Unit 6

Parallel Processing

- Parallel processing is a technique to provide simultaneous data processing tasks for the purpose of increasing the computational speed of a computer.
- Conventional computer processes each instruction sequentially.
- A parallel processing system is able to perform concurrent data processing to achieve faster execution time.
- For example:
    - While instruction is being executed in the ALU the next instruction can be read from memory.
    - The system may have two or more ALU's and be able to execute two or more instructions at the same time.
- The purpose of parallel processing is to speed up the computer processing capability and increase its throughput, that is, the amount of processing that can be accomplished during a given interval of time.
- Parallel processing can be viewed from various levels of complexity.
- At lowest level, we distinguish between parallel and serial operations by the type of registers used.
- Shift registers operate in serial fashion one bit at a time, while registers with parallel load operates will all the bits of the word simultaneously.
- Parallel processing at a higher level of complexity can be achieved by having a multiplicity of functional units that perform identical or different operations simultaneously.
- Parallel processing is established by distributing the data among the multiple functional units.
- Example:

- Arithmetic, logical and shift operations can be separated into three units and the operands diverted to each unit under the supervision of a control unit.

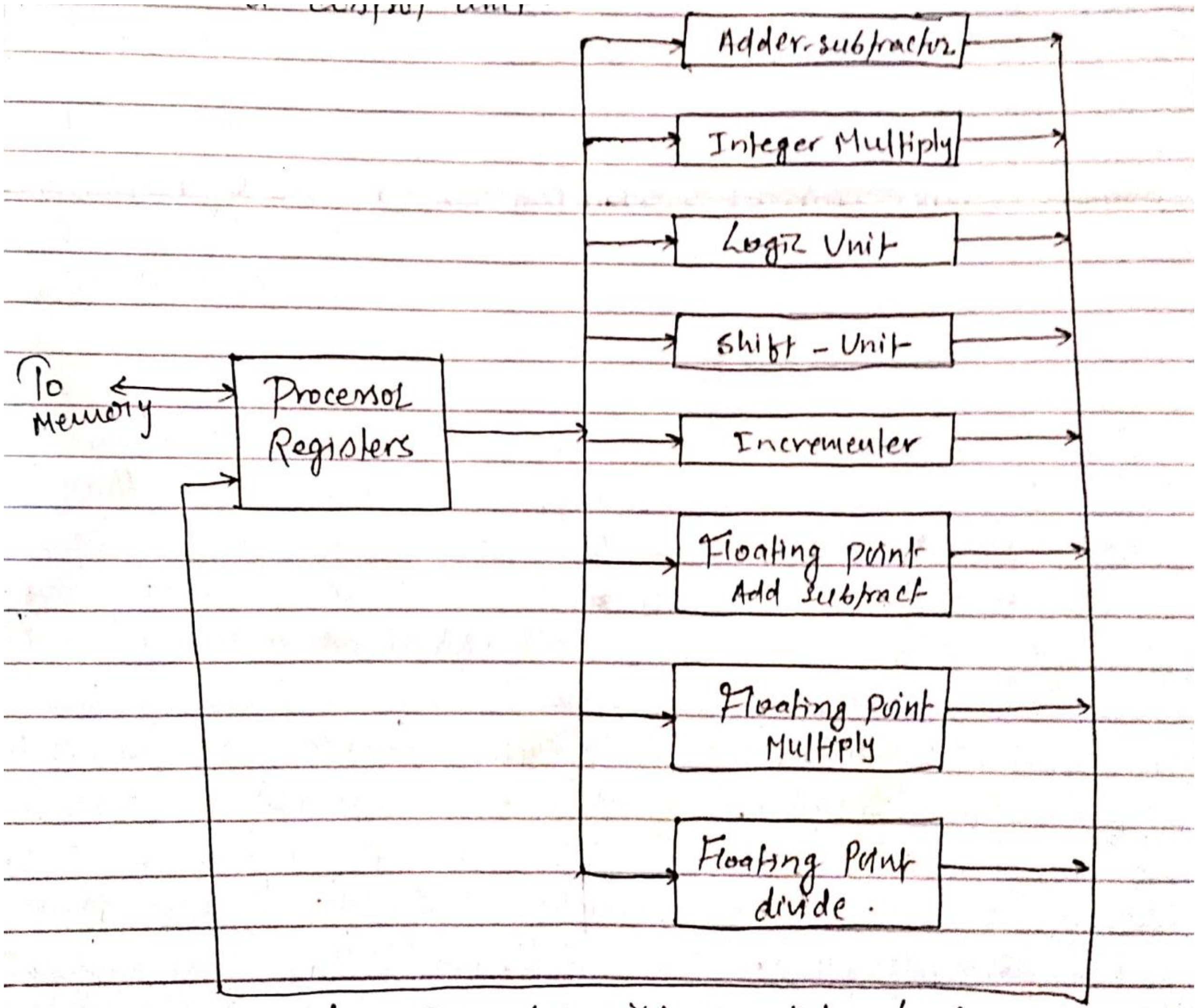
- 

fig:- Processor with Multiple functional Unit

M.J. Flynn classification of computer

- One classification of computers introduced by M.J. Flynn considers the organization of a computer system by the number of instructions and data items that are manipulated simultaneously.

- Flynn's classification divides computers into four major groups as follows:
    - **Single instruction stream, single data stream (SISD).**
    - **Single instruction stream, multiple data stream (SIMD)**
    - **Multiple instruction stream, single data stream (MISD).**
    - **Multiple instruction stream, multiple data stream (MIMD).**
- **HW. Explain all these four classifications in brief.**
- **HW. Define pipeline? Explain.**

Instruction Pipeline

- An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments.
- 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 the memory
    - Execute the instruction
    - Store the result in the proper place.
- There are certain difficulties that will prevent the instruction pipeline from operating at its maximum rate.
- Different segments may take different times to operate on the incoming information.
- Some segments are skipped for certain operations.

Example: Four segment instruction pipeline

- Assume that the decoding of the instruction can be combined with the calculation of the effective address into one segment.

- Assume further that most of the instructions place the result into a processor register so that the instruction execution and storing of the result can be combined into one segment.
- This reduces the instruction pipeline into four segments.

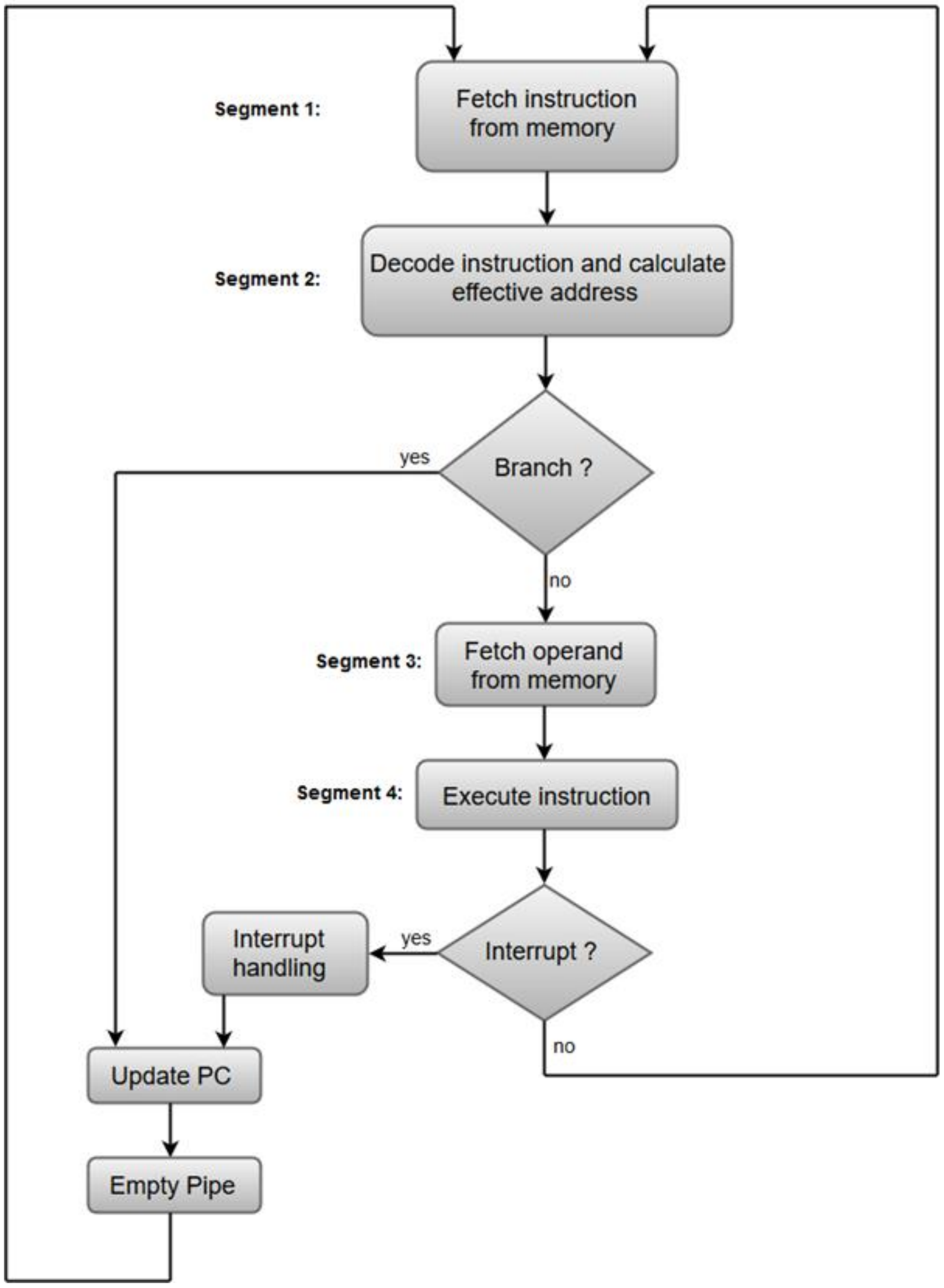
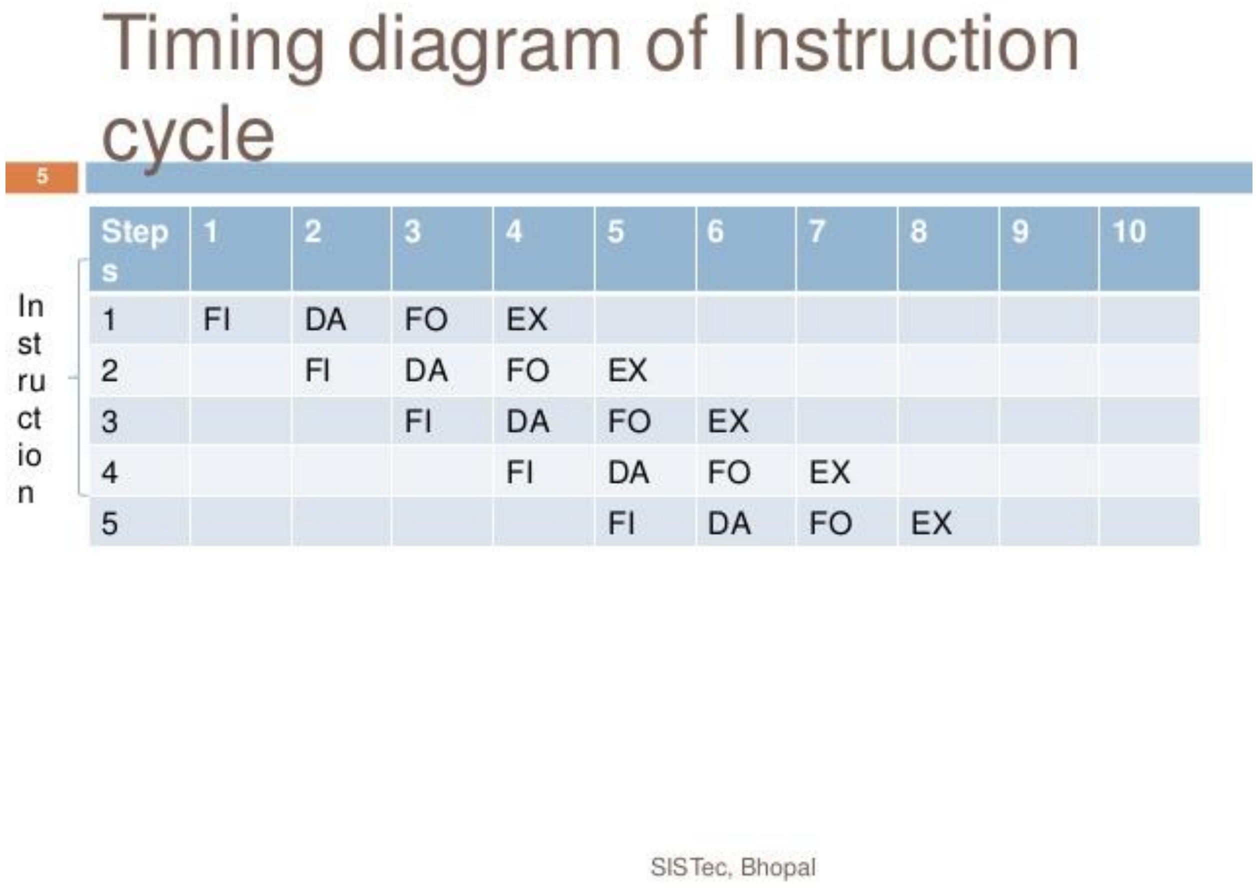


Fig: Four Segment CPU Pipeline

- It shows how the instruction cycle in the CPU can be processed with a four segment pipeline.



Timing diagram of Instruction cycle

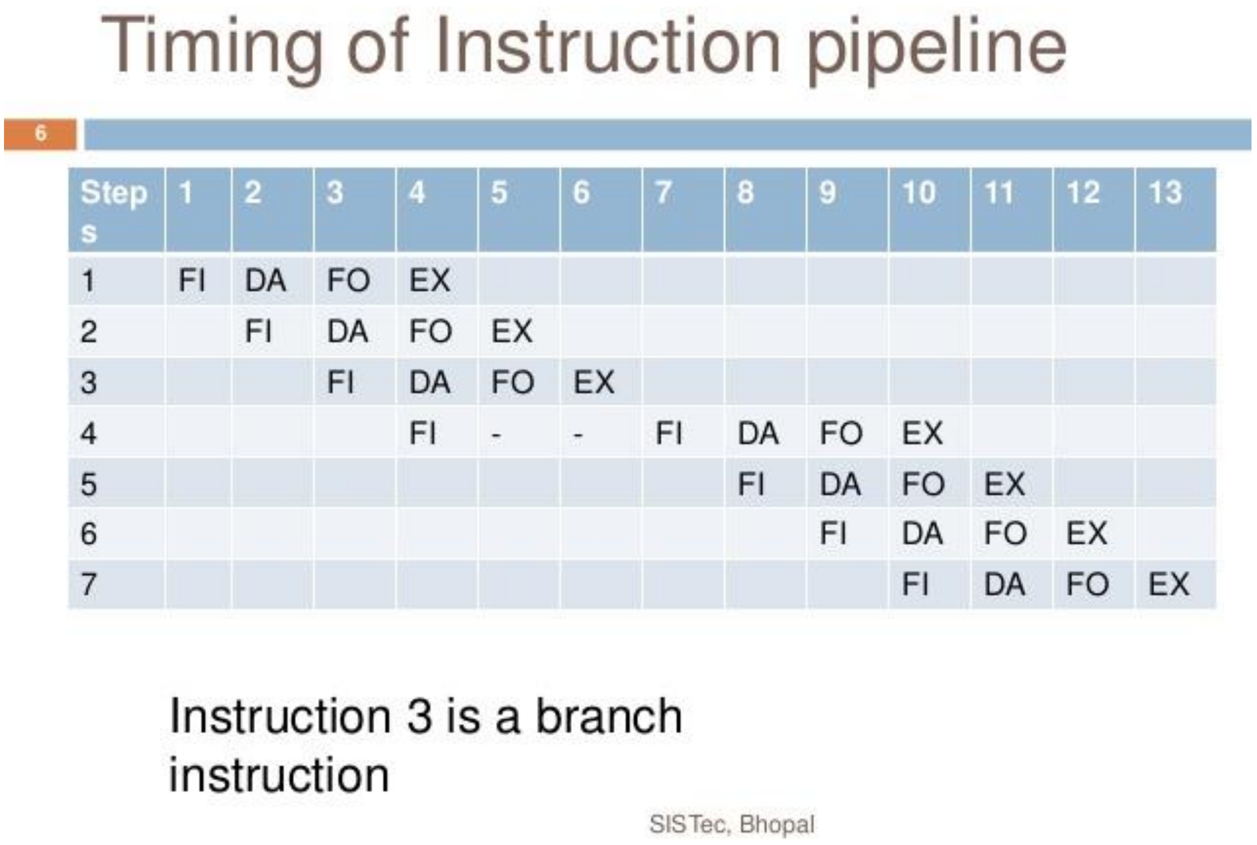
5

Instruction

Steps	1	2	3	4	5	6	7	8	9	10
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		

SISTec, Bhopal

fig



Timing of Instruction pipeline

6

Steps	1	2	3	4	5	6	7	8	9	10	11	12	13
1	FI	DA	FO	EX									
2		FI	DA	FO	EX								
3			FI	DA	FO	EX							
4				FI	-	-	FI	DA	FO	EX			
5								FI	DA	FO	EX		
6									FI	DA	FO	EX	
7										FI	DA	FO	EX

Instruction 3 is a branch instruction

SISTec, Bhopal

Fig: timing for instruction pipeline

- The above fig shows the operation of the instruction pipeline.
- The time is the horizontal axis is divided into steps of equal duration.
- The four segments are represented in a diagram with an abbreviated symbols.
  - FI is the segment that fetches an instruction
  - DA is the segment that decode the instruction and calculate the effective address.
  - FO is the segment that fetches the operand.
  - EX is the segment that executes the instruction.

- It is assumed that the processor has separated instruction and data memories so that the operation in FI and FO can proceed at the same time.

Qs. What do you mean by pipeline conflict (pipeline hazards).

- **Resource Conflict (soln)**
- **Data dependency conflict (soln)**
- **Branch difficulties conflict (soln)**

Pipeline conflicts (pipeline hazards)

- In general, there are three major difficulties that cause the instruction pipeline to deviate from its normal operation.
- **Resource Conflict:**
    - This is caused by access to memory by two segments at the same time.
    - Most of these conflicts can resolved by using separate instruction and data memories.
- **Data Dependency:**
    - This conflicts arise when an instruction depends on the result of a previous instruction, but this result is not yet available.
    - This problem can be resolved by inserting **hardware interlocks**. An interlock is a circuit that detects instructions whose source operations are destinations of instructions farther up in the pipeline.
    - Another technique called **operand forwarding** uses special hardware to detect a conflict and then avoid it by routing the data through special paths between pipeline segments.

- **Branch difficulties:**
  - This arise from branch and other instructions that change the value of PC.
  - This difficulty can be solved by:
    - Prefetching the target instruction.
    - By using branch target buffer.
    - By using loop buffer
    - By branch prediction
    - Delayed branch.
- HW: **Handling of Branch Instruction**

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 $V = [V_1\ V_2\ V_3\ \ldots\ V_n]$. It may be represented as a column vector if the data items are listed in a column.
- To examine the difference between a conventional scalar processor and a vector processor, consider the following DO loop:
- Consider the problem: C = A + B

```
        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. This is implemented in machine language by the following sequence of operation.

```
        Initialize I = 1
    20  Read A(I)
        Read B(I)
        Store C(I) = A(I) + B(I)
        Increment I = I + 1
        If I <= 100 go to 20
        Continue
```

- This constitutes a program loop that reads a pair of operations from arrays A and B and performs a floating-point addition. The loop control variable is then updated and the steps repeat 100 times.
- 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)

    1 ➔ Initial address

    100 ➔ Length.

Matrix Multiplication

- Matrix multiplication is one of the most computational intensive operations performed in computers with vector processors. The multiplication of two nXn matrices consists of n^2 inner products or n^3 multiply-add operations.
- In general matrix multiplication can be represented as:

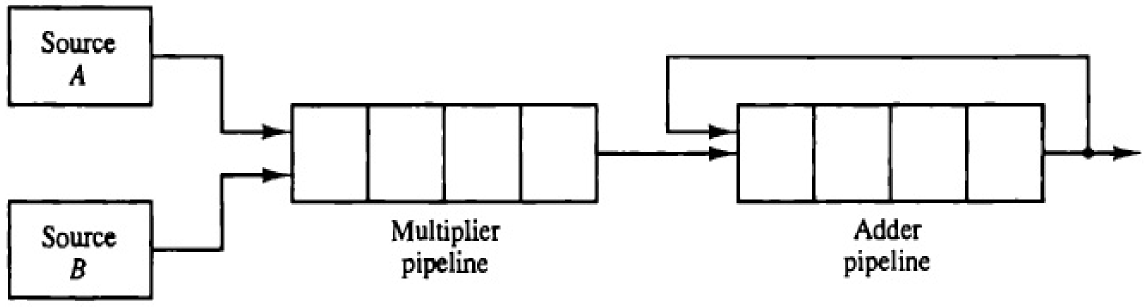
  $C_{ij} = \sum_{a=1}^{b} Aia \; X \; Baj$ where i and j are representing position of row and column respectively.

  Where matrix A is size of mXn and matrix B is size of nXk. The final result is in matrix C of size mXk.

Fig: Instruction format for vector processor

Operation code	Base address source 1	Base address source 2	Base address destination	Vector length

Figure 9-12 Pipeline for calculating an inner product.



$C = A_1B_1 + A_2B_2 + A_3B_3 + A_4B_4 + \ldots\ldots\ldots + A_kB_k$