ALTAIR 8800

THEORY of OPERATION MANUAL & SCHEMATICS

TABLE OF CONTENTS

MITS INC.

"Creative Electronics"

P.O. BOX 8636

ALBUQUERQUE, NEW MEXICO 87108

INTRODUCTION

The ALTAIR 8800 computer system is designed around Intel's 8080 microprocessor. The Intel 8080 is a complete central processing unit (CPU) on a single LSI chip using n-channel silicon gate MOS technology. This chip uses a separate 16-line address and 8-line bidirectional data bus configuration to greatly simplify design.

The ALTAIR 8800 uses a 100 line bus structure for data transfer between the CPU and memory or I/O devices. This bus structure contains all of the data and address lines, along with the unregulated supply voltages and all control and status signal lines. Cards other than the CPU will have control of the bus only when addressed by the CPU.

The schematic diagrams for the ALTAIR system are located at the end of this section. Specific schematics will be referred to in each particular section of the theory of operation.

On the schematics for each particular board components are identified by letters for the integrated circuits (A, B, C, etc.), and letters and numbers for the resistors and capacitors (R1, R2, C1, C2, etc.). Specific pins on the IC's are identified by numbers external to the symbol for that particular IC. The boxed numbers next to signal lines with arrows that exit or enter a given schematic refer to the bus number for those signals. Other notations on the schematics are self-explanatory.

CPU BOARD OPERATION

The 8800 CPU Board is the "heart" of the ALTAIR system. This board contains the 8080 microprocessor chip, bus drivers, the system clock, miscellaneous gating logic and the system status latch.

Refer to the following schematics for the CPU Board operation: 880-101, 880-102 and 880-103.

BUS DRIVERS

All signals entering or leaving the CPU Board are buffered using 8T97 tri-state drivers. These signals include: 16 address lines through IC's B, C and 4 gates of D; 8 data output lines through IC E and 2 gates of D; 8 data input lines through IC F and 2 gates of H.

The terms "in" and "out" are always defined with respect to the processor. Note that the 8080 bidirectional bus is split at the processor into an input and an output data bus.

The address and data out drivers (IC's B, C, D and E) can be disabled using the ADDR DSBL and DO DSBL bus signals through 2 of the gates of IC M. The data in drivers (IC F and 2 gates of H) are enabled under control of the processor through one gate of IC R, etc. (see schematic).

The 8 status output signals are buffered using 8T97's (4 gates of G and 4 gates of H). These signals are SINTA, SWO, SSTACK, SHLTA, SOUT, SMI, SINP, and SMEMR. The STATUS DSBL signal can be used to disable these outputs.

The 6 command/control output signals are also buffered using an 8T97 (IC J). These signals are SYNC, DBIN, WAIT, WR, HLDA, and INTE. The C/C DSBL signal can be used to disable these outputs.

The 4 command/control input signals (READY, HOLD, INT and RESET) are buffered using 4 gates of the 8T97 IC I. Note that the PRDY and PHOLD signals are synchronized to the leading edge of the Ø2 clock. This is required since the transition of either of these signals during the second half of Ø2 will cause the processor to enter an undefined state.

SYSTEM CLOCK

The ALTAIR 8800 system clock employs a standard TTL oscillator (IC P) with a 2.000 MHz crystal as the feedback element. The correct pulse widths and separations for the two phases are obtained using the dual single-shots (IC Q) and the delay circuit (R43 and C6). The 8080 processor requires a 12 volt swing on the clock. This is accomplished using a 7406 driver (IC N). TTL clock levels are sent to the system bus using 8T97 drivers (2 gates of IC I). The $\overline{\text{CLOC}}$ signal is sent to the system bus through one gate of the 8T97 IC G.

GATING LOGIC

The only external gating logic on the CPU Board consists of IC O (3 gates) and IC R (1 gate). If we define the output on IC O pin 13 to be G1 ENB (Data Input Enable), then:

G1 ENB = (DBIN + HLDA) • (RUN + SS) *

Further, if we define G1 DSB = $\overline{\text{G1 ENB}}$; then the output of IC R pin 8, which is the disable input for the input data drivers, is:

DI DSB = G1 DSB + SSW DSB

G1 DSB, as can be seen from the schematic, is a processor generated signal. When the 8080 is ready for input data, it will allow G1 DSB to go low thus enabling the input data drivers.

$\overline{\text{SSW DSB}}$ is a signal generated on the Display/Control Board. This signal is used to disable the input data drivers when an input from the sense switches (device address 377_8) takes place.** This is necessary since the sense switch inputs are tied directly to the bidirectional data bus at the processor.

SYSTEM STATUS LATCH

The system status latch consists of IC K (8212). At the start of each machine cycle the processor places the system status on the bidirectional data bus. When SYNC and Ø1 are coincident, this data is latched by IC K and remains latched for the remainder of the machine cycle.

*In these notations, + means <u>or</u>, and • means <u>and</u>.

**This address is listed in octal format. It is the same as the decimal address "255" listed in the assembly manual.

DISPLAY/CONTROL BOARD OPERATION

The 8800 Display/Control Board provides the operator with RUN/STOP and Single Step control of the processor. It also allows him to examine and modify the contents of any location in memory using the front panel switches.

Refer to the following schematics for the Display/Control Board operation: 880-104, 880-105 and 880-106.

The primary function of the D/C Board is controlling the ready line (PRDY), or a combination of the PRDY and the bidirectional data bus, to allow the above functions to be performed. Control of the PRDY line is exercised at IC O (7430). The output of IC O pin 8 (PRDY) logically appears:

PRDY = RUN + SS + EXM + EXM NXT *

For the ready line to be released one of these inputs to IC O must go high. The circuitry preceding IC O will insure that only one of these signals is high at any given time.

RUN/STOP

The RUN/STOP circuit consists of an R-S flip-flop and gating to establish the stop condition. The RUN/STOP flip-flop exercises control over PRDY as described above through its Q output. The gating insures that a STOP will occur when DO5, Ø2 and PSYNC are true and the STOP switch is depressed.

SS

The Single Step circuit consists of a dual single shot (IC M) for debounce and the SGL STP flip-flop (R-S type). When the machine is in a stopped mode, depressing the SS switch will set the SGL STP flip-flop. (The machine must be stopped for any of the front panel switches except RESET to be active.) This allows PRDY to go high. The machine will execute one machine cycle and PSYNC, on the next cycle, will reset the SGL STP flip-flop. This will pull PRDY low, stopping the machine.

EXM

The Examine circuit consists of a dual single shot (IC L) for debounce, a 2-bit counter (IC J), the top 3 sets of 7405's on schematic 880-106 (IC's A, B, C and 2 gates of D), and some gating.

* In this notation, + means <u>or</u>.

When the Examine switch is depressed the counter (IC J) is started. On the first count, a jump instruction (JMP 303) is strobed directly onto the bi-directional data bus at the processor. This is accomplished by enabling 2 gates of IC C and 2 gates of IC D through the output pin 6 of one gate of IC T. These open collecter gates then pull down data lines D2, D3, D4 and D5. This puts a 303 on the data bus, which is the code for a JMP.

On the second count, the settings of switches SA 0 through SA 7 are strobed onto the data bus in a similar manner to the JMP instruction through IC A and 2 gates of B. This provides the first byte of the JMP address.

The third count strobes the settings for switches SA 8 through SA 15 onto the bus. This provides the second byte of the JMP address. The processor will then execute the JMP to the location set on the switches SA 0 through SA 15, allowing the examination of the contents of that particular memory location.

The fourth count resets the counter and pulls the EXM line low, which in turn pulls PRDY low and stops the processor.

EXM NXT

Examine Next operates in the same manner as Examine, except a NOP is strobed onto the data lines through 4 gates of IC D and 4 gates of IC E. This causes the processor to step the program counter.

DEP

The Deposit circuit places a write pulse on the MWRITE line and enables the switches SA 0 through SA 7. This causes the contents of these eight switches to be stored in the memory location currently addressed.

DEP NXT

The Deposit Next circuit simply causes a sequential operation of the EXM NXT and the DEP circuits.

1K STATIC MEMORY BOARD OPERATION

The 8800 1K Static Memory Board is designed around the Intel 8101 256 X 4 bit static RAM. Two of these RAM's provide 256 8-bit bytes of memory. The board may be configured with a minimum of two 8101's (256 bytes) and may be expanded in increments of 256 bytes by adding pairs of 8101's up to 1024 bytes.

In addition to the RAM's, the board includes 4 circuit units: Address Decoding, Processor Slow Down Circuit, Memory Protect Circuit and Buffers and Buffer Disabling Gating.

Refer to the following schematics for the 1K Static Memory Board operation: 880-107 & 880-108.

ADDRESS DECODING

The address decoding circuitry is in the lower left corner of schematic 880-107. Address bits A10 through A15 are used to select a particular 1K of memory, using IC A and IC B. By patching the inputs of IC B to either the "1" or "0" address inputs for A10 through A15 a board can be assigned any address for a 1K block from 0 to 63.

Address bits A9 and A8 are used to select a particular 256 bytes within the 1K on the board. The gating (IC D, IC F, 2 gates of IC C and 4 gates of IC E) forms a standard 2 to 4 line decoder.

PROCESSOR SLOW DOWN CIRCUIT

Since the 8101 RAM's require 850 nanoseconds for stable data on a read output, it is necessary to insert 2 wait cycles (1us) when the processor reads data from memory.

This is accomplished by IC K, 2 gates of IC N and 1 gate of IC C. This circuit causes the output from pin 8 of IC K to go low for approximately 2 clock cycles starting with PSYNC. If the 1K card has been addressed, and the processor is in a memory read cycle, two of the drivers of IC H will be enabled. This will transmit the low from IC K pin 8 to PRDY on the bus; which will in turn cause the processor to wait for 1us for the data from memory to stabilize.

MEMORY PROTECT CIRCUIT

The Memory Protect circuit consists of an R-S flip-flop (IC L) which can be set or reset by the PROT and UNPROT lines from the system bus when the card is addressed (CE is true).

When the flip-flop is set the pin 11 output of IC N is disabled and MWRITE pulses from the bus cannot get to the 8101's. A status signal, $\overline{PS}$, is returned to the front panel display via the system bus to indicate when the protect flip-flop for a particular memory card is set.

BUFFERS

The output drivers on the 1K board are 8T97 tri-state drivers (IC's J & H). Gating for enabling and disabling these drivers is accomplished with IC G and 1 gate of IC C.

The logic for this is as follows: *

$$\overline{G2} = SINP + SOUT + \overline{CE}$$

OR

$$G2 = \overline{SINP} \bullet \overline{SOUT} \bullet CE$$

AND

$$\overline{G1} = \overline{SMEMR} + \overline{CE}$$

OR

$$G1 = SMEMR \bullet CE$$

* In this notation + means "<u>or</u>" and • means "<u>and</u>".

POWER SUPPLY OPERATION

The 8800 Power Supply provides two +8 and + & - 16 volts to the system bus and the display/control board. These voltages are unregulated until they reach the individual cards. Each separate card has all the necessary regulation for its own operation.

Refer to schematic 880-109 for the Power Supply operation.

Transformer T1 provides +8 volts unregulated to the system bus. This voltage is rated at 8 Amps. All boards except the display/control board use this supply for their regulated +5 volts.

Transformer T2 provides two unregulated voltages; +8 volts rated at 1 Amp for the display/control board, and +16 rated at .8 Amps to the system bus.

Transformer T3 provides the -16 volt supply rated at .3 Amps to the system bus.

All of the AC and DC voltages are wired to a terminal block for distribution to the other boards.

8800 SYSTEM BUS STRUCTURE

The 8800 system bus structure consists of 100 lines. These are arranged 50 on each side of the plug-in boards. Refer to drawing # 880-110 for the following explanation.

The following general rules apply to the 8800 system bus:

SYMBOLS: "P" prefix indicates a processor command/control signal

"S" prefix indicates a processor status signal

LOADING: All inputs to a card will be loaded with a maximum of 1 TTL low power load.

LEVELS: All bus signals except the power supply are TTL

BUS DEFINITION

No.	SYMBOL	NAME	FUNCTION
1	+8V	+8 volts	Unregulated input to 5v regulators
2	+16V	+16 volts	Positive unregulated voltage
3	XRDY	External Ready	For special applications: Pulling this line low will cause the processor to enter a WAIT state and allows the status of the normal Ready line (PRDY) to be examined
4	VI0	Vectored Interrupt Line #0	
5	VI1	Vectored Interrupt Line #1	
6	VI2	Vectored Interrupt Line #2	
7	VI3	Vectored Interrupt Line #3	
8	VI4	Vectored Interrupt Line #4	

BUS DEFINITION

No.	SYMBOL	NAME	FUNCTION
9	VI5	Vectored Interrupt Line #5	
10	VI6	Vectored Interrupt Line #6	
11	VI7	Vectored Interrupt Line #7	
12 to 17	TO BE DIFINED		
18	$\overline{\text{STA DSB}}$	$\overline{\text{STATUS DISABLE}}$	Allows the buffers for the 8 status lines to be tri-stated
19	$\overline{\text{C/C DSB}}$	$\overline{\text{COMMAND/CONTROL DISABLE}}$	Allows the buffers for the 6 output command/control lines to be tri-stated
20	UNPROT	UNPROTECT	Input to the memory protect flip-flop on a given memory board
21	SS	SINGLE STEP	Indicates that the machine is in the process of performing a single step
22	$\overline{\text{ADD DSB}}$	$\overline{\text{ADDRESS DISABLE}}$	Allows the buffers for the 16 address lines to be tri-stated
23	$\overline{\text{DO DSB}}$	$\overline{\text{DATA OUT DISABLE}}$	Allows the buffers for the 8 data output lines to be tri-stated
24	$\phi 2$	Phase 2 Clock	
25	$\phi 1$	Phase 1 Clock	
26	PHLDA	Hold Acknowledge	Processor command/control output signal which appears in response to the HOLD signal; indicates that the data and address bus will go to the high impedance state

BUS DEFINITION

No.	SYMBOL	NAME	FUNCTION
27	PWAIT	WAIT	Processor command/control output signal which acknowledges that the processor is in a WAIT state
28	PINTE	INTERRUFT ENABLE	Processor command/control output signal indicating interrupts are enabled: indicates the content of the CPU internal interrupt flip-flop; F-F may be set or reset by EI and DI instruction and inhibits interrupts from being accepted by the CPU if it is reset
29	A5	Address Line #5	
30	A4	Address Line #4	
31	A3	Address Line #3	
32	A15	Address Line #15	
33	A12	Address Line #12	
34	A9	Address Line #9	
35	DO1	Data Out Line #1	
36	DO0	Data Out Line #0	
37	A10	Address Line #10	
38	DO4	Data Out Line #4	
39	DO5	Data Out Line #5	
40	DO6	Data Out Line #6	
41	DI2	Data In Line #2	
42	DI3	Data In Line #3	
43	DI7	Data In Line #7	

BUS DEFINITION

No.	SYMBOL	NAME	FUNCTION
44	SM1	M1	Status output signal that indicates that the processor is in the fetch cycle for the first byte of an instruction
45	SOUT	OUT	Status output signal which indicates that the address bus contains the address of an output device and the data bus will contain the output data when $\overline{\text{PWR}}$ is active
46	SINP	INP	Status output signal which indicates that the address bus contains the address of an input device and the input data should be placed on the data bus when PDBIN is active
47	SMEMR	MEMR	Status output signal which indicates that the data bus will be used for memory read data
48	SHLTA	HLTA	Status output signal which acknowledges a HALT instruction
49	$\overline{\text{CLOCK}}$	$\overline{\text{CLOCK}}$	Inverted output of the 2MHz oscillator that generates the 2 phase clock
50	GND	GROUND	
51	+8V	+8 volts	Unregulated input to 5v regulators
52	-16V	-16 volts	Negative unregulated voltage
53	$\overline{\text{SSW DSB}}$	$\overline{\text{SENSE SWITCH DISABLE}}$	Disables the data input buffers so the input from the sense switches may be strobed onto the bidirectional data bus right at the processor
54	$\overline{\text{EXT CLR}}$	$\overline{\text{EXTERNAL CLEAR}}$	Clear signal for I/O devices (front panel switch closure to ground)

BUS DEFINITION

No.	SYMBOL	NAME	FUNCTION
55 to 67	TO BE DEFINED		
68	MWRT	MEMORY WRITE	Indicates that the current data on the Data Out Bus is to be written into the memory location currently on the address bus
69	$\overline{PS}$	$\overline{\text{PROTECT STATUS}}$	Indicates the status of the memory protect flip-flop on the memory board currently addressed
70	PROT	PROTECT	Input to the memory protect flip-flop on the memory board currently addressed
71	RUN	RUN	Indicates that the RUN/STOP flip-flop is Reset
72	PRDY	READY	Processor command/control input that controls the run state of the processor; if the line is pulled low the processor will enter a wait state until the line is released
73	$\overline{PINT}$	$\overline{\text{INTERRUPT REQUEST}}$	The processor recognizes an interrupt request on this line at the end of the current instruction or while halted. If the processor is in the HOLD state or the Interrupt Enable flip-flop is reset, it will not honor the request.
74	$\overline{PHOLD}$	$\overline{\text{HOLD}}$	Processor command/control input signal which requests the processor to enter the HOLD state; allows an external device to gain control of address and data buses as soon as the processor has completed its use of these buses for the current machine cycle

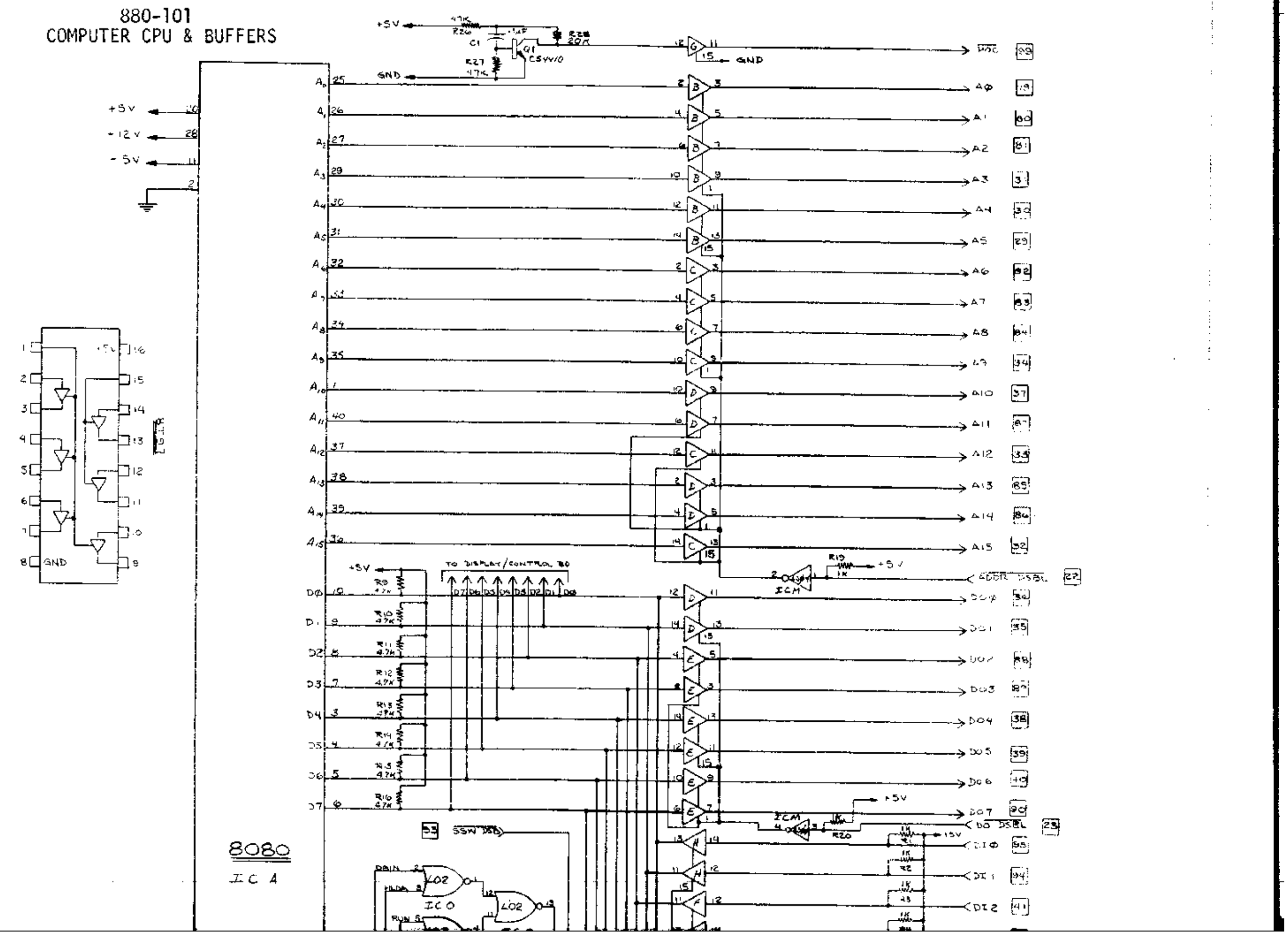
BUS DEFINITION

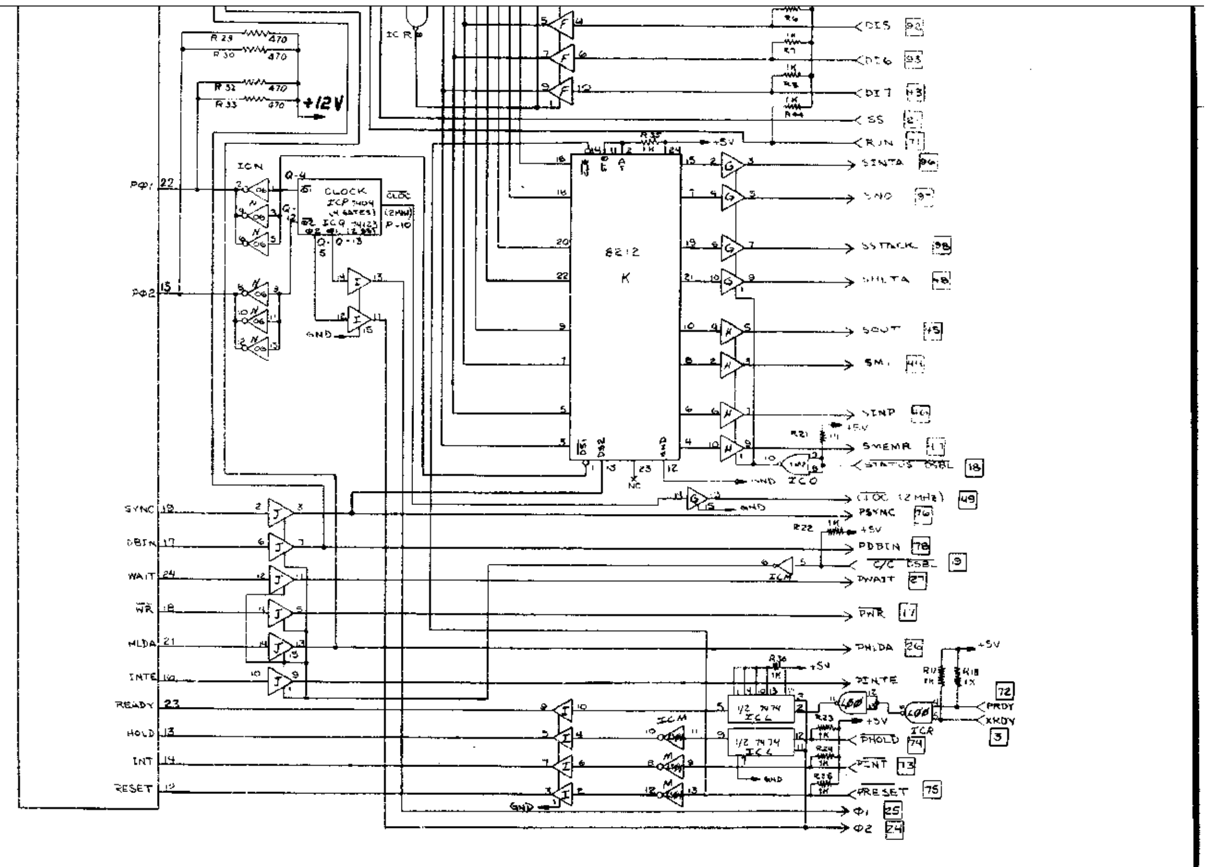
No.	SYMBOL	NAME	FUNCTION
75	$\overline{\text{PRESET}}$	$\overline{\text{RESET}}$	Processor command/control input; while activated the content of the program counter is cleared and the instruction register is set to 0
76	PSYNC	SYNC	Processor command/control output provides a signal to indicate the beginning of each machine cycle
77	$\overline{\text{PWR}}$	$\overline{\text{WRITE}}$	Processor command/control output used for memory write or I/O output control: data on the data bus is stable while the $\overline{\text{PWR}}$ is active
78	PDBIN	DATA BUS IN	Processor command/control output signal indicates to external circuits that the data bus is in the input mode
79	A0	Address Line #0	
80	A1	Address Line #1	
81	A2	Address Line #2	
82	A6	Address Line #6	
83	A7	Address Line #7	
84	A8	Address Line #8	
85	A13	Address Line #13	
86	A14	Address Line #14	
87	A11	Address Line #11	
88	DO2	Data Out Line #2	
89	DO3	Data Out Line #3	
90	DO7	Data Out Line #7	
91	DI4	Data In Line #4	

BUS DEFINITION

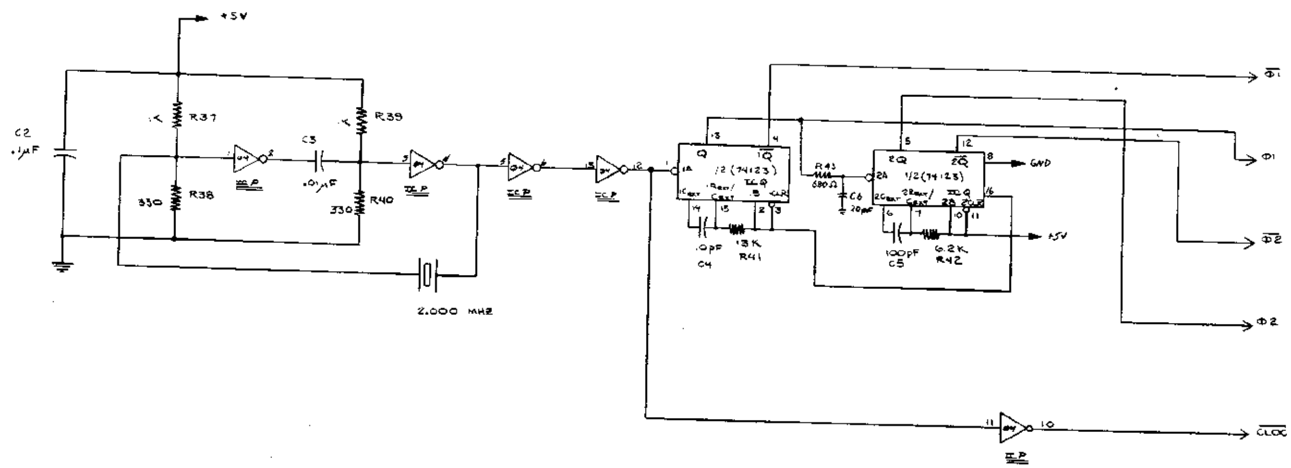
No.	SYMBOL	NAME	FUNCTION
92	DI5	Data In Line #5	
93	DI6	Data In Line #6	
94	DI1	Data In Line #1	
95	DI0	Data In Line #0	
96	SINTA	INTA	Status output signal to acknowledge signal for INTERRUPT request
97	SWO	WO	Status output signal indicates that the operation in the current machine cycle will be a WRITE memory or output function
98	SSTACK	STACK	Status output signal indicates that the address bus holds the pushdown stack address from the Stack Pointer
99	$\overline{POC}$	Power-On Clear	
100	GND	GROUND	

880-101
COMPUTER CPU & BUFFERS

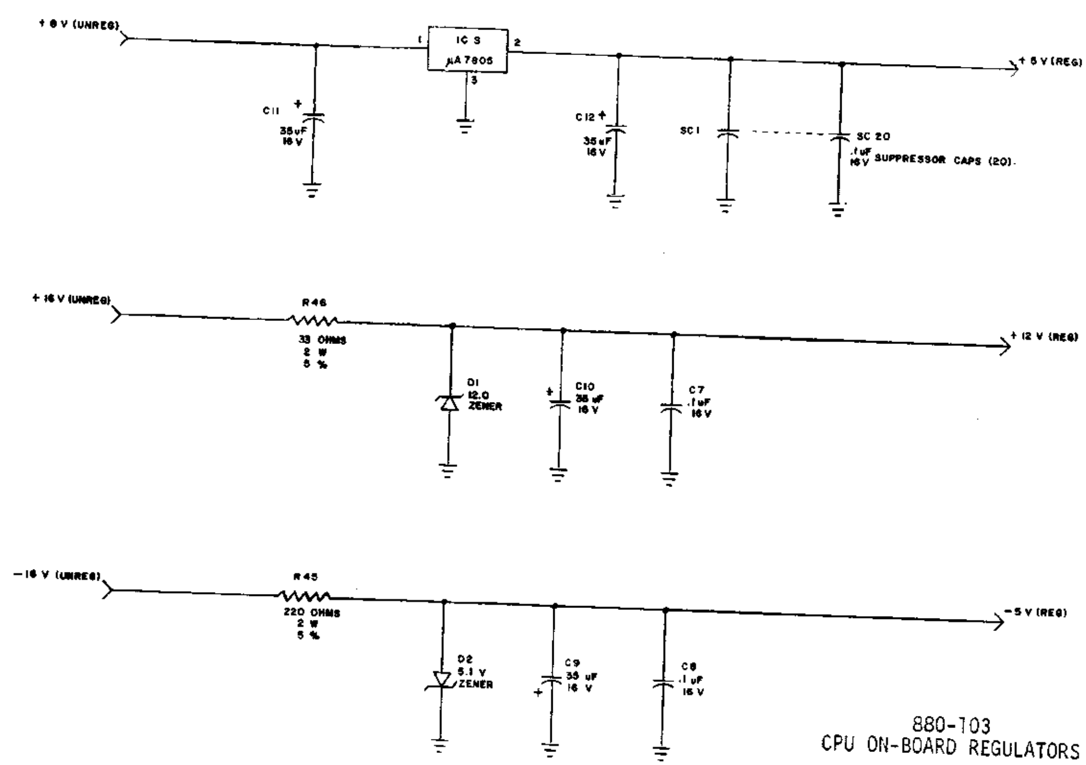




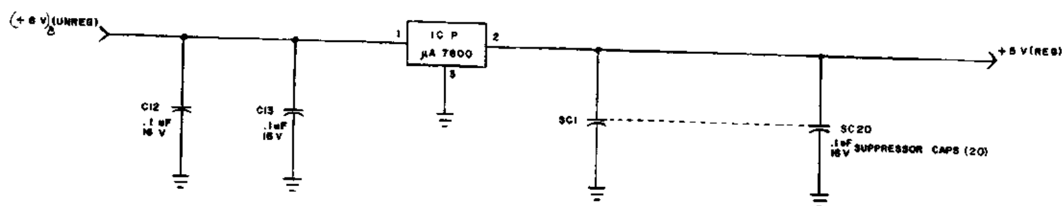
8212
K
CLOCK
+12V
SYNC
DBIN
WAIT
WR
HLDA
INTE
READY
HOLD
INT
RESET
PΦ1
PΦ2
SINTA
SWO
SSTACK
SHLTA
SOUT
SM1
SINP
SMEMR
STATUS DSBL
PSYNC
PDBIN
C/C DSBL
PWAIT
PWR
PHLDA
PINTE
PRDY
XRDY
PHOLD
PINT
PRESET
Φ1
Φ2
DI5
DI6
DI7
SS
RUN
GND
+5V



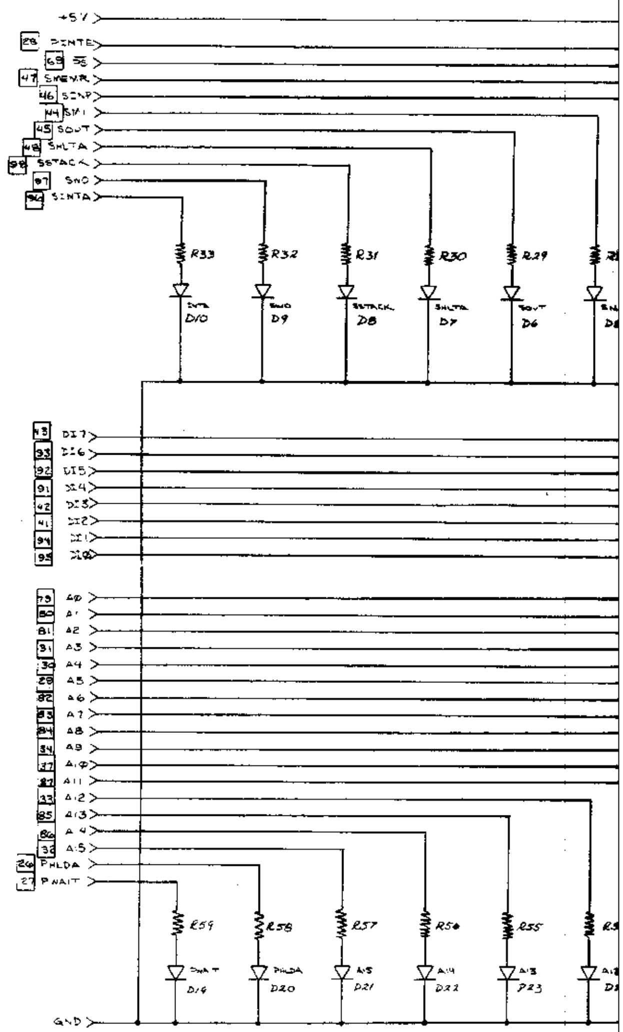
880-102
SYSTEM CLOCK



880-103
CPU ON-BOARD REGULATORS

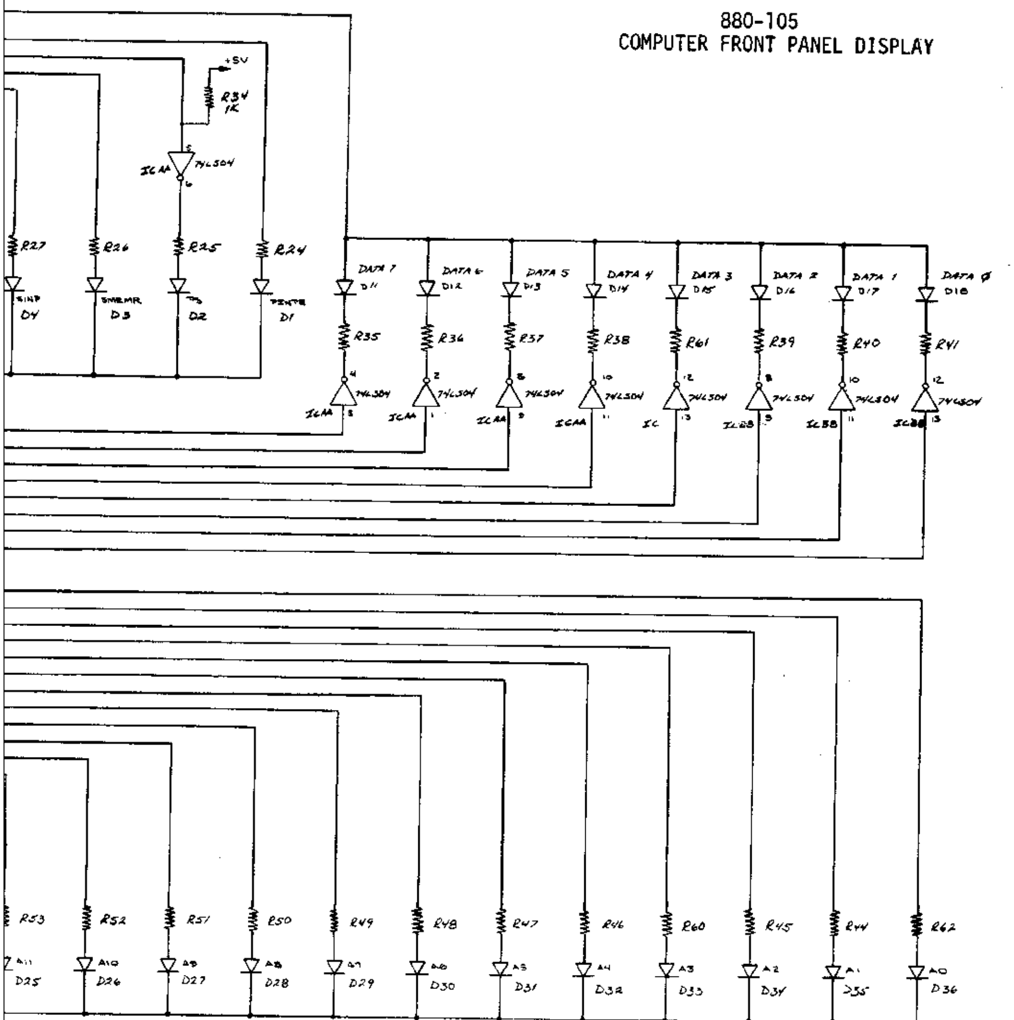


880-104
DISPLAY/CONTROL ON-BOARD REGULATOR

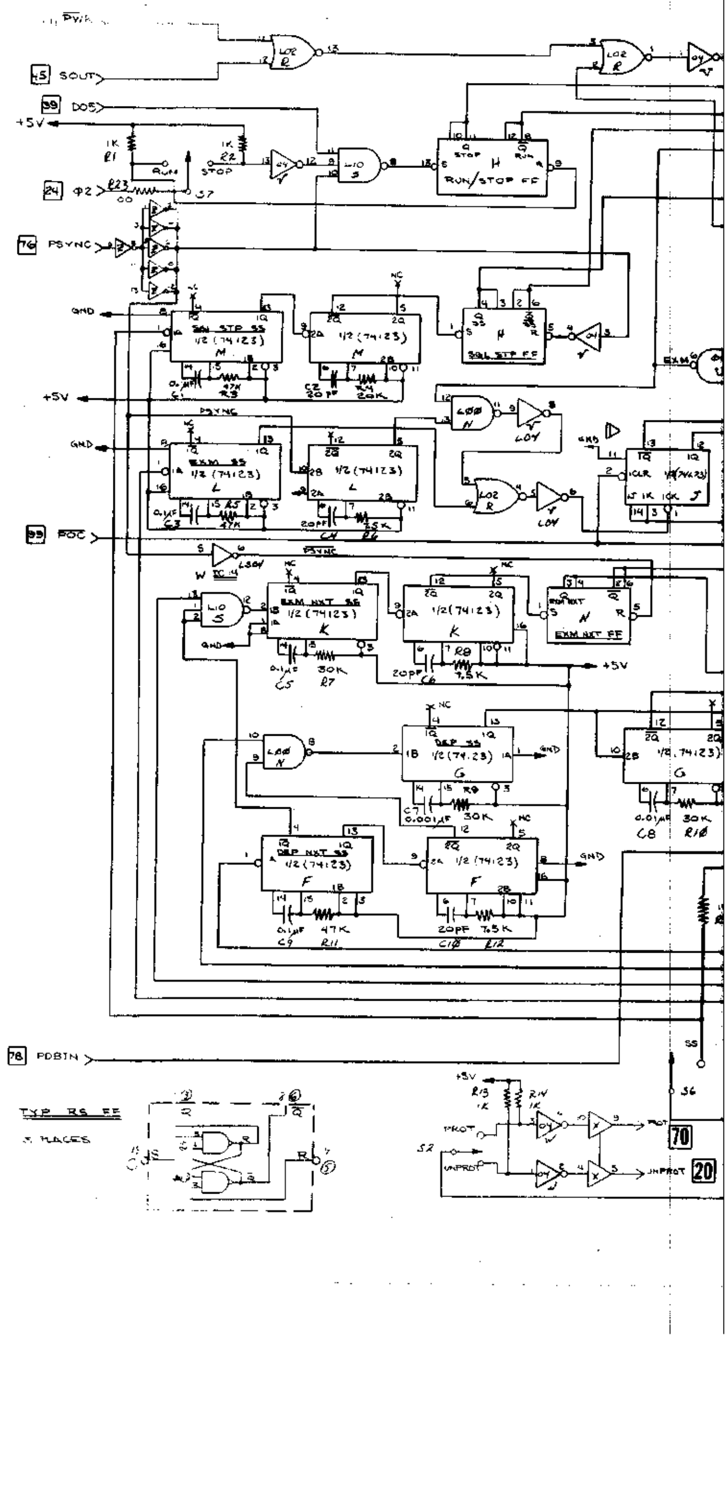


+5V
PINTE
SMEMR
SINP
SOUT
SHLTA
SSTACK
SINTA
R33
R32
R31
R30
R29
D10
D9
D8
D7
D6
DI7
DI6
DI5
DI4
DI3
DI2
DI1
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A11
A13
PHLDA
PWAIT
R59
R58
R57
R55
D20
D21
GND

880-105
COMPUTER FRONT PANEL DISPLAY

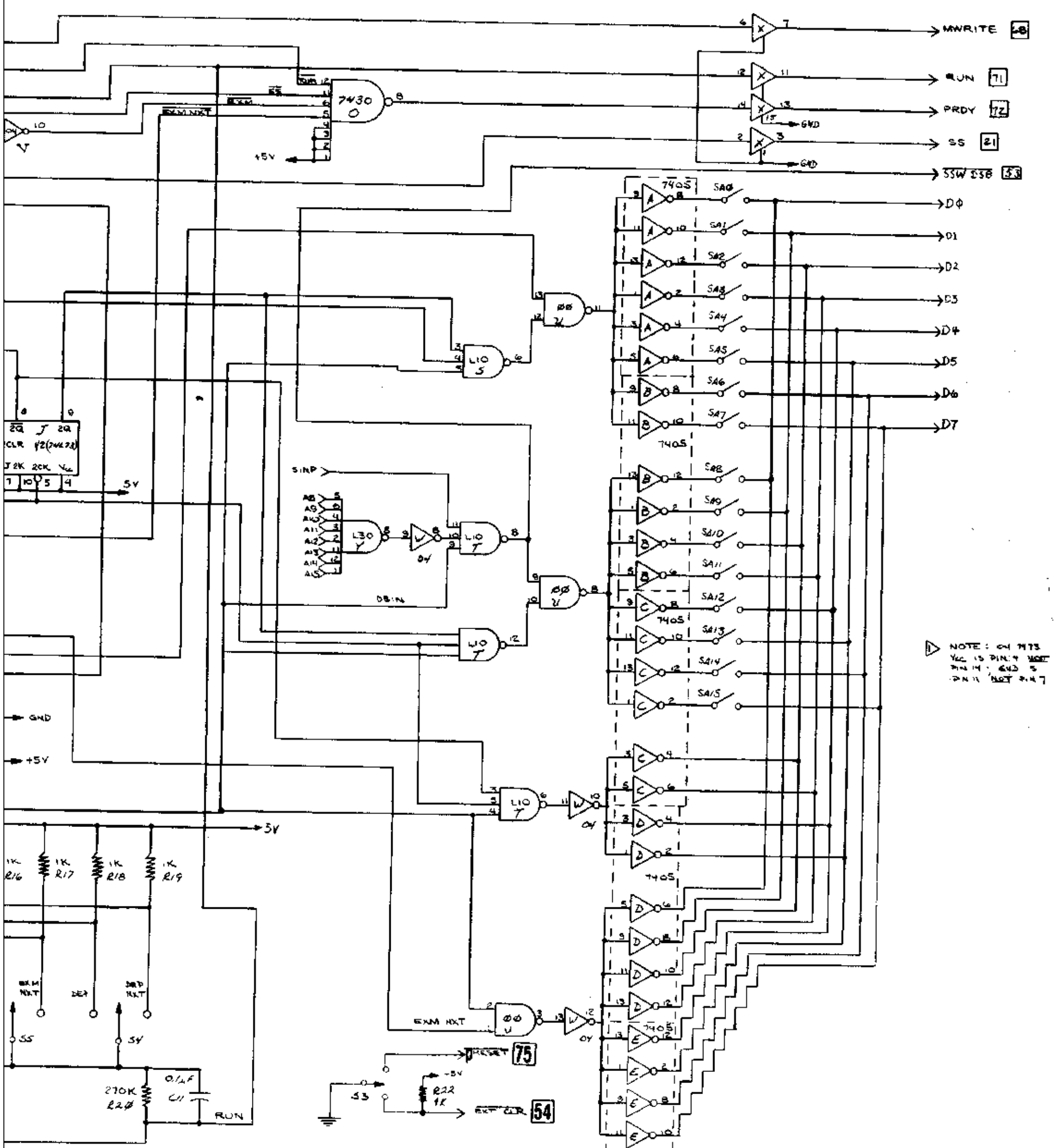


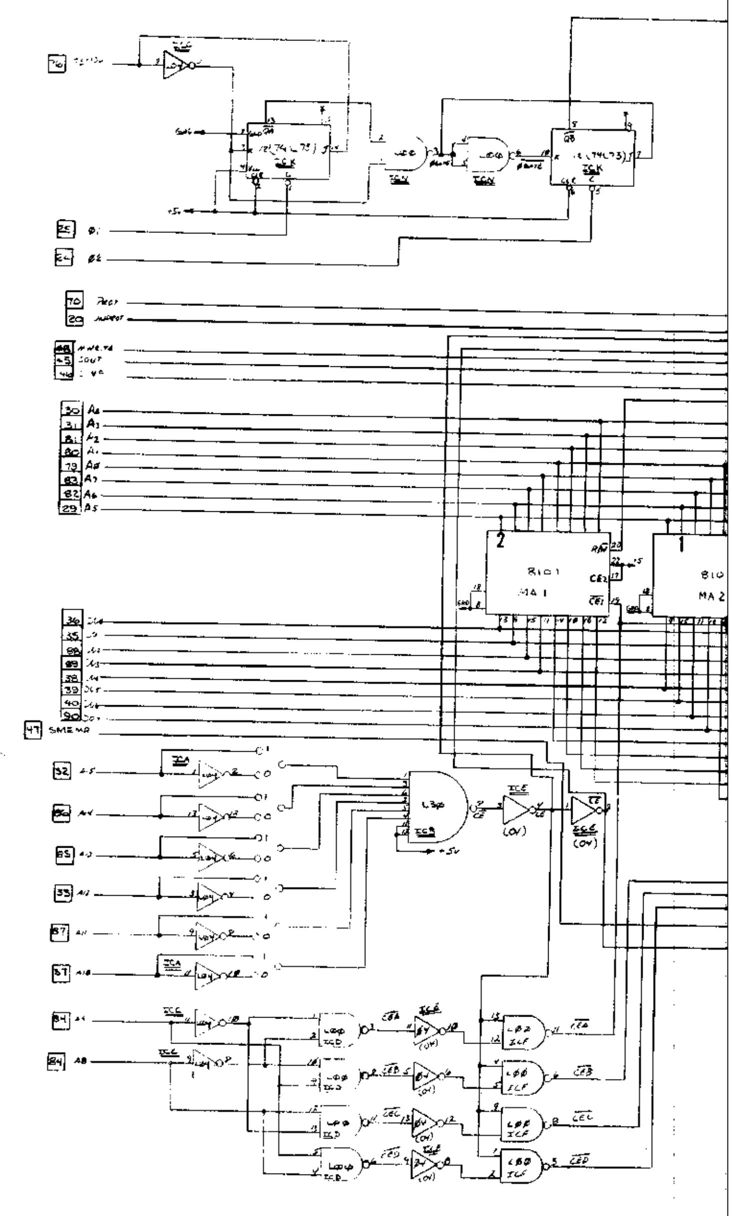
COMPUTER



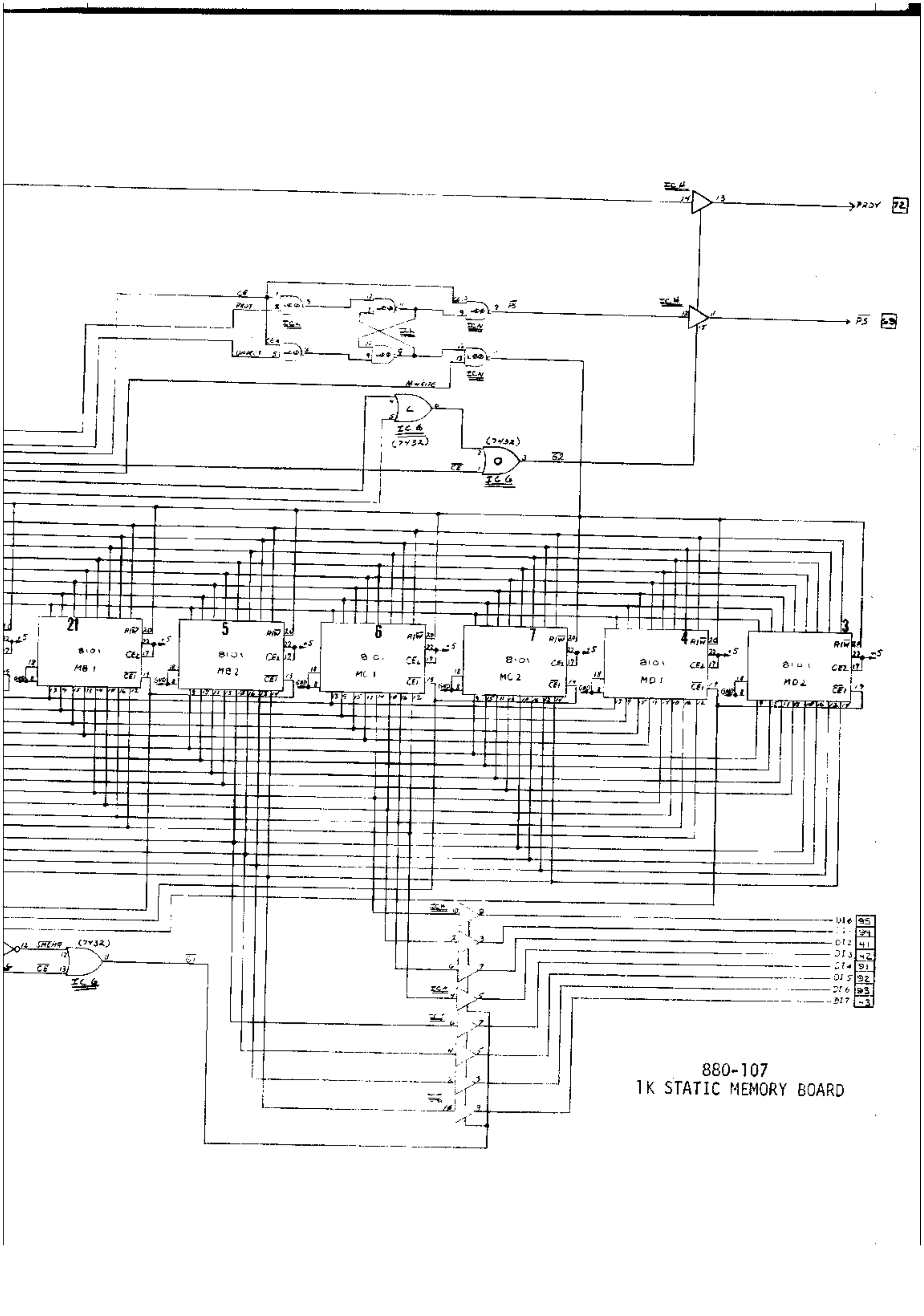
45 SOUT
39 DOS
+5V
STOP
RUN
24 Φ2
S7
76 PSYNC
RUN/STOP FF
SGL STP FF
SGL STP SS
1/2 (74123)
EXM SS
EXM NXT SS
EXM NXT FF
DEP SS
DEP NXT SS
GND
PSYNC
99 POC
78 PDBIN
TYP RS FF
3 PLACES
PROT
UNPROT
S2
S6
70
20

)-106
NT PANEL CONTROL

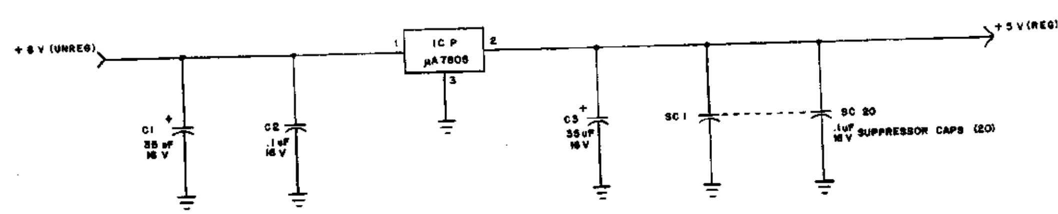




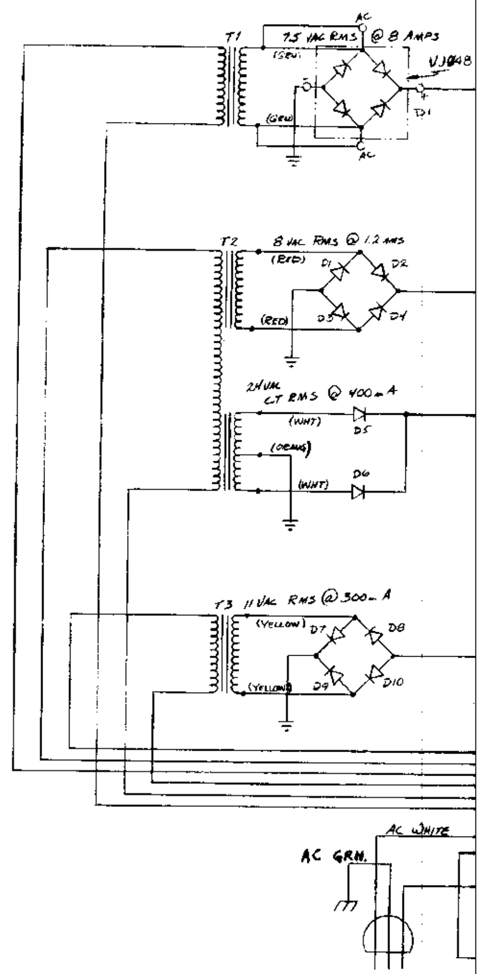
8101
MA 1
MA 2
SMEMR



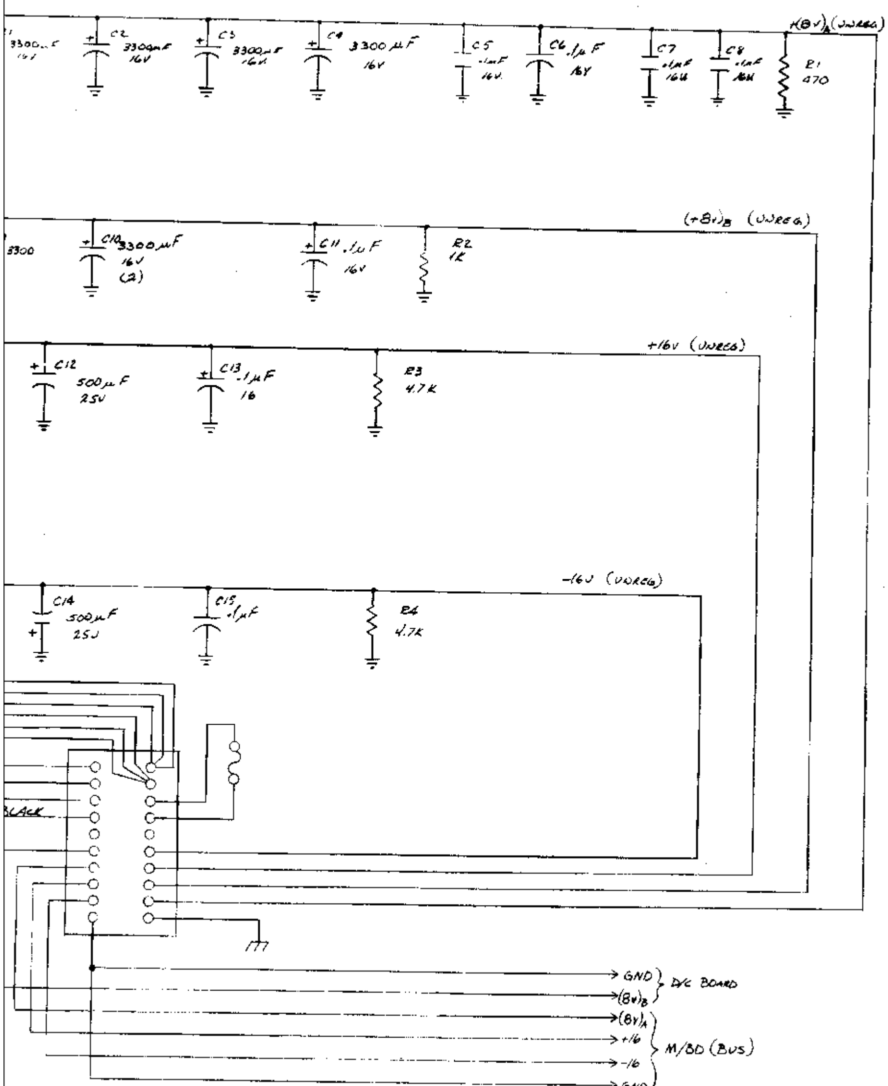
880-107
1K STATIC MEMORY BOARD



880-108
1K STATIC MEMORY ON-BOARD REGULATOR



880-109
POWER SUPPLY



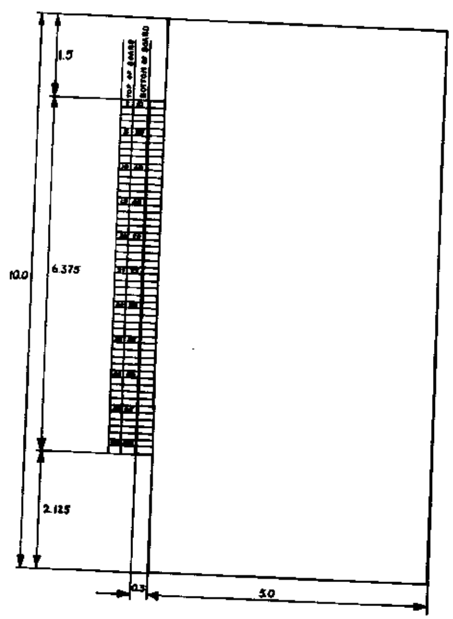
C2 3300µF 16V
C3 3300µF 16V
C4 3300µF 16V
C5 .1µF 16V
C6 .1µF 16V
C7 .1µF 16V
C8 .1µF 16V
R1 470
(8V)A (UNREG)
C10 3300µF 16V (2)
C11 .1µF 16V
R2 1K
(+8V)B (UNREG)
C12 500µF 25V
C13 .1µF 16
R3 4.7K
+16V (UNREG)
C14 500µF 25V
C15 .1µF
R4 4.7K
-16V (UNREG)
BLACK
GND
(8V)B
D/C BOARD
(8V)A
+16
-16
GND
M/BD (BUS)

TOP OF BOARD

1	+5 V
2	+15 V
3	XRDY
4	VI 0
5	VI 1
6	VI 2
7	VI 3
8	VI 4
9	VI 5
10	VI 6
11	VI 7
12	
13	
14	
15	
16	
17	
18	STAT DSBLE
19	C/C DSBLE
20	UNPROTECT
21	SS
22	ADD DSBL
23	DO DSBL
24	Φ2
25	Φ1
26	PHLDA
27	PWAIT
28	PINTE
29	A5
30	A4
31	A3
32	A15
33	A12
34	A9
35	DO1
36	DO0
37	A10
38	DO4
39	DO5
40	DO6
41	DI2
42	DI3
43	DI7
44	SM1
45	SOUT
46	SINP
47	SMEMR
48	SHLTA
49	CLOCK (2 MHz)
50	GND

BOTTOM OF BOARD

51	+5 V	A
52	-15 V	B
53	SSW DSB	C
54	EXT CLR	D
55		E
56		F
57		H
58		J
59		K
60		L
61		M
62		N
63		P
64		R
65		S
66		T
67		U
68	MWRITE	V
69	PS	W
70	PROTECT	X
71	RUN	Y
72	PRDY	Z
73	PINT	a
74	PHOLD	b
75	PRESET	c
76	PSYNC	d
77	PWR	e
78	PDBIN	f
79	A 0	h
80	A 1	j
81	A 2	k
82	A 6	l
83	A 7	m
84	A 8	n
85	A 13	p
86	A 14	r
87	A 11	s
88	DO 2	t
89	DO 3	u
90	DO 7	v
91	DI 4	w
92	DI 5	x
93	DI 6	y
94	DI 1	z
95	DI 0	AA
96	SINTA	AB
97	SWO	AC
98	SSTACK	AD
99	POC	AE
100	GND	AF



880-110
SYSTEM BUS STRUCTURE

OCTAL PROGRAM SET 8800

RETURNS	
RET	311
RNZ	300
RZ	310
RNC	320
RC	330
RPO	340
RPE	350
RP	360
RM	370

STACK	
POP B	301
POP D	321
POP H	341
POP SW	361
PUSH B	305
PUSH D	325
PUSH H	345
PUSH SW	365

INPUT	
IN	333
B2	
OUTPUT	
OUT	323
B2	
XCHG	353
XTHL	343
SPHL	371
PCHL	353

MACHINE	
HLT	166
NOP	000
DI	363
EI	373
SHIFTS	
RLC	007
RRC	017
RAL	027
RAR	037

ADD PAIRS	
DAD B	011
DAD D	031
DAD H	051
DAD SP	071
SET PAIRS	
STAX B	002
STAX D	022
LDAX B	012
LDAX D	032

JUMPS	
JMP	303
B2	< >
B3	< >
JNZ	302
B2	< >
B3	< >
JZ	312
B2	< >
B3	< >
JNC	322
B2	< >
B3	< >
JC	332
B2	< >
B3	< >
JPO	342
B2	< >
B3	< >
JPE	352
B2	< >
B3	< >
JP	362
B2	< >
B3	< >
JM	372
B2	< >
B3	< >

CALLS	
CALL	315
B2	< >
B3	< >
CNZ	304
B2	< >
B3	< >
CZ	314
B2	< >
B3	< >
CNC	324
B2	< >
B3	< >
CC	334
B2	< >
B3	< >
CPO	344
B2	< >
B3	< >
CPE	354
B2	< >
B3	< >
CP	364
B2	< >
B3	< >
CM	374
B2	< >
B3	< >

REGISTERS	
ADDR r	20S
ADC r	21S
SUB r	22S
SBB r	23S
ANA r	24S
XRA r	25S
ORA r	26S
CMP r	27S
INCREMENT ±	
INR r	0S4
DECREMENT	
DCR r	0S5
MOVES ↔	
MOV r1r2	1DS
MVI r	0DS
VALUES FOR S & D	
B = 0	
C = 1	
D = 2	
E = 3	
H = 4	
L = 5	
M = 6	
A = 7	

ACCUMULATOR	
ADI	306
B2	< >
ACI	316
B2	< >
SUI	326
B2	< >
SBI	336
B2	< >
ANI	346
B2	< >
XRI	356
B2	< >
ORI	366
B2	< >
CPI	376
B2	< >
DIRECT	
STA	062
B2	< >
B3	< >
LDA	072
B2	< >
B3	< >
DAA	047
CMA	057
STC	067
CMC	077
RST	3A7

LOAD IMMED PAIR	
LXI B	001
B2	< >
B3	< >
LXI D	021
B2	< >
B3	< >
LXI H	041
B2	< >
B3	< >
LXI SP	061
B2	< >
B3	< >
INCR PAIR	
INX B	003
INX D	023
INX H	043
INX SP	063
DECR PAIR	
DCX B	013
DCX D	033
DCX H	053
DCX SP	073
SHLD	042
B2	< >
B3	< >
LHLD	052
B2	< >
B3	< >

ALTAIR 8800 OPERATOR'S MANUAL

TABLE OF CONTENTS

MITS INC.
"Creative Electronics"

PRINTED IN U.S.A.

6328 LINN, N.E., P.O. BOX 8636, ALBUQUERQUE, N.M. 87108 U.S.A.
505/265-7553

PART 1 INTRODUCTION

Remarkable advances in semiconductor technology have made possible the development of the *ALTAIR 8800*, the most economical computer ever and the first available in both kit and assembled form. The heart of the *ALTAIR 8800* is Intel Corporation's Model 8080 Microcomputer, a complete Central Processing Unit on a single silicon chip. Fabricated with N-channel large scale integrated circuit (LSI) metal-oxide-semiconductor (MOS) technology, Intel's 8080 Microcomputer on a chip represents a major technological breakthrough.

This operating manual has been prepared to acquaint both the novice and the experienced computer user in the operation of the *ALTAIR 8800*. The computer has 78 machine language instructions and is capable of performing several important operations not normally available with conventional minicomputers. After reading this manual, even a novice will be able to load a program into the *ALTAIR 8800*.

Users of the *ALTAIR 8800* include persons with a strong electronics background and little or no computer experience and persons with considerable programming experience and little or no electronics background. Accordingly, this manual has been prepared with all types of users in mind. Part 1 of the manual prepares the user for better understanding computer terminology, technology, and operation with an introduction to conventional and electronic logic, a description of several important number systems, a discussion of basic programming, and a discourse on computer languages.

Parts 2 and 3 in the manual describe the organization and operation of the *ALTAIR 8800*. Emphasis is placed on those portions of the computer most frequently utilized by the user. Finally, Part 4 of the manual presents a detailed listing of the *ALTAIR 8800's* 78 instructions. An Appendix condenses the instructions into a quick reference listing.

Even if you have little or no experience in computer operation and organization, a careful reading of this manual will prepare you for operating the *ALTAIR 8800*. As you gain experience with the machine, you will soon come to understand its truly incredible versatility and data processing capability. Don't be discouraged if the manual seems too complicated in places. Just remember that a computer does only what its programmer instructs it to do.

A. LOGIC

George Boole, a ninteenth century British mathematician, made a detailed study of the relationship between certain fundamental logical expressions and their arithmetic counterparts. Boole did not equate mathematics with logic, but he did show how any logical statement can be analyzed with simple arithmetic relationships. In 1847, Boole published a booklet entitled Mathematical Analysis of Logic and in 1854 he published a much more detailed work on the subject. To this day, all practical digital computers and many other electronic circuits are based upon the logic concepts explained by Boole.

Boole's system of logic, which is frequently called Boolean algebra, assumes that a logic condition or statement is either true or false. It cannot be both true and false, and it cannot be partially true or partially false. Fortunately, electronic circuits are admirably suited for this type of dual-state operation. If a circuit in the ON state is said to be true and a circuit in the OFF state is said to be false, an electronic analogy of a logical statement can be readily synthesized.

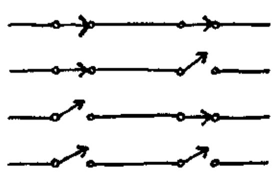
With this in mind, it is possible to devise electronic equivalents for the three basic logic statements: AND, OR and NOT. The AND statement is true if and only if either or all of its logic conditions are true. A NOT statement merely reverses the meaning of a logic statement so that a true statement is false and a false statement is true.

It's easy to generate a simple equivalent of these three logic statements by using on-off switches. A switch which is ON is said to be true while a switch which is OFF is said to be false. Since a switch which is OFF will not pass an electrical current, it can be assigned a numerical value of 0. Similarly, a switch which is ON does pass an electrical current and can be assigned a numerical value of 1.

We can now devise an electronic equivalent of the logical AND statement by examining the various permutations for a two condition AND statement:

CONDITIONS (Inputs)	CONCLUSION (Output)
1. True AND True	True
2. True AND False	False
3. False AND True	False
4. False AND False	False

The electronic ON-OFF switch equivalent of these permutations is simply:



CONDITIONS (ON-OFF)	CONCLUSION (OUTPUT)
1.	1
2.	0
3.	0
4.	0

Similarly, the numerical equivalents of these permutations is:

CONDITIONS (Inputs)	CONCLUSION (Output)
1. 1 AND 1	1
2. 1 AND 0	0
3. 0 AND 1	0
4. 0 AND 0	0

Digital design engineers refer to these table of permutations as <u>truth tables</u>. The truth table for the AND statement with two conditions is usually presented thusly:

A	B	OUT
1	1	1
0	1	0
1	0	0
0	0	0

FIGURE 1-1. AND Function Truth Table

It is now possible to derive the truth tables for the OR and NOT statements, and each is shown in Figures 1-2 and 1-3 respectively.

A	B	OUT
1	1	1
0	1	1
1	0	1
0	0	0

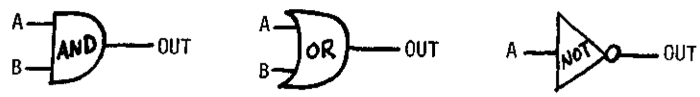
FIGURE 1-2. OR Function Truth Table

A	OUT
1	0
0	1

FIGURE 1-3. NOT Function Truth Table

B. ELECTRONIC LOGIC

All three of the basic logic functions can be implemented by relatively simple transistor circuits. By convention, each circuit has been assigned a symbol to assist in designing logic systems. The three symbols along with their respective truth tables are shown in Figure 1-4.



A	B	OUT
0	0	0
0	1	0
1	0	0
1	1	1

A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	1

A	OUT
0	1
1	0

FIGURE 1-4. The Three Main Logic Symbols

The three basic logic circuits can be combined with one another to produce still more logic statement analogies. Two of these circuit combinations are used so frequently that they are considered basic logic circuits and have been assigned their own logic symbols and truth tables. These circuits are the NAND (NOT-AND) and the NOR (NOT-OR). Figure 1-5 shows the logic symbols and truth tables for these circuits.

A	B	OUT
0	0	1
0	1	1
1	0	1
1	1	0

A	B	OUT
0	0	1
0	1	0
1	0	0
1	1	0

FIGURE 1-5. The NAND and NOR Circuits

Three or more logic circuits make a logic system. One of the most basic logic systems is the EXCLUSIVE-OR circuit shown in Figure 1-6.

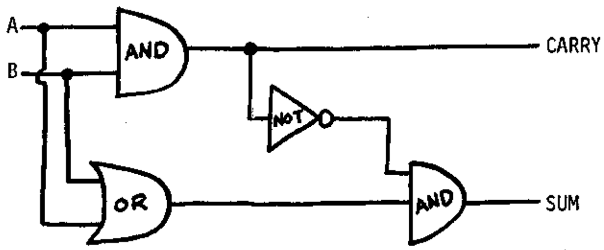


FIGURE 1-6. The EXCLUSIVE-OR Circuit

The EXCLUSIVE-OR circuit can be used to implement logical functions, but it can also be used to add two input conditions. Since electronic logic circuits utilize only two numerical units, 0 and 1, they are compatible with the binary number system, a number system which has only two digits. For this reason, the EXCLUSIVE-OR circuit is often called a binary adder.

Various combinations of logic circuits can be used to implement numerous electronic functions. For example, two NAND circuits can be connected to form a bistable circuit called a flip-flop. Since the flip-flop changes state only when an incoming signal in the form of a pulse arrives, it acts as a short term memory element. Several flip-flops can be cascaded together to form electronic counters and memory registers.

Other logic circuits can be connected together to form monostable and astable circuits. Monostable circuits occupy one of two states unless an incoming pulse is received. They then occupy an opposite state for a brief time and then resume their normal state. Astable circuits continually switch back and forth between two states.

C. NUMBER SYSTEMS

Probably because he found it convenient to count with his fingers, early man devised a number system which consisted of ten digits. Number systems, however, can be based on any number of digits. As we have already seen, dual-state electronic circuits are highly compatible with a two digit number system, and its digits are termed bits (binary digits). Systems based upon eight and sixteen are also compatible with complex electronic logic systems such as computers since they provide a convenient shorthand method for expressing lengthy binary numbers.

D. THE BINARY SYSTEM

Like virtually all digital computers, the *ALTAIR 8800* performs nearly all operations in binary. A typical binary number processed by the computer incorporates 8-bits and may appear as: 10111010. A fixed length binary number such as this is usually called a word or byte, and computers are usually designed to process and store a fixed number of words (or bytes).

A binary word like 10111010 appears totally meaningless to the novice. But since binary utilizes only two digits (bits), it is actually much simpler than the familiar and traditional decimal system. To see why, let's derive the binary equivalents for the decimal numbers from 0 to 20. We will do this by simply adding 1 to each successive number until all the numbers have been derived. Counting in any number system is governed by one basic rule: Record successive digits for each count in a column. When the total number of available digits has been used, begin a new column to the left of the first and resume counting.

Counting from 0 to 20 in binary is very easy since there are only two digits (bits). The binary equivalent of the decimal 0 is 0. Similarly, the binary equivalent of the decimal 1 is 1. Since both available bits have now been used, the binary count must incorporate a new column to form the binary equivalent for the decimal 2. The result is 10. (Incidentally, ignore any resemblance between binary and decimal numbers. Binary 10 is not decimal 10!) The binary equivalent of the decimal number 3 is 11. Both bits have been used again, so a third column must be started to obtain the binary equivalent for the decimal number 4 (100). You should now be able to continue counting and derive all the remaining binary equivalents for the decimal numbers 0 to 20:

DECIMAL	BINARY
0	0
1	1
2	10
3	11

DECIMAL	BINARY
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000
17	10001
18	10010
19	10011
20	10100

A simple procedure can be used to convert a binary number into its decimal equivalent. Each bit in a binary number indicates by which power of two the number is to be raised. The sum of the powers of two gives the decimal equivalent for the number. For example, consider the binary number 10011:

$$10011 = [(1 \times 2^4) + (0 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)]$$

$$= [(16) + (0) + (0) + (2) + (1)]$$

$$= 19$$

E. THE OCTAL SYSTEM

Since the binary system has only two bits, it doesn't take long to accumulate a long string of 0s and 1s. For example, a six-digit decimal number requires 19 bits.

Lengthy binary numbers can be simplified by dividing them into groups of three bits and assigning a decimal equivalent to each 3-bit group. Since the highest 3-bit binary number corresponds to the decimal 7, eight combinations of 0s and 1s are possible (0-7).

The basic *ALTAIR 8800* accepts a binary input, and any binary number loaded into the machine can be simplified into octal format. Of course the octal numbers must be changed back to binary for entry into the computer, but since only eight bit patterns are involved the procedure is both simple and fast. A typical binary instruction for the *ALTAIR 8800* is: 11101010. This instruction can be converted to octal by first dividing the number into groups of three bits beginning with the least significant bit: 11 101 010. Next, assign the decimal equivalent to each of the three bit patterns:

11	101	010
3	5	2

Therefore, 11 101 010 in binary corresponds to 352 in octal. To permit rapid binary to octal conversion throughout the remainder of this manual, most binary numbers will be presented as groups of three bits.

F. COMPUTER PROGRAMMING

As will become apparent in Part 2, the Central Processing Unit (CPU) of a computer is essentially a network of logic circuits and systems whose interconnections or organization can be changed by the user. The computer can therefore be thought of as a piece of variable hardware. Implementation of variations in a computer's hardware is achieved with a set of programmed instructions called software.

The software instructions for the *ALTAIR 8800* must be loaded into the machine in the form of sequential 8-bit words called machine language. This and other more advanced computer languages will be discussed later.

The basics of computer programming are quite simple. In fact, often the most difficult part of programming is defining the problem you wish to solve with the computer. Below are listed the three main steps in generating a program:

1. Defining the Problem
2. Establishing an Approach
3. Writing the Program

Once the problem has been defined, an approach to its solution can be developed. This step is simplified by making a diagram which shows the orderly, step-by-step solution of the problem. Such a diagram is called a flow diagram. After a flow diagram has been made, the various steps can be translated into the computer's language. This is the easiest of the three steps since all you need is a general understanding of the instructions and a list showing each instruction and its machine language equivalent.

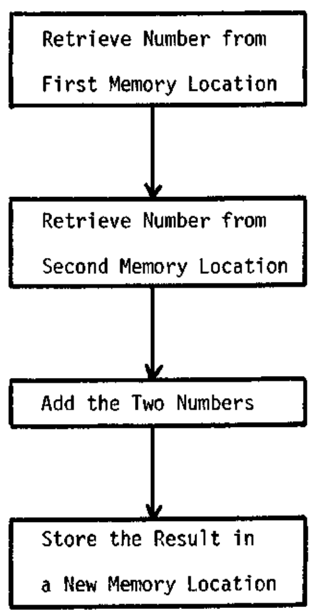
The *ALTAIR 8800* has an extensive programming capability. For example, a program can cause data to be transferred between the computer's memory and the CPU. The program can even cause the computer to make logical decisions. For example, if a specified condition is met, the computer can jump from one place in the program to any other place and continue program execution at the new place. Frequently used special purpose programs can be stored in the computer's memory for later retrieval and use by the main program. Such a special purpose program is called a subroutine. The *ALTAIR 8800* instructions are described in detail in Part 4 of this manual.

G. A SIMPLE PROGRAM

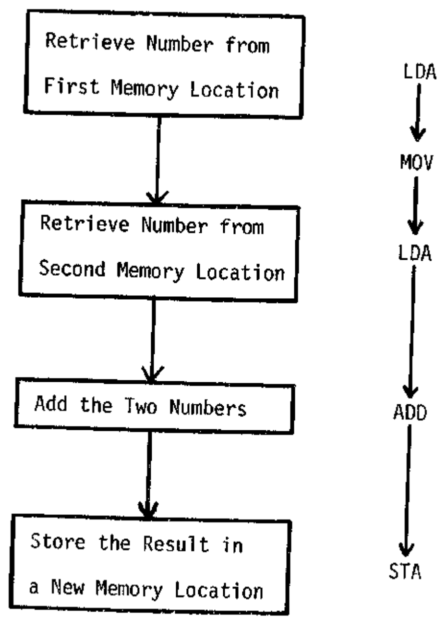
Assume you wish to use the *ALTAIR 8800* to add two numbers located at two different memory locations and store the result elsewhere in the memory. Of course this is a very simple problem, but it can be used to illustrate several basic programming techniques. Here are the steps used in generating a program to solve this problem:

1. Define the Problem--Add two numbers located in memory and store the result elsewhere in memory.

2. Establish an Approach--A flow diagram can now be generated:



3. Write the Program--Translating the flow diagram into a language or format suitable for use by the computer may seem complicated at first. However, a general knowledge of the computer's organization and operation makes the job simple. In this case, the four part flow diagram translates into five separate instructions:



These instructions may seem meaningless now, but their meaning and application will become much clearer as you proceed through this manual. For example, the need for the extra instruction (MOV) will become more obvious after you learn that the computer must temporarily store the first number retrieved from memory in a special CPU memory called a register. The first number is stored in the register until it can be added to the second number.

H. COMPUTER LANGUAGES

The software for any computer must be entered into the machine in the form of binary words called machine language. Machine language programs are generally written with the help of mnemonics which correspond to the bit patterns for various instructions. For example, 10 000 111 is an add instruction for the *ALTAIR 8800* and the corresponding mnemonic is ADD A. Obviously the mnemonic ADD A is much more convenient to remember than its corresponding machine language bit pattern.

Ultimately, however, the machine language bit pattern for each instruction must be entered into the computer one step at a time. Some instructions may require more than one binary word. For example, an *ALTAIR 8800* instruction which references a memory address such as JMP requires one word for the actual instruction and two subsequent words for the memory address.

Machine language programs are normally entered into the *ALTAIR 8800* by means of the front panel switches. A computer terminal can be used to send the mnemonics signal to the computer where it is converted into machine language by a special set of instructions (software) called an assembler.

Even more flexibility is offered by a highly complex software package called a compiler which converts higher order mnemonics into machine language. Higher order mnemonics are a type of computer language shorthand which automatically replace as many as a dozen or more machine language instructions with a single, easily recognized mnemonic. Advanced computer languages such as FORTRAN, BASIC, COBAL, and others make use of a compiler.

The higher computer languages provide a great deal of simplification when writing computer programs, particularly those that are lengthy. They are also very easy to remember. The potential versatility of machine language programming should not be underestimated, however, and an excellent way to realize the full potential of a higher language is to learn to apply machine language.

PART 2 ORGANIZATION OF THE ALTAIR 8800

A block diagram showing the organization of the *ALTAIR 8800* is shown in Figure 2-1. It is not necessary to understand the detailed electronic operation of each part of the computer to make effective use of the machine. However, a general understanding of each of the various operating sections is important.

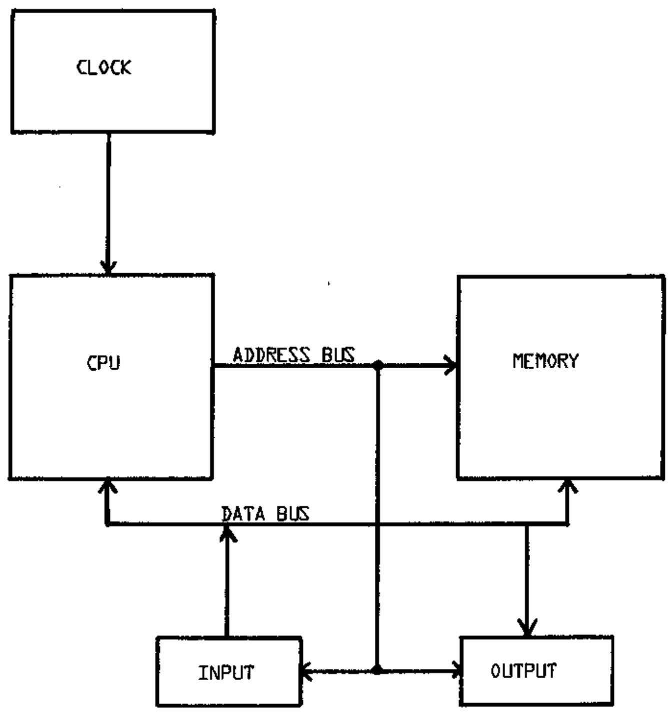


FIGURE 2-1

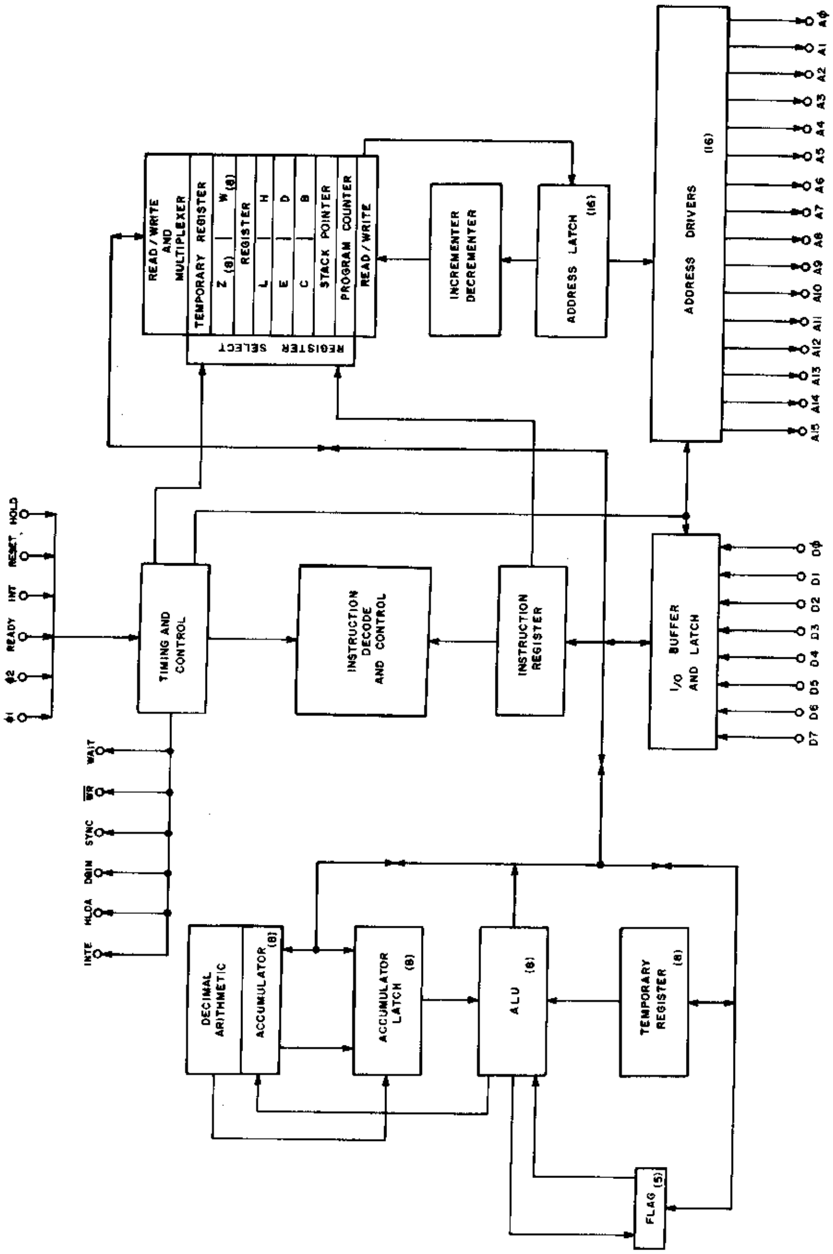


FIGURE 2-2. CPU Diagram

A. CENTRAL PROCESSING UNIT (CPU)

The Central Processing Unit (CPU) performs all arithmetic calculations, makes all logical decisions, controls access to the computer by input and output devices, stores and retrieves data from the memory, and coordinates the orderly execution of a program. The CPU is quite literally the heart of the computer.

Of course it is important to remember that the CPU is only as intelligent as the programmer, for the CPU must be instructed in precise terms just how to perform a particular operation. But since the CPU in the *ALTAIR 8800* can execute a complete instruction cycle in only 2 microseconds*, the computer can solve a highly complex problem in an incredibly brief time. In fact, the *ALTAIR 8800* can execute a six instruction addition program approximately 30,000 times in one second.

The compact size and economy of the *ALTAIR 8800* is in large part due to the CPU. Thanks to large scale integrated circuit techniques (LSI), the CPU used in the *ALTAIR 8800* is fabricated on a tiny silicon chip having a surface area of only a fraction of an inch. This chip, the Intel 8080, is installed in a protective dual-in-line mounting package having 40 pins.

The CPU is by far the most complex portion of the *ALTAIR 8800*. A complete block diagram of the CPU is shown in Figure 2-2, and while it is not necessary to possess a detailed understanding of this diagram it is important to understand the role of some of the CPU's more important systems. The interrelationship of each of these systems and their contribution to the operation of the CPU will then become more obvious.

1. TIMING AND CONTROL--The timing and Control System receives timing signals from the clock and distributes them to the appropriate portions of the CPU in order to insure coordinated instruction execution. The Timing and Control System also activates several front panel status indicators (HOLD, WAIT, INTE, STACK, OUT, IN, INP, MI MENR, HLTA, WO, INT).

*A microsecond is one millionth of a second.

2. INSTRUCTION REGISTER--Binary machine language instructions are temporarily stored in the Instruction Register for decoding and execution by the CPU.

3. ARITHMETIC--The Arithmetic System performs both binary and decimal arithmetic. All arithmetic operations are performed by addition. Multiplication is implemented by repetitive addition. Subtraction and division are implemented by inverse addition.

4. WORKING REGISTERS--The CPU contains seven 8-bit Working Registers. The most important of these is the Accumulator, the register into which the results of many operations are eventually loaded. In addition to acting as a primary storage point for results of many program operations, numerous arithmetic and logical operations can be performed with the Accumulator and any specified register or memory address.

The six remaining registers, which are arranged in pairs to permit 16-bit operation when necessary, are "scratch-pad" registers. This simply means they are used to store temporary data or addresses on a regular basis and are available for numerous program operations.

Figure 2-3 shows the arrangement and classification of the seven Working Registers. The additional register adjacent to the Accumulator, the Status Bit Register, is a special purpose register used to store the status of certain operations.

Register Pair B →	B	C
Register Pair D →	D	E
Register Pair H →	H	L
Register Pair PSW →	*	A

*Status Bit Register (See Text)

FIGURE 2-3. The Working Registers

5. STATUS BIT REGISTER--The Status Bit Register is a special purpose register which stores the status of five conditions which may or may not be affected by the result of a data operation. This register contains 8-bit positions, but only 5-bits are used to store the status information. The five status bits are:

a. Carry Bit--This bit is set to 1 if a carry has occurred. The Carry Bit is usually affected by such operations as addition, subtraction, rotation, and some logical decisions. The bit is set to 0 if no carry occurs.

b. Auxiliary Carry Bit--If set to 1, this bit indicates a carry out of bit 3 of a result. 0 indicates no carry. This status bit is affected by only one instruction (DAA).

c. Sign Bit--This bit is set to show the sign of a result. If set to 1, the result is minus; if set to 0 the result is plus. The Sign Bit reflects the condition of the most significant bit in the result (bit 7). This is because an 8-bit byte can contain up to the decimal equivalent of from -128 to +127 if the most significant bit is used to indicate the polarity of the result.

d. Zero Bit--This bit is set to 1 if the result of certain instructions is zero and reset to 0 if the result is greater than zero.

e. Parity Bit--Certain operations check the parity of the result. Parity indicates the odd or even status of the 1 bits in the result. Thus if there is an even number of 1 bits, the Parity Bit is set to 1, and if there is an odd number of 1 bits, the Parity Bit is set to 0.

6. PROGRAM COUNTER--The Program Counter is a special 16-bit register which stores the address of the next program step to be executed. The Program Counter is automatically advanced to the next sequential program address upon completion of a step execution. Sometimes called the P-Counter, the Program Counter is directly accessible to the programmer via machine language instructions which implement JUMP, CALL, and RETURN instructions.

7. STACK POINTER--The Stack Pointer is another special 16-bit register. A section of memory reserved for the temporary storage of data or addresses is called the stack.

Data can be pushed onto the stack for temporary storage and popped out of the stack via several instructions.

The Stack Pointer is used to store the contents of the Program Counter during the execution of subroutines. A RETURN instruction transfers the contents of the Stack Pointer to the Program Counter and sequential execution of the main program continues. The programmer selects the location of the stack in memory by loading the Stack Pointer with the desired memory address via a special instrution (LXI).

The interrelationship of the Working Registers, Program Counter, Stack Pointer, Arithmetic System, Instruction Register, and Timing and Control System should now be more meaningful. The Working Registers incorporate six scratchpad registers and an Accumulator into which numerous operation results are temporarily stored. The Program Counter causes sequential execution of a program by keeping track of the memory address of the next instruction to be executed. The Timing and Control System supplies timing pulses which coordinate orderly program execution. The Stack Pointer is used for temporary storage of the data contained in any register pair. The Stack Pointer also saves the address in the Program Counter for retrieval after a subroutine has been executed. All these operations combine to provide an enormously flexible and versatile CPU.

B. MEMORY

Though the Working Registers, Program Counter, and Stack Pointer certainly perform memory roles, the CPU does not contain memory as it is normally defined in a computer application. The primary memory in a computer is external to the CPU.

Simple programs can be implemented with a few dozen words of memory or even less, but more complex applications such as video processing require more memory. The *ALTAIR 8800* is expandable to 65,536 8-bit words of memory.

Access to the memory is always controlled by the CPU.* 16 address lines called the Address Bus connect the CPU to the Memory. These lines permit the CPU to input or output data to or from any memory address. The addresses are specified by two 8-bit bytes. The CPU processes each address as two sequential (serial) cycles, each containing 8-parallel bits. Data stored in the Memory is exchanged between the Memory and CPU via 8 data lines called the Data Bus. This interconnection format permits parallel operation. Thus, when data is inputted or outputted in or from Memory by the CPU, it is transmitted as a complete 8-bit word.

The basic Memory in the *ALTAIR 8800* contains up to eight 256 x 4 bit random access memories (RAMs). However, any conventional memory can be used in the computer if input loading on the buss does not exceed 50 TTL loads and if the buss is driven by standard TTL loads.

*An exception to this is when the computer is connected to a Direct Memory Access Controller. DMA takes control of the address and data lines from the CPU for direct transfers of blocks of data. These transfers can take place internally (from one memory location to another) or externally (from memory to an external device).

C. CLOCK

Orderly execution of a program by the CPU is controlled by a 2-MHz crystal controlled clock. Crystal control is used to permit the clock to operate at the maximum permissible CPU speed. A clock without crystal regulation might occasionally speed up beyond the CPU's capability and program execution errors would result.

D. INPUT/OUTPUT

The *ALTAIR 8800* can be interfaced with a great many external devices. Generally, these devices provide input information to the computer and accept output information from the computer. The CPU monitors the status of program execution and Input/Output devices and provides the necessary signals for servicing external devices. The programmer can instruct the CPU to either ignore or respond to interrupt signals provided by an external device. These interrupt signals, when accepted by the CPU, cause the program execution to be temporarily halted while the external device is serviced by the computer. When the external device has been serviced, the program resumes normal execution. The *ALTAIR 8800* will service up to 256 Input and 256 Output devices.

This concludes the description of the organization of the *ALTAIR 8800*. The overall operation of the computer as a powerful and efficient data processing system will become more apparent in Part 3, a discussion of the operation of the *ALTAIR 8800*.

PART 3. OPERATION OF THE *ALTAIR 8800*

Access to the basic *ALTAIR 8800* is achieved via the front panel, and at first glance the array of 25 toggle switches and 36 indicator and status LEDs may appear confusing. Actually, operation of the *ALTAIR 8800* is very straightforward and most users learn to load a program into the machine and run it in less than an hour. If you are a typical user, you will spend far more time developing and writing programs than actually operating the machine.

This part of the *ALTAIR 8800* Operating Manual explains the purpose and application of the front panel switches and indicator and status LEDs. A sample program is then loaded into the machine and run. A detailed discussion of the role and efficient use of memory is included next. Finally, several operating hints which will help you edit and "debug" programs are included.

I/O PORT STRUCTURE AUGUST 1976

VDM-1

Memory Address	CC00 (Hex)	146,000 (Octal)
Control Port	C8 (Hex)	310 (Octal) 200 (Decimal)

CROMEMCO DAZ.

Ports: 016, 017 (Octal)

CCC (CROMEMCO)

Ports: 020, 021, 022 (Octal)

SERIAL I/O (SINGLE CHANNEL)

Ports: 000, 001 110 Baud

SERIAL I/O (QUAL CHANNEL)

Ports: 010, 011, 012, 013

ACR

Ports: 006, 007 300 Baud

4P I/O

Ports: 040 to 057

SINGLE P I/O

Ports: 002, 003

D/A

Ports: 024, 025, 026, 027

PORTS AVAILABLE

004
005
014
015
030 - 037
060 - 277
301 - 377

CARD STRUCTURE AUGUST 1976

Number	Type
1	1-SI/O
1	2-SI/O
1	1-PI/O
1	4-PI/O
1	ACR
1	VDM-1
1	CDZ
1	D/A
7	4K RAM
1	CPV

Altair Software Library
#7-19-762
Author: Matthew Smith
Length: 12 Lines (BASIC)
Title: Bases

Computer Notes
August, 1976

Program Description

This sub-routine converts a number of said base (from 2 to 16) to its value in another base. It was designed to be used by BASIC programs such as Editors, Assemblers, Monitors or other machine language function programs. An example would be converting a decimal value obtained from the PEEK function to binary, octal, or hex for a memory dump program. Upon calling the BASES routine at line number 9000, the following variables must have been set up by the calling program:

C$ - should contain the value to be converted. Note, this string must not contain trailing spaces.

BT - the base that the value in C$ is to be converted to.

BF - the base that C$ is before conversion.

The routine will use the following variables:

E, B$, D$, C$, BT, BF, X, Y, F, D

In addition, 0Ø, 01, 02 will be used to represent the values of Ø, 1, and 2.

When returning from the routine, the following variables will be set:

C$ - converted value (in BT's base) of old C$.

F - C$'s value in decimal.

E - error code. E=Ø means no error, E=1 means BF is not in the range of Ø<=BF<=16, E=2 means BT not of range Ø<=BT<=16, E=3 means digits of C$ are not of base BF. For example C$="1F5" and BF=8.

*Note - this routine works by converting C$ from BF (base from) to decimal (value contained in F) to C$ at BT (base to), so if a large value is being converted, round off errors (due to conversion to E notation at variable F) or overflow errors (at line 9070 when D is computed) are possible.

```
9000 E=0:B$="":D$="0123456789ABCDEF":BT=INT(BT):BF=INT(BF):01=1:00=0:D=0:02=2
9010 IFBF>LEN(D$)ORBF<00THENE=1:RETURN
9020 IFBT>LEN(D$)ORBT<00THENE=2:RETURN
9030 FORY=01TOLEN(CS):FORX=00TOBF-01
9040 IFMID$(C$,Y,01)=" "THENY=Y+01:GOTO9040
9050 IFMID$(C$,Y,01)=MID$(D$,X+01,01)THEN9070
9060 NEXT:E=3:RETURN
9070 D=D*BF+X:NEXTY:C$="":F=D
9080 B$=B$+STR$(INT((D/BT-INT(D/BT))*BT)):D=INT(D/BT):IFD>=01/BTTHEN9080
9090 FORX-LEN(B$)TO02STEP-02
9100 IFMID$(B$,X,01)=" " THENX=X-01:GOTO9100
9110 C$=C$+MID$(D$,VAL(MID$(B$,X-01,02))+01,01):NEXT:RETURN
```

8800 CODIN FORM

12K Basic Load (for PROM)

TAG	MNEMONIC	ADDRESS (Octal)	OCTAL CODE	EXPLANATION
3 Start		000 001	041 256	Beginning of tape bootstrap
		002	057	037 for 8K Basic (See elsewhere for description)
		003	061	
		004	022	
		005	000	
		006	333	
		007	006	
		010	017	
		011	330	
		012	333	
		013	007	
		014	275	
		015	310	
		016	055	
		017	167	
		020	300	
		021	351	
		022	003	
		023	000	

TAG	MNEMONIC	ADDRESS	OCTAL CODE	EXPLANATION
2 Start		024	333	Beginning of 256 Test
		025	007	(See elsewhere for description)
		026	376	
		027	256	
		030	302	
		031	024	
		032	000	
		033	303	
		034	000	
		035	000	
		036	333	Beginning of serial output
		037	000	(See elsewhere for description).
		040	346	
		041	200	
		042	302	
		043	036	
		044	000	
		045	012	Load A with M at B & C

8800 CODING FORM

TAG	MNEMONIC	ADDRESS	OCTAL CODE	EXPLANATION
		046	000	Test for 377 Stop Code
		047	326	Sub Immediate from accumulator
		050	377	1's code (stop)
		051	312	Jump if zero to 256 test
		052	024	
		053	000	
		054	012	Otherwise, restore accumulator
		055	323	and output data
		056	001	
		057	003	Increment B & C
		060	303	Jump to Serial output
		061	036	
		062	000	
1 Start Here		063	001	Initialize B & C to 000 100
		064	100	
		065	000	
		066	303	Jump to Serial Output
		067	036	
		070	000	

TAG	MNEMONIC	ADDRESS	OCTAL CODE	EXPLANATION
		071	000	
		072	000	
		073	000	
		074	000	
		075	000	
		076	000	
		077	000	
Message Start		100	114	(Odd Parity) L 01/001/100
		101	117	O 01/001/111
		102	301	A 11/000/001
		103	304	D 11/000/100
		104	240	Space 10/100/00
		105	124	T 01/010/100
		106	301	A 11/000/001
		107	320	P 11/010/000
		110	105	E 01/000/101
		111	015	CR 00/001/101
		112	007	Bell
		113	012	form feed

8800 CODING FORM

VLCT I/O TEST PROGRAM

(VLCT Loc 2/3)

TAG	MNEMONIC	ADDRESS	OCTAL CODE	EXPLANATION
Input Start	In	0	333	Load byte from I/O into accumulator.
				Address of device follows
		1	002	Control immediate with accumulator
Ready Test	ANI	2	346	And immediate with accumulator
		3	002	00/000/010 Ended against control channel.
				DIO and DI1 are status bits.
				01 = output device ready
				10 = input device has sent data and
				computer can now access it.
Test Result	JZ	4	312	Jump if zero. If status of zero bit is 1, zero
				is present and jump to address occurs. Zero
				bit is 1 if instruction result is zero, and
				reset if greater than zero.
				Jump occurs if input not ready.
		5	000	Jump to input start occurs if input
		6	000	is not ready
	IN	7	333	Input to accumulator from I/O
		10	003	I/O Data channel address

8800 CODING FORM

TAG	MNEMONIC	ADDRESS	OCTAL CODE	EXPLANATION
	STA	11	062	Store accumulator direct
		12	040	Storage location
Input Finish		13	000	(Input stored in location 40)
Output Start	IN	14	333	Load byte from I/O into accumulator
		15	002	I/O control channel address
Output test	ANI	16	346	And immediate with accumulator.
				Same as step 2, but output status is being tested
		17	001	100/000/001 ended against accumulator.
Test Result	JZ	20	312	Jump if zero to address
		21	014	Jump is back to output test if
		22	000	not ready.
	LDA	23	072	Load accumulator direct with data at address below.
		24	040	Storage address
		25	000	
	Out	26	323	Output sent accumulaotr data
Output Finish		27	003	Output device address
Re-cycle	Jump	30	303	Jump to address below

8800 CODING FORM

TAG	MNEMONIC	ADDRESS	OCTAL CODE	EXPLANATION
Command		31	000	Goes back to input start.
		32	000	

8800 CODING FORM

VLCT LOAD PROGRAM (LOAD STARTS FROM XXX|XXX) (Device 2/3)

TAG	MNEMONIC	ADDRESS	OCTAL CODE	EXPLANATION
Load Address	LXIB	0	041	Load register pair immediate 00|(rp)0|001
				In this case, register pair 10, or H and L.
				Low byte goes in L, high byte in H.
		1	XXX	least significant bits (Address low)
		2	XXX	most significant bits (Address high)
Input test	IN	3	333	Input loaded into accumulator;
		4	002	control channel address of VLCT
	ANI	5	346	And immediate with accumulator to
		6	002	test status; 002 is the test
End test	JZ	7	312	Jump on Zero. If accumulator
		10	003	now zero, jump to address occurs.
		11	000	2 Address bytes
Read Input	IN	12	333	Input loaded into accumulator. This
		13	003	step is reached if input ready; data channel
Mov A to M	MOV	14	167	MOV Contents from source register to
				destination register w/o changing source
				register: 01|DDD|SSS
				B=0; C=1; D=2; E=3; H=4; L=5; M=6;
				A=7. M = Memory Reference.

TAG	MNEMONIC	ADDRESS	OCTAL CODE	EXPLANATION
Output test	IN	15	333	Input from device loaded into
		16	002	accumulator; Control Channel
	ANI	17	346	And immediate with accumulator;
		20	001	Output ready test
End test	JZ	21	312	Jump on zero to address below.
		22	015	
		23	000	
MOV M to A	MOV	24	176	Move memory register contents to A.
Output to	Out	25	323	Write A contents on device 3; Data
Data channel		26	003	channel
	STAX	27	022	Store accumulator in memory address
				given by registers H & L.
				Command: 0X2
				B&C = 0 D&E = 1 H&L = 2
Increment	INX	30	043	Increment register pair by one
Register Pair				B&C = 00; D&E = 01; H&L = 10;
				Flags and A = 11. Command =
				00|(rp)0|011

TAG	MNEMONIC	ADDRESS	OCTAL CODE	EXPLANATION
Return to	Jump	31	303	Jump to address shown
Step 3		32	003	
		33	000	

To run, examine location 000, push run. Three VLCT keystrokes will enter the octal code into accumulator, starting from location XXX|XXX. Push ready key for test echo.

8800 CODING FORM

COMPTER 256/SERIAL I/O TEST (Address 0/1)

The following machine language program sends a byte to memory and echoes it back

TAG	MNEMONIC	ADDRESS	OCTAL CODE	EXPLANATION
	IN	0	333	Input call
		1	000	Control Channel
	ANI	2	346	And Immediate
		3	001	Test (In ready)
	JZ	4	302	Jump on Zero
		5	000	Address
		6	000	
	IN	7	333	Input Call
		10	001	Data Channel
	MOV	11	062	Mov A to Memory
		12	040	Memory address
		13	000	
	IN	14	333	Input Call
		15	000	Control Channel
	ANI	16	346	And Immediate
		17	200	Test (Out ready)
	JZ	20	302	Jump on Zero

8800 CODING FORM

TAG	MNEMONIC	ADDRESS	OCTAL CODE	EXPLANATION
		21	014	Address
		22	000	
	MOV	23	072	Mov Memory to A
		24	040	Memory Address
		25	000	
	OUT	26	323	Output
		27	001	Data Channel
	JP	30	303	Jump (Unconditional)
		31	000	Address of program begin
		32	000	

8800 CODING FORM

PIO A/D OUTPUT

TAG	MNEMONIC	ADDRESS	OCTAL CODE	EXPLANATION
	OUT	0	323	Output
		1	001	

Accumulator is sent to PIO latch. Signal will show up immediately at output port if SBO is held high. A control signal can be used for the D/A strobe. STB (High) and ND (low) are suggested.

8800 CODING FORM

Binary Add (Octal Readout; VLCT) (Device 2/3)

TAG	MNEMONIC	ADDRESS	OCTAL CODE	EXPLANATION
Input Test	IN	0	333	Input from device loaded in A
		1	002	Control Channel
	ANI	2	346	And immediate with accumulator
		3	002	Input ready test
	JZ	4	312	Jump on zero
		5	000	Low Address
End Test		6	000	High Address
Input Read	IN	7	333	Input loaded into accumulator
		10	003	Data channel
Mov A to B	MOV	11	107	Move accumulator data to register B
Input Test	IN	12	333	IN to A
		13	002	Control channel
	ANI	14	346	And immediate
		15	002	Test
	JZ	16	312	Jump to zero
		17	012	Address of input test
END TEST		20	000	
Input Read	IN	21	333	Input loaded into accumulator
		22	003	Data channel

8800 CODING FORM

TAG	MNEMONIC	ADDRESS	OCTAL CODE	EXPLANATION
ADD (B+A)	ADD	23	200	Add register B to Accumulator
MOV SUM TO B	MOV	24	107	Sum moved to register B
Output Test	IN	25	333	Input loaded into accumulator
		26	002	Control channel
Test	ANI	27	346	And immediate with accumulator
		30	001	Test for output ready
	JZ	31	312	Jump on zero to output test
		32	025	Address
		33	000	
If program reaches here, sum is ready to be printed				
MOV B to A	MOV	34	170	Mov sum to accumulator
Output	OUT	35	323	Accumulator sent to output
		36	003	Data channel
Jump return		37	303	
		40	000	
		41	000	

To run, load first number; press ready. Load second number; press ready. Result appears.

PROGRAM # 92751
PROGRAM NAME: KILL THE BIT

PROGRAMMER: DEAN B. MCDANIEL

DATE WRITTEN: MAY 15, 1975

COMPUTER: ALTAIR 8800

CORE REQUIRMENTS: 24 WORDS (RELOCATABLE)

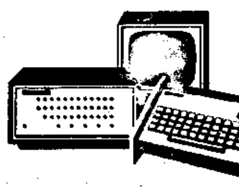
INPUT/OUTPUT REQUIRMENTS: SENSE SWITCHES

```
000000        LXI   H     041    INTIALIZE COUNTER
000001         D          000
000002         D          000
000003        MVI   D     026    SET-UP INTIAL DISPLAY BIT
000004         D          200
000005        LXI   B     001    LOAD SPEED DATA  (HIGHER
000006         D          016    THE VALUE THE FASTER THE
000007         D          000    DATA ROTATES)
000010  BEG:  LDAX  D     032    DISPLAY BIT PATTERN ON
000011        LDAX  D     032    UPPER 8 ADDRESS LIGHTS
000012        LDAX  D     032    ( IE. SENSE SWITCHES )
000013        LDAX  D     032
000014        DAD   B     011    INCREMENT DISPLAY COUNTER
000015        JNC   BEG   322
000016         A          010
000017         A          000
000020        IN    255D  333    INPUT DATA FROM SENSE SWITCHES
000021         A          377
000022        XOR   D     252    EXCLUSIVE OR WITH A-REG.
000023        RRC         017    ROTATE DISPLAY RIGHT ONE BIT
000024        MOV   D,A   127    MOVE DATA TO DISPLAY REG.
000025        JMP   BEG   303    REPEAT SEQUENCE
000026         A          010
000027         A          000
```

PROGRAM TYPE: DEMOSTRATION (GAME)

OBJECT: TO KILL THE ROTATING BIT. IF YOU MISS THE LIT BIT ANOTHER ONE AT THAT SWITCH POSITION WILL TURN ON, NOW LEAVING YOU 2 BITS TO DESTROY. MERELY TOGGLE THE SWITCH DON'T LEAVE THE S.S. IN THE UP POSITION. BEFORE STARTING MAKE SURE ALL THE S.S. ARE IN THE DOWN POSITION.

* HAVE FUN *



Computer Bits

By Jerry Ogdin

SOFTWARE TOOLS

NO MATTER how small the computer, even the most dedicated programmer will rapidly become bored with binary notation.

Some hobbyists use a Teletype as there are several older models on the market at reasonable prices. Even with a Teletype, though, you need some software to convert those keystrokes into something meaningful in memory. If you can't afford a Teletype, you can almost always use an encoded keyboard. These are frequently on the surplus market for less than $25.00.

A terminal is important, but it is only one tool in the computer hobbyist's kit. Once you've written a program and gotten it into storage, you ought to use the cassette interface (HIT) described in September's column. With this tool, you have to "button in" the program bit-by-bit only once. After it is in storage, you can write it out to be taped and read in the next time you want it. Of course, if your only storage medium is RAM, you'll lose the memory contents when you turn the computer off. So, it's a good idea to copy the latest version of a program out to tape as a backup.

Also, if your only storage medium is RAM, you'll have to reenter the tape reading routine laboriously through the switches (or the terminal) each time the computer power is turned on. That is a good reason for having a small program, called a *bootstrap loader*, kept in read-only memory. This program makes it possible to read data from the tape and then execute that data. Such a read-in program is usually somewhat larger and more powerful, so it reads in several records (perhaps using the bootstrap program as a subroutine), which make up an even larger and more sophisticated program. The effect, then, is to use one group of records to read the next

```
                ; THIS IS THE POPULAR ELECTRONICS SUPER-
                ; SIMPLE MONITOR.  COMMANDS ARE:
                ;     D XXXX    (DUMP FROM XXXX)
                ;     L XXXX    (LOAD FROM XXXX)
                ;     G XXXX    (GO TO XXXX)
                STACK  EQU  03FFH        ;YOUR STACK ORIGIN
                CRLF   EQU  0106H        ;YOUR CR, LF ROUTINE
                WRCHR  EQU  0103H        ;YOUR OUTPUT ROUTINE
                RDCHR  EQU  0100H        ;YOUR READING ROUTINE
                ;
                ; MONITOR ENTRY POINT
                ;
0040: 31FF03    PEMON: LXI  SP,STACK     ;INITIALIZE
      CD0601           CALL CRLF         ;ISSUE CARRIAGE RETURN,
      3E3F             MVI  A,'?'        ;   LINE FEED AND ?
      CD0301           CALL WRCHR
      CD0001           CALL RDCHR        ;AWAIT COMMAND
      E67F             ANI  07FH         ;STRIP OFF PARITY BIT
      FE4C             CPI  'L'
      CA6300           JZ   LOAD         ;"LOAD" COMMAND
      FE44             CPI  'D'
      CA7400           JZ   DUMP         ;"DUMP" COMMAND
      FE47             CPI  'G'
      C24000           JNZ  PEMON        ;ERROR
                ; THIS THE THE "GO" COMMAND PROCESSOR.  WE
                ; NOW EXPECT A 16-BIT DESTINATION ADDRESS
                ; AND THEN TRANSFER TO IT
005F: CDB700           CALL RDNUM
      E9               PCHL              ;"GO"
                ; THIS IS THE "LOAD" COMMAND PROCESSOR.  THE
                ; USER IS EXPECTED TO TYPE IN A 16-BIT ADDRESS
                ; FOLLOWED BY DATA BYTES TO LOAD INTO SUCCESSIVE
                ; LOCATIONS.  ANY NON-HEX CHARACTER SEPARATES
                ; THE BYTES FROM ONE ANOTHER.  ANY BYTE THAT IS
                ; TERMINATED WITH A COLON WILL BE IGNORED.
0063: CDCD00    LOAD:  CALL RDNR2        ;GET USER'S ADDRESS
      CDB700           CALL RDNUM        ;GET A BYTE
      FE3A             CPI  ':'
      CA6600           JZ   LOAD+3       ;SKIP IF FOLLOWED BY ':'
      7D               MOV  A,L          ;GET LAST TWO HEX DIGITS
      02               STAX B            ;STORE THEM AWAY
      03               INX  B
      C36600           JMP  LOAD+3       ;(GET W/ CPU RESET)
                ; THIS IS THE "DUMP" COMMAND PROCESSOR.
                ; THE USER IS EXPECTED TO SUPPLY A 16-BIT
                ; STARTING ADDRESS.  WHENEVER THE LEAST-SIG-
                ; NIFICANT FOUR BITS OF THE ADDRESS OF THE
                ; NEXT BYTE ARE ZERO, THE CARRIAGE IS RETURNED
                ; AND THE ADDRESS IS PRINTED (FOLLOWED BY COLON).
0074: CDCD00    DUMP:  CALL RDNR2        ;GET STARTING ADDRESS
      79               MOV  A,C          ;CHECK TO SEE IF
      E60F             ANI  15           ;  NEW-LINE TIME.
      C2A200           JNZ  MORE         ;NO.  JUST PRINT
      CD0601           CALL CRLF         ;START NEW LINE
      78               MOV  A,B          ;GET MSB OF ADDRESS
      1F               RAR
      1F               RAR
      1F               RAR
      1F               RAR
      CDED00           CALL CVTAS
      78               MOV  A,B          ;GET 2ND DIGIT
      CDED00           CALL CVTAS
      79               MOV  A,C          ;GET 3RD DIGIT
      1F               RAR
      1F               RAR
      1F               RAR
      1F               RAR
      CDED00           CALL CVTAS
      79               MOV  A,C          ;DO LAST DIGIT
      CDED00           CALL CVTAS
      3E3A             MVI  A,':'        ;MARK ADDRESS SPECIALLY
      CD0301           CALL WRCHR
```

```
      3E20             MVI  A,' '
      CD0301           CALL WRCHR
00A2: 0A        MORE:  LDAX B            ;FETCH BYTE TO DUMP
      1F               RAR
      1F               RAR
      1F               RAR
      1F               RAR
      CDED00           CALL CVTAS        ;ISSUE MSB
      0A               LDAX B
      CDED00           CALL CVTAS
      3E20             MVI  A,' '
      CD0301           CALL WRCHR
      03               INX  B            ;ACQUIRE NEXT CELL
      C37700           JMP  DUMP+3
                ; THIS ROUTINE READS ONE OR MORE HEXADECIMAL
                ; DIGIT CHARACTERS ('0'..'9', 'A'..'F') IN
                ; AND ACCUMULATES A SIXTEEN-BIT NUMBER IN
                ; THE (H,L).  EACH NEW DIGIT IS SHIFTED INTO
                ; THE LEAST-SIGNIFICANT FOUR BITS OF THE (H,L).
                ; CONTROL IS RETURNED WHENEVER THE DEPENDENT
                ; ROUTINE ("RDHEX") SETS THE CARRY BIT TRUE.
00B7: 210000    RDNUM: LXI  H,0          ;START OFF VALUE AT 0
      CDD600           CALL RDHEX        ;GO GET A HEX DIGIT
      DAB700           JC   RDNUM        ;AWAIT HEX DIGIT
00C0: 29        RDNXT: DAD  H            ;SHIFT (H,L) LEFT 4
      29               DAD  H
      29               DAD  H
      29               DAD  H
      B5               ORA  L            ;PLACE NEW FOUR BITS IN
      6F               MOV  L,A
      CDD600           CALL RDHEX        ;GO GET NEXT DIGIT
      D8               RC                ; THE NUMBER'S FINISHED
      C3C000           JMP  RDNXT        ;GO PROCESS NEXT DIGIT
                ; THIS ROUTINE READS A NUMBER VIA "RDNUM" AND
                ; PLACES IT INTO THE (B,C) PAIR
00CD: CDB700    RDNR2: CALL RDNUM        ;GET THE VALUE IN (H,L)
      44               MOV  B,H          ; THEN MOVE IT
      4D               MOV  C,L
      C9               RET
                ; THIS ROUTINE READS IN AN ASCII CHARACTER,
                ; STRIPS OFF PARITY AND EXAMINES IT FOR
                ; MEMBERSHIP IN THE HEX-DIGIT SET.  IF IT IS
                ; A HEX-DIGIT THE A-REGISTER IS LEFT AT THE FOUR
                ; BIT VALUE APPROPRIATE AND THE CARRY IS CLEARED.
                ; ANY OTHER CHARACTER IS LEFT UNTOUCHED AND THE
                ; CARRY IS SET.
00D3: D630      RDDIG: SUI  '0'          ;TRANSLATE '0'..'9'
      C9               RET
00D6: CD0001    RDHEX: CALL RDCHR        ;***ENTRY POINT***
      E67F             ANI  07FH         ;REMOVE PARITY BIT
      FE30             CPI  '0'
      D8               RC                ;CHAR LESS THAN '0' (NOT HEX)
      FE3A             CPI  '9'+1
      DAD300           JC   RDDIG        ;IN RANGE 0..9
      FE41             CPI  'A'
      D8               RC                ;BETWEEN 9 AND A
      FE47             CPI  'F'+1
      3F               CMC
      D8               RC                ;NOT HEX CHARACTER
      D637             SUI  'A'-10       ;TRANSLATE 'A'..'F'
      C9               RET
                ; THIS ROUTINE CONVERTS A BINARY NUMBER IN THE
                ; LEAST-SIGNIFICANT BITS OF THE A-REGISTER TO
                ; AN ASCII CHARACTER REPRESENTING THE HEX VALUE
                ; AND THEN ISSUES IT AS OUTPUT.
00ED: E60F      CVTAS: ANI  15           ;ISOLATE FOUR BITS
      C630             ADI  '0'          ;SHIFT DIGITS INTO ASCII
      FE3A             CPI  '9'+1        ; SEE IF IT WAS 0..9
      DA0301           JC   WRCHR
      C607             ADI  'A'-'0'-10   ;CONVERT TO 'A'..'F'
      C30301           JMP  WRCHR
```

group in, thus "pulling" the program in by its own bootstraps.

Using a Monitor. The ability to preserve a program for later recall is important, but it doesn't solve two major nuisances: (1) you still have to key in the program bit-by-bit the first time; and (2) every time you make an error in the program, you have to key in the changes, some of which may be traumatic and complex. One of the best ways to solve this kind of inconvenience is to provide a small *monitor*. A monitor is just another computer program, but one that is designed to make computer use more convenient. This program reads characters from a terminal (or a separate keyboard), with these characters specifying the bit patterns to put into memory.

The simplest monitor has three basic *commands:* Load, Dump and Go. A command is a single letter typed at a time when the monitor is not otherwise engaged in some activity. Typical commands are single letters like "L" for Load, "D" for Dump and "G" for Go. When you type in "L" you are directing the monitor program to accept keyboard input and load it into memory; "D" means you want to display contents of memory on your terminal or display device; "G" is your means of transferring control out of the monitor into the program you have previously loaded.

Most programmers now use the hexadecimal number system for communicating with the machine, although there are "pockets" of users of octal. Hex and octal are, of course, just shorthand notations for binary code. Hex digits allow us to specify four bits with one symbol, octal allows three. The hexadecimal digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. (The letters A through F stand for decimal equivalent values 10 through 15, respectively.) The monitor, for utmost simplicity, uses only hexadecimal digits for the specification of addresses and data byte values.

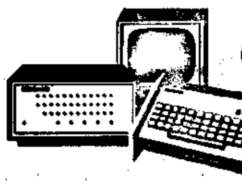
Each of the command letters (L, D or G) is followed by an address that specifies where to start. For the Load command, that address is where the first byte of data from the keyboard will be stored; for Dump, it is the address from which data will begin being displayed; for Go, it is the address that is to be placed into the CPU's program counter. Whenever the Go command's address has been supplied, control is transferred to that location. Whenever the Dump command's address has been given, data displaying will begin. However, after the Load command's address, the monitor will expect more bytes (one after another) to be loaded into successive locations in memory.

A small monitor for the 8080 microprocessor that you can use as a model is shown opposite. Each command is stopped by resetting the CPU, thus returning control to the top of the monitor. Notice some error correction conventions that have been instituted to save you some time: numbers are assumed to consist of any number of hex digits but if the monitor wants an address, only the least-significant four digits are used. Likewise, for a data byte, only the least-significant two hex digits are preserved. This means that if you've made an error, just keep typing. Hex digits end with any character that is not a hex digit; most people find that the space character is the most convenient.

News Items. The extremely popular 8080, originally from Intel, is now being supplied by other semiconductor makers as well. The TI TMS8080 is identical to the original 8080, which Intel no longer makes. Intel's newer device, the 8080A, is functionally identical but has better current drive capacity. Intel has another part, the 8080A-1, that'll go faster so that AMD's 9080 (which is supposed to run 50% faster than the original Intel part) will have a competitor. So, if you are using the 8080, be sure to check the diagrams to see that your part matches the requirements.

The new MOS Technology 6501 is destined to become a popular CPU among hobbyists, if only because of its dramatically low price ($20 at press time). The device is modelled after Motorola's 6800, although with some major differences. All of the Motorola support parts like memory I/O chips can be used with the 6501, so you can get on board quickly. The Motorola software, however, cannot be executed on the 6501 without revision.

The 6501 is capable of operation at twice the Motorola part's speed; some parts may operate three times as fast. The introduction of this part is likely to start the real price war that has been brewing in the microprocessor business. Even with the new support chip for the 8008 that Intel has announced, it seems unlikely that it can compete with the 6501 for hobbyist use. ◈



Computer Bits

By Jerry Ogdin

SOFTWARE TOOLS

NO MATTER how small the computer, even the most dedicated programmer will rapidly become bored with binary notation.

Some hobbyists use a Teletype as there are several older models on the market at reasonable prices. Even with a Teletype, though, you need some software to convert those keystrokes into something meaningful in memory. If you can't afford a Teletype, you can almost always use an encoded keyboard. These are frequently on the surplus market for less than $25.00.

A terminal is important, but it is only one tool in the computer hobbyist's kit. Once you've written a program and gotten it into storage, you ought to use the cassette interface (HIT) described in September's column. With this tool, you have to "button in" the program bit-by-bit only once. After it is in storage, you can write it out to be taped and read in the next time you want it. Of course, if your only storage medium is RAM, you'll lose the memory contents when you turn the computer off. So, it's a good idea to copy the latest version of a program out to tape as a backup.

Also, if your only storage medium is RAM, you'll have to reenter the tape reading routine laboriously through the switches (or the terminal) each time the computer power is turned on. That is a good reason for having a small program, called a *bootstrap loader*, kept in read-only memory. This program makes it possible to read data from the tape and then execute that data. Such a read-in program is usually somewhat larger and more powerful, so it reads in several records (perhaps using the bootstrap program as a subroutine), which make up an even larger and more sophisticated program. The effect, then, is to use one group of records to read the next

```
                ; THIS IS THE POPULAR ELECTRONICS SUPER-
                ; SIMPLE MONITOR.  COMMANDS ARE:
                ;    D XXXX   (DUMP FROM XXXX)
                ;    L XXXX   (LOAD FROM XXXX)
                ;    G XXXX   (GO TO XXXX)
                STACK  EQU  03FFH        ;YOUR STACK ORIGIN
                CRLF   EQU  0106H        ;YOUR CR, LF ROUTINE
                WRCHR  EQU  0103H        ;YOUR OUTPUT ROUTINE
                RDCHR  EQU  0100H        ;YOUR READING ROUTINE
                ;
                ; MONITOR ENTRY POINT
                ;
0040: 31FF03    PEMON: LXI  SP,STACK     ;INITIALIZE
      CD0601           CALL CRLF         ;ISSUE CARRIAGE RETURN,
      3E3F             MVI  A,'?'        ;   LINE FEED AND ?
      CD0301           CALL WRCHR
      CD0001           CALL RDCHR        ;AWAIT COMMAND
      E67F             ANI  07FH         ;STRIP OFF PARITY BIT
      FE4C             CPI  'L'
      CA6300           JZ   LOAD         ;"LOAD" COMMAND
      FE44             CPI  'D'
      CA7400           JZ   DUMP         ;"DUMP" COMMAND
      FE47             CPI  'G'
      C24000           JNZ  PEMON        ;ERROR
                ; THIS THE THE "GO" COMMAND PROCESSOR.  WE
                ; NOW EXPECT A 16-BIT DESTINATION ADDRESS
                ; AND THEN TRANSFER TO IT
005F: CDB700           CALL RDNUM
      E9               PCHL              ;"GO"
                ; THIS IS THE "LOAD" COMMAND PROCESSOR.  THE
                ; USER IS EXPECTED TO TYPE IN A 16-BIT ADDRESS
                ; FOLLOWED BY DATA BYTES TO LOAD INTO SUCCESSIVE
                ; LOCATIONS.  ANY NON-HEX CHARACTER SEPARATES
                ; THE BYTES FROM ONE ANOTHER.  ANY BYTE THAT IS
                ; TERMINATED WITH A COLON WILL BE IGNORED.
0063: CDCD00    LOAD:  CALL RDNR2        ;GET USER'S ADDRESS
      CDB700           CALL RDNUM        ;GET A BYTE
      FE3A             CPI  ':'
      CA6600           JZ   LOAD+3       ;SKIP IF FOLLOWED BY ':'
      7D               MOV  A,L          ;GET LAST TWO HEX DIGITS
      02               STAX B            ;STORE THEM AWAY
      03               INX  B
      C36600           JMP  LOAD+3       ;(GET W/ CPU RESET)
                ; THIS IS THE "DUMP" COMMAND PROCESSOR.
                ; THE USER IS EXPECTED TO SUPPLY A 16-BIT
                ; STARTING ADDRESS.  WHENEVER THE LEAST-SIG-
                ; NIFICANT FOUR BITS OF THE ADDRESS OF THE
                ; NEXT BYTE ARE ZERO, THE CARRIAGE IS RETURNED
                ; AND THE ADDRESS IS PRINTED (FOLLOWED BY COLON).
0074: CDCD00    DUMP:  CALL RDNR2        ;GET STARTING ADDRESS
      79               MOV  A,C          ;CHECK TO SEE IF
      E60F             ANI  15           ;  NEW-LINE TIME.
      C2A200           JNZ  MORE         ;NO.  JUST PRINT
      CD0601           CALL CRLF         ;START NEW LINE
      78               MOV  A,B          ;GET MSB OF ADDRESS
      1F               RAR
      1F               RAR
      1F               RAR
      1F               RAR
      CDED00           CALL CVTAS
      78               MOV  A,B          ;GET 2ND DIGIT
      CDED00           CALL CVTAS
      79               MOV  A,C          ;GET 3RD DIGIT
      1F               RAR
      1F               RAR
      1F               RAR
      1F               RAR
      CDED00           CALL CVTAS
      79               MOV  A,C          ;DO LAST DIGIT
      CDED00           CALL CVTAS
      3E3A             MVI  A,':'        ;MARK ADDRESS EPECIALLY
      CD0301           CALL WRCHR
      3E20             MVI  A,' '
      CD0301           CALL WRCHR
00A2: 0A        MORE:  LDAX B            ;FETCH BYTE TO DUMP
      1F               RAR
      1F               RAR
      1F               RAR
      1F               RAR
      CDED00           CALL CVTAS        ;ISSUE MSB
      0A               LDAX B
      CDED00           CALL CVTAS
      3E20             MVI  A,' '
      CD0301           CALL WRCHR
      03               INX  B            ;ACQUIRE NEXT CELL
      C37700           JMP  DUMP+3
                ; THIS ROUTINE READS ONE OR MORE HEXADECIMAL
                ; DIGIT CHARACTERS ('0'..'9', 'A'..'F') IN
                ; AND ACCUMULATES A SIXTEEN-BIT NUMBER IN
                ; THE (H,L).  EACH NEW DIGIT IS SHIFTED INTO
                ; THE LEAST-SIGNIFICANT FOUR BITS OF THE (H,L).
                ; CONTROL IS RETURNED WHENEVER THE DEPENDENT
                ; ROUTINE ("RDHEX") SETS THE CARRY BIT TRUE.
00B7: 210000    RDNUM: LXI  H,0          ;START OFF VALUE AT 0
      CDD600           CALL RDHEX        ;GO GET A HEX DIGIT
      DAB700           JC   RDNUM        ;AWAIT HEX DIGIT
00C0: 29        RDNXT: DAD  H            ;SHIFT (H,L) LEFT 4
      29               DAD  H
      29               DAD  H
      29               DAD  H
      B5               ORA  L            ;PLACE NEW FOUR BITS IN
      6F               MOV  L,A
      CDD600           CALL RDHEX        ;GO GET NEXT DIGIT
      D8               RC                ; THE NUMBER'S FINISHED
      C3C000           JMP  RDNXT        ;GO PROCESS NEXT DIGIT
                ; THIS ROUTINE READS A NUMBER VIA "RDNUM" AND
                ; PLACES IT INTO THE (B,C) PAIR
00CD: CDB700    RDNR2: CALL RDNUM        ;GET THE VALUE IN (H,L)
      44               MOV  B,H          ; THEN MOVE IT
      4D               MOV  C,L
      C9               RET
                ; THIS ROUTINE READS IN AN ASCII CHARACTER,
                ; STRIPS OFF PARITY AND EXAMINES IT FOR
                ; MEMBERSHIP IN THE HEX-DIGIT SET.  IF IT IS
                ; A HEX-DIGIT THE A-REGISTER IS LEFT AT THE FOUR
                ; BIT VALUE APPROPRIATE AND THE CARRY IS CLEARED.
                ; ANY OTHER CHARACTER IS LEFT UNTOUCHED AND THE
                ; CARRY IS SET.
00D3: D630      RDDIG: SUI  '0'          ;TRANSLATE '0'..'9'
      C9               RET
00D6: CD0001    RDHEX: CALL RDCHR        ;***ENTRY POINT***
      E67F             ANI  07FH         ;REMOVE PARITY BIT
      FE30             CPI  '0'
      D8               RC                ;CHAR LESS THAN '0' (NOT HEX)
      FE3A             CPI  '9'+1
      DAD300           JC   RDDIG        ;IN RANGE 0..9
      FE41             CPI  'A'
      D8               RC                ;BETWEEN 9 AND A
      FE47             CPI  'F'+1
      3F               CMC
      D8               RC                ;NOT HEX CHARACTER
      D637             SUI  'A'-10       ;TRANSLATE 'A'..'F'
      C9               RET
                ; THIS ROUTINE CONVERTS A BINARY NUMBER IN THE
                ; LEAST-SIGNIFICANT BITS OF THE A-REGISTER TO
                ; AN ASCII CHARACTER REPRESENTING THE HEX VALUE
                ; AND THEN ISSUES IT AS OUTPUT.
00ED: E60F      CVTAS: ANI  15           ;ISOLATE FOUR BITS
      C630             ADI  '0'          ;SHIFT DIGITS INTO ASCII
      FE3A             CPI  '9'+1        ; SEE IF IT WAS 0..9
      DA0301           JC   WRCHR
      C607             ADI  'A'-'0'-10   ;CONVERT TO 'A'..'F'
      C30301           JMP  WRCHR
```

group in, thus "pulling" the program in by its own bootstraps.

Using a Monitor. The ability to preserve a program for later recall is important, but it doesn't solve two major nuisances: (1) you still have to key in the program bit-by-bit the first time; and (2) every time you make an error in the program, you have to key in the changes, some of which may be traumatic and complex. One of the best ways to solve this kind of inconvenience is to provide a small *monitor*. A monitor is just another computer program, but one that is designed to make computer use more convenient. This program reads characters from a terminal (or a separate keyboard), with these characters specifying the bit patterns to put into memory.

The simplest monitor has three basic *commands:* Load, Dump and Go. A command is a single letter typed at a time when the monitor is not otherwise engaged in some activity. Typical commands are single letters like "L" for Load, "D" for Dump and "G" for Go. When you type in "L" you are directing the monitor program to accept keyboard input and load it into memory; "D" means you want to display contents of memory on your terminal or display device; "G" is your means of transferring control out of the monitor into the program you have previously loaded.

Most programmers now use the hexadecimal number system for communicating with the machine, although there are "pockets" of users of octal. Hex and octal are, of course, just shorthand notations for binary code. Hex digits allow us to specify four bits with one symbol, octal allows three. The hexadecimal digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. (The letters A through F stand for decimal equivalent values 10 through 15, respectively.) The monitor, for utmost simplicity, uses only hexadecimal digits for the specification of addresses and data byte values.

Each of the command letters (L, D or G) is followed by an address that specifies where to start. For the Load command, that address is where the first byte of data from the keyboard will be stored; for Dump, it is the address from which data will begin being displayed; for Go, it is the address that is to be placed into the CPU's program counter. Whenever the Go command's address has been supplied, control is transferred to that location. Whenever the Dump command's address has been given, data displaying will begin. However, after the Load command's address, the monitor will expect more bytes (one after another) to be loaded into successive locations in memory.

A small monitor for the 8080 microprocessor that you can use as a model is shown opposite. Each command is stopped by resetting the CPU, thus returning control to the top of the monitor. Notice some error correction conventions that have been instituted to save you some time: numbers are assumed to consist of any number of hex digits but if the monitor wants an address, only the least-significant four digits are used. Likewise, for a data byte, only the least-significant two hex digits are preserved. This means that if you've made an error, just keep typing. Hex digits end with any character that is not a hex digit; most people find that the space character is the most convenient.

News Items. The extremely popular 8080, originally from Intel, is now being supplied by other semiconductor makers as well. The TI TMS8080 is identical to the original 8080, which Intel no longer makes. Intel's newer device, the 8080A, is functionally identical but has better current drive capacity. Intel has another part, the 8080A-1, that'll go faster so that AMD's 9080 (which is supposed to run 50% faster than the original Intel part) will have a competitor. So, if you are using the 8080, be sure to check the diagrams to see that your part matches the requirements.

The new MOS Technology 6501 is destined to become a popular CPU among hobbyists, if only because of its dramatically low price ($20 at press time). The device is modelled after Motorola's 6800, although with some major differences. All of the Motorola support parts like memory I/O chips can be used with the 6501, so you can get on board quickly. The Motorola software, however, cannot be executed on the 6501 without revision.

The 6501 is capable of operation at twice the Motorola part's speed; some parts may operate three times as fast. The introduction of this part is likely to start the real price war that has been brewing in the microprocessor business. Even with the new support chip for the 8008 that Intel has announced, it seems unlikely that it can compete with the 6501 for hobbyist use. ◈