7c (B)

(2)

Pipelining: Basic and Intermediate Concepts

Md Saiful Ambia Chowdhury
Lecturer, dept of CSE
Leading University

What is Pipelining?

- Pipelining is an implementation technique whereby multiple instructions are overlapped in execution. It is the key implementation technique used to make fast CPUs.
- Each step in the pipeline completes a part of an instruction. Different steps are completing different parts of different instructions. Each of these steps is called *pipe stages* or *pipe segment.*

decrease

- *Pipelining ~~yields a reduction~~ in the average execution time per instruction.*
- *The throughput of an instruction pipeline is determined by how often an instruction exits the pipeline.*

increase the throughput.

Balancing Pipelining?

- *The time required between moving an instruction one step down the pipeline is called a processor cycle. Because all stages proceed at the same time, the length of a processor cycle is determined by the time required for the slowest pipe stage.*

- *The pipeline designer's goal is to balance the length of each pipeline stages. If the stages are perfectly balanced, then the time per instruction on the pipeline processor is equal to-*

$$\frac{\text{Time per instruction on unpipelined machine}}{\text{Number of pipe stages}}$$

But in reality, the stages can not be perfectly balanced.

The Basics of a RISC Instruction Set

- RISC stands for "reduced instruction set computer"

- Most RISC architecture have three classes of instructions:

  ALU instructions- these instructions take either two registers or a register and a sign-extended immediate instruction, which operates on them and store the result in a third register. Eg. Add (DADD), subtract (DSUB) and logical operations (AND or OR).

  Load and store instructions- these instructions take a register source, called base register and an immediate field, called offset, as operands. The sum (called the effective address) of these operands is used as a memory address. The second register operand is the source/destination of the data. Eg. Load word (LD) and store word (SD)

  Branched and jumps- branches are conditional transfers of control.

A simple implementation of a RISC instruction set

- Every Instruction in the RISC can be implemented in at most 5 clock cycles, the 5 clock cycles are as follows:
  - *Instruction fetch cycle* (IF)- send the program counter (PC) to memory and fetch the current instruction from memory. Update the PC.
  - *Instruction decode cycle* (ID)- decode the instruction and read the registers.
  - *Execution cycle* (EX)- ALU performs ~~one of the three functions~~ depending on the instruction type:
    - Memory reference:
    - Register-register ALU instruction
    - Register-immediate ALU instruction

A simple implementation of a RISC instruction set (cont.)

- *Memory access* (MEM): If the instruction is load, memory does a read using the effective address. If it is a store, then the memory writes the data from the second register.
- *Write-back cycle* (WB)- write the result into the register file ~~whether~~ it comes ~~from the memory system or~~ from the ALU;

A simple implementation of a RISC instruction set (cont.)

Instruction number	Clock number								
	1	2	3	4	5	6	7	8	9
Instruction i	IF	ID	EX	MEM	WB				
Instruction $i+1$		IF	ID	EX	MEM	WB			
Instruction $i+2$			IF	ID	EX	MEM	WB		
Instruction $i+3$				IF	ID	EX	MEM	WB	
Instruction $i+4$					IF	ID	EX	MEM	WB

Figure A.1 Simple RISC pipeline. On each clock cycle, another instruction is fetched and begins its 5-cycle execution. If an instruction is started every clock cycle, the performance will be up to five times that of a processor that is not pipelined. The names for the stages in the pipeline are the same as those used for the cycles in the unpipelined implementation: IF = instruction fetch, ID = instruction decode, EX = execution, MEM = memory access, and WB = write back.

A 5-stage pipeline

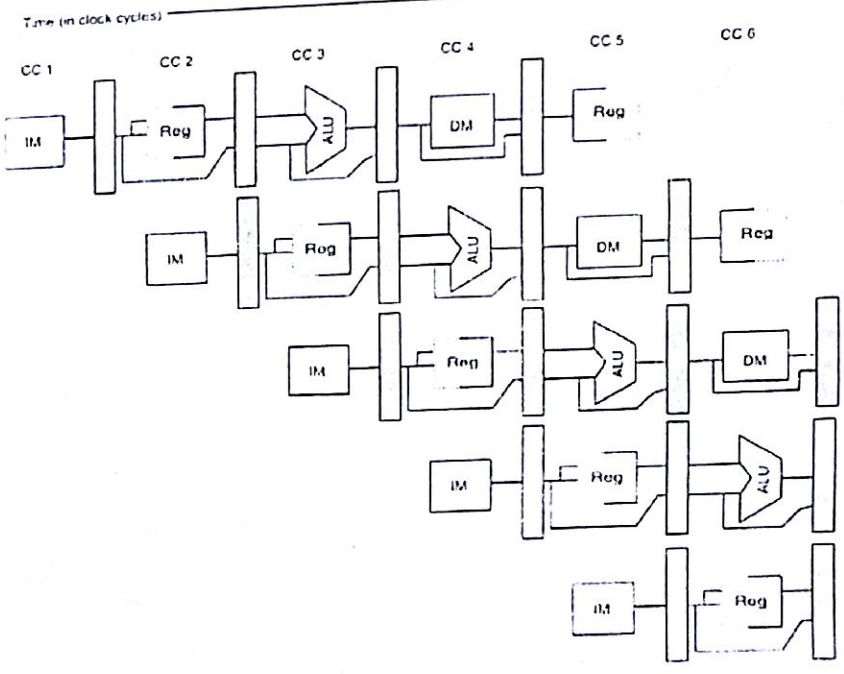

Fig: A pipeline showing the pipeline registers between successive pipeline stages.

Pipeline performance

Example Consider the unpipelined processor ~~in the previous section~~. Assume that it has a $\frac{1}{5}$ ns clock cycle and that it uses 4 cycles for ALU operations and branches and 5 cycles for memory operations. Assume that the relative frequencies of these operations are 40%, 20%, and 40%, respectively. Suppose that due to clock skew and setup, pipelining the processor adds 0.2 ns of overhead to the clock. Ignoring any latency impact, how much speedup in the instruction execution rate will we gain from a pipeline?

Answer The average instruction execution time on the unpipelined processor is

$$\begin{aligned}\text{Average instruction execution time} &= \text{Clock cycle} \times \text{Average CPI} \\ &= 1 \text{ ns} \times ((40\% + 20\%) \times 4 + 40\% \times 5) \\ &= 1 \text{ ns} \times 4.4 \\ &= 4.4 \text{ ns}\end{aligned}$$

In the pipelined implementation, the clock must run at the speed of the slowest stage plus overhead, which will be 1 + 0.2 or 1.2 ns; this is the average instruction execution time. Thus, the speedup from pipelining is

$$\begin{aligned}\text{Speedup from pipelining} &= \frac{\text{Average instruction time unpipelined}}{\text{Average instruction time pipelined}} \\ &= \frac{4.4 \text{ ns}}{1.2 \text{ ns}} = 3.7 \text{ times}\end{aligned}$$

Pipeline Hazards – Are a situation which cause a pipeline to stall.

- Hazards- there are situations, called hazards, that prevent the next instruction in the instruction stream from executing during its designated click cycle. Hazards reduce the performance from the ideal speedup gained by pipelining.

- There are three classes of hazards-
  - Structural hazards
  - Data hazards
  - Control hazards

Pipeline Hazards (Cont.)

- Structural hazard- arise from resource conflicts when the hardware cannot support all possible combinations of instructions simultaneously in overlapped execution.

- Data hazards- arise when an instruction depends on the result of a previous instruction in a way that is exposed by the overlapping of instructions in the pipeline.

- Control hazards- arise from the pipelining of branches and other instructions that changes the PC.

→ Structural Hazards: Means that attempts to use a resource two different way at same time.

→ Data Hazards: → Occur when there is data dependency between instruction.