EE116C/CS151B
Fall 2017

Instructor: Professor Lei He
TA: Yuan Liang

Homework Discussion

Problem 1

Assume for a given processor the CPI of arithmetic instructions is 1, the CPI of load/store instructions is 10, and the CPI of branch instructions is 3. Assume a program has the following instruction breakdowns:

500 million arithmetic instructions, 300 million load/store instructions, 100 million branch instructions.

Suppose that new, more powerful arithmetic instructions are added to the instruction set. On average, through the use of these more powerful arithmetic instructions, we can reduce the number of arithmetic instructions needed to execute a program by 25%, and the cost of increasing the clock cycle time by only 10%. Is this a good design choice? Why?

Homework Discussion

Problem 1

As: CPU time = Sum (instruction counts * CPI) * seconds/cycles

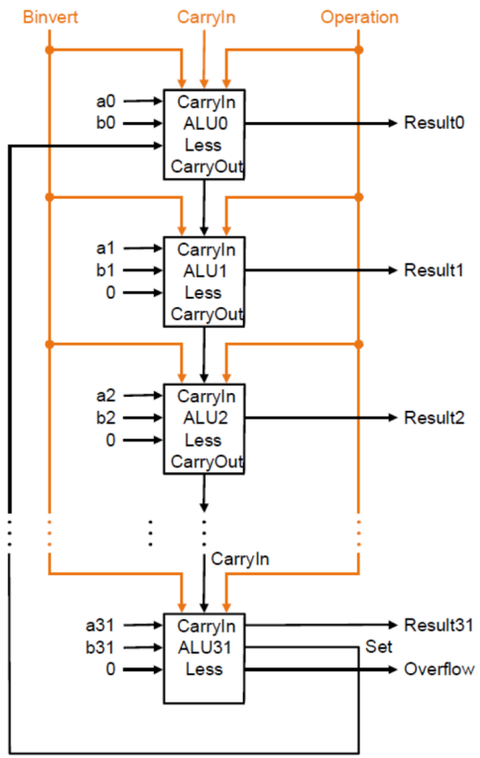
Thus:
Old CPU time = (500m*1+300m*10+100m*3)*CT = 3800m*CT;
New CPU time = (500m*0.75*1+300m*10+100m*3)*1.1CT = 4042.5m*CT;
Thus the new CPU time is larger than the old one. So it is not a good design.

Homework Discussion

Problem 2

The single cycle implementation of the MIPS processor uses for the mai n ALU the implementation shown in the figure below. Due to a circuit malfunction, the CarryIn signal to the least significant bit is always zero. The rest of the circuitry of the ALU and the rest of the processor operates normally.

Explain in full detail what will be the consequences of this fault when the processor executes programs — how will it change the behavior of the processor as observed by a user/ programmer who does not know a nd does not care how t he processor is implemented internally? Be sure to clearly identify each and every consequence of this fault.



Homework Discussion

Problem 2

When operating normally, the ALU does subtraction based on the equation:
A – B = A + (–B)
In those cases, the CarryIn is set to 1 as part of the computation of –B. Since the malfunction causes CarryIn to be 0, the result for subtraction will always be one less than it should be. Specifically, instead of computing A – B, the ALU will compute A – B – 1.

A user/programmer will observe incorrect results for all instructions that require the ALU to perform subtraction. Specifically:
A) The sub instruction will compute Rs – Rt – 1 instead of Rs – Rt.
B) The beq instruction will branch whenever Rs = = Rt + 1 instead of whenever Rs = = Rt.
C) The slt instruction will set Rd to 1 whenever Rs < Rt + 1 instead of whenever Rs < Rt.

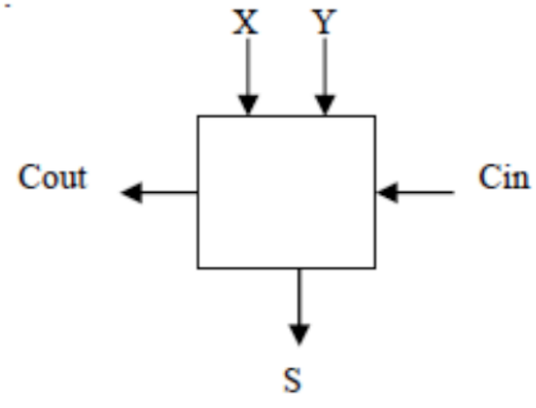
Homework Discussion

Problem 3

Draw a logic circuit of an adder of two 4 -bit numbers. Your design should be simple and modular.

Homework Discussion

A modular 1-bit adder:



Truth table for the 1-bit adder:

X	Y	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Karnaugh map for S:

S		YCin			
		00	01	11	10
X	0	0	1	0	1
	1	1	0	1	0

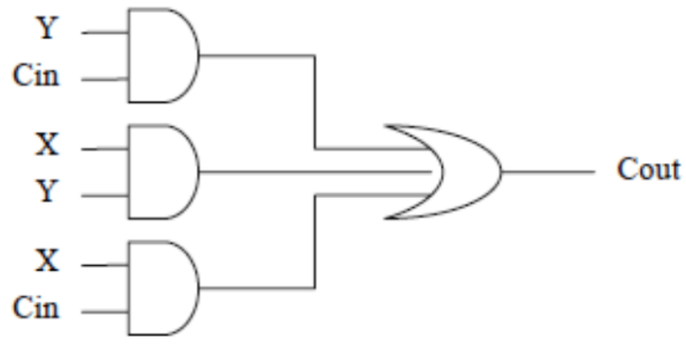
$$S = \overline{X}\overline{Y}Cin + \overline{X}Y\overline{Cin} + X\overline{Y}\overline{Cin} + XYCin$$

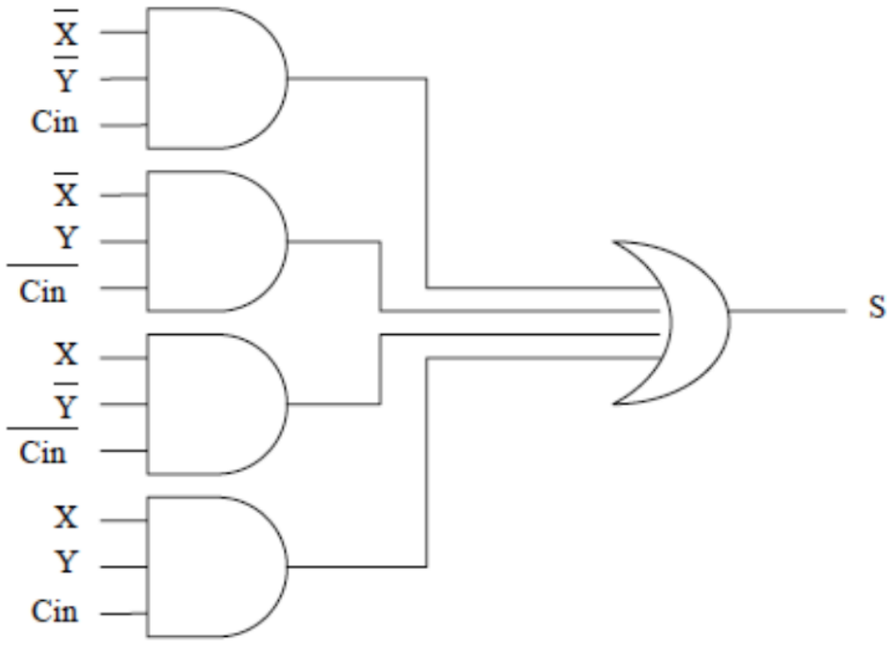
Homework Discussion

Karnaugh map for Cout:

Cout		YCin			
		00	01	11	10
X	0	0	0	1	0
	1	0	1	1	1

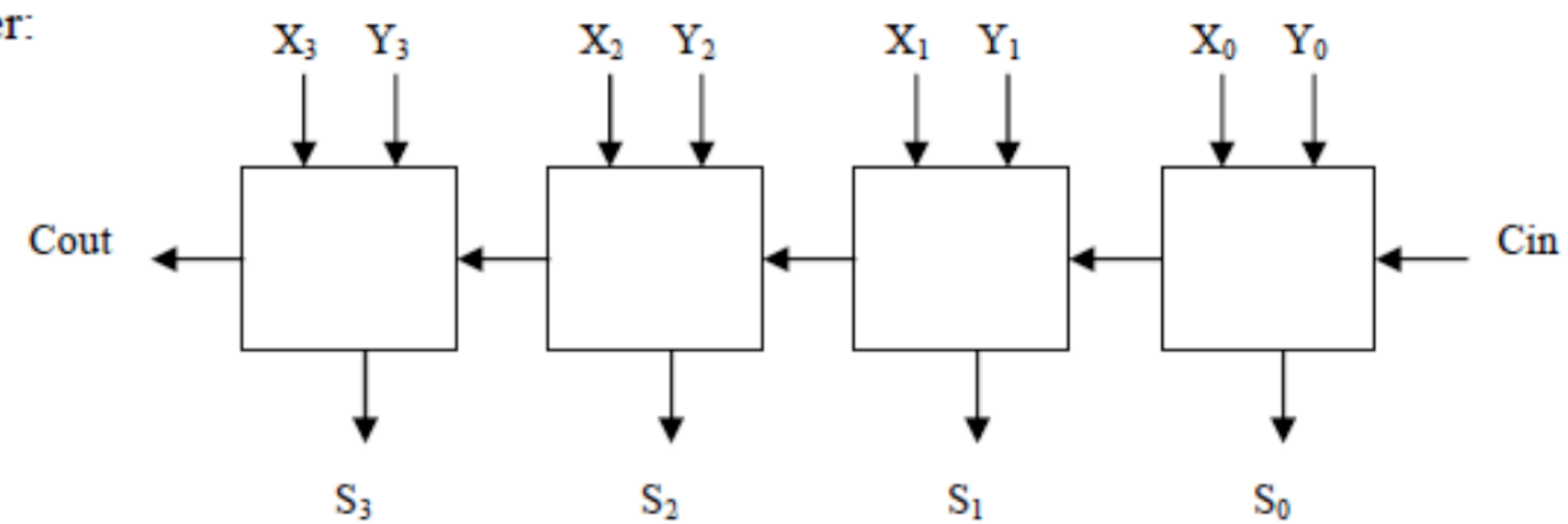
Cout = YCin + XY + XCin





Homework Discussion

4-bit adder:



Homework Discussion

problem 4

Using the circuit from problem 3 and D-flip-flops as a building blocks, show the design of a 4-bit counter that counts down.

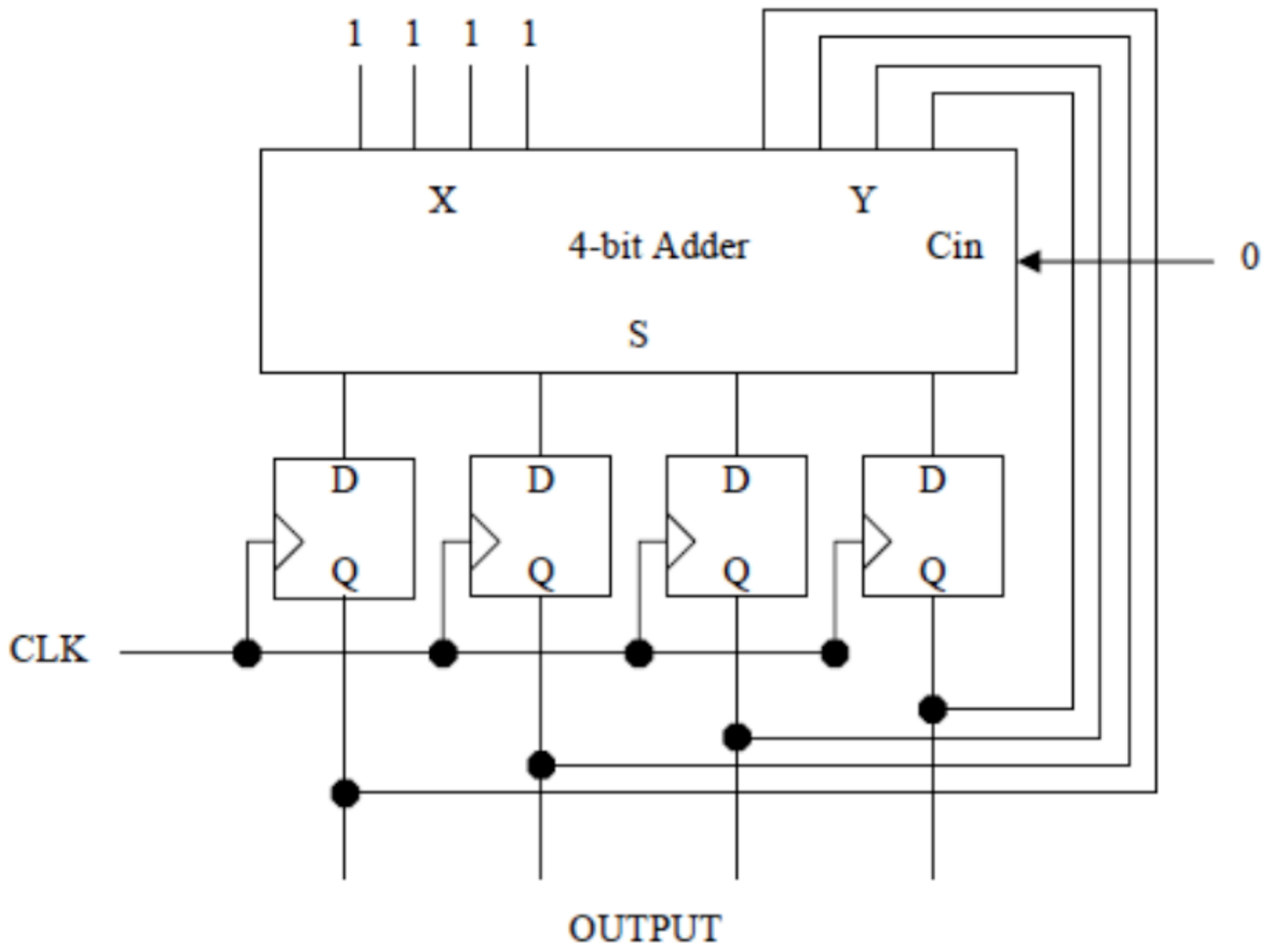
The only external input to this circuit is the clock. The output four bits continuously follow the sequence: . . . , 11 , 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, 15, 14, 13, 12, 11, . . .

Your goal is to minimize the circuitry needed in addition to the module from problem 3.

D-flip-flop is as shown below:

Homework Discussion

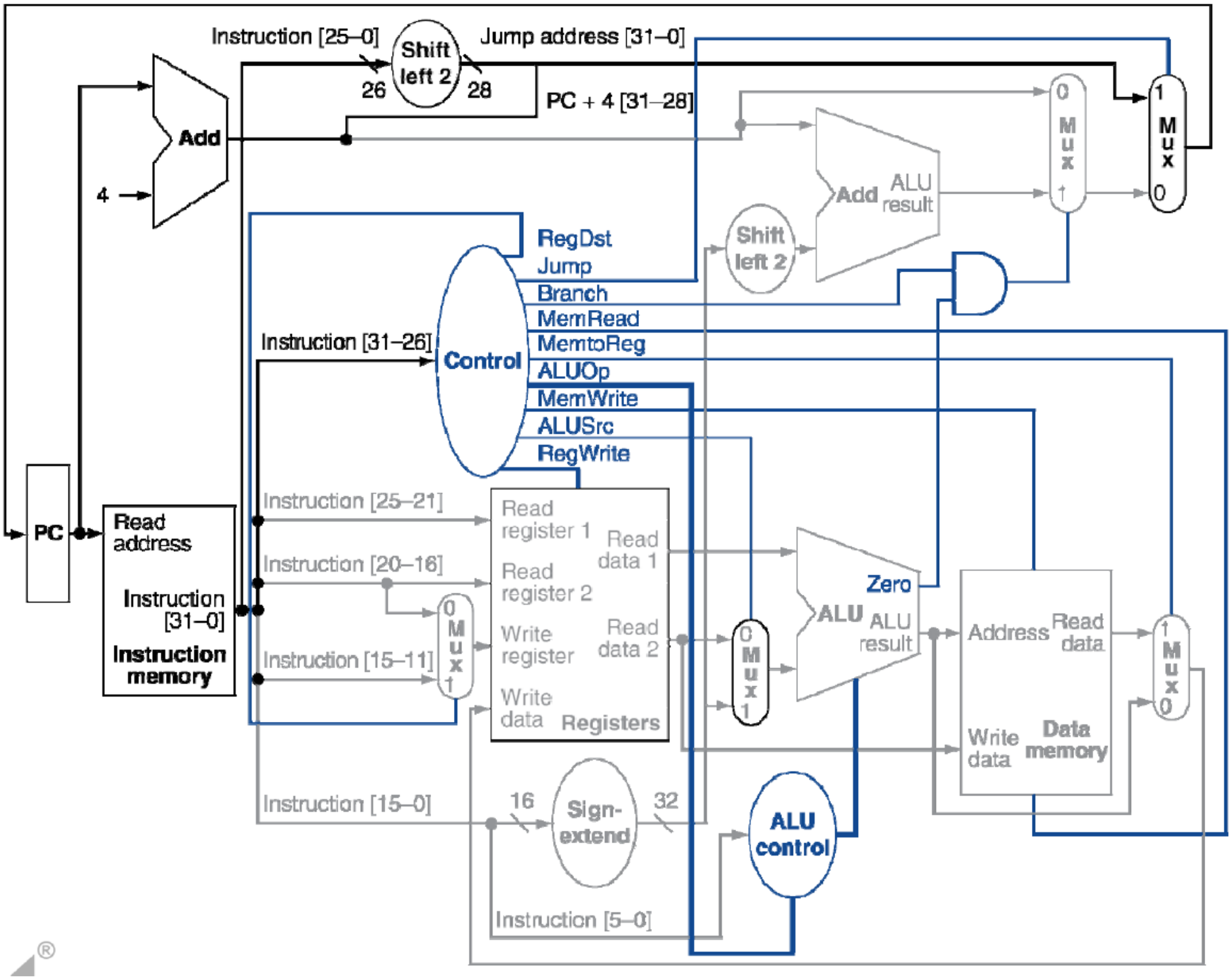
problem 4



Review

- **Single cycle data/control path**
  - **Look into components**
    - Input/output/control ports
      - why we need them
      - how many bits
- **Look into simple functions (*more covered in following lectures*)**
  - R type (add, sub, and, or, slt)
  - I type (lw, rw, beq)
  - J type (jump)

Review


Instruction [25–0]
Shift left 2
Jump address [31–0]
26
28
PC + 4 [31–28]
Add
4
0
M
u
x
1
0
1
M
u
x
0
Add
ALU result
Shift left 2
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [31–26]
Control
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 control
Zero
ALU
ALU result
Address
Read data
Write data
Data memory

Review

- **Overall workflow**

PC → instruction memory, fetch instruction

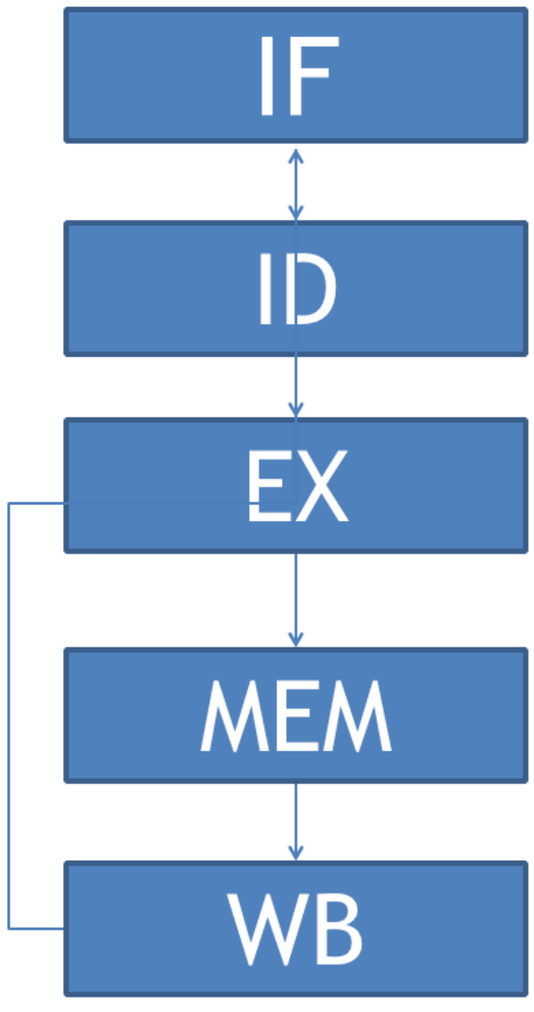
Register numbers → register file, read registers

Depending on instruction class

- Use ALU to calculate
  - Arithmetic result
  - Memory address for load/store
  - Branch target address
- Access data memory for load/store
- PC ← target address or PC + 4

Review

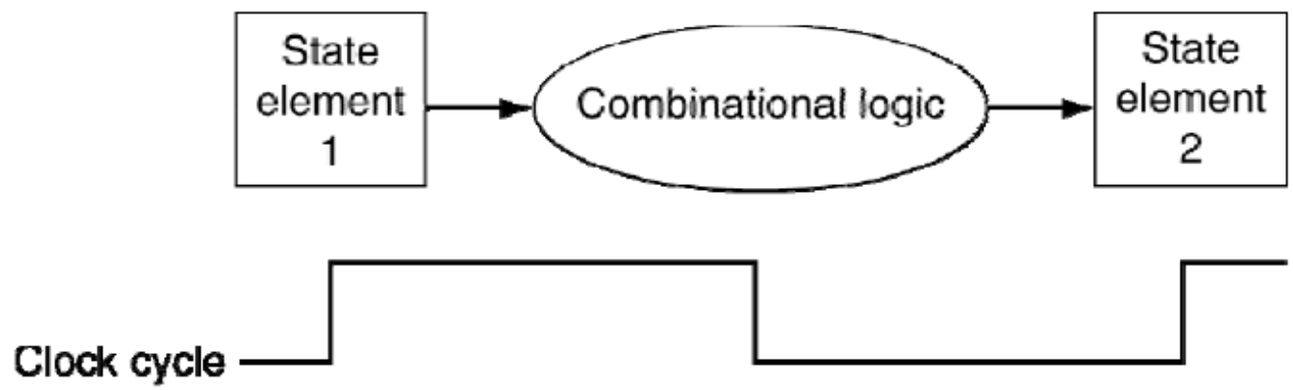
- **Overall workflow**



1. Read IM[PC]
2. Instruction Decode, PC =PC + 4, Register read
3. ALU operation, Branch address computation
4. LOAD/STORE in Data memory
5. Register Write

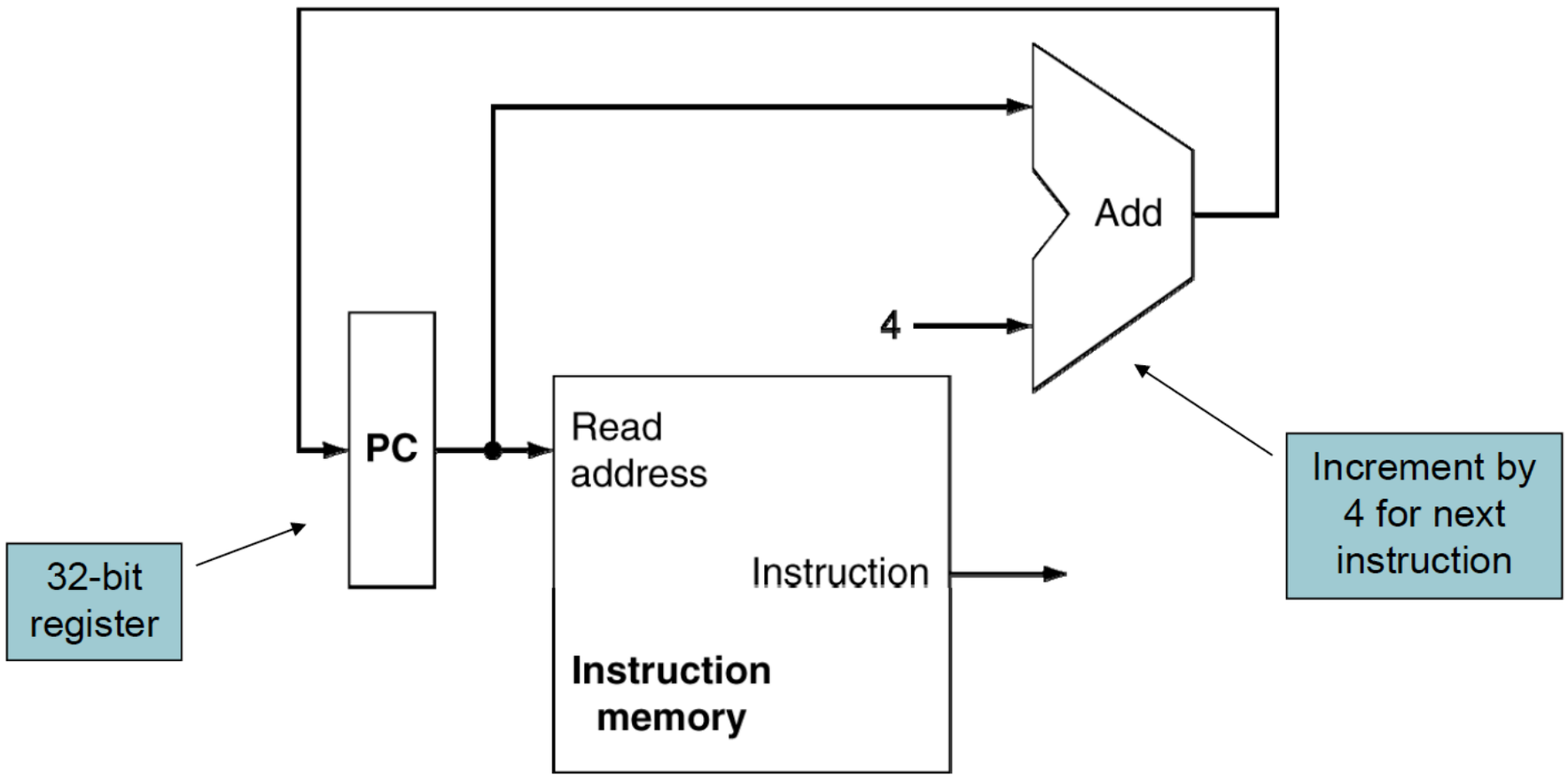
Review

- **How signals flow inside a processor**
  - **Information encoded in binary**
    - Low voltage = 0, High voltage = 1
    - One wire per bit
    - Multi-bit data encoded on multi-wire buses
- **Data transforms between logic components with clock cycles**
  - **Between clock edges**
    - Input from state elements, output to state element
    - Longest delay determines clock period



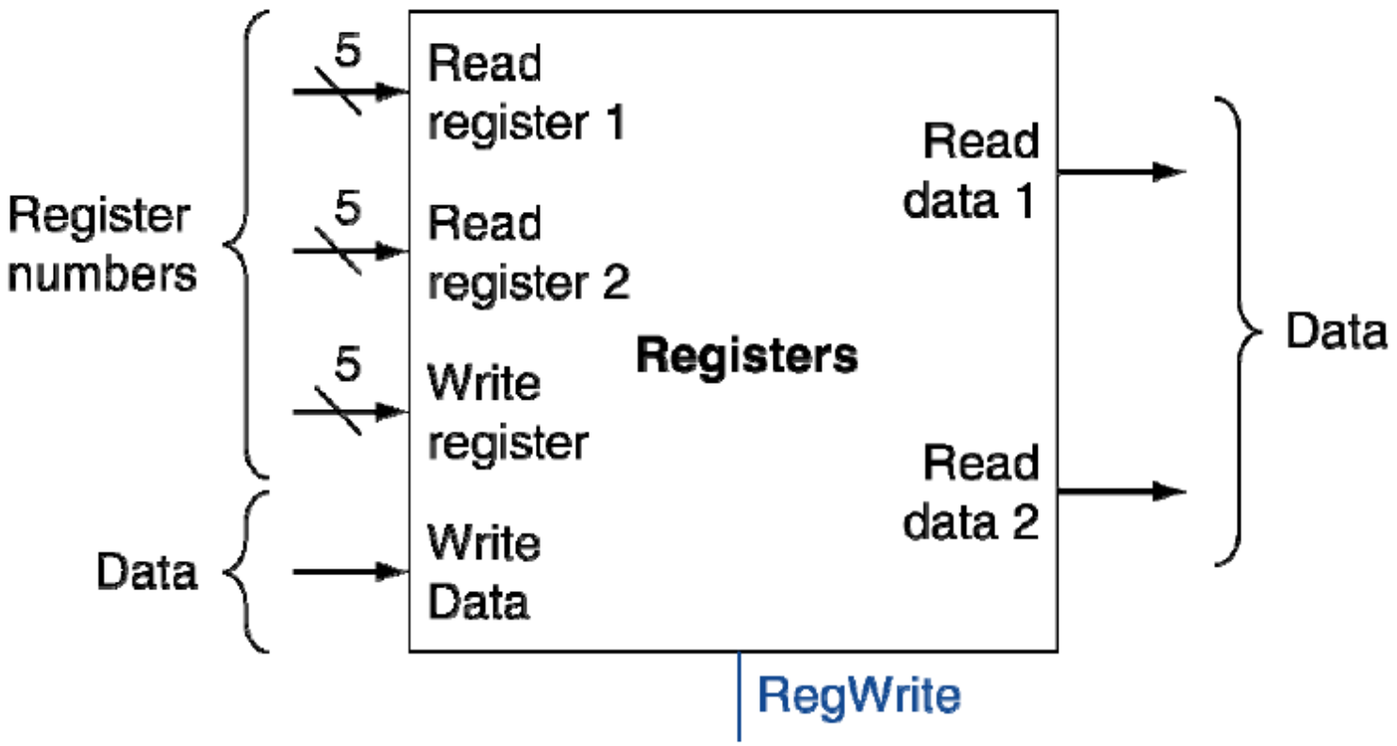
Review

- **Instruction memory**
- Updating the PC for next instruction
  - Sequential Code: PC <- PC + 4
  - Branch/Jump: PC <- “something else”



Review

- **Register**

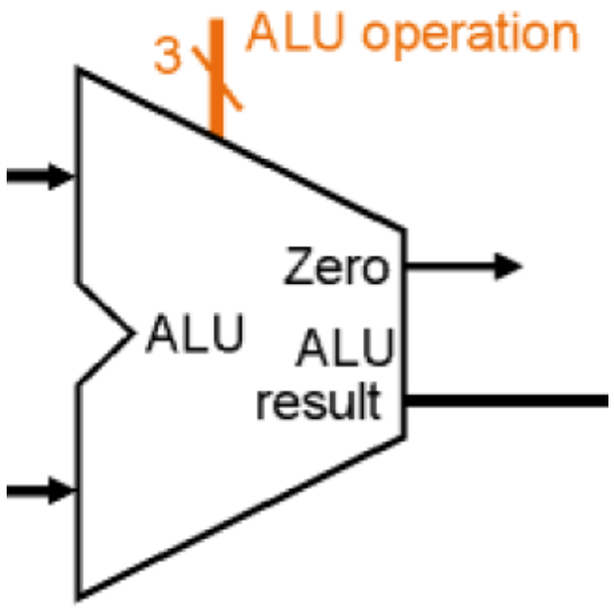


a. Registers

RegWrite selects the register to be written via Write Data when Write
Enable is 1

Review

- **Register**

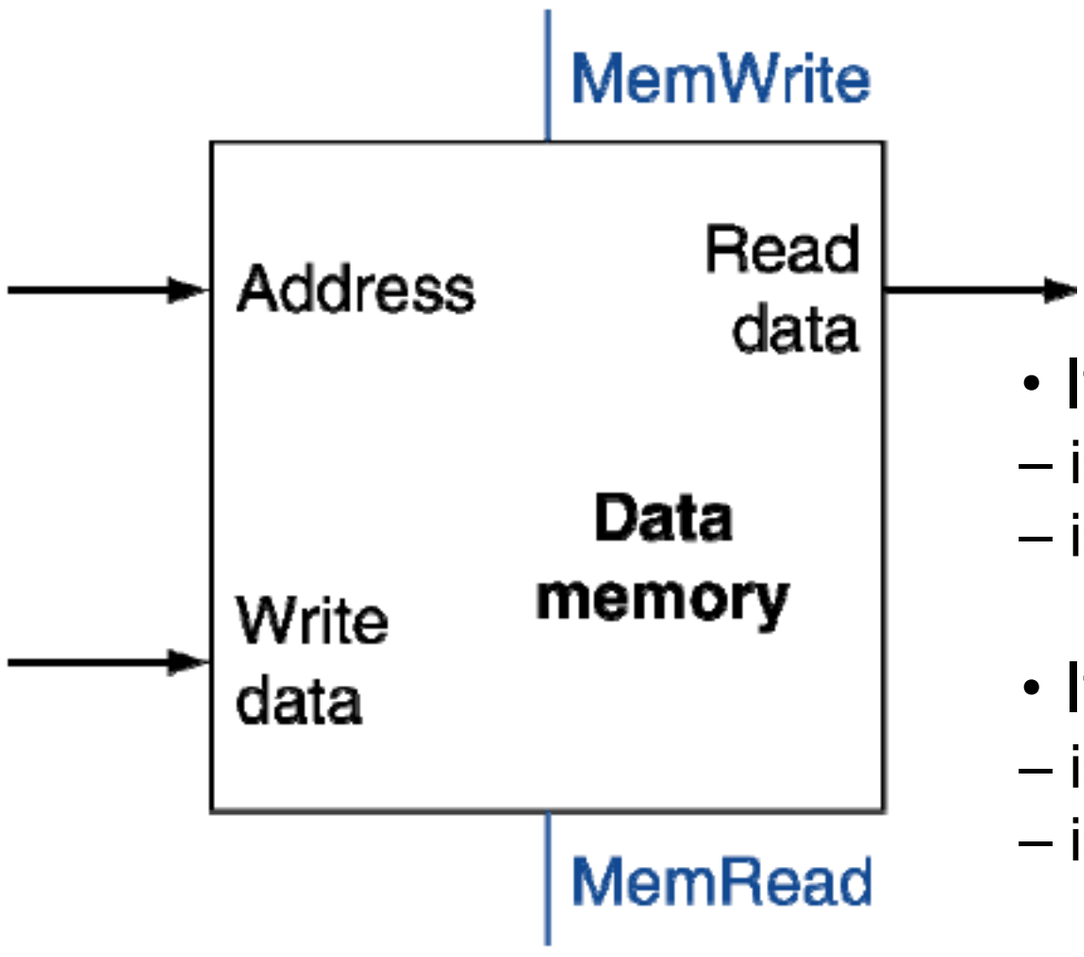


5 function ALU:

ALU Control Input	Function
000	AND
001	OR
010	add
110	subtract
111	set on less than

Review

- **Memory**

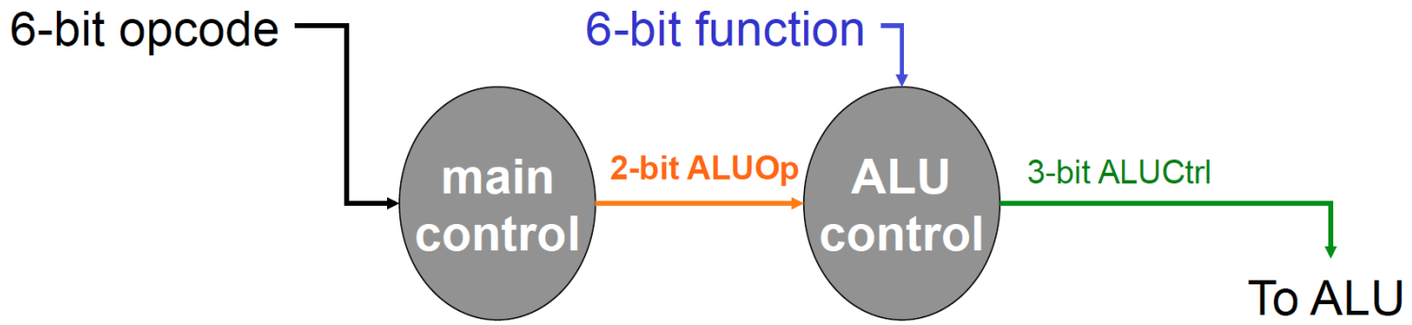


- If Write Enable
  - is 0, the memory location is put on Data Out bus
  - is 1, the memory location is overwritten by Data In

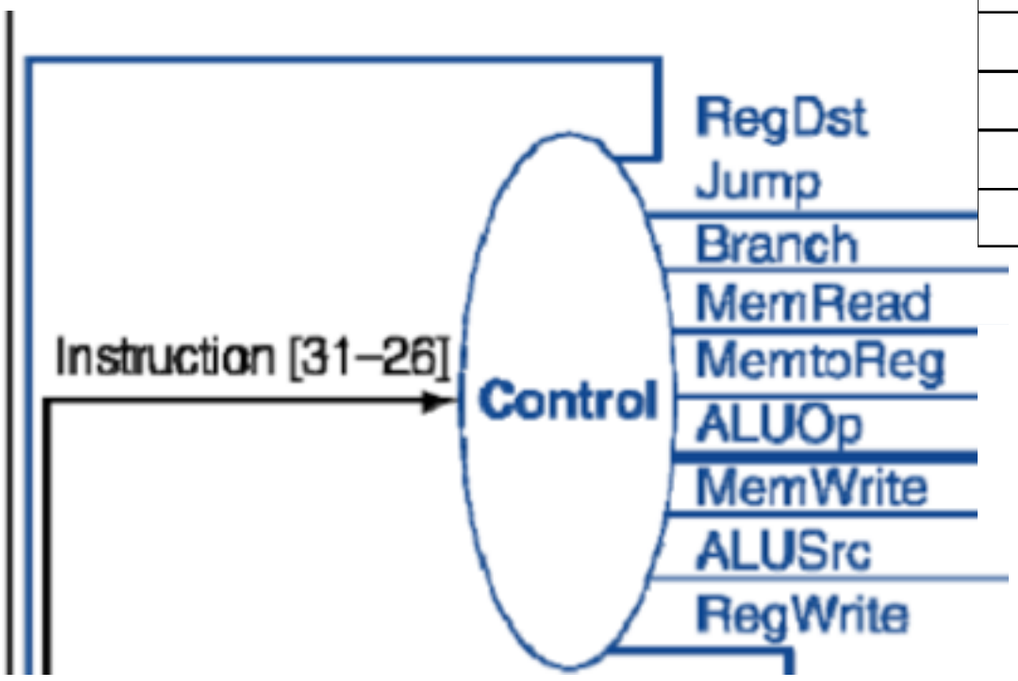
- If Read Enable
  - is 0.
  - is 1, the memory in address is put on out bus

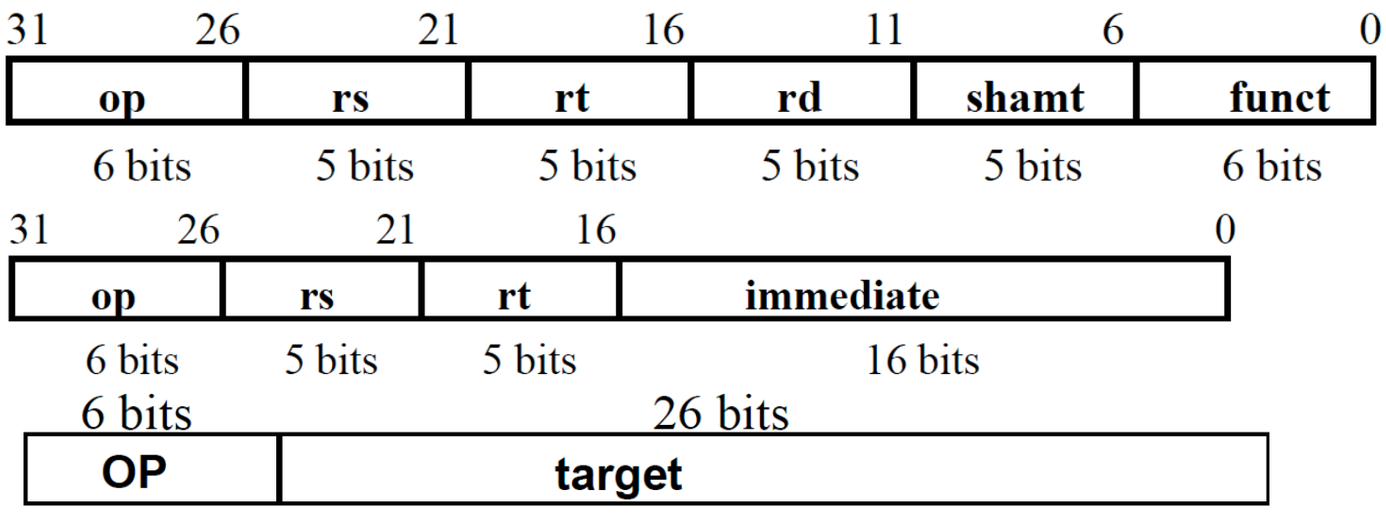
Review

- **Control unit**



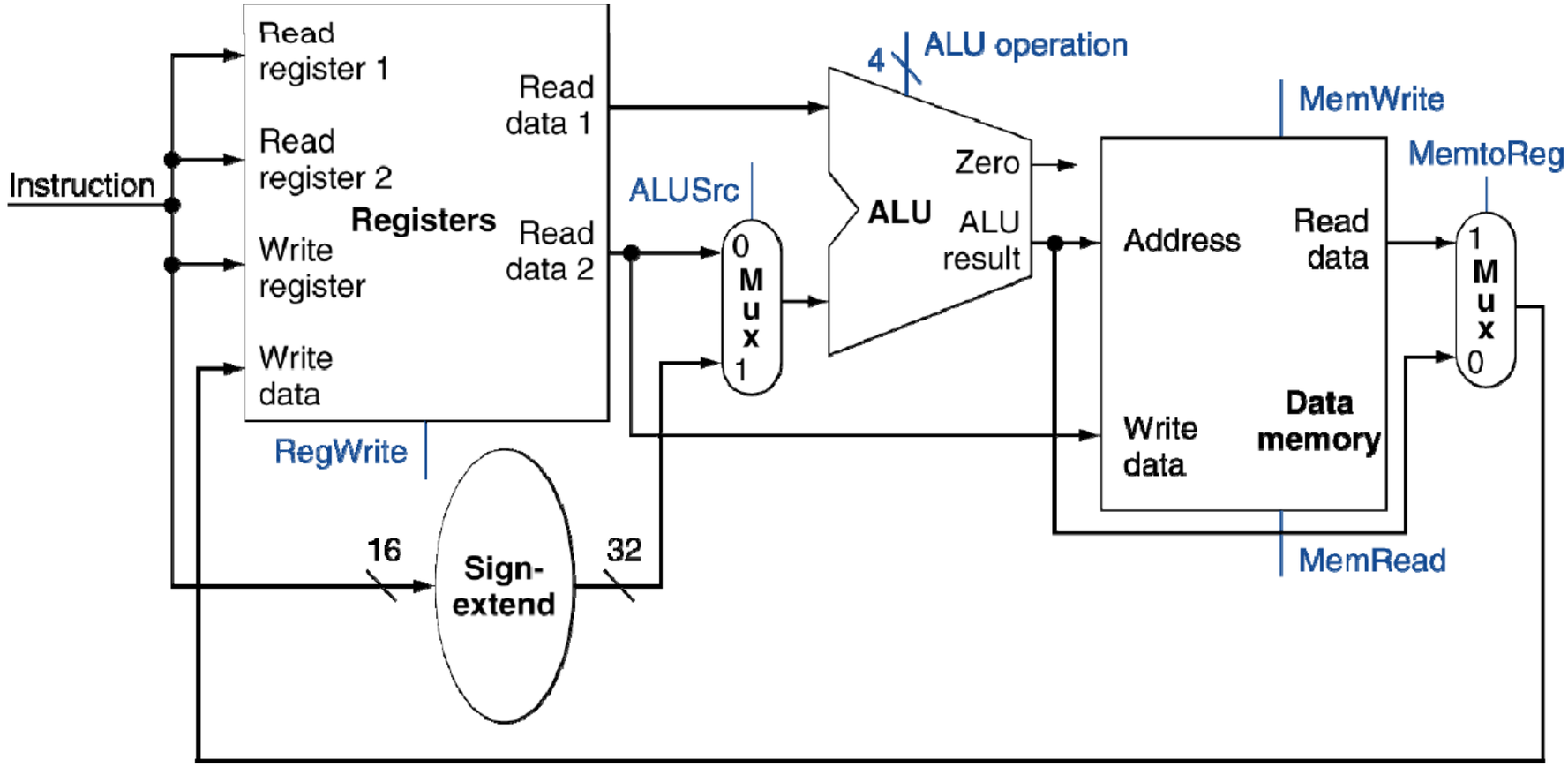
opcode	ALUOp	instruction	function	ALU Action	ALUCtrl
lw	00	load word	XXXXXX	add	010
sw	00	store word	XXXXXX	add	010
beq	01	branch equal	XXXXXX	subtract	110
R-type	10	add	100000	add	010
R-type	10	subtract	100010	subtract	110
R-type	10	AND	100100	AND	000
R-type	10	OR	100101	OR	001
R-type	10	SLT	101010	SLT	111





Review

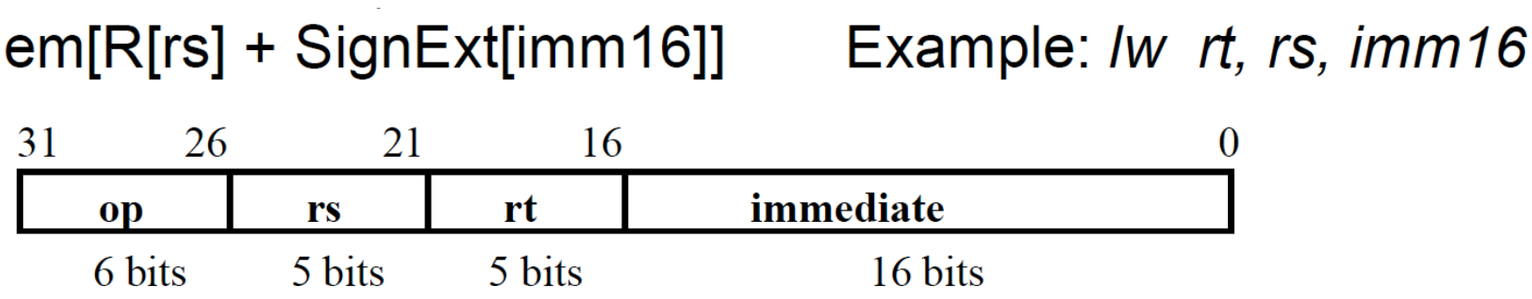
- **Instructions**
  - add, sub, and, or, slt (R-type)

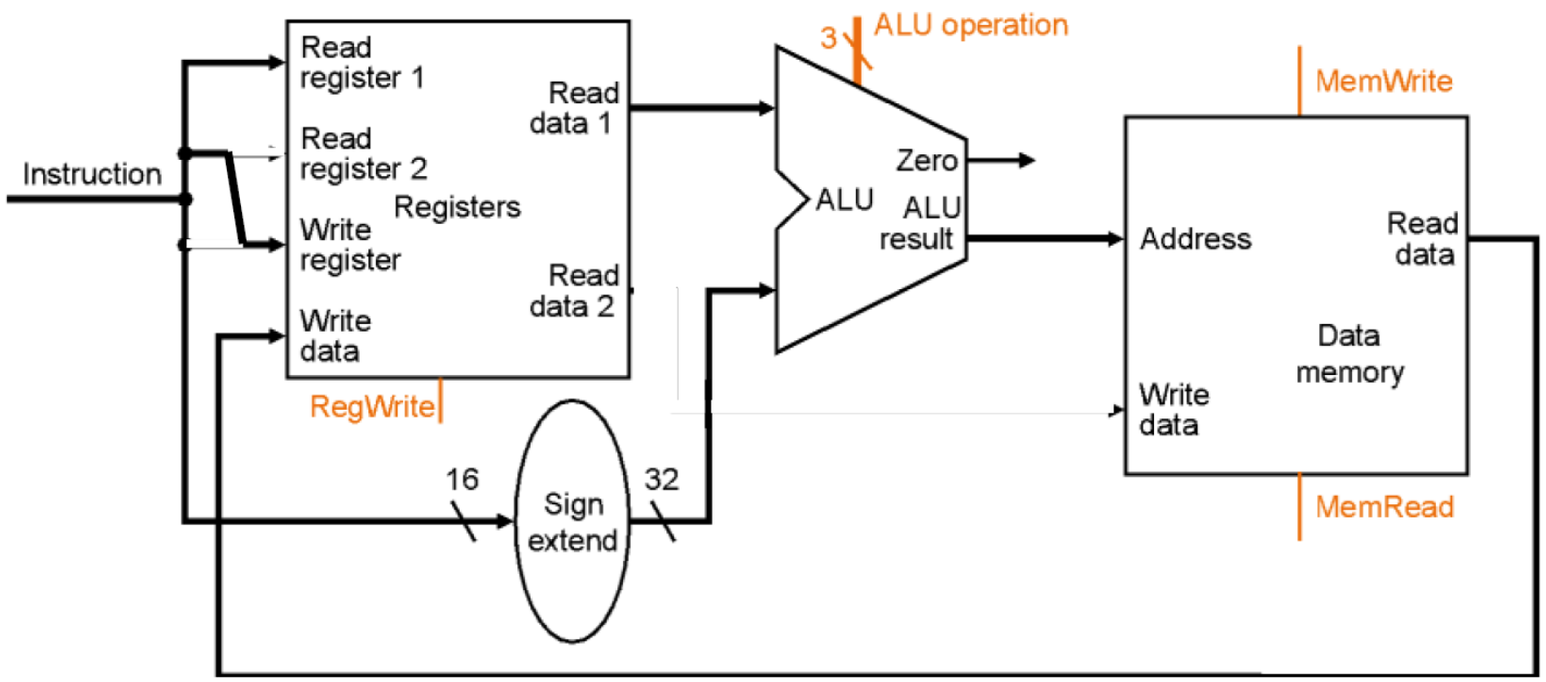


Review

- **Instructions**
  - rw, lw (I-type)

R[rt] <- Mem[R[rs] + SignExt[imm16]] Example: *lw rt, rs, imm16*

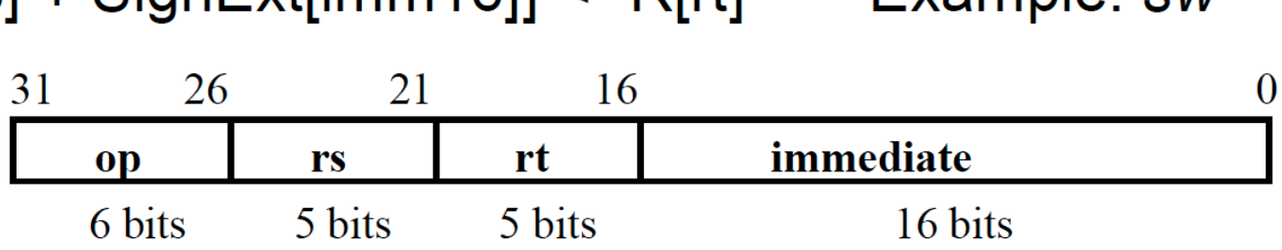


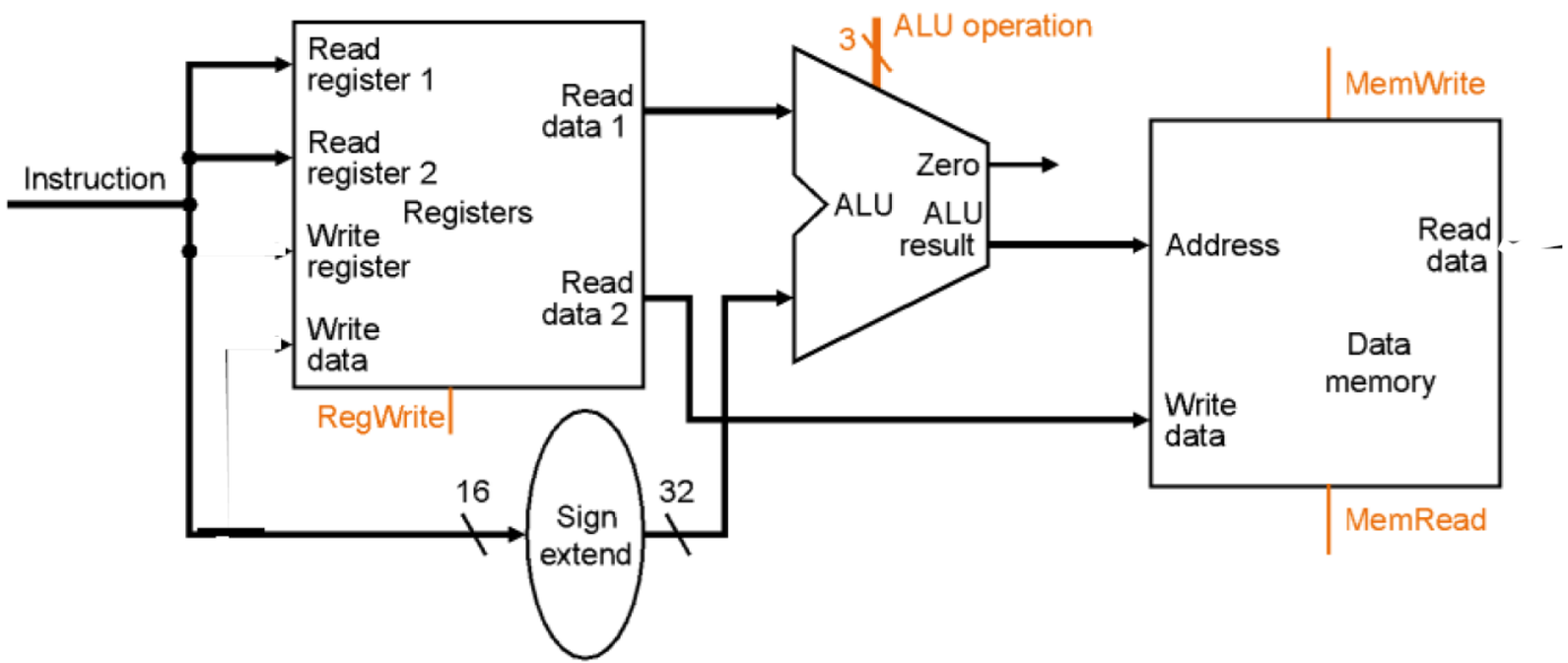


Review

- **Instructions**
  - rw, lw (I-type)

Mem[R[rs] + SignExt[imm16]] <- R[rt] Example: *sw rt, rs, imm16*

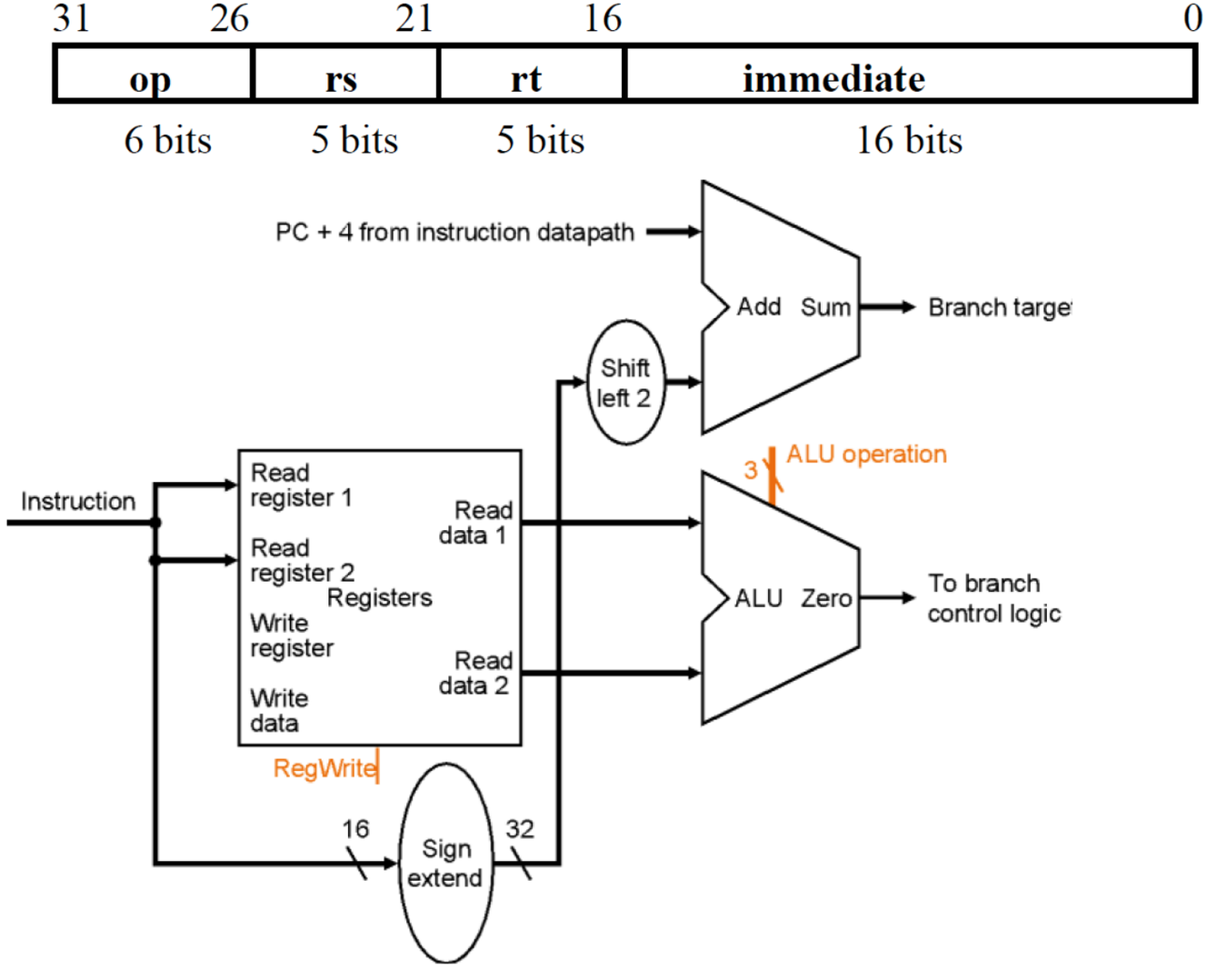




Review

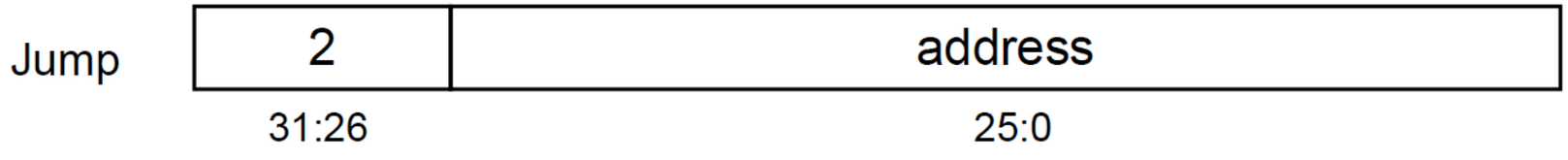
- **Instructions**
  - beq (I-type) *beq rs, rt, imm16*

We need to compare Rs and Rt

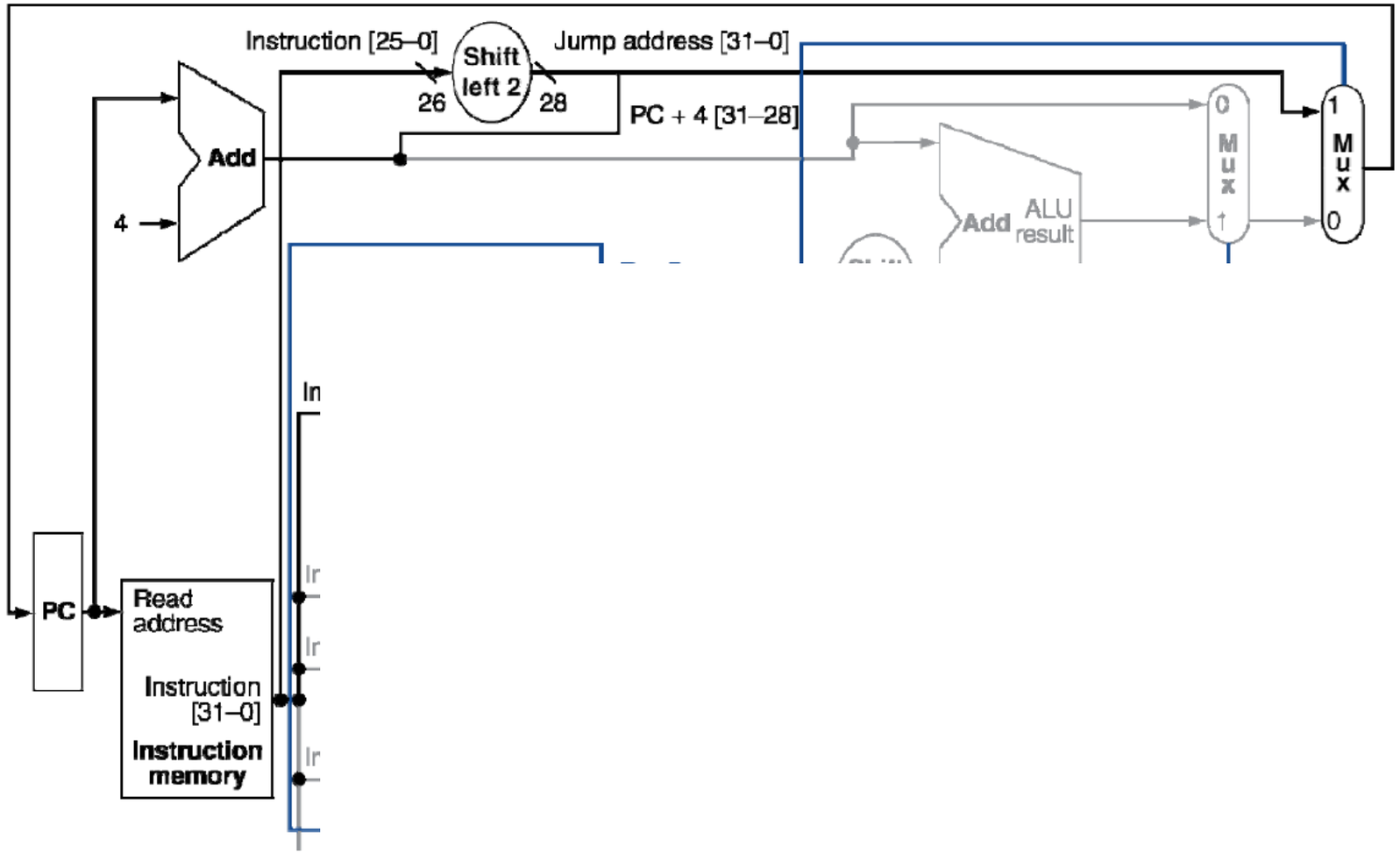


Review

- **Instructions**
  - jump (J-type)



Jump uses word address
  Update PC with concatenation of
    Top 4 bits of old PC
    26-bit jump address
    00

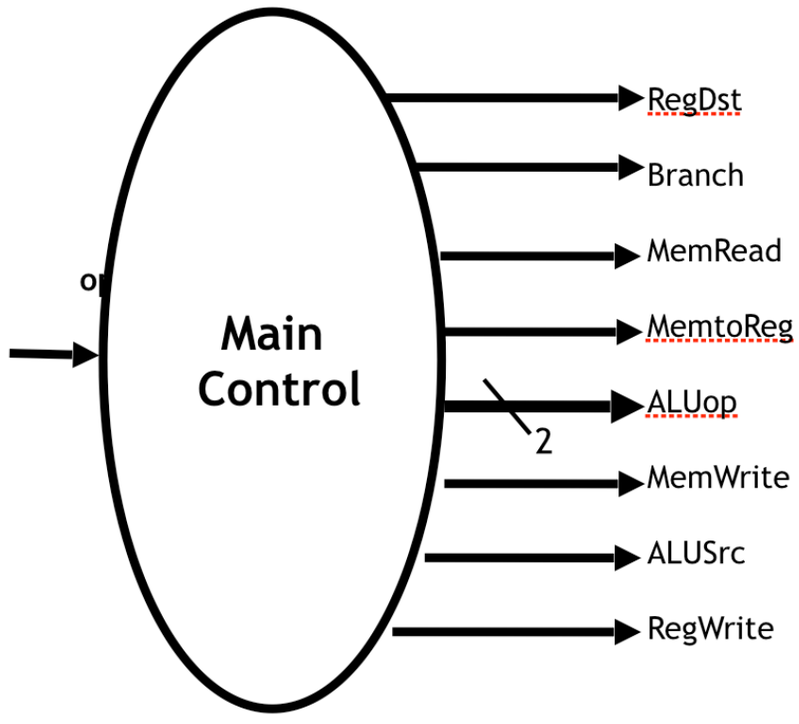


Review

- **Control Signal**

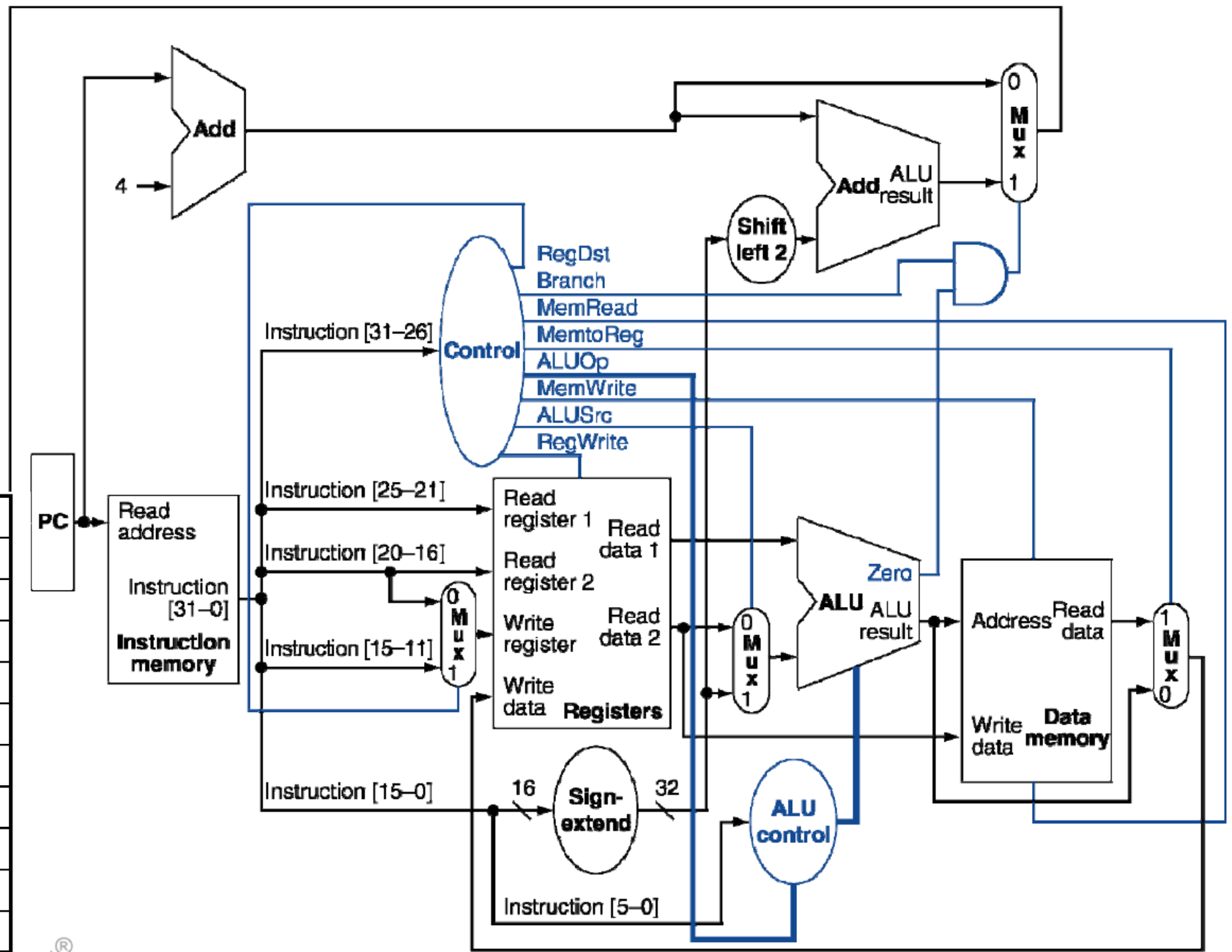
- What must be “controlled”?
  - Multiplexors (Muxes)
  - Writable state elements: Register File, Data Memory (Dmem)
  - ALU

Main Control has one 6-bit input, 9 outputs (7 are 1-bit, ALUOp is 2 bits)



Review

- **Control Signal**



		R-format	lw	sw	beq
Opcode		000000	100011	101011	000100
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
	ALUOp2	0	0	0	1

Sample question 1

- Refer to the following instructions:

a. AND Rd,Rs,Rt -> Reg[Rd]=Reg[Rs] AND Reg[Rt]

b. SW Rt,Offs(Rs) -> Mem[Reg[Rs] +Offs]=Reg[Rt]

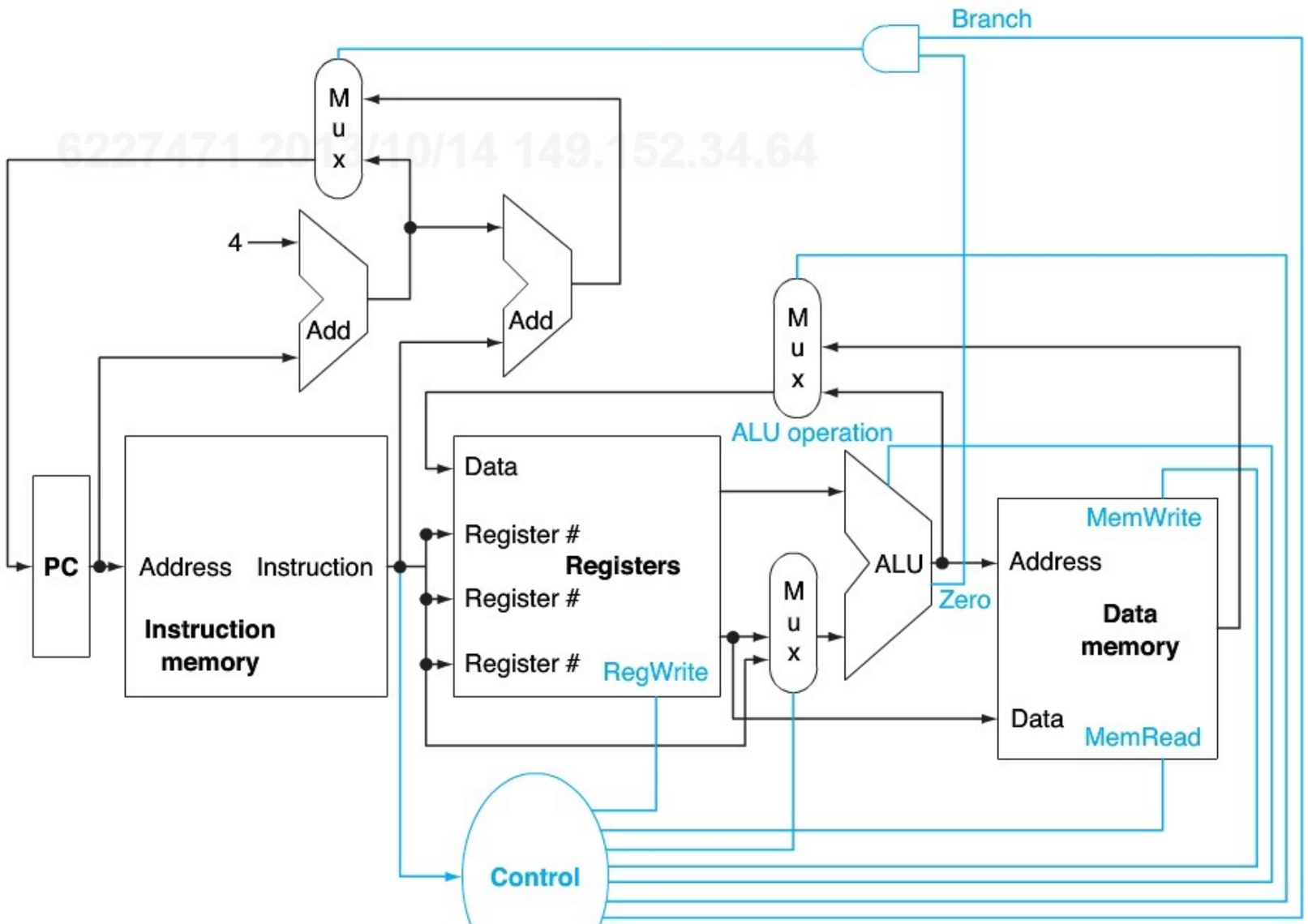
Answer the questions with respect to the figure in the next slide

Sample question 1

- What are the values of control signals generated by the control in the figure for these instructions?

AND Rd,Rs,Rt

SW Rt,Offs(Rs)



opcode	ALUOp	instruction
lw	00	load word
sw	00	store word
beq	01	branch equal
R-type	10	add
R-type	10	subtract
R-type	10	AND
R-type	10	OR
R-type	10	SLT

Sample question 1

RegDst	Branch	MemRead	MemtoReg	ALUOp	MemWrite	ALUSrc	RegW
1	0	0	0	10	0	0	1
0	0	1	1	00	0	1	1

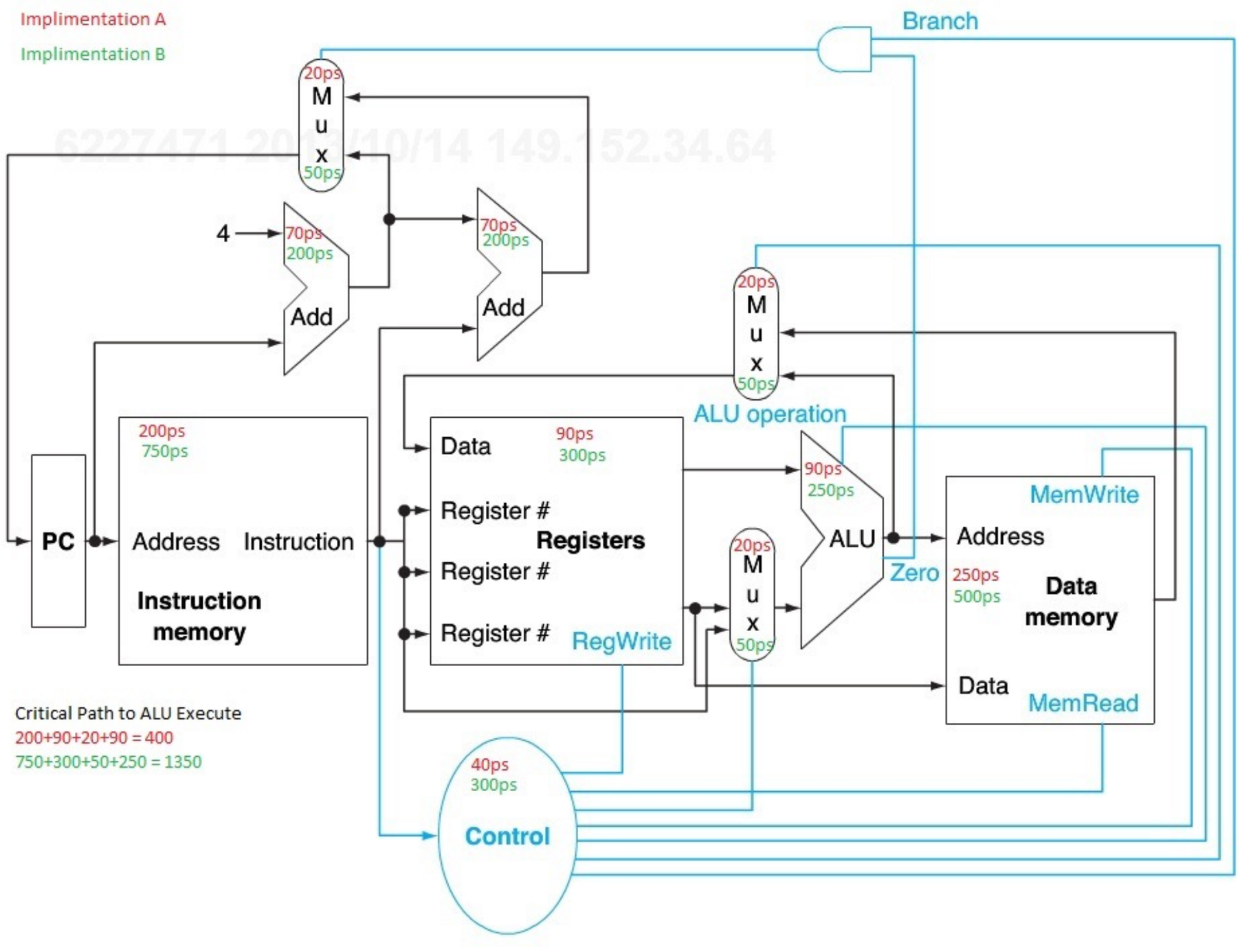
Sample question 1

Assume the following latencies for each resource

I-Mem	Add	Mux	ALU	Regs	D-Mem	Control
a. 200ps	70ps	20ps	90ps	90ps	250ps	40ps
b. 750ps	200ps	50ps	250ps	300ps	500ps	300ps

A. What is the critical path and minimum latency for a MIPS AND instruction?
B. What is the critical path for a MIPS load (LD) inst ruction?
C. What is the critical path for an MIPS BEQ instruct ion?

Sample question 1



A. a)200+90+20+90+20 = 420ps

b) 750+300+50+250+50=1400ps

B. a)200+90+20+90+250+20=670ps

b) 750+300+50+250+500+50=1900ps

C. a)200+90+20+90+20 = 420ps

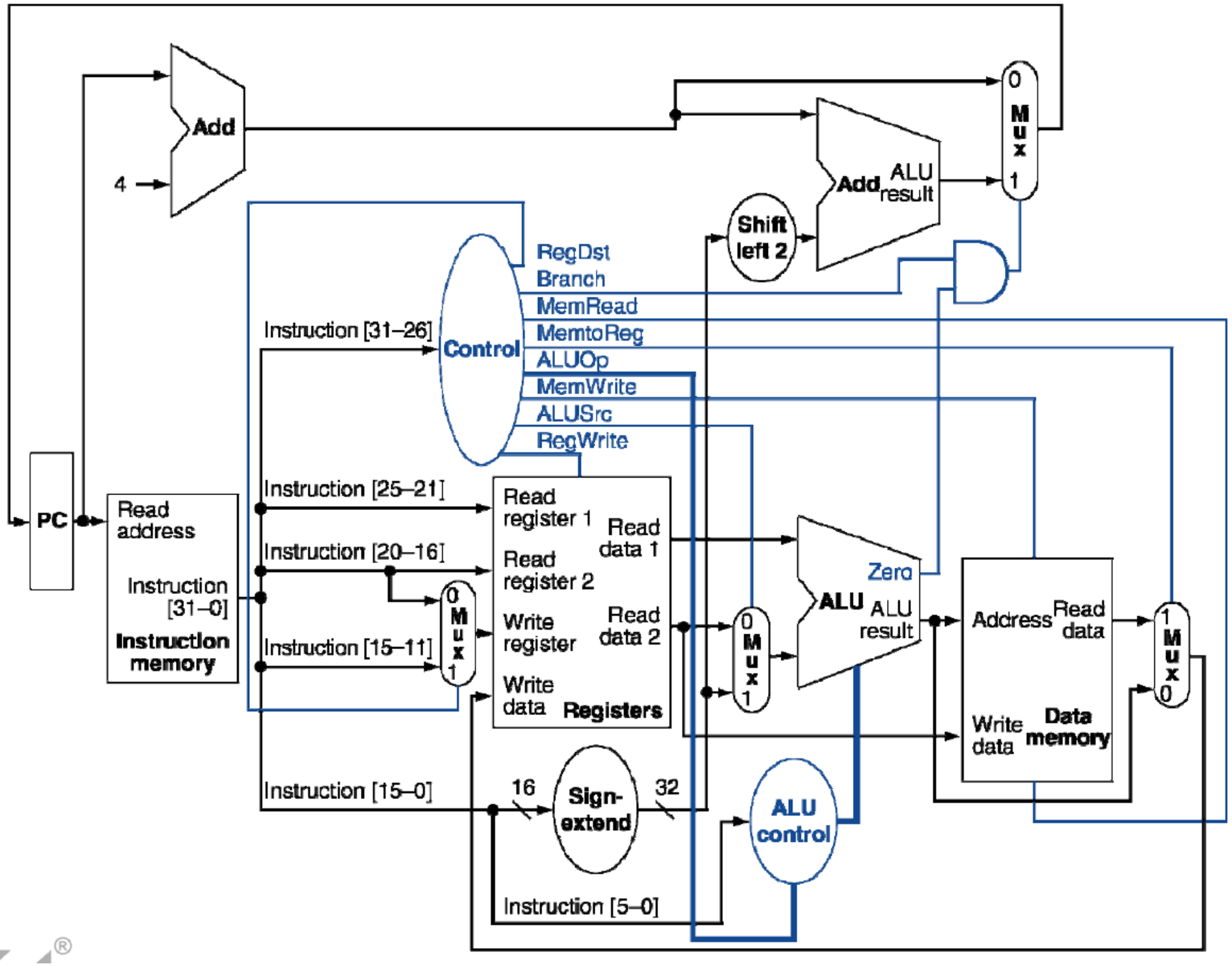
b) 750+300+50+250+50=1400ps

Sample question 2

- The basic single-cycle MIPS implementation can only implement some instructions. New instructions can be added to an existing ISA, but the decision whether or not to do that depends, among other things, on the cost and complexity such an addition introduces into the processor datapath and control. The new instructions are shown below:

jri (jump register + immediate)

- PC = R[rs] + 16-bit Sign Extend(I)
- jri $s0 IMM : PC = R[$s0] + Sign Extend(IMM)


Add
4
0
M
u
x
1
Add
ALU result
Shift left 2
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [31–26]
Control
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
Zero
ALU
ALU result
ALU control
Address
Read data
Write data
Data memory

Sample question 2

What type of instruction should this be?

Sample question 2

What type of instruction should this be?

We need a register and an immediate. I-type fits the bill.

– We'd have a unused register if we implement it as an I-type. We could possibly optimize by defining a new instruction type that has one register and one immediate.

Trade off?

Sample question 2

What new connections must be made?

Sample question 2

What new connections must be made?

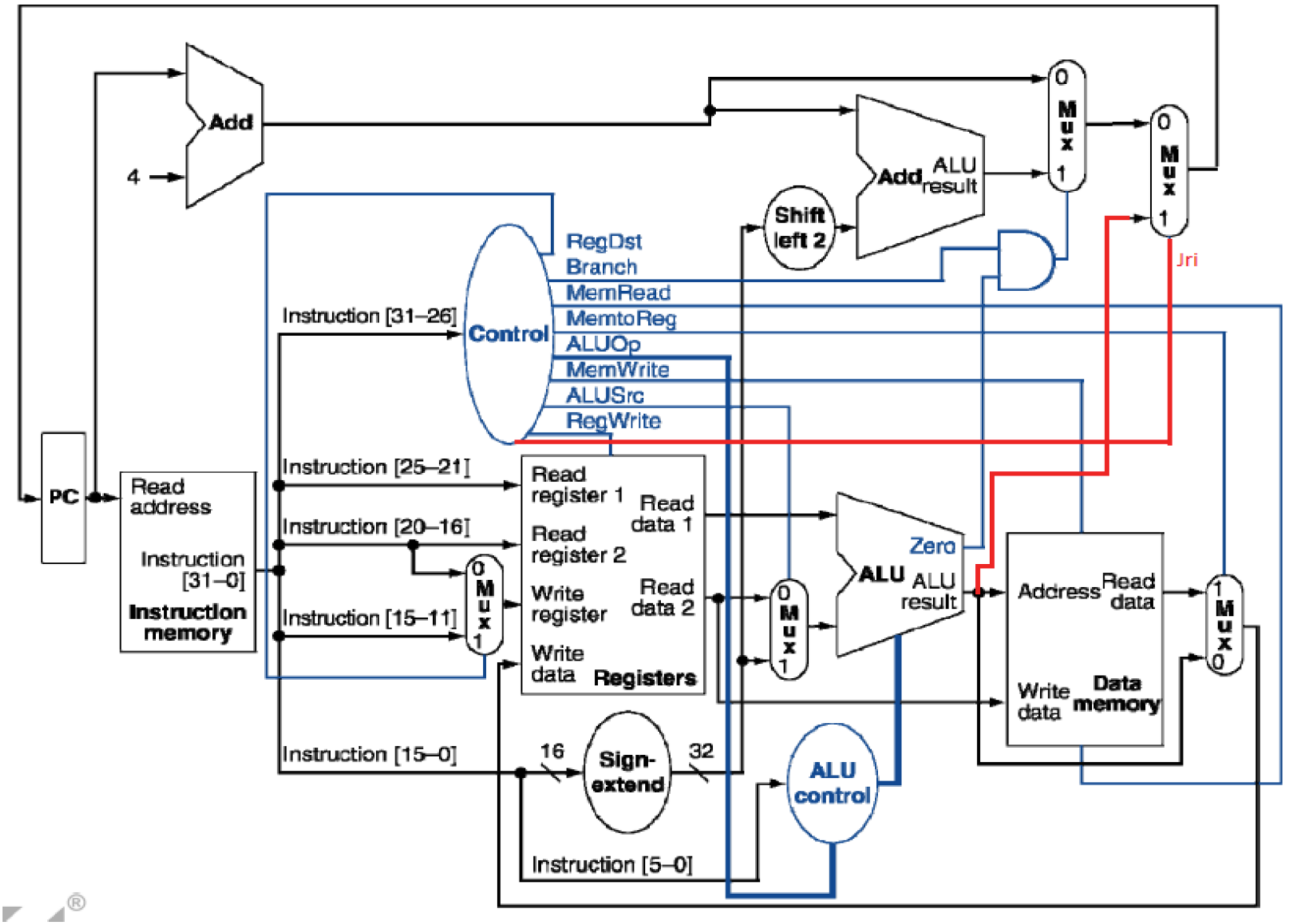
Additional connections:

- From output of ALU to PC
- Already hooked up in such a way as to do addition of register value and a sign extended immediate.

- How do we choose the output of the ALU as the input to the PC?

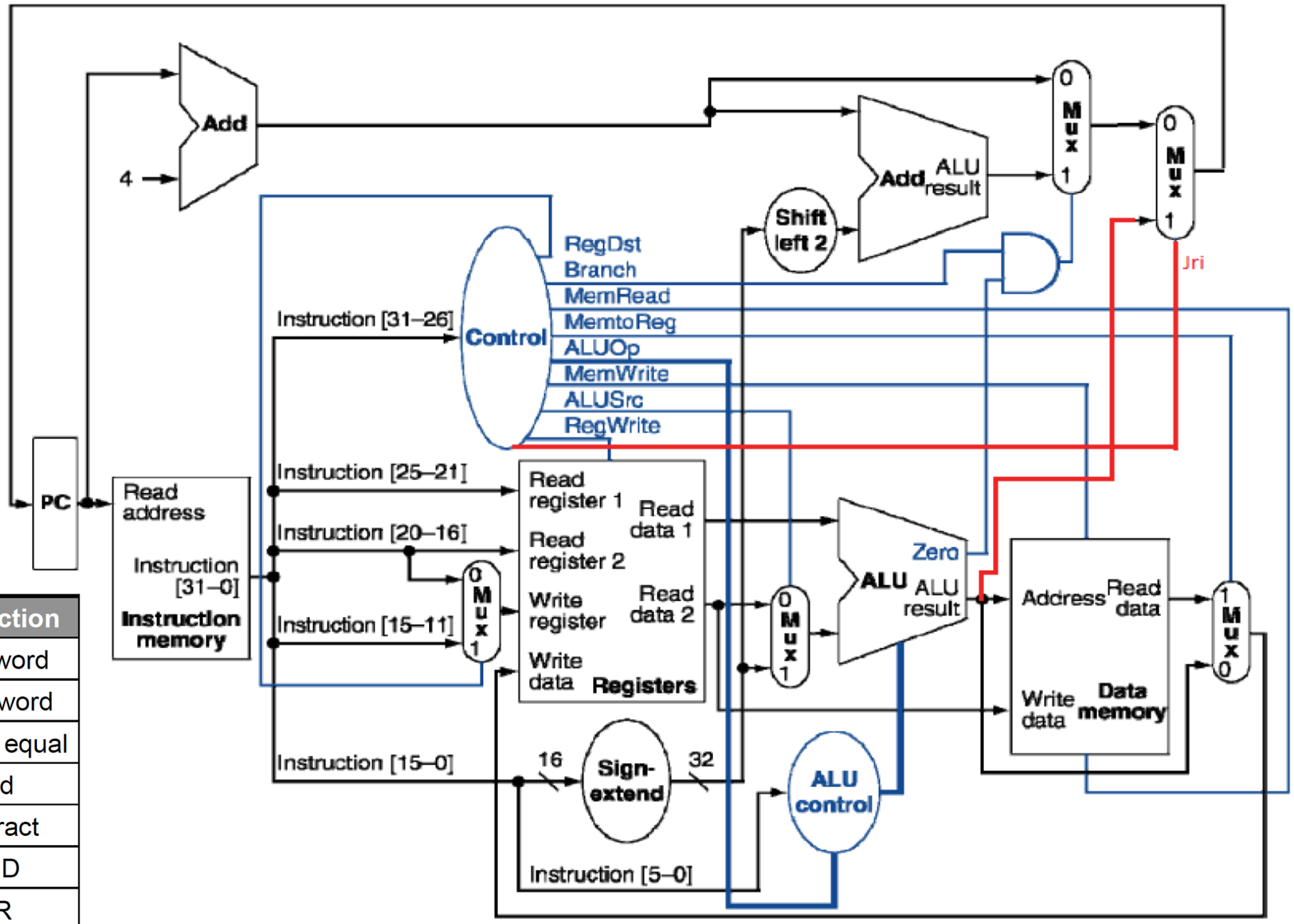
Need a MUX somewhere before the PC.

Presumably also a new control signal to indicate jri.


Add
4
0
M
u
x
1
Add
ALU result
Shift left 2
0
M
u
x
1
Jri
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [31–26]
Control
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 control
Zero
ALU
ALU result
Address
Read data
Write data
Data memory
1
M
u
x
0

Sample question 2

What are the control signals?



opcode	ALUOp	instruction
lw	00	load word
sw	00	store word
beq	01	branch equal
R-type	10	add
R-type	10	subtract
R-type	10	AND
R-type	10	OR
R-type	10	SLT

Sample question 2

- What are the control signals?
  - RegDst = don't care, not writing to registers
  - Branch = don't care, choosing from ALU output.
- It's okay if the branch signal ends up high since if we assume jri is high, then we won't use the original branch result.
  - MemRead = 0, not reading from mem
  - MemToReg = don't care, not writing to registers
  - MemWrite = 0, not writing to mem
  - ALUSrc = 1, add immediate
  - RegWrite = 0, not writing to regsiters
  - Jri = 1, because of course
  - **ALUOp = 00, add regardless of funct**

Sample question 2

- General strategy:
- What does the instruction do that the existing datapath already can?
  - Ex. Does it write to R[Rt]? Great. We can use that.
- What does the instruction do that the existing datapath currently CAN'T do?
  - Ex. Does it use M[x] as an input to the ALU?
  - Add a multiplexer in front of the port and connect the M[x] output to the input of the mux
- Determine the control signals required to make the new instruction work. Make sure everything else works.

Sample question 2

- The requirements:
  - Both the new and the old instructions should work given the correct control signals. The signals for old instructions probably shouldn't change.
- Optimality concerns you should worry about:
  - Hardware usage: if there is a solution that uses one mux and you have a solution that uses two muxes and an AND gate, you may(?) lose points?
- Optimality concerns you DON'T need to worry about:
  - The exact signals: if there is a solution that has a signal that is “don't care” and you have a solution where that signal must be 0, that's fine.