

ALTAIR 8800 OPERATOR'S MANUAL

TABLE OF CONTENTS

PART ONE: <i>Introduction.....</i>	2
<i>(Logic, Electric Logic, Number Systems, The Binary System, The Octal System, Computer Programming, A Simple Program, Computer Languages)</i>	
PART TWO: <i>Organization of the Altair.....</i>	19
<i>(Central Processing Unit, Memory, Clock, Input/Output)</i>	
PART THREE: <i>Operation of the Altair.....</i>	28
<i>(Front Panel Switches and LED's, Loading a Sample Program, Using the Memory, Memory Addressing, Operating Hints)</i>	
PART FOUR: <i>Altair 8800 Instruction Set.....</i>	42
<i>(Command Instructions, Single Register Instructions, Register Pair Instructions, Rotate Accumulator Instructions)</i>	
APPENDIX: <i>Instruction List.....</i>	87

© MITS, Inc., 1975



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 mini-computers. After reading this manual, even a novice will be able to load a program into the *ALTAIR 8800*.

2 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 nineteenth 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.

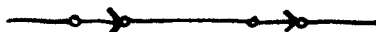



With this in mind, it is possible to devise electronic equivalents for the three basic logic statements: AND, OR and NOT. The AND statement is true if and only if either or all of its logic conditions are true. A NOT statement merely reverses the meaning of a logic statement so that a true statement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

5

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.

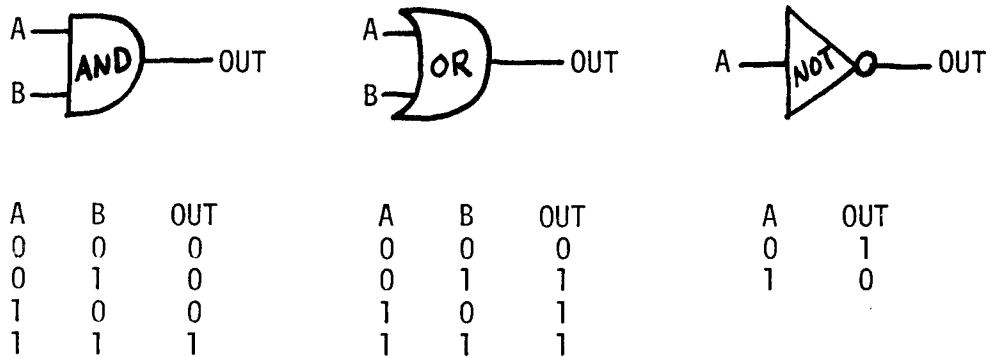


FIGURE 1-4. The Three Main Logic Symbols

6

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.

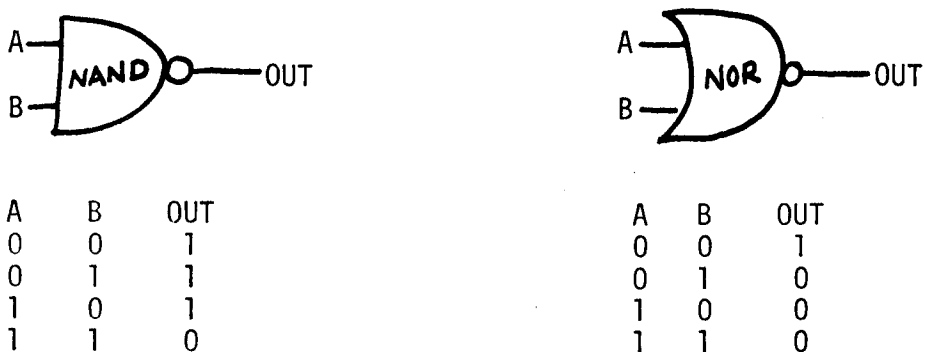


FIGURE 1-5. The NAND and NOR Circuits

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.

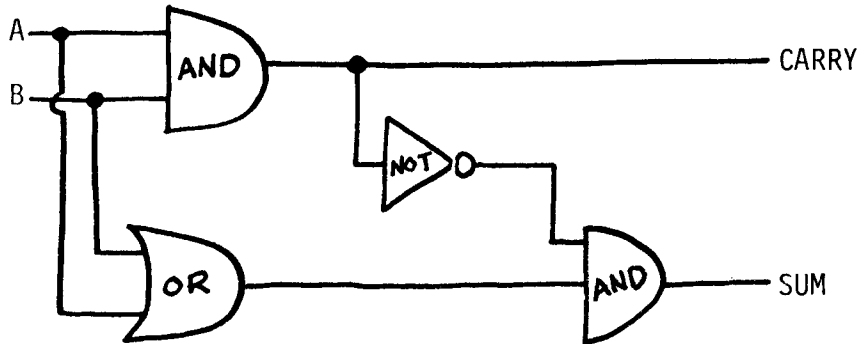


FIGURE 1-6. The EXCLUSIVE-OR Circuit

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

13

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

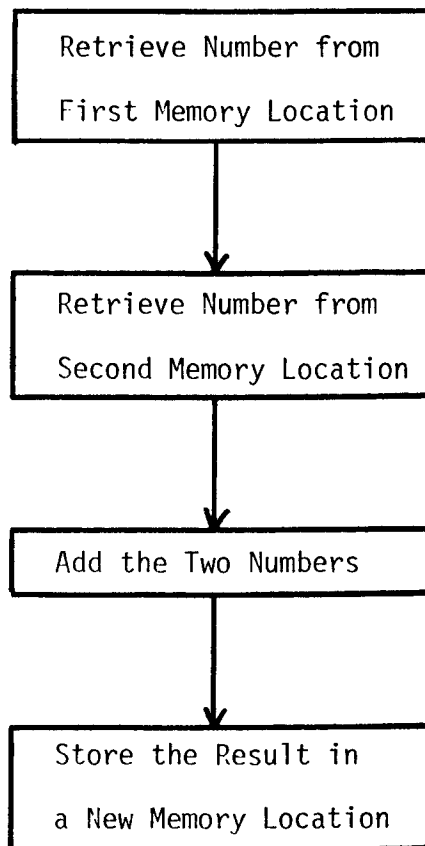
subroutine. The *ALTAIR 8800* instructions are described in detail in Part 4 of this manual.

G. A SIMPLE PROGRAM

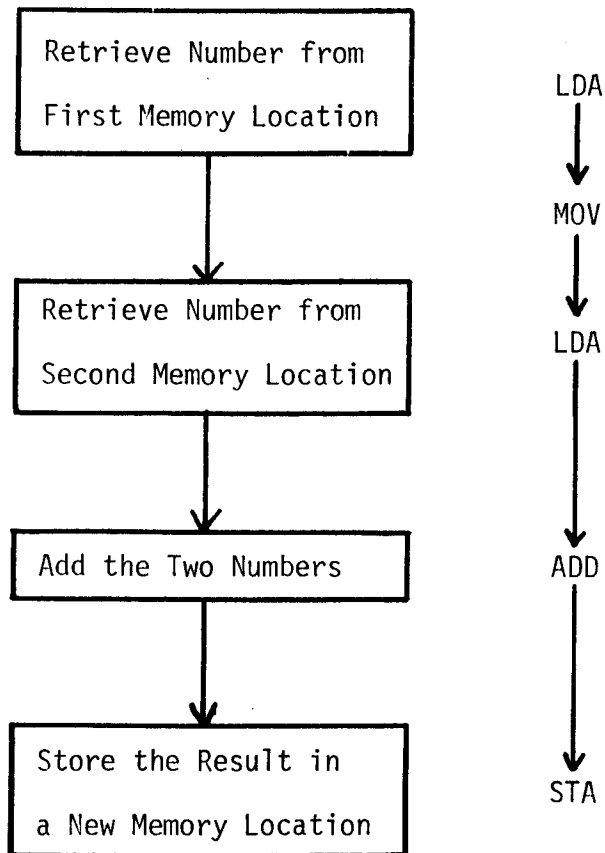
Assume you wish to use the *ALTAIR 8800* to add two numbers located at two different memory locations and store the result elsewhere in the memory. Of course this is a very simple problem, but it can be used to illustrate several basic programming techniques. Here are the steps used in generating a program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



16

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.

17

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 pro-

gramming 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.

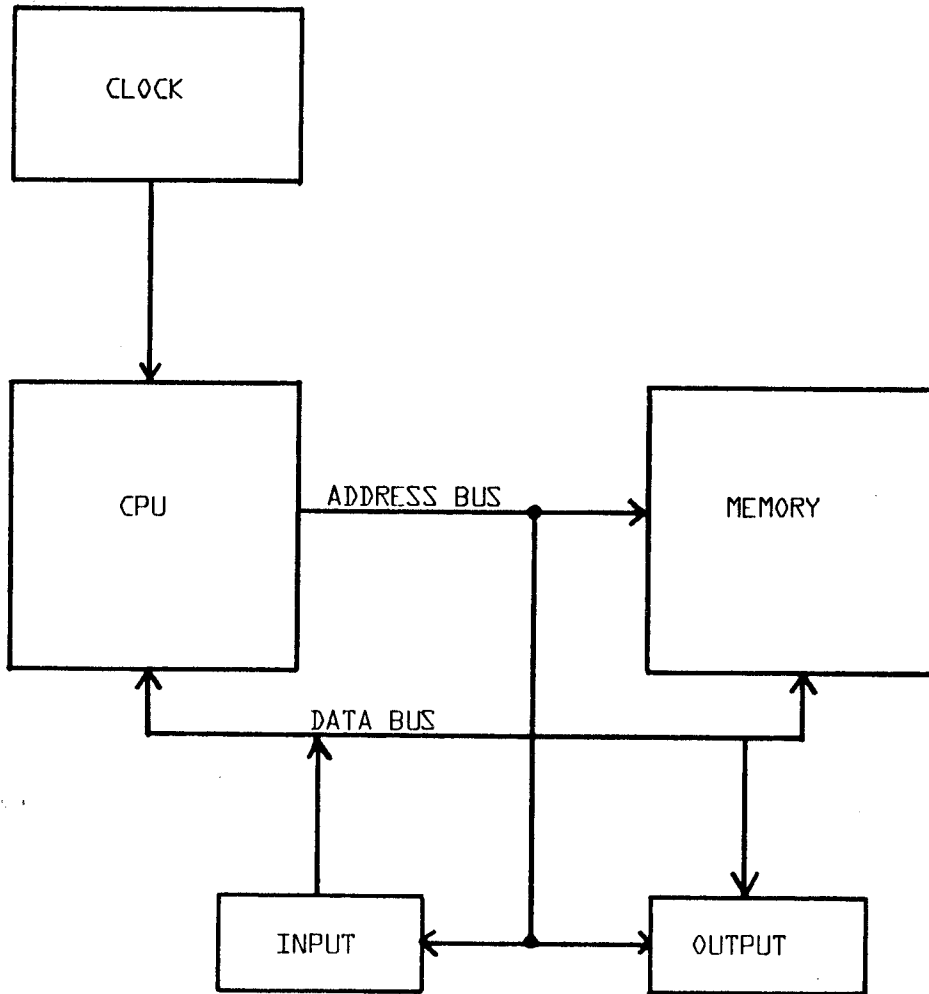
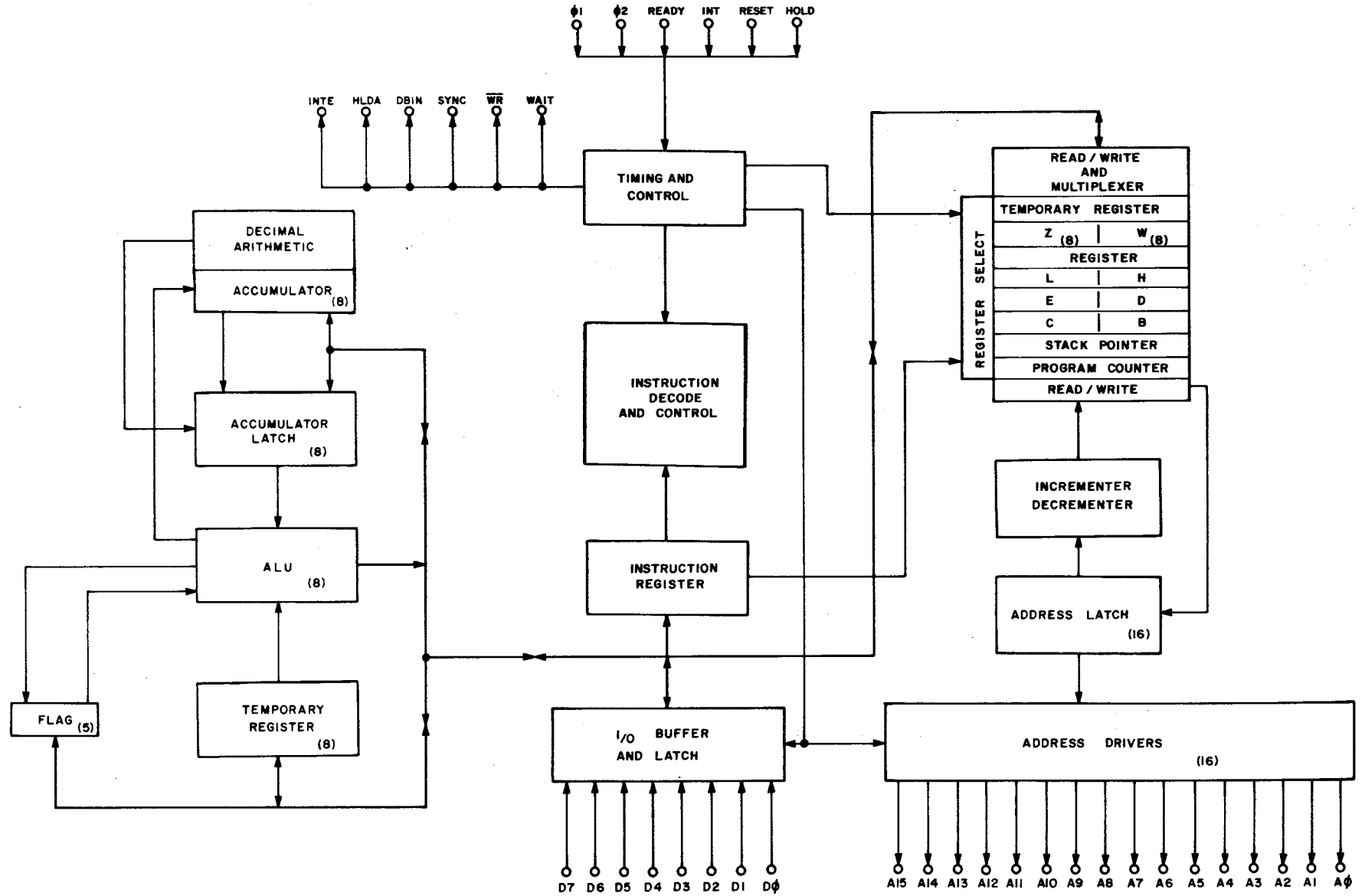


FIGURE 2-1

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.

21

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.

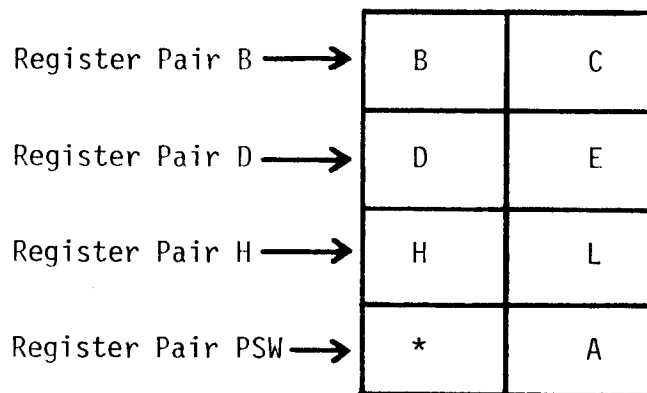
3. ARITHMETIC--The Arithmetic System performs both binary and decimal arithmetic. All arithmetic operations are performed by addition. Multiplication is implemented by repetitive addition. Subtraction and division are implemented by inverse addition.

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.

22

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.



*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 instruction (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 scratch-pad 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.

25

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).