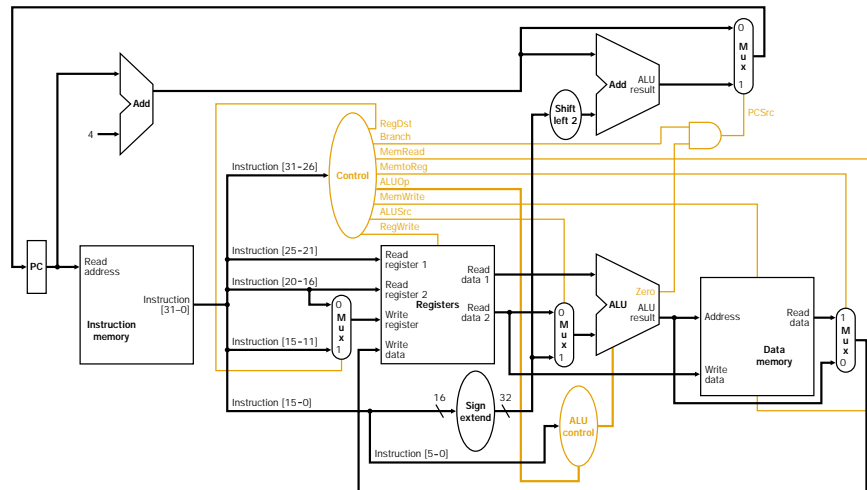


Designing a Pipelined CPU

CSE 141

Allan Snively

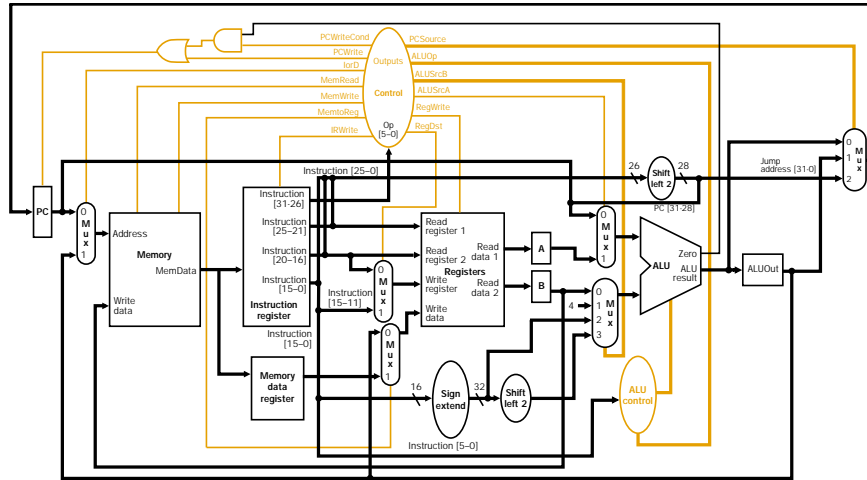
Review -- Single Cycle CPU



CSE 141

Allan Snively

Review -- Multiple Cycle CPU

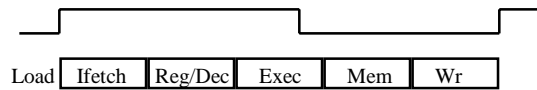


CSE 141

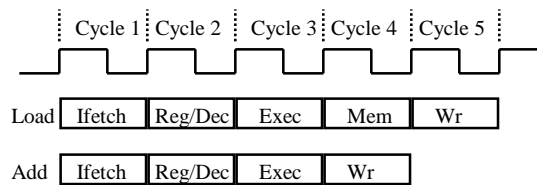
Allan Snaveley

Review -- Instruction Latencies

•Single-Cycle CPU



•Multiple Cycle CPU

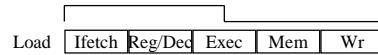


CSE 141

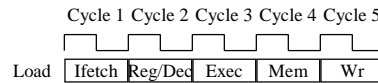
Allan Snaveley

Instruction Latencies and Throughput

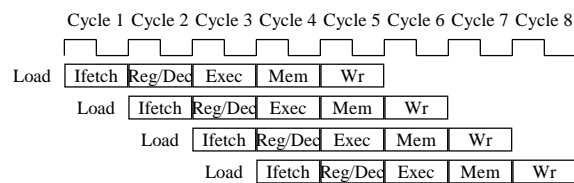
•Single-Cycle CPU



•Multiple Cycle CPU



•Pipelined CPU



CSE 141

Allan Snively

Pipelining Advantages

- Higher maximum throughput
- Higher utilization of CPU resources

- But, more complicated datapath, more complex control (?)

CSE 141

Allan Snively

Pipelining Advantages

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

Pipelining in Modern CPUs

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

A Pipelined Datapath

IF: Instruction fetch

ID: Instruction decode and register fetch

EX: Execution and effective address calculation

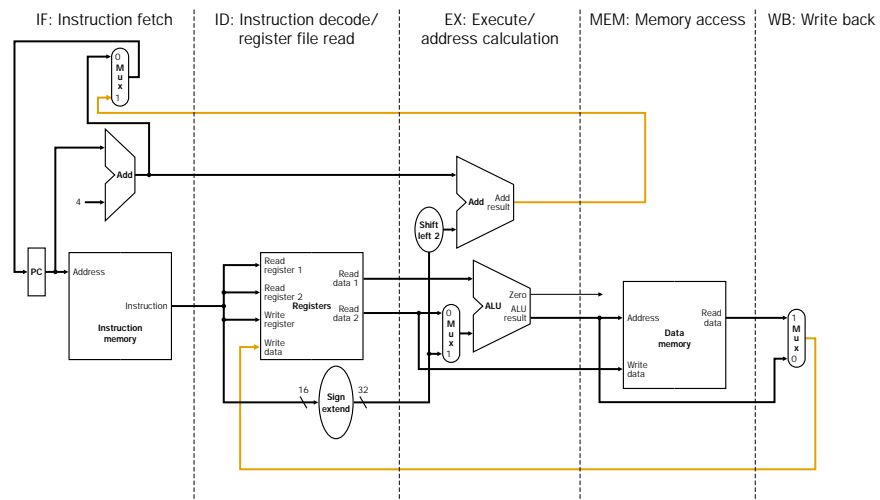
MEM: Memory access

WB: Write back

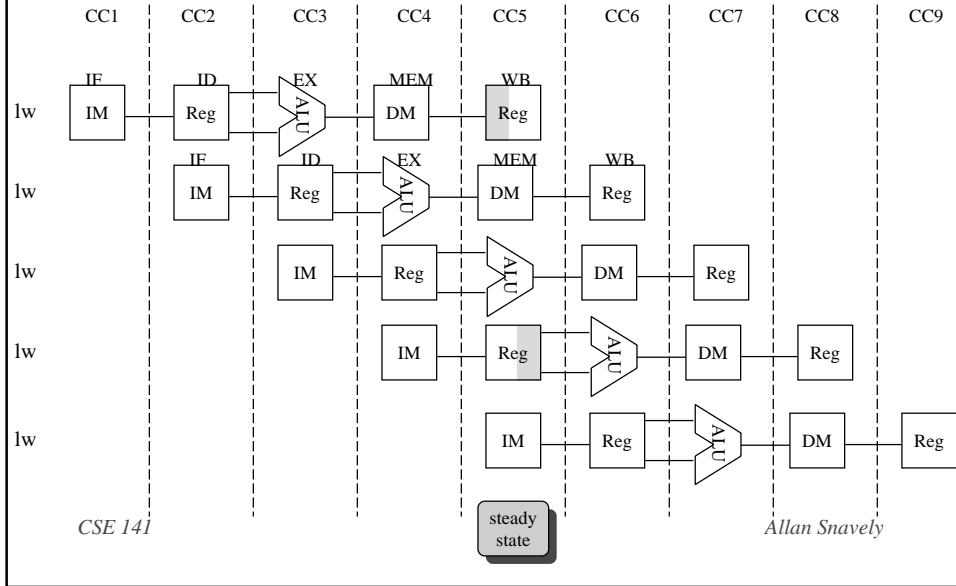
CSE 141

Allan Snively

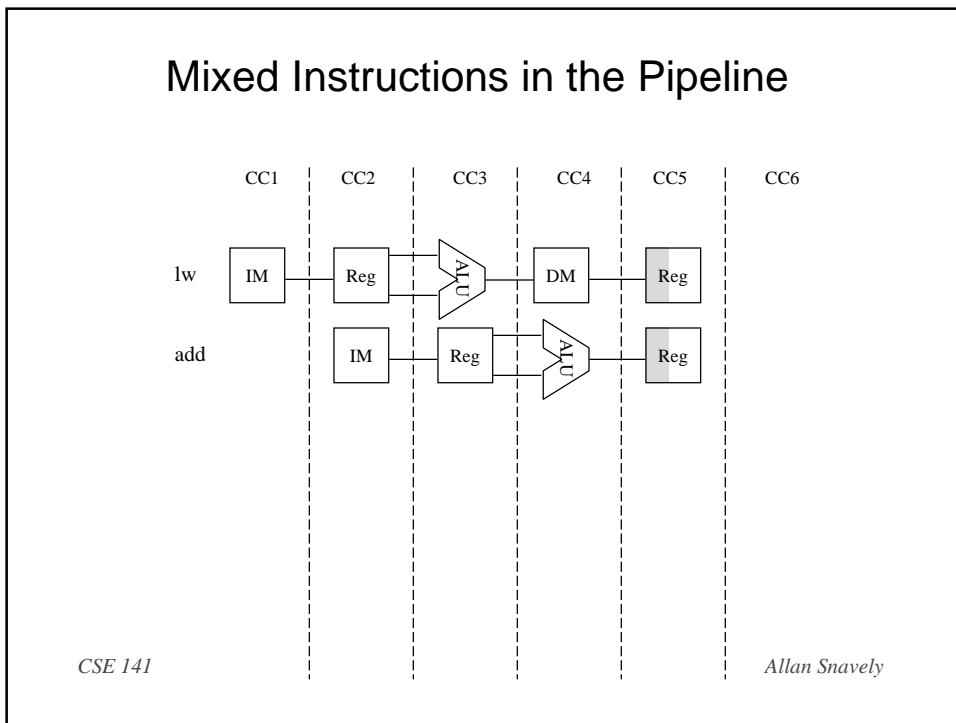
Pipelined Datapath



Execution in a Pipelined Datapath

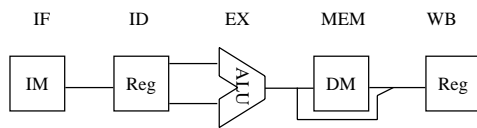


Mixed Instructions in the Pipeline



Pipeline Principles

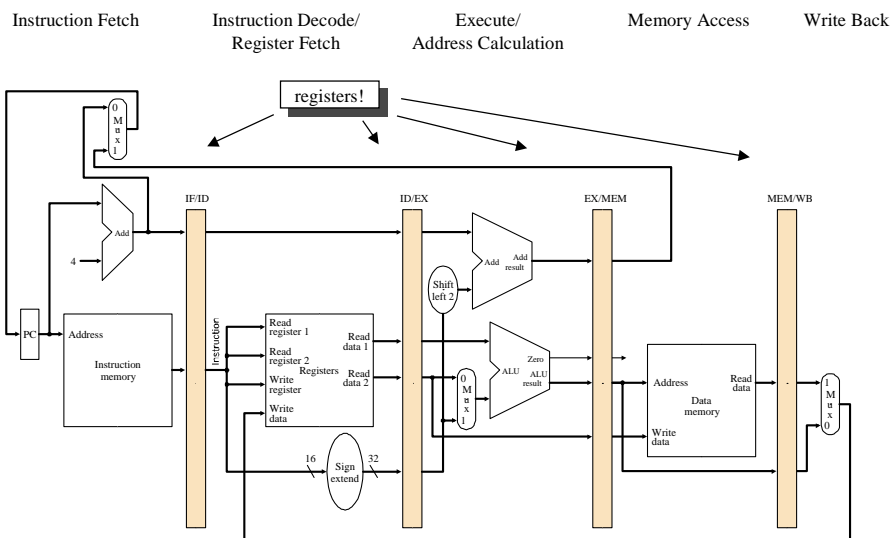
- All instructions that share a pipeline must have the same stages in the same order.
 - therefore, *add* does nothing during Mem stage
 - *sw* does nothing during WB stage
- All intermediate values must be latched each cycle.
- There is no functional block reuse



CSE 141

Allan Snaveley

Pipelined Datapath

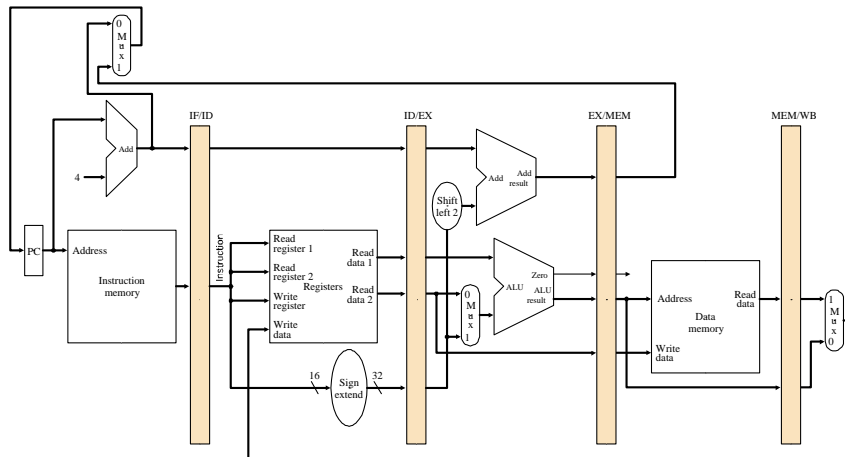


CSE 141

Allan Snaveley

The Pipeline in Execution

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

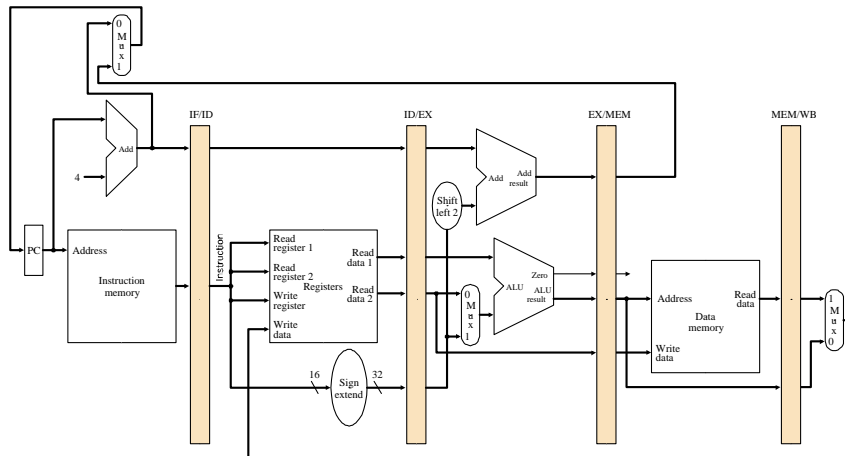


CSE 141

Allan Snively

The Pipeline in Execution

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

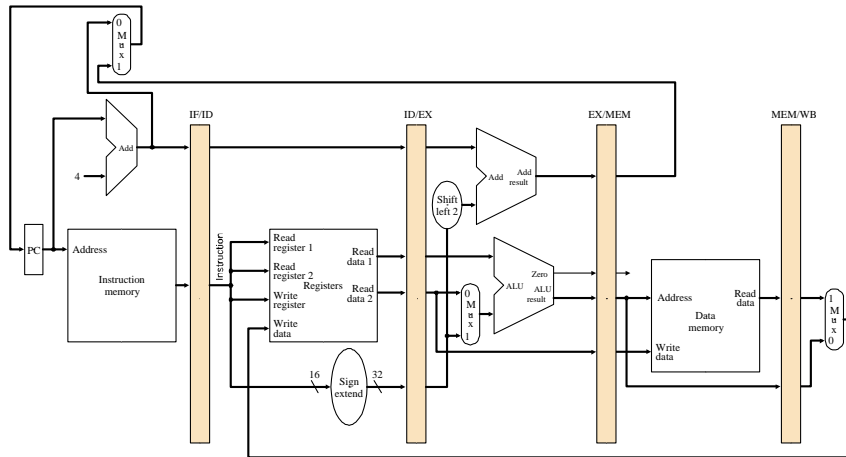


CSE 141

Allan Snively

The Pipeline in Execution

sub \$15, \$4, \$1 lw \$12, 1000(\$4) add \$10, \$1, \$2 Memory Access Write Back

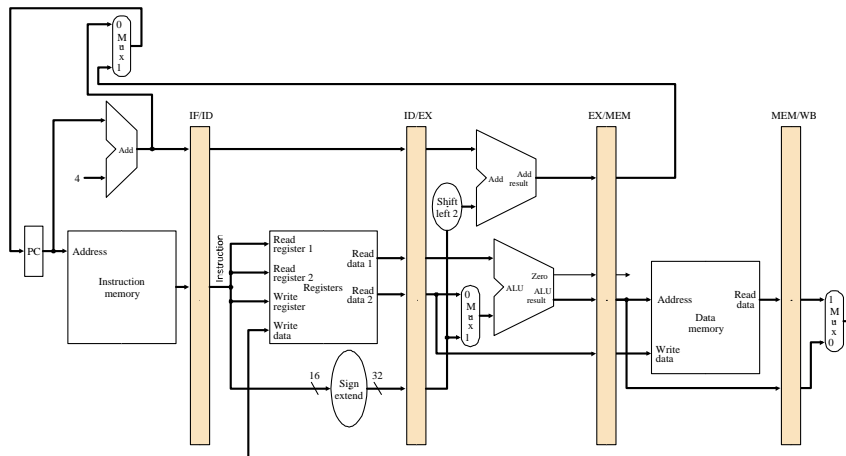


CSE 141

Allan Snively

The Pipeline in Execution

Instruction Fetch sub \$15, \$4, \$1 lw \$12, 1000(\$4) add \$10, \$1, \$2 Write Back

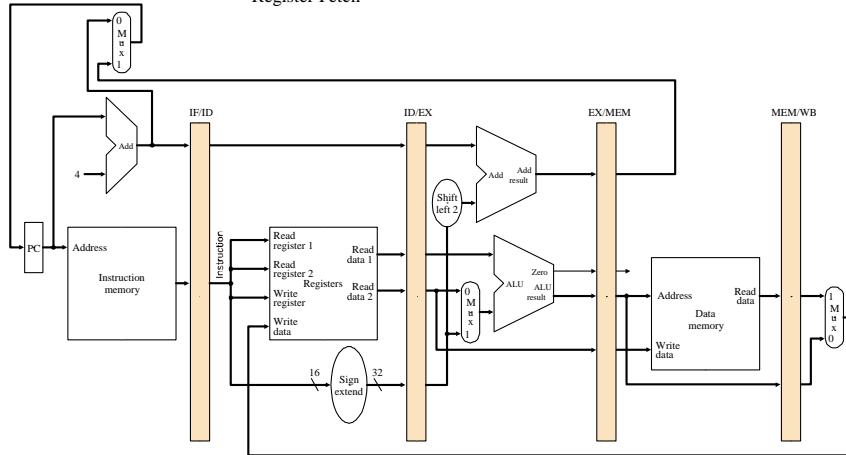


CSE 141

Allan Snively

The Pipeline in Execution

Instruction Fetch Instruction Decode/
Register Fetch **sub \$15, \$4, \$1** **lw \$12, 1000(\$4)** **add \$10, \$1, \$2**

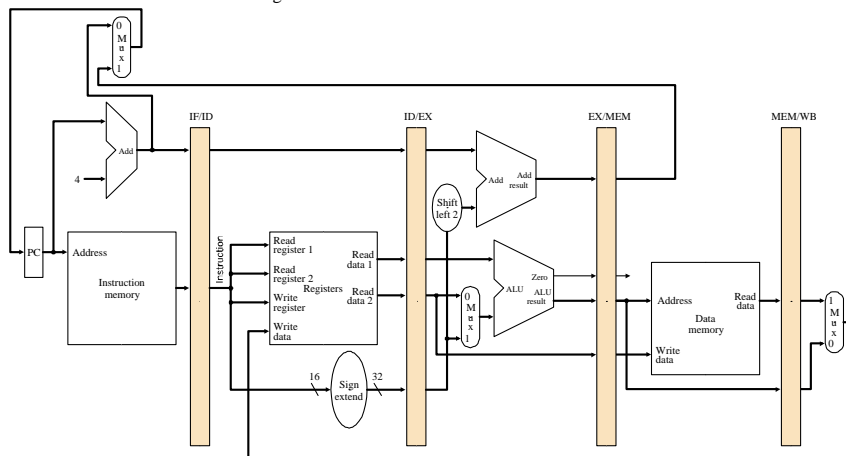


CSE 141

Allan Snively

The Pipeline in Execution

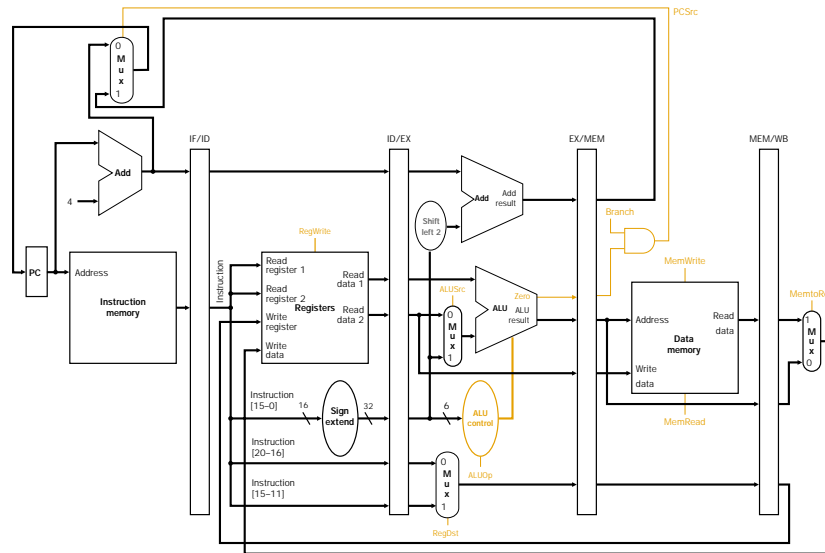
Instruction Fetch Instruction Decode/
Register Fetch Execute/
Address Calculation **sub \$15, \$4, \$1** **lw \$12, 1000(\$4)**



CSE 141

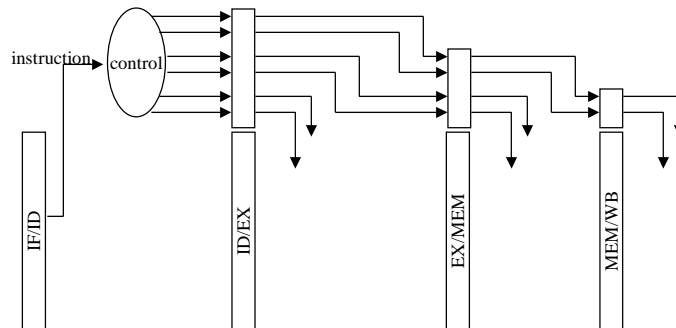
Allan Snively

The Pipeline in Execution, with controls



Pipelined Control

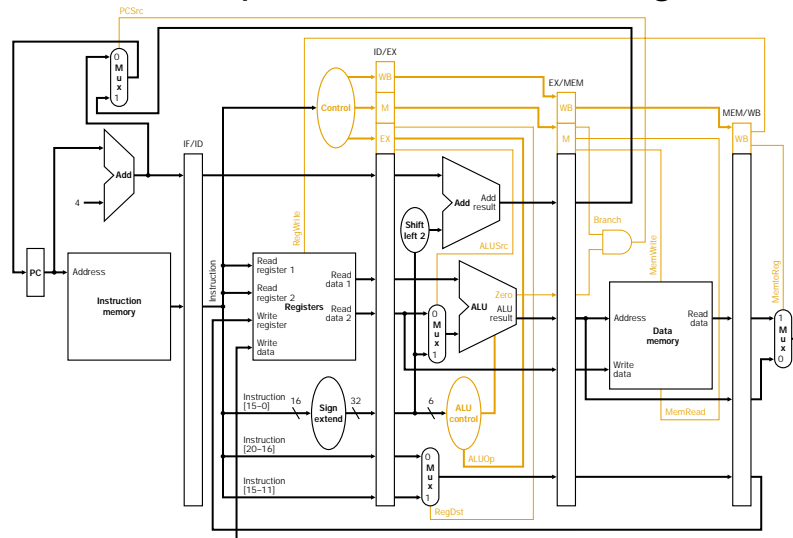
- can't use microprogram
- FSM not really appropriate
- **Combinational Logic!**
 - signals generated once, but follow instruction through the pipeline



Pipelined Control Signals

Instruction	Execution Stage Control Lines				Memory Stage Control Lines			Write Back Stage Control Lines	
	RegDst	ALUOp1	ALUOp0	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

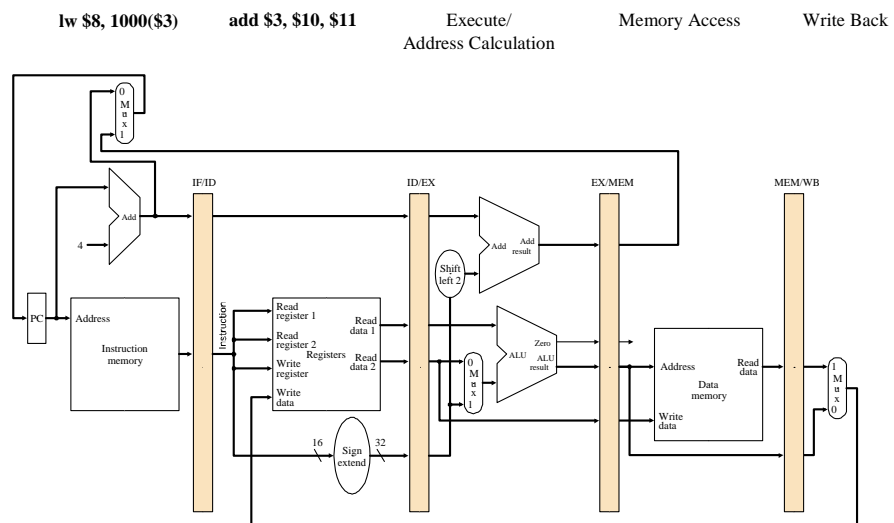
The Pipeline with Control Logic



Is it really that easy?

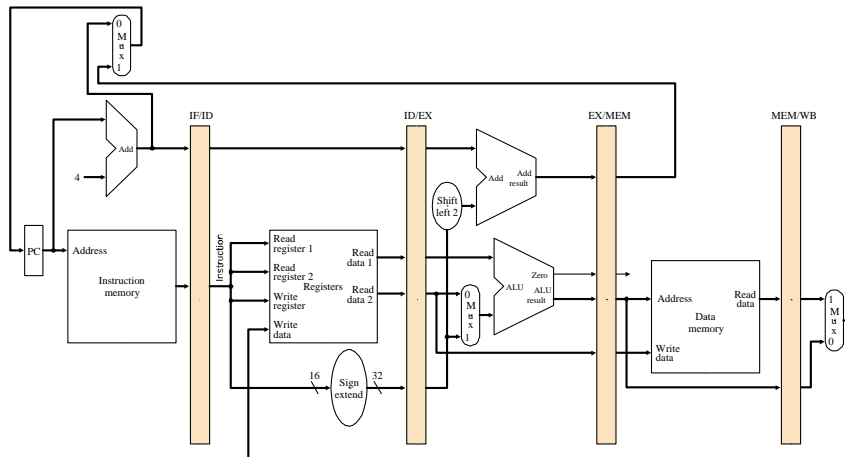
- What happens when...
add \$3, \$10, \$11
lw \$8, 1000(\$3)
sub \$11, \$8, \$7

The Pipeline in Execution



The Pipeline in Execution

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

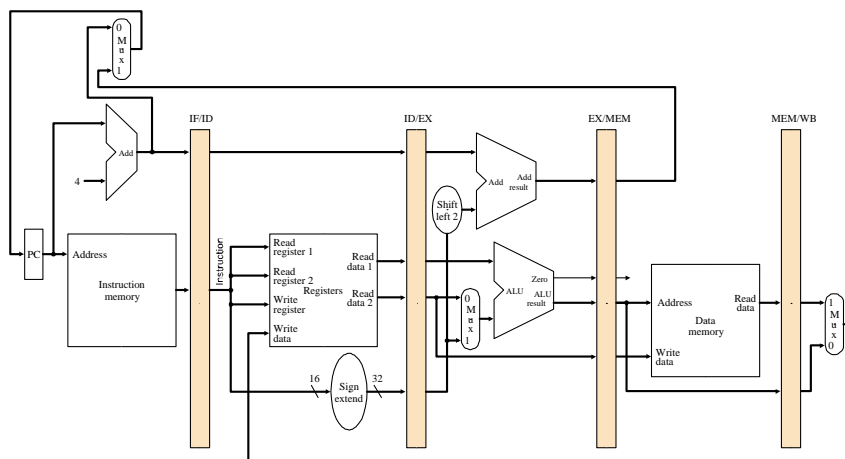


CSE 141

Allan Snively

The Pipeline in Execution

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



CSE 141

Allan Snively

Data Hazards

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

