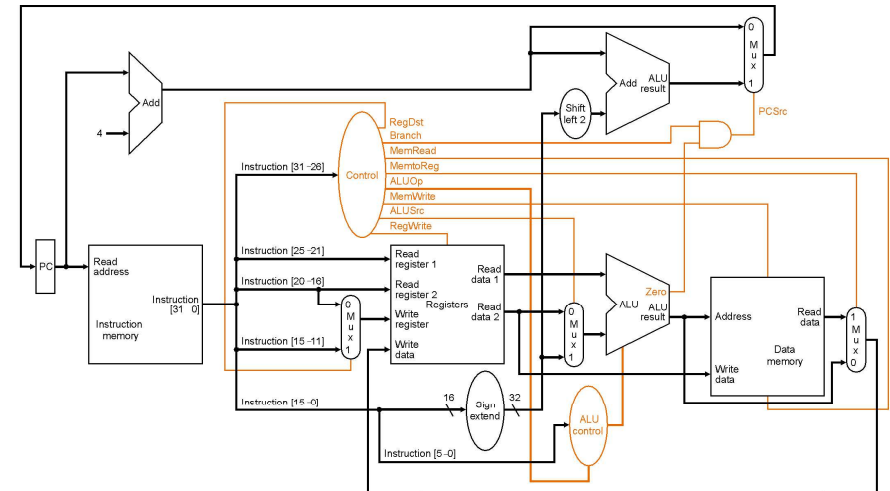


Designing a Pipelined CPU

CSE 141, S2'06

Jeff Brown

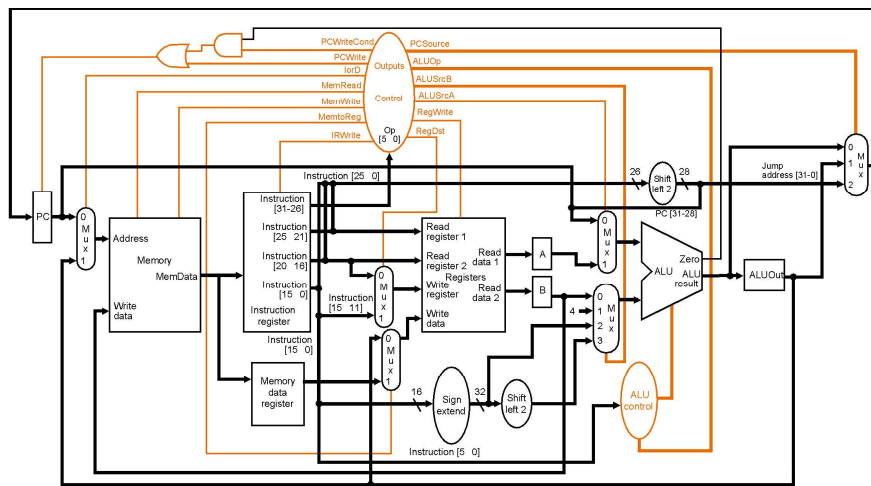
Review -- Single Cycle CPU



CSE 141, S2'06

Jeff Brown

Review -- Multiple Cycle CPU

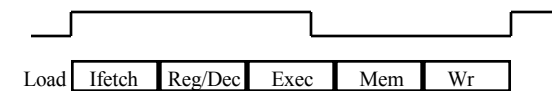


CSE 141, S2'06

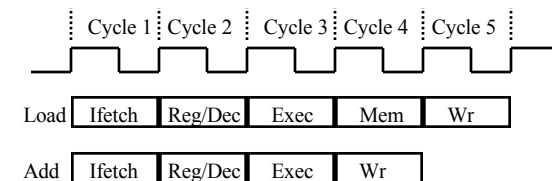
Jeff Brown

Review -- Instruction Latencies

•Single-Cycle CPU



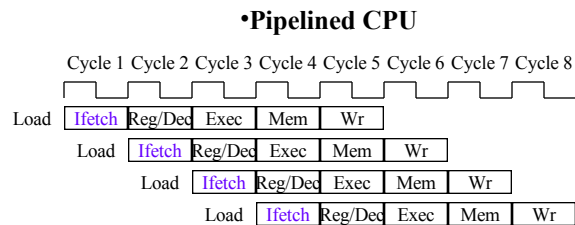
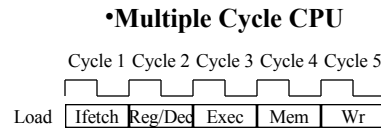
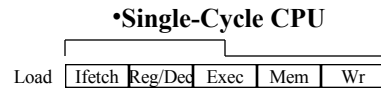
•Multiple Cycle CPU



CSE 141, S2'06

Jeff Brown

Instruction Latencies and Throughput



CSE 141, S2'06

Jeff Brown

Pipelining Advantages

- Higher *maximum* throughput
- Higher *utilization* of CPU resources
- But, more complicated *datapath*, more complex control(?)

CSE 141, S2'06

Jeff Brown

Pipelining Advantages

<u>CPU Design Technology</u>	<u>Control Logic</u>	<u>Peak Throughput</u>
Single-Cycle CPU	Combinational Logic	1
Multiple-Cycle CPU	FSM or Microprogram	1
Pipelined CPU		1

CSE 141, S2'06

Jeff Brown

Pipelining in Modern CPUs

- CPU Datapath
- Arithmetic Units
- System Buses
- Software (at multiple levels)
- etc...

CSE 141, S2'06

Jeff Brown

A Pipelined Datapath

IF: Instruction fetch

ID: Instruction decode and register fetch

EX: Execution and effective address calculation

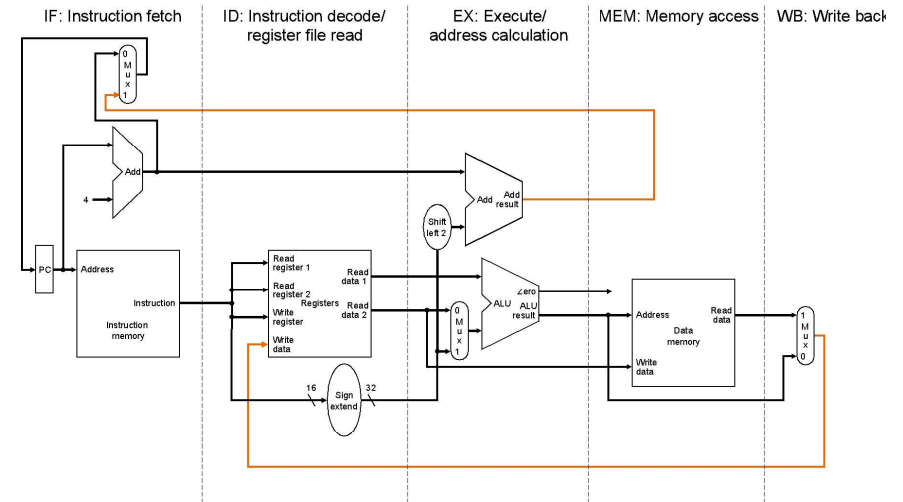
MEM: Memory access

WB: Write back

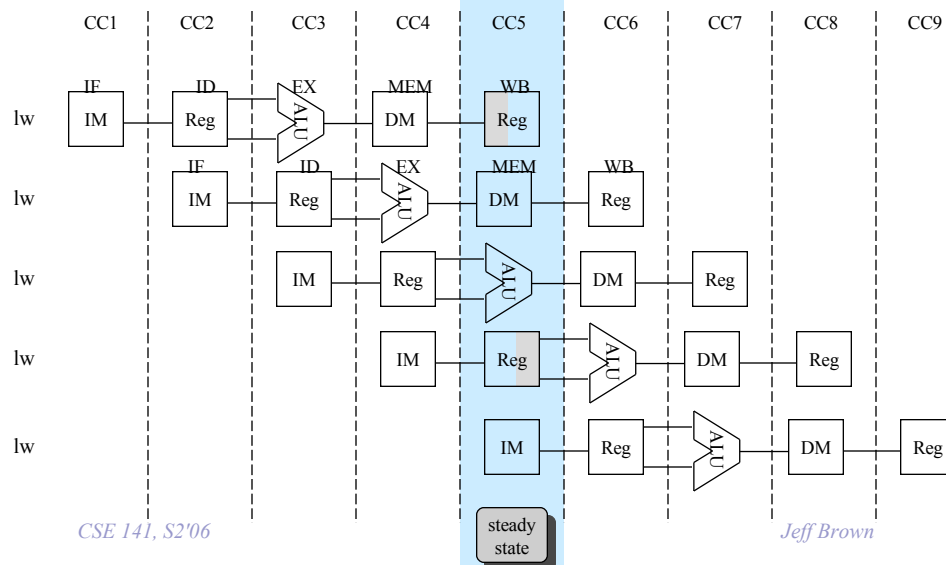
CSE 141, S2'06

Jeff Brown

Pipelined Datapath



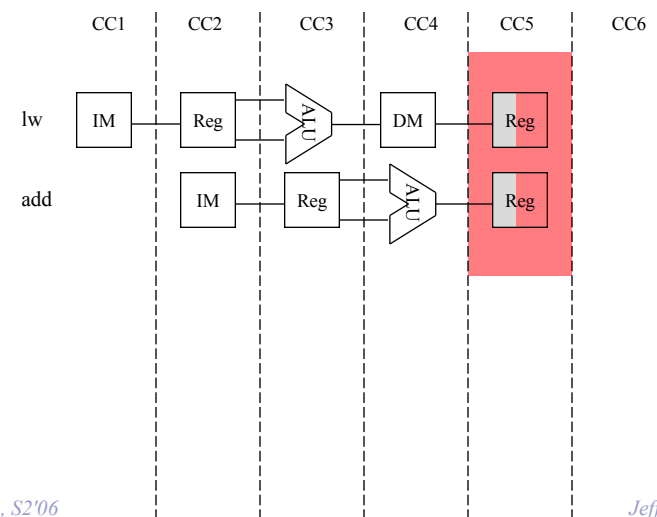
Execution in a Pipelined Datapath



CSE 141, S2'06

Jeff Brown

Mixed Instructions in the Pipeline

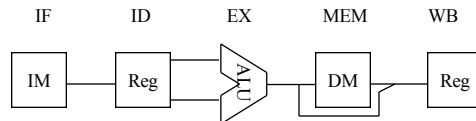


CSE 141, S2'06

Jeff Brown

Pipeline Principles

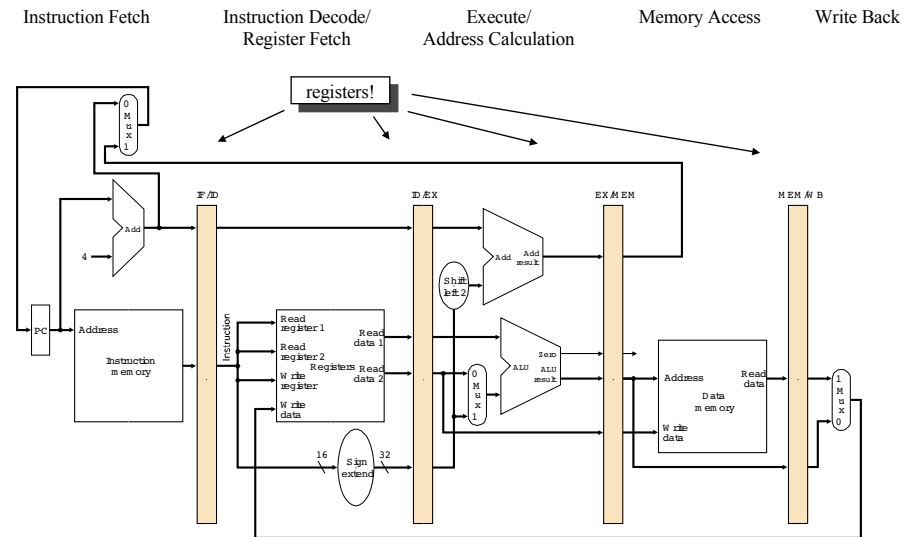
- All instructions that share a pipeline must have the same *stages* in the same *order*.
 - therefore, *add* does nothing during Mem stage
 - sw* does nothing during WB stage
- All intermediate values must be latched each cycle.
- There is no functional block reuse



CSE 141, S2'06

Jeff Brown

Pipelined Datapath

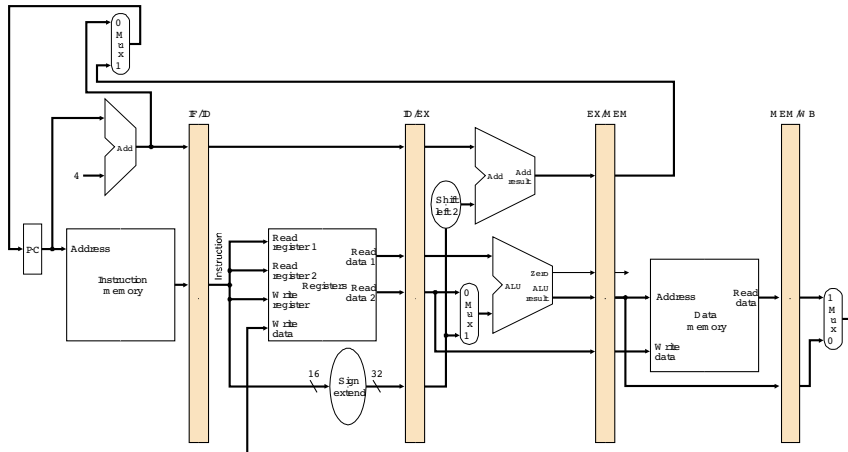


CSE 141, S2'06

Jeff Brown

The Pipeline in Execution

add \$10, \$1, \$2 Instruction Decode/
Register Fetch Execute/
Address Calculation Memory Access Write Back

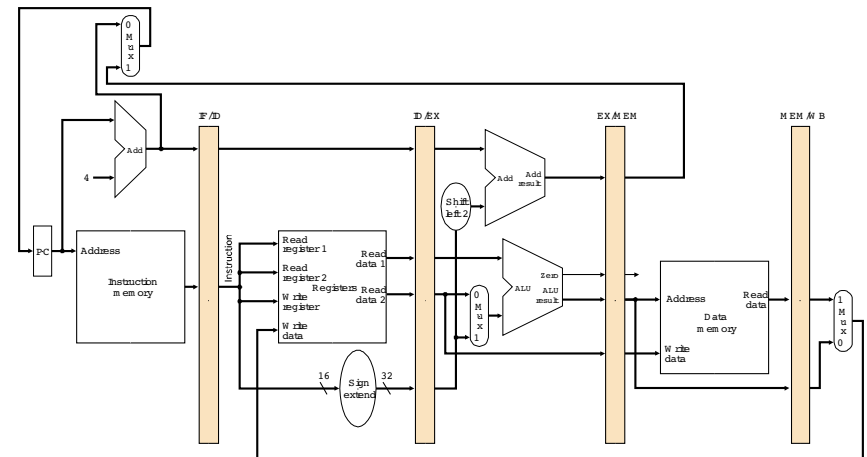


CSE 141, S2'06

Jeff Brown

The Pipeline in Execution

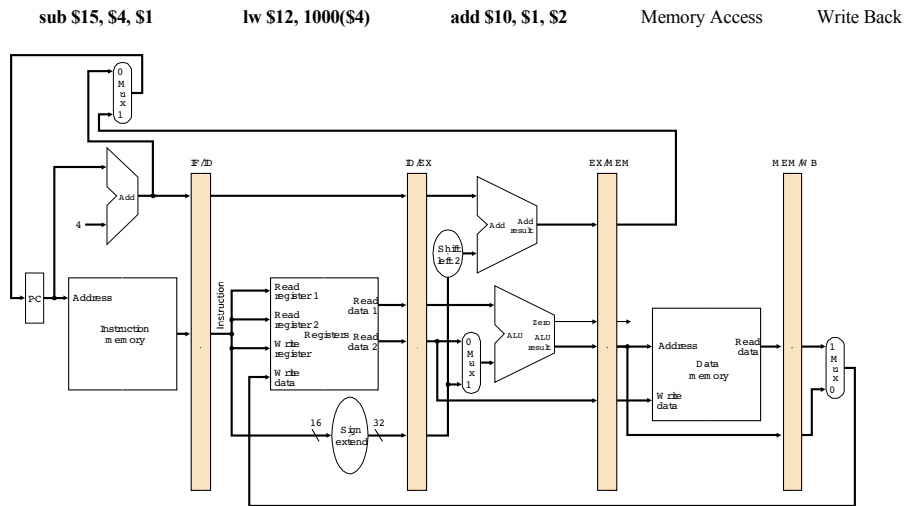
lw \$12, 1000(\$4) **add \$10, \$1, \$2** Execute/
Address Calculation Memory Access Write Back



CSE 141, S2'06

Jeff Brown

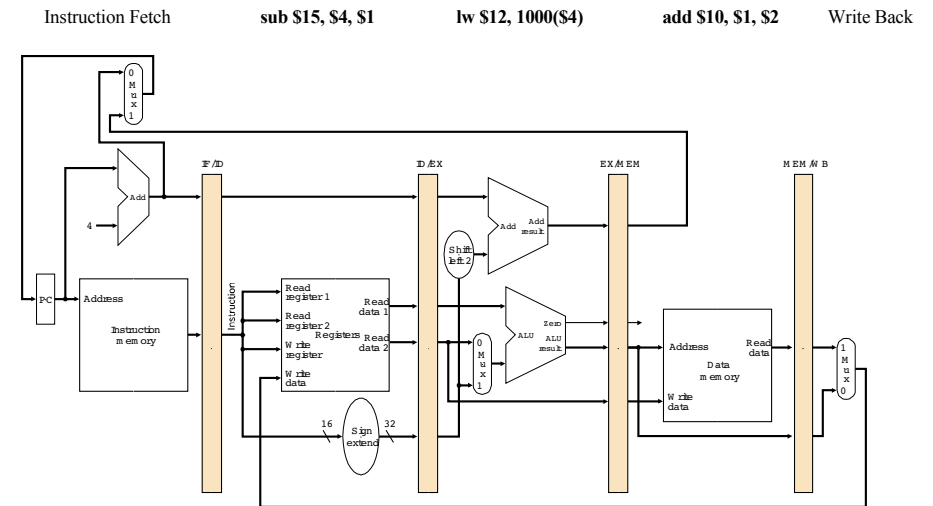
The Pipeline in Execution



CSE 141, S2'06

Jeff Brown

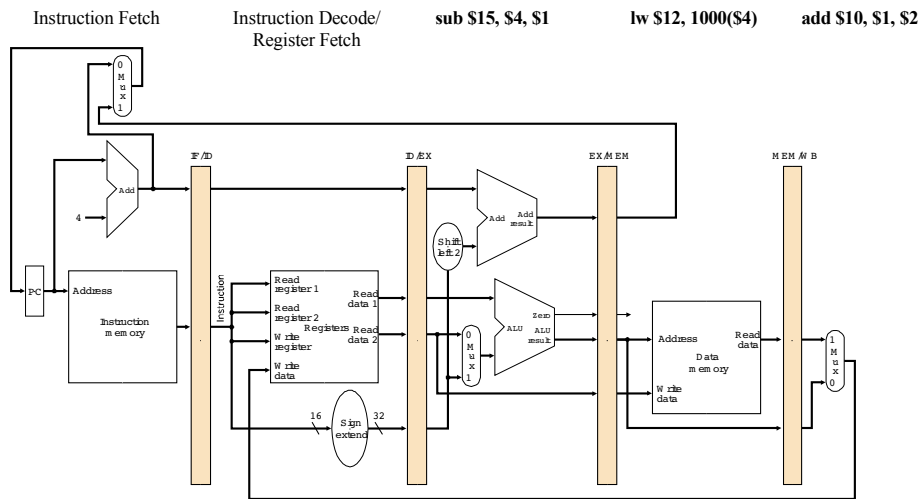
The Pipeline in Execution



CSE 141, S2'06

Jeff Brown

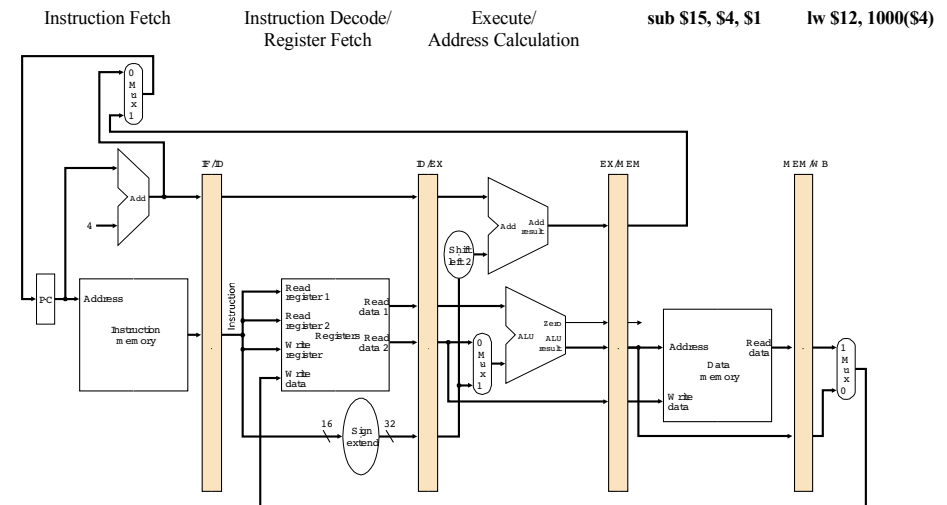
The Pipeline in Execution



CSE 141, S2'06

Jeff Brown

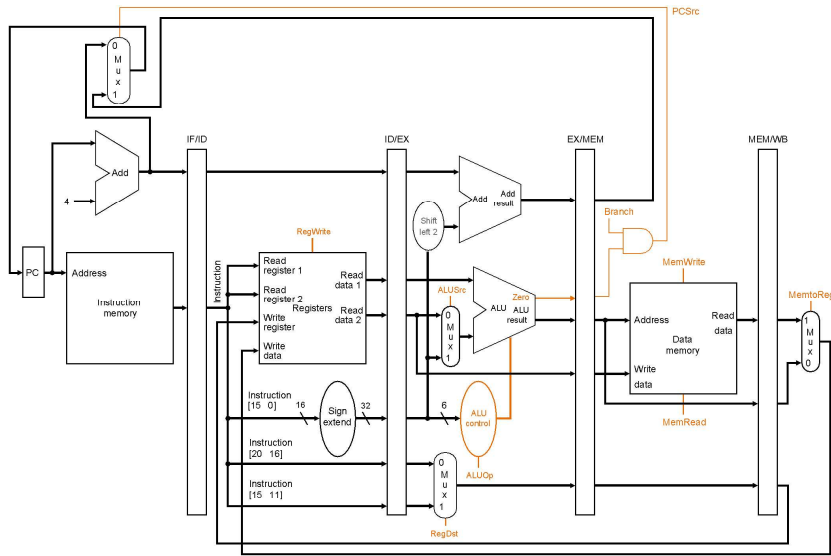
The Pipeline in Execution



CSE 141, S2'06

Jeff Brown

The Pipeline, with controls But....



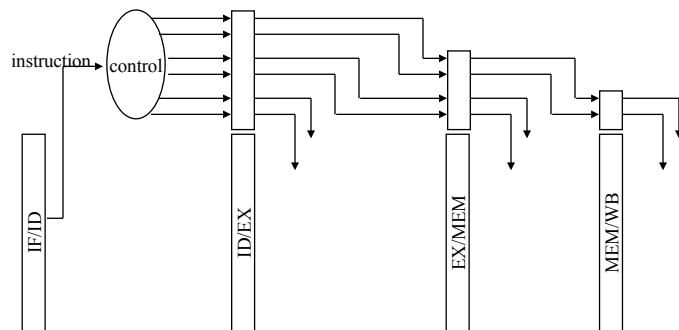
Pipelined Control

- can't use **microprogram**.
- **FSM** not really appropriate.
- **Combinational logic!**
 - signals generated **once**, but follow instruction through the pipeline

CSE 141, S2'06

Jeff Brown

Pipelined Control



CSE 141, S2'06

Jeff Brown

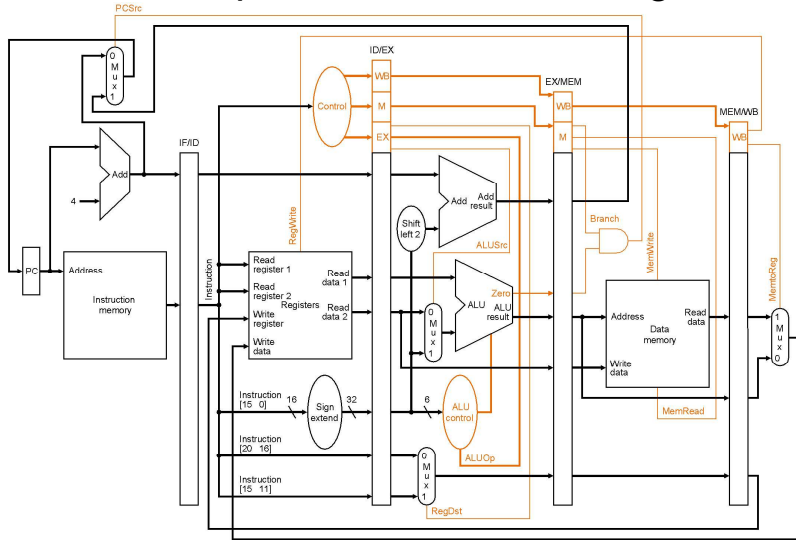
Pipelined Control Signals

Instruction	Execution Stage Control Lines				Memory Stage Control Lines			Write Back Stage Control Lines	
	RegDst	ALUSrc	ALUSrc	ALUSrc	Branch	MemRead	MemWrite	RegWrite	MemtoReg
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

CSE 141, S2'06

Jeff Brown

The Pipeline with Control Logic



CSE 141, S2'06

Jeff Brown

Is it really that easy?

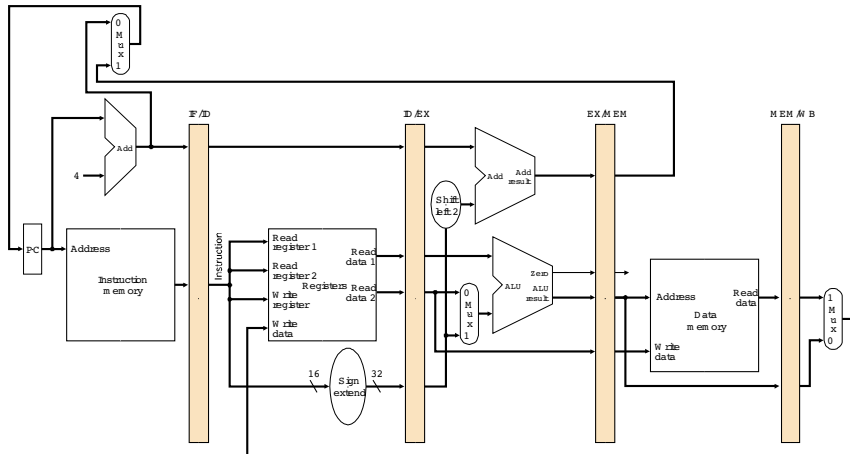
- What happens when...
 add \$3, \$10, \$11
 lw \$8, 1000(\$3)
 sub \$11, \$8, \$7

CSE 141, S2'06

Jeff Brown

The Pipeline in Execution

lw \$8, 1000(\$3) add \$3, \$10, \$11 Execute/
Address Calculation Memory Access Write Back

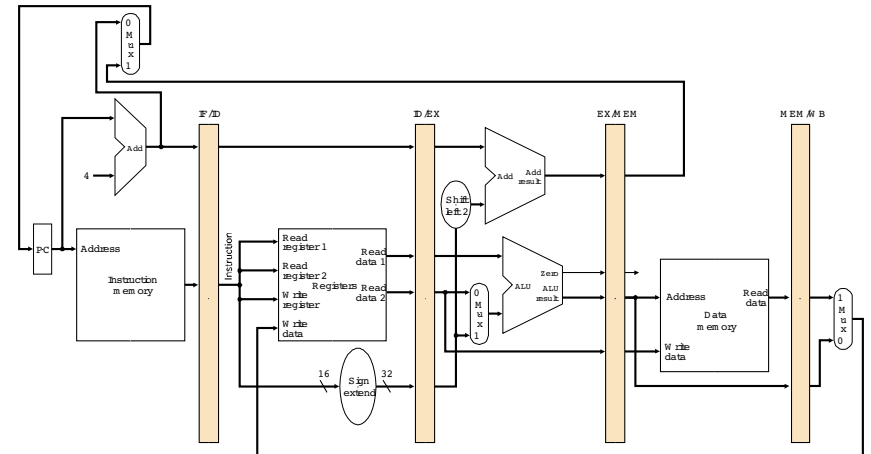


CSE 141, S2'06

Jeff Brown

The Pipeline in Execution

sub \$11, \$8, \$7 lw \$8, 1000(\$3) add \$3, \$10, \$11 Memory Access Write Back

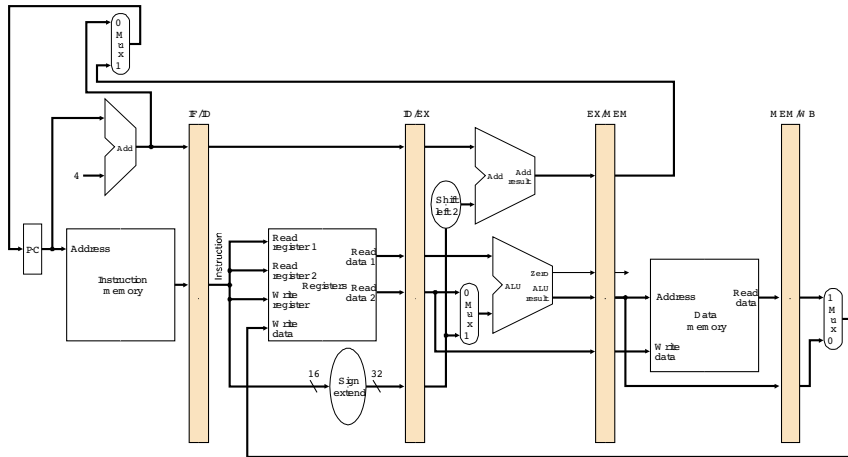


CSE 141, S2'06

Jeff Brown

The Pipeline in Execution

add \$10, \$1, \$2 sub \$11, \$8, \$7 lw \$8, 1000(\$3) add \$3, \$10, \$11 Write Back

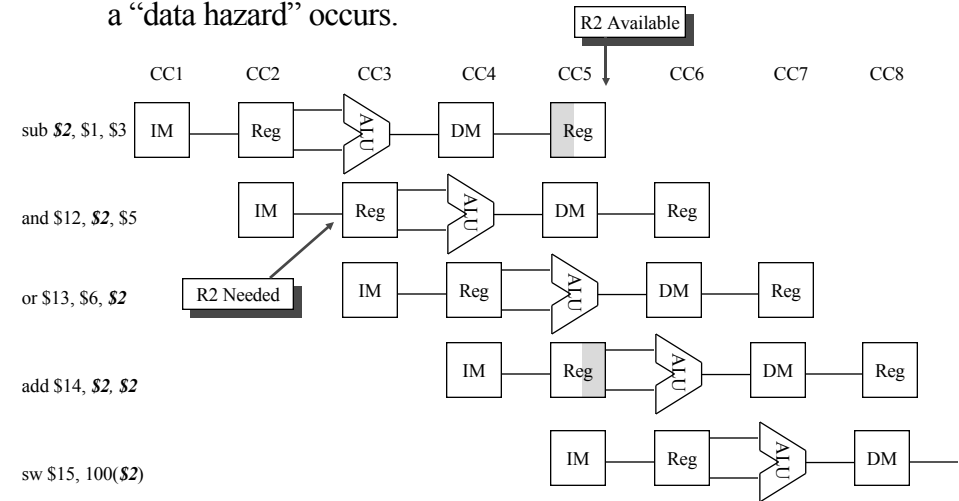


CSE 141, S2'06

Jeff Brown

Data Hazards

- When a result is needed in the pipeline before it is available, a “data hazard” occurs.



CSE 141, S2'06

Jeff Brown

Pipelining Key Points

- $ET = IC * CPI * CT$
- We achieve high *throughput* without reducing instruction *latency*.
- Pipelining exploits a special kind of parallelism (parallelism between functionality required in different cycles).
- Pipelining uses combinational logic (and registers to propagate) to generate control signals.
- Pipelining creates potential hazards.

CSE 141, S2'06

Jeff Brown