The Processor: Control

CSCM601252 - Pengantar Organisasi Komputer

Instructor: Erdefi Rakun + Tim Dosen POK

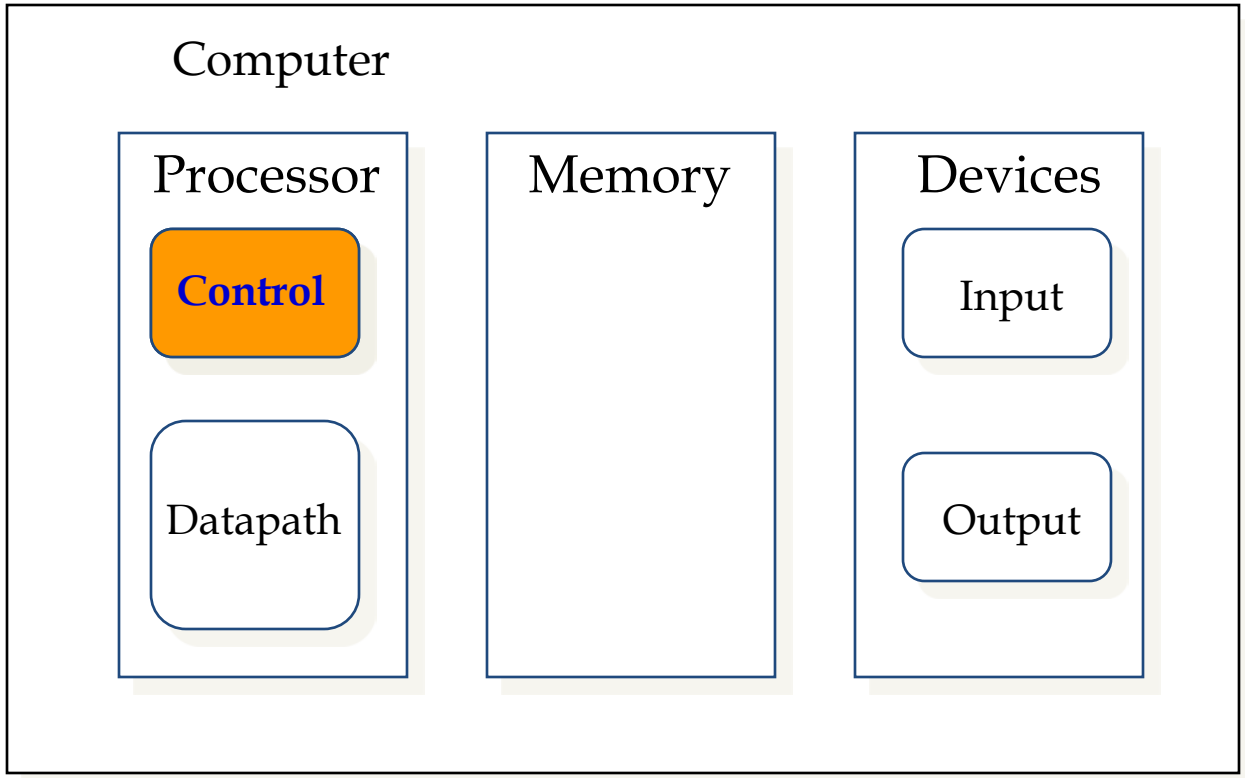
Fasilkom UI

Outline

- ALUcontrol signals
- Design of ALU Control Unit
- Control Signals
  - PCSrc
  - RegDst
  - ALUSrc
  - MemtoReg
  - MemRead
  - MemWrite
  - RegWrite

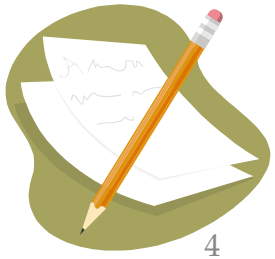
Note: These slides are taken from Aaron Tan's slide

Today's Focus

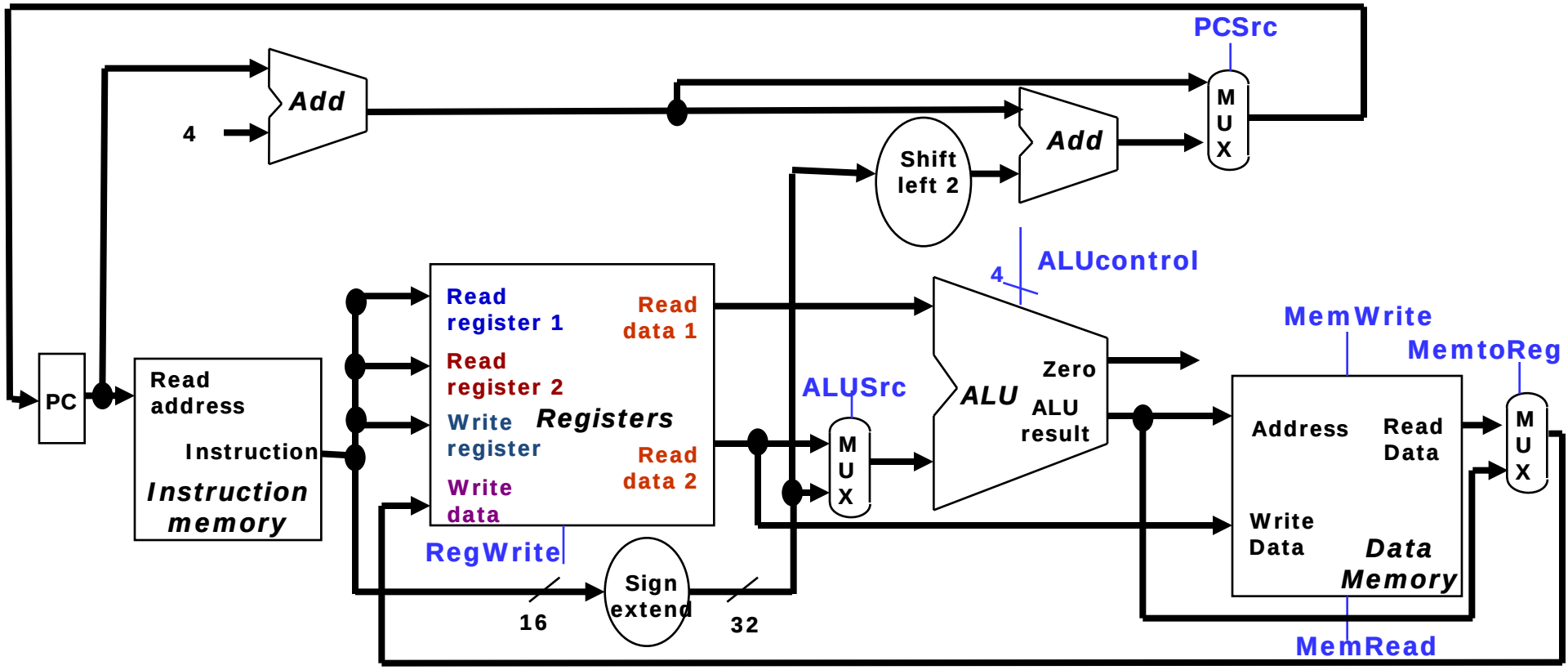


DATAPATH versus CONTROL

- Datapath: The collection of different elements that together provide a conduit for the flow and transformation of data in the processor during execution
- Control: Tells the datapath, memory, and I/O devices what to do according to the wishes of the instructions of the program
- Let's identify all control signals first!
- Control signals will be generated from machine language instructions
- Get ready with paper and pencil; we will do in-class exercises



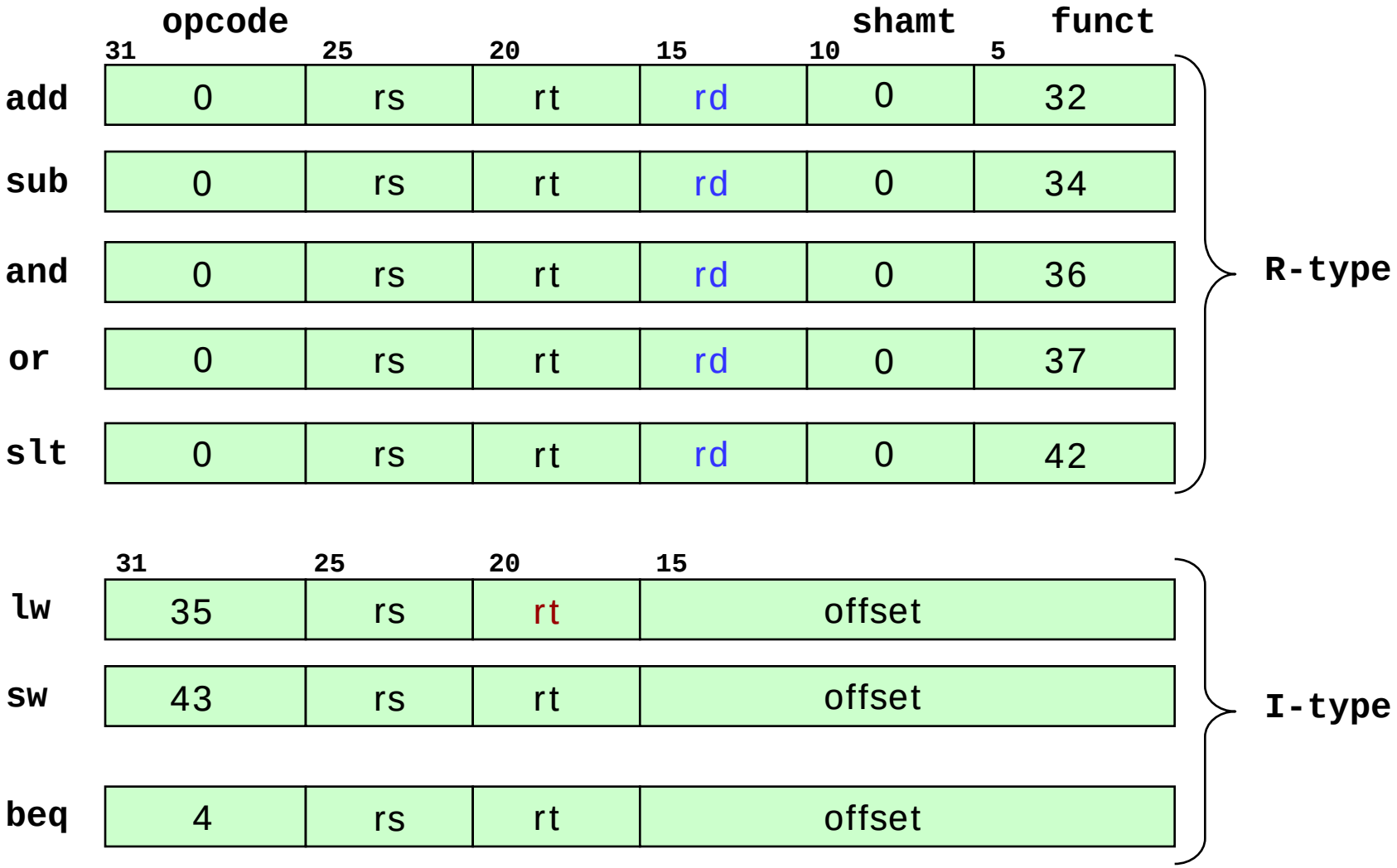
Review: Datapath



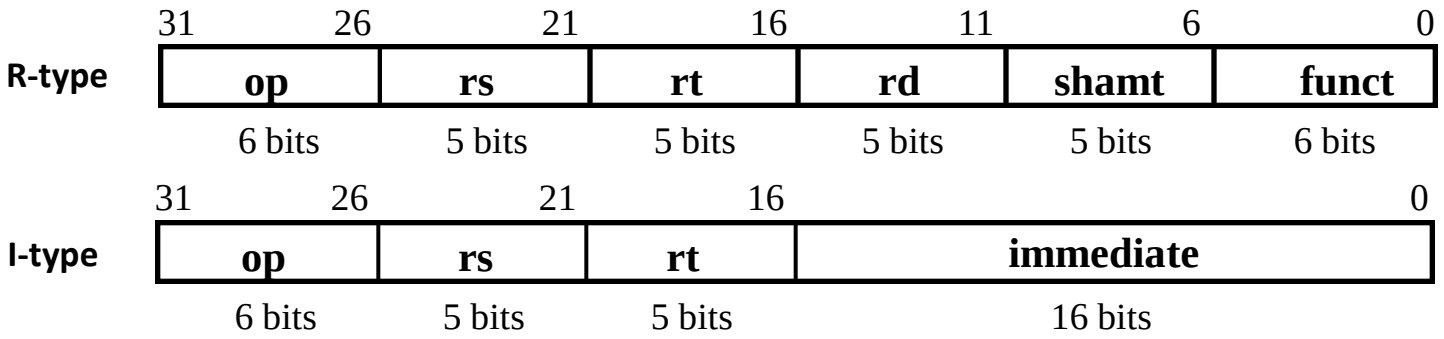
Blue lines are control signals.

Are there any more control signals?

Review: MIPS Instruction Subset

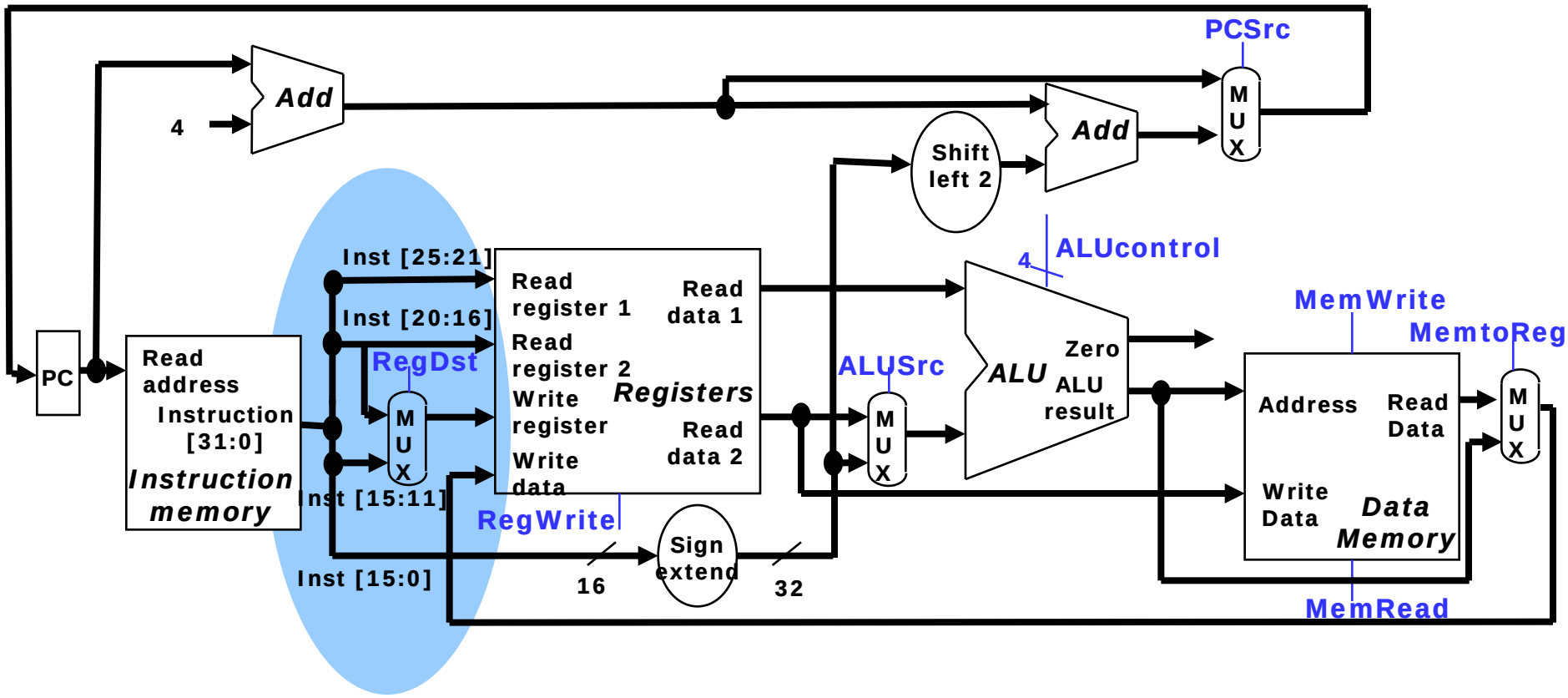


Review: Instruction Format



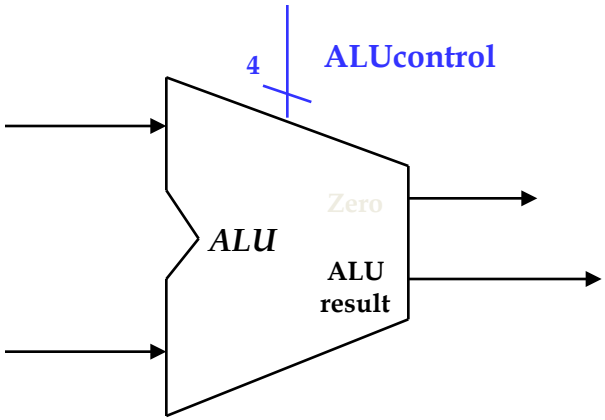
- Opcode field (Op[5:0]) is always in bits 31:26
- Two registers to be read are always specified through rs and rt fields at positions 25:21 and 20:16 for R-type instructions, 'branch equal', and for 'store' instruction
- 16-bit offset field for 'branch equal', 'load', 'store' is always in positions 15:0
- Problem with destination register: 20:16 (rt) for 'load' while 15:11 (rd) for R-type instructions ➔ use MUX

Datapath With Operands Decoding



More About ALUcontrol Signals

- Used by R-type, 'load/store' and 'branch equal' to instructions
- **4-bit ALUcontrol** is generated from
  - 6-bit funct field: Instruction[5:0] (only for R-type instructions)
  - 2-bit control field called **ALUOp**
    - ALUOp = 00 ➔ address calculation (add) for 'load/store'
    - ALUOp = 01 ➔ register comparison (subtract) for 'branch equal to'
    - ALUOp = 10 ➔ ALU operation determined by funct field Instruction[5:0]



NOR will not be considered any more

ALUcontrol	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	slt
1100	NOR

2-Level Implementation (1/2)

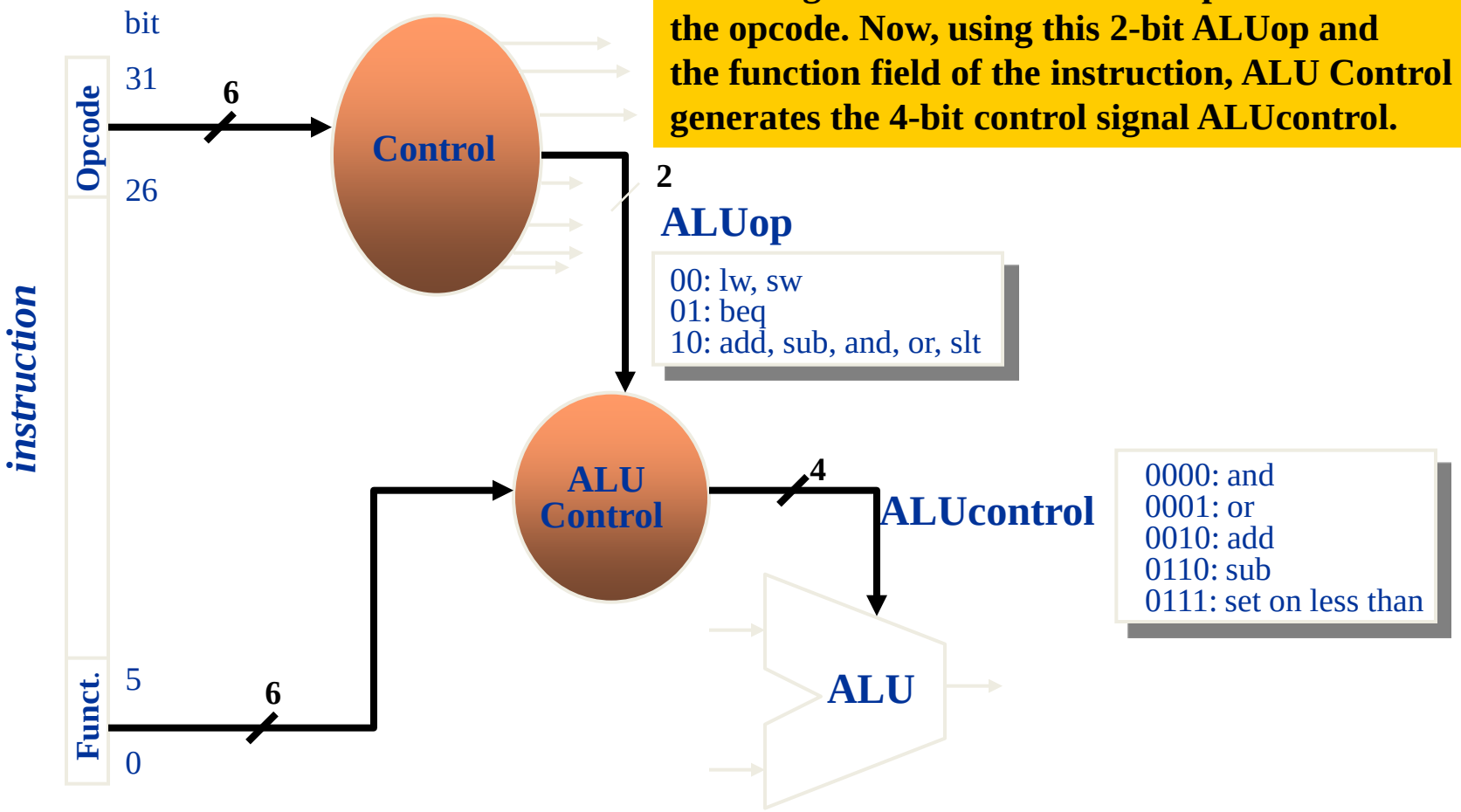
Opcode	ALUop	Instruction Operation	Funct field	ALU action	ALU control
lw		load word		add	
sw		store word		add	
beq		branch equal		subtract	
R-type		add		add	
R-type		subtract		subtract	
R-type		AND		AND	
R-type		OR		OR	
R-type		set on less than		set on less than	

ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	slt
1100	NOR

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

2-Level Implementation (2/2)

Control generates the 2-bit ALUop based on the opcode. Now, using this 2-bit ALUop and the function field of the instruction, ALU Control generates the 4-bit control signal ALUcontrol.



ALUcontrol will determine the function that the ALU will perform (ADD, OR, etc.)

Design Of ALU Control Unit (1/2)

- Input: 6-bit Funct field and 2-bit ALUop
- Output: 4-bit ALUcontrol
- Describe using Truth Table

ALUop		Funct Field						ALU control
ALUop1	ALUop0	F5	F4	F3	F2	F1	F0	

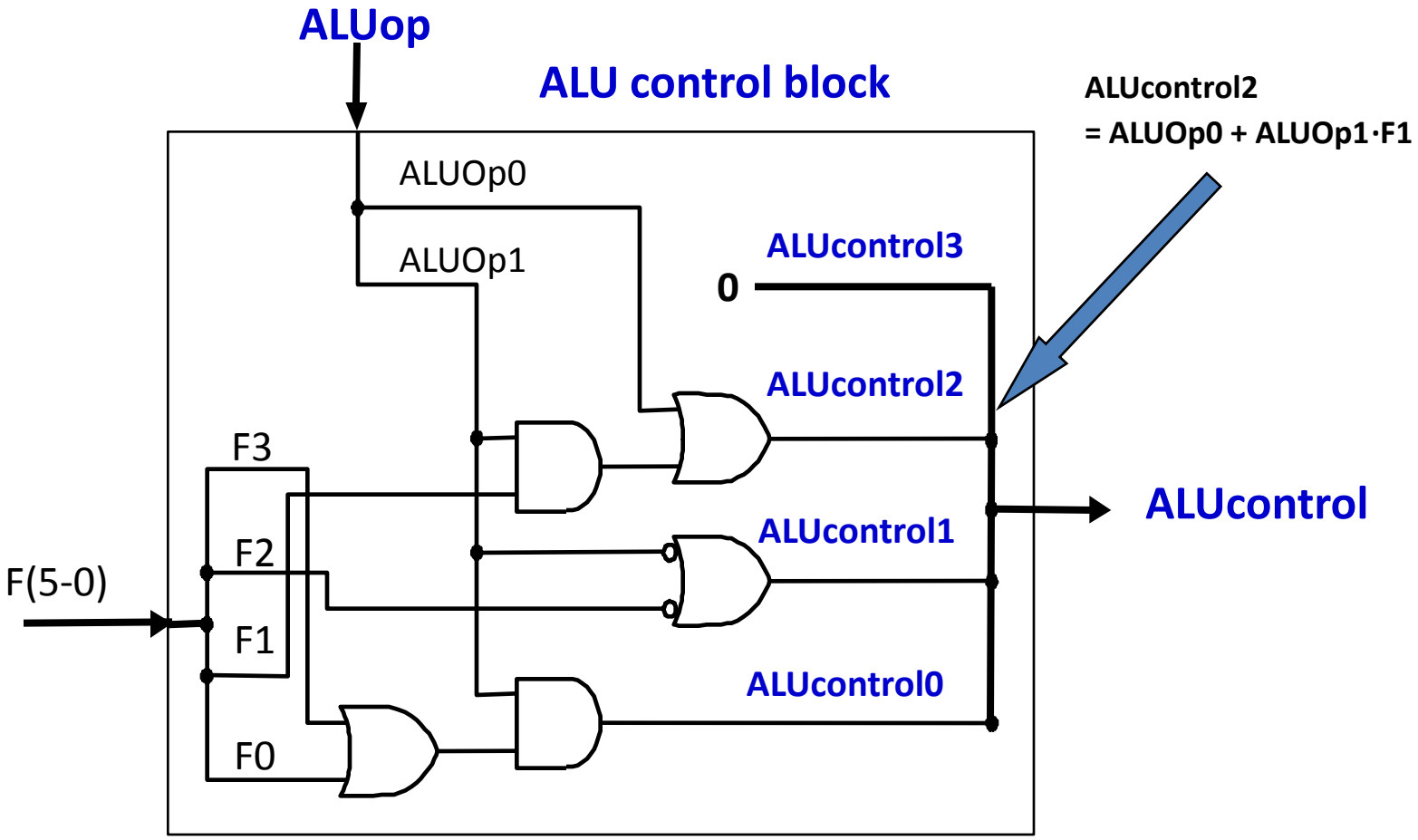
Op code	ALUop		Funct field	ALU action	ALU control
lw	00	load	XXXXXX	add	0010
sw	00	store	XXXXXX	add	0010
beq	01	beq	XXXXXX	sub	0110
R-type	10	add	100000	add	0010
R-type	10	sub	100010	sub	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	slt	101010	slt	0111

We will see how to set ALUop later

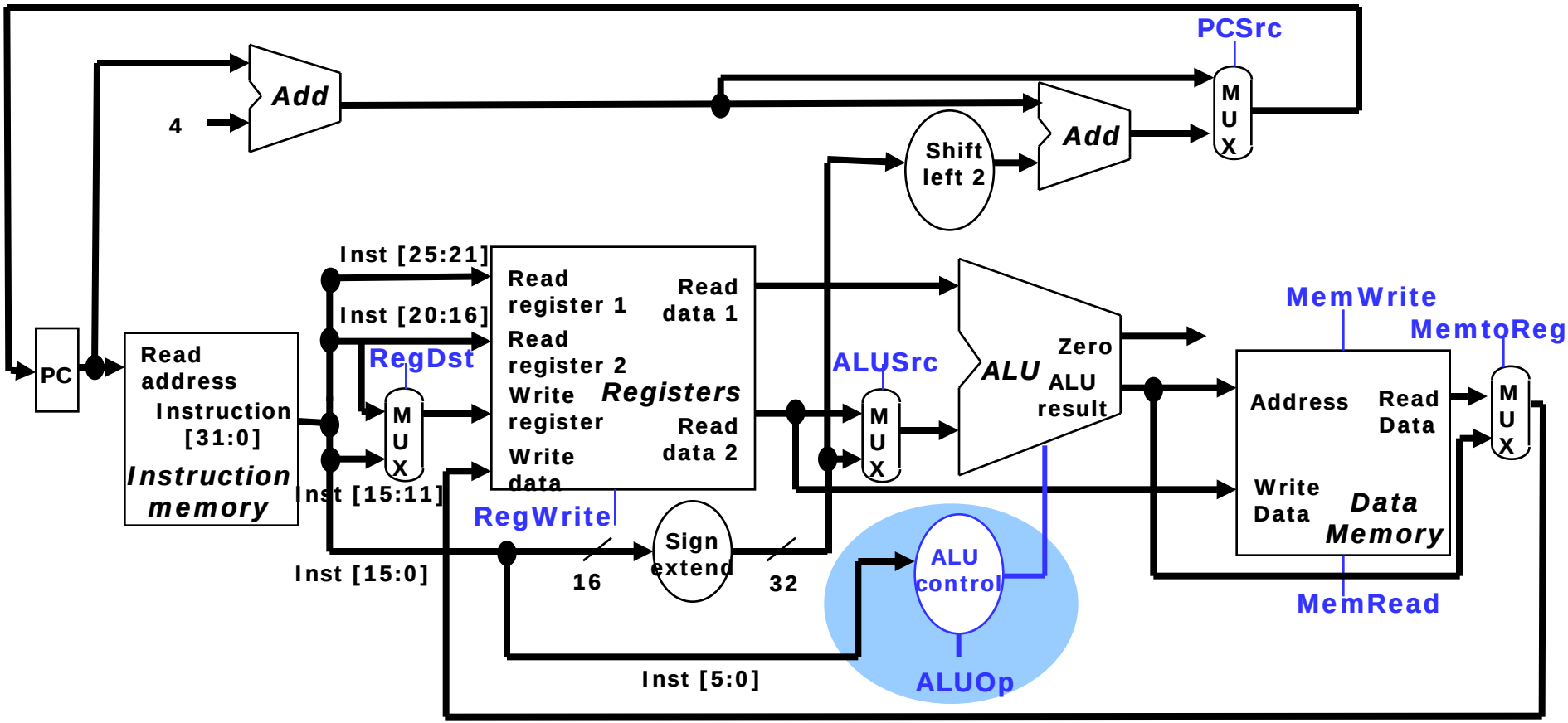
ALUOp		Funct Field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	0 0 1 0
0	1	X	X	X	X	X	X	0 1 1 0
1	0	X	X	0	0	0	0	0 0 1 0
1	0	X	X	0	0	1	0	0 1 1 0
1	0	X	X	0	1	0	0	0 0 0 0
1	0	X	X	0	1	0	1	0 0 0 1
1	0	X	X	1	0	1	0	0 1 1 1

Design Of ALU Control Unit (2/2)

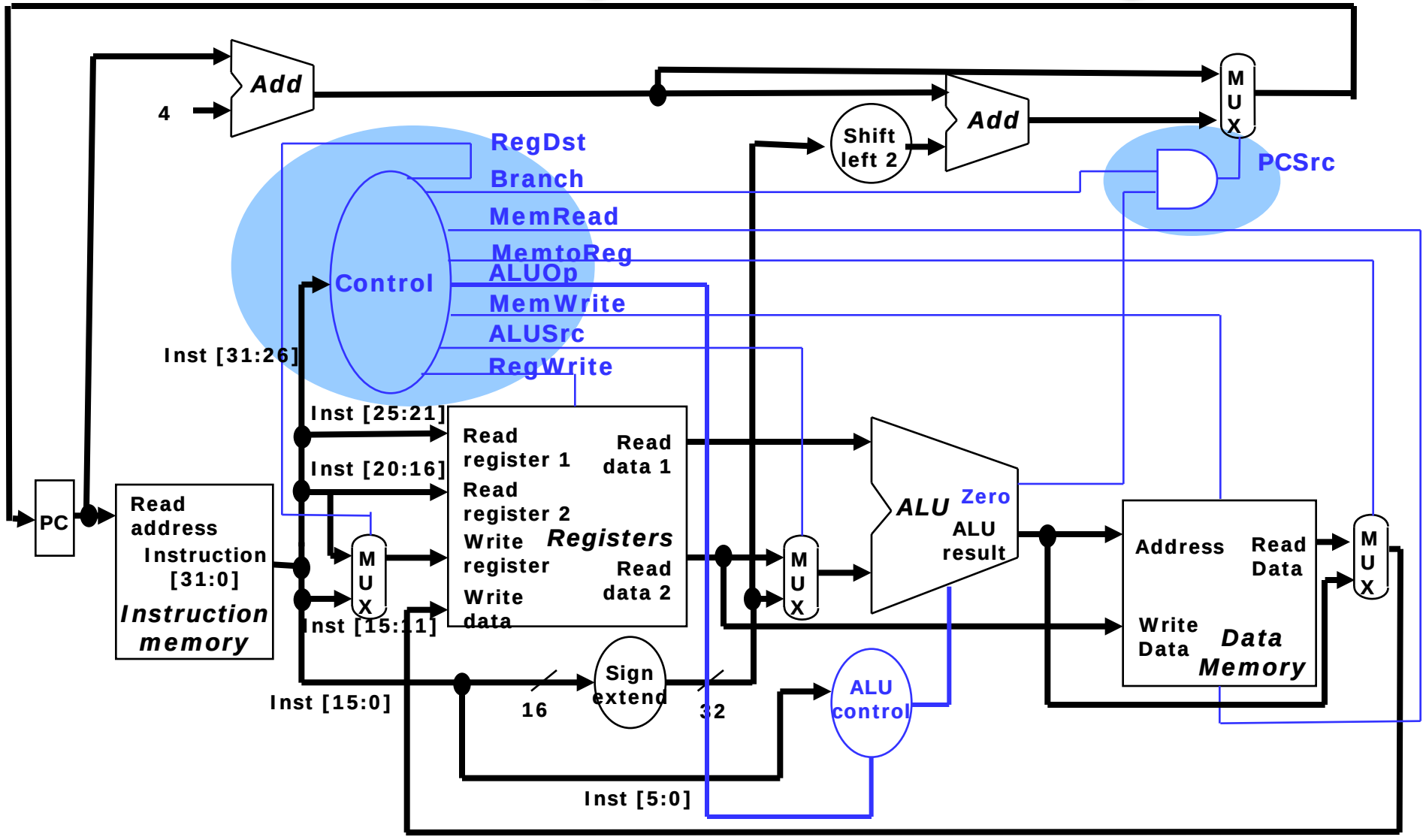
- Simple combinational logic



Datapath With All Control Signals

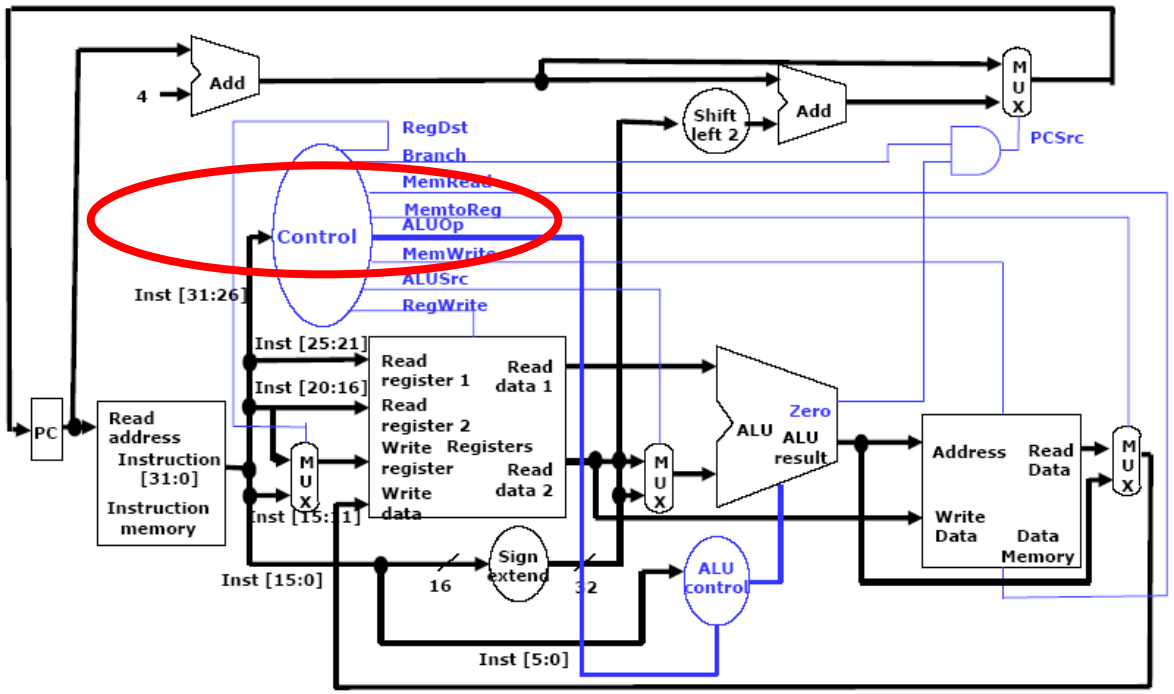


Consolidating All Control Signals



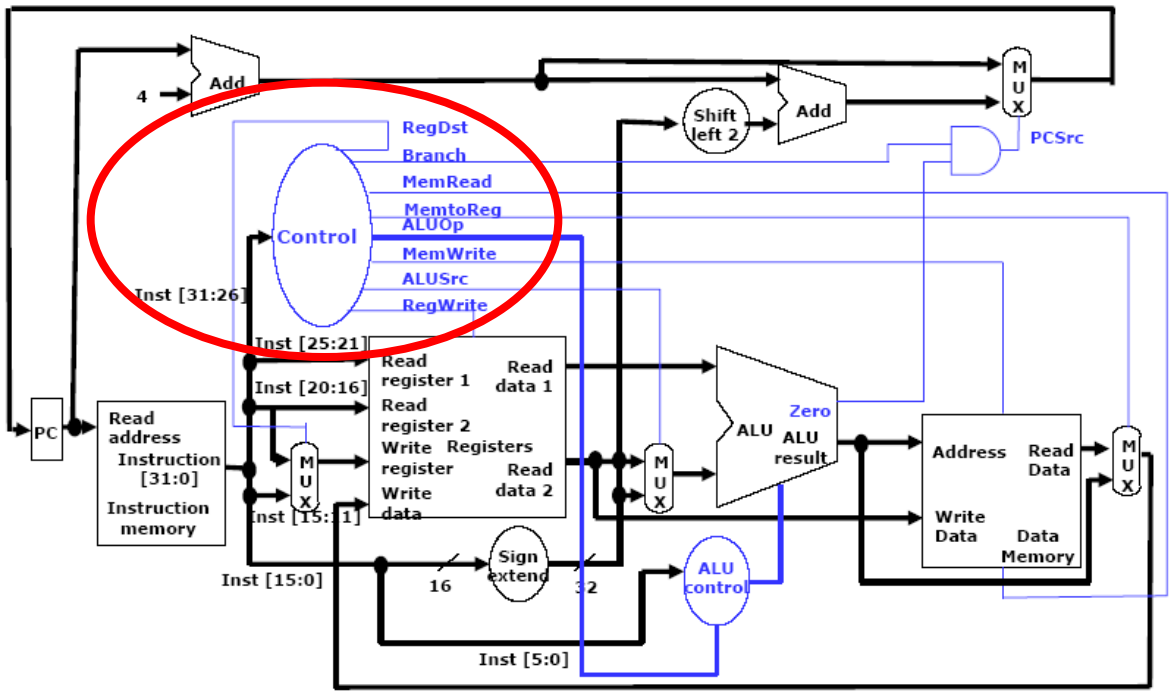
The Control Unit (1/2)

- Generates control signals: RegDst, Branch, MemRead, MemtoReg, ALUOp, MemWrite, ALUSrc, RegWrite
- Uses Op field [31:26]



The Control Unit (2/2)

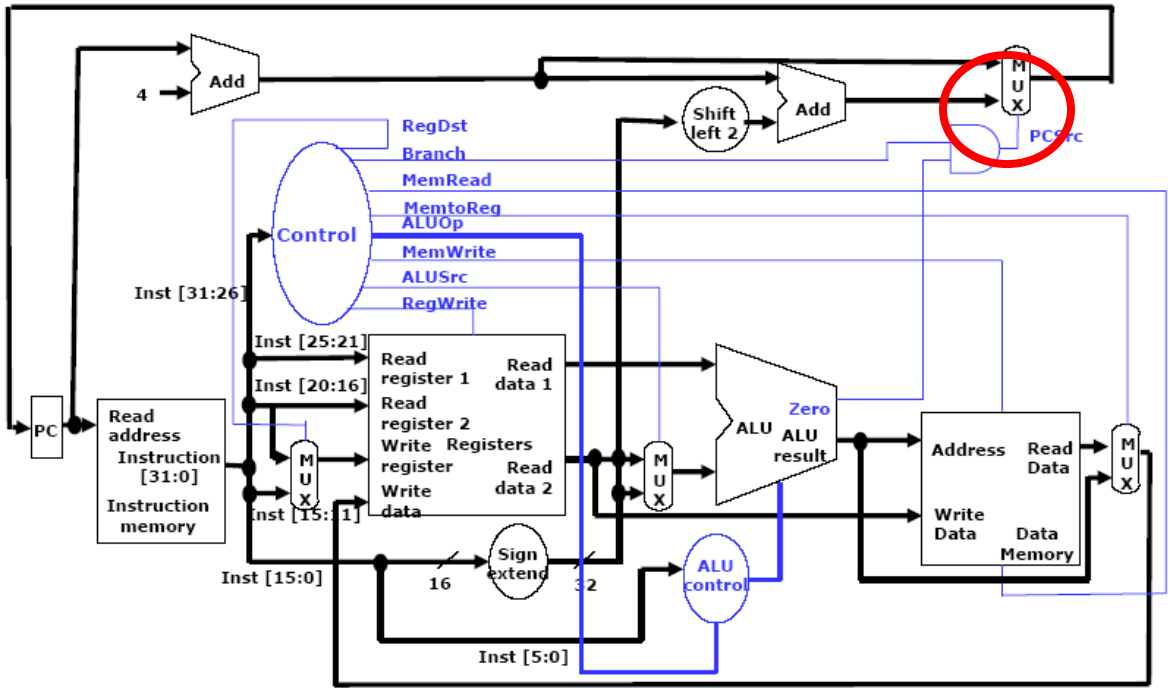
- Only one control signal cannot be generated simply from the instruction fields – can you detect which one?



Control Signals: PCSrc

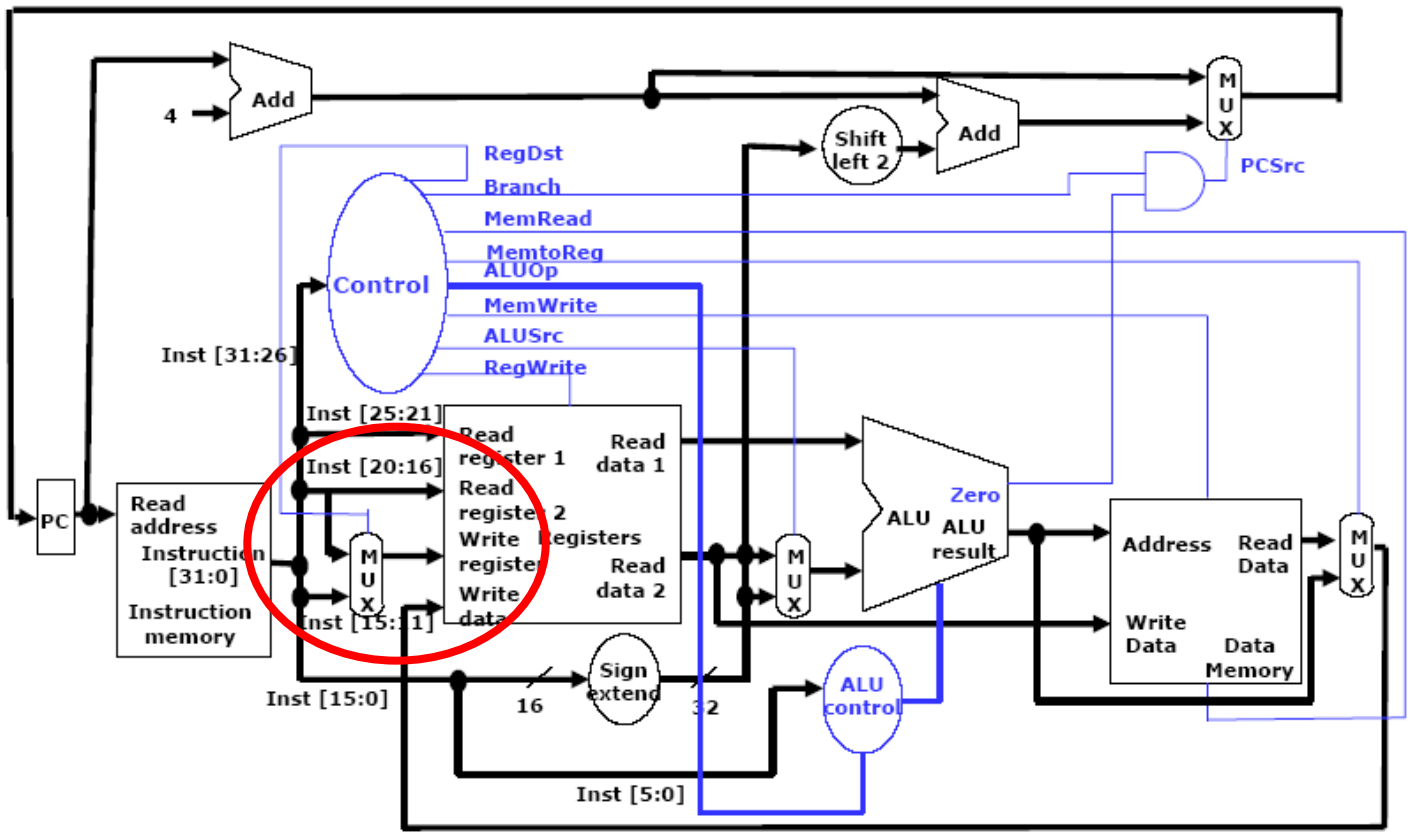
- True: PC = SignExt(Inst[15:0]) << 2 + (PC + 4)
- False: PC = PC + 4
- PCSrc = Branch AND Zero
  (Branch from Control, Zero from ALU)

MUX selector
True ≅ Asserted ≅ lower input
False ≅ Deasserted ≅ upper input



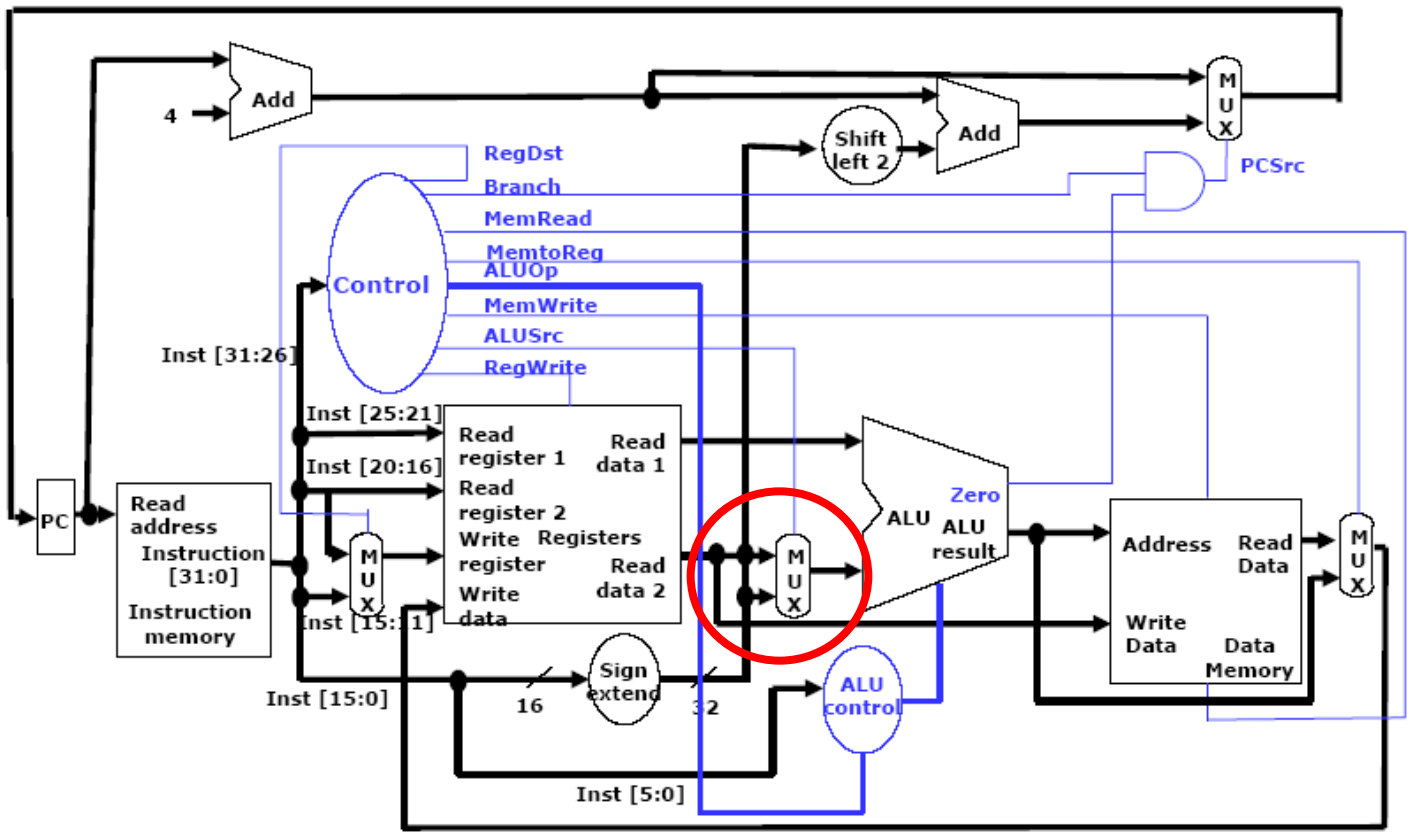
Control Signals: RegDst

- True: Write register = Inst[15:11]
- False: Write register = Inst[20:16]



Control Signals: ALUSrc

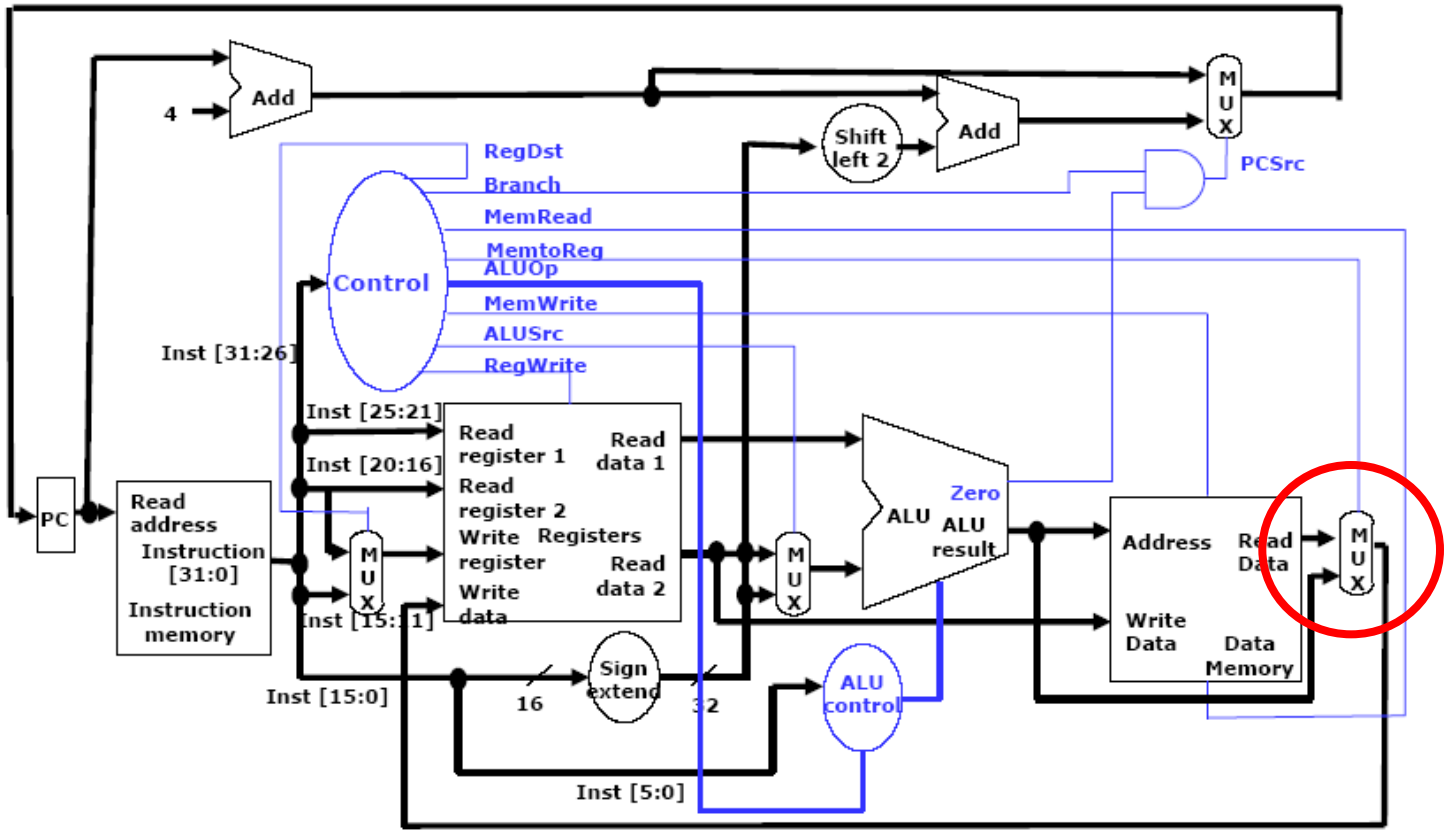
- True: second ALU operand= SignExt(Inst[15:0])
- False: second ALU operand = Register read data 2



Control Signals: MemtoReg

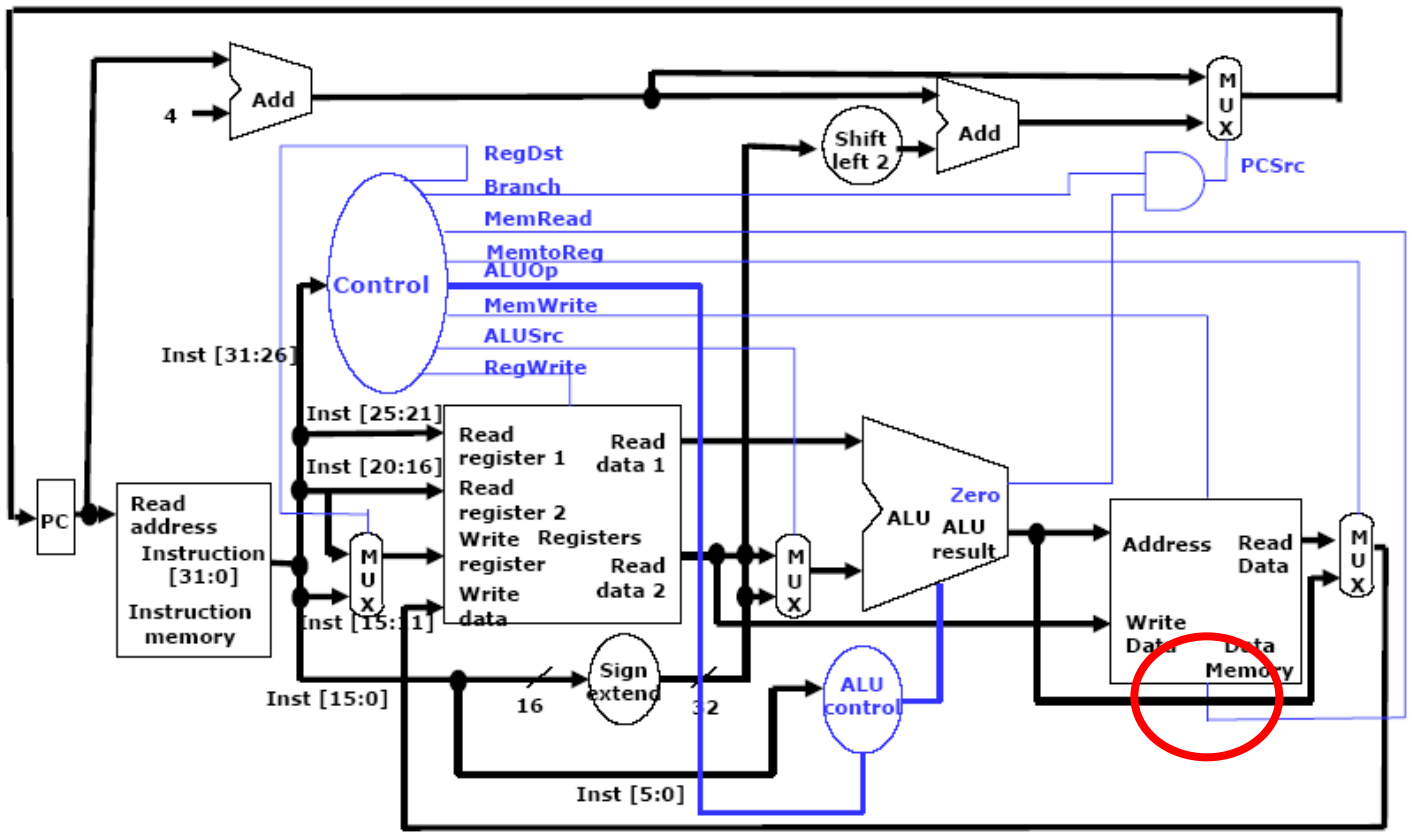
- True: Register write data = Memory read data
- False: Register write data = ALU result

MUX inputs are assumed to be inverted here. True → upper input, False → lower input



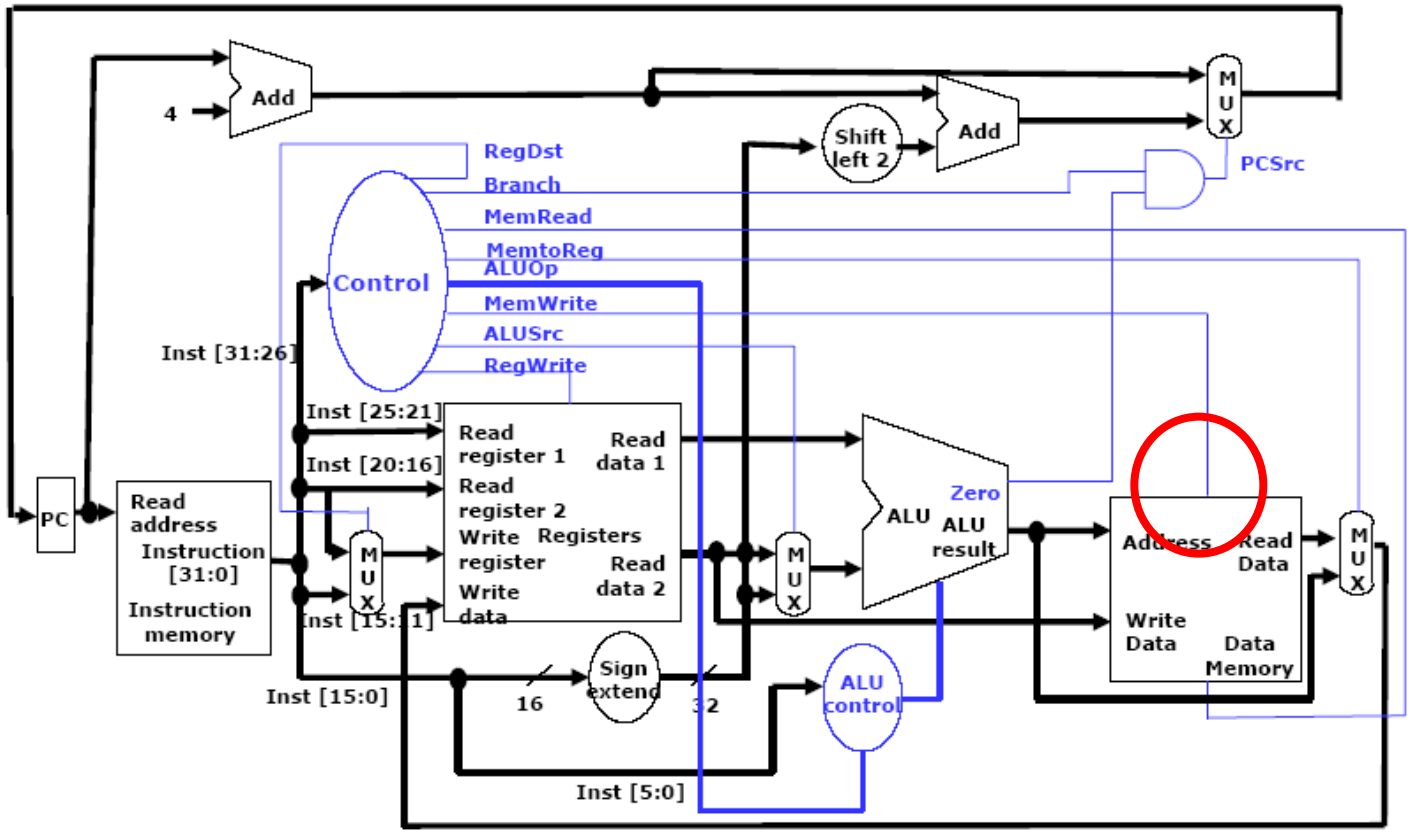
Control Signals: MemRead

- True: Read from memory using Address
- False: None



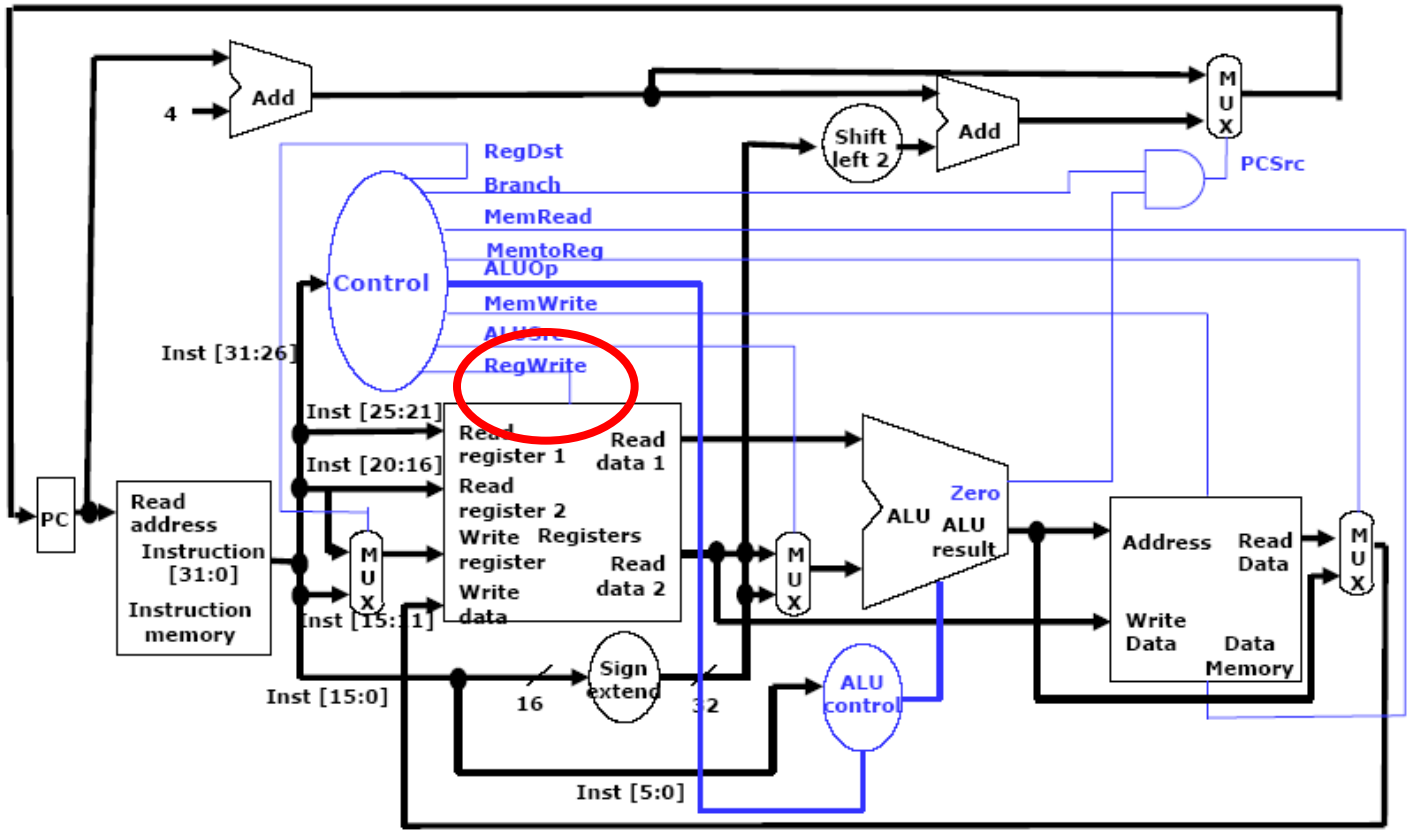
Control Signals: MemWrite

- True: Write to memory data read from register (Read data 2) using Address
- False: None



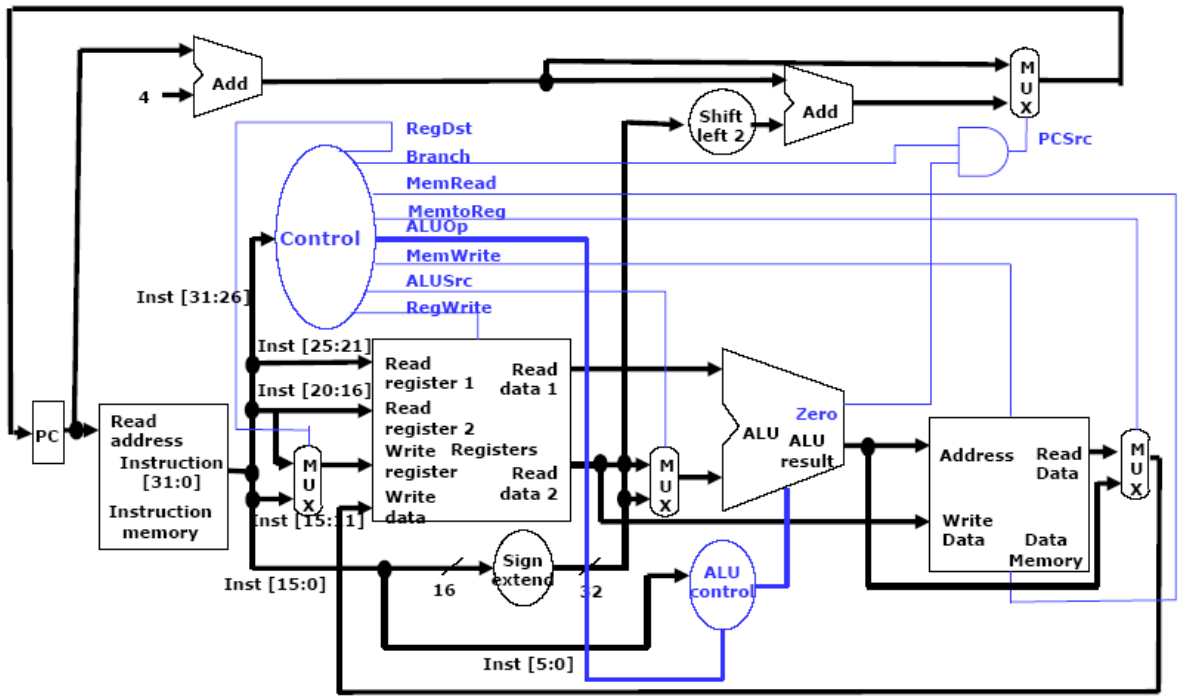
Control Signals: RegWrite

- True: Write "Write data" into register number "Write register"
- False: None



Control Design: Outputs

	RegDst	ALUSrc	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	ALUop1	ALUop0
R-type									
lw									
sw									
beq									



Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Control Design: Inputs

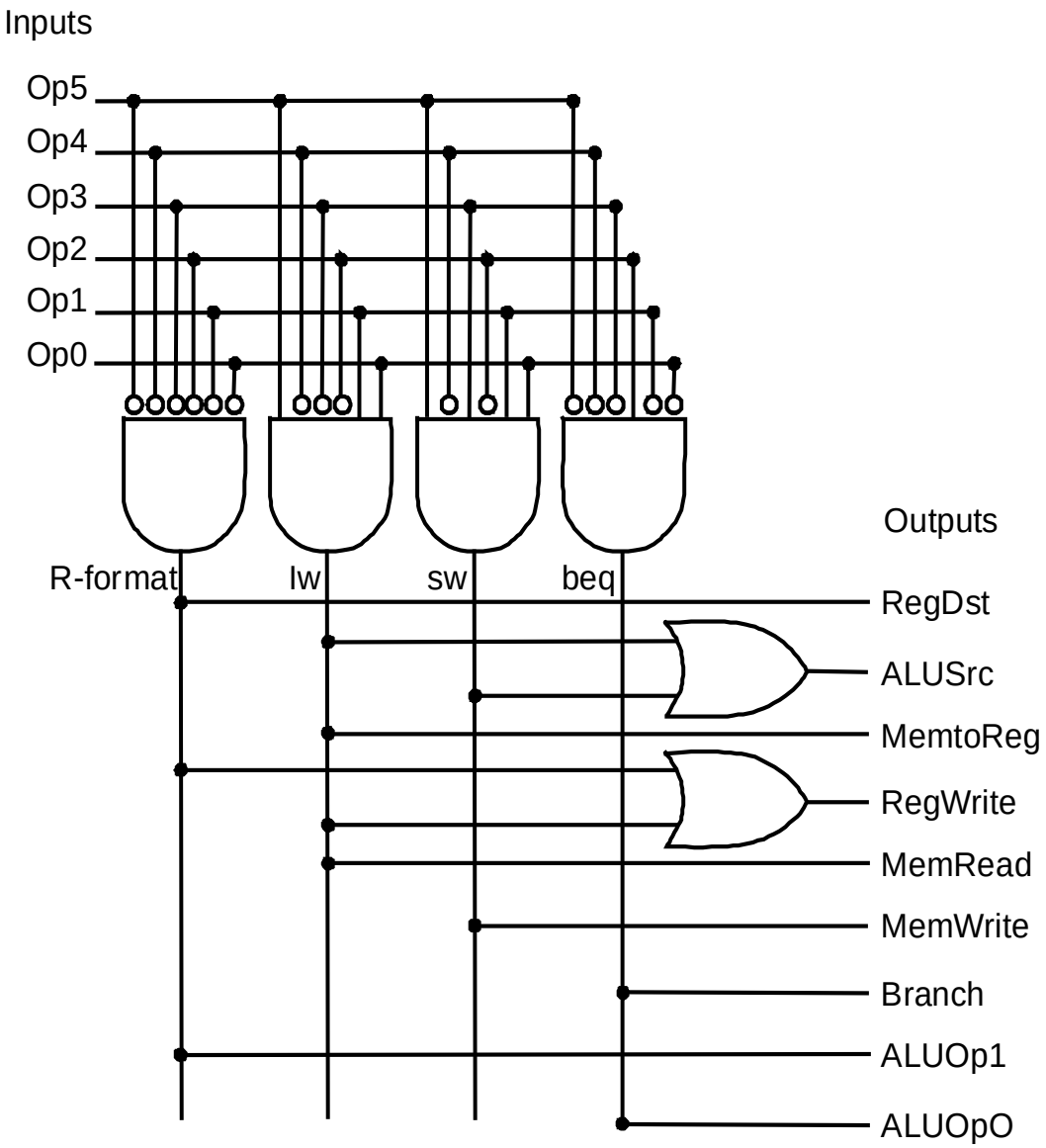
Opcode= Inst[31:26]	Op5= Inst[31]	Op4 = Inst[30]	Op3= Inst[29]	Op2= Inst[28]	Op1= Inst[27]	Op0= Inst[26]	Value
R-type							0_{16}
lw							23_{16}
sw							$2B_{16}$
beq							4_{16}

Now you have Inputs (Opcode) and Outputs (Control signals) – can you design the control logic?

Input or output	Signal name	R-format	lw	sw	beq
Inputs	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
Outputs	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

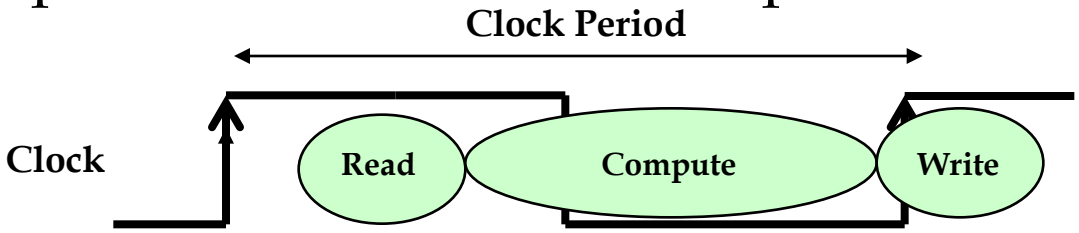
Combinational Implementation

Control logic



Big Picture: Instruction Execution

- Instruction execution is equivalent to
  - Read contents of one or more storage elements (register/memory)
  - Perform computation through some combinational logic
  - Write results to one or more storage elements (register/memory)
- All these performed within a clock period



Don't want to read a storage element when it is being written

What's Wrong With Single Cycle?

- Calculate cycle time assuming negligible delays except: memory (2ns), ALU/adders (2ns), register file access (1ns)

Inst	Inst Mem	Reg read	ALU	Data Mem	Reg write	Total
ALU	2	1	2		1	6
lw	2	1	2	2	1	8
sw	2	1	2	2		7
beq	2	1	2			5

- Long cycle time
- All instructions take as much time as the slowest one (i.e., load)

Solution 1: Multicycle Approach

- Break up the instructions into execution steps
  - Instruction fetch
  - Instruction decode and register read
  - ALU operation
  - Memory read/write
  - Register write
- Each execution step takes one clock cycle
  - Cycle time is much shorter, i.e., clock frequency is much higher
- Instructions take variable number of clock cycles to complete execution
- Not covered in class

Solution 2: Pipelining

- Break up the instructions into execution steps one per clock cycle
- Allow different instructions to be in different execution steps simultaneously
- Covered in next lecture

SUMMARY

- A very simple implementation of MIPS datapath and control for a subset of its instructions
- Concepts:
  - An instruction executes in a single clock cycle
  - Read storage elements, compute, write to storage elements
  - Datapath is shared among different instructions types using MUXs and control signals
  - Control signals are generated from the machine language encoding of instructions

READING ASSIGNMENT

- The Processor: Datapath and Control
  - 3rd edition: Chapter 5 Section 5.4
  - 4th edition: Chapter 4 Section 4.4