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**.

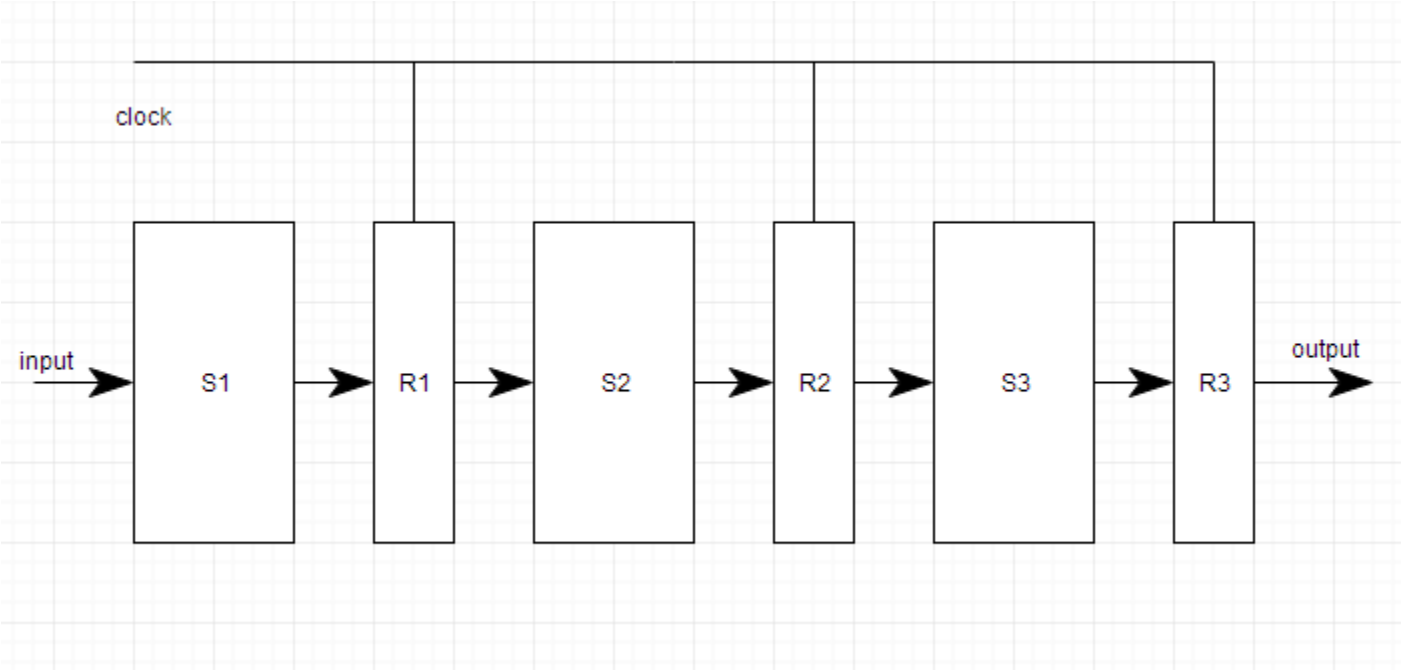
Before moving forward with pipelining, check these topics out to understand the concept better:

- Memory Organization
- Memory Mapping and Virtual Memory
- Parallel 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.

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.

A five-stage pipeline has stage delays of 150, 120, 150, 160 and 140 nanoseconds. The registers that are used between the pipeline stages have a delay of 5 nanoseconds each.

The total time to execute 100 independent instructions on this pipeline, assuming there are no pipeline stalls, is _____ nanoseconds.

Answer

: Given :

k (number of stages) =5, n (number of instruction) =100

Total time = (k+n-1)*tp ,

where

k = number of pipeline stages,
n = number of instructions,
tp = pipeline cycle time.

tp=max(stage delays) + register delay
tp=max(150,120,150,160,140)+5ns
tp=160+5=165ns

Total time =(5+100-1)*165=104*165=17160 ns.

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.

Pipeline hazards

1. Pipeline hazards are situations that prevent the next instruction in the instruction stream from executing during its designated clock cycles.
2. Any condition that causes a stall in the pipeline operations can be called a hazard.
3. There are primarily three types of hazards:

   i. Data Hazards

   ii. Control Hazards or instruction Hazards

   iii. Structural Hazards.

Data Hazards occur when an instruction depends on the result of previous instruction and that result of instruction has not yet been computed. whenever two different instructions use the same storage. the location must appear as if it is executed in sequential order.
There are four types of data dependencies: Read after Write (RAW), Write after Read (WAR), Write after Write (WAW), and Read after Read (RAR). These are explained as follows below.

- **Read after Write (RAW) True Dependency:**
  It is also known as True dependency or Flow dependency. It occurs when the value produced by an instruction is required by a subsequent instruction. For example,
  R3 ← R1 + R2
  R4 ← R2 * R3 Here R3 is produced by instruction 1 and later used by instruction 2

Stalls are required to handle these hazards.

- **Write after Read (WAR) Anti Dependency:**
  It is also known as anti dependency. These hazards occur when the output register of an instruction is used right after read by a previous instruction. For example,

  R1 ← R2 / R3

  R2 ← R4 * R3

  Here R2 in first instruction is being written after just after reading in first instruction

- **Write after Write (WAW) Output Dependency:**
  It is also known as output dependency. These hazards occur when the output register of an instruction is used for write after written by previous instruction. For example,

  R4 ← R2 - R3

  R4 ← R1 * R3

ii. **Structural Hazards:**

This situation arises mainly when two instructions require a given hardware resource at the same time and hence for one of the instructions the pipeline needs to be stalled.

The most common case is when memory is accessed at the same time by two instructions. One instruction may need to access the memory as part of the Execute or Write back phase while other instruction is being fetched. In this case if both the instructions and data reside in the same memory. Both the instructions can't proceed together and one of them needs to be stalled till the other is done with the memory access part. Thus in general sufficient hardware resources are needed for avoiding structural hazards.

iii. **Control hazards:**

The instruction fetch unit of the CPU is responsible for providing a stream of instructions to the execution unit. The instructions fetched by the fetch unit are in consecutive memory locations and they are executed.

However the problem arises when one of the instructions is a branching instruction to some other memory location. Thus all the instruction fetched in the pipeline from consecutive memory locations are invalid now and need to removed(also called flushing of the pipeline).This induces a stall till new instructions are again fetched from the memory address specified in the branch instruction.

Thus the time lost as a result of this is called a branch penalty. Often dedicated hardware is incorporated in the fetch unit to identify branch instructions and compute branch addresses as soon as possible and reducing the resulting delay as a result.