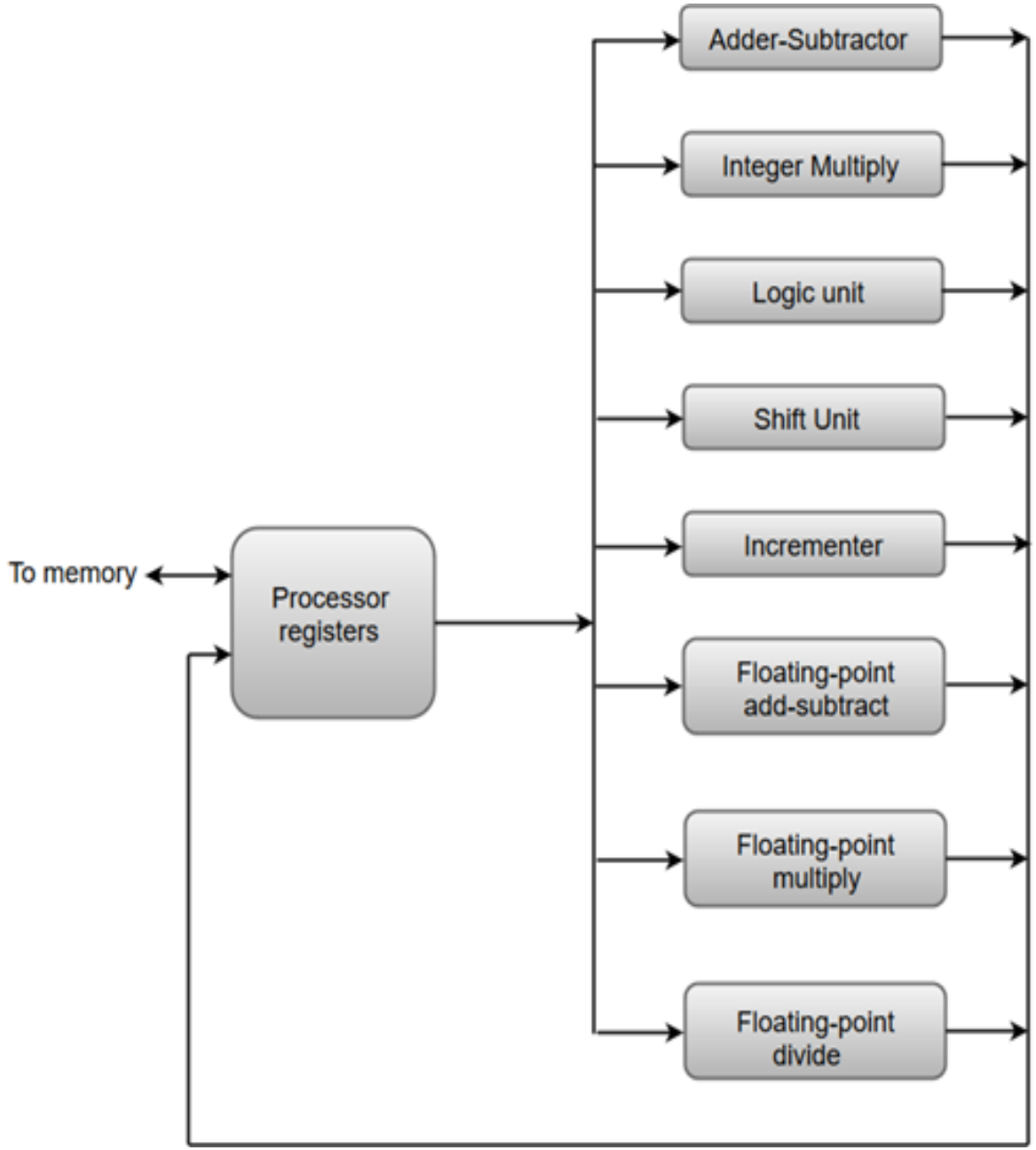
Chapter-6

Parallel Processing :-

- **Parallel processing** is a term used to denote a large class of techniques that are used to provide simultaneous data-processing tasks for the purpose of increasing the computational speed of a computer system.
- Instead of processing each instruction sequentially as in a conventional computer, a parallel processing system is able to perform concurrent data processing to achieve faster execution time.
- For example, while an instruction is being executed in the ALU, the next instruction can be read from memory. The system may have two or more ALUs and be able to execute two or more instructions at the same time.
- The amount of hardware increases with parallel processing. and with it, the cost of the system increases. However, technological developments have reduced hardware costs to the point where parallel processing techniques are economically feasible.


Adder-Subtractor
Integer Multiply
Logic unit
Shift Unit
Incrementer
To memory
Processor registers
Floating-point add-subtract
Floating-point multiply
Floating-point divide

- Parallel processing is established by distributing the data among the multiple functional units.
- For example, the arithmetic, logic, and shift operations can be separated into three units and the operands diverted to each unit under the supervision of a control unit.
- The adder and integer multiplier perform the arithmetic operations with integer numbers.
- The floating-point operations are separated into three circuits operating in parallel.
- The logic, shift, and increment operations can be performed concurrently on different data.
- All units are independent of each other, so one number can be shifted while another number is being incremented.
- A multifunctional organization is usually associated with a complex control unit to coordinate all the activities among the various components.

Flynn's Classification of Parallel Processing :-

- **M. J. Flynn** considers the organization of a computer system by the number of instructions and data items that are manipulated simultaneously.
- The normal operation of a computer is to fetch instructions from memory and execute them in the processor.
- The sequence of instructions read from memory constitutes an **instruction stream**. The operations performed on the data in the processor constitutes a **data stream**.
- Parallel processing may occur in the **instruction stream, in the data stream, or in both**

Flynn's classification divides computers into **four major groups** as follows:

1. Single instruction stream, single data stream (SISD)
2. Single instruction stream, multiple data stream (SIMD)
3. Multiple instruction stream, single data stream (MISD)
4. Multiple instruction stream, multiple data stream (MIMD)

- **SISD :-** It represents the organization of a single computer containing a control unit, a processor unit, and a memory unit.
- Instructions are executed sequentially and the system may or may not have internal parallel processing capabilities.
- Parallel processing in this case may be achieved by means of multiple functional units or by pipeline processing.
- **SIMD** represents an organization that includes many processing units under the supervision of a common control unit.
- All processors receive the same instruction from the control unit but operate on different items of data.
- The shared memory unit must contain multiple modules so that it can communicate with all the processors simultaneously.

- **MISD** structure is only of theoretical interest since no practical system has been constructed using this organization.

- **MIMD** organization refers to a computer system capable of processing several programs at the same time.
- Most multiprocessor and multicomputer systems can be classified in this category.

Pipelining :-

- **Pipelining** is a technique of decomposing a sequential process into sub-operations, with each sub-process being executed in a special dedicated segment that operates concurrently with all other segments.
- Each segment performs partial processing dictated by the way the task is partitioned. The result obtained from the computation in each segment is transferred to the next segment in the pipeline. The final result is obtained after the data have passed through all segments.
- It is characteristic of pipelines that several computations can be in progress in distinct segments at the same time.

Pipelining Example :-

- Suppose that we want to perform the combined multiply and add operations with a stream of numbers.

  $Ai * Bi + Ci$ for i=1,2,3,4,5,6,7

- Each sub-operation is to be implemented in a segment within a pipeline. Each segment has one or two registers and a combinational circuit

$R1 \leftarrow A_i, \quad R2 \leftarrow B_i$	Input A_i and B_i
$R3 \leftarrow R1 * R2, \quad R4 \leftarrow C_i$	Multiply and input C_i
$R5 \leftarrow R3 + R4$	Add C_i to product

Figure 9-2 Example of pipeline processing.

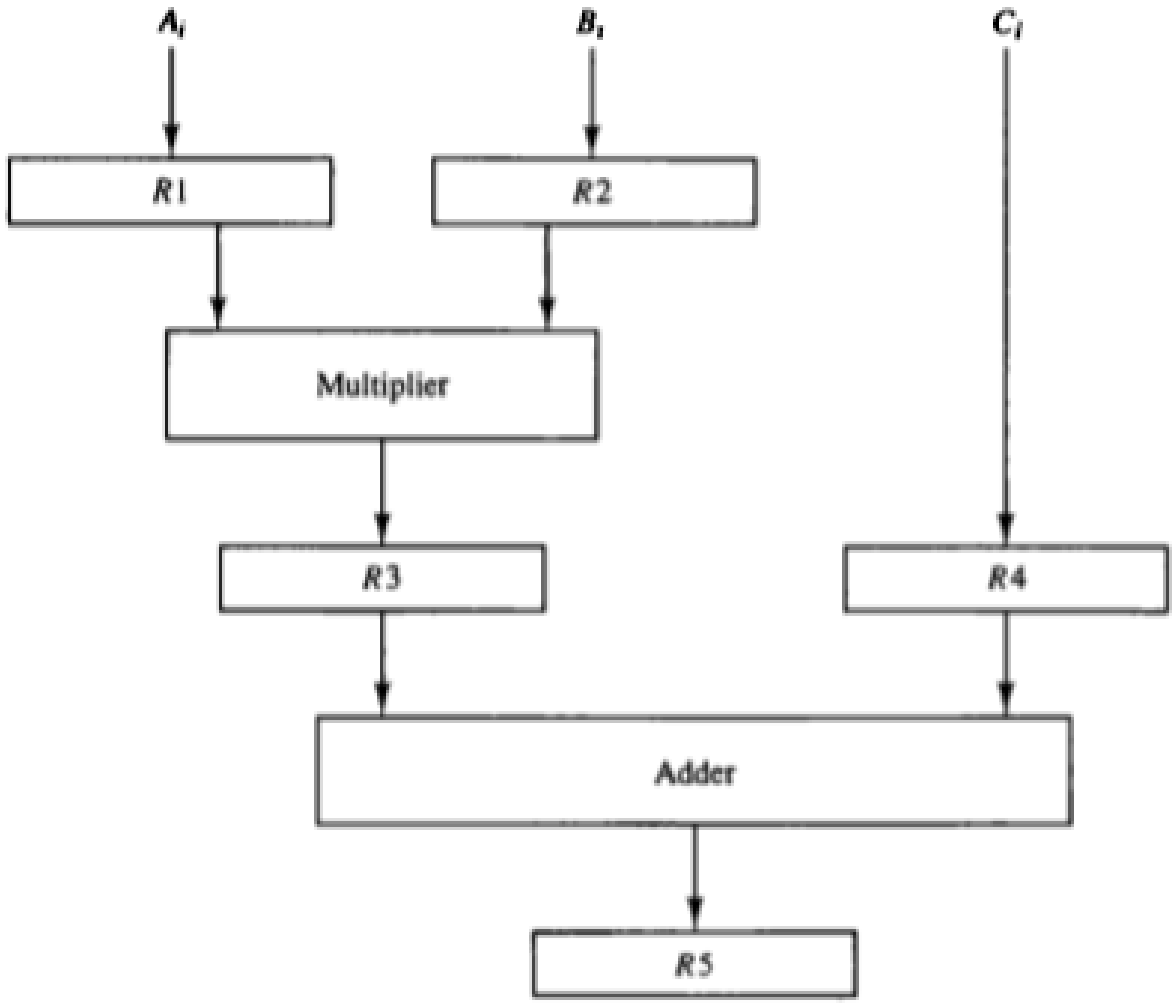


TABLE 9-1 Content of Registers in Pipeline Example

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	$R1$	$R2$	$R3$	$R4$	$R5$
1	A_1	B_1	—	—	—
2	A_2	B_2	$A_1 * B_1$	C_1	—
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
8	—	—	$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
9	—	—	—	—	$A_7 * B_7 + C_7$

Instruction Pipeline

- Pipeline processing can occur not only in the data stream but in the instruction stream as well.
- **An instruction pipeline** reads consecutive instructions from memory while previous instructions are being executed in other segments.
- This causes the instruction fetch and execute phases to overlap and perform simultaneous operations.
- Computers with complex instructions require other phases in addition to the fetch and execute to process an instruction completely.

- In the most general case, the computer needs to process each instruction with the following sequence of steps:
  - Fetch the instruction from memory.
  - Decode the instruction.
  - Calculate the effective address.
  - Fetch the operands from memory.
  - Execute the instruction.
  - Store the result in the proper place.
- The design of an instruction pipeline will be most efficient if the instruction cycle is divided into segments of equal duration. The time that each step takes to fulfill its function depends on the instruction and the way itis executed

- The four segments are represented as:
  - **FI**: segment 1 that fetches the instruction.
  - **DA**: segment 2 that decodes the instruction and calculates the effective address.
  - **FO**: segment 3 that fetches the operands.
  - **EX**: segment 4 that executes the instruction

Step	1	2	3	4	5	6	7	8	9
1	FI	DA	FO	EX					
2		FI	DA	FO	EX				
3			FI	DA	FO	EX			
4				FI	DA	FO	EX		
5					FI	DA	FO	EX	
6						FI	DA	FO	EX

Fig: timing diagram for 4-segment instruction pipeline

Pipeline Conflicts :-

- A pipeline hazard occurs when the instruction pipeline deviates at some phases, some operational conditions that do not permit the continued execution.
- In general, there are three major difficulties that cause the instruction pipeline to deviate from its normal operation.
- **Resource conflicts** caused by access to memory by two segments at the same time. Most of these conflicts can be resolved by using separate instruction and data memories.
- **Data dependency** conflicts arise when an instruction depends on the result of a previous instruction, but this result is not yet available.
- **Branch difficulties** arise from branch and other instructions that change the value of PC.

Vector Processing :-

- **Vector processing** is a procedure for speeding the processing of information by a computer, in which pipelined units perform arithmetic operations on uniform, linear arrays of data values, and a single instruction involves the execution of the same operation on every element of the array.
- There is a class of computational problems that are beyond the capabilities of a conventional computer. These problems are characterized by the fact that they require a vast number of computations that will take a conventional computer days or even weeks to complete.

Application Areas of Vector Processing

- Long-range weather forecasting
- Petroleum explorations
- Seismic data analysis
- Medical diagnosis
- Aerodynamics and space flight simulations
- Artificial intelligence and expert systems
- Mapping the human genome
- Image processing

Vector Operations :-

- Many scientific problems require arithmetic operations on large arrays of numbers. These numbers are usually formulated as vectors and matrices of floating-point numbers.
- A **vector** is an ordered set of a one-dimensional array of data items. A vector V of length n is represented as a row vector by V = [V1 , V2 , V3, · · · Vn].
- A conventional sequential computer is capable of processing operands one at a time. Consequently, operations on vectors must be broken down into single computations with subscripted variables. The element Vi of vector V is written as V(I) and the index I refers to a memory address or register where the number is stored.

- To examine the difference between a conventional scalar processor and a vector processor, consider the following Fortran DO loop:

```
      DO 20 I = 1, 100
20    C(I) = B(I) + A(I)
```

- This is a program for adding two vectors A and B of length 100 to produce a vector C.
- A computer capable of vector processing eliminates the overhead associated with the time it takes to fetch and execute the instructions in the program loop. It allows operations to be specified with a single vector instruction of the form

$$C(1 : 100) = A(1 : 100) + B(1: 100)$$

- The vector instruction includes the initial address of the operands, the length of the vectors, and the operation to be performed, all in one composite instruction.

Conventional computer

```
      Initialize I = 0
20    Read A(I)
      Read B(I)
      Store C(I) = A(I) + B(I)
      Increment I = i + 1
      If I ≤ 100 goto 20
```

Matrix Multiplication:-

- **Matrix multiplication** is one of the most computational intensive operations performed in computers with vector processors.
- Consider, for example, the multiplication of two 3 x 3 matrices A and B.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

- For example, the number in the first row and first column of matrix C is calculated by letting i = 1, j = 1, to obtain

$$c_{11} = a_{11}\,b_{11} + a_{12}\,b_{21} + a_{13}\,b_{31}$$

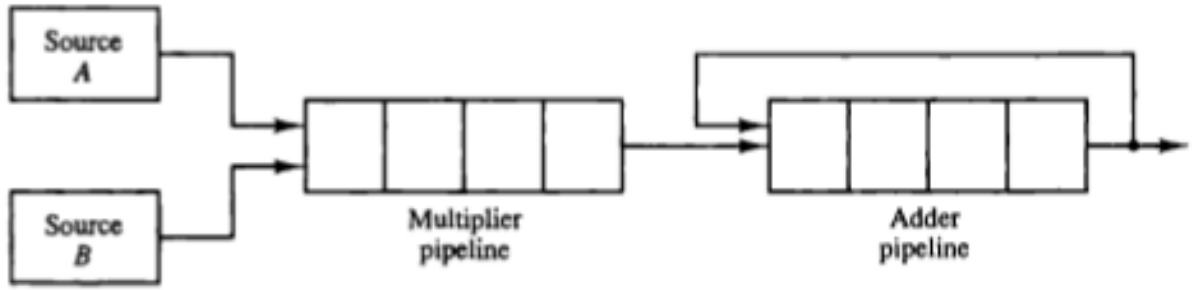
Inner Product:-

- In general, the inner product consists of the sum of k product terms of the form

$$C = A_1 B_1 + A_2 B_2 + A_3 B_3 + A_4 B_4 + \cdots + A_k B_k$$

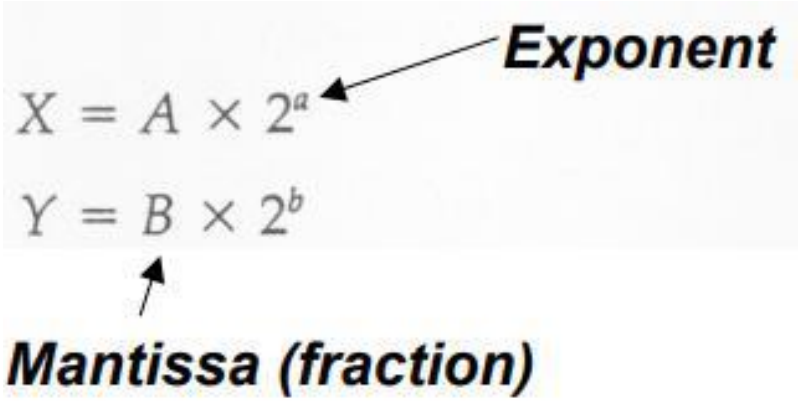
- In a typical application k may be equal to 100 or even 1000. The inner product calculation on a pipeline vector processor is shown below:

$$\begin{aligned} C = {} & A_1 B_1 + A_5 B_5 + A_9 B_9 + A_{13} B_{13} + \cdots \\ & + A_2 B_2 + A_6 B_6 + A_{10} B_{10} + A_{14} B_{14} + \cdots \\ & + A_3 B_3 + A_7 B_7 + A_{11} B_{11} + A_{15} B_{15} + \cdots \\ & + A_4 B_4 + A_8 B_8 + A_{12} B_{12} + A_{16} B_{16} + \cdots \end{aligned}$$



Arithmetic Pipeline :-

- **Pipeline arithmetic** units are usually found in very high speed computers. They are used to implement floating-point operations, multiplication of fixed-point numbers, and similar computations encountered in scientific problems.
- Let's take an example of a pipeline unit for floating-point addition and subtraction. The inputs to the floating-point adder pipeline are two normalized floating-point binary numbers.



4-Segment Pipeline :

1. Compare the exponents.
2. Align the mantissas.
3. Add or subtract the mantissas.
4. Normalize the result.

Floating point ADD/SUB Example :-

$$X = 0.9504 \times 10^3$$

$$Y = 0.8200 \times 10^2$$

The two exponents are subtracted in the first segment to obtain $3 - 2 = 1$. The larger exponent 3 is chosen as the exponent of the result. The next segment shifts the mantissa of Y to the right to obtain

$$X = 0.9504 \times 10^3$$

$$Y = 0.0820 \times 10^3$$

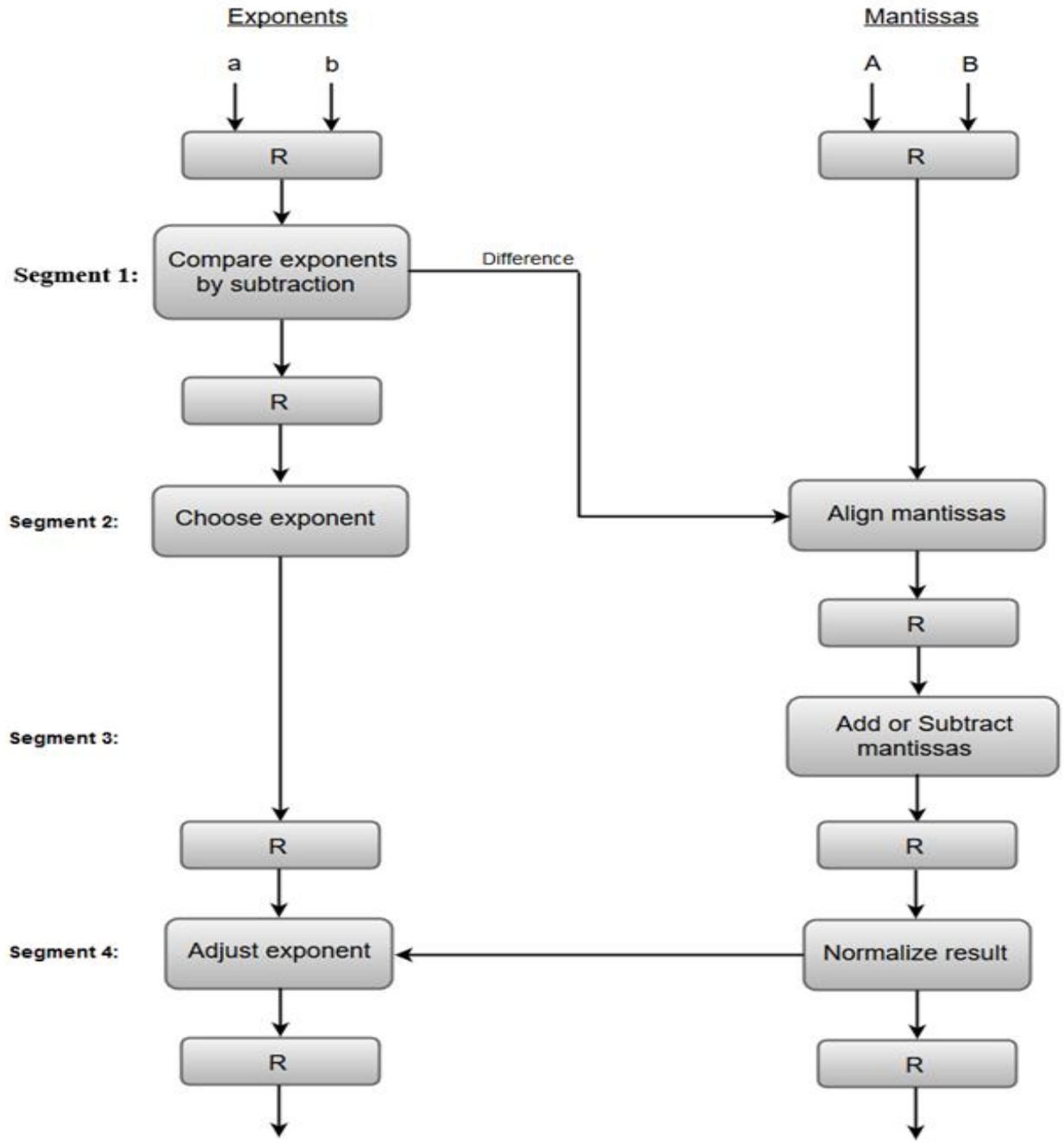
This aligns the two mantissas under the same exponent. The addition of the two mantissas in segment 3 produces the sum

$$Z = 1.0324 \times 10^3$$

The sum is adjusted by normalizing the result so that it has a fraction with a nonzero first digit. This is done by shifting the mantissa once to the right and incrementing the exponent by one to obtain the normalized sum.

$$Z = 0.10324 \times 10^4$$

Pipeline organization for floating point addition and subtraction:



Multiprocessor System :-

- A **multiprocessor** is a computer system with two or more central processing units (CPUs), with each one sharing the common main memory as well as the peripherals. This helps in simultaneous processing of programs.
- The key objective of using a multiprocessor is to boost the system's execution speed, with other objectives being fault tolerance and application matching.
- A multiprocessor is regarded as a means to improve computing speeds, performance and cost-effectiveness, as well as to provide enhanced availability and reliability.

- **Characteristics:**
  - Consists of more than one CPU.
  - Fast processing.
  - Reliability
  - Cost – Effective
  - Simultaneous processing of programs.

Interconnection Structures for Multiprocessor System :-

- The components that form a multiprocessor system are CPUs, IOPs(Input Output Processors) connected to input output devices, and a memory unit.
- There are several physical forms available for establishing an interconnection network.
  - Time-shared common bus
  - Multiport memory
  - Crossbar switch
  - Multistage switching network

1. Time Shared Common Bus :-

- A common-bus multiprocessor system consists of a number of processors connected through a common path to a memory unit.

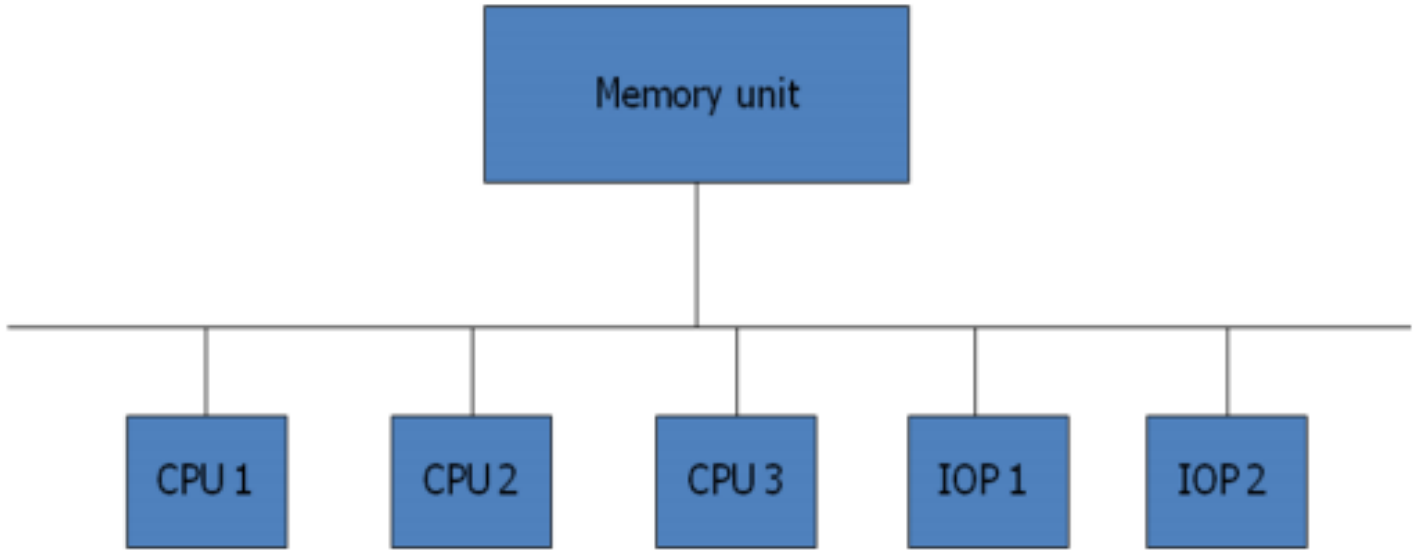


Fig: Time shared common bus organization

2. Multiport Memory

- A multiport memory system employs separate buses between each memory module and each CPU.

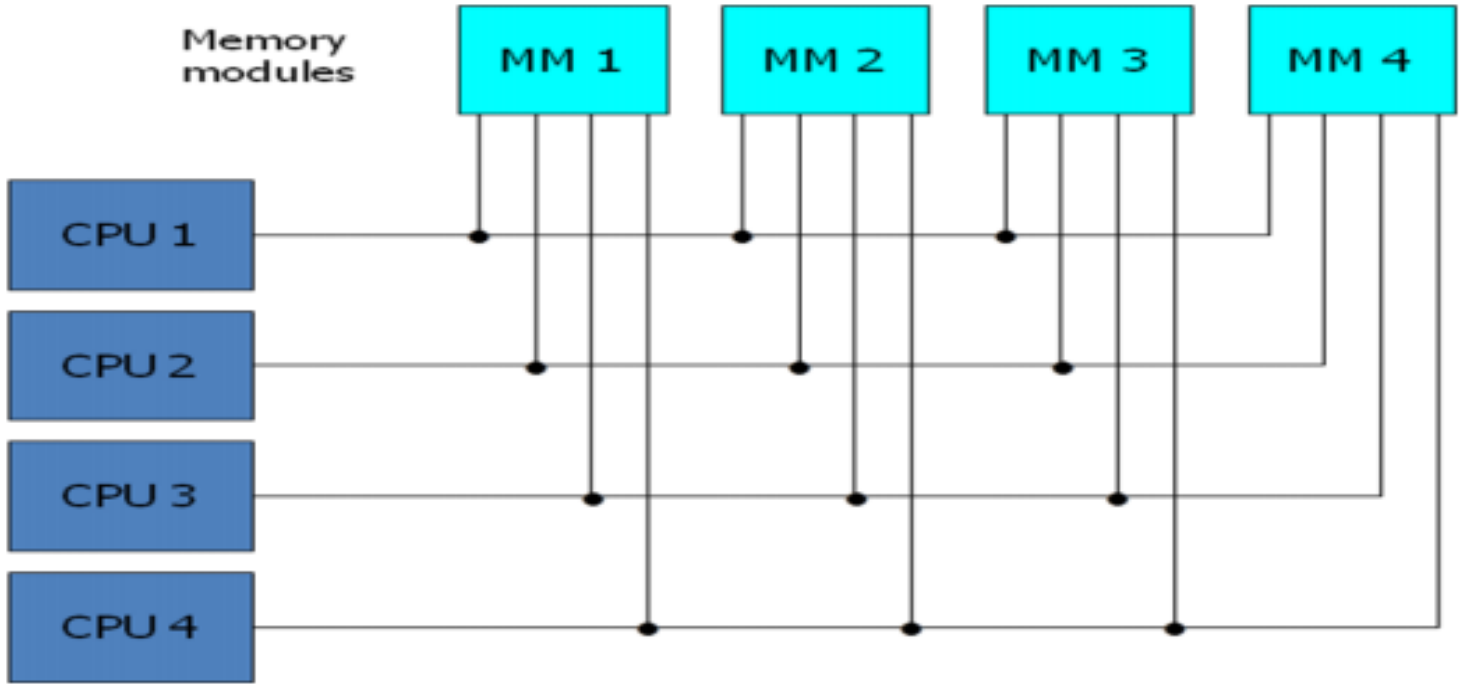


Fig: Multiport memory organization

3. Crossbar Switch

- Consists of a number of cross points that are placed at intersections between processor buses and memory module paths.

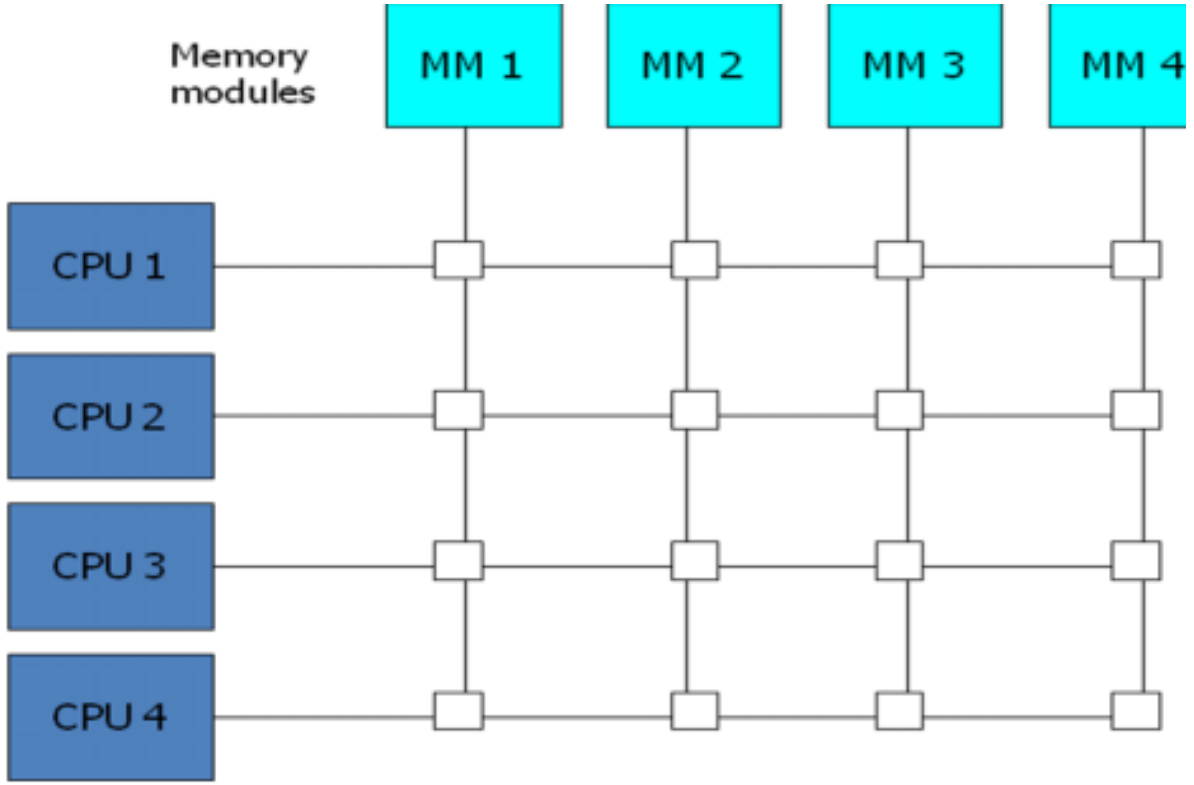


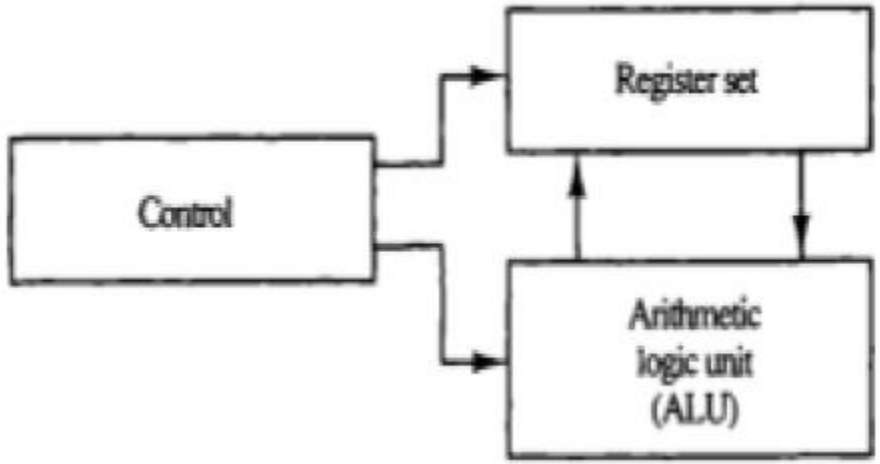
Fig: Crossbar switch

Unit-5
Central Processing Unit

Compiled by :- Er Nagendra karn

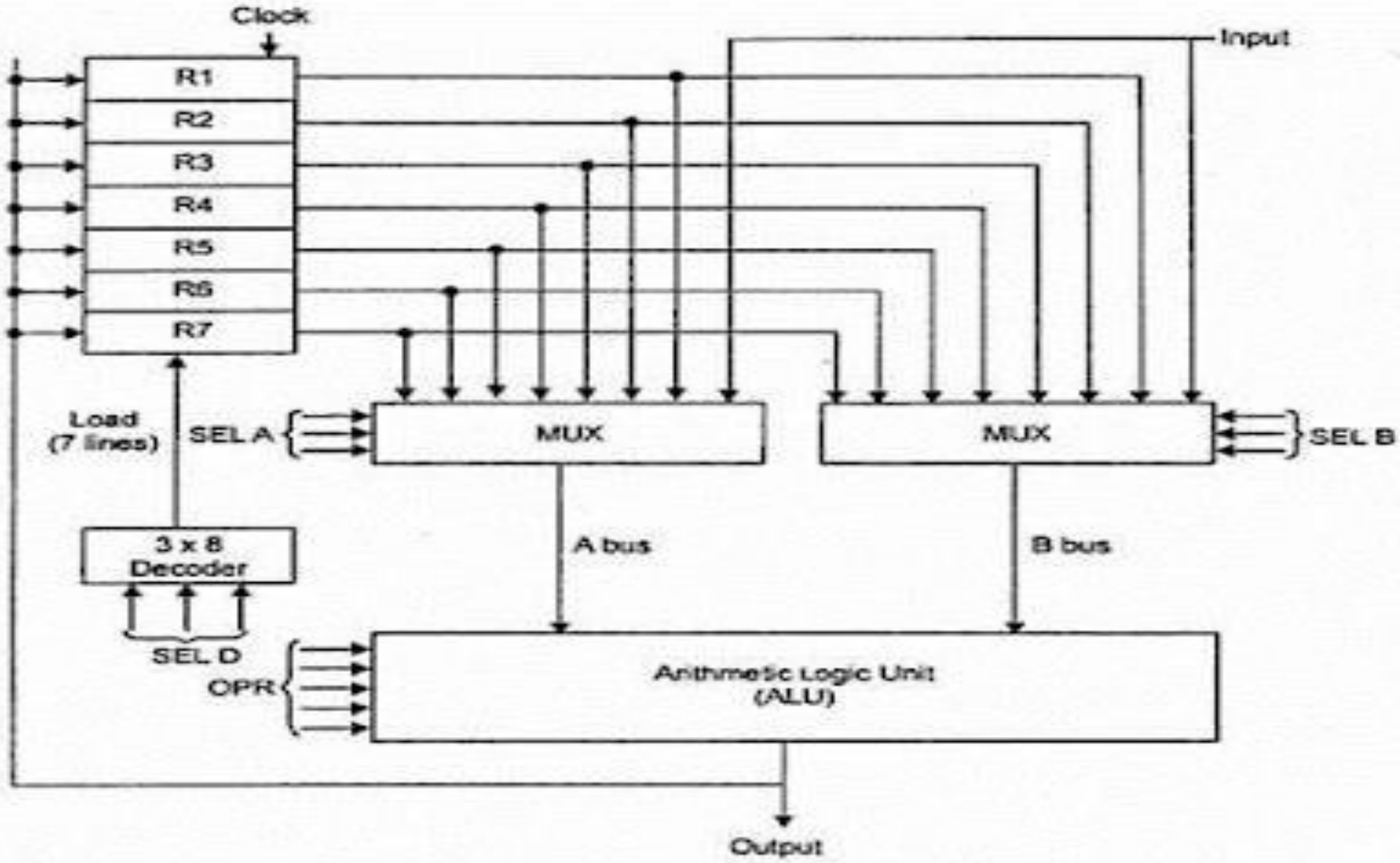
Introduction:-

- The part of the computer that performs the bulk of data-processing operations is called the central processing unit and is referred to as the CPU.
- The CPU is made up of three major parts, as shown in Fig.
  - ➢ The register set stores intermediate data used during the execution of the instructions.
  - ➢ The arithmetic logic unit (ALU) performs the required micro operations for executing the instructions.
  - ➢ The control unit supervises the transfer of information among the registers and instructs the ALU as to which operation to perform.



General Register Organization :-

- A set of flip-flops forms a register. A register is a unique high-speed storage area in the CPU.
- They include combinational circuits that implement data processing. The information is always defined in a register before processing. The registers speed up the implementation of programs.
- When a large number of registers are included in the CPU, it is most efficient to connect them through a common bus system.
- The registers communicate with each other not only for direct data transfers, but also while performing various micro operations.
- The CPU bus system is managed by the control unit. The control unit explicit the data flow through the ALU by choosing the function of the ALU and components of the system.


Clock
Input
R1
R2
R3
R4
R5
R6
R7
Load
(7 lines)
SEL A
MUX
MUX
SEL B
3 x 8
Decoder
A bus
B bus
SEL D
OPR
Arithmetic Logic Unit
(ALU)
Output

- The control unit that operates the CPU bus system directs the information flow through the registers and ALU by selecting the various components in the system. For example, to perform the operation

  $R1 \leftarrow R2+R3$

- the control must provide binary selection variables to the following selector inputs:
- **MUX A Selector (SELA)** − It can place R2 into bus A.
- **MUX B Selector (SELB)** − It can place R3 into bus B.
- **ALU Operation Selector (OPR)** − It can select the arithmetic addition (ADD).
- **Decoder Destination Selector (SELD)** − It can transfers the result into R1.
- The multiplexers of 3-state gates are performed with the buses. The state of 14 binary selection inputs determines the control word. The 14-bit control word defines a micro-operation.

Control Word :-

- There are 14 binary selection inputs in the unit, and their combined value specifies a control word.

FORMATE OF CONTROL WORD

- The three bit of SELA select a source registers of the **a** input of the ALU.
- The three bits of SELB select a source registers of the **b** input of the ALU.
- The three bits of SELED or SELREG select a destination register using the decoder.
- The four bits of SELOPR select the operation to be performed by ALU.
- A control word of 14 bits is needed to specify a micro operation in the CPU. The control word for a given micro operation can be derived from the selection variables.

Encoding of Register Selection Field:-

Binary code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

For example, the subtract micro operation given by the statement

R1←R2-R3

specifies R2 for the A input of the ALU, R3 for the B input of the ALU, R1 for the destination register, and an ALU operation to subtract A - B. Thus the control word is specified by the four fields and the corresponding binary value for each field is obtained from the encoding.

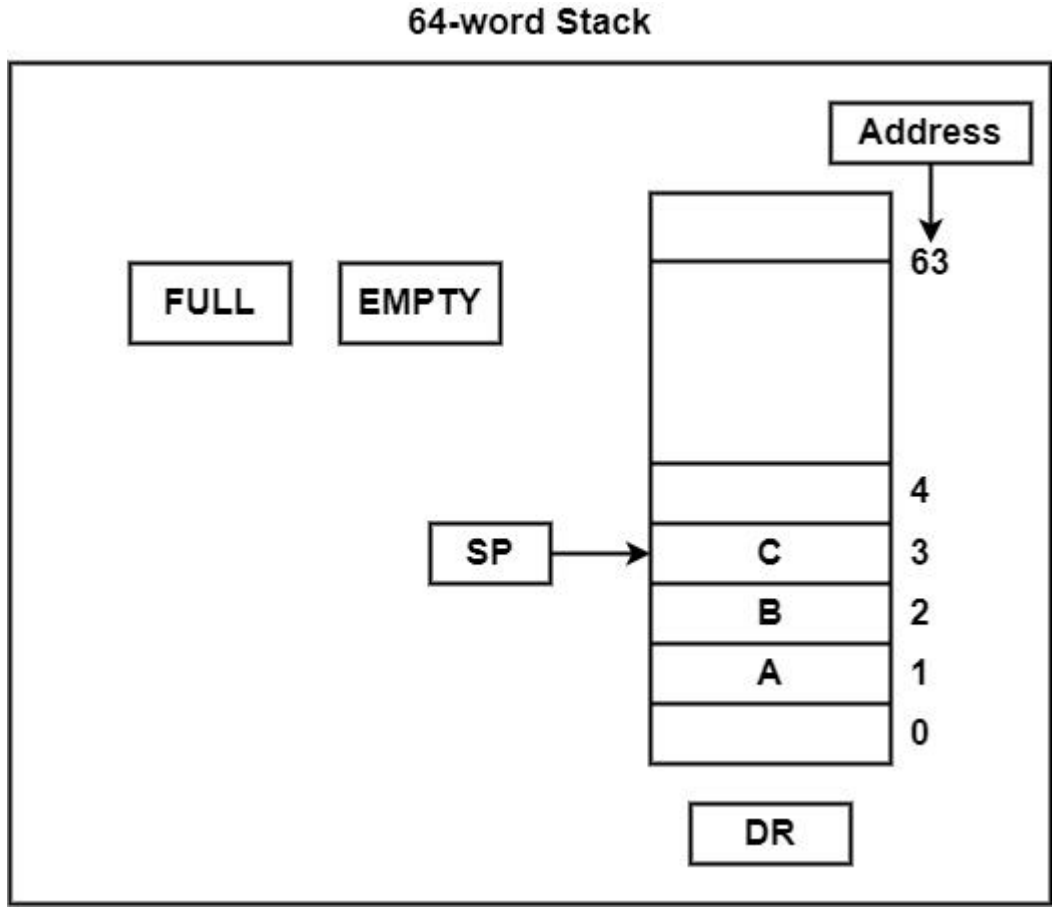
OPR Select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	ADD A+B	ADD
00101	SUB A-B	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift Right A	SHRA
11000	Shift left A	SHLA

Stack Organization :-

- Stack is also known as the Last In First Out (LIFO) list. It is the most important feature in the CPU. It saves data such that the element stored last is retrieved first.
- A stack is a memory unit with an address register. This register influence the address for the stack, which is known as Stack Pointer (SP). The stack pointer continually influences the address of the element that is located at the top of the stack.
- It can insert an element into or delete an element from the stack. The insertion operation is known as push operation and the deletion operation is known as pop operation. In a computer stack, these operations are simulated by incrementing or decrementing the SP register.

Register Stack :-

- A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or registers. Figure shows the organization of a 64-word register stack.

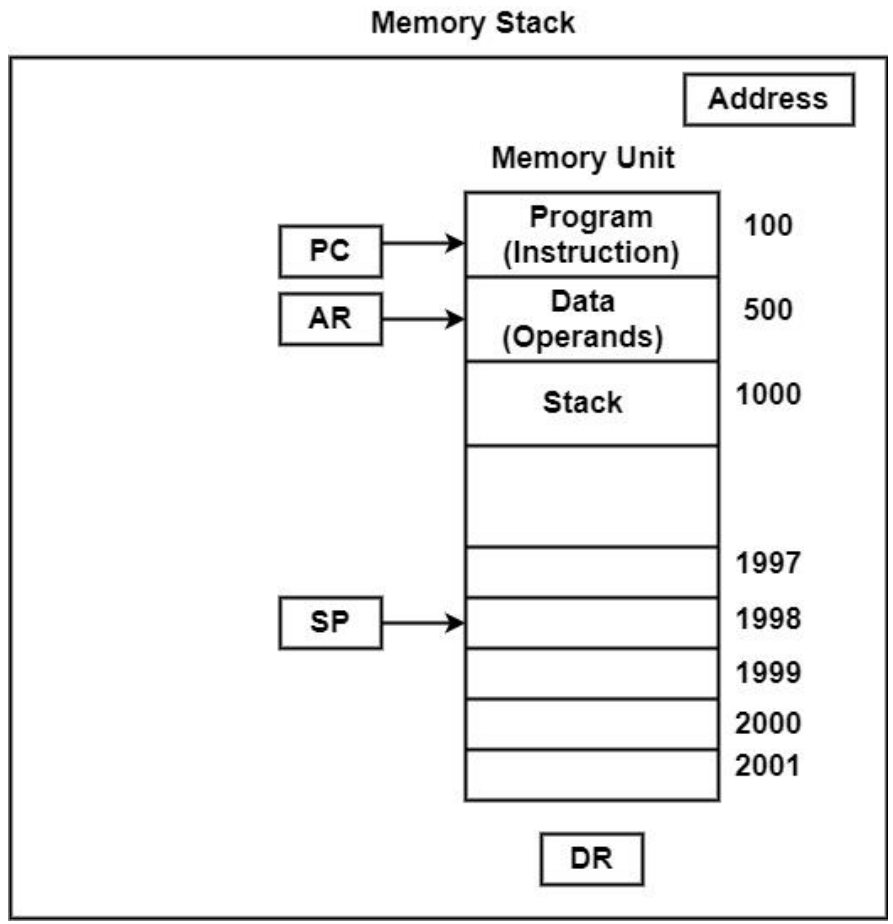


- The stack pointer register SP contains a binary number whose value is equal to the address of the word that is currently on top of the stack. Three items are placed in the stack: A, B, and C, in that order. Item C is on top of the stack so that the content of SP is now 3.
- To remove the top item, the stack is popped by reading the memory word at address 3 and decrementing the content of SP. Item B is now on top of the stack since SP holds address 2.
- To insert a new item, the stack is pushed by incrementing SP and writing a word in the next-higher location in the stack.
- Initially, SP is cleared to 0, EMTY is set to 1, and FULL is cleared to 0, so that SP points to the word at address 0 and the stack is marked empty and not full. If the stack is not full (if FULL = 0), a new item is inserted with a push operation.

- The push operation is implemented with the following sequence of micro operations;
- SP <- SP + 1 Increment stack pointer
- M[SP]<-DR Write item on top of the stack
- A new item is deleted from the stack if the stack is not empty (if EMTY = 0). The pop operation consists of the following sequence of micro operations:
- DR <--M[SP] Read item from the top of stack
- SP<--SP - 1 Decrement stack pointer

Memory Stack :-

- A stack can be implemented in a random-access memory attached to a CPU.
- The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer.
- The portion of computer memory is partitioned into three segments: program, data, and stack.
- The program counter PC points at the address of the next instruction in the program. The address register AR points at an array of data. The stack pointer SP points at the top of the stack.
- PC is used during the fetch phase to read an instruction. AR is used during the execute phase to read an operand. SP is used to push or pop items into or from the stack.


Memory Stack
Address
Memory Unit
PC
AR
SP
Program
(Instruction)
Data
(Operands)
Stack
100
500
1000
1997
1998
1999
2000
2001
DR

- In the figure, the SP points to a beginning value ‘2001’. Therefore, the stack increase with decreasing addresses. The first element is saved at address 2000, the next element is saved at address 1999 and the last element is saved at address 1000.
- The data register can read an element into or from the stack. It can use push operation to insert a new element into the stack.
    - SP<-SP - 1
    - M[SP]<-DR
- A new item is deleted with a pop operation as follows:
    - DR <-M[SP]
    - SP<-SP + 1
- The top item is read from the stack into DR. The stack pointer is then incremented to point at the next item in the stack.

Program Interrupt:-

- Program interrupt refers to the transfer of program control from a currently running program to another service program as a result of an external or internal generated request.
- Control returns to the original program after the service program is executed. The interrupt procedure is, in principle, quite similar to a subroutine call except for three variations:
- The interrupt is usually initiated by an internal or external signal rather than from the execution of an instruction (except for software interrupt);
- The address of the interrupt service program is determined by the hardware rather than from the address field of an instruction; and
- an interrupt procedure usually stores all the information necessary to define the state of the CPU rather than storing only the program counter.

Types of Interrupt :-

- There are three major types of interrupts that cause a break in the normal execution of a program. They can be classified as:
- **External Interrupt :-** External interrupts come from input/output (I/0) devices, from a timing device, from a circuit monitoring the power supply, or from any other external source. Examples that cause external interrupts are l/0 device requesting transfer of data, l/0 device finished transfer of data, elapsed time of an event, or power failure.
- **Internal Interrupt :-** Internal interrupts arise from illegal or erroneous use of an instruction or data. Internal interrupts are also called **traps**. Examples of interrupts caused by internal error conditions are register overflow, attempt to divide by zero, an invalid operation code, stack overflow, and protection violation.
- The difference between internal and external interrupts is that the internal interrupt is initiated by some exceptional condition caused by the program itself rather than by an external event.

Software Interrupts:-

- External and internal interrupts are initiated from signals that occur in the hardware of the CPU.
- A software interrupt is initiated by executing an instruction. Software interrupt is a special call instruction that behaves like an interrupt rather than a subroutine call.
- The most common use of software interrupt is associated with a supervisor call instruction. This instruction provides means for switching from a CPU user mode to the supervisor mode.
- Certain operations in the computer may be assigned to the supervisor mode only, as for example, a complex input or output transfer procedure.
- A program written by a user must run in the user mode. When an input or output transfer is required, the supervisor mode is requested by means of a supervisor call instruction. This instruction causes a software interrupt.

Instruction Format :-

- Computer perform task on the basis of instruction provided. **An instruction format** defines layout of bits of an instruction.
- The most common fields are:
  - Operation field which specifies the operation to be performed like addition.
  - Address field which contain the location of operand, i.e., register or memory location.
  - Mode field which specifies how operand is to be founded.
- An instruction is of various length depending upon the number of addresses it contain. Generally, CPU organization are of three types on the basis of number of address fields:
- Single Accumulator organization
- General register organization
- Stack organization

1. Single Accumulator Organization :-

- All operations are performed with an accumulator register. The instruction format in this type of computer uses one address field. e.g. **ADD X**

2. General Register Organization :-

The instruction format in this type of computer needs three register address fields. e.g. **ADD R1,R2,R3** or **ADD R1,R2**.

3. Stack Organization :-

Computers with stack organization would have PUSH and POP instructions which require an address field. e.g. **PUSH X, ADD**

Instruction Format :-

- To illustrate the influence of the number of addresses on computer programs, we will evaluate the arithmetic statement

$$X = (A + B) * (C + D)$$

using zero, one, two, or three address instructions.

- **Three Address Instruction :-**
- Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand.

ADD R1, A, B	**R1←M[A] + M[B]**
ADD R2, C, D	**R2←M[C] + M[D]**
MUL X, R1, R2	**M[X] ←R1*R2**

- The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions. The disadvantage is that the binary-coded instructions require too many bits to specify three addresses.

Two-Address Instruction :-

- Two-address instructions are the most common in commercial computers. Here again each address field can specify either a processor register or a memory word.

MOV R1, A	**R1←M[A]**
ADD R1, B	**R1←R1 + M[B]**
MOV R2, C	**R2←M[C]**
ADD R2, D	**R2←R2 + M[D]**
MUL R1, R2	**R1←R1*R2**
MOV X, R1	**M[X] ←R1**

One-Address Instruction :-

- One-address instructions use an accumulator (AC) register for all data manipulation. here we will neglect the second register and assume that the AC contains the result of all operations.

LOAD A	**AC←M[A]**
ADD B	**AC←AC+M[B]**
STORE T	**M[T]←AC**
LOAD C	**AC←M[C]**
ADD D	**AC←AC+M[D]**
MUL T	**AC ←AC*M[T]**
STORE X	**M[X] ←AC**

- All operations are done between the AC register and a memory operand.

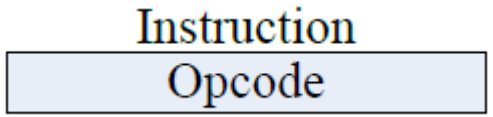
Zero-Address Instruction :-

- A stack-organized computer does not use an address field for the instructions **ADD** and **MUL**. The **PUSH** and **POP** instructions, however, need an address field to specify the operand that communicates with the stack.

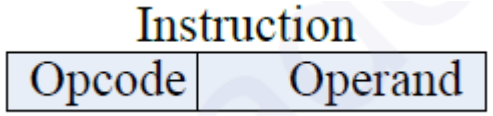
PUSH A	**TOS←A**
PUSH B	**TOS ←B**
ADD	**TOS ←(A+B)**
PUSH C	**TOS ←C**
PUSH D	**TOS ←D**
ADD	**TOS ←(C+D)**
MUL	**TOS ←(C+D)*(A+B)**
POP X	**M[X] ←TOS**

Addressing Mode :-

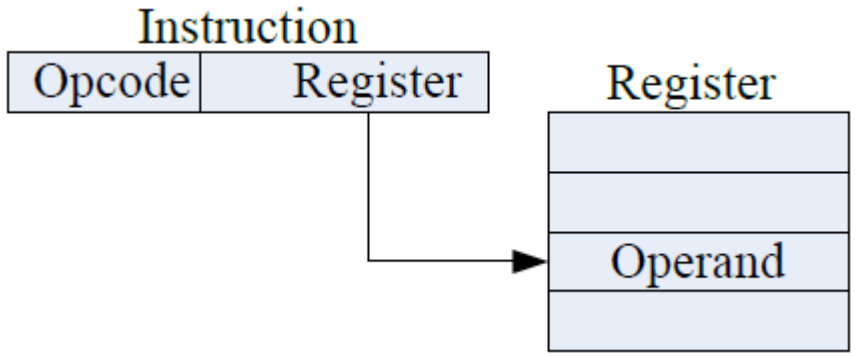
- **Implied Mode :-** The operand is hidden, and data to be operated is available in the instruction itself. e.g. CMA.
- All register reference instructions that use an accumulator are implied-mode instructions. Zero-address instructions in a stack-organized computer are implied-mode instructions since the operands are implied to be on top of the stack.



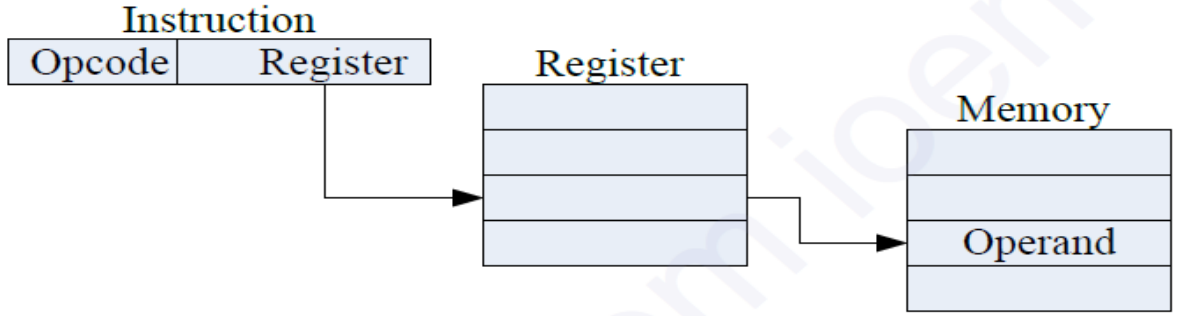
- **Immediate Mode :-** In this mode the operand is specified in the instruction itself. In other words, an immediate-mode instruction has an operand field rather than an address field.



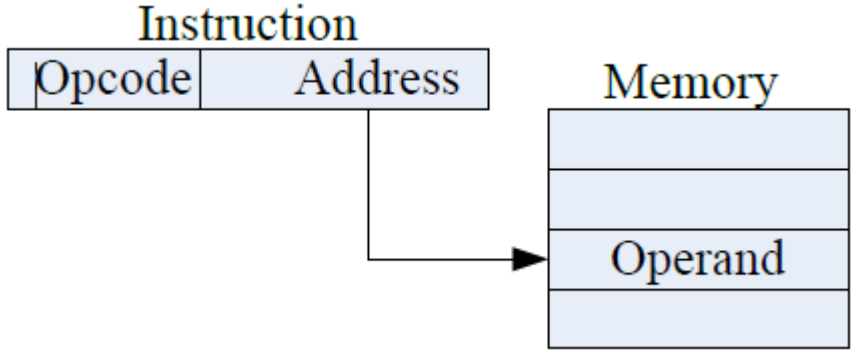
- **Register Direct Addressing Mode :-**
- In this mode the operands are in registers that reside within the CPU. The particular register is selected from a register field in the instruction. e.g. MOV A,B



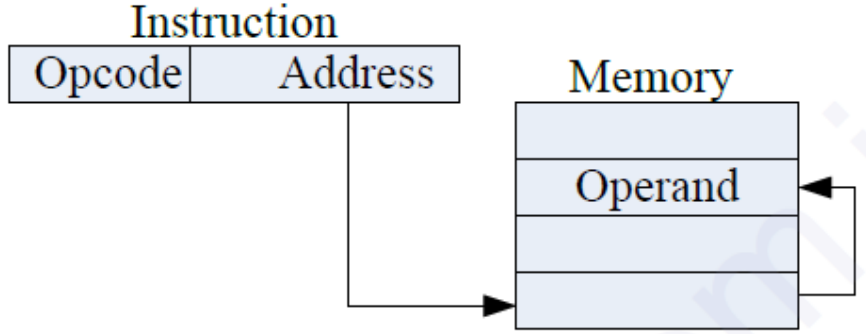
- **Register Indirect Addressing Mode :**
- In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory. In other words, the selected register contains the address of the operand rather than the operand itself.



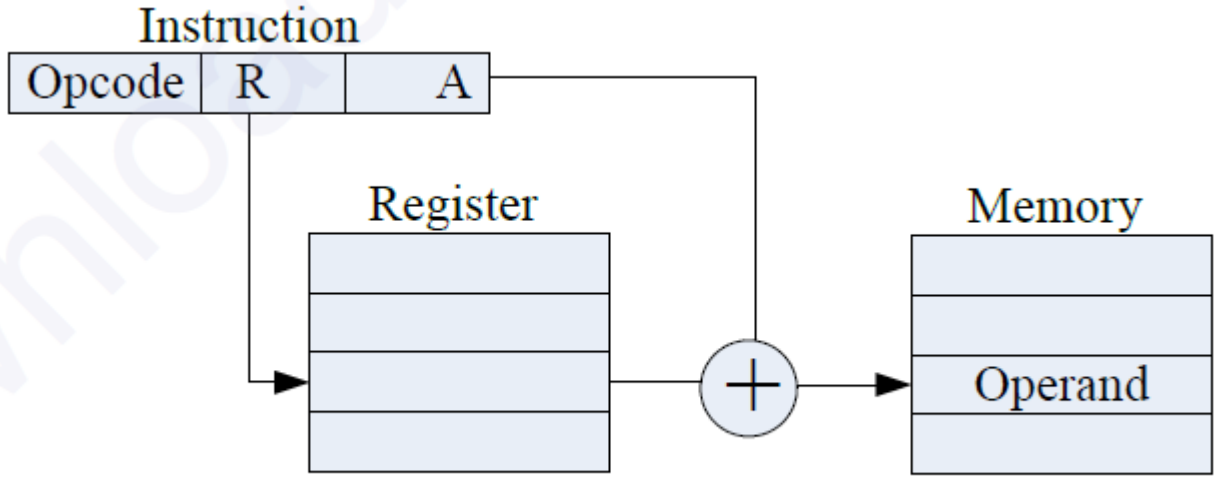
- **Auto Increment or Auto Decrement mode :-**
- This is similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory.
- After accessing the operand, the contents of this register are automatically incremented to the next value.
- Before accessing the operand, the contents of this register are automatically decremented and then the value is accessed.
- **Direct Addressing Mode :-**
- Here, the operand resides in memory and its address is given directly by the address field of the instruction. In a branch-type instruction the address field specifies the actual branch address.



- **Indirect Addressing Mode :-**
- In this mode the address field of the instruction gives the address where the effective address is stored in memory.



- **Displacement Addressing Mode :-**
- A very powerful mode of addressing combines the capabilities of direct addressing and register indirect addressing.
- The address field of instruction is added to the content of specific register in the CPU.



- **Relative addressing Mode :-** In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address.
- **Indexed addressing Mode :-** In this mode the content of an index register is added to the address part of the instruction to obtain the effective address. The index register is a special CPU register that contains an index value.
- **Base Register addressing Mode :-** In this mode the content of a base register is added to the address part of the instruction to obtain the effective address.
- This is similar to the indexed addressing mode except that the register is now called a base register instead of an index register.

Data Transfer Manipulation :-

- Data transfer instructions cause transfer of data from one location to another without changing the binary information. The most common transfer are between the
  - Memory and Processor registers
  - Processor registers and input output devices
  - Processor registers themselves
- **Typical Data transfer Instructions :-**

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
PUSH	PUSH
Pop	POP

Data Manipulation Instructions :-

- Data manipulation instructions perform operations on data and provide the computational capabilities for the computer. These instructions perform arithmetic, logic and shift operations.
- Arithmetic Instructions :-

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

- **Logical and Bit manipulation Instructions :-**
- Logical instructions perform binary operations on strings of bits stored in registers.
- They are useful for manipulating individual bits or a group of bits that represent binary-coded information.

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear Carry	CLRC
Set carry	SETC
Complement Carry	COMC
Enable interrupt	EI
Disable interrupt	DI

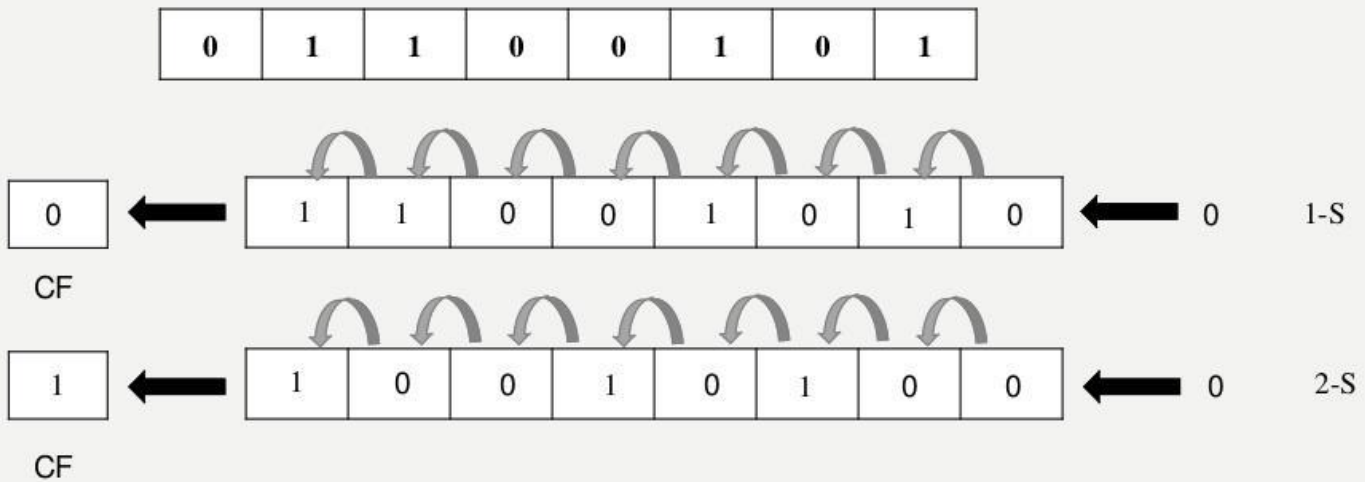
Shift Operations :-

- Shifts are operations in which the bits of a word are moved to the left or right.
- The bit shifted in at the end of the word determines the type of shift used.
- Shift instructions may specify either **logical shifts, arithmetic shifts, or rotate-type** operations.

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through Carry	RORC
Rotate left through Carry	ROLC

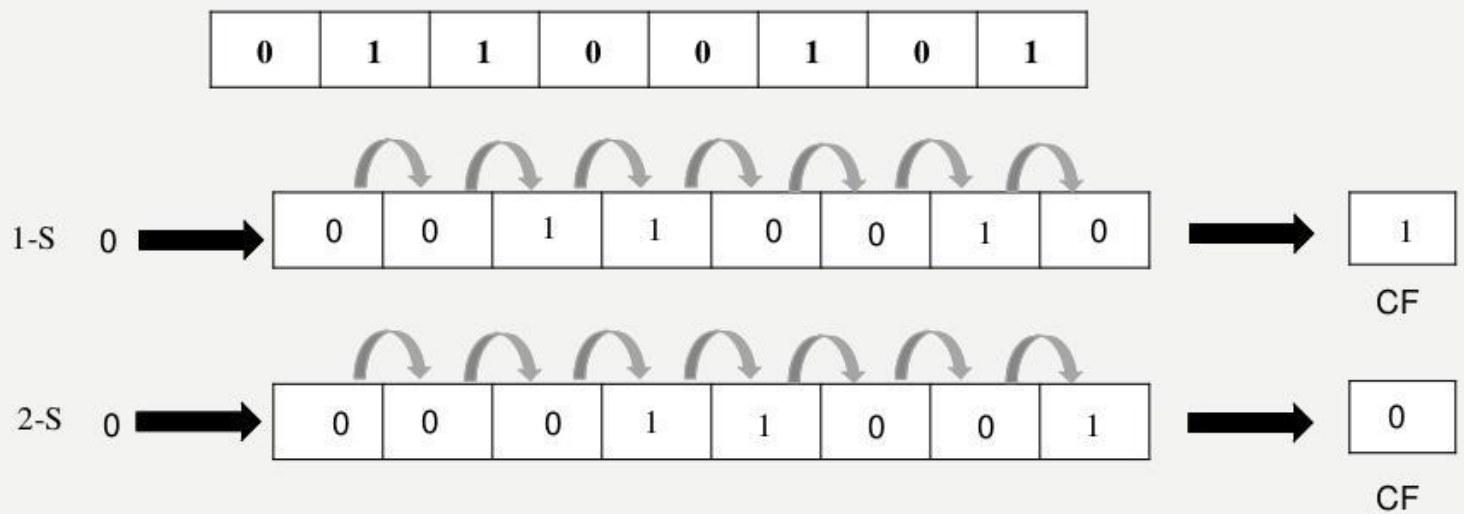
SHIFT INSTRUCTION: SHL AND SAL

- The SHL (shift left) instruction shifts the bits in the destination to the left.
- A 0 is shifted into the right most bit position and the msb is shifted into CF.



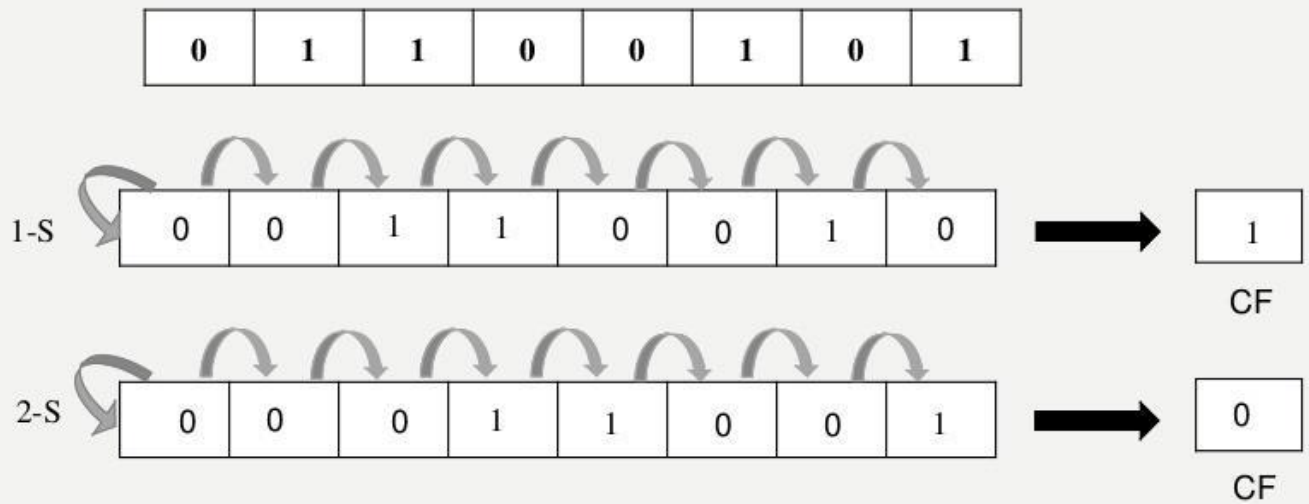
SHIFT INSTRUCTION: SHR

- The instruction SHR(shift right) performs right shift on the destination operand.
- A 0 is shifted into the msb position ,and the rightmost bit is shifted into CF.



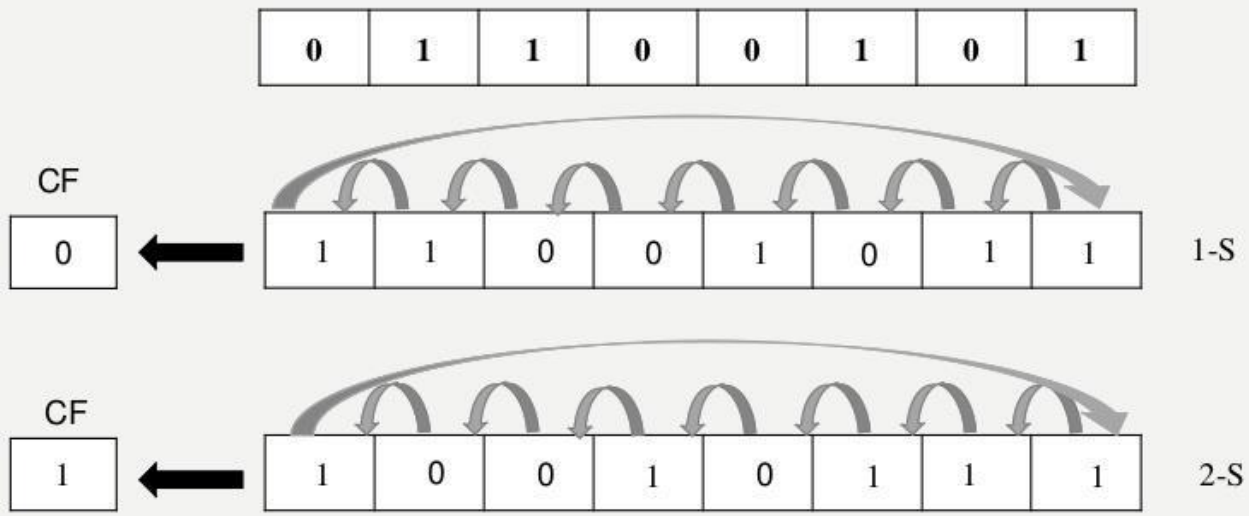
SHIFT INSTRUCTION: SAR

- The SAR instruction (shift arithmetic right) operates like SHR, with one difference: the msb retains its original value.



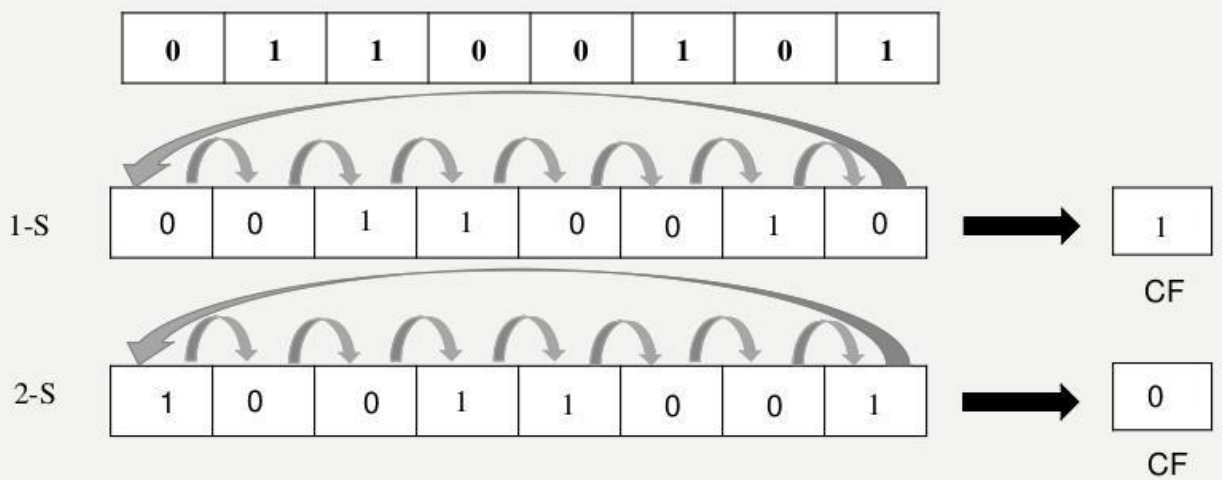
ROTATE INSTRUCTION: ROL

- The instruction ROL (rotate left)shifts bits to the left .The msb is shifted into the rightmost bit . The CF also gets the bit shifted out of the msb .



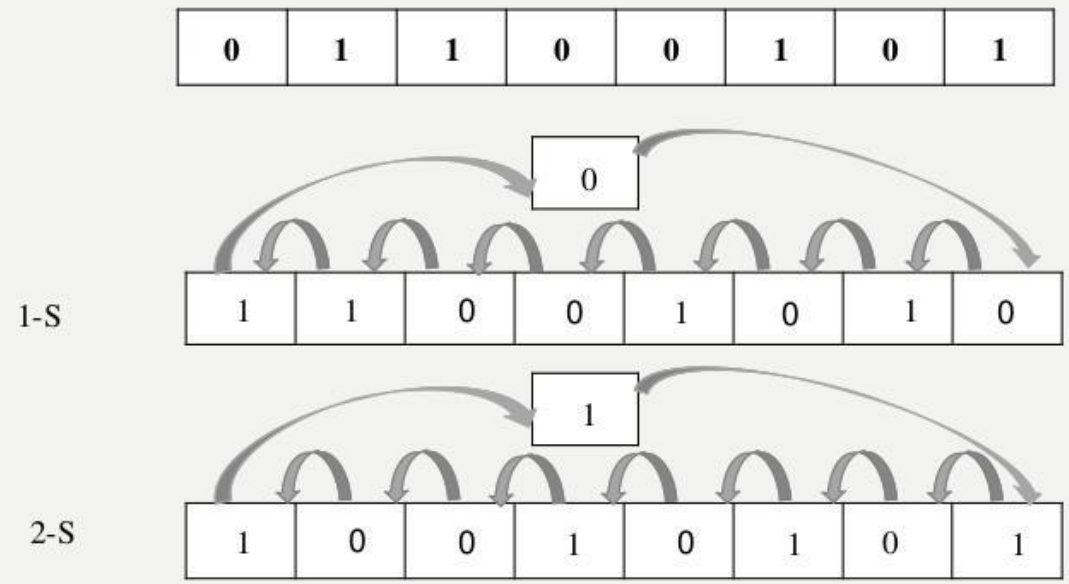
ROTATE INSTRUCTION: ROR

- The instruction ROR (rotate right) works just like ROL , except that the bits are rotate to the right .The rightmost bit is shifted into the msb , and also into The CF.



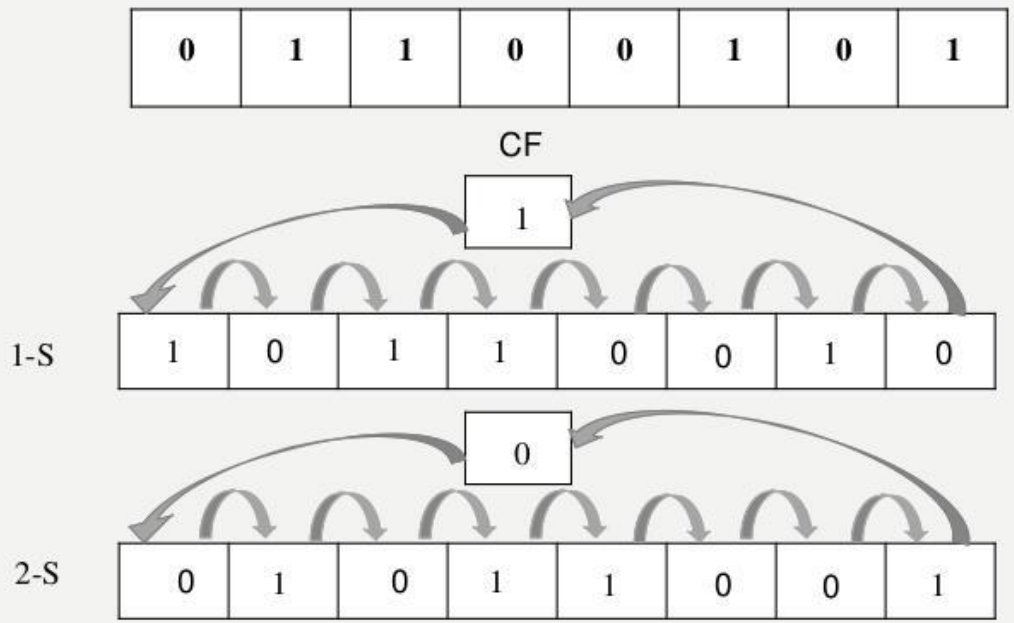
ROTATE INSTRUCTION: RCL

- The instruction **RCL** (Rotate through carry left) shifts the bits of the destination to the left. The msb is shifted into the CF, and the previous value of CF is shifted into the rightmost bit.



ROTATE INSTRUCTION: RCR

The instruction RCR(Rotate through carry right) works just like RCL, except that the bits are rotated to the right.



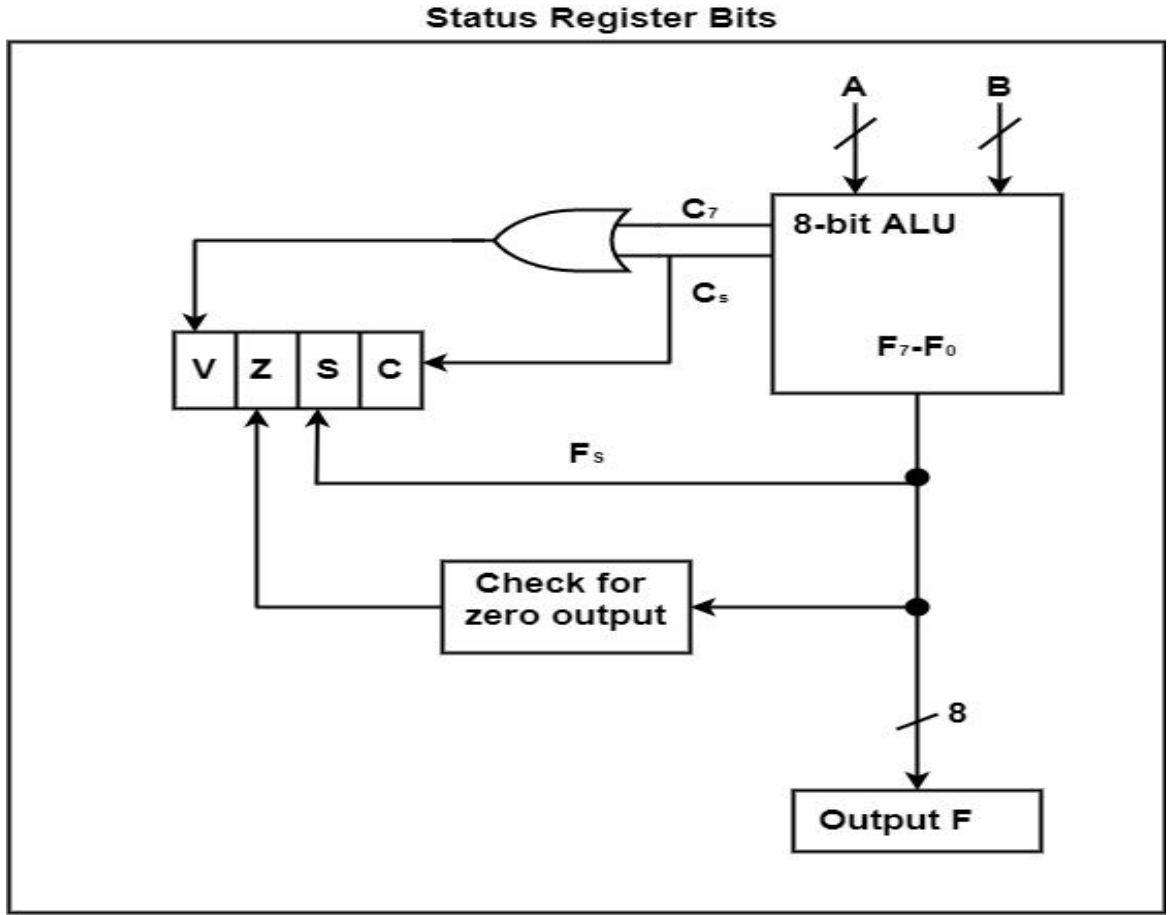
Program Control Instructions :-

- The program control instructions provide decision making capabilities and change the path taken by the program when executed in computer.
- These instructions specify conditions for altering the content of the program counter. The change in value of program counter as a result of execution of program control instruction causes a break in sequence of instruction execution.
- This is an important feature in **digital computers**, as it provides control over the flow of program execution and a capability for branching to different program segments.
- Some program control instructions are:

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP
Test (by Anding)	TST

Status Bit Conditions :-

- **Status bits** are also called **condition-code bits** or **flag bits**.
- Figure below shows the block diagram of an 8-bit ALU with a 4-bit status register. The four status bits are symbolized by C, S, Z, and V.



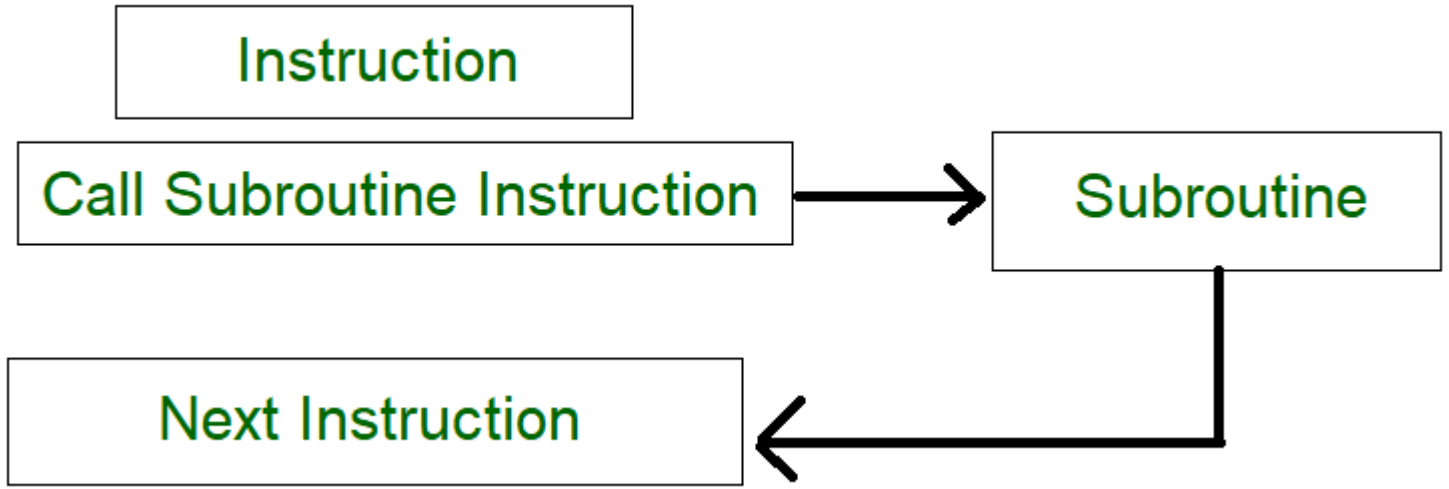
- The bits are set or cleared as a result of an operation performed in the ALU.
- **Bit C (carry)** is set to 1 if the end carry C8 is 1. It is cleared to 0 if the carry is 0.
- **Bit S (sign)** is set to 1 if the highest-order bit F, is 1. It is set to 0 if the bit is 0.
- **Bit Z (zero)** is set to 1 if the output of the ALU contains all O's. !tis cleared to 0 otherwise. In other words, Z = 1 if the output is zero and Z = 0 if the output is not zero.
- **Bit V (overflow)** is set to 1 if the exclusive-OR of the last two carries is equal to 1, and cleared to 0 otherwise.

Conditional Branch Instructions :-

Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	$Z = 1$
BNZ	Branch if not zero	$Z = 0$
BC	Branch if carry	$C = 1$
BNC	Branch if no carry	$C = 0$
BP	Branch if plus	$S = 0$
BM	Branch if minus	$S = 1$
BV	Branch if overflow	$V = 1$
BNV	Branch if no overflow	$V = 0$
Unsigned compare conditions ($A - B$)		
BHI	Branch if higher	$A > B$
BHE	Branch if higher or equal	$A \geq B$
BLO	Branch if lower	$A < B$
BLOE	Branch if lower or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$
Signed compare conditions ($A - B$)		
BGT	Branch if greater than	$A > B$
BGE	Branch if greater or equal	$A \geq B$
BLT	Branch if less than	$A < B$
BLE	Branch if less or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$

Subroutine Call & Return :-

- A set of instructions that are used repeatedly in a program can be referred to as Subroutine.
- Only one copy of this Instruction is stored in the memory. When a Subroutine is required it can be called many times during the Execution of a particular program.
- A *call Subroutine Instruction* calls the Subroutine. Care Should be taken while returning a Subroutine as Subroutine can be called from a different place from the memory.



- A **call subroutine** instruction consists of an operation code together with an address that specifies the beginning of the subroutine. The instruction is executed by performing two operations:
  - the address of the next instruction available in the program counter (the return address) is stored in a temporary location so the subroutine knows where to return, and
  - control is transferred to the beginning of the subroutine.
- The last instruction of every subroutine, commonly called **return** from subroutine, transfers the **return address** from the temporary location into the program counter.
- This results in a transfer of program control to the instruction whose address was originally stored in the temporary location.

Reduced Instruction Set Format :-

- A computer uses fewer instructions with simple constructs so they can be executed much faster within the CPU without having to use memory as often. It is classified as a reduced instruction set computer (**RISC**).
- RISC concept – an attempt to reduce the execution cycle by simplifying the instruction set
- Small set of instructions – mostly register to register operations and simple load/store operations for memory access
- Each operand – brought into register using load instruction, computations are done among data in registers and results transferred to memory using store instruction
- Simplify instruction set and encourages the optimization of register manipulation
- May include immediate operands, relative mode etc.

Characteristics of a RISC Architecture :-

- Relatively few instructions
- Relatively few addressing modes
- Memory access limited to load and store instructions
- All operations done within the registers of the CPU
- Fixed-length, easily decoded instruction format
- Single-cycle instruction execution
- Hardwired rather than microprogrammed control

Complex Instructions Set Computer :-

- A computer with a large number of instructions is classified as a complex instruction set computer (CISC).
- One reason for the trend to provide a complex instruction set is the desire to simplify the compilation and improve the overall computer performance.
- The essential goal of CISC architecture is to attempt to provide a single machine instruction for each statement that is written in a high-level language.
- **Major Characteristics of CISC Computer :-**
- A large number of instructions-typically from 100 to 250 instructions.
- Some instructions that perform specialized tasks and are used infrequently.
- A large variety of addressing modes-typically from 5 to 20 different modes.
- Variable-length instruction formats.
- A relatively large number of registers in the processor unit.

Microprogrammed Control

Compiled by :- Er. Nagendra Karn

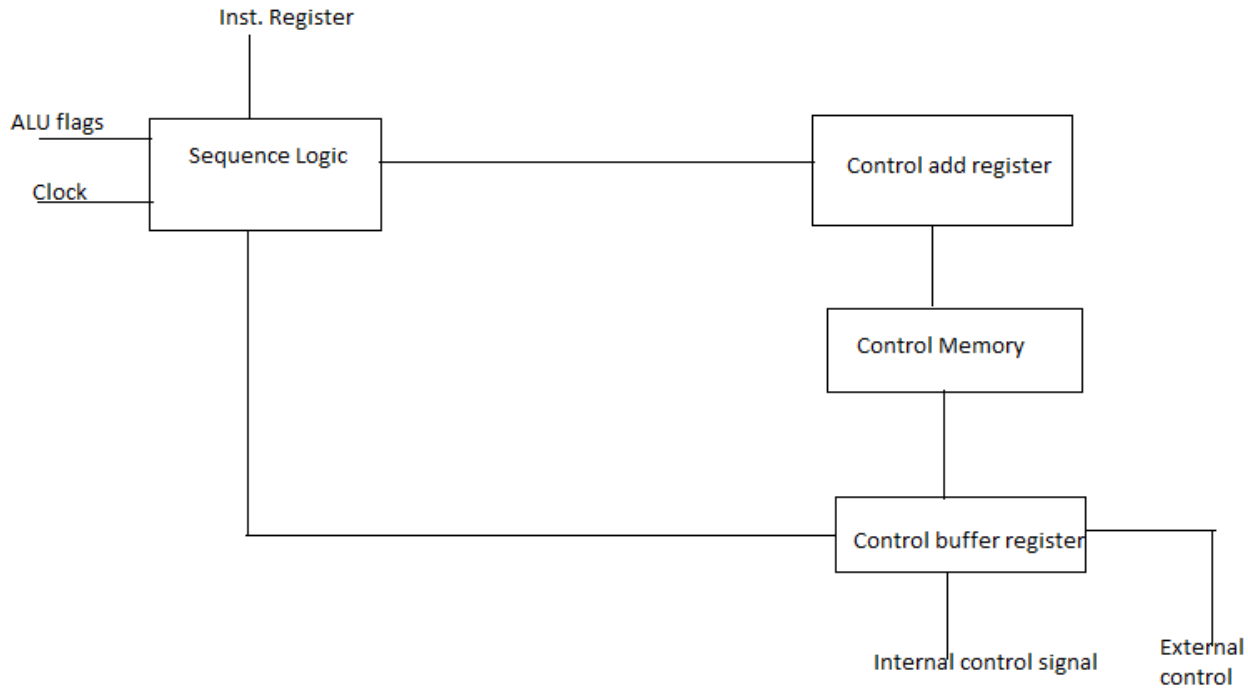
Microprogrammed Control unit :-

- It uses sequence of microinstructions in microprograming language.
- It acts as a mid way between hardware and software.
- It generates a set of control signals.
- It is easy to design, test and implement and it is flexible to modify.
- Microprogrammed control unit consists of set of **control signals**, **control variables**, **control word**, **control memory**, **micro-instructions** and **microprogram**.

Terminologies :-

- **Hardwired Control unit** :- when the control signals are generated by hardware using conventional logic design techniques, control unit is said to be hardwired.
- **Microprogramed control unit** :- A control unit whose binary control variables are stored in a memory is called a microprogrammed control unit.
- **Control memory** :- Control memory is the storage in a microprogrammed control unit to store the microprogram.
- **Control word** :- the control variables at any given time can be represented by a control word string of 1’s and 0’s called a control word.
- **Micro operation** :- it performs basic operations on data stored in one or more registers, including transferring data between register or between registers and external buses of cpu, and performing logical or arithmetic operation on registers.

- **Microcode** :- a very low level instruction set which is stored permanently in a computer or peripheral controller and controls the operation of the device.
- **Microinstruction** :- a single instruction in microcode. It is the most elementary instruction in the computer, such as moving the contents of a register to the arithmetic logic unit.
- **Microprogram** :- a set or sequence of microinstructions.



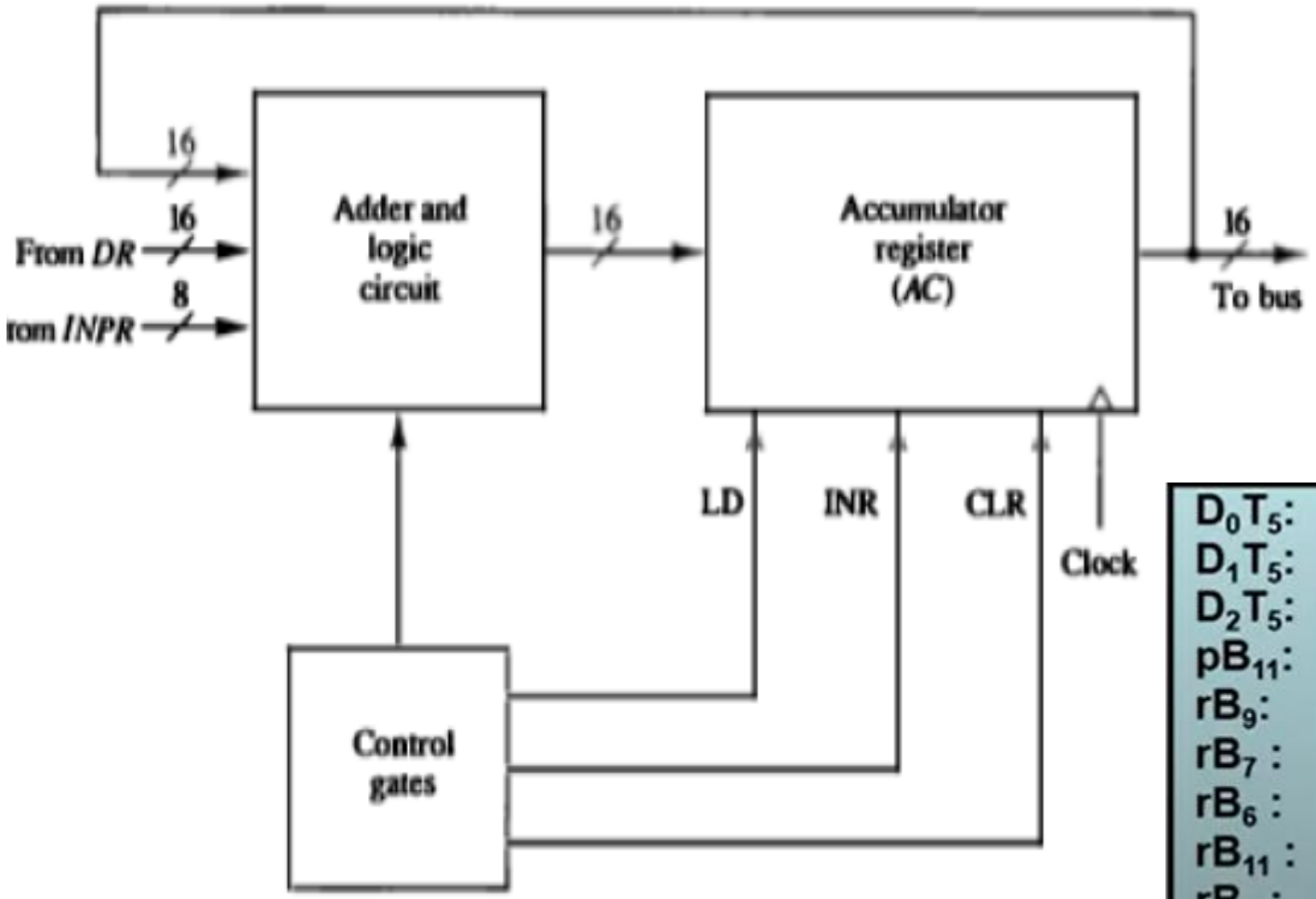
Design of basic computer :-

- The basic computer consists of following hardware components :
  - A memory unit with 4096 words of 16 bits each.
  - Nine registers :- AR, PC, DR, AC, IR, TR, OUTR, INPR, SC
  - Flip-flops : IEN (interrupt enable), I(indirect flip-flop), S(start flip-flop), E(extended flip-flop) ,FGI, and FGO
  - Two decoders : a 3*8 operation decoder and 4*16 timing decoder.
  - A 16-bit common bus.
  - Control logic gates
  - Adder and logic circuit connected to input of AC.

Design of Accumulator Logic

- The adder and logic circuit has three sets of inputs.
- One set of 16 input comes from the outputs of AC.
- Another set of 16 inputs comes from the output of data register (DR).
- A third set of eight inputs comes from the input register INPR.
- The outputs of the adder and logic circuit provide the data inputs for the register.
- In addition, it is necessary to include logic gate for controlling the LD, INR and CLR in the register and for controlling the operation of the adder and logic circuit.

Circuit Associated with AC



D_0T_5:	$AC \leftarrow AC \wedge DR$	AND with DR
D_1T_5:	$AC \leftarrow AC + DR$	Add with DR
D_2T_5:	$AC \leftarrow DR$	Transfer from DR
pB_{11}:	$AC(0\text{-}7) \leftarrow INPR$	Transfer from INPR
rB_9:	$AC \leftarrow AC'$	Complement
rB_7 :	$AC \leftarrow shr\ AC, AC(15) \leftarrow E$	Shift right
rB_6 :	$AC \leftarrow shl\ AC, AC(0) \leftarrow E$	Shift left
rB_{11} :	$AC \leftarrow 0$	Clear
rB_5 :	$AC \leftarrow AC + 1$	Increment

Control of AC register- Gate Structure

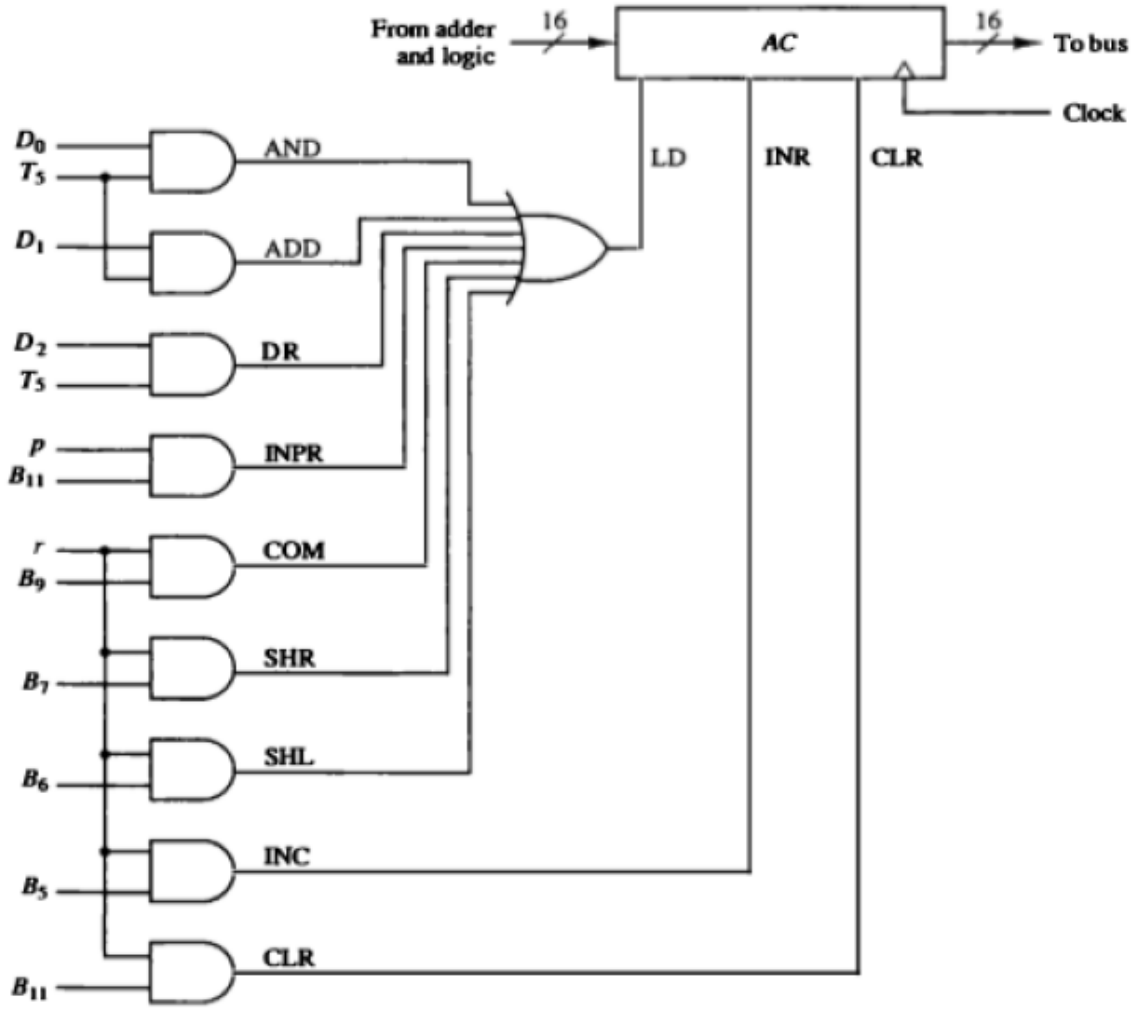


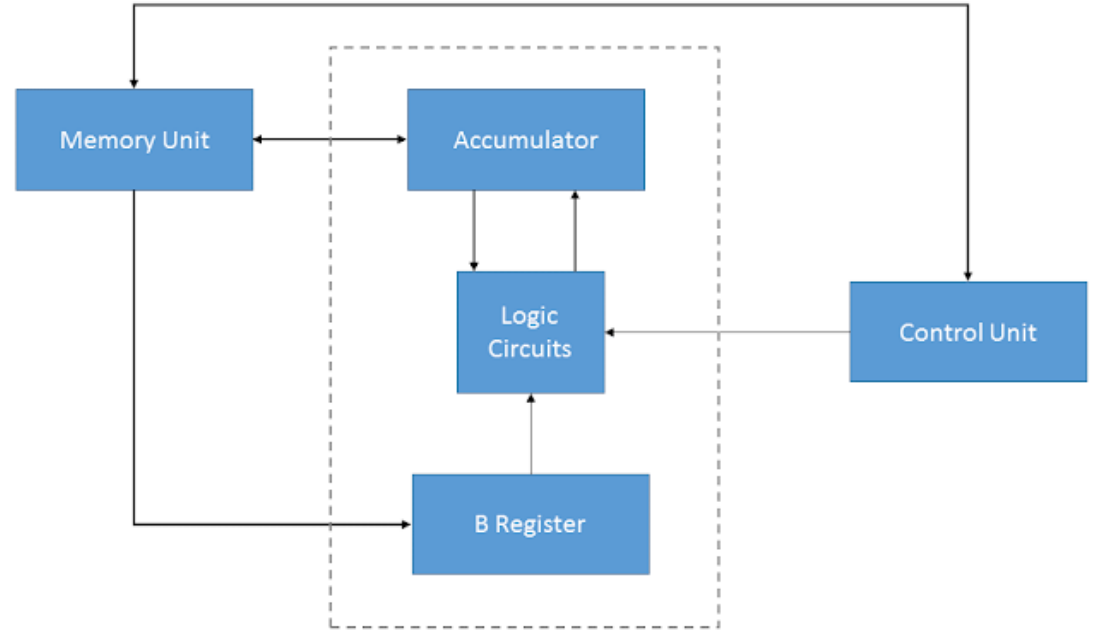
Fig :- Gate structure for controlling the LD, INR, and CLR of AC

Control of AC register-

- The gate structure that controls the LD, INR and CLR inputs of AC is shown in fig.
- The output of the AND gate that generates this control function is connected to the CLR input of the register.
- Similarly, the output of the gate that implements the increment micro operation is connected to the INR input of the register.
- The another seven micro operations are generated in the adder and logic circuit and are loaded into AC at the proper time.
- The outputs of the gates for each control function is marked with symbolic name. These outputs are used in the design of the adder and logic circuit.

ALU organization

- Various circuits are required to process data or perform arithmetical operations which are connected to microprocessor's ALU.
- Accumulator and Data Buffer stores data temporarily. These data are processed as per control instructions to solve problems. Such problems are addition, multiplication etc.



Organization of Arithmetic and Logic Unit

Function of ALU: functions of ALU can be categorized into following 3 categories.

- **Arithmetic operations** :- Additions, multiplications etc. are example of arithmetic operations. Finding greater than or smaller than or equality between two numbers by using subtraction is also a form of arithmetic operations.
- **Logical operations** :- Operations like AND, OR, NOR, NOT etc. using logical circuitry are examples of logical operations.
- **Data manipulations** :- Operations such as flushing a register is an example of data manipulation. Shifting binary numbers are also example of data manipulation.

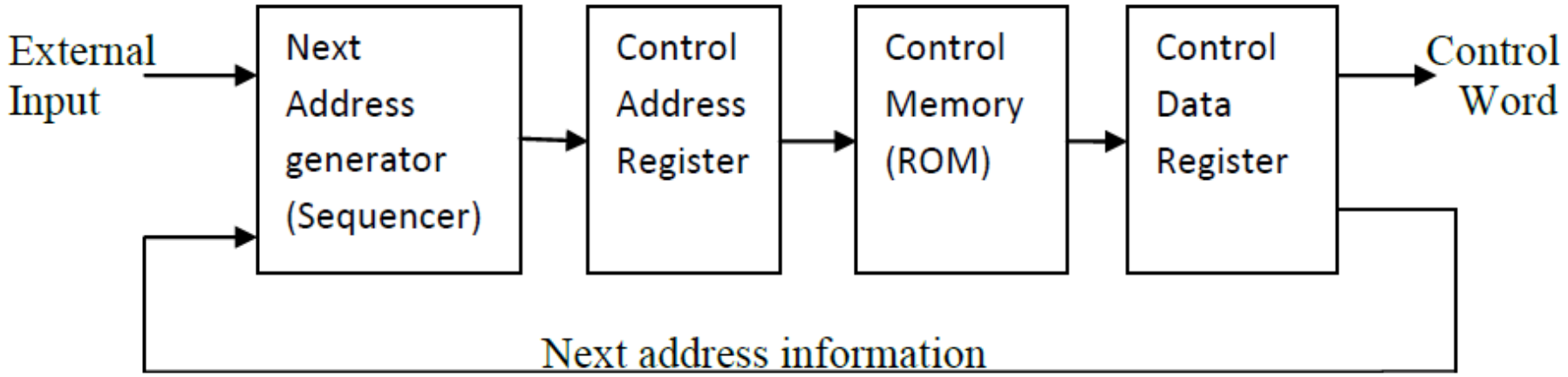
Control Memory

- 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 the programs. The contents of main memory may alter when the data are manipulated and every time that the program is changed. The user's program in main memory consists of machine instructions and data.
- In contrast, 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 microinstructions in control memory. These microinstructions generate the microoperations to fetch the instruction from main memory; to evaluate the effective address, to execute the operation specified by the instruction, and to return control to the fetch phase in order to repeat the cycle for the next instruction.

- The control unit initiates a series of sequential steps of micro operations. During any given time, certain micro operations are to be initiated, while others remain idle. The control variables at any given time can be represented by a string of 1's and 0's called a **control word**.
- As such, control words can be programmed to perform various operations on the components of the system.
- The **microinstruction** specifies one or more **microoperations** for the system. A sequence of microinstructions constitutes a **microprogram**.

Microprogrammed control organization

- The general configuration of microprogrammed control unit is demonstrated in the block diagram.
- The control memory is assumed to be a ROM, within which all control information is permanently stored.
- The control address register specifies the address of the microinstructions.
- The control data register holds the microinstructions read from memory.

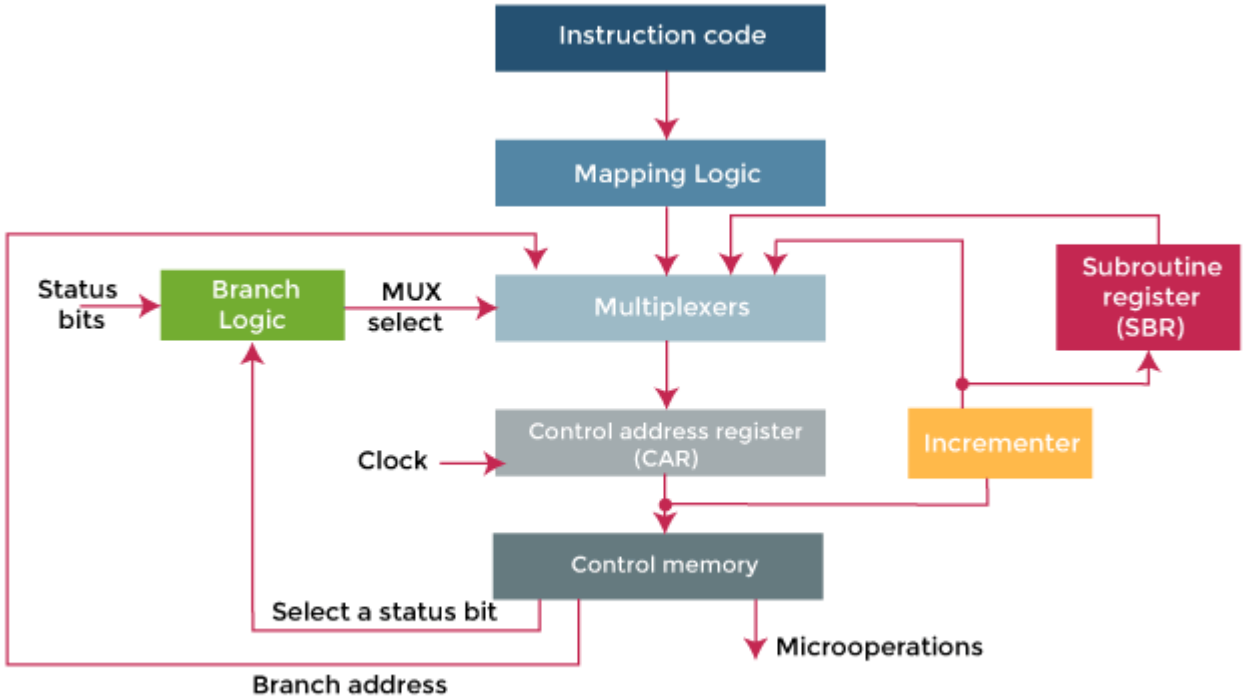


Address sequencing

- Microinstructions are stored in control memory in groups, with each group specifying a **routine**.
- Each computer instruction has its own microprogram routine in control memory to generate the microoperations that execute the instruction.
- To appreciate the address sequencing in a microprogram control unit, let us enumerate the steps that the control must undergo during the execution of a single computer instruction.
- An **initial address** is loaded into the control address register when power is turned on in the computer. This address is usually the address of the first microinstruction that activates the instruction fetch routine. At the end of the fetch routine, the instruction is in the instruction register of the computer.
- The control memory next must go through the routine that determines the effective address of the operand. When the effective address computation routine is completed, the address of the operand is available in the memory address register.

- The next step is to generate the microoperations that execute the instruction fetched from memory. The microoperation steps to be generated in processor registers depend on the operation code part of the instruction.
- When the execution of the instruction is completed, control must return to the fetch routine. This is accomplished by executing an unconditional branch microinstruction to the first address of the fetch routine.
- In summary, the address sequencing capabilities required in a control memory are:
- Incrementing of the control address register.
- Unconditional branch or conditional branch, depending on status bit conditions.
- A mapping process from the bits of the instruction to an address for control memory.
- A facility for subroutine call and return.
- The diagram shows the block diagram of a control memory and its associated hardware to support in choosing the next microinstruction. The microinstruction present in the control memory has a set of bits that facilitate to start off the micro-operations in registers.

- There are four different directions are showed in the figure from where the control address register recovers its address. The CAR is incremented by the incrementer and selects the next instruction. In multiple fields of microinstruction, the branching address can be determined to result in branching.
- It can specify the condition of the status bits of microinstruction, conditional branching can be applied. A mapping logic circuit can share an external address. A special register can save the return address so that when the microprogram needs to return from the subroutine, it can need the value from the unique register.



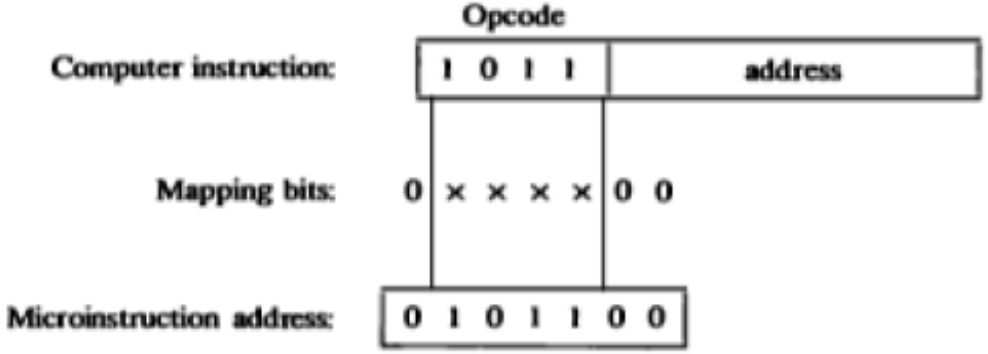
Selection of Address for Control Memory

Conditional Branching :-

- The branch logic provides decision-making capabilities in the control unit.
- The **status conditions** are special bits in the system that provide parameter information such as the carry-out of an adder, the sign bit of a number, the mode bits of an instruction, and input or output status conditions.
- Information in these bits can be tested and actions initiated based on their condition: whether their value is 1 or 0.
- The status bits, together with the field in the microinstruction that specifies a branch address, control the conditional branch decisions generated in the branch logic.
- The branch logic hardware may be implemented in a variety of ways. The simplest way is to test the specified condition and branch to the indicated address if the condition is met; otherwise, the address register is incremented.

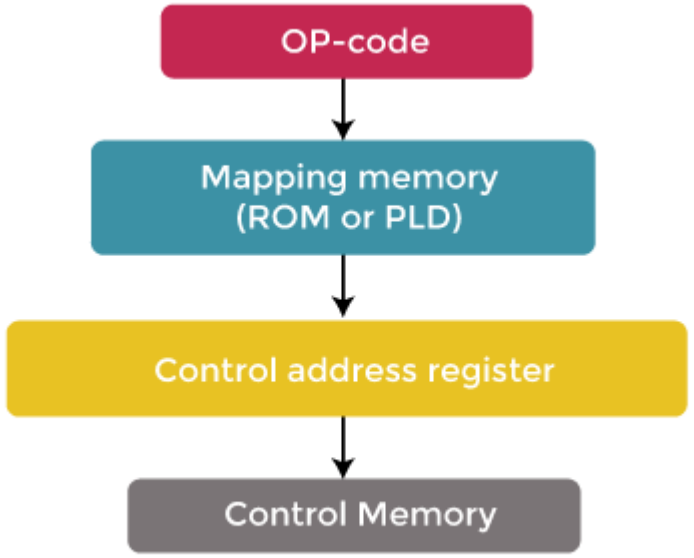
Mapping of Instruction :-

- Each instruction has its own microprogram routine stored in a given location of control memory. The transformation from the instruction code bits to an address in control memory where the routine is located is referred to as a **mapping process**.
- A mapping procedure is a rule that transforms the instruction code into a control memory address.
- For example, a computer with a simple instruction format as shown in Fig has an operation code of four bits. Assume further that the control memory has 128 words, requiring an address of seven bits. For each operation code there exists a microprogram routine in control memory that executes the instruction.

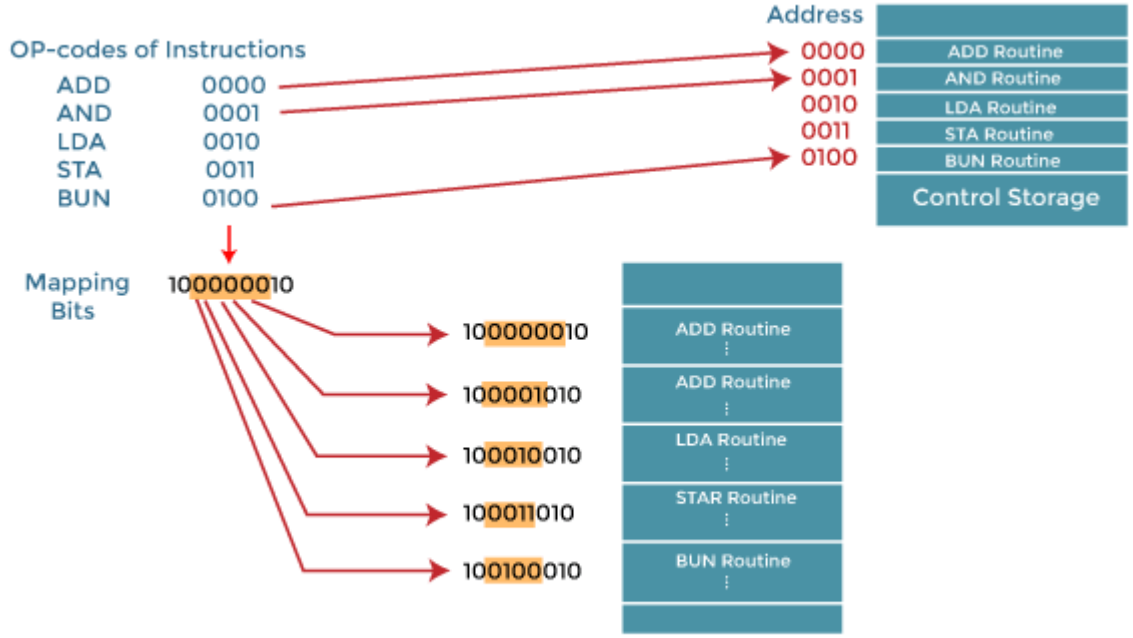


- One simple mapping process that converts the 4-bit operation code to a 7-bit address for control memory.
- This mapping consists of placing a 0 in the most significant bit of the address, transferring the four operation code bits, and clearing the two least significant bits of the control address register. This provides for each computer instruction a microprogram routine with a capacity of four microinstructions.
- If the routine needs more than four microinstructions, it can use addresses 1000000 through 1111111.
- If it uses fewer than four microinstructions, the unused memory locations would be available for other routines.

- The below image shows the mapping of address of microinstruction from the opcode of an instruction. In the execution program, this microinstruction is the starting microinstruction.



Mapping Function Implemented by ROM and PLD



Direct Mapping

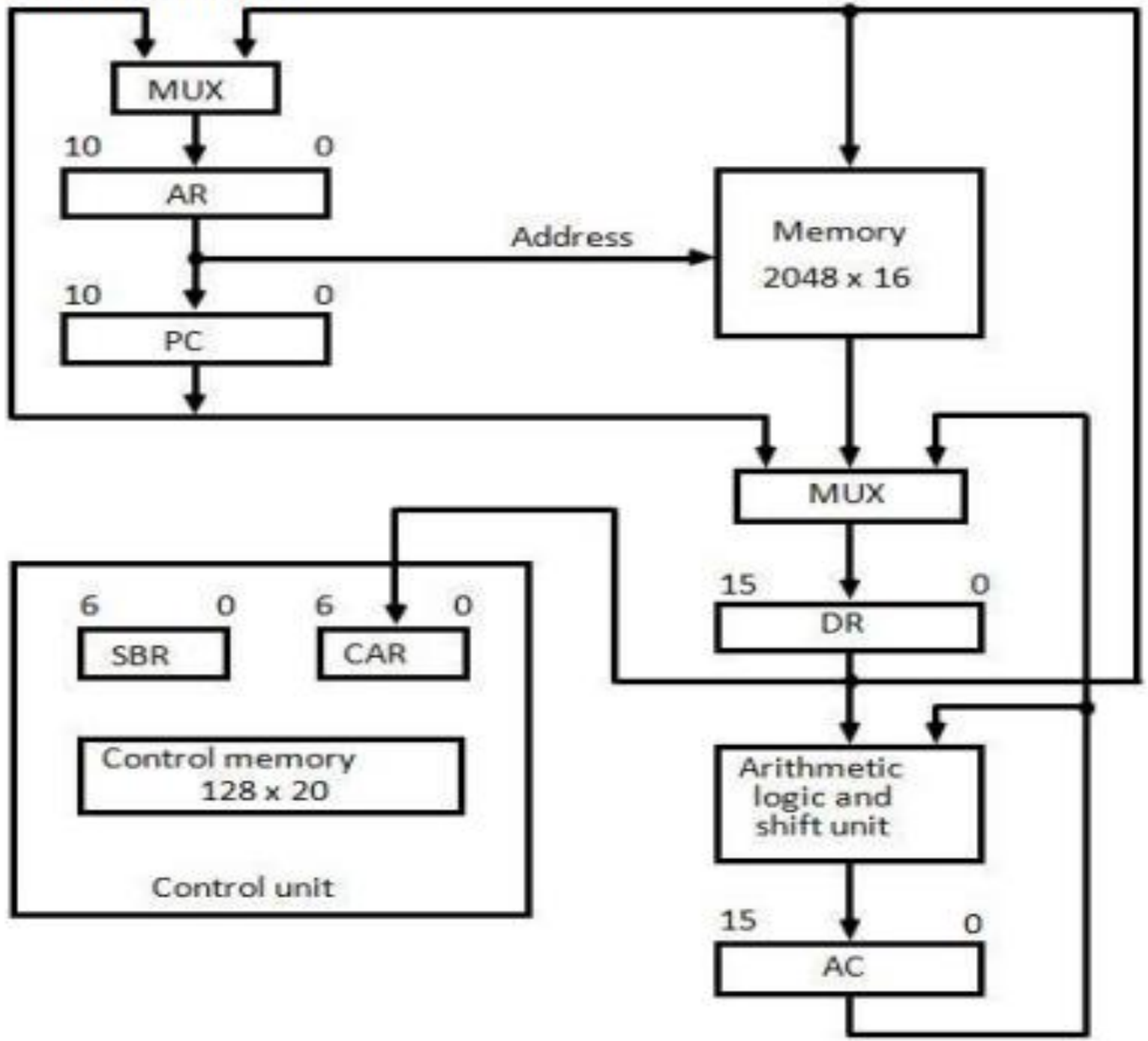
Subroutine :-

- Subroutines are programs that are used by other routines to accomplish a particular task.
- Microinstructions can be saved by employing subroutines that use common section of microcode. e.g. effective address computation.
- The subroutine register can then become the source for transferring the address for the return to the main routine.
- The best way to structure a register file that stores addresses for subroutine is to organize the registers in last-in, first-out (LIFO) stack.

Computer Configuration :-

- The block diagram of the computer is shown below. It has two memory unit.
  - Main memory for storing instructions and data.
  - Control memory for storing the microprogram.
- Four registers are associated with the processor unit :- program counter (PC), address register(AR), data register (DR), accumulator register (AC).
- The control unit has a control address register CAR, and a subroutine register SBR.
- The control memory and its register are organized as microprogrammed control unit, as shown in figure.
- The transfer of information among the registers in the processor is done through multiplexer rather than a common bus.

Computer Hardware Configuration

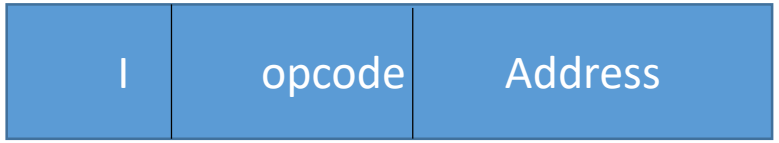


Microprogram:-

- **Microprogram** is a sequence of microinstructions that controls the operation of an arithmetic and logic unit so that machine code instructions are executed.
- It is a microinstruction program that controls the functions of a central processing unit or peripheral controller of a computer.

Computer Instruction Format:-

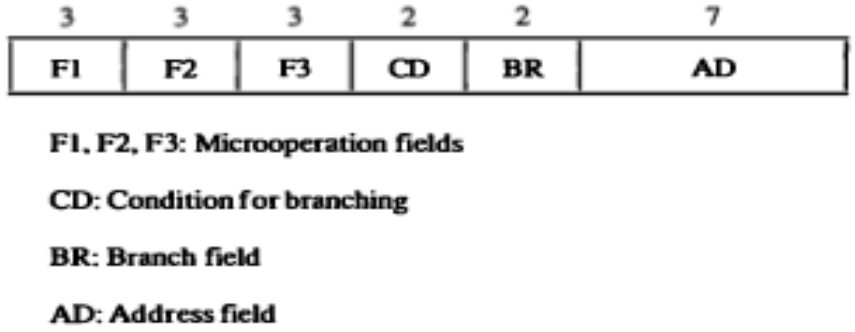
- It consists of three fields :-
  - A 1-bit filed for indirect addressing symbolized by I
  - 4 bit operation code (opcode).
  - An 11-bit address field .



Symbol	OP-code	Description
ADD	0000	AC ← AC + M[EA]
BRANCH	0001	if (AC < 0) then (PC ← EA)
STORE	0010	M[EA] ← AC
EXCHANGE	0011	AC ← M[EA], M[EA] ← AC

Microinstruction Format :-

- The 20 bits of the microinstruction are divided into four functional parts.
  - The three fields F1, F2, and F3 specify microoperations for the computer.
  - The CD field selects status bit conditions.
  - The BR field specifies the type of branch.
  - The AD field contains branch address.



- The three bits in each field are encoded to specify seven distinct micro operations as listed in table.
- No more than three micro operations can be chosen for a microinstructions, one from each fields.
- If fewer than three micro instructions are used, one or more of the fields will use binary code 000 for no operation.

- It is important to realize that two or more conflicting microoperations cannot be specified simultaneously. e.g. 010 001 000
- Each microoperation in the table is defined with register transfer statement and is assigned a symbol for use in a symbolic micro program.

F1	Microoperation	Symbol
000	None	NOP
001	AC←AC+DR	ADD
010	AC←0	CLRAC
011	AC←AC+1	INCAC
100	AC←DR	DRTAC
101	AR←DR(0-10)	DRTAR
110	AR←PC	PCTAR
111	M[AR] ←DR	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	AC←AC-DR	SUB
010	AC←AC v DR	OR
011	AC←AC ^ DR	AND
100	DR←M[AR]	READ
101	DR←AC	ACTDR
110	DR←DR+1	INCDR
111	DR(0-10) ←PC	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	AC←AC+DR	XOR
010	AC←AC’	COM
011	AC←shl AC	SHR
100	AC←shr AC	SHR
101	PC←PC+1	INCPC
110	PC←AR	ARTPC
111	RESERVED	

Condition Field

- A condition field includes 2 bit. They are encoded to define four status bit conditions. As stated in table, the first condition is always 1, with CD=0. the symbol that can indicate this condition is U. the table displays the multiple condition fields and their summary in an easy manner.
- **Condition field symbols and description:-**

	Condition	Symbol	Comments
00	Always =1	U	Unconditional branch
01	DR(15)	I	Indirect address bit
10	AC(15)	S	Sign bit of AC
11	AC=0	Z	Zero value in AC

Branch Field

- The BR (Branch field) includes 2 bits. It can be used by connecting with the AD (Address) field. The reason for connecting with the AD field is to select the address for the next microinstruction. The table illustrates the various branch fields and their functions.
- **Branch field symbols and descriptions :-**

BR	Symbol	Comments
00	JMP	CAR←AD if condition =1 CAR ←CAR+1 if condition =0
01	CALL	CAR←AD, SBR←CAR+1, if condition =1 CAR←CAR+1, If condition =0
10	RET	CAR←SBR (Return from Subroutine)
11	MAP	CAR(2-5) ←DR(11-14), CAR(0,1,6) ←0

Symbolic Microinstruction :-

- Symbols are used in microinstructions as in assembly language.
- A symbolic microprogram can be translated into its binary equivalent by microprogram assembler.

Sample format :-

Five fields :- label; micro-ops; CD; BR; AD

Label : may be empty or may specific a symbolic address terminated with a colon

Micro-ops : consists of one, two, or three symbols separated by commas

CD : one of {U, I, S, Z}, where **U**: unconditional branch

I: indirect address bit
S: sign of AC
Z: zero value in AC

BR: one of { JMP, CALL, RET, MAP }

AD: one of { symbolic address, NEXT, empty}

Symbolic microprogram :- Fetch Routine-

- During FETCH, Read an instruction from memory and decode the instruction and update PC.
- Sequence of microoperations in the fetch cycle :

  AR ← PC

  DR ←M[AR], PC ← PC+1

  AR ← DR(0-10), CAR (2-5) ← DR(11-14), CAR(0,1,6) ← 0

- Symbolic microprogram for the fetch cycle :

```
                ORG 64
FETCH:          PCTAR           U JMP NEXT
                READ, INCPC     U JMP NEXT
                DRTAR           U MAP
```

- Binary equivalents translated by assembler :

Binary address	F1	F2	F3	CD	BR	AD
1000000	110	000	000	00	00	1000001
1000001	000	100	101	00	00	1000010
1000010	101	000	000	00	11	0000000

Symbolic Microprogram :-

- Control storage: 128 20 bit words
- The first 64 words: Routines for the 16 machine instructions
- The last 64 words: used for other purpose (e.g., fetch routine and other subroutines)
- Mapping : OP-code XXXX into 0XXXX00 the first address for the 16 routines are 0(0 0000 00), 4(0 0001 00), 8, 12, 16, 20,……., 60

Partial symbolic microprogram :-

Label	Micro ops	CD	BR	AD
ADD:	ORG 0			
	NOP	I	CALL	INDIRECT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
BRANCH	ORG 4			
	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER:	NOP	I	CALL	INDIRECT
	ARTPC	U	JMP	FETCH
STORE:	ORG 8			
	NOP	I	CALL	INDIRECT
	ACTDR	U	JMP	NEXT
	WRITE	U	JMP	FETCH
EXCHANGE:	ORG 12			
	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ACTDR, DRTAC	U	JMP	NEXT
	WRITE	U	JMP	FETCH
FETCH:	ORG 64			
	PCTAR	U	JMP	NEXT
	READ, INCPC	U	JMP	NEXT
	DRTAR	U	MAP	
INDRCT:	READ	U	JMP	NEXT
	DRTAR	U	RET	

Micro Routine	Address		Binary Microinstruction					
	Decimal	Binary	F1	F2	F3	CD	BR	AD
ADD	0	0000000	000	000	000	01	01	1000011
	1	0000001	000	100	000	00	00	0000010
	2	0000010	001	000	000	00	00	1000000
	3	0000011	000	000	000	00	00	1000000
BRANCH	4	0000100	000	000	000	10	00	0000110
	5	0000101	000	000	000	00	00	1000000
	6	0000110	000	000	000	01	01	1000011
	7	0000111	000	000	110	00	00	1000000
STORE	8	0001000	000	000	000	01	01	1000011
	9	0001001	000	101	000	00	00	0001010
	10	0001010	111	000	000	00	00	1000000
	11	0001011	000	000	000	00	00	1000000
EXCHANGE	12	0001100	000	000	000	01	01	1000011
	13	0001101	001	000	000	00	00	0001110
	14	0001110	100	101	000	00	00	0001111
	15	0001111	111	000	000	00	00	1000000
FETCH	64	1000000	110	000	000	00	00	1000001
	65	1000001	000	100	101	00	00	1000010
	66	1000010	101	000	000	00	11	0000000
INDRCT	67	1000011	000	100	000	00	00	1000100
	68	1000100	101	000	000	00	10	0000000

Symbolic vs Binary Microprogram:-

Symbolic Microprogram	Binary Microprogram
It is a set of microinstructions written in a symbolic form.	It is a set of microinstructions written in a binary form.
It is a convenient form for writing microprograms in a way that people can read and understand.	It is written in binary form so can be difficult for people to read and understand.
To be stored in memory symbolic microprogram must be translated to binary either by means of an assembler program or by the user if the microprogram is simple enough.	Translation is not required for storing in memory because it is already written in binary form.
The symbolic representation is useful for writing microprograms in an assembly language format.	The binary representation is the actual internal content that must be stored in control memory.
Example: Symbolic microprogram for fetch routine:	Example: Translation of symbolic microprogram to binary microprogram.

```
FETCH:  ORG 64
        PCTAR          U    JMP    NEXT
        READ, INCPC    U    JMP    NEXT
        DRTAR          U    MAP
```

Binary Address	F1	F2	F3	CD	BR	AD
1000000	110	000	000	00	00	1000001
1000001	000	100	101	00	00	1000010
1000010	101	000	000	00	11	0000000

8086 microprocessor

Prepared by: Laxmi KC ,Lecturer
Advanced college of Engineering and Management, Department of Electronics and Computer

Features

- It is a 16-bit μp.
- 8086 has a 20 bit address bus can access up to 2^{20} memory locations (1 MB)
- It can support up to 64K I/O ports
- It provides 14, 16 -bit registers
- Word size is 16 bits and double word size is 4 bytes
- It has multiplexed address and data bus AD0- AD15 and A16 – A19
- 8086 is designed to operate in two modes : Minimum and Maximum
- It can pre-fetches up to 6 instruction bytes from memory and queues them in order to speed up instruction execution.
- It requires +5V power supply
- 40 pin dual in line package
- Address ranges from 00000H to FFFFFH

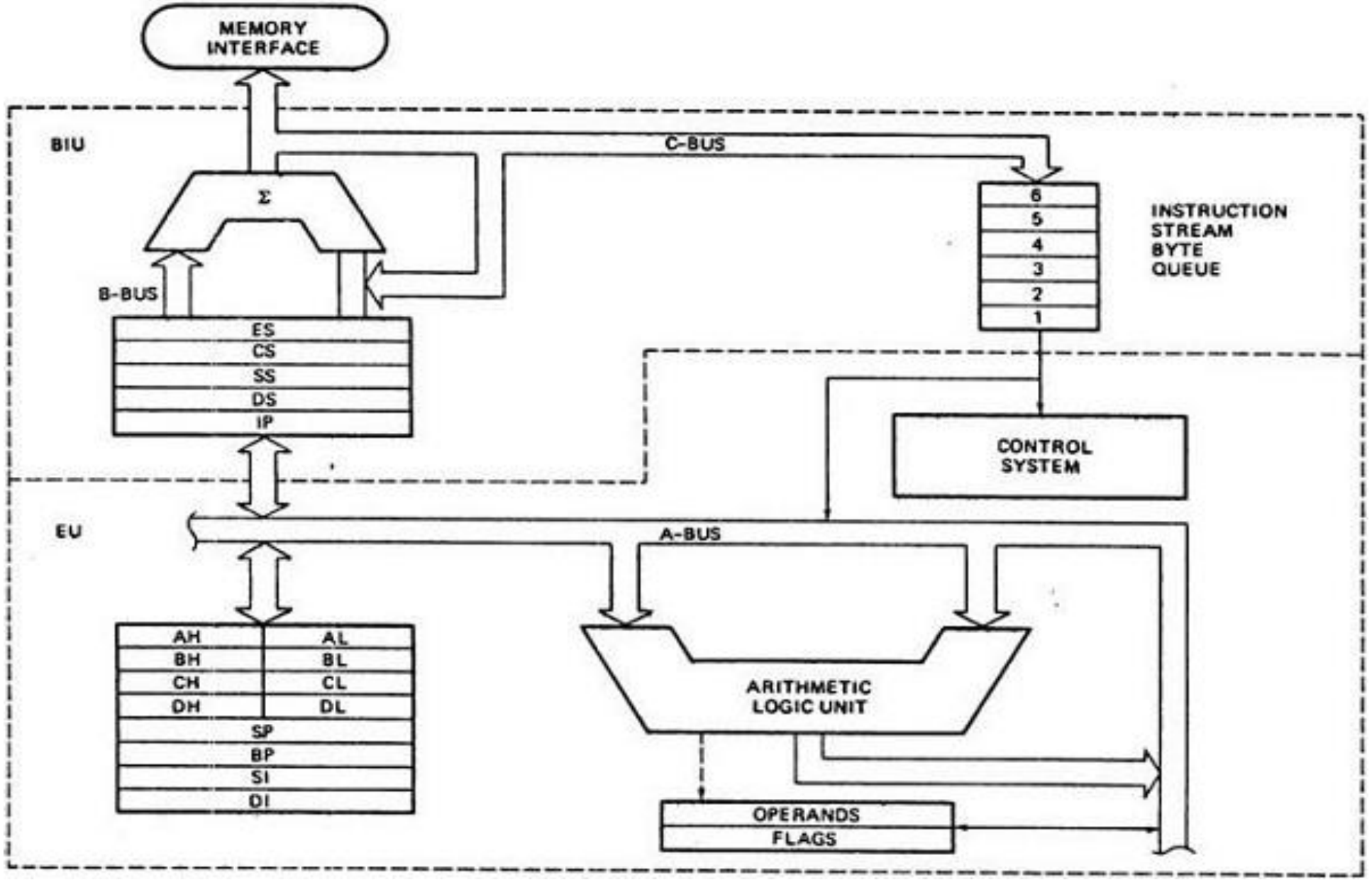


Figure: Internal architecture of 8086 microprocessor

Internal architecture of 8086

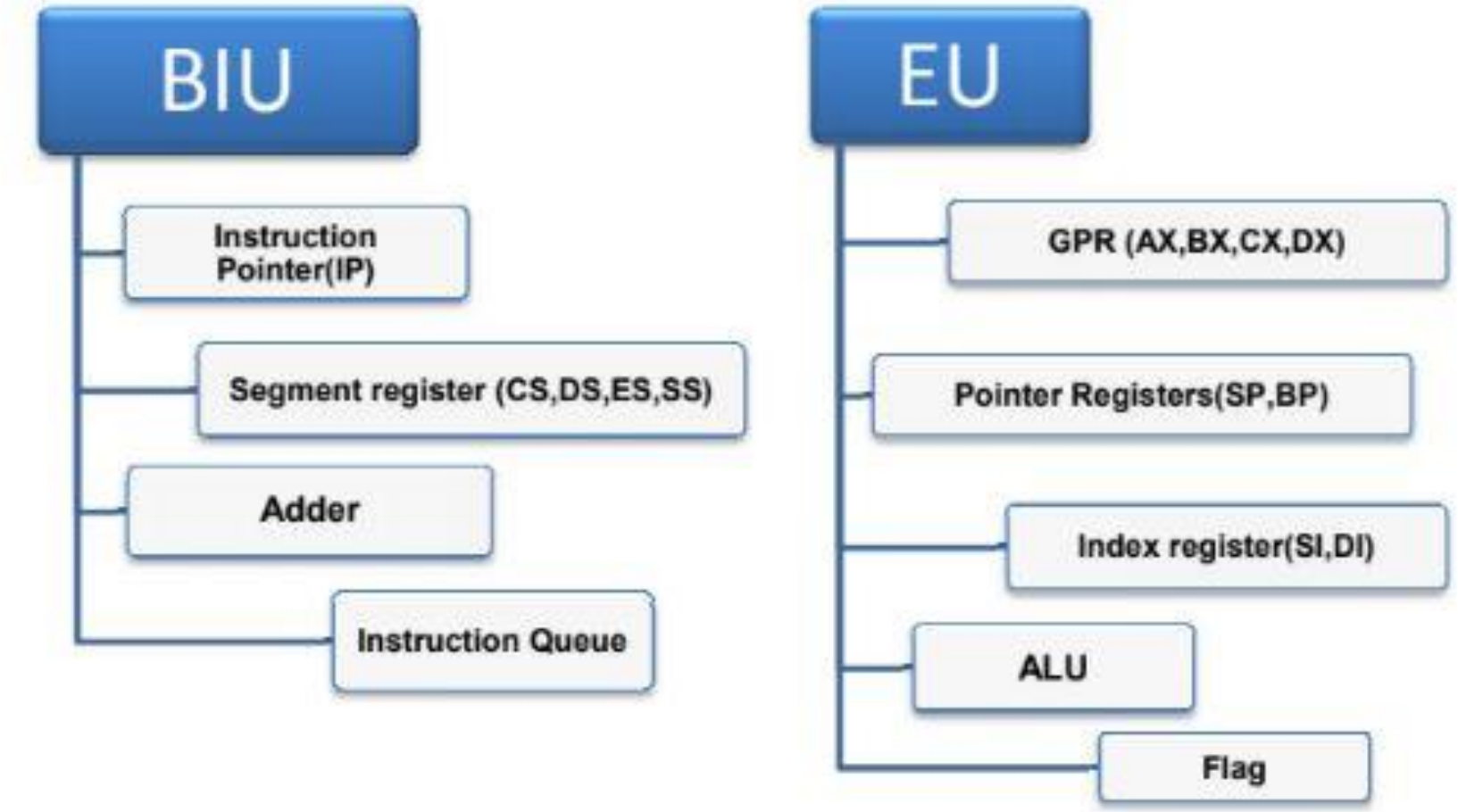
The 8086 has two parts, the Bus Interface Unit (BIU) and the Execution Unit (EU).

- The BIU fetches instructions, reads and writes data, and computes the 20-bit address.
- The EU decodes and executes the instructions using the 16-bit ALU.
- The two units functions independently.

Minimum and Maximum Modes:

- The minimum mode is selected by applying logic 1 to the MN / $\overline{MX}$ input pin. This is a single microprocessor configuration.
- The maximum mode is selected by applying logic 0 to the MN / $\overline{MX}$ input pin. This is a multi microprocessors configuration.

Components of BIU and EU



Internal architecture of 8086

Bus Interface Unit:

- The BIU performs all bus operations such as instruction fetching, reading and writing operands for memory and calculating the addresses of the memory operands.
- The instruction bytes are transferred to the instruction queue.
- It provides a full 16 bit bidirectional data bus and 20 bit address bus.
- The bus interface unit is responsible for performing all external bus operations.
- Functional units of BIU are:
  - Instruction queue
  - Segment registers
  - Instruction pointer
  - Address Summing block (Σ)

Internal architecture of 8086

Instruction queue:

- The BIU uses a mechanism known as an instruction stream queue to implement a pipeline architecture.
- This queue permits pre-fetch of up to 6 bytes of instruction code. Whenever the queue of the BIU is not full, it has room for at least two more bytes and at the same time the EU is not requesting it to read or write operands from memory, the BIU is free to look ahead in the program by pre-fetching the next sequential instruction.
- These prefetching instructions are held in its FIFO queue. With its 16 bit data bus, the BIU fetches two instruction bytes in a single memory cycle.
- The EU accesses the queue from the output end. It reads one instruction byte after the other from the output of the queue. If the queue is full and the EU is not requesting access to operand in memory. These intervals of no bus activity, which may occur between bus cycles, are known as Idle state.

Internal architecture of 8086

Segment registers:

- Segment register is 16-bit in size which contains starting address of segment
- BIU contains 4 16-bit segment registers which contains base addresses for program instructions, data and stack

➢ Code segment (CS) is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register.

➢ Stack segment (SS) is a 16-bit register containing address of 64KB segment with program stack. By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction.

➢ Data and Extra segment (DS and ES) is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, and DX) and index register (SI, DI) is located in the data and Extra segment.

Internal architecture of 8086

Address Summing block (Σ):

- The BIU also contains a dedicated adder which is used to generate the 20bit physical address that is output on the address bus. This address is formed by adding an appended 16 bit segment address and a 16 bit offset address.

Instruction pointer(IP):

- Holds the offset(effective address) of the next instruction to be executed
- The physical address of the next instruction to be fetched is formed by combining the current contents of the code segment CS register and the current contents of the instruction pointer IP register

Internal architecture of 8086

Execution Unit:

- The Execution unit is responsible for decoding and executing all instructions.
- The EU extracts instructions from the top of the queue in the BIU, decodes them, generates operands if necessary, passes them to the BIU and requests it to perform the read or write bus cycles to memory or I/O and perform the operation specified by the instruction on the operands.
- During the execution of the instruction, the EU tests the status and control flags and updates them based on the results of executing the instruction.
- When the EU executes a branch or jump instruction, it transfers control to a location corresponding to another set of sequential instructions. Whenever this happens, the BIU automatically resets the queue and then begins to fetch instructions from this new location to refill the queue

Internal architecture of 8086

- Functional units of EU are:
  - Control system(control circuitry and instruction decoder)
  - General purpose registers
  - Pointer registers
  - Index registers
  - ALU
  - Flag registers

Internal architecture of 8086

Control system:

- Receives an instruction from queue ,decode it and generates different control signals for execution

General purpose registers:

- AX (Accumulator):
  - It is consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX.
  - AL in this case contains the low-order byte of the word, and AH contains the high-order byte.
  - Accumulator can be used for I/O operations and string manipulation.
- BX (Base register):
  - It is consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX.
  - BL in this case contains the low-order byte of the word, and BH contains the high-order byte.
  - BX register usually contains a offset for data segment.

Internal architecture of 8086

- CX (Count register):
  - It is consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX. When combined, CL register contains the low-order byte of the word, and CH contains the high-order byte.
  - Count register can be used in Loop, shift/rotate instructions and as a counter in string manipulation.
  - 8086 has the LOOP instruction which is used for counter purpose when it is executed CX/CL is automatically decremented by 1.
  - Example

```
        MOV CL, 05H
START:  NOP
        LOOP START (here CL is automatically decremented by 1without DCR instruction)
```

Internal architecture of 8086

- DX (Data register):
  - It is consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX.
  - When combined, DL register contains the low-order byte of the word, and DH contains the high-order byte.
  - DX can be used as a port number in I/O operations.
  - In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.

Pointer registers:

- Stack Pointer (SP):
  - is a 16-bit register is used to hold the offset address for stack segment.
- Base Pointer (BP):
  - is a 16-bit register is used to hold the offset address for stack segment.
  - BP register is usually used for based, based indexed or register indirect addressing.

The difference between SP and BP is that the SP is used internally to store the address in case of interrupt and the CALL instruction

Internal architecture of 8086

Index registers:

These two 16-bit register is used to hold the offset address for DS and ES in case of string manipulation instruction

- Source Index (SI)
  - SI is used for indexed, based indexed and register indirect addressing, as well as a source data addresses in string manipulation instructions
- Destination Index (DI)
  - DI is used for indexed, based indexed and register indirect addressing, as well as a destination data addresses in string manipulation instructions.

ALU:

- Performs arithmetic and logical operations
- 8-bit or 16-bit mathematical operations such as addition ,subtraction, multiplication, division, data conversion, logical operation like AND, OR, NOT
- Also performs increment, decrement and shift operations

Internal architecture of 8086

Flag register:

A flag is a flip flop which indicates some conditions produced by the execution of an instruction or controls certain operations of the EU .

- In 8086 the EU contains,
  - 16 bit flag register
  - 9 of the 16 are active flags and remaining 7 are undefined
  - 6 flags indicates some conditions- status flags
  - 3 flags –control Flags

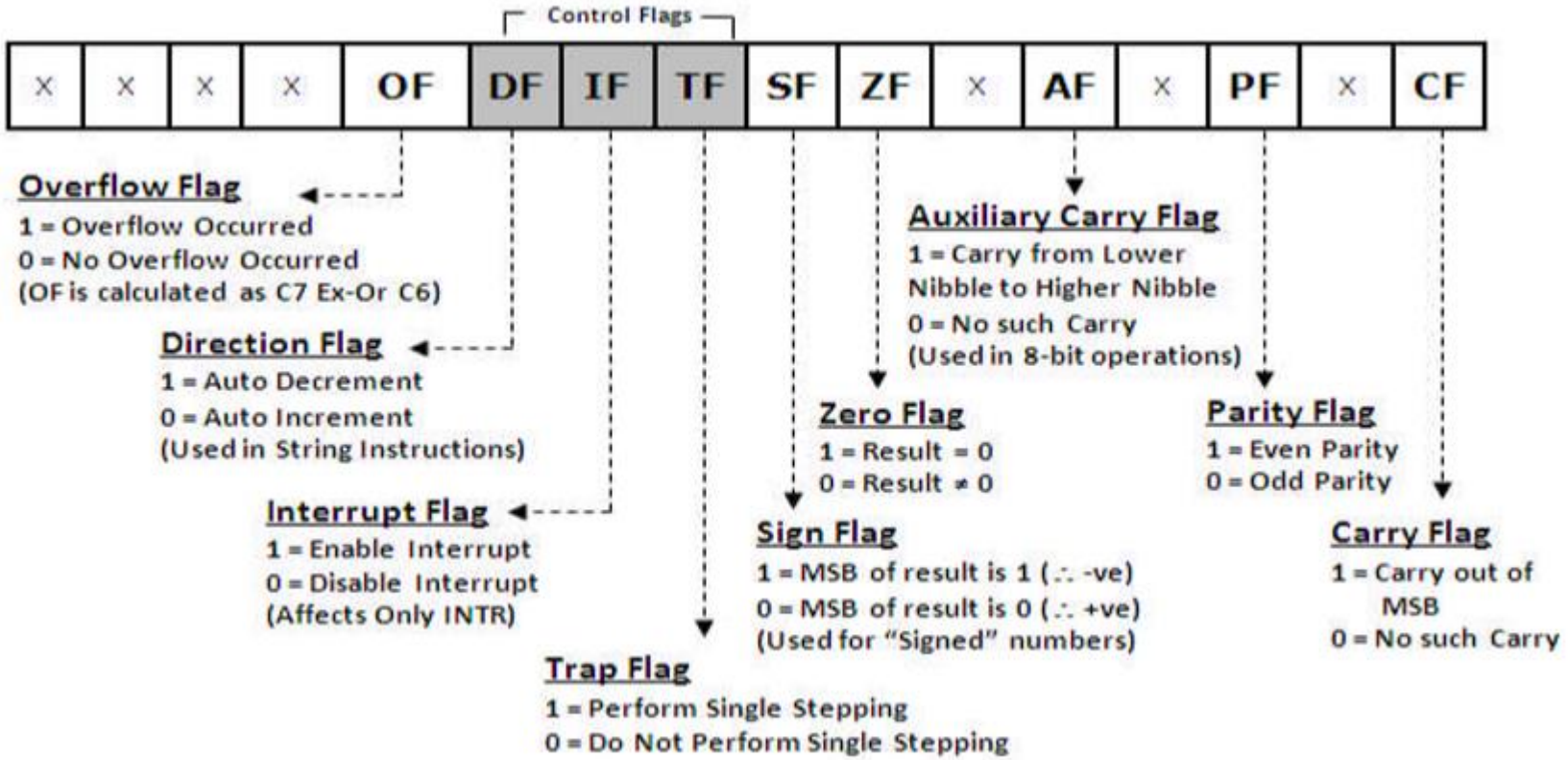


Figure : Flag register of 8086

Internal architecture of 8086

A flag is a 16-bit register containing 9 one bit flags

- Sign, zero, auxiliary carry , parity and carry flags are same as that of 8085

i. Overflow Flag (OF)

- This flag is set if an overflow occurs. i.e. if the result of a signed operation is large enough to be accommodated in a destination register.

ii. Direction Flag (DF)

- This is used by string manipulation instructions. If this flag bit is '0', the string is processed beginning from the lowest address to the highest address. i.e. auto-incrementing mode. Otherwise, the string is processed from the highest address towards the lowest address, i.e. auto-decrementing mode.

iii. Interrupt-enable Flag (IF)

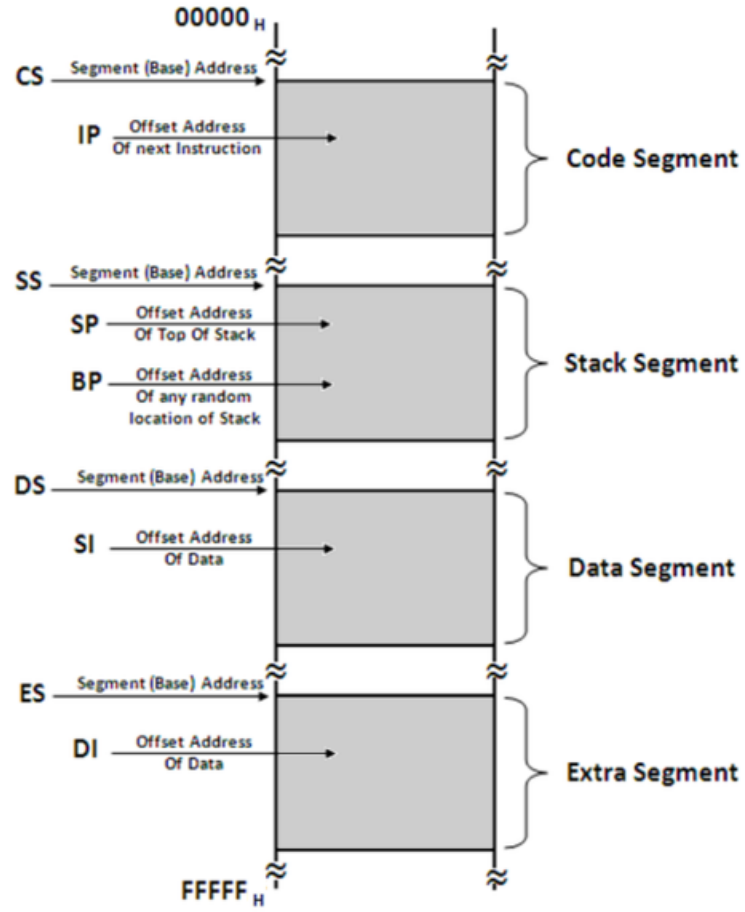
- If this flag is set, the maskable interrupts are recognized by the CPU. Otherwise they are ignored. Setting this bit enables maskable interrupts.

Internal architecture of 8086

iv. Trap flag/Single-step Flag(TF)

- If this flag is set, the processor enters the single step execution mode. In other words, a trap interrupt is generated after execution of each instruction. The processor executes the current instruction and the control is transferred to the Trap interrupt service routine.

MEMORY SEGMENTATION IN 8086



Segmented Memory

Reason for Segmented Memory:

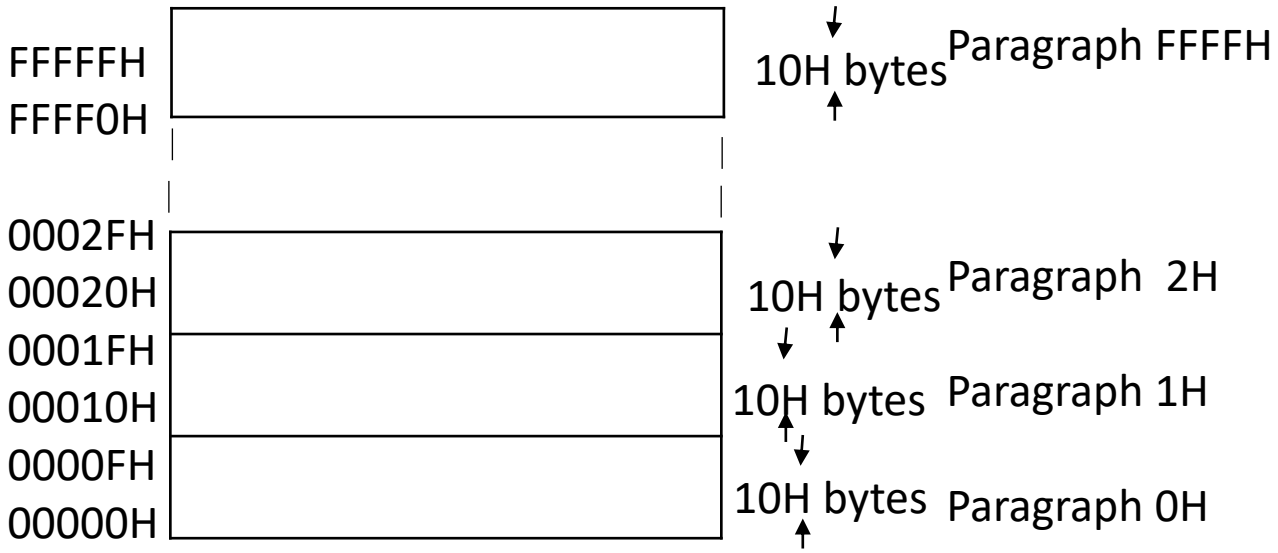
- 8086 has a 20-bit address bus. So it can address a maximum of 1MB of memory and each memory location is addressed by a 20 bit address.
- To hold a 20-bit address there must be a 20-bit address register available within processor but 8086 only has 16-bit registers. So 20-bit address can't be stored inside the 16-bit register.
- To avoid this problem segmented memory is used in 8086.
- 8086 can work with only four 64KB segments at a time within this 1MB range.
- Each location in a particular segment can be expressed by two addresses.
  - Segment Address (16 bit): It refers the starting address of a segment and it is fixed for whole of the segment.
  - Offset or Displacement Address (16 bit): It refers the individual location in that segment and it is varied location wise.
- By using these two addresses the 20 bit physical address can be calculated as below:

  Physical address (20 bit) = [Segment Address (16 bit) * 10]H + Offset Address(16 bit)
- According to this formula segment address is multiplied by 10 and is added to offset.
- This is equivalent to shifting of segment register content towards left 4 times so that four zero are added to right side (MSB) of the segment address and added with the offset address to get the physical address which is 20 bit.

Segmented Memory

Paragraphs:

- The Memory Address Space (MAS) is divided into 65,536 (i.e., 10,000H) paragraphs
- Each paragraph is 16 (i.e., 10H) consecutive bytes
- Thus each paragraph starts at a physical address whose rightmost hexadecimal digit is zero
- A physical memory segment is a block of 2^{16} (i.e., 64K or 10,000H) consecutive bytes **starting at a paragraph boundary**.



Segmented Memory

If Segment address= 1000H and Offset address =01FBH then,

Physical address = [Segment Address * 10]H + Offset Address

= 1000 *$(10)_{16}$ +01FBH

= 101FBH

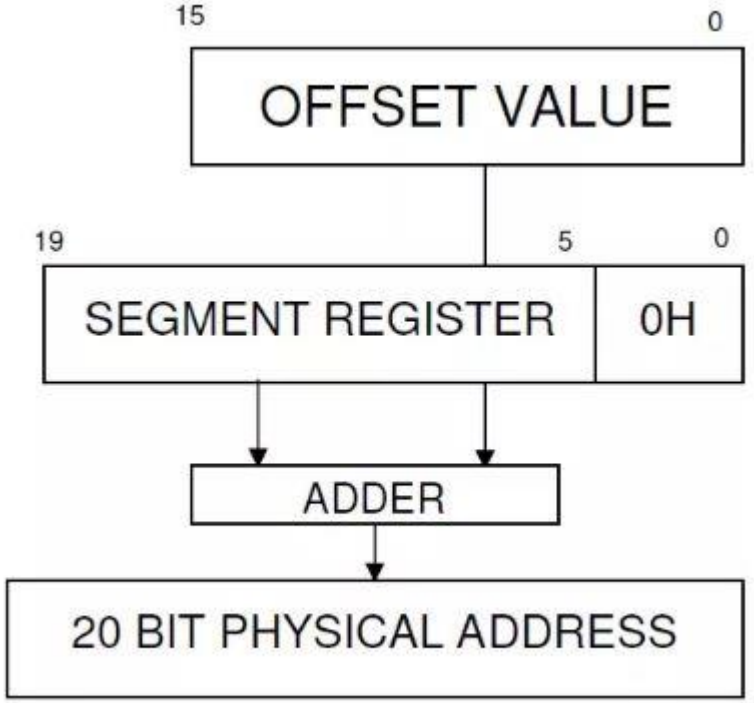


Figure: Physical address calculation

Assembling, Linking and Executing

1) Assembling:

- Assembling converts source program into object program if syntactically correct and generates an intermediate .obj file or module.
- Basically, the assembler goes through the program one line at a time and generates machine code for that instruction
- Assembler complains about the syntax error if any and does not generate the object module.
- Assembler creates .obj , .lst and .crf files and last two are optional files that can be created at run time.

Assembling, Linking and Executing

Types of assembler

One pass assembler:

- This assembler scans the assembly language program once and converts to object code at the same time.
- Works fine with backward referencing
- Problem with forward referencing

- Backward referencing	Forward Referencing
L1:	JNZ L2
……..	…….
……..	…….
JNZ L1	L2: ……

Two pass assembler :

- This type of assembler scans the assembly language twice.
- First pass generates symbol table of names and labels used in the program and calculates their relative address.
- This table can be seen at the end of the list file and here user need not define anything.
- Second pass uses the table constructed in first pass and completes the object code creation.
- This assembler is more efficient and easier than earlier.

Assembling, Linking and Executing

Linking:

- This involves the converting of .obj module into .exe(executable) module i.e. executable machine code
- It completes the address left by the assembler
- It combines separately assembled object files
- Linking creates .EXE, .LIB, .MAP files among which last two are optional files.

Loading and Executing:

- It Loads the program in memory for execution
- It resolves remaining address.
- This process creates the program segment prefix (PSP) before loading
- It executes to generate the result

Sample program: assembling → object Program linking → executable program

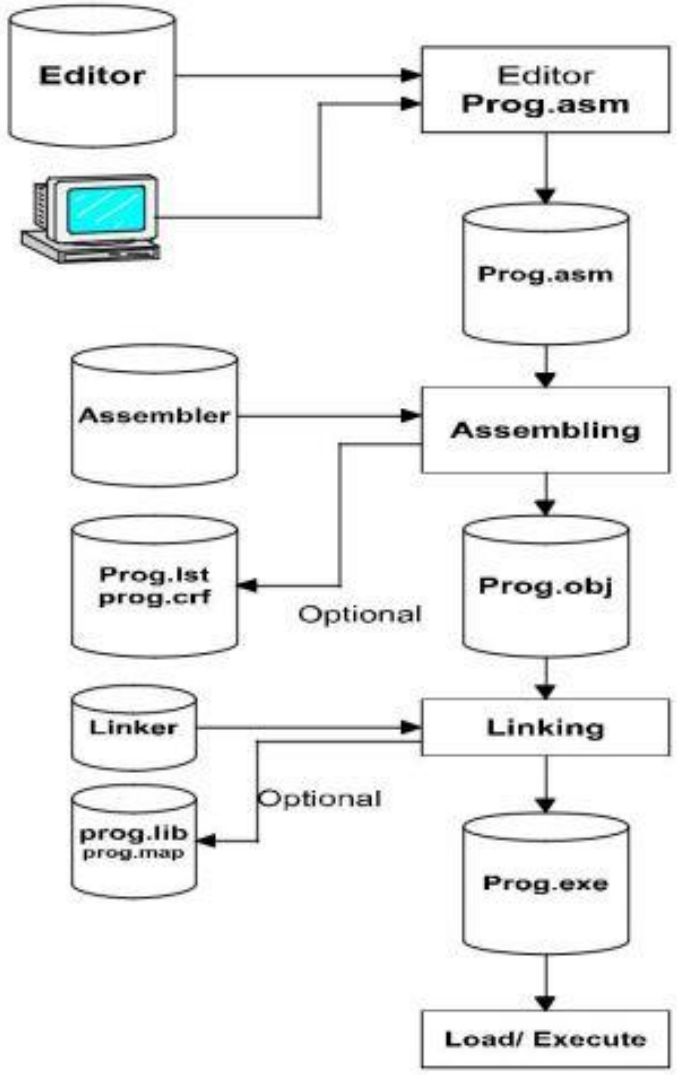
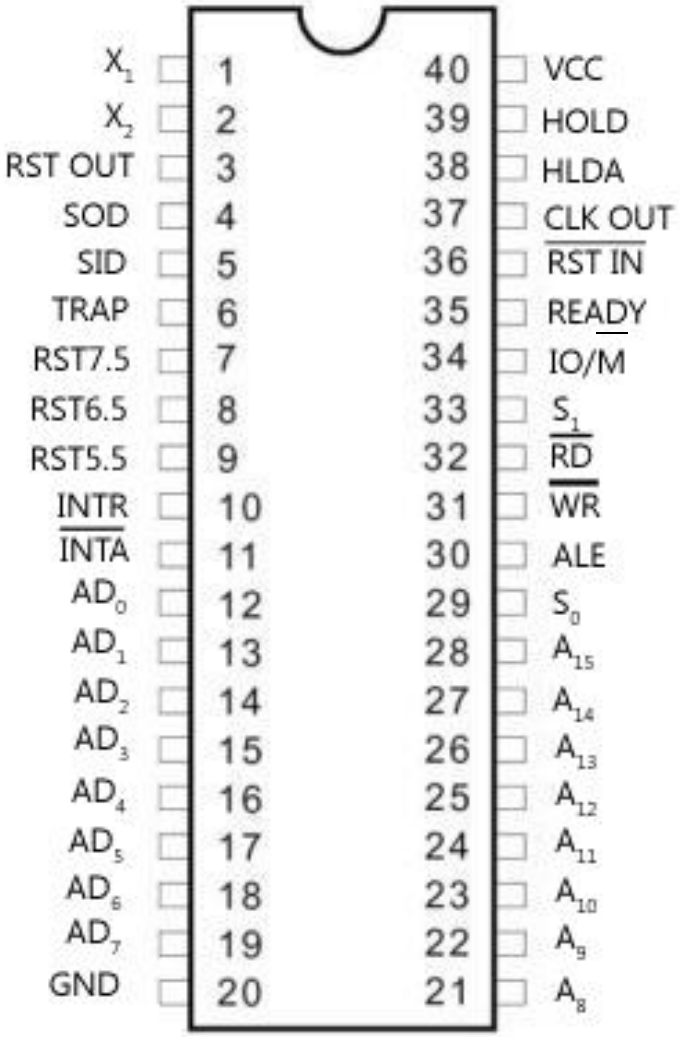


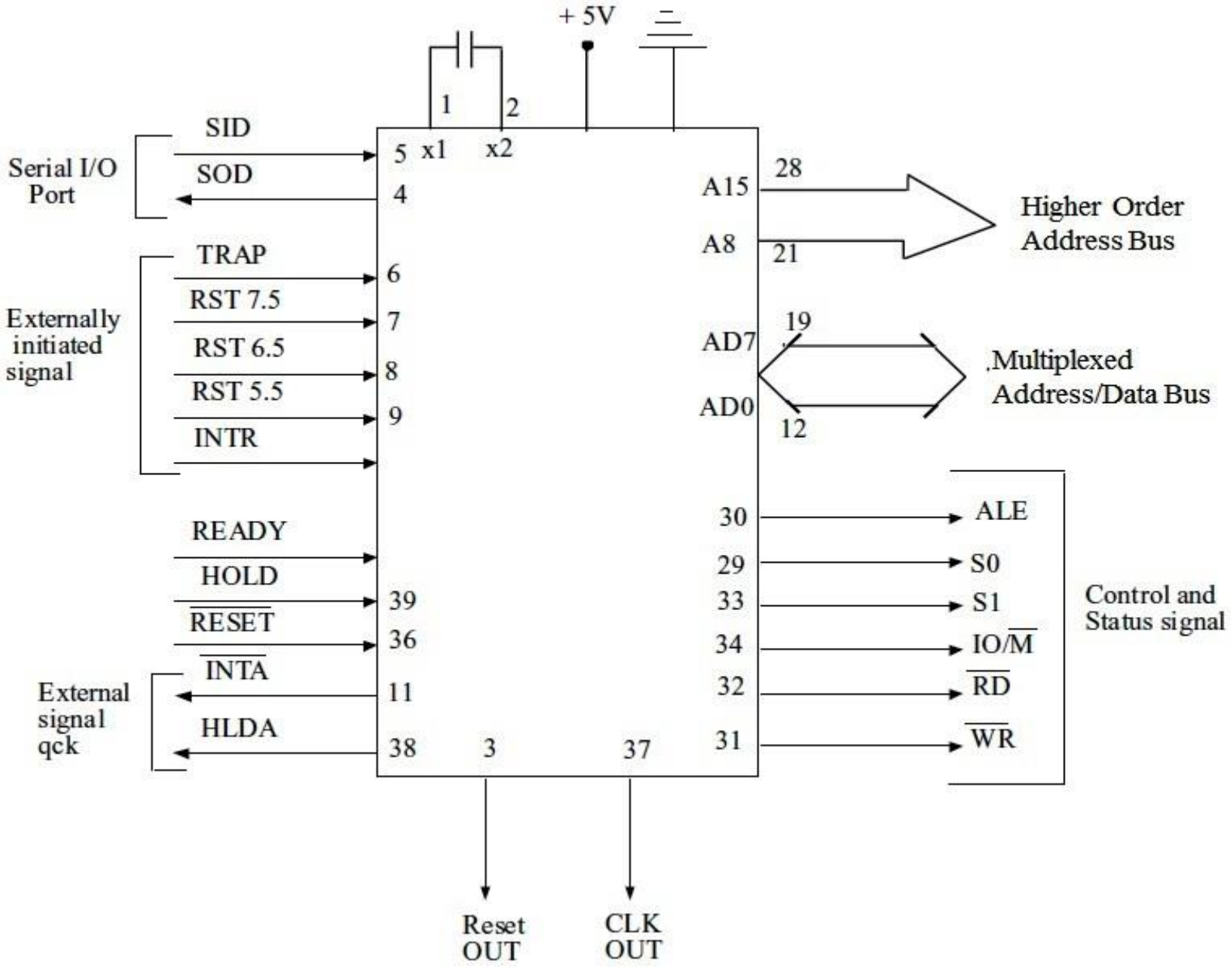
Figure: Steps in assembling , linking and executing

Microprocessor system

8085 Microprocessor Pin Diagram



8085 Microprocessor Pin Diagram



8085 Microprocessor Pin Diagram

All the signals can be classified into six groups:

1. Address bus
2. Data bus
3. Control and status signals
4. Power supply and frequency signals
5. Externally initiated signals and
6. Serial I/O ports

8085 Microprocessor Pin Diagram

- Among the 40 pins of 8085 μp, they are categorized into following types:
  - Unidirectional higher order address pins. ($A_8 – A_{15}$) – 8
  - Bidirectional multiplexed Address/Date pins ($AD_0 – AD_7$) – 8
  - Control input pins (11):
    - 8 externally initiated signals
    - 2 clock inputs XI & X2
    - 1 SID
  - Control output pins (11):
    - 6 control and status pins
    - 2 external acknowledgement signals
    - 1 CLK OUT +
    - 1 RESET OUT
    - 1 SOD
  - power supply & ground pins. – 2

8085 Microprocessor Pin Diagram

ADDRESS BUS

- The 8085 has 16 signal lines (pins), split into two segments: $A_{15}$to A_8 and AD_7 to AD_0
- $A_{15}$to A_8 are unidirectional used for the most significant bits, called the high-order address 16-bit address
- AD_7 to AD_0 are multiplexed address/data bus, bidirectional ,used as the low order address bus as well as the data bus

CONTROL AND STATUS SIGNALS

ALE(Address Latch Enable):

- This is a positive going pulse generated every time the 8085 begins an operation
- it indicates that the bits on AD_7 -AD_0 are address bits. This signal is used primarily to latch the low-order address from the multiplexed bus and generate a separate set of eight address lines, A7–A0

8085 Microprocessor Pin Diagram

$\overline{RD}$-Read:

- Read control signal (active low).
- This signal indicates that the selected I/O or memory device is to be read and data are available on the data bus.

$\overline{WR}$-Write:

- Write control signal (active low)
- This signal indicates that the data on the data bus are to be written into a selected memory or I/O location

S1and S0:

- These status signals, similar to IO/$\overline{M}$, can identify various operations

8085 Microprocessor Pin Diagram

IO/$\overline{M}$

- This is a status signal used to differentiate between I/O and memory operations
- When it is high, it indicates an I/O operation
- When it is low, it indicates a memory operation
- This signal is combined with RD (Read) and WR (Write) to generate I/O and memory control signals.

8085 Microprocessor Pin Diagram

Status			Machine cycle	Control signals
IO/$\overline{M}$	S1	S0		
0	1	1	Opcode fetch	$\overline{RD}$=0
0	1	0	Memory read	$\overline{RD}$=0
0	0	1	Memory write	$\overline{WR}$=0
1	1	0	I/O Read	$\overline{RD}$=0
1	0	1	I/O Write	$\overline{WR}$=0
1	1	1	Interrupt acknowledge	$\overline{INTR}$=0

Figure: 8085 Cycle status

8085 Microprocessor Pin Diagram

POWER SUPPLY AND CLOCK FREQUENCY

The power supply and frequency signals are as follows:

- Vcc: +5 V power supply
- Vss: Ground Reference
- X_1, X_2: A crystal is connected at these two pins. The frequency is internally divided by two; therefore, to operate a system at 3 MHz, the crystal should have a frequency of 6 MHz
- CLK (OUT)-Clock Output: This signal can be used as the system clock for other devices.

SERIAL I/O PORTS

In serial transmission , data bits are sent over a single line , one bit at a time

- SID: Serial input data
- SOD: Serial output data

8085 Microprocessor Pin Diagram

EXTERNALLY INITIATED SIGNALS, INCLUDING INTERRUPTS

INTR (Input) Interrupt request :

- Used as general-purpose interrupt

INTA (Output) Interrupt acknowledge

- used to acknowledge an interrupt

RST 7.5, RST 6.5, RST 5.5:

- Restart Interrupts
- These are vectored interrupts that transfer the program control to specific memory locations. They have higher priorities than the INTR interrupt. Among these three, the priority order is 7.5, 6.5. and 5.5.

TRAP (Input)

- a non-maskable interrupt and has the highest priority

8085 Microprocessor Pin Diagram

HOLD (Input)

- indicates that a peripheral such as a DMA (Direct Memory Access) controller is requesting the use of the address and data buses,

HLDA (Output)

- signal acknowledges the HOLD request.

READY (Input)

- is used to delay the microprocessor Read or Write cycles until a slow-responding peripheral is ready to send or accept data. When this signal goes low, the microprocessor waits for an integral number of clock cycles until it goes high.

$\overline{\text{RESET IN}}$

- Program counter is set to zero and MPU is reset

RESET OUT

- Used to reset other devices

Machine Cycles & Bus Timing Diagrams

- The total operations of a microprocessor can be classified into following four groups
  - Op-code Fetch
  - Memory Read & Write
  - I/O Read & Write
  - Request Acknowledge

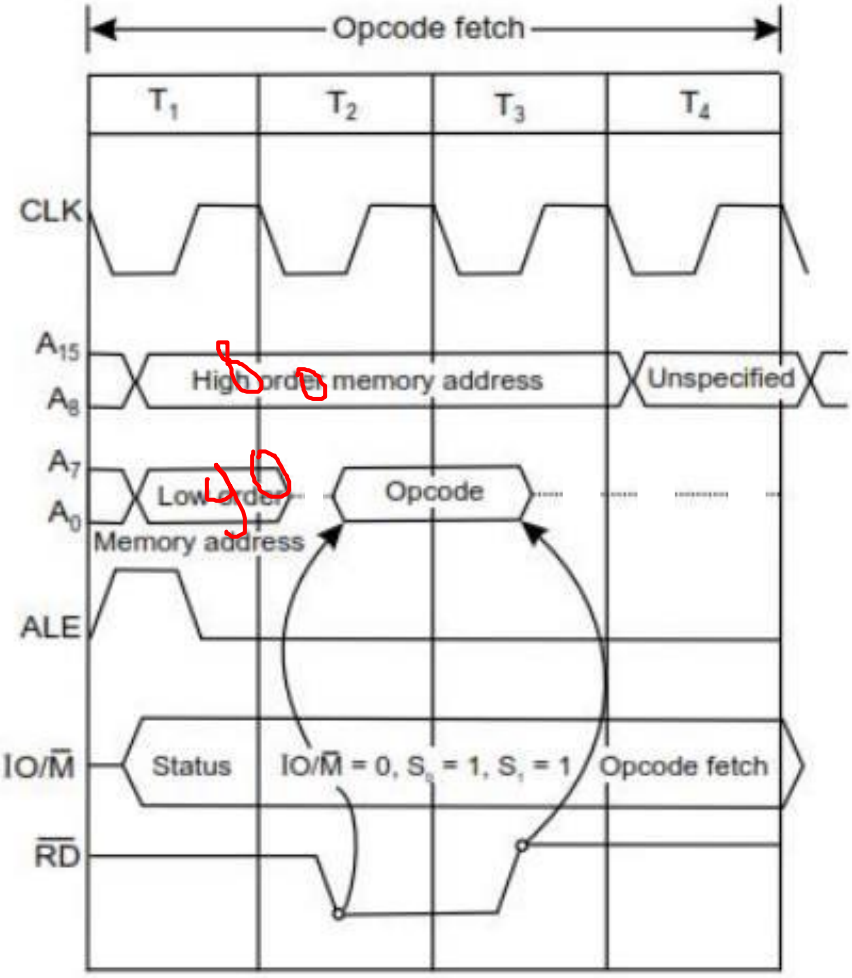
Op-Code fetch is an internal operation and other three are external operations

Instruction cycle: It is defined as the time required to complete the execution of an instruction . Instruction cycle consist of 1-6 machine cycles or operations

Machine Cycle: It is defined as the time required to complete one operation of accessing memory, I/O, or acknowledging an external request. This cycle may consist of three to six T-states.

Clock Cycle (T state): It is defined as one subdivision of the operation performed in one clock period. Each T-state is precisely equal to one clock period

Machine Cycles & Bus Timing Diagrams



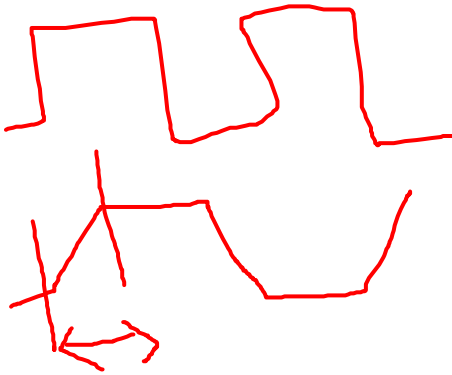


Fig 1.8 Opcode fetch machine cycle

Example:

Draw the timing diagram of the instruction MOV B,D.

- Let us consider the instruction MOV B,D be Stored at memory location C000H
- Op-Code for the instruction is 42H and Op-Code fetch cycle is of 4 clock cycles

Step1:

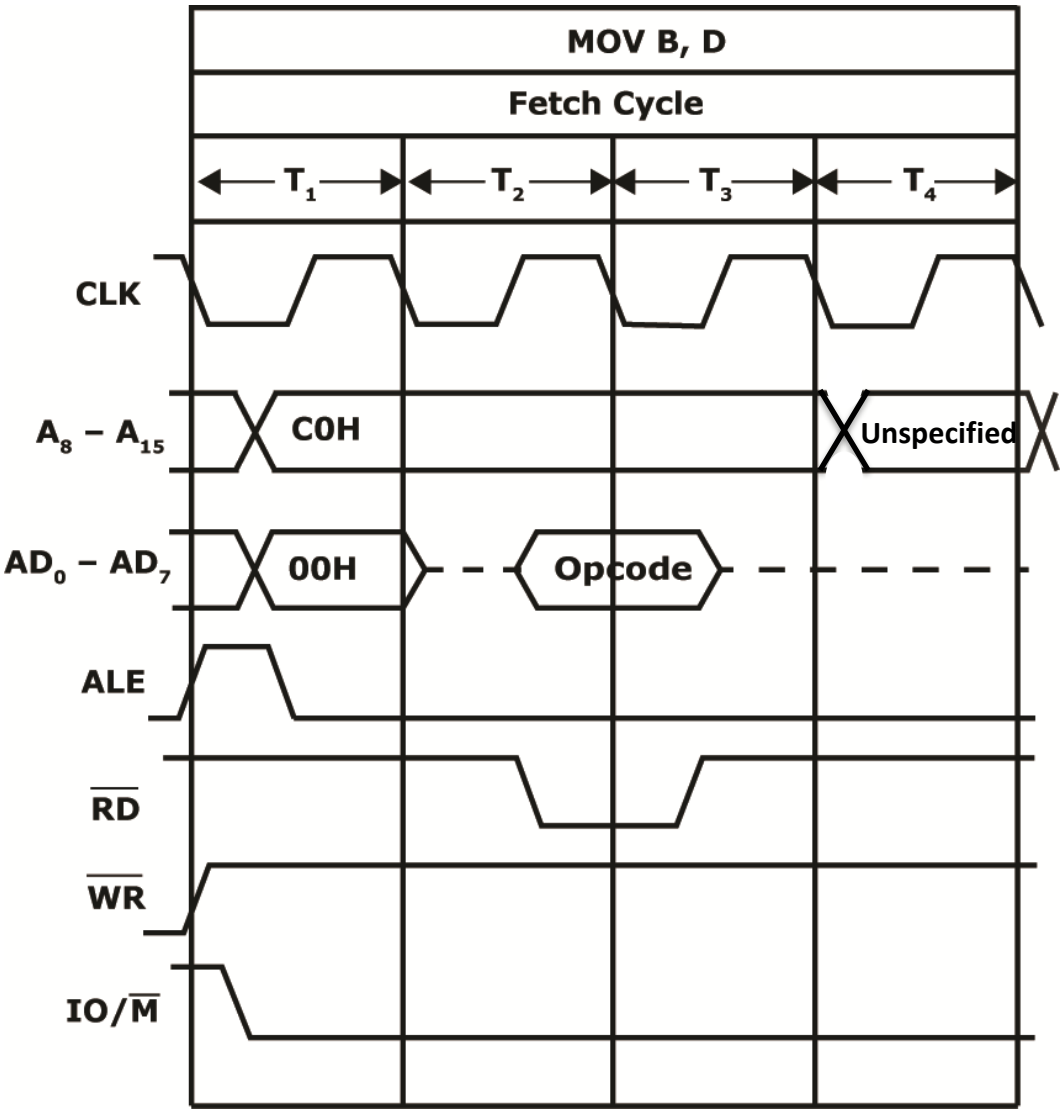
- Microprocessor places the 16 bit memory address from Program Counter on the address bus. At T1, high order address (C0) is placed at A8-A15 and lower order address (00) is placed at AD0- AD7
- ALE signal goes high.
- IO/M goes low and both S0 and Sl goes high for Op-Code fetch.

Step 2: The control unit sends the control signal RD to enable the memory chip and active during T2and T3

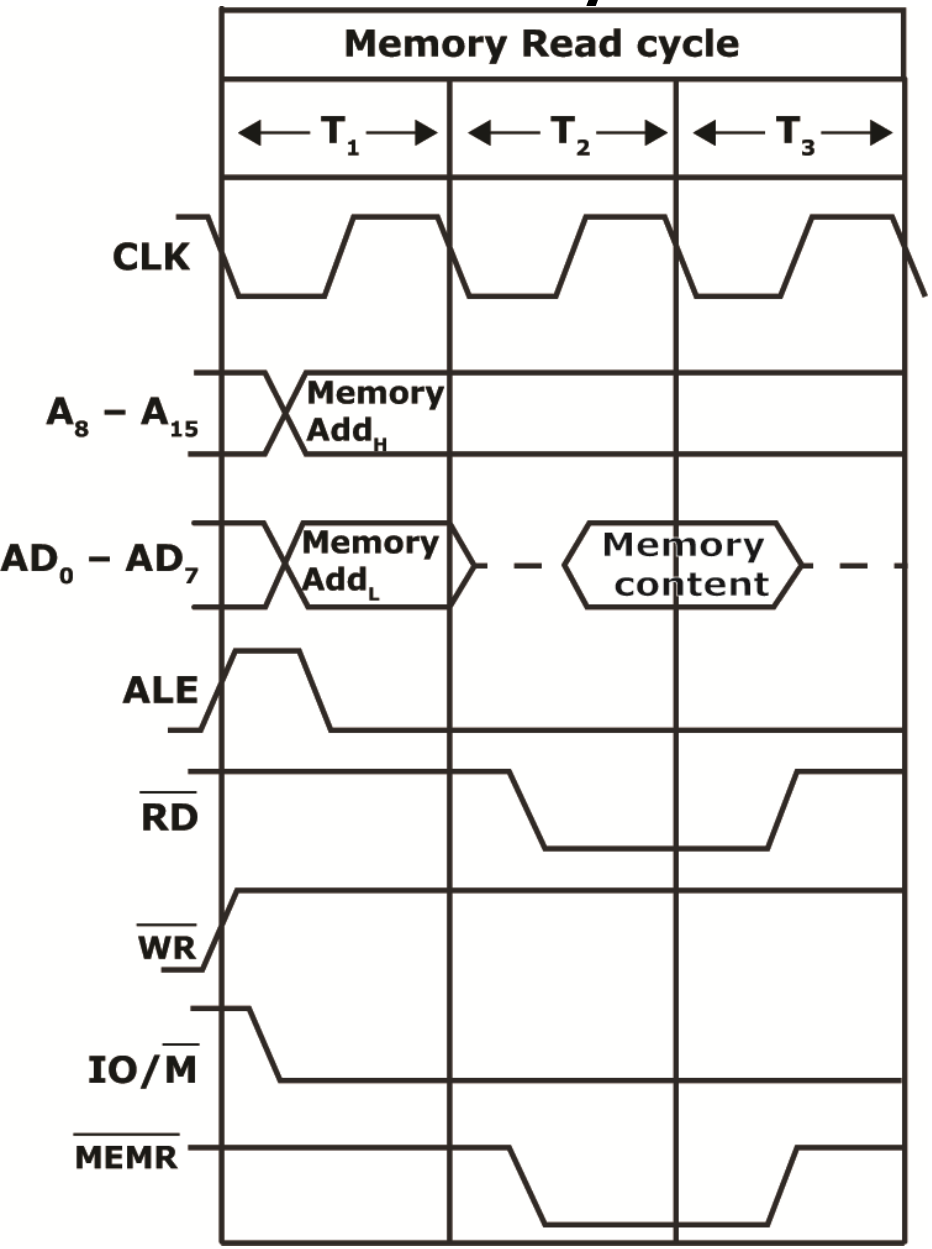
Step 3: The byte from the memory location is placed on the data bus .that is 42H into D0-D7 and RD goes high impedance

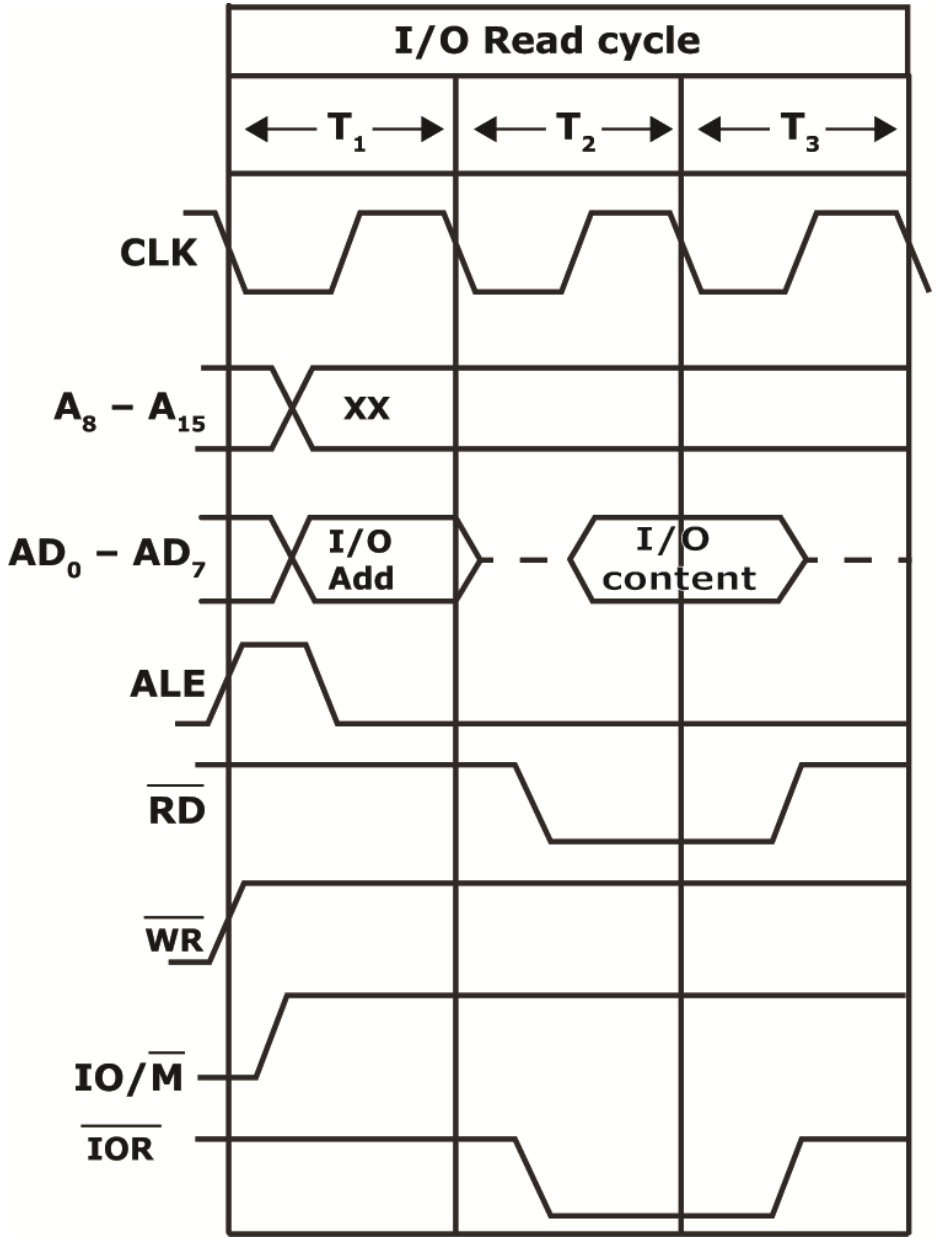
Step4: The instruction 4FH is decoded and content of register D will be copied into register B during clock cycle T4

Machine Cycles & Bus Timing Diagrams

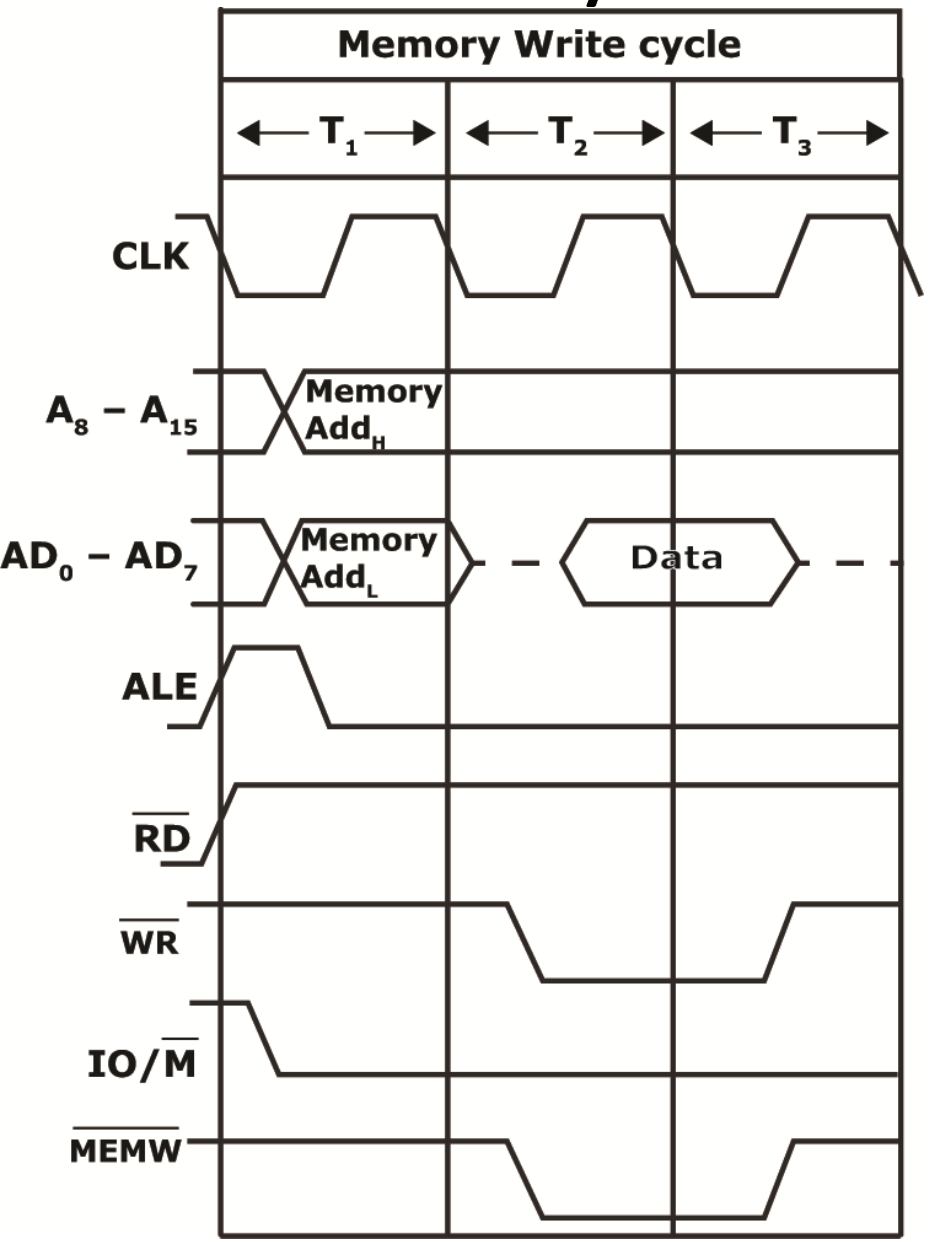


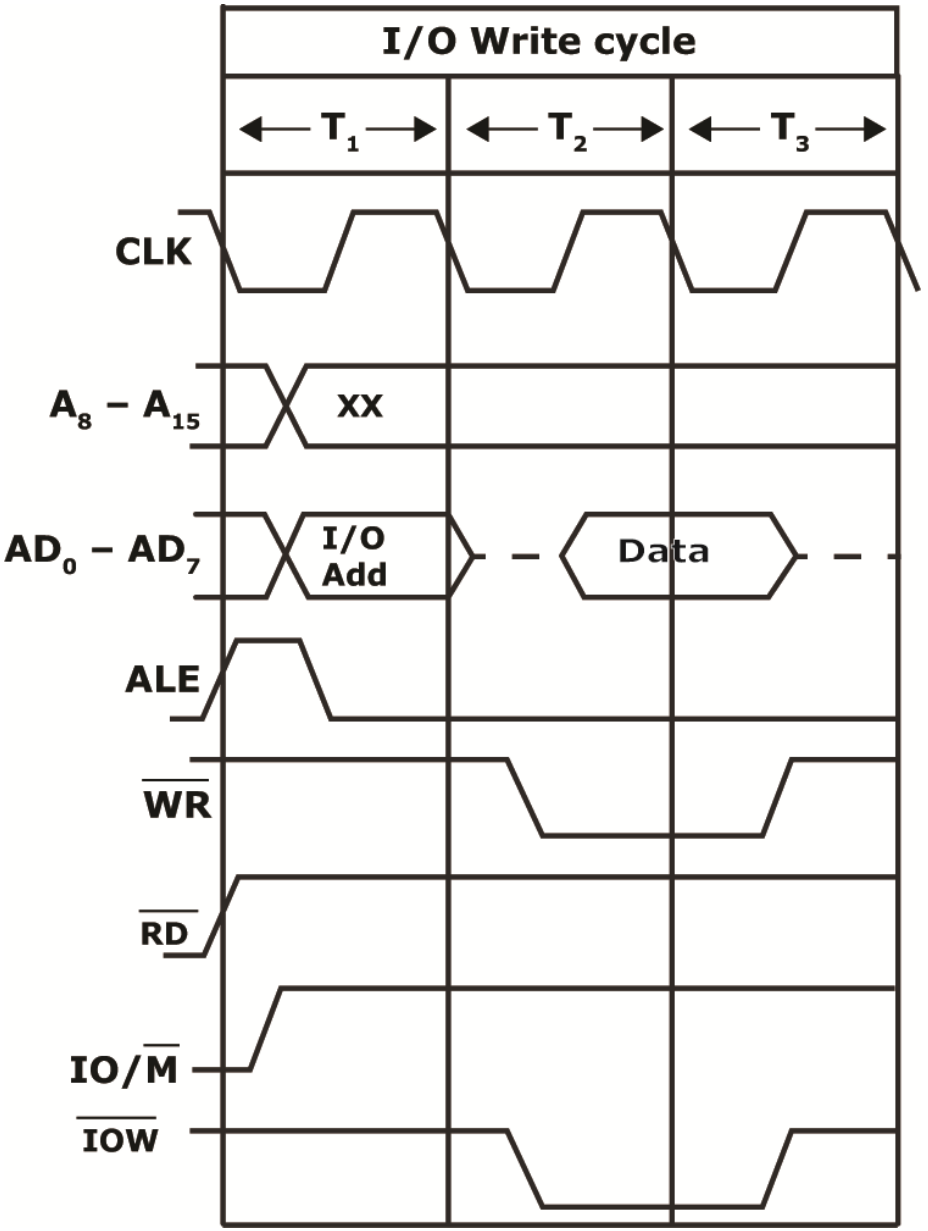
Machine Cycles & Bus Timing Diagrams





Machine Cycles & Bus Timing Diagrams





EXAMPLE: Draw the timing diagram of MVI B,20H. Also calculate the execution time for the same instruction.

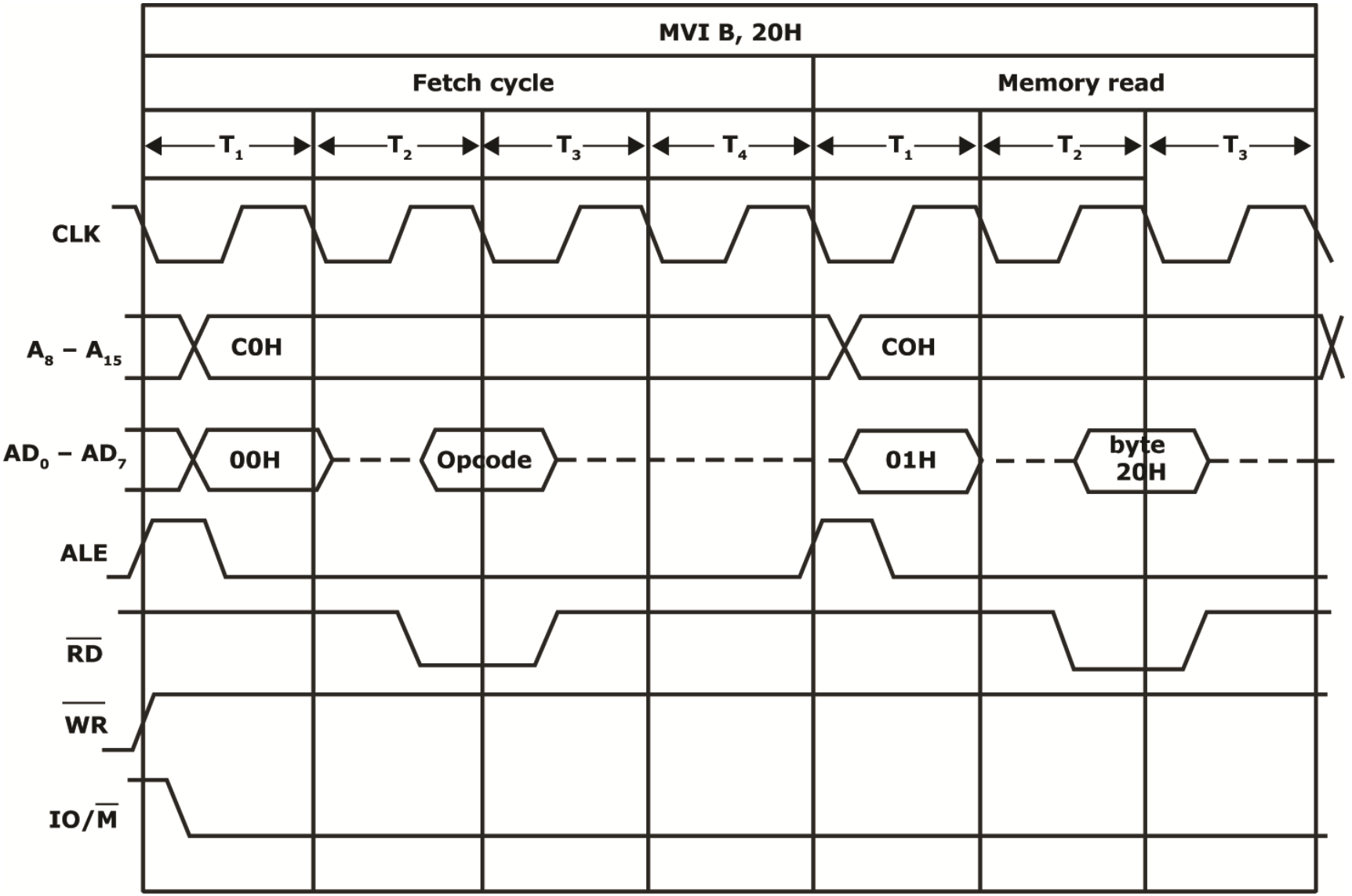
- Let's consider the instruction MVI B,20H stored at memory location C000H

  C000H: 06H

  C001H: 20H

- Here two machine cycles are presented, first is Op-Code fetch which consists of 4 clock cycles and second is memory read consist of 3 clock cycle
- The complete cycle is of 7 clock cycles (T-states)
- Opcode fetch is similar to above example
- At the end of Op-code fetch cycle, Program Counter (PC) increments to C001H and the instruction decoder contains 06H. After completion of the Op-code fetch cycle, the 8085 places the address C001H on address bus and increments the program counter to the next address C002H
- Microprocessor initiates the memory read machine cycle (IO/M=0, S=1 & So=0) of three clock periods. At T2, the RD signal becomes active and enables the memory chip
- During T2 state, the 8085 activates the data bus as an input bus, memory places the data byte 20H on the data bus, and the 8085 reads and stores the byte in the B register during T3

Machine Cycles & Bus Timing Diagrams



- For Execution time of instruction MVI B, 20H , since operating frequency of 8085 microprocessor is 3MHz

Clock period(T)=1/f= 1/3MHz=0.33µs

Execution time for opcode fetch :(4T)*0.33=1.32µs

Execution time for memory read :(3T)*0.33=0.99µs

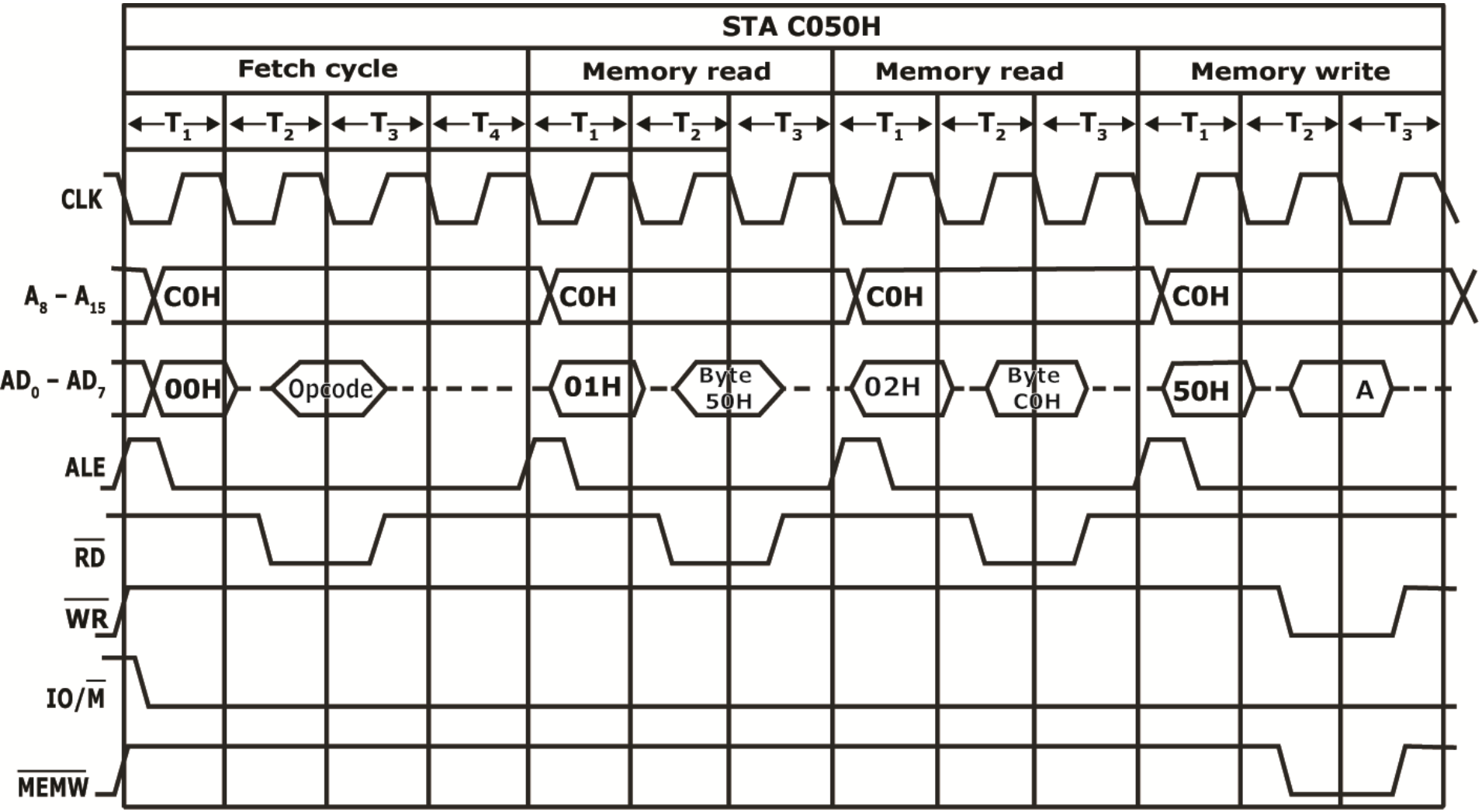
Execution time for instruction=1.32µs+0.99µs=2.31µs

Example:
Draw the bus timing diagram of STA C050H stored at memory location C000H.

Let us consider a 3 byte instruction - STA C050H stored in memory locations C000H, C001H, & C002H, which has four machine cycles with 13 T-states. Execution steps are as follows:

- In the first machine cycle, the 8085 places the address C000H on the address bus and fetches the Op-code 32H. **(4 T-states)**
- The second machine cycle is Memory Read. The processor places the address C001H and gets the low-order byte 50H. **(3 T-states)**
- The third machine cycle is also memory read; the 8085 gets the high order byte C0H from memory location C002H. **(3 T-states)**
- The last machine cycle Memory Write; The 8085 places the address C050H on the address bus, identifies the operation as Memory Write (IO/M=0, S,=0 & So=1). It places the contents of the accumulator on the data bus AD0 –AD7 and asserts the WR signal. During the last T- state, the contents of the data bus are placed in memory location C050H. **(3 T-states)**

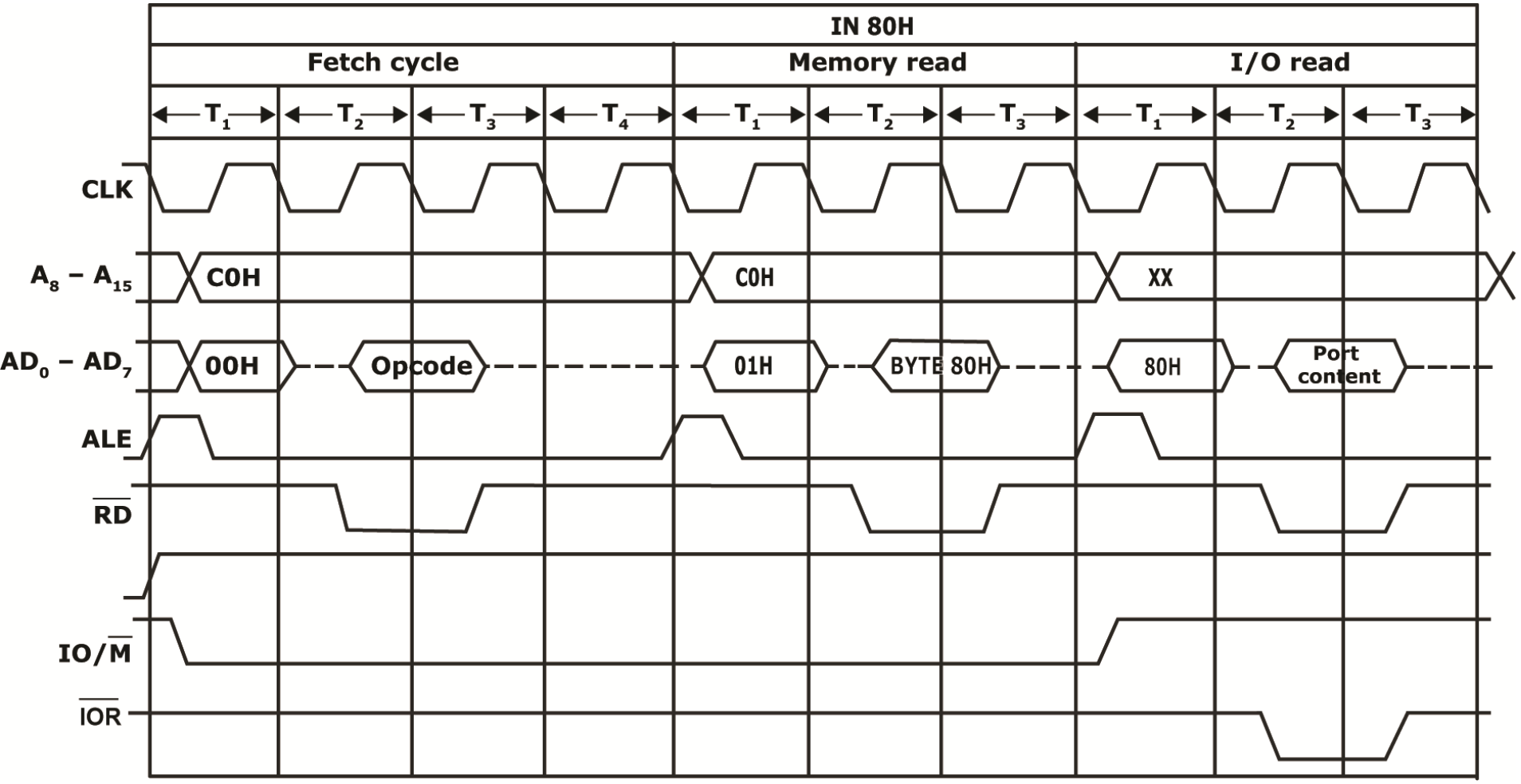
Machine Cycles & Bus Timing Diagrams



Example: Draw timing diagram for the instruction IN 80H

- The 8085 has IN instruction to read the data from input device. Let us consider IN 80H is stored in memory location C000H. This is two byte instruction with first byte op-code and the second byte the port address.
- When the microprocessor is asked to execute this instruction, it will first read the machine codes(or bytes) stored at locations C000H and C001H, then read the switch positions at port 80H by enabling the interfacing device of the input port.
- It has three machine cycle with earlier two (M1- opcode fetch & M2- memory read) cycles are identical to above instruction cycle. The last machine cycle M3 is I/O read machine cycle (3T-states), in which microprocessor places the address of the input port (84H) on low order address bus (ADO-AD7) as well as on the higher-order address bus (A8-A15) and asserts the RD signal, which is used to generate I/O read (TOR) signal.
- The IOR enables input port, and the data from the input port are placed on the data bus and transferred into the accumulator.

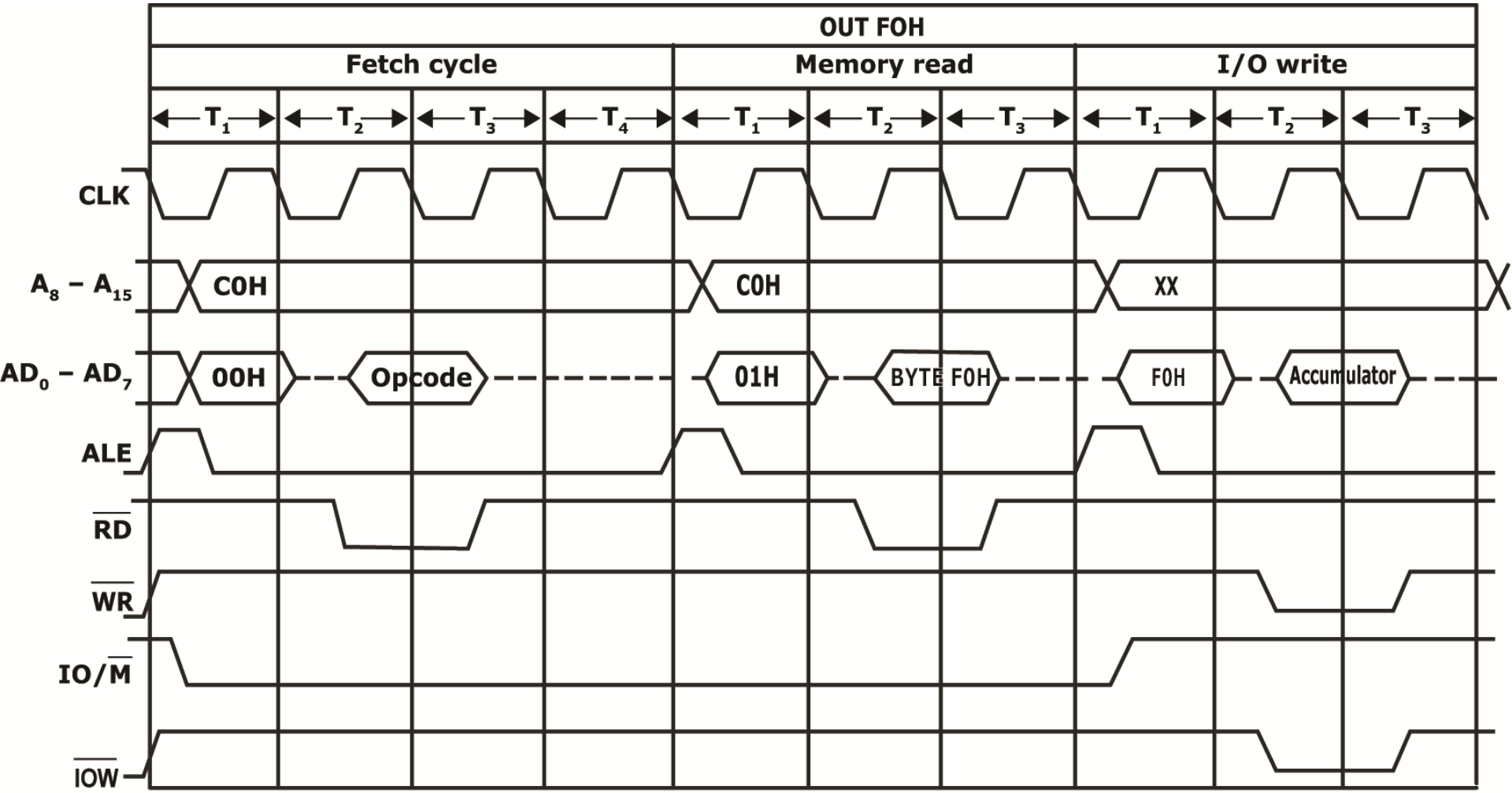
Machine Cycles & Bus Timing Diagrams



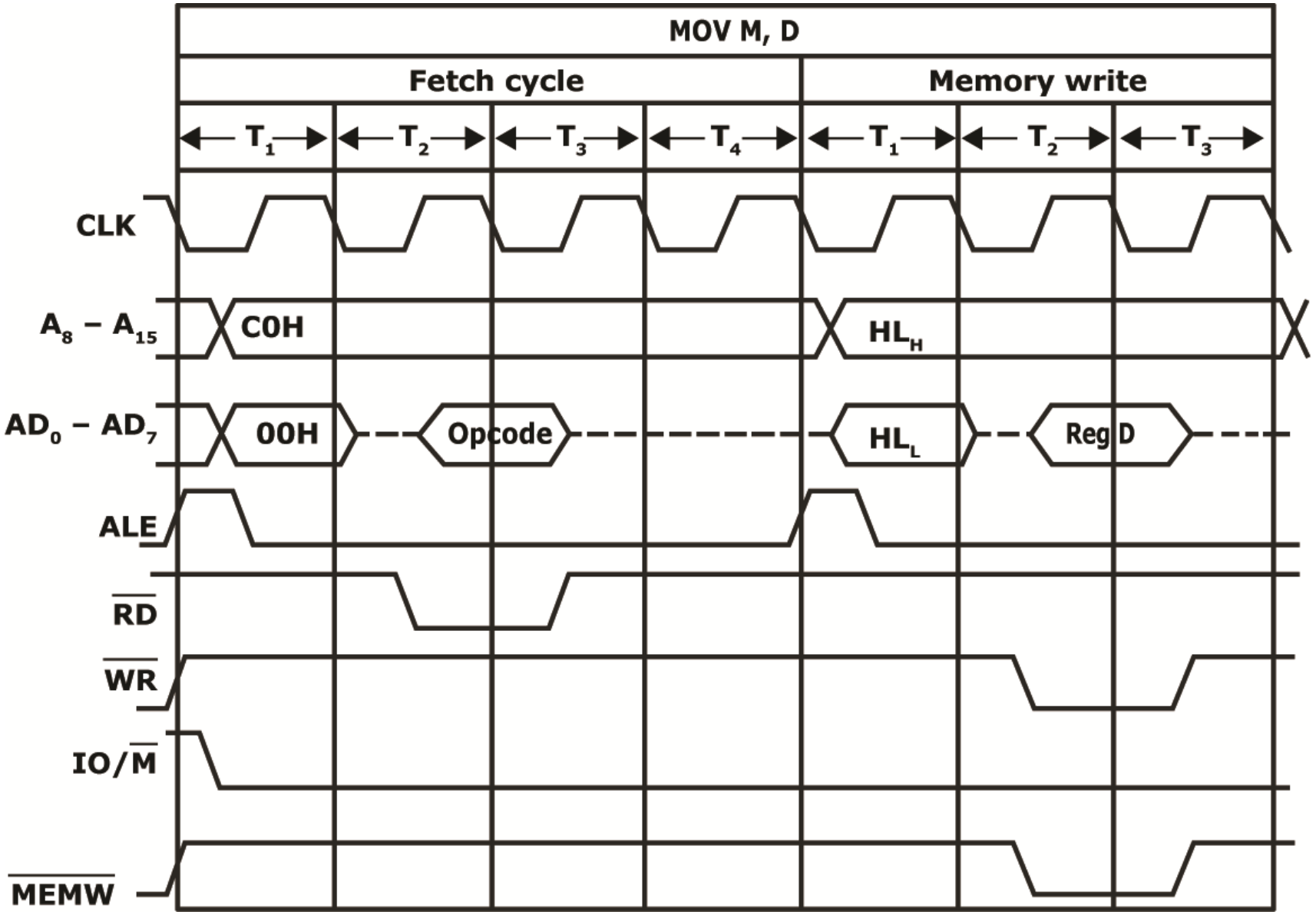
Example: Draw timing diagram for the instruction OUT F0H.

- Let us consider the instruction OUT F0H stored at memory location C000H. The Op-code of this instruction is D3H and the complete cycle is of 10 clock cycles (T-states). The execution of this instruction consists of three machine cycle Op-code fetch, memory read and I/O Write. The Op-code fetch and-memory read cycle are exactly similar to the previous one except the contents of the buses.
- In the third machine cycle, M3 (I/O Write), the 8085 places the device address F0H on the low order address as well as the high order address bus. The IO/M signal goes high to indicate that it is an I/O operation. At T2 the accumulator contents are placed on the data bus, followed by the control signal WR. By ANDing (logical operation) the IO/M and WR signals, the IOW signal can be generated to enable output device.

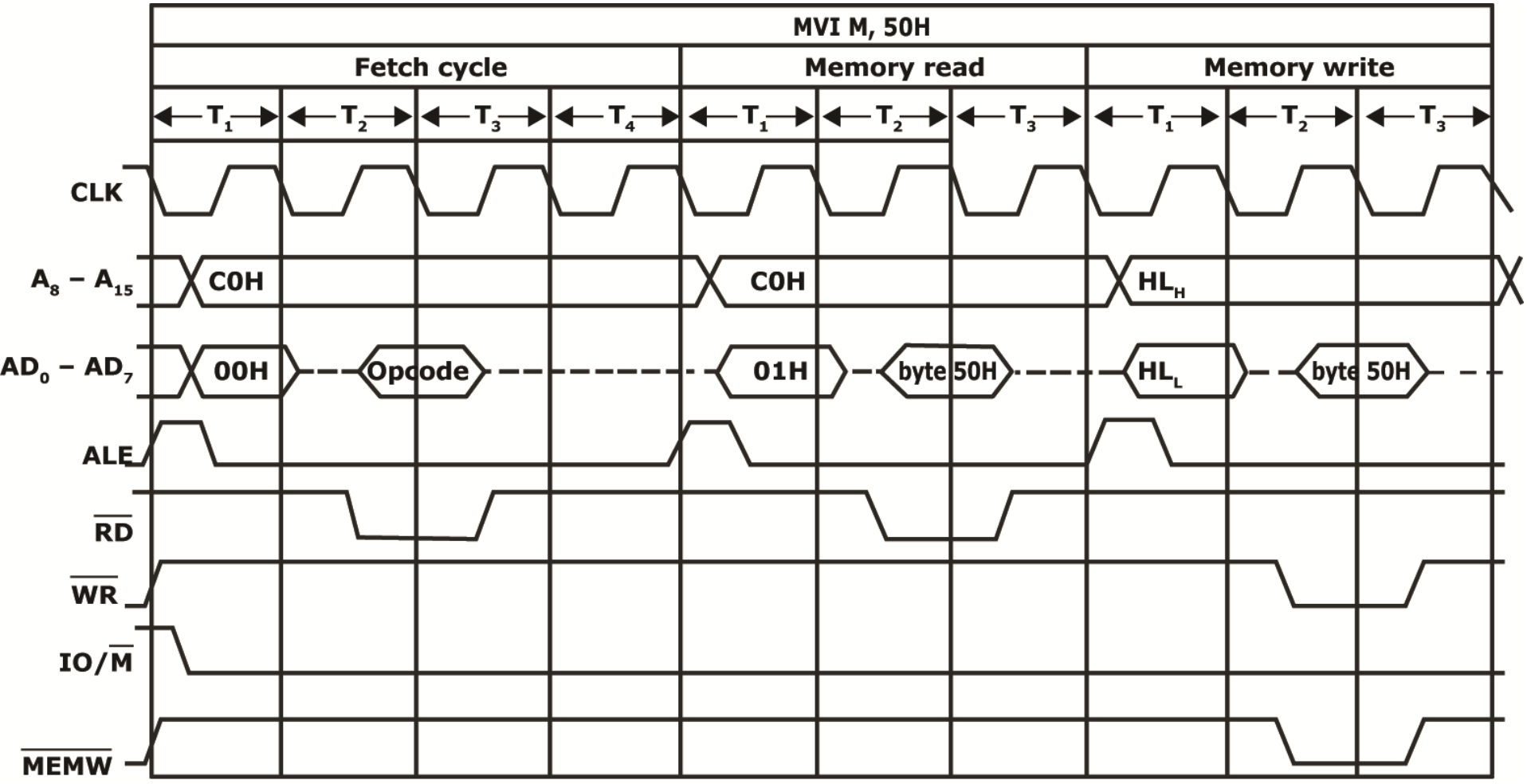
Machine Cycles & Bus Timing Diagrams



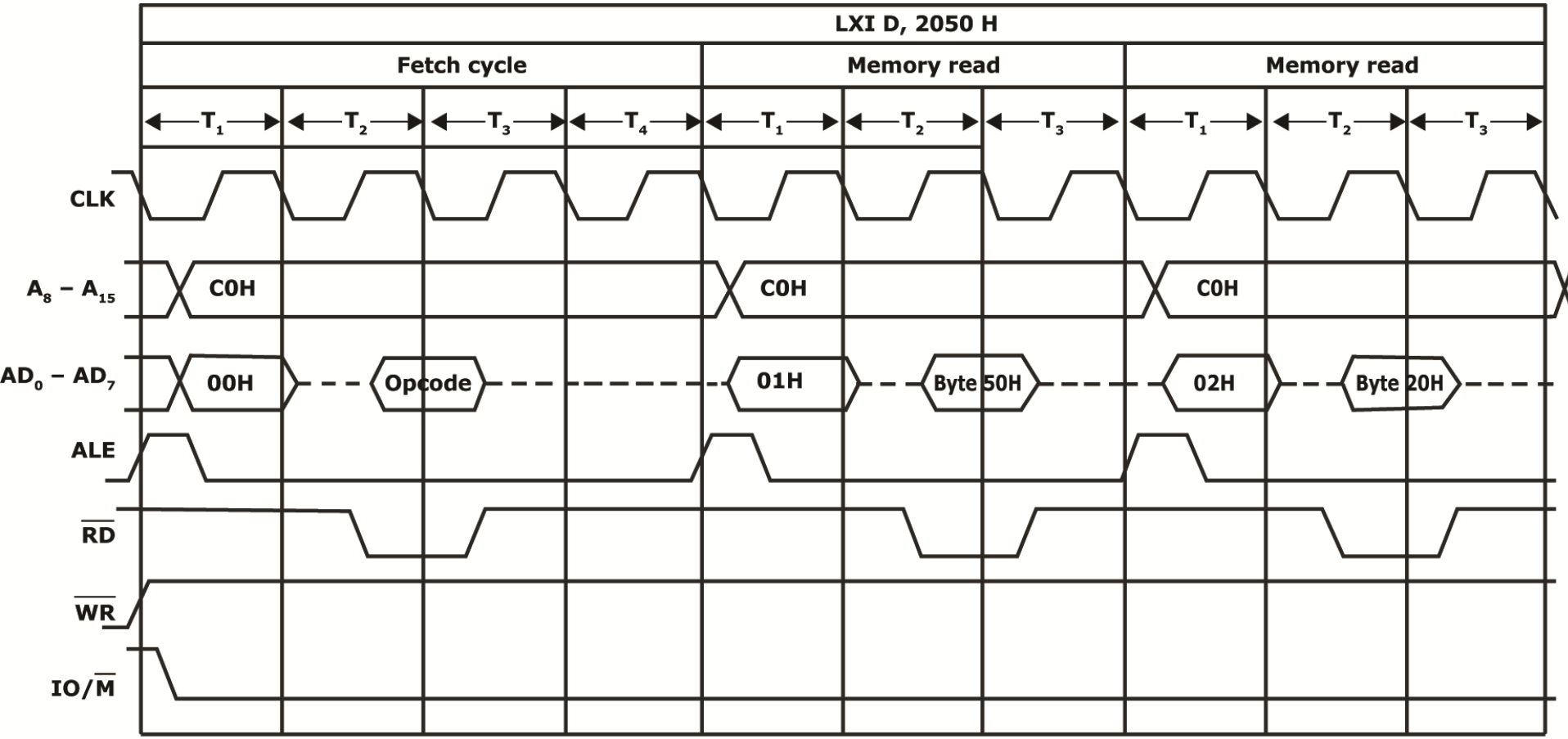
Machine Cycles & Bus Timing Diagrams



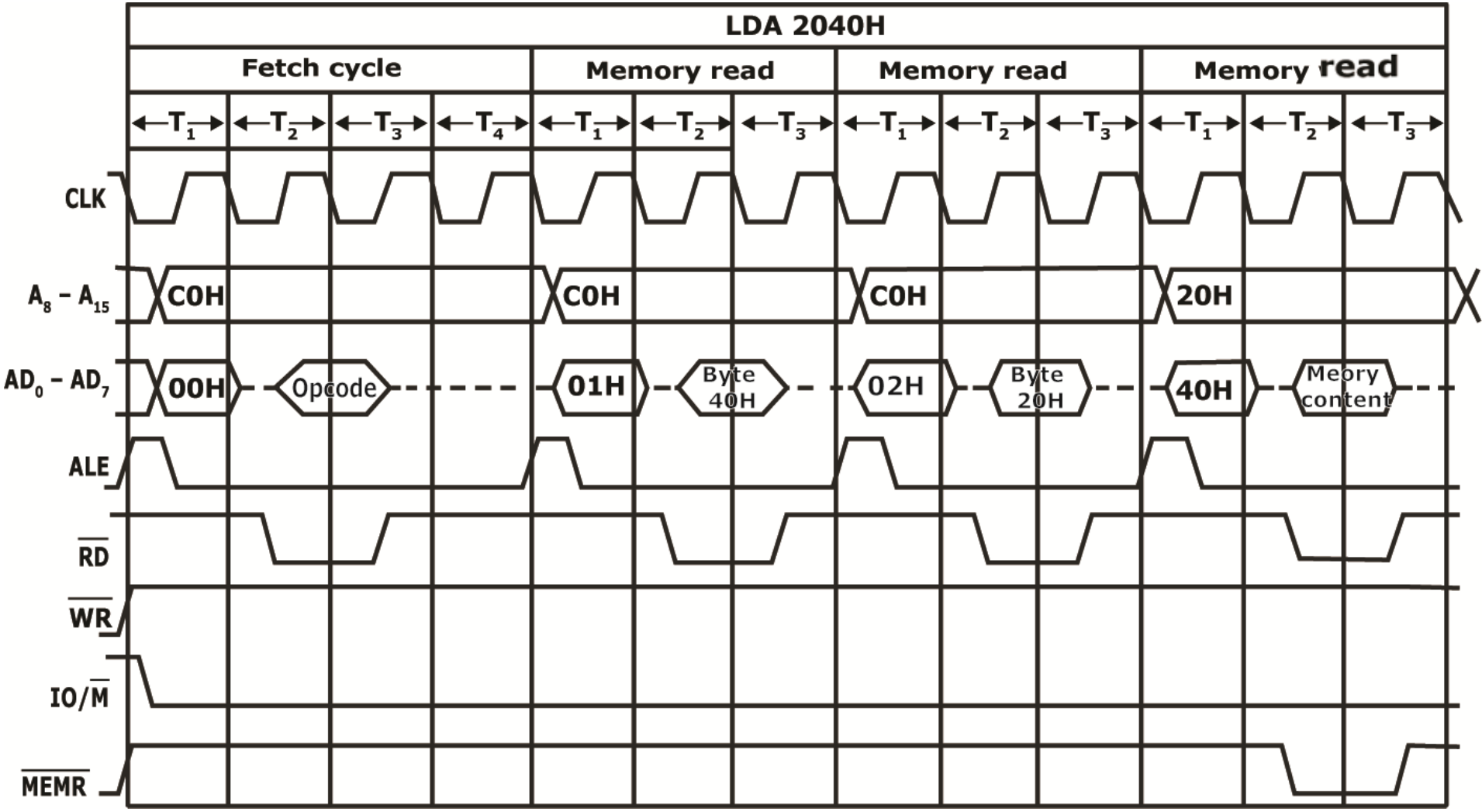
Machine Cycles & Bus Timing Diagrams



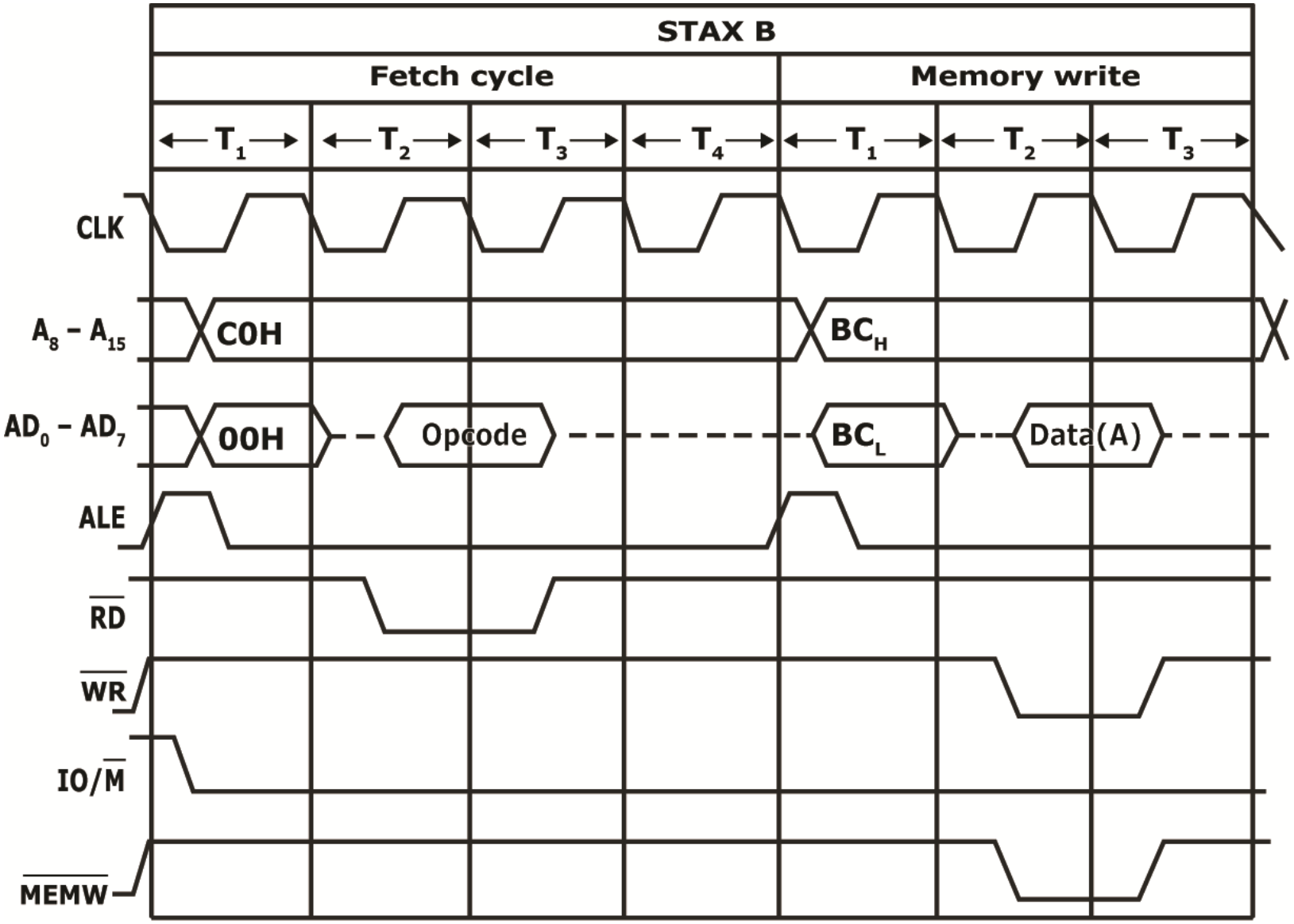
Machine Cycles & Bus Timing Diagrams



Machine Cycles & Bus Timing Diagrams



Machine Cycles & Bus Timing Diagrams



Rotate Instruction

- Each bit in the accumulator can be shifted either left or right to the next position

  - RLC(Rotate Accumulator left)
  - RAL(Rotate Accumulator left through carry)
  - RRC(Rotate Accumulator right)
  - RAR(Rotate Accumulator right through carry)

(Note: Rotate instruction are primarily used in arithmetic multiply and divide operations and serial data transfer)

Rotate Instruction

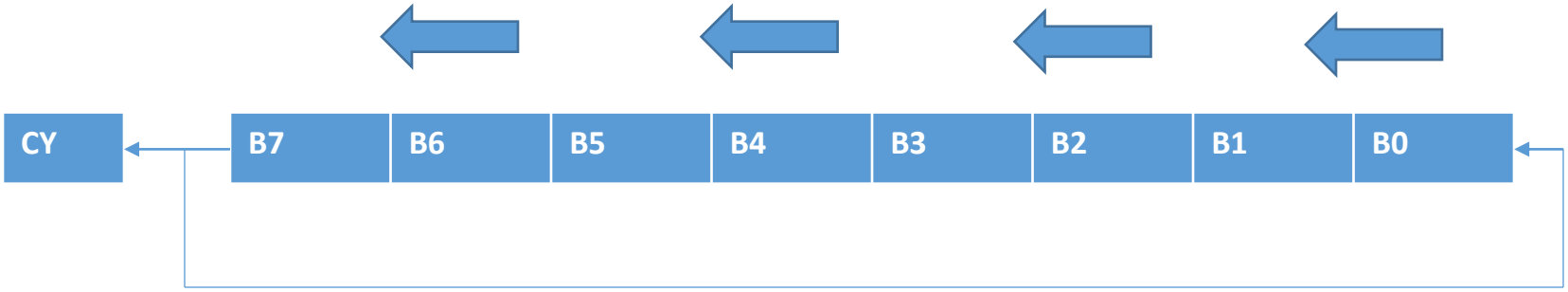
➢ RLC(Rotate Accumulator left)

Opcode	Operand	Description
RLC	None	Rotate accumulator left

- Each binary bit of the accumulator is rotated left by one position
- Bit D7 is placed in the position of D0 as well as in the Carry flag
- CY is modified according to bit D7
- S, Z, P, AC are not affected
- Example: RLC.

(Note: Rotating left can be viewed as multiplying by 2 but it is valid only if MSB is 0)

Before Execution



After Execution

Example:

MVI A,55H

RLC

0101 0101(55H)

RLC

1010 1010(AAH)

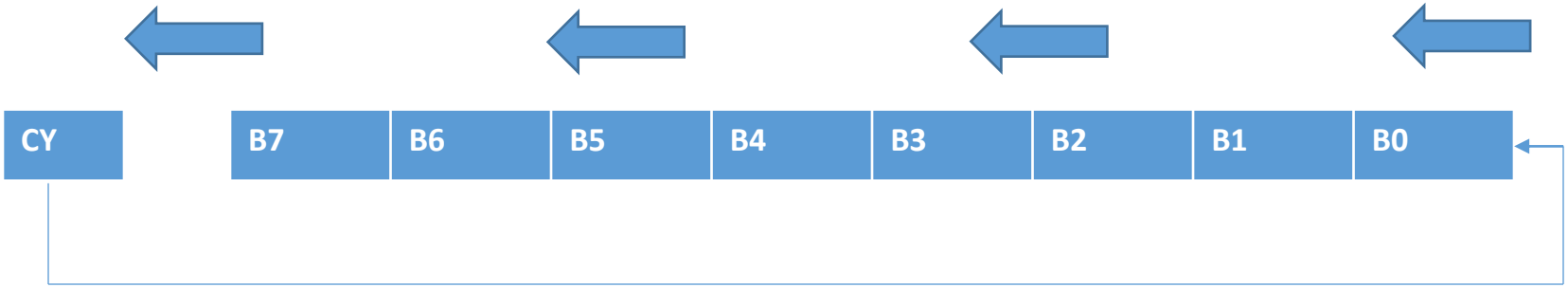
Rotate Instruction

- RAL(Rotate Accumulator left through carry)

Opcode	Operand	Description
RAL	None	Rotate accumulator left through carry

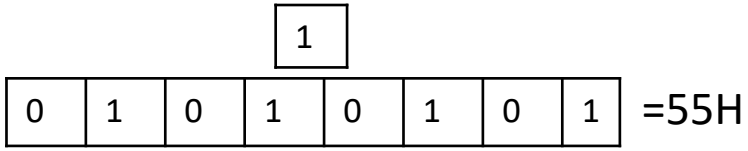
- Each binary bit of the accumulator is rotated left by one position through the Carry flag
- Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0
- CY is modified according to bit D7
- S, Z, P, AC are not affected
- Example: RAL.

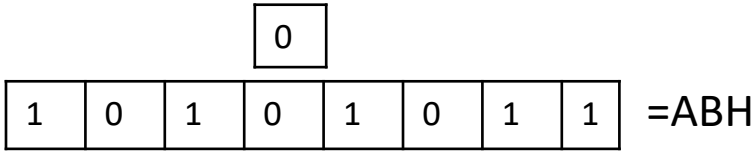
Before Execution



After Execution

Example:
If CY=1
MVI A, 55H
RAL





Rotate Instruction

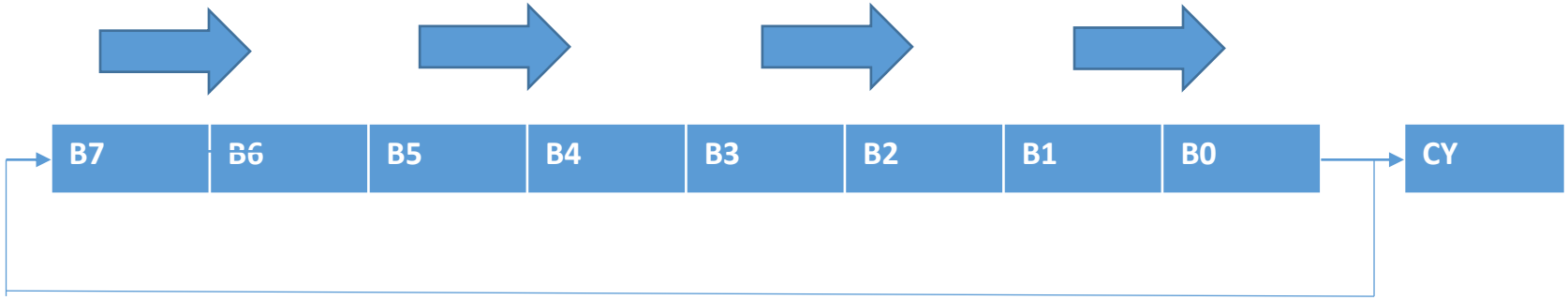
➢ RRC(Rotate Accumulator right)

Opcode	Operand	Description
RRC	None	Rotate accumulator right

- Each binary bit of the accumulator is rotated right by one position
- Bit D0 is placed in the position of D7 as well as in the Carry flag
- CY is modified according to bit D0
- S, Z, P, AC are not affected
- Example: RRC

(Note: Rotating right can be viewed as dividing by 2 but it is valid only if LSB is 0)

Before Execution



After Execution

Example:

MVI A,81H

RRC

1000 0001(81H)

RRC

1100 0000(C0H)

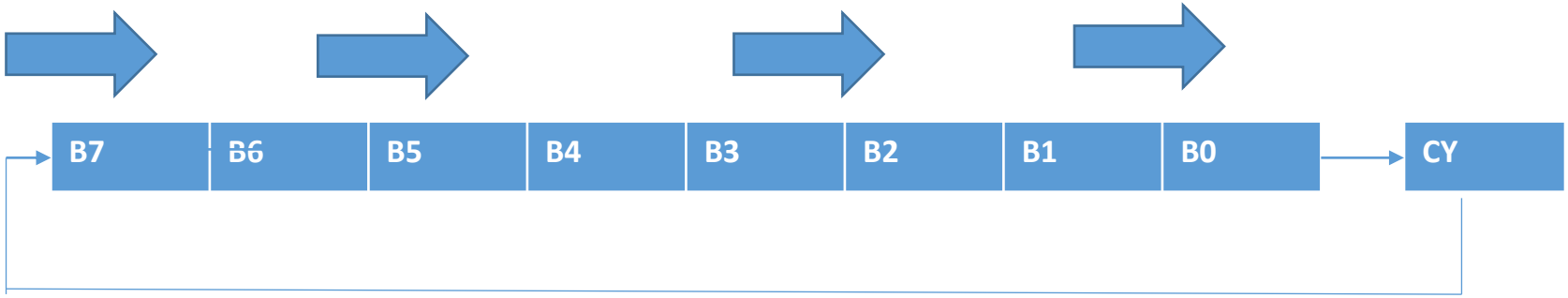
Rotate Instruction

- RAR(Rotate Accumulator right through carry)

Opcode	Operand	Description
RAR	None	Rotate accumulator right through carry

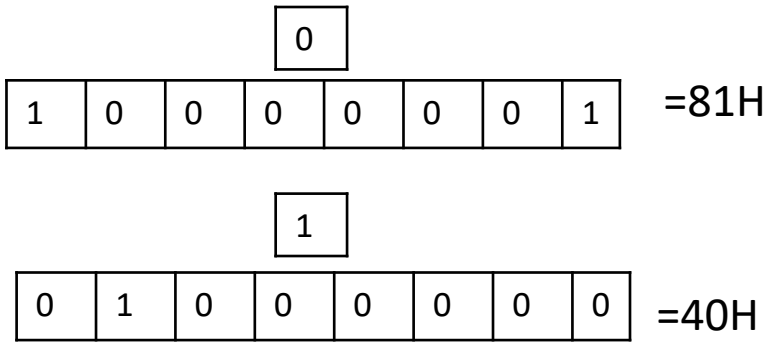
- Each binary bit of the accumulator is rotated right by one position through the Carry flag
- Bit D0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D7
- CY is modified according to bit D0
- S, Z, P, AC are not affected
- Example: RAR.

Before Execution



After Execution

Example:
If CY=0 ,
MVI A, 81H
RAL



Question

- Write a program to multiply 08H by 2 store the result in memory location 5050H and divide 08H by 2 store the result in Memory location 5060H.

MVI A, 08H

MOV B,A

RLC

STA 5050H

MOV A,B

RRC

STA 5060H

HLT

(Note: this procedure is invalid when logic 1 is rotated left from D7 to D0 or vice versa)

Branching Instructions

- The branching instruction alter the normal sequential flow
- These instructions alter either unconditionally or conditionally
- Branch instructions are categorised as:
  - Jump instructions
  - Call and Return instructions
  - Restart instructions

Jump instructions(Unconditional Jump)

➢ JMP

Opcode	Operand	Description
JMP	16-bit address	Jump unconditionally

- The program sequence is transferred to the memory location specified by the 16-bit address given in the operand
- 3 byte instruction
- Example: JMP 2034H.

Jump instructions(Unconditional Jump)

- Program sequence is transferred to the memory location specified by 16-bit address.

Example:

memory address	Instructions
9000H :	MVI A, 20H
9002H:	MVI B, 30H
9004H:	JMP 9040H
9007H :	……
	……..
9040H :	ADD B
9041H:	HLT

Jump instructions(Conditional Jump)

- Allows microprocessor to make decisions based on certain conditions indicated by flags
- Program sequence is changed based on flag conditions
- Example: JC 5050H

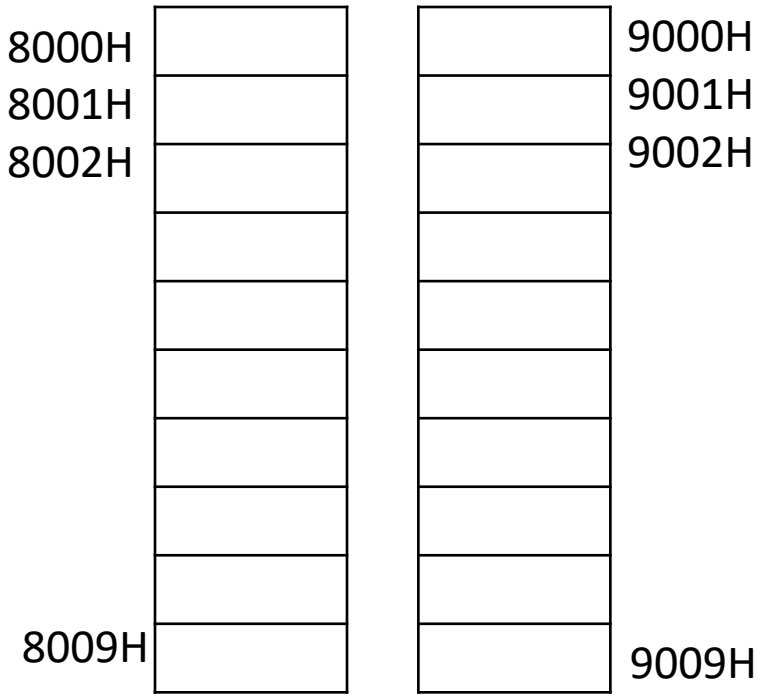
Opcode	Description	Status Flags
JC	Jump if Carry	CY = 1
JNC	Jump if No Carry	CY=0
JZ	Jump if Zero	Z=1
JNZ	Jump if No Zero	Z=0
JPE	Jump if Parity Even	P=1
JPO	Jump if Parity Odd	P=0
JP	Jump if positive	D7=0 i.e. S=0
JM	Jump if negative	D7=1 i.e. S=1

C-Carry , Z-Zero , P-Parity, S-Sign

Question

Write a program in 8085 to transfer 10 8-bit numbers stored in memory starting from 8000H to memory starting from 9000H.

```
        LXI H, 8000H  ; SOURCE TABLE
        LXI B, 9000H  ; B<-90, C<-00
        MVI D, 0AH    ; COUNTER
UP:     MOV A,M       ; A<-[HL]
        STAX B        ; [BC]<-A
        INX H         ; HL<-HL+1
        INX B         ; BC<-BC+1
        DCR D         ; D<-D-1
        JNZ UP
        HLT
```



Question

- Write a program in 8085 to read two numbers from memory C050H and C060H. If number in C050H is greater , subtract the number in C050H from number in C060H else add two numbers.

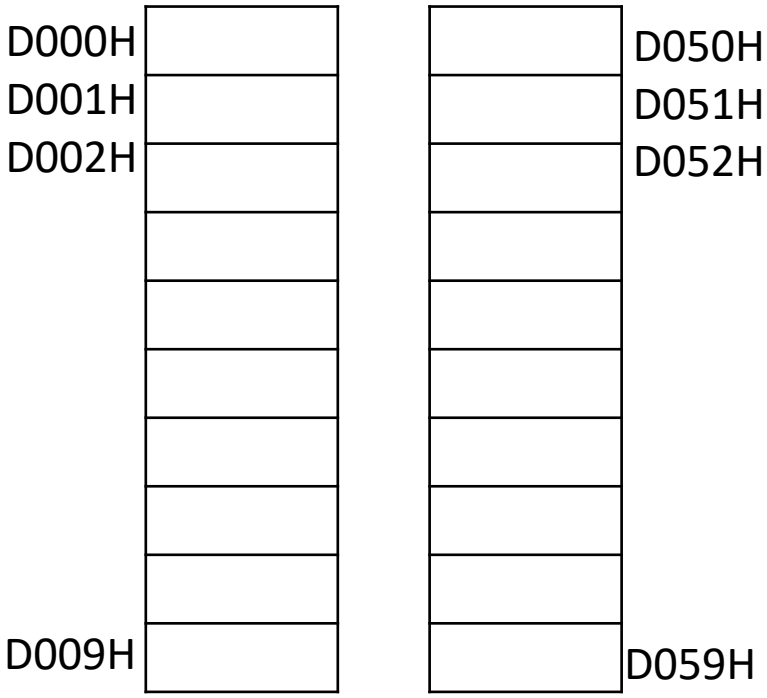
```
        LDA C060H; A<-[C060H]
        MOV B,A
        LDA C050H; A<-[C050H]
        CMP B; A-B
        JC PASS
        MOV C,A
        MOV A,B

        SUB C
        JMP LAST
PASS: ADD B
LAST: HLT
```

Condition	Zero flag (Z)	Carry flag(CY)
A> R/M	0	0
A=R/M	1	0
A<R/M	0	1

Question

- 10 8-bit numbers are stored in memory starting from D000H. Write a program in 8085 to transfer these numbers to memory starting from D050H only if number of source table is less than FFH, else store 00H in destination.

```
        LXI H, D000H; SOURCE TABLE
        LXI D, D050H; DESTINATION TABLE
        MVI B, 0AH; COUNTER
UP:     MOV A, M; A<-[HL]
        CPI FFH; A-FFH
        JC PASS
        MVI A, 00H
PASS:   STAX D; [DE] <- A
        INX H
        INX D
        DCR B
        JNZ UP
        HLT
```



CALL and RETURN instructions

Subroutine:

- Group of instructions written separately from main program to perform a function that occurs repeatedly in main program
- Two instructions to implement subroutine:
  - CALL
  - RET
- CALL instruction is used in main program to call a subroutine
- RET instruction is used in the end of subroutine to return to main program from a subroutine
- When a subroutine is called, the contents of the program counter which is the address of the next instruction following the CALL instruction is stored on the stack and program execution is transferred to the subroutine address
- When the RET instruction is executed at the end of the subroutine the memory address stored on the stack is retrieved
- The sequence of execution is resumed in main program

CALL and RETURN instructions

➢ CALL (Unconditional Call)

Opcode	Operand	Description
CALL	16-bit address	Call unconditionally

- The program sequence is transferred to the memory location specified by the 16-bit address given in the operand
- Before the transfer, the address of the next instruction after CALL (the contents of the program counter) is pushed onto the stack.
- No flags are affected
- Example: CALL 2034 H

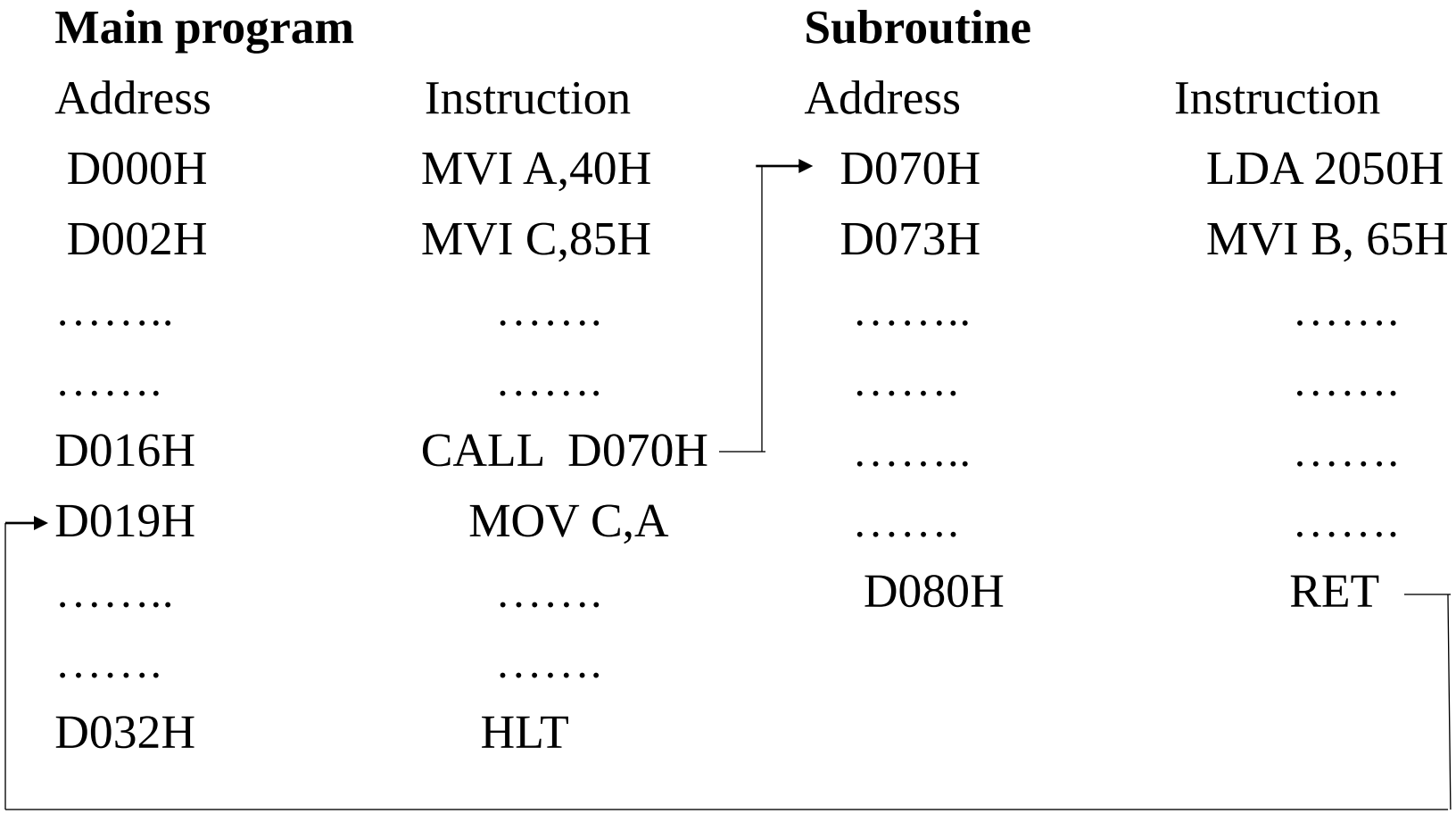
CALL and RETURN instructions

➢ RET(Unconditional return)

Opcode	Operand	Description
RET	None	Return unconditionally

- The program sequence is transferred from the subroutine to the calling program
- The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address
- No flags are affected
- Example: RET.

CALL and RETURN instructions



CALL and RETURN instructions

How CALL and RET works?

For CALL

- Saves the content of PC as a return address into the stack
- Loads the PC with new subroutine address

For RET

- Pops the return address from stack
- Loads that return address into the PC

Data Transfer Instructions

- **Exchange the value of HL and DE**
- XCHG; H ⟷ D, L ⟷ E
- Example:

  LXI H,8757H; H ← 87H, L ← 57H

  LXI D,5065H; D ← 50H ,E ← 65H

  XCHG;

Before Execution

D	50	E	65
H	87	L	57

XCHG

After Execution

D	87	E	57
H	50	L	65

Data Transfer Instructions

- **Load/Store H&L directly**

Opcode	Operand
LHLD	16-bit address
SHLD	16-bit address

- Example:

  LHLD 3035H; L<-[3035], H<-[3035+1]

  SHLD C035H; [C035]<-L, [C035+1]<-H

Data Transfer Instructions

- **Between register and stack memory**
- PUSH

Opcode	Operand
PUSH	Reg-Pair

- The contents of register pair are copied onto stack
- SP is decremented and the contents of high-order registers (B, D, H, A) are copied into stack
- SP is again decremented and the contents of low-order registers (C, E, L, Flags) are copied into stack
- Example: PUSH B

Data Transfer Instructions

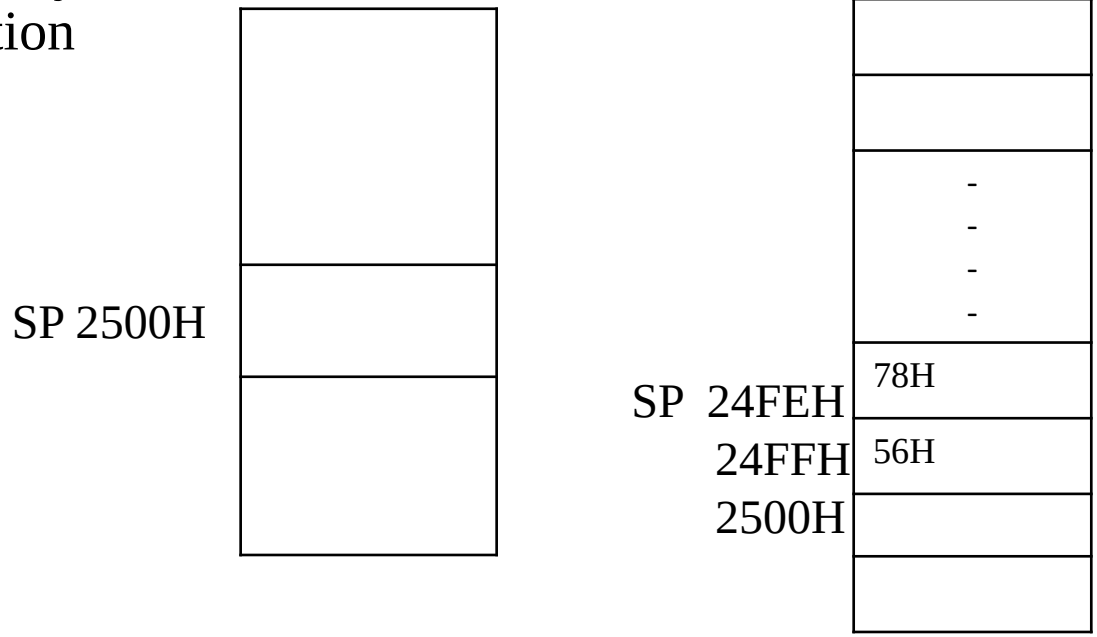
- For example:

  LXI H 5678H; H ← 56H, L ← 78H

  LXI SP, 2500H; initialise stack pointer to 2500H

  PUSH H;

After the execution of PUSH H instruction 56 H will be loaded to memory location 24FF H and 78 H will be loaded to 24FE H memory location



Data Transfer Instructions

- POP

Opcode	Operand
POP	Reg-Pair

- The contents of top of stack are copied into register pair
- The contents of location pointed out by SP are copied to the low-order register (C, E, L, Flags)
- SP is incremented and the contents of location are copied to the high-order register (B, D, H, A)
- SP is again incremented by 1
- Example: POP H

Suppose the data saved in stack is shown in figure. The stack pointer is shown as 24FC H.

Let

D = 12 H E = 6E H

before the execution of

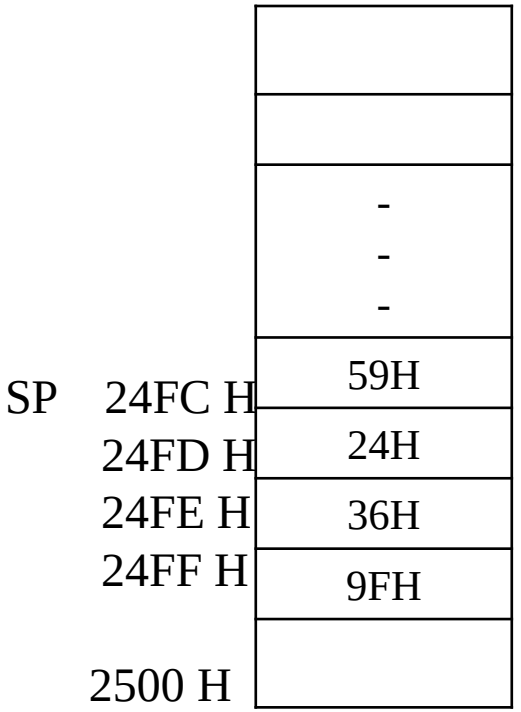
POP D instructions.

After the execution of these instructions we have the following data in the registers.

E = 59H

D = 24 H

Now 24FE H will be the new value of stack pointer.



Data Transfer Instructions

- **Between input port and accumulator**

Opcode	Operand
IN	8-bit Port address; A ← [port address]

- The content of input port is copied into accumulator
- Example : **IN 8C H**
- Before Execution

In Port 8CH

- After Execution

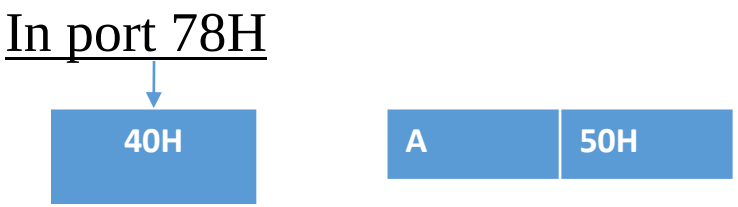
In Port 8CH

Data Transfer Instructions

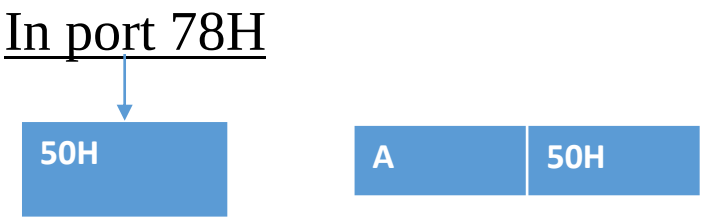
- **Between output port and accumulator**

Opcode	Operand
OUT	8-bit Port address; A ← [port address]

- The content of accumulator is copied into the output port
- Example: **OUT 78H**
- Before Execution



- After Execution



Data Transfer Instructions

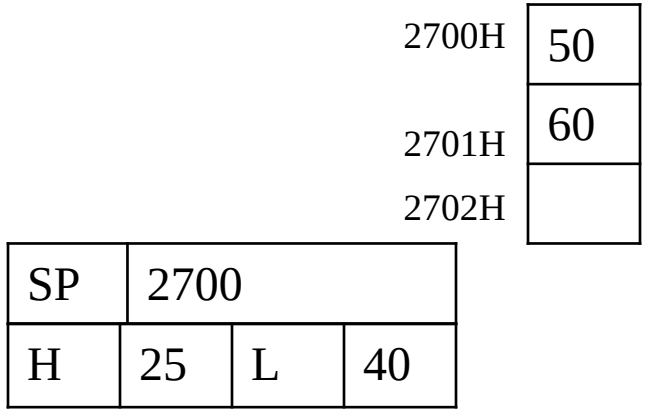
- **Between stack memory and HL**
- XTHL

Opcode	Operand
XTHL	None

- The contents of L register are exchanged with the location pointed out by the contents of the SP
- The contents of H register are exchanged with the next location (SP +1)
- Example: XTHL; [SP] ⟷ L, [SP+1] ⟷ H

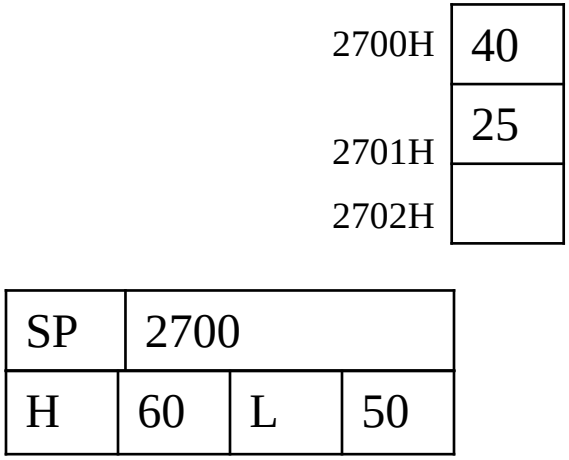
Data Transfer Instructions

Before Execution



XTHL

After Execution



Data Transfer Instructions

- **Between stack pointer (SP) and HL**
- SPHL

Opcode	Operand	Description
SPHL	None	This instruction loads the contents of H-L pair into SP.

- Example: LXI H,2500H; H<- 25H, L<- 00H
  SPHL; SP<- HL

Before Execution

SP			
H	25	L	00

After Execution

SP	2500		
H	25	L	00

Arithmetic Instructions

- These instructions perform the following operations :
  - Addition
  - Subtract
  - Increment
  - Decrement
  - Comparison

Addition

- Any 8-bit number, or the contents of register, or the contents of memory location can be added to the contents of accumulator.
- The result (sum) is stored in the accumulator
- No two other 8-bit registers can be added directly

❖Example: The contents of register B cannot be added directly to the contents of register C.

➢ **ADD(Add register to accumulator)**

Opcode	Operand	Description
ADD	R M	Add register or memory to accumulator

- The contents of register or memory are added to the contents of accumulator
- The result is stored in accumulator
- If the operand is memory location, its address is specified by H-L pair
- Example: ADD B or ADD M

Addition

Example:

MVI A,47H

MVI B, 21H

ADD B; A⟵ A+B

- Before Execution

A	47H	F	
B	21H	C	
D		E	
H		L	

ADD B

After Execution

A	68H	F	00H
B	21H	C	
D		E	
H		L	

Addition:

A: 47H= 0100 0111

B: + 21H= 0010 0001

68H= 0110 1000

Flags:

S	Z	X	AC	X	P	X	CY
0	0	X	0	X	0	X	0

Addition

Example:

LXI H 2050H

MVI A,76H

ADD M; A ← A+M

2050 | A2H

Before Execution

A	76	F	
B		C	
D		E	
H	20	L	50

ADD M

After Execution

A	18H	F	05H
B		C	
D		E	
H	20	L	50

Addition:

A : 76H= 0111 0110

$[2050]_{mem}$: A2H= +1010 0010

1/18H= 1/0001 1000

Flags:

S	Z	X	AC	X	P	X	CY
0	0	X	0	X	1	X	1

Addition

➢ **ADI(Add immediate to accumulator)**

Opcode	Operand	Description
ADI	8-bit data	Add immediate to accumulator

- The 8-bit data is added to the contents of accumulator
- The result is stored in accumulator
- All flags are modified to reflect the result of the addition

Addition

- Example:

  MVI A,4AH

  ADI 59H; A← A+59

Before Execution

A	4AH

ADI 59H

After Execution

A	A3H

Addition:

A : 4AH= 0100 1010

Data: +59H= 0101 1001

A3H= 1010 0011

Flags:

S	Z	X	AC	X	P	X	CY
1	0	X	1	X	1	X	0

Addition

- Question

1. Write a ALP(assembly language program) to add two A2H and 49H and store the result in register E. Also examine the flag register.

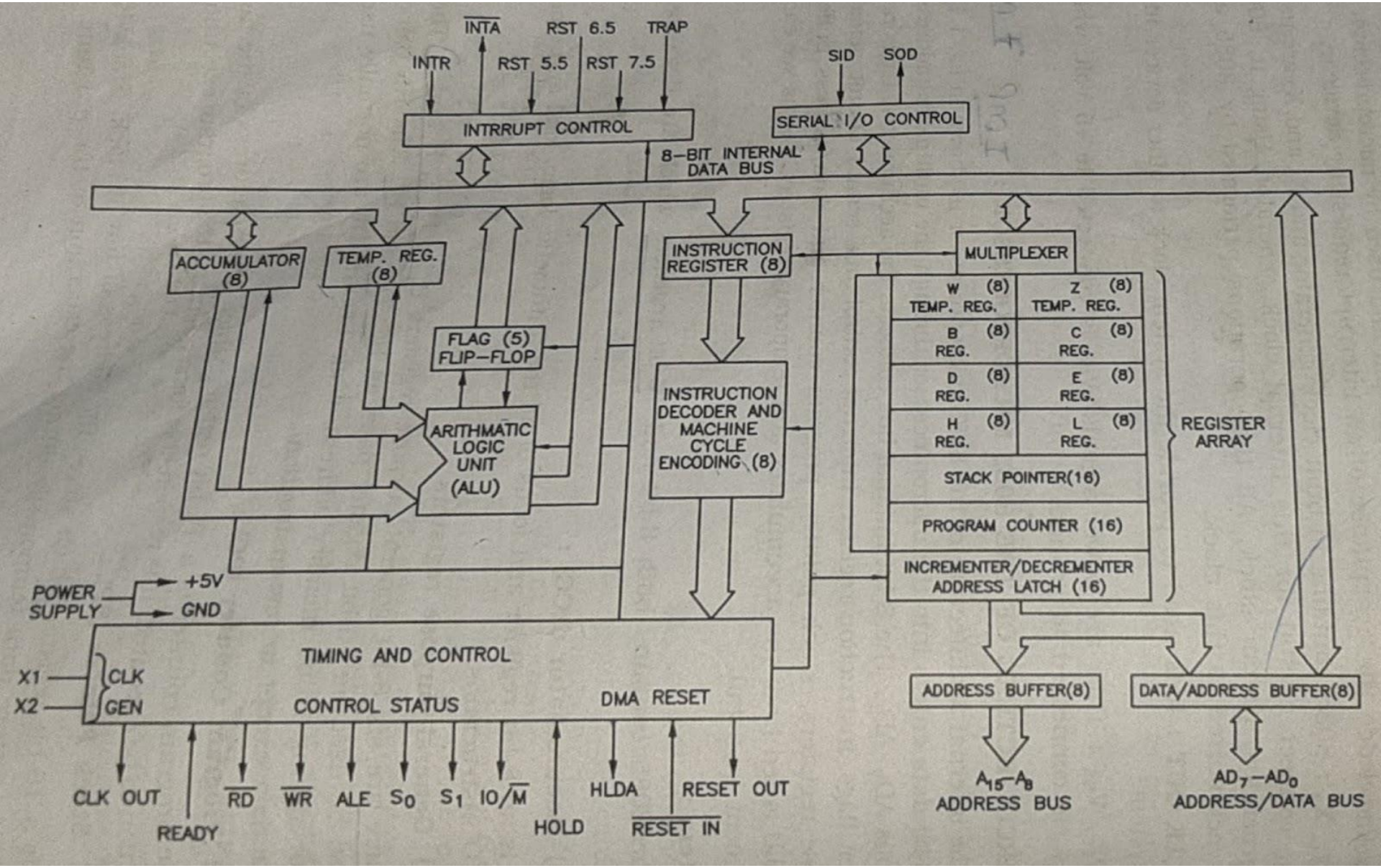
```
MVI A,A2H
ADI 49H
MOV E, A
HLT
```

```
MVI A, A2H
MVI B, 49H
ADD B
MOV E,A
HLT
```

2. Write a ALP(assembly language program) to add the contents of memory locations 8080H and 5206H. Add 65H to the obtained result and store the final result in C055H.

```
LDA 8080H
MOV B,A
LDA 5206H
ADD B
ADI 65H
STA C055H
HLT
```

Internal architecture of 8085 microprocessor



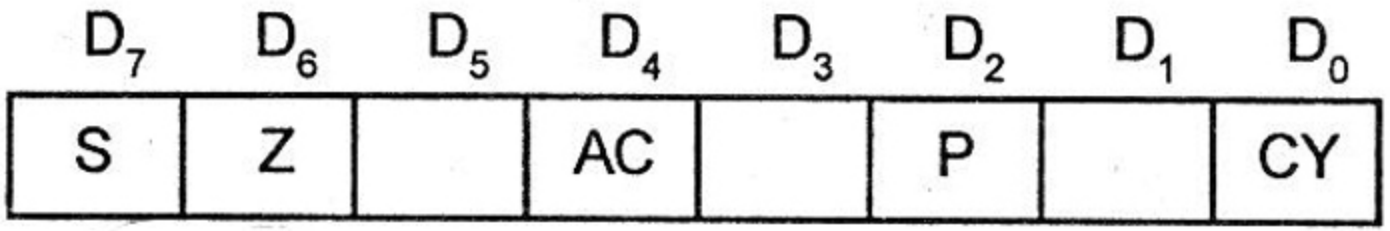
It has the following configuration −

- 8-bit data bus
- 16-bit address bus, which can address up to 64KB
- A 16-bit program counter
- A 16-bit stack pointer
- Six 8-bit registers arranged in pairs: BC, DE, HL
- Requires +5V supply to operate at 3.2 MHZ single phase clock

- ALU

- performs computing functions
- Includes the accumulator, the temporary register, the arithmetic and logic circuit and five flags
- Temporary register is used to hold data during arithmetic and logic operation
- Results are stored in accumulator
- Flags are set/reset according to the result of operation and are affected by the arithmetic and logic operations in ALU

FLAG REGISTER OF 8085



Flag is an 8-bit register containing 5 1-bit flags:

Sign - set if the most significant bit of the result is set.

Zero - set if the result is zero.

Auxiliary carry - set if there was a carry out from bit 3 to bit 4 of the result.

Parity - set if the parity (the number of set bits in the result) is even.

Carry - set if there was a carry during addition, or borrow during subtraction/comparison.

• S-sign flag – The sign flag is set if bit D7 of the accumulator is set after an arithmetic or logic operation(result negative).

• Z-zero flag – Set if the result of the ALU operation is 0. Otherwise is reset. This flag is affected by operations on the accumulator as well as other registers. (DCR B).

• AC-Auxiliary Carry – This flag is set when a carry is generated from bit D3 and passed to D4 . This flag is used only internally for BCD operations.

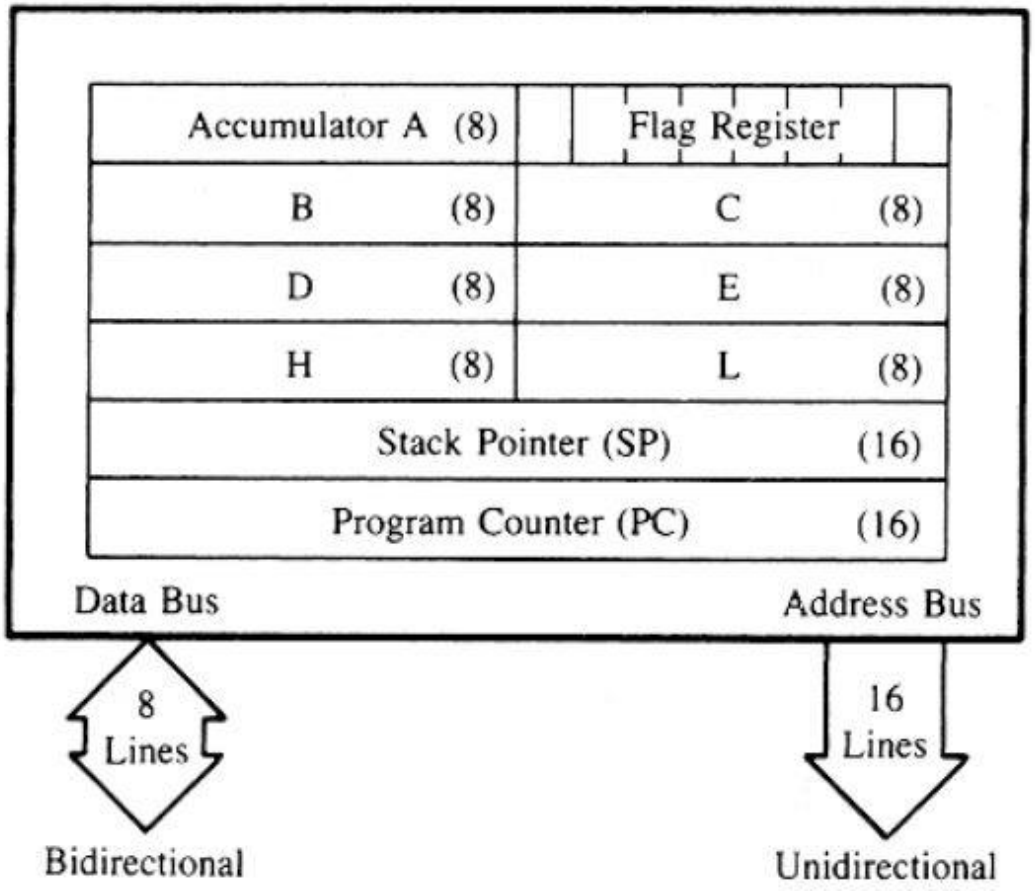
• P-Parity flag – After an ALU operation if the result has an even no. of 1's the p flag is set. Otherwise it is cleared. So, the flag can be used to indicate even parity.

• CY-carry flag – If an arithmetic operation results in a carry, the carry flag is set otherwise it is reset. It also serve as a borrow flag for subtraction.

- Timing and control unit
  - This unit synchronises all the microprocessor operations with the clock and generates the control signals necessary for communication between microprocessor and peripherals
  - RD and WR signals are control signals indicating the availability of data on the data bus
- Instruction register and decoder
  - When an instruction is fetched from memory, it is loaded in the instruction register
  - Decoder decodes the instruction and establishes a sequence of events to follow
  - Instruction register is not programmable and cannot e accessed through any instruction
- Register Array
  - Temporary registers W and Z are used to hold 8- bit data during execution of some instructions
  - Not available for programmer

*remaining registers will be discussed in upcoming slides

8085 Registers



Registers

- Six general purpose 8-bit registers: B, C, D, E, H, L
- They can also be combined as register pairs to perform 16-bit operations: BC, DE, HL
- Registers are programmable (data load, move, etc.)

Accumulator

- Single 8-bit register that is part of the ALU
- Used for arithmetic / logic operations – the result is always stored in the accumulator.

- Flag Bits
- Indicate the result of condition tests
- Carry, Zero, Sign, Parity, etc.
- Conditional operations (IF / THEN) are executed based on the condition of these flag bits.

Program Counter (PC)

- Contains the memory address (16 bits) of the instruction that will be executed in the next step.

Stack Pointer (SP)

- Contains the address (16 bits) of a memory location in R/W memory, called the stack

Instruction Format

- Is a command to the microprocessor to perform a given task on a particular data
- Each instruction (instruction format) is of two parts-

➢ **opcode** (task to be performed, called the operation code)

➢ **operand** (the data to be operated on)

- The operands or data can be specified in different ways
- may include an 8-bit or 16-bit data, an internal Register, a memory location , or 8-bit or 16-bit address
- In some instructions, the operand is implicit.

- 8085 instruction set is classified according to byte size or word size
- Types of instruction format
  - Fixed length(same numbers of bits for all types of instructions)
  - Variable length(variety of instruction format of different length)
- 8085 uses 3 variety of variable length instruction format-
    1. One byte instruction
    2. Two byte instruction
    3. Three byte instruction

ONE-BYTE INSTRUCTIONS

A 1-byte instruction includes the opcode and the operand in the same byte. For example:

Task	Opcode	Operand*	Binary Code	Hex Code
Copy the contents of the accumulator in register C.	MOV	C,A	0100 1111	4FH
Add the contents of register B to the contents of the accumulator.	ADD	B	1000 0000	80H
Invert (complement) each bit in the accumulator.	CMA		0010 1111	2FH

TWO-BYTE INSTRUCTIONS

In a 2-byte instruction, the first byte specifies the operation code and the second byte specifies the operand. For example:

Task	Opcode	Operand	Binary Code	Hex Code	
Load an 8-bit data byte in the accumulator.	MVI	A,Data	0011 1110	3E	First Byte
			DATA	Data	Second Byte

THREE-BYTE INSTRUCTIONS

In a 3-byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16-bit address. Note that the second byte is the low-order address and the third byte is the high-order address. For example:

Task	Opcode	Operand	Binary Code	Hex Code	
Transfer the program sequence to the memory location 2085H.	JMP	2085H	1100 0011	C3*	First Byte
			1100 0011	85	Second Byte
			0010 0000	20	Third Byte

Instruction format

- The following clues can be used to recognize the number of bytes in an instruction of the 8085 microprocessor:

➢One-byte instruction: A mnemonic followed by a letter (or two letters) representing the registers (such as A, B, C, D, E, H, L, M, and SP) is a one-byte instruction. Instructions in which registers are implicit are also one-byte instructions.

Examples: (a) MOV A, B; (b) DCX SP: (c) RRC

➢Two-byte instruction: A mnemonic followed by 8-bit (byte) is a two-byte instruction.

Examples: (a) MVI A, 8-bit; (b) ADI 8-bit

➢Three-byte instruction: A mnemonic followed by 16-bit (also terms such as address or data) is a three-byte instruction.

Examples: (a) LXI B, 16-bit (data); (b) JNZ 16-bit (address);

(c) CALL 16-bit (address)

Data format

In 8-bit processor systems, commonly used codes and data formats are:

- **ASCII Code-** American Standard Code for Information Interchange

➢This is a 7-bit alphanumeric code that represents decimal numbers, English alphabets, and nonprintable characters.

➢Example Hexadecimal 30H to 39H represent 0 to 9 decimal digits

➢41H to 5AH represent capital A through Z

- **BCD Code—**BCD stands for binary-coded decimal

➢Used for decimal numbers

➢The decimal numbering system has ten digits, 0 to 9. Therefore, we need only four bits to represent ten digits from 0000 to 1001.

➢The remaining numbers, 1010 (A) to 1111 (F), are considered invalid.

- **Signed Integer-**

➢Signed integer is either a positive number or a negative number

➢In an 8-bit processor, the most significant digit D_7 is used for the sign; 0 represents the positive sign and 1 represents the negative sign.

➢The remaining seven bits. D_6-D_o, represent the magnitude of an integer. Therefore, the largest positive integer that can be processed at one time is 7FH and remaining Hex numbers 80H to FFH are considered negative number

- **Unsigned Integers-**

➢An integer without a sign can be represented by all the 8 bits in a microprocessor register

➢Therefore, the largest number that can be processed at one time is FFH. However, this does not imply that the 8085 microprocessor is limited to handling only 8-bit numbers. Numbers larger than 8 bits (such as 16-bit or 24-bit numbers) are processed by dividing them in groups of 8 bits

Addressing Modes

- To perform any operation, we have to give the corresponding instructions to the microprocessor.
- In each instruction, programmer has to specify 3 things:
  - Operation to be performed i.e. Opcode
  - Address of source of data i.e. Operands
  - Address of destination of result.
- The various formats for specifying the operands in an instruction are called addressing modes.
  - The operands may be source only, destination only or both of them.
  - The source operands can be immediate value, registers, input port, or a memory location.
  - The destination operands can be registers, input port, or a memory location.

Addressing Modes

- Microprocessor uses addressing mode techniques for the purpose of accommodating one or both of the following provisions.
  - To give programming versatility to the user by providing such facilities.
  - To reduce the number of bits in the addressing field of the instruction.
- Following are the five addressing modes supported by 8085 microprocessor
  i. Implied addressing mode
  ii. Immediate addressing mode
  iii. Register direct addressing mode
  iv. Register indirect addressing modes
  v. Direct (absolute) addressing mode

Addressing Modes

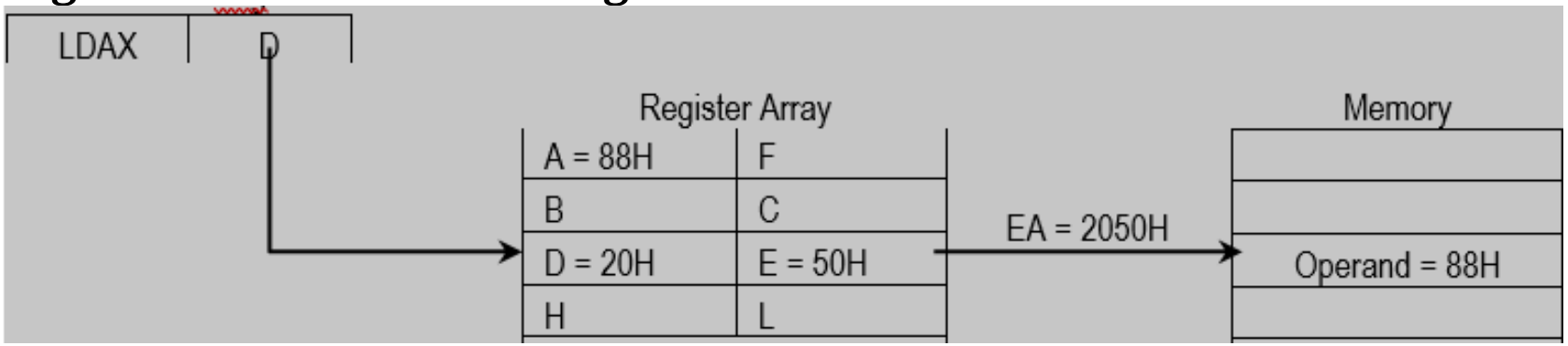
- **Implied Addressing Mode**
  - In this addressing mode, the operands are specified implicitly in the definition of an instruction.
    - E.g. CMA, RLC, SIM, STC etc.
- **Immediate Addressing Mode**
  - In this addressing mode, the operand is specified immediately after mnemonic in the instruction itself
  - The immediate mode instructions are mostly useful for initializing the registers to a constant value
    - E.g. MVI A, 29H; A <- 29H
      LXI B, C010H; B <- C0H, C <- 10H

Addressing Modes

- **Register Direct Addressing Mode**
  - In this addressing mode, the operands are the contents of registers itself given in the instruction
    - E.g. MOV H,C; H <- C
      INR A; A <- A+1
- **Register Indirect Addressing Mode**
  - In this addressing mode, the register specified in the register field contains the address of operand rather than the operand itself.
  - That means, the required operand is present in the memory location pointed by a value (address) present in the register field
    - E.g. LDAX D; A <- [DE]
      STAX B; [BC] <- A

Addressing Modes

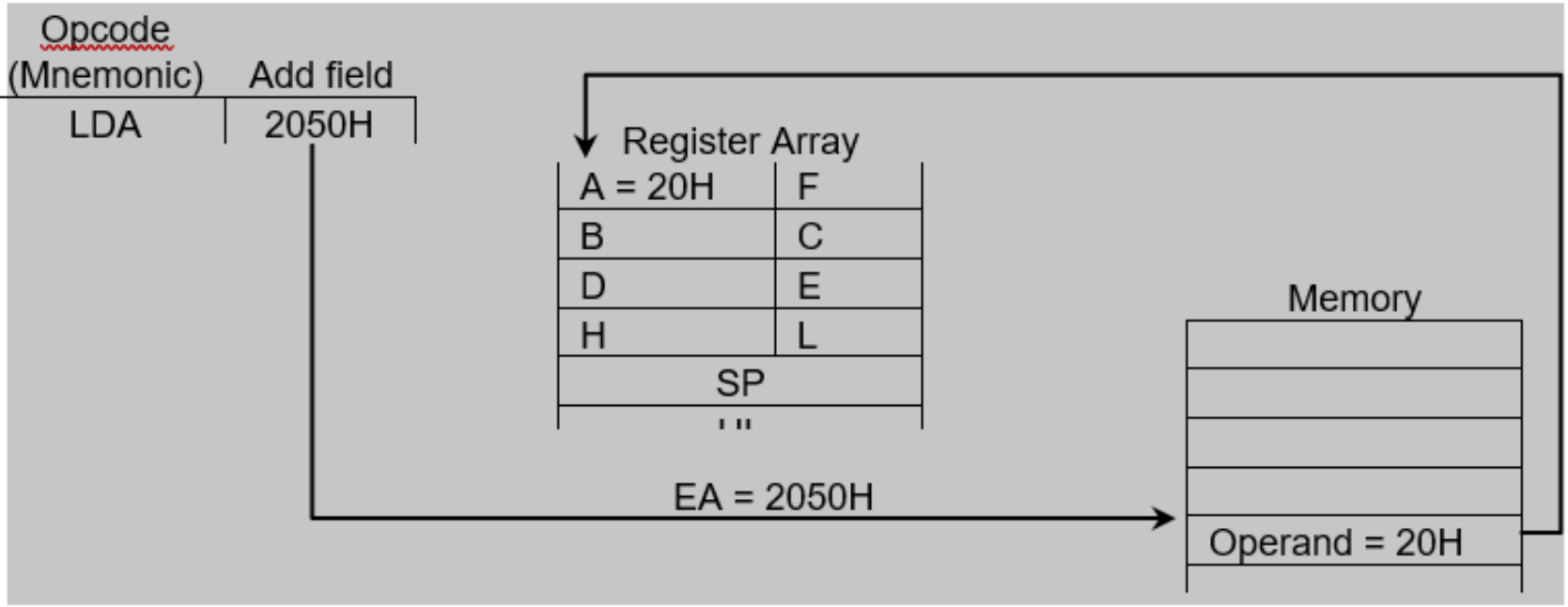
- **Register Indirect Addressing Mode**



- **Direct (Absolute) Address Mode**
  - In this addressing mode, address of operand is specified in instruction itself.
    - E.g. LDA 2050H; A <- [2050]
      STA B040H; [B040] <- A

Addressing Modes

- **Direct (Absolute) Address Mode**



Instruction set

- An instruction is a binary pattern designed inside a microprocessor to perform a specific function
- The entire group of instructions that a microprocessor supports is called Instruction Set
- 8085 has 246 instructions
- Each instruction is represented by an 8-bit binary value
- These 8-bits of binary value is called Op-Code or Instruction Byte.

Classification of Instruction Set

- Data Transfer Instruction
- Arithmetic Instructions
- Logical Instructions
- Branching Instructions
- Control Instructions
- Miscellaneous instructions

Data Transfer Instructions

- These instructions move data between registers, or between memory and registers
- These instructions copy data from source to destination
- While copying, the contents of source are not modified
- It does not affect the flags

Data Transfer Instructions

- **Load register with immediate value**

Opcode	Operand
MVI	Rd,8-bit data

Example:

MVI B,90H

Before Execution

A		F	
B		C	
D		E	
H		L	

After Execution

A		F	
B	90H	C	
D		E	
H		L	

Data Transfer Instructions

- **Load register pair with immediate value**

Opcode	Operand
LXI	Reg-pair,16-bit data

- Example:

  LXI H,2050H(H<-20, L<-50)

  LXI B,C020H(B<-C0, C<-20)

Data Transfer Instructions

- Before execution

A		F	
B		C	
D		E	
H		L	

LXI H,2050H

After execution

A		F	
B		C	
D		E	
H	20	L	50

2050H

52H

M=52H

A		F	
B		C	
D		E	
H		L	

LXI B,C020H

A		F	
B	C0	C	20
D		E	
H		L	

('M' represents memory whose address is pointed by content of HL)

Data Transfer Instructions

- **Load memory with immediate value**

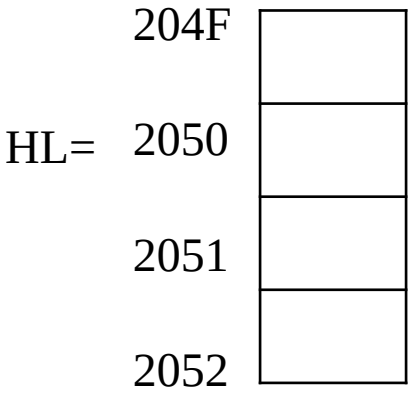
Opcode	Operand
MVI	M,8-bit data

- Example:

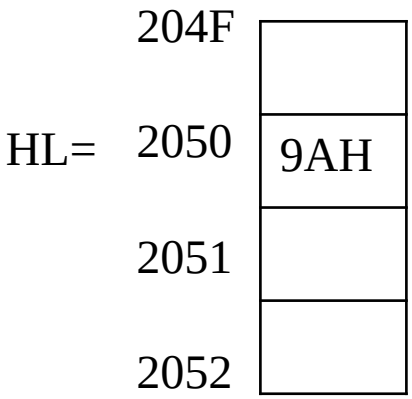
LXI H,2050H; H ←20H, L ←50H

MVI M,9AH; [HL<-9A]

Before Execution



After Execution



('M' represents memory whose address is pointed by content of HL)

BEFORE EXECUTION

A		F	
B		C	
D		E	
H		L	

MVI E,68H

AFTER EXECUTION

A		F	
B		C	
D		E	68H
H		L	

BEFORE EXECUTION

204F		
2050		
2051		
HL= 2052		

MVI M,68H

AFTER EXECUTION

204F		
2050		
2051		
HL= 2052		68H

Data Transfer Instructions

- **Between register to register**

OPcode	Operand
MOV	Rd, Rs

- Example:

  MOV A,B ;A ⟵ B

  MOV C,D ; C ⟵ D

  MOV H,A ;H ⟵ A

Question

1. Write a program in 8085 to load register A with 85H, D with 4AH, register B with C5H and E with 7FH .Then move the value of register E to register C.

MVI A,85H

LXI D, 4A7FH

MVI B , C5H

MOV C, E

2. Write the program to load value 8AH in memory with address 8090H.

LXI H, 8090H

MVI M, 8AH

Data Transfer Instructions

- **Between Accumulator and memory (direct)**

Opcode	Operand
LDA	16-Bit address; A<-[Address]
STA	16-Bit address; [Address]<-A

- Example:

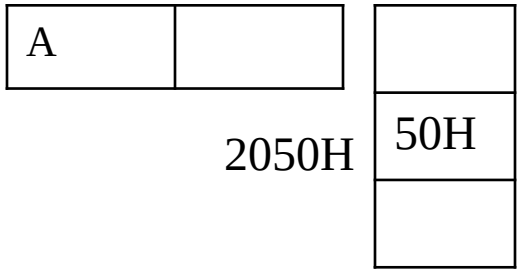
  LDA 2050; A<- [2050]

  STA C090;[C090]<- A

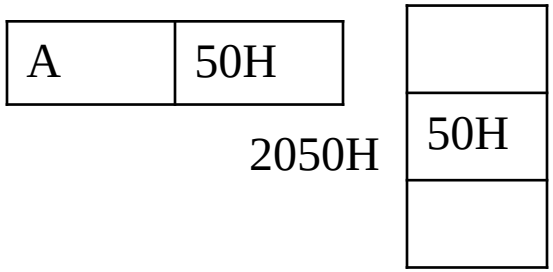
Data Transfer Instructions

- **LDA 2050H**

Before Execution

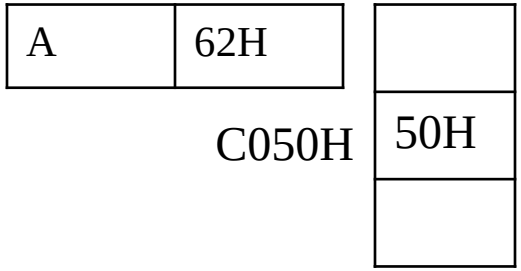


After Execution

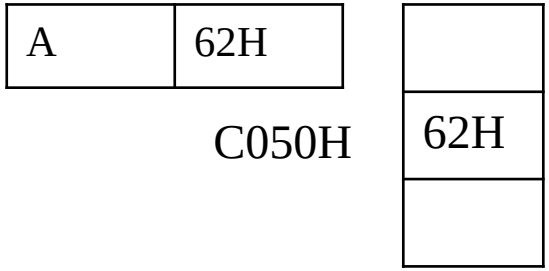


- **STA C050H**

Before Execution



After Execution



Data Transfer Instructions

- **Between Accumulator and memory (indirect)**

Opcode	Operand
LDAX	Reg-pair
STAX	Reg-pair

- Example:

LDAX B; A<-[BC] STAX B;[BC]<-A

LDAX D; A<-[DE] STAX D;[DE]<-A

❖ Note:

LDAX H

STAX H

Is not possible

Data Transfer Instructions

- Example:

  LXI D 3050H; D ← 30H, E ← 50H

  LDAX D; A ← [DE]

Before Execution

A		F	
B		C	
D	30	E	50

3050H	75H

After Execution

A	75H	F	
B		C	
D	30	E	50

3050H	75H

- Example: LXI B 5080H; B ← 50H, C ← 80H

  MVI A,5AH; A ← 5AH

  STAX B; [BC] ← A

Before Execution

B	50	C	80
A=5AH			

After Execution

5080H	5AH

Data Transfer Instructions

- **Memory to other registers and vice-versa**
- Example:

MOV B,M; B<-[HL]	MOV M,B; [HL]<-B
MOV C,M; C<-[HL]	MOV M,C; [HL]<-C
MOV D,M; D<-[HL]	MOV M,D; [HL]<-D
MOV E,M; E<-[HL]	MOV M,E; [HL]<-E

Data Transfer Instructions

- Example:

  LXI H,2050H; H ← 20H, L ← 50H

  MOV A,M; A ← [HL]

Before Execution

A		F	
B		C	
H	20	L	50

75H

2050H

MOV A,M

After execution

A	75H	F	
B		C	
H	20	L	50

75H

2050H

Data Transfer Instructions

- Example:

  MVI A, 98H; A←98H

  LXI H, C052H; H←C0H, L← 52H

  MOV M,A; [HL]← A

Before Execution

A	98H	F	
B		C	
H	C0	L	52

C052H

75H

MOV M,A

After execution

A	98H	F	
B		C	
H	C0	L	52

C052H

98H

FUNDAMENTAL OF MICROPROCESSOR

UNIT-1

Prepared by: Laxmi KC ,Lecturer
Advanced college of Engineering and Management, Department of Electronics and Computer

INTRODUCTION

- WHAT IS A MICROPROCESSOR?

A microprocessor is a multipurpose, programmable ,clock driven, register based electronic device that reads binary instructions from a storage device called memory, accepts binary data as inputs and processes data according to those instructions, and provide results as output.

- Instruction set: the set of instruction the microprocessor can execute
- Bandwidth: the number of bits processed in single instruction
- Clock speed: how many instructions per second the programmer can execute

- Programmable machine can represented by 4 components: microprocessor, input, output and memory
- These component work together to perform a given task

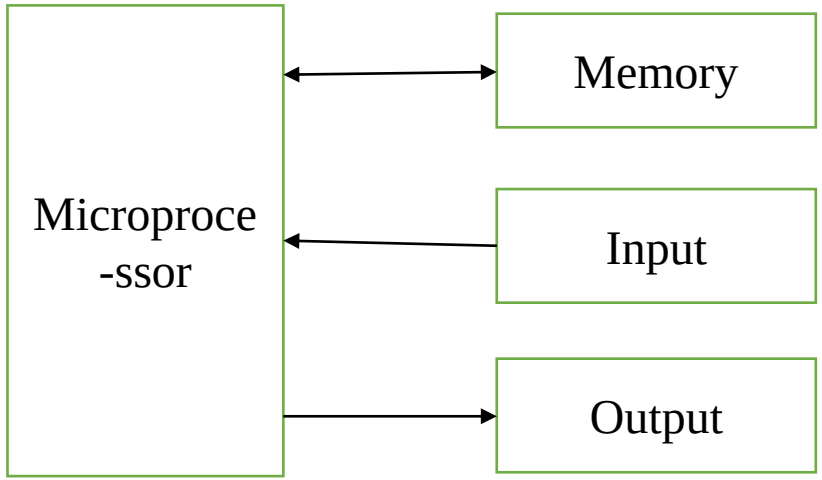


Figure: A programmable machine

- µp applications are classified in 2 categories:
  - Reprogrammable system
  - Embedded system
- Communicates and operates in binary numbers 0 and 1 called bits
- Each microprocessor has a fixed set of instruction in the form of binary patterns called a machine language
- Binary instruction are given abbreviated names called mnemonics which form the assembly language

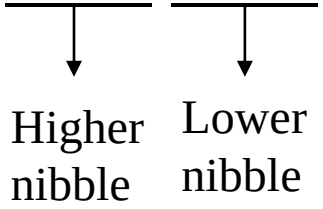
Example: MOV A,B

↓

Mnemonics

- µp recognises and processes a group of bits called word
- µp are classified according to their word length

- Processor with 8-bit word is known as an 8-bit microprocessor and those with 32- bit word is known as 32-bit microprocessor
- word length is also expressed in bytes
- 1 byte=8 bits
- 16 bit µp has word length of 2 bytes
- Group of 4 bits is called nibble
- Example: 0110 1101



History of microprocessor

The microprocessor became possible only after integrated circuit technology had advanced to the point where several thousands transistor switches could be integrated onto a single semi conductor chip.

- 1971 4-bit microprocessor Intel 4004 (2300 transistors,640bytes memory addressing capacity)
- 1972 8- bit intel 8008 microprocessor
- 1974 8- bit intel 8080 microprocessor
- Motorola 6800,Zilog Z80 Intel 8085 were developed as improvement over 8080

- Motorola 6800 was designed with different architecture and instruction set whereas Zilog Z80 and Intel 8085 were upward software compatible
- Most microcomputers are built with 32- bit and 64- bit microprocessor

TABLE 1.1
Intel Microprocessors: Historical Perspective

Processor	Year of Introduction	Number of Transistors	Initial Clock Speed	Address Bus	Data Bus	Addressable Memory
4004	1971	2,300	108 kHz	10-bit	4-bit	640 bytes
8008	1972	3,500	200 kHz	14-bit	8-bit	16 K
8080	1974	6,000	2 MHz	16-bit	8-bit	64 K
8085	1976	6,500	5 MHz	16-bit	8-bit	64 K
8086	1978	29,000	5 MHz	20-bit	16-bit	1 M
8088	1979	29,000	5 MHz	20-bit	8-bit*	1 M
80286	1982	134,000	8 MHz	24-bit	16-bit	16 M
80386	1985	275,000	16 MHz	32-bit	32-bit	4 G
80486	1989	1.2 M	25 MHz	32-bit	32-bit	4 G
Pentium	1993	3.1 M	60 MHz	32-bit	32/64-bit	4 G
Pentium Pro	1995	5.5 M	150 MHz	36-bit	32/64-bit	64 G
Pentium II	1997	8.8 M	233 MHz	36-bit	64-bit	64 G
Pentium III	1999	9.5 M	650 MHz	36-bit	64-bit	64 G
Pentium 4	2000	42 M	1.4 GHz	36-bit	64-bit	64 G

*External 8-bit and internal 16-bit data bus

Basic block diagram of a computer

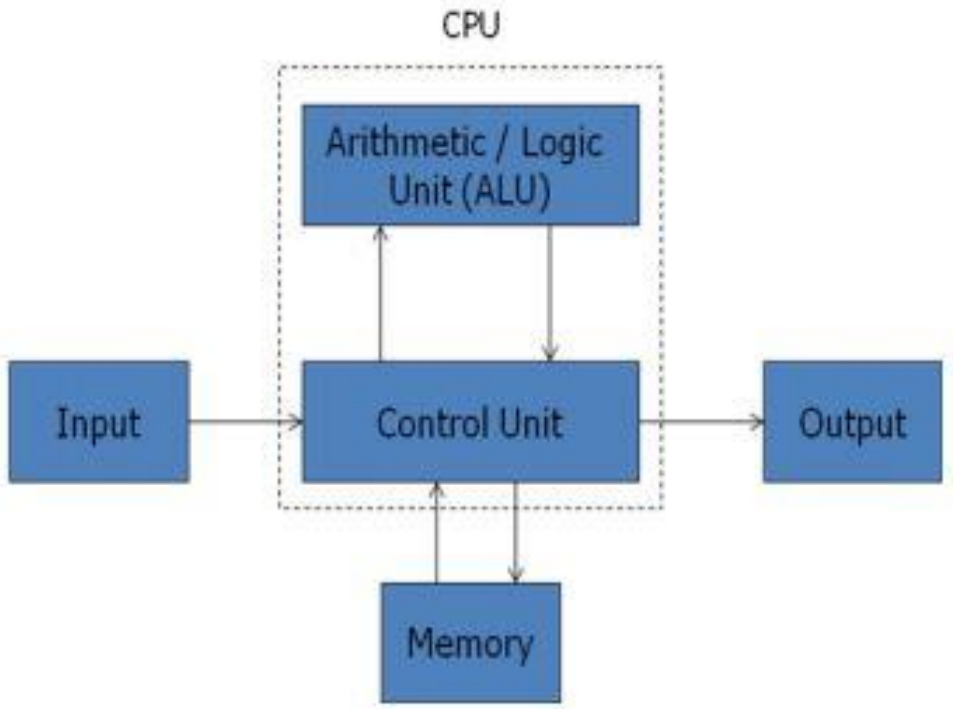


Figure: Traditional block diagram

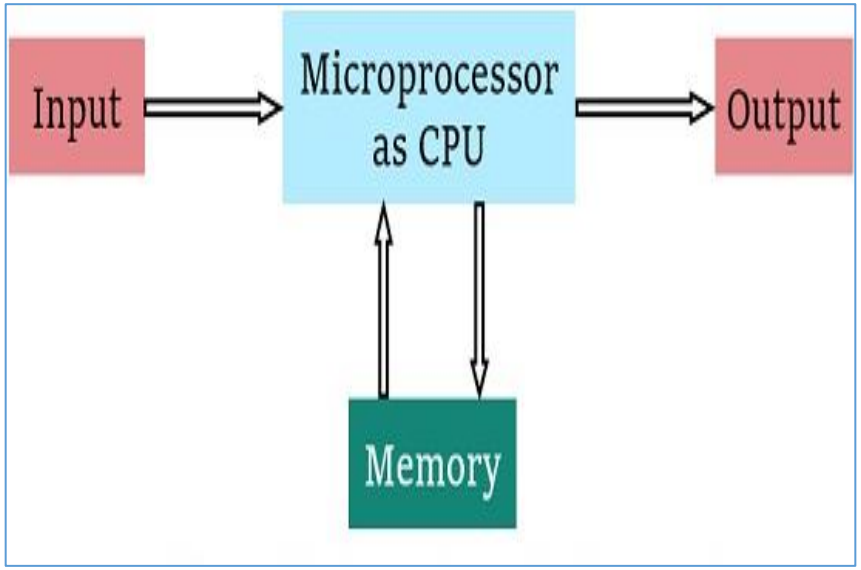


Figure: Block diagram of computer with microprocessor as CPU

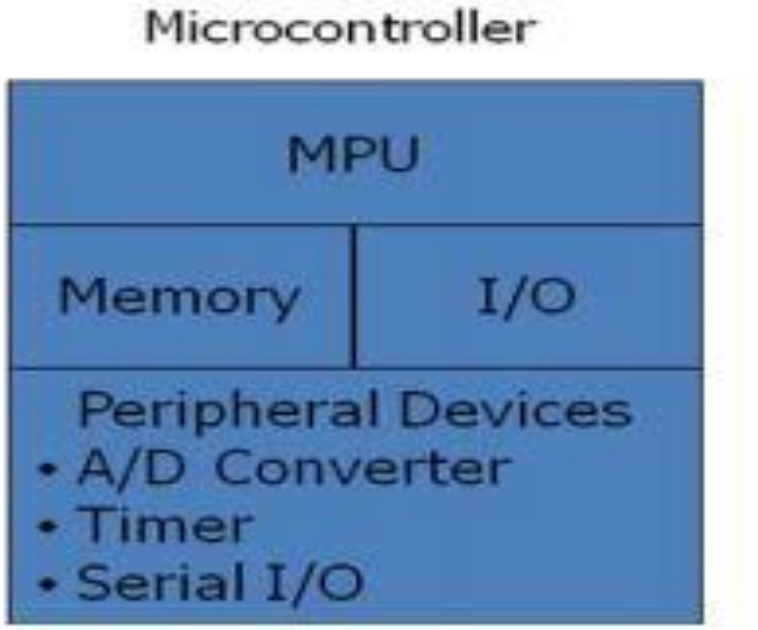


Figure: Block diagram of microcontroller

Organisation of microprocessor based system

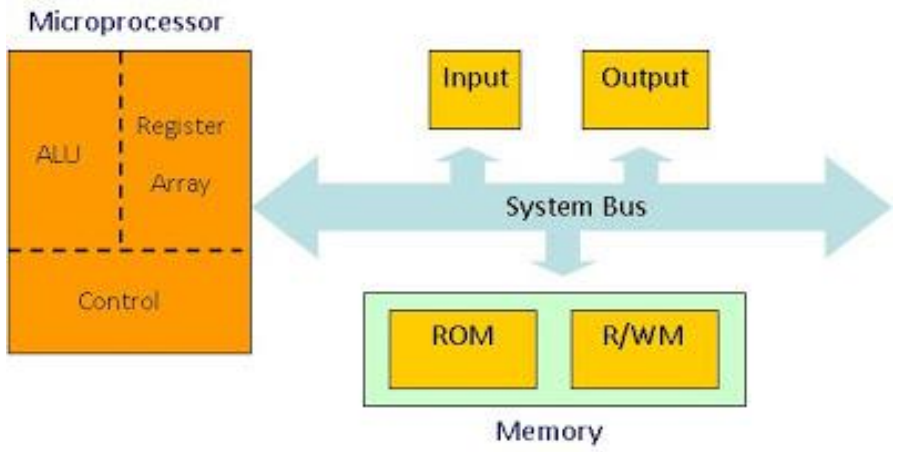


Figure: Block diagram of Microprocessor based system

- The function of various components of a microprocessor-based system can be summarized as follows:

➢ The microprocessor
  - reads instructions from memory
  - Communicates with all peripherals (memory and I/O) using the system bus
  - Controls the timing of information flow
  - Performs the computing tasks specified in a program

➢ The memory
  - Stores binary information, called instructions and data

- provides the instructions and data to the microprocessor on request
- stores results and data for the microprocessor

➢ The input device

- enters data and instructions under the control of a program such as a monitor program

➢ The output device

- accepts data from the microprocessor as specified in a program

➢ The bus

- carries bits between the microprocessor and memory and I/Os

Bus organisation

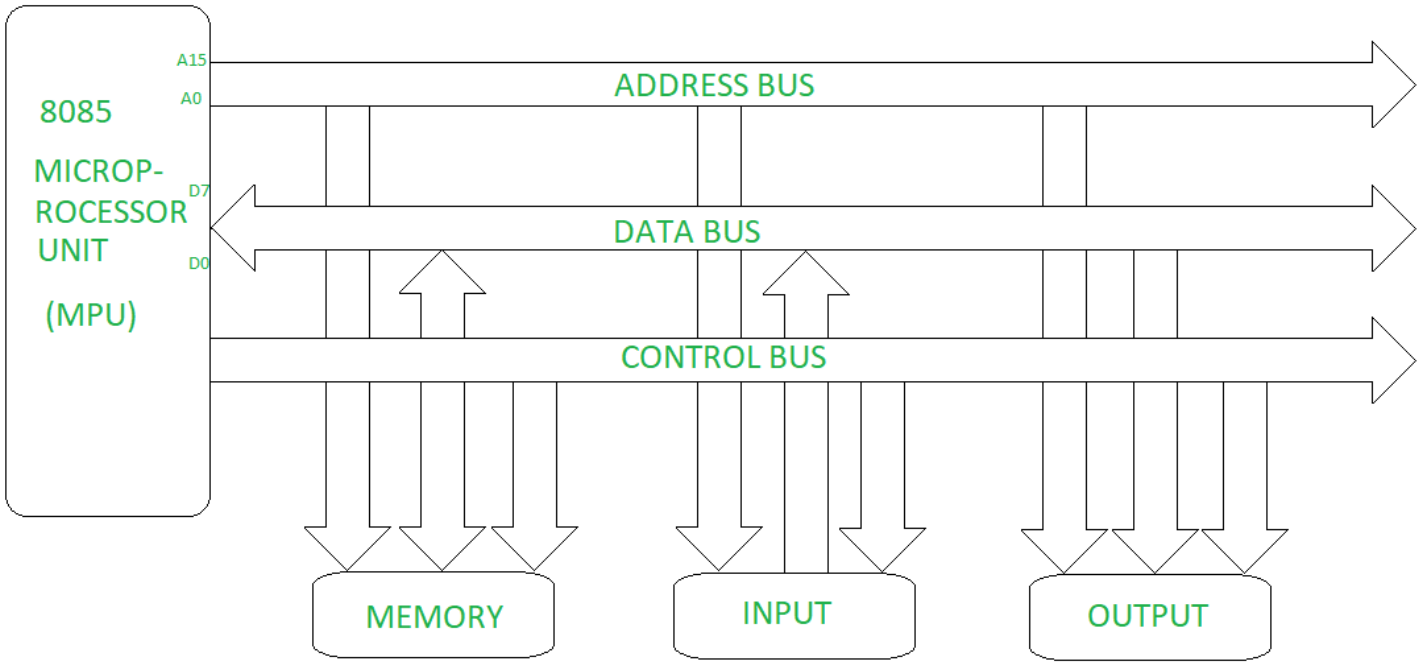


Figure: Bus organisation system of 8085 microprocessor

- System bus comprises of three buses -

1. Address bus(Unidirectional):
   - Carries address of memory in which the processor wishes to read or write
   - The number of bits of address bus determines the maximum size of memory that processor access
2. Data bus(Bidirectional):
   - Connection between and within CPU, memory and peripherals which carry data
3. Control bus(unidirectional):
   - Set of wires that carry different control signals
   - Used to identify a device type with which microprocessor intends to communicate

Instruction Cycle(Processing Cycle)

- An instruction cycle is the time period during which a computer processes a machine language instruction from its memory or the sequence of actions that the central processing unit (CPU) performs to execute each machine code instruction in a program
- Also called fetch-and-execute cycle
- The instruction must be fetched from main memory, and then executed by the CPU.

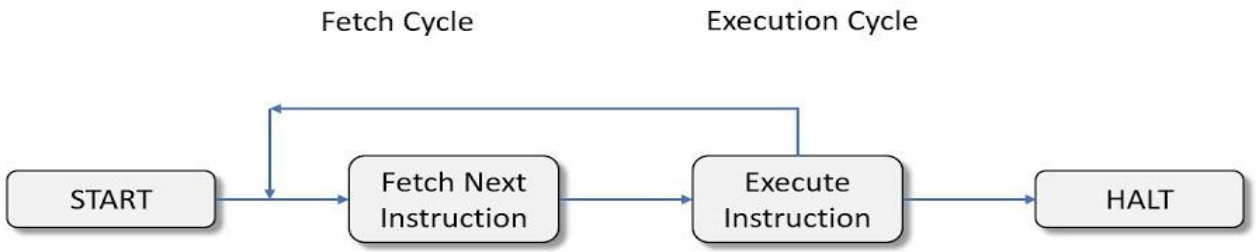
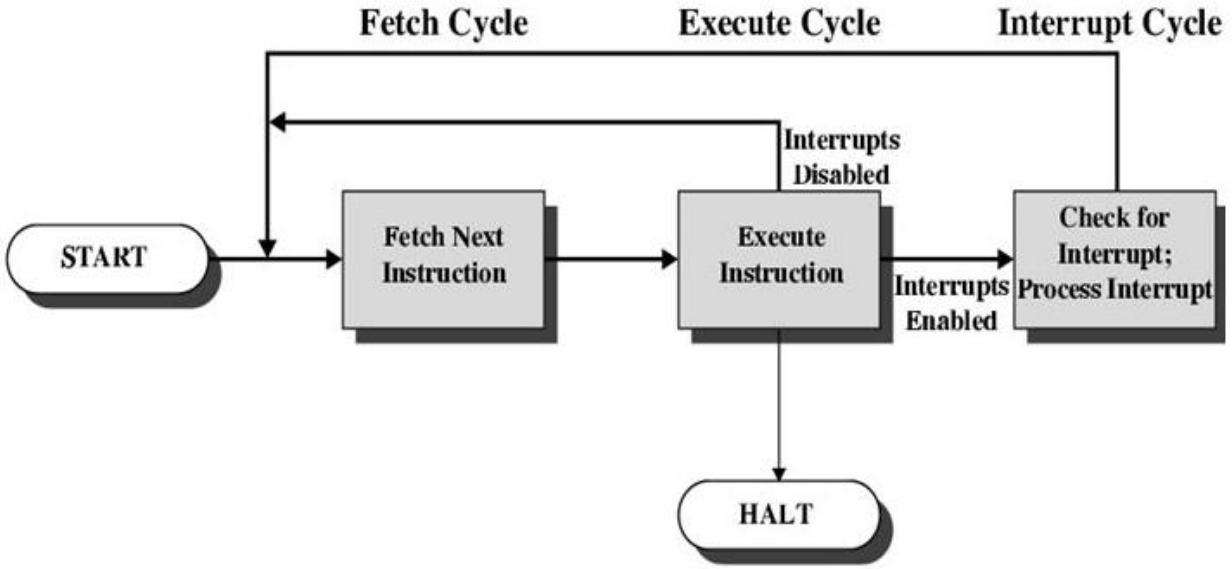
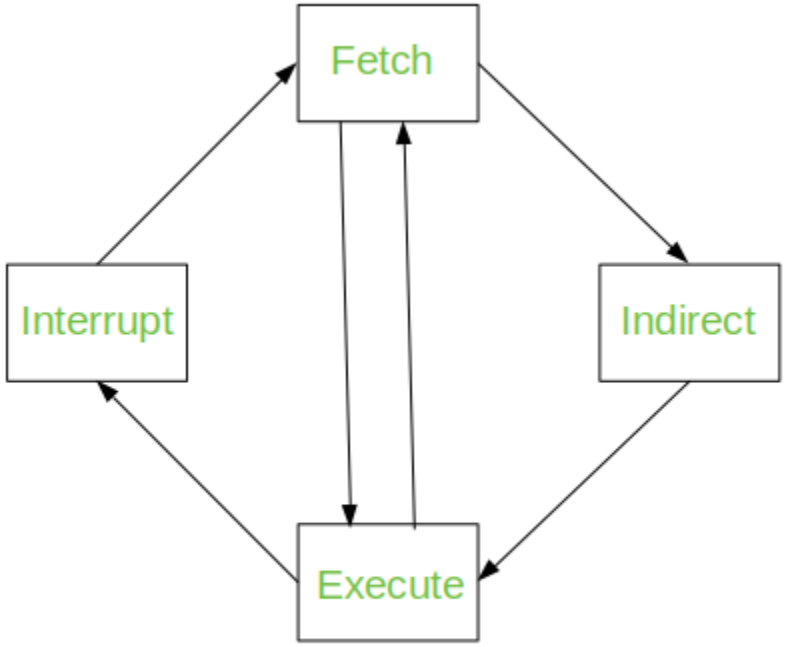


Figure: Computer Instruction Cycle

- In some cases machine (program) encounter a problem which may halt the system known as interrupt.
- To solve this interrupt CPU needs another sub-cycle called interrupt cycle



- If indirect addressing is used then additional memory access is required to fetch the indirect address called the indirect cycle



The Instruction Cycle