

CS 203A
Advanced Computer Architecture

Lecture 5
Pipeline

Instructor: Jun Yang

Questions from Last Lecture

- What does MIPS stand for?
 - MIPS stands for Microprocessor without Interlocked Pipeline Stages. It is a company commercialized from a Stanford VLSI project in early 1980s. The project produced a processor of the same name.
- The sign extension of "unsigned" arithmetic instructions.

When does MIPS Sign Extend?

- When value is sign extended, copy upper bit to full value:
 - Examples of sign extending 8 bits to 16 bits:


```
00001010 ⇒ 00000000 00001010
10001100 ⇒ 11111111 10001100
```
- When is an immediate value sign extended?
 - Arithmetic instructions (add, sub, etc.) sign extend immediates *even for the unsigned versions of the instructions!*
 - ❖ The term "unsigned" is a misnomer. It is 32(64)-bit modulo arithmetic that does not trap on overflow.
 - Logical instructions *do not sign extend*
- Load/Store half or byte *do sign extend*, but unsigned versions do not

Outline

- MIPS64
- RISC pipeline concept (Appendix A)

Non-Pipelined MIPS64 Implementation

- Up to 5 cycles for instruction execution
- Consider an integer subset of a RISC architecture
 - ❖ load/store word, branch, and integer ALU ops.

1. Instruction fetch (IF):

$IR \leftarrow Mem[PC];$
 $New PC \leftarrow PC + 4;$

2. Instruction decode and register fetch (ID):

Decode opcode	Read rs, rt	Sign extend immediate	Sign extend offset
	$rs == 0? \quad rt == 0?$	target ← NewPC + sign extended immediate	
	$rs == rt?$	$PC \leftarrow target$	

Time line

Non-Pipelined MIPS64 Implementation (2)

3. Execute and effective address calculation (EX):

- mem. ref. op.: $ALUout \leftarrow A + Imm;$
- reg.-reg. ALU op.: $ALUout \leftarrow A op B;$
- reg. Imm. ALU op.: $ALUout \leftarrow A op Imm;$
- branch: $ALUout \leftarrow New PC + Imm$
 $Cond \leftarrow A op O;$

4. Memory access and branch completion (MEM):

- Load: $LMD \leftarrow Mem[ALUout];$
- Store: $Mem[ALUout] \leftarrow B;$
- branch: *if (Cond) then* $PC \leftarrow ALUout$
else $PC \leftarrow New PC;$

Non-Pipelined MIPS64 Implementation (3)

5. Write back (WB):

- reg.-reg. ALU op.: $rd \leftarrow ALUout;$
- reg.-imm. ALU op.: $rt \leftarrow ALUout;$
- load: $rt \leftarrow LMD;$

• Performance:

- branches take 2 cycles, branches are 12% of workload; stores take 4 cycles, 10% of workload; all other instructions take 5 cycles: CPI = 4.54.
- Not optimal either in achieving the best performance or in using the minimal amount of hardware given the performance level.

Oct. 9, 2003

Lec. 5

7

What is Pipelining

• Problem:

- A new instruction can not start until the previous one finishes.

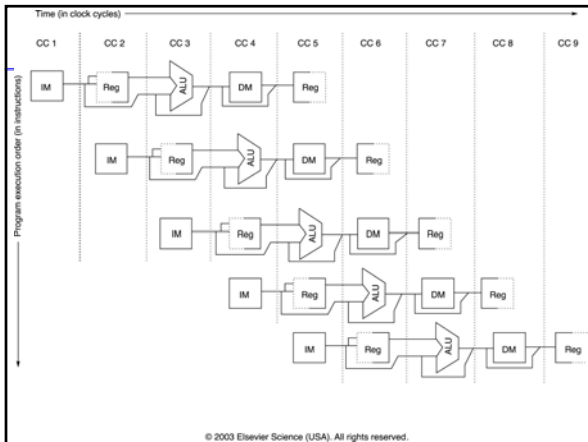
• Solution:

- Start instruction on **every** cycle, e.g. the new instruction can be fetched while the previous one is decoded - **pipeline**. Each cycle performing a specific task, e.g. IF, MEM, is called a **stage**.
- Require hardware resources being separate for each stage.

Oct. 9, 2003

Lec. 5

8



Simple RISC Pipeline

Inst. No.	1	2	3	4	5	6	7	8	9
Inst. I	IF	ID	EX	ME	WB				
Inst. I+1		IF	ID	EX	ME	WB			
Inst. I+2			IF	ID	EX	ME	WB		
Inst. I+3				IF	ID	EX	ME	WB	
Inst. I+4					IF	ID	EX	ME	WB

• New requirements:

- separate instruction and data memory access.
- Memory delivers 5x the throughput.
- 2 reads and one write of registers per cycle.
- Latches (pipeline registers) between neighboring stages to hold results of current stage, which is the input to the next stage.
 - ❖ e.g. rs of store is available after ID but used in MEM;
 - ❖ e.g. result of EX is written back in WB.

Oct. 9, 2003

Lec. 5

10

Basic Performance Issues in Pipelining

- Pipelining helps program run faster but no single instruction runs faster
 - Overhead in the control of the pipeline increases time per instruction.
 - Overhead comes from setup time of latches (input stable time before triggering a write) and clock skew (clock propagation delay between two latches)
 - Overhead puts a limit on the practical depth of a pipeline—how deep can we go?
 - Read the example on pp. A-10.
- Clock can run no faster than the slowest stage

Oct. 9, 2003

Lec. 5

11

Pipeline Hazards

• Hazards are caused by conflicts between instructions.

- They force stalls or bubbles in the pipeline: performance degradation.
- Three types:
 - ❖ Structural: resource conflicts (e.g. one memory port, unpipelined divider etc).
 - ❖ Data: dependent instructions.
 - ❖ Control: PC modifying instruction, changes control flow of program.

Oct. 9, 2003

Lec. 5

12

Pipeline Hazards (2)

- Performance impact of hazards:

$$\text{Pipeline speed-up} = \frac{\text{avg. inst. exec. time unpipelined}}{\text{avg. inst. exec. time pipelined}}$$

$$= \frac{\text{CPI unpipelined}}{\text{CPI pipelined}} \times \frac{\text{cc. unpipelined}}{\text{cc. pipelined}}$$

- CPI pipelined = Ideal CPI + pipeline stall cycles/instruction = 1 + stall cycles/inst.
- In a linear pipeline: CPI unpipelined = k = no. of stages.
- All stages are balanced and ignoring overhead:

$$\text{Speed-up} = \frac{k}{1 + \text{stall cycles/instruction}}$$

Oct. 9, 2003

Lec. 5

13

Structural Hazards

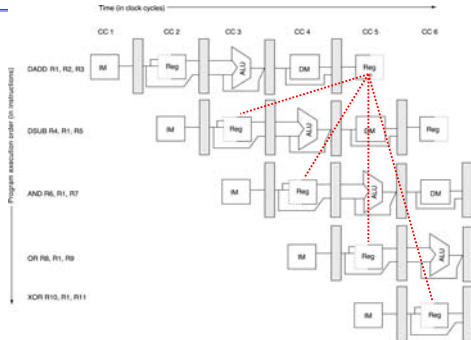
- Examples:
 - Non-pipelined or under-pipelined floating-point unit.
 - Single register write port or memory port.
- Why allow them:
 - Cost: pipelining all unit increases cost.
 - Increased latency on the common case which would not need these resources.
 - Example: non-pipelined floating-point unit, 5 cycles /MUL operation.
 - ❖ Assuming uniform distribution of MUL, machine can support 20% of instructions being MUL with no penalty.
 - ❖ If all MUL are clustered: 14% at 5 cycles each: 0.7 increase in CPI.
 - ❖ In practice, the penalty is much lower due to other hazards (data and control hazards).

Oct. 9, 2003

Lec. 5

14

Data Hazards



Oct.

© 2003 Elsevier Science (USA). All rights reserved.

5

Data Hazards (2)

True data dependence: called Read After Write (RAW) hazard (J tries to read a source before I writes it). Is most common data hazard. Cannot be eliminated.

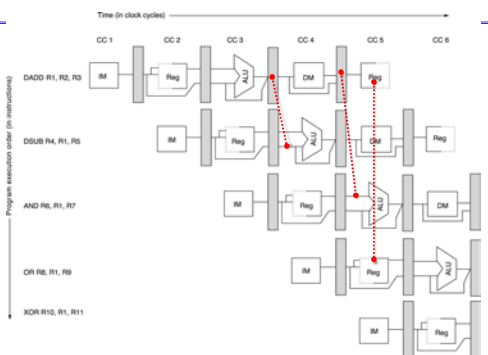
(will discuss other types of data hazards in Chapter 3)

Oct. 9, 2003

Lec. 5

16

Minimizing Data Hazard Stalls by Forwarding



17

© 2003 Elsevier Science (USA). All rights reserved.

Data Hazards Requiring Stalls

Inst. No.	1	2	3	4	5	6	7	8	9
LD R1, 0(R2)	IF	ID	EX	MEM	WB				
DSUB R4, R1, R5		IF	ID	EX	MEM	WB			
AND R6, R1, R7			IF	ID	EX	MEM	WB		
OR R8, R1, R9				IF	ID	EX	MEM	WB	

Inst. No.	1	2	3	4	5	6	7	8	9
LD R1, 0(R2)	IF	ID	EX	MEM	WB				
DSUB R4, R1, R5		IF	ID	stall	EX	MEM	WB		
AND R6, R1, R7			IF	stall	ID	EX	MEM	WB	
OR R8, R1, R9				stall	IF	ID	EX	MEM	WB

Oct. 9, 2003

Lec. 5

18

Homework 1

- Amdahl's Law
 - 1.2 on pp. 75. Due on 10/16 before class.