ALTAIR 8800b
SECTION II
OPERATOR'S GUIDE

2-1. GENERAL

The Operators Guide contains information on the ALTAIR 8800b computer (8800b) front panel controls and indicators. It includes general switch operation exercises and a sample program which is intended to familiarize the operator with the various front panel operations. Provided in this section are portions of the Intel 8080 Microcomputer Systems Users Manual which contain Central Processor Unit, Interface and Software information. Additional programs available to the user are described in the ALTAIR Software Library. Update information is contained with your unit.

2-2. FRONT PANEL SWITCHES AND INDICATORS

The Front Panel switches permit the operator to perform various ALTAIR 8800b operations, and the indicators display address information, data information, and primary status control line information. Refer to Figure 2-1 for the location of the switches and indicators and Table 2-1 for an explanation of each.

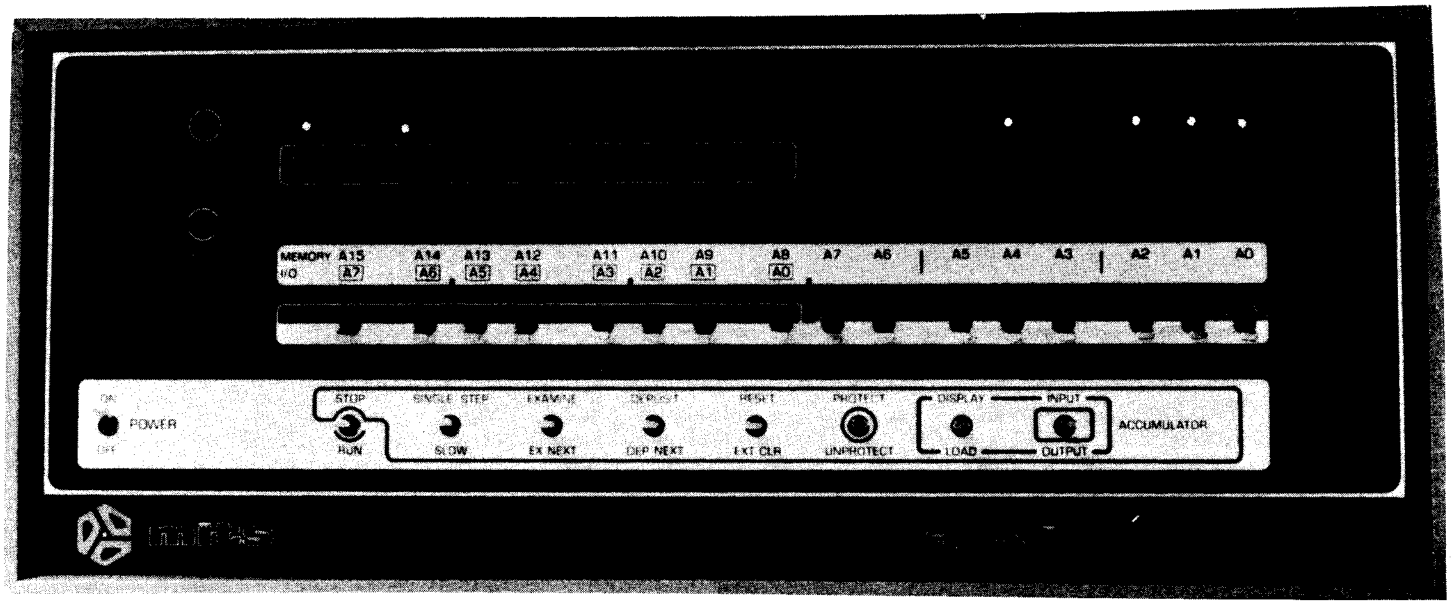

Figure 2-1. Altair 8800b Front Panel

Table 2-1. ALTAIR 8800b Switches and Indicators

Switch	Function or Indication
POWER ON/OFF	Applies power to the ALTAIR 8800b
STOP/RUN	The RUN position allows the CPU to process data and disables all functions on the front panel except reset. The STOP position conditions the CPU to a wait state and enables all functions on the front panel.
SINGLE STEP/ SLOW	The SINGLE STEP position allows execution of one machine cycle or one instruction cycle (depending upon the option selected). SLOW position allows execution of machine or instruction cycles at a rate of approximately 2 cycles per second. (Normal speed is approximately 500,000 machine cycles per second.) The CPU will execute the cycles as long as the SLOW position is maintained.
EXAMINE/ EX NEXT	The EXAMINE position allows the operator to examine the memory address selected on the A0-A15 MEMORY switches. The contents at that address are displayed on the DATA D0-D7 indicators. The EX NEXT position allows the operator to examine the next sequential memory address. Each time EX NEXT is actuated, the contents of the next sequential memory address are displayed.

Table 2-1. ALTAIR 8800b Switches and Indicators - Continued

Switch	Function or Indication
DEPOSIT/ DEP NEXT	The DEPOSIT position stores the contents of the lower address switches (A0-A7) into the memory address that is displayed on the MEMORY address A0-A15 indicators. The DEP NEXT position stores the contents of the lower address switches (A0-A7) into the next successive memory address.
RESET/ EXT CLR	The RESET position resets the program counter to zero and the interrupt enable flag in the CPU. The EXT CLR position produces an external clear signal on the system bus which generally clears an input/output.
PROTECT/ UNPROTECT*	The PROTECT position conditions the write protect circuits on the currently addressed memory board, preventing data in that block of memory from being changed. The front panel or the CPU cannot affect the memory when protected. UNPROTECT position allows the contents of memory to be changed.
ACCUMULATOR DISPLAY/LOAD	The DISPLAY position allows the contents of the CPU accumulator register to be displayed on the DATA D0-D7 indicators. The LOAD position allows the lower eight address switch (A0-A7) information to be stored in the CPU accumulator register.

*Protect switch only applies to memory boards with a protect circuit.

Table 2-1. ALTAIR 8800b Switches and Indicators - Continued

Switch or Indicator	Function or Indication
INPUT/ OUTPUT	The INPUT position allows an external device, selected on the I/O A0-A7 switches (upper eight address switches), to input data into the CPU accumulator. The OUTPUT position allows an external device, selected on the I/O A0-A7 switches, to receive data from the CPU accumulator register.
Address Switches A0-A15	These switches are used to select an address in memory or to enter data. The up position denotes a one bit and the down position denotes a zero bit.
SENSE switches A8-A15	The upper eight address switches (A8-A15) also function as SENSE switches. The data present on these switches is stored in the accumulator if an input from channel 377_8 (front panel) is executed.
MEMORY A0-A15	Display the memory address being examined or loaded with data.
PROTECT	Memory is protected.
INTE	Interrupts are enabled.
MEMR	The CPU is reading data from memory.
INP	An external device is inputting data to the CPU.
M1	The CPU is in machine cycle one of an instruction cycle.
OUT	The CPU is outputting data to an external device.

Table 2-1. ALTAIR 8800b Switches and Indicators - Continued

Indicator	Function or Indication
HLTA	The CPU is in a halt condition.
STACK	The address bus contains the address of the stack pointer.
WO	The CPU is writing out data to an external device or memory.
INT	The CPU has acknowledged an interrupt request.
DATA D0-D7	Data from memory, an external device, or the CPU
WAIT	The CPU is in a wait condition.
HLDA	The CPU has acknowledged a hold signal.

2-3. FRONT PANEL SWITCH APPLICATIONS

The following switch applications are intended to familiarize the operator with the ALTAIR 8800b front panel switches and indicators. Perform the operations in a sequential manner as shown in the following tables.

2-4. POWER ON SEQUENCE (Table 2-2)

The power on sequence resets the CPU program counter to the first memory address and places the CPU in a wait condition at the beginning of an instruction cycle.

Table 2-2. Power On Sequence

Step	Function	Indication
1	Position the POWER ON/OFF switch to ON.	MEMR, M1, and WAIT indicators are on. Some DATA D0-D7 indicators may also be on. All other indicators are off.

2-5. RUN OPERATION (Table 2-3)

The run operation releases the CPU from a wait condition, and allows it to execute a program. When the run operation is enabled, all other front panel switches are inactive except the RESET switch.

Table 2-3. Run Operation

Step	Function	Indication
1	Momentarily position the STOP/RUN switch to RUN.	WAIT indicator is off (or may be dimly lit). The machine can now execute a program.

2-6. STOP OPERATION (Table 2-4)

The stop operation places the CPU in a wait condition and allows the operator to use the switches on the 8800b front panel.

Table 2-4. Stop Operation

Step	Function	Indication
1	Position the STOP/RUN switch to STOP.	WAIT, MEMR, and M1 indicators are on. The operator now has control of the front panel.

2-7. EXAMINE MEMORY OPERATION (Table 2-5)

This procedure allows the operator to select a memory address and examine its contents.

Table 2-5. Examine Memory Operation

Step	Function	Indication
1	Position the address switches A0-A15 down.	
2	Position the EXAMINE/ EX NEXT switch to EXAMINE.	A0 through A15 indicators are off, indicating memory address location 000_8 is being examined. DATA D0 through D7 indicators are displaying the contents of location 000_8.
3	Position address switches A1 and A2 up.	
4	Position the EXAMINE/ EX NEXT switch to EXAMINE.	A1 and A2 indicators are on, indicating memory address 006_8 is being examined. DATA D0 through D7 indicators are displaying the contents of location 006_8.

2-8. ALTERING MEMORY CONTENTS (Table 2-6)

This procedure allows the operator to select a memory address and change its contents.

Table 2-6. Altering Memory Contents

Step	Function	Indication
1	Position address switch A5 up and the remaining switches down.	
2	Position the EXAMINE/ EX NEXT switch to EXAMINE	A5 indicator is on, indicating memory address 040_8. DATA D0 through D7 indicators are displaying the contents of location 040_8.
3	Position the A0 through A7 address switches up.	
4	Position the DEPOSIT/DEP NEXT to DEPOSIT	DATA D0 through D7 indicators are on, indicating the new data that has been placed in address location 040_8.

2-9. EXAMINE NEXT MEMORY LOCATION (Table 2-7)

This procedure allows the operator to examine the next sequential memory location, as determined by the address switches.

Table 2-7. Examine Next Memory Location

Step	Function	Indication
1	Position address switches A0 and A5 up, and the remaining switches down.	
2	Position the EXAMINE/EX NEXT switch to EXAMINE	A0 and A5 indicators are on, indicating memory address 041_8.

Table 2-7. Examine Next Memory Location - Continued

Step	Function	Indication
3	Position address switches A1, A4, and A6 up, and the remaining switches down.	
4	Position the DEPOSIT/DEP NEXT switch to DEPOSIT	DATA D1, D4, and D6 indicators are on.
5	Position address switch A5 up, and the remaining switches down.	
6	Position the EXAMINE/EX NEXT switch to EXAMINE	A5 indicator is on, indicating memory address 040_8. DATA D0 through D7 indicators are on.
7	Position the EXAMINE/EX NEXT switch to EX NEXT	A5 and A0 indicators are on, indicating address 041_8. DATA D1, D4, and D6 indicators are on.

2-10. ALTER NEXT MEMORY LOCATION CONTENTS (Table 2-8)

This procedure allows the operator to select a memory address and change the contents of the address that immediately follows.

Table 2-8. Altering Next Memory Contents

Step	Function	Indication
1	Position address switches A0 and A5 up, and the remaining switches down.	
2	Position the EXAMINE/EX NEXT switch to EXAMINE	A0 and A5 indicators are on.
3	Position address switches A0 through A7 up	

Table 2-8. Altering Next Memory Contents - Continued

Step	Function	Indication
4	Position the DEPOSIT/ DEP NEXT switch to DEP NEXT	A1 and A5 indicators are on, indicating 042_8. DATA D0 through D7 are on, displaying the new contents of location 042_8.
5	To verify, position address switches A5 and A1 up, and the remaining switches down.	
6	Position the EXAMINE/ EX NEXT switch to EXAMINE	A1 and A5 indicators are on, and DATA D0 through D7 are on.

2-11. LOADING AND DISPLAYING ACCUMULATOR DATA (Table 2-9)

This procedure allows the operator to load new data into the accumulator or check the contents of the accumulator.

Table 2-9. Loading and Displaying Accumulator Data

Step	Function	Indication
1	Position address switches A0, A1, and A2 up, and the remaining switches down.	
2	Position the ACCUMULATOR DISPLAY/LOAD switch to LOAD	
3	Position the ACCUMULATOR DISPLAY/LOAD switch to DISPLAY	DATA D0, D1, and D2 indicators are on while "DISPLAY" is activated.

2-12. LOADING A SAMPLE PROGRAM

The sample program is designed to retrieve two numbers from memory, add them together, and store the result in memory. The exact program in mnemonic form can be written as follows:

0. LDA
1. MOV B,A
2. LDA
3. ADD B
4. STA
5. JMP

The mnemonics for all 78 8800b instructions are explained in detail in the excerpt from the Intel 8080 Microcomputer System User's Manual contained in this section. However, the instructions used in this program are explained as follows:

0. LDA--Load the accumulator with the contents of a specified memory address.
1. MOV B,A--Move the contents of the accumulator into register B.
2. LDA--Same as 0.
3. ADD B--Add the contents of register B to the contents of the accumulator and store the result in the accumulator.
4. STA--Store the contents of the accumulator in a specified memory address.
5. JMP--Jump to the first step in the program.

Step 5, the JMP instruction (followed by the memory address of the first instruction), causes the CPU to "jump" back to the beginning of the sample program and execute the program repeatedly until the CPU is halted. Without a JMP instruction the CPU would continue to run randomly through memory.

2-13. LOADING THE PROGRAM

To load the program into the 8800b, first determine the memory addresses for the two numbers to be added and where the result is to be stored. Store the program instructions in successive memory addresses, beginning at the first memory address, 000_8. In this example the first number to be added will be located at memory address 200_8 (10 000 000), the second at memory address 201_8 (10 000 001), and the sum will be stored in memory address 202_8 (10 000 010). Now that the memory addresses have been specified, the program can be converted into its machine bit patterns (Table 2-10).

Table 2-10. Machine Language Bit Patterns

MNEMONIC	BIT PATTERN	EXPLANATION
LDA 200	00 111 010	Load Accumulator in the CPU with con-
	10 000 000	tents of Memory address 200_8 (2 bytes
	00 000 000	required for memory addresses)
MOV B,A	01 000 111	Move Accumulator data to Register B
LDA 201	00 111 010	Load Accumulator with the contents
	10 000 001	of Memory address 201_8
	00 000 000	
ADD B	10 000 000	Add Register B to Accumulator
STA 202	00 110 010	Store the Accumulator contents
	10 000 010	in Memory address 202_8
	00 000 000	
JMP 000	11 000 011	Jump to Memory location 0.
	00 000 000	
	00 000 000	

The octal equivalent of each bit pattern is also frequently included in the program listing. It is easy to load octal numbers on the front panel switches, since it is only necessary to know the binary equivalents for the numbers 0-7. The resulting program, including octal equivalents, may be written as shown in Table 2-11:

Table 2-11. Addition Program

MEMORY ADDRESS	MNEMONIC	BIT PATTERN	OCTAL EQUIVALENT
000	LDA 200	00 111 010	0 7 2
001	(address)	10 000 000	2 0 0
002	(address)	00 000 000	0 0 0
003	MOV B,A	01 000 111	1 0 7
004	LDA 201	00 111 010	0 7 2
005	(address)	10 000 001	2 0 1
006	(address)	00 000 000	0 0 0
007	ADD B	10 000 000	2 0 0
010	STA 202	00 011 010	0 6 2
011	(address)	10 000 010	2 0 2
012	(address)	00 000 000	0 0 0
013	JMP 000	11 000 011	3 0 3
014	(address)	00 000 000	0 0 0
015	(address)	00 000 000	0 0 0

Using the front panel switches, the program may now be entered into the computer. To begin loading the program at the first memory address 000, position the RESET/CLR switch to RESET. The data to be stored in address 000 is entered on address switches A0 through A7. After the address switches are set, position the DEPOSIT/DEP NEXT switch to DEPOSIT to enter the A0-A7 bit pattern into memory address 000. Enter the second byte of data on the address switches and position the DEPOSIT/DEP NEXT switch to DEP NEXT. The bit pattern will be loaded automatically into the next sequential memory address (001). Continue loading the data into memory for the remainder of the program. The complete program loading procedure is shown in Table 2-12:

Table 2-12. Addition Program Loading

MEMORY ADDRESS	ADDRESS SWITCHES DATA 0-7	CONTROL SWITCH
		RESET
000	00 111 010	DEPOSIT
001	10 000 000	DEPOSIT NEXT
002	00 000 000	DEPOSIT NEXT
003	01 000 111	DEPOSIT NEXT
004	00 111 010	DEPOSIT NEXT
005	10 000 001	DEPOSIT NEXT
006	00 000 000	DEPOSIT NEXT
007	10 000 000	DEPOSIT NEXT
010	00 110 010	DEPOSIT NEXT
011	10 000 010	DEPOSIT NEXT
012	00 000 000	DEPOSIT NEXT
013	11 000 011	DEPOSIT NEXT
014	00 000 000	DEPOSIT NEXT
015	00 000 000	DEPOSIT NEXT

The program is now ready to be run, but first it is necessary to store data at each of the two memory addresses (200_8 and 201_8) to be added together. To load the first address, set address switches A0-A7 to 10 000 000_2 and position the EXAMINE/EX NEXT switch to EXAMINE. Now load any desired number into this address by using address switches A0-A7. When the number has been loaded onto the switches, position the DEPOSIT/DEP NEXT to DEPOSIT to load the data into memory. To load the next address, enter a second number on the address switches A0-A7 and position the DEPOSIT/DEP NEXT switch to DEP NEXT. Since sequential memory addresses were selected, the number will be loaded automatically into the proper address (10 000 001_2). Once the program has been loaded and the two numbers have been stored in memory locations 200_8 and 201_8, the program can be run. Return to address 000 by positioning all A0-A7 address switches down and positioning the EXAMINE/EX NEXT switch to EXAMINE. Then position the STOP/RUN switch to RUN. Wait a moment and position the STOP/RUN switch to STOP. Check the answer of your addition program by selecting memory location 202_8 on the address switches and positioning the EXAMINE/EX NEXT switch to EXAMINE. The result is displayed on the DATA D0-D7 indicators.

2-14. INTEL 8080 MICROCOMPUTER SYSTEMS USER'S INFORMATION

Pages 2-16 through 2-65 are excerpts from the Intel 8080 Microcomputer Systems User's Manual, reprinted by permission of Intel Corporation, Copyright 1975. Included is detailed Central Processor Unit, Interface and Software information pertaining to the 8080 Microcomputer System.

CHAPTER 1 THE FUNCTIONS OF A COMPUTER

This chapter introduces certain basic computer concepts. It provides background information and definitions which will be useful in later chapters of this manual. Those already familiar with computers may skip this material, at their option.

A TYPICAL COMPUTER SYSTEM

A typical digital computer consists of:

a) A central processor unit (CPU)
b) A memory
c) Input/output (I/O) ports

The memory serves as a place to store **Instructions**, the coded pieces of information that direct the activities of the CPU, and **Data**, the coded pieces of information that are processed by the CPU. A group of logically related instructions stored in memory is referred to as a **Program**. The CPU "reads" each instruction from memory in a logically determined sequence, and uses it to initiate processing actions. If the program sequence is coherent and logical, processing the program will produce intelligible and useful results.

The memory is also used to store the data to be manipulated, as well as the instructions that direct that manipulation. The program must be organized such that the CPU does not read a non-instruction word when it expects to see an instruction. The CPU can rapidly access any data stored in memory; but often the memory is not large enough to store the entire data bank required for a particular application. The problem can be resolved by providing the computer with one or more **Input Ports**. The CPU can address these ports and input the data contained there. The addition of input ports enables the computer to receive information from external equipment (such as a paper tape reader or floppy disk) at high rates of speed and in large volumes.

A computer also requires one or more **Output Ports** that permit the CPU to communicate the result of its processing to the outside world. The output may go to a display, for use by a human operator, to a peripheral device that produces "hard-copy," such as a line-printer, to a peripheral storage device, such as a floppy disk unit, or the output may constitute process control signals that direct the operations of another system, such as an automated assembly line. Like input ports, output ports are addressable. The input and output ports together permit the processor to communicate with the outside world.

The CPU unifies the system. It controls the functions performed by the other components. The CPU must be able to fetch instructions from memory, decode their binary contents and execute them. It must also be able to reference memory and I/O ports as necessary in the execution of instructions. In addition, the CPU should be able to recognize and respond to certain external control signals, such as INTERRUPT and WAIT requests. The functional units within a CPU that enable it to perform these functions are described below.

THE ARCHITECTURE OF A CPU

A typical central processor unit (CPU) consists of the following interconnected functional units:

- Registers
- Arithmetic/Logic Unit (ALU)
- Control Circuitry

Registers are temporary storage units within the CPU. Some registers, such as the program counter and instruction register, have dedicated uses. Other registers, such as the accumulator, are for more general purpose use.

Accumulator:

The accumulator usually stores one of the operands to be manipulated by the ALU. A typical instruction might direct the ALU to add the contents of some other register to the contents of the accumulator and store the result in the accumulator itself. In general, the accumulator is both a source (operand) and a destination (result) register.

Often a CPU will include a number of additional general purpose registers that can be used to store operands or intermediate data. The availability of general purpose registers eliminates the need to "shuffle" intermediate results back and forth between memory and the accumulator, thus improving processing speed and efficiency.

Program Counter (Jumps, Subroutines and the Stack):

The instructions that make up a program are stored in the system's memory. The central processor references the contents of memory, in order to determine what action is appropriate. This means that the processor must know which location contains the next instruction.

Each of the locations in memory is numbered, to distinguish it from all other locations in memory. The number which identifies a memory location is called its **Address**.

The processor maintains a counter which contains the address of the next program instruction. This register is called the **Program Counter**. The processor updates the program counter by adding "1" to the counter each time it fetches an instruction, so that the program counter is always current (pointing to the next instruction).

The programmer therefore stores his instructions in numerically adjacent addresses, so that the lower addresses contain the first instructions to be executed and the higher addresses contain later instructions. The only time the programmer may violate this sequential rule is when an instruction in one section of memory is a **Jump** instruction to another section of memory.

A jump instruction contains the address of the instruction which is to follow it. The next instruction may be stored in any memory location, as long as the programmed jump specifies the correct address. During the execution of a jump instruction, the processor replaces the contents of its program counter with the address embodied in the Jump. Thus, the logical continuity of the program is maintained.

A special kind of program jump occurs when the stored program "**Calls**" a subroutine. In this kind of jump, the processor is required to "remember" the contents of the program counter at the time that the jump occurs. This enables the processor to resume execution of the main program when it is finished with the last instruction of the subroutine.

A **Subroutine** is a program within a program. Usually it is a general-purpose set of instructions that must be executed repeatedly in the course of a main program. Routines which calculate the square, the sine, or the logarithm of a program variable are good examples of functions often written as subroutines. Other examples might be programs designed for inputting or outputting data to a particular peripheral device.

The processor has a special way of handling subroutines, in order to insure an orderly return to the main program. When the processor receives a Call instruction, it increments the Program Counter and stores the counter's contents in a reserved memory area known as the **Stack**. The Stack thus saves the address of the instruction to be executed after the subroutine is completed. Then the processor loads the address specified in the Call into its Program Counter. The next instruction fetched will therefore be the first step of the subroutine.

The last instruction in any subroutine is a **Return**. Such an instruction need specify no address. When the processor fetches a Return instruction, it simply replaces the current contents of the Program Counter with the address on the top of the stack. This causes the processor to resume execution of the calling program at the point immediately following the original Call Instruction.

Subroutines are often **Nested**; that is, one subroutine will sometimes call a second subroutine. The second may call a third, and so on. This is perfectly acceptable, as long as the processor has enough capacity to store the necessary return addresses, and the logical provision for doing so. In other words, the maximum depth of nesting is determined by the depth of the stack itself. If the stack has space for storing three return addresses, then three levels of subroutines may be accommodated.

Processors have different ways of maintaining stacks. Some have facilities for the storage of return addresses built into the processor itself. Other processors use a reserved area of external memory as the stack and simply maintain a **Pointer** register which contains the address of the most recent stack entry. The external stack allows virtually unlimited subroutine nesting. In addition, if the processor provides instructions that cause the contents of the accumulator and other general purpose registers to be "pushed" onto the stack or "popped" off the stack via the address stored in the stack pointer, multi-level interrupt processing (described later in this chapter) is possible. The status of the processor (i.e., the contents of all the registers) can be saved in the stack when an interrupt is accepted and then restored after the interrupt has been serviced. This ability to save the processor's status at any given time is possible even if an interrupt service routine, itself, is interrupted.

Instruction Register and Decoder:

Every computer has a **Word Length** that is characteristic of that machine. A computer's word length is usually determined by the size of its internal storage elements and interconnecting paths (referred to as **Busses**); for example, a computer whose registers and busses can store and transfer 8 bits of information has a characteristic word length of 8-bits and is referred to as an 8-bit parallel processor. An eight-bit parallel processor generally finds it most efficient to deal with eight-bit binary fields, and the memory associated with such a processor is therefore organized to store eight bits in each addressable memory location. Data and instructions are stored in memory as eight-bit binary numbers, or as numbers that are integral multiples of eight bits: 16 bits, 24 bits, and so on. This characteristic eight-bit field is often referred to as a **Byte**.

Each operation that the processor can perform is identified by a unique byte of data known as an **Instruction**

Code or **Operation Code.** An eight-bit word used as an instruction code can distinguish between 256 alternative actions, more than adequate for most processors.

The processor fetches an instruction in two distinct operations. First, the processor transmits the address in its Program Counter to the memory. Then the memory returns the addressed byte to the processor. The CPU stores this instruction byte in a register known as the **Instruction Register**, and uses it to direct activities during the remainder of the instruction execution.

The mechanism by which the processor translates an instruction code into specific processing actions requires more elaboration than we can here afford. The concept, however, should be intuitively clear to any logic designer. The eight bits stored in the instruction register can be decoded and used to selectively activate one of a number of output lines, in this case up to 256 lines. Each line represents a set of activities associated with execution of a particular instruction code. The enabled line can be combined with selected timing pulses, to develop electrical signals that can then be used to initiate specific actions. This translation of code into action is performed by the **Instruction Decoder** and by the associated control circuitry.

An eight-bit instruction code is often sufficient to specify a particular processing action. There are times, however, when execution of the instruction requires more information than eight bits can convey.

One example of this is when the instruction references a memory location. The basic instruction code identifies the operation to be performed, but cannot specify the object address as well. In a case like this, a two- or three-byte instruction must be used. Successive instruction bytes are stored in sequentially adjacent memory locations, and the processor performs two or three fetches in succession to obtain the full instruction. The first byte retrieved from memory is placed in the processor's instruction register, and subsequent bytes are placed in temporary storage; the processor then proceeds with the execution phase. Such an instruction is referred to as **Variable Length.**

Address Register(s):

A CPU may use a register or register-pair to hold the address of a memory location that is to be accessed for data. If the address register is **Programmable**, (i.e., if there are instructions that allow the programmer to alter the contents of the register) the program can "build" an address in the address register prior to executing a **Memory Reference** instruction (i.e., an instruction that reads data from memory, writes data to memory or operates on data stored in memory).

Arithmetic/Logic Unit (ALU):

All processors contain an arithmetic/logic unit, which is often referred to simply as the **ALU.** The ALU, as its name implies, is that portion of the CPU hardware which performs the arithmetic and logical operations on the binary data.

The ALU must contain an **Adder** which is capable of combining the contents of two registers in accordance with the logic of binary arithmetic. This provision permits the processor to perform arithmetic manipulations on the data it obtains from memory and from its other inputs.

Using only the basic adder a capable programmer can write routines which will subtract, multiply and divide, giving the machine complete arithmetic capabilities. In practice, however, most ALUs provide other built-in functions, including hardware subtraction, boolean logic operations, and shift capabilities.

The ALU contains **Flag Bits** which specify certain conditions that arise in the course of arithmetic and logical manipulations. Flags typically include **Carry, Zero, Sign,** and **Parity.** It is possible to program jumps which are conditionally dependent on the status of one or more flags. Thus, for example, the program may be designed to jump to a special routine if the carry bit is set following an addition instruction.

Control Circuitry:

The control circuitry is the primary functional unit within a CPU. Using clock inputs, the control circuitry maintains the proper sequence of events required for any processing task. After an instruction is fetched and decoded, the control circuitry issues the appropriate signals (to units both internal and external to the CPU) for initiating the proper processing action. Often the control circuitry will be capable of responding to external signals, such as an interrupt or wait request. An **Interrupt** request will cause the control circuitry to temporarily interrupt main program execution, jump to a special routine to service the interrupting device, then automatically return to the main program. A **Wait** request is often issued by a memory or I/O element that operates slower than the CPU. The control circuitry will idle the CPU until the memory or I/O port is ready with the data.

COMPUTER OPERATIONS

There are certain operations that are basic to almost any computer. A sound understanding of these basic operations is a necessary prerequisite to examining the specific operations of a particular computer.

Timing:

The activities of the central processor are cyclical. The processor fetches an instruction, performs the operations required, fetches the next instruction, and so on. This orderly sequence of events requires precise timing, and the CPU therefore requires a free running oscillator clock which furnishes the reference for all processor actions. The combined fetch and execution of a single instruction is referred to as an **Instruction Cycle.** The portion of a cycle identified with a clearly defined activity is called a **State**. And the interval between pulses of the timing oscillator is referred to as a **Clock Period**. As a general rule, one or more clock periods are necessary for the completion of a state, and there are several states in a cycle.

Instruction Fetch:

The first state(s) of any instruction cycle will be dedicated to fetching the next instruction. The CPU issues a read signal and the contents of the program counter are sent to memory, which responds by returning the next instruction word. The first byte of the instruction is placed in the instruction register. If the instruction consists of more than one byte, additional states are required to fetch each byte of the instruction. When the entire instruction is present in the CPU, the program counter is incremented (in preparation for the next instruction fetch) and the instruction is decoded. The operation specified in the instruction will be executed in the remaining states of the instruction cycle. The instruction may call for a memory read or write, an input or output and/or an internal CPU operation, such as a register-to-register transfer or an add-registers operation.

Memory Read:

An instruction **fetch** is merely a special memory read operation that brings the instruction to the CPU's instruction register. The instruction fetched may then call for data to be read from memory into the CPU. The CPU again issues a read signal and sends the proper memory address; memory responds by returning the requested word. The data received is placed in the accumulator or one of the other general purpose registers (not the instruction register).

Memory Write:

A memory write operation is similar to a read except for the direction of data flow. The CPU issues a write signal, sends the proper memory address, then sends the data word to be written into the addressed memory location.

Wait (memory synchronization):

As previously stated, the activities of the processor are timed by a master clock oscillator. The clock period determines the timing of all processing activity.

The speed of the processing cycle, however, is limited by the memory's **Access Time**. Once the processor has sent a read address to memory, it cannot proceed until the memory has had time to respond. Most memories are capable of responding much faster than the processing cycle requires. A few, however, cannot supply the addressed byte within the minimum time established by the processor's clock.

Therefore a processor should contain a synchronization provision, which permits the memory to request a **Wait state**. When the memory receives a read or write enable signal, it places a request signal on the processor's READY line, causing the CPU to idle temporarily. After the memory has had time to respond, it frees the processor's READY line, and the instruction cycle proceeds.

Input/Output:

Input and Output operations are similar to memory read and write operations with the exception that a peripheral I/O device is addressed instead of a memory location. The CPU issues the appropriate input or output control signal, sends the proper device address and either receives the data being input or sends the data to be output.

Data can be input/output in either parallel or serial form. All data within a digital computer is represented in binary coded form. A binary data word consists of a group of bits; each bit is either a one or a zero. **Parallel** I/O consists of transferring all bits in the word at the same time, one bit per line. **Serial** I/O consists of transferring one bit at a time on a single line. Naturally serial I/O is much slower, but it requires considerably less hardware than does parallel I/O.

Interrupts:

Interrupt provisions are included on many central processors, as a means of improving the processor's efficiency. Consider the case of a computer that is processing a large volume of data, portions of which are to be output to a printer. The CPU can output a byte of data within a single machine cycle but it may take the printer the equivalent of many machine cycles to actually print the character specified by the data byte. The CPU could then remain idle waiting until the printer can accept the next data byte. If an interrupt capability is implemented on the computer, the CPU can output a data byte then return to data processing. When the printer is ready to accept the next data byte, it can request an interrupt. When the CPU acknowledges the interrupt, it suspends main program execution and automatically branches to a routine that will output the next data byte. After the byte is output, the CPU continues with main program execution. Note that this is, in principle, quite similar to a subroutine call, except that the jump is initiated externally rather than by the program.

More complex interrupt structures are possible, in which several interrupting devices share the same processor but have different priority levels. Interruptive processing is an important feature that enables maximum untilization of a processor's capacity for high system throughput.

Hold:

Another important feature that improves the throughput of a processor is the **Hold**. The hold provision enables **Direct Memory Access** (DMA) operations.

In ordinary input and output operations, the processor itself supervises the entire data transfer. Information to be placed in memory is transferred from the input device to the processor, and then from the processor to the designated memory location. In similar fashion, information that goes from memory to output devices goes by way of the processor.

Some peripheral devices, however, are capable of transferring information to and from memory much faster than the processor itself can accomplish the transfer. If any appreciable quantity of data must be transferred to or from such a device, then **system throughput** will be increased by having the device accomplish the transfer directly. The processor must temporarily suspend its operation during such a transfer, to prevent conflicts that would arise if processor and peripheral device attempted to access memory simultaneously. It is for this reason that a **hold** provision is included on some processors.