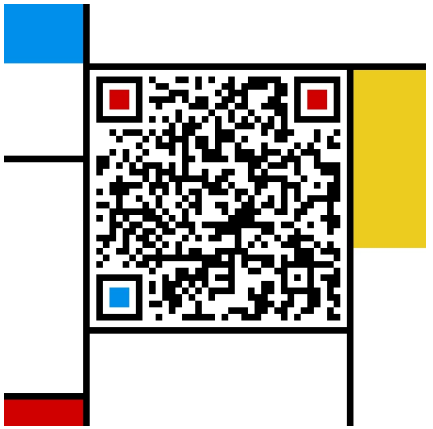
计算机组成与系统结构
Computer Organization & System Architecture

Huang Kejie (黄科杰) 百人计划研究员
Office: 玉泉校区老生仪楼 304
Email address: huangkejie@zju.edu.cn
HP: 17706443800



Recap: Complete RV32I ISA

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
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
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI

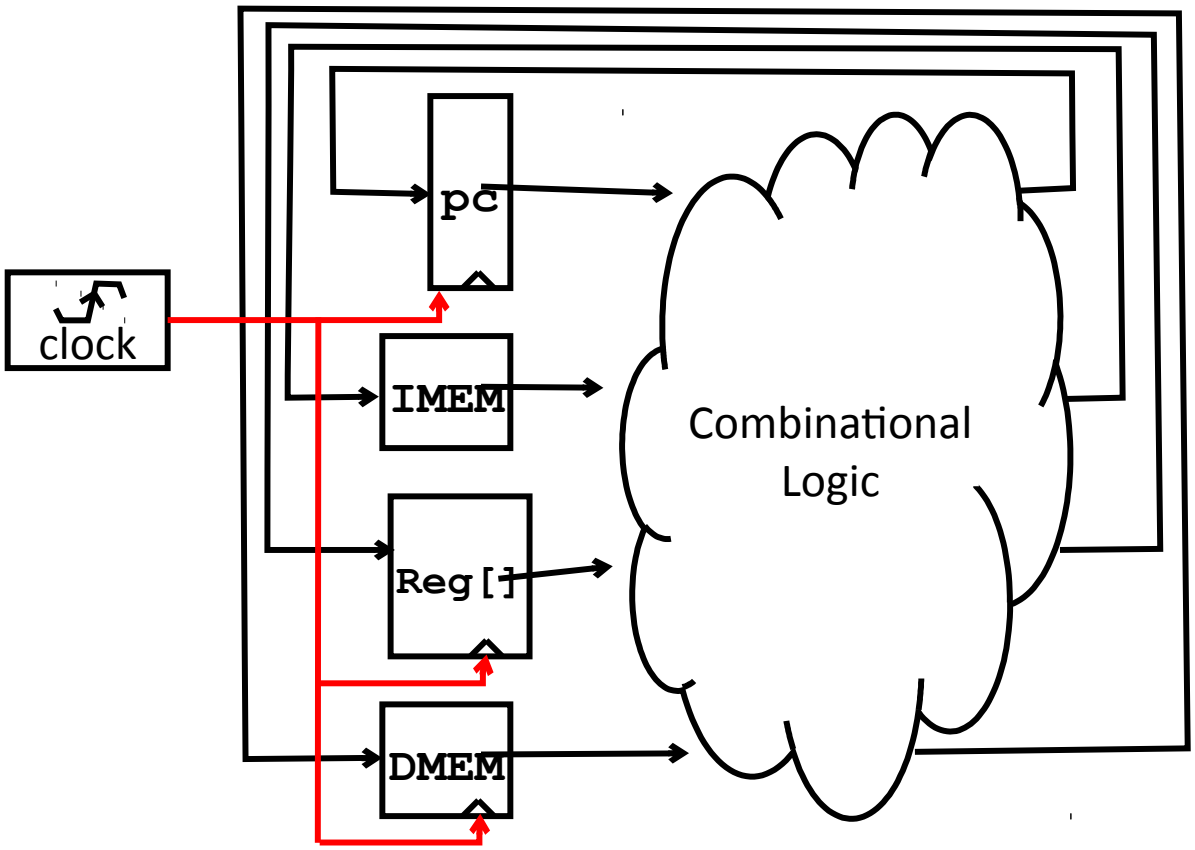
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
0000	pred	succ	00000	000	00000	0001111	FENCE
0000	0000	0000	00000	001	00000	0001111	FENCE.I
000000000000			00000	000	00000	1110011	ECALL
000000000001			00000	000	00000	1110011	EBREAK
csr			rs1	001	rd	1110011	CSRRW
csr			rs1	010	rd	1110011	CSRRS
csr			rs1	011	rd	1110011	CSRRC
csr			zimm	101	rd	1110011	CSRRWI
csr			zimm	110	rd	1110011	CSRRSI
csr			zimm	111	rd	1110011	CSRRCI

State Required by RV32I ISA

Each instruction reads and updates this state during execution:

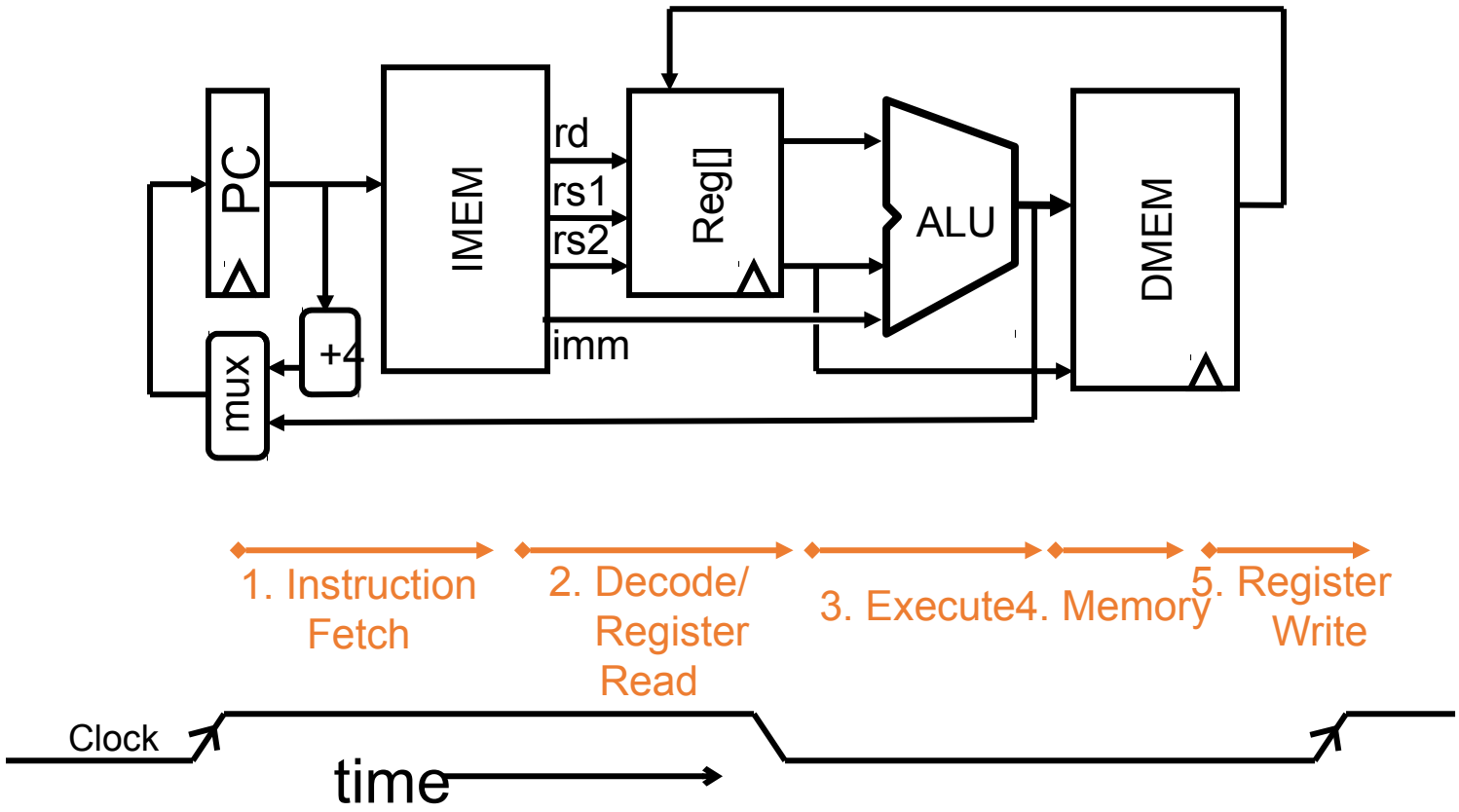
- Registers (**`x0..x31`**)
  - Register file (or regfile) **`Reg`** holds 32 registers x 32 bits/register: **`Reg[0].. Reg[31]`**
  - First register read specified by rs1 field in instruction
  - Second register read specified by rs2 field in instruction
  - Write register (destination) specified by rd field in instruction
  - **`x0`** is always 0 (writes to **`Reg[0]`** are ignored)
- Program Counter (**`PC`**)
  - Holds address of current instruction
- Memory (**`MEM`**)
  - Holds both instructions & data, in one 32-bit byte-addressed memory space
  - We'll use separate memories for instructions (**`IMEM`**) and data (**`DMEM`**)
    - Later we'll replace these with instruction and data caches
  - Instructions are read (fetched) from instruction memory (assume **`IMEM`** read-only)
  - Load/store instructions access data memory

One-Instruction-Per-Cycle RISC-V Machine



- On every tick of the clock, the computer executes one instruction
- Current state outputs drive the inputs to the combinational logic, whose outputs settles at the values of the state before the next clock edge
- At the rising clock edge, all the state elements are updated with the combinational logic outputs, and execution moves to the next clock cycle

Basic Phases of Instruction Execution



Implementing the **`add`** instruction

- RISC-V Assembly Instruction:

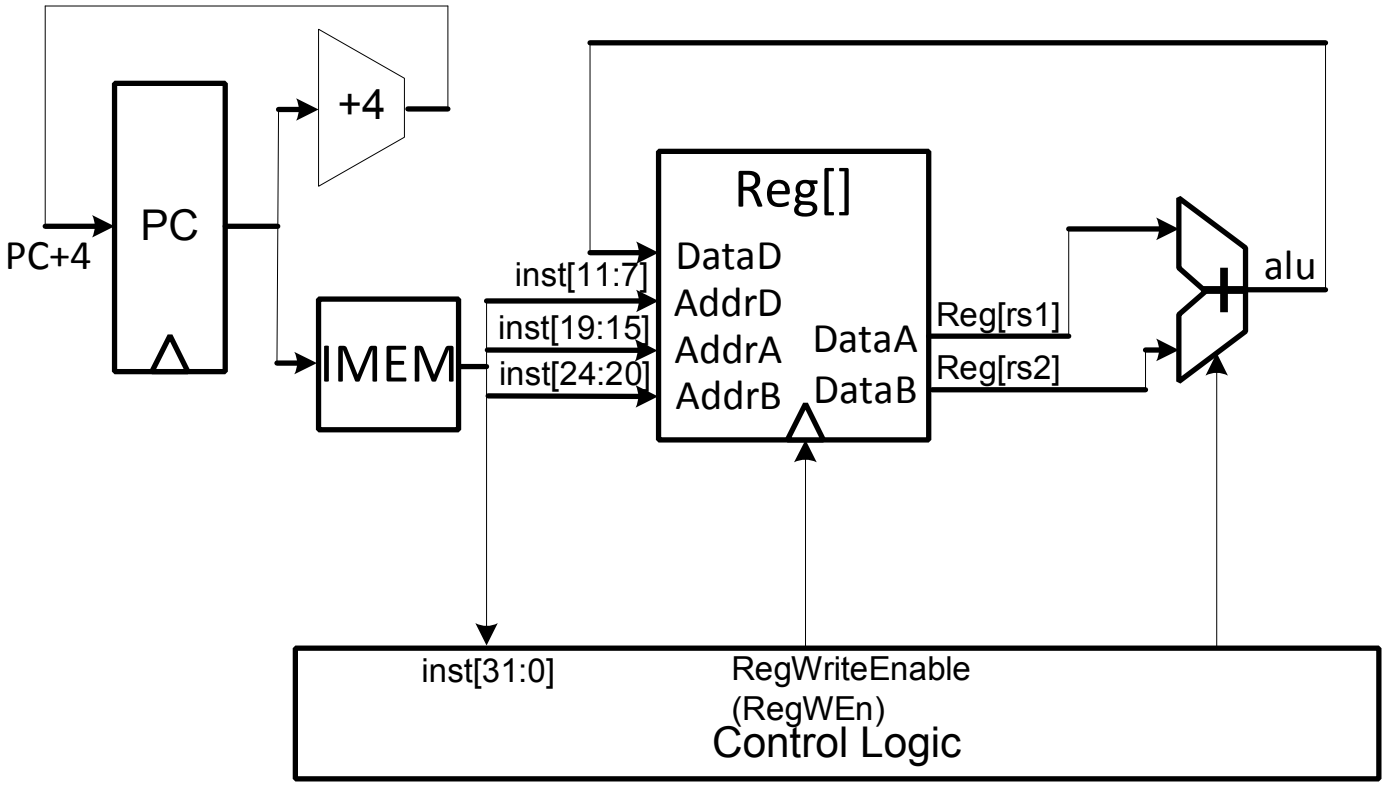
```
add rd, rs1, rs2
```

0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL

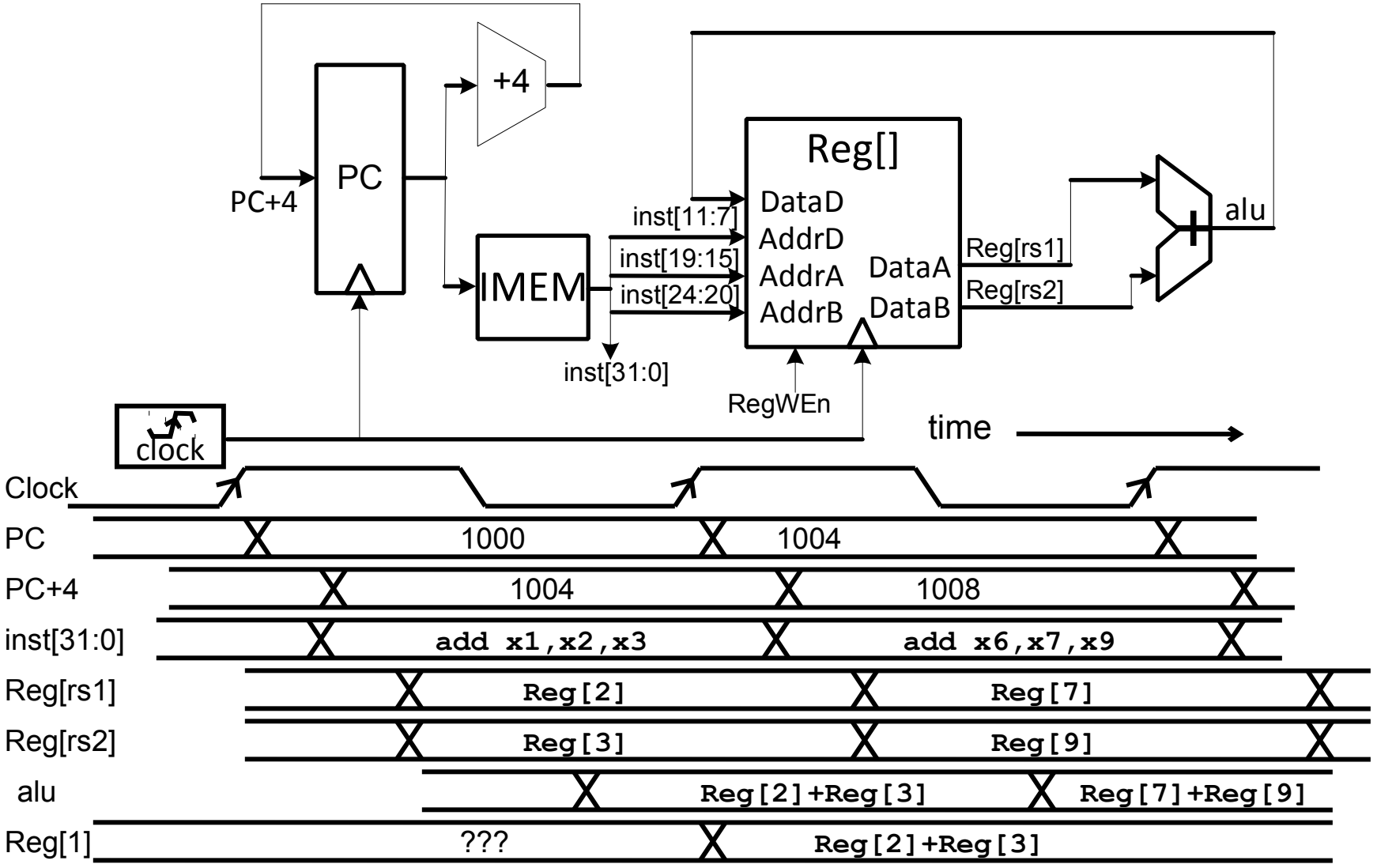
- Instruction makes two changes to machine's state:

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

Datapath for **`add`**



Timing Diagram for `add`



Implementing the **`sub`** instruction

- RISC-V Assembly Instruction:

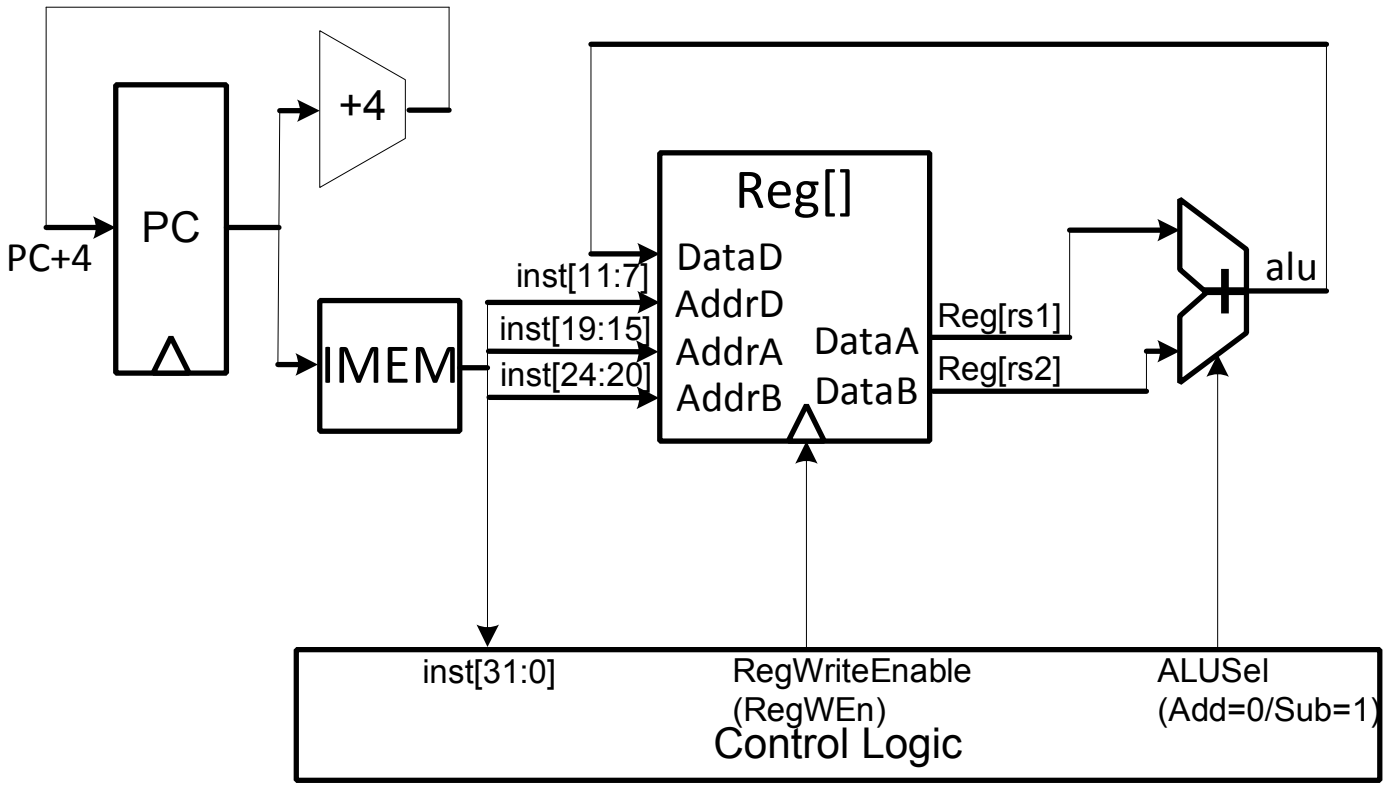
```
sub rd, rs1, rs2
```

0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT

- Almost the same as add, except now have to subtract operands instead of adding them

- **`inst[30]`** selects between add and subtract

Datapath for **`add/sub`**



Implementing other R-Format instructions

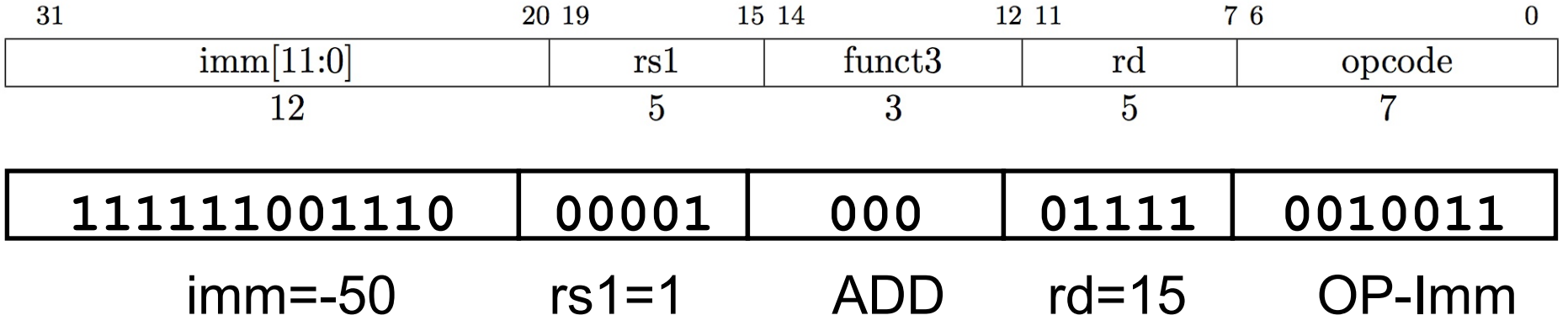
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

- All implemented by decoding funct3 and funct7 fields and selecting appropriate ALU function

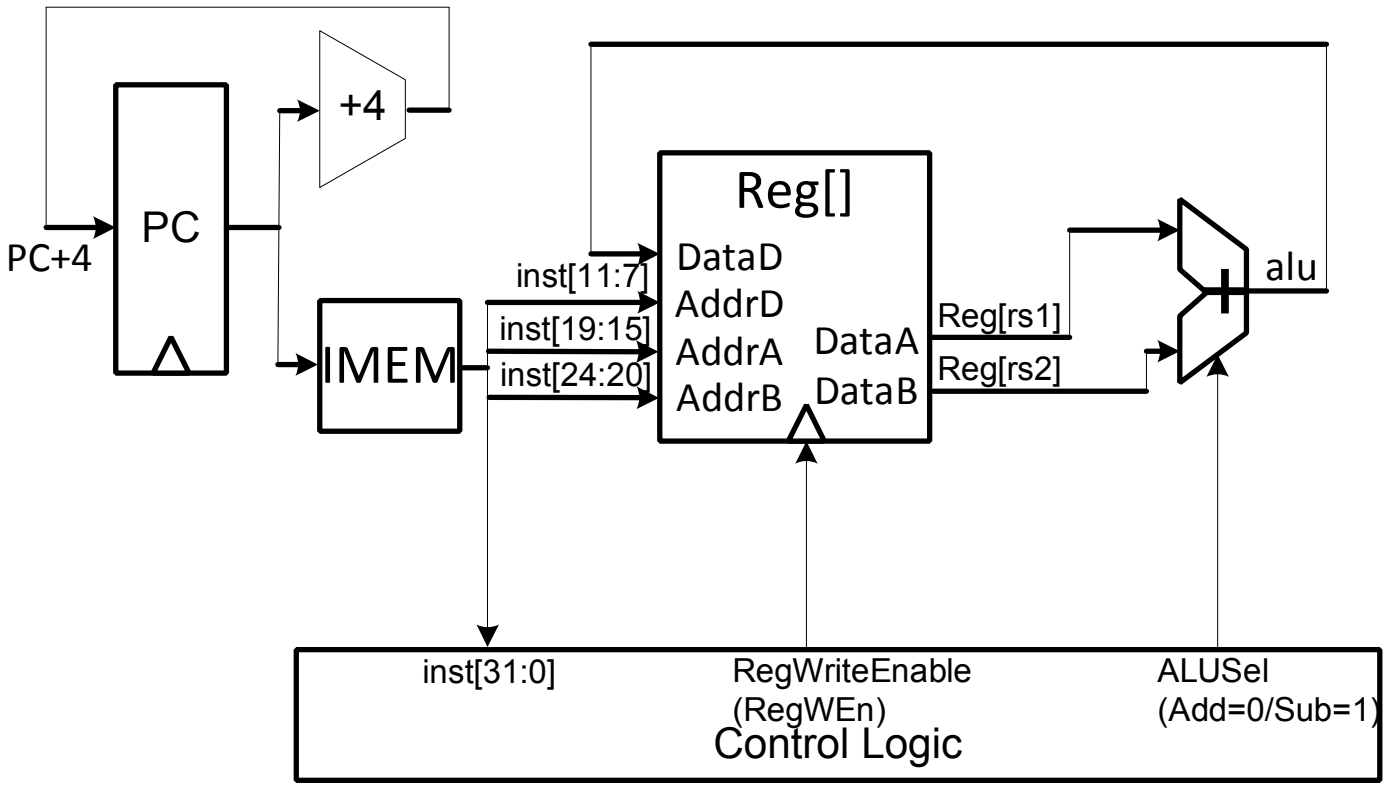
Implementing the **`addi`** instruction

- RISC-V Assembly Instruction:

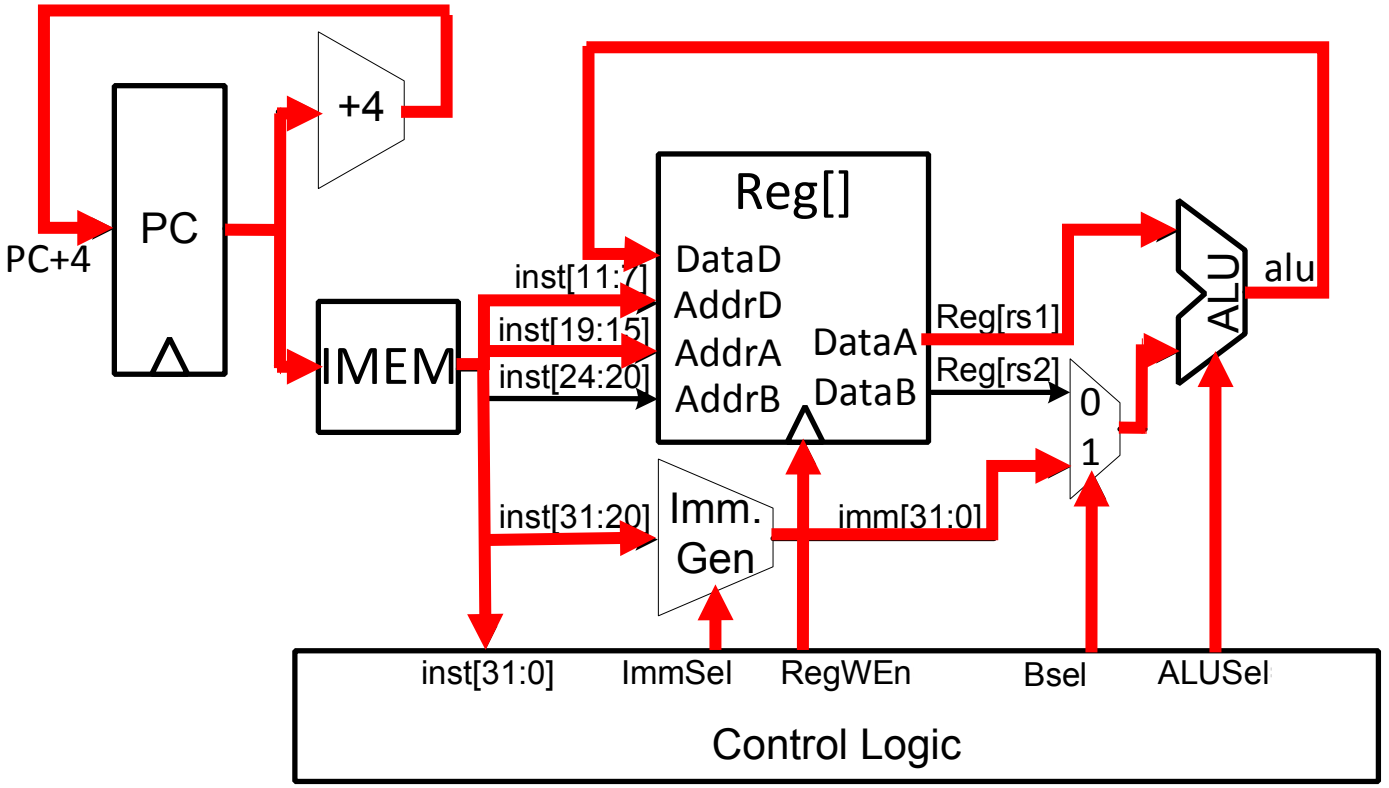
```
addi   x15,x1,-50
```



Datapath for `add/sub`

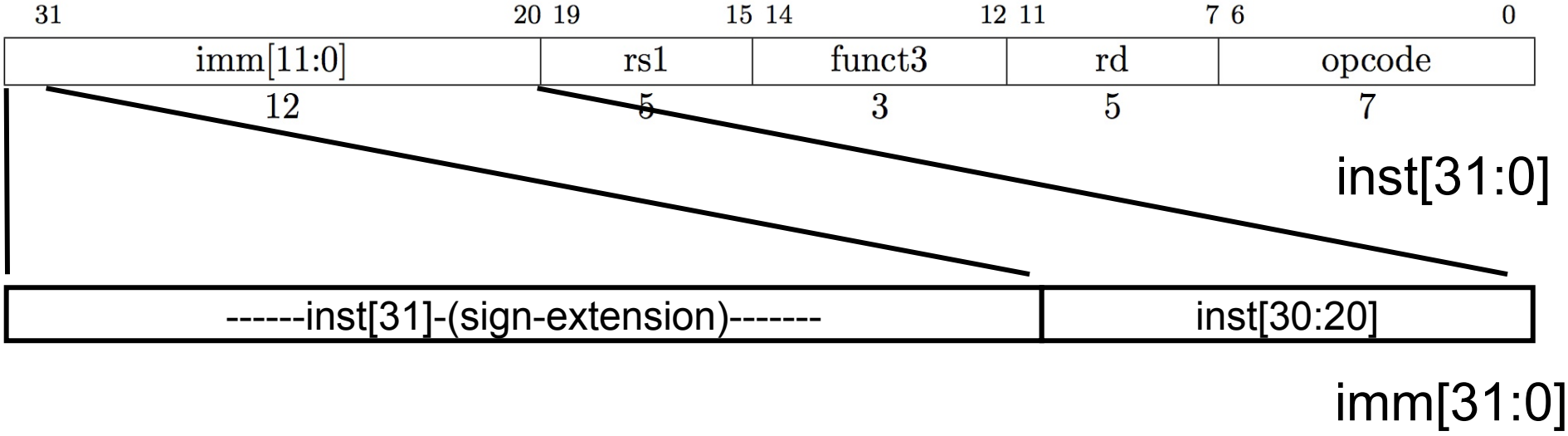


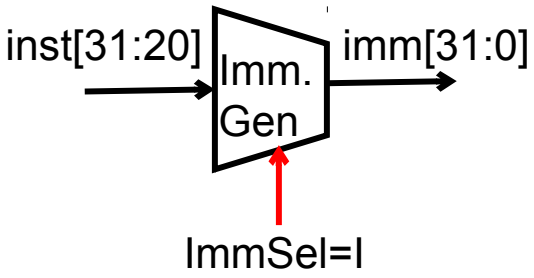
Adding **`addi`** to datapath



*Also works for all other I-format arithmetic instruction (**`slti,sltiu,andi,ori,xori,slli,srli,srai`**) just by changing ALUSel*

I-Format immediates



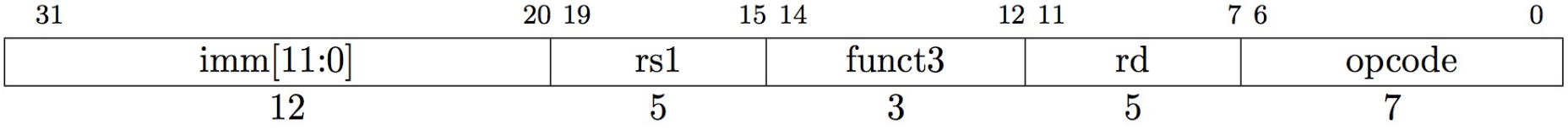


- High 12 bits of instruction (inst[31:20]) copied to low 12 bits of immediate (imm[11:0])
- Immediate is sign-extended by copying value of inst[31] to fill the upper 20 bits of the immediate value (imm[31:12])

Implementing Load Word instruction

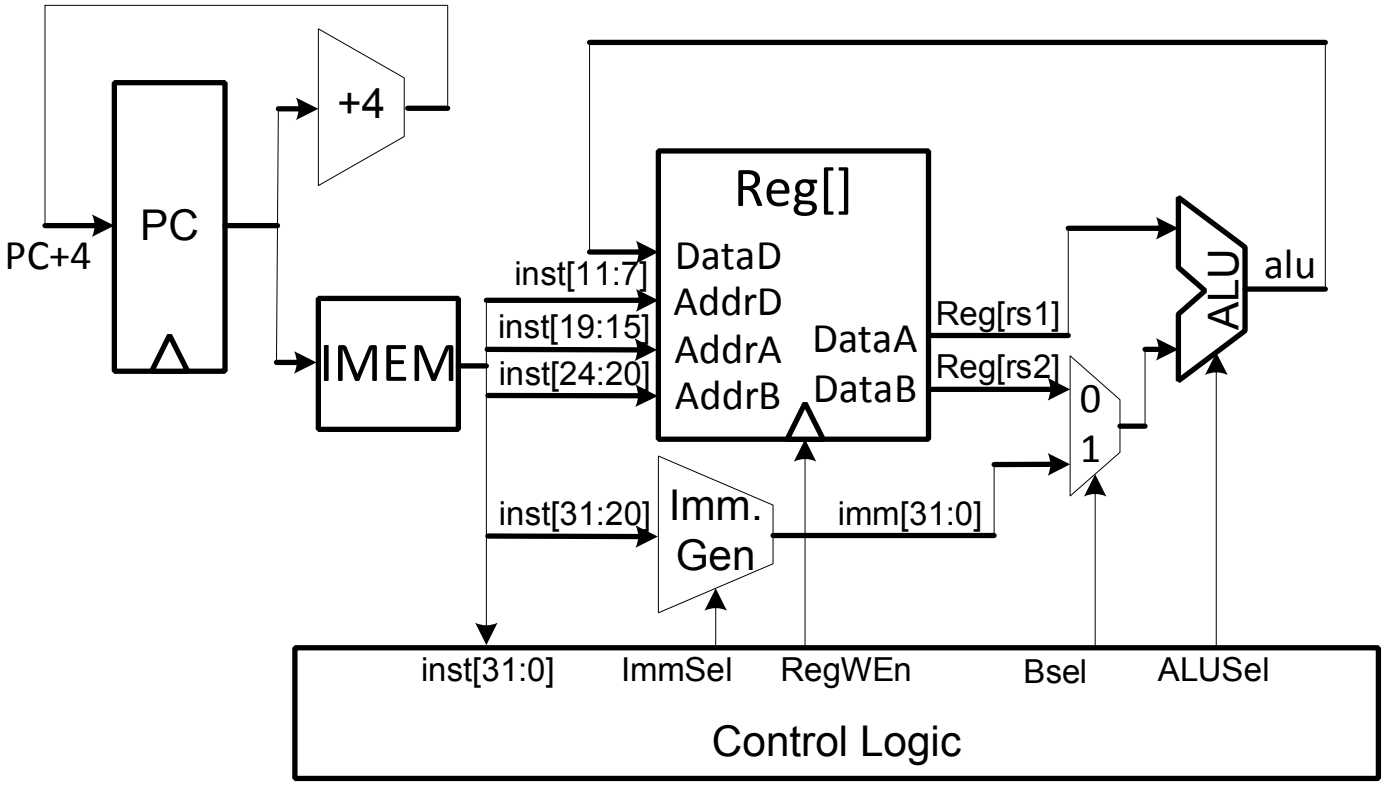
- RISC-V Assembly Instruction:

```
lw x14, 8(x2)
```

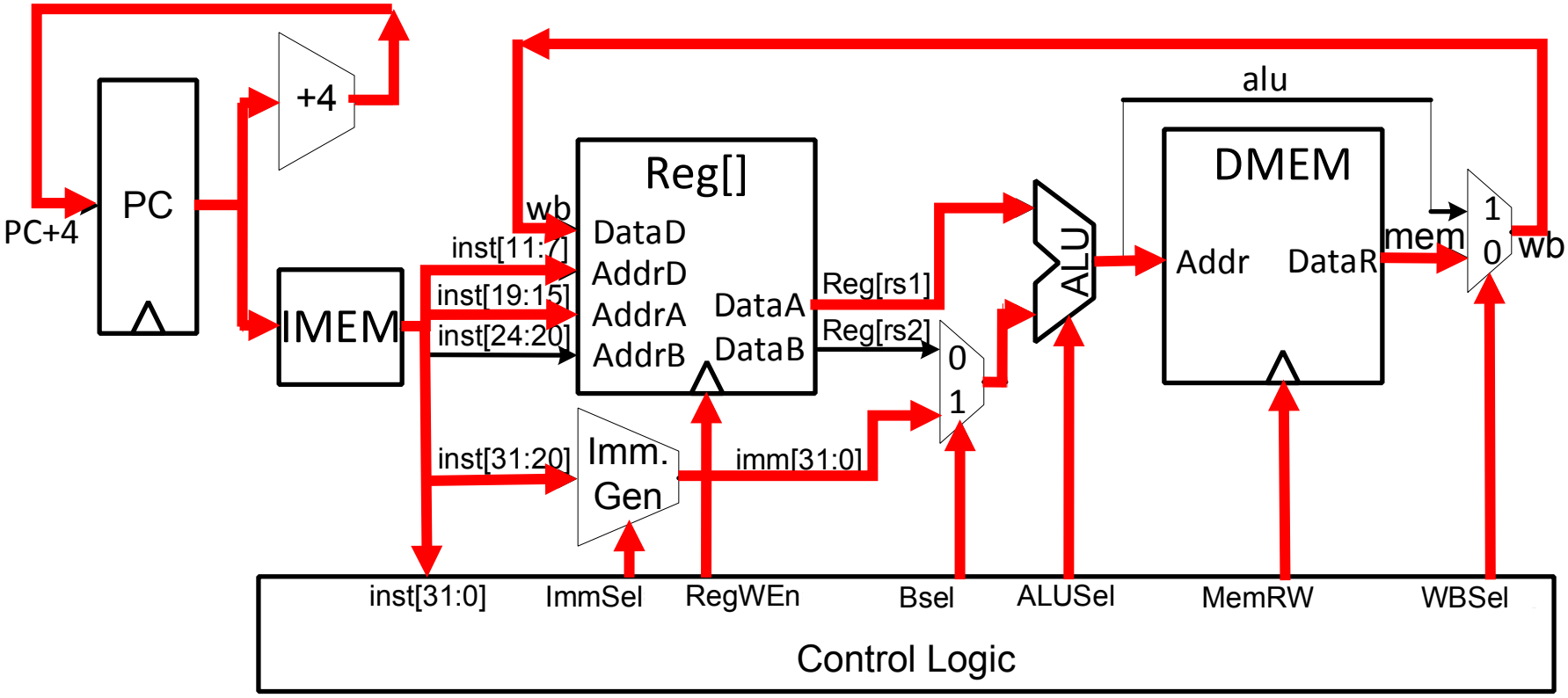


000000001000	00010	010	01110	0000011
imm=+8	rs1=2	LW	rd=14	LOAD

Adding `addi` to datapath



Adding `lw` to datapath



All RV32 Load Instructions

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

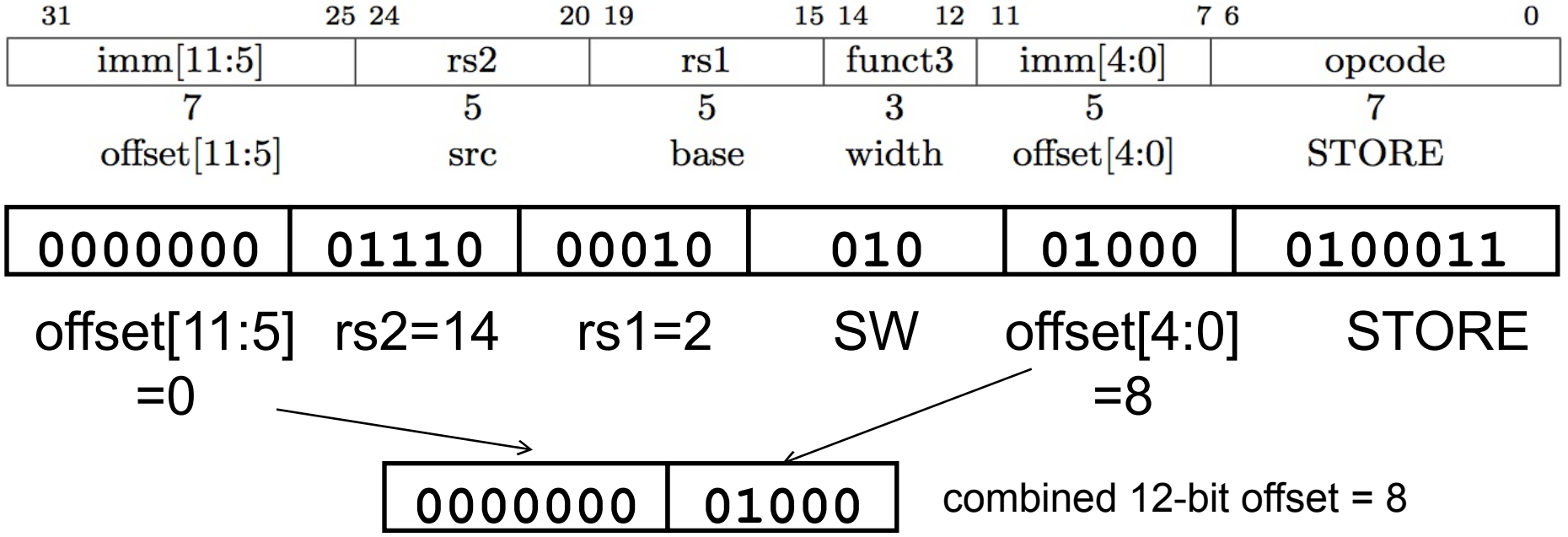
funct3 field encodes size and signedness of load data

- Supporting the narrower loads requires additional circuits to extract the correct byte/halfword from the value loaded from memory, and sign- or zero-extend the result to 32 bits before writing back to register file.

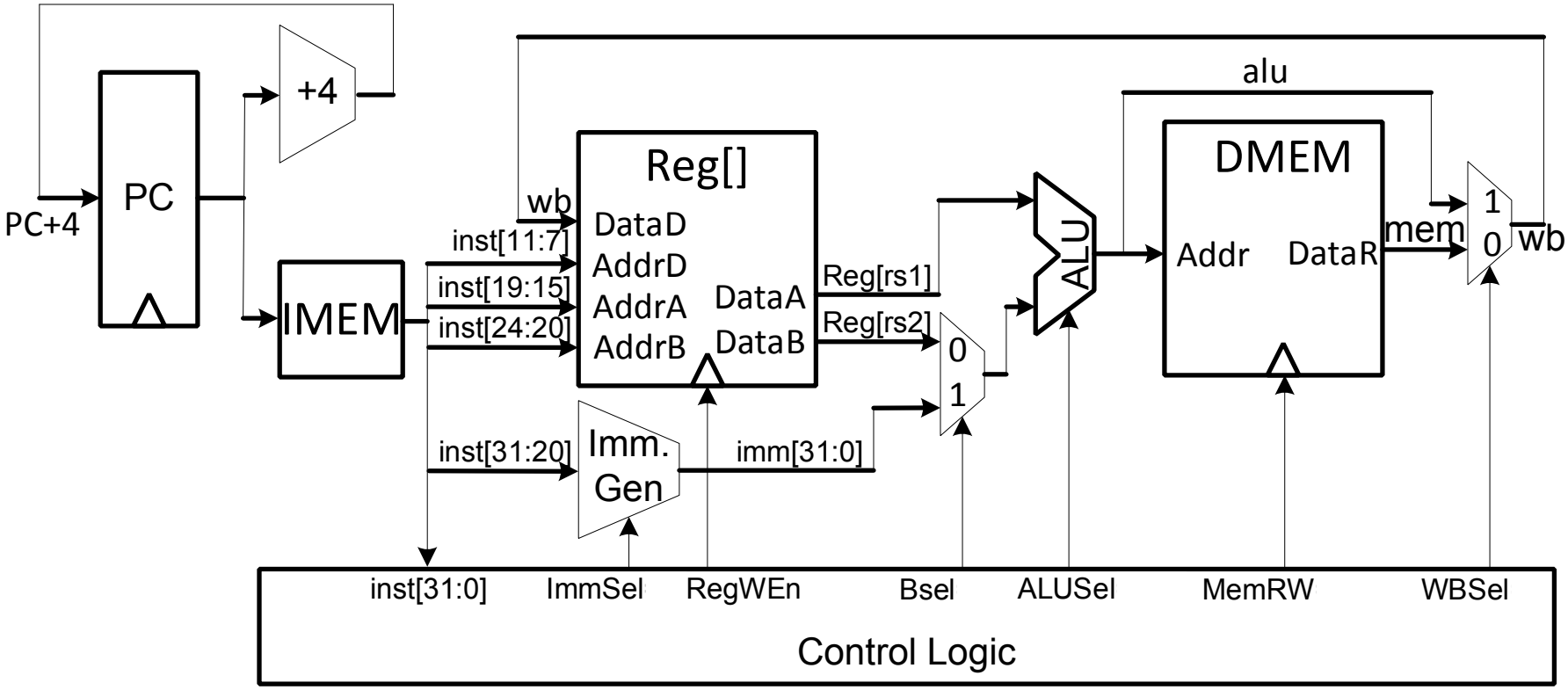
Implementing Store Word instruction

- RISC-V Assembly Instruction:

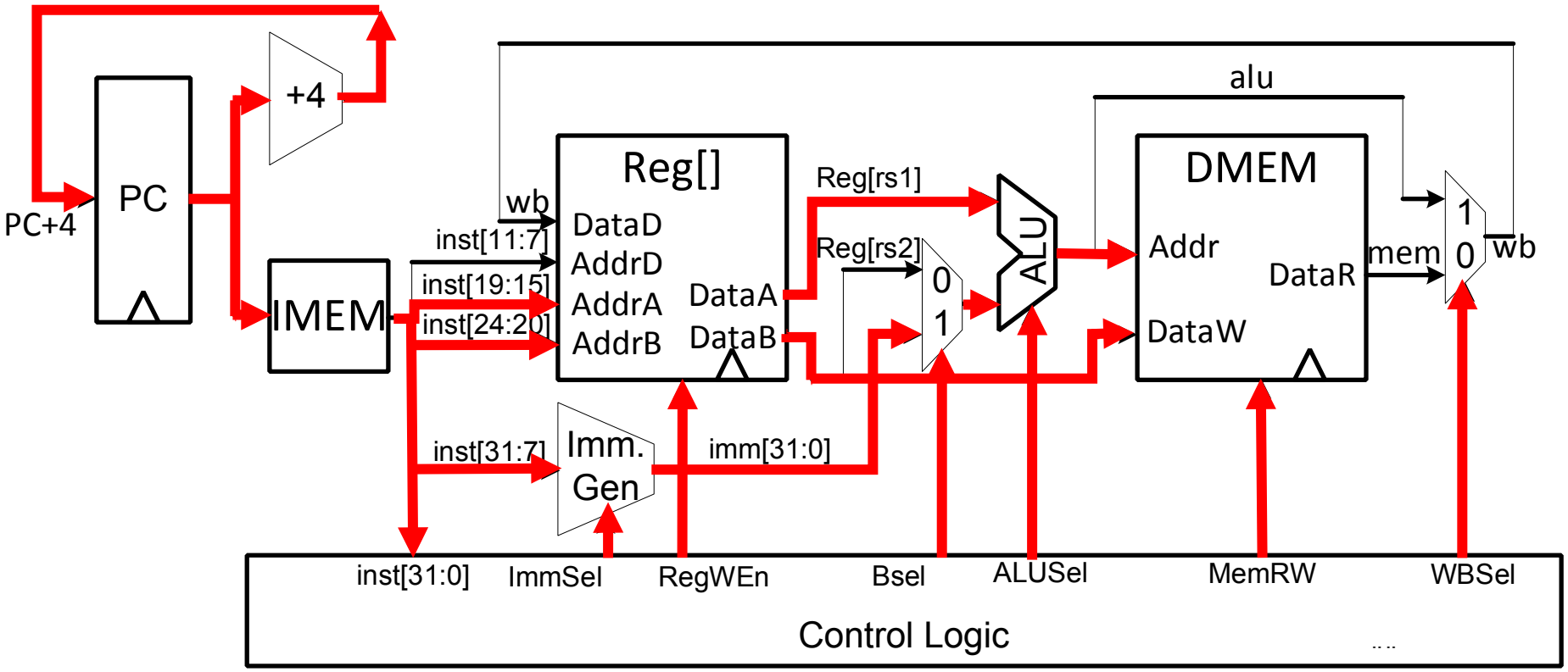
```
sw x14, 8(x2)
```



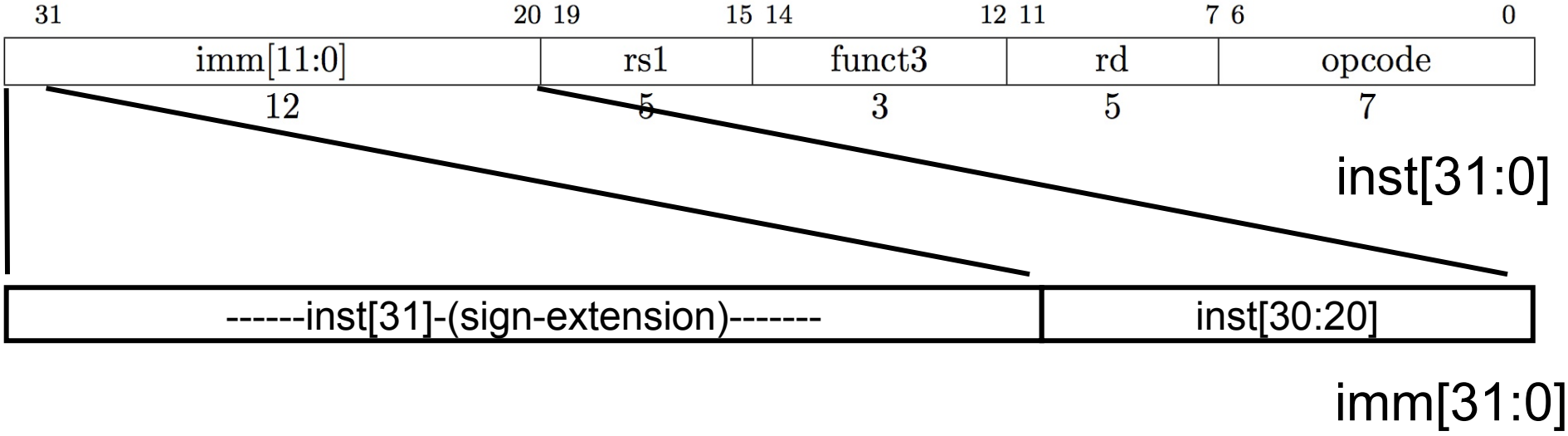
Adding `lw` to datapath

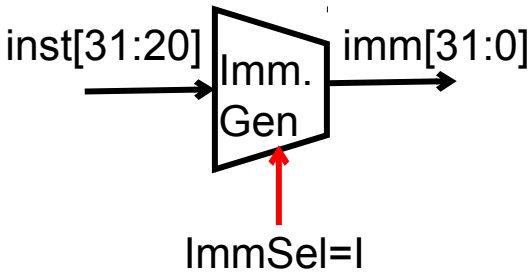


Adding `sw` to datapath



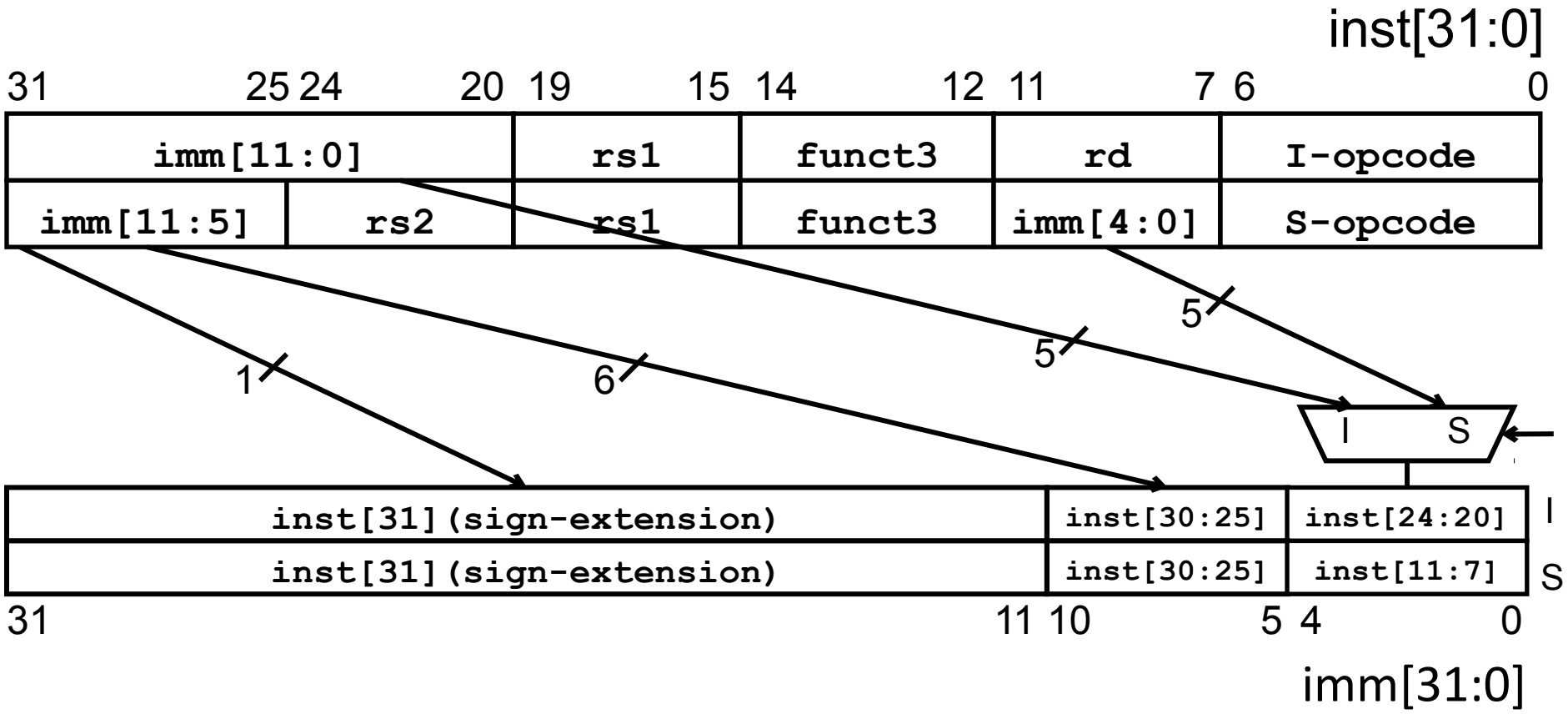
I-Format immediates





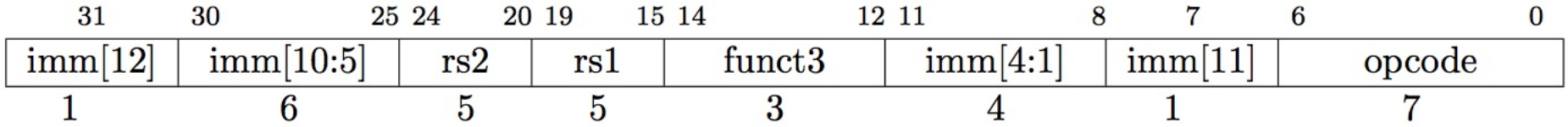
- High 12 bits of instruction (inst[31:20]) copied to low 12 bits of immediate (imm[11:0])
- Immediate is sign-extended by copying value of inst[31] to fill the upper 20 bits of the immediate value (imm[31:12])

I-Format immediates



- Just need a 5-bit mux to select between two positions where low five bits of immediate can reside in instruction
- Other bits in immediate are wired to fixed positions in instruction

Implementing Branches

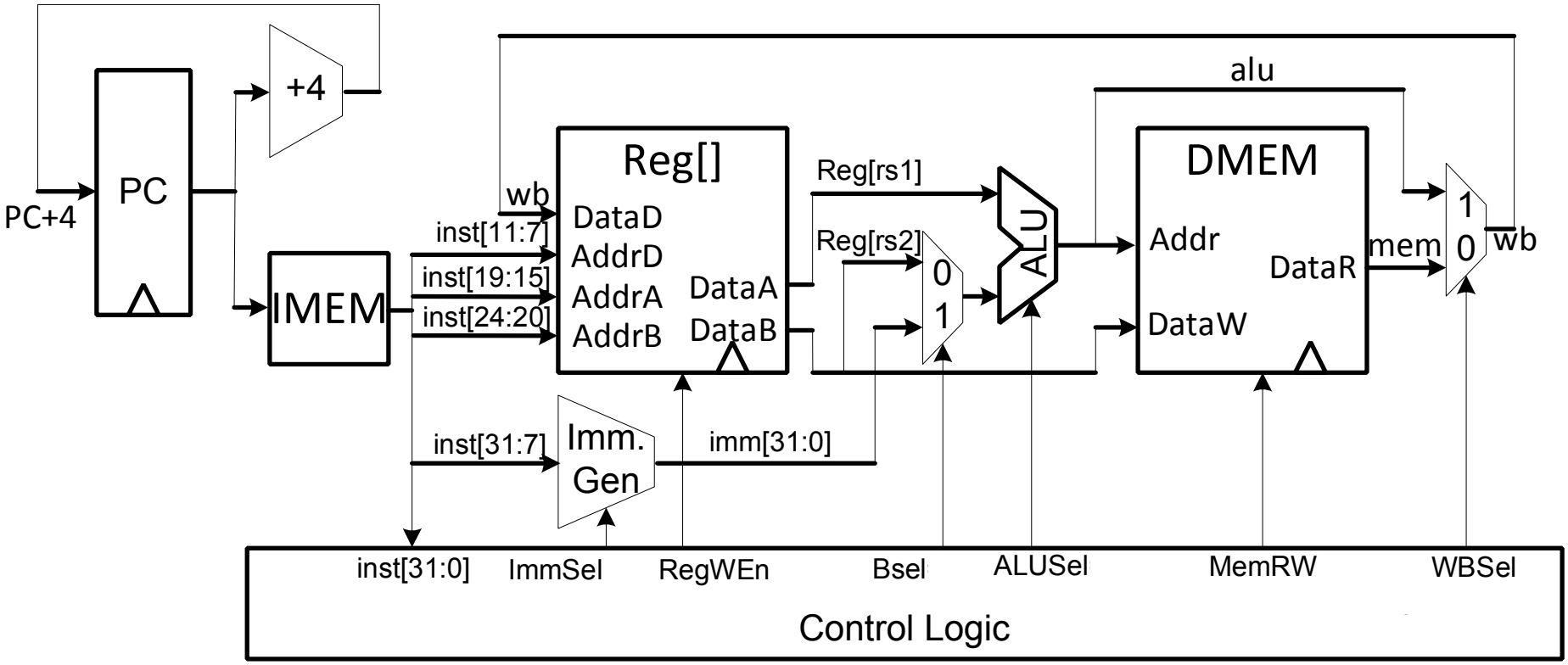


- B-format is mostly same as S-Format, with two register sources (rs1/rs2) and a 12-bit immediate

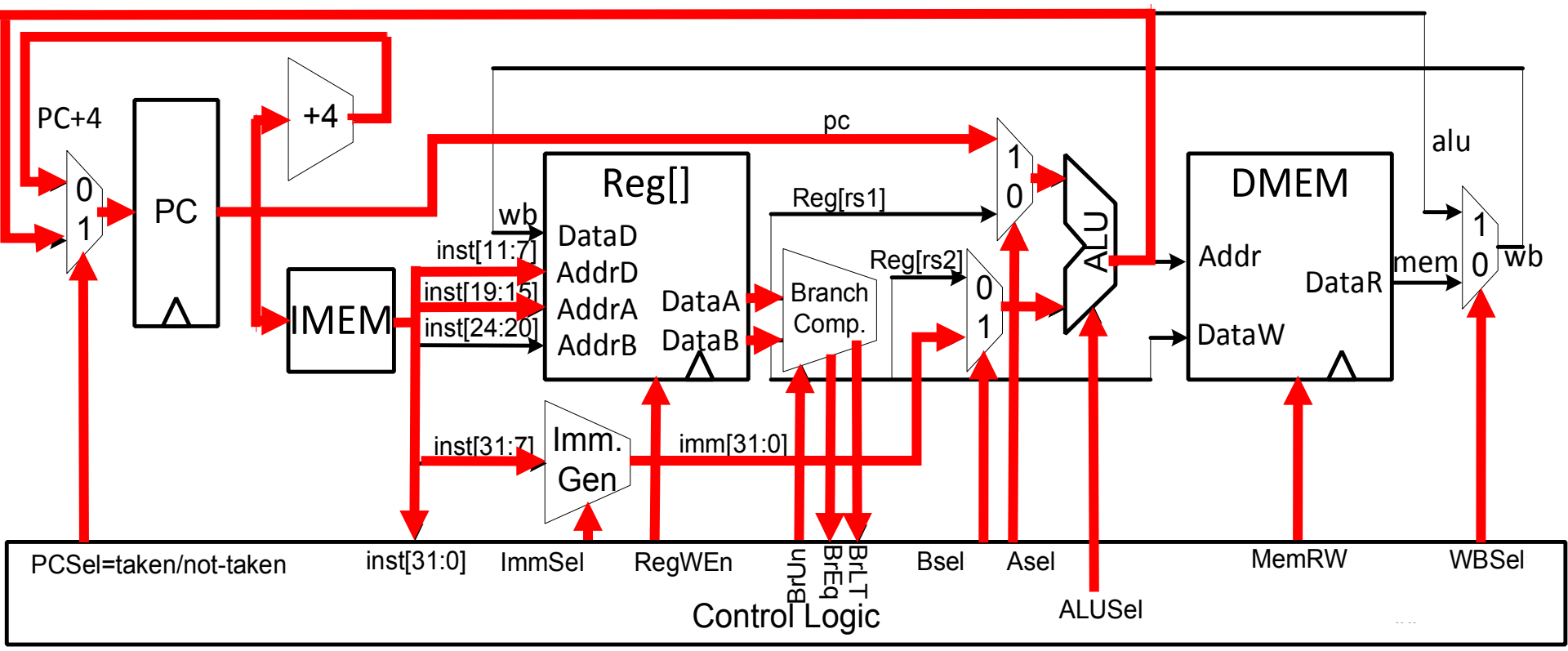
- But now immediate represents values -4096 to +4094 in 2-byte increments

- The 12 immediate bits encode even 13-bit signed byte offsets (lowest bit of offset is always zero, so no need to store it)

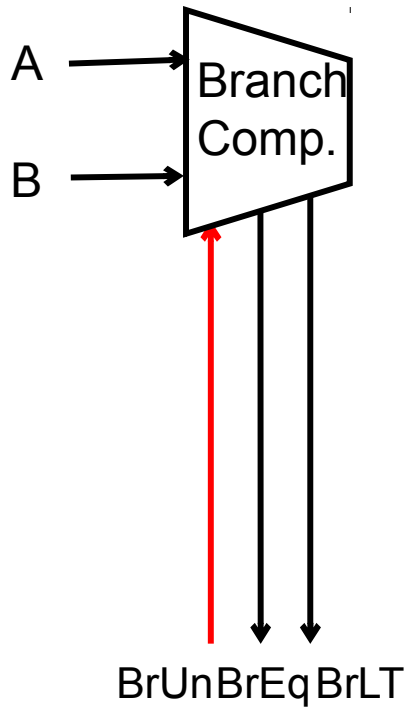
Adding `sw` to datapath



Adding branches to datapath



Branch Comparator

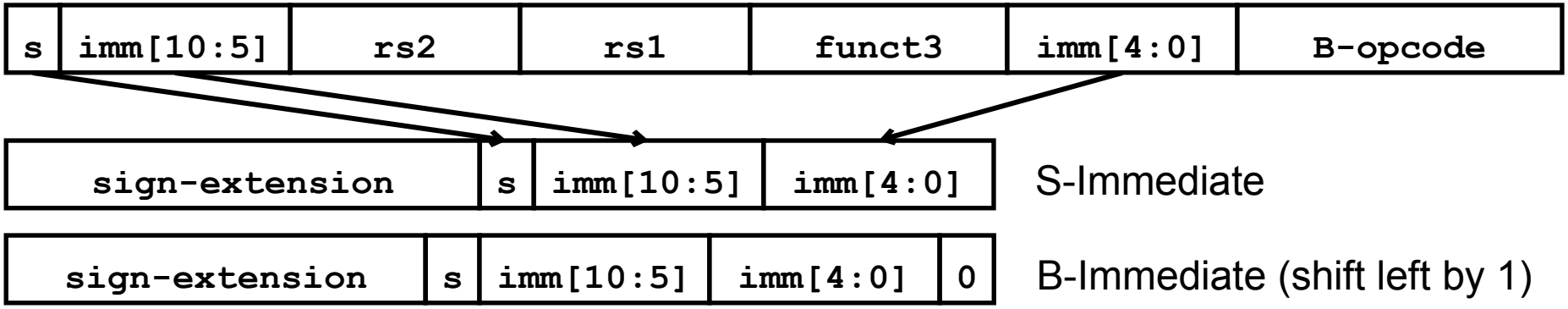


- BrEq = 1, if A=B
- BrLT = 1, if A < B
- BrUn =1 selects unsigned comparison for BrLT, 0=signed
- BGE branch: A >= B, if !(A<B)

Multiply Branch Immediates by Shift?

- 12-bit immediate encodes PC-relative offset of -4096 to +4094 bytes in multiples of 2 bytes

- Standard approach: treat immediate as in range -2048..+2047, then shift left by 1 bit to multiply by 2 for branches

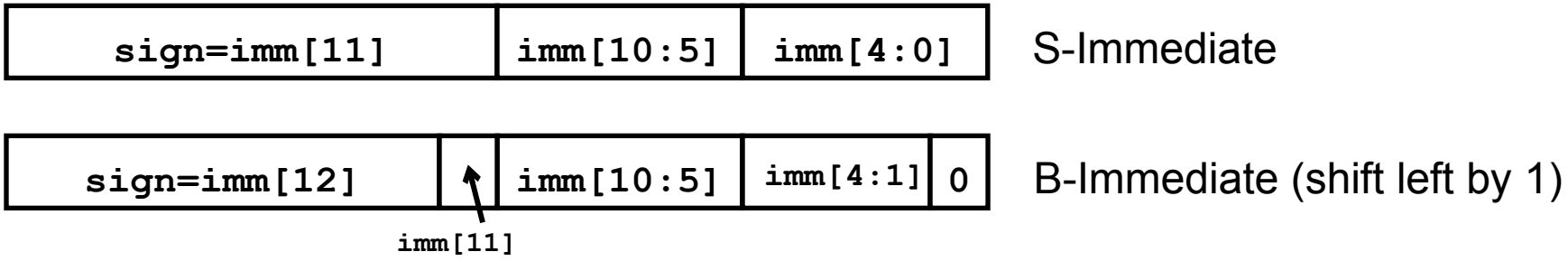


Each instruction immediate bit can appear in one of two places in output immediate value – so need one 2-way mux per bit

RISC-V Branch Immediates

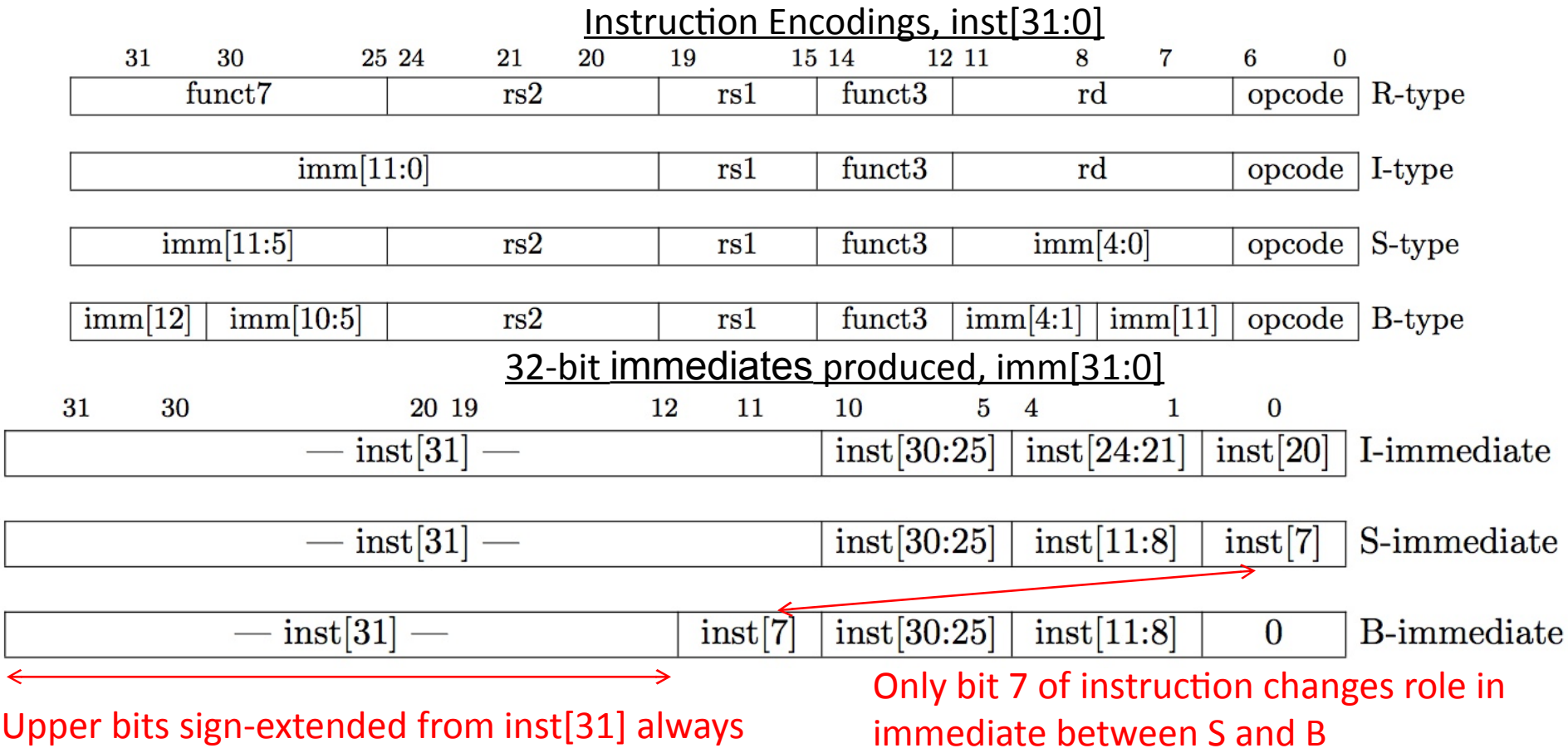
- 12-bit immediate encodes PC-relative offset of -4096 to +4094 bytes in multiples of 2 bytes

- RISC-V approach: keep 11 immediate bits in fixed position in output value, and rotate LSB of S-format to be bit 12 of B-format

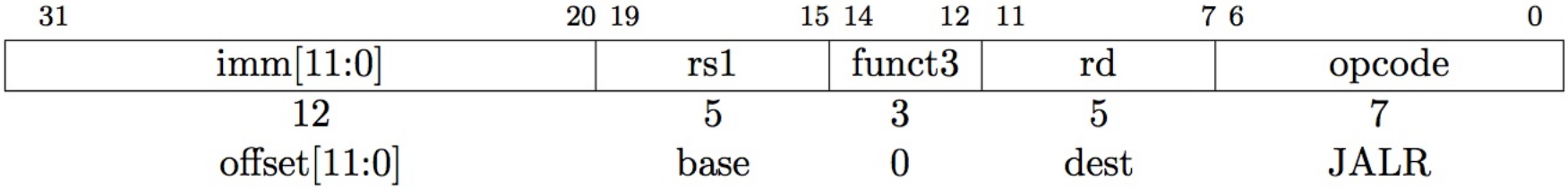


Only one bit changes position between S and B, so only need a single-bit 2-way mux

RISC-V Immediate Encoding

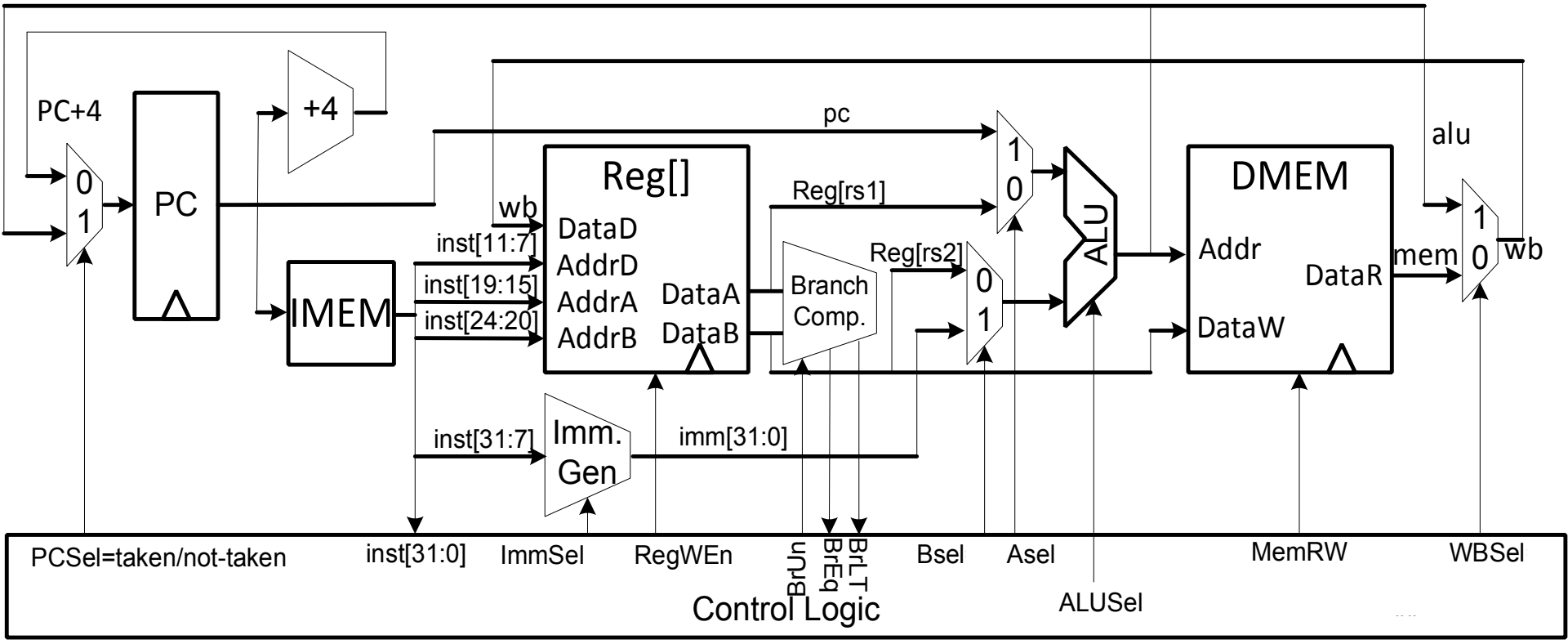


Implementing `JALR` Instruction (I-Format)

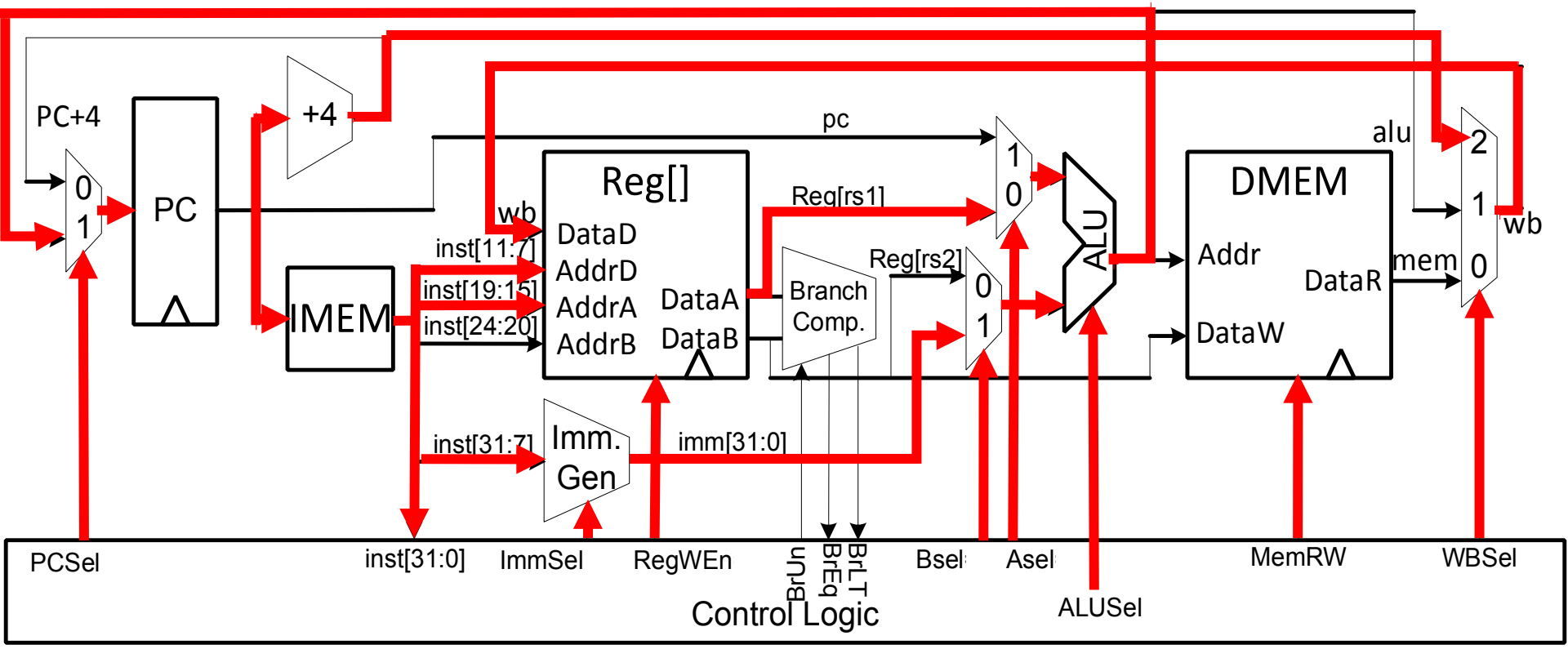


- JALR rd, rs, immediate
  - Writes PC+4 to Reg[rd] (return address)
  - Sets PC = Reg[rs1] + immediate
  - Uses same immediates as arithmetic and loads
    - no multiplication by 2 bytes

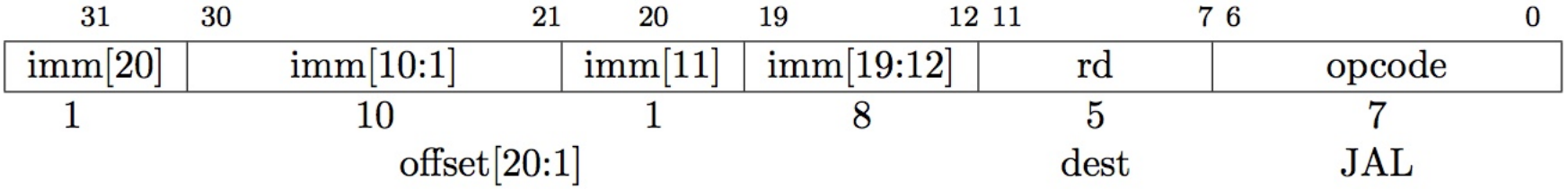
Adding branches to datapath



Adding `jalr` to datapath

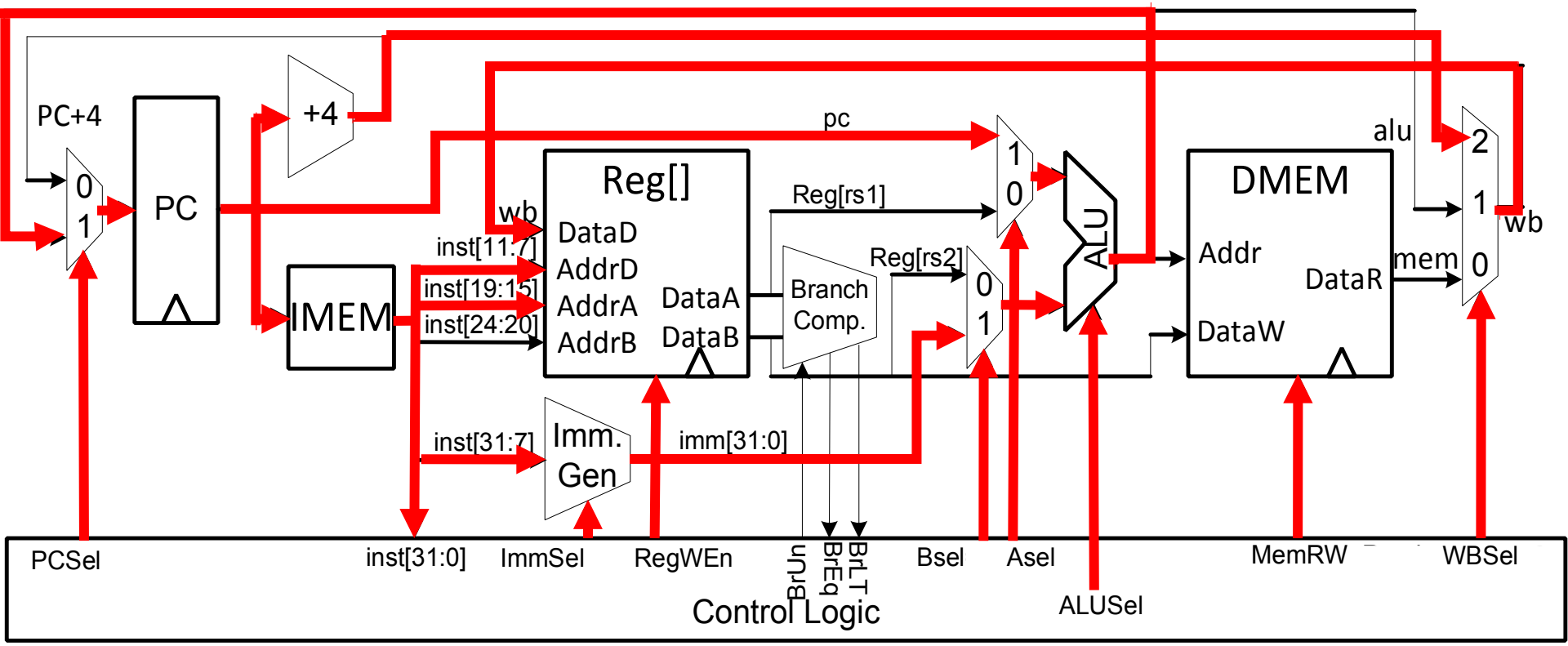


Implementing `jal` Instruction (I-Format)

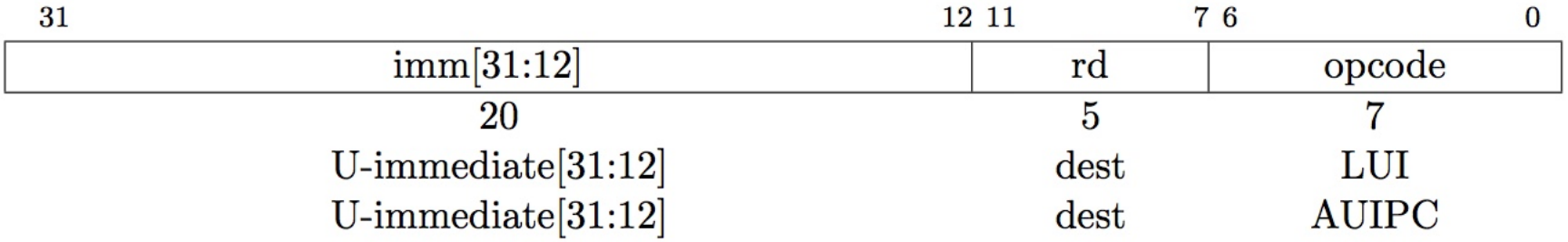


- JAL saves PC+4 in Reg[rd] (the return address)
- Set PC = PC + offset (PC-relative jump)
- Target somewhere within ±2^{19} locations, 2 bytes apart
  - ±2^{18} 32-bit instructions
- Immediate encoding optimized similarly to branch instruction to reduce hardware cost

Adding `jal` to datapath

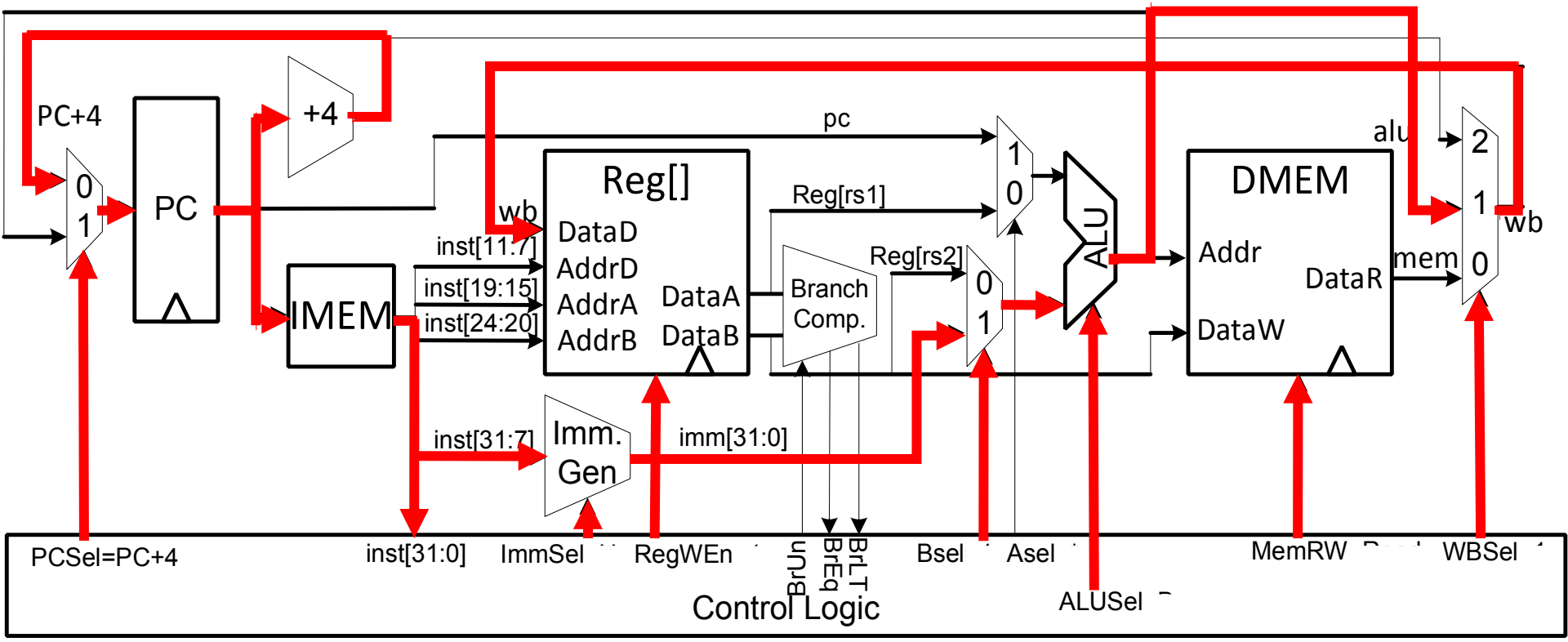


“Upper Immediate” instructions

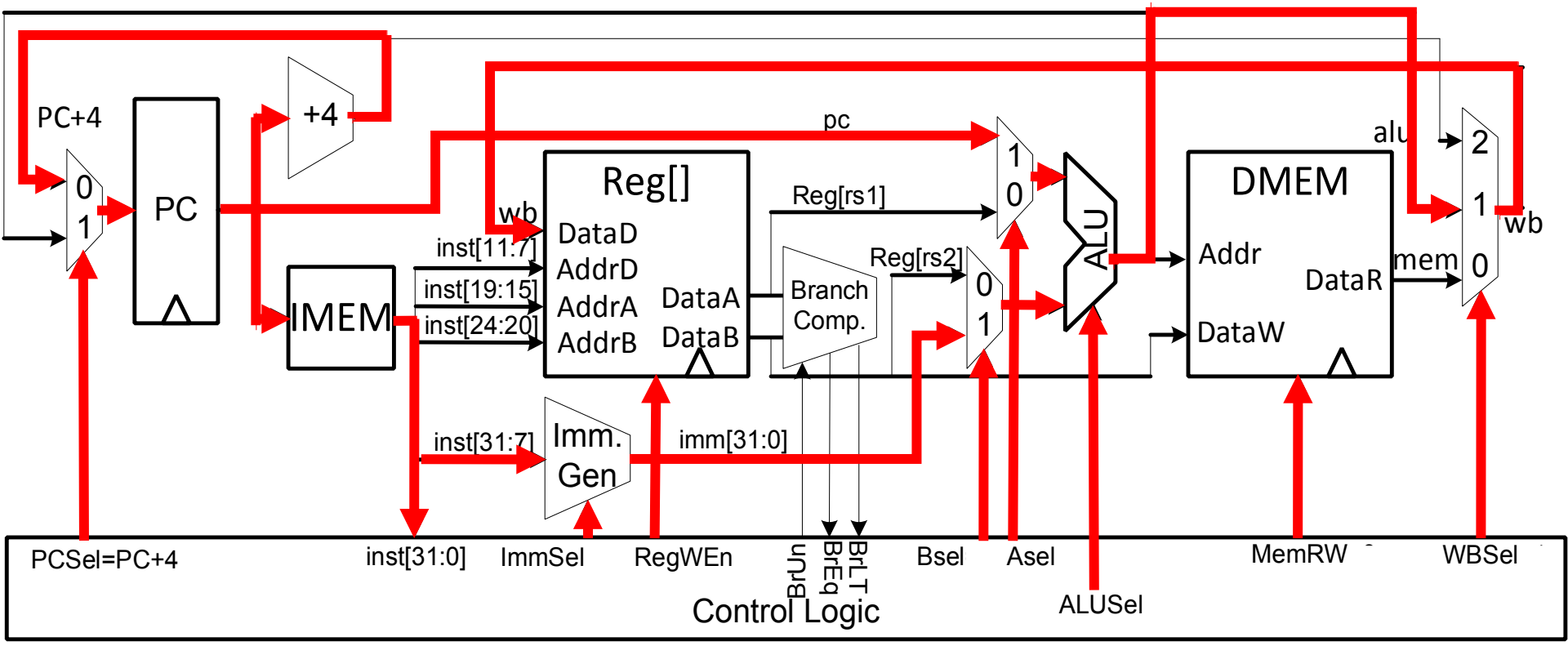


- Has 20-bit immediate in upper 20 bits of 32-bit instruction word
- One destination register, rd
- Used for two instructions
  - LUI – Load Upper Immediate (add to zero)
  - AUIPC – Add Upper Immediate to PC

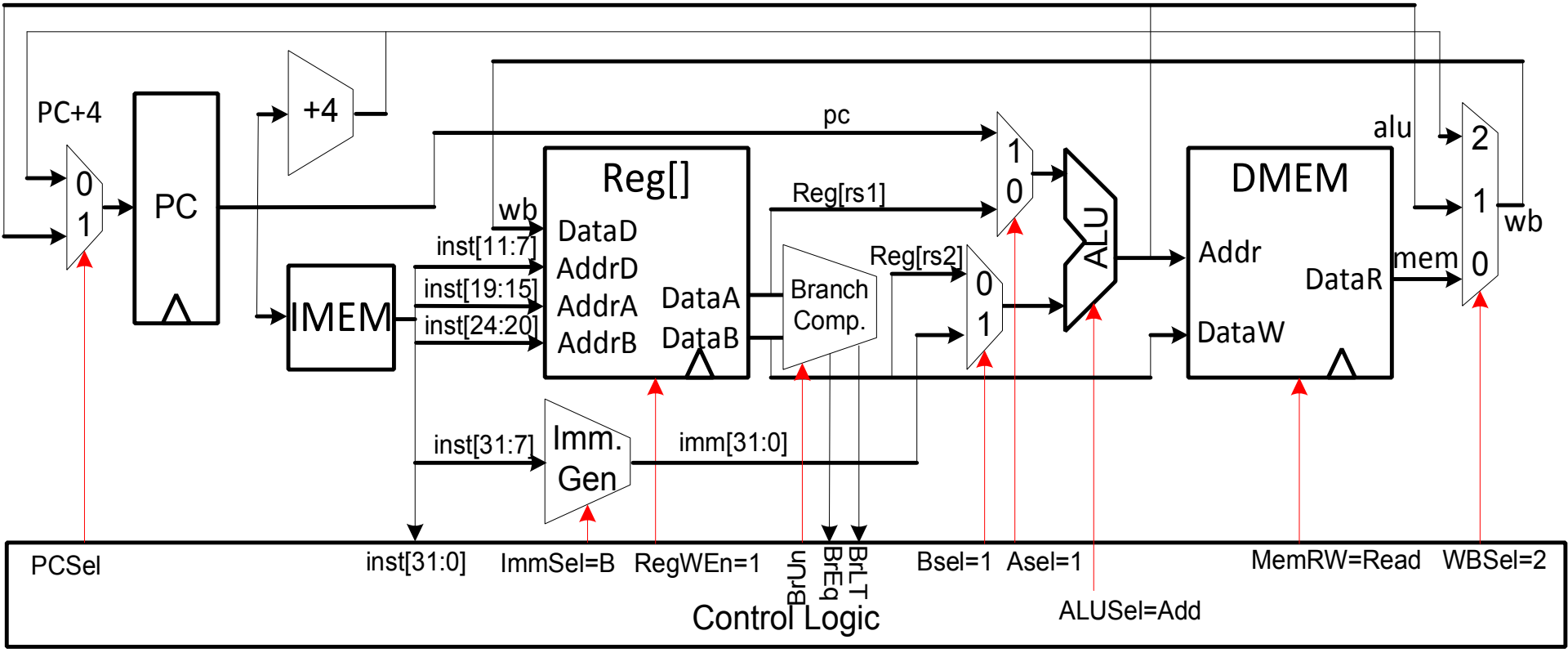
Implementing `lui`



Implementing `auipc`



Single-Cycle RISC-V RV32I Datapath



And in Conclusion, …

- Universal datapath
  - Capable of executing all RISC-V instructions in one cycle each
  - Not all units (hardware) used by all instructions
- 5 Phases of execution
  - IF, ID, EX, MEM, WB
  - Not all instructions are active in all phases
- Controller specifies how to execute instructions
  - what new instructions can be added with just most control?