CS/ECE 5381/7381 Computer Architecture Spring 2023

Dr. Manikas

Computer Science

Lecture 8: Feb. 16, 2023

Assignments

- Quiz 4 – due Sat., Feb. 18 (11:59 pm)
  - Covers concepts from Module 4 (this week)
- Project 1 – due next Tue, Feb 21 (11:59 pm)

Quiz 4 Details

- The quiz is open book and open notes.
- You are allowed 90 minutes to take this quiz.
- You are allowed 2 attempts to take this quiz - your highest score will be kept.
  - Note that some questions (e.g., fill in the blank) will need to be graded manually
- Quiz answers will be made available 24 hours after the quiz due date.

Exam 1

- Exam will be administered using Lockdown Browser
- Exam format will be like the quizzes, but longer
  - 25 questions
  - You will be allowed 120 minutes (2 hours) to take the exam
  - The exam will be available from **Thursday, Feb 23 at 12 am**
  - The exam must be completed and submitted by **Saturday, Feb 25 at 11:59 pm**

Exam 1

- **Exam 1 will cover the following materials:**
  - Modules: 1 - 4
  - Quizzes: 1 - 4
  - Text: Ch. 1, App. A, App. C
- **MATERIALS ALLOWED FOR EXAM:**
  - Open book and notes, including MIPS reference data sheet
  - Calculator

LOCKDOWN BROWSER

- Recall from earlier announcement: if you have not already done this
  - Please install and test Lockdown Browser on your computer
  - Use Practice Exam to verify set up and to get familiar with Lockdown Browser
- If you have issues with Lockdown Browser, please contact SMU OIT Help Desk: https://www.smu.edu/OIT/Help

Pipelining: Basic and Intermediate Concepts

(Appendix C, Hennessy and Patterson)
Note: some course slides adopted from publisher-provided material

Outline

- C.1 Introduction
- C.2 Pipeline Hazards
- C.3 Pipelining Implementation
- C.5 Extending MIPS Pipeline to Handle Multicycle Operations

Pipeline Hazards

- Hazard –prevents next instruction from executing during designated clock cycle
  1. Structural hazards
  2. Data hazards (continue from last lecture)
  3. Control hazards

Example C.2-1

We are given the following instruction sequence for our 5-stage MIPS pipeline:

```
LW      R2, 20(R1)
AND     R4, R2, R5
OR      R8, R2, R6
ADD     R9, R4, R2
SUB     R1, R6, R7
```

We will assume that there is no data forwarding or hazard detection hardware in our pipeline. This means that you will need to identify data hazards in the instruction section and insert stalls (NOP instructions) to ensure correct program execution. Determine the data hazards and insert NOP's into the given instruction sequence as needed.

Control Hazards

- If a program branches, the PC is changed
- Methods to handle?
  1. *Freeze* or *flush* pipeline
  2. Predicted-not-taken
  3. Predicted-taken
  4. Delayed branch

Freeze or flush pipeline

- Perform either operation on pipeline until branch destination is known
  - Freeze: hold instructions
  - Flush: delete instructions
- Simple, but requires refetch of instructions

Predicted-not-taken

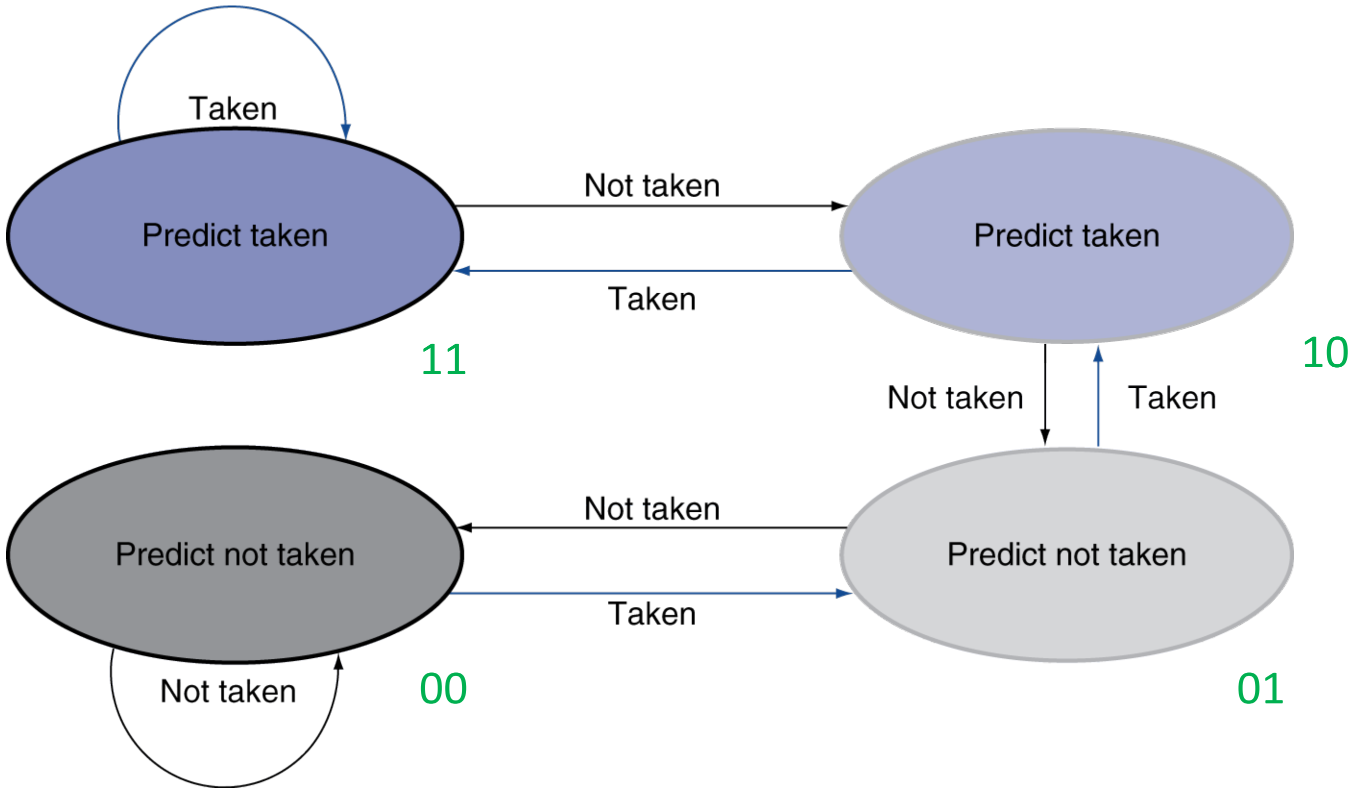
- Assume branches are *not* taken (i.e., no change in PC)
- If branch *is* taken, then refetch last instruction

Predicted-taken

- Assume branches *always* taken (PC changes)
  - Start next instruction from updated PC
- If branch is *not* taken, then refetch next instruction

2-Bit Predictor

- Only change prediction on two successive mispredictions



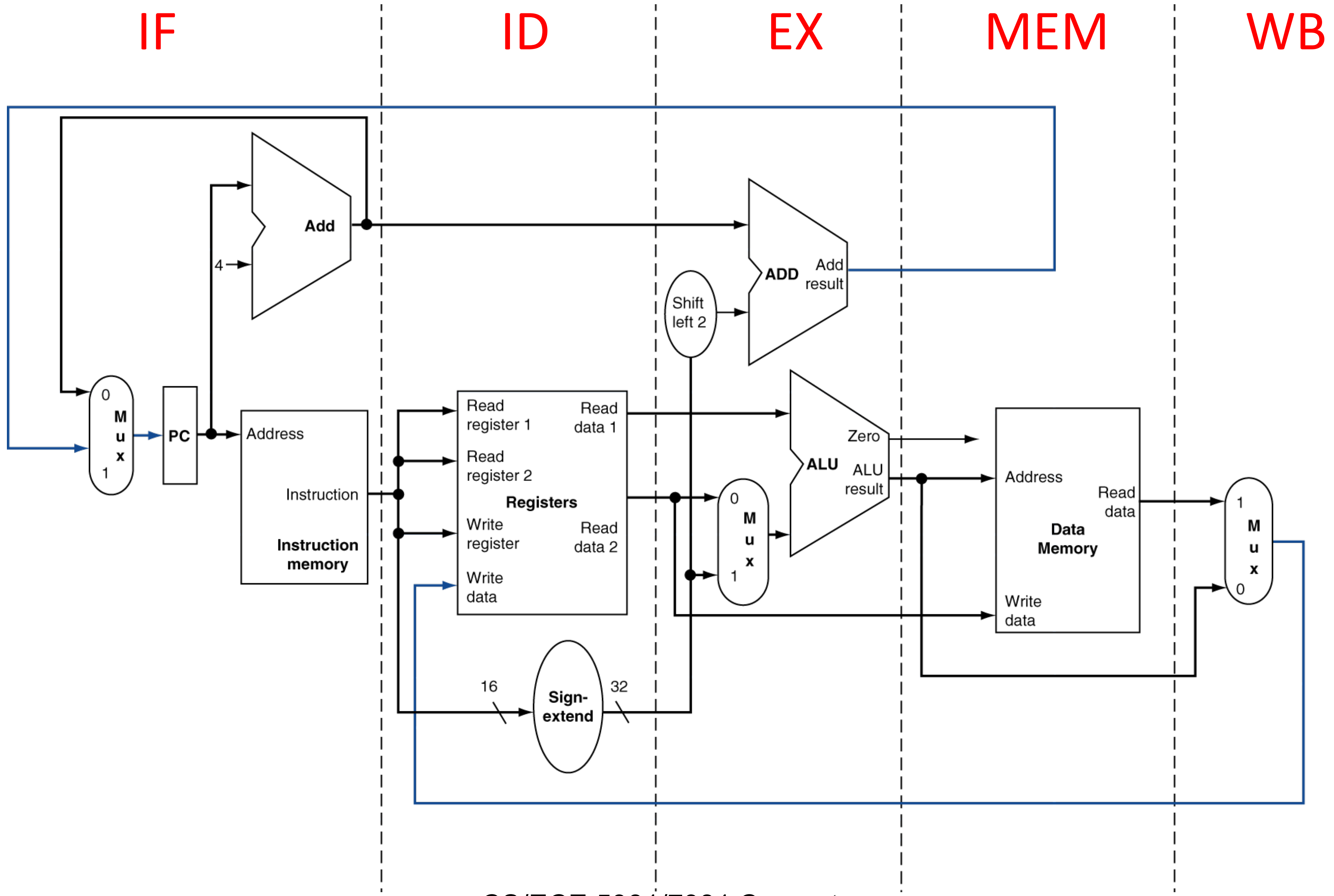
Example C.2-2

- We have the following repeating pattern of branch outcomes: NT, NT, NT, T, T
  - Where T = branch taken, NT = branch not taken

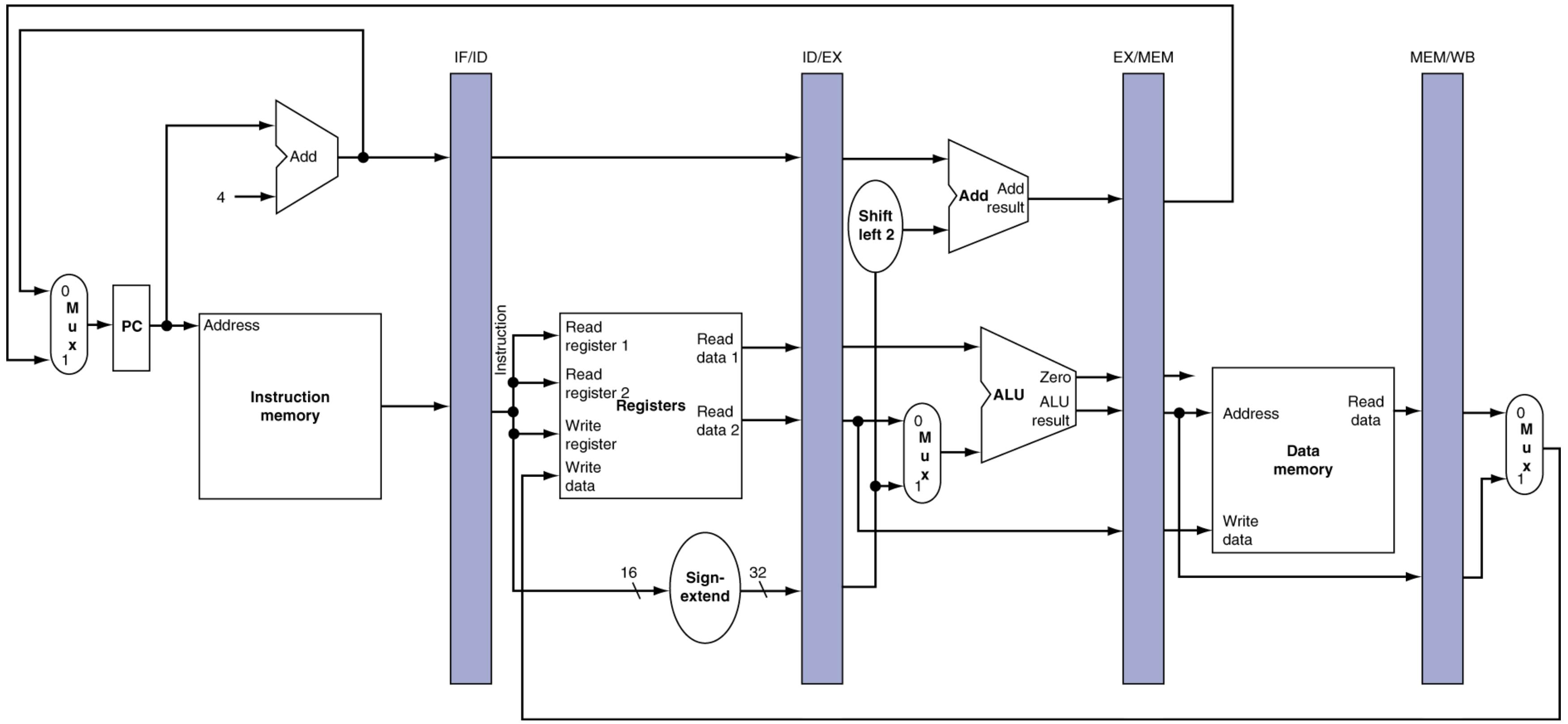
1. What is the accuracy of **always-taken** predictors for this sequence of branch outcomes?
2. What is the accuracy of **always-not-taken** predictors for this sequence of branch outcomes?
3. What is the accuracy of the **two-bit predictor** for this sequence, assuming that the predictor starts off in the bottom left state of the figure (predict not taken)?

Outline

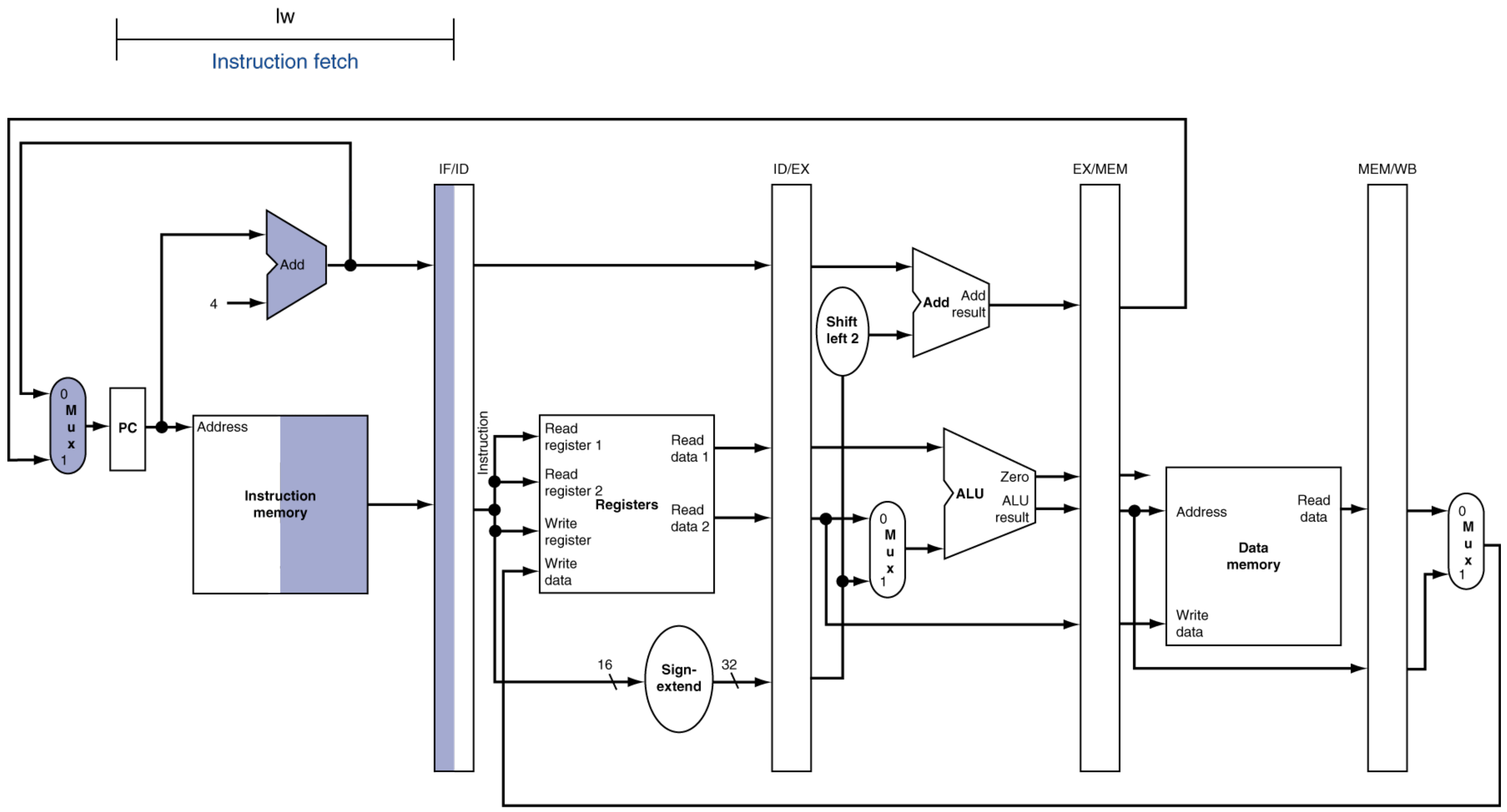
- C.1 Introduction
- C.2 Pipeline Hazards
- C.3 Pipelining Implementation
- C.5 Extending MIPS Pipeline to Handle Multicycle Operations


IF
ID
EX
MEM
WB
Add
4
M u x
0
1
PC
Address
Instruction
Instruction memory
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
ADD
Add result
ALU
Zero
ALU result
M u x
0
1
Address
Read data
Data Memory
Write data
M u x
1
0

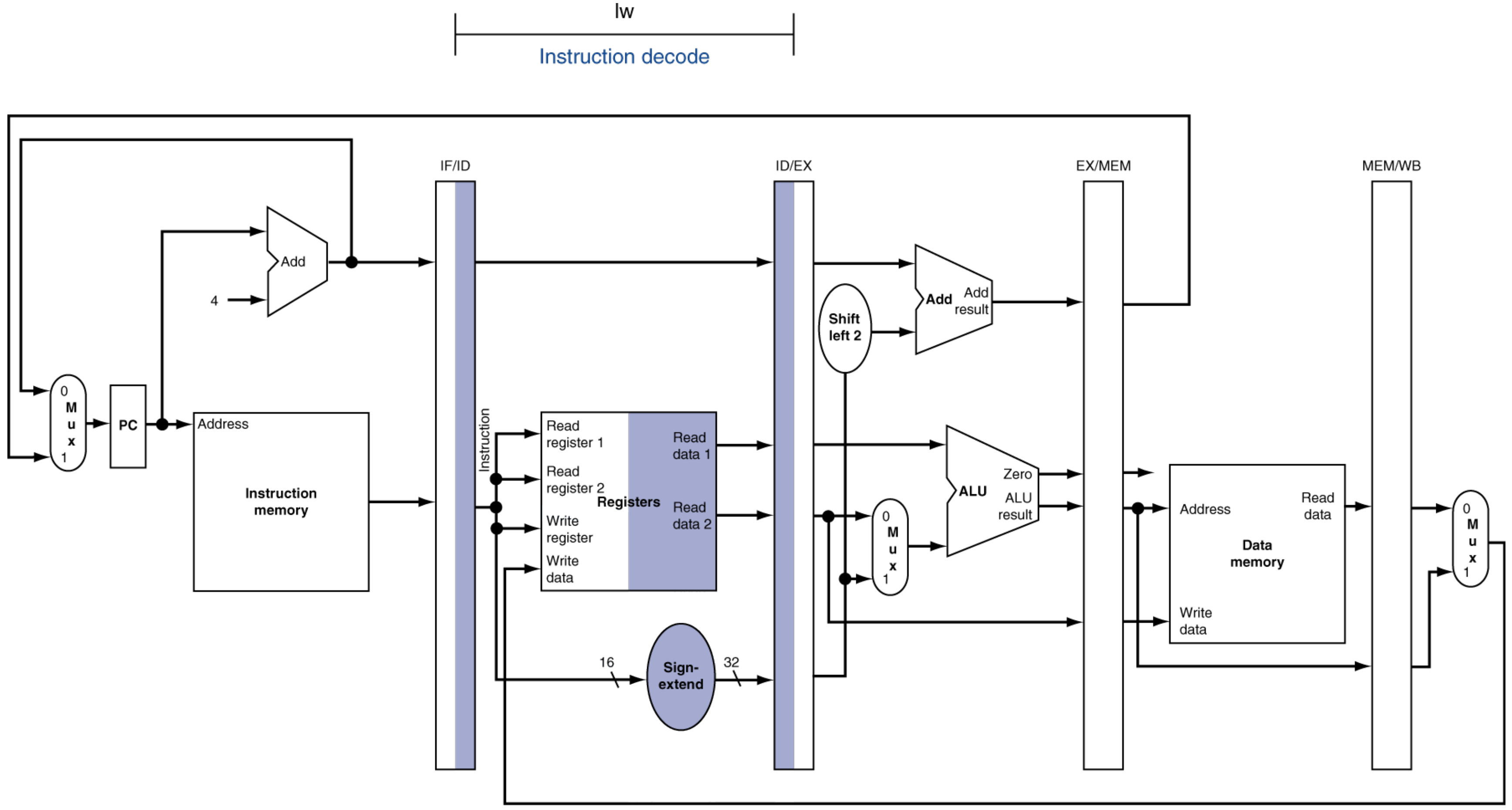
Pipeline Registers



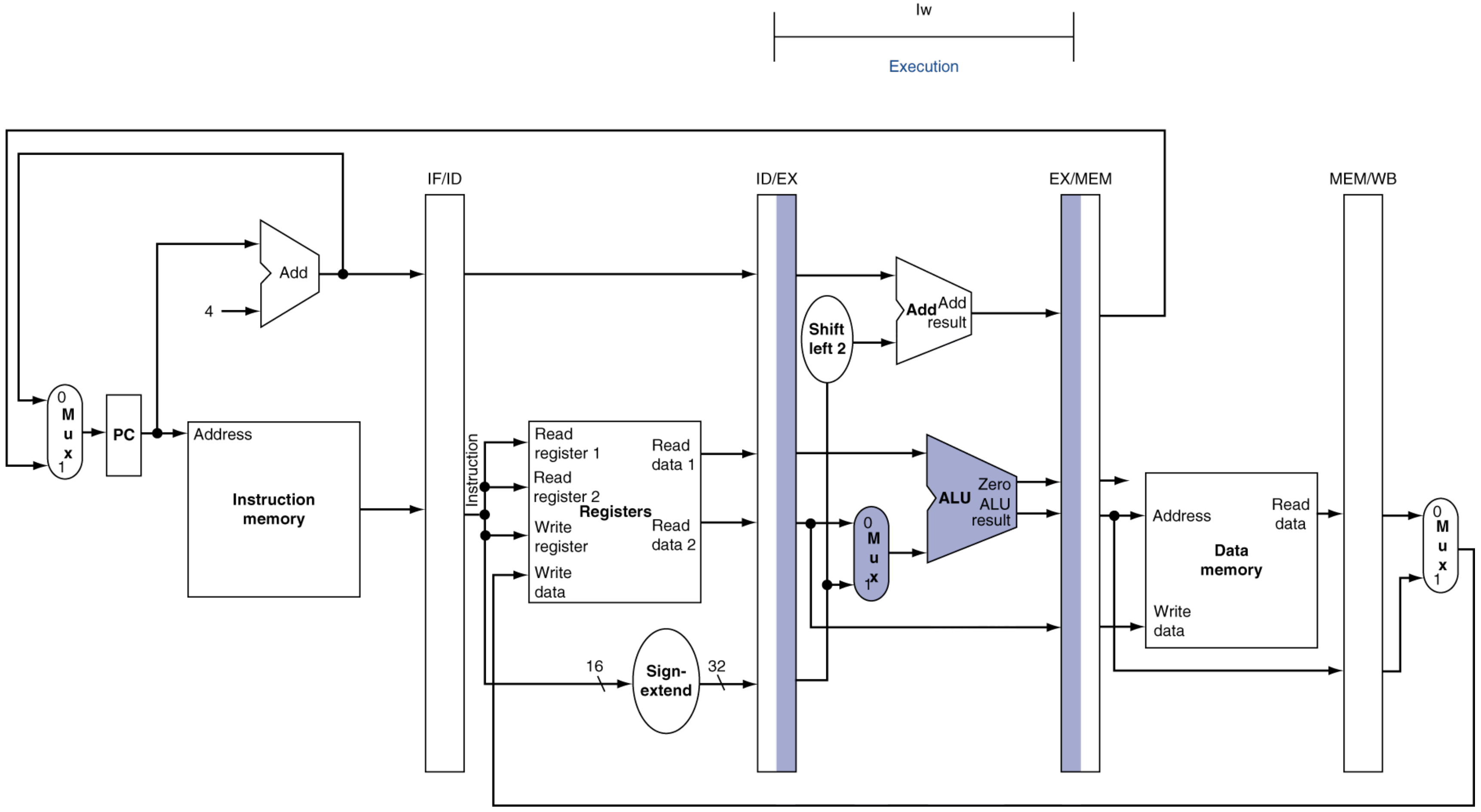
IF


lw
Instruction fetch
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Write data
Read data

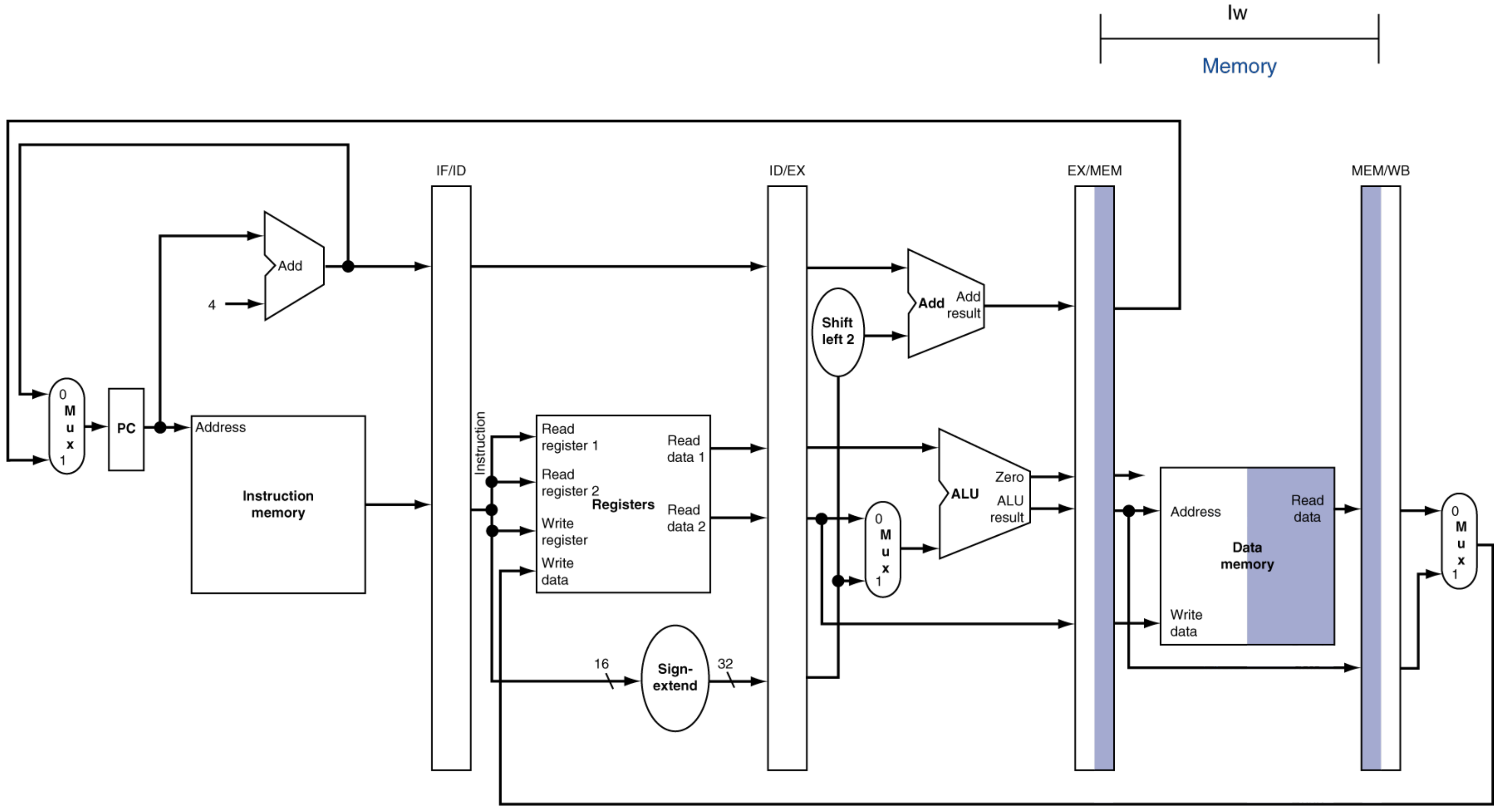
ID for Load


lw
Instruction decode
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Write data
Read data

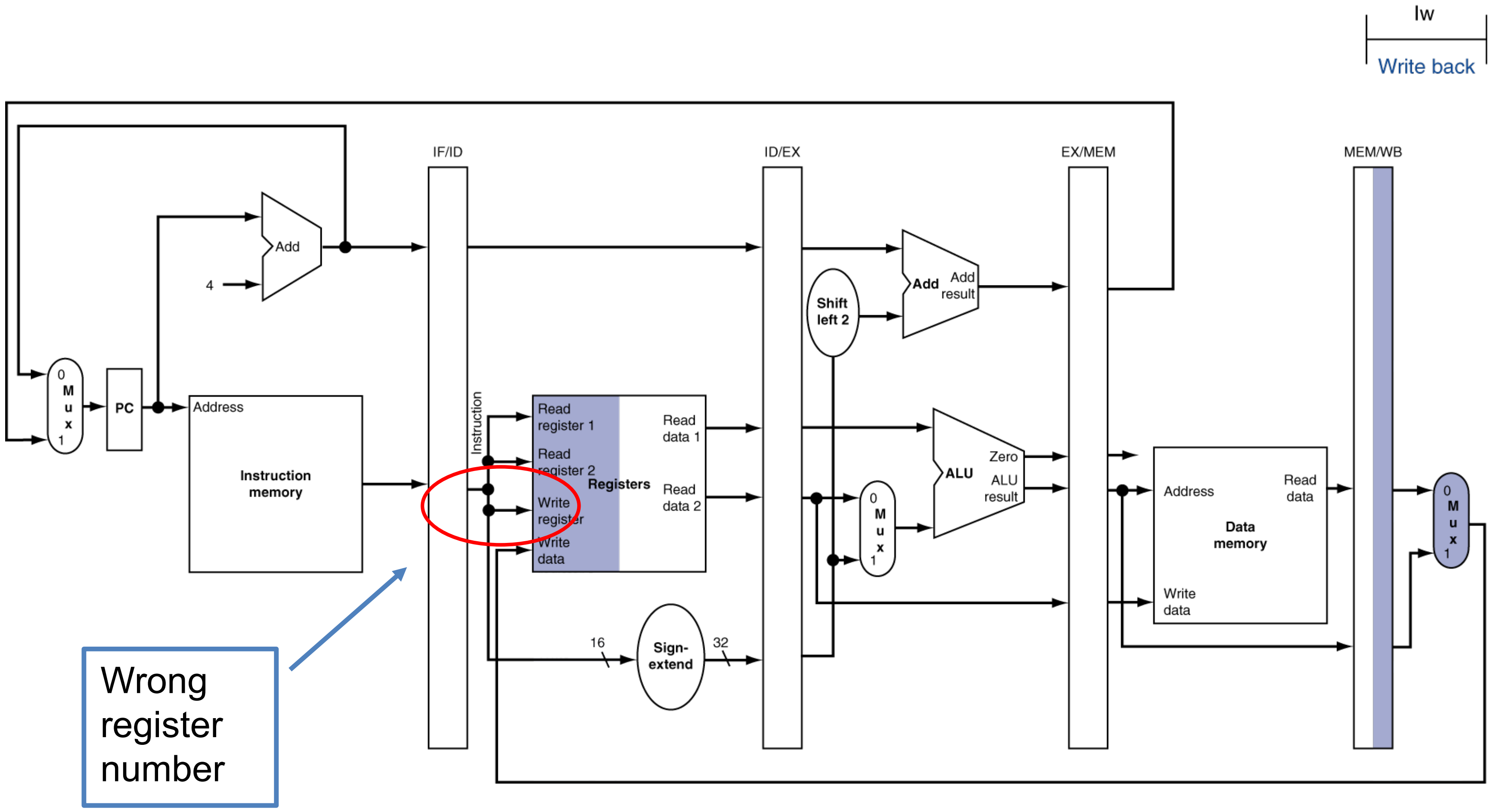
EX for Load


lw
Execution
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Write data
Read data

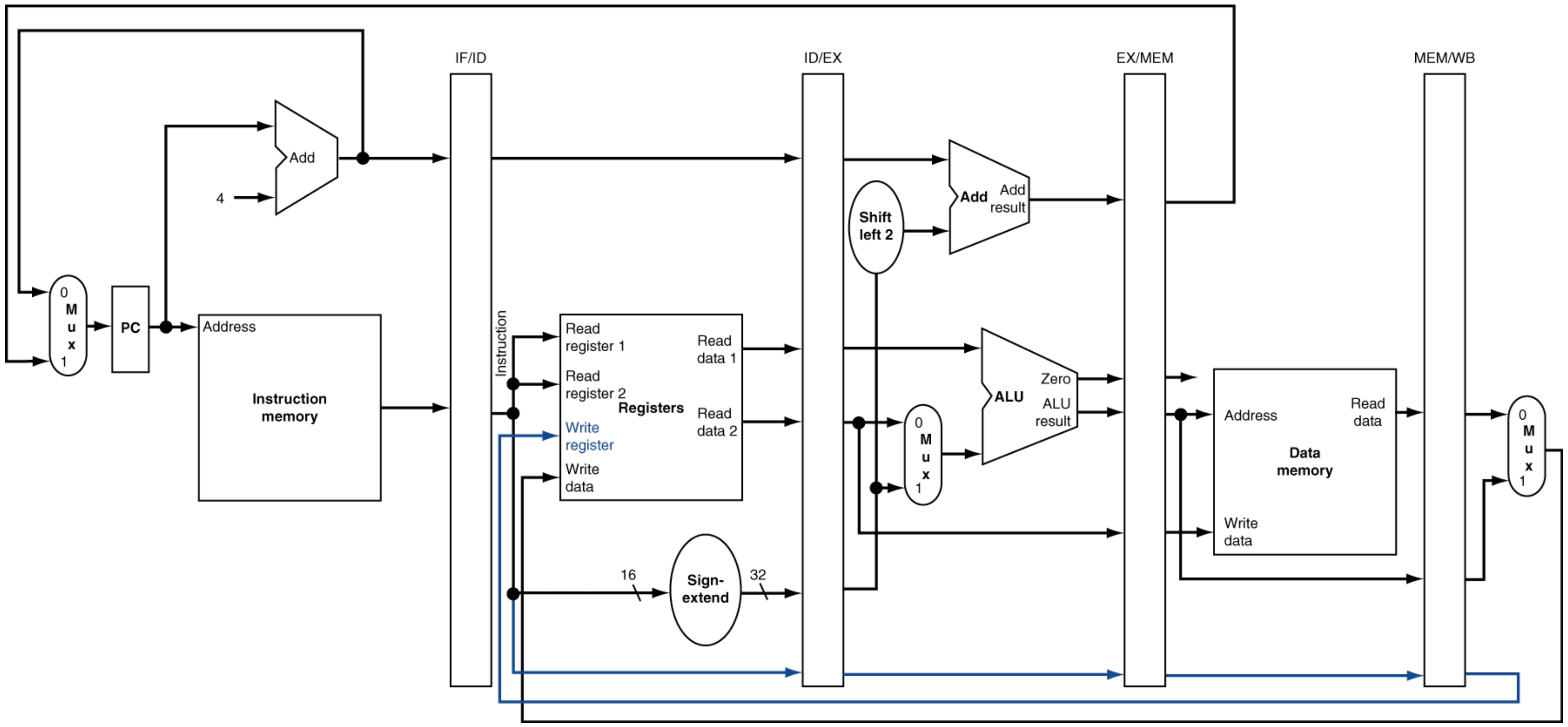
MEM for Load


lw
Memory
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data

WB for Load


lw
Write back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
Wrong register number

Corrected Datapath for Load

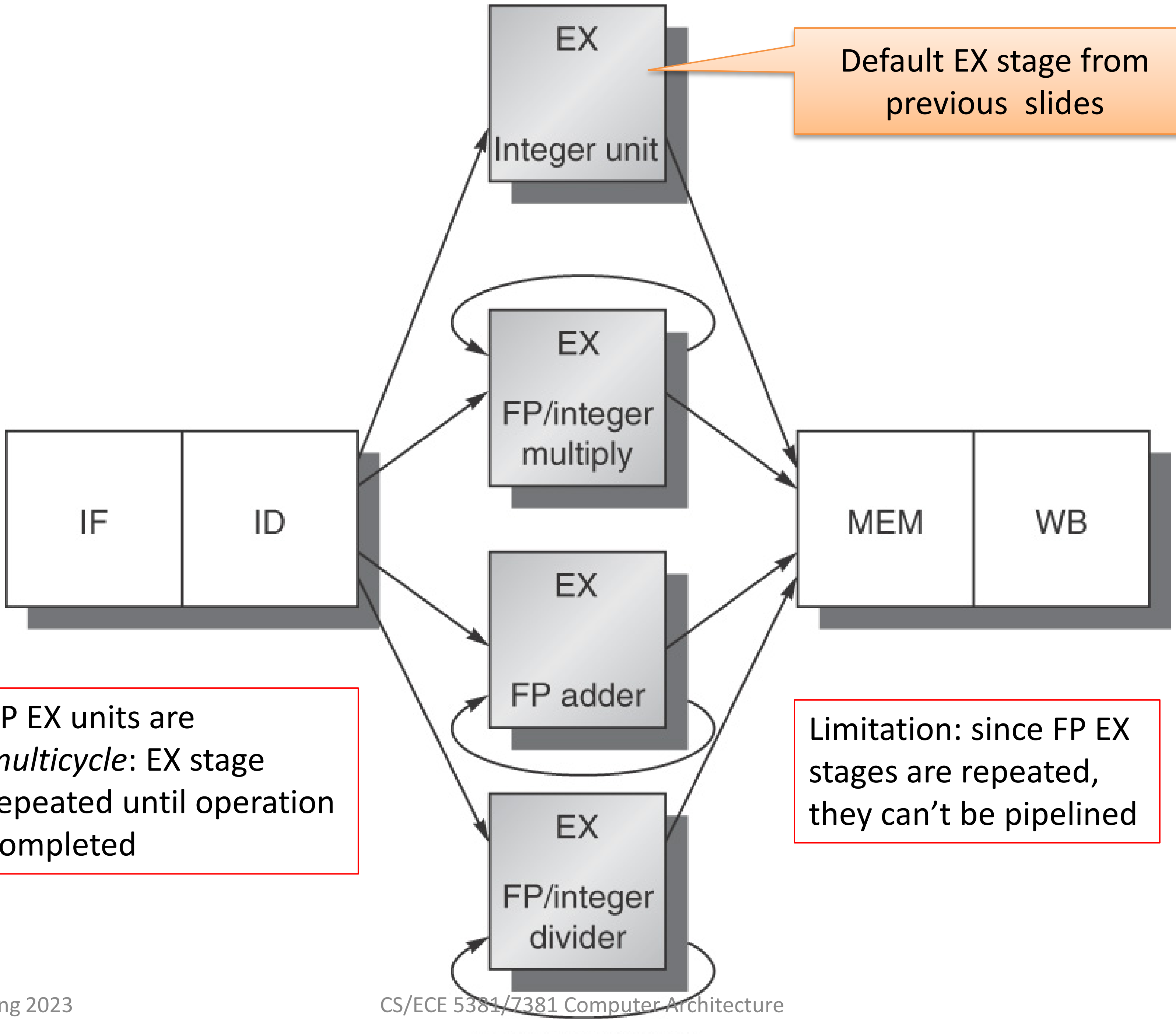


Outline

- C.1 Introduction
- C.2 Pipeline Hazards
- C.3 Pipelining Implementation
- C.5 Extending MIPS Pipeline to Handle Multicycle Operations

Extending MIPS Pipeline to Handle Multicycle Operations

- Previous assumptions for our MIPS pipeline
  - All operands are *integer*
  - Each operation (stage) takes *one* clock cycle
- Now, add *floating-point* (FP) operands
  - Operations may take *many* cycles
    - "multicycle operations"
  - Expand the EX stage to handle



Default EX stage from previous slides

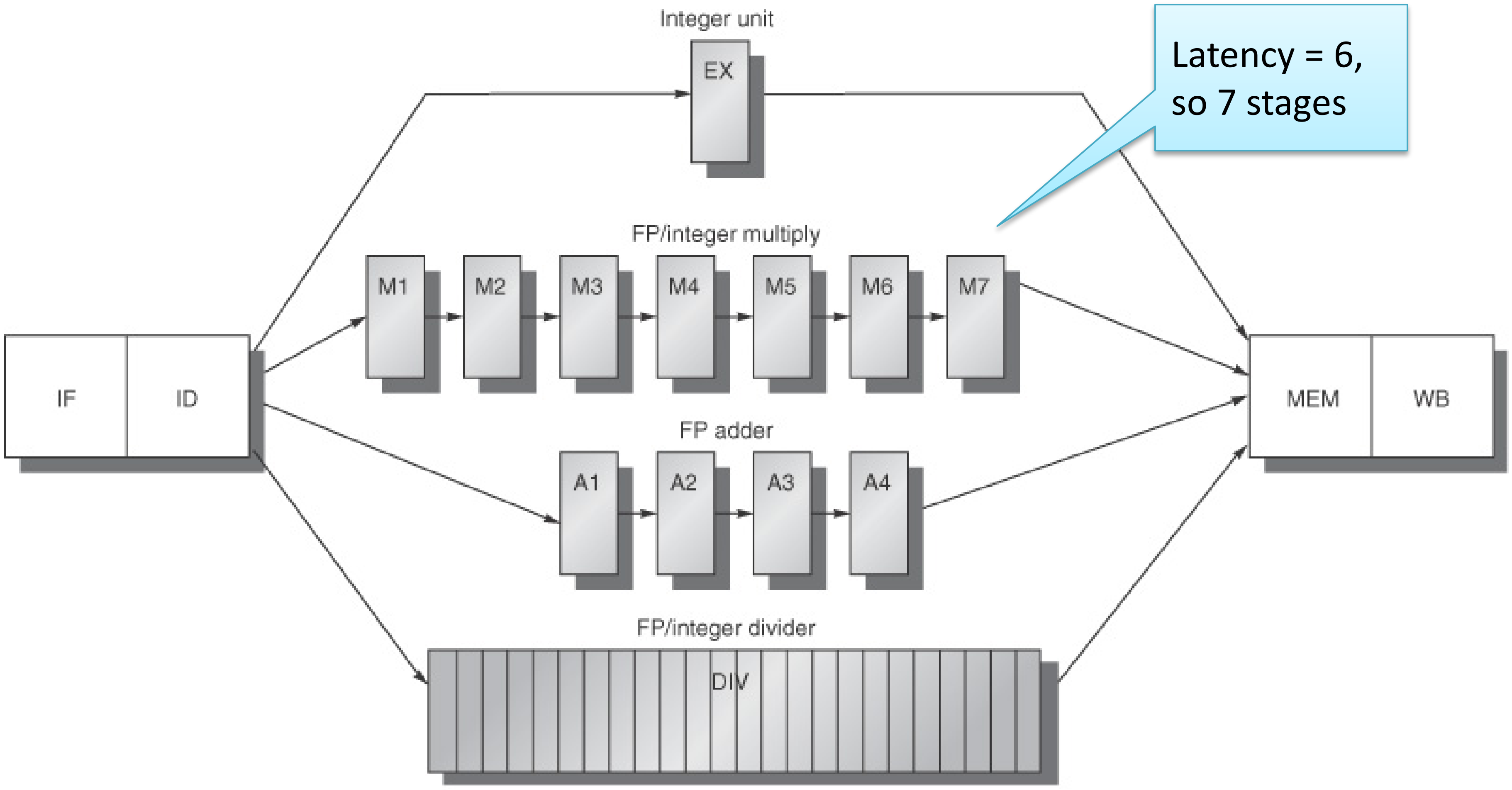
FP EX units are *multicycle*: EX stage repeated until operation completed

Limitation: since FP EX stages are repeated, they can't be pipelined

Pipelining FP Operations

- To pipeline FP operations, need to determine *number of extra cycles* (latency) for each stage
  - Integer operations have latency = 0 (no extra cycles)
  - FP multiply requires 6 extra cycles (latency = 6)
  - FP add: latency = 3
  - FP divide: latency = 24
- Thus, add an extra stage for each extra cycle

Pipelined floating-point and integer stages for MIPS



Instruction-Level Parallelism (ILP) and Its Exploitation

(Chapter 3, Hennessy and Patterson)

Note: some course slides adopted from publisher-provided material

Outline

Introduction

- Pipelining become universal technique in 1985
  - Overlaps execution of instructions
  - Exploits “Instruction Level Parallelism”
- Beyond this, there are two main approaches:
  - Hardware-based dynamic approaches
    - Used in server and desktop processors
  - Compiler-based static approaches
    - Not as successful outside of scientific applications

Instruction-Level Parallelism

- When exploiting instruction-level parallelism, goal is to maximize CPI (cycles per instruction)
  - Pipeline CPI =
    - Ideal pipeline CPI +
    - Structural stalls +
    - Data hazard stalls +
    - Control stalls
- Parallelism with basic block is limited
  - Typical size of basic block = 3-6 instructions
  - Must optimize across branches

Data Dependencies

- Instruction K is *data dependent* on instruction J if:
  - J produces a result that may be used by K, or
  - K is data dependent on instruction L, and L is data dependent on J

```
DADDUI R1,R1,#4          ;R1 ← R1 + 4

L.D    F0, 0(R1)         ;F0 ← Mem[R1]

ADD.D  F2, F1, F0        ;F2 ← F1 + F0
```

Name Dependence

- Two instructions use the same name but no flow of information
  - Not a true data dependence, *but is a problem when <u>reordering instructions</u> (to be discussed soon)*
  - *Antidependence*: instruction j **writes** a register or memory location that instruction i **reads**
    - Initial ordering (i before j) must be preserved
  - *Output dependence*: instruction i and instruction j write the same register or memory location
    - Ordering must be preserved

- To resolve, use renaming techniques

Data Hazards

- Recall data hazards due to data dependencies
- Can be addressed using *forwarding* and *stalls*
- Types of data hazards – *more specific*
  - Read After Write (RAW)
  - Write After Write (WAW)
  - Write After Read (WAR)
- <u>NOTE</u>: Read After Read (RAR) is <u>not</u> a hazard

Data Dependences

- Types of data dependences
  - Flow dependence (true data dependence – read after write)
  - Output dependence (write after write)
  - Anti dependence (write after read)

Data Dependences

- Which ones cause stalls in a pipelined machine?
  - For all of them, we need to ensure semantics of the program are correct
  - Flow dependences always need to be obeyed because they constitute true dependence on a value
  - Anti and output dependences exist due to limited number of architectural registers
    - They are dependence on a name, not a value
    - We will later see what we can do about them

Flow dependence

$r_3 \leftarrow r_1 \text{ op } r_2$

$r_5 \leftarrow r_3 \text{ op } r_4$

Read-after-Write (RAW)

Anti dependence

$r_3 \leftarrow r_1 \text{ op } r_2$

$r_1 \leftarrow r_4 \text{ op } r_5$

Write-after-Read (WAR)

Output-dependence

$r_3 \leftarrow r_1 \text{ op } r_2$

$r_5 \leftarrow r_3 \text{ op } r_4$

$r_3 \leftarrow r_6 \text{ op } r_7$

Write-after-Write (WAW)

Control Dependencies

```
ORIGINAL CODE:
Instruction J
Instruction K
IF (branch condition) THEN
{
            instruction L
            instruction M
}
```

Reordering not allowed – K is not control dependent

```
Instruction J
IF (branch condition) THEN
{
            instruction K
            instruction L
            instruction M
}
```

Reordering not allowed – L is control dependent

```
Instruction J
Instruction K
Instruction L
IF (branch condition) THEN
{
            instruction M
}
```

Outline

- 3.1 ILP Background
- 3.2 Basic Compiler Techniques for ILP
- 3.3 Branch Prediction
- 3.4 Data Hazards and Dynamic Scheduling
- 3.5 Dynamic Scheduling Algorithm
- 3.6 Hardware-Based Speculation