CS / EE 320
Computer Organization and Assembly Language
Spring 2025
Lecture 15

Shahid Masud

Topics: Building a Single Cycle MIPS CPU, Adding Branch, Jump, etc. and Control Path

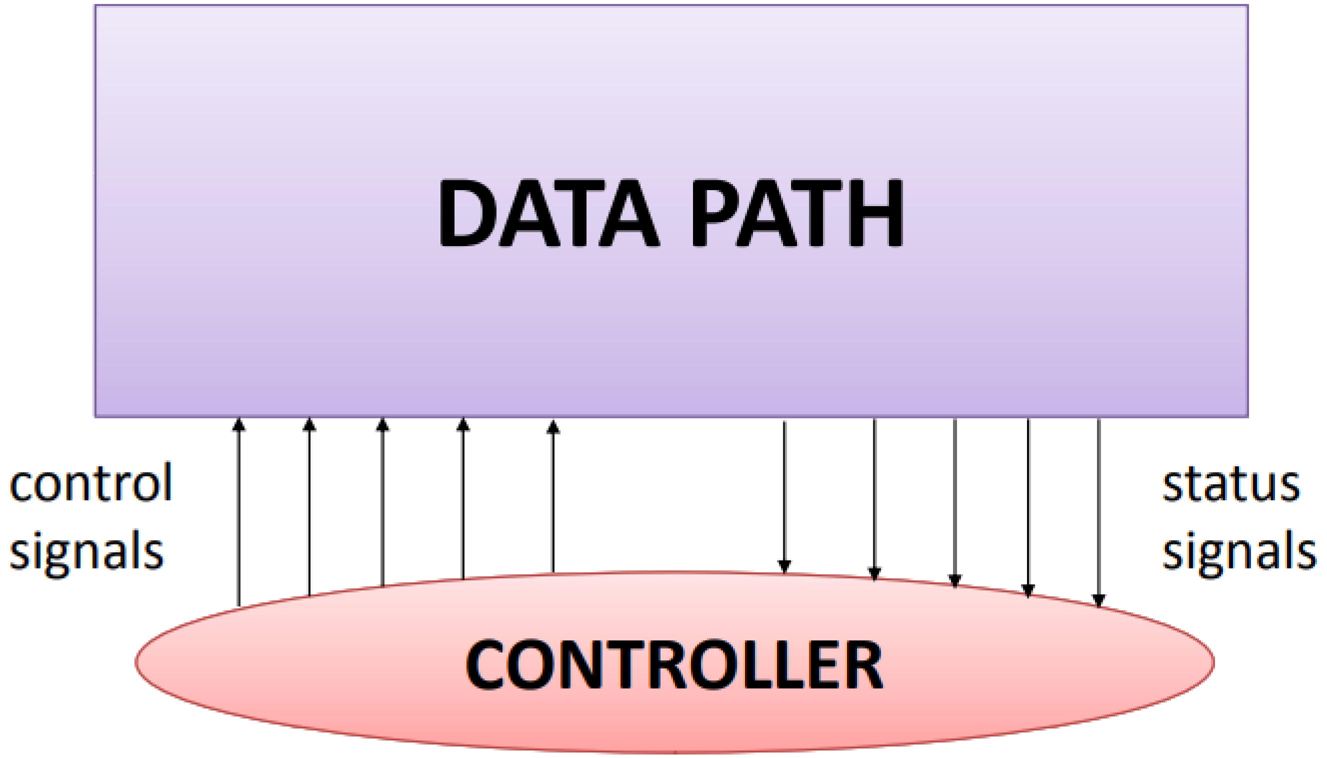
Important Announcement

- **Midterm Exam coming Friday, 14 March 2025**

Topics

- Remaining blocks of MIPS CPU
- Architecture of simple instructions in **Single Cycle MIPS**
  - J jump instruction
  - ADD instruction
  - ADDI instruction
  - BEQ instruction
  - LW load word instruction
  - SW store word instruction
- Control Path for Single Cycle MIPS CPU
- **Midterm Exam on Friday**

CPU consists of Data path and Control path



Develop Data path for add, sub, and, or, slt

- fetch instruction
- address the register file
- pass operands to ALU
- pass result to register file
- increment PC

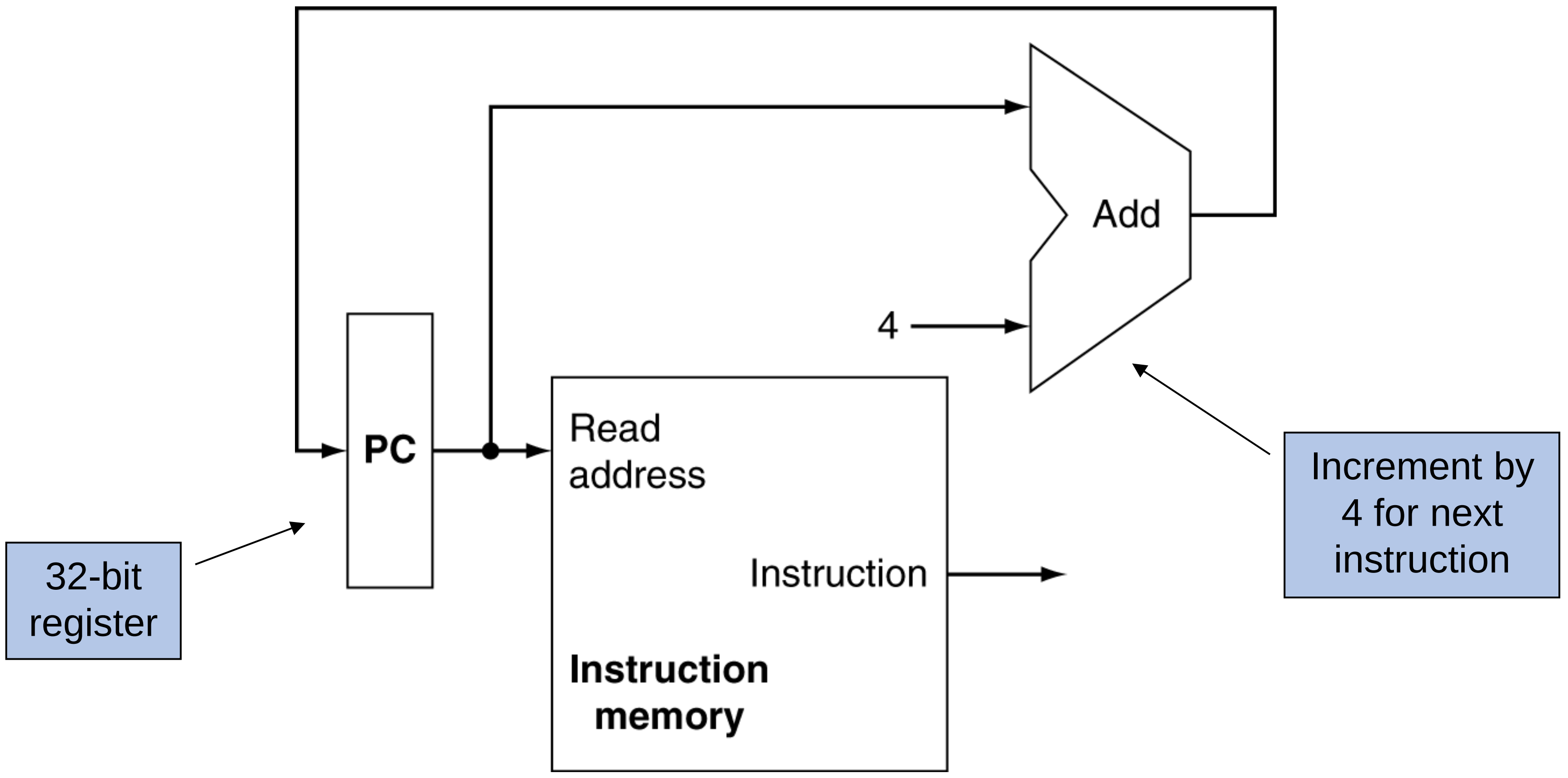
$\}$ actions required

Format: add $t0, $s1, $s2

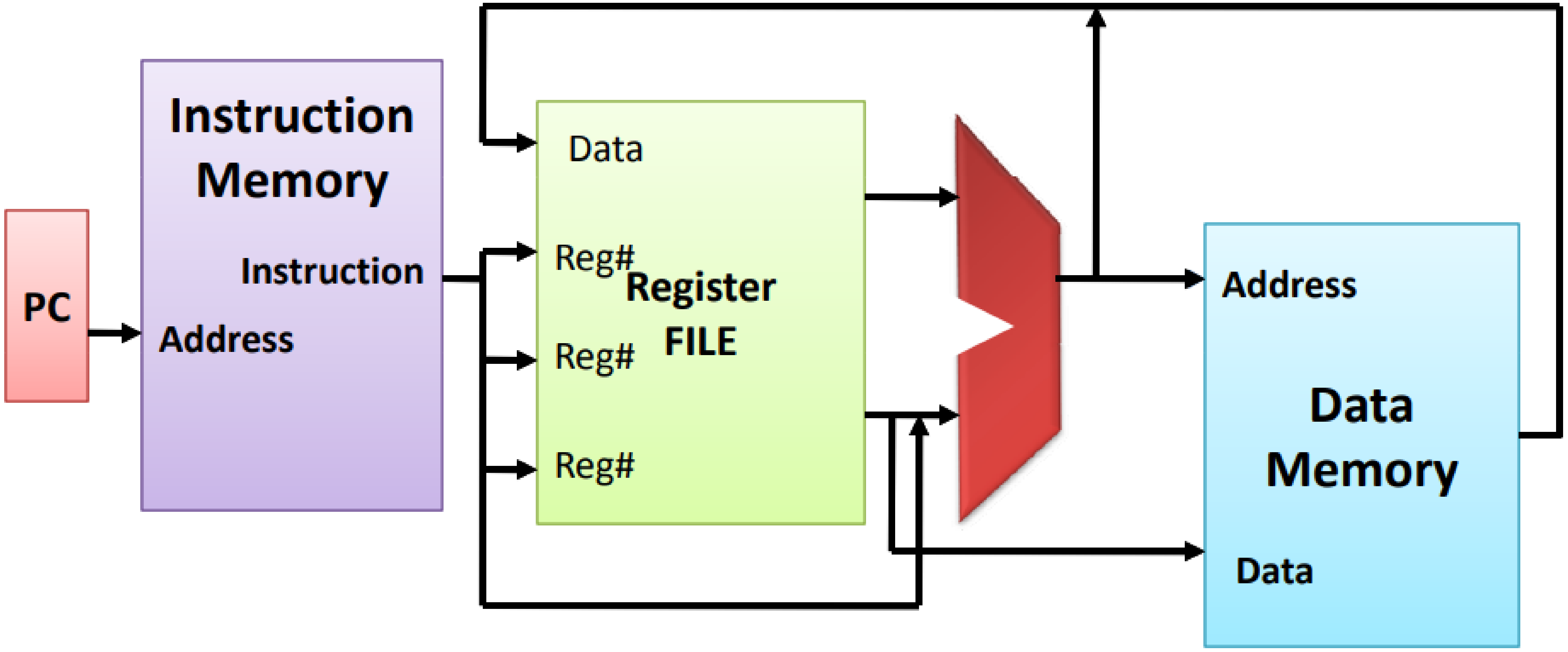
000000	10001	10010	01000	00000	100000
op	rs	rt	rd	shamt	funct

Instruction Fetch – PC Register operation

This block is needed for all instructions



Overview of Simple CPU Design for R Format



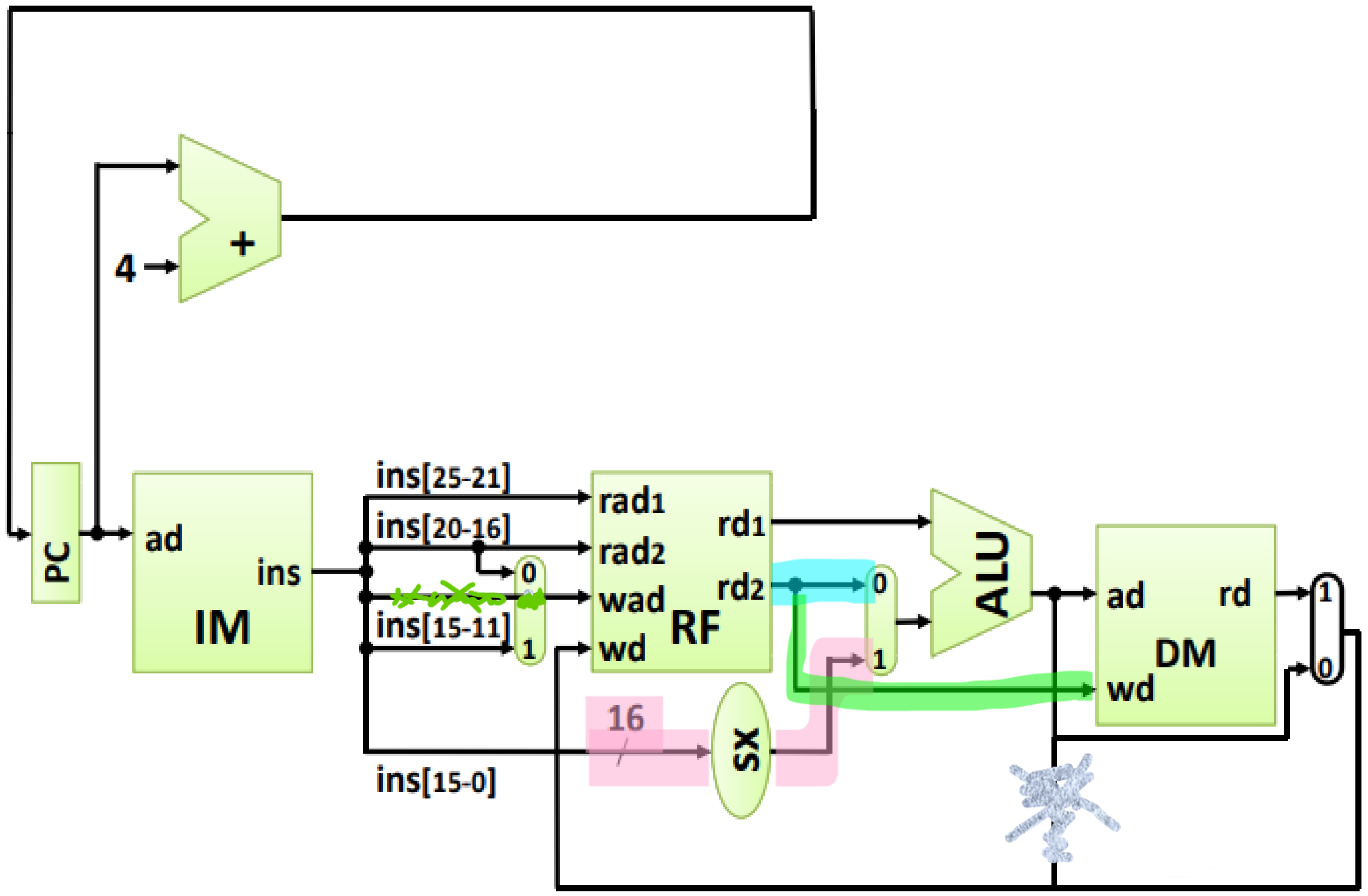
Implement the "lw" Instruction

Load and Store instructions

- format : I
- Example: lw $t0, 32($s2)

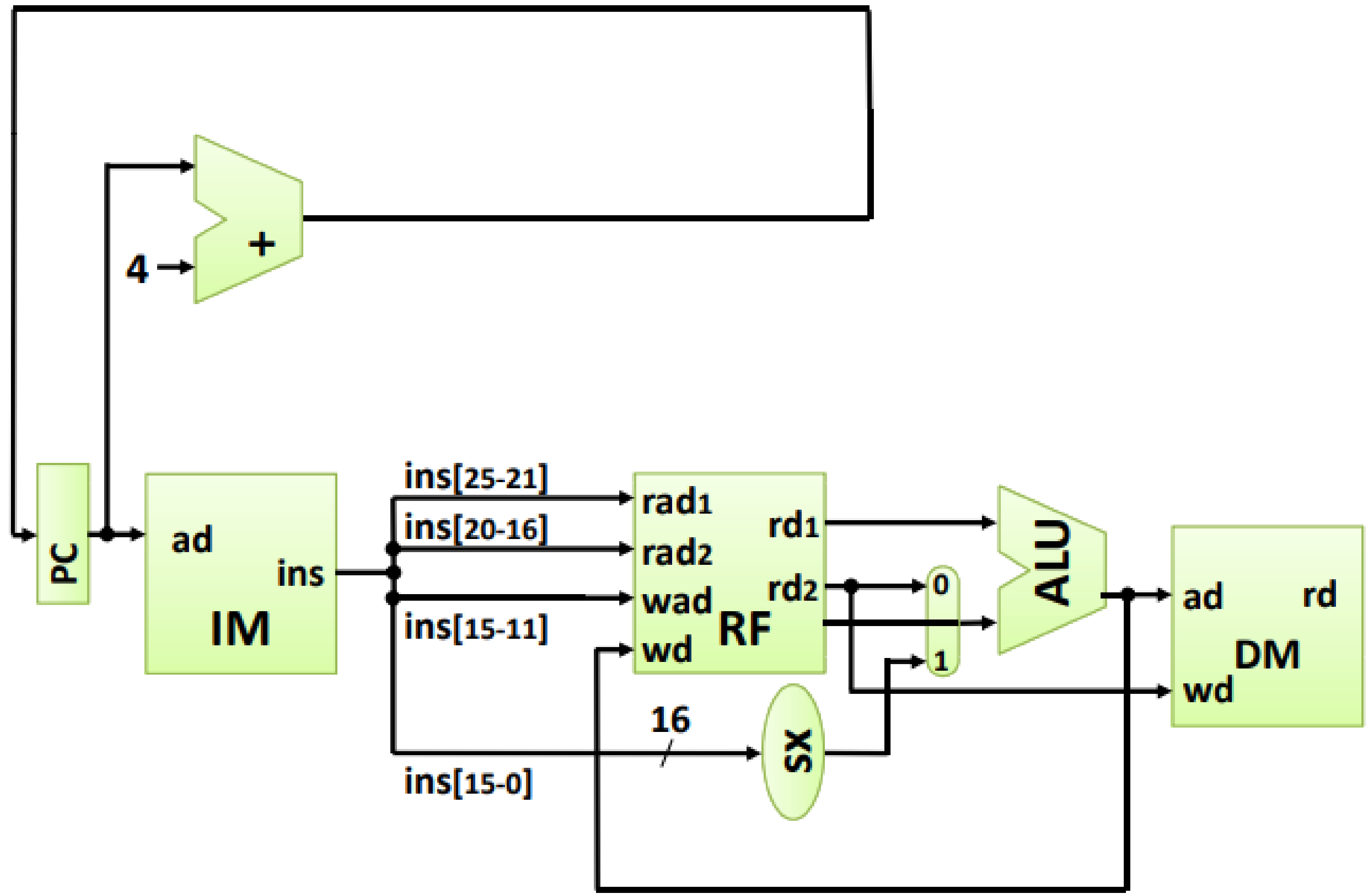
35	18	9	32
op	rs	rt	16 bit number

Adding "lw" Instruction



Implement the “sw” Instruction

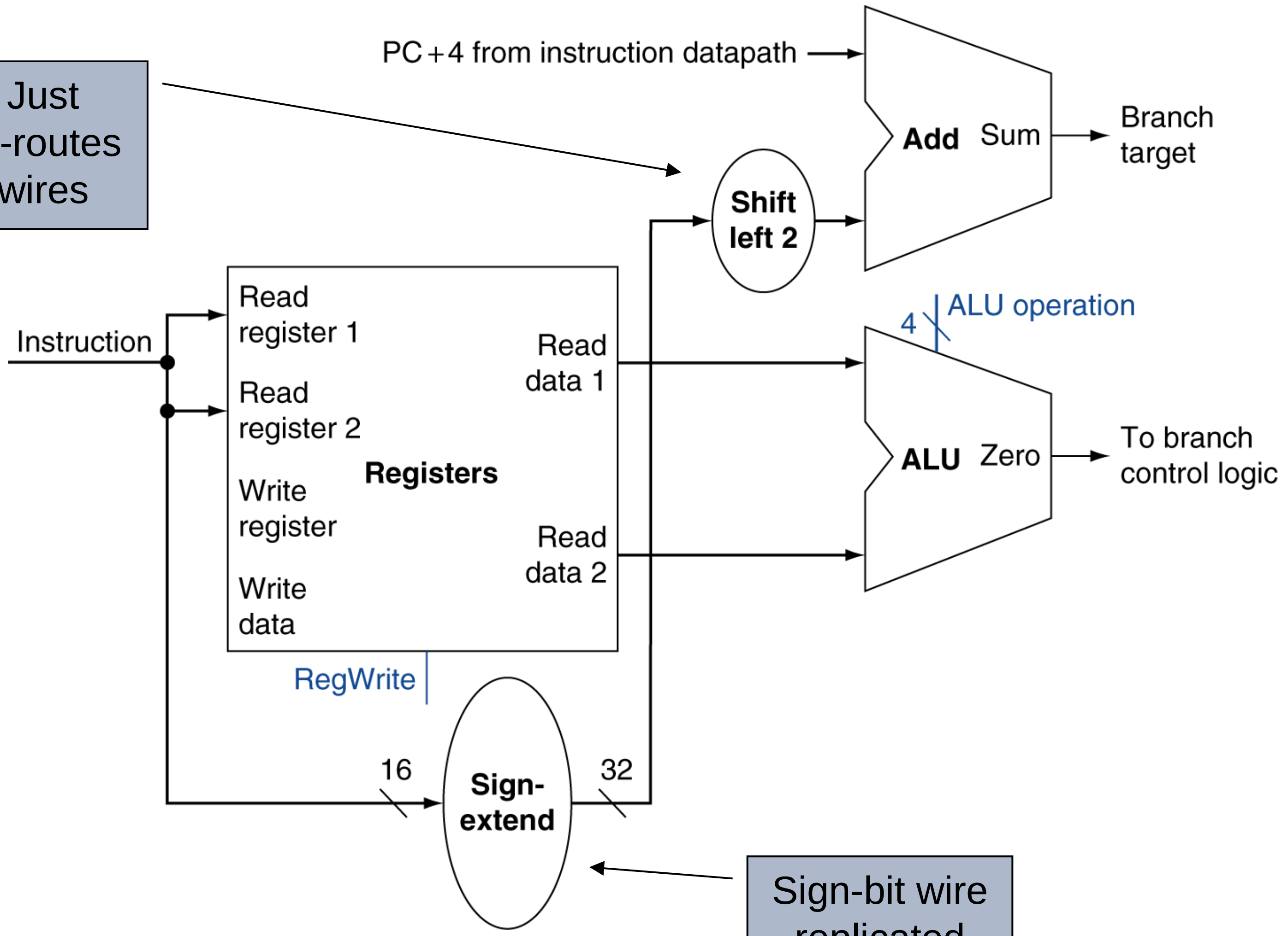
Adding “sw” Instruction



Branch Instructions

- 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

Datapath for Branch Instructions



Just re-routes wires

Sign-bit wire replicated

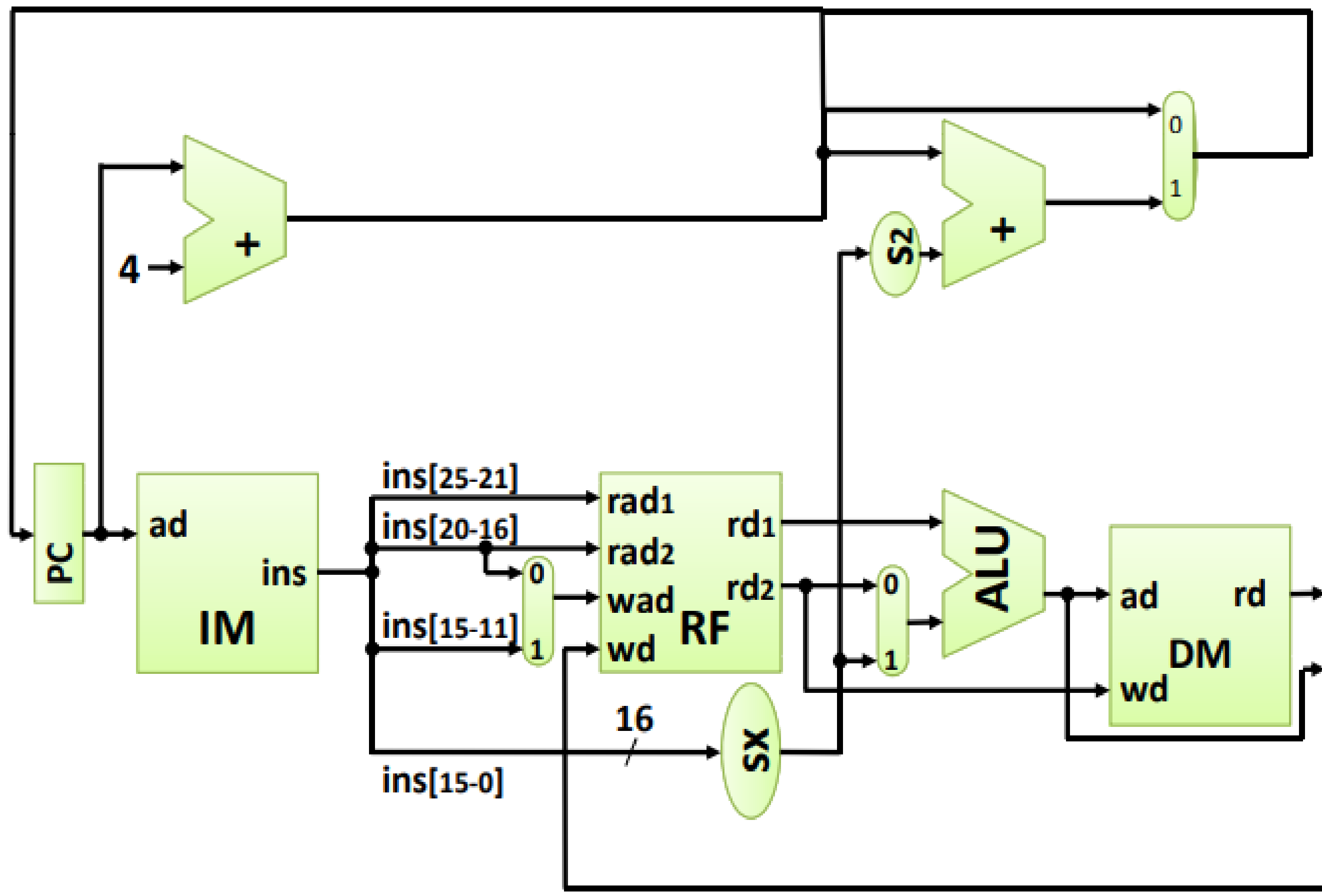
Implement “**beq**” Instruction

Format of beq instruction

- beq I - format

op	rs	rt	16 bit number

Adding “beq” 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

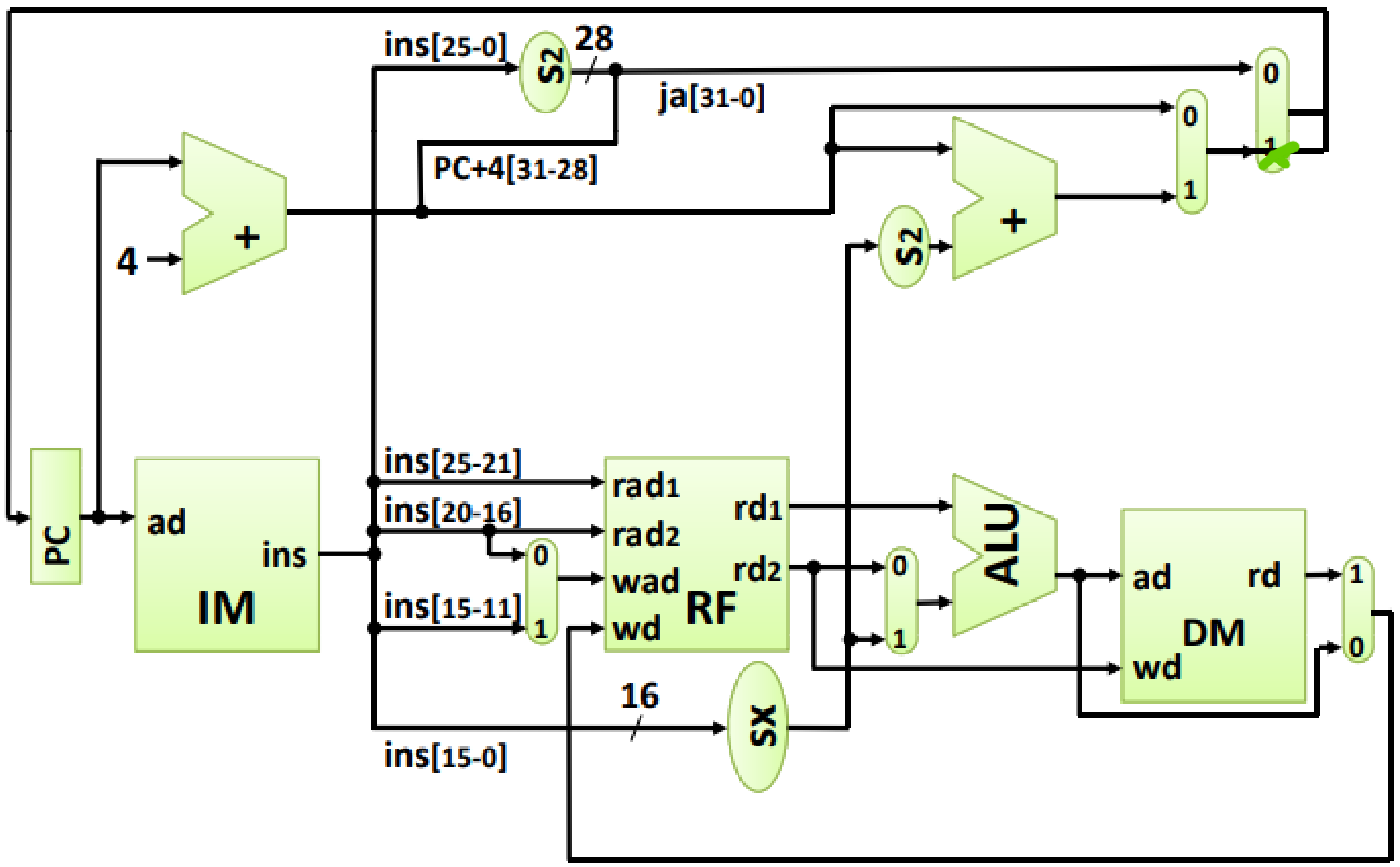
Implement Jump "j" instruction

Format of jump instruction

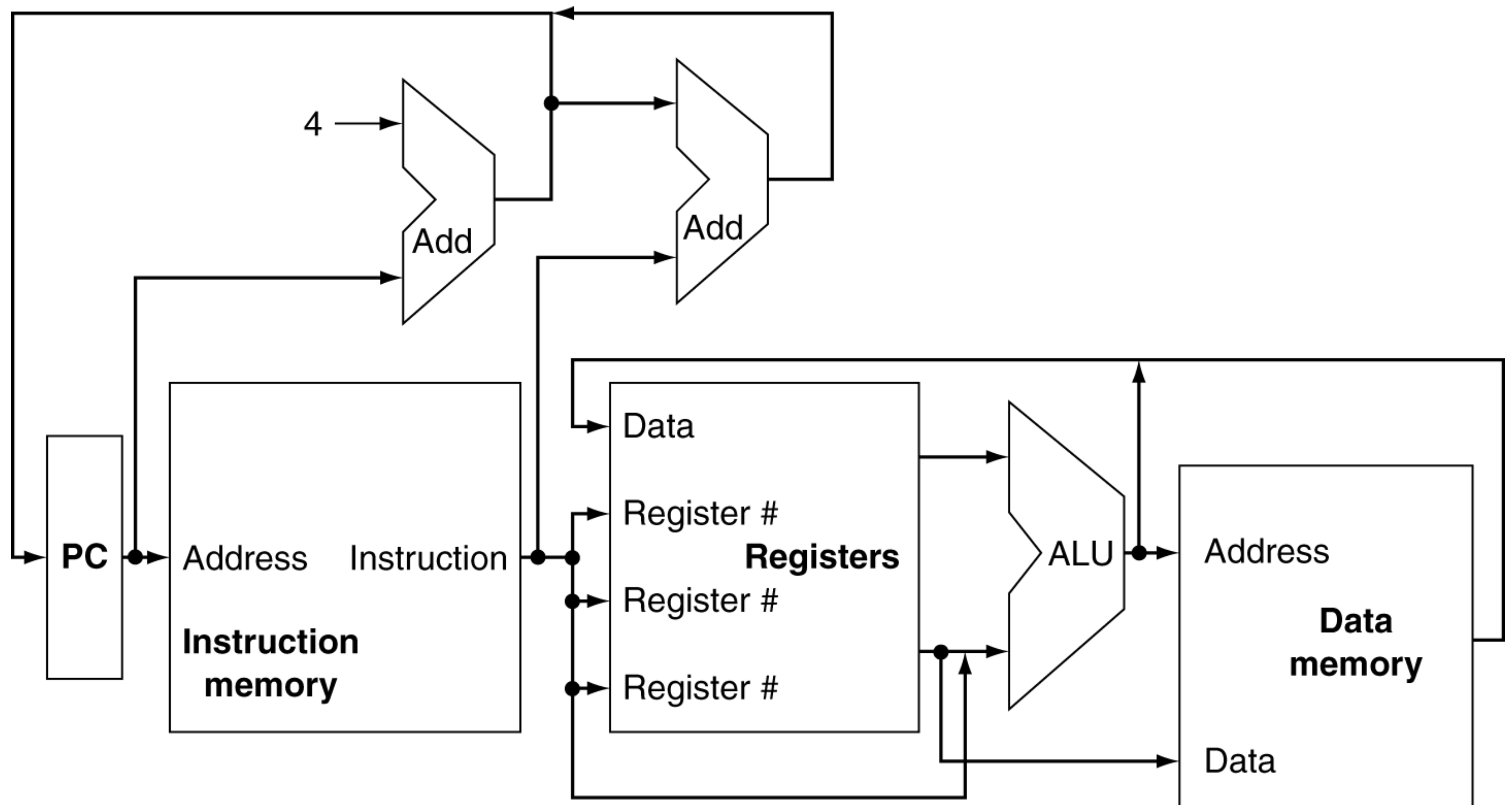
- j J - format

op	26 bit number

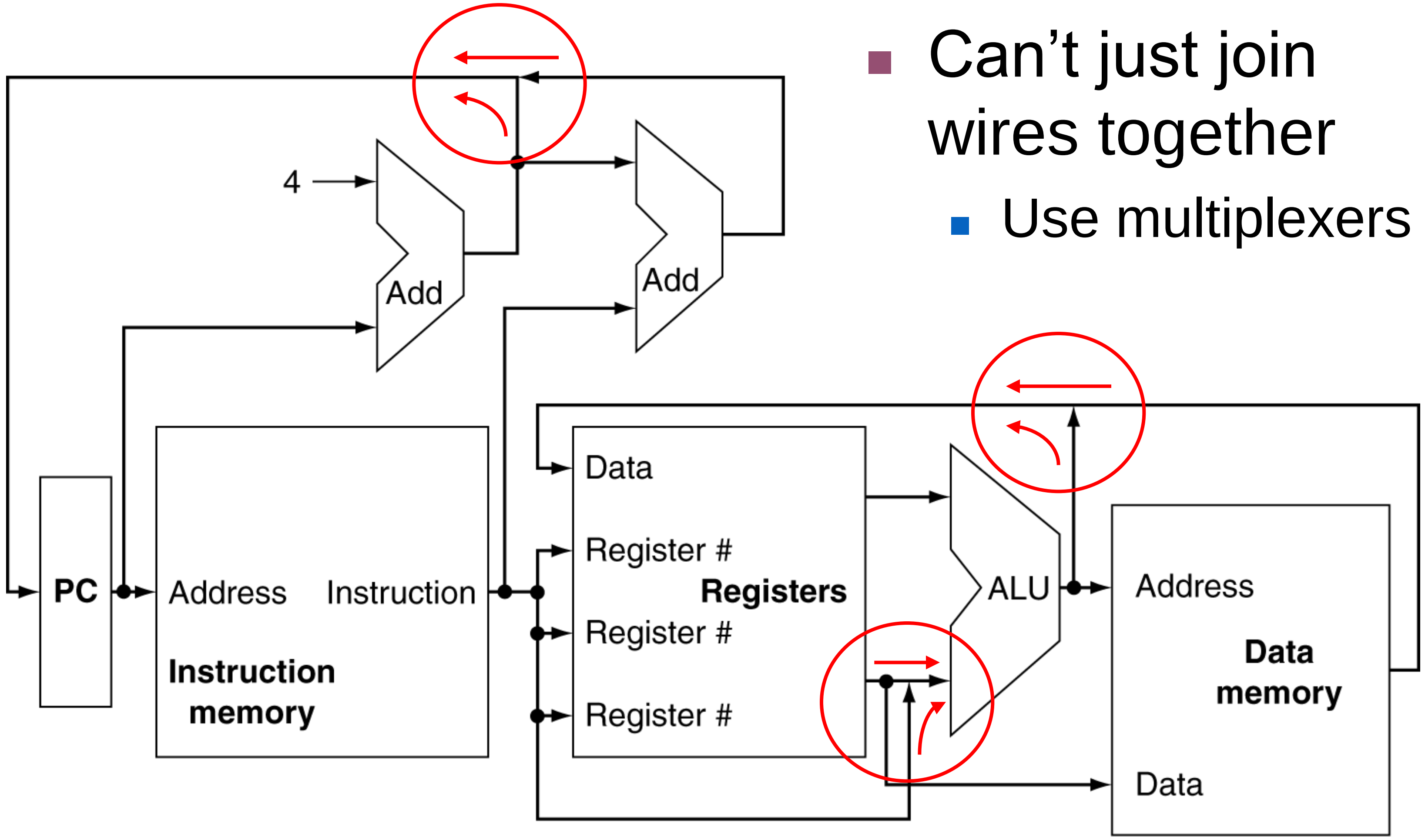
Adding "j" Instruction



Basic CPU Overview



Multiplexers Needed

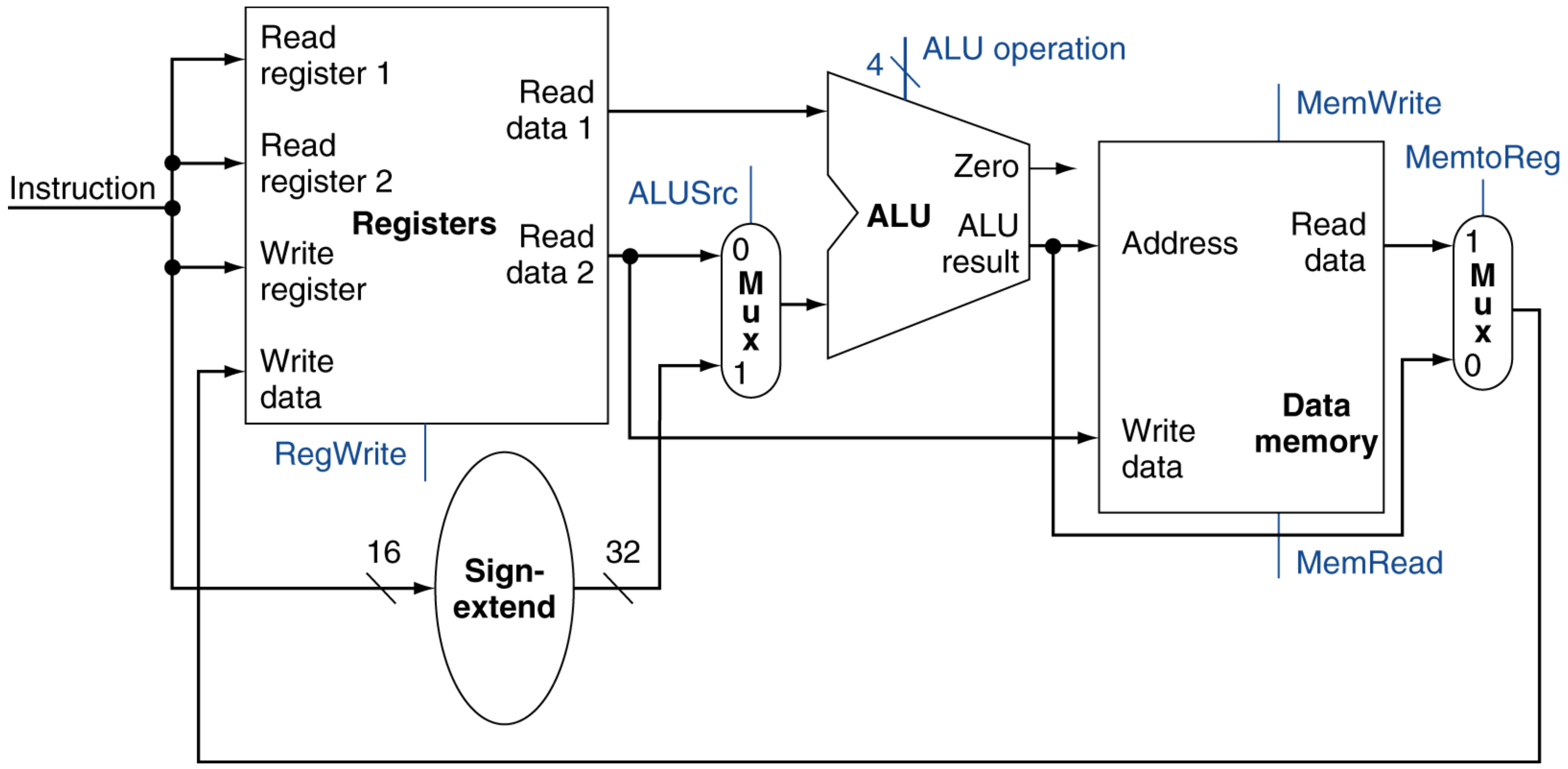


- Can't just join wires together
  - Use multiplexers

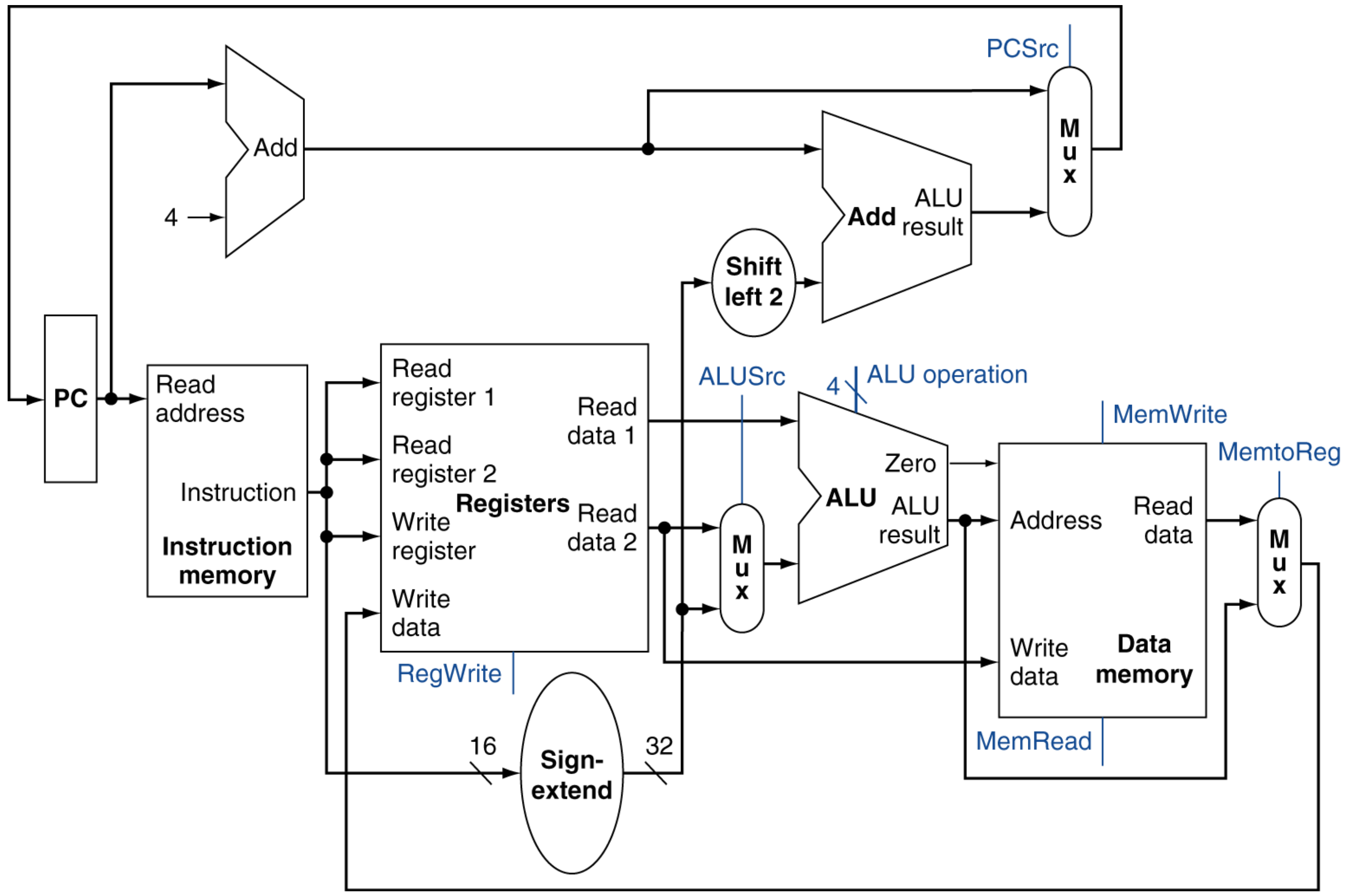
Composing the Elements

- First-cut data path does an instruction in one clock cycle
  - Each datapath element can only do one function at a time
  - Hence, we need separate instruction and data memories
- Use multiplexers where alternate data sources are used for different instructions

R-Type/Load/Store Datapath – Some Controls



Full Datapath



Data path for Memory Instructions

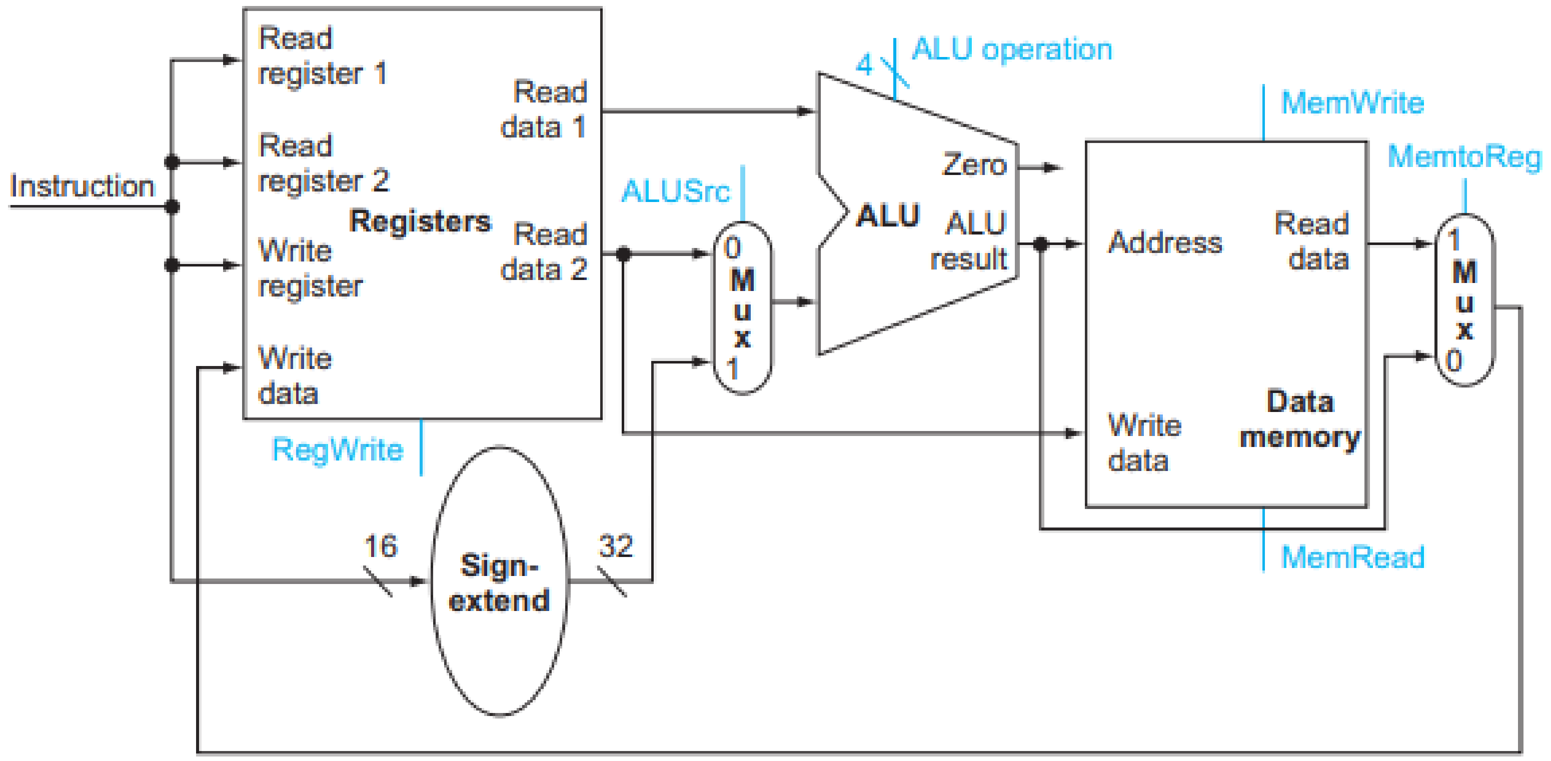
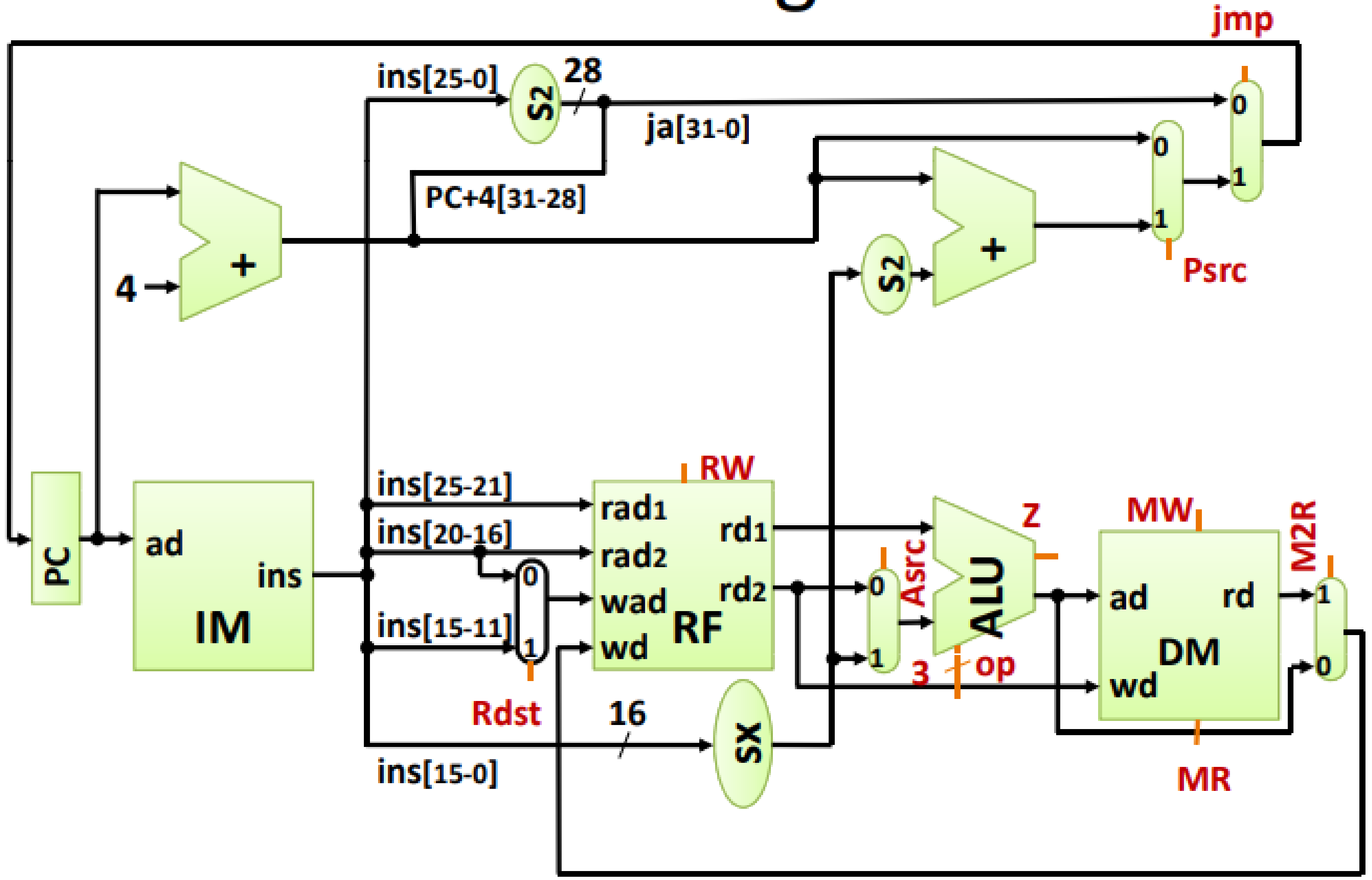


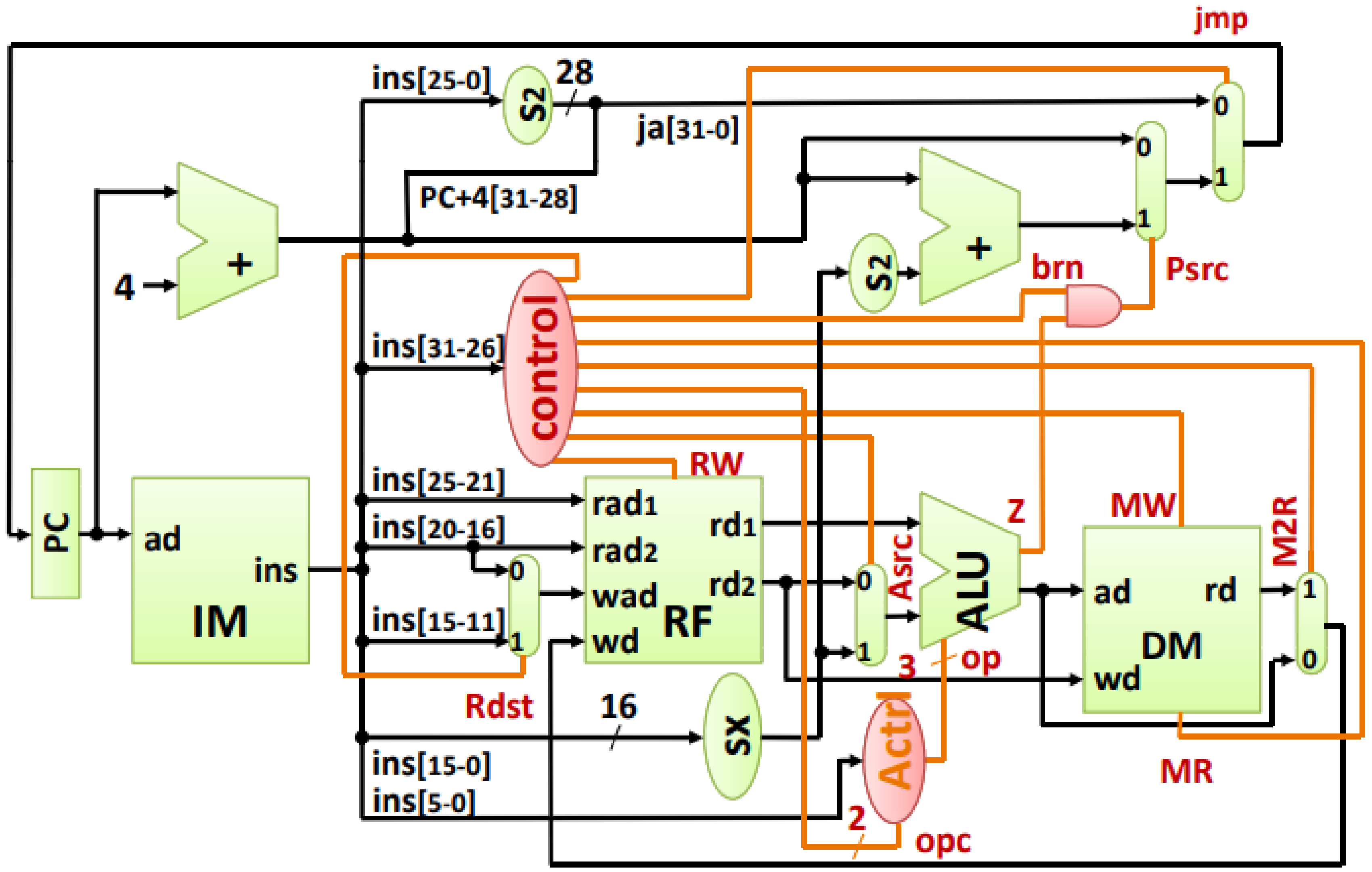
FIGURE 4.10 The datapath for the memory instructions and the R-type instructions. This example shows how a single datapath can be assembled from the pieces in Figures 4.7 and 4.8 by adding multiplexors. Two multiplexors are needed, as described in the example.

Identify Control Signals for Simple CPU

Control signals



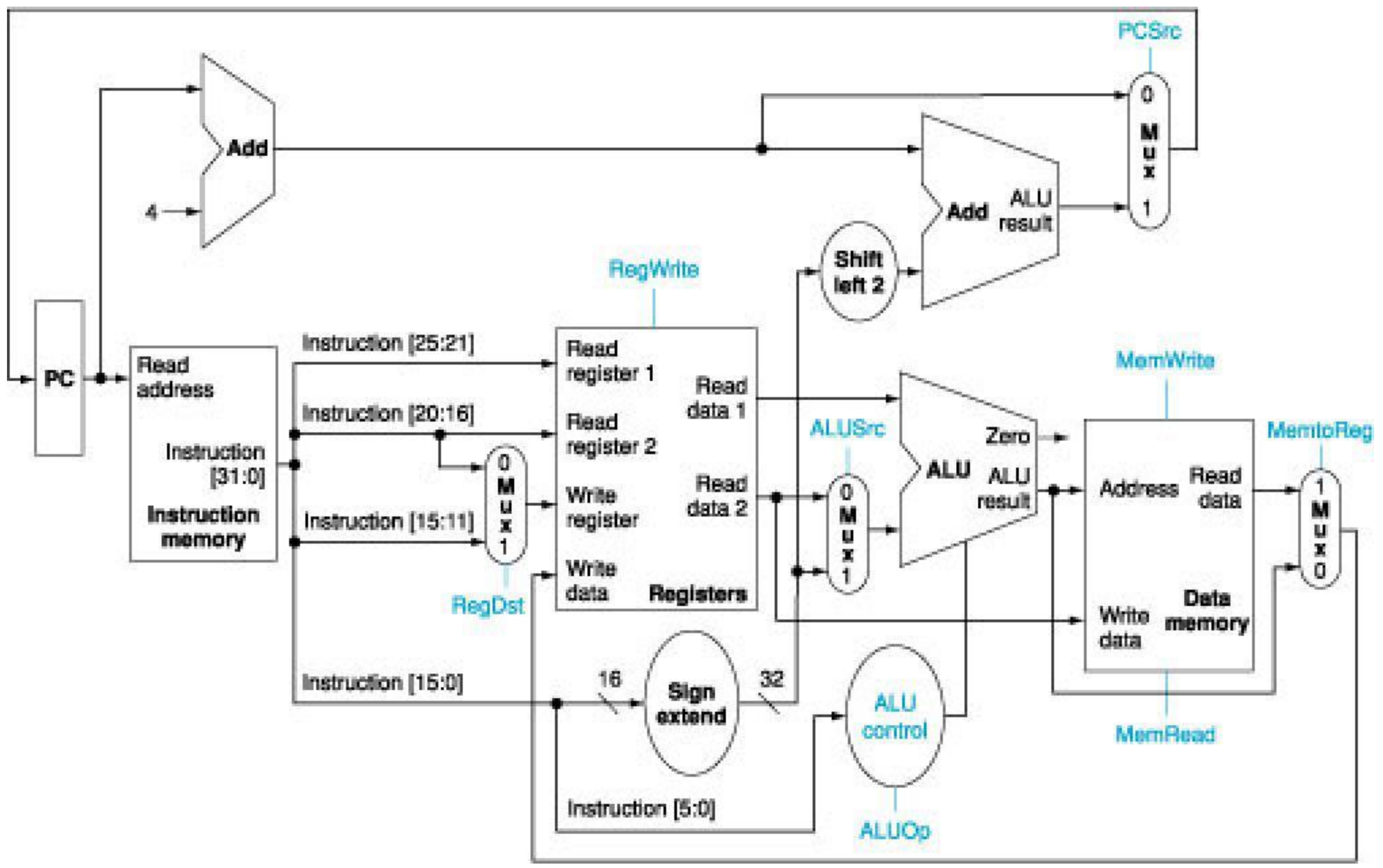
Combining Data path and Control path signals



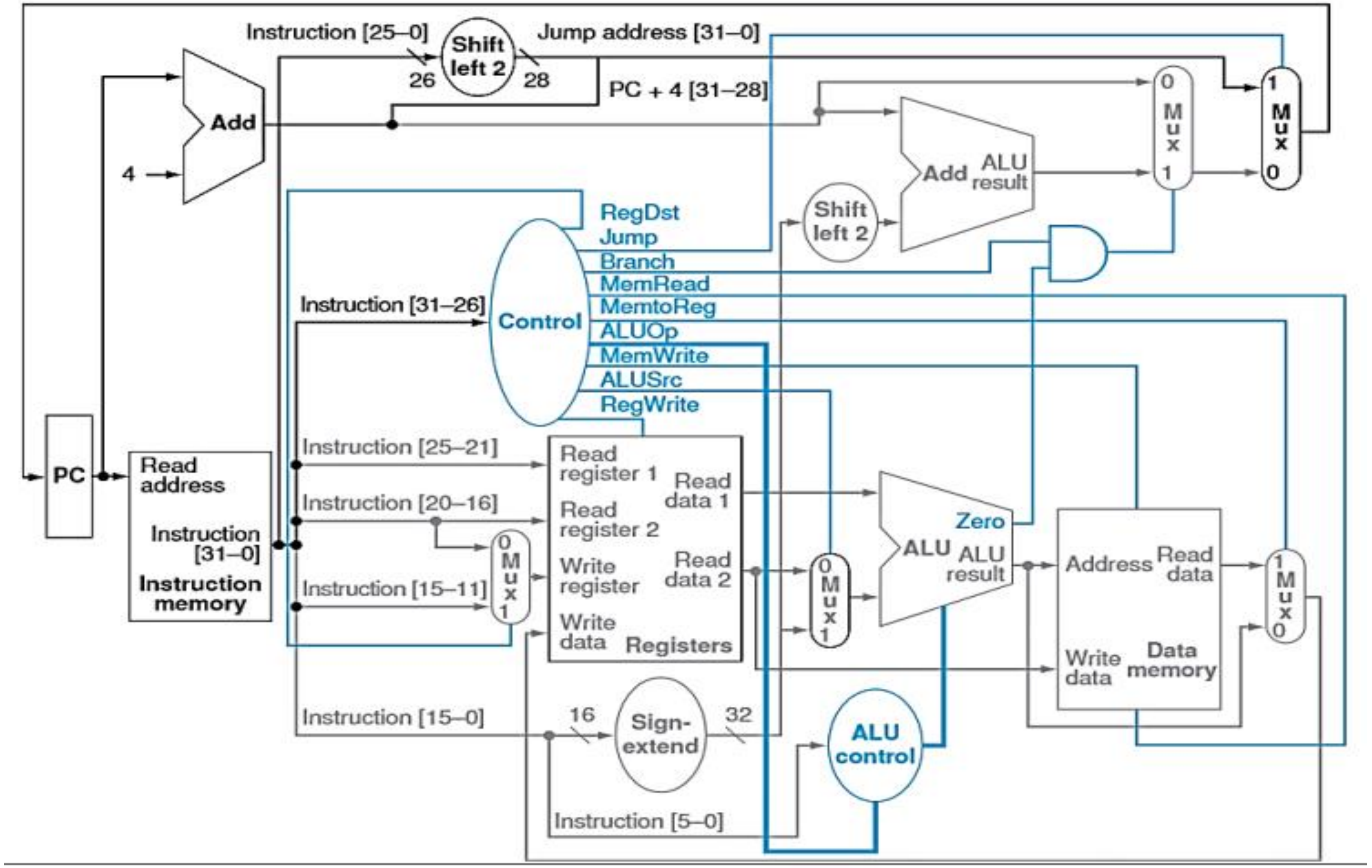
Control words in Simple MIPS CPU

Instruction	Opcode	Rdst	RW	Asrc	MW	MR	M2R	Brn	Jmp
Rtype	000000	1	1	0	0	0	0	0	0
Sw	101011	X	0	1	1	0	X	0	0
Lw	100011	0	1	1	0	1	1	0	0
Beq	000100	X	0	0	0	0	X	1	0
J	000010	X	0	X	0	0	X	X	1

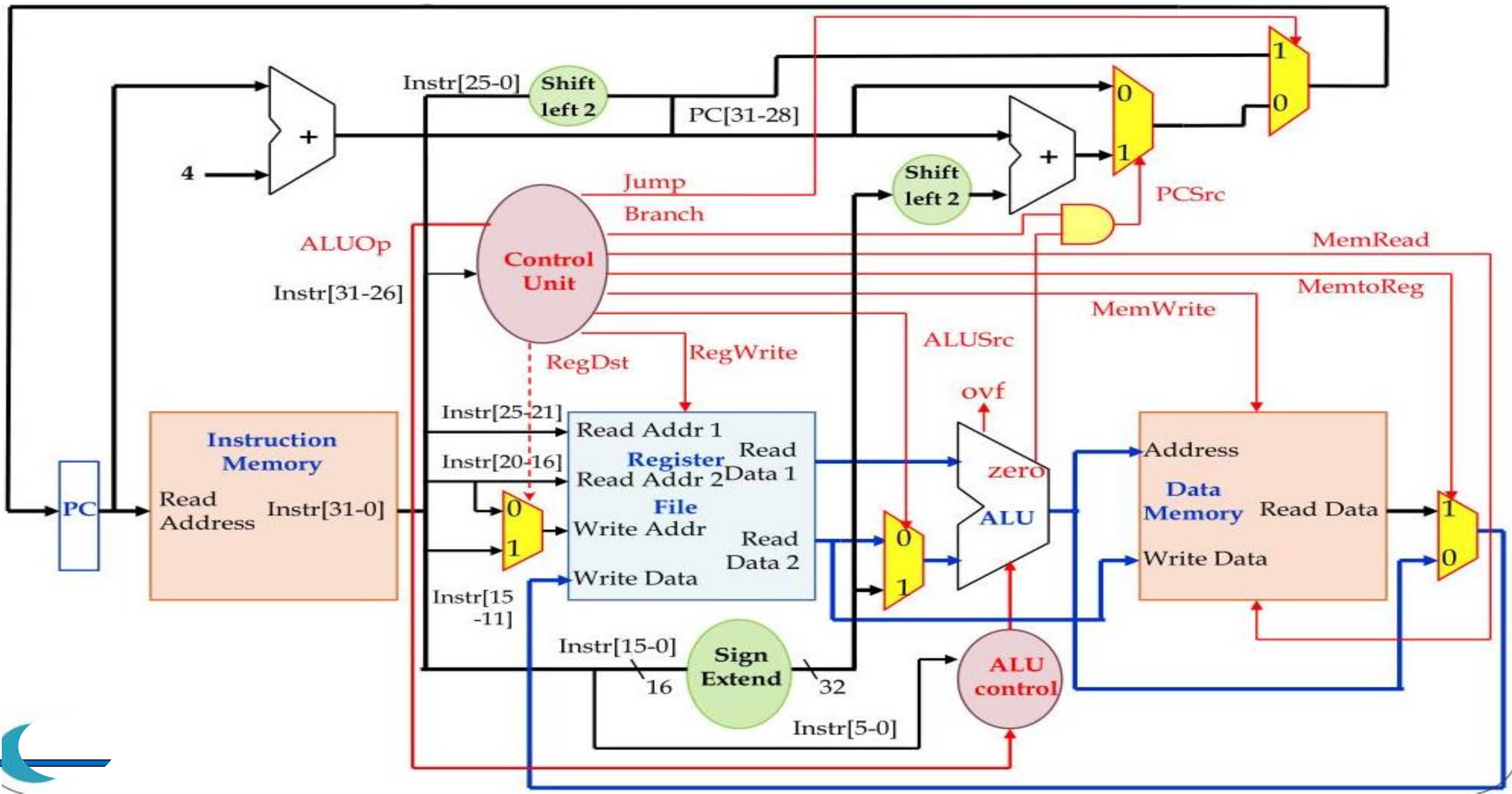
Control Signals needed in MIPS (P&H book style)



Data path with Control Signals in simple MIPS

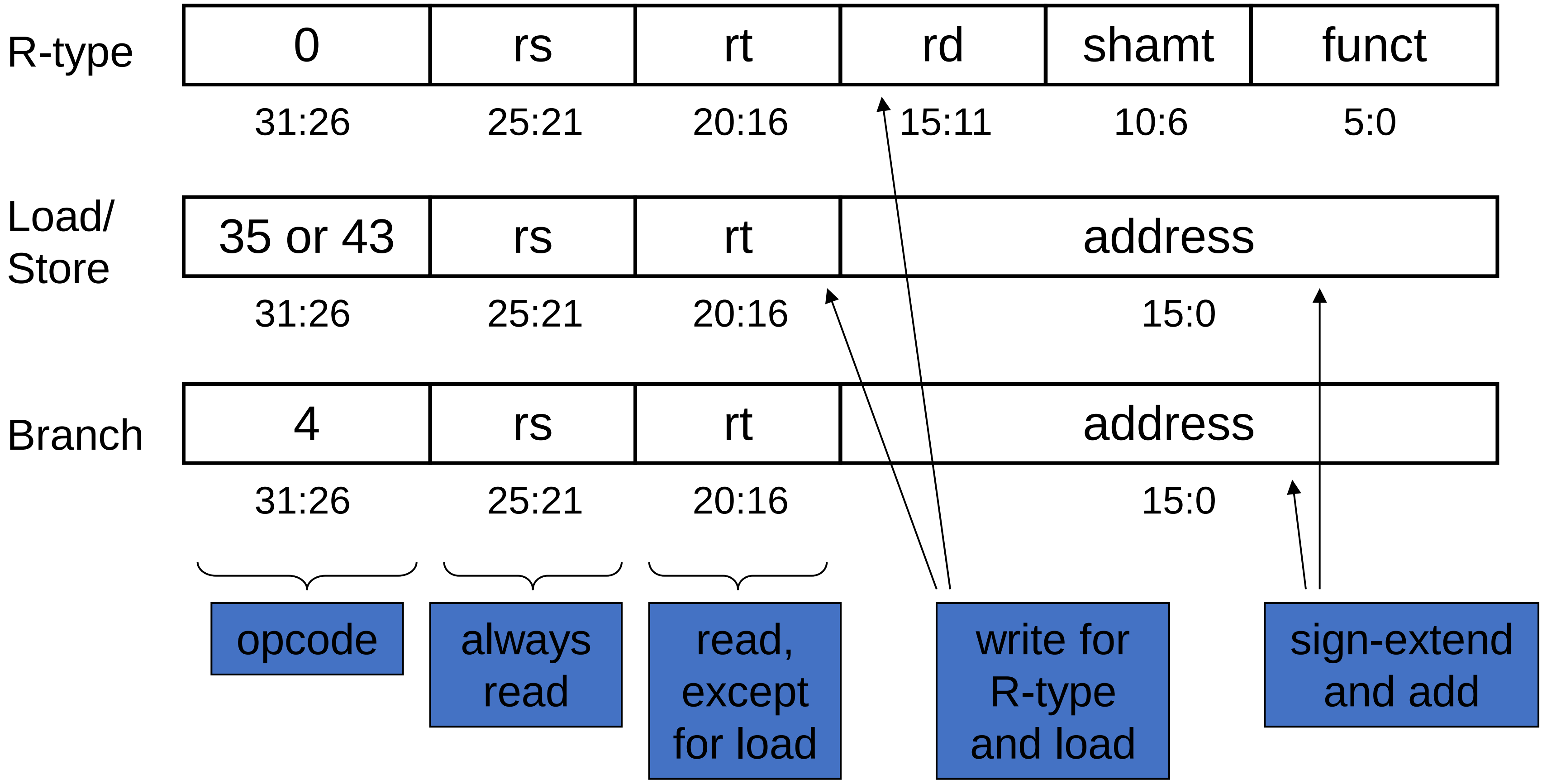


Another (clear) View of Control Signals



The Main Control Unit

- Control signals derived from instruction



Generating Control Signals

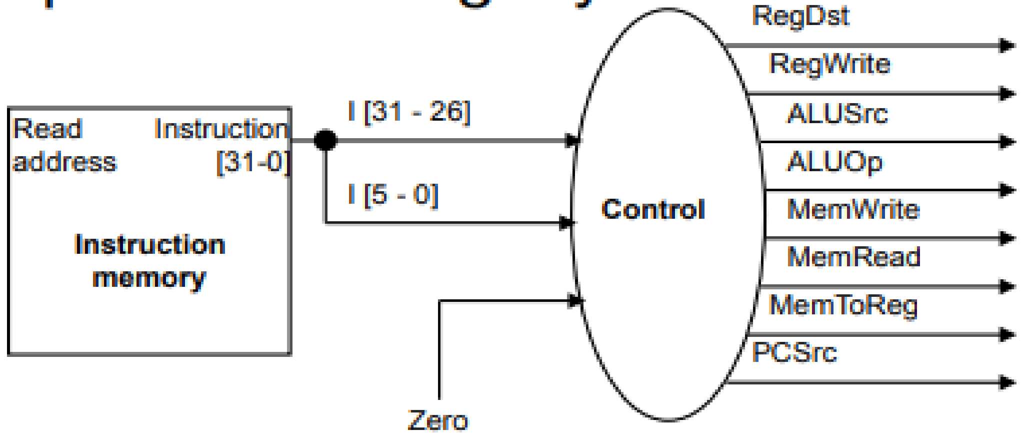
The control unit needs 13 bits of inputs

- Six bits make up the instruction's opcode
- Six bits come from the instruction's func field
- It also needs the Zero output of the ALU

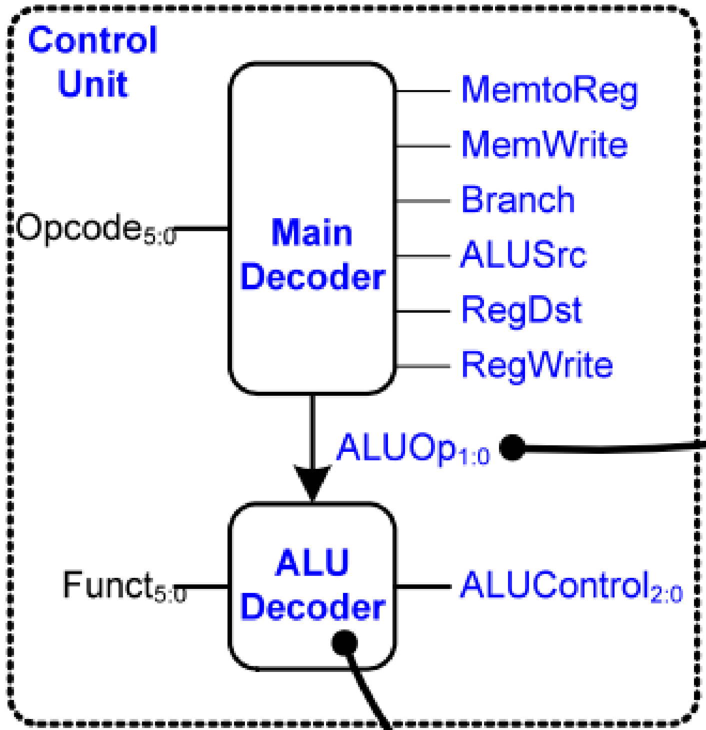
The control unit generates 10 bits of output, corresponding to the signals mentioned earlier

You can build the actual circuit by using big K-maps, big Boolean algebra, or big circuit design programs

The textbook presents a slightly different control unit



Control words with ALU Op codes



ALUOp$_{1:0}$	Meaning
0 0	Add
0 1	Subtract
1 0	Look at Funct
1 1	*not used*

ALUOp$_{1:0}$	Funct	ALUControl$_{2:0}$
0 0	x	010 (Add)
x 1	x	110 (Subtract)
1 x	100000 (add)	010 (Add)
1 x	100010 (sub)	110 (Subtract)
1 x	100100 (and)	000 (And)
1 x	100101 (or)	001 (Or)
1 x	101010 (slt)	111 (SLT)

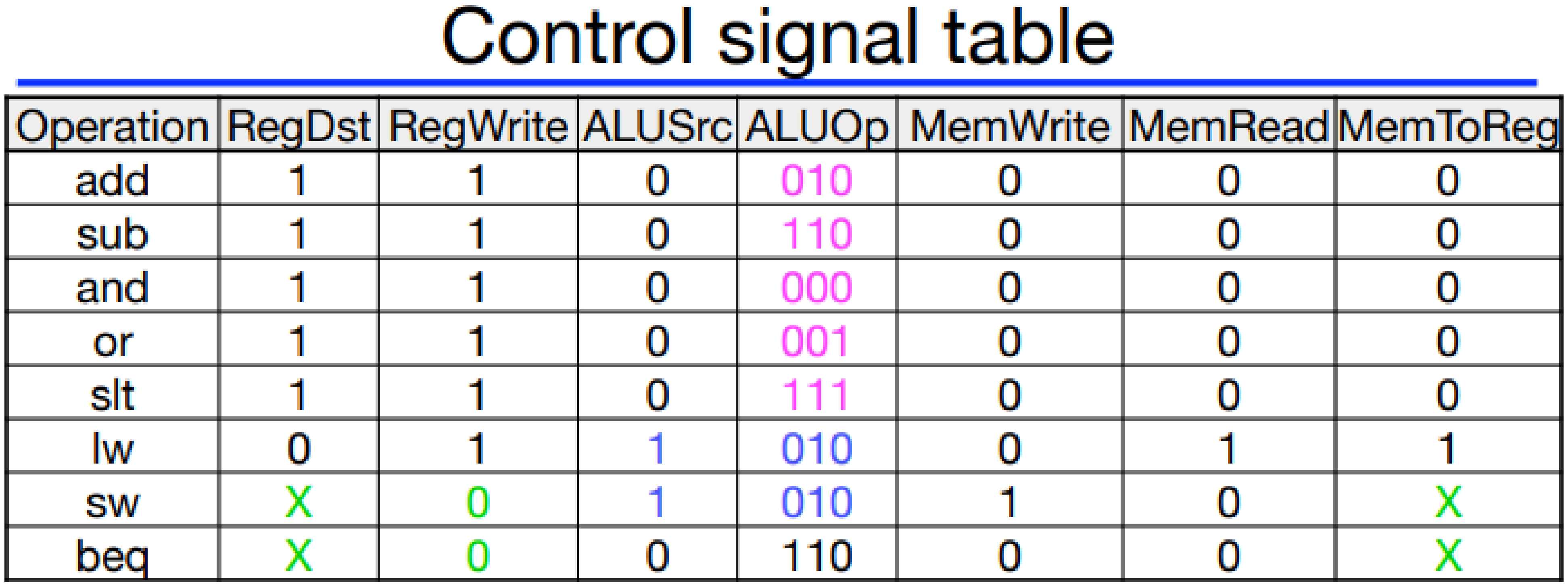
Copyright © 2007 Elsevier

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

Control words for different Instructions



Control signal table

Operation	RegDst	RegWrite	ALUSrc	ALUOp	MemWrite	MemRead	MemToReg
add	1	1	0	010	0	0	0
sub	1	1	0	110	0	0	0
and	1	1	0	000	0	0	0
or	1	1	0	001	0	0	0
slt	1	1	0	111	0	0	0
lw	0	1	1	010	0	1	1
sw	X	0	1	010	1	0	X
beq	X	0	0	110	0	0	X