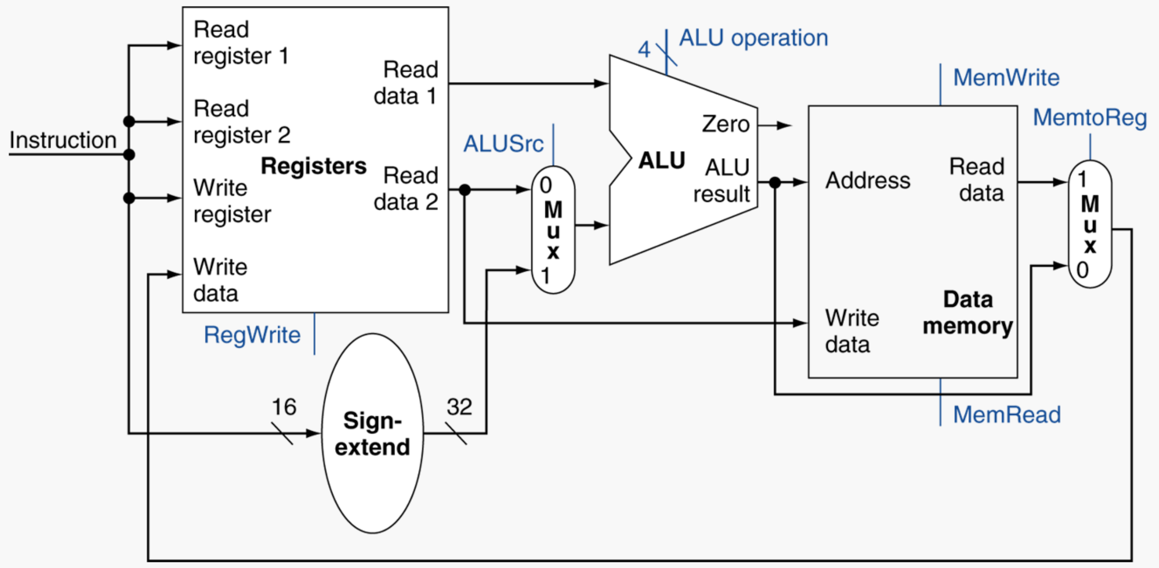
Composing the Elements

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

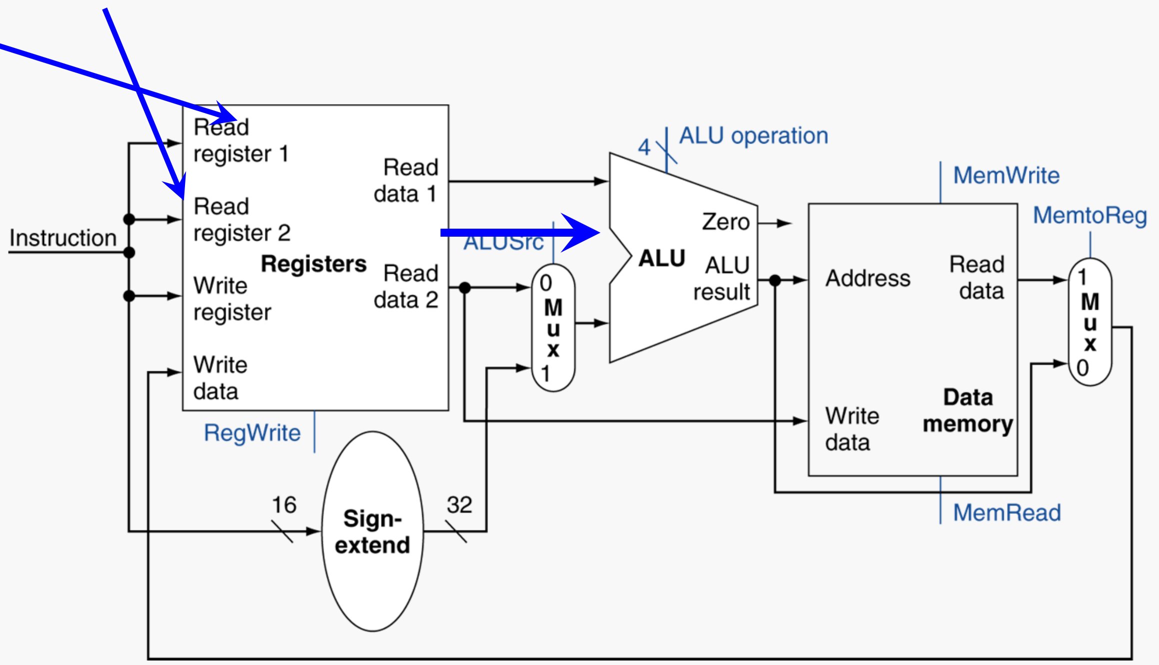


How would this execute: `add $s0, $s1, $s2;`

000000	10001	10010	10000	00000	100000
op	**rs**	**rt**	**rd**	**shamt**	**funct**

R-Type Execution

000000	10001	00011	10000	00000	100000
op	**rs**	**rt**	**rd**	**shamt**	**funct**



Reg #s for $s1 and $s2 are input to the register file's read register ports.

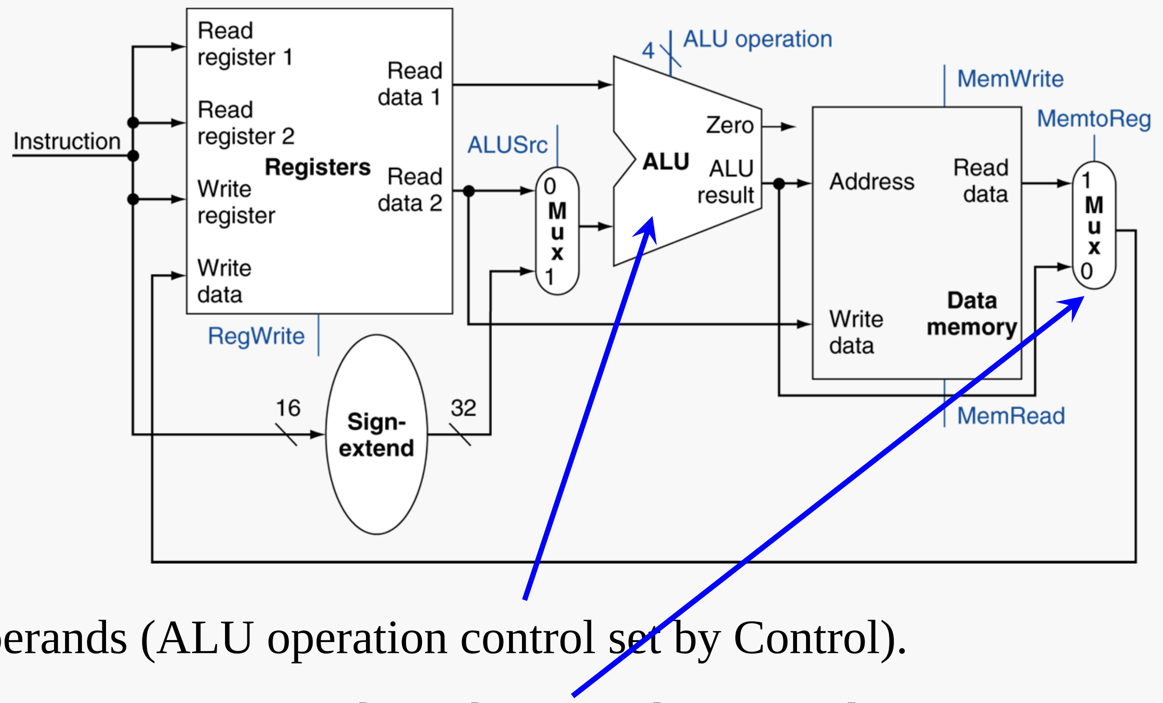
Low 16 bits from instruction are fed to the Sign-extend logic… irrelevant.

Values from $s1 and $s2 are fetched and passed to ALU (ALUSrc set to 0 by Control).

©2006-09 McQuain, Feng & Ribbens

R-Type Execution

000000	10001	00011	10000	00000	100000
op	**rs**	**rt**	**rd**	**shamt**	**funct**

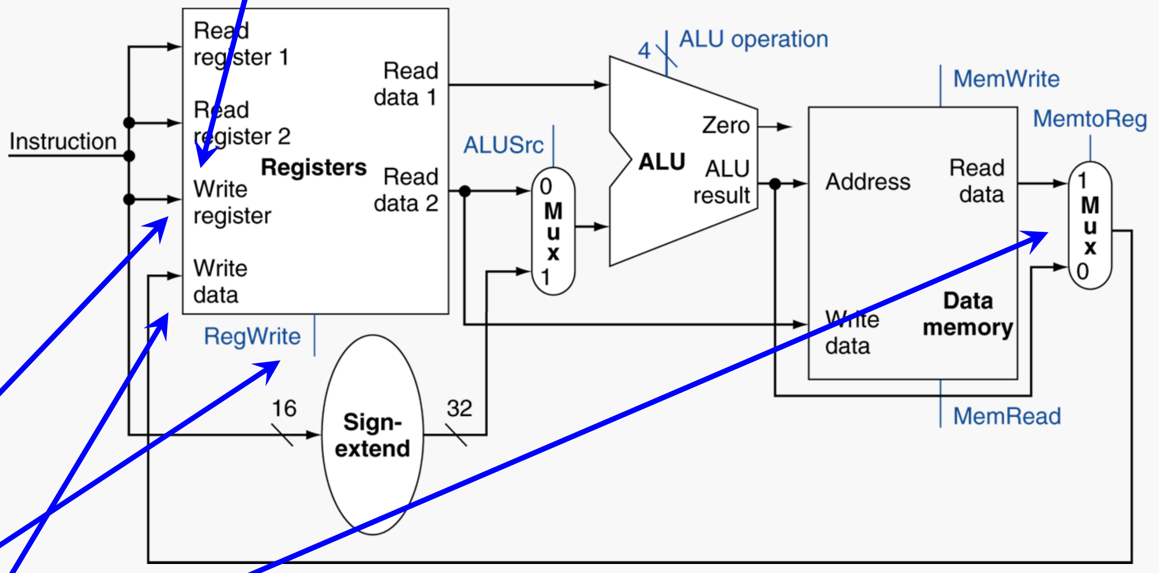


ALU computes sum of operands (ALU operation control set by Control).

ALU output tranferred to MUX, set to pass through input 0 by Control.

R-Type Execution

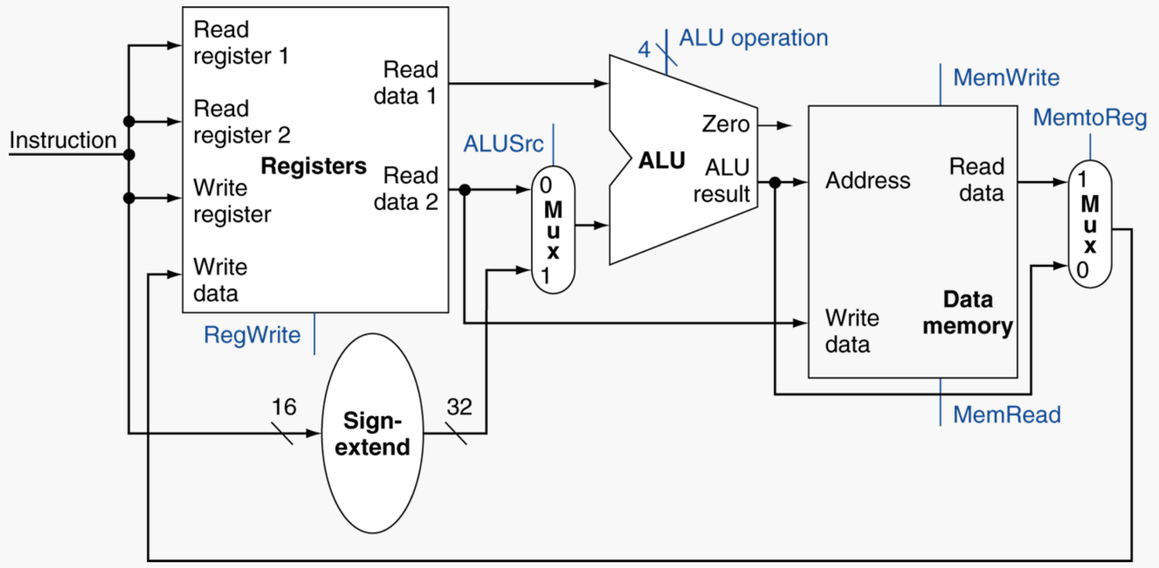
000000	10001	00011	10000	00000	100000
op	**rs**	**rt**	**rd**	**shamt**	**funct**



Reg# for $s0 is input to register file write register port.

ALU output transferred to register file Write data port.
RegWrite control line set by Control.

R-Type/Load/Store Datapath

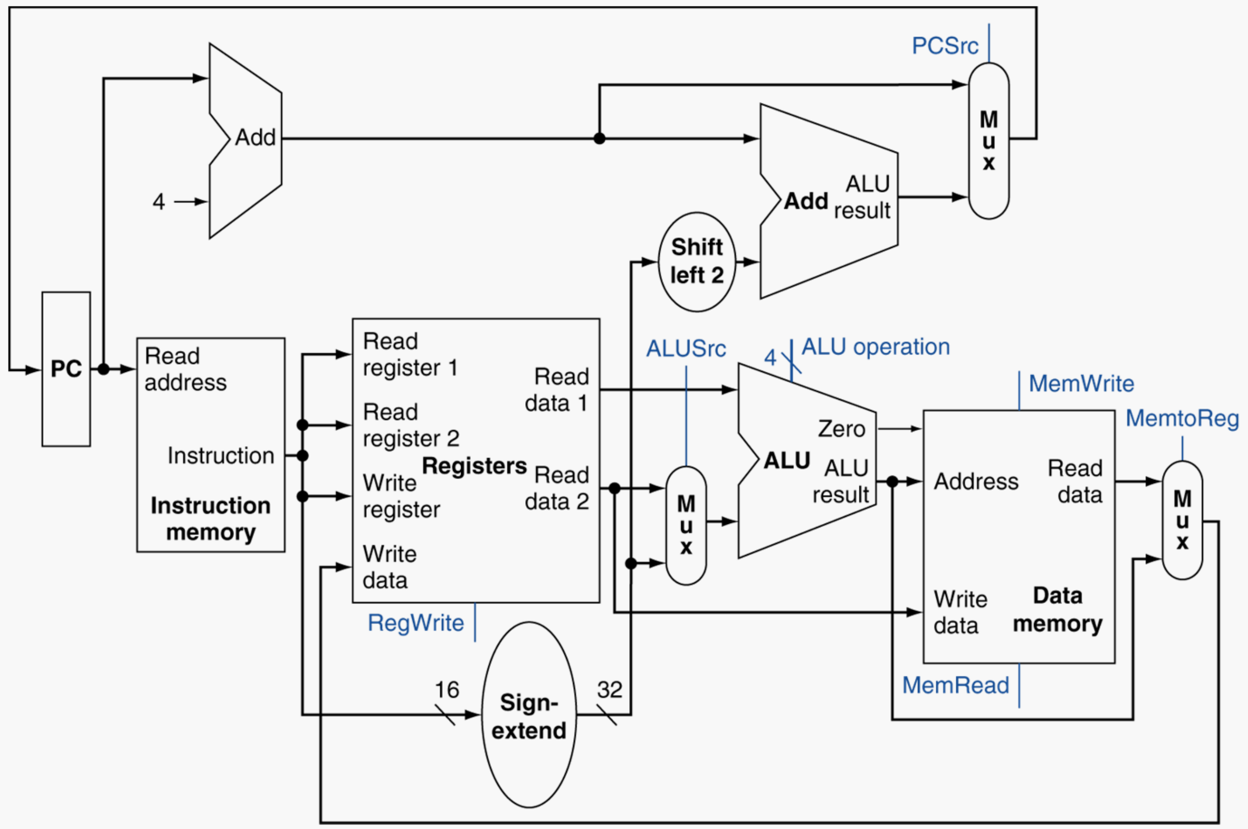


How would this execute: `lw $s0, 16($s1);`

100011	10000	10001	0000 0000 0010 0000
op	rs	rt	16-bit number

QTP Alert!!

Full Datapath



How would this execute: `beq $s0, $s1, Label0;`

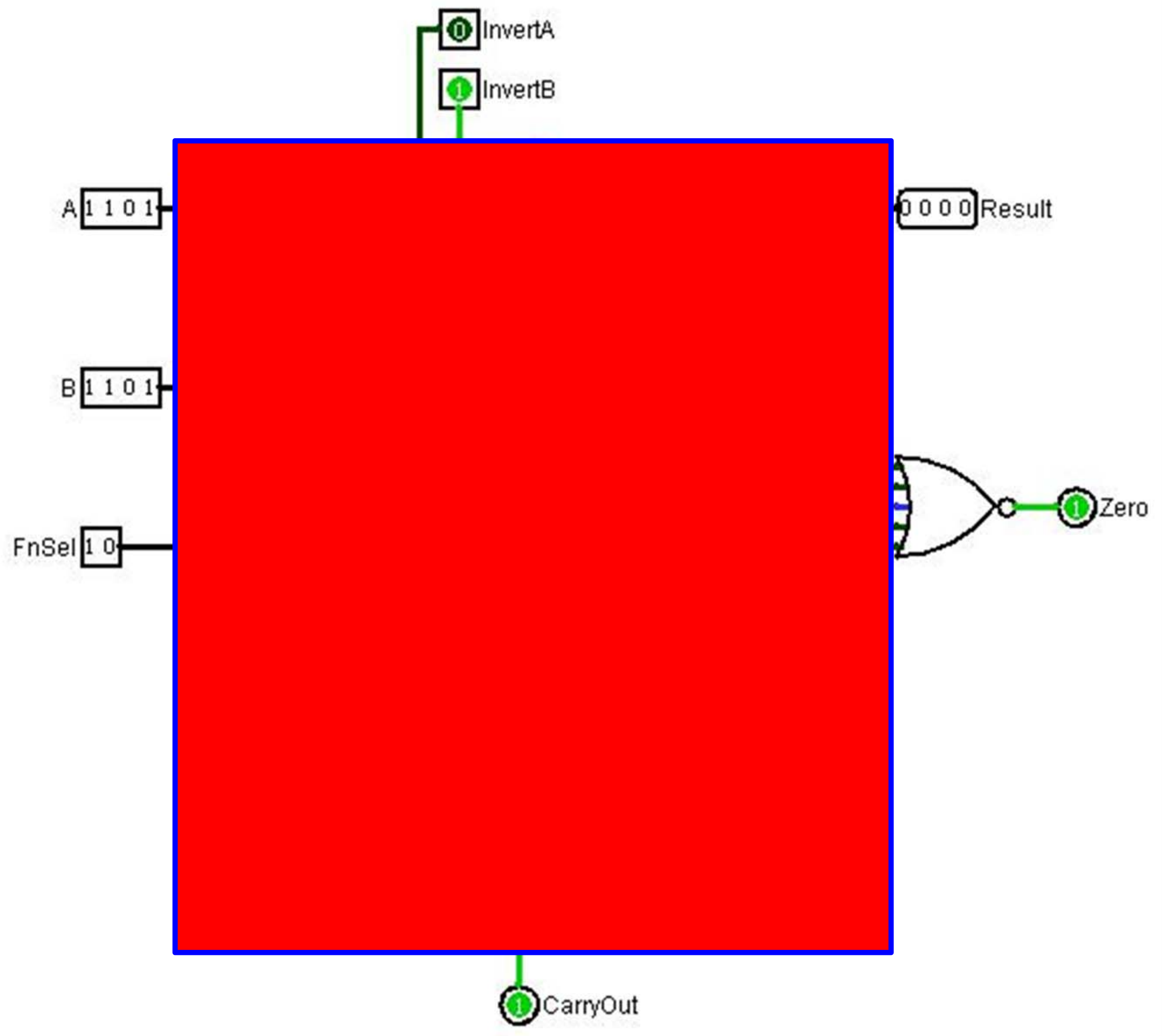
QTP Alert!!

Simplified ALU Design

Here's a simplified ALU for 4-bit operands that illustrates the control interface we need:

Effectively, this ALU has a 4-bit control mechanism for selecting the desired function.

InvA	InvB	FnSel	ALU Fn
0	0	00	AND
0	0	01	OR
0	0	10	add
0	1	10	sub
0	1	11	slt
1	1	00	NOR



©2006-09 McQuain, Feng & Ribbens

ALU Control

ALU used for

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

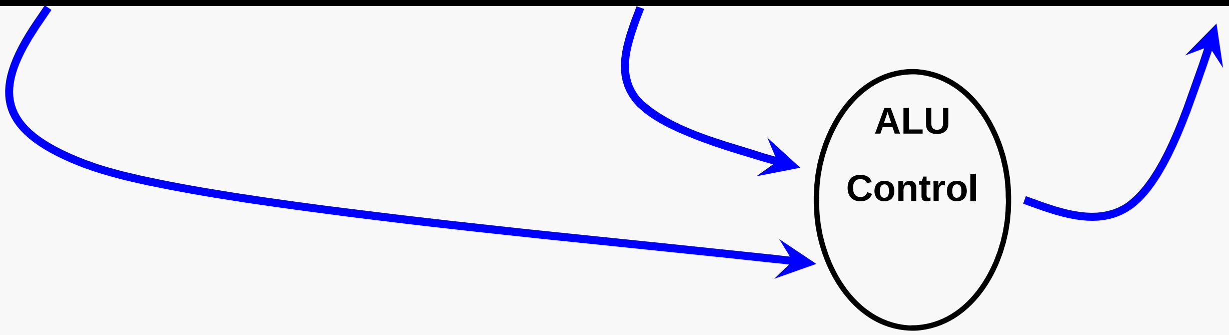
InvA	InvB	FnSel	ALU Fn
0	0	00	AND
0	0	01	OR
0	0	10	add
0	1	10	sub
0	1	11	slt
1	1	00	NOR

ALU Control

Assume 2-bit control signal ALUOp derived from opcode

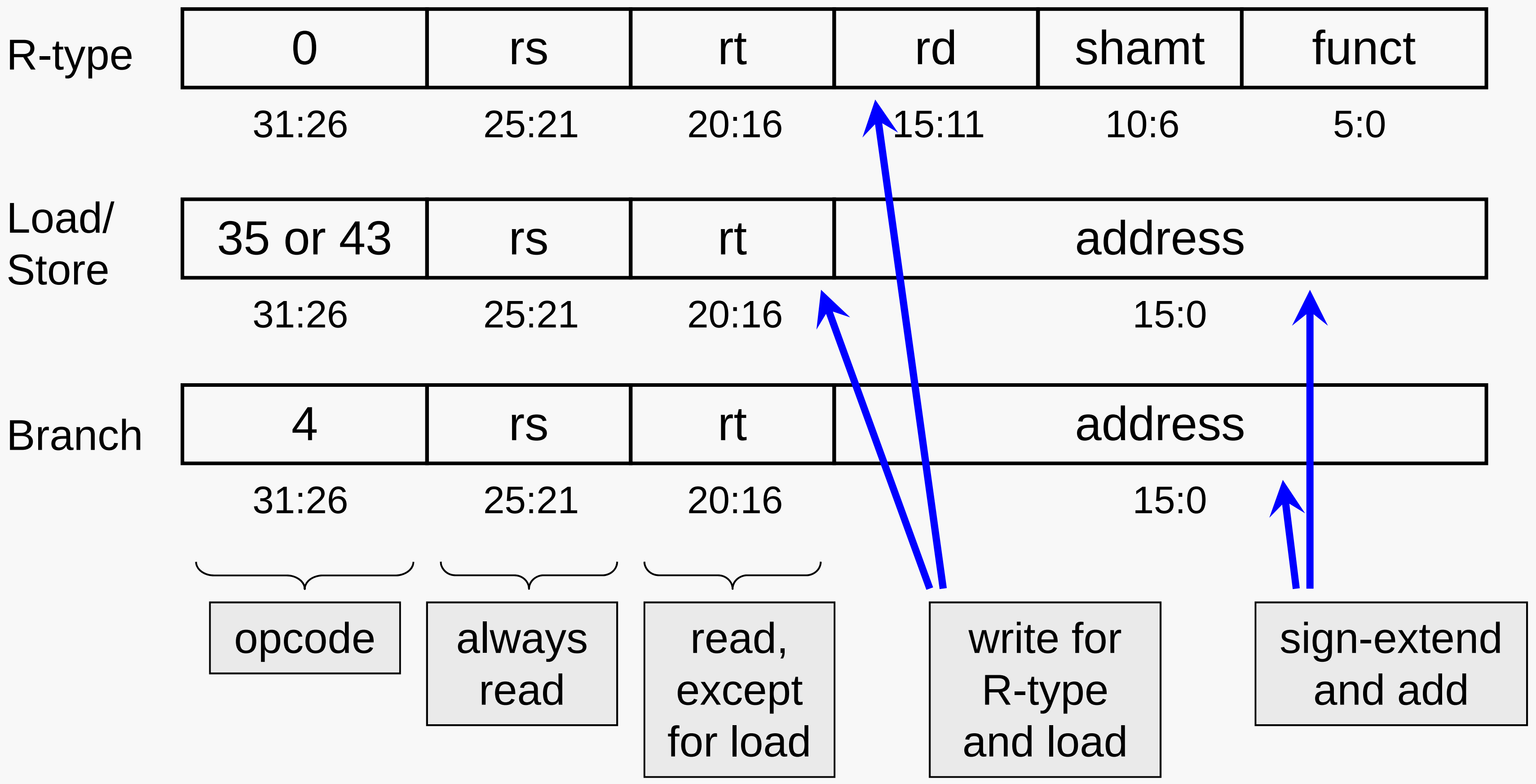
- Combinational logic derives ALU control

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

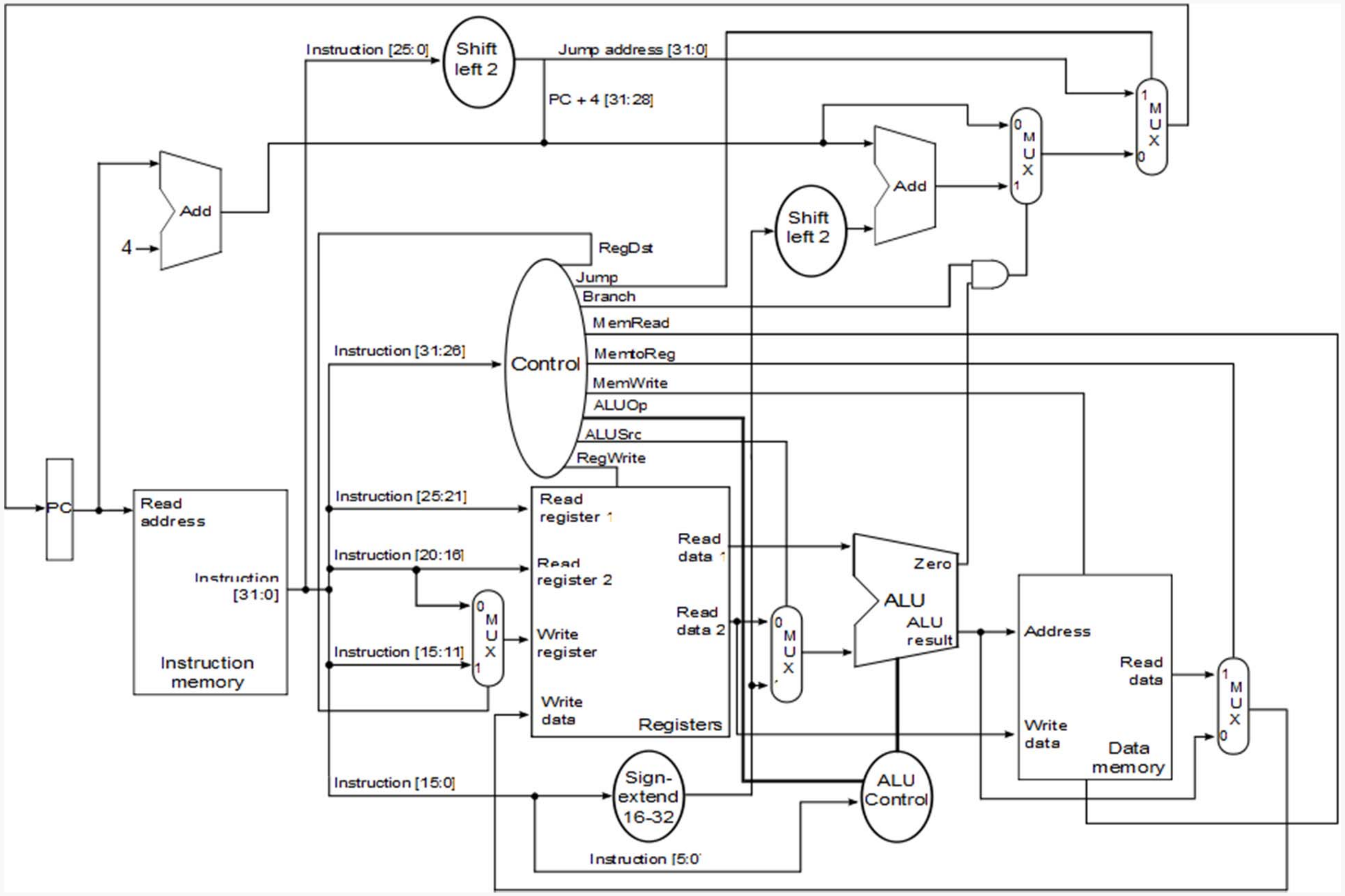


The Main Control Unit

Control signals derived from instruction

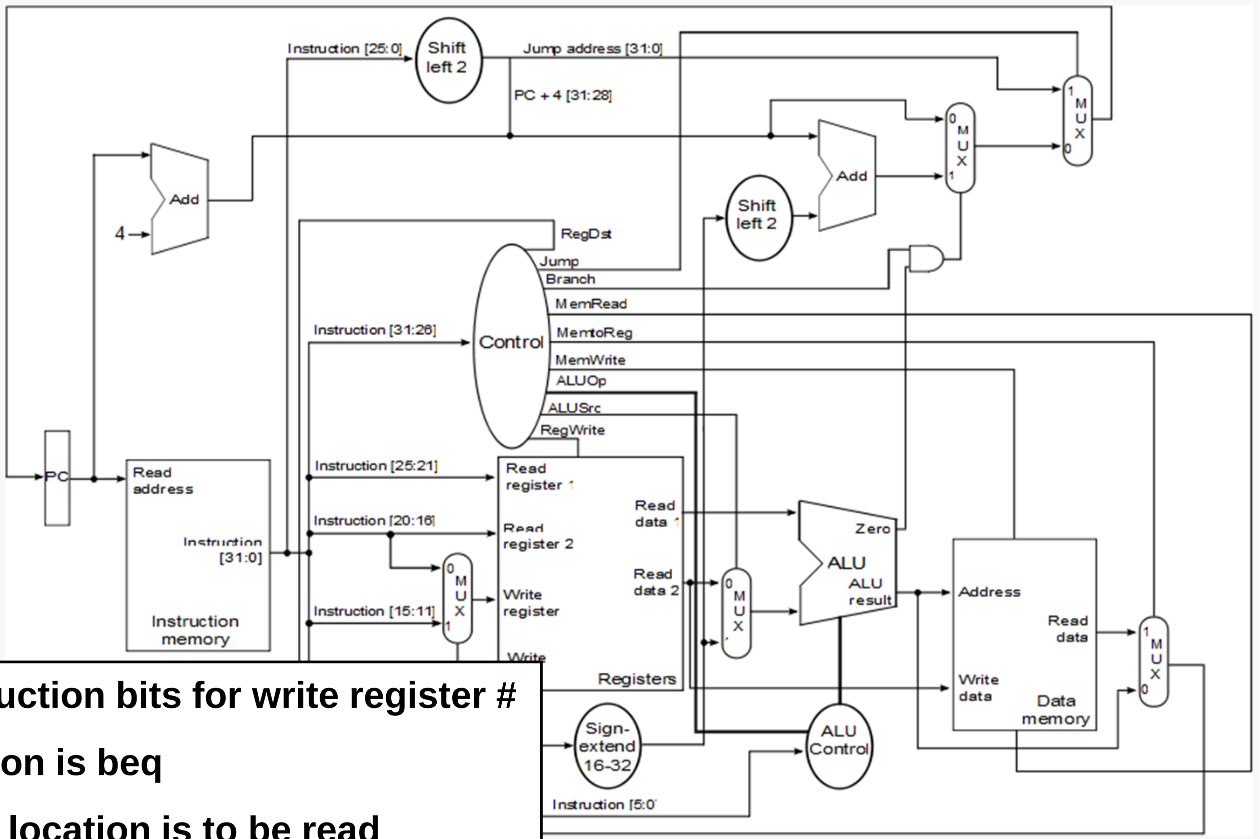


Datapath With Control



©2006-09 McQuain, Feng & Ribbens

Datapath Control Signals



RegDst	**selects instruction bits for write register #**
Branch	**1 iff instruction is beq**
MemRead	**1 iff memory location is to be read**
MemtoReg	**selects source for write data port**
ALUOp	**specifies instruction class to ALU control**
MemWrite	**1 iff memory location is to be written**
ALUSrc	**selects source for 2nd operand to ALU**
RegWrite	**1 iff register is to be written**

Main Control Truth Table

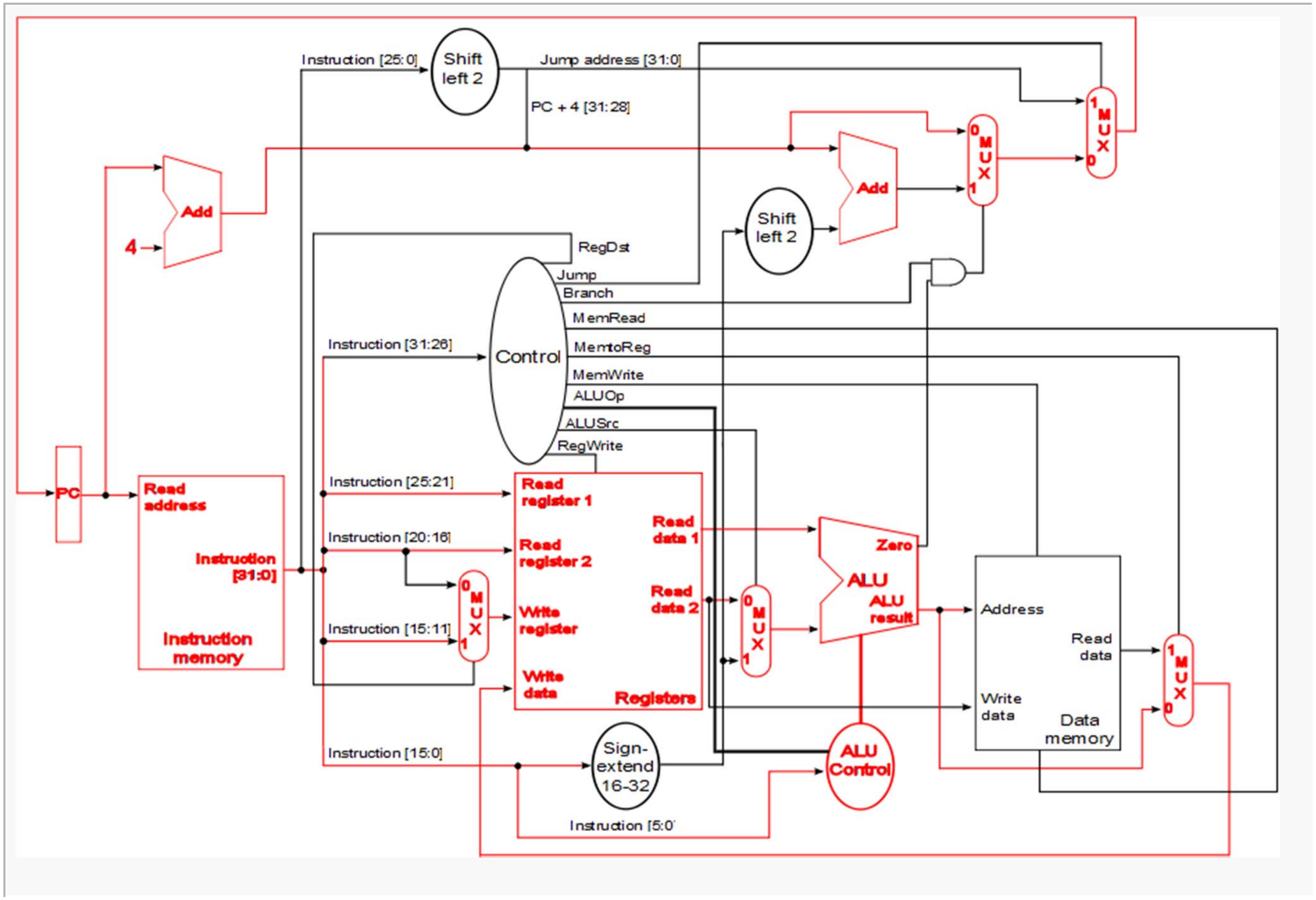
	Signal Name	R-format	lw	sw	beq
I N P U T	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
O U T P U T	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

Why are these don't-cares?

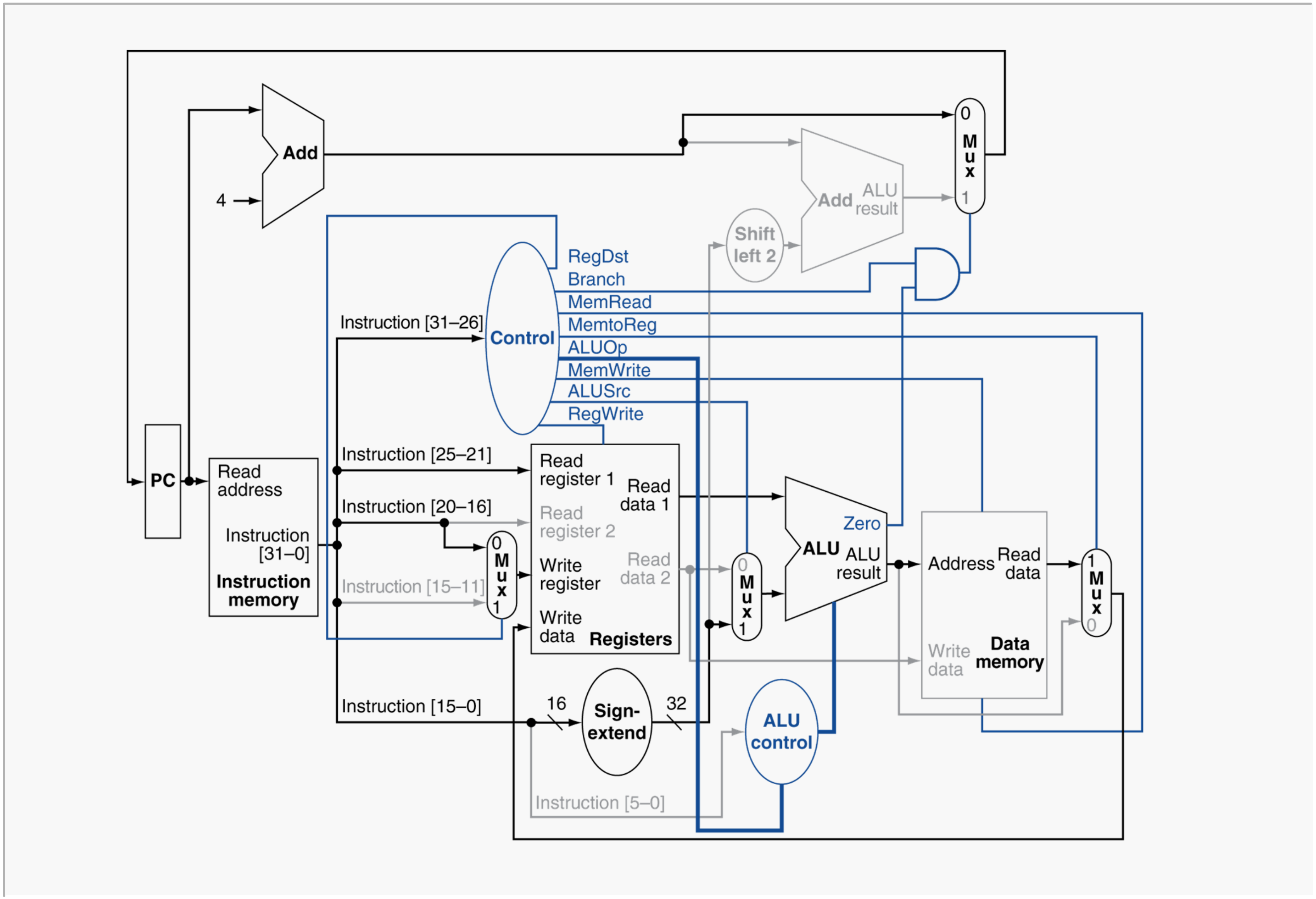
Why these values?

©2006-09 McQuain, Feng & Ribbens

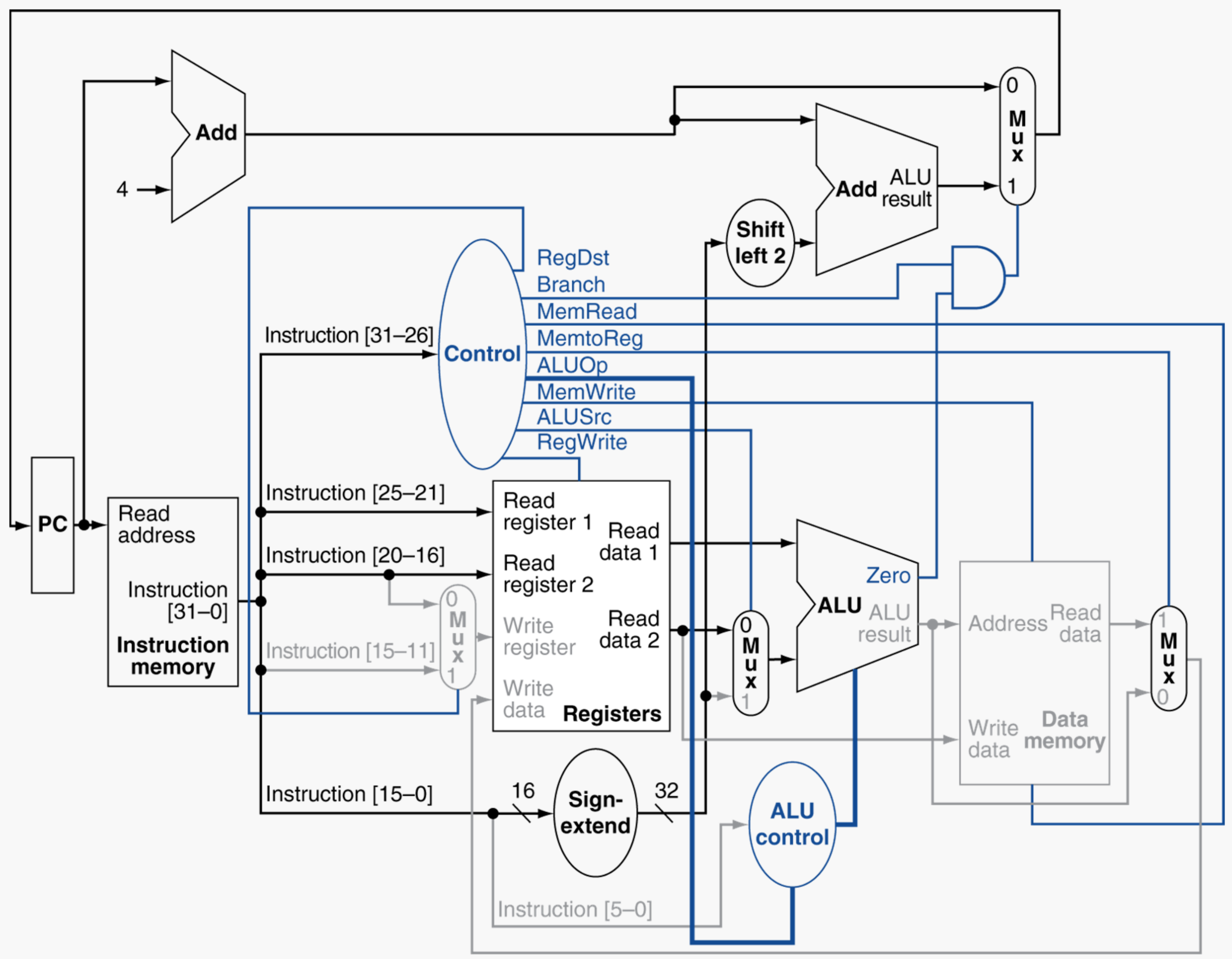
Relevancies for R-Type Instruction



Relevancies for Load Instruction



Relevancies for beq Instruction



©2006-09 McQuain, Feng & Ribbens

Implementing Jumps

Jump	2	address
	31:26	25:0

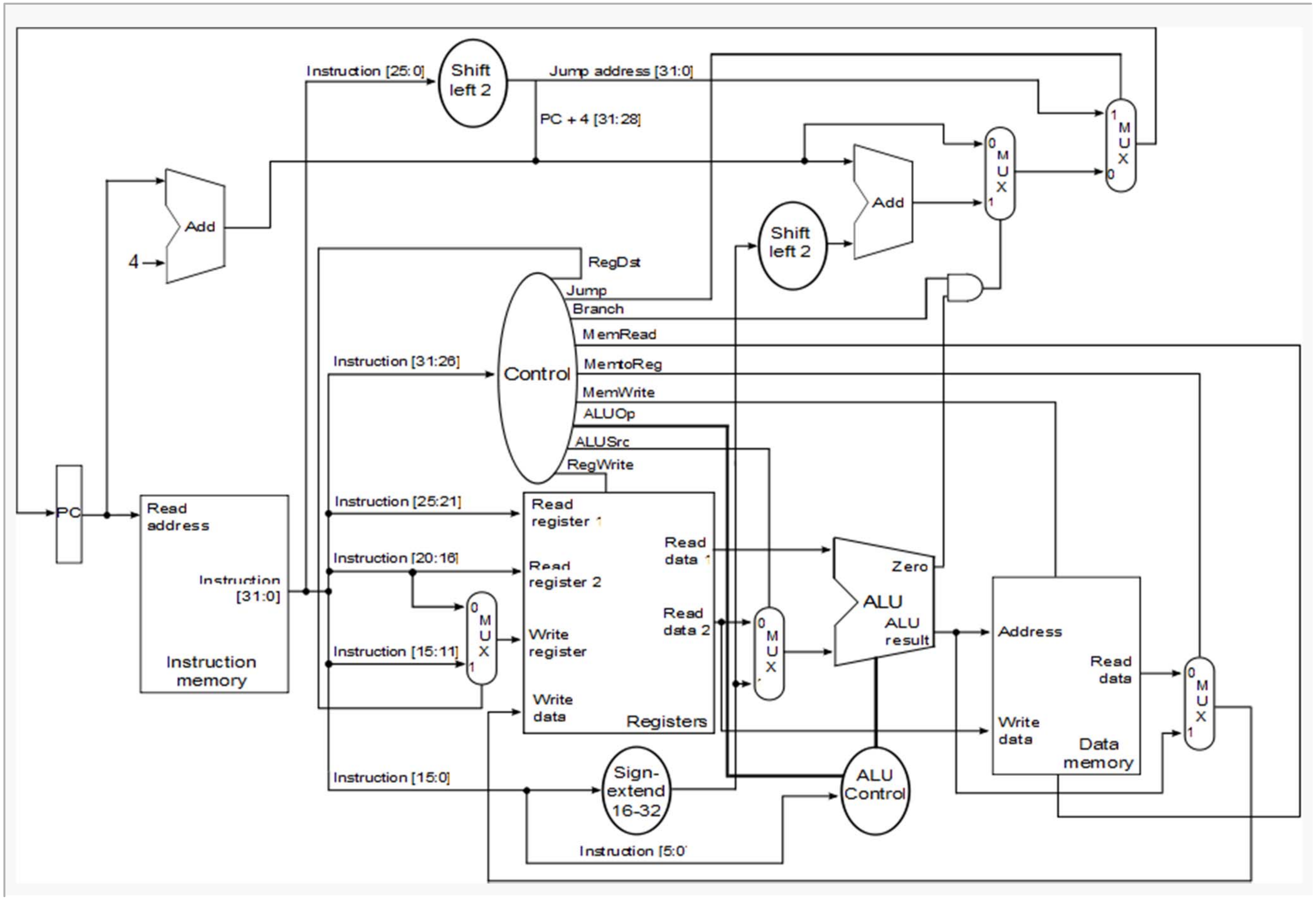
Jump uses word address

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



©2006-09 McQuain, Feng & Ribbens

Performance Issues

Longest delay determines clock period

- Critical path: load instruction
- Instruction memory → register file → ALU → data memory → register file

Not feasible to vary the clock period for different instructions

Violates design principle

- Making the common case fast

We will improve performance (in 2506) by pipelining