Pipelined Processor Design

Madiba Hudson-Quansah

Contents

Chapter 1

Introduction

Definition 1.0.1: Pipelining

The execution of an instruction is broken down into multiple stages, i.e. fetch, decode, etc, then stages of instruction execution are executed in an overlapping manner allowing multiple instructions to executed simultaneously.

1.1 Pipelining Implementation

Consider a task that can be divided into k subtasks, where k subtasks are executed on k different stages. Each subtask requires one time unit and the total execution time of the task is k time units. Pipelining is to overlap the execution such that the k stages work in parallel on k different tasks, where each task leaves and enters the pipeline at a rate of one task per time unit.

Let τ_i be time delay in stage S_i. The Clock Cycle is the maximum stage delay, i.e. $\tau = \max(\tau_i)$, and accordingly the clock rate/frequency is $\frac{1}{\tau}$ or $\frac{1}{\max(\tau_i)}$. A pipeline can process n tasks in $k + n - 1$ cycles where k cycles are needed to complete the first task and $n - 1$ cycles are needed to complete the remaining tasks. This gives the following equation:

$$S_k = \frac{\text{Serial execution in cycles}}{\text{Pipelined execution in cycles}} = \frac{nk}{k + n - 1} S_k = k \text{ for large } n$$

Given the five stages:

- Instruction Fetch (IF)
- Instruction Decode (ID)
- Execution (EX)
- Memory Access (MEM)
- Write Back (WB)

Example 1.1.1

Question 1

Consider a 5-stage instruction execution in which

- IF = ALU = Data Memory Access = 200 ps
- Register Read = Register Write = 150 ps

1. What is the clock cycle time for a single cycle processor
2. What is the clock cycle time for a pipelined processor
3. What is the speed-up factor of the pipelined processor

Solution:

1. $200 + 200 + 200 + 150 + 150 = 900\text{ps}$
2. 200ps, i.e. max stage delay
3. $$S_k = \frac{900}{200} = 4.5$$

1.2 Pipeline Data Path

A single cycle design can be converted into a pipelined design by introducing pipeline registers at the end of each stage, i.e. IF/ID, ID/EX, EX/MEM, MEM/WB. The pipeline registers store intermediate results and control signals and the same clock edge that triggers the next stage also triggers the pipeline registers. There arises a problem with this design where the instruction in the Instruction Decode (ID) stage is different from the one in the Write Back (WB) stage, i.e. the WB stage is writing to a different destination register. This is solved by pipelining the destination register from ID to WB, i.e. the destination register is passed along the pipeline. This is known as **Forwarding**.

Ideally all the pipeline is executing all the instruction stages at the same time, but in reality the pipeline is not always full, i.e. there are bubbles in the pipeline. Some instruction types skip stages, i.e.:

- Store instructions skip the WB stage
- ALU instructions skip the MEM stage
- Branch instructions skip the MEM and WB stages
- Jump instructions skip the ID, EX, MEM, and WB stages

With the addition of pipeline registers more control signals are needed to control the pipeline. These control signals are generated in the ID stage .

1.3 Pipeline Hazards

Definition 1.3.1: Hazard

A situation that would cause incorrect execution of the pipeline.

There are three types of hazards:

Structural Hazards - Caused by resource contention, i.e. two instructions require the same resource in the same cycle.

Data Hazards - Cause by data dependencies between instructions, i.e. An instruction computes a result needed by another instruction.

Control Hazards - Caused by instructions that change the control flow (branches/jumps), i.e. delays in changing the flow of control.

1.3.1 Structural Hazards

This hazard can be solved by:

Delay access to resource - Requires a mechanism to delay instruction access to resource.

Add more hardware resources - Requires more hardware resources to allow multiple instructions to access the same resource.

1.3.2 Data Hazards

Definition 1.3.2: Read After Write (RAW) Hazard

An instruction reads a register before a previous instruction writes to it.

An example of this hazard is the RAW (Read After Write) hazard. This can be solved by:

Stalling - Inserting a bubble in the pipeline, i.e. waiting for the previous instruction to write to the register. This wastes a cycle.

Forwarding - Forwarding data from the instruction that writes to the instruction that reads.