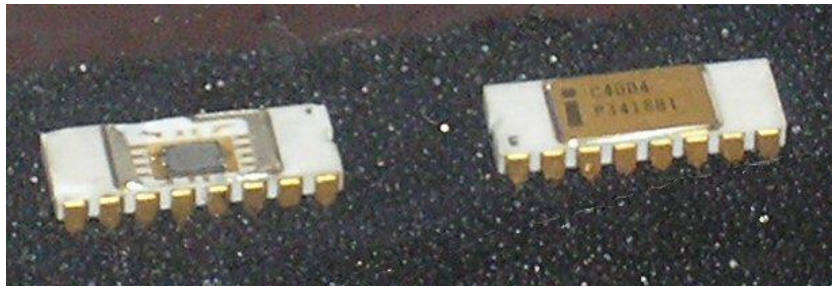
CSCI 210: Computer Architecture
Lecture 25: Control Path

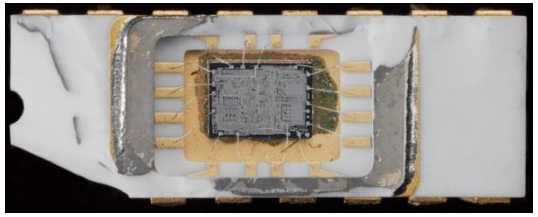
Stephen Checkoway
Slides from Cynthia Taylor

Announcements

- Problem Set 8 Due TONIGHT 10pm
  - Problem Sets 5 and 6 graded, resubmits up
- Lab 7 due Monday 10 pm
- Computational Skills Hours, King 225
  - Wednesday, Sunday 7 – 9 pm
- Cynthia's Student Hours King 139 B
  - Monday 4:30 – 5:30 pm
  - Thursday 10 am – noon

CS History: Intel 4004





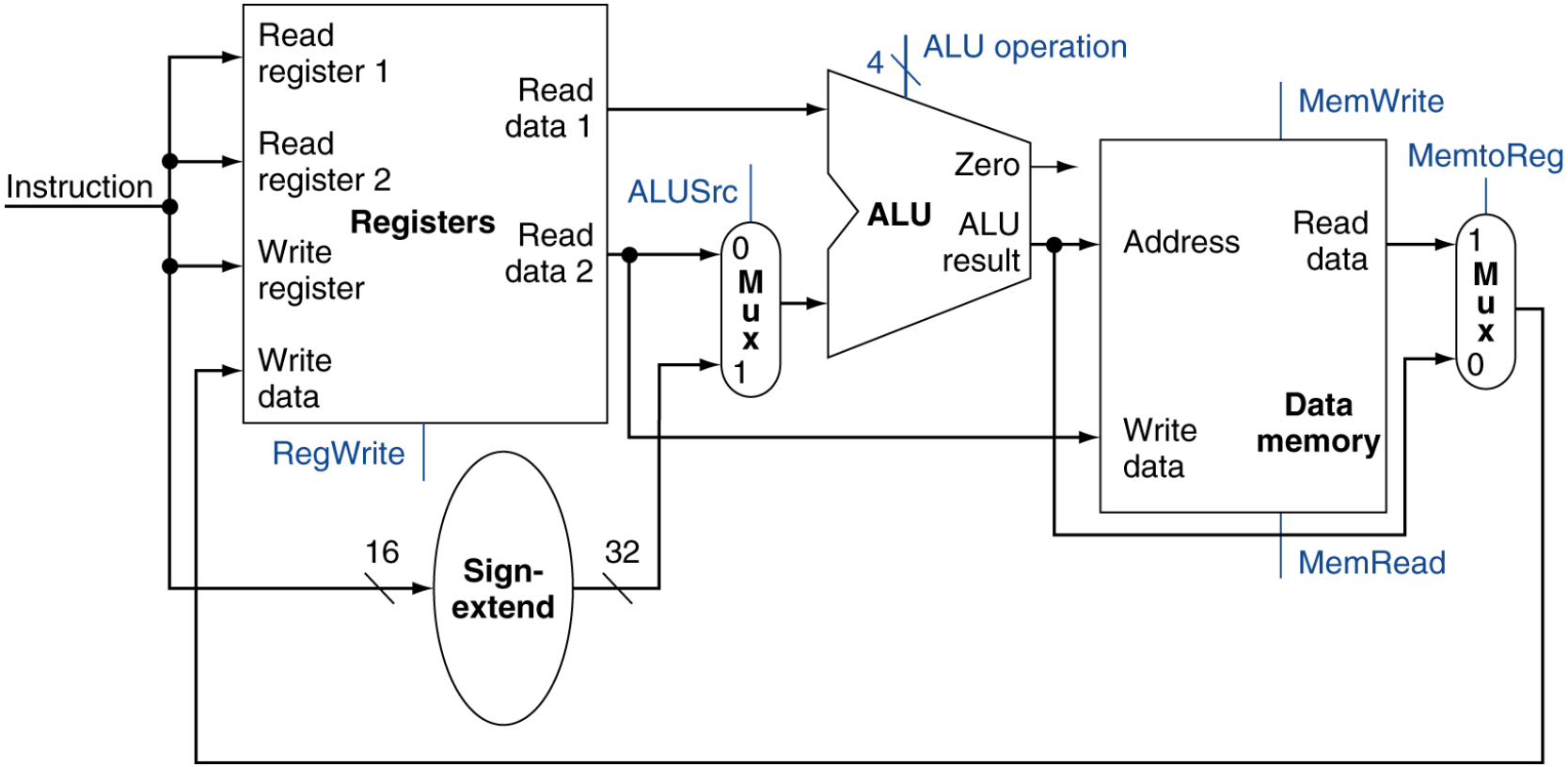
- First commercially available microprocessor (single chip with both data processing logic and control)
- Released in 1971
- Had 12-bit addresses, 8-bit instructions, and 4-bit data words
- 16 4-bit registers
- Designed for Binary-Coded Decimal, in which every decimal digit is stored as a 4-bit value
  - Still present in x86

The Processor: Datapath & Control

- We're ready to look at an implementation of MIPS simplified to contain only:
  - memory-reference instructions: `lw, sw`
  - arithmetic-logical instructions: `add, sub, and, or, slt`
  - control flow instructions: `beq`

lw $t1, 4($t0)

$t0 is register 8, $t1 is register 9
$t0 holds 0x07AB8110
0x07AB8114 holds 12



op = lw	rs = 8	rt = 9	imm = 4

Branch Instructions

- Read register operands
- Compare operands
  - Use ALU, subtract and check Zero output
- Calculate target address
  - Sign-extend offset
  - Shift left 2 bits (word offset)
  - Add to PC + 4
    - Already calculated during instruction fetch

Conditional Branch Instructions Require

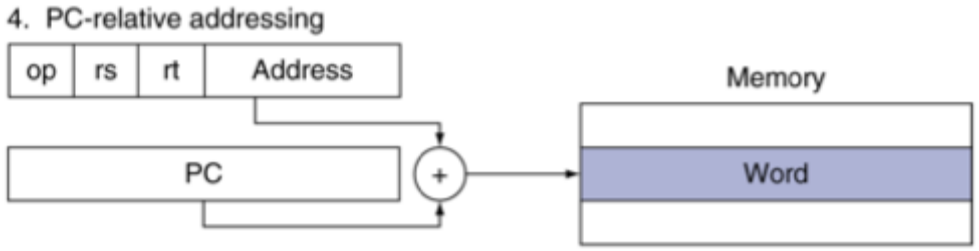
A. ALU

B. Registers and an ALU

C. Registers, ALU and Memory

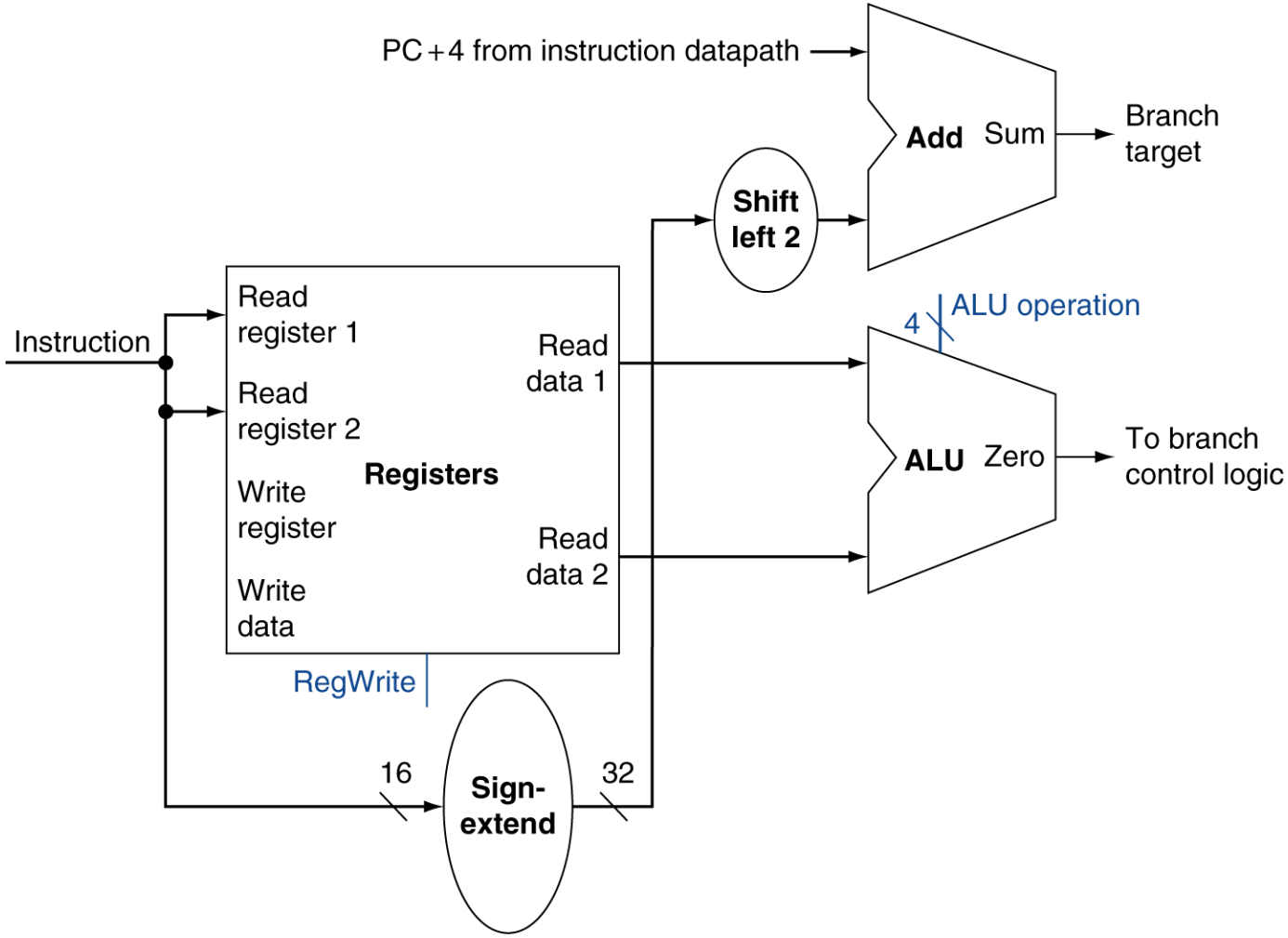
D. Registers, an ALU and an Adder

beq $t2, $t3, 0x4F35



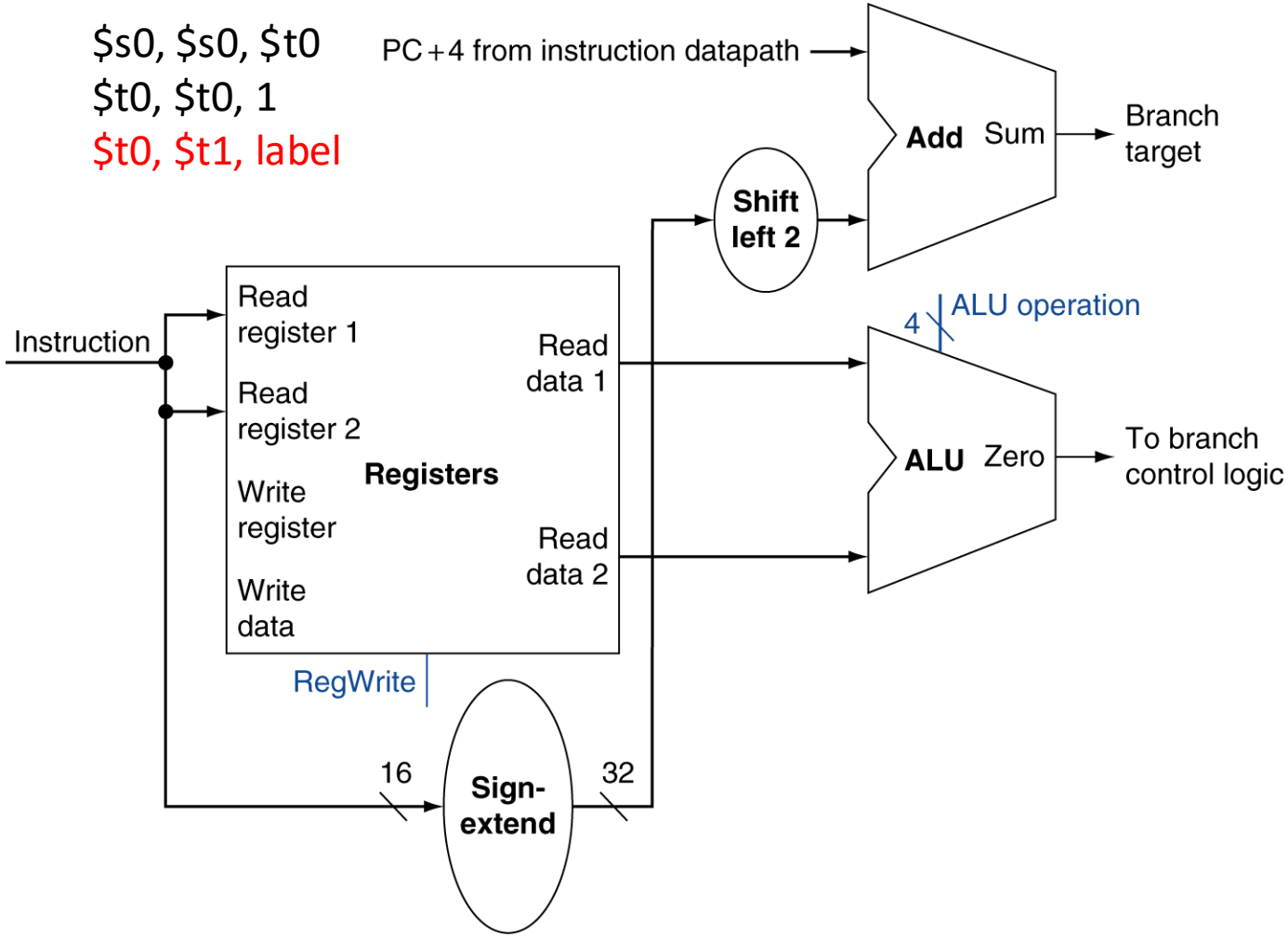
Read register operands
Compare operands
    Use ALU, subtract and check Zero output
Calculate target address
    Sign-extend offset
    Shift left 2 bits (word offset)
    Add to PC + 4
        Already calculated during instruction fetch

Branch Instructions



Branch Instructions

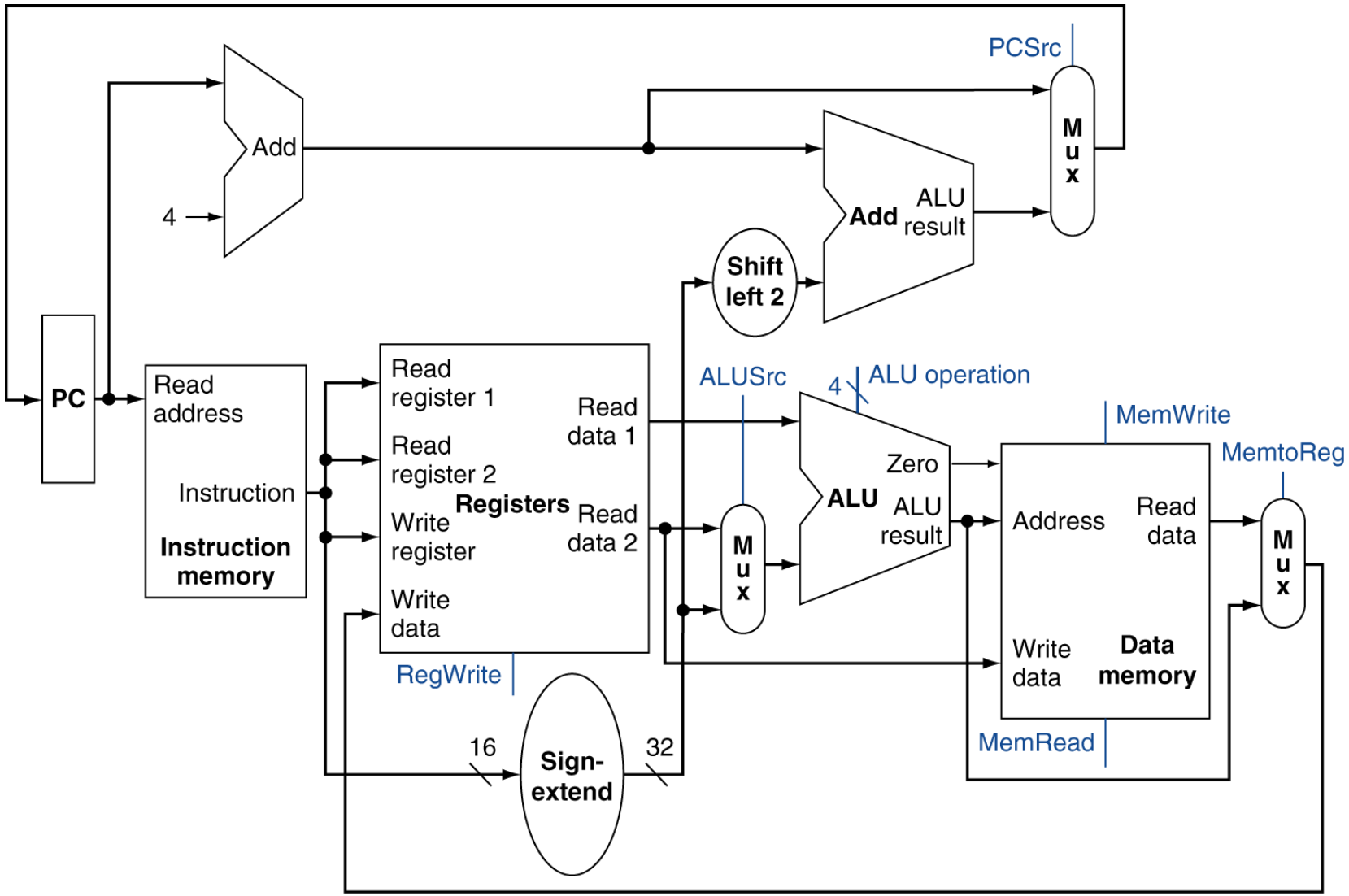
```
0x4045A130  label:  add     $s0, $s0, $t0
0x4045A134          addi    $t0, $t0, 1
0x4045A138          beq     $t0, $t1, label
```



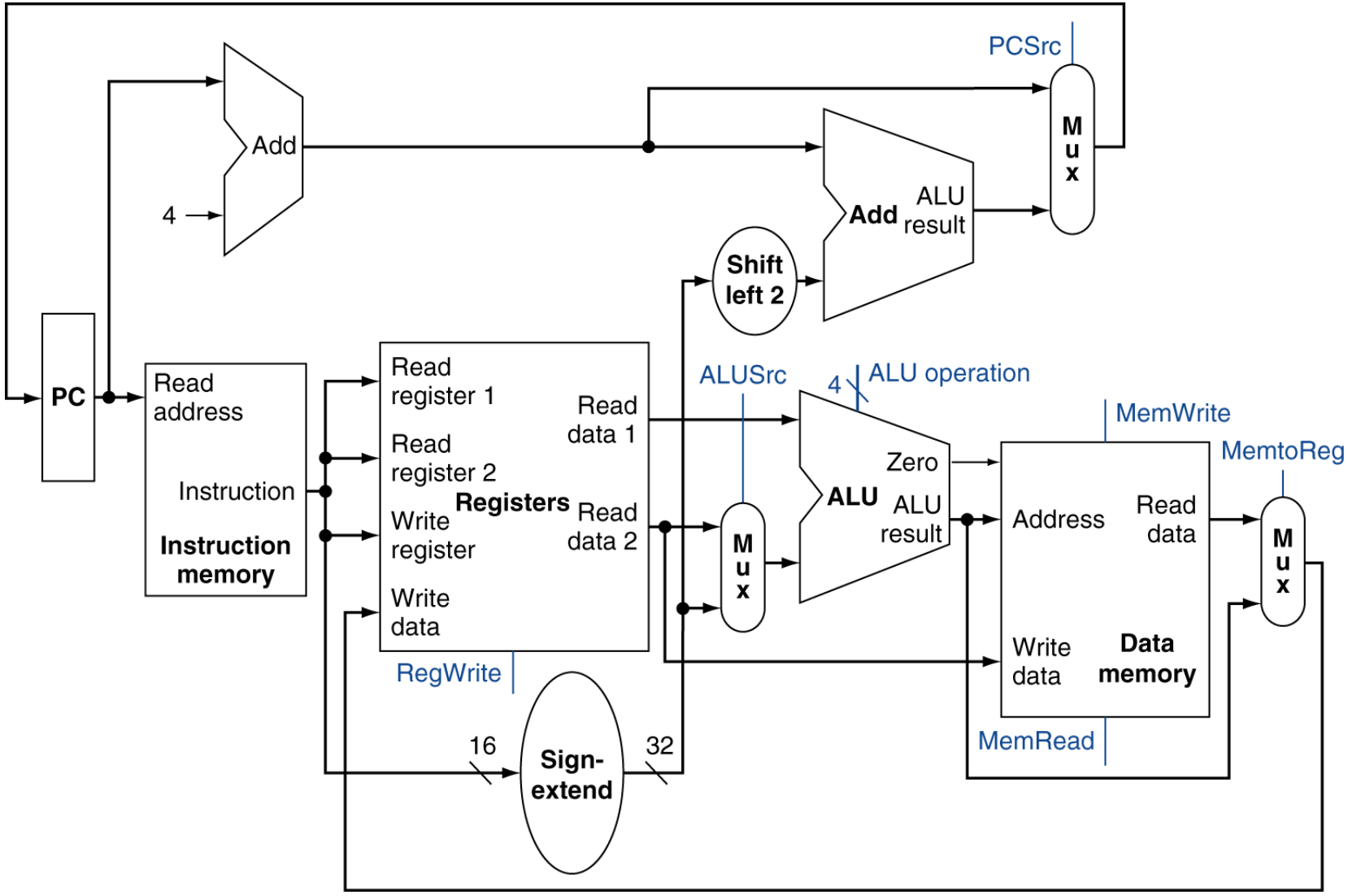
op = 0x04	rs = 8	rt = 9	imm = 0xFFFD

$t0 holds 5
$t1 holds 5

Datapath (still simplified a bit)



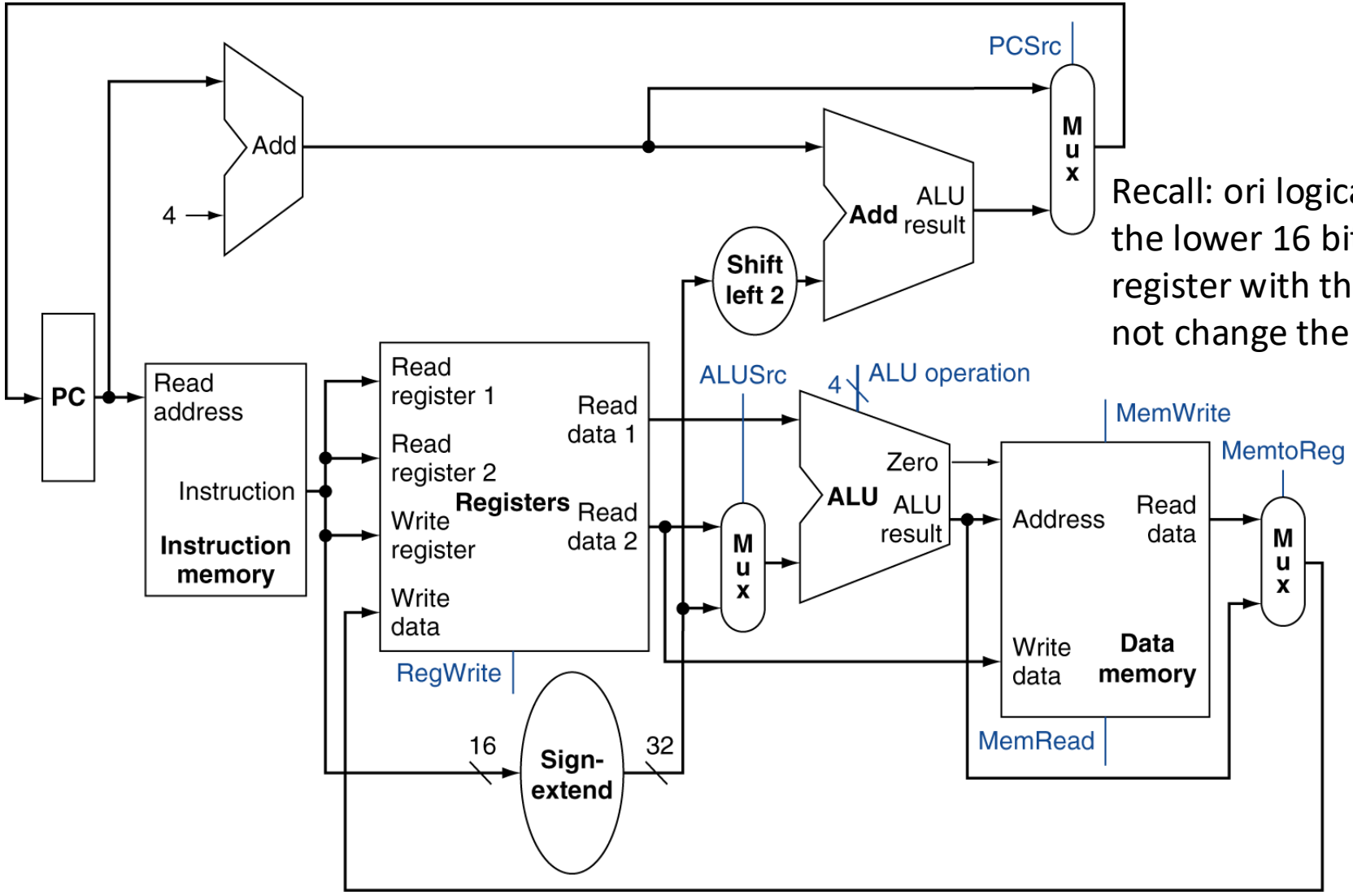
addi $t1, $t0, -1



$t0 holds 10

op = 0x08	rs = 8	rt = 9	imm = 0xFFFF

What do we need to add to support ori?

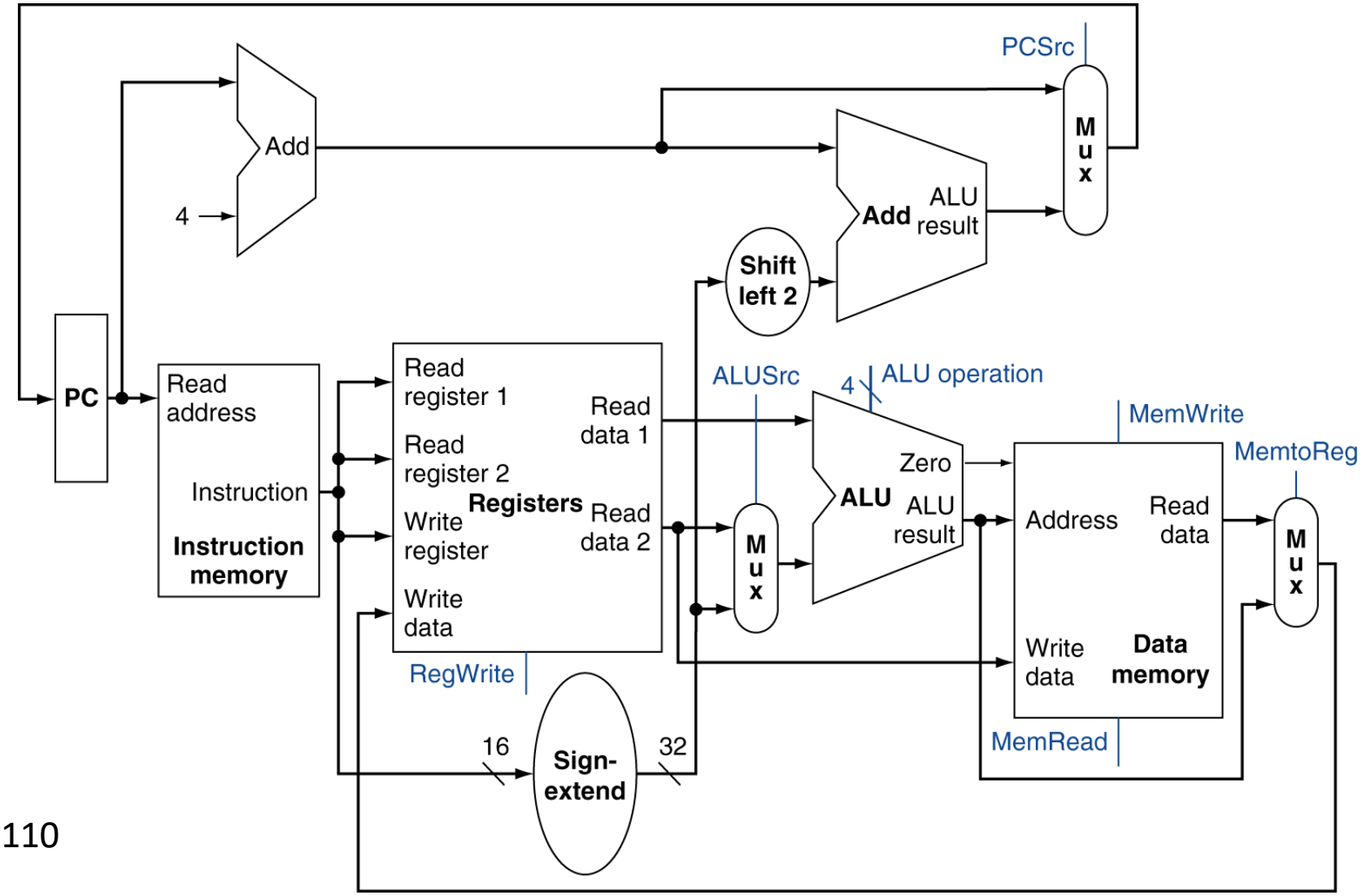


Recall: ori logically ors the lower 16 bits of the specified register with the immediate. It does not change the upper 16 bits.

ori $t0, $t1, -3
$t1 holds 5

op = 0x0D	rs = 9	rt = 8	imm = 0xFFFD

sw $t1, 8($t0)



$t0 holds 0x07AB8110

$t1 holds 5

op = 0x2B	rs = 8	rt = 9	imm = 0x0008

Composing the Elements

- Data path does an instruction in one clock cycle
  - Each data path element can only do one function at a time
  - Hence, we need separate instruction and data memories, ALU and adders, etc
- Use multiplexers where alternative data sources are used for different instructions

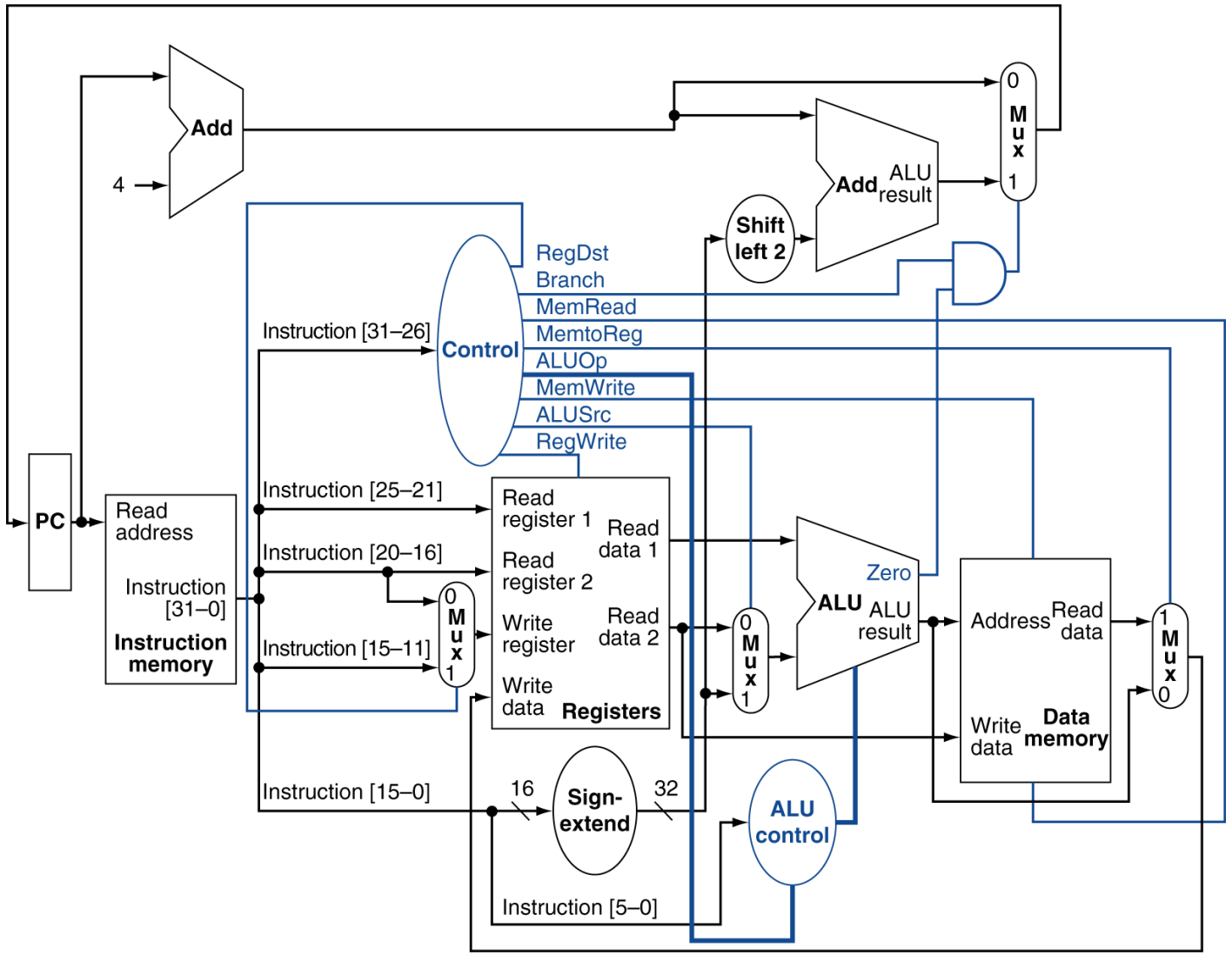
Key Points

- CPU is just a collection of state and combinational logic
- We just designed a very rich processor, at least in terms of functionality
- ET = IC * CPI * Cycle Time

Control Path

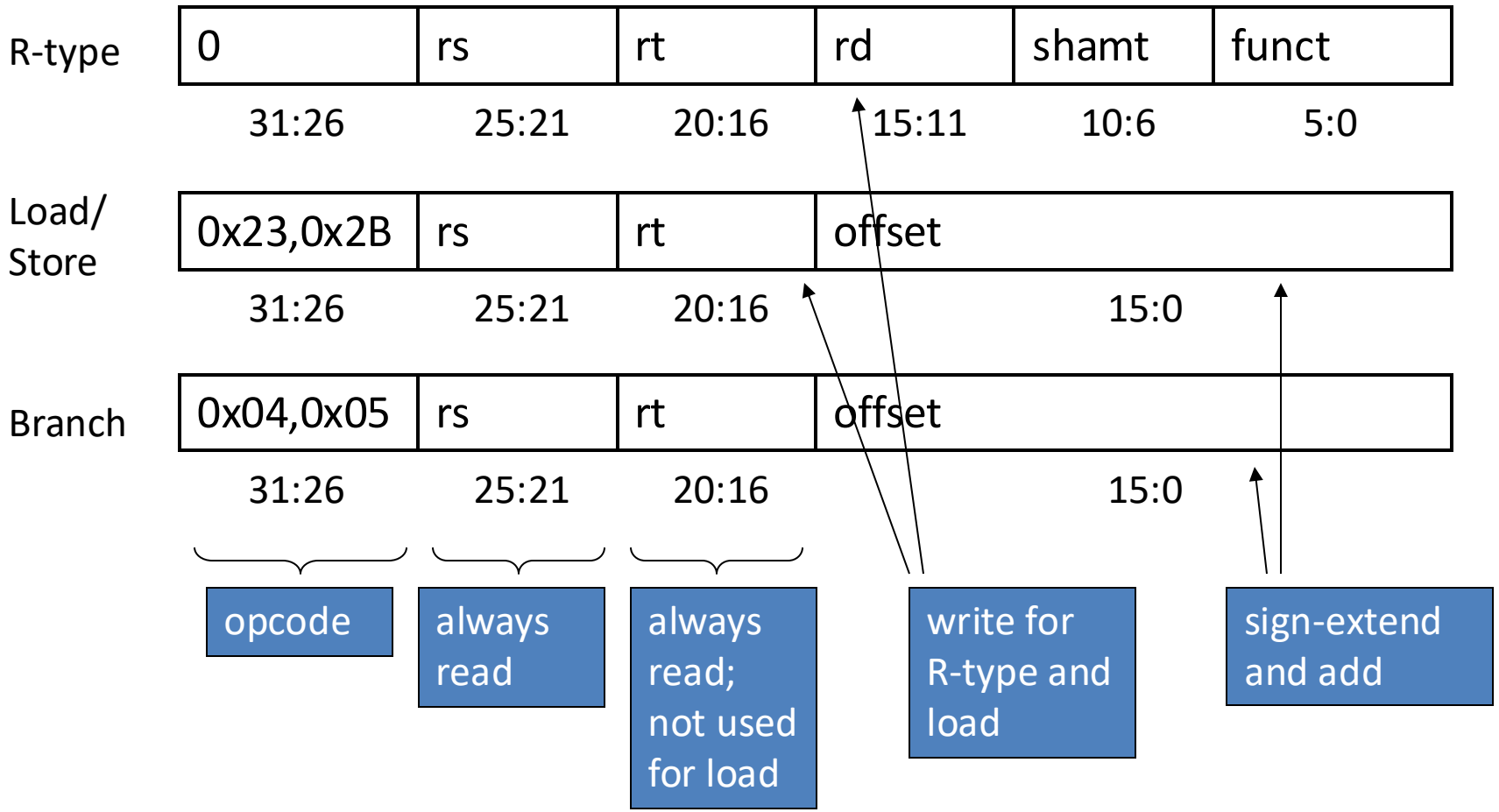
- Our datapath is complicated, and we don’t use each element every time
- How do we know which elements to use?

Datapath With Control



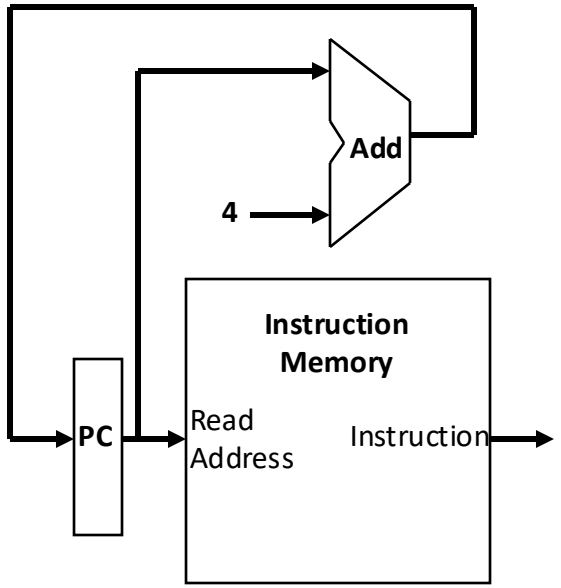
The Main Control Unit

Control signals derived from instruction

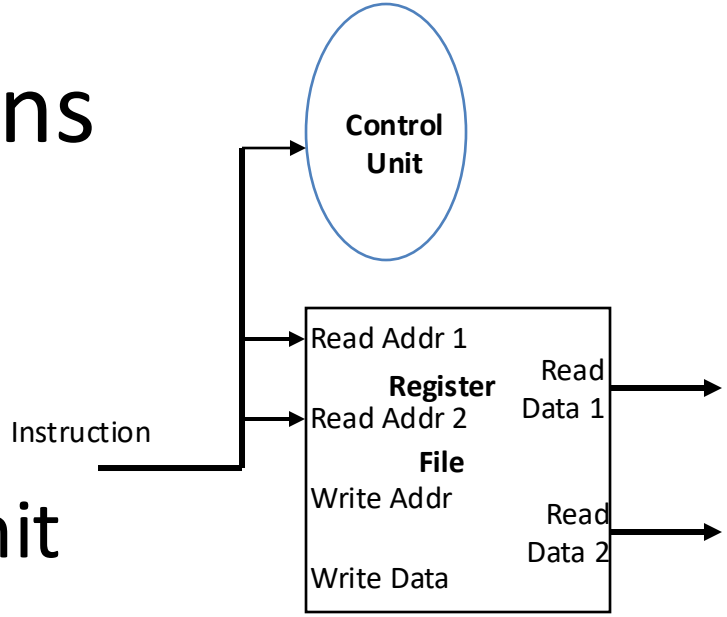


Fetching Instructions

- Read instruction from Instruction Memory
- Updating PC value to address of next (sequential) instruction
- PC is updated every clock cycle, so it does not need an explicit write control signal just a clock signal
- Read from memory each time, so we don’t need an explicit control signal



Decoding Instructions



- Send fetched instruction's opcode to the main control unit

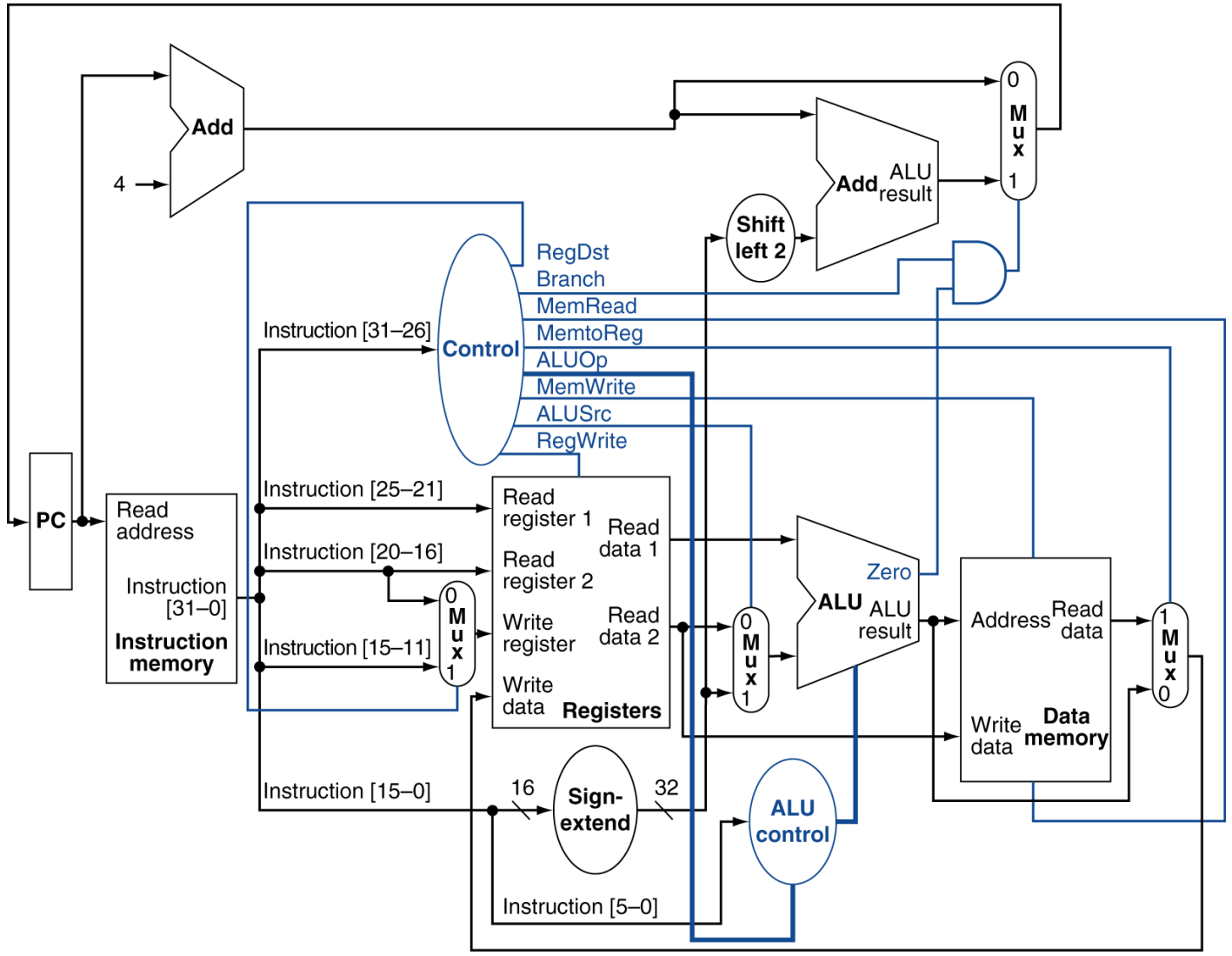
R-type	0	rs	rt	rd	shamt	funct
	31:26	25:21	20:16	15:11	10:6	5:0
Load/ Store	0x23,0x2B	rs	rt	address		
	31:26	25:21	20:16	15:0		

- Read two values from the Register File
- Register File addresses are contained in the instruction

After decode

After reading opcode

- Produce most control signals
- Includes the ALUOp control signal—which goes to the ALU control unit—and the ALUSrc control signal which selects the ALU's second operand



ALU Control Unit

- Combinational logic (the main control unit) derives 2-bit ALUOp signal from opcode
- ALU Control Unit takes ALUOp and instruction funct field as inputs and derives a 4-bit ALU control signal

opcode	ALUOp	Operation	ALU function
lw	00	load word	add
sw	00	store word	add
beq	01	branch equal	subtract
R-type	10	arithmetic/logic	depends on funct

For load/store, our ALU operation will be

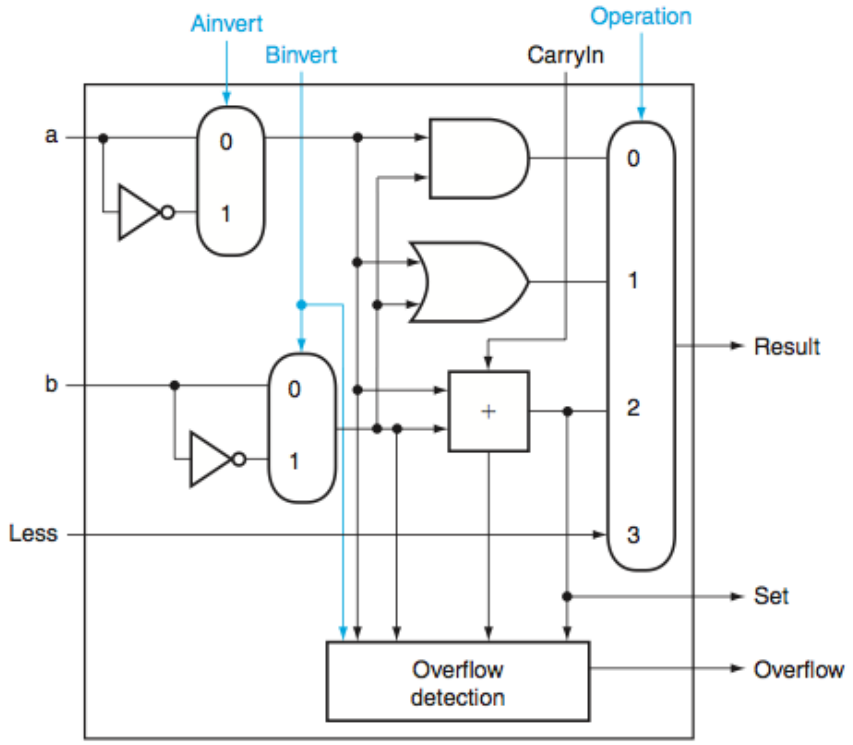
A. Add

B. And

C. Set less than

D. Subtract

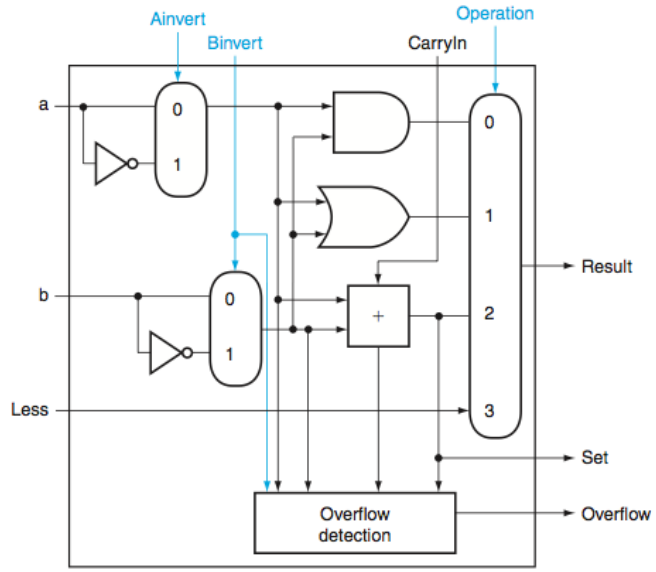
E. None of the above



lw $t0, 4($t1)

ALU Control

- ALU used for
  - Load/Store: op = add
  - Branch: op = subtract
  - R-type: op depends on funct field



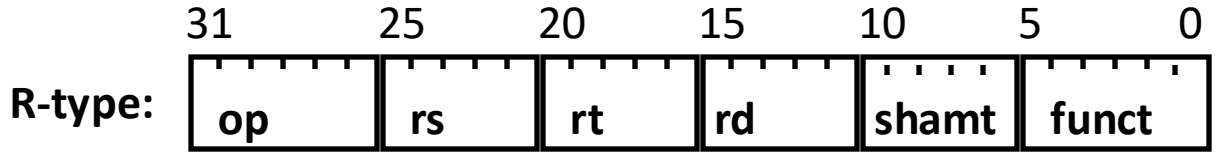
ALU control	Function	Ainvert	Binvert/CarryIn0	Operation
0000	AND	0	0	00
0001	OR	0	0	01
0010	add	0	0	10
0110	subtract	0	1	10
0111	set-on-less-than	0	1	11
1100	NOR	1	1	00

ALU Control

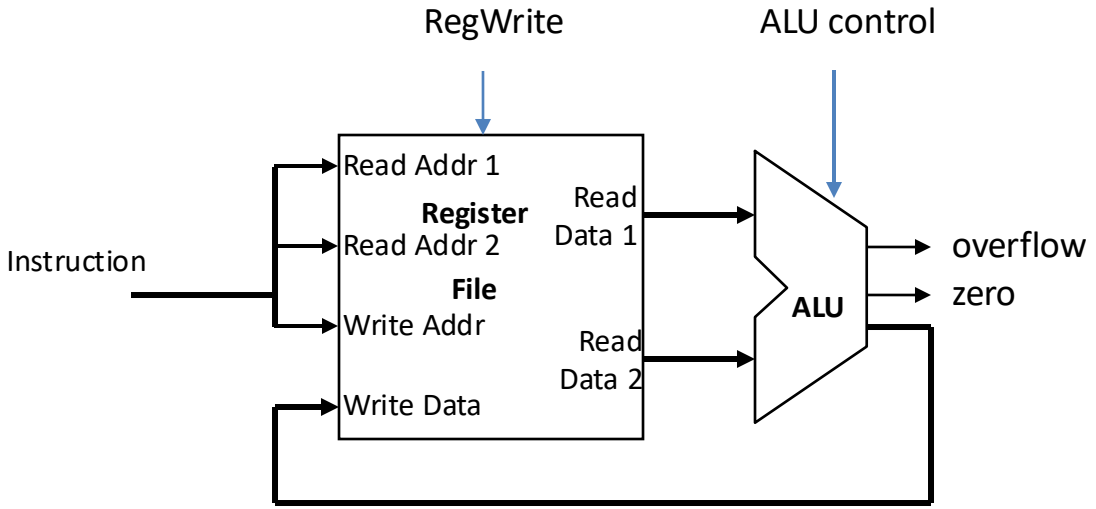
opcode	Instruction	ALUOp input	funct input	ALU function	ALU control output
lw	load word	00	XXXXXX	add	0010
sw	store word	00	XXXXXX	add	0010
beq	branch equal	01	XXXXXX	subtract	0110
R-type	add	10	100000	add	0010
	subtract		100010	subtract	0110
	AND		100100	AND	0000
	OR		100101	OR	0001
	set-on-less-than		101010	set-on-less-than	0111

Executing R Format Operations

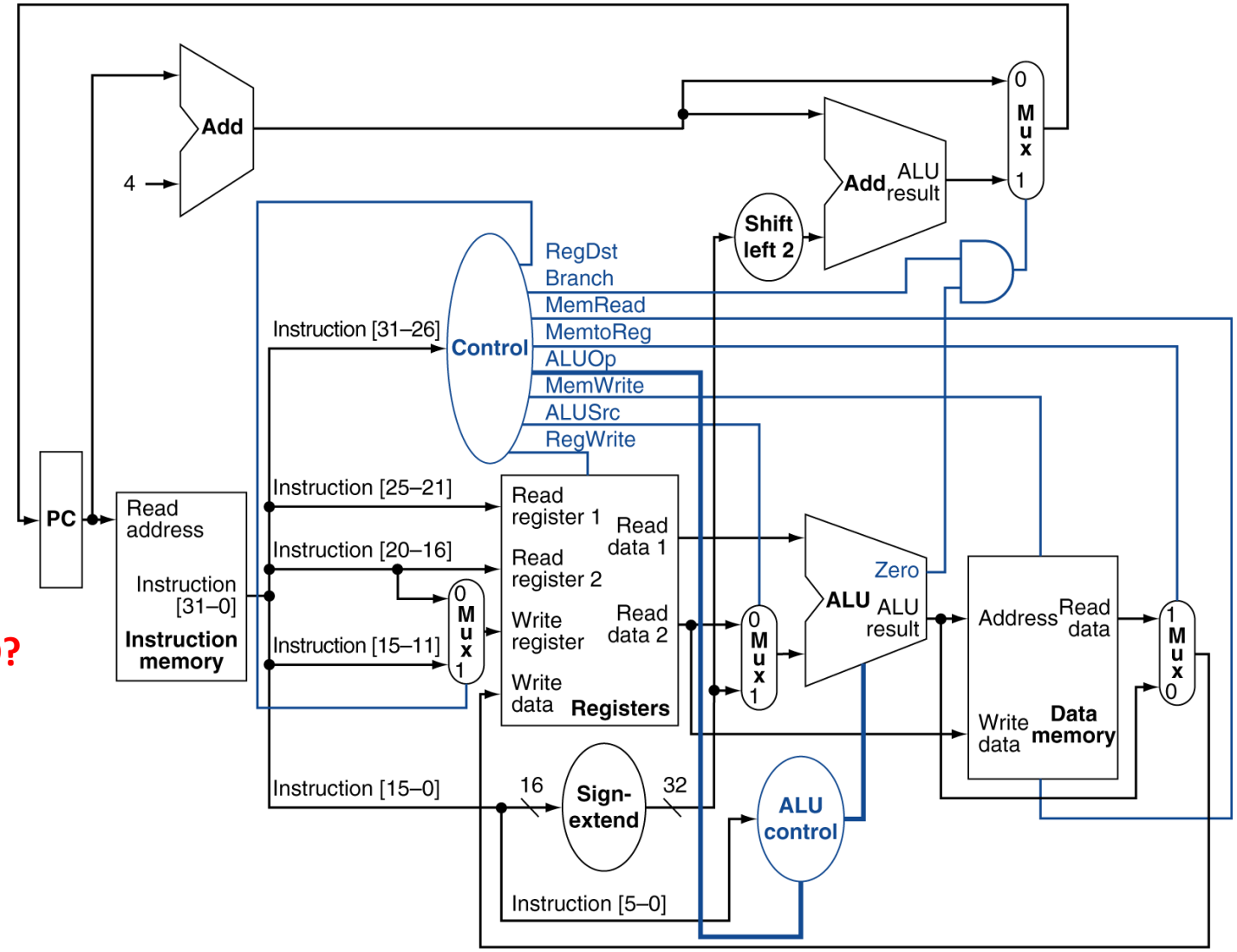
- R format operations (`add, sub, slt, and, or`)



  - perform operation (specified by funct) on values in rs and rt
  - store the result back into the Register File (into location rd)



Note that Register File is not written every cycle (e.g., **`sw`**), so we need an explicit write control signal for the Register File



instruction control signals for ADD?

Select	RegDst	MemToReg
A	0	X
B	1	X
C	0	1
D	1	0
E	None of the above	

R-type

0	rs	rt	rd	shamt	funct
31:26	25:21	20:16	15:11	10:6	5:0

Reading

- Next lecture: Control Path
  - Section 5.4