Course Contents

Unit-01: Introduction to Microprocessor

Unit-02: Intel 8085

Unit-03: Microoperations

Unit-04: Control Unit and Central Processing Unit

Unit-05: Fixed point Computer Arithmetic

Unit-06: Input and Output Organization

Unit-07: Memory Organization

Unit-08: Pipelining

Parallel Processing & Pipelining

- Concept of Pipelining and Flynn's Classification, Pipelining Example with Speed Up Ratio
- Arithmetic Pipeline: Pipeline for Floating-point Addition and Subtraction
- Instruction Pipeline: Four Segment Instruction Pipeline
- Data Dependency, Handling of Branch Instruction

Parallel Processing & Pipelining

Unit 8	Pipelining	4 Hours
8.1	Concept of Pipelining and Flynn's Classification, Pipelining Example with Speed Up Ratio	1 Hour
8.2	Arithmetic Pipeline , Pipeline for Floating-point Addition and Subtraction	1 Hour
8.3	Instruction Pipeline: Four Segment Instruction Pipeline	1 Hour
8.4	Data Dependency, Handling of Branch Instruction	1 Hour

Parallel Processing & Pipelining

Questions:

1. What is parallel processing? Explain the benefits of parallel processing.
2. Explain the classifications of the organization of a computer by M. J. Flynn.
3. What is pipelining? Explain the role of pipelining in computing.
4. Define instruction pipeline. Explain the four segment instruction pipeline.
5. Explain the Data Dependency in Pipelining?
6. How to handle the branch instruction in pipeline? Explain.
7. Write short notes on: Arithmetic pipeline.

Parallel Processing & Pipelining

Parallel Processing:

- Parallel processing is a 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.

Parallel Processing & Pipelining

Parallel Processing:

Parallel processing can be viewed as various levels of complexity.

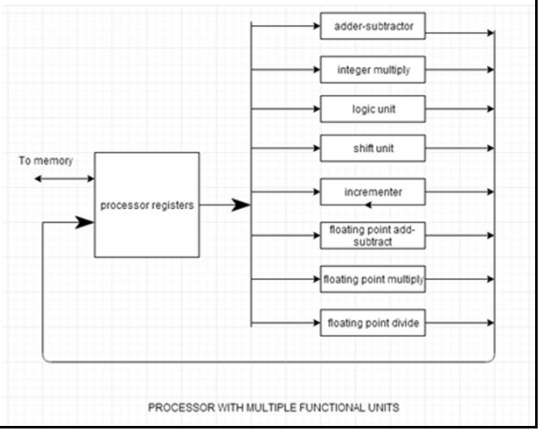
1. At the lowest level, we distinguish between parallel and serial operations by the type of registers used. Shift registers operate in serial fashion one bit at a time, while registers with parallel load operate with all the bits of the word simultaneously.
2. Parallel processing at a higher level of complexity can be achieved by having a multiplicity of functional units that perform identical or different operations simultaneously.

Parallel Processing & Pipelining

Parallel Processing:

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.



PROCESSOR WITH MULTIPLE FUNCTIONAL UNITS

Flynn's Classification

Flynn's Classification

Flynn's Classification

Classification of Computers by M. J. Flynn:

The normal operation of a computer is to fetch instructions from memory and execute them in the processor. The sequence of instructions read from memory constitutes an instruction stream. The operations performed on the data in the processor constitutes a data stream. Parallel processing may occur in the instruction stream, in the data stream, or in both.

Flynn's Classification

Classification of Computers by M. J. Flynn:

Understanding few terms used above:

- **Stream**→ Stream refers to sequence of objects (may be data or instructions) that is executed by a single CPU.
- **Instruction Stream**→ Instruction stream refers to sequence of instructions that is executed by the CPU.
- **Data Stream**→ Data stream refers to the sequence of data including input, partial or temporary results called for by instruction stream.

M. J. Flynn classified the organization of a computer system by the number of instructions and data items that are manipulated simultaneously.

Flynn’s Classification

Classification of Computers by M. J. Flynn:

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)

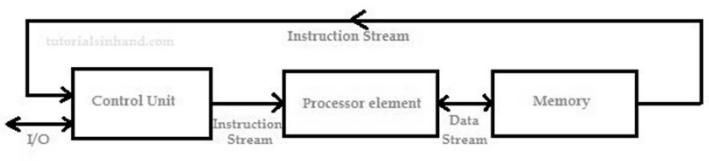
Flynn’s Classification

Classification of Computers by M. J. Flynn:

- **Single Instruction Stream, Single Data Stream (SISD) -**

It represents the organization of a single computer system containing a control unit, processor unit and a memory unit. Instructions are executed sequentially. Parallel operations can be achieved by pipelining or multiple functional units.

SISD computer may have more than one functional unit in them, but all are under supervision of one control unit. SISD architecture computers can process only scalar type instructions. Modern day computers are SISD.



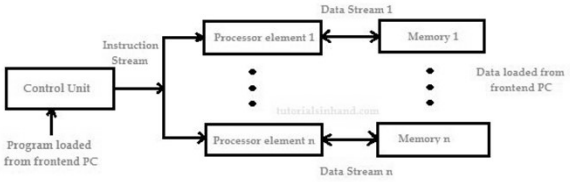
SISD Computer

Flynn's Classification

Classification of Computers by M. J. Flynn:

- **Single instruction stream, multiple data stream (SIMD)**

It represents an organization that includes multiple processing units under the control of a common control unit. All processing units(processors) receive the same instruction from control unit but operate on different parts of the data. They are highly specialized computers. They are basically used for numerical problems that are expressed in the form of vector or matrix. But they are not suitable for other types of computations.



SIMD Computer

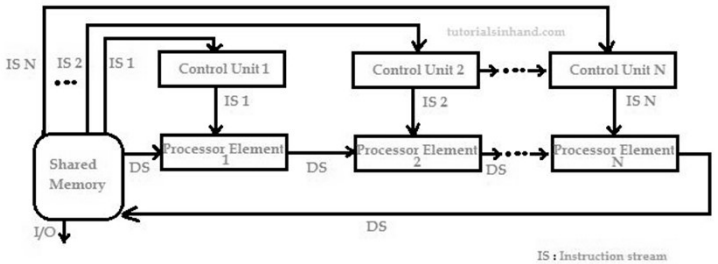
Flynn's Classification

Classification of Computers by M. J. Flynn:

- **Multiple Instruction Stream, Single Data Stream (MISD)**

In *MISD computer architecture*, there are n processor elements. Each processor elements receives distinct instructions to execute on the same data stream and its derivatives. Here the output of one processor element becomes the input of the next processor element in the series.

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



MISD Computer

Flynn's Classification

Classification of Computers by M. J. Flynn:

- **Multiple instruction stream, multiple data stream (MIMD) –**

MIMD computer category covers multiple computer system and multiprocessor systems. *MIMD computer* is of two types:

a) tightly coupled or Uniform Memory Access (UMA)

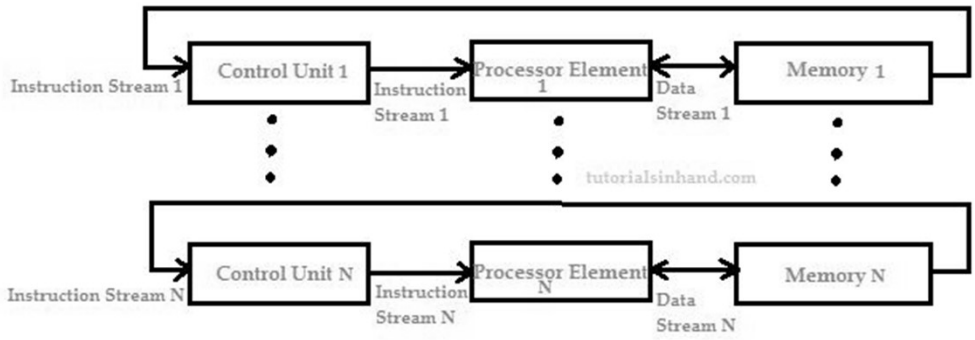
b) loosely coupled or Non-Uniform Memory Access (NUMA)

MIMD computer is called **tightly coupled or Uniform Memory Access (UMA)** if the degree of interaction among the processor is high. MIMD computer is called **loosely coupled or Non-Uniform Memory Access (NUMA)** if the degree of interaction among processors is low.

Flynn's Classification

Classification of Computers by M. J. Flynn:

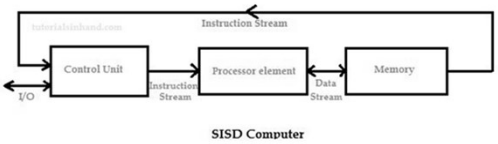
- **Multiple instruction stream, multiple data stream (MIMD) –**



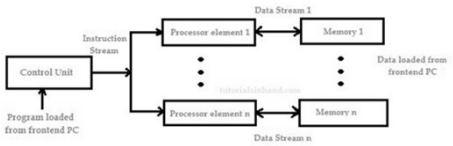
MIMD Computer

Flynn's Classification

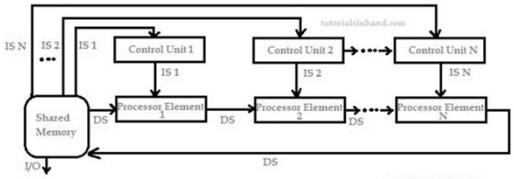
Classification of Computers by M. J. Flynn:



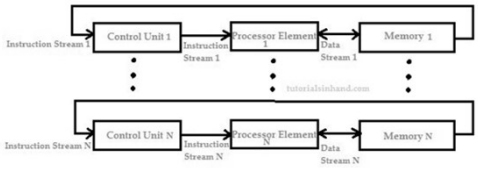
SISD Computer



SIMD Computer



MISD Computer



MIMD Computer

Concept of Pipelining

Concept of Pipelining:

Parallel Processing & Pipelining

Parallel Processing:

Parallel processing can be achieved through following methods:

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

Parallel Processing & Pipelining

Pipelining:

- Pipeline processing is an implementation technique where arithmetic sub-operations or the phases of a computer instruction cycle overlap in execution.
- Vector processing deals with computations involving large vectors and matrices.
- Array processors perform computations on large arrays of data.

Parallel Processing & Pipelining

Pipelining:

Pipelining is a technique of *decomposing a sequential process into suboperations,* with each sub-process being executed in a special dedicated segment that operates *concurrently* with all other segments.

There are two areas of computer design where the pipeline organization is applicable.

- Arithmetic pipeline
- Instruction pipeline

Parallel Processing & Pipelining

Pipelining: Arithmetic Pipeline

The pipeline organization will be demonstrated by means of a simple example. To perform the combined multiply and add operations with a stream of numbers

$A_i * B_i + C_i$ for $i = 1, 2, 3, \ldots, 7$

- Each suboperation is to be implemented in a segment within a pipeline.

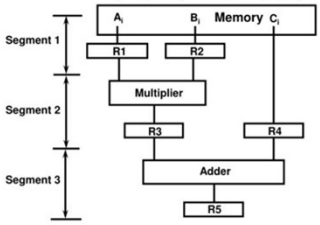
$R1 \leftarrow A_i, R2 \leftarrow B_i$	Input A_i and B_i
$R3 \leftarrow R1 * R2, R4 \leftarrow C_i$	Multiply and input C_i
$R5 \leftarrow R3 + R4$	Add C_i to product

Parallel Processing & Pipelining

Pipelining:

A technique of decomposing a sequential process into suboperations, with each subprocess being executed in a partial dedicated segment that operates concurrently with all other segments.

$A_i * B_i + C_i$ for i = 1, 2, 3, ... , 7



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

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

Table 4-1: Content of Registers in Pipeline Example

Parallel Processing & Pipelining

Pipelining: Arithmetic Pipeline

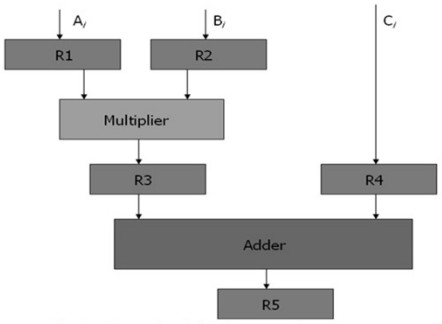


Fig 4-1: Example of pipeline processing

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

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

Table 4-1: Content of Registers in Pipeline Example

Parallel Processing & Pipelining

Pipelining: Arithmetic Pipeline

Conclusion:

Any operation that can be decomposed into a sequence of suboperations of about the same complexity can be implemented by a pipeline processor. The technique is efficient for those applications that need to repeat the same task many times with different sets of data.

Instruction Pipeline

Four Segment Instruction Pipeline

Instruction Pipeline

Pipelining: Four segment instruction pipeline

Computers with complex instructions require other phases in addition to the fetch and execute to process an instruction completely.

In the most general case, the computer needs to process each instruction with the following sequence of steps.

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.

Instruction Pipeline

Pipelining: 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.

FI	:	the segment that fetches an instruction
DA	:	the segment that decodes the instruction and calculate the effective address
FO	:	the segment that fetches the operand
EX	:	the segment that executes the instruction

So, it is called four segment instruction pipeline.

Instruction Pipeline

Pipelining: Four segment instruction pipeline

The normal operation can be decomposed into **four segment** as below. So, it is called four segment pipeline.

FI : the segment that fetches an instruction
DA : the segment that decodes the instruction and calculate the effective address
FO : the segment that fetches the operand
EX : the segment that executes the instruction

Thus up to four suboperations in the instruction cycle can overlap and up to four different instructions can be in progress of being processed at the same time.

Instruction Pipeline

Pipelining: Four segment instruction pipeline

Fig. 9-8 shows the operation of the instruction pipeline.

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

Fig 4-8: Timing of Instruction Pipeline

FI: the segment that fetches an instruction

DA: the segment that decodes the instruction and calculate the effective address

FO: the segment that fetches the operand

EX: the segment that executes the instruction

Instruction Pipeline

Pipelining: Four segment instruction pipeline

Figure shows how the instruction cycle in the CPU can be processed with a four-segment pipeline.
While an instruction is being executed in segment 4, the next instruction in sequence is busy fetching an operand from memory in segment 3.
The effective address may be calculated in a separate arithmetic circuit for the third instruction, and whenever the memory is available, the fourth and all subsequent instructions can be fetched and placed in an instruction FIFO.
Thus up to four suboperations in the instruction cycle can overlap and up to four different instructions can be in progress of being processed at the same time.

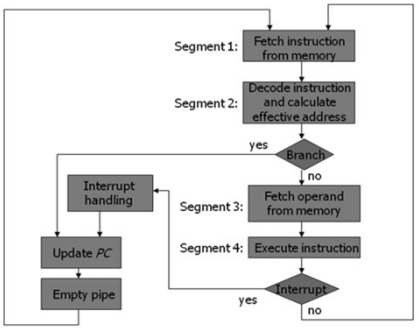


Fig 4-7: Four-segment CPU pipeline

Instruction Pipeline

Pipelining: Four segment instruction pipeline

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

- Thus up to four suboperations in the instruction cycle can overlap and up to four different instructions can be in progress of being processed at the same time.
- An instruction in the sequence may be causes a branch out of normal sequence.
  - In that case the pending operations in the last two segments are completed and all information stored in the instruction buffer is deleted.
  - Similarly, an interrupt request will cause the pipeline to empty and start again from a new address value.

Instruction Pipeline

Pipelining: Four segment instruction pipeline

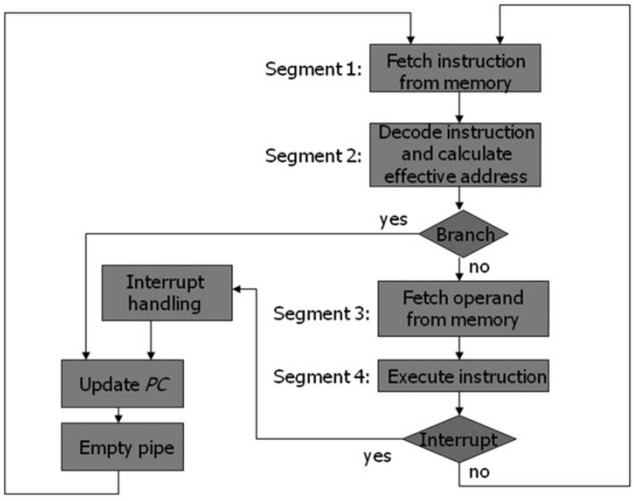


Fig 4-7: Four-segment CPU pipeline

Instruction Pipeline

Pipelining: Four segment instruction pipeline

Fig. 9-8 shows the operation of the instruction pipeline.

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

Fig 4-8: Timing of Instruction Pipeline

FI: the segment that fetches an instruction

DA: the segment that decodes the instruction and calculate the effective address

FO: the segment that fetches the operand

EX: the segment that executes the instruction

Pipelining Example with Speed up Ratio

Performance of a pipelined processor:

Figure 9-4 Space-time diagram for pipeline.

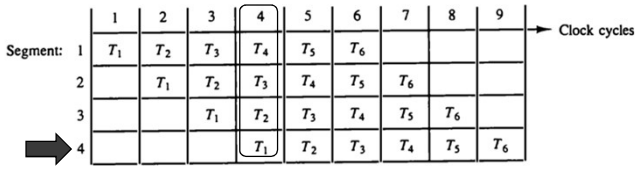


TABLE 9-1 Content of Registers in Pipeline Example

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

In space time diagram for pipeline, the horizontal axis displays the time in clock cycles and the vertical axis gives the segment number.
The first task T_1 is completed after the fourth clock cycle. From then on, the pipe completes a task every clock cycle. No matter how many segments there are in the system, once the pipeline is full, it takes only one clock period to obtain an output.

Pipelining Example with Speed up Ratio

Performance of a pipelined processor:

Consider a ‘k’ segment pipeline with clock cycle time as ‘t_p’. Let there be ‘n’ tasks to be completed in the pipelined processor. Now, the first instruction is going to take ‘k’ cycles to come out of the pipeline but the other ‘n – 1’ instructions will take only ‘1’ cycle each, i.e, a total of ‘n – 1’ cycles to complete n-1 instructions.

So, now consider the case where a k-segment pipeline with a clock cycle time t, is used to execute n tasks. The first task T_1 requires a time equal to kt, to complete its operation since there are k segments in the pipe. The remaining (n-1) tasks emerge from the pipe at the rate of one task per clock cycle and they will be completed after a time equal to (n - 1)t,.

Therefore, to complete n tasks using a k-segment pipeline requires k + (n - 1) clock cycles.

Pipelining Example with Speed up Ratio

Performance of a pipelined processor:

For example, the diagram of Fig. 9-4 shows four segments and six tasks. The time required to complete all the operations is 4 + (6 - 1) = 9 clock cycles, as indicated in the diagram.

Next consider a non-pipeline unit that performs the same operation and takes a time equal to t_n to complete each task. The total time required for n tasks is nt_n.

The **speedup** of a pipeline processing over an equivalent non-pipeline processing is defined by the ratio

$$S = \frac{nt_n}{(k + n - 1)t_p}$$

Where t_p is a clock cycle time of 'k' segment pipeline.

As the number of tasks increases, n becomes much larger than (k – 1), and (k+ n–1) approaches the value of n. Under this condition, the speedup becomes

$$S = \frac{t_n}{t_p}$$

Pipelining Example with Speed up Ratio

Performance of a pipelined processor:

If we assume that the time it takes to process a task is the same in the pipeline and non pipeline circuits, we will have $t_n = kt_p$. Including this assumption, the speedup reduces to

$$S = \frac{kt_p}{t_p} = k$$

This shows that the theoretical maximum speedup that a pipeline can provide is k, where k is the number of segments in the pipeline.

Pipelining Example with Speed up Ratio

Performance of a pipelined processor:

To check the speedup ratio, consider the following numerical example.

- Let the time it takes to process a suboperation in each segment be equal to t_p = 20 ns. Assume that the pipeline has k = 4 segments and executes n = 100 tasks in sequence. The pipeline system will take $(k + n - 1)t_p = (4 + 99)$ x 20 = 2060ns to complete.

- Assuming that $t_n = kt_p = 4$ x 20 = 80 ns, a non-pipeline system requires nkt_p= 100 x 80 = 8000ns to complete the 100 tasks.

- The speedup ratio is equal to 8000/2060 = 3.88. As the number of tasks increases, the speedup will approach 4, which is equal to the number of segments in the pipeline. If we assume that t_n = 60 ns, the speedup becomes 60/20 = 3.

Arithmetic Pipeline

Pipeline for Floating-point Addition and Subtraction:

- An arithmetic pipeline separates a given arithmetic problem into suboperations that can be executed in different pipeline segments. Arithmetic Pipelines are commonly used in various high-performance computers. They are used in order to implement floating-point operations, fixed-point multiplication, and other similar kinds of calculations that come up in scientific situations.

Let's look at an example to better understand the ideas of an arithmetic pipeline. We perform addition and subtraction of floating points on a unit of the pipeline.

Arithmetic Pipeline

Pipeline for Floating-point Addition and Subtraction:

Floating-point operations are easily decomposed into suboperations. We will now show an example of a pipeline unit for floating-point addition and subtraction.

The inputs in the floating-point adder pipeline refer to two different normalized floating-point binary numbers. These are defined as follows:

$X=A \times 2^a$ $\quad\quad$ $X = 0.9504 \times 10^3$

$Y=B \times 2^b$ $\quad\quad$ $Y = 0.8200 \times 10^2$

Where a and b refer to the exponents and A and B refer to two fractions representing the mantissa.

Arithmetic Pipeline

Pipeline for Floating-point Addition and Subtraction:

The floating-point addition and subtraction process is broken into four suboperations. The matching sub-operation to be executed in the specified pipeline is contained in each segment. The suboperations that are performed in the four segments are:

1. Comparing the exponents using subtraction
2. Aligning the mantissa
3. Adding or subtracting the mantissa
4. Normalizing the result

The sub-operations conducted in each pipeline segment are depicted in the block diagram below:

Arithmetic Pipeline

Pipeline for Floating-point Addition and Subtraction:

1. Comparing Exponents by Subtraction

The exponents are compared by subtracting them to determine their difference. The larger exponent is chosen as the exponent of the result. The exponent difference determines how many times the mantissa associated with the smaller exponent must be shifted to the right. This produces an alignment of the two mantissas.

The exponent difference, 3 – 2 = 1, defines the total number of times the mantissa associated with the lesser exponent should be shifted to the right.

$X = 0.9504 \times 10^3$

$Y = 0.8200 \times 10^2$

Arithmetic Pipeline

Pipeline for Floating-point Addition and Subtraction:

2. Aligning the Mantissa

As per the difference of exponents calculated in segment one, the mantissa corresponding with the smaller exponent would be moved (i.e the mantissa associated with the smaller exponent must be shifted to the right)

$X = 0.9504 \times 10^3$

$Y = 0.8200 \times 10^2$

$X = 0.9504 \times 10^3$

$Y = 0.0820 \times 10^3$

Arithmetic Pipeline

Pipeline for Floating-point Addition and Subtraction:

3. Adding the Mantissa

Both the mantissa would be added in the third segment.

$X = 0.9504 \times 10^3$

$Y = 0.0820 \times 10^3$

The addition of the two mantissas in segment 3 produces the sum

$z = x + y$

$Z = 1.0324 \times 10^3$

Arithmetic Pipeline

Pipeline for Floating-point Addition and Subtraction:

4. Normalizing the Result

After the process of normalization, the result would be written as follows:

$Z = 1.0324 \times 10^3$

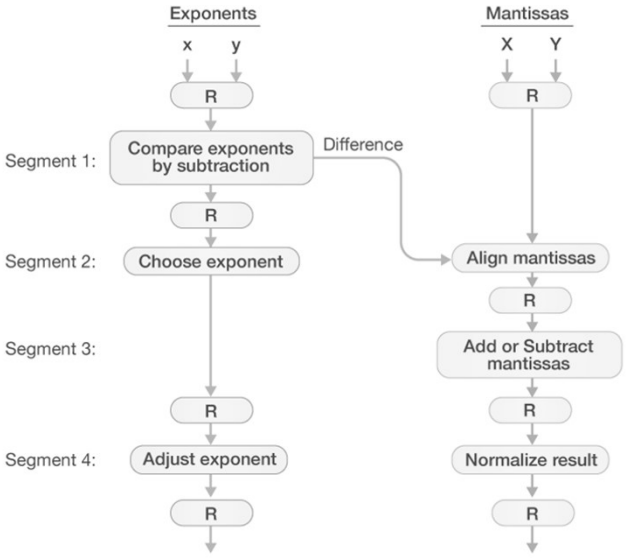
(After Normalization)

$Z = 0.10324 \times 10^4$

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

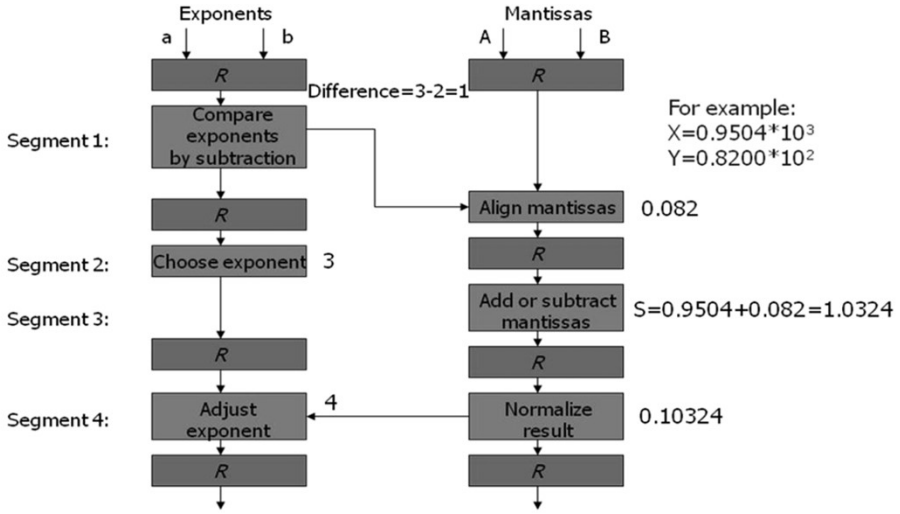
Arithmetic Pipeline

Pipeline for Floating-point Addition and Subtraction:



Arithmetic Pipeline

Pipeline for Floating-point Addition and Subtraction:



Arithmetic Pipeline

Pipeline for Floating-point Addition and Subtraction:

The comparator, shifter, adder-subtractor, incrementer, and decrementer in the floating-point pipeline are implemented with combinational circuits.

Suppose that the time delays of the four segments are t_1 = 60 ns, t_2 = 70 ns, t_3 = 100 ns, t_4 = 80 ns, and the interface registers have a delay of t_r = 10 ns.

The clock cycle is chosen to be $t_p = t_3 + t_r$ = 110 ns.

An equivalent non-pipeline floating point adder-subtractor will have a delay time $t_n = t_1 + t_2 + t_3 + t_4 + t_r$ = 320 ns.

In this case the pipelined adder has a speedup of 320/110 = 2. 9 over the non-pipelined adder.

(i.e. From above performance formula of pipelined, the speedup of pipelined becomes)

$$S = \frac{t_n}{t_p}$$

Arithmetic Pipeline

Pipeline for Floating-point Addition and Subtraction:

Problem: The time delay of the four segments in the arithmetic floating point pipeline are as follows: t_1 =50ns, t_2= 30ns, t_3= 95ns and t_4= 45ns. The interface registers delay time t_p= 5ns

a) How long would it take to add 100 pairs of numbers in the pipeline?

b) Calculate the speedup of pipelined operation.

Data Dependency

Data Dependency:

In pipeline operation, a data dependency occurs when an instruction (suboperations) needs data that are not yet available. A difficulty that may caused a degradation of performance in an instruction pipeline is due to possible collision of data or address. A collision occurs when an instruction cannot proceed because previous instructions did not complete certain operations.

Fig. 9-8 shows the operation of the instruction pipeline.

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

Fig 4-8: Timing of Instruction Pipeline

Data Dependency

Data Dependency

For example, an instruction in the FO segment may need to fetch an operand that is being generated at the same time by the previous instruction in segment EX. Therefore, the second instruction must wait for data to become available by the first instruction.

Similarly, an address dependency may occur when an operand address cannot be calculated because the information needed by the addressing mode is not available. For example, an instruction with register indirect mode cannot proceed to fetch the operand if the previous instruction is loading the address into the register. Therefore, the operand access to memory must be delayed until the required address is available.

Pipelined computers deal with such conflicts between data dependencies in a variety of ways.

Data Dependency

Data Dependency

There are mainly three types of dependencies possible in a pipelined:

1. Hardware interlock
2. Operand forwarding
3. Delayed load

Hardware interlock- An interlock is a circuit that detects instructions whose source operands are destinations of instructions farther up in the pipeline. Detection of this situation causes the instruction whose source is not available to be delayed by enough clock cycles to resolve the conflict. This approach maintains the program sequence by using hardware to insert the required delays.

Data Dependency

Data Dependency

Operand forwarding - Operand forwarding uses special hardware to detect a conflict and then avoid it by routing the data through special paths between pipeline segments. 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. This method requires additional hardware paths through multiplexers as well as the circuit that detects the conflict.

Delayed load- The compiler 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. This method is referred to as delayed load.

Handling of Branch Instruction

Handling of Branch Instruction

The branch instruction(can be conditional or unconditional) breaks the normal sequence of the instruction stream, causing difficulties in the operation of the instruction pipeline.

- 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.

Pipelined computers employ various hardware techniques to minimize the performance degradation caused by instruction branching.

Handling of Branch Instruction

Handling of Branch Instruction

Techniques are:

1. *Prefetch target instruction*
2. *Branch target buffer(BTB)*
3. *Loop Buffer*
4. *Branch prediction*
5. *Delayed branch*

Prefetch target instruction: To prefetch the target instruction in addition to the instruction following the branch. Both are saved until the branch is executed.

Branch target buffer(BTB): 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.

Handling of Branch Instruction

Handling of Branch Instruction

Loop Buffer: 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, the compiler detects the branch instructions and re-arranges the machine language code sequence by inserting useful instructions that keep the pipeline operating without interruptions.

- A procedure employed in most RISC processors.
- e.g. no-operation instruction