CS / EE 320
Computer Organization and Assembly Language
Spring 2024
Lecture 17

Shahid Masud

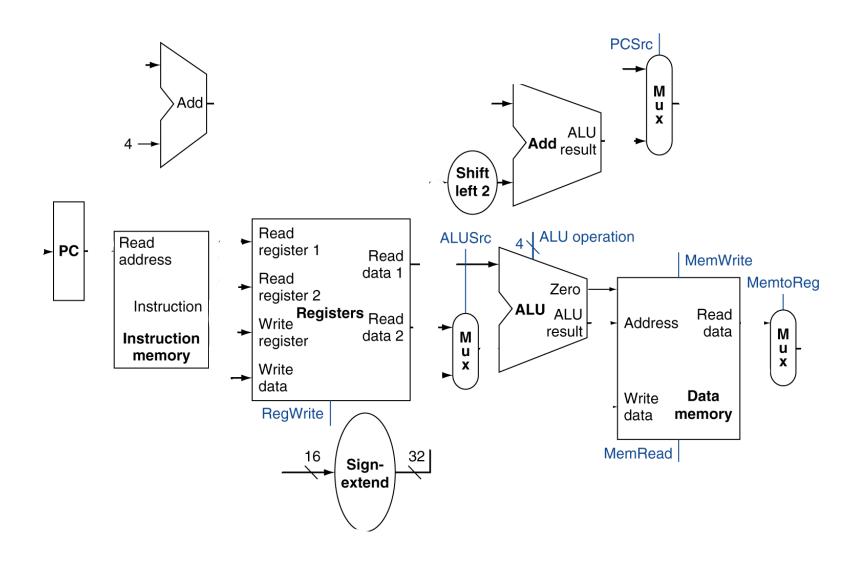**Topics: From Single Cycle MIPS to Multicyle MIPS, Pipelining**

Topics

- Timing calculations in Single-Cycle Simple MIPS CPU

- How to add pipelining in simple MIPS CPU

- Detailed analysis of 5-Stage Pipelined MIPS CPU

- Calculations Throughput, Latency, Critical Path of 5-Stage MIPS CPU

- Design of individual Pipelined stages in a 5-Stage MIPS CPU

- Hazards in Pipelined MIPS CIPS
    - Structural Hazard (Hardware Related)
    - Data Hazard (Due to Assembly program)
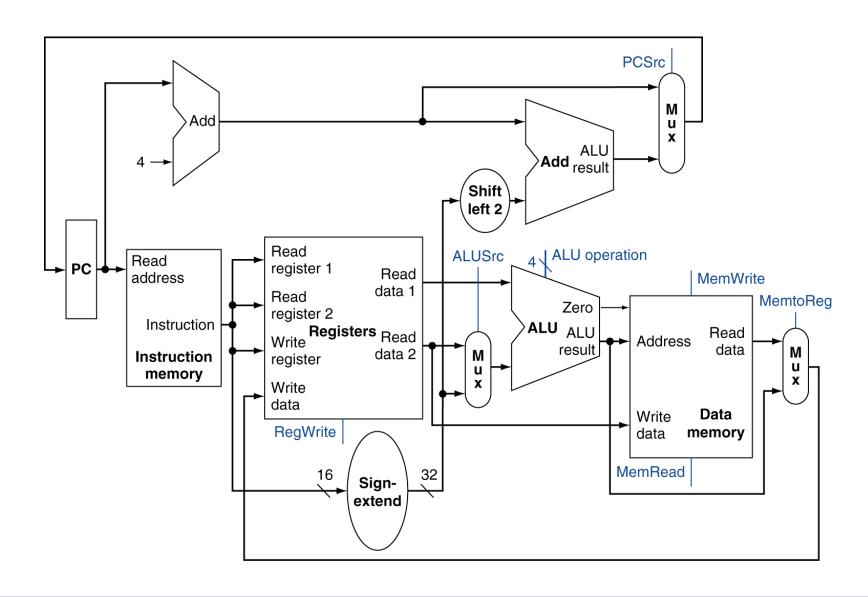   - Control Hazard (Architecture related)
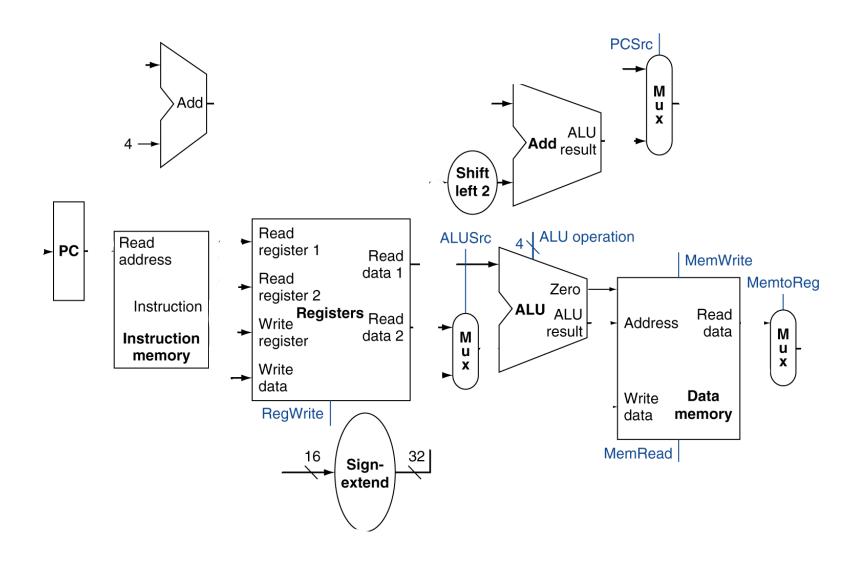
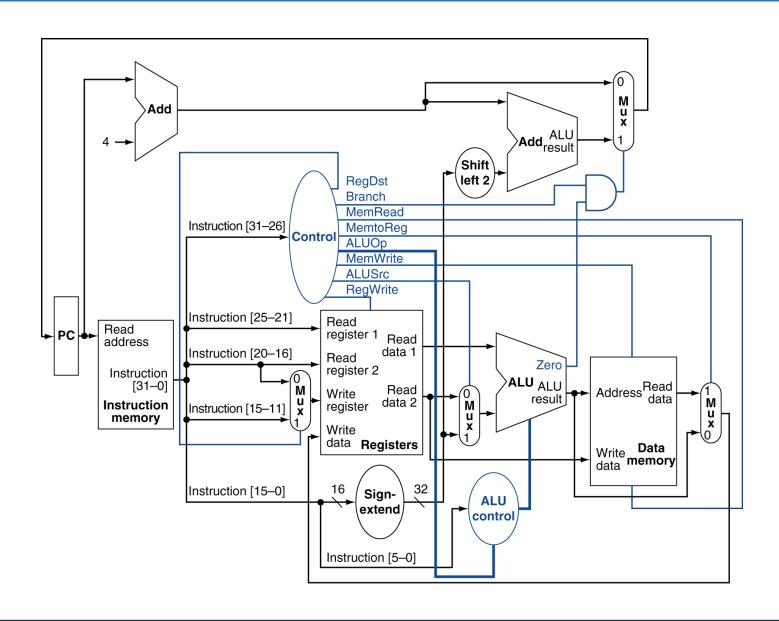# SIMPLE MIPS WITH NO PIPELINE

Full Datapath – for practice of connections

Full Datapath

Full Datapath – for practice of connections
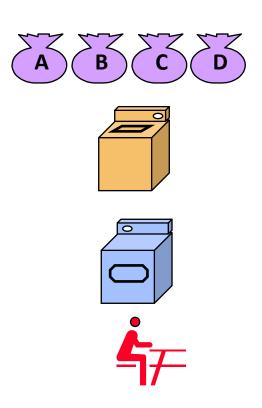
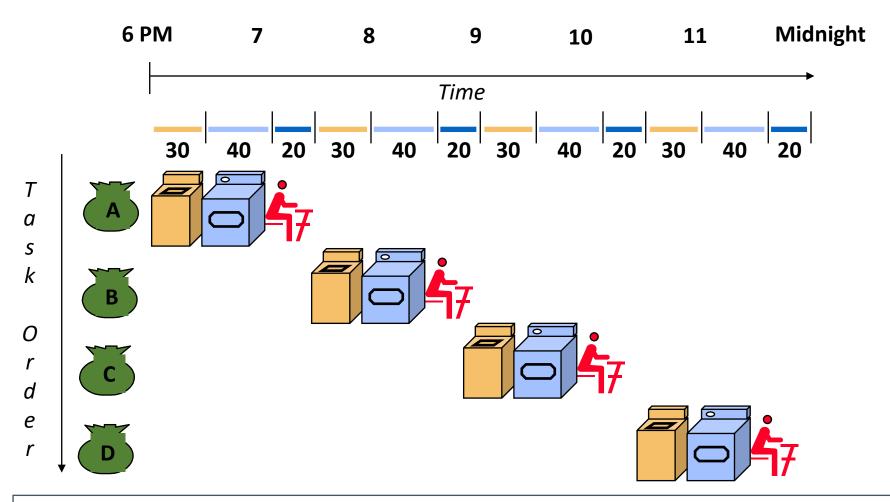# MIPS Datapath With Control

BASICS OF PIPELINING

What Is Pipelining

- Laundry Example

- Ann, Brian, Cathy, Dave
  each have one load of clothes
  to wash, dry, and fold

- Washer takes 30 minutes

- Dryer takes 40 minutes

- "Folding" takes 20 minutes
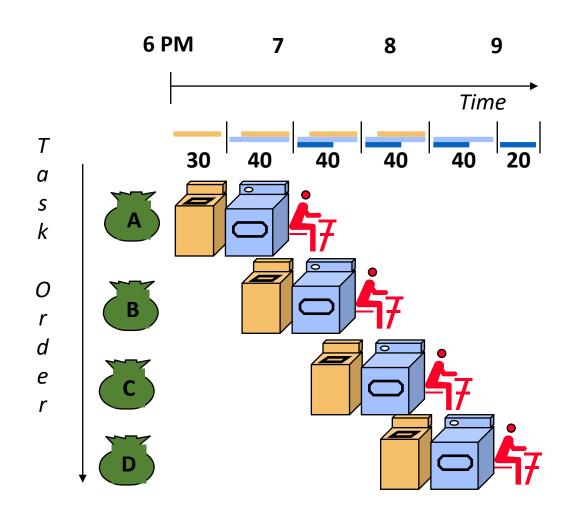
- One load takes 90 minutes

What Is Pipelining

Sequential laundry takes 6 hours for 4 loads

If they learned pipelining, how long would laundry take?

- Pipelining doesn't help latency of single task, it helps throughput of entire workload

- Pipeline rate limited by slowest pipeline stage

- Multiple tasks operating simultaneously

- Potential speedup = Number pipe stages

- Unbalanced lengths of pipe stages reduces speedup

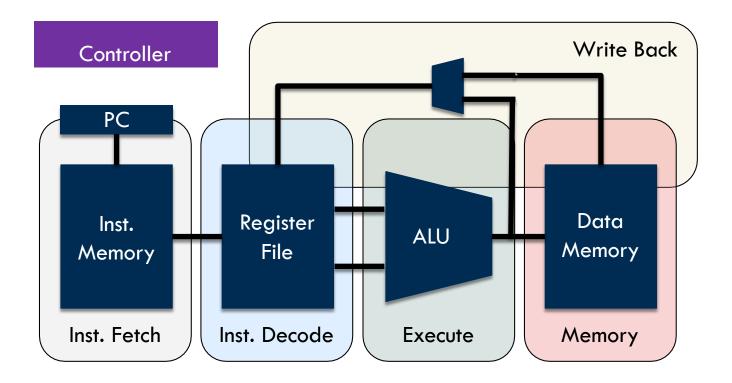- Time to "fill" pipeline and time to "drain" it reduces speedup

Introducing Pipelining for MIPS

Single-cycle MIPS RISC Architecture

D **Example: simple MIPS architecture**

Critical path includes all of the processing steps
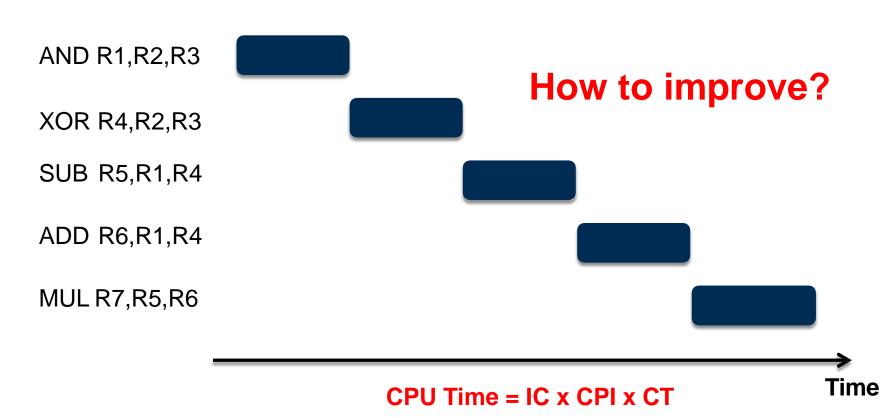
Execution on Single-cycle MIPS Architecture

D **Example program**

CT=6ns; CPU Time = 5 x 1 x 6ns = 30ns

AND R1,R2,R3

How to improve?

XOR R4,R2,R3

SUB R5,R1,R4

ADD R6,R1,R4

MUL R7,R5,R6

Time
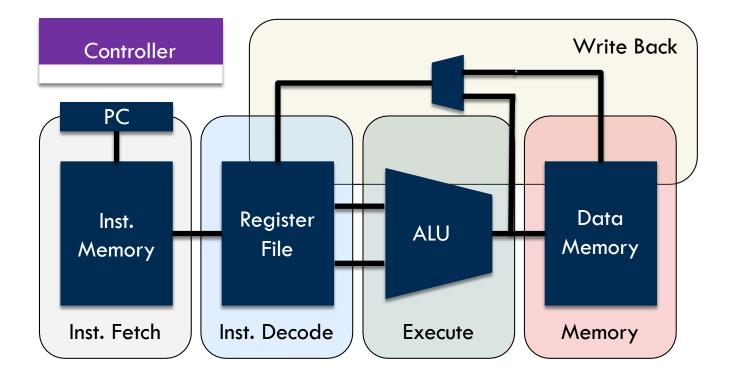
CPU Time = IC x CPI x CT

RISC Instruction Set

▣ Every instruction can be implemented in at most 5 clock cycles/ stages

  ◆ Instruction fetch cycle (IF): send PC to memory, fetch the current instruction from memory, and update PC to the next sequential PC by adding 4 to the PC.

  ◆ Instruction decode/register fetch cycle (ID): decode the instruction, read the registers corresponding to register source specifiers from the register file.

  ◆ Execution/effective address cycle (EX): perform Memory address calculation for Load/Store, Register-Register ALU instruction and Register-Immediate ALU instruction.

  ◆ Memory access (MEM): Perform memory access for load/store instructions.

  ◆ Write-back cycle (WB): Write back results to the dest operands for Register-Register ALU instruction or Load instruction.

Re-Using Idle Resources

D **Each processing step finishes in a fraction of a cycle**
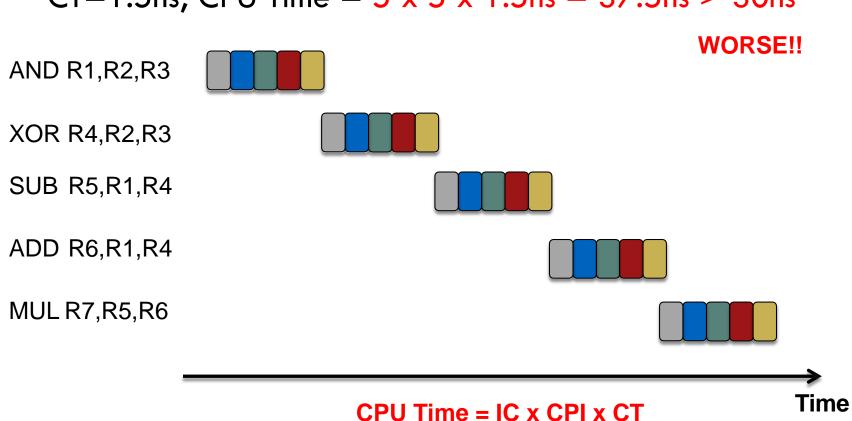
Idle resources can be reused for processing next
instruction

D **Example program**

CT=1.5ns; CPU Time = $5 \times 5 \times 1.5ns = 37.5ns > 30ns$

WORSE!!

AND R1,R2,R3

XOR R4,R2,R3

SUB R5,R1,R4

ADD R6,R1,R4

MUL R7,R5,R6
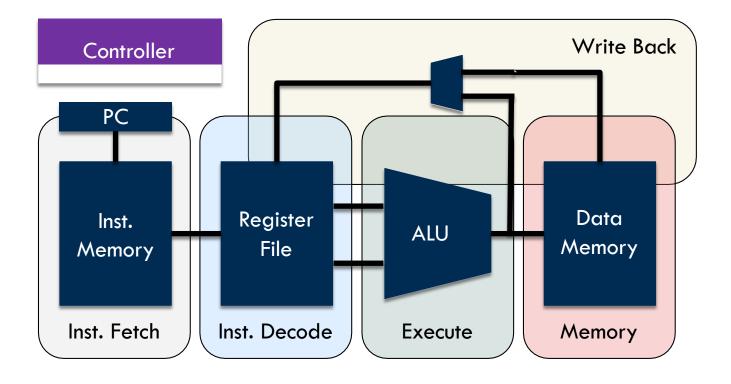
Time

CPU Time = IC x CPI x CT

Developing a Multi-cycle Data Path

5

Develop Simple Five Stage MIPS Pipeline

D A pipelined load-store architecture that processes up to one instruction per cycle
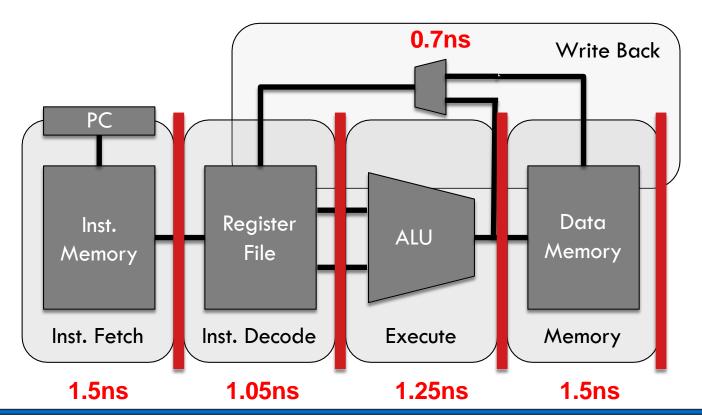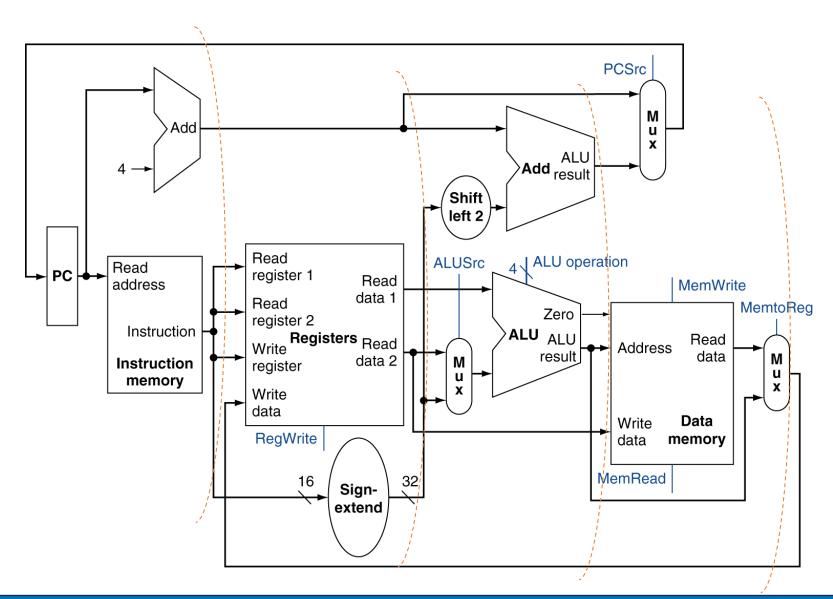
Five stage pipeline

Critical path determines the cycle time

Identify CPU Operations where Pipeline Registers could be Inserted
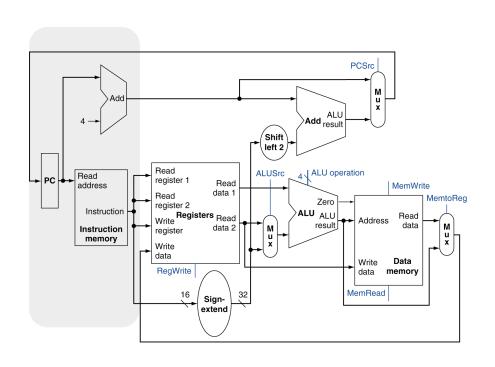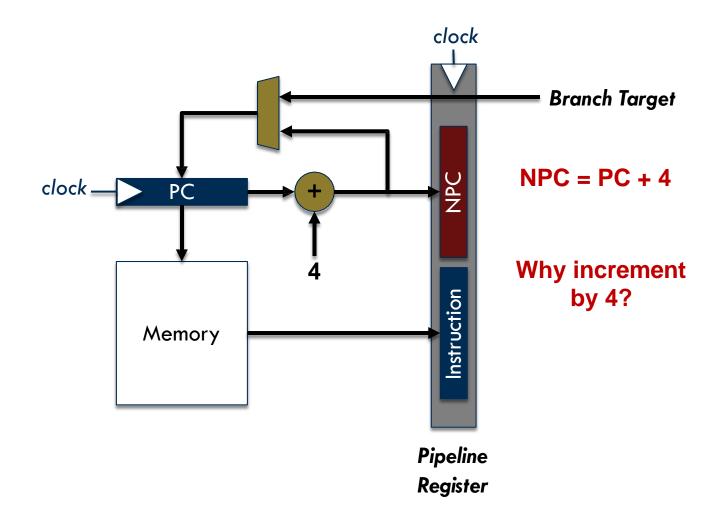
Instruction Fetch

- Read an instruction from memory (I-Memory)

  Use the program counter (PC) to index into the I- Memory

  Compute New PC (NPC) by incrementing current PC

  - What about branches?

- Update pipeline registers

  Write the instruction into the pipeline registers

Instruction Fetch

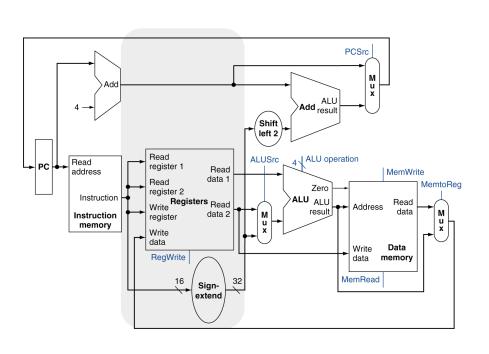**NPC = PC + 4**

Why increment by 4?
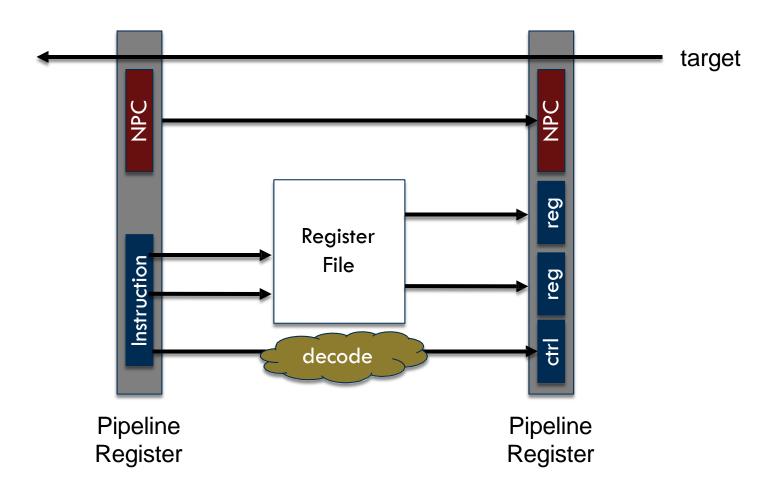
Instruction Decode

- Generate control signals for the opcode bits
- Read source operands from the register file (RF)

  Use the specifiers for indexing RF

  - How many read ports are required?

- Update pipeline registers

  Send the operand and immediate values to next stage

  Pass control signals and NPC to next stage

Execute Stage

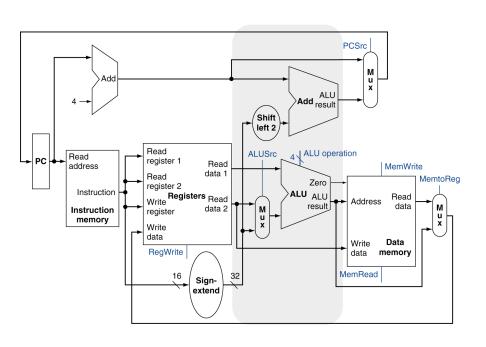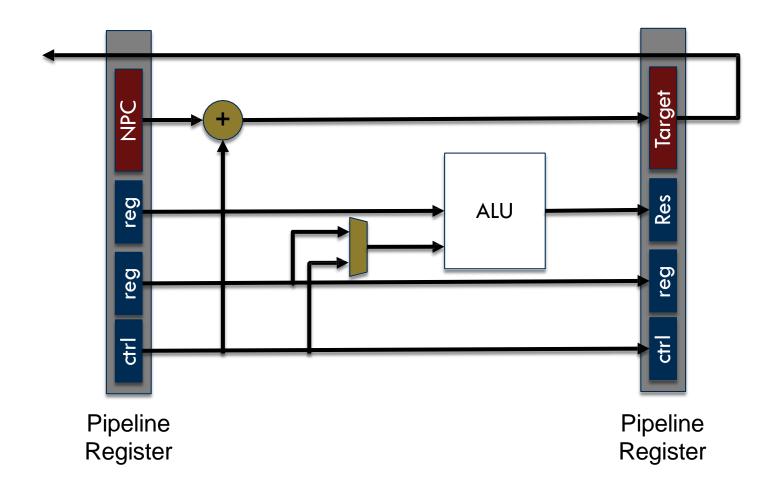- Perform ALU operation

  Compute the result of ALU

  - Operation type: control signals
  - First operand: contents of a register
  - Second operand: either a register or the immediate value

  Compute branch target

  - Target = NPC + immediate

- Update pipeline registers

  Control signals, branch target, ALU results, and destination

Execute Stage

Pipeline Register

Pipeline Register

Memory Access
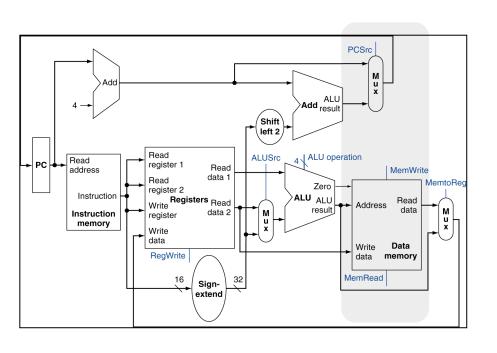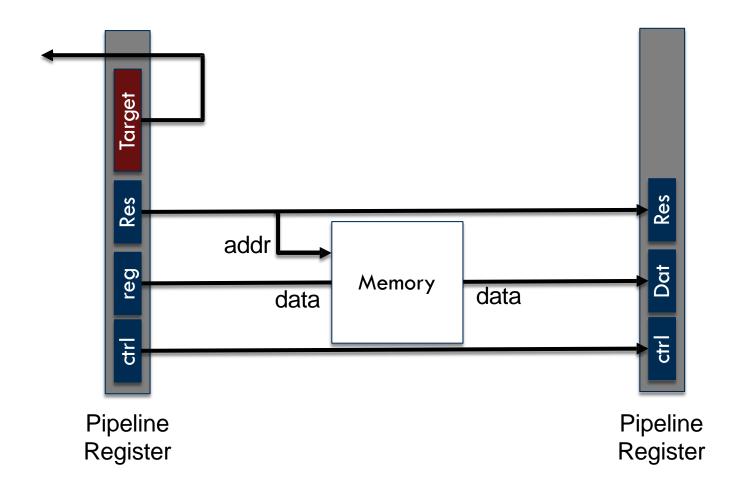
D **Access data memory**

   Load/store address: ALU outcome

   Control signals determine read or write access

D **Update pipeline registers**

   ALU results from execute

   Loaded data from D-Memory

   Destination register

Pipeline Register

Pipeline Register

Register Write Back

D **Update register file**

Control signals determine if a register write is needed

Only one write port is required

- Write the ALU result to the destination register, or
- Write the loaded data into the register file

Improving Resource Utilization
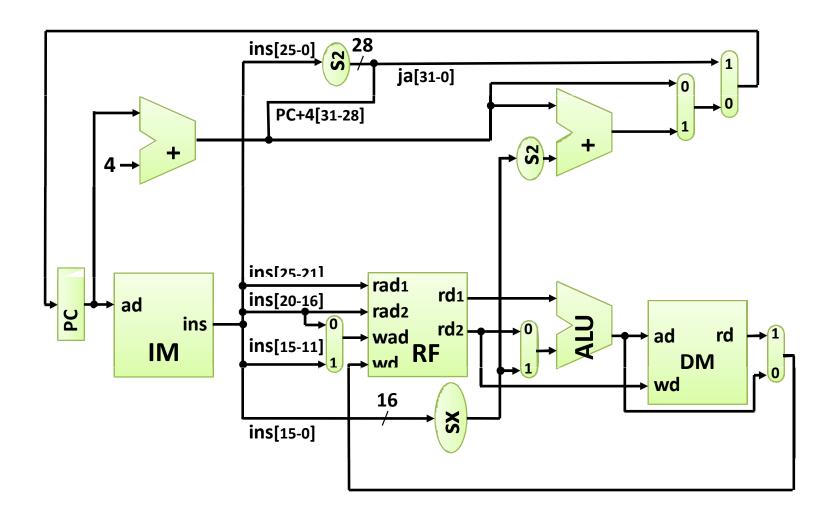
Efficient MIPS CPU Design

- Can we eliminate two adders?

- How to share (or reuse) a resource (say ALU) in different clock cycles?

- Store results in registers.

- Of course, more multiplexing may be required!

- Resources in this design: RF, ALU, MEM.

Single Cycle Datapath
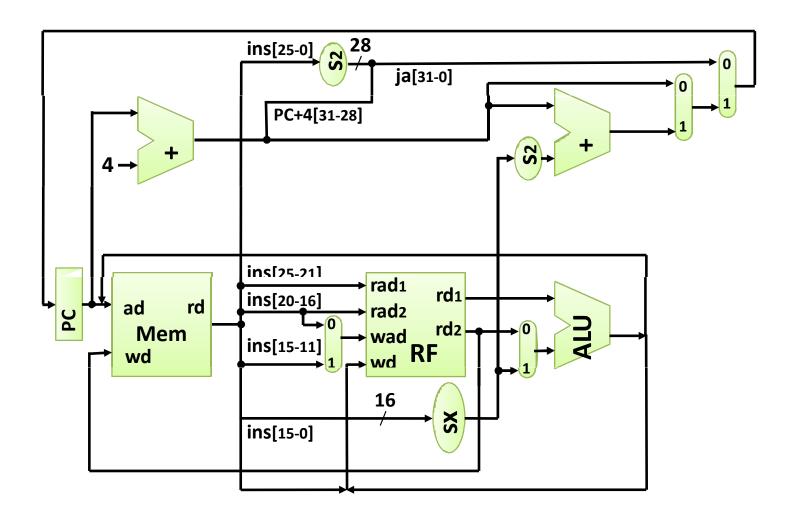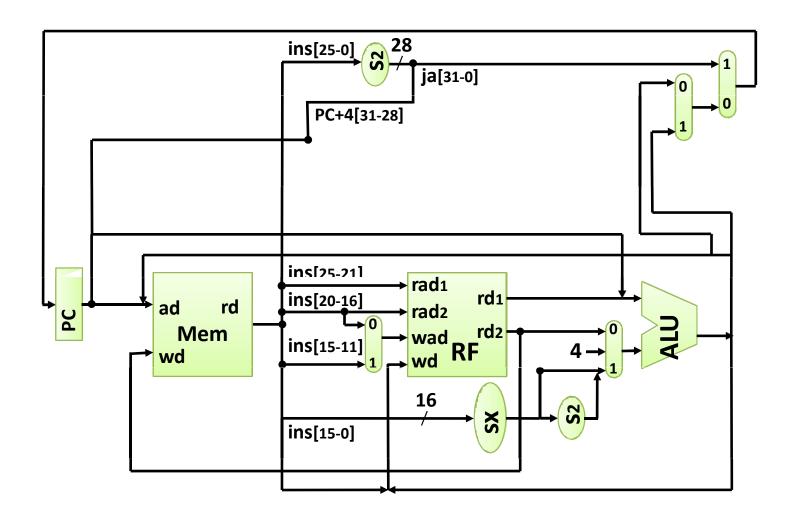
Combine Adders - Rearrange Diagram
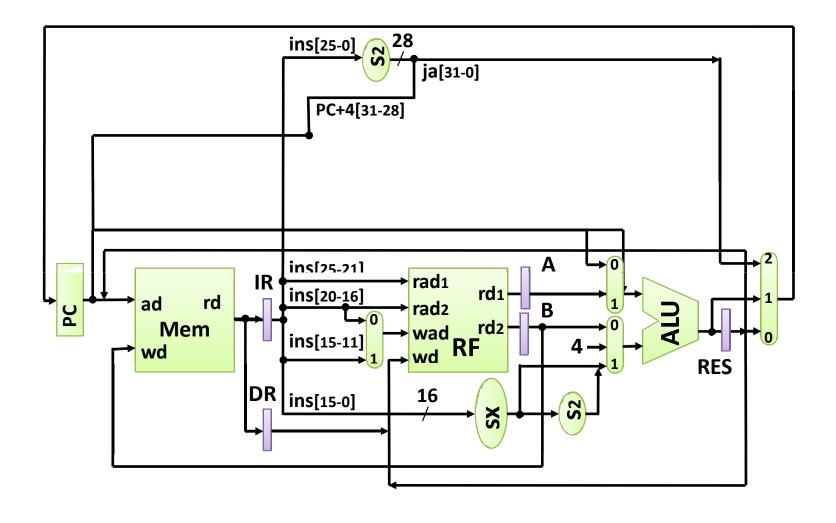
Introduce ALU multiplexers at Input and Output
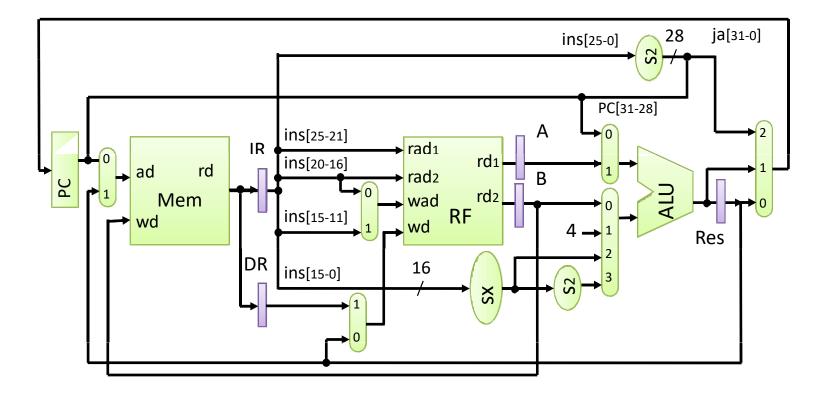
Final: Efficient Multi-Cycle Data Path

- Mux at Memory Input
- Mux at Register File Input
- One memory for lw / sw
- One ALU for EXE and Address Calculation
- Several MUXes introduced
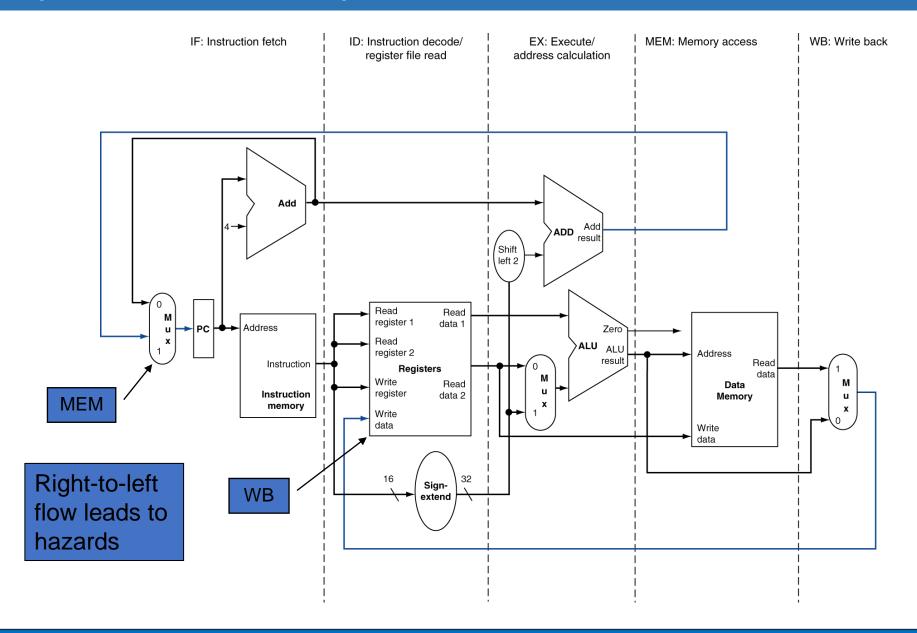
Multi-cycle Control Path

24

Multi-cycle Control Design
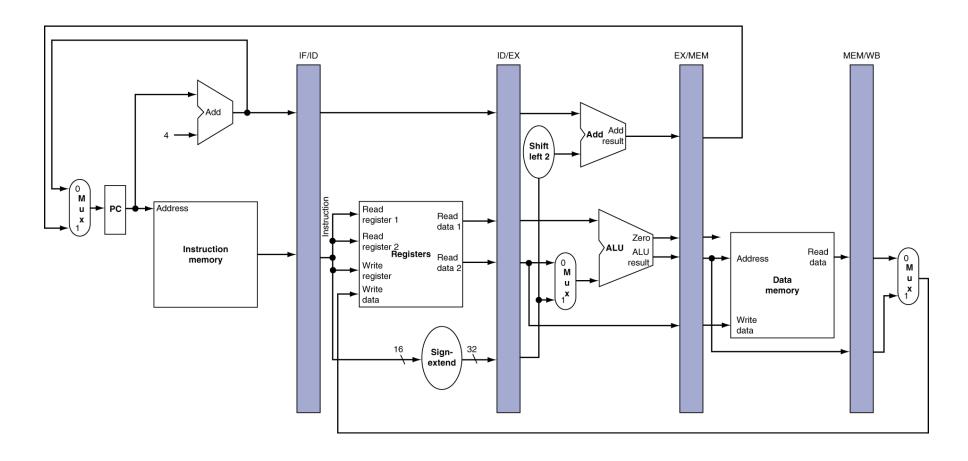
- Break instructions into cycles

- Put cycle sequences together

- Control signal groups and micro operations

- Control states and signal values

- Control state transitions

MIPS Pipelined Datapath

IF: Instruction fetch   ID: Instruction decode/ register file read   EX: Execute/ address calculation   MEM: Memory access   WB: Write back

MEM

WB

Right-to-left flow leads to hazards

Pipeline registers

- ## Need registers between stages
  - ### To hold information produced in previous cycle
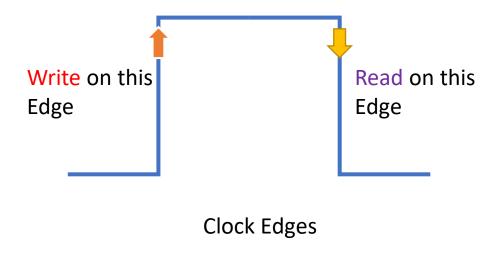
Pipeline Operation
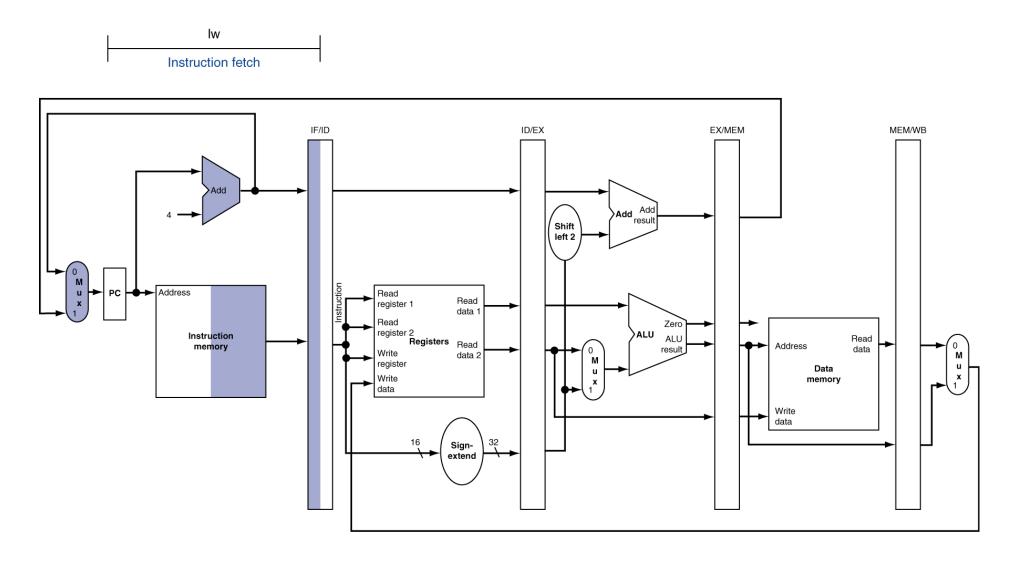
- Cycle-by-cycle flow of instructions through the pipelined datapath
  - "Single-clock-cycle" pipeline diagram
    - Shows pipeline usage in a single cycle
    - Highlight resources used
  - c.f. "multi-clock-cycle" diagram
    - Graph of operation over time

- We'll look at "single-clock-cycle" diagrams for load & store

RF is two operations in One Clock Cycle
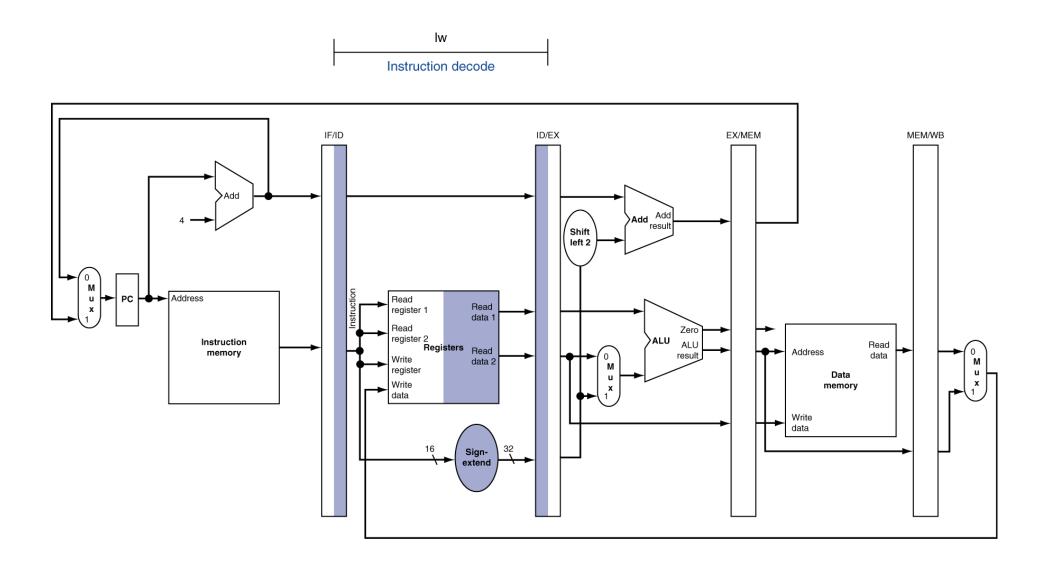
- Assume Write is when clock is going up
- Assume Read is when Clock is going down

(you can assume otherwise too, but two operations in one clock cycle)

Write on this Edge

Read on this Edge

Clock Edges
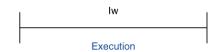
IF for Load, Store, …

ID for Load, Store, …
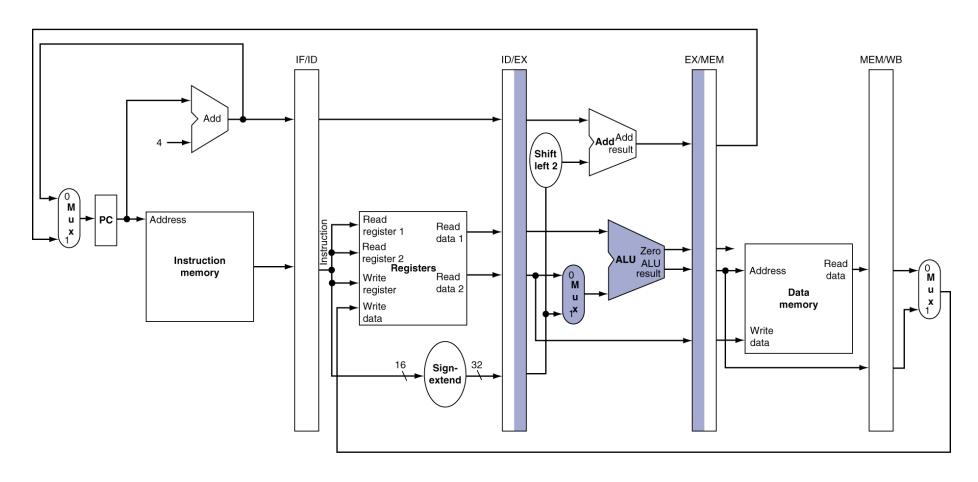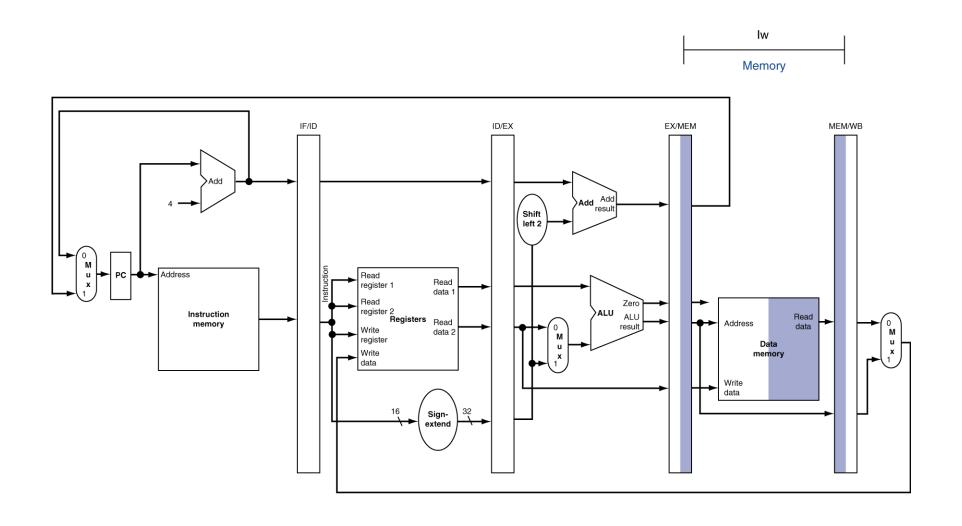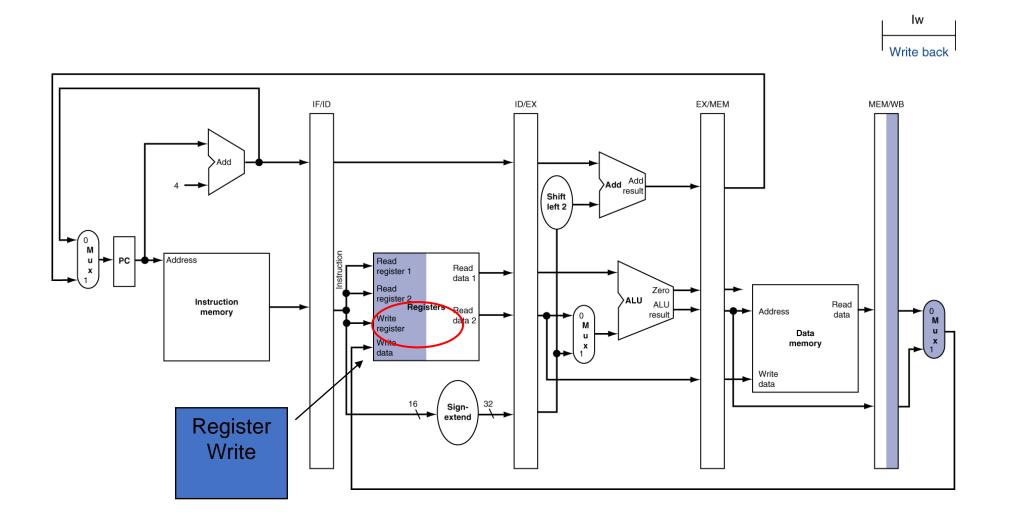
MEM for Load
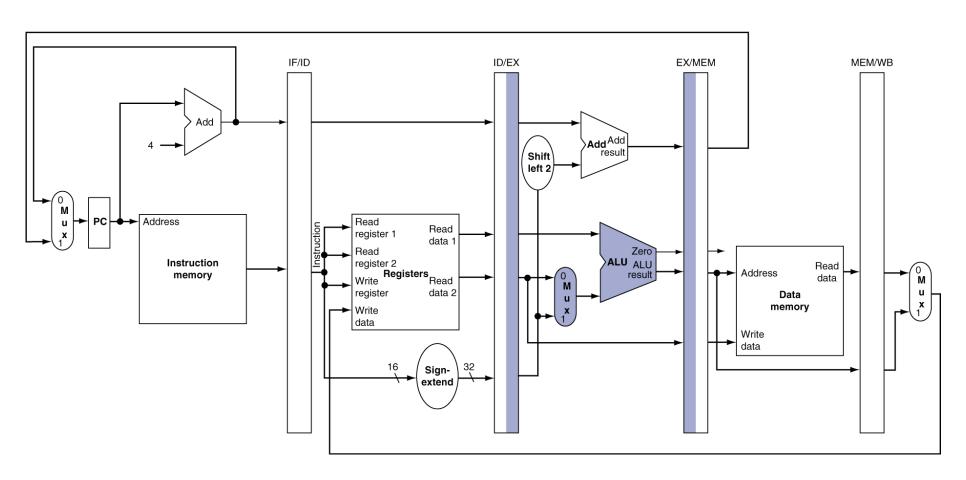
WB for Load
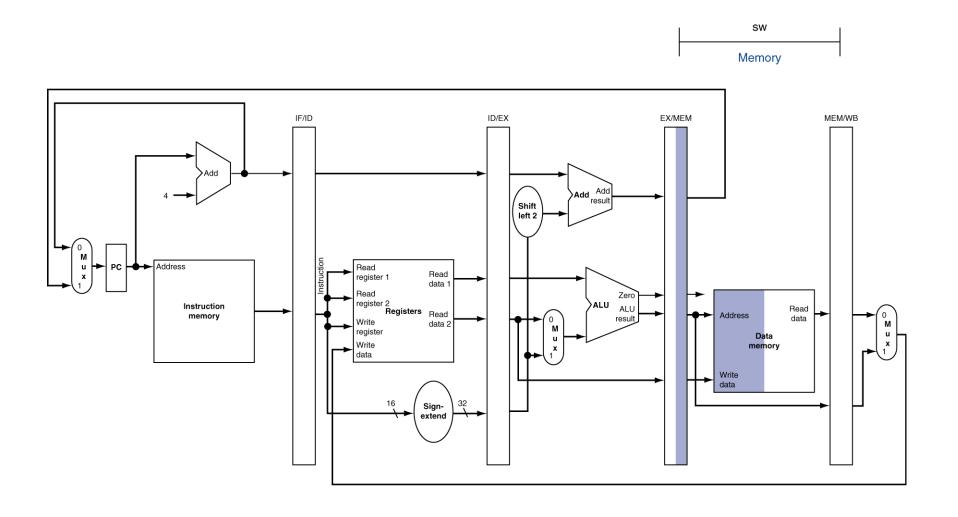
EX for Store
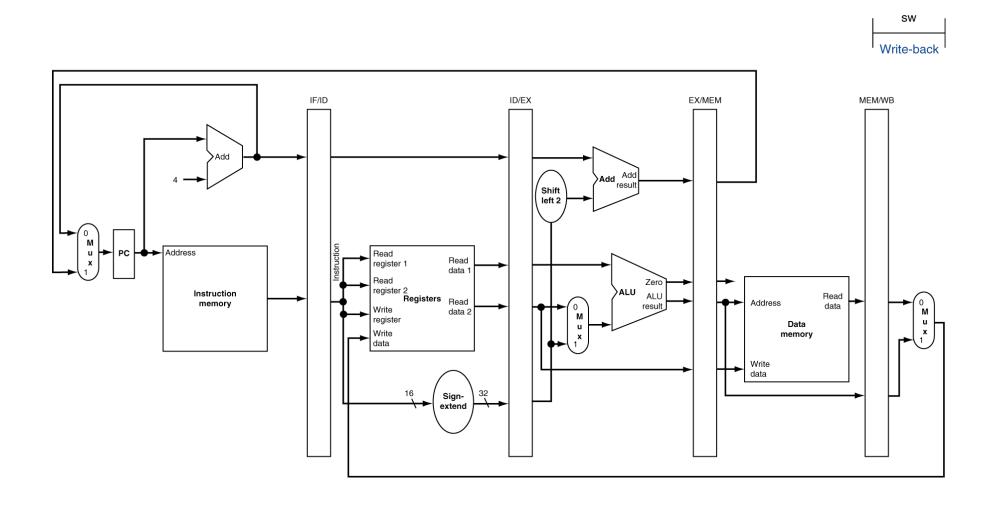
MEM for Store
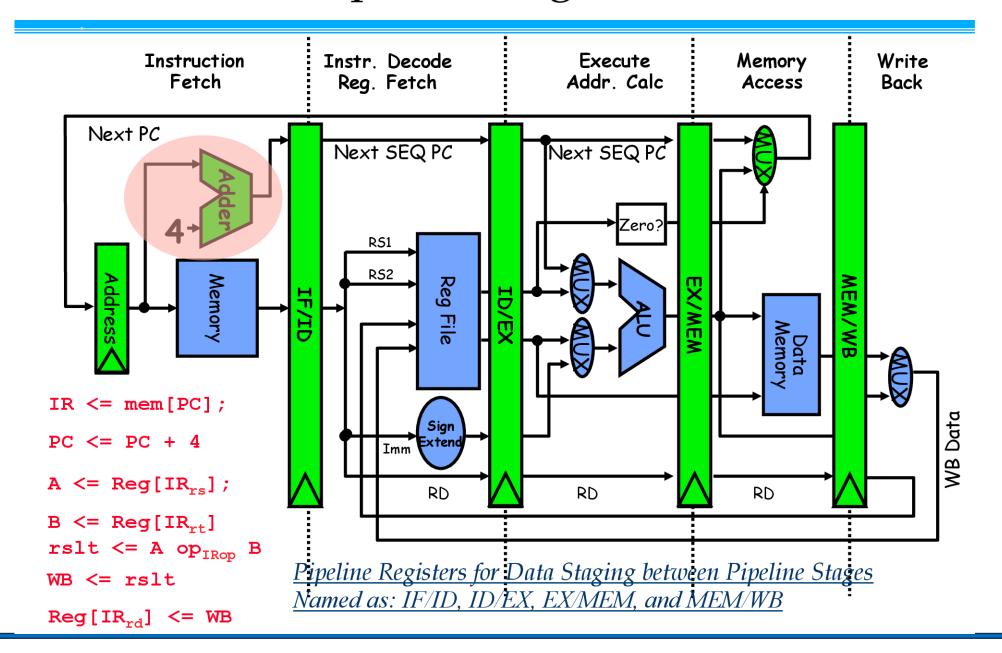
WB for Store

Pipeline Registers

Instruction Fetch | Instr. Decode Reg. Fetch | Execute Addr. Calc | Memory Access | Write Back

Next PC

Adder

4

Address

Memory

IF/ID

Next SEQ PC

RS1
RS2

Reg File

ID/EX

Next SEQ PC

Zero?

MUX

MUX

ALU

EX/MEM

MUX

Data Memory

MEM/WB

MUX

WB Data

Sign Extend

Imm

RD

RD

RD

```
IR <= mem[PC];

PC <= PC + 4

A <= Reg[IR_rs];

B <= Reg[IR_rt]
rslt <= A op_IRop B

WB <= rslt

Reg[IR_rd] <= WB
```

Pipeline Registers for Data Staging between Pipeline Stages
Named as: IF/ID, ID/EX, EX/MEM, and MEM/WB

Pipelined MIPS Datapath
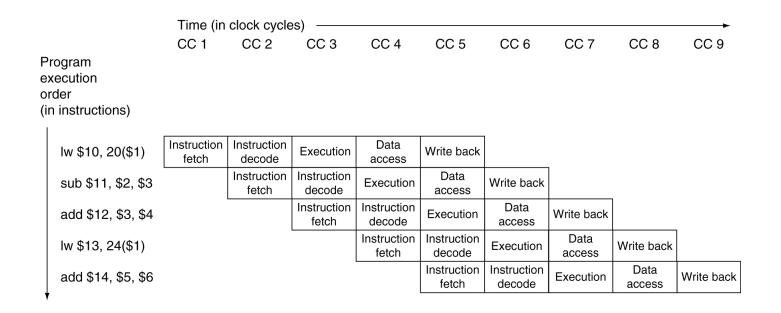
The Basic Pipeline For MIPS – Simplified Diagram
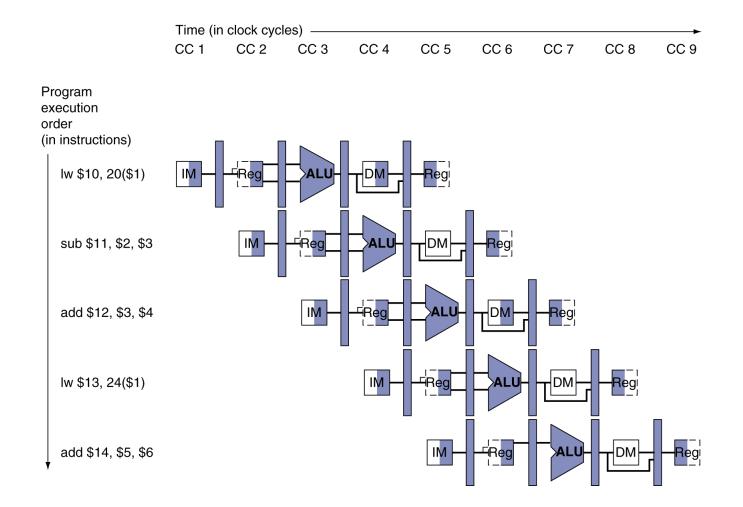
Multi-Cycle Pipeline Diagram

- Traditional form

Time (in clock cycles)

	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9

Program
execution
order
(in instructions)

Instruction	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
lw $10, 20($1)	Instruction fetch	Instruction decode	Execution	Data access	Write back				
sub $11, $2, $3		Instruction fetch	Instruction decode	Execution	Data access	Write back			
add $12, $3, $4			Instruction fetch	Instruction decode	Execution	Data access	Write back		
lw $13, 24($1)				Instruction fetch	Instruction decode	Execution	Data access	Write back	
add $14, $5, $6					Instruction fetch	Instruction decode	Execution	Data access	Write back

Multi-Cycle Pipeline Diagram

- Form showing resource usage for different instructions
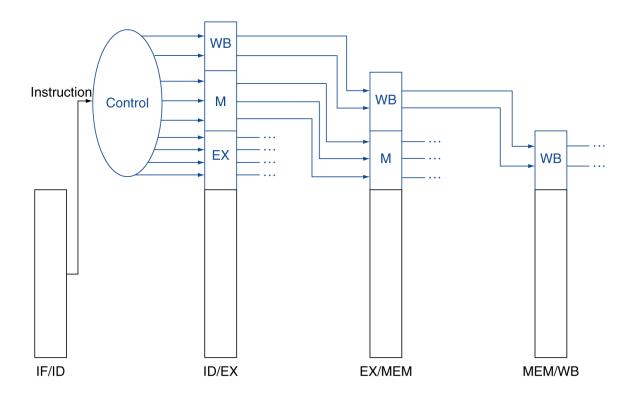
Single-Cycle Pipelined Diagram

- State of pipeline in a given cycle

add $14, $5, $6	lw $13, 24 ($1)	add $12, $3, $4	sub $11, $2, $3	lw $10, 20($1)
Instruction fetch	Instruction decode	Execution	Memory	Write-back
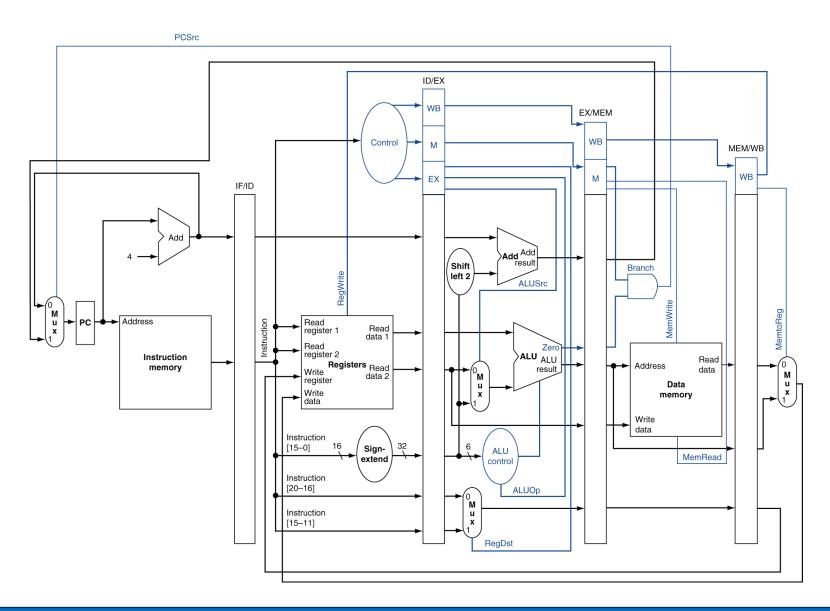
Pipelined Control

- Control signals derived from instruction
  - As in single-cycle implementation

Pipelined Control with Connections
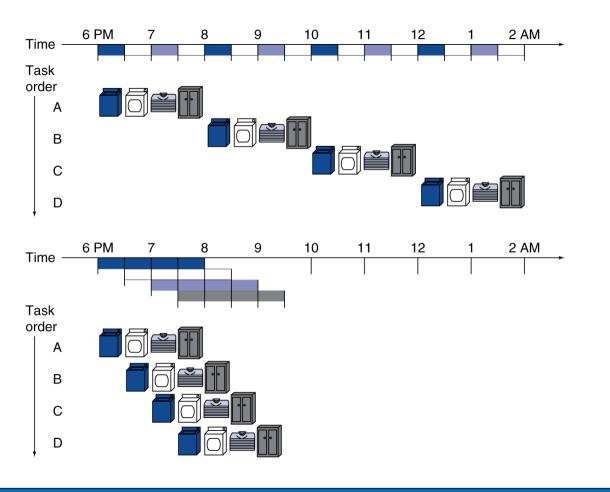
Pipeline Performance Estimation

Pipelining Analogy

- Pipelined laundry: overlapping execution
  - Parallelism improves performance

- Four loads:
  - Speedup = 8/3.5 = 2.3

Pipelined Architecture

D **Example program**

CT=1.5ns; CPU Time = 9 x 1 x 1.5ns = 13.5ns

AND R1,R2,R3

What is the cost of pipelining?

XOR R4,R2,R3

SUB R5,R1,R4

ADD R6,R1,R4

MUL R7,R5,R6

Time

CPU Time = IC x CPI x CT
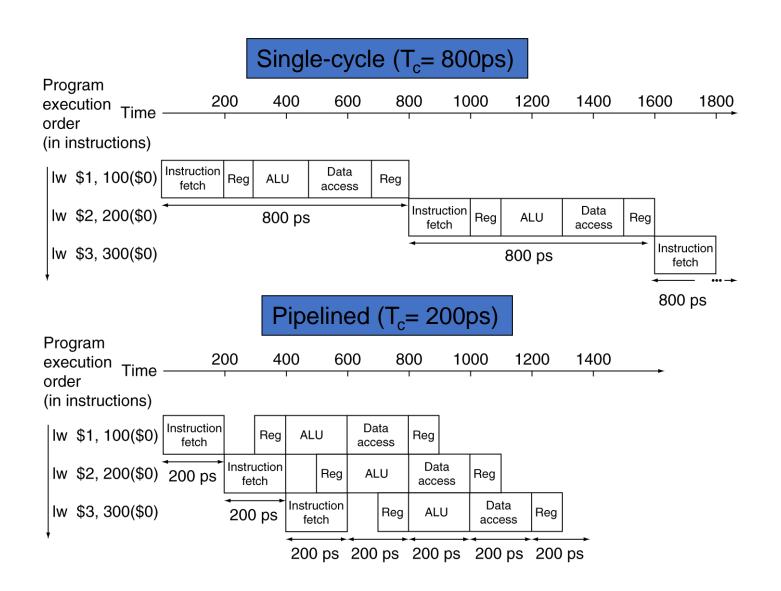
Pipeline Performance

- Assume time for stages is
  - 100ps for register read or write
  - 200ps for other stages
- Compare pipelined datapath with single-cycle datapath

| Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

Pipeline Performance

Single-cycle (T_c= 800ps)

Program execution order (in instructions)

Time: 200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800

lw $1, 100($0): Instruction fetch | Reg | ALU | Data access | Reg — 800 ps

lw $2, 200($0): Instruction fetch | Reg | ALU | Data access | Reg — 800 ps

lw $3, 300($0): Instruction fetch — 800 ps

Pipelined (T_c= 200ps)

Program execution order (in instructions)

Time: 200, 400, 600, 800, 1000, 1200, 1400

lw $1, 100($0): Instruction fetch | Reg | ALU | Data access | Reg

lw $2, 200($0): 200 ps | Instruction fetch | Reg | ALU | Data access | Reg

lw $3, 300($0): 200 ps | Instruction fetch | Reg | ALU | Data access | Reg

200 ps 200 ps 200 ps 200 ps 200 ps

Readings

- Chap 4 of P&H Textbook