C.S.E → 5th Sem 34

UNIT-III

Central Processing Unit: General register organization, stack organization, instruction formats (Zero, One, Two and Three Address Instruction), addressing modes, Data transfer and manipulation, Program control. CISC and RISC: features and comparison. Pipeline and vector Processing, Parallel Processing, Flynn's taxonomy, Pipelining, Instruction Pipeline, Basics of vector processing and Array Processors.

GENERAL REGISTER ORGANIZATION

When we are using multiple general purpose registers, instead of single accumulator register, in the CPU Organization then this type of organization is known as General register based CPU Organization. In this type of organization, computer uses two or three address fields in their instruction format. Each address field may specify a general register or a memory word. If many CPU registers are available for heavily used variables and intermediate results, we can avoid memory references much of the time, thus vastly increasing program execution speed, and reducing program size.

For example: MULT R1, R2, R3

This is an instruction of an arithmetic multiplication written in assembly language. It uses three address fields R1, R2 and R3. The meaning of this instruction is:

R1 <-- R2 * R3

This instruction also can be written using only two address fields as:

MULT R1, R2

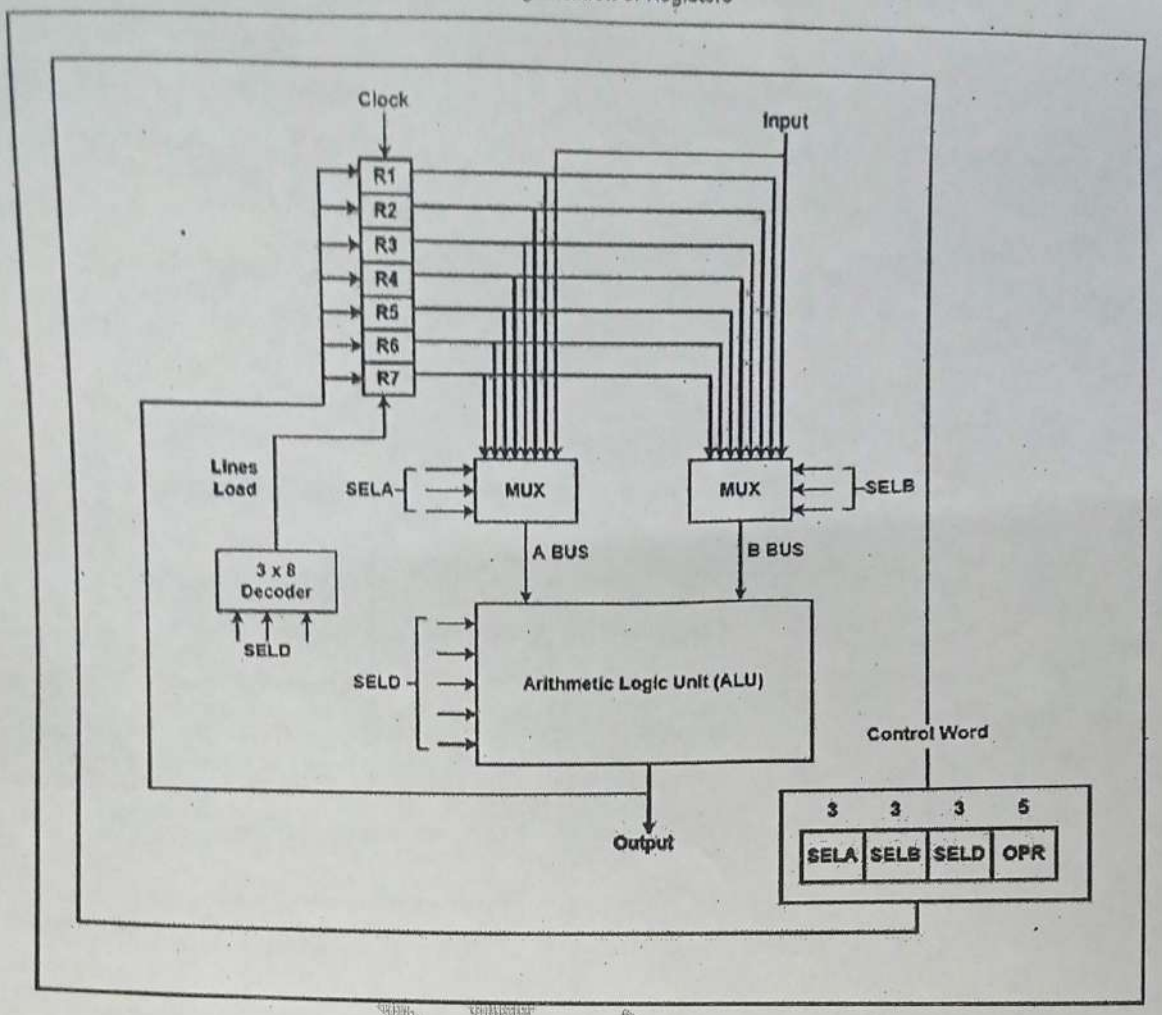
In this instruction, the destination register is the same as one of the source registers. This means the operation

R1 <-- R1 * R2

The use of large number of registers results in short program with limited instructions.

If a CPU includes some registers, therefore a common bus can link these registers. A general organization of seven CPU registers is displayed in the figure.

General Organization of Registers



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.

Consider R1 ← R2 + R3, the following are the functions implemented within the CPU –.

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 transfer 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.

The encoding of register selection fields is specified in the table.

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

Some examples of General register based CPU Organization are **IBM 360 and PDP- 11.**

The advantages of General register based CPU organization –

- Efficiency of CPU increases as there are large number of registers are used in this organization.
- Less memory space is used to store the program since the instructions are written in compact way.

The disadvantages of General register based CPU organization –

- Care should be taken to avoid unnecessary usage of registers. Thus, compilers need to be more intelligent in this aspect.
- Since large number of registers are used, thus extra cost is required in this organization.

General register CPU organization of two type:

1. Register-memory reference architecture (CPU with less register)– In this organization Source 1 is always required in register, source 2 can be present either in register or in memory. Here two address instruction format is the compatible instruction format.
2. Register-register reference architecture (CPU with more register)– In this organization ALU operations are performed only on a register data. So operands are required in the register. After manipulation result is also placed in register. Here three address instruction format is the compatible instruction format.

STACK ORGANIZATION

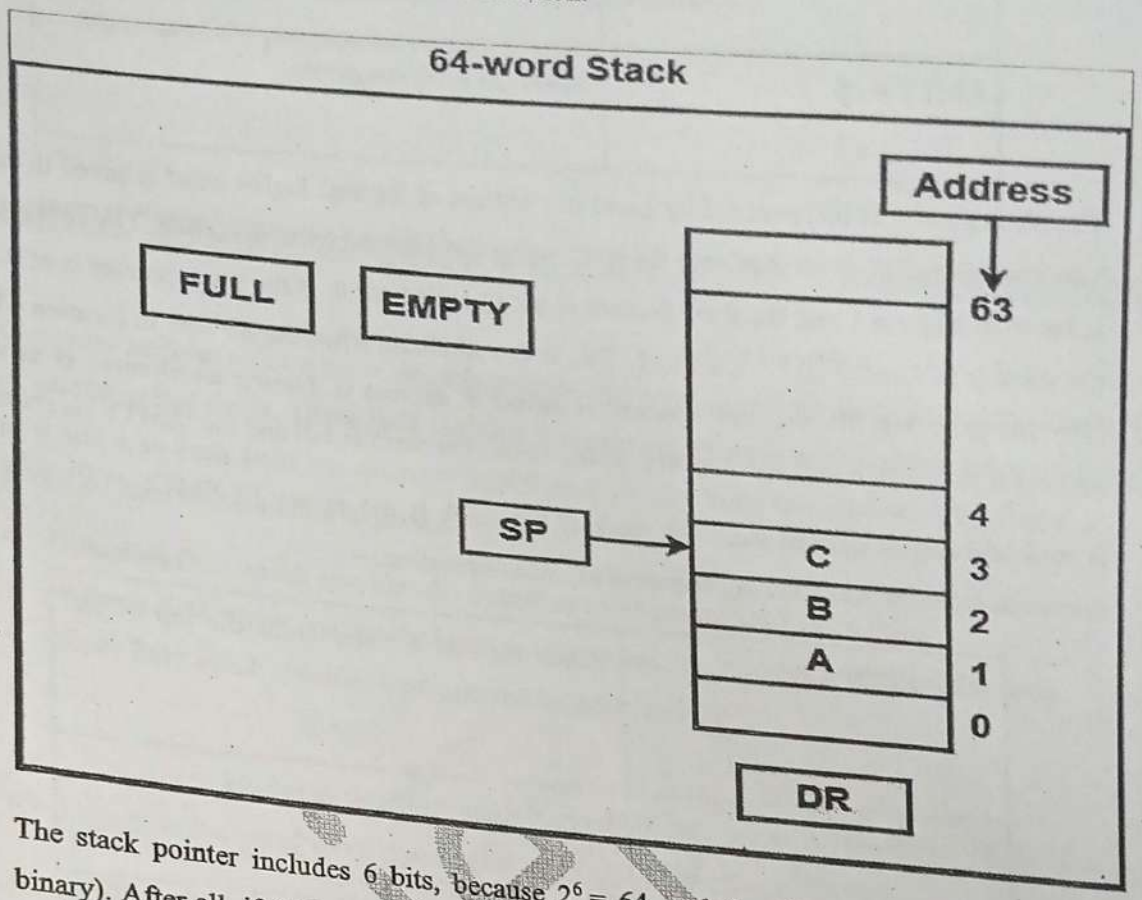
The computers which use Stack-based CPU Organization are based on a data structure called **stack**. The stack is a list of data words. It uses **Last In First Out (LIFO)** access method which is the most popular access method in most of the CPU. A register is used to store the address of the topmost element of the stack which is known as **Stack pointer (SP)**. In this organization, ALU operations are performed on stack data. It means both the operands are always required on the stack. After manipulation, the result is placed in the stack.
The main two operations that are performed on the operators of the stack are **Push** and **Pop**. These two operations are performed from one end only.

Register Stack

The stack can be arranged as a set of memory words or registers. Consider a 64-word register stack arranged as displayed in the figure. The stack pointer register includes a binary number, which is the address of the element present at the top of the stack. The three-element A, B, and C are located in the stack.

The element C is at the top of the stack and the stack pointer holds the address of C that is 3. The top element is popped from the stack through reading memory word at address 3 and decrementing the stack pointer by 1. Then, B is at the top of the stack and the SP holds the address

of B that is 2. It can insert a new word, the stack is pushed by **incrementing** the stack pointer by 1 and inserting a word in that incremented location.



The stack pointer includes 6 bits, because $2^6 = 64$, and the SP cannot exceed 63 (111111 in binary). After all, if 63 is incremented by 1, therefore the result is 0(111111 + 1 = 1000000). SP holds only the six least significant bits. If 000000 is decremented by 1 thus the result is 111111.

Therefore, when the stack is full, the one-bit register 'FULL' is set to 1. If the stack is null, then the one-bit register 'EMTY' is set to 1. The data register DR holds the binary information which is composed into or readout of the stack.

First, the SP is set to 0, EMTY is set to 1, and FULL is set to 0. Now, as the stack is not full (FULL = 0), a new element is inserted using the push operation.

The push operation is executed as follows –

SP←SP + 1	It can increment stack pointer
K[SP] ← DR	It can write element on top of the stack

If (SP = 0) then (FULL ← 1)	Check if stack is full
EMTY ← 0	Mark the stack not empty

The stack pointer is incremented by 1 and the address of the next higher word is saved in the SP. The word from DR is inserted into the stack using the memory write operation. The first element is saved at address 1 and the final element is saved at address 0. If the stack pointer is at 0, then the stack is full and 'FULL' is set to 1. This is the condition when the SP was in location 63 and after incrementing SP, the final element is saved at address 0. During an element is saved at address 0, there are no emptier registers in the stack. The stack is full and the 'EMTY' is set to 0.

A new element is deleted from the stack if the stack is not empty (if EMTY = 0). The pop operation includes the following sequence of micro-operations –

DR←K[SP]	It can read an element from the top of the stack
SP ← SP – 1	It can decrement the stack pointer
If (SP = 0) then (EMTY ← 1)	Check if stack is empty
FULL ← 0	Mark the stack not full

The top element from the stack is read and transfer to DR and thus the stack pointer is decremented. If the stack pointer reaches 0, then the stack is empty and 'EMTY' is set to 1. This is the condition when the element in location 1 is read out and the SP is decremented by 1.

The advantages of Stack based CPU organization –

- Efficient computation of complex arithmetic expressions.
- Execution of instructions is fast because operand data are stored in consecutive memory locations.

- Length of instruction is short as they do not have address field.

The disadvantages of Stack based CPU organization –

- The size of the program increases.

INSTRUCTION FORMATS (Zero, One, Two and Three Address Instruction)

Computer perform task on the basis of instruction provided. An instruction in computer comprises of groups called fields. These field contains different information as for computers everything is in 0 and 1 so each field has different significance on the basis of which a CPU decide what to perform. 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:

1. **Single Accumulator organization**
2. **General register organization**
3. **Stack organization**

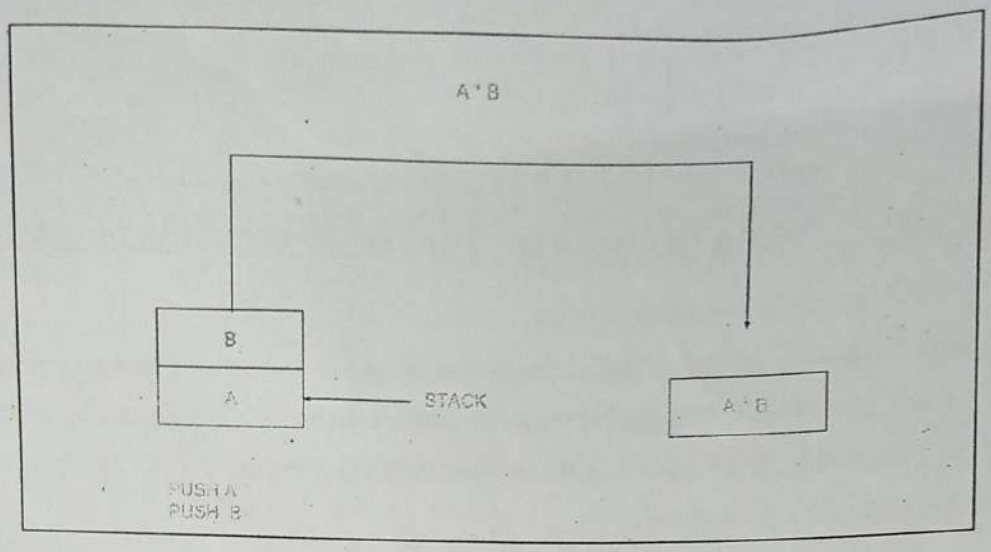
In first organization operation is done involving a special register called accumulator. In second on multiple registers are used for the computation purpose. In third organization the work on stack basis operation due to which it does not contain any address field. It is not necessary that only a single organization is applied a blend of various organization is mostly what we see generally.

On the basis of number of address, instruction is classified as:

Note that we will use X = (A+B) *(C+D) expression to showcase the procedure.

1. **Zero Address Instructions –**

A stack based computer do not use address field in instruction. To evaluate an expression first it is converted to revere Polish Notation i.e. Post fix Notation.



Expression: X = (A+B) *(C+D)

Post fixed: X = AB+CD+*

TOP means top of stack

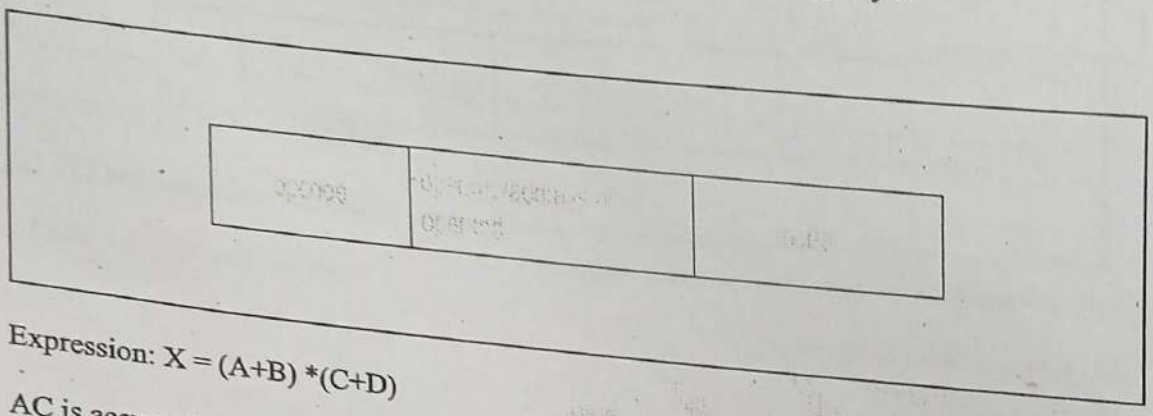
M[X] is any memory location

PUSH	A	TOP = A
PUSH	B	TOP = B
ADD		TOP = A+B
PUSH	C	TOP = C
PUSH	D	TOP = D
ADD		TOP = C+D
MUL		TOP = (C+D)*(A+B)

POP	X	M[X] = TOP

2. **One Address Instructions –**

This use a implied ACCUMULATOR register for data manipulation. One operand is in accumulator and other is in register or memory location. Implied means that the CPU already know that one operand is in accumulator so there is no need to specify it.



Expression: X = (A+B) *(C+D)

AC is accumulator

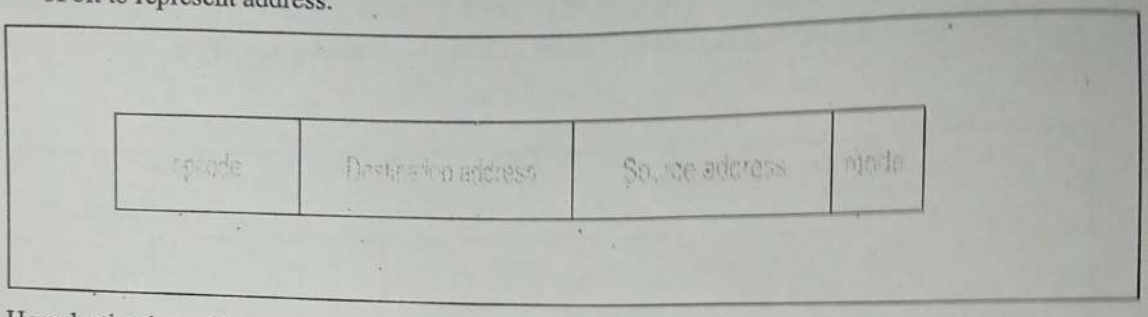
M [] is any memory location

M[T] is temporary location

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

3. **Two Address Instructions –**

This is common in commercial computers. Here two address can be specified in the instruction. Unlike earlier in one address instruction the result was stored in accumulator here result can be stored at different location rather than just accumulator, but require more number of bit to represent address.



Here destination address can also contain operand.

Expression: X = (A+B) *(C+D)

R1, R2 are registers

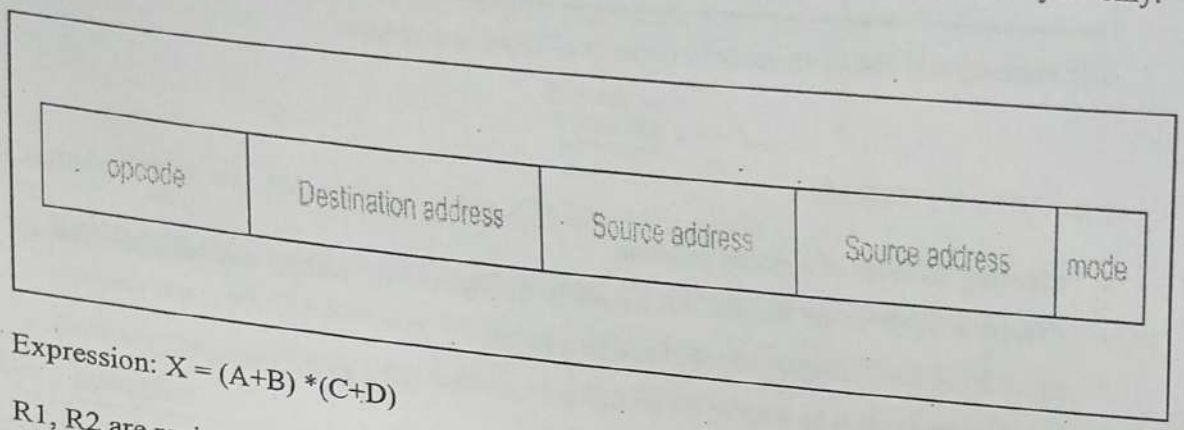
M [] is any memory location

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

4. **Three Address Instructions –**

This has three address field to specify a register or a memory location. Program created are much short in size but number of bits per instruction increase. These instructions make creation of program much easier but it does not mean that program will run much faster

because now instruction only contain more information but each micro operation (changing content of register, loading address in address bus etc.) will be performed in one cycle only.



Expression: X = (A+B) *(C+D)

R1, R2 are registers

M [] is any memory location

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

ADDRESSING MODES

The term addressing modes refers to the way in which the operand of an instruction is specified. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually executed.

Addressing modes for 8086 instructions are divided into two categories:

1) Addressing modes for data

2) Addressing modes for branch

The 8086 memory addressing modes provide flexible access to memory, allowing you to easily access variables, arrays, records, pointers, and other complex data types. The key to good assembly language programming is the proper use of memory addressing modes.

An assembly language program instruction consists of two parts

Opcode	Operand

The memory address of an operand consists of two components:

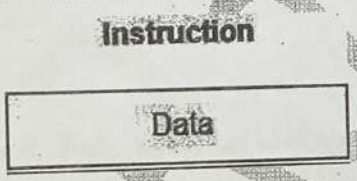
IMPORTANT TERMS

- **Starting address** of memory segment.
- **Effective address or Offset**: An offset is determined by adding any combination of three address elements: **displacement, base and index.**
- **Displacement:** It is an 8 bit or 16 bit immediate value given in the instruction.
- **Base**: Contents of base register, BX or BP.
- **Index**: Content of index register SI or DI.

According to different ways of specifying an operand by 8086 microprocessors, different addressing modes are used by 8086.

Addressing modes used by 8086 microprocessors are discussed below:

- **Implied mode:** In implied addressing the operand is specified in the instruction itself. In this mode the data is 8 bits or 16 bits long and data is the part of instruction. Zero address instruction are designed with implied addressing mode.

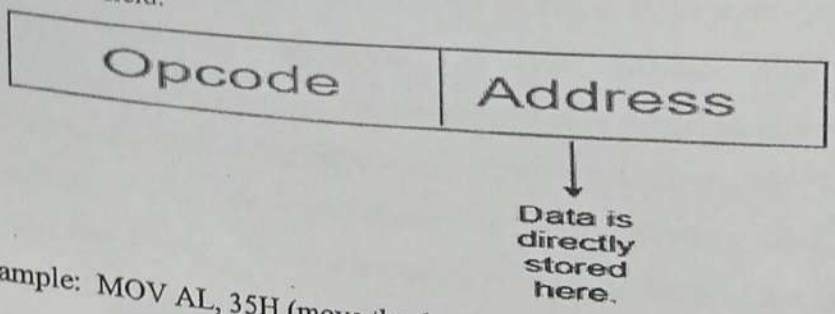


Example: CLC (used to reset Carry flag to 0)

- **Immediate addressing mode (symbol #):**In this mode data is present in address field of instruction .Designed like one address instruction format.

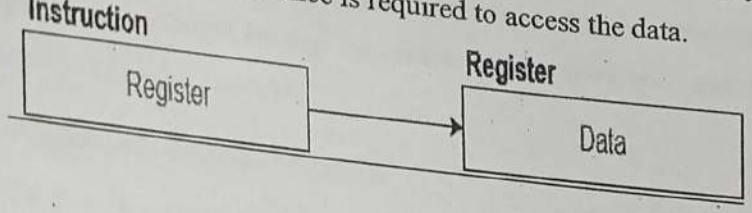
Note: Limitation in the immediate mode is that the range of constants are restricted by size of

address field.



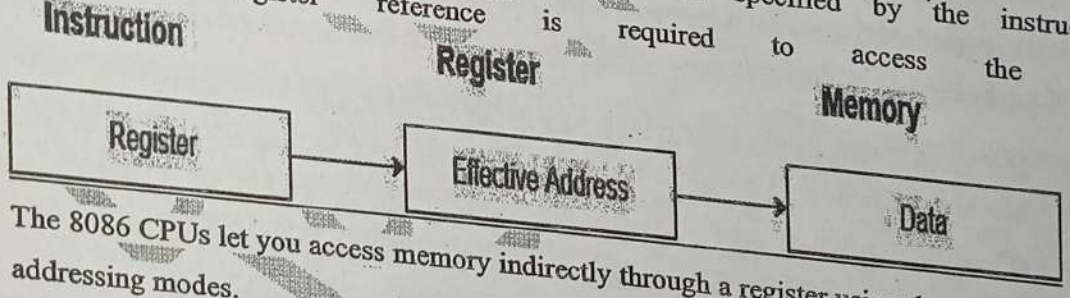
Example: MOV AL, 35H (move the data 35H into AL register)

- **Register mode:** In register addressing the operand is placed in one of 8 bit or 16-bit general purpose registers. The data is in the register that is specified by the instruction. Here one register reference is required to access the data.



Example: MOV AX, CX (move the contents of CX register to AX register)

- **Register Indirect mode:** In this addressing the operand's offset is placed in any one of the registers BX, BP, SI, DI as specified in the instruction. The effective address of the data is in the base register or an index register that is specified by the instruction. Here two register reference is required to access the data.



The 8086 CPUs let you access memory indirectly through a register using the register indirect addressing modes.

MOV AX, [BX](move the contents of memory location s

addressed by the register BX to the register AX)

- **Auto Indexed (increment mode):** Effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to point to the next consecutive memory location.(R1)+. Here one register reference, one memory reference and one ALU operation is required to

access the data.

Example:

Add R1, (R2) + // OR

R1 = R1 +M[R2] = R2 = R2 + d

- Useful for stepping through arrays in a loop. R2 – start of array d – size of an element
- **Auto indexed (decrement mode)**: Effective address of the operand is the contents of a register specified in the instruction. Before accessing the operand, the contents of this register are automatically decremented to point to the previous consecutive memory location. –**(R1)** Here one register reference, one memory reference and one ALU operation is required to access the data.

Example:

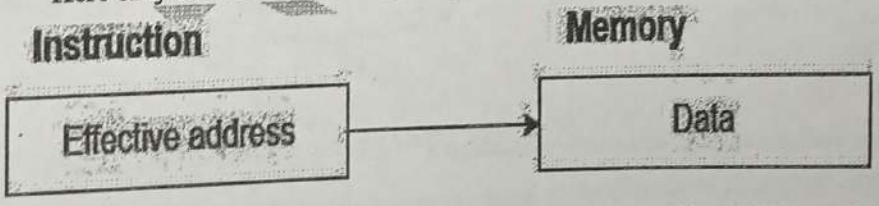
Add R1, -(R2) //OR

R2 = R2-d

R1 = R1 + M[R2]

Auto decrement mode is same as auto increment mode. Both can also be used to implement a stack as push and pop. Auto increment and Auto decrement modes are useful for implementing "Last-In-First-Out" data structures.

- **Direct addressing/ Absolute addressing Mode (symbol [ ]):** The operand's offset is given in the instruction as an 8 bit or 16 bit displacement element. In this addressing mode the 16 bit effective address of the data is the part of the instruction. Here only one memory reference operation is required to access the data.



Example: ADD AL, [0301] //add the contents of offset address 0301 to AL

- **Indirect addressing Mode (symbol @ or ()):** In this mode address field of instruction contains the address of effective address. Here two references are required.

  1st reference to get effective address.

  2nd reference to access the data.

Based on the availability of Effective address, Indirect mode is of two kinds:

1. Register Indirect: In this mode effective address is in the register, and corresponding register name will be maintained in the address field of an instruction. Here one register reference, one memory reference is required to access the data.
2. Memory Indirect: In this mode effective address is in the memory, and corresponding memory address will be maintained in the address field of an instruction. Here two memory reference is required to access the data.

- **Indexed addressing mode**: The operand's offset is the sum of the content of an index register SI or DI and an 8 bit or 16-bit displacement.

Example: MOV AX, [SI +05]

- **Based Indexed Addressing:** The operand's offset is sum of the content of a base register BX or BP and an index register SI or DI.

Example: ADD AX, [BX+SI]

Advantages of Addressing Modes

3. To give programmers to facilities such as Pointers, counters for loop controls, indexing of data and program relocation.
4. To reduce the number bits in the addressing field of the Instruction.

Numerical on Addressing Mode

An instruction is stored at location 200 with its address field at location 201. The address filed has the value 500. A processor register R1 contain the number 400 and index register value is 100. Evaluate the effective address if the addressing mode of the instruction is given in the statement as (i) Direct (ii) Immediate (iii) Indirect (iv) Relative (v) Index (vi) Register (vii) Register Indirect (viii) Auto Increment (ix) Auto Decrement

Address	Memory	
200	Load to AC	Mode
201	Address=500	
202	Next instruction	
:		
:		
399	450	
400	700	
:		
500	800	
:		
600	900	
:		
702	325	
:		
800	300	

Solution

Sr. No.	Addressing Mode	Effective Address	Content to AC
1.	Direct	500	800
2.	Immediate	201	500
3.	Indirect	800	300
4.	Relative	PC+Address = 202+500=702	325
5.	Index	Index Register (XR) XR+Address= 100+500=600	900
6.	Register	_____	400
7.	Register Indirect	400	700
8.	Auto Increment	400 (After Processing value increased by 1 ie. 400+1=401)	700
9.	Auto Decrement	400-1=399 (Value first decremented by 1)	450

DATA TRANSFER AND MANIPULATION

Instruction set of different computers differ from each other mostly in way the operands are determined from the address and mode fields.

The basic set of operations available in a typical computer are:

1. **Data Transfer Instructions**
2. **Data Manipulation Instruction**:perform arithmetic, logic and shift operation

- **Program Control Instructions:**decision making capabilities, change the path taken by theprogram when executed in computer.

3. **Data Transfer Instructions**: Move data from one place in computer to another without changing the data content Most common transfer: processor reg -memory, processor reg -I/O, between processor register themselves.

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

Some assembly language conventions modify the mnemonic symbol to differentiate between the different addressing modes

Mode	Assembly convention	Register transfer
Direct address	LD ADR	AC ¬ M[ADR]
Indirect address	LD @ADR	AC ¬ M[M[ADR]]
Relative address	LD $ADR	AC ¬ M[PC + ADR]
Immediate operand	LD #NBR	AC ¬ NBR
Index addressing	LD ADR(X)	AC ¬ M[ADR + XR]
Register	LD R1	AC ¬ R1
Register indirect	LD (R1)	AC ¬ M[R1]
Auto-increment	LD (R1)+	AC ¬ M[R1], R1 ¬ R1 + 1
Auto-decrement	LD -(R1)	R1 ¬ R1 - 1, AC ¬ M[R1]

4. **Data Manipulation**

These instruction performs operation on data and provide the computational capabilities for the computer.

Three Basic Types:

- **Arithmetic instructions**
- **Logical and bit manipulation instructions**
- **Shift instructions**

Four basic arithmetic 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 Instructions perform binary operations on string of bits stored in registers

→Useful for manipulating individual/ group of bits

→ Consider each bit separately

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

AND →Clear selected bits

OR →Set selected bits

XOR →Complement selected bits

Shift Instructions

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 thru carry	RORC
Rotate left thru carry	ROLC

Program Control Instruction: These instructions are of various types. These are used in assembly language by user also. But in assembly level language, user code is translated into machine code and thus instructions are passed to instruct the processor do the task. These are having decision making capabilities, change the path taken by the program when executed in computer. are the machine code that are used by machine or in assembly language by user to command the processor act accordingly.

Name	Mnemonic
Branch	BR
Jump	JMP

Skip	SKP
Call	CALL
Return	RTN
Compare(by –)	CMP
Test(by AND)	TST

CISC AND RISC: FEATURES AND COMPARISON

Reduced Set Instruction Set Architecture (RISC) –

The main idea behind is to make hardware simpler by using an instruction set composed of a few basic steps for loading, evaluating and storing operations just like a load command will load data, store command will store the data.

Complex Instruction Set Architecture (CISC) –

The main idea is that a single instruction will do all loading, evaluating and storing operations just like a multiplication command will do stuff like loading data, evaluating and storing it, hence it's complex.

Both approaches try to increase the CPU performance

- **RISC:** Reduce the cycles per instruction at the cost of the number of instructions per program.
- **CISC:** The CISC approach attempts to minimize the number of instructions per program but at the cost of increase in number of cycles per instruction.

$$CPU\ Time = \frac{Seconds}{Program} = \frac{Instructions}{Program} \times \frac{Cycles}{Instructions} \times \frac{Seconds}{Cycle}$$

Earlier when programming was done using assembly language, a need was felt to make instruction do more task because programming in assembly was tedious and error prone due to which CISC architecture evolved but with up rise of high level language dependency on assembly reduced RISC architecture prevailed.

Characteristic of RISC –

1. Simpler instruction, hence simple instruction decoding.
2. Instruction come under size of one word.

3. Instruction take single clock cycle to get executed.
4. More number of general purpose register.
5. Simple Addressing Modes.
6. Less Data types.
7. Pipeline can be achieved.

Characteristic of CISC –

1. Complex instruction, hence complex instruction decoding.
2. Instruction are larger than one-word size.
3. Instruction may take more than single clock cycle to get executed.
4. Less number of general purpose register as operation get performed in memory itself.
5. Complex Addressing Modes.
6. More Data types.

Difference –

RISC	CISC
Focus on software	Focus on hardware
Uses only Hardwired control unit	Uses both hardwired and micro programmed control unit
Transistors are used for more registers	Transistors are used for storing complex Instructions
Fixed sized instructions	Variable sized instructions
Can perform only Register to Register Arithmetic operations	Can perform REG to REG or REG to MEM or MEM to MEM
Requires more number of registers	Requires less number of registers
Code size is large	Code size is small

RISC	CISC
A instruction execute in single clock cycle	Instruction take more than one clock cycle
A instruction fit in one word	Instruction are larger than size of one word

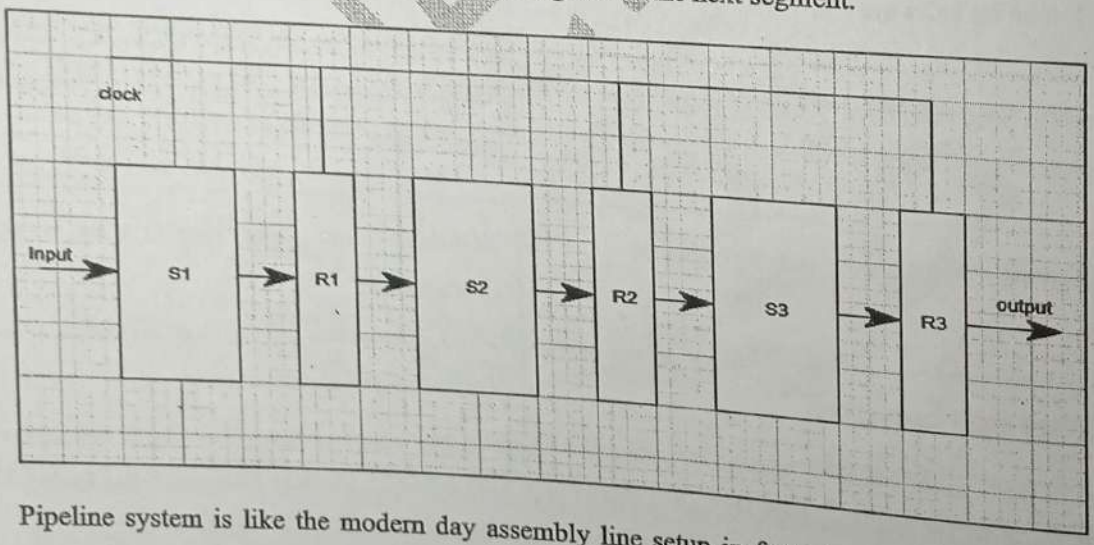
PIPELINE

Pipelining is the process of accumulating instruction from the processor through a pipeline. It allows storing and executing instructions in an orderly process. It is also known as **pipeline processing**.

Pipelining is a technique where multiple instructions are overlapped during execution. Pipeline is divided into stages and these stages are connected with one another to form a pipe like structure. Instructions enter from one end and exit from another end.

Pipelining increases the overall instruction throughput.

In pipeline system, each segment consists of an input register followed by a combinational circuit. The register is used to hold data and combinational circuit performs operations on it. The output of combinational circuit is applied to the input register of the next segment.



Pipeline system is like the modern day assembly line setup in factories. For example, in a car manufacturing industry, huge assembly lines are setup and at each point, there are robotic arms to perform a certain task, and then the car moves on ahead to the next arm.

Types of Pipeline

It is divided into 2 categories:

1. Arithmetic Pipeline
2. Instruction Pipeline

Arithmetic Pipeline

Arithmetic pipelines are usually found in most of the computers. They are used for floating point operations, multiplication of fixed point numbers etc. For example: The input to the Floating Point Adder pipeline is:

X=A*2^a

Y=B*2^b

Here A and B are mantissas (significant digit of floating point numbers), while **a** and **b** are exponents.

The floating point addition and subtraction is done in 4 parts:

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

Registers are used for storing the intermediate results between the above operations.

Instruction Pipeline

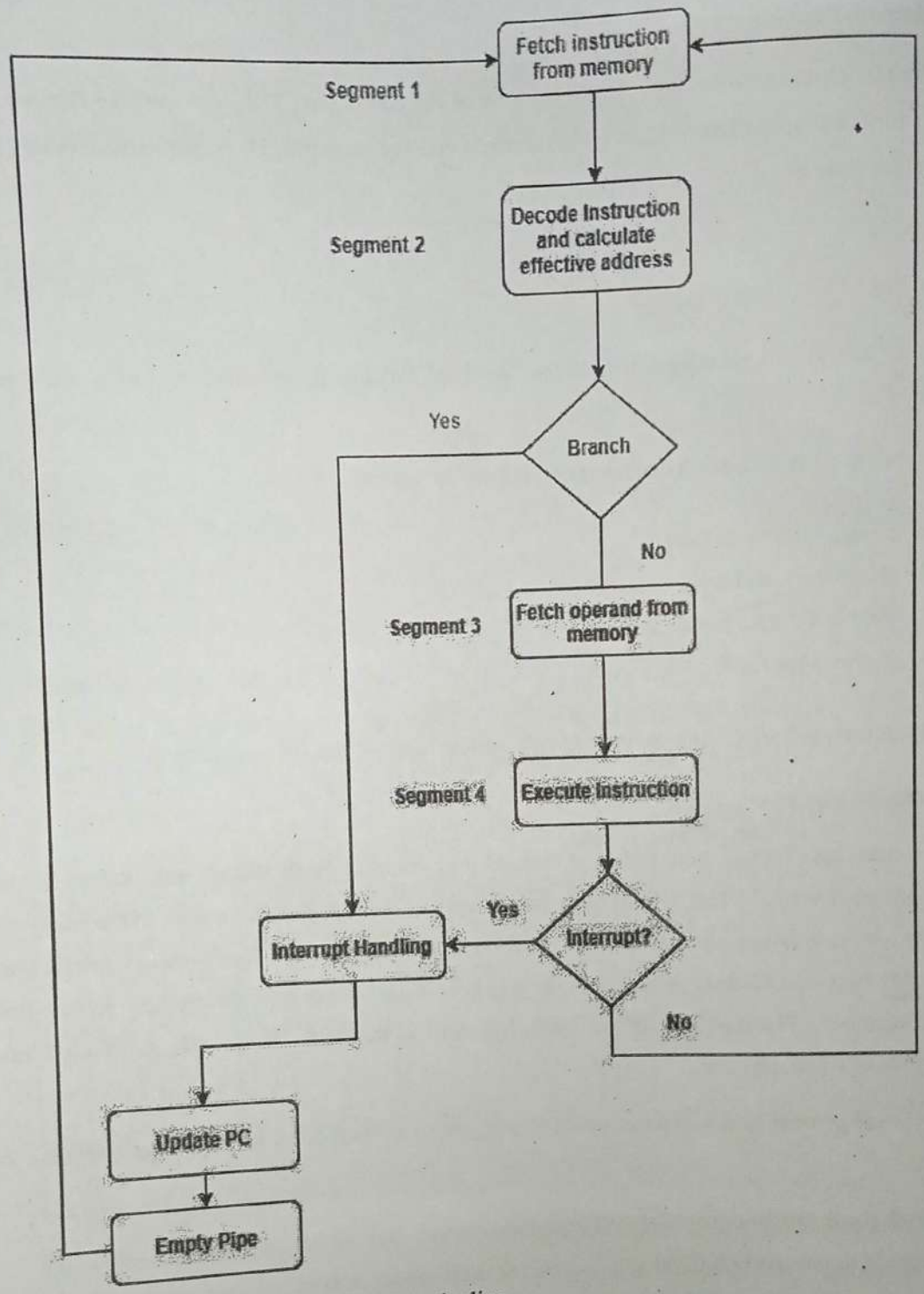
In this a stream of instructions can be executed by overlapping fetch, decode and execute phases of an instruction cycle. This type of technique is used to increase the throughput of the computer system. An instruction pipeline reads instruction from the memory while previous instructions are being executed in other segments of the pipeline. Thus we can execute multiple instructions simultaneously. The pipeline will be more efficient if the instruction cycle is divided into segments of equal duration.

In the most general case computer needs to process each instruction in following sequence of steps:

1. Fetch the instruction from memory (FI)
2. Decode the instruction (DA)
3. Calculate the effective address
4. Fetch the operands from memory (FO)
5. Execute the instruction (EX)

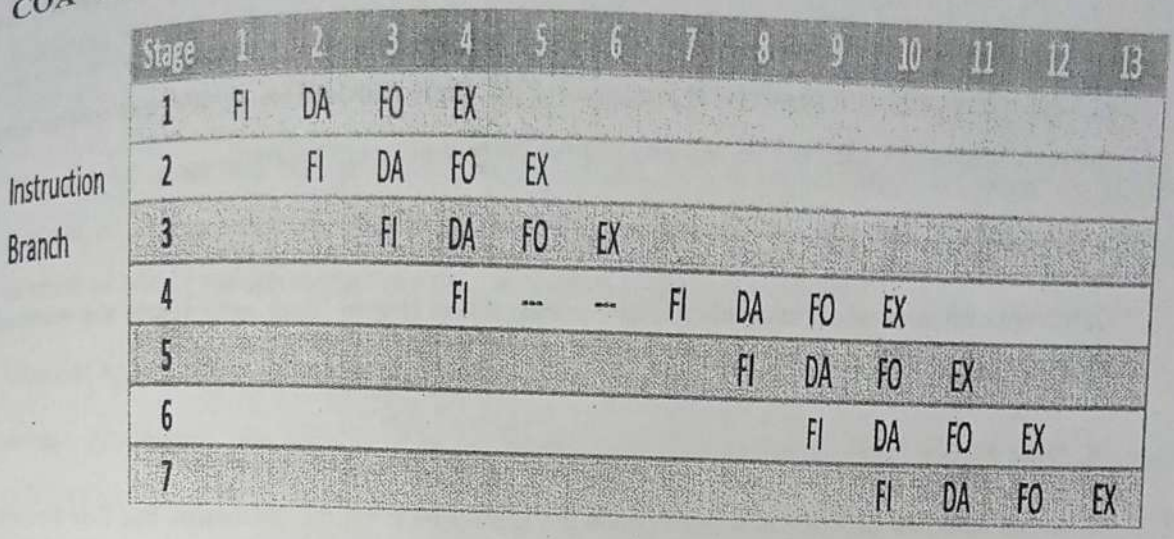
6. Store the result in the proper place

The flowchart for instruction pipeline is shown below.



Let us see an example of instruction pipeline.

Example:



Here the instruction is fetched on first clock cycle in segment 1. Now it is decoded in next clock cycle, then operands are fetched and finally the instruction is executed. We can see that here the fetch and decode phase overlap due to pipelining. By the time the first instruction is being decoded, next instruction is fetched by the pipeline.

In case of third instruction we see that it is a branched instruction. Here when it is being decoded 4th instruction is fetched simultaneously. But as it is a branched instruction it may point to some other instruction when it is decoded. Thus fourth instruction is kept on hold until the branched instruction is executed. When it gets executed then the fourth instruction is copied back and the other phases continue as usual.

Pipeline Conflicts

There are some factors that cause the pipeline to deviate its normal performance. Some of these factors are given below:

1. Timing Variations

All stages cannot take same amount of time. This problem generally occurs in instruction processing where different instructions have different operand requirements and thus different processing time.

2. Data Hazards

When several instructions are in partial execution, and if they reference same data then the problem arises. We must ensure that next instruction does not attempt to access data before the current instruction, because this will lead to incorrect results.

3. Branching

In order to fetch and execute the next instruction, we must know what that instruction is. If the present instruction is a conditional branch, and its result will lead us to the next instruction, then the next instruction may not be known until the current one is processed.

4. Interrupts

Interrupts set unwanted instruction into the instruction stream. Interrupts effect the execution of instruction.

5. Data Dependency

It arises when an instruction depends upon the result of a previous instruction but this result is not yet available.

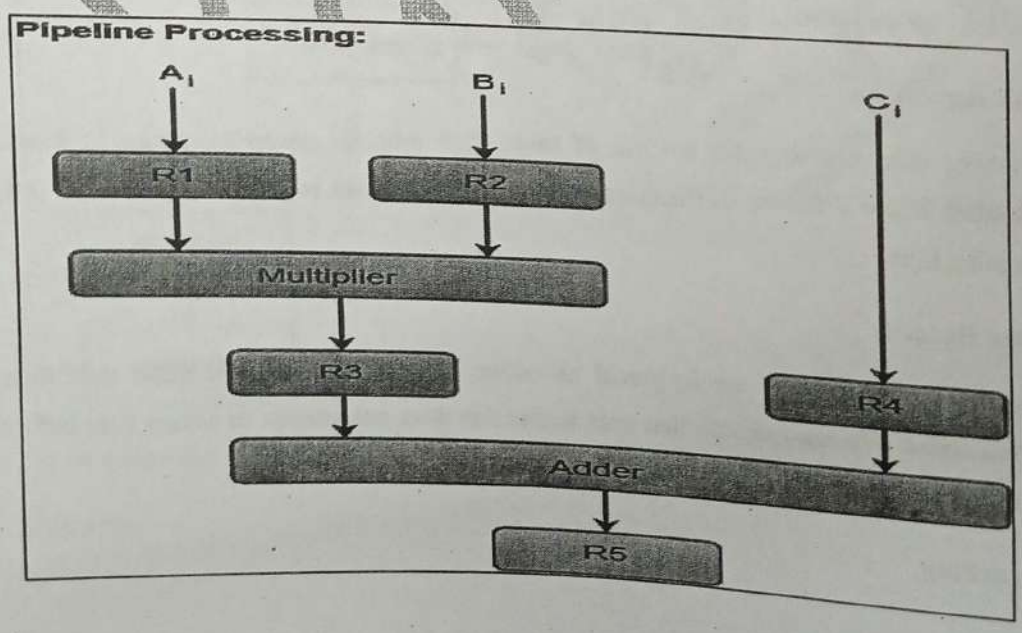
Advantages of Pipelining

1. The cycle time of the processor is reduced.
2. It increases the throughput of the system
3. It makes the system reliable.

Disadvantages of Pipelining

1. The design of pipelined processor is complex and costly to manufacture.
2. The instruction latency is more.

The following block diagram represents the combined as well as the sub-operations performed in each segment of the pipeline.



Registers R1, R2, R3, and R4 hold the data and the combinational circuits operate in a particular segment.

The output generated by the combinational circuit in a given segment is applied as an input register of the next segment. For instance, from the block diagram, we can see that the register R3 is used as one of the input registers for the combinational adder circuit.

Vector(Array) Processor and its Types

Array processors are also known as multiprocessors or vector processors. They perform computations on large arrays of data. Thus, they are used to improve the performance of the computer.

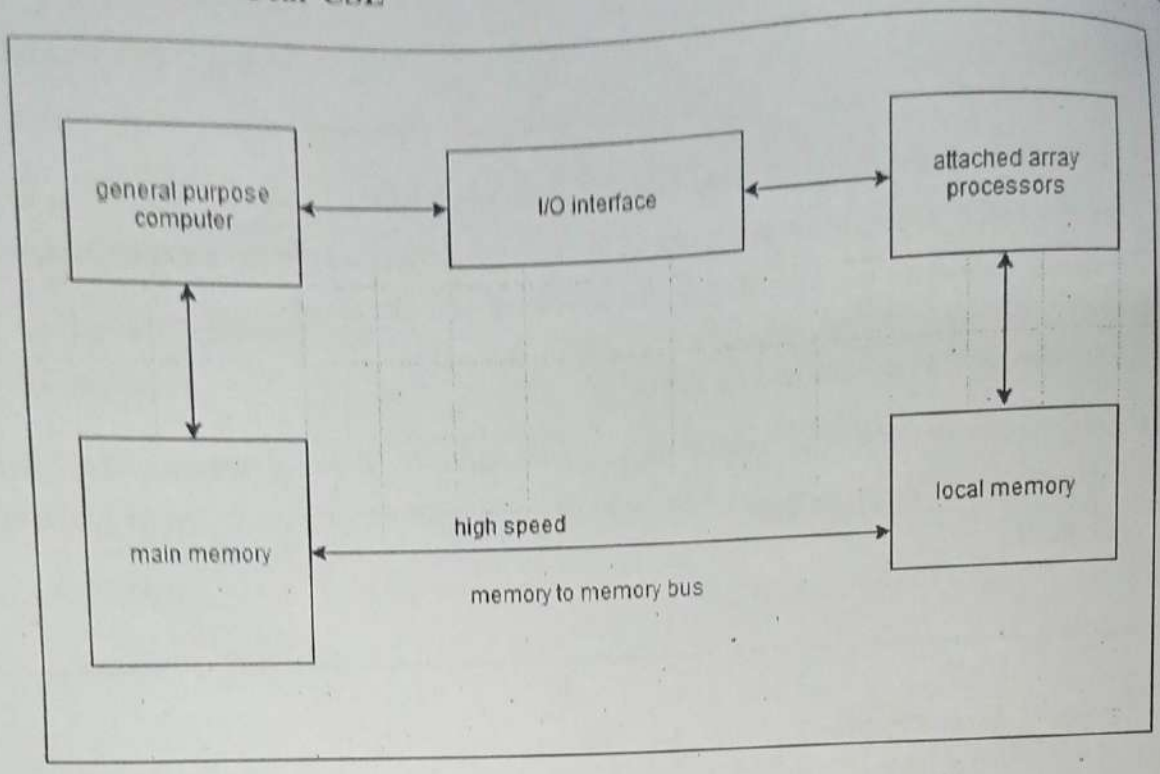
Types of Array Processors

There are basically two types of array processors:

1. Attached Array Processors
2. SIMD Array Processors

Attached Array Processors

An attached array processor is a processor which is attached to a general purpose computer and its purpose is to enhance and improve the performance of that computer in numerical computational tasks. It achieves high performance by means of parallel processing with multiple functional units.



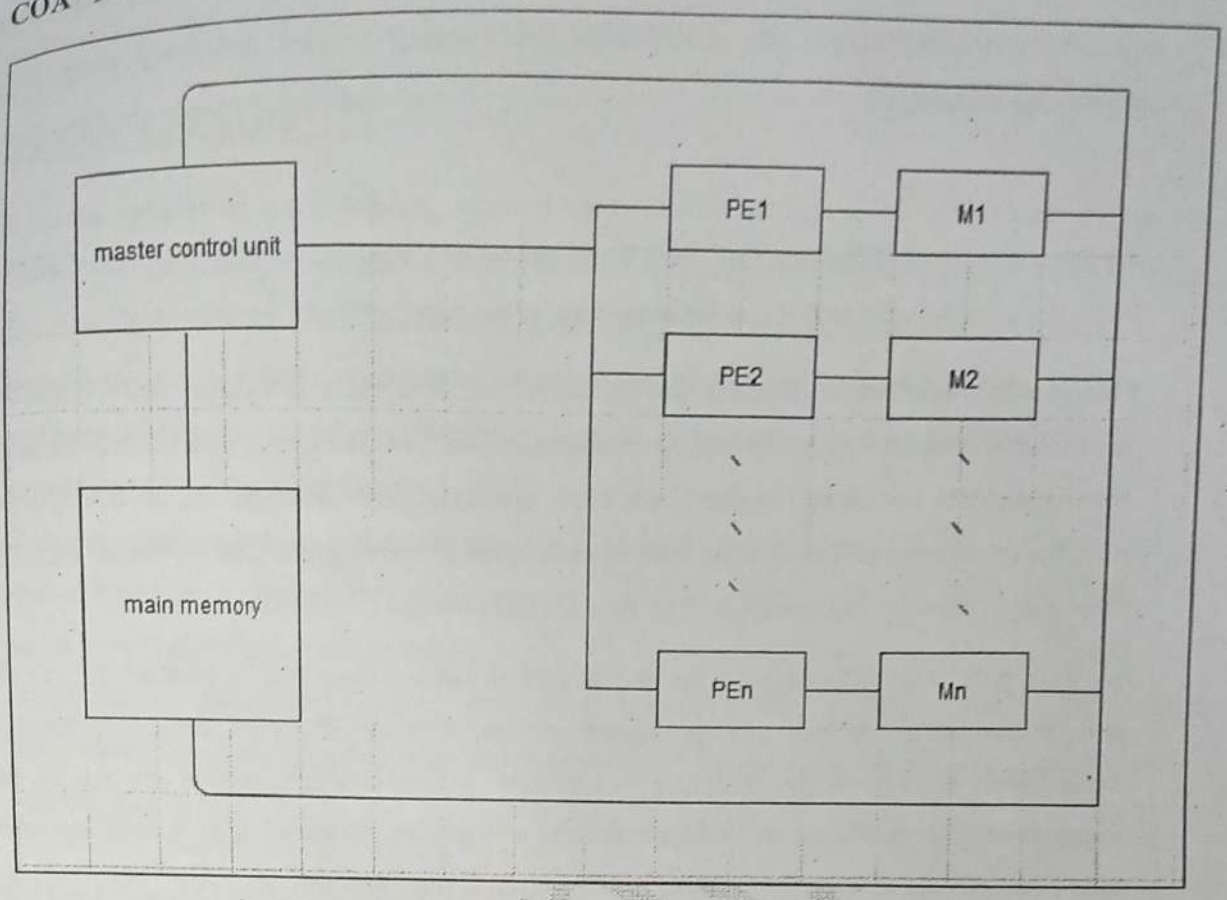
SIMD Array Processors

SIMD is the organization of a single computer containing multiple processors operating in parallel. The processing units are made to operate under the control of a common control unit, thus providing a single instruction stream and multiple data streams.

A general block diagram of an array processor is shown below. It contains a set of identical processing elements (PE's), each of which is having a local memory M. Each processor element includes an **ALU** and **registers**. The master control unit controls all the operations of the processor elements. It also decodes the instructions and determines how the instruction is to be executed.

The main memory is used for storing the program. The control unit is responsible for fetching the instructions. Vector instructions are send to all PE's simultaneously and results are returned to the memory.

The best known SIMD array processor is the **ILLIAC IV** computer developed by the **Burroughs corps**. SIMD processors are highly specialized computers. They are only suitable for numerical problems that can be expressed in vector or matrix form and they are not suitable for other types of computations.



PARALLEL PROCESSING

For the purpose of increasing the computational speed of computer system, the term '**parallel processing** 'employed to give simultaneous data-processing operations is used to represent a large class. In addition, a parallel processing system is capable of concurrent data processing to achieve faster execution times.

As an example, the next instruction can be read from memory, while an instruction is being executed in ALU. The system can have two or more ALUs and be able to execute two or more instructions at the same time. In addition, two or more processing is also used to speed up computer processing capacity and increases with parallel processing, and with it, the cost of the system increases. But, technological development has reduced hardware costs to the point where parallel processing methods are economically possible.

Parallel processing derives from multiple levels of complexity. It is distinguished between parallel and serial operations by the type of registers used at the lowest level. Shift registers work one bit at a time in a serial fashion, while parallel registers work simultaneously with all bits of simultaneously with all bits of the word. At high levels of complexity, parallel processing derives from having a plurality of functional units that perform separate or similar

operations simultaneously. By distributing data among several functional units, parallel processing is installed.

As an example, arithmetic, shift and logic operations can be divided into three units and operations are transformed into a teach unit under the supervision of a control unit.

One possible method of dividing the execution unit into eight functional units operating in parallel is shown in figure. Depending on the operation specified by the instruction, operands in the registers are transferred to one of the units, associated with the operands. In each functional unit, the operation performed is denoted in each block of the diagram. The arithmetic operations with integer numbers are performed by the adder and integer multiplier.

Floating-point operations can be divided into three circuits operating in parallel. Logic, shift, and increment operations are performed concurrently on different data. All units are independent of each other, therefore one number is shifted while another number is being incremented. Generally, a multi-functional organization is associated with a complex control unit to coordinate all the activities between the several components.

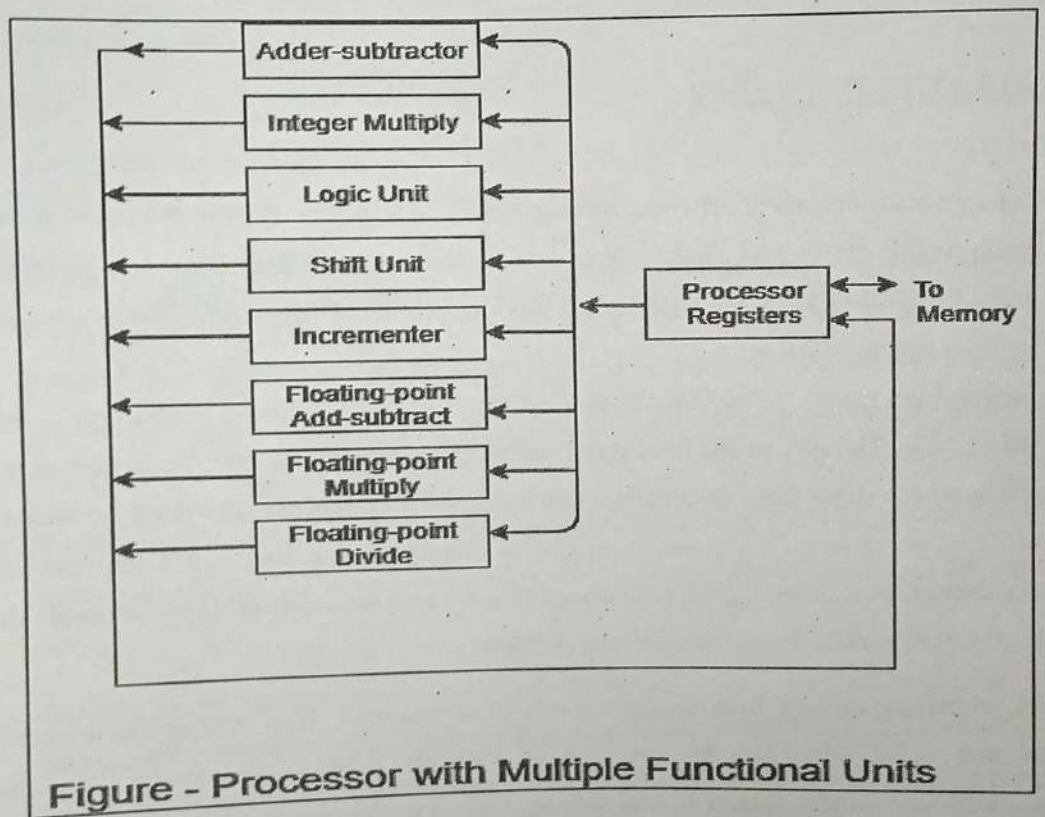


Figure - Processor with Multiple Functional Units

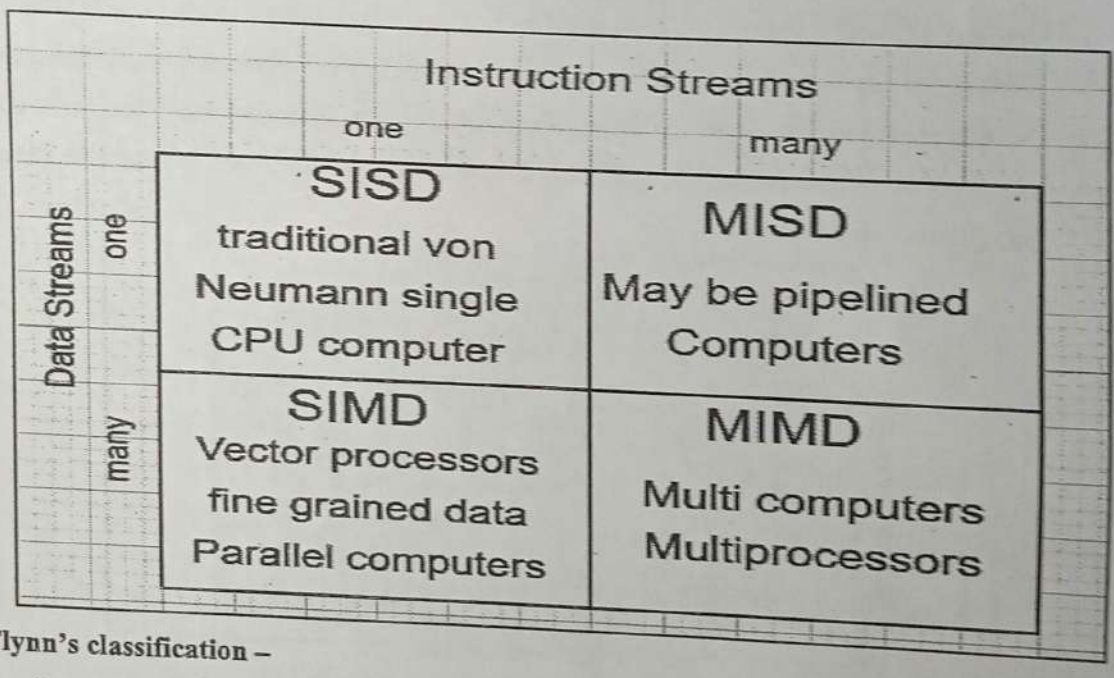
The main advantage of parallel processing is that it provides better utilization of system resources by increasing resource multiplicity which overall system throughput.

FLYNN'S TEXONOMY

Parallel computing is a computing where the jobs are broken into discrete parts that can be executed concurrently. Each part is further broken down to a series of instructions. Instructions from each part execute simultaneously on different CPUs. Parallel systems deal with the simultaneous use of multiple computer resources that can include a single computer with multiple processors, a number of computers connected by a network to form a parallel processing cluster or a combination of both.

Parallel systems are more difficult to program than computers with a single processor because the architecture of parallel computers varies accordingly and the processes of multiple CPUs must be coordinated and synchronized.

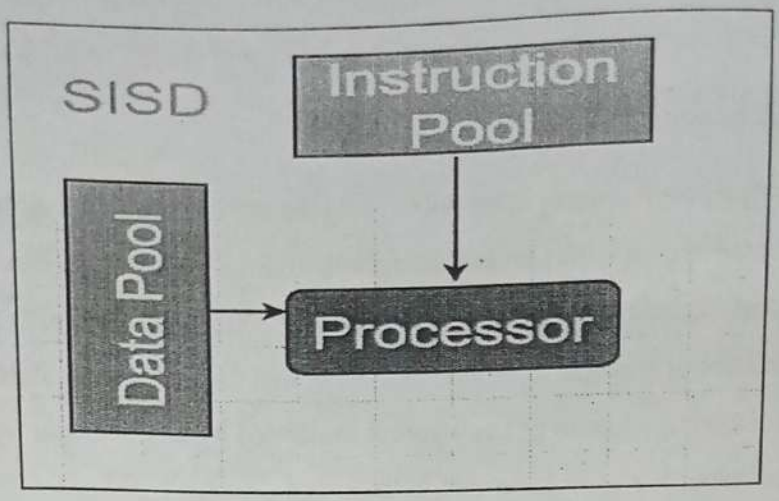
The crux of parallel processing are CPUs. Based on the number of **instruction and data** streams that can be processed simultaneously, computing systems are classified into four major categories:



Flynn's classification –

1. **Single-instruction, single-data (SISD) systems –**

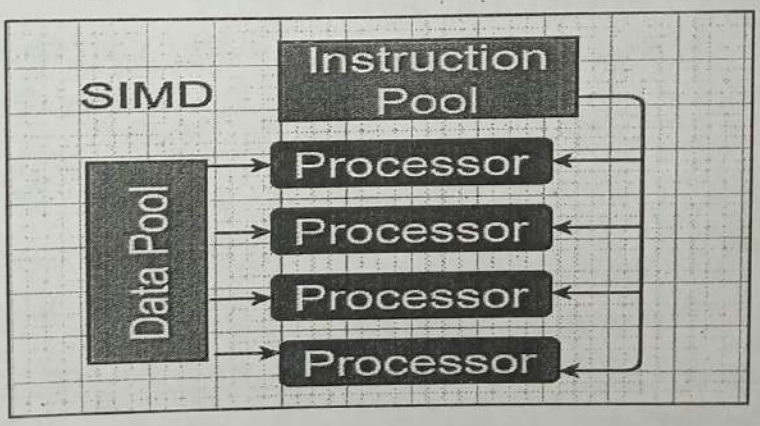
   An SISD computing system is a uniprocessor machine which is capable of executing a single instruction, operating on a single data stream. In SISD, machine instructions are processed in a sequential manner and computers adopting this model are popularly called sequential computers. Most conventional computers have SISD architecture. All the instructions and data to be processed have to be stored in primary memory.



The speed of the processing element in the SISD model is limited(dependent) by the rate at which the computer can transfer information internally. Dominant representative SISD systems are IBM PC, workstations.

2. **Single-instruction, multiple-data (SIMD) systems –**

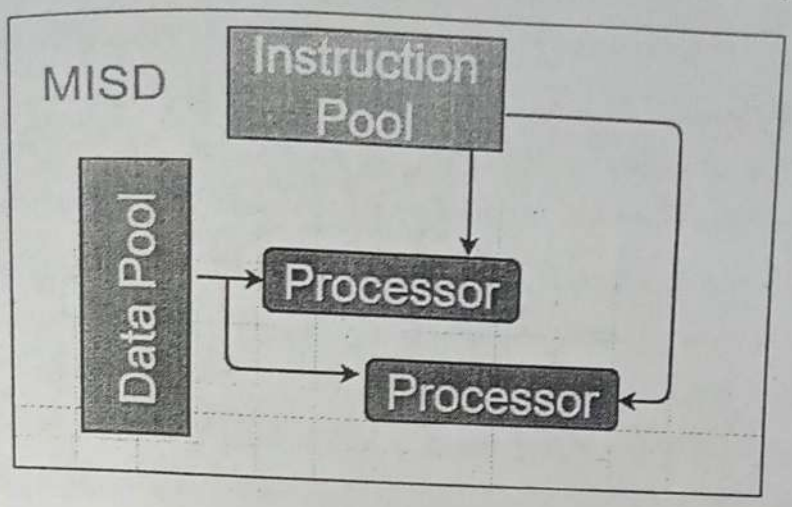
An SIMD system is a multiprocessor machine capable of executing the same instruction on all the CPUs but operating on different data streams. Machines based on an SIMD model are well suited to scientific computing since they involve lots of vector and matrix operations. So that the information can be passed to all the processing elements (PEs) organized data elements of vectors can be divided into multiple sets(N-sets for N PE systems) and each PE can process one data set.



Dominant representative SIMD systems is Cray's vector processing machine.

3. **Multiple-instruction, single-data (MISD) systems –**

An MISD computing system is a multiprocessor machine capable of executing different instructions on different PEs but all of them operating on the same dataset :

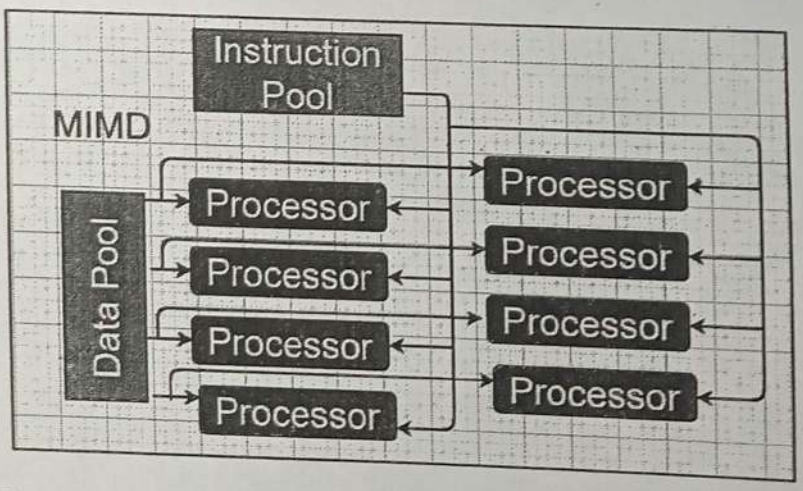


Example Z = sin(x)+cos(x)+tan(x)

The system performs different operations on the same data set. Machines built using the MISD model are not useful in most of the application, a few machines are built, but none of them are available commercially.

4. **Multiple-instruction, multiple-data (MIMD) systems –**

An MIMD system is a multiprocessor machine which is capable of executing multiple instructions on multiple data sets. Each PE in the MIMD model has separate instruction and data streams; therefore, machines built using this model are capable to any kind of application. Unlike SIMD and MISD machines, PEs in MIMD machines work asynchronously.



MIMD machines are broadly categorized into **shared-memory MIMD** and **distributed-memory MIMD** based on the way PEs are coupled to the main memory.In the **shared memory MIMD** model (tightly coupled multiprocessor systems), all the PEs are connected to a single global memory and they all have access to it. The communication between PEs in this model takes place through the shared memory, modification of the data stored in the global memory by one PE is visible to all other PEs. Dominant representative shared

memory MIMD systems are Silicon Graphics machines and Sun/IBM's SMP (Symmetric Multi-Processing). In **Distributed memory MIMD** machines (loosely coupled multiprocessor systems) all PEs have a local memory. The communication between PEs in this model takes place through the interconnection network (the inter process communication channel, or IPC). The network connecting PEs can be configured to tree, mesh or in accordance with the requirement. The shared-memory MIMD architecture is easier to program but is less tolerant to failures and harder to extend with respect to the distributed memory MIMD model. Failures in a shared-memory MIMD affect the entire system, whereas this is not the case of the distributed model, in which each of the PEs can be easily isolated. Moreover, shared memory MIMD architectures are less likely to scale because the addition of more PEs leads to memory contention. This is a situation that does not happen in the case of distributed memory, in which each PE has its own memory. As a result of practical outcomes and user's requirement, distributed memory MIMD architecture is superior to the other existing models.