Datapath

Discussion 8

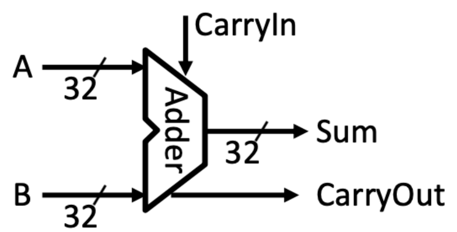
CS110

Yutong Wang

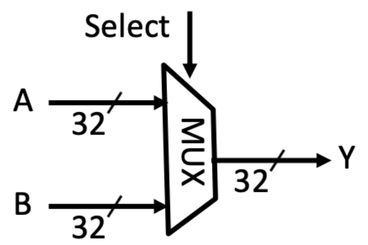
4/12/2024

CAS4ET Lab

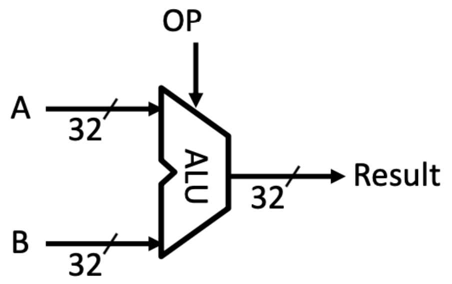
Datapath Components



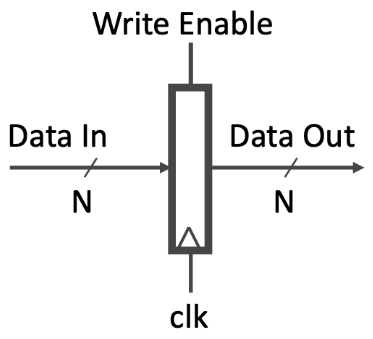
Adder



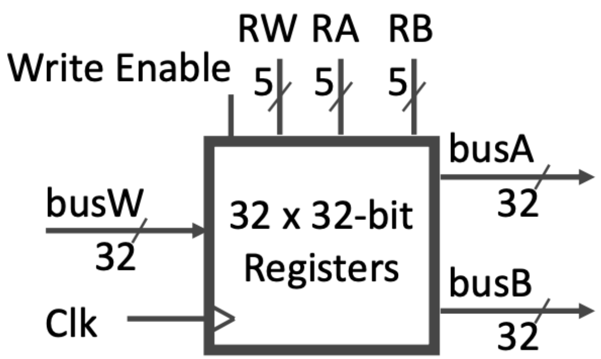
Multiplexer



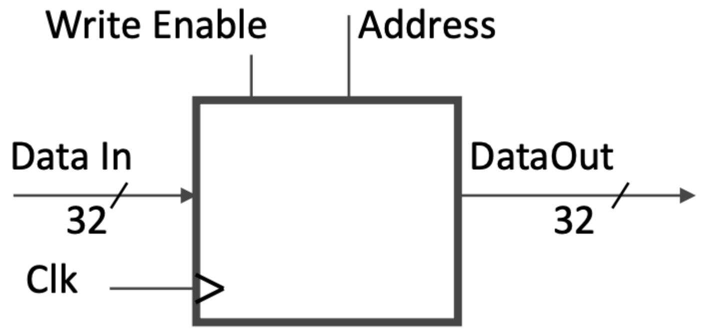
ALU



register



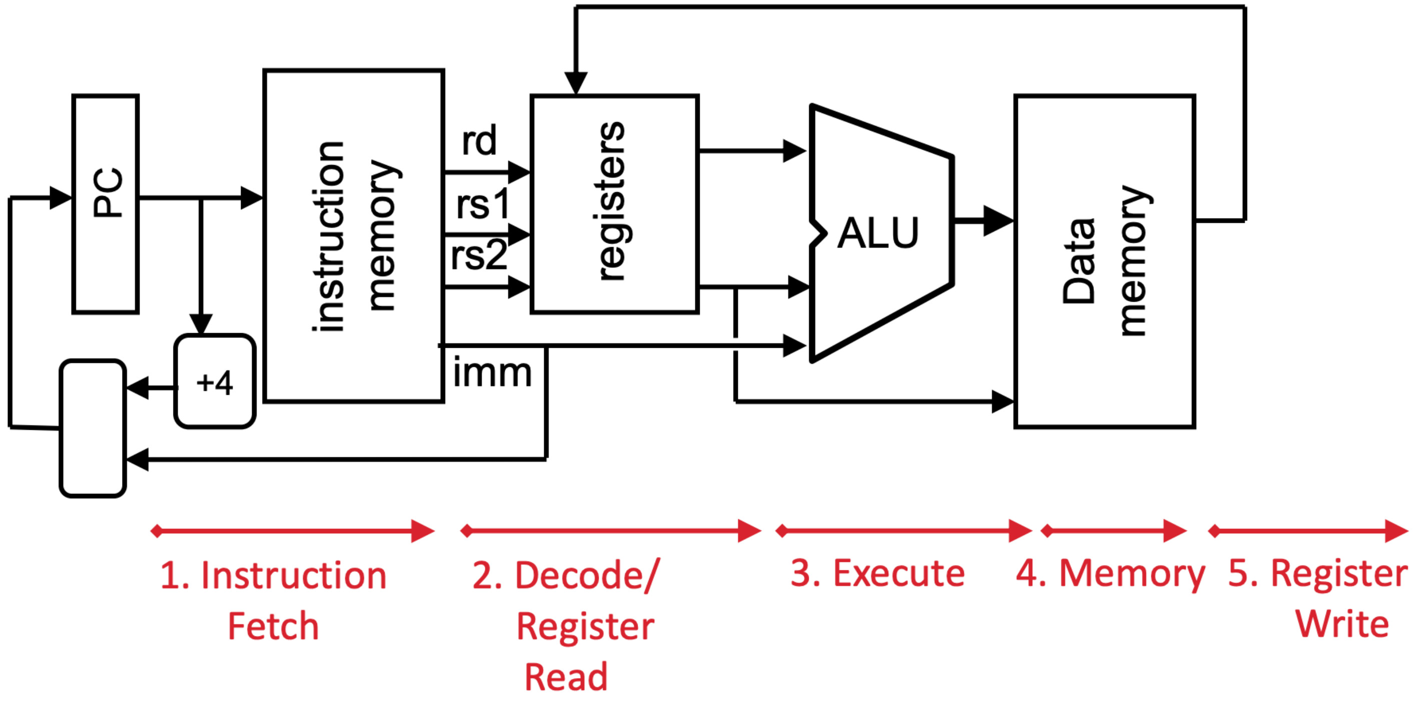
register file



memory

5 Stages of Datapath

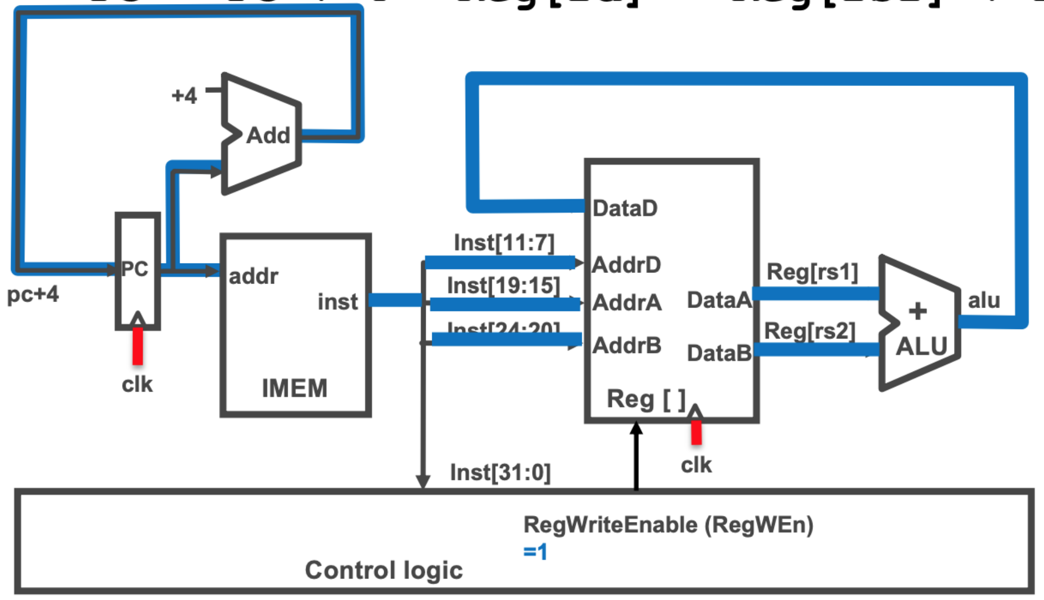
- **IF**: **I**nstruction **F**etch
- **ID**: **I**nstruction **D**ecode
- **EX**: **Ex**ecute
- **MEM**: **Mem**ory
- **WB**: **W**rite **B**ack



R-format: add

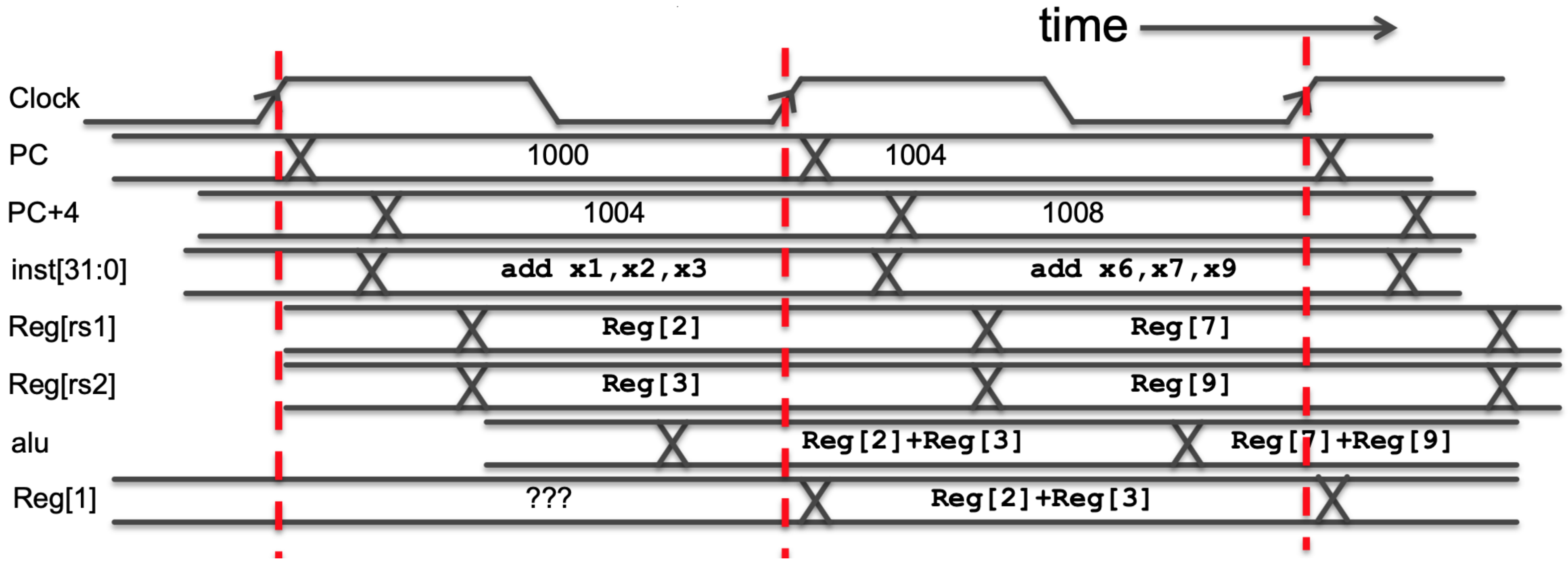
- **IF**
- **ID**
- **EX**: alu = R[rs1] + R[rs2]
- ~~**MEM**~~
- **WB**: R[rd] = alu
- PC = PC + 4

PC = PC + 4   Reg[rd] = Reg[rs1] + Reg[rs2]



31 25	24 20	19 15	14 12	11 7	6 0
0000000	rs2	rs1	000	rd	opcode
add	5	5	add	5	Reg-Reg OP

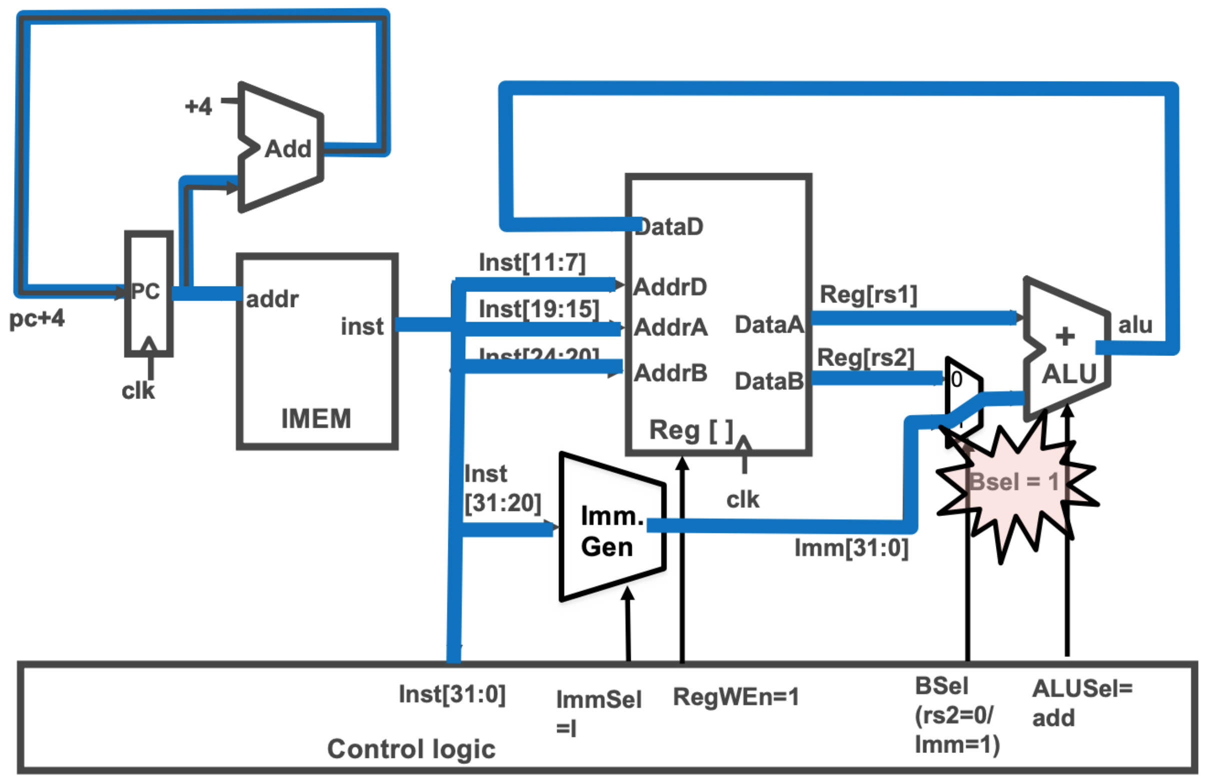
Time Fiagram



Attention! **WB** actually finishes after the next rising edge of Clock.

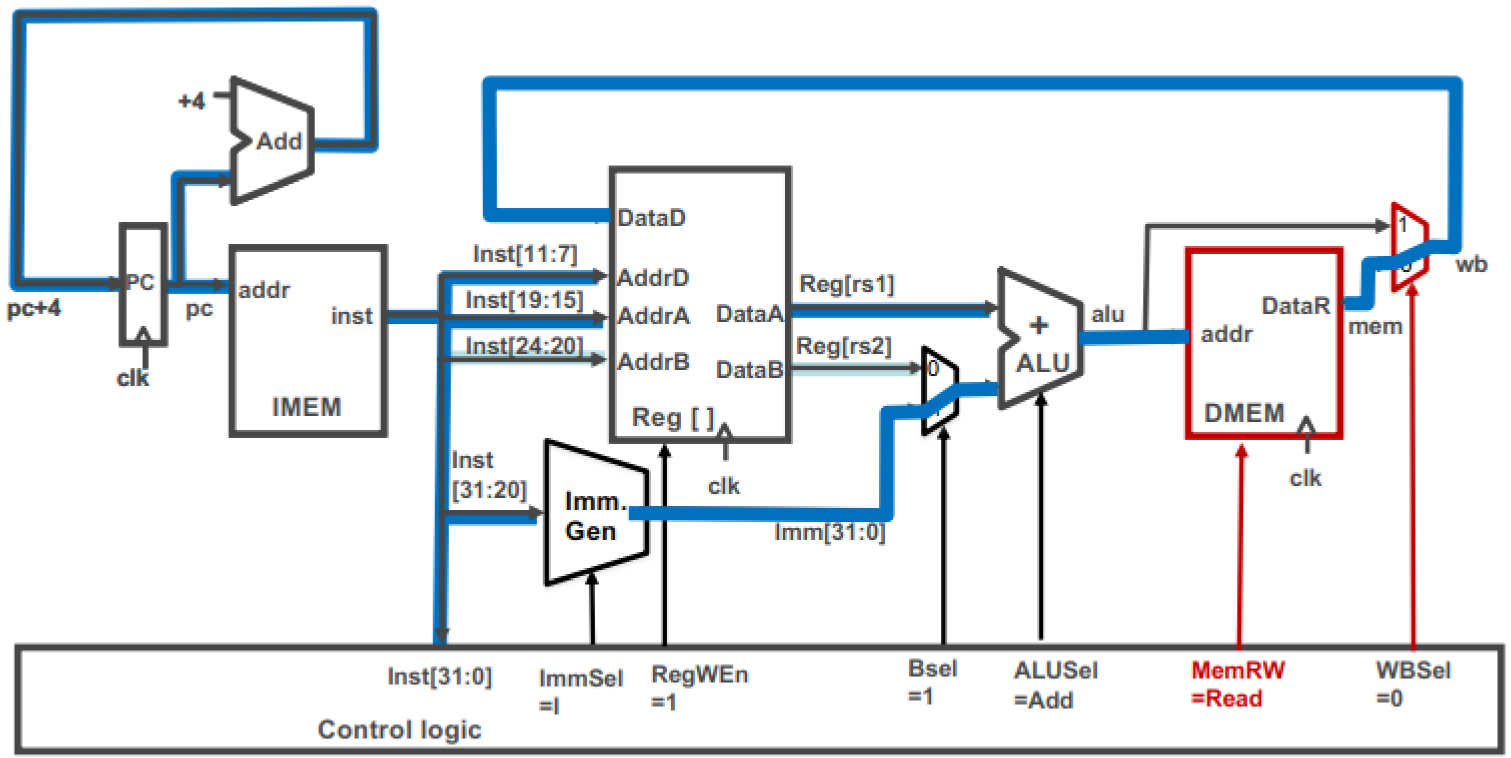
I-format: addi

- **IF**
- **ID**
- **EX**: alu = R[rs1] + imm
- ~~**MEM**~~
- **WB**: R[rd] = alu
- PC = PC + 4



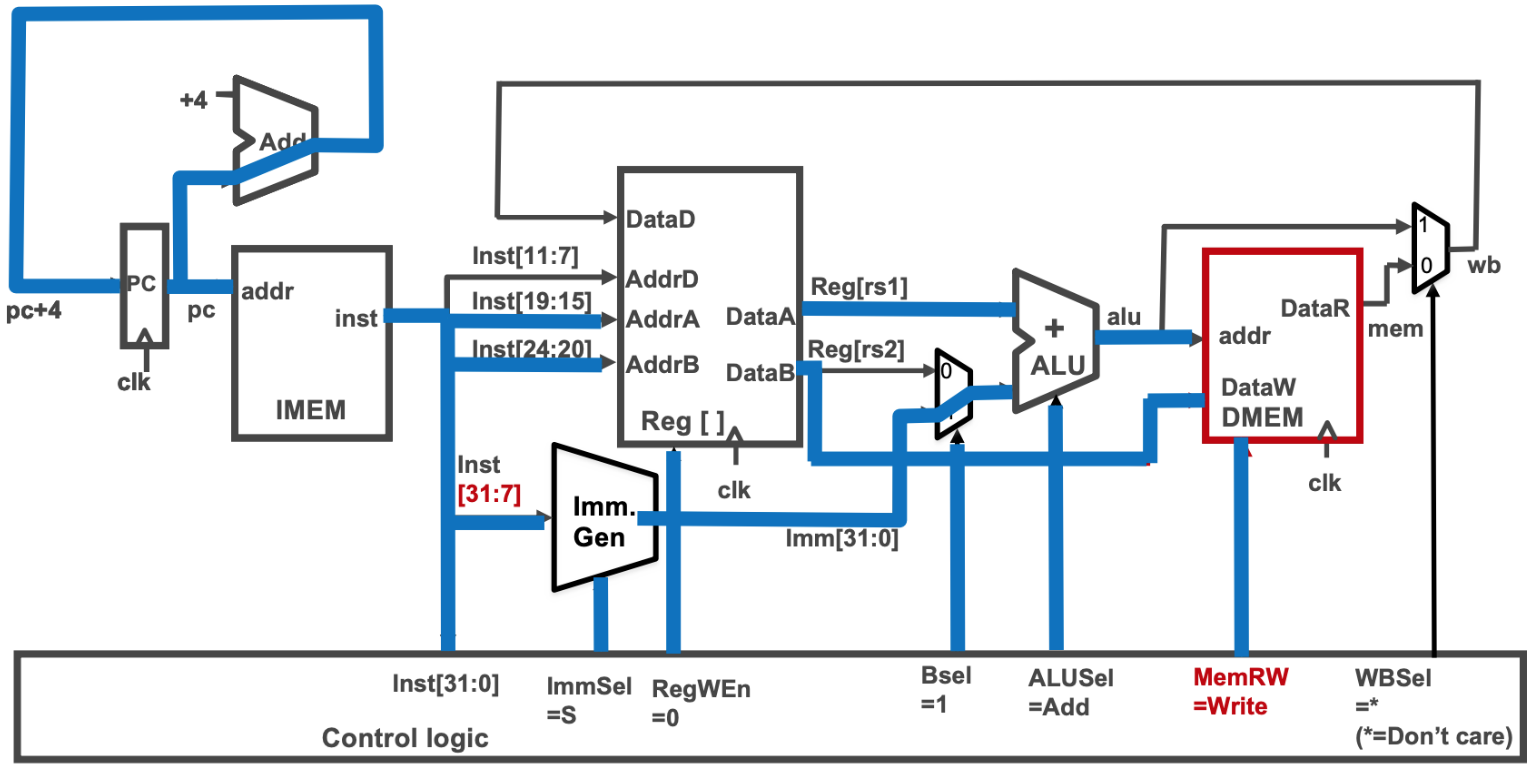
I-format: lw

- **IF**
- **ID**
- **EX**: alu = R[rs1] + imm
- **MEM**: mem = M[alu]
- **WB**: R[rd] = mem
- PC = PC + 4



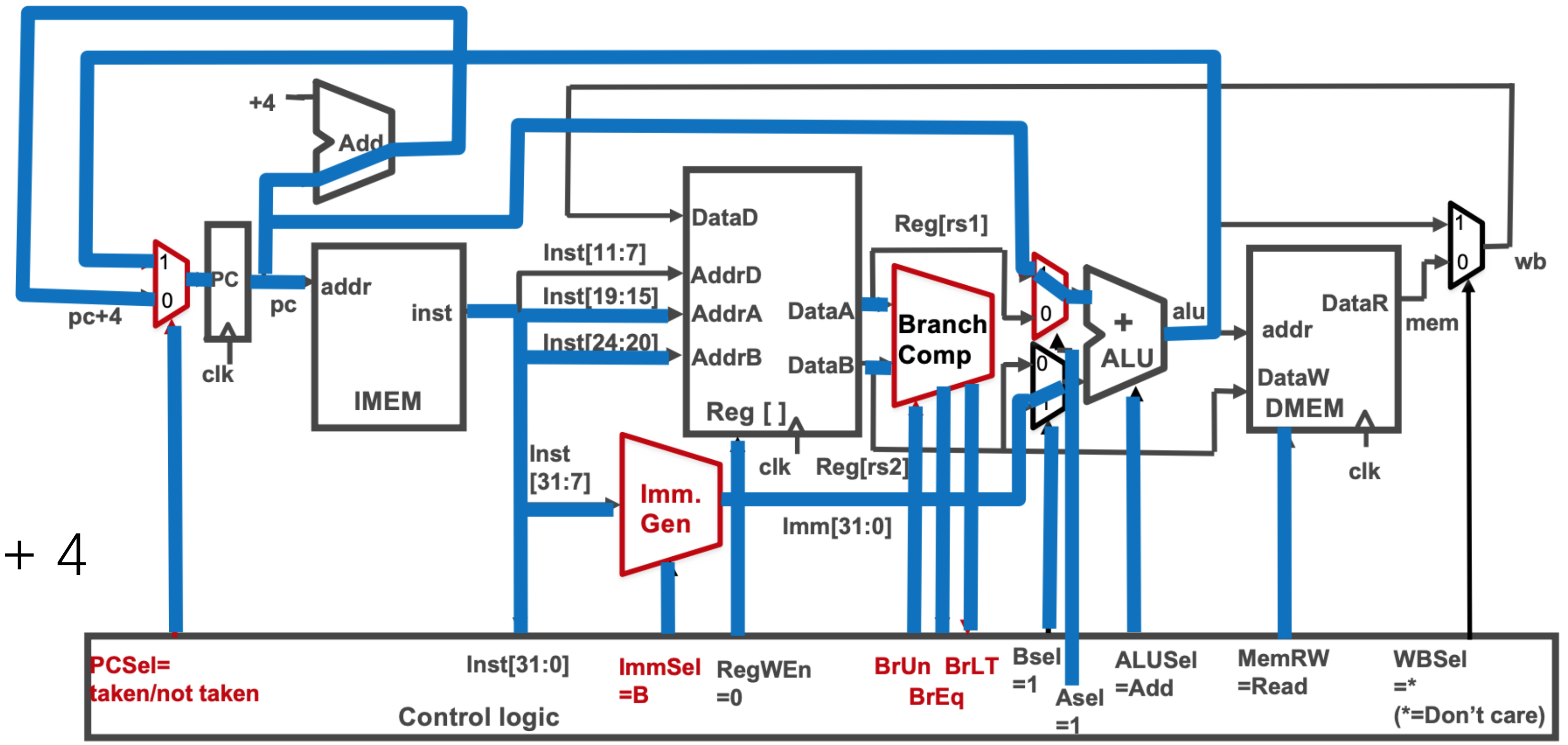
S-format: sw

- **IF**
- **ID**
- **EX**: alu = R[rs1] + imm
- **MEM**: M[alu] = R[rs2]
- ~~**WB**~~
- PC = PC + 4



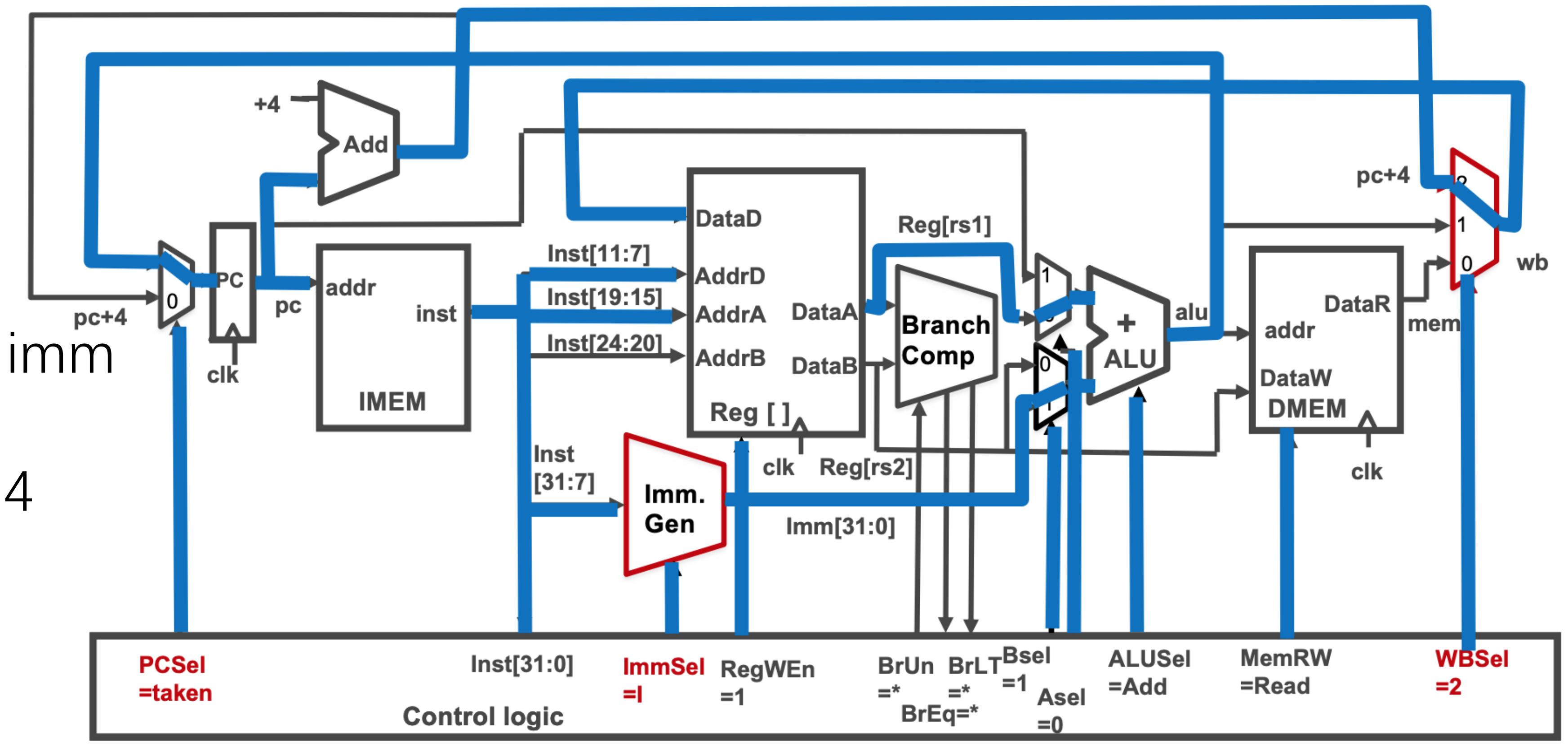
B-format

- **IF**
- **ID**
- **EX**: alu = PC + imm
- ~~**MEM**~~
- ~~**WB**~~
- PC = PCSel ? alu : PC + 4



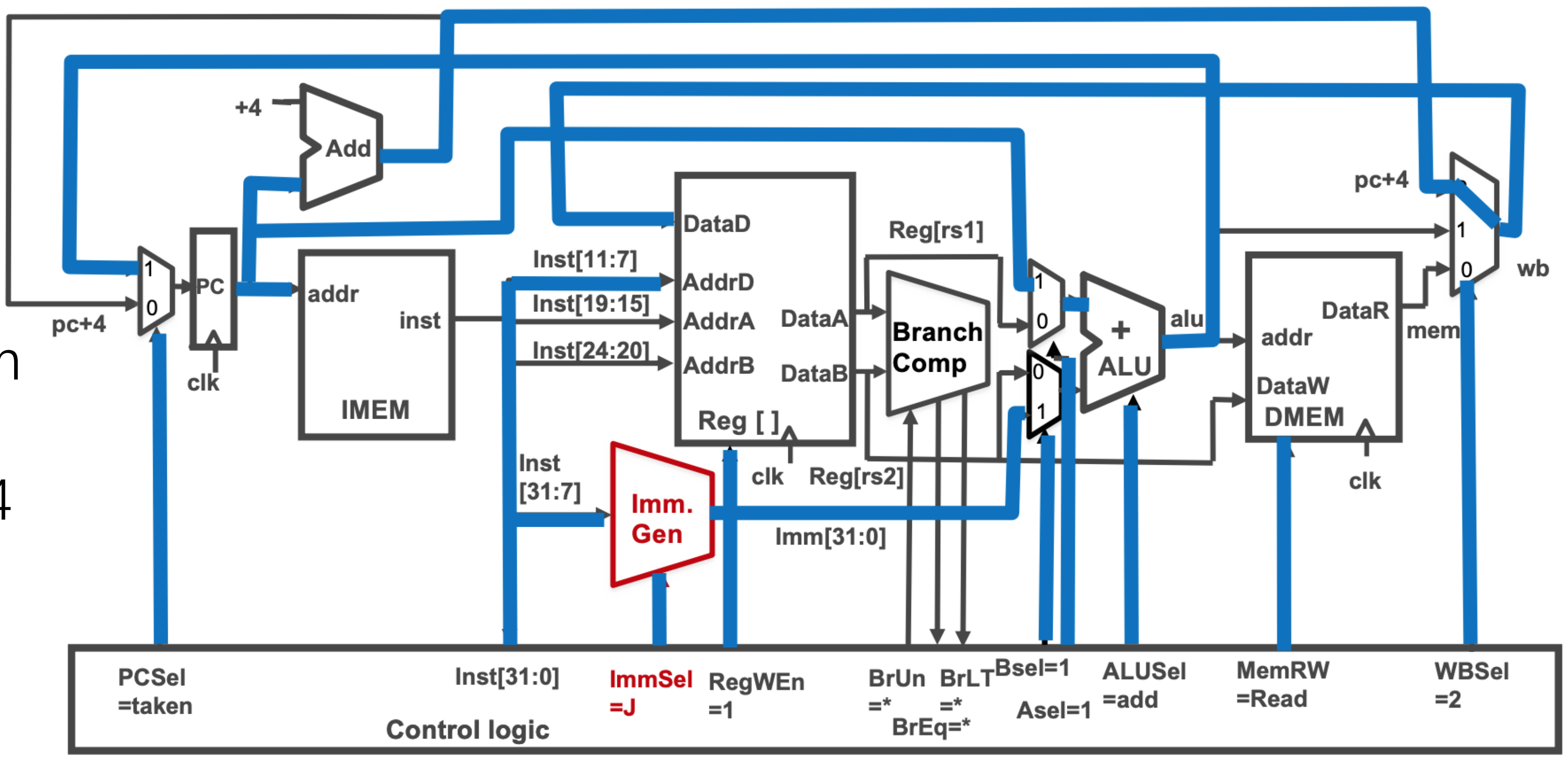
jalr

- **IF**
- **ID**
- **EX**: alu = R[rs1] + imm
- ~~**MEM**~~
- **WB**: R[rd] = PC + 4
- PC = alu



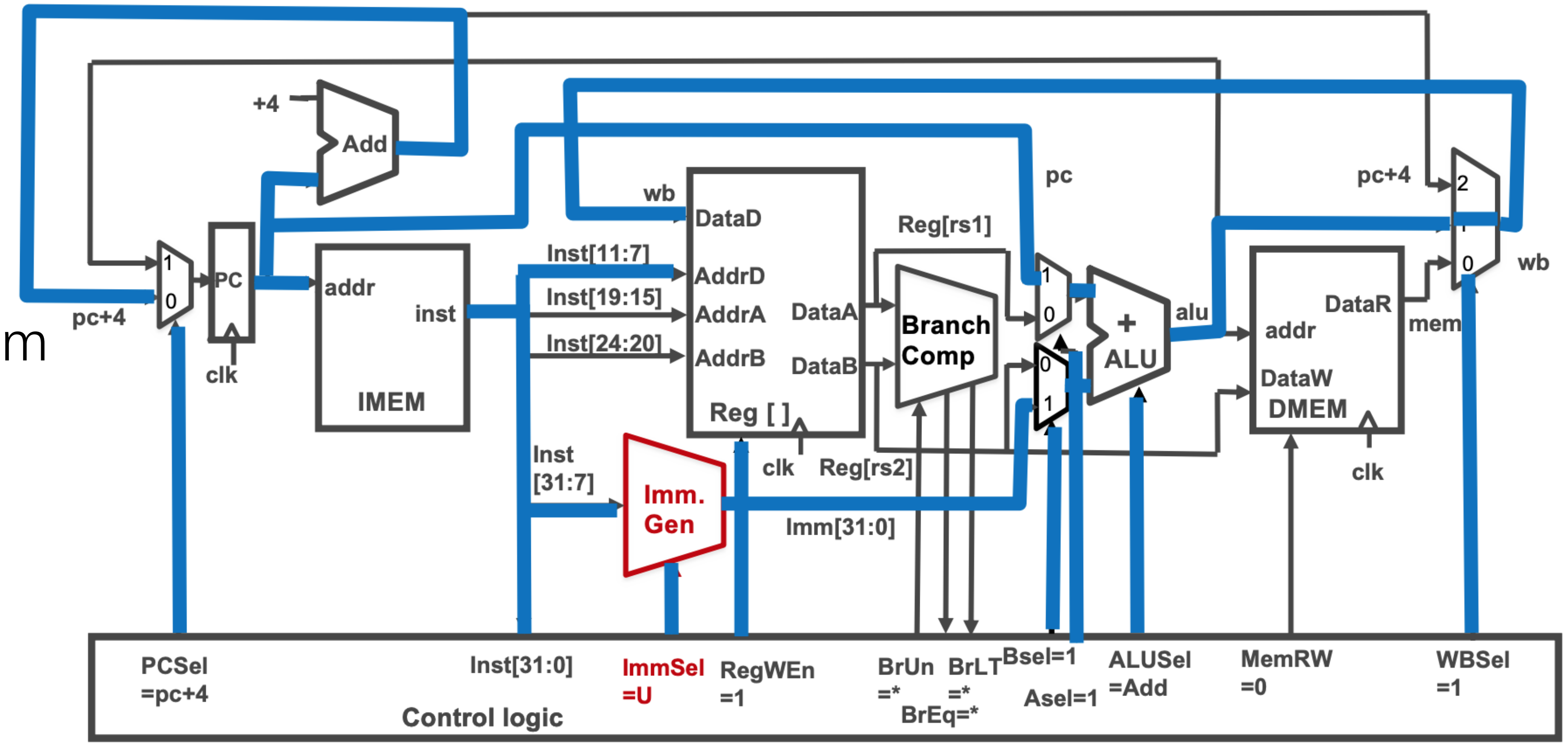
jal

- **IF**
- **ID**
- **EX**: alu = PC + imm
- ~~**MEM**~~
- **WB**: R[rd] = PC + 4
- PC = alu



U-format: auipc

- **IF**
- **ID**
- **EX**: alu = PC + imm
- ~~**MEM**~~
- **WB**: R[rd] = alu
- PC = PC + 4



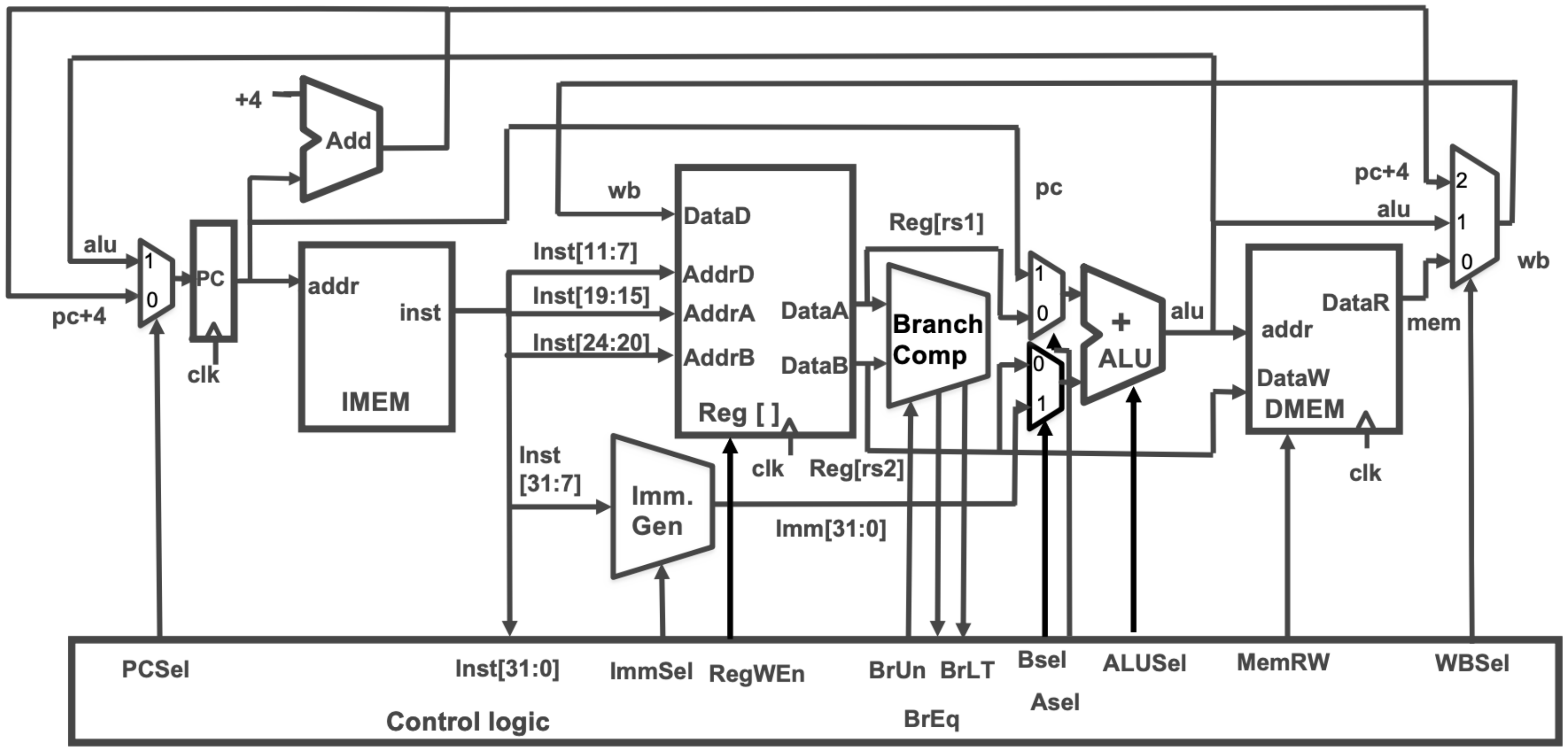
Example from CA 2022 final

For instruction `jalr ra`, select the correct value for the control logic.

1. PCSel:
A. alu
B. pc + 4

2. Asel:
A. 0
B. 1

3. WBsel:
A. pc + 4
B. alu
C. mem



Example from CA 2022 final

For instruction `jalr ra`, select the correct value for the control logic.

1. PCSel:

A. alu

B. pc + 4

2. Asel:

A. 0

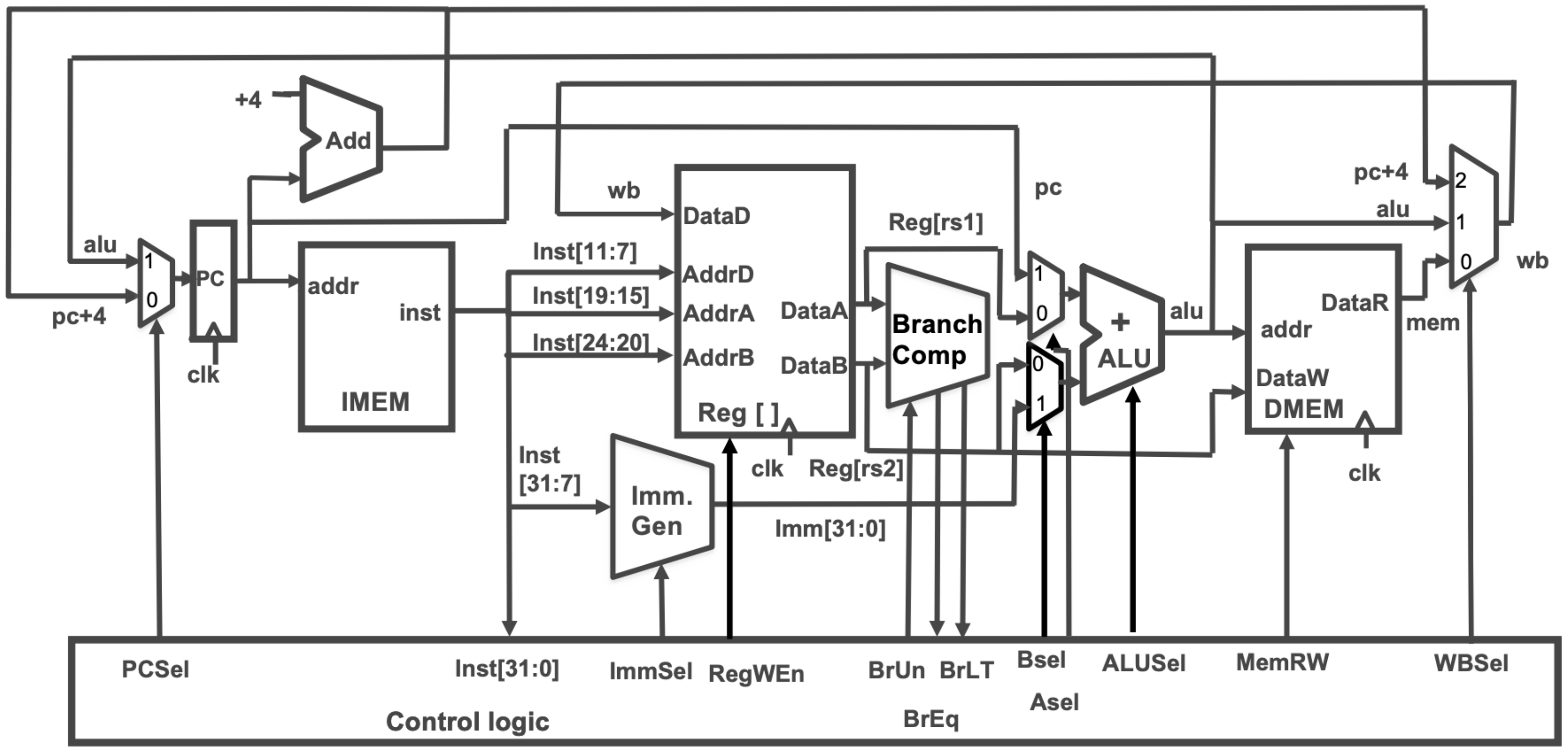
B. 1

3. WBsel:

A. pc + 4

B. alu

C. mem



Example from CA 2022 final

(True or False)

1. ( ) For instruction add, the value is written back to Reg as soon as it is computed by ALU.

2. ( ) Except Write Enable, all the input and output buses of register file are 32-bit.

3. ( ) For register file and memory, CLK is a factor ONLY during write operation.

4. ( ) Register file holds all the registers needed for instruction execution.

Example from CA 2022 final

(True or False)

1. (**F**) For instruction add, the value is written back to Reg as soon as it is computed by ALU.

2. (**F**) Except Write Enable, all the input and output buses of register file are 32-bit.

3. (**T**) For register file and memory, CLK is a factor ONLY during write operation.

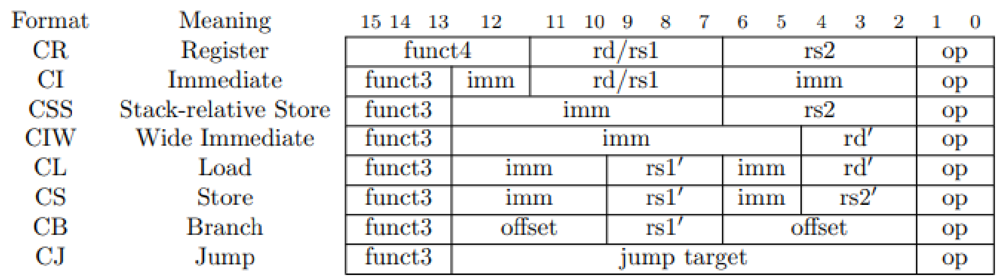
4. (**F**) Register file holds all the registers needed for instruction execution.

Project 2: A RV32C Toy CPU

2.1: Implement CI and CR type instructions

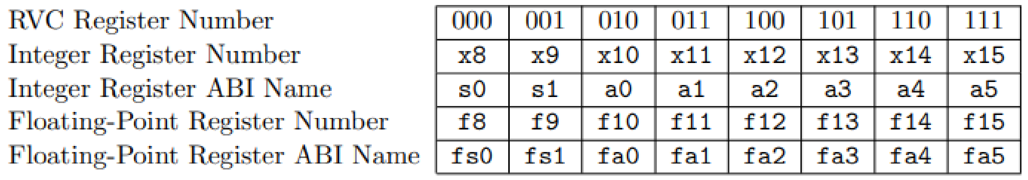
(DDL: May 7th)

Format	ISA	OP[1:0]	Funct3	Implementation
CR	add	10	100	x[rd] = x[rd] + x[rs2]
CR	mv	10	100	x[rd] = x[rs2]
CI	addi	01	000	x[rd] = x[rd] + sext(imm)
CI	slli	10	000	x[rd] = x[rd] << uimm
CI	li	01	010	x[rd] = sext(imm)
CI	lui	01	011	x[rd] = sext(imm[17:12] << 12)
CI	nop	01	000	None
CSS	swsp	10	110	M[x[2] + uimm][31:0] = x[rs2]
CIW	addi4spn	00	000	x[8+rd'] = x[2] + nzuimm
CL	lw	00	010	x[8+rd'] = sext(M[x[8+rs1'] + uimm][31:0])
CS	sw	00	110	M[x[8+rs1'] + uimm][31:0] = x[8+rs2']
CB	beqz	01	110	if (x[8+rs1'] == 0) pc += sext(offset)
CJ	j	01	101	pc += sext(offset)
CJ	jr	10	100	pc = x[rs1]



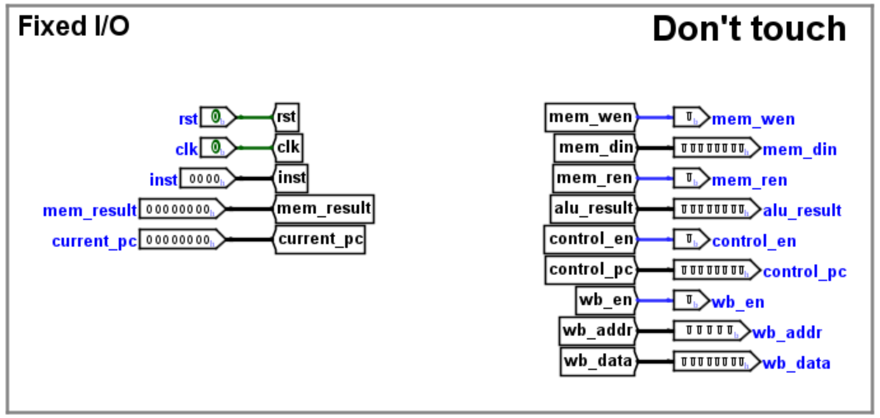
Format	Meaning	15 14 13	12	11 10	9 8 7	6 5	4 3 2	1 0
CR	Register	funct4		rd/rs1		rs2		op
CI	Immediate	funct3	imm	rd/rs1		imm		op
CSS	Stack-relative Store	funct3	imm			rs2		op
CIW	Wide Immediate	funct3	imm				rd′	op
CL	Load	funct3	imm		rs1′	imm	rd′	op
CS	Store	funct3	imm		rs1′	imm	rs2′	op
CB	Branch	funct3	offset		rs1′	offset		op
CJ	Jump	funct3	jump target					op

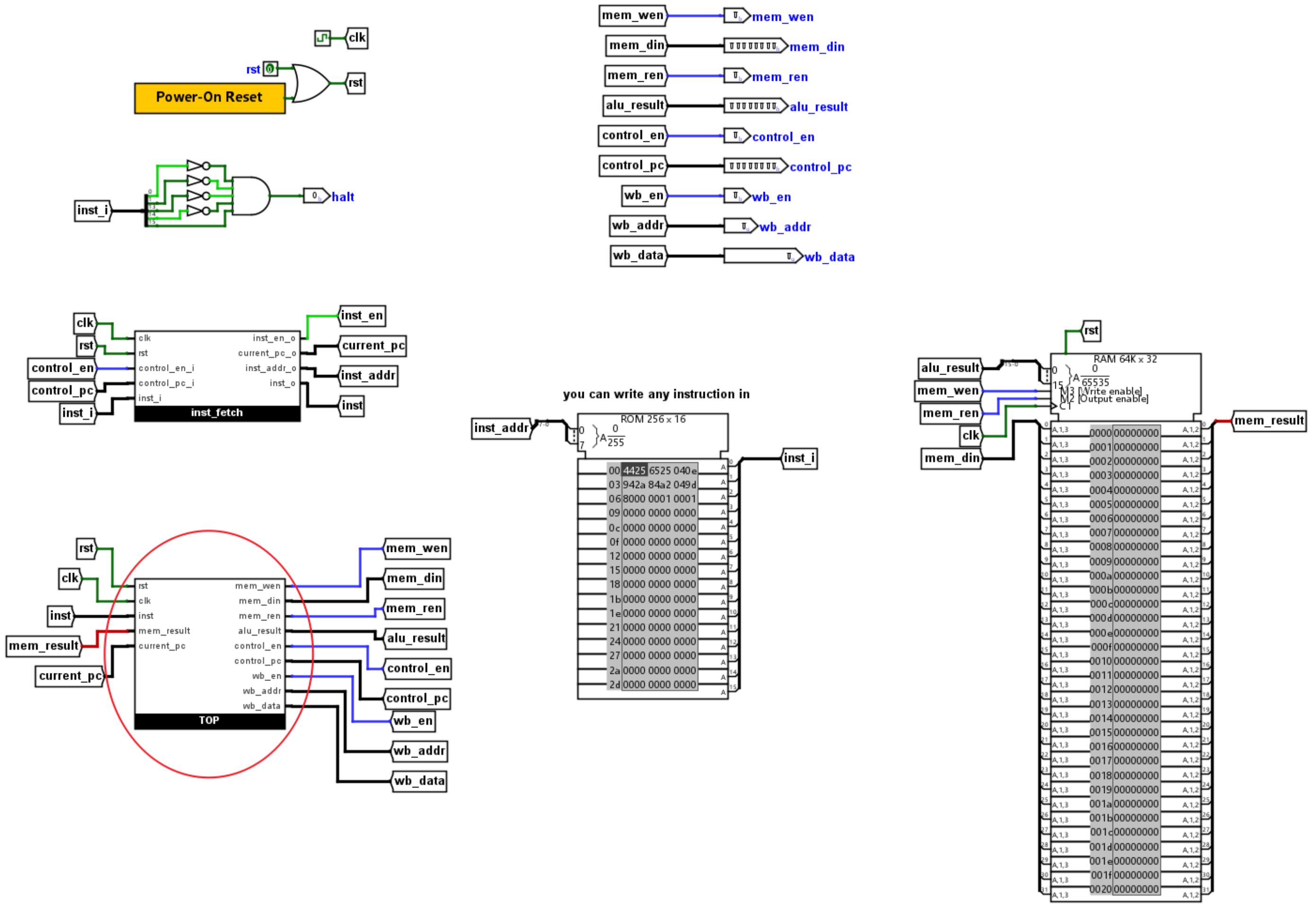
Table 1.1: Compressed 16-bit RVC instruction formats.



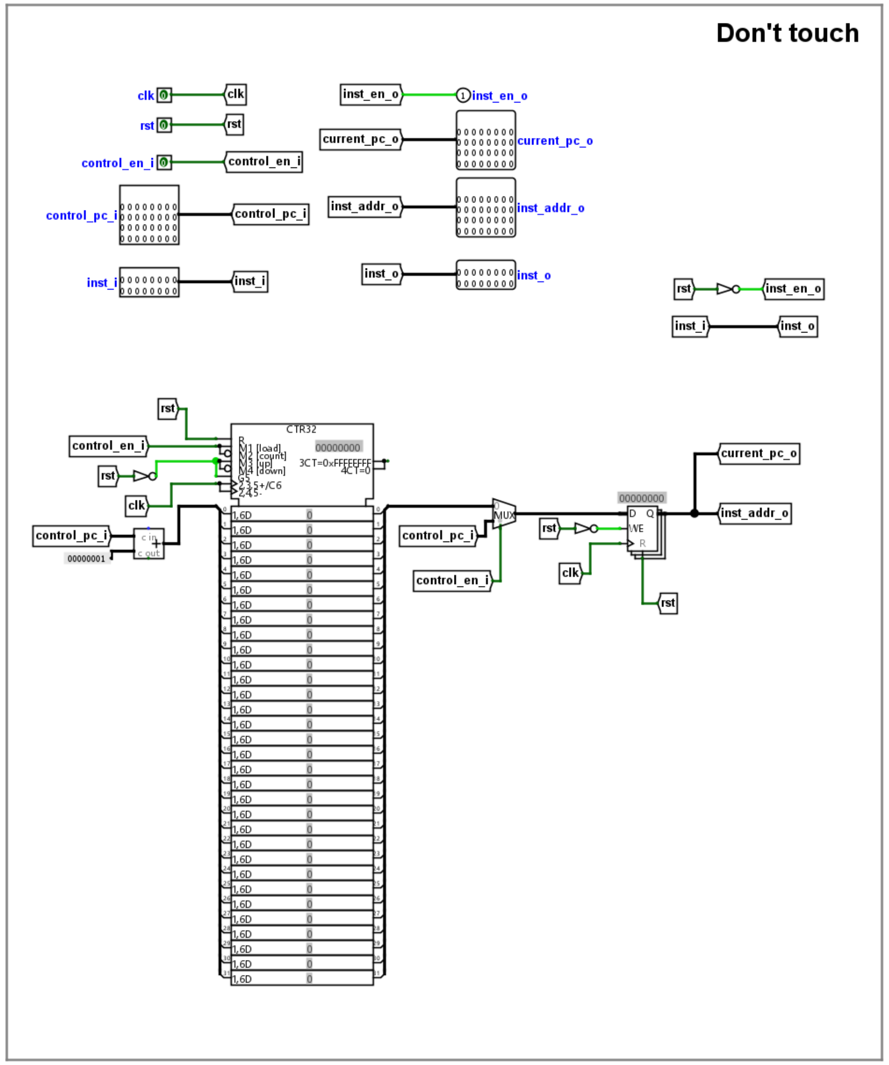
RVC Register Number	000	001	010	011	100	101	110	111
Integer Register Number	x8	x9	x10	x11	x12	x13	x14	x15
Integer Register ABI Name	s0	s1	a0	a1	a2	a3	a4	a5
Floating-Point Register Number	f8	f9	f10	f11	f12	f13	f14	f15
Floating-Point Register ABI Name	fs0	fs1	fa0	fa1	fa2	fa3	fa4	fa5

Top & testbench





Fetch


Don't touch
clk
rst
control_en_i
control_pc_i
inst_i
inst_en_o
current_pc_o
inst_addr_o
inst_o
CTR32
MUX

Restriction

•All data is represented in 2's complement form.

•The data memory (RAM) is word-addressable, i.e., each address refers to a 32-bit memory space.

•You should **reset your system** before carrying out any instructions. In other words, during the first cycle, all pins should output **0**.

•To reduce your workload, all test scripts will only involve ten integer registers **x0, x2, x8-x15**.

•All instructions in testcases are valid, and you don't need to consider instruction checking (please note that **NOP** is also a valid instruction in RVC and our CPU).

•Since Logisim supports memory with limited size, only the lower 8 bits of an 32-bit address are used to access instruction memory (ROM) and lower 16 bits to access data memory (RAM).