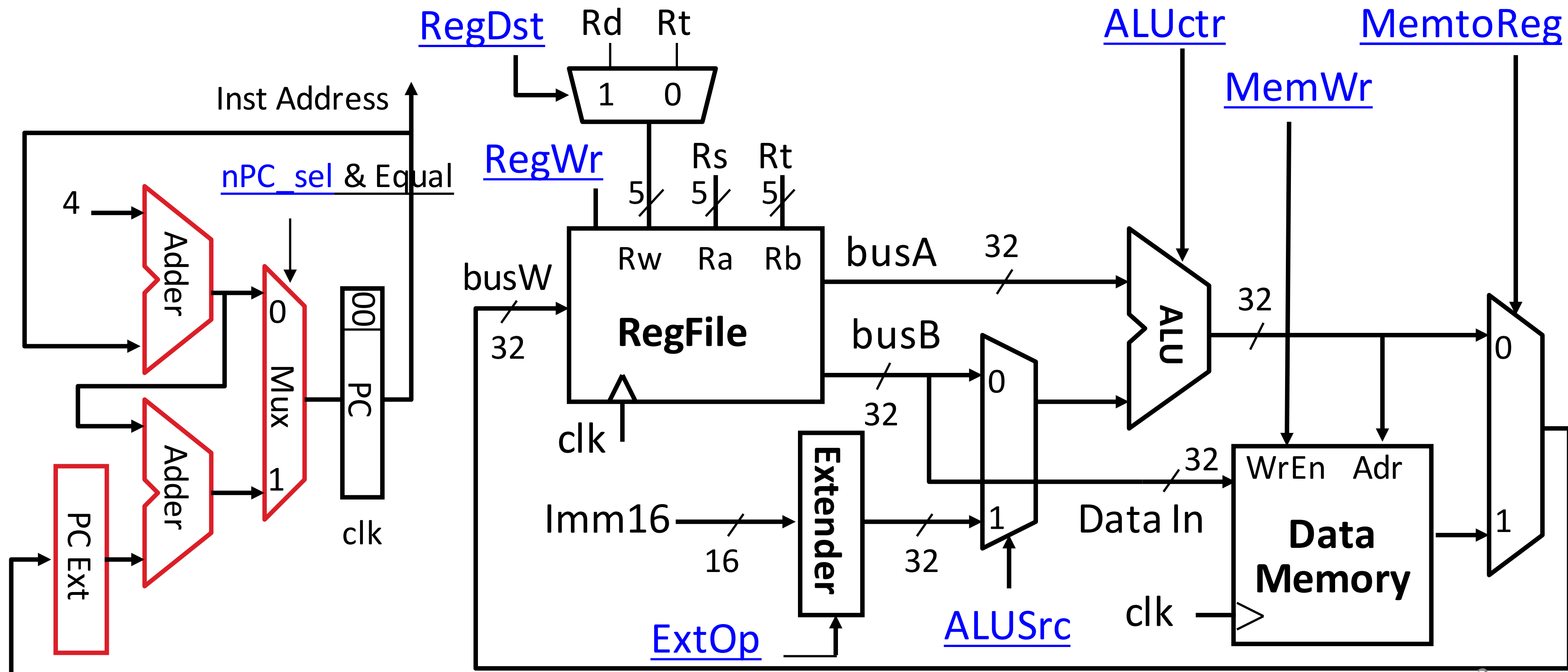
CS 61C: Great Ideas in Computer Architecture Control and Pipelining

Instructors:
Vladimir Stojanovic and Nicholas Weaver
http://inst.eecs.Berkeley.edu/~cs61c/sp16

Datapath Control Signals

- ExtOp: "zero", "sign"
- ALUsrc: 0 ⇒ regB; 1 ⇒ immed
- ALUctr: "ADD", "SUB", "OR"
- MemWr: 1 ⇒ write memory
- MemtoReg: 0 ⇒ ALU; 1 ⇒ Mem
- RegDst: 0 ⇒ "rt"; 1 ⇒ "rd"
- RegWr: 1 ⇒ write register



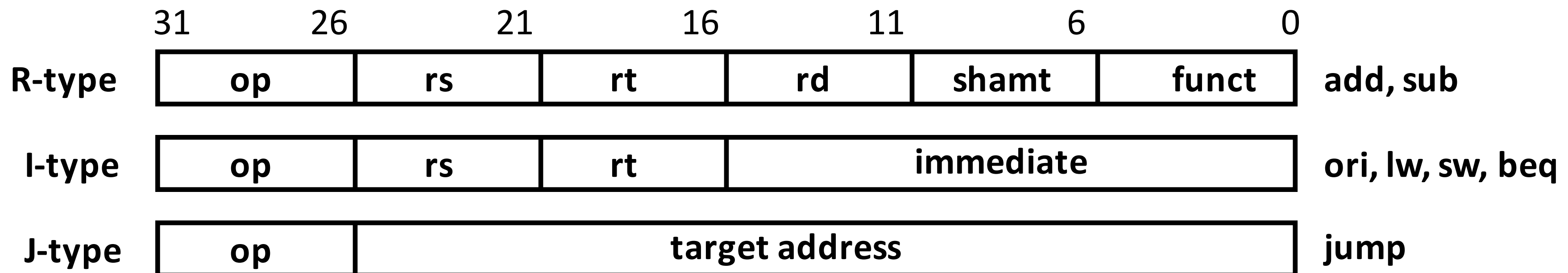
Summary of the Control Signals (1/2)

```
inst      Register Transfer

add       R[rd] ← R[rs] + R[rt]; PC ← PC + 4
          ALUsrc=RegB, ALUctr="ADD", RegDst=rd, RegWr, nPC_sel="+4"

sub       R[rd] ← R[rs] – R[rt]; PC ← PC + 4
          ALUsrc=RegB, ALUctr="SUB", RegDst=rd, RegWr, nPC_sel="+4"

ori       R[rt] ← R[rs] + zero_ext(Imm16); PC ← PC + 4
          ALUsrc=Im, Extop="Z", ALUctr="OR", RegDst=rt,RegWr, nPC_sel="+4"

lw        R[rt] ← MEM[ R[rs] + sign_ext(Imm16)]; PC ← PC + 4
          ALUsrc=Im, Extop="sn", ALUctr="ADD", MemtoReg, RegDst=rt, RegWr,
          nPC_sel = "+4"

sw        MEM[ R[rs] + sign_ext(Imm16)] ← R[rs]; PC ← PC + 4
          ALUsrc=Im, Extop="sn", ALUctr = "ADD", MemWr, nPC_sel = "+4"

beq       if (R[rs] == R[rt]) then PC ← PC + sign_ext(Imm16)] || 00
          else PC ← PC + 4
          nPC_sel = "br",  ALUctr = "SUB"
```

Summary of the Control Signals (2/2)

See Appendix A → func, op

func	10 0000	10 0010	We Don't Care :-)				
op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	add	**sub**	**ori**	**lw**	**sw**	**beq**	**jump**
RegDst	1	1	0	0	x	x	x
ALUSrc	0	0	1	1	1	0	x
MemtoReg	0	0	0	1	x	x	x
RegWrite	1	1	1	1	0	0	0
MemWrite	0	0	0	0	1	0	0
nPCsel	0	0	0	0	0	1	?
Jump	0	0	0	0	0	0	1
ExtOp	x	x	0	1	1	x	x
ALUctr<2:0>	Add	Subtract	Or	Add	Add	Subtract	x



Boolean Expressions 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   (assume ALUctr is 00 ADD, 01 SUB, 10 OR)
ALUctr[1] = or
```

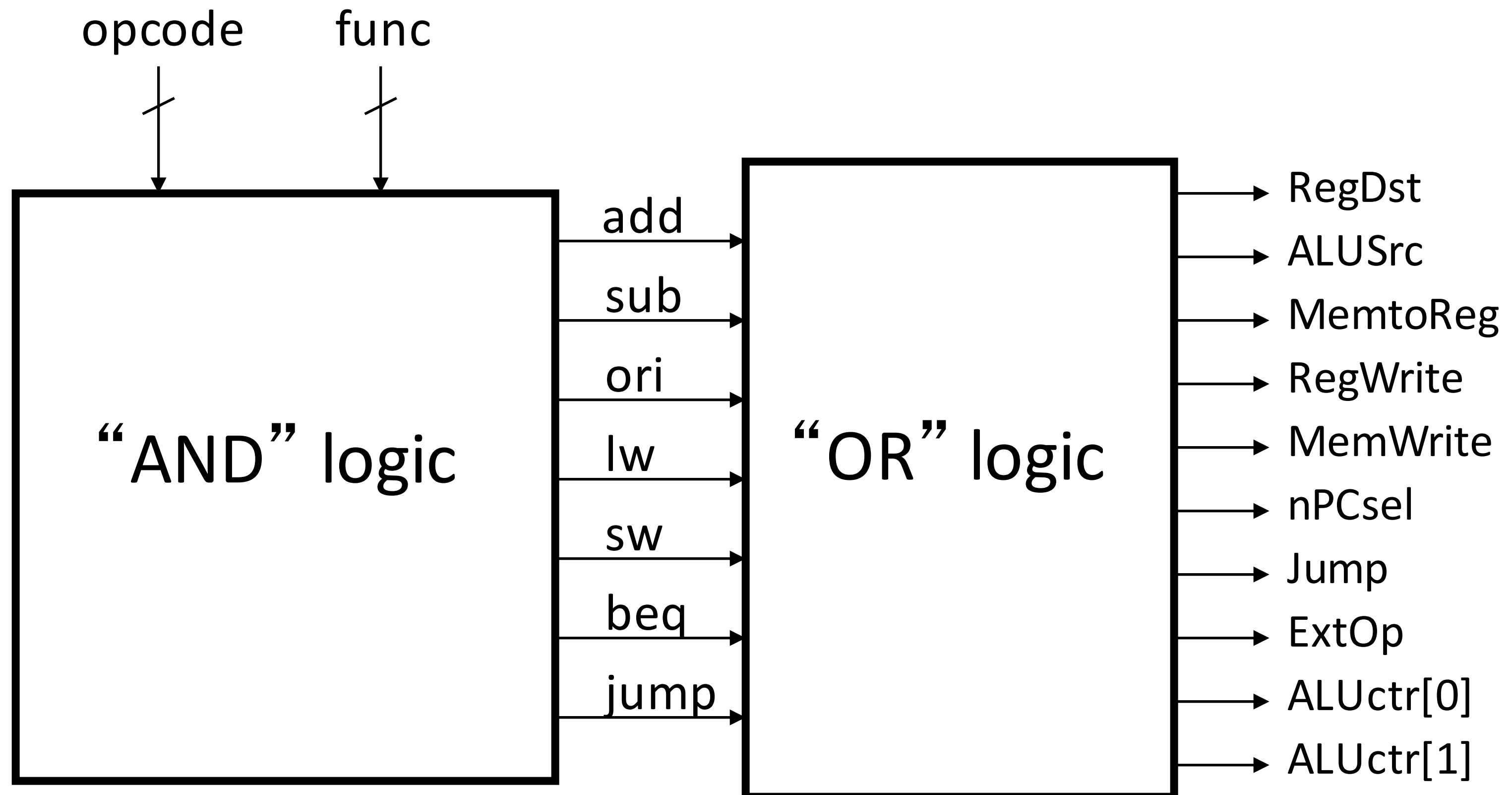
Where:

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

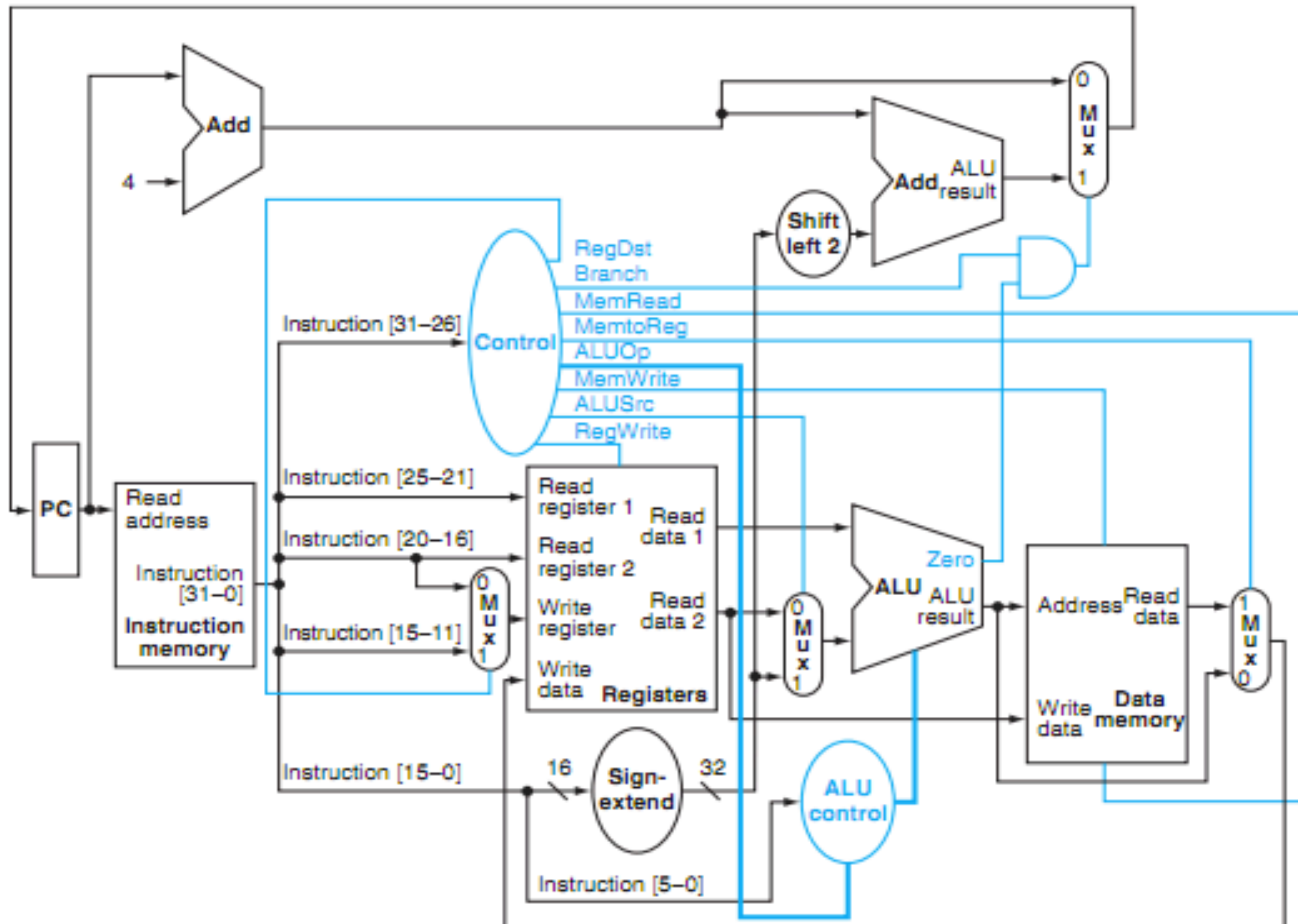
How do we implement this in gates?

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 $\bullet\ func_5 \bullet \sim func_4 \bullet \sim func_3 \bullet \sim func_2 \bullet func_1 \bullet \sim func_0$

Controller Implementation

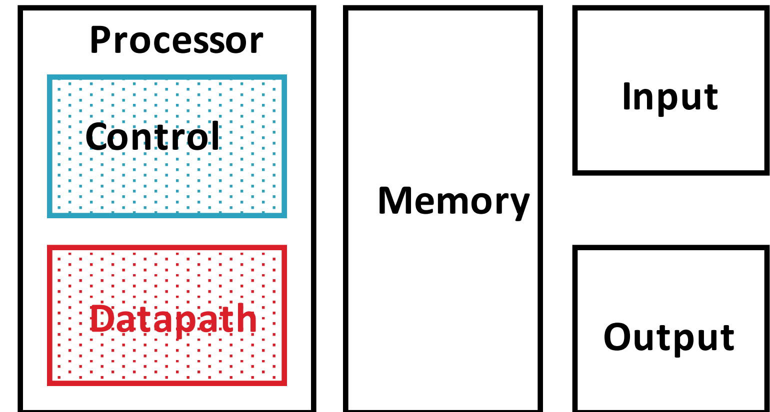


P&H Figure 4.17



Summary: 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 period 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

- Clock rate (cycles/second = Hz) = 1/Period (seconds/cycle)

Single Cycle Performance

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

Levels of Representation/Interpretation

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

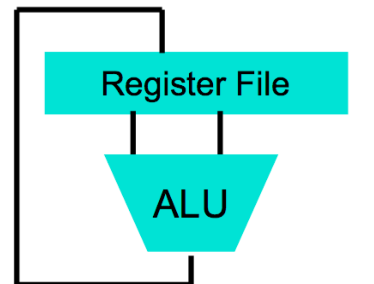
Logic Circuit Description (Circuit Schematic Diagrams)

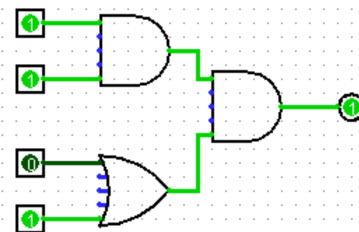
```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

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

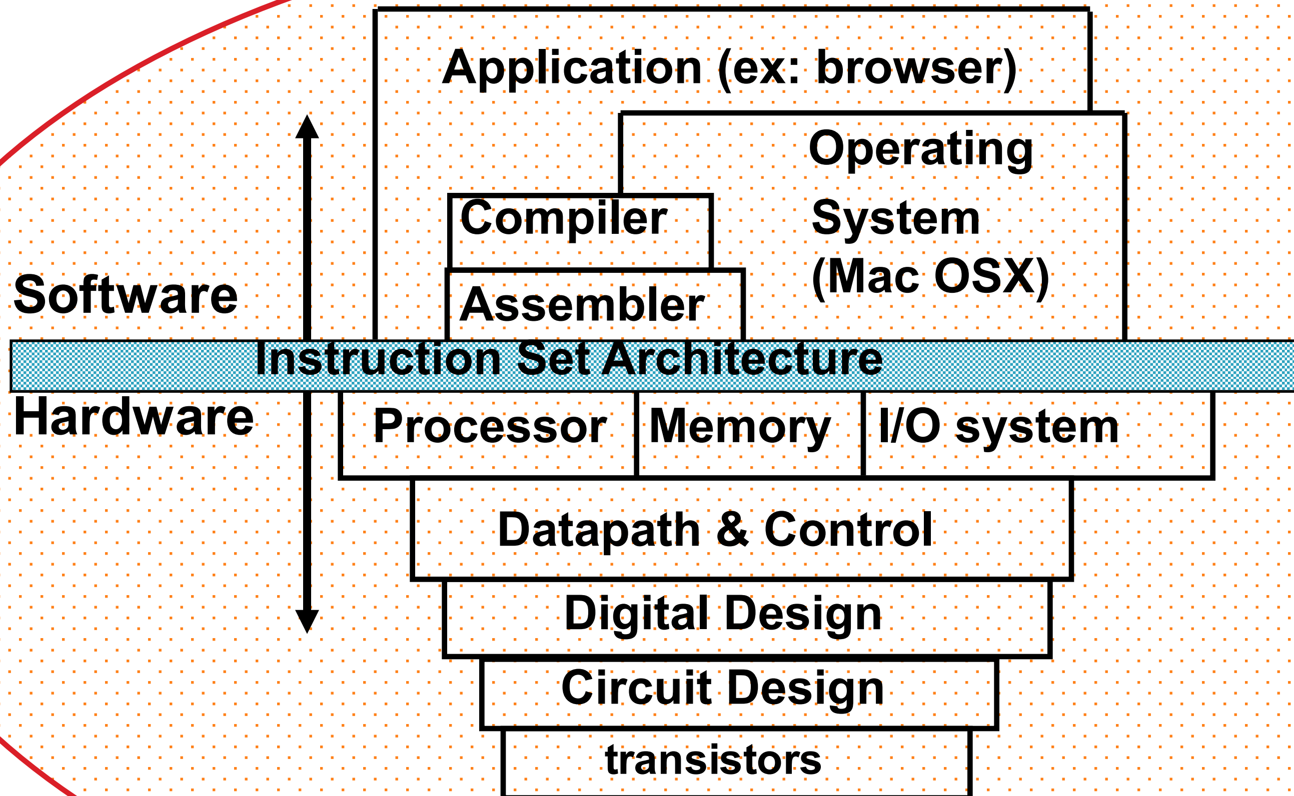
Anything can be represented as a *number*, i.e., data or instructions

```
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
```



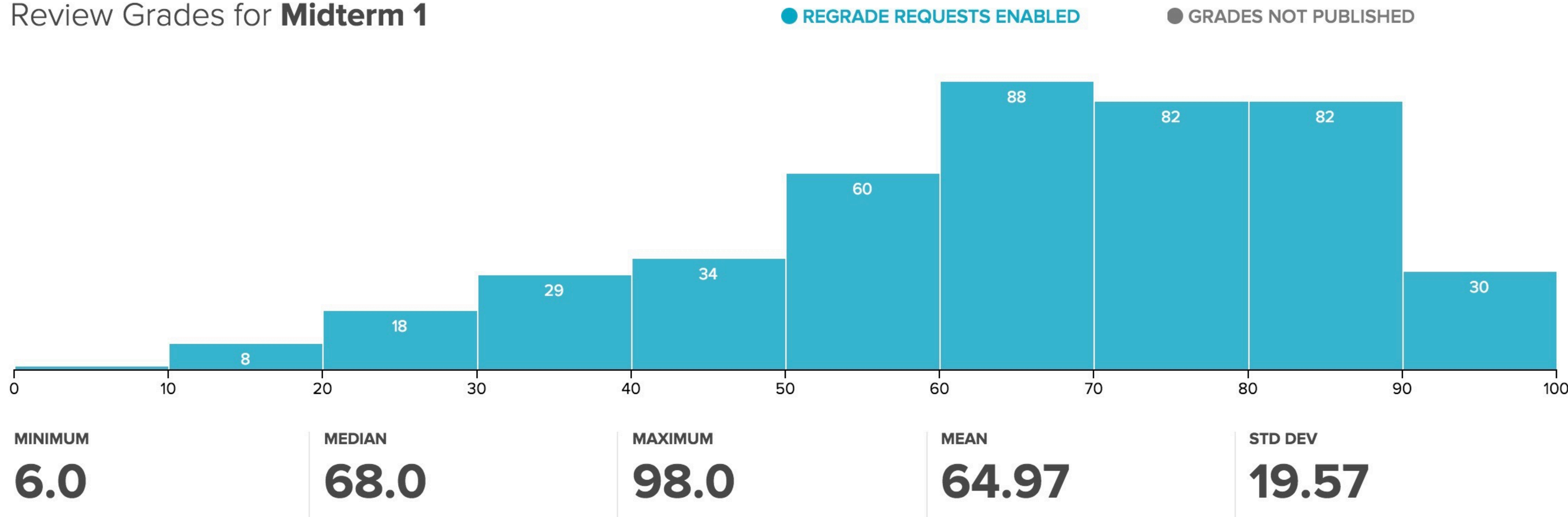


No More Magic!



CS61A
CS61B
CS61C ✔
CS61C ✔
CS61C ✔
CS61C
CS61C ✔
EE40
Phys 7B

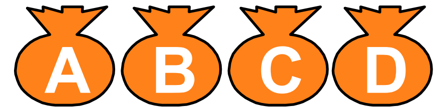
Administrivia



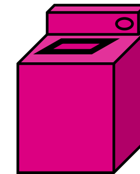
- Project 2-2 due 3/8 @ 23:59:59 (Tue)
- Guerrilla Sessions: MIPS CPU
  - Wed 3/09 3 - 5 PM @ 241 Cory
  - Sat 3/12 1 - 3 PM @ 651 @ 611 Soda

Gotta Do Laundry

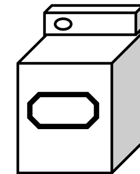
- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, fold, and put away



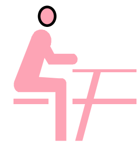
  - Washer takes 30 minutes



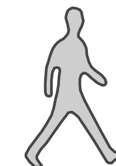
  - Dryer takes 30 minutes



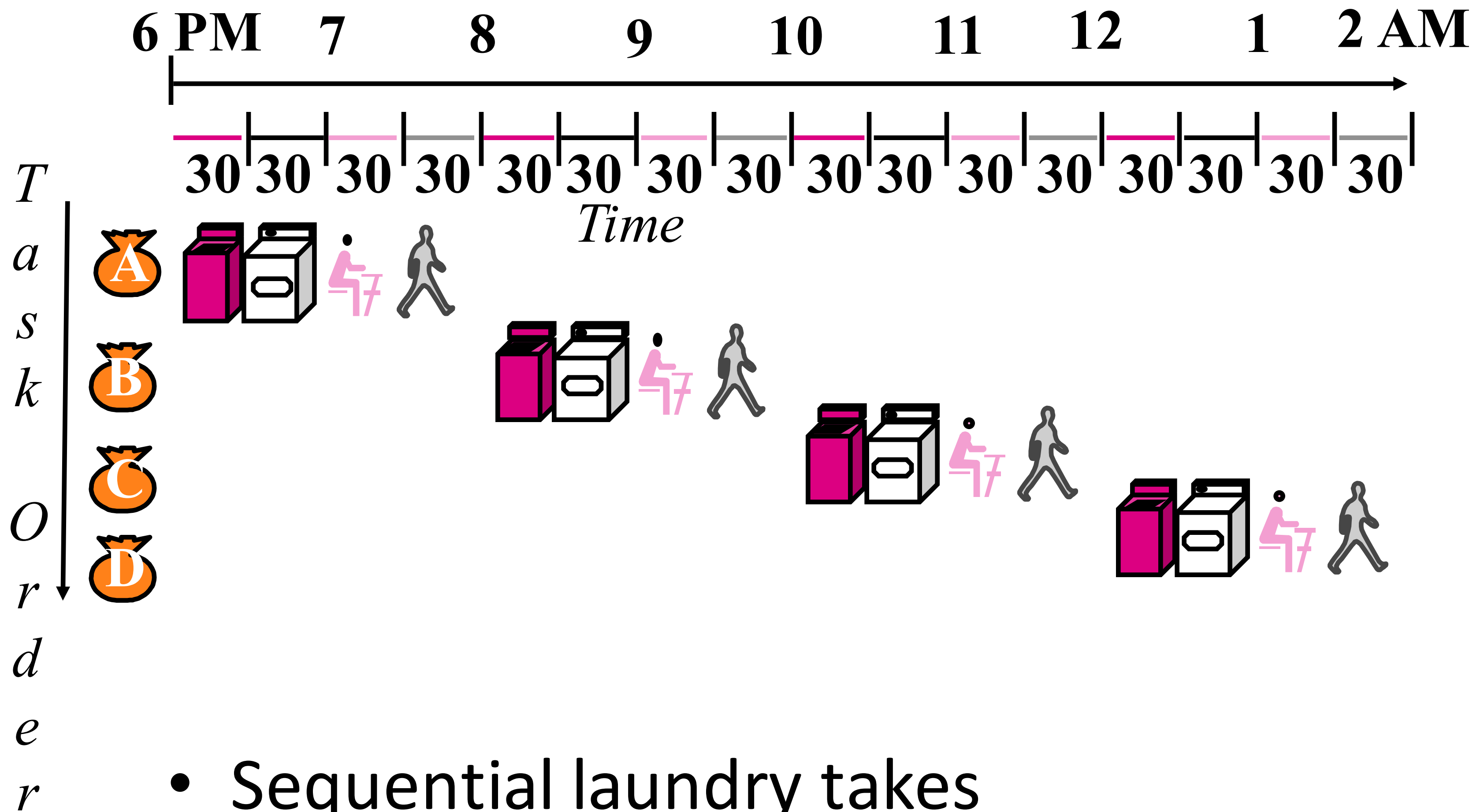
  - “Folder” takes 30 minutes



  - “Stasher” takes 30 minutes to put clothes into drawers

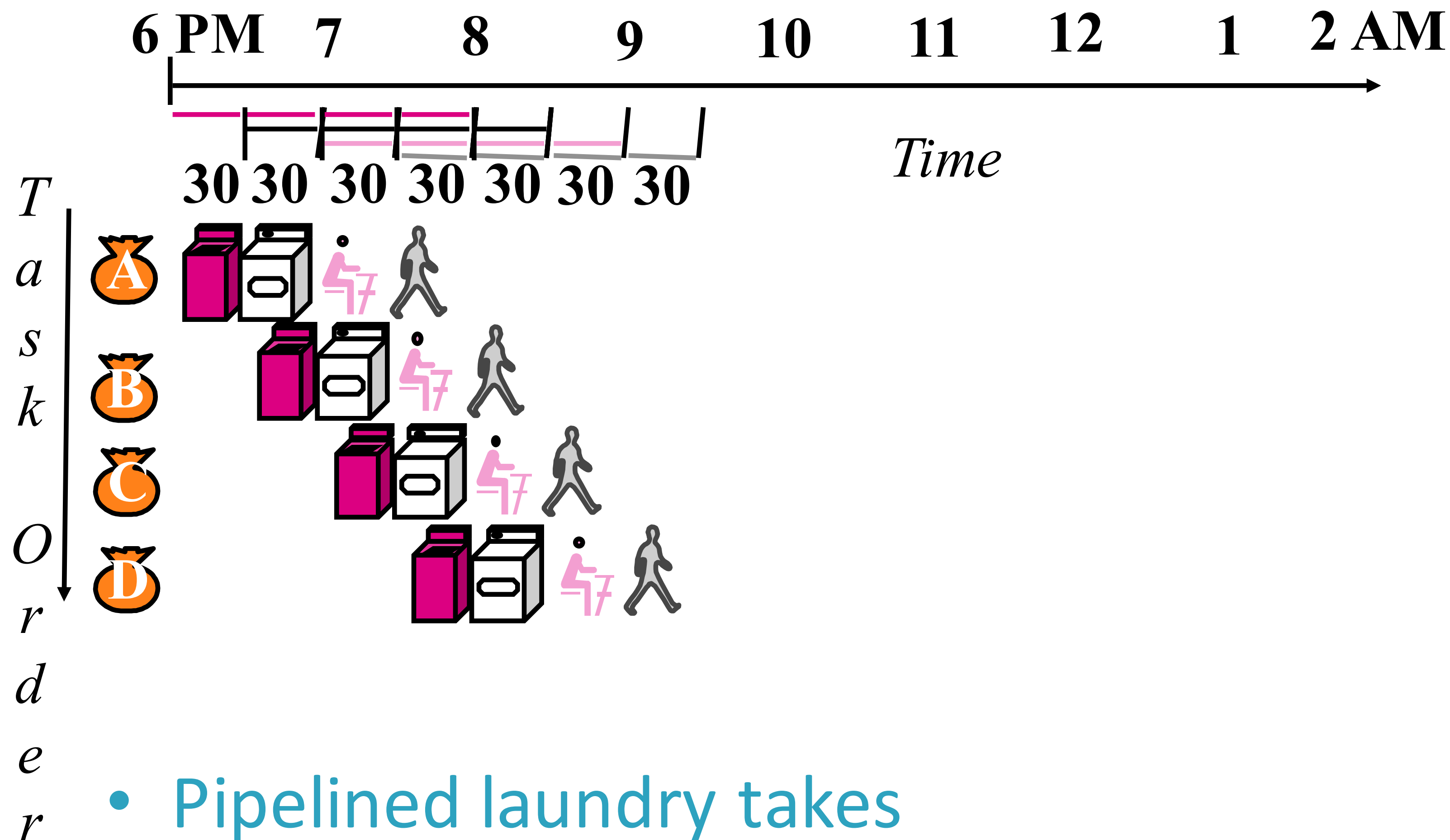


Sequential Laundry



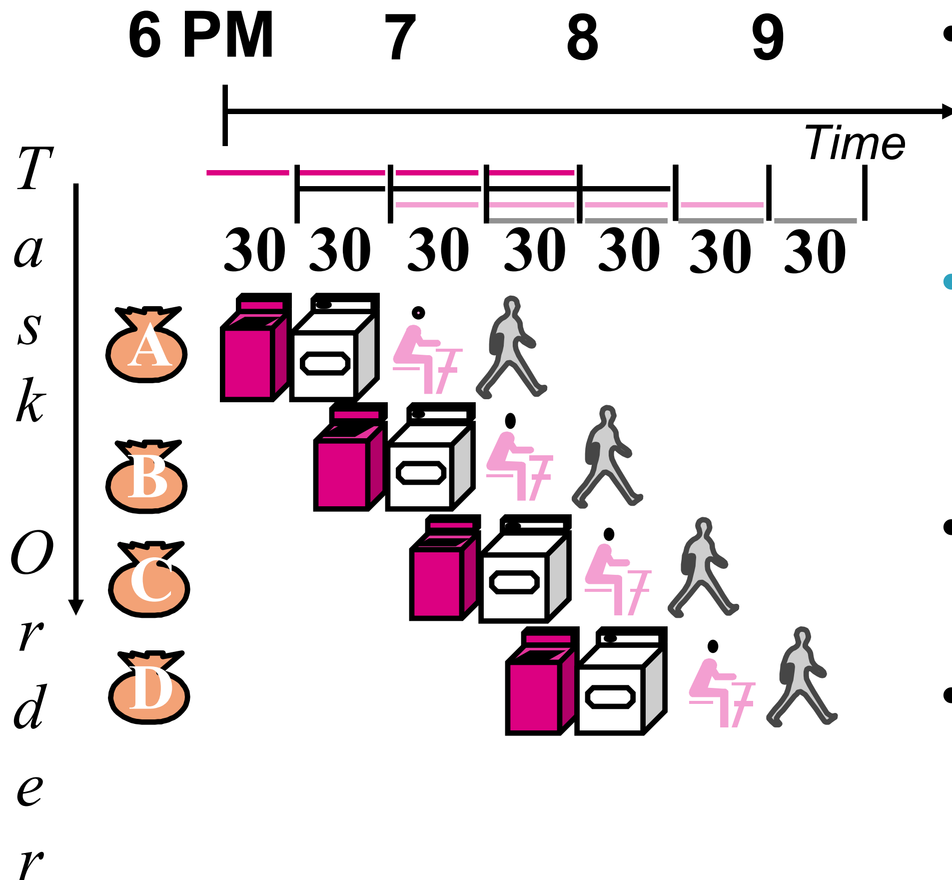
- Sequential laundry takes 8 hours for 4 loads

Pipelined Laundry



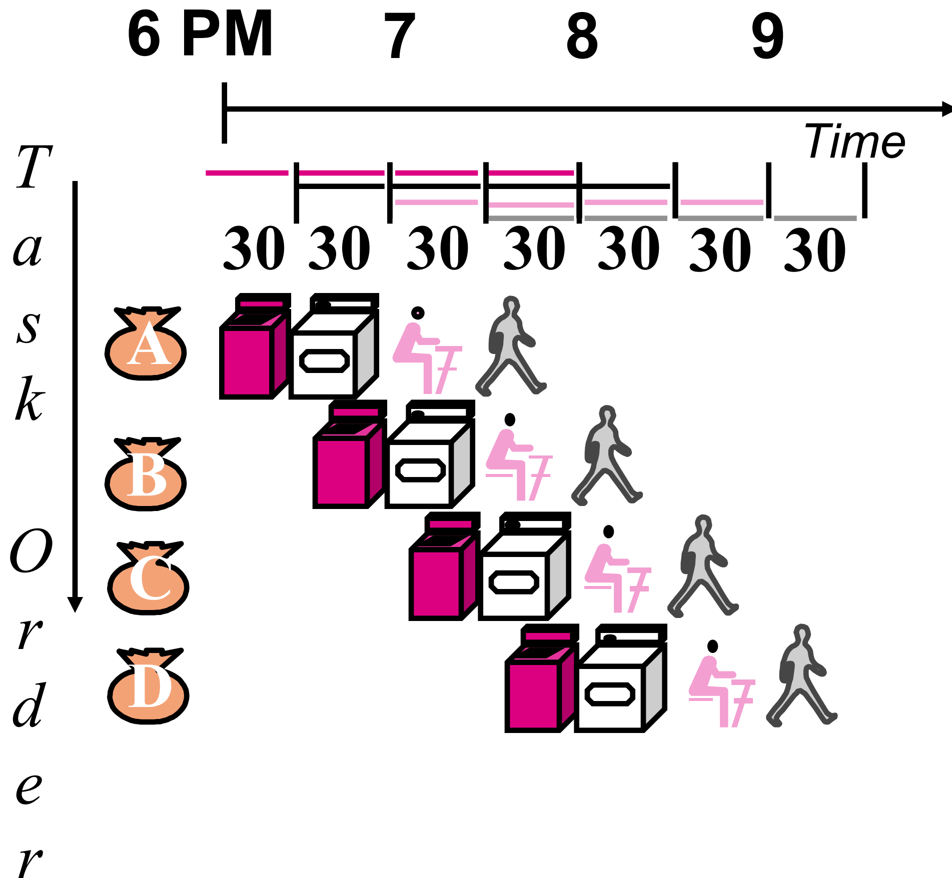
- Pipelined laundry takes 3.5 hours for 4 loads!

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 (8/3.5) v. 4x (8/2) in this example

Pipelining Lessons (2/2)

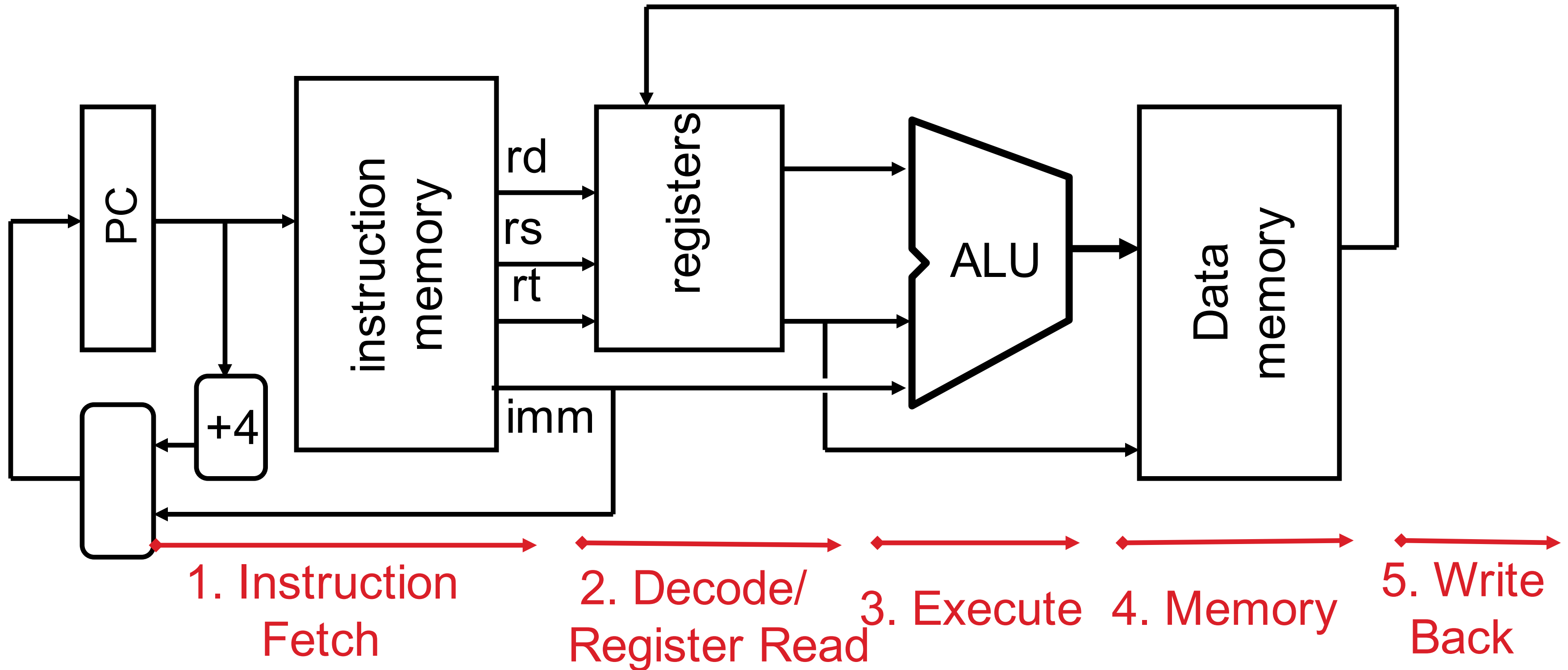


- Suppose new Washer takes 20 minutes, new Stasher takes 20 minutes. How much faster is pipeline?
- Pipeline rate limited by slowest pipeline stage
- Unbalanced lengths of pipe stages reduces speedup

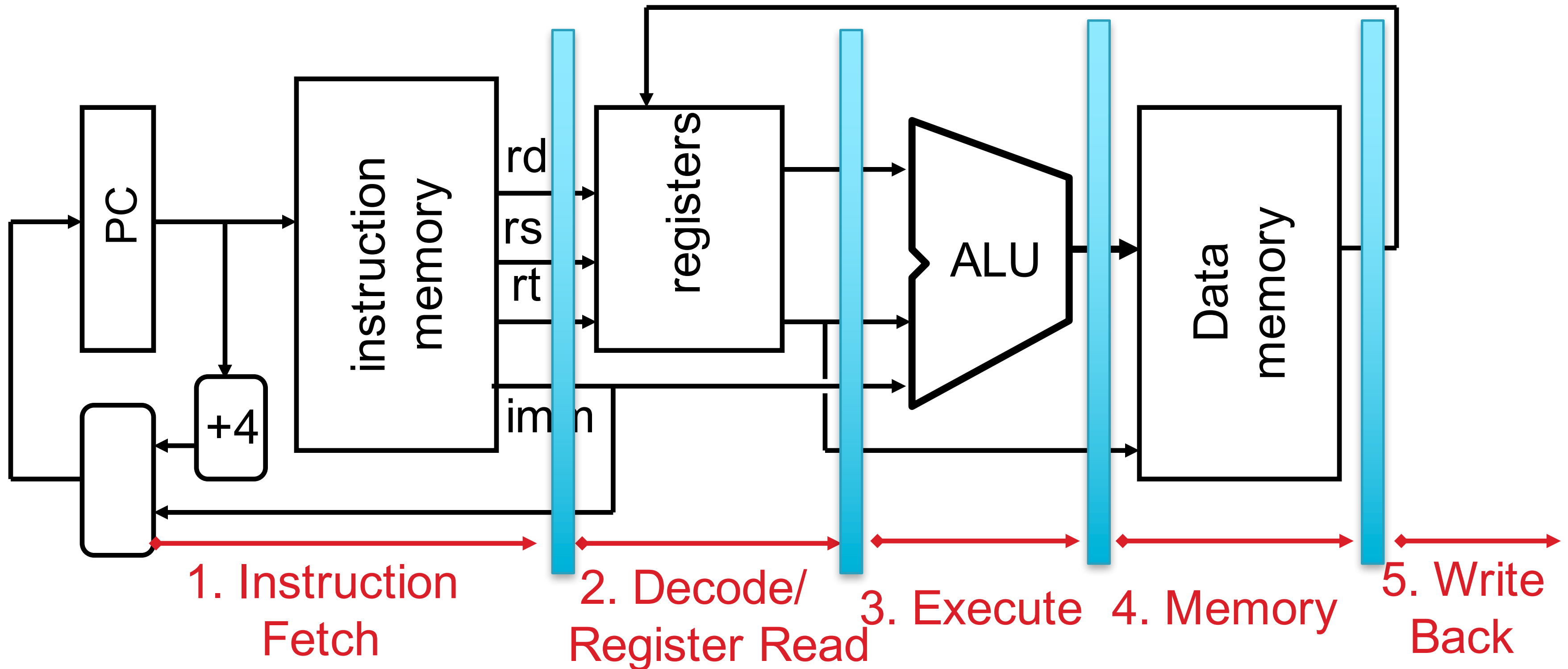
Execution Steps in MIPS Datapath

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

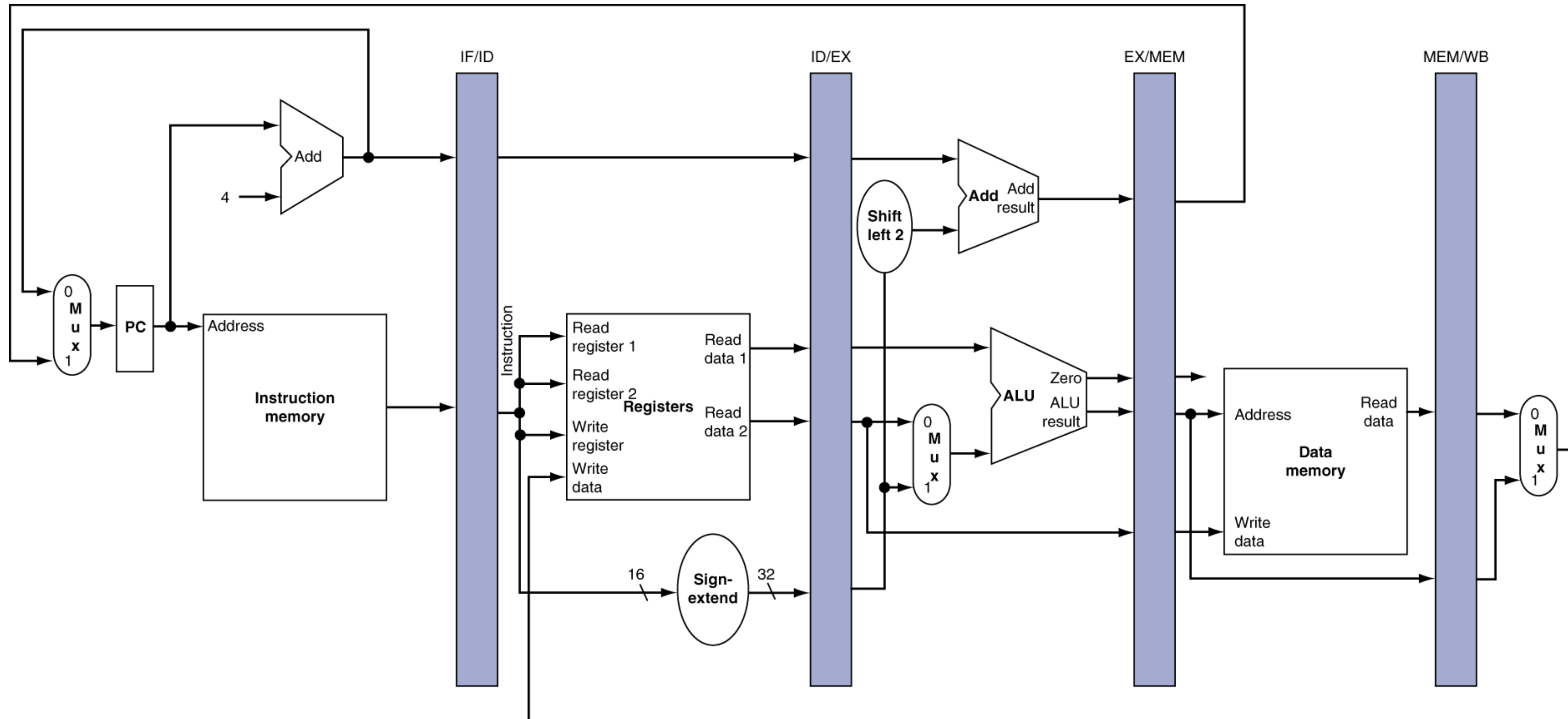


Pipeline registers

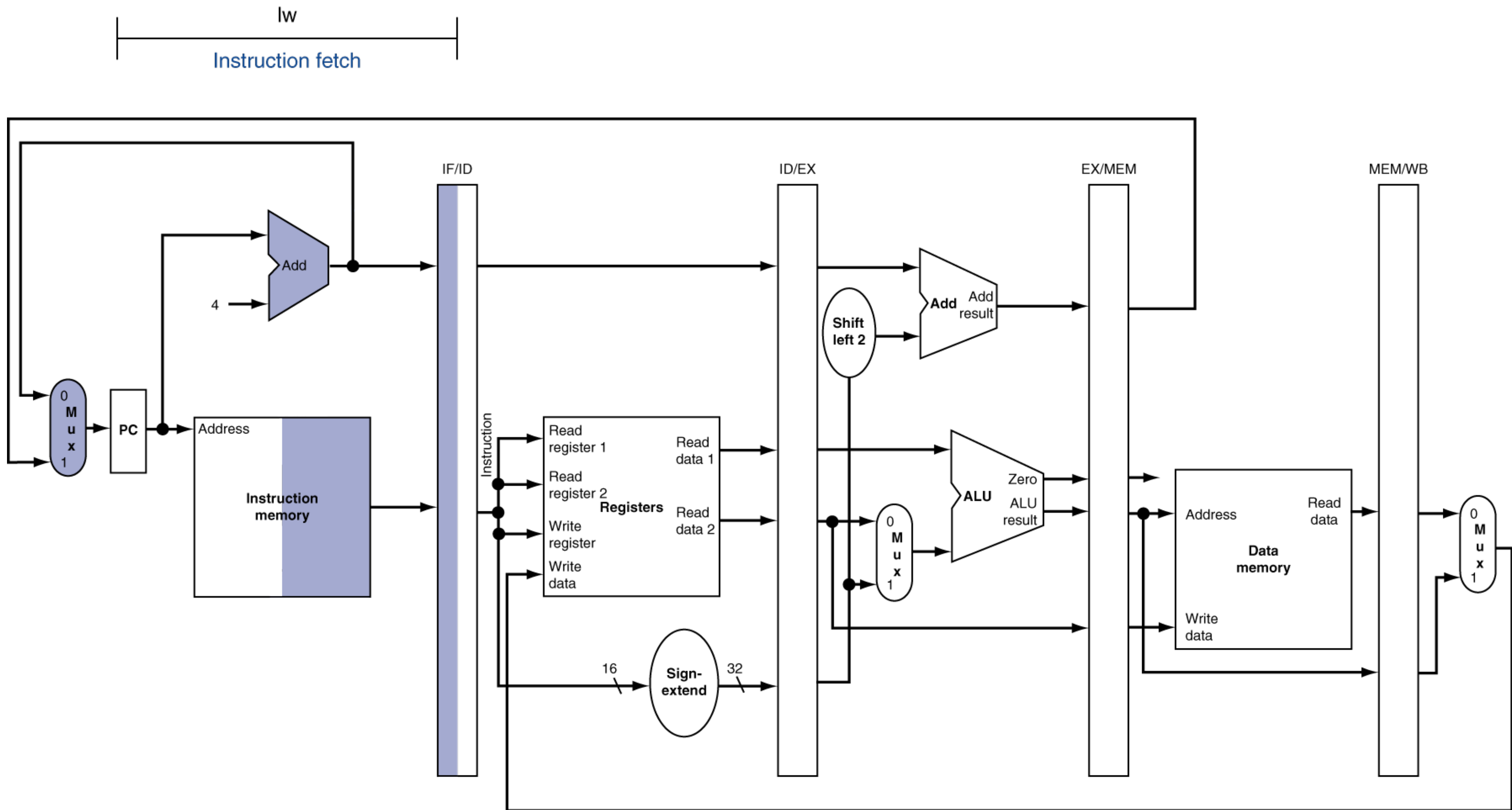


- Need registers between stages
  - To hold information produced in previous cycle

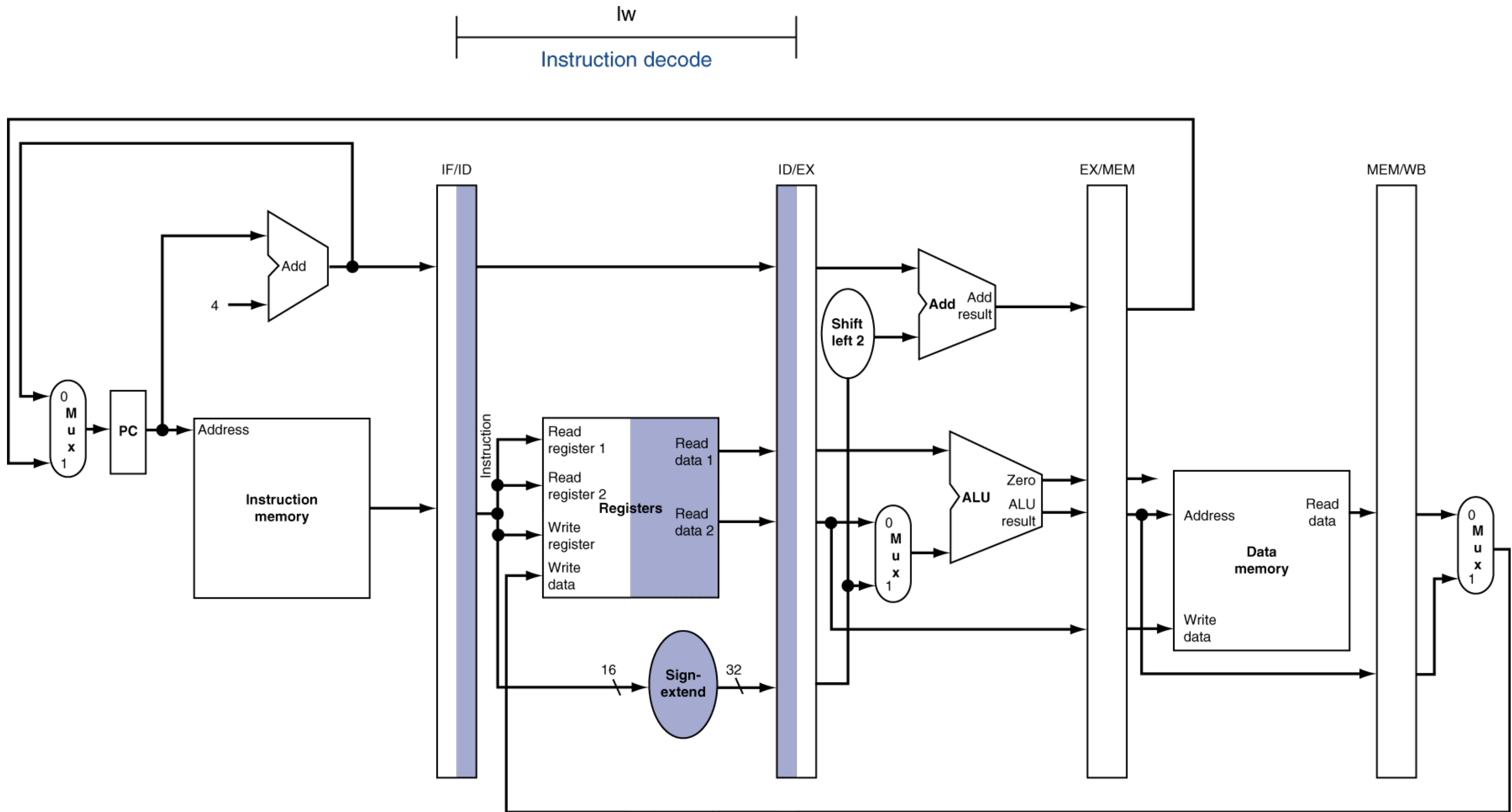
More Detailed Pipeline



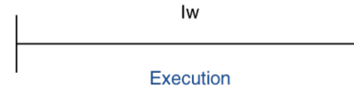
IF for Load, Store, …



ID for Load, Store, …

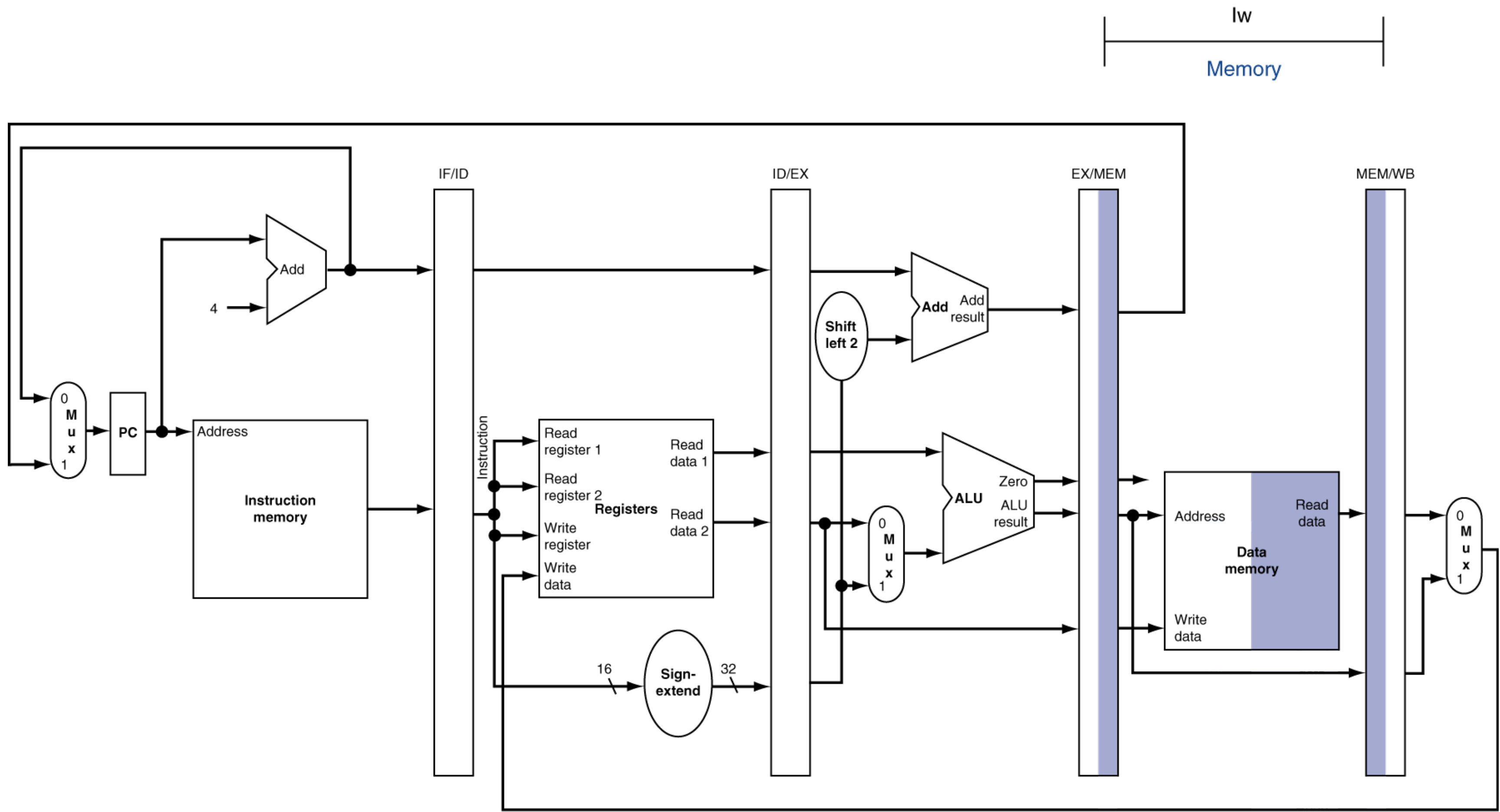


EX for Load

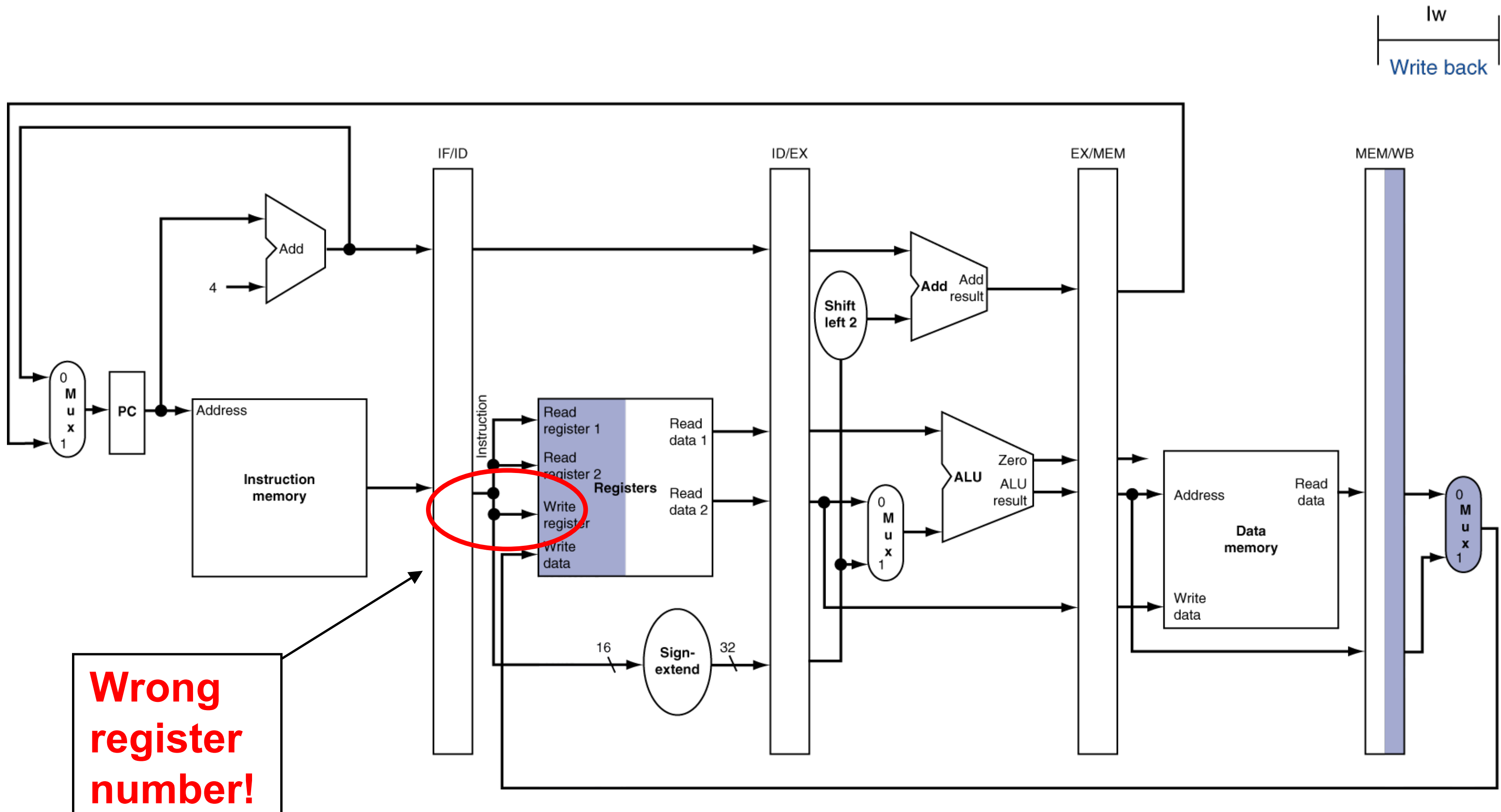


IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0 M u x 1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign-extend
32
Shift left 2
Add
Add result
0 M u x 1
ALU
Zero
ALU result
Address
Data memory
Write data
Read data
0 M u x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

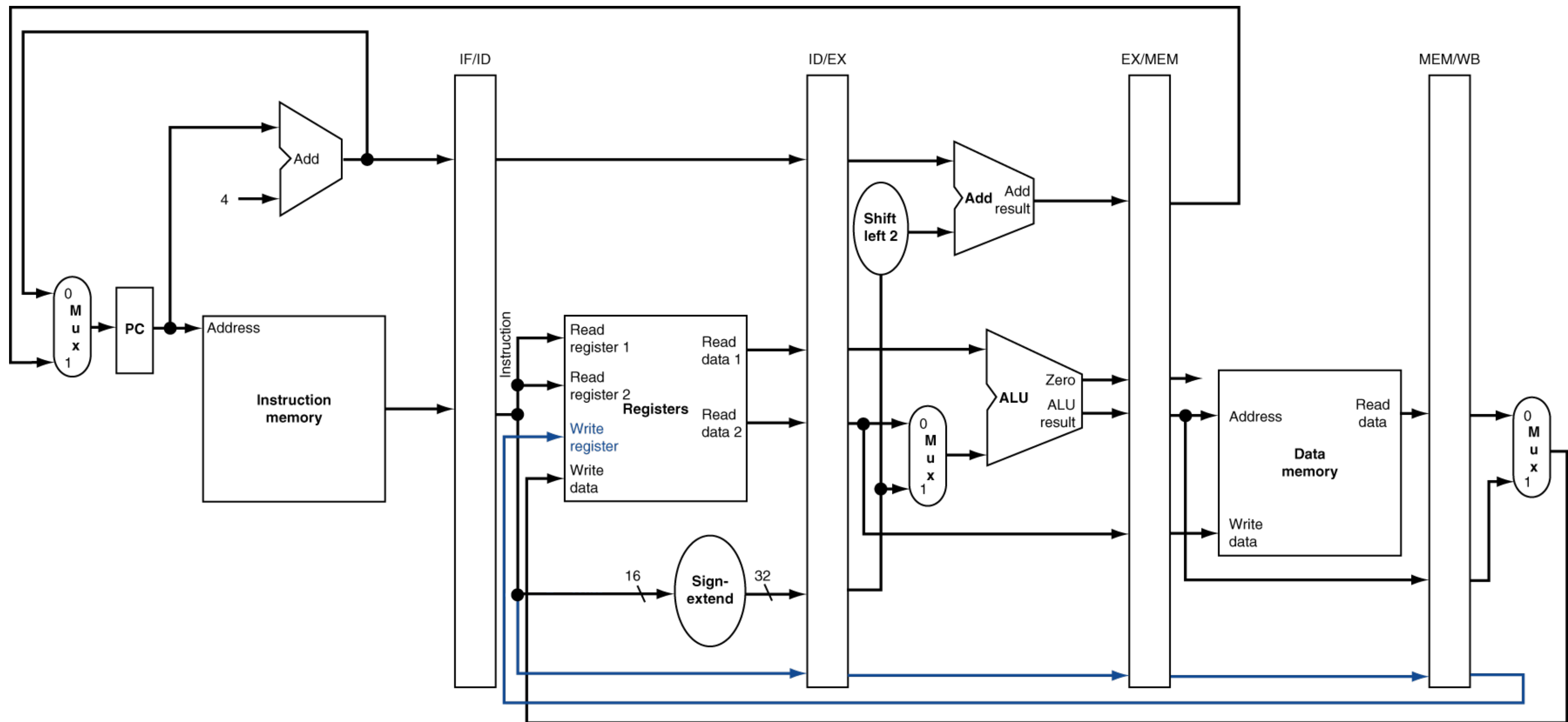
MEM for Load



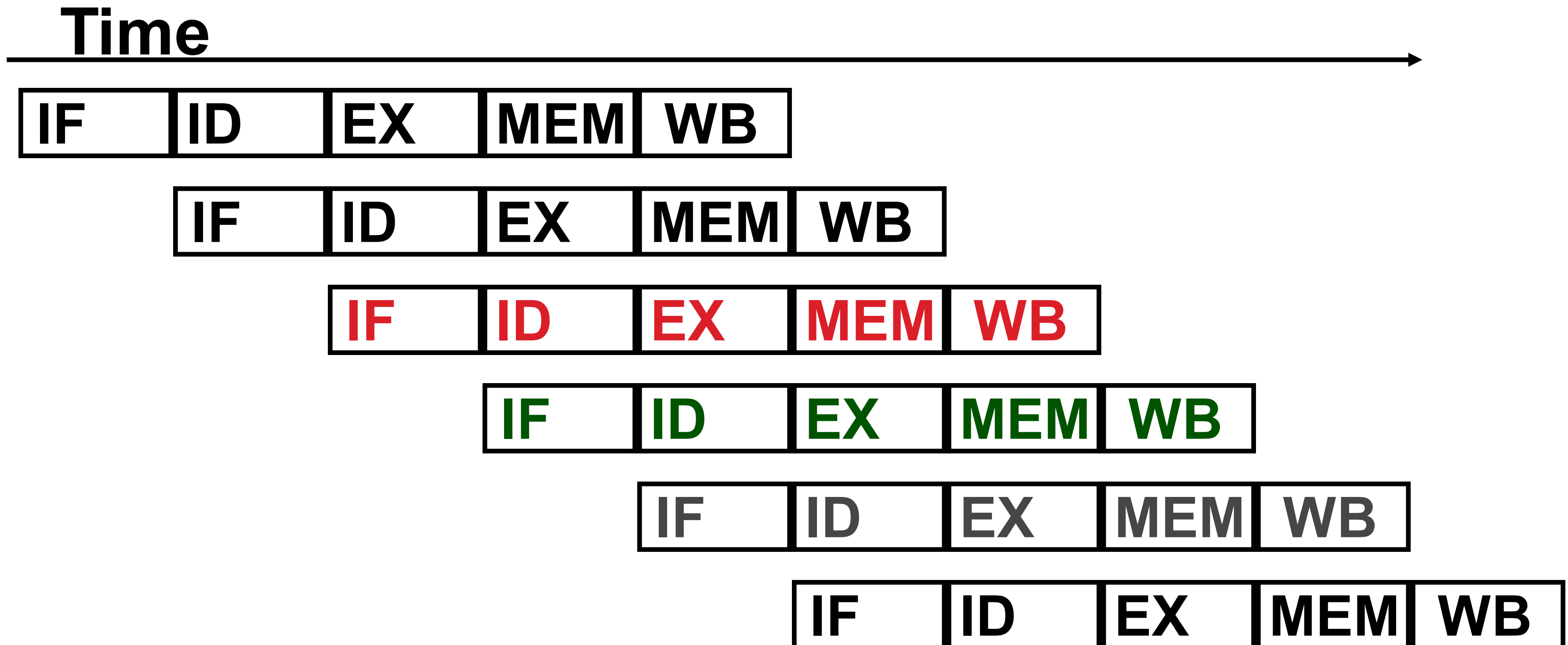
WB for Load – Oops!



Corrected Datapath for Load



Pipelined Execution Representation



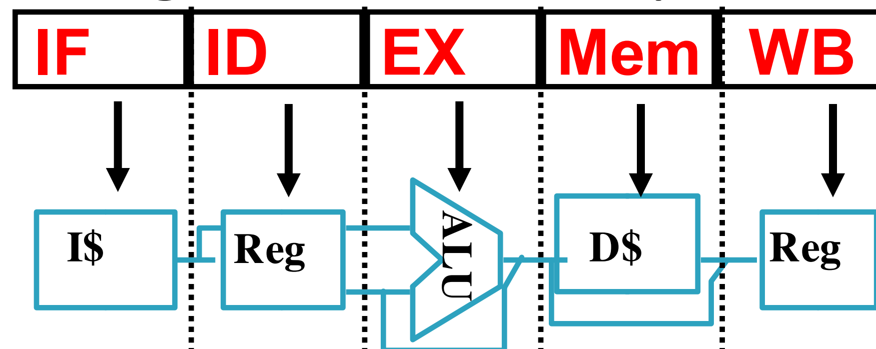
- Every instruction must take same number of steps, so some stages will idle
  - e.g. MEM stage for any arithmetic instruction

Graphical Pipeline Diagrams

PC
instruction memory
rd
rs
rt
imm
Register File
ALU
Data memory
+4
MUX

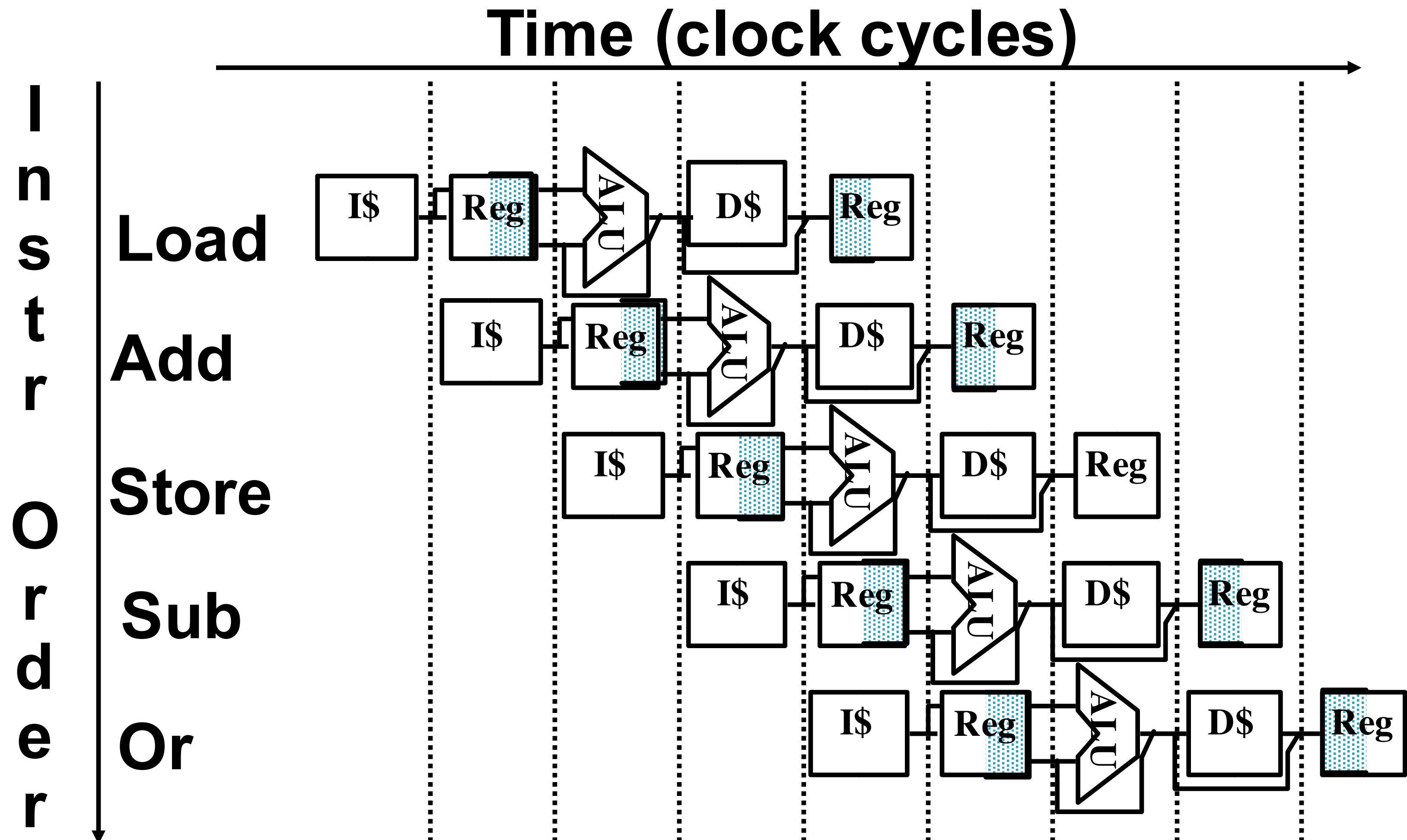
1. Instruction Fetch
2. Decode/ Register Read
3. Execute
4. Memory
5. Write Back

- Use datapath figure below to represent pipeline:



Graphical Pipeline Representation

- RegFile: left half is write, right half is read



Pipelining Performance (1/3)

- Use T_c (“time between completion of instructions”) to measure speedup
  - $T_{c,\text{pipelined}} \geq \dfrac{T_{c,\text{single-cycle}}}{\text{Number of stages}}$
  - Equality only achieved if stages are *balanced* (i.e. take the same amount of time)
- If not balanced, speedup is reduced
- Speedup due to increased *throughput*
  - *Latency* for each instruction does not decrease

Pipelining Performance (2/3)

- Assume time for stages is
  - 100ps for register read or write
  - 200ps for other stages

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

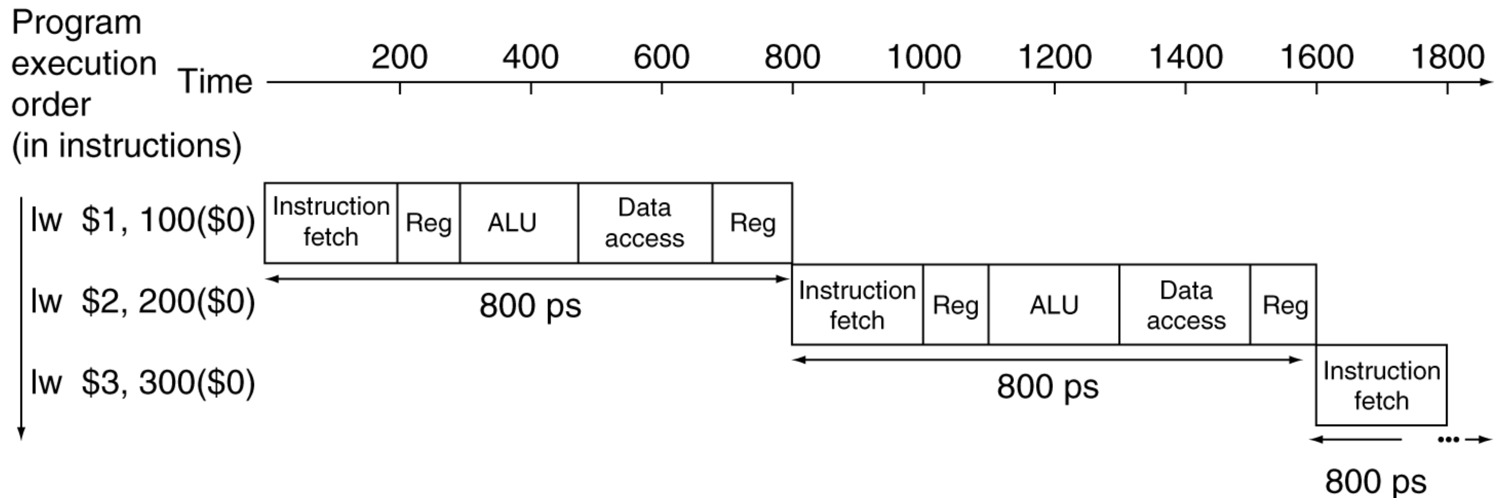
- What is pipelined clock rate?
  - Compare pipelined datapath with single-cycle datapath

Pipelining Performance (3/3)

Single-cycle

T_c = 800 ps

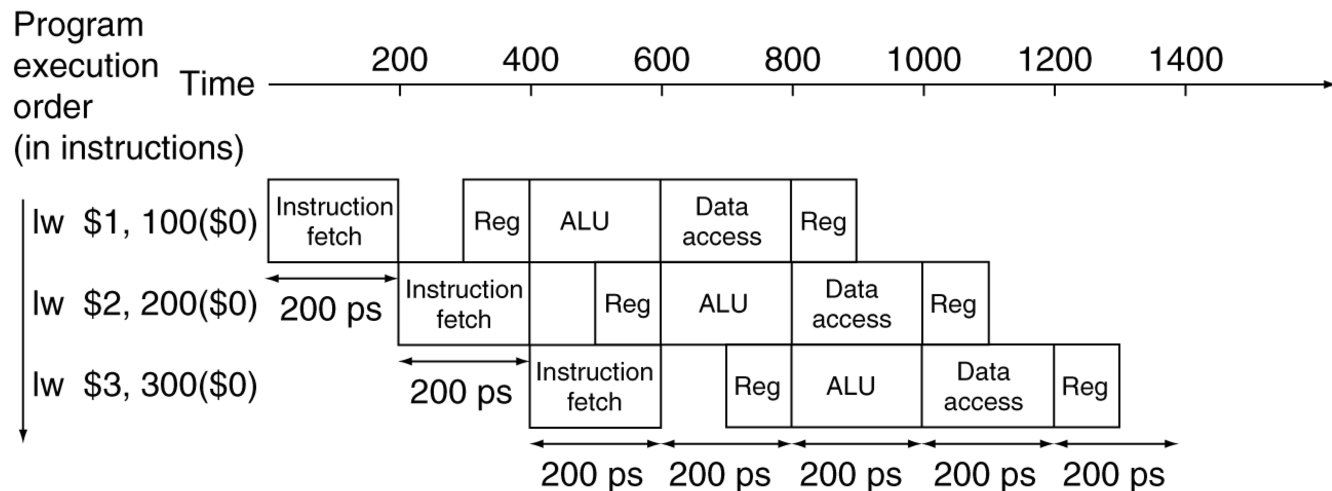
f = 1.25GHz



Pipelined

T_c = 200 ps

f = 5GHz



Clicker/Peer Instruction

Logic in some stages takes 200ps and in some 100ps. Clk-Q delay is 30ps and setup-time is 20ps. What is the maximum clock frequency at which a pipelined design can operate?

- A: 10GHz
- B: 5GHz
- C: 6.7GHz
- D: 4.35GHz
- E: 4GHz