جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

College of Sciences
Intelligent Medical System Department

Lecture 7

Pipelining Design Techniques

Subject: Computer Fundamentals

Level: First

Lecturer: Asst. Lect. Ali Saleem Haleem

Introduction

Pipelining is a CPU hardware design technique used to enhance overall performance. In a pipelined processor, operations are divided into stages that are executed in parallel. This allows multiple instructions to be processed simultaneously, each in a different stage of execution. Instead of completing one instruction at a time, the processor begins a new instruction before the previous one finishes, with each instruction progressing through different stages. This approach enables more efficient instruction handling by overlapping the execution steps.

Pipeline Processor

A **pipeline processor** is a type of CPU architecture that improves processing speed by dividing instruction execution into separate stages. Each stage handles a specific part of the instruction, such as fetching, decoding, executing, memory access, and writing results. While one instruction is being executed, others are moving through the earlier stages, allowing multiple instructions to be processed at the same time.

This is similar to an assembly line in a factory, where different workers (stages) handle different tasks on multiple products (instructions) simultaneously. This overlap increases the overall efficiency and throughput of the processor.

For example, consider how cars are built in a factory:

- One worker installs the engine.
- The next adds the wheels.
- Another paints the car.
- The last one performs final checks.

Similarly, in pipelining, different parts of multiple instructions are processed simultaneously at different stages.

Design of a basic Pipeline

- In a pipelined processor, a pipeline has two ends, the input end and the output end. Between these ends, there are multiple stages/segments such that the output of one stage is connected to the input of the next stage and each stage performs a specific operation.
- Interface registers are used to hold the intermediate output between two stages. These interface registers are also called latch or buffer.
- All the stages in the pipeline along with the interface registers are controlled by a common clock.

It consists of a sequence of m data-processing circuits, called stages or segments, which collectively perform a single operation on a stream of data operands passing through them.

Some processing takes place in each stage, but a final result is obtained only after the entire operand set has passed through the entire pipeline.

Components in the Diagram

The components used in the below diagram are given below **:**

- **Data In**: This is the input data that enters the pipeline.
- **Stages (S1, S2, ..., Sm)**: Each stage performs part of the operation. The pipeline is divided into multiple stages (S1, S2, ..., Sm), and each stage handles a specific operation.
- **Registers (R1, R2, ..., Rm)**: These are pipeline registers. They temporarily hold data between stages to ensure smooth transfer and isolation between operations. Example: R1 stores output of Stage S1 before passing it to S2.
- **Computation Units (C1, C2, ..., Cm)**: These perform the actual processing (like arithmetic or logical operations). Each computation unit corresponds to a specific stage.

- **Control Unit**: Manages the timing and control signals for each stage. Ensures that each stage operates in sync and processes the correct data at the right time.
- **Data Out :** The final output after processing is complete across all pipeline stages.

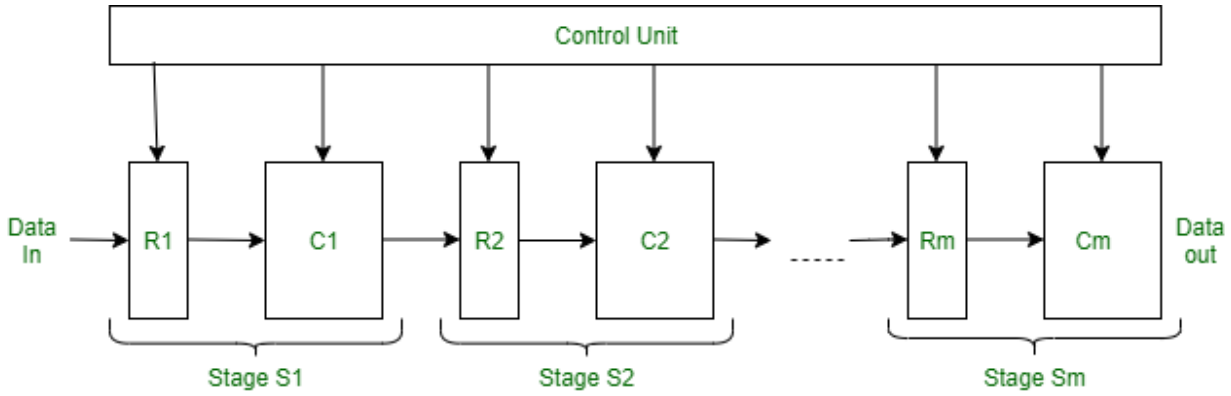

Figure - Structure of a Pipeline Processor

Working

- In **clock cycle 1**, data enters Stage S1 (R1 → C1).
- In **clock cycle 2**, that data moves to Stage S2 (R2 → C2), while new data enters Stage S1.
- This process continues, and each stage is simultaneously processing a different piece of data.
- Eventually, the output appears at the end after passing through all stages.

The working can be explained with the help of the table given below:

Clock Cycle	Stage S1 (R1→C1)	Stage S2 (R2→C2)	Stage S3 (R3→C3)	...	Stage Sm (Rm→Cm)
1	Data A	-	-	...	...
2	Data B	Data A	-	...	...
3	Data C	Data B	Data A	...	...
4	Data D	Data C	Data B	...	...
5	Data E	Data D	Data C	...	...
...	...	...	...	...	...

Instruction Execution In Pipelining

The pipeline will be more efficient if the instruction cycle is divided into segments of equal duration. In the most general case computer needs to process each instruction in following sequence of steps :

- FI : Fetches the instruction into the instruction register.
- DA : Instruction Decode, decodes the instruction for the opcode.
- FO : Data Fetch, fetches the operands into the data register.
- EX: Execution, executes the specified operation and stores it.

Example of Instruction pipeline

- In the first clock cycle, the first instruction is fetched in Segment 1.
- In the second clock cycle, the first instruction moves to the Decode stage, while the second instruction is fetched. This demonstrates instruction-level parallelism enabled by pipelining.
- This overlapping continues, improving throughput.

However, consider the third instruction, which is a branch instruction:

- While the third instruction is in the Decode stage, the fourth instruction is fetched.
- Since the branch might alter the flow of execution, the actual target of the next instruction may change.
- Therefore, the fourth instruction is held (or discarded in some designs) until the branch is resolved during the Execute stage.
- Once the branch outcome is known, either the next instruction is re-fetched (if the branch is taken), or the original fourth instruction continues (if the branch is not taken).

This scenario illustrates a control hazard, where the pipeline must stall or take corrective action due to the uncertainty introduced by branch instructions.

	Stage	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

Pipelining Hazards

Pipelining is not suitable for all kinds of instructions. When some instructions are executed in pipelining, they can stall the pipeline or flush it totally. This type of problems caused during pipelining is called Pipelining Hazards.

The three different types of hazards in computer architecture are:

Structural Hazards :

Structural Hazards arises due to the resource conflict in the pipeline. A resource conflict is a situation when more than one instruction tries to access the same resource in the same cycle. A resource can be a register, memory, or ALU.

Data 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)
Read after Read (RAR)

Control Hazards :

Control Hazards occurs during the transfer of control instructions such as BRANCH, CALL, JMP, etc. On many instruction architectures, the processor will not know the target address of these instructions when it needs to insert the new instruction into the pipeline. Due to this, unwanted instructions are fed to the pipeline.

Performance Evolution Factor for Pipelining

- **Latency**:
  Latency is the time taken for a single instruction to complete its execution. A lower latency indicates better performance.
- **Efficiency**:
  Pipeline efficiency measures how effectively the pipeline stages are utilized. It is calculated by dividing the total time spent on useful work by the total time taken for the entire pipeline.
- **Throughput** :
  Throughput refers to the number of instructions completed per unit of time. In a pipeline processor, a higher throughput indicates better performance.

Benefits of Pipelining

- Increases instruction throughput.
- Better utilization of processor components.
- Efficient for repetitive tasks or streaming data.

Disadvantages of Pipelining

- Designing of the pipelined processor is complex.
- The problem of hazard for branch instructions increases with the longer pipeline.
- It is difficult to predict the throughput of a pipelined processor.