Lecture 6: RISC-V Datapath II

CS10014 Computer Organization

Department of Computer Science
Tsung Tai Yeh
Thursday: 1:20 pm– 3:10 pm
Classroom: EC-022

Acknowledgements and Disclaimer

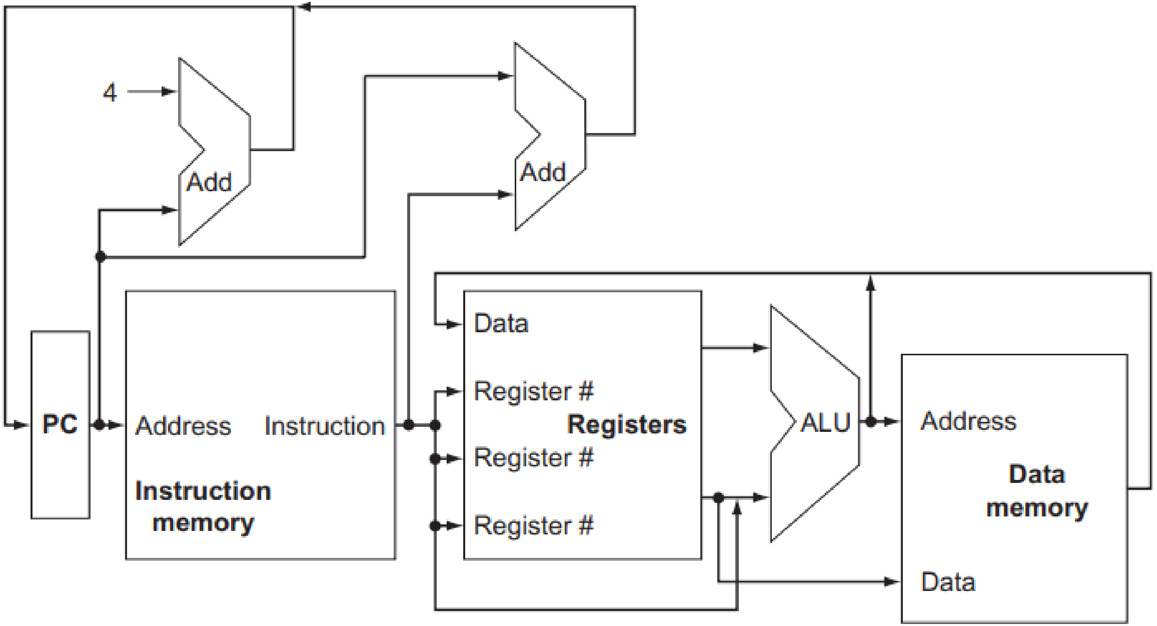
- Slides were developed in the reference with
    - CS 61C at UC Berkeley
        - https://inst.eecs.berkeley.edu/~cs61c/sp23/
    - CS 252 at UC Berkeley
        - https://people.eecs.berkeley.edu/~culler/courses/cs252-s05/
    - CSCE 513 at University of South Carolina
        - https://passlab.github.io/CSCE513/

Outline

- Processor Datapath
- Immediate Generator
- Load Instruction
- Store Instruction
- Conditional Branch
- Unconditional Jump
- U-Format Instruction

Processor Datapath (1/3)

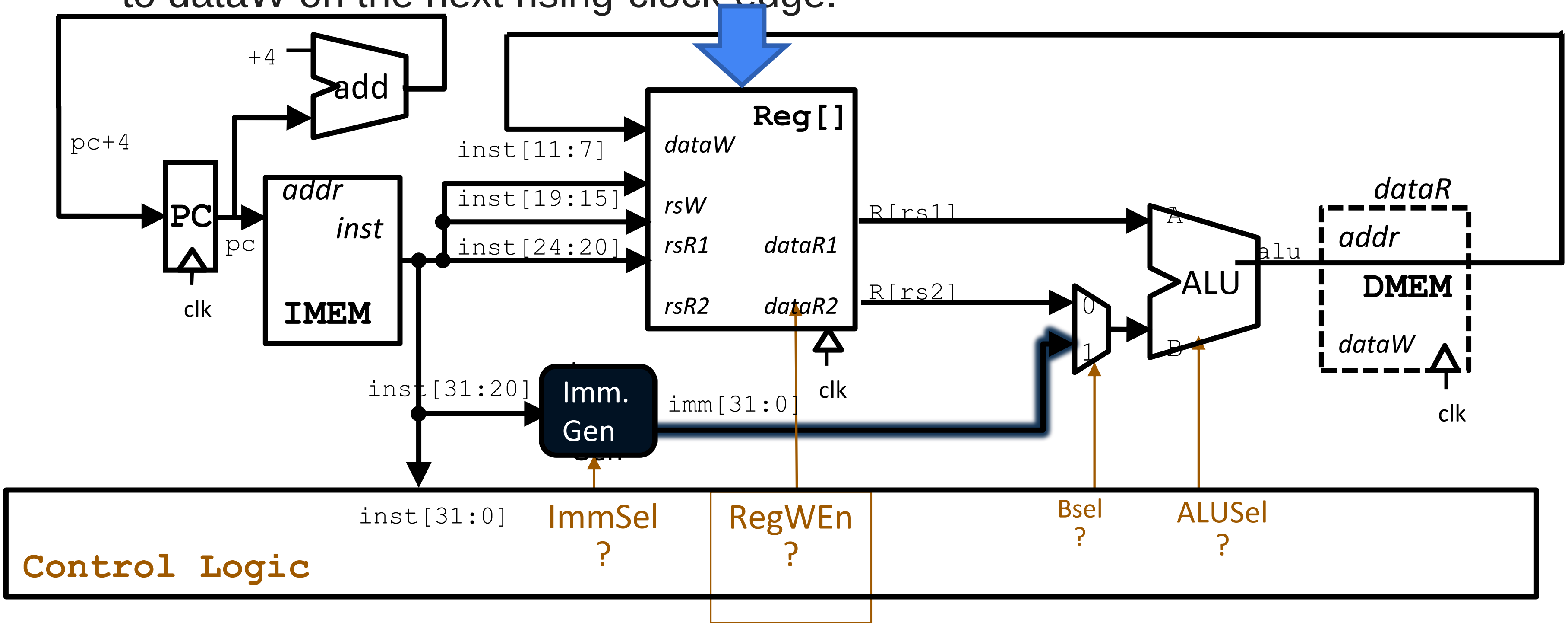
- **Datapath**
  - Elements/wires that process data and addresses in the CPU
    - Registers,
      ALU,
      MUX,
      Memories



Processor Datapath (2/3)

Edge-triggered write.
If RegWEn=1, RegFile updated with input to dataW on the next rising-clock edge.

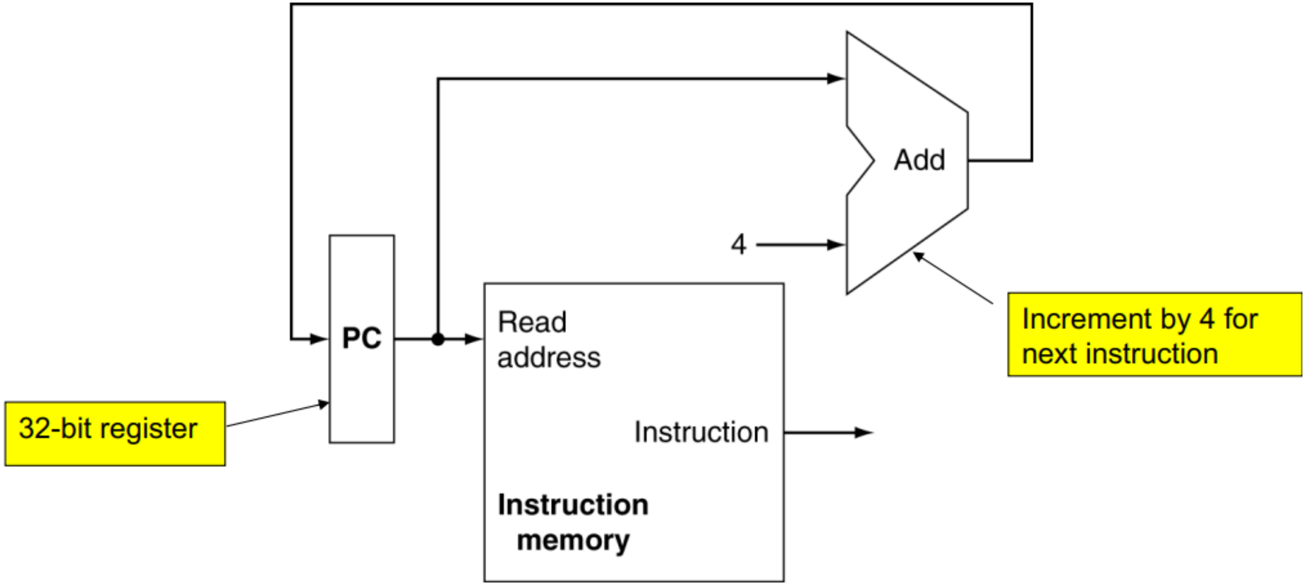
Bold black wires carry data.
Thin orange wires are control logic.



Processor Datapath (3/3)

- **Instruction Fetch**

```
0x0FFE1230:  add x6,   x12, x13
0x0FFE1234:  lw   x6,   24(x12)
0x0FFE1238:  sw   x13, 24(x12)
0x0FFE123C: beq  x12, x13, offset
```



32-bit register

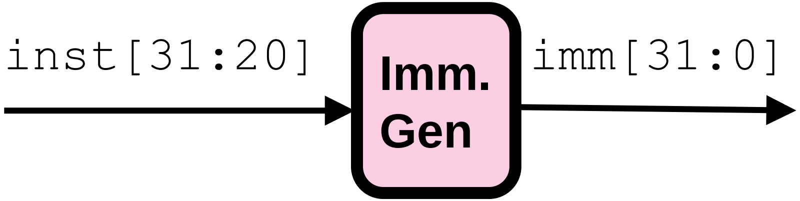
Increment by 4 for next instruction

Outline

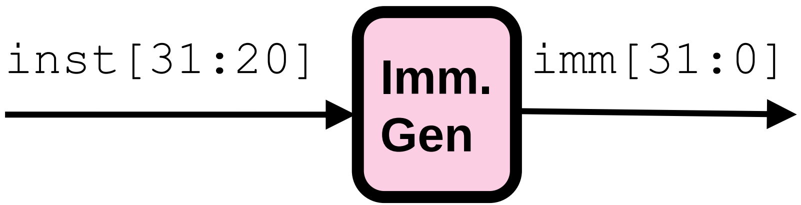
- Processor Datapath
- Immediate Generator
- Load Instruction
- Store Instruction
- Conditional Branch
- Unconditional Jump
- U-Format Instruction

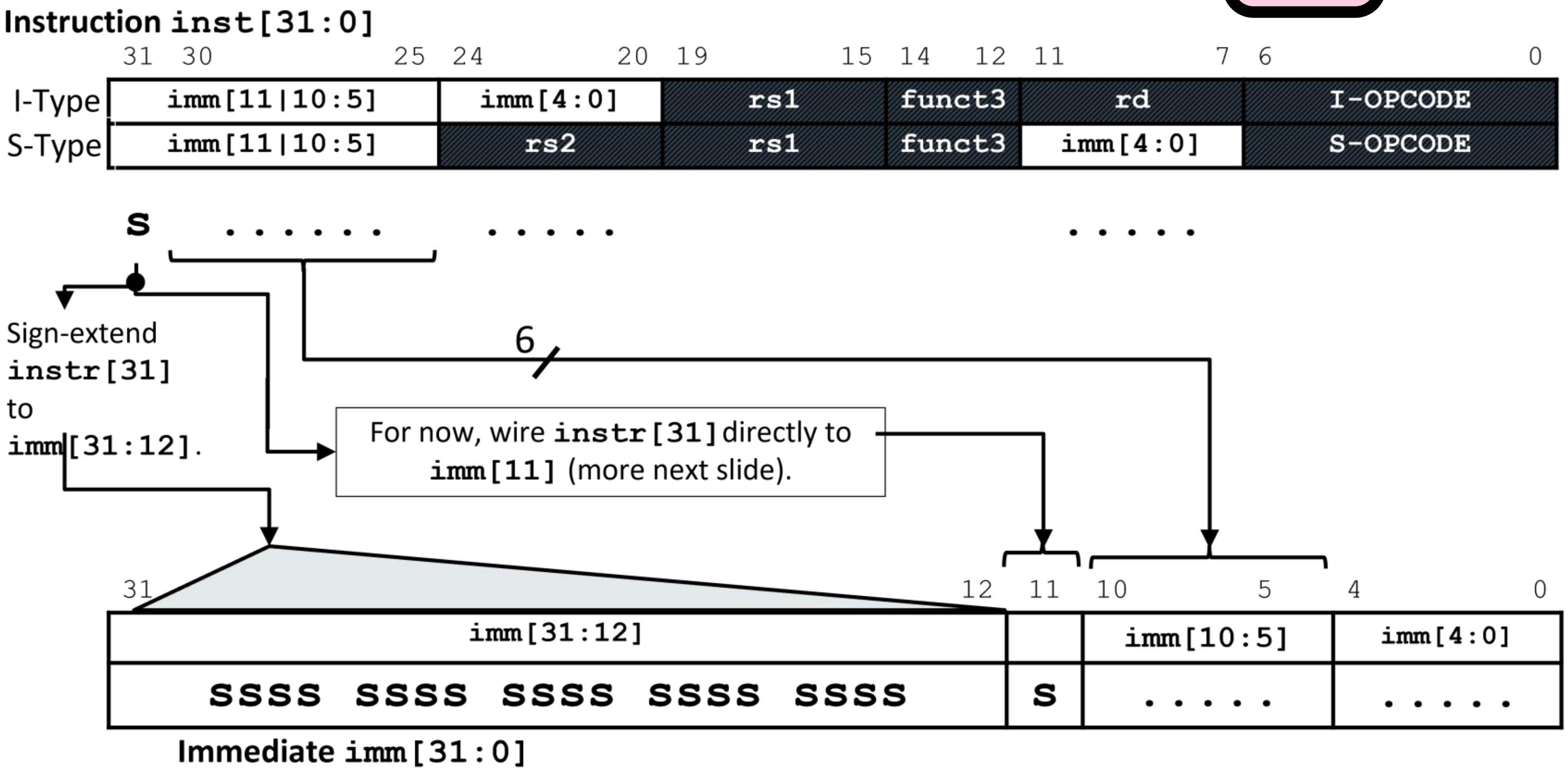
Immediate Gen.(1/4)

- **Immediate Generator**
  - Generate 32 or 64-bit immediate value (depending on whether we design 32-bit or 64-bit machine) from an instruction word
  - Select the 12-bit from the instruction word and sign-extended to 32 or 64-bit
  - Used for I, S, and SB-format (I-format ALU, load, store, and beq)



Immediate Gen.(2/4)





Instruction inst[31:0]

	31	30 ... 25	24 ... 20	19 ... 15	14 ... 12	11 ... 7	6 ... 0
I-Type	imm[11|10:5]		imm[4:0]	rs1	funct3	rd	I-OPCODE
S-Type	imm[11|10:5]		rs2	rs1	funct3	imm[4:0]	S-OPCODE

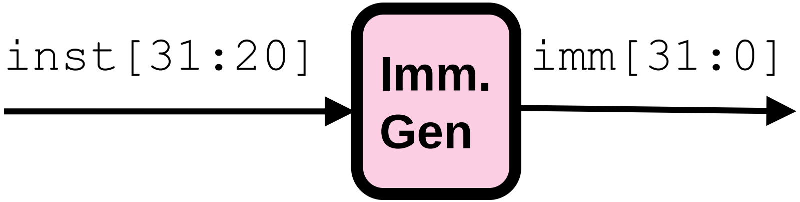
Sign-extend **instr[31]** to **imm[31:12]**.

For now, wire **instr[31]** directly to **imm[11]** (more next slide).

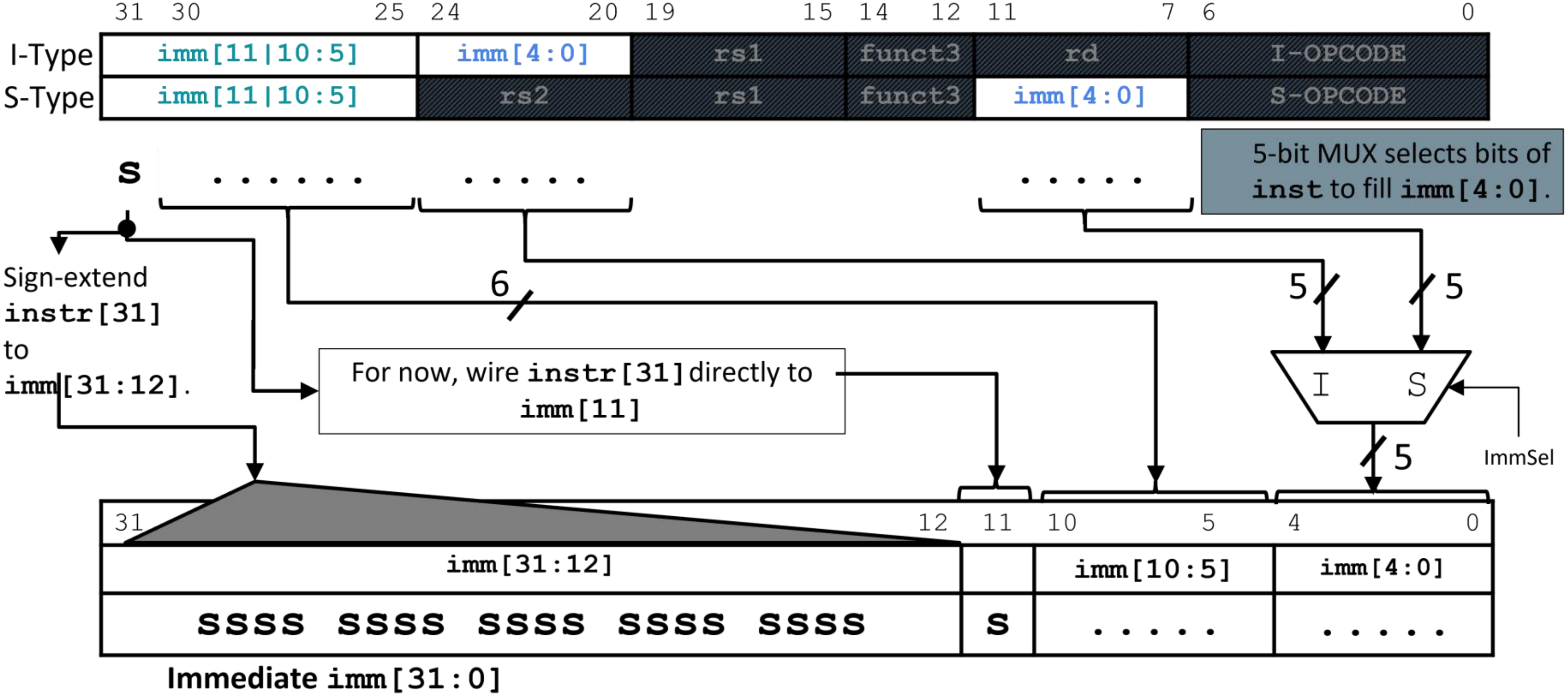
31 ... 12	11	10 ... 5	4 ... 0
imm[31:12]		imm[10:5]	imm[4:0]
SSSS SSSS SSSS SSSS SSSS	S	......	......

Immediate imm[31:0]

Immediate Gen.(3/4)



Instruction inst[31:0]



	31	30 25	24 20	19 15	14 12	11 7	6 0
I-Type	imm[11|10:5]		imm[4:0]	rs1	funct3	rd	I-OPCODE
S-Type	imm[11|10:5]		rs2	rs1	funct3	imm[4:0]	S-OPCODE

5-bit MUX selects bits of **inst** to fill **imm[4:0]**.

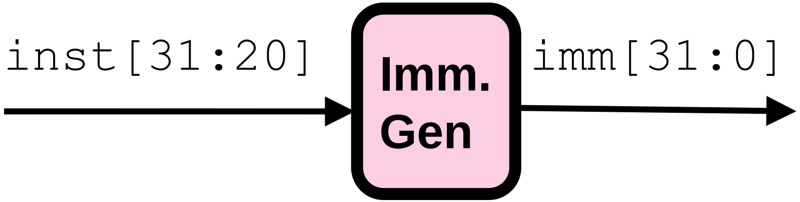
Sign-extend **instr[31]** to **imm[31:12]**.

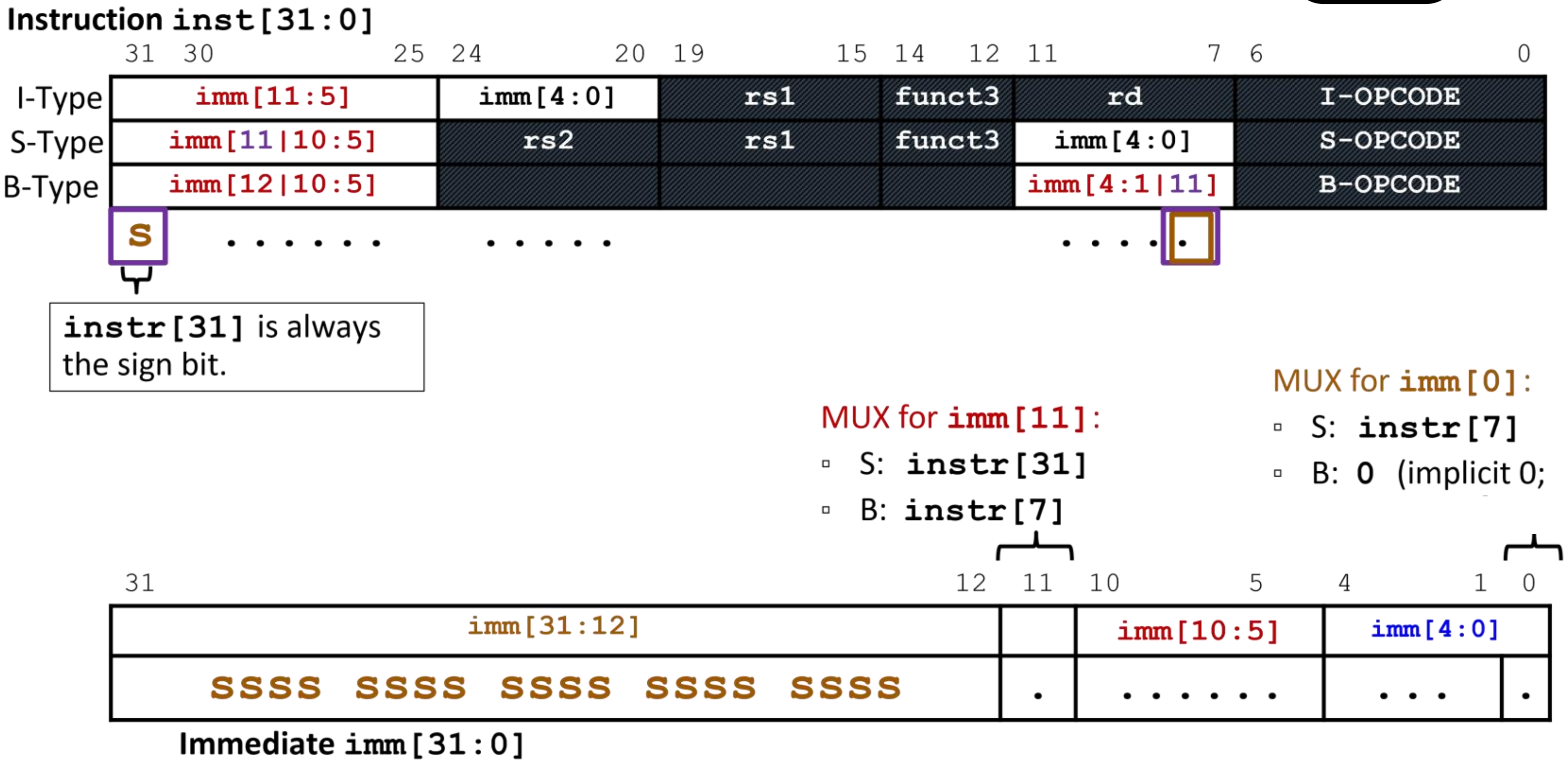
For now, wire **instr[31]** directly to **imm[11]**

31 12	11	10 5	4 0
imm[31:12]		imm[10:5]	imm[4:0]
SSSS SSSS SSSS SSSS SSSS	S	. . . . .	. . . . .

Immediate imm[31:0]

Immediate Gen.(4/4)





Instruction inst[31:0]

	31	30 … 25	24 … 20	19 … 15	14 … 12	11 … 7	6 … 0
I-Type	imm[11:5]		imm[4:0]	rs1	funct3	rd	I-OPCODE
S-Type	imm[11|10:5]		rs2	rs1	funct3	imm[4:0]	S-OPCODE
B-Type	imm[12|10:5]					imm[4:1|11]	B-OPCODE
	S	......	.....			....| .	

instr[31] is always the sign bit.

MUX for **imm[11]**:

- S: **instr[31]**
- B: **instr[7]**

MUX for **imm[0]**:

- S: **instr[7]**
- B: **0** (implicit 0;

31 … 12	11	10 … 5	4 … 1	0
imm[31:12]		imm[10:5]	imm[4:0]	
SSSS SSSS SSSS SSSS SSSS	.	......	...	.

Immediate imm[31:0]

Outline

- Processor Datapath
- Immediate Generator
- Load Instruction
- Store Instruction
- Conditional Branch
- Unconditional Jump
- U-Format Instruction

Load Instruction (1/4)

addr = (Base register **rs1**)
+ (sign-extended **imm** offset)

- **I-format: lw   x14, 8(x2)**

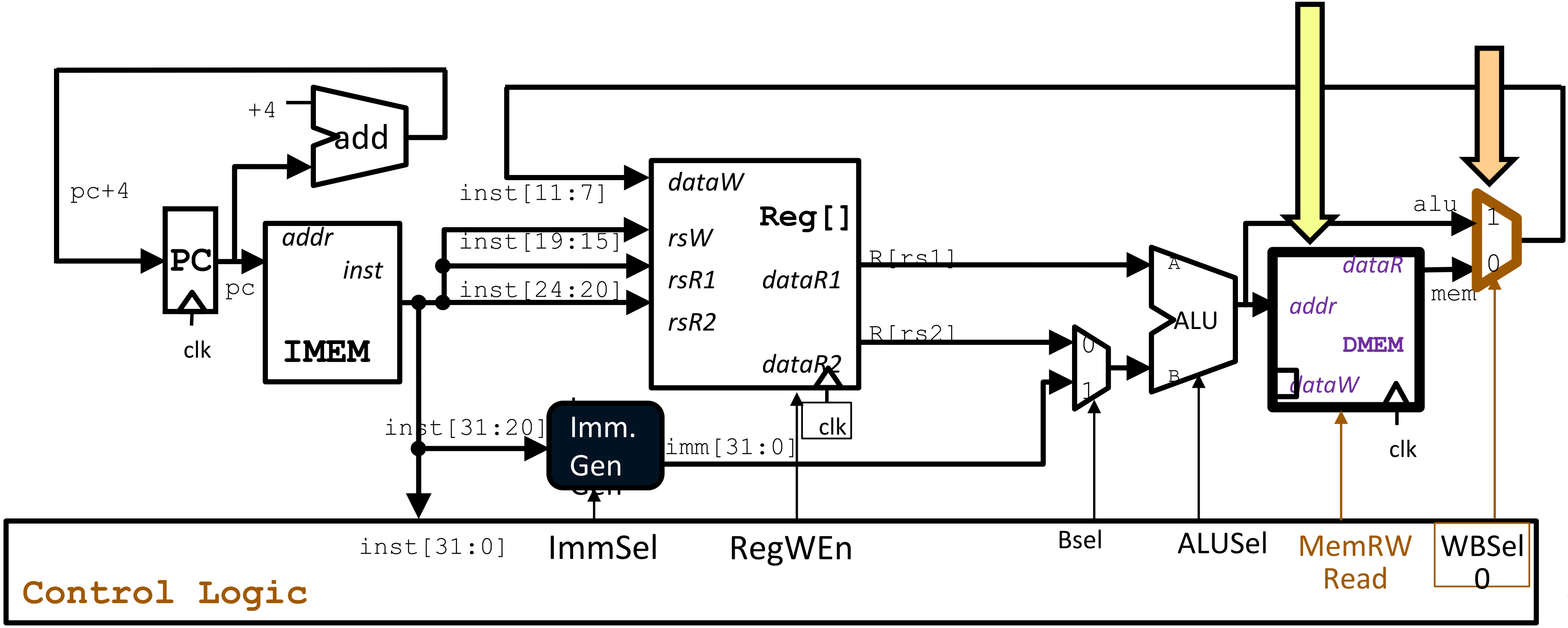
31 20	19 15	14 12	11 7	6 0
imm[11:0]	**rs1**	**funct3**	**rd**	**opcode**
000000001000	**00010**	**010**	**01110**	**0000011**
12	5	3	5	7
		“load word”		**LOAD**

- Creates an address
- State element access includes a memory read!
  - DMEM   (read word at address addr)
  - RegFile   Reg[rs1] # read; Reg[rd] # write
  - PC   PC = PC + 4

Load Instruction (2/4)

If load instruction, save **mem**.
Otherwise, save **alu**.

Read memory at address
alu = R[rs1] + imm.



Load Instruction (3/4)

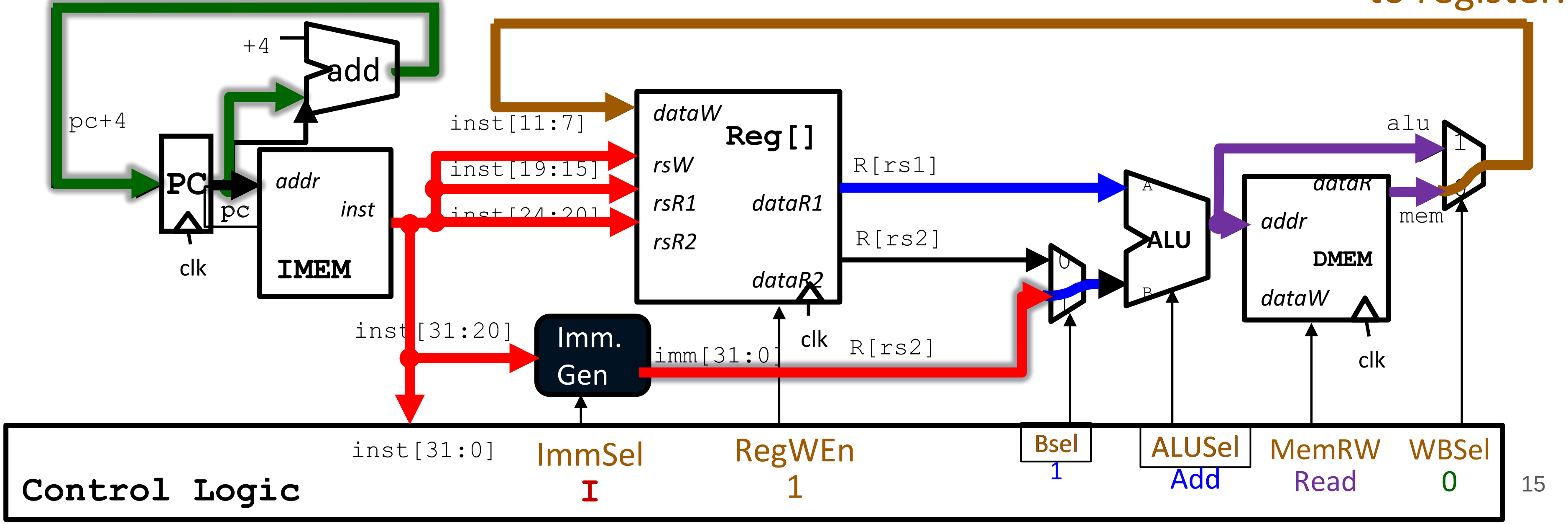
Increment PC to next instruction.

Immediate Generation Block builds a 32-bit immediate **imm**.

ALU computes address **alu = R[rs1] + imm**.

Read memory at address **alu**.

Write loaded memory value **mem** to register.



Load Instruction (4/4)

- **RV32I can load different width**
  - To support narrow loads (lb, lh, lbu, lhu)
  - Load 32-bit word from memory
  - Add additional logic to extract correct byte or halfword
  - Sign- or zero-extend result to 32 bits to write into RegFile

		funct3		opcode	
imm[11:0]	rs1	000	rd	0000011	lb
imm[11:0]	rs1	001	rd	0000011	lh
imm[11:0]	rs1	010	rd	0000011	lw
imm[11:0]	rs1	100	rd	0000011	lbu
imm[11:0]	rs1	101	rd	0000011	lhu

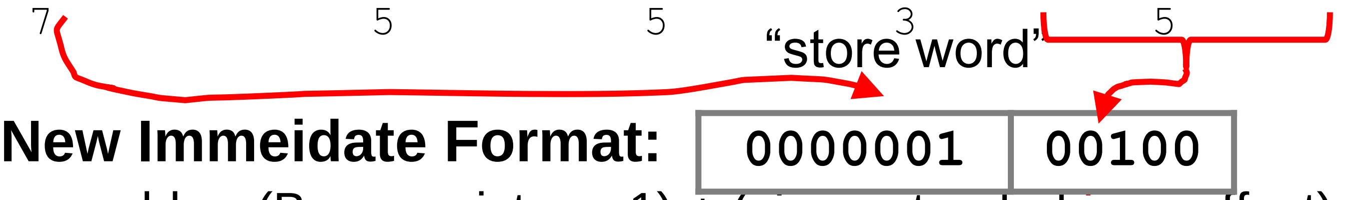
Outline

- Processor Datapath
- Immediate Generator
- Load Instruction
- Store Instruction
- Conditional Branch
- Unconditional Jump
- U-Format Instruction

Store Instruction (1/4)

- **S-format: sw   x14, 36(x2)**

31 25	24 20	19 15	14 12	11 7	6 0
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
0000001	01110	00010	010	00100	0100011
7	5	5	3	5	7
			“store word”		**STORE**



- **New Immeidate Format:**

0000001	00100

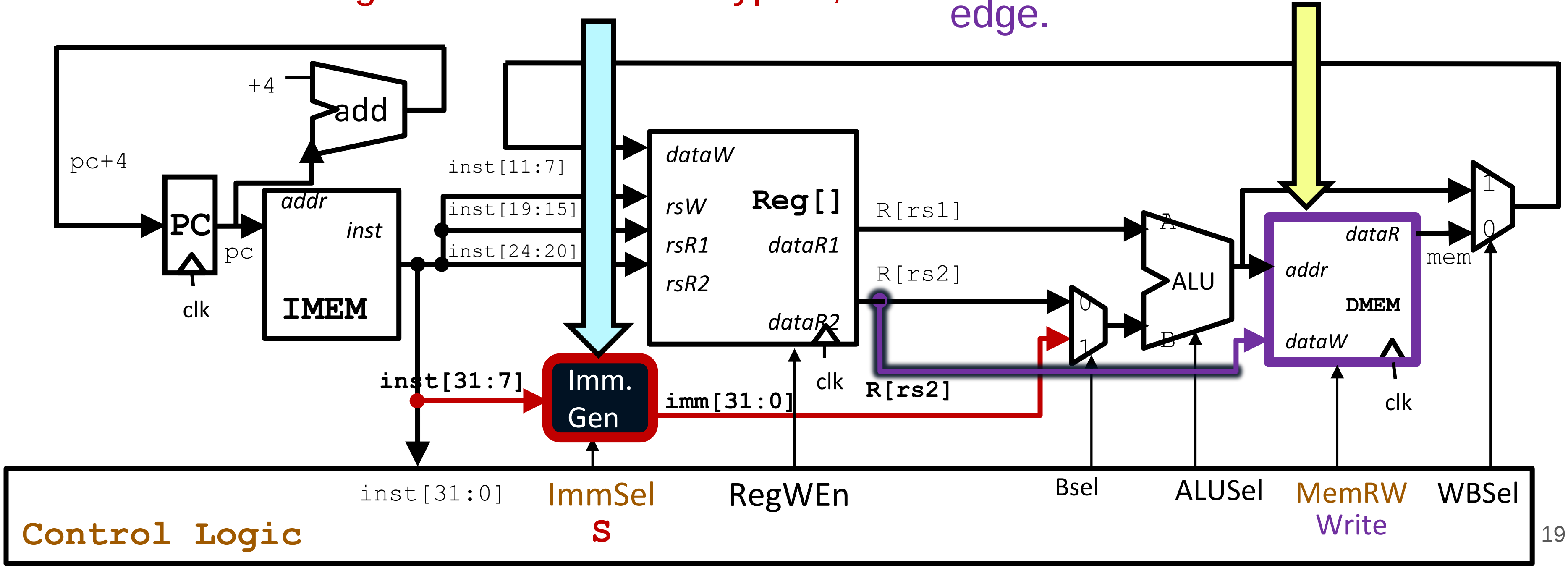
  - addr = (Base register rs1) + (sign-extended imm offset)
- **State Elements Accessed**
  - DMEM   (write R[rs2] to word at address addr)
  - RegFile   R[rs1] (base address), R[rs2] (value to store)
  - PC   PC = PC + 4

Store Instruction (2/4)

Update Blocks for sw

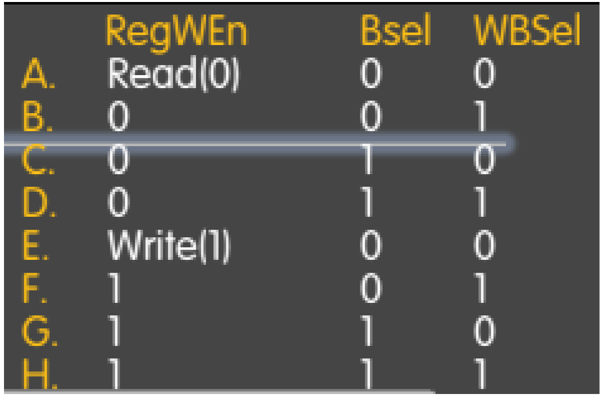
Control ImmSel selects how to generate immediate type: **I**, **S**.

Control MemRW=Write saves **rs2** to memory on the next rising clock edge.

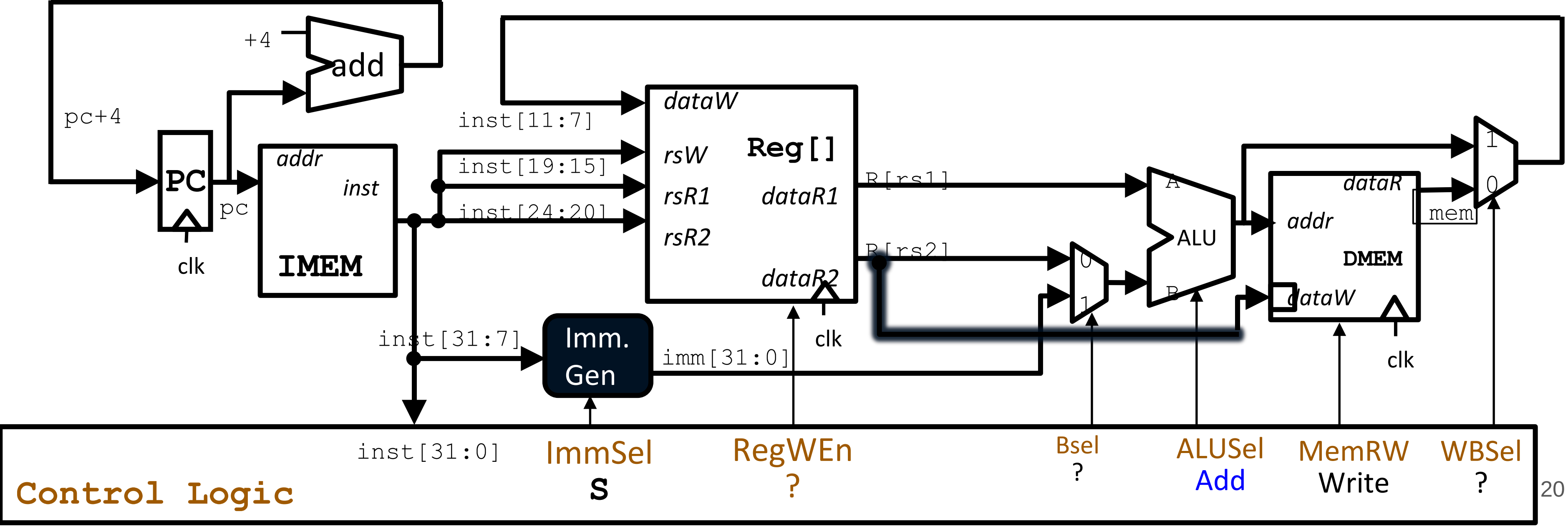


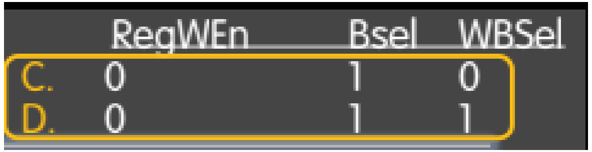
Store Instruction (3/4)

How to set sw control lines?



	RegWEn	Bsel	WBSel
A.	Read(0)	0	0
B.	0	0	1
C.	0	1	0
D.	0	1	1
E.	Write(1)	0	0
F.	1	0	1
G.	1	1	0
H.	1	1	1





	RegWEn	Bsel	WBSel
C.	0	1	0
D.	0	1	1

Store Instruction (4/4)

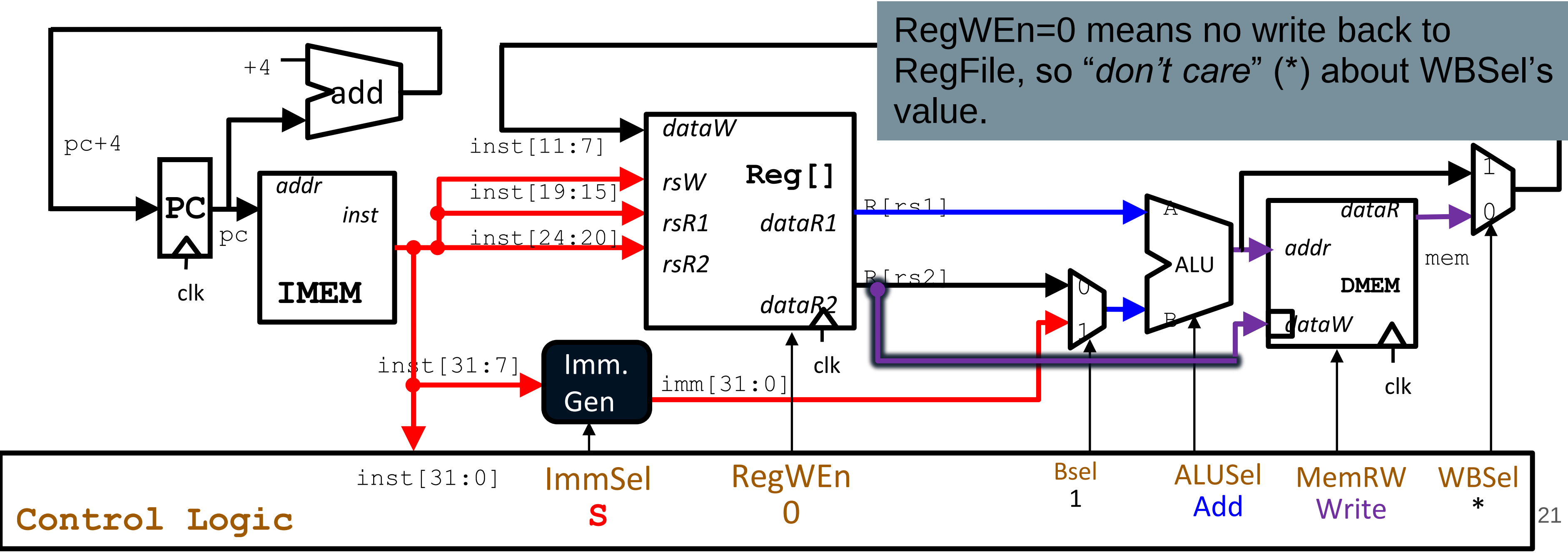
Increment PC to next instruction.

Build **imm** from S-type instruction.

ALU computes address **alu = R[rs1] + imm**.

Write memory at address **`alu`**.

RegWEn=0 means no write back to RegFile, so “*don’t care*” (*) about WBSel’s value.



Outline

- Processor Datapath
- Immediate Generator
- Load Instruction
- Store Instruction
- Conditional Branch
- Unconditional Jump
- U-Format Instruction

Conditional Branch (1/6)

- **SB-format: opname rs1, rs2, Label(simm13)**
  - BEQ/BNE/BLT/BLTU/BGE/BGEU

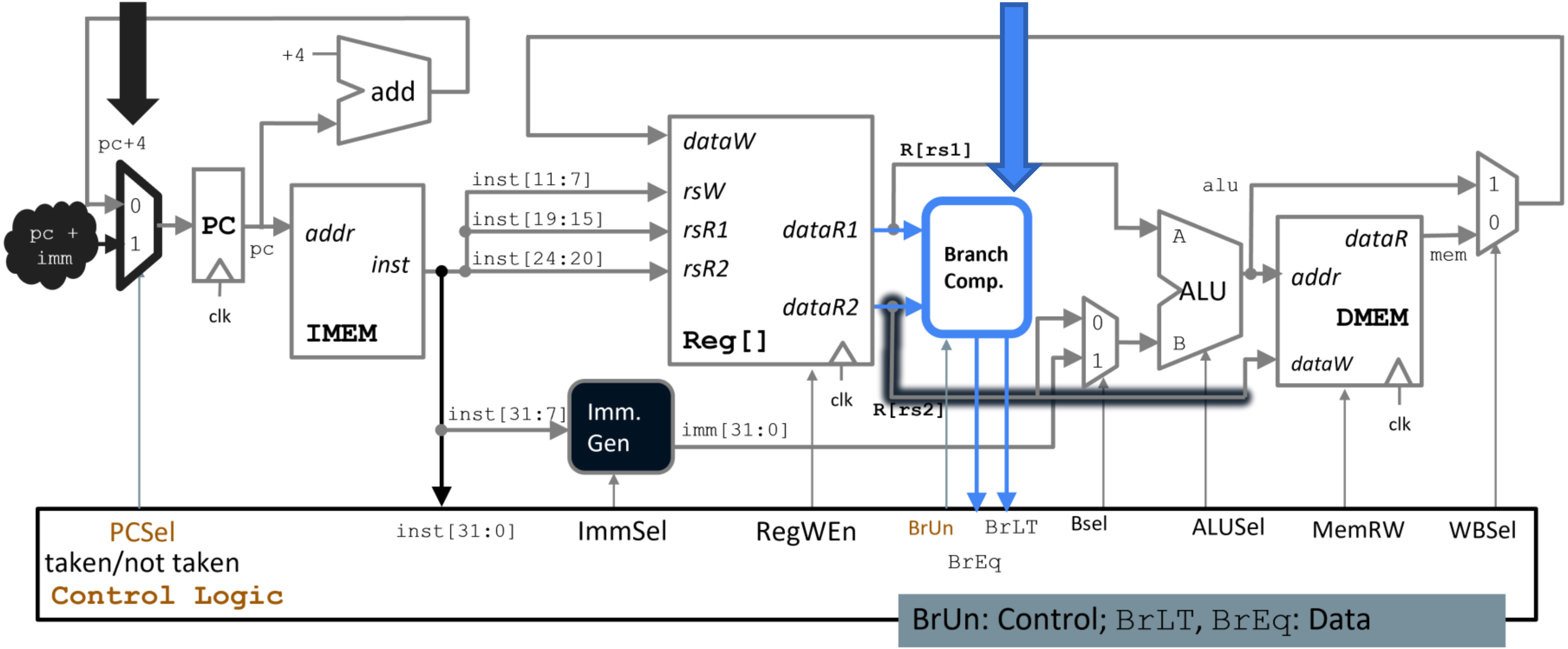
31 25	24 20	19 15	14 12	11 7	6 0
imm[12|10:5]	**rs2**	**rs1**	**funct3**	**imm[4:1|11]**	**opcode**
7	5	5	3	5	7
					BRANCH

- **PC state element now conditionally changes**
  - RegFile R[rs1] , R[rs2] (read only, for branch comparison)
  - PC PC = PC + imm (if branch taken)
    - PC = PC + 4 (otherwise, not taken)

Conditional Branch (2/6)

Update PC based on branch result.

Compare **R[rs1],R[rs2]**, **feed result to Control Logic.**



BrUn: Control; BrLT, BrEq: Data

Conditional Branch (3/6)

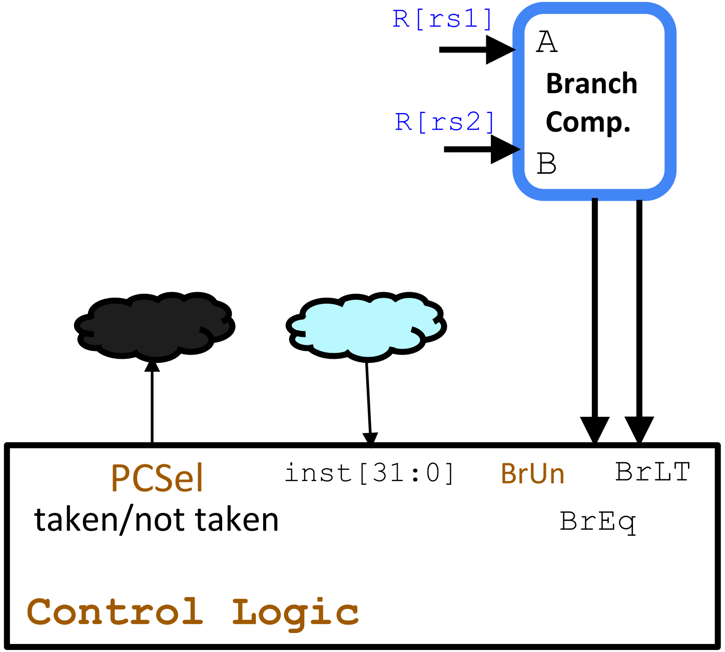
- **The branch comparator is a combinational logic block**

Input:

1. Two data busses A and B (datapath R[rs1] and R[rs2])
2. BrUn (“Branch Unsigned”) control bit

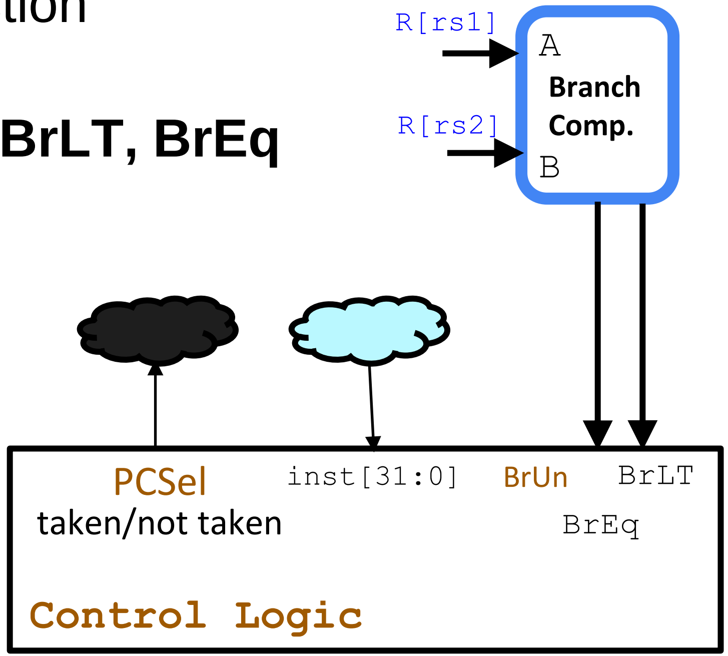
Output:

1. BrEq flag: 1 if A == B
2. BrLT flag: 1 if A < B

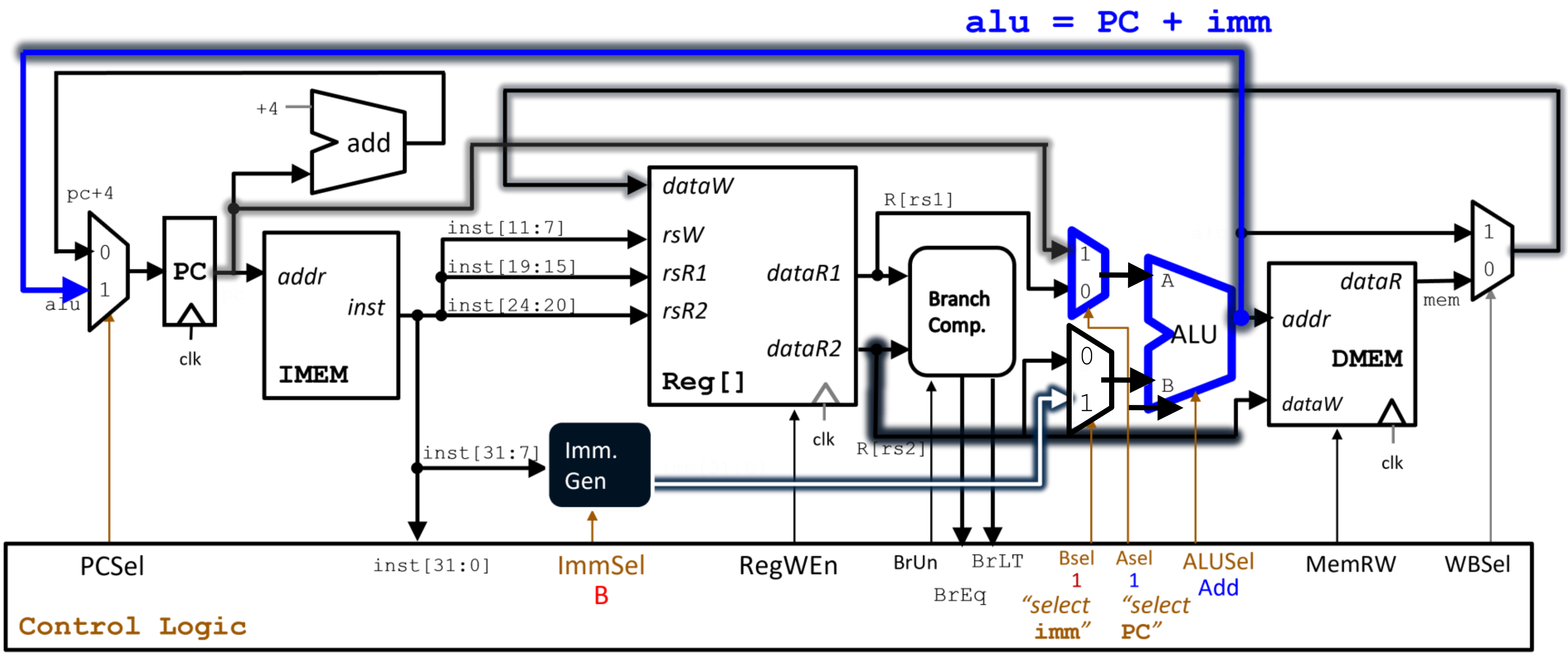


Conditional Branch (4/6)

- **Control Logic**
  - Set BrUn based on current instruction inst[31:0]
  - Set PCSel based on branch flags **BrLT, BrEq**
- **Examples**
  - **blt:**
    - If BrLT=1 and BrEq=0, then PCSel=taken
  - **bge:** $(A \geq B) = \overline{A < B}$
    - If BrLT=0, then PCSel=taken



Conditional Branch (5/6)



Conditional Branch (6/6)

If PCSel=taken, update PC to ALU output. Else, update to next instruction **PC + 4**.

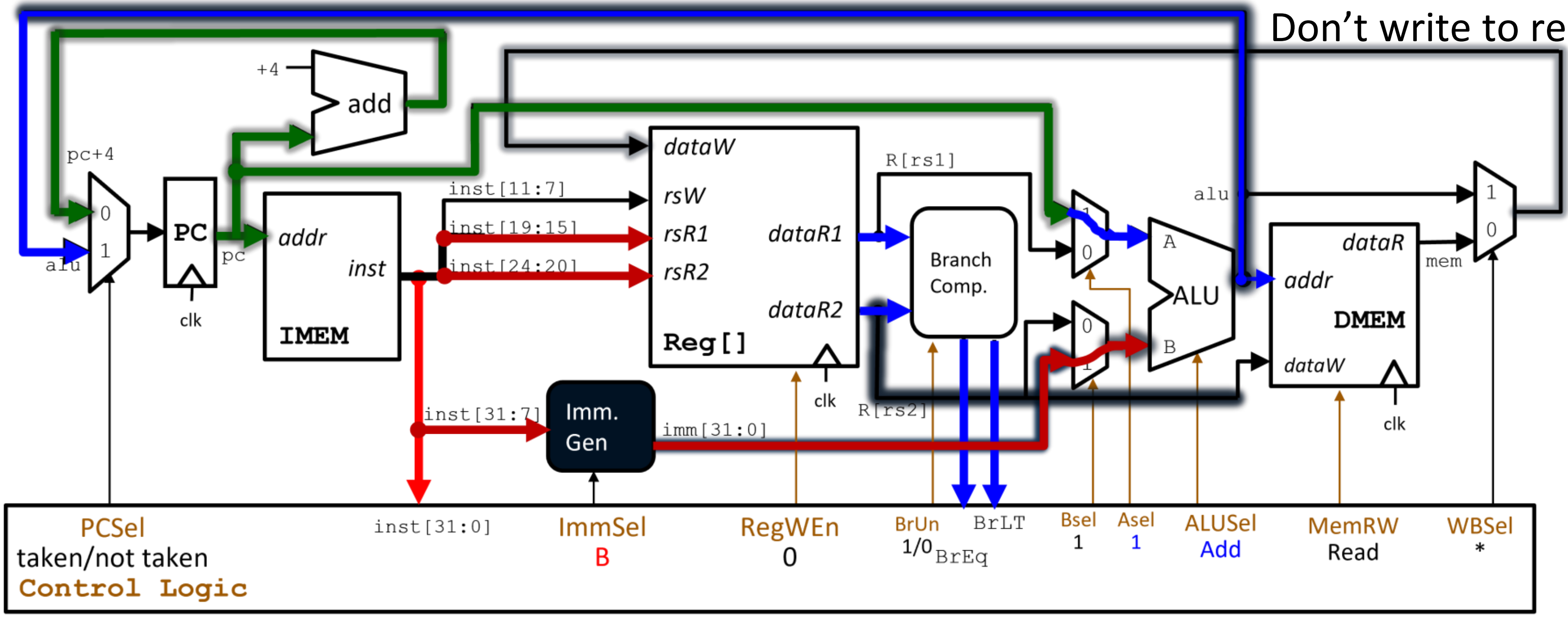
Build **imm** from B-type instruction.

Compute branch; feed to Control.

Compute **PC + imm**.

Don’t write to memory.

Don’t write to registers.



Outline

- Processor Datapath
- Immediate Generator
- Load Instruction
- Store Instruction
- Conditional Branch
- Unconditional Jump
- U-Format Instruction

Unconditional Jump (1/5)

- **UJ-Format: jal rd, Label (**simm21**)**

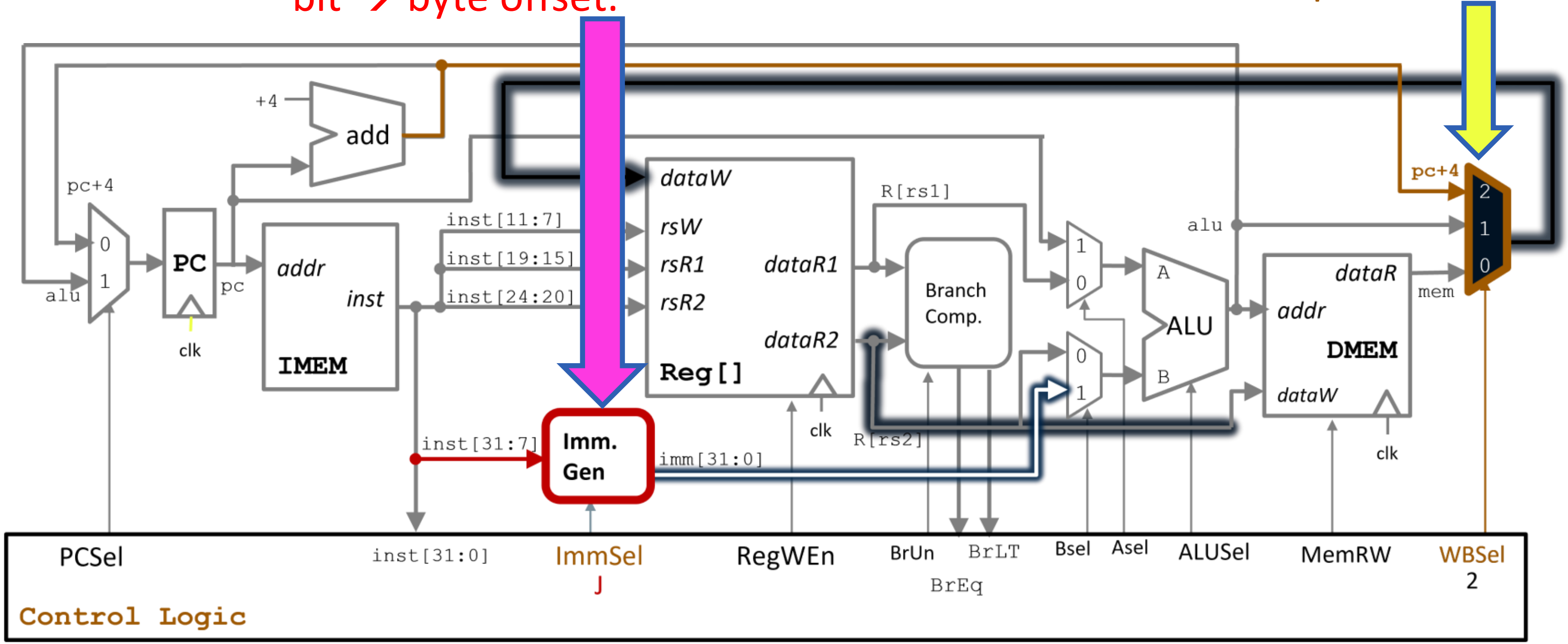
31 25	24 20	19 15 14 12	11 7	6 0
imm[20|10:5]	**imm[4:1,11]**	**imm[19:12]**	**rd**	**opcode**
7	5	8	5	7

- **State Elements updated**
  - PC PC = PC + imm (unconditional PC-relative jump)
  - RegFile rd = PC + 4 (save return address to RegFile destination register)

Unconditional Jump (2/5)

Immediate Generation Block needs to support J-Types: 20-bit → byte offset.

To save **rd = PC + 4**, WBSel now controls a 3-input MUX.



Unconditional Jump (3/5)

Feed PC into blocks.

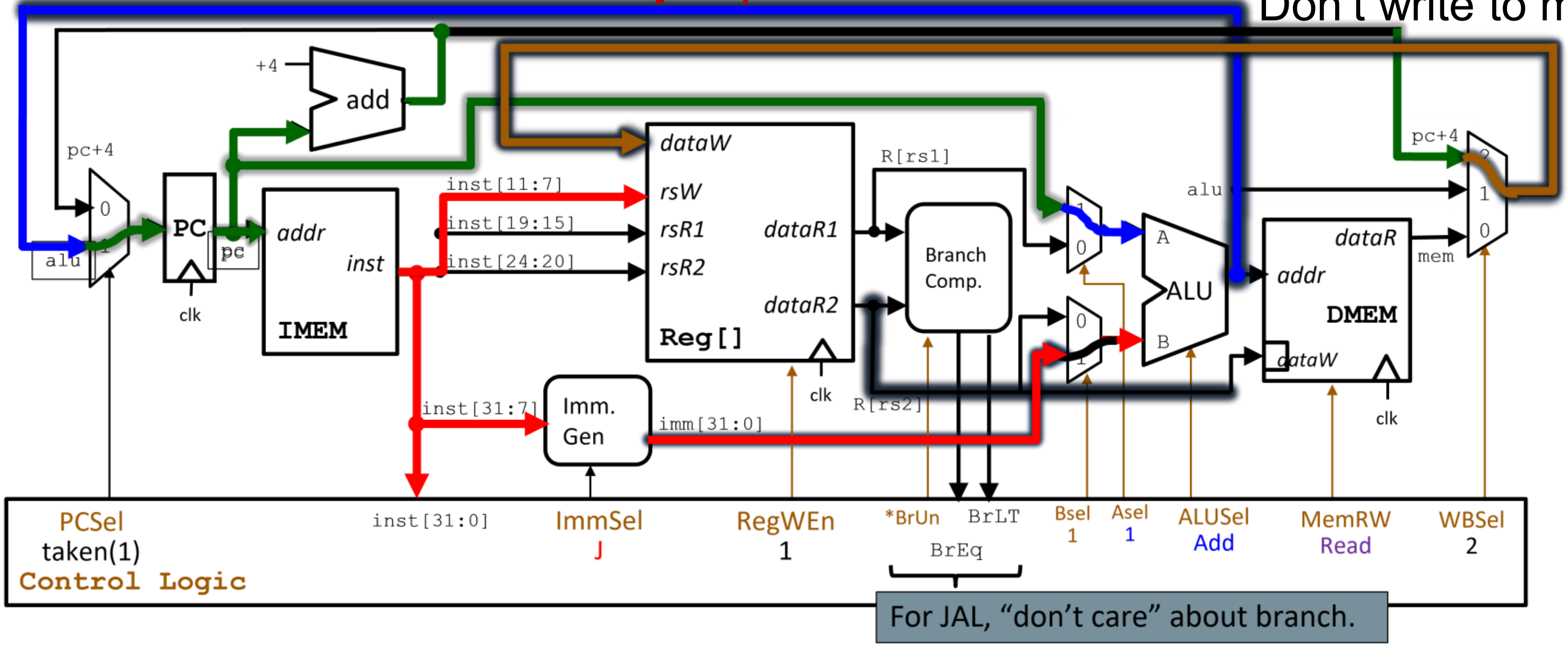
Write ALU output to PC.

Generate byte offset **imm** for 20-bit PC-relative jump.

Compute **PC + imm**.

Write **PC + 4** to destination register.

Don’t write to memory.



For JAL, “don’t care” about branch.

Unconditional Jump (4/5)

- **UJ-Format: jalr rd, rs1, imm (**simm12**)**

31 20	19 15	14 12	11 7	6 0
imm[11:0]	**rs1**	**funct3**	**rd**	**opcode**
12	5	3	5	7
				JALR

- **Two changes to state**
  - PC PC = R[rs1] + imm (absolute addressing)
  - RegFile R[rd] = PC + 4
  - jalr uses the same immediates as arithmetic/loads

Control ImmSel is based on instruction format, not instruction. So far: **I,S,B,J**

Unconditional Jump (5/5)

Feed PC into blocks.

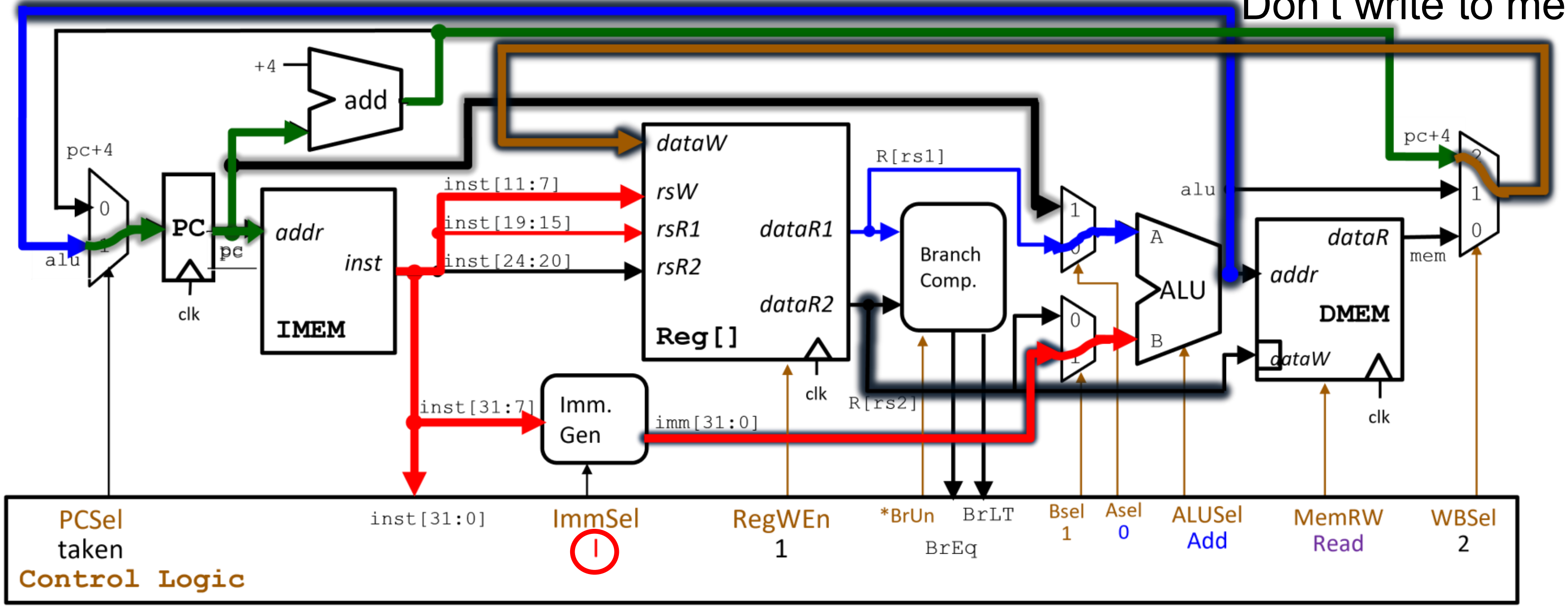
Generate 12-bit **imm** (**I**-Format).

Write ALU output to PC.

Compute **PC + imm**.

Write **PC + 4** to destination register.

Don’t write to memory.

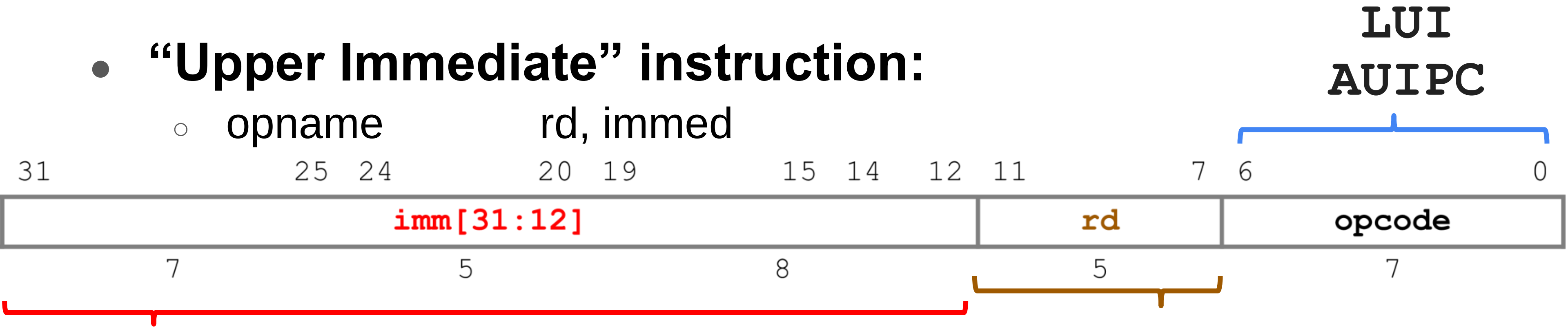


Outline

- Processor Datapath
- Immediate Generator
- Load Instruction
- Store Instruction
- Conditional Branch
- Unconditional Jump
- U-Format Instruction

U-Format Instruction(1/4)

- **“Upper Immediate” instruction:**
  - opname rd, immed



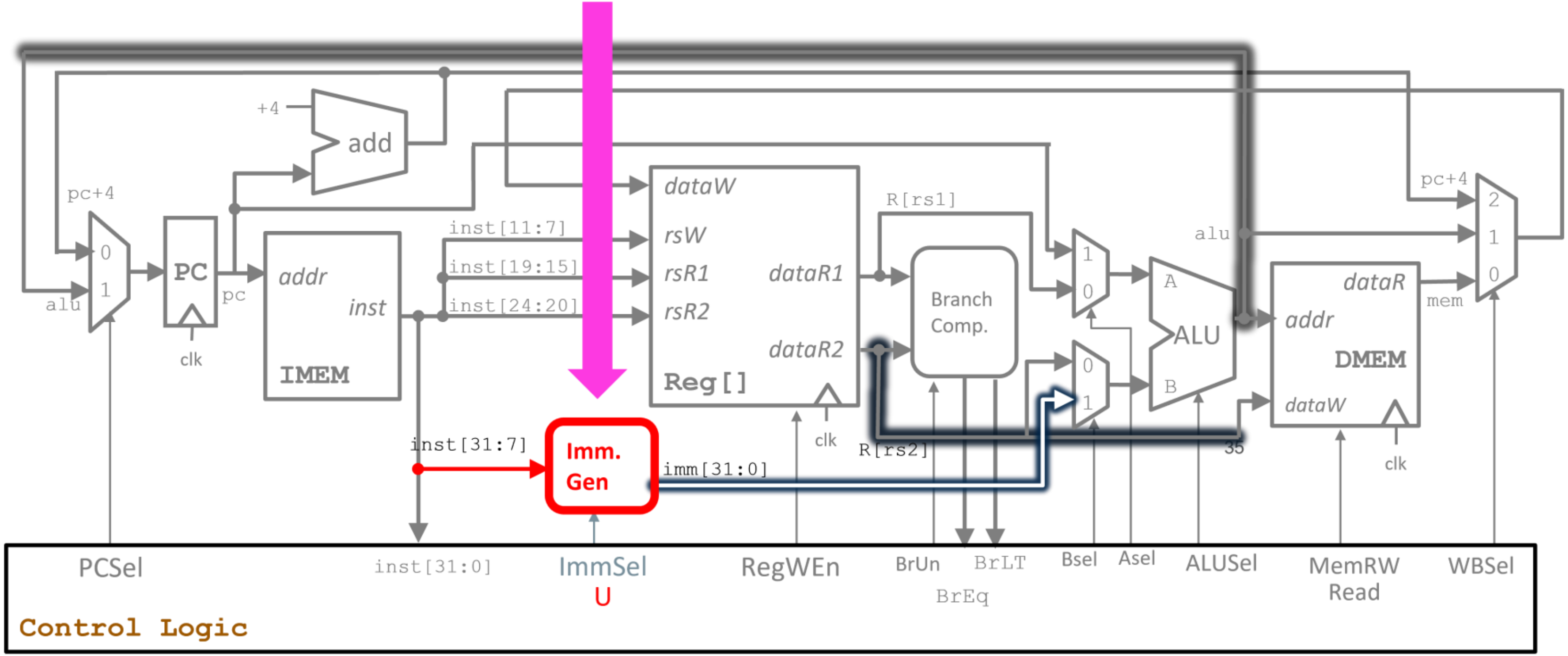
Immediate represents *upper 20 bits* of a 32-bit immediate **imm**.

“Destination” Register

  - **lui**: Load Upper Immediate (lui rd, uimm20)
  - **auipc**: Add Upper Immediate to PC (auipc rd, uimm20)
  - Both increment PC to next instruction and save to destination register

U-Format Instruction(2/4)

Generate **imm** with upper 20 bits. (U-format)



U-Format Instruction(3/4): LUI

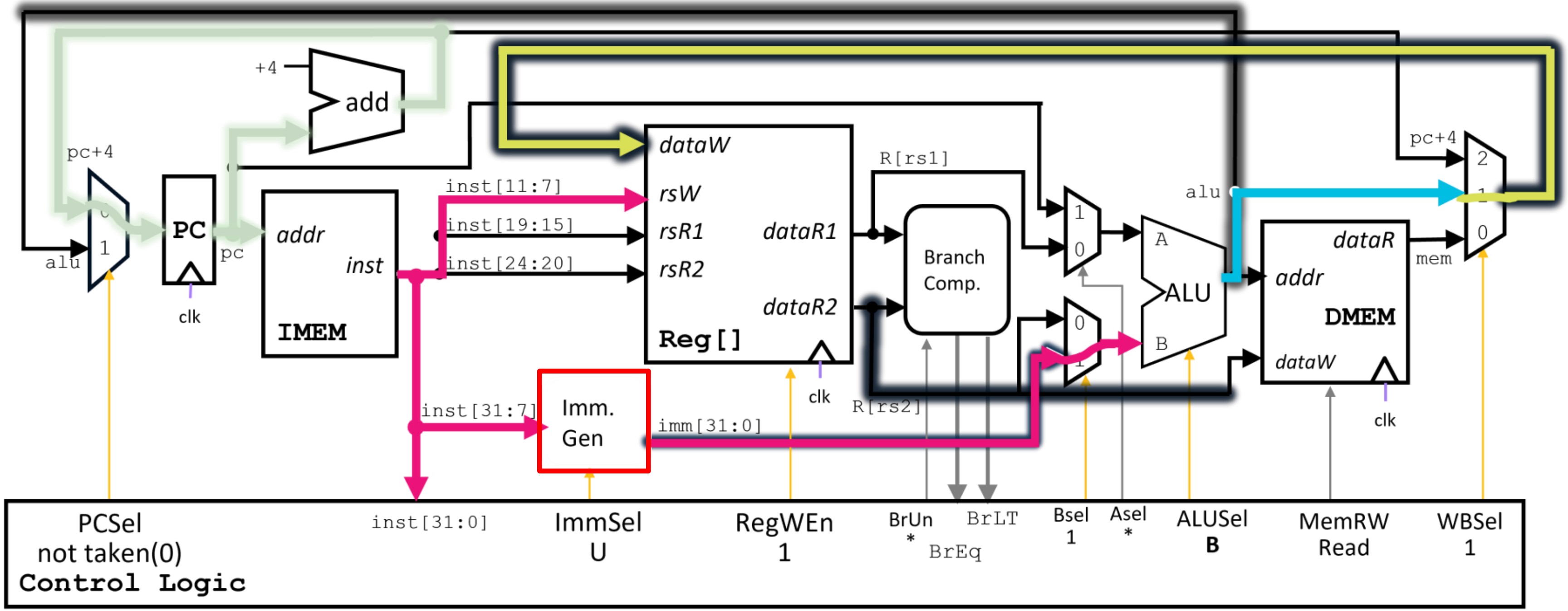
Increment PC to next instruction.

Generate **imm** with upper 20 bits. (U-format)

Grab only imm!
(ALUSel=B)

Write result to destination register.

Don’t write to memory.



U-Format Instruction(4/4): AUIPC

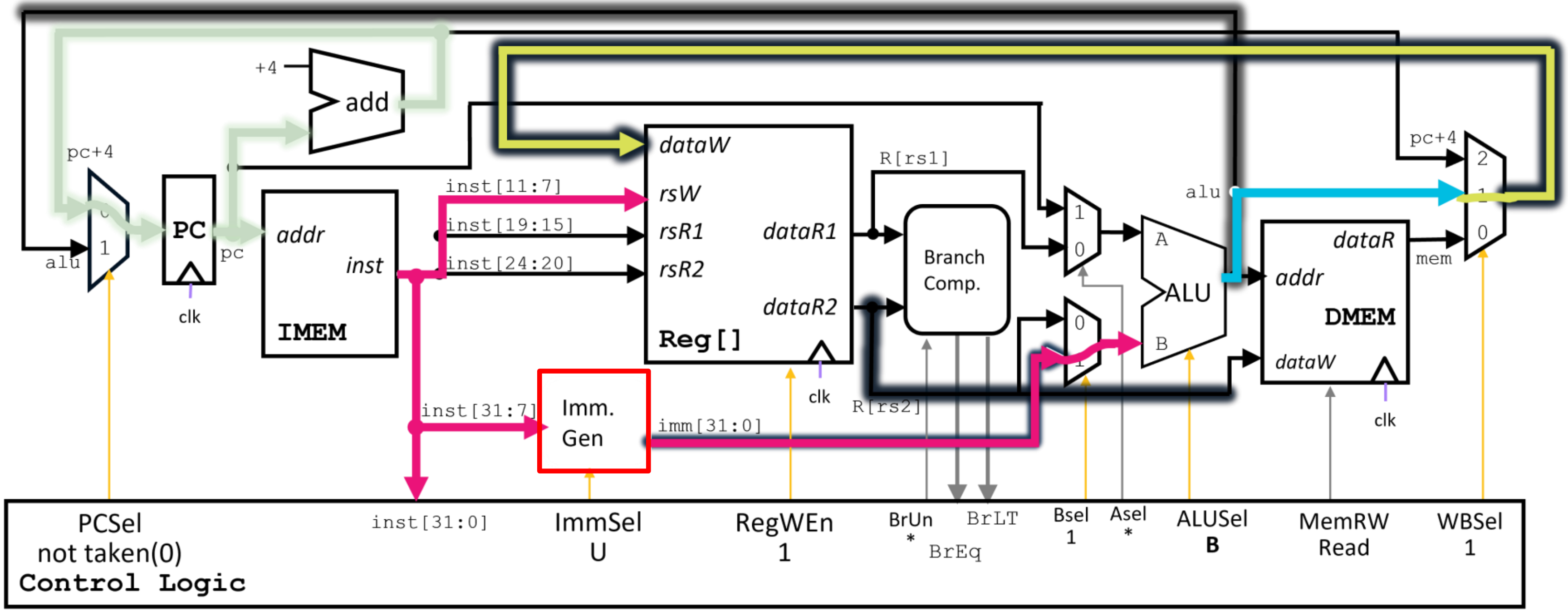
Increment PC to next instruction.

Generate **`imm`** with upper 20 bits. (U-format)

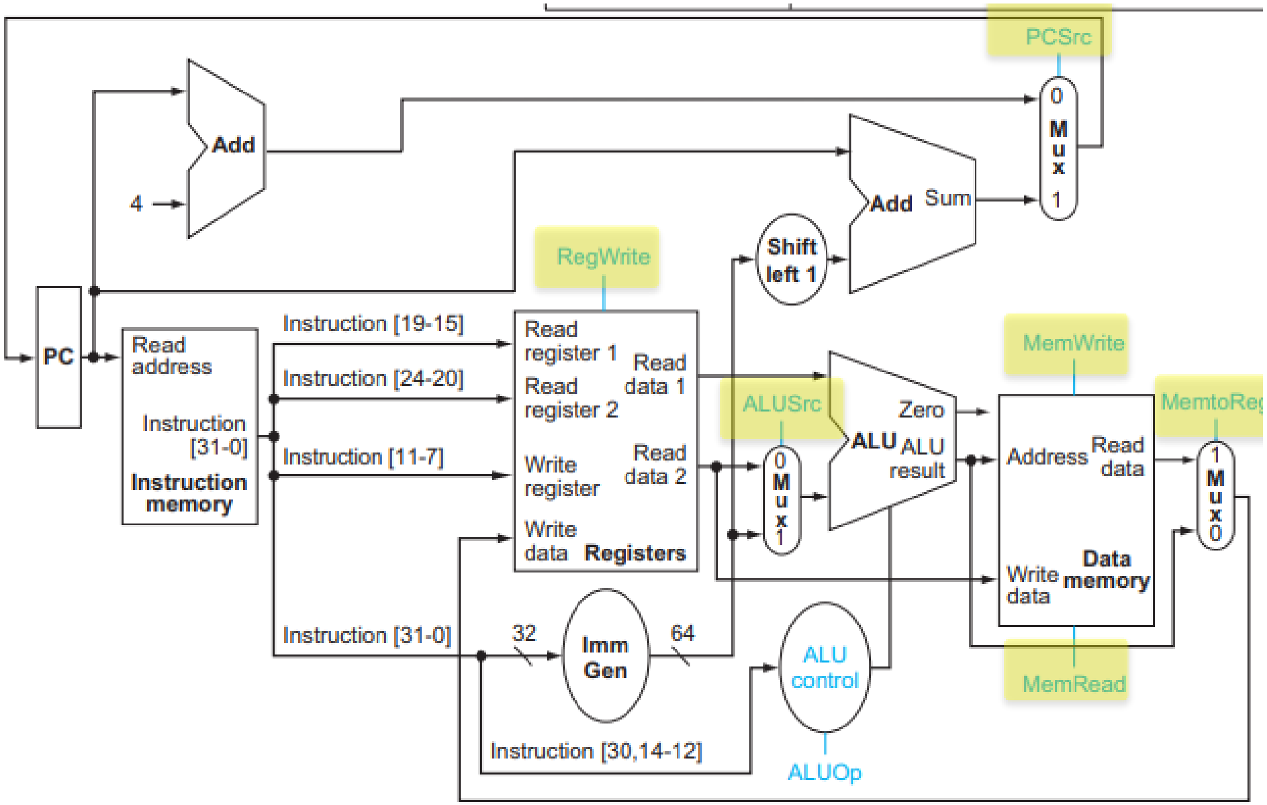
Add `PC + imm`!

Write result to destination register.

Don’t write to memory.

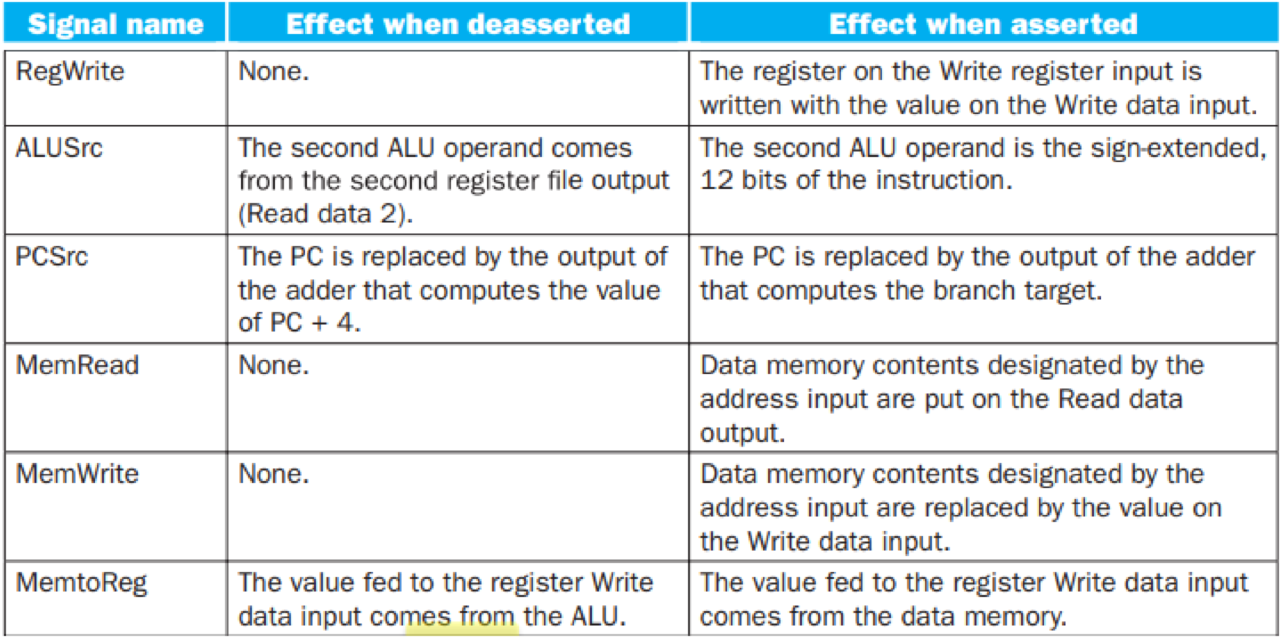


Datapath with Control Signal(1/6)



Datapath with Control Signal(2/6)

- Six Control Signals



Signal name	Effect when deasserted	Effect when asserted
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, 12 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

Datapath with Control Signal(3/6)

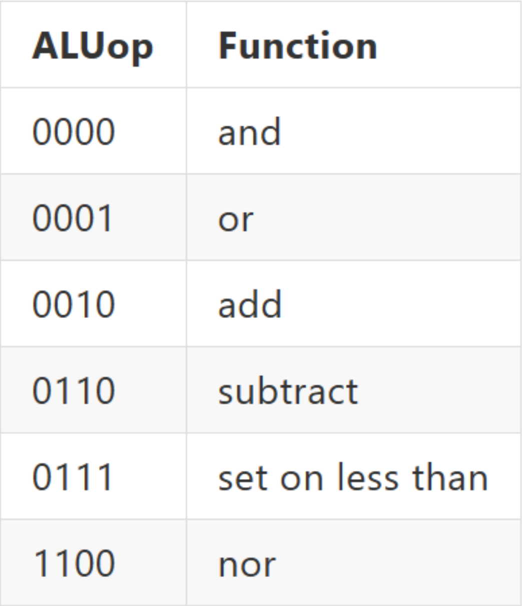
- **Six 1-bit control**
  - **PCSrc:**
    - MUX input to select PC+4 or PC+offset
    - For beq instruction to select next instruction
  - **ALUSrc**
    - MUX input to select input from rs2 or immediate
    - For R/I-type ALU and load instruction
  - **RegWrite**
    - Enable signal to enable write to register
    - For ALU, and load instruction (write to register)

Datapath with Control Signal(4/6)

- **Six 1-bit control**
  - **MemRead:**
    - Enable signal to enable read from memory
    - For load instruction
  - **MemWrite**
    - Enable signal to enable write from memory
    - For store instruction
  - **MemtoReg**
    - MUX input to select input to write to register from memory or ALU
    - For ALU, and load instruction (write to register)

Datapath with Control Signal(5/6)

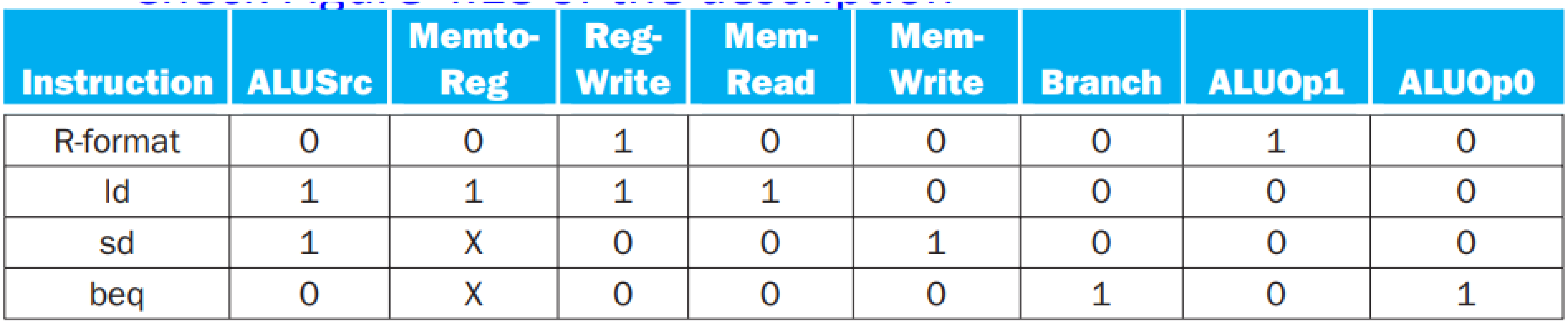
- **One 4-bit control: ALU operation**
  - **2-bit ALUOp (left)**
    - For enabling certain input (A invert or B invert)
  - **2-bit ALUOp (right)**
    - For enabling one MUX control signal
    - 00 is “and”, 01 is “or”, 10 is add, 11 is set on less than



ALUop	Function
0000	and
0001	or
0010	add
0110	subtract
0111	set on less than
1100	nor

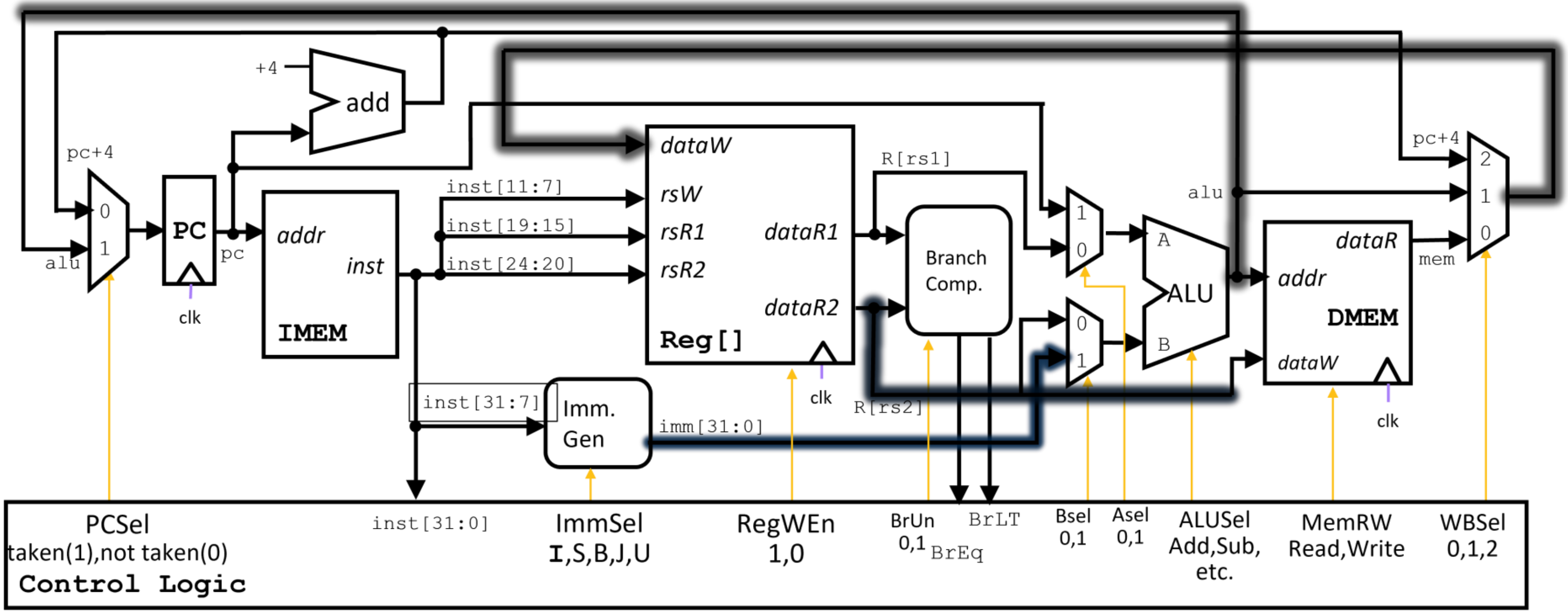
Datapath with Control Signal(6/6)

- **Setting six 1-bit controls**



Instruction	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	0	0	1	0	0	0	1	0
ld	1	1	1	1	0	0	0	0
sd	1	X	0	0	1	0	0	0
beq	0	X	0	0	0	1	0	1

RV32I Datapath


+4
add
pc+4
0
1
alu
PC
clk
pc
addr
inst
IMEM
inst[11:7]
inst[19:15]
inst[24:20]
dataW
rsW
rsR1
rsR2
dataR1
dataR2
Reg[]
clk
R[rs1]
R[rs2]
Branch Comp.
1
0
0
1
A
ALU
B
alu
addr
dataR
DMEM
dataW
clk
mem
pc+4
2
1
0
inst[31:7]
Imm. Gen
imm[31:0]
inst[31:0]
PCSel
taken(1),not taken(0)
Control Logic
ImmSel
I,S,B,J,U
RegWEn
1,0
BrUn
0,1
BrLT
BrEq
Bsel
0,1
Asel
0,1
ALUSel
Add,Sub,
etc.
MemRW
Read,Write
WBSel
0,1,2