pollev.com/kakiryan

COMP311: *COMPUTER ORGANIZATION!*

Lecture 21: I-Type Instructions!

tinyurl.com/comp311-fa25

RISC-V ASSEMBLY! ☺ YAY YAY YAY FUN FUN

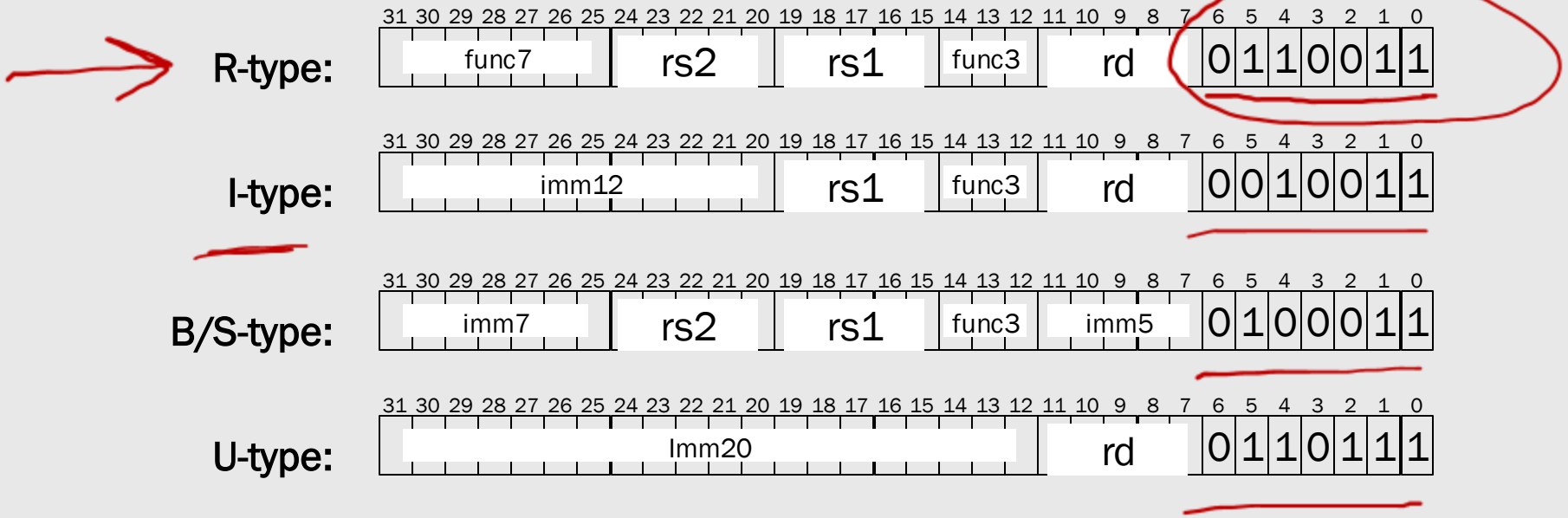
Reminder: How are programs turned into machine code?

High-level language → c = a + b

Assembly → add x5, x4, x3

Machine Code → 00000000001100100000001010110011

The miniRISC-V ISA



Four key instruction formats (each one has a corresponding opcode!):

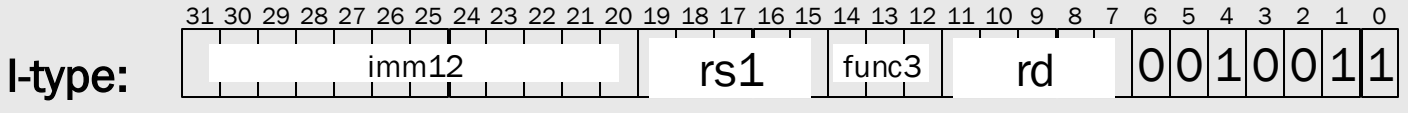
1) ALU with two register operands
2) ALU with a register and an immediate operand
3) Stores and Branches
4) Jumps and large constants (LUI, AUIPC)

R-format vs I-format Instructions

- R-format
  - *Used for operations between two registers*
- I-format
  - *Used for operations between a register and an immediate value*
    - in other words, if you want to use a constant that is not in a register

I-type Data Processing

Instructions that process one register and a constant:



Notice there is no SUBI instruction. Why?

000 - ADDI
001 - SLLI
010 - SLTI
011 - SLTUI
100 - XORI
101 - SRLI/SRAI
110 - ORI
111 - ANDI

```
andi   x5,x10,255
```

I-type instructions have the following template:

```
OPfunc3   rd,rs1,imm12
```

Is encoded as:

0000 1111 1111 0101 0111 0010 1001 0011

```
0x0ff57293
```

I-Format Syntax

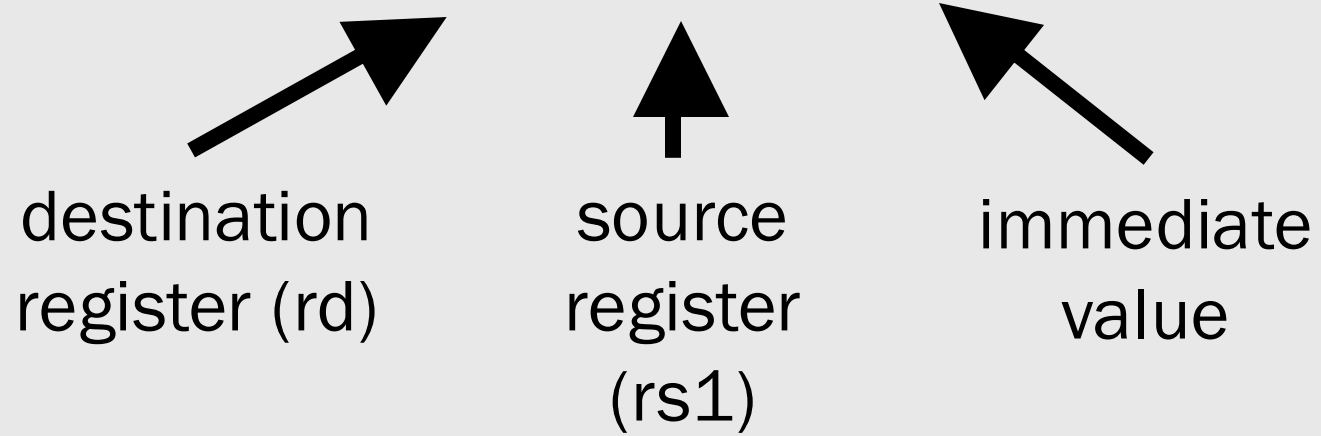
op rd, rs1, immediate12

this is where we will store
the result of the operation

I-Format Syntax

x5 = x4 + 12

addi x5, x4, 12



R-Format vs I-Format Syntax

R-Format

op rd, rs1, rs2

R[rd] = R[rs1] op R[rs2]

writes to rd

I-Format

op rd, rs1, immediate

R[rd] = R[rs1] op immediate

writes to rd

Register 0

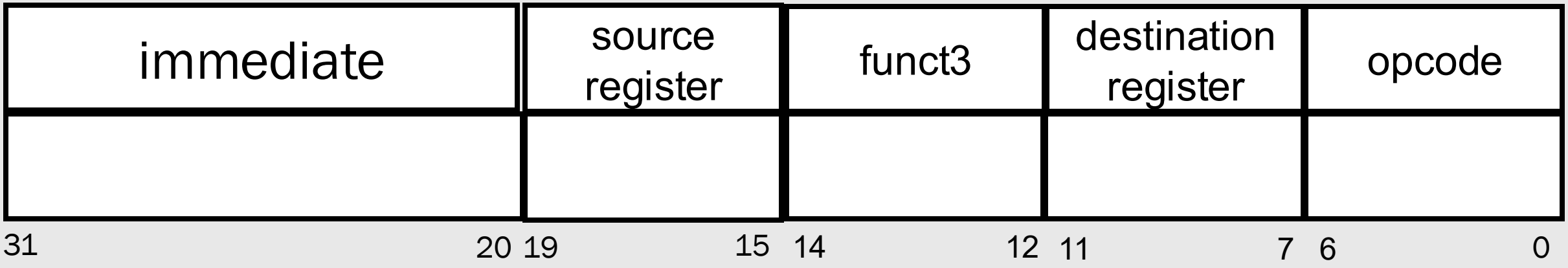
- Register 0 always holds the value 0
- This is very useful when we want to write a constant value to a register

Example Program

```
a = 5                    addi x10, x0, 5
b = 3                    addi x11, x0, 3
c = a + b                add x12, x10, x11
```

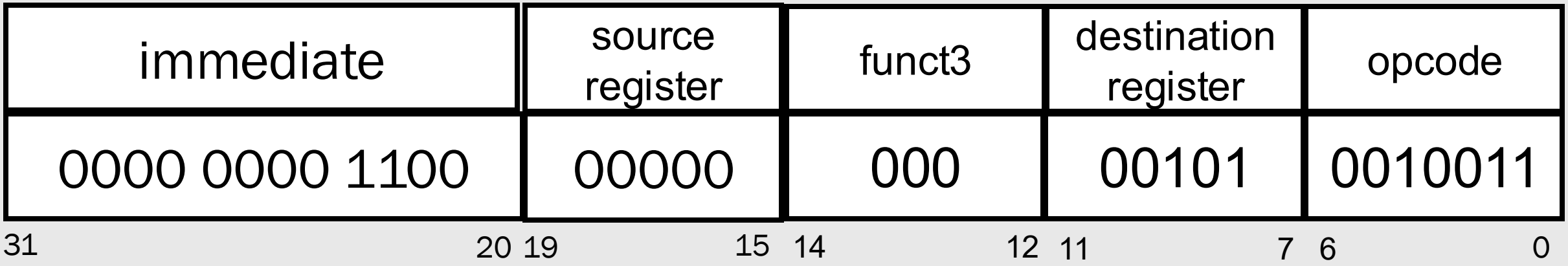
I-Format Syntax

addi x5, x0, 12



I-Format Syntax

addi x5, x0, 12

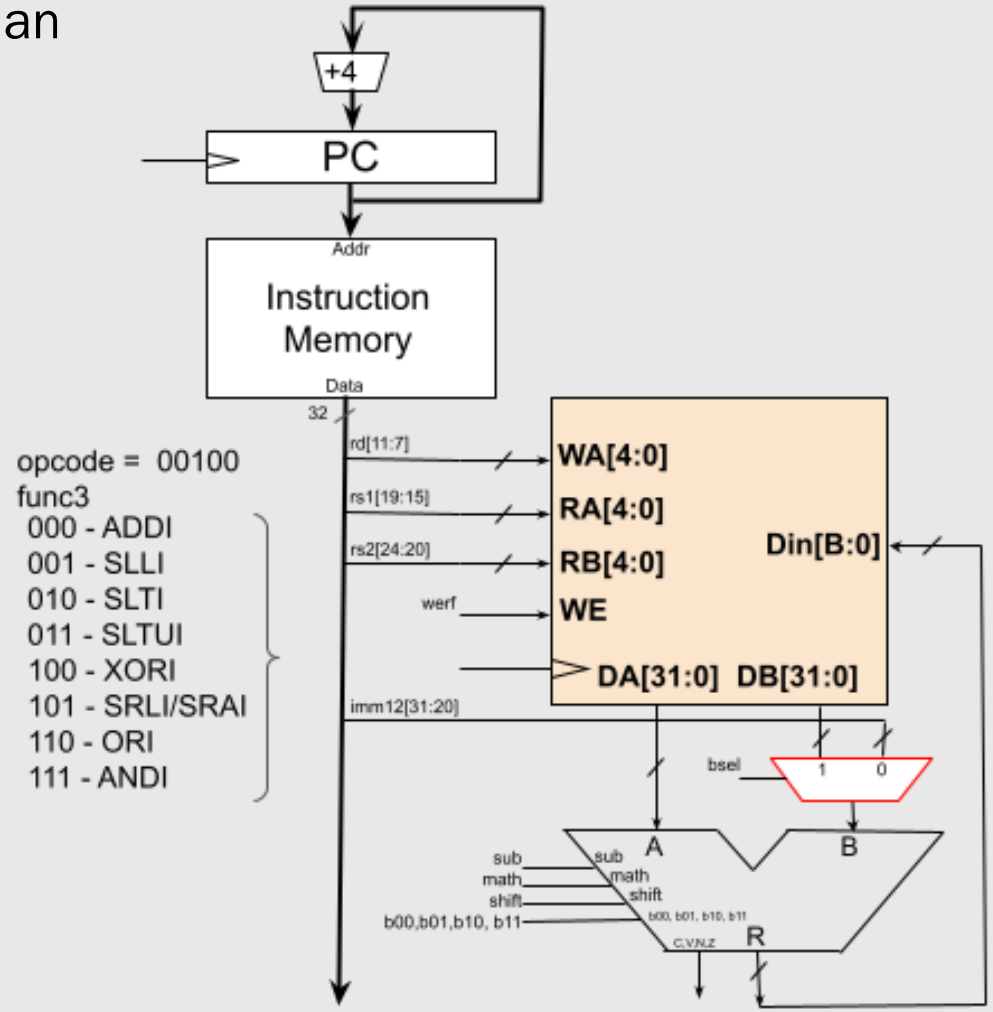


I-type Data Processing

ALU instructions with a register and an immediate operand

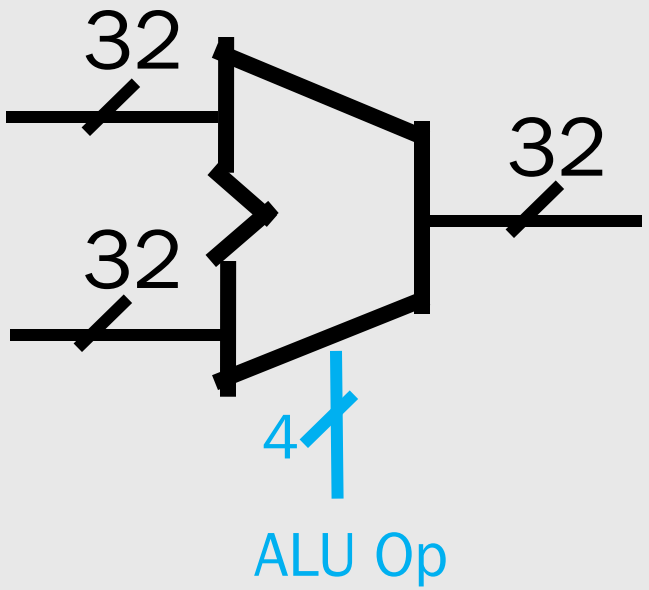
Rd - register file write address
Rs1 - register source operand
Imm12 - 12-bit immediate operand

Adds a mux to the B input of the ALU
- 12-bit immediate value is sign-extended 20-bits



Bit Extension

- The ALU operates on 32-bit values
- We need to extend the immediate value to make it compatible with the ALU



Logical Operations: Zero Extend Immediate

16 bits

$$A = A_{15}\ A_{14}\ A_{13}\ A_{12}\ A_{11}\ A_{10}\ A_9\ A_8\ A_7\ A_6\ A_5\ A_4\ A_3\ A_2\ A_1\ A_0$$

32 bits

$$A = 0000000000000000\ A_{15}\ A_{14}\ A_{13}\ A_{12}\ A_{11}\ A_{10}\ A_9\ A_8\ A_7\ A_6\ A_5\ A_4\ A_3\ A_2\ A_1\ A_0$$

Arithmetic Operations: Sign Extend Immediate

16 bits

$$A = A_{15}\ A_{14}\ A_{13}\ A_{12}\ A_{11}\ A_{10}\ A_9\ A_8\ A_7\ A_6\ A_5\ A_4\ A_3\ A_2\ A_1\ A_0$$

32 bits

$$A = A_{15}\ A_{15}\ A_{15}\ A_{15}\ A_{15}\ A_{15}\ A_{15}\ A_{15}\ A_{15}\ A_{15}\ A_{15}\ A_{15}\ A_{15}\ A_{15}\ A_{15}\ A_{15}\ A_{14}\ A_{13}\ A_{12}\ A_{11}\ A_{10}\ A_9\ A_8\ A_7\ A_6\ A_5\ A_4\ A_3\ A_2\ A_1\ A_0$$

Sign Extension Preserves the Value

16 bits

A = 0000000000001100

A = 12

32 bits

A = 00000000000000000000000000001100

A = 12

Sign Extension Preserves the Value

16 bits

A = 1111111111111011

A = -5

32 bits

A = 11111111111111111111111111111011

A = -5

Zero Extension Changes Negative Values

16 bits

A = 1111111111111011

A = -5

32 bits

A = 00000000000000001111111111111011

A = 65531

Instruction Operations

op rd, rs, immediate

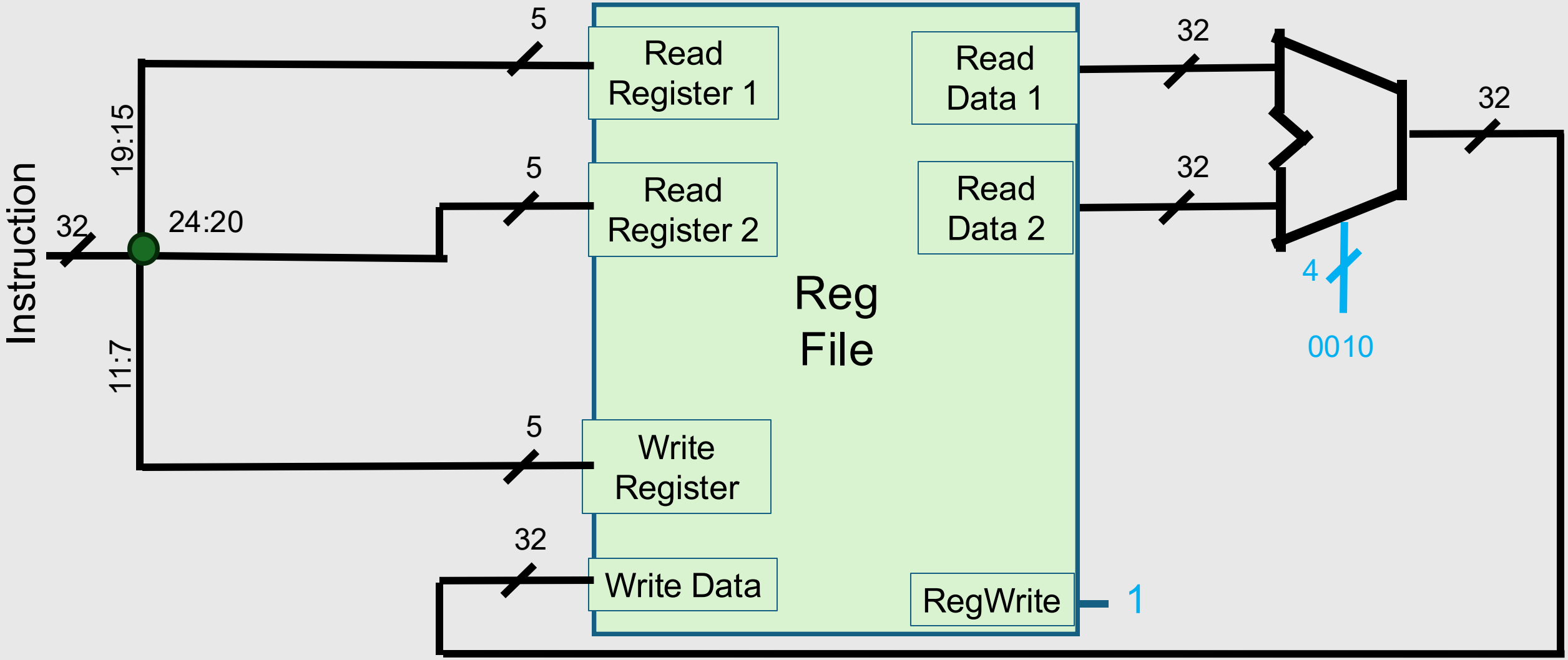
Name	Operation
addi	
andi	
ori	
slti	

Instruction Operations

op rd, rs1, immediate

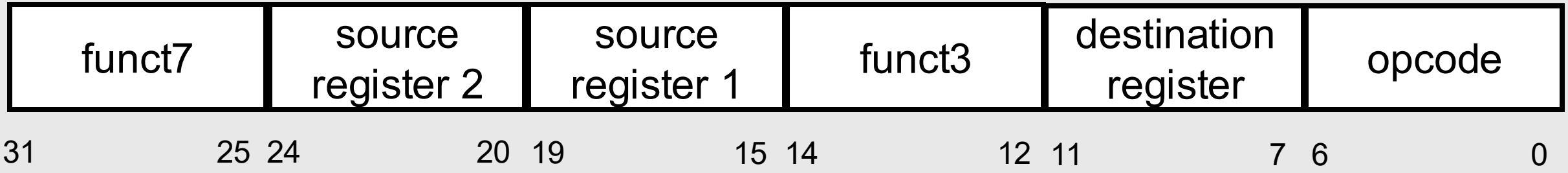
Name	Operation
addi	Reg[rd] ← Reg[rs1] + sign_extend(imm12)
andi	Reg[rd] ← Reg[rs1] & sign_extend(imm12)
ori	Reg[rd] ← Reg[rs1] | sign_extend(imm12)
slti	R[rd] = (R[rs1] < SignExtImm) ? 1 : 0

Original Datapath

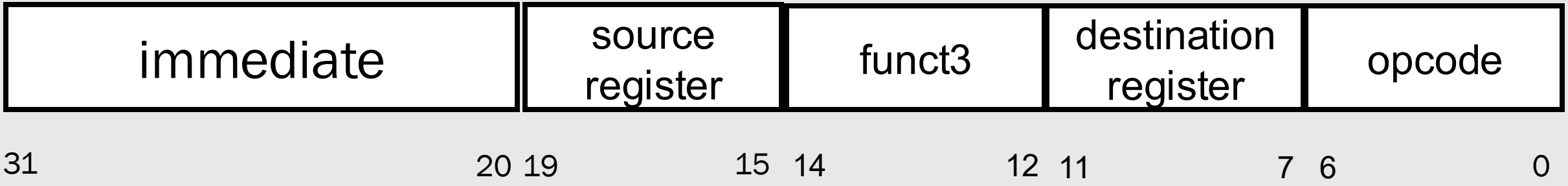


R-Format vs I-Format Instructions

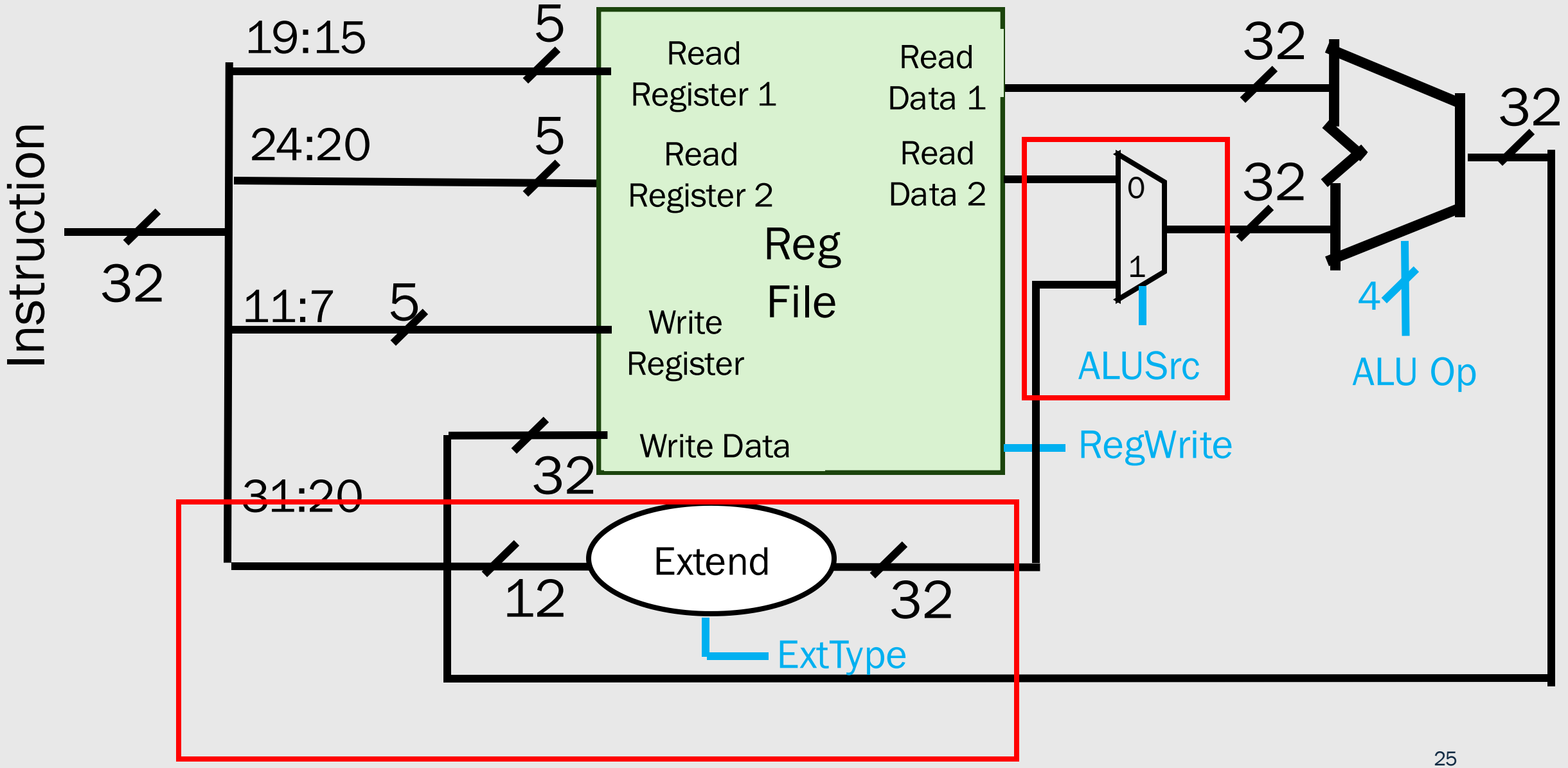
R-Format



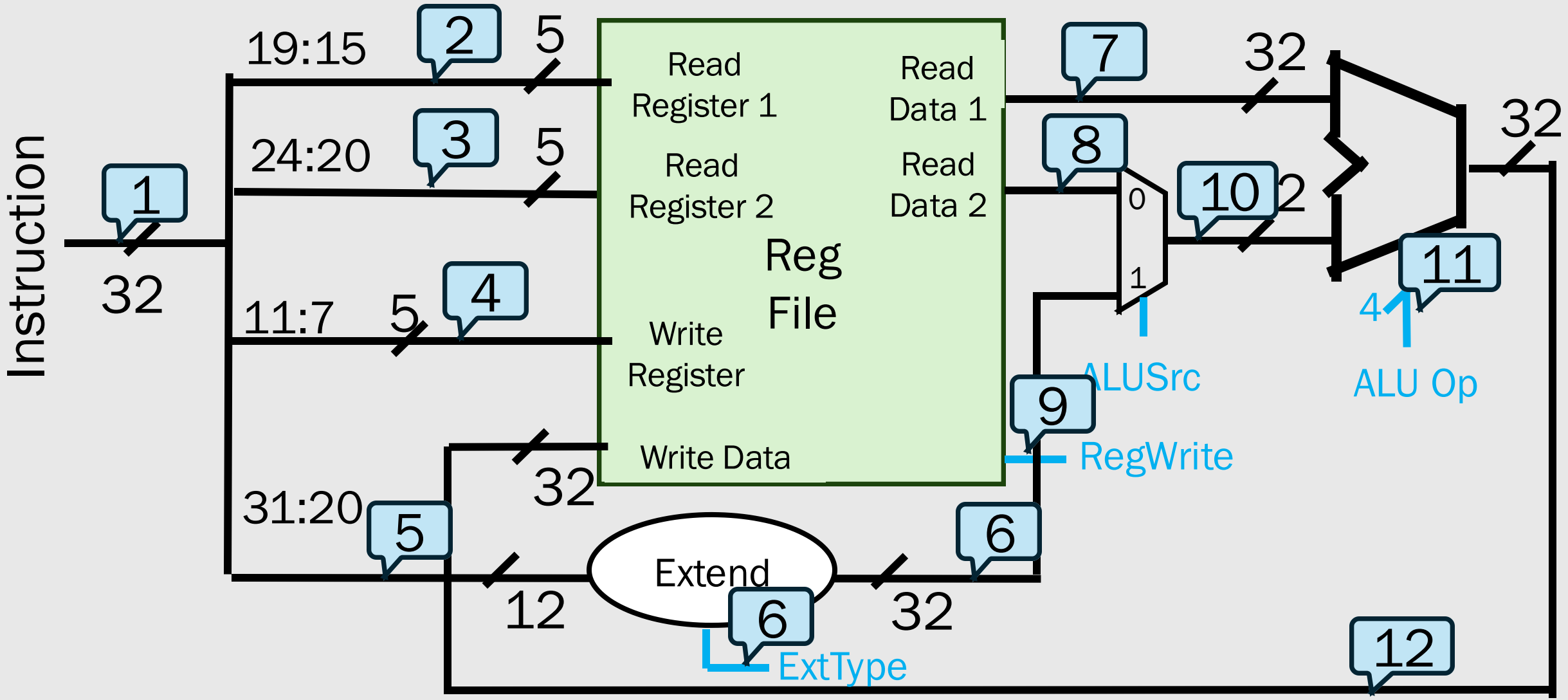
I-Format



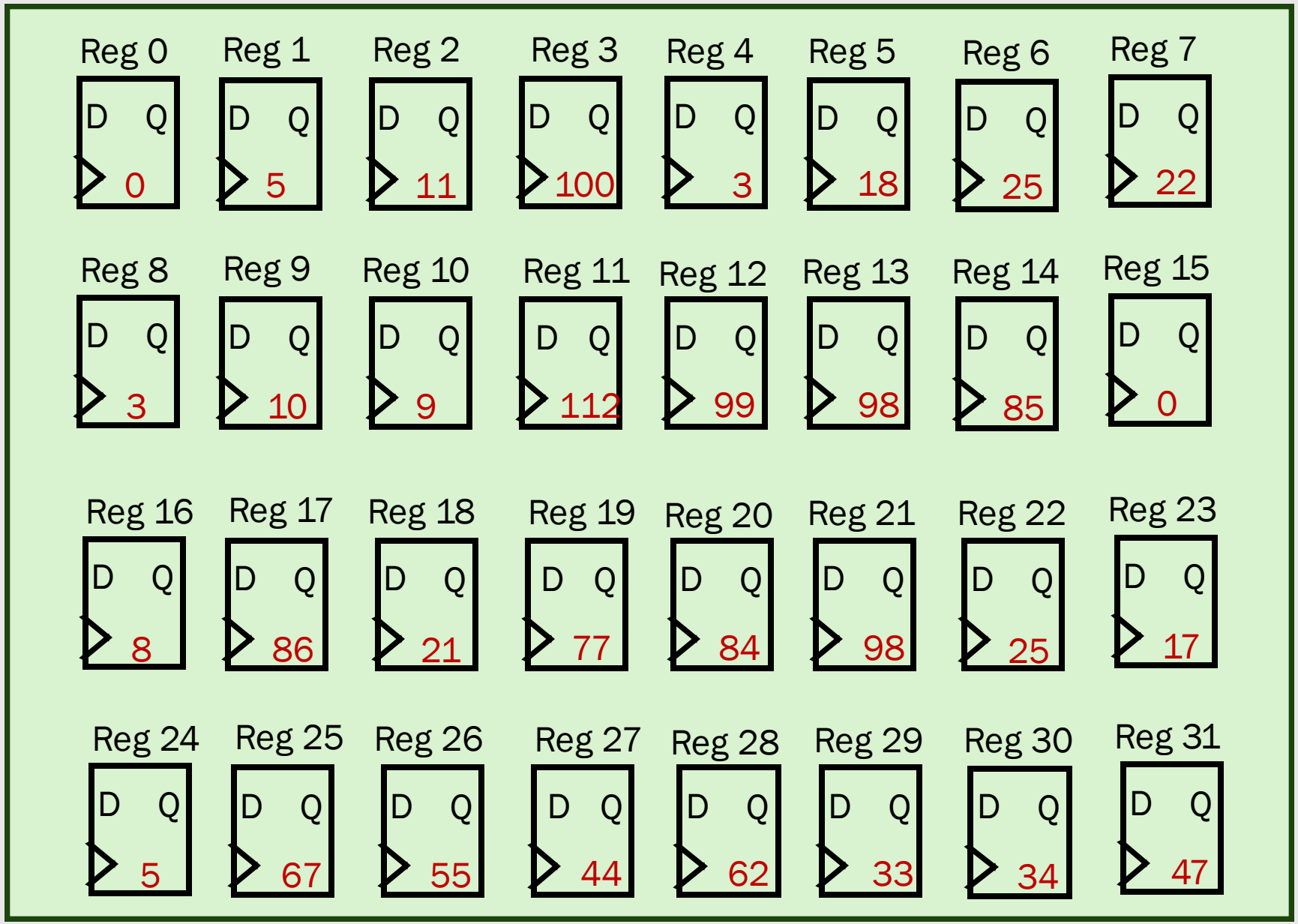
Hardware Support for I-Format Instructions



What is the value of each signal when executing `ori x4, x5 20`?



Contents of RegFile



What is the value of each signal when executing `ori x4, x5, 20`