

RISC-V Processor Datapath

Recap: Complete RV32I ISA

imm[31:12]					rd	0110111
imm[31:12]					rd	0010111
imm[20:10:11:19:12]					rd	1101111
imm[11:0]					rs1	000
imm[12:10:5]		rs2	rs1	000	imm[4:1:11]	1100011
imm[12:10:5]		rs2	rs1	001	imm[4:1:11]	1100011
imm[12:10:5]		rs2	rs1	100	imm[4:1:11]	1100011
imm[12:10:5]		rs2	rs1	101	imm[4:1:11]	1100011
imm[12:10:5]		rs2	rs1	110	imm[4:1:11]	1100011
imm[12:10:5]		rs2	rs1	111	imm[4:1:11]	1100011
imm[11:0]					rs1	000
imm[11:0]					rs1	001
imm[11:0]					rs1	010
imm[11:0]					rs1	100
imm[11:0]					rs1	101
imm[11:5]		rs2	rs1	000	imm[4:0]	0100011
imm[11:5]		rs2	rs1	001	imm[4:0]	0100011
imm[11:5]		rs2	rs1	010	imm[4:0]	0100011
imm[11:0]					rs1	000
imm[11:0]					rs1	010
imm[11:0]					rs1	011
imm[11:0]					rs1	100
imm[11:0]					rs1	110
imm[11:0]					rs1	111

LUI
 AUIPC
 JAL
 JALR
 BEQ
 BNE
 BLT
 BGE
 BLTU
 BGEU
 LB
 LH
 LW
 LBU
 LHU
 SB
 SH
 SW
 ADDI
 SLTI
 SLTIU
 XORI
 ORI
 ANDI

0000000	shamt	rs1	001	rd	0010011
0000000	shamt	rs1	101	rd	0010011
0100000	shamt	rs1	101	rd	0010011
0000000	rs2	rs1	000	rd	0110011
0100000	rs2	rs1	000	rd	0110011
0000000	rs2	rs1	001	rd	0110011
0000000	rs2	rs1	010	rd	0110011
0000000	rs2	rs1	011	rd	0110011
0000000	rs2	rs1	100	rd	0110011
0000000	rs2	rs1	101	rd	0110011
0100000	rs2	rs1	101	rd	0110011
0000000	rs2	rs1	110	rd	0110011
0000000	rs2	rs1	111	rd	0110011

SLLI
 SRLI
 SRAI
 ADD
 SUB
 SLL
 SLT
 SLTU
 XOR
 SRL
 SRA
 OR
 AND
 FENCE
 FENCE.I
 ECALL
 EBREAK
 CSRRW
 CSRRS
 CSRRC
 CSRRWI
 CSRRSI
 CSRRCI

0000	pred	succ	00000	000	00000	0001111
0000	0000	0000	00000	001	00000	0001111
0000000000000			00000	000	00000	1110011
0000000000001			00000	000	00000	1110011
csr			rs1	001	rd	1110011
csr			rs1	010	rd	1110011
csr			rs1	011	rd	1110011
csr			zimm	101	rd	1110011
csr			zimm	110	rd	1110011
csr			zimm	111	rd	1110011

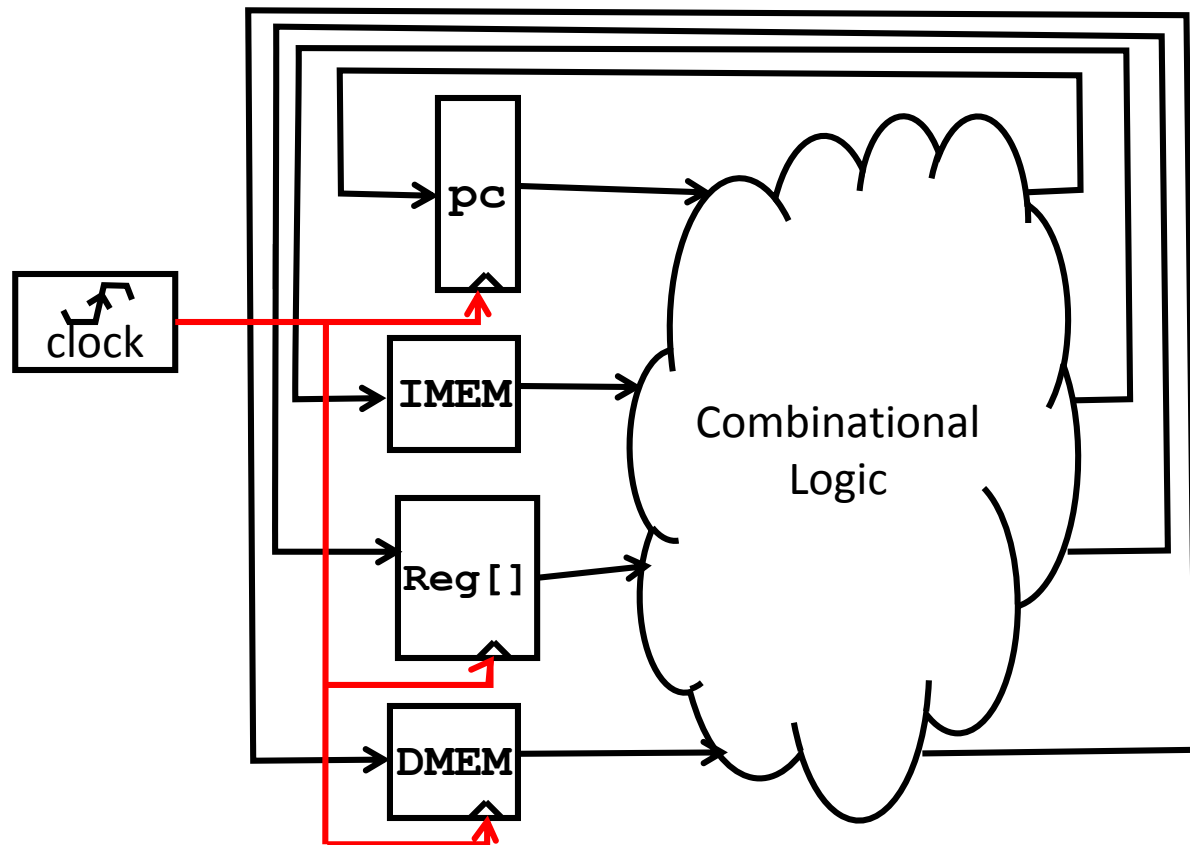
Not in this course

State Required by RV32I ISA

Each instruction reads and updates this state during execution:

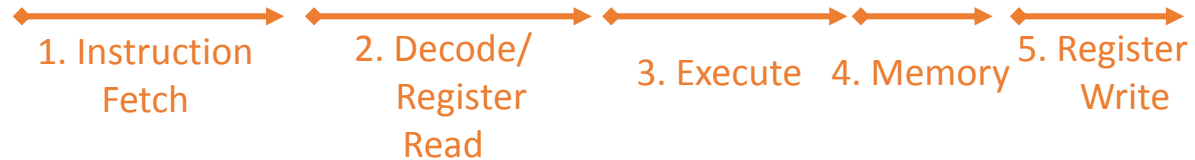
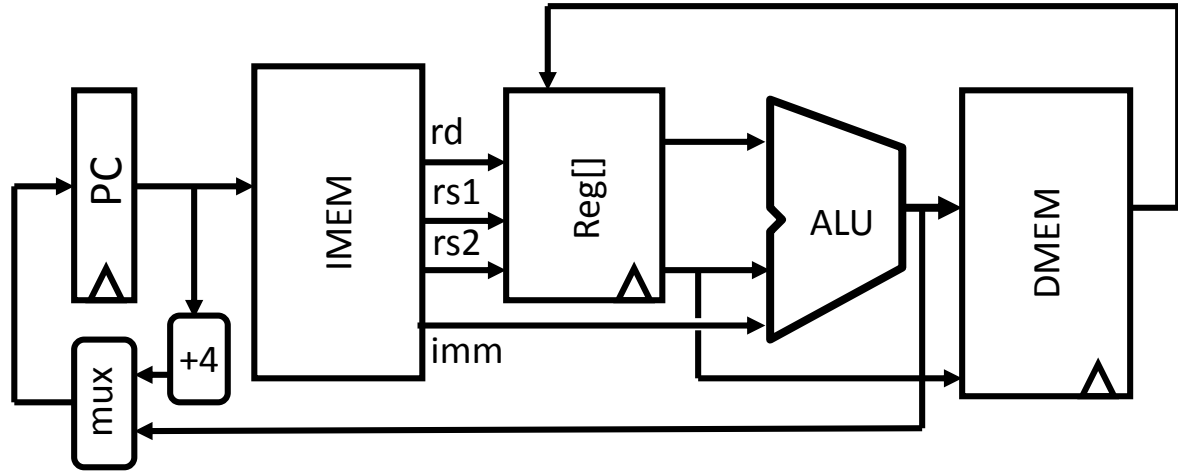
- **Registers (**x0** . . **x31**)**
 - Register file (or *regfile*) **Reg** holds **32 registers x 32 bits**/register: **Reg[0]** . . **Reg[31]**
 - First register read specified by *rs1* field in instruction
 - Second register read specified by *rs2* field in instruction
 - Write register (destination) 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 settle 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



Clock

time

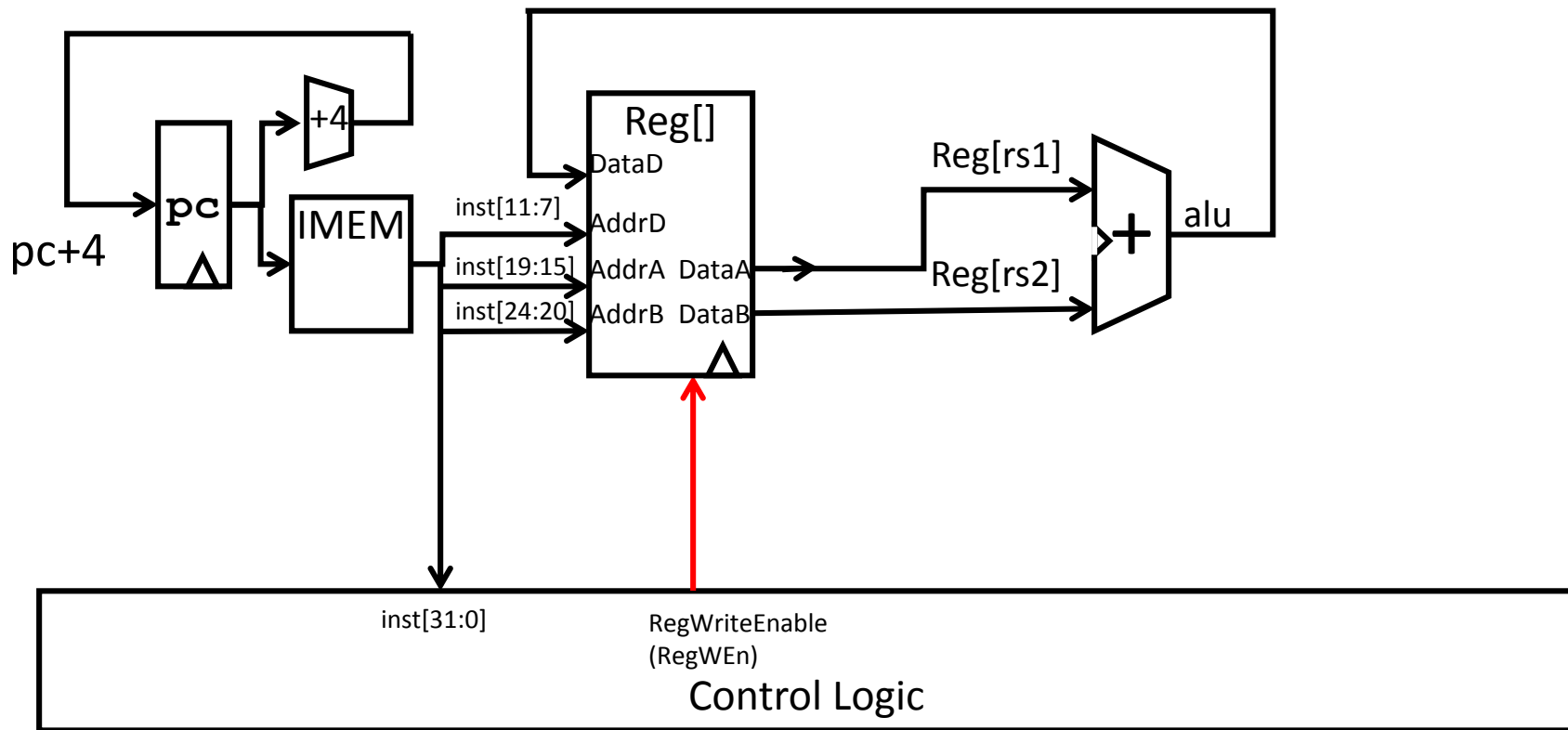
Implementing the **add** instruction

0000000	rs2	rs1	000	rd	0110011	ADD
---------	-----	-----	-----	----	---------	-----

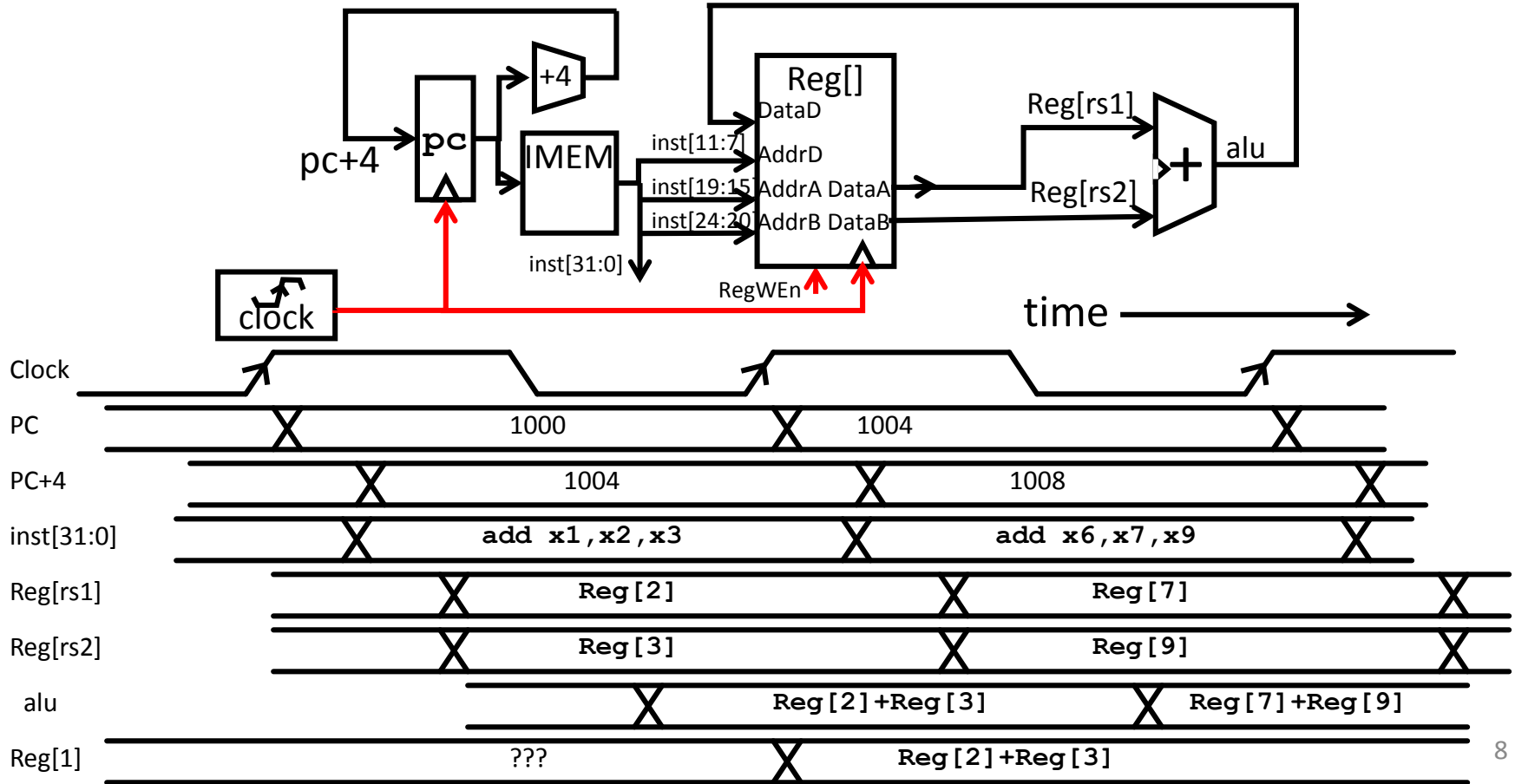
add rd, rs1, rs2

- Instruction makes two changes to machine's state:
 - **Reg[rd] = Reg[rs1] + Reg[rs2]**
 - **PC = PC + 4**

Datapath for **add**



Timing Diagram for **add**



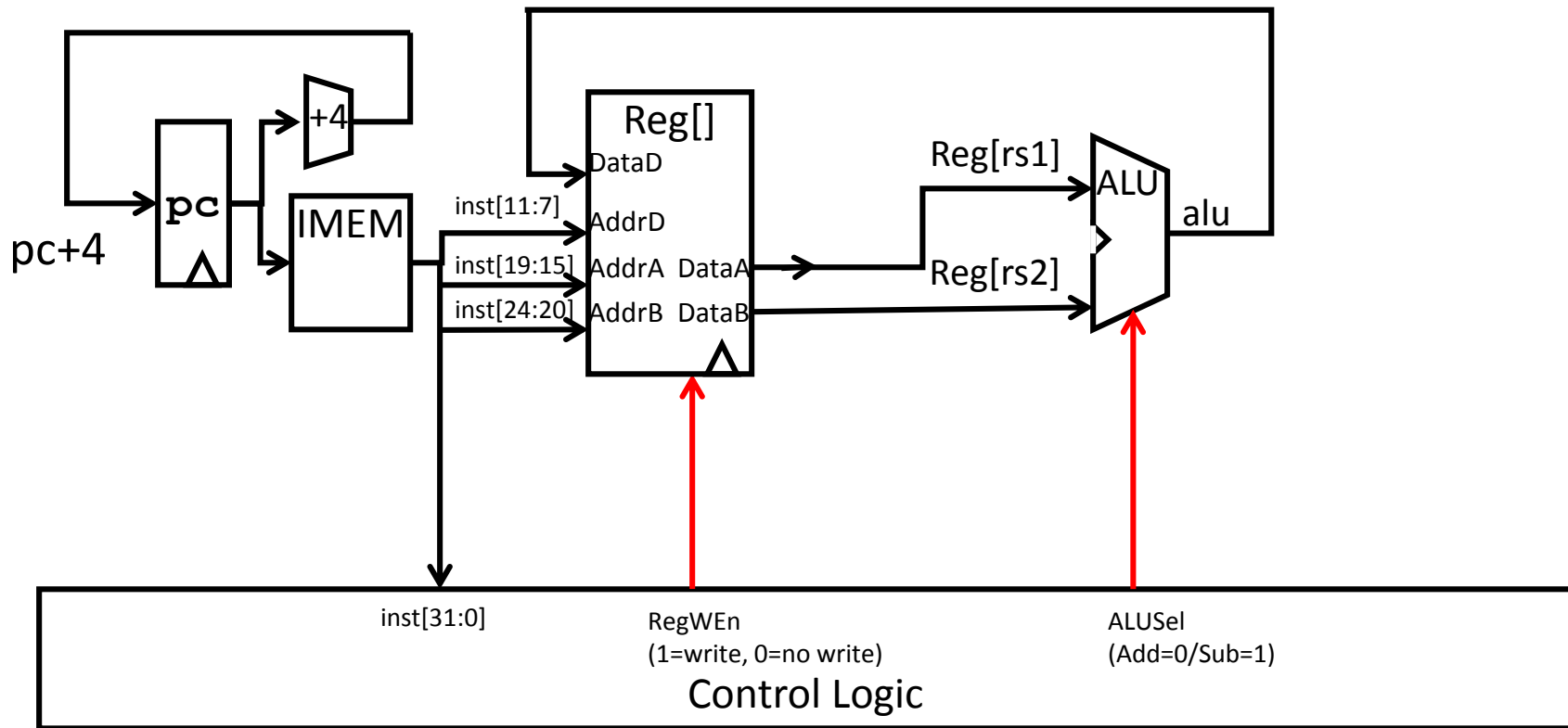
Implementing the **sub** instruction

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

sub rd, rs1, rs2

- 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

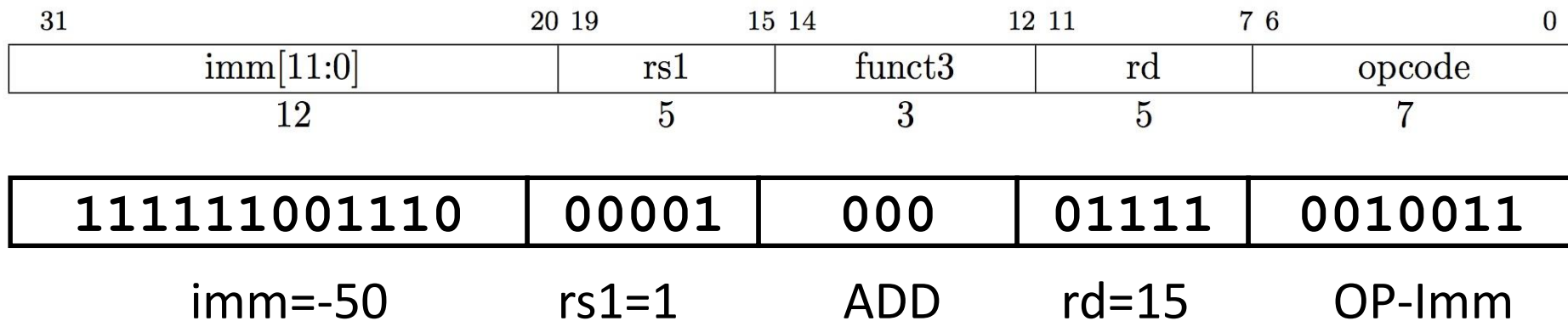
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110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

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

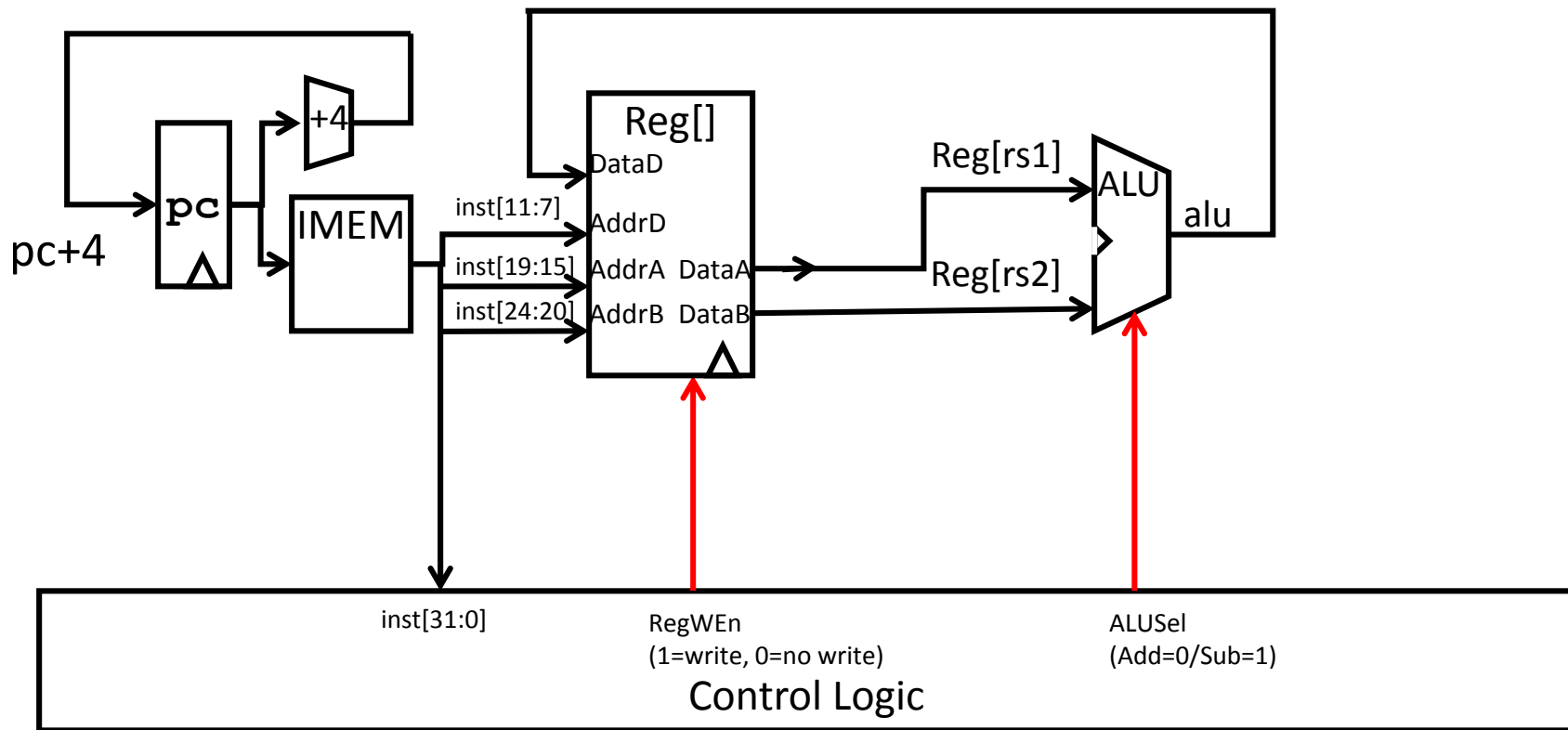
Implementing the **addi** instruction

- RISC-V Assembly Instruction:

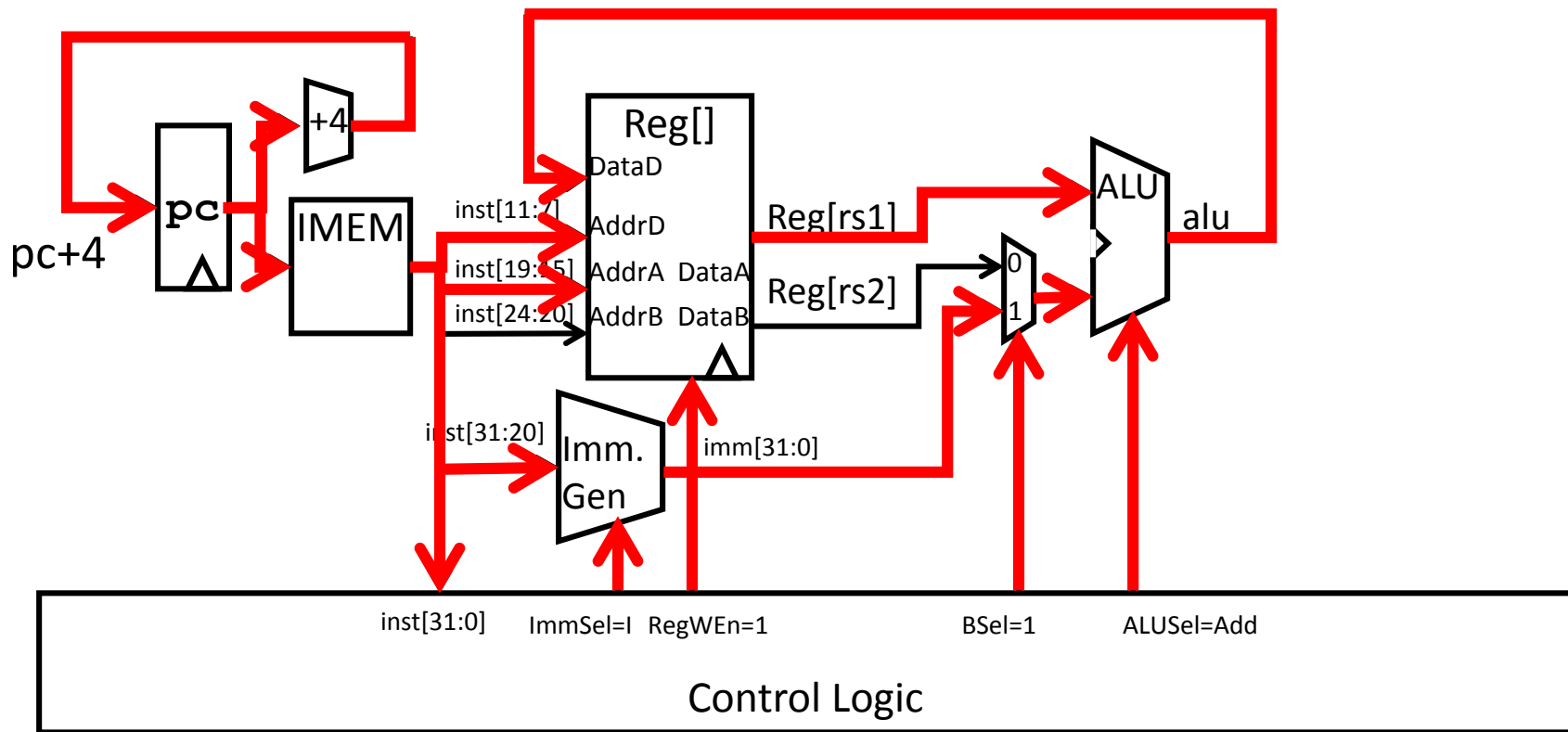
addi x15, x1, -50



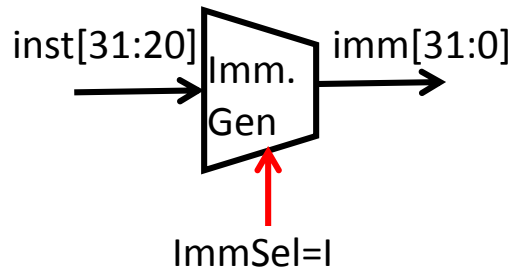
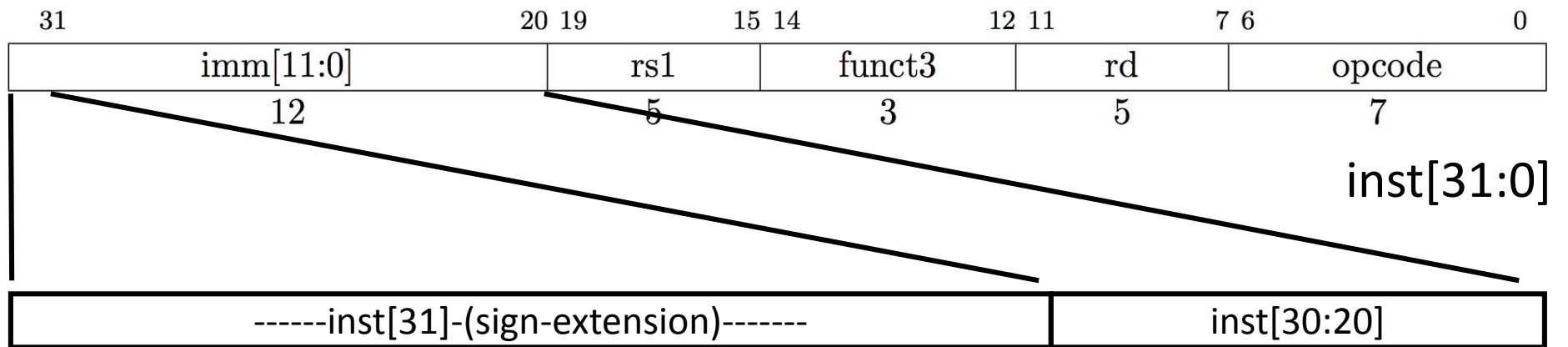
Datapath for **add/sub**



Adding **addi** to datapath

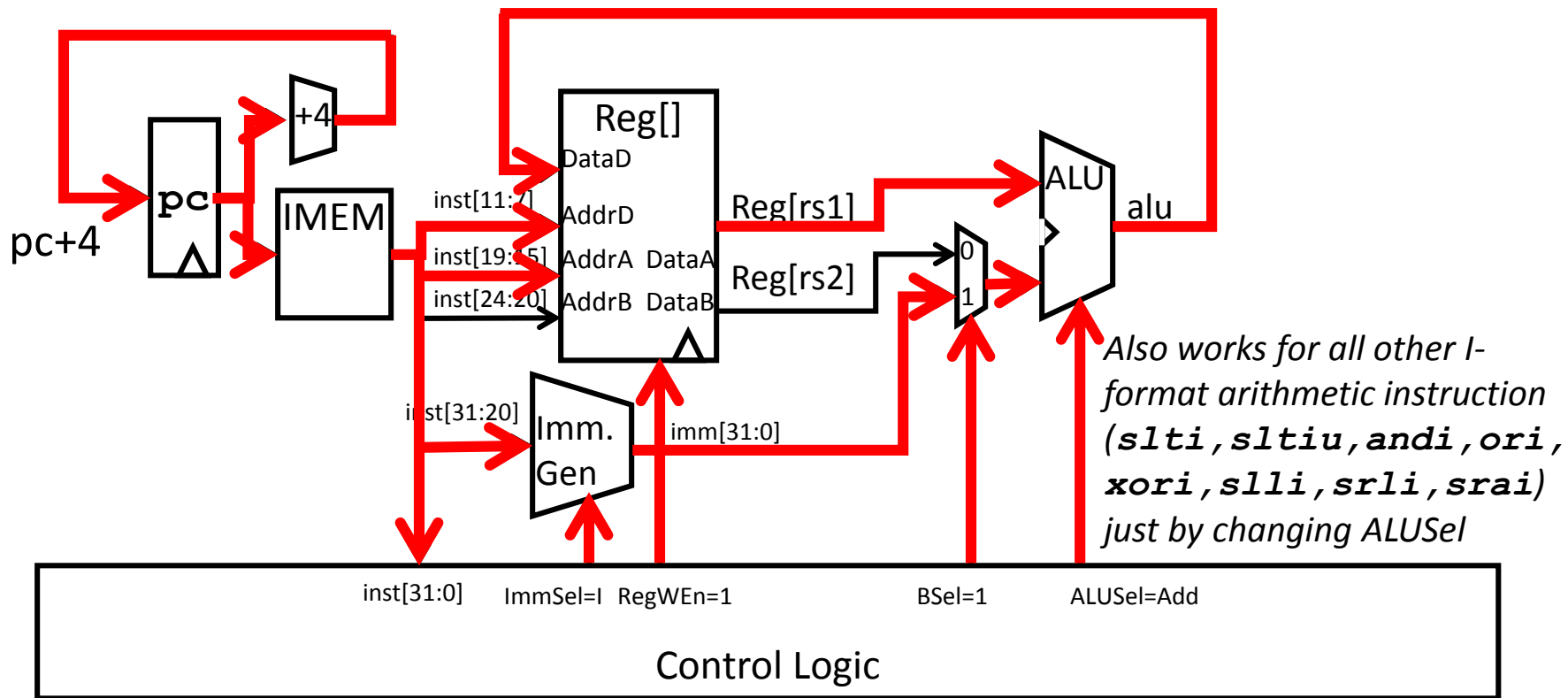


I-Format immediates



- 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])

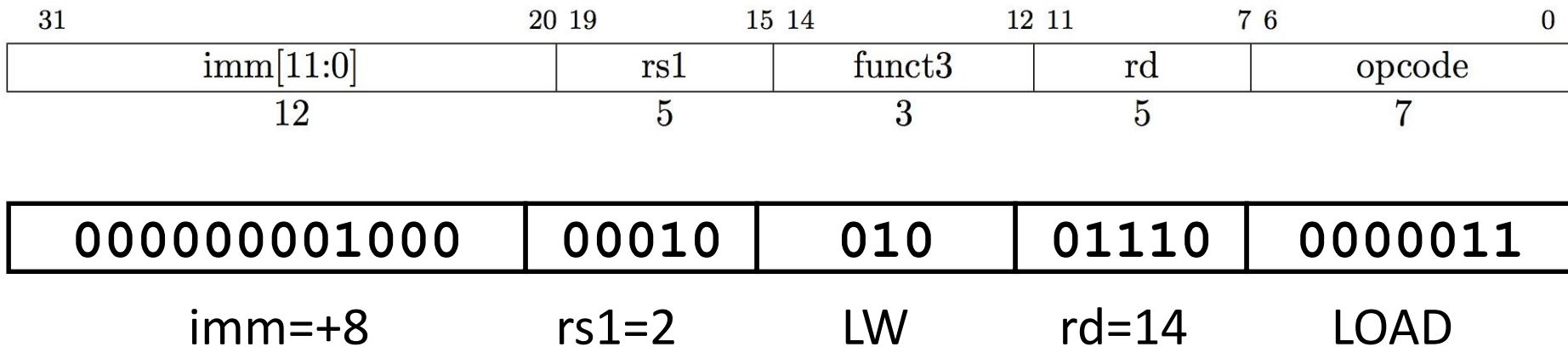
Adding **addi** to datapath



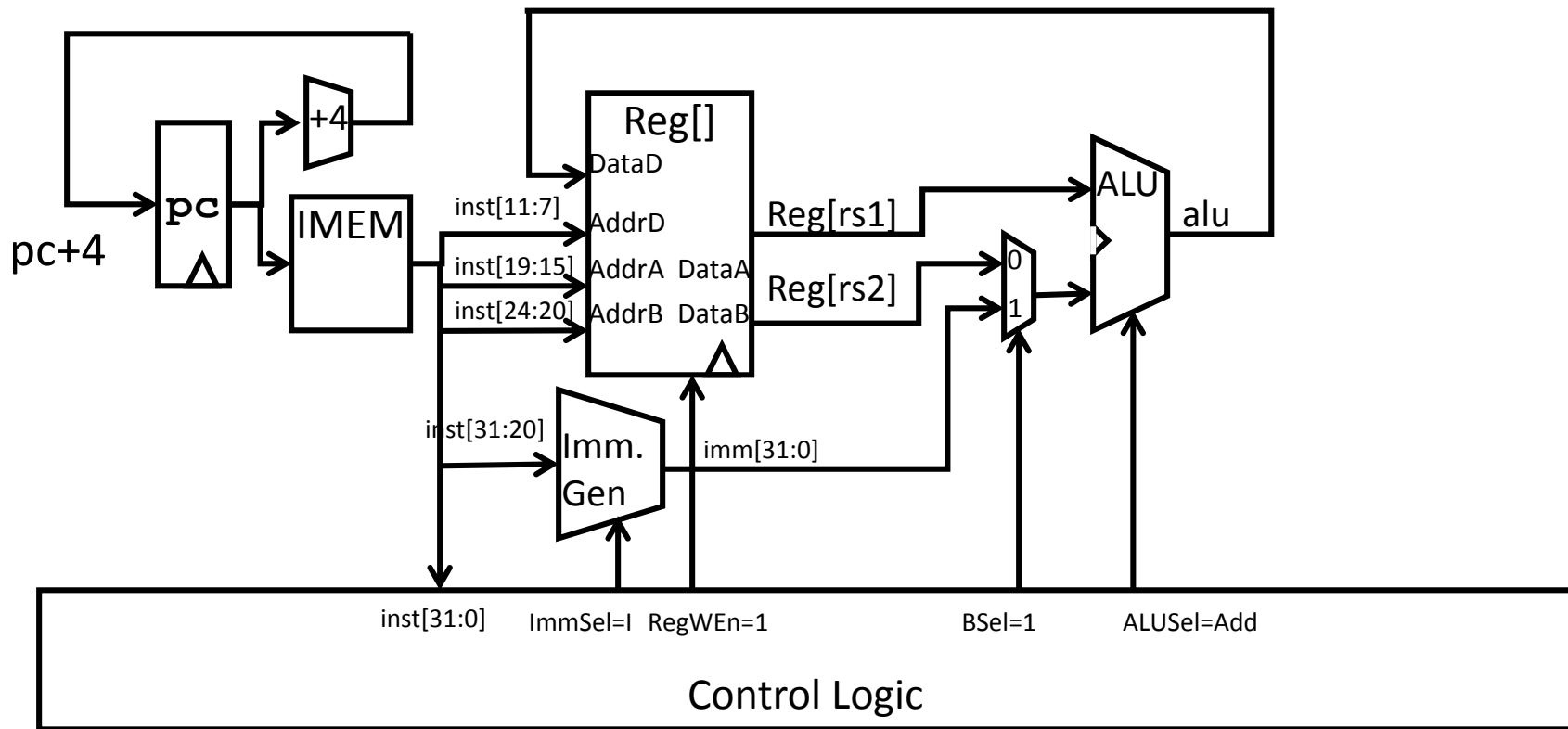
Implementing Load Word instruction

- RISC-V Assembly Instruction:

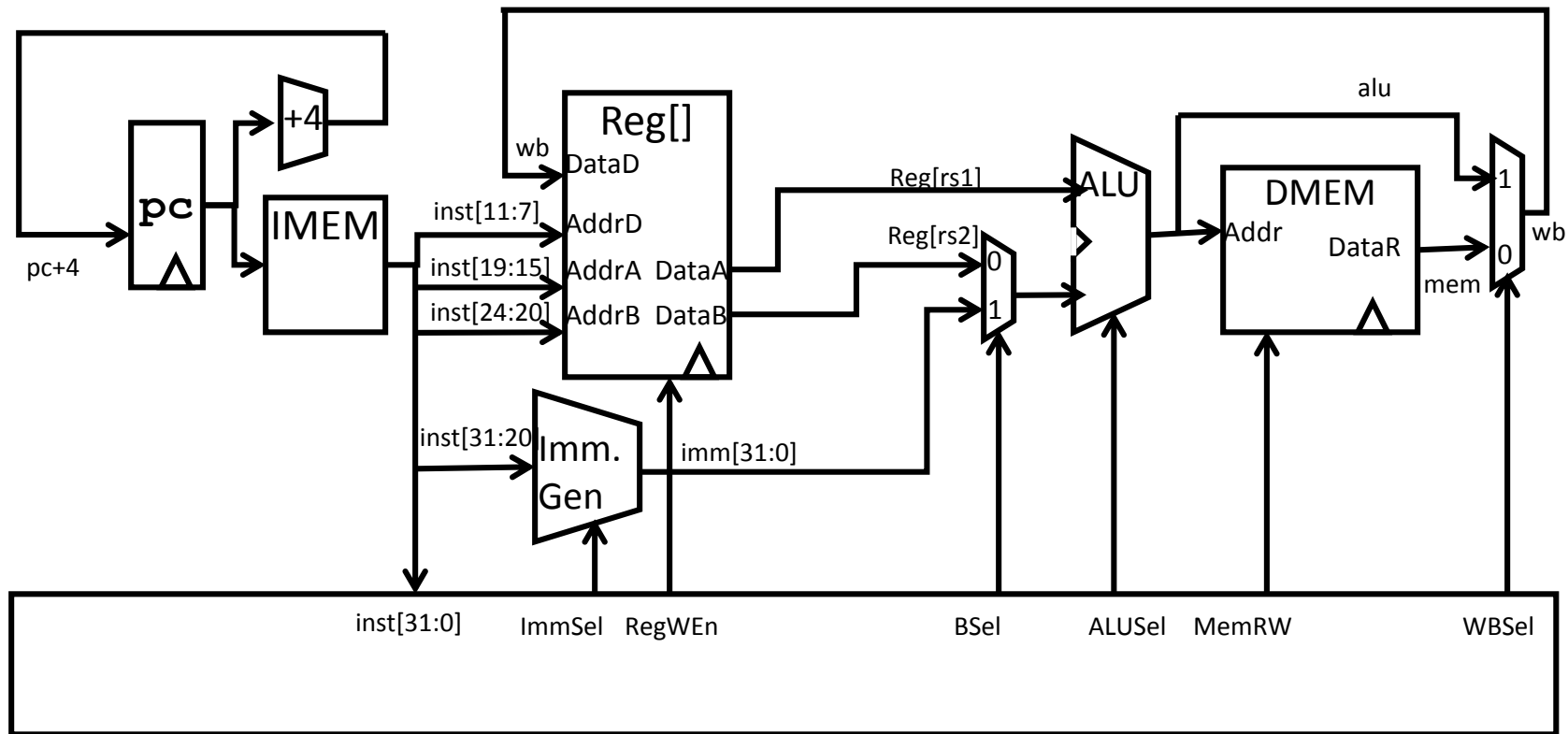
lw x14, 8(x2)



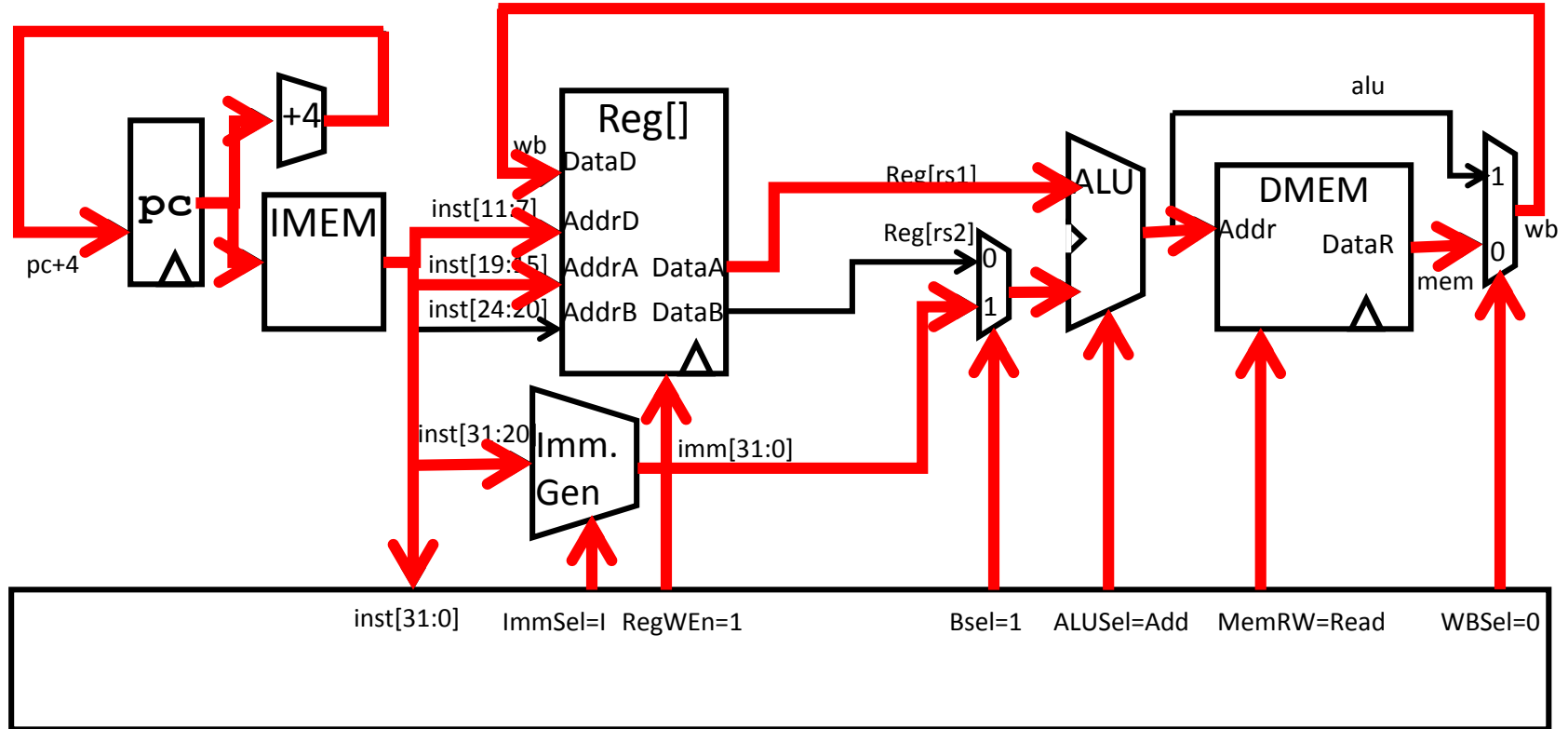
Adding **addi** to datapath



Adding **lw** 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

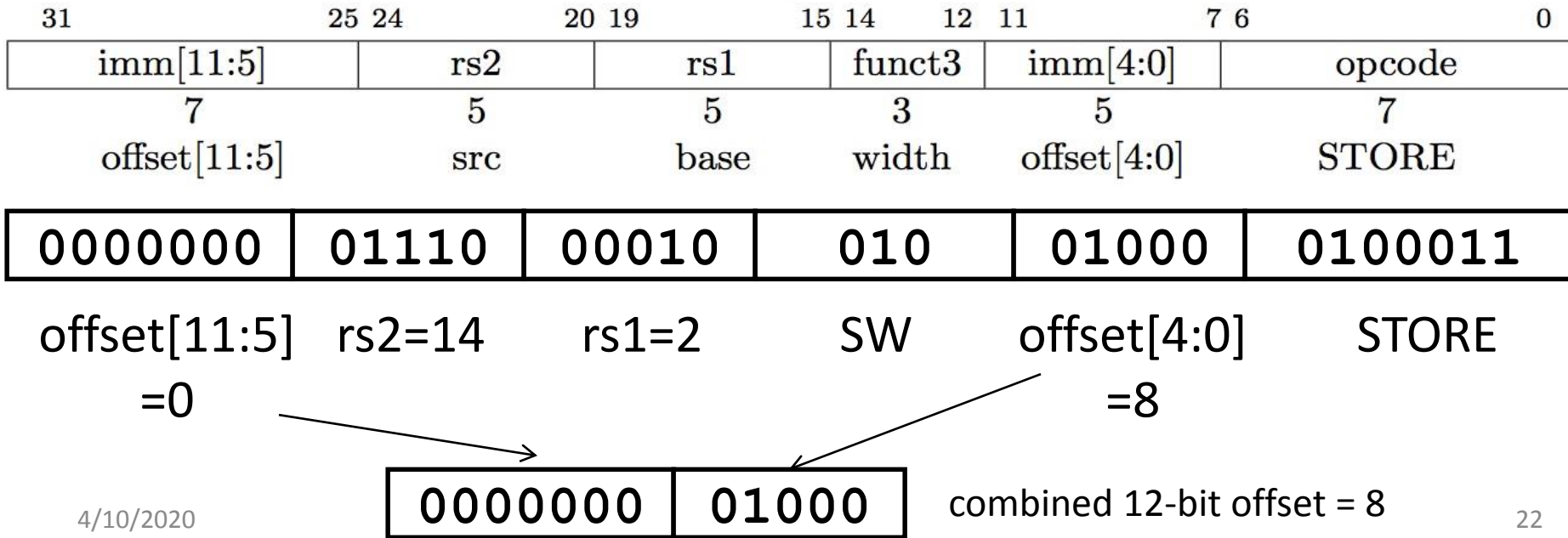
↑
funct3 field encodes size and
signedness of load data

- 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.

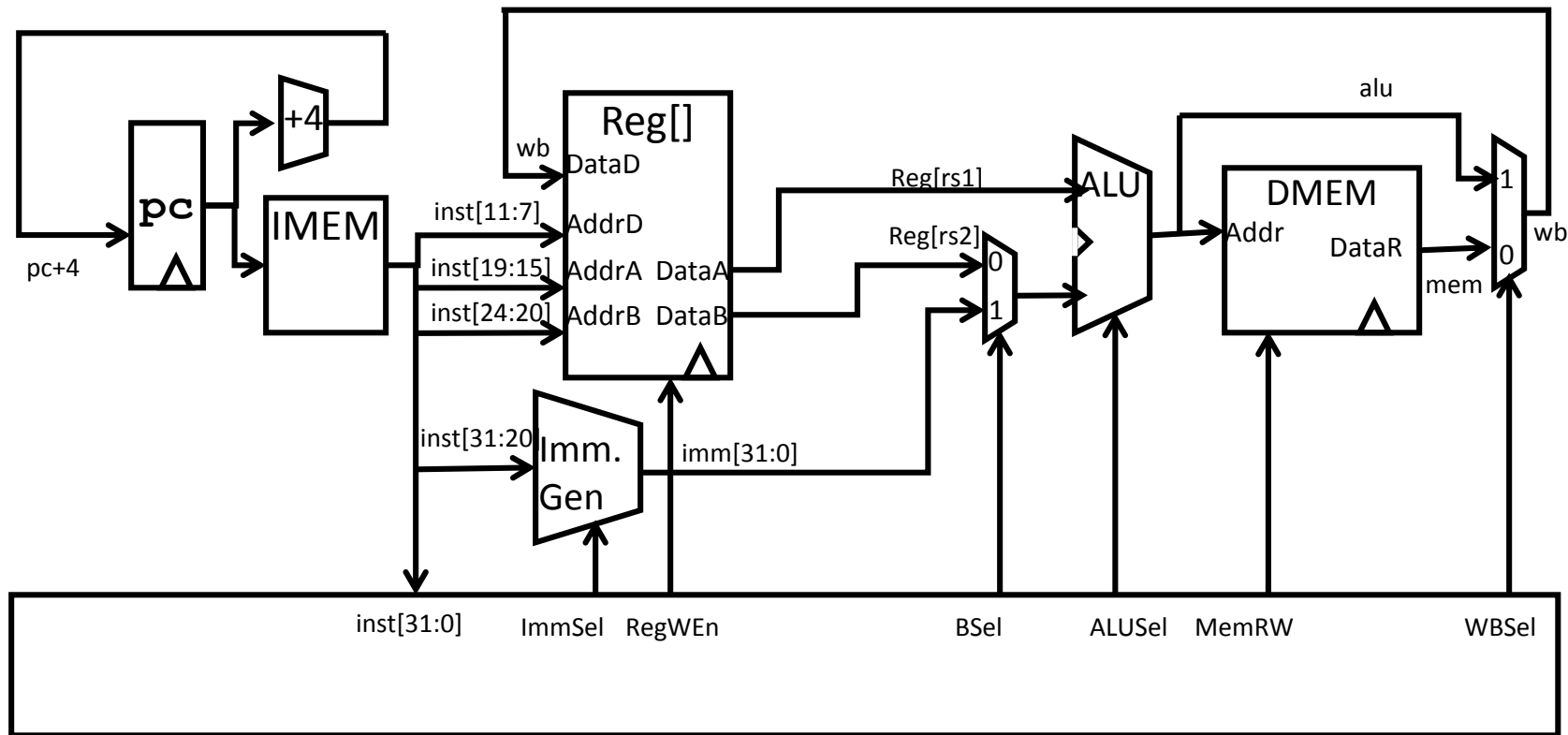
Implementing Store Word instruction

- RISC-V Assembly Instruction:

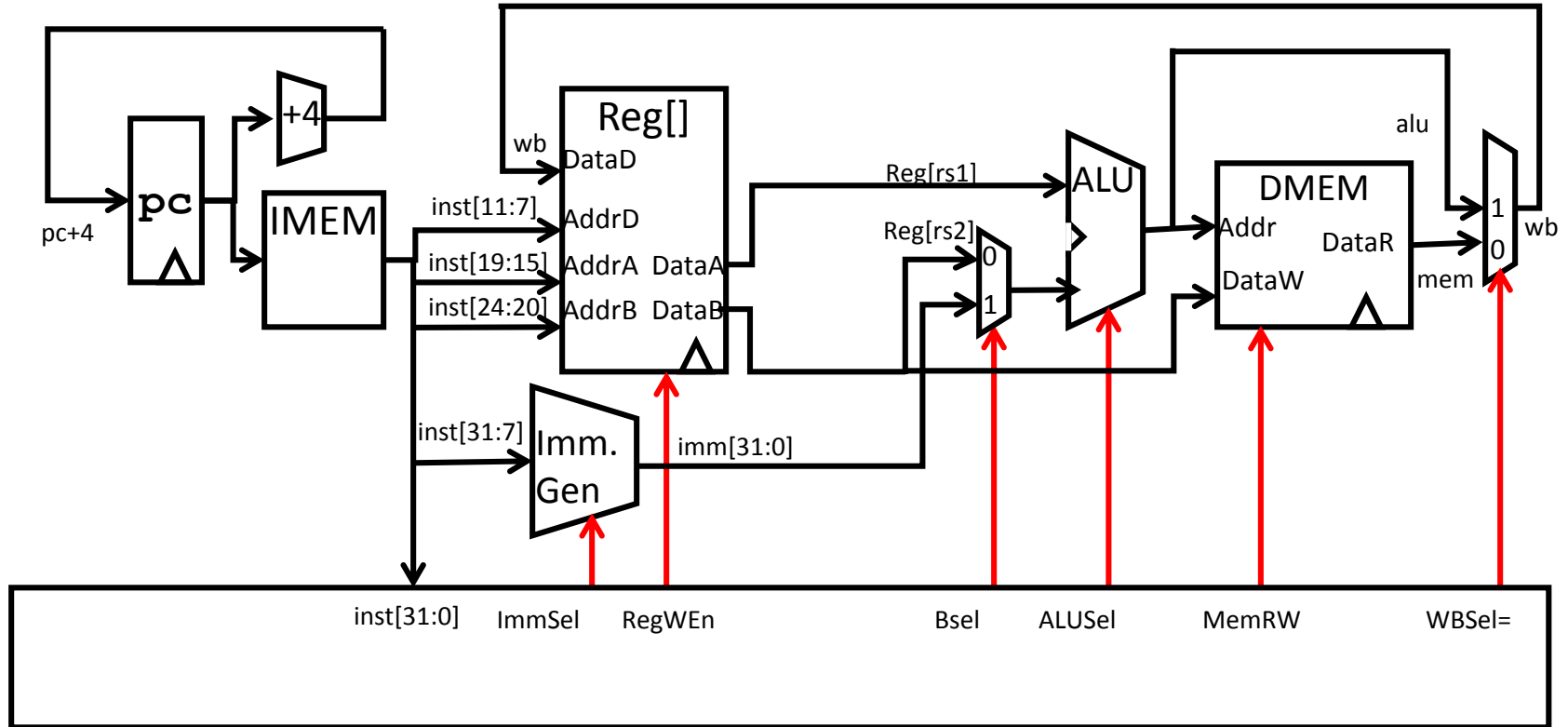
sw x14, 8(x2)



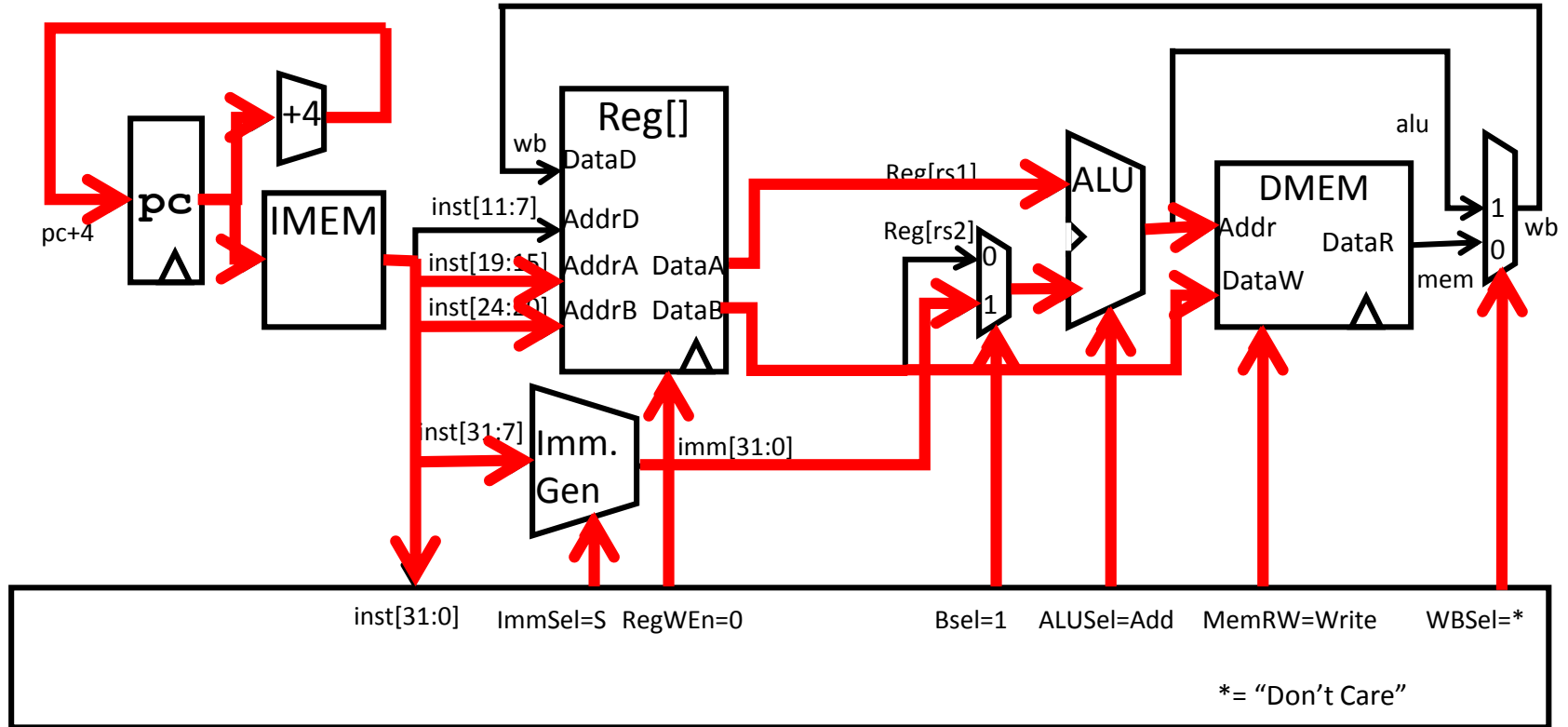
Adding **lw** to datapath



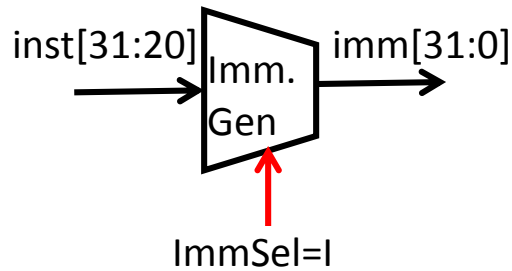
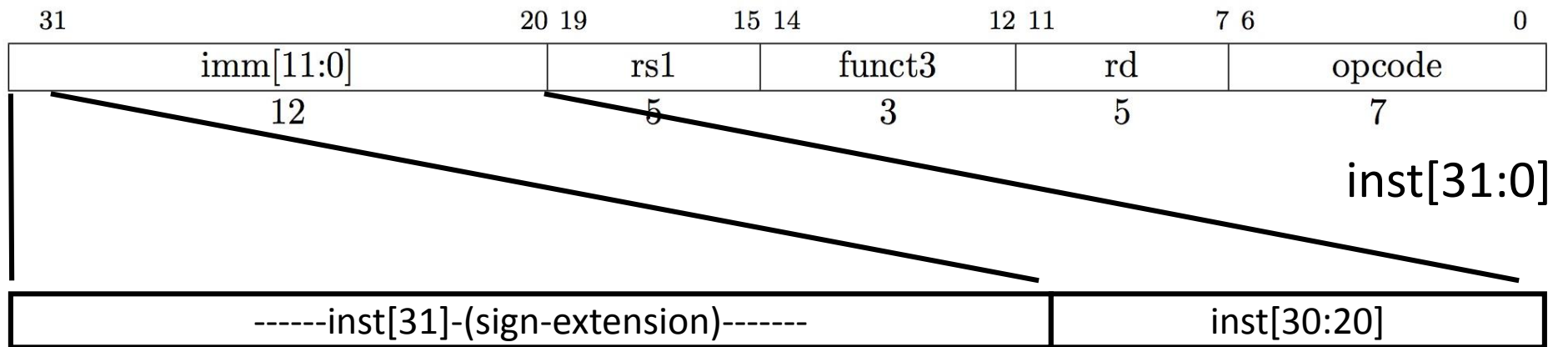
Adding **sw** to datapath



Adding **sw** to datapath

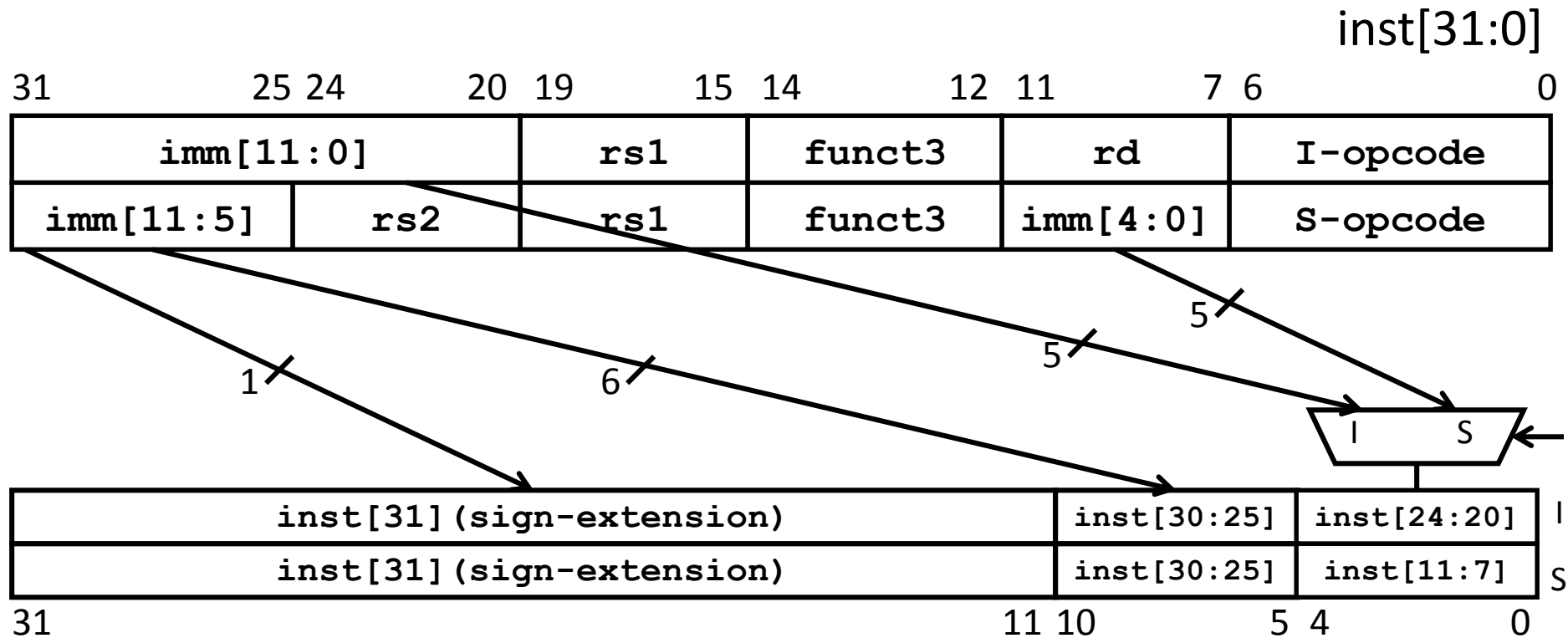


I-Format immediates



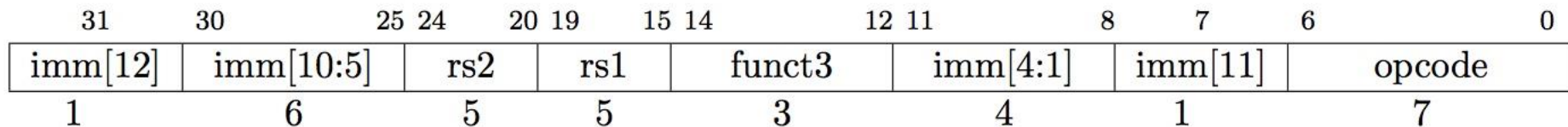
- High 12 bits of instruction (inst[31:20]) copied to low 12 bits of immediate (imm[11:0])
- Immediate is sign-extended by copying value of inst[31] to fill the upper 20 bits of the immediate value (imm[31:12])

I & S Immediate Generator



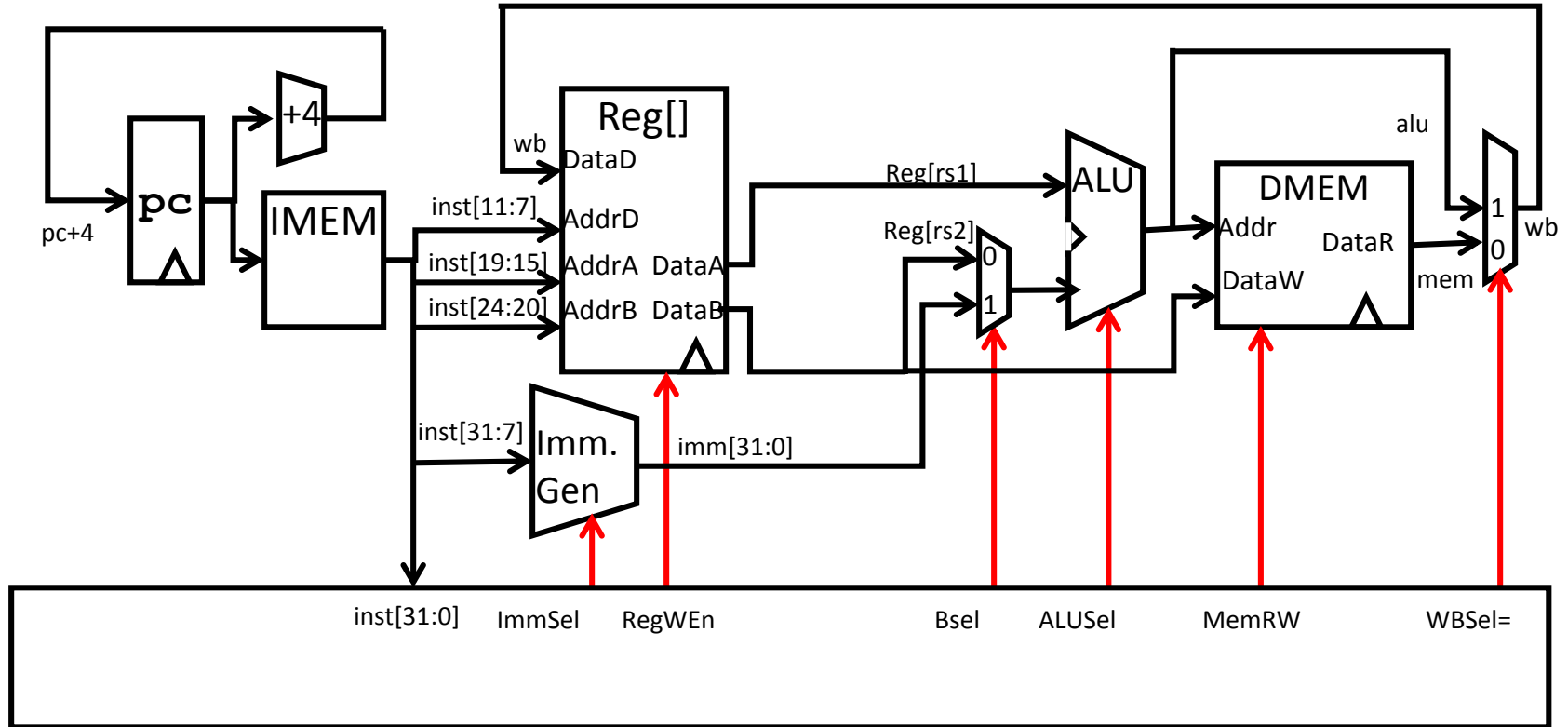
- Just need a 5-bit mux to select between two positions where low five bits of immediate can reside in instruction
- Other bits in immediate are wired to fixed positions in instruction

Implementing Branches

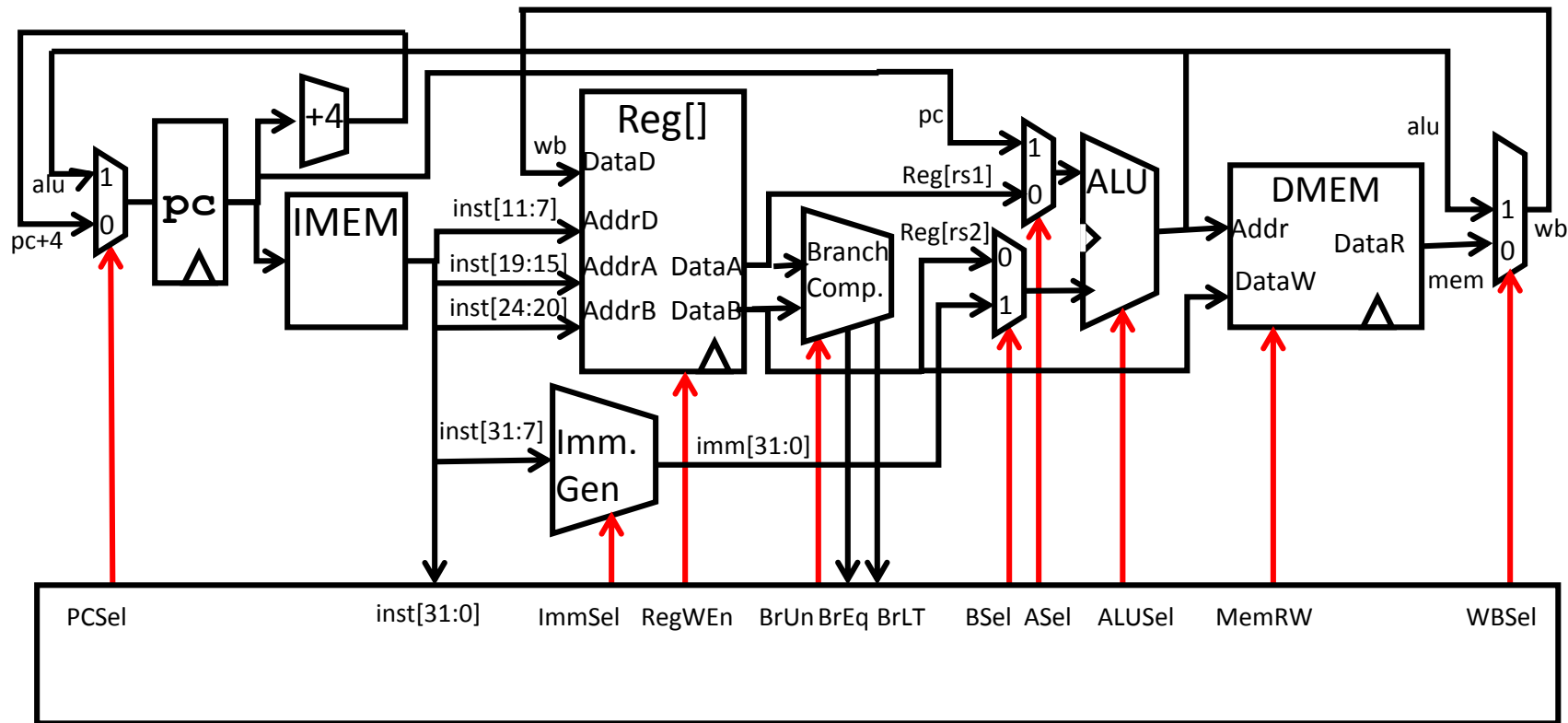


- B-format is mostly same as S-Format, with two register sources (rs1/rs2) and a 12-bit immediate
- But now immediate represents values -4096 to +4094 in 2-byte increments
- The 12 immediate bits encode *even* 13-bit signed byte offsets (lowest bit of offset is always zero, so no need to store it)

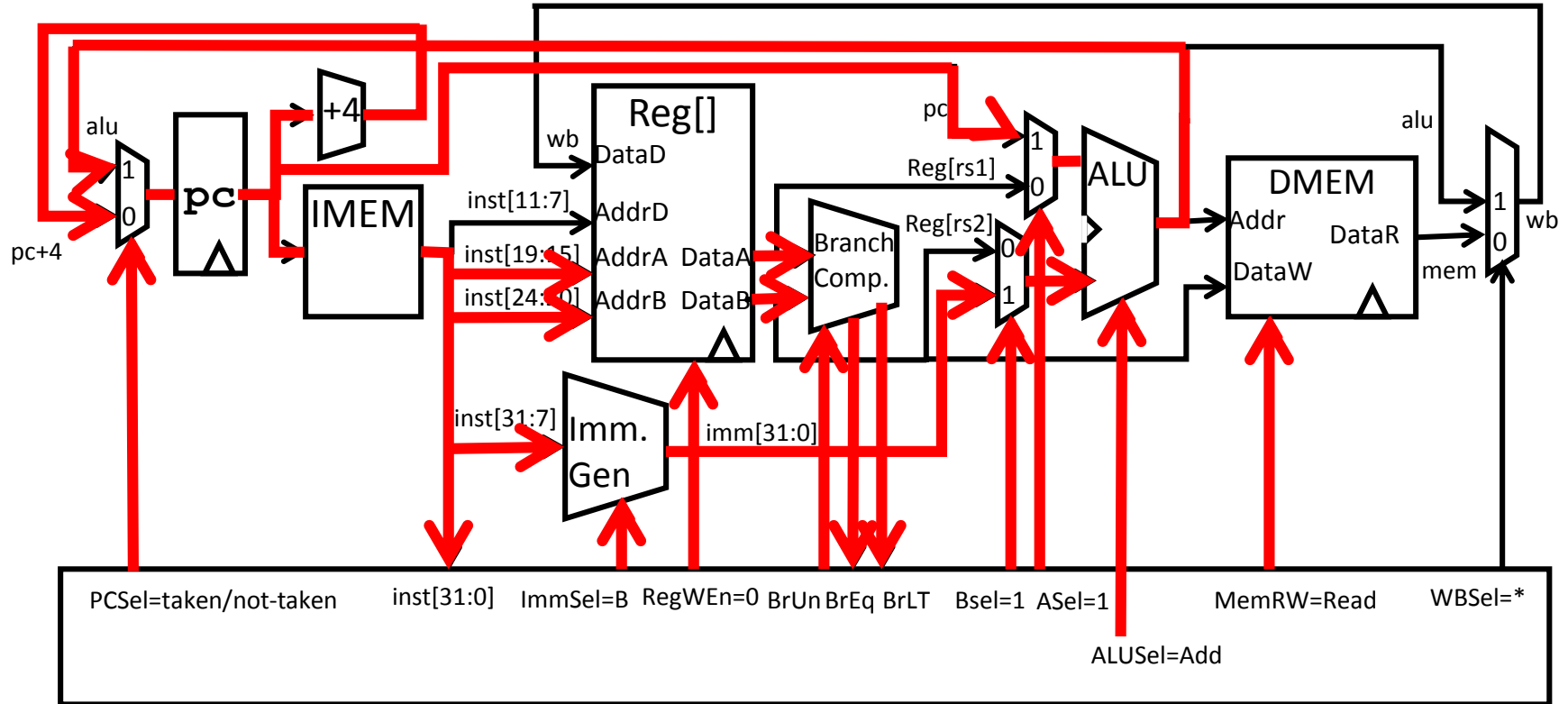
Adding **sw** to datapath



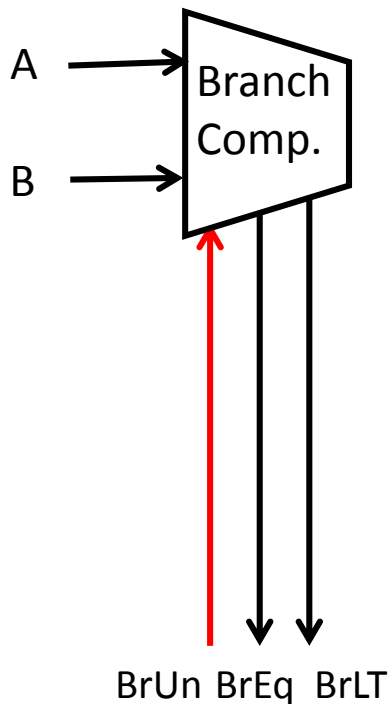
Adding branches to datapath



Adding branches to datapath



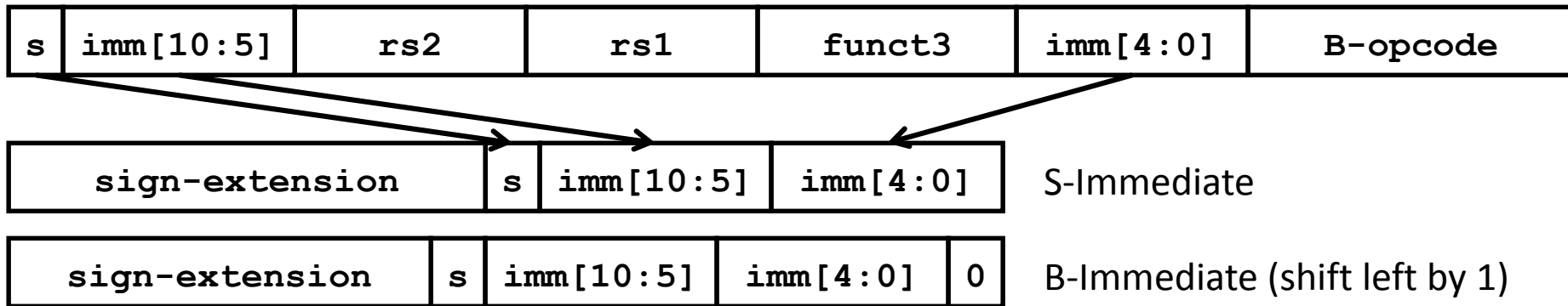
Branch Comparator



- $\text{BrEq} = 1$, if $A=B$
- $\text{BrLT} = 1$, if $A < B$
- $\text{BrUn} = 1$ selects unsigned comparison for BrLT , 0=signed
- BGE branch: $A \geq B$, if $\neg(A < B)$

Multiply Branch Immediates by Shift?

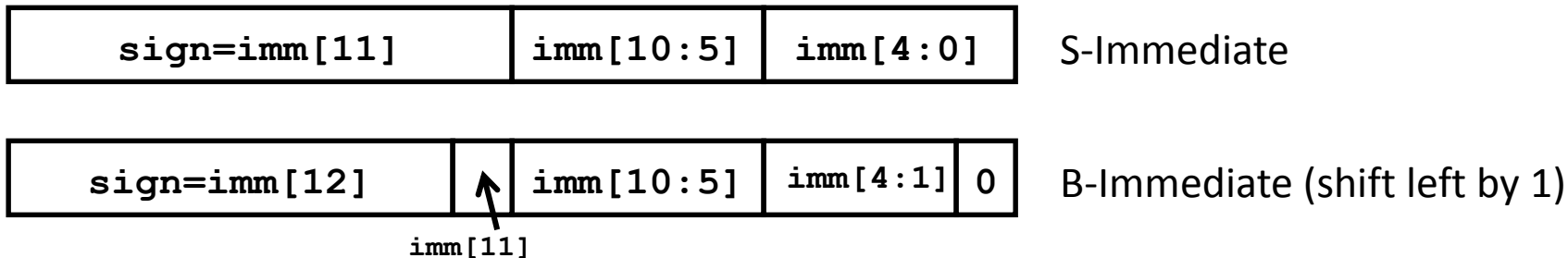
- 12-bit immediate encodes PC-relative offset of -4096 to +4094 bytes in multiples of 2 bytes
- Standard approach: treat immediate as in range -2048..+2047, then shift left by 1 bit to multiply by 2 for branches



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

Instruction Encodings, inst[31:0]

31	30	25	24	21	20	19	15	14	12	11	8	7	6	0		
funct7				rs2			rs1		funct3		rd			opcode		R-type

imm[11:0]										rs1		funct3		rd			opcode	I-type
-----------	--	--	--	--	--	--	--	--	--	-----	--	--------	--	----	--	--	--------	--------

imm[11:5]					rs2				rs1		funct3		imm[4:0]			opcode	S-type
-----------	--	--	--	--	-----	--	--	--	-----	--	--------	--	----------	--	--	--------	--------

imm[12]	imm[10:5]				rs2				rs1		funct3		imm[4:1]	imm[11]	opcode	B-type
---------	-----------	--	--	--	-----	--	--	--	-----	--	--------	--	----------	---------	--------	--------

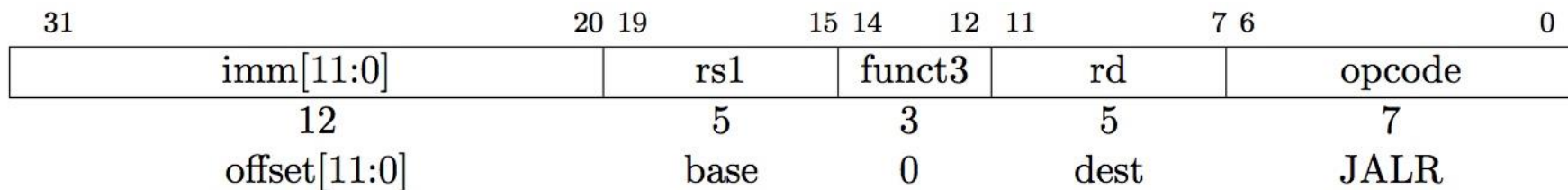
32-bit immediates produced, imm[31:0]

31	30		20	19		12	11	10		5	4		1	0		
— inst[31] —										inst[30:25]		inst[24:21]		inst[20]	I-immediate	
— inst[31] —										inst[30:25]		inst[11:8]		inst[7]	S-immediate	
— inst[31] —										inst[7]	inst[30:25]		inst[11:8]		0	B-immediate

← Upper bits sign-extended from inst[31] always

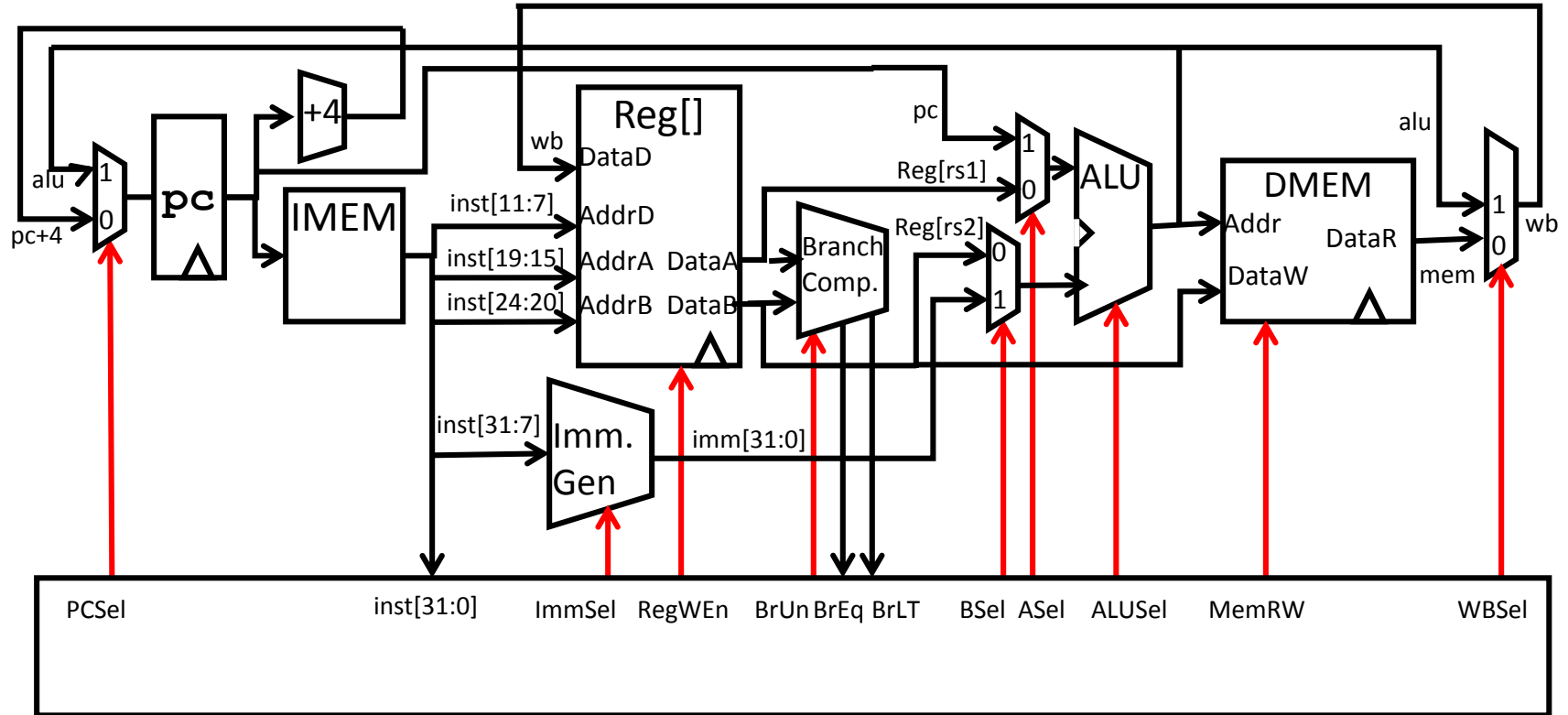
Only bit 7 of instruction changes role in immediate between S and B

Implementing **JALR** Instruction (I-Format)

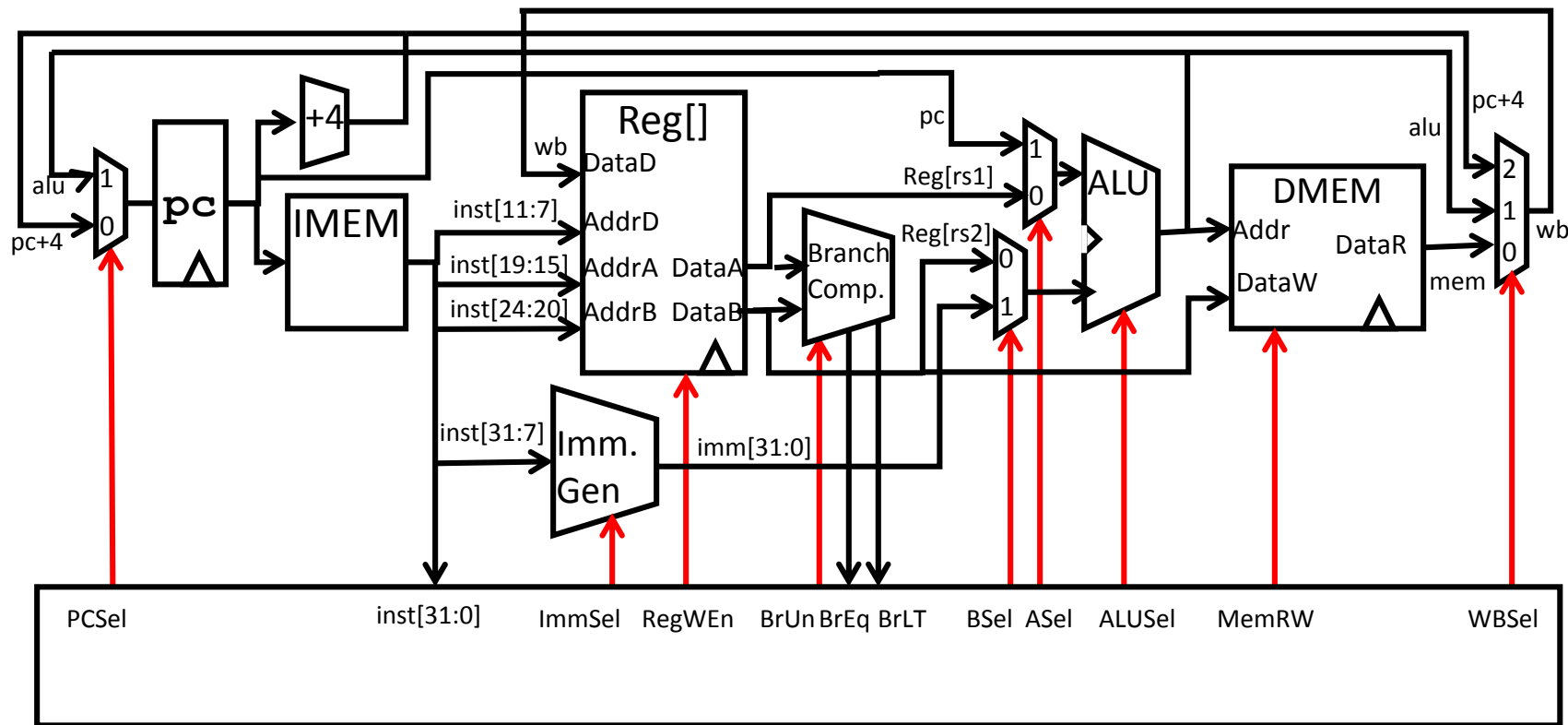


- JALR rd, rs, immediate
 - Writes PC+4 to Reg[rd] (return address)
 - Sets PC = Reg[rs1] + immediate
 - Uses same immediates as arithmetic and loads
 - **no** multiplication by 2 bytes

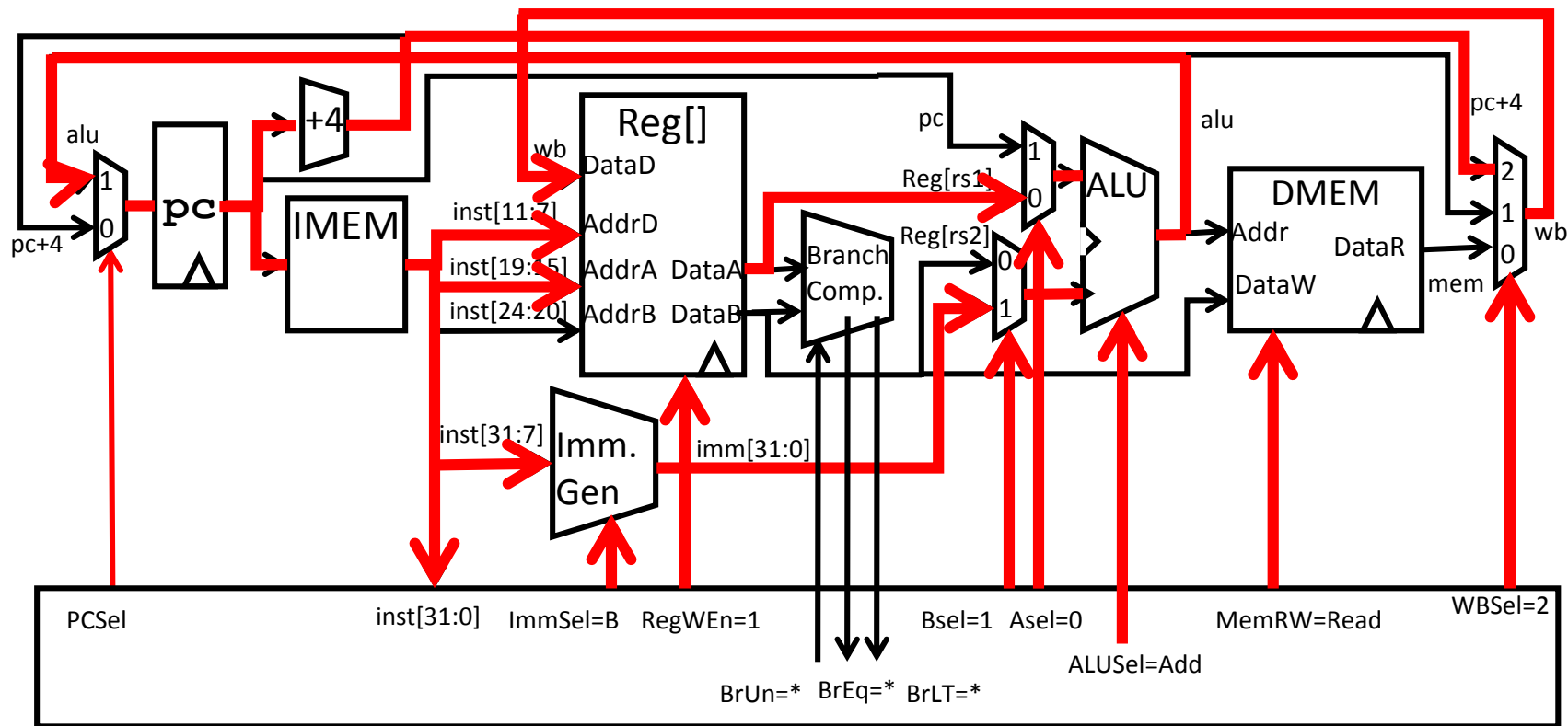
Adding branches to datapath



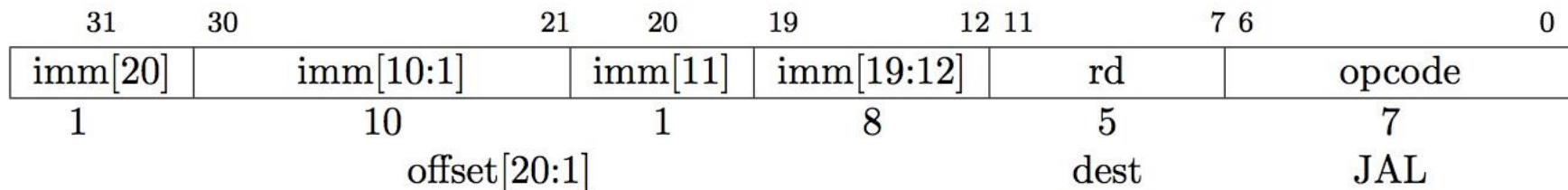
Adding **j**alr to datapath



Adding **j**alr to datapath

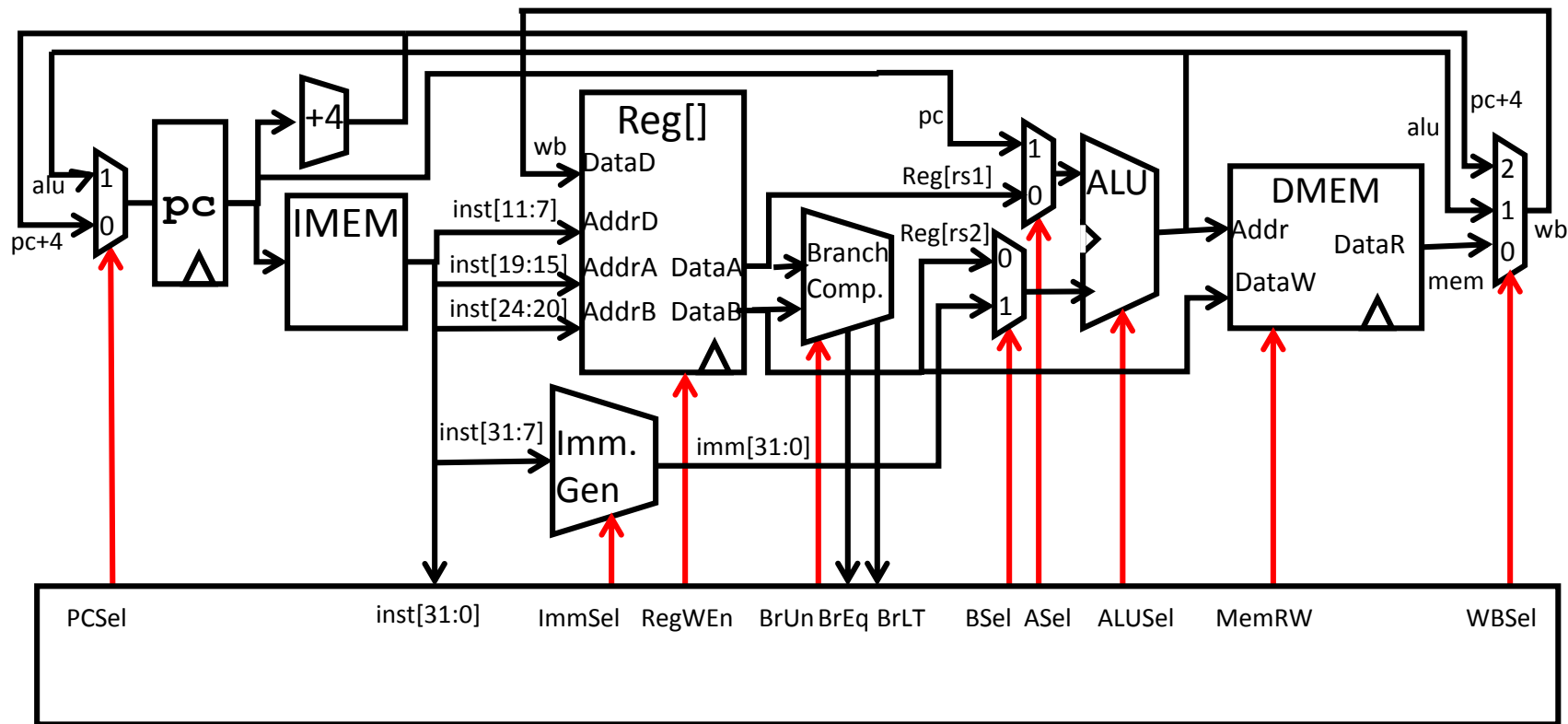


Implementing **j_{al}** Instruction

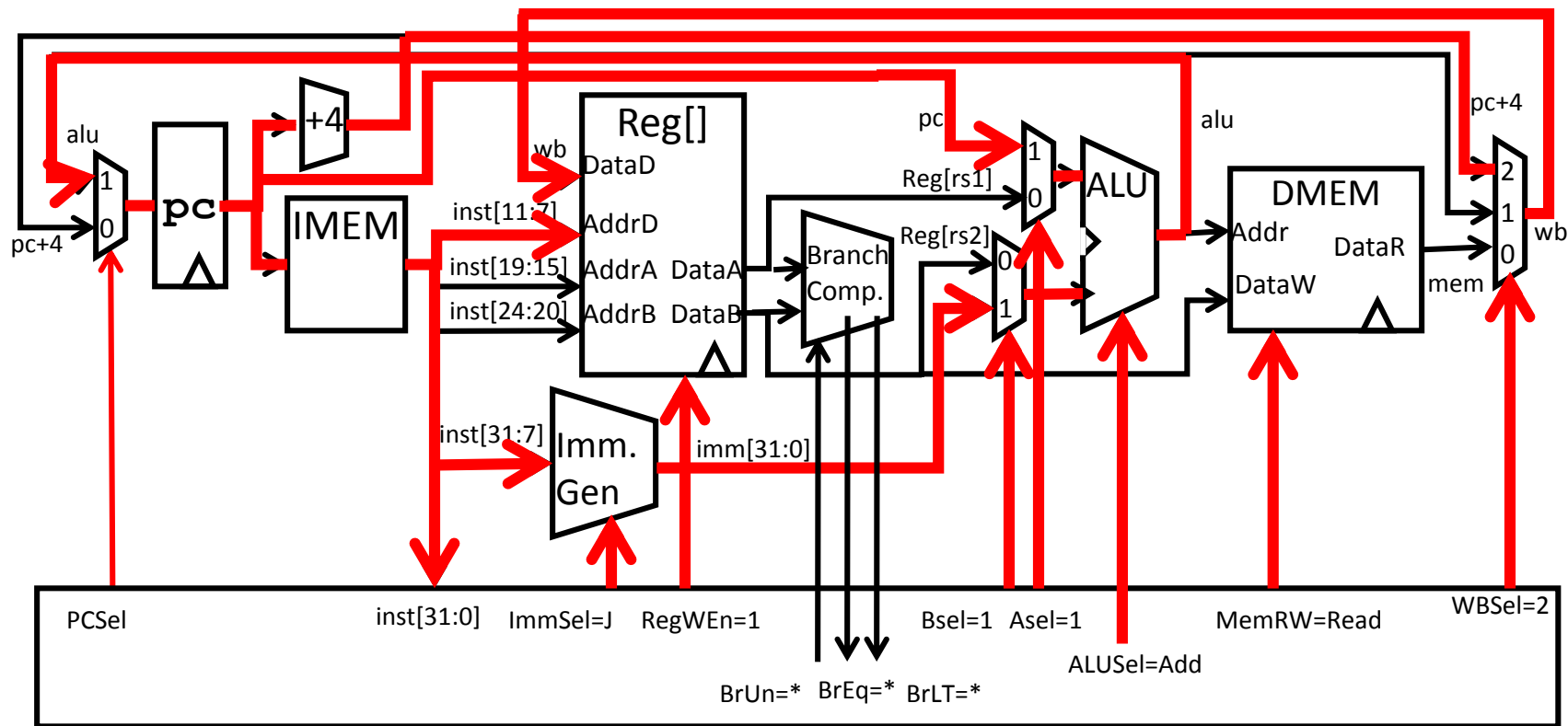


- JAL saves PC+4 in Reg[rd] (the return address)
- Set PC = PC + offset (PC-relative jump)
- Target somewhere within $\pm 2^{19}$ locations, 2 bytes apart
 - $\pm 2^{18}$ 32-bit instructions
- Immediate encoding optimized similarly to branch instruction to reduce hardware cost

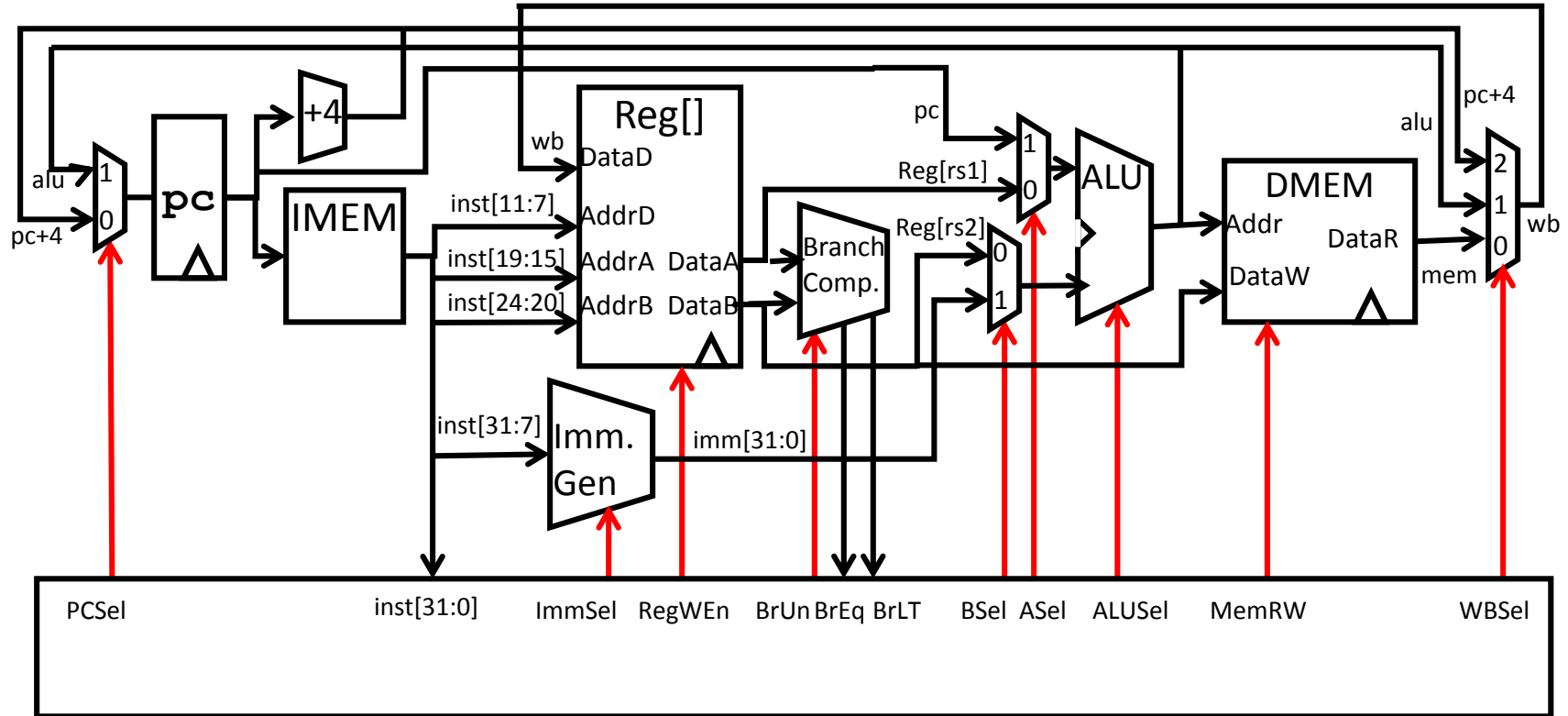
Adding **jal** to datapath



Adding **jal** to datapath



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?