

# EECS151/251A

## Introduction to Digital Design and ICs

### Sophia Shao

## Lecture 8: RISC-V Datapath I



### NVIDIA and SoftBank Group Announce Termination of NVIDIA's Acquisition of Arm Limited

SANTA CLARA, Calif., and TOKYO – Feb. 7, 2022 – NVIDIA and SoftBank Group Corp. (“SBG” or “SoftBank”) today announced the termination of the previously announced transaction whereby NVIDIA would acquire Arm Limited (“Arm”) from SBG. The parties agreed to terminate the Agreement because of significant regulatory challenges preventing the consummation of the transaction, despite good faith efforts by the parties. Arm will now start preparations for a public offering.



<https://nvidianews.nvidia.com/news/nvidia-and-softbank-group-announce-termination-of-nvidias-acquisition-of-arm-limited>

# Summary

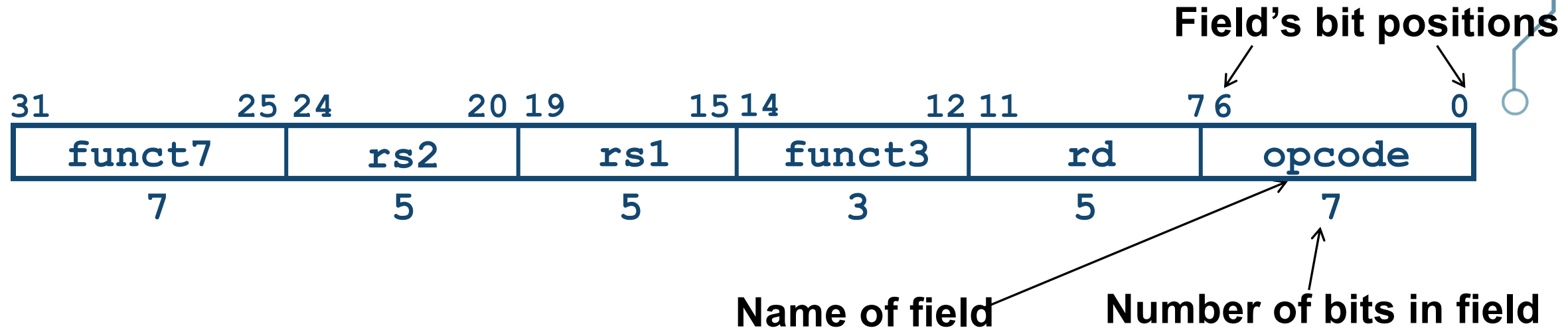
- State machines:
  - Specify circuit function
  - Draw state transition diagram
  - Write down symbolic state-transition table
  - Assign encodings (bit patterns) to symbolic states
  - Code as Verilog behavioral description
- RISC-V processor
  - A large state machine
  - Datapath Elements



## • RISC-V Datapath & Control

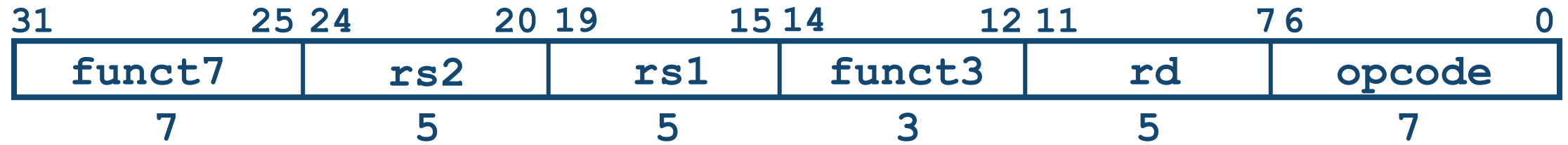
- R-type
- I-type
- S-type
- B-type
- J-type
- U-type
- Control Logic

# R-Format Instruction Layout



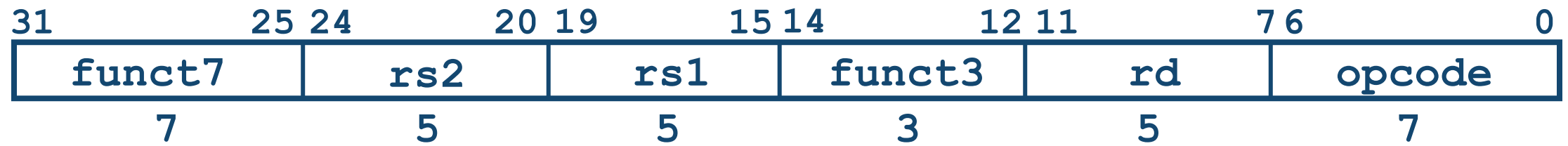
- 32-bit instruction word divided into six fields of varying numbers of bits each:  $7+5+5+3+5+7 = 32$
- Examples
  - **opcode** is a 7-bit field that lives in bits 6-0 of the instruction
  - **rs2** is a 5-bit field that lives in bits 24-20 of the instruction

# R-Format Instructions opcode/funct fields



- **opcode**: partially specifies what instruction it is
  - Note: This field is equal to **0110011**<sub>two</sub> for all R-Format register-register arithmetic instructions
- **funct7+funct3**: combined with **opcode**, these two fields describe what operation to perform

# R-Format Instructions register specifiers

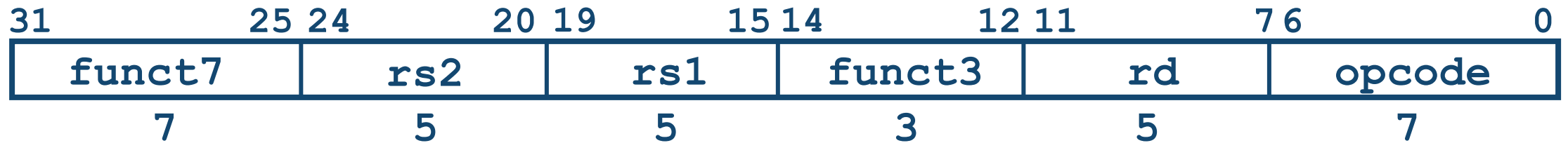


- rs1 (**S**ource **R**egister #1): specifies register containing first operand
- rs2 : specifies second register operand
- rd (**D**estination **R**egister): specifies register which will receive result of computation
- Each register field holds a 5-bit unsigned integer (0-31) corresponding to a register number (**x0-x31**)

# R-Format Example

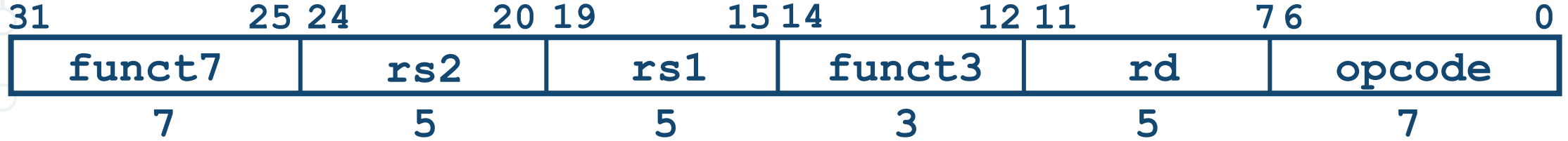
- RISC-V Assembly Instruction:

`add x18,x19,x10`



`add`      `rs2=10 rs1=19`      `add`      `rd=18`      `Reg-Reg OP`

# Implementing the `add` instruction



`add`      `rs2`      `rs1`      `add`      `rd`      Reg-Reg OP

`add rd, rs1, rs2`

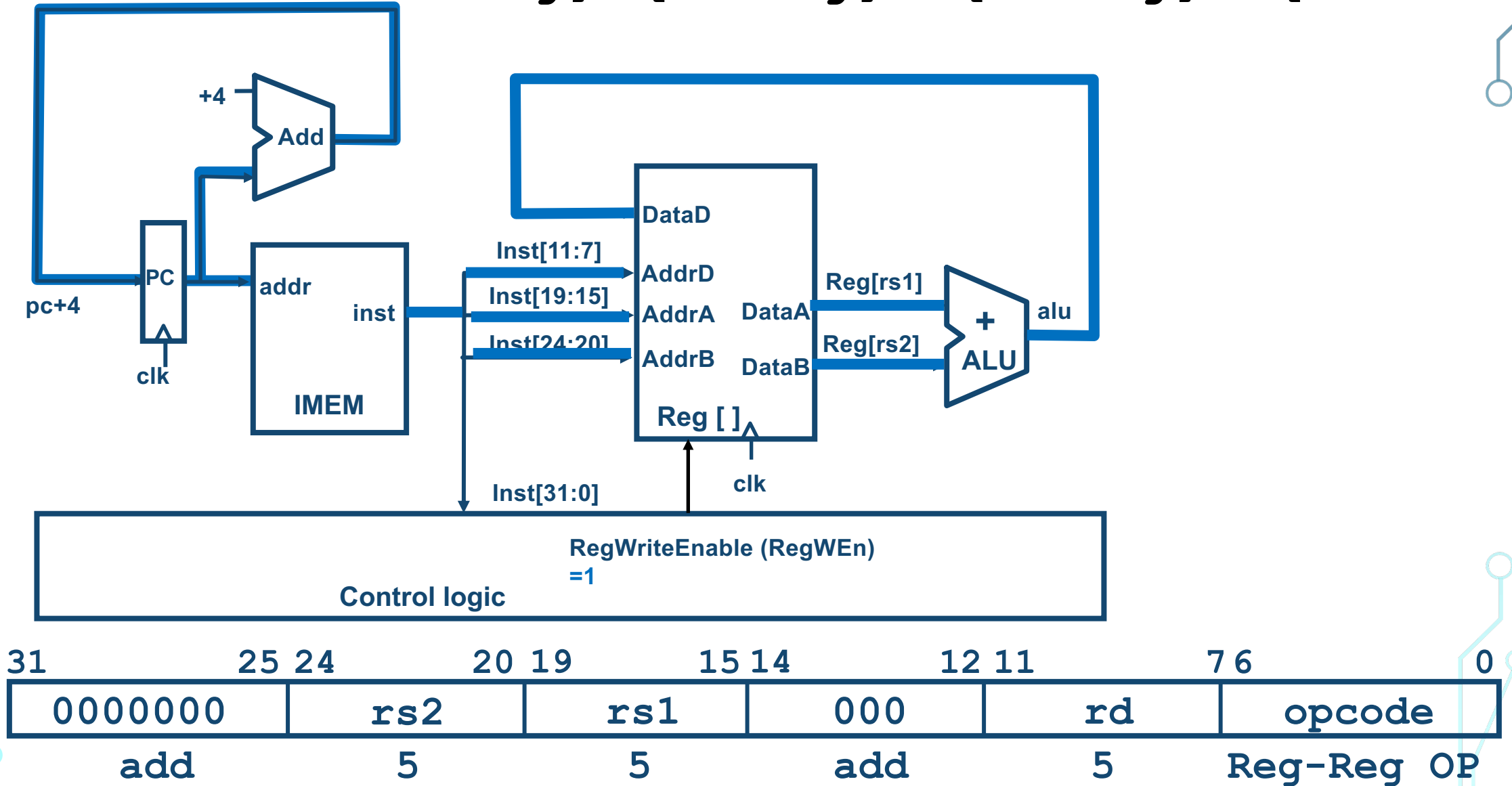
- Instruction makes two changes to machine's state:
  - $\text{Reg}[\text{rd}] = \text{Reg}[\text{rs1}] + \text{Reg}[\text{rs2}]$
  - $\text{PC} = \text{PC} + 4$



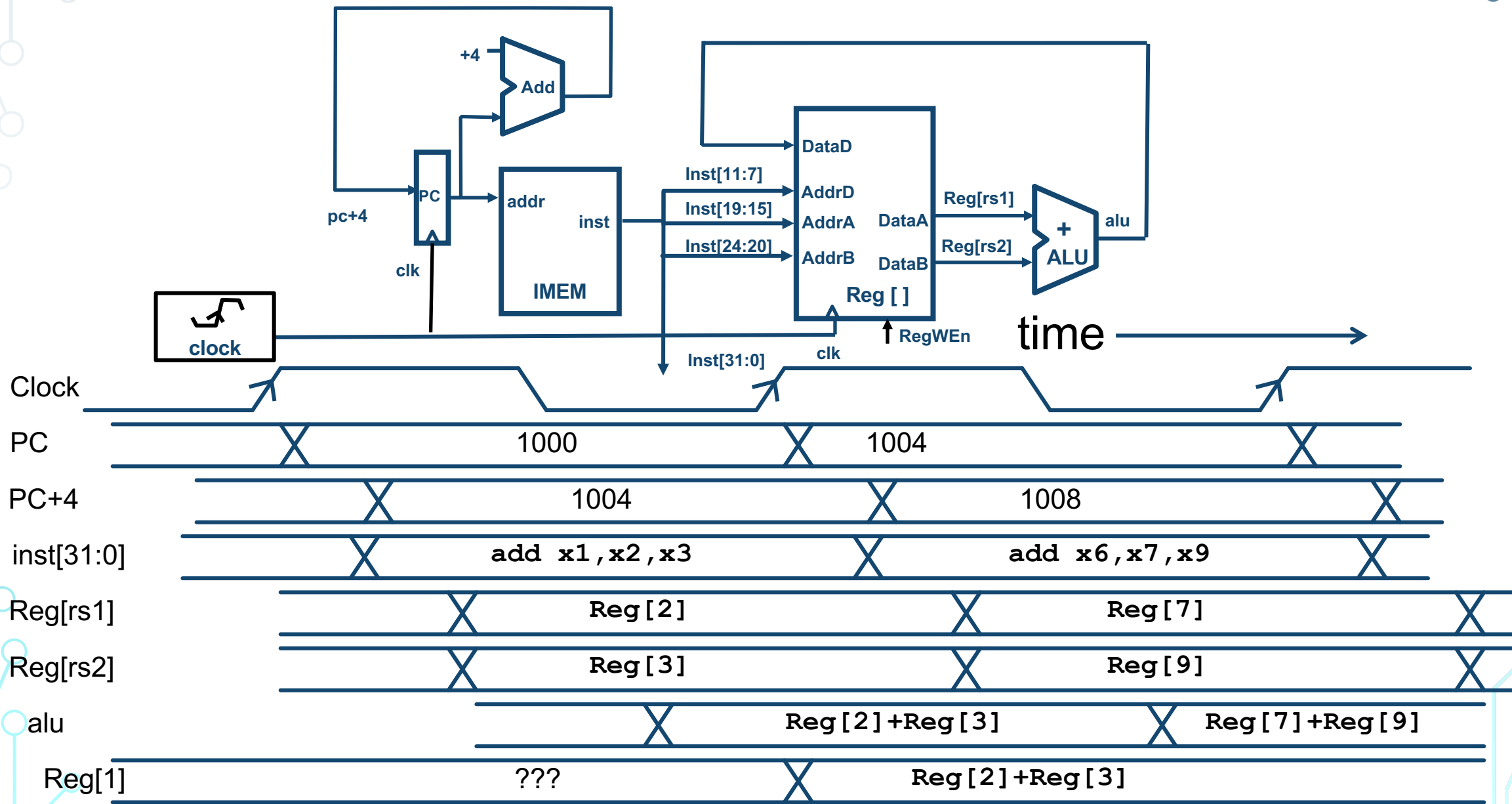
# Datapath for add

$$PC = PC + 4$$

$$Reg[rd] = Reg[rs1] + Reg[rs2]$$



# Timing Diagram for add



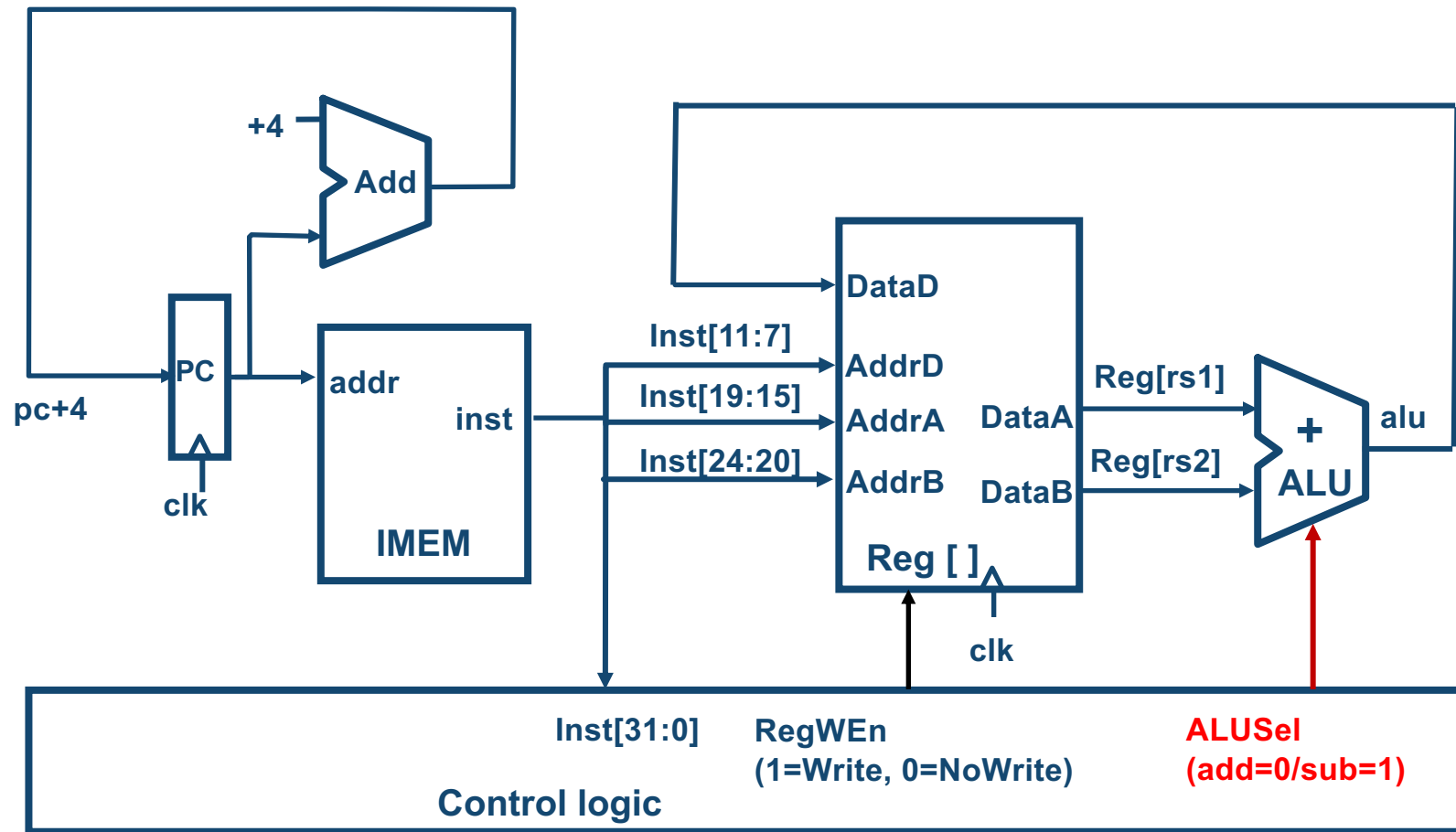
# Implementing the **sub** instruction

31	25	24	20	19	15	14	12	11	7	6	0		
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

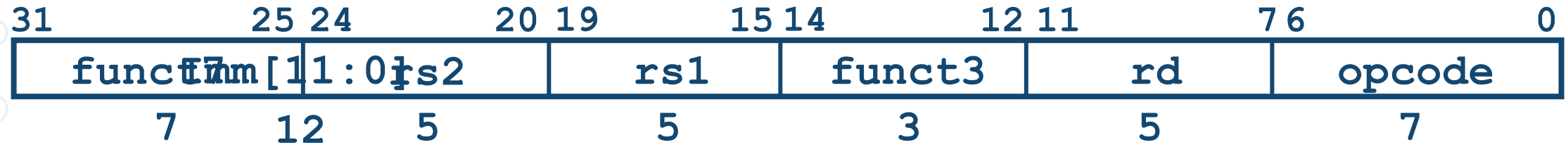
- All implemented by decoding funct3 and funct7 fields and selecting appropriate ALU function



## • RISC-V Datapath & Control

- R-type
- I-type
- S-type
- B-type
- J-type
- U-type
- Control Logic

# I-Format Instruction Layout



- Only one field is different from R-format, rs2 and funct7 replaced by 12-bit signed immediate, **imm[11:0]**
- Remaining fields (rs1, funct3, rd, opcode) same as before
- imm[11:0] can hold values in range  $[-2048_{\text{ten}}, +2047_{\text{ten}}]$
- Immediate is always sign-extended to 32-bits before use in an arithmetic operation
- Other instructions handle immediate > 12 bits

# All RV32 I-format Arithmetic Instructions

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

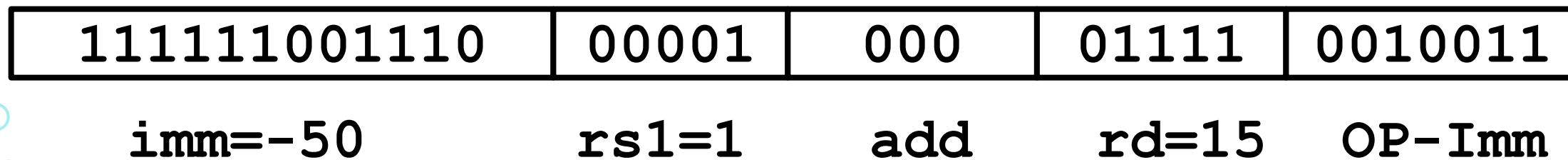
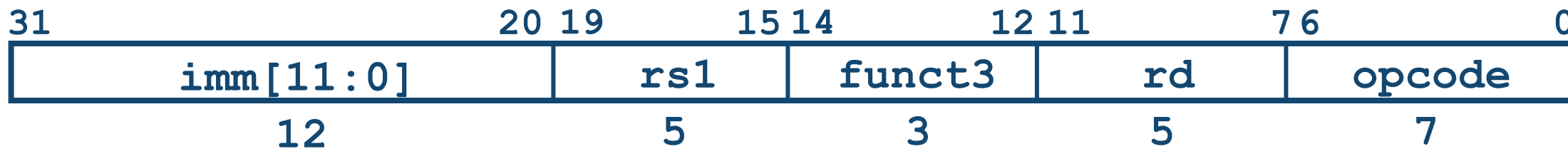
The same Inst[30] immediate bit is used to distinguish “shift right logical” (SRLI) from “shift right arithmetic” (SRAI)

“Shift-by-immediate” instructions only use lower 5 bits of the immediate value for shift amount (can only shift by 0-31 bit positions)

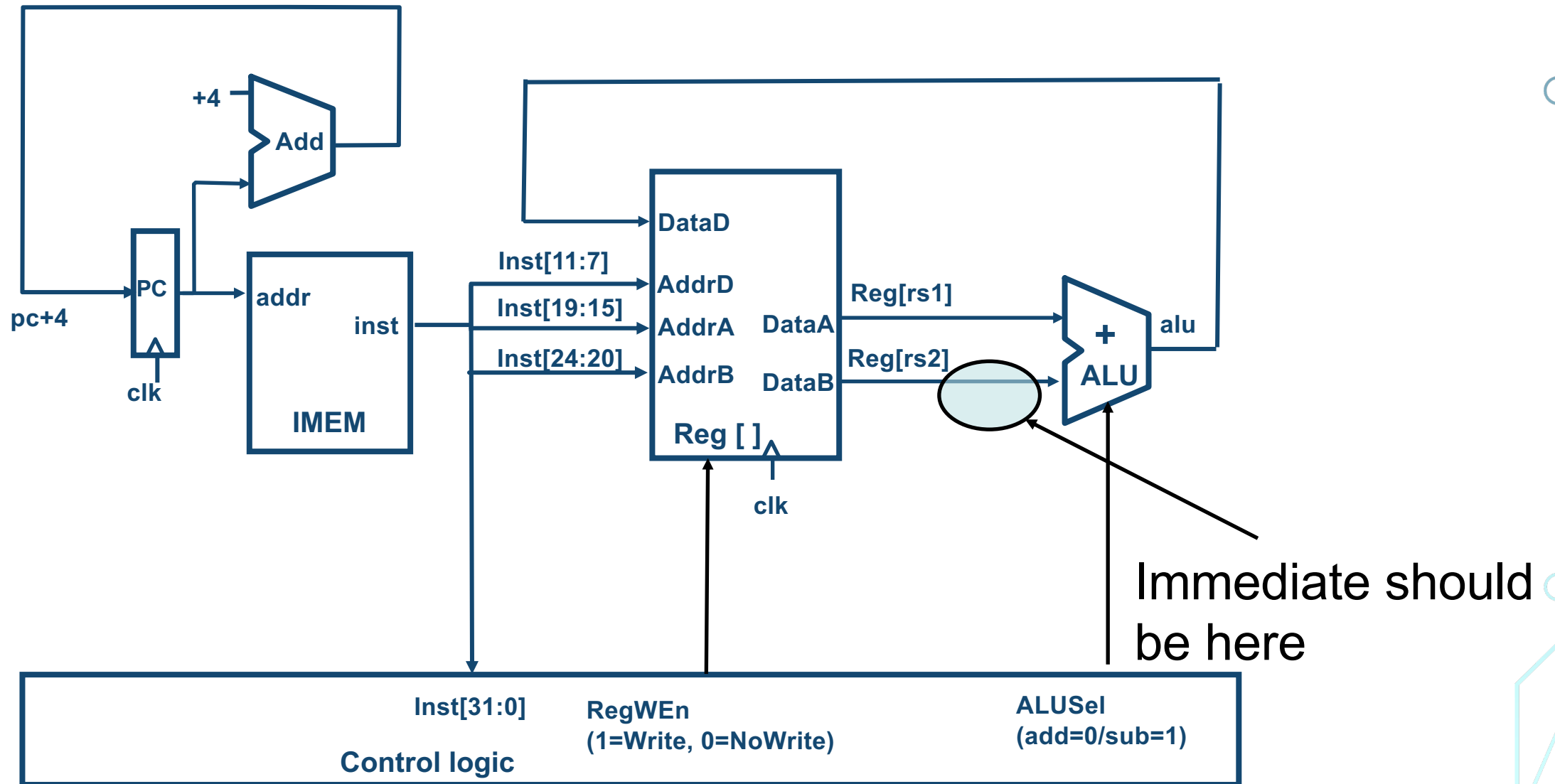


# Implementing I-Format - `addi` instruction

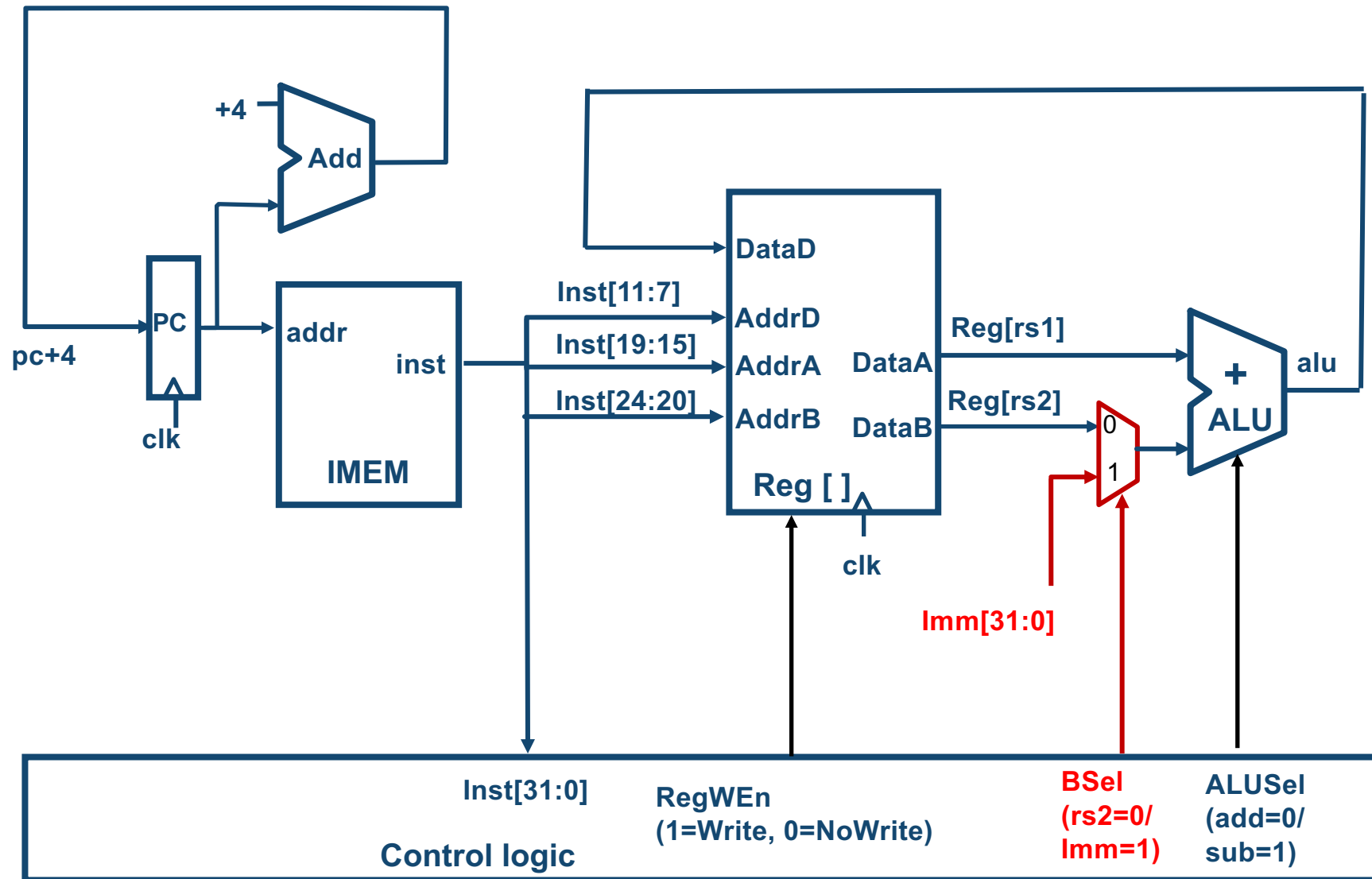
`addi x15,x1,-50`



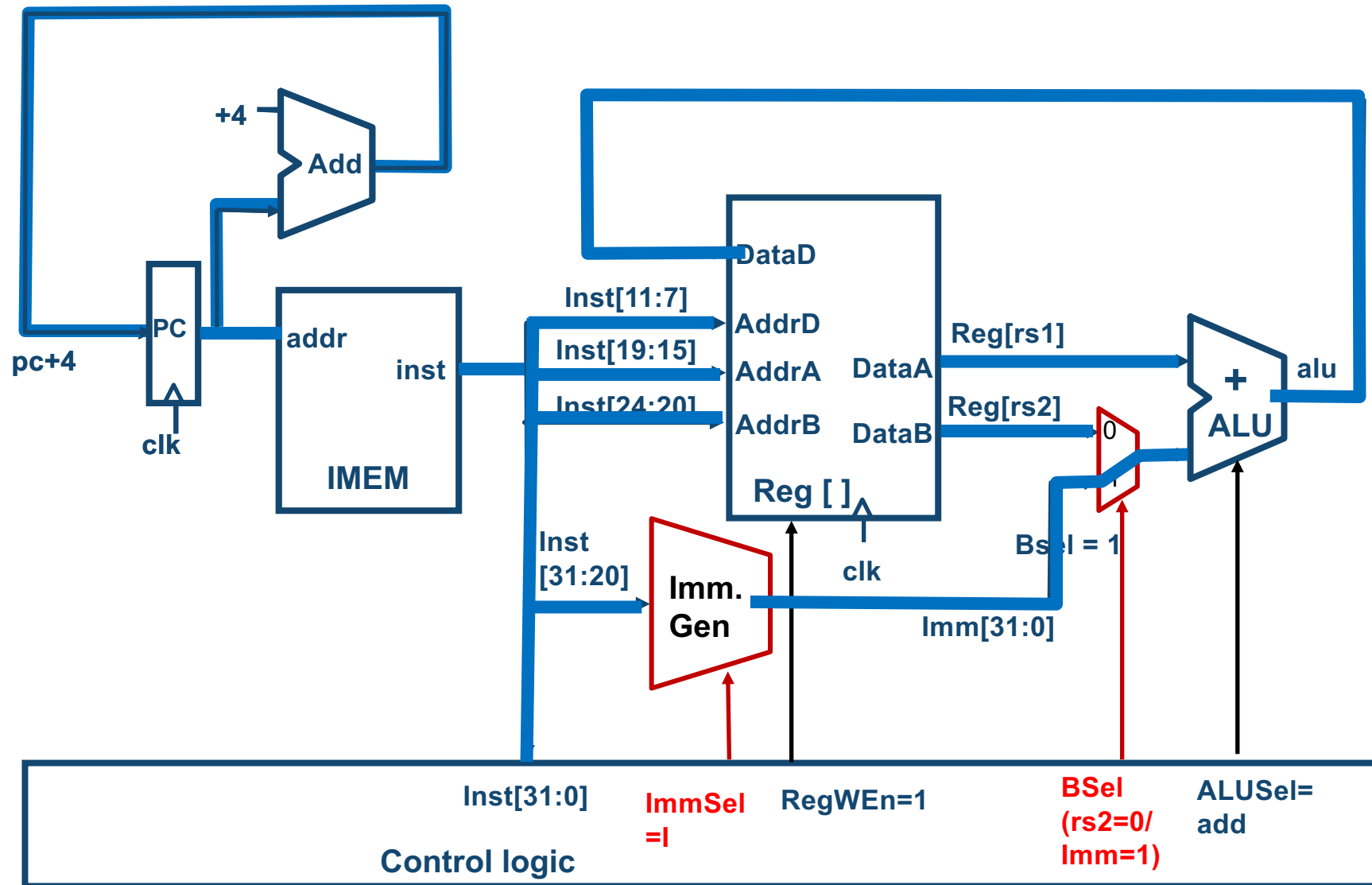
# Datapath for add/sub



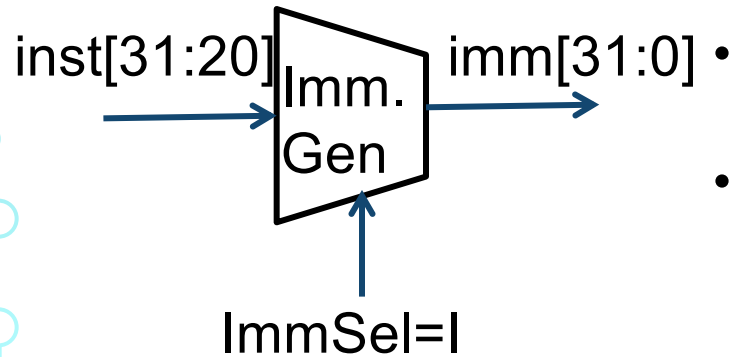
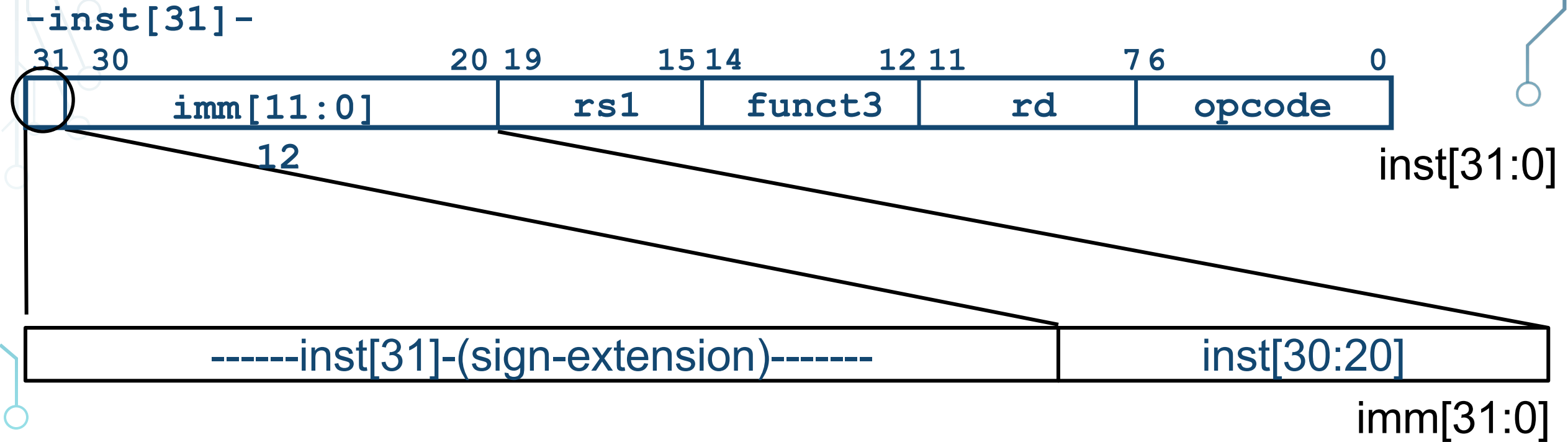
# Adding addi to Datapath



# Adding addi to Datapath

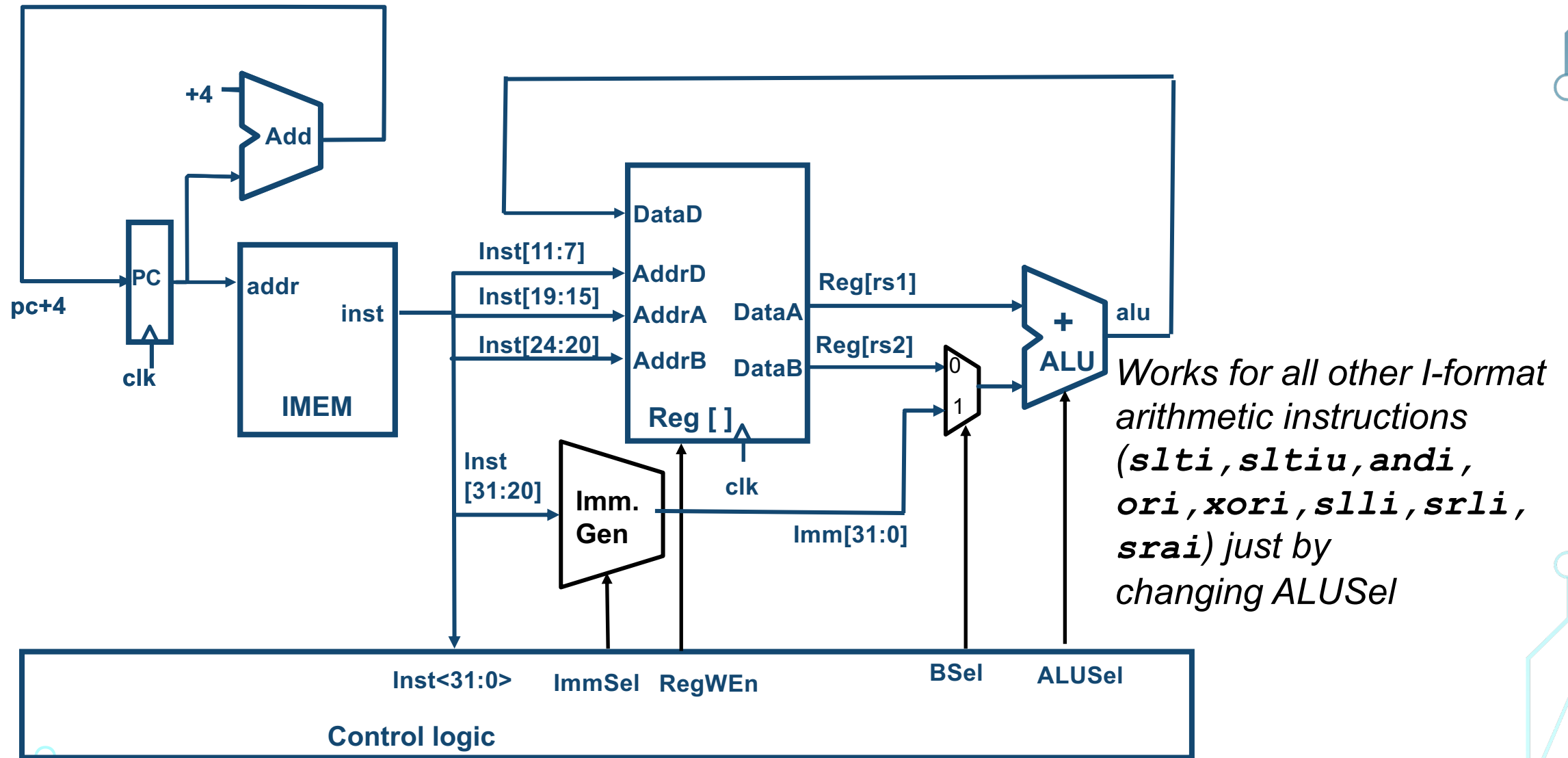


# I-Format immediates

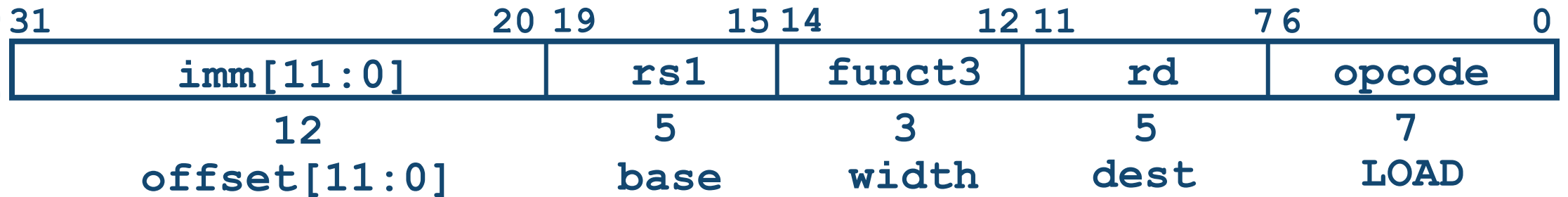


- High 12 bits of instruction (`inst[31:20]`) copied to low 12 bits of immediate (`imm[11:0]`)
- Immediate is sign-extended by copying value of `inst[31]` to fill the upper 20 bits of the immediate value (`imm[31:12]`)

# R+I Datapath



# Load Instructions are also I-Type

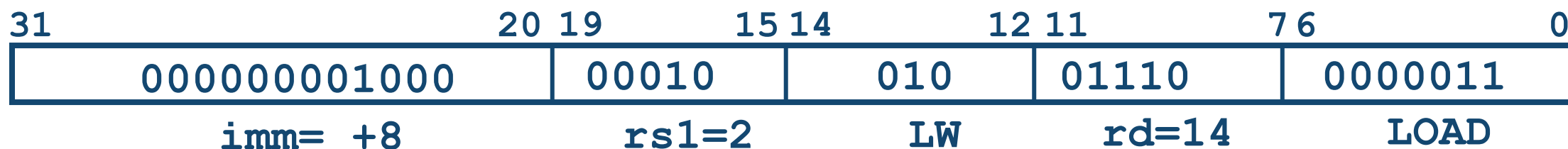
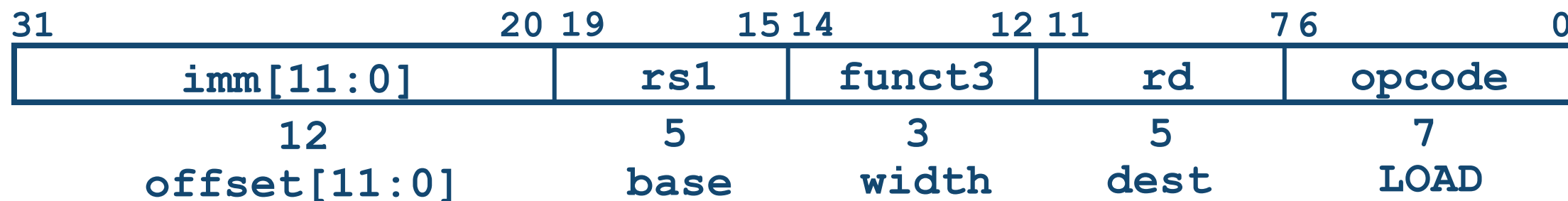


- $\text{Reg}[\text{rd}] = \text{Mem}[\text{Reg}[\text{rs1}] + \text{offset}]$ 
  - The 12-bit signed immediate is added to the base address in register `rs1` to form the memory address
    - This is very similar to the add-immediate operation but used to create address not to create final result
  - The value loaded from memory is stored in register `rd`

# Add lw to Datapath

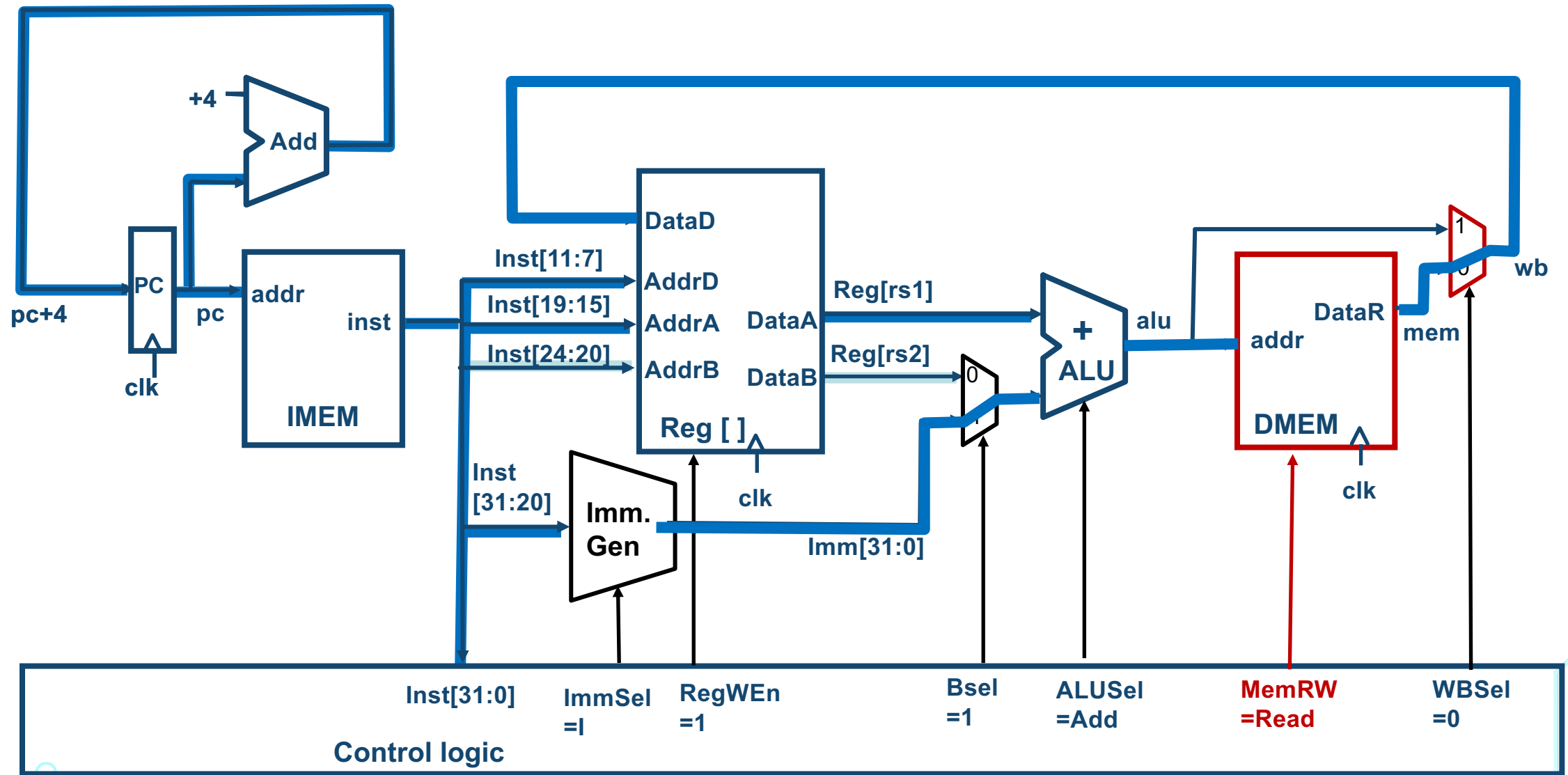
- RISC-V Assembly Instruction (I-type):

**lw x14, 8(x2)**





# Adding 1w to Datapath



# All RV32 Load Instructions

<code>imm[11:0]</code>	<code>rs1</code>	000	<code>rd</code>	0000011	lb
<code>imm[11:0]</code>	<code>rs1</code>	001	<code>rd</code>	0000011	lh
<code>imm[11:0]</code>	<code>rs1</code>	010	<code>rd</code>	0000011	lw
<code>imm[11:0]</code>	<code>rs1</code>	100	<code>rd</code>	0000011	lbu
<code>imm[11:0]</code>	<code>rs1</code>	101	<code>rd</code>	0000011	lhu

funct3 field encodes size and 'signedness' of load data

- lbu: load unsigned byte; lh: load halfword (halfword == 16bits == 2bytes)
- 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 for load extend, similar to sign extension for immediates

# Administrivia

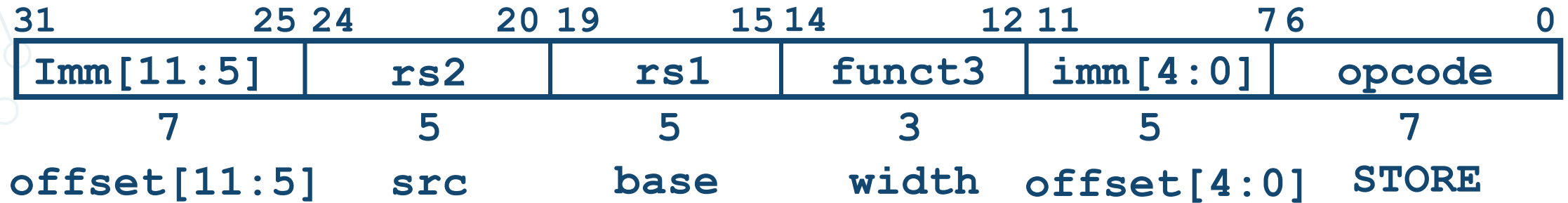
- Lab 4 starts this week.
  - Lab 3 deadline extended by 1 week.
  - 2-week lab moving forward.
- Guest lecture 2/24
  - FPGA Usecases
- In-class midterm
  - Tuesday 3/1
  - Will cover topics by next week (pipelining)



## • RISC-V Datapath & Control

- R-type
- I-type
- S-type
- B-type
- J-type
- U-type
- Control Logic

# S-Format Used for Stores

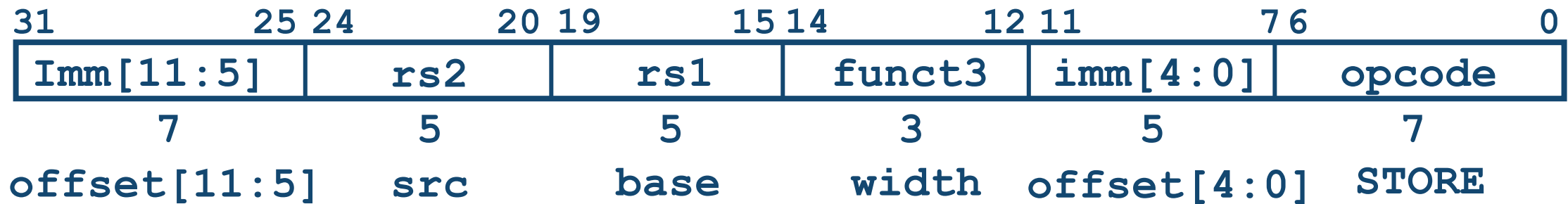


- $\text{Mem}[\text{Reg}[\text{rs1}] + \text{offset}] = \text{Reg}[\text{rs2}]$ 
  - Store needs to read two registers, rs1 for base memory address, and rs2 for data to be stored, as well immediate offset!
  - Note that stores don't write a value to the register file, **no rd!**
- Immediate in two parts:
  - Can't have both rs2 and immediate in same place as other instructions!
  - RISC-V design decision is move low 5 bits of immediate to where rd field was in other instructions – keep rs1/rs2 fields in same place
    - register names more critical than immediate bits in hardware design

# Adding `sw` Instruction

- `sw`: Reads two registers, `rs1` for base memory address, and `rs2` for data to be stored, together with immediate offset.

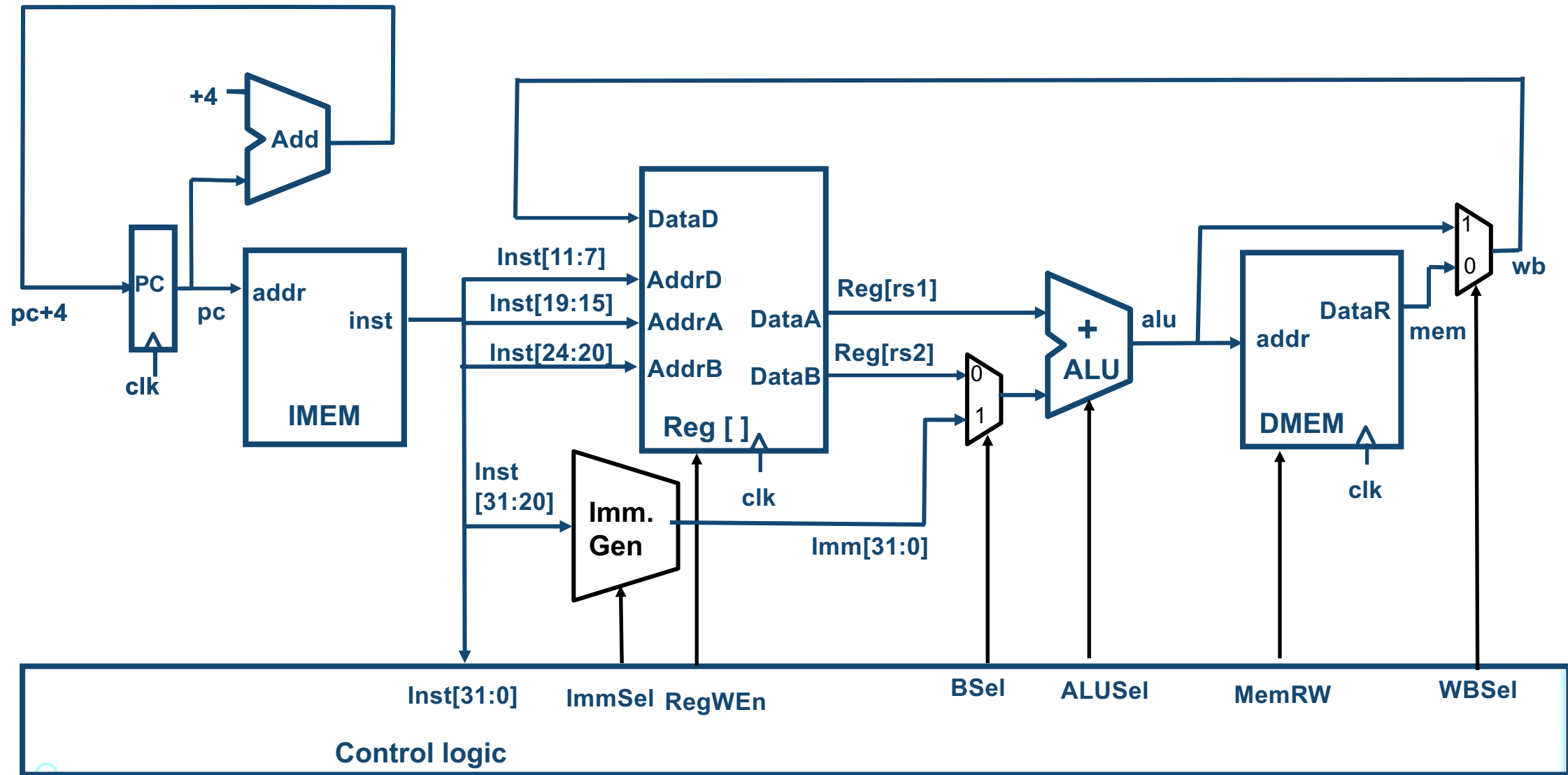
`sw x14, 8(x2)`



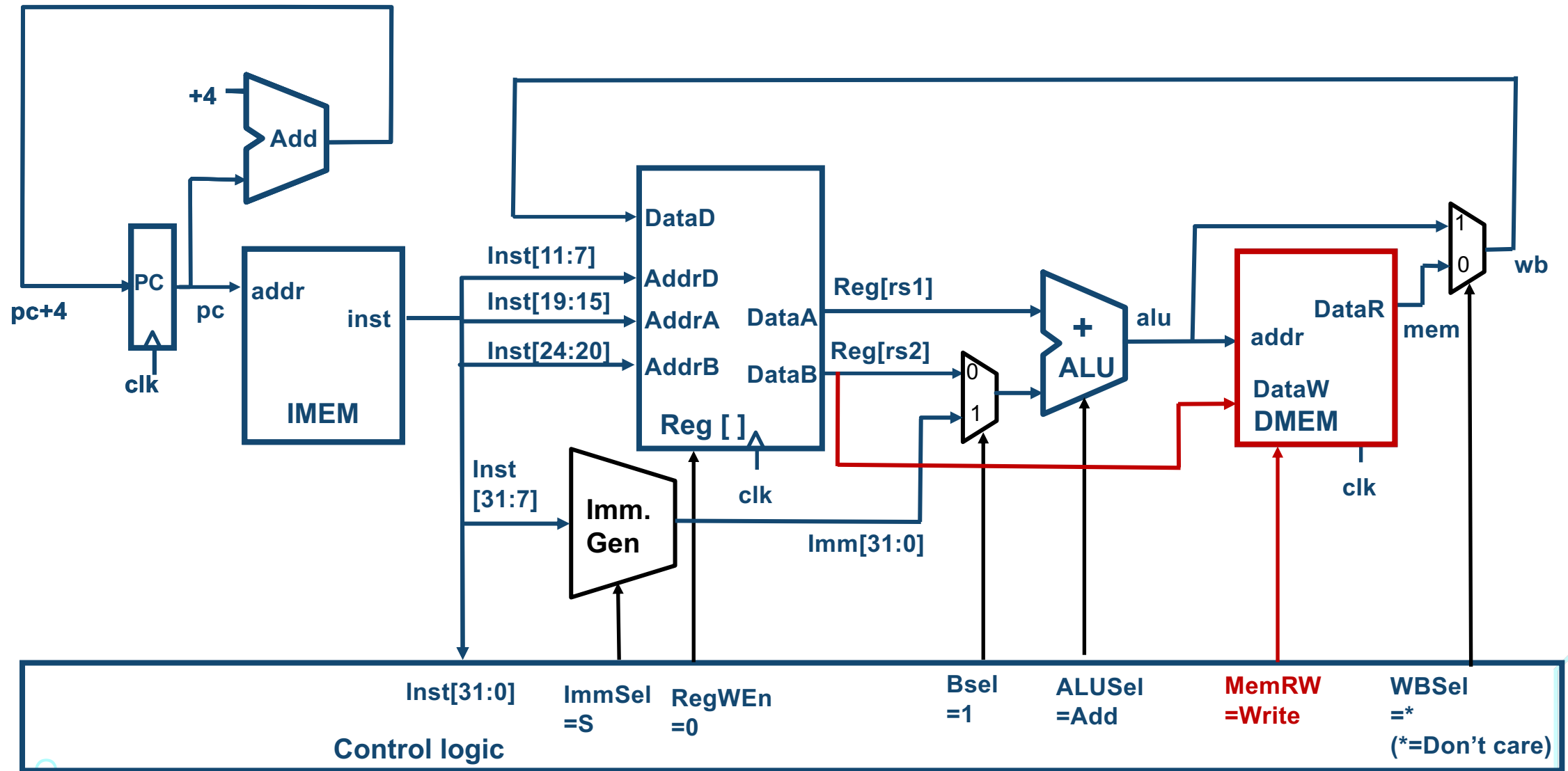
offset[11:5] = 0      rs2=14      rs1=2      SW      offset[4:0] = 8      STORE

0000000 01000 combined 12-bit offset = 8

# Datapath with 1w

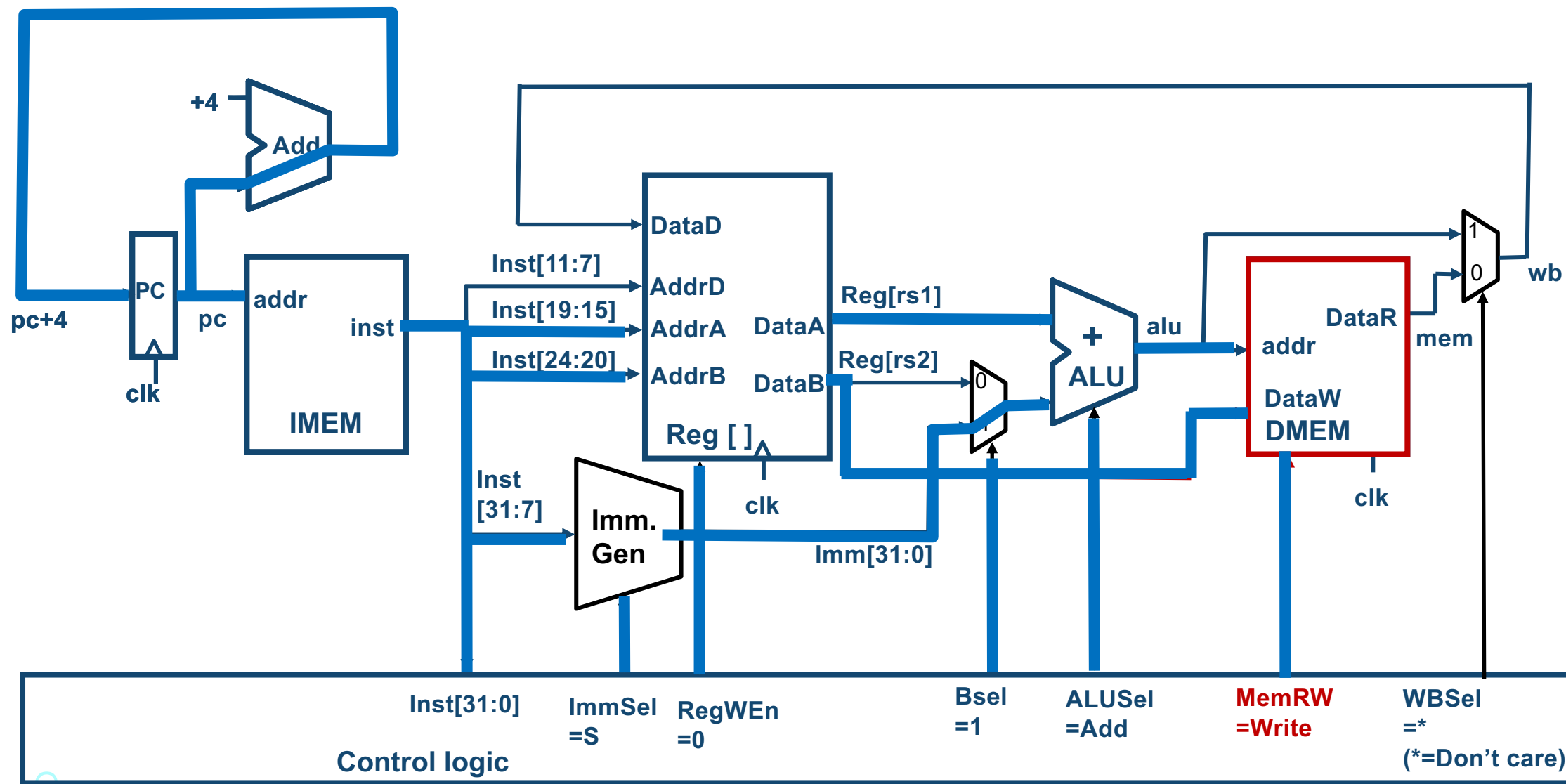


# Adding sw to Datapath





# Adding sw to Datapath



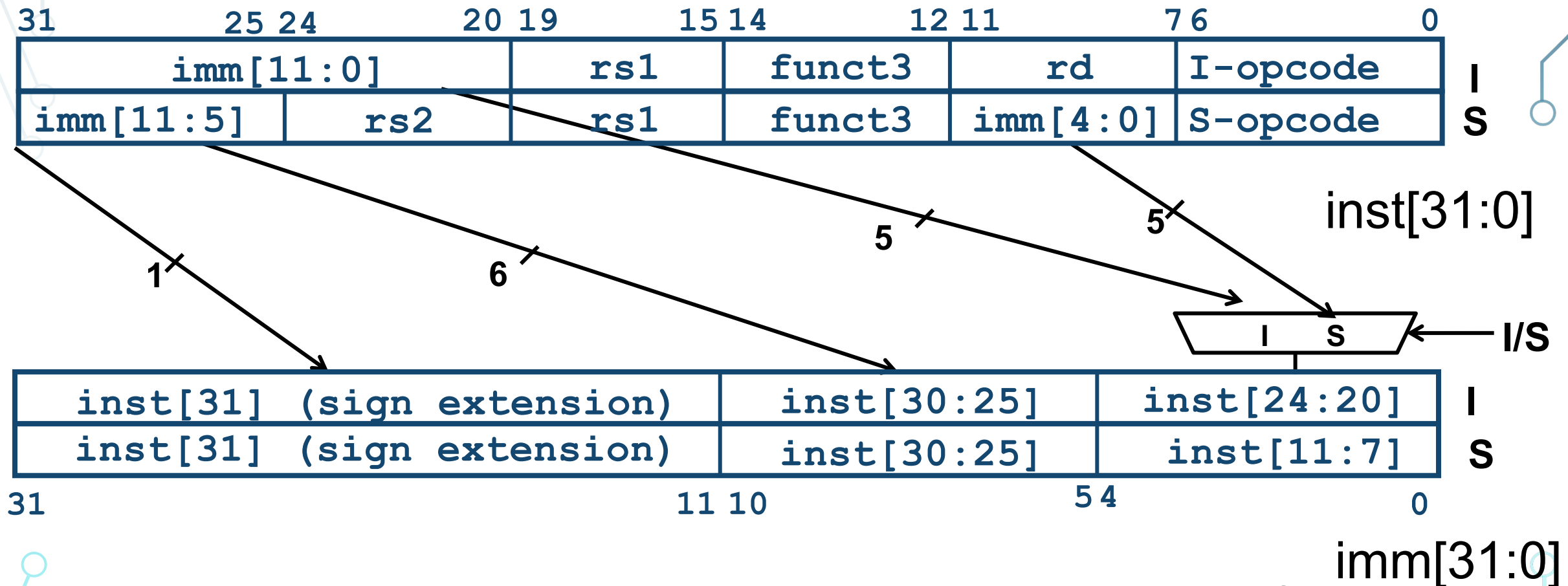
# All RV32 Store Instructions

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

width

- Store byte, halfword, word

# I+S Immediate Generation



- 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



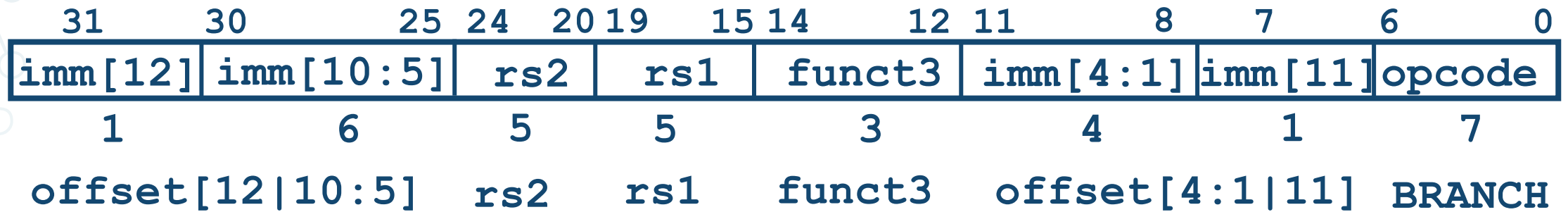
## • RISC-V Datapath & Control

- R-type
- I-type
- S-type
- **B-type**
- J-type
- U-type
- **Control Logic**

# B-Format - RISC-V Conditional Branches

- E.g., **BEQ x1, x2, Label**
- Branches read two registers but don't write a register (similar to stores)
- How to encode label, i.e., where to branch to?

# Implementing Branches



- B-format is similar to S-format, with two register sources (rs1/rs2) and a 12-bit immediate
- The 12 immediate bits encode 13-bit signed byte offsets (lowest bit of offset is always zero, so no need to store it)
- But now immediate represents values  $-2^{12}$  to  $+2^{12}$  in 2-byte increments

# Branch Example, complete encoding

**beq**    **x19,x10,**    **offset = 16 bytes**

13-bit immediate, imm[12:0], with value 16

0000000010000

imm[0] discarded,  
→ always zero

imm[12]

imm[11]

0	000000	01010	10011	000	1000	0	1100011
---	--------	-------	-------	-----	------	---	---------

imm[10:5] rs2=10    rs1=19    BEQ    imm[4:1]    BRANCH

# 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

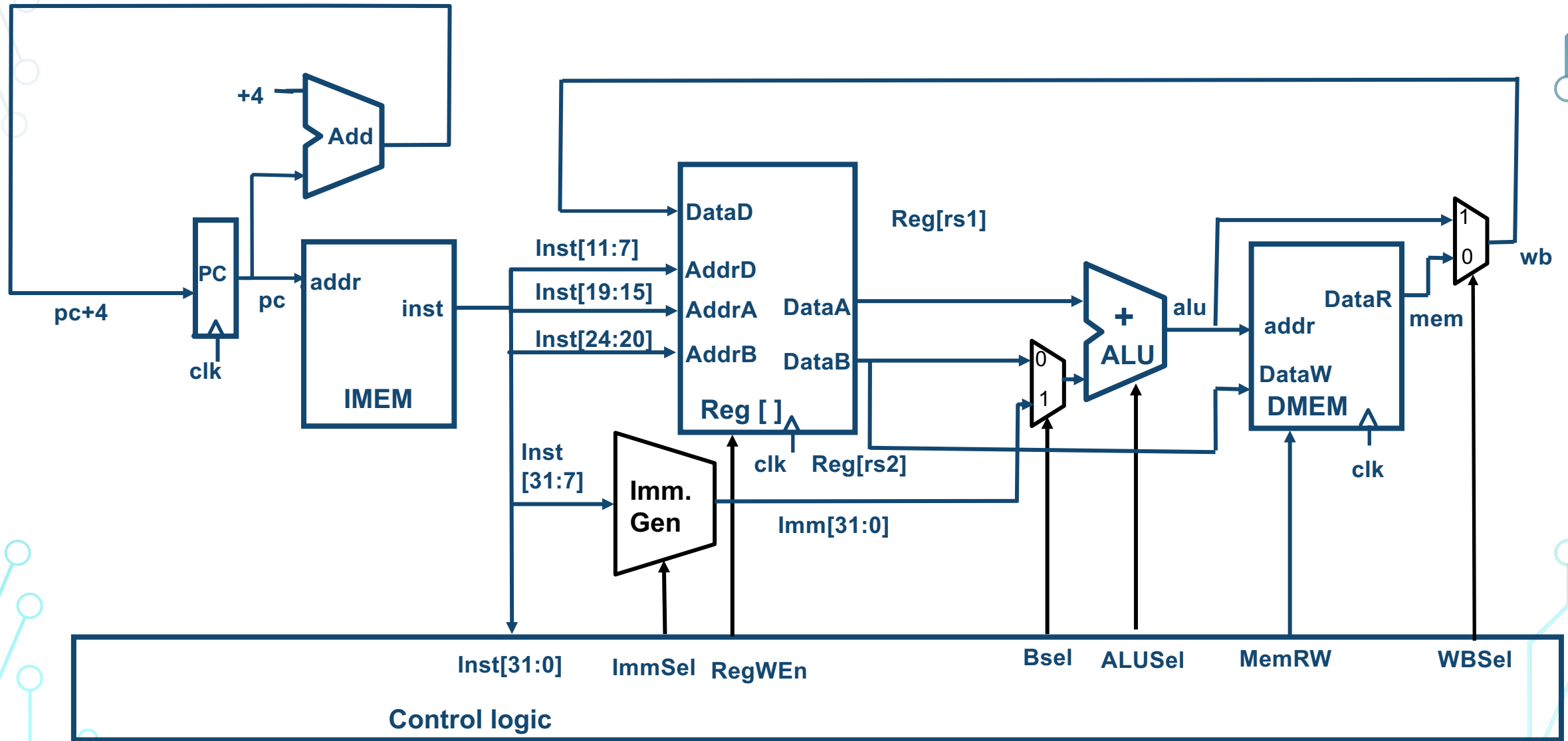
Only bit 7 of instruction changes role in  
immediate between S and B



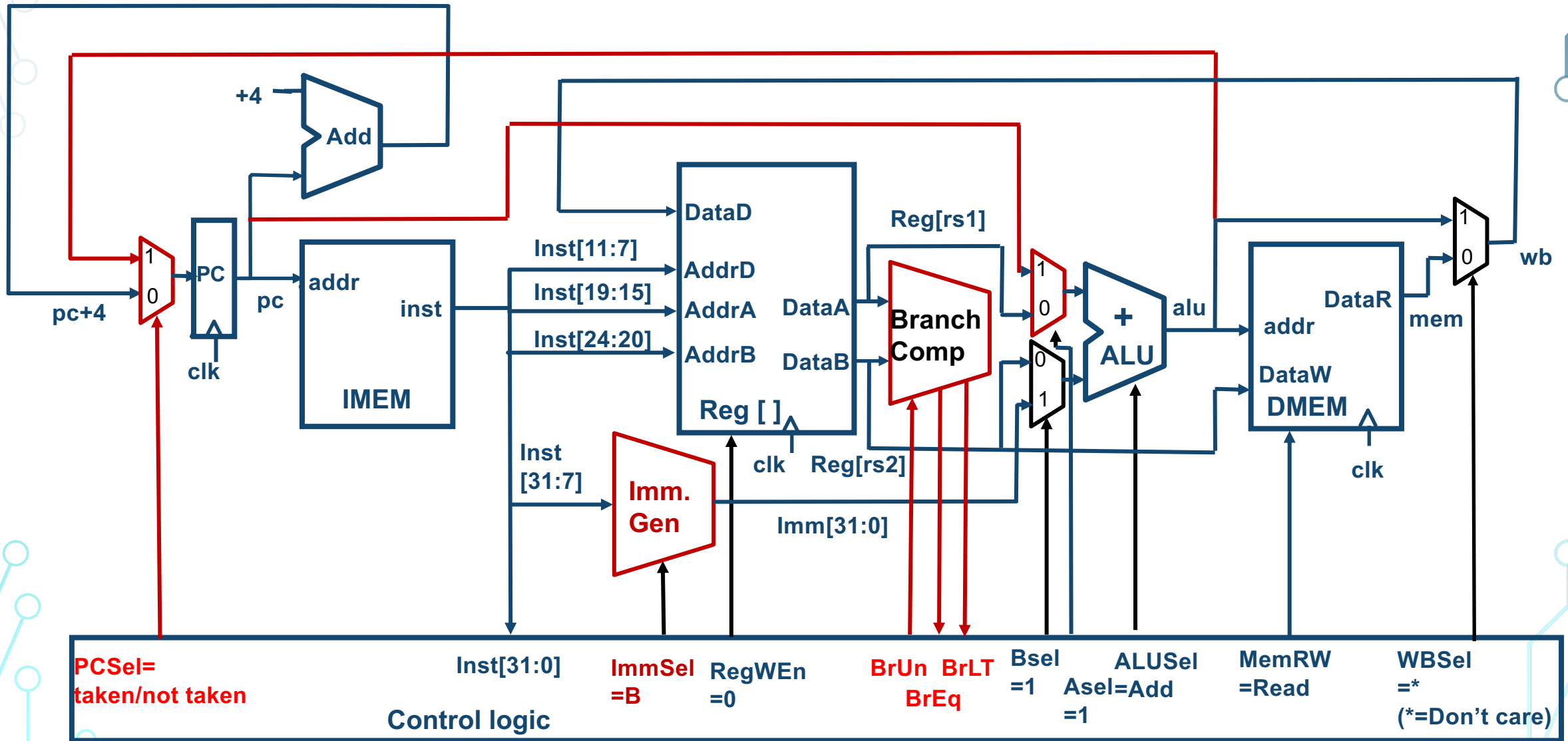
# To Add Branches

- Different change to the state:
  - $PC = PC + 4,$                       branch not taken  
     $PC + \text{immediate},$  branch taken
- Six branch instructions: **BEQ**, **BNE**, **BLT**, **BGE**, **BLTU**, **BGEU**
- Need to compute  $PC + \text{immediate}$  and to compare values of **rs1** and **rs2**
  - Need another add/sub unit

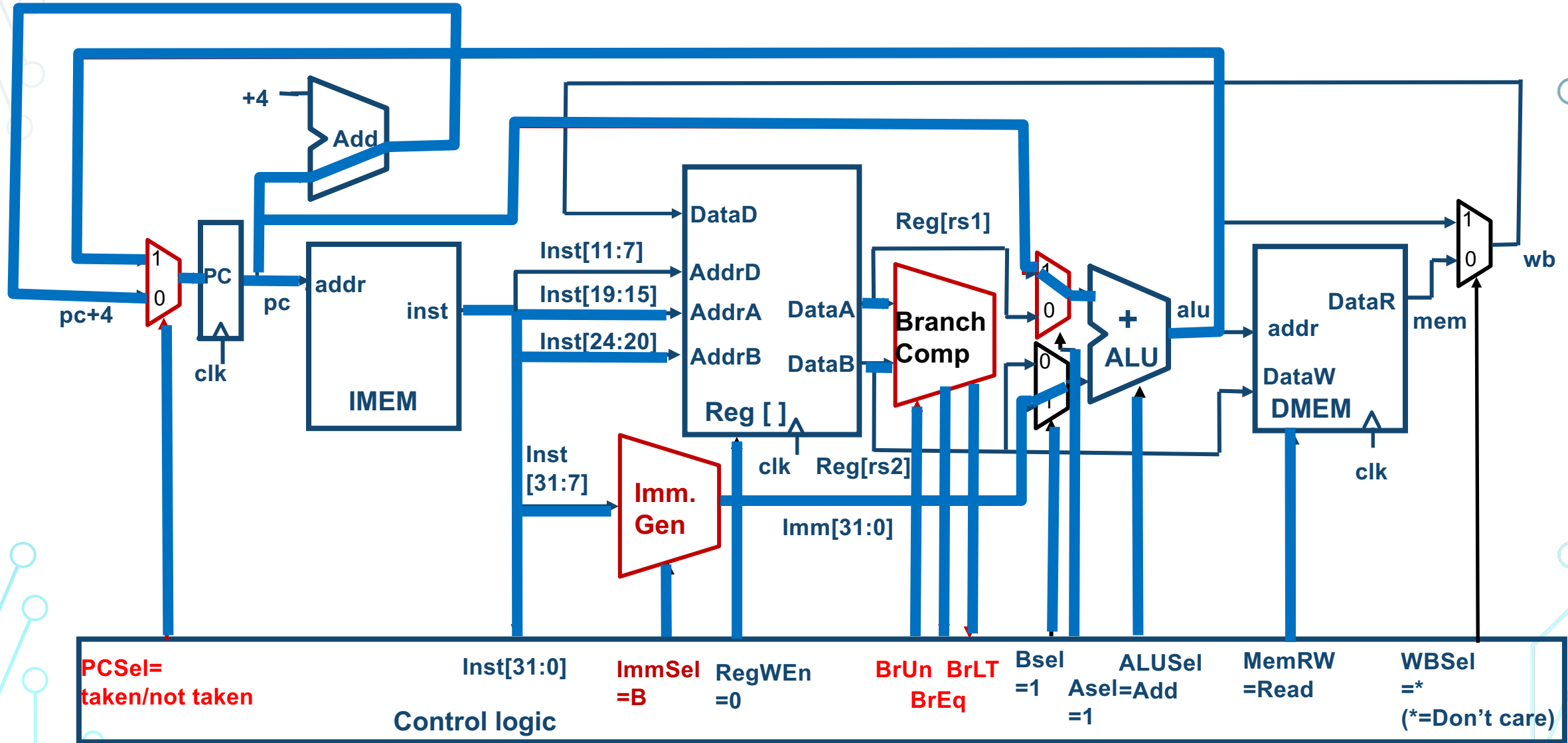
# Datapath So Far



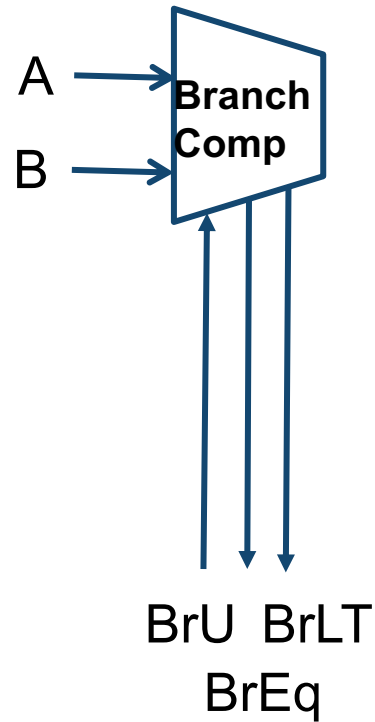
# Adding Branches



# Adding Branches



# Branch Comparator



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

# All RISC-V Branch Instructions

<code>imm[12 10:5]</code>	<code>rs2</code>	<code>rs1</code>	000	<code>imm[4:1 11]</code>	1100011
<code>imm[12 10:5]</code>	<code>rs2</code>	<code>rs1</code>	001	<code>imm[4:1 11]</code>	1100011
<code>imm[12 10:5]</code>	<code>rs2</code>	<code>rs1</code>	100	<code>imm[4:1 11]</code>	1100011
<code>imm[12 10:5]</code>	<code>rs2</code>	<code>rs1</code>	101	<code>imm[4:1 11]</code>	1100011
<code>imm[12 10:5]</code>	<code>rs2</code>	<code>rs1</code>	110	<code>imm[4:1 11]</code>	1100011
<code>imm[12 10:5]</code>	<code>rs2</code>	<code>rs1</code>	111	<code>imm[4:1 11]</code>	1100011

BEQ

BNE

BLT

BGE

BLTU

BGEU

# Summary

- We have covered the implementation of the base ISA for RV32I!!!
  - Get yourself familiar with the ISA Spec.
- Instruction type:
  - R-type
  - I-type
  - S-type
  - B-type
  - J-type
  - U-type
- Implementation suggested is straightforward, yet there are modalities in how to implement it well at gate level.
- Single-cycle datapath is slow – need to pipeline it