Pipelining

Contents

- **6.1 Introduction :** Parallel Processing, Multiple Functional Units, Flynn's Classification
- **6.2 Pipelining:** Concept and Demonstration with Example, Speedup Equation, Floating Point addition and Subtraction with Pipelining
- **6.3 Instruction Level Pipelining:** Instruction Cycle, Three & Four-Segment Instruction Pipeline, Pipeline Conflicts and Solutions (Resource Hazards, Data Hazards, Branch hazards)
- **6.4 Vector Processing:** concept and Applications, Vector Operations, Matrix Multiplication

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. Or A system may have two or more processors operating concurrently.
- 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.

- The amount of increase 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.

- Parallel processing can be viewed from various levels of complexity.
- At the 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 operate with all the bits of the word simultaneously.
- At a higher level of complexity cab be achieved by have 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.
- 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.

- Figure shows one possible way of separating the execution unit into eight functional units operating in parallel. The operands in the registers are applied to one of the units depending on the operation specified by the instruction associated with the operands.
- The adder and integer multiplier performs the arithmetic operation 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.

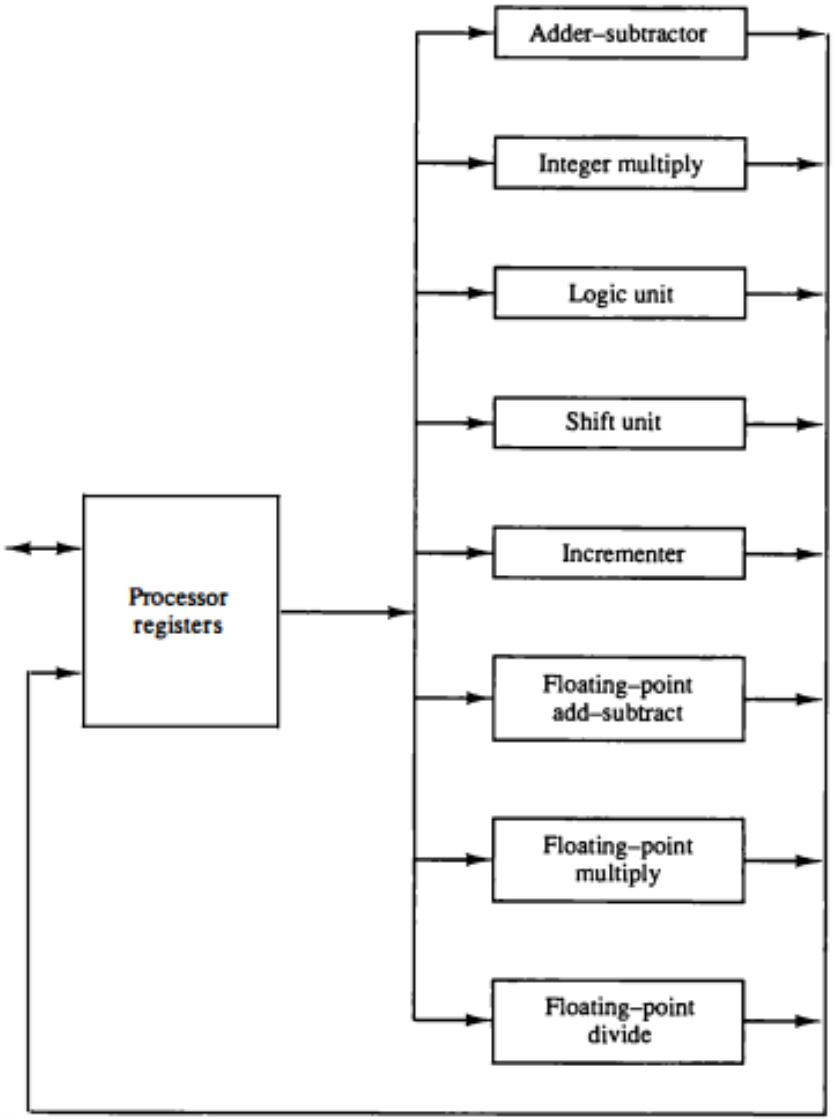


Figure : Processor with multiple functional units.

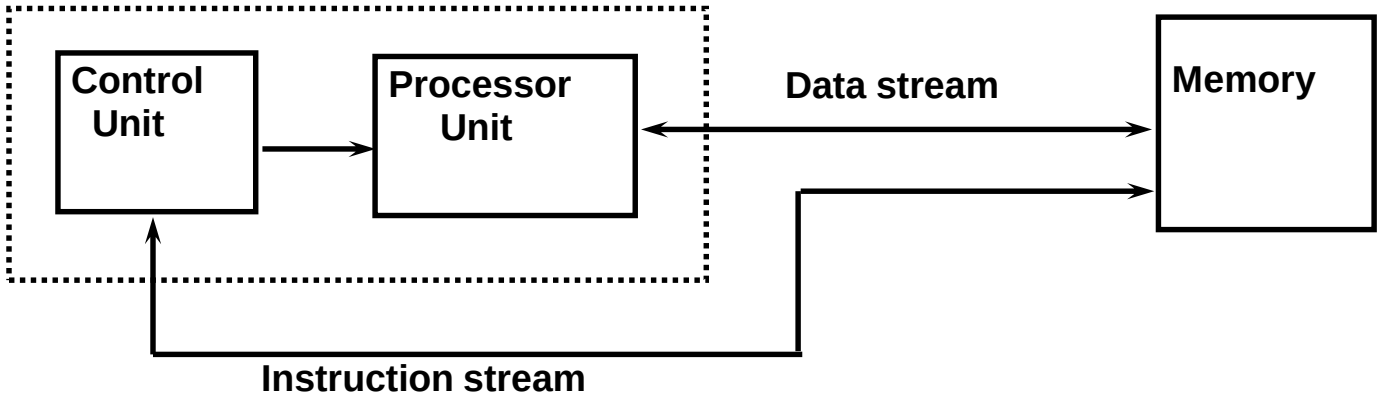
- There are a variety of ways that parallel processing can be classified.
- There are variety of ways in which the parallel processing can be classified
  - the internal organization of the processors,
  - the interconnection structure between processors, or
  - the flow of information through the system.
- One classification introduced by M.J. Flynn considers the organization of a computer system by the number of instructions and data items that are manipulated simultaneously.
- The sequence of instructions read from memory constitutes **an instruction stream**.
- The operations performed on the data in the processor constitute **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:

- Single instruction stream, single data stream (SISD)
- Single instruction stream, multiple data stream (SIMD)
- Multiple instruction streams, single data stream (MISD)
- Multiple instruction streams, multiple data stream (MIMD)

		Number of *Data Streams*	
		Single	Multiple
Number of *Instruction Streams*	Single	SISD	SIMD
	Multiple	MISD	MIMD

SISD

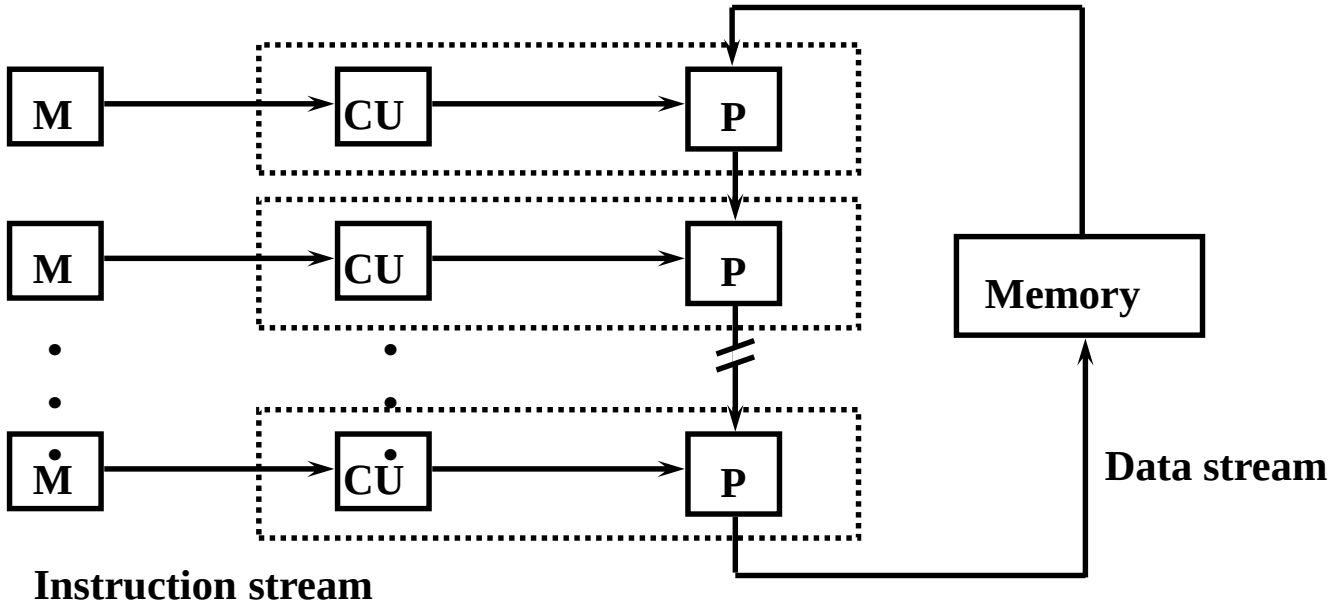
- SISD represents the organizations of a single computer containing a control unit, a processor unit, and a memory unit.
- Instructions are executed sequentially and the system may or may not have internal parallel processing capabilities.
- Parallel processing in this case may be achieved by means of multiple functional units or by pipeline processing.
- Most conventional computers have SISD architecture like the traditional Von-Neumann computers.



- Instructions are decoded by the Control Unit and then the Control Unit sends the instructions to the processing units for execution.
- Data Stream flows between the processors and memory bi-directionally.
- Examples**:** Older generation computers, minicomputers, and workstations

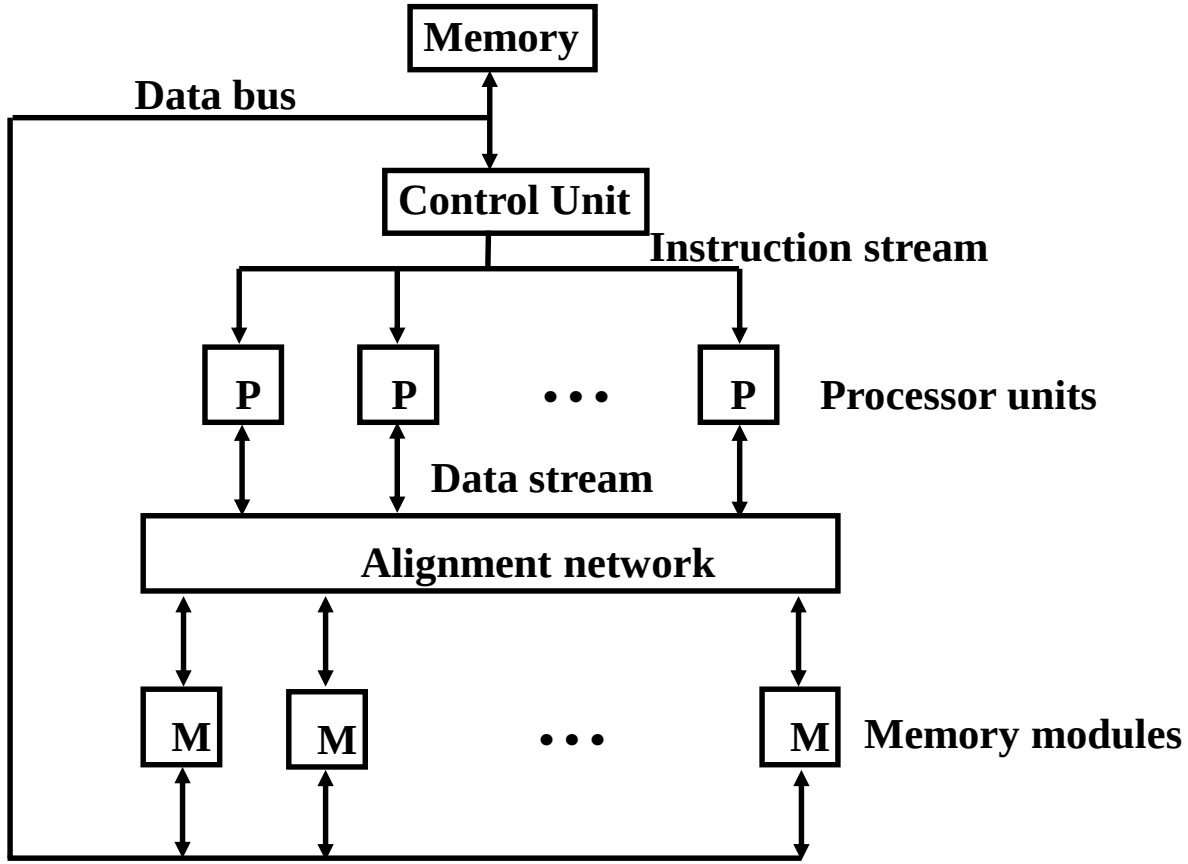
MISD

- MISD structure is only of theoretical interest since no practical system has been constructed using this organization.
- In MISD, multiple processing units operate on one single-data stream. Each processing unit operates on the data independently via separate instruction stream.



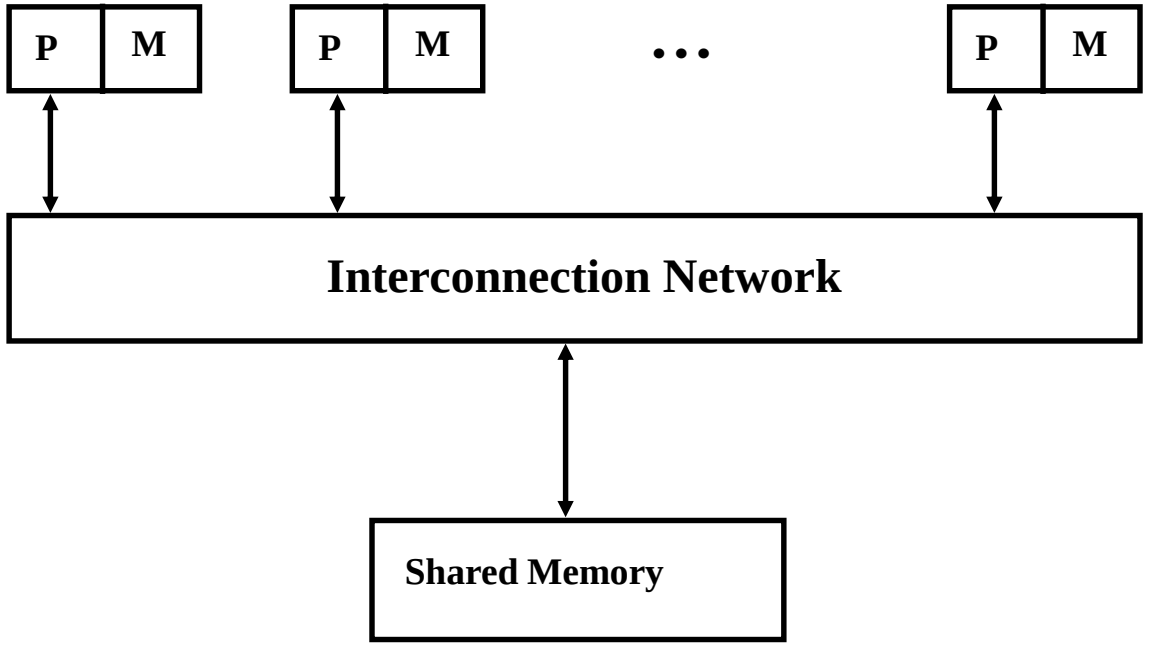
SIMD

- 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.
- SIMD is mainly dedicated to array processing machines. However, vector processors can also be seen as a part of this group.
- Example : Wireless MMX unit designed by intel.



MIMD

- MIMD organization refers to a computer system capable of processing several programs at the same time.
- In MIMD, each processor has a separate program and an instruction stream is generated from each program.
- Most multiprocessor and multi-computer systems can be classified in this category.
- Examples: Cray T90, Cray T3E, IBM-SP2.
- Types of MIMD computer systems
  - Shared memory multiprocessors
  - Message-passing multicomputer



- Shared memory processor , All processors have equally direct access to one large memory address space.

Here we are considering parallel processing under the following main topics:

1. Pipeline processing
2. Vector processing
3. Array processors

- Pipeline processing
  - Is an implementation technique where arithmetic suboperations or the phases of a computer instruction cycle overlap in execution.
- Vector processing
  - Deals with computations involving large vectors and matrices.
- Array processing
  - Perform computations on large arrays of data.

Pipelining

- Pipelining is a technique of decomposing a sequential process into sub operations, with each sub process being executed in special dedicated segments that operates concurrently with all other segments.
- A pipeline can be visualized as a collection of processing segments through which binary information flows. Each segment performs partial processing dictated by the way the task 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.

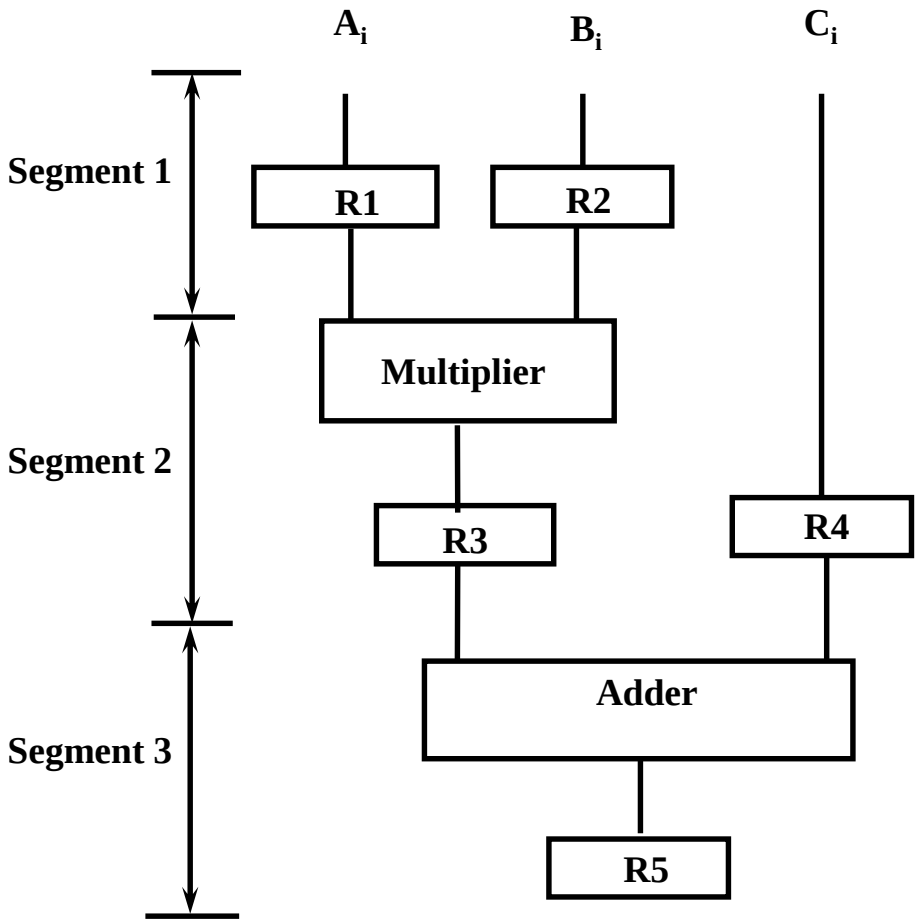


Figure : Example of pipeline processing.

- Suppose that we want to perform the combined multiply and add operations with a stream of numbers. A *B + C, for i = 1, 2, 3, . . . , 7
- Each segment has one or two registers and a combinational circuit as shown in figure. R1 through R5 are registers that receive new data with every clock pulse.
- The multiplier and adder are combinational circuits. The sub operations performed in each segment of the pipeline are as follows:

$R1 \leftarrow A_i$, $R2 \leftarrow B_i$	Load A_i and B_i
$R3 \leftarrow R1 * R2$, $R4 \leftarrow C_i$	Multiply and load C_i
$R5 \leftarrow R3 + R4$	Add C_i to product

- The first clock pulse transfers A1 and B1 into R1 and R2.
- The second clock pulse transfers the product of R1 and R2 into R3 and C1 into R4.
- The same clock pulse transfers A2 and B2 into R1 and R2.
- The third clock pulse operates on all three segments simultaneously.
- It places A3 and B3 into R1 and R2, transfers the product of R1 and R2 into R3, transfers C2 into R4, and places the sum of R3 and R4 into R5.
- It takes three clock pulses to fill up the pipe and retrieve the first output from R5.
- When no more input data are available, the clock must continue until the last output emerges out of the pipeline.

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A1	B1	-	-	-
2	A2	B2	A1 * B1	C1	-
3	A3	B3	A2 * B2	C2	A1 * B1 + C1
4	A4	B4	A3 * B3	C3	A2 * B2 + C2
5	A5	B5	A4 * B4	C4	A3 * B3 + C3
6	A6	B6	A5 * B5	C5	A4 * B4 + C4
7	A7	B7	A6 * B6	C6	A5 * B5 + C5
8	-	-	A7 * B7	C7	A6 * B6 + C6
9	-	-	-	-	A7 * B7 + C7

Table : Content of Registers in Pipeline Example.

General Pipeline

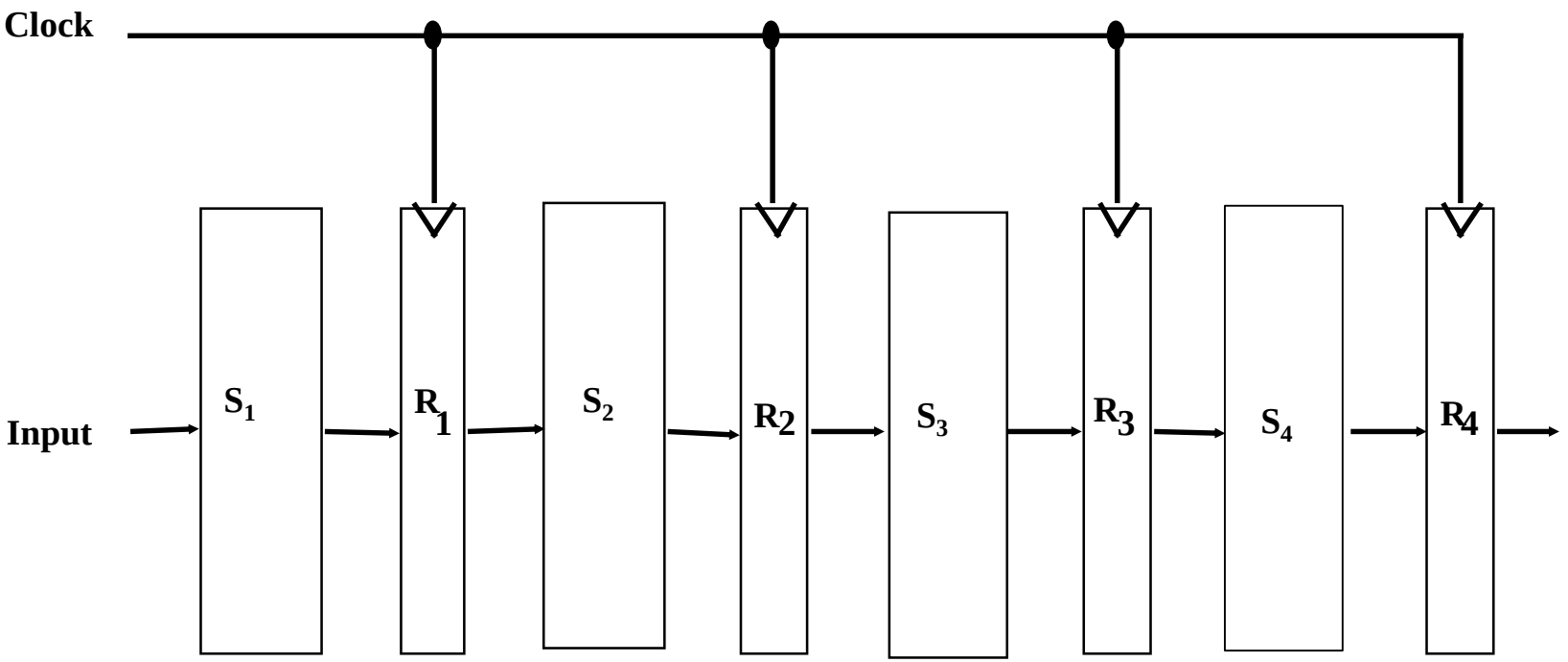


Figure : General Structure of a 4-Segment Pipeline.

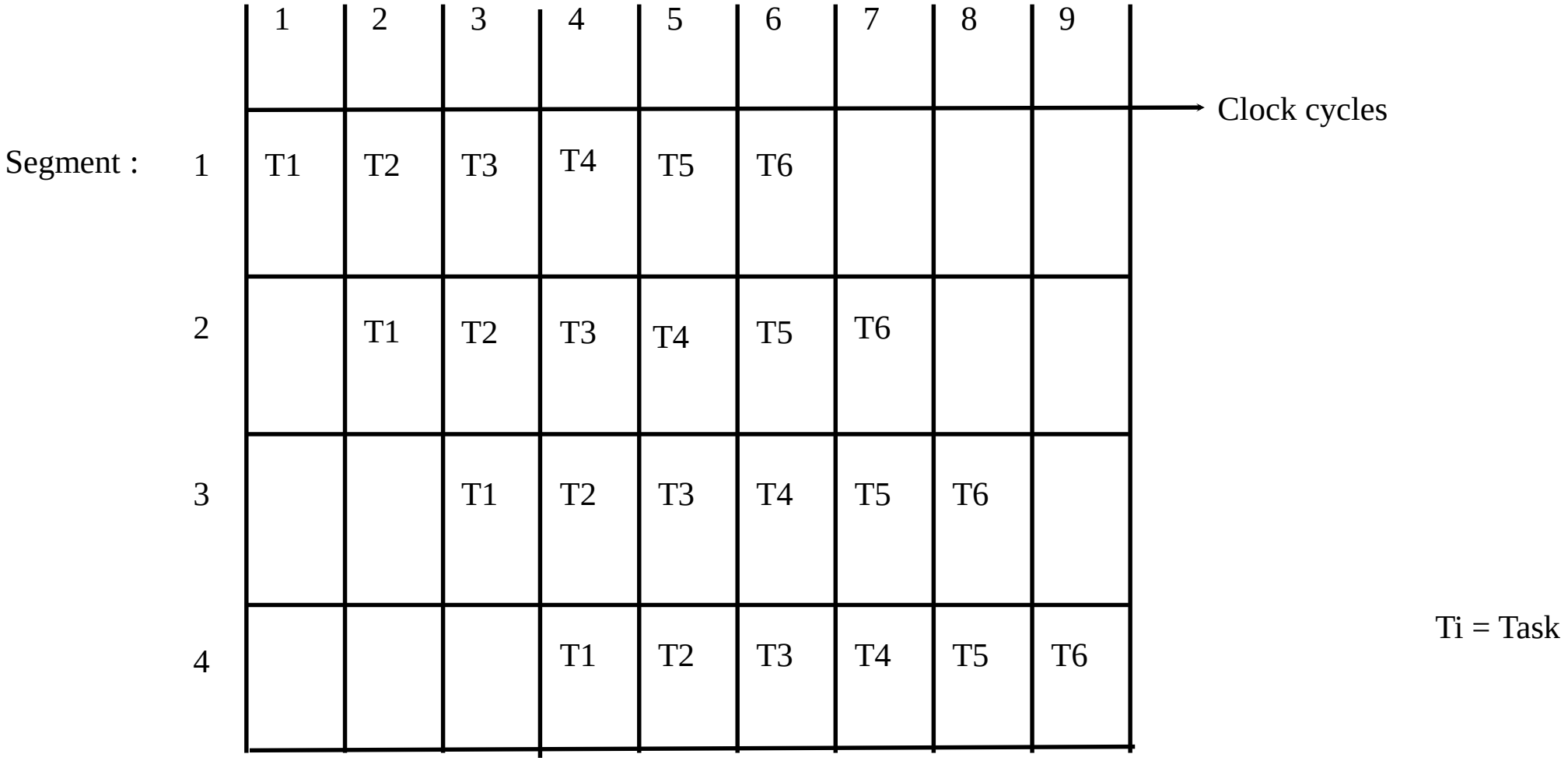
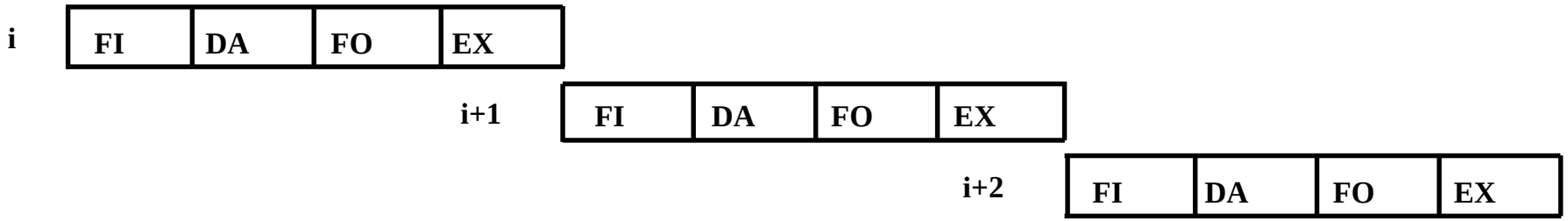


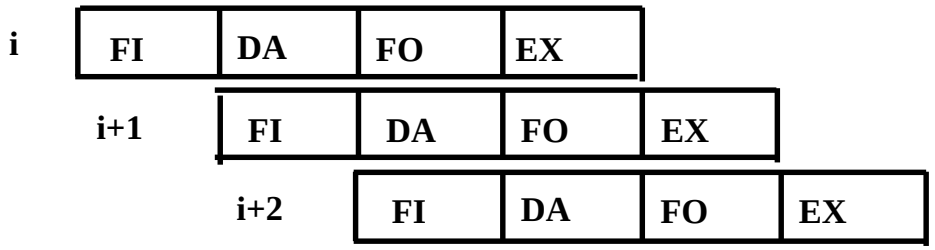
Figure : Space-time diagram for pipeline.

Execution of Three Instructions in a 4-Stage Pipeline

Conventional



Pipelined



Pipeline Speedup

n: Number of tasks to be performed

k: segments(stage)

Conventional Machine (Non-Pipelined)

t_n: Clock cycles

t_1: Time required to complete the n tasks

Total time $t_1 = n * t_n$

Pipelined Machine (k stages)

t_p: Clock cycle (time to complete each suboperation,time of the Largest segment)

t_k: Time required to complete the n tasks

Phase duration=(K+n-1), where,(n-1) remaining tasks completion = $(n-1)\ t_p$

Total time $(t_k) = (k + n - 1) * t_p$

Speedup

S_k: Speedup

Speedup ratio= Non-pipelined execution time /pipelined execution time = $S_k = n*t_n / (k + n - 1)*t_p$

$$S_k = \frac{t_n}{t_p} \quad (\text{ if } n >= k, \text{ if } t_n = k * t_p)$$

Example:

- 4-stage pipeline (k =4)
- subopertion in each stage; t_p = 20nS
- 100 tasks to be executed (n=100)
- 1 task in non-pipelined system; 20*4 = 80nS (t_n or k*t_p)

Pipelined System

$(k + n - 1)*t_p = (4 + 99) * 20 = 2060nS$

Non-Pipelined System

$n* tn = n*k*t_p = 100 * 80 = 8000nS$

Speedup

$S_k = 8000 / 2060 = 3.88$

Q. Non pipelined system takes 130ns to process an instruction . A program of 1000 instructions is executed in non pipelined system. Then same program is processed with processor with 5 segment pipeline with clock cycle of 30 ns/stage. Determine speed up ratio of pipeline.

Solution:

For a non-pipelined system:

Total number of instruction/task (n)=1000 ,Total time required to perform a single task in pipelined processor (T_p)=130 ns

total time to execute 1000 instructions in non-pipe line model = 1000 * 130 ns

For a pipelined system:

Total number of stages (k)=5, Total number of instruction/task (n)=1000

Total time required to perform a single task in pipelined processor (T_p)=30 ns

Total time to execute 1000 instructions in pipe line model = (k+n-1)*clock time of pipeline=(5+1000-1)* 30 ns = 1004 * 30

∴ Speed Up of pipeline = Time for non-pipeline mode/Time for pipeline model = 1000∗130/1004∗30 = 4.316 ns.

Q. Consider a pipeline having 4 phases with duration 60, 50, 90 and 80 ns. Given latch delay is 10 ns.

Calculate

1. Pipeline cycle time
2. Non-pipeline execution time
3. Speed up ratio
4. Pipeline time for 1000 tasks
5. Sequential time for 1000 tasks
6. Throughput

Solution:

Given-

Four stage pipeline is used

Delay of stages = 60, 50, 90 and 80 ns

Latch delay or delay due to each register = 10 ns

1: Pipeline Cycle Time-

Cycle time

= Maximum delay due to any stage + Delay due to its register

= Max { 60, 50, 90, 80 } + 10 ns

= 90 ns + 10 ns

= 100 ns

2: Non-Pipeline Execution Time-

Non-pipeline execution time for one instruction

= 60 ns + 50 ns + 90 ns + 80 ns

= 280 ns

3: Speed Up Ratio-

Speed up

= Non-pipeline execution time / Pipeline execution time

= 280 ns / Cycle time

= 280 ns / 100 ns

= 2.8

4: Pipeline Time For 1000 Tasks-

Pipeline time for 1000 tasks

= Time taken for 1st task + Time taken for remaining 999 tasks

= 1 x 4 clock cycles + 999 x 1 clock cycle

= 4 x cycle time + 999 x cycle time

= 4 x 100 ns + 999 x 100 ns

= 400 ns + 99900 ns

= 100300 ns

5: Sequential Time For 1000 Tasks-

Non-pipeline time for 1000 tasks

= 1000 x Time taken for one task

= 1000 x 280 ns

= 280000 ns

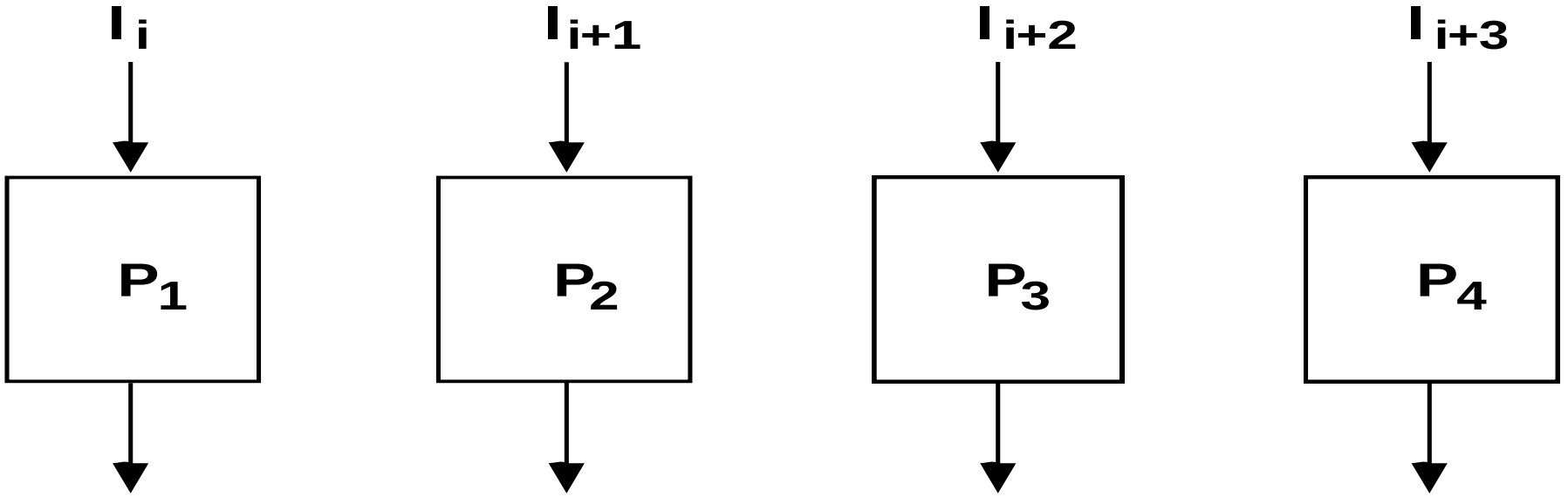
6: Throughput-

Throughput for pipelined execution = Number of instructions executed per unit time

= 1000 tasks / 100300 ns

Multiple Functional Units

- 4-Stage Pipeline is basically identical to the system with 4 identical function units.



Arithmetic pipeline

- 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.
- A pipeline multiplier is essentially an array multiplier , with special adders designed to minimize the carry propagation time through the partial products.
- The floating point addition and subtraction can be performed in four segments.
- We will discuss 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.

  $X = A \times 2^a$

  $Y = B \times 2^b$

floating point adder: The sub operations that are performed in the four segments are:

1. Compare the exponents
2. Align the mantissa
3. Add/sub the mantissa
4. Normalize the result

$X = 0.9504 \times 10^3$

$Y = 0.8200 \times 10^2$

Compare the exponent 3-2 = 1

Align the mantissa

$X = 0.9504 \times 10^3$

$Y = 0.0820 \times 10^3$

Add the mantissa

$Z = X + Y = 0.9504 * 10^3 + 0.0820 * 10^3 = 1.0324 * 10^3$

Normalize the result

$Z = 0.10324 \times 10^4$

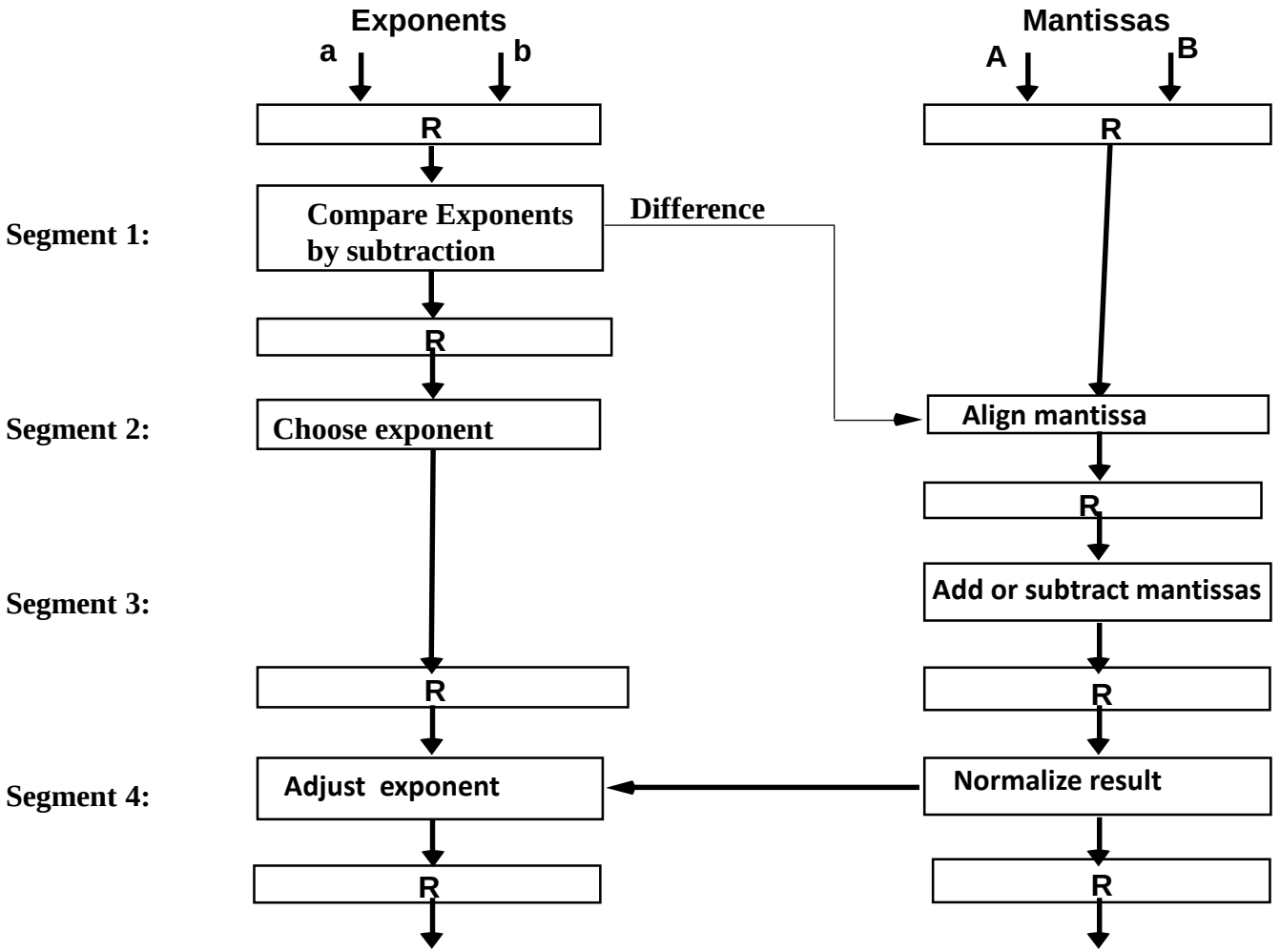


Figure : pipeline for floating-point addition and subtraction .

Instruction Pipeline

- Pipeline processing can occur not only in the data stream but in the instruction stream as well.
- An instruction pipeline reads consecutive instruction from memory while previous instruction are being executed in other segments. This caused the instruction fetch and execute segments to overlap and perform simultaneous operation

Steps in the Instruction Cycle

- Pipeline processing can occur also in the instruction stream. An **instruction pipeline** reads consecutive instruction from memory while previous instruction are executed in various segments of pipeline.
- The computer needs to process each instruction with the following sequence of steps.
    1. Fetch the instruction from memory.
    2. Decode the instruction.
    3. Calculate the effective address.
    4. Fetch the operands from memory.

5. Execute the instruction.

6. Store the result in the proper place

- Some instructions skip some phases
- Effective address calculation can be done in the part of the decoding phase
- Storage of the operation result into a register is done automatically in the execution phase

Example: Four-Segment Instruction Pipeline

The four segments are represented in the diagram with an abbreviated symbol.

1. FI: is the segment that fetch an instruction from memory
2. DA: is segment that decode the instruction and calculate the effective address of the operand
3. FO: is segment that fetch the operand
4. EX: is segment executes the instruction

Instruction execution in a 4-stage pipeline

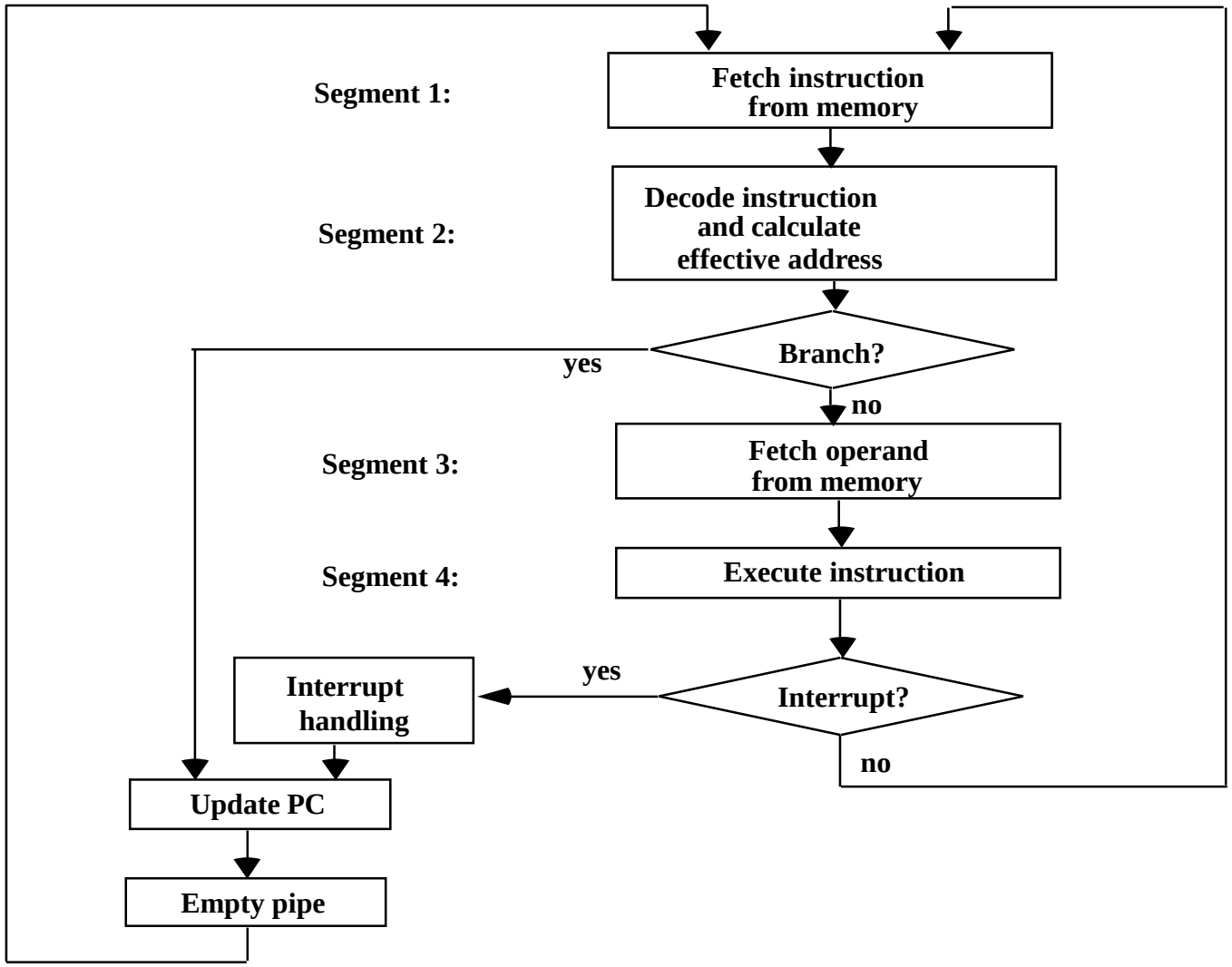


Figure : Four-segment CPU pipeline.

	Step:	1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction :	1	FI	DA	FO	EX									
	2		FI	DA	FO	EX								
(Branch)	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

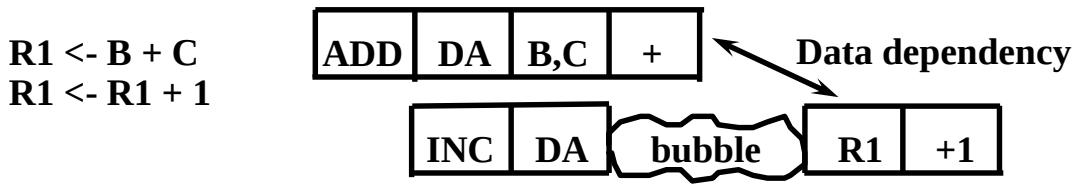
Figure : Timing of instruction pipeline.

Major hazards in pipelined execution

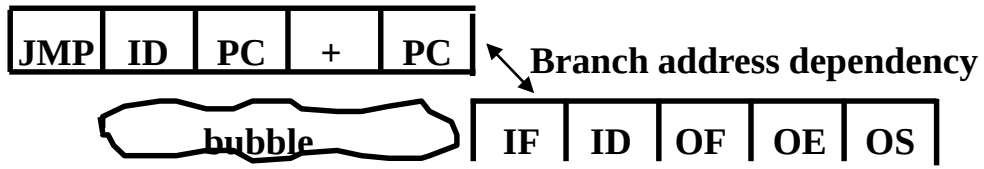
There are situations, called hazards, that prevent the next instruction in the instruction stream from executing during its designated cycle.

1. **Structural hazards (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.

2. **Data hazards (Data Dependency Conflicts)** conflicts arise when an instruction depends on the result of a previous instruction, but this result is not yet available.

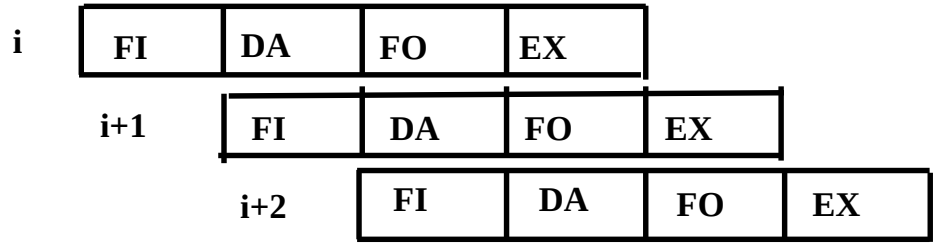


3. Control hazards(Branch Difficulties) arise branches and other instructions that change the PC make the fetch of the next instruction to be delayed.



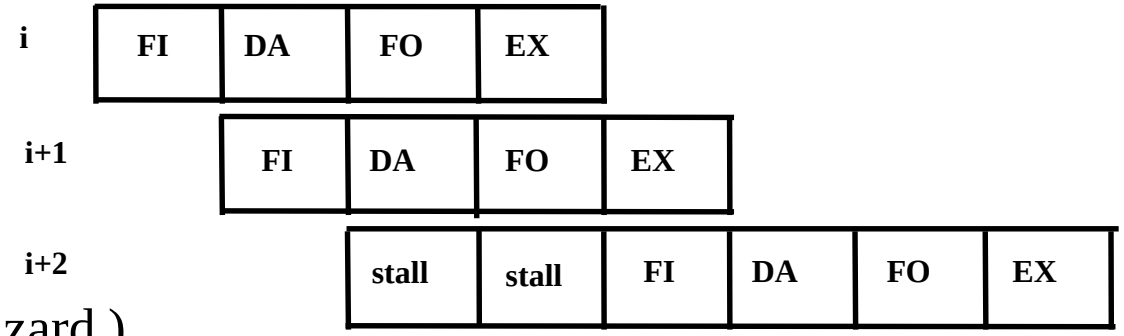
Structural hazards(Resource Conflicts)

- Occur when two instructions require a given hardware resource at the same time.
- Example: With one memory-port, a data and an instruction fetch cannot be initiated in the same clock.



Solution to Resource conflicts:

- Most of these conflicts can be resolved by using separate instruction and data memories.
- Use of pipeline stall (it is delay in the execution of an instruction in order to resolve a hazard).



Data hazards

- Data Hazards occurs when the execution of an instruction depends on the results of a previous instruction but this result is not yet available.

```
ADD     R1, R2, R3
SUB     R4, R1, R5
```

- Data hazard can be deal with either hardware techniques or software technique.
- Similarly, an address dependency may occur when an operand address cannot be calculated because the information needed by the addressing mode is not available.

Data Hazard Classification

Three types of data hazards

1. RAW : Read After Write

- A read after write (RAW) data hazard refers to a situation where an instruction refers to a result that has not yet been calculated or retrieved.
- This can occur because even though an instruction is executed after a previous instruction, the previous instruction has not been completely processed through the pipeline.
- Example: $I_{1.}$ R2 <- R1 + R3

  I_2. R4 <- R2+R3

2. Write After Read (WAR)

- A write after read (WAR) data hazard represents a problem with concurrent execution.
- Example : $I_{1.}$ R4<- R1+R5

  $I_{2.}$ R5<- R1 + R2

3. Write After Write (WAW)

- A write after write (WAW) data hazard may occur in a concurrent execution environment.
- We must delay the WB (Write Back) of i_2 until the execution of i_1
- Example: I_1. R2<-R4+R7

  I_2. R2 <-R1 +R3

Data hazard deals with

Hardware Technique

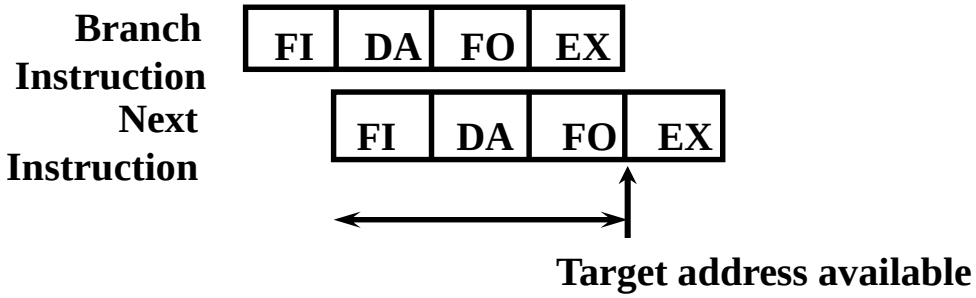
- **Hardware Interlock :** hardware detects the data dependencies and delays the scheduling of the dependent instruction by stalling enough clock cycles.
- **Operand Forwarding (bypassing, short-circuiting) :** uses special hardware to detect a conflict and then avoid it by routing the data through special paths between pipeline segments.
    - This method requires additional hardware paths through multiplexers as well as the circuit that detects the conflict.
    - For example, instead of transferring an ALU result into a destination register, the hardware checks the destination operand, and if it is needed as a source in the next instruction, it passes the result directly into the ALU input, bypassing the register file.

Software Technique

Delayed load: the compiler for such computers is designed to detect a data conflict and reorder the instructions as necessary to delay the loading of the conflicting data by inserting no-operation instructions.

Control hazards

- One of the major problems in operating an instruction pipeline is the occurrence of branch instructions.
  - An unconditional branch always alters the sequential program flow by loading the program counter with the target address.
  - In a conditional branch, the control selects the target instruction if the condition is satisfied or the next sequential instruction if the condition is not satisfied.
- Solution to Branch difficulties
  - **Prefetch Target Instruction**
  - **Branch Target Buffer**
  - **Loop Buffer**
  - **Branch Prediction**
  - **Delayed Branch**



- **Prefetch Target Instruction**
  - Fetch instructions in both streams, branch not taken and branch taken
  - Both are saved until branch branch is executed. Then, select the right instruction stream and discard the wrong stream
- **Branch Target Buffer (BTB ; Associative Memory)**
  - The BTB is an associative memory included in the fetch segment of the pipeline.
    - Each entry in the BTB consists of the address of a previously executed branch instruction and the target instruction for that branch.
    - It also stores the next few instructions after the branch target instruction
  - If found, fetch the instruction stream in BTB;
  - If not, new stream is fetched and update BTB
- **Loop Buffer(High Speed Register file)**
  - This is a small very high speed register file maintained by the instruction fetch segment of the pipeline.

- **Branch Prediction**
  - A pipeline with branch prediction uses some additional logic to guess the outcome of a conditional branch instruction before it is executed.
- **Delayed Branch**
  - In this procedure, compiler detects the branch and rearranges the instruction sequence by inserting useful instructions that keep the pipeline busy in the presence of a branch instruction.
    - A procedure employed in most RISC processors.
    - e.g. no-operation instruction

RISC Pipeline

RISC

- Machine with a very fast clock cycle that executes at the rate of one instruction per cycle
- Simple Instruction Set
- Fixed Length Instruction Format
- Register-to-Register Operations
- To use an efficient instruction pipeline
    - To implement an instruction pipeline using a small number of suboperations, with each being executed in one clock cycle.
    - Because of the fixed-length instruction format, the decoding of the operation can occur at the same time as the register selection.
    - Therefore, the instruction pipeline can be implemented with two or three segments.
        - One segment fetches the instruction from program memory
        - The other segment executes the instruction in the ALU
        - Third segment may be used to store the result of the ALU operation in a destination register

- The data transfer instructions in RISC are limited to load and store instructions.
  - These instructions use register indirect addressing. They usually need three or four stages in the pipeline.
  - To prevent conflicts between a memory access to fetch an instruction and to load or store an operand, most RISC machines use two separate buses with two memories.
  - Cache memory: operate at the same speed as the CPU clock
- One of the major advantages of RISC is its ability to execute instructions at the rate of one per clock cycle.
  - In effect, it is to start each instruction with each clock cycle and to pipeline the processor to achieve the goal of single-cycle instruction execution.
  - RISC can achieve pipeline segments, requiring just one clock cycle.
- Compiler supported that translates the high-level language program into machine language program.
  - Instead of designing hardware to handle the difficulties associated with data conflicts and branch penalties.
  - RISC processors rely on the efficiency of the compiler to detect and minimize the delays encountered with these problems.

Example: Three-Segment Instruction Pipeline

- The processor unit consists of a number of registers and an arithmetic logic unit (ALU).
- A data memory is used to load or store the data from a selected register in the register file

Data Manipulation Instructions

I: Instruction Fetch

A: Decode, Read Registers, ALU Operations

E: Write a Register

Load and Store Instructions

I: Instruction Fetch

A: Decode, Evaluate Effective Address

E: Register-to-Memory or Memory-to-Register

Program Control Instructions

I: Instruction Fetch

A: Decode, Evaluate Branch Address

E: Write Register(PC)

Delayed Load

- Consider the operation of the following four instructions:

  LOAD: R1 ← M[address 1]

  LOAD: R2 ← M[address 2]

  ADD: R3 ← R1 + R2

  STORE: M[address 3] ← R3

- There will be a data conflict in instruction 3 because the operand in R2 is not yet available in the A segment.
- This can be seen from the timing of the pipeline shown in Figure (a).
  - The E segment in clock cycle 4 is in a process of placing the memory data into R2.
  - The A segment in clock cycle 4 is using the data from R2.
- It is up to the compiler to make sure that the instruction following the load instruction uses the data fetched from memory.

- This concept of delaying the use of the data loaded from memory is referred to as delayed load.
- Figure (b) shows the same program with a no-op instruction inserted after the load to R2 instruction.
- Thus the no-op instruction is used to advance one clock cycle in order to compensate for the data conflict in the pipeline.
- The advantage of the delayed load approach is that the data dependency is taken care of by the compiler rather than the hardware .

Clock cycles:	1	2	3	4	5	6
1. Load R1	I	A	E			
2. Load R2		I	A	E		
3. Add R1+R2			I	A	E	
4. Store R3				I	A	E

Figure (a): Three segment pipeline timing - Pipeline timing with data conflict.

Clock cycle:	1	2	3	4	5	6	7
1. Load R1	I	A	E				
2. Load R2		I	A	E			
3. No-operation			I	A	E		
4. Add R1+R2				I	A	E	
5. Store R3					I	A	E

Figure (b): Three segment pipeline timing - Pipeline timing with delayed load.

Delayed Branch

- The method used in most RISC processors is to rely on the compiler to redefine the branches so that they take effect at the proper time in the pipeline. This method is referred to as delayed branch.
- The compiler is designed to analyze the instructions before and after the branch and rearrange the program sequence by inserting useful instructions in the delay steps.
- It is up to the compiler to find useful instructions to put after the branch instruction. Failing that, the compiler can insert no-op instructions.
- **An Example of Delayed Branch**
  - The program for this example consists of five instructions.
    - Load from memory to R1
    - Increment R2
    - Add R3 to R4
    - Subtract R5 from R6
    - Branch to address X

- In Figure(a) the compiler inserts two no-op instructions after the branch.
  - The branch address X is transferred to PC in clock cycle 7 .
- The program in Figure(b) is rearranged by placing the add and subtract instructions after the branch instruction.
  - PC is updated to the value of X in clock cycle 5.

Clock cycles:	1	2	3	4	5	6	7	8	9	10
1. Load	I	A	E							
2. Increment		I	A	E						
3. Add			I	A	E					
4. Subtract				I	A	E				
5. Branch to X					I	A	E			
6. NOP						I	A	E		
7. NOP							I	A	E	
8. Instr. in X								I	A	E

Figure (a): Using no operation instruction.

Clock cycles:	1	2	3	4	5	6	7	8
1. Load	I	A	E					
2. Increment		I	A	E				
3. Branch to X			I	A	E			
4. Add				I	A	E		
5. Subtract					I	A	E	
6. Instr. in X						I	A	E

Figure (b): Rearranging the instructions

Vector processing

- In many science and engineering applications, the problems can be formulated in terms of vectors and matrices that lend themselves to vector processing.
- Vector processing as the process of using vectors to store a large number of variables for high-intensity data processing.
- Computers with vector processing capabilities are in demand in specialized applications. e.g
  - 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 Processor (computer)

- Ability to process vectors, and related data structures such as matrices and multi-dimensional arrays, much faster than conventional computers.
- Vector Processors may also be pipelined.
- To achieve the required level of high performance it is necessary to utilize the fastest and most reliable hardware and apply innovative procedures from vector and parallel processing techniques.

Vector Operations

- Many scientific problems require arithmetic operations on large arrays of 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,…,Vn].
- 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 implemented in machine language by the following sequence of operations.

```
        Initialize I=0
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
```

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

Figure : Instruction format for vector processor

- 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. C(1:100) = A(1:100) + B(1:100)
- A possible instruction format for a vector instruction is shown in Figure.
  - This assumes that the vector operands reside in *memory*.
- It is also possible to design the processor with a large number of *registers* and store all operands in registers prior to the addition operation.
  - The base address and length in the vector instruction specify a group of CPU registers.

Matrix Multiplication

- Matrix multiplication is one of the most computational intensive operations performed in computers with vector processors.
- The multiplication of two n x n matrices consists of n^2 inner products or n^3 multiply-add operations.
  - Consider, for example, the multiplication of two 3 x 3 matrices A and B.
  - $c_{11} = a_{11}\, b_{11} + a_{12}\, b_{21} + a_{13}\, b_{31}$, i.e. Inner product : $c_{ij} = \sum_{k=1}^{3} a_{ik} * b_{kj}$
  - This requires three multiplication and (after initializing c_{11} to 0) three additions.
- An n x m matrix of numbers has n rows and m columns and may be considered as constituting a set of n row vectors or a set of m column vectors. Consider, for example, the multiplication of two 3 x 3 matrices A and B .

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

- In general, the inner product consists of the sum of *k* product terms of the form $C=A_1B_1 +A_2 B_2 +A_3 B_3+\ldots+A_k B_k$.
- In a typical application k may be equal to 100 or even 1000.
- The inner product calculation on a pipeline vector processor is shown in Figure.

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

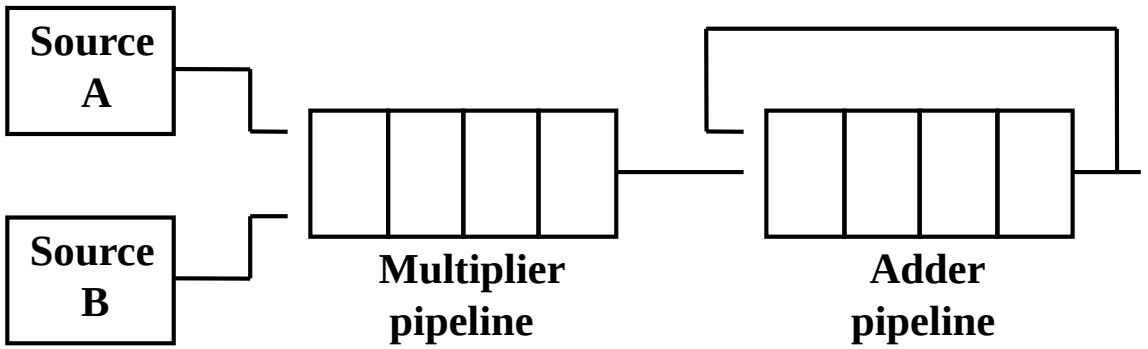


Figure: Pipeline for calculating an inner product

Memory Interleaving

- Pipeline and vector processors often require simultaneous access to memory from two or more sources.
  - An instruction pipeline may require the fetching of an instruction and an operand at the same time from two different segments.
  - An arithmetic pipeline usually requires two or more operands to enter the pipeline at the same time.
- Instead of using two memory buses for simultaneous access, the memory can be partitioned into a number of modules connected to a common memory address and data buses.
  - A memory module is a memory array together with its own address and data registers.
- Figure shows a memory unit with four modules.
- The advantage of a modular memory is that it allows the use of a technique called interleaving.
- In an interleaved memory, different sets of addresses are assigned to different memory modules.
- By staggering the memory access, the effective memory cycle time can be reduced by a factor close to the number of modules.

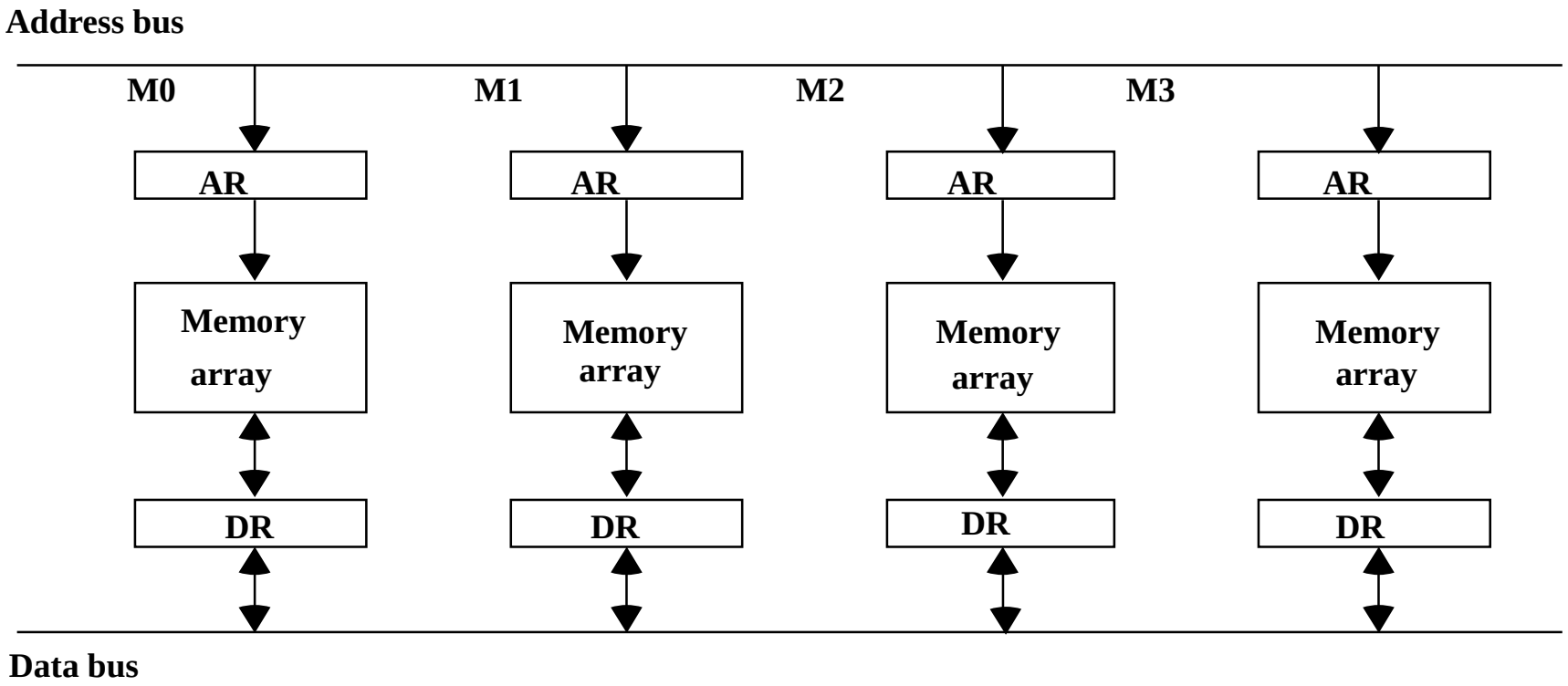


Figure: Multiple module memory organization

Supercomputers

- A commercial computer with vector instructions and pipelines floating-point arithmetic operations is referred to as a supercomputer.
- Supercomputers are very powerful. High-performance machines and used mostly for scientific computations.
- To speed up the operations, the components are packed tightly together to minimize the distance that the electronic signals have to travel.
- Supercomputers also use special techniques for removing the heat from circuits to prevent then from burning up because of their close proximity.
- The instruction set of supercomputer contains the standard data transfer, data manipulation, and program control instructions.
- A supercomputer is a computer system best known for its high computational speed, fast and large memory systems, and the extensive use of parallel processing.

- The measure used to evaluate computers in their ability to perform a given number of floating-point operations per second is referred to as *flops*.
- The term *megaflops* is used to denote million flops and *gigaflops* to denote billion flops.
- Typical supercomputer has a basic cycle time of 4-20 ns.
- If the processor can calculate a floating-point operations through a pipeline each cycle time, it will have the ability to perform 50 to 250 megaflops.

- The first supercomputer developed in 1976 is the Cray-1 supercomputer.
  - It uses vector processing with 12 distinct functional units in parallel.
  - Each functional unit is segmented to process the data through pipeline.
  - A floating-point operation can be performed on two set of 64-bit operands during one clock cycle of 12.5 ns.
  - This gives a rate of 80 megaflops
  - It has a memory capacity of 4 millions 64-bit words.
  - The memory is divided into 16 banks, with each bank having 50-ns access time.
  - This means that when all 16 banks are accessed simultaneously, the memory transfer rate is 320 million words per second.
  - Later version are Cray X-MP, Cray Y-MP, Cray-2 (12 times powerful that the Cray-1)
  - Another supercomputers are Fujitsu VP-200, VP-2600, PARAM Computers.

Array Processors

- An array processor is a processor that performs computations on large arrays of data.
- The term is used to refer to two different types of processors.
  - An attached array processor
  - An SIMD array processor

An attached array processor :

- 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.

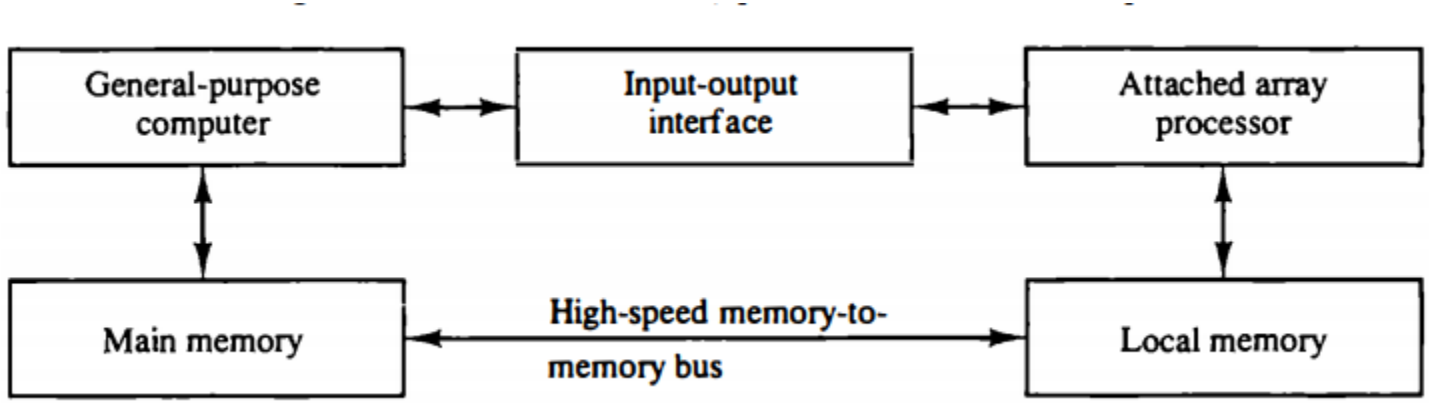


Figure : Attached array processor with host computer.

SIMD array processor :

- 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.

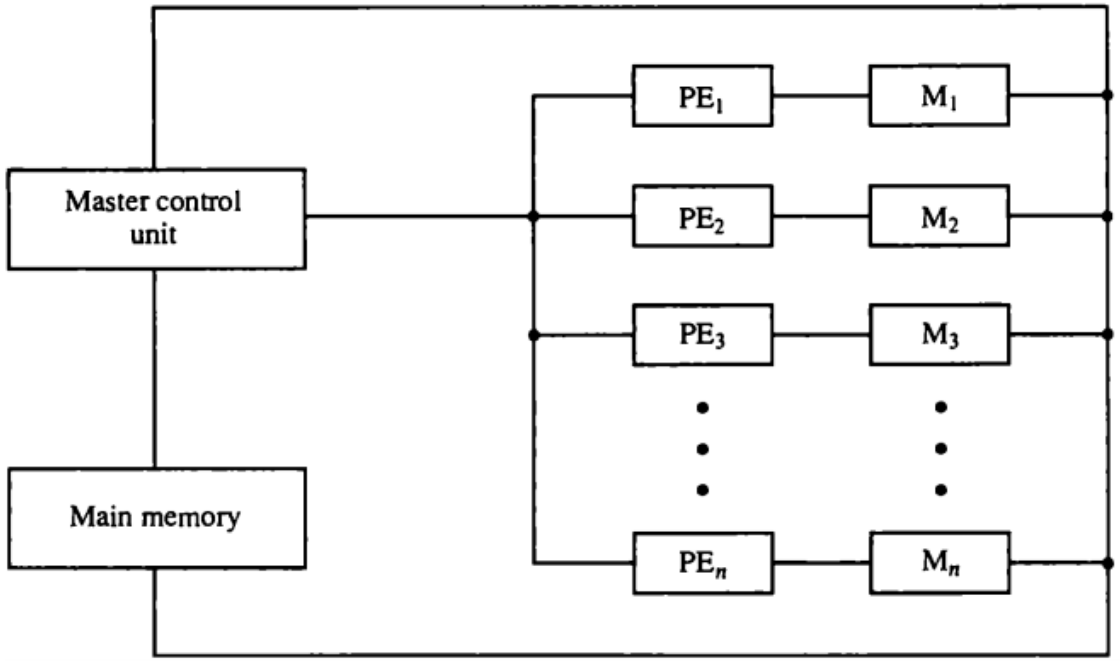


Figure : SIMD array processor organization

- 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.
- For example, the vector addition C = A + B. The master control unit first stores the i^{th} components a, and b, of A and B in local memory M_i for i = 1, 2, 3, . . . , n. It then broadcasts the floating-point add instruction $c_i = a_i + b_i$ to all PEs, causing the addition to take place simultaneously. The components of c_i are stored in fixed locations in each local memory. This produces the desired vector sum in one add cycle
- The best known SIMD array processor is the **ILLIAC IV** computer developed by the **Burroughs corps**. SIMD processors are highly specialized computers.
- only suitable for numerical problems that can be expressed in vector or matrix form and they are not suitable for other types of computations.