Pipelining

Pipelining

- Pipelining is the process of arrangement of hardware elements of CPU such that the overall performance is increased.
- Simultaneous execution of more than one instruction takes place in pipelined process.
- In pipelining, multiple instructions are overlapped in execution

Non pipelining

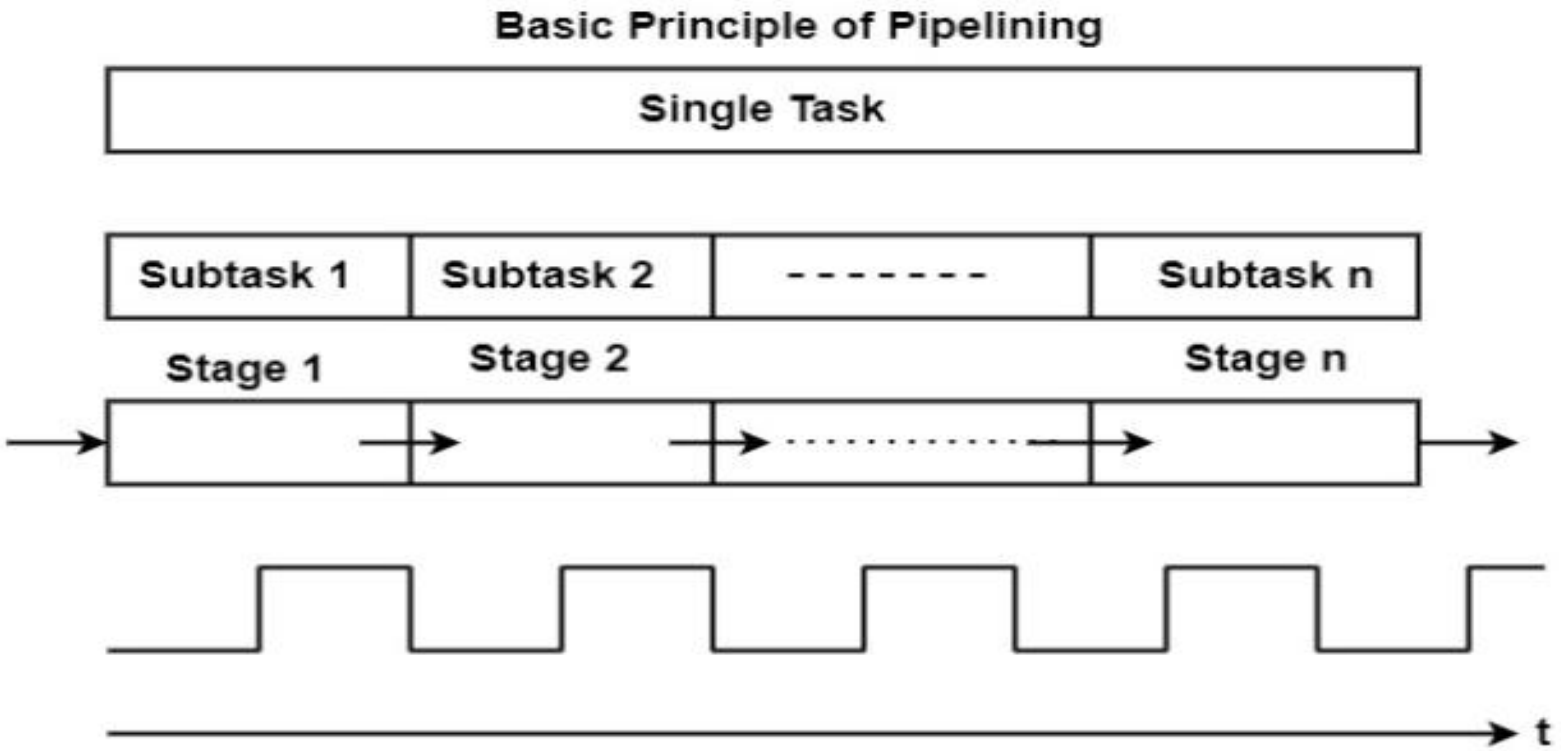
- A process is started and until it is completed , next process will not be started.

- This is the reason why there comes the problem of performance in non pipelining process.

Pipelining

- A basic pipeline processes a sequence of tasks, including instructions, as per the following principle of operation –
- Each task is subdivided into multiple successive subtasks as shown in the figure. For instance, the execution of register-register instructions can be broken down into instruction fetch, decode, execute, and writeback.

Pipelining


Basic Principle of Pipelining
Single Task
Subtask 1
Subtask 2
- - - - - - -
Subtask n
Stage 1
Stage 2
Stage n
t

Pipelining

- Registers are kept in intermediate paths between the stages to store the datas.

Overlapping

In pipelining, **one process** is running in one particular stage and **some other process** is running in **some other stages**, this is known as **overlapping**.

Why we divide a process in multiple stages?

Think!

Pipelining

- A pipeline phase related to each subtask executes the needed operations.

- A similar amount of time is accessible in each stage for implementing the needed subtask.

Pipelining

- All pipeline stages work just as an assembly line that is, receiving their input generally from the previous stage and transferring their output to the next stage.

Pipelining

- Finally, it can consider the basic pipeline operates clocked, in other words synchronously.

- This defines that each stage gets a new input at the beginning of the clock cycle, each stage has a single clock cycle available for implementing the needed operations, and each stage produces the result to the next stage by the starting of the subsequent clock cycle.

Advantages of Pipelining

- The **cycle time of the processor** is decreased. It can improve the instruction throughput. Pipelining doesn't lower the time it takes to do an instruction. Rather than, it can raise the multiple instructions that can be processed together ("at once") and lower the delay between completed instructions (known as 'throughput').

Advantages of Pipelining

- If pipelining is used, the **CPU** Arithmetic logic unit (**ALU**) can be designed quicker, but more complex.

Advantages of Pipelining

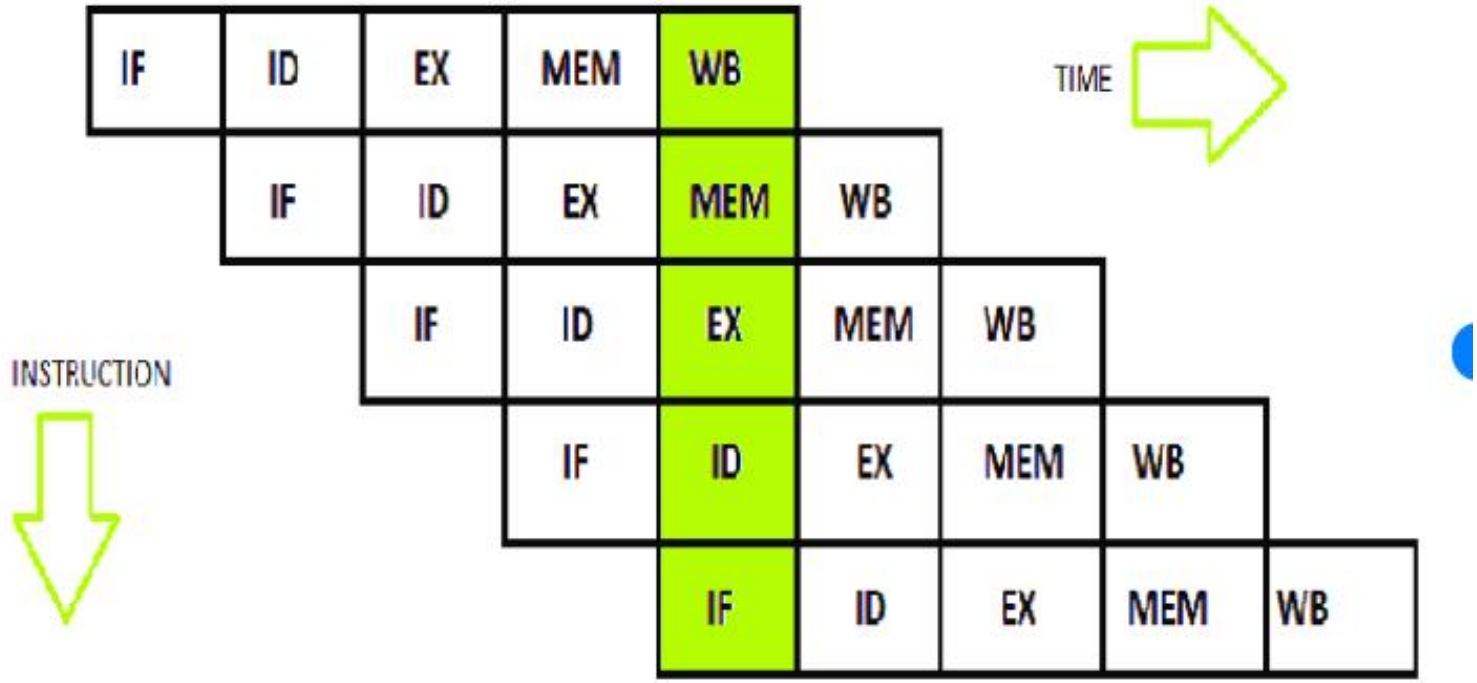
- Pipelining increases execution over an un-pipelined core by an element of the multiple stages (considering the clock frequency also increases by a similar factor) and the code is optimal for pipeline execution.

Advantages of Pipelining

- Pipelined CPUs frequently work at a higher clock frequency than the RAM clock frequency, (as of 2008 technologies, **RAMs** operate at a low frequency correlated to CPUs frequencies) increasing the computer's global implementation.

Thank You

Five Stages of Pipeline



Pipelining

- Let a program has 5 Instructions.
- Each stage is completed in 1 clock cycle (cc).
- An instruction has 5 stages.
- Therefore instruction requires 5 cc and the program requires 5X5=25cc.
- This is for non pipeline instructions.

Pipelining

- In computer architecture, the concept of pipelining is a critical technique that enhances the efficiency and performance of microprocessors.

- Pipelining allows multiple instructions to be overlapped in execution, much like an assembly line in a factory.

Pipelining

- This article delves into the **five primary stages** of a pipeline: Instruction Fetch, Instruction Decode, Execute, Memory Access, and Write Back.
- Each stage is crucial for the smooth operation of modern CPUs, contributing to faster processing and more efficient computing.

Instruction Fetch (IF)

- The Instruction Fetch stage is the first step in the pipeline process. Here, the CPU retrieves an instruction from the program memory. The primary tasks in this stage include:
- *-Program Counter (PC) Management:*
- The Program Counter holds the address of the next instruction to be fetched. It increments after each fetch, pointing to the subsequent instruction in the sequence.

Instruction Fetch (IF)

- *- Instruction Memory Access:*
- The instruction is fetched from the instruction memory (usually a cache or RAM). This access is typically very fast due to the high-speed memory technologies used.

Instruction Fetch (IF)

- *- Buffering*:
- The fetched instruction is placed into an Instruction Register (IR) or buffer to be used in the subsequent stages.
- Efficiency at this stage is paramount as delays here can stall the entire pipeline, leading to performance bottlenecks. Advanced CPUs often use techniques like prefetching to mitigate potential delays.

Instruction Decode (ID)

- Once the instruction is fetched, it enters the Instruction Decode stage. This stage involves interpreting the fetched instruction and preparing the necessary operands for execution. Key activities include:
- *- Opcode Decoding:*
- The instruction is broken down into its constituent parts, such as the operation code (opcode), source operands, and destination operands.

Instruction Decode (ID)

- *- Register Read:*
- The required operands are read from the register file. Modern CPUs often have multiple read ports to facilitate simultaneous access to several registers.

Instruction Decode (ID)

- - *Instruction Classification:*
- Instructions are classified into various types (e.g., **arithmetic, logical, load/store, control**) to determine the subsequent steps in the pipeline.
- This stage is also responsible for hazard detection and forwarding to resolve data dependencies and prevent pipeline stalls.

Execute (EX)

- The Execute stage is where the actual computation or operation specified by the instruction takes place. This stage includes:
- - *ALU Operations:*
- Arithmetic and logical operations are performed using the Arithmetic Logic Unit (ALU). This includes operations like **addition, subtraction, AND, OR, and shifts**.

Execute (EX)

- - *Address Calculation:*
- For memory access instructions, the effective address is calculated by the ALU.

Execute (EX)

- - *Branch Evaluation:*
- If the instruction involves a branch, the branch condition is evaluated, and the Program Counter may be updated accordingly.
- The execution stage is critical for the overall performance, as complex instructions can take multiple cycles, potentially causing pipeline stalls.

Memory Access (MEM)

- After execution, some instructions require access to memory to read or write data. The Memory Access stage handles these operations. Key functions include:
- - *Load Operations:*
- Data is read from memory and placed into a register.

Memory Access (MEM)

- - *Store Operations:*
- Data from a register is written to memory.
  - *Address Translation:*
- In systems with virtual memory, the effective address calculated during the Execute stage is translated to a physical address.
- Efficiency in this stage is achieved through techniques like caching, which reduces latency by storing frequently accessed data closer to the CPU.

Write Back (WB)

- The final stage in the pipeline is Write Back. Here, the results of the executed instructions are written back to the CPU's register file. This stage involves:
- - *Register Write:*
- The computed data or the data fetched from memory is written into the destination register.

Write Back (WB)

- - *Completion Logging:*
- The completion of the instruction is logged, and any necessary updates to status registers or flags are made.
- Write Back is crucial for ensuring that subsequent instructions have access to the correct and updated data, maintaining the integrity and consistency of the processor's state.

Write Back (WB)

- Pipelining is a fundamental technique in modern CPU design that significantly improves instruction throughput.
- Understanding the five stages of a pipeline—Instruction Fetch, Instruction Decode, Execute, Memory Access, and Write Back—provides insights into how processors manage and execute multiple instructions efficiently.

Write Back (WB)

- Each stage is intricately linked and optimized to minimize delays and maximize performance.
- Advanced techniques like parallelism, hazard detection, and caching are employed to further enhance the pipeline's efficiency, ensuring that modern processors can handle complex and demanding computational tasks with speed and precision.

Thank You

Hazards in pipeline

- As we all know, the CPU's speed is limited by memory. There's one more case to consider, i.e. a few instructions are at some stage of execution in a pipelined design.
- There is a chance that these sets of instructions will become dependent on one another, reducing the pipeline's pace. Dependencies arise for a variety of reasons, which we will examine shortly.

Hazards in pipeline

- The dependencies in the pipeline are referred to as hazards since they put the execution at risk.

Hazards in pipeline

- We can swap the terms, dependencies and hazards since they are used interchangeably in computer architecture.

- A **hazard**, in essence, prevents an instruction present in the pipe from being performed during the specified clock cycle.

Hazards in pipeline

- Since each of the instructions may be in a separate machine cycle, we use the term clock cycle.
- In pipelining, CPI = 1
- Achieving CPI = 1 is very difficult.
- The problems faced to achieve this value of CPI are know as hazards.

Hazards in pipeline

- Due to Hazards, delay occurs in the system.
- This delay does not allows the CPI value to be equal to 1.

Types of Pipeline Hazards

The **three different types** of hazards in computer architecture are:

1. Structural
2. **Data**
3. Control

Data Hazards

- RAW – Read after write(True dependancy- TD)
- WAR – Write after Read (Anti dependancy - AD)
- WAW – Write after write (Output dependancy - OD)

Types of Pipeline Hazards

- Dependencies can be addressed in a variety of ways. The easiest is to introduce a bubble into the pipeline, which stalls it and limits throughput.

- The bubble forces the next instruction to wait until the previous one is completed.

Structural Hazard

- Hardware resource conflicts among the instructions in the pipeline cause structural hazards.

- Memory, a GPR Register, or an ALU might all be used as resources here.

Structural Hazard

- **When more than one instruction in the pipe requires access to the very same resource in the same clock cycle, a resource conflict is said to arise.**

- In an overlapping pipelined execution, this is a circumstance where the hardware cannot handle all potential combinations.

Data Hazards

- Data hazards in pipelining emerge when the **execution of one instruction is dependent on the results of another instruction that is still being processed in the pipeline**.

Data Hazards

- The order of the READ or WRITE operations on the register is used to classify data threats into three groups.

Control Hazards

- Branch hazards are caused by branch instructions and are known as **control hazards** in computer architecture. The flow of program/instruction execution is controlled by branch instructions.
- Remember that conditional statements are used in higher-level languages for iterative loops and condition testing (correlate with while, for, and if case statements).

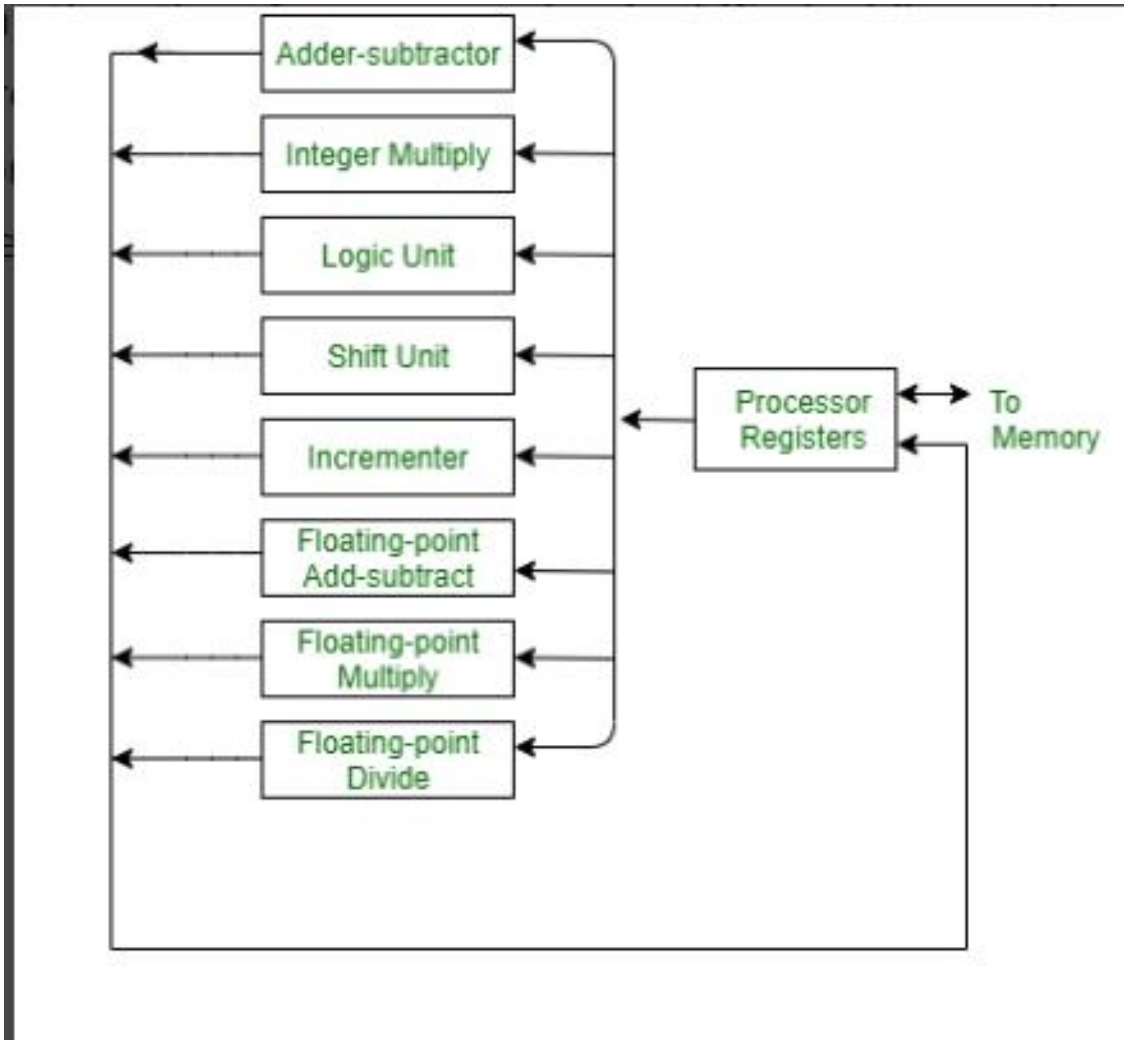
Control Hazards

- These are converted into one of the BRANCH instruction variations.

- As a result, when the decision to execute one instruction is reliant on the result of another instruction, such as a conditional branch, which examines the condition's consequent value, a conditional hazard develops.

- Parallel processing

  SISD
  SIMD
  MISD
  MIMD


Adder-subtractor
Integer Multiply
Logic Unit
Shift Unit
Incrementer
Floating-point Add-subtract
Floating-point Multiply
Floating-point Divide
Processor Registers
To Memory

- Parallel processing is studied under the following main topics

  Pipeline processing
  Vector processing
  Array processors

Thank You