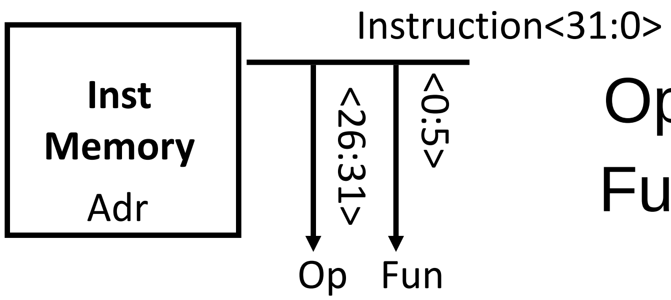
CS 61C: Great Ideas in Computer Architecture (Machine Structures) Lecture 30: Pipeline Parallelism 1

Lecturer:
Alan Christopher

Boolean Exprs for Controller



Op 0-5 are really Instruction bits 26-31

Func 0-5 are really Instruction bits 0-5

rtype $= \sim op_5 \bullet \sim op_4 \bullet \sim op_3 \bullet \sim op_2 \bullet \sim op_1 \bullet \sim op_0$,

ori $= \sim op_5 \bullet \sim op_4 \bullet op_3 \bullet op_2 \bullet \sim op_1 \bullet op_0$

lw $= op_5 \bullet \sim op_4 \bullet \sim op_3 \bullet \sim op_2 \bullet op_1 \bullet op_0$

sw $= op_5 \bullet \sim op_4 \bullet op_3 \bullet \sim op_2 \bullet op_1 \bullet op_0$

beq $= \sim op_5 \bullet \sim op_4 \bullet \sim op_3 \bullet op_2 \bullet \sim op_1 \bullet \sim op_0$

jump $= \sim op_5 \bullet \sim op_4 \bullet \sim op_3 \bullet \sim op_2 \bullet op_1 \bullet \sim op_0$

add = **rtype** $\bullet func_5 \bullet \sim func_4 \bullet \sim func_3 \bullet \sim func_2 \bullet \sim func_1 \bullet \sim func_0$

sub = **rtype** $\bullet func_5 \bullet \sim func_4 \bullet \sim func_3 \bullet \sim func_2 \bullet func_1 \bullet \sim func_0$

How do we implement this in gates?

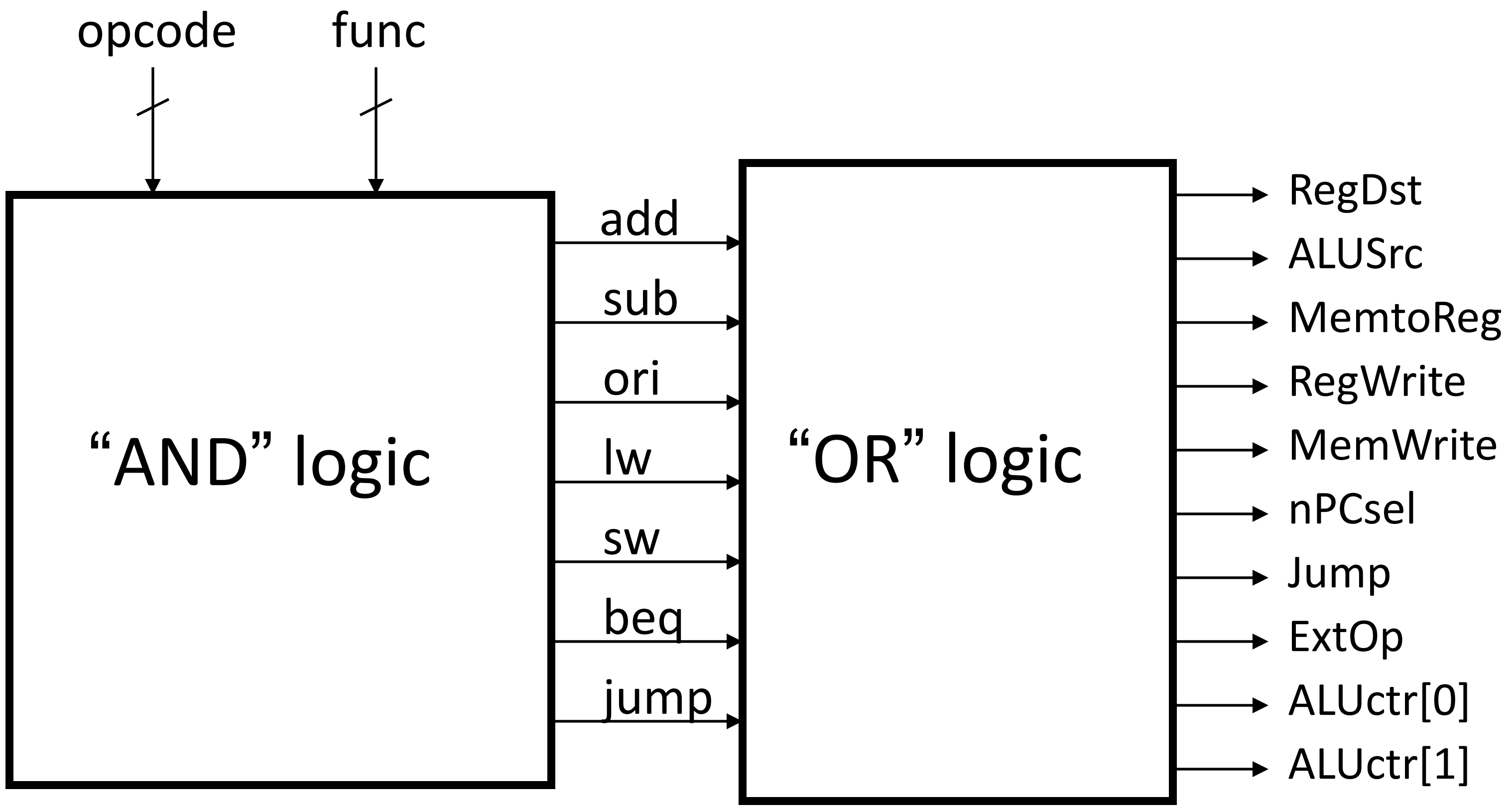
Boolean Exprs for Controller

RegDst = add + sub
ALUSrc = ori + lw + sw
MemtoReg = lw
RegWrite = add + sub + ori + lw
MemWrite = sw
nPCsel = beq
Jump = jump
ExtOp = lw + sw
ALUctr[0] = sub + beq
ALUctr[1] = ori

(assume ALUctr is 00 ADD, 01 SUB, 10 OR)

How do we implement this in gates?

Controller Implementation



Call home, we've made HW/SW contact!

High Level Language Program (e.g., C)

Compiler

Assembly Language Program (e.g.,MIPS)

Assembler

Machine Language Program (MIPS)

Machine Interpretation

Hardware Architecture Description (e.g., block diagrams)

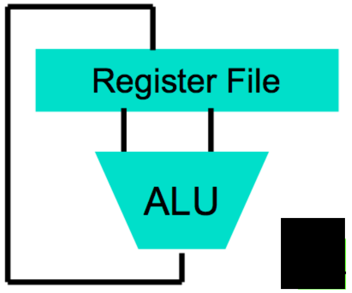
Architecture Implementation

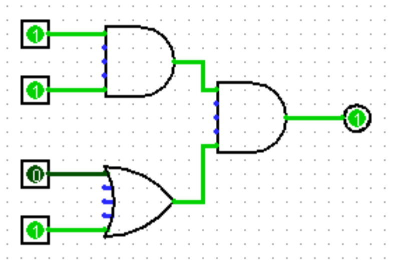
Logic Circuit Description (Circuit Schematic Diagrams)

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw    $t0, 0($2)
lw    $t1, 4($2)
sw    $t1, 0($2)
sw    $t0, 4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```





Administrivia/Clicker

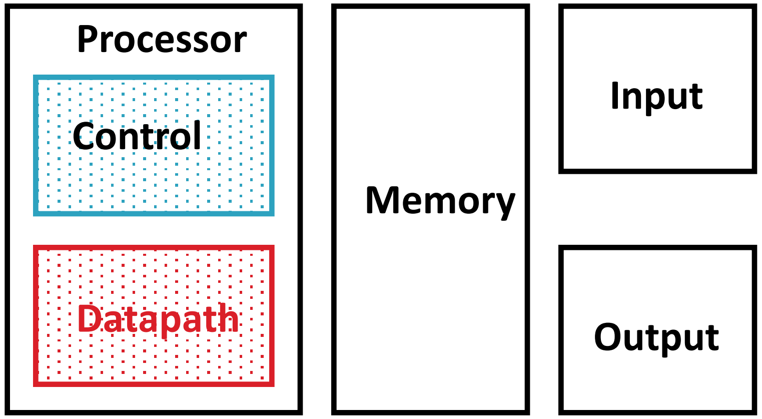
- How many hours of fun from proj3 so far?
  a) $0 <= F <= 4$
  b) $4 < F <= 8$
  c) $8 < F <= 12$
  d) $12 < F <= 16$
  e) $16 < F$

Administrivia/Clicker

- How many Gflop/s right now?
  a) $0 <= F <= 4$
  b) $4 < F <= 8$
  c) $8 < F <= 12$
  d) $12 < F <= 16$
  e) $16 < F$

Review: Single-cycle Processor

- Five steps to design a processor:
  1. Analyze instruction set → datapath requirements
  2. Select set of datapath components & establish clock methodology
  3. Assemble datapath meeting the requirements
  4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
  5. Assemble the control logic
     - Formulate Logic Equations
     - Design Circuits



Single Cycle Performance

- Assume time for actions are
  - 100ps for register read or write; 200ps for other events
- Clock rate is?

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

- What can we do to improve clock rate?
- Will this improve performance as well?

  Want increased clock rate to mean faster programs

Single Cycle Performance

- Assume time for actions are
  - 100ps for register read or write; 200ps for other events
- Clock rate is?

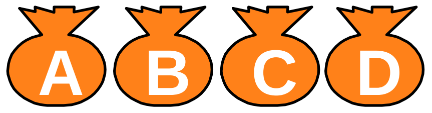
Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

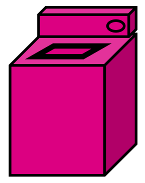
- What can we do to improve clock rate?
- Will this improve performance as well?

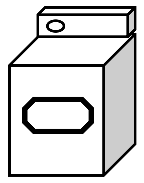
  Want increased clock rate to mean faster programs

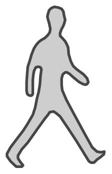
Gotta Do Laundry

- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, fold, and put away
  - Washer takes 30 minutes
  - Dryer takes 30 minutes
  - "Folder" takes 30 minutes
  - "Stasher" takes 30 minutes to put clothes into drawers

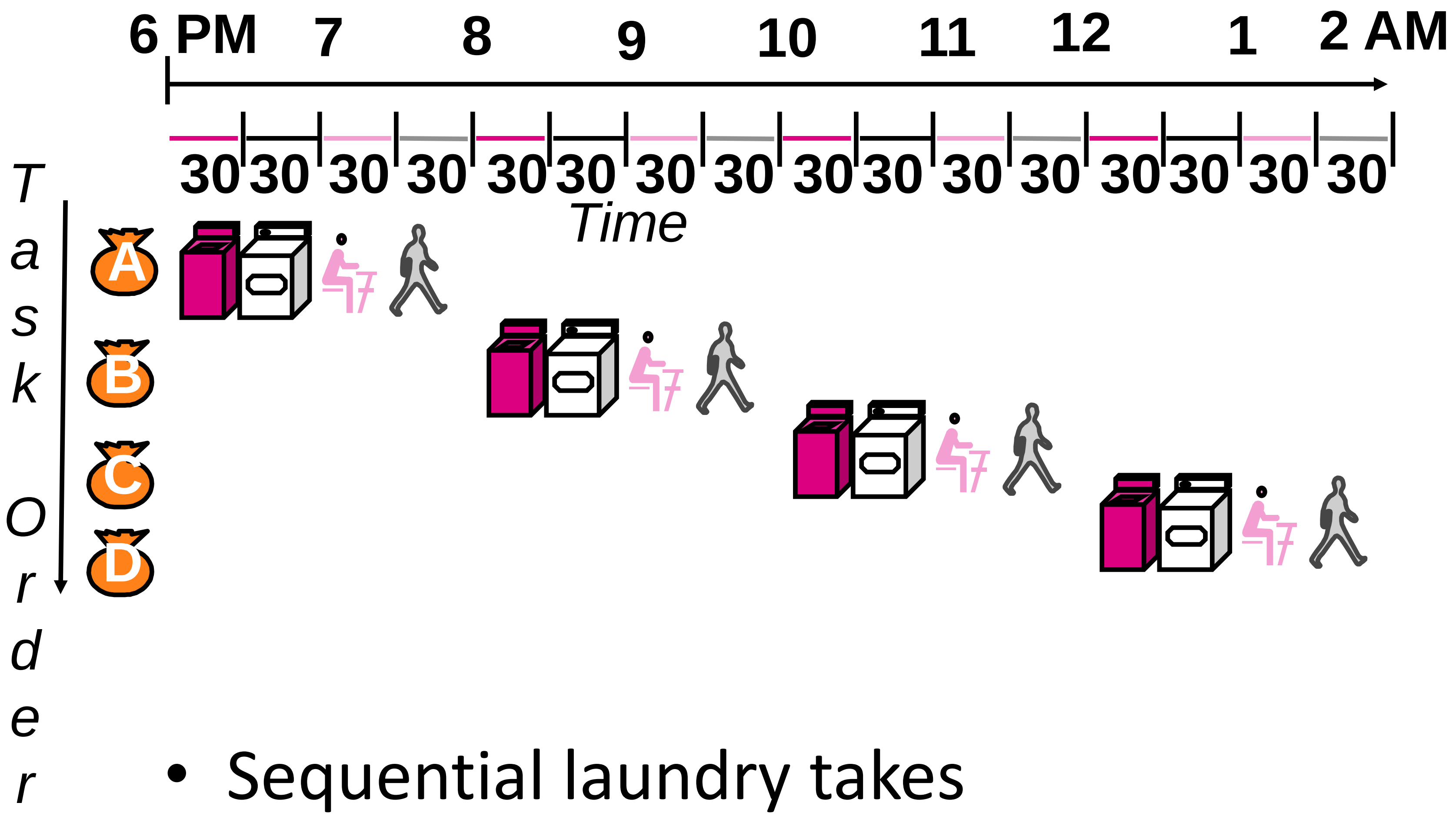






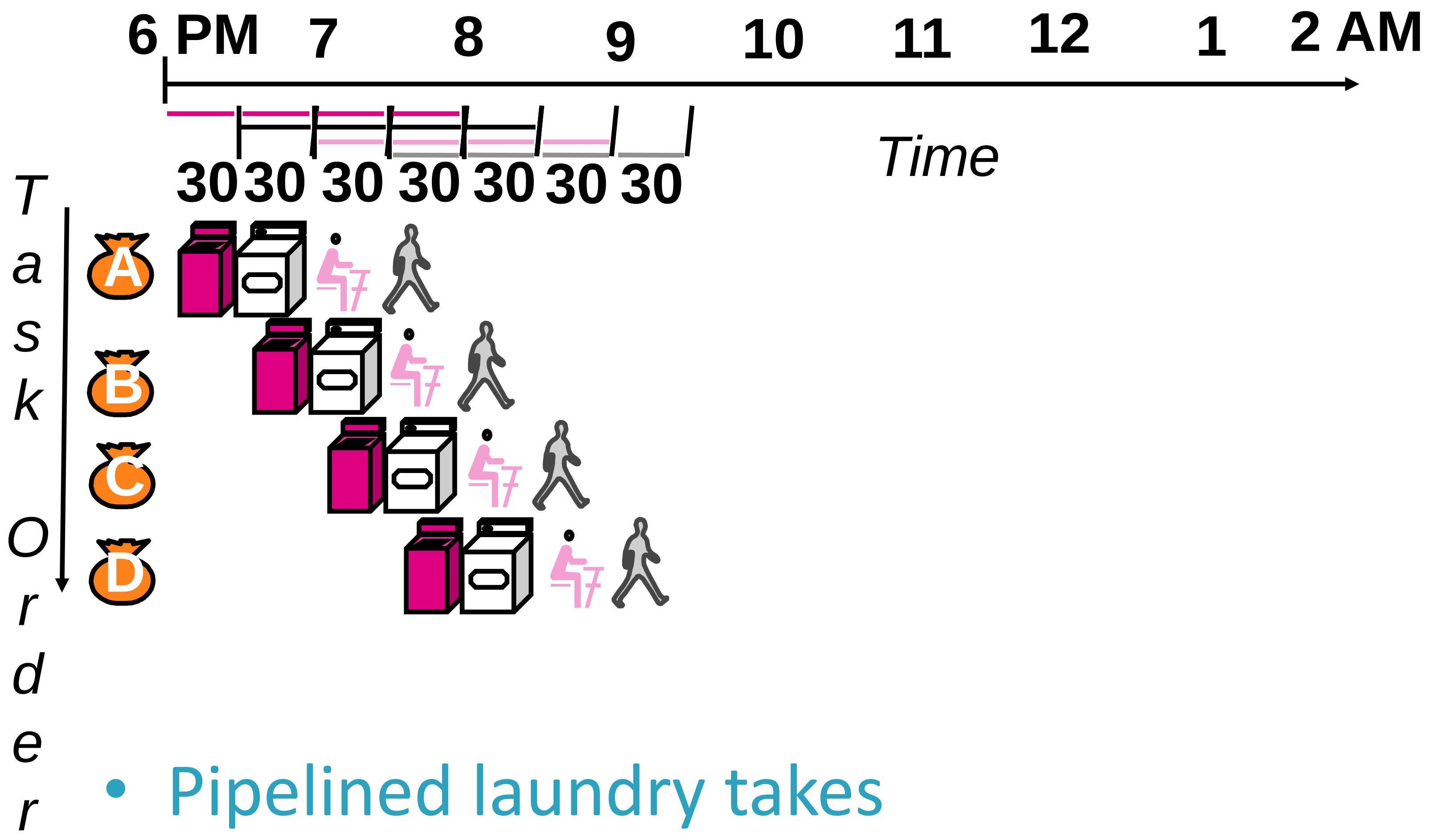


Sequential Laundry



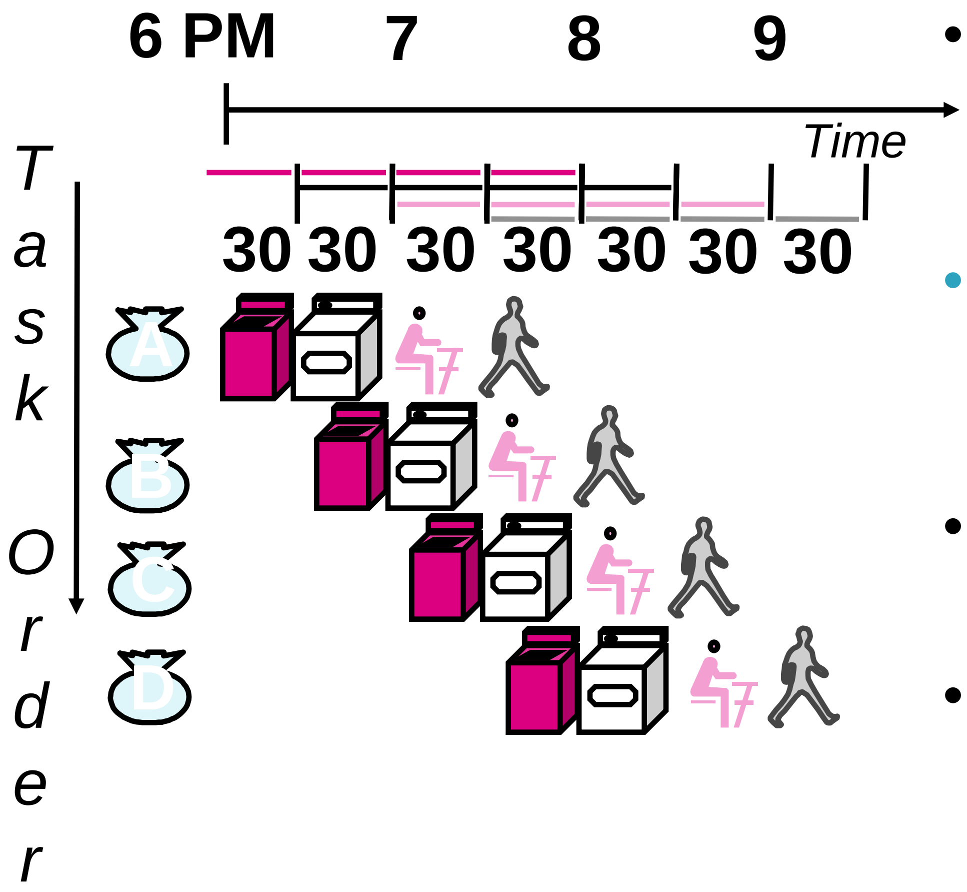
- Sequential laundry takes 8 hours for 4 loads

Pipelined Laundry



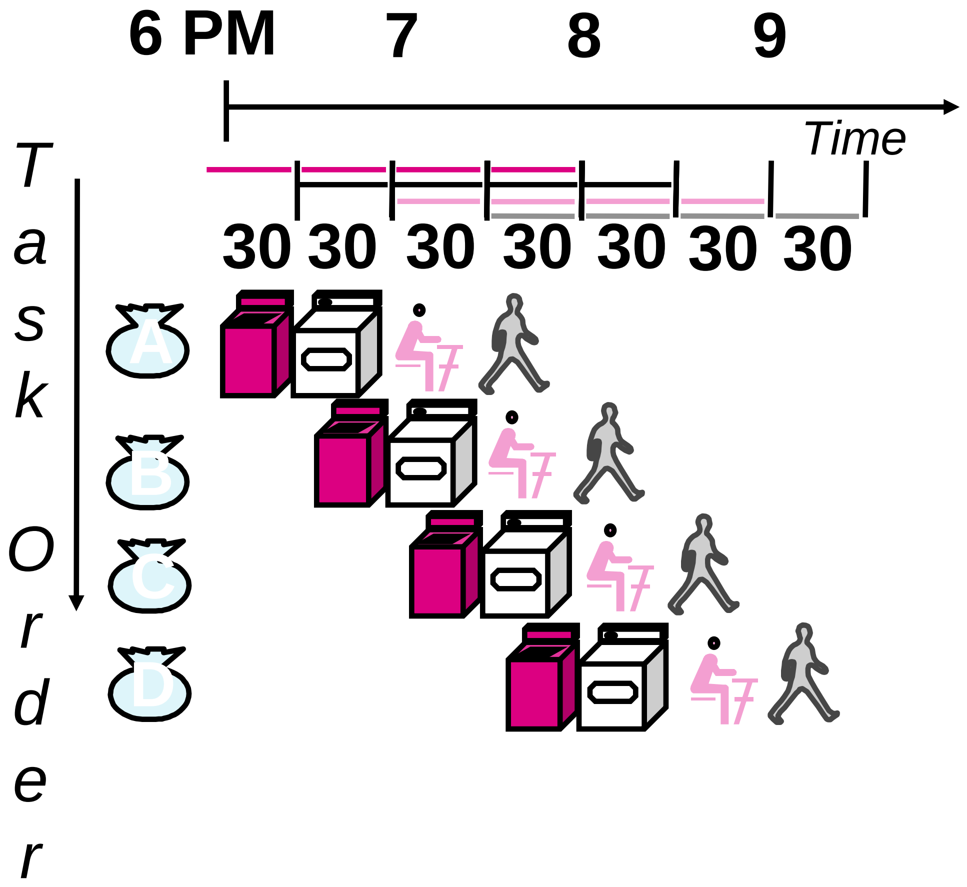
- Pipelined laundry takes 3.5 hours for 4 loads!

Pipelining Lessons (1/2)



- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Multiple tasks operating simultaneously using different resources
- Potential speedup = Number pipe stages
- Time to "fill" pipeline and time to "drain" it reduces speedup: 2.3X v. 4X in this example

Pipelining Lessons (2/2)

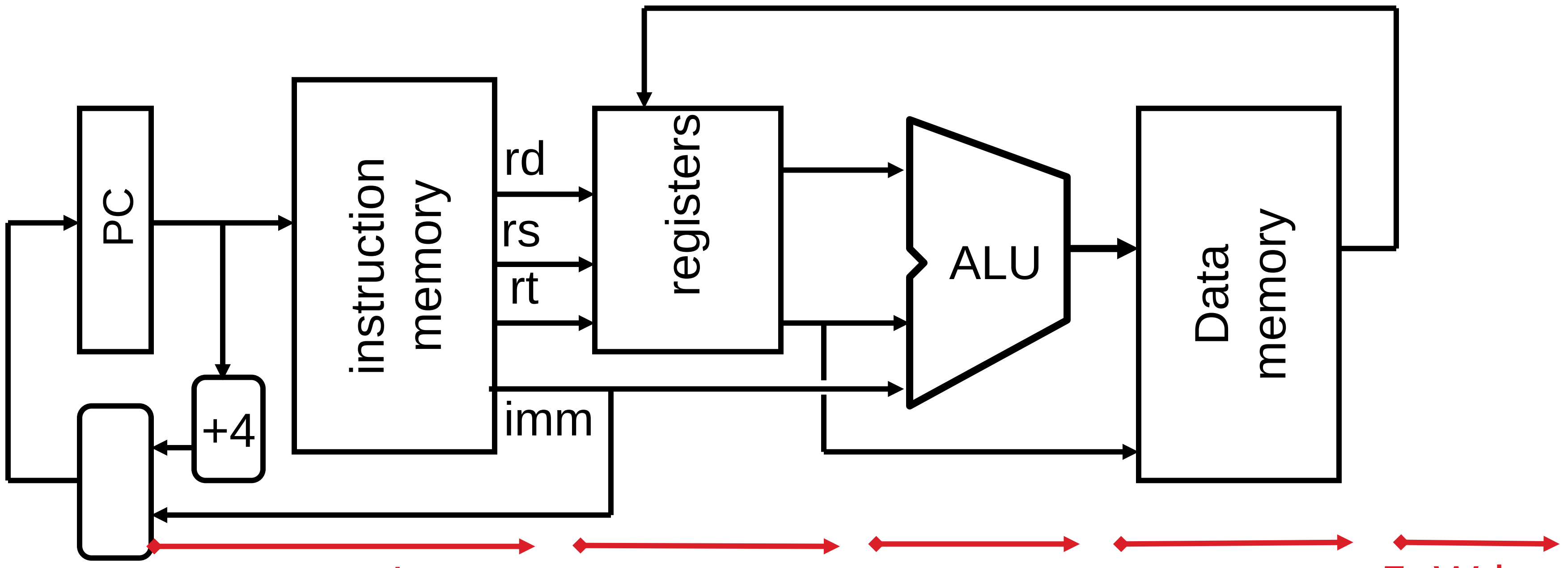


- Suppose new Washer takes 20 minutes, new Stasher takes 20 minutes. How much faster is pipeline?
- Pipeline rate limited by slowest pipeline stage
- Unbalanced lengths of pipe stages reduces speedup

Steps in Executing MIPS

1) IFtch: Instruction Fetch, Increment PC
2) Dcd: Instruction Decode, Read Registers
3) Exec:
   - Mem-ref: Calculate Address
   - Arith-log: Perform Operation
4) Mem:
   - Load: Read Data from Memory
   - Store: Write Data to Memory
5) WB: Write Data Back to Register

Single Cycle Datapath



1. Instruction Fetch

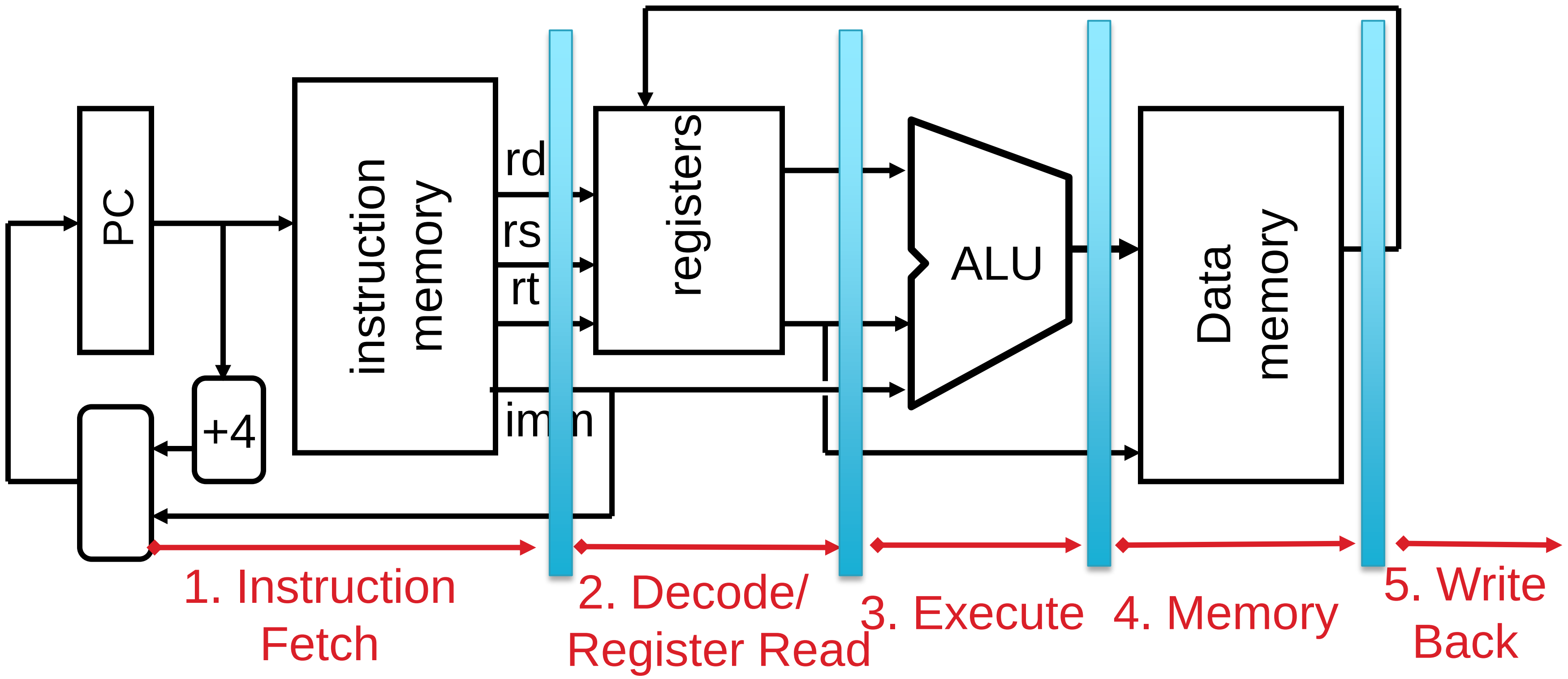
2. Decode/ Register Read

3. Execute

4. Memory

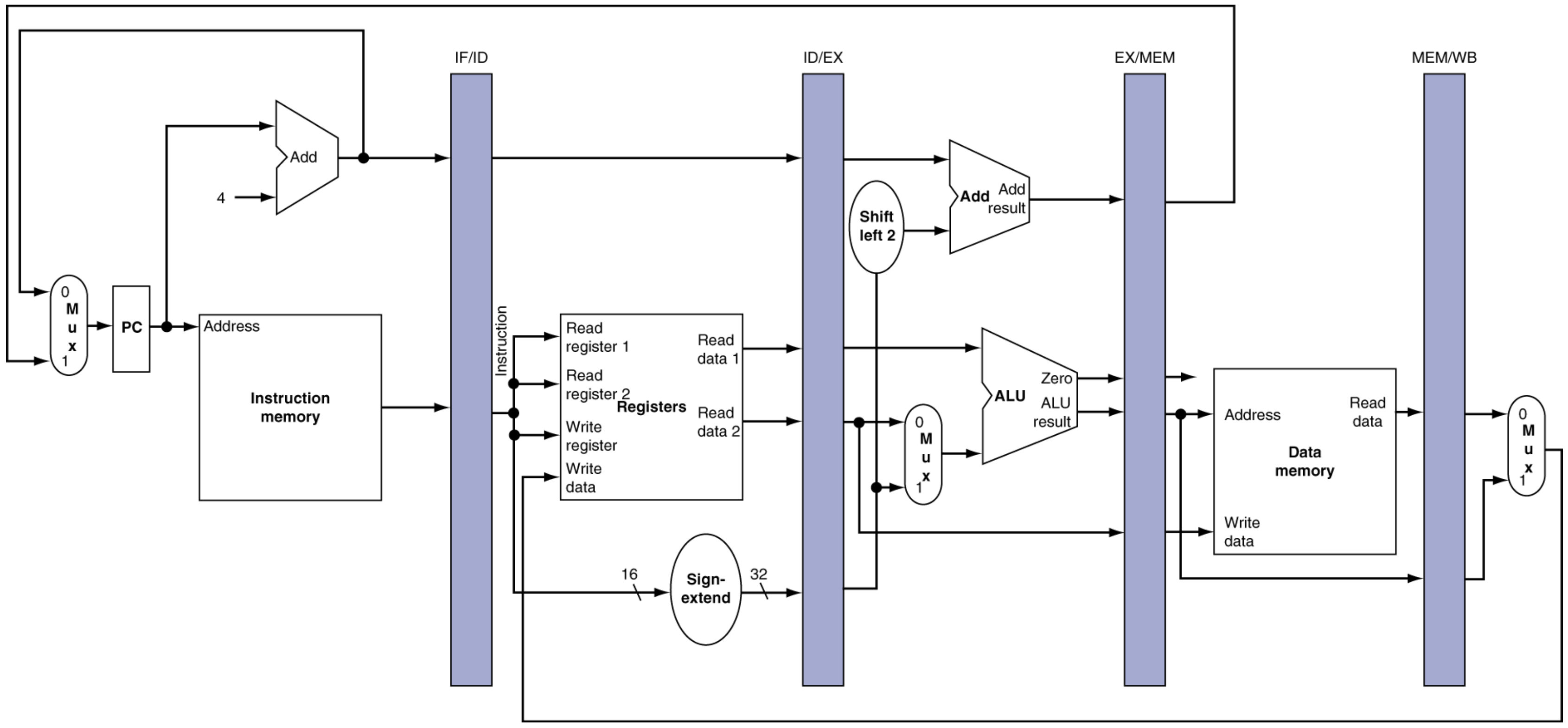
5. Write Back

Pipeline registers

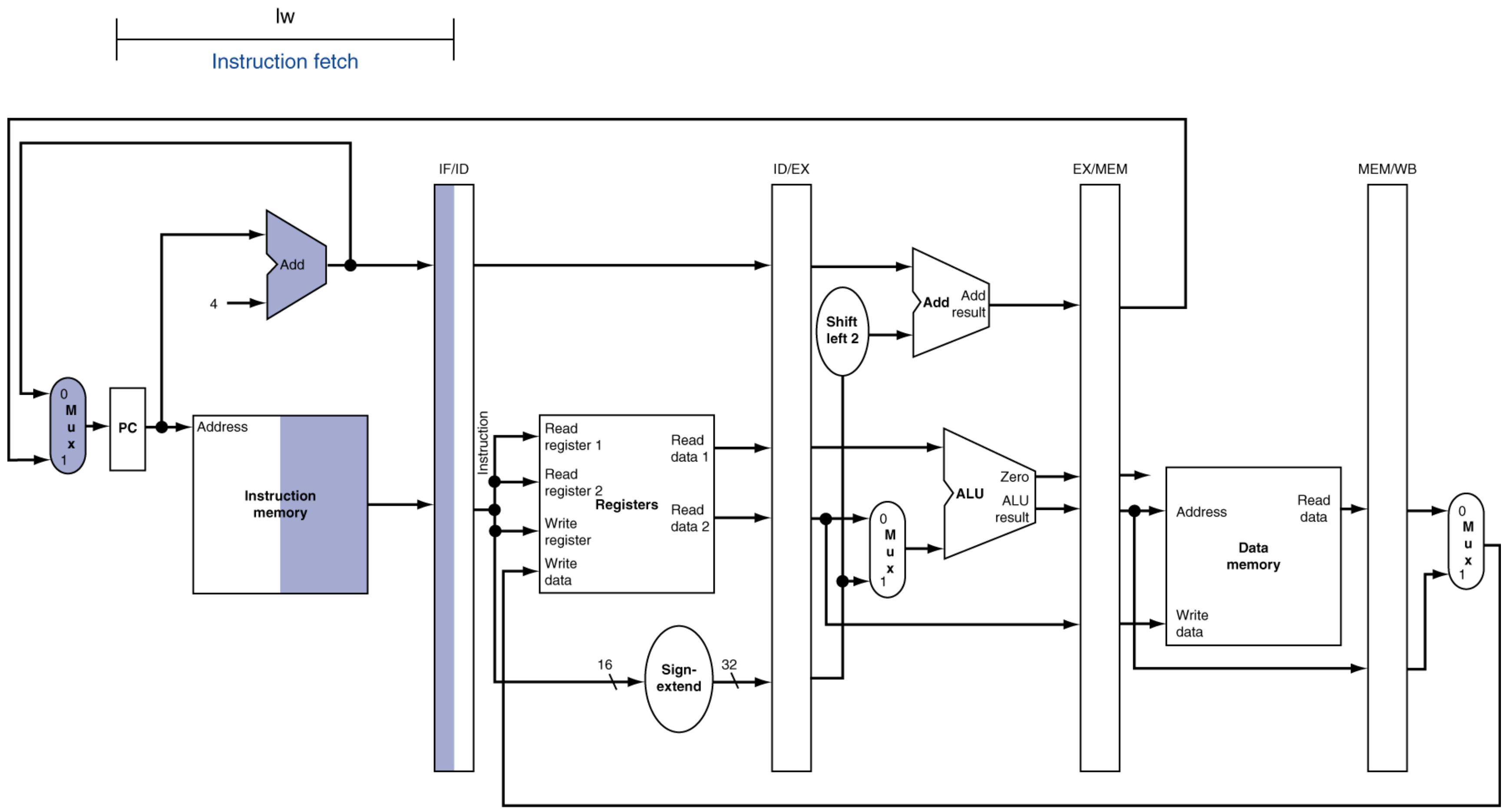


- Need registers between stages
  - To hold information produced in previous cycle

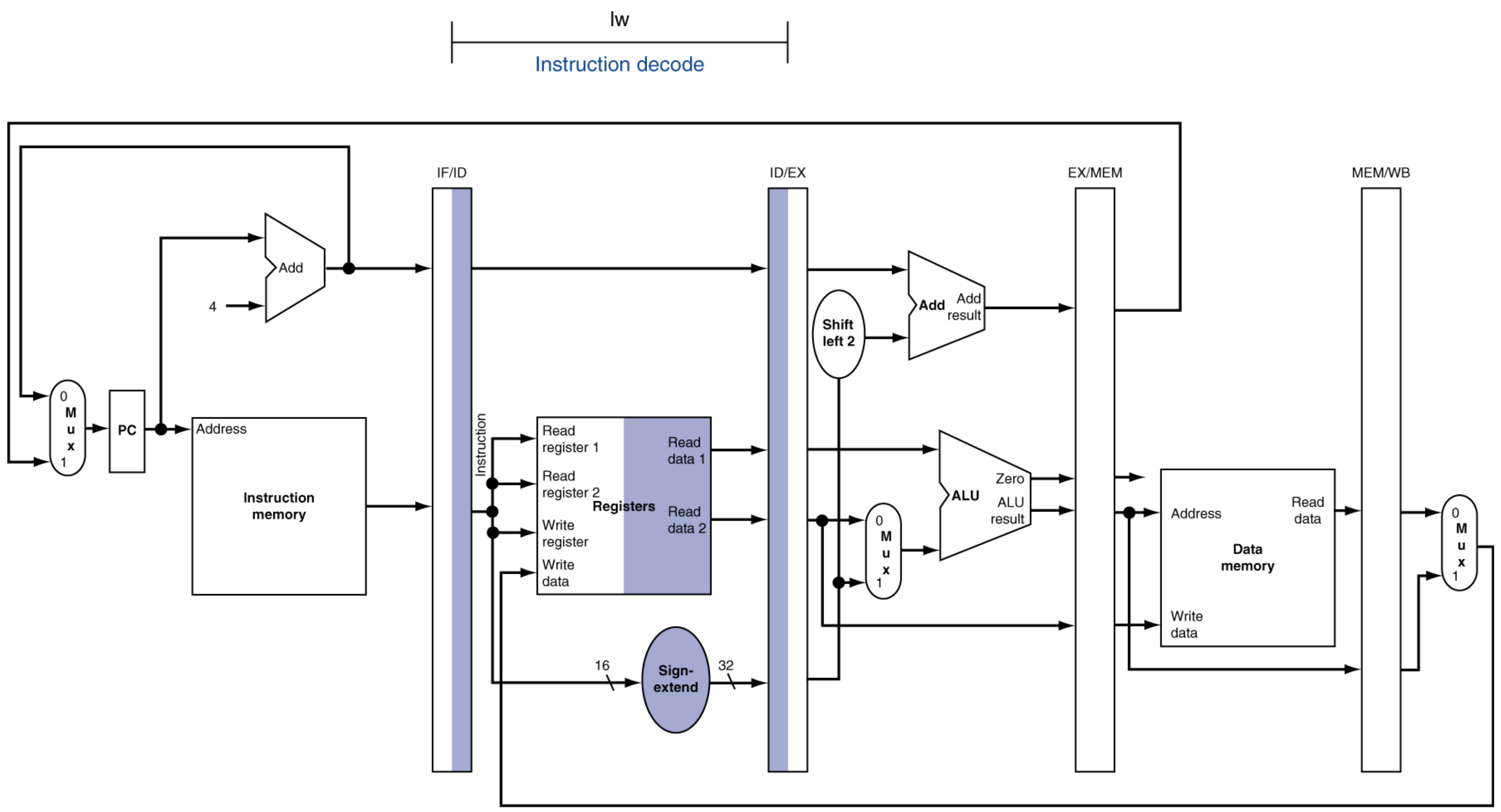
More Detailed Pipeline



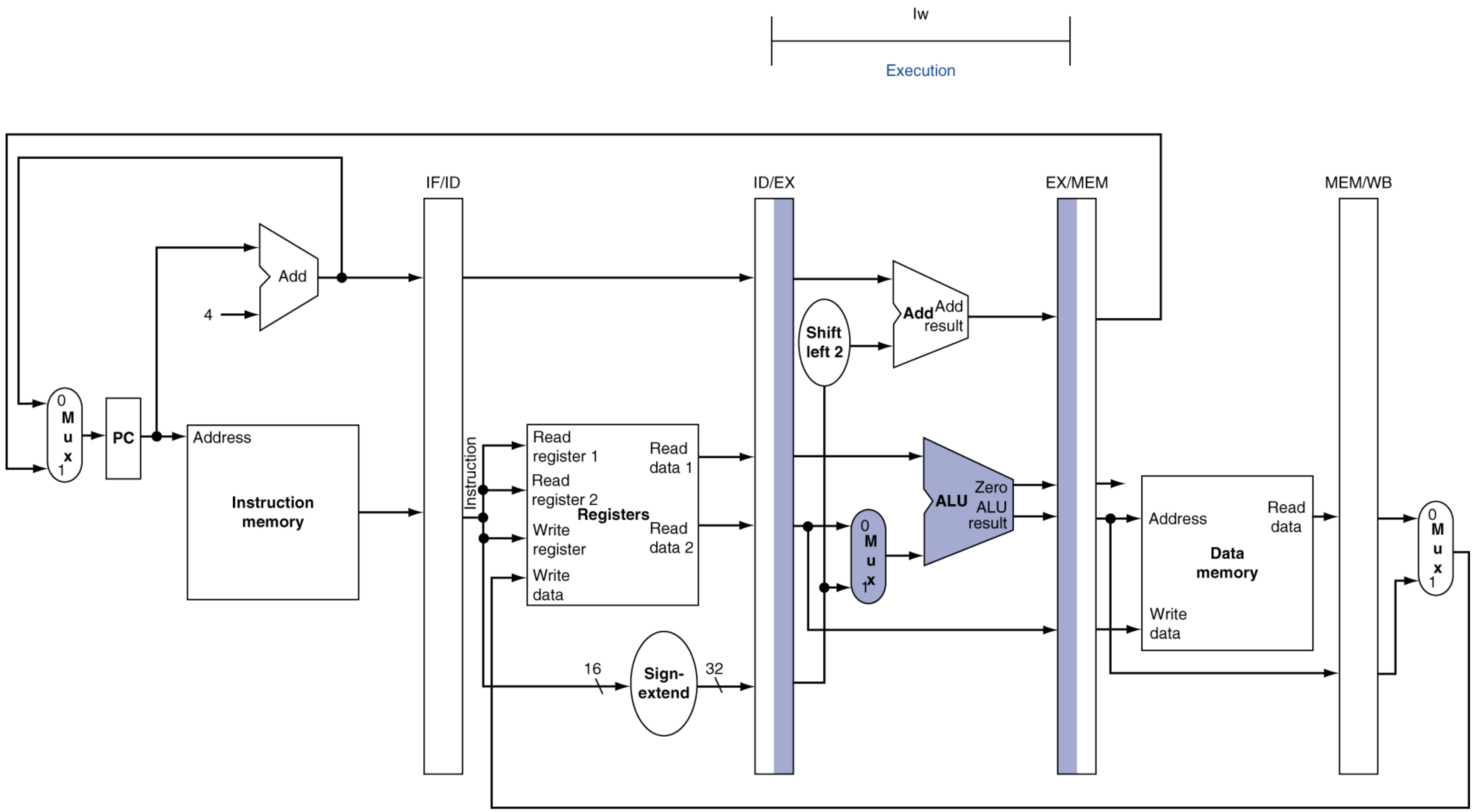
IF for Load, Store, …



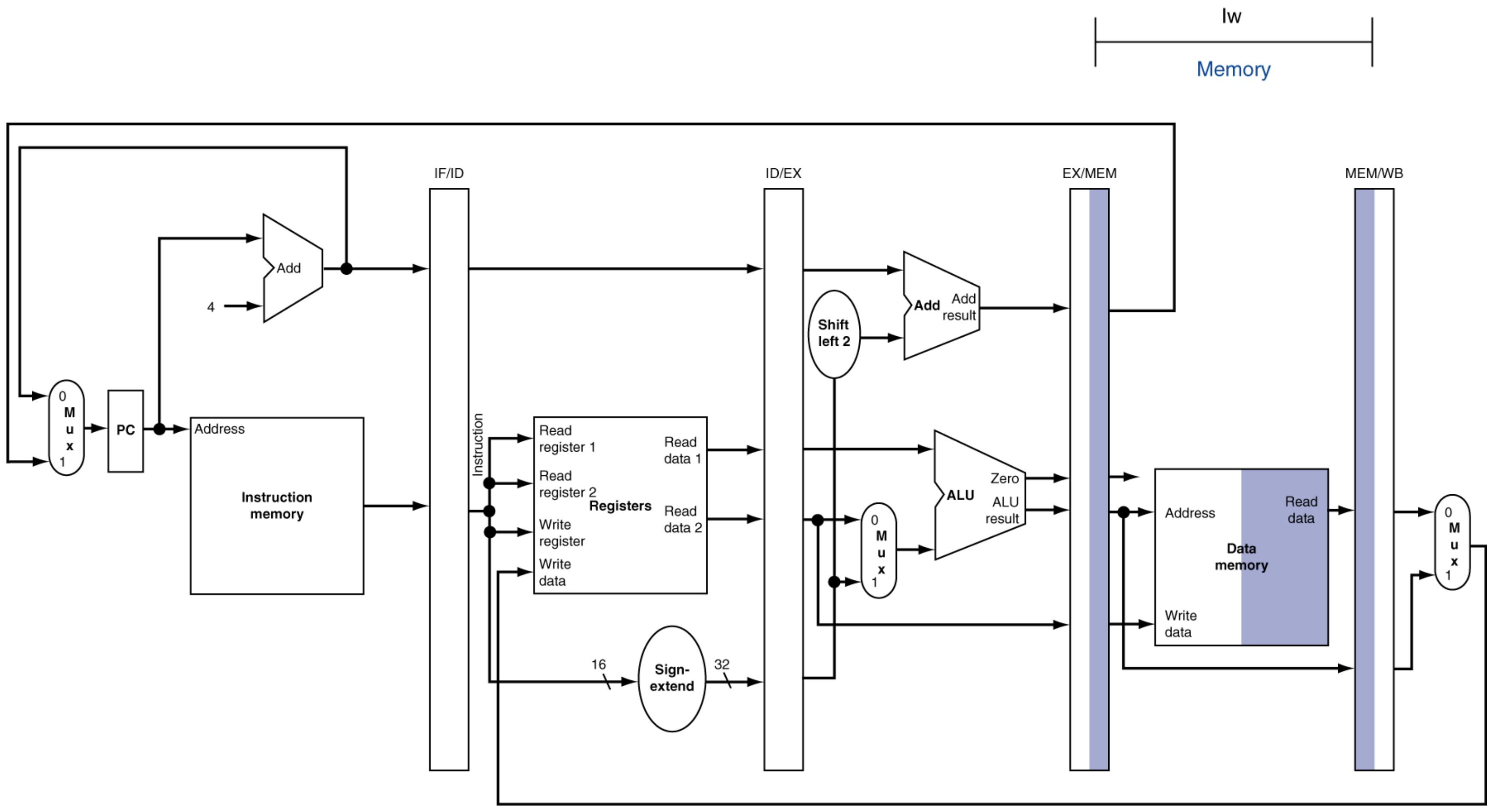
ID for Load, Store, …



EX for Load

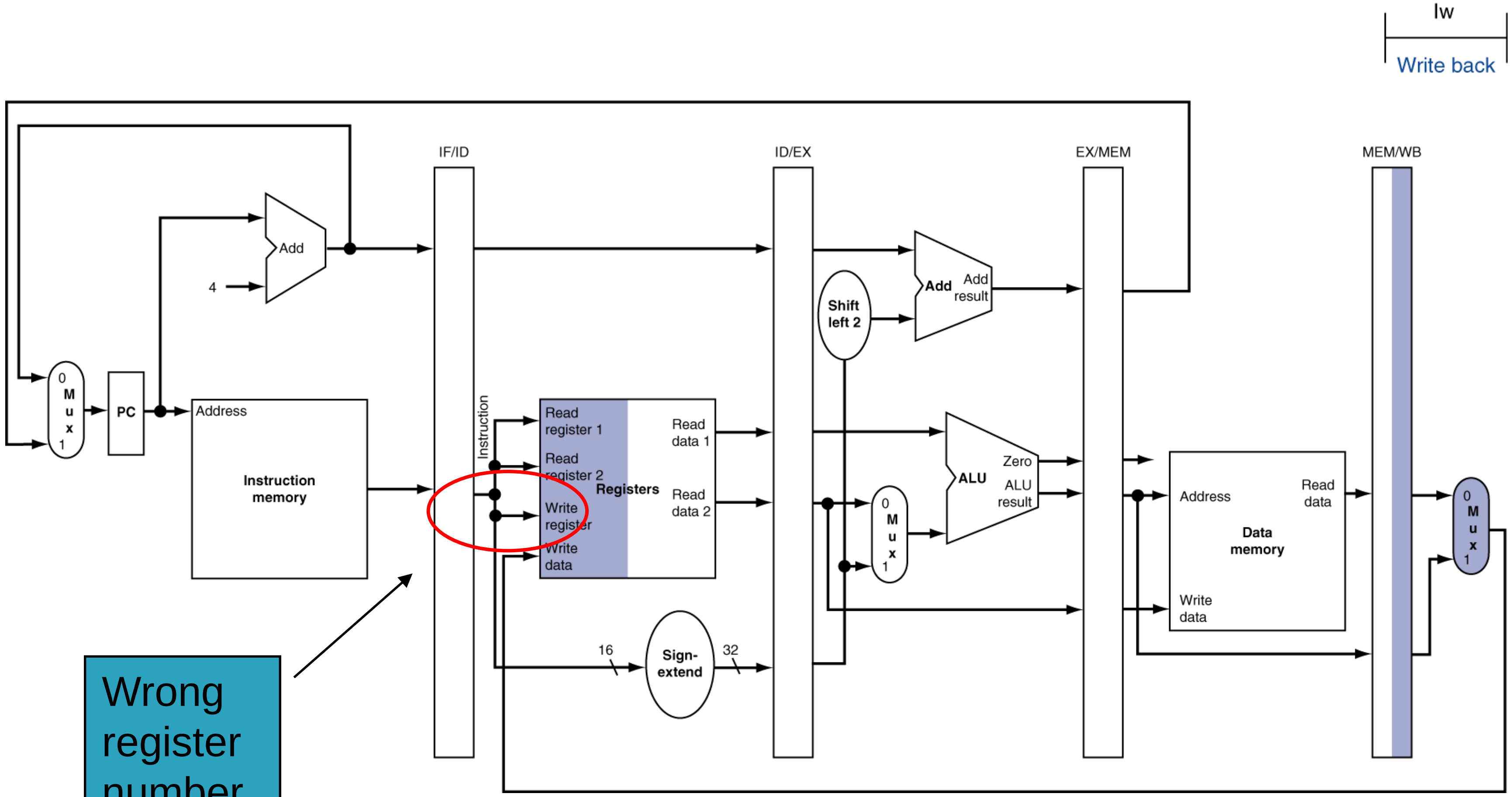

lw
Execution
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
0
M
u
x
1
Address
Data memory
Write data
Read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

MEM for Load



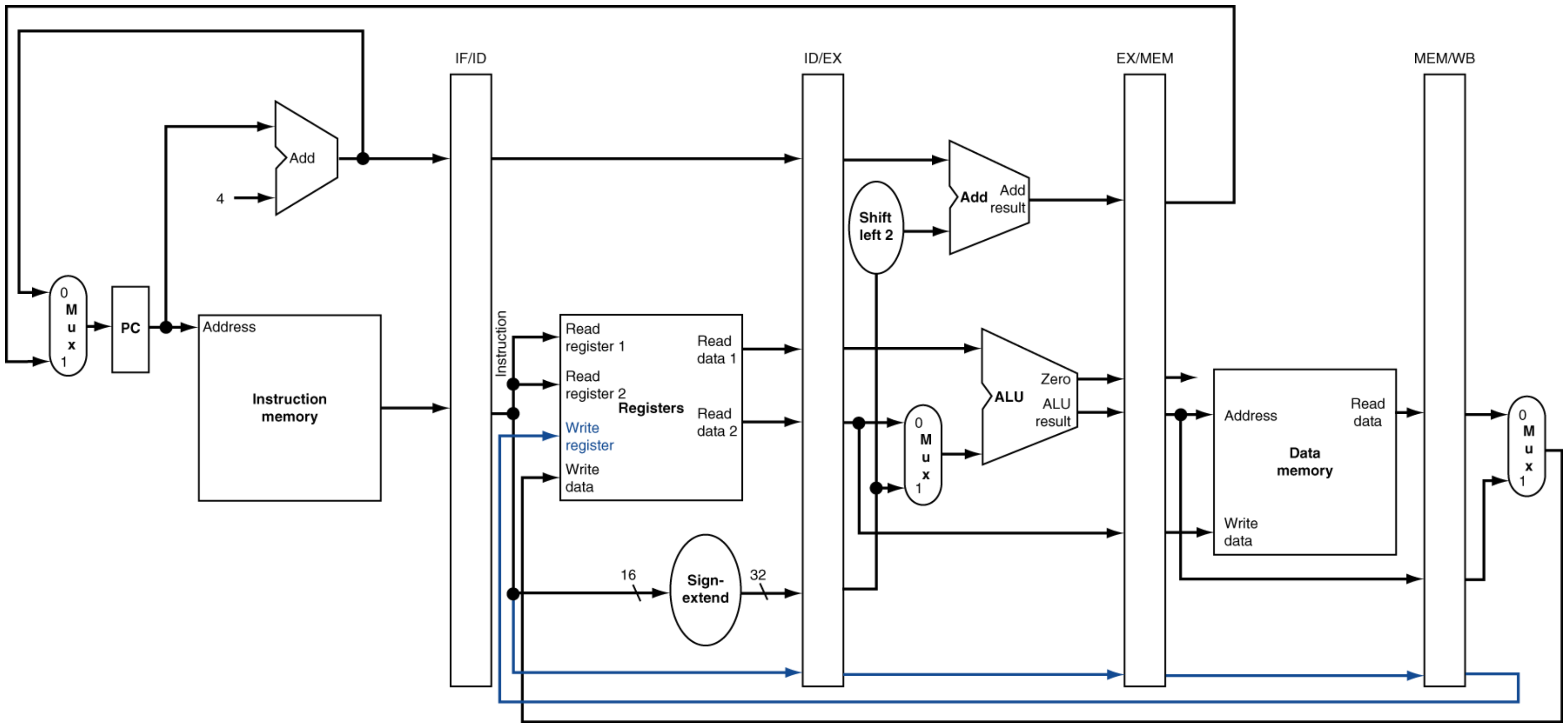
lw
Memory
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data

WB for Load – Oops!



Wrong register number

Corrected Datapath for Load



So, in conclusion

- You now know how to implement the control logic for the single-cycle CPU.
  - (actually, you already knew it!)
- Pipelining improves performance by increasing instruction throughput: exploits ILP
  - Executes multiple instructions in parallel
  - Each instruction has the same latency
- Next: hazards in pipelining:
  - Structure, data, control