RISC-V (cont.) & CPU Stages

CS 3410: Computer System Organization and Programming

Spring 2025

Prelim 1

- Tonight at **7:30pm** in **Statler Auditorium (STL 185)**
- 120 minutes (2 hours)
- Closed-book, closed-notes
  - Reference sheet is provided
- **Bring your student ID**

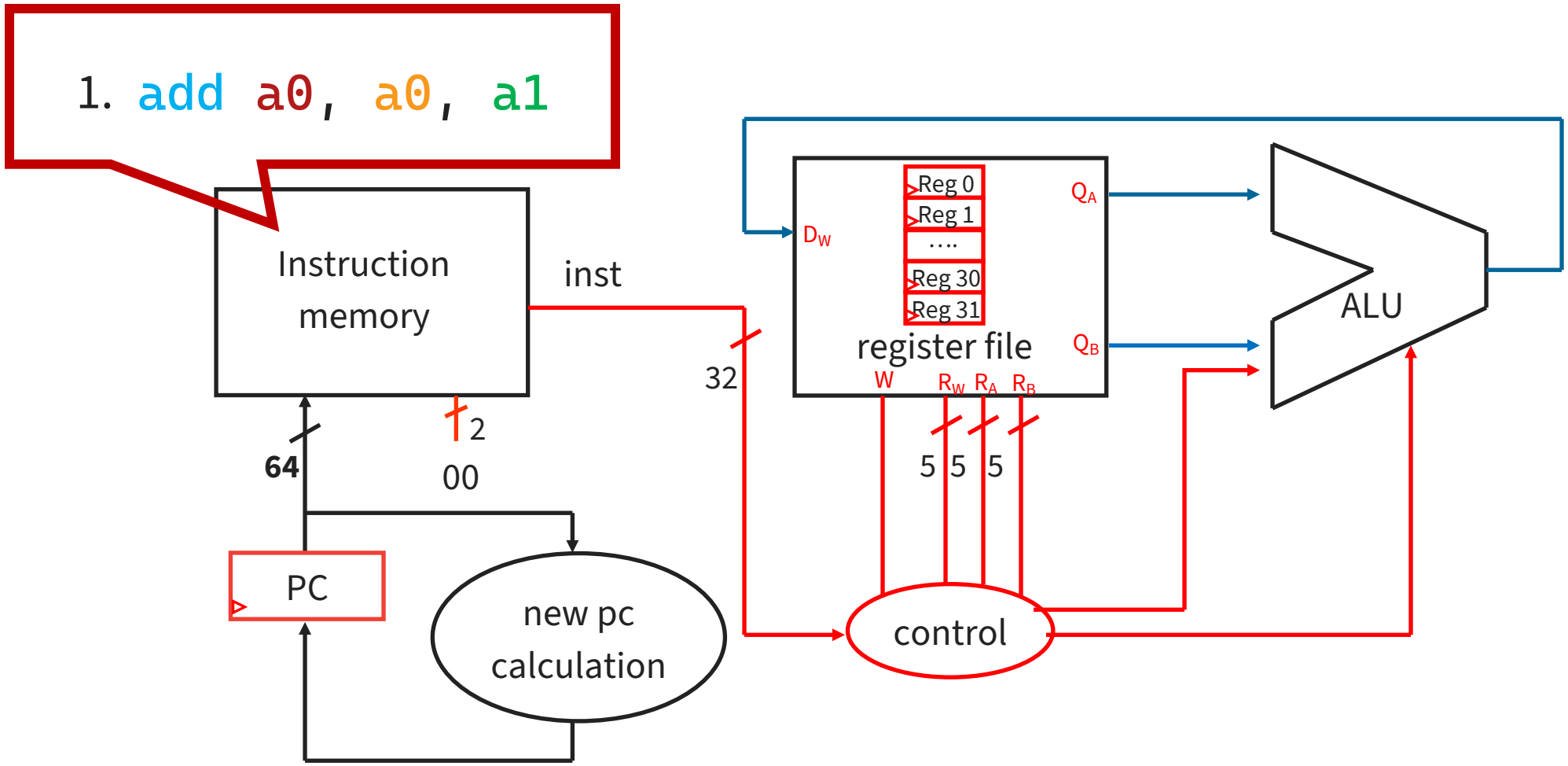
How do you feel about the prelim?

Today's Plan

- RISC-V Instructions
  - Loads & Stores
  - Branch If Equal
- Stages of a CPU
- End early!

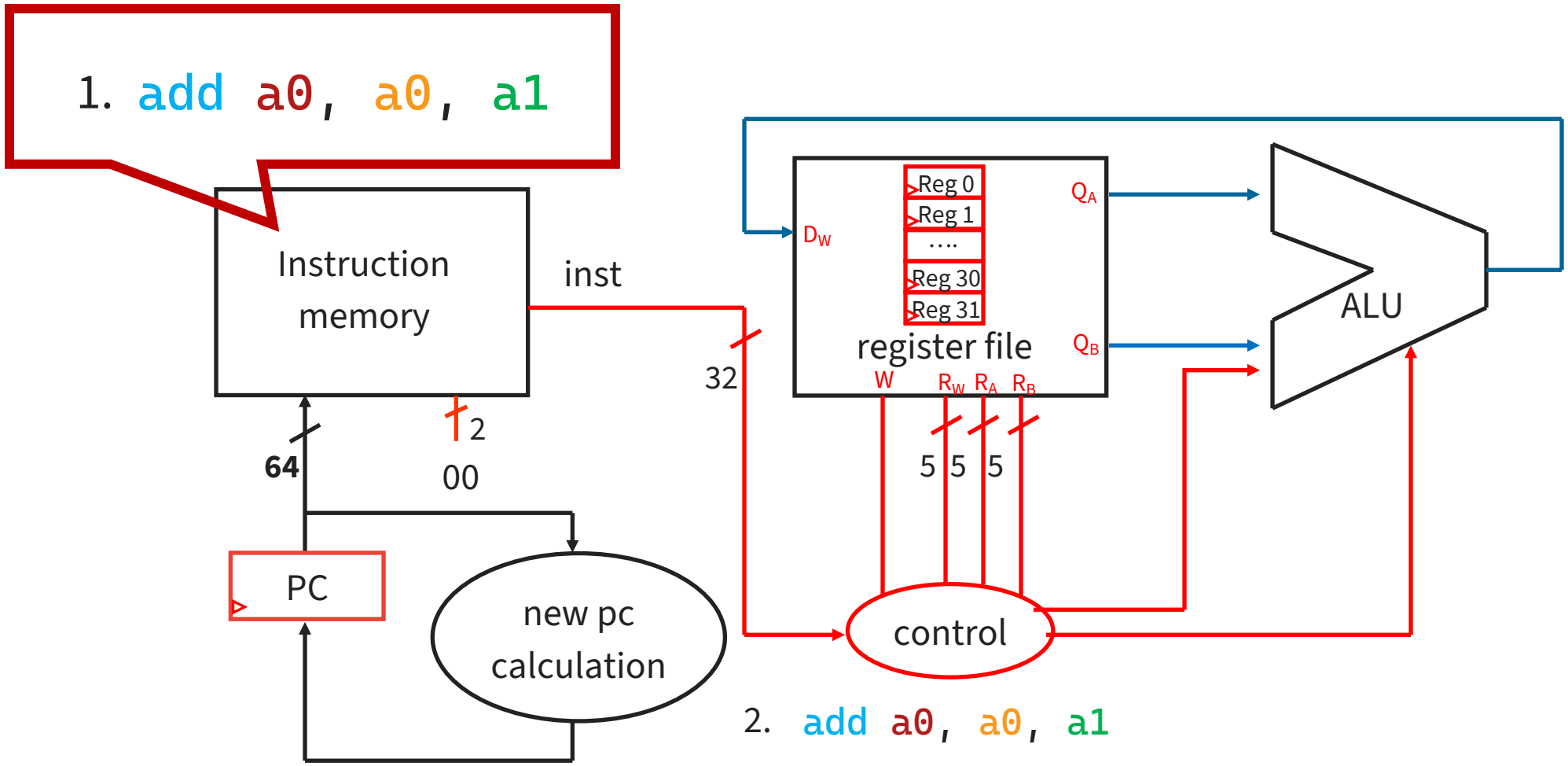
On Last Week's Episode...

```
1. add a0, a0, a1
```



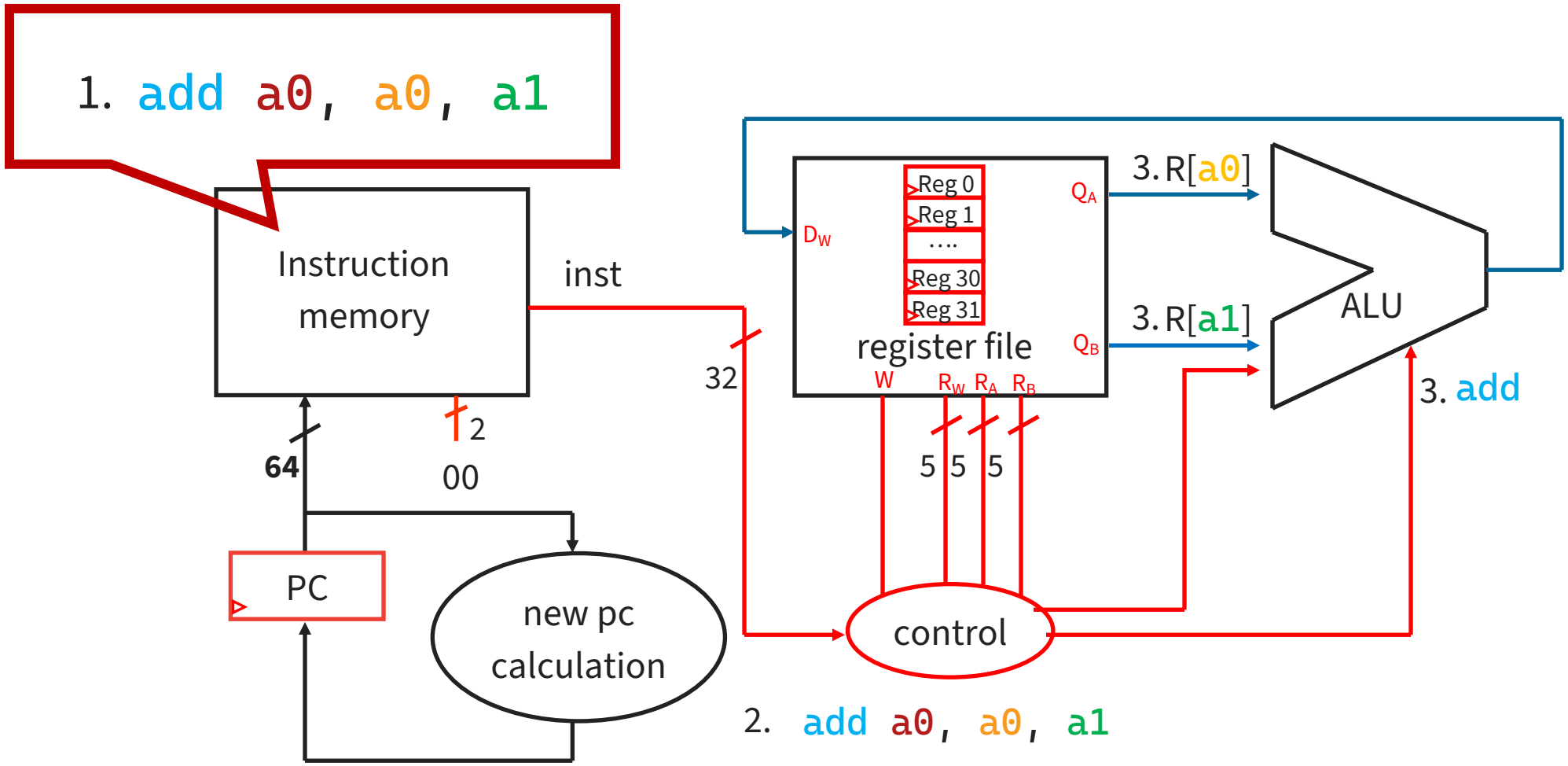
On Last Week's Episode...

1. `add a0, a0, a1`

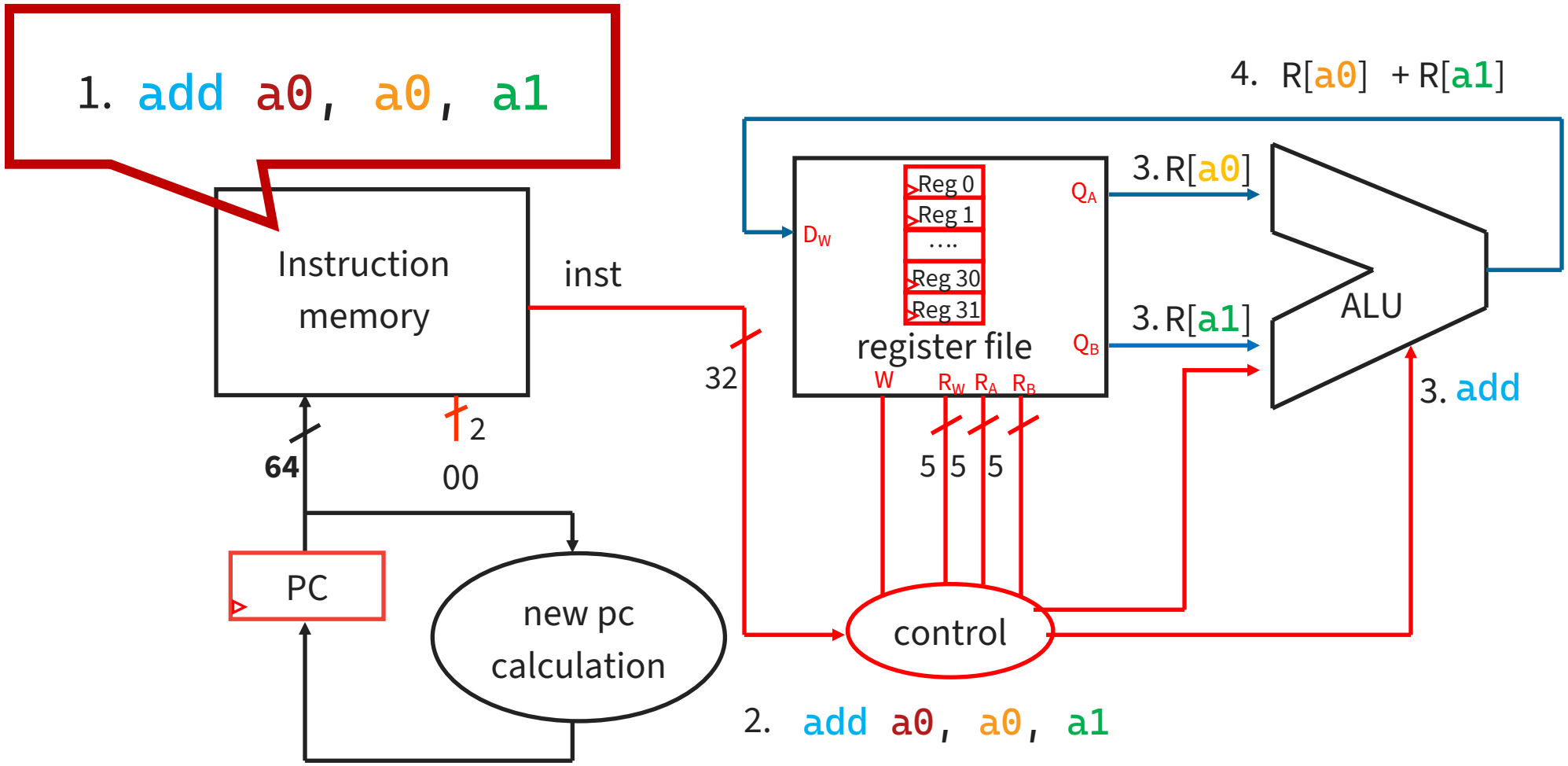


2. `add a0, a0, a1`

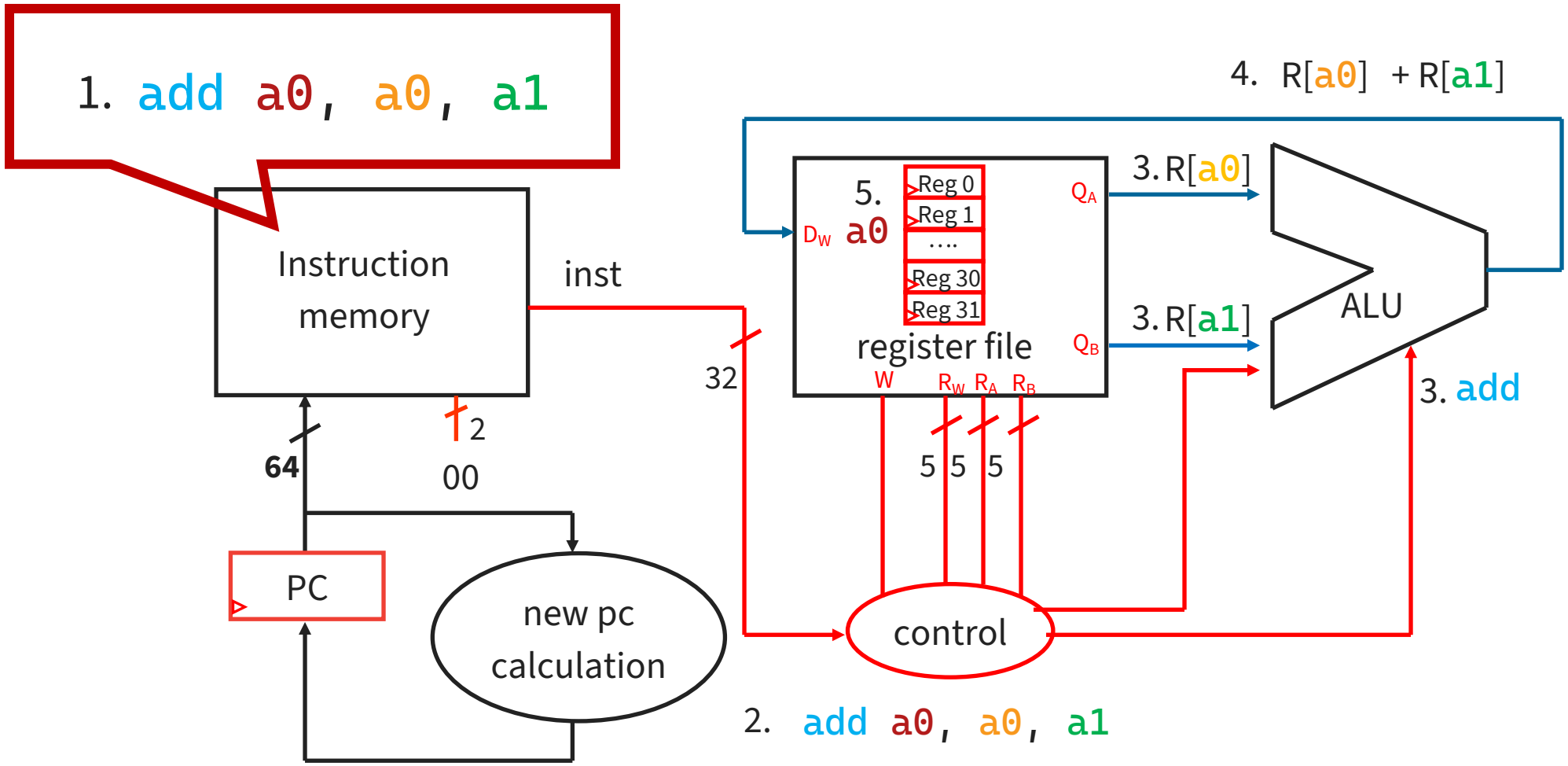
On Last Week's Episode...



On Last Week's Episode...



On Last Week's Episode...



Assembly Programming

Pseudocode

```
a0 = 34
a1 = a0 - 13
a2 = a1 * 2
```

Assembly

Assembly Programming

Pseudocode

```
a0 = 34
a1 = a0 - 13
a2 = a1 * 2
```

Assembly

How do we put 34 into register a0?

Assembly Programming

Always zero!

Pseudocode

```
a0 = 0 + 34
a1 = a0 - 13
a2 = a1 * 2
```

Assembly

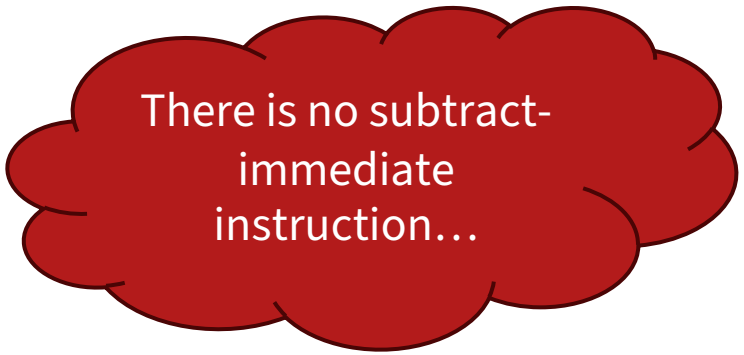
```
addi a0, x0, 34
```

How do we put 34 into register a0?

Assembly Programming

Pseudocode

```
a0 = 0 + 34
a1 = a0 - 13
a2 = a1 * 2
```



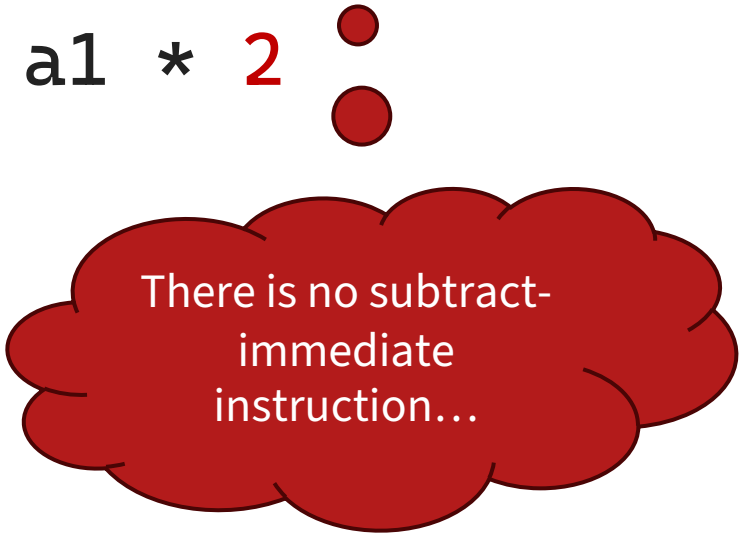
Assembly

```
addi a0, x0, 34
```

Assembly Programming

Pseudocode

```
a0 = 0 + 34
a1 = a0 - 13
a2 = a1 * 2
```



Assembly

```
addi a0, x0, 34
addi a1, a0, -13
```

Assembly Programming

Pseudocode

```
a0 = 0 + 34
a1 = a0 - 13
a2 = a1 * 2
```

Assembly

```
addi a0, x0, 34
addi a1, a0, -13
```

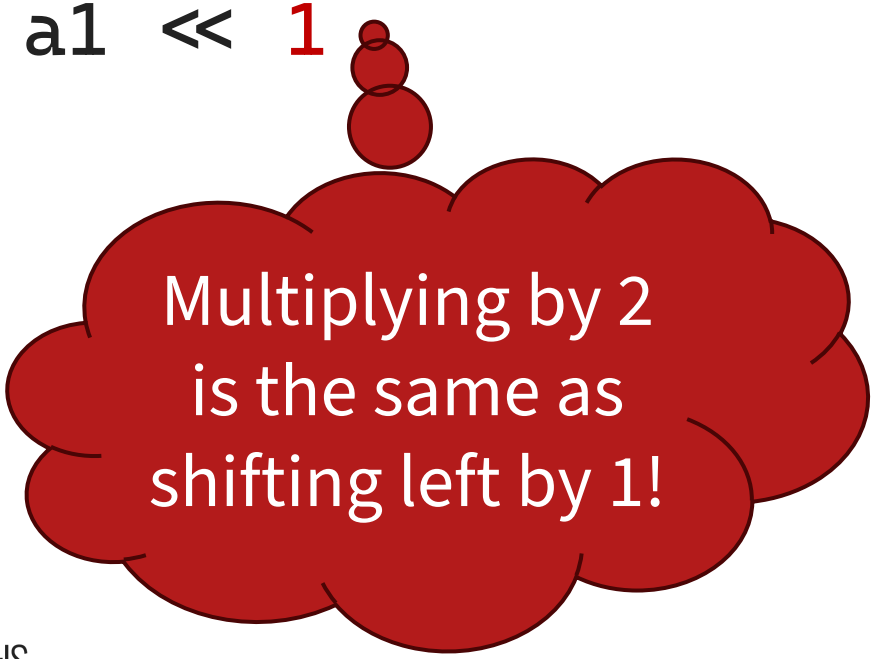
Assembly Programming

Pseudocode

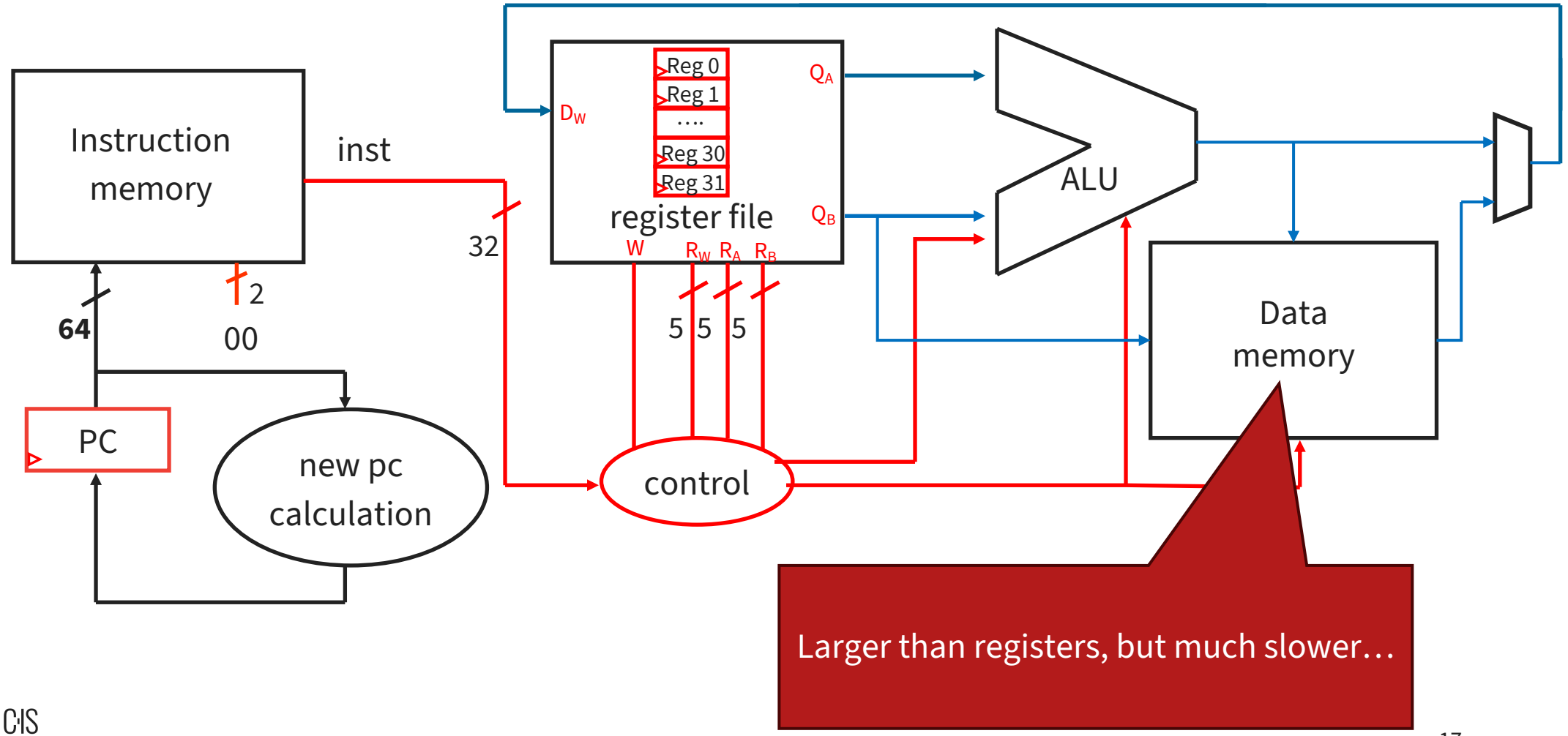
```
a0 = 0 + 34
a1 = a0 - 13
a2 = a1 << 1
```

Assembly

```
addi a0, x0, 34
addi a1, a0, -13
slli a2, a1, 1
```



Data Memory!


Instruction memory
inst
32
64
2
00
PC
new pc calculation
Reg 0
Reg 1
….
Reg 30
Reg 31
register file
D_W
Q_A
Q_B
W
R_W R_A R_B
5 5 5
ALU
Data memory
control
Larger than registers, but much slower…

RISC-V Instruction Formats

R-type

funct7	rs2	rs1	funct3	rd	op
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

`add,and,sll`

I-type

imm	rs1	funct3	rd	op
12 bits	5 bits	3 bits	5 bits	7 bits

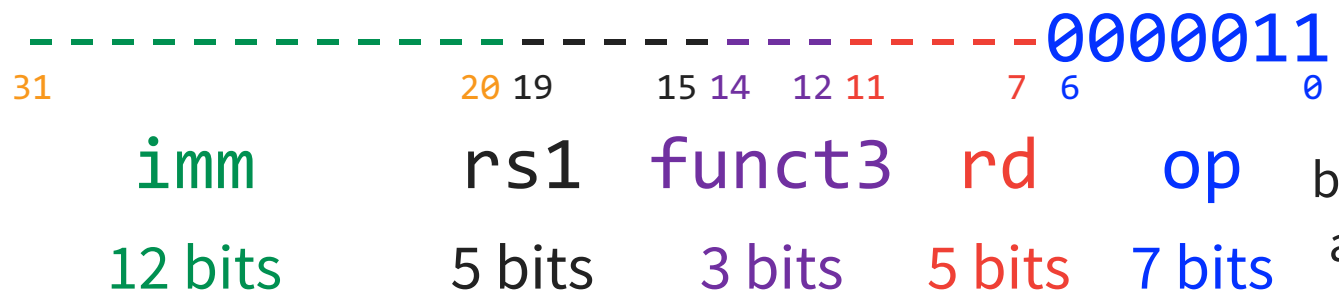
`addi,andi`

U-type

imm	rd	op
20 bits	5 bits	7 bits

`lui`

I-Type: Load Instructions

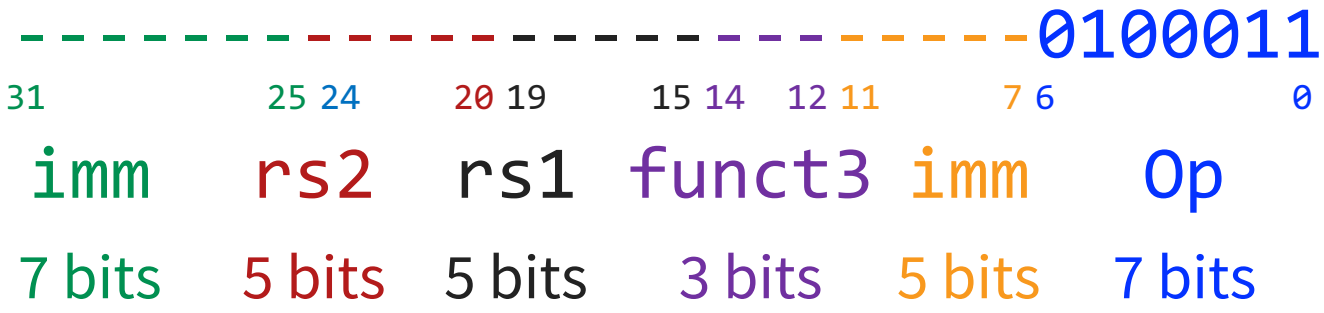


base + offset addressing

	funct3	mnemonic	Description
load byte	000	LB rd, imm(rs1)	R[rd] = sign_ext(Mem[imm+R[rs1]])
load word	010	LW rd, imm(rs1)	R[rd] = sign_ext(Mem[imm+R[rs1]])
load double word	011	LD rd, imm(rs1)	R[rd] = Mem[imm+R[rs1]])

imm field is signed

S-Type: Store Instructions



base + offset addressing

	funct3	mnemonic	description
store byte	000	SB rs2, imm(rs1)	Mem[imm+R[rs1]] = R[rs2]
store word	010	SW rs2, imm(rs1)	Mem[imm+R[rs1]] = R[rs2]
store double word	011	SD rs2, imm(rs1)	Mem[imm+R[rs1]] = R[rs2]

imm field is signed

Example: Loading From Memory

C

```c
// `x` → x12
int x = 5;
// `arr` → x15
int arr[100];
int y = x + arr[3];
```

RISC-V Assembly

```
lw x10, 12(x15)
```

Example: Loading From Memory

C

```c
// `x` → x12
int x = 5;
// `arr` → x15
int arr[100];
int y = x + arr[3];
```

RISC-V Assembly

```
lw x10, 12(x15)
add x11, x12, x10
```

Example: Storing To Memory

C

```c
// `x` → x12
int x = 5;
// `arr` → x15
int arr[100];
arr[10] = x + arr[3];
```

RISC-V Assembly

```
lw x10, 12(x15)
add x11, x12, x10
sw x11, 40(x15)
```

Instruction Types So Far

- **Arithmetic/Logical**
  - Manipulates / changes data
- **Memory Access**
  - Moves data between data memory and registers
- **Control Flow**
  - Affects the PC: what instruction to execute next

Branch If Equal: `beq`

If `rs1 = rs2` then go to the instruction at `label`

```
beq rs1, rs2, label
```

Example: Loading From Memory

C

```
if (x1 ≠ x2) {
   x3 += 42;
}
x3 += 27;
```

RISC-V Assembly

```
   beq x1, x2, EXIT


EXIT:
```

Example: Loading From Memory

C

```
if (x1 ≠ x2) {
   x3 += 42;
}
x3 += 27;
```

RISC-V Assembly

```
   beq x1, x2, EXIT
   addi x3, x3, 42
EXIT:
```

Example: Loading From Memory

C

```
if (x1 ≠ x2) {
   x3 += 42;
}
x3 += 27;
```

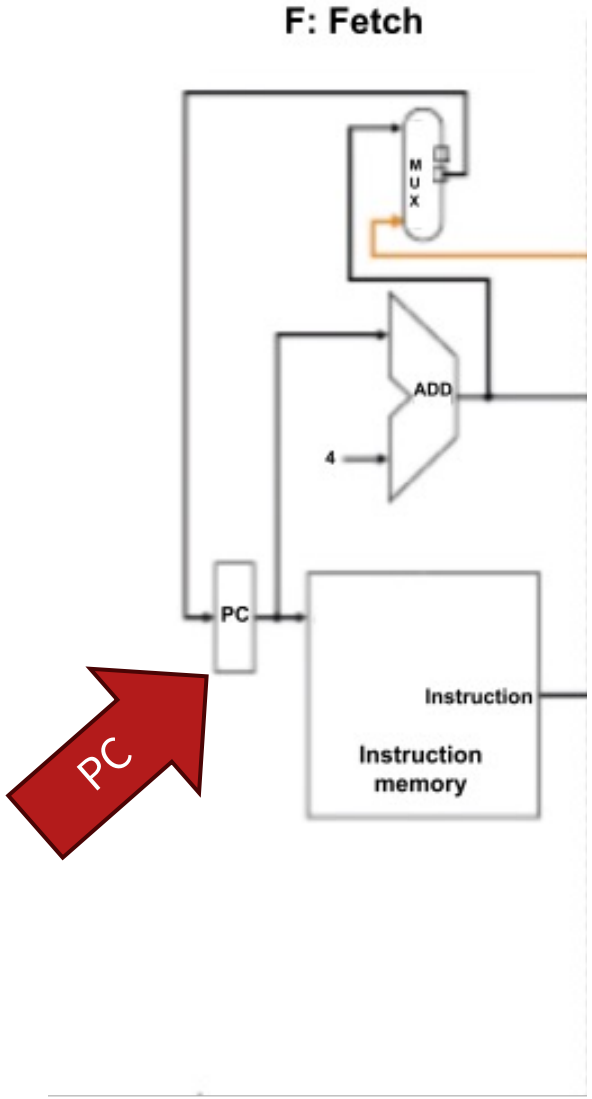
RISC-V Assembly

```
   beq x1, x2, EXIT
   addi x3, x3, 42
EXIT:
   addi x3, x3, 27
```

Today's Plan

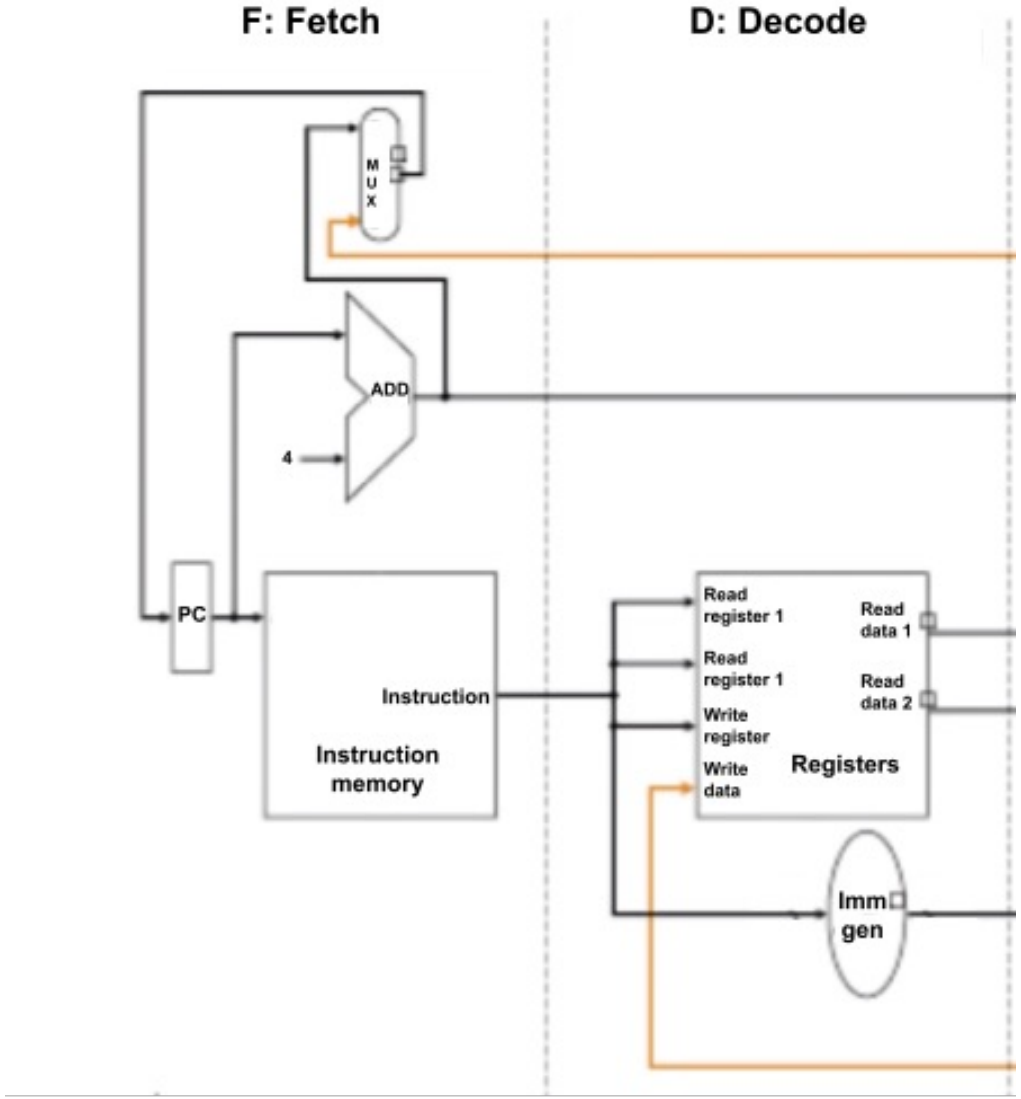
- RISC-V Instructions
  - Loads & Stores
  - Branch If Equal
- **Stages of a CPU**
- End early!

Stage 1: Fetch



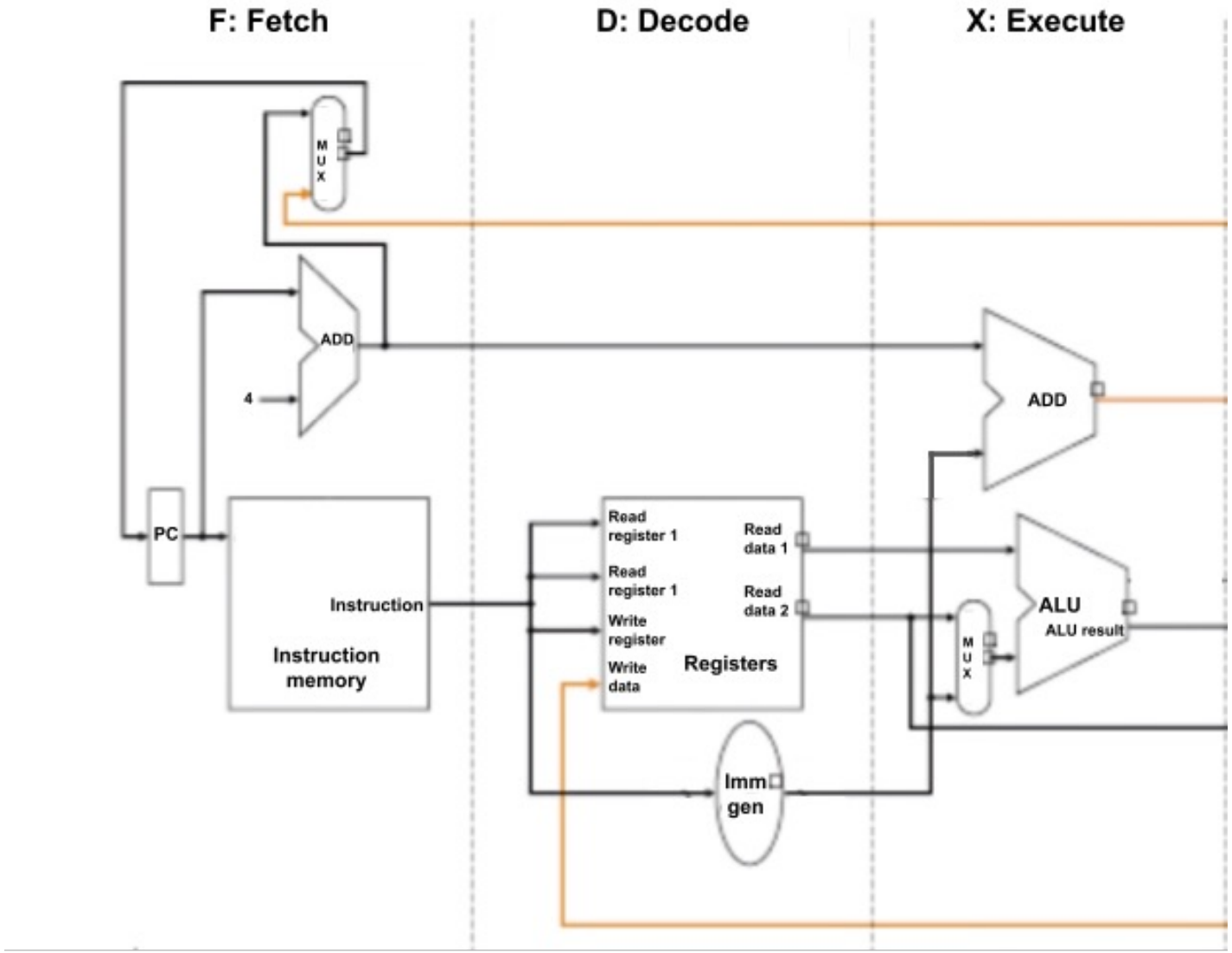
Fetch the next instruction from the instruction memory

Stage 2: Decode



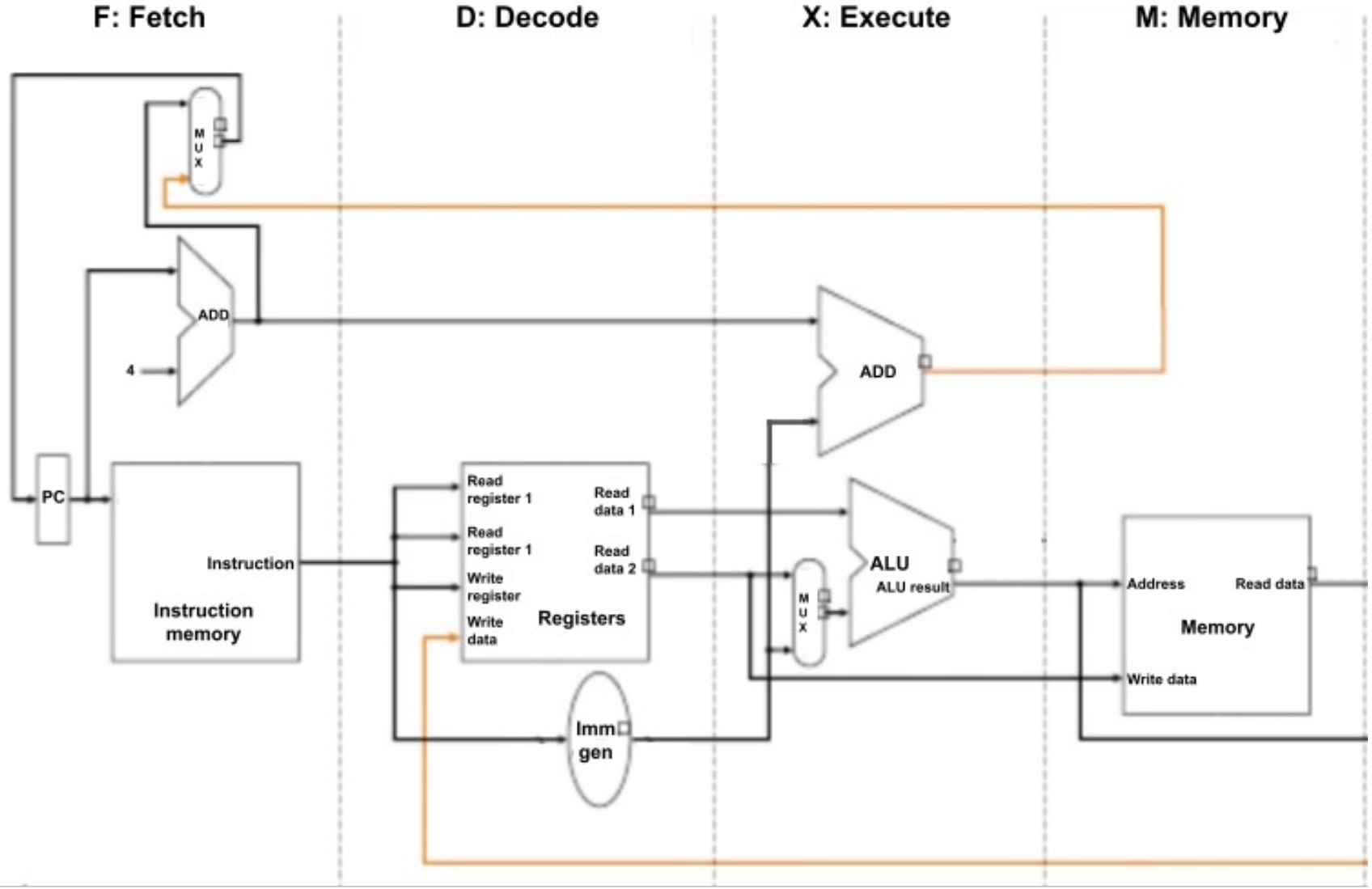
Decode the instruction bits, producing control signals

Stage 3: Execute



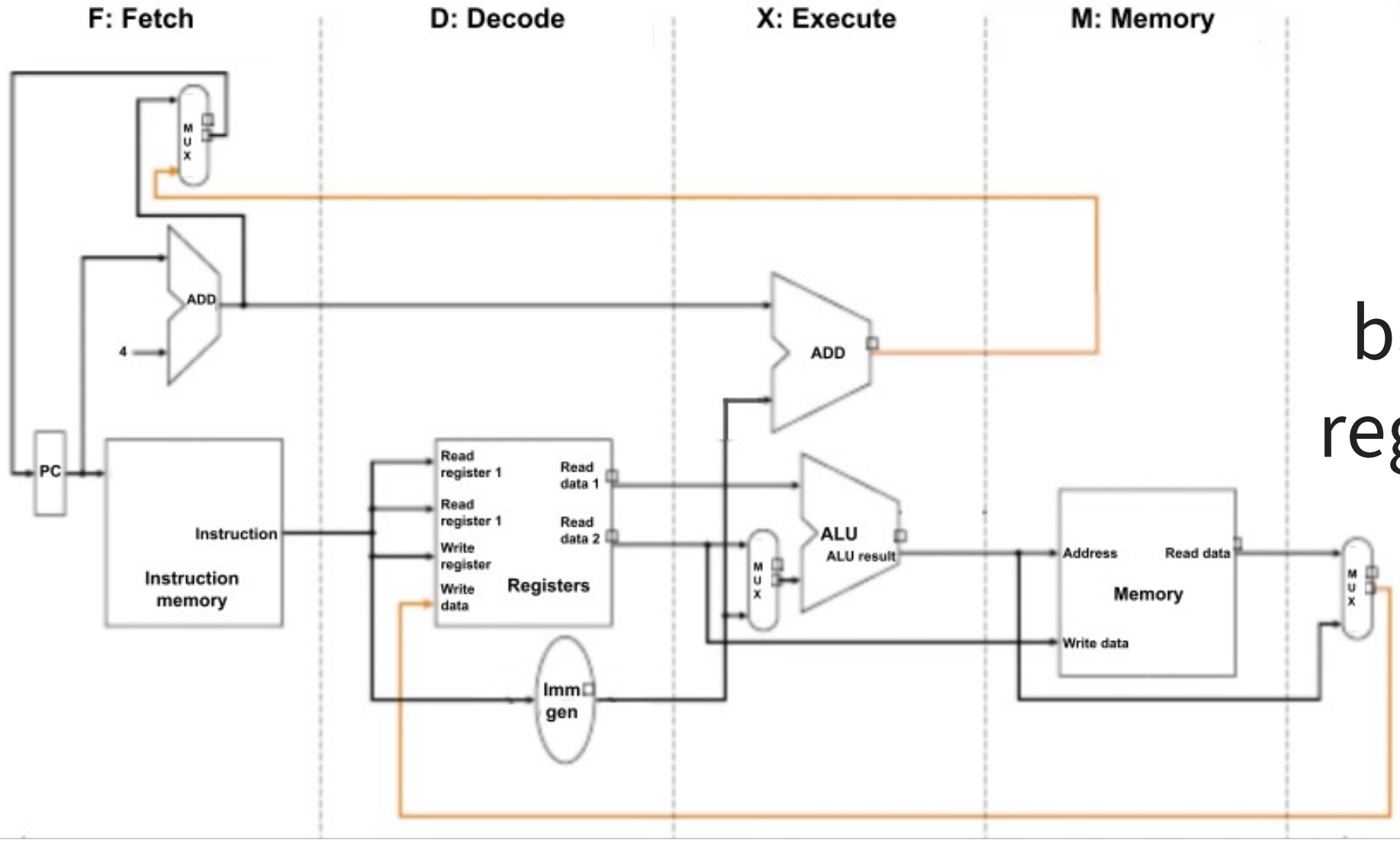
Execute the instruction using the ALU

Stage 4: Access Memory



Access ***memory***

Stage 5: Write back



Write results back into register file

Today's Plan

- RISC-V Instructions
  - Loads & Stores
  - Branch If Equal
- Stages of a CPU
- **End early!**