UMBC
AN HONORS UNIVERSITY

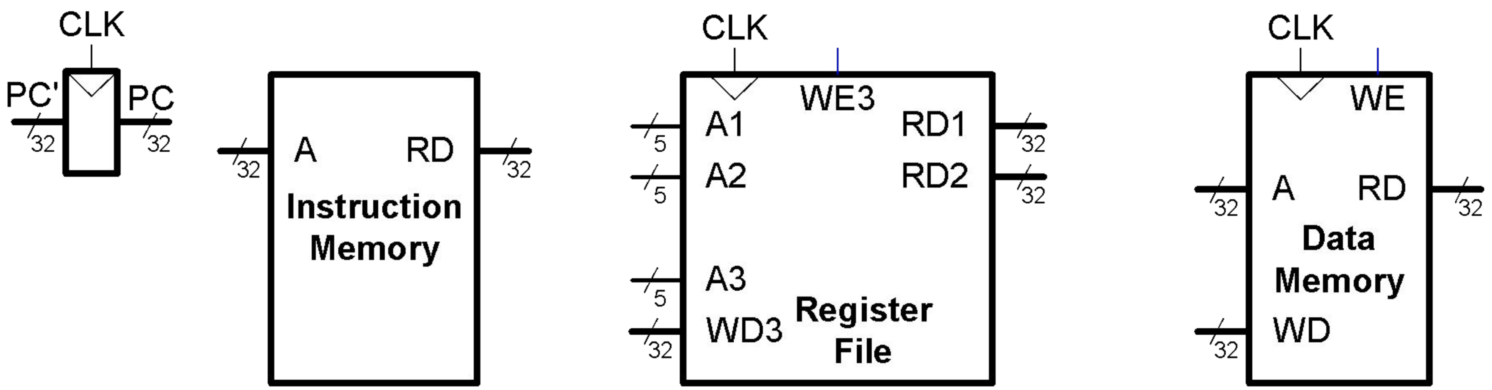
CMSC 411 | Lectures 14

Single Cycle Datapath and Control

Part II: Building a **Single-Cycle** Datapath

Ergun Simsek

MIPS State Elements



- ***Program counter:***

  32-bit register

- ***Instruction memory:***

  Takes input 32-bit address A and reads the 32-bit data (i.e., instruction) from that address to the read data output RD.

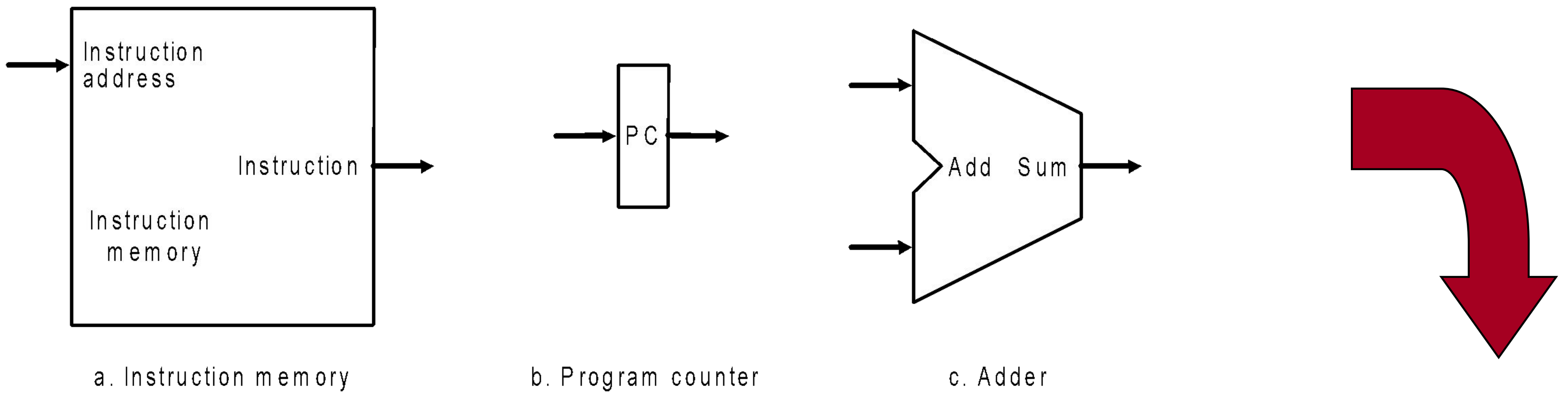
- ***Register file:***

  The 32-element, 32-bit register file has 2 read ports and 1 write port

- ***Data memory:***

  Has a single read/write port. If the write enable, WE, is 1, it writes data WD into address A on the rising edge of the clock. If the write enable is 0, it reads address A onto RD.

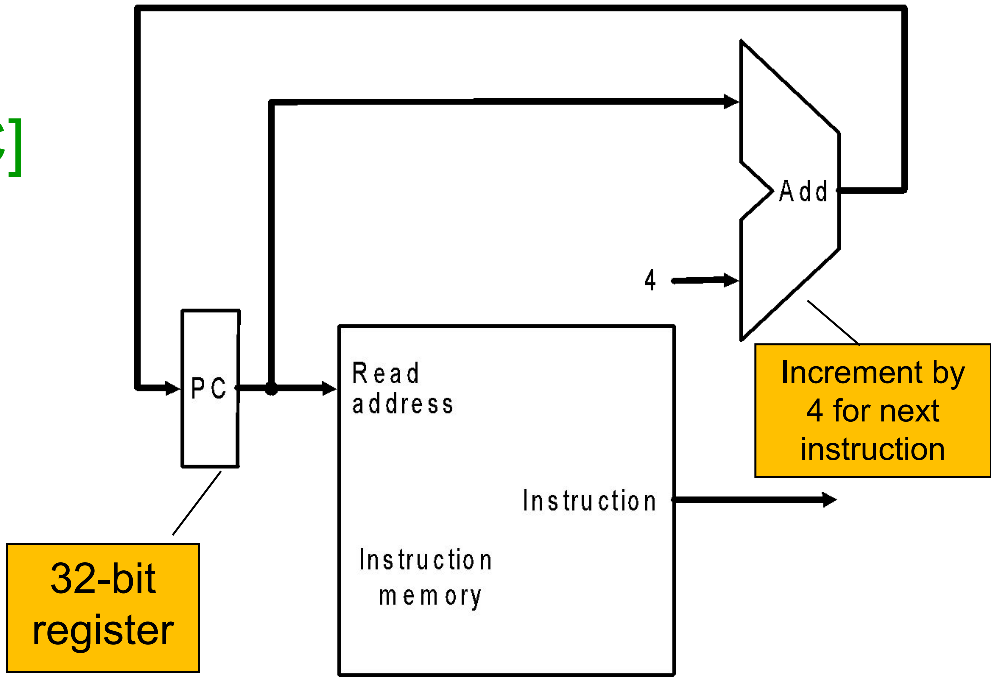
Instruction Fetch Unit



- Fetch the Instruction: mem[PC]
- Update the program counter:
  - Sequential Code:

    PC <- PC + 4
  - Branch and Jump:

    PC <- "something else"



R-Format Instructions

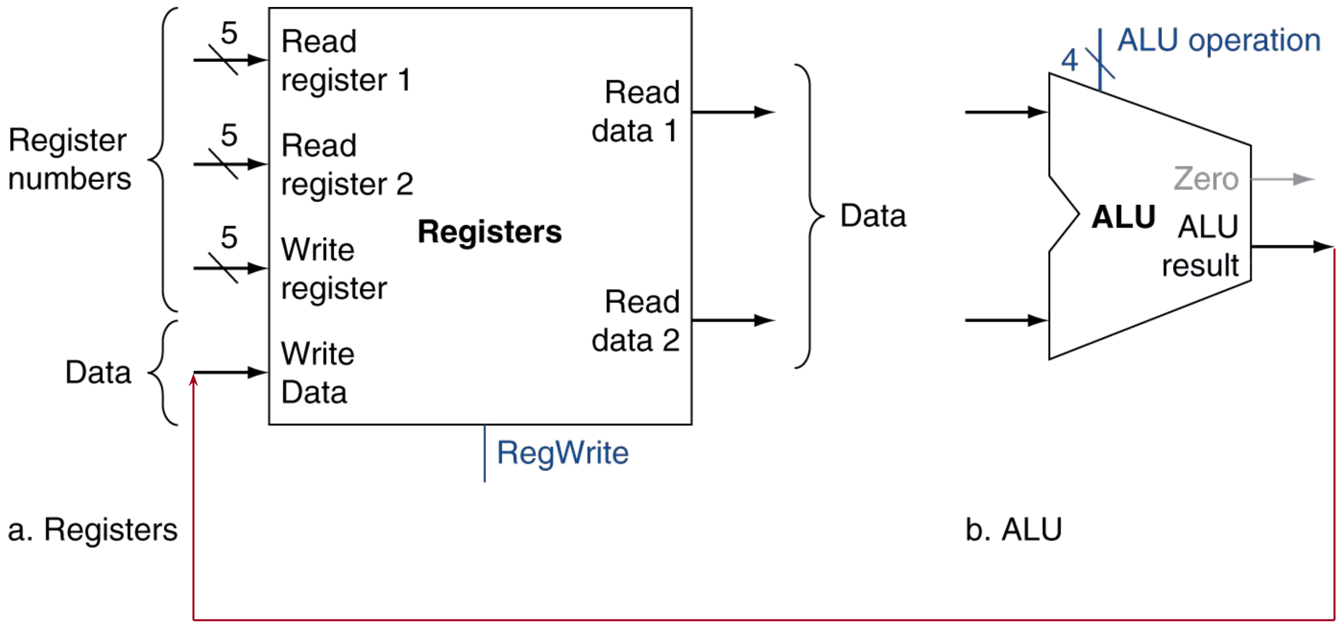
31 – 26	26 – 21	21 – 16	16 – 11	11 – 6	6 – 0
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Read two register operands

Perform arithmetic/logical operation

Write register result

R[rd] = R[rs] + R[rt]



Load/Store Instructions

31–26	25–21	20–16	15–0
op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits

R[rt] = M[R[rs]+SignExtImm]

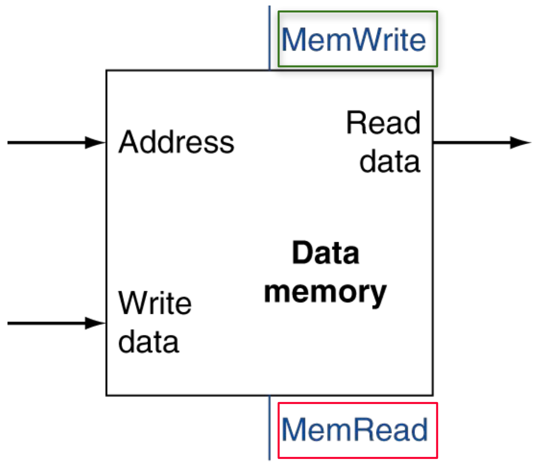
Read register operands

Calculate address using 16-bit offset

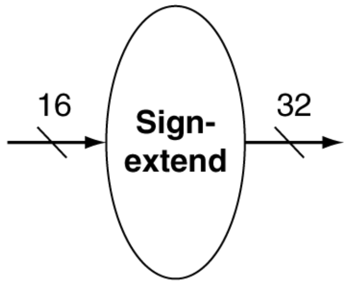
- Use ALU, but sign-extend offset

Load: Read memory and update register

Store: Write register value to memory



a. Data memory unit



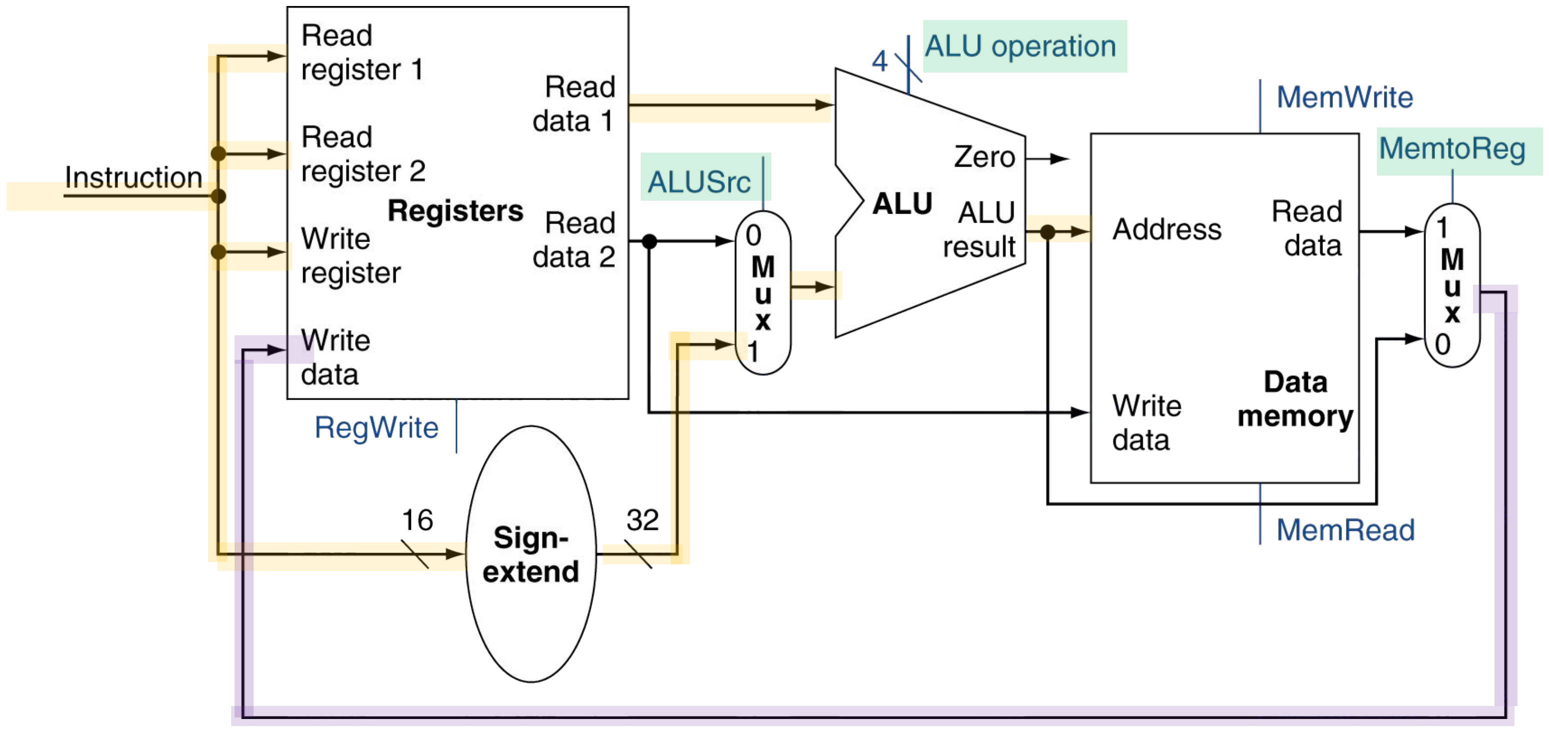
b. Sign extension unit

R-Type/Load/Store Datapath

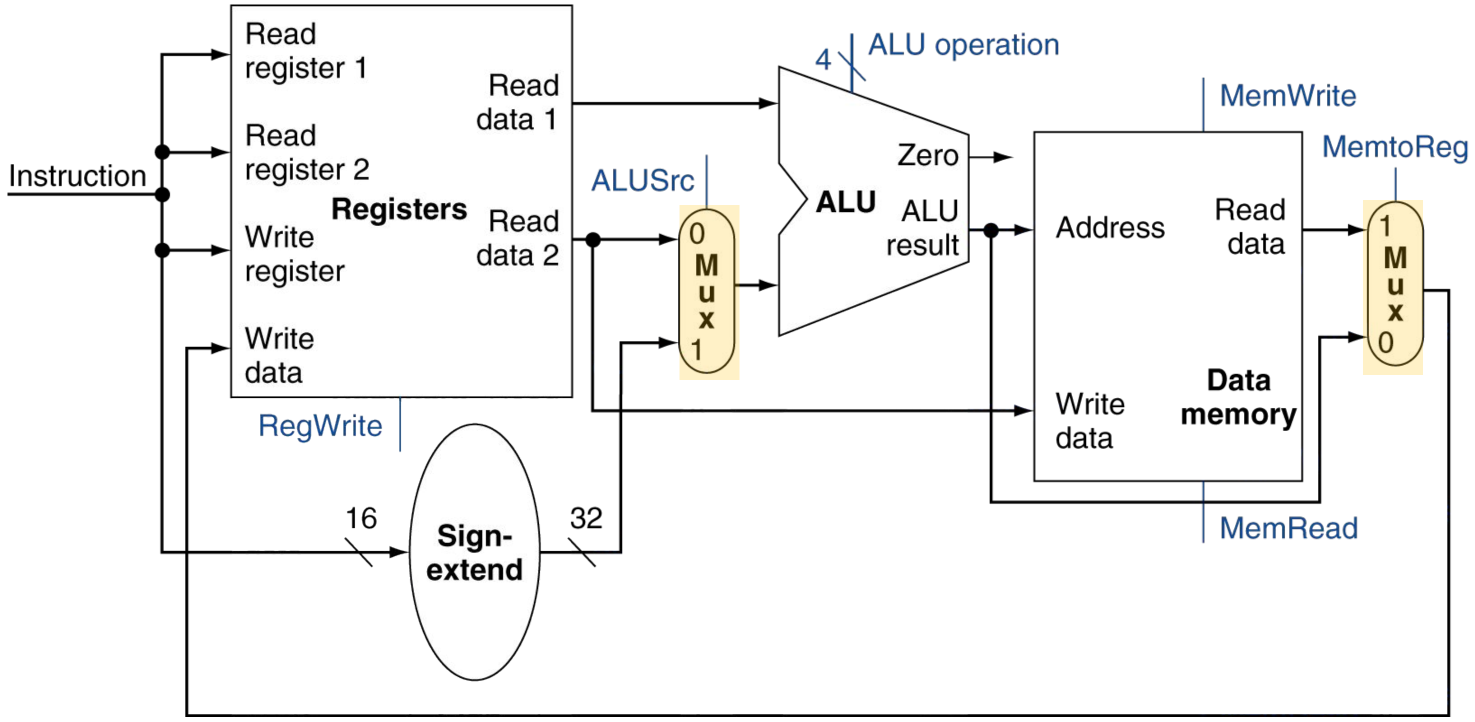
31 26 21 16 0

op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits

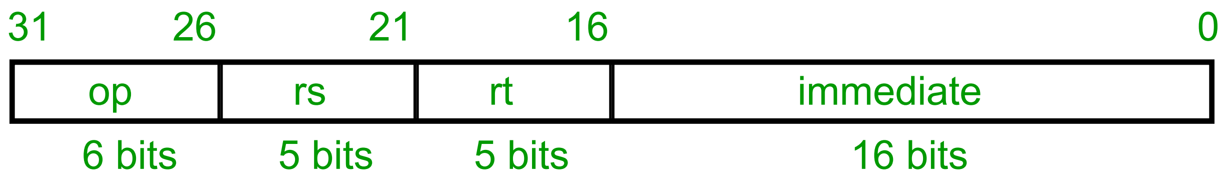
R[rt] = M[R[rs]+SignExtImm]



R-Type/Load/Store Datapath



Branch Instructions



beq rs, rt, imm16

```
if(R[rs]==R[rt])
    PC=PC+4+BranchAddr
```

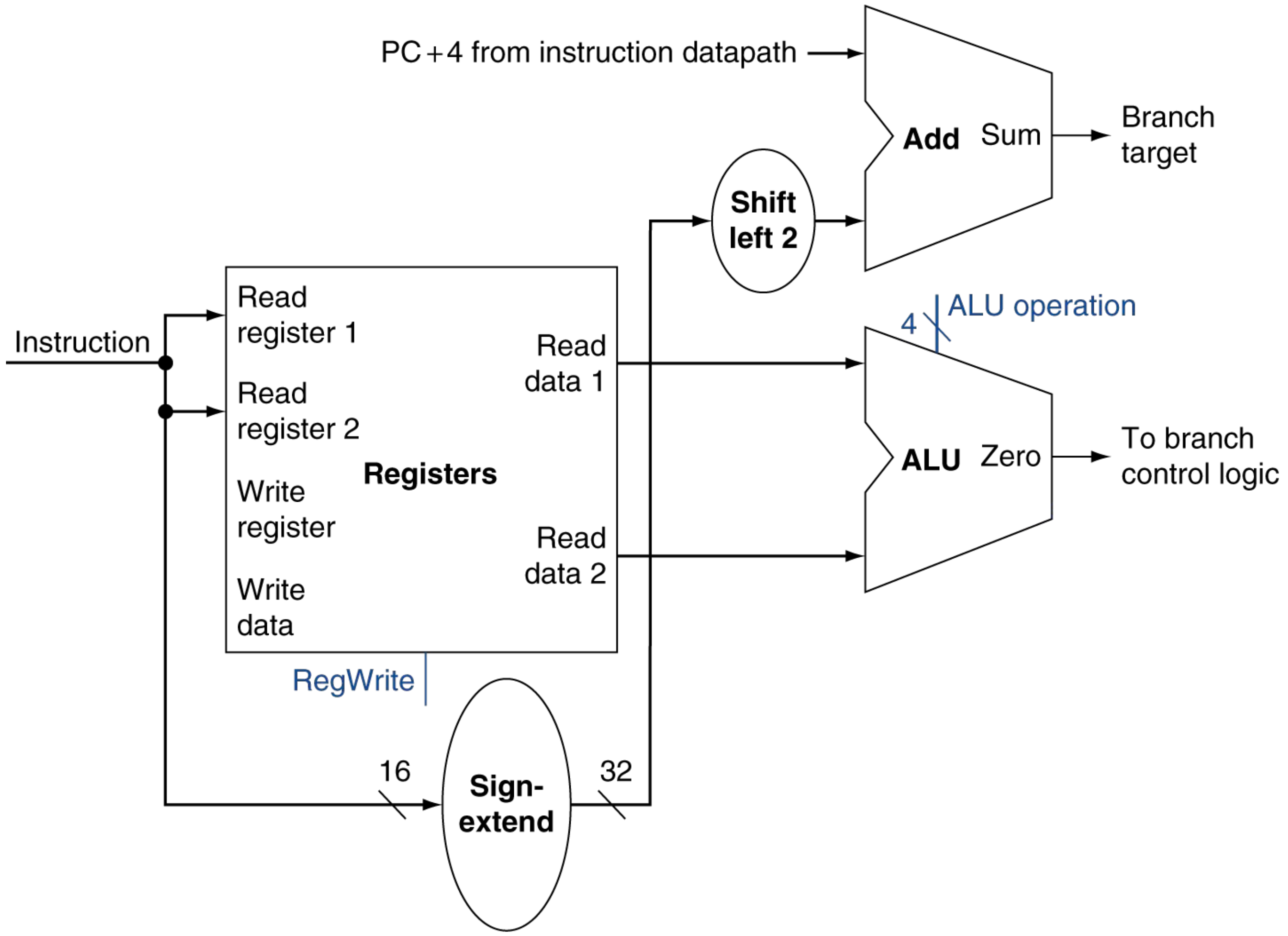
- Read register operands
- Compare operands
  - Use ALU, subtract and check Zero output
- Calculate target address
  - Sign-extend displacement
  - Shift left 2 places (word displacement)
  - Add to PC + 4
    - Already calculated by instruction fetch

Branch Instructions

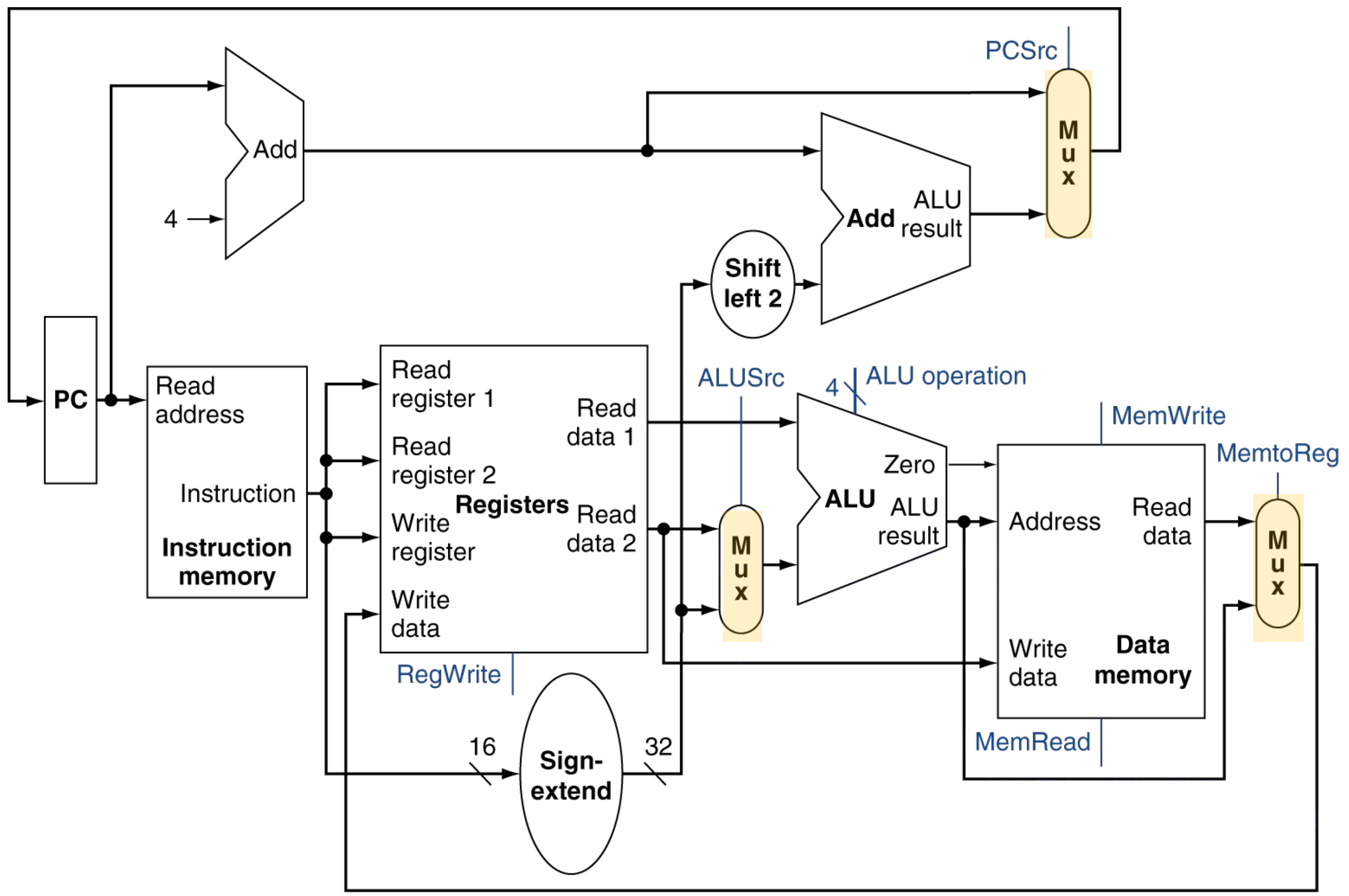
31–26	25–21	20–16	15–0
op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits

beq rs, rt, imm16

if(R[rs]==R[rt])
    PC=PC+4+BranchAddr



Full Datapath



How can we control (choose) the ALU and MUX operations??

Let's start with ALU control…

ALU Control

ALU used for

- Load/Store: Function = add
- Branch: Function = subtract
- R-type: Function depends on funct field

ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR

⬆

We can generate these 4-bit ALU control input using a small control unit that has as inputs the **function** field of the instruction and a **2-bit control** field, which we call ALUOp.

ALU Control

Assume 2-bit ALUOp derived from opcode

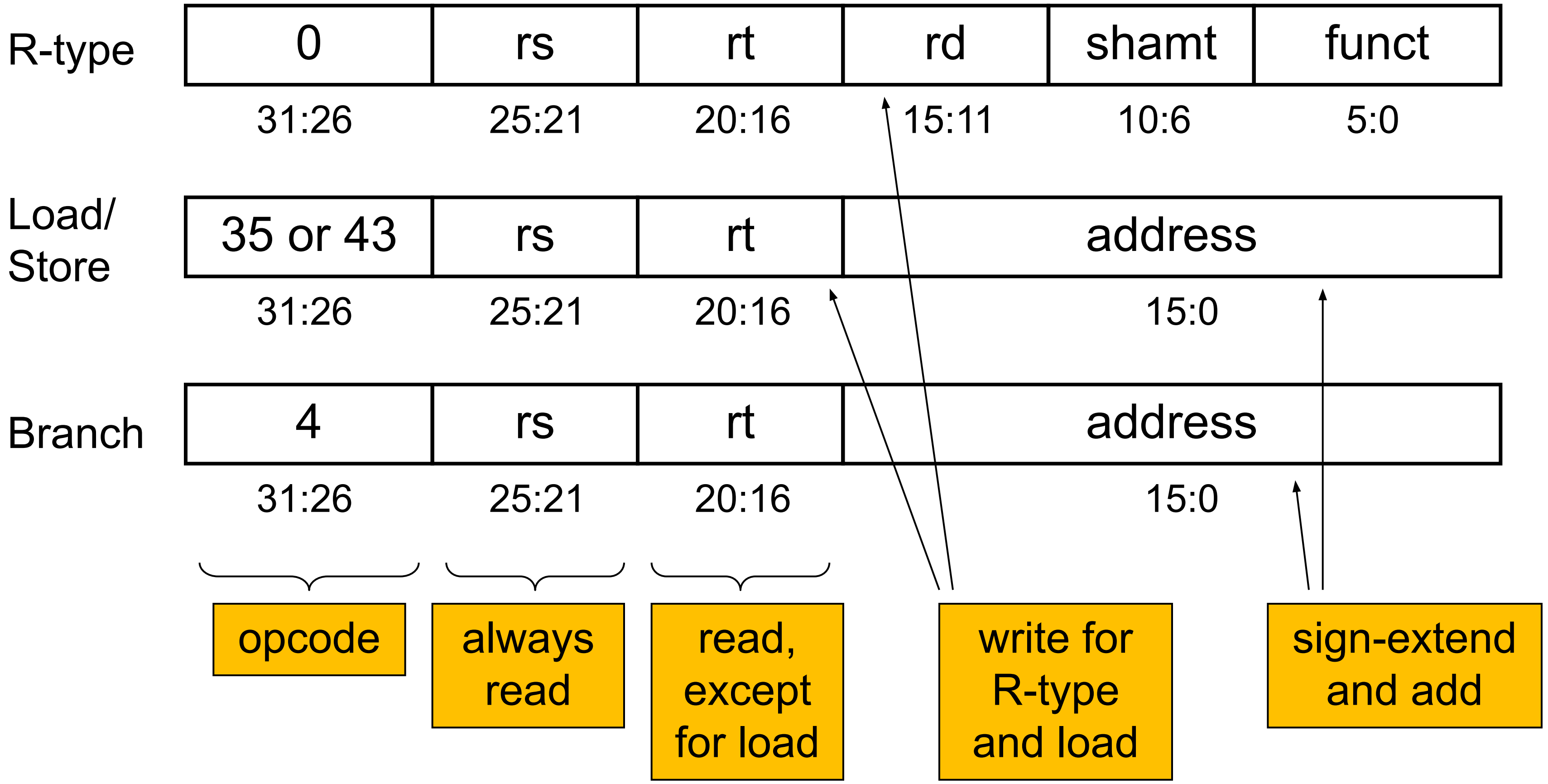
- Combinational logic derives ALU control

opcode	ALUOp	Operation	funct	ALU function	ALU control
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	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

We can easily generate a truth table which yields ALU control bits from 2-bit ALUOp and 6-bit funct code (see Fig. 4.13)

The Main Control Unit

Control signals derived from instruction



Control Signals (1-4)

Branch — Asserted for branch instructions to select the branch target address as the next instruction address.

Jump — Asserted for jump instructions to select the jump target address

ALUSrc — Selects the second ALU input from either rt or the sign extended immediate field of the instruction.

ALUOp — Determines the operation performed by the ALU. It has three values: "add", "subtract", or "decoded from function field"

Control Signals (5-9)

MemRead — Asserted for load instructions, tells memory to do a read.

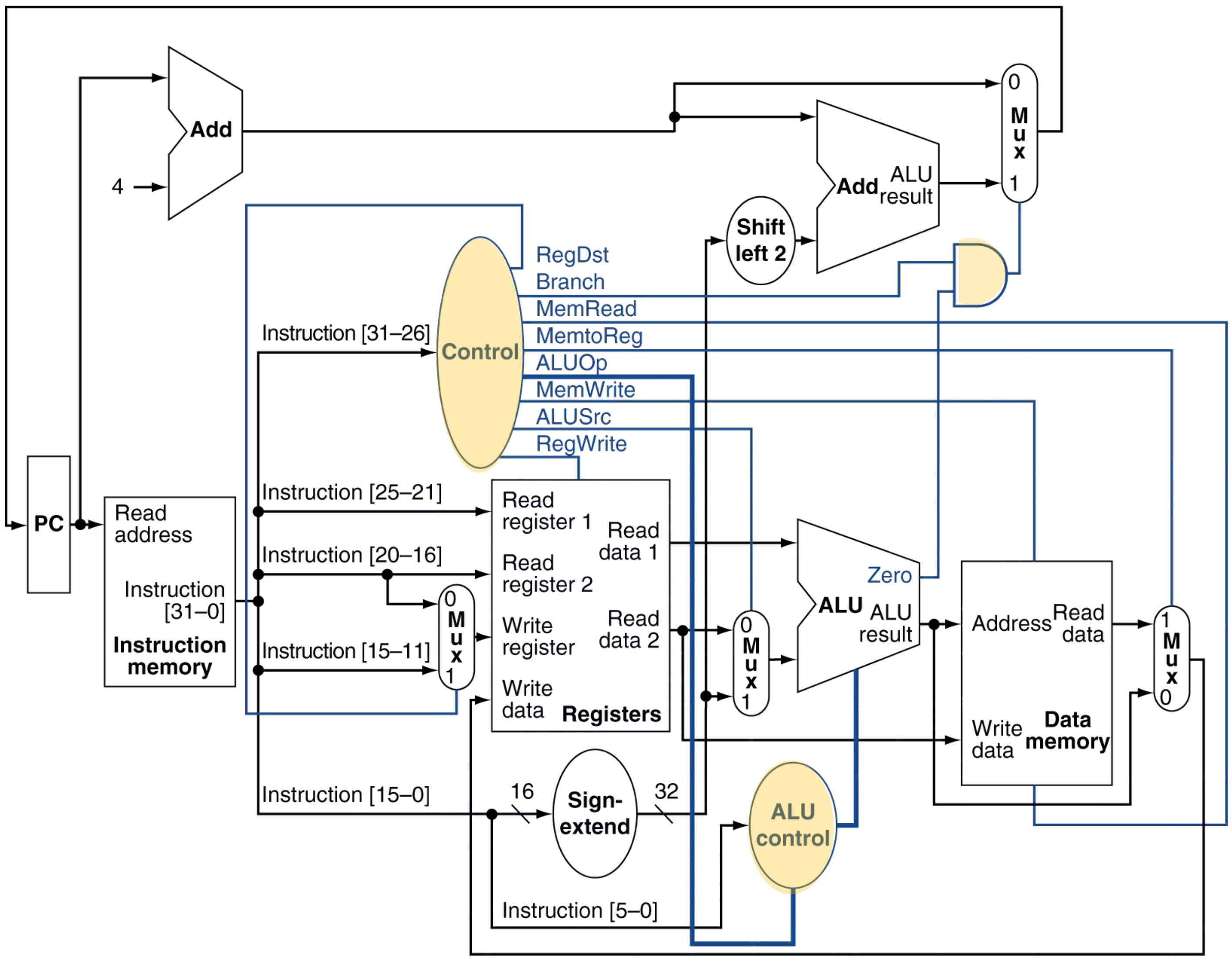
MemWrite — Asserted for store instructions, tells memory to do a write.

RegWrite — Asserted if a result needs to be written to a register.

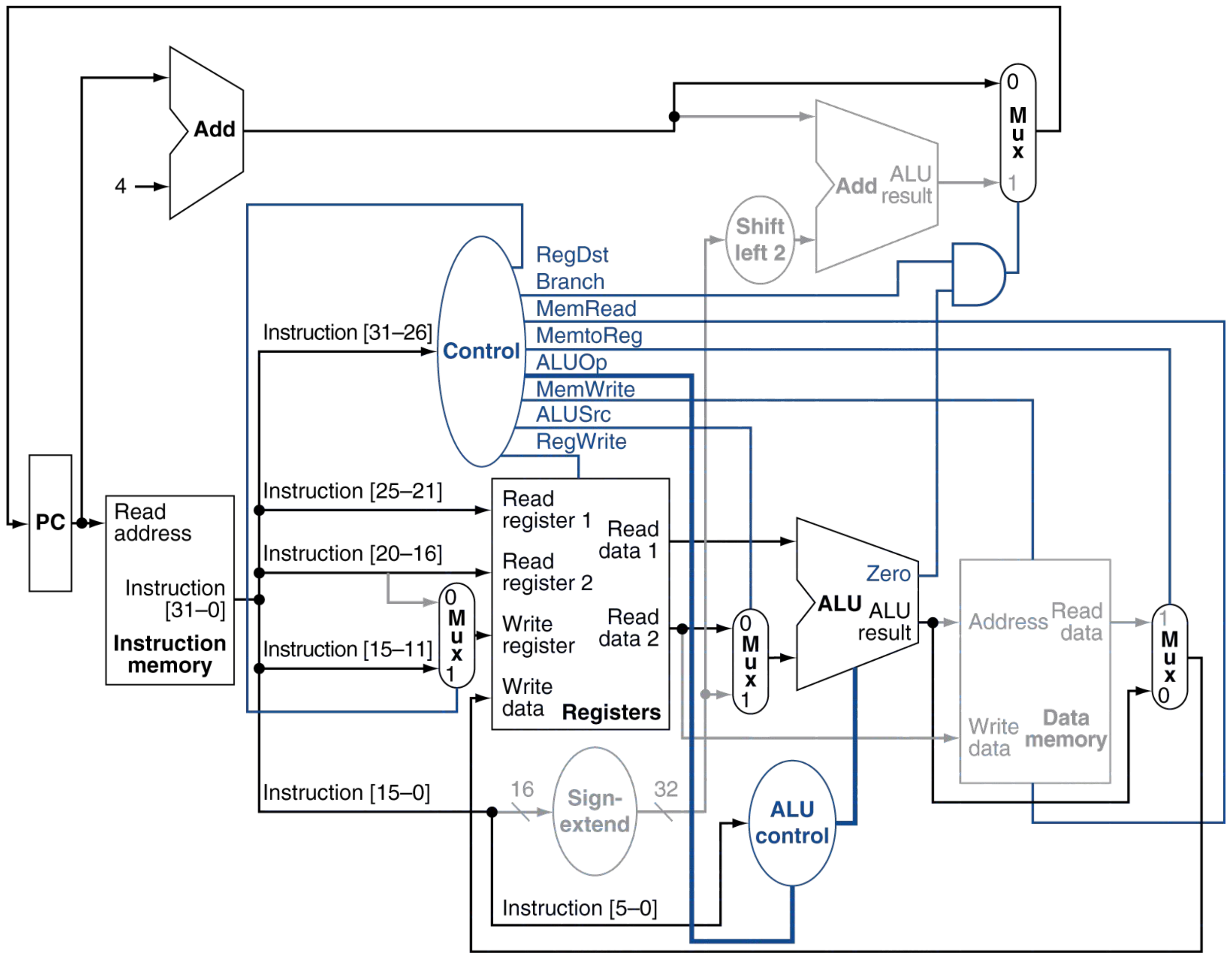
RegDst — Selects the destination register as either rd (Rtype instructions) or rt (Itype instructions).

MemtoReg — Selects the source value for the register write as either the ALU result or memory.

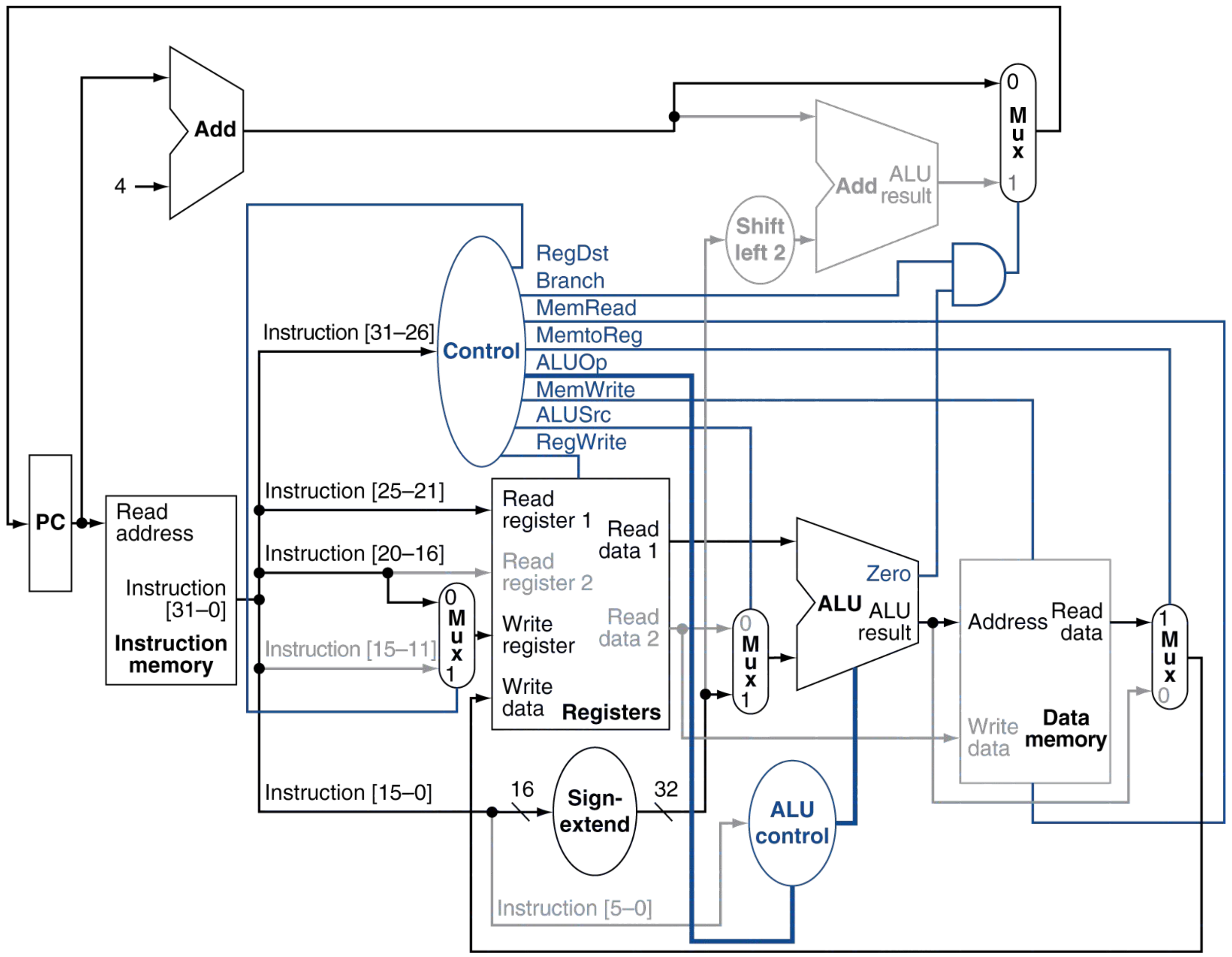
Datapath With Control



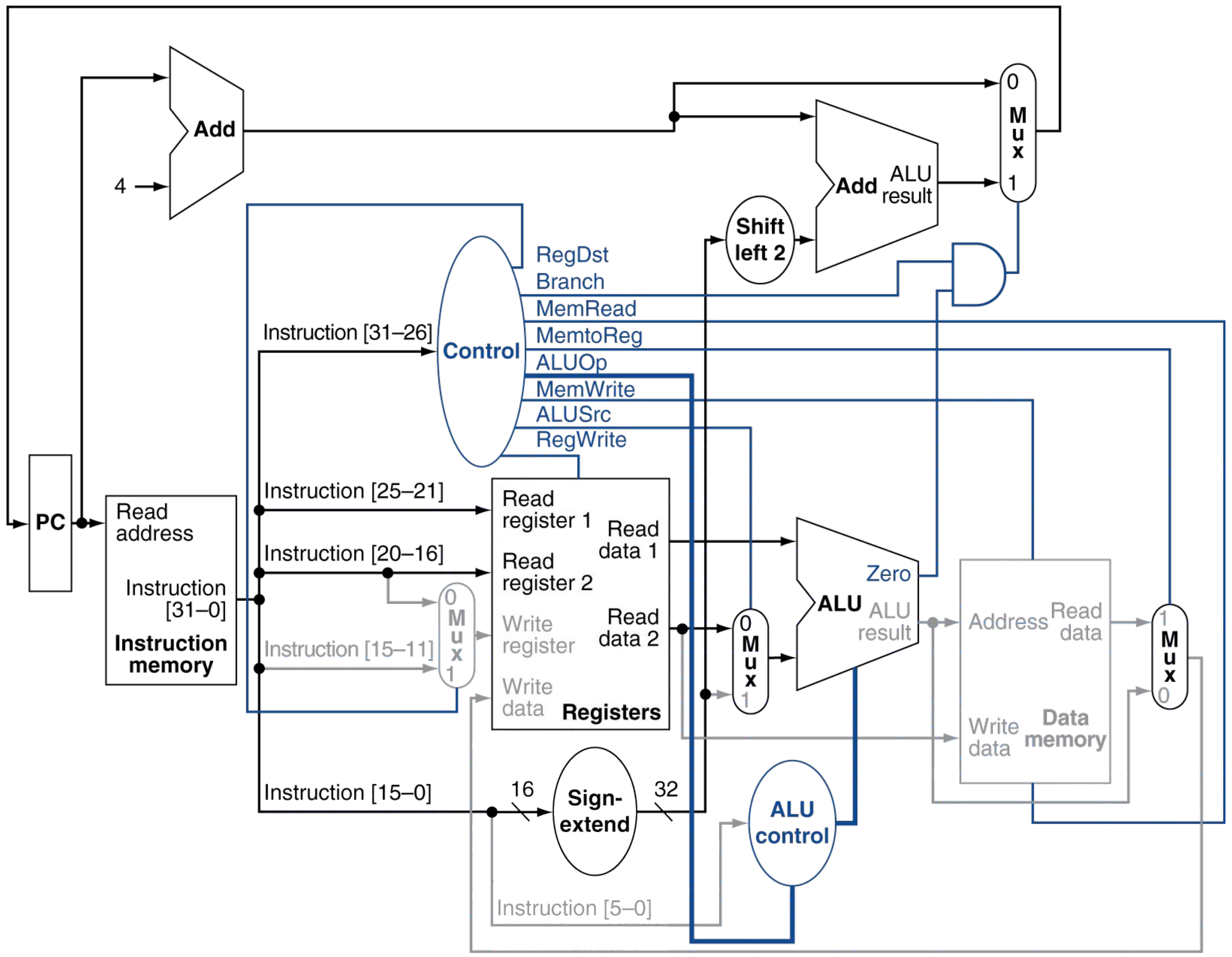
R-Type Instructions



Load


Add
4
Add
ALU result
Shift left 2
M u x
0
1
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [31–26]
Control
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
16
32
Sign-extend
Instruction [5–0]
ALU control
Zero
ALU
ALU result
Address
Read data
Write data
Data memory

Branch-on-Equal Instruction



Implementing Jumps

Jump	2	address
	31:26	25:0

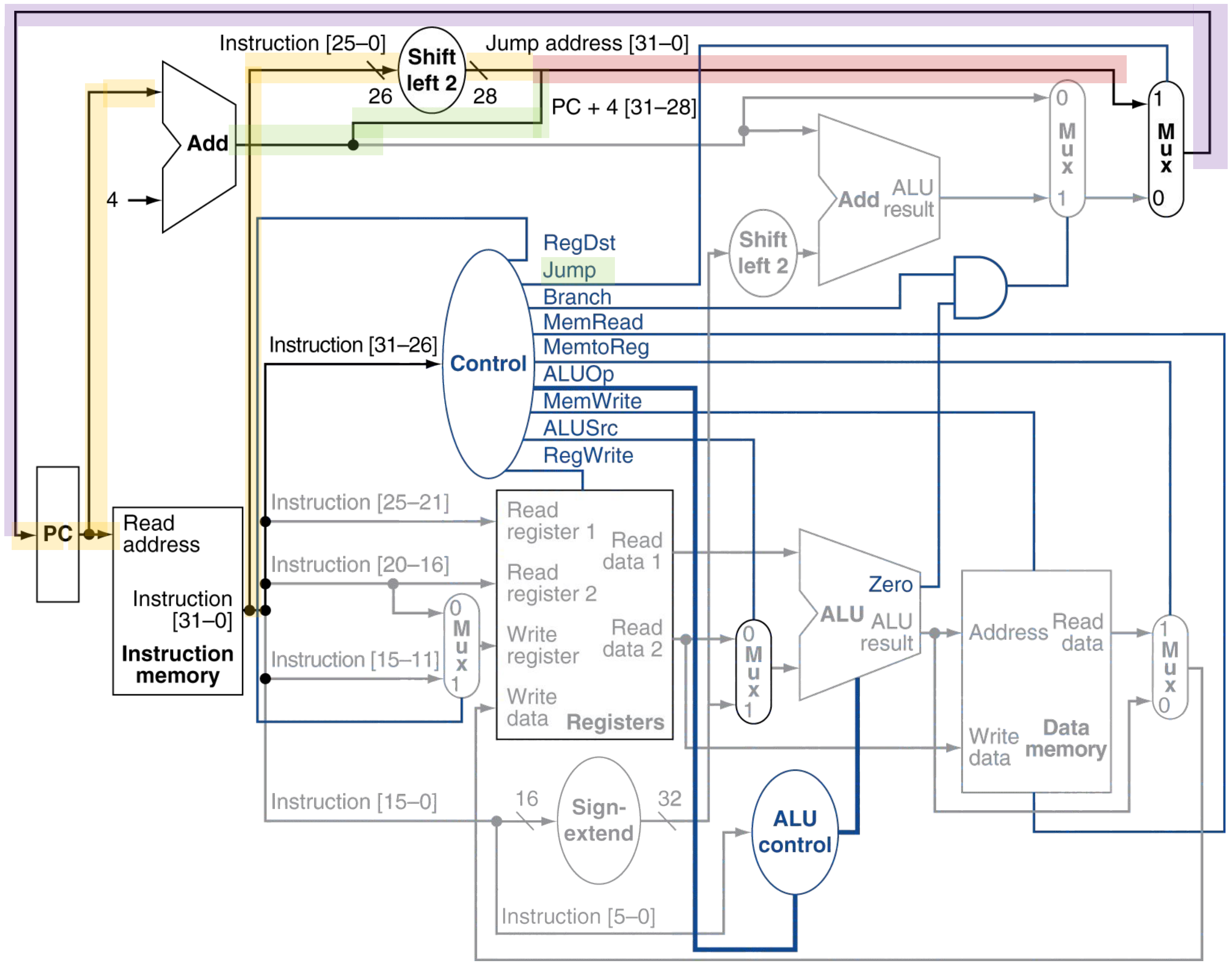
Jump uses word address

Update PC with concatenation of

- Top 4 bits of old PC
- 26-bit jump address
- 00

Need an extra control signal decoded from opcode

Datapath With Jumps Added



Recap: Single Cycle Datapath

- The Instruction Fetch Unit gives us the instruction.
- The OP field is fed to the Main Control for decode and the Func field is fed to the ALU Control for local decoding.
- The Rt, Rs, Rd, and Imm16 fields of the instruction are fed to the data path.
- Based on the OP field of the instruction, the Main Control of sets the control signals RegDst, ALUSrc, ... etc
- The ALUctr uses the ALUop from the Main conrol and the func field of the instruction to generate the ALUctr signals to ask the ALU to do the right thing: Add, Subtract, Or, and so on.
- This processor will execute each of the MIPS instruction in the subset in one cycle.