ	CLP3	YES, RETURN TO COMMAND LEVEL	
00787A	F3DD	39			CKBRK1	RTS		

00789 \*  
 00790 \* MDOS COMMAND  
 00791 \*  
 00792 F3DE A MDOS EQU \*  
 00793A F3DE 17 0721 FB02 LBSR EXEC SET EXEC MAP  
 00794A F3E1 7E E800 A JMP MDOSE

00796 \*  
 00797 \* INPUT FIELD TO BUFFER  
 00798 \*  
 00799 F3E4 A INPUT EQU \*  
 00800A F3E4 8E FF92 A LDX #BUF INIT BUFFER POINTER  
 00801A F3E7 17 FCFA F0E4 INPUT1 LBSR CMNDI GET CHAR  
 00802A F3EA A7 80 A STA 0,X+ SAVE CHAR, INC BUF POINTER  
 00803A F3EC 5C INCB INC CHAR COUNT  
 00804A F3ED 8D 0B F3FA BSR CKTRM TERMINATOR?  
 00805A F3EF 24 14 F405 BCC CKTRM1 YES  
 00806A F3F1 8C FFA6 A CPX #BUF+20 BUFFER FULL?  
 00807A F3F4 25 F1 F3E7 BCS INPUT1 NO  
 00808A F3F6 20 D2 F3CA BRA SEC YES, ERROR

00810 \*  
 00811 \* GET CHARACTER FROM BUFFER - CHECK IF TERMINATOR  
 00812 \*  
 00813 F3F8 A GETC EQU \*  
 00814A F3F8 A6 80 A LDA 0,X+ GET CHAR, UPDASTE POINTER  
 00815 \* FALL INTO CKTRM

00817 \*  
 00818 \* CHECK FOR TERMINATOR  
 00819 \*  
 00820A F3FA 31 8C 09 CKTRM LEAY CTRMTB, PCR CHECK AGAINST TERM TABLE  
 00821A F3FD 6D A4 A CKTRM3 TST 0,Y CHECKED ALL OF TABLE?  
 00822A F3FF 27 C9 F3CA BEQ SEC YES  
 00823A F401 A1 A0 A CMPA 0,Y+  
 00824A F403 26 F8 F3FD BNE CKTRM3 NOT TERMINATOR  
 00825A F405 39 CKTRM1 RTS IS TERMINATOR  
 00826 \*TERMINATOR TABLE  
 00827 \* SPACE, CR, LF, COMMA, SLASH, SEMICOLON, DOLLAR SIGN, PERIOD  
 00828 F406 A TRMTB EQU \*  
 00829A F406 20 A FCB \$20,\$D,\$A,'/,/,/,/,/\$,  
 00830A F40E 00 A FCB 0 END OF TABLE

		OPT	LIST
02080		ORG	\$FF00
02081A FF00	0002	A ATOP RMB	2 TOP OF EXEC MAP VECTORS
02087	FF02	A CHARNL EQU	* SPEED FLAG AND CHAR (NON-CR) NULL PAD VALUE
02088A FF02	0001	A SPEED RMB	1 0=DC, -=TI CNTL, OTHER=COMB
02089A FF03	0001	A CRNL RMB	1 CR NULL PAD VALUE
02090A FF04	0002	A HALTAD RMB	2 HALT ADDRESS
02091A FF06	0002	A TEND RMB	2 TRACE ENDING ADDRESS
02092A FF08	0002	A RMB	2 RESERVED DATA STACK POINTER
02093A FF0A	0002	A BEGA RMB	2 BEGINNING ADDR - BEGEND
02094A FF0C	0002	A ENDA RMB	2 ENDING ADDRESS - BEGEND
02095A FF0E	0002	A CMDBEG RMB	2 EXTENDED CMND TABLE BEGIN
02096A FF10	0002	A CMDEDN RMB	2 EXTENDED CMND TABLE END
02097A FF12	0002	A TEMP A RMB	2 TEMP STORAGE
02098A FF14	0002	A TEMP B RMB	2 TEMP STORAGE
02099		* PSEUDO-REGISTERS FOR TARGET PROGRAM	
02100A FF16	0001	A PRPCH RMB	1 TGT PC HIGH
02101A FF17	0001	A RMB	1 TGT PC LOW
02102A FF18	0001	A PRURH RMB	1 TGT U REG HIGH
02103A FF19	0001	A RMB	1 TGT U REG LOW
02104A FF1A	0001	A PRYRH RMB	1 TGT Y REG. HIGH
02105A FF1B	0001	A RMB	1 TGT Y REG. LOW
02106A FF1C	0001	A PRXRH RMB	1 TGT X REG. HIGH
02107A FF1D	0001	A RMB	1 TGT X REG. LOW
02108A FF1E	0001	A PRDPR RMB	1 TGT DPR
02109A FF1F	0001	A PRACCA RMB	1 TGT ACC A
02110A FF20	0001	A PRACCB RMB	1 TGT ACC B
02111A FF21	0001	A PRCC RMB	1 TGT CONDITION CODES .. HINZVC
02112A FF22	0002	A SPSAVE RMB	2 TGT SP
02113		* END OF PSEUDO-REGISTERS	
02114		* BEGINNING OF EXBUG 1 CLEARED AREA	
02115A FF24	0010	A BKADDR RMB	16 BREAKPOINT ADDRESSES
02116A FF34	0001	A BKINS RMB	1 BKPT INSTRUCTIONS, (ALTERNATING BYTES)
02117A FF35	0001	A EFLAG RMB	1 E COMMAND FLAG
02118A FF36	0001	A RMB	1 BKPT INS 2
02119A FF37	0001	A ZFLAG RMB	1 Z COMMAND FLAG
02120A FF38	0001	A RMB	1 BKPT INS 3
02121A FF39	0001	A WMSK RMB	1 MEMORY SEARCH MASK
02122A FF3A	0001	A RMB	1 BKPT INS 4
02123A FF3B	0001	A HALTF RMB	1 HALT ON ADDR ACTIVE FLAG
02124A FF3C	0001	A RMB	1 BKPT INS 5
02125A FF3D	0001	A TRACE RMB	1 TRACE ACTIVE FLAG
02126A FF3E	0001	A RMB	1 BKPT INS 6
02127A FF3F	0001	A INTMAP RMB	1 MAP LAST INTERRUPT WAS IN: +=EXEC, -=USER
02128A FF40	0001	A RMB	1 BKPT INS 7
02129A FF41	0001	A TEMP C RMB	1 TEMP STORAGE
02130A FF42	0001	A RMB	1 BKPT INS 8
02131A FF43	0001	A PCOUNT RMB	1 PARAM COUNT, NULL FIELD FLAG
02132A FF44	0010	A BKCNT RMB	16 BKPT PASS COUNT
02133A FF54	0001	A BKPTIN RMB	1 BKPTS IN FLAG
02134A FF55	0002	A BUFPNT RMB	2 BUFFER POINTER
02135A FF57	0001	A TEMP D RMB	1 TEMP STORAGE
02136A FF58	0001	A AECHO RMB	1 NON-ECHO FLAG
02137A FF59	000E	A RMB	14 DATA STACK AREA
02138	FF67	A DATAS EQU	* START OF DATA STACK
02139A FF67	0001	A CASSET RMB	1 PUNCH ON FLAG
02140	FF68	A ZEND EQU	*
02141		* END OF EXBUG 1 CLEARED AREA	

02142A FF68	0028	A	RMB	40	EXBUG 1 STACK AREA
02143	FF90	A STACK	EQU	*	
02144A FF90	0001	A BCKSM	RMB	1	BUFFER CHECKSUM
02145A FF91	0001	A BCNT	RMB	1	BUFFER CONTROL CHARACTER
02146A FF92	0053	A BUF	RMB	83	RECORD BUFFER
02147	FFE5	A XSTACK	EQU	*	EXBUG 2 STACK
02148A FFE5	0001	A MODE	RMB	1	MAP MODE
02149A FFE6	0002	A Q	RMB	2	DEBUG OFFSET
02150A FFE8	0010	A	RMB	16	PIC VECTORS
02151A FFF8	0002	A	RMB	2	IRQ VECTOR
02152A FFFA	0002	A	RMB	2	SWI VECTOR
02153A FFFC	0002	A	RMB	2	NMI VECTOR
02154A FFFE	0002	A	RMB	2	RESET VECTOR

02156 F000 A END PWRUP

TOTAL ERRORS 00000--00000

TOTAL WARNINGS 00000--00000

FCF5 ACIADT 00075\*00301 00474 00592  
 FCF4 ACIASC 00074\*00298 00471 00542 00636 00779  
 F54A ACMND 01042\*02004  
 F54D ACMND1 00352 01044\*01059 01095 01102 01109 01119  
 F55E ACMND3 01049 01050 01052\*01072  
 F46C ADD 00900 00906\*00955 01152  
 FF58 AECHO 00302 00304 00524 00878 02136\*  
 0001 ALL 00069\*  
 FF00 ATOP 00624 00680 01470 01592 01709 02086\*  
 FF90 BCKSM 00606 00607 01872 02144\*  
 F55F BCMND 01057\*02005  
 FF91 BCNT 00431 00480 00491 00502 02145\*  
 FF0A BEGA 00167 00175 00423 01735 02093\*  
 F046 BEGEND 00104 00164\*00169 00178  
 FF24 BKADDR 00693 01306 01307 01344 02115\*  
 FF44 BKCNT 02132\*  
 FF34 BKINS 01306 01350 02116\*  
 FF54 BKPTIN 01317 01365 01385 01559 02133\*  
 F6A5 BMTCH 01290 01327\*01564  
 F6A8 BMTCH1 01287 01303 01328\*  
 F6AA BRKSB 01285 01320 01342\*01382  
 F6B3 BRKSB1 01347\*01354  
 F6EB BRKSB3 01351 01372\*  
 F6B8 BRKSB5 01345 01349\*01357 01366 01369 01377  
 FF92 BUF 00402 00569 00608 00762 00767 00800 00806 00847 01199 01847 01862 01876 01879 02146\*  
 FF55 BUFPNT 02134\*  
 FF67 CASSET 00270 00379 00390 00410 00446 02139\*  
 F081 CBCDH1 00195 00204 00207\*  
 F076 CBCDH3 00199\*00206  
 F06C CBCDHX 00106 00193\*00616  
 F583 CCMND 01093\*02006  
 F079 CDEC 00197 00203\*  
 FF02 CHARNL 00389 01183 02087\*  
 F104 CHEX1 00313 00315 00341 00343\*00354 00358 00365  
 F0F6 CHEXL 00108 00286 00328\*  
 F0FA CHEXR 00109 00289 00337\*  
 F3CD CKBRK 00778\*01715 01852 01946  
 F3DD CKBRK1 00769 00781 00787\*  
 F3D3 CKBRK7 00782\*00784  
 F3FA CKTRM 00804 00820\*  
 F405 CKTRM1 00805 00825\*  
 F3FD CKTRM3 00821\*00824  
 F35A CLP1 00711 00714\*  
 F385 CLP13 00726 00735\*00739  
 F38F CLP15 00740\*00748 00753  
 F39C CLP17 00747\*00750  
 F3A8 CLP19 00752\*00755  
 F3B1 CLP3 00717 00737 00742 00757\*00786 00891  
 F399 CLP5 00719 00746\*  
 F380 CLP7 00723 00728 00732\*  
 F35E CLP9 00716\*00730  
 FF0E CMDBEG 00751 02095\*

F3B6 CMDCHK 00747 00752 00761\*  
 FF10 CMDEND 00754 02096\*  
 FOAD CMNDE 00114 00269\*00287 00291 00303 00317  
 F0B9 CMNDE1 00275\*00360 00362  
 F0E4 CMND1 00310\*00733 00801 00936 01762 01798 01878  
 F413 CNVRT 00720 00845\*  
 F44C CNVRT7 00869 00875\*00890  
 F42A CNVRT9 00852 00857\*  
 F731 CONT1 01398 01404 01415\*01456  
 F72F CONT3 01400 01414\*01573 01721  
 F716 CONT5 01401\*01706  
 F71A CONT9 01403\*01582  
 F648 COUTAD 01250 01260\*  
 FB61 CRCMD 00627 00746 02026\*  
 FB91 CRCMDE 00749 02035\*  
 FF03 CRNL 00381 02089\*  
 FF67 DATAS 01087 01718 02138\*  
 F588 DCMND 01100\*02007  
 F53B DEC 00957 00963 01015 01024\*01409 01719  
 F541 DEC1 01022 01028\*  
 F892 DECP1 01606\*01653  
 F88A DECPC 01567 01586 01602\*  
 F8DE DELAY1 01656\*01657  
 F1CF DLE 00414 00458\*00554 00697  
 F58D ECMND 01107\*01979  
 FF35 EFLAG 01108 01387 01584 02117\*  
 F069 END1 00145 00179\*  
 FFOC ENDA 00172 00424 00439 01224 01749 02094\*  
 F053 ENDIN 00170\*00174  
 F47C ERR 00918\*00935 00962 00990 01076  
 FB02 EXEC 00647 00793 01648 01955\*02034  
 F0F3 F0F3 00323\*  
 F564 F564 01064\*  
 F5C2 F5C2 01149\*  
 F8A4 F8A4 01620\*  
 F9CF F9CF 01782\*  
 FAA1 FETCH 00435 00927 00996 01361 01367 01533 01631 01742 01829 01900\*01924  
 F8AC FETCH2 01471 01580 01627\*01685  
 F8B2 FETCHX 01628 01631\*01668  
 F40F GCFLD 00838\*00930 00933 01047 01136 01142  
 F77E GCMND 01461\*01980  
 F795 GCMND3 01469 01471\*  
 F79D GCMND5 01466 01474\*  
 F7A0 GCMND7 01473 01475\*  
 F7AC GCMND9 01478 01480\*  
 F3F8 GETC 00813\*00851 00860 00866  
 FF04 HALTAD 01198 01203 01510 02090\*  
 FF3B HALTF 01203 01508 02123\*  
 F5ED HCMND 01197\*01981  
 F5F9 HCMND1 01201 01203\*  
 F5F0 HCMND5 01199\*01212  
 FBE7 HDNG 00701 02062\*  
 F431 HEX 00858 00861\*00867  
 FCF9 HPIAAC 00078\*01514 01607 01688  
 FCF8 HPIAAD 00077\*00078 00079 00080 00081 00082 00083 00084 01512 01689  
 FCFB HPIABC 00080\*01696  
 FCFA HPIABD 00079\*01511 01697  
 F2FD IACIA1 00661 00664\*

F605 ICMND 01217\*01982  
F610 ICMND1 01223\*01227  
F06A IN1H 00184\*00219  
F2EC INACIA 00656\*00659  
F088 INADD1 00219\*00224  
F093 INADD3 00220 00225\*00274 00276  
F084 INADDR 00110 00215\*  
F533 INC 01009 01018\*01453 01748 01846 01929  
F0D2 INCH 00112 00297\*00300 00530  
F225 INCHNP 00113 00529\*  
F57C INDSP 00715 01036 01085\*01561  
F222 INNPNE 00311 00523\*00536 00782  
F3E4 INPUT 00716 00799\*00839  
F3E7 INPUT1 00404 00801\*00807  
FF3F INTMAP 01519 02127\*  
F3C8 INX6 00766 00771\*  
F800 ISP1 01520 01531\*  
F806 ISP5 01533\*01537  
F7EF ISPREP 01517\*01558 01652  
F5E1 KCMND 01182\*01983  
F5E4 KCMND1 01184\*01192  
F71D LCMND1 01397 01406\*  
F217 LDR 00417 00445 00513\*  
F21A LDR1 00516\*00518  
EBCC LIST 00091\*00275  
0000 LISTNG 00071\*00096 00099 00832 02079 02083  
F9CB LOAD 01774\*02028  
EBC0 LPINIT 00090\*00366  
FBA5 MABR 01692 02051\*  
F6EE MATCH 01353 01375\*  
FB91 MBEG 00165 02046\*  
F481 MCHG 00926\*00982  
F4BC MCHG1 00939 00955\*00964  
F4B8 MCHG11 00953\*00966  
F4E3 MCHG13 00968 00976\*  
F51F MCHG21 00977 01006\*  
F529 MCHG25 01007 01012\*  
F4FB MCHG27 00989\*01013  
F4D5 MCHG3 00932 00967\*  
F4C8 MCHG5 00938 00961\*  
F499 MCHG7 00936\*00970  
F4E9 MCHG9 00947 00954 00979\*01010 01016  
F47F MCHNG 00924\*01978  
FBC1 MCKS 01874 02055\*  
FBBA MCLV 01760 02054\*  
F592 MCMND 01114\*01984  
F3DE MDOS 00792\*02032  
E800 MDOSE 00094\*00794  
FB96 MEND 00170 02047\*  
F098 MERGE1 00233\*00236  
F094 MERGEH 00221 00230\*00864  
FB9B MERR 00973 02048\*  
FBE2 MHDR 00400 02061\*  
FBB4 MHLT 01700 02053\*  
FBDC MMSK 01116 02060\*  
FFE5 MODE 00710 01468 01477 01504 01576 01963 02148\*  
FBAC MPAR 01658 02052\*  
FCFE MPIAAC 00083\*01234 01649

FCFC MPIAAD 00081\*01436 01484 01507 01650 01892 01901 01964 02071  
 FCFI MPIABC 00084\*01322 01518  
 FCFD MPIABD 00082\*00085 01522  
 FBC7 MSC 01796 02056\*  
 FB40 MSWI 01587 02050\*  
 F501 MUPDT 00978 00991\*01008 01014  
 F51C MUPDT3 00993 00994 00998 01003\*  
 F519 MUPDT5 00946 01002\*  
 FBCC MVHD 01835 02057\*  
 F768 NCMD 01446\*01985  
 F774 NCMD1 01448 01453\*  
 F777 NCMD3 01452 01454\*  
 F94F NMI11 01703 01708\*  
 F958 NMI13 01705 01712\*  
 F974 NMI15 01717 01723\*  
 F97C NMI17 01720 01726\*  
 F971 NMI19 01721\*01725  
 F909 NMI21 01663 01665 01675\*  
 F920 NMI23 01674 01685\*  
 F924 NMI3 01676 01678 01688\*  
 F936 NMI7 01691 01696\*  
 F945 NMI9 01699 01703\*  
 F8C3 NMISRV 00687 01645\*  
 F1D7 OCHAR 00373 00386 00416 00448 00461 00469\*00488 00490 00516 00558 01782  
 F1D9 OCHAR1 00471\*00473  
 F4B0 OCMND3 00940 00948\*  
 F4B6 OCMND5 00942 00944 00949 00952\*  
 F4AA OCMND7 00945\*00951  
 0000 ONEK 00068\*00071  
 F0C2 OUT2H 00251 00257 00283\*  
 F0C4 OUT2H1 00285\*00501 01002 01856  
 F0A9 OUT2HS 00115 00256\*  
 F0A7 OUT4HS 00116 00250\*  
 F128 OUTCH 00271 00272 00372\*00590  
 F13D OUTCH1 00380 00382\*00387 00391  
 F150 OUTCH3 00384 00392\*  
 F148 OUTCH5 00377 00389\*  
 F0A5 OUTHEX 00244\*00965 00981 01854 01921  
 F458 PAR2 00886\*00898  
 F464 PAR2A 00897\*00925 01286 01302 01474  
 F461 PAR2E 00840 00862 00879 00891\*  
 F621 PARIT1 01225 01235\*01245  
 F61C PARITY 01232\*01995 01997  
 F475 PARN 00914\*01284 01300 01395 01447 01465  
 F47E PARN1 00917 00919\*00979  
 F5A5 PCCMND 01131\*02009  
 FB45 PCMDT 00721 02003\*  
 F708 PCMND 01394\*01986  
 F5A8 PCMND1 00168 00173 01126 01133\*01163 01170 01177 01205  
 F5A9 PCMND2 01134\*01185  
 F5CB PCMND3 01139 01153\*  
 F5C5 PCMND4 01141 01151\*  
 F5D1 PCMND7 01138 01156\*01204  
 F5CF PCMND8 01155\*  
 F5C8 PCMND9 01143 01152\*  
 F6A2 PCMTCH 01326\*01399 01410  
 FF43 PCOUNT 00846 00853 00855 00887 00915 00929 01070 01086 01137 01343 01352 01733 01794 02131\*  
 F03B PCRLF 00117 00153\*

F02D PDATA 00119 00136\*  
 F02F PDATA1 00121 00142\*00148  
 F4DD PERR 00757 00952 00973\*01000  
 FA53 PINFO 01747 01838 01852\*  
 FA5E PINFO1 00928 01840 01856\*  
 F153 PNCH 00398\*02031  
 F156 PNCH1 00400\*00406  
 F19D PNCH11 00428 00430\*  
 F1A8 PNCH13 00435\*00437  
 F1B8 PNCH15 00420 00423\*  
 F161 PNCH3 00404\*00408  
 F187 PNCH5 00421\*00422  
 F18E PNCH7 00424\*00440  
 F19B PNCH9 00429\*  
 F1CD PNOFF3 00452\*00698  
 F17A PNON1 00412 00415\*  
 FF1F PRACCA 01043 01270 02109\*  
 FF20 PRACCB 01058 02110\*  
 FF21 PRCC 00668 01094 01273 01482 01531 01639 01661 01683 02111\*  
 FF1E PRDPR 00666 01101 01273 01274 01481 01547 02108\*  
 F7D1 PREP 01432 01503\*02070  
 F7D8 PREP1 01480 01505 01507\*  
 F7EE PREP3 01476 01509 01515\*  
 FA64 PRINTR 01757 01804 01860\*  
 FBD7 PRM1 00708 01622 02058\*  
 FAA9 PRNT 01909\*02030  
 FAC7 PRNT1 01923\*01932  
 FAED PRNT11 01938 01941\*  
 FAE1 PRNT3 01935\*01944  
 FABF PRNT5 01920\*01949  
 FAEF PRNT9 01940 01942\*  
 FF16 PRPCH 01132 01242 01268 01326 01416 01417 01541 01581 01596 01603 01605 01723 02100\*  
 FF18 PRURH 01125 01274 01275 02102\*  
 FF1C PRXRH 01169 01268 01269 02106\*  
 FF1A PRYRH 01176 01269 01270 02104\*  
 F44F PSH1 00850 00878\*00987 01472 01918  
 F4F3 PSHQ 00901 00969 00984\*01151  
 F7C5 PSHRG1 01495\*01498  
 F7CE PSHRG2 01441 01500\*  
 F7BF PSHRGS 01491\*01597 01636  
 FOAB PSPACE 00123 00262\*  
 F202 PUNT1 00436 00499\*  
 F206 PUNT3 00501\*00506  
 F216 PUNT5 00450 00503 00509\*  
 F000 PWRUP 00103\*01529 02156  
 FFE6 Q 00633 00985 01191 02149\*  
 F5E8 QCMMND 01190\*01987  
 F693 RBKPT 01316\*01608  
 F69C RBKPT1 01318 01321\*  
 F69E RBKPT3 01322\*01386 01388  
 F546 RCHNG 01035\*01045 01134  
 F622 RCMND 01240\*01712 01988  
 F625 RCMND1 01242\*01614  
 F62B RCMND2 01244\*01255 01257  
 F637 RCMND3 01247 01250\*  
 F643 RCMND4 01253 01256\*  
 F2A4 RDBYT1 00609 00611\*  
 F289 RDBYTE 00578 00581 00596\*

F22B RDIN 00535\*00538 00561 00570 00615  
 F2A5 RDINHX 00597 00603 00614\*  
 F244 RDON1 00552 00557\*  
 F232 RDON3 00541\*00560 00588  
 F27A RDROFF 00587\*00699  
 F239 READR 00549\*01620 01861  
 F24C READR1 00555 00561\*00568 00576  
 F255 READR3 00564 00567\*  
 F269 READR5 00573 00577\*  
 F27E READR6 00589\*00653 00654  
 F271 READR7 00581\*00583  
 F285 READR8 00591 00592\*  
 F23E READR9 00553\*00566  
 F6D9 REMOVE 01359 01364\*  
 F309 RENTER 00671\*01064  
 F313 RENTR1 00677\*01694  
 F346 RENTR2 00323 00705\*00743 00758 00764 01149 01589 01727  
 F31D RENTR4 00683\*00686  
 F8B5 RINT 01578 01591 01635\*01666 01680 01708  
 F89A RMSG 01588 01611\*01693  
 F64F RTAB 01243 01266\*  
 FBFB4 RTN1 02070\*02075  
 FBFB RTNUSR 02073\*  
 FCFD SBIT 00085\*00541 01677  
 F6F5 SBKPT 01381\*01401 01475  
 F707 SBKPT1 01383 01389\*01402 01406 01408 01411 01450 01464  
 F567 SBYTE 00992 01048 01069\*  
 FB0E SCMD 00724 01976\*  
 F5D2 SCMND 01161\*02010  
 F3CA SEC 00772\*00808 00822  
 F1E5 SETUP 00409 00442 00479\*  
 008C SKIP2 00087\*01523  
 FF02 SPEED 00411 00449 00551 02088\*  
 FF22 SPSAVE 00674 01162 01275 01417 01483 01637 01667 01671 02112\*  
 F9AF SRCH 01756\*01758 01759 01764 02027  
 F9B6 SRCH3 01760\*01768  
 FF90 STACK 00673 02143\*  
 F2AA START 00103 00621\*  
 FA99 STORE 00995 01223 01364 01500 01672 01828 01891\*  
 F812 SWAP 01492 01540\*01640  
 F819 SWAP2 01544\*01548  
 F817 SWAP3 01543\*01549  
 003F SWI 00086\*01363 01368  
 F871 SWI11 01588\*01623 01660 01701  
 F876 SWI13 01575 01577 01591\*  
 F86C SWI17 01586\*  
 F87D SWI21 01594\*01710  
 F867 SWI3 01560 01565 01584\*  
 F8A7 SWI5 01570 01622\*  
 F853 SWI9 01575\*01585  
 F827 SWISRV 00689 01555\*  
 F600 TCMND 01210\*01989  
 FF12 TEMPA 02097\*  
 FF14 TEMPB 01948 02098\*  
 FF41 TEMPc 01792 01826 01841 02129\*  
 FF57 TEMPd 01793 01833 01837 02135\*  
 FF06 TEND 01211 01724 02091\*  
 83FF TOPTGT 00076\*00625

F1EE TPUN 00419 00433 00444 00486\*  
 F200 TPUN1 00421 00493 00495 00496\*  
 FF3D TRACE 01396 01455 01463 01613 01704 01713 01714 01716 01726 02125\*  
 F406 TRMTB 00820 00828\*  
 F67C UCMND 01299\*01990  
 F68A UCMND1 00694 01307\*  
 F68D UCMND3 01308\*01310  
 F692 UCMND5 01288 01292 01311\*  
 F688 UCMND7 01301 01306\*  
 F764 UPSH 01419 01421 01423 01425 01427 01429 01439\*01496  
 F5A0 URCMND 01124\*02012  
 FB05 USER 01521 01961\*02033  
 FB07 USER1 01957 01963\*  
 FA75 VCKSM 01806 01871\*  
 FA98 VCKSM3 01865 01873 01882 01886\*  
 FA7A VCKSM5 01874\*01884  
 F663 VCMND 01282\*01991  
 F677 VCMND1 01292\*01304  
 F9D5 VERF 01766 01776 01790\*02029  
 F9E0 VERF1 01796\*01802  
 FA07 VERF11 01809 01817\*  
 FA14 VERF13 01823\*01848  
 FA21 VERF15 01827 01829\*  
 FA45 VERF16 01831 01845\*  
 FA47 VERF17 01842 01846\*  
 FA37 VERF19 01834 01838\*  
 F9F1 VERF3 01800 01803\*01814  
 F9F3 VERF7 01804\*01805 01807 01811 01850  
 F982 WCMND 01732\*01992  
 F994 WCMND3 01738 01741\*01750  
 F9A5 WCMND5 01746 01748\*  
 F9D2 WCMND7 01734 01740 01784\*01795  
 FF39 WMSK 01118 01745 02121\*  
 F003 XBEGEN 00104\*00399 01115 01220 01910  
 F006 XCBCDH 00106\*00861  
 F009 XCHEXL 00108\*  
 F00C XCHEXR 00109\*  
 F5D7 XCMND 01168\*02013  
 F64C XCOUT 01248 01261 01263\*  
 F21F XCRLF 00507 00519\*  
 F00F XINADD 00110\*  
 F012 XINCH 00112\*  
 F015 XINCHN 00113\*00185  
 F01B XOUT2H 00115\*01046 01254 01927  
 F01E XOUT4H 00116\*01135 01256 01348 01877  
 F018 XOUTCH 00114\*00146 00155 00157 00159 01263 01942  
 F021 XPCRLF 00117\*00137 00519 00980 01241 01853 01920  
 F027 XPDAT1 00121\*00713 00974  
 F024 XPDATA 00119\*00166 00171 00401 00702 00709 01117 01612 01761 01797 01836 01867 01875  
 F4F8 XPSH 00987\*  
 F4F6 XPSH1 00986\*01037 01327 01736 01822  
 F02A XPSPAC 00123\*00740 00956 01857 01933  
 FFE5 XSTACK 00622 00672 00706 01557 01647 02147\*  
 F5DC YCMND 01175\*02014  
 F105 ZCMND 00348\*01993  
 F122 ZCMND1 00356 00364\*  
 FF68 ZEND 00693 02140\*  
 FF37 ZFLAG 00273 00277 00349 02119\*



## APPENDIX C

### PERIPHERAL REQUIREMENTS FOR EXORciser OPERATION

#### GENERAL

The EXORciser depends on the concept that a data terminal be used to "communicate" with the debug system and with the system under development.

All address or data inputs are entered via the keyboard, and all status or other results are printed (or displayed) in ASCII-Hex characters. This exchange of information is easier to interpret and much faster than could be done with lights or switches. The operator communicates in bytes and words, rather than in 1's and 0's. In addition to the keyboard and printer functions, however, is the need to save programs and be able to re-enter them into memory.

The EXORciser uses the ASCII (American National Standard Code for Information Interchange) communications protocol, which defines the serial asynchronous transmission and reception of characters with start-stop and parity bits appended, as shown in Figure 1. This figure shows the timing for baud rates of 10, 15, and 30 characters per second (cps), but the same relationships apply at any higher speed. The EXORciser has a crystal controlled communication baud rate clock with selectable positions from 110 to 9600 baud. Each character consists of:

- 1 Start bit
- 7 Data bits (ASCII) - See Figure 2.
- 1 Parity bit
- 1 or 2 Stop bits (2 at 110 baud)

In most terminals, the parity bit is sent by the keyboard, and is selected to be EVEN, ODD, ONE, or ZERO by strapping of switching options. The EXORciser ignores the parity bit from the keyboard, and normally transmits a ZERO (0) as the parity bit. Most terminals, including the TTY printer, also ignore this function. If a terminal requires a specific type of parity, it can be accommodated by revising the program in PROM U38 on the DEbug Module. Parity checking is considered unnecessary in this application.

#### TERMINAL FUNCTIONAL REQUIREMENTS

Four basic functions must be performed by the terminal/peripheral equipment in systems development work with the EXORciser. They are:

- a. Keyboard entry
- b. Printer (or Display)
- c. Program loading (into memory)
- d. Program saving (storage)

Many operations require a hard copy, such as a program listing during an assembly. Some terminals, such as the teletype and TI 733ASR, provide all four functions in one unit, and use tape to handle the program storage. Since both of these terminals use the ASCII codes (DC1, DC2, DC3, and DC4) to control the tapes, they work compatibly with the EXbug control program, as well as with the Editor/Assembler tapes available for software development.

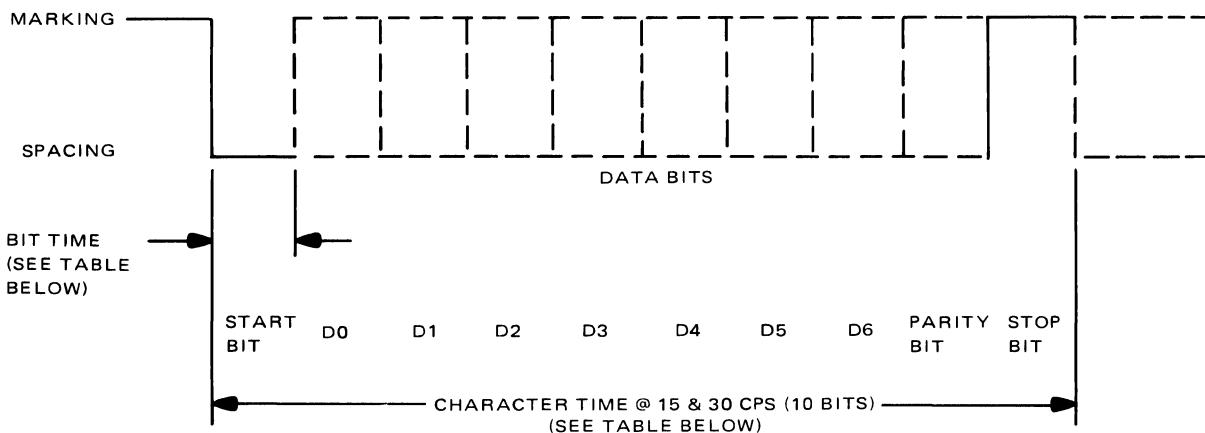
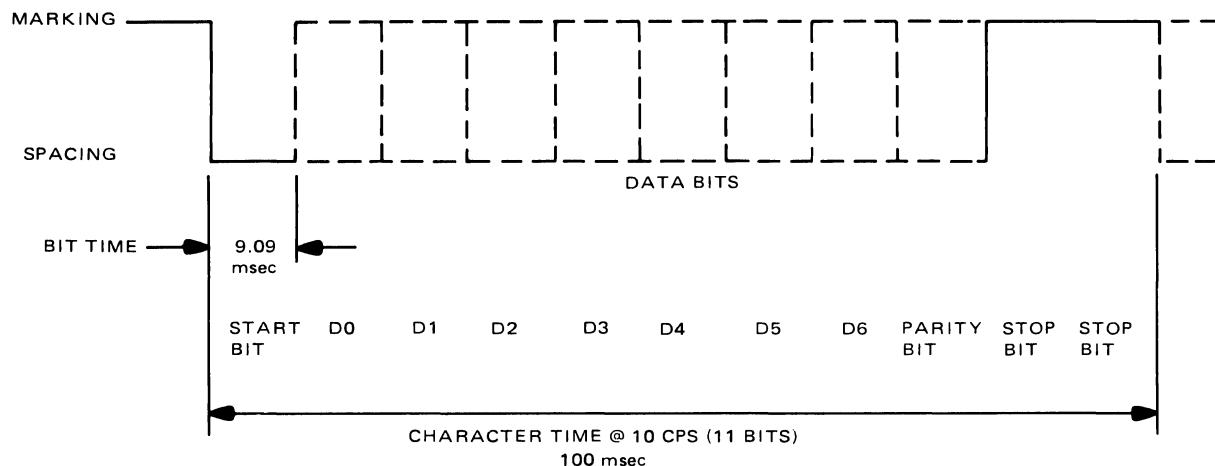
Other terminals have various combinations of these functions, but two or more peripheral devices are usually needed to meet the complete requirements. For example, HP2644 and HP2645 CRT terminals have dual tape mechanisms, but require the addition of a line printer to be complete. Also, the HP terminals do not use standard ASCII codes for tape control. (A Motorola User Library program is available to interface them.)

Tape storage, particularly cassette tape, is quite acceptable to many users, because the loading time has been reduced from 25 minutes for a TTY to three or four minutes with cassettes (for a 6-kilobyte program). The desire for still higher speeds exists and has been provided by the EXORdisk Floppy Disk System.

Since users of these more capable storage systems usually deal in large programs, it is frequently advisable to add a high speed line printer to the system. This is accompanied by using one of the standard interface modules which plug into the EXORciser. Any "Centronics interface" printer can be directly connected.

When data storage and line printer requirements are covered by other peripherals, a CRT terminal such as the EXORterm 100 is advantageous because of its higher display speed (up to 9600 baud), enhanced editing, and the elimination of the use of paper (see the EXORterm 100 User's Guide for more details).

**110 BAUD  
SERIAL ASCII DATA TIMING**



BAUD RATE	150	300
CHARACTERS/SEC	15	30
BIT TIME (msec)	6.67	3.33
CHARACTER TIME (msec)	66.7	33.3

$$\text{BIT TIME} = \frac{\text{SEC}}{\text{BAUD RATE}}$$

FIGURE 1. 110, 150, and 300 Baud Serial ASCII Data Timing

BITS 4 THRU 6		-	0	1	2	3	4	5	6	7
BITS 0 THRU 3	0	NUL	DLE	SP	0	@	P		p	
	1	SOH	DC1	!	1	A	Q		q	
	2	STX	DC2	"	2	B	R		r	
	3	ETX	DC3	#	3	C	S		s	
	4	EOT	DC4	\$	4	D	T		t	
	5	ENQ	NAK	%	5	E	U		u	
	6	ACK	SYN	&	6	F	V		v	
	7	BEL	ETB	,	7	G	W		w	
	8	BS	CAN	(	8	H	X		x	
	9	HT	EM	)	9	I	Y		y	
	A	LF	SUB	*		J	Z		z	
	B	VT	ESC	+	:	K	-		k	
	C	FF	FS	.	<	L	/		/	
	D	CR	GS	-	=	M	\		m	
	E	SO	RS	.	>	N	↑		o	≈
	F	SI	US	/	?	O	←			DEL

SEND A 7 BIT ASCII CHAR. "H"

EVEN PARITY — 2 STOP BITS

$$H = 48_{16} = 1001000_2$$

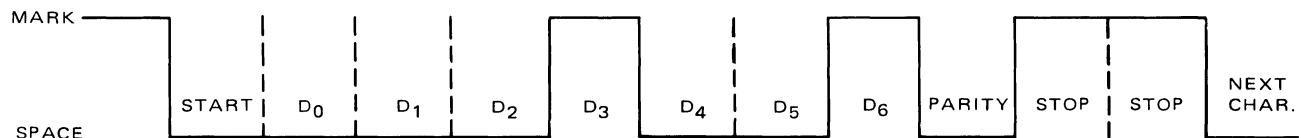


FIGURE 2. ASCII Example

## APPENDIX D

### USE OF THE ASR33 TELETYPEWRITER WITH EXORciser

#### GENERAL

The ASR33 teletypewriter is one of the standard data terminals for use with computers. It uses the ASCII protocol, as described in Appendix C. The basic TTY is designed to interface via a dc current loop. The 20 mA option is required for use with the EXORciser.

#### FULL DUPLEX 20 mA CURRENT LOOP CONNECTIONS

In order to use the 20 mA neutral current loop interface with EXORciser, the RS-232C interface signals must be translated to those of the current loop interface. This can be done easily by using Micromodule 11 (M68MM11). The 48-inch flat ribbon cable with keyed edge connector is plugged directly into connector P4 on the DDebug Module, after removing the connector and cable from the rear panel of the EXORciser chassis. The connections on the terminal, as well as the connections to the Micromodule, are all described in the Micromodule 11 User's Guide. Additional comments are provided here to aid the user in understanding this interface.

The teletype keyboard is normally wired to provide an EVEN parity bit (bit 7). It can be changed to be always ONE (1) or always ZERO (0) for each character. The EXORciser ignores the parity bit so it doesn't matter which type the keyboard transmits. The EXORciser normally sends each character to the terminal with a zero (0) in the parity bit position; but since the TTY does not test parity, the bit is ignored.

The ASR33 teletypewriter should be equipped with an automatic tape reader to be fully compatible with the EXbug control program. The Co-Resident Editor and Assembler tape programs cannot be used unless an automatic reader is provided. The lack of an automatic punch requires either careful manual operation or that a third pass of the source tape be made to generate an object tape (only) with the punch ON.

The TTY can be equipped with either an automatic or manual reader. The automatic reader will start reading tape when a "DC1" control character (hex 11) is received, and will stop when a DC3 (hex 13) is received. The manual reader can be converted by means of a relay to operate automatically, if desired. In this case, the reader is started and stopped with a dc output signal provided by the EXORciser. Older teletypes can also be easily updated by local TTY service organizations to provide the automatic punch or readers.

#### AUTOMATIC READER/PUNCH CONATROL MODIFICATION

The EXORciser is designed to work with a TTY terminal having automatic reader/punch control, or a manual TTY terminal modified for automatic reader/punch control. The following paragraphs discuss modifying a manual TTY terminal for automatic reader/punch control.

## Components Required

Table 1 identifies the parts required to modify a TTY terminal.

TABLE 1. Automatic Reader/Punch Modification Parts List

COMPONENT	QUANTITY	DESCRIPTION
Reed Relay	1	12 V, 600-ohm coil, Potter Brumfield Part Number JR-1005 or equivalent
Resistor	1	470 ohm, 1/2 watt
Capacitor	1	0.1 uf, 600 Vdc
Component Board	1	Vector Board (2.5 x 3.8 inches)

## Modification Procedures

- a. Install the resistor, capacitor, and relay on the component board, as depicted in Figure 1.

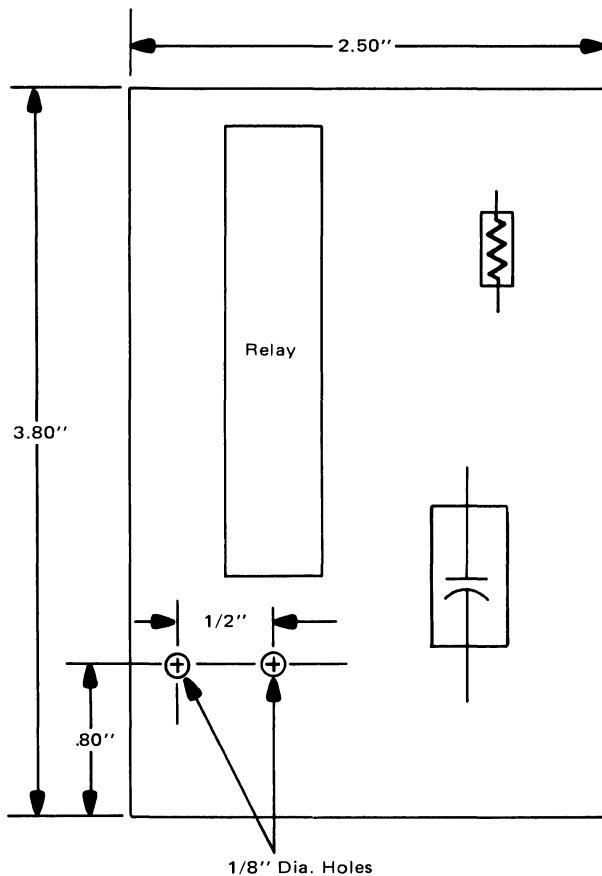


FIGURE 1. Component Board

- b. Mount the component board onto the TTY terminal, using the holes in the terminal mounting plate. Refer to Figure 2 for the component board location.



FIGURE 2. Component Board Location

- c. Refer to the schematic diagram in Figure 3; connect the relay, capacitor, and resistor into the terminal circuitry. Wire A may be spliced to the brown wire near the connector plug. Both the LINE and LOCAL lines must be connected to the MODE switch, as shown.
- d. For operation, the reader switch is left in the ON position. The added relay controls the starting and stopping of the tape.

**CAUTION**

IN THE SYSTEM, MAKE CERTAIN NO VOLTAGES FROM THE TERMINAL ARE ACCIDENTALLY CONNECTED TO THE RELAY PRIMARY WINDINGS. SUCH A VOLTAGE MAY DAMAGE THE DEBUG MODULE AND POWER SUPPLY.

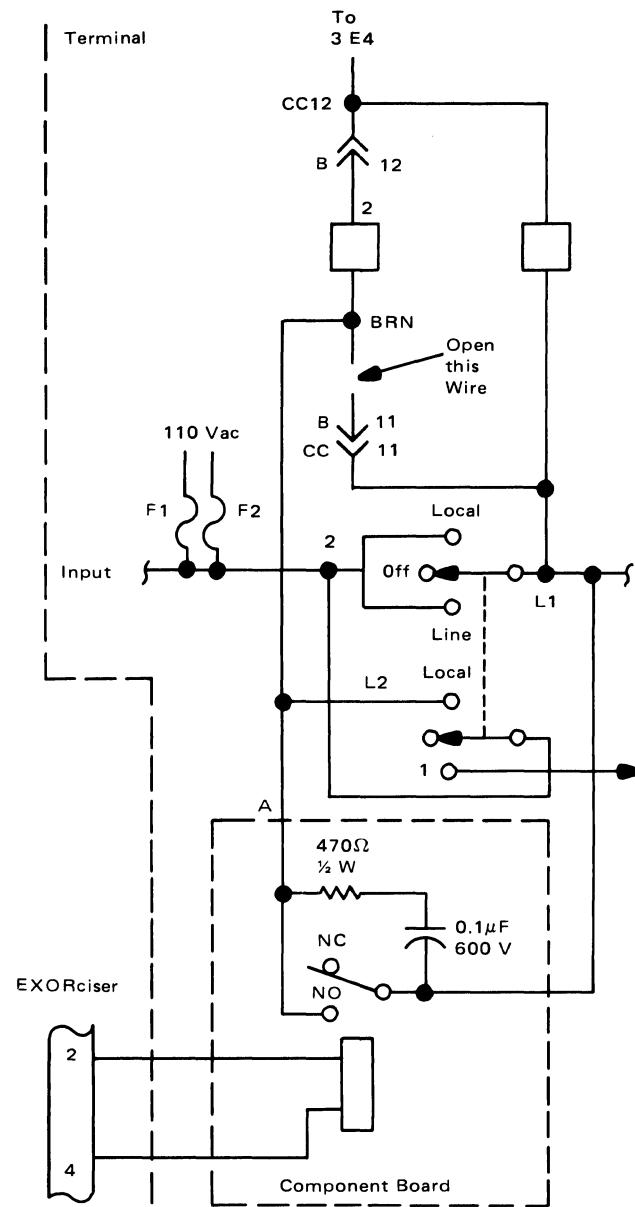


FIGURE 3. TTY Terminal Schematic Incorporating Automatic Reader Control

## APPENDIX E

### RS-232C STANDARD

The RS-232C standard adopted by the Electronic Industries Association (EIA) uses the following terminology to describe signal levels:

Binary state	"1"	"0"
Signal Condition	MARK	SPACE
Voltage	negative	positive
Paper Tape	hole	no hole

For control circuits:

Control function	OFF	ON
Voltage	negative	positive
Binary state	"1"	"0"

The specific RS-232C standard defines any voltage between -3 and -15 as a binary "1", and any voltage between +3 and +15 as a binary "0".

The connector used for an RS-232C equipped terminal is standardized and made by several manufacturers. It should be a Cinch DB-25P or equivalent.

The pin connections and signal characteristics for the RS-232C standard are listed in Table 1. These signals are available at the 25-pin connector located on the rear panel of the EXORciser chassis.

TABLE 1. RS-232C/EXORciser Interconnections

PIN NUMBER	SIGNAL MNEMONIC	SIGNAL NAME AND DESCRIPTION
1	POWER GND	POWER GROUND - Chassis ground. This line provides a safety ground connection to the RS-232C compatible terminal.
2	RX DATA	RECEIVE DATA - This line receives the input from an RS-232C compatible terminal (keyboard).
3	TX DATA	TRANSMIT DATA - This line transfers data to an RS-232C compatible terminal (printer).
4		Not used.
5	CTS	CLEAR TO SEND - This line is ON (plus) when an RS-232C data terminal is connected to the EXORciser, and DTR is ON.
6	DSR	DATA SET READY - This line is ON (plus) when an RS-232C data terminal is connected to the EXORciser, and DTR is ON.
7	SIGNAL GND	SIGNAL GROUND - This line provides a common signal connection to the RS-232C data terminal.
8	DCD	DATA CARRIER DETECT - This line is ON (plus) when an RS-232C data terminal is connected to the EXORciser, and DTR is ON.
9-19		Not used.
20	DTR	DATA TERMINAL READY - This line from the RS-232C data terminal indicates that the data terminal is ready, when ON (plus).
21-25		Not used.

## APPENDIX F

### TI TERMINAL DESCRIPTION

The TI 733ARS terminal is functionally equivalent to the ASR33 teletypewriter, but provides 10, 15, or 30 characters per second (CPS) operation. When equipped with the ADC (Automatic Device Control) Module, the tape cassettes respond to the same ASCII commands (DC1, DC2, DC3, or DC4) as used by the TTY. Proper automatic operation with an EXORciser up to 30 CPS, is thus provided. To get maximum performance, however, it is suggested that the Remote Device Control (RDC) Module and 1200 baud option be used. This provides the same ASCII cassette control features as the ADC Module does, but adds several more which are utilized to realize the advantages of the 1200 baud operation.

The added commands are:

DLE,: RDC on command

DLE,9 Printer ON

DLE,7 Cassette Block Forward

DLE,0 Printer OFF

These commands are used to allow turning the printer ON and OFF, when necessary to avoid garble. The DLE,7 command was found necessary to provide a "Block Forward" instead of DC1, when reading the tape. DLE,: is used during initialization. TI has numerous options on the RDC Module, so it is necessary to specify the (-3) version. Most of the difficulties experienced are due to the wrong options being used. When a terminal is received, it normally inhibits the printer at all times when 1200 baud is used. This need not be done with the EXORciser because of the nulls provided, so this option should be changed. This is done by removing a 10 ohm 1/4 W resistor from the center of the 1200 baud receiver card (Green Ejector). Also, the DIP package toggle switches should be as specified in the TI manual (1 through 6 are ON, and 7 and 8 are OFF).

Revision K of the terminal control card will not work properly, but prior and later revisions will work properly. The TI field service stations all have been provided with instructions on the use of this terminal with an EXORciser. If the user has any difficulty, he should call Motorola for assistance.

A TI 733ASR terminal, which is equipped for 1200 baud and RDC options, can be easily modified for 2400 baud operation. Two printed circuit runs on the 1200 baud receiver card are cut (E2 to E4 and E1 to E3), and two jumpers are added (E1 to E2 and E3 to E4). When done, the terminal will transmit and receive at 2400 baud when the HI-LO SPEED switch (located near the power switch) is in the HI-SPEED position. The terminal performance will not be affected when this switch is in the LO-SPEED position. (The baud rates will be determined by the switch inside, as before.)

The EXbug program has features which work with the TI terminal to allow Assembly and Editing at 1200 and 2400 baud. The higher rates are used between the units so that the cassettes will read or record at these speeds. Since the printer

cannot exceed 300 baud, the EXORciser EXbug Firmware, for example, inserts three nulls between each character (every fourth character is printed) and 23 nulls after each CR (at 1200 baud) to allow time for the carriage to return. At 2400, it is 7 and 47 nulls, respectively. This software mode is selected by typing the appropriate ;K command when starting the EXORciser. This technique will work with any mechanical terminal which ignores nulls. Some terminals will not. These numbers can be used with any terminal that takes 200 milliseconds to return. If a slower terminal is used, the ;K command of EXbug can be used to insert any number of nulls. (See Chapter 3 for more details on EXbug.)

## APPENDIX G

### EXECUTIVE MAP/USER MAP INTERFACE

#### INTRODUCTION

The dual map feature of EXORciser is provided primarily to ease program development by giving the user a full 64K memory map without sharing space with EXbug, MDOS interface, or other peripherals, while providing complete EXbug debug capabilities. However, the dual map also presents the possibility of running a program in the User map with the I/O performed in the Executive map. These last two capabilities are discussed here, along with the procedures for transferring information and control between the two maps.

Since the User map was developed as a separate 64K map for testing applications programs, it was not given access to the Executive map. The Executive map has the ability to store, fetch, and start execution at any location within the User map. Control is returned to the Executive map from the User map by means of interrupts. Since the Executive map also contains all EXbug I/O, the Executive map normally has control of the EXORciser.

The two maps are provided by splitting the VMA signal into two other signals: VUA and VXА. VUA enables the User map, while VXА enables the Executive map. These two signals are generated by the DEbug Module. When the DEbug Module is in the Single Map mode, valid addresses less than F000 generate VUA, while valid addresses equal to or greater than F000 generate VXА. In the Dual Map mode, VUA and VXА are generated from a shift register controlled by a PIA. This PIA is addressable only from the Executive map.

#### VUA GENERATION -- DUAL MAP MODE

Figure 1 is a simplified diagram of the VUA/VXA generation circuitry for the Dual Map mode. As illustrated, the A peripheral register of the PIA determines the VUA/VXA pattern that will be generated. A pulse on the CA2 line loads the shift register and starts the sequence that controls access to the User map. The A control register generates the pulse on the CA2 line. To accomplish this, the A control register contains either \$2C if memory parity error interrupts are not enabled, or \$2D if enabled. Therefore, once the PIA A data register is properly set up, the User map can be repeatedly accessed by reading the PIA A data register for each access. This is accomplished by using load, test, and compare instructions in the Executive map.

Figure 2 shows typical instruction sequences for accessing and passing control to the User map, along with the corresponding shift register cycles and PIA A peripheral register contents. The instructions TST and CPX are used to load the shift register, because they provide the correct timing and do not modify the processor registers except for the condition code. Other instruction sequences may be used to access the User map. However, the VUA pulse must occur when data is to be read from or written to the User map. The MC6809 Data Sheet contains a summary of cycle-by-cycle instruction operation. If other instruction sequences are used to access the User map, note that the pattern loaded in the PIA is transferred from the shift register on the third cycle after the PIA A register is read. Since the CLR instruction reads the location to be cleared before clearing, CLR should not be used to clear the PIA A peripheral register, as this

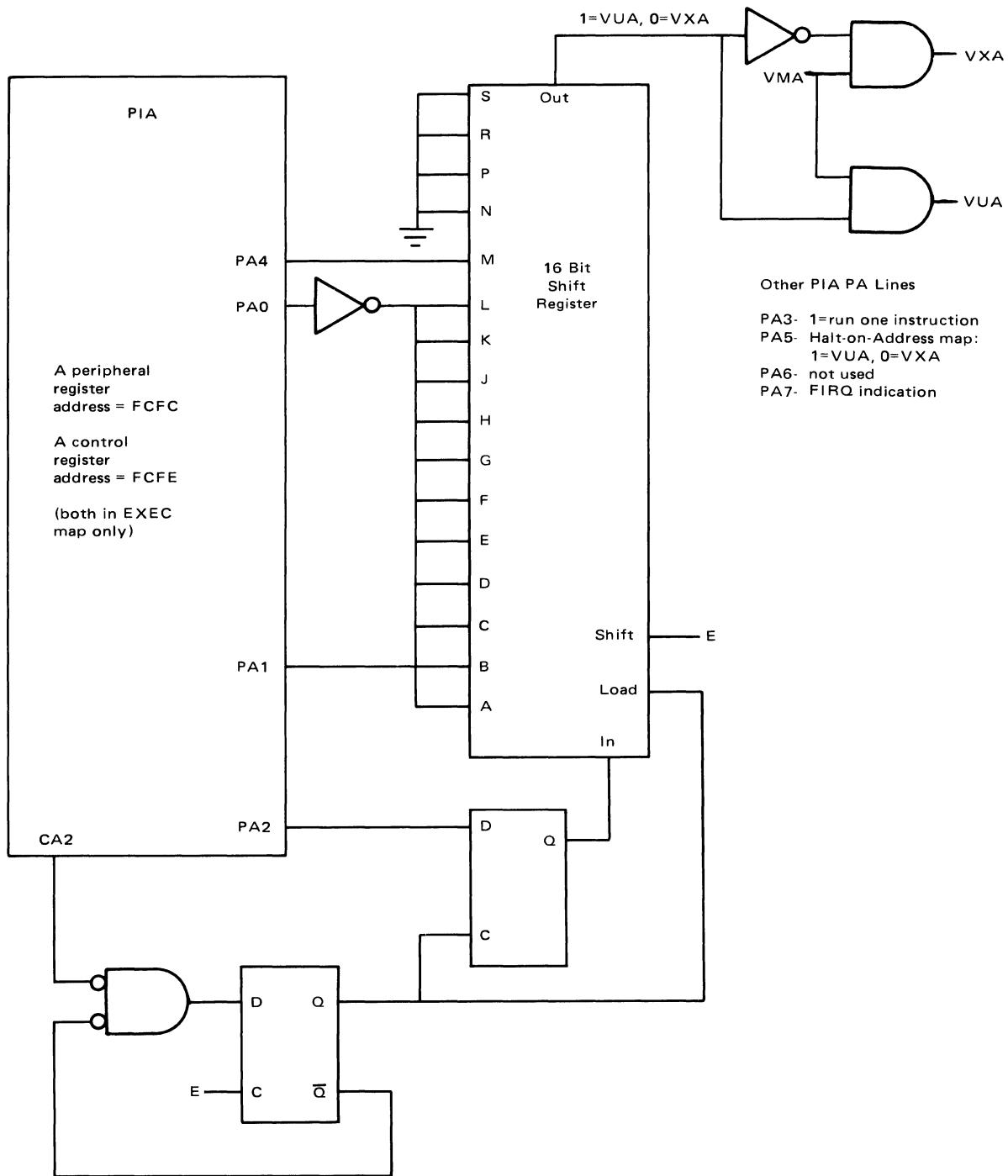


FIGURE 1. VUA/VXA Generation Circuitry - Dual Map Mode

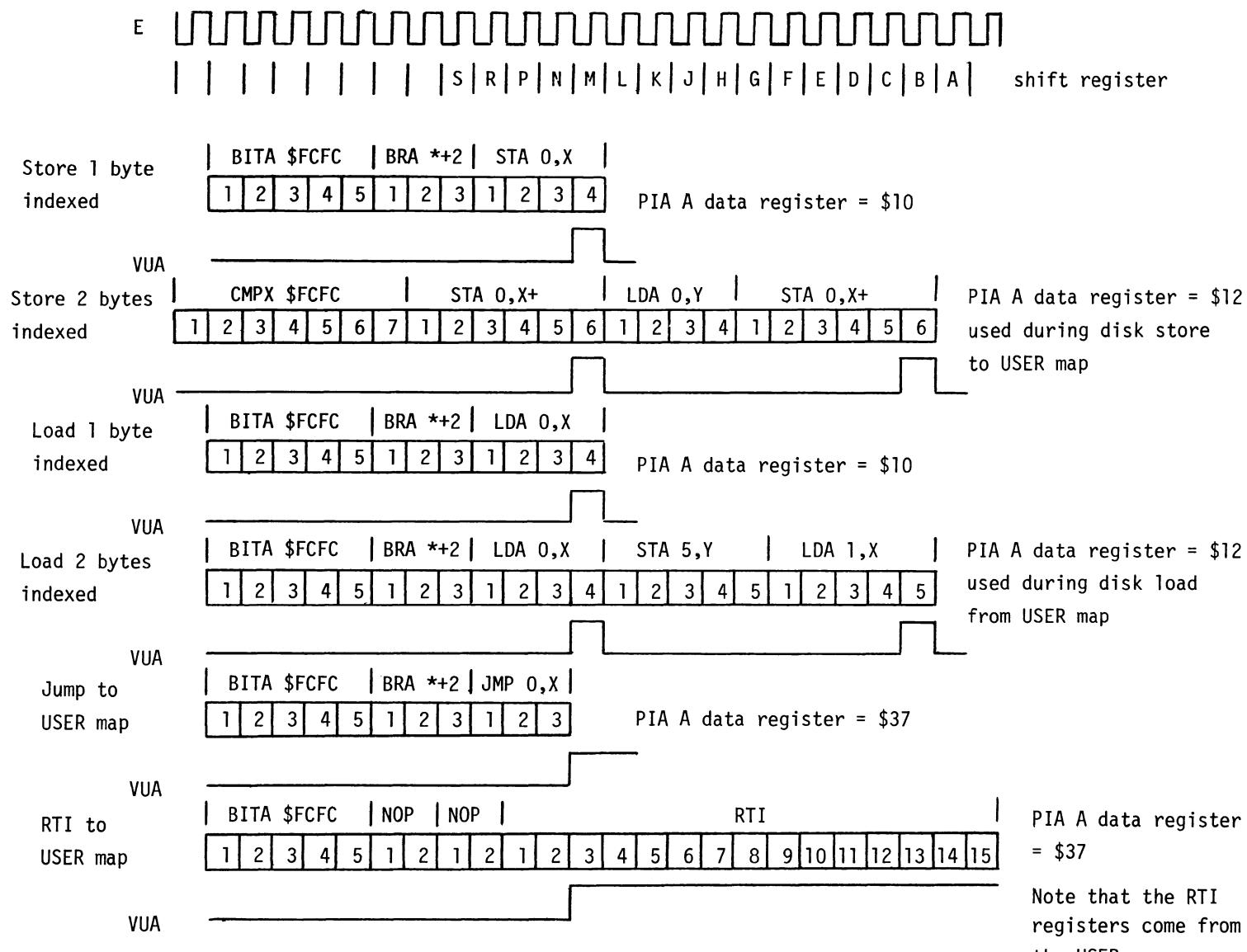


FIGURE 2. User Map Access Instruction Sequences

will cause the shift register to be loaded from the PIA before the PIA is cleared. The recommended procedure for clearing the A peripheral register is to first clear the accumulator and then store the accumulator contents in the PIA peripheral register.

Typically, the User map is accessed a byte at a time. In this case, the PIA A register is set up with the proper pattern initially, and does not require changing until program execution is to be passed to the User map. In all the instruction sequences illustrated here, except the RTI, the X index register contains the base address for the load, store, or execution. A non-zero index offset can be used in the sequences; however, the instruction sequence would have to be changed. A zero index offset is probably the most useful. The store-to-User-map sequence also requires that the data to be stored be in the appropriate accumulator before the sequence is started. The load and store sequences are useful in passing parameters between the maps under the control of the Executive map. The jump sequence is used to start programs in the User map, since it does not specify register contents. Return to the User map from service calls to the Executive map is accomplished using the RTI sequence.

#### RETURNING CONTROL FROM USER MAP TO EXECUTIVE MAP

Once a program is running in the User map, there are two controlled ways in which control can be returned to the Executive map. One way to return is an EXbug-generated NMI. EXbug-generated NMIs occur for abort, memory parity error, run-one-instruction, and halt-on-address. Restart will also return control to the Executive map if the restart map switch on the DDebug Module is in the Executive map position.

The controlled way to return to the Executive map from the User map is by an SWI. If this feature is enabled on the DDebug Module, executing an SWI in the User map will cause control to return to the Executive map. When the SWI is executed, the processor registers are first pushed on the stack in the User map; then, when the SWI vector is fetched, control is passed to the Executive map and the vector is read from FFFA, FFFB of the Executive map. For this type of operation, the user should replace the normal EXbug vector in the Executive map with the address of his own service routine. The "SWI" return to Executive map is enabled by the CB2 line of the PIA which controls the map control shift register. The address of the control register which determines the state of CB2 is FCFF in the Executive map. When this register contains \$37, the "SWI" return to Executive map is enabled, and the sequence of events described above will occur when an SWI is executed in the User map. When this register contains \$3F, the "SWI" return to Executive map is disabled, and SWI's are serviced in the User map. The EXbug E command can also be used to enable the "SWI" return to Executive map. If the E command is used, EXbug will put the appropriate value in the CB2 control register when the next program execution command (;P or ;G) is entered. Typically, the "SWI" return to Executive map is enabled initially using the EXbug E command. After program execution has begun, there is no need to change the "SWI" return enable.

#### SWI Nesting

For some uses of the SWI to return to the Executive map, it may be desirable to nest SWI's. That is, the routine that services the Executive return SWI might wish to execute SWI's of its own for other functions. These other SWI's would also be serviced in the Executive map. The problem that arises in this

situation is that the routine that responds initially to the SWI must know which map the SWI occurred in. Service of a User map SWI will probably require the saving of the stack pointer and then using a different stack pointer value, as well as setting the PIA A register for User map access. These operations will not be done if the SWI occurred in the Executive map. Also, the return from interrupt will have to be handled differently for a User map return than for an Executive map return. Return from an Executive map SWI can be handled with an RTI. However, return from a User map SWI requires restoring the stack pointer to the User map value and passing control back to the User map.

An indication of the map in which the SWI occurred is provided on the DEbug Module. The CB1 line of the map control PIA gets a low to high transition when a User map to Executive map change occurs. This shows up as the sign bit of the B control register being set. The B control register address is FCFF in the Executive map. This register normally contains \$37 or \$3F, depending on whether the SWI return to Executive is enabled or disabled. If the map change indication is set, the B data register at FCFD in the Executive map must be read to reset it. The map change flag is used on entry to the SWI service routine to determine if the stack pointer must be saved and the Executive map value used. To determine the proper actions for the SWI return, the map change indication can be pushed onto the stack. In the return sequence, the map change indicator is pulled off the stack to determine proper return action. The actual SWI service can be done in a subroutine that returns to the SWI return routine.

The flow chart in Figure 3 illustrates the SWI return sequence. The most general case of nested SWI's is covered in this flow chart. Asterisks in this chart mark steps that are not required if SWI's are not nested.

Another problem encountered in nested SWI's is the accessing of parameters from the Executive map SWI's. If the same routines are used as for accessing the User map, loading a value 00 in the PIA A register before the User map access sequence will cause the Executive map to be accessed instead of the User map. A value of \$30 will have to be restored to the PIA A data register before the User map is to be accessed again. Another solution is to use separate routines to obtain the Executive map parameters. These routines would read the Executive map directly without loading the shift register before reading or writing to memory. In this case, the value in the PIA will not have to be changed before accessing Executive map parameters, or restored after the parameters are accessed.

#### FUNCTION REQUEST SPECIFICATION

It is very likely that the User map will have to call the Executive map for more than one function. However, all SWI's are vectored to a single routine. The problem, then, is to communicate to the common SWI service routine the desired function, so that an appropriate subroutine can be called to accomplish that function. To solve this, a separate number can be assigned to each of the required functions. Then the common SWI service routine has to get the number of the desired function.

There are several different ways that the common SWI service routine can get the function number. In one method, the function number is put in the byte following the SWI. This does not require programmed set-up in the User map, but the SWI service routine must make a number of accesses to the User map to get the function number. The service routine must first get the program counter from User map stack. It then uses this address to get the function number from the User map. Finally, it must correct the program counter on the User stack so

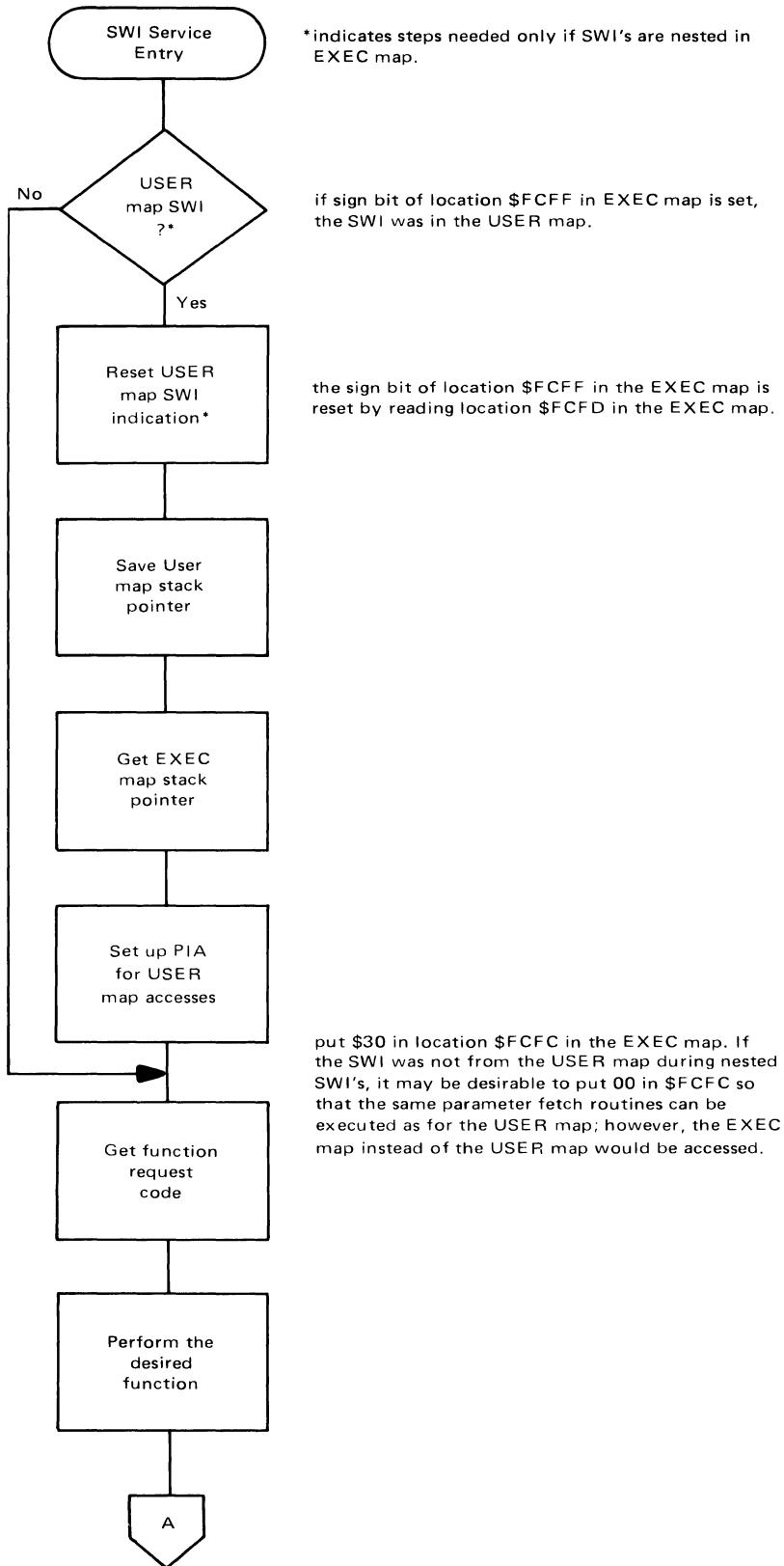


FIGURE 3. SWI Service Sequence (Sheet 1 of 2)

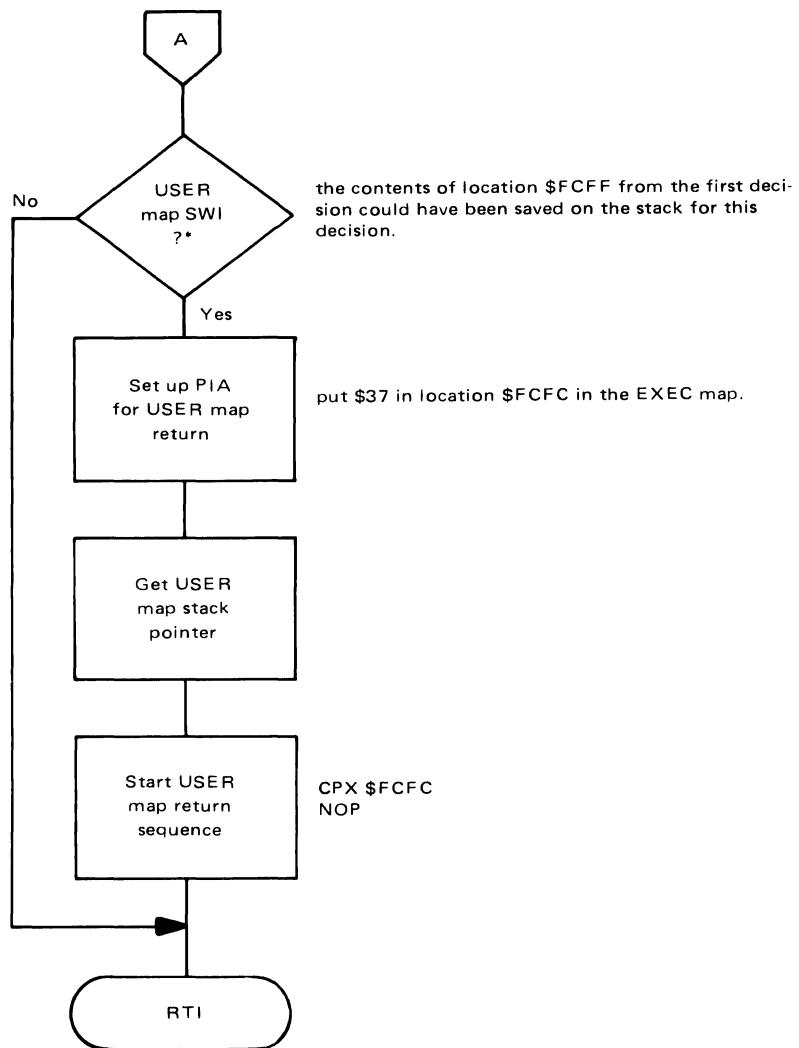


FIGURE 3. SWI Service Sequence (Sheet 2 of 2)

that the SWI return skips over the function number. Another method is for the User map program to always put the function number at a specific location that is known to the SWI service routine, and then execute the SWI. This requires some programmed set-up in the User map. However, the SWI service routine only needs to access the User map once to get the function number. Perhaps the best way to pass the function number to the SWI service routine is to put it in one of the processor accumulators before executing the SWI. Only a minimal amount of programmed set-up is required in the User map. Also, the SWI service routine does not have to access the User map to get the function number.

## Run Sequence

Once the program has been assembled or compiled, the next step is to load it into the system and test it. The following steps can be used to load the program from tape or MDOS. The only additional steps for MDOS are that MDOS must be started initially and must be restarted between steps a and b.

- a. Load the User map.
- b. Load the Executive map.
- c. Take over the Executive map SWI vector. Place the SWI service routine entry address in FFFA, FFFB.
- d. Enable User map SWI's to be serviced in the Executive map by entering a -1 for the EXbug E command value.
- e. Enter the User map mode by typing the USER command and a carriage return.
- f. Start the User map program using the addr;G command.

## Debugging

If the program does not work properly, the next step is to debug it. This is somewhat difficult since EXbug was not designed to debug in both maps at the same time. Also, since the EXbug SWI vector has been replaced by the user Executive map SWI vector, breakpoints cannot be used. The solution to the first problem is to debug each map separately. The Executive map routines can be tested by dummying up calls to them in the Executive map. EXbug Halt-on-Address and Trace functions can be used to verify the program operation. There is a precaution that must be observed while testing the Executive map routines in this mode. All locations where the map control PIA is set up to access or transfer control to the User map must be patched to store 00 in the PIA. This is required since the SWI's and data are in the Executive map.

After the Executive map program has been debugged, the User map program can be tested. Here again, the Trace and Halt-on-Address functions can be used to verify program operation. While testing the User map program, the Trace function cannot be used to trace through SWI's. During the testing of the User map program, it makes its normal calls to the debugged Executive map routines.

## Example

As an example of what is needed to run a program in the User map and have it use the Executive map for I/O, a program that was originally written for a single map environment was converted to the dual map system. The converted program runs in the User map and uses SWI's to transfer control to the Executive map for I/O.

The original program used five of the MIKbug I/O routines:

OUTEEE sends the character in the A accumulator to the system terminal.

PDATA1 prints a data string pointed to by the index register and terminated by an EOT to the system terminal.

INEEE	accepts a character from the system terminal and returns it in the A accumulator with bit 7 reset. The character is echoed to the system terminal.
OUTHR	converts the four least significant bits of the A accumulator to ASCII hex and sends it to the system terminal.
OUTS	sends a space to the system terminal.

In the original program, these labels were defined using EQU directives. A given routine was then called by a JSR to the appropriate label. To convert the program to dual map operation, the EQU definitions were replaced with the subroutines of the listing in Figure 4 and the program re-assembled. The subroutines of Figure 4 are all similar. The only difference is the unique I/O code assigned to each subroutine. Each subroutine saves the B accumulator, gets the I/O code in the B accumulator, and then executes an SWI to transfer control to the Executive map. Note that the B accumulator is used to pass the I/O code to the Executive map. On return from the SWI, the original contents of the B accumulator are restored and an RTS is executed to return from the I/O subroutine.

Figure 5 contains the listing of the Executive map SWI service routine. It uses three EXbug subroutines:

XOUTCH	sends the character in the A accumulator to the system terminal.
XCHEXR	converts the four least significant bits of the A accumulator to an ASCII hex character which is returned in the A accumulator.
XINCHN	accepts a character from the system terminal and returns it in the A accumulator with bit 7 reset. The character is echoed to the system terminal.

The XPDATA1 function is duplicated in the SWI service routine because the EXbug PDATA1 does not access the User map (which is where the string to be printed is). The EXbug XPSPAC was not used for OUTS because it was just as easy and a byte shorter to load the A accumulator with a space character and fall into a call to the XOUTCH routine. After the appropriate function is completed, the User map stack must be updated with the current register contents, since the registers are restored from stack when the RTI is executed. The PDATA1 routine updates the index register on the User stack to point to the end of the EOT of the string that was printed. All the routines update the A accumulator on the User stack. This is not necessary for OUTEEE, but it does simplify the program to treat all the functions alike. Once the User map stack has been updated with the current register contents, the RTI sequence is used to transfer control back to the User map.

## CONCLUSION

Although EXbug was not designed to debug in both maps concurrently, programs can be developed and run which utilize both maps. This type of operation defeats the original purpose of the dual map configuration, that of providing a completely separate 64K map for program development. However, it does open up the possibility of running programs of a size that was not possible before.

00434

\* I/O CALLS TO EXEC MAP

00436	041F	A OUTEEE	EQU	*
00437A	041F	34	PSHB	SAVE B
00438A	0421	C6	CLRB	I/O CODE
00439A	0422	3F	SWI	GO TO EXEC
00440A	0423	35	PULB	RESTORE B
00441A	0425	39	RTS	

00443	0426	A PDATA1	EQU	*
00444A	0426	34	PSHB	
00445A	0428	C6	A LDAB	#1
00446A	042A	3F	SWI	
00447A	042B	35	PULB	
00448A	042D	39	RTS	

00450	042E	A INEEE	EQU	*
00451A	042E	34	PSHB	
00452A	0430	C6	A LDAB	#2
00453A	0432	3F	SWI	
00454A	0433	35	PULB	
00455A	0435	39	RTS	

00457	0436	A OUTHR	EQU	*
00458A	0436	34	PSHB	
00459A	0438	C6	A LDAB	#3
00460A	043A	3F	SWI	
00461A	043B	35	PULB	
00462A	043D	39	RTS	

00464	043E	A OUTS	EQU	*
00465A	043E	34	PSHB	
00466A	0440	C6	A LDAB	#4
00467A	0442	3F	SWI	
00468A	0443	35	PULB	
00469A	0445	39	RTS	

FIGURE 4. User Map I/O Routines to call Executive Map

```

00001          NAM    SWISRV
00002          *
00003          * SERVICES USER MAP SWI'S IN EXEC MAP
00004          *
00005          * EXBUG EQUATES
00006          *
00007      FCFC  A MPIAAD EQU  $FCFC
00008      F018  A XOUTCH EQU  $F018
00009      F00C  A XCHEXR EQU  $F00C
00010      F015  A XINCHN EQU  $F015
00011          *
00012          * TEMP VARIABLES
00013          *
00014A 0000          ORG    0
00015A 0000  0002  A SPSAVE RMB  2
00016A 0002  0002  A TEMPX RMB  2
00017          *
00018          * SWI SERVICE ROUTINE
00019          *
00020A 1000          ORG    $1000
00021      1000  A STACK EQU  *      EXEC MAP STACK
00022          *
00023          * ENTER HERE ON SWI
00024          *
00025      1000  A SWISRV EQU  *
00026          *
00027A 1000 10DF 00  A      STS    SPSAVE  SAVE USER MAP STACK POINTER
00028          *
00029A 1003 10CE 1000  A      LDS    #STACK  USE EXEC MAP STACK
00030          *
00031A 1007 34  04          PSHB   SAVE I/O REQUEST TYPE
00032          *
00033A 1009 C6  30  A      LDAB   #$30   SET UP USER MAP ACCESS
00034A 100B F7  FCFC  A      STAB   MPIAAD
00035          *
00036A 100E 35  04          PULB   GET I/O TYPE
00037A 1010 5A          DECB
00038          *
00039A 1011 2B  0A  101D  BMI    OUTEEE  0=OUTPUT A ACC
00040          *
00041A 1013 27  30  1045  BEQ    PDATA1  1=OUTPUT STRING
00042          *
00043A 1015 5A          DECB
00044A 1016 27  23  103B  BEQ    INEEE   2=INPUT TO A ACC
00045          *
00046A 1018 5A          DECB
00047A 1019 27  1B  1036  BEQ    OUTHR   3=OUTPUT 4 LSB OF A ACC AS HEX
00048          *
00049A 101B 86  20  A      LDAA   #$20   OUTPUT SPACE IS LEFT (4)
00050          *
00051A 101D BD  F018  A OUTEEE JSR  XOUTCH
00052          *
00053          * COME HERE TO RETURN TO USER MAP
00054          *
00055A 1020 9E  00  A RTNA  LDX    SPSAVE  UPDATE SWI A ACC
00056A 1022 B5  FCFC  A      BITA   MPIAAD  LOAD SR FOR USER MAP WRITE
00057A 1025 12          NOP
00058A 1026 A7  01  A      STAA   1,X    DELAY

```

FIGURE 5. Executive Map SWI Service Routine (Page 1)

00059		*						
00060A	1028	86	37	A	LDAA	#\$37	SET UP USER MAP RETURN	
00061A	102A	B7	FCFC	A	STAA	MPIAAD		
00062				*				
00063A	102D	10DE	00	A	LDS	SPSAVE	GET USER MAP STACK POINTER	
00064				*				
00065A	1030	B5	FCFC	A	BITA	MPIAAD	LOAD SR FOR USER MAP RETURN	
00066A	1033	12			NOP		TIMING DELAY	
00067A	1034	12			NOP			
00068A	1035	3B			RTI			
00069				*				
00070A	1036	BD	F00C	A	OUTHR	JSR	XCHXR	CONVERT 4 LSB OF A ACC TO ASCII HEX
00071A	1039	20	E2	101D		BRA	OUTEEE	
00072				*				
00073A	103B	BD	F015	A	INEEE	JSR	XINCHN	INPUT CHARACTER TO A ACC
00074A	103E	20	E0	1020		BRA	RTNA	PUT CHARACTER ON USER STACK & RETURN
00075				*				
00076A	1040	BD	F018	A	PDATA3	JSR	XOUTCH	PRINT CHARACTER
00077A	1043	30	01		INX			UPDATE POINTER
00078A	1045	B5	FCFC	A	PDATA1	BITA	MPIAAD	LOAD SR FOR USER MAP READ
00079A	1048	20	00	104A		BRA	#+2	DELAY
00080A	104A	A6	84	A	LDAA	0,X		
00081A	104C	81	04	A	CMPA	#4		END OF PRINT STRING?
00082A	104E	26	F0	1040		BNF	PDATA3	NO
00083A	1050	9F	02	A	STX	TEMPX		YES, UPDATE USER MAP SWI X REG
00084A	1052	9E	00	A	LDX	SPSAVE		
00085A	1054	D6	02	A	LDAB	TEMPX		
00086A	1056	B5	FCFC	A	BITA	MPIAAD		LOAD SR FOR USER MAP WRITE
00087A	1059	12			NOP			DELAY
00088A	105A	E7	04	A	STAB	4,X		
00089A	105C	D6	03	A	LDAB	TEMPX+1		
00090A	105E	B5	FCFC	A	BITA	MPIAAD		LOAD SR FOR USER MAP WRITE
00091A	1061	12			NOP			DELAY
00092A	1062	E7	05	A	STAB	5,X		
00093A	1064	20	BA	1020		BRA	RTNA	RETURN

00095 1000 A END SWISRV  
 TOTAL ERRORS 00000—00000  
 TOTAL WARNINGS 00000—00000

103B INEEE 00044 00073\*  
 FCFC MPIAAD 00007\*00034 00056 00061 00065 00078 00086 00090  
 101D OUTEEE 00039 00051\*00071  
 1036 OUTHR 00047 00070\*  
 1045 PDATA1 00041 00078\*  
 1040 PDATA3 00076\*00082  
 1020 RTNA 00055\*00074 00093  
 0000 SPSAVE 00015\*00027 00055 00063 00084  
 1000 STACK 00021\*00029  
 1000 SWISRV 00025\*00095  
 0002 TEMPX 00016\*00083 00085 00089  
 F00C XCHXR 00009\*00070  
 F015 XINCHN 00010\*00073  
 F018 XOUTCH 00008\*00051 00076

FIGURE 5. Executive Map SWI Service Routine (Page 2)

## APPENDIX H

### COMPATIBILITY AMONG M6809 EXORciser, M6800 EXORciser II AND M6800 EXORciser I MODULES

#### USING M6800 EXORciser II MODULES IN THE M6809 EXORciser

The M6809 EXORciser is designed to be compatible with the M6800 EXORciser II. Basically, the M6809 EXORciser is an M6800 EXORciser II, with a few of the motherboard bus signals redefined. These signals are listed below.

<u>Pin Number</u>	<u>New Signal</u>	<u>Old Signal</u>
J	E	$\emptyset 2$
N	<u>BUSREQ</u>	<u>TSC</u>
R	<u>MNRDY</u>	MEM RDY
S	LIC*	Normally Not Used (REF CLK on MEX6815-1 Module Only)
<u>A</u>	<u>FIRQ</u>	Not Used
7	Q	$\emptyset 1$
15	BUSGNT	TSG
23	BS	Normally Not Used ( <u>ACT</u> on MEX6816-1 Module Only)

\*Denotes that signal is applicable to M6809E EXORciser only.

A description of the function of each of these signals is given in Appendix A.

This redefined bus is compatible with the EXORciser II bus in the sense that all memory and standard I/O modules designed for use in EXORciser II can be used in the M6809 EXORciser.

Of course, any module designed with an MPU other than an MC6809 or MC6809E type processor installed in it, (such as M6800, M6801, M6802 MPU modules; M6800, M6802 USE modules; or M6800 Micromodules), will not function in an M6809 system. Also, only the M6809 DEbug Module will work in the system. DEbug II is not designed to function properly in the M6809 system without modification.

The M6800 System Analyzer II Module can be used in an M6809 System when used with the M6809 System Analyzer Intercept Module and M6809 PROM's installed.

When working with EXORDisk II or III, it is necessary to replace the M6800 PROM's with M6809 PROM's on the Floppy Disk Controller Module. A low on the MNRDY line will cause the system clocks to be stretched. For this reason, it is necessary for the disk drivers to operate off the 1 MHz controller clock if the drivers are to function properly when slow memories are installed in the M6809 system.

## USING M6800 EXORciser I MODULES IN THE M6809 EXORciser

Most EXORciser I modules are not rated to operate above 1 MHz. The system speed is limited to the maximum clock rate specified for the slowest module in the system. Therefore, when using EXORciser I modules, it is necessary to configure the M6809 system to 1 MHz.

Dual map capability requires that all peripherals and memories must be able to be assigned to the proper map (VUA or VXA). All EXORciser I modules will automatically respond to VUA when operating in an M6809 EXORciser (or EXORciser II) system. In order to assign these modules to VXA, the user must modify the individual modules by cutting and jumpering a single connection at the edge connector of the EXORciser I module. The modification is the same for all EXORciser I modules, and is illustrated in Figure 1. Cut the incoming VUA signal track from pin 10 near the edge connector finger. Solder a jumper wire from the VXA signal track (pin 19) to the circuitry side of the track cut just performed.

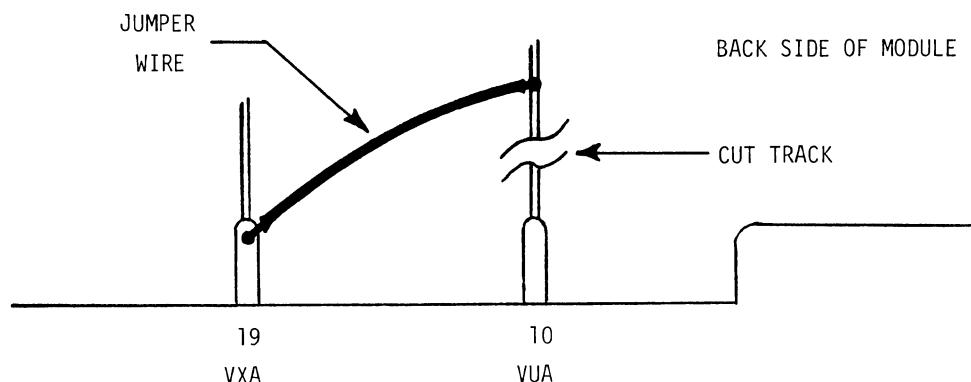


FIGURE 1. Assigning EXORciser I Modules to VXA

## 8K Dynamic RAM Module (MEX6815-1)

The early 8K Dynamic RAM Module (MEX6815-1) requires some precaution as well as modification before use in an M6809 system. This module transfers data asynchronously. Timing problems will result if this module is used when working with the M6809 USE Module. If the USE function is not used, the 8K Dynamic RAM Module (MEX6815-1) may be used in the M6809 EXORciser if the following precautions are taken and changes are made.

Only a few early versions of the 8K Dynamic RAM Modules (MEX6815-1) used a 60-microsecond refresh cycle. This was changed to 30 microseconds. If any difficulty is encountered using 8K-1 memories with newer memories, a quick check of the refresh time should be made. The 60-microsecond timing cannot be used with newer modules.

Also important when mixing dynamic RAM's is the early 8K-1 requirement to provide +12VDC to the battery backup signal (pin U on EXORciser I bus). On EXORciser I, this bus signal was normally jumpered to +12VDC (pin T) when the

product was manufactured. In the M6809 EXORciser, pin U is still designated as STANDBY, but is not jumpered to +12VDC at the factory (since the application is more general in the M6809 EXORciser). When using 8K-1 dynamic RAM's in an M6809 EXORciser chassis, the user must re-arrange the module +12VDC track connections as illustrated in Figure 2, and perform the following modifications:

- a. If jumper wires exist between pins T and U (front side of module) or pins 16 and 17 (back side of module), cut or remove the applicable jumper wire.
- b. Cut the incoming +12VDC track from pin 17 near the edge connector finger.
- c. Solder a jumper wire from pin 16 to the +12VDC feed-through terminal on the back side of the module.
- d. Solder a jumper wire from pin T to the same +12VDC feed-through terminal on the front side of the module.

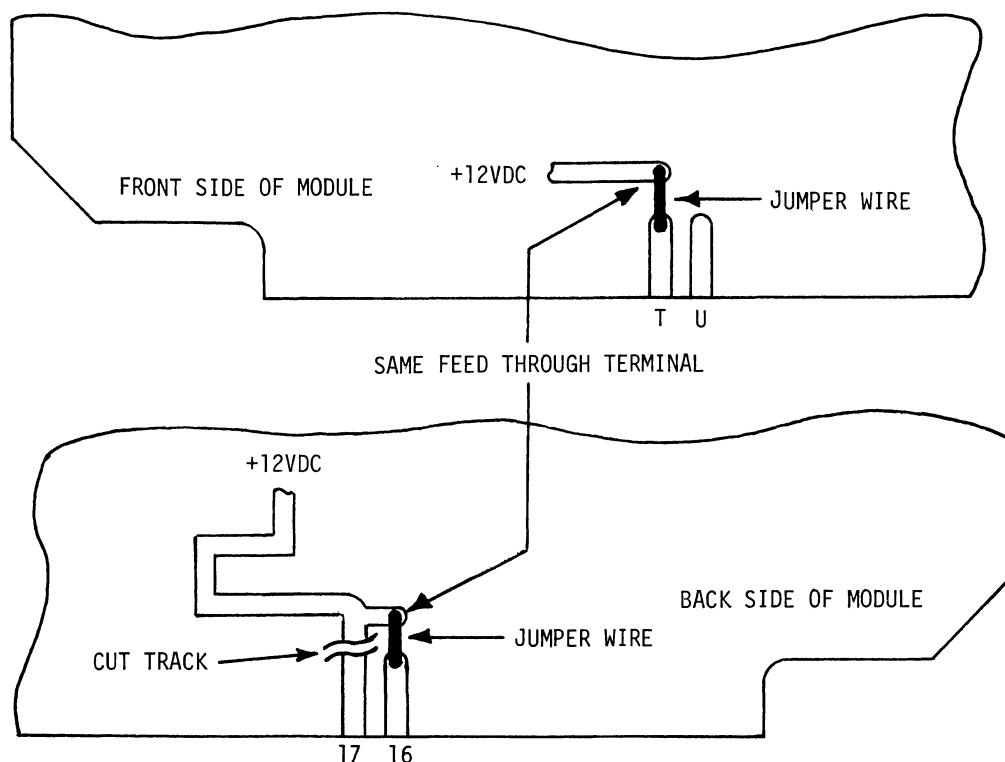


FIGURE 2. +12 VDC Modification to 8K Dynamic RAM Module (MEX6815-1)

When using 8K-1 modules with other dynamic RAM's (including Series II RAM modules), the 8K-1 module must be assigned as the master in the system.

Pin S on the M6809 EXORciser bus has been redefined as the M6809E signal LIC (Last Instruction Cycle). Previously this line was used by the 8K-1 modules and was identified as a REF CLK (Refresh Clock) signal. The 8K-1 module is the only module which uses this signal. To avoid multi-signal conflicts on this line, simply disable the LIC line from the MPU Module by removing jumper connection K8 on the MPU Module.

When working with the MC68B09E MPU, the user has two options to avoid multi-signal conflicts. First, the user can use the same method previously explained. Make sure there are no modules in the system requiring the LIC signal, then disable the LIC signal from the MPU Module by removing jumper connection K8. The second option is to remove the REF CLK signal from the M6809 EXORciser bus. As long as there is only one 8K-1 module in the system, this is possible. If there is more than one 8K-1 module in the system, the REF CLK signal is required to synchronize the module refresh circuitry. In this case, only the first option can be used.

The REF CLK signal can be disabled from the 8K-1 module by inserting a jumper connection at POS1 on jumper platform JA. Refer to Figure 3.

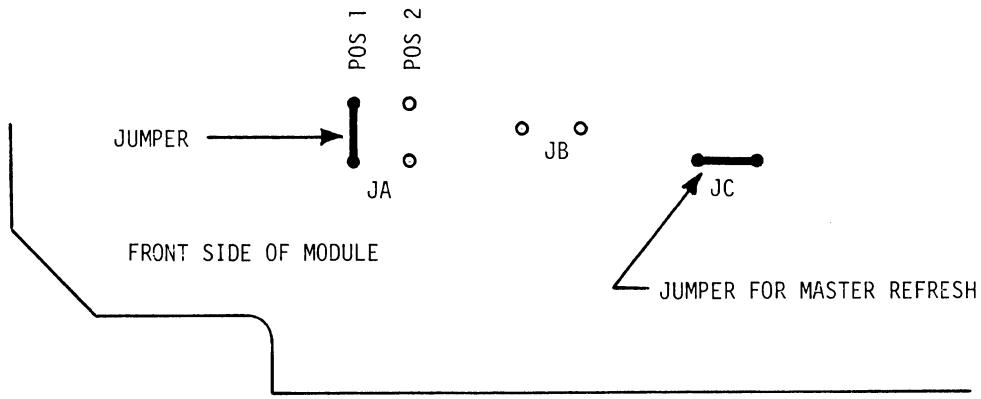


FIGURE 3. REF CLK Signal Disabled on 8K Dynamic RAM Module (MEX6815-1)

#### 16K Dynamic RAM Modules (MEX6816-1)

The early 16K Dynamic RAM Module (MEX6816-1) requires modification before use in an M6809 EXORciser. These modules generate an ACT (Activate) signal on pin 23. This signal has been redefined in the M6809 EXORciser as BS (Bus Status). It is necessary to disable one of these signals in order to avoid damage to either module. The recommended modification is to disable the ACT signal on the 16K Dynamic RAM Module because it is not used on any other M6800 or M6809 module. Refer to Figure 4. To disable the ACT signal, cut the ACT signal track (pin 23) above the finger on the module edge connector, as illustrated.

It is also possible to disable the BS signal from the M6809 MPU Module by removing the jumper connection at K2 of the MPU Module. Prior to performing this modification, the user must make sure that none of the modules in the system requires the BS signal.

The 16K Dynamic RAM Module (MEX6816-1) transfers data on the bus asynchronously. This causes a timing problem when operating with the M6809 USE Module. In order to enable the RAM module to transfer data synchronously, perform the following modifications on the back side of the RAM module:

- a. Remove +5VDC connection from U21 pin 13 (refer to Figure 4). This is accomplished by cutting the tracks between pins 13 and 14 and between +5VDC and pin 13, as illustrated.

- b. Restore +5VDC connection to U21 by soldering a jumper wire between +5VDC and U21 pin 14.
- c. Connect a delayed CLK (Clock) signal to U21 pin 13 by soldering a jumper wire between U17 pin 10 and U21 pin 13, as illustrated.

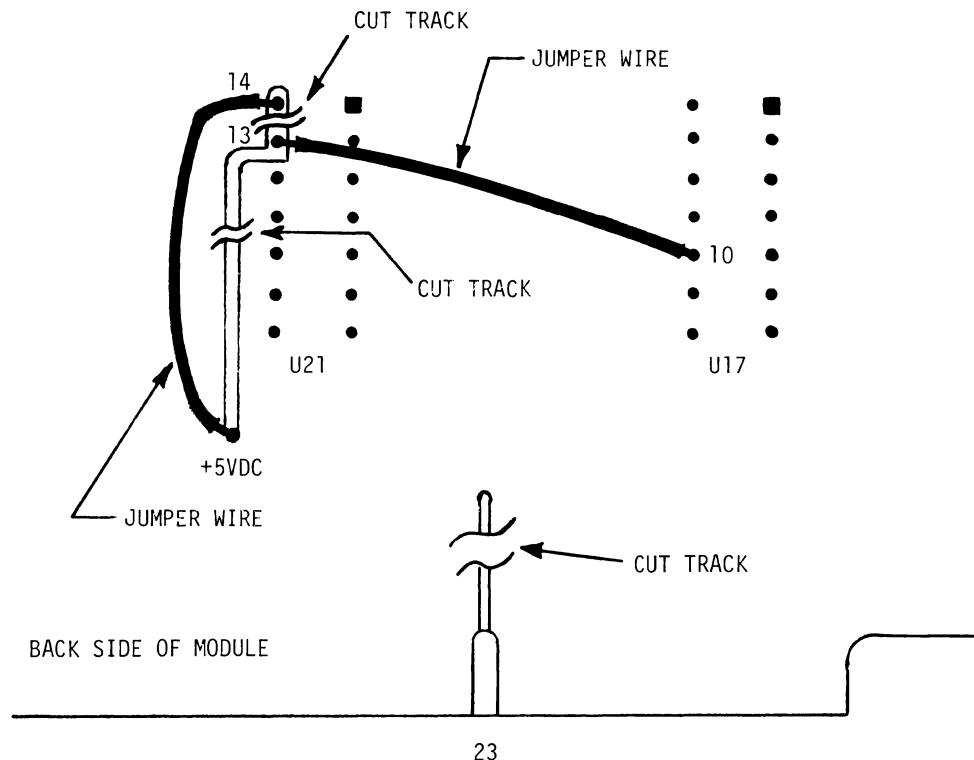


FIGURE 4. USE Modification for 16K Dynamic RAM Module (MEX6816-1)

The M6800 Systems Analyzer I can work at 1 MHz in an M6809 system if the M6809 PROM's are installed and the M6809 Systems Analyzer Intercept Module is used.

It is not possible to use or modify the DEbug I Module.

EXORDisk I will work with a 1 MHz M6809 system as long as slow memories are not installed. Using slow memories with EXORDisk I will cause read/write errors.



## APPENDIX I

### DIRECT MEMORY ACCESS ON THE DEVELOPMENT SYSTEM BUS

## ACCESSING PROCEDURES

DMA capability allows for high performance (typically next bus cycle access) and automatically arbitrates between the MPU execution requirements, memory refresh request, and the DMA device(s) access request(s). The interface is simple and is centered around two bus signals BUSREQ and BUSGNT. The BUSGNT signal is designed to allow cycle-by-cycle bus master arbitration, eliminating any need for dead bus cycles during bus mastership transfers. Rules for bus access are specific allowing for guaranteed results and thus preventing bus access conflicts.

The steps for accessing the bus may be generalized as follows:

1. The DMA device should prepare for the data transfer by setting up the memory address (and data if a write is to occur) as well as appropriate VMA and R/W signal states.
2. The DMA device activates BUSREQ 40ns prior to the rising edge of E. (See Figure 1.)
3. The DMA device waits for a BUSGNT.
4. Upon receiving a BUSGNT, the device may execute a data transfer during that bus cycle. The device must be prepared to relinquish bus mastership during the next bus cycle because either the M6809 MPU or refresh may regain bus mastership during the next cycle. During data transfers, the DMA device must control the address and data buses as well as VMA and R/W (as necessary).
5. If more data is to be transferred, then the device continues to hold BUSREQ active and returns to step 3. If the device will complete its transfer following the present bus cycle it must deactivate BUSREQ prior to the rising edge of E of its last transfer cycle.

## REQUEST AND GRANT SIGNALS

The user who is designing a DMA interface to the Development System should study the following more detailed discussion of the two bus signals BUSGNT and BUSREQ.

BUSGNT is the only signal which directly indicates the availability of the system bus. BA (MPU bus available) and BS (MPU bus state) may be used to determine the status of the M6809 MPU but there is no simple relationship between these signals and bus availability (BUSGNT).

BUSREQ is the only signal which will initiate activation of BUSGNT. HALT will not activate BUSGNT (though it may affect BA and BS). The relationship between BUSREQ and BUSGNT is not direct: it also depends on REFREQ (refresh request), and on how long the MPU has been inactive.

If the conditions of the processor and REFREQ are agreeable, then BUSGNT will be issued without delay in response to the BUSREQ.

BUSGNT is inactive (low) during the first 50ns after the falling edge of E of every bus cycle whether it will be inactive for that cycle or not. Thus, actual recognition of bus mastership should be done on the rising edge of Q. (See Figure 1.)

This signal (BUSGNT) should be directly attached by the user to the address, data, VMA, and R/W bus driver enables of the DMA peripheral. Thus a non-conflicting cycle-by-cycle bus arbitration can occur since all bus drivers are disabled during the first 50ns of each cycle and BUSGNT is used to select between bus masters.

Figure 1 shows the pertinent bus signals (BUSGNT, BUSREQ, REFREQ, REFGNT, E, Q) during DMA access and applies for any bus speed (E cycle length).

The top part of the figure shows the timing when there is no refresh request. The bottom part of the figure shows how REFREQ causes a REFGNT output and disables BUSGNT.

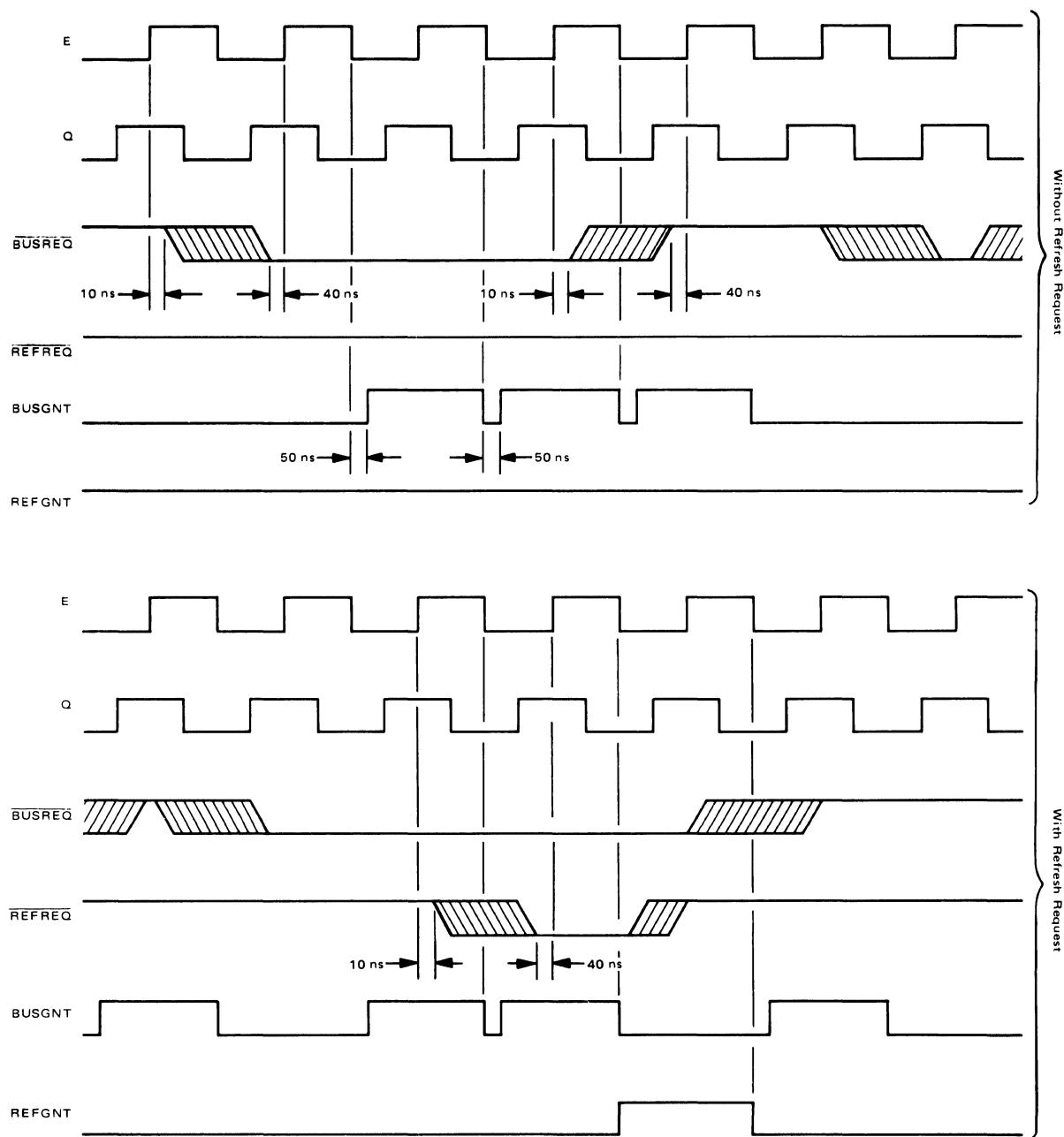


FIGURE 1. M6809 DMA Timing

## DMA HARDWARE DETAILS

Two basic methods of DMA access are possible under the above bus access protocol, cycle steal or halt steal.

Cycle steal is the basic method of DMA access using BUSREQ and waiting for a BUSGNT. Halt steal involves requesting an MPU halt by activating HALT (P1 pin 4), waiting for BA = 1 and BS = 1, and then placing the BUSREQ. Note that halt steal is acknowledged only after completion of the current instruction. Figures 2 and 3 show the circuitry for implementing both methods.

Selection between the two methods is made strictly on the basis of the DMA device characteristics. During the cycle steal mode, the MPU will regain bus mastership for one cycle every 14 cycles during a constant BUSREQ. For halt steal, no interruption by the MPU will occur once the bus is granted, but the delay until the bus is granted may be extreme (up to 20 cycles for a CWAII). It should also be noted that a REFREQ will cause a lost cycle in either mode. Thus, the cycle steal mode is by far the most preferable and the DMA device interface should sufficiently buffer the data stream to allow for lost cycles to MPU and to refresh.

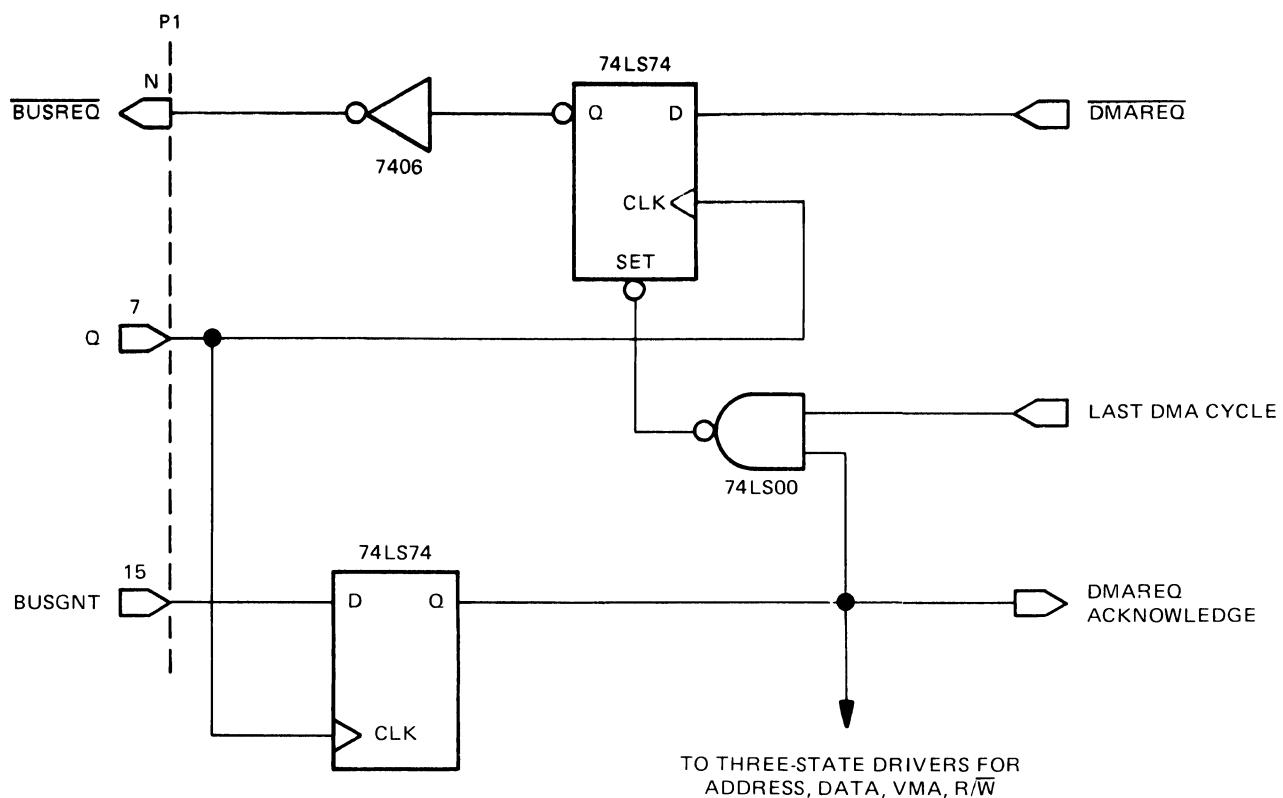


FIGURE 2. Simple Cycle Steal DMA Interface

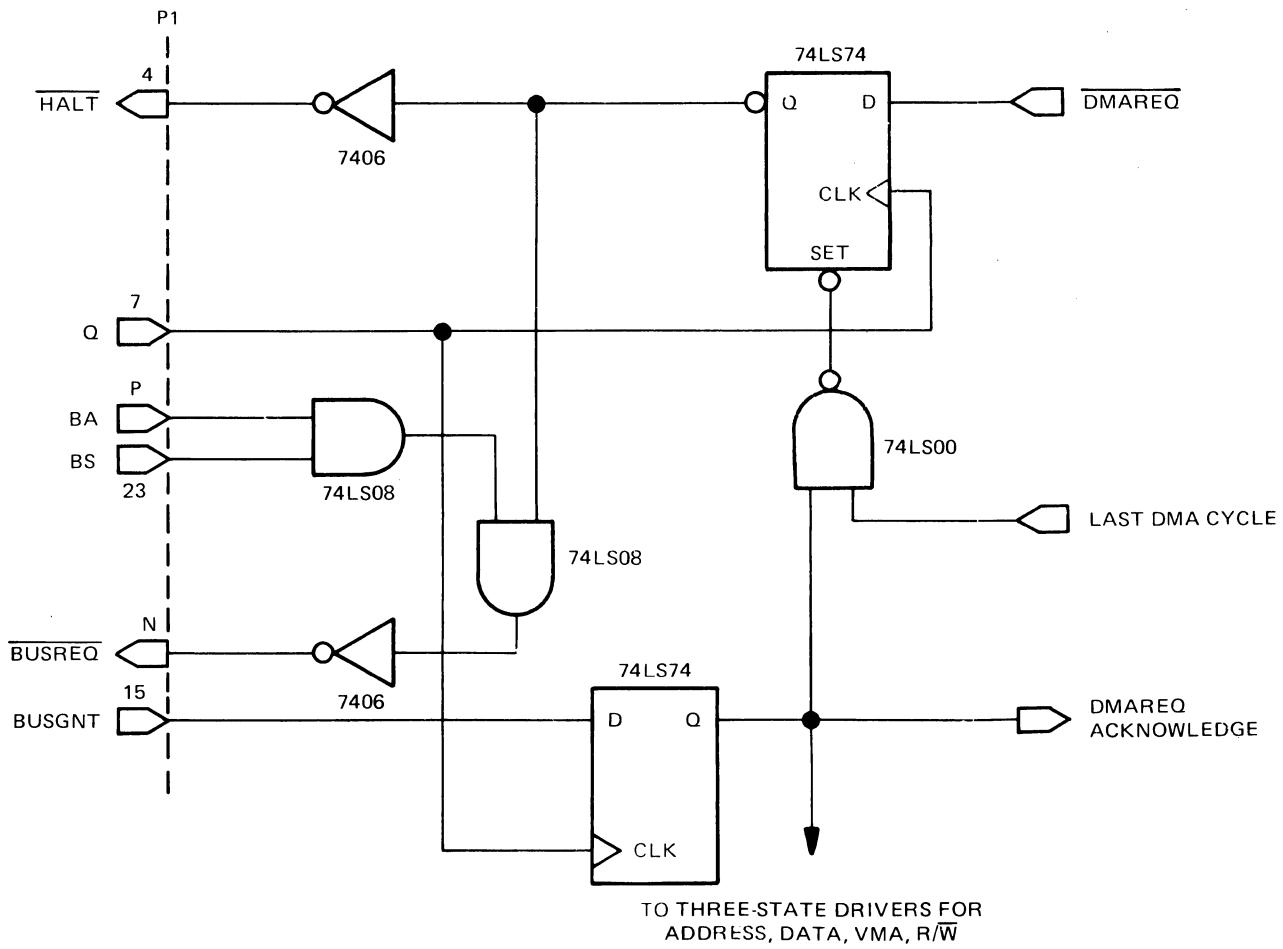


FIGURE 3. Simple Halt Steal DMA Interface

The scheme developed for performing DMA on the Development System bus will allow expansion to multiple DMA devices or bus masters. Bus access prioritization among multiple devices may be done using a daisy-chain or spatial method alone or in conjunction with a rotating enable technique.

The spatial priority method is accomplished by running a signal line from device to device to form the daisy-chain. This line is referred to as the BRQOUT by the source device and BRQIN by the receiving device. Thus each device has a bus access priority input and output, BRQIN and BRQOUT respectively.

The interconnect is illustrated in Figure 4. Thus, if BRQIN to device A is inactive, that is, no device higher in the chain requires the bus, then device A is allowed access to the bus. If device A does not require access to the bus, then its BRQOUT and consequential device B BRQIN is inactive thus allowing device B access to the bus, etc. Figure 5 shows the circuitry for implementing this for a given device.

The daisy-chain scheme alone has some draw backs, in that the device last on the chain may never gain access to the bus if the other higher-priority devices are always busy. The rotating enable technique solves this problem by allowing each device only one access to the bus until all other devices have their requests serviced. This lockout may be done on a single cycle or single full request basis. Figure 6 shows the circuitry.

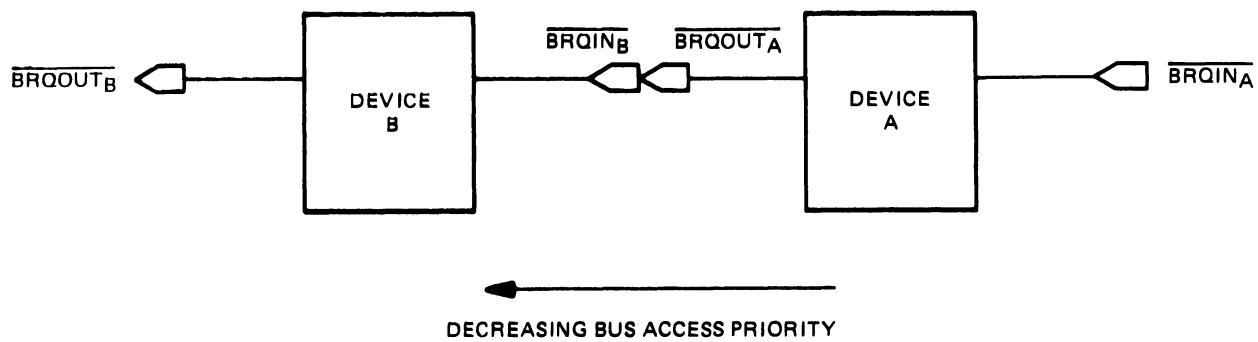


FIGURE 4. Spatial Prioritizing for DMA

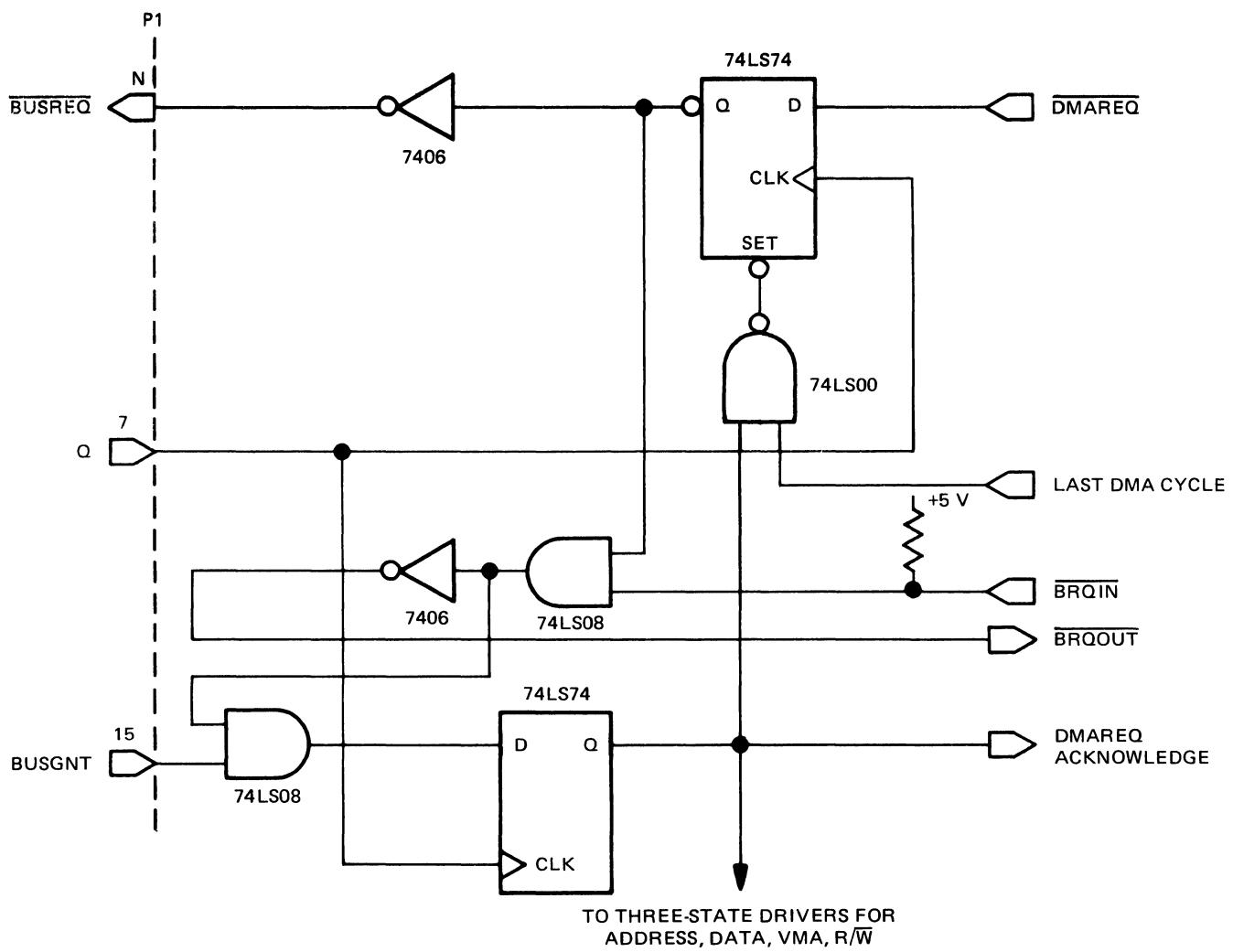


FIGURE 5. Spatial Prioritizing DMA Device Interface

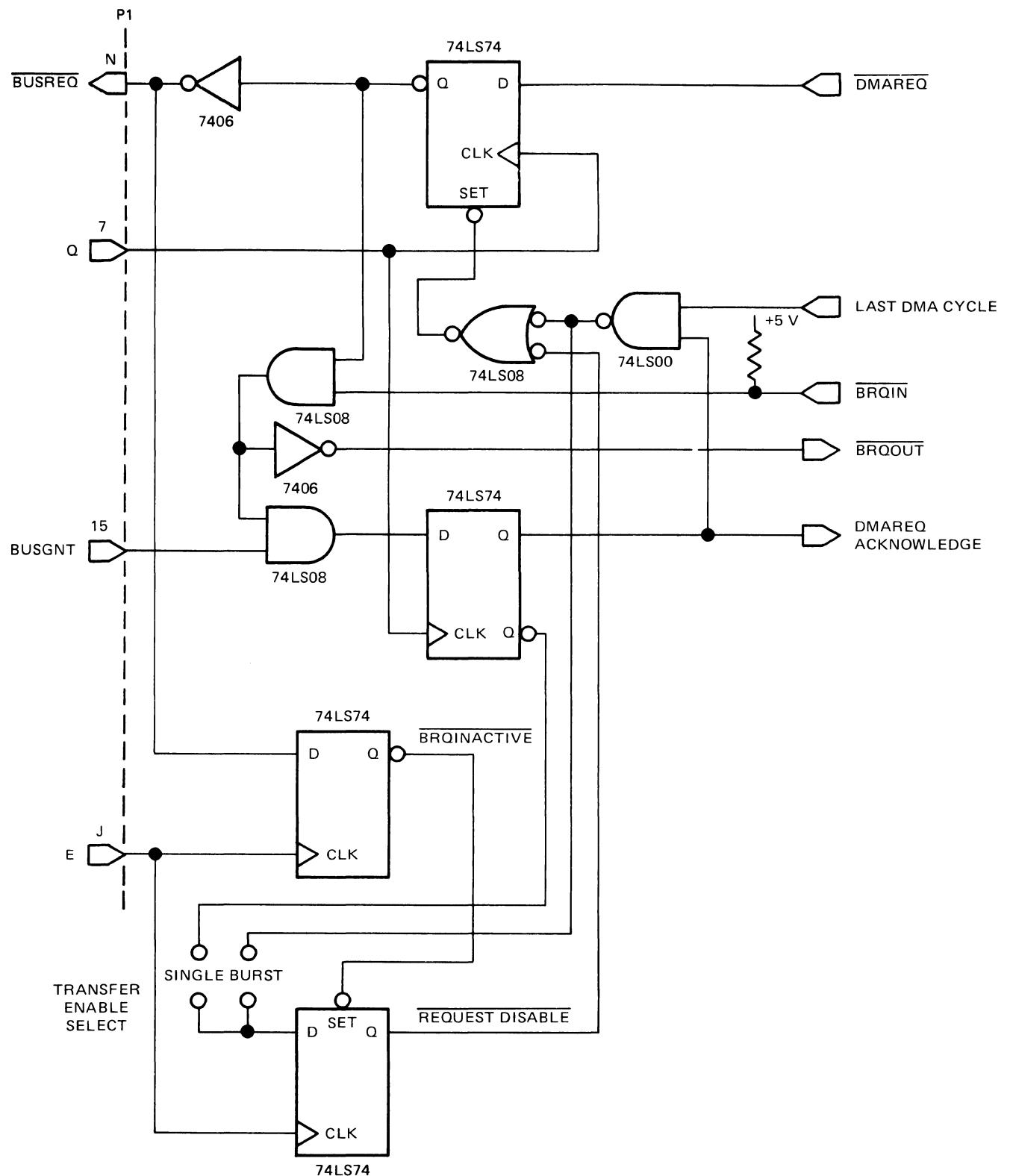


FIGURE 6. Rotating Priority with Spatial Arbitration DMA Device Interface



# SUGGESTION/PROBLEM REPORT

Motorola welcomes your comments on its products and publications. Please use this form.

To: Motorola Microsystems  
P.O. Box 20912  
Attention: Publications Manager  
Mail Drop M374  
Phoenix, Az. 85036

Comments

Product: \_\_\_\_\_

Manual: \_\_\_\_\_

*Please Print*

Name \_\_\_\_\_

Title \_\_\_\_\_

Company \_\_\_\_\_

Division \_\_\_\_\_

Street \_\_\_\_\_

Mail Drop \_\_\_\_\_ Phone Number \_\_\_\_\_

City \_\_\_\_\_

State \_\_\_\_\_ Zip \_\_\_\_\_

HARDWARE SUPPORT: (800) 528-1908  
SOFTWARE SUPPORT: (602) 831-4108





**MOTOROLA Semiconductor Products Inc.**

P.O. BOX 20912 • PHOENIX, ARIZONA 85036 • A SUBSIDIARY OF MOTOROLA INC.