M

ALS-8

ALS-8 FIRMWARE MODULE

ASSEMBLY INSTRUCTIONS

Processor Technology

6200 Hollis Street
Emeryville, CA 94608

Phone: (415)652-8080

SECTION I

INTRODUCTION and GENERAL INFORMATION

ALS-8 FIRMWARE MODULE

1.1 INTRODUCTION

This manual supplies the information needed to assemble, test and use the ALS-8 Firmware Module. We suggest that you first scan the entire manual before starting assembly. Then make sure you have all the parts and components listed in the "Parts List" (Table 2-1) in Section II. When assembling the module, follow the instructions in the order given.

Should you encounter any problem during assembly, call on us for help if necessary. If your completed module does not work properly, recheck your assembly step by step. Most problems stem from poor soldering, backward installed components, and/or installing the wrong component. Once you are satisfied that the module is correctly assembled, feel free to ask for our help.

1.2 GENERAL INFORMATION

1.2.1 ALS-8 Firmware Module Description

The ALS-8 Firmware Module is a highly versatile resident assembler that provides "turn-on-the-switch" ability to instantly develop and run programs. Up to six assembly language source programs can be stored in memory as named files and called at will to be listed, edited or assembled by line number. Files can also be stored in any external storage device for later assembly from any input device you select.

Features of the ALS-8 include labels, comments, expressions and constants, along with relative symbolic addressing. Symbolic addressing includes the ability to chain common symbols from one program to another, regardless of when the other program was assembled. The ALS-8 also has the ability to dynamically adjust the system's I/O (input/output) handling configuration under program control. And up to 20 custom commands can be entered and called in exactly the same way as the standard resident commands. In combination, all of these features allow the ALS-8 to be customized in your system to meet your needs.

Additional capabilities can be added to the ALS-8 with the Processor Technology SIM-1 Interpretive Simulator and TXT-2 Text Editing Firmware. The SIM-1--which allows 8080 programs to be simulated on an Altair, IMSAI or Intellec computer--adds a powerful debugging capability. TXT-2 provides full text editing capability to your system: single characters, complete lines and portions of lines can be inserted, deleted and moved.

The ALS-8 is plug-in compatible with the Altair 8800 bus. It requires +7.5 to +10 V dc at 600 mA (max.) operating power. The memory capacity is 5120 bytes in EPROM (hex address EØØØ - F3FF). Worst case access and cycle times are one microsecond.

1.2.2 Receiving Inspection

When your module arrives, examine the shipping container for signs of possible damage to the contents during transit. Then inspect the contents for damage. (We suggest you save the shipping materials for use in returning the module to Processor Technology should it become necessary to do so.) If your ALS-8 kit is damaged, please write to us at once describing the condition so that we can take appropriate action.

1.2.3 Warranty Information

In brief, the parts supplied with the module, as well as the assembled module, are warranted against defects in materials and workmanship for a period of 6 months after the date of purchase. Refer to Appendix I for the complete "Statement of Warranty".

1.2.4 Replacement Parts

Order replacement parts by component nomenclature (e.g., DM8131) and/or a complete description (e.g., 6.8 ohm, ½ watt, 5% resistor).

1.2.5 Factory Service

In addition to in-warranty service, Processor Technology also provides factory repair service on out-of-warranty products. Before returning the product to Processor Technology, first obtain authorization to do so by writing us a letter describing the problem. When you receive our authorization to return the product, proceed as follows:

1. Write a description of the problem.

2. Pack the product with the description in a container suitable to the method of shipment.

3. Ship prepaid to Processor Technology Corporation, 6200 Hollis Street, Emeryville, CA 94608.

The product will be repaired as soon as possible after receipt and return shipped to you prepaid.

SECTION II

ASSEMBLY

ALS-8 FIRMWARE MODULE

2.1 PARTS AND COMPONENTS

Check all parts and components against the "Parts List" (Table 2-1 on Page II-2). If you have difficulty in identifying any parts by sight, refer to Figure 2-1 on Page II-3.

2.2 ASSEMBLY TIPS

1. Scan Sections II and III in their entirety before you start to assemble your ALS-8 Firmware Module.

2. In assembling your ALS-8, you will be following a step-by-step assembly procedure. Follow the instructions in the order given.

3. Assembly steps and component installations are preceded by a set of parentheses. Check off each installation and step as you complete them. This will minimize the chances of omitting a step or component.

4. When installing components, make use of the assembly aids that are incorporated on the ALS-8 PC board and the assembly drawing: (These aids are designed to assist you in correctly installing the components.)

   a. The circuit reference (R3, C10 and IC20, for example) for each component is silk screened on the PC board near the location of its installation.

   b. Both the circuit reference and value or nomenclature (1.5K and 7400, for example) for each component are included on the assembly drawing near the location of its installation.

5. To simplify reading resistor values after installation, install resistors so that the color codes read from left-to-right and top-to-bottom as appropriate (board orientation as defined in Paragraph 2.5).

6. Install disc capacitors as close to the board as possible.

7. Should you encounter any problem during assembly, call on us for help if needed.

2.3 ASSEMBLY PRECAUTIONS

2.3.1 Handling MOS Integrated Circuits

The memory ICs used in the ALS-8 are MOS devices. They can be damaged by static electricity discharge. Always handle MOS ICs so

Table 2-1. ALS-8 Firmware Module Parts List.

INTEGRATED CIRCUITS

1	LM741C (IC29)	2	74LS175 (IC27, 28)
1	74LS00 (IC17)	2	74367,8097 or 8T97 (IC24,25)
3	74LS08 (IC21,22,23)	1	8836 or 8T380 (IC26)
1	74LS136 (IC18)	10	S6834 (IC1 through 10)
2	74LS138 (IC19,20)		

REGULATORS

1	340T-5.0 or 7805UC (IC30)

TRANSISTORS

2	2N2907 (Q1, 2)
1	2N6107 (Q3)

RESISTORS

1	1 ohm, ¼ watt, 5%
1	100 ohm, ¼ watt, 5%
2	2.2 Kohm, ¼ watt, 5% or 1.5 Kohm, ¼ watt, 5%
1	1.69Kohm, ¼ watt, 1% or 1691 ohm, ¼ watt, 1%
1	4.02Kohm, ¼ watt, 1%

CAPACITORS

23	0.1 ufd, disc ceramic
2	1 ufd, tantalum, dipped
3	15 ufd, tantalum, dipped

MISCELLANEOUS

1	ALS-8 PC Board (8KPROM-34)	1	Length #22 Bare Wire
1	Heat Sink	1	Length #22 Insulated Wire
6	14-pin DIP Sockets	3	6-32 Screws
6	16-pin DIP Sockets	1	6-32 Teflon Screw
16	24-pin DIP Sockets	4	6-32 Lockwashers
23	Augat Pins on Carriers	4	6-32 Nuts
1	Mica Insulator (for Q3)	1	Manual
1	Length Solder		

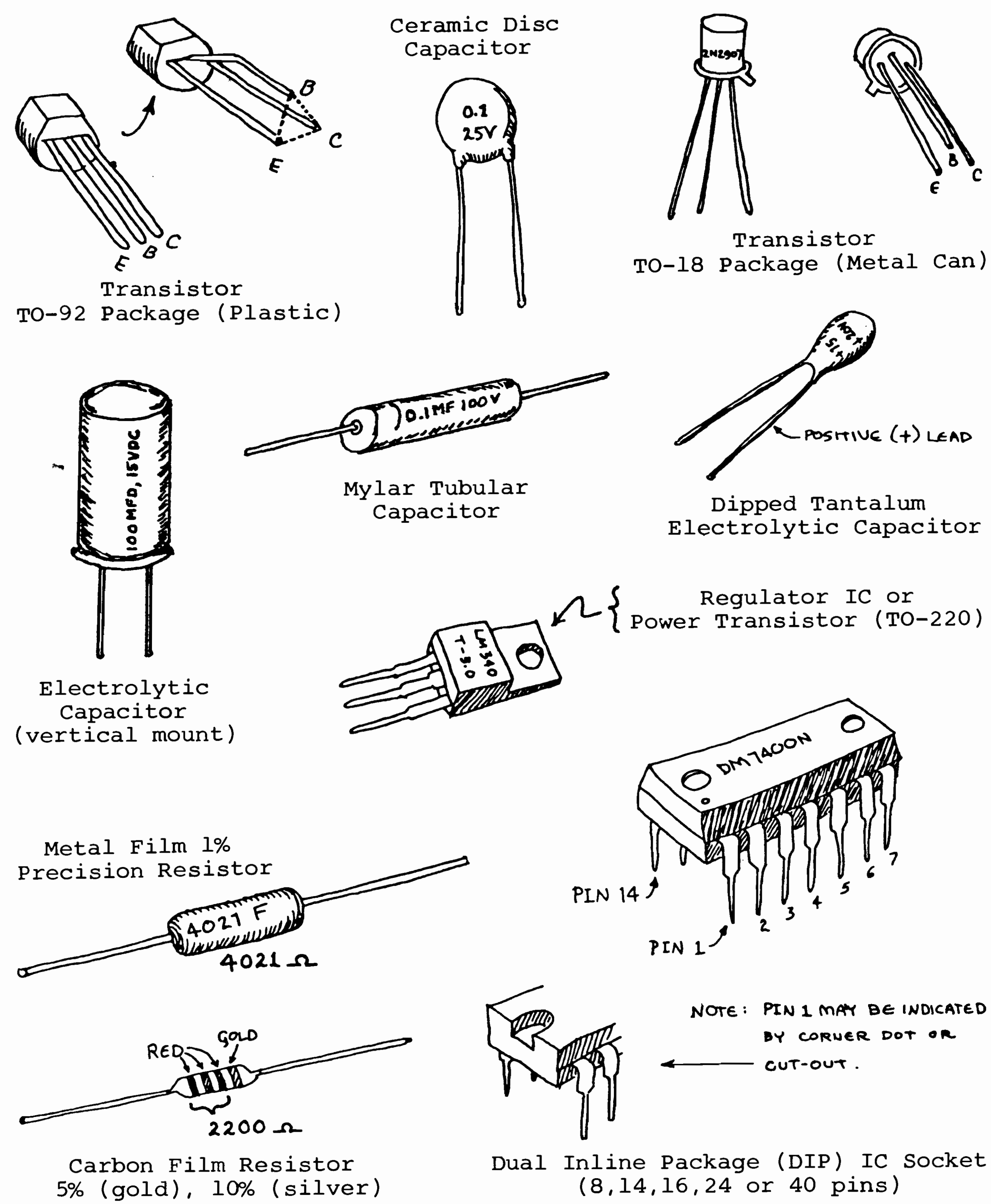


Figure 2-1. Identification of components.

that no discharge will flow through the IC. Also, avoid unnecessary handling and wear cotton--rather than synthetic--clothing when you do handle these ICs.

2.3.2 Soldering **IMPORTANT**

1. Use a low-wattage iron, 25 watts maximum.

2. Solder neatly and quickly as possible.

3. DO NOT press top of iron on pad or trace. This can cause the pad or trace to "lift" off the board and permanently damage it.

4. Use only 60-40 rosin-core solder. NEVER use acid-core solder or externally applied fluxes.

5. The ALS-8 uses a circuit board with plated-through holes. Solder flow through to the component (front) side of the board can produce solder bridges. Check for such bridges after each component is installed.

6. The ALS-8 circuit board has an integral solder mask (a lacquer coating) that shields selected areas on the board. This mask minimizes the chances of creating solder shorts during assembly.

7. Additional pointers on soldering are provided in Appendix III of this manual.

2.3.3 Installing and Removing ALS-8

NEVER install the ALS-8 in, or remove it from, the computer with the power on. To do so can damage the board.

2.3.4 Installing and Removing Integrated Circuits

NEVER install or remove integrated circuits with power applied to the ALS-8.

2.3.5 Use of Clip Leads

NEVER attach clip leads to the top edge of the board when power is applied. To do so will short the +8 V dc, -16 V dc and possibly the +5 V dc busses to one another.

2.4 REQUIRED TOOLS, EQUIPMENT AND MATERIALS

The following tools, equipment and materials are recommended for assembling the ALS-8 Firmware Module:

1. Needle nose pliers

2. Diagonal cutters

3. Controlled heat soldering iron, 25 watts

4. 60-40 rosin-core solder (supplied)

5. Volt-ohm meter

2.5 ORIENTATION

The heat sink area (large foil area) will be located in the upper righthand corner of the board when the edge connector is positioned at the bottom of the board. In this position, the front (component) side of the board is facing up. Subsequent position references assume this orientation.

2.6 ASSEMBLY PROCEDURE

Refer to assembly drawing in Section IV.

CAUTION

THIS DEVICE USES MOS MEMORY INTEGRATED CIRCUITS (IC1 - 10) WHICH CAN BE DAMAGED BY STATIC ELECTRICITY DISCHARGES. HANDLE THESE IC's SO THAT NO DISCHARGE FLOWS THROUGH THE IC. AVOID UNNECESSARY HANDLING AND WEAR COTTON, RATHER THAN SYNTHETIC, CLOTHING WHEN HANDLING THESE IC's. (STATIC CHARGE PROBLEMS ARE MUCH WORSE IN LOW HUMIDITY ENVIRONMENTS.)

( ) Step 1. Check circuit board to insure that the -16-volt bus, +8-volt bus and +5-volt bus are not shorted to ground. Using an ohmmeter, make the following measurements:

( ) -16-volt Bus Test. Measure between edge connector pin 52 (second bottom--or back--pin from left end of the connector) and pin 50 or 100 (right end of connector). There should be no continuity.

( ) +8-volt Bus Test. Measure between edge connector pin 1 or 51 (left end of connector) and pin 50 or 100. There should be no continuity.

( ) +5-volt Bus Test. Measure between positive mounting pad for C14 and pin 50 or 100 of the edge connector. There should be no continuity.

( ) 16-8-5 Volt Bus Tests. Measure between edge connector pins 1 or 51 and 52, between edge connector pins 1 or 51 and the positive mounting pad for C14, and between edge connector pin 52 and the positive mounting pad for C14. There should be no continuity in any of these three measurements.

If you measure continuity in any of the preceding tests, the PC board is defective. Return it to Processor Technology for replacement. If none of the measurements show continuity, proceed to Step 2.

( ) Step 2. Install heat sink. Position the large, black heat sink (flat side to board) over the square foil area in the upper right corner of the PC board. Orient the sink so that the two triangles of mounting holes are under the triangular cut-outs in the sink. Using two 6-32 screws, lockwashers and nuts, attach heat sink to board. Insert screws from back (solder) side of board.

( ) Step 3. Install IC30 (340T-5.0 or 7805UC). Position IC30 on heat sink and observe how the leads must be bent to fit the mounting holes. Note that the center lead (3) must be bent down at a point approximately 0.2 inches further from the body than the other leads. Bend the leads so that no contact is made with the heat sink when IC30 is flat against the sink and its mounting hole is aligned with the hole in the sink. Fasten IC30 to sink using 6-32 screw, lockwasher and nut. Insert screw from back (solder) side of board. Solder and trim the leads.

( ) Step 4. Install Q3 (2N6107). Place mica insulator on heat sink so that its holes are aligned with the mounting holes in the PC board. Position Q3, with its identifying nomenclature up, over the mica insulator and observe how the leads must be bent to fit the mounting holes as well as the holes in the mica insulator. Note that the center lead must be bent down at a point approximately 0.2 inches further from the body than the other leads. Bend the leads so that no contact will be made with the heat sink when Q3 and the mica insulator are flat against the sink when Q3's mounting hole is aligned with the hole in the heat sink and PC board. Place mica insulator between Q3 and the heat sink and fasten Q3 to the sink using the 6-32 Teflon screw and a 6-32 lockwasher and nut. Insert screw from back (solder) side of board. Solder and trim the leads.

( ) Step 5. Install Q1 and Q2 (2N2907) in their indicated locations. The emitter lead on each (lead closest to tab on can) is oriented to the top and the base lead is oriented toward the left. Start leads into mounting holes and push straight down on transistor until it is stopped by the leads. Solder and trim.

( ) Step 6. Install the five tantalum capacitors in the following locations. Take care to observe the proper values and orientations.

LOCATION	VALUE (ufd)	ORIENTATION
( ) C5	1	"+" lead top
( ) C6	15	"+" lead right
( ) C11	15	"+" lead top
( ) C13	15	"+" lead top
( ) C14	1	"+" lead left

Check capacitors for correct value and orientation, bend leads outward on solder (back) side of board, solder and trim.

( ) Step 7. Install all disc capacitors in numerical order in the indicated locations. Insert, pull down snug to board, bend leads outward on solder (back) side of board, solder and trim.

NOTE

Disc capacitor leads are usually coated with wax during the manufacturing process. After inserting leads through mounting holes, remove capacitor and clear the holes of any wax. Reinsert and install.

LOCATION	VALUE (ufd)	TYPE
( ) C1	0.1	Disc Ceramic
( ) C2	0.1	" "
( ) C3	0.1	" "
( ) C4	0.1	" "
( ) C7	0.1	" "
( ) C8	0.1	" "
( ) C9	0.1	" "
( ) C10	0.1	" "
( ) C12	0.1	" "
( ) C15	0.1	" "
( ) C16	0.1	" "
( ) C17	0.1	" "
( ) C18	0.1	" "
( ) C19	0.1	" "
( ) C20	0.1	" "
( ) C21	0.1	" "
( ) C22	0.1	" "
( ) C23	0.1	" "

(Continued on Page II-8.)

( ) Step 7. (Continued)

LOCATION	VALUE (ufd)	TYPE
( ) C24	0.1	Disc Ceramic
( ) C25	0.1	" "
( ) C26	0.1	" "
( ) C27	0.1	" "
( ) C28	0.1	" "

( ) Step 8. Install all resistors in numerical order in the indicated locations. Bend leads to fit distance between the mounting holes, insert, pull down snug to board, bend leads outward on solder (back) side of board, solder and trim.

LOCATION	VALUE (ohms)	COLOR CODE
( ) R1	1	brown-black-gold
( ) R2	4.02K	metal film
( ) R3	1.69K (or 1691)	metal film
( ) R4	100	brown-black-brown
( ) R5	2.2K (or 1.5K)	red-red-red*
( ) R6	2.2K (or 1.5K)	red-red-red*

*brown-green-red if 1.5K ohm

( ) Step 9. Install Augat pins as follows:

NOTE

You will find it helpful to hold the board between two objects so that it stands on one edge.

( ) Area A. Cut one Augat pin carrier across the short dimension to obtain 12-pin carrier (6 pins per side). With the pins still attached, insert them in the 12 mounting holes in Area A from the component (front) side of board. Solder pins from solder (back) side of board so that the solder "wicks up" to the front side. (This will hold the pins firmly in place.) Remove the carrier.

( ) Area B. Remove three pins from carrier and insert them into mounting holes L, C and R in Area B from front (component) side of board. Solder pins from back (solder) side of board as you did the Area A pins. Then insert a component lead into one pin and reheat solder. Use lead to adjust pin until it is perpendicular to board.

Allow solder to cool while holding the pin as steady as possible. Repeat this procedure with the other two pins.

( ) Step 9. (Continued)

NOTE

If the cooled solder is mottled or crystalized, a "cold joint" is indicated, and the solder should be reheated.

Check each installation in Area B for cold joints and solder bridges

( ) Area D. Remove six pins from carrier and install them in the six mounting holes in Area D. Use the same technique as you used to install the pins in Area B. Check for cold joints and solder bridges.

NOTE

There is no Area C on the 8KPROM-34 PC board.

( ) Area E. Remove remaining two pins from carrier and install them in Area E. Use the same technique you used to install the pins in Area B. Check for cold joints and solder bridges.

( ) Step 10. Fill feed-through hole located in heat sink foil to the right and below C13 with solder.

( ) Step 11. Using #22 bare wire, install jumpers in Areas A, B, D and E according to your selection of the options that are described in Section III.

( ) Step 12. Install DIP sockets. Install each socket in the indicated location with its end notch oriented as shown on the PC board and assembly drawing. Take care not to create solder bridges between the pins and/or traces.

LOCATION	TYPE SOCKET
( ) IC1 through 16	24 pin
( ) IC17 and 18	14 pin
( ) IC19 and 20	16 pin
( ) IC21, 22 and 23	14 pin
( ) IC24 and 25	16 pin
( ) IC26	14 pin
( ) IC27 and 28	16 pin

( ) Step 13. Install IC29 (LM741C) in the indicated location. Pay careful attention to the proper orientation.

(Step 13 continued on Page II-10.)

( ) Step 13. (Continued)

NOTE

A dot on the assembly drawing indicates the location of pin 1 of the IC.

Check for proper orientation and load IC29 (refer to "Loading DIP Devices" in Appendix III). Avoid creating solder bridges between pins and/or traces.

( ) Step 14. Check regulator operation. This check is made to prevent potential damage to the IC's from incorrect voltages.

( ) Install ALS-8 in computer. (The use of a Processor Technology EXB Extender Board is recommended.)

CAUTION

NEVER INSTALL OR REMOVE ALS-8 WITH POWER ON. TO DO SO CAN DAMAGE THE MODULE.

( ) Turn power on and make the following voltage measurements:

MEASUREMENT POINT	VOLTAGE
Positive lead of C14	+5 V dc $\pm$5%
Negative lead of C5	-12 V dc $\pm$5%

( ) If either voltage is incorrect, determine and correct the cause before proceeding. Especially check for solder shorts.

If voltages are correct, go on to Step 15.

( ) Step 15. Install the following IC's in the indicated locations. Pay careful attention to the proper orientation.

NOTE

Pin 1 is indicated by a dot on the PC board and assembly drawing.

IC NO.	TYPE
( ) IC17	74LS00
( ) IC18	74LS136
( ) IC19	74LS138
( ) IC20	74LS138
( ) IC21	74LS08
( ) IC22	74LS08

(Continued on Page II-11.)

( ) Step 15. (Continued)

IC NO.	TYPE
( ) IC23	74LS08
( ) IC24	74367, 8097 or 8T97
( ) IC25	74367, 8097 or 8T97
( ) IC26	8836 or 8T380
( ) IC27	74LS175
( ) IC28	74LS175

( ) Step 16. Install IC1 through IC10 in numerical order in their respective locations. Pay careful attention to the proper orientation.

IC1 through IC10 (Type S6834) are MOS devices. Refer to the CAUTION on Page II-5.

NOTE

IC11 through IC16 are not supplied with the basic ALS-8. They are supplied, three each, with the SIM-1 and TXT-2 options.

If you did not receive the SIM-1 or TXT-2 options with your ALS-8, skip Step 17 and proceed to Step 18.

( ) Step 17. If you received the SIM-1 or TXT-2 options, install the following IC's in their indicated locations. Pay careful attention to the proper orientation.

OPTION	INSTALL	IC TYPE
SIM-1	IC11, 12 and 13	S6834
TXT-2	IC14, 15 and 16	S6834

IC11 through IC16 are MOS devices. Refer to the CAUTION on Page II-5.

( ) Step 18. Your ALS-8 is now ready to use for developing and running programs.

SECTION III

OPTION SELECTION

ALS-8 FIRMWARE MODULE

3.1 START-UP/RUN OPTIONS

There are two ways to start up and run the ALS-8 program, manual and semi-automatic.

1. Manual Operation. Manual operation is performed by examining location EØØØ (hex) from the computer front panel and flipping the computer RUN switch.

2. Semi-automatic Operation. Semi-automatic operation is performed by simply flipping the computer RUN switch immediately after power is turned on.

Normally, semi-automatic operation is preferred for its simplicity and operator convenience. Hardware modifications in some systems, however, may be needed. If you select the semi-automatic option, follow the instructions in Paragraph 3.3.

If you select the manual option, or if the ALS-8 is to be used in the Processor Technology Sol Terminal Computer System, follow the instructions in Paragraph 3.2.

NOTE

> The Processor Technology Sol Terminal Computer System incorporates fully automatic start-up circuitry. If you intend to use your ALS-8 in a Sol system, connect the ALS-8 for manual start-up as described in Paragraph 3.2.

3.2 MANUAL OPERATION

3.2.1 ALS-8 Jumper Selection

Use the following jumper selection instructions in conjunction with the illustrations provided and the assembly drawing in Section IV.

1. Address Location (Area A)

   The jumper arrangement in Area A determines the address location for the ALS-8.

   Select address location EØØØ to FFFF (hex) by installing jumpers (#22 bare wire is recommended) between A13 and Ø, A14 and Ø and A15 and Ø pins in Area A as shown in Figure 3-1.

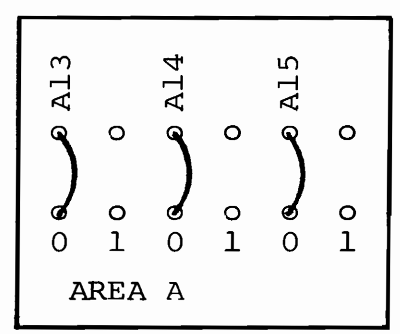


Figure 3-1. Area A jumpers, manual operation.

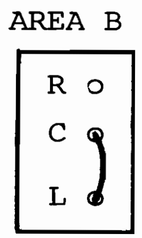


Figure 3-2. Area B jumper, manual operation.

2. PRESET (Area B)

The jumper arrangement in Area B determines whether the power-up PRESET connection on the ALS-8 is enabled or disabled.

Disable power-up PRESET by installing a jumper (#22 bare wire is recommended) between the C and L pins in Area B as shown in Figure 3-2.

NOTE

There is no Area C on the 8KPROM-34 PC board.

3. Wait States (Area D)

The jumper arrangement in Area D determines the number of wait states (1,2,3 or 4).

In an 8080 system operating at 2MHz clock frequency, the ALS-8 should normally be jumpered for one wait state. Select one wait state by installing a jumper (#22 bare wire is recommended) between the 1 and W pins in Area D as shown in Figure 3-3.

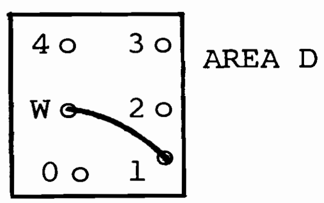


Figure 3-3. Area D jumper, manual or semiautomatic operation.

4. PHANTOM (Area E)

DO NOT install a jumper in Area E.

3.2.2 Operating Procedure

The ALS-8 requires at least 2048 bytes of unprotected read/write (RAM) memory at locations DØØØ - D7FF (hex). 4096 bytes from DØØØ to DFFF (hex), however, are recommended.

To manually initialize and run the ALS-8 program:

1. Examine EØØØ (hex) from computer front panel.

2. Flip the computer RUN switch.

3.3 SEMI-AUTOMATIC OPERATION

3.3.1 ALS-8 Jumper Selection

Use the following jumper selection instructions in conjunction with the illustrations provided and the assembly drawing in Section IV.

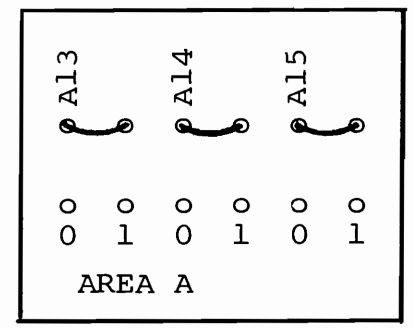


Figure 3-4. Area A jumpers, semi-automatic operation.

1. Address Location (Area A)

The jumper arrangement in Area A determines the address location for the ALS-8.

Select address location EØØØ to FFFF (hex) with PHANTOM recognition by installing jumpers in Area A (#22 bare wire is recommended) as shown in Figure 3-4.

2. PRESET (Area B)

The jumper arrangement in Area B determines whether the power-up PRESET connection on the ALS-8 is enabled or disabled.

If your ALS-8 is to be used in a Processor Technology Sol Terminal Computer System or an IMSAI 8080, disable power-up PRESET by installing a jumper (#22 bare wire is recommended) between the C and L pins in Area B. (See Figure 3-5(a).)

If your ALS-8 is to be used in an Altair 8800, enable power-up PRESET by installing a jumper (#22 bare wire is recommended) between the C and R pins in Area B. (See Figure 3-5(b).) With this jumper installed, a power on clear (POC) pulls PRESET low.

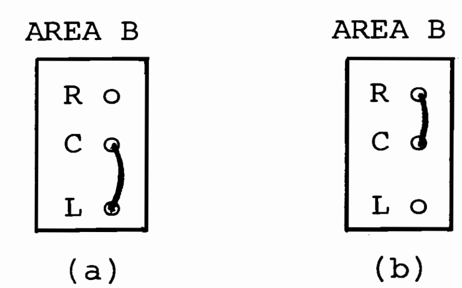


Figure 3-5. Area B jumper, semi-automatic operation.

NOTE

There is no Area C on the 8KPROM-34 PC board.

3. Wait States (Area D)

The jumper arrangement in Area D determines the number of wait states (1, 2, 3 or 4).

In an 8080 system operating at a 2MHz clock frequency, the ALS-8 should normally be jumpered for one wait state. Select one wait state by installing a jumper (#22 bare wire is recommended) between the 1 and W pins in Area D. (See Figure 3-3 on Page III-2.)

4. PHANTOM (Area E)

The Area E jumper arrangement determines whether the PHANTOM output from the ALS-8 on Bus Pin 67 is enabled or disabled.

Enable PHANTOM by installing a jumper (#22 bare wire is recommended) between the two pins in Area E as shown in Figure 3-6.

AREA E

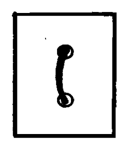


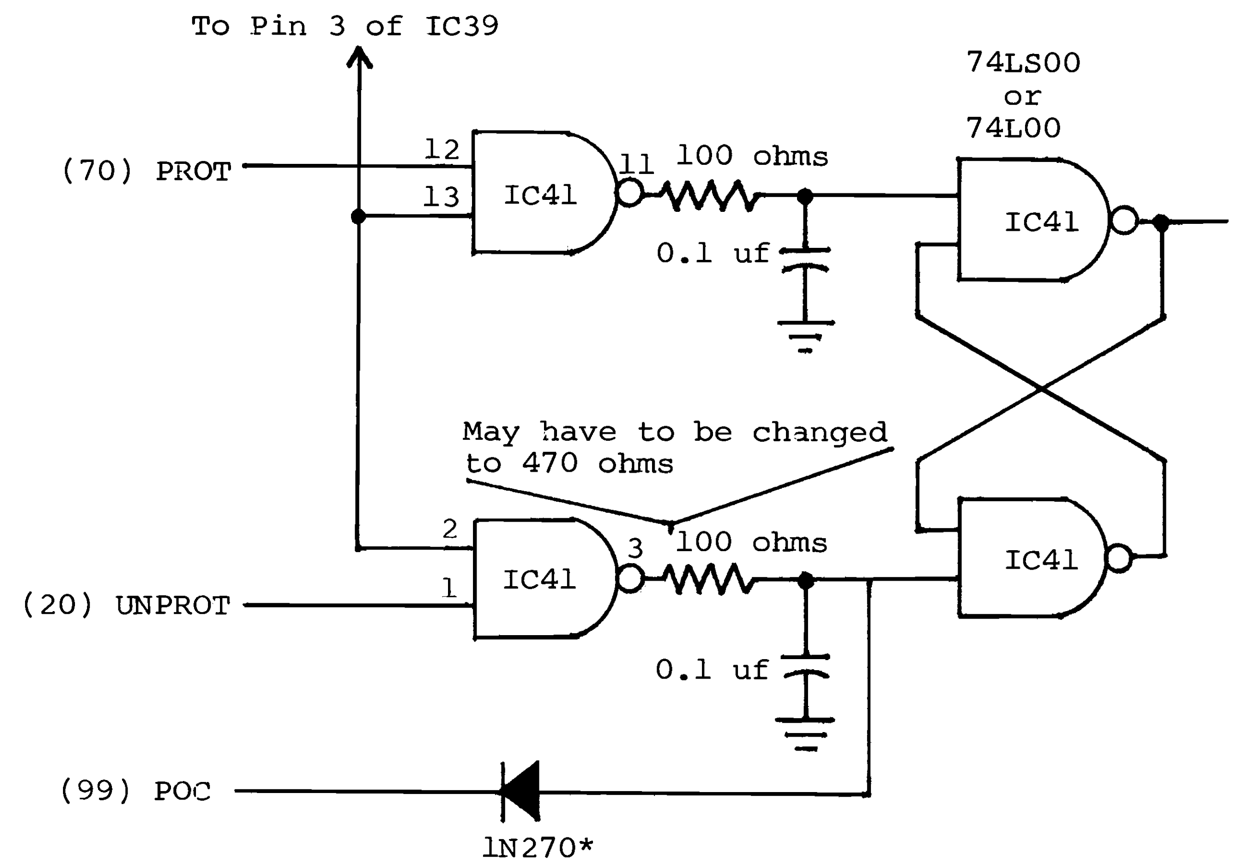
Figure 3-6. Area E jumper, semi-automatic operation.

3.3.2 Hardware Modifications

RAM Memory. The ALS-8 requires at least 2048 bytes of read/write (RAM) memory at locations DØØØ - D7FF (hex). 4096 bytes from DØØØ to DFFF (hex), however, are recommended. This memory must come up unprotected immediately after power is turned on. And if it is to be used at address zero, it must be capable of recognizing the PHANTOM signal supplied by the ALS-8 on Bus Pin 67.

Recent Processor Technology 4KRA and all of our 8KRA Static Read/Write Memory Modules incorporate power-up initialization circuitry to determine whether they come up in the protected or unprotected mode. Earlier 4KRA memories (prior to Revision E) can be modified as shown in Figure 3-7 to come up unprotected.

Later versions of Processor Technology 4KRA and all of our 8KRA memories incorporate the PHANTOM disable capability. Instructions for enabling this capability are provided in the 4KRA and 8KRA Assembly and Test Instructions.



*Add 1N270 germanium diode to IC41 circuit (lower left corner of 4KRA PC board) as shown to enable memory to come up unprotected after power is turned on.

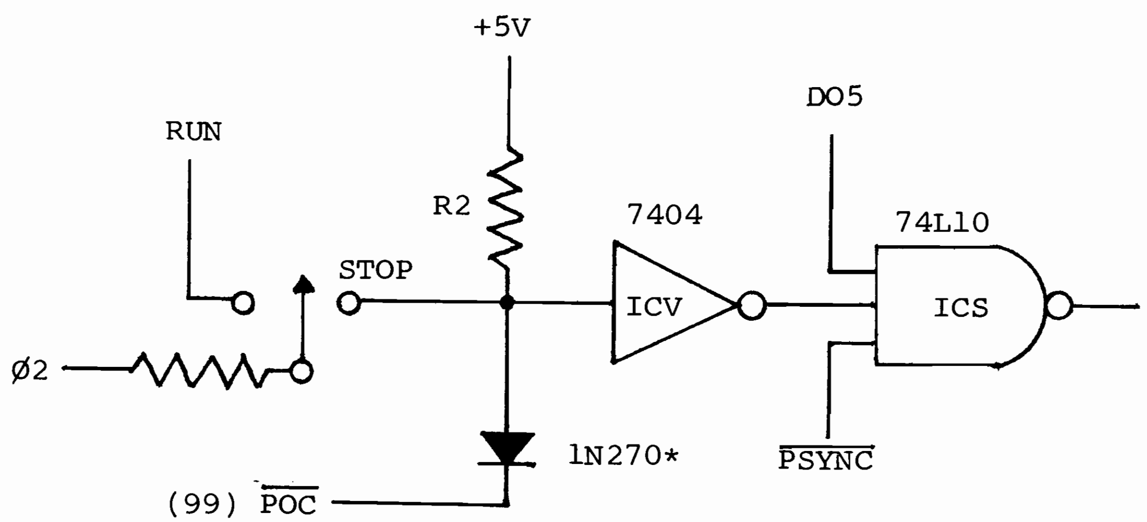
NOTE: REVISION E AND ABOVE 4KRA MEMORY BOARDS DO NOT REQUIRE THIS MODIFICATION.

Figure 3-7. Power on unprotected mode modification for early (before Revision E) 4KRA memories.

Computer. The computer must come up in the stopped state immediately after power is turned on. The Altair 8800 front panel can be modified as shown in Figure 3-8 to meet this requirement. Figure 3-9 shows how to modify the IMSAI 8080 front panel for the same purpose.

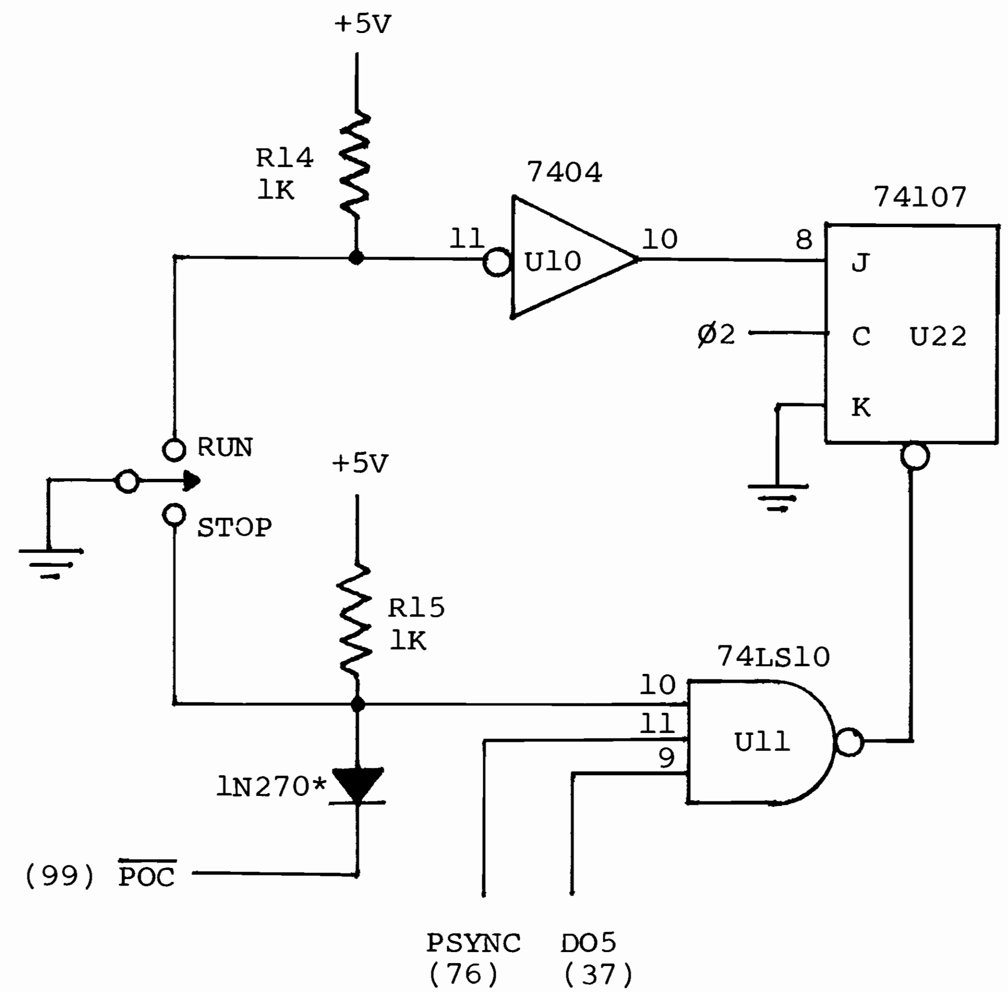
3.3.3 Operating Procedure

To initialize and run the ALS-8 program in the semi-automatic mode, simply flip the computer RUN switch.



*Add 1N270 germanium diode (DO NOT SUBSTITUTE) to front panel circuit as shown to insure computer comes up in stopped state after power is turned on.

Figure 3-8. Altair 8800 front panel modification.



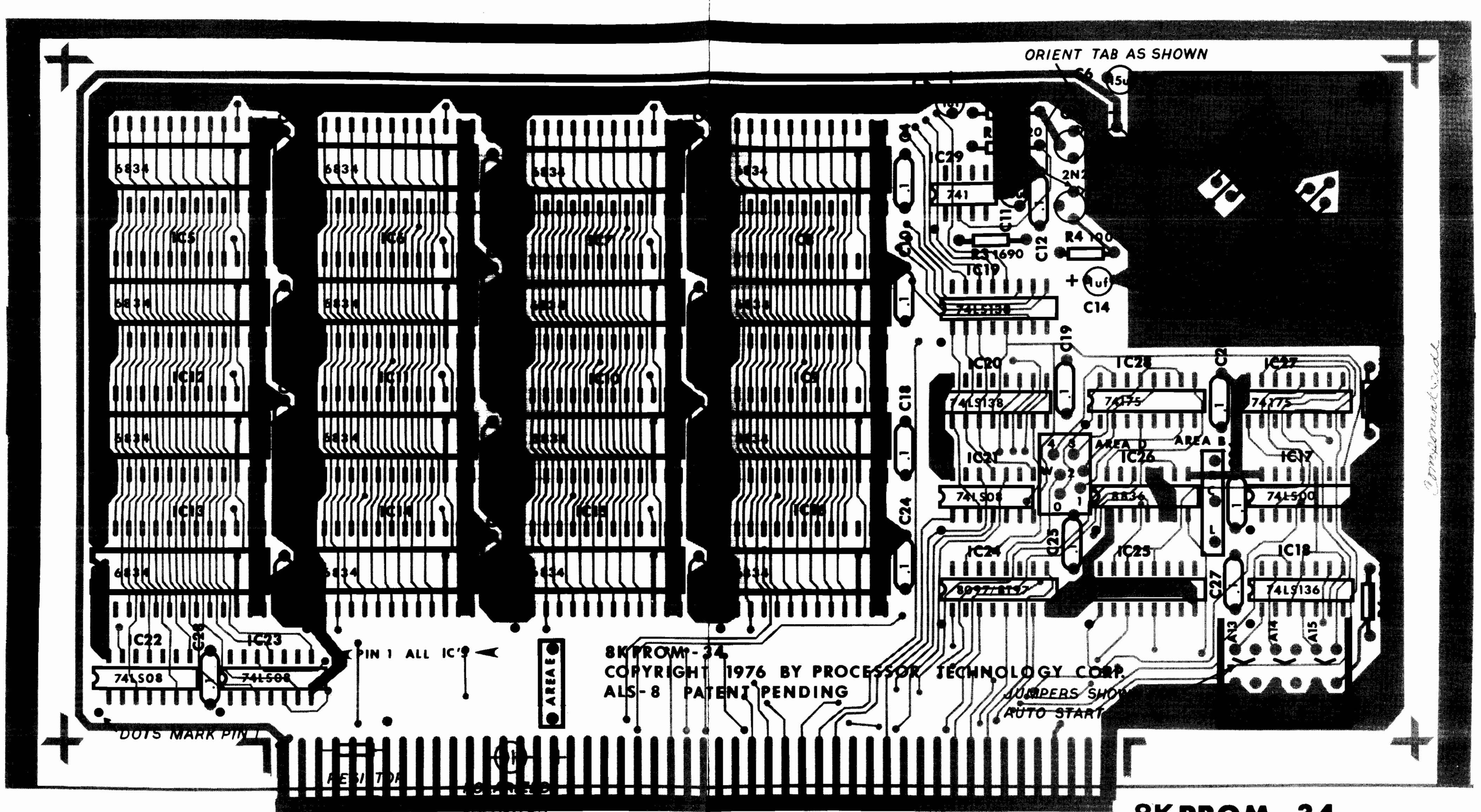
*Add 1N270 germanium diode (DO NOT SUBSTITUTE) to front panel circuit as shown to insure computer comes up in stopped state after power is turned on.

Figure 3-9. IMSAI 8080 front panel modification.

SECTION IV

DRAWINGS

ALS-8 FIRMWARE MODULE



DISK
CAPACITOR

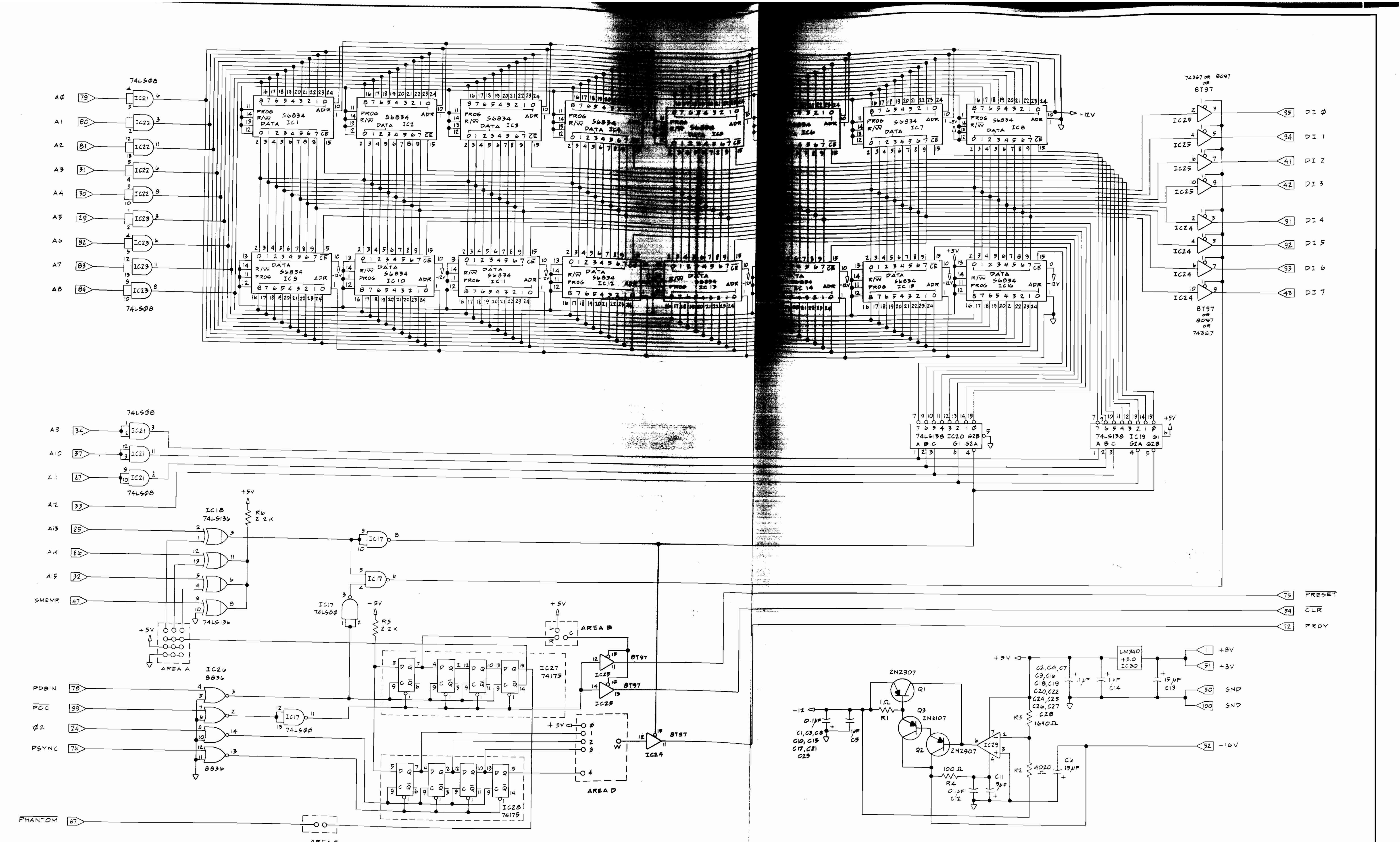
8KPROM - 34

foil

	front	back	both
sheet	5		1
rev	A		A

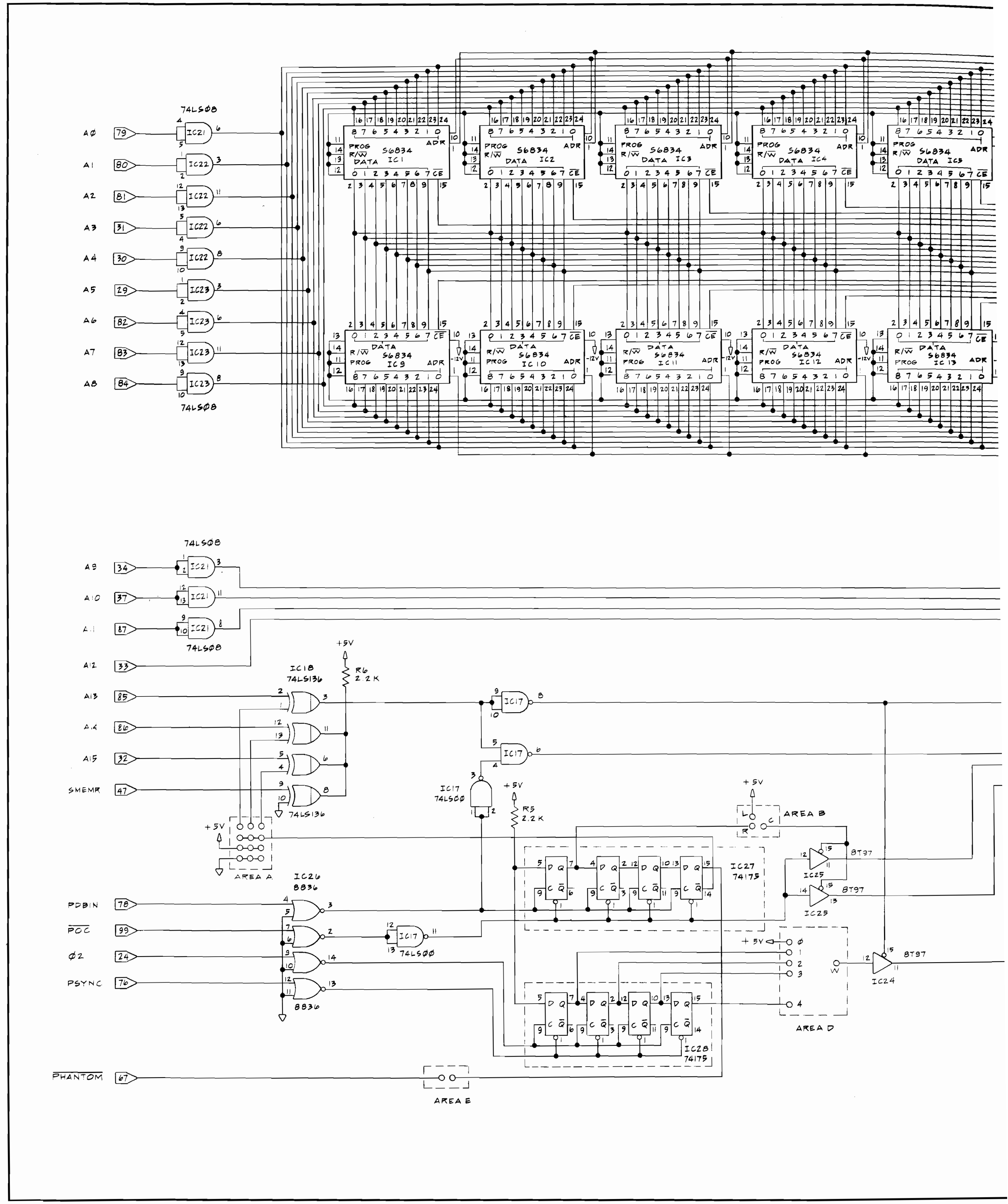
assembly

T.H. Schmidt



8K EROM BOARD/8K PROM-34		
SCALE:	APPROVED BY:	DRAWN BY LITO
DATE: 07-2-76	ROBERT M. MARSH	REVISED 5/76
PROCESSOR TECHNOLOGY		
ALS-8		DRAWING NUMBER

PRINTED ON NO. 1000H-10 CLEARPRINT


74LS08
A0
A1
A2
A3
A4
A5
A6
A7
A8
IC21
IC22
IC23
S6834
PROG
R/W̄
DATA
ADR
CE
IC1
IC2
IC3
IC4
IC5
IC9
IC10
IC11
IC12
IC13
A9
A10
A11
A12
A13
A14
A15
SMEMR
IC18
74LS136
+5V
R6
2.2K
IC17
74LS00
R5
2.2K
AREA A
AREA B
IC27
74175
IC25
8T97
PDBIN
POC
Ø2
PSYNC
IC26
8836
IC24
AREA D
IC28
74175
PHANTOM
AREA E

APPENDICES

Warranty

PROCESSOR TECHNOLOGY CORPORATION, in recognition of its responsibility to provide quality components and adequate instruction for their proper assembly, warrants its products as follows:

All components sold by **Processor Technology Corporation** are purchased through normal factory distribution and any part which fails because of defects in workmanship or material will be replaced at no charge for a period of 3 months for kits, and one year for assembled modules, following the date of purchase. The defective part must be returned postpaid to **Processor Technology Corporation** within the warranty period.

Any malfunctioning module, purchased as a kit and returned to **Processor Technology** within the warranty 3 month period, which in the judgement of **PTCO** has been assembled with care and not subjected to electrical or mechanical abuse, will be restored to proper operating condition and returned, regardless of cause of malfunction, with a minimal charge to cover postage and handling.

Any modules purchased as a kit and returned to **PTCO** which in the judgement of **PTCO** are not covered by the above conditions will be repaired and returned at a cost commensurate with the work required. In no case will this charge exceed $20.00 without prior notification and approval of the owner.

Any modules, purchased as assembled units are guaranteed to meet specifications in effect at the time of manufacture for a period of at least one year following purchase. These modules are additionally guaranteed against defects in materials or workmanship for the same one year period. All warranted factory assembled units returned to **PTCO** postpaid will be repaired and returned without charge.

This warranty is made in lieu of all other warranties expressed or implied and is limited in any case to the repair or replacement of the module involved.

Processor Technology

Processor Technology Corporation
6200 Hollis Street
Emeryville CA 94608

00	NOP	28	---	50	MOV D,B	78	MOV A,B	A0	ANA B	C8	RZ	F0	RP
01	LXI B,D16	29	DAD H	51	MOV D,C	79	MOV A,C	A1	ANA C	C9	RET	F1	POP PSW
02	STAX B	2A	LHLD Adr	52	MOV D,D	7A	MOV A,D	A2	ANA D	CA	JZ	F2	JP Adr
03	INX B	2B	DCX H	53	MOV D,E	7B	MOV A,E	A3	ANA E	CB	---	F3	DI
04	INR B	2C	INR L	54	MOV D,H	7C	MOV A,H	A4	ANA H	CC	CZ Adr	F4	CP Adr
05	DCR B	2D	DCR L	55	MOV D,L	7D	MOV A,L	A5	ANA L	CD	CALL Adr	F5	PUSH PSW
06	MVI B,D8	2E	MVI L,D8	56	MOV D,M	7E	MOV A,M	A6	ANA M	CE	ACI D8	F6	ORI D8
07	RLC	2F	CMA	57	MOV D,A	7F	MOV A,A	A7	ANA A	CF	RST 1	F7	RST 6
08	---	30	---	58	MOV E,B	80	ADD B	A8	XRA B	D0	RNC	F8	RM
09	DAD B	31	LXI SP,D16	59	MOV E,C	81	ADD C	A9	XRA C	D1	POP D	F9	SPHL
0A	LDAX B	32	STA Adr	5A	MOV E,D	82	ADD D	AA	XRA D	D2	JNC Adr	FA	JM Adr
0B	DCX B	33	INX SP	5B	MOV E,E	83	ADD E	AB	XRA E	D3	OUT D8	FB	EI
0C	INR C	34	INR M	5C	MOV E,H	84	ADD H	AC	XRA H	D4	CNC Adr	FC	CM Adr
0D	DCR C	35	DCR M	5D	MOV E,L	85	ADD L	AD	XRA L	D5	PUSH D	FD	---
0E	MVI C,D8	36	MVI M,D8	5E	MOV E,M	86	ADD M	AE	XRA M	D6	SUI D8	FE	CPI D8
0F	RRC	37	STC	5F	MOV E,A	87	ADD A	AF	XRA A	D7	RST 2	FF	RST 7
10	---	38	---	60	MOV H,B	88	ADC B	B0	ORA B	D8	RC		
11	LXI D,D16	39	DAD SP	61	MOV H,C	89	ADC C	B1	ORA C	D9	---		
12	STAX D	3A	LDA Adr	62	MOV H,D	8A	ADC D	B2	ORA D	DA	JC Adr		
13	INX D	3B	DCX SP	63	MOV H,E	8B	ADC E	B3	ORA E	DB	IN D8		
14	INR D	3C	INR A	64	MOV H,H	8C	ADC H	B4	ORA H	DC	CC Adr		
15	DCR D	3D	DCR A	65	MOV H,L	8D	ADC L	B5	ORA L	DD	---		
16	MVI D,D8	3E	MVI A,D8	66	MOV H,M	8E	ADC M	B6	ORA M	DE	SBI D8		
17	RAL	3F	CMC	67	MOV H,A	8F	ADC A	B7	ORA A	DF	RST 3		
18	---	40	MOV B,B	68	MOV L,B	90	SUB B	B8	CMP B	E0	RPO		
19	DAD D	41	MOV B,C	69	MOV L,C	91	SUB C	B9	CMP C	E1	POP H		
1A	LDAX D	42	MOV B,D	6A	MOV L,D	92	SUB D	BA	CMP D	E2	JPO Adr		
1B	DCX D	43	MOV B,E	6B	MOV L,E	93	SUB E	BB	CMP E	E3	XTHL		
1C	INR E	44	MOV B,H	6C	MOV L,H	94	SUB H	BC	CMP H	E4	CPO Adr		
1D	DCR E	45	MOV B,L	6D	MOV L,L	95	SUB L	BD	CMP L	E5	PUSH H		
1E	MVI E,D8	46	MOV B,M	6E	MOV L,M	96	SUB M	BE	CMP M	E6	ANI D8		
1F	RAR	47	MOV B,A	6F	MOV L,A	97	SUB A	BF	CMP A	E7	RST 4		
20	---	48	MOV C,B	70	MOV M,B	98	SBB B	C0	RNZ	E8	RPE		
21	LXI H,D16	49	MOV C,C	71	MOV M,C	99	SBB C	C1	POP B	E9	PCHL		
22	SHLD Adr	4A	MOV C,D	72	MOV M,D	9A	SBB D	C2	JNZ Adr	EA	JPE Adr		
23	INX H	4B	MOV C,E	73	MOV M,E	9B	SBB E	C3	JMP Adr	EB	XCHG		
24	INR H	4C	MOV C,H	74	MOV M,H	9C	SBB H	C4	CNZ Adr	EC	CPE Adr		
25	DCR H	4D	MOV C,L	75	MOV M,L	9D	SBB L	C5	PUSH B	ED	---		
26	MVI H,D8	4E	MOV C,M	76	HLT	9E	SBB M	C6	ADI D8	EE	XRI D8		
27	DAA	4F	MOV C,A	77	MOV M,A	9F	SBB A	C7	RST 0	EF	RST 5		

HEX-ASCII TABLE

Printing			Characters	
30	0		40	@
31	1		20	space
32	2		21	!
33	3		22	"
34	4		23	#
35	5		24	$
36	6		25	%
37	7		26	&
38	8		27	'
39	9		28	(
			29	)
41	A		2A	*
42	B		2B	+
43	C		2C	,
44	D		2D	-
45	E		2E	.
46	F		2F	/
47	G		3A	:
48	H		3B	;
49	I		3C	<
4A	J		3D	=
4B	K		3E	>
4C	L		3F	?
4D	M		5B	[
4E	N		5C	\
4F	O		5D	]
50	P		5E	↑ (∧)
51	Q		5F	← (-)
52	R			
53	S			
54	T			
55	U			
56	V			
57	W			
58	X			
59	Y			
5A	Z			

HEX-ASCII TABLE

Non-Printing	
00	NULL
07	BELL
09	TAB
0A	LF
0B	VT
0C	FORM
0D	CR
11	X-ON
12	TAPE
13	X-OFF
14	
1B	ESC
7D	ALT MODE
7F	RUB OUT

D8 = constant, or logical/arithmetic expression that evaluates to an 8 bit data quantity

D16 = constant, or logical/arithmetic expression that evaluates to a 16 bit data quantity.

Adr = 16 bit address

Processor Technology

Processor Technology Corporation
6200 Hollis Street
Emeryville CA 94608

JUMP

C3	JMP	Adr
C2	JNZ	Adr
CA	JZ	Adr
D2	JNC	Adr
DA	JC	Adr
E2	JPO	Adr
EA	JPE	Adr
F2	JP	Adr
FA	JM	Adr
E9	PCHL	

CALL

CD	CALL	Adr
C4	CNZ	Adr
CC	CZ	Adr
D4	CNC	Adr
DC	CC	Adr
E4	CPO	Adr
EC	CPE	Adr
F4	CP	Adr
FC	CM	Adr

RETURN

C9	RET
C0	RNZ
C8	RZ
D0	RNC
D8	RC
E0	RPO
E8	RPE
F0	RP
F8	RM

RESTART

C7	RST	0
CF	RST	1
D7	RST	2
DF	RST	3
E7	RST	4
EF	RST	5
F7	RST	6
FF	RST	7

ROTATE†

07	RLC
0F	RRC
17	RAL
1F	RAR

CONTROL

00	NOP
76	HLT
F3	DI
FB	EI

MOVE IMMEDIATE

06	MVI	B, D8
0E	MVI	C, D8
16	MVI	D, D8
1E	MVI	E, D8
26	MVI	H, D8
2E	MVI	L, D8
36	MVI	M, D8
3E	MVI	A, D8

Acc IMMEDIATE*

C6	ADI	D8
CE	ACI	D8
D6	SUI	D8
DE	SBI	D8
E6	ANI	D8
EE	XRI	D8
F6	ORI	D8
FE	CPI	D8

LOAD IMMEDIATE

01	LXI	B, D16
11	LXI	D, D16
21	LXI	H, D16
31	LXI	SP, D16

DOUBLE ADD†

09	DAD	B
19	DAD	D
29	DAD	H
39	DAD	SP

STACK OPS

C5	PUSH	B
D5	PUSH	D
E5	PUSH	H
F5	PUSH	PSW
C1	POP	B
D1	POP	D
E1	POP	H
F1	POP	PSW*
E3	XTHL	
F9	SPHL	

INCREMENT**

04	INR	B
0C	INR	C
14	INR	D
1C	INR	E
24	INR	H
2C	INR	L
34	INR	M
3C	INR	A
03	INX	B
13	INX	D
23	INX	H
33	INX	SP

DECREMENT**

05	DCR	B
0D	DCR	C
15	DCR	D
1D	DCR	E
25	DCR	H
2D	DCR	L
35	DCR	M
3D	DCR	A
0B	DCX	B
1B	DCX	D
2B	DCX	H
3B	DCX	SP

LOAD/STORE

0A	LDAX	B
1A	LDAX	D
2A	LHLD	Adr
3A	LDA	Adr
02	STAX	B
12	STAX	D
22	SHLD	Adr
32	STA	Adr

SPECIALS

EB	XCHG
27	DAA*
2F	CMA
37	STC†
3F	CMC†

INPUT/OUTPUT

D3	OUT	D8
DB	IN	D8

MOVE

40	MOV	B,B
41	MOV	B,C
42	MOV	B,D
43	MOV	B,E
44	MOV	B,H
45	MOV	B,L
46	MOV	B,M
47	MOV	B,A
48	MOV	C,B
49	MOV	C,C
4A	MOV	C,D
4B	MOV	C,E
4C	MOV	C,H
4D	MOV	C,L
4E	MOV	C,M
4F	MOV	C,A
50	MOV	D,B
51	MOV	D,C
52	MOV	D,D
53	MOV	D,E
54	MOV	D,H
55	MOV	D,L
56	MOV	D,M
57	MOV	D,A

MOVE (cont)

58	MOV	E,B
59	MOV	E,C
5A	MOV	E,D
5B	MOV	E,E
5C	MOV	E,H
5D	MOV	E,L
5E	MOV	E,M
5F	MOV	E,A
60	MOV	H,B
61	MOV	H,C
62	MOV	H,D
63	MOV	H,E
64	MOV	H,H
65	MOV	H,L
66	MOV	H,M
67	MOV	H,A
68	MOV	L,B
69	MOV	L,C
6A	MOV	L,D
6B	MOV	L,E
6C	MOV	L,H
6D	MOV	L,L
6E	MOV	L,M
6F	MOV	L,A
70	MOV	M,B
71	MOV	M,C
72	MOV	M,D
73	MOV	M,E
74	MOV	M,H
75	MOV	M,L
....		
77	MOV	M,A
78	MOV	A,B
79	MOV	A,C
7A	MOV	A,D
7B	MOV	A,E
7C	MOV	A,H
7D	MOV	A,L
7E	MOV	A,M
7F	MOV	A,A

ACCUMULATOR*

80	ADD	B	A8	XRA	B
81	ADD	C	A9	XRA	C
82	ADD	D	AA	XRA	D
83	ADD	E	AB	XRA	E
84	ADD	H	AC	XRA	H
85	ADD	L	AD	XRA	L
86	ADD	M	AE	XRA	M
87	ADD	A	AF	XRA	A
88	ADC	B	B0	ORA	B
89	ADC	C	B1	ORA	C
8A	ADC	D	B2	ORA	D
8B	ADC	E	B3	ORA	E
8C	ADC	H	B4	ORA	H
8D	ADC	L	B5	ORA	L
8E	ADC	M	B6	ORA	M
8F	ADC	A	B7	ORA	A
90	SUB	B	B8	CMP	B
91	SUB	C	B9	CMP	C
92	SUB	D	BA	CMP	D
93	SUB	E	BB	CMP	E
94	SUB	H	BC	CMP	H
95	SUB	L	BD	CMP	L
96	SUB	M	BE	CMP	M
97	SUB	A	BF	CMP	A
98	SBB	B			
99	SBB	C			
9A	SBB	D			
9B	SBB	E			
9C	SBB	H			
9D	SBB	L			
9E	SBB	M			
9F	SBB	A			
A0	ANA	B			
A1	ANA	C			
A2	ANA	D			
A3	ANA	E			
A4	ANA	H			
A5	ANA	L			
A6	ANA	M			
A7	ANA	A			

PSEUDO INSTRUCTION

ORG	Adr
END	
EQU	D16
DS	D16
DB	D8 . .
DW	D16 . .

CONSTANT DEFINITION

0BDH, 1AH	Hex
105D, 105	Decimal
72O, 72Q	Octal
11011B, 00110B	Binary
'TEST', 'A' 'B'	ASCII

OPERATORS

STANDARD SETS

A	SET	7
B	SET	0
C	SET	1
D	SET	2
E	SET	3
H	SET	4
L	SET	5
M	SET	6
SP	SET	6
PSW	SET	6

D8 = constant, or logical/arithmetic expression that evaluates to an 8 bit data quantity

* = all Flags (C Z S P) affected

D16 = constant, or logical/arithmetic expression that evaluates to a 16 bit data quantity

† = only CARRY affected

Adr = 16 bit address

** = all Flags except CARRY affected; (exception INX & DCX affect no Flags)

LOADING DIP (DUAL IN-LINE PACKAGE) DEVICES

Most DIP devices have their leads spread so that they can not be dropped straight into the board. They must be "walked in" using the following procedure:

(1) Orient the device properly. Pin 1 is indicated by a small embossed dot on the top surface of the device at one corner. Pins are numbered counterclockwise from pin 1.

(2) Insert the pins on one side of the device into their holes on the printed circuit card. Do not press the pins all the way in, but stop when they are just starting to emerge from the opposite side of the card.

(3) Exert a sideways pressure on the pins at the other side of the device by pressing against them where they are still wide below the bend. Bring this row of pins into alignment with its holes in the printed circuit card and insert them an equal distance, until they begin to emerge.

(4) Press the device straight down until it seats on the points where the pins widen.

(5) Turn the card over and select two pins at opposite corners of the device. Using a fingernail or a pair of long-nose pliers, push these pins outwards until they are bent at a 45 degree angle to the surface of the card. This will secure the device until it is soldered.

SOLDERING TIPS

(1) Use a low-wattage iron — 25 watts is good. Larger irons run the risk of burning the printed-circuit board. Don't try to use a soldering gun, they are too hot.

(2) Use a small pointed tip and keep it clean. Keep a damp piece of sponge by the iron and wipe the tip on it after each use.

(3) Use 60-40 rosin-core solder ONLY. DO NOT use acid-core solder or externally applied fluxes. Use the smallest diameter solder you can get.

NOTE: DO NOT press the top of the iron on the pad or trace. This will cause the trace to "lift" off of the board which will result in permanent damage.

(4) In soldering, wipe the tip, apply a light coating of new solder to it, and apply the tip to both parts of the joint, that is, both the component lead and the printed-circuit pad. Apply the solder against the lead and pad being heated, but not directly to the tip of the iron. Thus, when the solder melts the rest of the joint will be hot enough for the solder to "take," (i.e., form a capillary film).

(5) Apply solder for a second or two, then remove the solder and keep the iron tip on the joint. The rosin will bubble out. Allow about three or four bubbles, but don't keep the tip applied for more than ten seconds.

(6) Solder should follow the contours of the original joint. A blob or lump may well be a solder bridge, where enough solder has been built upon one conductor to overflow and "take" on the adjacent conductor. Due to capillary action, these solder bridges look very neat, but they are a constant source of trouble when boards of a high trace density are being soldered. Inspect each integrated circuit and component after soldering for bridges.

(7) To remove solder bridges, it is best to use a vacuum "solder puller" if one is available. If not, the bridge can be reheated with the iron and the excess solder "pulled" with the tip along the printed circuit traces until the lump of solder becomes thin enough to break the bridge. Braid-type solder remover, which causes the solder to "wick up" away from the joint when applied to melted solder, may also be used.

STANDARD COLOR CODE FOR RESISTORS AND CAPACITORS

COLOR	SIGNIFICANT FIGURE	DECIMAL MULTIPLIER	TOLERANCE (%)	VOLTAGE RATING*
Black	0	1		---
Brown	1	10		100
Red	2	100		200
Orange	3	1,000		300
Yellow	4	10,000		400
Green	5	100,000		500
Blue	6	1,000,000		600
Violet	7	10,000,000		700
Gray	8	100,000,000		800
White	9	1,000,000,000		900
Gold	-	0.1	5	1000
Silver	-	0.01	10	2000
No Color	-	---	20	500

*Applies to capacitors only.

Title System Application Notes

Function

Release Date: 7/1/76
Revision No. Ø
Level No. A1

System Application Notes

System Application Notes will be issued on a regular basis to members of the ALS-8 Systems Group. These notes will cover a broad spectrum of information, and it is the purpose of this note to specify the classification system used for the information catagories covered.

GUIDE TO SYSTEMS APPLICATION NOTES

ALS-8

Program Development System

Title: System Application Notes

Function:

Release Date: 7/2/76
Revision No. Ø
Level No. A1
Page 2 of 2

System Application Notes will be issued with an alphabetic letters designating the level of information, followed by a number indicating the correct sequence within that level.

From time to time notes may be re-issued containing revised information. These notes containing the lower revision number should be discarded with the latter one being filed in its place.

At the present time the following characters have been assigned to the levels indicated.

A	System Familiarization
B	Advanced System Techniques
C	General Operating Notes
D	Programming Techniques
J	Support Utility specification/description
K	System subroutine specification/description
L	Support Program description
R	System Parameter Specification
Z	Notes, scratches and misdirected miscellany

Title System Start Up

Function

Release Date: 7/2/76
Revision No. 0
Level No. A2
Page 1 of 3

SYSTEM START UP

ALS-8 ENTRIES

ALS-8

Program Development System

The circuit board which holds the ALS-8 system contains circuitry which will normally start-up as well as address the computer for running the ALS-8. This start up address represents the first byte of the system located at EØØØ.

This address is only "one" of the ALS-8 start-up points and each different point performs a given function.

Within the ALS-8 certain parameters require initialization prior to operation of the system. Of these, the I/O driver code, which is moved from PROM to RAM is most important. This code is moved "fresh" into ram following entry to EØØØ.

The terminal width, MODE and STAB parameters are also reset after entry at this point.

An alternate point, address EØ6Ø, does no initialization but only restores the driver to SYSIO and prints the "READY" message.

The third direct entry point is used to zero the file, custom command and system symbol table values. Because this function is required the operator should EXEC this address when power is first applied. (EXEC EØ24)

In summary:

Address	USE
EØ24	Use when power is first applied or after a major program crash.
EØØØ	Use after a minor crash to reinitialize the RAM I/O drivers. Following this the File, IODR, CUST, and symbol Table can be examined for possible affects.
EØ6Ø	Use to return to the ALS-8 after an operational program has gone into an endless loop or halted.

Title CUST and EXEC

Function Further Explanation prior to bulletin

Release Date: 7/2/76
Revision No. Ø
Level No. A3
Page 1 of 1

CUST and EXEC at first seem to have the same effect but a closer examination will indicate the CUSTOM command is able to retrieve and use all of the parameter passing of the ALS-8 system.

How to pick up these parameters is a complex operation determined even by the requirements of the routine picking them up. Further bulletins will describe each type of parameter in detail and how to use them with custom commands.

CUST OR EXEC?

ALS-8

Program Development System

Title System Application Notes

Function

Release Date: 7/1/76
Revision No. Ø
Level No. A1

System Application Notes

System Application Notes will be issued on a regular basis to members of the ALS-8 Systems Group. These notes will cover a broad spectrum of information, and it is the purpose of this note to specify the classification system used for the information catagories covered.

GUIDE TO SYSTEMS APPLICATION NOTES

ALS-8

Program Development System

System Application Notes will be issued with an alphabetic letters designating the level of information, followed by a number indicating the correct sequence within that level.

From time to time notes may be re-issued containing revised information. These notes containing the lower revision number should be discarded with the latter one being filed in its place.

At the present time the following characters have been assigned to the levels indicated.

A	System Familiarization
B	Advanced System Techniques
C	General Operating Notes
D	Programming Techniques
J	Support Utility specification/description
K	System subroutine specification/description
L	Support Program description
R	System Parameter Specification
Z	Notes, scratches and misdirected miscellany

ALS-8 I/O System Driver

This System Application Bulletin describes the operation of the SYSIO driver associated with the ALS-8 executive. Sufficient information is given to allow "system start-up" under conditions where the standard input device is not available.

I/O DRIVER

ALS-8

Program Development System

The ALS-8 executive make use of a SYSTEM driver pair known by the name "SYSIO". The operating code for this pair of drivers as well as the name is stored in the PROM holding the ALS-8 system.

When the initialization procedure is run the executive moves this code into random access memory for use by the system. The initialization procedure also places the driver name, "SYSIO", in the I/O driver table as well as the address associated with the drivers just loaded.

At this time this is the only driver known to the system and it will loop waiting for input from the operator. If a properly implemented input device is available the operator can then change the SYSIO driver to any other specification desirable as long as the corresponding drivers are available.

If a standard input device is not available the system operator can manually, via the computer front panel, change either the driver code or driver addresses to correspond to his device. Once this is done the all normal system operations can begin.

A. If a version of the non standard driver is in memory two addresses must be changed for the system to recognize the new driver as the SYSIO...

For Input:

The driver address should be placed in the following memory locations using the standard Intel format of low byte, high byte.

Address	Present Value
DØCD	98
DØCE	DØ
DØ94	98
DØ95	DØ

If a standard output driver is not available but a known driver already exists in memory the driver address should be placed in the following double byte memory locations.

DØDØ	A9
DØD1	DØ
DØ96	A9
DØ97	DØ

Title: ALS-8 I/O System Driver

Function:

Release Date: 7/6/76
Revision No. Ø
Level No. A4

If the available input or output devices are compatible with the system drivers but use different status bits or port numbers the system operator can change the driver code to correspond to his requirements. A listing of this code is provided here to aid in implementing the changes necessary.

The user is cautioned however, that the system will restore the standard I/O driver any time EØØØ or EØ24 is executed. In addition, the simulator and VDM drivers assume that the standard system driver is available.

```
D098            *
D098            *
D098            *                  ALS-8 SYSTEM I/O DRIVERS
D098            *
D098            *
D098            *
D098            *      INPUT DRIVER
D098            *
D098 CD A4 D0   INP8    CALL   STAT    GET STATUS
D09B CA 98 D0           JZ     INP8    LOOP UNTIL AVAILABLE
D09E            *
D09E DB 01              IN     UDATA   GET DATA FROM INPUT PORT
D0A0 E6 7F              ANI    127     STRIP OFF PARITY
D0A2 47                 MOV    B,A     PUT COPY IN ALTERNATE REGISTER
D0A3 C9                 RET
D0A4            *
D0A4 DB 00      STAT    IN     USTA
D0A6 E6 40              ANI    DAV     TEST FOR DATA AVAILABLE
D0A8 C9                 RET
D0A9            *
D0A9            *      OUTPUT DRIVER
D0A9            *
D0A9 CD A4 D0   OUTP8   CALL   STAT    GET INPUT STATUS
D0AC CA B8 D0           JZ     NOCHR   JUMP IF NO INPUT HAS BEEN RECEIVED
D0AF DB 01              IN     UDATA   GET CHARACTER
D0B1 E6 7F              ANI    127
D0B3 FE 1B              CPI    ESC     IS IT AN ESCAPE?
D0B5 CA 60 E0           JZ     EORMS   IF SO CHANGE DRIVER AND OUTPUT "READY"
D0B8 DB 00      NOCHR   IN     USTA
D0BA E6 80              ANI    TBE     IS PORT READY FOR OUTPUT?
D0BC CA B8 D0           JZ     NOCHR
D0BF 78                 MOV    A,B     GET CHARACTER FOR OUTPUT
D0C0 D3 01              OUT    UDATA
D0C2 C9                 RET
D0C3            *
D0C3            UDATA   EQU    1       DATA PORT NUMBER
D0C3            USTA    EQU    0       STATUS PORT NUMBER
D0C3            DAV     EQU    40H     DATA AVAILABLE AT BIT 6
D0C3            TBE     EQU    80H     TRANSMITTER BUFFER EMPTY AT BIT 7
D0C3            ESC     EQU    1BH     ESCAPE CHARACTER
D0C3            *
D0C3            *

DAV     0040    ESC     001B    INP8    D098    NOCHR   D0B8
OUTP8   D0A9    STAT    D0A4    TBE     0080    UDATA   0001
USTA    0000
```

Subroutine and Command Returns

Returning to ALS-8

ALS-8

Program Development System

Title: Subroutine and command returns

Function:

Release Date: 7/2/76
Revision No. Ø
Level No. C1
Page 2 of 4

When used with either EXEC or a custom command the ALS-8 exits with a normal 8080 calling sequence. If normal stack operations are used within the routine a RET instruction will return directly to the ALS-8 instruction. The stack used by the system will allow sixteen levels of external stack operations without affecting the system's global area.

Two parameters are passed to the external routine each time a command is given. These parameters are known symbolically as SWCH 1 and SWCH 2. If they are non-zero on return they will affect the response of the executive in a-determined way.

SWCH 1 is tested first by the executive and if non zero its effect cancels that of the second parameter.

SWCH 1 is called the "outgoing switch" and if it is non-zero on return the executive will change back to the system driver and output the "READY" message.

Title: Subroutine and Command Returns

Function:

Release Date: 7/2/76
Revision No. 1
Level No. C1
Page 3 of 4

SWCH 2 is tested if SWCH 1 is zero. This parameter is called the "driver hold request". If SWCH 2 is non zero on return the executive will not switch back to the system driver and no message will be given.

If both switches indicate Ø the executive will change back to the systems driver but will output only a CRLF.

NAME	ADDRESS
SWCH 1	DØFD
SWCH 2	DØFE

Any external routine or program can also return to the system by JMP ing to one of the following locations.

ADDRESS	SYMBOLIC NAME
EØB7	EOR
EØD1	EORNS
EØ6Ø	EORMS
E7DD	WHAT
E7EØ	MESS

Title: Subroutine and command returns

Function:

Release Date: 7/2/76
Revision No. Ø
Level No. C1
Page 4 of 4

NAME	FUNCTION
EOR	Perform parameter tests as per standard RET.
EORNS	Return to system leaving I/O driver set.
EORMS	Return to system, switching to system driver and outputting "READY" message.
WHAT	Return to system, switching to system drivers and outputting "WHAT" message.
MESS	Return to system, switching to system drivers, and outputting message addressed by H + L. Message must end with a ØDH Byte. (13 decimal)

TEMPORARY OPERATORS MANUAL
CHAPTER I

The ALS-8 is a single terminal operating system designed for use with "8080" based micro-computers. The system software is contained on a printed circuit board in programable read-only memory. This same board also has circuitry which will normally start the operating system once the computer is turned on. This configuration, called a "turnkey system", eliminates the start-up procedure usually required from the computer's front panel switches. The fact that the ALS-8 program is always stored in memory, regardless of power conditions, eliminates the system load, or "bootstrapping" normally needed by small machines.

In this manual the name "ALS-8" will refer not only to the circuit board but also the operating system program contained on the board. The manual will describe the many capabilities of the ALS-8 and how they are used. Chapter Two also describes the hardware requirements needed for running an ALS-8.

The ALS-8 is a personalized operating system which attempts to maximize convenience in program development without over-controlling the machine. Operating systems, even the large computer variety, can be guilty of "over-control" when design assumptions become user restrictions. The ALS-8 has assumptions incorporated into its design as must any program, but the ALS-8 allows access to "parameters" which can redefine these assumptions. In this way various input/output devices or memory configurations can be accommodated. Another personalized feature allows the user to expand the ALS-8 by adding his own functions to it. Each of the initial operating system functions resides in its own section of the ALS-8 memory and is activated by a command word or "key word" sent from the terminal. Additional functions only have to be given a memory start address and a name for the associated command. The new function is executed whenever the ALS-8 sees the custom command name associated with that function.

The ALS-8 relies heavily on the concept of parameters in its internal design and its command interpretation. The fundamental idea is contained in the observation that two similar tasks differing by some element should be a single task which modifies its operation based on the value of this "element." A simple example of this concept is the ALS-8 output formatting routine. A number of printing terminals are available which could be interfaced to the computer containing the ALS-8, and these terminals often vary in the width of paper they accept. Some standard widths are 72, 80, 110, and 132 characters per line. It is conceivable then that a separate ALS-8 package could be written to handle the specific terminal attached to its computer. The parameter principle suggests instead that a single ALS-8 be made with provision for defining or redefining this parameter, the terminal width. This is, in fact, exactly what is done. Before printing, the output routine checks this value to see how it should format the output line. The ALS-8 has several such parameters which it uses to control its various functions.

This concept of parameters is carried into the command structure in much the same way. While interpreting a command, the ALS-8 checks for an optional list of "arguments", which could be one or two numbers, and for a name enclosed in slash marks (/). These values are stored in the order found and if the function chosen by the command name needs this information for its own functioning, it retrieves it from a predetermined location in memory. The only appreciable difference between arguments and parameters is that arguments are temporarily stored and only for the current command, while parameters describe conditions which may be of interest to many functions. Parameters also keep their values until explicitly redefined. Using the features which arise from this principle, the user can tailor the operating system to his own personal requirements.

The ALS-8 contains an assembler, file handling routines, editing, and management functions. The functions within these logically distinct sections of the operating system can be combined in many ways to aid in the writing and debugging of programs. The text for a program, and often times data, is written from the terminal onto a "file" in memory where it can be examined, altered, added to, or saved for later. The ALS-8 resident asembler can convert the program text on such a file into the numeric machine language required by the CPU. This machine language is then stored by the assembler at some user designated memory location where it can be run. Up to six of these files can be managed at one time by the ALS-8.

A very important aspect of the ALS-8 in program development is the fact that any user program has access to all the ALS-8 functions and support routines. For many problems this means that half the program is written, debugged, and ready as soon as the computer is powered up. All the user's program must do is is call the already existing routines. Naturally the user program has to be aware of the conventions and assumptions associated with the routines it calls, but it is far simpler and much faster to learn these than to write such routines from scratch each time a particular function is needed. Later sections of this manual deal with this feature in more detail.

Another important design feature of the ALS-8 is its ability to maintain and effectively use a SYSTEM SYMBOL TABLE. The user, through the appropriate commands, can enter and delete names in this list or "table." These names carry only an associated number with them which is usually interpreted as a memory address. This table is accessible to the assembler and any other function (user program) which cares to reference it. This can be used quite effectively to link together programs written at different times. The address (or value) of a certain quantity does not have to be known at the time that a program is being assembled. That program can, instead, contain code which looks for this value in the symbol table.

CHAPTER 2

MEMORY AND PROGRAM STRUCTURE OF THE ALS-8

A structural description of the ALS-8 is given here to define the minimum hardware requirements and to outline the principles behind its construction so that the fullest advantage may be made of the features available. The program ALS-8 is distributed on the printed circuit board mentioned in the first section and it is this board that defines some of the hardware constraints. The program itself could be used on any 8080 based computer which has retained the 64K addressing scheme of the 8080 chip. The circuit board, however, restricts the present ALS-8 to computers having a connector with the correct mechanical and electrical characteristics available.

The circuit board also determines the location in memory for the program. The board itself is capable of holding 8K bytes of PROM of which the ALS-8 takes over half. This memory page is hardwired on the board to reside in the last 8K page of memory so that it addresses from E000 hex to FFFF hex. The program itself also has memory requirements, the software assumes that at least 1K of random access memory (RAM) resides in memory starting at location D000 hex.

While this memory configuration is enough to let the ALS-8 operate, it is insufficient for most programming requirements. It is strongly suggested that a separate memory be provided in the low part of memory, preferably starting at 0000 to serve as the user's free space for putting programs, files, and data. This is suggested because there is very little free space around the D000 RAM; in fact, it is also suggested that the system RAM board be 4K (from D000 to DFFF).

The ALS-8 is very flexible with regard to peripheral devices, but it does make some initial assumptions about the terminal which constitute a hardware requirement. Devices attached to any 8080 based computer identify themselves to the computer with a number called a "device code." There are 256 posible codes for input devices and 256 for output devices. As initialized, it is assumed than the keyboard is input device code 1 and that the print mechanism is output device 1. It is also assumed that the computer, or the ALS-8 in this case, can retrieve status information about the terminal from input device 0, the most significant bit, 10000000, represents the busy status of the output device and the next lower bit, 01000000, has the busy status of keyboard. The terminal printer is busy when its bit is 0, and data is assumed available from the keyboard when its bit is 1. This I/O driver is in the System RAM area and can be changed by the user following system initialization but since this convention is assumed by a good deal of the software written for 8080 based computers it is suggested that it be followed.

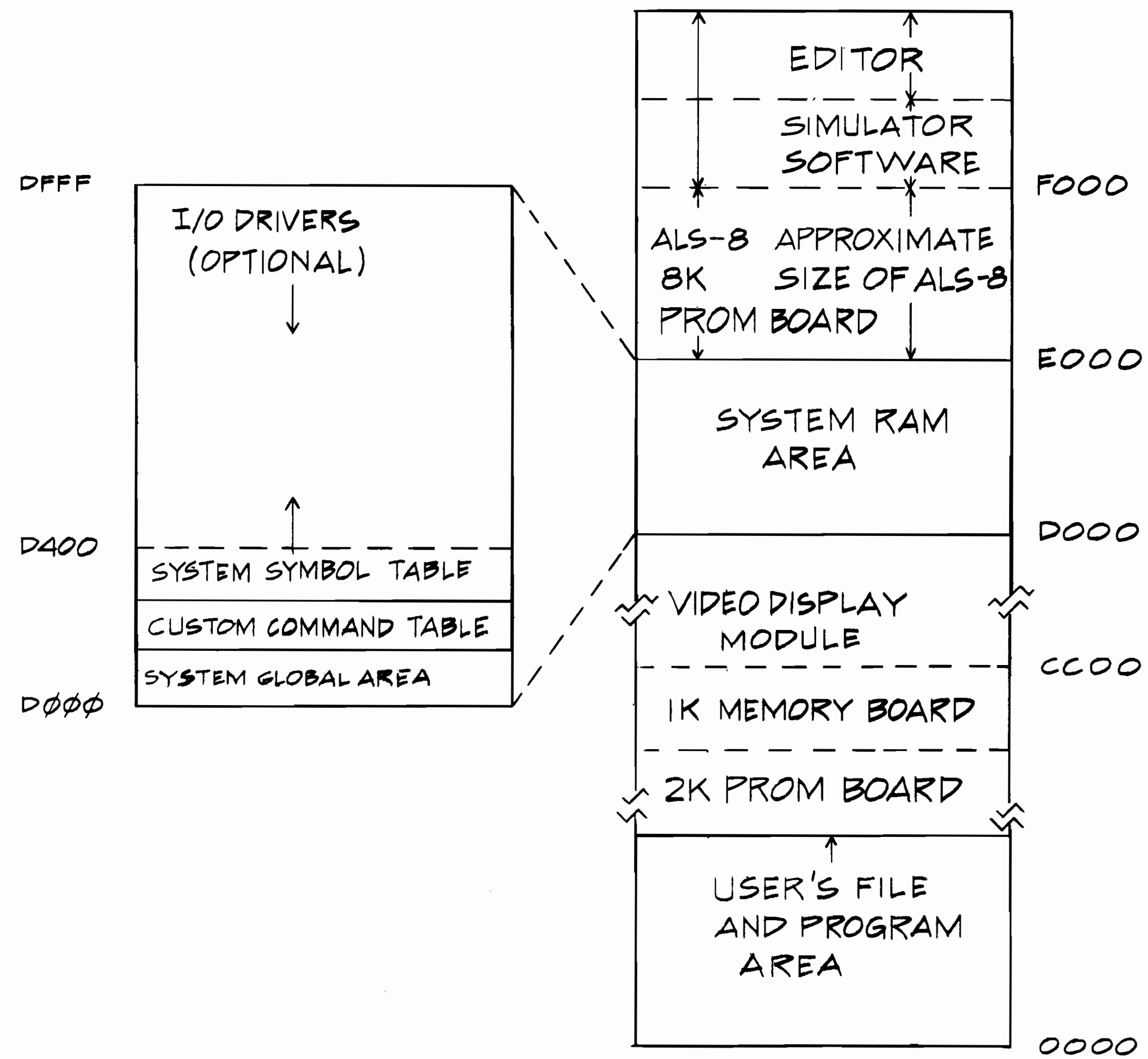
The ALS-8 keeps a great deal of information in the system RAM area and, to use the ALS-8 to its fullest, the reader should learn how this information is used. In the following discussion on the system RAM area, it will be assumed that the 4K space reserved for it is actually filled with memory. The reasons will become clear as the discussion progresses.

The first block of information in this area occupies addresses D000 to D25C and is called the System Global Area. Parameters defining or describing I/O devices, program status, and other information are stored here. Immediately following this is the Custom Command Table which contains a list of names defined by the user with the CUSTE command which will be described in some detail later. Each entry in this table is paired with an address given when the command was defined. When the user types out one of these custom names, the ALS-8 realizes it isn't a name from its own command set. It then searches this custom table, picks up the corresponding address and performs a subroutine jump (call) to that address. This table ends at D2FF which leaves room for 22 custom names.

The System Symbol Table follows the custom commands and continues out to DFFF where the ALS-8 software starts. This table, like the Custom Command Table, contains names and corresponding sixteen bit numbers which are usually thought of as addresses. This is used most often by the ALS-8 resident assembler but it is open for use to any user routine which cares to access it. It allows user routines to be parametrized so that the routine can access information not available at the time it is written and assembled. This is especially useful for connecting programs and subroutines written at much different times. Note that systems having only 1K board at D000 will be restricting this System Symbol Table to the area D300 to D377, only sixty-four bytes of memory. This severely limits the usefulness of this feature.

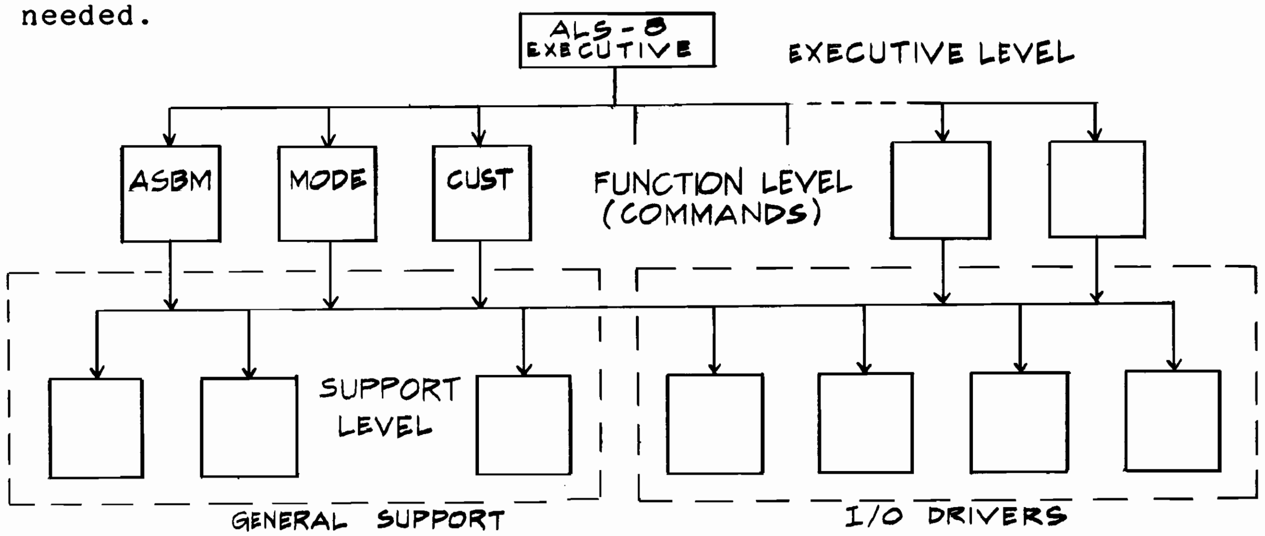
It was suggested earlier that RAM be placed in the low part of memory space for the user. This is suggested to minimize congestion and the possible memory conflicts arising between the system and user software. In keeping with this philosophy, special user written routines designed to handle I/O devices should be stored somewhere in the system. These routines, called I/O drivers, can be put anywhere but should probably be located in the RAM just under the E000 start of the ALS-8 program until they are put in more permanent form. This still gives the System Symbol Table as much room as possible while maintaining the system/user separation. The following diagram summarizes the memory map described so far and shows the suggested locations for the Video Display Module and optional memory.

INSERT DIAGRAM



The separation of system space from user space results in an upward progression of address values for user memory and a downward progression for system memory. Future products have assumed that this policy has been carried out and that the Video Display Module (VDM), for instance is located just below the D000 start of system-RAM. This VDM should then start at location CC00 hexidecimal. The presence of the VDM in the C000-CFFF block means that no 4K board could be placed there. It is, however, suited to a 2K PROM board and perhaps a 1K memory board, should it become important to fill up this space completely. The space from 9000 all the way to BFFF has been marked as the best location for further extensions of the System. As I/O drivers, loaders and other user software is developed it is suggested that they be placed in PROM in the C000 to C7FF block. Future software packages will assume this memory structure.

The program structure of the ALS-8 is most easily described with the aid of the following diagram. The conceptual parts to the program are shown as parts of a heirarchy not completely unlike the structure of a government or a business. In such a diagram, it is assumed that the higher levels are able to command the lower levels but not the other way around. In the program sense then, the top most level can call on any of the routines below as subordinates. It is assumed also in this diagram that routines on the same level may call each other as needed.



The top level, the executive level in this diagram, represents the control center. It is this section which controls the communications with the terminal, decides which funtion is to be executed, and reports on errors to the user. Each block on the function level corresponds to a command from the ALS-8 command set. These routines, for efficieny sake, make heavy use of the support routines on the next level making the overall package much smaller. These support routines have been divided into two parts, general support and I/O drivers. The I/O drivers are support routines which handle the transfer of data to or from external devices. They are logically distinct from the general support routines beause only the drivers handle I/O and because the ALS-8 allows the user to define his own routines as drivers thereby adding to this part of the system. Each new driver added usually has charge of just a single device. They could be used, as will be described in the chapter on I/O drivers, to control high speed paper tape readers, cassette recorders, or printers. The custom commands also add to the structure diagram but do so on the function level. They too can make use of all the general support, I/O drivers, or other function level blocks to minimize their own size and complexity. Other complete, self-contained programs may be considered custom functions (like BASIC or FOCAL) and this interaction with support routines or drivers is only a convenience, not a requirement.

It is important to realize that many of the decisions made by the ALS-8 in choosing support routines or drivers for a given task depend on status information kept in the system RAM area. Although there may be quite a number of I/O driver routines identified to the system, only one input driver and one output driver are considered current at any one time and their identities are kept in this memory area. Similarly, certain parameters will influence the flow of control through the program structure.

CHAPTER III - TALKING TO THE ALS-8

The command set recognized by the ALS-8 can be naturally divided into five catagories; MEMORY, FILE, EDITING, I/O and SYSTEM commands. The memory commands are used to enter data into memory or examine the contents of a section of memory. Usually these data transfers are between memory and the keyboard and printer of the terminal, but with proper equipment and drivers, the memory commands become a means of saving and restoring programs. The file commands verify, relocate, and manage up to six files of information in memory while the edit commands manipulate the contents of the files. The category of system commands includes all the commands which define system parameters, symbols, and drivers. It also contains commands which execute the assembler, the optional simulator, or any user designated location(s) in memory. The following table lists the command names in their respective categories. The names marked with an asterisk are commands used only by the optional VDM Editor or Simulator software packages.

MEMORY	FILE	EDIT	SYSTEM
ENTR	FILE	DELT	IODR
DUMP	FILES	EDIT (*)	SWCH
	FCHK	LIST	MODE
	FMOV	TEXT	ASSI
	FIND (*)	RNUM	ASSM
			EXEC
			SIMU (*)
			AUTO (*)
			SYML
			SYMLE
			SYMLD
			STAB
			CUST
			TERM
			FORM
			NFOR

The above list represents the default command set recognized by the ALS-8 executive routine. Individual ALS-8 functions, while operating, will recognize other lines as inputs. The ENTR command, for example, takes control of the terminal and expects to receive numeric input data to place in memory. This function must be given a special character signifying the end of input before it will return control to the ALS-8 executive. The ENTR function will not recognize entries from the executive's command set. An error message is output to the terminal when an entry line is unrecognizable.

Other than custom commands which have been briefly covered, the ALS-8 executive does recognize a command line type not shown in the command set list. Lines beginning with a number are assumed to be line entries to a file of information stored in memory. Files are a very powerful feature of the ALS-8 which will be thoroughly covered in Chapter V. For the moment, it suffices to note that they contain text, usually program text for the assembler, and that they normally sequence their contents by line numbers. The text you are now reading however, is an example of a text file without line numbers using the optional TXT-2 extension to the ALS-8.

A number of the executive commands accept "arguments" as modifiers for the associated function. The ALS-8 executive allows a maximum of two numeric values and one ASCII argument as modifiers to a command. How the arguments are used, if they are used at all, depends on the command chosen. In use, the arguments are interpreted by the order in which they appear. Commands using an ASCII argument will expect it to be the first argument given. The ASCII argument, usually a name in one of the many tables used by the ALS-8 also has the requirement that it must be enclosed in slash marks (/). The following example shows a number of commands as they might appear with arguments.

```
ASSM 2000
ASSM  2000    3000
FILE   /FNAME/  100
DUMP 101  110
CUSTE /HACF/307
IODR   /TAPES/ DF00 DF80
```

Most of the ALS-8 functions contain logic to handle instances where an argument has been omitted. In such instances, a default rule peculiar to the command and argument in question will be applied. The "ASSM" command shown in the examples above can be used with one or two arguments. The command starts the assembler which begins by checking for a pair of arguments. It interprets the first argument as the origin (ORG) address for the program being assembled. The second argument specifies the starting address for the assembler's binary output (machine instructions). If this second argument is missing, the assembler will take the value given in the first argument for both arguments. The assembler has no provision for defaulting two arguments so it will signal an error if the ASSM command is given with no arguments. Default rules for all executive commands will be given in the detailed description of these commands in the upcoming chapters. Again it is mentioned that the user functions attached to custom commands have full use of the argument handling support routines; the treatment of default conditions is naturally up to the programmer.

Finally, it must be noted that there are some minor rules to be observed in the use of command inputs with arguments. The ALS-8 executive needs to separate the characters belonging to the command from those of the arguments. Similarly, it needs to separate arguments from one another. The requirement is therefore put on the userto put at least one blank after the command word and at least one blank between a pair of numeric arguments. The slash at the end of an ASCII name argument is sufficient to separate the name from any following numbers. Numeric arguments may follow an ASCII argument with no separs-ting blanks as long as the argument was terminated with a slash mark.

Responses from the ALS-8 in general depend upon the command chosen. For the standard ALS-8 command set, the user is always assured of a response; if a response is not a normal duty for a command, the ALS-8 executive will send the word "READY" to the user's terminal.

CHAPTER IV - MEMORY RELATED COMMANDS

The simplest commands in the ALS-8 repertoire are the memory related commands, ENTR and DUMP. They provide a means of changing and examining memory locations directly from the users terminal. The output printing format of the DUMP command has been made compatible with input format requirements of the ENTR command. This permits these commands to be used for saving programs on a mass storage device and returning it to memory at a later time. This feature will be covered here and in the chapter on I/O drivers.

The ENTR command requires a single argument defining the starting address for the data to be entered. The command starts the corresponding ENTR function which assumes control of the user's selected input device until receiving the character "/" signifying the end of the input stream. The actual input to the ENTR function is a list of values, each betweeen 0 and 255 decimal in magnitude. These values must be listed in the order they are to be placed in memory and each must be separated from adjacent values by at least one blank. The following shows typical sequences using this command. Note that the input list may use any number of lines up to the "/" mark.

```
ENTR   100
20 303    55  40
16  12
107 200 303    100  0
/
READY
ENTR    2001
101 200   /
READY
ENTR   3
0  7/
READY
```

The argument and input list can be in octal, as shown above or in hexidecimal depending on the current mode parameter set by the system class command MODE. The MODE command affects the operation of other ALS-8 commands, not just memory commands. It takes a single decimal argument, 8 or 16, which is stored in the system parameter defining the base for command inputs. If any inputs are received which are impossible to decode with the current base, a "WHAT?" will be sent to the user's terminal. The ALS-8 initializes this parameter at start time to 16 and this value is changed only with MODE. The following shows possible errors associated with the MODE parameter.

```
MODE 16
ENTR 156000  (Octal address)
WHAT?

MODE 8
ENTR CC0D   (Hex address)
WHAT?
```

```
MODE 16
ENTR BF2
52 49 EE 4F 52 F6 43 50
5 A0 0 84 E4
43 2 303 22
WHAT?
```

In the last of the examples the values up to the error are properly stored by the ENTR function. The corrected input will have to restart at the place of the error.

An added feature of the ENTR command is that the present storage address may be changed during input without having to stop the process and restart with a new argument. The "present storage address" always starts with the value given by the attached argument to ENTR and the first input value is put in this location; inputs are placed in successive locations. The user has an opportunity at the start of each input line to re-define this current address. If the first value is followed immediately by a colon (:), it is treated as a new address rather than a memory value. While this seems only a minor convenience, it becomes the key to making the output of DUMP compatible with ENTR input. The following shows the first example of this chapter re-written using this feature.

```
MODE 8
ENTR 100
2 303 55 40 16 12 107 200
303 100 0
2001: 101 200
3: 0 3 /
READY
```

The DUMP command displays the contents of memory starting at the address specified in the first argument and continuing to the address specified by the second. As with ENTR, both the arguments and the output follow the base parameter set by MODE. The DUMP command can also be used with just a single argument; in this case it types out only the location specified in the first argument.

The lines output by the DUMP command each start with the current address followed by a colon. The remainder of the line contains the hexidecimal or octal contents of the memory locations beginning with the printing address. In either the octal or hexidecimal mode, the DUMP command puts sixteen values on each line. Because this output is formulated properly for ENTR, those users with a paper tape punch can save the output directly on tape and re-read it later with ENTR. In this case the standard ALS-8 I/O driver could be used; saving programs on other devices will require using special drivers. The following shows a simple example of DUMP in the hexidecimal mode.

```
DUMP 40 52

0040: 0A D8 D6 07 C9 DB 00 E6 45 00 DC 01 D3 02 F8 CF
0050: E6 7F C9
```


The ALS-8 relies very heavily on the use of files; for they represent a very powerful way of managing data in text form. A file is a sequence of information stored in user designated memory. The information is broken into "lines" which are duplicates of the terminal input lines which define them. Each line, both as it is input and as it is stored in memory, starts with a line number defining its position in the file relative to other lines. Lines with the lower line numbers are at the start or "top", of the file while higher numbered lines have positions farther "down" in the file. The lines do not have to be entered in numeric order by the line numbers. The ALS-8 will reposition other lines to make sure the proper order is kept internally. Once in memory, files can be renumbered using the RNUM command.

Files are known to the ALS-8 by name and up to six files can be defined and managed at any one time. File names may have up to five characters. Rather than having each file-related command specify which file is to be operated on, the ALS-8 has the user define "Current File." Using the FILE command the user can specify which of his defined files is to be considered "current." All file operations will apply to this file until the Current File is redefined with the FILE command.

To create a file the user must give a name for the file and a starting address for it. This is done by using the FILE command with an ASCII argument for the FILE NAME and a numeric argument as the START ADDRESS for that file. In this way, the FILE command can be used to create a new file as well as make an already existing file current. File names are kept in the system RAM area in a table called the "File Name Table." These names can also be removed from this list of defined files by using the FILE command; a numeric argument of zero erases the name from the table but does not affect the memory containing that file. These file parameters may be restored later with the FCHK command thereby allowing the user to actually have more than six files of information in memory at one time. The ALS-8 does not, however, keep track of more than six. The following shows three short files being created. Note that the FILE command used with no arguments returns a message to the terminal defining the Current File, its start and end addresses.

```
FILE/ONE/ 100

ONE 100 100    (RETURNED BY ALS-8)
1 This is the first line of file ONE.
26 THIS IS THE SECOND.
29 Line 3
FILE /TWO/ 200

TWO 200 200
```

```
FILE /THREE/ 6A1

THREE 6A1 6A1
10 Dear John,
12      Pay me or I won't be
14 your friend.
15           See you soon,
17            Igor
FILE /TWO/

TWO 200 200
1300 File Two gets this line.
1984 UPPER CASE OK.
1000 lower case ok.
2710 End TWO
FILE

TWO 0200 20C0
```

This example points out a number of requirements and features omitted in the discussion so far. Line numbers, for instance, are normally followed by a blank but this is not required by the editor functions. The example also illustrates the fact that line numbers do not have to be absolutely consecutive numbers. File line numbers are always decimal and must lie in the range 0 to 9999.

A file, "TWO" in the example, can be entered into the File Name Table and saved during the definition of "THREE" although it is empty. Later it can be made the current file and information can be entered into it.

Files naturally have a length as well as a start location and the user must be careful that, in adding text to a file, he does not accidently write file information over a program or another file. The ALS-8 assumes that the user knows where file information and programs are located. To help the user manage his files, the ALS-8 provides three file related commands: FILES (different from FILE), FMOV, and FCHK.

The FILES command produces a listing of the files in the File Name Table. This listing includes the start and end addresses for the files so it is a simple matter for the user to spot and avoid memory conflicts. Should a memory conflict threaten, the current file can be moved to a different location in memory with the FMOV command. FMOV requires only a single argument defining the destination address for the Current File. This argument may not be zero, but no other restrictions are placed on it.

The last of the file related commands is FCHK which verifies the internal structure of the Current File and updates the file end address if necessary. If for any reason the file is not properly formatted in memory, FCHK will send the message "FILE ERR" to the terminal. This command can be very useful in restoring files. Earlier it was mentioned that the contents of a file were not affected by removing the file's name from the list of defined files. Assuming that subsequent operations have

not altered the memory contents for that file's information, FCHK can return it to an active, useful status. Similarly, the contents of a previously saved file could be ENTR'ed into memory and reactivated with FCHK. The following example shows some typical uses of FCHK.

```
FILE /COPY/ 700    define a file name.  Leave empty.

COPY 700 700
FILE /OLD/ 600     define file "OLD". store program in it.

OLD 600 600
10 WAIT  IN 377
15       CMP A
20       JZ WAIT
25        RET
40        END
FMOV 700          move OLD to start of COPY.

OLD 700 736

FILES
OLD 700   736     OLD is O>K>
COPY 700  700
FILE /OLD/ 0      delete OLD from list.
FILES             check defined files.

COPY 700   700    only COPY. thought to be empty.
FILE /COPY/       make it the current file.

COPY 700 700
FCHK              redefine end address.
COPY 700 736
FILES

COPY 700   736
FILE /NEW/600     data from OLD still 600 to 636.
FILES

NEW 600   600
COPY 700  736
FCHK             examine file starting at 600.
NEW 600 636
FILE             check Current File, "NEW".

NEW 600   636    contents recovered.
```

CHAPTER VI - EDIT COMMANDS

The ALS-8 contains a number of editing commands designed to manipulate the contents of a file. All of these commands operate on the Current File so the user is cautioned to check the status, and perhaps identity, of the Current File before using these functions. This, as described in the last chapter, can be done with the FILE command. All the EDIT commands use decimal line numbers as arguments where required. (NOTE: These commands are separate from the optional VDM EDITOR package, TXT-2, sold by Processor Technology.)

The EDIT command set contains two commands designed to print the contents of the Current File: LIST and TEXT. The LIST command outputs the Current File ordered by increasing line number. It accepts up to two arguments defining the start and stop line number for the printing. If only one argument is given, the LIST function assumes that it is only to print the single line identified by the first argument. When both arguments are omitted the entire file is printed. The following example exercises these options. (Examples show formatted output.)

```
FILE  /XMPL/   1A2B
0 WAIT EI
10  JMP WAIT+1
20 *   THIS SETS INTERUPT AND WAITS
24  END

LIST  10   50
0010            JMP   WAIT+1
0020 *   THIS SETS INTERUPT AND WAITS
0024            END

LIST  0
0000  WAIT      EI

LIST
0000  WAIT      EI
0010            JMP   WAIT + 1
0020  *      THIS SETS INTERUPT AND WAITS
0024            END
```

The TEXT command is very much like LIST; the only difference is that its output omits the line numbers. This feature is generally used for files containing regular text as opposed to program code. This allows letters, notices, or papers to be printed without line numbers. Since the user must specify line numbers for arguments in edit commands, the the TEXT command obeys the argument conventions used for LIST. The following shows the last example reprinted using TEXT.

```
TEXT
WAIT     EI
         JMP    WAIT+1
*  THIS SETS INTERUPT AND WAITS
         END
```


The ALS-8 system RAM has two parameters pertaining to LIST and TEXT, the formatting flag and the terminal width parameter. "Formatting" refers to the spacing or layout of the printed results from the two functions. A formatting "flag" parameter is a word in system RAM which tells LIST or TEXT whether or not they should rearrange the contents of each line in a form expecially suited to assembly language output. This parameter is controlled by two system commands, FORM and NFOR, which indicate "formatting" and "no formatting" respectively. Naturally a file not containing a program is more readable when not formatted. The FORM and NFOR commands require no arguments and the parameter set by them remains in effect until explicitly reset by the user.

The terminal width parameter, set by the command TERM, contains an integer which represents the line width for the current output device measured in characters. This parameter has no influence on LIST or TEXT when the formatting feature is suppressed. When formatting output for either output command, the terminal width value determines the extent of formatting. When it is less than 80 minimum formatting is performed, while above 80 the maximum takes place. Terminal Width also controls the maximum length of input lines as well as the acceptable line length during FCHK.

The DELT command allows the user to delete a line or group of lines from the Current File. It accepts one or two arguments identifying the first and last line numbers of the group to be deleted from the file. When used with only one argument, DELT assumes that it is only to delete the single line designated by the first argument. The ALS-8 executive, however, rejects line numbers input with no follwing characters and this is a simpler way of deleting a single line. Thus line 40 in the following can be deleted with "DELT 40" or simply 40 followed by a carriage return.

```
FORM
FILE

A      0280 02AF

LIST   36   44

0036   DUP   LXI   H,0
0039         DAD   SP
0040         SHLD  HOLD
0044         RET
DELT   40

LIST   36   44

0036   DUP   LXI   H,0
0039         DAD   SP
0044         RET
```

The last command in the edit set is RNUM which renumbers a file given a start line number and increment. When finished the Current File´s line numbers will begin with this first number and all adjacent line numbers will differ by the value of the second argument. If the second argument is omitted, the RNUM function will use five as the increment. The largest value allowed for this increment is twenty-five. The RNUM function also will change the increment to one if the line numbers exceed 9000. The example below shows a small program being renumbered.

```
LIST

0025  INSTAT   IN     TTS
0030           ANI    DR
0035           JZ     INSTAT

RNUM 8000 10
TEST  1000 1030

LIST

8000  INSTAT   IN     TTS
8010           ANI    DR
8020           JZ     INSTAT
```


CHAPTER VII - I/O DRIVERS AND COMMANDS

The term "I/O driver" refers to a routine used to transfer textual data between the ALS-8 routines (or user routines) and an associated input or output device. Its basic duties are to interpret a request for data transfer from some calling routine and to translate it into a sequence of reads or writes suited to the conventions assumed by the electronics of the external device. This relieves the calling routine from the responsibility of handling separate conventions for many devices. Conceptually, an ALS-8 routine can ask for data in the same way from any input device or send data to any output device in the same way. It must formulate the request and simply choose the routine to handle the request and the device.

The ALS-8 has a table of driver routines in its system RAM area and a parameter identifying the current pair of drivers (input and output). When and ALSL-8 function requires input or output of a character, it uses this parameter to choose the proper driver. The table for these routines contains a name and pair of addresses for each entry. The IODR command nandles entries to and deletions from this table as well as defining the "current" driver and printing out the table's contents. Used with a name argument of one to five characters and two numeric arguments obeying the current value of MODE, the IODR command will enter the name and addresses into the table. If used with no arguments at all, IODR prints the contents of the table. Since drivers are selected as pairs, special functions can be implemented such as read from high speed paper tape both with, and without, printout. Entries can be deleted by using IODR with the entry name as an argument followed by a single zero argument. The example shows IODR being used in these ways.

```
IODR /TAPES/  DF00   DF40

TAPES DF00 DF40

IODR /TVTWT/  DF80   DFC0

TVTWT DF80 DFC0

IODR
SYSIO E200    E240
TAPES DF00    DF40
TVTWT DF80    DFC0

IODR /TVTWT/  0

IODR
SYSIO E200    E240
TAPES DF00    DF40
```

SYSIO, shown in the above, is the default I/O driver which handles the main terminal. It remains the current driver until another driver from the list is explicitly defined by IODR in yet another form; IODR with just a name argument. Making a driver "current" assumes that the corresponding routines are loaded and ready for use because the subsequent ALS-8 commands will have switched to using those addresses for I/O. Assuming that "TAPES" in these examples represents drivers for a cassette recording unit, data could be loaded into memory with the following:

```
IODR  /TAPES/
ENTR  200
(the ENTR function will retrieve data from the cassette
 and not the terminal keyboard)
```

The discussion on drivers so far has covered only the basic duties of drivers. Because the system only has to know where the routine starts, the programmer has an enormous amount of flexibility. The driver is a program capable of handling any number of devices in a single call if desired. It has access to system parameters and tables so it can check status words or find file information. When used with functions like ENTR, the driver can accept data in whatever form the device will provide it and then reformat it so that the necessary address and colon are appended to the start of each line. There is also no restriction that more than one driver can't be assigned to a single device. One line printer driver might simply echo the data given to it on the page. Another driver in the list might count lines so it can automatically skip the paper folds and print headings at page tops. Similarly, a set of drivers could exist for comunication with the VDM as within the TXT-2 extension package.

These capabilities are further enhanced by the fact that any user program has access to the driver list. It can, if desired, ignore the "current" driver pair, search the table for a specific name, retrieve the corresponding addresses and begin using those routines. To write such a program, the user must know the addresses of the table, the parameter identifying the current driver, and the ALS-8 routines which search tables. The conventions for the routines and memory storage must also be learned but the enormous flexibility compensates for the trouble.

CHAPTER VII - SYSTEM COMMANDS

The commands described in this chapter cover a wide range of functions. ASSM, ASSI, and their derivatives assemble a program and load the resultant machine instructions into a designated section of memory. CUST and its derivatives, CUSTE and CUSTD, manipulate the Custom Command Table stored in system RAM. SYML, SYMLE and SYMLD are like the CUST set except that they manage the System Symbol Table in the system RAM. Other commands in this group define I/O drivers, set system parameters, and execute routines starting at user defined addressees.

All of the commands related to the ALS-8 resident asembler accept one or two arguments. The first argument defines the origin for the program while the second, if given, specifies the start address for the machine language output of the assembler. If only one arguement is given the asembler uses it for both the program origin and the start address for the binary form of the program. The binary machine language output by the assembler is known as "object code;" it is the only form executable by the 8080 CPU. The program text by contrast is not executable but much more readable for humans and is called "source code."

The set of assembler related commands ASSM, ASSME, ASSMX, ASSMS, ASSI, ASSIE, ASSIX, ASSIS all produce assembled object code programs for the program source code. Each has, however, its own option associated with it. The fourth and, where applicable, the fifth character in these command names are used to select the options to be used on a particular assembly run. The fourth character, M or I, divides the group into two sets of four commands. These sets differ in the source they use for program text. The M group uses the Current File as its source; whereas, the I group reads the source program through the CURRENT INPUT DRIVER. The fifth character of the assembly command names control options for the assembler output listing. If ommitted, as in ASSM or ASSI, the listing is a one output line per source line printout identifying errors, addresses, and machine language values produced from the program's instructions. An E suffix suppresses all printout except for those lines containing errors. S and X suffixes list the contents of the symbol table immediately following the program source listing. The X option adds cross reference information between program symbol names and the line numbers they occured in. The output listing of the assembler is formatted depending on the parameter defining terminal width AND THE "FORM" SWITCH.

The CUST command prints out the current contents in the Custom Command Table. The custom names must be four or five characters and are considered unique to only four characters. When a custom name is given to the ALS-8 as a command, this address is retrieved from the table and the ALS-8 passes control to this address (as a subroutine call). Entries to this table are made with the CUSTE command which requires an ASCII argument to be used as the new name and an address to be called for the command. The address argument follows the base set by the last MODE command. CUSTD deletes custom names from the table. It requires only the single name argument. Users are cautioned that the twenty-two custom name limit is their responsibility to watch as the ALS-8 does not warn when the number of entries exceeds the table's boundry.

Custom commands can be attached to any kind of program. The FOCAL and BASIC software packagess both load starting at address zero so they cannot be in the machine at the same time. Either could be loaded, though, and its name entered as a custom command. Both software packages come with a short program which must be ENTR'ed first; this program loads INTEL format paper tapes. This loader is then started and the paper tape data is stored in memory. The following outlines such a sequence.

```
MODE 16
ENTR 1800
(type in hexicdecimal for INTEL paper tape loader)
/
CUSTE /LOAD/ 1800
LOAD
(start paper tape-when done reading restart ALS-8 at E060)
READY
CUSTE /FOCAL/ 0
CUST
LOAD 1800     FOCAL  0
FOCAL
*  (this is the ready asterisk from FOCAL)
```

The System Symbol Table is managed with the SYML, SYMLE, and SYMLD commands, SYML, like CUST, only prints out the contents of the table. SYMLE and SYMLD enter and delete names and their associated values from the symbol table. SYMLE requires a name argument of five letters or less and a numeric argument representing the symbol's value. SYMLD handles the deletion of symbol names from the table and, like CUST, requires only the name argument. Unlike the custom table, the System Symbol Table is not restricted much by a maximum length. Its physical location allows it just over 3K of memory and it is all but inconceivable that this could be overrun. The user can effectively set a maximum length of his own by setting up other tables or drivers in this 3K expanse. The example here shows two important symbol names being entered into the System Symbol Table.

```
SYMLE /SP/ 6
SYMLE /PSW/ 6
SYML
SP    6  PSW 6
D30E  (End of Table address printed following listing)
```

The symbols shown in the example above are needed by the resident assembler for programs which access the 8080 Stack Pointer, "SP", or the Program Status Word, "PSW". The resident assembler can only recognize single letter register names like B, C, D, E, H, L, and A. The user can define the SP and PSW symbols in each program he writes or enter them once

in the System Symbol Table for all the assemblies he performs. The assembler produces a table for the symbols it finds in a program and this table, inaccessible to the user, is called the Assembly Symbol Table. It is created from scratch for each assembly. If the program instructions make reference to a symbol which has been given no value in the program itself, the assembler will try to fetch the value from the system's table. It is a great convenience then to be able to define symbols once in this System Symbol Table rather than each time in a program. This makes programs both shorter and more versatile as single changes in the symbol table values can affect the origins, parameters, or subroutine connections for a number of programs.

The ALS-8 allows the user the freedom of specifying where the Assembly Symbol Table should start in memory. The STAB command defines this location from an argument which obeys the current MODE value. This start location must be defined before the first assembly is made and it is suggested that this table be placed at D700 hexidecimal. This puts it well into the system RAM area leaving over 1K for the System Symbol Table. It also leaves over 2K for the assembly Symbol Table which is sufficient for all but the largest programs; this assumes naturally that the area between D700 and E000 is not full of I/O driver routines (see Chapter II). The following might be used to start an assembly.

```
STAB D700
ASSM 1A0
```

The loaded output of the assembler, the object code, can be executed without having to make an entry in the Custom Command Table., The EXEC command generates a subroutine call to the address specified by its argument. When finished, the program at this location only has to generate a return with the 8080 RET assembly instruction and control will return to the ALS-8 executive. The argument to the EXEC command naturally follows the number type specified by the MODE parameter. In an earlier example, the name "FOCAL" was entered into the Custom Command Table with an associated address of zero. When "FOCAL" was given as a command the address 0 was given control by the ALS-8. This could also have been done by giving the command EXEC 0".

In the event that a program does not automatically return to the ALS-8, it will be necessary to stop the machine from the front panel, set the address switches to E060 and hit the RESET, EXAMINE, RUN switches. FOCAL, BASIC, and INTEL LOADER are examples or programs which normally do not have an ALS-8 return. If a user program goes awry the same procedures can be used to restart the ALS-8. The user may want to check his files and data to ascertain whether or not they have been damaged by the errant program.

This chapter contains a summary of the ALS-8 commands in the order they were presented. The reader is advised to consult earlier chapters for any details omitted here. Following chapters will cover the ALS-8 assembly language instruction set. The descriptions given here use the convention of enclosing an argument in parentheses when it is optional. Arguments will be signified by lower case names suggestive of their use; "addr1" for instance, will be an argument representing an address.

ENTR addr

This command reads numeric data from the current input driver and stores it in consecutive memory locations starting with the address specified by the argument. The data may continue for any number of lines; the function will return control to the ALS-8 executive only when it encounters a slash (/). At the beginning of every line, the current address pointer can be changed by specifying a new value followed by a colon (:). Both the data and addresses are interpreted in octal or hexidecimal according to the currently defined MODE. The length of any input line is limited by the current value of terminal width.

DUMP addr1 (addr2)

This command displays the contents of memory from "addr1" to address "addr2". If only one argument is given, only the contents of address "addr1" are displayed. The arguments and printed results obey the number base set by MODE.

MODE base

The argument "base" for this command sets an ALS-8 parameter which is used in converting binary data to readable form. The argument is decimal and must be either 8 for octal or 16 for hexidecimal. All ALS-8 arguments representing memory data or addresses will be affected by this command. Arguments which specify setting terminal width or line number will always be decimal. Initially the ALS-8 assumes a mode of 16.

FILE COMMANDS

The FILE command has many different forms each with its own distinct function. THe following describes each partuclar form. All name arguments may be one to five characters long.

FILE

This form will print the name of the current file, its start address and end address.

FILE /fname/

This will search through the current list of file names for "fname". When found, this file will be marked as the current file and all subsequent file operations will be made on it. If not found, the error message "WHAT" is sent to the terminal.

FILE /fname/ addr

This enters a file name, "fname", into the list of names kept in the file table. The argument sets both the start and stop addresses associated with the name. If the file already exists in the table an error message FCON is output to the SYSIO output device. The file "fname" always becomes the Current File. Address "addr" must not be zero.

FILE /fname/ 0

File "fname" is removed from the file table and forgotten. There wil be no Current File when this command is finished.

FILES

The FILES command uses no arguments. It lists the names, start and end addresses for all the files known by the ALS-8. This command does not affect the status of the Current File.

FCHK

This command checks the structure of the Current File. It begins at the start address contained in the file table and continues until it finds an end of file mark (01 hexidecimal) or an error. An error is signaled with the message "FILE ERR." followed by the address of the error. The location of the end of file mark becomes the end adress of the Current File. Using FCHK files may be input directly into memory from magnetic tape or disc and recreated.

FMOV addr

The Current File is moved by this function to memory locations starting at "addr". The start and end address values associated with the file are also changed. The copy remains the Current File and an FCHK is automaticaly performed. If the file was inadvertently moved to a location without memory a new file can be created at the old address and the contents recovered using the FCHK command.

While there is no restriction prohibiting a file from being moved to an address contained by the original, the user should note that only the copy will have a valid structure after such a move.

Text can be input to a file by simply specifying the line number and contents for that line. The line number is an

...normally is followed by one blank. If the file contains a line with this same number, the new data is entered in place of the old. The contents of any file can be interpreted as text or as assembly language source. Lines intended for the assembler are composed of distinct fields which are separated by groups of blanks. These fields can be repositioned during printout by an automatic formatting feature controlled by the TERM, FORM, and NFOR commands. The TERMINAL WIDTH parameter also controls the maximum length of lines input to the file.

TERM width

The ALS-8 parameter representing terminal width is initially set to 80. The user can, however, reset this at any time with the TERM command. The decimal argument "width" contains the size of the terminal line. This influences not only output formatting, but also input line length for files (FCHK). The maximum value for TERM is 120.

FORM

This command sets a parameter in the system RAM for the ALS-8 which specifies whether or not printed listings of assembler source or files are to be formatted.

NFOR

This deactivates the formatting feature described above. The ALS-8 is initialized to the non-formatted state.

LIST line1 (line2)

This is used to print out contents of a file between the specified line numbers. When only one argument is used, the single line identified by "line1" is printed. Line numbers and line number arguments are always decimal numbers. This command prints the contents of each line following the corresponding line number. (When using the optional VDM EDITOR the LIST command will list files entered without line numbers.)

TEXT line1 (line2)

Like LIST, this command prints file contents from "line1" to "line2". It does not, however, print out the line numbers at the start of each line. This is a useful feature for letter copy. Both TEXT and LIST contain the formatting routine which is controlled by FORM, NFOR, and TERM.

DELT line1 (line2)

DELT removes a line or series of lines from the Current File starting at line number "line1" and continuing through "line2". In its single argument form, only the line specified by "line1" is deleted; it is usually easier to delete single lines, however, by typing the line number followed by just a carriage return.

RNUM line# (increment)

RNUM renumbers the Current File so that its first line number will be "line#" and each successive line number will be greater than the last by the quantity defined in "increment." If "increment" is omitted, RNUM will use a default increment of five. The largest allowable value for the increment is twenty-five and, regardless of increment value at the outset, RNUM will use an increment of one after the line numbers reach 9000. RNUM ends by calling FCHK thereby checking the file after renumbering.

ASSEMBLER COMMANDS

The ALS-8 resident assembler is activated with different options from the eight commands sumarized below. Each requires an origin which is used as the address from which the routine must eventually be run. The second argument to each of these commands is the start address for the storage of the assembled program. A program "origin" and "load point" must agree if it is to be run rather than temporarily stored. The variations in the commands mainly affect listing length and input source.

ASSM origin (load address)

This form assembles from source contained on the Current File. If the "load address" argument is omitted, the assembler will load at the address given by "origin." A full listing of the assembly and errors is written to the current output driver.

ASSME origin (load address)

This is the same as ASSM except that only lines containing errors are listed.

ASSMS origin (load address)

This form produces a full listing and adds a listing of the assembler's symbol table to the end. The current values, usually addresses, of the symbols are also given.

ASSMX origin (load address)

This is a further expansion of ASSMS in that the symbol table listing provided at the end is cross referenced to file line numbers. The summary for each symbol then contains its name, value, and a list of locations which used it.

The four remaining assembler commands ASSI, ASSIE, ASSIS, ASSIX are similar to the four commands just listed except for the source of the assembly language code. These four use the I/O driver selected by IODR for reading the program source. A special driver is required for this use and the user is referred to the ALS-8 Specification sheet outlining the requirements of this driver.

```
ASSI  origin  (load address)  assemble with full listing.
ASSIE origin  (load address)  assemble. list only errors.
ASSIS origin  (load address)  assemble. list with symbol table
ASSIX origin  (load address)  assemble. list with cross
                                   reference table.
```

STAB address

This command sets the starting location for the Assembler Symbol Table. This address is not initialized to a usable value so this command must be called before any assemblies are attempted.

CUST

This will print out the contents of the Custom Command Table. Each output line will contain name and address pairs. The addresses are printed according to the base by MODE and the end address of the table is printed following the list of names.

CUSTE /cname/ address

This will enter the name, "cname", into the Custom Command Table with its associated address value. If this name already exists in the table, it is merely given a new associated value. The name may be four or five characters long, but it is only unique to four. Thus "HEART" is the same custom name as "HEAR." A maximum of twenty-two such names is permitted each requiring eight bytes of table space. The table must not go beyond D300 or interference with the System Symbol Table will result.

CUTSTD /cname/

This deletes the specified name from the Custom Command Table.

EXEC addr

The EXEC command performs a subroutine call to the address specified by "addr." The argument, being an address, obeys the number convention set by MODE.

SYML

This command lists the contents of the System Symbol Table. The values listed in the name/value pair are assumed to be addresses and, as such, will follow the current MODE for type. The names can be one to five characters in length. The end address of the table is printed following the list of names and values.

SYMLE /sname/ addr

SYMLE is used to enter a name and its corresponding value into the System Symbol Table.

SYMLD /sname/

This will delete the symbol, "sname", from the System Symbol Table.

I/O DRIVER COMMANDS

There are only two names in the I/O driver command set but one, IODR, has many forms. The following summarizes its functions and describes the other command, SWCH.

IODR /dname/ in out

This form of IODR enters the name "dname" into the I/O driver table with the two addresses, "in" and "out." When this driver pair becomes active, the ALS-8 functions will try to read text data through a routine located at the address "in." Similarly, output from these functions will be sent to the routine assumed to be at address "out." This form of the command does not activate this driver pair, only defines it. If address "in" is zero, followed by a proper output address, the current SYSIO input driver will be assigned as the input driver. Also, if the output driver address is zero the current SYSIO output driver will be assigned. If the output address is omitted, after being preceeded by a valid input address, a special output address will be assigned to allow no output. (BIT BUCKET)

IODR /dname/ 0

This will remove the driver pair "dname", from the table. A maximum of six driver pairs can be defined in the I/O driver table at any one time.

IODR

Used without arguments, this command prints out the contents of the I/O driver table. Each line of the printed summary contains the name, the input driver address, and the output driver address.

IODR /dname/

This informs the ALS-8 that the default system driver, SYSIO, is to be used for one more command line. The driver pair, "dname", is then used until an ALS-8 command returns control to the executive. This one command delay enables the user to choose an ALS-8 function from his terminal before switching control to the new drivers. SYSIO, the terminal driver pair, is automatically reactivated at the conclusion of the ALS-8 function or under error conditions.

SWCH

When used after the above form of IODR, the new drivers are activated for use by the ALS-8 executive, not an ALS-8 function. The executive then will read a command and any associated data with these drivers before returning to SYSIO.

CHAPTER X - THE ALS-8 ASSEMBLER

The resident assembler is perhaps the strongest feature of the ALS-8. It is a program designed to convert the text for a program into the binary machine code form of a program. The textual representation, called "source code," is very readable by humans but only binary form is executable by the computer hardware. In typical use the source program is written onto a file and edited. This is then assembled with one of the ASSM commands and the resultant binary, or "object code," is stored in memory. There it can be used as a driver, a custom command, or a program to be run by the EXEC command.

A source program written in assembly language is interpreted by the assembler on a line by line basis. Since files are also line structured, they become a natural storage area for program source. (The ASSI command series insures that ALS-8 files are not the only storage medium for programs.)

Each line of the program must conform to certain rules in order to be assembled correctly. An asterisk at the start of a line identifies the line as being a comment and its contents are not subject to the rules of the assembly language. Lines without an asterisk are "statements" and these can be divided into as many as four separate parts called "fields." Each field has an entirely different function to the assembler. The first, the "label field," gives a symbolic name to that line which can be referenced by any statement in the programs. The label must start with an alphabetic character in column 1 of the line (after any file line numbers). It may be any number of continuous characters though the assembler will ignore all characters beyond the fifth. This means that the label names "bridge," "bridg," and "bridget" will all represent the same label. All fields are separated from one another by one or more blanks.

STATEMENTS may contain either symbolic 8080 machine instructions or pseudo-ops. The four fields of each statment, NAME OPERATION, OPERAND and COMMENT are scanned left to right by the assembler. The assembler requires at least one blank

NAME OPERATION OPERAND COMMENT

between each field for identification. For automatic formatting however, the comment field must be preceeded by at least TWO BLANKS. Instructions which use only the operation field as does RZ should be followed by a "dummy" operand if comments are to be used with the statement. Blanks in the following example are shown as dashes "-" for clarity.

```
RZ-.--COMMENTS ADDED AFTER TWO SPACES
```

CONSTANTS
****

The ALS-8 Assembler allows the use of constants within the operand field. Both hexidecimal and decimal as well as octal constants may be used. When using either octal or hexidecimal the value should be followed by a "Q" or "H" to indicate OCTAL and HEX respectively. When a value does not include a following identifier it defaults to DECIMAL but a "D" may be used for clarity when desired.

```
MVI A,128      Move 128 decimal to register A.
LXI H,2FH      Move 2F hexidecimal to registers H&L.
MVI B,40Q      Move 40 octal to register B.
JMP   0FFH     Jump to address FF hexidecimal.
```

As shown by the last example, all constants must begin with a numeric quantity. When hexidecimal values begin with the letters A-F they should be preceeded by the numeric value zero.

EXPRESSIONS
****

An expression is a sequence of one or more SYMBOLS, CONSTANTS or other expressions separated by arithmetic operators. The ALS-8 Assembler allows the use of four primary operators. ADDITION (+), SUBTRACTION (-), MULTIPLICATION (*), and DIVISION (/). Expressions are scanned left to right with no precedence given to any operator. Calculations are made using 16 bit arithmetic (modulo 65536) and overflow of values is allowed. Single byte values for immediate instructions (as with MVI A) must evaluate to a value between -256 to +255 or an assembler error will result.

```
MVI     A,255D/10H
LDA     POTTS/256*OFSET
LXI     SP,30*2+STACK
```

ASSEMBLER ERROR INDICATIONS
****

The following error flags are output by the assembler when the error occurs. As determined by the type of error, some of the flags are output during pass one to indicate an invalid assembly.

O -- OPCODE ERROR	The symbol found in the operation field was not recognized as a valid 8080 instruction or pseudo operation of the assembler.
L -- LABEL ERROR	The symbol found in the name name field contains improper characters.
D -- DUPLICATE LABEL	Two labels with the same name within the assembly.
M -- MISSING LABEL	Instruction requiring a label doesn´t have symbol in name field
V -- VALUE ERROR	Expression in operand field is ouside range required.

U -- UNDEFINED SYMBOL	Name given for operand cannot be found in symbol tables.
S -- SYNTAX ERROR	Syntax of statement does not follow the requirements of the assembler.
R -- REGISTER ERROR	False name given to register.
A -- ARGUMENT ERROR	Argument for operand improper.

Since the label field is optional, the assembler must have a convention for identifying the second type of field, the operation field, when the label is missing. The operation field must, for this reason, be preceded by at least two blanks when it starts a line. The contents of this field will be a two, three, or four letter mnemonic chosen from the assembly language set. This mnemonic defines the general instruction to be assembled and it uses, where necessary, the third field, the "operand", to modify or complete the instruction. An "ADD" in the operation field tells the assembler that one of the 8080 registers is to be added to the 8080 accumulator.

The fourth possible field is the comment field which, as its name implies, is reserved for comments. The assembler, then, disregards anything after the third field. In statements which have no operand field, it is a good idea to preceed the comment with a period followed by two blanks. Since no operand is required the period has no affect and the listing will be properly formatted. Most of the examples in this chapter are listed as though they were formatted and printed by the TEXT command. The example below shows how a sample program file might actually be input and exist in memory. Blanks are written as "-" to show their significance; File line numbers are also shown.

```
3-*-THIS-SUBROUTINE-SHIFTS-(H,L)-CIRCULAR-LEFT
5-LUP-XRA-A--CLEAR-THE-CARRY
8--CMP-B--SEE-IF-SHIFT-COUNT-DOWN
13--RZ-.--RETURN-TO-CALLING-ROUTINE
14--DCR-B--DECREMENT-COUNT
16--MVI-A,80H--TEST-MSB-OF-HL
22--ANA-H--COMMENTS-OPTIONAL
24--DAD-H--SHIFT-LEFT
26--JZ-LUP--IF-MSB-WAS-ZERO
29--INX-H--CIRCULAR-BIT-IN
35--JMP-LUP
40--END
```

The above illustrates the fact that "column 1" of each program statement line must be separated from the file line by at least one blank. When printed with the TEXT function the above becomes:

```
* THIS SUBROUTINE SHIFTS (H,L) CIRCULAR LEFT
LUP     XRA    A       CLEAR THE CARRY
        CMP    B       SEE IF SHIFT COUNT DONE
        RZ     .       RETURN TO CALLING ROUTINE
        DCR    B       DECREMENT COUNT
```

```
        MVI     A,80H   TEST MSB OF H,L
        ANA     H       COMMENTS OPTIONAL
        DAD     H       SHIFT LEFT
        JZ      LUP     IF MSB WAS ZERO
        INX     H       CIRCULAR BIT IN
        JMP     LUP
        END
```

Instructions in the assembly language manipulate seven 8-bit registers, a 16-bit program counter called "PC", memory, I/O devices, and a 16-bit stack pointer "SP". Both the assembler and the hardware use a number convention for identifying these registers. The numbers 0,1,2,3,4,5, and 7 each represent one of th 8-bit registers. Depending on the instruction, a 6 can represent memory, the stack pointer, or a special program status word, "PSW". Many of the instructions assume a destination register for the results they generate and many will also make assumptions on one of their input operands. Addition, for example, is handled by the ADD instruction in the assembly language and it assumes that the contents of register 7, called the accumulator, will be added to an eight bit quantity from memory (6) or the registers (1 through 5). Its result always goes to register 7. The operand for this register is a number specifying which 8-bit value is to be added to register 7. This operand appears in the operand field for the instruction as shown.

```
LABL    ADD     7       DOUBLE THE ACCUMULATOR
        ADD     0       ADD IN REGISTER 0
XAD     ADD     3       ADD IN REGISTER 3
        ADD     6       ADD IN VALUE FROM MEMORY
```

The assembler us3s a pair of tables, the Assembler Symbol Table and the System Symbol Table, to find number values associated with a symbol name. Label names from the label field are stored into the Assembler Symbol Table along with the addresses they represent in the object code. Assembling the short example above would have added the names "LABL" and "XAD" to this table. The assembler always has eight entires in this table, B,C,D,E,H,L,M, and A, for which it has the values 0 through 7. These are the names given to the registers and the assembler will replace one of these names found in an instruction with the appropriate register number. The last example could be rewritten:

```
LABL    ADD     A       DOUBLE THE ACCUMULATOR
        ADD     B       ADD IN REGISTER B
XAD     ADD     E       ADD IN REGISTER E
        ADD     M       ADD IN VALUE FROM MEMORY
```

A number of the 8080 operations use pairs of registers for 16-bit operands and, for these operations, register B is paired with C, D with E, H with L, and the program status word PSW is paired with A. B, D, H, and PSW are the high order bytes in these values. Tne instruction DAD, for instance, performs a "double add" between the (H,L) pair and the (B,C) or (D,E) pair. the result is stored again in (H,L). For these instructions, the pair is designated the name of the most significant byte so the possible DAD instructions are:


```
B
D
H
H
SP
```

:h are quivalent to:

```
0
2
4
6
```

that 6, which could represent memory, SP, or PSW, is :he DAD instruction hardware to mean the stack pointer. · "DAD PSW" are equivalent to "DAD 6" and will then be ʳ the hardware as "add SP to (H,L)." Note also the the .st of register names does not include PSW or SP. be entered into either the System Symbol Table with executive command or into the Assembler Symbol Table :QU assembler instruction (to be described). The will first try to fetch a value for a symbol from its and, failing, will then try the System Symbol Table.

ıber of the 8080 instructions are "conditionals" ıat the full operation is performed only if a conmet. The program status word, PSW, uses five of its ı to represent the testable conditions. Tnese bits l Sign, Zero, Aux, Parity, and Carry, and they reflect of the accumulator after certain instructions. The ficant bit of the accumulator is copied to Sign by ıstructions. Similarly, certain instructions will set ıit (to 1) when the accumulator contains a zero value reset (to 0) when A is non-zero. Parity is set to 1 ıtains an odd number of binary 1's and is reset when : Carry bit's function is most easily described with ıtual aid of a ninth bit on the accumulator. Some ıns will put the opposite (0 for 1; 1 for 0) of the ıe into Carry; others will copy carry into Carry. ' is again reminded that some instructions do not : values in PSW regardless of the contents of A. ıs taken by each instruction concerning the PSW con-.s will be given with the description of each ın.

ıe upcoming instruction summary, two types of instructions will be described: executable ıns and "pseudo-ops." The executable instructions assembly statements which must be converted into ect form for eventual execution by the CPU. :, or pseudo-operations, have the appearance of other atements but do not produce object code for the CPU. ıey are used to pass information to the assembler self. "ORG" for instance, is used with its operand the "current address counter" for that position in the ing assembled. "END," another pseudo-op, signals the · assembly language source code; the assembler will not d or interpret lines beyond the line containing "END."

```
MVI     A,80H     TEST MSB OF H,L
ANA     H         COMMENTS OPTIONAL
DAD     H         SHIFT LEFT
JZ      LUP       IF MSB WAS ZERO
INX     H         CIRCULAR BIT IN
JMP     LUP
END
```

Instructions in the assembly language manipulat 8-bit registers, a 16-bit program counter called "PC I/O devices, and a 16-bit stack pointer "SP". Both assembler and the hardware use a number convention f fying these registers. The numbers 0,1,2,3,4,5, and represent one of th 8-bit registers. Depending on t tion, a 6 can represent memory, the stack pointer, c program status word, "PSW". Many of the instruction destination register for the results they generate a also make assumptions on one of their input operands for example, is handled by the ADD instruction in th language and it assumes that the contents of registe the accumulator, will be added to an eight bit quant memory (6) or the registers (1 through 5). Its resu goes to register 7. The operand for this register i specifying which 8-bit value is to be added to regis operand appears in the operand field for the instruc shown.

```
LABL    ADD     7       DOUBLE THE ACCUMULATOR
        ADD     0       ADD IN REGISTER 0
XAD     ADD     3       ADD IN REGISTER 3
        ADD     6       ADD IN VALUE FROM MEMORY
```

The assembler uses a pair of tables, the Assemt Table and the System Symbol Table, to find number va associated with a symbol name. Label names from the are stored into the Assembler Symbol Table along wit addresses they represent in the object code. Assemt short example above would have added the names "LABI to this table. The assembler always has eight enti table, B,C,D,E,H,L,M, and A, for which it has the va through 7. These are the names given to the regist assembler will replace one of these names found in a tion with the appropriate register number. The last could be rewritten:

```
LABL    ADD     A       DOUBLE THE ACCUMULATOR
        ADD     B       ADD IN REGISTER B
XAD     ADD     E       ADD IN REGISTER E
        ADD     M       ADD IN VALUE FROM MEMORY
```

A number of the 8080 operations use pairs of r 16-bit operands and, for these operations, register with C, D with E, H with L, and the program status paired with A. B, D, H, and PSW are the high order these values. Tne instruction DAD, for instance, p "double add" between the (H,L) pair and the (B,C) o the result is stored again in (H,L). For these ins the pair is designated the name of the most signifi the possible DAD instructions are:


```
DAD     B
DAD     D
DAD     H
DAD     H
DAD     SP
```

which are quivalent to:

```
DAD     0
DAD     2
DAD     4
DAD     6
```

Note that 6, which could represent memory, SP, or PSW, is taken by the DAD instruction hardware to mean the stack pointer. "DAD M" or "DAD PSW" are equivalent to "DAD 6" and will then be treated by the hardware as "add SP to (H,L)." Note also the the default list of register names does not include PSW or SP. These may be entered into either the System Symbol Table with the SYMLE executive command or into the Assembler Symbol Table with the EQU assembler instruction (to be described). The assembler will first try to fetch a value for a symbol from its own table and, failing, will then try the System Symbol Table.

A number of the 8080 instructions are "conditionals" meaning that the full operation is performed only if a condition is met. The program status word, PSW, uses five of its eight bits to represent the testable conditions. Tnese bits are called Sign, Zero, Aux, Parity, and Carry, and they reflect the state of the accumulator after certain instructions. The more significant bit of the accumulator is copied to Sign by certain instructions. Similarly, certain instructions will set the Zero bit (to 1) when the accumulator contains a zero value and it is reset (to 0) when A is non-zero. Parity is set to 1 when A contains an odd number of binary 1's and is reset when even. Tne Carry bit's function is most easily described with the conceptual aid of a ninth bit on the accumulator. Some instructions will put the opposite (0 for 1; 1 for 0) of the carry value into Carry; others will copy carry into Carry. The reader is again reminded that some instructions do not affect the values in PSW regardless of the contents of A. The actions taken by each instruction concerning the PSW condition bits will be given with the description of each instruction.

In the upcoming instruction summary, two types of assembler instructions will be described: executable instructions and "pseudo-ops." The executable instructions are those assembly statements which must be converted into binary object form for eventual execution by the CPU. Pseudo-ops, or pseudo-operations, have the appearance of other program statements but do not produce object code for the CPU. Instead they are used to pass information to the assembler program itself. "ORG" for instance, is used with its operand to define the "current address counter" for that position in the program being assembled. "END," another pseudo-op, signals the end of the assembly language source code; the assembler will not try to read or interpret lines beyond the line containing "END."

ASSEMBLY LANGUAGE INSTRUCTIONS

This section describes the assembly language instructions and their function ordered by increasing complexity. An alphabetically ordered summary will be given later with the object codes generated for each instruction. In the following description, optional fields will be enclosed in parentheses and operands for the instructions will be represented by a short lower case mnemonic. The operand "reg" represents any constant, symbol, or expression with a value from zero to seven. This value is used to select one of the seven registers or memory: B, C, D, E, H, L, M, A. Operand "addr" can be an expression, constant, or symbol which gives a value to be used as a 16-bit argument, usually an address. A numeric argument is represented by "const 8" and "const;" values supplied for "const 8" must be 8 bits or less in magnitude.

The following three instructions provide the most direct means of transferring 8-bit data from register to register, memory to register, or register to memory. There is no single instruction to transfer from one memory location directly to another.

(label) LDA addr - LOAD Accumulator

This instruction fetches a byte from the memory location specified by "addr". This value is then stored in A. PSW is not affected.

(label) MOV dreg, sreg - move register to register

This instruction moves the contents of the source register, "sreg", to the destination register, "dreg". B, C, D, E, H, L, M, and A (0 through 7) are legal values for "sreg" and "dreg" except that both may not specify memory (M). When either "sreg" or "dreg" specify memory, the CPU uses the contents of the (H,L) register pair as the address of the memory byte to fethc or store. Tne contents of the source register are not affected. PSW is also not affected by the instruction.

```
        MOV   M,E   move contents of E into memory
                    location specified by (H,L).
MOVER   MOV   E,B   copy B into E
        MOV   C,M   load C from memory
```

(lable) STA addr - STORE accumulator

STA transfers the contents of the accumulator to the memory location specified by "addr". PSW is unaffected.

Arithmetic, logical, and comparison operations are handled by eight instructions. Each of these operations is assumed to take place between the accumulator and a register (or memory location) specified in the operand field. All, except CMP, produce an 8-bit result which is placed in the accumulator. The program status word bits in PSW are all affected by any of these instructions.

(label) ADD reg - ADD register to accumulator

The value in register "reg" is added to the accumulator and PSW is updated. PSW "Carry" is set to 1 if the arithmetic produces an overflow from the most significant bit (MSB).

(label) SUB reg - Subtract register from A

This instruction subtracts the value specified by "reg" and placed the result in A. The PSW carry bit is set to 1 if a borrow was necessary during the subtraction; the actual ninth bit carry, discussed earlier, would actually be zero in a borrow situation. This is an example of carry being inverted for storage in Carry.

(label) ADC reg - Add the specified register and Carry to the accumulator

The specified register and the current contents of Carry are added to A and the result is placed in A. This is used primarily in "multiple precision" additions in which a number is actually contained in several (usually adjacent) memory locations. Such an addition starts at the low order end of the two numbers with the Carry bit reset to zero. Successive additions with ADC on more significant bytes in the numbers are corrected for overflow from the last (less significant) addition.

(label) SBB reg - Subtract with borrow from A

This is the multiple precision form of SUB. It subtracts the Carry (borrow) from A as well as the value in "reg". This is actually done by adding the Carry bit to the value in "reg" befpre the subtraction is made. Tne PSW status bits are updated after the subtraction.

(label) ANA reg - logically AND reg and A

This function performs a "logical and" (a boolean multiplication) on the contents of "reg" and the accumulator. Conceptually this operation is performed independently on each bit position of the two operands (A and "reg"). The correspondig bit position in the result is set to 1 if and only if both of the operand bits are 1's. 00110011 and 01010101 will leave the value 00010001 in A. The Carry bit is always reset; other status bits are set or reset according to the result.

(label) ORA reg - logically OR reg and A

This instruction performs a bit-wise "logical or" (boolean) add) on the accumulator and the specified. Each bit of the result is set to 1 if either of the corresponding operand bits is 1. 00110011 OR 01010101 will produce 01110111 for a result. The Carry bit is always reset to zero. Other status bits are set as dictated by the properties of the result.

(label) XRA reg - logical EXCLUSIVE OR reg and A

XRA is a bit-wise logical "exclusive-OR" function for the operands, A and "reg". Each bit of the result will be 1 if one and only one of the corresponding operand bits is 1. The operand values 00110011 and 01010101 produce an "exclusive-OR" result in the accumulator of 01100110. PSW status bits are handled as in ANA, ORA. This function is often used to clear the accumulator and Carry with an "XRA A".

(label) CMP reg - Comapare reg to A

This instruction performs the internal subtraction A-"reg" but does not store the result. It is used to set the PSW status bits. "CMP A" is often used to update the status bits when the value in A has been fetched from memory. Note that the Carry bit conventions will follow a seven bit signed compare and that zero is greater than 255. For a full 8 bit compare the SUB instruction snould be used.

There are eight instructions much like the register operations described above and they are called the Immediate Instructions. They differ from register operations in that a register (or memory) value is not used as an operand. Instead the operands are the accumulator, as before, and an eight bit value which is given in the operand field of the instruction. This operand value may be the result of an expression, the value of a symbol, or a constant, so long as the magnitude of the value does not exceed eight bits. As with register operations all PSW bits are affected by these instructions.

(label) ADI const8 - add value of const8 to A

The 8-bit value of "const8" is added to the accumulator. as in ADD, its register operation counterpart, all PSW bits are affected.

(label) SUI const8 - subtract immediate from A

The immediate value is subtracted from A. PSW bits, including Carry, follow conventions of SUB.

(label) ACI const8 - add value and Carry to A

"Const8" and the Carry bit are added to A. PSW is affected.

(label) SBI const8 - subtract immediate with borrow

This instruction subtracts Carry bit and immediate value.

(label) ANI const8 - AND the immediate with A.

ANI performs a logical AND on the immediate value and the accumulator. It is often used to isolate certain bits in A for testing. THe logical operation is described in ANA.

(label) ORI const8 - immediate OR with A

This function performs a logical OR on the immediate Value and register A.

(label) XRI const8 - immediate exclusive OR on A

This produces an exclusive-OR result from A and the value following. See XRA.

(label) CPI const8 - compare immediate with A

The CPI instruction performs a signed, seven bit, compare of register A and the immediate value following.

There are several other commands which affect the contents of the 8 bit registers. They have been separated since they behave differently with respect ot the program status word, PSW. Note that these instructions affect some condition bits and not others.

(label) MVI reg,const8 -move value into register

This instruction is similar in some ways to the immediate instructions though it does not affect the PSW. The 8-bit value of "const8" is moved into the specified register.

(label) INR reg - increment register

The register specified by "reg" is incremented by one and all the PSW bits EXCEPT CARRY are updated.

(label) DCR reg - decrement register

The register, or memory location addressed by the H & L registers, is decremented by 1. As with INR all PSW bits except carry are affected.

(label) CMA - complement the accumulator

This instruction reverses each bit of the accumulator. 1´s become 0´s and 0´s become 1´s. The PSW is not affected.

There are four instructions used to shift the contents of of accumulator. Each of these instructions shifts the contents only one place left or right depending on the particular instruction. None of the shipts affect any PSW bits except carry. The direction "right" or "left" in these descriptions assumes that the more significant bits of the accumulator lie to the left.

(label) RLC - rotate left, through carry

This is a circular left shift in which the carry bit receives the bit value shifted from the most significant bit. of the accumulator. This same value shifted into carry is also shifted into the least significant bit of A. 01101110 becomes 11011100 after the shift and the Carry bit is left as 0. Another shift of this value gives 1011001 and a Carry value of 1.

(label) RRC - rotate right, through carry.

This shift is a right shift similar to RLC except the least significant bit is shifted to Carry and the MSB position.

(label) RAL - 9 bit shift left.

This function shifts the accumulator one place left. The most significant bit is shifted into Carry as in RLC, but the old value of Carry is shifted into the low end of the reg A. Shifting 01101110 with a value of 1 in Carryproduces 11011101 and a Carry of 0. Asecond shift of this data produces 10111010 and a Carry of 1.

(label) RAR - 9 bit right shift

The accumulator contents are shifted one place right with the least significant bit being sent to Carry and the old value of carry being shifted into the MSB of the accumulator.

(label) LDAX negbd - load A from memory

The accumulator is loaded with the value from memory whose address is obtained from the register pairs (B,C) or (D,E). The operand, "negbd", can then only equal "B" or "D".

(label) STAX negbd - store A into memory

The contents of A are stored in memory at the address given by the (B,C) or (D,E) register pairs. The pair is chosen by the operand "negbd" which may only be "B" or "D".

The 8080 is also equipped with a full set of "transfer instructions" which have the ability to alter the flow of a program through execution. There are three categories of transfers: "jumps", "subroutine related instructions", and "interrupt transfers". Of the ten jump instructions, only two are "unconditional transfers" meaning that the execution sequence of the program is always altered by them. The "conditional transfers", on the other hand, examine the status word PSW to see if the proposed jump is to be made. If the condition bits of the PSW do no meet the requirements of the instruction, no transfer is made and the program will resume execution at the next instruction in memory.

UNCONDITIONAL TRANSFERS

(label) JUMP addr

This instruction always transfers control to the address in memory specified by the operand field, "addr". The next instruction to be executed will be the one starting at this address.

(label) PCHL

This performs the same function as the JUMP instruction except the address for the transfer is taken from the H and L pair of registers and not the operand field. Generally this instruction is used to branch to a routine in memory whose address has been located in a table. It could be used to branch to a computed address but any small errors in the calculation could produce some mysterious bugs.

CONDITIONAL TRANSFERS

(label) JZ addr

JZ examines the status bit "ZERO" of the PSW and transfers to the address "addr" if this bit is set to 1. This 1 in the ZERO bit represents a zero value in a register at the last time the condition bits were set by an instruction. Most of the instructions affecting the PSW reflect the status of the accumulator, register A, though a few (INR DCR) will change the ZERO bit and others when their result goes to any of the registers.

(label)   JNZ   addr

This instruction also examines the ZERO bit of the PSW but it transfers when the last pertinent result was a non-zero value. A non-zero result resets the ZERO status bit to 0.

(label)   JP   addr

JP examines the SIGN bit within the PSW and transfers when this bit is zero. A zero for the SIGN bit represents a positive value for the last pertinent operation.

(label)   JM   addr

JM examines the SIGN bit and transfers when it represents a negative value (minus) for the last result.

(label)   JC   addr   - jump if CARRY

This instruction jumps if the CARRY bit has been set on the last operation. For addition operations, a jump is made if the sum of the two operands has exceeded the limit of 8-bit numbers. The overflow bit is stored in the PSW bit, CARRY. Subtractions requiring a "borrow" will also set this CARRY.

(label)   JNC   addr   - jump if no CARRY

A jump to the address, "addr", is made if the last operation did not produce a CARRY.

(label)   JPE   addr   - jump if PARITY even

The PARITY bit of the PSW is "even" when the number of bits set to 1 in the result is even. This instruction transfers to "addr" when this condition exists.

(label)   JPO   addr   -jump if PARITY odd

JPO transfers to the address "addr" when the PARITY bit in the PSW represents a result with odd parity. Parity is generally used to verify data transmitted from an external device.

CARRY BIT INSTRUCTIONS

There are two special instructions which manipulate only the status of the CARRY bit in the PSW. These will affect all CARRY related conditionals as well as the addition, subtraction, and shift instructions which use CARRY. These two instructions are frequently used to return a simple status condition from a subroutine.

(label)   STC   - set CARRY (to 1)

This instruction sets the value of CARRY to 1. No other condition bits are affected by this command.

(label)   CMC   - complement CARRY

CMC reverses (complements) the current value of CARRY. If CARRY equaled 1, this instruction will change it to a 0. If CARRY was 0, CMC changes it to a 1.

SUBROUTINE TRANSFERS

A transfer to a subroutine is made with one of the CALL instructions described below. When a CALL instruction is made, two addresses become important. The "transfer address", the address of the subroutine being called, is contained in the operand field of the CALL instruction. Program control will be transferred to this address immediately following the call. As the call is being made, however, a "return address" is computed and stored on the next position of the stack. When the subroutine is finished it can execute one of the RETURN instructions which will fetch this address from the stack (pop the stack) and a jump will be made to this address. This return address represents the location of the instruction immediately following the call instruction which gave control to the subroutine. Subroutine calls within subroutines store their return addresses at successive stack locations so the corresponding return instructions can properly locate their return addresses.

As with the jump instructions, both the CALL and RETURN operations are divided into unconditionals and conditionals with the same suffix convention as used with JUMPS.

(label)   CALL   addr   - call the subroutine at "addr"

This instruction performs an unconditional subroutine call to the address specified by the operand "addr".

(label)   RET   - return to address found on stack

RET pops a value off the stack which it uses as a transfer address for a jump. Since it always retrieves its "operand" from the stack, it does not need anything in the operand field. This return is unconditional.

SUBROUTINE CONDITIONAL INSTRUCTIOS

The reader is reminded that only certain instructions influence the condition bits of the PSW (program status word). A full description is given at the beginning of this chapter.

(label)   CZ   addr   - call if last result equaled 0

This instruction calls the routine located at address "addr" if the ZERO bit of the PSW is set to 1 representing a zero result in the last operation.

(label)   CNZ   addr   - cal. if last result was non-zero

A call is made if the last PSW related operation produced a non-zero result.

(label) CP addr - call if result positive

This instruction examines the status of the SIGN bit within the PSW and performs a subroutine call if this bit indicates a positive result from the last instruction.

(label) CM addr - call if negative result (minus)

CM calls the routine at address if the SIGN bit is set representing a negative result from the last PSW related instruction.

(label) CC addr - call if CARRY

CC calls the subroutine at "addr" if the CARRY bit has been set to 1. CARRY is set to 1 when there is a carry from an addition, a borrow from a subtraction, or there is a bit 1 produced by one of the shift or Carry instructions.

(label) CNC addr - call if no CARRY

This instruction calls the subroutine at address "addr" if the CARRY bit is zero.

(label) CPE addr - call if PARITY even

This instruction calls "addr" if the PARITY bit was reset by the last PSW related operation. "Resetting" PARITY is equivalent to making it a zero. Even parity for a result indicates that it contained an even number of binary 1´s (and 0´s).

(label) CPO addr - call if PARITY =1,"parity odd"

The subroutine call is made if the PARITY bit of the PSW is set to 1 indicating "odd parity".

(label) RZ - return if last result was zero

A return from subroutine is made if the last result recorded in the PSW was a zero.

(label) RNZ - return if last result was non-zero

This instruction returns from the present subroutine if the last result was non-zero.

(label) RP - return if positive

A return, using the address pulled off the stack, is made if the last result had a zero sign (was positive).

(label) RM - return if minus

This returns from the routine if the last result was minus.

(label) RC - return if CARRY (=1)

This instruction performs a subroutine return if the PSW bit CARRY is set to 1. CARRY is set by the Carry instructions, shifts, additions with overflow, or subtractions with borrows.

(label) RNC - return if no CARRY (=0)

RNC returns if there was no CARRY generated from the last instruction. See the above instruction.

(label) RPE - return if PARITY even

A return is executed if the value of the PARITY bit is 0 indicating even parity in the last operation.

(label) RPO - return if PARITY odd

A return is made for PARITY of 1 indicating an odd parity.

Another instruction, RST, also performs transfers but it is rarely used as such. It will be described later with the interrupt related instructions.

16-BIT OPERATIONS

A number of the 8080 functions can perfrom arithmetic operations on 16-bit values stored in register pairs. The B and C registers form a pair as do D,E and H,L; the Stack Pointer, SP, is used as a fourth possible operand for these instructions. None of these instructions affect any of the condition bits.

(label) LHLD addr -load H,L with the values at "addr"

This instruction moves two bytes from memory into the H,L register pair. The operand, "addr", identifies the address of the byte to be transfered to the L register and the next memory address is used for H.

(label) SHLD addr -store H,L into memory at "addr"

The contents of the L register are moved to the address specified by "addr" and the contents of the H register are moved to memory location "addr+1".

(label) LXI rp,const -store 16-bit constant in pair "rp"

The register pair "rp" is given a 16-bit value as determined by the second operand, "const". Numerically the operand "rp" must equl 0,2,4,6 which are generally represented by the symbolic names B,D,H, SP. Either operand may be an expression acceptable to the assembler which will produce a register pair integer or a 16-bit value for those operand positions.

(label) INX rp - increment register pair "rp"

This instruction adds one to the register pair specified by the operand "rp". No condition bits are affected even if carries are produced internally for the operation.

(label) DCX rp - decrement register pair "rp"

DCX subtracts one from the register pair "rp". As with INX and the other 16-bit instructions, none of the condition bits in PSW are affected.

(label) DAD rp - add rp to H,L

This performs a 16-bit add between the operand register pair, "rp", and the H,L registers; the result is stored in the H,L pair. The operand can be B,C ("B"), D,E ("D"), H,L ("H"), or SP.

(label) XCHG - exchange the contents of D,E with H,L

SCHG swaps the contents of the D,E register pair with the contents of the H,L pair.

STACK OPERATIONS

The "stack" is an area in memory identified and manipulated through the 16-bit address held in the "Stack Pointer", SP. As previously described, it is used by the subroutine related instructions, "CALL" and "RET" (and their conditional relatives) In operation, a 16-bit value, an address for the subroutine instructions, is sent to tow memory locations identified by the address in the SP. The specific locations chosen are SP-1 for the "most significant" byte and SP-2 for the lower order byte. The SP contents are then decremented by two to be ready for the next stack operation. Such an operation is called a "push" and the reverse operation where data is removed from the stack is known as a "pop". Note that the pointer moves "down" in memory with successive pushes and moves "up" for pops.

The operations about to be described give the programmer direct control of the stack and its pointer. The stack can be a very versatile data storage area for particular applications, but the programmer must be careful that the data stored in the stack is not confused with the return addresses stored there from subroutine calls.

Two of the stack instructions use a register pair operand which will be denoted by "rp" in the following. This operand identifies B,C , D,E , H,L , and PSW,A. In the last case, the Program Status Word is placed at location SP-1 and the accumulator is placed at SP-2 for stack pushes. This form of saving the PSW is necessary for interrupt handling or some subroutine calling sequences.

(label) PUSH rp - push contents of rp onto stack

The contents of the register pair "rp" is placed on the stacvk and the poiinter, SP, is decremented by 2. Numerically "rp" must be 0,2,4,6 which represent the pairs, B,C D,E H,L and PSW,A.

(label) POP rp - pop data from stack into rp

Data is removed from the stack and placed into the registers identified by the operand "rp". The ordering of the bytes taken from the stack follows the same rules used for PUSH. The pointer SP is incremented by 2 at the end of the operation.

(label) SPHL - move H,L contents into SP

The contents of the H,L pair is moved into the stack pointer destroying its previous contents. This provides a convenient way of changing the SP during a program thereby allowing two or more stacks to exist at one (one data, one subroutine control, etc.). The SP is usually initialized by the LXI instructions.

(label) XTHL - exchange SP and H,L contents

The contents of the H,L register pair and the SP are swapped. The most frequent uses are outlined above in the SPHL description.

INPUT/OUTPUT INSTRUCTIONS

The two input/output instructions for the 8080, IN and OUT, both operate on the accumulator contents. The operand field is used to define a "device code" which identifies the external device which is to produce or receive an 8-bit value. This device number can be nay number between 0 and 377 octal. Each device attached to the computer has such a number assigned at the time it is wired to the machine and the device code given in the I/O command must equal that of the device before it will respond. Reading a non-existent device number with the IN instruction will put an octal 377 in the accumulator.

(label) IN dev - read device number "dev"

The external device with input device number "dev" will return an 8-bit value which is stored in the acculator. None of the PSW condition bits are affected. The default input device for the ALS-8 is assumed to be4 device 1 and its status (busy or idle) is accessible through input device 0.

(label) OUT dev - send contents of A to device "dev"

The contents of the accumulator A are sent to the output device numbered "dev". The ALS-8 assumes by default that an output device 1 exists and that its condition can be checked also through input device zero.

INTERRUPT RELATED INSTRUCTIONS

The 8080 is prepared to accpet signals from external devices whic can alter its program flow. This is invaluable for handling certain types of sporadic or slow devices. It can allow the CPU to work on aprogram without worrying constantly about the status of devices. This is accomplished with the aid of the "Interrupt Enable Flag", also know as "INTE". When this flag is on, "enabled", a device can force an interrupt which initiates a sequence of events in the computer. The "INTE" flag is immediately disable to keep other devices from confusing things while the first interrupt is being handled. The CPU is then required ot take an instruction (8-bits only) from the interrupting device, execute it and then copntinue. Special hardware can be attached to the computer which will cause the CPU to jump to any predetermined location in memory. Without this special "vetor interrupt" hardware, the normal convetnion has the interrupting device issue a Restart instruction which is a subroutine like jump to one of eight possible memory locations: 0, 10,20,30,40,50,60,70 octal. At the location specified by the vector hardware or the restart, there should be a subroutine capable of handling the interrupt condition. The restart instruction ("RST") pushes a return address onto the stack so the program which was operating can be properly resumed with an RET instruction executed in the interrupt routine.

(label) EI - enable interrupts

This instruction enables the interrupt flag, "INTE". Devices attempting to interrupt while this flag is disabled will be ignored by the CPU and its related hardware, INTE is automatically diabled when an interrupt occurs.

(label) DI - disable interrupts

This disables the interrupt flag preventing any devices from altering program flow with an interrupt. The computer is in the disabled state when the front panel switch "RESET" is activated. For machines with no interrupting devices, the INTE light on the front panel can be used by these instructions to signal certain program states like "program done" or "error".

(label) RST n - call routine at location n*8

This transfer instruction generates a subroutine call to an address which is computed from the operand "n". The operand, which must itself be between 0 and 7 in magnitude, is multiplied by 8 to produce one of the following addresses: 0,10,20, 30,40,50,60,70 octal. The subroutine call is then made to this

address with the return address being stored on the stack as in any other subroutine call. An "RET" in the subroutine located by the RST will return control to an address pulled from the stack. Devices using this instruction during interrupt put the 8-bit equivalent of this instruction on the data lines for the CPU to execute.

(label) HLT - halt the CPU and wait for interrupt

The CPU is completely stopped by this instruction and can only be reactivated by an interrupt. Should the interrupt flag happen to be disabled at the time this instruction executes, the whole machine must be reset from the front panel. The halt conditions is reflected in the front panel light marked "HLTA".

VARIABLE STORAGE AND THE NO OP

The instructions presented so far represent operations or functions within the 8080 hardware. The ALS-8 assembler converts the textual form of these instructions into a binary form which will be executed by the hardware. The assembler also recognizes a number of instructions which do not produce "executable" code. In general, this class of assembler instructions defines storage arrangements, addresses, or contents for the program under construction. These instructions are called "Pseudo-ops" (being "false" in the sense that they don´t produce executable code).

An instruction, the NOP, generates a binary instruction of zero which is ignored by the execution hardware. It is sometimes used in programs to "pad" areas of code where changes are expected to be made via the front panel. The versatility of the ALS-8 makes this unnecessary, but the instruction can still be used to generate zero bytes for variable storage. As will be shown, there are instructions from the pseudo-op set which can allocate blocks of memory for variables much more easily than successive NOP´s.

(label) NOP - do nothing. (reserve this space)

This assemlby language instruction corresponds to an operation code (binary) of zero which is ignored by the CPU when executed.

(label) DS amount - reserve an "amount" of memory

This pseudo-op reserves a number of successive memory locations starting at the current position in the program. The number of memory locations is determined by the operand "amount" which can be any 16-bit number, or equivalent expression. The contents of these locations is not defined.

(label) DB n - define contents for single byte

This instruction reserves a single memory location and defines for it a value as determined by the operand "n". The value of the operand must not exceed eight bits.

(label) DW n - define word and contents (16-b)

The operand for this instruction is evaluated as a 16-bit quantity and stored in two memory locations. The least significant byte of the quantity is stored at the "current address" and the most significant is stored below it.

(label) ASC #string# - put character string in memory

This puts a string of characters into successive memory locations starting at the current location and continuing until the entire string has been put in memory. The special symbols # at either end of the above example are called "delimiters"; they define the beginning and end of the Ascii character string. The assembler uses the first non-blank character found after the mnemonic "ASC" as the delimiter. The string is defined as starting immediately after the first delimiter and ending just before the second occurence of the delimiter. Characters to the right of the second delimiter are assumed to be comments. A carriage return will act as the second delimiter in cases where it is omitted. When formatting is used, the string must not contain two or more successive spaces within the first four characters.

(label) ORG n - define the origin

This instruction, used without a label, defines the "current address" value for the assembler. The next assembled instruction (producing executable code) will be converted to binary with the assumptio that it is to be loaded and executed at this address. The "current address" value is increased for each instruction by the number of memory locations used by that instruction. The ORG instruction may be used at any time to redefine this pointer.

(label) EQU n - assign value n to symbol "label"

The symbol in the label field for this instruction is entered into the assembler's symbol table with the 16-bit value found in the operand field. Note that both the label field and and operand field are required for this instruction.

(label) COM n -put value and symbol in System Table

The label field symbol is entered into the System Symbol Table along with the value obtained from the operand field. As with the EQU instruction, both of these fields are required. This is equivalent in every respect to the system command "Symle".

NLST - suppress printed output of assembly listing

This instruction sets a flag in the assembler which will

suppress the printing listing from this line until that flag is reset by the LST instruction. Neither NLST or LST may have a label field.

LST - begin assembly listing

This reactivates the listing feature which will remain on until turned off by NLST. If the listing feature is already active when this instruction is encountered, it is simply ignored. Neither NLST or LST affect memory position or contents in any way.

END - marks the end of the program

This instruction is a signal to the assembler that no more statements are to be assembled from the current device or file being assembled. For programs being assembled from a file in memory, this instruction is not necessary as the end of file mark performs the same function.

SIMULATOR EXTENSION PACKAGE
****

Temporary Operation Manual

The SIM-1 Extension Package for the ALS-8 is a program designed to "run" 8080 machine language in the same manner as the 8080 computer running the simulator program. Because the Simulator is an operating program, the user has full control of the "run" allowing powerful program debugging as well as a direct view of the computers operation. Since each instruction, as well as its effects can be viewed on a single step basis, the Simulator represents an ideal "teaching" machine for 8080 Micro-Computer operation.

By using the Simulator commands the user can modify or display storage, set simulated 8080 flags and registers, perform or test input and output operations, set and reset breakpoints and realtime run addresses as well as trace program flow.

The Simulator is entered from the ALS-8 by giving the SIMU command. On entry the program does a carriage return/linefeed on the last selected output device, followed by an asterisk prompt. The last selected MODE also remains in effect and is used by the Simulator.

After giving the prompt the simulator is ready to receive a command indicating the operation desired. Some commands, such as "run" (G for go), start operation of the software computer. Prior to running the program however certain commands allow the operator to set the PROGRAM COUNTER or REGISTERS in order to set the proper conditions for the simulation prior to the simulated computer start-up.

SET COMMANDS
****

P address(H,Q,D) --SET PROGRAM COUNTER

Set program counter to the value of "address". Conversion of the parameter is determined by the last selected "MODE" or by the following, optional, parameter.

S regx=value (regy=value..) --SET REGISTER VALUE

Set register"x,y.." to "value". Where value given according to MODE or following parameter (H-HEX,Q-OCTAL,D-DECIMAL). Multiple assigments per line are allowed however each register name must be followed by the equal sign and then the selected value. The next register name must then be preceeded by a space. Valid register names are A,B,C,D,E,H,L with "S" and "F" used to indicate the Stack Pointer and Flags (PSW) respectively.

All commands can be used any time the Simulator has given a prompt. While running, the program checks the front panel switches as well as the SYSIO input port for display and/or break indicators. Control "X" causes the Simulator to stop running and return to the command mode.

The two high-order sense switches determine the display mode of the simulator as it simulates the running program. If no breakpoint has been set these switches are interpreted as follows:

SWITCHES		DISPLAY MODE
**********		***
7	6	
0	0	SINGLE STEP MODE Execute one instruction and display on current output device. If C/X is input to the System input driver then return to the command mode. If any other character is received then execute and display one more instruction.
0	1	CONTINUOUS RUN (With Display) Execute and display each instruction until receiving C/X.
1	0	Execute and display one instruction then return to the command mode.
1	1	Force return to command mode from any Simulator condition.

The output display from the Simulator indicates the current status of the software 8080 as well as the current conditions of the program. The display is initialized to follow the last MODE setting but may be changed to decimal by giving a simulator mode command.

The display consists of the current location of the program counter followed by the FLAGS as set by the last instruction executed. These are then followed by each of the registers and the current memory location pointed to by the H & L registers. The stack pointer and instruction just executed then end the display. This is illustrated below:

```
PPPP CZSPI AA  BB CC DD EE HH LL  MM SSSS  B1 B2 B3
```

Where: PPPP -is the address of the simulated instruction. The display shows results following execution of the instruction.

```
      C  - Carry Flag   (0 or 1)
      Z  - Zero Flag
      S  - Sign Flag
      P  - Parity Flag
      I  - Interdigit Carry Flag

     AA  - Accumulator   (reg A)
     BB  - Register B
     CC  -          C
     DD  -          D
     EE  -          E
     HH  -          H
     LL  -          L

     MM  - Memory contents pointed to by HL.
   SSSS  - Current address of the Stack Pointer.

     B1  - Current instruction
     B2  - Byte two of the instruction (if used)
     B3  - Byte three of the instruction (if used)
```

In addition to this display the operator may dump selected memory locations or enter data to memory locations using the DUMP and ENTR commands.

D address (address)	This command dumps the contents of address to address following the conventions of the ALS-8 dump command.
E address	Enter data to memory following ALS-8 ENTR conventions.

The GO command starts the simulator at the current value of the program counter. It is used to initially start simulation as well as continuing after stopping.

G   Go-- Start simulation

X   Exit-- Return to ALS-8.

At this point the user is advised to write a short program and assemble it to a known location in memory. After obtaining a listing the Simulator commands described so far should be used in actual practice.

BREAKPOINTS AND "REAL TIME RUN" ADDRESSES

Running a simulation with the display on is normally used only through the problem areas of the program. In order to reach these areas, or to test values during a program loop, a BREAKPOINT is set to stop simulation and display only at the address given by the breakpoint. The breakpoint is not cleared at each display so program loops may be checked repeatedly by giving a new GO command following each display. Also, if single step operation is again desired, the breakpoint should be cleared prior to giving the GO command.

B address -- SET BREAKPOINT

Breakpoint is set to "address" and the simulator will display each time the program reaches this address.

CB -- CLEAR BREAKPOINT

The sense switches are interpreted as follows when a breakpoint is set:

SWITCHES		DISPLAY MODE
7	6	
0	0	Execute program until breakpoint is reached, display current status and return to command mode after giving prompt.
0	1	Same as above.
1	0	Execute only one instruction and return to command mode.
1	1	Unconditional return to command mode.

Some sub-routines require a speed of operation beyond that of the Simulator. In order to meet this requirement the Real Time mode of operation should be used. If the real time address is that of a 8080 CALL instruction the simulator will make a REAL TIME CALL to that location, effectively giving up control.

The subroutine must end with a valid 8080 RETURN instruction in order for the Simulator to return in control.

R address -- SET REALTIME RUN ADDRESS

CR -- CLEAR REALTIME RUN ADDRESS

INPUT INSTRUCTIONS

During simulation input operations can be performed in three different modes, SIMULATED, REALTIME and PRE-SET. Each method is used depending on the information needed by the user.

SIMULATED

If an input instruction is encountered during the simulation for a port defined as SIMULATED, the Simulator will stop and obtain input values from the operator. The following information is printed prior to receiving input:

INPUT PORT n=

Where "n" equals the port given in the program being run by the simulator. The simulator stops to the right of the equals sign and waits for input from the operator. Since input goes to the accumulator the value input must lie in the range 0-255.

REALTIME INPUT

If an input instruction is encountered during the simulation for a port defined as REALTIME the simulator will obtain the required input directly from the indicated port. This operation is identical to the standard 8080 obtaining input.

PRE-SET INPUT

The preset option allows any input port to have a value preset between 0 and 255.

OUTPUT INSTRUCTIONS

Program output, during simulation, can take one of three forms for any desired output port. These options, SIMULATED, ASCII or REALTIME are selected depending on the information required by the user.

SIMULATED

If a output instruction is encountered during simulation for an output port defined as SIMULATED the Simulator will indicate that an output has occured to the indicated port. This includes both the port number and output value as indicated below. (No actual output to the port occurs.)

OUTPUT PORT n=NN

Where "n" equals the port number and NN equal the value that would have been sent to the port.

ASCII OUTPUT

The ASCII output option is similar to Simulated output except the value "NN" is output as an ASCII character. If the value is a Control Character its output is identical to Simulated operation.

REALTIME OUTPUT

As implied, REALTIME OUTPUT sends the value to the indicated port just as though the actual 8080 was operating.

INPUT/OUTPUT COMMANDS
****

IC	portn	SET SIMULATED INPUT PORT Set "portn" to SIMULATED mode. (All ports are in this mode on first entry to the simulator)
IS	portn value	SET PRESET PORT Set "portn" to PRESET "value"
IR	portn	SET REALTIME PORT Set "portn" to realtime mode.
CI		Clear all input port assignments and set all to simulated mode.
OC	portn	SET SIMULATED OUTPUT PORT Set "portn" to Simulated. All ports are initialized to this mode on entry to the simulator.
OA	portn	SET ASCII OUTPUT PORT Set "portn" to simulated ASCII output.
OR	portn	SET REALTIME OUTPUT PORT Define "portn" as realtime port.
CO		CLEAR ALL OUTPUT DEFINITIONS

DISPLAY MODE
****

The display mode of the Simulator is normally determined by the ALS-8 MODE on entry to simulation. This being either octal or hexidecimal usually presents the proper information required by the operator. The Simulator has one additional display mode, DECIMAL, which can be selected at any time during simulation.

This mode command "M" will select Decimal output if it is followed by the value 10. (20 if entry mode was octal.) Any other value following the command will return to the default condition of entry.

OPTIONAL SIMULATOR ENTRY POINT
****

Often, during simulator operation, it is desireable to return to the ALS-8. In order to return to the simulator without clearing I/O port definitions it is required that the command SIMU followed by any non-blank character be used. SIMUS is recommended. This allows the exact conditions on exit to be restored upon re-entry.

OTHER SIM-1 EXTENSION FUNCTIONS
****

AUTO COMMAND

Every ALS-8 contains code to recognize commands other than the standard set. Auto is one such command whose actual operating code is contained in the SIM-1 Extension Package. (Making it rather dangerous for those without it to use the AUTO command.) In use the AUTO command allows input to standard ALS-8 files with the AUTO code adding the line numbers.

COMMAND FORM: AUTO (n)

When used without the optional parameter "n" the AUTO command will start sequencing line numbers beginning at one and incrementing by one for each additional line. If the optional parameter is included then line numbers will begin one beyond the last line in the current file. The parameter "n" can be any value between 0 and 7 with no significance placed on what the value is. Return from the driver to the standard ALS-8 is made by depressing the "ESC" key as the first character of a file. (Note if there are NO LINES IN A FILE do not use the optional parameter.)

As a note of interest the code comprising the AUTO command represents a special I/O driver implemented to preprocess input from the selected I/O driver. This of course is a driver on top of a driver but then the ALS-8 was designed for such nonsense.

TXT-2 EXTENSION PACKAGE
****

PRELIMINARY OPERATORS MANUAL

The TXT-2, an optional extension to the ALS-8, opens a new dimension to the powerful file operation and management of the ALS-8. In addition to an EDITOR the TXT-2 also contains a VDM output driver and the FIND command. Code for one additional function is also within the package though the name of the command is not known to the ALS-8 (a minor matter). The use of these commands will be described following the description and operating proceedure of the EDITOR.

EDITOR
****

The TXT-2 converts the contents of the "current" ALS-8 file into a contiguous display on the VDM screen. Single letter control character commands allow cursor, as well as direct file line movement, on the screen. Since all file operations are direct and the contents of the file are always displayed on the screen, editing becomes a simple matter either with or without file line numbers.

Upon entry, the EDITOR program takes control of the current ALS-8 File and displays the file contents (or lack there of) on the screen sixteen lines at a time. Command keys are provided to roll through the file or to position the cursor over any character within the file. (Even in a position where none exist) Also provided are controls to insert and delete characters, or lines as required by the result desired.

As with all memory files, a file beginning and end address exist. The TXT-2 EDITOR also has one additional parameter, a value indicating the end of assigned memory. This parameter can be given any value and is used to prevent a file from growing beyond assigned bounds.

The editor is entered by using the EDIT command of the ALS-8. The current file is displayed on the screen and if there are less than sixteen lines, a number of fill characters. As lines are added these fill characters disappear off the bottom of the screen.

Since a file must first exist, the user must create or select a file prior to entering the editor. The ALS-8 FILE command is used for these operations.

In the explanation that follows the user is urged to try each command on an actual file. No words can describe the visual effect each operation performs on the screen. For best "learning" results the file should have, or be given, at least thirty-two lines.

Prior to using the editor the end of assigned memory parameter should be set to a known value. The parameter can be set to a null value by giving the command EXEC FFED (HEX). This nulifies the proper operation of the parameter and a further explanation will cover the correct usage later in the manual.

CONTROL/ M (Carriage Return) scroll up and insert one line

Carriage return scrolls up one line and inserts a blank line in the file. The cursor is moved to the first character position of the new line.

OTHER COMMANDS
****

CONTROL/ F exit command

On EXIT the editor clears the screen and does an FCHK on the file prior to returning to the ALS-8 executive. For long files some delay may be experienced (about 1/2 second) before receiving the "READY" message.

CONTROL/ Y repeat command

The repeat command requires two keystrokes following the command. The first represents the command or character to be repeated, while the second is the number of repeat increments.

The repeat increment is offset by an ASCII bias to allow the numbers 1-9 to represent their actual values. All other characters have an equivalent value as determined by their ASCII representation.

CONTROL/Y----->> COMMAND OR CHARACTER------->> # OF REPEATS

OTHER FUNCTIONS PROVIDED BY THE EXTENSION PACKAGE

FIND
****

As was mentioned the TXT-2 extension also contains code for the ALS-8 FIND COMMAND. Tnis command gets an input string from the user and prints all occurances of the string within the current file.

After receiving the FIND command, followed by a carriage return, a colon (:) prompt will print on the current output device. At this point the desired string is input, once again followed by a carriage return. Following this all occurances of the string will print out on the current output driver.

ESET COMMAND
****

The VDM EDITOR uses a parameter to limit the maximum address the file may reach. Code has been included within the TXT-2 to set this value but no corresponding command has been provided. The standard ALS-8 CUST command can be used to insert this command if the following sequence is executed:

CUSTE /ESET/ FFED

After this the command ESET, followed by and address, will set the parameter to the value of the address given. It should be noted that the file may reach but not exceed this value.

VDM DRIVER

Also included in the TXT-2 package is a driver to allow the ALS-8 to use the VDM as an output device. This driver is in PROM allowing access at all times. The address for the driver is FE77 (hex) and the IODR command is used to enter the name in the DRIVER TABLE. For use as a stand-by driver the following sequence is reccomended.

IODR /VDM/ input address FE77

The driver may also be made the SYSTEM DRIVER by using the following sequence:

IODR /SYSIO/ 0 FE77

The standard terminal output driver can then be assigned as a hard-copy supplemental driver by using the following:

IODR /PNTR/ 0 D0A9

The VDM driver is especially suited to commanding the ALS-8 and it is reccomended that it be changed to the SYSIO driver right after system initialization. The following special keys are implemented in the driver.

CONTROL/ Z clear screen
A turn cursor on or off
S set display speed prior to operation

The display speed command will output the message: SPEED? on the VDM screen whenever it is given. The user should respond with a value between 1 and 9 indicating the display speed desired. A value of 1 represents aprox 2000 lines per second, while 9 is rather slow at 3 characters per second.

The speed may also be changed any time during output by pressing the corresponding key between 1 and 9. The display can also be stopped by depressing the "space bar". Once stopped any character other than speed values or another space bar will continue the output at the same speed. The space bar will allow one character to be printed for each sequential space character received.

During all output operations with either the standard ALS-8 terminal driver or with the VDM driver, a test for the ESC character is made. If received, all output will be discontinued and a return made to the SYSIO driver with a "READY" message.