Pipelining

CS2052 Computer Architecture

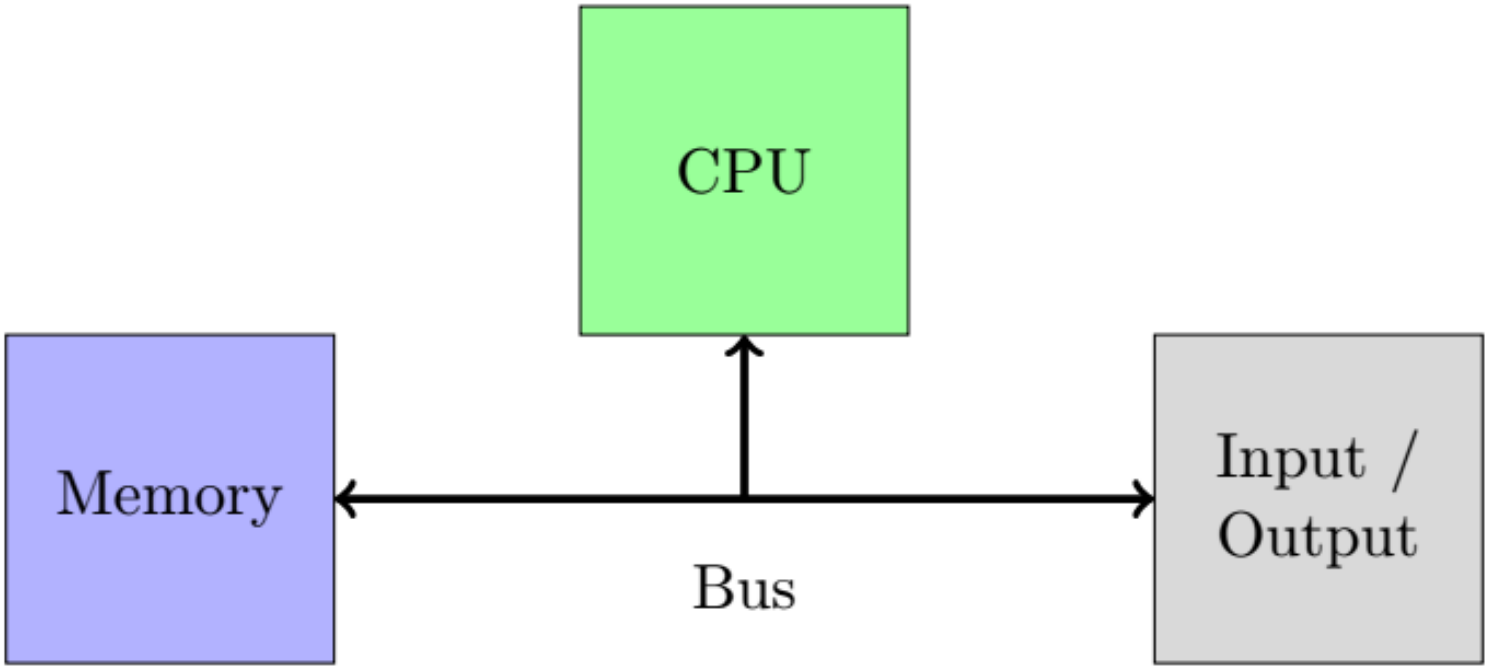
Computer Science & Engineering

University of Moratuwa

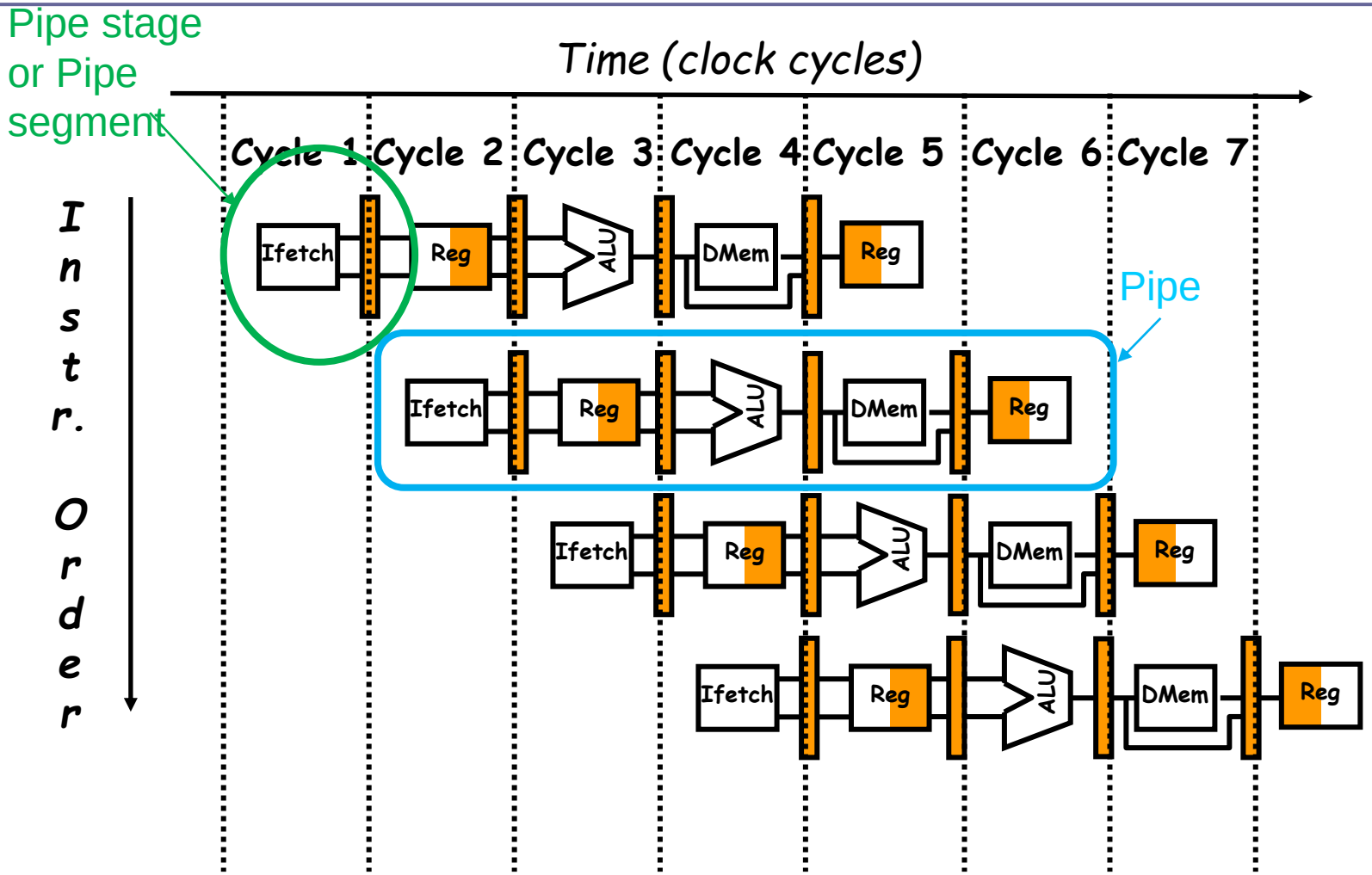
Sulochana Sooriyaarachchi

sulochanas@cse.mrt.ac.lk

Introduction



Pipelining concepts and terms



Terminology

- Throughput = number of instructions coming out of the pipe per unit time
- Processor cycle = time for moving an instruction one step down the pipeline
  - Depends on slowest stage
- Balanced pipeline = all the pipeline stages have the same duration
  - Time per instruction is given by;

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

Terminology cntd.

- Speedup
  - Time for an instruction in non pipelined method / time for an instruction in pipelined method
  - For a balanced pipeline: speedup = #number of stages

Pipelining concept

- Overlap instruction processing
- Instruction level parallelism (ILP)
- Datapath implications
- Hazards
- Performance

Pipelining with RISC architecture

- RISC characteristics
  - All operations on data apply to entire data register
  - Only operations on memory are *store* and *load*
    - memory ⟺ register
    - Can transfer data < full register
  - All instructions are typically one size and register specifiers are always in the same place
- Above leads to simplified pipeline implementation
- Example: 5-stage pipeline
  - IF→ID→EX→MEM→WB

Example Pipeline Implementation

- Stages
  - Instruction Fetch (IF)
    - Refer PC in memory, fetch the instruction from memory, update the PC to next instruction address
  - Instruction Decode/ Register Fetch (ID)
    - Fixed field decoding
    - Parallel decoding of opcode and registers
  - Execution/ Effective Address (EX)
    - ALU operates on operands (mem ref, reg-reg, ref-imm, cond branch)
  - Memory Access (MEM) – use the effective addr
  - Write Back (WB) – write results to register file

Pipeline issues

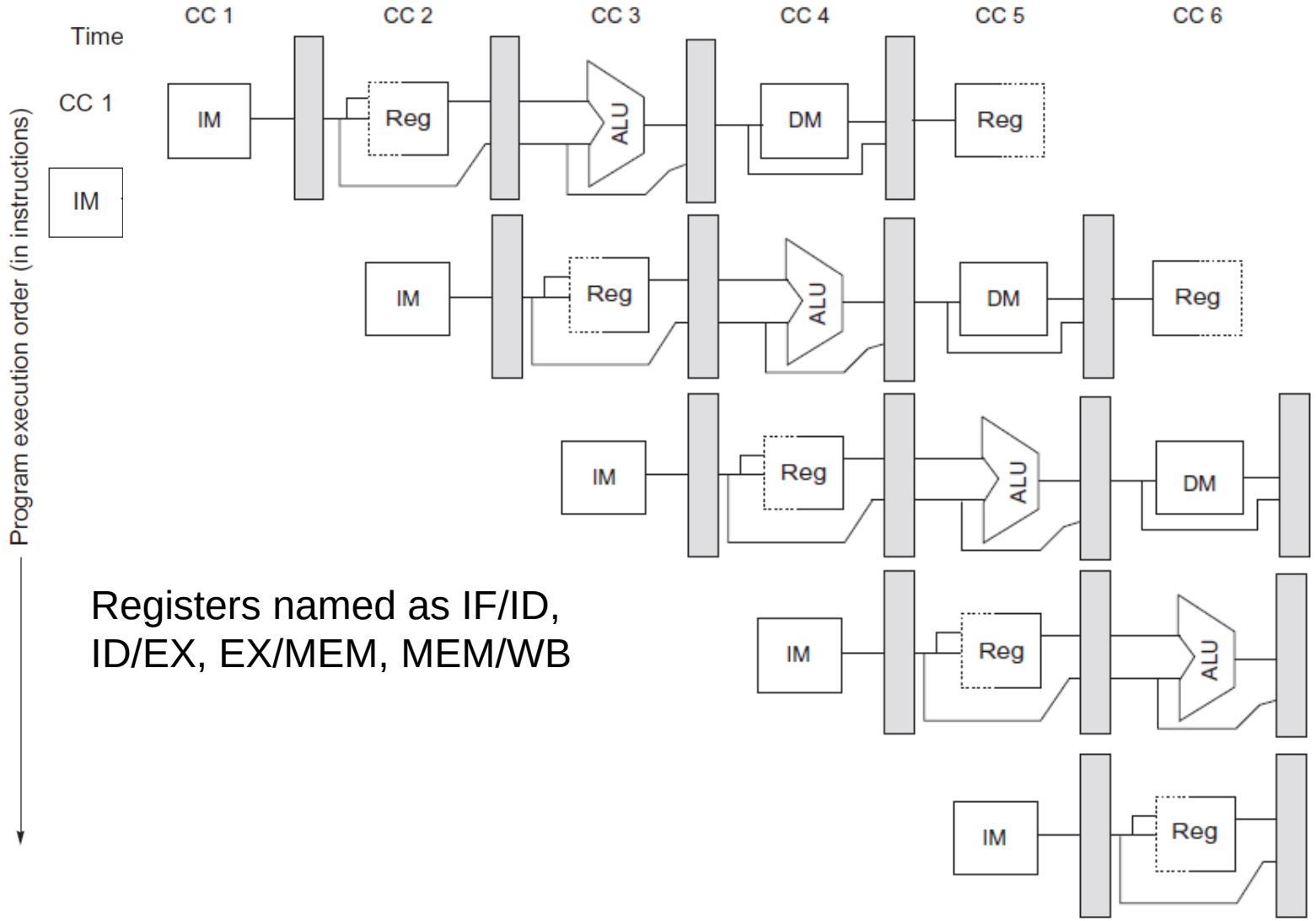
- What can go wrong?
  - Two different operations with same data path resources on the same clock cycle
  - Overlapping operations have conflicts?

	Clock number								
Instruction 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

How to omit Pipeline Issues

- Use separate instruction and data memories with separate caches
- Register file usage:
  - Reading from and writing to the same register – half of the clock cycle
- Instructions in different stages interfering with each other
  - → use pipeline registers between successive stages

Pipeline registers



Registers named as IF/ID, ID/EX, EX/MEM, MEM/WB

Performance Issues

- Pipelining increases throughput but introduces overhead to control the pipeline
- Imbalance of pipeline stages
  - Speedup depends on slowest pipeline stage
- Pipeline register delay and clock skew
  - Registers need setup time and propagation time of clock cycle
  - Clock skew – time between arrival of clock edge at two registers

In class quiz

Example Consider the unpipelined processor in the previous section. Assume that it has a 2GHz clock (or a 0.5 ns clock cycle) and that it uses four cycles for ALU operations and branches and five 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 0.1 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

$$\begin{aligned}\text{Average instruction execution time} &= \text{Clock cycle} \times \text{Average CPI}\\ &= 0.5\,\text{ns} \times [(40\% + 20\%) \times 4 + 40\% \times 5]\\ &= 0.5\ \text{ns} \times 4.4\\ &= 2.2\ \text{ns}\end{aligned}$$

Speed up

$$\frac{\text{Average instruction time unpipelined}}{\text{Average instruction time pipelined}}$$

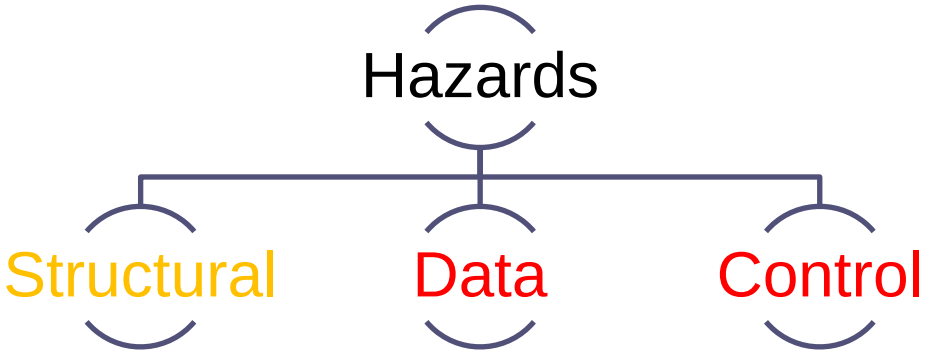
In the pipelined implementation, the clock must run at the speed of the slowes stage plus overhead, which will be 0.5 + 0.1 or 0.6 ns; this is the average instructio execution time. Thus, the speedup from pipelining is

$$\text{Speedup from pipelining} = \frac{\text{Average instruction time unpipelined}}{\text{Average instruction time pipelined}}$$

$$= \frac{2.2 \text{ ns}}{0.6 \text{ ns}} = 3.7 \text{ times}$$

Pipeline Hazards

- Hazards – situations preventing next instruction execution in designated clock cycle



- Structural hazard– resource conflict
- Data hazard – next instruction depending on result of current instruction during overlap
- Control hazards – branch and PC modifying instructions

Pipeline Stall

- During a hazard some instructions need delaying – stall
- All instructions after a stalled instruction also stop
- Instructions issued earlier than the stalled one must continue → clear the stall
- No new instructions fetched during stall

Data Hazards

Instruction i on register x

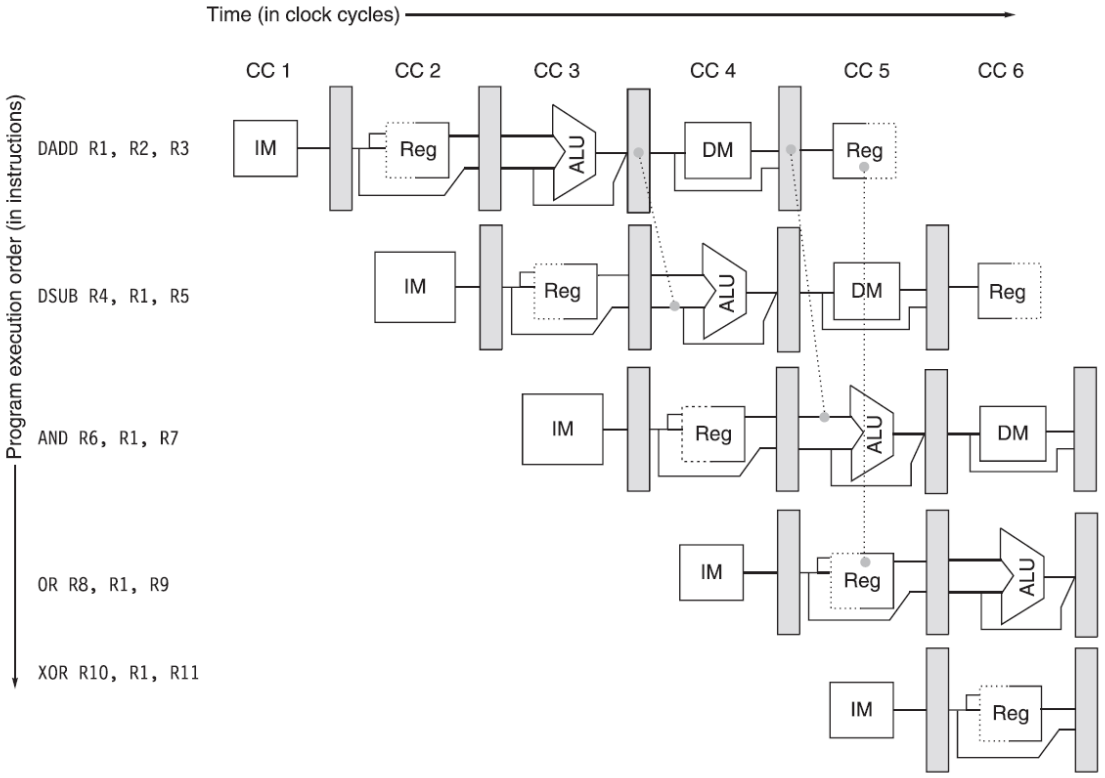
Instruction j on register x

- Read after write (RAW) – j reads before i writes
- Write after read (WAR) – i reads after j writes
- Write after write (WAW) – i writes after j writes

Minimizing Data Hazard Stalls

- *Forwarding (=bypassing = short-circuiting)*
  - Copying the pipeline register where a result is generated to where the result is used

```
add     x1,x2,x3
sub     x4,x1,x5
and     x6,x1,x7
or      x8,x1,x9
xor     x10,x1,x11
```



Branch Hazard

- Greatest performance loss
- If a branch changes PC → *taken* branch
- Other wise *untaken*
- PC changes only after ID

Branch instruction	IF	ID	EX	MEM	WB		
Branch successor		IF	IF	ID	EX	MEM	WB
Branch successor + 1				IF	ID	EX	MEM
Branch successor + 2					IF	ID	EX

Minimizing Control Hazard Stalls

- Freeze or flush the pipeline
- Predicted-untaken
- Predicted-taken
- Delayed-branch

Read appendix C in recommended text
Take home assignment – short note!

THANK YOU