

MOTOROLA
Semiconductor Products Inc.

MEK6800D2

MANUAL

Circuit diagrams external to Motorola products are included as a means of illustrating typical Microprocessor applications; consequently, complete information sufficient for construction purposes is not necessarily given. The information in this manual has been carefully checked and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies. Furthermore, such information does not convey to the purchaser of the semiconductor devices described any license under the patent rights of Motorola Inc. or others.

Motorola reserves the right to change specifications without notice.

EXORciser, JBUG and MINibug are trademarks of Motorola Inc.

Second Edition
© MOTOROLA INC., 1977
First Edition © 1976
"All Rights Reserved"

Printed in U.S.A.

TABLE OF CONTENTS

CHAPTER 1: Introduction

1-1	General Description and Capability	1-1
1-2	Preparation for Use	1-4
1-2.1	Construction Hints	1-5
1-3	Start-up Procedure	1-6
1-4	Operating Procedures	1-6
1-4.1	Memory Examine and Change	1-7
1-4.2	Escape (Abort)	1-7
1-4.3	Register Display	1-7
1-4.4	Go to User Program	1-9
1-4.5	Punch from Memory to Tape	1-9
1-4.6	Load from Tape to Memory	1-9
1-4.7	Breakpoint Insertion and Removal	1-10
1-4.8	Trace One Instruction	1-10
1-4.9	Calculate Offset to Branch Destination	1-11
1-5	Operating Example	1-11

CHAPTER 2: Hardware Description

2-1	General Description	2-1
2-2	Memory Organization	2-1
2-3	Input/Output Devices	2-3
2-4	System Clock	2-3
2-5	Keyboard/Display	2-3
2-6	Trace (Execute Single Instruction)	2-4
2-7	Audio Cassette Interface	2-4
2-8	Kit Expansion	2-10

CHAPTER 3: Software Description (JBUG Monitor)

3-1	General Description	3-1
3-2	Restart/Initialization Routine	3-3
3-3	Display Routine	3-3
3-4	Keyboard Scan and Decode Routine	3-4
3-5	Memory Examine/Change Routine	3-7
3-6	Register Display Routine	3-8
3-7	Punch and Load Routines	3-9
3-8	Interrupt Handling Routine	3-11

Appendix 1: Assembly Listing of JBUG Monitor	A1-1
--	------

Appendix 2: Assembly Drawings and Parts Lists	A2
---	----

Appendix 3: Schematic Diagrams	A3-1
--------------------------------------	------

Appendix 4: Power Supply Information	A4-1
--	------

CHAPTER 1

INTRODUCTION

1-1 GENERAL DESCRIPTION AND CAPABILITY

This manual provides a general description and operating instructions for the Motorola MEK6800D2 Evaluation Kit II. The Kit, when assembled, is a fully functional microcomputer system based on the MC6800 Microprocessing Unit (MPU) and its family of associated memory and I/O devices. The family is described in the *M6800 Microcomputer System Design Data* book (included with the Kit) and in the *M6800 Microprocessor Applications Manual*. Detailed programming information is included in the *M6800 Programming Reference Manual*.

The MEK6800D2 is designed to provide a completely self-contained method for evaluating the characteristics of the M6800 family. The standard Kit includes the following devices:

Qty.	Device
1	MC6800 MPU
1	MCM6830 ROM with JBUG Monitor (SCM44520P)
3	MCM6810 RAM (128 x 8)
2	MC6820 Peripheral Interface Adapter (PIA)
1	MC6850 Asynchronous Communications Interface Adapter (ACIA)
1	MC6871B Clock Generator

As assembled Kit is shown in Figure 1-1-1 (all components shown are included with the standard Kit.)

The Microcomputer Module printed circuit board is preengineered to accept the following additional components for expanding its capability:

Qty.	Device
2	MCM6810 RAM (128 x 8)
2	MCM68708 EPROM (Equivalent to 2708)
3	MC8T97 Buffer
2	MC8T26 Bidirectional Buffer

The expansion capability provides for a variety of user operating modes.

The integral Keyboard/Display Module can be used in conjunction with the JBUG monitor program for entering and debugging user programs. Programs can also be loaded and dumped via the Audio Cassette Interface. The Keyboard, Display and Audio Cassette circuitry are on a separate printed circuit board so that the ACIA and a second PIA are available if the user has access to an RS-232 or TTY terminal. Wire-wrap space for up to twenty 16-pin DIP packages is available for user designed circuitry on the Microcomputer Module. A user generated terminal control program designed to interface with either the PIA or the ACIA can be entered via the integral keyboard. Alternatively, the Kit will accept (in place of JBUG) the Motorola MINIbug II monitor program. MINIbug II has monitor and diagnostic capabilities similar to JBUG but is intended for use with RS-232 and TTY type terminals. (See Appendix E of the *Programming Reference Manual* included in the Kit.)

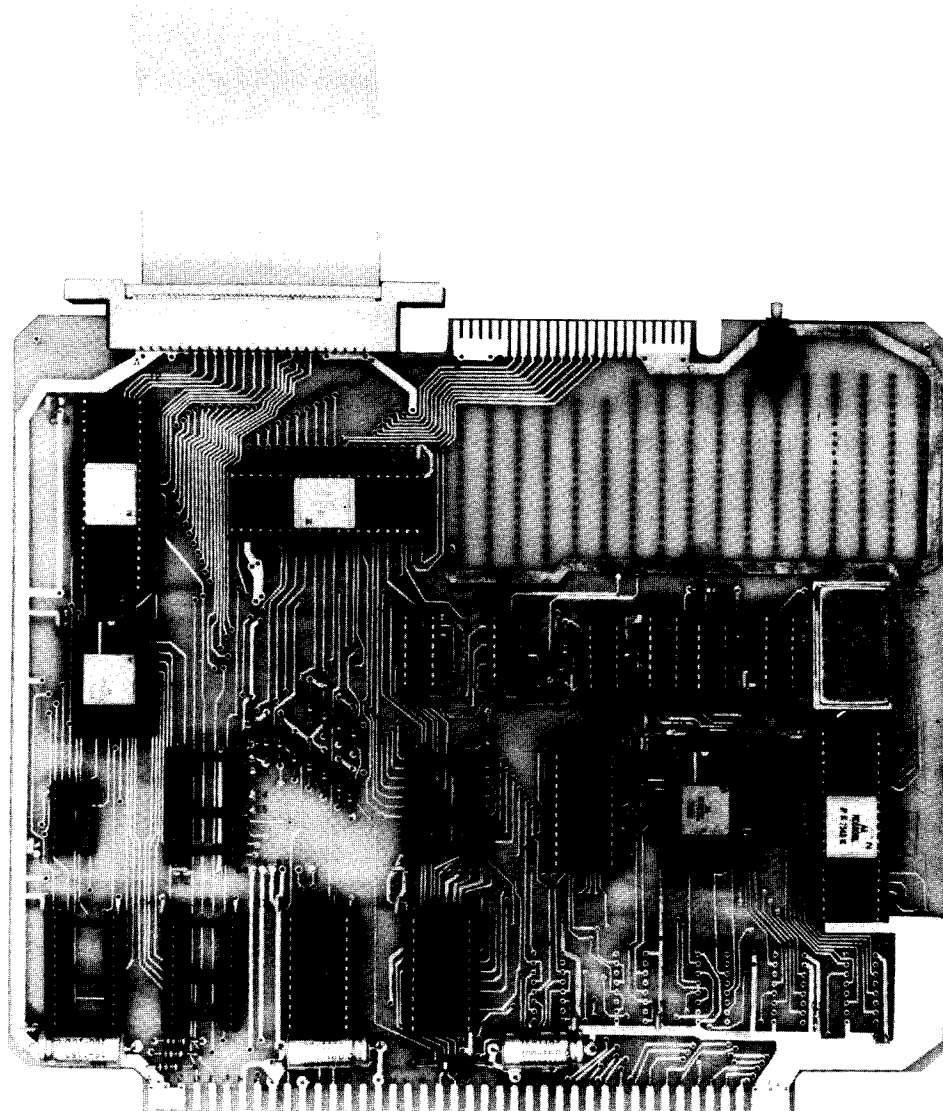
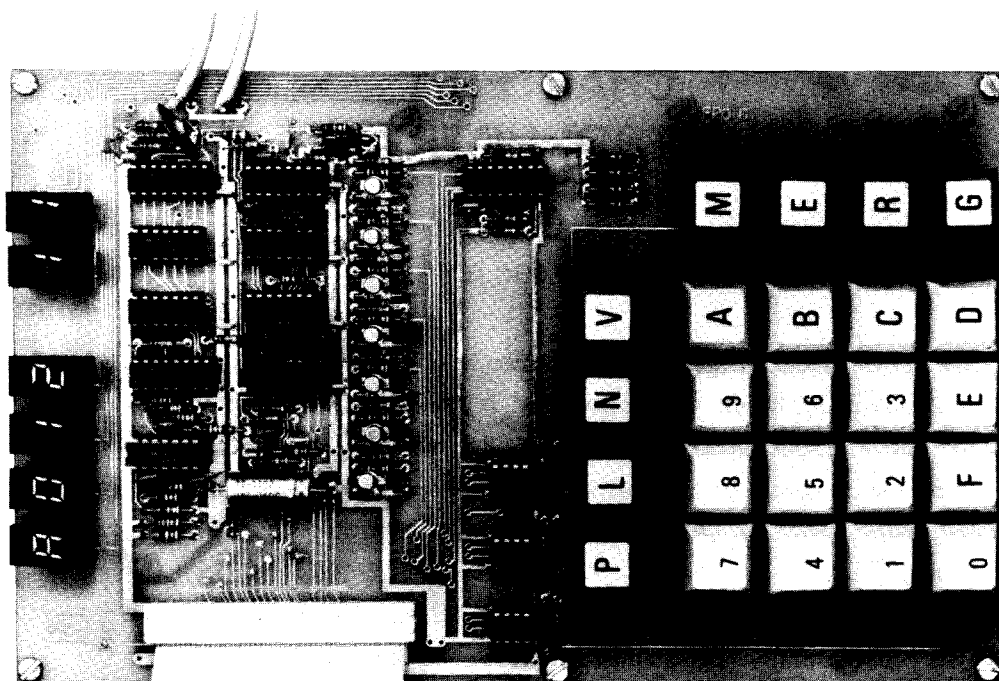


FIGURE 1-1-1.

The Kit also permits several different memory configurations. The two MCM6810 128 x 8 RAMs provided with the standard Kit will accommodate programs of up to 256 bytes in length (the third MCM6810 is reserved for use by the monitor program). Addition of the two additional optional RAMs expands the capability to 512 bytes. Strapping options for the additional ROM sockets permits any of the following combinations:

- 1024 bytes in 512 x 8 bit PROMs (MCM7641)
- 2048 bytes in 1024 x 8 bit EPROMs (MCM68708)
- 2048 bytes in 1024 x 8 bit Mask-Programmed ROMs (MCM68308 — same pin-out as MCM68708)
- 4096 bytes in 2048 x 8 bit Mask-Programmed ROMs (MCM68316 — same pin-out as MCM68708 except EPROM programming pin is used as additional addressing pin.)

The general memory organization of the Kit is shown in Figure 1-1-2.

Adding the optional buffers in the spaces provided upgrades the Kit to EXORciser-compatible status; hence, all the EXORciser I/O and Memory modules (see included data sheets) can also be used with the Kit. For example, addition of MINIBug II, an 8K Memory board, and the EXORciser's Resident Editor/Assembler to the Microcomputer Module creates a complete development/prototyping tool.

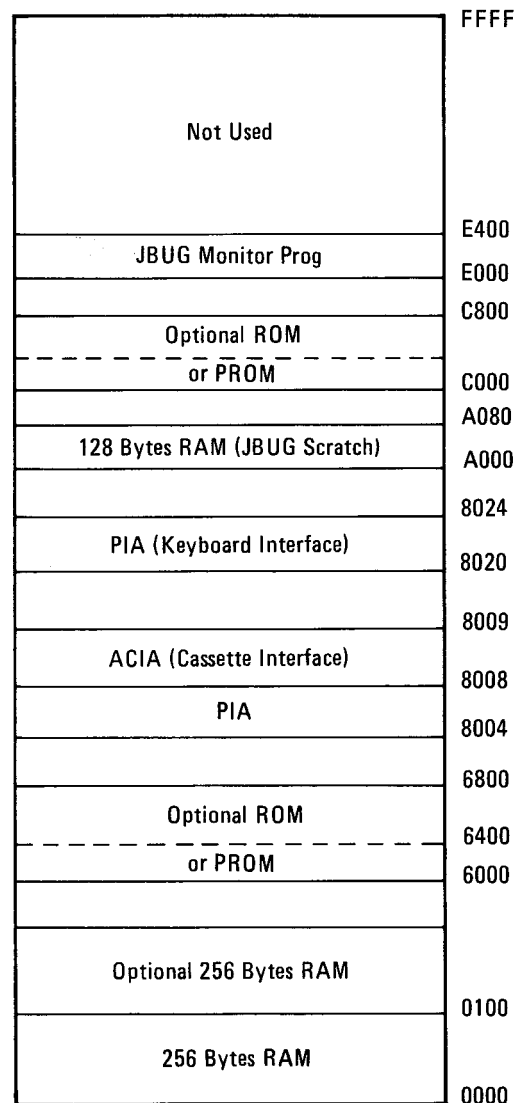


FIGURE 1-1-2. Memory Map for MEK6800D2

PREPARATION FOR USE AND OPERATION PROCEDURES

The Kit can be assembled by referring to the assembly diagrams of Figures A2-a and A2-b (Appendix 2) for component placement. Recommended procedures for the handling of MOS and CMOS integrated circuits are reviewed in Table 1-2-1 and should be followed during assembly. The Kit is completely self-contained and required only the addition of a 5-volt dc power supply. Additional ± 12 -volt dc supplies are required only if electrically programmable read only memories (EPROMs) are used or if RS-232 capability is to be added to the Kit. The switches, connectors and display indicators are identified in Figure 1-1-3.

Caution must be exercised to avoid any electrostatic or high-voltage charge from coming in contact with the MOS gate elements. The gate oxide is approximately 1000 to 1200 Å thick and can be ruptured by static potentials as small as 80 volts. Most MOS circuits employ various input protective schemes. However, an electrostatic charge may still cause damage to the gate oxide during the finite time required for the protective device to turn on.

The following handling precautions are recommended for MOS circuits:

1. All MOS devices should be stored or transported in conductive material so that all exposed leads are shorted together. MOS devices must not be inserted into conventional plastic foam or plastic trays of the type used for the storage and transportation of other semiconductor devices.
2. All MOS devices should be placed on a grounded bench surface and the operators should ground themselves prior to handling devices. This is done most effectively by having the operator wear a grounded conductive wrist strap.
3. Silk or Nylon clothing should not be worn while handling MOS circuits.
4. Do not insert or remove MOS devices from test sockets with power applied.
5. Check all power supplies to be used for testing MOS devices to be certain no voltage transients are present.
6. When lead straightening or hand soldering is necessary, provide ground straps for the apparatus used.
7. Do not exceed the maximum electrical voltage ratings specified by the manufacturer.
8. Double check test equipment setup for proper polarity of voltage before conducting parametric or functional testing.
9. Cold chambers using CO₂ for cooling should be equipped with baffles, and devices must be contained on or in conductive material.
10. All unused device inputs should be connected to VDD or VSS.
11. All power should be turned off in a system before printed circuit boards containing MOS devices are inserted or removed.
12. All printed circuit boards containing MOS devices should be provided with shorting straps across the edge connector when being carried or transported.

TABLE 1-2-1: MOS Handling Recommendations

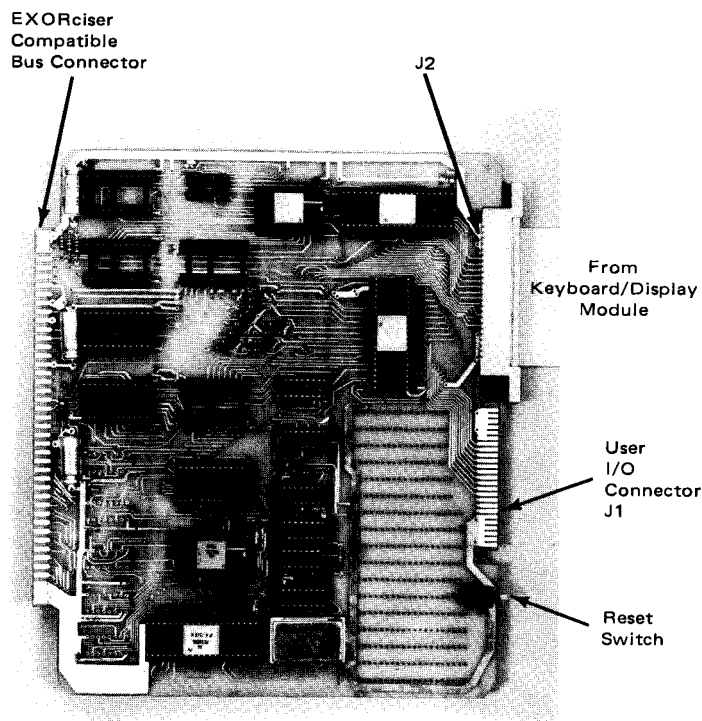


FIGURE 1-1-3a. Microcomputer Module

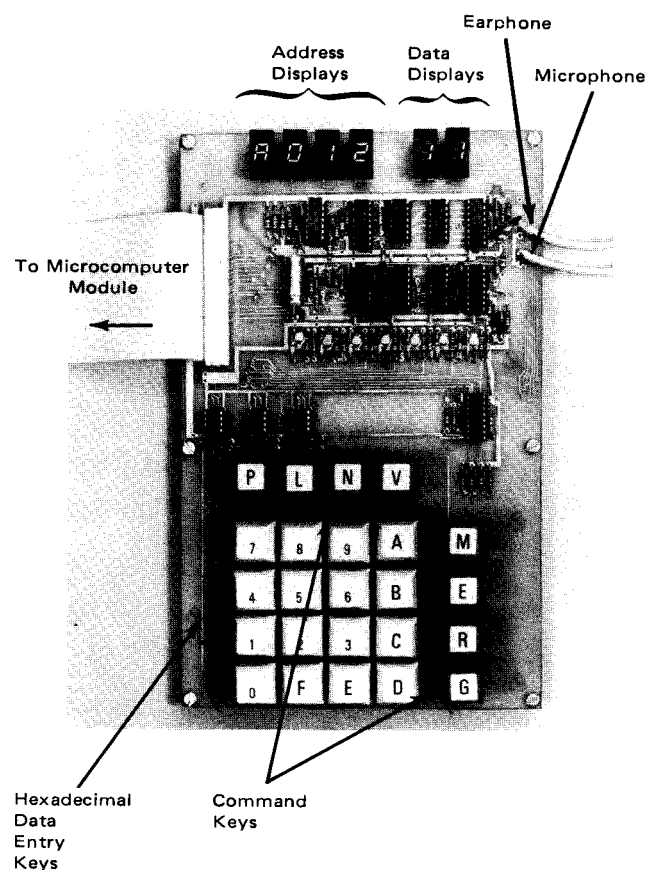


FIGURE 1-1-3b. Keyboard/Display Module

1-2.1 CONSTRUCTION HINTS

The 24-pin socket supplied for the clock must be modified to fit the PC Board. This can be done by removing the protective strips on the bottom of the socket and pulling out unwanted pins from the bottom. The pins that must be removed are 2, 4, 6, 8, 9, 10, 11, 14, 15, 16, 17, 19, 21, and 23.

The Keyboard has 6 pre-drilled holes for use with standoffs or machine screws to support the board while in use. It is recommended that the board be supported above the bench a minimum of 1/4 inch to prevent accidentally shorting conductors on the bottom of the board.

When inserting CMOS devices, it is recommended that a low wattage soldering iron with a grounded tip be used. This will prevent damaging the part. Another alternative would be using sockets for the parts.

The cable assembly consists of five items.

- | | |
|--|-----------------------|
| 1. Edge connector | (part no. 3415-0001) |
| 2. Edge connector cover | (part no. 3415) |
| 3. 50 pin PC Board connector | (part no. 3426-0000T) |
| 4. PC Board connector cover | (part no. 3426) |
| 5. Approximately 3 feet of 50 conductor flat cable | (part no. 3365) |

The cable may be assembled as follows:

Step 1: Solder the 50-pin PC board connector (3426-0000T) in place on the Keyboard/Display Module.

- Step 2: Remove protective liner from the PC Board Connector Cover (3426) by first pressing along length of liner (this will insure good adhesive coverage) and then applying lateral thumb pressure on liner to displace it enough to be peeled off.
- Step 3: Press deeply ribbed side of cable (3365) into alignment grooves of cover, positioning it as required in step 4. Check visually to insure that the cable is aligned in cover grooves and is even with the edge of the connector.
- Step 4: Place cap and cable over PC Board connector with the cable running away from the Keyboard/Display Module with the red stripe corresponding to pin 1 of the connector. Then press the assembly together using a bench vise.
- Step 5: Repeat steps 2 and 3 with edge connector and cap on the other end of the cable, keeping the red conductor aligned with pin 1 of the edge connector. Press this assembly together using the vise.
- Step 6: The female edge connector will now mate with the male edge connector (J2) on the Microcomputer Module. The female conductor labeled "1" should align with the male conductor labeled "A". (The cable "approaches" the back of the Microcomputer Module.)

1-3 **START-UP PROCEDURE**

Connect the cable attached to the Keyboard/Display Module to connector J2 on the Microcomputer Module. Apply 5-volt dc power. Pushing the reset switch on the Microcomputer Module should now cause the JBUG prompt symbol, "dash", to be displayed in the left-most display indicator on the Keyboard/Display Module. The remaining five displays will be blanked. The JBUG control and monitor program is now in operation and any of the functions described in the next section may be invoked by means of the data and command keys on the Keyboard/Display Module.

1-4 **OPERATING PROCEDURES**

The Keyboard/Display Module, in conjunction with JBUG, provides a means of examining operation of the Microcomputer Module and entering and trouble-shooting programs. The Keypad has sixteen keys labeled 0-F for entry of hexadecimal data and eight keys for commanding the following functions:

- M — Examine and Change Memory
- E — Escape (Abort) from Operation in Progress
- R — Examine Contents of MPU Registers P, X, A, B, CC, S
- G — Go to Specified Program and Begin Execution of Designated Program
- P — Punch Data from Memory to Magnetic Tape
- L — Load Memory from Magnetic Tape
- N — Trace One Instruction
- V — Set (and Remove) Breakpoints

Operating procedures for each of these functions are described in the following paragraphs. The display should be showing the prompt "dash" before any command is invoked.

1-4.1 **MEMORY EXAMINE AND CHANGE (M)**

This function permits examination and, if necessary, change of memory locations. A map of the MC6800 instructions is included as Table 1-4.1-1 and is useful in translating memory data to instruction mnemonics.

Open the memory location to be examined by entering the address (as 4-digits of hex via the hex keypad) followed by closure of the M key (hhhhM). The display will now show the address that was entered in its group of four displays on the left and the contents in the two on the right. The user at this point has three options: (1) Leave this location unchanged and move to the next location by closing the G key. The new address and its data would then be displayed. (2) Change the data by simply entering the new data via the hex keypad (hh). In this case the display would then be showing the new data that was entered. In the event that an attempt is made to change Read Only Memory (ROM), the display will continue to show the original data. (3) Close the Memory Examine function by means of the E key. Closure of the E key will return operation to the monitor and the prompt will again be displayed.

1-4.2 **ESCAPE (ABORT)**

This function provides an orderly exit from the other functions and/or user programs. Examples of its use are included in the accompanying descriptions of the other functions.

1-4.3 **REGISTER DISPLAY (R)**

This function permits examination of the MPU's registers and may be invoked at any time the JBUG prompt is being displayed by closing the R key. Following closure of R, the display will show a 4-digit hex value, the present contents of the Program Counter. The remaining registers may now be examined by sequencing with the G key and will appear in the following order: Index Register, Accumulator A, Accumulator B, Condition Code Register, Stack Pointer.¹

This display is circular, i.e., a G key closure following display of the Stack Pointer will cause the Program Counter to be displayed again. The E key may be used to escape back to the monitor at any point in the display sequence. If required the contents of any register can be changed by using the Memory Change function. The monitor executed an interrupt sequence when R was invoked. In servicing an interrupt, the MC6800 saves its registers on a stack in memory (it is these memory locations that the R function "examines"). On exit from the R interrupt service routine, the MPU retrieves these values and reloads its registers; hence if the data on the stack is changed with the M function, the new data will go into the MPU. The following locations are used to stack the registers:

\$A008² — High order byte of Stack Pointer

\$A009 — Low order byte of Stack Pointer

S + 1 — Condition Code Register

S + 2 — Accumulator B

S + 3 — Accumulator A

S + 4 — High order byte of Index Register

¹It is a characteristic of the display routine that the value displayed for the Stack Pointer is seven less than the actual value.

²In this manual, hexadecimal data is identified by preceeding it with a dollar sign symbol, \$.

LSB MSB	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	*	NOP (INH)	*	*	*	TAP (INH)	TPA (INH)	INX (INH)	DEX (INH)	CLV (INH)	SEV (INH)	CLC (INH)	SEC (INH)	CLI (INH)	SEI (INH)	
1	SBA	CBA	*	*	*	TAB (INH)	TBA (INH)	*	DAA (INH)	*	ABA (INH)	*	*	*	*	*
2	BRA (REL)	*	BHI (REL)	BLS (REL)	BCC (REL)	BCS (REL)	BNE (REL)	BEQ (REL)	BVC (REL)	BVS (REL)	BPL (REL)	BMI (REL)	BGE (REL)	BLT (REL)	BGT (REL)	BLE (REL)
3	TSX (INH)	INS (INH)	PUL (A)	PUL (B)	DES (INH)	TXS (INH)	PSH (A)	PSH (B)	*	RTS (INH)	*	RTI (INH)	*	*	WAI (INH)	SWI (INH)
4	NEG (A)	*	*	COM (A)	LSR (A)	*	ROR (A)	ASR (A)	ASL (A)	ROL (A)	DEC (A)	*	INC (A)	TST (A)	*	CLR (A)
5	NEG (B)	*	*	COM (B)	LSR (B)	*	ROR (B)	ASR (B)	ASL (B)	ROL (B)	DEC (B)	*	INC (B)	TST (B)	*	CLR (B)
6	NEG (IND)	*	*	COM (IND)	LSR (IND)	*	ROR (IND)	ASR (IND)	ASL (IND)	ROL (IND)	DEC (IND)	*	INC (IND)	TST (IND)	JMP (IND)	CLR (IND)
7	NEG (EXT)	*	*	COM (EXT)	LSR (EXT)	*	ROR (EXT)	ASR (EXT)	ASL (EXT)	ROL (EXT)	DEC (EXT)	*	INC (EXT)	TST (EXT)	JMP (EXT)	CLR (EXT)
8	SUB (A) (IMM)	CMP (A) (IMM)	SBC (A) (IMM)	*	AND (A) (IMM)	BIT (A) (IMM)	LDA (A) (IMM)	*	EOR (A) (IMM)	ADC (A) (IMM)	ORA (A) (IMM)	ADD (A) (IMM)	CPX (A) (IMM)	BSR (REL)	LDS (IMM)	*
9	SUB (A) (DIR)	CMP (A) (DIR)	SBC (A) (DIR)	*	AND (A) (DIR)	BIT (A) (DIR)	LDA (A) (DIR)	STA (A) (DIR)	EOR (A) (DIR)	ADC (A) (DIR)	ORA (A) (DIR)	ADD (A) (DIR)	CPX (A) (DIR)	*	LDS (DIR)	STS (DIR)
A	SUB (A) (IND)	CMP (A) (IND)	SBC (A) (IND)	*	AND (A) (IND)	BIT (A) (IND)	LDA (A) (IND)	STA (A) (IND)	EOR (A) (IND)	ADC (A) (IND)	ORA (A) (IND)	ADD (A) (IND)	CPX (A) (IND)	JSR (IND)	LDS (IND)	STS (IND)
B	SUB (A) (EXT)	CMP (A) (EXT)	SBC (A) (EXT)	*	AND (A) (EXT)	BIT (A) (EXT)	LDA (A) (EXT)	STA (A) (EXT)	EOR (A) (EXT)	ADC (A) (EXT)	ORA (A) (EXT)	ADD (A) (EXT)	CPX (A) (EXT)	JSR (EXT)	LDS (EXT)	STS (EXT)
C	SUB (B) (IMM)	CMP (B) (IMM)	SBC (B) (IMM)	*	AND (B) (IMM)	BIT (B) (IMM)	LDA (B) (IMM)	*	EOR (B) (IMM)	ADC (B) (IMM)	ORA (B) (IMM)	ADD (B) (IMM)	*	*	LDX (IMM)	*
D	SUB (B) (DIR)	CMP (B) (DIR)	SBC (B) (DIR)	*	AND (B) (DIR)	BIT (B) (DIR)	LDA (B) (DIR)	STA (B) (DIR)	EOR (B) (DIR)	ADC (B) (DIR)	ORA (B) (DIR)	ADD (B) (DIR)	*	*	LDX (DIR) ^(B)	STX ^(B) (DIR)
E	SUB (B) (IND)	CMP (B) (IND)	SBC (B) (IND)	*	AND (B) (IND)	BIT (B) (IND)	LDA (B) (IND)	STA (B) (IND)	EOR (B) (IND)	ADC (B) (IND)	ORA (B) (IND)	ADD (B) (IND)	*	*	LDX (IND)	STX (IND)
F	SUB (B) (EXT)	CMP (B) (EXT)	SBC (B) (EXT)	*	AND (B) (EXT)	BIT (B) (EXT)	LDA (B) (EXT)	STA (B) (EXT)	EOR (B) (EXT)	ADC (B) (EXT)	ORA (B) (EXT)	ADD (B) (EXT)	*	*	LDX (EXT)	STX (EXT)

DIR = Direct Addressing Mode
 EXT = Extended Addressing Mode
 IMM = Immediate Addressing Mode
 IND = Index Addressing Mode
 INH = Inherent Addressing Mode
 REL = Relative Addressing Mode
 A = Accumulator A
 B = Accumulator B
 *Unimplemented Op Code

TABLE 1-4.1-1. M6800 Instruction Map

- S + 5 — Low order byte of Index Register
- S + 6 — High order byte of Program Counter
- S + 7 — Low order byte of Program Counter

where “S” is the current Stack Pointer as saved in \$A008 and \$A009. Note that it is necessary to exit the R display function and enter the M in order to change register values.

1-4.4 GO TO USER PROGRAM (G)

If the Prompt is being displayed, and assuming that a meaningful program has been previously entered, the MPU can be directed to go execute the program simply by entering the starting address of the program (via the hex keypad) followed by closure of the G key (hhhhG). The resulting blanking of the displays is an indication that the MPU has left the monitor program and is executing the user's program. The MPU will continue executing the user program until either an Escape (E key) is invoked or the program “blows”. Control, indicated by the prompt “dash”, can normally be obtained with the E key. It is possible that an incorrect program could have caused the monitor's variable data to be modified. In this case, it is necessary to regain control using the reset switch on the Microcomputer Module.

1-4.5 PUNCH FROM MEMORY TO TAPE

The Punch function allows the user to save selected blocks of memory on ordinary audio tape cassettes. Before invoking Punch, the Memory Change function should be used to establish which portion of memory is to be recorded. Using Memory Change, enter the desired starting address into locations \$A002 and \$A003 (high order byte into \$A002, low order byte into \$A003). Similarly, enter the high and low order bytes of the desired ending address into \$A004 and \$A005, respectively. Escape from Memory Change via the E key, thus obtaining the monitor prompt dash. With the audio recorder's microphone input connected to the corresponding point on the Keyboard/Display Module and the prompt present, the Punch function is performed as follows. Position the tape as desired (fully rewound is recommended) and put the recorder in its record mode. Close the P key. The prompt will disappear during the Punch process and then re-appear to indicate that the Punch operation is completed. Typically, the prompt is “off” for over 30 seconds since the recording format specifies that a thirty second header of all ones be recorded ahead of the data. See sections 2-7 and 3-7 for additional details on the recording format.

1-4.6 LOAD FROM TAPE TO MEMORY

The Load function can be used to retrieve from audio magnetic tape data that was recorded using the Punch function described in the preceding section. With the audio recorder's earphone output connected to the corresponding input on the Keyboard/Display Module (and with the monitor prompt present on the display), the Load function is performed as follows. To load the desired record, position the tape at the approximate point from which the Punch was started and then put the recorder into its playback mode. Close the L key. The prompt will disappear, then re-appear when the Load function is completed. After the prompt re-appears, the Memory Examine function can be used to examine locations \$A002 and \$A003. They will contain the beginning address of the block of data that was just moved into memory. The end address is not recovered by the function, hence the data in locations \$A004 and \$A005 is not significant during the Load function.

1-4.7 **BREAKPOINT INSERTION AND REMOVAL (V)**

Because of the difficulty in analyzing operation while a program is executing, it is useful during debug to be able to set breakpoints at selected places in the program. This enables the user to run part of the program, then examine the results before proceeding. The breakpoints are set by entering the hex address of the desired breakpoint followed by a V key closure (hhhhV). This may be repeated up to five times. The breakpoint entry function can be exited after any entry by using the E key. The monitor program will retain all the breakpoints until they are cleared.

If at any time an hhhhV entry is made and the hhhh (hex data) does not appear on the display, there were already five breakpoints stored and the last one was ignored. At any time the prompt is displayed, entry of a V command not preceded by hex data will cause the current breakpoints to be removed. If a breakpoint is entered and the program is subsequently executed to that point, the display will show the current value of the Program Counter in the four indicators on the left. (This will be the same as the breakpoint address that was inserted.) The right hand two displays will contain the data stored at that location — that is, the operating code. At this point the G key can be used to sequence through the other MPU registers exactly as in the register display function. If it is desirable to proceed on from the breakpoint simply use E (to get the prompt) and then the G key. At this point, the MPU will reload its registers from the stack and continue with the user's program until another breakpoint is encountered or the E key is used again.

1-4.8 **TRACE ONE INSTRUCTION (N)**

The Trace function permits stepping through a program one instruction at a time. The Trace function can be invoked any time the user program is at a breakpoint or has been aborted with the E key. However, tracing cannot begin from start-up because the trace routine does not know where the starting address is. Therefore, an hhhhV command must be given at least once before Trace can be used.

Enter the Trace function by first setting a breakpoint at the location from which it is desired to trace and then invoking hhhhG to begin program execution. The breakpoint can be set at the very beginning of the program if desired.³ Following the hhhhG command, the program will run to the breakpoint and stop, displaying the Program Counter as before. If the N key is now closed, the MPU executes the next program instruction and again halts. The display will then show the address of the next instruction (Program Counter) and the operating code located there. The G key can be used to sequence the other registers on to the display as for a breakpoint if desired. The N key can now be used to trace as many instructions as desired.⁴

The Trace function cannot be used directly to trace through user IRQ interrupts. The NMI is higher priority and will cause the IRQ to be ignored. Repeated attempts to execute the Trace command when user IRQ interrupts are active will result in JBUG continuously returning with the same address. See sections 2-6 and 3-8 of this manual and the *M6800 Microprocessor Applications Manual* for additional information.

³This procedure assumes the program is in RAM since breakpoints are handled by substituting an SWI for the op-code. If the program to be traced is entirely in ROM, use a convenient RAM location to insert a jump to the desired ROM address. Then set a breakpoint at the address of the jump instruction and proceed as above.

⁴It is a characteristic of the Trace function that all breakpoints in effect at the time Trace is invoked will be removed and must be re-installed following exit from Trace.

Interrupt service routines may be traced by setting a breakpoint at the beginning of the service routine. The Go function may then be used to start program execution, allowing a normal entry into the $\overline{\text{IRQ}}$ service routine. Once in the service routine, Trace can be used as usual. The E key may be used to exit from Trace at any time.

1-4.9 CALCULATION OF THE OFFSET TO A BRANCH DESTINATION

The instruction format for conditional branch instructions calls for the offset to the destination to be entered immediately following the branch instruction op-code as a signed two's complement number. Mental calculation of the offset is awkward due to the required two's complement format. A short program for making this calculation is included in JBUG (lines 62-70 of the assembly listing included as Appendix 1 of this manual). Use the following procedure with this program:

1. Obtain the prompt "dash" by escaping from the current operation.
2. Find the current value of the stack pointer by entering the Register Display.
3. Exit from Register Display and open memory location S+2, where S is the current value of the stack pointer as obtained in Step 2. S+2 is the location of the current stacked value of Accumulator B. Enter the high order byte of the destination address in this location. Next, enter the low order byte of the destination into Accumulator A in location S+3.
4. Put the high and low order bytes of the branch instruction's op-code address into S+4 and S+5, respectively. This loads the stacked Index Register with the op-code address.
5. Use the "E" key to exit from the Memory Examine/Change function and then enter \$E000G to begin executing the program starting at location \$E000 in JBUG.
6. The program runs to location \$E013 and hits the SWI breakpoint located there. Examine the contents of Accumulators A and B by invoking Register Display and sequencing through the Registers with the G key. The offset, in the correct form for entry in the program, is now in Acc.A. If Acc.B contains \$FF, the offset is valid (within the allowed range) and is in the negative direction. If Acc.B contains \$00, the offset is valid and in the positive direction. Any other value indicates that the destination is beyond the allowed range.

1-5 OPERATING EXAMPLE

The following example program is suitable for gaining familiarity with the JBUG monitor features. The program adds the five values in locations \$10 through \$14 using Acc. A and stores the final result in location \$15. The intermediate total is kept in Acc. A; Acc. B is used as a counter to count down the loop. The Index Register contains a "pointer" (i.e., X contains the address) of the next location to be added. The program, as follows, contains an error which will be used later to illustrate some of JBUG's features.

In the following listing, the leftmost column contains the memory address where a byte (8 bits) of the program will be stored. The next column contains the machine language op-code and data for a particular

microprocessor instruction. The next four columns contain the mnemonic representation of the program in assembler format.

				*
				* Add 5 numbers at locations 10-14
				* Put answer in location 15
				*
0020	8E	STRT	LDS \$FF	DEFINE STACK IN USER AREA
0021	00			
0022	FF			
0023	4F		CLRA	TOTAL # 0
0024	C6		LDAB #4	INITIALIZE COUNTER
0025	04			
0026	CE		LDX #\$10	POINT X TO LOCATION 10
0027	00			
0028	10			
0029	AB	LOOP	ADDA O,X	ADD 1 LOCATION TO TOTAL
002A	00			
002B	08		INX	POINT X TO NEXT LOCATION
002C	5A		DECB	DONE ALL 5 LOCATIONS?
002D	26		BNE LOOP	BRANCH IF NOT.
002E	FA			
002F	97		STAA \$15	SAVE ANSWER
0030	15			
0031	3F		SWI	GO TO JBUG

A detailed procedure for entering and debugging this program is shown in the following steps.

1. Start Up and Enter the Program in RAM
 - A. Turn power on. Push reset button on the main card. JBUG will respond with a “—”.
 - B. Type 0020 followed by the M key. This displays the current contents of location 0020.
 - C. Type 8E. This replaces the contents of 0020 with 8E which is the op-code for the first instruction, LDS.
 - D. Type G. This steps to the next location (0021) and displays the contents.
 - E. Type 00.
 - F. Type G.
 - G. Type next byte of op-code or operand (FF in this case).
 - H. Repeat steps F and G for remaining instructions.
 - I. Type E. Abort input function.

2. Verify That the Program Was Entered Correctly
 - A. Type 0020M. Location 20 will be displayed.
 - B. Type G. Next location will be displayed.
 - C. Repeat step B until done, visually verifying data entered in Step 1.
 - D. Type E.
3. Enter Data in Locations 10-14
 - A. Same as 1 except type 0010M to start the sequence. Any data may be entered; however, for purposes of this example 01, 02, 03, 04, 05 should be entered.
 - B. Type E.
4. Verify Data
 - A. Repeat step 2 except type 0010M to begin the sequence. Verify that the memory contains the values 01, 02, 03, 04, 05 in sequential order.
5. Run the Program
 - A. Type E to insure no other option is active.
 - B. Type 0020G. The program will run down to the "SWI" instruction at location 31 which will cause it to go to JBUG and show 0031 3F on the display.
6. Check the Answer
 - A. Type E.
 - B. Type 0015M. (The answer is stored in location 15). Note that it says 0A (decimal 10). The correct answer is 0F or decimal 15; therefore, there is a problem in the program as originally defined. The next steps should help isolate the problem and correct it.
7. Breakpoint and Register Display
 - A. It might be helpful to see what the program was doing each time it went through the loop. Therefore, set a breakpoint at the beginning of the loop, location 0029. To do this type E, then tye 0029V.
 - B. A breakpoint could also be set at location 002F to see the results. Type E. Type 002FV.
 - C. JBUG must be told where to begin, so type E and then 0020G. JBUG will run to the breakpoint and then display 0029 AB. At this point the program is suspended just before location 29 and is in JBUG. On detecting this breakpoint, JBUG automatically displays the PC and is in the register display mode.
 - D. Type G (Go to next register). The display should read 0010. This is the value of the X Register.
 - E. Type G. Display = 00 (A Register).

- F. Type G. Display = 04 (B Register).
 - G. Type G. Display = D0 (Condition Code Register).
 - H. Type G. Display = 00F8 (Stack pointer). Even though the program set the stack pointer to FF the action of the breakpoint used a software interrupt to store the registers on the stack, thus decrementing it by 7 locations. When JBUG returns to the user's program the stack will return to FF.
 - I. Type G. Display = 0029 (PC). The register display is circular and steps D through H could be repeated.
 - J. Type E. Abort the register display portion of the breakpoint. Type G to return to the example program and resume executing. Since the breakpoint at location 0029 is in a loop it will again be the next breakpoint and the display will contain 0029 AB. At this point the registers may be displayed again as per steps D through I. If this were done the A would be seen to contain the partial sum and the B would be decremented. The X Register would be one greater than previously.
 - K. Type E.
 - L. Type G (Proceed). Display will type 0029 AB. Once again the registers may be examined.
 - M. Type E.
 - N. Type G (Proceed). Same comment as L.
 - O. Type E.
 - P. Type G (Proceed). Display will now type 002F 97. The program has now successfully completed the loop four times and the A-Register contains the incorrect sum.
8. Correcting the Program
- A. From above it is evident that although the program was supposed to add five numbers, the loop was executed only four times. Therefore, the LDAB #4 instruction at location 24 and 25 should have initialized B to five. There are two approaches to fix the problem; one is temporary, the other is permanent. First the temporary one:
 - B. Type E.
 - C. Type V. Clears existing breakpoints.
 - D. Type 0026V. Set a breakpoint just after B register was loaded.
 - E. Type E.
 - F. Type 0020G. The program will execute up until 0026 and then go to JBUG, Display = 0026 CE.
 - G. Type G five times. This displays the current stack pointer (00F8). The B register contains the counter we wish to modify and is located at location SP + 2 (FA).

- H. Type E.
 - I. Type 00FAM. The display = 00FA 04.
 - J. Type 05. The display will change to 00FA 05.
 - K. Type E.
 - L. Type G. Proceed from user breakpoint down to the SWI instruction.
 - M. Type E.
 - N. Type 0015M. Display = 0015 0F. The program has now calculated the correct value for the addition of the five numbers 1-5. This verifies the fix but would be inconvenient to do each time the program was executed. A permanent change would be:
 - O. Type E, then type V. This clears all breakpoints.
 - P. Type 0025M. The display = 0025 04.
 - Q. Type 05. The display = 0025 05. This will now permanently change the LDAB #4 instruction to a LDAB #5 instruction.
 - R. Type E.
 - S. Type 0020G. Execute the program.
 - T. Type E.
 - U. Type 0015M. Display = 0015 0F, the expected answer; the program is permanently fixed.
9. Trace Through the Program
- A. Type E. In order to execute a trace, the program must first be stopped at a breakpoint. To trace from the beginning do:
 - B. Type V. This clears the existing breakpoints.
 - C. Type 0020V. This sets a breakpoint at the first instruction.
 - D. Type E.
 - E. Type 0020G (Go to user program). JBUG will immediately get the breakpoint and type 0020 8E.
 - F. Type N. The program will execute one instruction and display 0023 4F. At this point the user can either display the registers by depressing the G key or can continue to the next instruction. To continue:
 - G. Type N. Go to next instruction. Display register if desired.
 - H. Continue step G for as long as desired. Note: Do not try to trace after executing the SWI instruction; a restart will be necessary before continuing.
 - I. Type E. Clear trace mode.

10. Offset Calculation Including Register Modification

- A. Assume that the SWI instruction at location 31 is to be changed to a branch always (BRA) to location 20. This will cause the program to remain in an infinite loop (i.e., the program has no end and will run continuously unless interrupted by some outside stimuli). Type 0031 to open the memory location. The display = 0031 3F.
- B. The op-code for a BRA is a 20, so type 20. The display = 0031 20.
- C. The second byte of the BRA instruction should be the two's complement negative offset to location 20. Since doing this calculation in hex is tedious and error prone, a small unsophisticated (there was only a little ROM left) program that does offset calculation was provided at location E000 in the JBUG ROM.
- D. Type E.
- E. Type R, then type five G's. This will display the current stack pointer so that the registers can be located and set up.
- F. Type E.
- G. Type in hhhhM where hhhh = SP + 2. This displays the current B register.
- H. Type 00. This is the high byte of the destination address of the branch.
- I. Type G. This displays location SP + 3 which contains the A-register value.
- J. Type 20. This is the low byte of the destination address.
- K. Type G. Display high byte of X register.
- L. Type 00. Insert high byte of the branch op-code address.
- M. Type G. Display low byte of X register.
- N. Type 31. Insert low byte of the branch op-code address.
- O. Type E.
- P. Type E000G. When the program is completed it will return to JBUG via the SWI at location E013 and the PC will be displayed.
- Q. Type G twice. The A register is now displayed and contains ED which is the correct offset.
- R. Type G. The B register will contain an FF to indicate the branch was within range.
- S. Type E.
- T. Type 0032M.
- U. Type ED. Insert the branch offset.

11. Executing and Aborting

- A. Type E.
- B. Type 0020G. The program will begin executing and the JBUG prompt “—” will disappear since the program now contains an infinite loop.
- C. Type E. This aborts (Exits) the program and returns control to JBUG. The prompt has now returned.
- D. Type R. Display the PC and any other registers of interest.
- E. Type E.
- F. Type G. Program will again execute.
- G. Type E. Abort program and return to JBUG.
- H. Repeat F and G for as many times as you wish.

12. Punch Program to Cassette

- A. Rewind the cassette. Type E.
- B. Type A002M.
- C. Type 00. Enter high byte of beginning address.
- D. Type G.
- E. Type 20. Enter low byte of beginning address.
- F. Type G.
- G. Type 00. Enter high byte of ending address.
- H. Type G.
- I. Type 32. Enter low byte of ending address.
- J. Type E.
- K. Turn on cassette in Record mode.
- L. Type P. Wait for JBUG prompt to return (approximately 30 seconds).

13. Load Program from Cassette

- A. Turn off power. This will cause the program in memory to be lost. Turn power back on.
- B. Push the Reset button and get the JBUG prompt.
- C. Rewind cassette.
- D. Start cassette in playback mode.
- E. Type L. Wait for the JBUG prompt. Test the program by any of the options described above.

CHAPTER 2

HARDWARE DESCRIPTION

2-1 GENERAL DESCRIPTION

The MEK6800D2 Kit consists of two printed circuit board assemblies, the Microcomputer Module and the Keyboard/Display Module. The Keyboard/Display Module includes interface circuitry for using standard Audio Cassette tape recorders as an off-line magnetic storage medium. The Keyboard/Display Module provides an economical operator interface to the Microcomputer Module and is supplied as a separate board in order to facilitate using the Microcomputer Module with other terminals or as an end-item in the user's system development.

The Keyboard/Display Module is used in conjunction with a monitor program (called JBUG) supplied in an MCM6830 ROM to permit an operator to communicate with and control the Microcomputer Module. A detailed description of the available functions and commands is included in the Operating Procedures section (Section 1-4 of Chapter 1). The features are, in summary:

1. Examine and Change Memory
2. Display and Change MPU Registers
3. Go to User's Program
4. Trace One Instruction
5. Set and Clear up to Five Breakpoints
6. Proceed from Breakpoint
7. Abort from User's Program
8. Calculate Offset to Relative Branch Destination
9. Transfer Designated Memory Locations to Magnetic Tape
10. Load Memory Locations from Magnetic Tape

2-2 MEMORY ORGANIZATION

The general memory organization of the Kit is shown in Figure 1-1-2 of Chapter 1. The memory map is shown in tabular form in Table 2-2-1. In the M6800 system, memory location assignments are determined by the combinations of MPU address lines that are applied to the device chip select lines.

In Table 2-2-1, the signals designated as $\overline{\text{ROM}}$, $\overline{\text{PROM}}$, etc., are the outputs of an MC74155 One-of-Eight Decoder. The MC74155 decodes the MPU's VMA, A15, A14, and A13 lines. For example, when these lines are all high, corresponding to memory address $\$E000 (2^{15} + 2^{14} + 2^{13})$, the $\overline{\text{ROM}}$ output of the Decoder is low. This signal is applied to the chip select line $\overline{\text{CS1}}$ of the JBUG ROM, thus selecting this

SIGNALS DECODED																						
DEVICE	ADDRESSES	$\phi 2$	R/W	SYMBOL	VMA	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
ROM	E000-E3FF	1	1	$\overline{ROM}$ =	1	1	1	1				x	x	x	x	x	x	x	x	x	x	
PROM	C000-C3FF			$\overline{PROM}$ =	1	1	1	0			+	x	x	x	x	x	x	x	x	x	x	
RAM (Stack)	A000-A07F	1	x	$\overline{STACK}$ =	1	1	0	1	0				0	0	x	x	x	x	x	x	x	
PIA	8020-8023	1	x	$\overline{I/O}$ =	1	1	0	0								1		0*	0*	x	x	
ACIA	8008-8009	1	x	$\overline{I/O}$ =	1	1	0	0								0*		1	0*		x	
PIA	8004-8007	1	x	$\overline{I/O}$ =	1	1	0	0								0*		0*	1	x	x	
PROM	6000-7FFF			$\overline{6/7}$ =	1	0	1	1			+	x	x	x	x	x	x	x	x	x	x	
USER	4000-5FFF			$\overline{4/5}$ =	1	0	1	0														
USER	2000-3FFF			$\overline{2/3}$ =	1	0	0	1														
RAM (User)	0000-007F	1	x	$\overline{RAM}$ =	1	0	0	0				0	0	0	x	x	x	x	x	x	x	
RAM (User)	0080-00FF	1	x	$\overline{RAM}$ =	1	0	0	0				0	0	1	x	x	x	x	x	x	x	
RAM (User)	0100-017F	1	x	$\overline{RAM}$ =	1	0	0	0				0	1	0	x	x	x	x	x	x	x	
RAM (User)	0180-01FF	1	x	$\overline{RAM}$ =	1	0	0	0				0	1	1	x	x	x	x	x	x	x	

x = Decoded by the device addressed

* = Required but not decoded by the device addressed

+

TABLE 2-2-1: MEK6800D2 Evaluation Kit II Address Map

device whenever the MPU outputs addresses in the range of \$E000 to \$EFFF. The particular locations within the ROM are selected by applying MPU address lines A0 thru A9 to the ROM address inputs. The JBUG ROM is located at the highest addresses in the kit's memory field. Note that A12 from the MPU is not applied to this ROM so it will also be selected when the MPU outputs its Restart and Interrupt Vector addresses, \$FFF8 — \$FFFF. Start-up and interrupt capability is obtained by placing the appropriate interrupt vector addresses in locations \$EEE8 — \$EFFF of the monitor program.

Additional addresses are decoded for the optional ROMs that can be added for user-generated programs. The Microcomputer Module is layed out to accept either two MCM68708 1024 x 8 bit Electrically Programmable Read Only Memories (EPROM) or two MCM7641 TTL 512 x 8 bit Programmable Read Only Memories. The PROMs are more economical but cannot be erased like the EPROM. Two MCM68316 2048 x 8 bit ROMs can also be used in the PROM locations. In this case, MPU address line A10 is applied to the MCM68316 for decoding the additional 1024 bytes. Jumpers on the PCB are provided for selecting the desired combination of ROM (see note 6 on the schematic diagram of Figure A3-a).

The MC6810 (128 x 8) RAM occupying memory locations \$A000 — \$A07F is used by the MPU for temporary storage of its internal registers during interrupts and subroutines and is selected by the signal $\overline{STACK}$. The MPU also uses this area for storage of flags and temporary data used by the JBUG monitor. This organization allows a clean separation between monitor requirements and user RAM. The system assigns, via the $\overline{RAM}$ signal, the four user RAMs to the bottom of memory in locations \$0000 — \$01FF (first 512 bytes). This RAM is useful for small user programs or for scratchpad memory in the MPU's direct addressing range for larger user programs. To prevent contention with these RAMs, expanded systems should avoid these memory

locations; however, the board is easily modified (see Section 2-8 on system expansion) to accommodate external memory in this range.

The two signals $\overline{2/3}$ and $\overline{4/5}$ are brought to the edge connector and may be used to select two external 8K-byte blocks of memory. The $\overline{2/3}$ line decodes the second 8K-byte block (\$2000 — \$3FFF) of the memory space; $\overline{4/5}$ decodes the next 8K locations (\$4000 — \$5FFF).

2-3 INPUT/OUTPUT DEVICES

Three I/O devices are provided with the Kit and are selected by the $\overline{I/O}$ signal. The PIA at addresses \$8004 — \$8007 is provided for user specified peripheral devices. Its input/output lines are brought out of the J1 edge connector. A wire-wrap area is provided for any buffering or interface circuitry that might be required. In normal kit operation, the PIA at addresses \$8020 — \$8023 is used to interface the Keyboard/Display to the MPU. If a terminal and the MINIBug monitor are used, this PIA is also available (via the J2 edge connector) for user specified I/O. The ACIA at memory locations \$8008 — \$8009 is used to interface with the Audio Cassette circuitry on the Keyboard/Display Module, but can alternatively be used to interface to serial RS-232 or TTY type terminals (with the MINIBug monitor) if desired. Note that the address lines A2, A3, and A5 are applied to the chip select lines of the \$8004 PIA, the ACIA, and the \$8020 PIA, respectively. This insures the selection of only one of the three I/O devices when the $\overline{I/O}$ signal is active. Note also that connecting the A2, A3, and A5 address lines to the PIA and ACIA chip select lines will cause a wider range of addresses than is required to be selected. For example, when the $\overline{I/O}$ signal is low (A15, A14, A13 = 110) and A5 is high, any address in the range \$8020 — \$802F may be present on the bus, depending on the states of A0 — A3. The \$8020 PIA does not decode the A2 or A3 lines; therefore, addresses in the range \$8024 — \$802F will also select this PIA. However, it is not necessary to use additional decoding if the use of these addresses is avoided in the user program.

2-4 SYSTEM CLOCK

The Kit uses a 614.4 kHz MC6871B system clock. The frequency was selected in order to provide a simple means of obtaining a 4800 Hz reference frequency used by the 300 baud serial data rate tape cassette circuitry. The 4800 Hz signal is obtained by dividing the MC6871B's $2f_0$ output (1.2288 MHz) by 256 with an MC14040 counter. The 4800 Hz signal is applied to the cassette interface circuitry, along with the ACIA outputs, via the J2 edge connector.

2-5 KEYBOARD/DISPLAY

The Keyboard/Display Module is provided as a separate printed circuit board in order to facilitate the use of other terminals and to make the U21 PIA readily available for eventual expansion of the system. The Keyboard/Display Module connects to the Microcomputer Module via a ribbon cable and connector provided with the Kit. A scanning technique is used on both the display and the keyboard in order to minimize system cost. Since operation of this circuitry is intimately related to the control program, refer to the software discussion (Section 3-4) and the assembly listing, as well as the schematic diagram of Figure A3-b with the following description.

The scanning procedure uses lines PB0 — PB5 of the PIA, corresponding to SCNREG in the JBUG assembly listing. The digit patterns to be displayed are put out on lines PA0 — PA6 and are designated as DISREG in the listing. The JBUG monitor program alternates between refreshing the display and checking for a key closure in the following manner.

The OUTDS subroutine places the digit pattern for the left-most display on PA0 — PA6 and then sets PB5 high, causing that digit to be lighted. During this time, PB4 — PB0 are low, thus the other digits are off. This digit of the display is held on for approximately 1.0 ms, after which the pattern for the second digit is put on lines PA0 — PA6. PB5 is switched low, and PB4 is taken high to select the second digit. This sequence continues until the right-most digit has been selected, at which time the program goes to the KEYDC subroutine to check for key closures. The blanking pattern (\$FF) is placed on PA0 — PA6 to blank the display so that lines PB0 — PB5 can be used to interrogate and decode the keyboard. Following the keyboard check, operation returns to the display sequence. The refresh rate is fast enough that the displays appear to be on continuously.

An MC14539 CMOS One-of-Four Data Selector (U10) is used to sequentially select each column in the keypad matrix and route it to PA7 for testing by the monitor program. The address data for selecting each column is output to the Data Selector on lines PB6 and PB7. Refer to the description of the monitor program in Section 3-4 for details of the keyboard decoding technique. Note that CB1, a PIA interrupt input, is directly connected to column 2. This allows the E key to be used for generating an $\overline{\text{NMI}}$ interrupt for escaping from “blown” user programs. The MC75452 buffers serves to increase the PIA’s drive capability.

2-6 TRACE (EXECUTE SINGLE INSTRUCTION)

A hardware trace function is provided that permits a user’s program to be executed one instruction at a time. Results of the execution, including MPU Register contents, can be examined between each Trace command. The Trace function will operate on programs in either RAM or ROM and is useful as a debugging aid. The circuitry consists of an MC8316 Counter and two MC7479 D-flip-flops connected as shown in Figure 2-6-1. Refer to this figure also for the associated timing waveforms.

When a Trace command occurs, the system is normally in the Register display mode from either a previous Trace or having run to a Breakpoint. Thus, the user’s Register values are stacked and the monitor program is alternating between refreshing the displays and checking for new key closures. The user Program Counter value saved on the stack is pointing to the next user instruction to be executed. Invoking a Trace command at this point causes the MPU to start the Trace Counter (via CA2 of the Keyboard/Display PIA) and then execute a Return from Interrupt (RTI) instruction. This causes the MPU to reload its Registers from the stack and begin executing the next user instruction. In the meantime the Trace counter is counting machine cycles. The eleventh cycle after the counter is started will be a fetch of the op-code for the next user instruction (RTI takes ten cycles to execute). The Trace circuitry detects the eleventh cycle and generates a low going $\overline{\text{NMI}}$ signal. Since the shortest instruction is at least two cycles long, $\overline{\text{NMI}}$ will always be low at the end of the first instruction and will cause a return to the JBUG monitor program via an $\overline{\text{NMI}}$ interrupt. The $\overline{\text{NMI}}$ service routine sets CA2 back high, resetting the counter in readiness for another command. The $\overline{\text{NMI}}$ service routine is described in Section 3-8 in greater detail. From the user’s point of view, closure of the N (Trace) key causes the system to execute one instruction and then stop so that the results can be examined.

2-7 AUDIO CASSETTE INTERFACE

Circuitry for interfacing an ACIA to an audio cassette recorder/player is included on the Keyboard/Display Module. This circuitry enables the user to store and retrieve data on ordinary audio cassettes at a 300

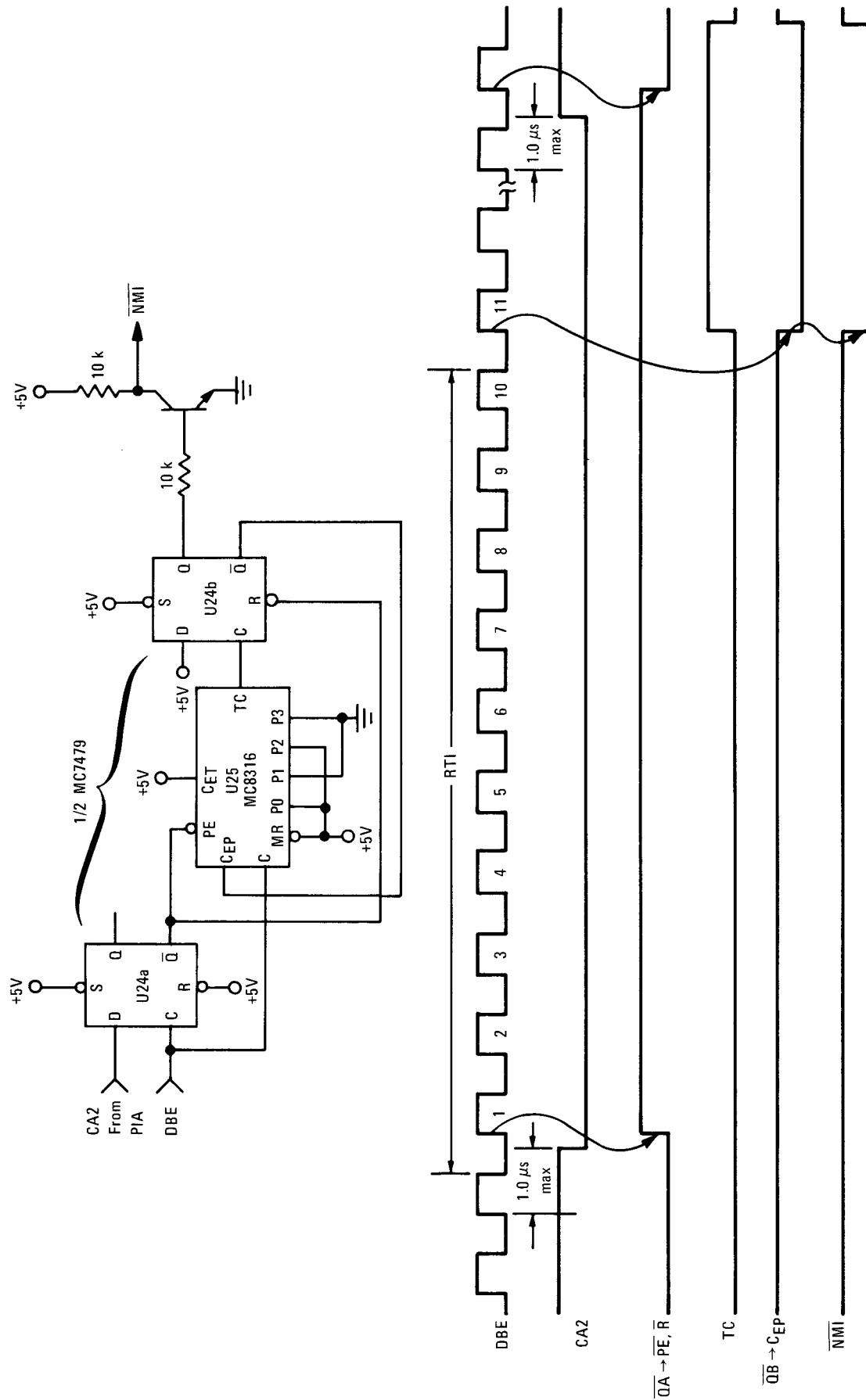


FIGURE 2-6-1. Trace Circuitry and Timing Waveforms

baud (30 characters per second) serial clock rate. Data is stored on the tape using the “Kansas City Standard” recording format, so-called due to its formulation during a symposium sponsored by *BYTE* Magazine in Kansas City, Missouri in November, 1975. The format is designed to eliminate errors due to audio system speed variations⁵ and has the following characteristics:

1. A Mark (logical one)⁶ is recorded as eight cycles of a 2400 Hz signal.
2. A Space (logical zero) is recorded as four cycles of a 1200 Hz signal.
3. A recorded character consists of a Space as a start bit, eight data bits, and two or more Marks as stop bits.
4. The interval between characters consists of an unspecified amount of time at the Mark frequency.
5. In the data character, the least significant bit (LSB) is transmitted first and the most significant bit (MSB) is transmitted last.
6. The data is organized in blocks of arbitrary and optionally variable length preceded by at least five seconds of Marks.
7. Meaningful data must not be recorded on the first 30 seconds of tape following the clear leader.

A control program in JBUG causes this format to be followed and incorporates the following additional characteristics:

1. At the beginning of tape (BOT), the ASCII character for the letter “B” is recorded following 1024 Marks (approximately 30 seconds).
2. The “B” is followed by one byte containing the block length (up to 256 bytes in a particular block).
3. The next two bytes recorded contain the starting address in memory from which the data is coming.
4. Up to 256 bytes of data are then recorded and followed by 25 marks and the ASCII character for the letter “G”.

The control program uses the additional features to insure that the Punch and Dump functions are performed in an orderly manner (see the explanation in Section 3-7 for additional information).

The cassette interface circuit diagram of Figure 2-7-1 serves as an aid to understanding the following description of the Punch and Load operations. The Punch (transfer of data from the Microcomputer Module’s memory to tape) and Load (transfer from tape to memory) commands are accomplished by a combination of the control program, the MC6850 Asynchronous Interface Adapter (ACIA), and the cassette interface circuitry.

The ACIA is, in effect, a bus-oriented, universal, asynchronous receiver/transmitter (UART). In the transmit mode (Punch), it accepts parallel 8-bit data from the MPU bus, adds the formatting start bit and stop bit, and then converts the data to a serial binary stream (Tx Data in Figure 2-7-1). The desired format is established by instructions from the MPU as it executes the Punch command. In the receive mode (Load), the ACIA accepts an incoming serial data stream (Rx Data) and a sampling clock (Rx Clk). It strips off the start/stop bits and passes each incoming byte to the MPU for transfer to memory, again under control of the MPU as the

⁵The circuitry provided with the kit will accommodate speed variation of approximately $\pm 25\%$.

⁶Logical ones and zeros will be alternatively referred to as Marks and Spaces, respectively, in accordance with serial data transmission conventions.

program executes. The ACIA's Request-to-Send, $\overline{\text{RTS}}$, acts as a gating signal to switch the interface circuitry between the Punch and Load modes. The reference documents may also be referred to for additional details on the ACIA's characteristics.

Timing waveforms corresponding to the appropriate signals in Figure 2-7-1 are provided as Figures 2-7-2, 2-7-3, and 2-7-4 as an aid to study of the cassette interface circuitry.

During a Punch operation the interface circuitry operates on the serial data to convert each logical one (Mark) to an 8-cycle burst of 2400 Hz signal and each logical zero (Space) to a 4-cycle burst of 1200 Hz signal which is then recorded on tape.

The circuitry reverses this procedure during a Load operation; it decodes the incoming frequency-modulated signal in order to recover the binary data and a sampling clock.

In Figure 2-7-1, the MC14053 Multiplexer/Demultiplexer, U20, (Data Router, for simplicity) is used to steer signals to their required points during both Load and Punch operations. For instance, during Punch, B and C are high while A is derived from the binary data on Tx Data. For this combination of control signals Y is connected to Y1 (because B is high); thus the 4800 Hz Tx Clk signal from the Microcomputer Module is applied to the clock input of the MC14024 Counter, U19. Also, because C is high, Z is connected to Z1, but this signal is not used during Punch. The 2400 Hz and 1200 Hz signals are obtained by selecting either the $\div 2$ (Q1) or the $\div 4$ (Q2) outputs of the Counter as it is clocked at 4800 Hz.

The signals at X0 and X1 are 1200 and 2400 Hz sine waves obtained via the bandpass filters of U16a and U16d. One or the other of these signals (depending on the Tx Data logic level at A) will be level shifted, attenuated, and applied to the microphone output terminals.

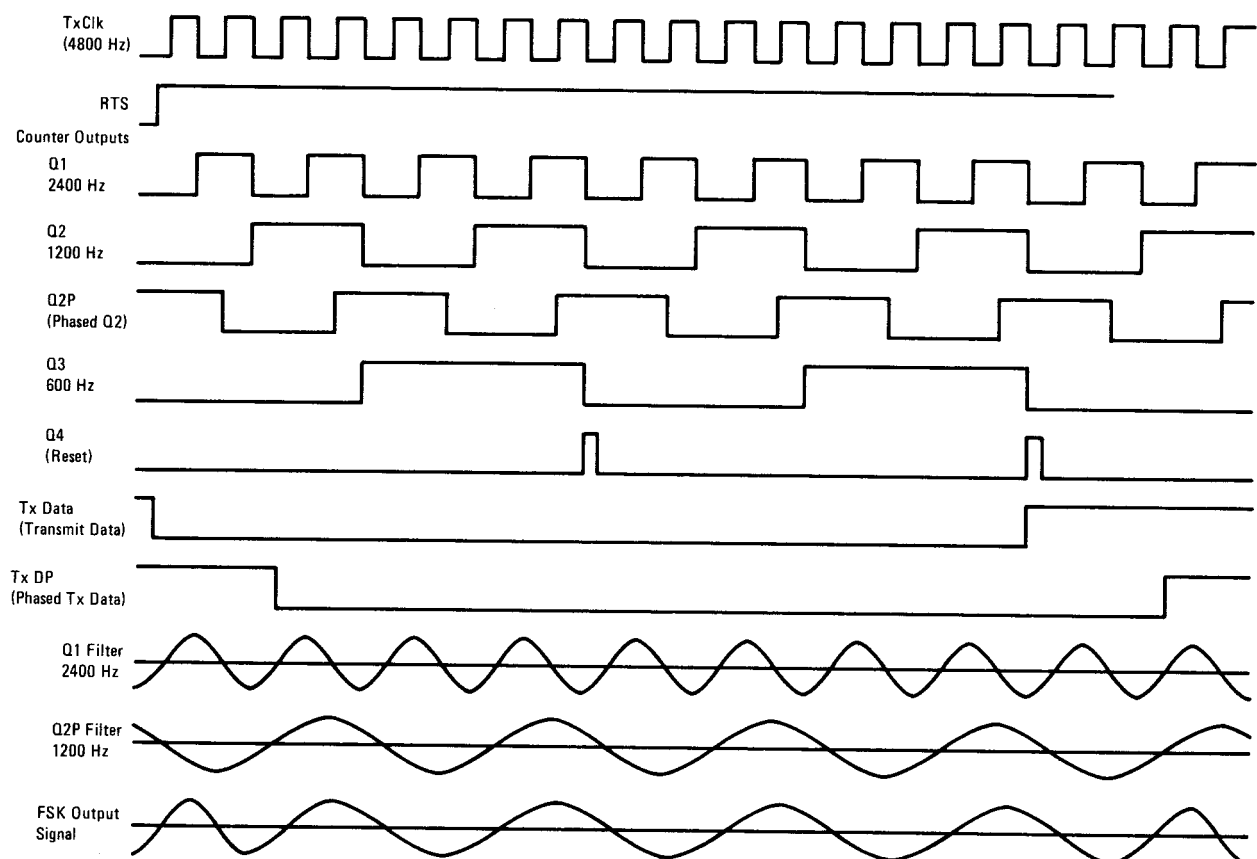


FIGURE 2-7-2. Transmit Waveforms

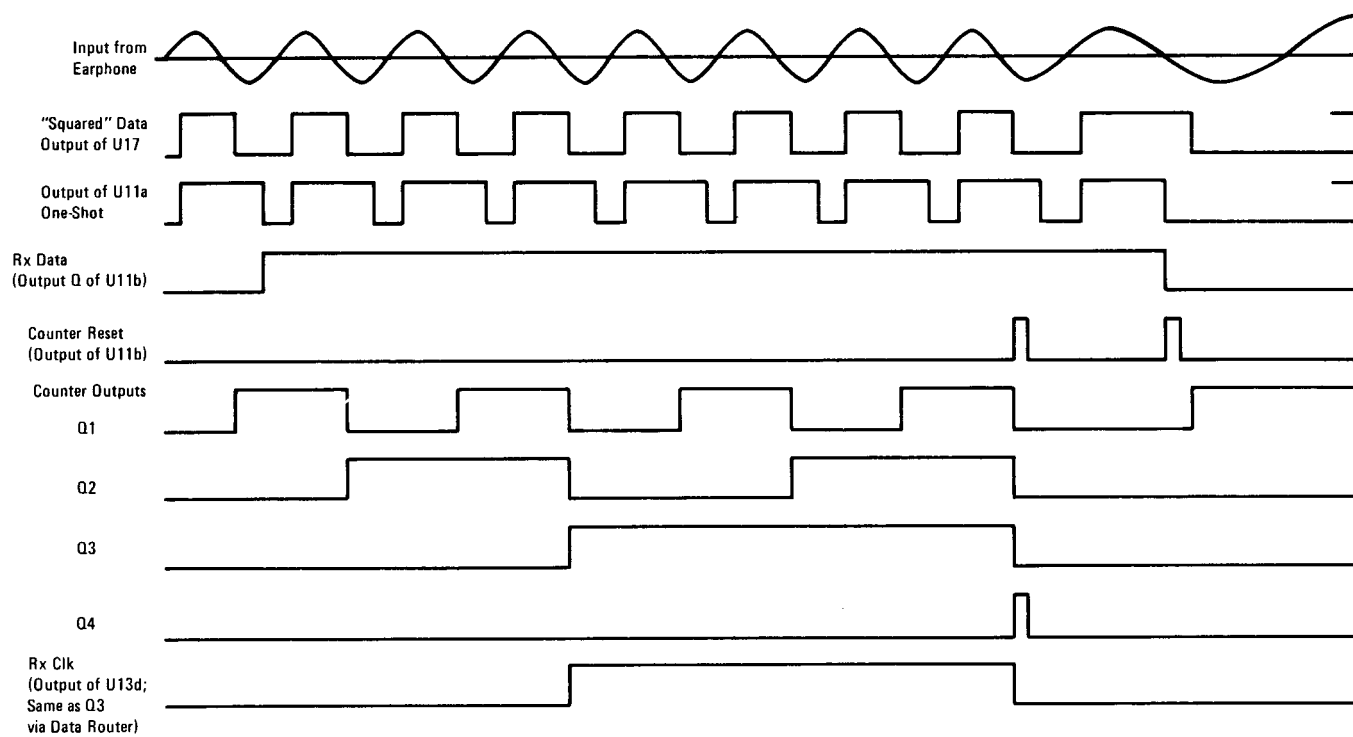


FIGURE 2-7-3. Receive Waveforms, Space-to-Mark Transition

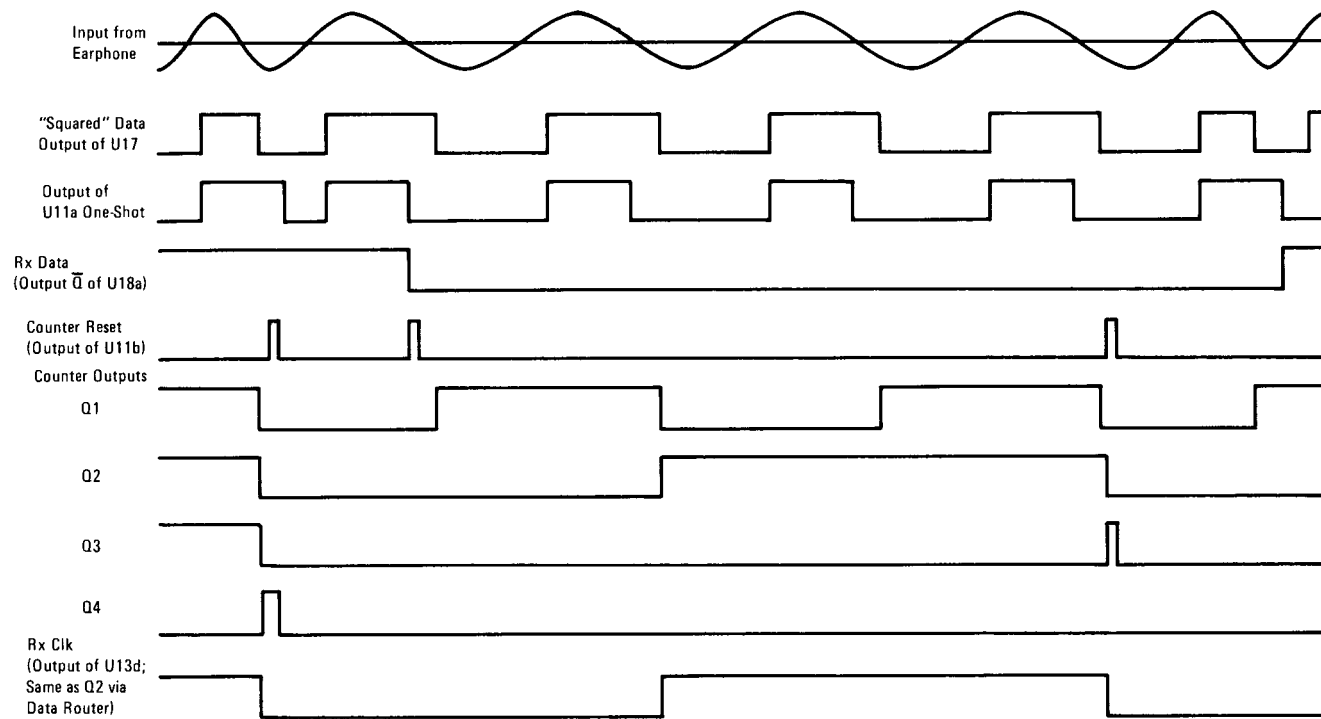


FIGURE 2-7-4. Receive Waveforms, Mark-to-Space Transition

Note that the 1200 Hz square wave is obtained from the output of U12a rather than the Q2 output of the MC14024. This, together with the gating of U13 and the delay associated with U12b, insures that switching of output frequencies will occur only when the outputs of U16a and U16d are at essentially the same voltage. (Refer to the timing diagram of Figure 2-7-2.)

During a Load operation, the incoming signal from the cassette earphone is filtered, amplified and squared by the U17 Line Receiver. (U17 is connected as a Schmitt trigger to reduce noise problems.) This results in a signal, at digital levels, that varies between 2400 Hz and 1200 Hz according to the one-zero pattern that was recorded on the tape. This frequency-modulated signal is then converted to logical ones and zeros by the pulse width discriminator formed by the U11a MC14538 Monostable Multivibrator (or One-Shot) and the U18a type D flip-flop. Incoming signals less than 1800 Hz are decoded as zeros; frequencies higher than 1800 Hz are decoded as ones. The Received Data will be present at the $\overline{Q}$ output of U18a.

The required Rx Clk signal, a positive transition at the mid-point of each bit-time and a negative transition at the end of each bit-time, is generated as follows:

During Load the digital level 2400/1200 Hz signal, instead of the 4800 Hz Tx Clk signal, is steered to the Counter clock input. The Counter's $\div 8$ (Q3) and $\div 16$ (Q4) outputs are connected to the inputs of U14b and U14a, respectively. The control inputs of U14a and b are connected to Received Data and applied to the Set input of U18b. The Output of U18b triggers the Counter Reset one-shot, U11b. Hence, either the $\div 8$ or $\div 16$ Counter output is steered back (via X) as a reset, depending on whether the data is a zero or a one, respectively. The Counter is also reset by every Mark-to-Space transition via the U11b One-Shot. The Counter's $\div 4$ and $\div 8$ outputs are connected to Z0 and Z1, respectively. These connections combined with the reset signals result in a positive transition at the Z output of the Data Router after either four cycles of 2400 Hz or two cycles of 1200 Hz. Thus, the Rx Clk (Z gated by $\overline{RTS}$) has a positive transition in the middle of each bit-time and a negative transition at the end of each bit-time.

2-8 KIT EXPANSION

Provision is made for buffering circuitry to allow the Microcomputer Module to be implemented into a larger system. The buffers and pinouts selected on the bottom edge connector are compatible with the EXORciser so its I/O and Memory Modules can be used with this kit. The direction of data flow across the data bus buffers is controlled by the MC7430 NAND gate, U7. This decoding provides for data flow off the board to the external system when there is a Memory Read Cycle at an address that is not decoded by the devices on the Microcomputer Module itself. Note that the signal $\overline{RAM}$ decodes the lowest 8K bytes of memory which are reserved for on-board memory (MCM6810's). Should the user want to assign the lowest 8K of memory addresses to off-board memory, the following changes are required:

Remove the MCM6810's decoding addresses 0000, 0080, 0010 and 0180; remove the signal $\overline{RAM}$ from pin 4 of the MC7430 and tie pin 4 to +5 V. The signal provided at the bus connector called $\overline{RAM}$ can be used on outside memory to indicate an MPU access to an address in the bottom 8K bytes of memory which now resides off the module.

Provision has been made for using a zener diode (1N4733) to generate a -5 V supply for the 2708 PROMs (if they are used) from -12 V in case this kit is operated in an EXORciser-type system which does not have -5 V available. Should -5 V be available, the zener diode and associated 68 ohm resistor can be omitted and the -5 V brought in through the bus connector.

CHAPTER 3

SOFTWARE DESCRIPTION (JBUG MONITOR)

3-1 GENERAL DESCRIPTION

The control and diagnostic capability of the MEK6800D2 Kit is provided by the JBUG monitor program resident in the MCM6830 1K x 8 bit ROM supplied with the Kit. The characteristics of this program are described in the following sections. An assembly listing of JBUG is included (Appendix 1) and may also be referred to in studying the flow of the program.

Several RAM locations are used for temporary data storage and as flags by the monitor in communicating between the various routines. Some of the more significant ones are described below and are referred to in the description of JBUG.

SP (\$A008)	A RAM location in which the user's Stack Pointer is saved whenever the monitor resumes control. The user's Stack Pointer is required for locating user Registers on the stack and to restore these Register when returning to the user program.
DISBUF (\$A00C)	Eight RAM locations used as a buffer to hold the current values being displayed. In the first six locations, the high order 4 bits of each location represent the display digit-count while the low order 4 bits contain the value that is to be displayed on that digit. For example, the high order 4 bits of the sixth location in DISBUF identify the right-most display. The last two locations in DISBUF are used for temporary storage of data that is input from the keypad during a Memory Change function.
DIGIN4 (\$A014)	A flag that is set to one (LSB) when at least four hex digits have been entered from the keyboard (as in Memory Examine)
DIGIN8 (A015)	A flag that is set to one (LSB) when six hex digits have been entered from the keyboard (as in Memory Change)
MFLAG (\$A016)	A flag that is set to one (LSB) when the M key is depressed to invoke the Memory Examine Mode.
RFLAG (\$A017)	A flag that is set to one (LSB) when the R key is depressed to invoke the Register Display Mode.
NFLAG (\$A018)	A flag that is set to one (LSB) when the N key is depressed to invoke the Trace Mode.
VFLAG (\$A01D)	A flag that is set to the number of breakpoints (up to five) that have been set.
XKEYBF (\$A01A)	A pointer to the next empty location in DISBUF where the next hex key entry will be stored.

The flow of JBUG is straightforward and is shown in Figure 3-1-1. After release of the RESET button, the monitor goes through an initialization sequence in which the stack pointer is initialized to \$A078,

the PIA for the Keyboard and Display is configured, the flags which communicate between routines are cleared and a dash (-) is placed in the first location of DISBUF to be displayed on the lefthand digit as a prompt to indicate that the MPU is executing the JBUG monitor. After initialization the display is scanned; this involves displaying the contents of DISBUF (first six locations). The display scan takes about 6 ms (6 digits at 1.0 ms per digit) after which the Keyboard is scanned and decoded (KEYDC). A test is made to see if any key is depressed and if none is found the program returns to OUTDS. If a key is found to be depressed, a decoding process takes place to debounce the key and to determine which key is depressed. If the key is a hex key (0-F) then its value is placed in the next open location in DISBUF. If the key is one of the command functions, that command is decoded and executed before returning to the display routine OUTDS. As shown in Figure 3-1-1, the basic background program flow alternates between refreshing the display and checking for key closures.

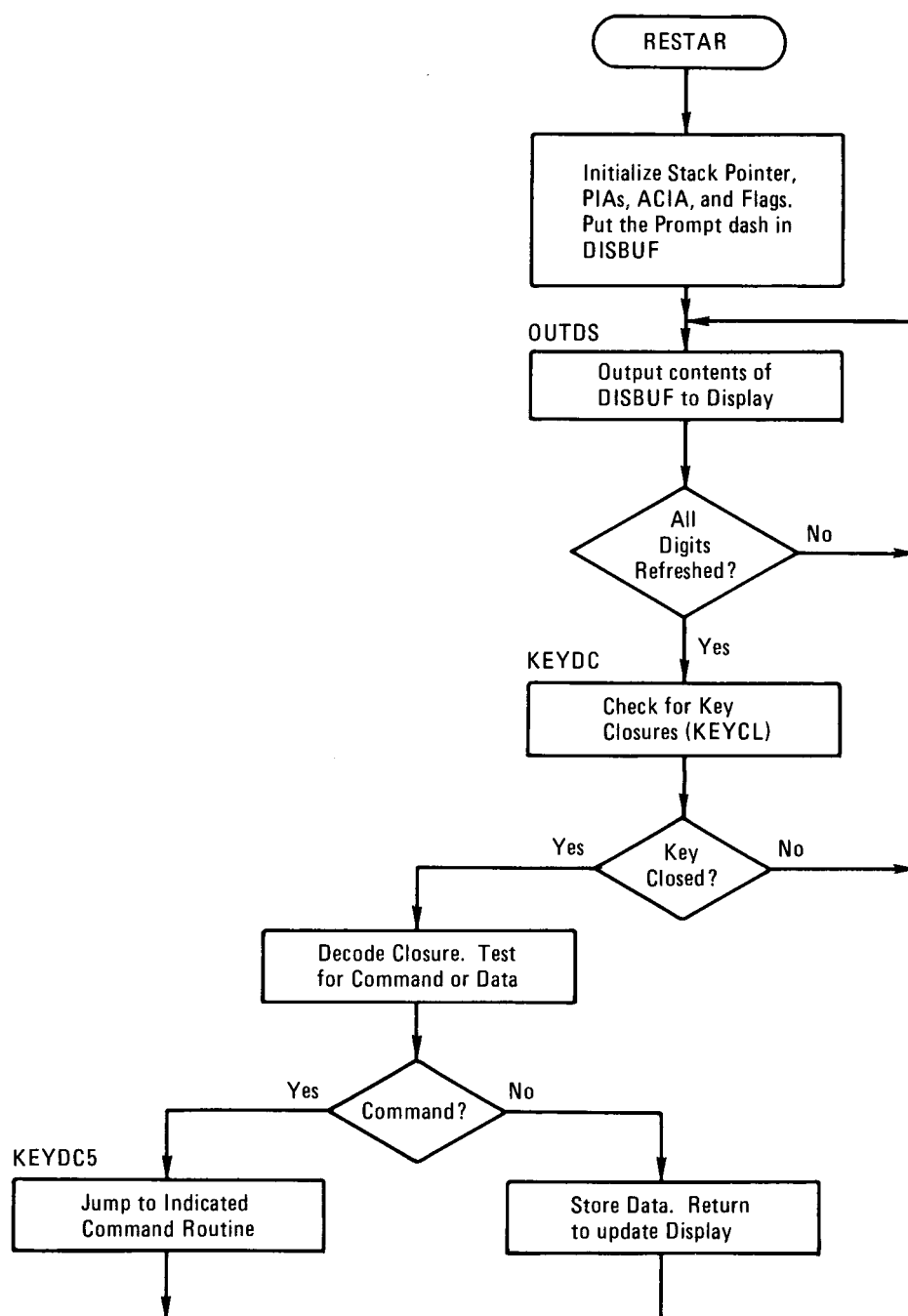


FIGURE 3-1-1. Overall Program Flow for JBUG Monitor

3-2 RESTART/INITIALIZATION ROUTINE

When the RESET push button is released, the MPU outputs addresses \$FFFE and \$FFFF in order to bring in the starting address of the restart routine. Because this system does not require full address decoding (see Section 2-2), the top two locations of the JBUG ROM (\$E3FE and \$E3FF) respond with \$E08D, the beginning address of the restart routine, RESTAR. RESTAR first initializes the Stack Pointer to \$A078 and then sets the $\overline{\text{NMI}}$ interrupt pointer to \$E14E. The $\overline{\text{NMI}}$ interrupt pointer is placed in RAM so that the user can change it and force $\overline{\text{NMI}}$ interrupts to do something other than go to the JBUG monitor (if this is done all diagnostic capability of JBUG will be lost). The Keyboard/Display PIA, U21, is then configured to match the hardware connections shown in the Keyboard/Display Module Schematic Diagram, Figure A3-b. The flags are cleared and a code to blank the display (\$17) is stored in all locations of DISBUF. A dash (-) is written in the first location of DISBUF to indicate that the MPU is executing the monitor program. Flow then branches to the OUTDS routine whose function is to move the contents of the DISBUF out to the LED displays.

3-3 DISPLAY ROUTINE

The display routine, OUTDS, is detailed in the flow chart of Figure 3-3-1 and begins at line 260 (address \$E0FE) of the assembly listing. The first value in DISBUF is loaded into Accumulator A (Acc.A). The

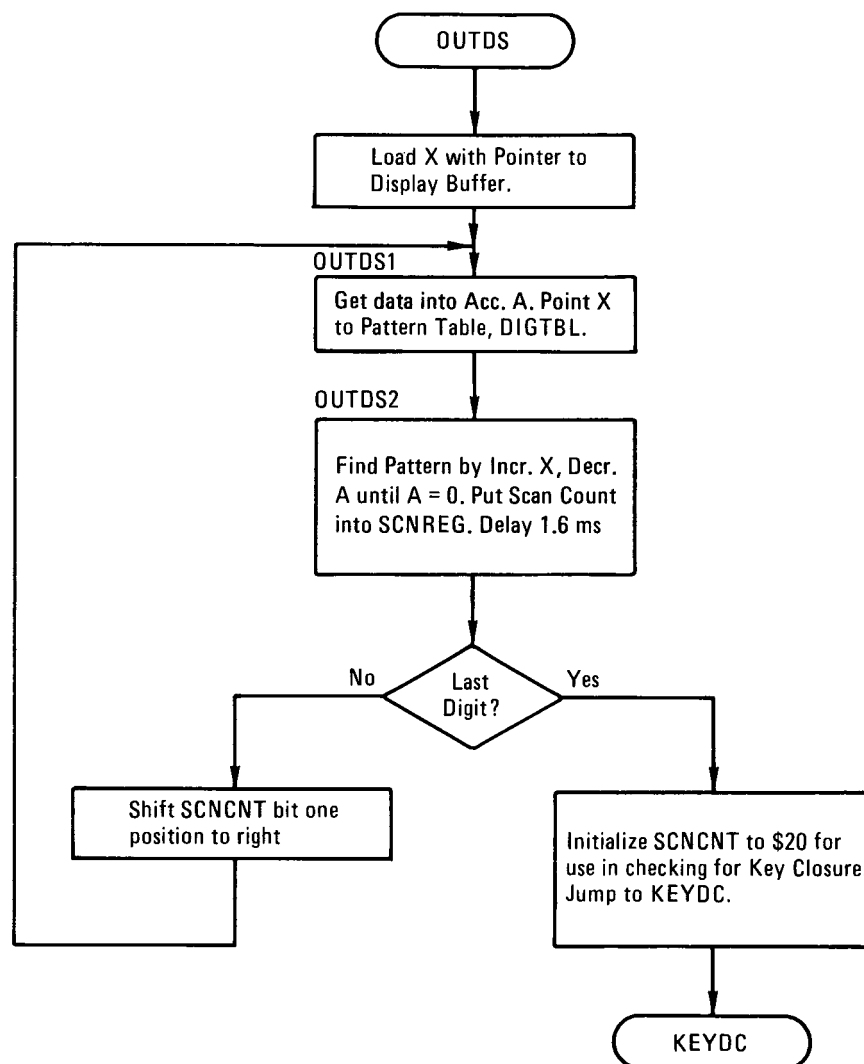


FIGURE 3-3-1. Program Flow for Output Display Routine

Index Register is then pointed to the beginning of DIGTBL, a table which has the correct bit patterns for the character set to be displayed. The Index Register, X, is then moved to the table location corresponding to the required pattern by decrementing Acc.A while X is incremented until Acc.A = 0. This pattern is then put out to DISREG (the anodes of the seven segment display) as the first digit of display is selected by SCNREG (the cathodes of the display).

This process is repeated for all six positions by moving a “one” through SCNREG as each position’s data appears in DISREG. In this manner, the data in the first six locations of DISBUF are output to their respective display positions and turned on for about 1.0 ms each (using the DLY1 delay loop. After all six positions have been scanned, the variable SCNCNT is reset to \$20 (corresponding to the left-most display) in readiness for use during the next refresh scan cycle.

3-4 **KEYBOARD SCAN AND DECODE ROUTINE**

Following each display refresh cycle, the monitor jumps to KEYDC (line 302, address \$E14E, flow charts in Figures 3-4-1 and 3-4-2), the routine for scanning and decoding the Keyboard. The Keyboard is first tested by subroutine KEYCL to determine if a key has been depressed. The display is blanked by storing \$FF to avoid flicker while the SCNREG lines are being used to interrogate the keyboard. Storing \$3F to SCNREG applies logical zeros to the rows of the keyboard matrix. KEYCL1 then tests each column in sequence to determine if a key is closed. (A depressed key will couple the zero on its row through to PA7 when tested.) The KEYCL routine returns to the caller, KEYDC, with status information in Acc.A. If no key was closed, Acc.A will contain \$00 and the program will branch back to OUTDS for a display refresh. If a key was closed, the program branches to a 20 ms delay (DLY20) to allow time for key debounce. KEYDC1 then scans the keyboard one row at a time using KEYCL to scan the columns looking for the closed key.

An exit back to OUTDS occurs (line 312) if the last row has been scanned without finding a closure. If there was a closure, KEYDC2 compares the value returned in Acc.A with codes in table KEYTBL to determine the key value. The KEYTBL values are related to the column and row position for each key. Each key is represented by a value in the range 0-23 with the first 16 values representing hex numbers. Once the key value has been found, the program enters the KEYDC4 routine to wait for the key to be released. After release is detected, the program again delays for 20 ms to provide time for debounce. Line 327 begins decoding the key value into either hex or command. Hex keys are entered into DISBUF at the location pointed to by XKEYBF and then tested to see if four digits have been entered yet. If four digits have been entered, DIGIN4 is set to enable further operations such as Memory Examine. Command key values are routed to KEYDC5, a jump table resulting in a branch to one of eight locations depending on the command key depressed. The following action is taken on each command key:

- P-KEYDC8 The display buffer, DISBUF, is cleared and the program jumps to subroutine PNCH. Upon return from the punch routine, a dash (-) is written to DISBUF (to inform the operator that the punch has been accomplished) and the program jumps to OUTDS.
- L-KEYDC9 The display buffer (DISBUF) is cleared and the subroutine LOAD is called. After the data has been loaded from tape the monitor dash is written into DISBUF and the OUTDS routine called to inform the operator that the load is complete.

- N-KEYDCA** Breakpoints, if any, are removed by clearing VFLAG. The NFLAG is set (LSB) to identify the TRACE mode and CA2 of the Keyboard/Display PIA is switched low to start the trace counter. An RTI instruction is then executed to reload the stack into the MPU and go on with the next user instruction.
- V-KEYDCB** The DIGIN4 flag is tested to determine if it is in the clear or set breakpoint mode. If four digits have been entered, the DIGIN4 flag will be set and the program will call the set breakpoint (SETBR) subroutine and then go to the OUTDS routine. If the DIGIN4 flag is clear, then V was a clear breakpoint command and the VFLAG is cleared thus clearing any breakpoints which may have been set.
- M-KEYDCC** The MFLAG is set to indicate that the Memory mode has been selected. The DIGIN4 flag is tested to make sure a full memory address has been entered. If four digits have been entered, the Memory Display Subroutine (MDIS) is called; otherwise the program goes back to OUTDS.
- E-KEYDCD** Causes the MPU to clear the DISBUF locations, write the monitor prompt dash to DISBUF, and then branch to the display refresh routine. When a user program is in progress the E key generates an $\overline{\text{NMI}}$ interrupt, providing an abort function.
- R-KEYDCE** The RFLAG is incremented to designate the Register Display mode and then the Register Display subroutine is called.
- G-KEYDCF** The G key performs one of three functions depending on the current mode of operation. If the monitor program is in the Memory Examine or Register Display mode, the G command causes the next location to be displayed. If neither of these modes is in effect, G can be used to either go to a user program or proceed from a breakpoint. These operations are described in greater detail in the next paragraph.

When a G command is decoded the jump table directs program flow to KEYDCF (line 431, address \$E20E) and the MFLAG is tested to determine if the current G key closure is a command to go to the next memory location. If MFLAG is set, the Memory Increment (MINC) subroutine is called and will be followed by the Memory Display (MDISO) subroutine. If MFLAG is clear, the RFLAG is tested to determine if this G closure meant go to the next Register location. If RFLAG is set, the subroutine to display next Register (REGST1) is called.

If neither MFLAG or RFLAG is set, the G closure is interpreted as a Go to User Program command, from either a specific address or from the location indicated by the current value of the Program Counter saved on the stack. The DIGIN4 flag is tested (line 436) to determine if a new starting address has been entered. If DIGIN4 is set, the program replaces the stacked value of the Program Counter with the new Go address is saved in the first four locations of the Display Buffer, DISBUF. After checking to see if there are any breakpoints to install, the MPU executes a Return from Interrupt (RTI) to the user program.

If DIGIN4 is clear, a proceed from current Program Counter mode is indicated. In this case, the GETXB routine is called to determine if any breakpoints have been set. If no breakpoints are in effect, keyboard interrupts are enabled (TGC, line 464) and the MPU executes an RTI back to the user's program. If breakpoints are indicated, the trace routine (TRACE, line 384) is called to step one instruction. On receiving the NMI interrupt caused by the trace, the NMI routine (NONMSK, line 91) checks to see if both trace and breakpoint

flags are set. If set, JBUG then installs the breakpoints (TGC, line 464) and returns to the user's program. This procedure is necessary to insure that the instruction at the current breakpoint location will itself be executed on a proceed and that the breakpoint location will contain the SWI the next time it is executed. This is especially important when the breakpoint is in a loop in the user's program.

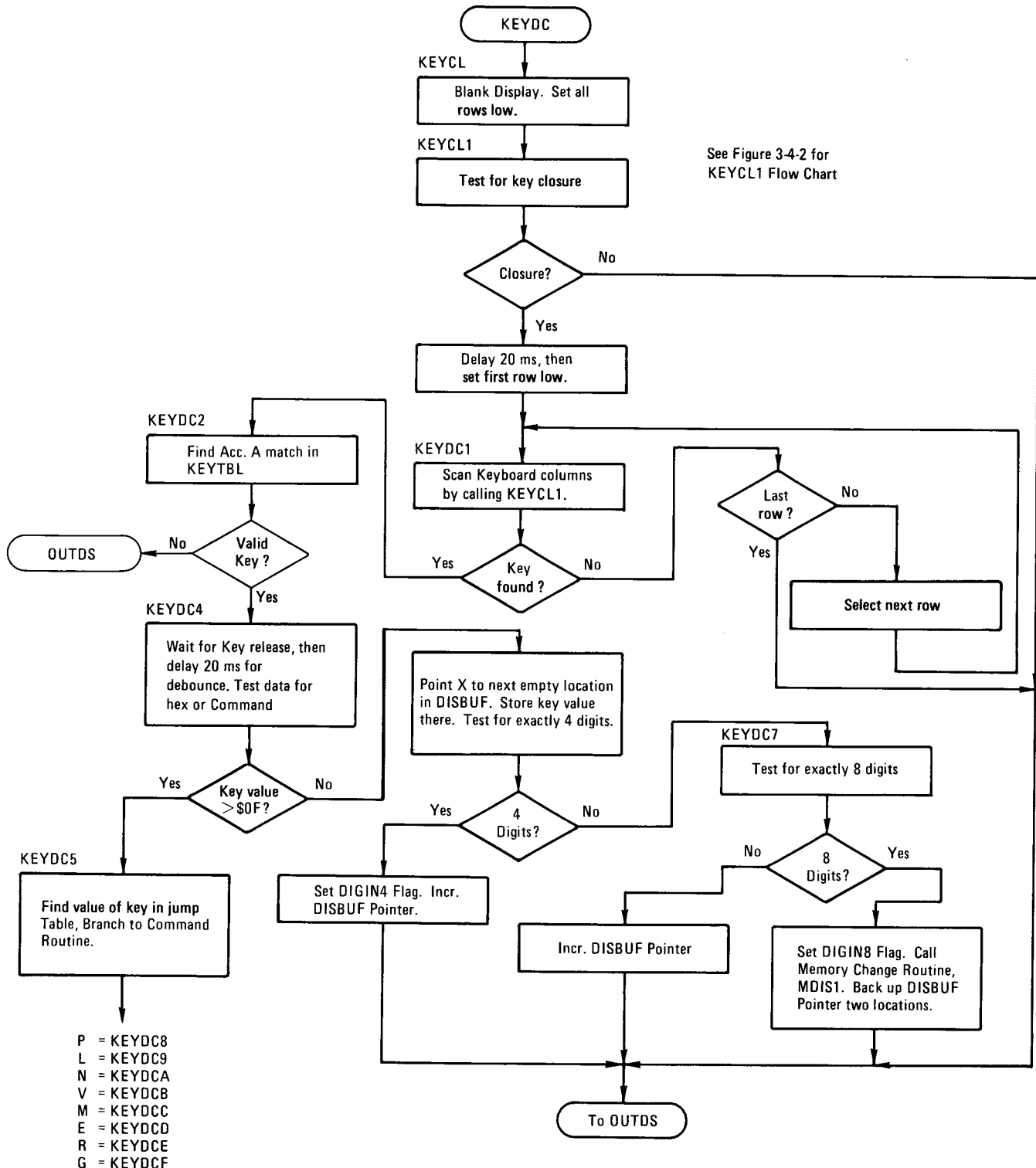
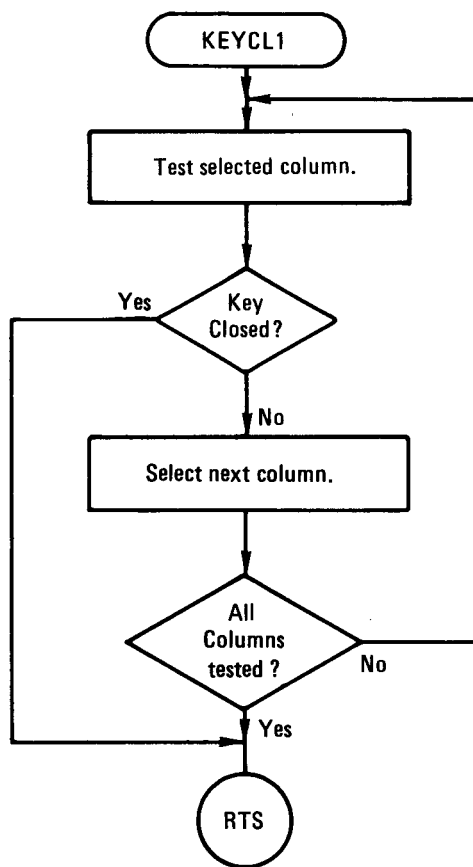


FIGURE 3-4-1. Program Flow for Keyboard Scan and Decode Routine

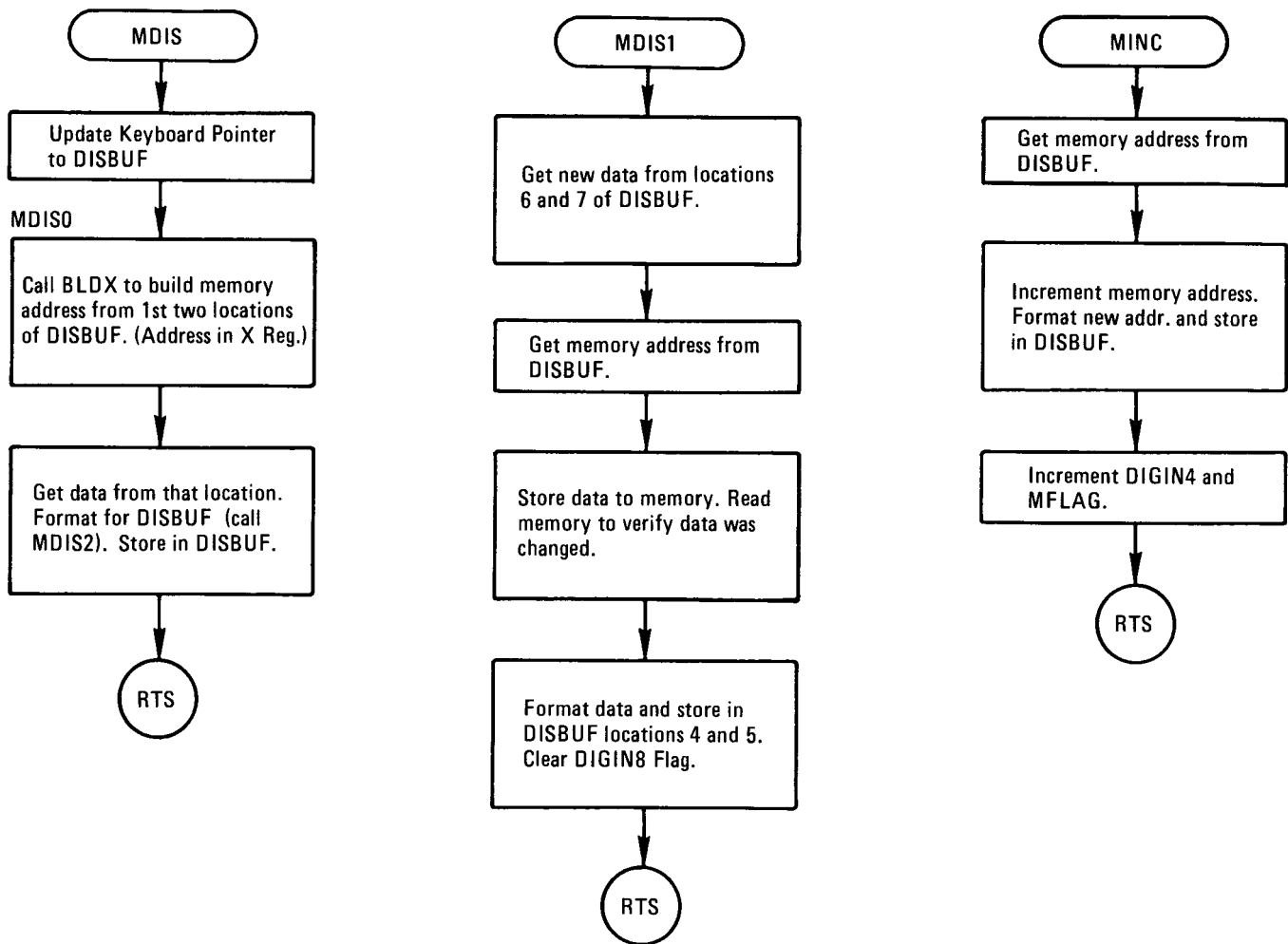


Returns with state of SCNREG in Acc. A when key closure is detected.

FIGURE 3-4-2. Program Flow for KEYCL1 Subroutine

3-5 MEMORY EXAMINE/CHANGE ROUTINE

Flow charts for the Display and Change Memory routines are shown in Figure 3-5-1. The Memory Display routine (MDIS, line 483) causes display of the contents of the memory location pointed to by the first four DISBUF locations. KEYBF, the pointer to the next empty location in DISBUF, is advanced by two in order to point to locations six and seven in DISBUF when new memory data is entered. The BLDX routine, via a jump through KEYD3F, builds a memory pointer from the data in the first four locations of DISBUF and loads it into the Index Register. The data from the location pointed to by X is loaded into Acc.A, split into nibbles (half-bytes or 4-bit words) by the MDIS2 subroutine, and stored in DISBUF locations four and five. Should a memory change be required, MDIS1 (line 496) is called, which gets the new data from locations six and seven in DISBUF (the keyboard entry) and stores it in the memory location referenced. A read of that location is then performed to get the actual data (someone might try to alter a ROM) which is put back in DISBUF+4 and DISBUF+5 to be displayed, giving the operator a visual indication that the change occurred. The Memory Increment Subroutine (MINC) is called when the G key is used to advance to the next memory location. This routine simply does a 16 bit increment of the four nibbles stored in the first four locations of DISBUF. MDIS is then called to display the contents of the incremented address.



(a) Display Memory

(b) Change Memory

(c) Increment Memory

FIGURE 3-5-1. Program Flow for Memory Display, Change, and Increment

3-6 REGISTER DISPLAY/CHANGE ROUTINE

The subroutine to display the registers (REGST, flow chart in Figure 3-6-1) transfers the User's Registers from his stack (User's Stack Pointer is always saved in SP) to the display for operator inspection. The registers are displayed in the order they are stacked: PC, X, A, B, C. A new register can be selected by pressing the G key while in the Register Display mode. This causes the register display routine to be entered at REGST1 (line 556). TEMP2, a RAM buffer, is used as a counter in this routine to determine whether the register is one or two bytes long, and which register to display next.

The Program Counter is displayed first so that when the Register Display routine is called from the Trace or Breakpoint routine, the Program Counter appears automatically, allowing the operator to easily follow program flow. REGST points the Index Register to the top of the user's Stack where the high byte of the program counter is located. REGST1 clears the display buffer, DISBUF, and determines from the count in TEMP2 which register is to be displayed. When the count gets to 3, all registers have been displayed and the user's Stack Pointer is loaded from location SP and displayed.

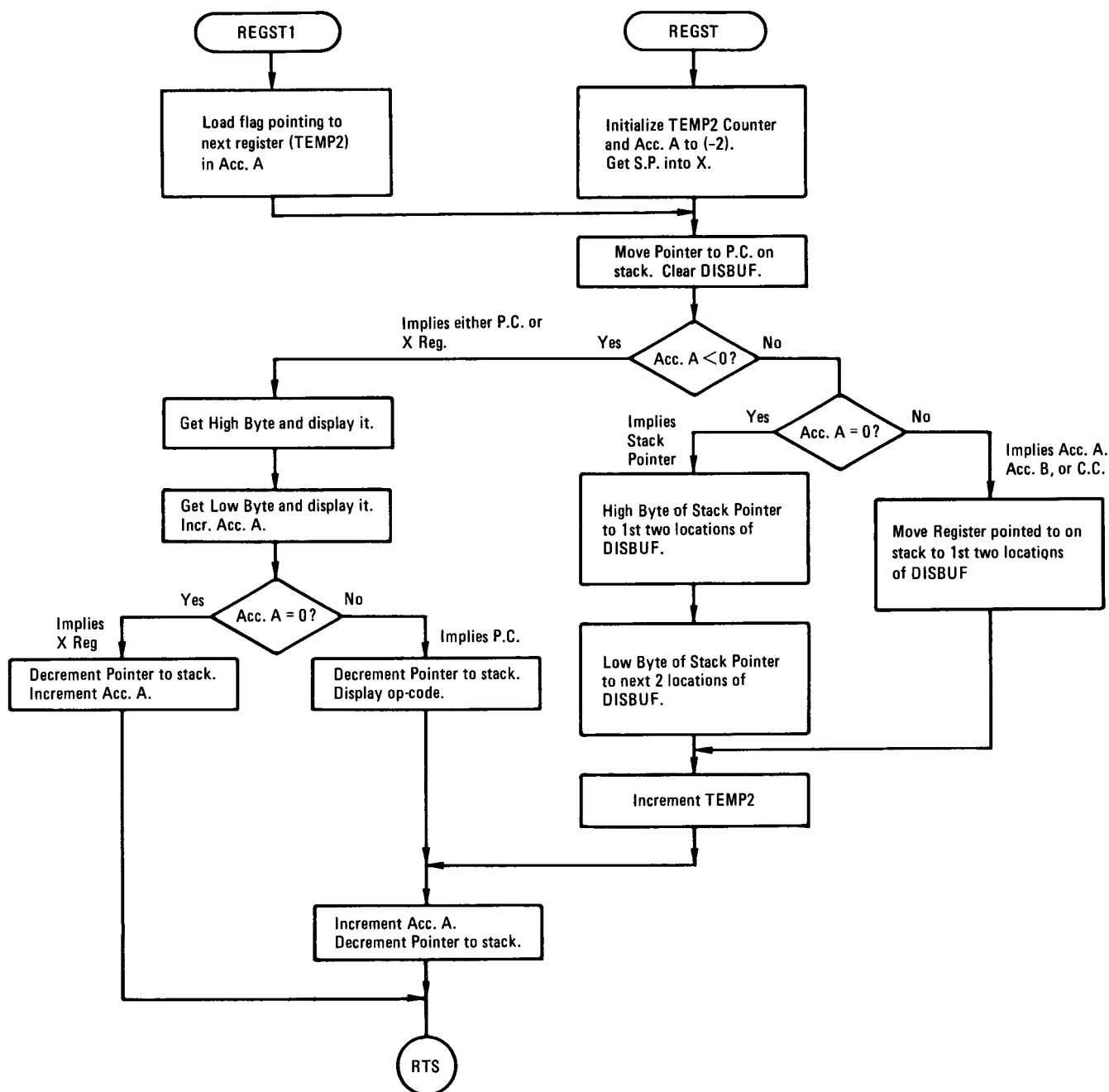


FIGURE 3-6-1. Program Flow for Register Display Function

3-7 PUNCH AND LOAD ROUTINES

The Punch routine (line 609, address \$E32F, flow chart in Figure 3-7-1) is entered via a decode of a P key closure. Initially, the ACIA is reset causing the $\overline{\text{RTS}}$ signal to go low. This is followed by ACIA programming to set $\overline{\text{RTS}}$ high, establish eight bits for data length, no parity, and two stop bits. Additionally, the ACIA is set up to transmit serial data at one sixteenth of the clock frequency. A leader is then punched (using the PNLDR Subroutine) consisting of 1024 ones.

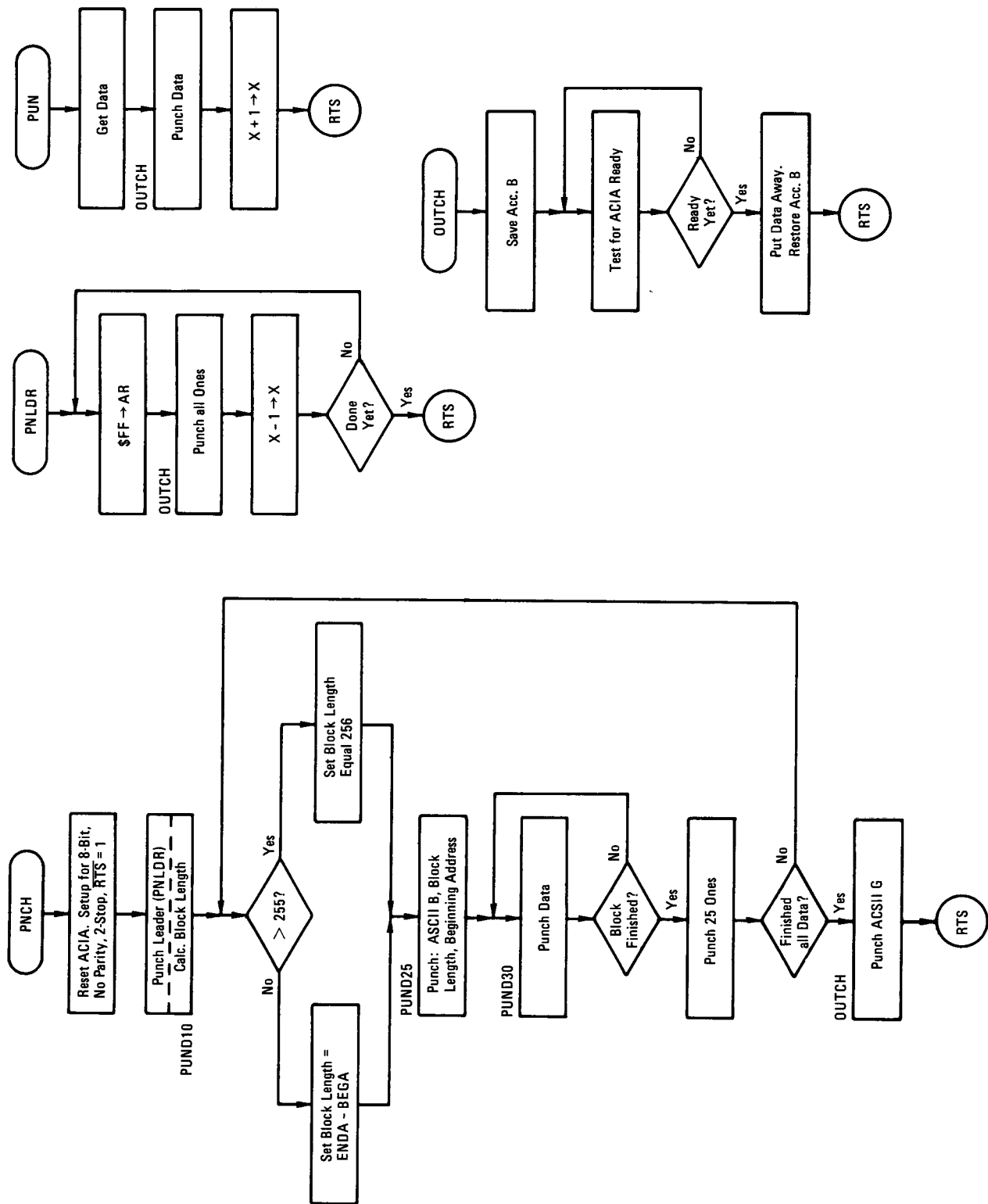


FIGURE 3-7-1. Program Flow for PUNCH Function

After the leader is punched, the program compares the beginning address (located in \$A002, \$A003) to the ending address (located in \$A004, \$A005). If the difference is greater than 256 (hex FF), the first block is assumed to be 256 bytes long. When the difference is less than 256, the block length is set equal to the difference.

Once this determination has been completed an ASCII "B" is punched on the tape. This is followed by the block length (one byte). The next information stored on the tape is the two byte beginning address of the data being put on the tape. After the block of data is outputted to the tape recorder, a leader of 25 ones data is put onto the tape. At this point the beginning address is again compared to the ending address in order to see if all the data has been punched. To provide a control to validate that all data has been recorded and for ease of recovery, an ASCII "G" is then punched on the tape. When the beginning address and the ending address are different, another block of data must be processed. This cycle is continued until the beginning and ending addresses are the same. Return to control is accomplished with an RTS instruction.

This routine destroys the beginning address originally put in the locations \$A002 and \$A003. When the punch routine is complete the data in the ending address is unchanged and the beginning address locations contain a value one greater than the end address.

The Load routine (line 674, address \$E395, flow chart in Figure 3-7-2) is entered via a decode of an L key closure. This routine sets up the ACIA to receive data in the same format that is used by the Punch routine: data length equals 8 bits, no parity, two stop bits. The Receive Clock mode is set to divide-by-one and $\overline{\text{RTS}}$ is set low, indicating that the ACIA is now ready to receive data from the cassette interface circuitry.

Each data byte is brought in by calling the Input One Character routine, INCHR (line 699, address \$E3C0). This routine continuously checks the ACIA's Status Register until there is an indication that a byte is ready to be transferred. The MPU then fetches the byte from the ACIA Data Receive Register and returns to the LOAD routine with the data in Acc.A. The data is then tested to determine if it is an ASCII "B" or "G". When a "B" is received, the program branches to the Read Data Block routine, RDBLCK. The block length is read and saved in Acc.B and the beginning address is read and stored into locations \$A002 and \$A003. Data in the current block is then brought in and stored to the indicated memory locations. After the block of data is read, the software branches back to the BILD Routine to look for another block of data or an end of file command. When other blocks of data are present in this file, they are processed as described above. Eventually, the end of file is reached. End of file recognition is accomplished by recognizing an ASCII "G" in the BILD routine. Recognition of this "G" provides the means for orderly exit from this routine by the execution of the RTS instruction.

3-8 INTERRUPT HANDLING ROUTINES

The JBUG monitor program handles all three types of M6800 interrupts: Software Interrupt (SWI), Maskable Interrupt Request ($\overline{\text{IRQ}}$), and Non-Maskable Interrupt ($\overline{\text{NMI}}$). In handling interrupts, the MC6800 completes execution of its current instruction, saves the results on the stack and then outputs the appropriate vector address. At that address it expects to find the beginning address of the selected interrupt service routine (see the reference literature for more details). Beginning addresses of the service routines are placed in the vector locations during program development.

The $\overline{\text{IRQ}}$ interrupt is reserved for the user. In servicing an $\overline{\text{IRQ}}$ interrupt, the MPU fetches the address \$E014 from memory locations \$E3F8 and \$E3F9 near the top of the JBUG ROM. Beginning at location \$E014 (line 83), the MPU loads the Index Register with the contents of RAM locations \$A000 and \$A001, then

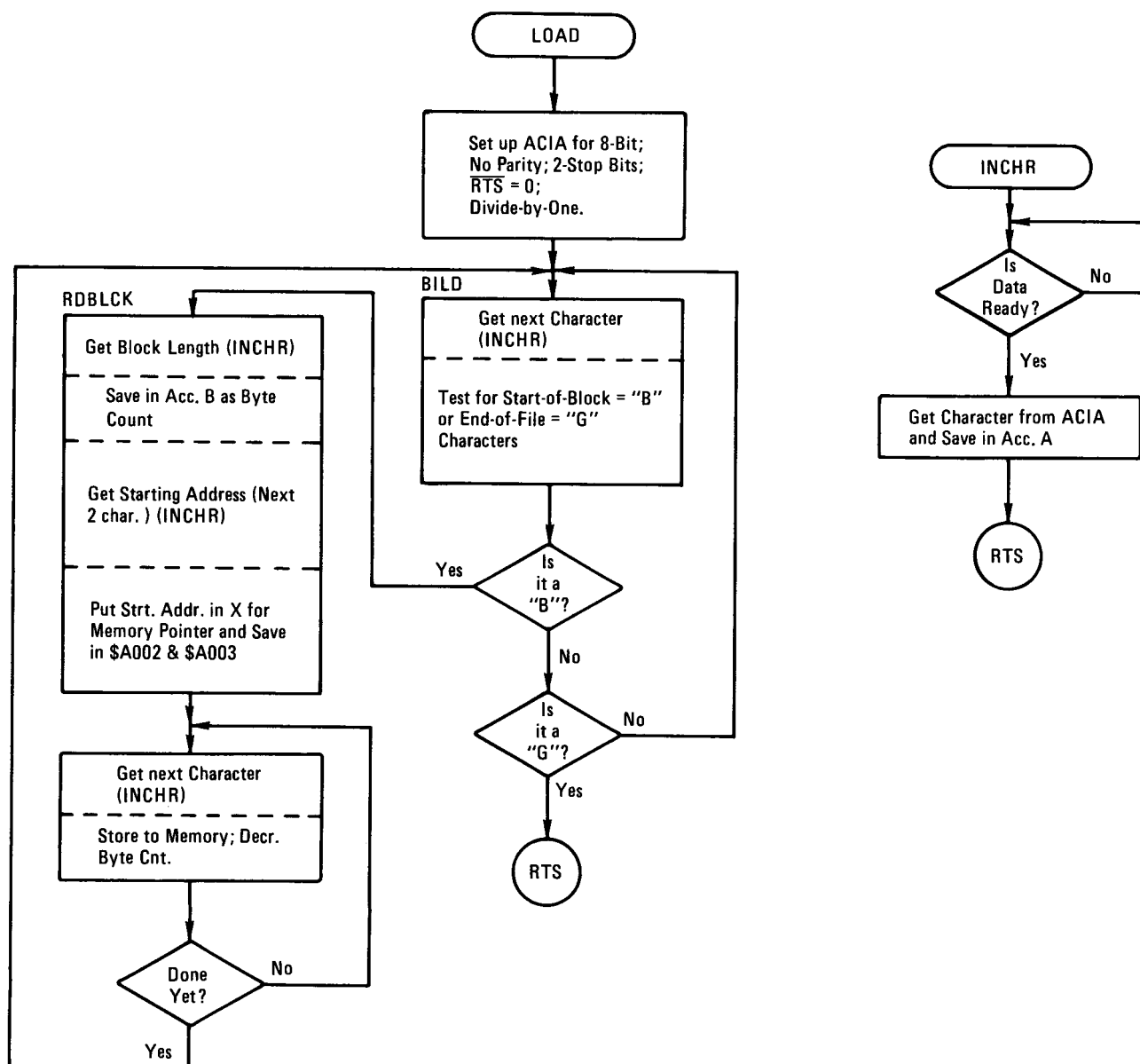


FIGURE 3-7-2. Program Flow for LOAD Function

executes an indexed jump. This, in effect, maps the IRQ vector through the JBUG ROM, allowing the user to reach his interrupt service routine by loading its beginning address into RAM locations \$A000 (high order byte) and \$A001 (low order byte).

The MPU is directed to location \$E019 (line 91) by NMI interrupts. The flow of the subroutine located there, NONMSK, is shown in Figure 3-8-1. NONMSK can be entered due to either a Trace command (breakpoints may be either active or clear) or because of an interrupt from the keyboard PIA, U21. If the interrupt was not a Trace command, then the trace flag, NFLAG, is cleared and the program flows to NONMK1 (line 100). The MPU loads the Index Register with the contents of memory locations \$A006 and \$A007 and then jumps to that location to begin executing the Keyboard Service Routine, KEYDC. This address was loaded into \$A006 and \$A007 during the Restart initialization sequence. The user may cause NMI interrupts to vector to other locations by loading the desired starting address into \$A006 and \$A007.

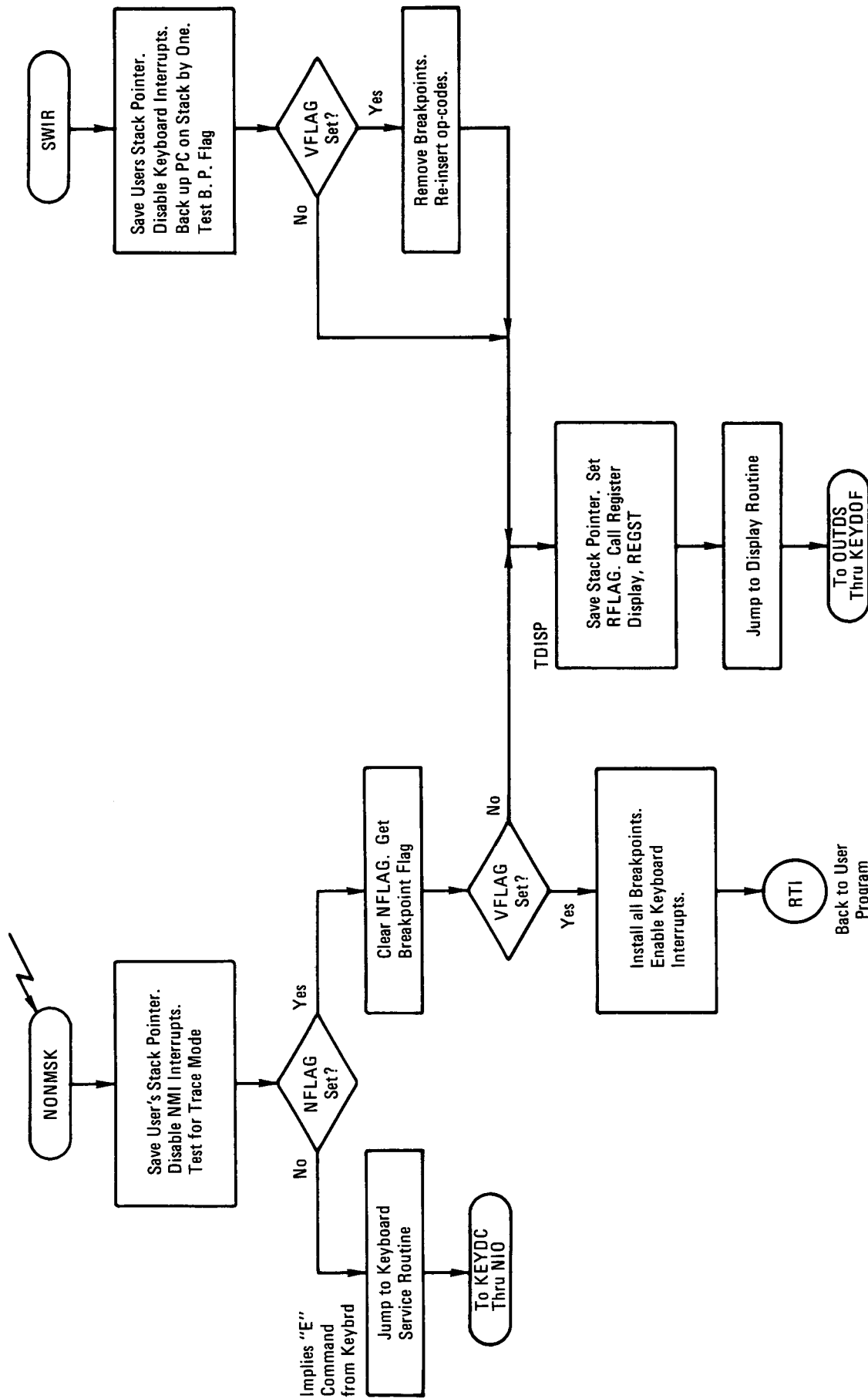


FIGURE 3-8-1. Program Flow for NMI and SWI Interrupt Handling

If the Trace flag (NFLAG) was set, the program checks to see if breakpoints are active. If breakpoints are active, it is assumed that the purpose of the Trace command was to get off of a breakpoint. In this case, the breakpoints are installed, further keyboard interrupts are enabled, and flow is passed back to the user program by execution of an RTI instruction. If there were no active breakpoints, it is assumed that the Trace command was invoked in order to execute a single instruction. In this case, the stack pointer is saved in SP and then the program jumps to the Register Display Routine.

Software Interrupts (SWI) are used by the JBUG monitor to implement breakpoints (up to a maximum of five are allowed). Upon entry from a SWI instruction SWIR (line 107), the user's Stack Pointer is saved in location SP for use by the Register Display Routine. Keyboard interrupts are disabled so that the normal Keyboard and Display scanning functions do not cause multiple $\overline{\text{NMI}}$ interrupts. Lines 109-113 cause a 16 bit decrement of the Program Counter saved on the Stack so that it points back to the instruction that was replaced by the SWI used to make the breakpoint. The subroutine GETXB is called (line 145) to examine the VFLAG and determine if any breakpoints are set. If there are, TZONK removes all of the SWI instructions so that the operator doesn't see them. The address of the breakpoints and their op-codes are saved in the Breakpoint Table, BPTAB. The Register Display Routine is then called so that the operator can examine the registers on the stack.

APPENDIX 1

ASSEMBLY LISTING OF JBUG MONITOR

PAGE 001 JBUG

[illegible]

[illegible]

```

00108 E035 8D 4D          BSR    DISNMI    DISABLE NMI INTERRUPTS
00109 E037 30            TSX      DECR PC BY 1
00110 E038 6D 06          TST     6,X      BACKUP PC ON STACK
00111 E03A 26 02          BNE     ++4
00112 E03C 6A 05          DEC     5,X
00113 E03E 6A 06          DEC     6,X
00114 E040 8D 21          BSR     GETXB     GET TAB ADDR AND VFLAG
00115 E042 27 14          BEQ     TDISP    NO BRKPTS, GO DISPLAY REGS
00116
00117          *
00118          * REMOVE BRKPTS WHILE WE ARE IN JBUG. THEY
00119          * WILL BE RESTORED ON A GO OR PROCEED
00120          *
00121          *+++++HERE TO REMOVE BREAKPOINTS+++++
00121 E044 FF A01E TZONK STX    BPADR    SAVE IN TEMP
00122 E047 A6 02          LDA     A 2,X      GET OP CODE TO RESTORE
00123          * SAFEGUARD AGAINST MULTI DEFINED BRKPTS
00124          *
00125 E049 81 3F          CMP     A #$3F
00126 E04B 27 07          BEQ     GENA      BRANCH IF MULTI-DEF
00127 E04D EE 00          LDX     0,X      GET ADDR OF BKPT
00128 E04F A7 00          STA     A 0,X      RESTORE OP CODE
00129 E051 FE A01E          LDX     BPADR    GET TABLE POSITION
00130 E054 8D 08 GENA BSR    ADD3X     GET NEXT POSITION AND DECB
00131 E056 26 EC          BNE     TZONK    GO AGAIN
00132 E058 BF A008 TDISP STS     SP      SAVE USER'S STACK POINTER
00133 E05B 7E E206          JMP     KEYDCE   GO DISPLAY REGS
00134
00135          *
00136          *++++SUBROUTINE TO GET NEXT TABLE ENTRY
00137          *
00137 E05E 08          ADD3X INX
00138 E05F 08          INX
00139 E060 08          INX
00140 E061 5A          DEC     B          DECR CTR
00141 E062 39          RTS              LET CALLER DO CTR CHECK
00142
00143          *
00144          *++++SUB TO GET TABLE ADDR IN X VFLAG IN B
00145          *
00145 E063 CE A022 GETXB LDX     #BPTAB    GET TABLE BASE ADDR
00146 E066 F6 A01D          LDA     B VFLAG
00147 E069 39          RTS
00148
00149          *
00149          *++SUBROUTINE TO SET A BREAKPOINT (MAKE AN
00150          *++++ENTRY INTO BREAKPOINT TABLE) IF ENOUGH
00151          *++++SPACE EXISTS
00152          * THE ACTUAL BRKPTS ARE PUT IN MEMORY
00153          * ON THE 'G' COMMAND
00154          *
00155 E06A 8D F7 SETBR BSR     GETXB     GET TAB ADDR AND VFLAG
00156 E06C 27 08          BEQ     TZDT     NO BKPTS, GO INSERT ONE
00157 E06E C1 05          CMP     B #$5      ENOUGH ROOM?
00158 E070 2C 52          BGE     CLRDS    NO, CLEAR DISPLAY AND RTS
00159          *++++GET TO FIRST FREE SPACE IN TABLE++++
00160 E072 8D EA TP16 BSR     ADD3X     ADD 3 TO X AND DECB
00161 E074 26 FC          BNE     TP16     BRANCH IF NOT DONE

```

```

00162          ♦♦♦♦♦INSERT NEW BKPT IN TABLE♦♦♦♦♦
00163 E076 7C A01D TZDT   INC    VFLAG      INCR FLAG
00164 E079 B6 A01E          LDA A   BPADR      INSERT IN TABLE
00165 E07C A7 00          STA A    0,X
00166 E07E B6 A01F          LDA A   BPADR+1
00167 E081 A7 01          STA A    1,X
00168 E083 39              RTS
00169          ♦
00170          ♦♦♦♦SUBROUTINE TO DISABLE NMI INTERRUPTS♦♦♦♦
00171          ♦
00172 E084 86 3C  DISNMI LDA A   #$3C
00173 E086 B7 8021          STA A   DISCTR     INTR MASKED CA1 ACTIVE LOW
00174 E089 B7 8023          STA A   SCNCTR     INTR MASKED CB1 ACTIVE LOW
00175 E08C 39              RTS
00176          ♦
00177          ♦
00178          ♦♦♦♦RESTART ROUTINE♦♦♦♦
00179          ♦
00180          ♦
00181 E08D 8E A078 RESTAR LDS    #$A078
00182 E090 BF A008          STS     SP          INITIALIZE STACK POINTER
00183 E093 CE E14E          LDX     #KEYDC     GO DECODE KEYBOARD
00184 E096 FF A006          STX     NID        INITIALIZE NMI INTERRUPT
00185          ♦INITIALIZE KEYBOARD/DISPLAY PIA
00186 E099 86 FF          LDA A   #$FF
00187 E09B B7 8022          STA A   SCNREG     PB0-PB7 OUTPUTS
00188 E09E 44              LSR A
00189 E09F B7 8020          STA A   DISREG     PA0-PA6 OUTPUTS, PA7 INPUT
00190 E0A2 8D E0          BSR     DISNMI     DISABLE KEYBOARD/TRACE
00191          ♦♦INITIALIZE ACIA♦♦
00192 E0A4 86 03          LDA A   #3
00193 E0A6 B7 8008          STA A   ACIAS      RESET THE ACIA
00194 E0A9 7F A01D          CLR     VFLAG     INITIALIZE VFLAG
00195 E0AC 8D 04  INIT    BSR     CLFLG      CLEAR DISPLAY AND FLAGS
00196 E0AE 8D 27          BSR     HDR        WRITE PROMPT "--"
00197 E0B0 20 4C          BRA     OUTDS
00198          ♦
00199          ♦♦♦♦SUBROUTINE TO CLEAR DISPLAY BUFFER AND FLAGS♦♦♦♦
00200          ♦
00201 E0B2 CE A014 CLFLG  LDX     #DIGIN4
00202 E0B5 4F          CLR A
00203 E0B6 A7 00  CLFLG1 STA A   0,X      CLEARS DIGIN4 AND DIGIN8
00204 E0B8 08          INX
00205 E0B9 8C A01A          CPX     #DIGIN4+6  END?
00206 E0BC 26 F8          BNE     CLFLG1     NO LOOP BACK
00207 E0BE CE A00C          LDX     #DISBUF
00208 E0C1 FF A01A          STX     XKEYBF     INITIALIZE XKEYBF
00209 E0C4 86 7F  CLRDS  LDA A   #$7F
00210 E0C6 B7 8020          STA A   DISREG     BLANK DISPLAY
00211 E0C9 86 11          LDA A   #17
00212 E0CB CE A00C          LDX     #DISBUF
00213 E0CE A7 00  CLRDS1 STA A   0,X      CLEAR OUT DISPLAY BUFFER
00214 E0D0 08          INX
00215 E0D1 8C A014          CPX     #DISBUF+8  END?

```

```

00216 E0D4 26 F8      BNE      CLRDS1
00217 E0D6 39          RTS
00218                *
00219                *SUBROUTINE TO WRITE PROMPT ON DISPLAY
00220                *
00221 E0D7 86 10 HDR      LDA A    #16
00222 E0D9 B7 A00C      STA A    DISBUF  OUTPUT -
00223 E0DC 39          RTS
00224                *
00225                *SUBROUTINE TO DELAY 20 MS OR X MS
00226                *  WHEN ENTERING AT DLY1 THE XREG MUST CONTAIN
00227                *  THE DESIRED DELAY CT (APX 13USEC/COUNT)
00228                *
00229 E0DD CE 0600 DLY20  LDX      #$0600
00230 E0E0 09 DLY1      DEX
00231 E0E1 26 FD      BNE      DLY1
00232 E0E3 39          RTS
00233                *
00234                *SUBROUTINE TO BUILD TWO BYTE ADDRESS FROM
00235                *SUBROUTINE TO BUILD TWO BYTE ADDRESS FROM
00236                *  ADDRESS IS IN X-REG AND 'BPADR' ON EXIT
00237                *
00238 E0E4 CE A00C BLDX    LDX      #DISBUF
00239 E0E7 A6 00      LDA A      0,X      GET FIRST BYTE
00240 E0E9 48          ASL A
00241 E0EA 48          ASL A
00242 E0EB 48          ASL A
00243 E0EC 48          ASL A      MOVE TO HIGH NIBBLE
00244 E0ED AA 01      ORA A      1,X      OR WITH LOW NIBBLE
00245 E0EF A7 12      STA A      BPADR-DISBUF,X  STORE IN BPADR
00246 E0F1 A6 02      LDA A      2,X      GET SECOND BYTE
00247 E0F3 48          ASL A
00248 E0F4 48          ASL A
00249 E0F5 48          ASL A
00250 E0F6 48          ASL A      MOVE TO HIGH NIBBLE
00251 E0F7 AA 03      ORA A      3,X      OR WITH LOW NIBBLE
00252 E0F9 A7 13      STA A      BPADR+1-DISBUF,X  STORE IN BPADR+1
00253 E0FB EE 12      LDX      BPADR-DISBUF,X  ADDRESS TO XREG
00254 E0FD 39          RTS
00255                *
00256                *
00257                *SUBROUTINE TO DISPLAY 6 DIGITS IN DISBUF
00258                *
00259                *
00260 E0FE CE A00C OUTDS  LDX      #DISBUF  GET STARTING ADDRESS
00261 E101 A6 00 OUTDS1 LDA A      0,X      GET FIRST DIGIT
00262 E103 4C          INC A
00263 E104 08          INX
00264 E105 FF A020      STX      XDSBUF  SAVE POINTER
00265 E108 CE E3C9      LDX      #DIGTBL-1
00266 E10B 08 OUTDS2 INX
00267 E10C 4A          DEC A      POINT TO PATTERN
00268 E10D 26 FC      BNE      OUTDS2
00269 E10F 7F 8022      CLR      SCNREG  BLANK DISPLAY

```



```

00270 E112 A6 00          LDA A 0,X      GET PATTERN
00271 E114 B7 8020        STA A DISREG   SET UP SEGMENTS
00272 E117 B6 A01C        LDA A SCNCNT
00273 E11A B7 8022        STA A SCNREG   SELECT DIGIT
00274 E11D CE 004D        LDX # $4D     SETUP FOR 1MS DELAY
00275 E120 8D BE          BSR DLY1      DELAY 1 MS
00276 E122 FE A020        LDX XDSBUF    RECOVER POINTER
00277 E125 8C A012        CPX #DISBUF+6
00278 E128 27 1F          BEQ OUTDS3
00279 E12A 74 A01C        LSR SCNCNT    NO MOVE TO NEXT DIGIT
00280 E12D 20 D2          BRA OUTDS1
00281
00282          ◆
00283          ◆◆◆◆SUBROUTINE TO SCAN KEYBOARD◆◆◆◆
00284 E12F 86 FF          KEYCL LDA A # $FF
00285 E131 CE 8020        LDX #DISREG
00286 E134 A7 00          STA A 0,X      BLANK DISPLAY
00287 E136 86 3F          LDA A # $3F
00288 E138 A7 02          STA A 2,X      ALL ROWS LOW
00289 E13A A6 02          KEYCL1 LDA A 2,X
00290 E13C 6D 00          TST 0,X
00291 E13E 2A 08          BPL KEYCL2     KEY DOWN?
00292 E140 8B 40          ADD A #64
00293 E142 A7 02          STA A 2,X      SELECT NEXT COLUMN
00294 E144 84 C0          AND A # $C0
00295 E146 26 F2          BNE KEYCL1    LAST COLUMN SCANNED?
00296 E148 39            KEYCL2 RTS      NO KEY FOUND
00297 E149 86 20          OUTDS3 LDA A # $20
00298 E14B B7 A01C        STA A SCNCNT   INITIALIZE SCNCNT
00299
00300          ◆
00301          ◆◆◆◆ROUTINE TO SCAN AND DECODE KEYBOARD◆◆◆◆
00302 E14E 8D DF          KEYDC BSR KEYCL
00303 E150 27 AC          BEQ OUTDS      NO KEY CLOSED
00304 E152 8D 89          BSR DLY20
00305 E154 CE 8020        LDX #DISREG   RESTORE X
00306 E157 86 01          LDA A # $01    SETUP SCAN FOR FIRST ROW
00307 E159 A7 02          STA A 2,X
00308 E15B 8D DD          KEYDC1 BSR KEYCL1 SCAN KEYBOARD, GET KEY
00309 E15D 26 0A          BNE KEYDC2     KEY FOUND
00310 E15F A6 02          LDA A 2,X      CLEARS NMI INTERRUPT
00311 E161 81 20          CMP A # $20
00312 E163 27 99          BEQ OUTDS      LAST ROW
00313 E165 68 02          ASL 2,X      SHIFT LEFT
00314 E167 20 F2          BRA KEYDC1
00315 E169 5F            KEYDC2 CLR B     INITIALIZE COUNTER
00316 E16A CE E3DC        LDX #KEYTBL
00317 E16D A1 00          KEYDC3 CMP A 0,X SEARCH TABLE
00318 E16F 27 09          BEQ KEYDC4
00319 E171 8C E3F4        CPX #KEYTBL+24 END OF TABLE?
00320 E174 27 61          BEQ KEYD0F     NO KEY FOUND IN TABLE
00321 E176 08            INX
00322 E177 5C            INC B           ADVANCE
00323 E178 20 F3          BRA KEYDC3

```

```

00324 E17A 8D B3 KEYDC4 BSR KEYCL WAIT FOR KEY RELEASE
00325 E17C 26 FC BNE KEYDC4
00326 E17E BD E0DD JSR DLY20 DELAY 20 MSEC
00327 E181 C1 0F CMP B #$0F
00328 E183 2E 27 BGT KEYDC5
00329 E185 FE A01A LDX XKEYBF POINTER IN DISBUF
00330 E188 E7 00 STA B 0,X STORE KEY VALUE
00331 E18A 8C A00F CPX #DISBUF+3 4 DIGITS IN?
00332 E18D 26 09 BNE KEYDC7 NO
00333 E18F 7C A014 INC DIGIN4 YES
00334 E192 08 KEYDC6 INX
00335 E193 FF A01A STX XKEYBF
00336 E196 20 3F BRA KEYD0F
00337 E198 8C A013 KEYDC7 CPX #DISBUF+7 8 DIGITS IN?
00338 E19B 26 F5 BNE KEYDC6
00339 E19D 7C A015 INC DIGIN8 SET FLAG
00340 E1A0 BD E27E JSR MDIS1 DISPLAY NEW DATA
00341 E1A3 FE A01A LDX XKEYBF
00342 E1A6 09 DEX BACK UP POINTER
00343 E1A7 FF A01A STX XKEYBF SAVE
00344 E1AA 20 2B BRA KEYD0F
00345
00346 *
00347 * HERE TO DISPATCH TO A KEYBOARD OPTION
00348 **
00349 *
00349 E1AC CE E196 KEYDC5 LDX #JMPTAB-32
00350 E1AF 08 KYDC5 INX GET TO ADDRESS IN JUMP TABLE
00351 E1B0 08 INX
00352 E1B1 5A DEC B
00353 E1B2 26 FB BNE KYDC5 THIS ONE?
00354 E1B4 6E 00 JMP 0,X YES
00355 E1B6 20 0E JMPTAB BRA KEYDC8 P KEY
00356 E1B8 20 14 BRA KEYDC9 L KEY
00357 E1BA 20 1E BRA KEYDCA N KEY
00358 E1BC 20 28 BRA KEYDCB V KEY
00359 E1BE 20 37 BRA KEYDCC M KEY
00360 E1C0 20 41 BRA KEYDCD E KEY
00361 E1C2 20 42 BRA KEYDCE R KEY
00362 E1C4 20 48 BRA KEYDCF G KEY
00363
00364 *
00365 * HERE ON P KEY
00366 * PUNCH MEMORY TO AUDIO CASSETTE
00367 *
00367 E1C6 BD E0C4 KEYDC8 JSR CLRDS CLEAR DISPLAY
00368 E1C9 BD E32F JSR PNCH PUNCH DATA TO CASSETTE
00369 E1CC 20 06 BRA KEYDCH
00370
00371 *
00372 * HERE ON L KEY
00373 * LOAD MEMORY FROM AUDIO CASSETTE
00374 *
00374 E1CE BD E0C4 KEYDC9 JSR CLRDS CLEAR DISPLAY
00375 E1D1 BD E395 JSR LOAD LOAD DATA FROM CASSETTE
00376 E1D4 BD E0D7 KEYDCH JSR HDR WRITE HEADER
00377 * RETURN TO DISPLAY HEADER

```

```

00378 E1D7 7E E0FE KEYD0F JMP      OUTDS      DISPLAY HEADER
00379
00380
00381
00382
00383 E1DA 7F A01D KEYDCA CLR      VFLAG
00384 E1DD 7C A018 TRACE  INC      NFLAG
00385 E1E0 86 34          LDA A  #$34      SET UP HARDWARE TO TRACE
00386 E1E2 B7 8021          STA A  DISCTR  CA2 LOW START TRACE
00387 E1E5 3B          RTI
00388
00389
00390
00391
00392
00393 E1E6 7D A014 KEYDCB TST      DIGIN4    4 DIGITS IN?
00394 E1E9 26 05          BNE  ++7      YES, INSERT BP
00395 E1EB 7F A01D          CLR      VFLAG
00396 E1EE 20 E7          BRA      KEYD0F    GO DISPLAY
00397 E1F0 8D 74          BSR      KEYD3F    YES, INSERT BREAKPOINT
00398 E1F2 BD E06A          JSR      SETBR
00399 E1F5 20 E0          BRA      KEYD0F
00400
00401
00402
00403
00404 E1F7 7C A016 KEYDCC INC      MFLAG    SET FLAG
00405 E1FA 7D A014          TST      DIGIN4    4 DIGITS IN?
00406 E1FD 27 D8          BEQ      KEYD0F    NO
00407 E1FF 8D 68          BSR      MDIS      YES, DISPLAY MEMORY
00408 E201 20 D4          BRA      KEYD0F
00409
00410
00411
00412
00413 E203 7E E0AC KEYDCD JMP      INIT      CLEAR DISPLAY AND FLAGS
00414
00415
00416
00417
00418 E206 7C A017 KEYDCE INC      RFLAG    REGISTER DISPLAY
00419 E209 BD E2C6          JSR      REGST
00420
00421 E20C 20 C9          KEYDCG BRA      KEYD0F
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431 E20E 7D A016 KEYDCF TST      MFLAG    MEMORY MODE?

```

```

00432 E211 26 48      BNE     KEYD1F    YES
00433 E213 7D A017    TST     RFLAG
00434 E216 26 49      BNE     KEYD2F
00435                ♦ IS IT A 'GO' OR 'PROCEED'?
00436 E218 7D A014    TST     DIGIN4    4 DIGITS IN?
00437 E21B 26 07      BNE     KEYDCJ    NO, PROCEED MODE
00438                ♦ HERE ON PROCEED
00439 E21D BD E063     JSR     GETXB      GET ADDR AND VFLAG
00440 E220 27 2B      BEQ     TGC        BRANCH IF NO BREAKPOINTS
00441 E222 20 B9      BRA     TRACE      GO TRACE
00442                ♦ HERE ON GO MODE
00443 E224 8D 40      KEYDCJ BSR     KEYD3F  GET ADDR
00444 E226 30          TSX
00445 E227 A7 06      STA A    6,X      MODIFY LOW BYTE
00446 E229 F6 A01E    LDA B    BPADR    GET LOW BYTE
00447 E22C E7 05      STA B    5,X      MODIFY HIGH BYTE
00448 E22E BD E0C4     JSR     CLRDS     CLEAR DISPLAY
00449 E231 BD E063     JSR     GETXB      GET TAB ADDR&VFLAG
00450 E234 27 17      BEQ     TGC        BRANCH IF NO BP
00451                ♦♦♦♦INSTALL ALL BREAKPOINTS♦♦♦♦
00452 E236 FF A01E    TGB      STX     BPADR    SAVE IN TEMP
00453 E239 EE 00      LDX     0,X      GET ADDR OF BP
00454 E23B A6 00      LDA A    0,X      GET OP-CODE
00455 E23D 36        PSH A          SAVE
00456 E23E 86 3F      LDA A    #$3F      INSTALL A SWI
00457 E240 A7 00      STA A    0,X
00458 E242 FE A01E    LDX     BPADR    GET BACK CURR TAB LOC
00459 E245 32        PUL A          GET BACK OP-CODE
00460 E246 A7 02      STA A    2,X      SAVE IT IN A TABLE
00461 E248 BD E05E    JSR     ADD3X     GET NEXT TAB LOC
00462 E24B 26 E9      BNE     TGB      MORE TO DO?
00463                ♦ PREPARE TO RETURN TO USER
00464 E24D 86 20      TGC      LDA A    #$20
00465 E24F B7 8022    STA A    SCNREG    SETUP FOR KB INTR
00466 E252 F6 8022    LDA B    SCNREG    DUMMY READ TO CLEAR INTR
00467 E255 86 3D      LDA A    #$3D
00468 E257 B7 8023    STA A    SCNCTR    ENABLE KB INTR
00469 E25A 3B        RTI            BACK TO USER
00470                ♦ HERE TO DISPLAY NEXT MEM LOC
00471 E25B 8D 47      KEYD1F BSR     MINC     MEMORY INCREMENT
00472 E25D 8D 12      BSR     MDIS0     MEMORY DISPLAY
00473 E25F 20 AB      BRA     KEYDCG
00474                ♦ HERE ON DISPLAY NEXT REGISTER
00475 E261 BD E2D7    KEYD2F JSR     REGST1   REGISTER DISPLAY
00476 E264 20 A6      BRA     KEYDCG
00477 E266 7E E0E4    KEYD3F JMP     BLDX
00478                ♦♦
00479                ♦
00480                ♦♦SUBROUTINE TO DISPLAY MEMORY AND CHANGE IT♦♦
00481                ♦
00482                ♦
00483 E269 FE A01A    MDIS     LDX     XKEYBF
00484 E26C 08        INX
00485 E26D 08        INX

```

```

00486 E26E FF A01A      STX    XKEYBF    UPDATE POINTER
00487 E271 8D F3      MDIS0  BSR    KEYD3F    GET ADDR OF MEM LOCATION
00488 E273 A6 00      LDA    A    0,X        GET MEMORY DATA
00489 E275 8D 23      BSR    MDIS2    FORMAT DATA
00490 E277 B7 A010      STA    A    DISBUF+4  STORE DATA IN DISBUF
00491 E27A F7 A011      STA    B    DISBUF+5
00492 E27D 39          RTS
00493
00494          *
00494          * SUB TO PUT NEW DATA IN MEMORY AND DISPLAY IT
00495          *
00496 E27E F6 A012 MDIS1 LDA    B    DISBUF+6  GET NEW DATA
00497 E281 58          ASL    B
00498 E282 58          ASL    B
00499 E283 58          ASL    B
00500 E284 58          ASL    B
00501 E285 FA A013      ORA    B    DISBUF+7  OR WITH LOW NIBBLE
00502 E288 8D DC      BSR    KEYD3F    GET MEMORY ADDR AGAIN
00503 E28A E7 00      STA    B    0,X        STORE NEW DATA
00504 E28C A6 00      LDA    A    0,X        ACTUAL DATA IN MEMORY
00505 E28E 8D 0A      BSR    MDIS2    FORMAT
00506 E290 B7 A010      STA    A    DISBUF+4  ACTUAL DATA TO DISPLAY
00507 E293 F7 A011      STA    B    DISBUF+5
00508 E296 7F A015      CLR    DIGIN8    SETUP FOR NEW DATA ENTRY
00509 E299 39          RTS
00510
00511          *
00511          *◆◆SUBROUTINE TO MOVE LOW NIBBLE OF A TO B AND TO
00512          *◆◆◆◆MOVE HIGH NIBBLE OF A TO LOW NIBBLE OF A
00513          *
00514 E29A 16          MDIS2  TAB
00515 E29B C4 0F      AND    B    #$0F    MASK LOW NIBBLE
00516 E29D 84 F0      AND    A    #$F0    MASK HIGH NIBBLE
00517 E29F 44          LSR    A
00518 E2A0 44          LSR    A
00519 E2A1 44          LSR    A
00520 E2A2 44          LSR    A
00521 E2A3 39          RTS
00522
00523          *
00523          *◆ SUBROUTINE TO INC MEMORY DISPLAY AND CH6?
00524          *
00525 E2A4 8D C0      MINC   BSR    KEYD3F    GET MEMORY ADDRESS
00526 E2A6 08          INX
00527 E2A7 FF A00A      STX    TEMP1    SAVE
00528 E2AA B6 A00A      LDA    A    TEMP1    GET HIGH BYTE
00529 E2AD 8D EB      BSR    MDIS2    FORMAT FOR DISBUF
00530 E2AF CE A00C      LDX    #DISBUF
00531 E2B2 A7 00      STA    A    0,X
00532 E2B4 E7 01      STA    B    1,X        PUT IN DISPLAY BUFFER
00533 E2B6 B6 A00B      LDA    A    TEMP1+1  GET LOW BYTE
00534 E2B9 8D DF      BSR    MDIS2    FORMAT
00535 E2BB A7 02      STA    A    2,X
00536 E2BD E7 03      STA    B    3,X
00537 E2BF 7C A014      INC    DIGIN4    FOUR DIGITS ENTERED
00538 E2C2 7C A016      INC    MFLAG    SETUP FOR MEMORY EXAMINE
00539 E2C5 39          RTS

```

```

00540      *
00541      *
00542      ***SUBROUTINE TO DISPLAY REGISTERS ON USERS STACK
00543      *
00544      * ORDER OF DISPLAY IS: PC,X,A,B,CC,SP
00545      *   TEMP2 STARTS AT -2 AND ADVANCES TO +3 AND
00546      *   CORRESPONDS TO THE ORDER OF DISPLAY
00547      *
00548 E2C6 86 FE REGST LDA A  #$FE      INITIALIZE COUNTER
00549 E2C8 B7 A019      STA A  TEMP2
00550 E2CB FE A008      LDX   SP        GET USER'S SP
00551 E2CE 86 06      LDA A  06
00552 E2D0 08 REGST0 INX              POINT TO TOP OF STACK
00553 E2D1 4A      DEC A
00554 E2D2 26 FC      BNE   REGST0
00555 E2D4 FF A00A      STX   TEMP1     TEMP X LOCATION
00556 E2D7 BD E0C4 REGST1 JSR   CLRDS   CLEAR DISPLAY
00557 E2DA FE A00A      LDX   TEMP1     RESTORE X
00558 E2DD B6 A019      LDA A  TEMP2
00559 E2E0 2B 0E      BMI   REGST2     PC AND X REGS
00560 E2E2 81 03      CMP A  03        IS IT SP?
00561 E2E4 27 21      BEQ   REGST3     YES
00562 E2E6 81 04      CMP A  04        ALL REGS OUT START OVER
00563 E2E8 27 DC      BEQ   REGST
00564 E2EA A6 00      LDA A  0,X      OUTPUT A,B,CC
00565 E2EC 8D 2E      BSR   REGST5     DISPLAY ONE BYTE
00566 E2EE 20 21      BRA   REGST4     UPDATE COUNTER
00567 E2F0 36 REGST2 PSH A          SAVE A
00568 E2F1 A6 00      LDA A  0,X      GET HIGH BYTE
00569 E2F3 8D 27      BSR   REGST5     DISPLAY
00570 E2F5 FE A00A      LDX   TEMP1
00571 E2F8 A6 01      LDA A  1,X      GET LOW BYTE
00572 E2FA 8D 2B      BSR   REGST6     DISPLAY
00573 E2FC 32      PUL A          RESTORE A
00574 E2FD 4C      INC A          X REG? (A=0)
00575 E2FE 27 11      BEQ   REGST4     YES
00576 E300 8D 12      BSR   REGST8     DEC POINTER
00577 E302 BD E271      JSR   MDIS0
00578 E305 20 0A      BRA   REGST4     UPDATE COUNTER
00579 E307 B6 A008 REGST3 LDA A  SP      SP TO DISPLAY
00580 E30A 8D 10      BSR   REGST5     DISPLAY
00581 E30C B6 A009      LDA A  SP+1
00582 E30F 8D 16      BSR   REGST6
00583 E311 7C A019 REGST4 INC   TEMP2   UPDATE COUNTER
00584 E314 FE A00A REGST8 LDX   TEMP1   INCREMENT X
00585 E317 09      DEX
00586 E318 FF A00A      STX   TEMP1     SAVE X
00587 E31B 39      RTS
00588      *
00589      ***SUBROUTINE TO MOVE TWO DIGITS IN A TO FIRST TWO
00590      ***LOCATIONS IN THE DISPLAY BUFFER (DISBUF)
00591      *
00592 E31C BD E29A REGST5 JSR   MDIS2     FORMAT
00593 E31F CE A00C      LDX   0DISBUF

```

```

00594 E322 A7 00 REGST7 STA A 0,X FIRST DIGIT(OR THIRD)
00595 E324 E7 01 STA B 1,X SECOND DIGIT
00596 E326 39 RTS
00597
00598 ♦♦♦SUBROUTINE TO MOVE TWO DIGITS IN A TO SECOND TWO L
00599 ♦♦♦LOCATIONS IN THE DISPLAY BUFFER (DISBUF)
00600 ♦
00601 E327 BD E29A REGST6 JSR MDIS2 FORMAT
00602 E32A CE A00E LDX #DISBUF+2 THIRD & FOURTH DIGITS
00603 E32D 20 F3 BRA REGST7
00604 ♦
00605 ♦
00606 ♦♦♦SUBROUTINE TO PUNCH DATA TO CASSETTE TAPE♦♦♦
00607 ♦ AUDIO CASSETTE WITH KC STANDARD
00608 ♦
00609 E32F 86 51 PNCH LDA A #01010001 8 BIT CHR PAR 2 STOP
00610 E331 B7 8008 STA A ACIAS DIVIDE BY 16 WITH RTS NOT HIG
00611 E334 CE 03FF LDX #03FF
00612 E337 8D 54 BSR PNLDR PUNCH LEADER
00613 E339 F6 A005 PUND10 LDA B ENDA+1 FORM END TEMP REG
00614 E33C F0 A003 SUB B BEGA+1
00615 E33F B6 A004 LDA A ENDA
00616 E342 B2 A002 SBC A BEGA
00617 E345 27 02 BEQ PUND25 DIFF LESS THAN 255
00618 E347 C6 FF LDA B #0FF YES, SET BLOCK=256
00619 E349 86 42 PUND25 LDA A #1B PUNCH B
00620 E34B 8D 2D BSR DUTCH
00621 E34D 37 PSH B
00622 E34E 30 TSX
00623 E34F 8D 36 BSR PUN
00624 E351 32 PUL A GET BYTE COUNT
00625 E352 4C INC A ADJUST IT
00626 E353 B7 A019 STA A TEMP2
00627 E356 CE A002 LDX #BEGA PUNCH ADDR
00628 E359 8D 2C BSR PUN
00629 E35B 8D 2A BSR PUN
00630 E35D FE A002 LDX BEGA PUNCH DATA
00631 E360 8D 25 PUND30 BSR PUN
00632 E362 7A A019 DEC TEMP2 DONE YET?
00633 E365 26 F9 BNE PUND30 NO
00634 E367 FF A002 STX BEGA SAVE XR VALUE
00635 E36A CE 0019 LDX #0019
00636 E36D 8D 1E BSR PNLDR PUNCH 25 ONES
00637 E36F FE A002 LDX BEGA RESTORE XR
00638 E372 09 DEX
00639 E373 BC A004 CPX ENDA
00640 E376 26 C1 BNE PUND10 NO
00641 E378 86 47 LDA A #16 PUNCH G
00642 ♦
00643 ♦♦♦SUBROUTINE TO PUNCH DATA BYTE♦♦♦
00644 ♦
00645 E37A 37 DUTCH PSH B SAVE B
00646 E37B F6 8008 DUTC1 LDA B ACIAS IS DATA READY YET?
00647 E37E 57 ASR B

```

```

00648 E37F 57          ASR B
00649 E380 24 F9      BCC     OUTC1      XMIT NOT READY YET
00650 E382 B7 8009    STA A   ACIAD      OUTPUT ONE CHAR
00651 E385 33        PUL B           RESTORE B
00652 E386 39        RTS
00653
00654          *
00655          * SUB TO PUNCH ONE BYTE PTED TO BY XREG.
00656          * ALSO INCREMENTS XREG BEFORE RETURN
00657          *
00657 E387 A6 00    PUN      LDA A   X           GET DATA
00658 E389 8D EF          BSR      OUTCH      PUNCH IT
00659 E38B 08          INX           UPDATE ADDR
00660 E38C 39        RTS
00661
00662          *
00663          *◆◆◆PUNCH LEADER◆◆◆
00664          *
00664 E38D 86 FF    PNLDR   LDA A   #$FF      OUTPUT ALL ONES
00665 E38F 8D E9          BSR      OUTCH      OUTPUT
00666 E391 09          DEX           DECREMENT COUNTER
00667 E392 26 F9          BNE      PNLDR      IF NOT DONE THEN LOOP
00668 E394 39        RTS
00669
00670          *
00671          *◆◆◆◆◆SUBROUTINE TO LOAD DATA FROM CASSETTE TAPE◆◆◆◆◆
00672          *
00673          *
00674 E395 86 10    LOAD     LDA A   #200010000  DIVIDE BY ONE
00675 E397 B7 8008    STA A   ACIAS
00676 E39A 8D 24    BILD     BSR      INCHR
00677 E39C 81 42          CMP A   #'B      START OF BINARY?
00678 E39E 27 05          BEQ      RDBLCK   YES
00679 E3A0 81 47          CMP A   #'G      END OF FILE?
00680 E3A2 26 F6          BNE      BILD
00681 E3A4 39        RTS           YES
00682 E3A5 8D 19    RDBLCK   BSR      INCHR      GET BYTE COUNT
00683 E3A7 16          TAB           PUT IN B
00684 E3A8 5C          INC B          ADJUST IT
00685 E3A9 8D 15          BSR      INCHR      GET START ADDR HI
00686 E3AB B7 A002    STA A   BEGA
00687 E3AE 8D 10          BSR      INCHR      GET START ADDR LO
00688 E3B0 B7 A003    STA A   BEGA+1
00689 E3B3 FE A002    LDX      BEGA      ADDR TO X REG
00690 E3B6 8D 08    STBLCK   BSR      INCHR      NOT DONE
00691 E3B8 A7 00          STA A   X          STRE IT
00692 E3BA 08          INX           INC ADDR
00693 E3BB 5A          DEC B          DEC BYTE COUNT
00694 E3BC 26 F8          BNE      STBLCK   NOT DONE
00695 E3BE 20 DA          BRA      BILD
00696
00697          *
00698          *◆◆◆◆◆INPUT ONE CHR TO A REG◆◆◆◆◆
00699          *
00699 E3C0 B6 8008    INCHR   LDA A   ACIAS
00700 E3C3 47          ASR A
00701 E3C4 24 FA      BCC      INCHR      DATA READY?

```



```

00702 E3C6 B6 8009      LDA A ACIAD      INPUT CHAR
00703 E3C9 39          RTS
00704                  ◆
00705                  ◆◆◆◆SEVEN SEGMENT PATTERNS - USED BY OUTDS◆◆◆◆
00706                  ◆
00707 E3CA 40          DIGTBL FCB      0 1 2 3 4 5 6 7
                                $40,$79,$24,$30,$19,$12,$02,$78
                                E3CB 79
                                E3CC 24
                                E3CD 30
                                E3CE 19
                                E3CF 12
                                E3D0 02
                                E3D1 78
00708                  ◆
00709 E3D2 00          FCB      8 9 A B C D E F
                                $00,$18,$08,$03,$46,$21,$06,$0E
                                E3D3 18
                                E3D4 08
                                E3D5 03
                                E3D6 46
                                E3D7 21
                                E3D8 06
                                E3D9 0E
00710                  ◆
00711 E3DA BF          FCB      - BLANK
                                $BF,$7F
                                E3DB 7F
00712                  ◆◆◆◆KEY VALUE LOOKUP TABLE - USED BY KEYDC
00713                  ◆
00714 E3DC 01          KEYTBL FCB      0 1 2 3 4 5 6 7
                                $01,$02,$42,$82,$04,$44,$84,$08
                                E3DD 02
                                E3DE 42
                                E3DF 82
                                E3E0 04
                                E3E1 44
                                E3E2 84
                                E3E3 08
00715                  ◆
00716 E3E4 48          FCB      8 9 A B C D E F
                                $48,$88,$C8,$04,$C2,$C1,$81,$41
                                E3E5 88
                                E3E6 C8
                                E3E7 C4
                                E3E8 C2
                                E3E9 C1
                                E3EA 81
                                E3EB 41
00717                  ◆
00718 E3EC 10          FCB      P L N V M E R G
                                $10,$50,$90,$D0,$20,$60,$A0,$E0
                                E3ED 50
                                E3EE 90
                                E3EF D0
                                E3F0 20
                                E3F1 60
                                E3F2 A0
                                E3F3 E0
00719                  ◆

```

```

00720          ♦♦♦♦♦KEYBOARD/DISPLAY REGISTER ASSIGNMENT
00721          ♦
00722      8020  DISREG EQU    $8020    DISPLAY SEGMENTS REGISTER
00723      8021  DISCTR EQU    $8021    DISPLAY SEGMENTS CONTROL
00724      8022  SCNREG EQU    $8022    KEYBOARD/DISPLAY SCAN REG
00725      8023  SCNCTR EQU    $8023    KEYBOARD/DISPLAY SCAN CTR
00726      8008  ACIAS  EQU    $8008    ACIA CTRL OR STATUS REG
00727      8009  ACIAD  EQU    $8009    ACIA XMIT OR RCV REGS
00728          ♦
00729          ♦♦♦♦INTERRUPT VECTORS♦♦♦♦
00730          ♦
00731  E3F8          ORG    $E3F8
00732  E3F8 E014    FDB    ID          IRQ INTERRUPT VECTOR
00733  E3FA E032    FDB    SWIR        SOFTWARE INTERRUPT VECTOR
00734  E3FC E019    FDB    NONMSK      NMI INTERRUPT VECTOR
00735  E3FE E08D    FDB    RESTAR      RESTART INTERRUPT VECTOR

```

```

00737      *
00738      ******VARIABLE PARAMETERS*****
00739      *   SYSTEM RAM
00740      *
00741      *
00742      * CAUTION: IF THE USER MODIFY'S THIS PROGRAM
00743      * (GENERATES HIS OWN PROM) THE ORDER OF SOME
00744      * OF THE FOLLOWING VARIABLES IS CRITICAL FOR
00745      * CORRECT OPERATION
00746      *
00747  A000      ORG      $A000
00748      * THE USER CAN STORE THE ADDRESS OF HIS IRQ
00749      * ROUTINE HERE.
00750  A000 0002  IDV      RMB      2      IRQ INTERRUPT POINTER
00751  A002 0002  BEGA     RMB      2      PUNCH BEGINNING ADDRESS
00752  A004 0002  ENDA     RMB      2      PUNCH ENDING ADDRESS
00753  A006 0002  NID      RMB      2      NMI INTERRUPT POINTER
00754  A008 0002  SP       RMB      2      TEMP STACK BUFFER
00755  A00A 0002  TEMP1    RMB      2      SCRATCH
00756  A00C 0008  DISBUF   RMB      8      DISPLAY BUFFER
00757  A014 0001  DIGIN4   RMB      1      4 DIGITS ENTERED FLAG
00758  A015 0001  DIGIN8   RMB      1      8 DIGITS ENTERED FLAG
00759  A016 0001  MFLAG    RMB      1      MEMORY CHANGE MODE FLAG
00760  A017 0001  RFLAG    RMB      1      REGISTER DISPLAY MODE FLAG
00761  A018 0001  NFLAG    RMB      1      TRACE MODE FLAG
00762  A019 0001  TEMP2    RMB      1      COUNTER IN REG DISPLAY, AUDIO
00763  A01A 0002  XKEYBF   RMB      2      NEXT LOC IN DISPLAY BUFFER
00764  A01C 0001  SCNCNT   RMB      1      KEYBOARD /DISPLAY SCAN COUNT
00765  A01D 0001  VFLAG    RMB      1      CONTAINS THE NBR OF ACTIVE BR
00766  A01E 0002  BPADR    RMB      2      TEMP ADDR OF BP&XREG TEMP
00767  A020 0002  XDSBUF   RMB      2      XREG TEMP LOCATION
00768      *
00769      * BREAKPOINT AND OPCODE TABLE
00770      *   EACH BRKPT REQUIRES 3 BYTES,
00771      *   BYTES 1,2 ARE THE ADDRESS OF THE BRKPT
00772      *   BYTE 3 IS THE REPLACED OP CODE
00773      *   CHECK VFLAG TO SEE HOW MANY OF THE BRKPTS
00774      *   ARE VALID
00775      *
00776  A022 000F  BPTAB     RMB      15      BREAKPOINT & OP CODE TABLE
00777      END

```

ID	E014	KEYCL2	E148	KEYD3F	E266	DIGTBL	E3CA
NONMSK	E019	OUTDS3	E149	MDIS	E269	KEYTBL	E3DC
TNMI	E023	KEYDC	E14E	MDIS0	E271	DISREG	8020
NONMK1	E02D	KEYDC1	E15B	MDIS1	E27E	DISCTR	8021
SWIR	E032	KEYDC2	E169	MDIS2	E29A	SCNREG	8022
TZONK	E044	KEYDC3	E16D	MINC	E2A4	SCNCTR	8023
GENA	E054	KEYDC4	E17A	REGST	E2C6	ACIAS	8008
TDISP	E058	KEYDC6	E192	REGST0	E2D0	ACIAD	8009
ADD3X	E05E	KEYDC7	E198	REGST1	E2D7	IDV	A000
GETXB	E063	KEYDC5	E1AC	REGST2	E2F0	BEGA	A002
SETBR	E06A	KYDC5	E1AF	REGST3	E307	ENDA	A004
TPI6	E072	JMPTAB	E1B6	REGST4	E311	NID	A006
TZDT	E076	KEYDC8	E1C6	REGST8	E314	SP	A008
DISNMI	E084	KEYDC9	E1CE	REGST5	E31C	TEMP1	A00A
RESTAR	E08D	KEYDCH	E1D4	REGST7	E322	DISBUF	A00C
INIT	E0AC	KEYD0F	E1D7	REGST6	E327	DIGIN4	A014
CLFLG	E0B2	KEYDCA	E1DA	PNCH	E32F	DIGIN8	A015
CLFLG1	E0B6	TRACE	E1DD	PUND10	E339	MFLAG	A016
CLRDS	E0C4	KEYDCB	E1E6	PUND25	E349	RFLAG	A017
CLRDS1	E0CE	KEYDCC	E1F7	PUND30	E360	NFLAG	A018
HDR	E0D7	KEYDCD	E203	OUTCH	E37A	TEMP2	A019
DLY20	E0DD	KEYDCE	E206	OUTC1	E37B	XKEYBF	A01A
DLY1	E0E0	KEYDCG	E20C	PUN	E387	SCNCNT	A01C
BLDX	E0E4	KEYDCF	E20E	PNLDR	E38D	VFLAG	A01D
OUTDS	E0FE	KEYDCJ	E224	LOAD	E395	BPADR	A01E
OUTDS1	E101	TGB	E236	BILD	E39A	XDSBUF	A020
OUTDS2	E10B	TGC	E24D	RDBLCK	E3A5	BPTAB	A022
KEYCL	E12F	KEYD1F	E25B	STBLCK	E3B6		
KEYCL1	E13A	KEYD2F	E261	INCHR	E3C0		

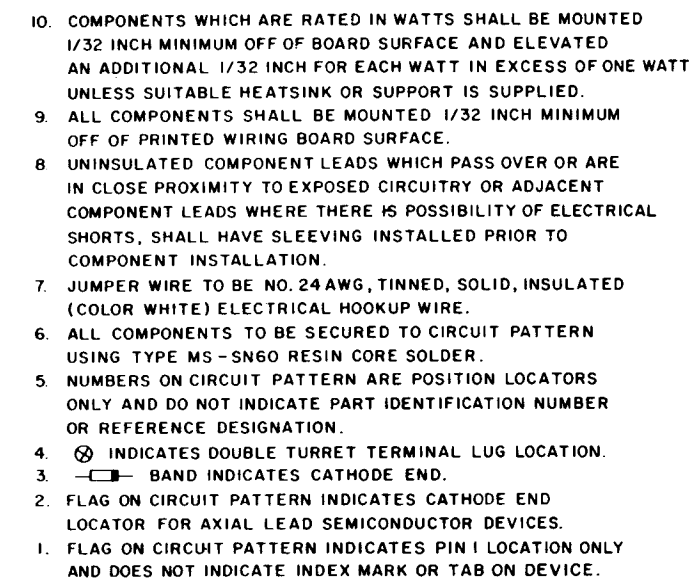
APPENDIX 2

ASSEMBLY DRAWINGS AND PARTS LIST

MEK6800D2 Keyboard/Display Module Parts List				
ITEM	NUMBER REQUIRED	DESCRIPTION	CATALOG NUMBER	DESIGNATION
1	3	Integrated Circuit: Peripheral Driver	MC75452P	U7, U8, U9
2	6	Integrated Circuit: 7Segment LED Display (Litronix or Monsanto)	Litronix DL704 Monsanto MAN72 or 74	U1 — U6
3	1	Integrated Circuit: Dual 4-Channel Data Selector	MC14539BCP	U10
4	1	Integrated Circuit: Dual Monostable Multivibrator	MC14538BCP	U11
5	2	Integrated Circuit: Dual D Flip-Flop	MC14013BCP	U12, U18
6	1	Integrated Circuit: Quad 2-Input AND Gate	MC14081BCP	U13
7	1	Integrated Circuit: Quad Analog Switch	MC14016BCP	U14
8	1	Integrated Circuit: Quad Op-Amp	MC3301P	U16
9	1	Integrated Circuit: Dual Line Receiver	MC75140P1	U17
10	1	Integrated Circuit: Seven Stage Ripple Counter	MC14024BCP	U19
11	1	Integrated Circuit: Analog Multiplexer/Demultiplexer	MC14053BCP	U20
12	7	Transistor, PNP	MPS2907	Q1 — Q7
13	1	Capacitor: 100 μ F, 16 volts		C1
14	14	Capacitor: 0.1 μ F		C2, C5, C9, C10, C14, C16-C23, C25
15	2	Capacitor: 0.05 μ F		C6, C13
16	3	Capacitor: 0.001 μ F		C3, C4, C24
17	3	Capacitor: 0.002 μ F		C7, C8, C15
18	1	Capacitor: 2400 pF Dipped Duramica		C11
19	7	Resistor: 4700 Ω , 1/4 W, 5%		R1, R4, R7, R10
20	29	Resistor: 10 k Ω , 1/4 W, 5%		R13, R16, R19 R2, R5, R8, R11, R14, R17, R20, R22-34, R46, R49, R53, R55, R56, R59, R60, R61, R57
21	7	Resistor: 68 Ω , 1/4 W, 5%		R3, R6, R9, R12, R15, R18, R21
22	2	Resistor: 27 k Ω , 1/4 W, 5%		R35, R40
23	8	Resistor: 100 k Ω , 1/4 W, 5%		R37, R38, R39, R41, R43, R47, R54, R58
24	2	Resistor: 100 Ω , 1/4 W, 5%		R48, R51
25	2	Resistor: 1000 Ω , 1/4 W, 5%		R52, R62
26	2	Resistor: 180 k Ω , 1/4 W, 5%		R36, R42
27	3	Resistor: 22 k Ω , 1/4 W, 5%		R44, R45, R50
28	24	Switch (Stackpole)	LO — PR05	S1 — S24
29	16	Keytops, Double-Shot, Molded, White (Stackpole)	Used with S1 — S24, Item 32	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
30	8	Keytops, Double-Shot, Molded, Blue (Stackpole)		E, G, L, M, N, P, R, V
31	1	Connector Cable		
32	1	Printed Wiring Board		

MEK6800D2 Microcomputer Module Parts List

ITEM	NUMBER REQUIRED	DESCRIPTION	CATALOG NUMBER	DESIGNATION
1	1	Printed Wiring Board		
2	None	Integrated Circuit: 3-State Hex Driver (Optional — Reference only)	MC8T97	U1, U2, U3
3	None	Integrated Circuit: 3-State Transmitter/Receiver (Optional — Reference only)	MC8T26	U4, U5
4	None	Integrated Circuit: 8-Input NAND Gate (Optional — Reference only)	MC7430	U7
5	1	Integrated Circuit: Microprocessing Unit (MPU)	MC6800	U6
6	1	Integrated Circuit: MCM6830 ROM (JBUG)	SCM44520P	U8
7	1	Integrated Circuit: 3-State Hex Driver	MC8T96	U9
8	None	Integrated Circuit: Electrically Programmable ROM (Optional — Reference only)	MCM68708	U10, U12 (Alternate)
9	None	Integrated Circuit: Programmable ROM (Optional — Reference only)	MCM7641	U10, U12 (Alternate)
10	None	Integrated Circuit: Mask Programmed ROM (Optional — Reference only)	MCM68316E	U10, U12 (Alternate)
11	1	Integrated Circuit: One-of-Eight Decoder	MC74155P	U11
12	3	Integrated Circuit: Random Access Memory (RAM) (128x8)	MCM6810	U13, U14, U16 (U18, U19 Optional)
13	1	Integrated Circuit: 614.4 kHz Clock	MC6871B	U15
14	1	Integrated Circuit: 12-Bit Binary Counter	MC14040BCP	U17
15	2	Integrated Circuit: Peripheral Interface Adapter (PIA)	MC6820	U20, U21
16	1	Integrated Circuit: Quad 2-Input NAND Gate	MC7400P	U22
17	1	Integrated Circuit: Asynchronous Communications Interface Adapter (ACIA)	MC6850	U23
18	1	Integrated Circuit: Dual D Flip-Flop	MC7479P	U24
19	1	Integrated Circuit: Binary Counter	MC8316P	U25
20	1	Capacitor: 100 μ F, 16 volt		C1
21	22	Capacitor: 0.1 μ F (Note: Ref. Designations C20 and C21 are not used)		C2 — C19, C22 — C25
22	None	Diode, Zener, 5-volt (Optional — Reference only)	1N4733	CR1
23	1	Transistor, NPN	MPS2222	Q1
24	18	Resistor: 10 k Ω , 1/4 W, 5%		R1, R6-R22
25	3	Resistor: 3300 Ω , 1/4 W, 5%		R2, R3, R4
26	None	Resistor: 68 Ω , 1.0 W, 5% (Optional — Reference only)		R5
27	None	Capacitor: 160 μ F, 16 volt (Optional — Reference only)		C26, C27 R20 — R22
28	10	Socket, 24-Pin (Robinson-Nugent or Equiv)	ICN—246—S4T	
29	3	Socket, 40-Pin (Robinson-Nugent or Equiv)	ICN—406—S4T	
30	1	Switch, Pushbutton (Control)	B8600	Reset
31	1	Cap, Pushbutton Switch (Control)		
32	None	Connector, 86-Pin (SAE) (Optional — Reference only)	SAC 43D/1 — 2	(For P1)
33	None	Connector, Edge, 50-Pin (SAE) (Optional — Reference only)	CPH7000 — 50 ST	(For J1)



A2-3

APPENDIX 3 SCHEMATIC DIAGRAMS

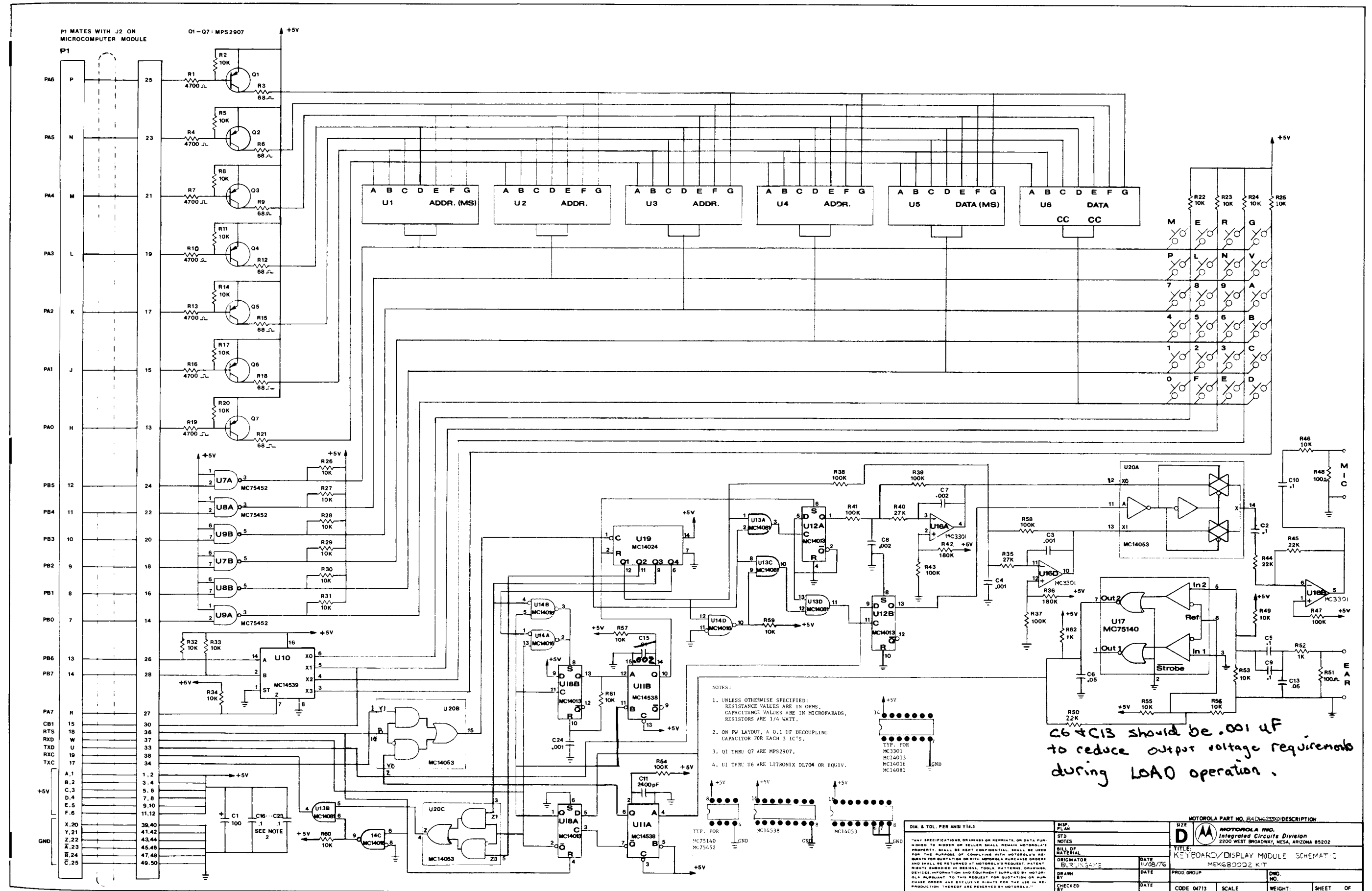


FIGURE A3-a. Keyboard/Display Module Schematic

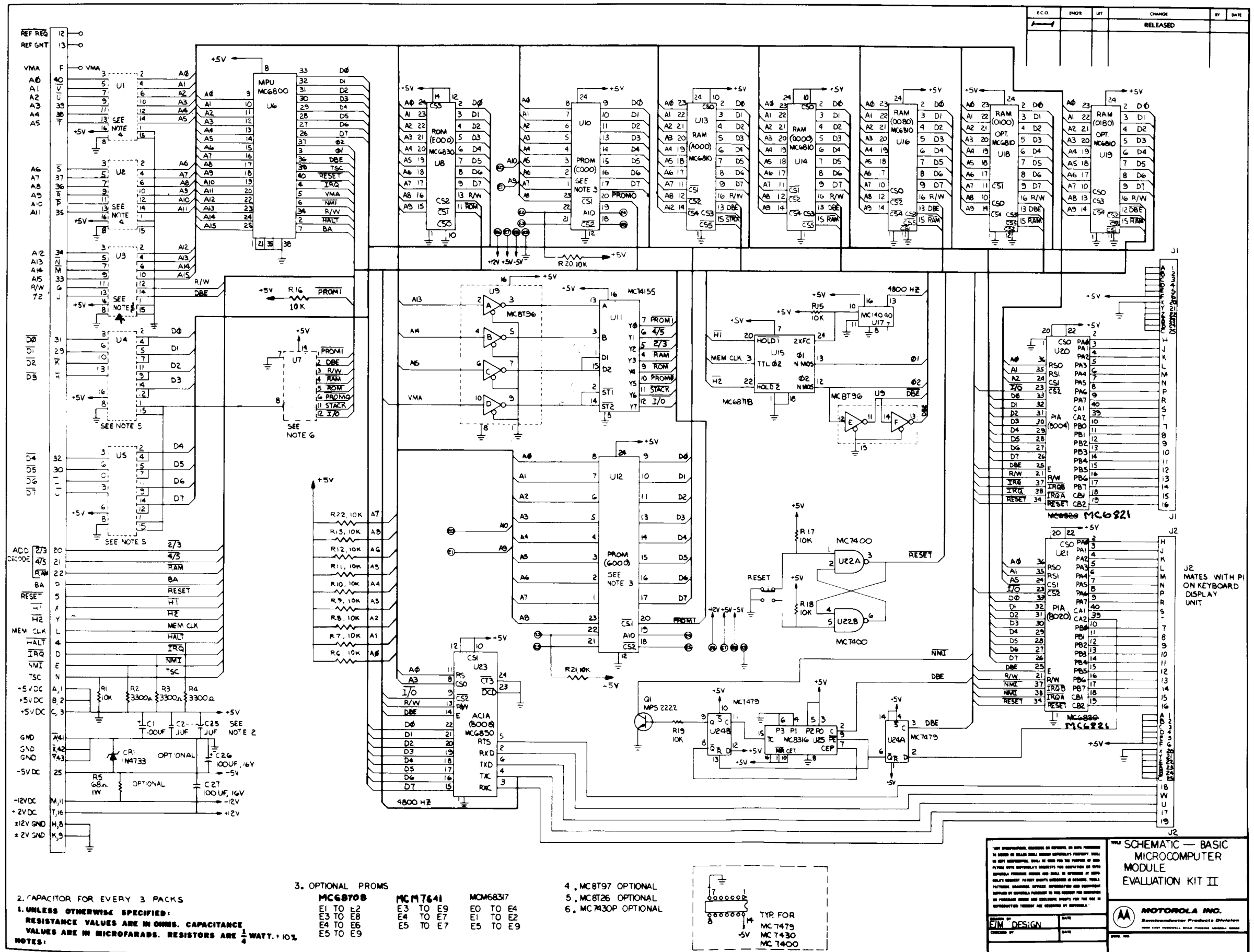
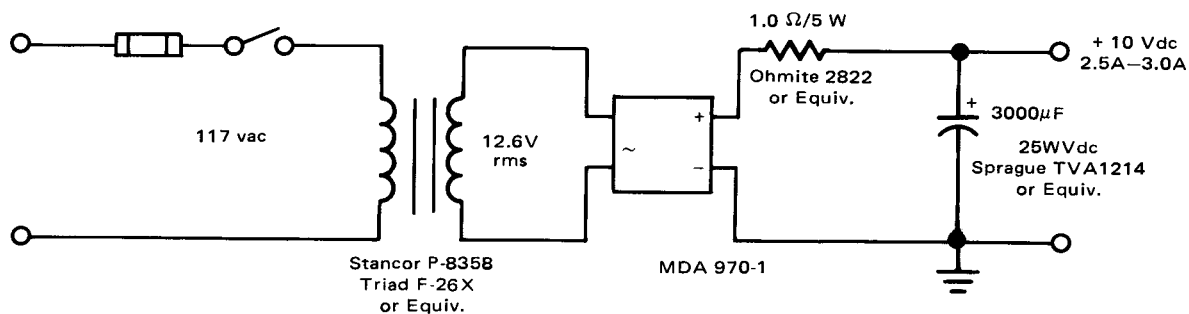


FIGURE A3-b. Microcomputer Module

APPENDIX 4

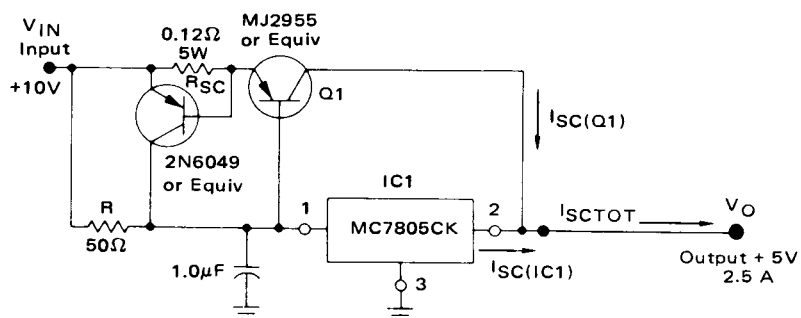
POWER SUPPLY INFORMATION

RECTIFIER ASSEMBLY FOR REGULATED POWER SUPPLY



Note: Ground filter capacitor return lead near negative terminal of rectifier to minimize ground loops.

REGULATOR



R: used to divert IC regulator bias current and determines at what output current level Q1 begins

conducting. $0 < R \leq \frac{V_{BEON}(Q1)}{I_{BIAS}(IC1)}$; $R_{SC} \approx \frac{0.6V}{I_{SC}(Q1)}$; $I_{SCTOT} = I_{SC}(Q1) + I_{SC}(IC1)$

Note: The Regulator Assembly is capable of supplying 5 A with 2.5°C/W and 1°C/W heatsink on IC1 and Q1 respectively ($T_A = 70^{\circ}\text{C}$).

Refer to the Motorola VOLTAGE REGULATOR HANDBOOK for additional information.