CS/ECE 552: Single Cycle Control Path

Prof. Matthew D. Sinclair

Lecture notes based in part on slides created by Mark Hill, Mikko Lipasti, David Wood, Guri Sohi, John Shen and Jim Smith

Control Overview

- Single-cycle implementation
  - Datapath: combinational logic, I-mem, regs, D-mem, PC
    - Last three written at end of cycle
  - Need control – just combinational logic!
  - Inputs:
    - Instruction (I-mem out)
    - Zero (for beq)
  - Outputs:
    - Control lines for muxes
    - ALUop
    - Write-enables (Register File, Data Memory)
    - Read-enable (Data Memory)

Control Overview

- Fast control
  - Divide up work on “need to know” basis
  - Logic with fewer inputs is faster
- E.g.:
  - Global control need not know which ALUop

ALU Control

- Assume ALU uses

000	and
001	or
010	add
110	sub
111	slt (set less than)
others	don’t care

ALU Control

Instruction	Operation	Opcode	Function
add	add	000000	100000
sub	sub	000000	100010
and	and	000000	100100
or	or	000000	100101
slt	slt	000000	101010

- ALU-ctrl = f(opcode,function)

But…don't forget

Instruction	Operation	Opcode	function
lw	add	100011	xxxxxx
sw	add	101011	xxxxxx
beq	sub	000100	100010

- To simplify ALU-ctrl
  - ALUop = f(opcode)

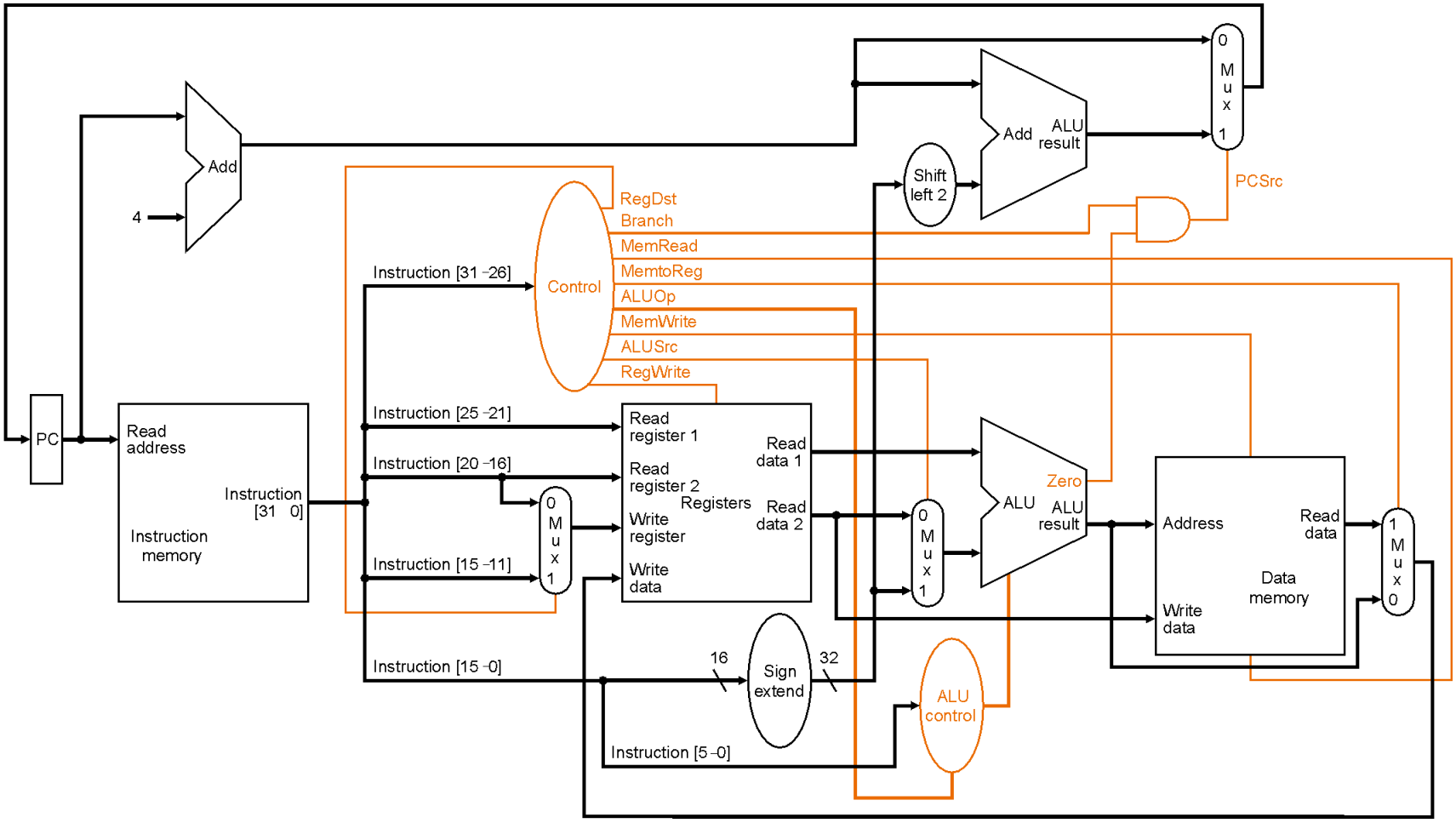
    2 bits 6 bits

ALU Control

10	add, sub, and, …
00	lw, sw
01	beq

- ALU-ctrl = f(ALUop, function)
- 3 bits 2 bits 6 bits
- Requires only five gates plus inverters

Control Signals Needed



Global Control

- R-format: opcode rs rt rd shamt function
  6 5 5 5 5 6
- I-format: opcode rs rt address/immediate
  6 5 5 16
- J-format: opcode address
  6 26

Global Control

- Route instruction[25:21] as read reg1 spec
- Route instruction[20:16] are read reg2 spec
- Route instruction[20:16] (load) and instruction[15:11] (others) to
  - Write reg mux
- Rename instruction[31:26] op[5:0]

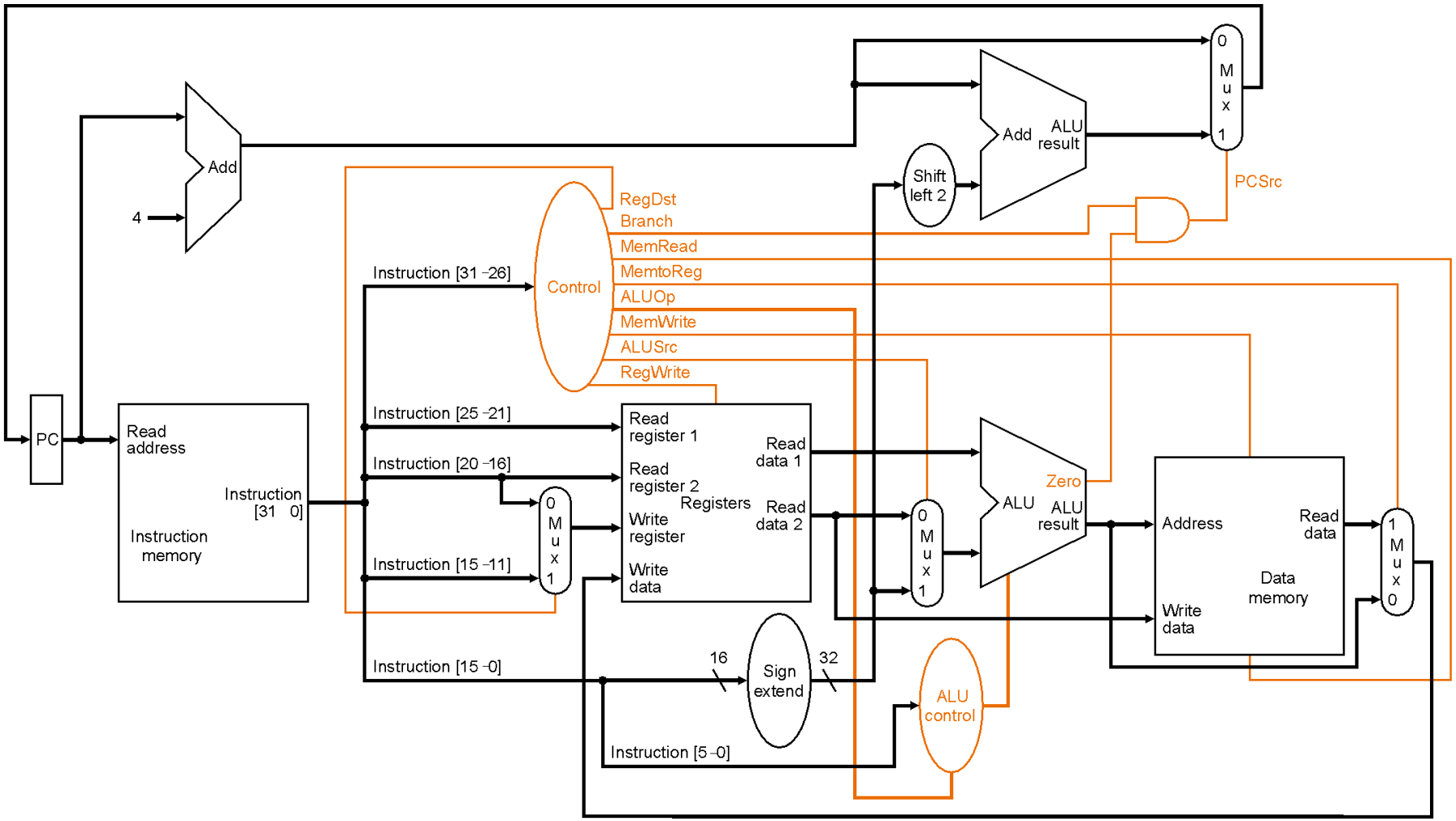
Global Control

- Global control outputs
  - ALU-ctrl - see previous slides
  - ALU src - R-format, beq vs. ld/st
  - MemRead - lw
  - MemWrite - sw
  - MemtoReg - lw
  - RegDst - lw dst in bits 20:16, not 15:11
  - RegWrite - all but beq and sw
  - PCSrc - beq taken

Global Control

- Global control outputs
  - Replace PCsrc with
    - Branch beq
    - PCSrc = Branch * Zero
- What are the inputs needed to determine above global control signals?
  - Just Op[5:0]

Control Signals Needed



Global Control

Instruction	Opcode	RegDst	ALUSrc
rrr	000000	1	0
lw	100011	0	1
sw	101011	x	1
beq	000100	x	0
???	others	x	x

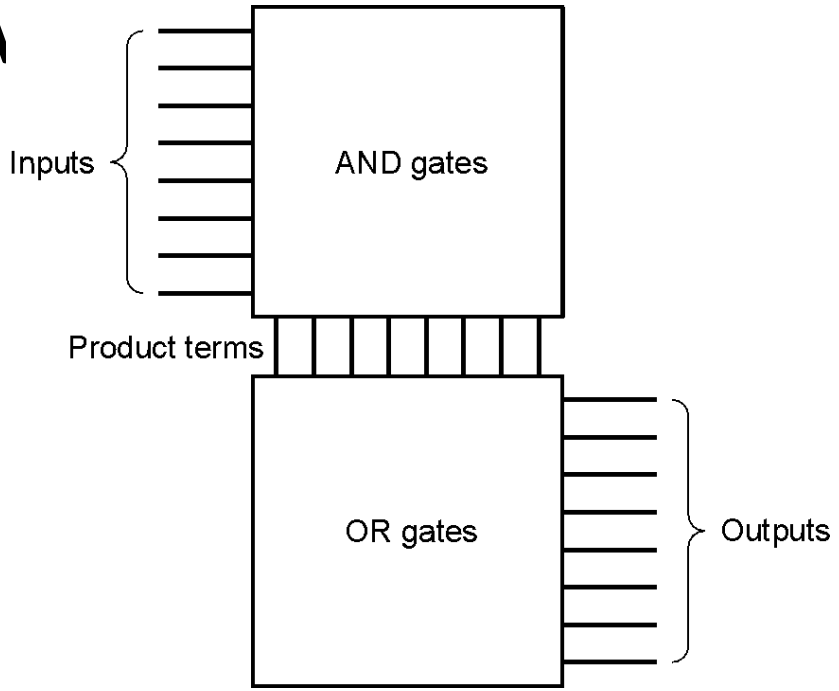
- RegDst = ~Op[0]
- ALUSrc = Op[0]
- RegWrite = ~Op[3] * ~Op[2]

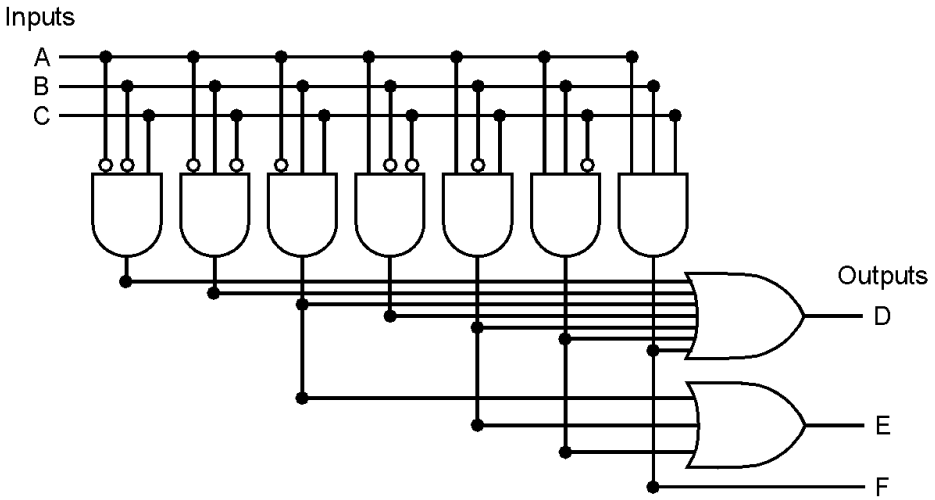
Global Control

- More complex with entire MIPS ISA
  - Need more systematic structure
  - Want to share gates between control signals
- Common solution: PLA
  - MIPS opcode space designed to minimize PLA inputs, minterms, and outputs
- Refer to MIPS Opcode map

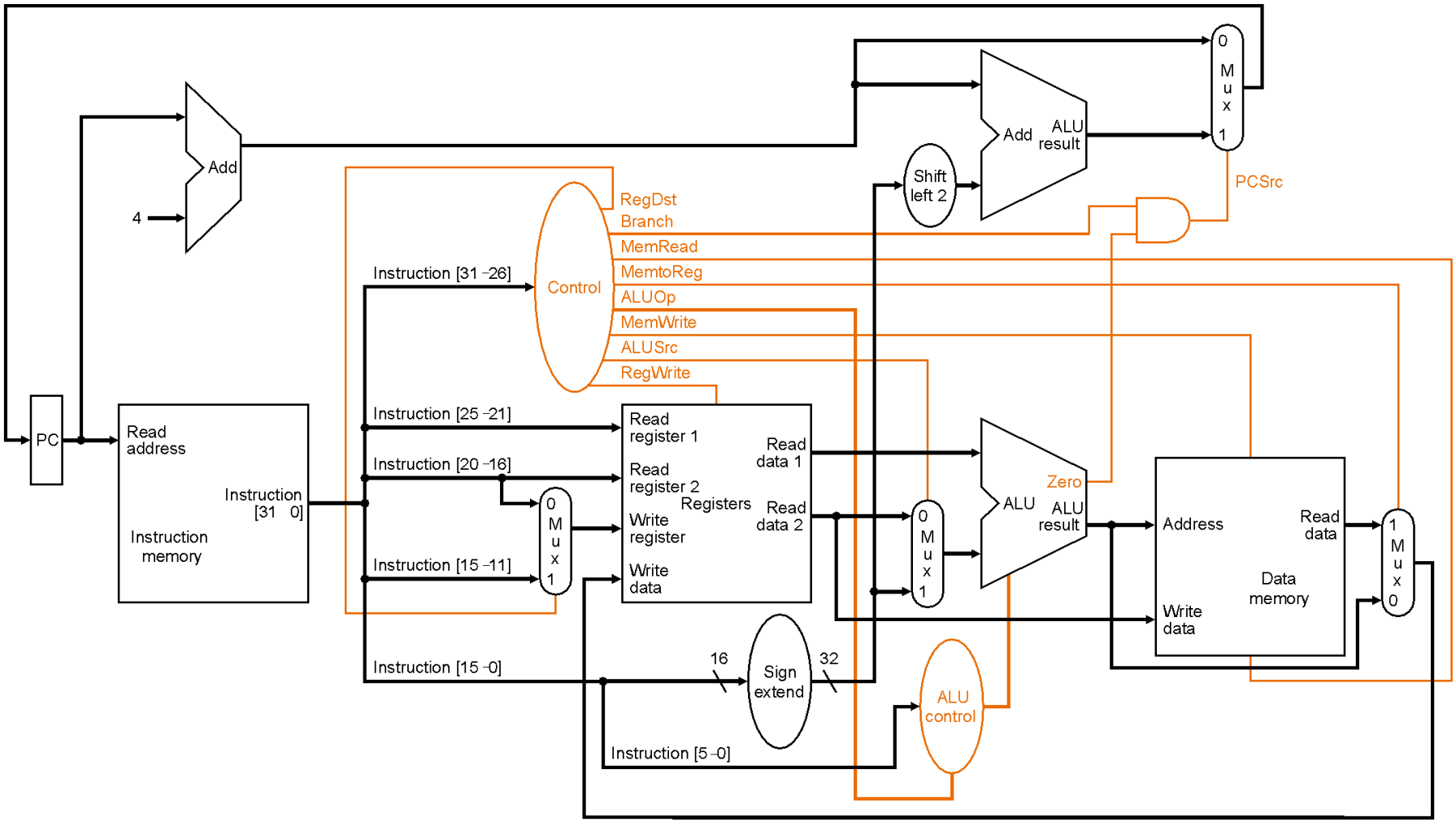
PLA

- In AND-plane, & selected inputs to get minterms
- In OR-plane, | selected minterms to get outputs
- E.g.:

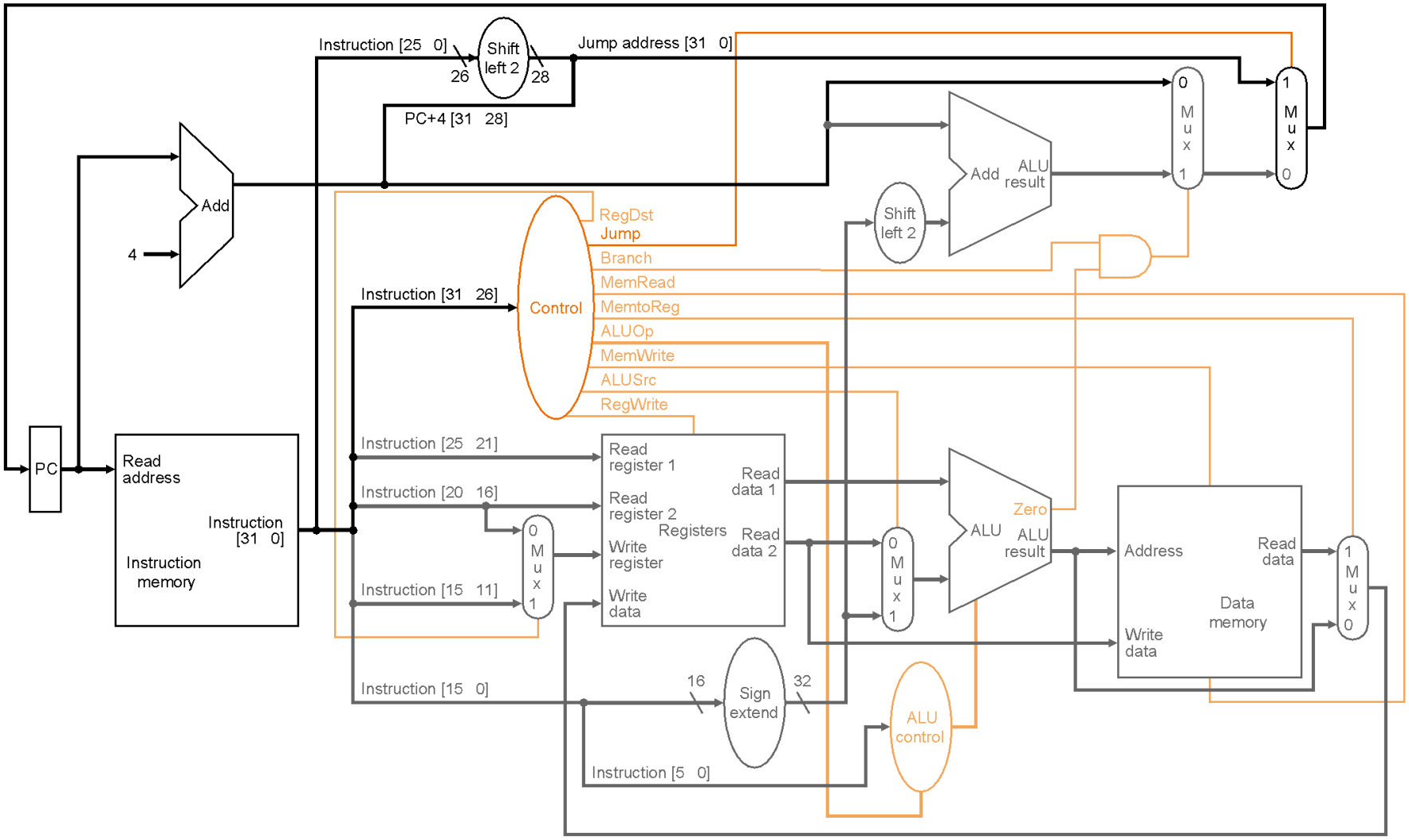




How Can we add Jumps?



Control Signals w/Jumps



What's wrong with single cycle?

$$\frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

(code size) **(CPI)** **(cycle time)**

- Critical path probably lw:
  - I-mem, reg-read, alu, d-mem, reg-write
- Other instructions faster
  - E.g., rrr: skip d-mem
- Instruction variation much worse for full ISA and real implementation:
  - FP divide
  - Cache misses (what the heck is this? – later)

Single Cycle Implementation

- Solution
  - Variable clock?
    - Too hard to control, design
  - Fixed short clock
    - Variable cycles per instruction
- Multicycle control (next lecture)

Summary

- Processor implementation
  - Datapath
  - Control
- Single cycle implementation