inst.eecs.berkeley.edu/~cs61c

CS61C : Machine Structures

Lecture 30 – RISC-V Datapath II

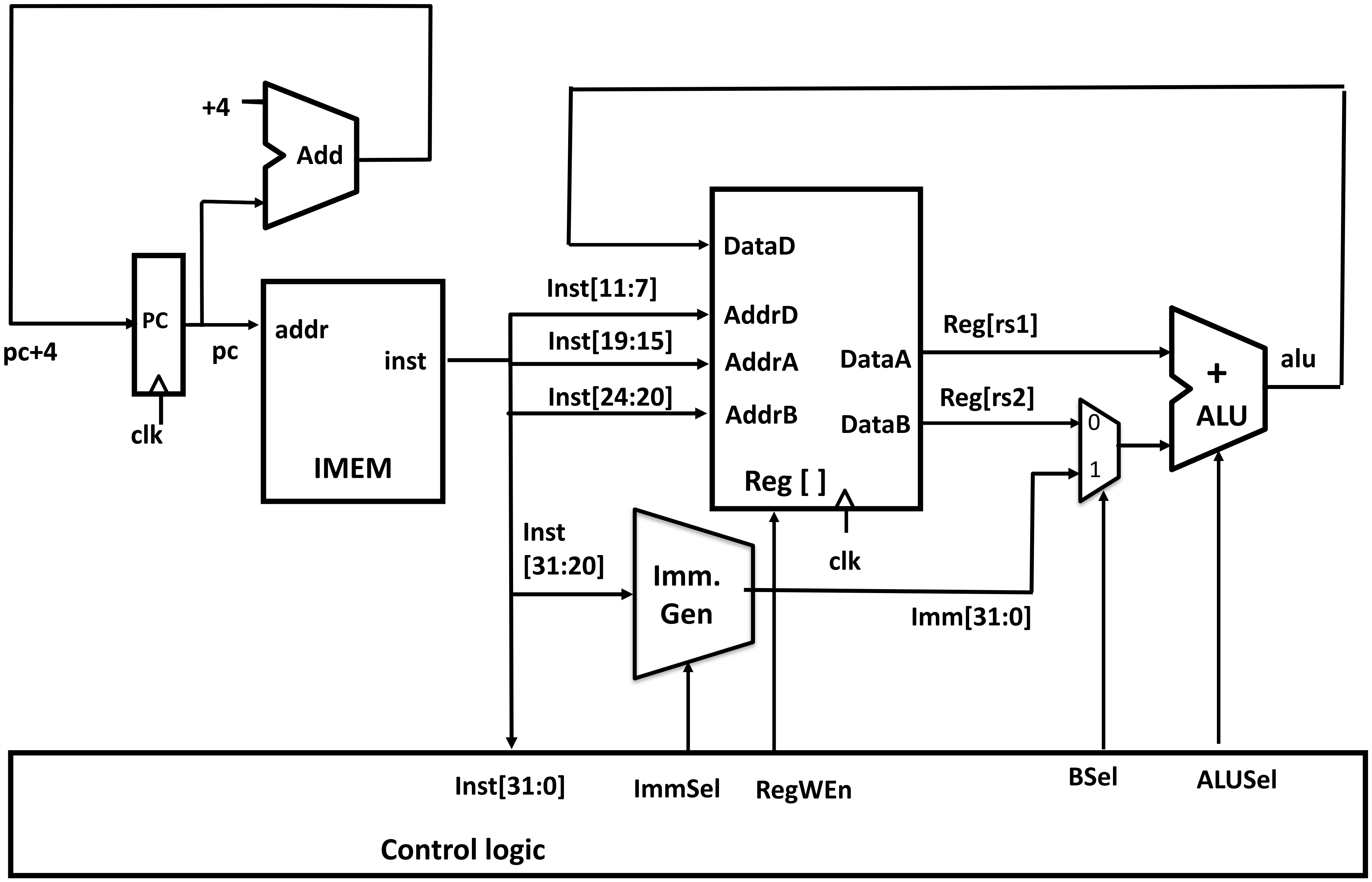
Instructors
Dan Garcia and Bora Nikolic

2018-10-31

Review

- CPU design involves datapath and control logic
- Typical five stages of execution
  - Fetch, Decode, Execute, Memory access, Write back
- We built a datapath for arithmetic and logic for RISC-V R and I instructions
  - And sketched out control
- Will add data memory access (loads, stores), branches, jumps, etc…

R+I Datapath


+4
Add
pc+4
PC
clk
pc
addr
inst
IMEM
Inst[11:7]
Inst[19:15]
Inst[24:20]
DataD
AddrD
AddrA
AddrB
DataA
DataB
Reg [ ]
clk
Reg[rs1]
Reg[rs2]
0
1
+
ALU
alu
Inst
[31:20]
Imm.
Gen
Imm[31:0]
Inst[31:0]
ImmSel
RegWEn
BSel
ALUSel
Control logic

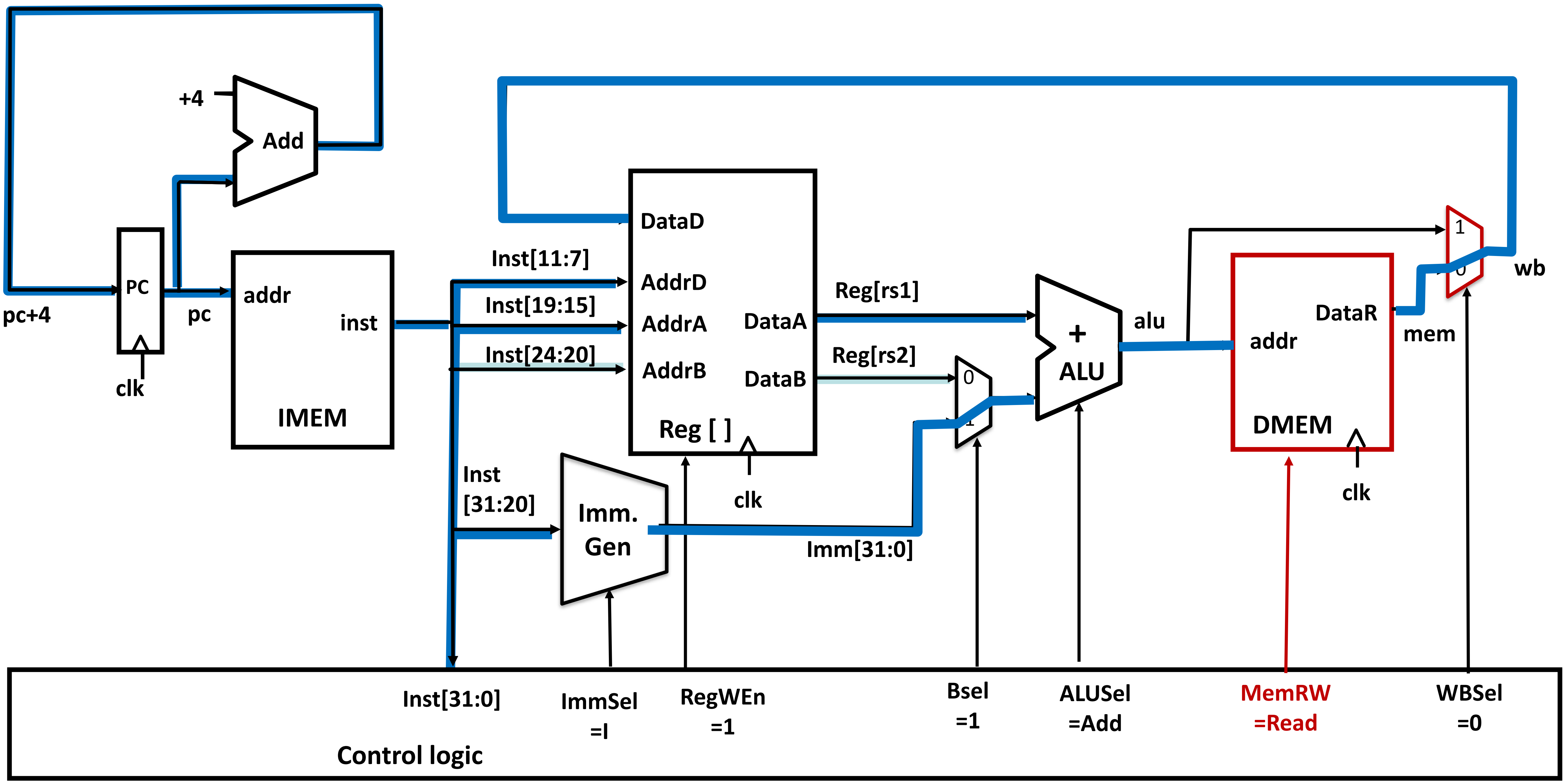
Add `lw`

- RISC-V Assembly Instruction (I-type): `lw x14, 8(x2)`

31 20	19 15	14 12	11 7	6 0
imm[11:0]	rs1	funct3	rd	opcode
12	5	3	5	7
offset[11:0]	base	width	dest	LOAD
31 20	19 15	14 12	11 7	6 0
000000001000	00010	010	01110	0000011
imm= +8	rs1=2	LW	rd=14	LOAD

- The 12-bit signed immediate is added to the base address in register rs1 to form the memory address
  - This is very similar to the add-immediate operation but used to create address not to create final result
- The value loaded from memory is stored in register rd

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 logic 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.
  - It is just a mux mod

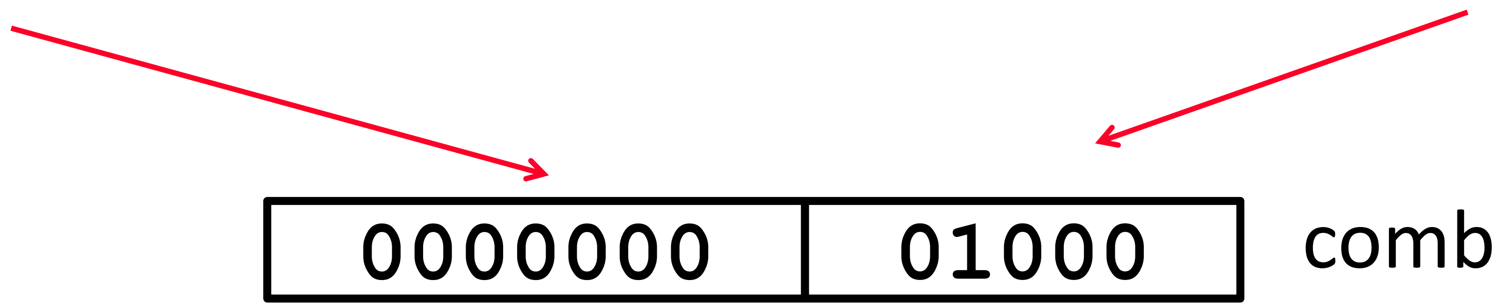
Adding `sw` Instruction

- sw: Reads two registers, rs1 for base memory address, and rs2 for data to be stored, as well immediate offset!

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

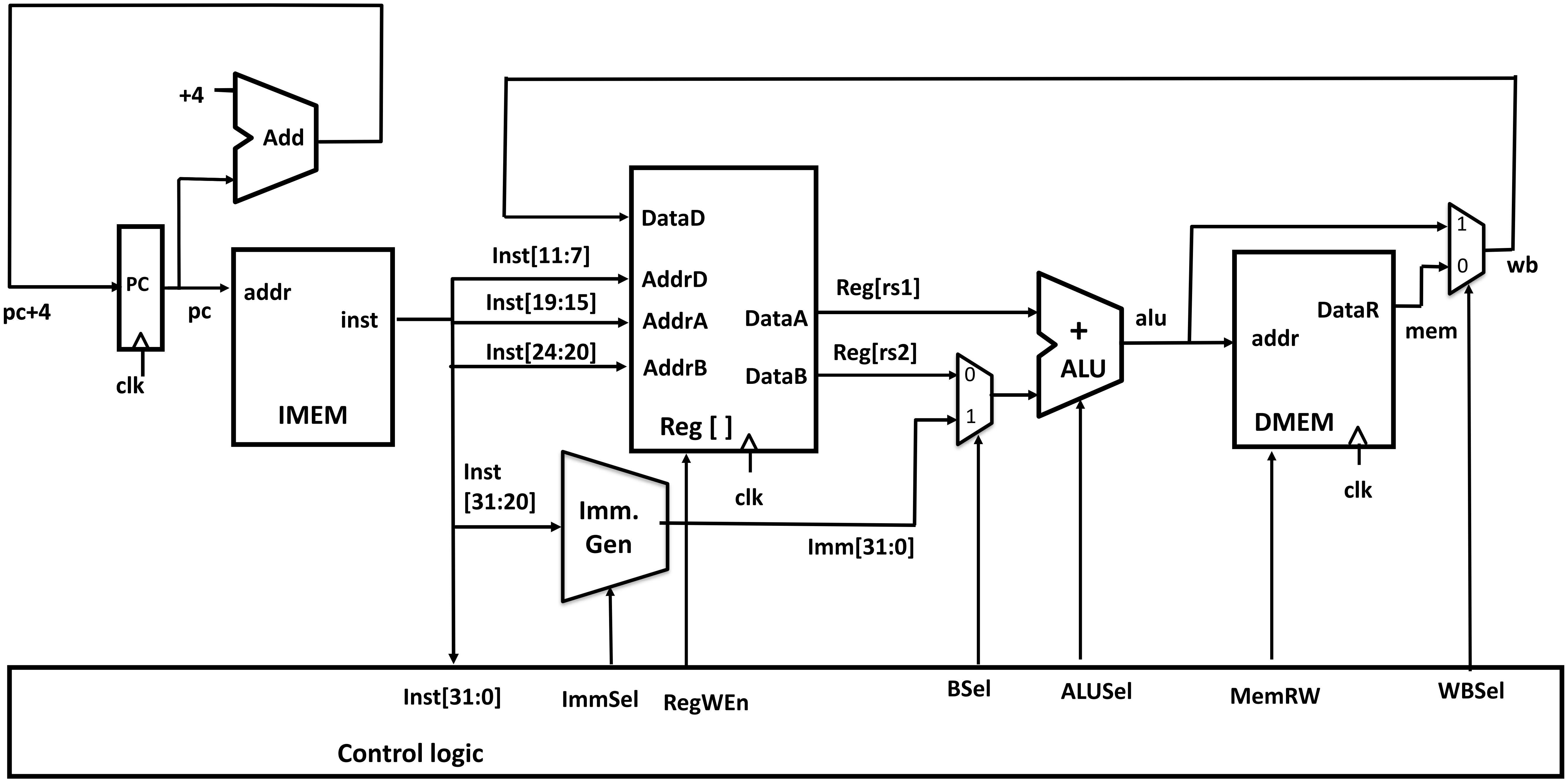
31 25	24 20	19 15	14 12	11 7	6 0
Imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
7	5	5	3	5	7
offset[11:5]	src	base	width	offset[4:0]	STORE

0000000	01110	00010	010	01000	0100011
offset[11:5] =0	rs2=14	rs1=2	SW	offset[4:0] =8	STORE

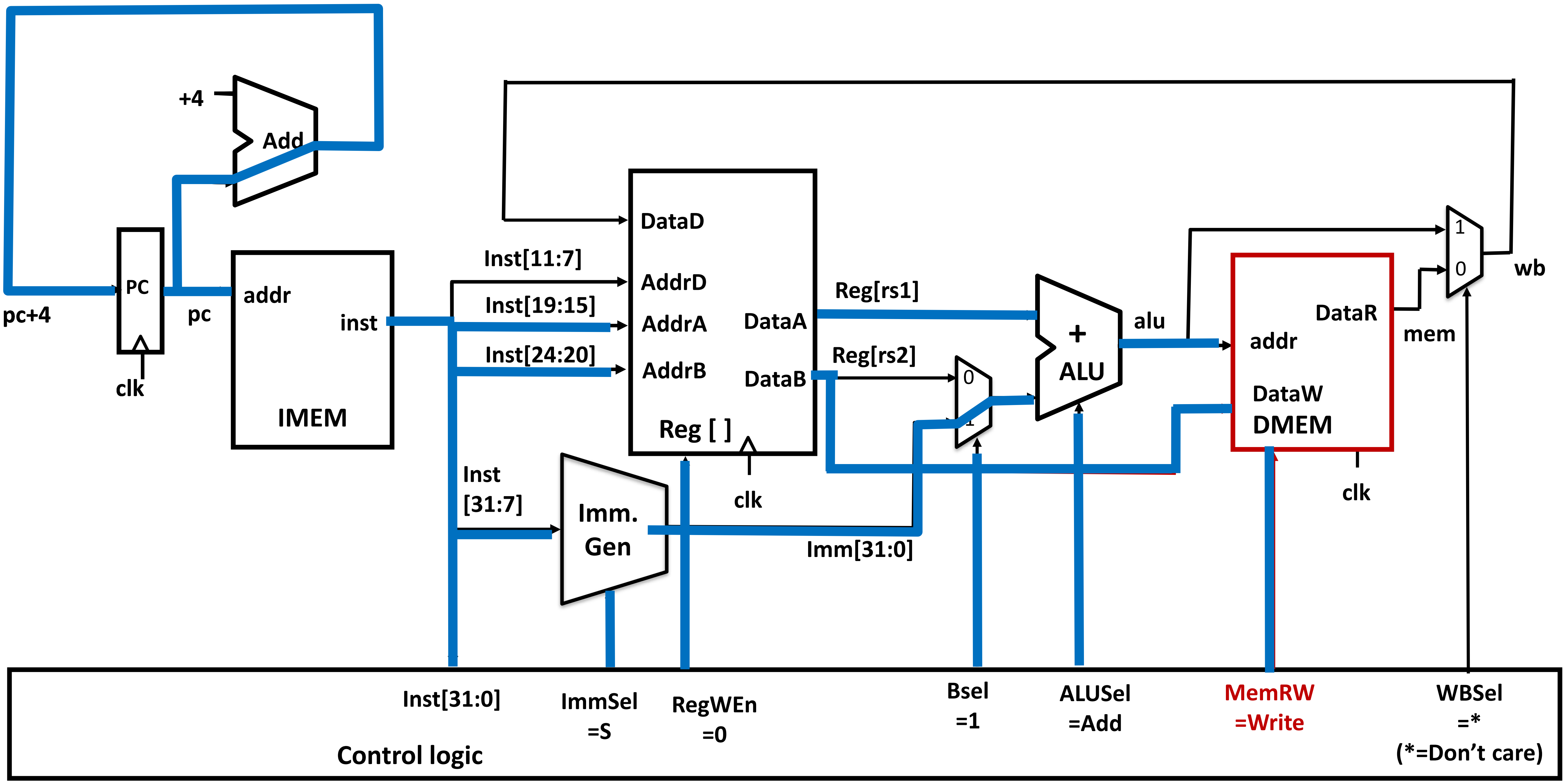


combined 12-bit offset = 8

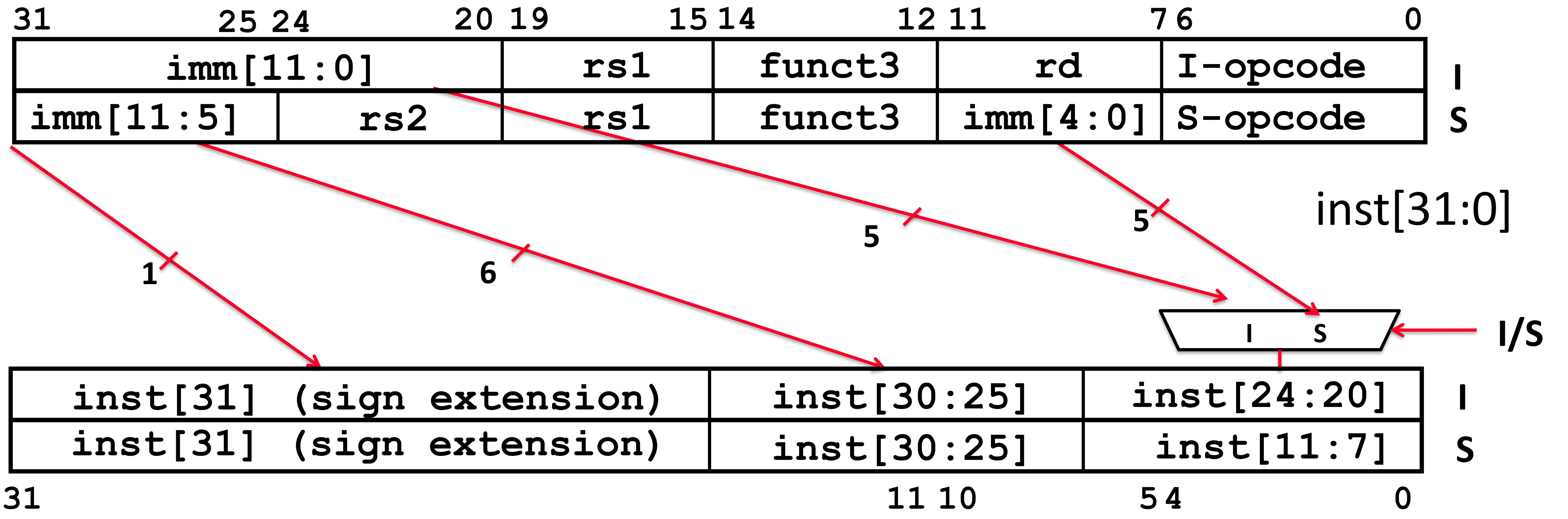
Datapath with `lw`



Adding sw to Datapath



I+S Immediate Generation



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

31	11 10 5	4 0	
inst[31] (sign extension)	inst[30:25]	inst[24:20]	I
inst[31] (sign extension)	inst[30:25]	inst[11:7]	S

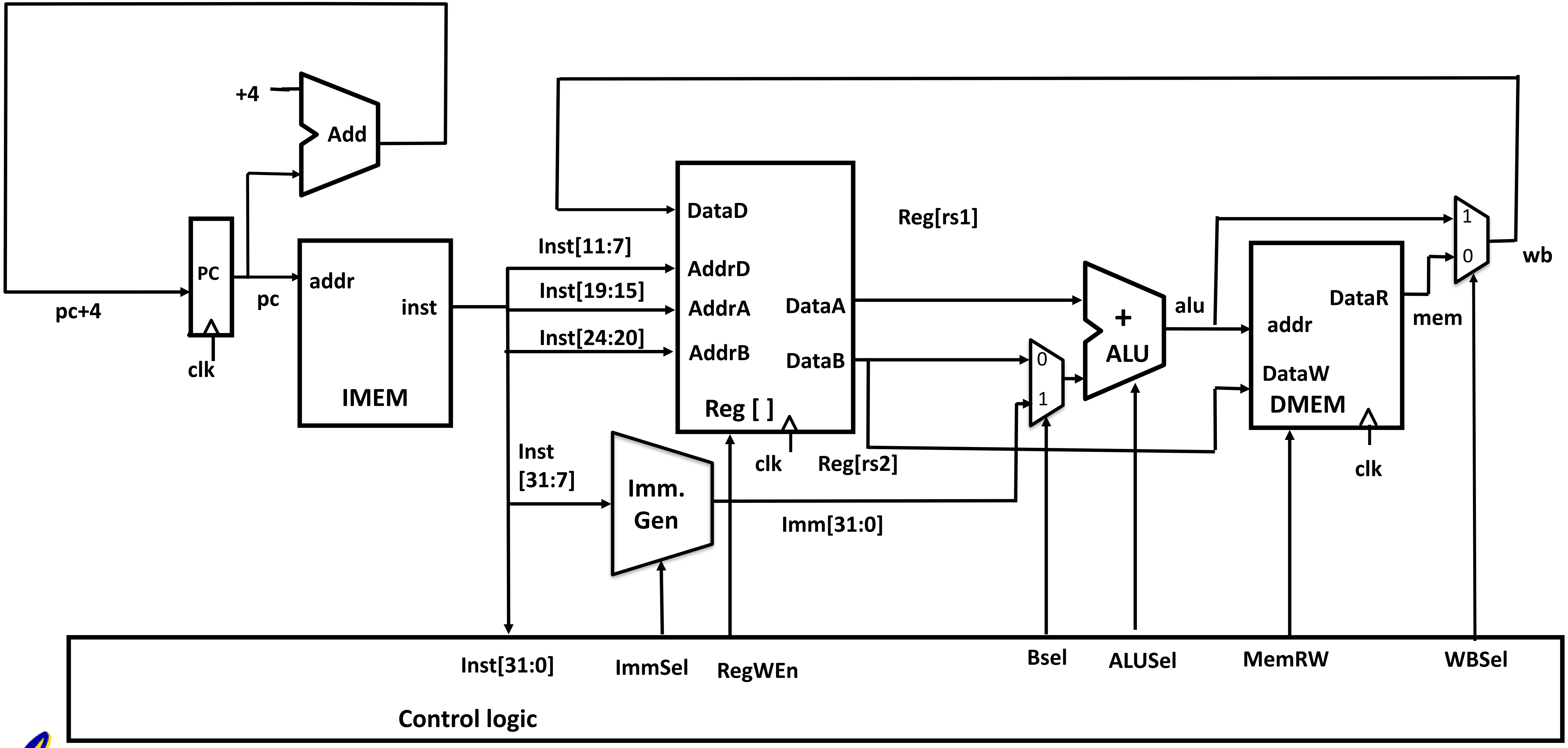
- Just need a 5-bit mux to select between two positions where low five bits of immediate can reside in instruction
- Other bits in immediate are wired to fixed positions in instruction

Implementing Branches

31	30 25	24 20	19 15	14 12	11 8	7	6 0
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode
1	6	5	5	3	4	1	7
offset[12|10:5]		rs2	rs1	funct3	offset[4:1|11]		BRANCH

- 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)

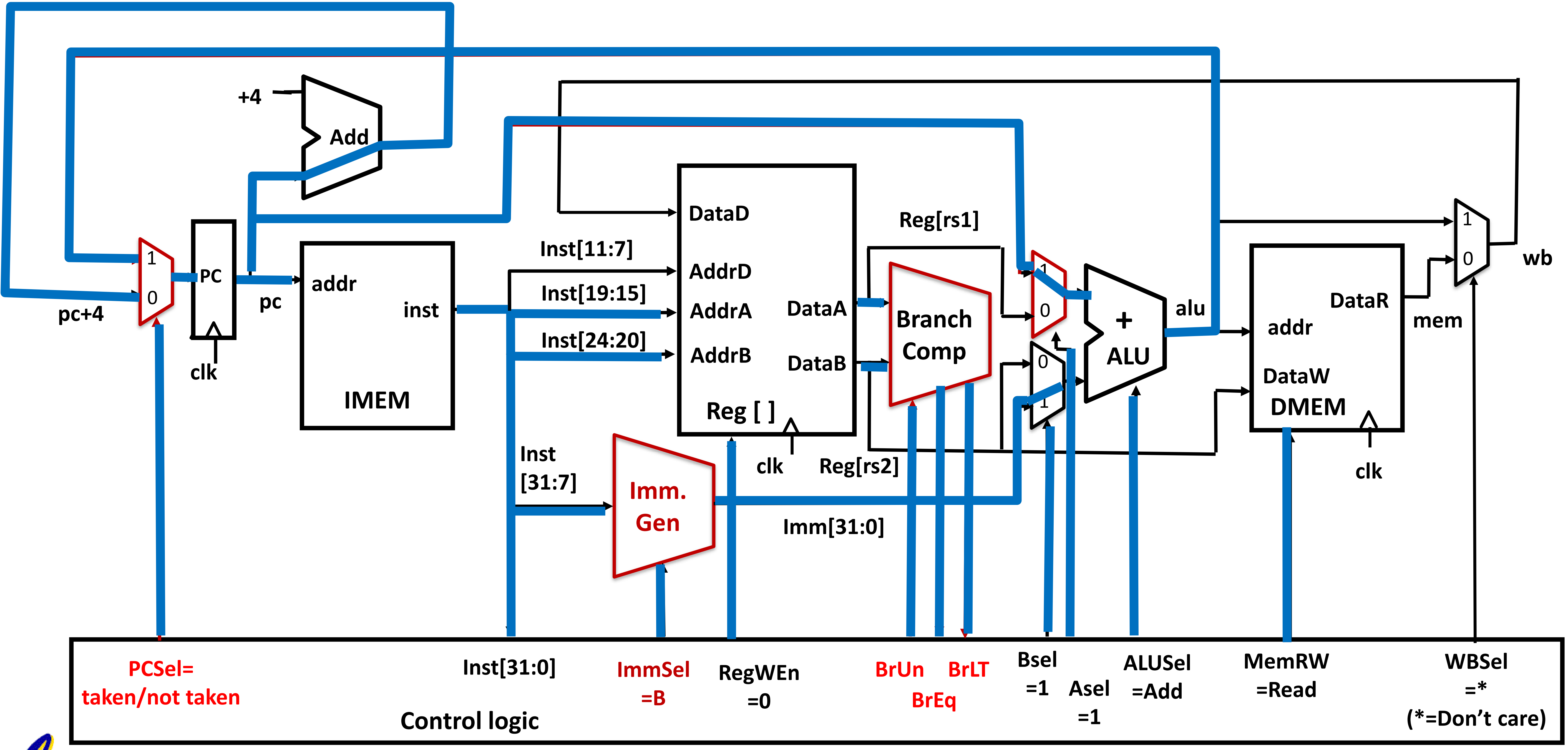
Datapath So Far



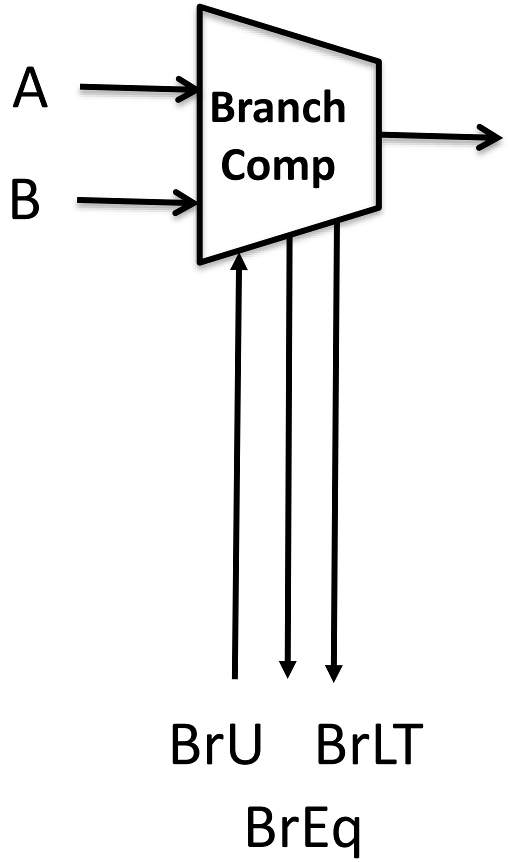
To Add Branches

- Different change to the state:
  - $\texttt{PC} = \begin{cases} \texttt{PC + 4,} & \texttt{branch not taken} \\ \texttt{PC + immediate,} & \texttt{branch taken} \end{cases}$
- Six branch instructions: `BEQ, BNE, BLT, BGE, BLTU, BGEU`
- Need to compute `PC + immediate` and to compare values of `rs1` and `rs2`
  - But have only one ALU – need more hardware

Adding Branches



Branch Comparator



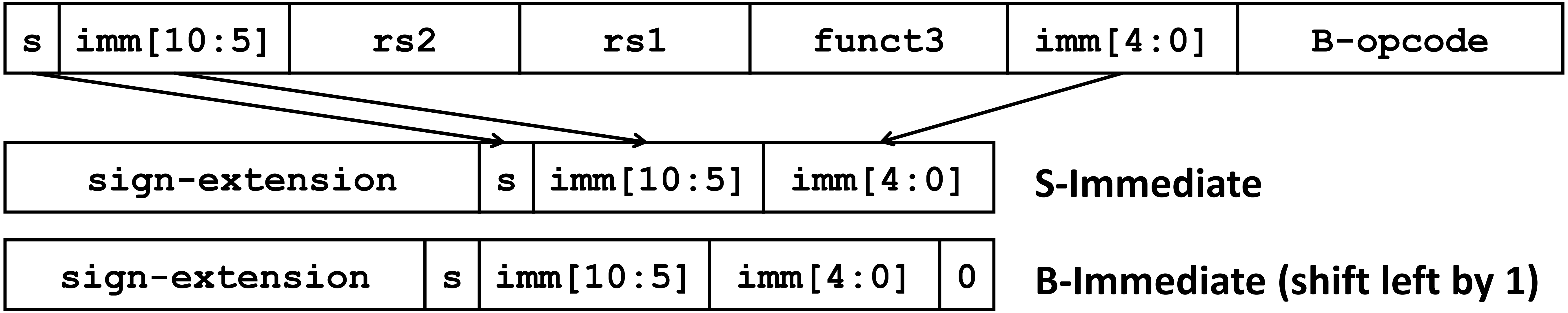
- BrEq = 1, if A=B
- BrLT = 1, if A < B
- BrUn =1 selects unsigned comparison for BrLT, 0=signed

- BGE branch: A >= B, if $\overline{A<B}$

$\overline{A<B}$ = !(A<B)

Branch Immediates (In Other ISAs)

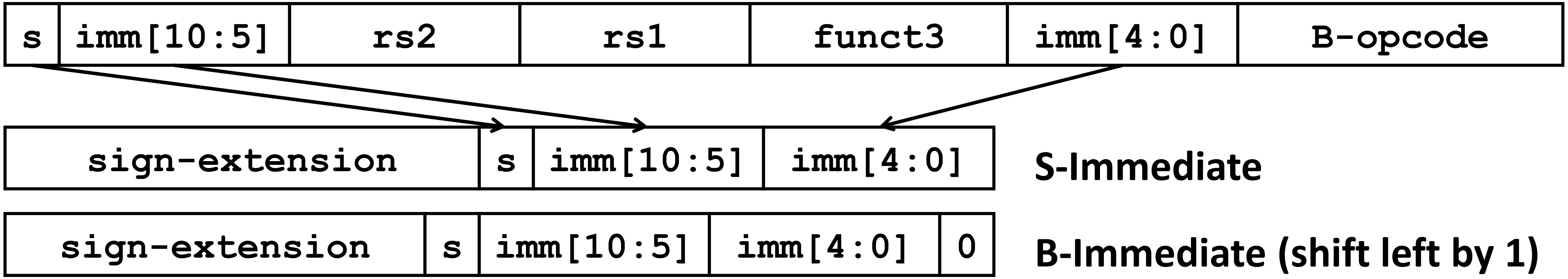
- 12-bit immediate encodes PC-relative offset of -4096 to +4094 bytes in multiples of 2 bytes
- Standard approach: Treat immediate as in range -2048..+2047, then shift left by 1 bit to multiply by 2 for branches



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



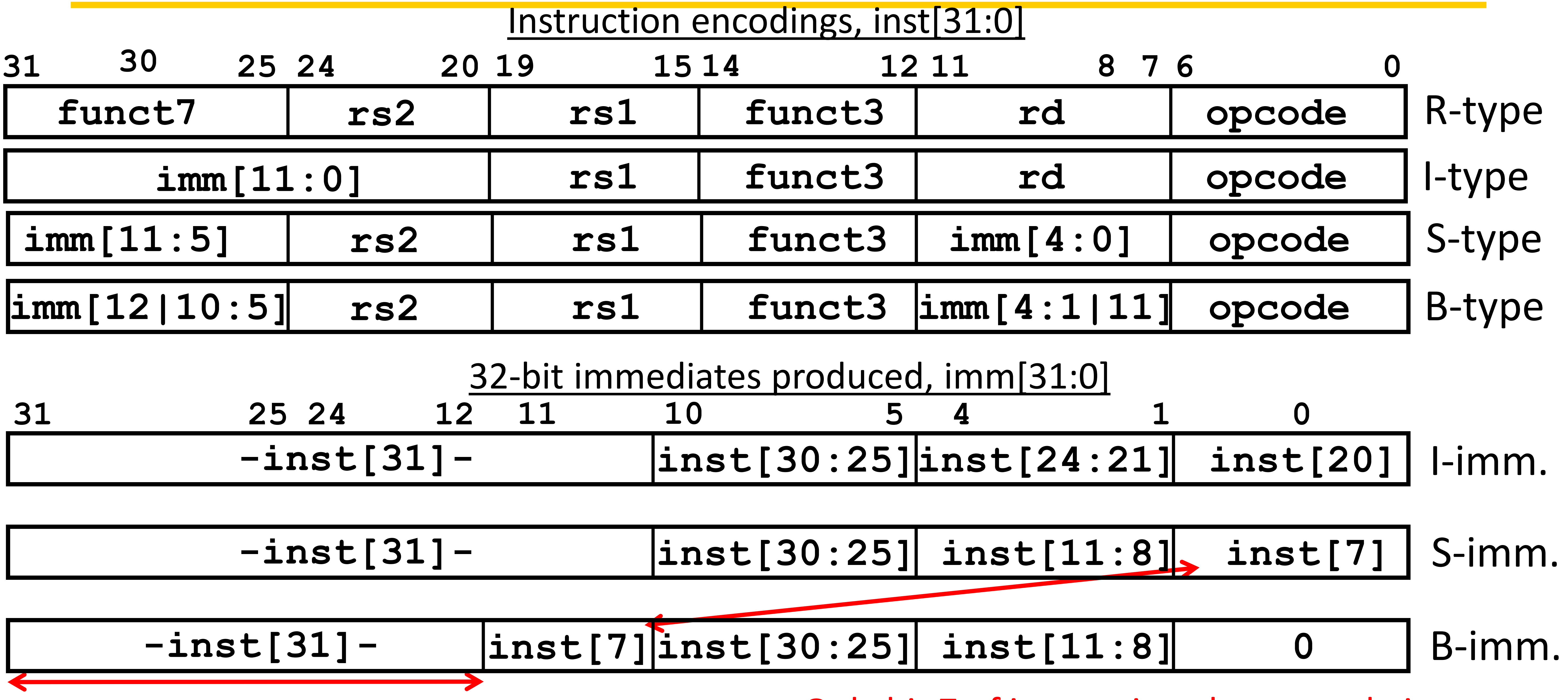
s	imm[10:5]	rs2	rs1	funct3	imm[4:0]	B-opcode

sign-extension	s	imm[10:5]	imm[4:0]	S-Immediate

sign-extension	s	imm[10:5]	imm[4:0]	0	B-Immediate (shift left by 1)

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

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

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

Upper bits sign-extended from inst[31] always

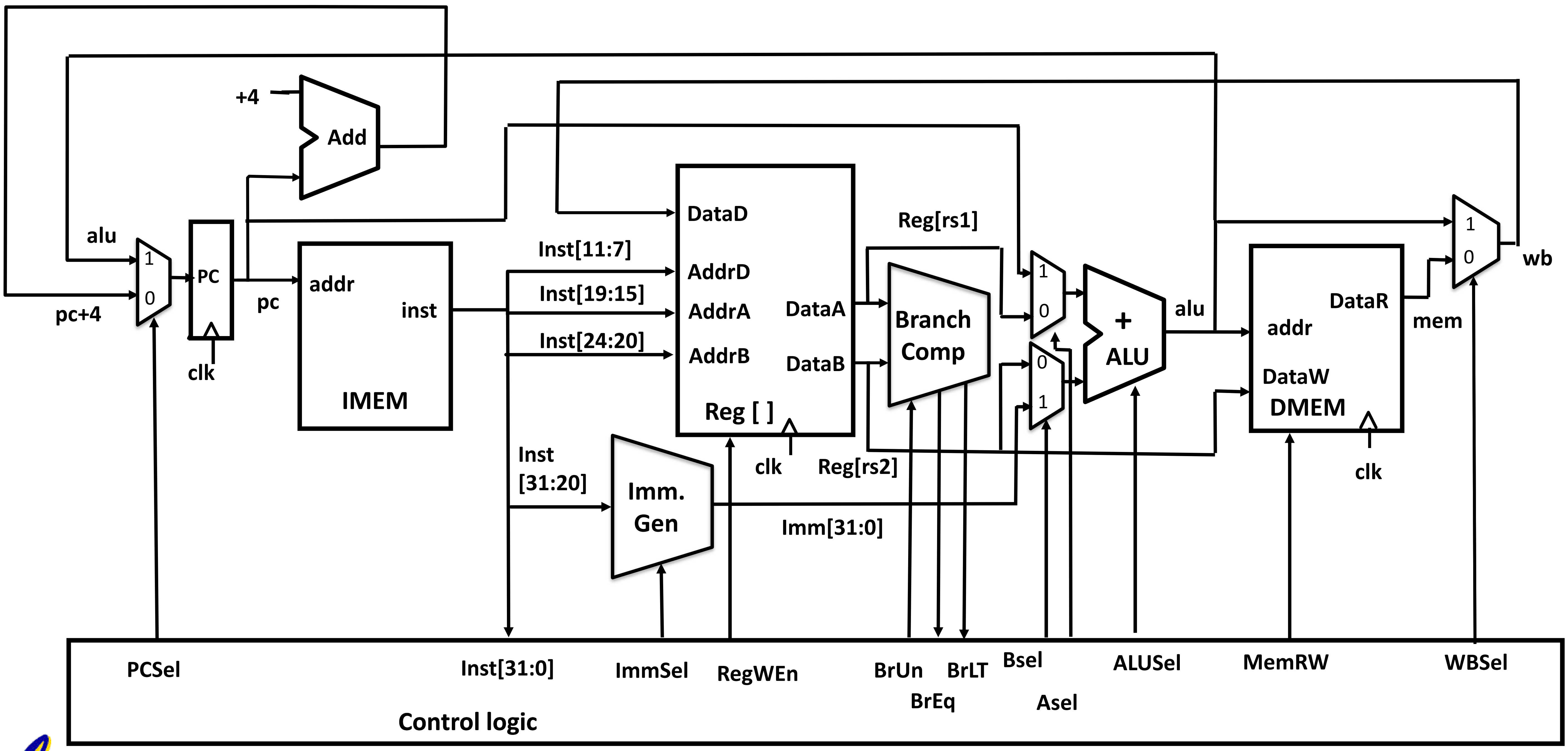
Only bit 7 of instruction changes role in immediate between S and B

Let’s Add JALR (I-Format)

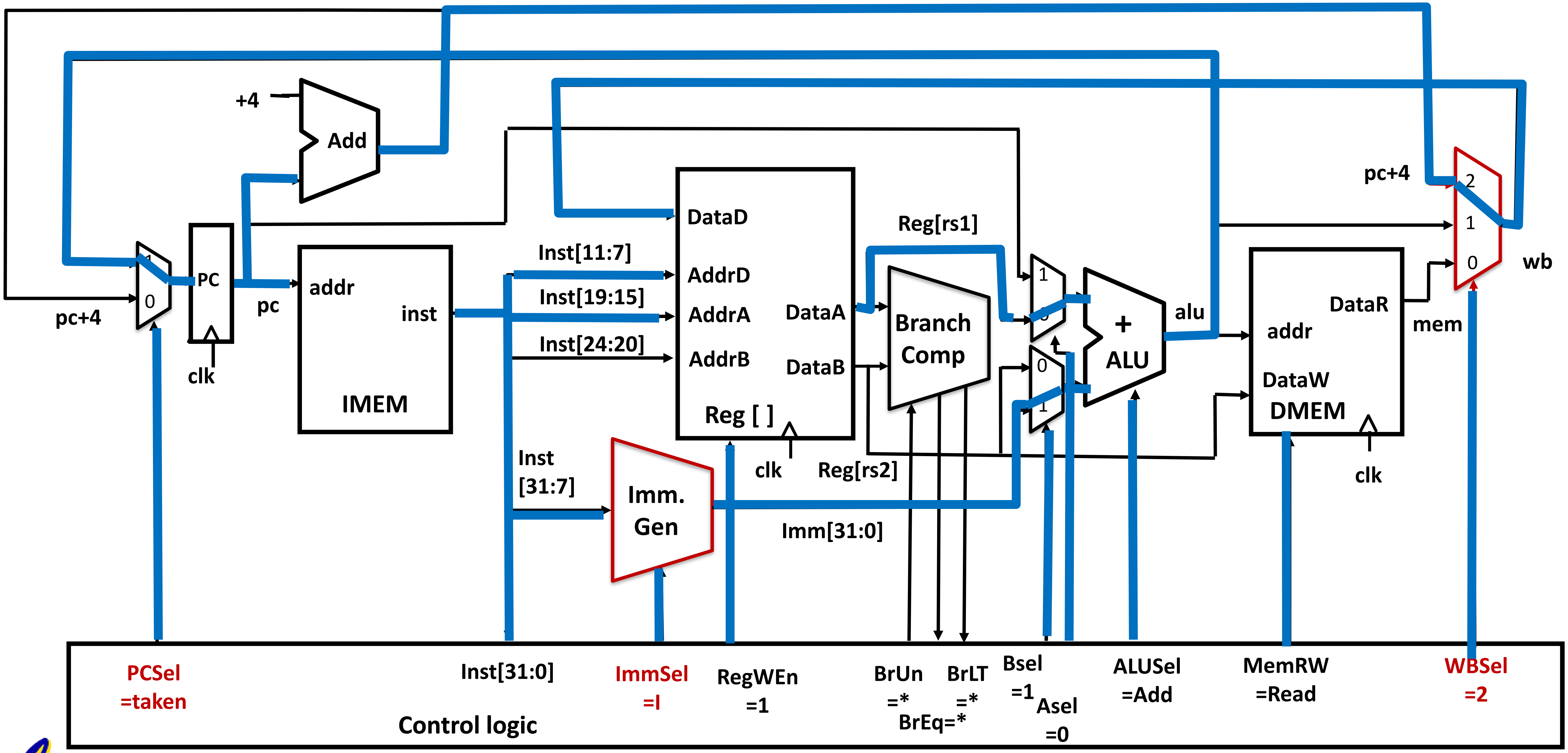
31 20	19 15	14 12	11 7	6 0
imm[11:0]	rs1	func3	rd	opcode
12	5	3	5	7
offset[11:0]	base	0	dest	JALR

- JALR rd, rs, immediate
- Two changes to the state
  - Writes PC+4 to rd (return address)
  - Sets PC = rs + immediate
  - Uses same immediates as arithmetic and loads
    - *no* multiplication by 2 bytes
    - LSB is ignored

Datapath So Far, with Branches



Adding JALR

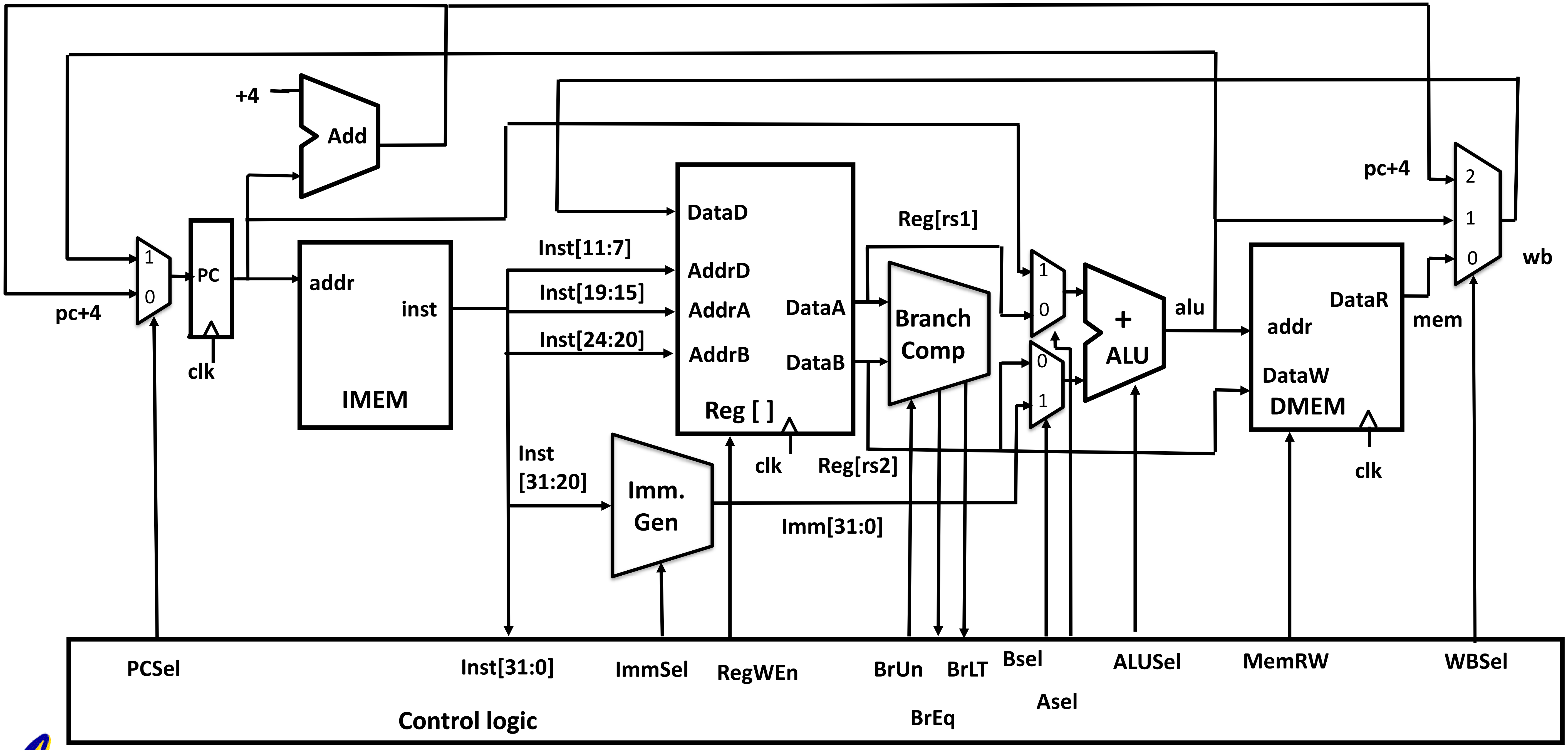

+4
Add
PC
clk
pc+4
1
0
pc
addr
inst
IMEM
Inst[11:7]
Inst[19:15]
Inst[24:20]
DataD
AddrD
AddrA
AddrB
DataA
DataB
Reg [ ]
clk
Inst
[31:7]
Imm.
Gen
Imm[31:0]
Reg[rs1]
Reg[rs2]
Branch
Comp
1
0
0
1
+
ALU
alu
addr
DataR
DataW
DMEM
clk
mem
pc+4
2
1
0
wb
PCSel
=taken
Inst[31:0]
ImmSel
=I
RegWEn
=1
BrUn
=*
BrLT
=*
BrEq=*
Bsel
=1
Asel
=0
ALUSel
=Add
MemRW
=Read
WBSel
=2
Control logic

Adding `JAL`

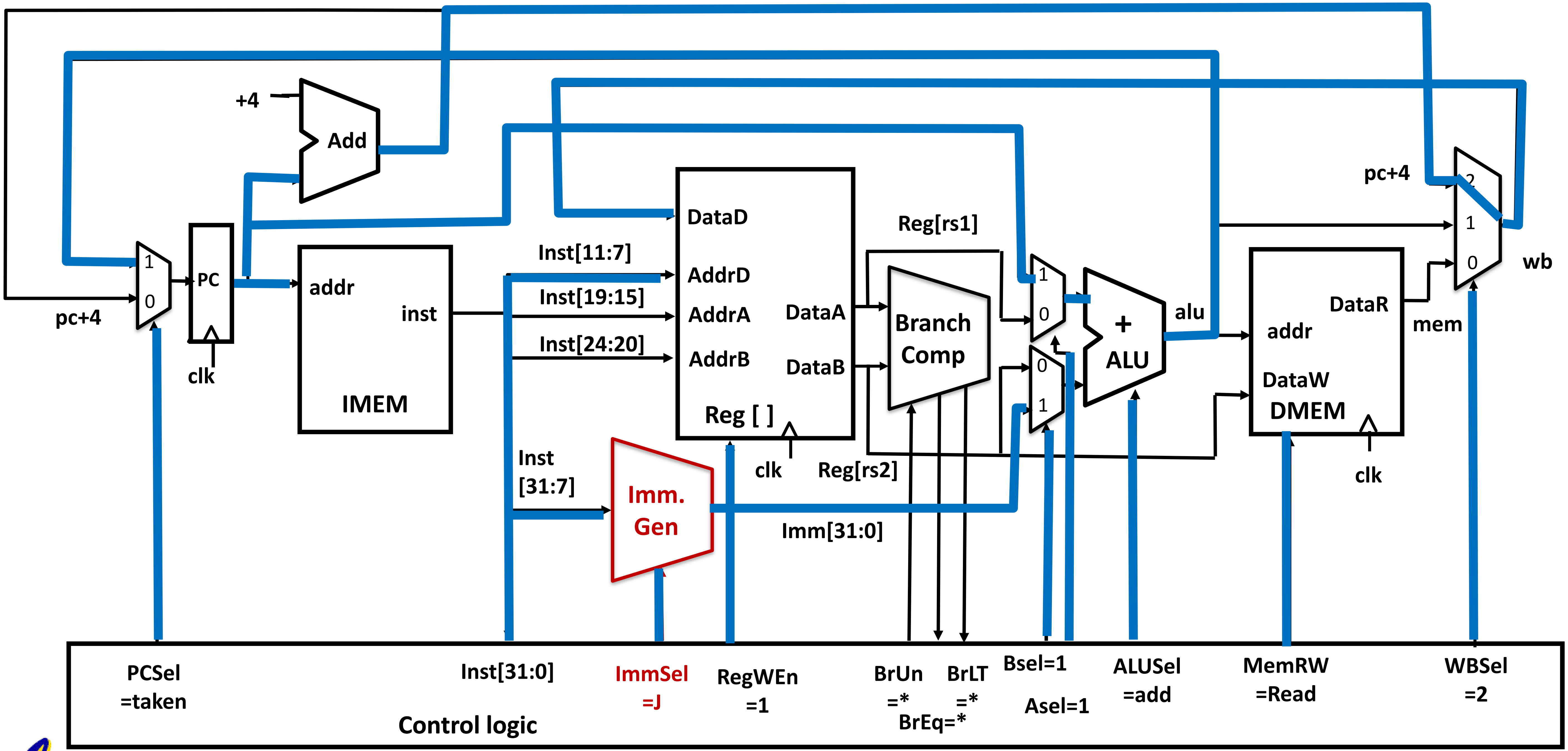
31	30 21	20	19 12	11 7	6 0
imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode
1	10	1	8	5	7
offset[20:1]				dest	JAL

- JAL saves PC+4 in register 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

Datapath with JALR



Adding JAL

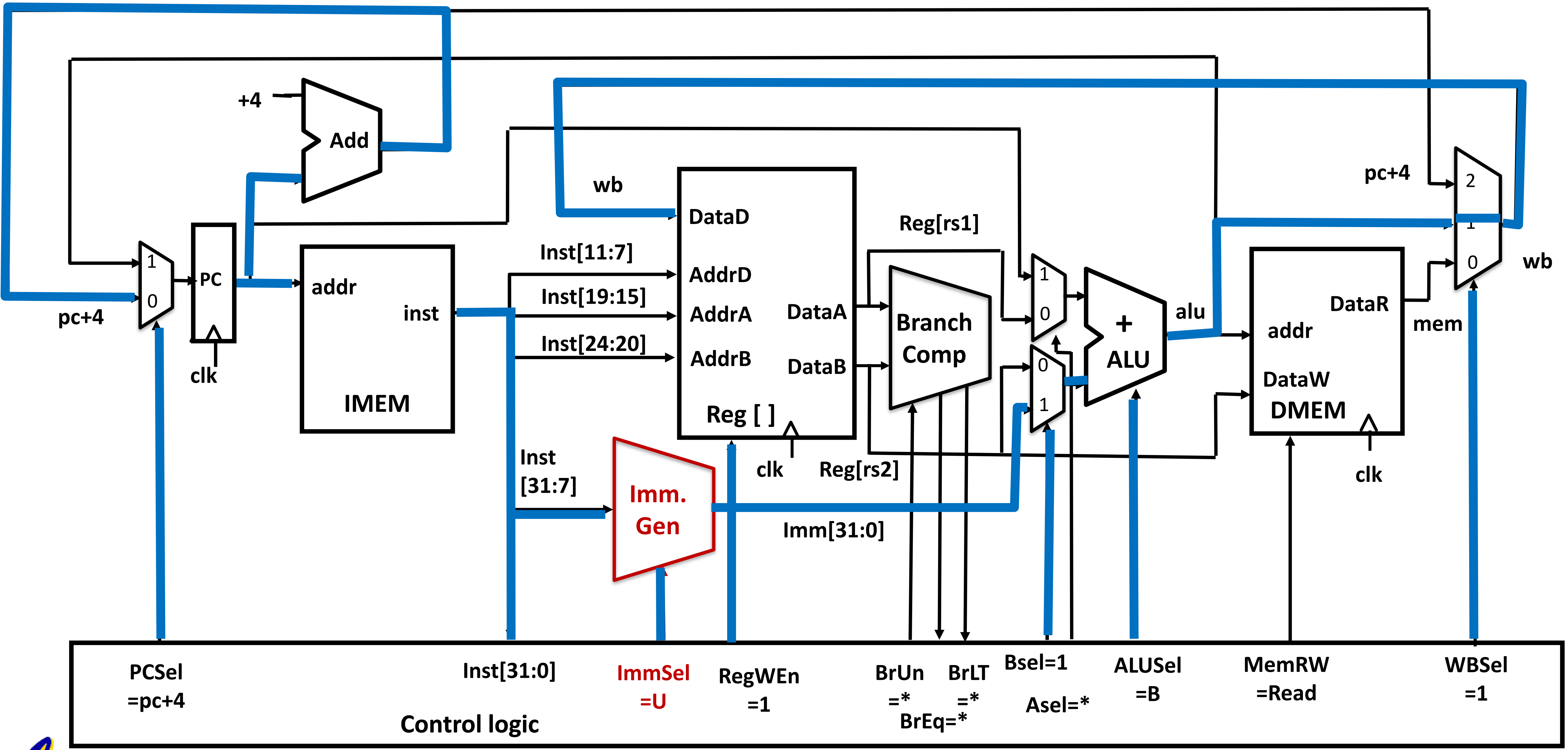

+4
Add
pc+4
1
0
PC
clk
addr
inst
IMEM
Inst[11:7]
Inst[19:15]
Inst[24:20]
DataD
AddrD
AddrA
AddrB
DataA
DataB
Reg [ ]
clk
Inst
[31:7]
Imm.
Gen
Imm[31:0]
Reg[rs1]
Reg[rs2]
Branch
Comp
1
0
0
1
+
ALU
alu
addr
DataR
DataW
DMEM
clk
mem
pc+4
2
1
0
wb
PCSel
=taken
Inst[31:0]
ImmSel
=J
RegWEn
=1
BrUn
=*
BrLT
=*
BrEq=*
Bsel=1
Asel=1
ALUSel
=add
MemRW
=Read
WBSel
=2
Control logic

U-Format for “Upper Immediate” Instructions

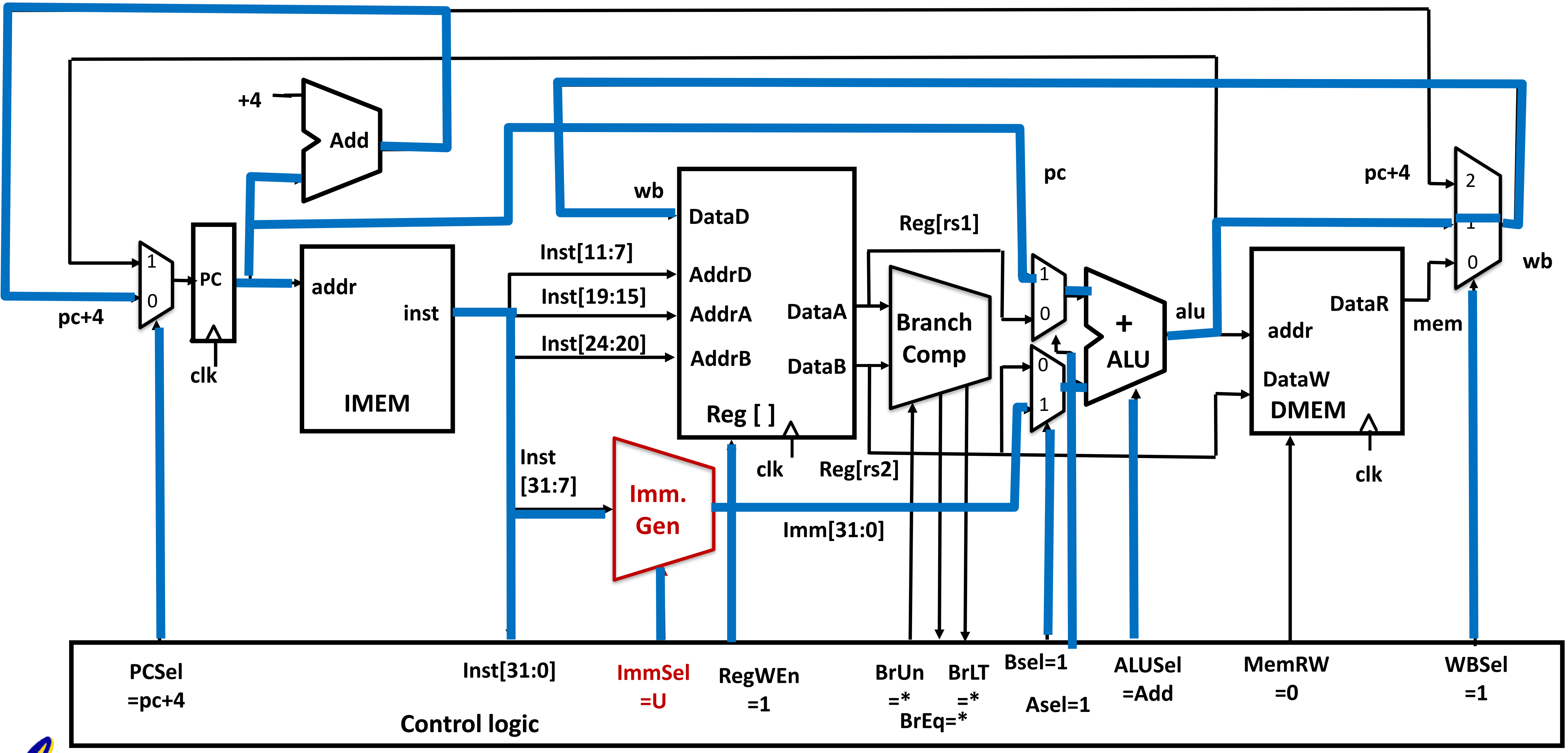
31 12	11 7	6 0
imm[31:12]	rd	opcode
20	5	7
U-immediate[31:12]	dest	LUI
U-immediate[31:12]	dest	AUIPC

- 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
  - AUIPC – Add Upper Immediate to PC

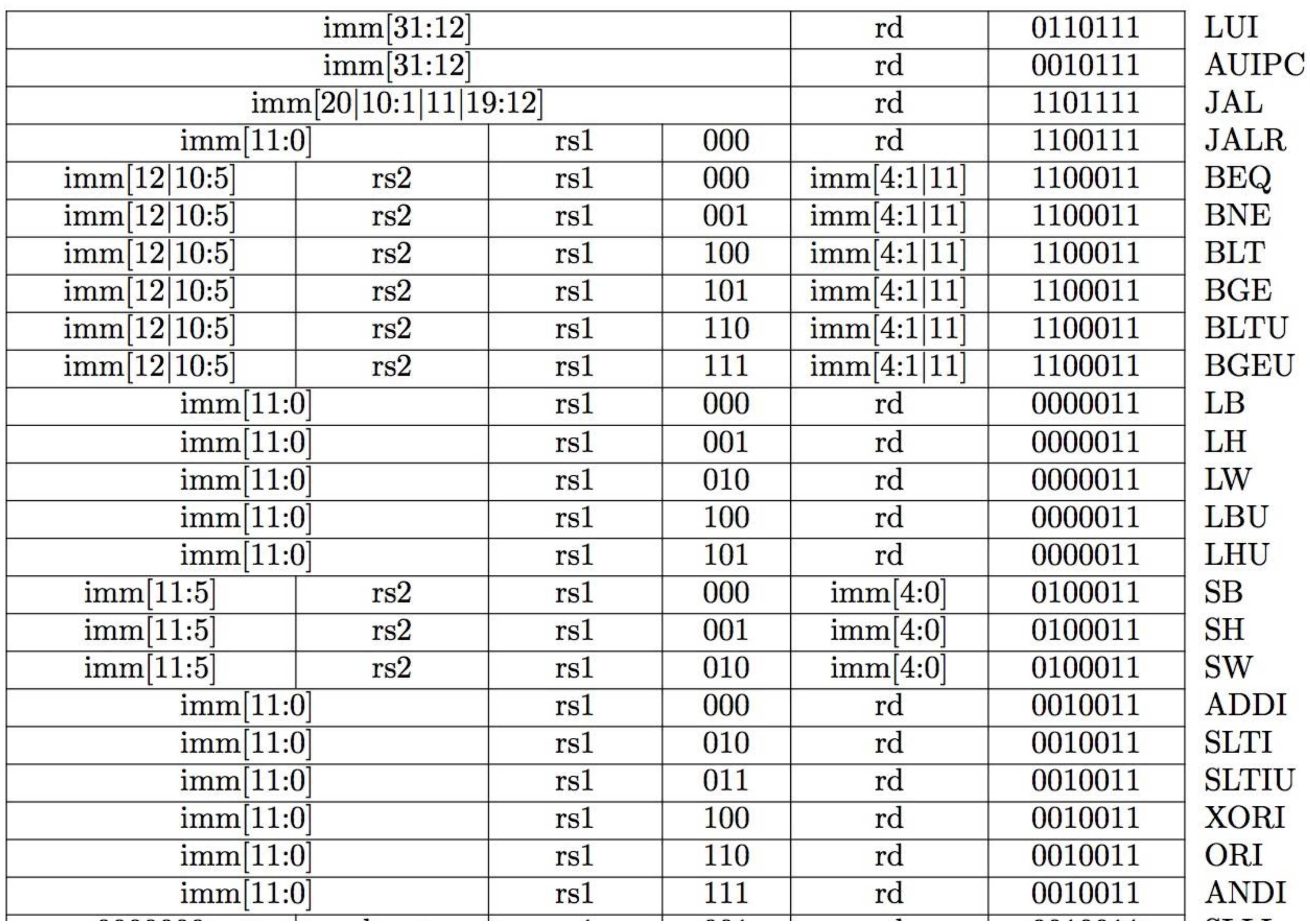
Implementing `LUI`



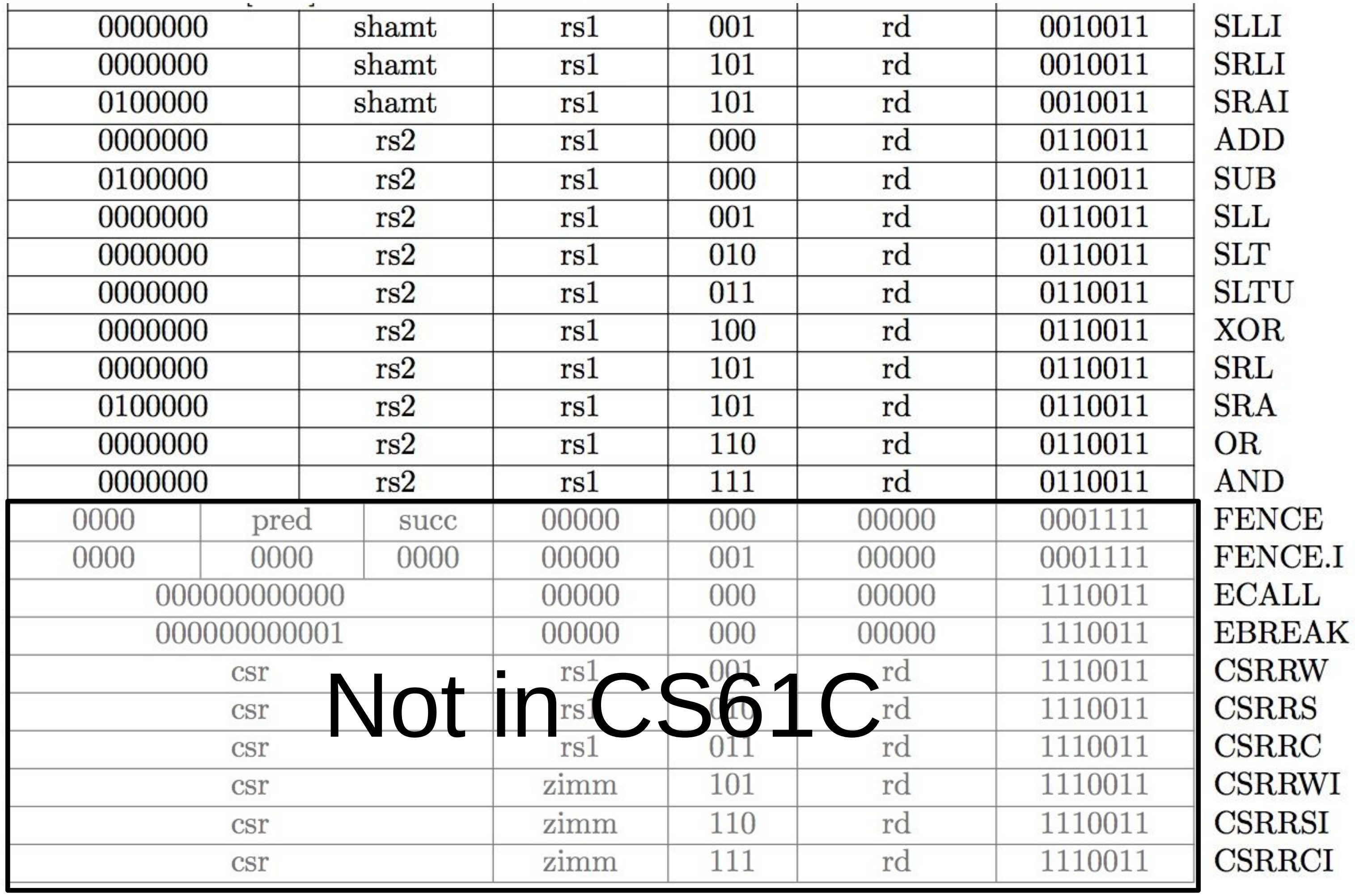
Implementing AUIPC



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

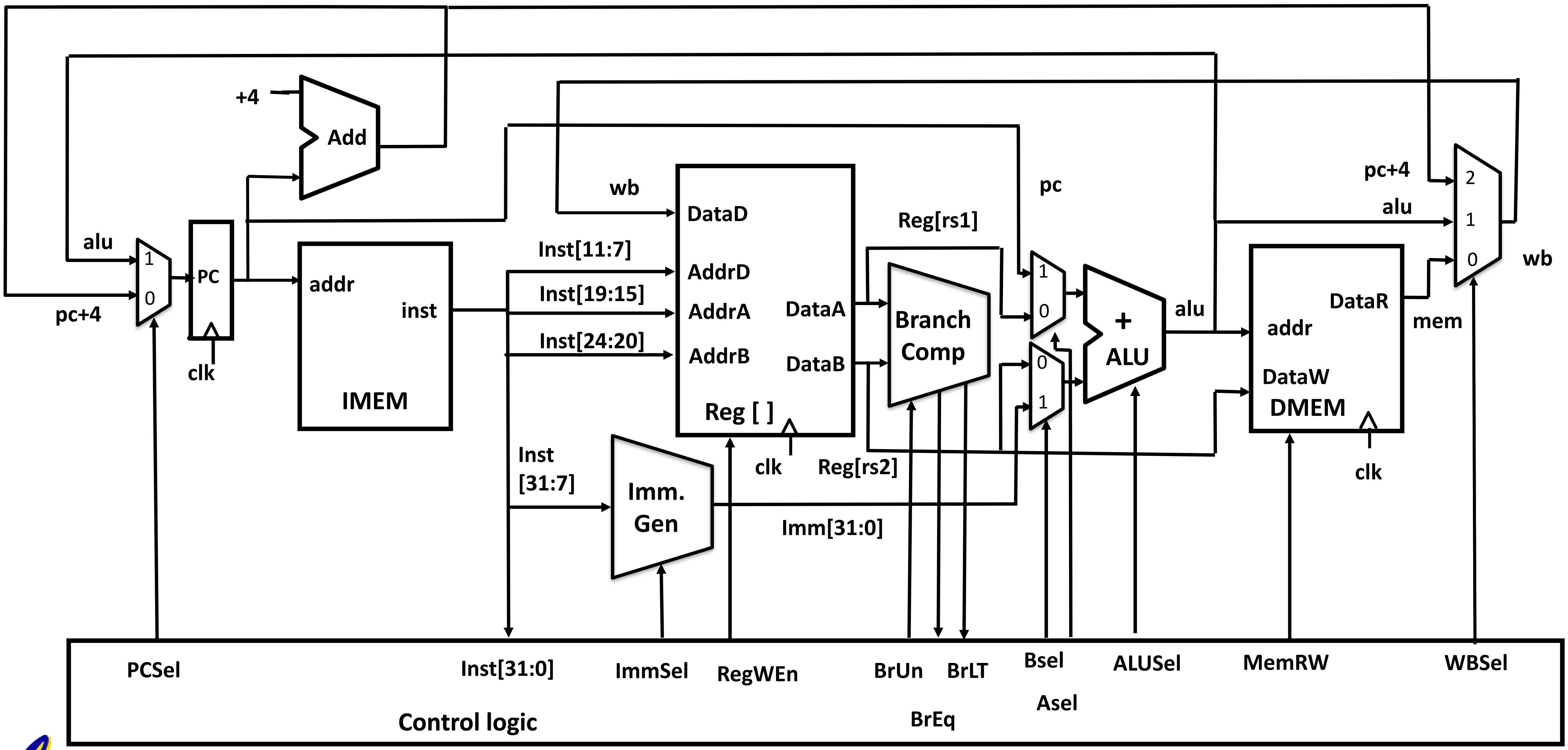


0000000		shamt	rs1	001	rd	0010011	SLLI
0000000		shamt	rs1	101	rd	0010011	SRLI
0100000		shamt	rs1	101	rd	0010011	SRAI
0000000		rs2	rs1	000	rd	0110011	ADD
0100000		rs2	rs1	000	rd	0110011	SUB
0000000		rs2	rs1	001	rd	0110011	SLL
0000000		rs2	rs1	010	rd	0110011	SLT
0000000		rs2	rs1	011	rd	0110011	SLTU
0000000		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

Not in CS61C

- RV32I has 47 instructions
- 37 instructions are enough to run any C program

Complete RV32I Datapath!



"And In conclusion…"

- We have designed a complete 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