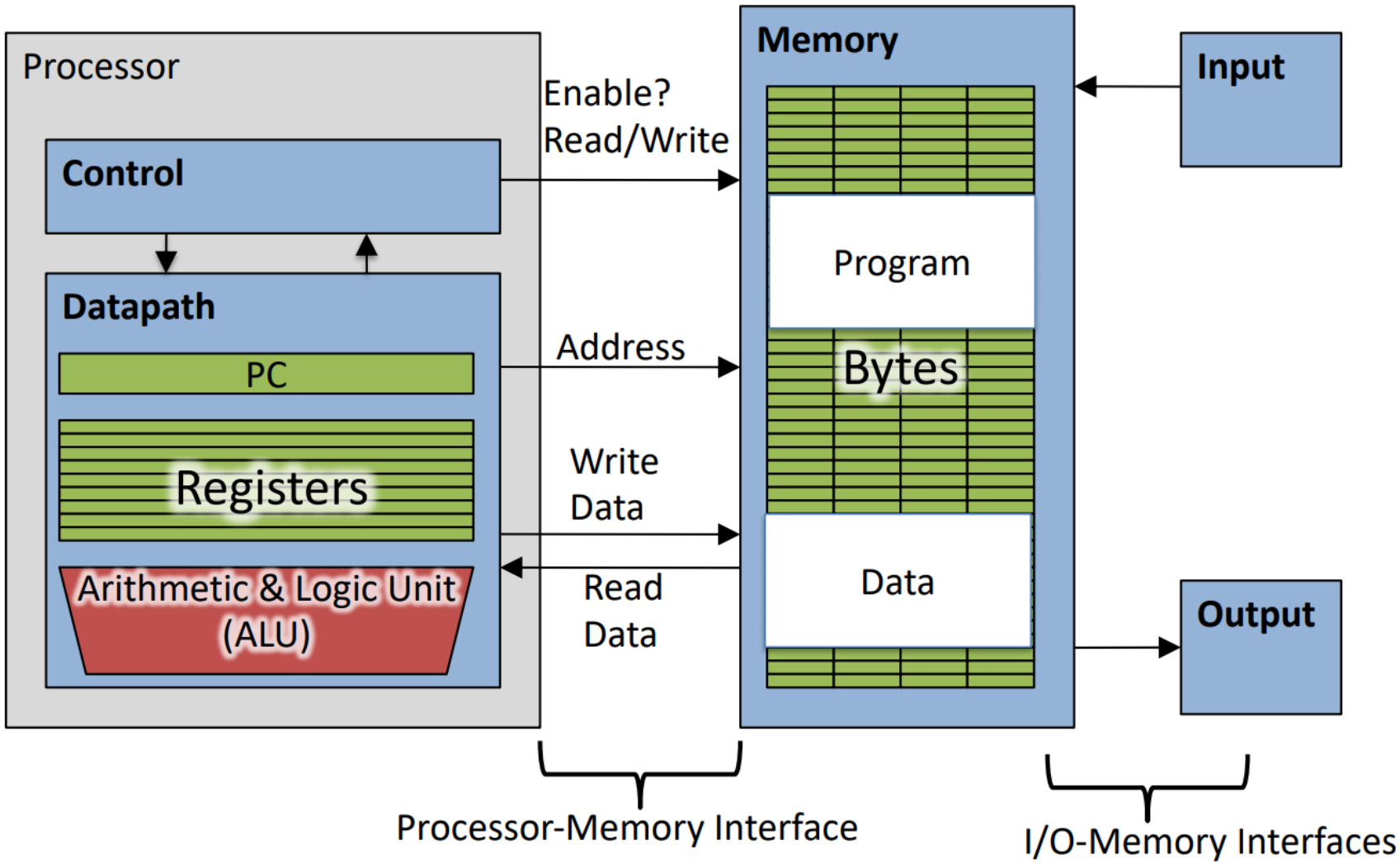
Discussion 8

RISC-V Datapath

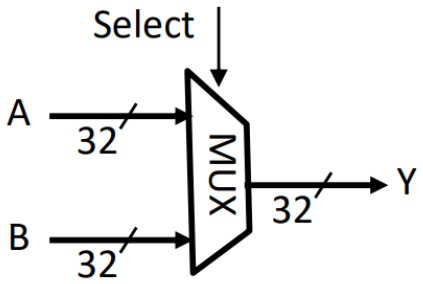
Zhe Ye
yezhe@shanghaitech.edu.cn
Computer Architecture I 2021sp

Components of a Computer



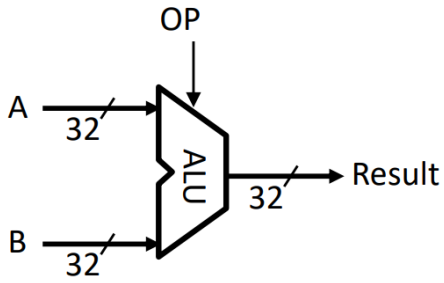
CL Components

Muxer

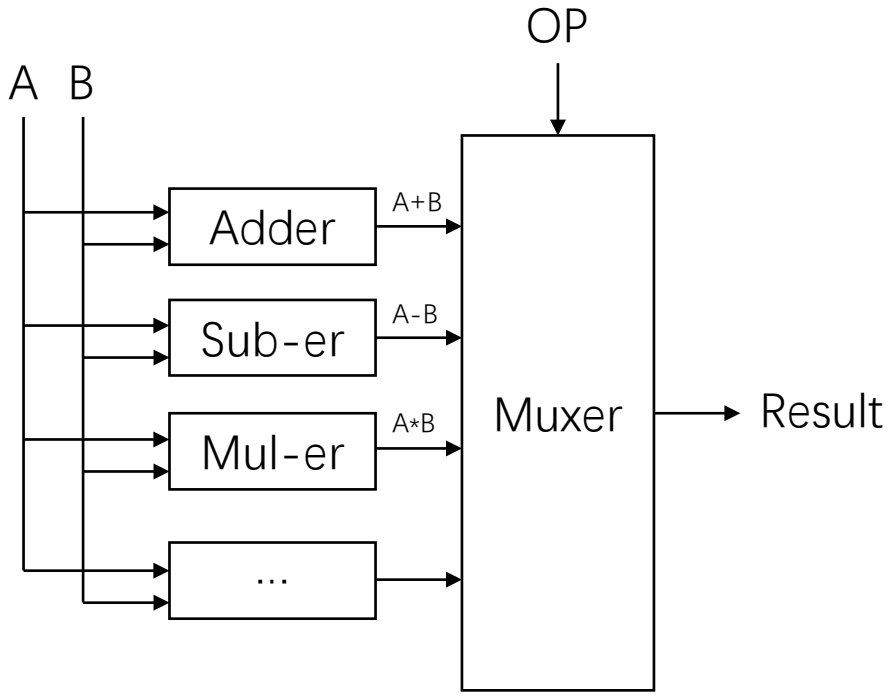


Select	Y
0	A
1	B

ALU



OP	Result
Add	A+B
Sub	A-B
op	op(A,B)



Five Stage of Instruction Execution

Instruction Fetch (IF)
Send address to the instruction memory, and read IMEM at that address.

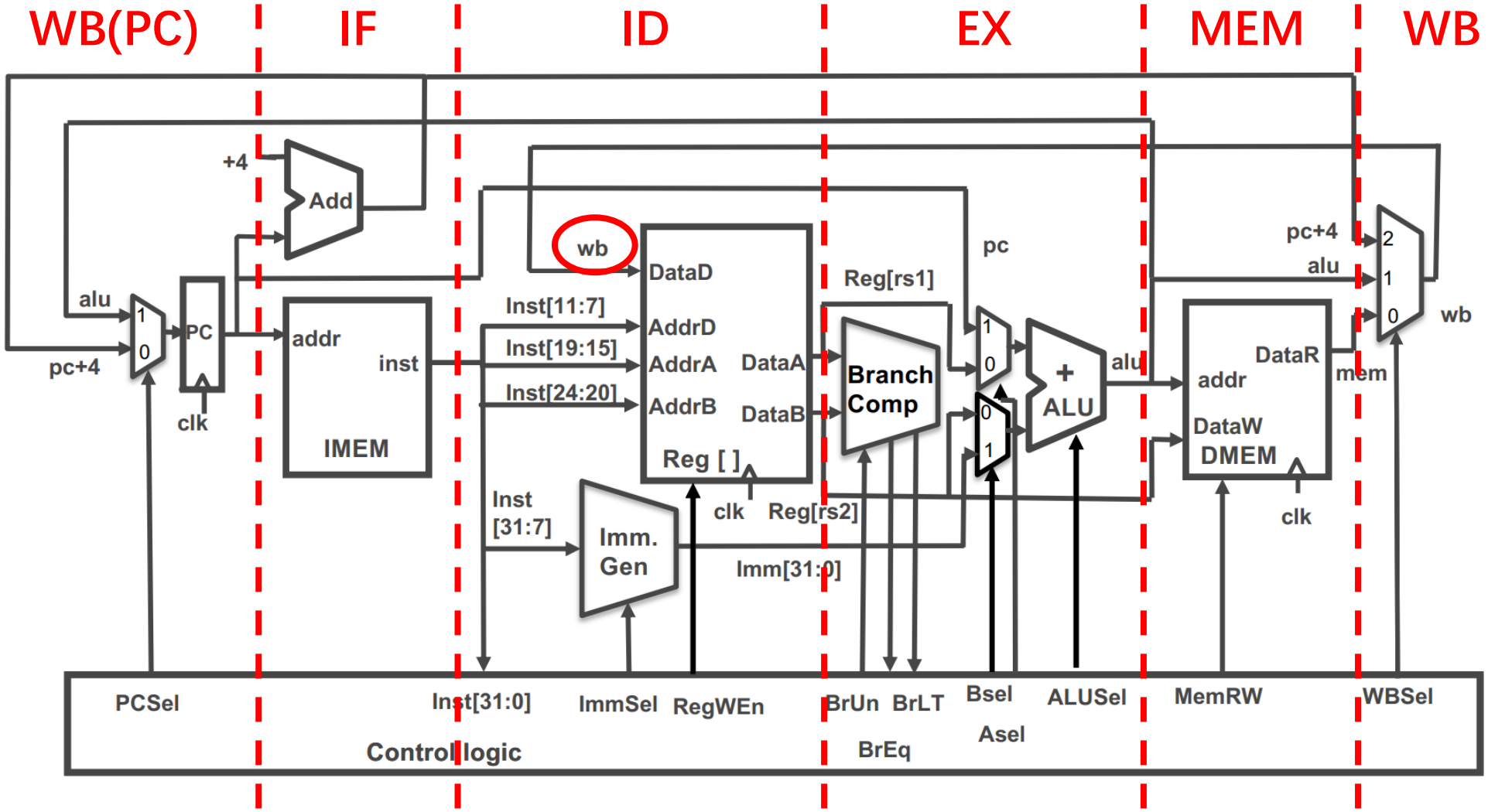
Instruction Decode (ID)
Generate control signal from the instruction bits, generate the immediate, and read registers from the RegFile.

Execute (EX)
Perform ALU operations, and do branch comparison.

Memory (MEM)
Read from or write to the data memory.

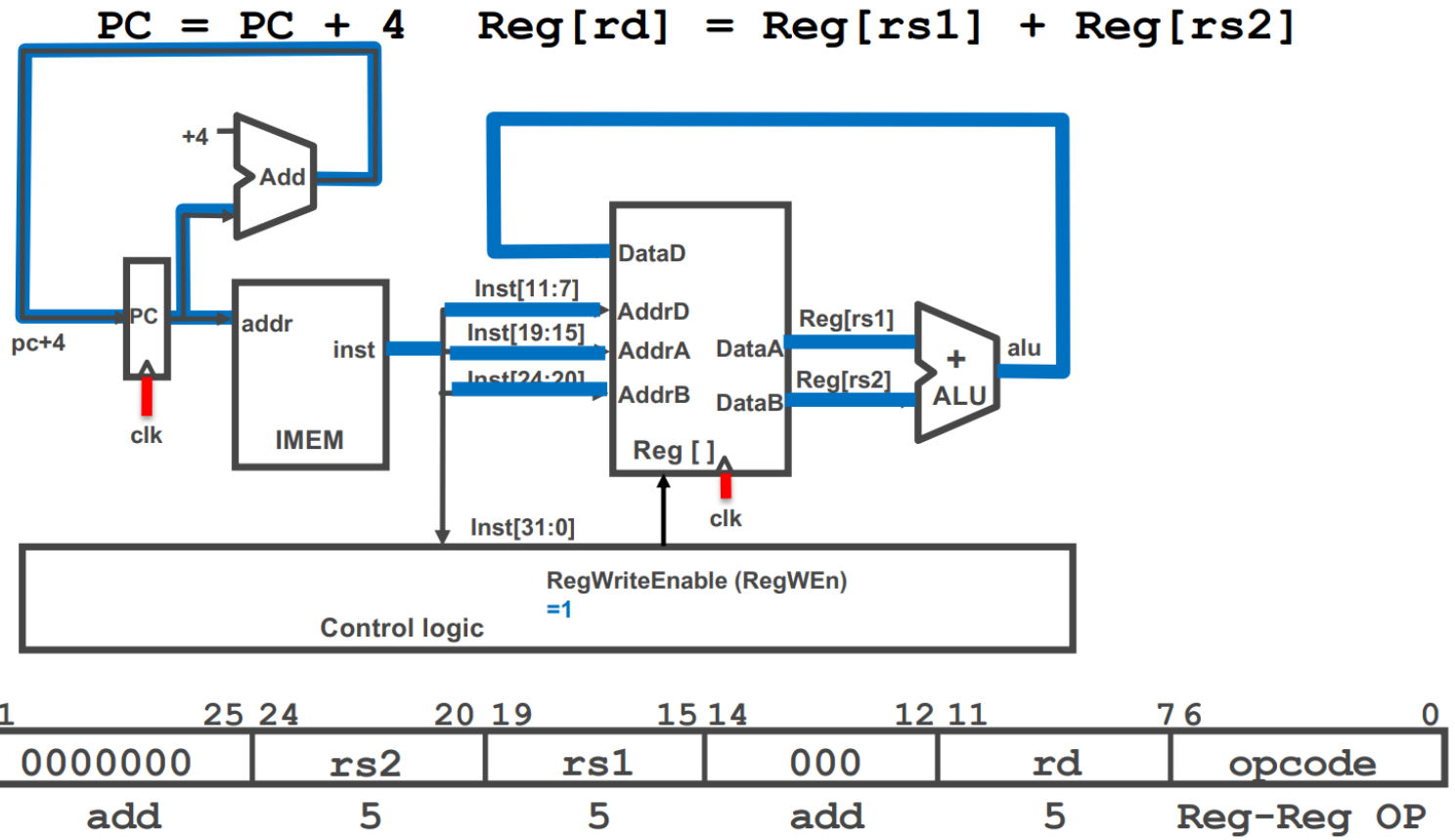
Writeback (WB)
Write back the PC+4, the result of the ALU operation, or data from memory to the RegFile.

Five Stage of Instruction Execution



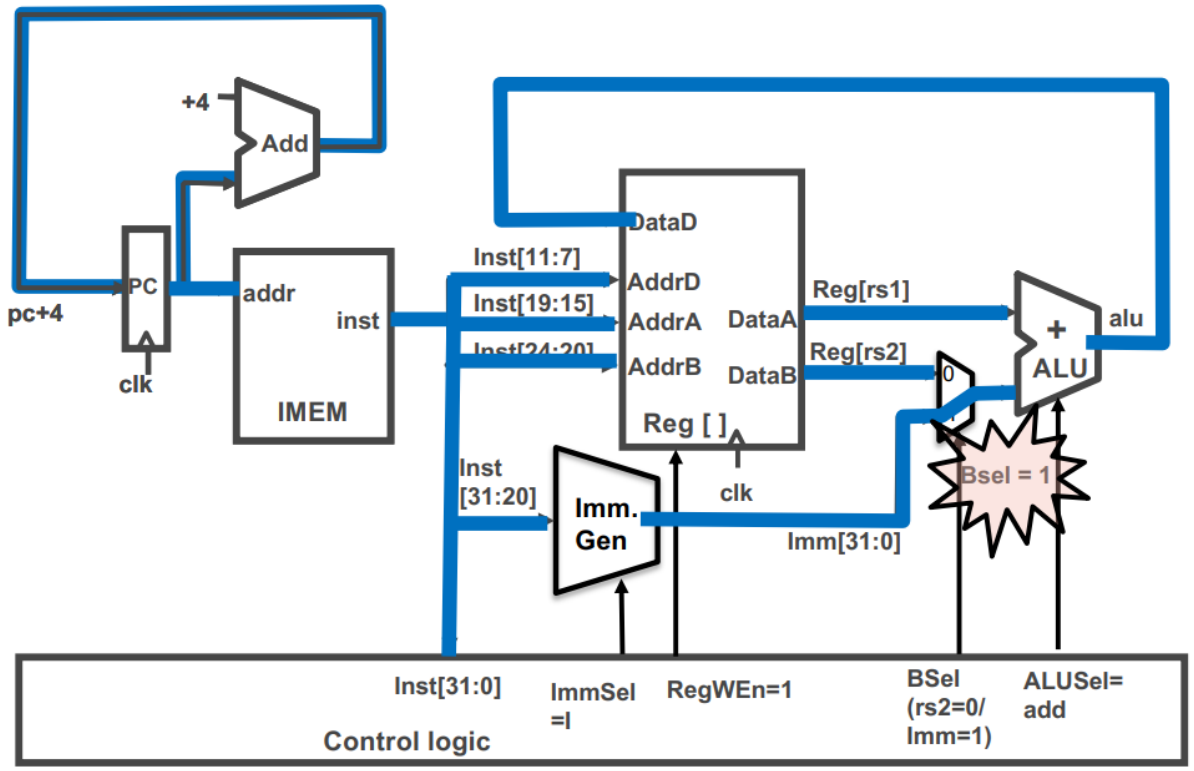
Build a Datapath (S-type)

Datapath for add

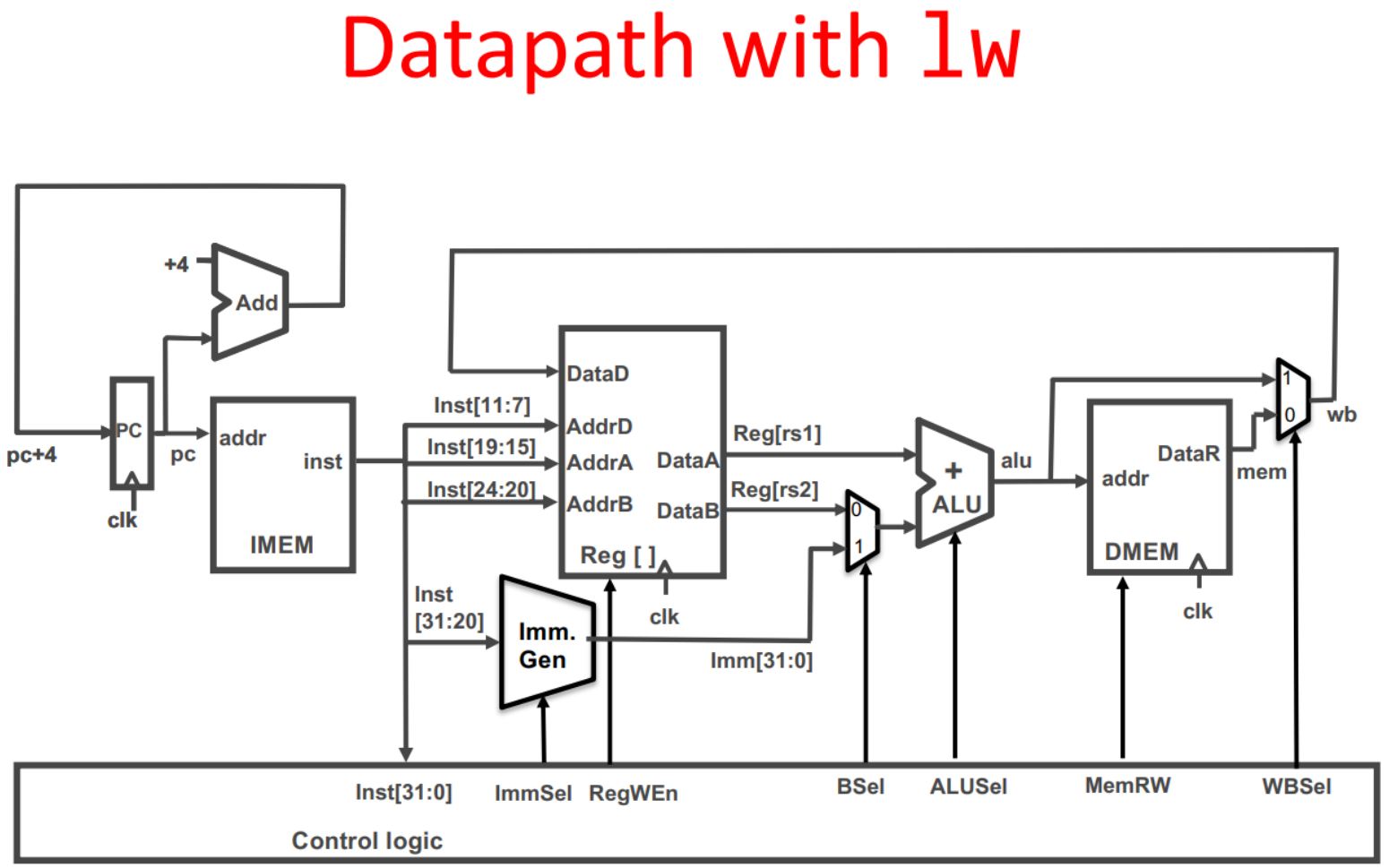


Build a Datapath (+ I-type w/o mem)

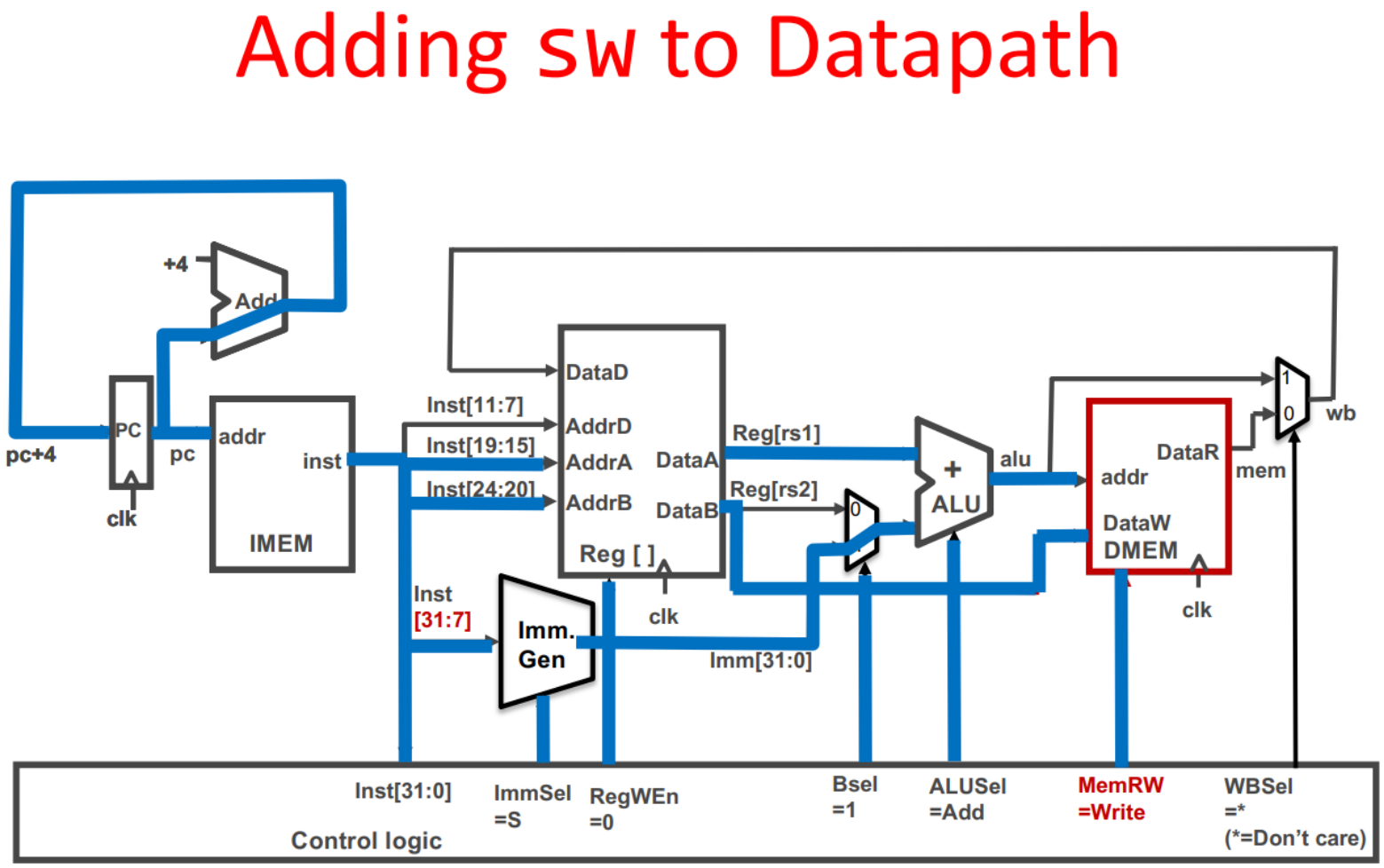
Adding `addi` to Datapath



Build a Datapath (+ load)

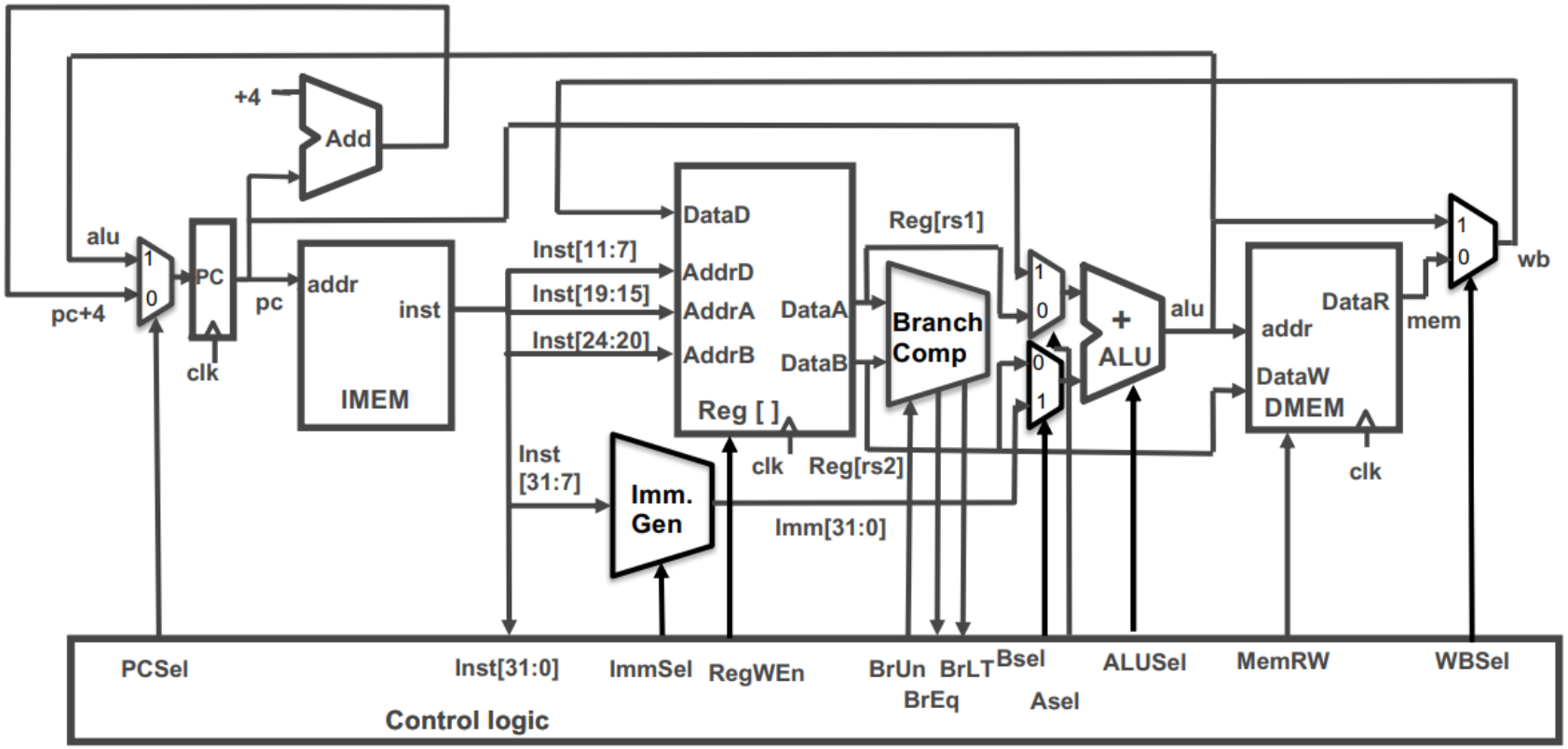


Build a Datapath (+ store)

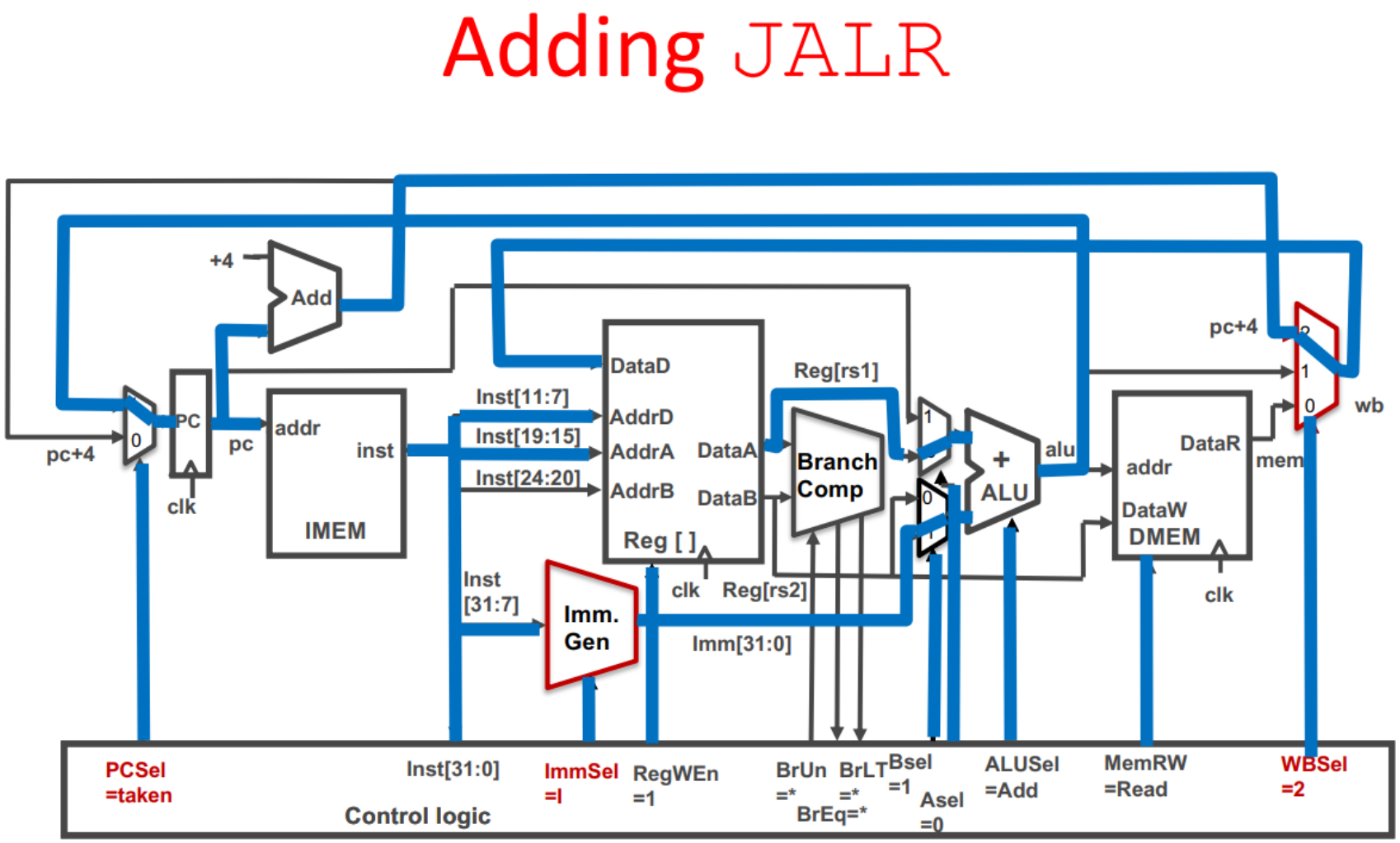


Build a Datapath (+ branch)

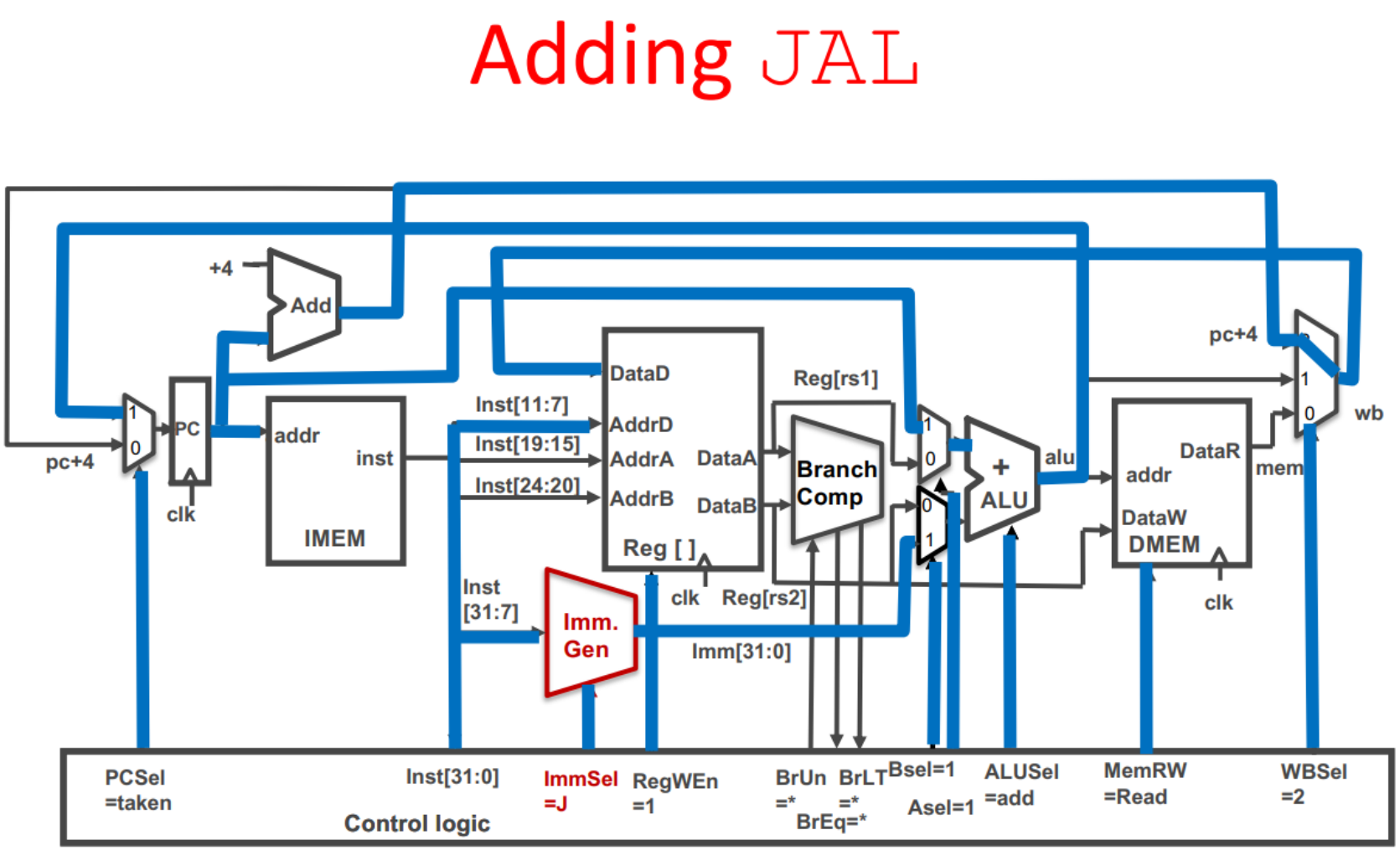
Datapath So Far, with Branches



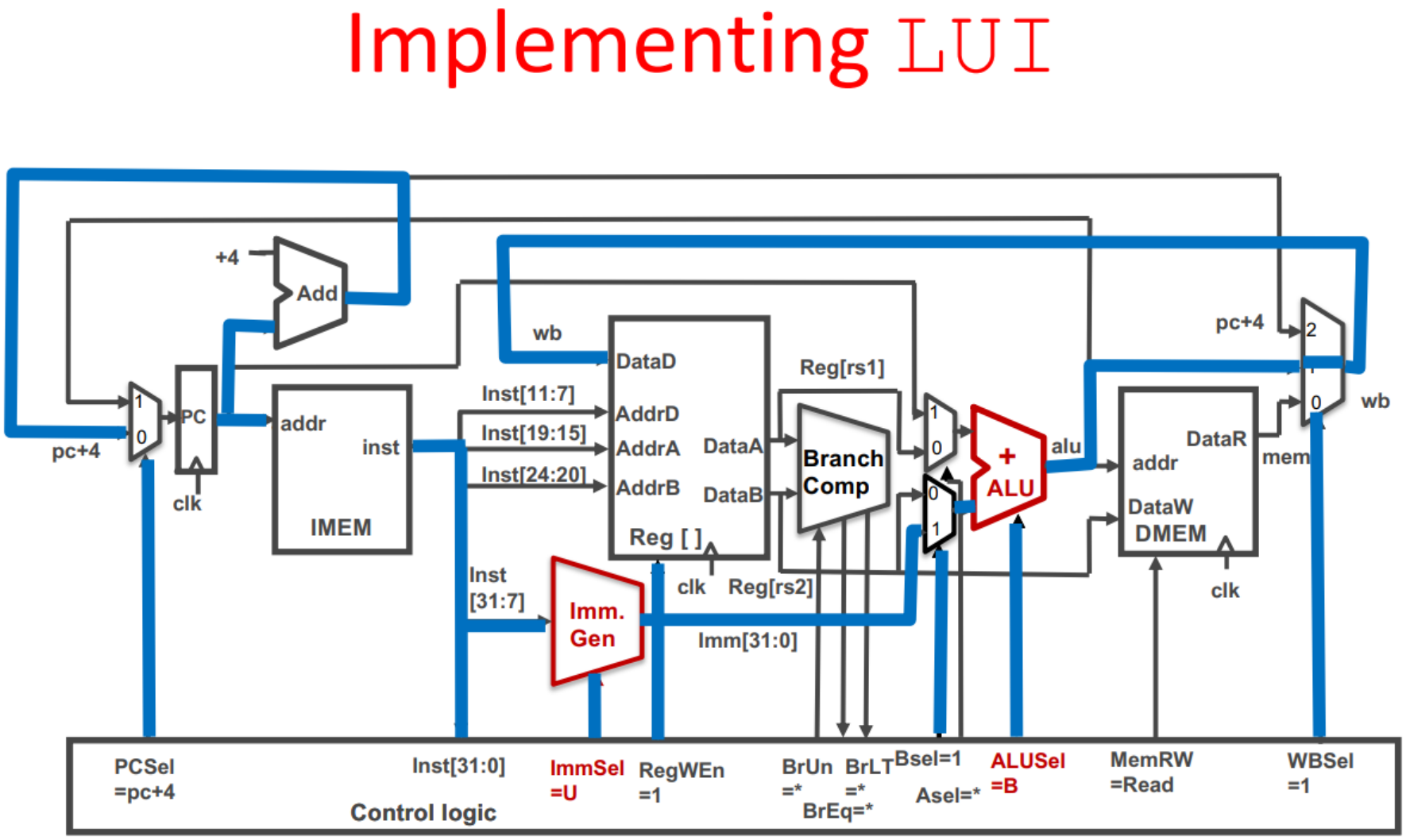
Build a Datapath (+ JALR)



Build a Datapath (+ JAL)

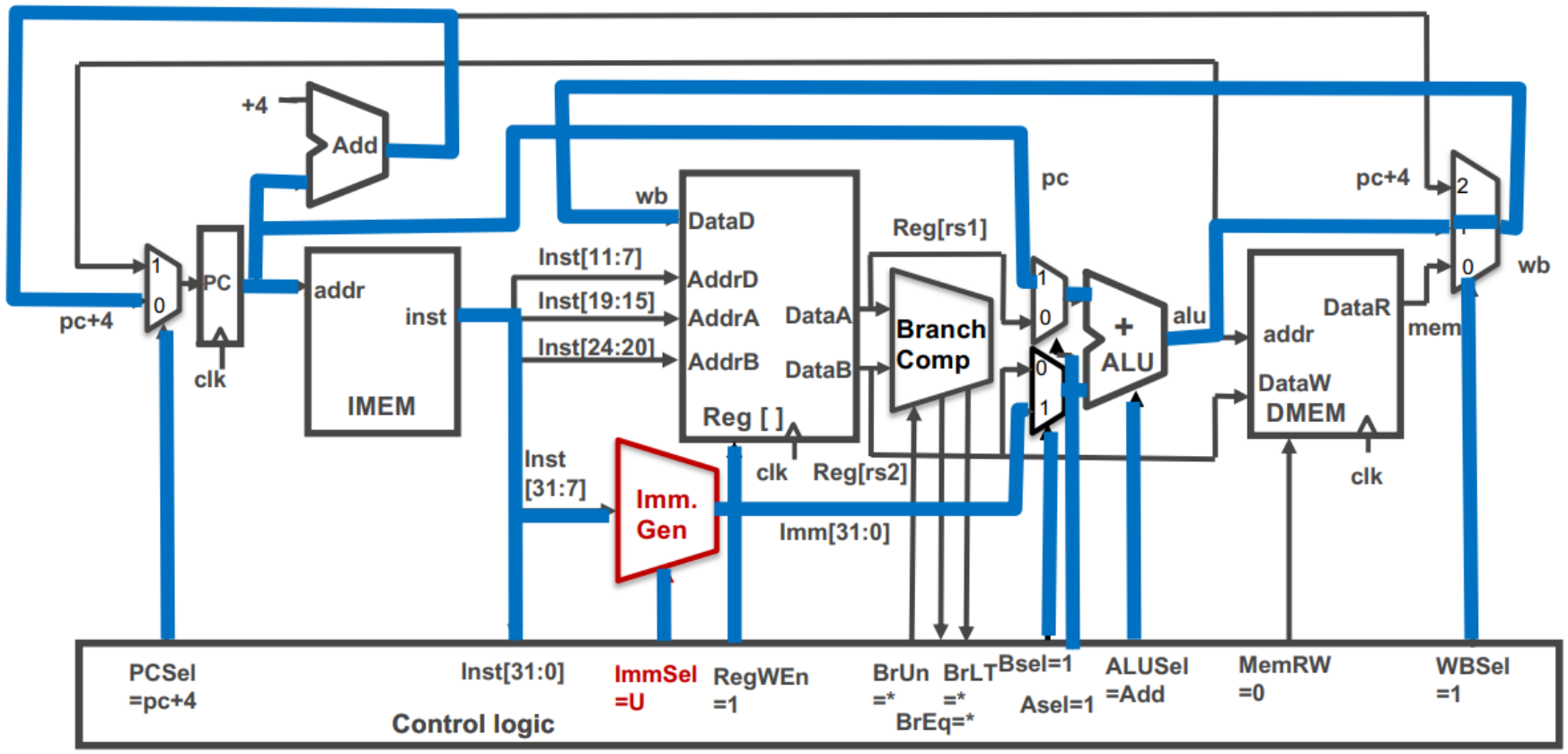


Build a Datapath (+ LUI)

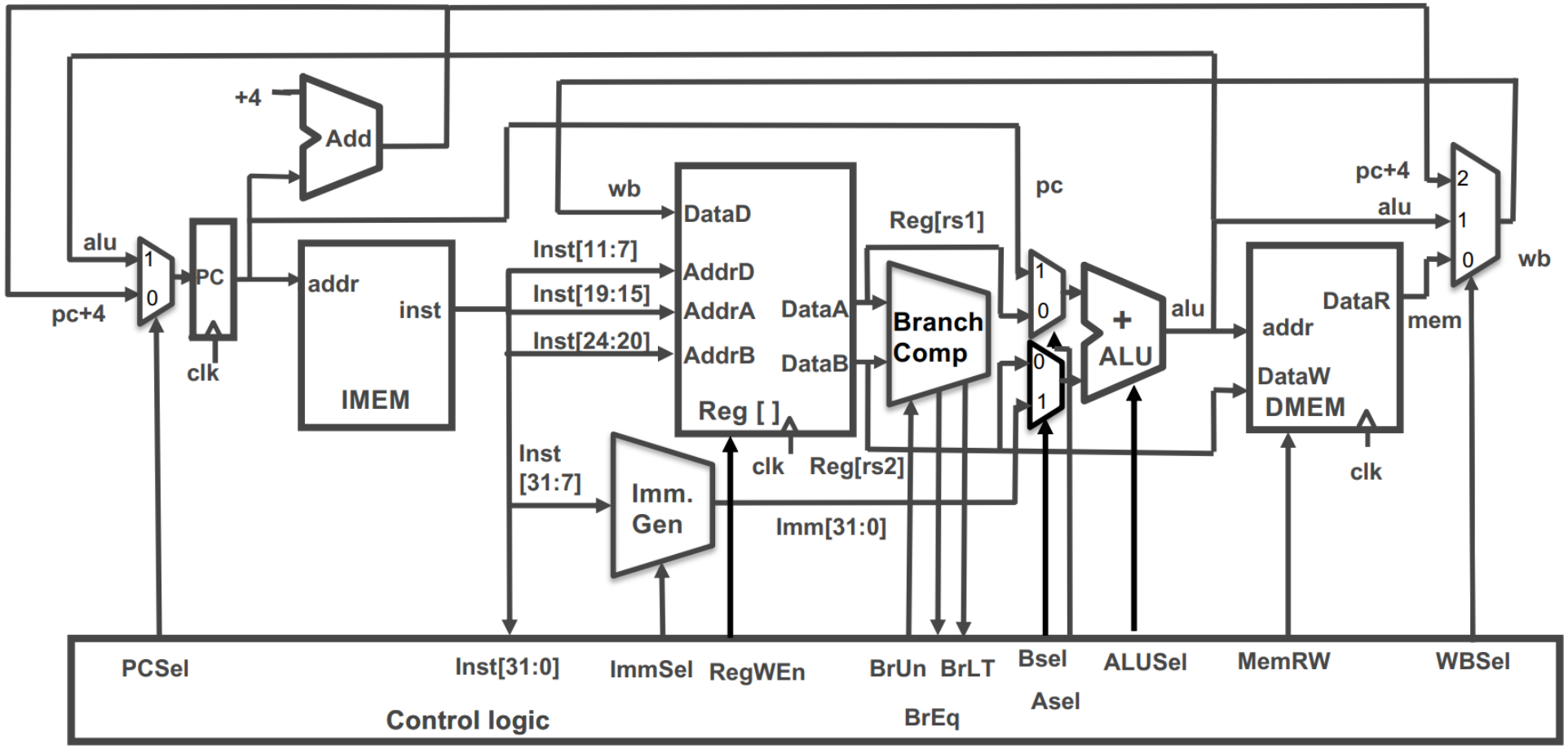


Build a Datapath (+ AUIPC)

Implementing AUIPC

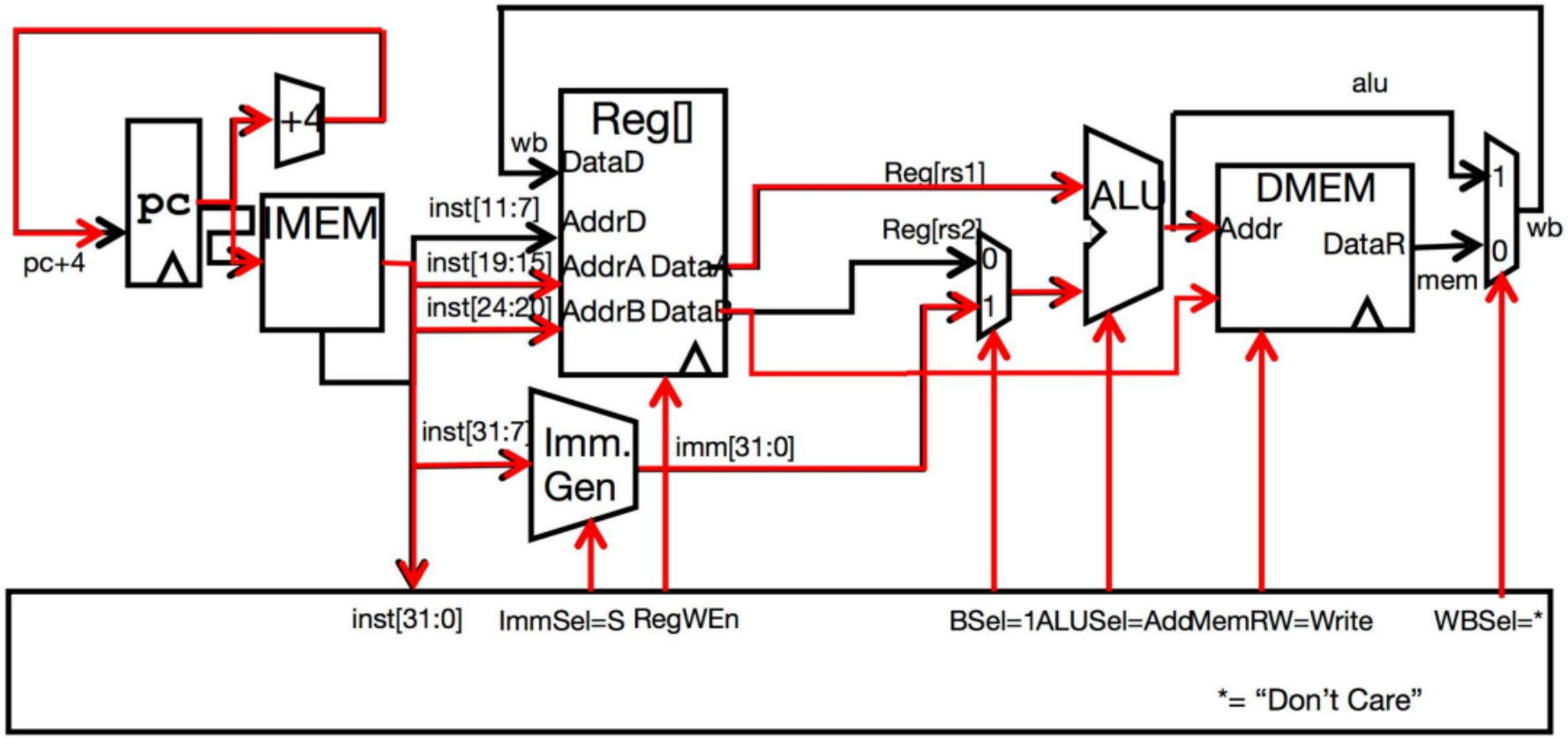


Build a Datapath!



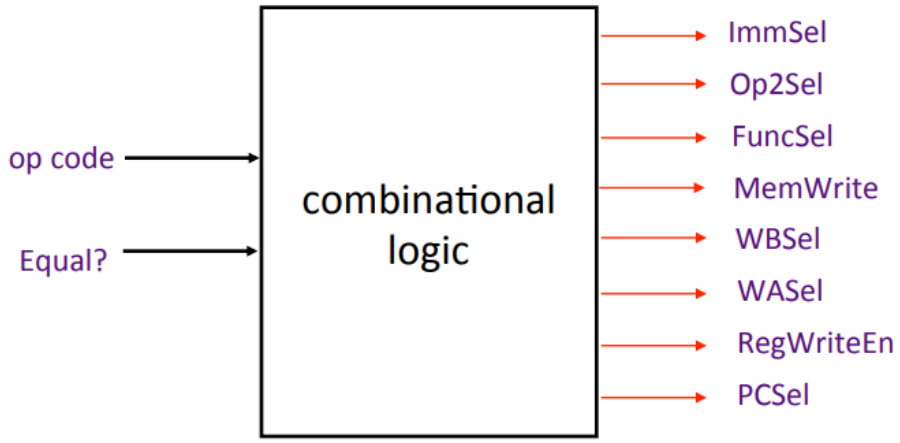
Example

sw S M[R[rs1]+imm](31:0) = R[rs2](31:0), PC = PC + 4



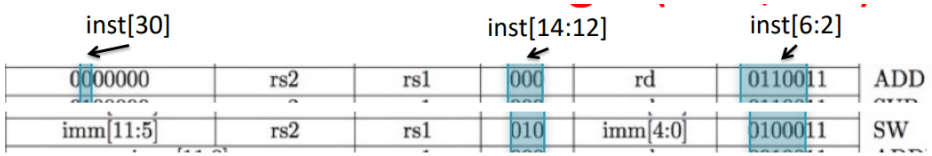
Control Logic

A *BIG* truth table!
2 ways of implementation: ROM / CL
Easy: ROM; Faster: CL



Inst[31:0]	BrEq	BrLT	PCSel	ImmSel	BrUn	ASel	BSel	ALUSel	MemRW	RegWEn	WBSel
add	*	*	+4	*	*	Reg	Reg	Add	Read	1	ALU
sub	*	*	+4	*	*	Reg	Reg	Sub	Read	1	ALU
(R-R Op)	*	*	+4	*	*	Reg	Reg	*(Op)*	Read	1	ALU
addi	*	*	+4	I	*	Reg	Imm	Add	Read	1	ALU
lw	*	*	+4	I	*	Reg	Imm	Add	Read	1	Mem
sw	*	*	+4	S	*	Reg	Imm	Add	Write	0	*
beq	0	*	+4	B	*	PC	Imm	Add	Read	0	*
beq	1	*	ALU	B	*	PC	Imm	Add	Read	0	*
bne	0	*	ALU	B	*	PC	Imm	Add	Read	0	*
bne	1	*	+4	B	*	PC	Imm	Add	Read	0	*
blt	*	1	ALU	B	0	PC	Imm	Add	Read	0	*
bltu	*	1	ALU	B	1	PC	Imm	Add	Read	0	*
jalr	*	*	ALU	I	*	Reg	Imm	Add	Read	1	PC+4
jal	*	*	ALU	J	*	PC	Imm	Add	Read	1	PC+4
auipc	*	*	+4	U	*	PC	Imm	Add	Read	1	ALU

Control Logic (How to CL)



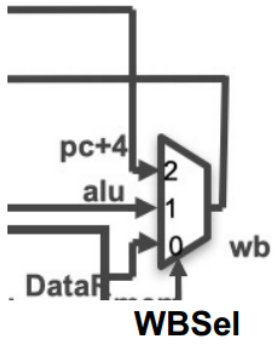
- add= $\overline{inst[2]} \bullet \overline{inst[3]} \bullet inst[4] \bullet inst[5] \bullet \overline{inst[6]} \bullet \overline{inst[12]} \bullet \overline{inst[13]} \bullet \overline{inst[14]} \bullet \overline{inst[30]}$
- sw= $\overline{inst[2]} \bullet \overline{inst[3]} \bullet \overline{inst[4]} \bullet inst[5] \bullet \overline{inst[6]} \bullet \overline{inst[12]} \bullet inst[13] \bullet \overline{inst[14]}$

AND Controller Logic

	2	3	4	5	6	12	13	14	30	unused
add	xor(i[2], 1) • xor(i[3], 1) • xor(i[4], 0) • xor(i[5], 0) • xor(i[6], 1) • xor(i[12], 1) • xor(i[13], 1) • xor(i[14], 1) • (xor(i[30], 1) +									0)
sw	xor(i[2], 1) • xor(i[3], 1) • xor(i[4], 1) • xor(i[5], 0) • xor(i[6], 1) • xor(i[12], 1) • xor(i[13], 1) • xor(i[14], 1) • (xor(i[30], 1) +									1)
...										

- bne_t = bne • **$\overline{BrEQ}$**
- bne_f = bne • **BrEQ**

- MemRW = sw + sh + sb
- WBSel[0] = add + (all reg. to reg.) + AUIPC + …
- WBSel[1] = JALR + JAL
- PCSel = beq_t + bne_t + blt_t + bltu_t + jal + jalr

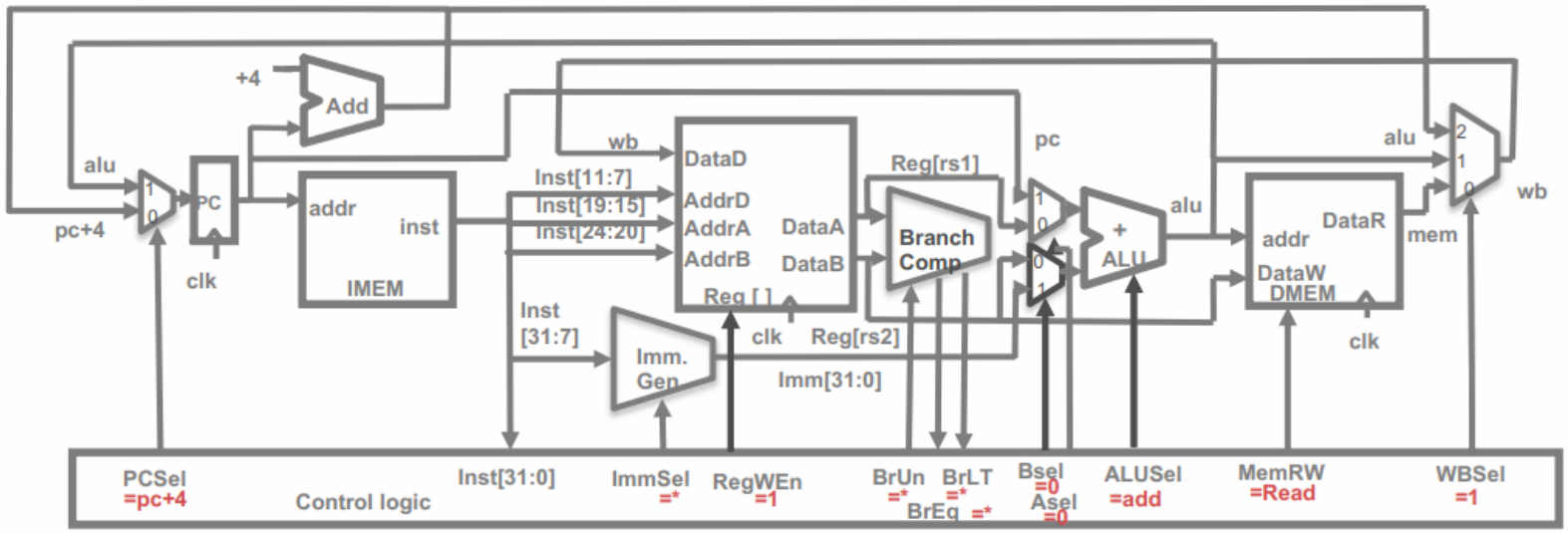


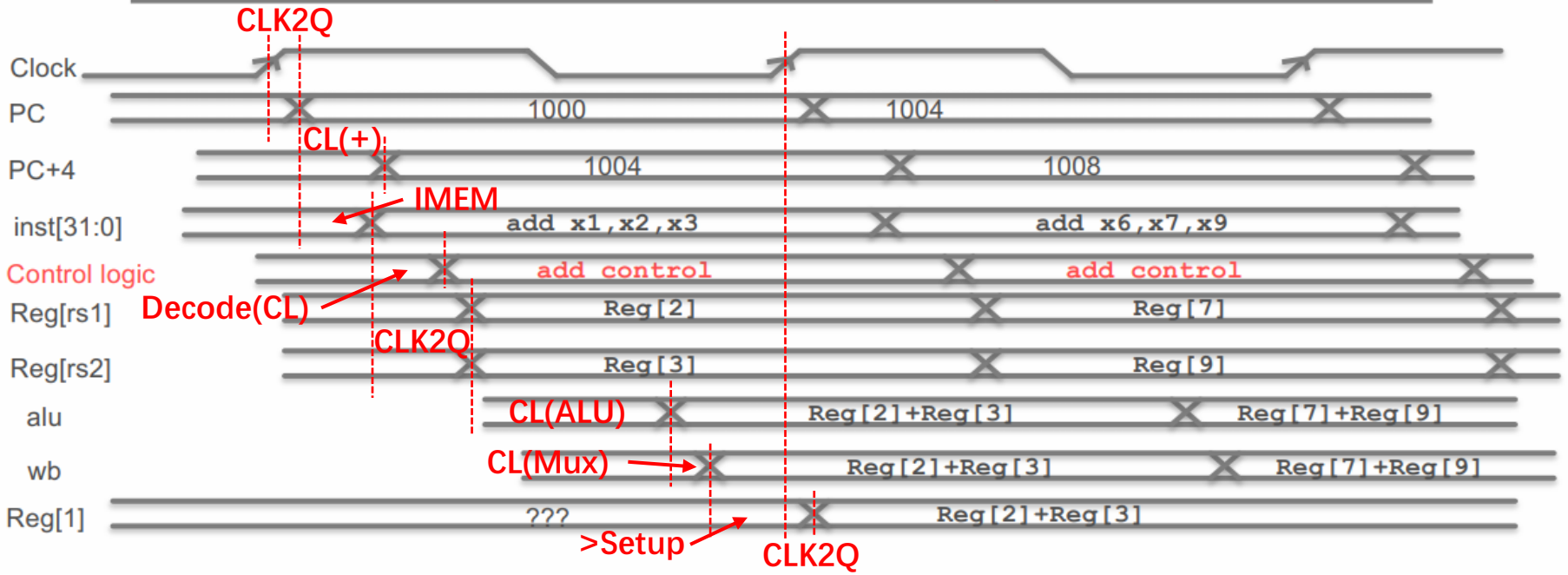
Input

Output

Timing

add Execution

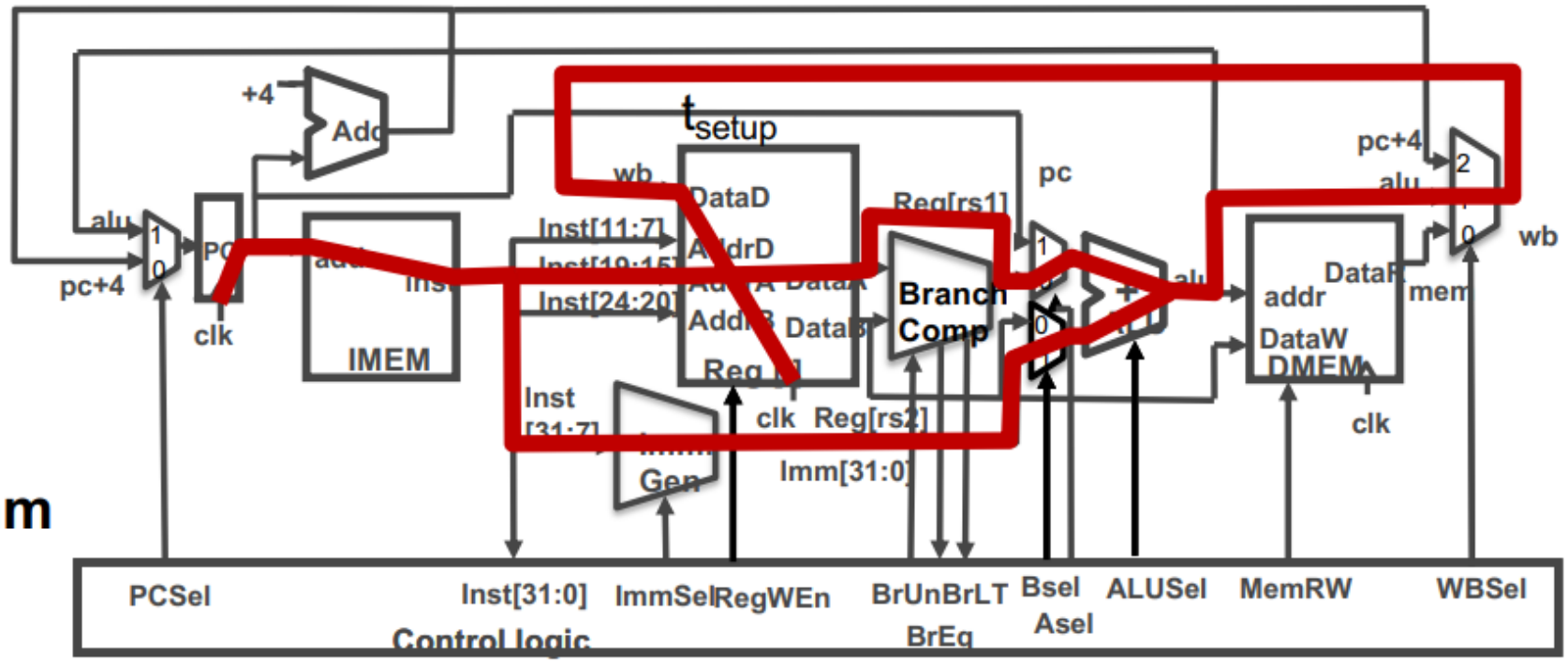




Question: Critical Path

Critical path for xori

R[rd] = R[rs1]+imm



A. $t_{clk\text{-}q} + t_{IMEM} + \max\{t_{Reg}, t_{Imm}\} + t_{ALU} + 2*t_{mux} + t_{Setup}$

B. $t_{clk\text{-}q} + t_{Add} + t_{IMEM} + t_{Reg} + t_{BComp} + t_{ALU} + t_{DMEM} + t_{mux} + t_{Setup}$

C. $t_{clk\text{-}q} + t_{IMEM} + \max\{t_{Reg}, t_{Imm}\} + t_{ALU} + 3*t_{mux} + t_{DMEM} + t_{Setup}$

D. None of the above

Timing

clock period is sufficiently long for all of the following steps to be “completed”:

1. Instruction fetch
2. Decode and register fetch
3. ALU operation
4. Data fetch if required
5. Register write-back setup time

=> tC > tIFetch + tRFetch + tALU+ tDMem+ tRWB

At the rising edge of the following clock, the PC, register file and memory are updated

Questions

1. Which instruction exercise the critical path?

Load Word (lw). It uses all 5 stages.
IF+ID must; EX for offset; MEM for load; WB for write reg.

2. Why is the single cycle datapath inefficient?

At any given time, most of the parts of the single cycle datapath are sitting unused. Also, even though not every instruction exercises the critical path, the datapath can only be clocked as fast as the slowest instruction.

3. How can you improve its performance? What is the purpose of pipelining?

Performance can be improved with pipelining, or putting registers between stages so that the amount of conditional logic between registers is reduced, allowing for a faster clock time.