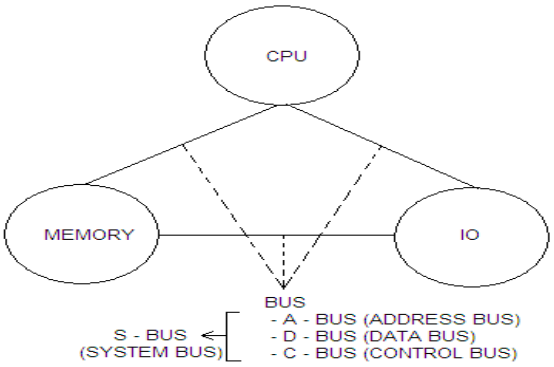
Input – Output Architecture→

A computer system organization is defined as a collection of component connected together in a logical sense, i.e. the logical structure of a computer system is termed computer organization.

The collection of physical component such as CPU, Memory, I/O arranged constitutes computer architecture. Thus the physical definition of the system is architecture.

Organization = Logical

Architecture = Physical



"TRIAD"

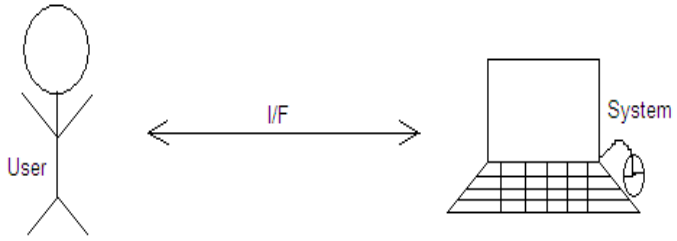
A computer system can be defined as a TRIAD comprising of three fundamental components:

(1) The CPU (2) The Memory & (3) The I/O

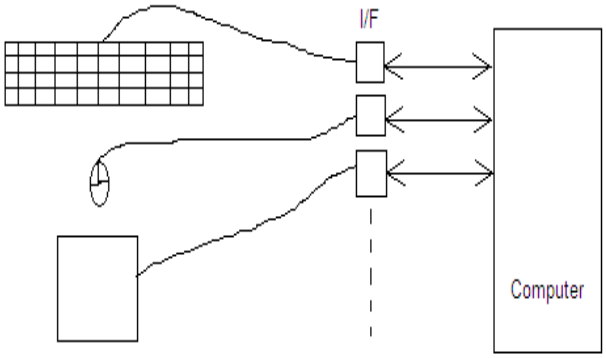
These three devices communicates with each other with the help of a set of wires called System Bus, where the System Bus comprises of three buses as –

1. The Address Bus
2. The Data Bus and
3. The Control Bus.

A computer system requires an interface with the user to execute a program. The interaction is accomplished with the help of I/O devices. (The Input – Output subsystem of a computer, referred to as I/O, provides an efficient mode of communication between the central system and the outside environment).



The I/O devices are connected to an entity called an interface (I/F) to have an interaction with the system.



Organization and Architecture→

In describing computer system, a distinction is often made between computer architecture and computer organization. Computer architecture refers to those attributes of a system visible to a programmer or put another way, those attributes that have a direct impact on the logical execution of a program. Computer organization refers to the operational units and their interconnections that realize the architectural specifications. Example of architectural attributes include the instruction set, the number of bits used to represent various data types (e.g. numbers, characters), I/O mechanisms, and techniques for addressing memories. Organizational attributes include those hardware details transparent to the programmer, such as control signals, interface between the computer and peripherals and the memory technology used.

Register transfer language

A digital system is an interconnection of digital hardware modules combined together to perform a specific processing task. The modules are constructed using registers ,decoders, arithmetic elements and control logic. These various modules are interconnected with common data and control paths to form a digital computer system.

A digital module can be best defined as the register it contains and the operations that are performed on the data stored in them. The operations executed on data stored in registers are called micro operations. A micro-operation is defined as an elementary operation carried out on one clock pulse, over the information stored in one or more registers.eg. shift, count, clear load.

The organization of a digital computer can be best defined by:

1)The set of registers it contains and their function

2)sequence of micro-operations performed on the data stored in the registers

3)control functions that initiate the sequence of micro-operations.

The symbolic notations used to describe the micro-operation transfers among registers is called a register transfer language. As a programming language is a procedure for writing symbols to specify a given computational process similarly a register transfer language ia a system for expressing the micro-operation sequences in a symbolical form among the

registers of a digital module. Unlike a programming language; a register transfer language is directly related to and cannot be separated from the registers and other hardware that it defines.

INTER-REGISTER TRANSFER

Data transfer among registers is accomplished by means of inter-register transfer micro-operations. These micro-operations perform a direct transfer of binary information from one register to another. The destination register receives the information from the source register retaining the contents in source register.

Information transfer from one register to the another can be performed either in parallel or in serial.

Parallel transfer

Parallel transfer is a simultaneous transfer of all bits from the source register to destination register and is accomplished during one clock pulse. This micro-operation is designated in symbolic form as follows:

$$A \leftarrow B$$

And denotes a transfer of the content of register B into register A.The binary function which determines when this transfer will take place is called a control function.A control function is a binary function that is it can be either equal to 0 or 1.Thus the modified symbology for the above transfer can be written as

$$P: A \leftarrow B$$

The control function P symbolizes the fact that the stated micro-operation is executed by the hardware only if P=1.

Fig 1 shows the hardware for implementing the micro-operations $P: A \leftarrow B$. The output of register B are connected to the inputs of register A. Reg A has parallel load capability that is ,the transfer occurs only if the load input is equal to 1.

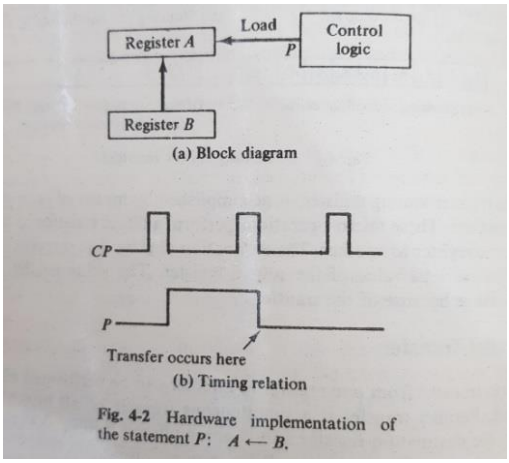


Fig. 4-2 Hardware implementation of the statement $P: A \leftarrow B$.

Understanding the basic symbols for the register transfer language

1. Capital letters are used to denote registers and subscripts denote individual cells of the register.eg A, B.

2.Paranthesis used to denote a portion of a register.eg. I (1-5), MBR(AD)

3.Arrow must be present in every micro-operation statement and denotes a transfer of the content of the register listed on the right side of the arrow into the register listed on the left side of the arrow.

4.Colon denotes a control function and the comma is used to separate two or more micro-operations when executed at the same time. Eg. F:A←B,B←A

SERIAL TRANSFER

For serial transfer, both the source and destination registers are shift-registers.The information is transferred one bit at a time by shifting the bits out of the source register into the destination register. In order not to lose the information stored in the source register it is necessary that the information shifted out of the source register be circulated and shifted back at the same time.

The serial transfer of information from Reg B to Reg A is done with shift registers as shown in the block diagram of fig 2.Both the registers A and B are 4-bit registers.The serial output of reg B comes from the output of the rightmost flip-flop B_4 .The serial input of reg A goes into the left-most flip-flop A_1.When the shift-right control S=1,and a clock pulse occurs,the content of reg A and B are shifted once to the right and the value of B_4 transferred to flip-flops A_1 and B_1.Due to this one bit is transferred from B to A and at the same time one bit is circulated back to reg B.

The symbolical notation for the transfer is as follows:

$$S:A_1 \leftarrow B_4, B_1 \leftarrow B_4, A_i \leftarrow A_{i-1}, B_i \leftarrow B_{i-1} \quad \text{where, } i=2,3,4$$

$$S:A \leftarrow B, B \leftarrow B$$

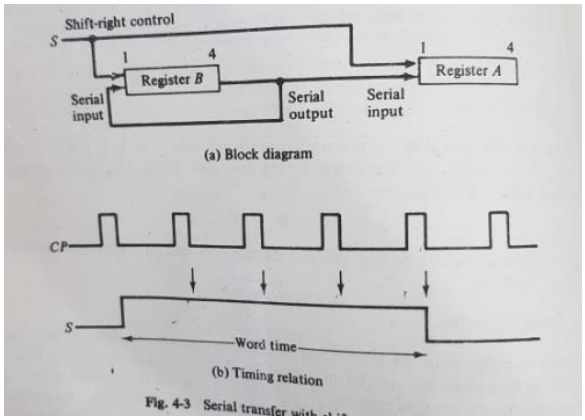


Fig. 4-3 Serial transfer

BUS TRANSFER

In a system with N registers, the transfer from each register to another register requires lines to be connected from the output of each flip-flop in the source register to input of each flip-flop in the destination register.

The fig below shows transfer of data among 3 registers R1, R2, R3. There are six data paths between these 3 registers. If each register consists of N flip-flops, there is a need of 6n lines for parallel transfer from one reg to another. As the number of register increases the number of lines also increases many-folds.

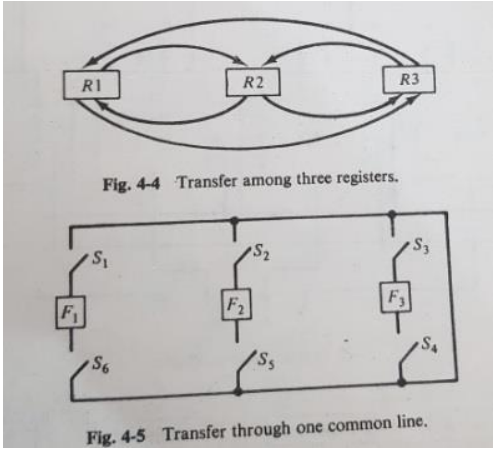


Fig. 4-4 Transfer among three registers.

Fig. 4-5 Transfer through one common line.

However, if one register is allowed to transfer at a time, we can reduce the number of lines as shown in the next fig where the output and input of each flip-flop of the register is connected to a common line through an electronic circuit that acts like a switch. All the switches are OPEN until a transfer is required. For a transfer of R1←R3, switch S3 when closed transfers data from R3 to BUS and then switch S6 when closed transfers the data from BUS to R1.

This scheme can be extended to registers with n flip -flops and require n common lines since each flip-flop of the register must be connected to one common line.

Bus system of data transfer using four registers

A group of wires through which binary information is transferred among registers is called a BUS.

A bus system is formed with multiplexer circuits. A multiplexer selects data from many lines and directs it to a single output line. The fig shows how four registers are connected through multiplexers to form one set of common bus lines. Each register has 4 bits. Each 4X1 multiplexer has four data input lines, two selection lines, and one output line. Reg A is connected to I_0 of the multiplexers, Reg B to I_1 and so forth. The selection lines act as control to all the multiplexers such that if xy =00, multiplexer inputs I_0 are selected and transferred to the output bus, i.e. 4-bit contents of reg A are transferred to the common bus. Activating the load input of a particular register will cause a transfer of information from the bus into register. When zw=00/01/10/11 then data gets transferred from bus to resp registerA/B/C/D.

xy =10 : BUS←C

zw=01 : R1(A)←BUS

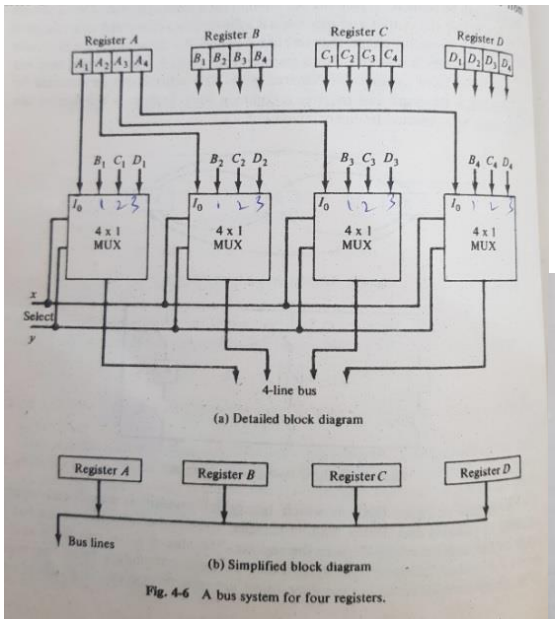


Fig. 4-6 A bus system for four registers.

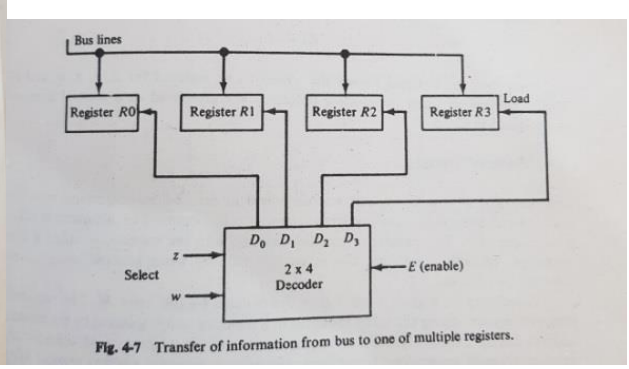


Fig. 4-7 Transfer of information from bus to one of multiple registers.

Memory Transfer

The transfer of information from memory to the external environment is called a Read Operation.

The transfer of information into the memory is called a write operation. In both operations, the particular memory word selected must be specified by an address. MAR a memory address register is connected to the address terminals of the memory.

A read micro-operation is a transfer from the selected memory register M into MBR.

MBR←M

This causes a transfer of one word into MBR from the selected mem reg M whose address is given by the address in MAR.

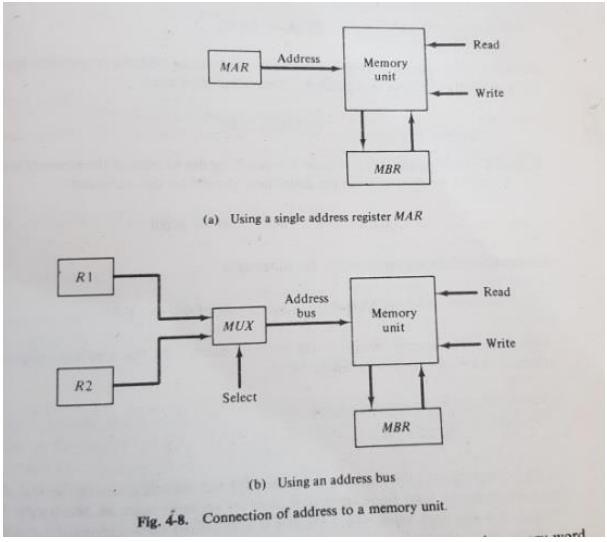


Fig. 4-8. Connection of address to a memory unit.

A *write* micro-operation is a transfer from MBR to the selected memory word M.This is shown symbolically as

M←MBR

This causes the transfer of a word from MBR into the mem reg M whose address is given by the address in MAR.

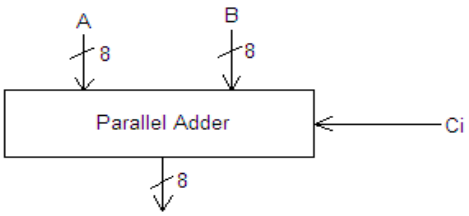
MBR←M[R1]

Denotes a read operation from the memory word whose address is specified by the reg R1.

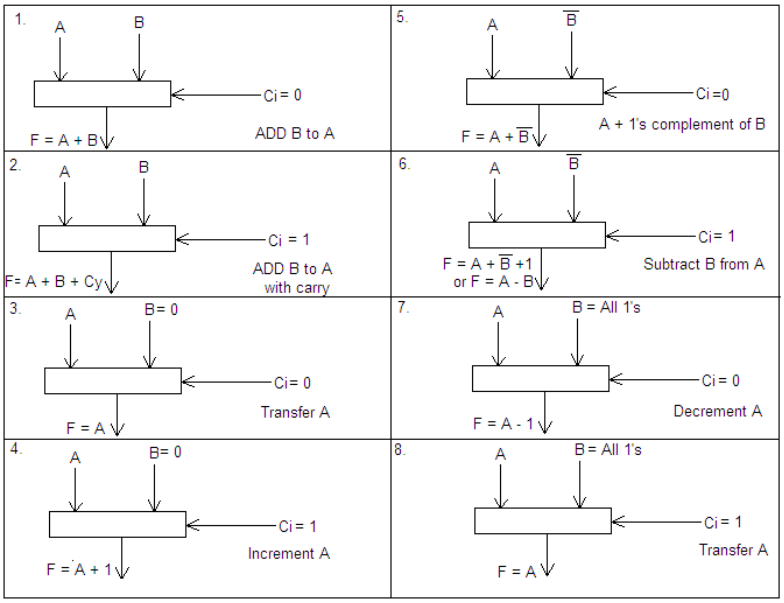
ARITHMETIC MICRO-OPERATIONS

ALU (Arithmetic & Logic Unit) –

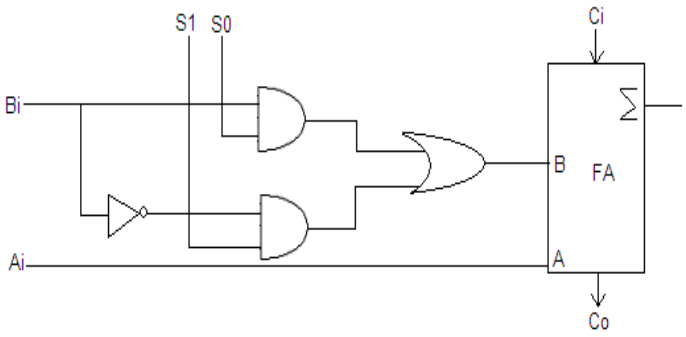
Arithmetic Circuit:



The ALU, Arithmetic and Logic Unit carry out the arithmetic and logic functions over the content of operands stored in register and memory. The general arithmetic functions are: addition, subtraction, increment, decrement etc.; whereas negotiation, AND, OR, XOR are the general logical operations that are carried out by this unit. An ALU is associated with a set of micro – instructions where every micro – instruction carried out a specific task.

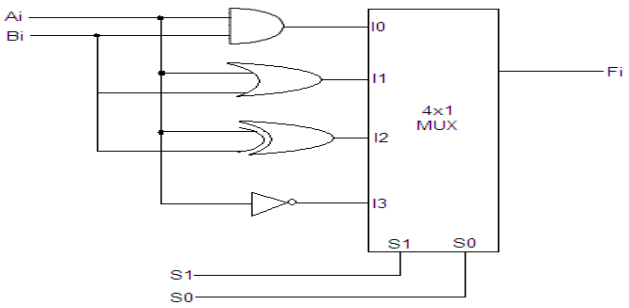


Arithmetic Circuit: A binary parallel adder can be used to perform some fundamental arithmetic operation over the operands. Thus, a simple arithmetic circuit can be constructed with the help of parallel adder and by providing different inputs in form of A, B and Ci, many arithmetic operations can be carried out. Some fundamental arithmetic operations can be obtained by a parallel adder is shown in figure –



S1	S0	Ci	Output	Remark
0	0	0	F : A	Transfer A
0	0	1	F : A+1	Increment A
0	1	0	F : A+B	ADD B to A
0	1	1	F : A+B+1	ADD B to A with carry
1	0	0	F : A+$\overline{B}$	ADD $\overline{B}$ to A
1	0	1	F : A+$\overline{B}$+1	Subtract B from A
1	1	0	F : A-1	Decrement A
1	1	1	F : A	Transfer A

Logic Circuit:



S1	S0	Fi		
0	0	I0	A^B=>A.B	AND operation
0	1	I1	AˇB=>A+B	OR operation
1	0	I2	$A \oplus B$	XOR operation
1	1	I3	$\overline{A}$	Complement of A

Logic Circuit: A logic unit basically carries out logical operations over the content of CPU registers. The given figure illustrates a simple logic circuit created with the help of some fundamental gates and a multiplexer. The circuit accepts two inputs Ai and Bi (a bit of value A and value B) and with reference to control S1 & S0, perform a relevant logic operations as depicted in table. When S1, S0 equals 00, the unit produces A.B output.

This one stage of the logic circuit can be repeated n number of times to carry out logic operation over n – bit data.

Truth table for 16 functions of two variables

x	y	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

F0=All 0’s

F1=x and y

F2=xy’

F3=xy’+xy=x(y+y’)=x

F4=x’y

F5=x’y+xy=y(x’+x)=y

F6=x xor y

F7=x or y

F8=x nor y

F9=x xnor y

F10=x’y’+xy’=y’(x’+x)=y’

F11=x’y’+xy’+xy=y’+xy

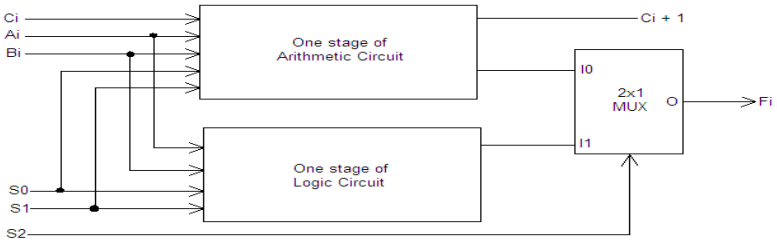
F12=x’y’+x’y=x’(y’+y)=x’

F13=x’y’+x’y+xy=x’+xy

F14=x nand y

F15=All 1’s

Arithmetic and Logic Unit –



The ALU can be constructed by simply combining the arithmetic unit along with the logic unit to carry out the arithmetic and logic operations. Figure illustrate a typical ALU where it comprises of one stage of arithmetic unit and one stage of logic unit connected to a 2x1 multiplexer. The circuit now has an additional control S2 which selects one of the operation i.e., arithmetic or logic produced at output. On whole the circuit now comprises of three controls S2, S1 and S0 that selects a arithmetic or logic operation. The arithmetic operation selection is also depending on the carry input Ci. The table below shows all the operations that are carried out in the above circuit.

S2	S1	S0	Ci	FUNCTION	
0	0	0	0	Transfer A	Arithmetic Operation
0	0	0	1	Increment A	
0	0	1	0	ADD B to A	
0	0	1	1	ADD B to A with carry	
0	1	0	0	ADD 1's complement of B to A	
0	1	0	1	Subtracts B from A	
0	1	1	0	Decrement A	
0	1	1	1	Transfer A	
1	0	0	×	A.B (A AND B)	Logical Operation
1	0	1	×	A+B (A OR B)	
1	1	0	×	$A \oplus B$ (A XOR B)	
1	1	1	×	$\overline{A}$	

CONTROL FUNCTIONS

The timig of all registers in a synchronous digital system is controlled by a master clock generator,whose clock pulse are applied to all flip-flops in the system. The binary variables that control the enable inputs of registers are called *control functions.*

The hardware control network that generates control functions can be organized in three different ways

1) As a sequential circuit
2) As a sequence of timing signals coupled with control conditions
3) As a control memory

TIMING SEQUENCES

Timing signals that control the sequence of operations in a digital computer can be generated with a binary counter and a decoder.

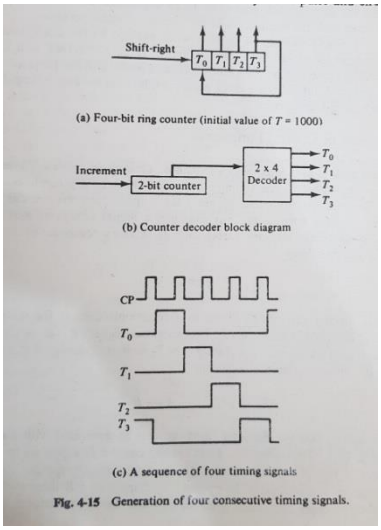


Fig. 4-15 Generation of four consecutive timing signals.

Generation of control functions

Each computer cycle is associated with a sequence of micro-operations. These micro-operations are controlled by the timing signals and other binary conditions in the system .The Boolean functions that generate the control decisions are the control functions.For eg

T1: MBR←M

T1=read-control input for memory unit

FT_1+R'T3: A←B

The control function being a Boolean function can be generated with logic gates as shown in the fig below. The gates that generate the control function go to the load input of reg A to initiate the transfer. The condition may also initiate the same transfer.

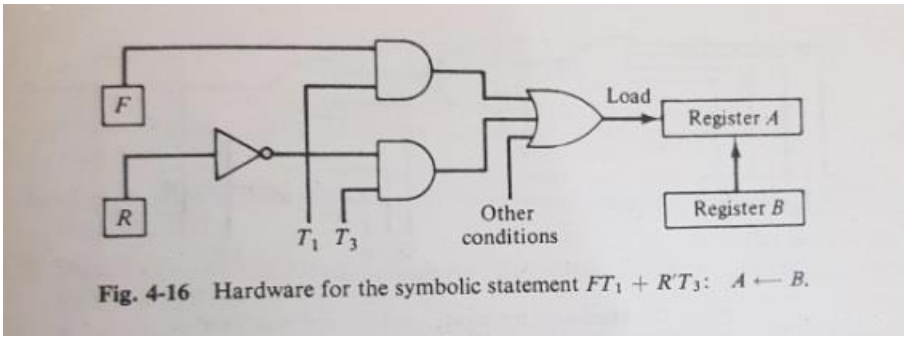


Fig. 4-16 Hardware for the symbolic statement $FT_1 + R'T_3$: $A \leftarrow B$.

BASIC COMPUTER ORGANIZATION AND DESIGN

Instruction codes

The internal organization of a computer system is defined by the sequence of micro-operations it performs on data stored in registers. A digital computer is a general purpose digital system. The user of such a system can control the process by means of a

program, that is a set of instructions that specifies the operations, operands and the sequence by which processing has to occur.

An instruction code is a group of bits that tell the computer to perform a specific operation. It is usually divided into more than one field ,the most basic of them being the operation field. The operation field specifies operations such as add, sub, shift left, logical and etc.

At this point we must clearly understand the relation between an operation and a micro-operation. An operation is a part of an instruction stored in computer memory. It is a binary code that tells the computer to perform a specific operation. The control unit receives the instruction from memory and interprets the operation code bits. It then issues a sequence of control functions that perform micro-operations in internal computer registers. For every operation code, the control issues a sequence of micro-operations needed for the hardware implementation of the specified operation. Due to this reason operation code is also referred to as macro-operation because it specifies a set of micro-operations.

Fig depicts an organization. Instructions are stored in one section of memory and data in another. For a memory unit with 4096 words we need 12 bits for address since 2^{12} =4096. If we store each instruction code in one 16-bit memory word, then we have four bits available for 16 possible operations and 12 bits to specify the address of the operand over which the operation will execute.

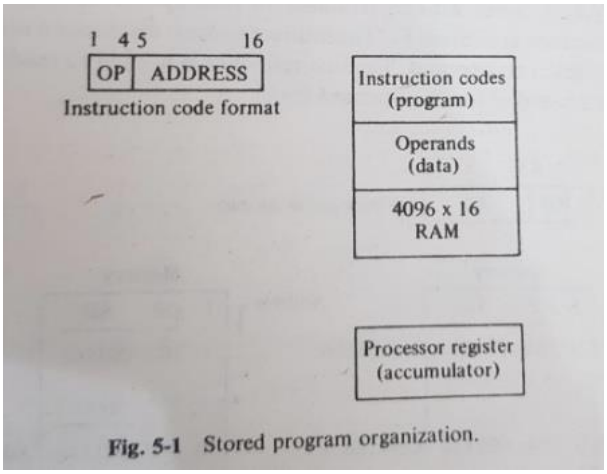


Fig. 5-1 Stored program organization.

Demonstration of direct and indirect instructions

It is sometimes convenient to use the address bits of an instruction code not as an address but as the actual operand. When the second part of an instruction code specifies an operand, the instruction is said to have an immediate operand. When the second part specifies the address of an operand, the instruction is said to have a direct address. The third possibility is an indirect address, where the bits in the second part of the instruction designate an address of a memory word in which the address of the

operand is found. The MSB generally differentiates between a direct or an indirect instruction format.

To illustrate the concept let us take an example of an instruction code format

It consists of a three-bit operation code designated by OP, a six-bit address part designated by AD, and an indirect-address mode designated by I. If I=0 it designates a direct address and if a 1 indicates indirect address.

A direct address instruction is shown in fig.2(b). It is placed in address 2 in memory. The I bit is 0,so the instruction is recognized as a direct address instruction. Since the address part Ad is equal to binary equivalent of 9(001001),the control finds the operand in memory at address 9. In fig 2(c) I=1 ,therefore it is a indirect address instruction. The address part is 9 so the control goes to 9 and finds the *address of operand*. The indirect address instruction needs two references to memory to fetch an operand.

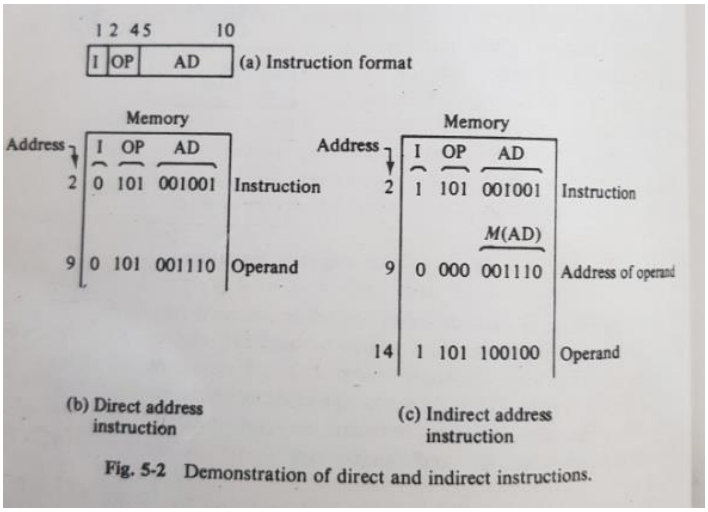


Fig. 5-2 Demonstration of direct and indirect instructions.

Computer Instructions

Computer instructions are stored in consecutive memory locations and are executed sequentially one at a time. The control reads an instruction from a specific address in memory and executes it and then continues onto the next instruction in sequence and so on. Memory words cannot communicate with the processor directly without going through an address and buffer register. It is also necessary to provide a register in control unit for storing operation codes after they are read from memory.

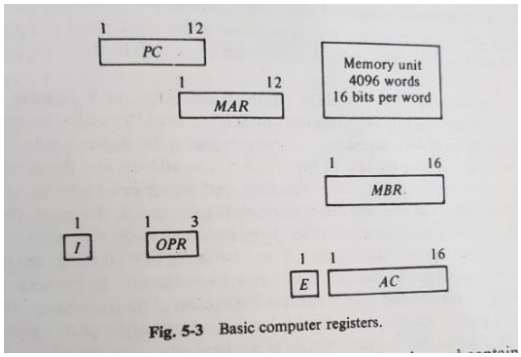


Fig. 5-3 Basic computer registers.

The memory unit has a capacity of 4096 words and each word contains 16 bits. Twelve words are needed to specify the address of an operand. This leaves four bits for the operation part of the instruction. However, only 3 bits will be used to specify the operation code. Where the fourth bit will be used as a direct an indirect address bit. The memory buffer register (MBR) consists of 16 bits ,as does the accumulator register. The E register is an extension of the accumulator and will be used during shift operations for end around carry. The I register has a single cell for storing the mode bit and the operation register OPR stores the three-bit operation code from memory.

The memory address register (MAR)has 12 bits storing the 12 bit of the address. The program counter(PC) is also 12 bit long since it stores the address of the next address to be fetched by the processor. This register goes through a counting sequence and causes the computer to read sequential instructions stored in memory.

To read an instruction, the content of PC is transferred to MAR, a memory read cycle is initiated, and the PC is incremented by 1.This place the instruction code into MBR and prepares PC for the address of the next instruction. The operation code is transferred to OPR, the mode bit into I and the address part into MAR.

The basic computer has three different instruction code formats. The operation part contains three bits; the meaning of the remaining thirteen bits depends on the operation code encountered.

I	OPR	Instruction type
0	000	Direct-memory Reference instruction
0	001	
0	010	
0	011	
0	100	
0	101	
0	110	
0	111	Register-reference instruction
1	000	Indirect memory Reference instruction
1	001	
1	010	
1	011	
1	100	
1	101	
1	110	
1	111	IO reference instruction

A computer should have all the necessary and sufficient operations to carry out any conceivable processing task. It should contain instructions in each of the following categories:

1. Arithmetic, logic, and shift micro operations
2. Instructions for moving info to and from memory and processor registers
3. Instruction that check status info to provide decision making capabilities.
4. Input and output instructions.
5. The capability of stopping computer.

TIMING and CONTROL

When an instruction is read from memory, the computer is said to be in an instruction fetch cycle. When the word read from memory is an address of an operand the computer is in an indirect cycle. When the word read from the memory is an operand, the computer is in a data execute cycle. It is the function of the control to keep track of the various cycles.

The control unit uses two flip-flops to distinguish between the three cycles. These flip-flops are denoted by the letters F and R. A 2X4 decoder associated with these flip-flops provide four outputs , three of which can be used to differentiate between the above mentioned cycles.

The block diagram of the control unit for the basic computer is show in fig. The timing in the computer is generated by a 2-bit sequence counter(SC) and a 2X4 decoder. The timing signals out of the decoder are designated by t_0,t_1,t_2 and t_3.

Flip-Flops F R	Decoder Output	Computer Cycle
0 0	c0	Fetch Cycle
0 1	c1	Indirect Cycle
1 0	c2	Execute Cycle
1 1	c3	Interrupt Cycle

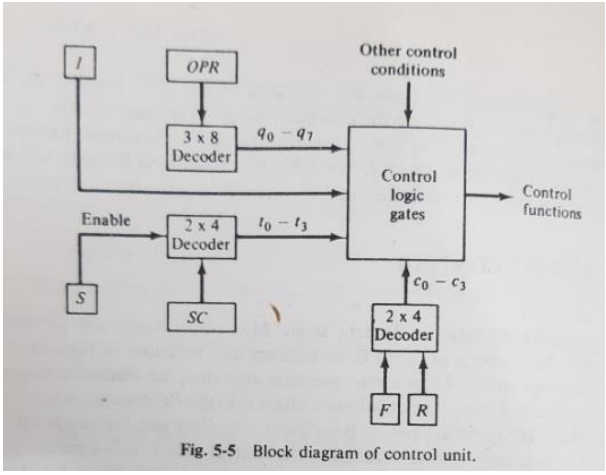
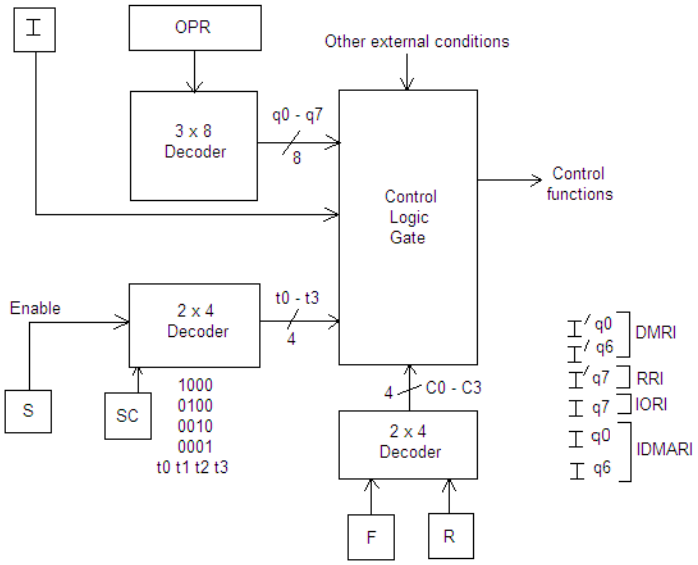


Fig. 5-5 Block diagram of control unit.

Timing and Control –



F	R	Decoder O/P	Computer cycles
0	0	C0	Fetch Cycle
0	1	C1	Indirect Cycle
1	0	C2	Execute Cycle
1	1	C3	Interrupt Cycle

A digital computer operates over a given set of instructions to accomplish a problem solution. The instruction comprises of a sequence of micro operations that are needed to be carried out when the instruction is executed.

The unit of CPU that generates the sequence of micro operations for a given instruction is termed timing and control unit. A simple computer system CPU comprises of four machine cycle: Fetch Cycle, Indirect Cycle, Execute Cycle and Interrupt Cycle.

Figure illustrates a typical control unit block diagram. It consist of a control logic circuit associated with several functional blocks generating various control functions for a given instruction. It consists of following:

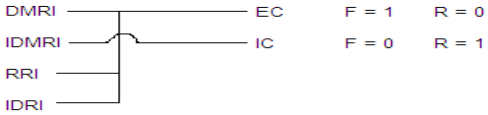
1**. 3 x 8 Decoder:** It accepts the three bit operation code of instruction and accordingly generate control signals q0…….q7. For example, if OPr is 011 than q3 is enable.

2**. I**: It is the indirect flip – flop that accept a value from I field of instruction. If I is 1 than q7 is determine. If q7 is 1 than an I–O instruction is carried out. If I is 0 and q7 is 1 than a register operation is carried out.

3. **2 x 4 Timing Decoder:** This decoder provides four clocks t0 – t3, i.e., every cycle comprises of four clocks. This decoder is driven by the sequence counter SC and by a enable flip – flop S. if S is 1 than only the cycle is executed.

4. **2 x 4 Decoder (Control 2 x4 Decoder):** This decoder accepts F and R bits and depending on their values generates the respective cycle as depicted in table.

C0 t0	:	MAR ← PC	
C0 t1	:	MBR ← M,	PC ← PC + 1
C0 t2	:	OPR ← MBR (OP),	I ← MBR (2)
q'7 IC0t3	:	R ← 1	
(q7 + I') C0t3	:	F ← 1	

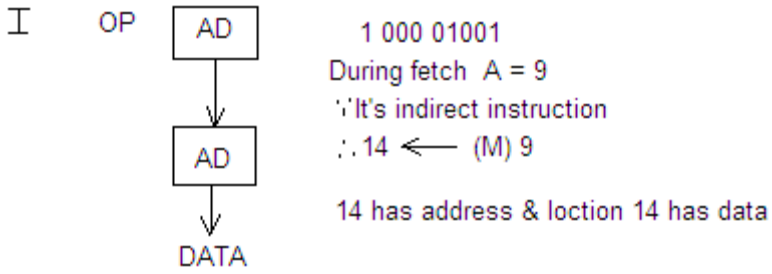


I	OP	AD
0	q0 000 ⋮ q6 110	DMRI
0	q7 111	RRI
1	q0 000 ⋮ q6 110	IDMRI
1	q7 111	IORI

Fetch Cycle: It is the first machine cycle of every instruction cycle. The task of this cycle is to fetch the instruction from the memory, to decode it and then to call the relevant machine cycle as either executes cycle or indirect cycle. The various tasks performed in fetch cycle at different clocks are shown above. In first clock to the content of program counter is transfer to MAR. In second clock t1, two tasks are carried out, i.e. the program counter is incremented by one as well as the memory location is read out in MBR. In third clock t2, the I and OP fields are filled up from fetched instruction. In fourth clock t3, the design is taken to call indirect cycle or execute cycle. This design is based on q7 and I. if q7 = 0 and I = 1 then indirect cycle is called else the execute cycle is called.

Indirect cycle:

C1 t0	:	MAR ← MBR (ADDR)
C1 t1	:	MBR ← M
C1 t2	:	NOP
C1 t3	:	F ← 1, R ← 0



The indirect cycle is used to fetch the address of operand. The indirect cycle receives the pointer value from fetch cycle. In the given machine C1 represents the indirect cycle. On the first clock of this machine cycle the address part obtained from the fetch cycle is used to get the address of operand, it is done in second cycle. The third cycle is NOP (No OPeration). The fourth cycle is used to call the execute cycle.

ADD to AC (Accumulator)

q1 C2 t0	:	MAR ← MBR (AD)
q1 C2 t1	:	MBR ← M
q1 C2 t2	:	EAC ← AC + MBR
q1 c2 t3:	Fetch Cycle Call	

Instruction Cycle:

An instruction cycle is a composition of various machine cycles. The general machine cycles are:

1. Fetch instruction
2. Decode instruction
3. Fetch Operand if any
4. Store results.
5.

Every machine cycle in term is a composition of clocks where on each clock micro operations are carried out.

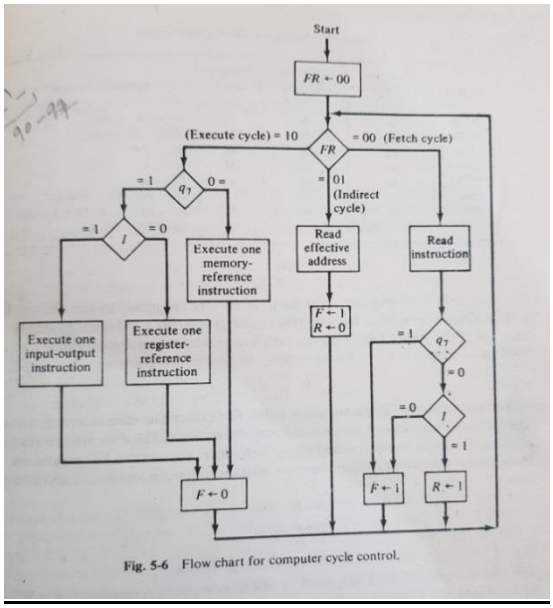


Fig. 5-6 Flow chart for computer cycle control.

Interrupts –

An interrupt is a signalling mechanism by which the external entity or the internal program itself can as the CPU to perform some special services that are dedicates routines or called Interrupt Service Routine (ISR).

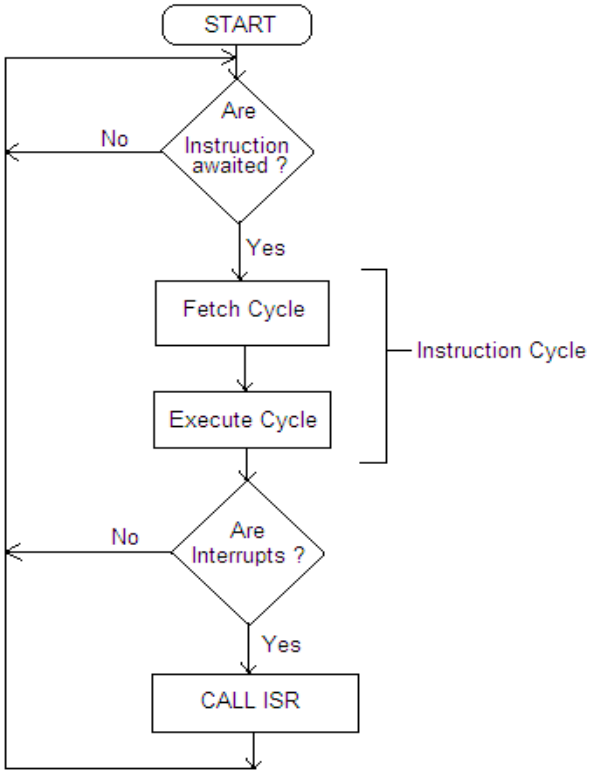
A computer system comprises of two types of interrupt called the Hardware Interrupt and Software Interrupt. The interrupt that are generated over the pin of a CPU (Microprocessor) are termed hardware interrupt whereas the interrupt generated programmatically are termed software interrupt.

The interrupt may be vector or non – vector. If the service routine address of an interrupt is fixed that is assign with a fixed address than is called a non – vector interrupt else if the address of interrupt is provided by the interrupting device than it is called a vector interrupt.

Figure illustrates a flow chart for interrupt handling. Initially the CPU waits for an instruction. When an instruction is submitted the fetch and execute cycles are performed i.e. the instruction cycle is carried out. At the end of instruction cycle the interrupt status is checked. If more than one interrupts are there than their priority are resolved and the interrupt with highest priority is served.

Calling an executive interrupt service routine:

Once the interrupt is identified the CPU saves the content of register onto the stack, transfer the control to ISR, disable interrupts, and services interrupt enable the interrupt and reload the saved register values and contents.



PROCESSOR BUS ORGANIZATION

The part of a computer that performs the bulk of data processing operations is called the central processor unit and is referred to as the CPU. When a large number of registers are included in a processor unit, it is most efficient to connect them through a bus system. The registers communicate with each not only for direct data transfers, but also while performing various micro-operations. Hence it is necessary to provide a common unit that can perform all arithmetic, logic, and shift micro-operations in the processor.

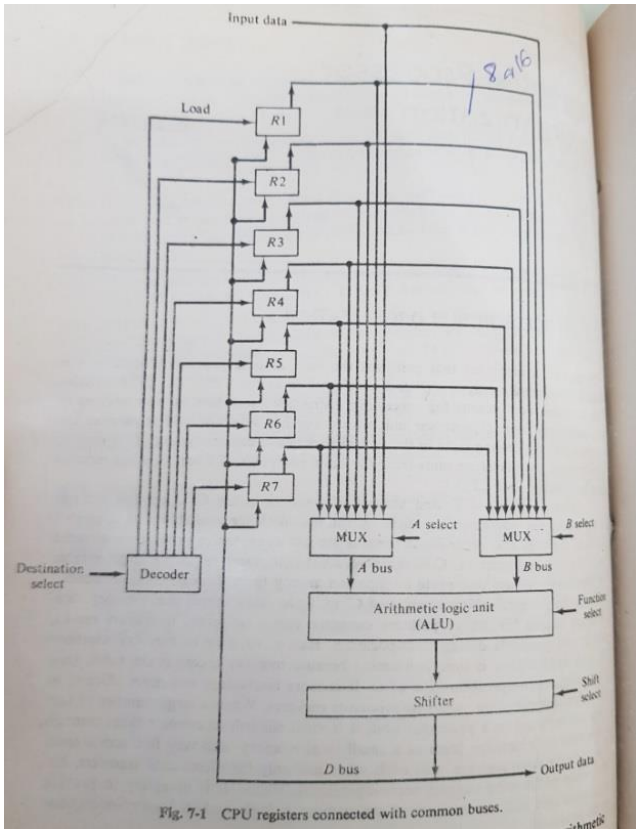


Fig. 7-1 CPU registers connected with common buses.

The fig shows a bus organization for seven CPU registers. The output of each register is connected to two multiplexers to form input buses A and B. The selection lines in each multiplexer select one register or the input data for the particular bus. The A and B buses form the inputs to a common arithmetic logic unit(ALU). The function selected in the ALU determines the arithmetic or logic micro-operation that is to be performed. The shift micro-operations are implemented in the shifter. The shifter unit is attached to the output of the ALU to provide a post shift capability. The result of the micro-operations is available for output data and also goes through the destination bus and into the inputs of all registers. The destination register that receives the information from the D bus is selected by a decoder.The decoder activates one of the register load inputs,thus providing a transfer path between the data in the D bus and the inputs of the selected destination register.

R1← R2+R3

The control must provide binary selection variables to the following selector inputs:

1. MUX A selector: place the contents of R2 into bus A
2. MUX B selector: place the contents of R3 into bus B
3. ALU function selector: to provide the arithmetic operation A+B
4. Shift selector: for direct transfer from the output of ALU into output bus D (no shift)

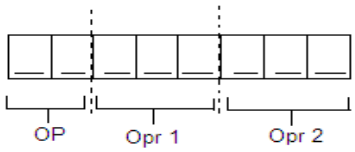
Decoder destination selector: to transfer the contents of bus D into R1.

INSTRUCTION:

→ Operation

→ Operand(s)

ADD A, B A (Operation) ←(Operand) A (Operation) +(Operand) B (Operation)



0 0	ADD	OPR1 / OPR 2	
0 1	SUB	Register A	0 0 0
1 0	AND	Register B	0 0 1
1 1	OR	Register C	0 1 0
		Register D	0 1 1
		Register E	1 0 0
		Register F	1 0 1
		Register G	1 1 0
		Register H	1 1 1

Example: ADD A, B

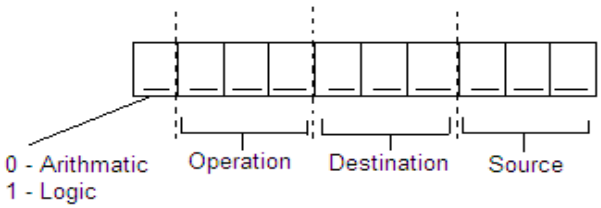
00 000 001

01 H

Example: AND C, D

10 010 011

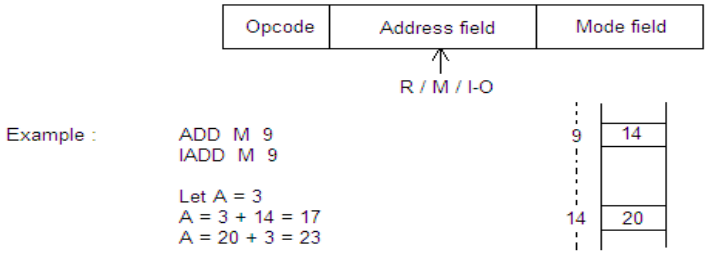
93 H



	Arithmetic	Logic	Destination / Source	
000	NOP	NOP	000	Register A
001	ADD	AND	001	Register B
010	SUB	OR	010	Register C
011	MUL	XOR	011	Register D
100	DIV	NOT	100	Register E
101	INR	____	101	Register F
110	DCR	____	110	Register G
111	____	____	111	Register H

Instruction Format:

1. Opcode
2. Address field
3. Mode field



1. Single Register (Accumulator Based) – One address
2. Multi Register (General Register) –
   - 3 Address
   - 2 Address

3. Zero Register (Stack Oriented) – Zero address

One address: ADD B(Source) : A ← A + B Default destination is accumulator

Three address: ADD A(Destination), B(Source S1), C(Source S2) : A ← B + C

Two address: ADD A(Destination D / Source S1), B(Source S2) : A ← A + B

Zero address:

- PUSH
- POP
- ADD

(On Stack)

Instruction Format: Computer program comprises of a sequence of instructions where an instruction is defined as a collection of operators and operand. The operators may be arithmetical, logical, relational, assignment etc. whereas the operands are the values stored in register, memory or the value of I/O port. The way of representing the operators and the operands with a mechanism how to access operands is termed as an instruction format. Thus an instruction format is a sequence

of bit representation where the bits are divided into groups called fields and the binary value of each field is defined to carry out a specific task.

For example if we consider 8 – bit instruction format than →

1. Two bits for operation:

(a) 00 ADD

(b) 01 SUB

(c) 10 MUL

(d) 11 DIV

2. 3 + 3 bits for any of six register and memory, i.e.,

(a) 000 Register A

(b) 001 Register B

(c) 010 Register C

(d) 011 Register D

(e) 100 Register E

(f) 101 Register F

(g) 110 Register G

(h) 111 Register H

Thus, ADD A,B in instruction format is written as

0 1

00 000 001

ADD A B

Thus 01 H becomes a code for this instruction.

The most common fields found in an instruction format are:

1. An operation code field specifying what operation to do.
2. An operation field(s) specifying the address of source and destination operands.
3. The MOD (mode) field that specifies the way of accessing operands.

The instruction format typically depends on the computer organization i.e., depends on the capability of CPU.

A computer system may have several instructions with several instructions formats. Most computers fall in one of the three types of CPU organization:

1. Single accumulator organization.

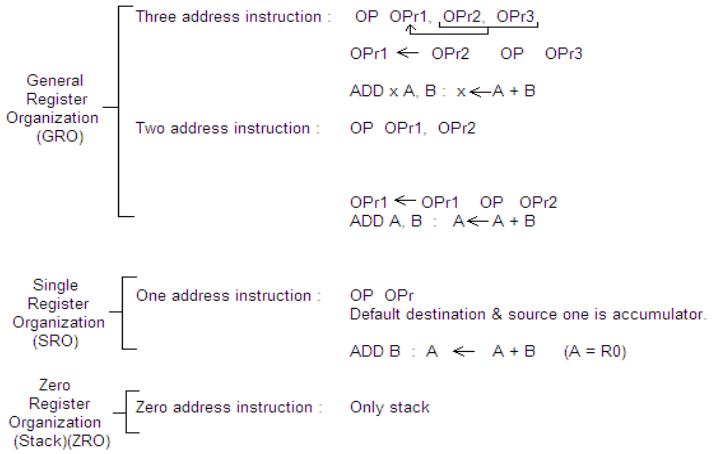
2. General register organization.

3. Stack organization.

A CPU comprising of only one register called accumulator represented by A or R_0 where the CPU does not have any other GPR is termed a Single accumulator organization. Here all the instruction with respect to computational tasks are executed over accumulator only.

The General register organization comprises of more than one register i.e., other than accumulator there are several GPR's in this kind of system there exist a flexibility of defining large instruction set increasing the computational power. For example INTEL 8085 falls in this category.

In Stack organization system the CPU do not posses any kind of GPR and hence all operations are carried out with reference the data structure called stack. This is also called zero register organization.



Example: $x = (A + B) * (C + D)$

Solve this by all instruction format organization.

Solution:

1. (Three register)

```
ADD   R1, A, B
ADD   R2, C, D
MUL   X, R1, R2
```

2. (Two register)

```
MOV   R1, A
ADD   R1, B
MOV   R2, C
ADD   R2, D
MUL   R1, R2
MOV   X, R1
```

3. (One register)

```
LOAD  A
ADD   B
STORE T1
LOAD  C
ADD   D
MUL   T1
STORE X
```

(NOTE: Here A, B are variables)

4. (Zero register)

```
PUSH  A
PUSH  B
ADD
PUSH  C
PUSH  D
ADD
MUL
POP   X
```

Three address instruction: This type of instruction format comprises of three operands called OPr1, OPr2 and OPr3. The operation is performed over OPr2 and OPr3 and the result is transferred to OPr1. Thus, there are four fields in three address instructions: The first for operation and remaining three for operands. This kind of instruction format exists in general register CPU organization.

Two address instruction: This is the most widely used instruction format in general register organization. This type of instruction format comprises of three fields called operation, OPr1 and OPr2. The operation is performed over OPr1 and OPr2 and the result is stored in OPr1. Thus OPr1 acts as Source & Destination register.

One address instruction: This instruction is available in single register organization as well as in general registers organization. The instruction comprises of two fields only: The operation and the operand. Here the default destination is accumulator register or in other words the operation is performed over specified operand and accumulator, store result in accumulator.

Zero address instruction: This instruction format is exist in stack oriented system, where the operands are stored onto the stack and the specified operations are carried out over the content of stack only.

Addressing Mode→

The way to access an operand in an instruction is termed as addressing mode or the way of getting an operand (OPr) from the system memory or register to carry out an instruction is defined as an addressing mode. The addressing mode technique provides a greater facility of accessing the operands while creating pointers indexes and relocations. The different addressing modes available are:

1. **Implied addressing mode:** In this mode the instruction itself is self-explanatory showing the operation to be carried out over the operands i.e. abbreviated in instruction itself.

For example CMA, complement accumulator instruction indicates that the content of accumulator is to be complemented. The zero address instruction or stack oriented instruction also falls in this category.

For example ADD, automatically adds the two top contents of stack.

2. **Immediate mode:** In this mode the operand is specified in the instruction itself i.e., the operand value is part of instruction or one of the field of instruction is operand.

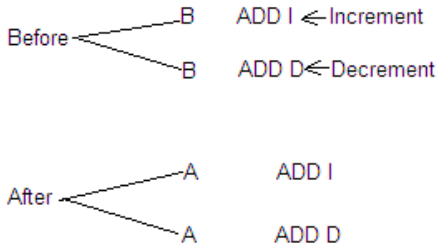
For example CPI 32H; MVI A, data; ANI 0FH; etc.

3. **Register direct mode:** In this mode the operand lie in one of the register of CPU. Thus, in this mode one field of instruction holds an address of a register.

For example MOV A, B; ADD B

4. **Register indirect mode:** In this addressing mode the specified register or register pair holds the address of operand i.e., on field of the instruction specifies a register and the specified register holds an address of operand.

5. **Auto increment / auto decrement:**



This addressing mode is similar to that of register indirect mode accept that before accessing the memory contents, the value of register is incremented or decremented before or after wards.

For example:

Let R = B

B = 06

B	ADD I	[R]	$R \leftarrow R + 1$ $A \leftarrow A + [M]_{R+1}$	B = 7 $A = A + [M]_7$
B	ADD D	[R]	$R \leftarrow R - 1$ $A \leftarrow A + [M]_{R-1}$	B = 5 $A = A + [M]_5$
A	ADD I	[R]	$A \leftarrow A + [M]_R$ $R \leftarrow R + 1$	$A \leftarrow A + [M]_6$ B = 7
A	ADD D	[R]	$A \leftarrow A + [M]_R$ $R = R - 1$	$A \leftarrow A + [M]_6$ B = 5

Direct Address Mode

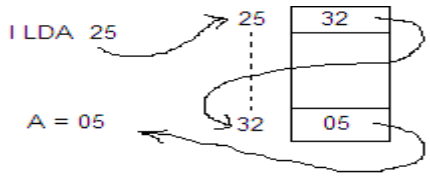
Example: LDA 5900

ADD 6501

The direct addressing mode indicates that the address of operand is specified as part of instruction. In other words one field of instruction comprises the address of the operand.

Indirect Address Mode

EA (Effective Address) = PC (Program Counter) + offset



In this mode the specified address field contains the address of operand indirectly, i.e. the specified location holds the address of the operand.

For example ILDA 25 indicates the location 25 contains the address of the operand.

Relative Addressing Mode

5904 : REL ADD 50 (2 bytes)
5906 :

EA = 5906
+50
5956

In relative address mode the address of operand is obtained by adding the specified data value to the content of program counter. Thus, if REL ADD 50, a two byte instruction at 5904 computes an address 5956 as an address of operand.

Indexed and Base Addressing Mode

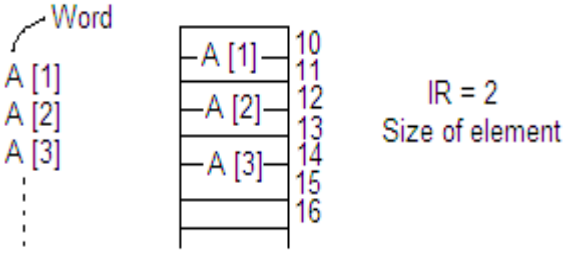
Indexed addressing mode : EA = Index register + Offset

Base addressing mode : EA = Base register + Offset

The index and base addressing technique are similar to relative index technique where in the indexed mode, the effective address is obtained by adding the offset to the index register.

In case of base addressing mode, the effective address is obtained by adding the offset of the instruction to base register.

The index address is generally use in array and string manipulation whereas the base addressing is used for relocation of program.



Stack Organization

A very useful feature that is included in the CPU of many computers is a stack or LIFO list. A stack is a storage device which store information in such a manner that the item stored last is the first retrieved item. The stack in digital computers is essentially a memory unit with an address register that can only count; no other value is ever loaded in the address register.

This type of address register is called a stack pointer because its value always points to the address of the top word in the stack.

The two operations of the stack are the insertion and deletion of items. The operation of insertion is called *push* and the operation of deletion from the stack is called *pop*.

Fig shows the organization of a 64 word memory stack.The stack pointer register SP contains the address of the word which is currently on the top of the stack.

In a 64-word memory stack,the stack pointer contains 6 bits since 2^6=64.The one bit registers FULL and EMPTY are set when the stack is full or empty of items,resp.The MBR is the memory buffer register that holds the binary items to be written into or read out of the memory stack.

PUSH operation

SP←SP+1	Increment stack pointer
M[SP]←MBR	write item on top of stack
If (SP=0) then (FULL←1)	check if stack is full
EMPTY←0	mark the stack not empty

POP operation

MBR←M[SP]	Read item from the top of the stack
SP←SP-1	decrement stack pointer
If (SP=0) then (EMPTY←1)	check if stack is empty
FULL←0	mark the stack not full

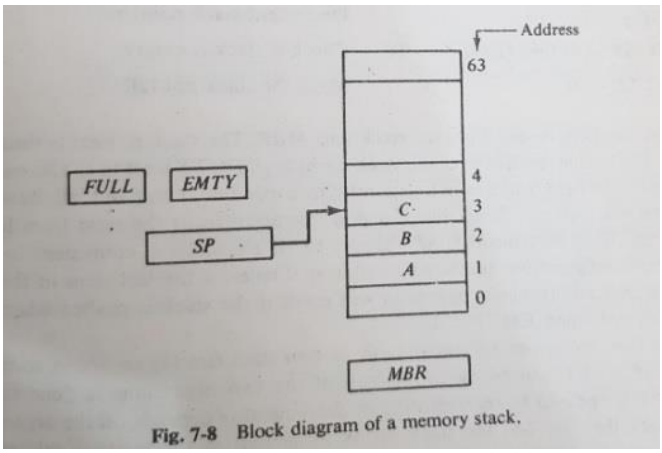


Fig. 7-8 Block diagram of a memory stack.

INPUT-OUTPUT ORGANIZATION

The input-output subsystem of a computer provides an efficient mode of communication between the central system and the outside environment. Programs and data must be entered into computer memory for processing and results obtained from computation must be recorded or displayed for the user. A computer serves no useful purpose without the

ability to receive information from an outside source and to transmit result in a meaningful form.

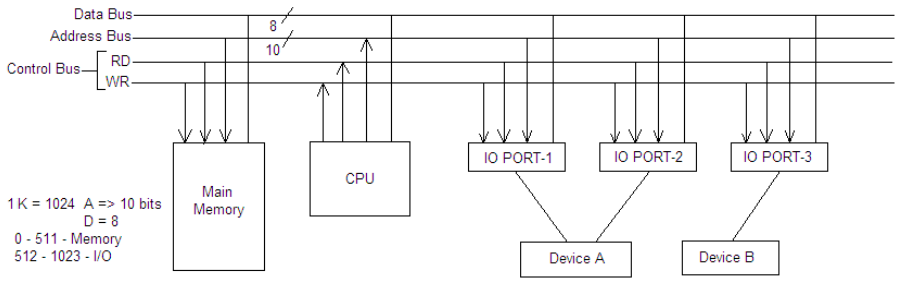
I/O interface

Input-output (I/O)interface provides a method for transferring binary information between internal storage, such as memory and CPU registers, and external I/O devices. The purpose of the communication link is to resolve the differences that exist between the central computer and each peripheral. The major differences are

1. Peripherals are electro-mechanical devices and their manner of operations is different from the operation of the CPU and memory which are electronic devices.
2. The data transfer rate of peripherals is much slower then the transfer rate in the central computer.
3. The operation of the peripherals must be synchronized with the operation of the CPU and memory unit.
4. Data formats in peripherals differ from the word format in the central processor.
5. The operations of each peripheral must be controlled so as not to disturb the operation of the central computer and other peripherals connected to the system.

To resolve all these differences, computer system invariably includes special hardware components between the central computer and peripherals to supervise all input and output transfers. Large computers include an I/O processor also sometimes referred t to as a data channel controller since it controls and regulates the flow of data to and from the internal and external parts of the computer.

Programmed I/O→



Memory Mapped I/O

A computer system with CPU, Memory and I/O requires mapping of memory and I/O devices such that the CPU can access them. There are two fundamental approach of defining memory space and I/O space. These are –

1. Memory Mapped I/O

2. Isolated I/O

1.Memory Mapped I/O –

Figure shows a typical memory mapped I/O scheme comprising of

(i) CPU
(ii) Main Memory
(iii) I/O Ports

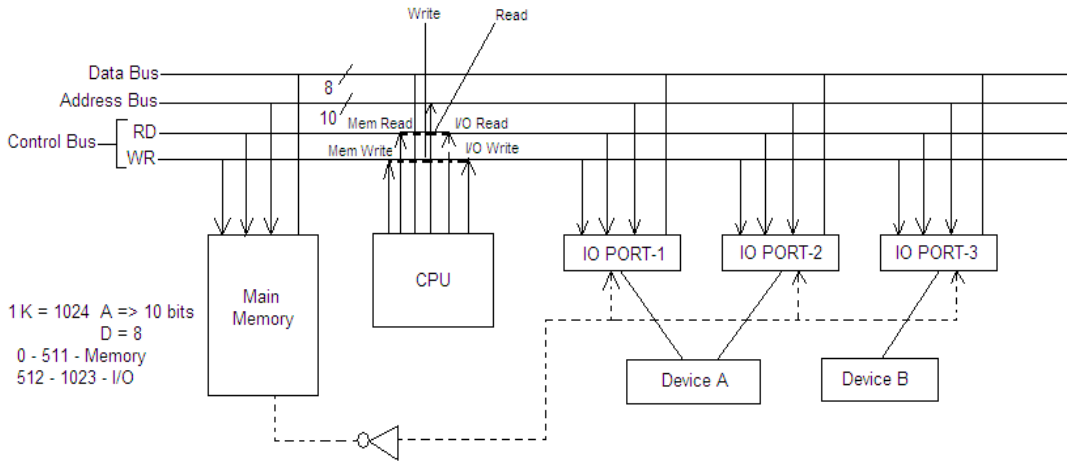
The device defined above shares three buses; The Address Bus, the Data Bus and Read – Write signals (i.e. Control Bus). The address is assumed to be generated only by CPU, the Read – Write signals are also generated by CPU.

Here the addressing range of CPU is divided into two parts where the lower part is dedicated to memory addressing and the higher order address are defined for I/O devices. The configuration in this manner does not discriminated between memory and I/O and hence simple Read – Write signals can be used to read or write both memory as well as input – output devices. Assuming 1K addressing range and 8 – bit data, we have 10 address lines and 8 data lines. Again the addressing is divided as 0 – 511 for memory and 512 – 1023 for I/O. if the CPU generates an instruction Read [235] then a memory location is read and if a CPU generates an instruction Read [812] then the CPU reads an I/O. thus a single type of instruction can be used to access both Memory as well as I/O.

The advantage of this concept is that the architecture is simple, same set of R-W instruction can be used to read or write both memory and I/O.

The disadvantage of this technique is that if we increase the memory addressing, the I/O addressing decreases and vise-versa. Thus for lower addressing this technique is useful and if memory requirement is too high along with high I/O, this technique is not used.

2. Isolated I/O –



Isolated I/O

A = 10 bits

Memory: 0 – 1023

I/O: 0 – 1023

MR	MW	I/O R	I/O W	
0	0	0	0	No Operation
1	0	0	0	Memory Read
0	1	0	0	Memory Write
0	0	1	0	I/O Read
0	0	0	1	I/O Write

Read Memory [Address] Read I/O [Address]
Write Memory [Address] Write I/O [Address]

The draw backs of memory Mapped I/O can be overcome by Isolated I/O scheme. It is also called as I/O Mapped I/O. The figure shows a typical isolated I/O scheme. It comprises of CPU, Memory and I/O devices. All the three devices shares a common address bus as well as data bus but the control bus can be of two types:

Type 1: Separate R-W signals for both memory and I/O.

Type 2: Same R-W signals for both memory and i/O with an additional IO/M signals to discriminate between them (Shown by dotted lines).

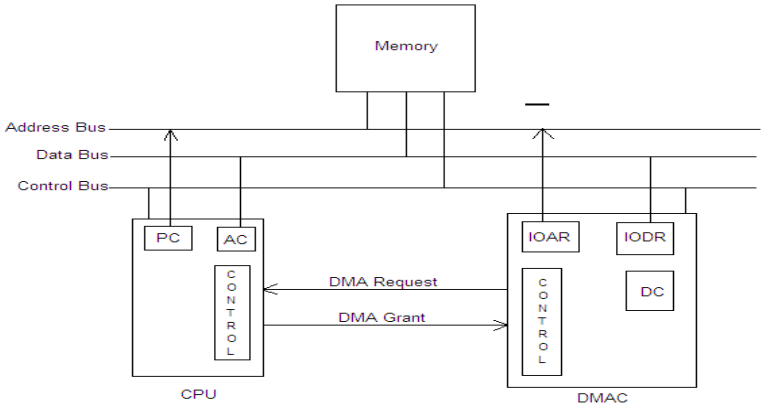
Here the same addressing range is available for both the memory and I/O devices. Assuming 10 bit address, the range 0 – 1023 is available for both memory as well as I/O. To read a memory, a read memory instruction is used where as to read I/O a read I/O instruction is used. Thus in this concept separate R – W instruction is needed which is an overhead in form of pins of CPU as well as in form of instruction.

In other concept we have same R – W instruction but we have to generate a separate discriminates signal in form of IO/M such that when it is 1 an I/O is read or written and when it is 0 a memory is read or written.

The advantage of this technique is that a large address range is available for the capability of CPU for both memory & I/O.

The disadvantage of this technique is large instruction is requirement or a separate signal generator which is an overhead.

Direct Memory Access (DMA) –



Direct Memory Access: The Direct Memory Access is a mechanism of transferring data between memory and I/O devices without intervention of CPU but under the control of CPU.

Under normal circumstances called PROGRAM – IO – MODE, the data transfer between I/O devices and Memory is through CPU i.e. the CPU reads memory than gives data to I/O or CPU reads data from I/O and provides it to memory. In this concept a considerable system time as well as the CPU time is wasted in simply transferring the data between memory & I/O.

To speed up the data transfer and to save the system time, DMA is used. Figure illustrates a typical DMA mechanism. It comprises of three elements: CPU, Memory & DMAC (Direct Memory Access Controller).

The CPU is shown with PRORAM COUNTER: The most Special Purpose Register (SPR) that holds the address of next location to be accessed and the ACCUMULATOR: The most General Purpose Register (GPR) for reading and writing data.

The DMA Controller comprises of –

2. **IOAR:** The Input – Output Address Register used to generate the address of memory that is to be accessed.
3. **IODR:** The Input – Output Data Register is used to read data from memory or is used to write data.
4. **DC:** The Data Count register is a special register which holds a value of the number of data to be transferred during DMA operation.

The CPU and DMAC are associated with two hand shake signals –

1. DMA Request
2. DMA Grant

DMA Transfer: Under normal circumstances the CPU access the memory. When a I/O devices wants a transfer from memory than it generates the address in IOAR, fills the DC with numbers of transfer required and then generated the DMA request consequently the CPU responds by generating DMA Grant signal. At this point the CPU is isolated from memory and the DMAC gets the control of memory. The DMAC than transfer the data and decrement the DC register. The transfer is goes on till DC is not equal to zero.

When DC reaches zero, the DMAC disable DMA Request and subsequently the CPU disable DMA Grant. The CPU then regains the control of memory and resume normally.

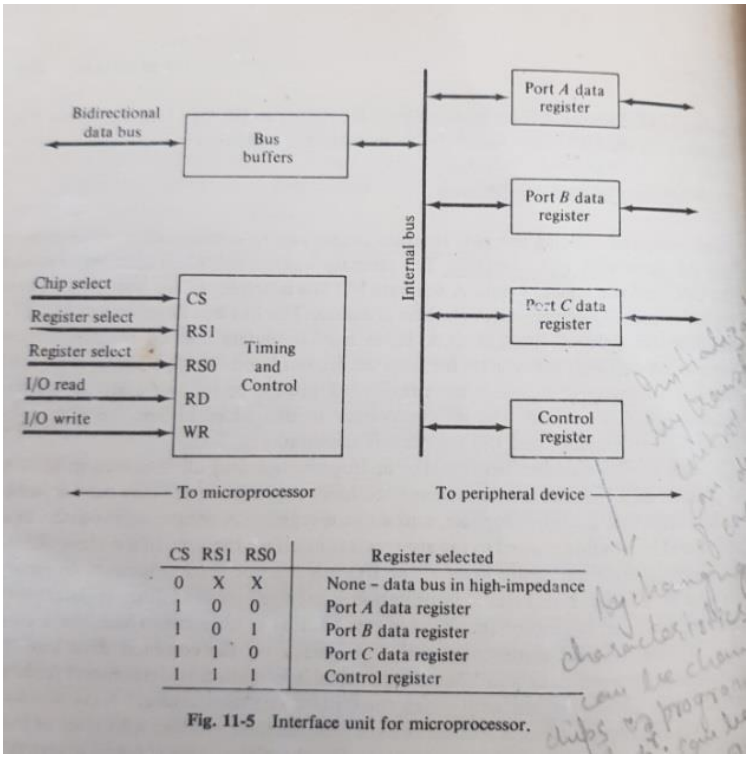
CYCLE STEALING: During the DMA operation, when the DMA is transferring the data from memory at that time if the CPU wants the memory transfer than it disable the DMA Grant signal. On receiving DMA Grant low the DMAC should disable the DMA Request but before disabling DMA Request the DMAC may transfer few bits. This is called Cycle Stealing.

In another concept of Cycle Stealing may transfer some bytes sensing the CPU not accessing the memory.

MICROPROCESSOR INTERFACE

Some computer systems use two separate buses, one to communicate with memory and the other with I/O interfaces. The memory bus is used to communicate between the CPU and the memory unit. A separate I/O bus is employed for transfers between the CPU and I/O devices through their interface.

A typical example of an interface unit for a microprocessor is shown in the fig.



CS	RS1	RS0	Register selected
0	X	X	None – data bus in high-impedance
1	0	0	Port *A* data register
1	0	1	Port *B* data register
1	1	0	Port *C* data register
1	1	1	Control register

Fig. 11-5 Interface unit for microprocessor.

It consists of three data registers called *ports*, a control register, a tri-state buffers for the data bus, and timing and control circuits.

The interface registers communicate with the microprocessor through the data bus. The address bus selects the interface unit through the chip select and the two register lines. The two register select inputs RS1 and RS0 select one of the four registers in the interface. The content of the selected register is transferred into the microprocessor via the data bus when the I/O read input is enabled. The microprocessor transfers binary information into the selected register via the data bus when the I/O write input is enabled. The peripherals unit attached to the interface communicates with the data in each of the three ports.

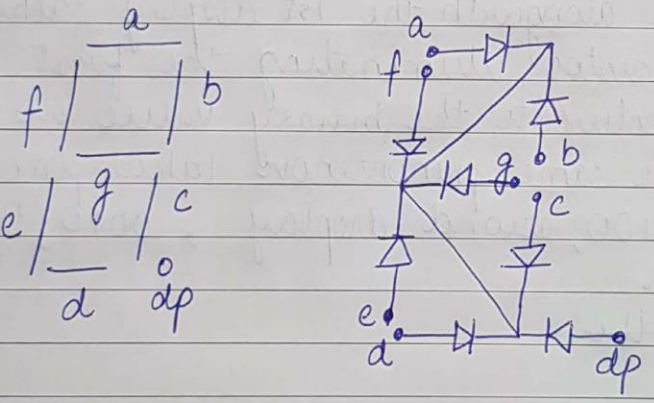
Port A can be assigned the task of transferring data to and from the CPU and the I/O device. Port B can be used to specify a command for the peripheral unit and Port C could function as a status register which accumulates status bit conditions to be read by the CPU at a later time.

A 7-segment LED type displays, provide a very convenient way of displaying information or digital data in the form of numbers, letters or even alpha-numeric characters.

Typically 7-segment displays consists of seven individual LEDS

The common cathode display

In the common cathode display, all the cathode connections of the LEDS are joined together to logic "0" or the ground. The individual segments are illuminated by application of a "HIGH", logic "1" signal to the individual ~~anno~~ anode terminals.



common cathode display

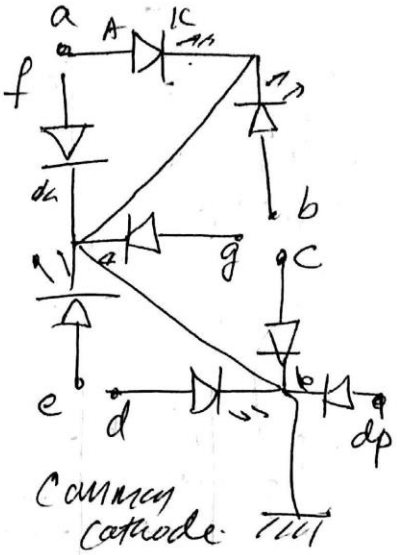
a	b	c	d	e	f	g
1	1	1	1	1	1	0
0	1	1	0	0	0	0
1	1	0	1	1	0	1

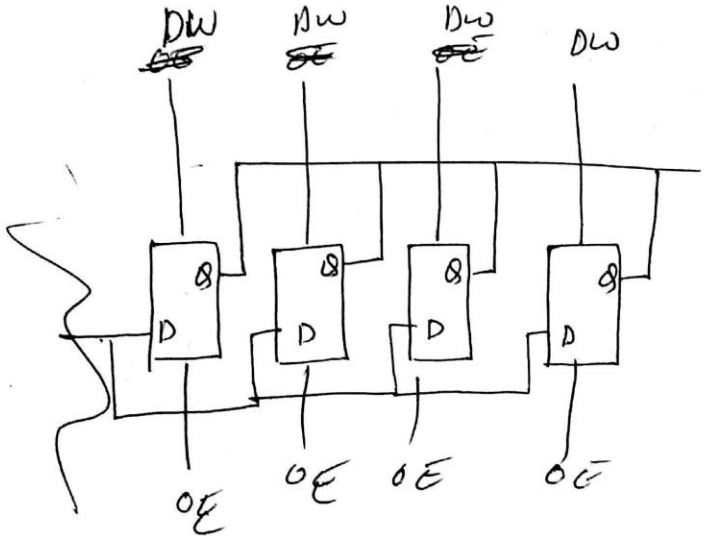
so on

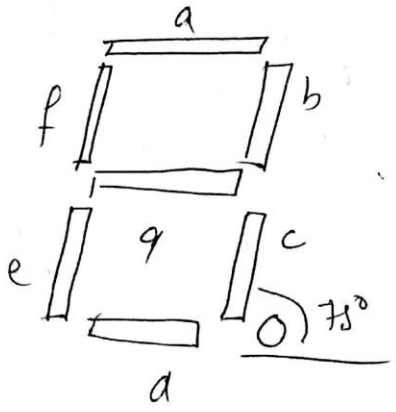
a b c d e f g dp

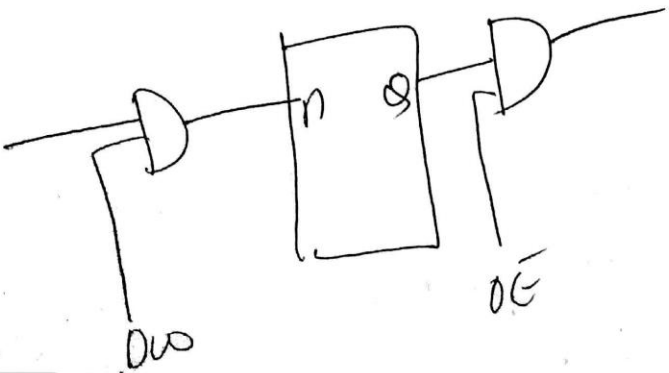
1 1 1 1 1 1 0 0

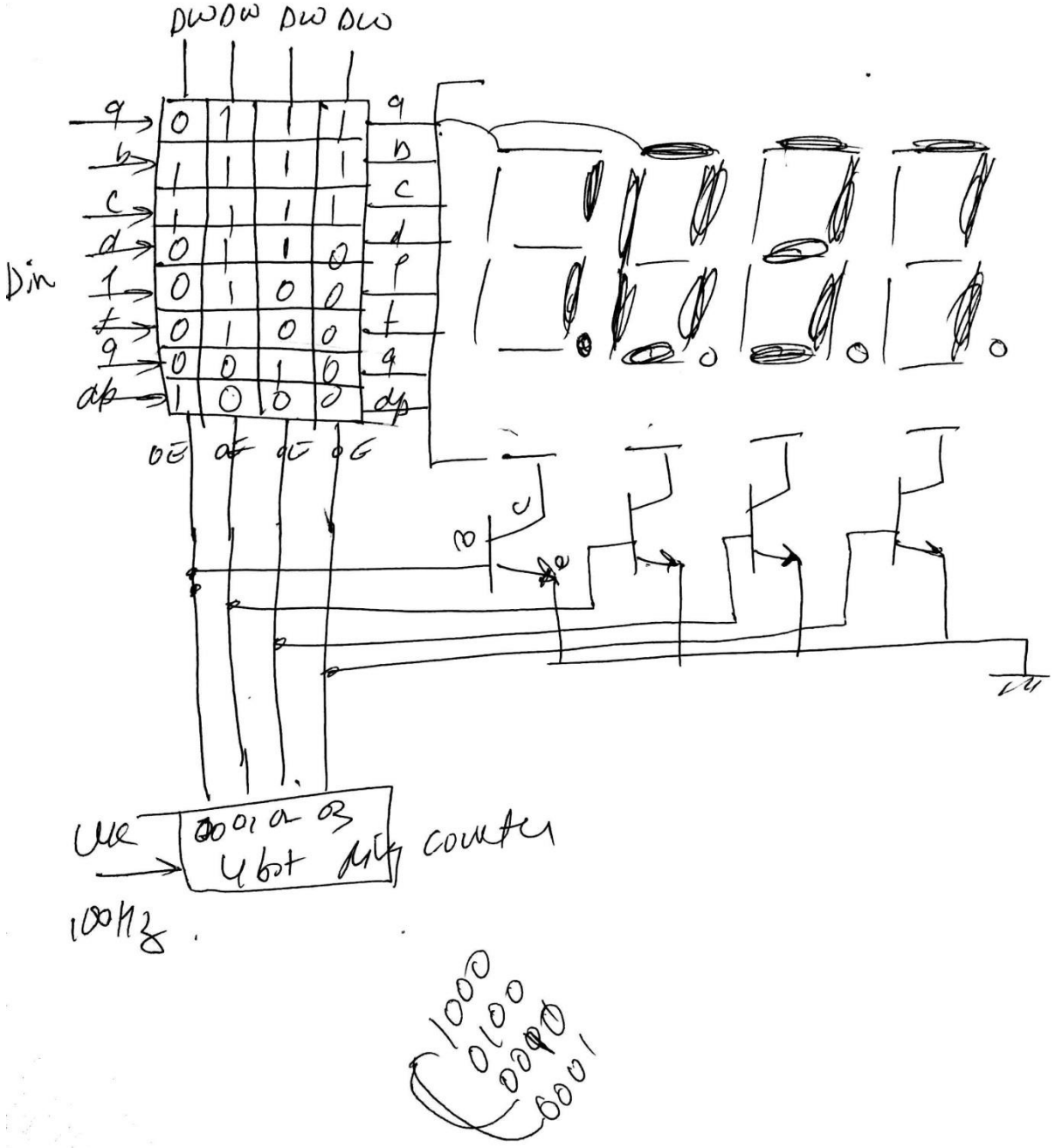
0 1 1 0 0 0 0 0 0











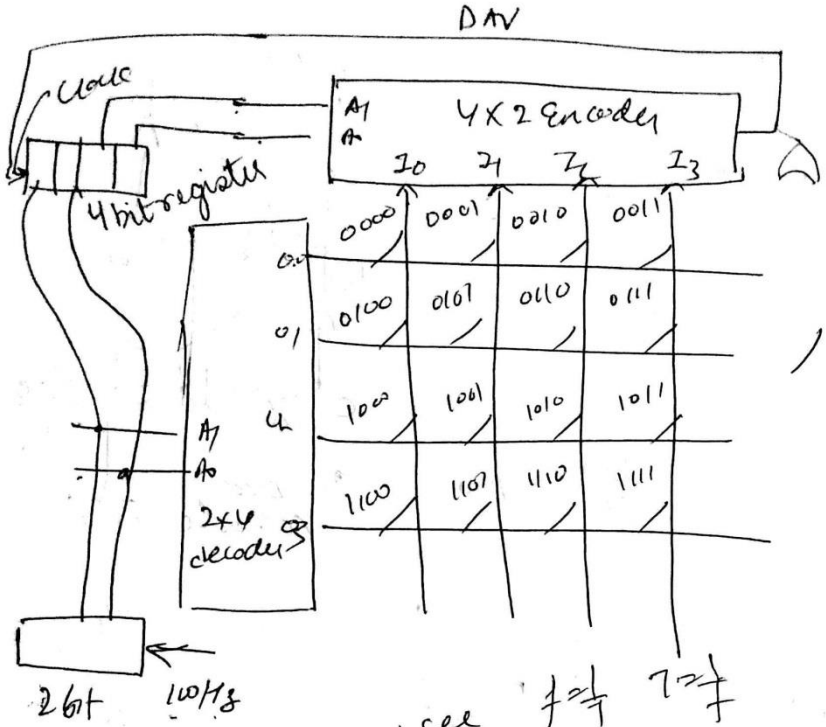
fig

The diagram shows 4 LED 7 segment displays connected to 4 8-bit registers respectively. ~~Those registers are hold the four bit BCD~~

The display works on the principle of time-scan multiplexing accomplished by continuous scanning and digit formation on individual display in such a manner that it crosses the ~~pers~~ persistance of vision and thus displays a stable visual output.

The 8-bit parallel register values are output to respective display one-by-one with the help of a 4-bit ring counter operating at atleast 100Hz. The first value 1000 enables the first 8-bit register alongwith the 1st display cathode through transistor illuminating the first segment respective to the binary value in the register. The same phenomenon takes place for the 0100; second display, 0010; third display and so on.

~~Since the~~



1Hz = 1 sec

10Hz = .1sec

100Hz = .01sec

= .01 X 1000 ms

= 10 ms

100Hz = 100 clocks / sec

1sec → 100 clock

x → 1 clock

$x = \frac{1}{100} = .01$sec

1sec = 1000ms

x = .01 X 1000 ms

= 10 ms

$T = \frac{1}{100}$

= .01sec

1sec = 1000 ms

= .01 X 1000

= 10 ms

50ms

100Hz

1s = 100 clock

25

The hex keyboard is a peripheral for input. It has 16 buttons in a 4 by 4 grid, labelled with hexadecimal digits 0 to F.

Internally the structure of hexkeyboard consists of wires that run in vertical columns connected to $I_0 - I_3$ of the 4X2 encoder and in horizontal rows connected to $O_0 - O_3$ of the 2X4 decoder. Each key on the keypad is essentially a switch that connects a row wire to column wire. When a key is pressed it makes an electrical connection between the row and column.

The 2X4 decoder is driven by a 100 Hz clock that varies the inputs $A_1 A_0$ from 00 to 11 thus enabling one of the O_i at a time. ~~$A_1 A_0$~~ corresponding to the key press the respective I_i of the encoder will enable $A_2 A_3$ with values 00 to 11.

$A_1 A_0 A_2 A_3$ are fed into a 4 bit buffer register that is latched by

the encoder via a DAV signal, such that as soon as $A_2 A_3$ is transferred to the buffer DAV signal is enabled that latches the input ~~to the system~~ (the key press value) to the system.

Microprogram Control Memory

The function of a control unit in a digital system is to initiate sequence of micro-operations. The number of different micro-operations available in a given system is finite. The principle of microprogramming is an elegant and systematic method for generating the micro-operation sequences in a digital system.

The control unit initiates a series of sequential steps of micro-operations. A control unit whose micro-operation steps are stored in a memory is called a micro-programmed control unit. Each control word of memory is called a microinstruction and a sequence of words is called a microprogram. The utilization of a microprogram involves placing all micro-operations steps in words of ROM for use by the control unit through successive read operations. The content of the words in ROM are fixed and cannot be altered by simple programming since no writing capability is available in ROM.ROM words are made permanent during the hardware production of the unit.

A computer that employs a microprogrammed control unit will have two separate memories: a main memory and a control memory. The main memory is available to the user for storing his programs. The contents of main may alter when the program is executed and whenever the program is modified. The user's program in main memory consists of machine instruction and data. The control memory holds a fixed microprogram that cannot be altered by the occasional user. The microprogram consists of microinstructions that specify various internal control signals for execution of register micro-operations. Each machine instruction initiates a series of micro-operations in control memory. These microinstructions generate the micro-operations to

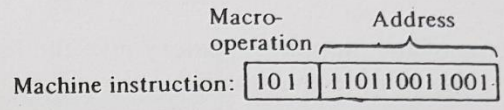
1)fetch the instruction from main memory

2)evaluate the effective address of the operand

3)execute the operation specified by the instruction

4)return control to the beginning of the fetch cycle to repeat the sequence again for next instruction

MAPPING OF MACRO-OPERATION



Mapping bits: 0XXXX00

Microinstruction address: 0101100

Fig. 8-4 A mapping process from macro-operation to a microinstruction address.

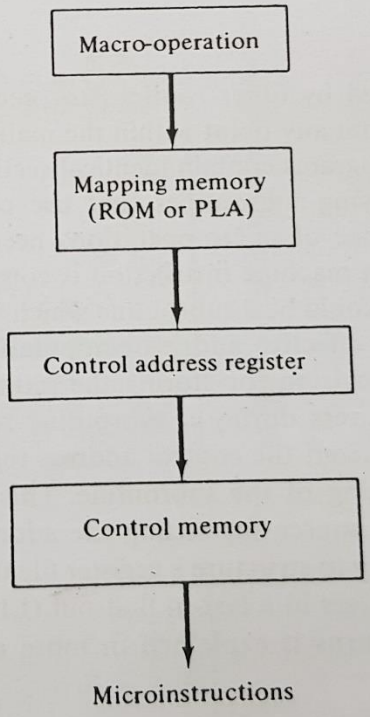
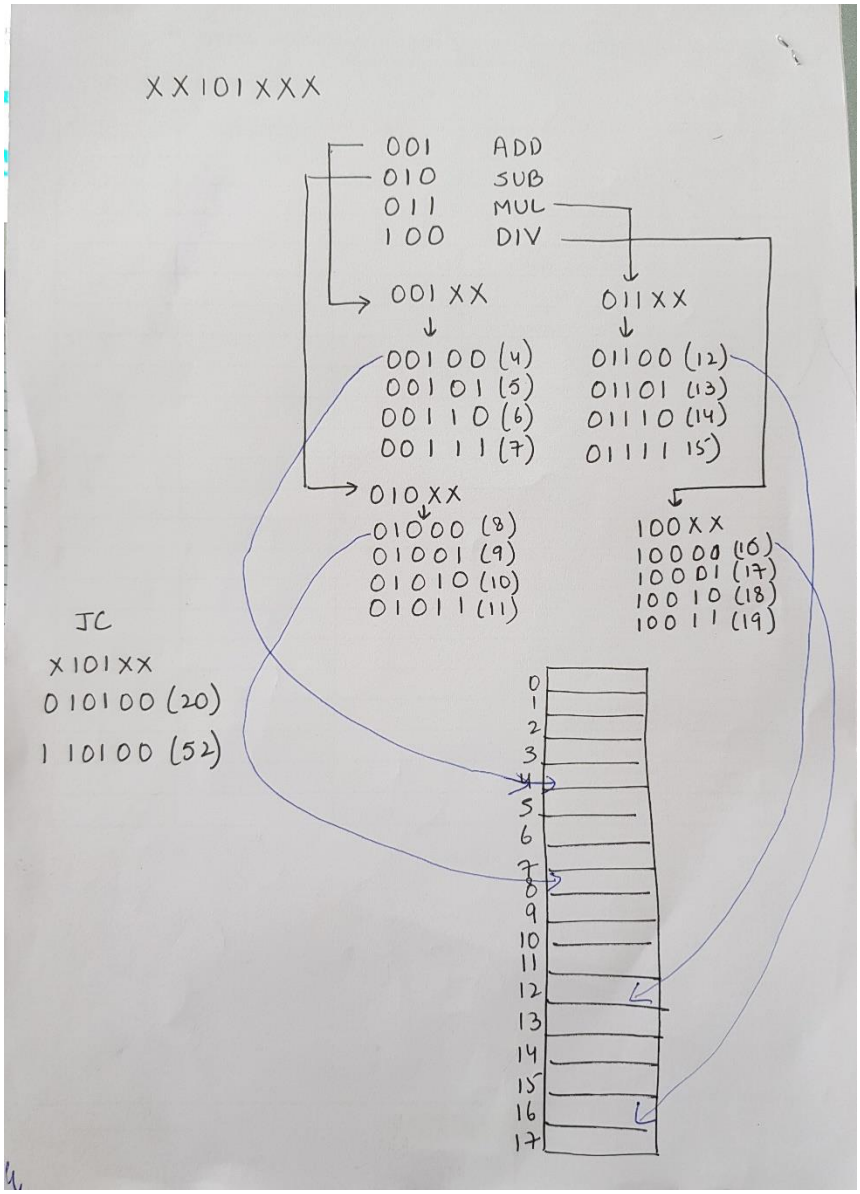


Fig. 8-5 Mapping memory for transforming a macro-operation to a control memory address.



A computer with a simple instruction format as shown in fig 8-4 has an operation code of four bits which can specify up to 16 distinct macro-operations. Assume further that the control memory has 128 words requiring an address of 7 bits. For each macro-operation there exists a routine in control memory that executes the macro-operation. One specific mapping process that converts the 4-bit operation code to a 7-bit address is shown in fig 8-4. The mapping of a 4-bit macro-instruction to a 7-bit micro-instruction address

0	**XXXX**	**00**
0	**0000**	**00**
		01
		10
		11
0	**0001**	**00**
		01
		10
		11

0	**0010**	**00**
		01
		10
		11
	……	
	…….	
0	**1111**	**00**
		01
		10
		11
1	**0000**	**00**
		01
		10
		11
	……	
1	**1111**	**00**
		01
		10
		11

Thus, from the table it is clear that a 4-bit macro-operation will be mapped to a 7-bit micro-instruction. Each macro-operation will contain 4-micro-operations. And the 0 or 1 at the MSB bit maps the micro for branching and control. Let us take an example of a macro mapping to a micro.

Let us assume a macro-operation 1011, it is mapped as 0101100, thus the micro-operations written for this instruction in the control memory are

0101100

0101101

0101110

0101111

1101100

1101101

1101110

1101111

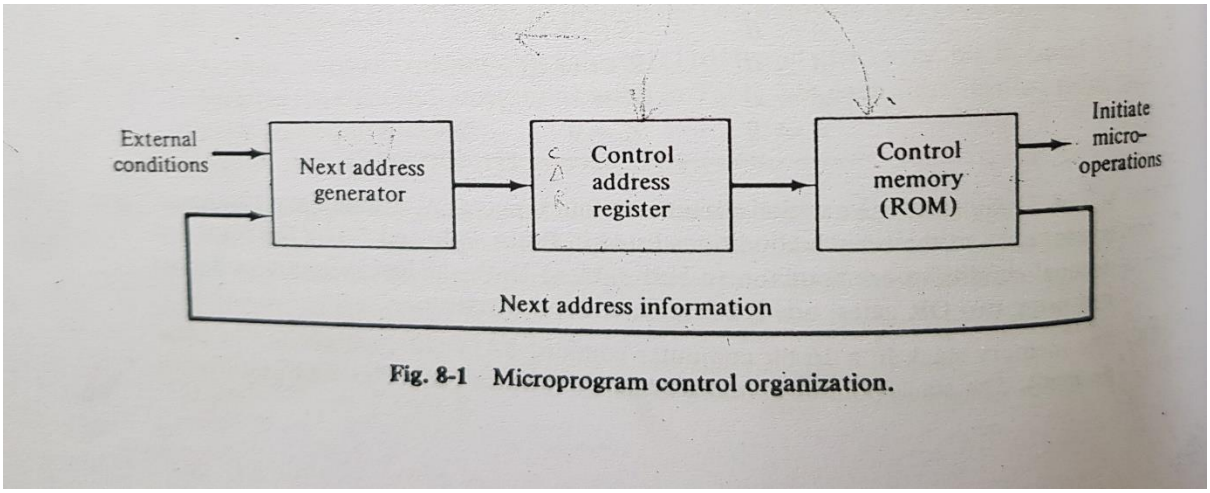


Fig. 8-1 Microprogram control organization.

Fig 8-1 illustrates the general configuration of the microprogrammed control unit. The control memory is assumed to be a ROM, within which all control information is permanently stored. Control address register (CAR) specifies the word read from control memory. The word corresponding to the address in CAR is generated as output.

The word read from the control memory represents a microinstruction. The micro-instruction specifies one or more micro-operation. When these micro-operations are executed the control, memory has to determine its next address. The location of the next micro-instruction may be one next in sequence or it may be located somewhere else in the control memory. Thus, it is necessary to use some bits of the microinstruction to control the generation of the address for the next microinstruction. So, a microinstruction contains bits for controlling the micro-operations in the digital system and bits that determine the address sequence of the control memory itself.

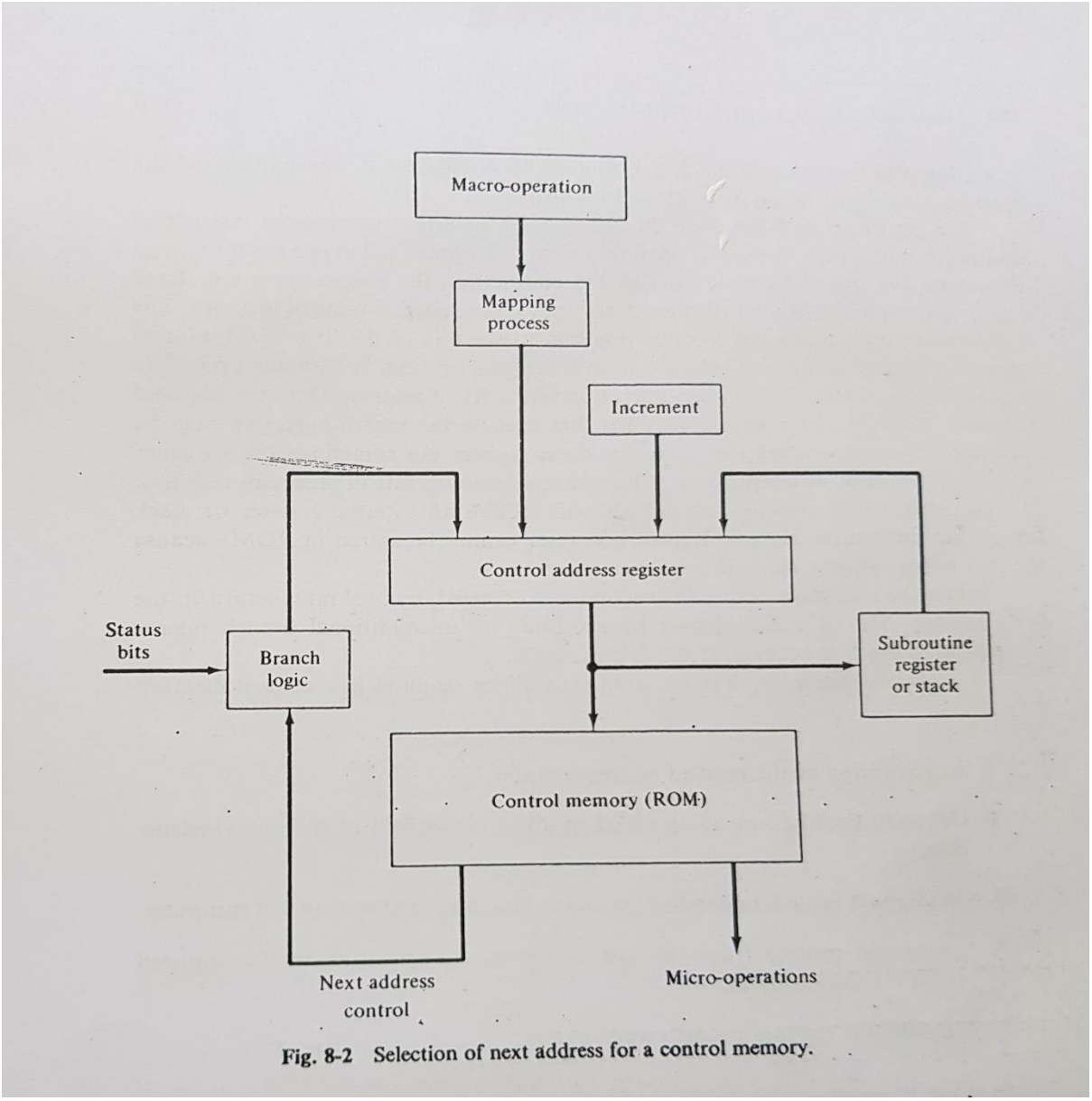


Fig. 8-2 Selection of next address for a control memory.

Fig8-2 shows a block diagram of a control memory and the associated hardware needed for selecting the next microinstruction address.

The macroinstruction is mapped into a micro-operation address which is passed onto the control address register whereby this address can be the address of the micro-instruction in control memory or it can be an address of a subroutine or a stack address value. The address resolution is done and the corresponding micro-operation is generated. The specifics bits in the micro-instruction are used to generate the next address where the next address can be the one next in sequence or someplace else in the control memory. In all the cases the address will be resolved and address of the next micro-operation will be generated.

In summary, the address sequencing capabilities required in a control memory are:

1. Incrementing the control address register.
2. Unconditional branch as specified by the address field of the micro-instruction.
3. Conditional branch depending on status bits in the register of the computer.
4. A mapping process from the bits of the macro-operations to the required bits of a ROM address.
5. A facility for subroutine calls and returns.

CONDITIONAL BRANCH

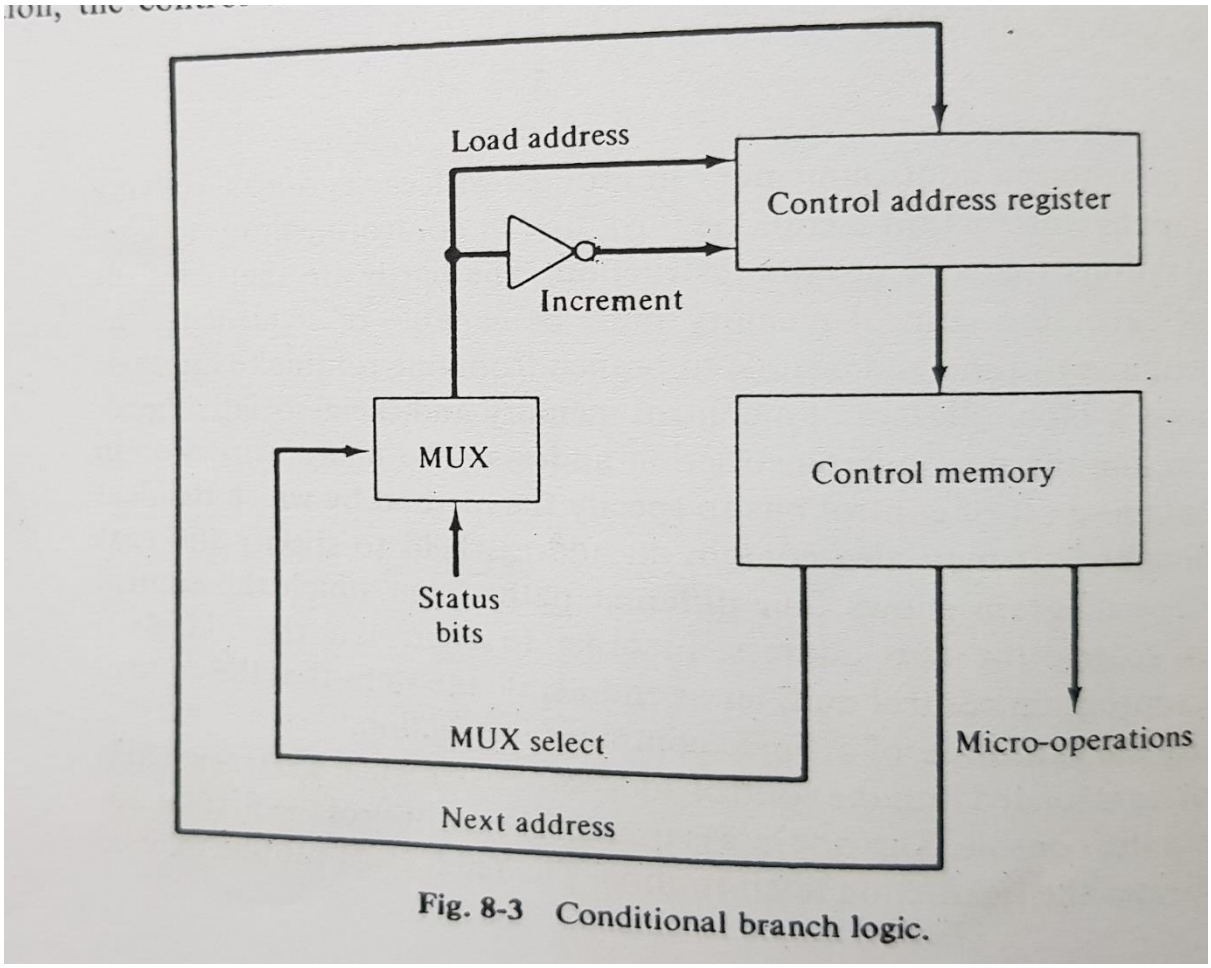


Fig. 8-3 Conditional branch logic.

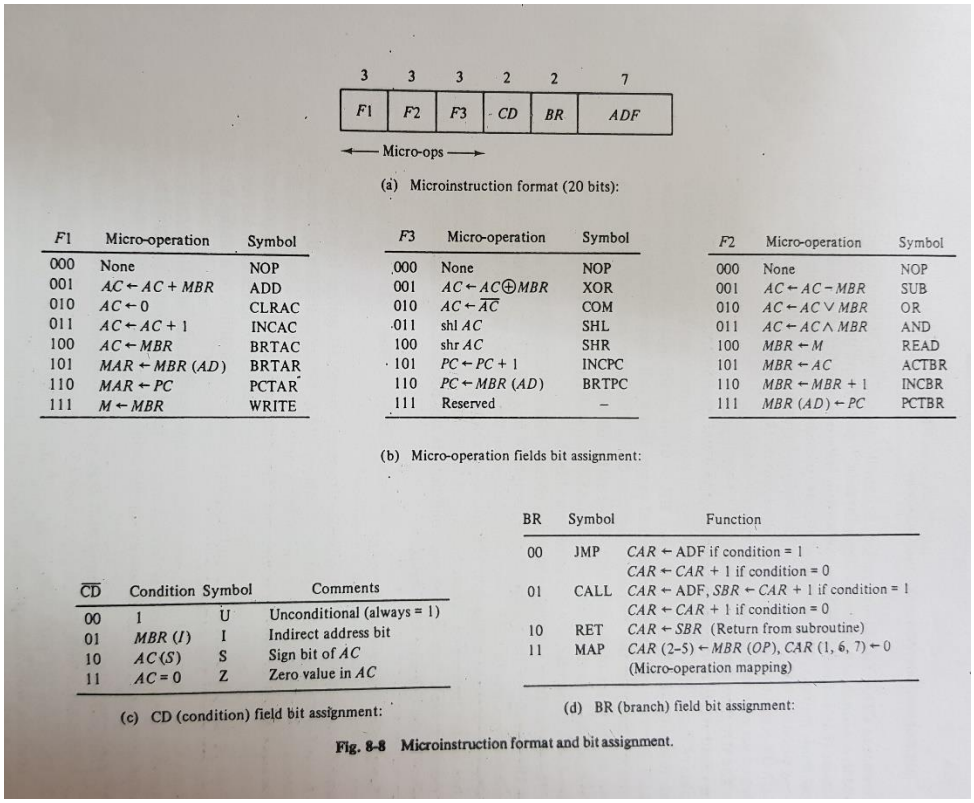
The branch logic in fig provides decision making capabilities in the control unit. The status bits overflow indication, adder carry out, sign bit of a register, the mode bit of an instruction, and the input/output status conditions. Information in these bits can be tested and actions can be initiated based on their conditions.

Suppose that we have eight status bits to consider. Three bits in the microinstruction are used to specify any one of the eight conditions. These three bits provide the selection variables for the multiplexer. If the selected status bit is a state 1, the output of multiplexer is 1 otherwise 0.A 1 output in the multiplexer generates a control signal to load the address bits of the microinstruction into the address register. A 0 output in the multiplexer causes the address register to be incremented by 1.

MICROINSTRUCTION FORMAT

The micro-instruction format for the control memory is shown in fig 8-8(A). The 20-bits of the microinstruction are divided into four functional fields. The micro-ops field specify micro-operations for the computer. The CD field selects status bit conditions, the BR field specifies the type of branch, and the ADF field contains an address. The address field is seven bits long since the control memory has $128=2^7$ words.

The micro-ops field is subdivided into three fields F1, F2, and F3, of three bits each. The three bits in each field are encoded to specify seven distinct micro-operations. This gives a total of 21 micro-operations. No more than three micro-operations can be chosen for a micro-instruction, one from each field. If less than three micro-operations are used, one or more of the fields will use the binary code 000 for no operation.



3	3	3	2	2	7
F1	F2	F3	CD	BR	ADF

← Micro-ops →

(a) Microinstruction format (20 bits):

F1	Micro-operation	Symbol
000	None	NOP
001	$AC \leftarrow AC + MBR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow MBR$	BRTAC
101	$MAR \leftarrow MBR\ (AD)$	BRTAR
110	$MAR \leftarrow PC$	PCTAR
111	$M \leftarrow MBR$	WRITE

F3	Micro-operation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus MBR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	shl AC	SHL
100	shr AC	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow MBR\ (AD)$	BRTPC
111	Reserved	–

F2	Micro-operation	Symbol
000	None	NOP
001	$AC \leftarrow AC - MBR$	SUB
010	$AC \leftarrow AC \vee MBR$	OR
011	$AC \leftarrow AC \wedge MBR$	AND
100	$MBR \leftarrow M$	READ
101	$MBR \leftarrow AC$	ACTBR
110	$MBR \leftarrow MBR + 1$	INCBR
111	$MBR\ (AD) \leftarrow PC$	PCTBR

(b) Micro-operation fields bit assignment:

$\overline{CD}$	Condition	Symbol	Comments
00	1	U	Unconditional (always = 1)
01	$MBR\ (I)$	I	Indirect address bit
10	$AC\ (S)$	S	Sign bit of AC
11	$AC = 0$	Z	Zero value in AC

(c) CD (condition) field bit assignment:

BR	Symbol	Function
00	JMP	$CAR \leftarrow$ ADF if condition = 1 $CAR \leftarrow CAR + 1$ if condition = 0
01	CALL	$CAR \leftarrow$ ADF, $SBR \leftarrow CAR + 1$ if condition = 1 $CAR \leftarrow CAR + 1$ if condition = 0
10	RET	$CAR \leftarrow SBR$ (Return from subroutine)
11	MAP	$CAR\ (2\text{-}5) \leftarrow MBR\ (OP)$, $CAR\ (1, 6, 7) \leftarrow 0$ (Micro-operation mapping)

(d) BR (branch) field bit assignment:

Fig. 8-8 Microinstruction format and bit assignment.

The bits of a microinstruction are usually divided into two parts called fields, with each field defining a distinct, separate function. The various fields in the microinstruction format provide following functions:

1. A control word that initiates micro-operations in the system. The bits of the control word are sometimes divided into separate fields.
2. Control and address bits to specify the evaluation of the next micro instruction address.
3. Other special fields for transferring data to a given destination.

MICROPROGRAM SEQUENCER

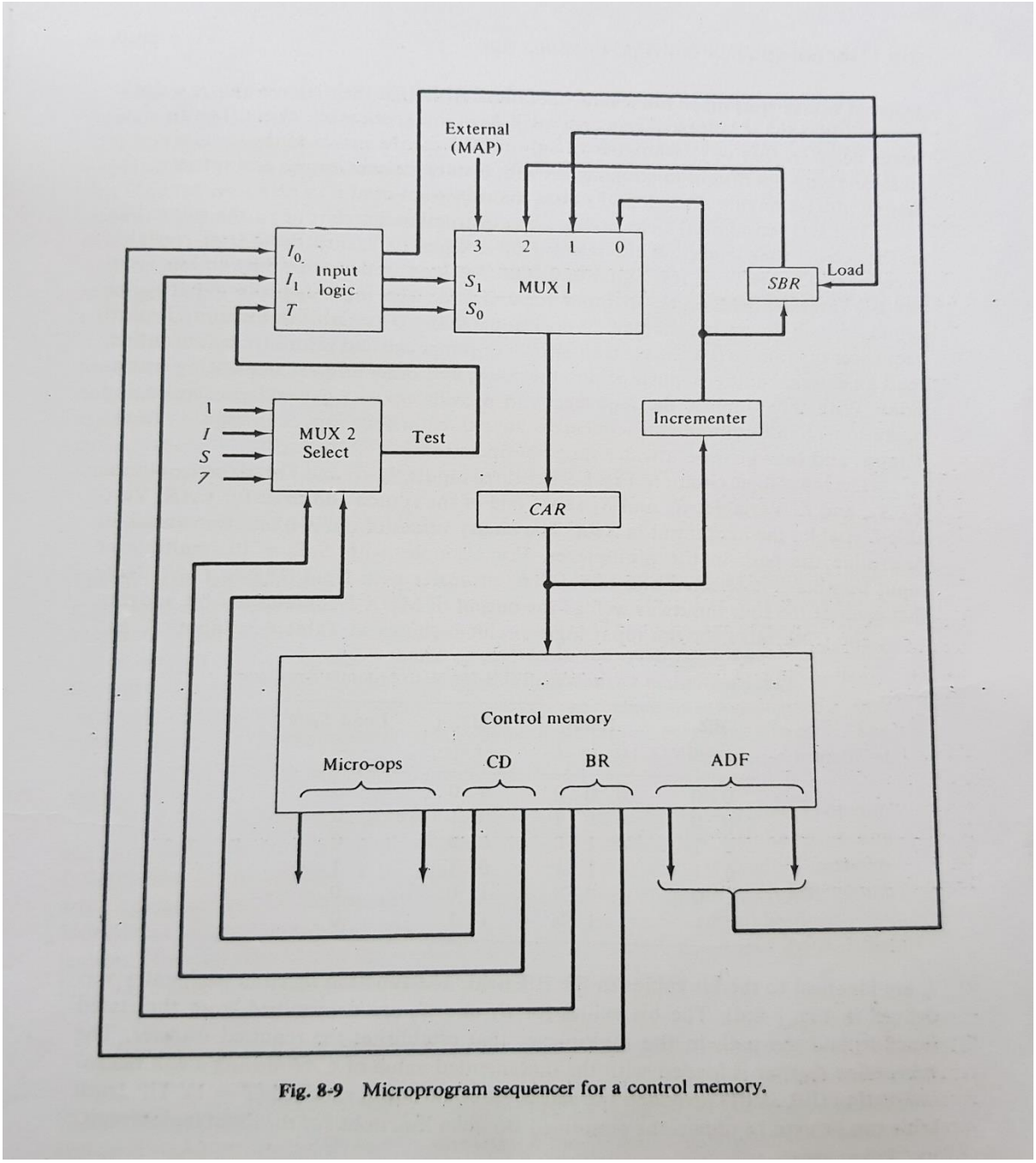


Fig. 8-9 Microprogram sequencer for a control memory.

The block diagram shows interconnection between the sequencer and control memory attached to it. There are two multiplexers in the circuit. The first multiplexer selects an address from one of four sources and routes it into a control address register (CAR). The second multiplexer tests the value of a selected status bit and the result of the test is applied to an input logic circuit. The output from CAR provides the address for the control memory. The content of CAR is incremented and applied to one of the multiplexers inputs and to the subroutine register SBR. The other three inputs to multiplexer number 1 come from the address field of the present micro-instruction, from the output of SBR, and from an external source that maps the macro-operation.

The (CD) condition field of the microinstruction selects one of the status bits in the second multiplexer. If the selected bit is equal to 1, then the T(test) variable is equal to

1; otherwise, it is equal to 0. The T value together with the two bits from the BR field go to an input logic circuit. The input logic in a sequencer determines the type of operations that are available in the unit like for eg increment, branch or jump, call and return from subroutine, load an external address, push pop over stack and other sequencing operations. With three inputs, the sequencer can provide up to eight address sequencing operations.

The input logic circuit has three inputs I_0, I_1, and T, and three outputs, S_0, S_1 and L. S_0 and S_1 select one of the source addresses for CAR. Variable L enables load input in SBR. For example, if S_1S_0=10, multiplexer input no. 2 is selected and establishes a transfer path from SBR to CAR.

$S_1=I_1$

$S_0=I_1I_0+I_1'T$

$L=I_1'I_0T$

8085 Micro – Processor

A digital circuit made up with the help of micron technology that executes micro operations in micro seconds is called a microprocessor.

A digital circuit that performs arithmetic, logic and other operation over the data in accordance in the instruction is called a microprocessor.

NOTE: Every CPU is essentially a microprocessor but every microprocessor is not a CPU.

4004 (First Microprocessor) is 4 – bit microprocessor (Only +, -, AND, OR & NOT could be performed).

8008 is 8 – bit microprocessor it can handle more data and had greater computational power.

8085 is generic microprocessor (Approximately 1982).

Generic – from this processor generation was started.

8080 – 16 – bit computational possible internally but externally 8 – bit microprocessor.

8086 – True 16 – bit microprocessor.

80186 – used only for industrial purpose.

80286 – 16 bit microprocessor.

8087 – Co-processor or math processor used with 8086.

80186 – 80187

80286 – 80287

80386 – It is first 32 bit processor came around 1885.

[*80386 + 80387 + cache*] = **80386 DX**

80386 SX – Had compatibility with peripherals internally works as 32 – bit and externally 16 – bit processor, speed is 25 MHz.

80386 DX2 – 50166 MHz

80386 DX4 – 100 MHz (10^8 cycles / sec.)

80486 – Has multimedia technology (was incorporated)

Pentium

Pentium Pro

Pentium MMS

Pentium Xeon

Pentium III

Pentium IV

Itanium – First 64 – bit processor, 10^9 GHz

8085 is an 8 bit generic microprocessor produced by INTEL corporation using N – MOS technology and is available in 40 pin DIP (Dual In-line Package) package.

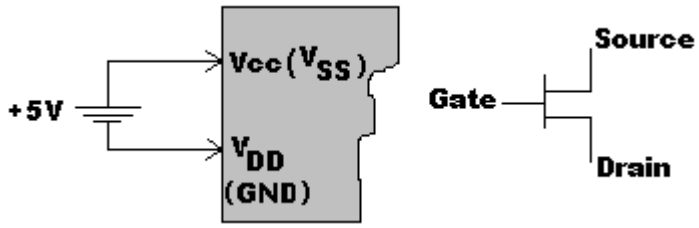
It has an operating frequency ranging from 2 MHz to 10 MHz.

The microprocessor has 16 bit addressing capacity producing 64 K distinct addresses and 8 bit data capacity where address and data buses are multiplexed in nature. The operating voltage of 8085 is +5 volts. The clock generator is inbuilt, thus requires a crystal or a tank circuit for clock generation. The microprocessor supports PMA serial communication, memory I / O mappings etc.

<u>The description of 40 pins is as;</u>

(1) POWER SUPPLY : VCC, GROUND

These operating voltage, +5V to the microprocessor are applied between pins VSS and VDD. The operating current of this microprocessor is around 500 miliampere

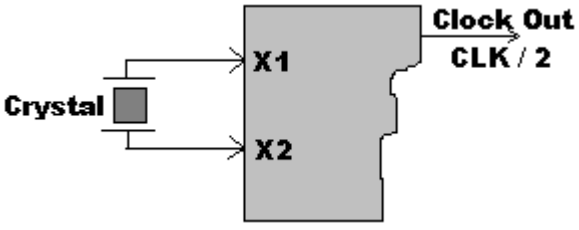


(2) CLOCK GENERATOR : X1, X2, CLOCK OUT

Periodic variation of voltage between finite times is called ***CLOCK***.

The operating clock to the microprocessor is generated by an inbuilt oscillator associated with clock pins X1 and X2 a relevant crystal with a frequency range 2 – 10 MHz can be connected between these two points providing the working clock to the microprocessor.

The operating clock of the microprocessor is available at CLOCK OUT pin. This pin can be used to provide operating clock to connected devices such as memory and I / O's, so that all the system can work synchronously. Internally the clock is divided by 2 for thermal stability and the same clock is available at clock output pin.



(3) READ / WRITE CONTROLS : $\overline{RD}$, $\overline{WR}$

These are low enable output pins that signifies a read or write operation to be carried out over input / output devices or the memory.

For example – If $\overline{RD}$ is low that means microprocessor has requested a read operation.

Table for read write operation –

$\overline{RD}$	$\overline{WR}$	Operation
0	0	Not Valid
0	1	Read
1	0	Write
1	1	Nothing

IO / $\overline{M}$	$\overline{RD}$	$\overline{WR}$	Operation
0	0	1	Read from memory
1	0	1	Read from some I/O device
0	1	0	Wants to write to memory
1	1	0	Wants to write something to I/O
0	1	1	Neither wants to neither read nor write.

(4) STATUS CONTROL : IO / M, S1, $\overline{S0}$

The IO/$\overline{M}$ pin is used to designate an Input / Output operation or a memory operation into consideration that is if this pin is high then an Input / Output operation is carried out by microprocessor and if this pin is low then a memory operation is carried out.

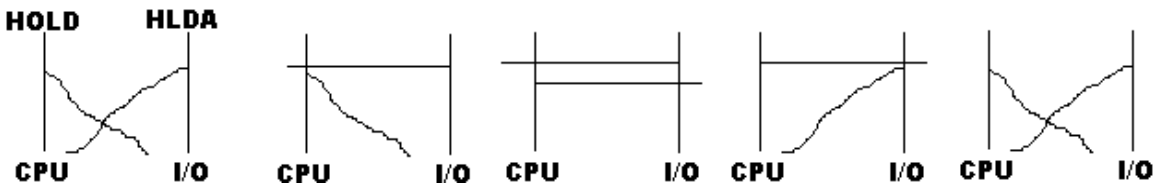
S1, S0 pins are called status pins, acting as output pins provides the current machine cycle status of the microprocessor.

(5) DIRECT MEMORY ACCESS (DMA) : HOLD, HLDA

DMA is process by which the input / output devices can interact directly with the memory without intervention but under the control of CPU.

HOLD pin is an input pin acting as DMA request where as HLDA is an output pins acting as DMA grant signal. These pins are called handshaking pins.

Initially both the pins are zero. When an I/O device wants DMA transfer it sends high logic over HOLD pin, on sending it the microprocessor may grant HLDA signal (on accomplishing current machine cycle). When HLDA is granted, the microprocessor is tri stated, the system bus can be now taken over by I/O device for memory transfer. On DMA finish the HOLD pin is made low in response the microprocessor makes HLDA low and comes out of tri – state resuming normal.



(6) READY :

This is an input pin that is used to establish synchronization between slower Input / Output devices and faster microprocessor. When a I / O devices wants to transfer microprocessor in ideal mode that it disables this pin by it low and consequently the microprocessor is driven in no operation state. When this pin is made high, the microprocessor resumes execution.

(7) SERIAL CONTROL : SID (Serial In Data), SOD (Serial Out Data)

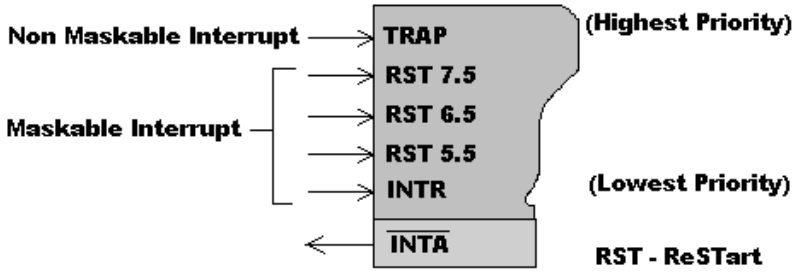
The bit by bit communication i.e. the serial communication of the content of the system can be done using the SID and SOD pins of the microprocessor.

The SID or Serial In Data is the input pin that is used to read external data bit by bit serially. The read data is transferred to accumulator.

The SOD pin is an output pin that provides serial data bit by bit of the accumulator to external devices.

The communication in 8085 follows a byte transmission with LSB first.

INTERRUPT : TRAP, RST 7.5, RST 6.5, RST 5.5, INTR, INTA



Mask able Interrupt – Those interrupt which can be disabled. (RST 7.5, RST 6.5, RST 5.5, INTR).

Non Mask able Interrupt – Which can not be disabled. (TRAP (Highest Priority)).

Table of Hardware and Software Interrupt –

Hardware Interrupt –

Interrupt	Vector
TRAP	0024 H
RST 7.5	003C H
RST 6.5	0034 H
RST 5.5	002C H

Software Interrupt –

Interrupt	Vector
RST 0	0000 H
RST 1	0008 H
RST 2	0010 H
RST 3	0018 H
RST 4	0020 H
RST 5	0028 H
RST 6	0030 H
RST 7	0038 H

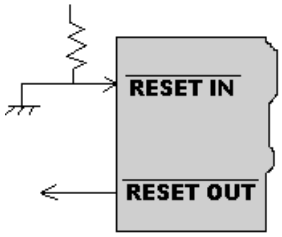
INTR – can generate 8 – independent interrupt.

The microprocessor comprises of 5 hardware interrupt that can be signalled externally by the connected device to accomplish a specific task. The interrupts are named TRAP, a non-maskable interrupt with highest priority, RST 7.5, RST 6.5, RST 5.5 and INTR.

INTR is the lowest priority interrupt and is vectored interrupt i.e. when this interrupt occurs, the microprocessor enables INTR and in response the connected device can send a binary value ranging from 0 – 7 with respect to which a software interrupt is carried out. If more then one simultaneous interrupt occur then the internal priority encoder unit resolves the priority and accordingly executes the highest priority interrupt.

The first five pins are input pins where as INTA is output pin with the TRAP having highest priority and non mask able in nature and for the rest pins the priority decreases in nature and are mask able in nature. The interrupts are encloses through INTA.

(8) RESET : RESET IN, RESET OUT



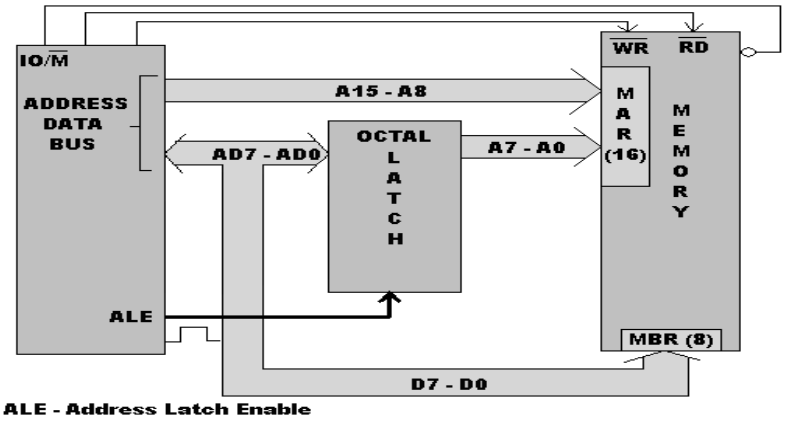
The microprocessor can be made reset using reset control. It comprises of two pins labelled RESET IN and RESET OUT. When RESET goes low for at least 10 clocks and then goes high, the microprocessor is initialized to some predefined condition. In

continuation RESET OUT pin goes high, generating a stroke which is used to reset the connected peripherals.

(9) ADDRESS / DATA BUS, ALE :

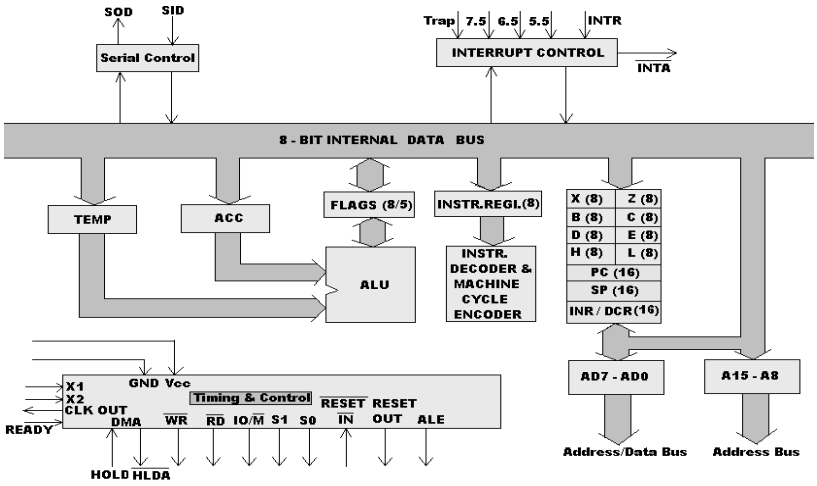
The microprocessor supports 16 – bit addressing along with 8 – bit data bus. To perform address data bus operation we require 24 – bits but we have a theory that address data requirements are mutually exclusive and hence 16 wires suffice. The purpose to accomplish the movement of data and address over 16 wires we use address data multiplexing which is achieved through a separate pin called ALE, the Address Latch Enable. A separate octal latch is also required.

The address and address data bus are designated as A15 – A8 and AD7 – AD0 respectively. The upper byte of bus (A15 – A8) is connected directly to memory whereas lower bus is connected through a latch diverging the bus to MAR and MBR of the memory. The microprocessor generates 16 – bit address, after settle time enables ALE and consequently the 16 – bit address is available to memory since the lower address is latched, The AD bus is free for movement of data which is accomplished using read write signals.

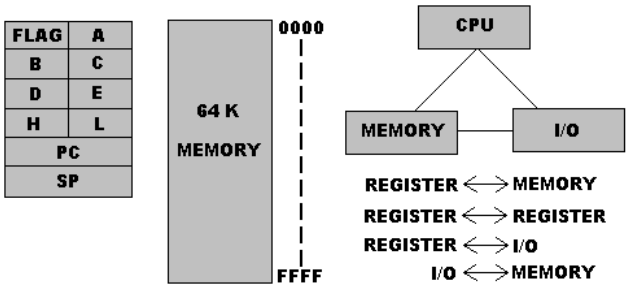


ALE - Address Latch Enable

Block Diagram of 8085 Micro – Processor



Programming (Software) Model of 8085



MNEMONICS: The abbreviated form of instruction is known as Mnemonics.

OPCODE: Operation Code value of the instruction.

ASSEMBLER: Assembler is the translator that converts a Mnemonics instruction to Opcode.

The instruction set of 8085 is divided into five groups. Each group is dedicated for a specific set of instructions.

The groups are –

(A) Data transfer group
(B) Arithmetic group
(C) Logical group
(D) Branching group
(E) Machine control group

(A) Data transfer group – This group comprises of the instructions related with movement of data between memory and registers.

(1) MOV R1, R2 : R1 ← R2

OR

MOV D, S

R1, R2 $\in$ { R }

R = {A, B, C, D, E, H, L}

It copies the content of register R2 to R1 retaining the value of R2.

After execution of this instruction, the content of R1 is over – written by the content of R2.

Suppose B = 63 H

C = A5 H

MOV B, C [B = A5 H, C = A5 H]

MOV C, B [B = 63 H, C = 63 H]

NOTE: There are 49 variants of this instruction with the help of which the data of any GPR can be transferred to any other.

(2) MOV R, M : R ← $[M]_{HL}$

NOTE: The memory in 8085 system is designated by M. by default or until and otherwise specified, the address of memory is given by HL pair i.e. M or M_{HL} are same.

It moves the content of memory location M whose address is specified by HL pair to register R.

NOTE: There are 7 variants of this instruction.

(3) MOV M, R : $[M]_{HL}$ ← R

It copies the content of register R to the memory location M whose address is specified by HL pair.

NOTE: There are 7 variants of this instruction.

(4) MVI R, DATA : R ← $DATA_8$

It immediately moves the data part of instruction to data part of instruction to specified register R.

NOTE: There are 7 variants.

Let A = 05 H

MVI A, 62 H

A = 62 H

Program – Copy the content of memory location 5801 to 5902.

```
MVI H, 58
MVI L, 01
MOV B, M
MVI H, 59
MVI L, 02
MOV M, B
```

Program – WAP to write a data value 20 at location 5901.

```
MVI A, 20
MVI H, 59
MVI L, 01
MOV M, A
```

Program – Swap the content of two consecutive locations X and X+1.

Let X = 5800, X + 1 = 5801

```
MVI H, 58
MVI L, 00
MOV B, M
```

MVI L, 01

MOV A, M

MOV M, B

MVI L, 00

MOV M, A

(5) MVI M, DATA

It copies the data part of instruction to memory location M whose address is specified by HL pair.

Program – WAP to write a data value 65 H at location 5804.

MVI H, 58 H

MVI L, 04 H

MVI M, 65 H

NOTE: No variant.

(6) LXI R_P, $DATA_{16}$

It copies the 16 bit data part specified in instruction to the designated register pair. There are three variants of this instruction –

(i) LXI H
(ii) LXI B
(iii) LXI D

Let H = 25 H, L = 04 H

LXI H, 5806 [H = 58, L = 06]

Program – Exchange the content of two consecutive locations.

LXI H, 5800

MOV B, M

LXI H, 5801

MOV A, M

MOV M, B

LXI H, 5800

MOV M, A

(7) LDA $ADDRESS_{16}$: A ← $[M]_{Address}$

It loads the accumulator by the content of memory location M whose address is specified as part of instruction.

Example: Let at address 5906 the data is B5 H.

MVI A, 40 H [A = 40 H]

LDA 5906 H [A = B5 H]

MVI H, 59 H *LXI H, 5906 H* *LDA 5906 H*

MVI L, 06 H ≡ *MOV A, M* ≡

MOV A, M

(8) STA ADDRESS$_{16}$: [M]$_{Address}$ ← A

It stores the content of accumulator to the memory location M whose address is specified as part of instruction.

Example: Let at address 5906 the data is B5 H.

MVI A, 40 H [A = 40 H]

STA 5906 H [The value at memory location 5906 is 40 H]

MVI H, 59 H *LXI H, 5906 H* *STA 5906 H*

MVI L, 06 H ≡ *MOV M, A* ≡

MOV M, A

Program – Exchange the content of two consecutive locations.

LDA 5906 H

LXI H, 5907 H

MOV B, M

STA 5907 H

MOV A, B

STA 5906 H

(9) LHLD ADDRESS$_{16}$: L ← [M]$_{Address}$, H ← [M]$_{Address+1}$

Load HL Direct.

Given address loads the HL pair by content of two successive memory locations whose address is given as part of instruction. Thus the content of location M is transferred to L and content to location M+1 is transferred to H.

Example: Let the data value at memory location 5802 is 58 H and at memory location 5803 is 20 H.

LXI H, 0000 H [H = 00 H, L = 00 H]

LHLD 5802 H [L = 58 H, H = 20 H, then HL = 2058]

NOTE: It is a 16 bit memory to 16 bit register movement.

(10) SHLD ADDRESS$_{16}$: [M]$_{Address}$ ← L, [M]$_{Address+1}$← H

It stores the content of HL pair to two successive locations whose address is specified as part of instruction. The content of register L is moved to $M_{Address}$ whereas the content of register H is moved to $M_{Address+1}$.

Example: Let at memory location 5801 H the data value is 58, at memory location 5802 H the data value is 00 H.

LXI H, 5902 H [H = 59 H, L = 02 H]

MVI L, 64 H

SHLD 5801 [M_{5801} = 64 H, M_{5802} = 59 H]

(11) LDAX R_P : A ← [M]$_{RP}$

LoaD Accumulator with eXtended pair.

Loads the accumulator by the content of memory M whose address is given by the specified register pair. It is register indirect movement instruction.

NOTE: There are only two variants of this instruction.

LDAX B & LDAX D

Example: Let at memory location 5900 H the data value is 5B H.

MVI A, 59 H [A = 59 H]

LXI B, 5900 H [BC = 5900 H]

LDAX B [A = 5B H]

(12) STAX R_P : [M]$_{RP}$ ← A

Store accumulator with extended pair stores the content of accumulator to the memory location M whose address is specified by the given register pair.

NOTE: There are only two variants STAX B and STAX D.

Example: Let $[M]_{5900}$ is 5B H.

LXI D, 5900 H [D = 59 H, E = 00 H]

MVI A, 60 H [A = 60 H]

STAX D [M_{5900} = 60 H]

(13) XCHG : HL ↔ DE

It exchanges (swap) the contents of HL and DE pair (It uses XZ register pair for the purpose).

Example: LXI H, C1C2 H [HL = C1C2 H]

MVI D, 45 H [D = 45 H]

MVI E, 64 H [E = 64 H]

XCHG [HL = 4564 H, DE = C1C2 H]

NOTE: There are only one variant is available for this instruction.

(B) Arithmetic group – This group comprises of instructions related to basic arithmetic operations such as addition and subtraction. The group comprises of arithmetic manipulation instructions with reference to register and memory.

(1) ADD R : A ← A+R

This instruction adds the content of specified register R to accumulator retaining the result in accumulator; Flags are affected (Z – Zero, CY – Carry and AC – Auxiliary Carry).

Program – WAP to add two numbers.

```
MVI A, 04 H          [A = 04 H]
MVI B, FF H          [B = FF H]
ADD B          [A = A+B => 04 H + FF H = 03 H, CY = 1]
```

Program – WAP to add FF H and 01 H.

```
MVI A, FF H          [A = FF H]
MVI B, 01 H          [B = 01 H]
ADD B          [A = 00 H, CY = 1, Z = 1]
```

Program – Multiply the content of accumulator by 2.

```
MVI A, DATA    [A = DATA]
ADD A          [A = A+A]
```

NOTE: There are 7 variant of this instruction.

(2) ADD M : A ← A + $[M]_{HL}$

It adds the content of specified memory location M, whose address is given by HL pair to accumulator retaining result in accumulator.

Example: Let M_{5101} = 2C H.

```
MVI A, 07 H          [A = 07 H]
LXI H, 5101 H  [HL = 5101 H]
ADD M          [A = A+M_HL => A = 33 H]
```

Program – WAP to add the content of two successive memory locations X and X+1, storing the result at X+2.

```
LDA 5101
LXI H, 5102
ADD M
STA 5103
```

Program – WAP to add content of register B and C along with content of memory location X. store the result at location Y.

```
MVI B, DATA
MVI C, DATA
LDA X
ADD B
ADD C
STA Y
```

(3) ADI DATA : A ← A + DATA

It adds the data part of instruction to accumulator retaining the result in accumulator.

Program – WAP to increment the content of accumulator by 1.

```
MVI A, DATA
ADI 01 H
```

Program – WAP to decrement the content of accumulator by 1.

```
MVI A, DATA
ADI FF H
```

Program – WAP to decrement the content of memory location X.

```
LDA location X
ADI FF H
STA location X
```

(4) ADC R : A ← A + R + CY

It adds the specified register R along with carry flag value to accumulator.

NOTE: There are 7 variants of this instruction.

Example:

```
MVI A, 06 H          [A = 06 H]
MVI B, FE H          [B = FE H]
ADD B          [A = 06 + FE = 04 H, CY = 1]
MVI C, 04 H          [C = 04 H]
ADC C          [A = 04 + 04 + 1 = 09 H]
```

(5) ADC M : A ← A + $[M]_{HL}$ + CY

It adds the content of specified memory M whose address is given by HL pair along with carry flag to accumulator.

Example: Let data value at memory location 5801 H = FF H, 5802 H = FF H, 5803 H = FF H.

LDA 5801 H	[A = FF H]
LXI H, 5802 H	[HL = 5802 H]
ADD M	[A=A+M_{HL} => A=FFH + FFH => A=FEH, CY=1]
LXI H, 5803 H	[HL = 5803 H]
ADC M	[A=A+M_{HL}+CY => A=FE+FF+1=> A=FD, CY=1]

(6) ACI DATA : A ← A + DATA + CY

It adds the data part of instruction along with the content of carry to accumulator.

Example:

MVI A, 60 H	[A = 60 H]
ACI 40 H	[A = 60 H + 40 H + CY]

Since the carry flag is not given then result is unpredictable.

(7) DAD R_P : HL ← HL + RP

DAD H : HL ← HL + HL

DAD B : HL ← HL + BC

DAD D : HL ← HL + DE

It is a 16 bit instruction that adds specified one of the three register pairs to HL pair retaining the result in HL.

Example:

LXI H, 5132 H	[HL = 5132 H]
LXI D, 51FF H	[DE = 51FF H]
DAD D	[HL = 5132 H + 51FF H = A331 H]

Program – Given two numbers of 16 bit stored at X and X+2, add them and store the result back to X+4.

Let first number is B265 H and second number is BDA1 H.

LHLD 5100 H	[HL=B265]	5100	65	X
XCHG	[HL=Garbage, DE=B265]	5101	B2	
LHLD 5102 H	[HL=BDA1]	5102	A1	X+2

DAD D	[HL=B265+BDA1= 7006]	5103	BD	
SHLD 5104 H			5104	06
			5105	70

(8) SUB R : A ← A – R

The subtraction instructions considers the carry flag as borrow value i.e. during the subtraction if carry flag is set then it indicates that there is a borrow.

It subtracts the content of specified register R from accumulator retaining result in accumulator.

Example:

MVI A, 2B H	[A = 2B H]
MVI C, 1F H	[C = 1F H]
SUB C	[A = A – C = 2B – 1F = 0C H, CY = 1]

NOTE: There are 7 variants of this instruction.

Example: SUB A [Z = 0, A = 00 H, CY = 0, AC = 0, P = 1]

(9) SUB M : A ← A – $[M]_{HL}$

It subtracts the content of memory location M whose address is specified by HL pair from accumulator, retaining the result in accumulator.

Program – Subtract the content of location X+1 from X and store the result at X+2.

```
LDA loc.X
LXI H, loc.X+1
SUB M
STA loc.X+2
```

(10) SUI DATA : A ← A – DATA

It subtracts the data part of instruction from accumulator, retaining result in accumulator.

Example: MVI A, 06 H [A = 06 H]

SUI FC H [A = 0A H, CY = 1, Z = 0, AC = 0, S = 0, P = 0]

(11) SBB R : A ← A – (R + CY)

: A ← A – R – CY

It subtracts the content of specified register R along with content of carry flag (acting as borrow) from accumulator, retaining result in accumulator.

NOTE: There are 7 variant of this instruction.

Example: MVI A, 32 H [A = 32 H]

MVI B, 46 H [B = 46 H]

MVI C, 58 H [C = 58 H]

SUB B [A = 32 – 46 = EC H, CY = 1]

SBB C [A = EC – (58 + 1) = 93 H, CY = 0]

(12) SBB M : A ← A – (M_{HL} +CY)

: A ← M_{HL} – CY

It subtracts the content of memory location M whose address is specified by HL pair, along with carry from accumulator.

Example: Let data value at memory location 5800 is 58 H.

MVI A, 32 H [A = 32 H]

LXI H, 5800 H [HL = 5800 H]

MVI B, 46 H [B = 46 H]

SUB B [A = 32 – 46 = EC H, CY = 1]

SBB M [A = EC – (58 + 1) = 93 H, CY = 0]

(13) SBI DATA : A ← A – DATA – CY

: A ← A – (DATA + CY)

It subtracts the data part of instruction along with carry from accumulator.

Example: Let data value at memory location 5800 is 58 H.

MVI A, 32 H [A = 32 H]

MVI B, 46 H [B = 46 H]

SUB B [A = 32 – 46 = EC H, CY = 1]

SBI 58 H [A = EC – (58 + 1) = 93 H, CY = 0]

(14) INR R : R ← R + 1

It increments the content of specified register R by 1.

Example: MVI A, FF H [A = FF H]

MVI B, 3F H [B = 3F H]

INR A [A = 00 H]

INR B [B = 40 H]

Program – Add the content of memory location X and X+1 and store result at X+2.

LDA loc.X

LXI H, loc.X+1

ADD M

INR L

MOV M, A

Let at memory location 5801 the data value is 31 H, at 5802 data value is 32 H and at 5803 data value is 33 H.

LXI H, 5801 H [HL = 5801 H]

MOV A, M [A = 31 H]

INR L [L = 02 H]

ADD M [A = 31 + 32 = 63 H]

INR L [L = 03 H]

MOV M, A [$[M]_{HL}$ = 63 H]

NOTE: There are 7 variants.

(15) INR M : $M_{HL} \leftarrow M_{HL} + 1$

It increments the content of memory location M by 1.

Example: Let at memory location 5901 the data value is 33 H.

LXI H, 5901 H [HL = 5901 H, M_{HL} = 33 H]

INR M [M_{HL} = 34 H]

(16) INX R_P : $R_P \leftarrow R_P + 1$

It increments the register pair by 1.

NOTE: There are three variants of this instruction.

INX B, INX D, INX H.

Example: LXI H, 58FF H [HL = 58FF H]

LXI B, 00F1 H [BC = 00F1 H]

LXI D, 3123 H [DE = 3123 H]

INX H [HL = 5900 H]

INX B [BC = 00F2 H]

INX D [DE = 3124 H]

Program - LXI H, 58FF H

MOV A, M

INX H

ADD M

INX H

MOV M, A

(17) DCR R : R ← R – 1

It decrements the content of specified register R by 1.

NOTE: There are 7 variants of this instruction.

Example 1: MVI L, 00 H [L = 00 H]

DCR L [L = FF H, CY = 1]

Example 2: MVI H, A0 H [H = A0 H]

DCR H [H = 9E H]

(18) DCR M : M_{HL} ← M_{HL} – 1

It decrements the content of memory location M by 1.

Example: Let data value at memory location 58FF H is 31 H.

LXI H, 58FF H [HL = 58FF H]

DCR M [M_{HL} = 30 H]

(19) DCX RP : RP ← RP -1

It decrements the content of specified register pair by 1.

NOTE: There are three variants of this instruction.

Example: LXI H, 0000 H [HL = 0000 H]

LXI B, FF00 H [BC = FF00 H]

LXI D, 00FF H [DE = 00FF H]

DCX H [HL = FFFF H]

DCX B [BC = FEFF H]

DCX D [DE = 00FE H]

(20) DAA : Decimal Adjust Accumulator.

This instruction is used in BCD arithmetic. When the value of lower nibble exceeds 9 or if an auxiliary carry is generated, it adds 6 to lower nibble.

Similarly, if the upper nibble exceeds by 9 or carry flag is set then it adds 6 to upper nibble.

Example: MVI A, 88_d [A = 88_d]

MVI B, 89_d [B = 89_d]

ADD B

DAA [A = 77_d, CY = 1]

1

(88) = 1000 1000 (BCD)

+ (89) = + 1000 1001 (BCD)

(177) 1 0001 0001

+ 0110 0110

0001 0111 0111

(C) Logical group – This group comprises of the instructions related to logical operations carried out over the content of registers and memory.

The basic logical operators are – AND, OR, XOR and NOT. All the operations are bit wise type.

(21) ANA R : A ← A AND R

It performs the bitwise logical ANDing between the content of accumulator and specified register R, retaining result in accumulator.

NOTE: There are 7 variants.

Example: MVI A, B6 H [A = B6 H = 1011 0110]

MVI B, 5C H [B = 5C H = 0101 1100]

ANA B [A = 14 H "Logical ANDing is given below"]

A = 1011 0110

B = 0101 1100

A = 0001 0100 = 14 H

(22) ANA M : A ← A AND M_{HL}

It performs bitwise logical ANDing between the contents of accumulator and memory M, retaining result in accumulator.

Example: Let at memory location 5901 the data value is 46 H, at 5902 the data value is 44 H.

LDA 5901 H [A = 46 H]

LXI H, 5902 H [HL = 5902]

ANA M

STA 5902 [A = 44 H "Logical ANDing is given below"]

A = 0100 0101

M_{5902} = 0100 0110

0100 0100 = 44 H

(23) ANI $DATA_8$: A ← A AND DATA

It performs bitwise logical AND operation between the content of accumulator and the data part of instruction, retaining result in accumulator.

Example: MVI A, 0F H [A = 0F H]

ANI 56 H [A = 06 H]

A = 0000 1111

56 = 0101 0110

0000 0110 = 0 6 H

(24) ORA R : A ← A OR R

It performs bitwise logical OR operation over the content of accumulator and specified register R, retaining result in accumulator.

NOTE: There are 7 variants.

Example: MVI A, AA H [A = AA H = 1010 1010]

MVI B, 55 H [B = 55 H = 0101 0101]

ORA B [A = 1111 1111 = FF H]

(25) ORA M : A ← A OR MHL

It performs bitwise logical ORing between the contents of accumulator and memory M, retaining result in accumulator.

Example: Let data at memory location 5800 is C1 H.

MVI A, 45 H [A = 45 H = 0100 0101]

LXI H, 5800 H [HL = 5800 H, MHL = C1 H = 1100 0001]

ANA M [A = 1100 0101 = C5 H]

(26) ORI $DATA_8$: A ← A OR DATA

It performs bitwise logical OR operation between the content of accumulator and the data part of instruction, retaining result in accumulator.

Example: MVI A, 07 H [A = 07 H = 0000 0111]

ORI 63 H [63 H = 0110 0011, A = 0110 0111 = 67 H]

(27) XRA R : A ← A ⊕ R

It carryout the logical bitwise XOR operation between the content of register R and accumulator, retaining result in accumulator.

NOTE: There are 7 variants.

Example: MVI A, 6B H [A = 6B H = 0110 1011]

MVI H, 43 H [H = 43 H = 0100 0011]

XRA H [A = 0010 1000 = 28 H]

To clear the content of accumulator we use following instruction –

XRA A *[A = 00 H]*

(28) XRA M : A ← A $\oplus$ M_{HL}

It performs bitwise logical XORing between the contents of accumulator and memory M, retaining result in accumulator.

Example: Let data value at memory location 5901 is 33 H and data at 5902 is 34 H.

LXI H, 5901 H [HL = 5901 H]

MOV A, M [A = M_{HL} = 33 H]

INX H [HL = HL + 1 = 5902 H]

XRA M [A = 33 H $\oplus$ 34 H = 07 H]

INX H [HL = HL + 1 = 5903 H]

MOV M, A [M_{HL} = M_{5903} = 07 H]

(29) XRI $DATA_8$: A ← A $\oplus$ DATA

It bitwise XOR's the content of accumulator with data part of instruction, retaining result in accumulator.

Example: MVI A, 0F H [A = 0F H = 0000 1111]

XRI F0 H [A = 0F $\oplus$ F0 = 0000 1111 $\oplus$ 1111 0000 = FF H]

(30) CMA : A ← $\overline{A}$

It complements bitwise the content of accumulator or it provides 1's complement over content of accumulator.

Example: MVI A, 6B H [A = 6B H = 0110 1011]

CMA [A = 1001 0100 = 94 H]

Program – Perform 2's complement subtraction over data of register B and C, storing the result in D.

Let B = 57 H, C = 4C H

MVI B, 57 H [B = 57 H]

MVI C, 4C H [C = 4C H]

MOV A, C [A = 4C H = 0100 1100]

CMA [A = 1011 0011 = B3 H]

INR A [A = B4 H]

ADD B [A = B4 + 57 = 0B H, CY = 1]

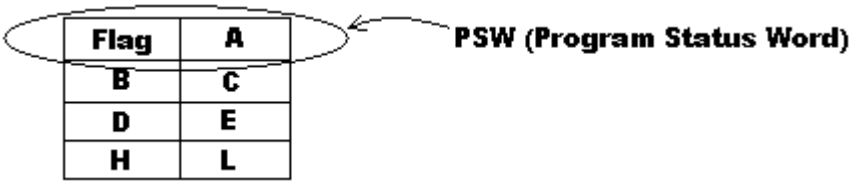
MOV D, A [D = 0B H, CY = 1]

(31) STC : CY ← 1

It sets the carry to 1 i.e. irrespective of the value of carry, this instruction makes carry 1.

(32) CMC : CY ← $\overline{CY}$

Complement carry inverts the carry bit value.



STC

CMC

XRA A

The flags associated with accumulator with flags as Most Significant Byte (MSB) is referred to as Program Status Word or PSW.

Above three instructions clears PSW.

P = 1, S = 0, Z = 1, AC = 0, C = 0, A = 0.

(33) CMP R : Compares A and R

	Z	***CY***
A < B	***0***	***1***
A = B	***1***	***0***
A > B	***0***	***0***

TEMP ← A (Save A)

A ← A – R (Subtract R from A, affecting flag)

A ← TEMP (Restore A)

It compares the contents of accumulator with specified register R, without changing the content of A or R but flags are affected. (None of register contents are modified).

To perform this instruction –

Content of A are saved in temporary register, content of register R are subtracted from A, flags are affected, A is restored by temporary register.

Example: MVI A, 30 H

MVI B, 57 H

CMP B

Result: Z = 0, C = 1.

(34) CMP M : Compare A and M_{HL}

It compares the content of A with memory, affecting flags without affecting the content of memory or A.

Example: MVI A, 30 H

LXI H, 5800 H

MVI M, 57 H

CMP M

Result: Z = 0, C = 1.

(35) CPI DATA : Compare A and DATA

It compares the data part of instruction with A, affecting flags.

Example: MVI A, 05 H

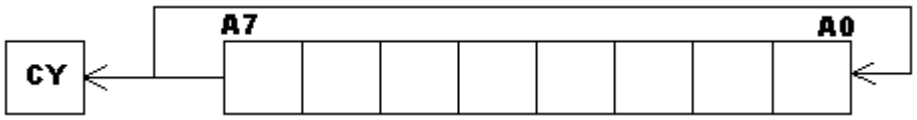
CPI 06 H

Result: Z = 0, C = 1.

ROTATION –

The rotation instructions are used to rotate the content of A with reference to carry flag. Those instructions are used in multiplication, division and serial communication.

(36) RLC : Rotate Left to Carry

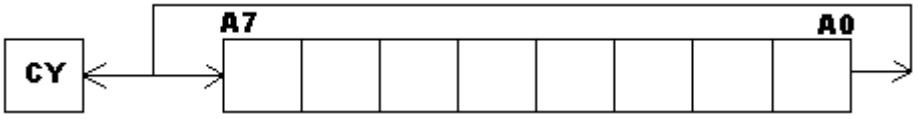


x abcdefgh

a bcdefgha

This instruction rotates the content of accumulator to left hand side (L.H.S.), where by A7 is copied to A0 and CY. The carry is destroyed.

(37) RRC : Rotate Right in Carry

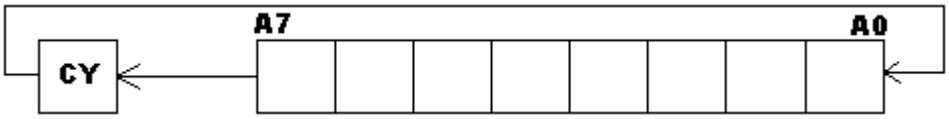


x abcdefgh

h habcdefg

This instruction rotates the content of accumulator to right by copying to A0 to A7 and CY. The carry is destroyed.

(38) RAL : Rotate All Left

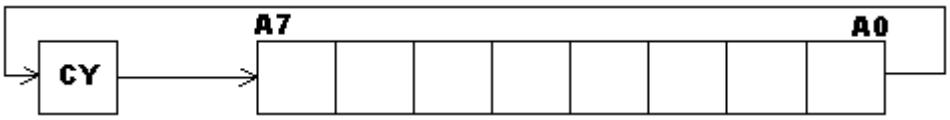


x abcdefgh

a bcdefghx

This instruction is equivalent to 9 – bit rotation to left. The content of carry along with content of accumulator is rotated left. A7 is copied to CY. CY is copied to A0, A0 is copied to A1 and so on.

(39) RAR : Rotate All Right



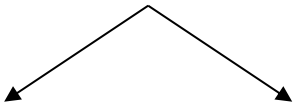
x abcdefgh

h xabcdefg

This instruction is also similar to rotating 9 – bits to right. The content of carry are copied to A7, the content of A0 is copied to CY, the content of A1 is copied to A0 and so on.

Example:	STC	[CY = 1]
	CMC	[CY = 0]
	MVI A, A5 H	[A = A5 H, 0 1010 0101]
	RAL	[1 0100 1010, A = 4A H, CY = 1]
	RAR	[0 1010 0101, A = A5 H, CY = 0]
	RRC	[1 1101 0010, A = D2 H, CY = 1]
	RLC	[1 1010 0101, A = A5 H, CY = 1]

(D) Branch group –



Unconditional Conditional

This group comprises of the instruction related with transfer of control. The transfer of control means that content of Program Counter (PC) is loaded by address of some other location from the current instruction and not the next address. The transfer of control can be Unconditional or Conditional one. In 8085 since there are 5 conditions abbreviated as flags, we can have 10 conditional transfers of controls.

In concept of modular programming an assembly language program can also have subprograms called Macros.

The subprogram can also be called Unconditional or Conditional. Every subprogram comprises of at least one RETURN statement that may be Unconditional or Conditional one.

The Unconditional jump condition transfer the control to specified label (Memory Address) that is after execution of this instruction the control is transferred not to next instruction but to address specified by label.

(40)	**JC**	**Label**	**If**	**CY = 1**	**Then**	**PC ← Label**	
(41)	**JNC**	**Label**	**If**	**CY = 0**	**Then**	**PC ← Label**	
(42)	**JZ**	**Label**	**If**	**Z = 1**		**Then**	**PC ← Label**
(43)	**JNZ**	**Label**	**If**	**Z = 0**		**Then**	**PC ← Label**
(44)	**JP**	**Label**	**If**	**S = 0**		**Then**	**PC ← Label**
(45)	**JM**	**Label**	**If**	**S = 1**		**Then**	**PC ← Label**
(46)	**JPE**	**Label**	**If**	**P = 1**		**Then**	**PC ← Label**
(47)	**JPO**	**Label**	**If**	**P = 0**		**Then**	**PC ← Label**

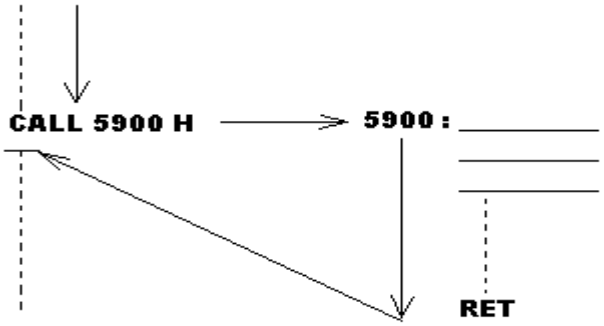
```
LABEL  MNEMONICS      ADDRESS     OPCODE        REMARKS

; The program multiplies two numbers
                ORG 5800 H
START  XRA A          5800   :    AF            ; Clear accumulator
       MVI B, DATA1   5801   :    06, 04  ___ ; Move multiplicand to B
       MVI C, DATA2   5803   :    0E, 05  ___ ; Move multiplicand to C
LOOP   :  ADD B       5805   :    80            ; Add B to A
       DCR C          5806   :    0D            ; Decrement C
       JNZ LOOP       5807   :    C2, 05, 58    ; Jump to LOOP if not 0
       HLT            580A   :    76            ; Stop processing
DATA1  EQU            04 H
DATA2  EQU            05 H
       END                       // Finish of pass 1.
```

SYMBOL TABLE –

NAME	VALUE
START	5800 H
DATA1	04 H
DATA2	05 H
LOOP	5805 H

SUBROUTINES – **– CALL**

– RETURN



A subroutine in 8085 can be called conditionally or unconditionally and the return from subroutine can be also conditional or unconditional. The program can comprises of combinations of any type of CALLs and RETURNs i.e. the conditional CALL can return unconditionally or conditionally, whereas the unconditional CALL can return unconditionally.

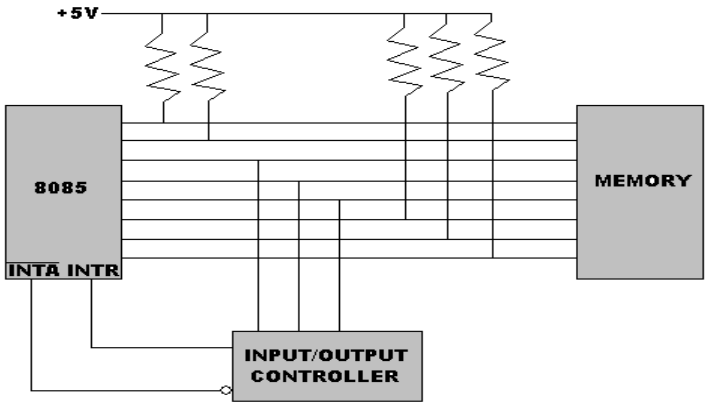
A CALL statement is always the part of calling program associated with either the address of subroutine or the subroutine name. The RETURN is part of the called program.

Unconditional→		CALL	Address {Label}		RET
Conditional	If	CY = 1	CC	Address {Label}	RC
	If	CY = 0	CNC	Address {Label}	RNC
	If	Z = 1	CZ	Address {Label}	RZ
	If	Z = 0	CNZ	Address {Label}	RNZ
	If	S = 0	CP	Address {Label}	RP
	If	S = 1	CM	Address {Label}	RM
	If	P = 1	CPE	Address {Label}	RPE
	If	P = 0	CPO	Address {Label}	RPO

RST n **n = [0,7]**

		OPCODE			
RST	0	C7	11	000	111
RST	1	CF	11	001	111
RST	2	D7	11	010	111
RST	3	DF	11	011	111
RST	4	E7	11	100	111
RST	5	EF	11	101	111
RST	6	F7	11	110	111
RST	7	FF	11	111	111

TABLE 1



The software interrupts can be called in two ways –

(i) Internally by the program, by writing RST 0, RST 1 etc.
(ii) Externally by the interrupting device.

TABLE 1 shows the interrupts along with their operation code and respective vectors. In context of INTR the microprocessor expects the OPCODE to be supplied by the interrupting device. On accepting on this OPCODE the microprocessor transfers the control to respective vector. To accomplish the INTR call a specific circuit arrangement is needed over the data bus as shown in above figure. From the OPCODE set of these interrupts we find a typical resemblance between the codes that the first two and last three bits are a one 0and the internal three bits ranges from 000 to 111, indicating the restart number. Thus on the occurrence of INTR, the interrupting device is require to provide only three bit value i.e. the restart number.

PCHL : PC ← HL

Example:

5800 : MVI A, DATA	PC : 5800
5802 : LXI H, 5908	PC : 5802
5805 : MOV M, A	PC : 5805

5806 : PCHL PC : 5806

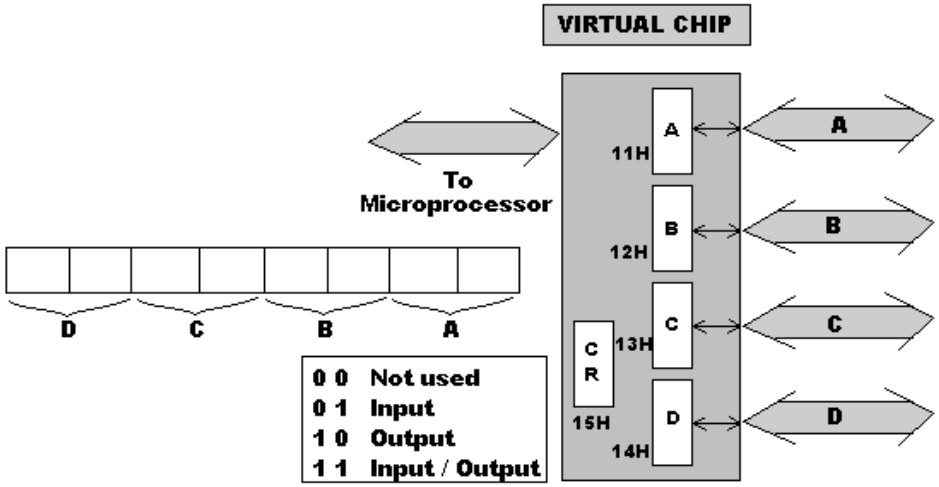
PC : 5807 → PC : 5908

(E) Machine control group –

This group comprises of the instruction related to interaction on the I / O devices with microprocessor. It also comprises the instruction related to stack manipulations.

PORT – A port is defined as an entity of a micro computer that establishes a communication mechanism between the microprocessor and I / O devices.

In 8085 the number of ports are 0 – 255.



IN PORT : A ← PORT

This instruction reads the specified port.

OUT PORT : PORT ← A

It transfers the content of accumulator to the specified port.

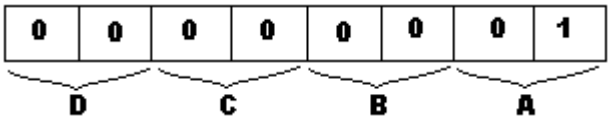
NOTE: In 8085 there can be 256 ports ranging from 00 H to FF H.

Example:

MVI A, 01 H [A = 01 H = 00 00 00 01]

OUT 15 H [The data of accumulator is given to the Control Register]

IN 11 H [Read the data from port 11H means from port A into Acc.]



Following program is used to read two ports –

MVI A, 05 H

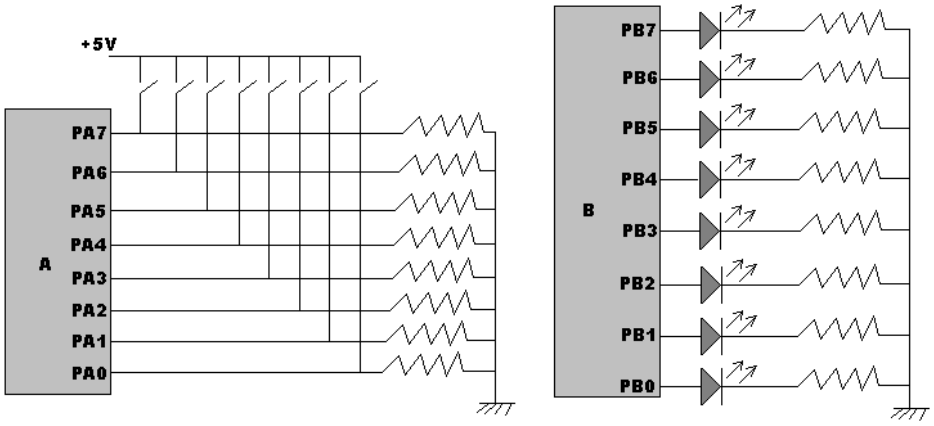
OUT 15 H

IN 11 H

MOV B, A

IN 12 H

MOV C, A



	MVI A, 09 H	MVI A, 08 H
	OUT 15 H	OUT 15 H
xyz :	IN 11 H	MVI A, 55 H ***[Program for running light]***
	OUT 12 H	xyz : OUT 12 H
	JMP xyz	CMA
		JMP xyz

STACK →

PUSH Rp : $[M]_{SP-1}$ ← High Register

: $[M]_{SP-2}$ ← Low Register

This instruction pushes the content of specified register pair onto the stack with higher register first and lower register next. The content of stack pointer is decremented by 1 and the higher register is stored there. Again the stack pointer is decremented by 1 (effectively by 2) and the lower register is stored there. The stack pointer remains at SP(Stack Pointer) – 2.

NOTE: There are 3 variants of this instruction i.e. PUSH B, PUSH D and PUSH H.

Example: Let the Stack Pointer is at 58FF H and the data value at 58FF H is DD H.

LXI D, 35B4 H [DE = 35B4 H]

PUSH D [M_{SP-1} = 35 H, M_{SP-2} = B4 H & SP = 58FD H]

PUSH PSW : $[M]_{SP-1}$ ← Flags

: $[M]_{SP-2}$ ← Accumulator

It pushes the PSW (Program Status Word) onto the stack with Flags first and Accumulator next. In other words $[M]_{SP-1}$ is stored with Flags and $[M]_{SP-2}$ is stored with Accumulator.

Example: Let Flags = 37 H, Accumulator = 50 H and SP = 5808 H.

PUSH PSW [M_{SP-1} = 37 H, M_{SP-2} = 50 H and SP = 5806 H]

POP Rp : Register Low ← $[M]_{SP}$

Register High ← $[M]_{SP+1}$

SP ← SP + 2

It pop backs (retrieves) the content of stack value into the specified register pair. During pop operation, the content of $[M]_{SP}$ is transferred to lower order register and content of $[M]_{SP+1}$ is transferred to higher order register. The SP is kept as SP+2.

NOTE: It has three variants.

Example: Let SP = 5932 H and the data value at 5932 H memory location is C5 H and the value at 5933 H memory location is 8B H.

LXI B, 0000 H [BC = 0000 H]

POP B [C = M_{SP} = C5, B = M_{SP+1} = 8B & SP = 5934 H]

Program – Write a program to exchange the content of HL and DC.

```
LXI H, FF00 H
LXI B, 55AA H
PUSH H
PUSH B
POP H
POP B
```

POP PSW : Accumulator ← $[M]_{SP}$

Flags ← $[M]_{SP+1}$

SP ← SP +2

This instruction retrieves 16 – bit value from the stack into PSW (Program Status Word). The first value is transferred to Accumulator and the next to Flags.

HLT : HOLT

It is the logical end of the program whereby after execution of this instruction the processing stops till next intervention.

XTHL : HL ↔ STACK

It exchanges the content of HL pair with the two top most stack elements.

XZ ← HL

L ← [M]SP

H ← [M]SP+1

[M]SP+1 ← X

[M]SP ← Z

Example: Let SP = 5909 H and data value at memory location 5909 is 56 H and data value at memory location 590A is 78 H.

LXI H, 5332 H [HL = 5332 H]

XTHL [M5909 = 32 H, M590A = 53 H & HL = 7856 H]

SPHL : SP ← HL

It is used to assign the memory address value to the Stack Pointer i.e. to create a Stack Pointer we first move address into HL pair and then call this instruction.

Example: LXI H, 3FFE H [HL = 3FFE H]

SPHL [SP = 3FFE H]

INTERRUPTS →

SIM [Set Interrupt Mask]

RIM [Read Interrupt Mask]

EI [Enable Interrupt]

DI [Disable Interrupt]

NOP : No OPeration

This instruction doesn't perform any operation or is called No Operation. It is used in two major cases –

1. Where a short delay is requisited.
2. During the development of program we may require some empty lines that will be filled after words. Writing of NOP for empty lines removes the problem of address relocation of program.

INTERRUPTS →

FLOW CHART

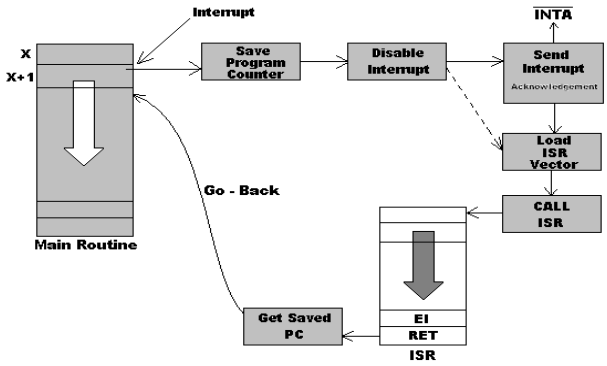


FIGURE - 1

SIM : Set Interrupt Mask

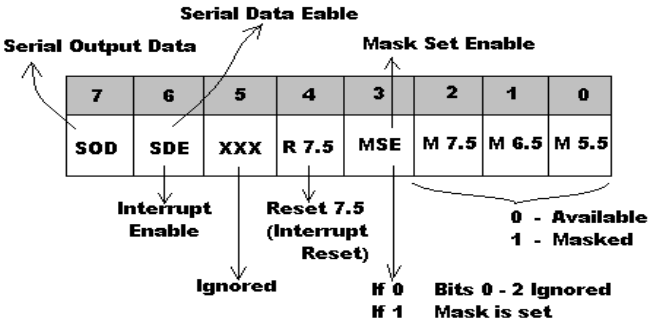


FIGURE – 2.

MVI A, sim pattern

SIM

If 6th bit of accumulator is 1 then 7th bit of accumulator is available at SOD pin.

If 6th bit of accumulator is 0 then SOD pin of microprocessor is unchanged.

RST 7.5 is retriggrable interrupt.

To mask 5.5 and 7.5, the SIM instructions are used as:

MVI A, ××××1101b

SIM

RIM : Read Interrupt Mask

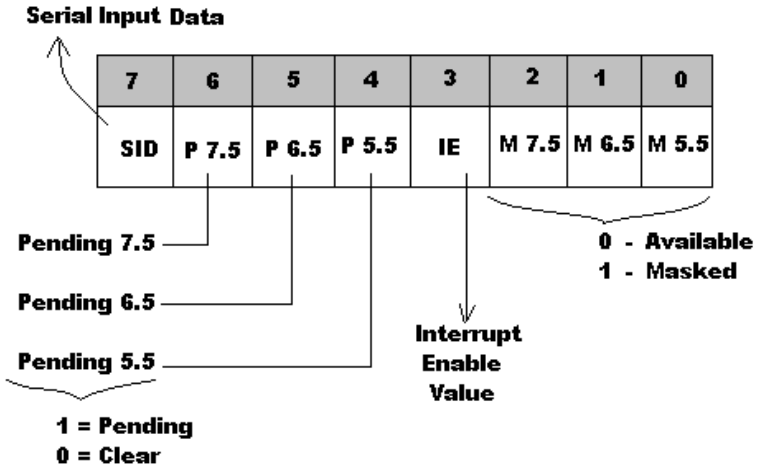


FIGURE - 3

Interrupt – A signalic mechanism by which an external device or internal to an entity performs a transfer of control for getting a specific service done are termed interrupts.

Interrupts can be classified as –

1. *External and Internal interrupts*
2. *Hardware and Software interrupts*
3. *Vectored and Non – vectored interrupts*

The interrupts that are generated by applying a voltage signal over the pin of a microprocessor are termed *Hardware interrupts* and they are also called *External interrupts*. Whereas the interrupts that are generated programmatically are called *Software interrupts* or *Internal interrupts*.

Every interrupts requires a specific routine i.e. called on interrupt occurrence. It is specifically named ISR or Interrupt Service Routine. A ISR is hooked to an address. If the address is fixed and is not provided by the interrupting device then is stated a *Vectored interrupt*. Whereas if the interrupting device is asked for address of ISR then the interrupt is stated a *Non – vectored*.

The interrupts can also be Non – mask – able or mask – able ones. If the interrupt can't be ignored by any means then it's stated *Non – mask – able* else is called *mask – able* interrupt.

INTERRUPT NAME	MASKABLE	VECTORED
INTR	YES	NO
RST 5.5	YES	YES
RST 6.5	YES	YES
RST 7.5	YES	YES
TRAP	NO	YES

An interrupt call is similar to that of calling a subroutine where, when an interrupt is generated, the current PC (Program Counter) value is pushed onto the stack; a DI instruction is called (to disable nesting of interrupts). Then in case of vectored its address is loaded and in case of non vectored the address is demanded. The control is transferred to ISR whereby the interrupt is serviced. At the end of ISR EI is needed to be called for activating the next interrupt occurrence if any. The ISR is always terminated by a return instruction that get back the previous address from the stack, the point from which the interrupt was called. The schematic of interrupt service is shown in figure 1.

Instructions related to interrupts –

1. ***DI :*** This instruction disables all the mask-able interrupts to occur.
2. ***EI :*** This instruction enables all the mask-able interrupts occurrence subject to they are not masked.
3. ***SIM :*** The Set Interrupt Mask instruction is used for performing two functions :
    i. Serial communication
    ii. Masking of interrupts etc.

The accumulator bit pattern of SIM instruction is illustrated in figure 2.

The bit specifications are –

a. ***SOD (A.7) :*** This bit of accumulator pattern is associated with SOD pin. The binary value of it is reflected as a voltage level over SOD pin subject to A.6 = 1.

   If A.7A.6 = 1 1 then SOD pin is at +5 volt on execution of SIM instruction. Similarly A.7 A.6 = 01, the SOD pin is at 0 volt after execution of SIM instruction. If A.7 A.6 = × 0 then SOD pin remains unchanged.

b. ***SDE (A.6) :*** The Serial Data Enable flag controls the data of A.7 to appear on SOD pin. If it is 1 SOD pin data depends on A.7, if it is 0 SOD pin remains unchanged.
c. ***××× (A.5) :*** This bit specification is unused / not defined.
d. ***R 7.5 (A.4) :*** This bit is used to reset (clear) the RST 7.5 interrupt such that it can be retriggered some next time (because it is an edge triggered interrupt). With this bit = 1 and the SIM instruction is executed, the RST 7.5 interrupt memory is cleared.
e. ***MSE (A.3) :*** The Mask Set Enable bit is used to enable or disable the mask pattern of RST 7.5, RST 6.5 and RST 5.5. If this bit is 1, the mask is applied.
f. ***M 7.5, M 6.5, M 5.5 (A.2, A.1, A.0) :*** These the mask bit pattern of respective pattern. A zero at bit position enables the interrupt, where a one masks the interrupt subject to MSE is 1.

4. ***RIM :*** The Read Interrupt Mask instruction is also having functionality for serial communications as well as states the interrupt status.

a. ***SID (A.7) :*** The Serial Input Data bit is associated with the SID pin. On execution of RIM instruction, this bit gets the status of SID pin. If SID pin is at logic high value, this bit position becomes 1 and if SID pin is at low logic, this bit becomes a zero.
b. ***P 7.5, P 6.5, P 5.5 (A.6, A.5, A.4) :*** These bit shows the pending status of respective interrupts such that if an interrupt is waiting, this bit is high. Thus on returning from an interrupt, if RIM instruction is executed, we can analyze the pending interrupt whereby the servicing higher one.
c. ***IE (A.3) :*** The Interrupt Enable bit position indicates that the interrupt were masked or not.
d. ***M 7.5, M 6.5, M 5.5 (A.2, A.1, A.0) :*** These bit position shows the mask pattern of respective interrupts, a 1 indicating mask interrupt.

INTERNAL ARCHITECTURE OF INTERRUPTS –

Figure illustrates the basic circuit needed for handling the interrupts of 8085. The TRAP interrupt since is non mask-able and should not occur by spurious signals, requires a typical mechanism for its sensing. The scheme of TRAP is level as well as edge triggering, accomplished by flip flop FF5. The propagation delay of FF5 provides a condition for avoiding the intermittent signals. When the TRAP occurs, the level transition from 0 to 1 triggers the clock making Q = 1 but at the same time the Q appears after the propagation delay of flip flop. For the propagation delay the TRAP should hold a one then only the gating produces a one generating the TRAP signal. This circuit action avoids false triggering because of the fact that if the TRAP input is not sustained for propagation delay of flip flop, it can't be generated.

The remaining interrupts are mask-able and hence are controlled by FF0, the master controller. The gating of all the interrupts (mask-able) is controlled by this flip flop are associated with EI and DI instructions respectively. When DI instruction is executed, it forcely reset FF0 disabling all the AND gates, there by disabling all the interrupts. Similarly a EI instruction asynchronously makes Q of FF0 a 1, this action enables the gating where now the occurrence of interrupt depends on SIM instruction and accumulator bit pattern. The INTR is directly controlled by DI and EI.

Flip flop FF1 is used to control and sense RST 5.5. Its D is associated with A.0 and the clock is generated SIM instruction. Since D is bubbled, A.0 = 0 with SIM execution, enables RST 5.5 to occur.

The explanation exists for RST 6.5 and RST 7.5 along with the SIM instruction. The SIM clock is applied to flip flop's only if A.3 = 1 (Mask Set Enable is true).

RST 7.5 is a retrigger-able / reset-able interrupt that can by made reset by A.4 or by INTA.

FF4 is used for this purpose where it can be made clear by either $\overline{\text{INTA}}$ or A.4 = 1 along with SIM instruction.

S.No.	**Interrupt Name**	**Mask-able / Non Mask-able**	**Vectored / Non Vectored**	**Memory**	**Trigger**
1.	TRAP	NM	V	No	Edge & Level triggered
2.	RST 7.5	M	V	Yes	Edge triggered
3.	RST 6.5	M	V	No	Level triggered
4.	RST 5.5	M	V	No	Level triggered
5.	INTR	M	NV	No	Level triggered