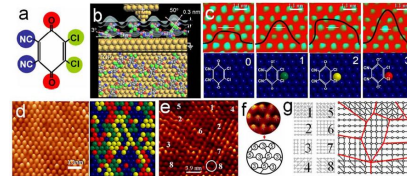


## 61C in the News0

### Is organic computing finally here?

SPOTLIGHT | NOVEMBER 2, 2011

Japanese scientists have made organic molecules perform parallel computations like neurons in the human brain. They created this promising new approach with a ring-like molecule called 2,3-dichloro-5,6-dicyano-p-benzoquinone, or DDQ.



Today, computer chips can process data at 10 trillion (10<sup>13</sup>) bits per second. But, even though neurons in the human brain fire at a rate of 100 times per second, the brain still outperforms the best computers at various tasks. **The main reason being that calculations done by computer chips happen in isolated pipelines one at a time.**

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

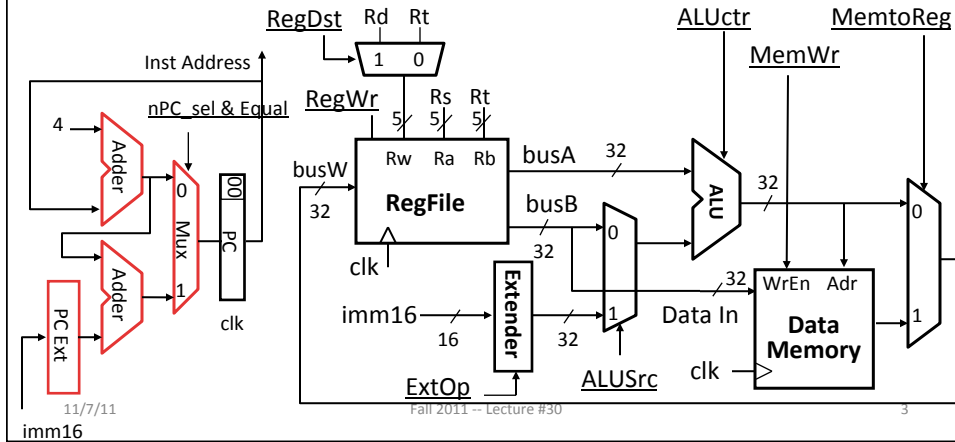
Instructors:  
Mike Franklin

Dan Garcia

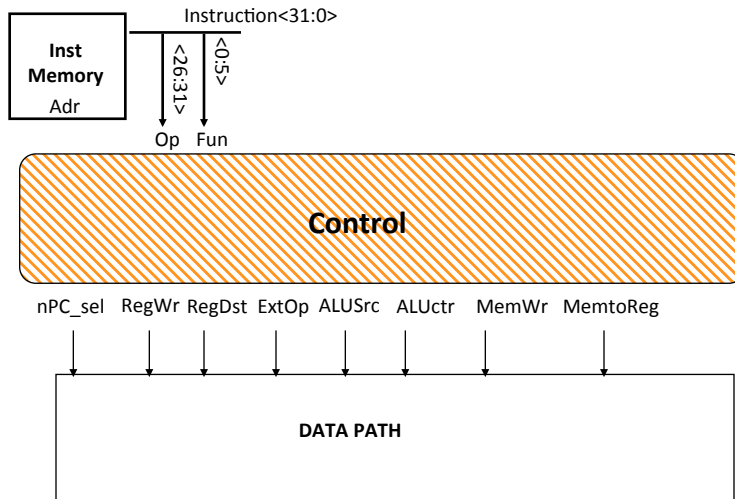
<http://inst.eecs.Berkeley.edu/~cs61c/fa11>

## Datapath Control Signals

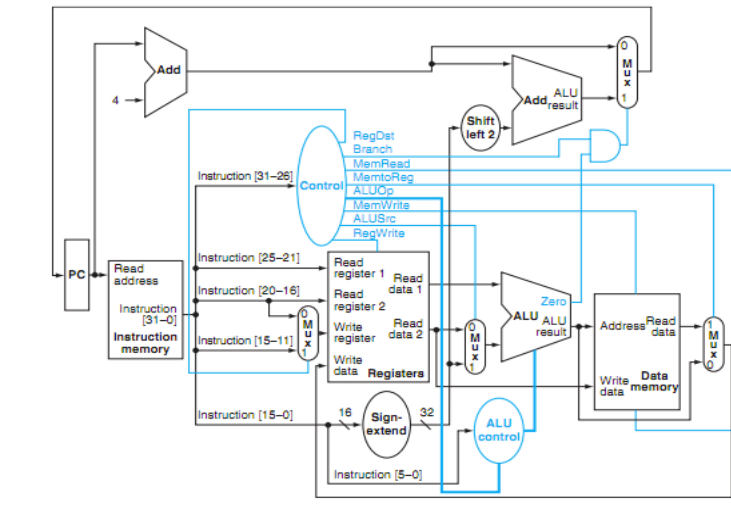
- ExtOp: “zero”, “sign”
- ALUSrc: 0 ⇒ regB;  
1 ⇒ immed
- ALUctr: “ADD”, “SUB”, “OR”
- MemWr: 1 ⇒ write memory
- MemtoReg: 0 ⇒ ALU; 1 ⇒ Mem
- nPC\_sel: 0 ⇒ “+4”; 1 ⇒ “br”
- RegDst: 0 ⇒ “rt”; 1 ⇒ “rd”
- RegWr: 1 ⇒ write register



## Where Do Control Signals Come From?



## P&H Figure 4.17



11/7/11

Fall 2011 -- Lecture #30

5

## 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"
  
```

11/7/11

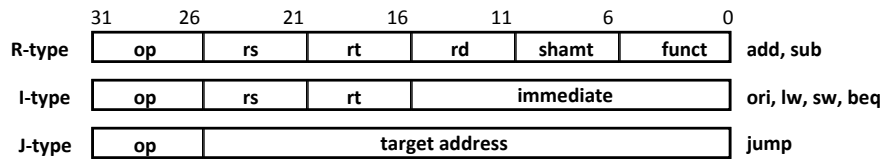
Fall 2011 -- Lecture #30

6

## Summary of the Control Signals (2/2)

See Appendix A → **func**  
 → **op**

	10 0000	10 0010	We Don't Care :-)				
	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

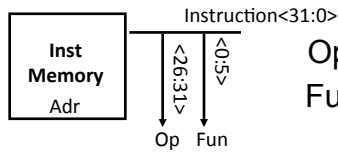


11/7/11

Fall 2011 -- Lecture #30

7

## Boolean Exprs for Controller



Op 0-5 are really Instruction bits 26-31  
 Func 0-5 are really Instruction bits 0-5

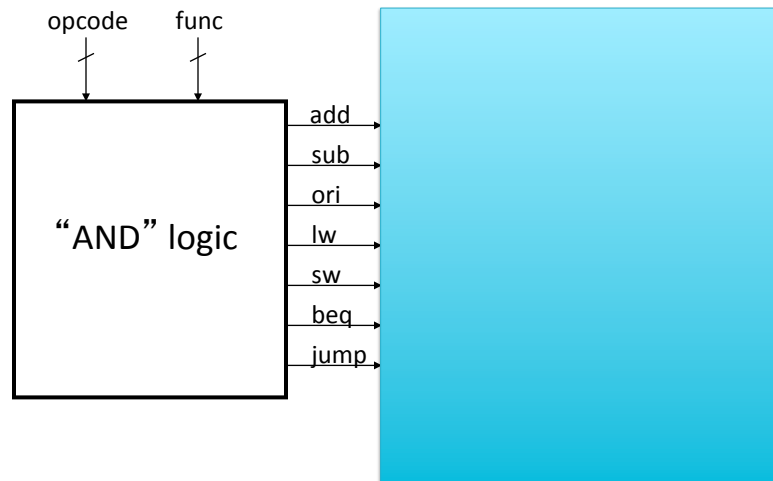
$$\begin{aligned}
 \mathbf{rtype} &= \sim op_5 \cdot \sim op_4 \cdot \sim op_3 \cdot \sim op_2 \cdot \sim op_1 \cdot \sim op_0, \\
 ori &= \sim op_5 \cdot \sim op_4 \cdot op_3 \cdot op_2 \cdot \sim op_1 \cdot op_0 \\
 lw &= op_5 \cdot \sim op_4 \cdot \sim op_3 \cdot \sim op_2 \cdot op_1 \cdot op_0 \\
 sw &= op_5 \cdot \sim op_4 \cdot op_3 \cdot \sim op_2 \cdot op_1 \cdot op_0 \\
 beq &= \sim op_5 \cdot \sim op_4 \cdot \sim op_3 \cdot op_2 \cdot \sim op_1 \cdot \sim op_0 \\
 jump &= \sim op_5 \cdot \sim op_4 \cdot \sim op_3 \cdot \sim op_2 \cdot op_1 \cdot \sim op_0 \\
 add &= \mathbf{rtype} \cdot func_5 \cdot \sim func_4 \cdot \sim func_3 \cdot \sim func_2 \cdot \sim func_1 \cdot \sim func_0 \\
 sub &= \mathbf{rtype} \cdot func_5 \cdot \sim func_4 \cdot \sim func_3 \cdot \sim func_2 \cdot func_1 \cdot \sim func_0
 \end{aligned}$$

11/7/11

How do we implement this in gates?

8

## Controller Implementation



11/7/11

Fall 2011 -- Lecture #30

9

## 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)

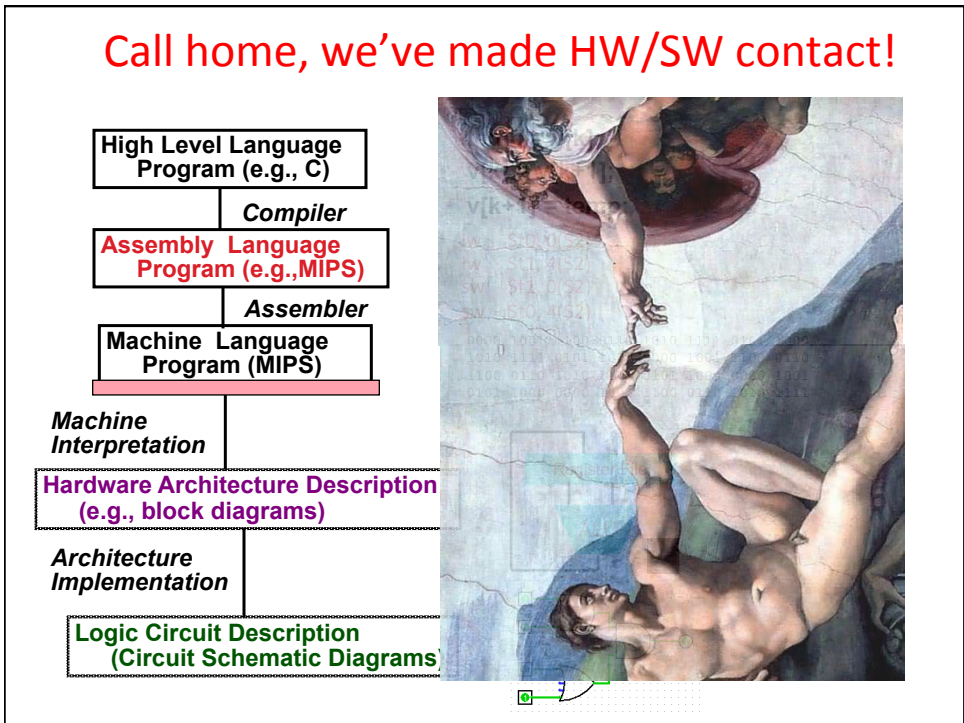
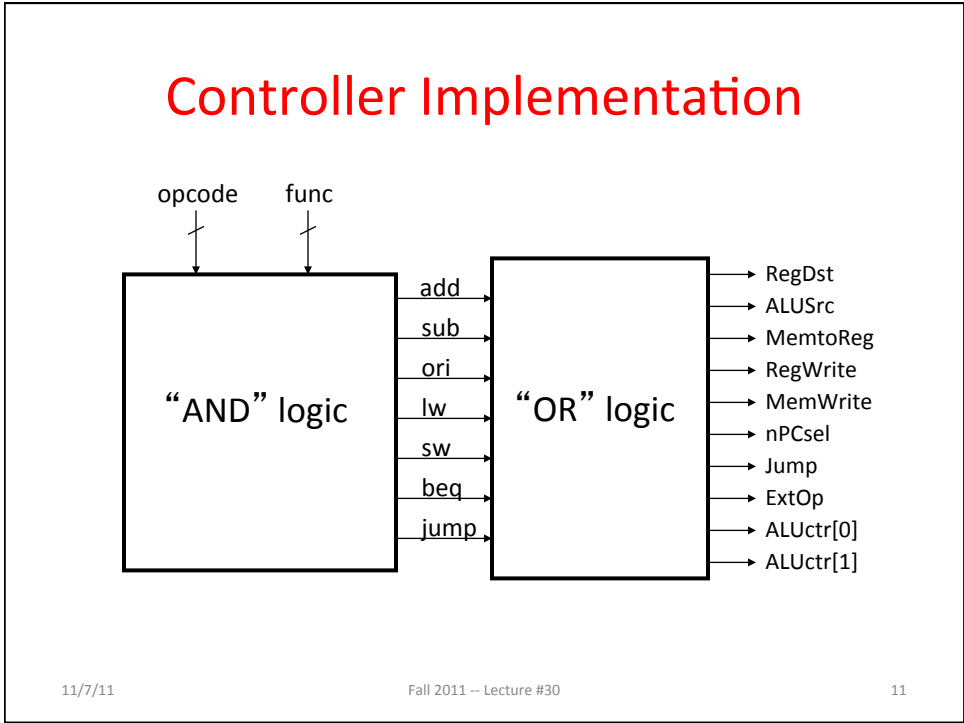
```

How do we implement this in gates?

11/7/11

Fall 2011 -- Lecture #30

10



## Administrivia

- Due to time constraints – we can only allow the use of a maximum of 2 slip days on Project 4.
- Thus, while we always encourage you to get your work done on time, if you still have 3 slip days left, you may want to consider using one prior to project 4

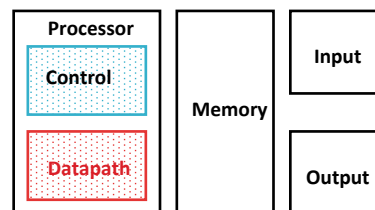
11/7/11

Fall 2011 -- Lecture #30

13

## 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



11/7/11

Fall 2011 -- Lecture #30

14

## 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

11/7/11 -- Lecture #30

15

## 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

11/7/11 -- Lecture #30

16

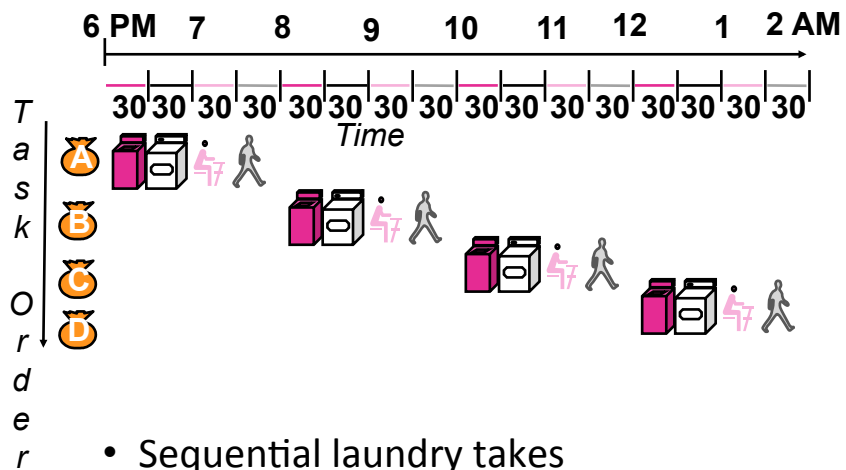


## Gotta Do Laundry

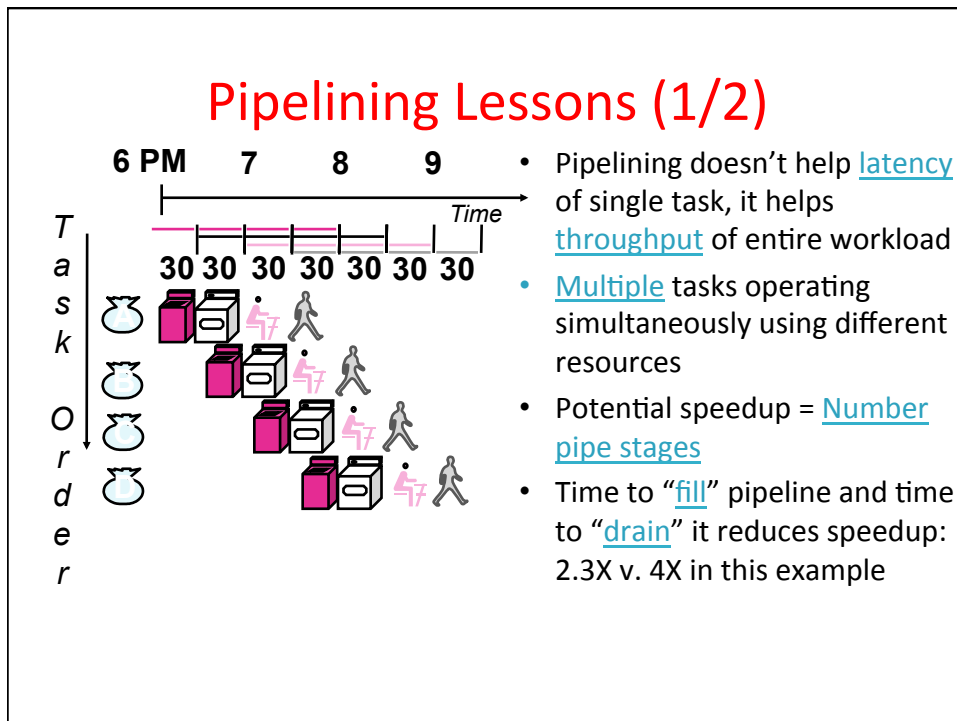
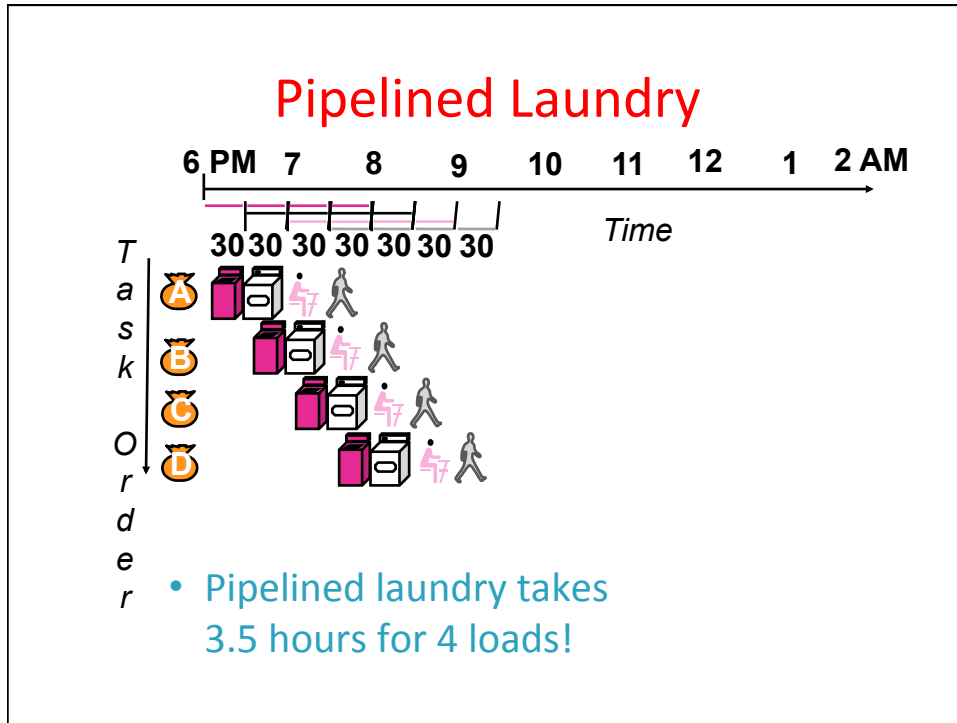
- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, fold, and put away
  - Washer takes 30 minutes 
  - Dryer takes 30 minutes 
  - “Folder” takes 30 minutes 
  - “Stasher” takes 30 minutes to put clothes into drawers 



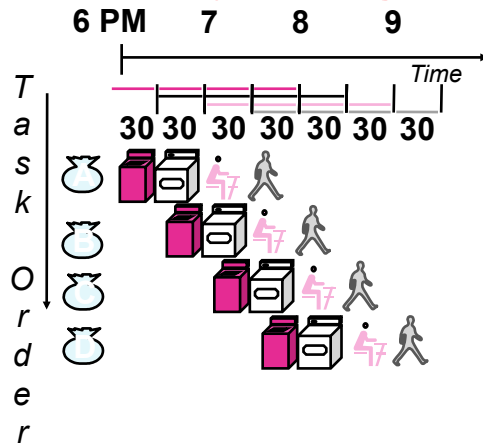
## Sequential Laundry



- Sequential laundry takes 8 hours for 4 loads



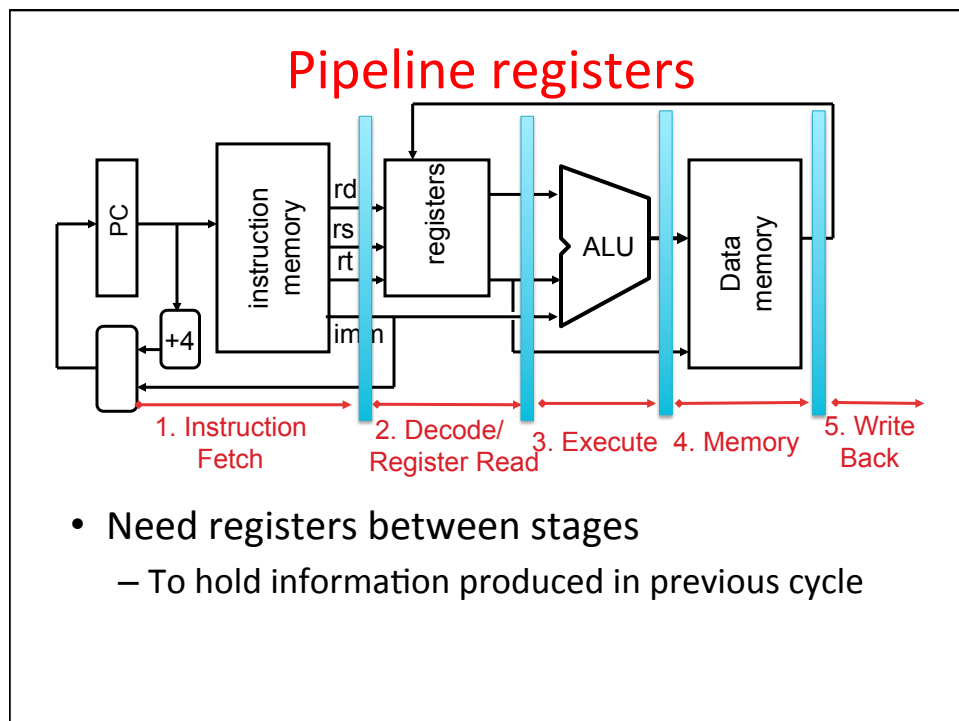
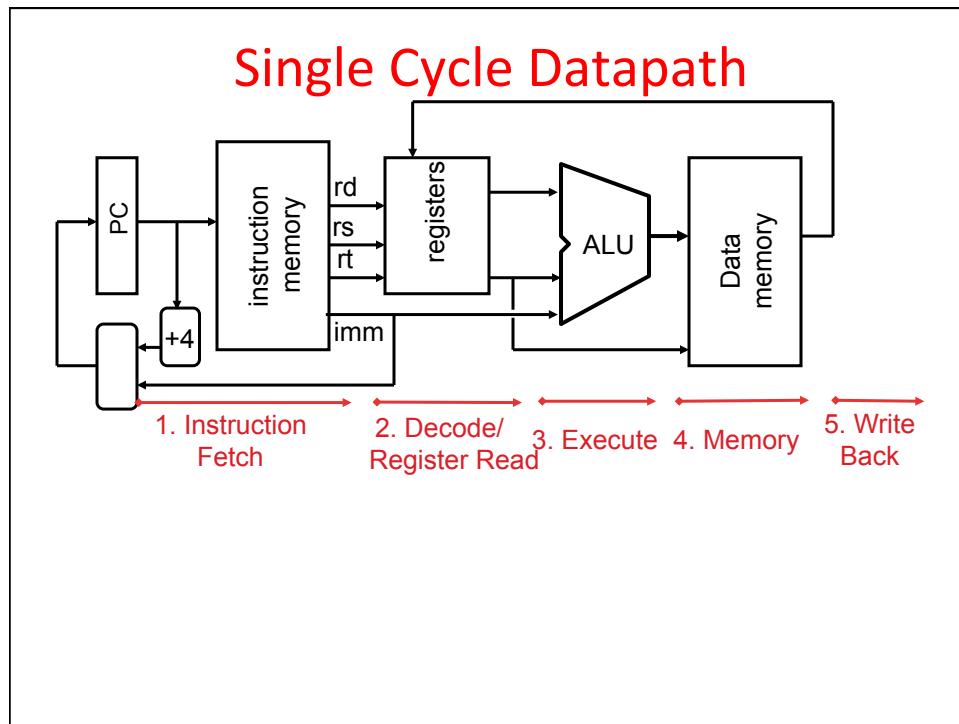
## Pipelining Lessons (2/2)



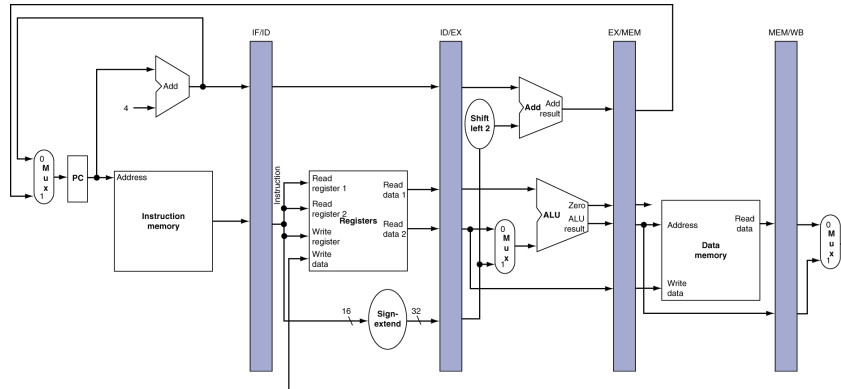
- Suppose new Washer takes 20 minutes, new Stasher takes 20 minutes. How much faster is pipeline?
- Pipeline rate limited by slowest pipeline stage
- Unbalanced lengths of pipe stages reduces speedup

## 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

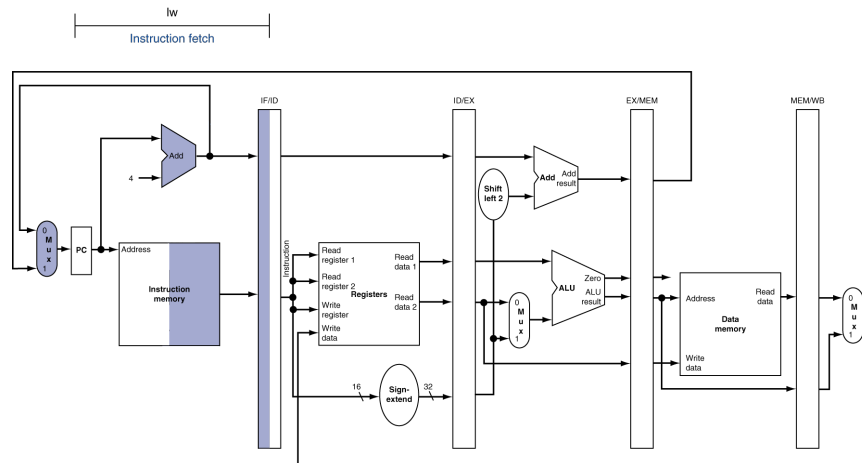


## More Detailed Pipeline



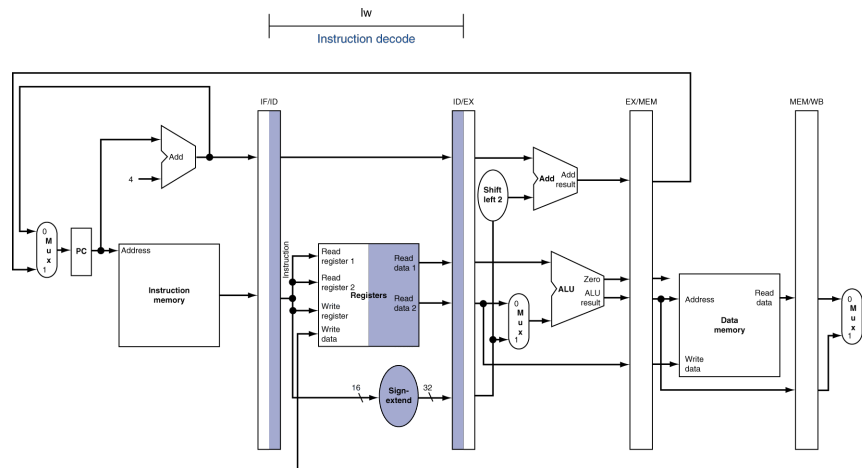
Chapter 4 — The Processor —  
25

## IF for Load, Store, ...



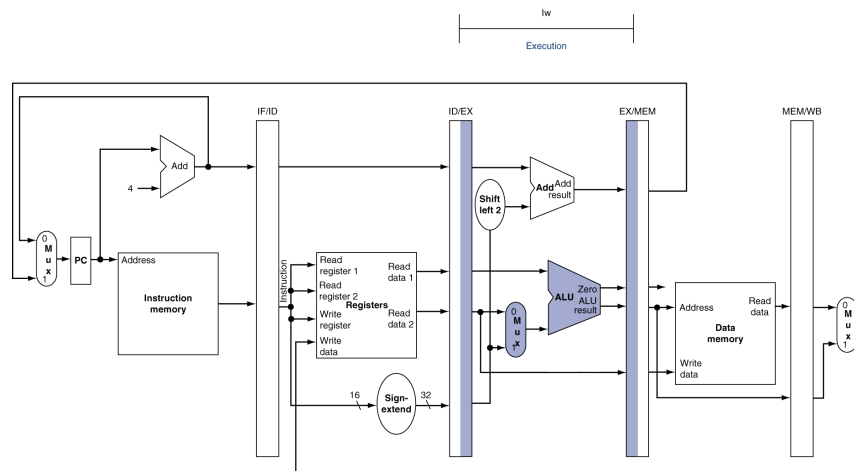
Chapter 4 — The Processor —  
26

## ID for Load, Store, ...



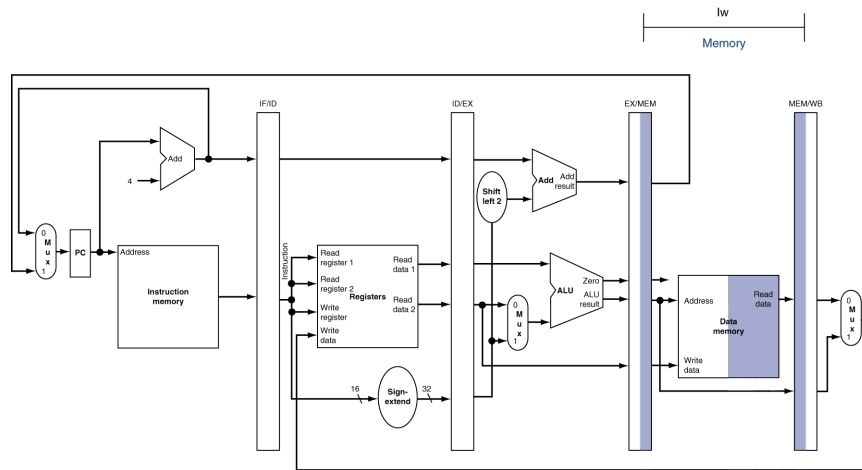
Chapter 4 — The Processor —  
27

## EX for Load



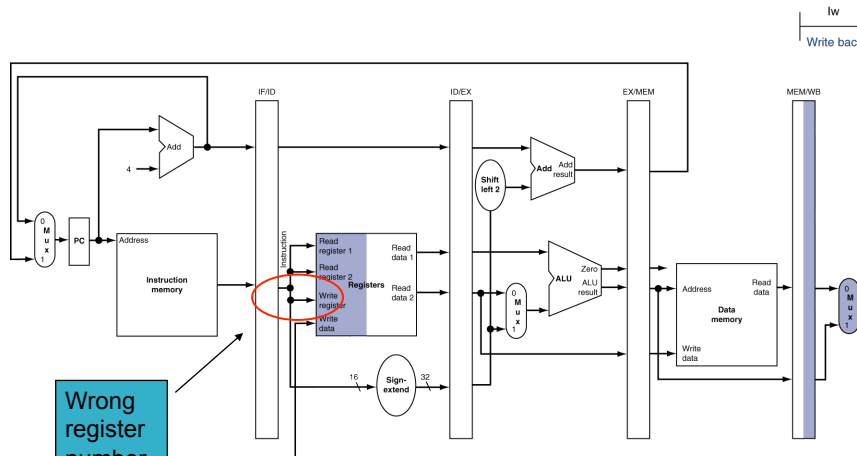
Chapter 4 — The Processor —  
28

# MEM for Load



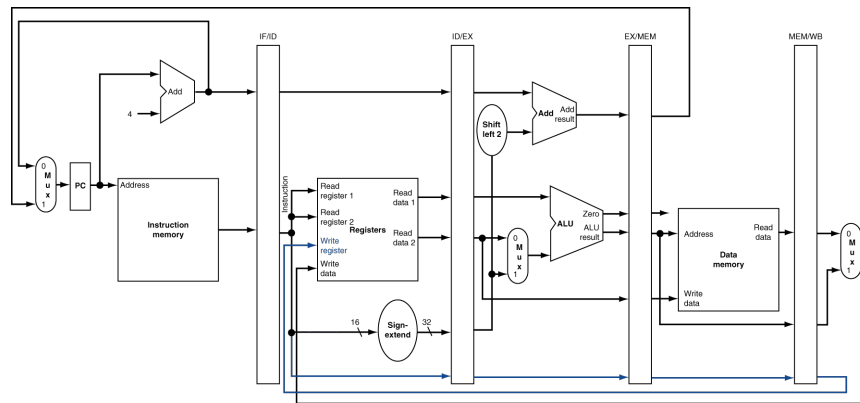
Chapter 4 — The Processor — 29

# WB for Load – Oops!



Chapter 4 — The Processor — 30

## Corrected Datapath for Load



Chapter 4 — The Processor —  
31

## 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