



Lecture 5: RISC-V Datapath

CS10014 Computer Organization

Department of Computer Science

Tsung Tai Yeh

Thursday: 1:20 pm– 3:10 pm

Classroom: EC-022



Acknowledgements and Disclaimer

- Slides were developed in the reference with
 - CS 61C at UC Berkeley
 - <https://inst.eecs.berkeley.edu/~cs61c/sp23/>
 - CS 252 at UC Berkeley
 - <https://people.eecs.berkeley.edu/~culler/courses/cs252-s05/>
 - CSCE 513 at University of South Carolina
 - <https://passlab.github.io/CSCE513/>

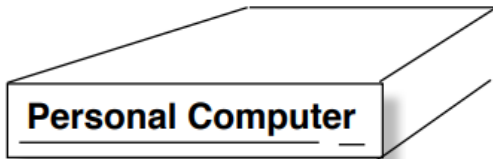


Outline

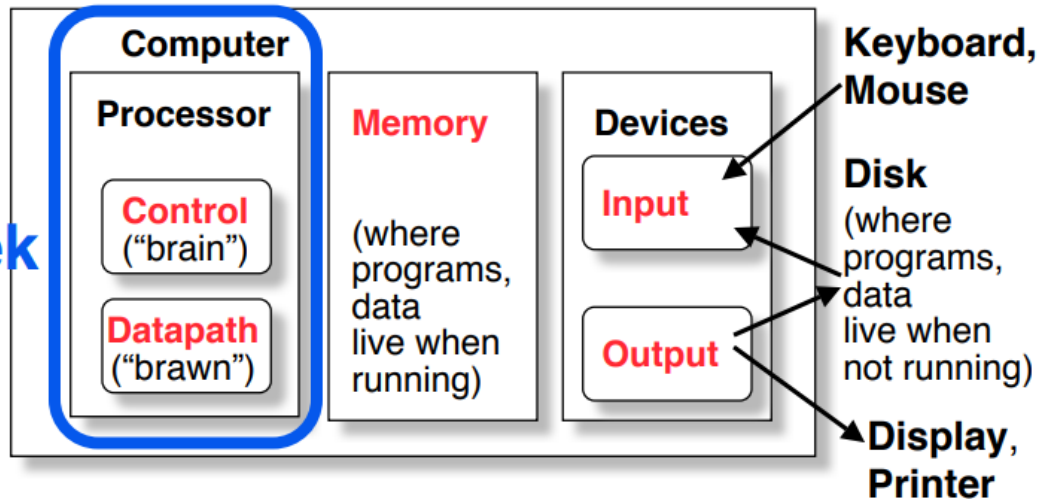
- State Element
- Design the Datapath
- R-Type Datapath
- I-Type Datapath



5 Components in a Computer



This week
and next





Single Core Processor

- **Processor (CPU)**

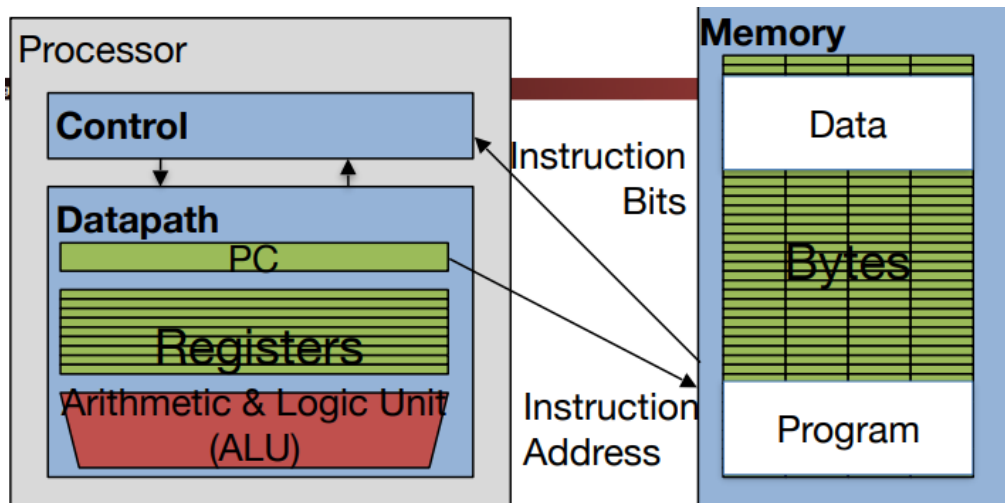
- The active part of the computer that does all the work (data manipulation and decision-making)

- **Datapath**

- Contains hardware necessary to perform operations required by the processor

- **Control**

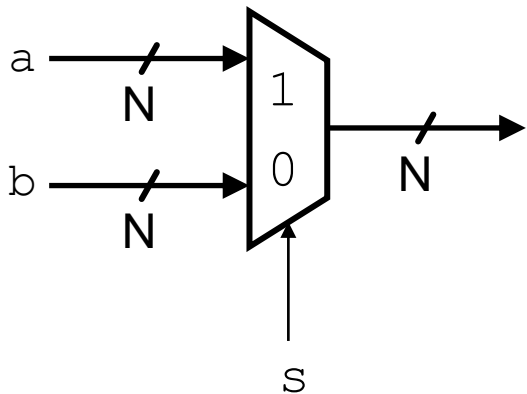
- Tells the datapath what needs to be done



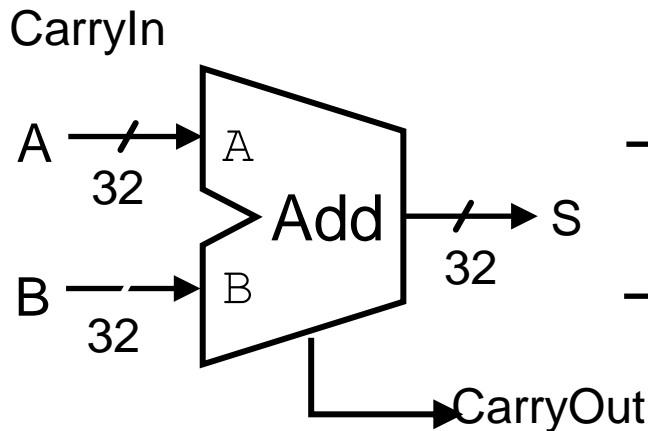


Combinational Logic Blocks

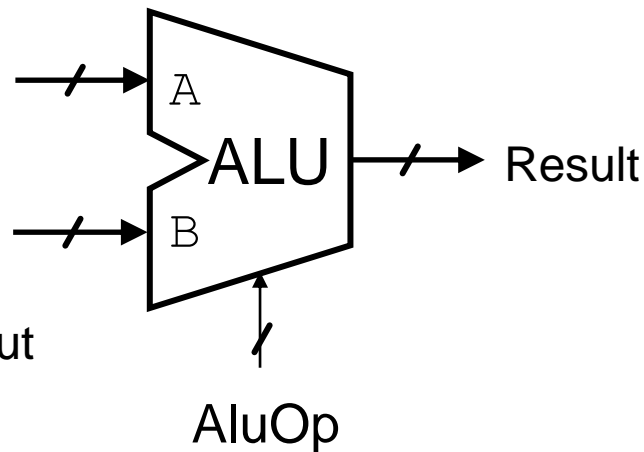
- Will think of all combinational logic subcircuits as block diagrams



32-bit-wide 2-to-1
MUX



32-Bit Adder

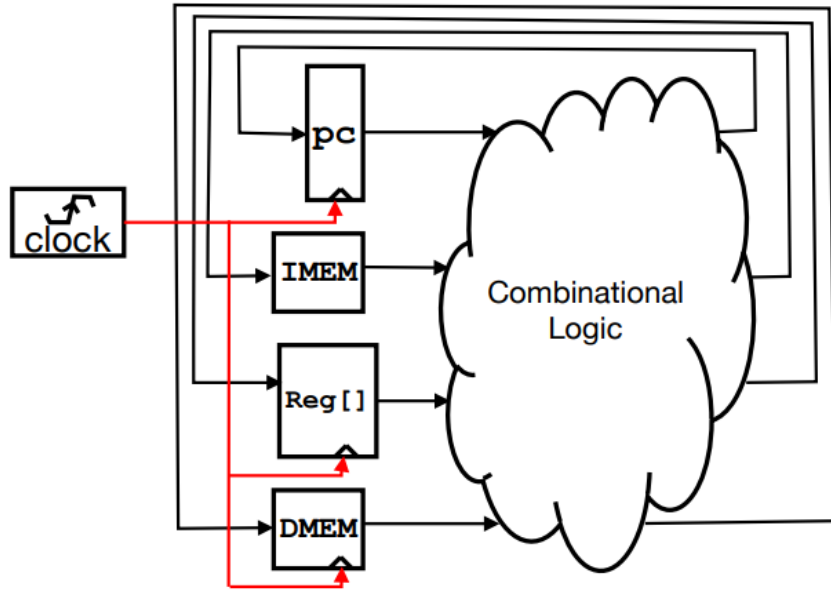


ALU (ALUOp selects from
multiple operations)



RISC-V Single Cycle Processor

- The CPU comprises two types of circuit
 - One-instruction-per-cycle
 - One every tick of the clock, the computer executes one instruction
 - At the rising clock edge
 - All the state elements are updated with the combinational logic outputs
 - Execution moves to the next clock cycle





Outline

- **State Element**
- Design the Datapath
- R-Type Datapath
- I-Type Datapath

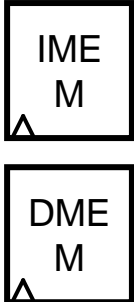


State Elements (1/6)

- State elements required by RV32I ISA
 - During CPU execution, each RV32I instruction reads and/or updates these state elements

Program Counter 

Register File Reg 

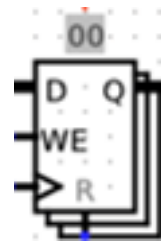
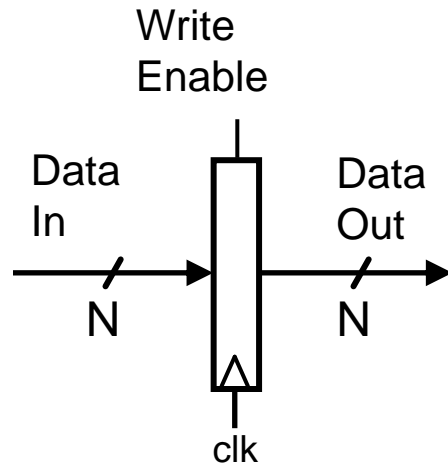
Memory MEM 



State Elements (2/6)

- **Program Counter**

- The program counter is a 32-bit register
- **Input**
 - N-bit data input bus
 - Write Enable: “Control” bit (1: asserted/high 0: de-asserted/0)
- **Output**
 - N-bit data output bus
- **Behavior**
 - If write enable is 1 on the rising clock edge, set Data Out = Data in
 - At all other times, Data out will not change, it will output its current value



A register in Logisim



State Elements (3/6)

- **Register File**

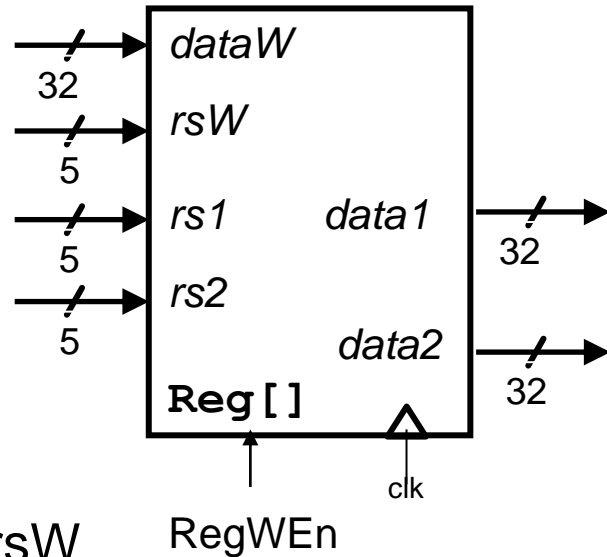
- The Register File (RegFile) has 32 registers

- **Input**

- One 32-bit input data bus, dataW
- Three 5-bit select busses, rs1, rs2, and rsW

- **Output**

- Two 32-bit output data busses, data1 and data2

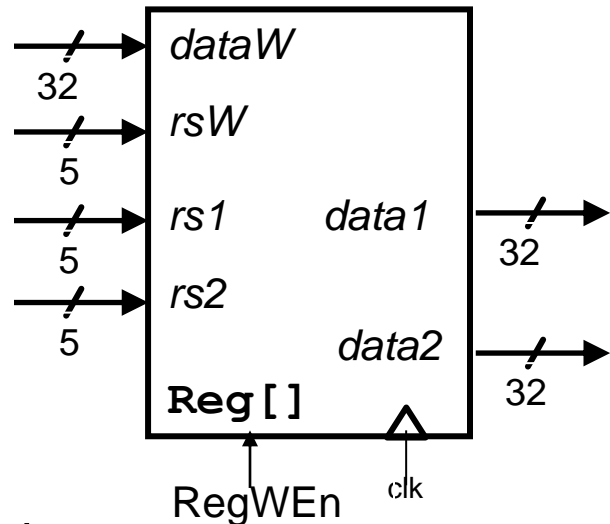




State Elements (4/6)

● Register File

- Registers are accessed via their 5-bit register numbers
 - R[rs1]: rs1 selects register to put on data1 bus out
 - R[rs2]: rs2 selects register to put on data2 bus out
 - R[rd]: rsW selects register to be written via dataW when RegWEn = 1
- Clock behavior: Write operation occurs on **rising clock edge**
 - Clock input only a factor on write!
 - All read operations behave like a combinational block
 - If rs1, rs2 valid, then data1, data2 valid after access time

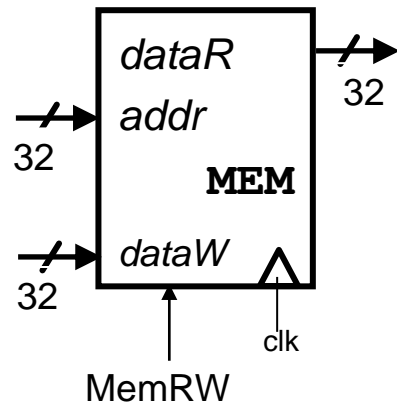




State Elements (5/6)

● Memory

- 32-bit byte-addressed memory space
- Memory access with 32-bit words
- Memory words are accessed as follows
 - **Read:** Address *addr* selects word to put on *dataR* bus
 - **Write:** Set $MemRW = 1$
Address *addr* selects word to be written with *dataW* bus
- Like RegFile, clock input is only a factor on write
 - If $MemRW = 1$, write occurs on rising clock edge
 - If $MemRW = 0$ and *addr* valid, then *dataR* valid after access time

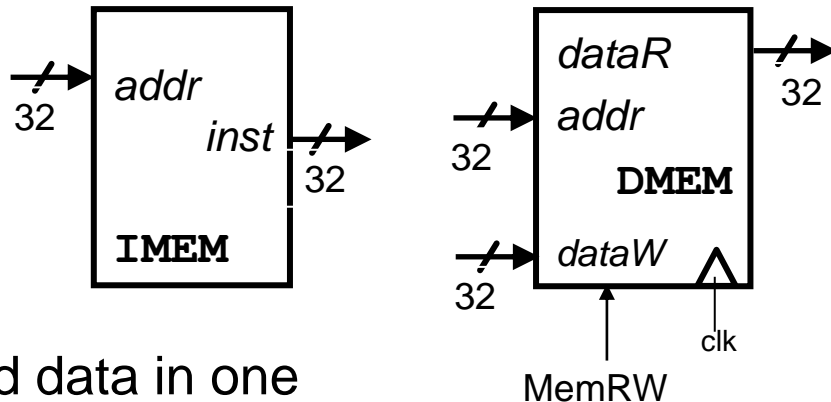




State Elements (6/6)

- **Two Memories (IMEM, DMEM)**

- Memory holds both instructions and data in one contiguous 32-bit memory space
- The processor will use two “separate” memories
 - **IMEM**: A **read-only** memory for fetching instruction
 - **DMEM**: A memory for loading (read) and storing (write) data words
- Because IMEM is read-only, it always behaves like a combinational block
 - If *addr* valid, then *instr* valid after access time





Outline

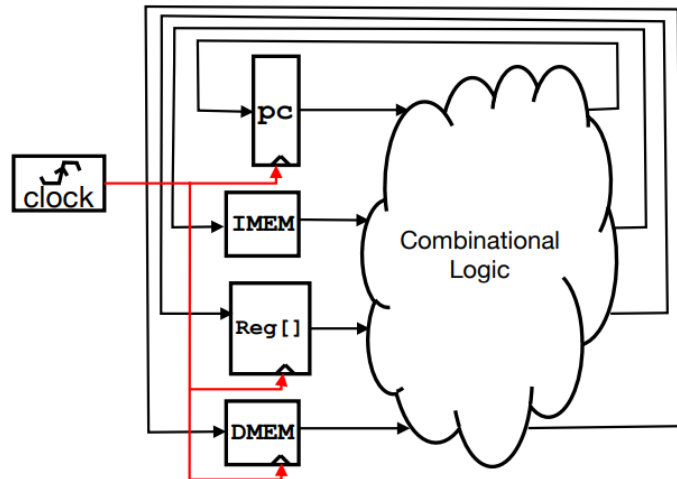
- State Element
- Design the Datapath
- R-Type Datapath
- I-Type Datapath



Design the Datapath (1/3)

- **Task: “Execute an instruction”**

- All necessary operations starting with fetching the instruction
- Problem:
 - A single “monolithic” block would be bulky and inefficient
- Solution:
 - Break up the process into **stages**, then connect the stages to create the whole datapath
 - Smaller stages are easier to design!
 - Modularity: Easy to optimize one stage without touching the others



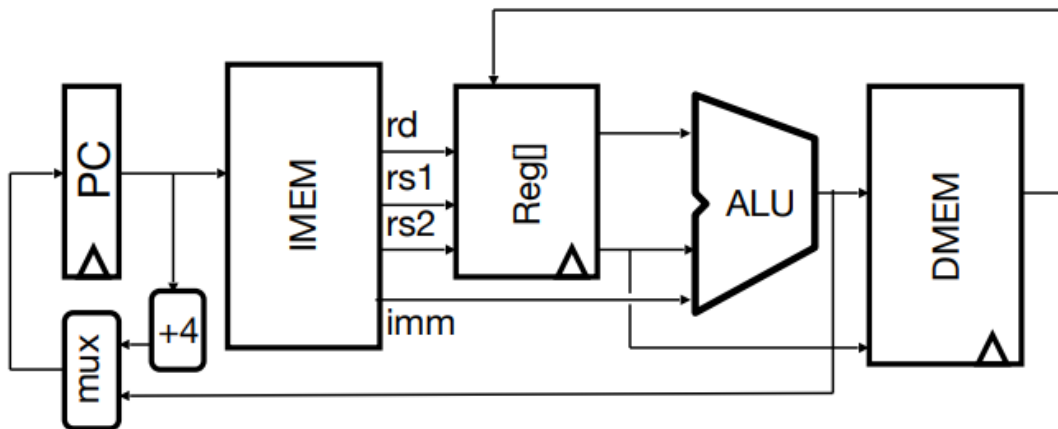


Design the Datapath (2/3)

- **The single-cycle processor**

- All stages of one RV32I instruction execute within the same clock cycle

5 basic stages of instruction execution

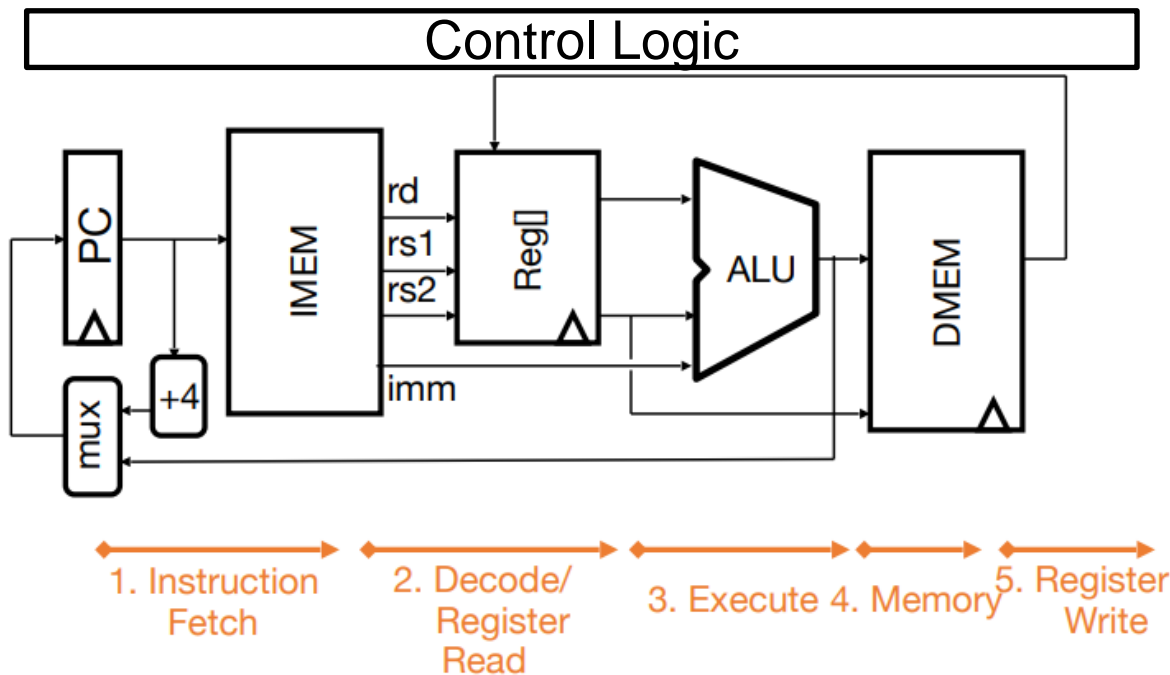




Design the Datapath (3/3)

- The **control logic** selects “needed” datapath lines based on the instruction

- MUX selector
- ALU op selector
- Write Enable ...



Not all instructions need all 5 stages



Outline

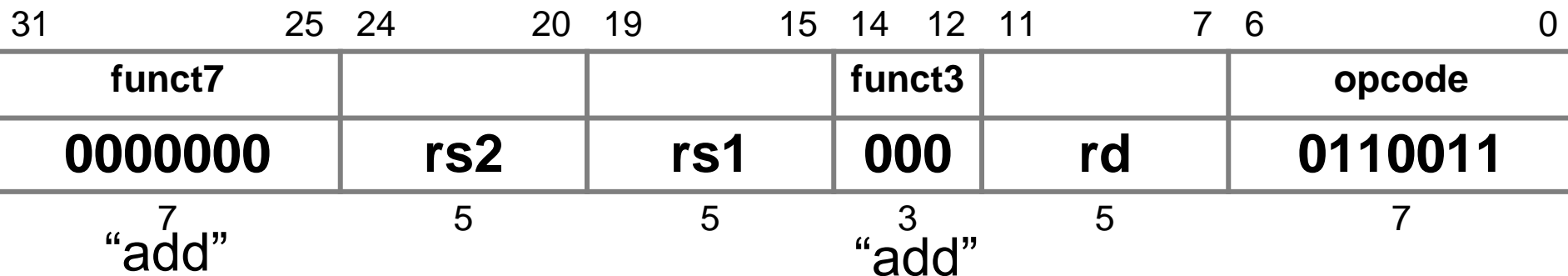
- State Element
- Design the Datapath
- **R-Type Datapath**
- I-Type Datapath



R-Type Datapath: add (1/3)

- Implementing the **add** instruction

`add rd, rs1, rs2`





R-Type Datapath: add (2/3)

- The add instruction makes **two changes** to processor state
 - RegFile $\text{Reg}[\text{rd}] = \text{Reg}[\text{rs1}] + \text{Reg}[\text{rs2}]$
 - PC $\text{PC} = \text{PC} + 4$

add rd, rs1, rs2



R-Type Datapath: add (3/3)

$$PC = PC + 4$$

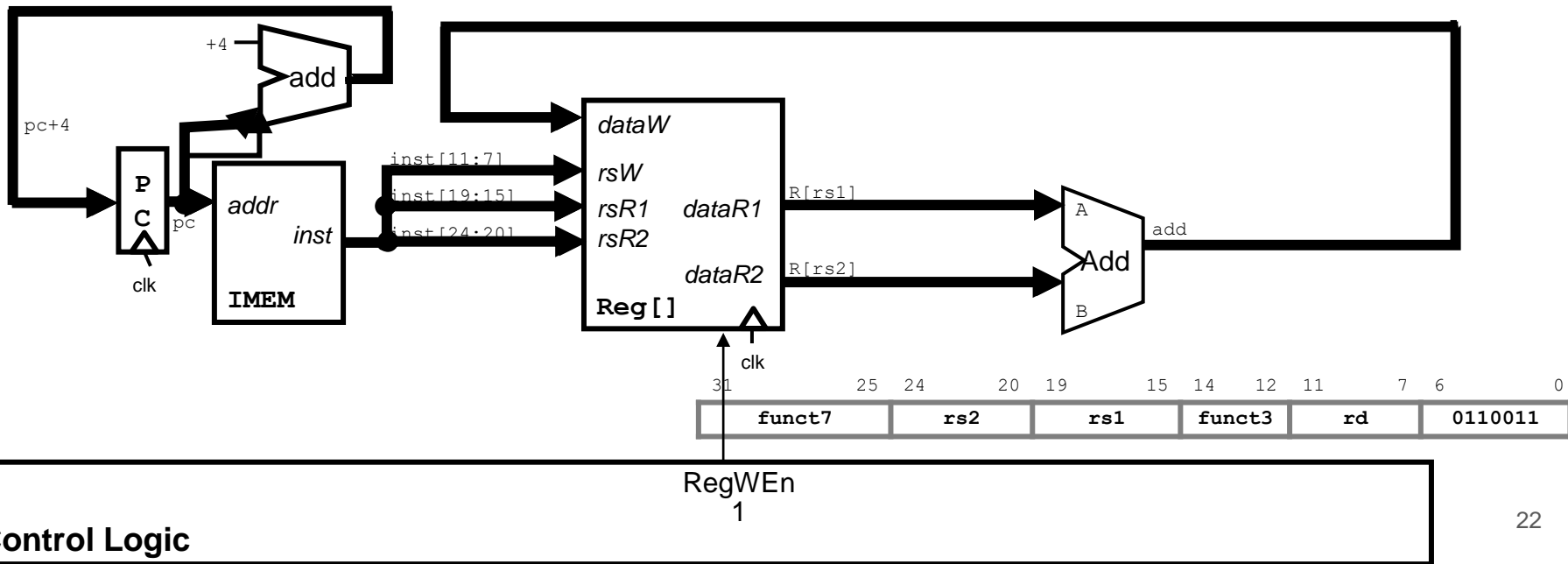
$$R[rd] = R[rs1] + R[rs2]$$

Increment PC to next instruction.

Split instruction to index into RegFile.

Feed read register values into Add.

Write Add output to destination register.





R-Type Datapath: sub (1/3)

- Implementing the **sub** instruction

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

funct7

funct3

opcode

funct7	rs2	rs1	funct3	rd	opcode	
0000000	rs2	rs1	000	rd	0110011	add
0 1 00000	rs2	rs1	000	rd	0110011	sub

- sub **subtracts** operands instead of adding them
 - RegFile** $\text{Reg}[\text{rd}] = \text{Reg}[\text{rs1}] - \text{Reg}[\text{rs2}]$
 - PC** $\text{PC} = \text{PC} + 4$
 - Instruction bit `inst[30]` selects between add/sub
 - Details left to control logic



R-Type Datapath: sub (2/3)

$$PC = PC + 4$$

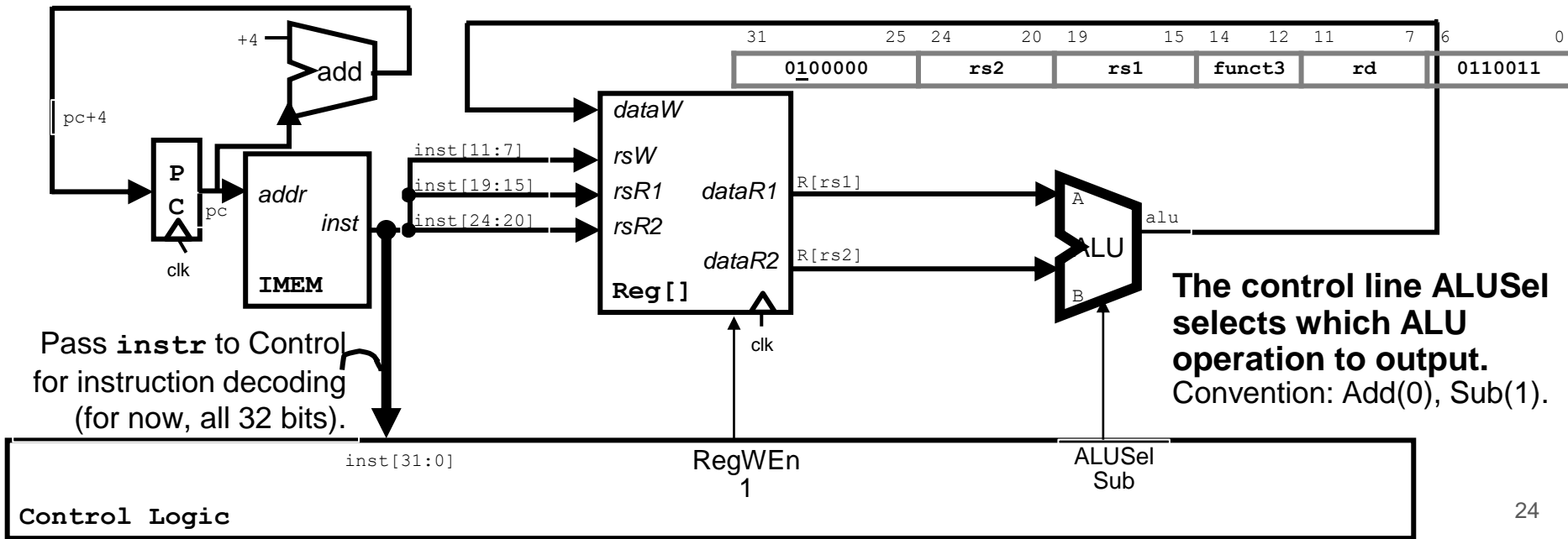
$$R[rd] = R[rs1] - R[rs2]$$

Increment PC to next instruction.

Split instruction to index into RegFile.

Feed read register values into ALU.

Write ALU output to destination register.



The control line **ALUSel** selects which ALU operation to output. Convention: Add(0), Sub(1).



R-Type Datapath: sub (3/3)

$$PC = PC + 4$$

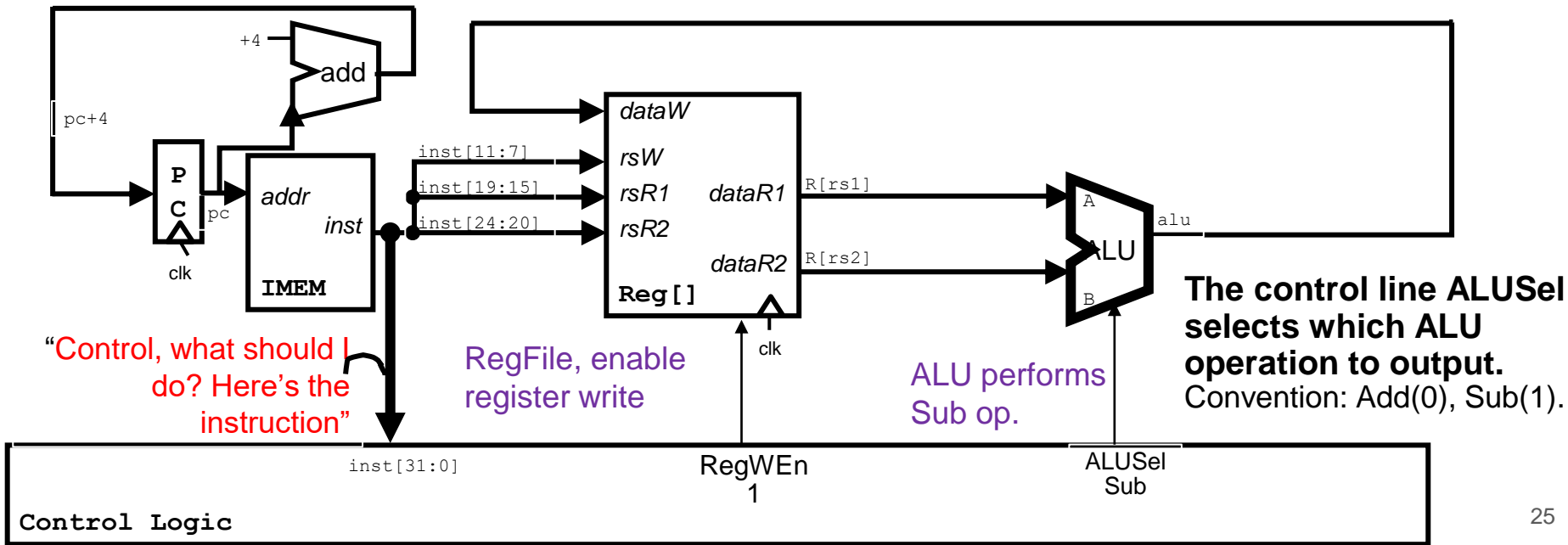
$$R[rd] = R[rs1] - R[rs2]$$

Increment PC to next instruction.

Split instruction to index into RegFile.

Feed read register values into ALU.

Write ALU output to destination register.





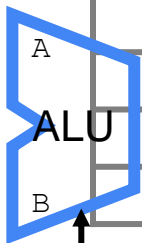
R-Type Datapath: all R-Type Instruction

Funct7

funct3

opcode

Funct7	rs2	rs1	funct3	rd	opcode	Instruction
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



ALUSel

add,sub,xor,and,or,
slt,sltu,sll,sra,srl

The Control Logic decodes funct3, funct7 instruction fields and selects appropriate ALU function by setting the control line ALUSel



Outline

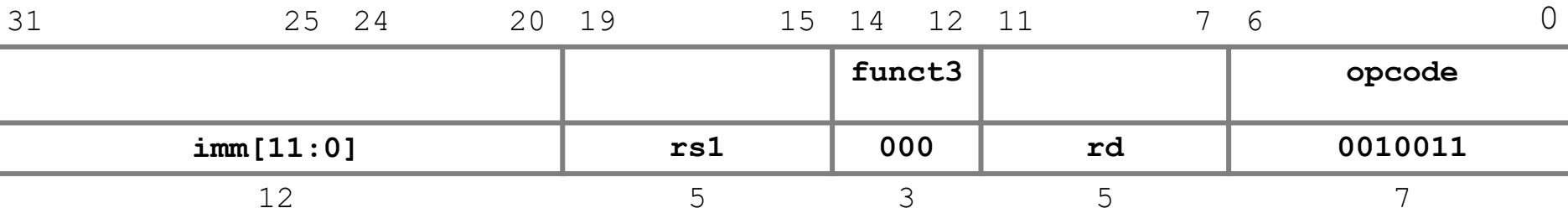
- State Element
- Design the Datapath
- R-Type Datapath
- **I-Type Datapath**



I-Type Datapath: addi

- RV32I I-format addi

`addi rd, rs1, imm`



- The addi instruction needs to build an **immediate imm!**
 - RegFile** $\text{Reg}[\text{rd}] = \text{Reg}[\text{rs1}] + \text{imm}$
 - PC** $\text{PC} = \text{PC} + 4$



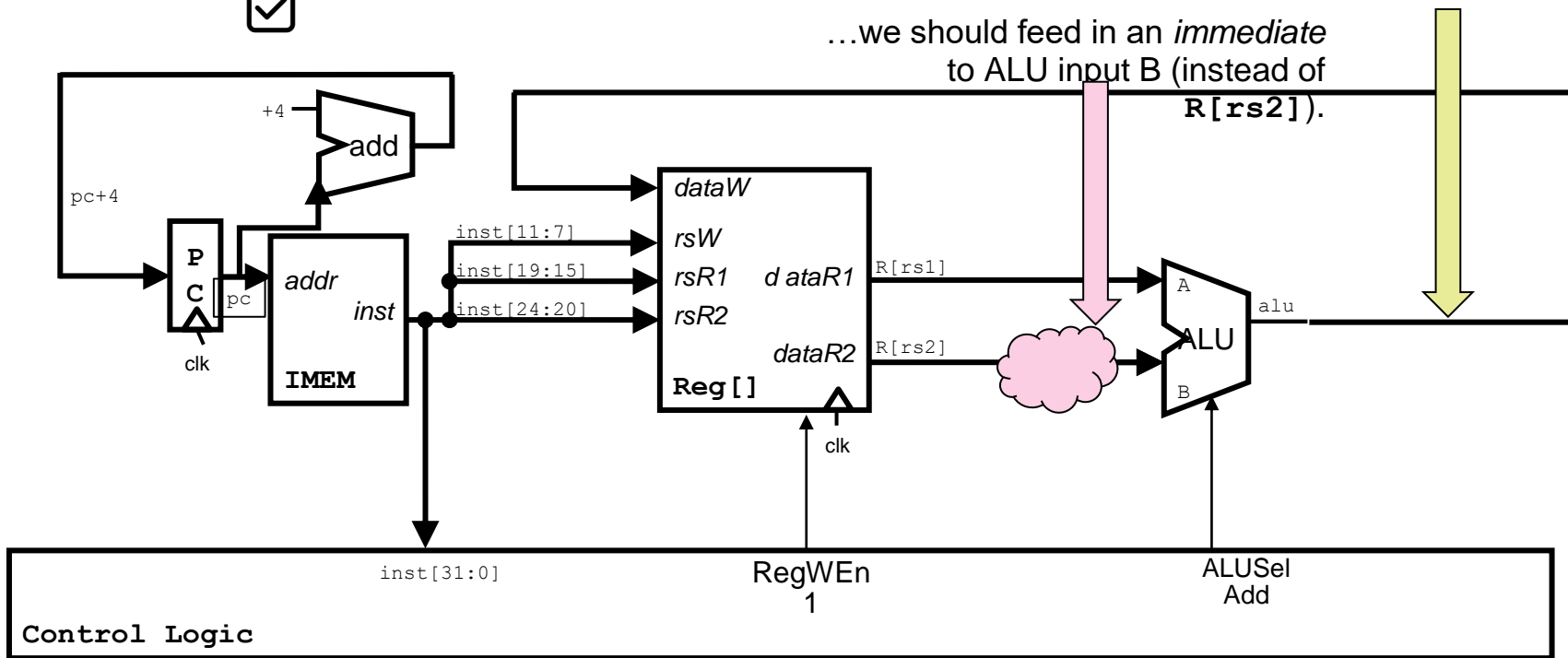
I-Type Datapath: addi (1/5)

$$PC = PC + 4$$

$$R[rd] = R[rs1] + imm$$

To compute $alu = R[rs1] + imm...$

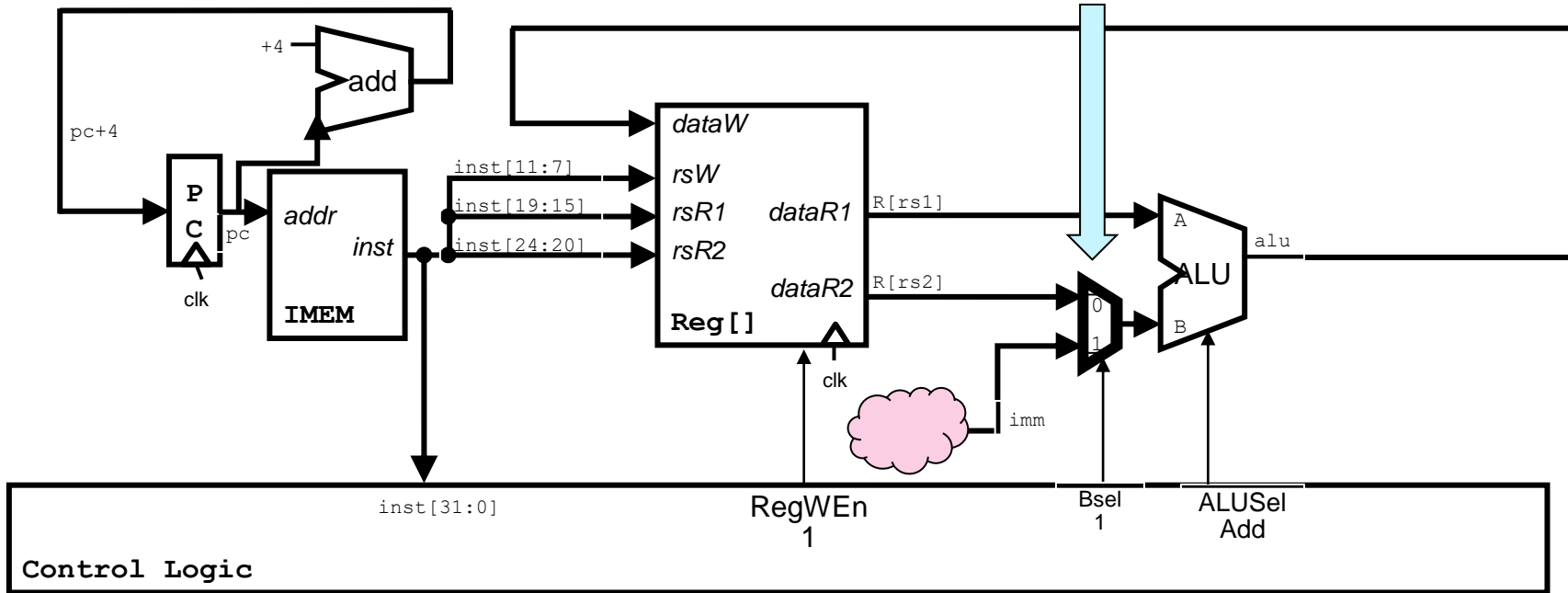
...we should feed in an *immediate* to ALU input B (instead of $R[rs2]$).





I-Type Datapath: addi (2/5)

1. Control line
Bsel=1 selects the
generated immediate
imm for ALU input B.

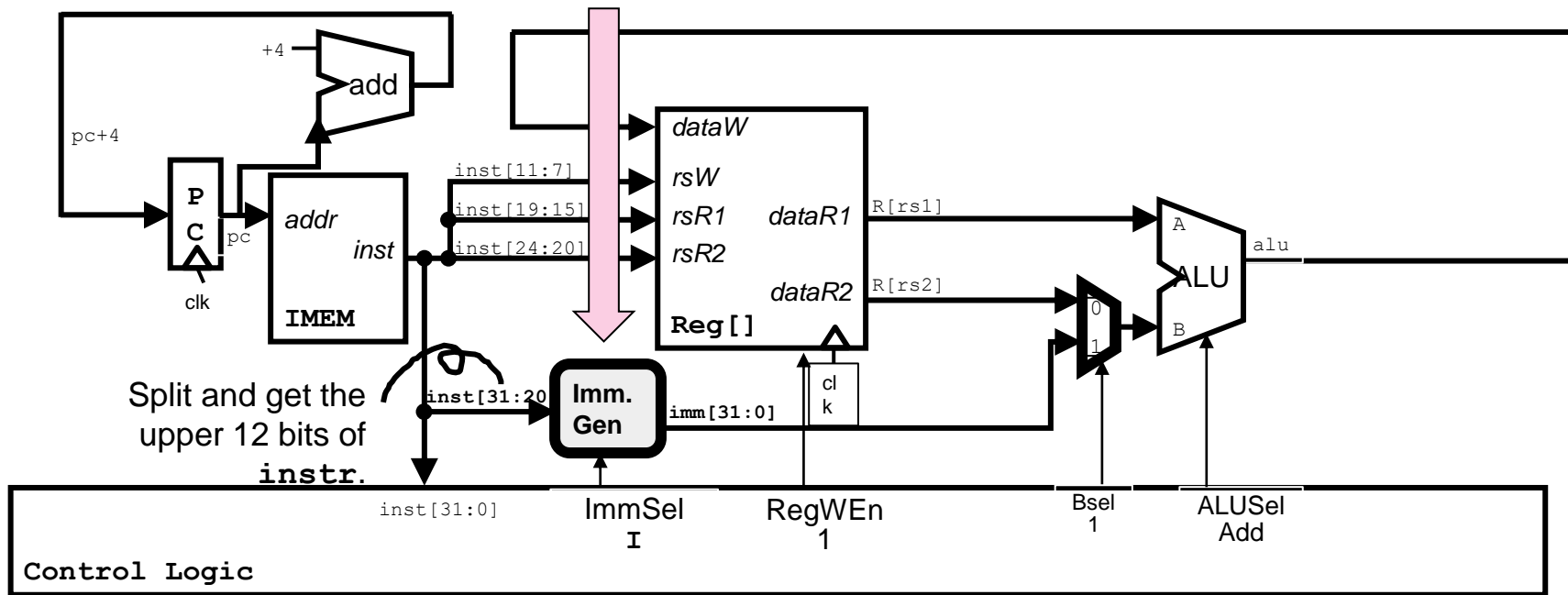




I-Type Datapath: addi (3/5)

2. *Immediate Generation Block* builds a 32-bit immediate *imm* from instruction bits.

1. Control line *Bsel=1* selects the generated immediate *imm* for ALU input B.





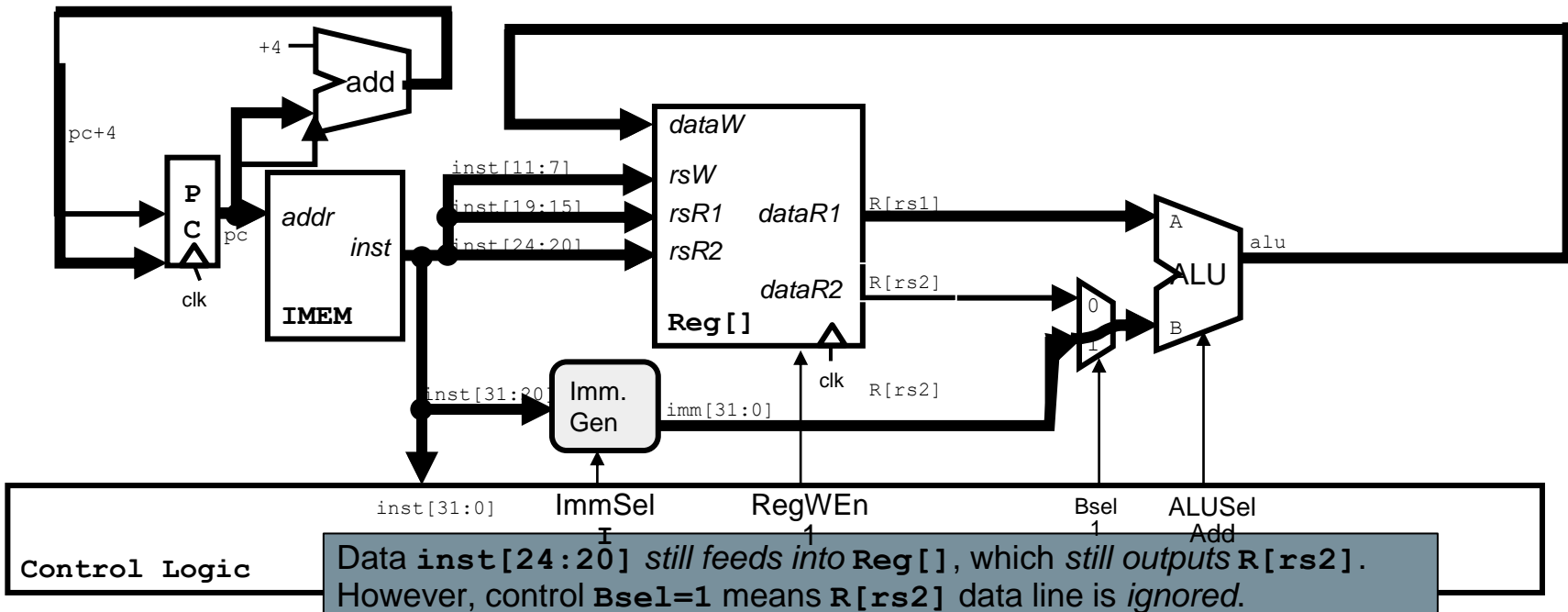
I-Type Datapath: addi (4/5)

Increment PC to next instruction.

Immediate Generation Block builds a 32-bit immediate *imm* from instruction bits.

Control line *Bsel=1* selects the generated immediate *imm* for ALU input B.

Write ALU output to destination register.

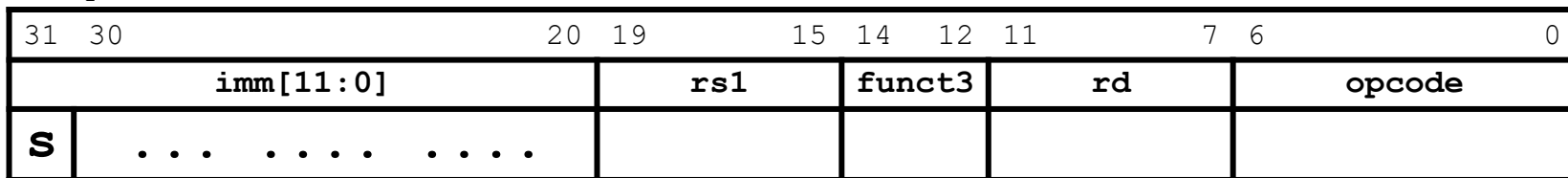
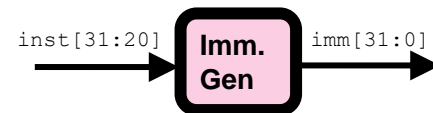




Immediate Generation Block

Instruction

$inst[31:0]$

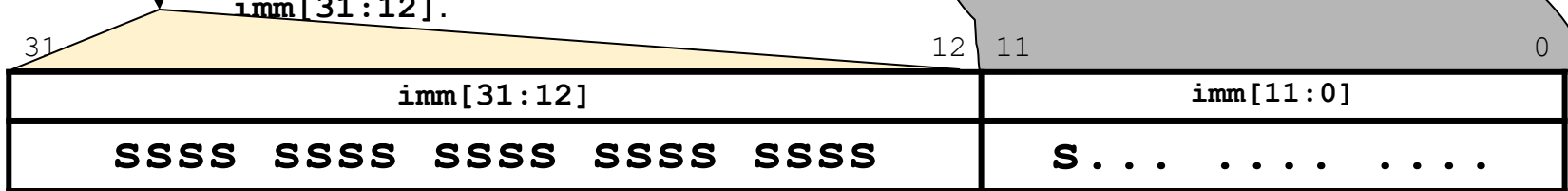


Copy upper 12 bits of instruction, $inst[31:20]$, to lower 12 bits of immediate, $imm[11:0]$.

Sign-extend: Copy $inst[31]$ to upper 20 bits of immediate, $imm[31:12]$.

Immediate

$imm[31:0]$





Summary: Arithmetic/Logical Datapath

- All data lines carry information.
- Control logic determines what is “useful/needed” vs. what is “ignored.”
 - e.g., ALUSel: chooses ALU operation; Bsel: chooses register/immediate for ALU input B.

