What is Machine Language?

Humans can understand High-level programming languages. It is not necessary to have a deep understanding of the internal CPU, to program using high-level languages. They follow a syntax similar to the English language. Java, C, C++, Python are some high-level programming languages. A computer recognizes machine language but does not understand high-level languages. Therefore, those programs should be converted to computer understandable machine language. This translation is done using a compiler or an interpreter.

A machine language consists of binary digits which are zeros and once. A computer is a digital electronic device, so it uses binary for operations. One indicates the true state / on state while zero indicates the false state / off state. The way of converting a program from high-level language to machine language depends on the CPU.

What is Assembly Language?

Assembly language is the intermediate language between high-level programing languages and machine language. It is one level above machine language. Assembly language is easier to understand than machine language but harder than high-level programming languages. This language is also known as a low-level language because it is close to the hardware level. In order to write effective programs using Assembly, the programmer should have a good understanding of the computer architecture and the register structure. A special compiler known as an assembler is used to convert assembly language instructions to machine code or object code.

Assembly language statements have four sections. They are a label, mnemonic, operand, comment. Label and comments are optional. Mnemonic is the instruction to execute and operands are parameters for the command. Assembly language also supports macros. A macro can be defined as a set of instructions with a name. It can be used elsewhere in the program.

Some examples of Assembly language statements are as follows.

MOV SUM,50 – This instruction, copies the value 50 to the variable SUM.

ADD VALUE1,20 – This is to add 20 to the VALUE1 variable

ADD AH, BH – This instruction is to copy the content in AH register to BH register.

INC COUNT – This is to increment the variable COUNT by one.

AND VALUE1,100 – This is to perform AND operation on variable VALUE1 and 100.

MOV AL,20 – This is to copy value 20 to AL register

What is the Difference Between Machine Language and Assembly language?

Machine Language vs Assembly Language	
Machine language is the lowest level programming language where the instructions execute directly by the CPU.	Assembly language is a low-level programming language which requires an assembler to convert to machine code/object code.
Comprehensibility	
Machine language is comprehensible only to the computers.	Assembly language is comprehensible to humans.
Syntax	
A machine language consists of binary digits.	Assembly language follows a syntax similar to the English language.
Dependency	
Machine language varies depending on the platform.	Assembly language consists of a standard set of instructions.
Applications	
Machine language is machine code.	Assembly language is using for microprocessor-based, real-time systems.

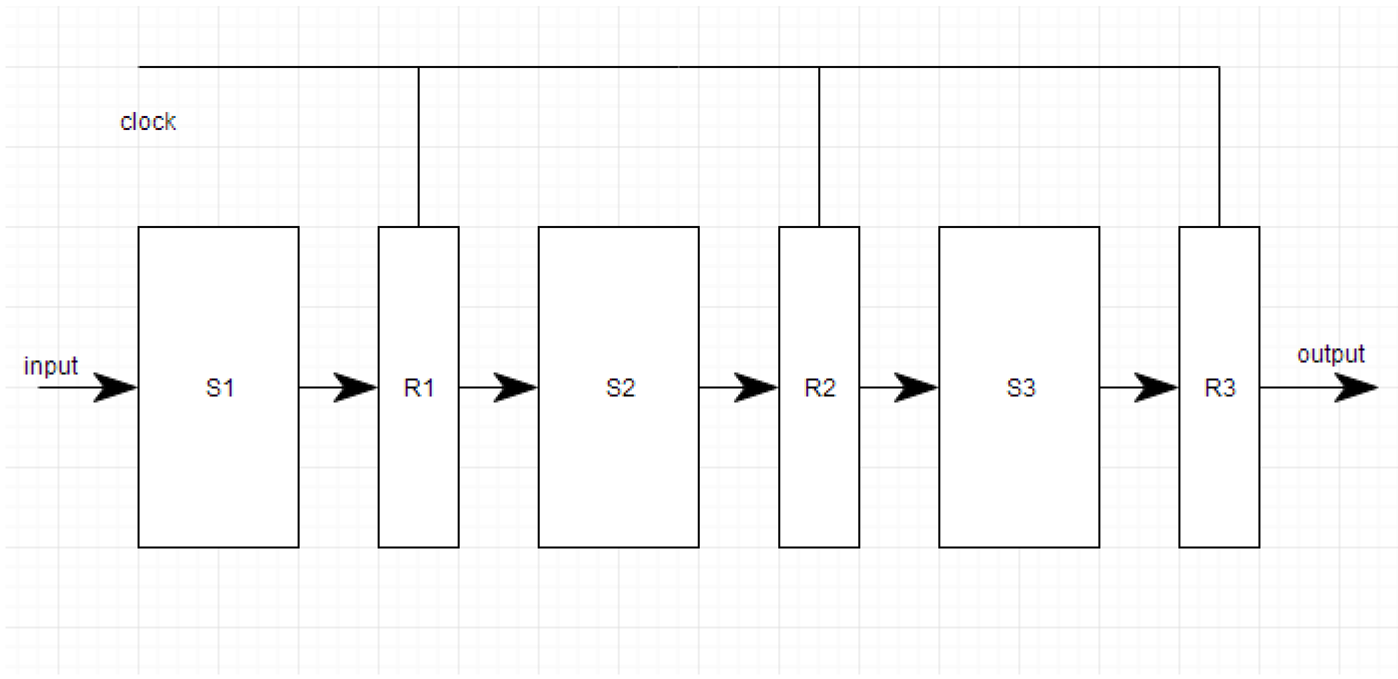
What is Pipelining?

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

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

Pipelining increases the overall instruction throughput.

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



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

Types of Pipeline

It is divided into 2 categories:

1. Arithmetic Pipeline
2. Instruction Pipeline

Arithmetic Pipeline

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

*X = A*2^a*

*Y = B*2^b*

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

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

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

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

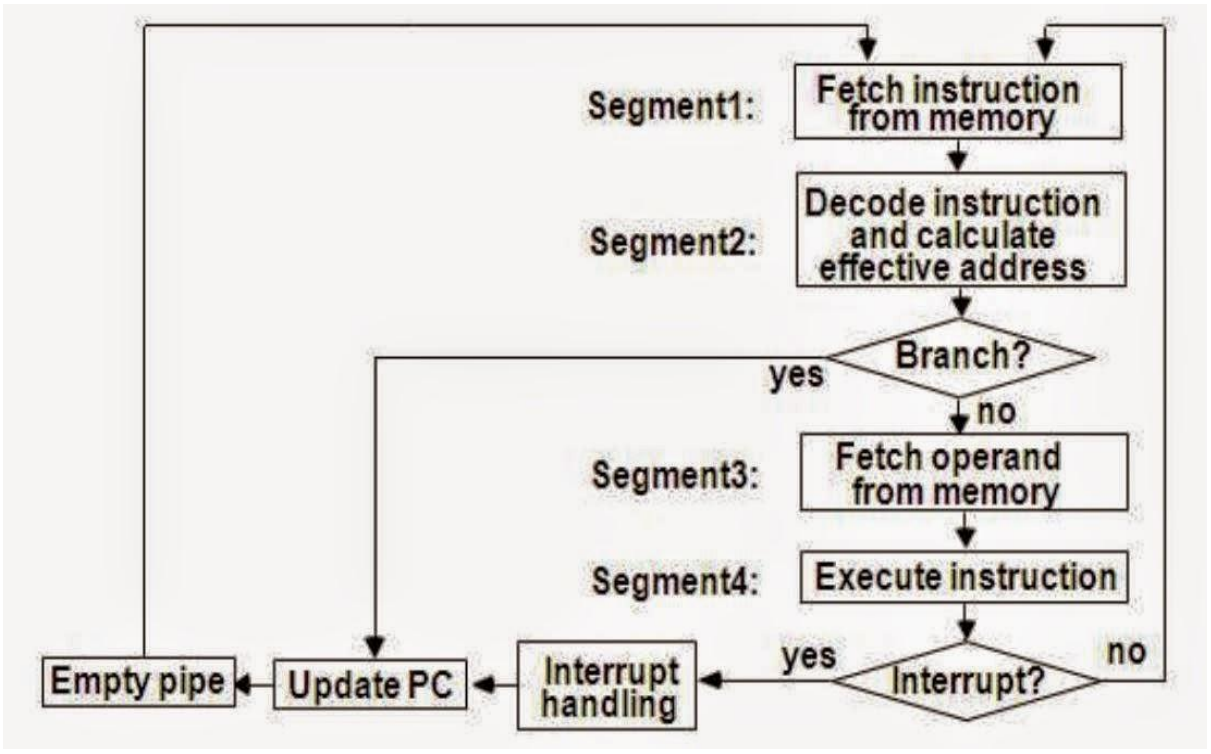
Instruction Pipeline

Instruction pipelining is a technique for implementing instruction-level parallelism within a single processor. Pipelining attempts to keep every part of the processor busy with some instruction by dividing incoming instructions into a series of sequential steps performed by different processor units with different parts of instructions processed in parallel. It allows faster CPU throughput than would otherwise be possible at a given clock rate, but may increase latency due to the added overhead of the pipelining process itself.

In this a stream of instructions can be executed by overlapping *fetch*, *decode* and *execute* phases of an instruction cycle. This type of technique is used to increase the throughput of the computer system.

An instruction pipeline reads instruction from the memory while previous instructions are being executed in other segments of the pipeline. Thus we can execute multiple instructions simultaneously. The pipeline will be more efficient if the instruction cycle is divided into segments of equal duration.

Example of four segment Instruction Pipeline



Pipeline Conflicts

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

1. Timing Variations

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

2. Data Hazards

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

3. Branching

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

4. Interrupts

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

5. Data Dependency

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

Advantages of Pipelining

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

Disadvantages of Pipelining

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

RISC

The term RISC stands for ''Reduced Instruction Set Computer''. It is a CPU design plan based on simple orders and acts fast.

This is small or reduced set of instructions. Here, every instruction is expected to attain very small jobs. In this machine, the instruction sets are modest and simple, which help in comprising more complex commands. Each instruction is about the similar length; these are wound together to get compound tasks done in a single operation. Most commands are completed in one machine cycle. This pipelining is a crucial technique used to speed up RISC machines.

. The features of RISC include the following

- The demand of decoding is less
- Few data types in hardware
- General purpose register Identical
- Uniform instruction set
- Simple addressing nodes

The main common concept of RISC design was a five-stage execution instruction pipeline. During operation, each pipeline stage worked on one instruction at a time. Each of these stages consisted of an initial set of flip-flops and combinational logic that operated on the outputs of those flip-flops.

Fetch

- In the Fetch stage, the processor reads the instruction from instruction memory.

Decode

- In the Decode stage, the processor reads the source operands from the register file and decodes theinstruction to produce the control signals.

Execute

- In the Execute stage, the processor performs a computation with the ALU.

Memory

- In the Memory stage, the processor reads or writes data memory.

Writeback

- Writeback stage, the processor writes the result to the register file, when applicable.

How Hazards Come into picture in a Five Stage Pipeline:

A central challenge in pipelined systems is handling hazards that occur when the results of one instruction areneeded by a subsequent instruction before the former instruction has completed.

Types of Hazards:

Structure Hazards:

This type of Hazard occurs when there is a conflict for use of a common resource. Because somewhere in the pipeline there can be a situation where instruction fetch and the data fetch can happen at the same time. So in that case instruction fetch would have to stall for that cycle, which would cause a pipeline bubble. Hence the pipelined data paths require separate instruction/data memories.

Data Hazards:
This type of Hazard occurs when an instruction depends on the result of a previous instruction which is still in the pipeline.

Possible Solutions for Data Hazards:

- Solving Data Hazards with Forwarding:

Some data hazards can be solved by forwarding (also called bypassing) a result from the memory or write back stage to a dependent instruction in the Execute stage.

- Solving Data Hazards with Stalls:

The alternative solution is to stall the pipeline, holding up operation until the data is available.

Control Hazards:
Control hazards occur when the decision of what instruction to fetch has not been made by the time the next instruction must be fetched. Control hazards are solved by predicting which instruction should be fetched and flushing the pipeline if the prediction is later determined to be wrong. Moving the decision as early as possible minimizes the number of instructions that are flushed on a mis-prediction.

CISC Architecture

The term CISC stands for ‘’Complex Instruction Set Computer’’. It is a CPU design plan based on single commands, which are skilled in executing multi-step operations.

CISC computers have small programs. It has a huge number of compound instructions, which takes a long time to perform. Here, a single set of instruction is protected in several steps; each instruction set has additional than 300 separate instructions. Maximum instructions are finished in two to ten machine cycles. In CISC, instruction pipelining is not easily implemented.

Comparison between RISC and CISC:

	RISC	**CISC**
Acronym	It stands for ‘Reduced Instruction Set Computer’.	It stands for ‘Complex Instruction Set Computer’.

Definition	The RISC processors have a smaller set of instructions with few addressing nodes.	The CISC processors have a larger set of instructions with many addressing nodes.
Memory unit	It has no memory unit and uses a separate hardware to implement instructions.	It has a memory unit to implement complex instructions.
Program	It has a hard-wired unit of programming.	It has a micro-programming unit.
Design	It is a complex complier design.	It is an easy complier design.
Calculations	The calculations are faster and precise.	The calculations are slow and precise.
Decoding	Decoding of instructions is simple.	Decoding of instructions is complex.
Time	Execution time is very less.	Execution time is very high.
External memory	It does not require external memory for calculations.	It requires external memory for calculations.
Pipelining	Pipelining does function correctly.	Pipelining does not function correctly.
Stalling	Stalling is mostly reduced in processors.	The processors often stall.

Code expansion	Code expansion can be a problem.	Code expansion is not a problem.
Disc space	The space is saved.	The space is wasted.
Applications	Used in high end applications such as video processing, telecommunications and image processing.	Used in low end applications such as security systems, home automations, etc.

Applications of RISC and CISC

RISC is used in high-end applications like video processing, telecommunications and image processing. CISC is used in low-end applications such as security systems, home automation, etc.