

MICRO-68000 USERS MANUAL

Published by:

**Computer System Associates, Inc.
7564 Trade Street
San Diego, CA 92121
(619) 566-3911
Telex 333693**

**-2 Edition
First Printing
August, 1985**

**Copyright 1985 by CSA, Inc.
All rights reserved.**

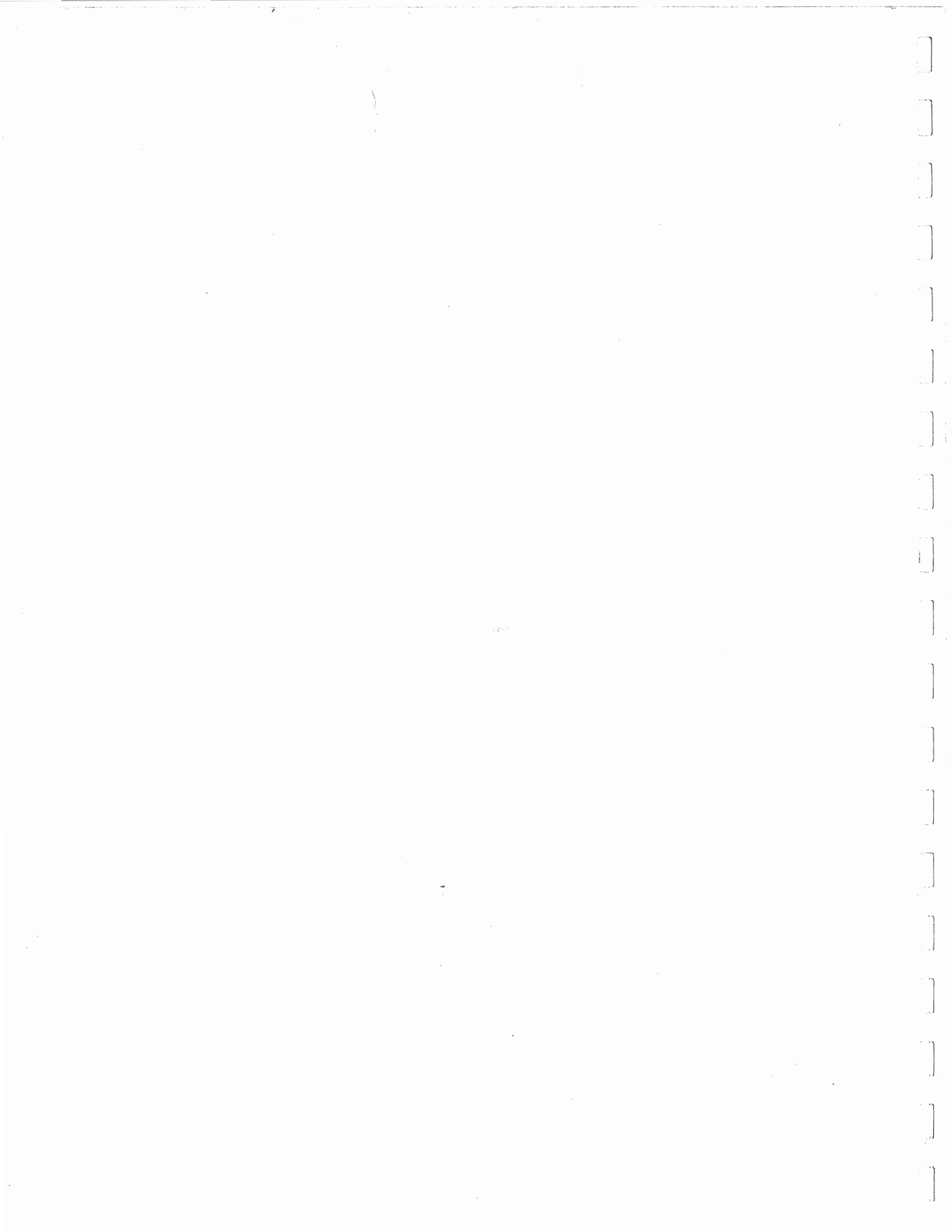


TABLE OF CONTENTS

Table of Contents	i
List of Illustrations	iv
List of Tables	iv
Preface	
The CSA Trainer	
The Users Manual	
In Case of Difficulty	
CHAPTER 1 GENERAL DESCRIPTION	
Introduction	1-1
CSA Trainer Documentation	1-1
CSA Trainer Features and Specifications	1-1
Equipment and Accessories Supplied	1-2
CSA Trainer Assemblies	1-2
CSA Trainer Keypad	1-6
CSA Trainer Display	1-7
Summary	1-8
CHAPTER 2 OPERATION/POWER UP	
Introduction	2-1
Installation	2-1
Power Up	2-1
Theory of Trainer Operation	2-2
The MC68000 MPU At Power Up	2-2
Versabus in the CSA Trainer	2-4
The MPU, Bus and Petebug	2-5
Summary of Operation	2-9
Initial Trainer Tests	2-9
Demo 1 Procedures	2-9
Demo 2 Procedures	2-9
Master Mind	2-10
Master Mind Game Play	2-14
Summary	2-15

CHAPTER 3 FUNCTIONAL DESCRIPTION

Introduction	3-1
Major Assemblies	3-1
Power Supply	3-1
MPU Board	3-2
MPU Section	3-2
The MPU Bus and Versabus	3-5
The MPU Peripheral Section	3-6
EEROMS	3-7
Memory Map	3-8
Address Decoders	3-13
Initial Start and Reset	3-15
DTACK and FPGA Devices	3-16
EEROM and FPGA Devices	3-16
I/O and FPGA Devices	3-16
FPGA Summary	3-17
Parallel Input/Output (PIA)	3-18
Serial Input/Output (ACIA)	3-18
The Keyboard/Display Assembly	3-19
The Keyboard	3-19
The Displays	3-21
Petebug Routines	3-23
Refresh the Displays	3-24
Display Values in Seven Segments	3-24
Display Value in a Set of LED's	3-25
Scan the Keyboard	3-25
The Stepper Motor	3-26
The Stepper Motor Program	3-26
Stepper Motor Basics	3-27

CHAPTER 4 USING PETEBUG

Introduction	4-1
Petebug Command Table	4-2
Petebug Command Summary	4-13
Summary	4-47

CHAPTER 5 MC68000 ARCHITECTURE

Introduction	5-1
Quick Reference Guide	5-2

CHAPTER 6 MC68000 INSTRUCTIONS

Introduction	6-1
Quick Reference Guide	6-2

CHAPTER 7 MC68000 INPUT/OUTPUT

Introduction	7-1
Quick Reference Guide	7-2

CHAPTER 8 TINY BASIC

Introduction	8-1
Numbers	8-1
Variables	8-1
Functions	8-1
Arithmetic and Compare Operators	8-1
Expressions	8-2
Program Lines	8-2
Tiny Basic Commands	8-3
REMARK	8-3
LET	8-3
PRINT	8-4
INPUT	8-4
POKE	8-5
CALL	8-5
IF	8-5
GOTO	8-5
GOSUB	8-5
RETURN	8-6
FOR and NEXT	8-6
STOP	8-7
BYE	8-7
Direct Commands	8-7
RUN	8-7
LIST	8-7
NEW	8-7
SAVE	8-7
LOAD	8-8
Stopping Program Execution	8-8
Abbreviations and Blanks	8-8
Error Reports	8-8
Error Corrections	8-9
Running Tiny Basic	8-9
New Features	8-9
Saving and Loading Tiny Basic Programs in EEROM	8-10

CSA Users Manual (CSA-UMM68KTA)
for, CSA TRAINER (CSA-M68000TA)
TABLE OF CONTENTS Page #4

CHAPTER 9 A/D CONVERTER

Introduction	9-1
Program Overview	9-1
General Organization	9-1
Data Direction Registers	9-1
Display PIA	9-2
Keyboard PIA	9-2
Sending Control Signals	9-3
Displaying Data	9-3
Sample Program	9-3
Summary	9-4

CHAPTER 10 D/A CONVERTER

Introduction	10-1
Program Overview	10-1
General Organization	10-1
Tone Generation by the Sample Program	10-1
D/A Sample Program	10-2
Summary	10-2

APPENDIX A

GLOSSARY	G-1/G-24
----------	----------

LIST OF ILLUSTRATIONS

CSA Trainer Rear View	2-1
MC68000 Instruction Format	2-2
Master Mind Display	2-3
Digital Logic Circuits	2-4
MPU Component Location Diagram	3-1
● Trainer Display Schematic	3-1A
Versabus Pinout	3-2
Memory Jumper Selection	3-3
Memory Map	3-4
FPGA Program Charts	3-5
Keyboard Key Labels	3-6
Keyboard Matrix	3-7
Display Layout	3-8
Display Block Diagram	3-9
Stepper Motor Schematic Diagram	3-10
Digital Gates/Logic	3-11

CSA Users Manual (CSA-UMM68KTA)
for, CSA TRAINER (CSA-M68000TA)
TABLE OF CONTENTS Page #5

• Micro-68000 Trainer Schematic Diagram	3-12
Petebug Diagram	4-1
Auto Flow Chart	4-2
Change Flow Chart	4-3
Display Flow Chart	4-4
A/D Converter Schematic	9-5
D/A Converter Schematic	10-3

LIST OF TABLES

CSA Trainer Documentation List	1-1
CSA M68000TA Specifications	1-2
CSA M68000TA Equipment List	1-3
Petebug Subroutines	2-1
Master Mind Keyboard Commands	2-2
AO Bit Byte Addressing	3-1
Trainer Designated Addresses	3-2
FPGA Input Signals	3-3
FPGA Outputs	3-4
Petebug Keyboard Input	4-1
A/D Control Signals	9-1
A/D Input Selection	9-2

CSA Users Manual (CSA-UMM68KTA)
for, CSA TRAINER (CSA-M68000TA)
SAFETY AND NOTICE PAGE

SAFETY NOTICE

The operation of the Micro 68000 Trainer / Prototyping System (CSA-M68000TA) presents no electrical or mechanical danger to personnel when installed and operated as directed by this manual and COMPUTER SYSTEM ASSOCIATES. However, ALL PERSONNEL having access to this equipment should be aware of and observe all practical safety precautions, as proscribed for equipments operating from facility line AC HIGH VOLTAGE, (110/230 VAC, 60 Hertz, 50 Watts, single phase).

=====

FEDERAL COMMUNICATIONS COMMISSION (FCC)

required warning for CLASS A computing devices

* WARNING *

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instructions contained in this manual, may cause interference to radio communications. The CSA-M68000TA has been tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference in which case the user, at his own expense, will be required to take whatever measures may be required to correct the interference

PREFACE
GENERAL CSA INFORMATION

BACKGROUND

The CSA M68000TA Trainer (henceforth referred to as the CSA Trainer or Trainer) has been designed to meet the critical need for a valid training device in the microprocessing industry. The training required to acquire a working knowledge of sophisticated microprocessors, such as the MC68000 (32 bit microprocessor), created the need for a complex training device. This need, was further complicated, in that, the training device should not require more attention to use, than the topic that is to be taught. The CSA Trainer has successfully supplied a solution to both of these needs. The CSA Trainer is an easy to use interface that will fulfill all of the requirements necessary for an excellent MC68000 training device.

THE CSA TRAINER

The CSA Trainer grants total access to the MC68000 internal structure and instruction set (at both the machine language level and at the assembly language level), while presenting the student with a real time, interactive, microprocessor environment. The ease of use that the CSA Trainer employs, allows the Trainer to become a transparent background, therefore the total attention of the student can be focused on the MC68000 microprocessor unit (MPU) and on the learning topic at hand. Some of the features of the CSA Trainer are as follows:

- a. All of the necessary CSA Trainer/MC68000 input keys are on one keypad. These keys (hexadecimal, binary and command keys) are oversized, clearly labeled and are color coded.
- b. The display is separated into two clearly definable hexadecimal (alphanumeric segments) and binary (LED) areas. During all trainer operations, a separate group of LEDs presents an indication of the status currently functioning on the CSA trainer.
- c. Additional features such as serial and parallel input/output (I/O), extended Versabus (wire wrap compatible), jumper selectable hardware/firmware options and open circuit architecture all contribute to the overall versatility and user control of the CSA trainer.
- d. The CSA trainer may be expanded to include EEROM user memory, a video terminal (needed for Tutor operation) and demonstrator equipment (stepper motor, A/D converter, D/A-speaker options).

THE CSA TRAINER MANUAL

This manual reflects only a part of the documentation

delivered with the CSA Trainer (see Table 1-1). The Motorola MC68000 Manual and the Assembly Language Manual are sources of invaluable reference data and represent the type of materials that the student will need to use in the industry. The CSA Laboratory Manual has been included to facilitate the use of "hands on" training through the various processes of programming using the binary (base 2) and hexadecimal (base 16) number systems. This manual contains the information and instructions required to operate and utilize the Trainer. Information from Motorola in direct relation to the MC68000 has been included to supplement the Trainer documentation. A copy of the applicable Tutor documentation has been included to aid in using Tutor. Tutor may be used for machine language programming and for assembly and disassembly of the MC68000 Instruction Set. In addition, a training aids section and a glossary have been included to support training. A look at the Table of Contents will contribute to an understanding of the organization and presentation of the material contained within this manual.

IN CASE OF DIFFICULTY

CSA is prepared to assist you with any problem that you may encounter with the CSA trainer. Prior to asking CSA for assistance, please check the section of this manual that applies to your problem. Many problems are not problems at all, just a lack of proper information. CSA has made every effort to ensure that the information, required to operate and use your Trainer is available to you. When phoning CSA ensure that you have the Serial Number of the CSA Trainer (located on the label, on the rear of the unit). Also have pencil and paper ready, should a Return Material Number (RMN) be required. The address and phone number for CSA are:

COMPUTER SYSTEMS ASSOCIATES (CSA)
7564 TRADE STREET
SAN DIEGO, CA 92121
(619) 566-3911
Telex 333693

CHAPTER ONE
GENERAL DESCRIPTION

INTRODUCTION

This chapter will introduce the CSA Trainer and the accessories required to operate the unit. The assemblies of the CSA Trainer will be described and you will be introduced to the Trainer controls and displays. A brief overview of the CSA Trainer hardware and firmware is included in this chapter, however, detailed descriptions of these features are presented in chapters 3 and 4, respectively. Tables showing the complete documentation and specifications of the CSA Trainer are also presented and may be used for future quick reference. The unpacking and repacking instructions that were shipped with this Trainer should be added to the end of this chapter, in case of future need.

CSA TRAINER DOCUMENTATION

As described in the Preface, this manual is only a part of the support documentation that is supplied with the CSA Trainer. Table 1-1 presents a complete list of all the support documentation that is supplied.

CSA TRAINER FEATURES AND SPECIFICATIONS

The features and specifications of the CSA Trainer are listed in Table 1-2. The CSA Trainer specifications in the table are separated into areas of general, hardware, firmware, and expansion options.

Table 1-1 CSA Trainer Documentation

CSA Number	Title	Use
CSA - UMM68KTA	CSA Users Manual	Training
CSA - LMM68KTA	CSA Laboratory Manual	Training
CSA - TBM68KTA	68000 TUTOR MANUAL (Appendix A)	Training
CSA - ALM68KTA	68000 ASSEMBLY LANGUAGE (McGraw - Hill)	Reference
CSA - MOM68KTA	16-Bit Microprocessor Users Manual (Motorola)	Reference
CSA - SCM68KTAM	680010, 16-Bit Programming card (Motorola)	Quick Ref.

EQUIPMENT AND ACCESSORIES SUPPLIED

Table 1-3 presents a list of the equipment (other than the documentation) that was shipped with the CSA Trainer. Whether the Trainer you received was a long unit or the short briefcase type, the accessories will be the same, only the packaging differs between the two Trainers. A check of this table against the materials actually received will determine whether your shipment was complete. Should a shortage of some item be discovered, notify CSA immediately (during working hours). The address and phone number for CSA are given in the Preface.

CSA TRAINER ASSEMBLIES

- The CSA Trainer has three major assemblies, as follows:
- a. The Power Supply Assembly
 - b. The MC68000 MPU board assembly
 - c. The Keyboard/Display Assembly.

Table 1-2 CSA M-68000TA Specifications

GENERAL SPECIFICATIONS:		
Name	Description	Comment
Power Supply	Switching type, rated +5 VDC @ 6 Amps -12 VDC @ 0.2 Amps, and +12 VDC @ 3 Amps	(Max. Load) (4 Amps Peak)
MC68000 MPU Facility Power	16-bit microprocessor 115/230 VAC, 60 Hz, Single Phase, 50 Watts	Motorola Source Power
Packaging	Two Types: a. Long Case Hardwood Case See Through Cover b. Short Case Hardwood Case (2 units) See Through Cover	Name: Long Short
Dimensions	Long: 31L, 11.75W, 3.125H Short: 17L, 11.5W, 4 H	Inches Inches
Weight	Trainer: 10 Pounds	Each
FCC Rated	Class A Computer	Commercial
Documentation	Fully Supported	See Table 1-1

HARDWARE SPECIFICATIONS:

Name	Description	Comment
Major Assemblies	Three: Power Supply MPU Board Keyboard/Display	See Text (all) Switching MC68000 Main Control
Memory	Installed: RAM / 16 K Bytes PROM / 16 K Bytes PROM / 16 K Bytes	Jumper Selectable System / User Petebug Tutor
Serial I/O	2 Supported: a. TERM b. HOST	Connectors: Female Male
Parallel I/O	2 Supplied: 32 Data Lines 4 Control lines	MPU to Keyboard/ Display Assy
VERSABUS	On MPU Board (BUS) 140 Lines Avail2. 16 Data Lines 24 Address Lines MPU Control Lines	Extended to rear of unit Wire wrap connections
Clock Crystal	2 Available	1 Supplied
(MPU Y1)	Freq. Set: 4.9152 Mz.	See Options

FIRMWARE SPECIFICATIONS:

Monitor (Programs)	3 Supplied	Reference
Petebug	Hex / Binary Input to memory and MPU registers	Chapter 4
Tutorbug	Hex / Binary Input Plus Asssembler / Disassembler	* Appendix A
Tiny Basic	High Level Language	* Chapter 8

* Requires a video terminal

EXPANSION OPTIONS: *

Carrying Case	2 Styles: Both are heavy constructed, padded, finely crafted. Long or Short briefcase type.	1 Each Unit
Video Terminal	RS232 Compatible	Tutor Control
Stepper Motor	Programmable Automation Demonstrator	CSA Option
EEROMs	Memory Expansion 16 K Electrical Eraseable ROMs.	CSA Option
Analog to Digital Converter	8 Bit	
Digital to Analog Converter	8 Bit	With speaker
Clock Crystal	User Specified Frequency	Special Order Call CSA
Serial I/O	Connecting Ribbon Cables	2 Types, Option Call CSA

* CSA will consider requests for any User specified special
options, call or write CSA for an estimate.

Table 1-3 CSA M-6800OTA Equipment Supplied

Quantity	Description	Comment
1	Power Supply	Assembly
1	68000 MPU	Assembly
1	Keyboard/Display	Assembly
1	Hardwood Case	Long Unit
2	Hardwood Cases	Short Unit
1	Clear F Cover	Long Unit
2	Clear Covers	Short Unit
1	AC Power Cord	Black
1	Parallel Ribbon	3 Connector
1	Installed Fuse	3 Amp / Rear
1	Document Package	List / Table 1-1
1	Pack / Unpack	Instructions
1	CSA Warranty	Return to CSA
1	User's Grp. Reg.	Mail
1	User's Report / Product Profile	Fill out & Return CSA

EXPANSION OPTIONS // USER FILL IN FOR REFERENCE.

Name	Description	Serial #
-----	-----	-----
-----	-----	-----
-----	-----	-----
-----	-----	-----

The assemblies listed in table 1-3 are self identifying, the metal box with a single red LED, is the sealed Power Supply Assembly. The MPU assembly has the large MC68000 microprocessor installed in a Zero Insertion Force (ZIF) socket, on the far left of the MPU board. The keyboard/display assembly is obvious by its components (keypad and display indicators). These assemblies will be described in detail in chapter 3. Notice that all of the IC chips on the CSA Trainer have been mounted in IC sockets. This will not only allow for quick replacement of chips if required, but allows instructors to purposely install training (adjusted) chips for troubleshooting exercises. There are two large ZIF sockets in the lower mid portion of the MPU board. These sockets are for the optional EEROM memory chips that will be described later. Should these two sockets be empty on your Trainer, it is probably no mistake, only that the EEROMs were not ordered with your unit. However, should the EEROMs be ordered at some future date, CSA has prepared the MPU board to accommodate the installation of the new chips.

CSA TRAINER KEYPAD

The CSA Trainer keyboard consists of twenty color-coded keys arranged in a 4 column, by 5 row rectangle. Across the top row are the command keys, ENTER KEY (in this manual the convention <ENTER> will be used), HEX/BIN key, (hexadecimal or binary), BREAK key, (referred to as ABORT in some instances) and the RESET key (hardware reset...memory is not cleared). Across the second row are the hexadecimal (Hex) literal numeric keys (C through F), and in columns 3 and 4 of the next row are Hex A and B. Look at the keypad closely and you will see that these keys (except E) have a command function label also. These command functions will be understood by the CSA Trainer dependent on the Mode and order of entry into the keypad. Briefly, these functions are as follows:

CHANGE (C key) -	Used to change data in memory or register
DISPLAY (D key) -	Used to display data in memory or register
FWD F(F key) -	Used to step forward in memory or register count
AUTO (A key) -	Used for automatic sequence of data input
BACK (B key) -	Used to step back in memory or register (Not allowed in Memory Instruction Mode)

The remaining keys are the numeric keys (0 through 9); there are also commands associated with these keys that are invoked in the same manner as the previous command keys.

Beginning at row 3, column 1, these command keys are:

RUN (8 key) -	Used to run a program in memory
STEP (9 key) -	Used to Single Step one program instruction
(4 key) -	Used in binary entry to move cursor right
(6 key) -	Used in binary entry to move cursor left
+ (1 key) -	Used for Hex addition
- (2 key) -	Used for Hex subtraction
MODE (3 key) -	Used to select Mode of operation (memory, or register and data size)

CSA TRAINER DISPLAY

The CSA Trainer Display is organized in a logical manner as follows:

The top eight Hex digit displays are the Address display, these display segments are also used for Petebug error (HUH?) or status messages (68000 UP, ABORT).

The lower five groups, of four Hex segments each, on the Hex display are used for displaying data or 68000 instructions.

The first 16 bit LED display is located next to the Hex instruction display, and will display data and instructions in the binary format.

There are three more groups of LEDs alongside of the Hex instruction extension displays (extension 1 through 3) that will display instruction extensions in binary.

Below the Hex display groups, there are eight LEDs that are used to display the STATUS of the operation of the Trainer during various functions.

There are more detailed descriptions of the displays functional capabilities and Trainer uses in Chapter 3 and Chapter 4. At present, you only have to be able to recognize whether a value is being displayed or not. When a Hex digit or an LED is illuminated, this indicates that data or instructions are being displayed. All binary LEDs will not light, just those signifying binary one's.

SUMMARY

In this chapter, through the use of text and tables, you have been introduced to a number of specifications and characteristics related to the CSA Trainer. At this point the CSA Trainer should be unpacked and inspected for damage, and the student should be aware of the Assemblies, major hardware, and firmware that makes up the CSA Trainer. In the next chapter the operation of the CSA Trainer will be discussed as well as how to properly apply Power to the unit.

Ensure that the entire chapter (2) is read and understood, prior to applying AC Power to the CSA Trainer.

CHAPTER TWO OPERATION AND POWER

INTRODUCTION

In Chapter 1 the CSA Trainer was described and the major assemblies and components were introduced. In this chapter the installation (interconnects) of the Trainer, the power connections and controls, and a description of Trainer theory of operation will be presented. With primary power connected to the Trainer, you may perform an operational test on the Trainer, with the two demonstration programs available in Petebug. Then there will be an introduction to the game of Master Mind, where the User may match wits with the computer. To complete the presentation of the operational abilities of the CSA Trainer a list of the subroutines available in Petebug is presented in Table 2-1. CSA recommends that this chapter be read in its entirety, prior to connecting AC power or any attempts to operate the Trainer.

INSTALLATION

The CSA Trainer is a portable unit and easily installed for operation. However, to ensure that an important step is not overlooked, follow this procedure:

1. On the rear of the Trainer, (see Figure 2-1), ensure that the ON/OFF (rocker type) Power switch has the lower switch arm pressed into the case (OFF).
2. Locate the black AC Power Cord and connect the female end to the CSA Trainer and the male end to the source power connection (wall receptacle).

POWER UP

The CSA Trainer is now installed and ready to receive AC power. As shipped, the Trainer will power up and the MC68000 will execute the Petebug monitor operating program. The procedures for connecting a video terminal and executing the Tutor operating program are presented in Appendix A and in the CSA Laboratory Manual. The procedures for using the Tiny Basic language are shown in Appendix C. Also, in Chapter 3 of this manual, the details for jumper selecting any of three MC68000 Vector Start options, during power up, are described. To apply AC power and make the initial checks on the CSA Trainer, proceed as follows:

1. Press the upper rocker arm of the Power switch in toward the Trainer case (ON).

CAUTION

High voltage AC power is now applied to the CSA Trainer. Although CSA has taken every precaution to protect the User from contact with the primary power (sealed and shielded power supply), the User is reminded to observe all safety procedures applicable to operating electronic equipment.

2. Observe the power LEDs on each of the three major assemblies and verify that the LEDs are illuminated.
3. Observe the display and verify that the message "68000 UP" is displayed. This message indicates that the CSA Trainer has properly executed the Petebug monitor program (during power up), and is ready for operation.

If conditions are as stated in steps 2 and 3 then proceed to the next paragraph. If conditions are not as stated, remove AC power from the Trainer and review the installation steps for an error. If a malfunction still exists after review, see the

THEORY OF TRAINER OPERATION

ADDITIONAL REFERENCE MATERIAL.....

- Motorola (16 bit) Micro Manual...Chapters 1 & 2 to Para 2.11
- Assembly Language ManualChapter 3, pp 3-1 to 3-6
- CSA Laboratory ManualChapter 1 (All)

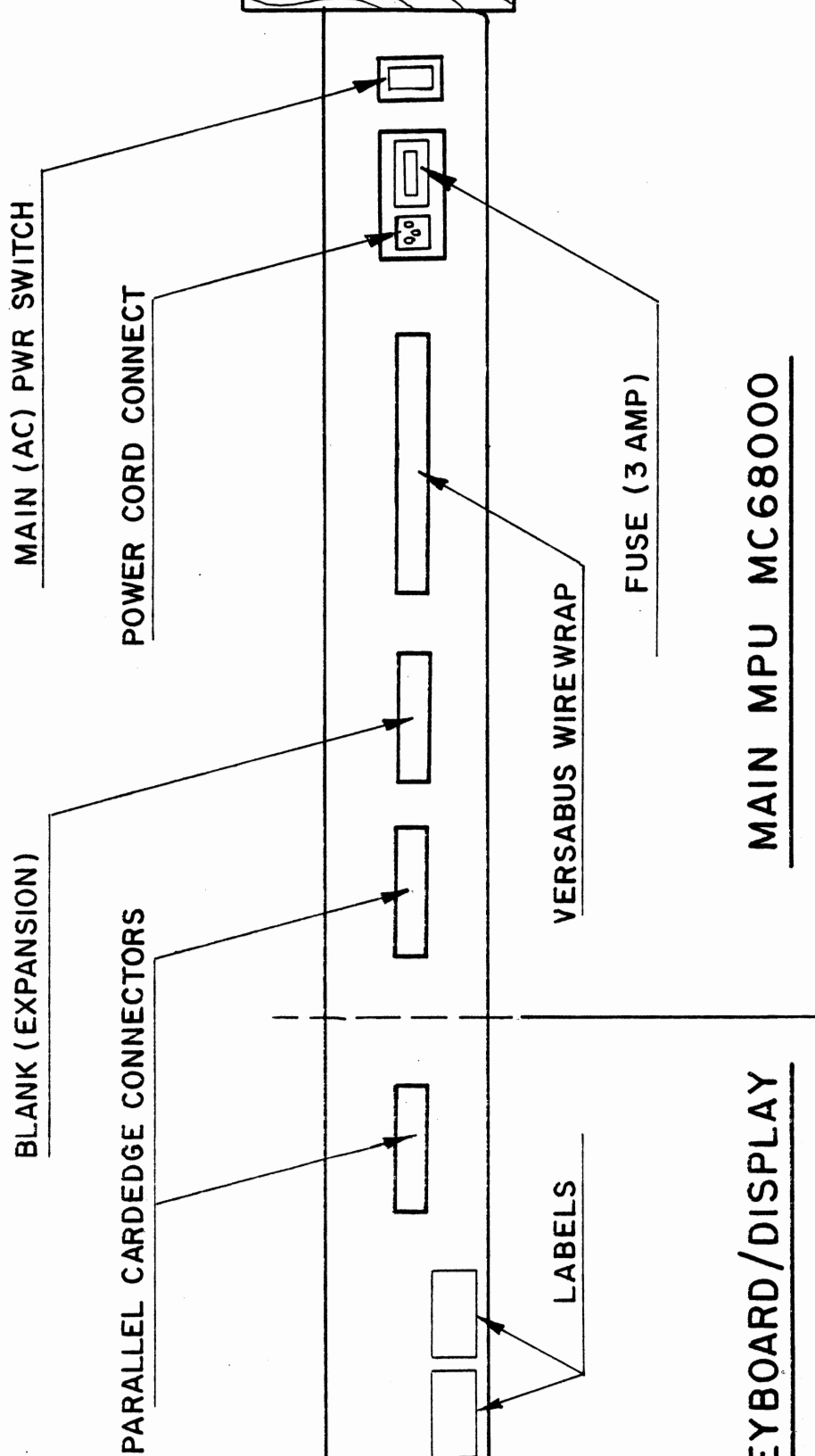
The CSA Trainer, as a computer system, meets the three basic requirements of all computer systems:

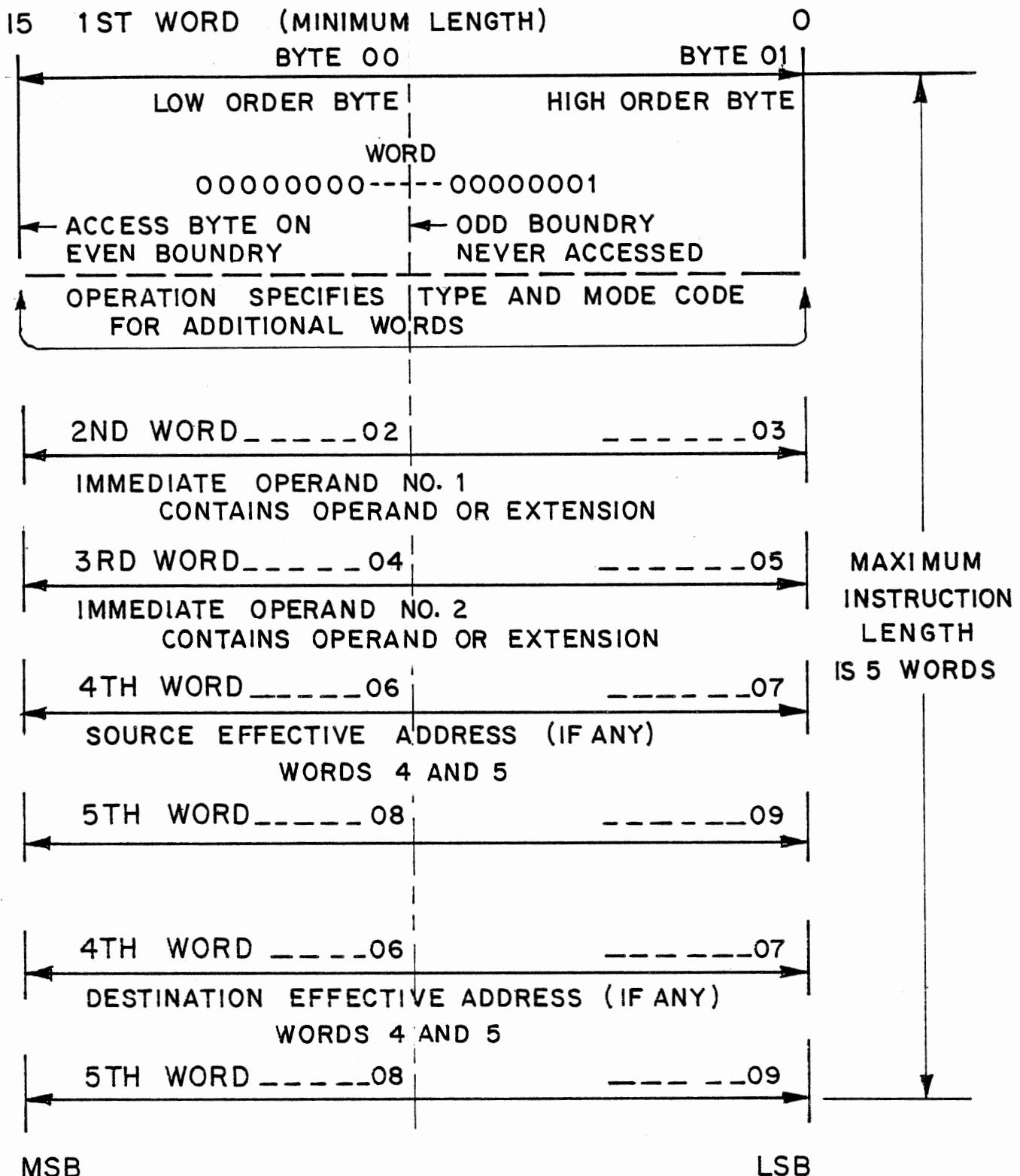
BASIC SYSTEM	CSA TRAINER
a. Central Processing Unit	MC68000 microprocessor (MPU)
b. Memory	RAM, PROM, Optional EEROM
c. Input and Output (I/O)	Keyboard, input/Display, output

In addition to these minimum essentials listed above, the CSA Trainer has three operational programs installed in PROM (Petebug, Tutor and Basic). As the Trainer is configured to vector to and RUN Petebug, at initial Power Up, we shall only discuss the Petebug monitor during this description.

The MPU (MC68000) at Power Up

When the power is first applied to the CSA Trainer, or when RESET is pressed, the MPU clears all registers and seeks an address in the Program Counter (PC) register to fetch (load) an





NOTE: INSTRUCTIONS ARE ALWAYS AT LEAST ONE WORD (16 BITS) LONG, AN 8 BIT OR LESS WILL BE PADDED WITH ZEROS TO FILL IN 16 BITS

CSA COMPUTER SYSTEM ASSOCIATES
MC 68000

INSTRUCTION FORMAT
FIGURE 2-2

a. The MC68000 fetches an instruction from a memory location (Petebug) specified by the PC register, decodes the instruction and executes the instruction

b. The MC68000, through the Trainer hardware, generates the control signals to transfer data on the data lines from the instruction specified address (if any), loads the data (which may be another instruction), decodes the new instruction, and/or executes the former instruction on the data retrieved.

c. The MPU increments the PC register the correct amount of word (16 bits) increments, so as to fetch the next instruction and repeats the cycle started at "a" above.

This cycle can be interrupted, or diverted but essentially this is the cycle that the MPU performs over and over again. In the discussion of the MPU thus far many terms have been used that may require further explanation. The following terms are commonly used in the microprocessor industry:

INSTRUCTION - The machine code (binary) that has been designed into the MC68000 to communicate with external directions (instruction codes). Instructions that are part of the MC68000 Instruction set are from one to fives words in length, (see Figure 2-2). Instructions are planned as a sequence of steps in a computer program or routine, organized to accomplish either an entire sequence of operations (program) or a particular task (routine). Instructions normally have two parts. the first part is the the Operation Code (OPcode). The OPcode specifies exactly, the MPU operationn that the MC68000 is to perform. The second part of an instruction, the Operand, is the data, (or specifies the address of the data), that the OPcode is to operate on. The MC68000 has an entire set of instructions to perform tasks such as fetch, store, arithmetic, logic and other microprocessor functions, (see Motorola, 16 bit Microprocessor Users Manual, Pages 79 through 182).

ADDRESSING - This refers to the MPU's ability to locate an individual cell (location, amongst thousands) in memory. or to specify an I/O device (address). There are several forms of addressing available to the programmer when planning MPU instructions that will perform an operation or task. These forms of addressing will be described in Chapter 6 of this manual. An address is the specific identibty (in binary) assigned to a memory cell or I/O device in order that the MPU may locate and communicate with that cell or device.

DATA - Data as a general term relates to binary information required to perfrom a function within the computer system (CSA Trainer). Data is moved, transfered in memory, or to registers, input from external devices (keyboard), output to

external devices (display), and combined with or compared to other data to create new data. The data types for the MC68000 must be grouped in a specific manner. Refer to the Motorola 16-bit Microprocessor Manual, Section 2, page 13, for a detailed description of MC68000 data organization. The MC68000 acceptable data types are listed below:

- a. Bit Data (1, 4, 8, 16, or 32 bits)
 - 4 bits = nibble
 - 8 bits = byte
 - 16 bits = word
 - 32 bits = long word
- b. Addresses up to 32 bits. Only the low order 24 bits are significant in the CSA Trainer as there are only 23 address lines
- c. Binary Coded Decimal (4 bits). Binary coded decimal is a coding system to represent the numbers 0 through 9 in a special four bit code

See the Assembly Language Manual, Chapter 3, page 3-3 through 3-6 for a description of data and addresses in MPU registers and in memory.

VERSABUS In The CSA Trainer

In order for addresses to be specified, data to be transferred, and synchronization of actions (control) to exist, there must be multiple paths to allow these separate but related signals to travel from place to place. When several related signals are carried by a group of circuit board traces or wires, the group is commonly referred to as a bus. The overall bus structure for the MPU system in the CSA Trainer is called the Versabus (see Chapter 3). The three major busses within the Versabus and on the MPU board are;

- a. Address bus (23 lines, A1 through A23)

NOTE

The A0 bit in an address is internally generated by the MC68000. The MPU then generates the Upper Data Strobe (UDS) if the A0 bit is low (0) or the Lower Data Strobe (LDS) if the A0 bit is high (1). See Chapter 3.

- b. Data bus (16 lines, D0 through D15)
- c. Control bus (address and data strobes and various other control signals).

MPU, Petebug, and Busses

In this paragraph, as the title suggests, the individual elements of the CSA Trainer will be brought together to work in harmony as a system. The MPU is constantly seeking new instructions, at a rate that is determined by the CLOCK oscillator. As each instruction is fetched and processed, the MPU increments the PC register and begins a new instruction cycle. These instructions are supplied to the MPU by the Petebug program, that CSA has developed to reside within the PROM memory. The program instructions tell the MPU what to do (OPcodes) and specify where to find the data to do it (Operands). The MPU carries out the instructions by addressing memory or I/O devices, generating the control and timing signals, and following the program instructions. Addresses are located on the Address bus and data is transferred on the Data bus. The Control bus supplies the signals to determine if data is to be loaded into an MPU register (READ) or sent from the MPU to memory (WRITE). Control signals also determine when addresses and data are stable on their respective busses and allow the MPU to make the required transfers. Much of the hardware on the MPU board is in support of the control signals and used to maintain order within the Trainer system. The FPGAs, as previously mentioned, decode the addresses and ensure that the proper memory location or I/O device is addressed. Although the MPU is the heart of the system, there would be no operation without instructions that direct the MC68000 to perform.

The MC68000 fetches these instructions from Petebug and into the designated (Instruction) register. The current address of the instruction remains in the PC register until the MPU performs the functions of the instruction. Once the instruction has been completed, the MPU increments the PC register the proper amount of words and fetches the next instruction. The instructions from Petebug may contain a jump or branch instruction. In this instance the PC register is incremented to the next instruction address and that address is stored in memory for use later. Then the MPU fetches the address of the jump or branch and performs the routine or subroutine beginning at the fetched address. Routines and subroutines are sets of coded instructions designed to perform one small task (see Glossary). Routines that are part of a larger program are called subroutines, and are usually subordinate to the program. Upon completion of the routine, control will be passed back to the main program. The MPU uses the previously stored address to enter the main program one instruction increment greater than the jump or branch it previously carried out. Petebug contains the subroutines that the CSA Trainer requires to perform the house keeping tasks of the Trainer system. The keyboard has to be periodically scanned to see if a key has been pressed. If a key has been pressed, the display must be updated to indicate the detection of the

keystroke. The keystroke data must be gathered and displayed until <ENTER> signifies that the input is complete. When the input has been completed and entered, Petebug makes the input available for a fetch from the MPU for action. While all of this is going on, the display has to be refreshed continually or it will go black, (no indication). The MPU uses a Petebug subroutine to refresh the display periodically and ensure that it stays bright and readable. Some of the subroutines in Petebug that will instruct the MPU to perform various functions are available by external User's calls entered at the keyboard. These subroutines are activated from the keyboard by entering RUN (8 key) and the start address of the subroutine. Table 2-1 is a list of the subroutines available in the Petebug Monitor. The subroutines that are initiated by the MPU are labeled AUTO and the User subroutines are labeled MAN.

Table 2-1 Petebug Subroutines

Add.	Name	Description	Type
FF800C to FF800F	START	This is the address and label for the actual start of routines in Petebug	AUTO
FF8010 to FF8013	REFRESH	Refresh the display to maintain bright illumination. Values (data) for the display are stored in an area of memory (RAM) designated the REFRESH BUFFER.	AUTO
FF8014 to FF8017	DSPVAL	Display a numeric value as a Hex digit in the seven segment Hex display.	AUTO
FF8018 to FF801B	DSPLED	Display a byte value (in reverse bit order) in the LED display. The reversal of bits is required in order that the value will be displayed in logical order.	AUTO
FF801C to FF801F	SCNKP	Keyboard scan routine - Detects when a key has been pressed and returns key value to register D0. BREAK and RESET are interrupts and are not detected by this routine.	AUTO
FF8020 to FF8023	GNUM	Get Number - This routine collects Hex number input from the keyboard and displays each hex digit as it is input. This routine also checks for length of input and returns the ZERO Flag as follows: a. Set to 0 - length good b. Set to 1 - length no good Routine is terminated by <ENTER>.	AUTO
FF8024 to FF8027	RPSAVE	This routine is called from Petebug by the "7" key. The routine requests more input (see Chapter 3), then READS an EEROM program into RAM.	MAN
FF8028 to FF802B	WPSAVE	This routine is called from Petebug by the "5" key. The routine requests more input (see Chapter 3), then WRITES a RAM program to EEROM.	MAN

Table 2-1 Petebug Subroutines (Cont.'d)

Addr.	Name	Description	Type
NOTE			
<p>The next two routines are similar to RPSAVE and WPSAVE, except that they are Tutor routines. These routines (RTSAVE and WTSAVE) are accessed from Tutor by entering the jump address, followed by either a GO (G) or a GO DIRECT (GD) command.</p>			
FF802C to FF802F	RTSAVE	This routine is called from Tutor and requests more input. The function is the same as RPSAVE	MAN
FF8030 to FF8033	WTSAVE	This routine is called from Tutor and requests more input. The function is the same as WPSAVE	MAN
FF8034 to FF8037	ERMCOM	This subroutine is used by the MPU to program EEROMs.	AUTO
FF8030 to FF803B	DEMO1	This routine is called from Petebug by pressing the RUN key followed by the address. This routine will cycle the lights (OFF/ON) on the display, as well as act as an overall test of the Trainer.	MAN
FF803C to FF803F	DEMO2	This routine is called from Petebug and is very similar to DEMO1 (above) except that the scan pattern is different	MAN
FF8040 to FF8043	STEPPER	This routine may be called from Petebug, but requires special I/O and programming. (See Chapter 4). This is the Stepper Motor routine.	MAN
Key 0	MASTER MIND	This program is called from Petebug by the Trainer keyboard, ZERO (0) key. This game program (Master Mind) may be played to familiarize the User with the CSA Trainer System.	MAN

Summary of Operation

During this description, of necessity, the explanations have been kept brief and only the major functions have been described. The I/O capabilities of the CSA Trainer are much more complex and versatile than we have described and the MPU is much more than a fetch and storage machine. The Arithmetic Logic Unit (ALU) of the MC68000 is very powerful and may perform MOVES of large data blocks in memory, compare bytes and words, as well as many other sophisticated logic and math functions. The power of the MC68000 has to be implemented by the program instructions that the MPU receives. These instructions must come from either, one of the two on board monitor programs, or they must be entered into memory by the User. In the course of your studies of the MC68000 microprocessor you will have the opportunity to program instructions into the Trainer memory and have the MPU execute your program. Refer to the CSA Laboratory Manual for some examples of the types of programming that may be practiced with the CSA Trainer.

CSA TRAINER INITIAL TESTS

To understand the following tests the Theory of Operation paragraphs should have been read. While it is not required to understand the tests to perform them, the training nature of this manual strongly suggests that the effort be extended. This paragraph will present the procedures for performing three of Petebug's built in routines. The first two, DEMO1 and DEMO2, are quite similar except for the scanning pattern of the display indicators. The third routine is actually a game program and CSA recommends that the game be used to become familiar with the keyboard and display of the CSA Trainer. The routines are a good indication of the Trainer's readiness as most of the Trainer interfaces and controls are required to successfully complete the performance. The Petebug program is used by the MPU as a source of instructions. The parallel interface between the keyboard and the display are brought into action as addresses or commands are input and a display of activities is output. In addition, while DEMO1 or DEMO2 is running every display indicator will be cycled ON/OFF and a faulty indicator may be visually identified. To ensure the fitness of the CSA Trainer and to avoid future problems, perform the routines as presented and you can be assured that the Trainer is functioning properly.

DEMO1 PROCEDURES

NOTE

During any of the following procedures, Pressing the BREAK key will abort the procedure in process. At any time the RESET key may be pressed to clear all MPU registers and return to the Petebug start message, "68000 UP".

To RUN the DEMO1 routine, follow these procedures:

1. On the Trainer keyboard, press the RUN (8) key.
2. On the Trainer keyboard, press the following series of letters and numerals to input the jump address for the DEMO1 subroutine:

F, F, 8, 0, 3, 8 (FF8038 - NO COMMAS)

3. Observe the address display and ensure that the display shows the correct address, (above). Press <ENTER> to input the jump address to Petebug.
4. After the DEMO1 program has run through the display cycle a few times, and the indicators have been visually checked, press BREAK to abort the program. RESET may be pressed to bring the CSA Trainer back to the Petebug message "68000 UP"

DEMO2 PROCEDURES

NOTE

During any of the following procedures, Pressing the BREAK key will abort the procedure in process. At any time the RESET key may be pressed to clear all MPU registers and return to the Petebug start message, "68000 UP".

To RUN the DEMO2 routine, follow these procedures:

1. On the Trainer keyboard, press the RUN (8) key.
2. On the Trainer keyboard, press the following series of letters and numerals to input the jump address for the DEMO2 subroutine:

F, F, 8, 0, 3, C (FF803C - NO COMMAS)

3. Observe the address display and ensure that the display shows the correct address, (above). Press <ENTER> to input the jump address to Petebug.
4. After the DEMO2 program has run through the display cycle a few times, and the indicators have been visually checked, press BREAK to abort the program. RESET may be pressed to bring the CSA Trainer back to the Petebug message "68000 UP"

MASTER MIND

The original game of Master Mind, is played by two players. Player One selects four colored pegs, from a group of several

colored pegs, and arranges these four pegs in a row. Player Two, or the challenger, has to guess the color of the four pegs selected and the position of each colored peg within the row. For example: player one selects colors red (r), orange (o), blue (b) and green (g). When placing the selected pegs in a row, the same order is preserved (r,o,b,g). The challenger does not know the colors or the order of the colored pegs selected. To win the game, both of these unknowns must be guessed in a limited number of moves or guesses. As the game progresses, Player One is required to give Player Two certain hints to aid in finding the correct solution. The challenger is told after each guess, two valuable pieces of information:

- a. Any color that was guessed correctly, regardless of position.
- b. That one or more pegs are correct color and position, but not specifically which pegs.

By using these two clues (a and b), the challenger can keep refining the next guess to logically get closer to the correct solution. The play continues until the correct solution of all four colors and their matching positions is guessed (WIN), or the challenger uses all of the allowed number of guesses (lose).

Master Mind, as programmed into the CSA Trainer, operates in a very similar manner to the original game. The color of the selected pegs has been translated to a number weight for a digit in the range of 0 to 9. To simulate selecting four colors, the Trainer system selects four digits, and arranges them in a set order within a row. For instance: should the Trainer select 3967, then the digits 3, 9, 6, and 7 would simulate the colors (r,o,b,g). The position of each digit within the row of digits (3967) is fixed, as it was with the colored peg positions in the original game. The user becomes Player Two or the challenger, and is allowed 99 "moves" or guesses to find the four digit code that exactly matches the Trainer's randomly selected four digit number. As in the original game, the display on the Trainer will provide clues, (see Figure 2-3), to aid in finding the correct solution. Suppose the CSA Trainer has selected 3967, as the secret number and the User enters 9217. In this example, digit 7 is in the correct position and is the correct weight, therefore one LED on the left of the display would illuminate. Looking again at the user's guess (9217) it can be seen that digits 2 and 1 do not match in any manner and they would be ignored (no LED effect). However, digit 9 is a correct weight but it is in the wrong position. An LED on the right of the display would indicate the relationship right weight, wrong position (digit 9) by illuminating.

NOTE

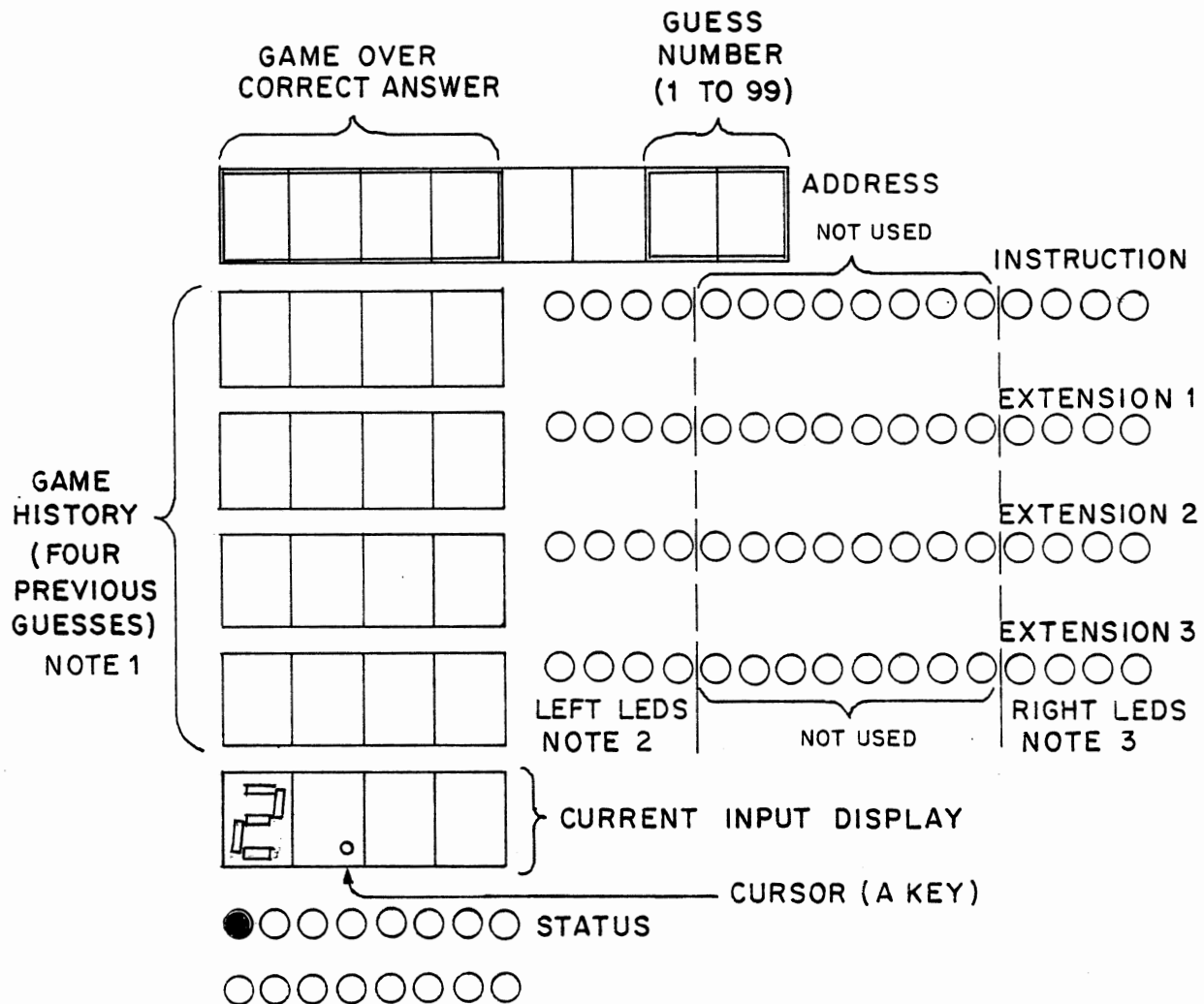
In the event that the Game Number (Trainer selected) contains identical digits (e.g. 9923 or 3999) and a single digit in the guess matches these duplicate numbers, then the RIGHT LEDs will indicate for all matches of right value/wrong position. See Games 2 and 3, Figure 2-3.

Going back to our original guess example (9217), the player can assume at this point, (one LEFT LED and one RIGHT LED illuminated) from these LED hints that:

a. One of the four input digits is correctly positioned and of the correct value, (LEFT LED indication), but in an incorrect position, (RIGHT LED indication).

b. One of the four input digits is of the correct value (weight) but in an incorrect position, (RIGHT LED indication).

The keyboard becomes the communication device between Player One (the Trainer) and Player Two (the User). A list of the keyboard entries, used during the game, is provided in Table 2-2. The next paragraph will present game playing procedures.



LED Examples:

Symbols: 0 = LED OFF • = LED ON

Game 1 No. = 3967
Guess 1 No. = 9217

LEFT RIGHT (LEDs)
•000 000•

Game 2 No. = 9923
Guess 2 No. = 6789

LEFT RIGHT (LEDs)
0000 00••

Game 3 No. = 3999
Guess 3 No. = 6789

LEFT RIGHT (LEDs)
0000 0•••

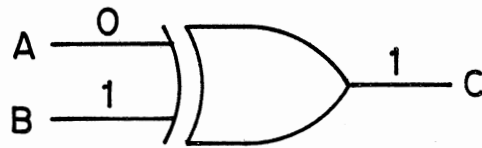
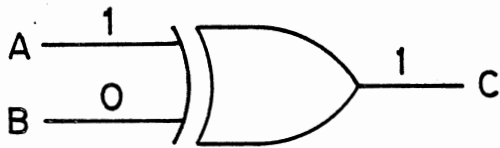
NOTES:

- #1. History (guesses) may be scrolled BACK and FWD (See Table 2-2)
- #2. LEFT LEDs indicate correct value, correct position
- #3. RIGHT LEDs indicate correct value, wrong position



MASTER MIND DISPLAY

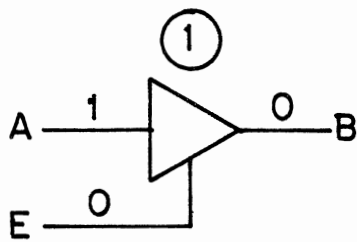
FIGURE 2-3



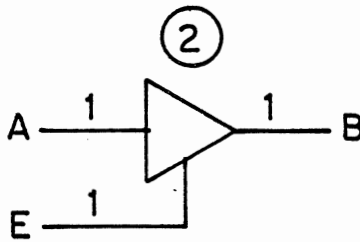
TRUTH TABLE

IN		OUT
A	B	C
1	0	1
0	1	1
1	1	0
0	0	0

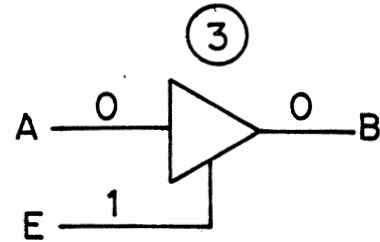
EXCLUSIVE OR GATE



DISABLED
OFF
HIGH Z



ENABLED
ON

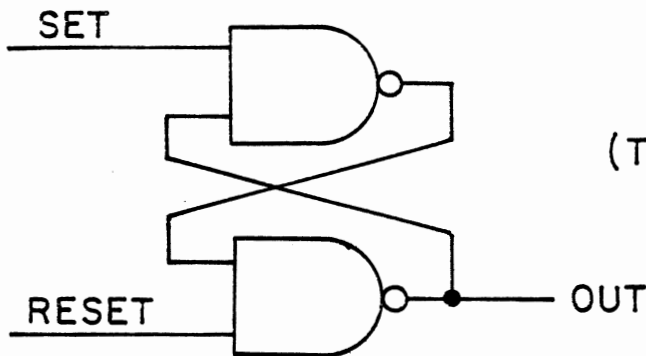


ENABLED
NO
OUTPUT

TRI-STATE

STATES

- ① OUTPUT LINE SHUT OFF / HIGH IMPEDANCE
- ② ACTIVE INPUT / ENABLE / OUTPUT
- ③ ACTIVE / ENABLE / NO INPUT / OUTPUT



FLIP - FLOP

(TRIGGERED BI-STABLE)

CSA COMPUTER SYSTEM ASSOCIATES

DIGITAL LOGIC/CIRCUITS

FIGURE 2-4

Table 2-2 Master Mind Keyboard Commands

Command	Description	Comment
ZERO (0)	Call Master Mind Program (ignore display)	Followed by <ENTER>
BREAK	End random number generation, store game number.	Game Start
RESET	End (anytime)	Petebug "68000 UP"
0 through 9 (digits)	Pressed to input each number digit	Four inputs then <ENTER>
BACK	Allows review of previous guesses	
FWD	Allows forward memory stepping	Cannot FWD beyond current guess position
A Key	Controls cursor Dot allows changing an input digit.	Appears to right of current position See Cursor (below)
C Key	Cancel current guess	Save guess count, begin input again
D Key	No Function	Do Not Use
E Key	Blank character (not allowed)	Do Not Use
HEX/BIN	Play again on game end.	See game Procedures
CURSOR	Dot, appears to right of hex digit display position, prior to input, then moves right one position.	Aid only, has no effect on input or game. See A key

GAME PLAY (MASTER MIND)

To play Master Mind on the CSA Trainer, proceed as follows:

1. Press RESET to clear the MPU registers and return to Petebug "68000 UP".
2. Press the ZERO (0) key, and then press <ENTER>.

NOTE

After <ENTER> has been pressed, the Trainer will enter a random number generation cycle; ignore the display during this cycle as presentations may appear erratic.

3. Press the BREAK key to end the random number generation and select the secret game number... This is the actual START of the game. At this time, the CSA Trainer has selected the four digit game number that the Player will try to guess.
4. Input a four digit guess number at the keyboard, see Table 2-2, for additional key inputs.
5. When four digits have been input, press <ENTER>. The Hex display will show the last guesses, (up to four) entered, see Figure 2-3. The display will also present the guess hints after each input guess, LEDs (LEFT) will indicate that a digit is correct number/correct position, (but not specifically which digit). LEDs (RIGHT) will indicate correct number/incorrect position, (again, not specifically which digit).

NOTE

These LED indications are better understood if interpreted as one of four, two of four, etc. LED indicators never relate to a specific digit value or digit position of the input, only one or more of four.

6. Continue to input number guesses, using the displayed hints to refine your inputs. Play continues until the secret number is matched by the input, or 99 moves (guesses) have been entered.

NOTE

When the game has been completed, see step 6, the input guesses may still be scrolled BACK and FWD, see Table 22. GAME OVER is indicated by the game number appearing in the Address hex space, (see Figure 23), or by the guess count reaching 99 and the game number displayed.

7. To play Master Mind again, press the HEX/BIN key, and allow the random number selection to proceed, (see note step 2), then repeat steps 3 through 6.

Master Mind may be aborted and a return to Petebug "68000

UP" may be performed at any time by pressing the RESET key. There are optional Master Mind keyboard inputs listed in Table 2-2 that may be very helpful toward winning the game. A read through of Table 2-2 could mean the difference between winning and losing. Also, to use the display hints to their full capacity the display presentation shown in Figure 2-3 should be understood. With an understanding of these aids, Master Mind is fun to play and your chances of winning are increased. Follow the procedures, study and understand Table 2-2 and Figure 2-3, and begin Play... GOOD LUCK!

SUMMARY

Chapter 2 has presented many new features and concepts of the CSA Trainer. If you have been following along and performing the recommended procedures, the Trainer should be powered up and tested. You should have an understanding of the Trainer's operation and a general knowledge of programs and routines. An example of the difference between a program and a routine has been demonstrated by running the DEMO routines and the Master Mind program. The User has been given the fundamental training to power up and operate the CSA Trainer and to use Petebug. Chapter 3 will describe more about the functional capabilities of the CSA Trainer and in Chapter 4 a detailed description of all Petebug commands and operations will be presented.

CHAPTER 3

INTRODUCTION

This chapter contains the functional descriptions of the CSA Trainer hardware. Tables and illustrations are provided to aid the User in locating various functional groups and components. In some instances, the functional operation of Trainer hardware is more related to the software operation. In those instances, reference to the appropriate chapter of this manual or to a specific support reference document has been included. CSA recommends that this chapter be used with all reference material and with an operating CSA Trainer to gain maximum training benefit from the material presented.

MAJOR ASSEMBLIES

The CSA Trainer contains three major assemblies:

- a. Power Supply
- b. MPU Board
- c. Keyboard/Display

Each of these major assemblies will be described separately and their contribution to the overall system (Trainer) function will be explained.

POWER SUPPLY

The Trainer power supply is a self contained unit that is replaceable as a part. The unit contains a switching type AC to DC rectifier and all filtering required to produce stable DC voltages and currents for Trainer operation. The power supply voltages are distributed throughout the Trainer by circuit traces and connectors and each assembly is equipped with an LED to indicate (ON) when DC power is applied.

The power supply will accept either one of two source input voltages:

- a. 90 to 130 VAC, single phase
- b. 180 to 260 VAC, single phase

at 47 to 450 HZ. The efficiency of the unit is rated at 70 to 80 percent, with an output ripple (high frequency noise) at no more than 10 mV RMS. The power supply may operate at a maximum of 50

watts continuous output for four voltage levels (+5VDC, -5VDC, +12VDC and -12VDC). See Table 1-2 for further specifications.

MPU BOARD

The MPU Board is the largest assembly in the CSA Trainer and is easily identified by the large MC68000/MPU component mounted in a Zero Insertion Force (ZIF) socket.

CAUTION

If the MPU is removed from the board, ensure that the component is properly oriented when reinstalled in the ZIF socket. Else severe damage to the component and the Trainer may occur.

The MPU Board contains two functional sections:

- a. The MPU section
- b. The peripheral support section.

The following paragraphs will describe each of these sections.

MPU Section

The MPU Section of the MPU Board contains the MC68000 and the Clock Interrupt Priority Encoder and the data and address line buffers, and MPU/signal bus interfacing.

The MPU (MC68000) is a 64 pin Integrated Circuit (IC) contained in a Dual In Line Package (DIP). The MPU is physically mounted on the board in a ZIF socket for easy removal and insertion.

The MC68000 is a 16 bit microprocessor containing seventeen, 32 bit general purpose registers and additionally a 32 bit PC register and 16 bit SR register. The seventeen general purpose registers are named D0 through D7 (data registers) and A0 through A6 (address registers). Two stack pointer registers (A7) named User Stack Pointer (USP) and Supervisor Stack Pointer (SSP) may also be used as general purpose registers. All seventeen registers may be used as index registers. A complete description of the MPU architecture is presented in Chapter 5 and the Motorola 16-bit Users Manual, Chapters 1 and 2.

The HALT and RESET lines of the MPU are connected to the circuit consisting of U30 and U31. The RESET line is taken from this circuit through the ribbon connector to the keyboard. See Schematic Diagram, rear of manual. A jumper connection from U31, through a resistor to ground is available to install a switch controlled MPU, RESET. To the right of the MPU (see Figure 3-1) are the address line buffer amplifiers (U5, U6, U7) and the data line receiver/transmitters (U8 and U9). Directly below the MPU are the clock crystals Y1 (4.9152 MHz) and Y2 (User option), and the clock amplifier (U43). The crystal (Y1) supplies the timing for all MPU functions and for the Band Rate generator for the RS-232 serial I/O. The Band Rate is jumper selectable (see Trainer Schematic, rear of manual) and frequencies are multiples of the clock frequency. A second crystal (Y2) may be optionally installed to alter the MPU clock rate but Y1 must always remain installed for RS-232 timing.

Above the MPU is the Interrupt priority encoder and the Interrupt Request (IRQ) jumper connections (0 through 7). See Motorola's 16 bit User's manual, page 61, for interrupt data. Of the eight IRQ connections, only number 7 is not available to the User as it is used for the abort function (BREAK). The VMA signal (for 6800 control) and the UDS and LDS signals are routed through buffer U3. The VMA and E (Enable) control lines are used to control the PIAs and ACIAs instead of employing DTACK. A close look at the MPU section of the schematic will reveal there are only 23 physical address lines. The MPU internally generates an A0 bit to control the UDS and LDS signals (see Table 3-1) thereby providing 24 lines of address. In addition to the gating and buffer circuits required for control signals, the MPU section contains one more important component. The Latch (U53) is used to send a bus error signal (BERR) to the MPU. This function will be activated when non-existing addresses are input on the keyboard (DTACK is not received). This is a jumper selected function and the jumper must be installed for proper Trainer operation. With the jumper installed, an address error as described above will cause the following:

- a. In Petebug - 68000 UP (RESET)
- b. In Tutor - All registers displayed

Table 3-1 Byte Addressing with A0 (bit)

MPU A0	Active		R/W	Select	
	UDS	LSD		D0-D7	D8-D15
1	No	Yes	R	Yes	No
0	Yes	No	R	No	Yes
1	No	Yes	W	*(Yes)	Yes
0	Yes	No	W	Yes	*(Yes)

*Temporary-may change in future.

- 1 See Motorola 16 bit Users Manual (Pages 38, 39 and 34)
- 2 See Chapter 5, READ and WRITE CYCLES, Pages 11-14

The MPU Bus and VERSAbus

The bus interface on the MPU Board connects the MPU to the peripheral section of the board and to the VERSAbus (see Schematic). The MPU bus lines are:

16 data lines D0-D7 and D8-D15
23 address lines A1-A8, A9-A16 and A17-A23

FC0-FC3 - Privilege State (Supervisor/User)
RESET - Clear Registers
HALT - Break/Abort
VPA - (6800) Address Line
VMA - (6800) Serial/Parallel I/O
E - (Enable) Control
UDS - A0 Bit - Upper Data Strobe
LDS - A0 Bit - Lower Data Strobe
AS - Address Strobe
DTACK - Data Transfer Acknowledge

NOTE

A jumper selection point is provided to set DTACK timing in accordance with the slowest device to be accessed (see Figure 3-1, U27).

CLK - Clock
BG - Bus Grant
R/*W - Read/not WRITE
VDD - +5VDC
IPL0, IPL1, IPL2 - Interrupt (Auto Vector Lines)
*BERR - Bus Error (jumper selected/hardware)
BGACK - Bus Grant Acknowledge
BR - Bus Request
VSS - Ground
VSS - Ground

These signals are used in the Trainer and are connected to the VERSAbus external wire wrap connector.

For a detailed description of these signals and their functions, see Chapter 5 and 16 bit Users Manual, Section 4. The Trainer uses the VMA and E signals for controlling serial I/O (ACIAs) and parallel I/O (PIAs) thereby not using the DTACK control signal. A diagram of the VERSAbus pin out and orientation is presented in Figure 3-2.

The MPU, Peripheral Section

Connected to the MPU by the on-board bus, this section of the MPU board contains components which are essential to practical use of the MPU. The major functions supported in the MPU peripheral section are as follows:

- a. Memory (RAM, EPROM and EEROM)
- b. Address Decoding (Field Programmable Gated Arrays, FPGAs)
- c. Serial I/O (ACIAs) and Band Rate Selection (Jumper/U35)
- d. Parallel I/O (PIAs)

Each of these MPU functional support circuits will be described in the following paragraphs.

Memory

The CSA Trainer has a unique and versatile memory hardware circuit. Each of the four pair of hardware sockets (8 total) may accept memory components in accordance with the jumper positions selected for the memories. The MPU Schematic diagram (rear of manual) shows the eight jumper positions available for each chip.

The memory sockets will accept devices such as 2716, 2732, or 2764 (ROM or EPROMs) and 6116, 6264 type CMOS RAMS. As these memory components are eight bit devices and the MPU bus requires 16 bit devices, pairs (2) of memory components are used. Figure 3-? shows the possible jumper selections available for memory devices in the Trainer. To prevent writing to a lower address RAM, when only an upper is to be written (8 bits), the UDS and LDS lines are gated with R/W before being sent to these devices. The memory in the Trainer may be delivered with or without options as shown in the memory map in Figure 3-4.

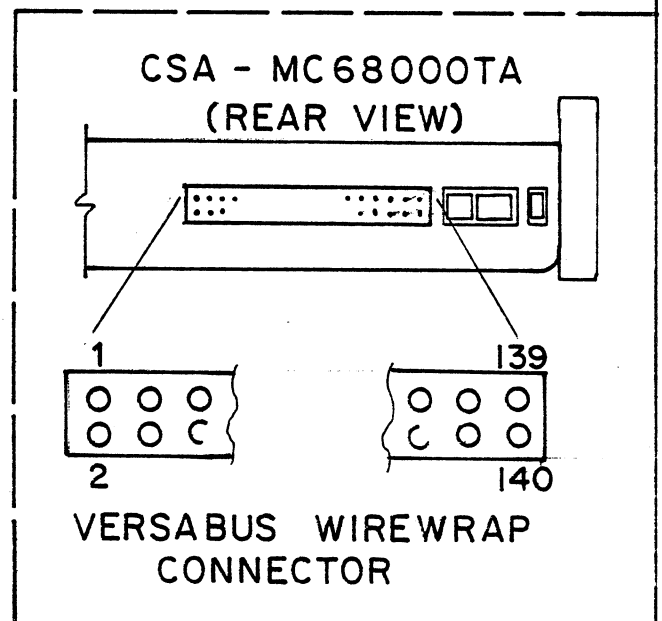
CAUTION

To remove or install CMOS RAM devices, ensure that the MPU board and the personnel are grounded. CMOS devices may be destroyed by static electricity in equipment or personnel.

The labels on the memory map describe the type of component, the address boundaries, the capacity (in kilobytes) and the component identity (UXX) The Petebug EPROM are described functionally in Chapters 2 and 4. The Tutor EPROM functions are described in the CSA Laboratory Manual, Appendix C and D and in this manual in Appendix A. The RAM Locations (addresses) and

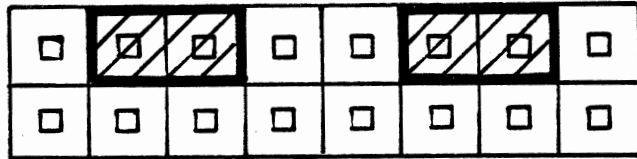
108	BR1
81	BERR
112	BBSY
30	BAS
99	BG1 IN
58	BA23
57	BA22
56	BA21
55	BA20
54	BA19
53	BA18
52	BA17
51	BA16
50	BA15
49	BA14
48	BA13
47	BA12
46	BA11
45	BA10
44	BA 9
43	BA8
42	BA7
41	BA6
40	BA5
39	BA4

38	BA3
37	BA2
36	BA1
34	BR/W
26	BUDS
25	BLDS
20	BD15
19	BD14
18	BD13
17	BD12
16	BD11
15	BD10
14	BD9
13	BD8
12	BD7
11	BD6
10	BD5
9	BD4
8	BD3
7	BD2
6	BD1
5	BDO
29	BDTACK
74	BRESET
70	CLK

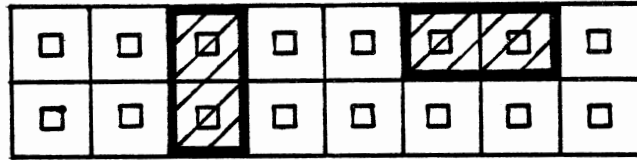


VERSABUS PINOUTS

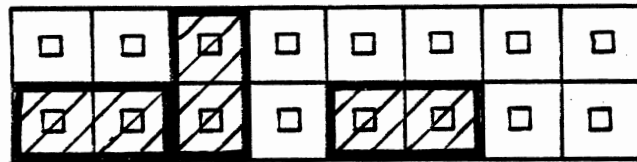
FIGURE 3-2



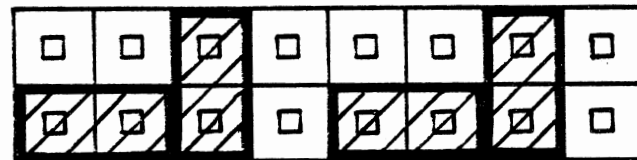
2716



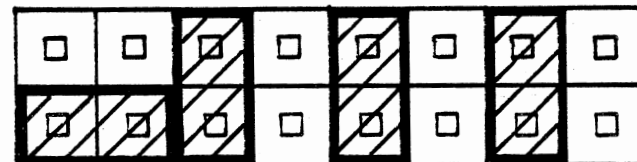
2732



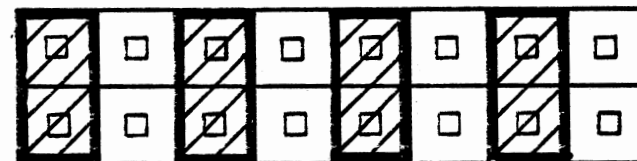
2764



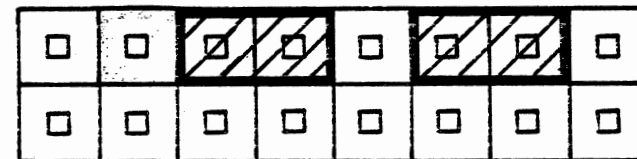
27128



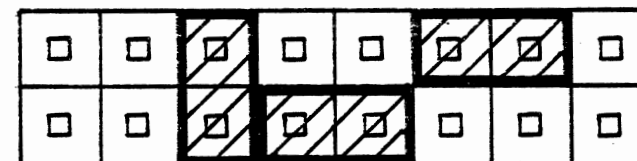
27258



27512



6116



6264

JUMPER CONFIGURATION

MEM. TYPE

MEM JUMPER SELECT

FIGURE 3-3

functions are described in the User programs in the CSA Laboratory Manual and through all references as used by the MPU. A list of designated addresses for the Trainer is shown in Table 3-2. At the rear of this chapter a description of Petebug subroutines is available. The optional EEROMs are described in the following paragraphs.

Optional EEROMs

Two optional memories (U51 and U52) are available for the CSA Trainer and are Electrical Erasable E2 ROM (EEROM). These type of components have the ability to retain their contents when power is removed (ROM) yet are programmable while in the system (Trainer). The Trainer is equipped with two ZIF sockets for these components. These EEROM memories may be programmed (SAVE) from RAM using Petebug or tutor. The User may also

program RAM using Petebug or Tutor. The User may also program RAM (LOAD) from the EEROMs with the monitor programs. There are certain parameters for working with the EEROMs that should always be observed to avoid problems:

1. The EEROMs will accept byte data, but always operate on a word (16 bits). Therefore all START, BYTE, and OFFSET inputs should always be an even number.
2. The User is responsible for recording how much memory is occupied and how much is free within the EEROM. If a program or data is resident in the first 4k of EEROM memory and an OFFSET of at least 4k is not input, the resident data will be written over by the new data and destroyed (lost).

Table 3-2 Trainer Designated Addresses

<u>Address</u>	<u>Description</u>
<u>Start--End</u>	
000000-000003	Contains address of initial trainer stack pointer
000004-000007	Contains address of trainer start up location
000008-0003FF	Contains 6800 interrupt vectors
From Pointer	Refresh buffer (located by Pointer - Petebug)
001000-003FFF	Memory available for user programs and data under Petebug
000F80-000FFF	Trainer stack area
FD0000	Keyboard PIA: A side data
FD0001	Display PIA: A side data
FD0002	Keyboard PIA: B side data
FD0003	Display PIA: B side data
FD0004	Keyboard PIA: A side control
FD0005	Display PIA: A side control
FD0006	Keyboard PIA: B side control
FD0007	Display PIA: B side control
FD0041	ACIA 1 Status and control register
FD0043	ACIA 1 Data register
FD0061	ACIA 2 Status and control register
FD0063	ACIA 2 Data register
FD8000-FDFFFF	The EEROM address space (2k actual)
FF0000-FF7FFF	Address space for Sockets U19-U20 (Tutor)

0 1 2 3	4 5 6 7	8 9 A B	C D E F
0000 0FFE	00FF E806	0000 0082	6000 07FB
STACK pointer	START UP	VERSION #	BRA START

80415 RTS 4E75
BRA0806

**CSA, Users Manual (CSA-UMM68KTA)
for, CSA TRAINER (CSA-M68000TA)
CHAPTER 3 Page 9**

Table 3-2 Trainer Designated Addresses (cont'd)

Address	Description
<u>Start--End</u>	
FF0000-FF0003	Contains address of Tutor initial stack pointer
FF0004-FF0007	Contains address of Tutor start up location
FF8000-FFFFFF	Address space for Sockets U17-U18 (Petebug)
FF8000-FF8003	Contains address of Petebug initial stack pointer
FF8004-FF8007	Contains address of Petebug start up location <u>00FFE806</u>
FF8008-FF800B	Contains trainer monitor version number <u>00FFEDDA</u>
FF800C-FF800F	Contains: <u>BRA START</u> <u>6000 07FB</u>
FF8010-FF8013	Contains: <u>BRA REFRSH</u> <u>6000 01C2</u>
FF8014-FF8017	Contains: <u>BRA DSPVAL</u> <u>6000 0248</u>
FF8018-FF801B	Contains: <u>BRA DSPLED</u> <u>6000 0276</u>
FF801C-FF801F	Contains: <u>BRA SCNKP</u> <u>6000 02FA</u>
FF8020-FF8023	Contains: BRA GNUM
FF8024-FF8027	Contains: BRA RPSAVE
FF8028-FF802B	Contains: BRA WPSAVE
FF802C-FF802F	Contains: BRA RTSAVE
FF8030-FF8033	Contains: BRA WTSAVE
FF8034-FF8037	Contains: BRA ERMCOM
FF8038-FF803B	Contains: BRA DEM01
FF803C-FF803F	Contains: BRA DEM02
FF8040-FF8043	Contains: JMP STEPPER
FF8054	CONTAINS: A/D CONVERTER
FF8058	CONTAINS: D/A CONVERTER

3. When moving data from RAM to EEROM the START address for a SAVE is the source data start address in RAM.
4. When moving data from EEROM to RAM the START address for the LOAD is the destination address in RAM.
5. The OFFSET address is always in EEROM and must be an even number. The User is responsible for keeping track of OFFSET addresses and programs resident in EEROM.

Operating in Petebug, the two EEROM commands are as follows:

5 key - SAVE command - Write to EEROM from RAM.

7 key - LOAD command - READ into RAM from EEROM.

These two Petebug commands are always followed by three prompts:

1. Start? (SAVE=SOURCE/LOAD=Destination, in RAM even number)
Input hex number - ENTER
2. Bytes? (Number of bytes to move, must be even number)
Input hex number - ENTER
3. Offset? (The "safe" address to start of program - even number - User supplied).
Input hex number - ENTER


The User responds to each prompt with the appropriate hex number, followed by ENTER. Upon completion of the response to the final prompt (OFFSET) and ENTER, the data transfer (LOAD or SAVE) will begin immediately. The transfer will take some time and the display will show the message "done" when complete.

U NO.	HEX	DEC	BLOCK
U23 AND U24	00 0000	0	16K
	4095 00 0FFF	4,095	
U21 AND U22	00 1000	4,096	8K
	12287 00 3FFF	16,383	
*	00 4000	16,384	15 MEG
	00 7FFF	32,767	
I/O ADDR	00 8000	32,768	32K
	FC FFFF	16,580,607	
U51 AND U52	FD 0000	16,580,608	16K
	FD 7FFF	16,613,375	
*	FD 8000	16,613,376	64K
	FD FFFF	16,646,143	
U19 AND U20	FE 0000	16,646,144	16K
	FE FFFF	16,711,679	
U17 AND U18	FF 0000	16,711,680	16K
	FF 7FFF	16,744,447	
	FF 8000	16,744,448	16K
	FF FFFF	16,777,215	


(END MEMORY)

MEMORY MAP


FIGURE 3-4

GATE		AS	A23	A22	A21	A20	A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	
ACT LEV		I15	I14	I13	I12	I11	I10	I9	I8	I7	I6	I5	I4	I3	I2	I1	I0	
F0	H	H	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	
F1	H	H	H	H	H	H	H	H	L	H	L							
F2	H	H	H	H	H	H	H	H	H	H	H							
F3	H	H	H	H	H	H	H	H	H	H	L							
F4	H	H	L	L	L	L	L	L	L	L	L	H						
F5	H	H	L	L	L	L	L	L	L	L	L	L						
F6	H	H	H	H	H	H	H	H	L	H	L	L	L	L	L	L	L	
F7	L	H	H	H	H	H	H	H	L	H	H							EEROM
F8	L	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
INPUT $I_m \Rightarrow H$ $\bar{I}_m \Rightarrow L$ DON'T CARE $\Rightarrow -$																		
GATE ACTIVE HIGH $\Rightarrow H$ ACTIVE LOW $\Rightarrow L$												82S103 (T.S.)						<input checked="" type="checkbox"/>

U25 (FPGA NO. 1)

GATE		F6	F5	F4	F3	F2	F1	F0	A8	A7	A6	A5	A4	A3	A2	A1		
ACT LEV		I15	I14	I13	I12	I11	I10	I9	I8	I7	I6	I5	I4	I3	I2	I1	I0	
F0	H							H	L	L	L	L	L	L			<input checked="" type="checkbox"/>	FU/IO
F1	L							H	L	L	L	L	L	L				START
F2	L					H												PETEBUG
F3	L				H													TUTOR
F4	L			H														OPTION
F5	L		H														L	RAM
F6	H	H							L	L	L	L	L					PIA
F7	H	H							L	L	H							ACIA
F8	L						H		L									VPA
INPUT $I_m \Rightarrow H$ $\bar{I}_m \Rightarrow L$ DON'T CARE $\Rightarrow -$												82S102 (O.C.)						<input checked="" type="checkbox"/>
GATE ACTIVE HIGH $\Rightarrow H$ ACTIVE LOW $\Rightarrow L$																		

U26 (FPGA NO. 2)

- FUNCTION/INPUT OUTPUT
-  INPUT (ADDRESS) VARIABLE

BRA

01100000 8bit displacement
16 bit displacement

60

JSR

0100 1110 10
4 E 2

Transfer of sixteen bit data to eight bit devices requires special circuitry and programming which directs the MPU to make the appropriate data transfer. The EEROM devices have the capacity to store 2000 (eight bit) data bytes. Each EEROM has this capability. By using two EEROMs (U51, U52), the Trainer increases the EEROM memory capacity to 2000, 16 bit words. However, the software transfer routines must direct the MPU to store eight bits of each word from RAM (16 bits) into each EEROM. The reverse is true when loading RAM from EEROM, half of each 16 bit word (8 bits) must come from each EEROM.

While operating in Tutor, the EEROM commands are:

- LE - Similar to Petebug LOAD
- PE - Similar to Petebug SAVE

The three prompts and parameters are the same as for Petebug but the data transfer will be serial I/O.

In Tutor, the RTSAVE and WTSAVE (READ/WRITE EEROMs respectively) may be called with the GO (G) or GO Direct (GD) commands to transfer data to and from Tutor. See the CSA Laboratory Manual and Table 4-1 for details.

The EEROM circuitry can be accessed directly in software. To read the EEROM, one need only read the address, as there is no distinction between EEROM and any other memory. Writing to the EEROM is more difficult.

In order to write a word (and it must be done word at a time) to the EEROM, there are two very similar steps. They both involve setting a flip flop to allow an extended write, doing the write, delaying and resetting the flip flop. The first time it is done is to erase the byte in each EEROM, and the second time is to write the data.

For each of the above two cycles, the following procedure is followed: First the flip flop needs to be reset, and then allowed to be set. This is done by setting CA2 of the Keypad PIA high and then low. Next write the data to the word in the EEROM, which will be held until the flip flop is reset. Now delay for 1 millisecond or 10 milliseconds depending on the type of EEROM. Finally, raise CA2 of the keypad PIA to high to complete the cycle. This cycle needs to be done twice for each word, as stated above.

There are four subroutines in Petebug, accessible through the addresses in Table 2-1. The key to using these routines is the following codes:

W = WRITE to EEROM
R = READ from EEROM
T = Tutor
P = Petebug

The four subroutines are WTSAVE, RTSAVE, WPSAVE and RPSAVE. These subroutines will present the three prompts (START, bytes, OFFSET), do the transfer, and return. To be safe, presume that all registers are used or effected by these subroutines. See the descriptive list of Petebug subroutines at the rear of this chapter.

Address Decoders (Field Programmable Gated Arrays, FPGA's)

The Trainer employs two FPGA's (see Trainer Schematic at U25 and U26) to decode addressing from the MPU to memory and devices. The capabilities of FPGA include the asset that the devices (82S103's) are reprogrammable. The User may modify the allocation of addresses (see memory map) to suit his needs. The FPGA's used in the Trainer receive 16 input signals which are decoded to 9 output signals. Figure 3-5 is a program chart for each FPGA and Table 3-3 is a list of the input signals and pin connections for each FPGA.

In the following description FPGA/U25 will be FPGA #1 and FPGA/U26 will be FPGA #2. Refer to the tables and program charts (above) for clarity. An address is decoded as follows. When an address is presented to this section of the board by any valid bus master, and the Address Strobe (AS) is asserted (brought low), the AS signal is at the address inputs of both FPGA's. If the high order address bits, combined with the Address Strobe match any pattern programmed into FPGA #1, then one of its outputs will go from low to high. The outputs from FPGA #1 (F0 through F6) are connected to the input lines of FPGA #2.

Table 3-3 FPGA Input Signals

*Reference Schematic, Trainer MPU Board

FPGA (U25, 82S103)

(I) Input	Signal (From)
I15	Address Strobe (active true)/high
I14	A23 (address line)
I13	A22 (address line)
I12	A21 (address line)
I11	A20 (address line)
I10	A19 (address line)
I9	A18 (address line)
I8	A17 (address line)
I7	A16 (address line)
I6	A15 (address line)
I5	A14 (address line)
I4	A13 (address line)
I3	A12 (address line)
I2	A11 (address line)
I1	A10 (address line)
I0	A9 (address line)

Table 3-3 FPGA Input Signals (cont'd)

FPGA (U25, 82S103)

(I) Input	Signal (From)
I15	Output F6 from FPGA (U25)
I14	Output F5 from FPGA (U25)
I13	Output F4 from FPGA (U25)
I12	Output F3 from FPGA (U25)
I11	Output F2 from FPGA (U25)
I10	Output F1 from FPGA (U25)
I9	Output F0 from FPGA (U25)
I8	A8 (address line)
I7	A7 (address line)
I6	A6 (address line)
I5	A5 (address line)
I4	A4 (address line)
I3	A3 (address line)
I2	A2 (address line)
I1	A1 (address line)
I0	Output F0 from FPGA (U26) 1

1 Used for disabling addresses 0-7 (RAM)

Table 3-4 FPGA Outputs

U25

F0	U26
F1	U26
F2	U26
F3	U26
F4	U26
F5	U26
F6	U26
F7	EEROMs (U51, U52)
F8	N.C.

U26

F0	To Io U26 (output to input)
F1	START /Petebug (Jumper Select)
F2	(U17, U18) /Petebug Program
F3	(U19, U20) /Tutor Program
F4	EPROM (U21, U22) /8K Option
F5	RAM (U23, U24) /User
F6	- PIA (U36, U37) /Parallel I/O
F7	- ACIA (U33, U34) /Serial I/O
F8	- VPA /6800 device (PIA's/ACIA's)

When a high output from FPGA #1 (coupled and input to FPGA #2) forms a programmed match with the low order address bits (A1 through A8), FPGA #2 will generate a valid address output. The asserted state of the FPGA #2 output is programmed for each type of device as follows:

- a. RAM and EPROM memory - Active Low
- b. PIA's, ACIA's - Active High
- c. VPA - Active Low (6800 device I/O)

Trainer Initial Start and RESET

When the MPU is initialized (power up or RESET), it fetches the contents from the memory address in the PC register. These initial addresses are a function of the internal MPU and are called the start vector. The addresses of the start vector are 0 to 7 of the RAM (see memory map). The start vector could be in ROM, but this would interfere with changing start and trap vector addressing. For versatility and latitude in addressing, the first 1000 (1K) words of addresses should be RAM addresses.

The memory map shows Trainer RAM address (000000 to 0003E8) as reserved for vectors and MPU functions.

The problem arises, with the use of RAM vector addresses, that there is no immediate data available if power has been removed. The MPU will fetch from RAM for vector start, and find no jump instruction to begin processing. The CSA Trainer solves the RAM versus ROM start problem in the following manner:

- a. The Trainer hardware is designed to make a double address fetch.
- b. Through jumper selection (see (Trainer Schematic, U25 F1/out and U29/in), three options are available for start vector:
 - 0 - Petebug
 - 1 - Tutor
 - 2 - User (specified)

These jumper connections (SEL) are on the MPU board and the Trainer is configured for Petebug, when shipped, unless otherwise requested.

To accomplish vector start, the FPGA's are programmed to make a double fetch. The initial fetch by the MPU to RAM (0 to 7) is decoded and output on FPGA #2, F0. The F0 output is wired to FPGA #2's Io input, simulating a second address fetch, the resultant output from FPGA #2, F1 is OR gated by jumper selection to the appropriate ROM or User start address. The decoding and

feedback of F0 causes the addresses in RAM (0 to 7) to be effectively shut off from MPU access. This allows the ROMs selected to simulate the first eight bytes of RAM and send the start instructions to the MPU.

FPGA's and DTACK

When any of the memory devices are selected, the Data Transfer Acknowledge (DTACK) circuit is enabled. DTACK provides the MPU with the cycle end signal that the data operation has been performed. However, the device addresses employ the VPA signal, together with the VMA and E signals for control thereby avoiding the use of DTACK.

There is a jumper selection for the DTACK signal, that allows the User to lengthen DTACK time for slower memory chips and devices (see Trainer Schematic, at U27). The eight (8) jumper positions will sequentially (1-8) lengthen the DTACK time period. When setting this jumper, it should be adjusted for the slowest rated (time) device that the MPU will interface with.

FPGA's and EEROM

The EEROM's require a special technique for READ and WRITE that requires a time interval much longer than DTACK can provide. The EEROM circuit is addressed from FPGA #1, F07. The EEROM circuit consists of two EEROM's (U51, U52) parallel input/output (I/O) and a flip-flop (U54)

The EEROM WRITE signal must stay asserted (low) for 1 to 10 milliseconds (MS). The flip-flop is used to hold the WRITE line low until parallel I/O resets the flip-flop and holds it high. The flip-flop (U54) is also RESET by the RESET line when asserted. The circuit operation is started by signals from FPGA #1, F07 (inverted) to U32 and DTACK from U27. The flip-flop (U54) is set, and remains set until PIA, U36, CA2 (parallel I/O) or RESET are received at U32 (bottom), inverted and coupled to U54.

FPGA's and I/O Devices

The decoding for the I/O devices needs a bit of explanation. The 68000 processor supports special cycles for 6800 type devices (like the ACIA and PIA). In order to activate these cycles, the VPA line of the processor is asserted (brought low), rather than having a DTACK signal returned. In order to facilitate easy experimentation with the Micro 68000 in terms of adding I/O devices, the range of memory addresses for which 6800 type cycles are run was made much larger than the space used by the ACIA's and PIA's alone. This requires that the PIA's and ACIA's be fairly close to each other in address range, and there

MUST be a VPA decoded in the space decoded for those parts. The VPA will NOT be generated by other hardware. A good point to remember when reprogramming the FPGA's.

FPGA Summary

The only restrictions to FPGA reprogramming, other than those in the preceding paragraphs, are that there are only seven output lines used to decode 8 functions. The functions are shown in Table 3-4, FPGA outputs. As mentioned previously, the FPGA's may be reprogrammed - provided that all memory and device requirements are adequately understood and provided.

Parallel Input/Output (I/O)

The parallel I/O consists of two Parallel Interface Adapters (PIA's), which are dual 8 bit configurable parallel I/O devices. These use the VPA/VMA cycles of the 68000, and are decoded by the address decoding. The I/O pins go to the 50 pin connector, and if the keyboard is not being used, then these lines may be used for parallel I/O for a control application. See Petebug subroutines, Stepper Motor at rear of this chapter.

The parallel I/O on the Trainer is controlled by the VPA, VMA and E control signals and the PIA interfacing. As used in the Trainer, the PIA's (U36, U37) are similar to 6800 devices.

Serial Input/Output (I/O)

The serial I/O consists of 2 Asynchronous Communications Interface Adapters (ACIA's) and associated circuitry for RS232 levels, and a baud rate generator. The two serial devices are accessed the same way as the PIA's, except that the fifth address line is used to select between them, as they both use the same data lines. For each ACIA, there are five RS232 level buffers, these are for the following signals:

- a. Transmit data
- b. Receive data
- c. Carrier detect
- d. Request to send
- e. Clear to send

These signals go to two on board 25 pin RS232 compatible connectors. The configuration of pins on the TERM port is set up to interface to DTE (data terminal equipment, most terminals), while the HOST port is set to interface to DCE (data communications equipment, most computers). Baud rates from 300 to 38,400 are available as jumper selections (see Trainer Schematic, U35). These baud rate parameters are generated by the CLOCK Crystal (Y1). Baud rate is individually selectable for each ACIA (U33, U34).

THE KEYBOARD/DISPLAY ASSEMBLY

Although the keyboard and display circuits share a common assembly board, each of these circuits operates independently of the other. The keyboard/display assembly is connected to the MPU assembly through a three connector (one unused) ribbon cable with 50 pin edgeboard connectors on each end. The signals to the keyboard/display assembly are parallel I/O and are processed by the PIAs on the MPU board. PIA, U36 processes signals for the keyboard and PIA, U37 processes signals for the display. The CSA Laboratory Manual, Appendix E-2 and E-3 show the signal paths of the PIAs, ribbon connectors, and circuit of the Display (E-2) and Keyboard (E-3). The following paragraphs will discuss the keyboard circuits and the display circuits, each will be discussed separately.

The Keyboard

The CSA Trainer keyboard consists of twenty keys, arranged in a four horizontal row by five vertical column matrix. The keys are labeled as shown in Figure 3.6.

The input (B side) of the keyboard is connected to a PIA and receives output signals from the MPU board. The output (A side) of the keyboard is connected to the same PIA and transfers signals to the input of the MPU board. (See Figure 3.7).

The BREAK and RESET keys are connected as interrupts and are not connected into the keyboard matrix. The following description of keyboard signals refers to matrix keys, the BREAK and RESET keys are not included.

The MPU, directed by Petebug's keyboard scan routine (SCNKP), outputs a negative true low to each of the four B side columns of the keyboard. These outputs from the MPU are sequentially sent to each of the four columns in sequential order and many times a second (loop). If no key is pressed, there will be no output. As shown in Figure 3.7 (detail), each column/row interconnect is a normally open keyswitch.

When a key is pressed, the input from the column is connected to the output from the row of the pressed key. This output from the A side of the keyboard is connected to the PIA, inverted, and input to the MPU. The keyboard to MPU inputs are decoded in the following manner:

Row inputs (key pressed) to the MPU.

Bit 0 on = top row keys ENTER through RESET

Bit 1 on = next row down keys C through F

Bit 2 on = next row down keys 8 through B

Bit 3 on = next row down keys 4 through 7

Bit 4 on = bottom row keys 0 through 3

Column inputs (column scanned) from the MPU.

Bit 1 off = scan the 0 through ENTER column

Bit 2 off = scan the 1 through H/B column

Bit 3 off = scan the 2 through BREAK column

Bit 4 off = scan the 3 through RESET column

The keyboard is the Trainer/User interface for MPU input. The inputs on the keyboard are processed by the MPU and Petebug subroutines. The User receives immediate feedback on the display of the keys that he has pressed. The display circuits are described in the following paragraph.

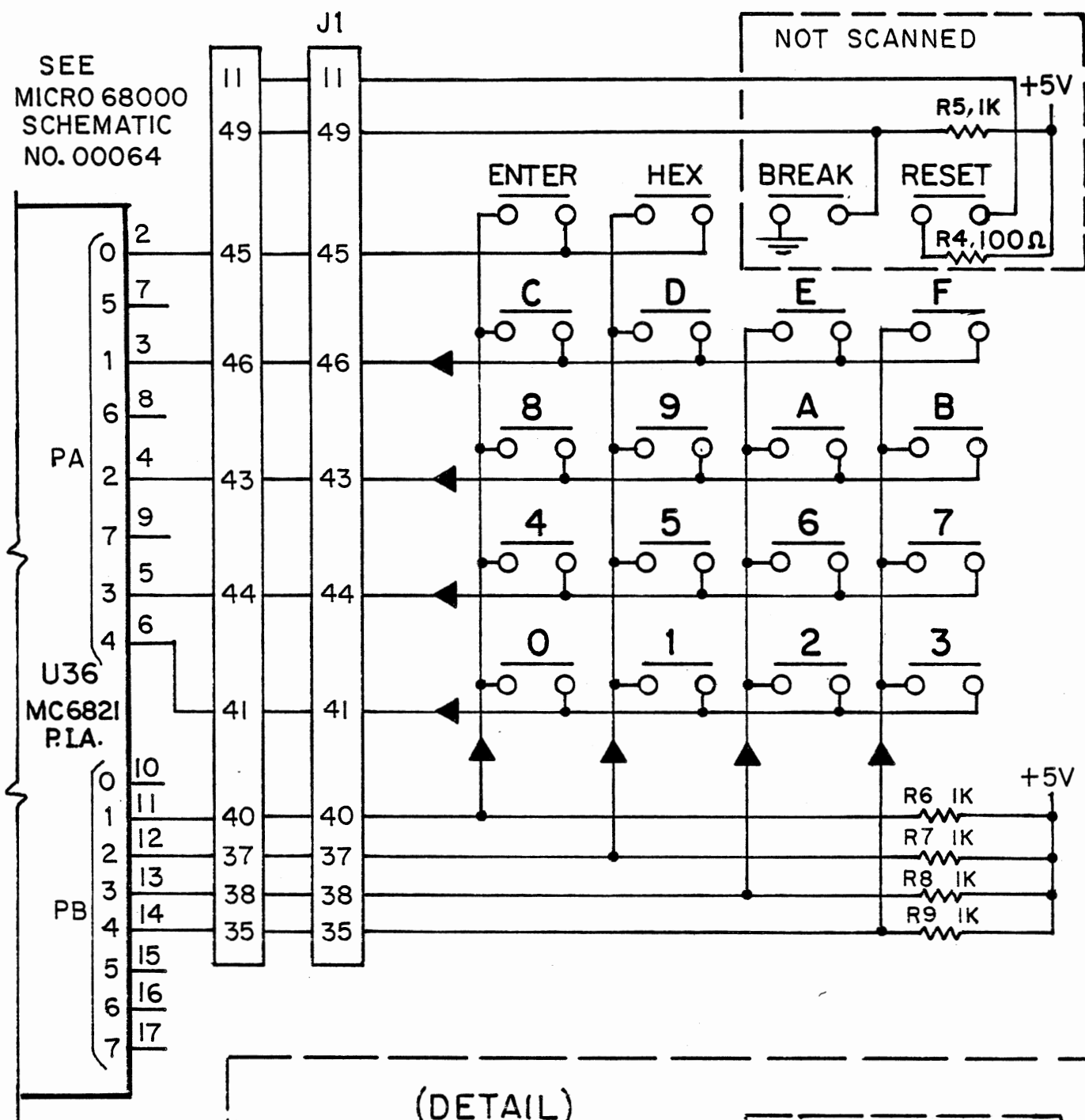
ENTER	HEX BIN	BREAK	RESET
CHG C	DSPY D	E	FWD F
RUN 8	STEP 9	AUTO A	BACK B
← 4	5	→ 6	7
0	+ 1	- 2	MODE 3

CSA  **COMPUTER SYSTEM ASSOCIATES**

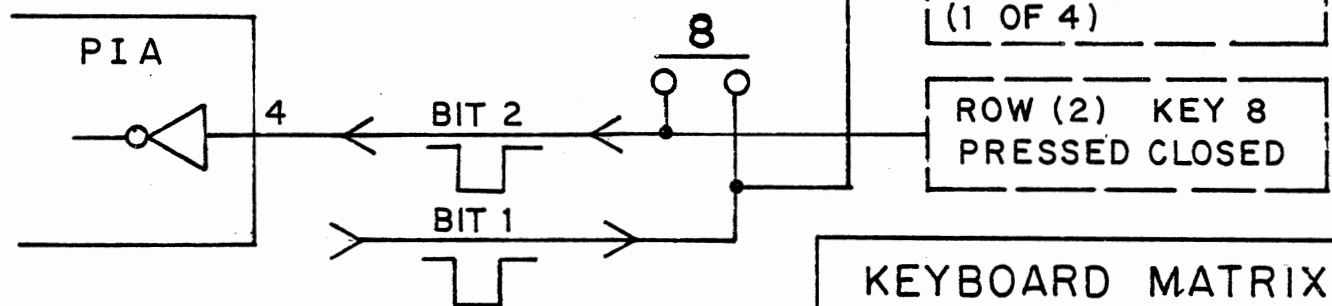
KEYBOARD KEYS

FIGURE 3-6

SEE
MICRO 68000
SCHEMATIC
NO. 00064



(DETAIL)
(OUTPUT BITS 0-4)



KEYBOARD MATRIX

Displays

The CSA Trainer has two types of displays (hex and LED) that share the same circuitry. The hex display components are common cathode type, seven segment displays with a decimal point. See Figure 3.8.

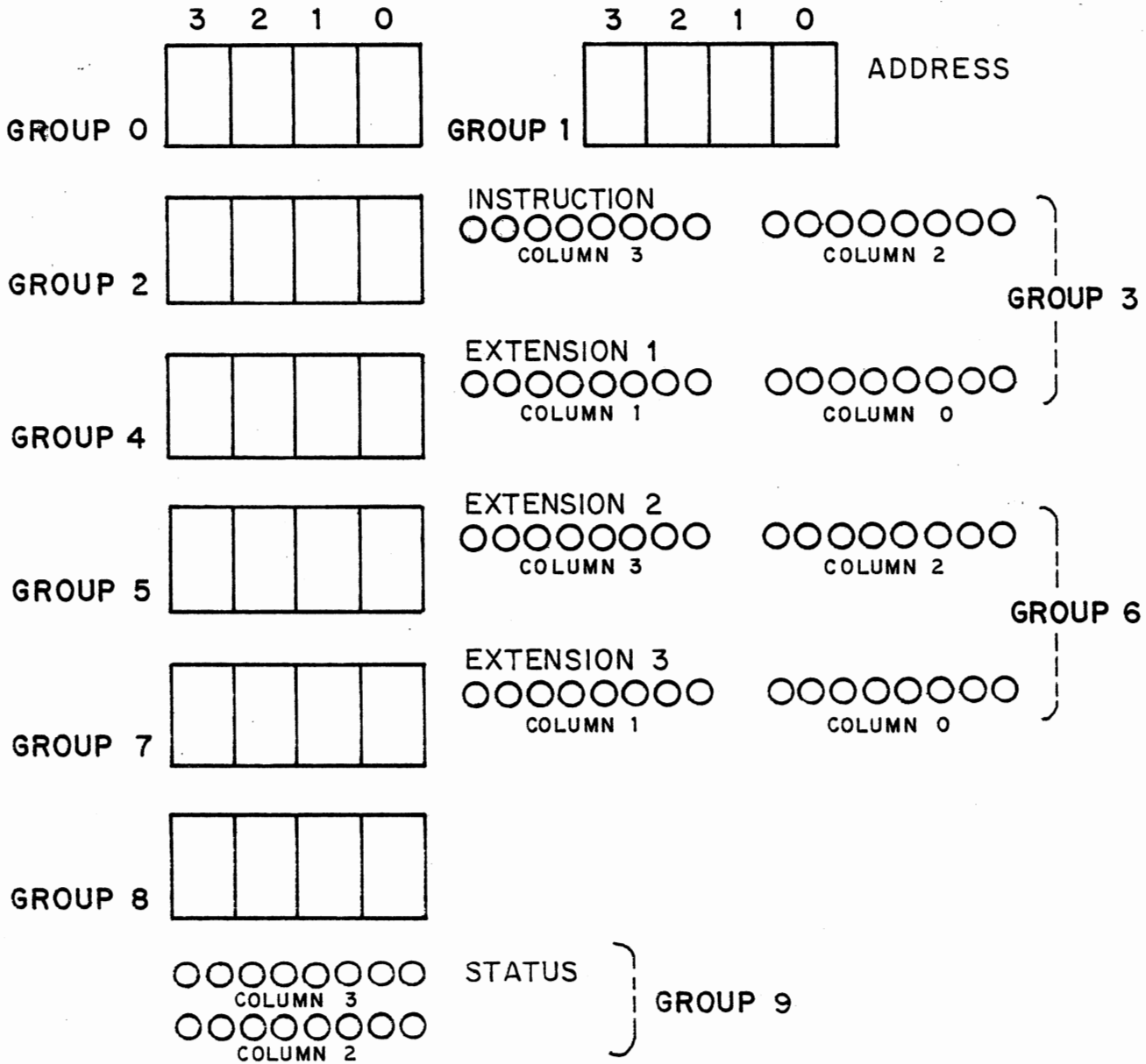
The displays are constantly scanned (data input) by row and strobed (power input) by column. The MPU, directed by the Petebug subroutines (REFRESH, DSPVAL, DSPLED) controls the displays on the Trainer. For a detailed description of operation of the seven segment and LED displays, see the CSA Laboratory Manual, Page 23. Also to understand the bit pattern of the display data output byte, see the CSA Laboratory Manual, Page 25.

A block diagram (Figure 3.9) of the display circuits is shown at the rear of this chapter. Notice in the diagram that the data is presented at each Octal Latch (U1 through U10), but only those latches that are enabled by the multiplex decoder (U11) and the scan drivers (U13, U14) will output to the displays. The column power drivers (Q1 through Q4) are also turned on/off (strobed) by the MPU/PIA input to the Decoder (U12) and the four strobe driver lines (U15). Multiplexing the display is a time-sharing technique used to save on power consumption. If each display were to remain lighted (constant) when data was displayed, each would need its own driver. The cost in components and power would be expensive. By multiplexing the relatively high transistor output power, the display can be pulsed at a rapid rate saving on parts and power. When the rate of strobing is high enough (100 hz or more), the displays appear to be lighted continuously. The REFRESH subroutine in Petebug directs the MPU to continuously refresh (strobe) the displays. A table of Petebug routines is presented in Chapter 2, with the start addresses for each routine.

The data to be displayed is stored in an area of RAM memory referred to as the Refresh Buffer. This data is changed by keyboard input and MPU output. A pointer in Petebug locates this data for the MPU during REFRESH.

As there are only enough latches to hold one column of data, the display must be continuously refreshed in order to keep a pattern displayed in all the seven segment and LED displays. If this is not done, only the last column selected will be displayed. In addition to this, unless the updating of the displays is done in an extremely careful fashion, bleed through (interference from one display element to another) will result. What follows is a description of how to refresh the displays so that bleed through does not occur:

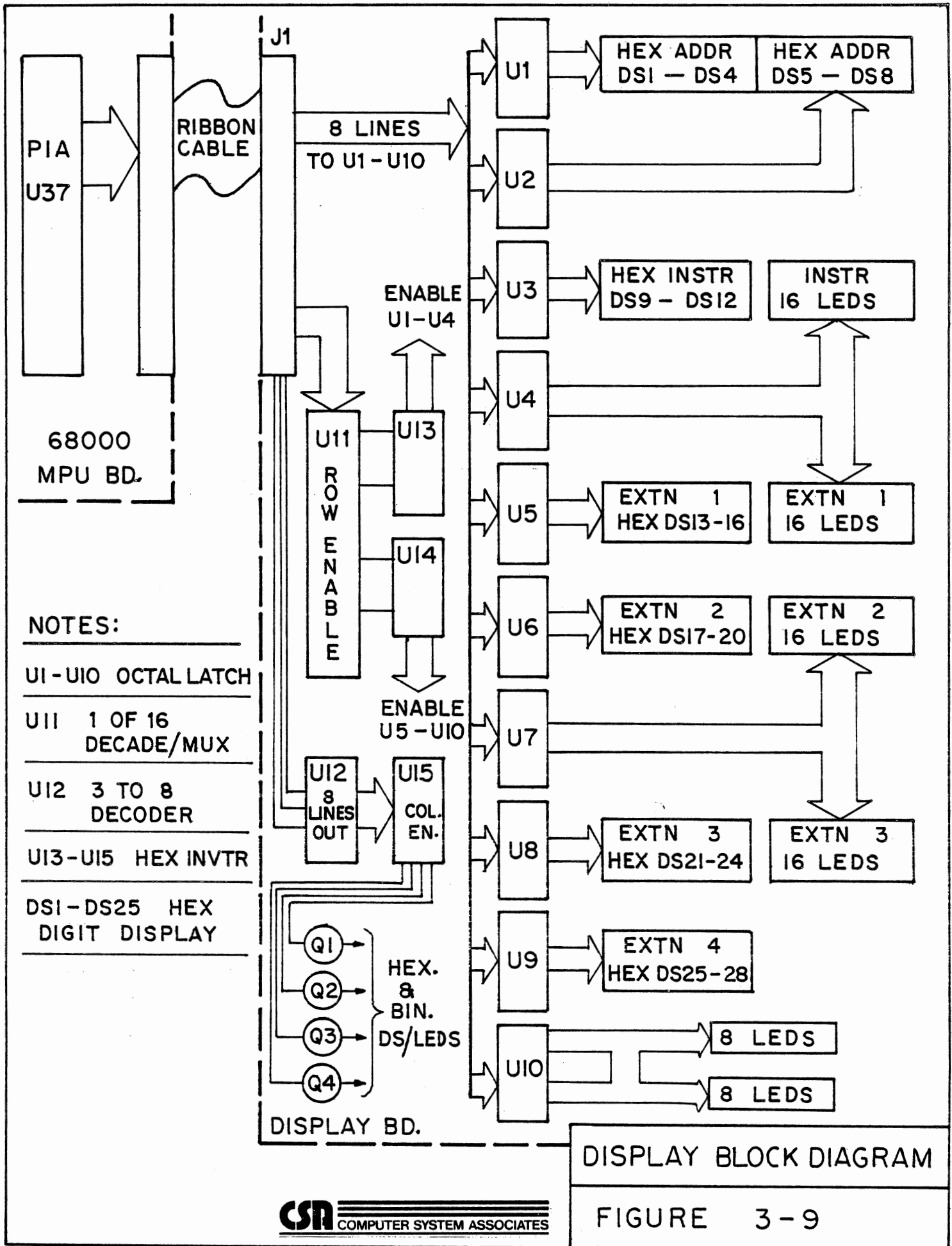
1. Set both the A and B sides of the display PIA for output.
2. The current values for all data to be displayed should be in a refresh buffer.
3. The refreshing of the displays will be done a column at a time. So the following steps will be repeated four times, once for each column. Column numbers are 0 through 3.
4. Merge in the blank display bit into a byte which contains the current column number (in the column number field bits 5 through 4) with a group select of group zero.
5. Loop through the group numbers (0 through 9) setting the group number in bits 3-0 of the value to select the column and group.
6. Output the data for the selected column and group to the A side of the display PIA.
7. Output the group and column along with the display blanking bit on to the B side of the display PIA. This latches the data into the desired location to be displayed.
8. Output a value to the B side of the display PIA to select an illegal group with the display blanking bit still set. (The value 4F hexadecimal works fine). This keeps the next data which will be output from overwriting that which we just output.
9. Output the value for the last legal group and column with the blanking display bit turned off. This enables the display.
10. Delay about 25 milliseconds to enable displays to reach brightness.
11. Continue with loop or return when all groups and columns have been refreshed.
12. End.



CSA COMPUTER SYSTEM ASSOCIATES

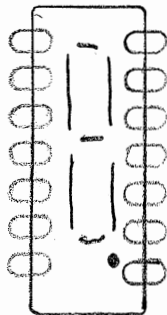
DISPLAY LAYOUT

FIGURE 3-8



PETEBUG SUBROUTINES

1.	REFRESH	Page 24
2.	DSPVAL	Page 24
3.	DSPLED	Page 25
4.	SCHNKP	Page 25
5.	GNUM	Page 25
6.	DEMO1/DEMO2	Page 26
7.	STEPPER MOTOR	Page 26



Using Petebug Subroutines

As has been seen, both scanning the keyboard and refreshing the displays are non trivial matters. Fortunately the User of the 68000 Trainer has the Petebug monitor available to scan the keyboard and refresh the display. What follows is a list of subroutines which enable the User to easily access the 68000 Trainer keyboard and displays.

Refresh the Displays

```
JSR          $FF8010          CALL REFRSH SUBROUTINE
```

This subroutine must be called continuously if all the displays are to remain on. There is an area of RAM memory called the refresh buffer. This memory contains the values to be displayed. The refresh subroutine takes the values from the refresh buffer and outputs them to the displays in the fashion described in the CSA Laboratory Manual (Pages 23, 25).

Note: prior to calling this subroutine, the User must have placed the values to be displayed in the refresh buffer. To change the values being displayed by this subroutine, one need only change the values in the refresh buffer. The following registers are used by the REFRSH subroutine: D0-D3 and A0-A2.

Display Value in 7 Segment

```
MOVE.L      VAL,D0          VALUE TO DISPLAY  
MOVE.L      OFFSET,D1      OFFSET INTO REFRESH BUFFER  
MOVE.B      NHDIGIT,D2     NUMBER OF HEX DIGITS TO DSPLY  
  
JSR          $FF8014          CALL DSPVAL SUBROUTINE
```

This subroutine allows the User to display a numeric value as hexadecimal digits in the seven segment displays. The first argument placed in D0 is the actual value to display. The second argument placed in D1 is the offset into the refresh buffer where the first digit of the number of digits to be displayed will be placed. The third argument placed in D2 is the number of hexadecimal digits to display at the specified location. The value will be displayed right justified and zero filled to the left. Note: care must be taken to insure that the number of digits specified from the initial offset does not cross into an area of the display which contains LEDs instead of seven segment displays. The following registers are used by the DSPVAL subroutine: D0-D5 and A0-A1.

Display value in a set of LEDs

MOVE.B	VAL,DO	BYTE VALUE TO DISPLAY
MOVE.L	OFFSET,D1	OFFSET INTO REFRESH BUFFER
JSR	\$FF8018	CALL DSPLED SUBROUTINE

This subroutine places the byte value given in D0 in the refresh buffer at the location specified by the offset given in D1. This subroutine reverses the bits so that the value will be displayed in a logical fashion (see description of how values are displayed in the LEDs in CSA Laboratory Manual).

Note: care must be taken not to specify an offset to a seven segment display instead of a group of LEDs. The following registers are used by the DSPLED subroutine: D0-D4 and A0.

Scan Keyboard

JSR	\$FF801C	CALL SCNKP SUBROUTINE
-----	----------	-----------------------

This subroutine scans the keyboard to see if the User is striking any one of the keys. It returns a byte value in D0. If the value \$FF is returned, it means that no key or multiple keys have been hit by the User. Otherwise the values 0-F reflect that the User hit one of the keys 0-F, the value \$10 means the User hit the ENTER key and the value \$11 means the User hit the HEX/BIN key.

Note: the BREAK and RESET keys cause interrupts and therefore cannot be detected in this fashion. This subroutine does not wait until the User strikes a key. It will always return immediately. The following registers are used by the SCNKP subroutine: D0-D4 and A0-A1.

Get Hex Number From Keyboard (Displaying Digits as Input)

MOVE.L	OFFSET,D0	OFFSET INTO RFBUF FOR DISPLAY
MOVE.B	MXDGIN,D1	MAX NO. OF DIGITS TO INPUT
JSR	\$FF8020	CALL GNUM SUBROUTINE

This subroutine enables the User to collect a hexadecimal number from the keyboard while at the same time displaying each hexadecimal digit as it is input. The first argument in D0 is the offset into the refresh buffer for the area to display the digits as they are input. The second argument in D1 is the maximum number of digits to allow the User to input. This subroutine will return the value collected from the keyboard in D0. It will also return with the condition code set to non-zero if the User inputs a reasonable number or with the condition code set to zero if the User inputs an unreasonable number (too long

or included the HEX/Bin key). The User terminates the number by striking the ENTER key. The GNUM subroutine uses the following registers: D0-D7, A1-A2 and A5-A6.

The Demonstration Programs

The demonstration programs are listed in the Chapter 2. In all revisions of Petebug (after ver 8.5) the jump addresses for these demonstration programs will remain constant. See Table 2-1, for address and description.

The Stepper Motor

The Trainer contains a program to control an optional stepper motor, and the address of that program is also in the jump table of the prom. This is only true for Petebug revision 8.7 and later. The program is described in the following paragraphs.

Stepper Motor Program

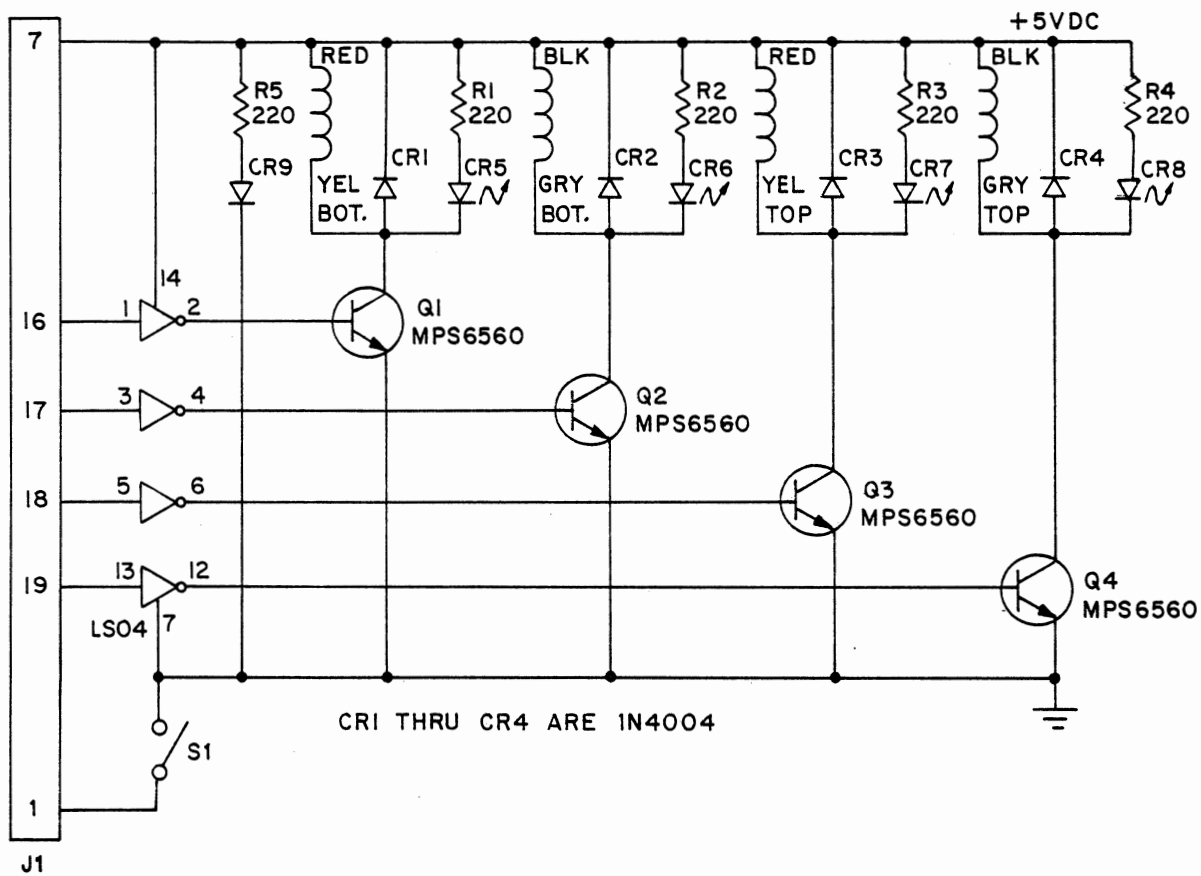
The Micro 68000 has an option that allows it to control stepper motor. This makes learning easier, as visual results of coding can be seen. Either the User can write the control to control the motor directly, or use a program supplied with the Petebug monitor package. All revisions of Petebug greater than 8.5 have this in them. No attempt will be made here to describe the operation of a stepper motor, just the use of the program in Petebug.

Stepper Motor

The 68K stepper consists of a four-coiled stepper motor, which is driven off four lines of the B side of the display PIA. Operation of this motor can be accomplished by using a program located in Petebug. This program works by changing the state of the four lines connected to the four coils in a particular sequence. The operation of this program consists of giving a set of instructions, in a code used by this program, which are interpreted to give the desired stepping function.

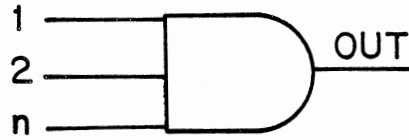
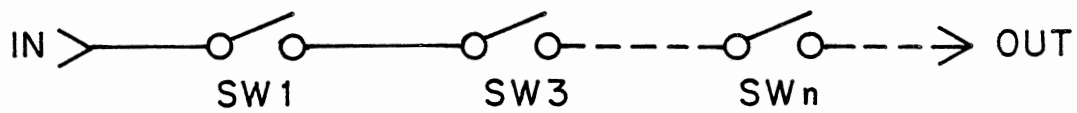
Stepper Motor

To operate the 68K stepper, one must provide a set of commands located at \$000A00. These commands are one word wide and tell the function speed and number of steps for each instruction. Each part of the each word gives a particular portion of that instruction. A word consists of 16 bits, which is 4 nibbles. The high nibble is bits 15-12, and the low is 3-0.

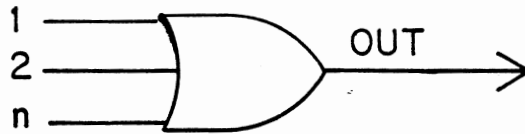
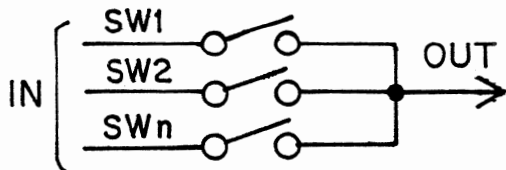


STEPPER MOTOR
SCHEMATIC

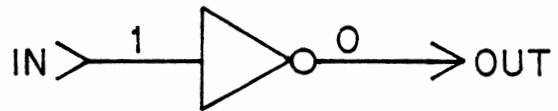
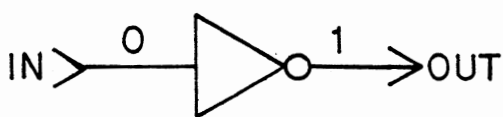
FIGURE 3-10



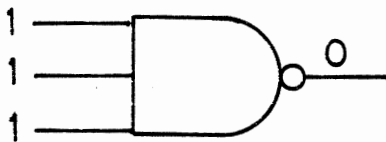
AND GATE



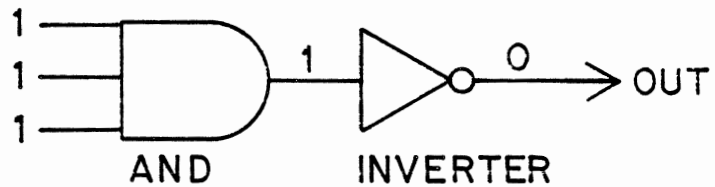
OR GATE



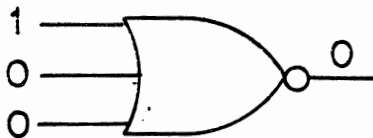
INVERTER



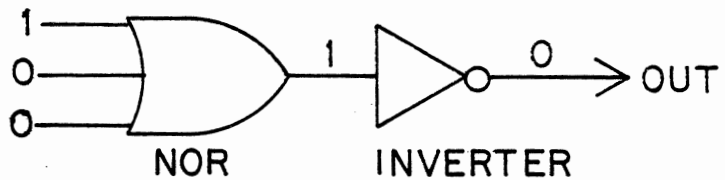
=



NAND GATE



=



NOR GATE

CSA COMPUTER SYSTEM ASSOCIATES

DIGITAL GATES/ LOGIC

FIGURE 3-11

Top Nibble (15-12)

This nibble tells the function, and the available functions are:

- | | |
|---|--|
| F | Clockwise steps |
| B | Counter-Clockwise steps |
| D | Delay-waste time as though stepping |
| E | End-return to Petebug |
| C | Continue program - re-run from the start |

Second Nibble (8-11)

This nibble controls the speed, 0 is the slowest, and \$F is the fastest.

Lower Byte (2 nibbles, 0-7)

This byte controls the number of steps, from 0 to 255.

STEPPER MOTOR BASICS

permanent magnet stepping motors belong to the class of stepping motors frequently identified as "can-stack" stepping motors with step angles typically in the range of 7.5 to 20 degrees. The motors contain two stacked sets of toothed stator poles and circular coils and a permanent magnet rotor with radial alternating north and south poles as shown in figure 1. The number of rotor poles is equal to the number of stator teeth in each set of poles. The stator pole sets are offset by 1/4 of the pole pitch. With both stator coils energized, the rotor will align itself between the two equal stator fields.

A single step of the rotor is the result of a change of magnetic polarity of one set of stator teeth. This change in polarity is brought about by reversing the direction of current flow in the coil associated with those teeth. The rotor motion for a single step with no load applied is that of a damped oscillation as shown in figure 2. The damping characteristics of this curve may be modified by frictional and inertial loading, the sequence in which windings are energized, and the electronic damping in the drive circuitry.

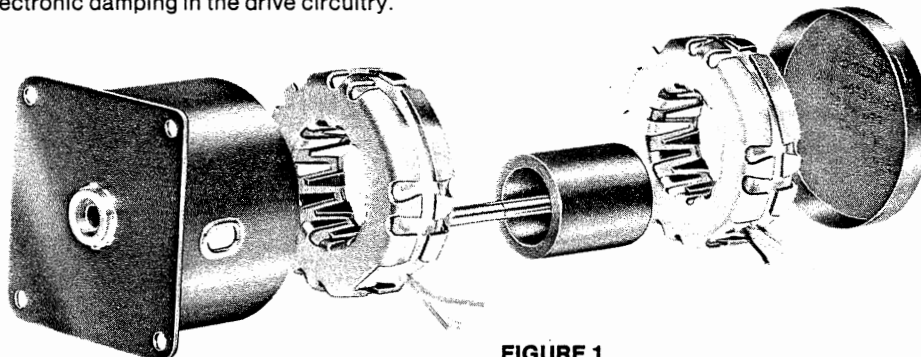
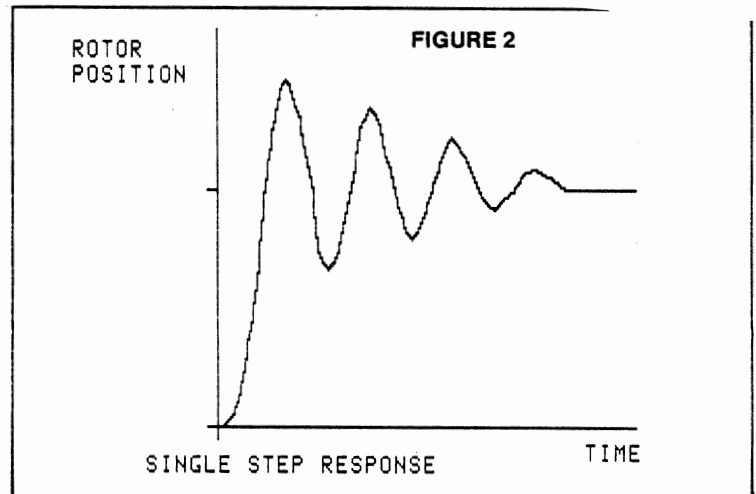


FIGURE 1

STEPPING SEQUENCES

For continuous rotation a repeating sequence of changing tooth polarity is required. Differences in motor performance characteristics result from different types of sequences.

The most commonly used scheme for stepping the motor is to energize both stator coils and to reverse the current in alternate coils with each successive step. This results in a four step sequence as shown in figure 3. Reversing the sequence reverses the direction of rotation. This is called a full step mode with two phases on.

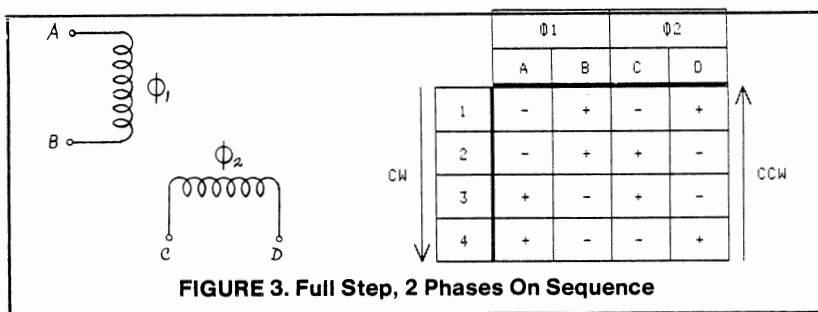


FIGURE 3. Full Step, 2 Phases On Sequence

It is also possible to step the rotor with the same angular increment by energizing only one phase each step as shown in figure 4. This is also a four step sequence and is commonly known as a wave drive. Since only half the copper volume is being used, the efficiency is lower and there is less damping with this sequence than with two phases on.

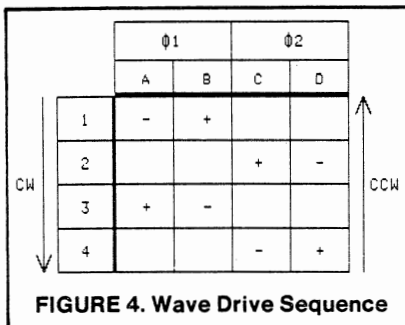


FIGURE 4. Wave Drive Sequence

A third sequence alternates between one and two phases energized to produce 1/2 the step angle of the previous sequences. The half step sequence shown in figure 5 requires eight steps. Although angular resolution may be improved with half-stepping, an important characteristic to note is the lower torque on alternate steps when only one phase is energized. The smaller step angle does provide an improvement in damping, and half-stepping may be advantageous in applications which require operation of the motor at or near resonant frequencies.

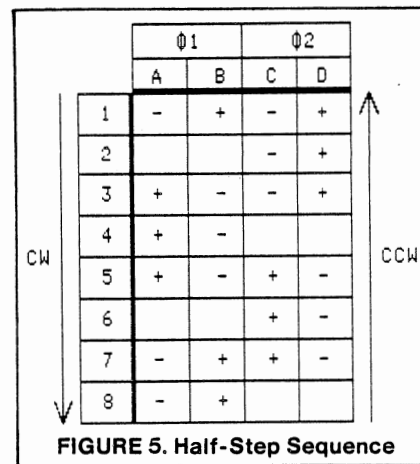


FIGURE 5. Half-Step Sequence

TORQUE CHARACTERISTICS

The maximum torque developed by the motor is the static or holding torque. It is measured while displacing the rotor one step with two phases energized (full step mode). The torque developed during continuous stepping decreases with increasing stepping rate since the current rise time when a phase is energized is limited by the inductance to resistance ratio of that coil.

A typical dynamic torque curve is shown in figure 6. The lower curve represents the maximum torque load which the motor will start and stop without losing steps. The upper curve represents the maximum torque which the motor can develop at a given pulse rate or alternately, the maximum rate to which a given load can be accelerated.

The curve of figure 6 is obtained while operating the motor at a constant voltage over the entire range of pulse rates. Thus the input power to the motor is substantially decreased at higher pulse rates. The torque can be increased at higher pulse rates by increasing the input using a variety of drive techniques. These include simple schemes such as increasing the voltage directly or decreasing the time constant by adding external series resistance, and more elaborate techniques such as bi-level voltage drives or chopper type drives which sense the winding current.

When overdriving techniques are used to increase motor performance, consideration must be given to the maximum permissible temperature rise of the motor windings based on the insulation rating of the motor.

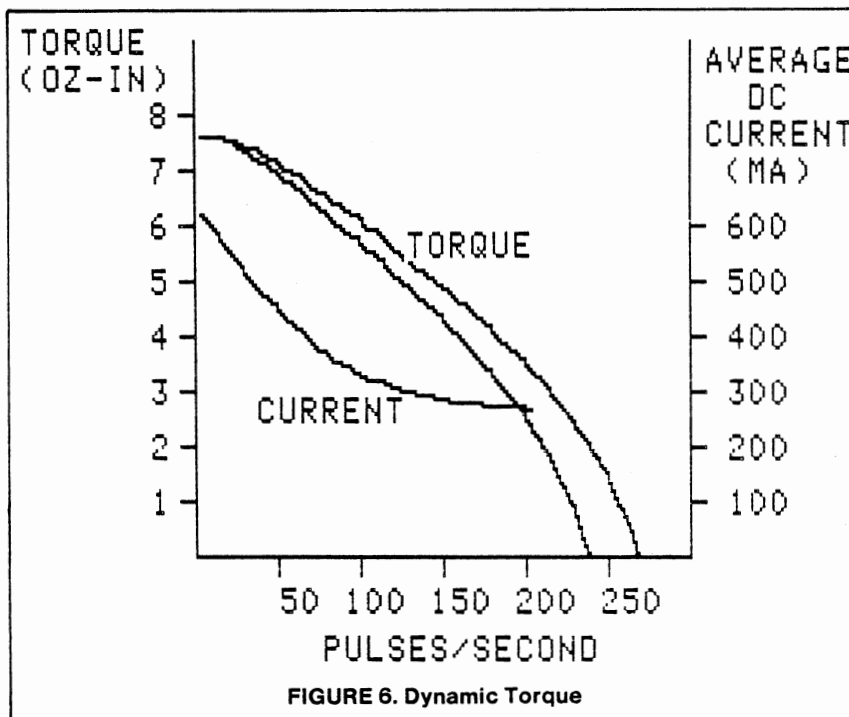


FIGURE 6. Dynamic Torque

RESONANCE

All stepping motors exhibit resonance at certain pulse rates. In typical can-stack type stepper applications the most commonly encountered resonances occur at lower frequencies (less than 100 pulses per second). Although there is no loss of steps at these frequencies, there is an increase in vibration and noise. This becomes even more noticeable when a gear train is coupled to the motor. When operation at resonant frequencies cannot be avoided, some improvement may be made by such methods as increased frictional damping, reduced input power, modified drive circuitry or half-stepping.

STEP ANGLE ACCURACY

The average value of the measured step angles of an unloaded stepping motor over 360 degrees will be equal to the nominal step angle. The maximum deviation of the individual steps from the nominal step angle is the error usually specified as a non-cumulative or incremental step angle error.

BIFILAR AND BIPOLAR OPERATION

The terms bifilar and bipolar refer to two different types of coil windings that may be used in the stator coils. Bipolar windings contain a single coil in each stator half. The switching circuitry used to reverse the direction of current flow with this coil is typically of the full bridge or dual supply type (figure 7). Bifilar windings contain two windings in each stator half. When they are connected as shown in figure 8, the magnetic field may be reversed by switching from one winding to the other.

Note that although a bifilar-wound motor does contain four coils or "phases", it is operated as a two phase motor. The bifilar-wound PM steppers are widely used because of the drive circuit simplicity. However, there are performance differences between the two types of windings. Since the winding volume per phase of a bifilar-wound stepper is only half that of a bipolar-wound stepper, the attainable ampere-turns for a given input power will necessarily be lower for the bifilar-wound motor. As expected the torque is therefore lower. However, it is only lower in a holding mode or at low stepping rates. The reason is that the bipolar coil with its larger volume will also have a larger time constant (L/R) and at higher stepping rates the bipolar-wound motor's torque will decrease to approximately the same level as that of the bifilar-wound motor.

The choice of winding type will depend upon the application. The holding torque for a bipolar version of a given motor will be 20-30% higher than the bifilar version and the dynamic torque will be higher at low stepping rates. Difference in dynamic performance will be small at higher stepping rates. These performance differences must then be weighed against the drive circuit complexity.

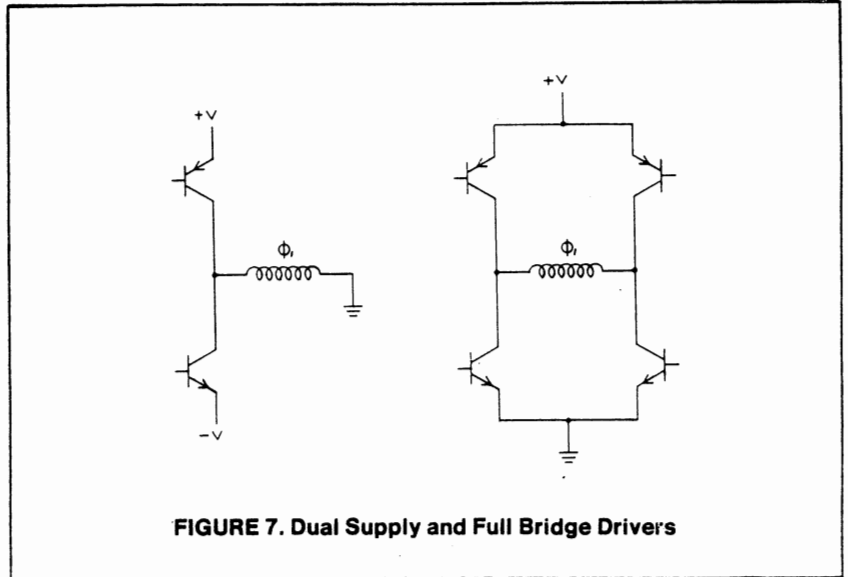


FIGURE 7. Dual Supply and Full Bridge Drivers

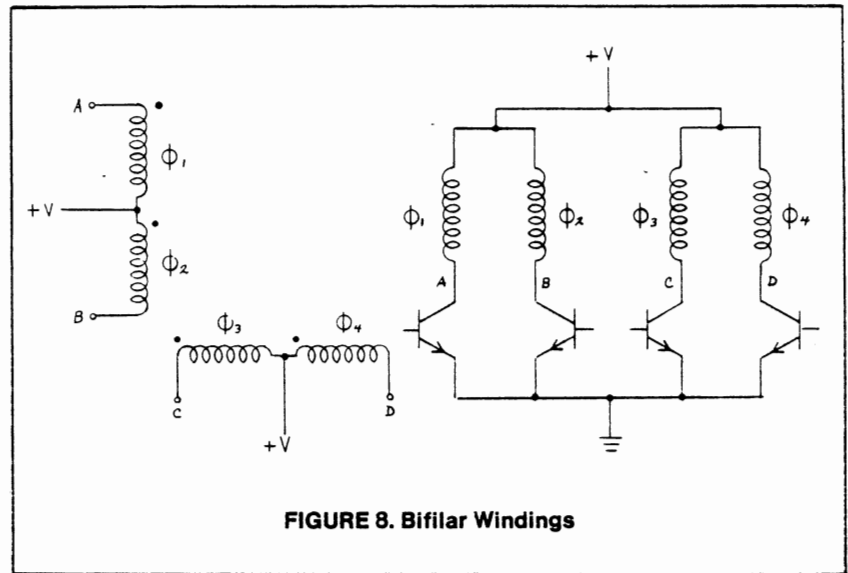
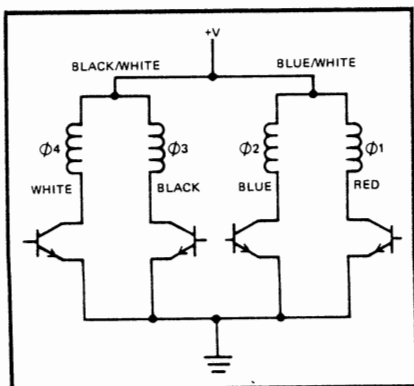


FIGURE 8. Bifilar Windings

"stock motors" are supplied as "four phase" bifilar-wound motors. The standard lead wire configuration is six leads. The color code and switching sequence for the full step, two-phase-on mode is shown below.

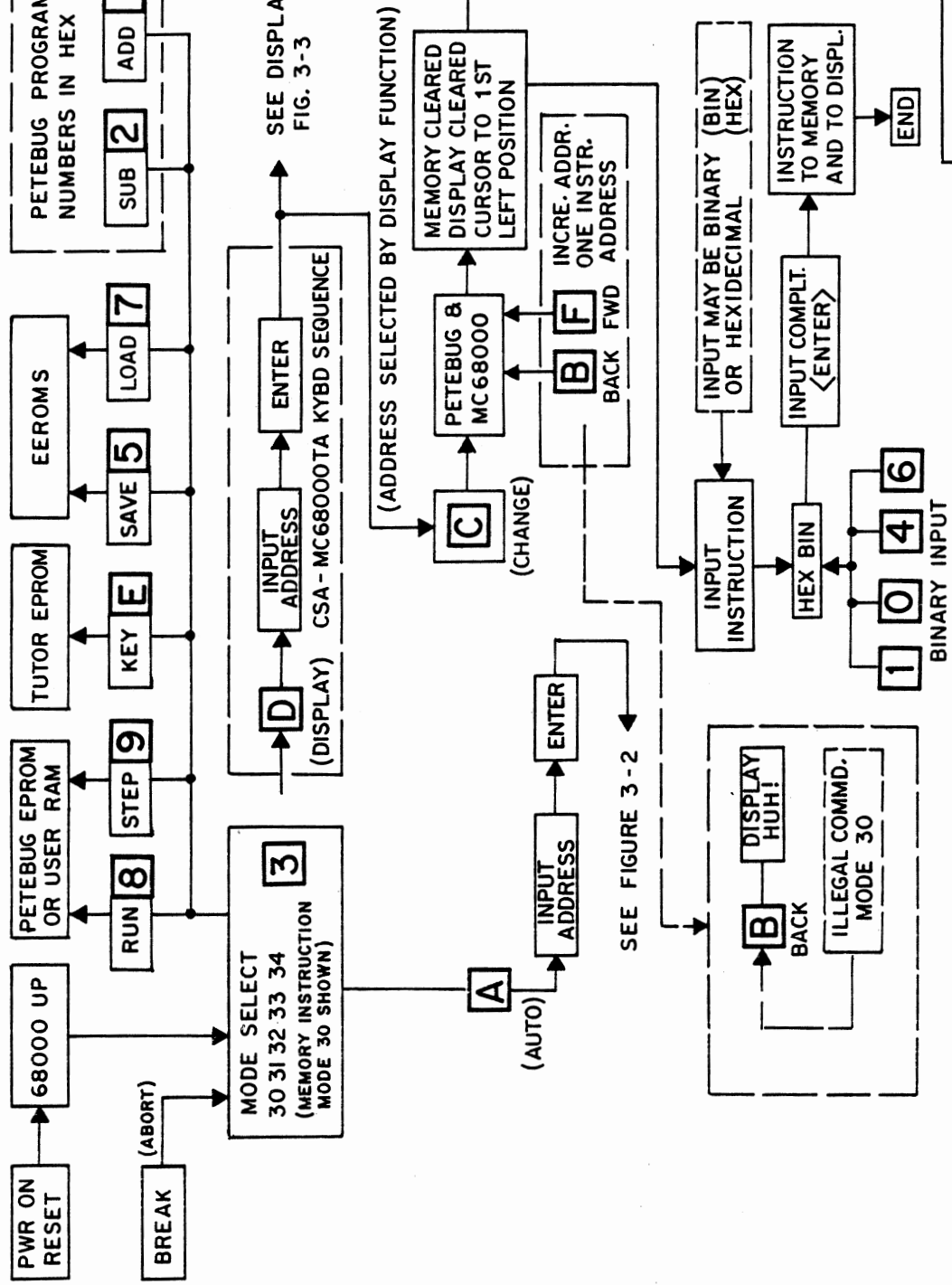


4 PHASE BIFILAR STEPPER WINDINGS STANDARD WINDING COLOR CODE: 6 LEADS

	ϕ_4 WHITE	ϕ_3 BLACK	ϕ_2 BLUE	ϕ_1 RED	
CW ROTATION	1	0	1	0	CW ROTATION
	1	0	0	1	
	0	1	0	1	
	0	1	1	0	

1 = ON, 0 = OFF

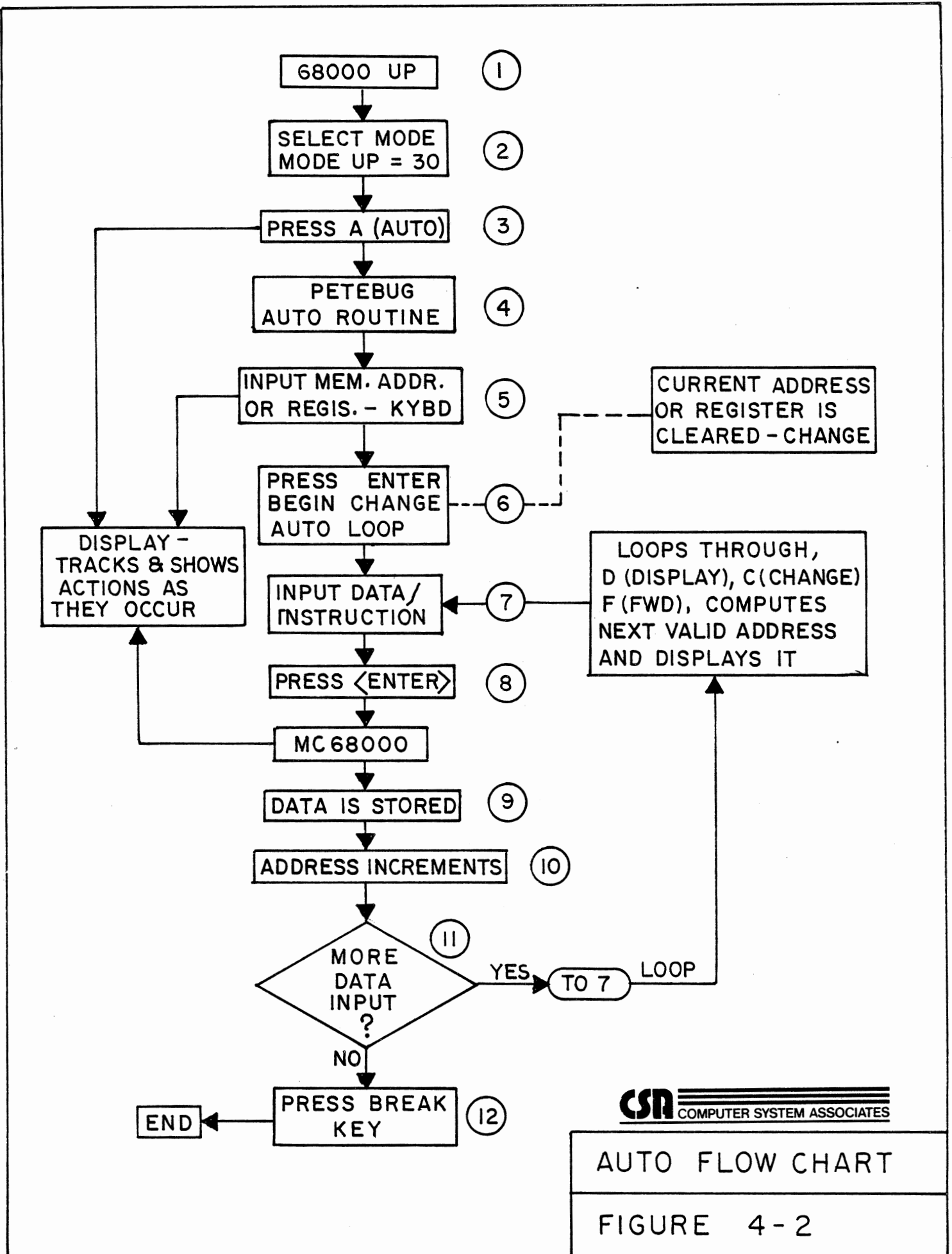
SWITCHING SEQUENCE



PETEBUG

FIGURE





CSA COMPUTER SYSTEM ASSOCIATES

AUTO FLOW CHART

FIGURE 4-2

Table 4-1 Petebug Keyboard Input (Cont'd)

Input	Description
HEX BIN	<p>Mode: Memory See Figure 4-1. Oper: Toggle hex to binary/binary to hex Form: Press key HEX/BIN once each toggle Def : This command toggles the hexadecimal/ binary input mode. It can be issued as an independent command, when the START COMMAND LED is lit, or it can be issued in the middle of a <u>CHANGE</u> or <u>AUTO</u> command while in the instruction mode (30). When in the hexadecimal mode, all input is done in hexadecimal format. When in binary mode, all data (not address) input is in binary format.</p>
0	<p>Mode: Command Oper: Select Master Mind game Form: RESET -0- ENTER Def : See Master Mind, Chapter 2.</p>
0	<p>Mode: Binary Oper: Enter 0 to memory Form: Press key 0 Def : 0 is entered</p>
1	<p>Mode: Binary Oper: Enter 1 to memory Form: Press key 1 Def : 1 is entered</p>
1	<p>Mode: Command Oper: Add Form: 1[data#1] ENTER [data#2] ENTER displays sum in address field. Def : This command performs hexadecimal addition. After the ONE command, enter the first number, then <u>ENTER</u> the second number, then <u>ENTER</u>. The sum of the two numbers will be displayed.</p>

Table 4-1 Petebug Keyboard Input (Cont'd)

Input	Description
2	Mode: Command Oper: Subtract Form: 2[data#1] ENTER [data#2] ENTER displays difference in address field Def : This command allows hexadecimal subtraction. After the TWO command, enter the two numbers, each followed by the <u>ENTER</u> key. The trainer will then display the result of the subtraction of the second number less the first (data #1 minus data #2=displayed result).

NOTE

Mode (Key 3) is always a two key function to change modes. In the select register mode three keys are required for some registers.

3 0	Mode: Command Oper: Select memory instruction mode (opcodes/programs enter to memory) Form: (from other modes): BREAK 3 0 ENTER Def : Memory is displayed as machine instructions on both the seven segment displays and the binary LEDs.
3 1	Mode: Command Oper: Select Memory (data 8 bit) Form: (from other modes): BREAK 3 1 ENTER Def : Memory is displayed as 8 bit bytes on the last two digits of the first data row of the seven segment displays only.

Table 4-1 Petebug Keyboard Input (Cont'd)

Input	Description
3 2	Mode: Command Oper: Select memory (data, 16 bit) Form: (from other modes): BREAK 3 2 ENTER Def : Memory is displayed as 16 bit words on the first and second lines of the seven segments only.
3 3	Mode: Command Oper: Select memory (data, 32 bit) Form: (from other modes): BREAK 3 3 ENTER Def : Memory is displayed as 32 bits long words on the first and second lines of the seven segments only.
3 4	Mode: Command Oper: Select register Form: (from other modes): BREAK 3 4 ENTER Def : Registers are displayed as 32 bits (16 for the status register) on the first and second lines of the seven segment area only.
	*SELECT REGISTER SUB MODES
3 4 D 0 ENTER	Select user stack pointer (USP)
3 4 D 1 ENTER	Select supervisor stack pointer (SSP)
3 4 D 2 ENTER	Select program counter (PC)
3 4 D 3 ENTER	Select status register (SR)
4 (CURSOR LEFT)	Mode: Binary Oper: Move cursor left one bit Form: Press 4 key Def : This command is only valid when changing data in the binary mode. It moves the cursor one position to the left.

Table 4-1 Petebug Keyboard Input (Cont'd)

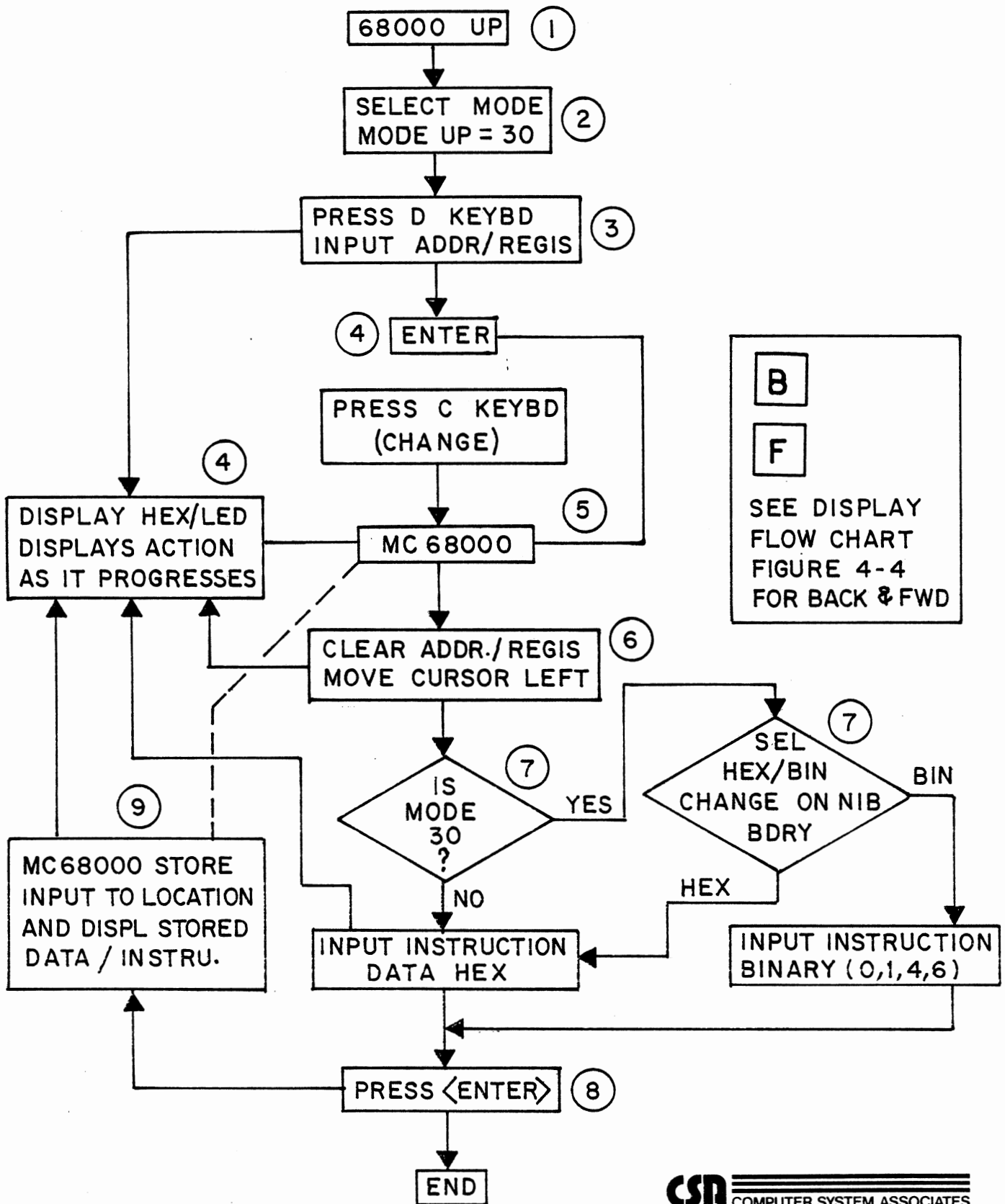
Input	Description
5 (SAVE)	<p>Mode: Memory instructions (30) save to EEROMS Oper: See EEROMS, Chapter 3 Form: 5 ENTER</p> <ol style="list-style-type: none"> 1. Start address in RAM (Save=source area/load=destination area) 2. Number of <u>bytes</u> (8 bits) to Save or Load. 3. Offset address (must be an even number). <p>Def : This command is used to write an area of RAM into the EEROMS in order to save it. Once this key has been pressed, Petebug will begin a sequence to save an area to EEROMS. See Chapter 3, EEROMS for details.</p>
6 (CURSOR RIGHT)	<p>Mode: Binary Oper: Move cursor right one bit Form: Press 6 key Def : This command, only valid when changing data in binary mode. It moves the cursor one position to the right.</p>
7 (LOAD)	<p>Mode: Memory instructions (30) load from EEROMS Oper: See EEROMS, Chapter 3 Form: 7 ENTER</p> <ol style="list-style-type: none"> 1. Start address in RAM (Save=source area/load=destination area) 2. Number of <u>bytes</u> (8 bits) to Save or Load. 3. Offset address (must be an even number). <p>Def : This command is used to load an area of RAM from the EEROMS. Once this key has been pressed, Petebug will begin a sequence to load from the EEROMS. See Chapter 3, EEROMS for details.</p>

Table 4-1 Petebug Keyboard Input (Cont'd)

Input	Description
8 (RUN)	<p>Mode: Command Oper: RUN user program Form: (from other modes): BREAK 8 ENTER Def : This command starts the execution of any program. The command key (8) is followed by the starting address for the execution and the <u>ENTER</u> key. If no address is given, then the current program counter location will be used. The trainer keeps track of the registers used by the user's program. When the trainer is first turned on or RESET, all the data and address registers are cleared, and the other registers have these values.</p>
9 (STEP)	<p>Mode: Command Oper: Single STEP an instruction. Form: (from other modes): BREAK 9 ENTER Def : This command traces one instruction from the current program counter location. After the instruction is executed, the new program counter is displayed on the trainer display. Registers and memory may be displayed with the D command, and another STEP command may be performed. The registers are saved and loaded as described in the run command.</p>
A (AUTO)	<p>Mode: Mode Dependent (Mode 30-34) See Figure 4-2. Oper: On ENTER advances to next address Form: (from selected mode): A [address]. . . ENTER [data] ENTER [data] ENTER . . . Data is cleared from current register or memory address and input data is stored. See CHANGE command. BREAK (must be terminated with BREAK).</p>

Table 4-1 Petebug Keyboard Input (Cont'd)

Input	Description
A (AUTO)	<p>Def : This command is used to enter large amounts of data to memory or registers, without having to hit the <u>FORWARD</u> and <u>CHANGE</u> keys for each additional location. After pressing the <u>AUTO</u> key, enter the starting address or register and press <u>ENTER</u>. The trainer will display the location being changed, now you enter the data for that location. Then press <u>ENTER</u> to store the data. <u>AUTO</u> will automatically advance to the next location. The only way to end the <u>AUTO</u> command is to use the <u>BREAK</u> key.</p>
(BACK)	<p>Mode: Mode dependent (Mode 31-34) (*Not valid mode 30/error message: huh?)</p> <p>Oper: Displays previous register or fixed length data word</p> <p>Form: Press B</p> <p>Def : This command displays the previous memory or register that is consistent with the display mode. The registers are in reverse order of <u>FORWARD</u>.</p>
C (CHANGE)	<p>Mode: Mode dependent See Figure 4-3.</p> <p>Oper: Change the contents of register or memory cell</p> <p>Form: D-Address-[ENTER]-C . . . (clears current contents, input new data from keyboard) . . . [ENTER]</p> <p>Def : This command is used to change the contents of what is currently being displayed. In any mode except mode 30, data is exclusively entered in hexadecimal, ignoring the HEX/BIN flag. Binary mode can only be used in mode 30 (memory as instructions), when the HEX/BIN flag is on. When entering data in hexadecimal mode, enter the new data, and press <u>ENTER</u>. When changing in binary, the keys 0 and 1 change the data under the blinking</p>

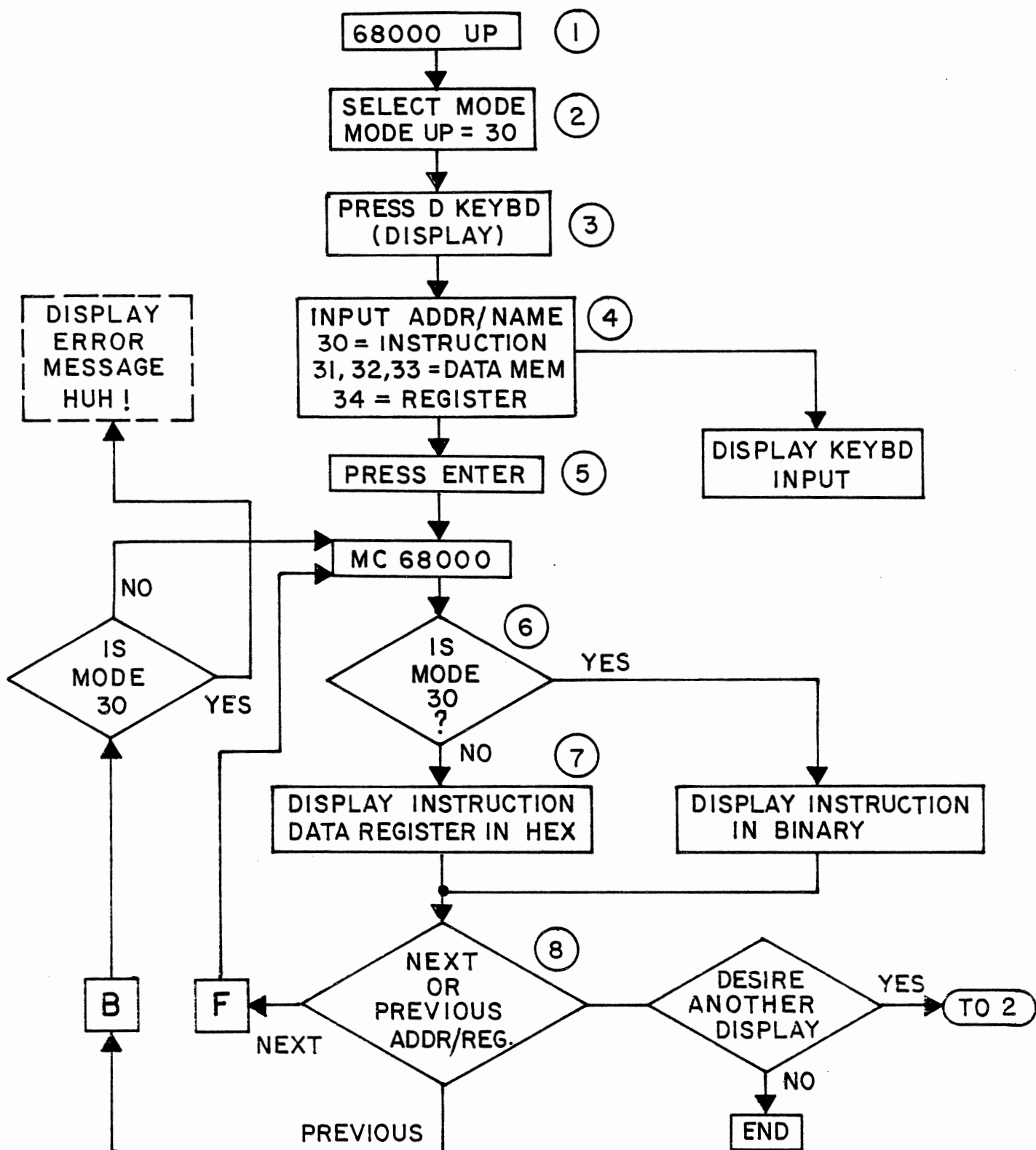


B
F
 SEE DISPLAY
 FLOW CHART
 FIGURE 4-4
 FOR BACK & FWD

CSA 
 COMPUTER SYSTEM ASSOCIATES

CHANGE FLOW CHART

FIGURE 4-3



CSA COMPUTER SYSTEM ASSOCIATES

DISPLAY FLOW CHART

FIGURE 4-4

Table 4-1 Petebug Keyboard Input (Cont'd)

Input	Description
C (CHANGE)	<p>cursor, and the left and right arrows (4 and 6 respectively) move the blinking cursor. When all the binary changes have been made, press the <u>ENTER</u> key.</p> <p>Note: If binary mode is legal, you may switch from binary to hexadecimal at any 4 bit (nibble) boundary.</p> <p>All registers are 32 bits, except SR, which is 16. Early revision Petebugs treat input of the SR as 32 bits, where the upper 16 bits are put in the register, thus the desired value is typed in followed by four extra digits.</p>
D (DISPLAY)	<p>Mode: Mode dependent (30-34) See Figure 4-4.</p> <p>Oper: Display current instruction, memory contents, data or register contents depending on mode.</p> <p>Form: Select mode, press D , address, then [ENTER]</p> <p>Def : This command is used to display the contents of memory or registers, and it is also used to setup the address for the <u>CHANGE</u>, <u>FORWARD</u> and <u>BACK</u> commands. The <u>MODE</u> command (see 30-34) is used to specify the format and content of the display. If memory is being displayed, the <u>D</u> (display) key is followed by a valid 68000 memory address and the <u>ENTER</u> key. The contents of the specified memory location(s) will be displayed in the current display format. Note that 68000 machine instructions are variable length, but the proper number of words will be displayed by Petebug.</p>

Table 4-1 Petebug Keyboard Input (Cont'd)

Input	Description
D (DISPLAY)	An attempt to display non-existent memory will cause the trainer to hang up, waiting for a reply that will never come. All revision I trainers will do this, but with revision II trainers, a jumper can be installed to cause a bus error if this happens. If this happens, you must reset the trainer to bring it back. If registers are being displayed, the <u>D</u> key is followed by the register name from the table below.
E (TUTOR)	<p>Mode: Command</p> <p>Oper: Causes "Execute Tutor"</p> <p>Form: Press E</p> <p>Def : If Petebug is running, pressing this key will cause the RS-232 based monitor, (TUTOR) to execute. (See Appendix A.)</p>
F (FWD)	<p>Mode: Mode dependent (30-34) See Figure 4-1.</p> <p>Oper: Advance display forward to next higher address, register, instruction, dependent on mode.</p> <p>Form: Select mode, display address, press F to advance</p> <p>Def : This command displays the next register or memory location, depending on the display mode. For register displays, the order is as follows: D0 to D0, A0 to A6, user stack pointer, supervisor stack pointer, program counter and then status register. The register set wraps around; ie. after the status register, D0 will be</p>

Table 4-1 Petebug Keyboard Input (Cont'd)

Input	Description
F (FWD)	displayed. When used for memory, the next location displayed depends on the current display mode. For bytes, words and long words, the address is incremented the appropriate number of bytes (1, 2 and 4 respectively). For instructions, the display is incremented by the length of the previous instruction, so as to display the beginning of the next instruction.

NOTE

The following commands are Tutor commands (see Appendix A) and are presented here for quick reference.

GP (GO PETEBUG)	This command is similar in function to the Petebug E command. When GP followed by ENTER is input from Tutor, the Trainer will initialize the Petebug monitor program (68000 UP).
LE (LOAD EEROM)	While operating Tutor, this (LE) command will cause a program stored in EEROM to be LOADED to RAM under Tutor control (similar to Petebug 7 (LOAD) command).
PE (SAVE EEROM)	While operating Tutor, this (PE) command will cause a program stored in RAM to be loaded into EEROM (saved) under Tutor control (similar to Petebug 5 (SAVE) command).

Table 4-1 Petebug Keyboard Input (Cont'd)

Input	Description
G (GO)	While in Tutor, the GO (G) command, followed by the desired address and ENTER will cause Trainer operation to jump to the program starting at the address specified (similar to Petebug's RUN (8) command).
GD (GO DIRECT)	This command will bypass all intermediate Tutor program functions and jump to the address specified.

Resetting Petebug and the Processor

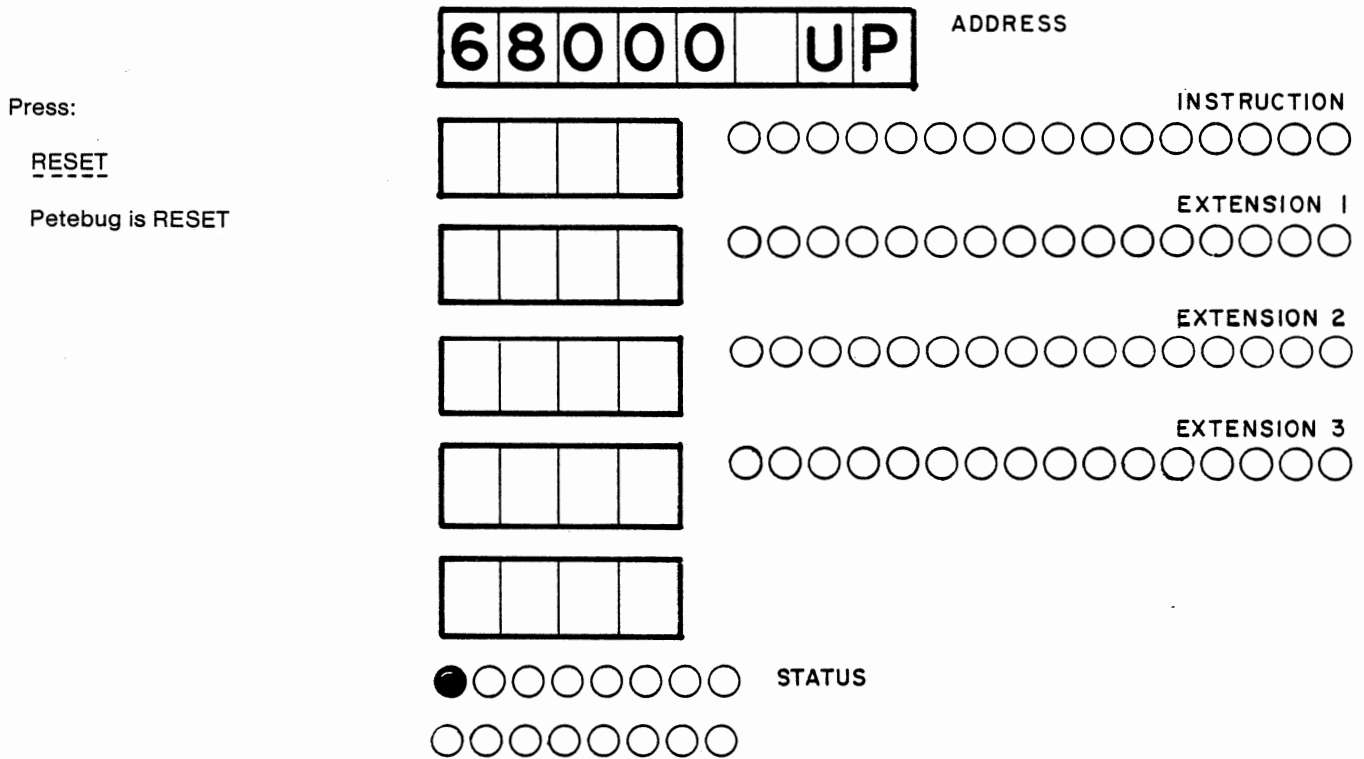
SUMMARY

RESET - while in the Petebug Monitor Program, causes a soft reset that clears the registers of the MPU, but does not disturb or clear memory.

The **RESET** key and function will become one of the most often used commands while operating with the Petebug Monitor Program. The command will be used to ensure that Petebug and MPU are initialized and prepared to begin new operations. However, due to the training nature of the CSA Trainer, the memory (and any programming that may have been saved in memory) remain intact after a **RESET** has been performed.

There is a jumper connection on the MPU board, lower right corner that may be wired to perform the exact same function as the **RESET** key on the keyboard, (see Chapter 3).

EXAMPLE



Example: Reset of Petebug

Arithmetic Commands (Add and Subtract)

SUMMARY

- 1 <data1> **ENTER** <data2> **ENTER** displays <data1> +<data2> in the address field of the display.
- 2 <data1> **ENTER** <data2> **ENTER** displays <data1> -<data2> in the address field of the display.

These instructions form the minimal calculator for use with relative displacements of instructions. The form of the two commands are identical; press the command key, and then the two data in hexadecimal, each followed by the **ENTER** key. The result is calculated with a numeric wraparound (modulus) at 2 to 32nd power.

For example, to calculate an instruction offset: the 68000 uses the address at the end of the instruction, plus the offset contained in the instruction. If the instruction is at \$A3E, and is a one word (2 byte) instruction jumping to \$A10, then the result should be \$D0. This can be done using the calculator's subtract command; subtract \$A3E from \$A10. From this result (\$FFFFFFD2) subtract 2, giving \$FFFFFFD0, from which only the last 8 bits are significant.

- a) 2 <A10> **ENTER** <A3E> **ENTER** displays <FFFFFFD2>
- b) 2 <FFFFFFD2> **ENTER** <2> **ENTER** displays <FFFFFFD0>

EXAMPLE

To add \$FF74 to
\$C022:

1 F F 7 4 ENTER
C 0 2 2 ENTER

0001bF96

ADDRESS

--	--	--	--

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

INSTRUCTION

--	--	--	--

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 1

--	--	--	--

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 2

--	--	--	--

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 3

--	--	--	--

●○○○○○○○○○○
○○○○○○○○○○

STATUS

ADDRESS

0000bC50

To subtract \$3276
from \$EEC6:

2 E E C 6 ENTER
3 2 7 6 ENTER

--	--	--	--

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

INSTRUCTION

--	--	--	--

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 1

--	--	--	--

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 2

--	--	--	--

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 3

--	--	--	--

●○○○○○○○○○○
○○○○○○○○○○

STATUS

Example: Addition and subtraction

Setting the Display/Change Mode (Key 3)

SUMMARY

To change the mode between instruction, 8 bit data, 16 bit data and 32 bit data, and register.

- 3 0** mode is changed to: memory as 68000 machine code.
- 3 1** mode is changed to: memory as 8 bit data
- 3 2** mode is changed to: memory as 16 bit data
- 3 3** mode is changed to: memory as 32 bit data
- 3 4** mode is changed to: 68000 registers

Most of Petebug's instructions operate on a certain size of data, and Petebug must know the size it is using for it to perform correctly. For example, trying to put a 32 bit value into memory while Petebug is in 8 bit mode would take 4 separate operations, whereas it would only take one operation if Petebug were in the 32 bit mode.

The mode instruction is listed first because it applies to almost all of Petebug's commands. The commands affected are: *Change*, *Display*, *Forward*, *Backward* and *Auto*. In the *Change* and *Auto* commands it prevents entering the wrong length of data, or using binary mode when inappropriate. In the *Forward*, *Backward* and *Display* commands, it causes the data to be displayed in the correct format. In the *Forward*, *Backward* and *Auto* commands, it causes the increment to the next piece of data to be the correct number of bytes.

The command is always a two key command, the first is **3** , and the following key sets the actual mode. Valid second keys and their corresponding modes are shown in the summary box. The current mode is displayed in three LEDs in the status line. The following table shows the correspondence between the LEDs in the status display and the mode, where LED 7 is the leftmost LED in the group.

7	6	5	4	3	2	1	0	Mode
x	x	x	0	0	0	x	x	Memory (Instruction)
x	x	x	0	0	1	x	x	Memory (8 bit)
x	x	x	0	1	0	x	x	Memory (16 bit)
x	x	x	0	1	1	x	x	Memory (32 bit)
x	x	x	1	0	0	x	x	Registers

EXAMPLE

Press:

3 3

To set to 32
bit mode.

68000 UP

ADDRESS

INSTRUCTION

[][][][]

○ ○

EXTENSION 1

[][][][]

○ ○

EXTENSION 2

[][][][]

○ ○

EXTENSION 3

[][][][]

○ ○

[][][][]

Press:

3 4

To set to
register mode.

● ○ ○ ○ ● ● ○ ○

STATUS

○ ○ ○ ○ ○ ○ ○ ○ ○ ○

68000 UP

ADDRESS

INSTRUCTION

[][][][]

○ ○

EXTENSION 1

[][][][]

○ ○

EXTENSION 2

[][][][]

○ ○

EXTENSION 3

[][][][]

○ ○

[][][][]

● ○ ○ ○ ● ○ ○ ○ ○ ○

STATUS

○ ○ ○ ○ ○ ○ ○ ○ ○ ○

Example: Change Mode command

Binary input in Instruction mode

SUMMARY

HEX/BIN causes Petebug to switch from Hexadecimal to Binary, or from Binary to Hexadecimal. Can be used at command level, or while data entry cursor is at a nibble boundary when changing or entering an instruction.

4 causes binary cursor bit to move left one bit.

6 causes binary cursor bit to move right one bit.

0 and **1** enter data in binary mode.

When using the *change* or *auto* commands in the instruction mode, there are two forms of input available: hexadecimal or binary. To switch from one mode to the other, use the **HEX/BIN** key. This can be done when at the command level, or when changing or entering a number. There is a restriction; if switching out of binary mode, it must be done on a nibble boundary. A nibble is four bits, so the boundaries fall at the end of every hexadecimal digit. The current mode is displayed in one of the status LEDs.

While entering data in the binary mode, there are also several extra commands. There are two commands for moving the flashing binary cursor; one to move it left one bit, and one to move it right one bit.

Once in the correct position, individual bits may be entered using the **0** and **1** keys. These enter the appropriate bit, and advance the cursor one bit to the right.

EXAMPLE

Press:

RESET

to bring to HEX

HEX/BIN

to bring to BIN

68000 UP

ADDRESS

Four empty boxes for data entry.

Four empty boxes for data entry.

Four empty boxes for data entry.

Four empty boxes for data entry.

Four empty boxes for data entry.

STATUS indicator: 10 circles, with the 1st and 2nd filled.

INSTRUCTION: 16 empty circles.

EXTENSION 1: 16 empty circles.

EXTENSION 2: 16 empty circles.

EXTENSION 3: 16 empty circles.

HEX/BIN

to return to HEX

68000 UP

ADDRESS

Four empty boxes for data entry.

Four empty boxes for data entry.

Four empty boxes for data entry.

Four empty boxes for data entry.

Four empty boxes for data entry.

STATUS indicator: 10 circles, with the 1st filled.

INSTRUCTION: 16 empty circles.

EXTENSION 1: 16 empty circles.

EXTENSION 2: 16 empty circles.

EXTENSION 3: 16 empty circles.

Example: Use of Binary mode (see *change instruction* command)

Running User Programs

SUMMARY

[8] < address> **ENTER** starts running a user program with the current register set at the address given.

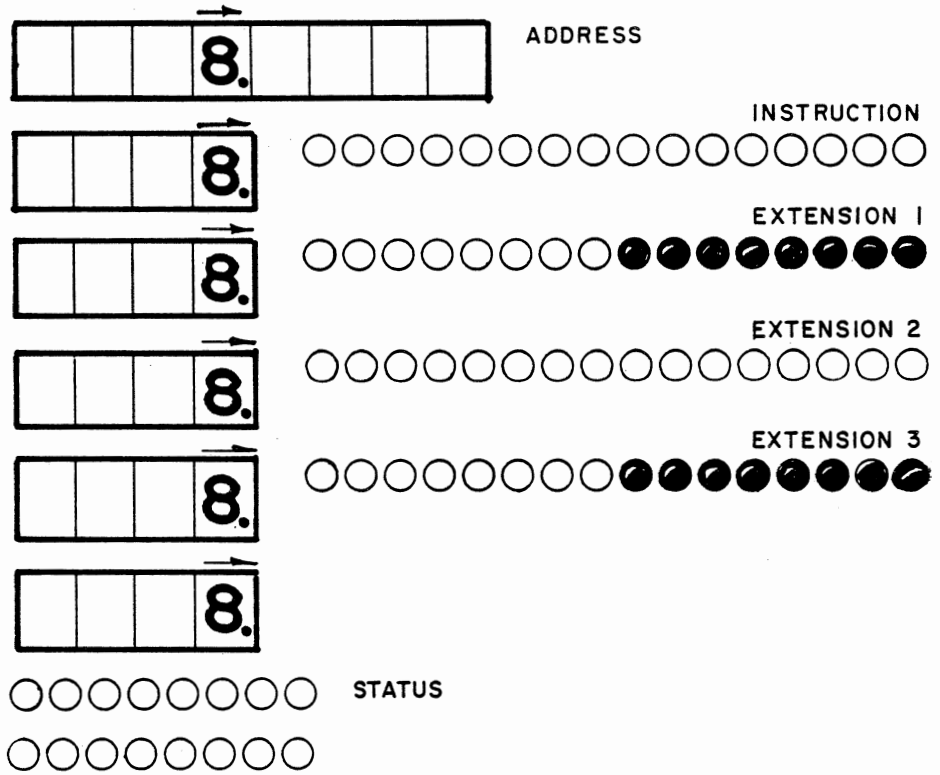
This command is used to run user programs. Before jumping to the given address, the registers are loaded with the values that can be seen with the display register command.

Before jumping, the display is cleared and a user program LED in the status group is lit.

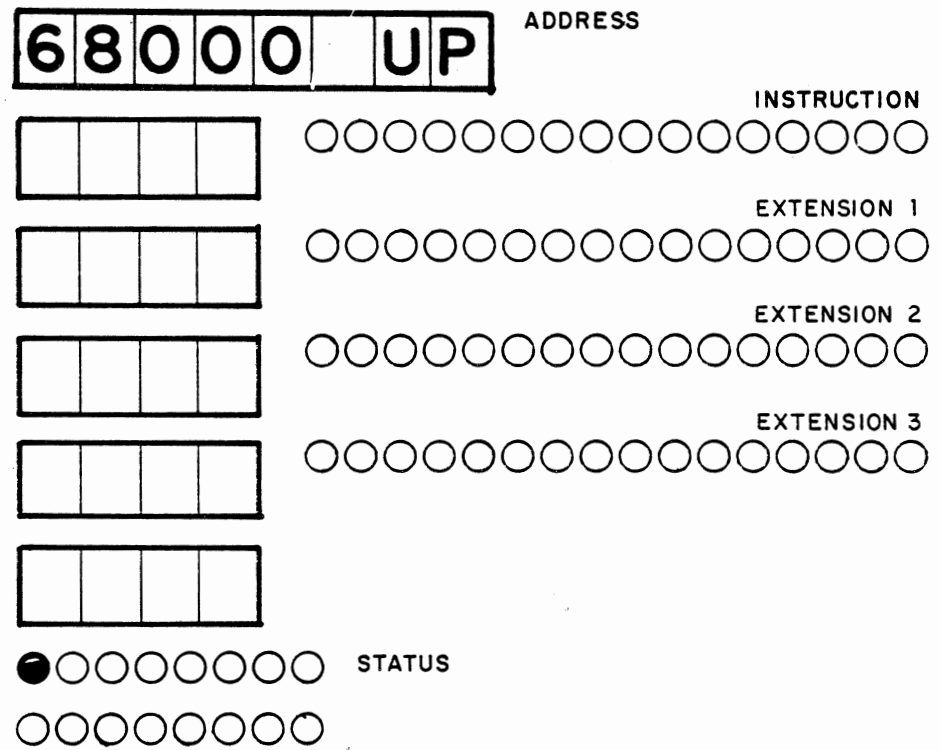
EXAMPLE

To avoid entering a program, run the demo program (at address FF8038).

8 FF8038



Depress RESET to stop the demo program.



Example: Running a user program

Single Stepping a User Program

SUMMARY

9 executes a single instruction from the current user program counter. All registers may be examined following this command.

Using the *step* command causes a single instruction to be executed, and control then returns to Petebug. The instruction is at the current value of PC, and the registers are loaded with their proper values before executing the instruction.

Following the instruction, the registers are saved, and can be examined with the *display register* command. The PC value after the execution is displayed immediately to show where the program is operating.

Any number of *step* commands may be done in sequence, or with other intervening commands.

EXAMPLE

Using the demo #1 program as an example for single stepping.

Set register mode.

3 4

Display the PC.

D 2

Change to \$FF8038.

C F F 8 0 3 8 ENTER

Now single step.

9 9 9 ... 9 (SINGLE STEP)

P	C						
---	---	--	--	--	--	--	--

ADDRESS

0	0	F	F
---	---	---	---

○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

INSTRUCTION

8	0	3	8
---	---	---	---

○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

EXTENSION 1

--	--	--	--

○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

EXTENSION 2

--	--	--	--

○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

EXTENSION 3

--	--	--	--

●	○	○	○	○	○	○	○
---	---	---	---	---	---	---	---

STATUS

○	○	○	○	○	○	○	○
---	---	---	---	---	---	---	---

Example: Single stepping a user program

Entering Streams of Data - Auto

SUMMARY

A < address> [**ENTER** < data> **ENTER**] ... **BREAK**
allows the entry of any number of 8, 16 or 32 bit words, or instructions or registers into sequential locations. (Must be terminated with **BREAK** or **RESET**).

The *auto* command is in fact a conglomeration of the *display*, *change* and *forward* commands. It works in any mode, with each individual operation taking on slightly different meanings in different modes.

Once the address is keyed in, followed by the **ENTER** key, the address is displayed, along with the data that is currently in the word; a cursor is in the leftmost position. Data is stored following subsequent presses of the **ENTER** key. The entire operation is terminated with the **BREAK** key. Thus if the data being entered for the current word is no longer desired, then it can be aborted before it is stored.

When used in the instruction mode, the registers come in the order shown in the table from the display register instruction with wraparound.

When in one of the memory data modes, the correct length is used for the forward operation. In addition, only the correct number of bytes may be entered.

When in the instruction mode, the binary input mode is also usable, and can be switched on and off at nibble boundaries as in the change instruction. In addition, the number of words that is stored is the number that is keyed, while the number of words to the next instruction is calculated by disassembling the instruction, so it is possible to key more or less than the number of words moved forward.

This instruction can make the keying of a large quantity of data a faster task than using the *forward* and *change* instructions.

Moving Forward or Backward in Memory or Registers

SUMMARY

- F** displays next higher register, instruction or fixed length data word.
- B** displays previous register or fixed length data word. (not allowed in instruction mode)

The *forward* and *backward* commands operate on the *current address* and *current register* variables, as set by the *display* and *auto* commands. It also depends on the current mode (see the *change mode* command to set the mode). These commands are single key commands, and require no additional parameters.

For memory data modes, the *forward* command adds the length of the current data size to the *current address*, and then displays the contents of that location. For example if the mode is set to 32 bit data, and the current address is hexadecimal 8046, then the current address would be set to 804A, and the contents of 804A would be displayed as a 32 bit value. The value, as in the *display* command, is displayed in the hexadecimal displays only. For memory data modes, the *backward* command operates in an analogous way, the only difference is that the length is subtracted from the current address instead of being added to it. Overflows, both negative and positive, are ignored so that the address arithmetic is done modulo 2 to the 32nd power.

For the instruction mode, the length that is determined by Petebug (by disassembling the instruction) is added to the current address, and then the contents of the new address are displayed as an instruction in both the hexadecimal and binary displays. The contents of the new current address are disassembled before display so that the correct number of word will be displayed, as in the *display* command. If the address overflows, the carry is ignored, and the lower 32 bits are taken as the new current address. Since it is very difficult to correctly disassemble instructions going backwards, this command is not allowed, and will result in the error message "HUH?" being displayed by Petebug.

For the register mode, the new current register is taken as the previous or next register in the table presented in the *display register* command. The list is treated in a circular fashion; USP is one forward of A6, and A6 is one backward of USP.

EXAMPLE

First set to instruction mode.

3 0

Now set the current address.

D F F 8 6 A Z ENTER

Now move to next forward instruction.

F

First set to 16 bit data mode.

3 2

Now set current address by displaying.

D F F 8 0 4 8 ENTER

Now move backward one word.

B

00FF86A2

ADDRESS

42AC

○●○○○○○●○○○○○●○○●○○○

INSTRUCTION

0004

○○○○○○○○○○○○○○○○○○○●○○○

EXTENSION 1

□□□□

○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 2

□□□□

○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 3

□□□□

●○○○○○○○○○

STATUS

○○○○○○○○○○○

00FF8048

ADDRESS

0802

○○○○○○○○○○○○○○○○○○○○○○○○

INSTRUCTION

□□□□

○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 1

□□□□

○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 2

□□□□

○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 3

□□□□

●○○○○●○○○

STATUS

○○○○○○○○○○○

Example: Forward and Backward commands

Changing Memory in Instruction Mode

SUMMARY

C <data> **ENTER** causes the data, entered in hex or binary, as <data> to be stored at the current address. The current address is read back after storing the data, and the result is displayed. (must be in the memory instruction mode)

The *change* command takes the entered value, and stores it at the current address. There is only one change command with different modes, so the correct mode must be selected before starting the *change* command. To change to mode, see the *change mode* command. This description applies to changing memory, represented as machine instructions.

After entering the **C** of the change command, the data area is cleared, and a cursor is placed in the first byte of the data area, or the first bit of the binary display, depending on the input mode. The data may be entered in either in hexadecimal or binary, depending on the input mode, but in either case the data must be terminated with the **ENTER** key.

Any digits or bits that are entered remain in the position they area entered, and are not moved (unlike the *change* command in data modes).

Once the **ENTER** key is pressed, the data is stored at the current address. Following the store operation, the contents of the current address are read, and then displayed in the data field of the displays. When the data is re-displayed, Petebug does not disassemble the instruction, but displays the number of digits entered.

When writing into RAM, the data read back should be the same as the data stored. If attempting to store into ROM, the value of the ROM locations will not change, so the displayed value will be the same both before and after the store.

EXAMPLE

First set to instruction mode.

3 0

Now set current address and display location \$1000.

D 1 0 0 0 ENTER

Now change the first two nibble in hexadecimal to \$4E. Then change the third and fourth nibbles to \$75.

C 4 E HEX/BIN
0 1 1 1 0 1 0 1
ENTER

And return to hexadecimal (optional)

HEX/BIN

Now change this instruction to \$4E59, using binary mode.

C HEX/BIN 6 6 6
6 6 6 6 6 6 0
1 1 0 6 6
HEX/BIN ENTER

Note: 6 is the Cursor Right Command for binary input

00001000 ADDRESS

4E75

INSTRUCTION
●○○○●●○○●●○○●●○○●●○○●●

EXTENSION 1
○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 1
○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 2
○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 2
○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 3
○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 3
○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

STATUS
●○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

Note: The starting display may differ; change will produce the same result.

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

00001000 ADDRESS

4E59

INSTRUCTION
●○○○●●○○●●○○●●○○●●○○●●○○●●

EXTENSION 1
○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 1
○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 2
○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 2
○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 3
○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 3
○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

STATUS
●○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

Example: Change of Memory in Instruction mode

Changing Memory in Memory Data Modes

SUMMARY

C <data> **ENTER** causes the data, entered in hex, as <data> to be stored at the current address. The current address is read back after storing the data, and the result is displayed. (must be in one of the memory data modes)

The *change* command takes the entered value, and stores it at the current address. There is only one change command with different modes, so the correct mode must be selected before starting the *change* command. To change to mode, see the *change mode* command. This description applies in the modes treating memory as 8, 16 or 32 bit quantities.

After entering the **C** of the change command, the data area is cleared, and a cursor is placed in the first byte of the data area. The data must be entered in hexadecimal, followed by the **ENTER** key. If fewer than the maximum number of digits are keyed, it is assumed that the digits keyed go in the least significant digits, while the remaining digits (the most significant) are filled with zeroes. (This is opposite to the instruction mode.)

Once the **ENTER** key is pressed, the data is stored at the current address. Following the store operation, the contents of the current address are read, and then displayed in the data field of the displays.

When writing into RAM, the data read back should be the same as the data stored. If attempting to store into ROM, the value of the ROM locations will not change, so the displayed value will be the same both before and after the store. When writing to certain registers in the ACIAs and PIAs, it is useful to be able to see the immediate effect of the changes.

EXAMPLE

First set to 8 bit data mode.

3 1

Now set current address to RAM.

D 1 0 0 0 ENTER

Location \$1000, contents unknown, is displayed. Now change to \$45

C 4 5 ENTER

First set to 32 bit data mode.

3 3

Now set current address to RAM.

D 1 0 0 0 ENTER

Location \$1000 is displayed. Now change to \$80010221

C 8 0 0 1 0 2 2 1 ENTER

00001000

ADDRESS

45

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

INSTRUCTION

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 1

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 2

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 3

●○○○○○●○○○

STATUS

○○○○○○○○○○

00001000

ADDRESS

8001

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

INSTRUCTION

0221

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 1

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 2

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 3

●○○○○○●●○○○

STATUS

○○○○○○○○○○

Example: Change of Memory in data mode

Changing Registers

SUMMARY

C <data> **ENTER** causes the data, entered in hex, as <data> to be stored in the current register. The current register is displayed at the end of the command. (must be in the register mode)

The *change* command takes the entered value, and stores it at the current register. There is only one change command, with different modes, so the correct mode must be selected before starting the *change* command. To change to mode, see the *change mode* command. This description applies in the modes treating memory as 8, 16 or 32 bit quantities.

After entering the **C** of the change command, the data area is cleared, and a cursor is placed in the first byte of the data area. The data must be entered in hexadecimal, followed by the **ENTER** key. If fewer than the maximum number of digits are keyed, it is assumed that the digits keyed go in the least significant digits, while the remaining digits (the most significant) are filled with zeroes.

Once the **ENTER** key is pressed, the data is stored in the current register. The register is displayed in full 32 bit (16 for SR) form following the **ENTER** key.

EXAMPLE

Set to register mode.

3 4

Display data register 4 (D4).

D D 4

Change to \$00001010.

C 0 0 0 0 1 0 1
0 ENTER

d 4

ADDRESS

0 0 0 0

INSTRUCTION

0 0 0 0

EXTENSION 1

EXTENSION 2

EXTENSION 3

STATUS

STATUS

d 4

ADDRESS

0 0 0 0

INSTRUCTION

1 0 1 0

EXTENSION 1

EXTENSION 2

EXTENSION 3

STATUS

STATUS

Example: Change of Registers

Displaying Memory as Instructions

SUMMARY

While in the instruction mode, use the *display instruction* command to display the contents of memory as disassembled instructions in both hexadecimal and binary displays.

(Must be in instruction mode)

D < address> **ENTER** displays the contents of < address>

All *display* commands are essentially similar, and they depend on the mode for correct operation (see *change mode* command for setting the mode). The *display instruction* command takes the address given to it, and displays the contents of it as 68000 machine instructions. The instruction is disassembled by Petebug to find its length, the length is used to display the correct number of 16 bit words in both the 7 Segment hexadecimal displays and the binary LED groups.

The address used for the *display* command must be given in hexadecimal. It may be from 0 to 8 hex digits in length, but due to the 24 bit addressing used by the 68000, the uppermost two digits in an 8 digit address will be ignored. Since 68000 instructions are 16 bits, the address given must be even, or an error will be displayed. Address input must be terminated by the **ENTER** key. If no address is given, it is presumed to be 0.

Using the display command sets the *current address* to the value given by the user. The current address is used by the *forward*, *backward* and *change* commands.

EXAMPLE

Press:

3 0

To set to
instruction mode

D F F 8 1 5 E ENTER

To display the
instruction at

\$ F F 8 1 5 E

00FF815E

ADDRESS

207C

0000

0802

INSTRUCTION

○ ○ ● ○ ○ ○ ○ ○ ○ ○ ● ● ● ● ○ ○

EXTENSION 1

○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○

EXTENSION 2

○ ○ ○ ○ ● ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ● ○

EXTENSION 3

○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○

STATUS

● ○ ○ ○ ○ ○ ○ ○ ○ ○
○ ○ ○ ○ ○ ○ ○ ○ ○ ○

Example: Display Instruction command

Displaying Memory as Data (8/16/32 bit)

SUMMARY

While in the appropriate data mode (modes 1 through 3), use the *display data* command to display the contents of memory as data of that size.

(Must be in modes 1-3)

D < address> **ENTER** displays the contents of < address>

All *display* commands are essentially similar, and they depend on the mode for correct operation (see *change mode* command for setting the mode). The *display data* command takes the address given to it, and displays the contents of it as 1, 2 or 4 bytes in the hexadecimal displays only.

The address used for the *display* command must be given in hexadecimal. It may be from 0 to 8 hex digits in length, but due to the 24 bit addressing used by the 68000, the uppermost two digits in an 8 digit address will be ignored. For the 16 and 32 bit modes, the address given must be even, since 68000 words always start at even addresses. Address input must be terminated by the **ENTER** key. If no address is given, it is presumed to be 0.

Using the *display* command set the *current address*, which is used by the *forward*, *backward* and *change* commands.

EXAMPLE

Press:

3 3

To set to 32
bit mode

D F F 8 1 5 E ENTER

To display data
at FF815E as 32
bit data.

00FF815E

ADDRESS

207C

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

INSTRUCTION

0000

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 1

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 2

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 3

Press:

3 1

To set to 8
bit mode.

D F F 8 1 5 E ENTER

To display data
at FF815E as 8
bit data.

00FF815E

ADDRESS

21

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

INSTRUCTION

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 1

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 2

○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

EXTENSION 3

○○○○○○○○○○

STATUS

○○○○○○○○○○

Example: Display Data command

Displaying the 68000 Registers

SUMMARY

While in the register mode, use the *display register* command to display the contents of 68000 registers.

(Must be in register modes)

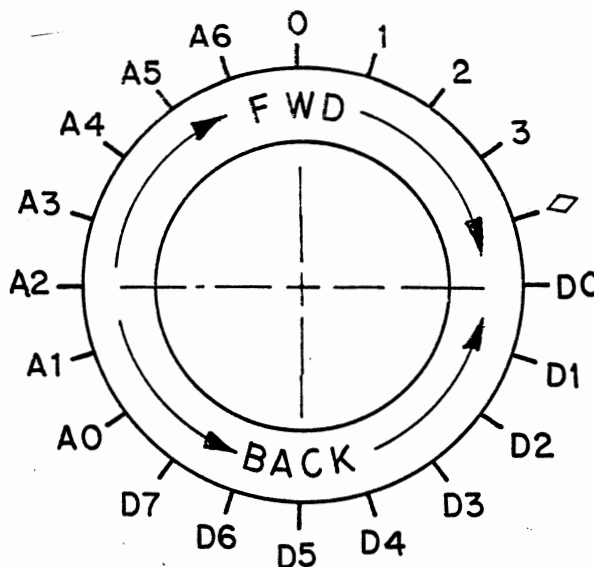
D <register name> displays the contents of <address>

All *display* commands are essentially similar, and they depend on the mode for correct operation (see *change mode* command for setting the mode). The *display register* command displays the contents of the given register as a 32 bit value in the hexadecimal displays only. (The status register is only displayed as 16 bits).

The register name is not terminated, as they are unique. For uses of the *forward* and *backward* commands, the registers are presumed to be in an order. The order can be read from the following table by reading down the left column, and then the right column. The register names must be taken from the following table.

Keys	Register	Keys	Register
0	USP	D, 6	D6
1	SSP	D, 7	D7
2	PC	A, 0	A0
3	SR	A, 1	A1
D, 0	D0	A, 2	A2
D, 1	D1	A, 3	A3
D, 2	D2	A, 4	A4
D, 3	D3	A, 5	A5
D, 4	D4	A, 6	A6
D, 5			

Using the display command sets the *current register*, which is used by the *forward*, *backward* and *change* commands.



EXAMPLE

Press:

3 4

To set to register mode.

D A 0

To display address register 0.

A0 | | | | | | | | ADDRESS

0000 | | | |

0000 | | | |

| | | |

| | | |

| | | |

INSTRUCTION
○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○
EXTENSION 1
○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○
EXTENSION 2
○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○
EXTENSION 3
○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

Press:

3 4

To set to register mode.

D 1

To display the supervisor stack pointer.

●○○○●○○○○○ STATUS
○○○○○○○○○○

SSP | | | | | | | | ADDRESS

0000 | | | |

04FC | | | |

| | | |

| | | |

| | | |

INSTRUCTION
○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○
EXTENSION 1
○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○
EXTENSION 2
○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○
EXTENSION 3
○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

●○○○●○○○○○ STATUS
○○○○○○○○○○

Example: Display Register command

Saving Data in EEROM

SUMMARY

5 <starting address of data to be stored in EEROM> **ENTER** <number of bytes to be stored> **ENTER** <offset address in EEROM at which data is to be stored> **ENTER** causes data to be transferred to EEROM for retention even with power off.

Upon depressing **5**, the prompt "START ____" will appear. This request is for the starting address at which the data to be stored is presently residing. Upon depressing **ENTER**, the prompt "BYTES ? ____" will appear. This request is for the number of bytes to be transferred to EEROM. Upon depressing **ENTER**, the prompt "EEADDR? ____" will appear. This request is for the relative (offset) EEROM address (000-7FF) at which data is to be stored. 000 is the first address in EEROM, 7FF is the last address in EEROM. Upon depressing **ENTER**, the message "DONE" will appear, indicating that the transfer of data has been successfully completed.

EXAMPLE

First put some recognizable data at memory location \$1000 (8 digits)

5 1 0 0 0 ENTER

\$1000 is the first address of data to be moved to EEROM.

S t A r t ADDRESS

1 0 0 0

-

STATUS

INSTRUCTION
EXTENSION 1
EXTENSION 2
EXTENSION 3

4 ENTER

Four bytes (eight hex digits) of data to be moved.

b Y t E S ? ADDRESS

4 -

STATUS

INSTRUCTION
EXTENSION 1
EXTENSION 2
EXTENSION 3

Example: Saving data in EEROM

EXAMPLE

0 ENTER

Move the data to EEROM starting at the first address in EEROM.

EEAddr?

ADDRESS

0_

INSTRUCTION

EXTENSION 1

EXTENSION 2

EXTENSION 3

STATUS

done

ADDRESS

0_

INSTRUCTION

EXTENSION 1

EXTENSION 2

EXTENSION 3

STATUS

Indicates that the transfer has been successfully completed.

Example: Saving data in EEROM

Retrieving Data from EEROM

SUMMARY

[7] <starting address of data to be retrieved from EEROM> [ENTER] <number of bytes to be retrieved> [ENTER] <offset address in EEROM from which data is to be retrieved> [ENTER] causes data to be transferred from EEROM to RAM.

Upon depressing [7], the prompt "START ____" will appear. This request is for the starting address in RAM at which data from the EEROM is to be stored. Upon depressing [ENTER], the prompt "BYTES? ____" will appear. This request is for the number of bytes to be transferred from EEROM. Upon depressing [ENTER], the prompt "EEADDR? ____" will appear. This request is for the relative (offset) EEROM address (000-7FF) from which data is to be transferred. 000 is the first address in EEROM, 7FF is the last address in EEROM. Upon depressing [ENTER], the message "DONE" will appear, indicating that the transfer of data to RAM has been successfully completed.

EXAMPLE

Recall the first 4 bytes (8 hex digits) from EEROM and put it at address \$1200 in RAM.

7 1 2 0 0 ENTER

\$1200 is the first RAM address where EEROM data is to be stored.

StArT

ADDRESS

1 2 0 0

INSTRUCTION

-

EXTENSION 1

EXTENSION 2

EXTENSION 3

STATUS

bYtES?

ADDRESS

4 ENTER

Four bytes (eight hex digits) of data to be transferred.

4

INSTRUCTION

EXTENSION 1

EXTENSION 2

EXTENSION 3

STATUS

Example: Retrieving Data from EEROM

EXAMPLE

O ENTER

Move the data from EEROM starting at the first address in EEROM.

E E A d d r ?

ADDRESS

O -

INSTRUCTION

INSTRUCTION

EXTENSION 1

EXTENSION 1

EXTENSION 2

EXTENSION 2

EXTENSION 3

EXTENSION 3

STATUS

d o n E

ADDRESS

Indicates that the EEROM data has been successfully transferred to RAM memory.

O -

INSTRUCTION

INSTRUCTION

EXTENSION 1

EXTENSION 1

EXTENSION 2

EXTENSION 2

EXTENSION 3

EXTENSION 3

STATUS

Example: Retrieving Data in EEROM

SUMMARY

The Petebug Commands are much easier to learn if practiced with the Trainer. The CSA Laboratory Manual contains several exercises that are simple to perform, yet improve keyboard and Petebug Command understanding through use. As suggested in Chapter 2, Master Mind is an ideal method for learning and practicing some fundamental commands and becoming familiar with the CSA Trainer's response. Some suggestions to improve User interaction with the CSA Trainer and Petebug are as follows:

1. Use RESET to begin all new operations or to start an operation over again from the beginning.
2. Always use BREAK (abort) if the MPU registers and Trainer status are to be saved (not cleared).
3. Visually check the display prior to pressing <ENTER> while inputting keyboard instructions or data, to ensure that your input is correct.
4. During the early learning process, slips of paper between each group of four (4) LEDs will simplify understanding the LED indicators.
5. Practice with the FWD and BACK commands, as well as the AUTO command. These commands will save time and ease data or instruction input when used properly.
6. Use the STEP command to debug programming problems. This command allows the User to display register status immediately following the execution of an instruction.
7. When entering a group of data or instructions and an error occurs, no need to start over, use the CHANGE command to correct the error and proceed.
8. As User progress and complexity of input becomes more involved, plan the input prior to keyboard entry. Know exactly what you need to do.
9. Follow directions exactly when entering programs from the CSA Laboratory Manual or User programs. While operating at the hexadecimal or binary level to input to the MPU, there is no latitude for error.
10. Pay attention to detail, use notes to track progress, and know the options and capabilities of the CSA Trainer system.

In the Chapters that follow (5, 6, and 7) the Motorola Specification Sheets will be presented. These spec sheets are the

same as those used by engineers and designers in industry. Although the material contained in the Spec Sheets is available in other User references, it is presented in this manual for quick reference while engaged in programming the MPU and working with the CSA Trainer.

CHAPTER 5

INTRODUCTION

** MC68000 ARCHITECTURE - This chapter contains the latest Motorola MC68000 timing, control signal, and MC68000 operation Data Sheets. This information is clearly presented, complete, yet brief in presentation and of the same format and content that students will normally use in their careers. These sheets are the most frequent form of updates and changes to microprocessor characteristics that are made available. Students, especially future design engineers, would increase their career opportunities by becoming familiar with the manner that data and information is presented in these types of "Spec Sheets."

To aid in use and provide quick access, the Spec Sheets have been divided into three groups, as follows:

- Chapter 5 - MC68000 Architecture
- Chapter 6 - MC68000 Instruction Set
- Chapter 7 - MC68000 Support Chips

Each group (chapter) is preceded by a CSA introductory paragraph and a "Quick Reference Guide." The introductory paragraph will identify the type of data that is contained within the chapter. The Quick Reference Guide is designed to locate the exact data the User is seeking.

The data contained in this chapter is related to the MPU internal timing cycles and external bus control cycles. Particular attention is directed to the READ, WRITE, Bus Arbitration, and Exception Processing cycles. It is imperative that these functions be understood to properly program the MPU. Almost as important are the Memory and Data organization, the Exception Vectors and External Interrupt operations. A Quick Reference Guide is provided for access to the data in this chapter.

Quick Reference Guide MC 68000 Architecture

Data	Page
Identity and Physical Characteristics (Registers and Pin Out)	1
Ratings and Electrical Characteristics	2
Loading and Clock Time	3
AC Electrical Specifications	4
Read Cycle Timing	5
Write Cycle Timing	6
Bus Arbitration	7
Signal Description	8
Register and Data Organization	10
Data Organization in Memory	11
Word/Byte Flow Charts/Timing Cycles	13
Read-Modify-Write Flow Chart	16
Bus Arbitration Cycle Flow Chart	17
Bus Arbitration Control	18
Bus Error And Halt Operation	18
Exception Sequence	20
*DTACK, *BERR, and *HALT	23
RESET Operation	24
Processing/Privilege States	24
Exception Processing	25
Exception Vector Assignments	26
Exceptions/RESET/Interrupts	27
Interrupt Acknowledge Sequence Flow Chart	28
Interrupts/Traps/Tracing/Bus Error	29
Address Error/Interface MC68000 Peripherals	30
M6800 Interfacing Flow Chart	31
M6800 Timing Diagram (Best)	32
M6800 Timing Diagram (Worst)	33
Interrupt Operation	34
Data Types/Addressing Modes	35
*MC68000 Dimensions	50