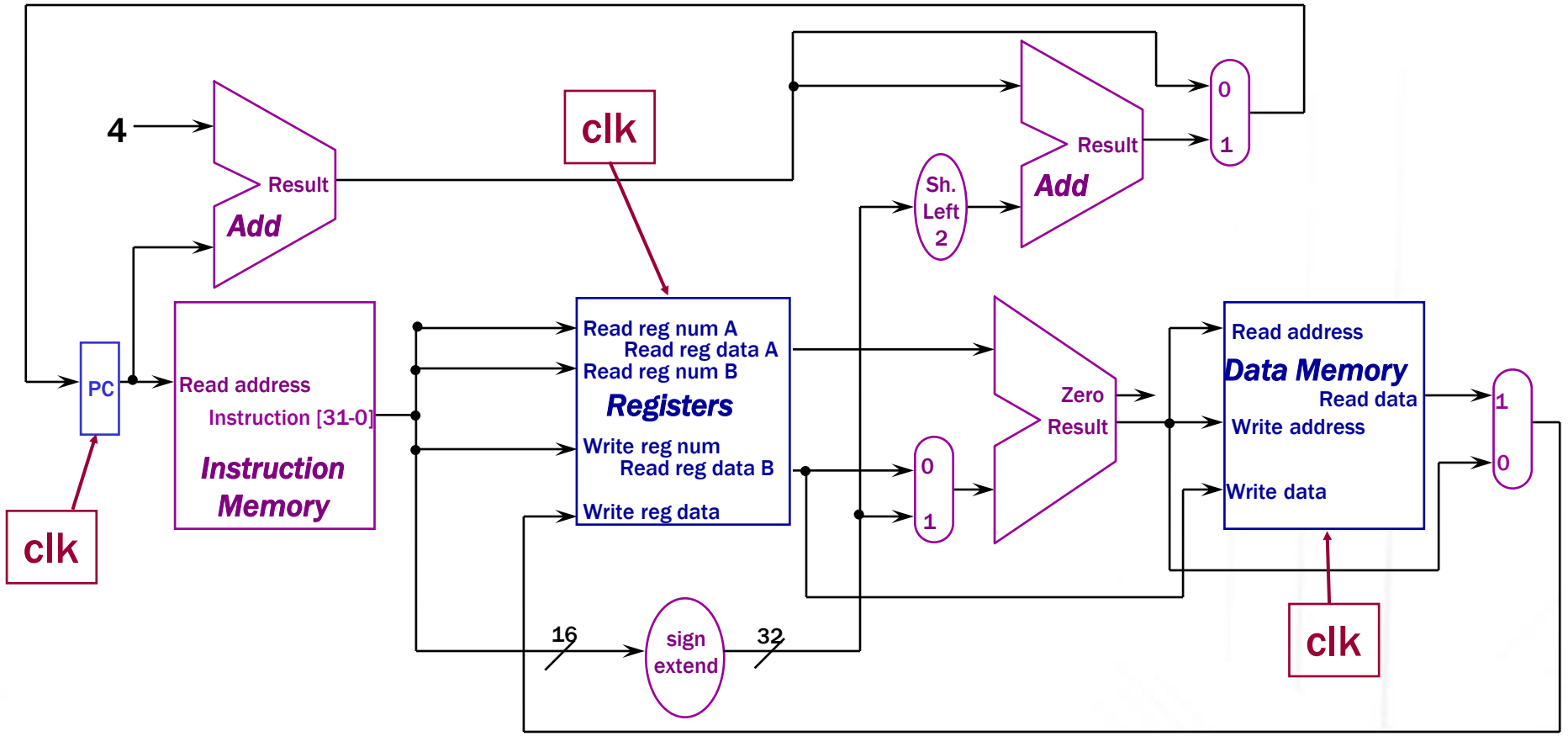
Computer Organization & Architecture

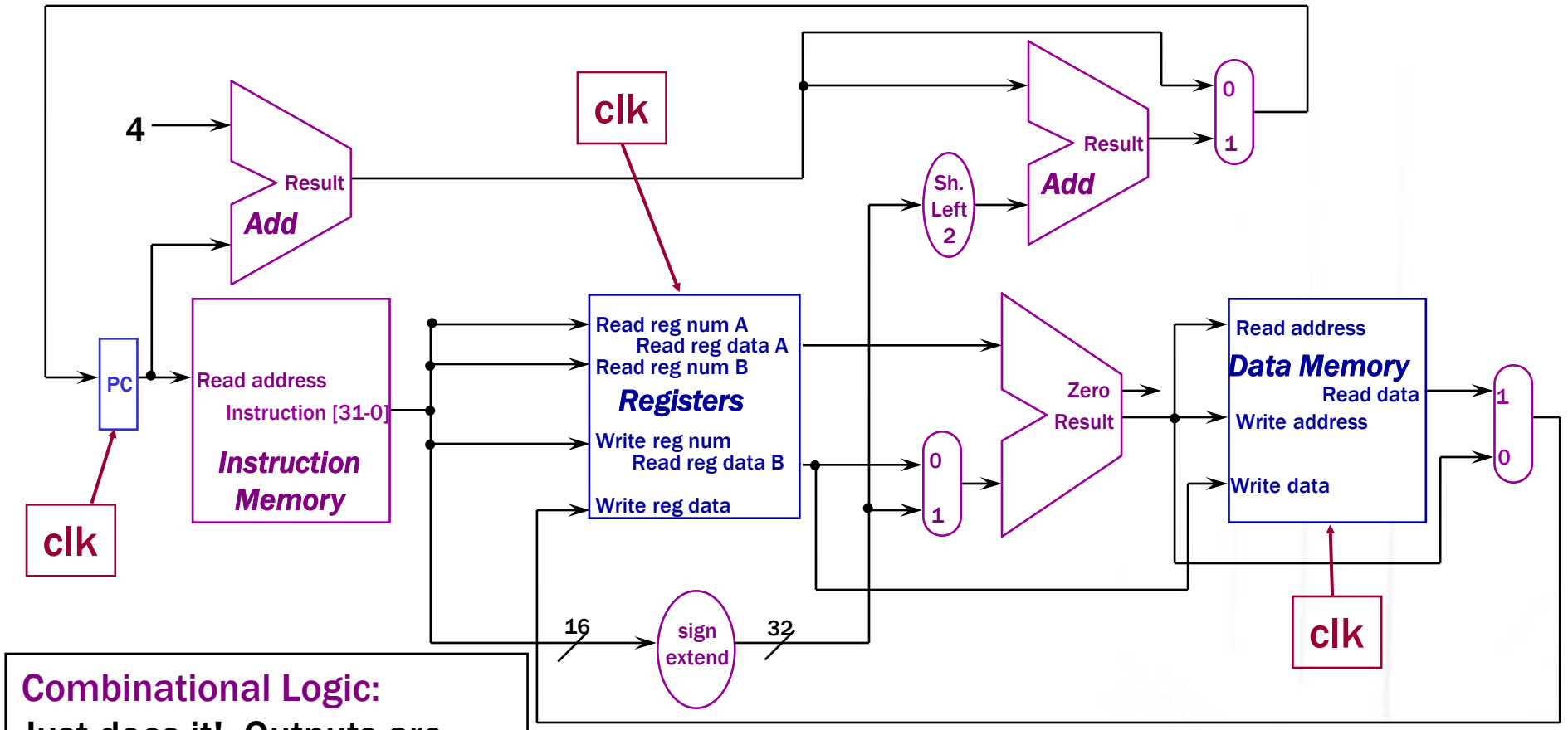
Agenda…

- Understanding ALU
- Building Control Unit

Single Cycle Processor Datapath



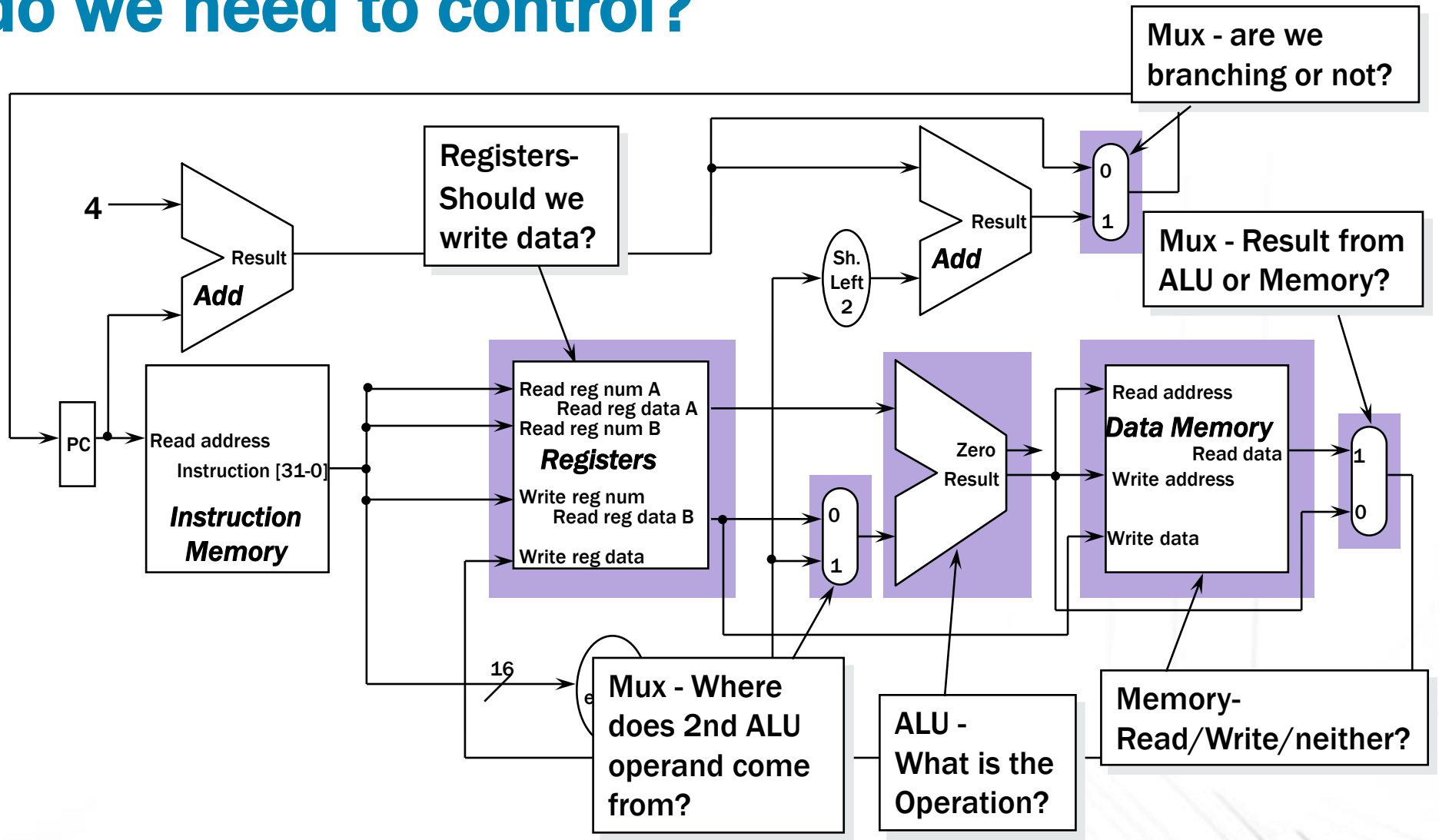
When does everything happen?



Combinational Logic:
Just does it! Outputs are always just a function of its inputs (with some delay)

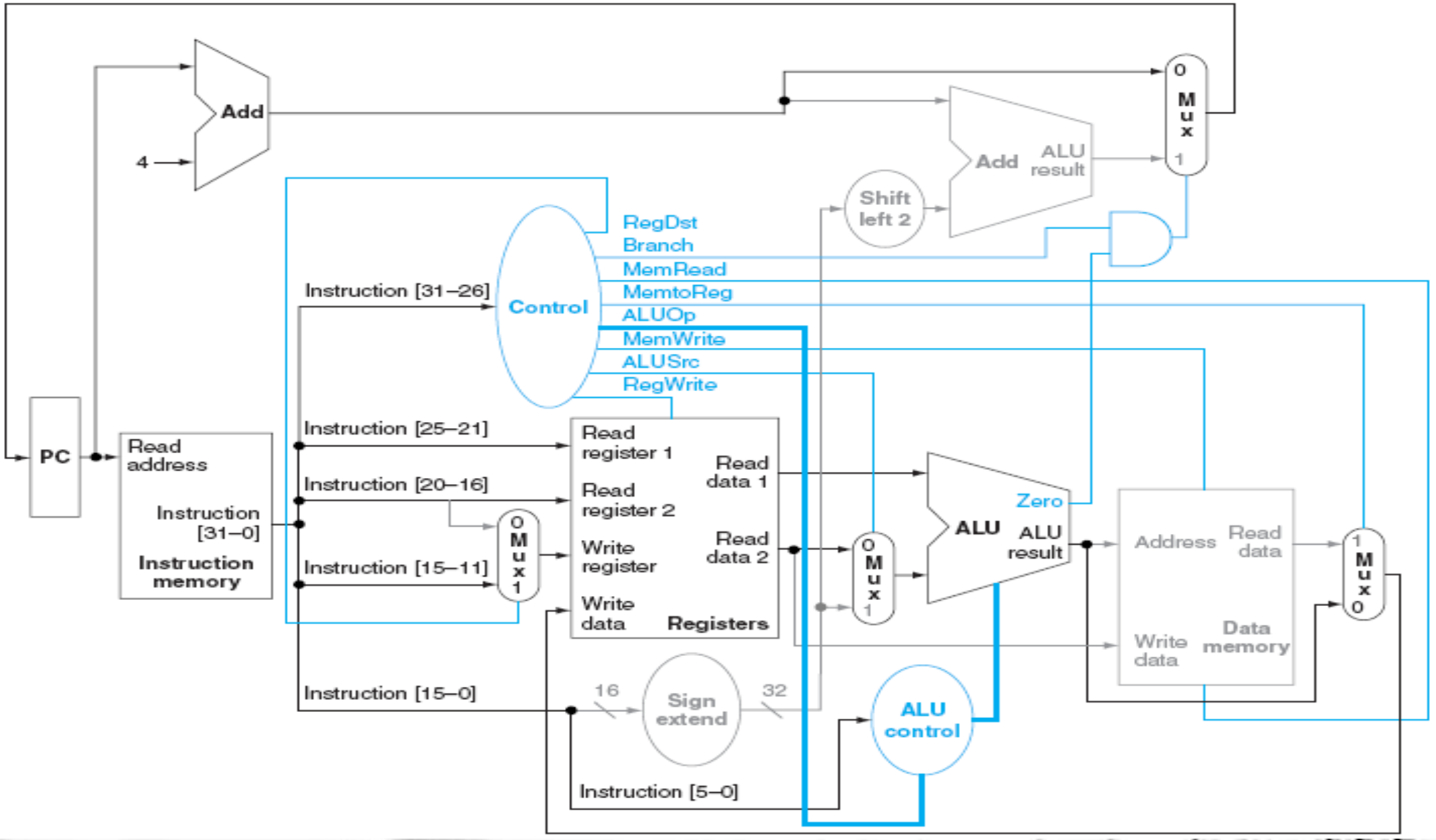
Registers: Written at the end of the clock cycle. (Rising edge triggered).

What do we need to control?



Almost all of the information we need is in the instruction!

Datapath with Control Unit



The ALU control

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

- R-type instructions perform AND, OR, add, subtract, slt etc..
- Add operation is also performed by memory reference instructions (lw, sw)

8

How do we generate ALU control input?

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

- ALU control input is generated using two different signals
  - ALUOp
  - Instruction function field

9

Setting the ALU controls

- The instruction Opcode and Function give us the info we need
  - For R-type instructions, Opcode is zero, function code determines ALU controls
    - For I-type instructions, Opcode determines ALU controls

New control signal: ALUOp is 00 for memory, 01 for Branch, and 10 for R-type

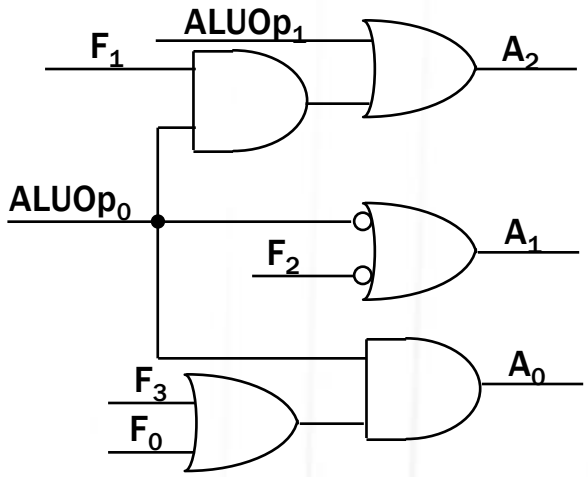
Instruction	Opcode	ALUOp	Funct. Code	ALU action	ALU control	
					sub	op
add	R-type	10	100000	add	0	10
sub	R-type	10	100010	subtract	1	10
and	R-type	10	100100	and	0	00
or	R-type	10	100101	or	0	01
SLT	R-type	10	101010	SLT	1	11
load word	LW	00	xxxxxx	add	0	10
store word	SW	00	xxxxxx	add	0	10
branch equal	BEQ	01	xxxxxx	subtract	1	10

Controlling the ALU

ALUOp is determined by Opcode - separate logic will generate ALUOp

For ALUOp = 00 or 01, function code is unused

ALUOp	F_5	F_4	F_3	F_2	F_1	F_0	Function	ALU Ctrl
00	x	x	x	x	x	x	Add	0 10
x1	x	x	x	x	x	x	Sub	1 10
1x	x	x	0	0	0	0	Add	0 10
1x	x	x	0	0	1	0	Sub	1 10
1x	x	x	0	1	0	0	And	0 00
1x	x	x	0	1	0	1	Or	0 01
1x	x	x	1	0	1	0	SLT	1 11



Since ALUOp can only be 00, 01, or 10, we don't care what $ALUOp_0$ is when $ALUOP_1$ is 1

A 6-input truth table - use standard minimization techniques

Designing main control unit

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

a. R-type instruction

Field	35 or 43	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

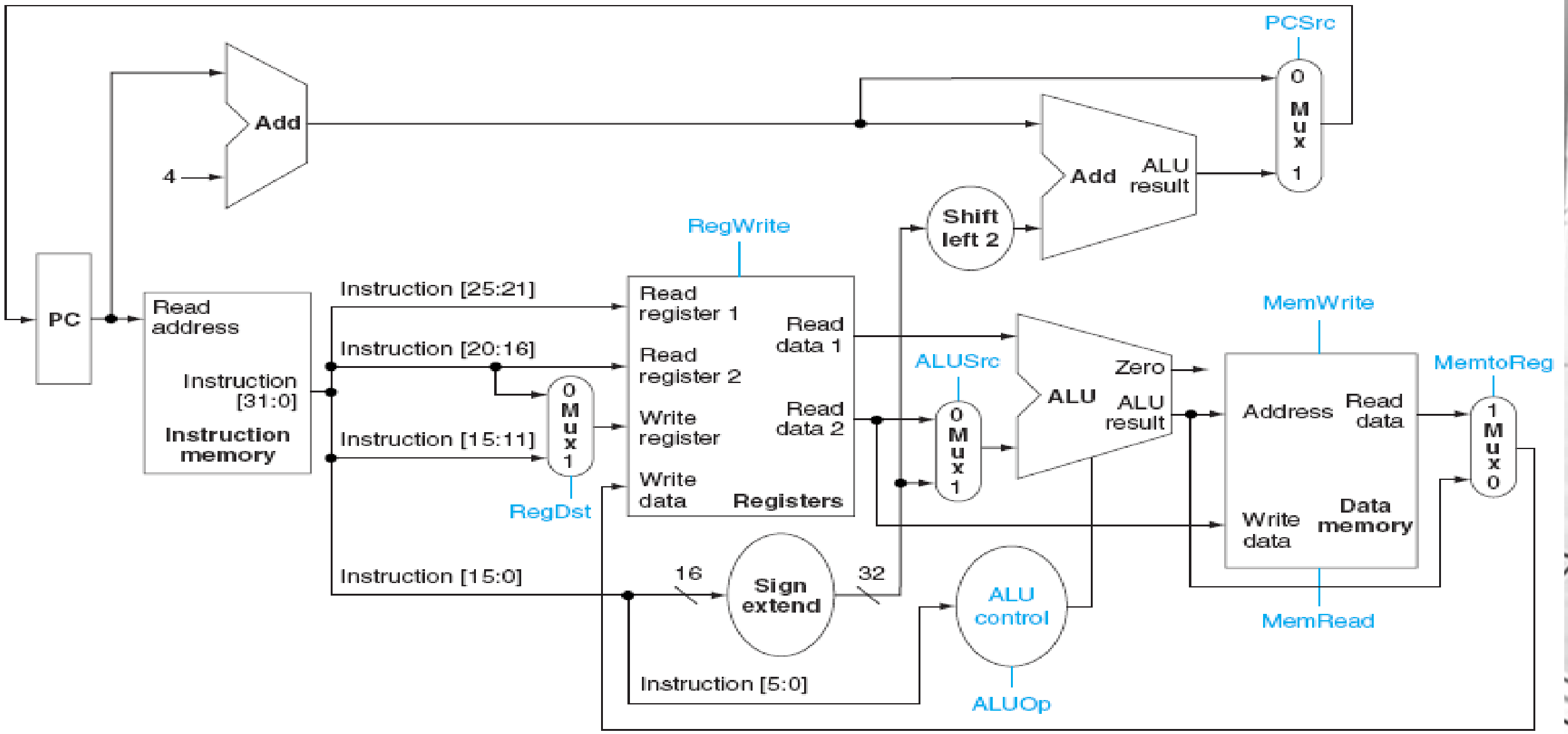
b. Load or store instruction

Field	4	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

c. Branch instruction

- Op code always contained in 31:26
- Registers to be read rs, rt specified at 25:21, 20:16. True for all instructions except load
- Base register for load and store is rs
- 16 bit offset is in position 15:0
- Destination register is either rd for R-type (15:11) or rt (20:16) for load instruction

12


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

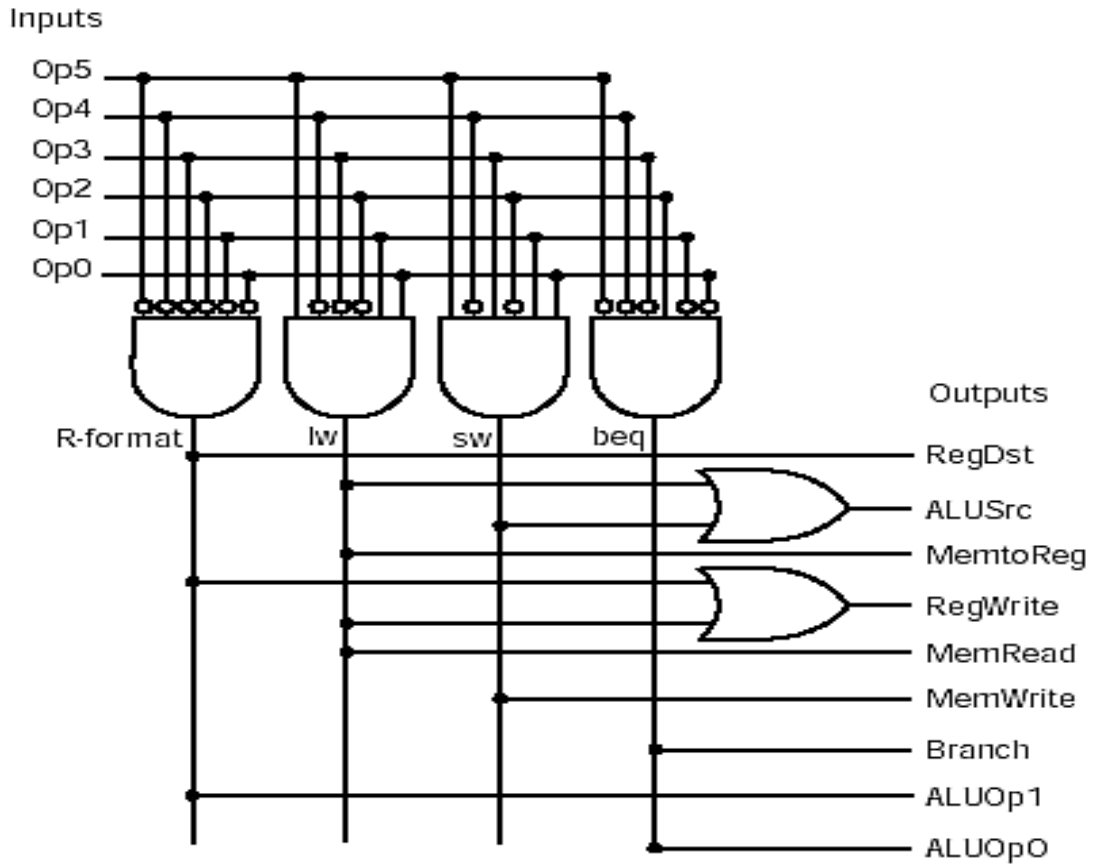
Inside the control oval(Main Control Unit)

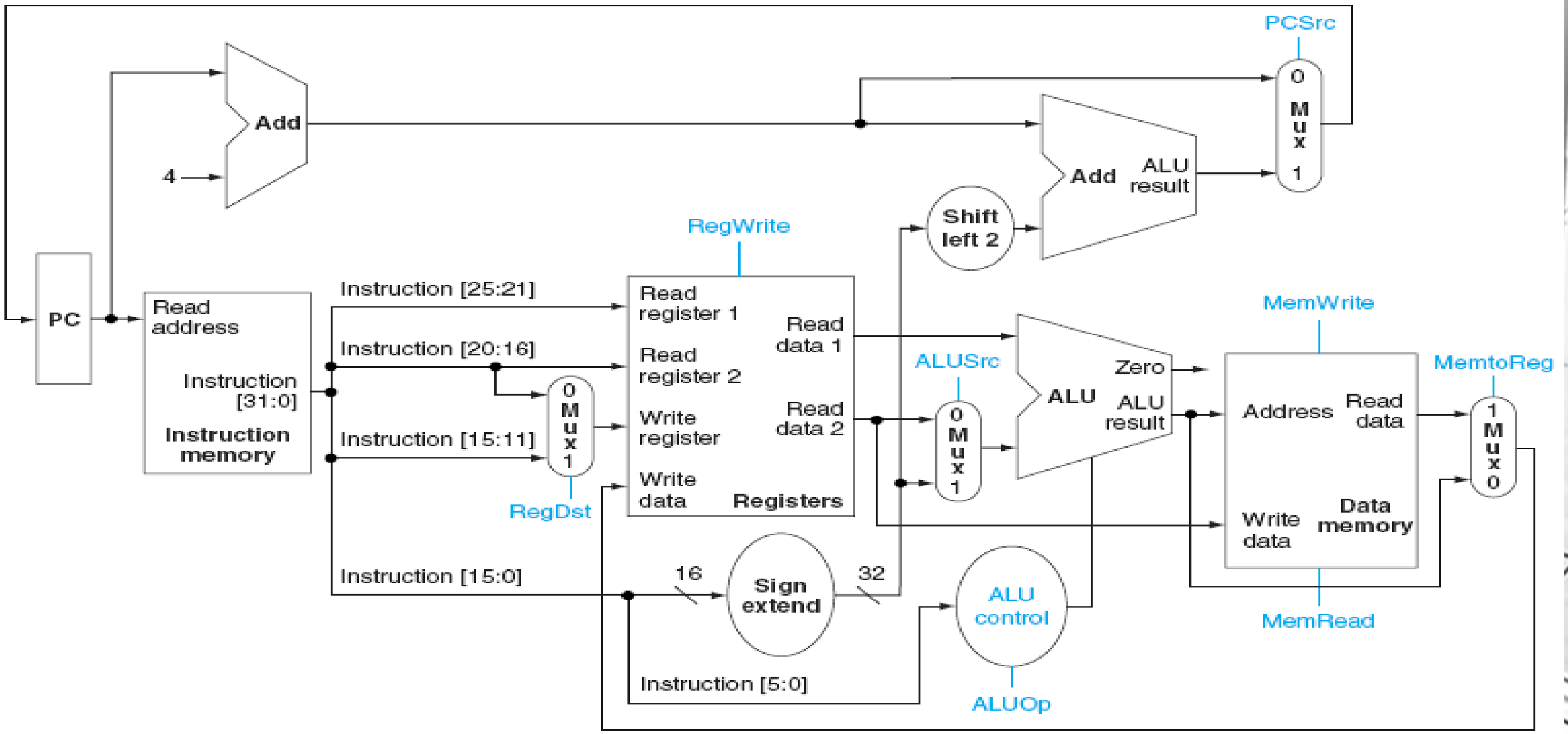
Instruction	Opcode	Reg Write	ALU Src (0:Reg 1:Imm)	Mem To Reg (1:Mem 0:ALU)	Reg Dest (0:Rt 1:Rd)	Mem Read	Mem Write	PCSrc (1:Branch)	ALUOp (00:Mem 01:Branch 10:R-type)
R-format	000000	1	0	0	1	0	0	0	10
LW	100011	1	1	1	0	1	0	0	00
SW	101011	0	1	x	x	0	1	0	00
BEQ	000100	0	0	x	x	0	0	1	01

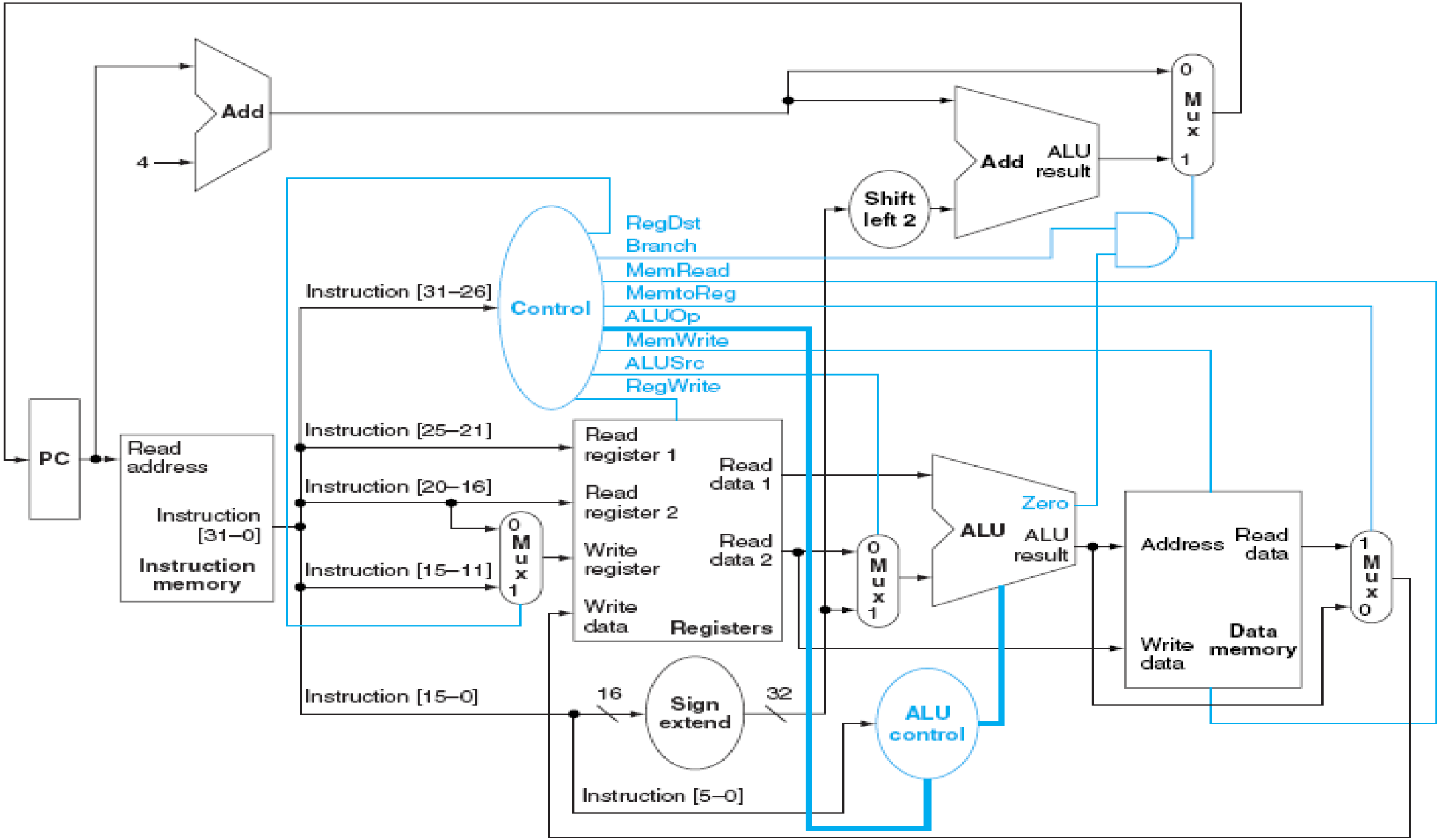
- This control logic can be decoded in several ways:
  - Random logic, PLA, PAL
- Just build hardware that looks for the 4 opcodes
  - For each opcode, assert the appropriate signals

Note: BEQ must also check the *zero* output of the ALU…

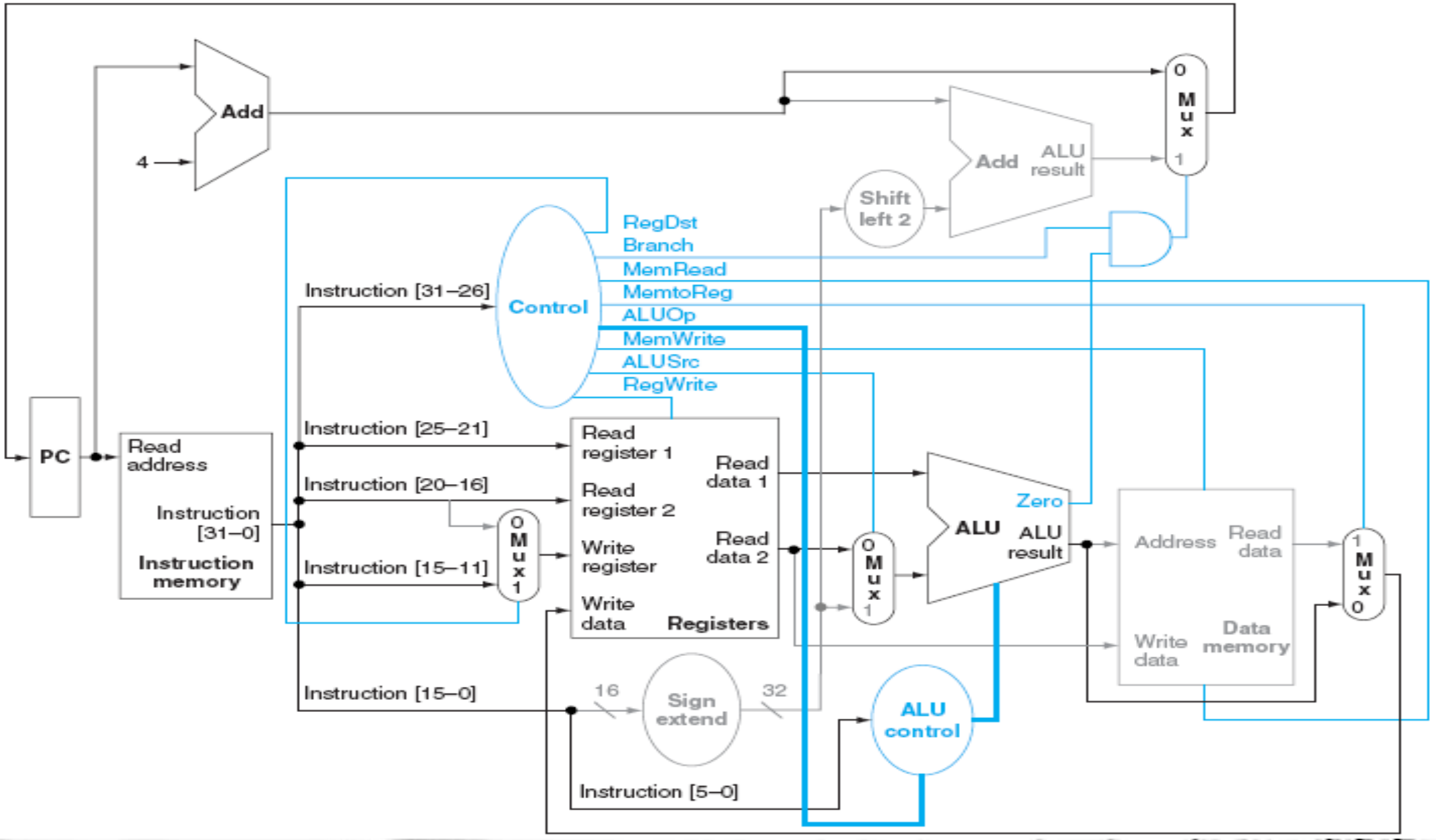
Control Unit Implementation




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


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

Datapath with Control Unit



Implementing jumps

Field	000010	address
Bit positions	31:26	25:0