

# CSE 141 – Computer Architecture Spring 2005

## Lectures 12 Pipelining

Pramod V. Argade  
May 9, 2005

### CSE141: Introduction to Computer Architecture

**Instructor:** Pramod V. Argade (p2argade@cs.ucsd.edu)  
Office Hour:  
Mon. 5:00 - 6:00 PM (AP&M 5218)

**TAs:**

Baris Arslan: barslan@cs.ucsd.edu  
Chris Roedel: croedel@cs.ucsd.edu  
Raid Ayoub: rayoub@cs.ucsd.edu  
Leo Porter: leporter@cs.ucsd.edu

**Textbook:** Computer Organization & Design  
The Hardware Software Interface, 3<sup>rd</sup> Edition.  
Authors: Patterson and Hennessy

**Web-page:** <http://www.cse.ucsd.edu/classes/sp05/cse141>

# Announcements

- **Reading Assignment**

Chapter 6: Enhancing Performance with Pipelining

– 6.1 - 6.3

- **Homework 5:**

– 6.1, 6.2, 6.3, 6.6, 6.14, 6.17, 6.21, 6.22

- **Quiz**

**When:** Mon., May 16th, First 10 minutes of the class

**Topic:** Pipeline Hazards, Chapter 6    **Need:** Paper, pen

# Course Schedule

Lecture #	Date	Day	Lecture Topic	Quiz Topic	Homework Due
1	3/28	Monday	Introduction, Ch. 1	-	-
2	3/30	Wednesday	Performance, Ch. 4	-	-
3	4/4	Monday	ISA, Ch. 2	Performance	#1
4	4/6	Wednesday	Arithmetic, Ch. 3	-	-
5	4/11	Monday	Arithmetic, Ch. 3	ISA	#2
6	4/13	Wednesday	Single cycle CPU, Ch. 5	-	-
7	4/18	Monday	Single cycle CPU, Ch. 5	Arithmetic	#3
8	4/20	<b>Wednesday</b>	<b>Multi-cycle CPU, Ch. 5</b>	-	-
9	4/25	Monday	Multi-cycle CPU, Ch. 5	Single Cycle CPU	#4
10	4/27	Wednesday	Review for the Midterm	-	-
	5/2	<b>Monday</b>	<b>Mid-term Exam Center 101</b>	-	-
11	5/4	Wednesday	Exceptions, Ch. 5 and Pipelining, Ch. 6	-	-
12	5/9	<b>Monday</b>	<b>Pipelining, Ch. 6</b>	-	-
13	5/11	Wednesday	Data and control hazards, Ch. 6	-	-
14	5/16	Monday	Data and control hazards, Ch. 6	Pipeline Hazards	#5
15	5/18	Wednesday	Memory & cache design, Ch. 7	-	-
16	5/23	Monday	Memory & cache design, Ch. 7	Cache	#6
17	5/25	Wednesday	Virtual Memory & cache design, Ch. 7	-	-
No Class	5/30	Monday	Memorial Day Holiday	-	-
18	6/1	Wednesday	Course Review	-	-
	6/10	<b>Friday</b>	<b>Final Exam</b>	-	-

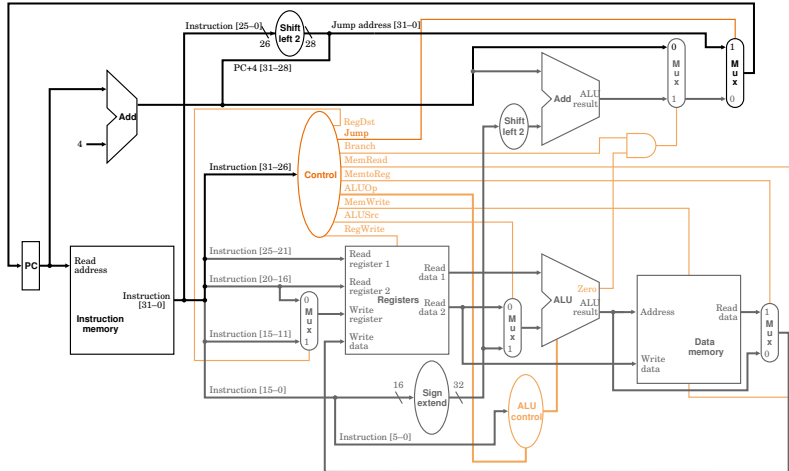
## MIPS ISA and Pipelining

- All instructions are the same length
  - Easier to fetch from instruction memory
  - Easier to decode in second stage
- Only a few instruction formats
  - Register field location(s) fixed
  - Operand fetch and instruction decode in parallel
- Load/store architecture
  - Memory operands appear only in load and store instructions
  - Execute stage calculates memory address and result for R-type
- Operands must be aligned in memory
  - Single memory data transfer requires single memory access
- Following instructions to be implemented
  - LW, SW, ADD, SUB, AND, OR, SLT, BEQ

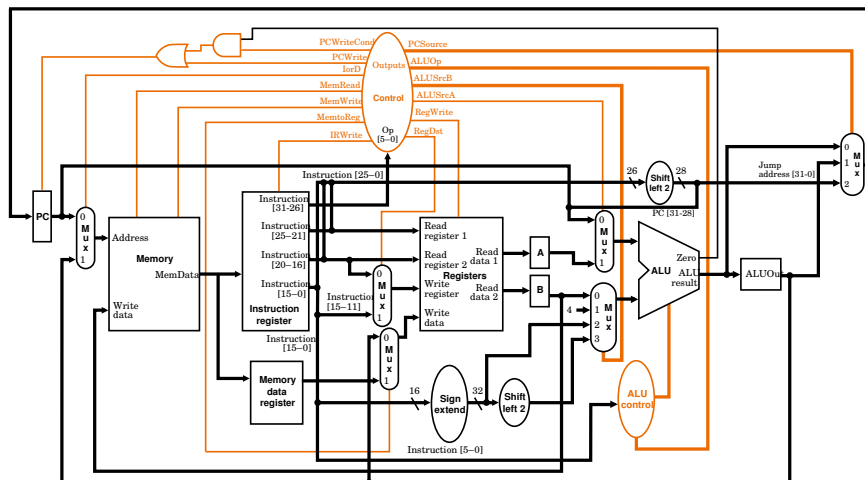
## Pipelining Challenges

- Hazards: Situation where next instruction cannot execute
  - Structural hazards:
    - Suppose we had only one memory for instructions and data
  - Control hazards:
    - Need to worry about branch instructions
  - Data hazards:
    - An instruction depends on the result of preceding instruction
- We'll talk about modern processors and what really makes it hard:
  - Exception handling
  - Trying to improve performance with out-of-order execution, etc.

# Review: Single-cycle CPU

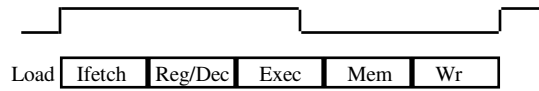


# Review: Multi-cycle CPU

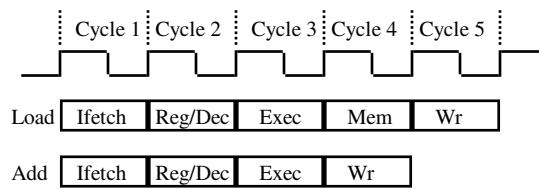


# Review -- Instruction Latencies

## •Single-Cycle CPU

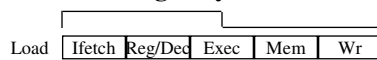


## •Multiple Cycle CPU

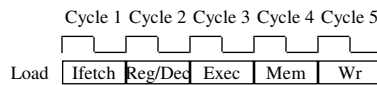


# Instruction Latencies and Throughput

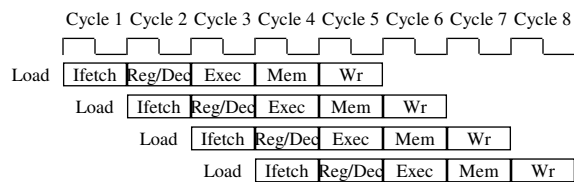
## •Single-Cycle CPU



## •Multiple Cycle CPU



## •Pipelined CPU



## Pipeline Performance Considerations

- CPU throughput = Instructions Per Cycle (IPC)
  - Number of instructions completed per cycle =  $1/\text{CPI}$
- Execution Time = (Instruction Count) \* CPI \* (Cycle Time)
- Complexity has a cost
  - e.g., Register overhead
  - Uneven stage latencies
- Pipeline clock cannot run faster than
  - Slowest pipeline stage
- Pipeline overhead
- Can't always keep the pipeline full
  - Why not?

## Pipeline Stages

IF: Instruction fetch

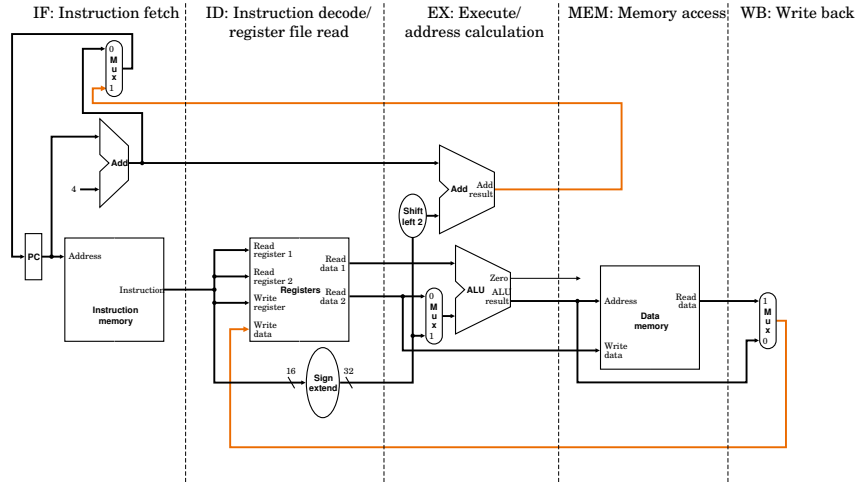
ID: Instruction decode and register fetch

EX: Execution and effective address calculation

MEM: Memory access

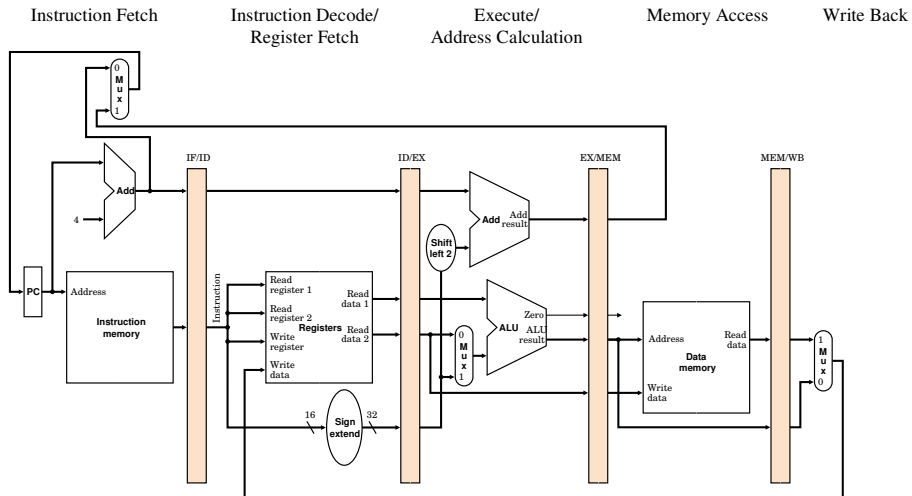
WB: Write back

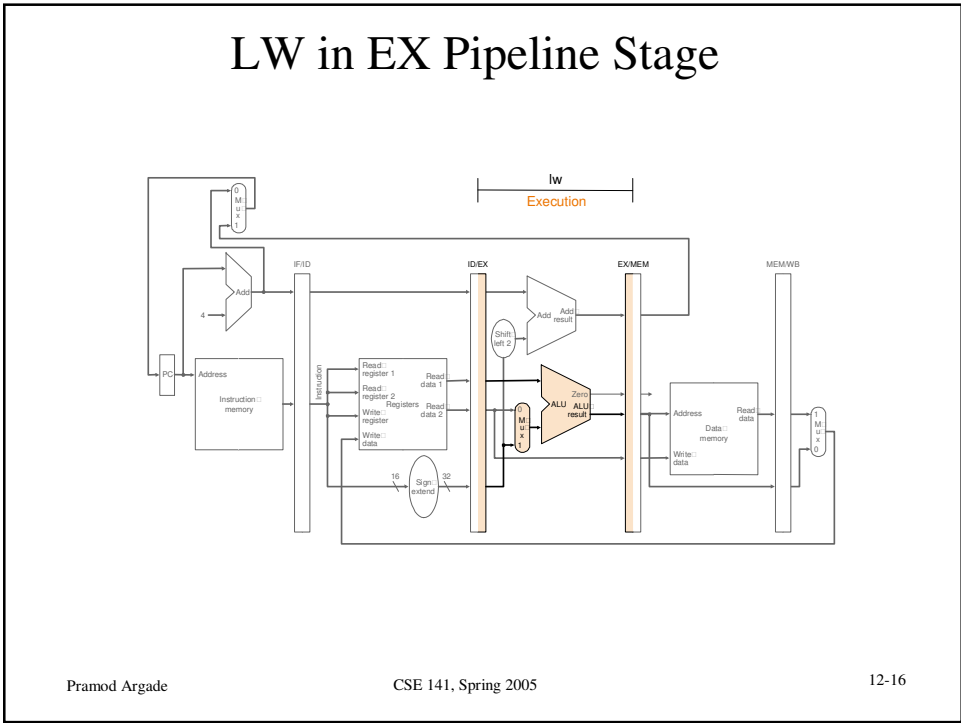
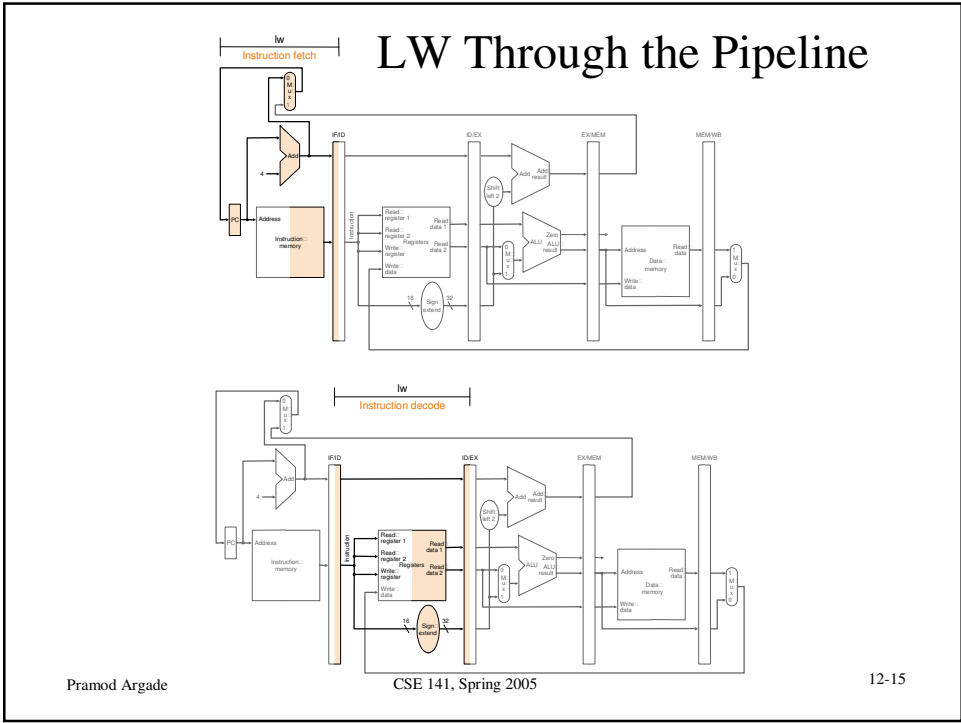
# Pipelining: Basic Idea

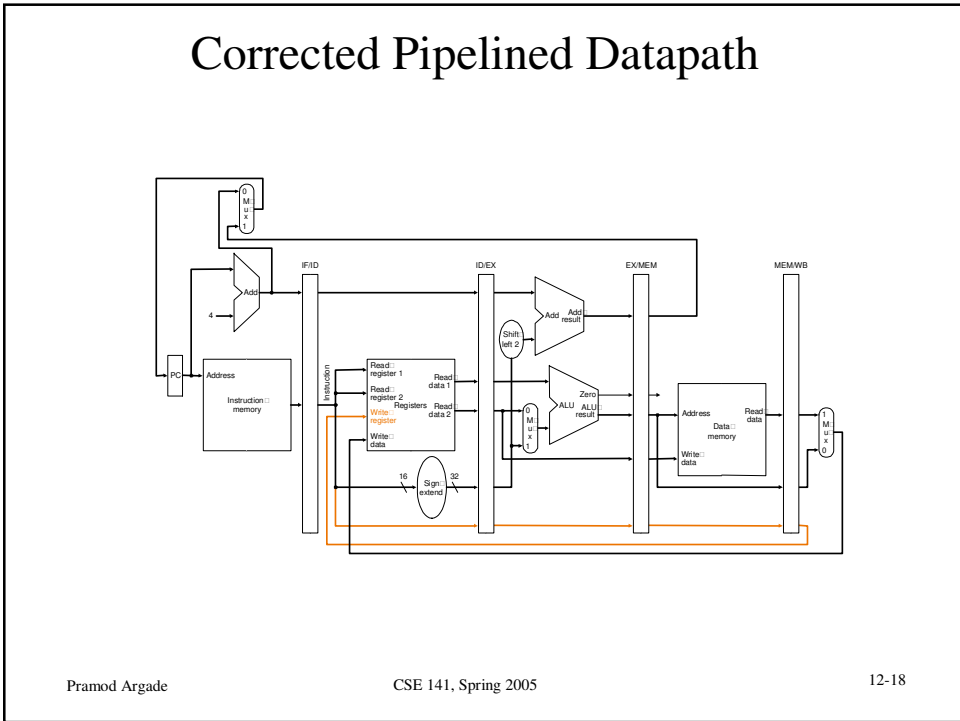
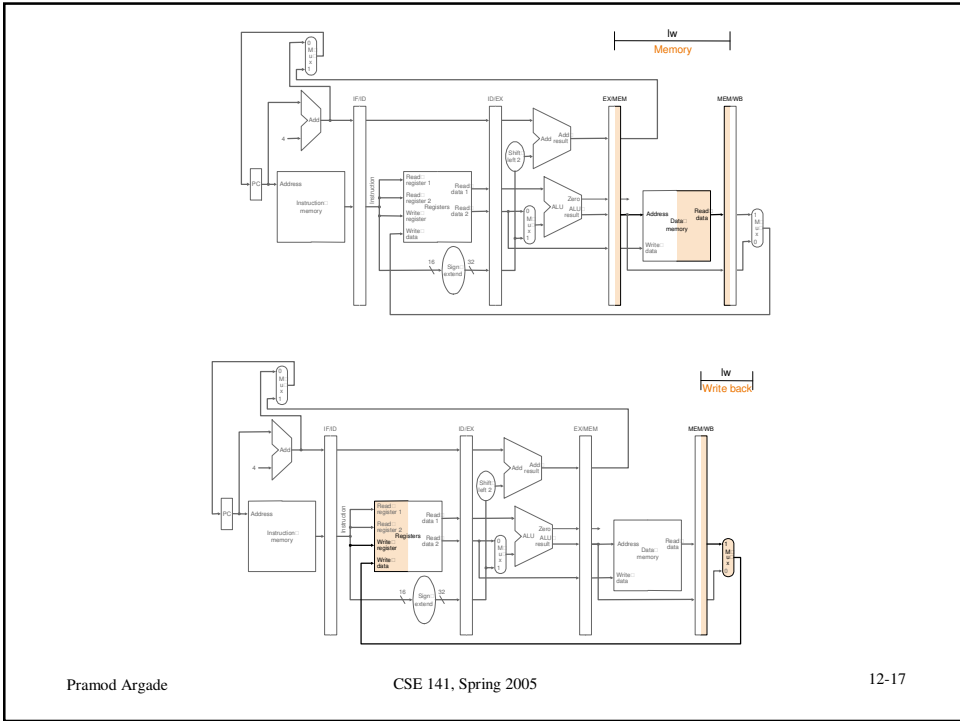


*What do we need to add to actually split the datapath into stages?*

# Pipelined Datapath



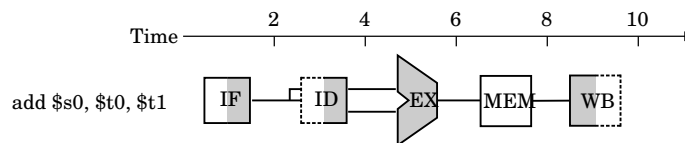




## Observations

- Instructions advance from one stage to another every clock
- Instructions and data move from left to right
  - Exceptions
    - WB stage writes to register file (Potential data hazard)
    - PC = Branch address from Mem stage (Potential control hazard)
- No registers in WB stage
  - Write registers already exist

## Graphical Representation of a Pipeline

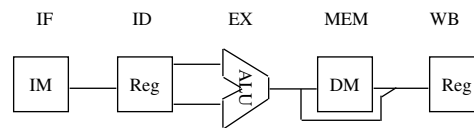


- Shading indicates that the element is used by the instruction
  - e.g. ADD instruction does not use MEM
- Shading in left half means that the element is written in that stage
- Shading in the right half means that the element is read in that stage

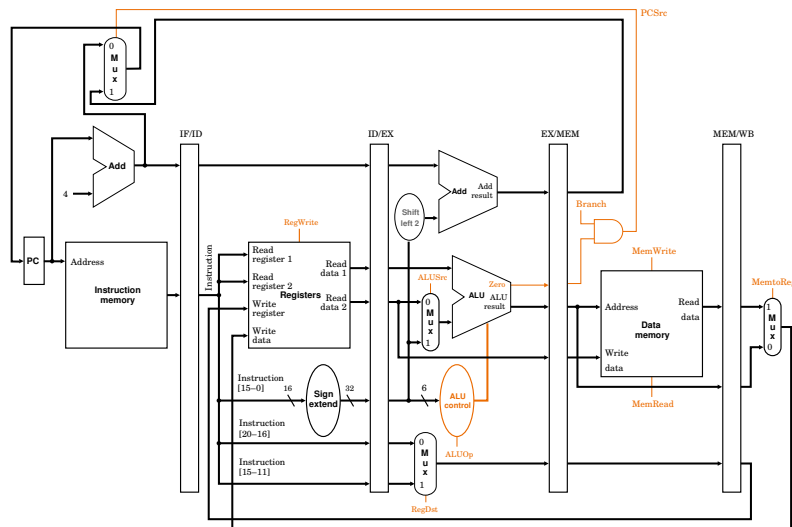


# Pipeline Principles

- All instructions that share a pipeline must have the same stages in the same order
  - Therefore, *add* does nothing during Mem stage
  - *SW* does nothing during WB stage
- All intermediate values must be registered each cycle
- There is no functional block reuse



# Pipeline with Control



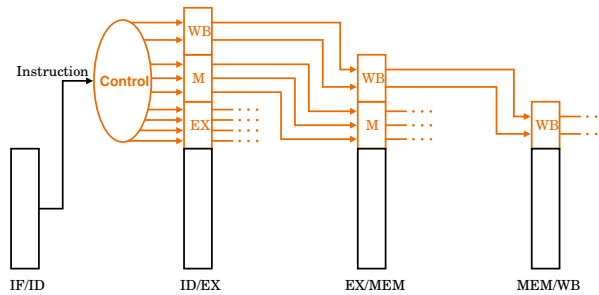
# Pipeline control

- We have 5 stages. What needs to be controlled in each stage?
  - Instruction Fetch and PC Increment
  - Instruction Decode / Register Fetch
  - Execution
  - Memory Stage
  - Write Back
- Centralized control?
  - Too complicated?
- Better approach
  - Control embedded in the pipeline
  - Compute control information in decode stage
    - Pass it along through pipeline registers

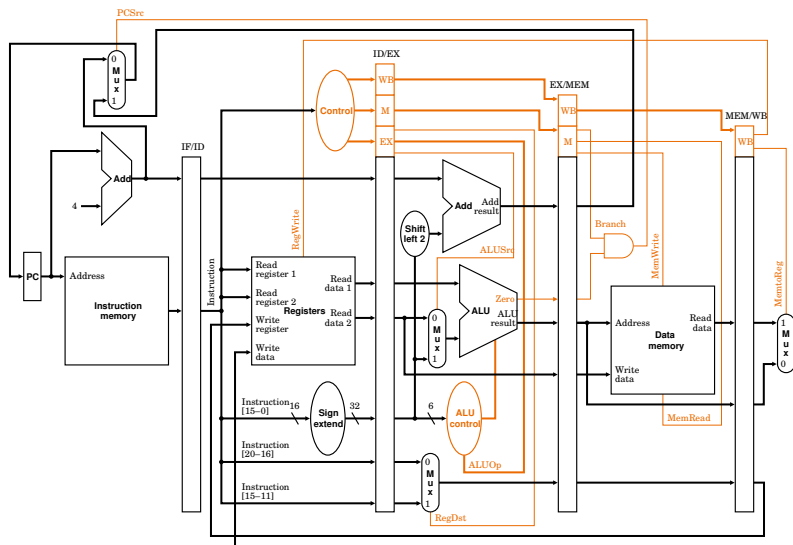
# Pipeline Control

- Use combinational Logic!
  - Signals generated once, but follow instruction through the pipeline

Instruction	Execution/Address Calculation stage control lines				Memory access stage control lines			Write-back stage control lines	
	Reg Dst	ALU Op1	ALU Op0	ALU Src	Branch	Mem Read	Mem Write	Reg write	Mem to Reg
R-format	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X



# Pipelined Datapath and Control



Pramod Argade

CSE 141, Spring 2005

12-27

## Would our pipeline design work in any case?

- What happens when...
  - add \$3, \$10, \$11
  - lw \$8, 1000(\$3)
  - sub \$11, \$8, \$7

Pramod Argade

CSE 141, Spring 2005

12-28

# Announcements

- **Reading Assignment**

Chapter 6: Enhancing Performance with Pipelining

– 6.1 - 6.3

- **Homework 5:**

– 6.1, 6.2, 6.3, 6.6, 6.14, 6.17, 6.21, 6.22

- **Quiz**

**When:** Mon., May 16th, First 10 minutes of the class

**Topic:** Pipeline Hazards, Chapter 6   **Need:** Paper, pen