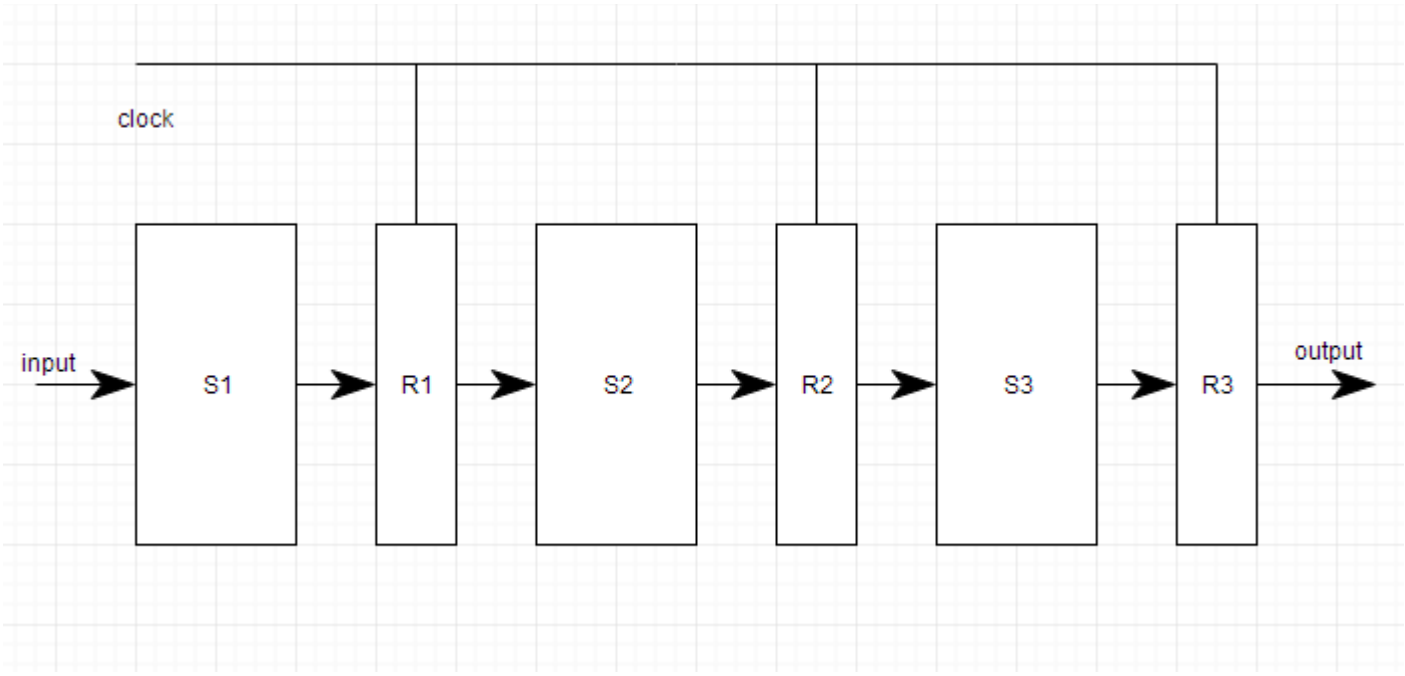
What is Pipelining?

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

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.

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.

Vector(Array) Processing and Superscalar Processors

A **Scalar processor** is a normal processor, which works on simple instruction at a time, which operates on single data items. But in today's world, this technique will prove to be highly inefficient, as the overall processing of instructions will be very slow.

What is Vector(Array) Processing?

There is a class of computational problems that are beyond the capabilities of a conventional computer. These problems require vast number of computations on multiple data items, that will take a conventional computer(with scalar processor) days or even weeks to complete.

Such complex instructions, which operates on multiple data at the same time, requires a better way of instruction execution, which was achieved by Vector processors.

Scalar CPUs can manipulate one or two data items at a time, which is not very efficient. Also, simple instructions like **ADD A to B, and store into C** are not practically efficient.

Addresses are used to point to the memory location where the data to be operated will be found, which leads to added overhead of data lookup. So until the data is found, the CPU would be sitting ideal, which is a big performance issue.

Hence, the concept of **Instruction Pipeline** comes into picture, in which the instruction passes through several sub-units in turn. These sub-units perform various independent functions, **for example**: the **first** one decodes the instruction, the **second** sub-unit fetches the data and the **third** sub-unit performs the math itself. Therefore, while the data is fetched for one instruction, CPU does not sit idle, it rather works on decoding the next instruction set, ending up working like an assembly line.

Vector processor, not only use Instruction pipeline, but it also pipelines the data, working on multiple data at the same time.

A normal scalar processor instruction would be ADD A, B, which leads to addition of two operands, but what if we can instruct the processor to ADD a group of numbers(from 0 to n memory location) to another group of numbers(lets say, n to k memory location). This can be achieved by vector processors.

In vector processor a single instruction, can ask for multiple data operations, which saves time, as instruction is decoded once, and then it keeps on operating on different data items.

Applications of Vector Processors

Computer with vector processing capabilities are in demand in specialized applications. The following are some areas where vector processing is used:

1. Petroleum exploration.
2. Medical diagnosis.

3. Data analysis.
4. Weather forecasting.
5. Aerodynamics and space flight simulations.
6. Image processing.
7. Artificial intelligence.

Superscalar Processors

It was first invented in 1987. It is a machine which is designed to improve the performance of the scalar processor. In most applications, most of the operations are on scalar quantities. Superscalar approach produces the high performance general purpose processors.

The main principle of superscalar approach is that it executes instructions independently in different pipelines. As we already know, that Instruction pipelining leads to parallel processing thereby speeding up the processing of instructions. In Superscalar processor, multiple such pipelines are introduced for different operations, which further improves parallel processing.

There are multiple functional units each of which is implemented as a pipeline. Each pipeline consists of multiple stages to handle multiple instructions at a time which support parallel execution of instructions.

It increases the throughput because the CPU can execute multiple instructions per clock cycle. Thus, superscalar processors are much faster than scalar processors.

A **scalar processor** works on one or two data items, while the **vector processor** works with multiple data items. A **superscalar processor** is a combination of both. Each instruction processes one data item, but there are multiple execution units within each CPU thus multiple instructions can be processing separate data items concurrently.

While a superscalar CPU is also pipelined, there are two different performance enhancement techniques. It is possible to have a non-pipelined superscalar CPU or pipelined non-superscalar CPU. The superscalar technique is associated with some characteristics, these are:

1. Instructions are issued from a sequential instruction stream.
2. CPU must dynamically check for data dependencies.

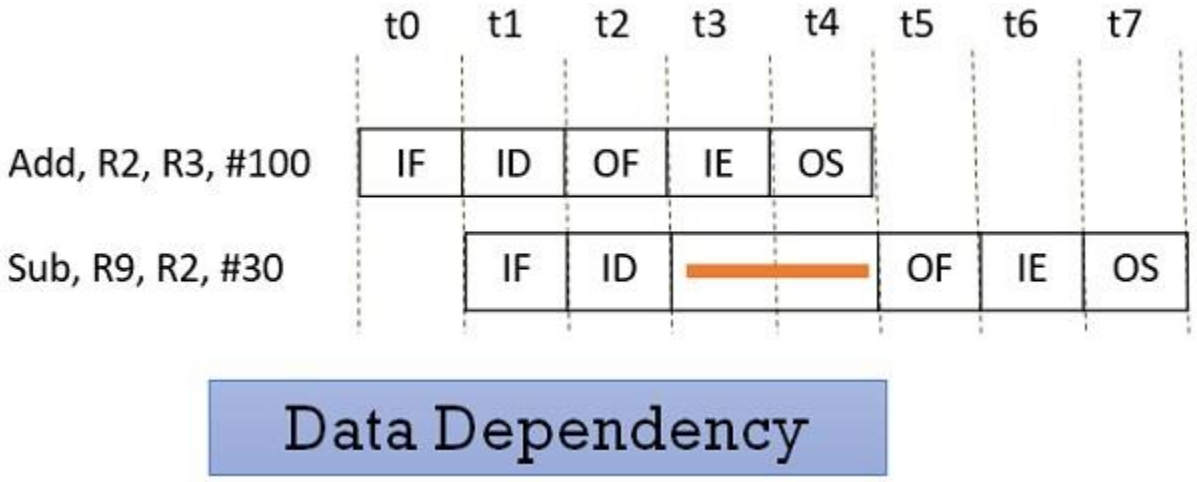
3. Should accept multiple instructions per clock cycle.

Pipelining Hazards

Whenever a pipeline has to stall due to some reason it is called pipeline hazards. Below we have discussed four pipelining hazards.

1. Data Dependency

Consider the following two instructions and their pipeline execution:



In the figure above, you can see that result of the **Add** instruction is stored in the register **R2** and we know that the final result is stored at the end of the execution of the instruction which will happen at the clock cycle **t4**.

But the **Sub** instruction need the value of the register **R2** at the cycle **t3**. So the Sub instruction has to **stall** two clock cycles. If it doesn't stall it will generate an incorrect result. Thus depending of one instruction on other instruction for data is **data dependency**.

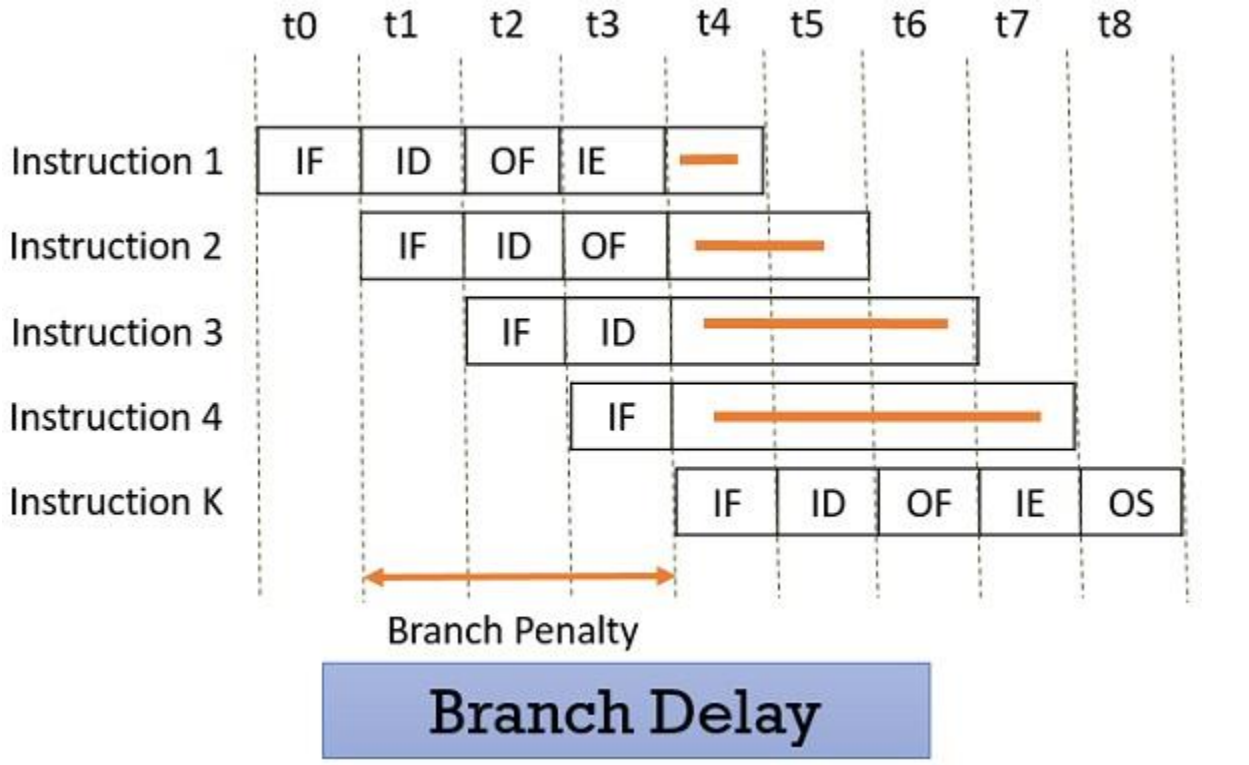
2. Memory Delay

When an instruction or data is required, it is first searched in the cache memory if not found then it is a **cache miss**. The data is further searched in the memory which may take ten or more cycles. So, for that number of cycle the pipeline has to stall and this is a **memory delay** hazard. The cache miss, also results in the delay of all the subsequent instructions.

3. Branch Delay

Suppose the four instructions are pipelined I_1, I_2, I_3, I_4 in a sequence. The instruction I_1 is a branch instruction and its target instruction is I_k. Now, processing starts and instruction I_1 is fetched, decoded and the target address is computed at the 4^{th} stage in cycle t3.

But till then the instructions I_2, I_3, I_4 are fetched in cycle 1, 2 & 3 before the target branch address is computed. As I_1 is found to be a branch instruction, the instructions I_2, I_3, I_4 has to be discarded because the instruction I_k has to be processed next to I_1. So, this delay of three cycles 1, 2, 3 is a **branch delay**.



Prefetching the target branch address will reduce the branch delay. Like if the target branch is identified at the decode stage then the branch delay will reduce to 1 clock cycle.

4. Resource Limitation

If the two instructions request for accessing the same resource in the same clock cycle, then one of the instruction has to stall and let the other instruction to use the resource. This stalling is due to **resource limitation**. However, it can be prevented by adding more hardware.

Advantages

1. Pipelining improves the throughput of the system.
2. In every clock cycle, a new instruction finishes its execution.
3. Allow multiple instructions to be executed concurrently.

Advantages and Disadvantages of Pipelining[change | change source]

Advantages of Pipelining:

1. The cycle time of the processor is reduced; increasing the instruction throughput. Pipelining doesn't reduce the time it takes to complete an instruction; instead it increases the number of instructions that can be processed simultaneously ("at once") and reduces the delay between

completed instructions (called 'throughput').
The more pipeline stages a processor has, the more instructions it can process "at once" and the less of a delay there is between completed instructions. Every predominant general purpose microprocessor manufactured today uses at least 2 stages of pipeline up to 30 or 40 stages.

2. If pipelining is used, the CPU Arithmetic logic unit can be designed faster, but more complex.
3. Pipelining in theory increases performance over an un-pipelined core by a factor of the number of stages (assuming the clock frequency also increases by the same factor) and the code is ideal for pipeline execution.
4. Pipelined CPUs generally work at a higher clock frequency than the RAM clock frequency, (as of 2008 technologies, RAMs work at a low frequencies compared to CPUs frequencies) increasing computers overall performance.

Disadvantages of Pipelining:

Pipelining has many disadvantages though there are a lot of techniques used by CPUs and compilers designers to overcome most of them; the following is a list of common drawbacks:

1. The design of a non-pipelined processor is simpler and cheaper to manufacture, non-pipelined processor executes only a single instruction at a time. This prevents branch delays (in Pipelining, every branch is delayed) as well as problems when serial instructions being executed concurrently.
2. In pipelined processor, insertion of flip flops between modules increases the instruction latency compared to a non-pipelining processor.
3. A non-pipelined processor will have a defined instruction throughput. The performance of a pipelined processor is much harder to predict and may vary widely for different programs.
4. Many designs include pipelines as long as 7, 10, 20, 31 and even more stages; a disadvantage of a long pipeline is when a program branches, the entire pipeline must be flushed (cleared). The higher throughput of pipelines falls short when the executed code contains many branches: the processor cannot know in advance where to read the next instruction, and must wait for the branch instruction to finish, leaving the pipeline behind it empty. This disadvantage can be reduced by predicting whether the a conditional branch instruction will branch based on previous activity. After the branch is resolved, the next instruction has to travel all

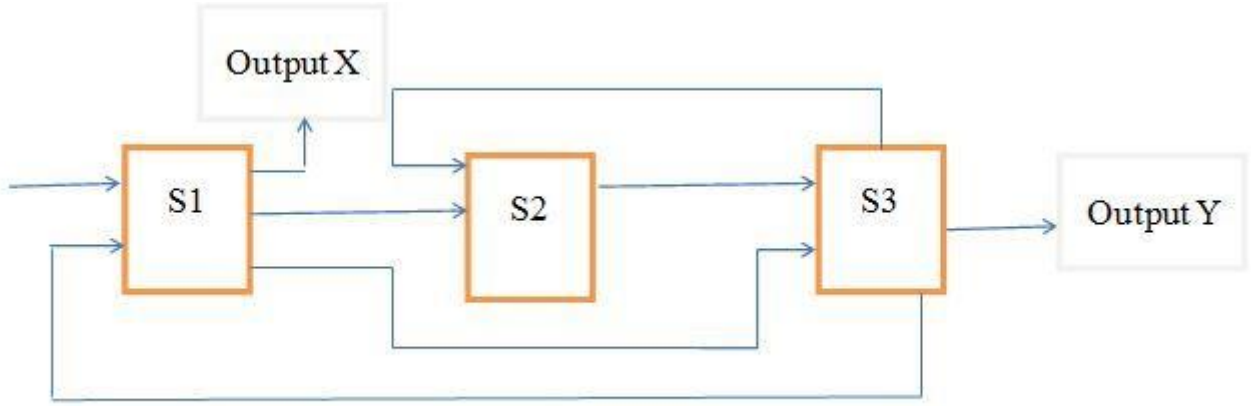
the way through the pipeline before its result becomes available and the processor resumes "working" again. In such extreme cases, the performance of a pipelined processor could be worse than non-pipelined processor.

5. Unfortunately, not all instructions are independent. In a simple pipeline, completing an instruction may require 5 stages. To operate at full performance, this pipeline will need to run 4 subsequent independent instructions while the first is completing. Any of those 4 instructions might depend on the output of the first instruction, causing the pipeline control logic to wait and insert a stall or wasted clock cycle into the pipeline until the dependency is resolved. Fortunately, techniques such as forwarding can significantly reduce the cases where stalling is required.
6. Self-modifying programs may fail to execute properly on a pipelined architecture when the instructions being modified are near the instructions being executed. This can be caused by the instructions may already being in the Prefetch Input Queue, so the modification may not take effect for the upcoming execution of instructions. Instruction caches make the problem even worse.
7. **Hazards**: When a programmer (or compiler) writes assembly code, they generally assume that each instruction is executed before the next instruction is being executed. When this assumption is not validated by pipelining it causes a program to behave incorrectly, the situation is known as a **hazard**.
   Various techniques for resolving hazards or working around such as forwarding and delaying (by inserting a stall or a wasted clock cycle) exist.

Difference Between Linear and Non-Linear pipeline:

Linear Pipeline	Non-Linear Pipeline
Linear pipeline are static pipeline because they are used to perform fixed functions.	Non-Linear pipeline are dynamic pipeline because they can be reconfigured to perform variable functions at different times.
Linear pipeline allows only streamline connections.	Non-Linear pipeline allows feed-forward and feedback connections in addition to the streamline connection.
It is relatively easy to partition a given function into a sequence of linearly	Function partitioning is relatively difficult because the pipeline stages are

ordered sub functions.	interconnected with loops in addition to streamline connections.
The Output of the pipeline is produced from the last stage.	The Output of the pipeline is not necessarily produced from the last stage.
The reservation table is trivial in the sense that data flows in linear streamline.	The reservation table is non-trivial in the sense that there is no linear streamline for data flows.
Static pipelining is specified by single Reservation table.	Dynamic pipelining is specified by more than one Reservation table.
All initiations to a static pipeline use the same reservation table.	A dynamic pipeline may allow different initiations to follow a mix of reservation tables.



A Three stage Pipeline

Reservation Table: Displays the time space flow of data through the pipeline for one function evaluation.

Time **1 2 3 4 5 6 7 8**

X					**X**		**X**
	X		**X**				
		X		**X**		**X**	

S1

Stage **S2**

S3

Reservation function for a function x

Latency: The number of time units (clock cycles) between two initiations of a pipeline is the latency between them. Latency values must be non-negative integers.

Collision: When two or more initiations are done at same pipeline stage at the same time will cause a collision. A collision implies resource conflicts between two initiations in the pipeline, so it should be avoided.

Forbidden and Permissible Latency: Latencies that cause collisions are called **forbidden latencies**. (E.g. in above reservation table 2, 4, 5 and 7 are forbidden latencies).

Latencies that do not cause any collision are called **permissible latencies**. (E.g. in above reservation table 1, 3 and 6 are permissible latencies).

Latency Sequence and Latency Cycle: A **Latency Sequence** is a sequence of permissible non-forbidden latencies between successive task initiations.

A **Latency cycle** is a latency sequence which repeats the same subsequence (cycle) indefinitely.