1. Explain Flynn's classification for computers.

Flynn's classification

- It is based on the multiplicity of Instruction Streams and Data Streams

Instruction Stream

Sequence of Instructions read from memory

Data Stream

Operations performed on the data in the processor

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

Figure 6.1: Flynn's Classification

SISD:

- Single instruction stream, single data stream.
- SISD represents the organization of a single computer containing a control unit, a processor unit, and a memory unit.
- Instructions are executed sequentially and the system may or may not have internal parallel processing capabilities.

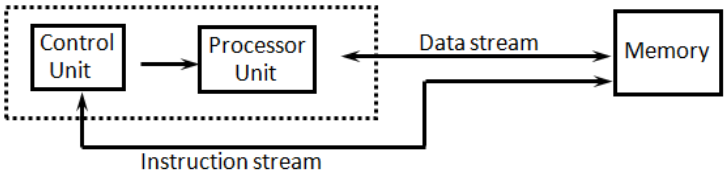

Figure 6.2: SISD Organization

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.

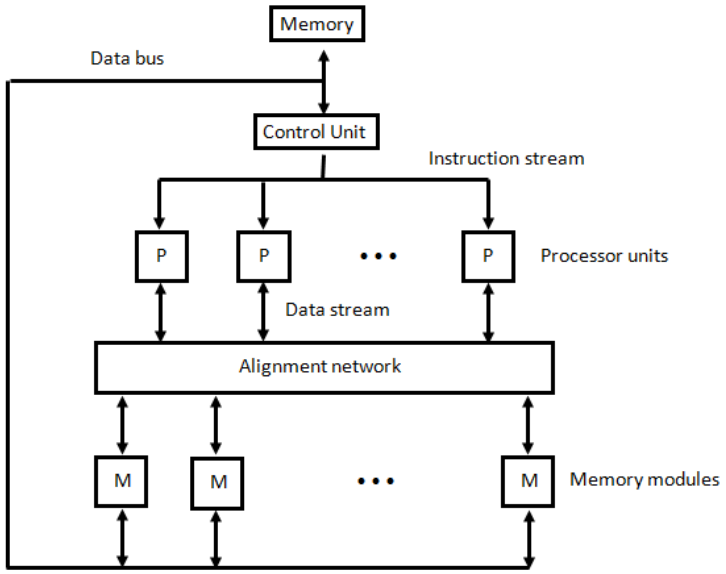

Figure 6.3: SIMD Organization

MISD:

- There is no computer at present that can be classified as MISD.
- MISD structure is only of theoretical interest since no practical system has been constructed using this organization.

MIMD:

- MIMD organization refers to a computer system capable of processing several programs at the same time.
- Most multiprocessor and multicomputer systems can be classified in this category.
- Contains multiple processing units.
- Execution of multiple instructions on multiple data.

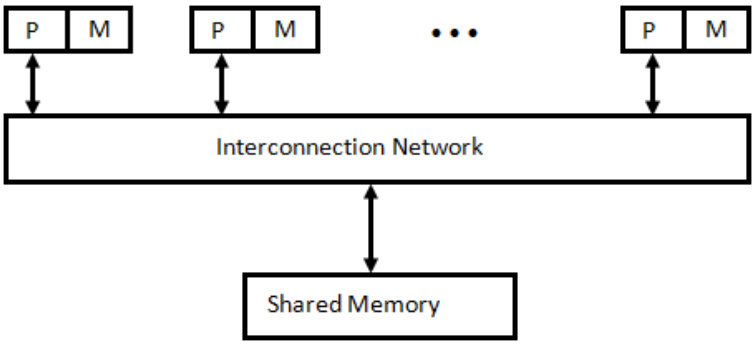

Figure 6.4: MIMD Organization

2. Explain pipelining technique. Draw the general structure of four segment pipeline.

- Pipeline is a technique of decomposing a sequential process into sub operations, with each sub process being executed in a special dedicated segment that operates concurrently with all other segments.
- 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 is partitioned.
- The result obtained from the computation in each segment is transferred to the next segment in the pipeline.
- It is characteristic of pipelines that several computations can be in progress in distinct segments at the same time.
- The overlapping of computation is made possible by associating a register with each segment in the pipeline.
- The registers provide isolation between each segment so that each can operate on distinct data simultaneously.
- Any operation that can be decomposed into a sequence of sub operations 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.
- The general structure of a four-segment pipeline is illustrated in Figure 6.5.

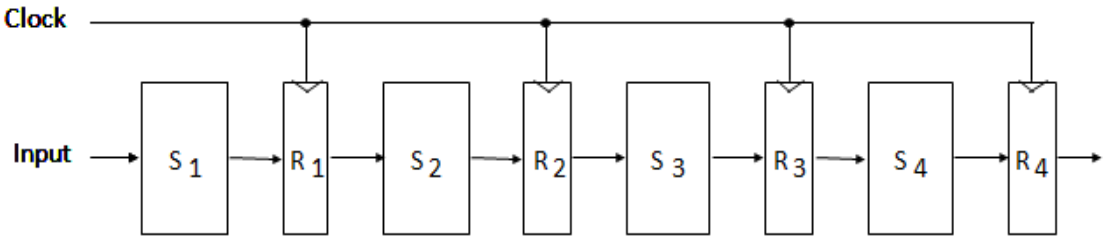

Figure 6.5: General Structure of Four-Segment Pipeline

- The operands pass through all four segments in a fixed sequence.
- Each segment consists of a combinational circuit S, which performs a sub operation over the data stream flowing through the pipe.
- The segments are separated by registers R, which hold the intermediate results between the stages.
- Information flows between adjacent stages under the control of a common clock applied to all the registers simultaneously.
- We define a task as the total operation performed going through all the segments in the pipeline.

3. Draw and explain Arithmetic Pipeline.

- The inputs to the floating-point adder pipeline are two normalized floating-point binary numbers.

$X = A \times 2^a$

$Y = B \times 2^b$

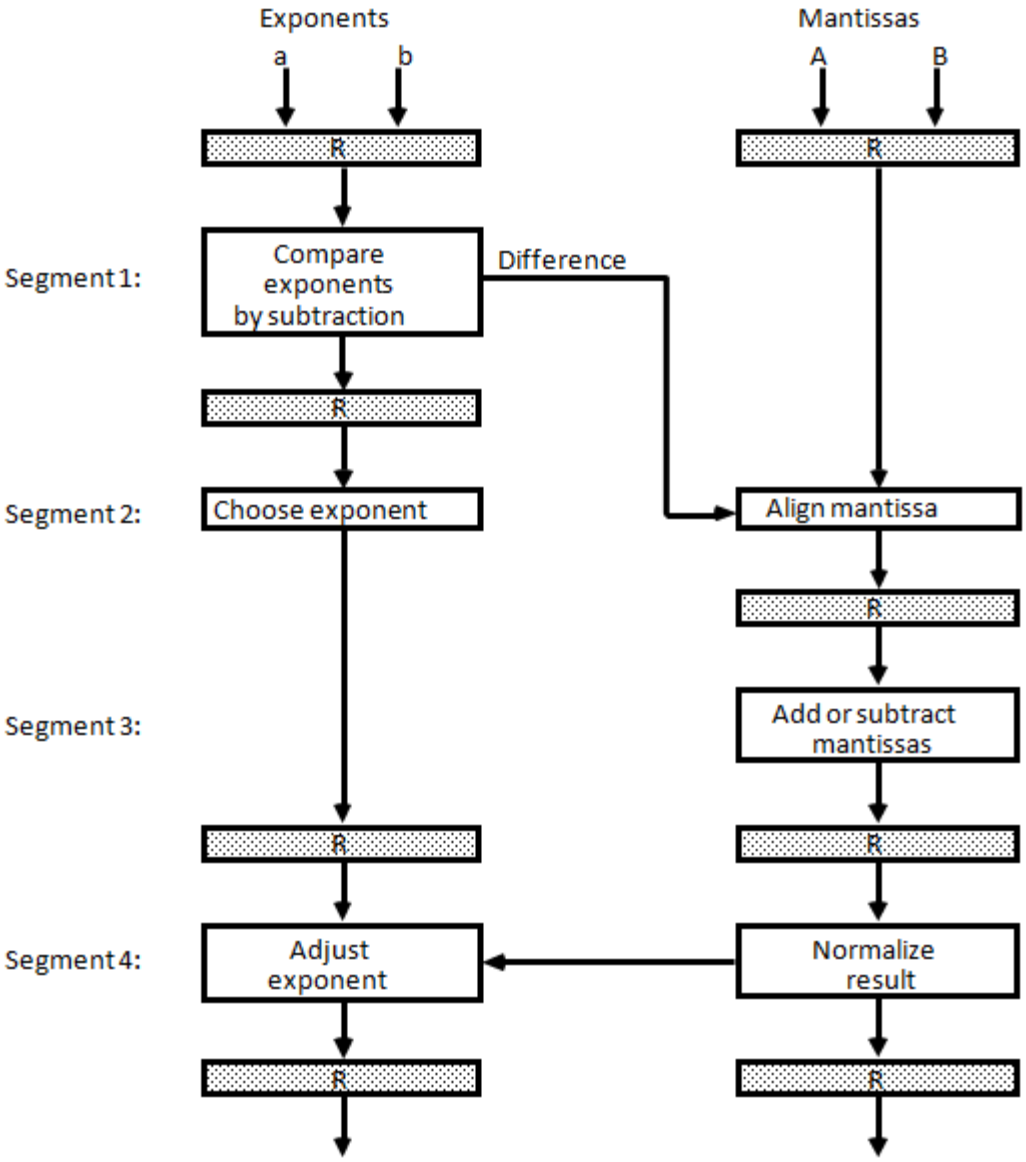

Figure 6.6: Pipeline for floating-point addition and subtraction

- A and B are two fractions that represent the mantissas and a and b are the exponents.
- The floating-point addition and subtraction can be performed in four segments, as shown in Figure 6.6.
- The registers labeled R are placed between the segments to store intermediate results.

- The sub-operations that are performed in the four segments are:

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

- The following numerical example may clarify the sub-operations performed in each segment.
- For simplicity, we use decimal numbers, although Figure 6.6 refers to binary numbers.
- Consider the two normalized floating-point numbers:

  $X = 0.9504 \times 10^3$

  $Y = 0.8200 \times 10^2$
- The two exponents are subtracted in the first segment to obtain 3 - 2 = 1.
- The larger exponent 3 is chosen as the exponent of the result.
- The next segment shifts the mantissa of Y to the right to obtain

  $X = 0.9504 \times 10^3$

  $Y = 0.0820 \times 10^3$
- This aligns the two mantissas under the same exponent. The addition of the two mantissas in segment 3 produces the sum

  $Z = 1.0324 \times 10^3$

4. Explain the Instruction Pipelining with example. OR Discuss four-segment instruction pipeline with diagram(s)

- Pipeline processing can occur in data stream as well as in instruction stream.
- An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments.
- This causes the instruction fetch and executes phases to overlap and perform simultaneous operations.
- One possible digression associated with such a scheme is that an instruction may cause a branch out of sequence.
- In that case the pipeline must be emptied and all the instructions that have been read from memory after the branch instruction must be discarded.
- Consider a computer with an instruction fetch unit and an instruction execution unit designed to provide a two-segment pipeline.
- The instruction fetch segment can be implemented by means of a first-in, first-out (FIFO) buffer.
- The buffer acts as a queue from which control then extracts the instructions for the execution unit.

Instruction cycle:

- 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.

- 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.
- The design of an instruction pipeline will be most efficient if the instruction cycle is divided into segments of equal duration.
- The time that each step takes to fulfill its function depends on the instruction and the way it is executed.

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 registers so that the instruction execution and storing of the result can be combined into one segment.
- This reduces the instruction pipeline into four segments.
    1. FI: Fetch an instruction from memory
    2. DA: Decode the instruction and calculate the effective address of the operand
    3. FO: Fetch the operand
    4. EX: Execute the operation
- Figure 6.7 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 sub operations in the instruction cycle can overlap and up to four different instructions can be in progress of being processed at the same time.

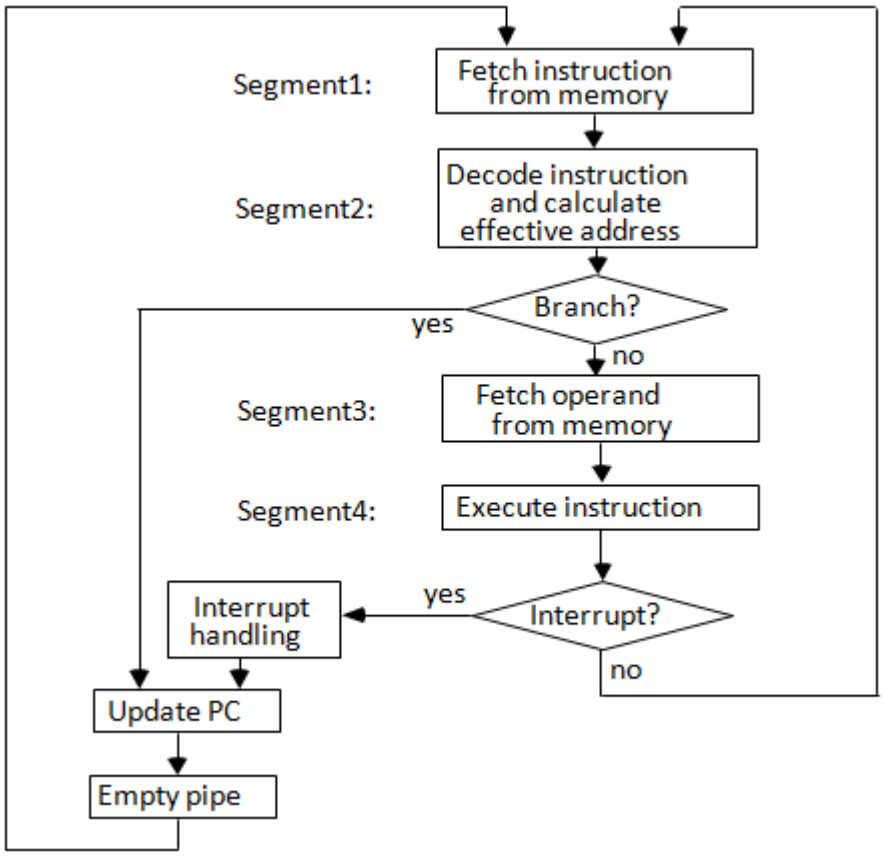

Figure 6.7: Four-segment CPU pipeline

- Figure 6.8 shows the operation of the instruction pipeline. The time in the horizontal axis is divided into steps of equal duration. The four segments are represented in the diagram with an abbreviated symbol.

	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 6.8: Timing of Instruction Pipeline

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

- Thus, in step 4, instruction 1 is being executed in segment EX; the operand for instruction 2 is being fetched in segment FO; instruction 3 is being decoded in segment DA; and instruction 4 is being fetched from memory in segment FI.
- Assume now that instruction 3 is a branch instruction.
- As soon as this instruction is decoded in segment DA in step 4, the transfer from FI to DA of the other instructions is halted until the branch instruction is executed in step 6.
- If the branch is taken, a new instruction is fetched in step 7. If the branch is not taken, the instruction fetched previously in step 4 can be used.
- The pipeline then continues until a new branch instruction is encountered.
- Another delay may occur in the pipeline if the EX segment needs to store the result of the operation in the data memory while the FO segment needs to fetch an operand.
- In that case, segment FO must wait until segment EX has finished its operation.

5. What is pipeline conflict? Explain data dependency and handling of branch instruction in detail.

Pipeline conflict:

There are three major difficulties that cause the instruction pipeline conflicts.

1. Resource conflicts caused by access to memory by two segments at the same time.
2. Data dependency conflicts arise when an instruction depends on the result of a previous instruction, but this result is not yet available.
3. Branch difficulties arise from branch and other instructions that change the value of PC.

Data Dependency:

- A collision occurs when an instruction cannot proceed because previous instructions did not complete certain operations.
- A data dependency occurs when an instruction needs data that are not yet available.
- Similarly, an address dependency may occur when an operand address cannot be calculated because the information needed by the addressing mode is not available.
- Pipelined computers deal with such conflicts between data dependencies in a variety of ways as follows.

Hardware interlocks:

- 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.

Operand forwarding:

- It 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.

Delayed load:

- Sometimes compiler has the responsibility for solving data conflicts problems.
- 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 instruction, this method is called delayed load.

Handling of Branch Instructions:

- One of the major problems in operating an instruction pipeline is the occurrence of branch instructions.
- A branch instruction can be conditional or unconditional.
- The branch instruction breaks the normal sequence of the instruction stream, causing difficulties in the operation of the instruction pipeline.
- Various hardware techniques are available to minimize the performance degradation caused by instruction branching.

Pre-fetch target:

- One way of handling a conditional branch is to prefetch the target instruction in addition to the instruction following the branch.
- If the branch condition is successful, the pipeline continues from the branch target instruction.
- An extension of this procedure is to continue fetching instructions from both places until the branch decision is made.

Branch target buffer:

- Another possibility is the use of a branch target buffer or 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.
- The advantage of this scheme is that branch instructions that have occurred previously are readily available in the pipeline without interruption.

Loop buffer:

- A variation of the BTB is the loop buffer. This is a small very high speed register file maintained by the instruction fetch segment of the pipeline.
- When a program loop is detected in the program, it is stored in the loop buffer in its entirety, including all branches.

Branch Prediction:

- A pipeline with branch prediction uses some additional logic to guess the outcome of a conditional branch instruction before it is executed.
- The pipeline then begins pre-fetching the instruction stream from the predicted path.
- A correct prediction eliminates the wasted time caused by branch penalties.

Delayed branch:

- A procedure employed in most RISC processors is the delayed branch.
- In this procedure, the compiler detects the branch instructions and rearranges the machine language code sequence by inserting useful instructions that keep the pipeline operating without interruptions.