CMPE110 Lecture 10
Building a Processor 2

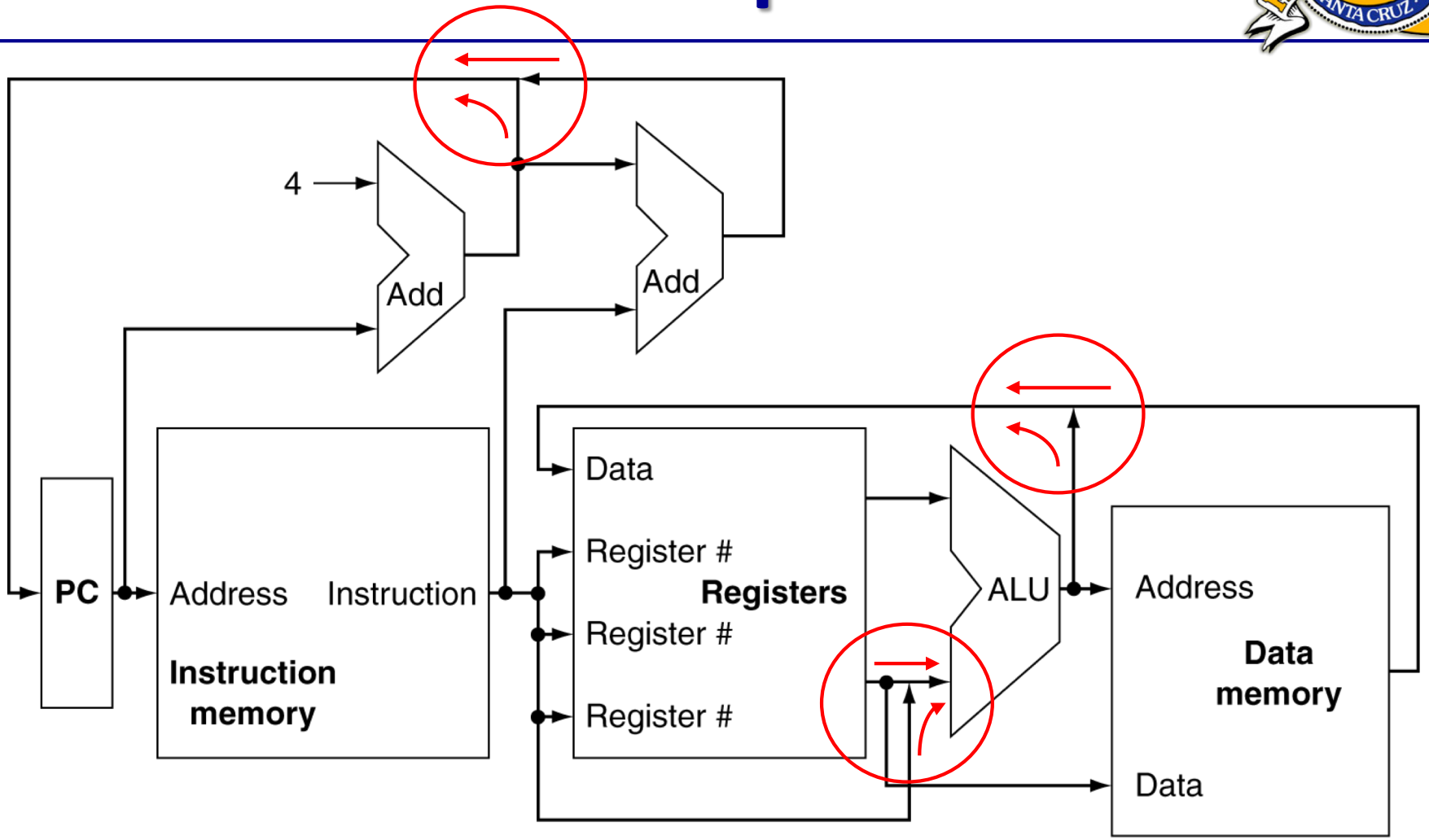
Heiner Litz

https://canvas.ucsc.edu/courses/19290

Announcements

Review

Initial Processor Datapath

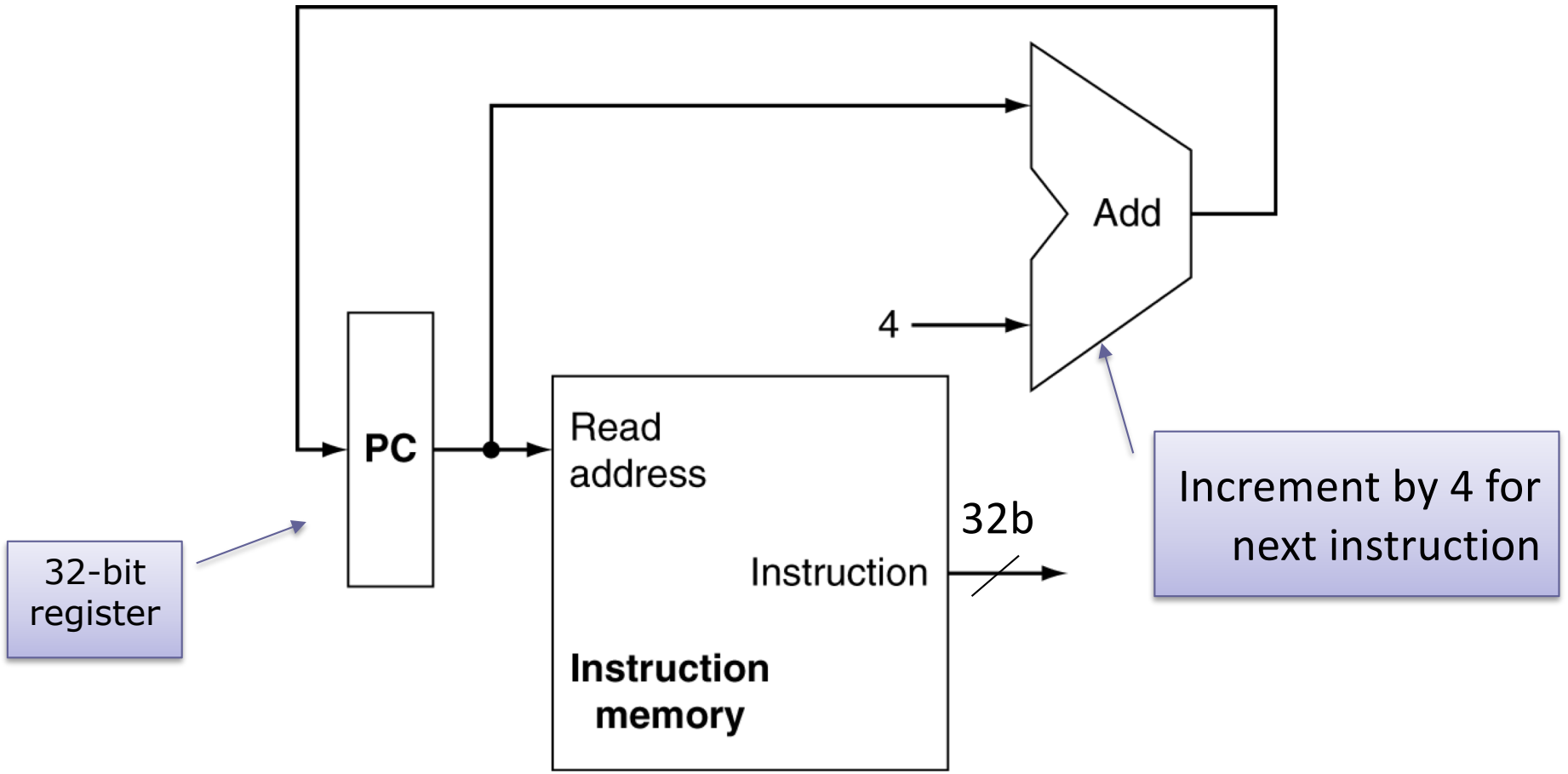


- Cannot just join wires together
  - These connections will actually require multiplexors

Fetching the Instruction

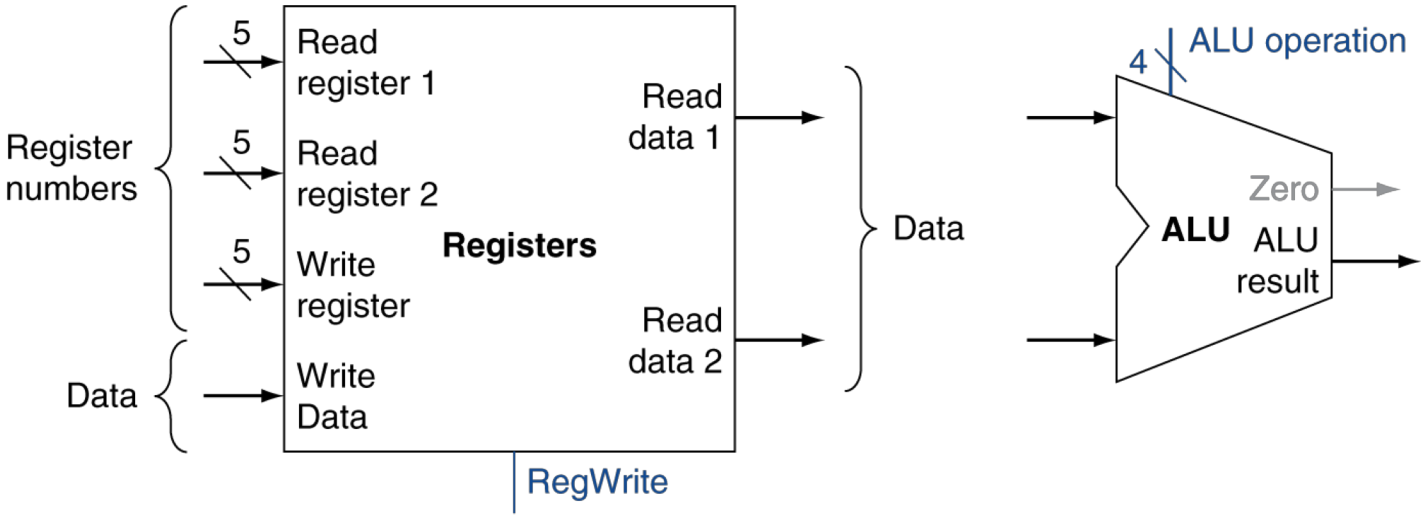
- Not that complex
- Instruction = Memory[PC]
  - Fetch the instruction from memory
  - Always 32-bits
- Update program counter for next cycle
  - What is the address of the next instruction?

Fetching the Instruction



Arithmetic Instructions

- Read two register operands
- Perform arithmetic/logical operation
- Write register result



a. Registers

b. ALU

ORI Instruction

- OR immediate instruction

```
ori rd, rs1, imm #R[rd]<-R[rs1] OR ZeroExt(imm)
```

- Need to get instr[11:0] into the datapath

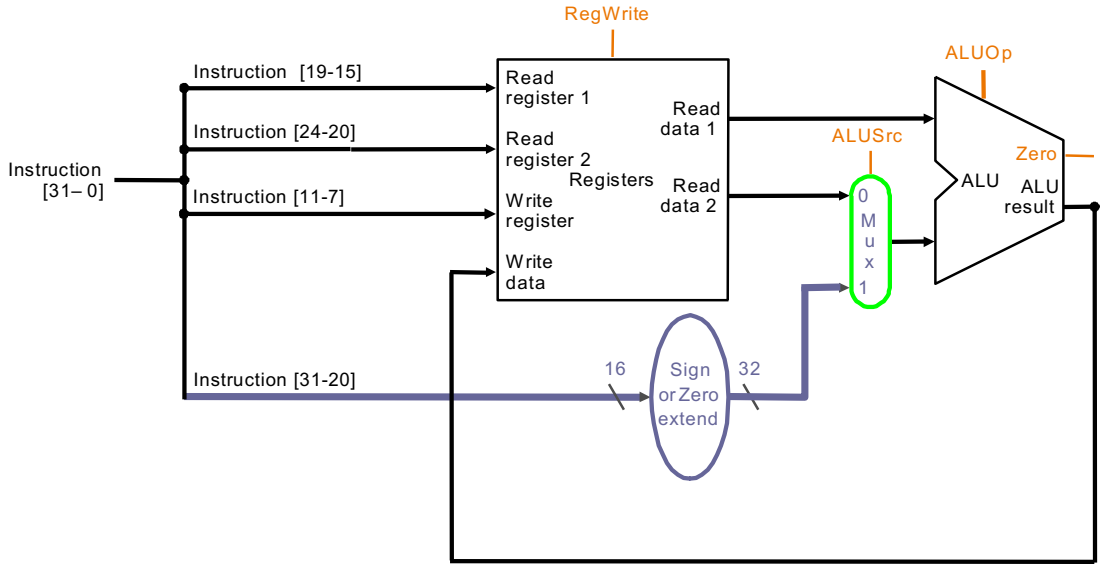
I-type

Datapath: ORI Instruction

- Read data 2 is ignored for immediates
- ALUsrc and ALUOp set based on instruction

I-type

immediate[11:0]	rs1	funct3	rd	opcode



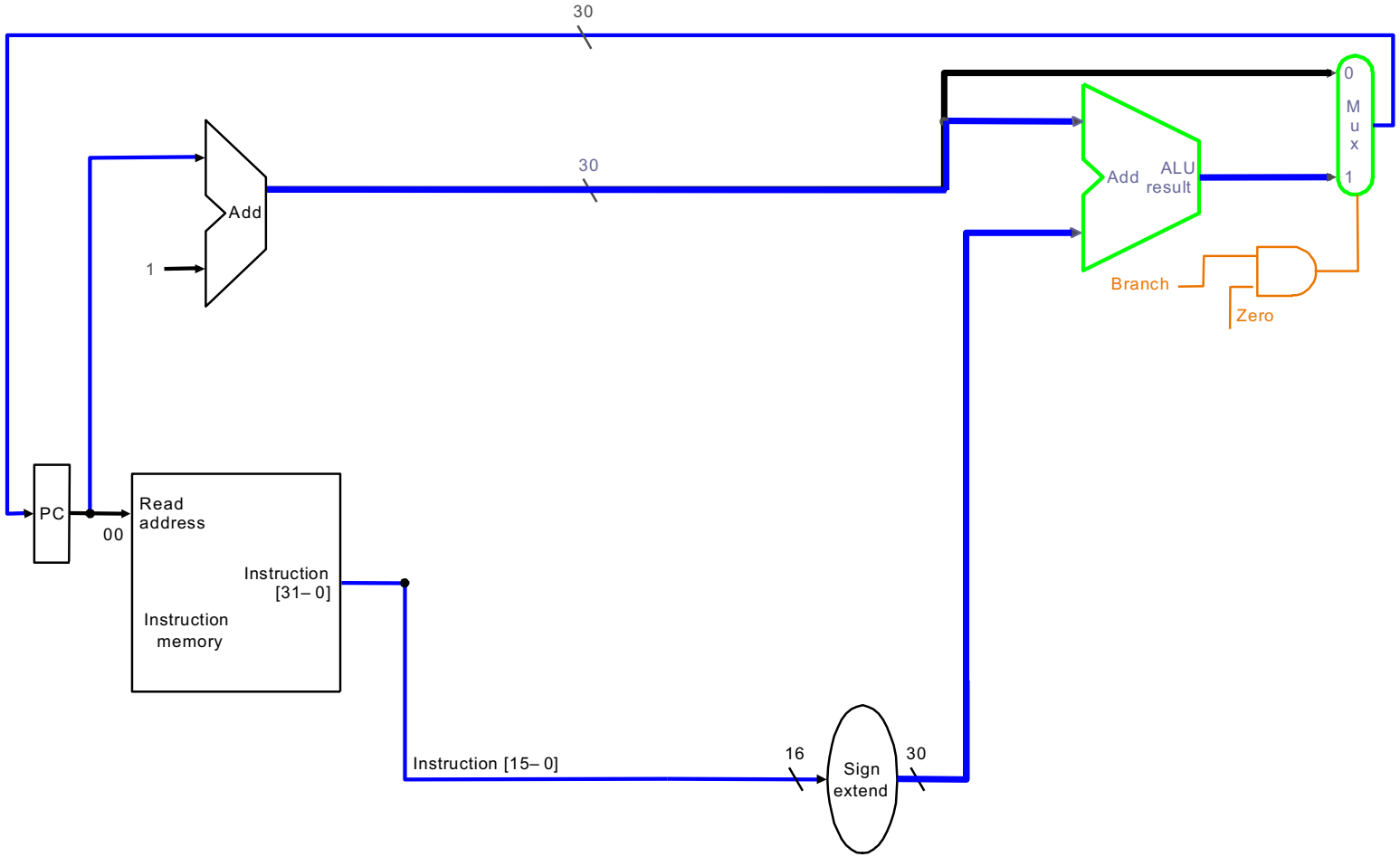
Branch Instruction

- Branch instruction: `beq rs1, rs2, immediate`

```
Cond <- R[rs1] - R[rs2]
if (cond eq 0)
        PC <- PC + 4 + SignExt(imm)*4
else
        PC <- PC + 4;
```

S/B-type

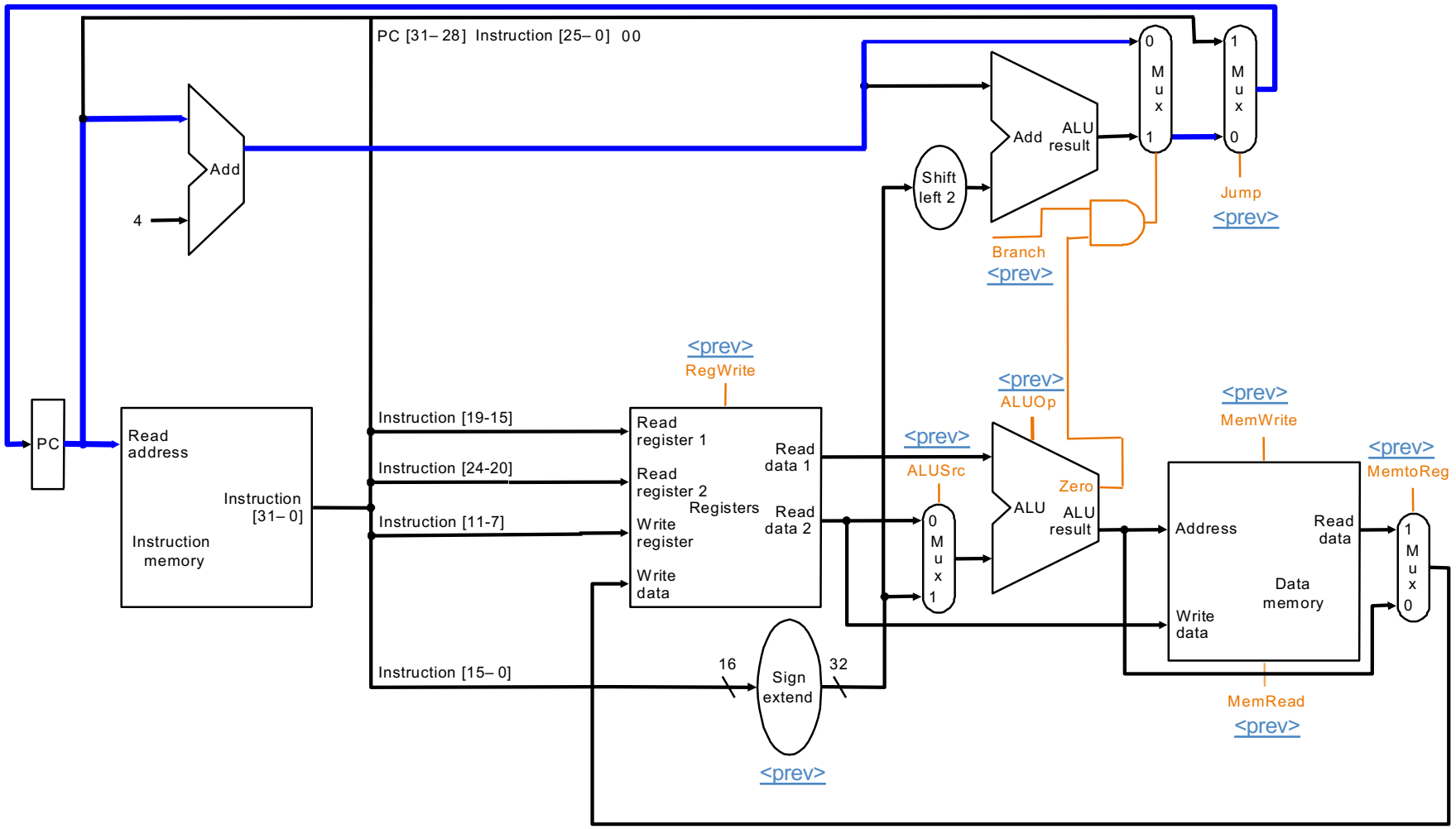
Datapath for the PC



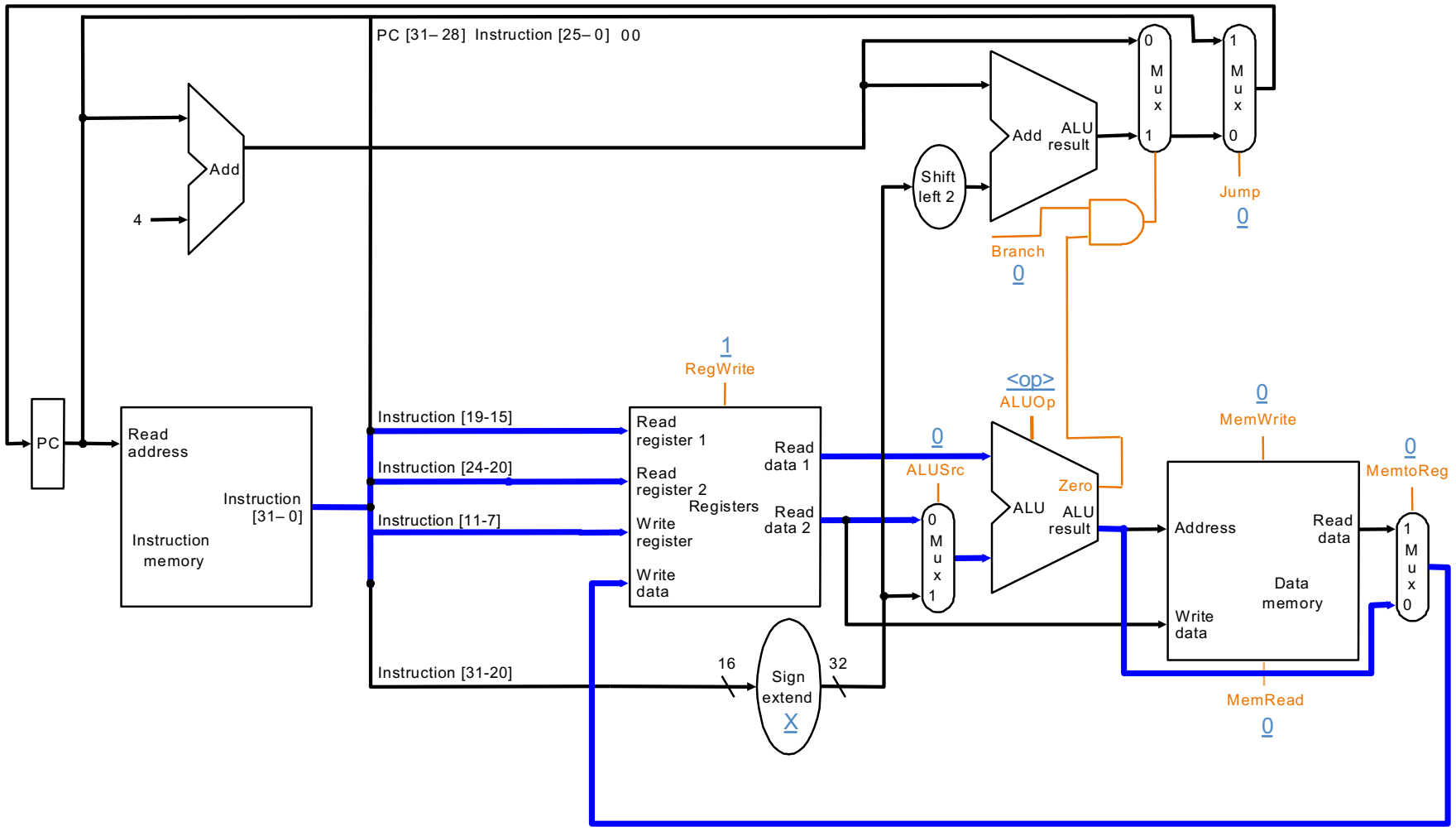
Control

- State free
  - Every instruction takes a single cycle
  - Just decode instruction bits
- There are also few control points
  - Control on the multiplexers
  - Operation type for the ALU
  - **Write control on the register file & data memory**

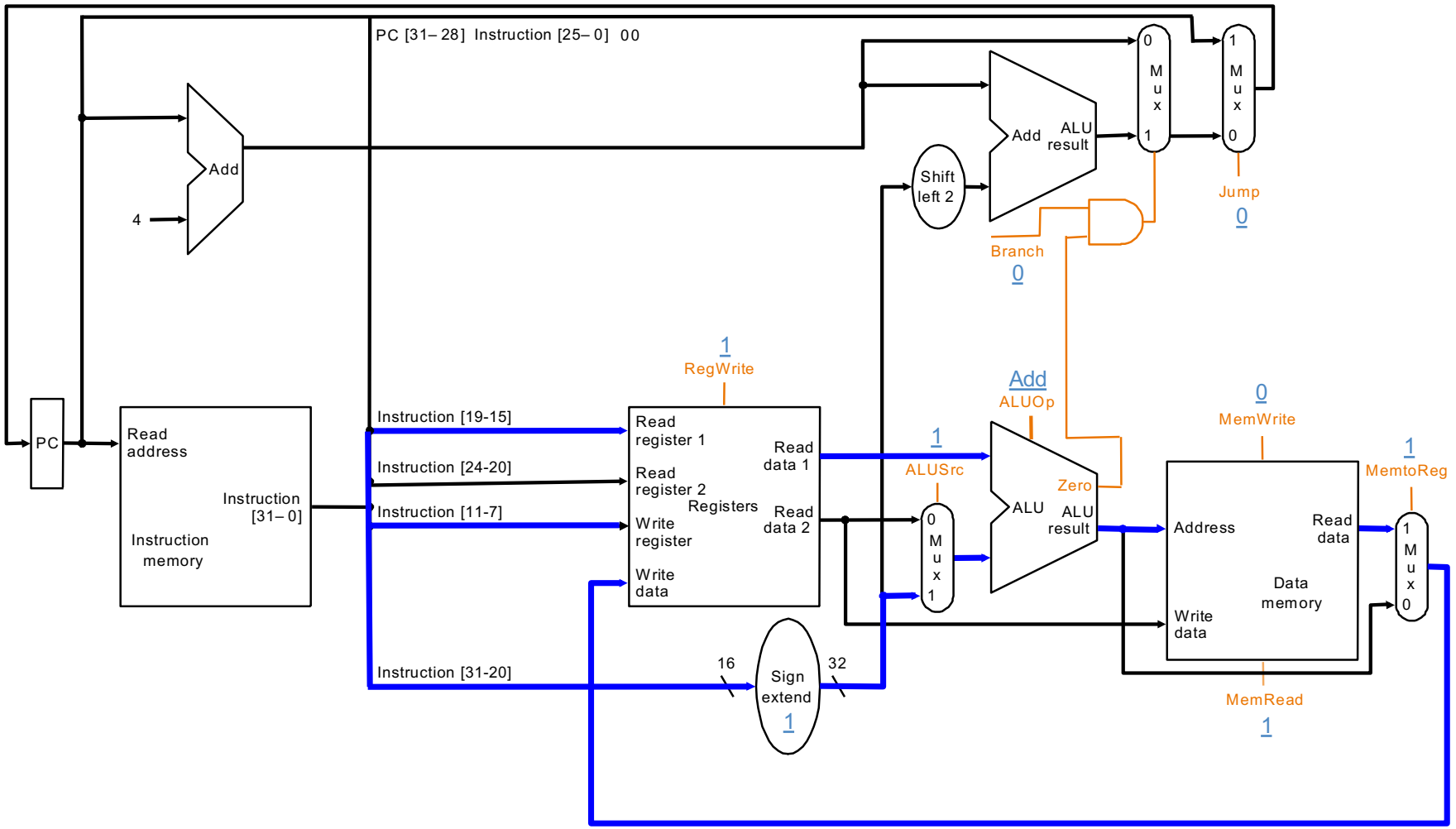
Control: Instruction Fetch



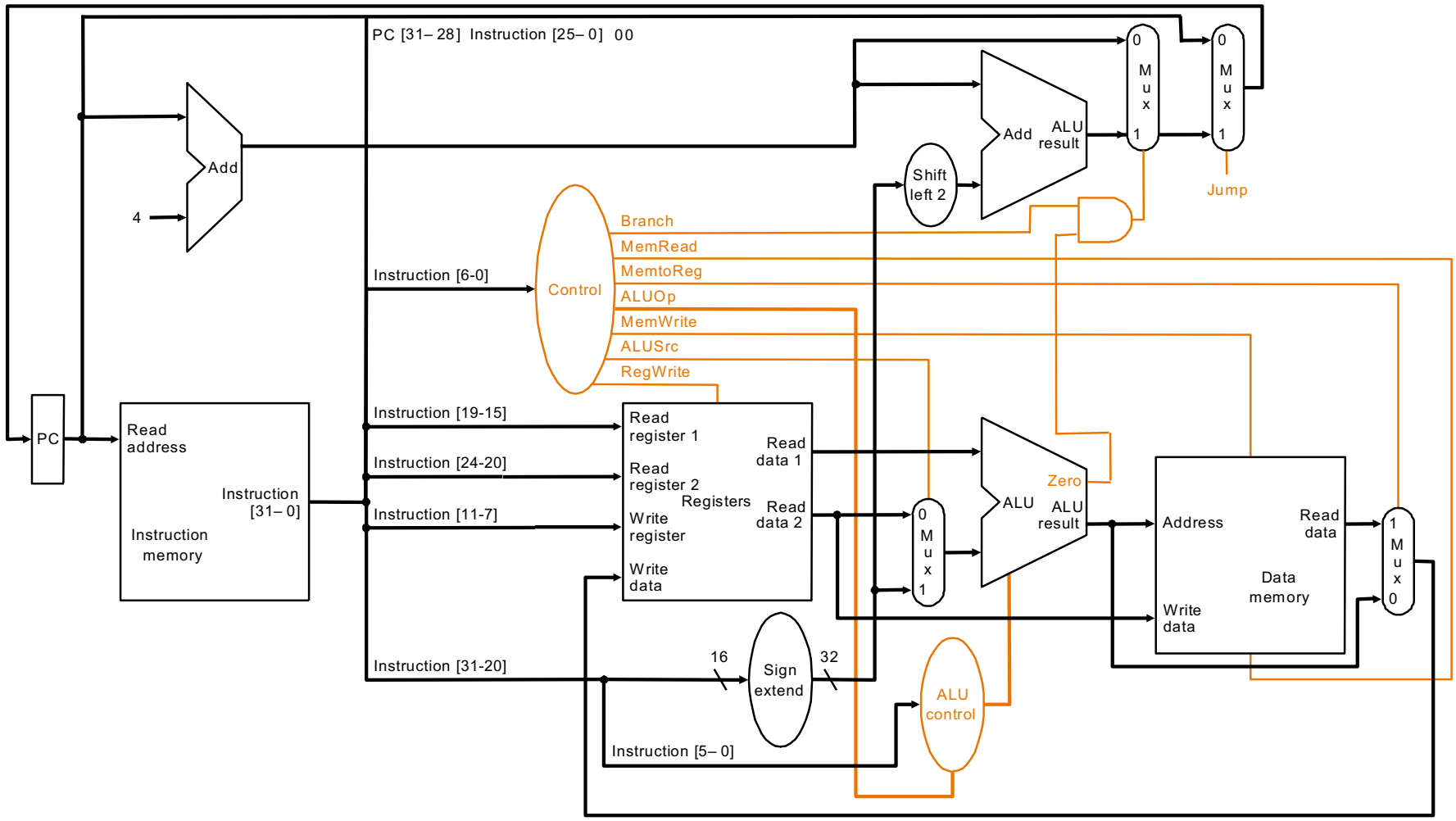
Control: addu



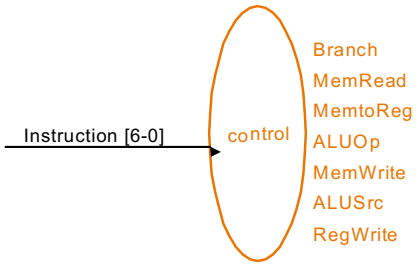
Control: Load



Putting It All Together: Our First Processor



How to generate control signals?

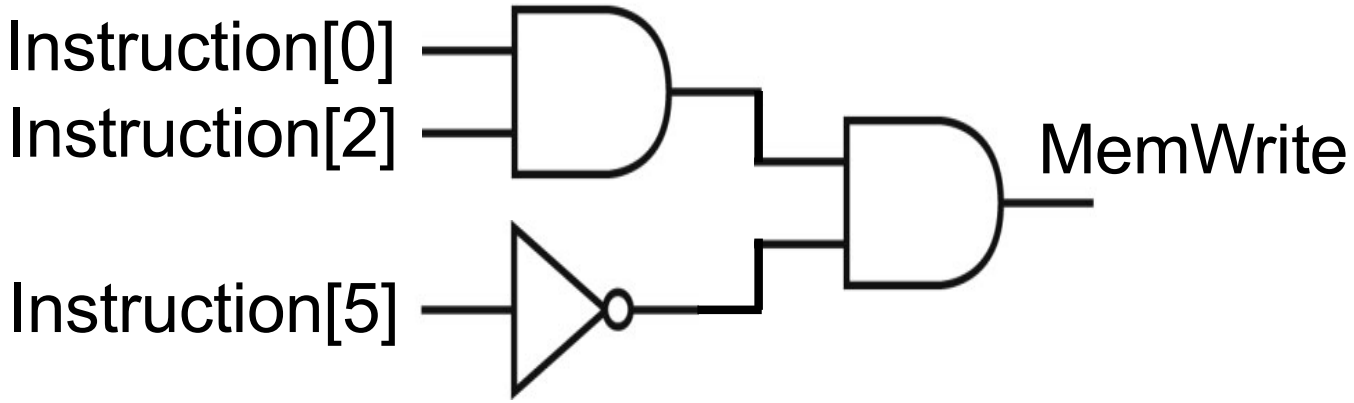


- **Consider the hypothetical example:**
  - **MemWrite equals 1 if:**

    Instruction[0] & Instruction[2] &

    ! Instruction[5]

- **Build using combinatorial logic**



Single Cycle Processor Performance

- Functional unit delay
  - Memory: 200ps
  - ALU and adders: 200ps
  - Register file: 100 ps

Instruction Class	Instruction memory	Register read	ALU operation	Data memory	Register write	Total
R-type	200	100	200		100	600
load	200	100	200	200	100	800
store	200	100	200	200		700
branch	200	100	200			500
jump	200					200

- CPU clock cycle = 800 ps = 0.8ns (1.25GHz)

Single Cycle RISC-V Processor

- Pros
  - Single cycle per instruction makes logic simple
- Cons
  - Cycle time is the worst case path → long cycle times
    - Worst case = load
  - Hardware is underutilized
    - ALU and memory used only for a fraction of clock cycle
    - Not well amortized!
  - Best possible CPI is 1

Variable Clock Single Cycle Processor Performance

- Instruction Mix
  - 45% ALU
  - 25% loads
  - 10% stores
  - 15% branches
  - 5% jumps

Instruction Class	Instruction memory	Register read	ALU operation	Data memory	Register write	Total
R-type	200	100	200		100	600
load	200	100	200	200	100	800
store	200	100	200	200		700
branch	200	100	200			500
jump	200					200

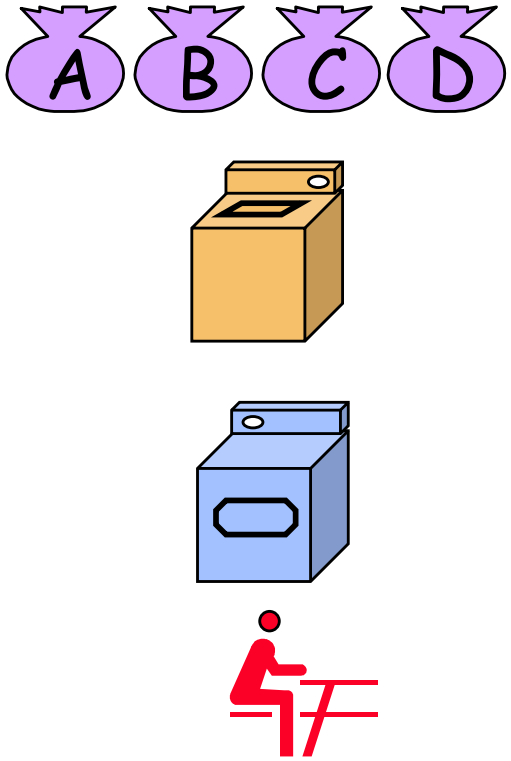
- CPU clock cycle = 0.6x45% + 0.8x25% + 0.7x10% + 0.5x15% + 0.2x5%
  = 0.625 ns (1.6GHz)
- Difficult to implement

Key Tools for System Architects

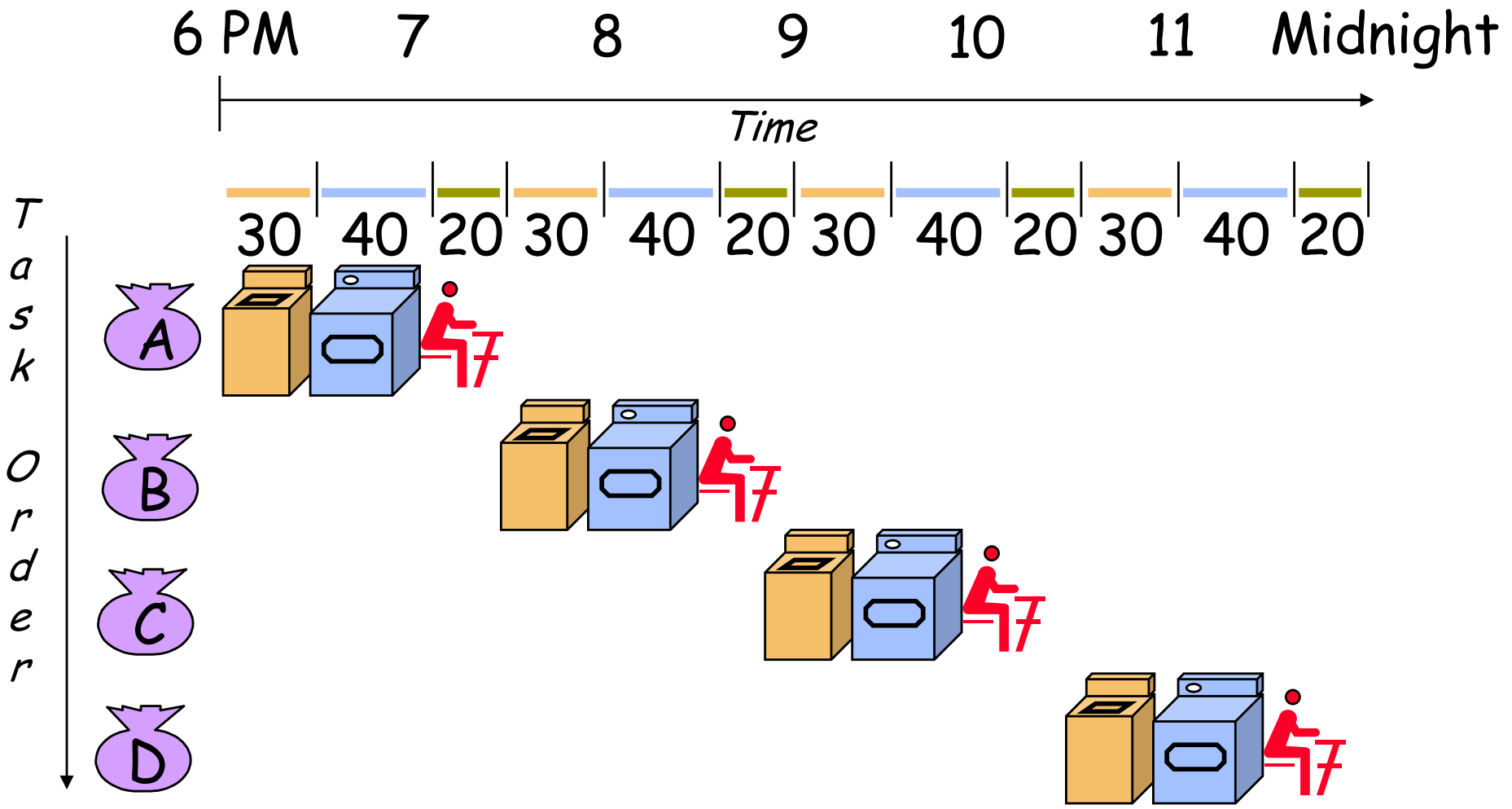
1. **Pipelining**
2. Parallelism
3. Out-of-order execution
4. Prediction
5. Caching
6. Indirection
7. **Amortization**
8. Redundancy
9. Specialization
10. Focus on the common case

Pipelining: The Laundry Analogy

- Ann, Brian, Cathy, Dave doing laundry
- Washer takes 30 minutes
- Dryer takes 40 minutes
- "Folding bench" takes 20 minutes

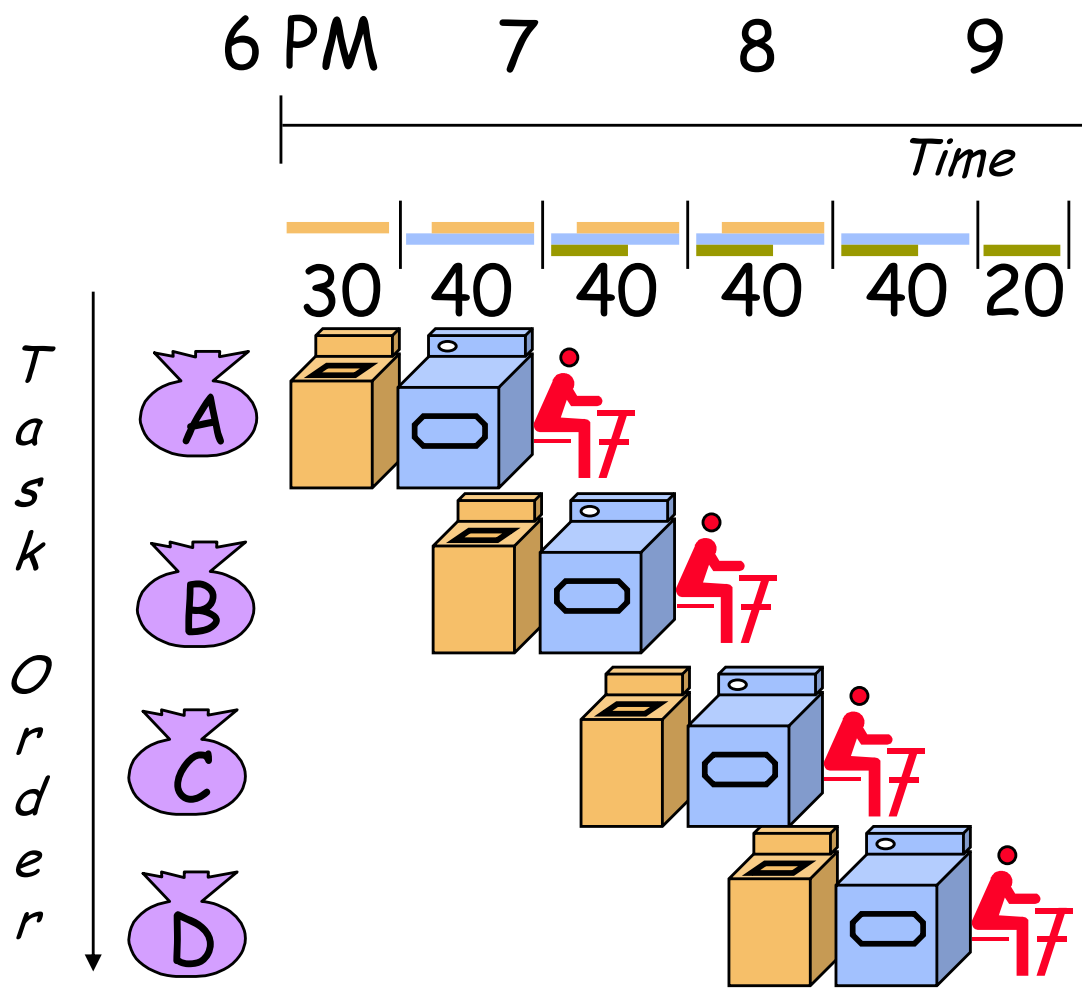


Single-cycle Laundry



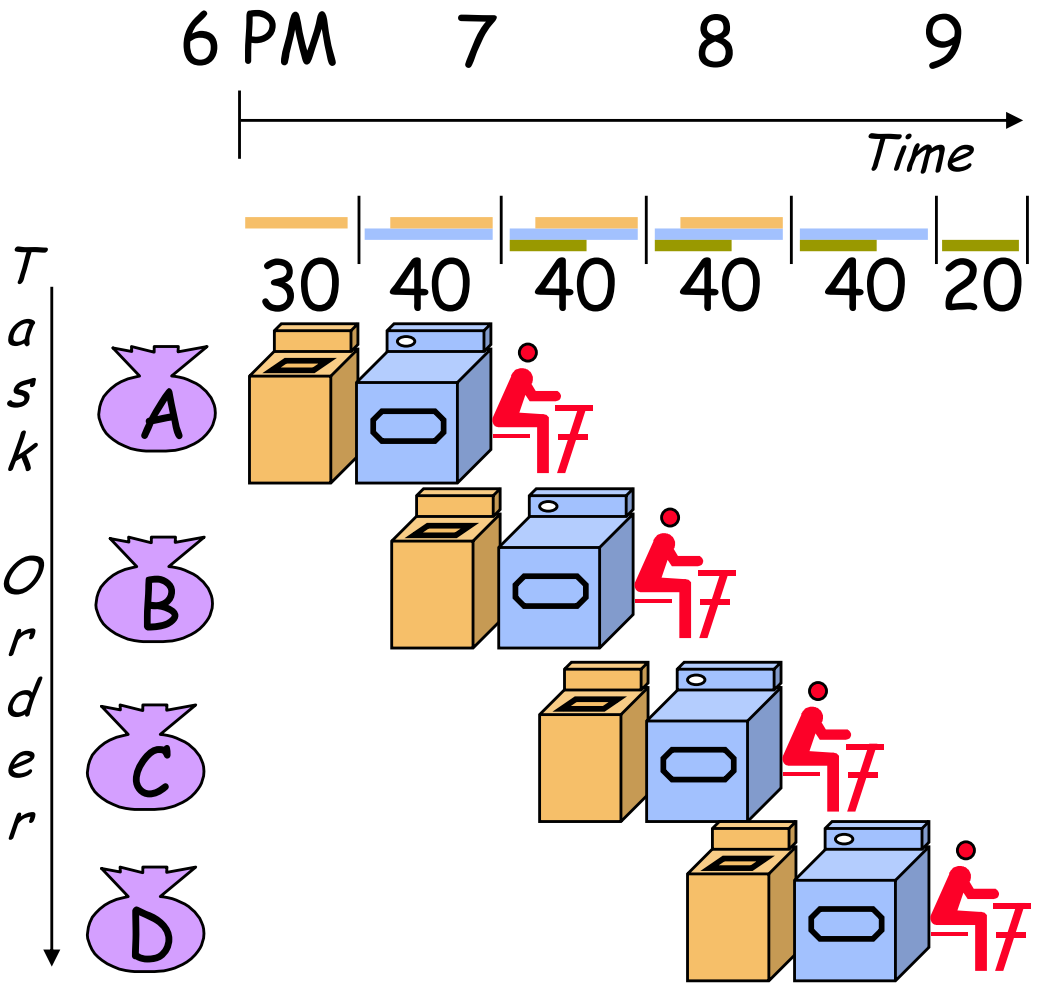
Single-cycle laundry takes 6 hours for 4 loads

Pipelined Laundry



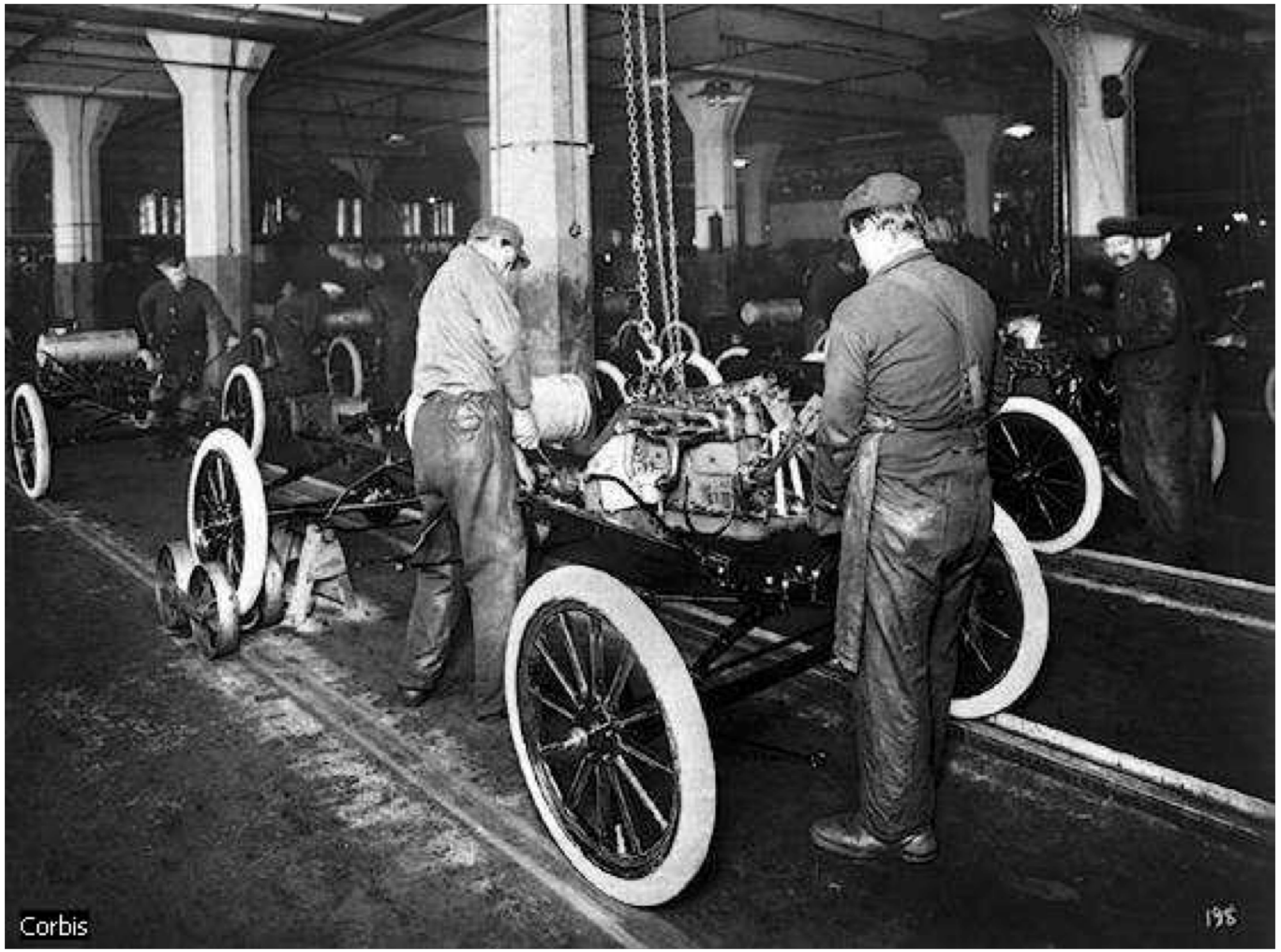
Pipelined laundry takes 3.5 hours for 4 loads

Lessons from Laundry Analogy



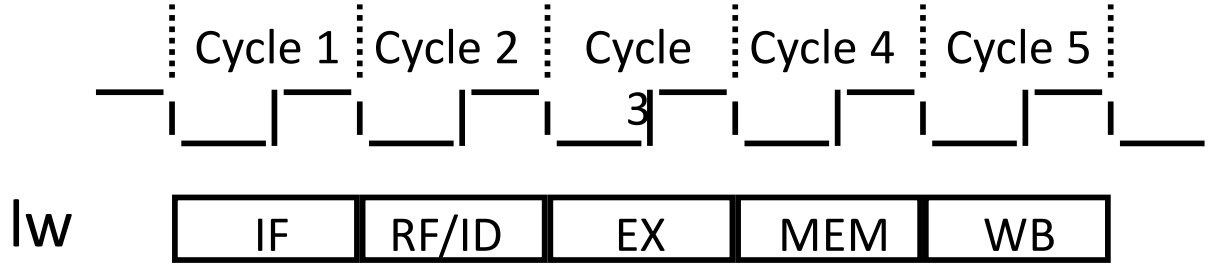
- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Multiple tasks operating simultaneously
- Potential speedup = Number pipe stages
- Pipeline rate limited by slowest pipeline stage
- Unbalanced lengths of pipe stages reduces speedup
- Time to "fill" pipeline and time to "drain" it reduces speedup

Another Analogy: Model T Assembly Line



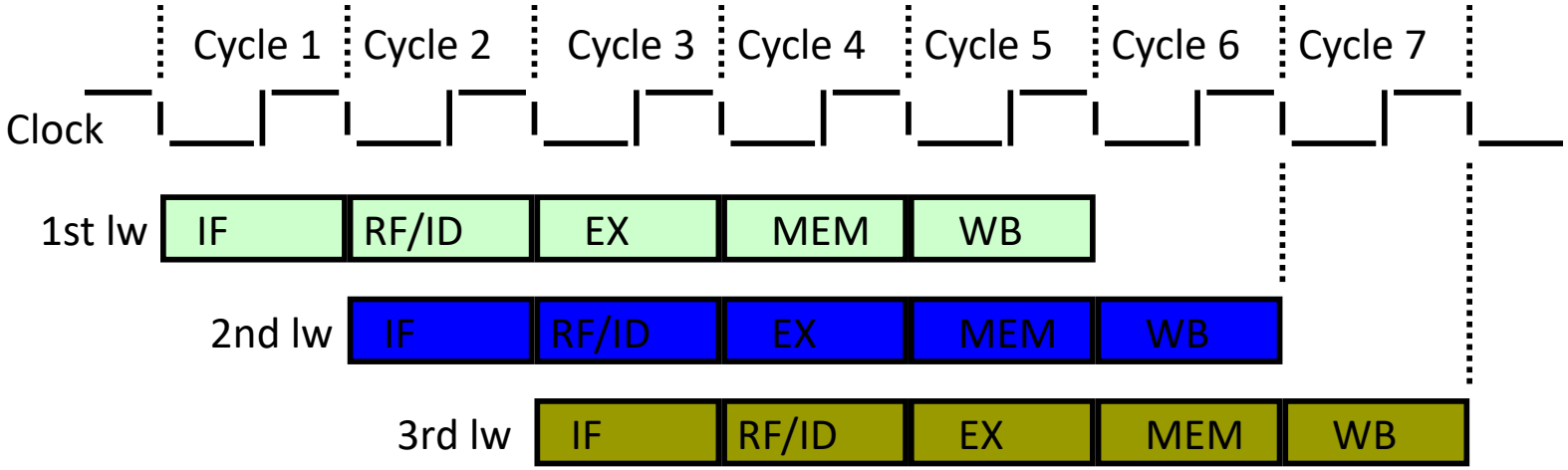
Pipelining the Processor

- 5 stages, one clock cycle per stage
  - IF: instruction fetch from memory
  - ID: instruction decode & register read
  - EX: execute operation or calculate address
  - MEM: access memory operand
  - WB: write result back to register



Pipelining the Processor

- Overlap instructions in different stages
  - All hardware used all the time (amortization)
  - Clock cycle is fast
  - CPI is still 1



To Be Continued

- Pipelined datapath and control
- Pipeline dependencies, hazards, and stalls
- The limits of pipelining