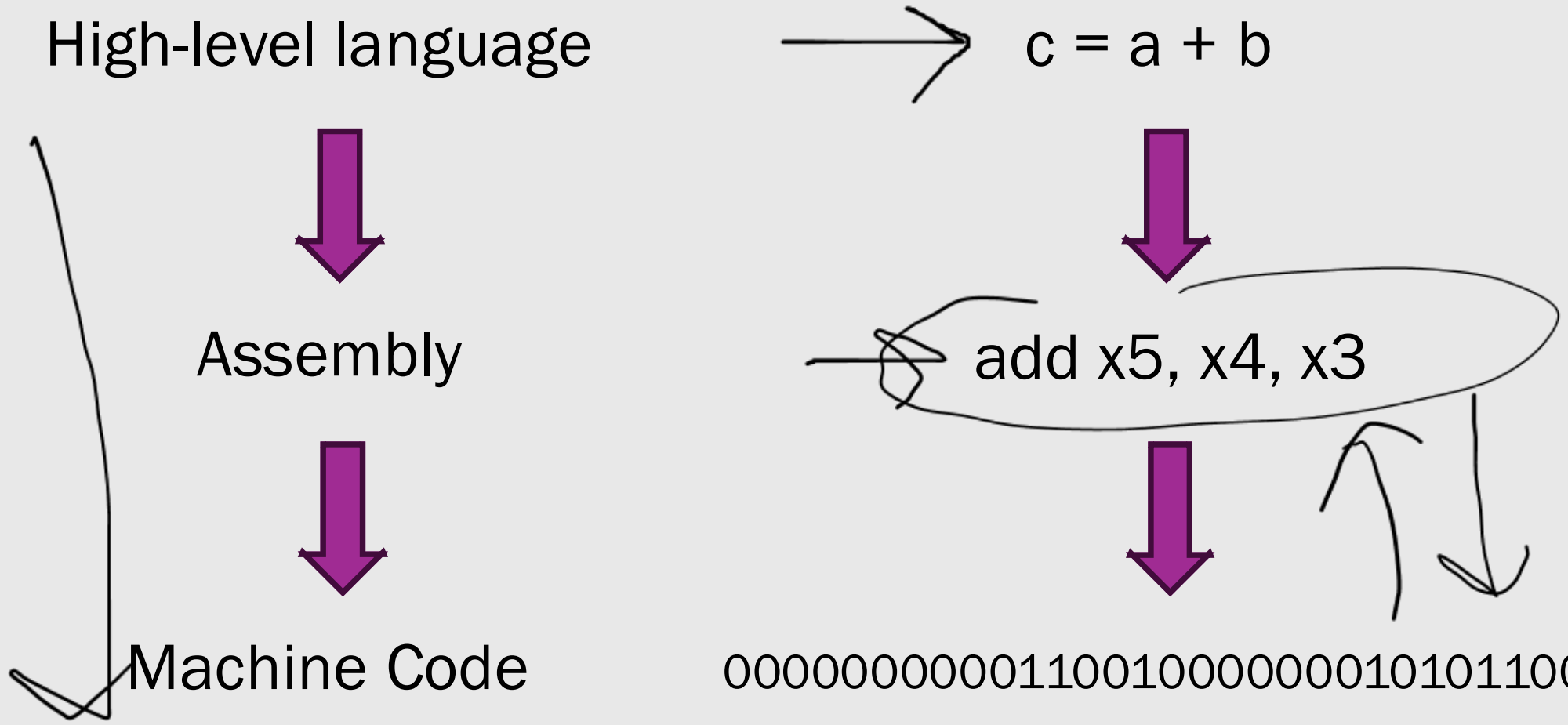
COMP311: *COMPUTER ORGANIZATION!*

Lecture 11: R-Type vs I-Type, Intro to the ALU

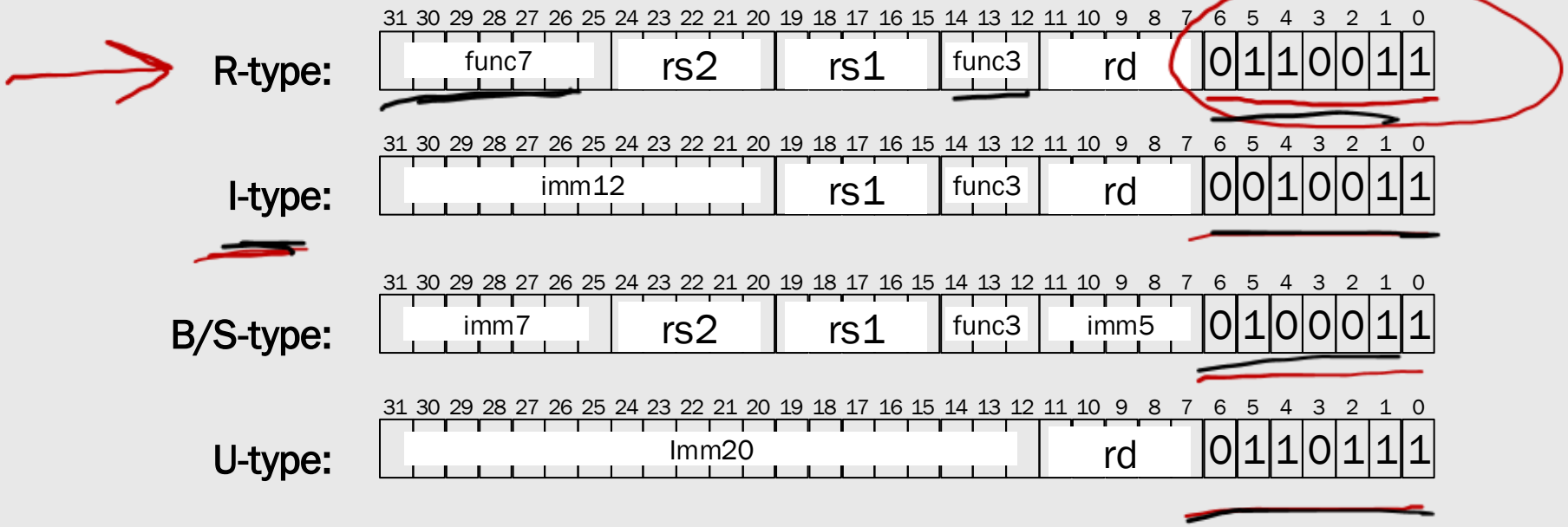
tinyurl.com/comp311-sp26

R-TYPE VS. I-TYPE INSTRUCTIONS

Reminder: How are programs turned into machine code?



The miniRISC-V ISA

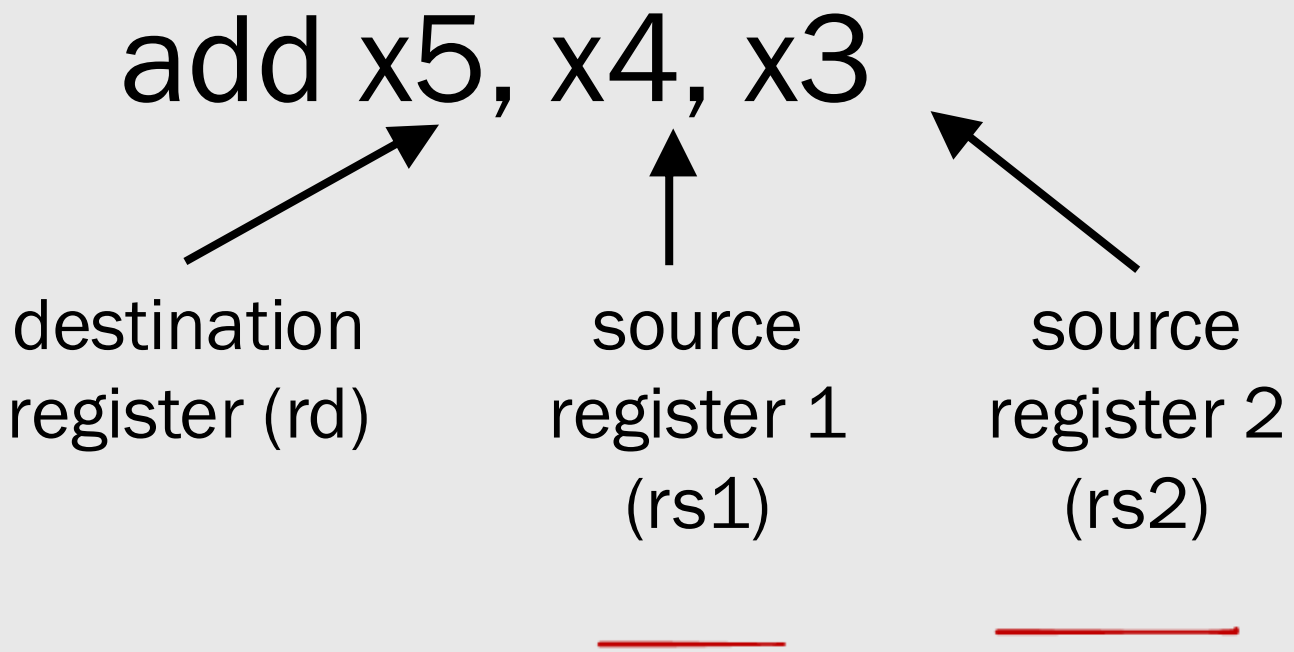


Four key instruction formats (each one has a corresponding opcode!):

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)

Syntax

register 5 = register 4 + register 3



R-Format Syntax

add x5, x4, x3

destination register (rd)

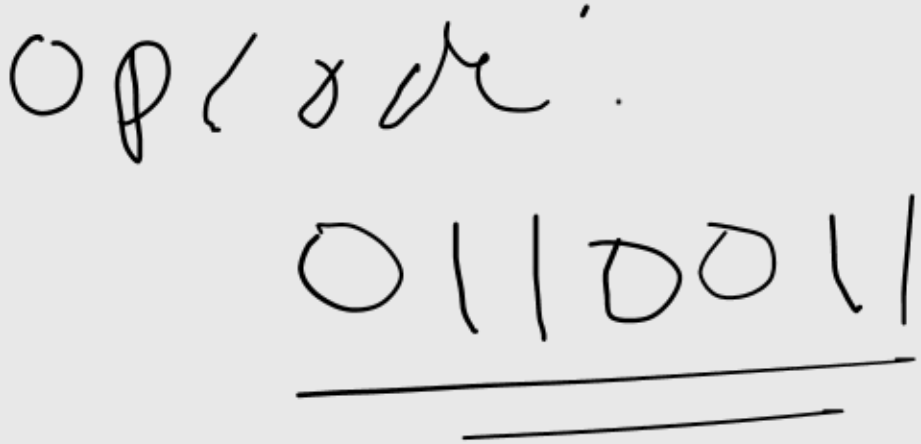
source register 1 (rs1)

source register 2 (rs2)

funct7	source register 2	source register 1	funct3	destination register	opcode
0000000	00011	00100	000	00101	0110011
31 25	24 20	19 15	14 12	11 7	6 0

R-Format Operations

- add
- sub
- AND
- OR
- SLT (set on less than)
- SLTU (Unsigned)
- SLL (shift left logical)
- SRL (shift right logical)
- SRA (shift right arithmetic)



funct7 / funct3
↳ choose the op

Subtract Instruction

sub rd, rs1, rs2

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

value of the register rd

value of the register rs1

value of the register rs2

32 regs!

Translating from C to RISC-V

```
c = a + b;
```

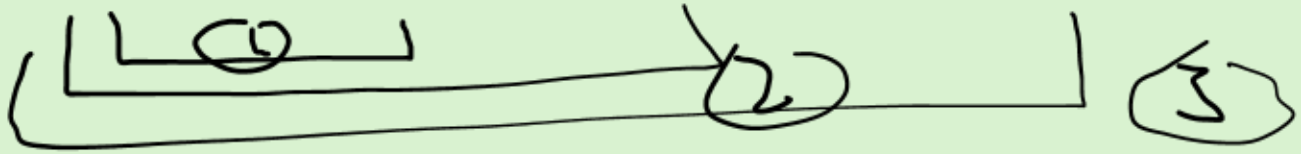
x5 x8 x9

```
add x5, x8, x9
```

Translate this line from C to RISC-V

```
a = b + c + d - e;
```

x14 x15 x16 x17 x18



You may use additional registers in your solution

1) add x14, x15, x16

2) add x14, x14, x17

3) sub x14, x14, x18

Translate this line from C to RISC-V

```
a = b + c + d - e;
```

x14 x15 x16 x17 x18

```
add x14, x15, x16
add x14, x14, x17
sub x14, x14, x18
```

or

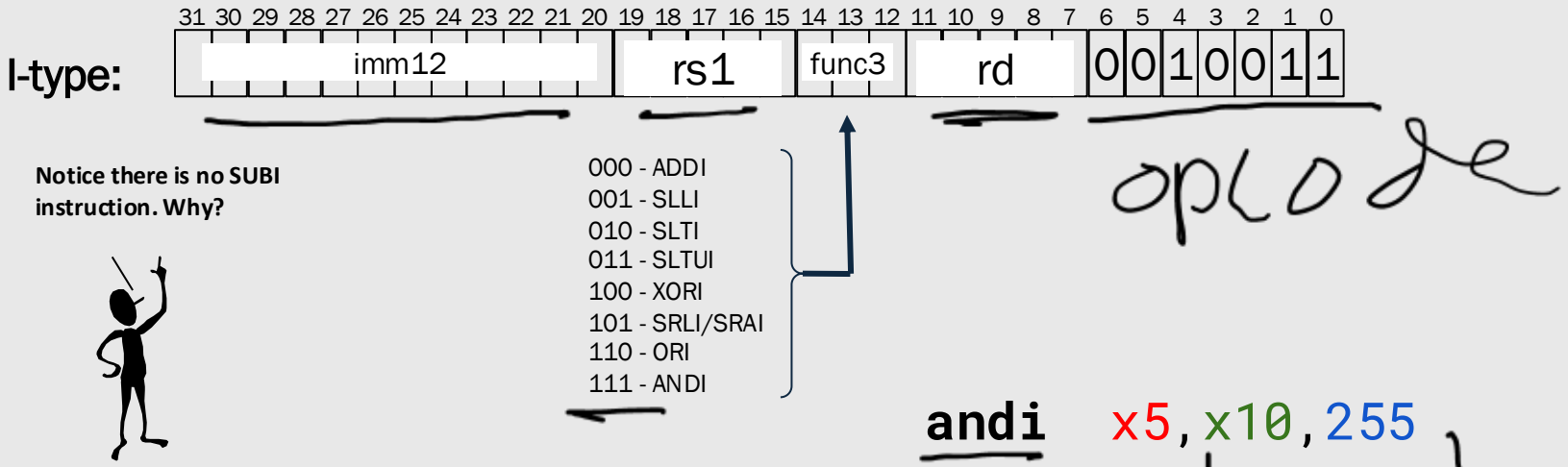
```
add x5, x15, x16
sub x6, x17, x18
add x14, x5, x6
```

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:



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

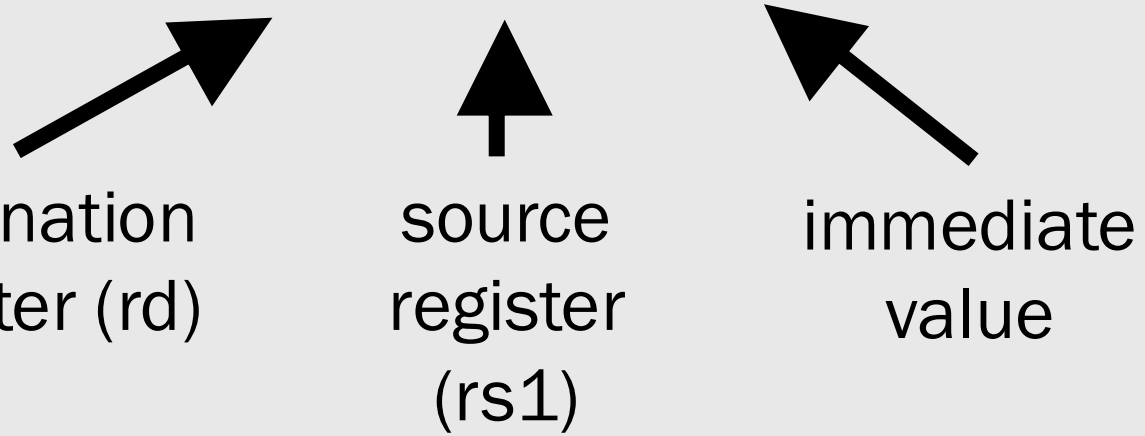
op rd, rs1, immediate12

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

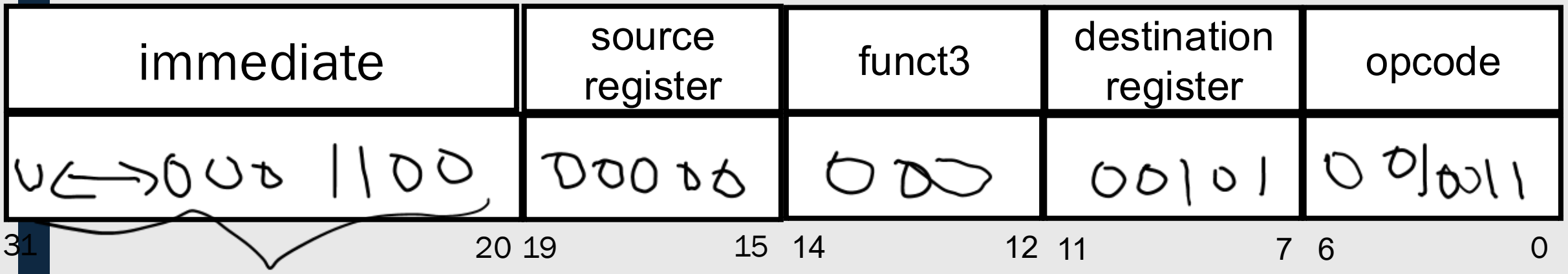
Register 0

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

I-Format Syntax

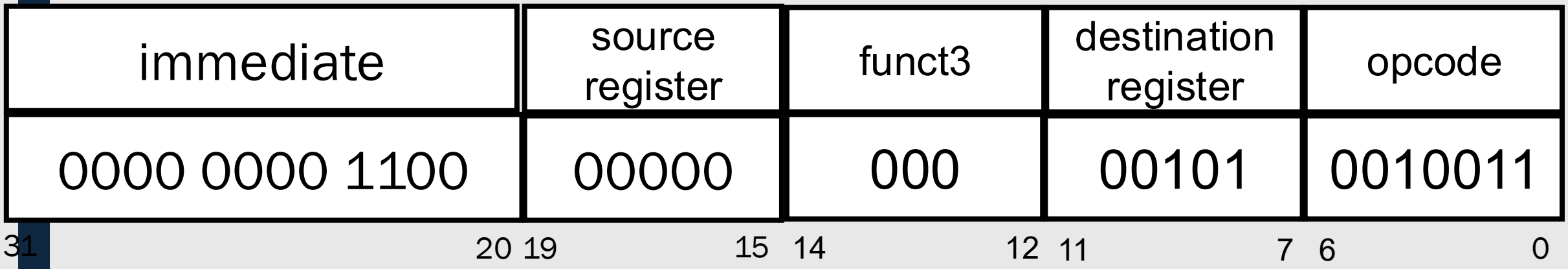
addi x5, x0, 12

x5 = x0 + 12 = 12



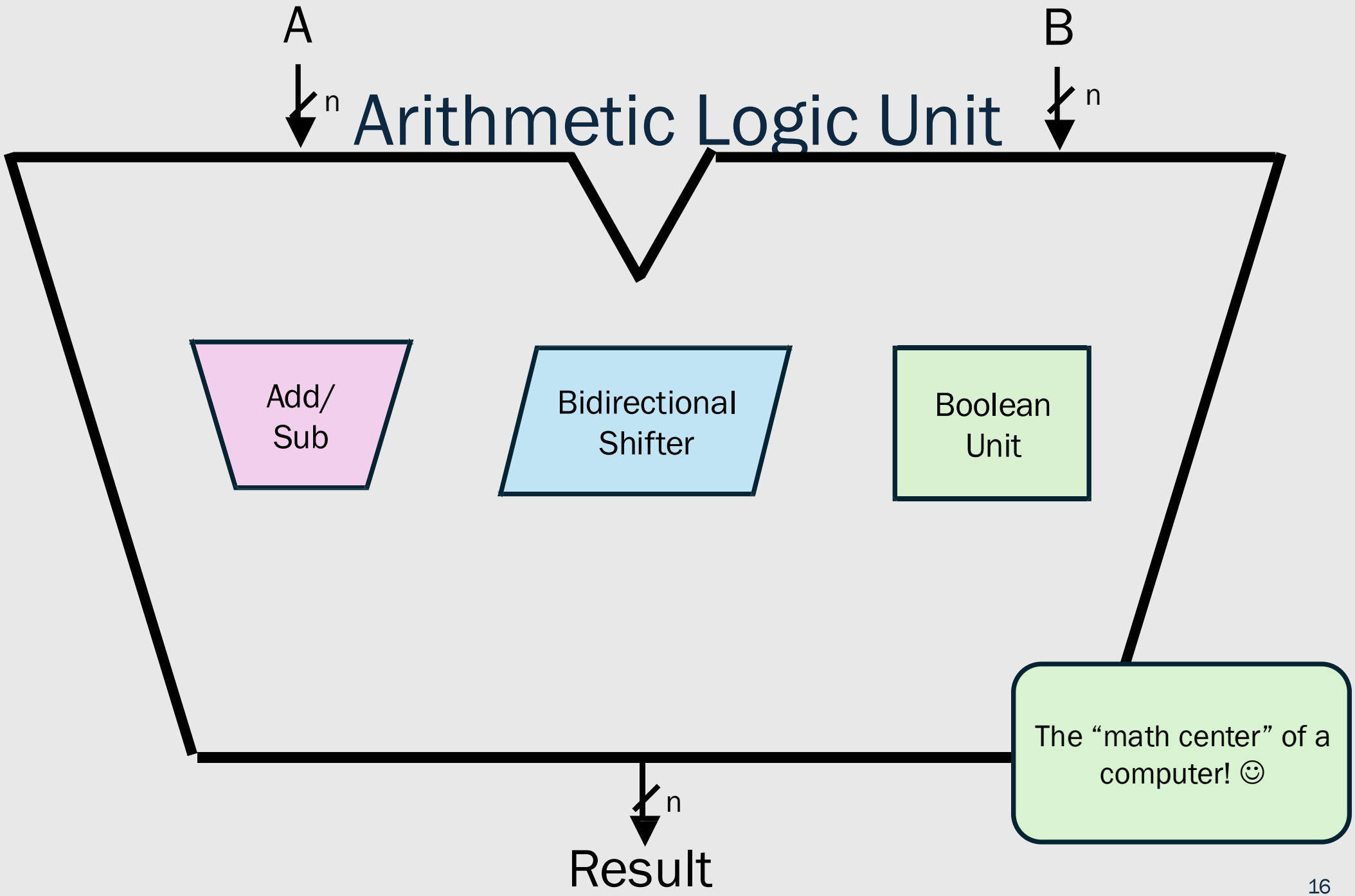
I-Format Syntax

addi x5, x0, 12

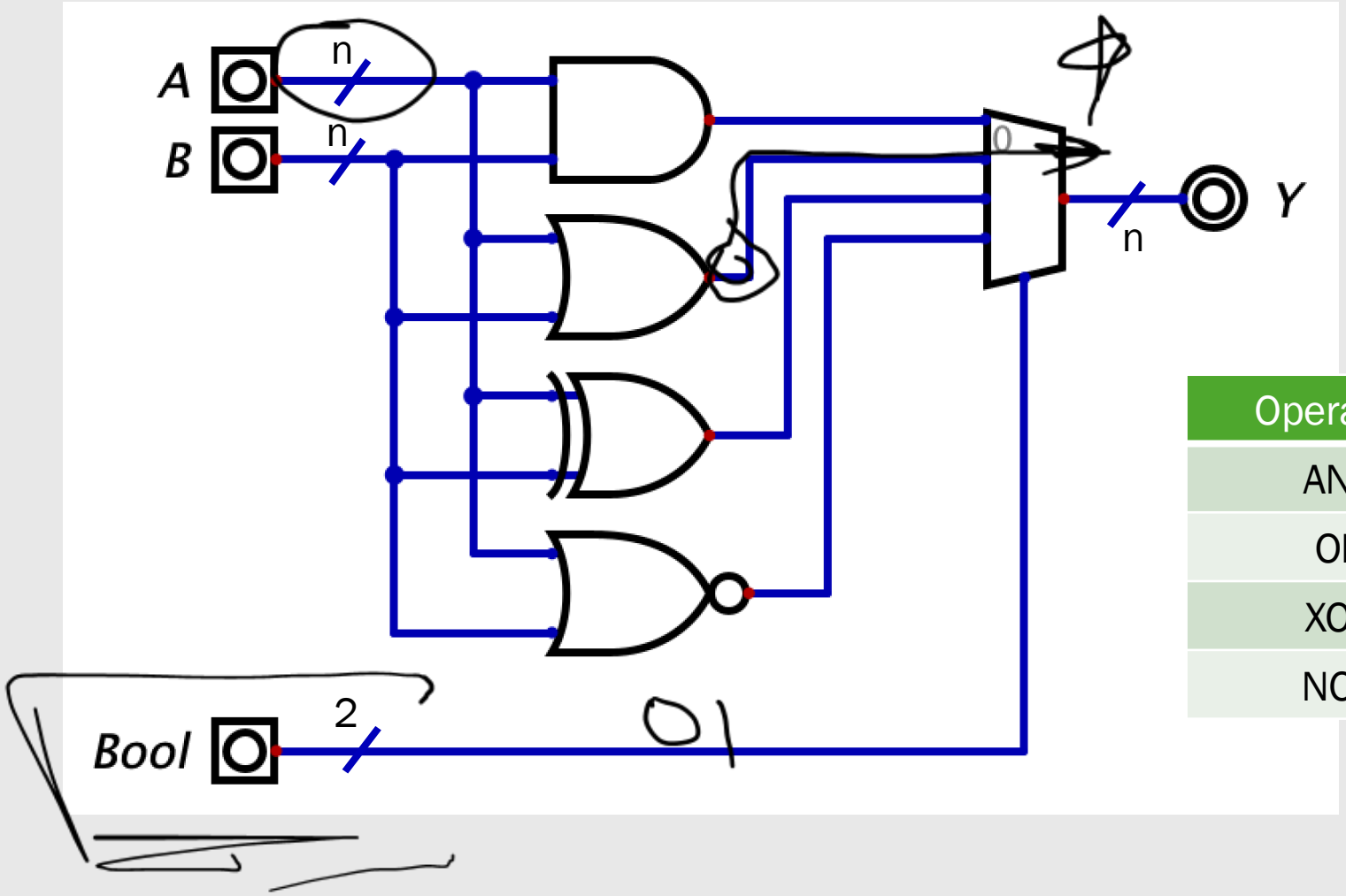


ARITHMETIC LOGIC UNIT (ALU) ☺

Arithmetic Logic Unit

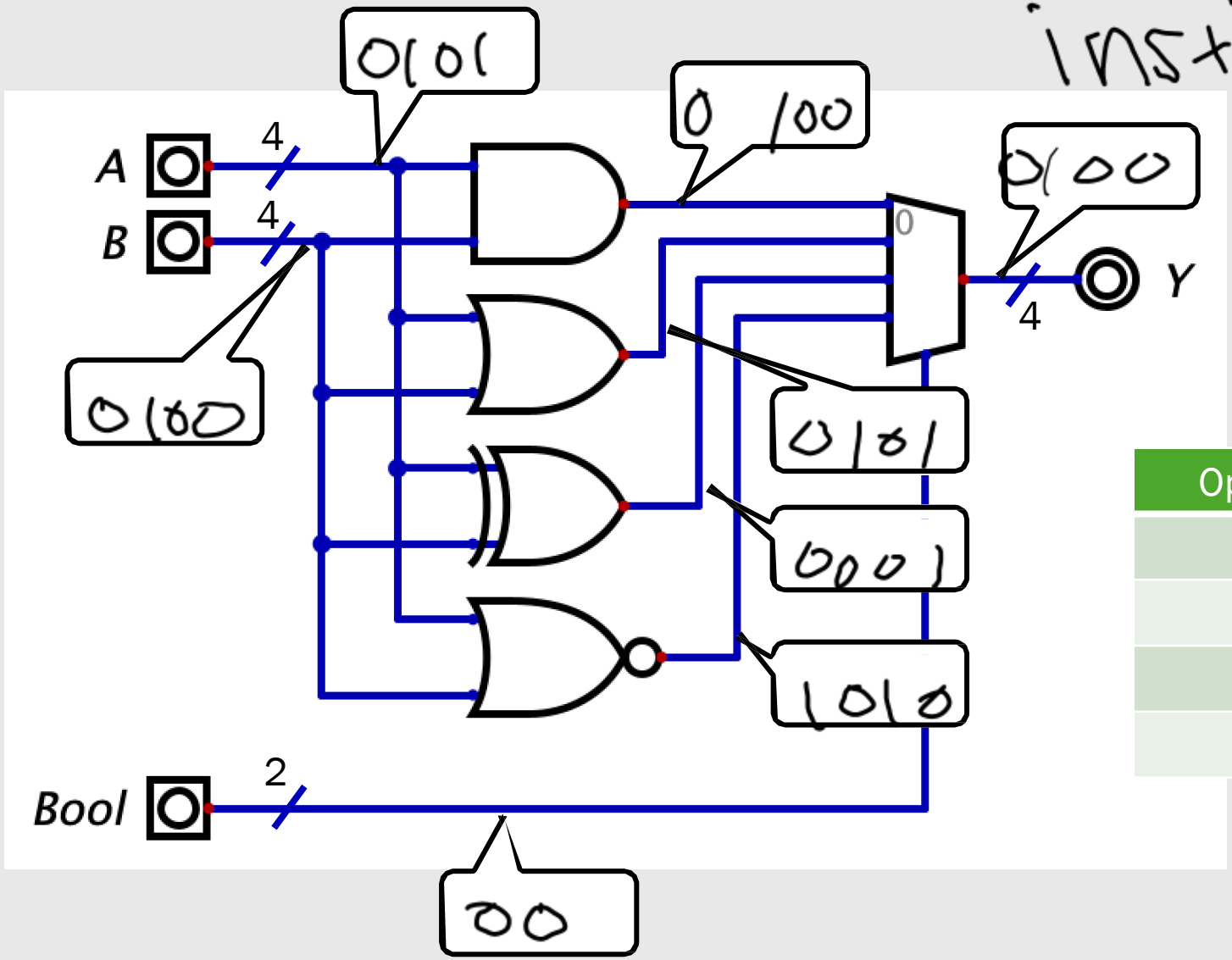


An Example Boolean Unit



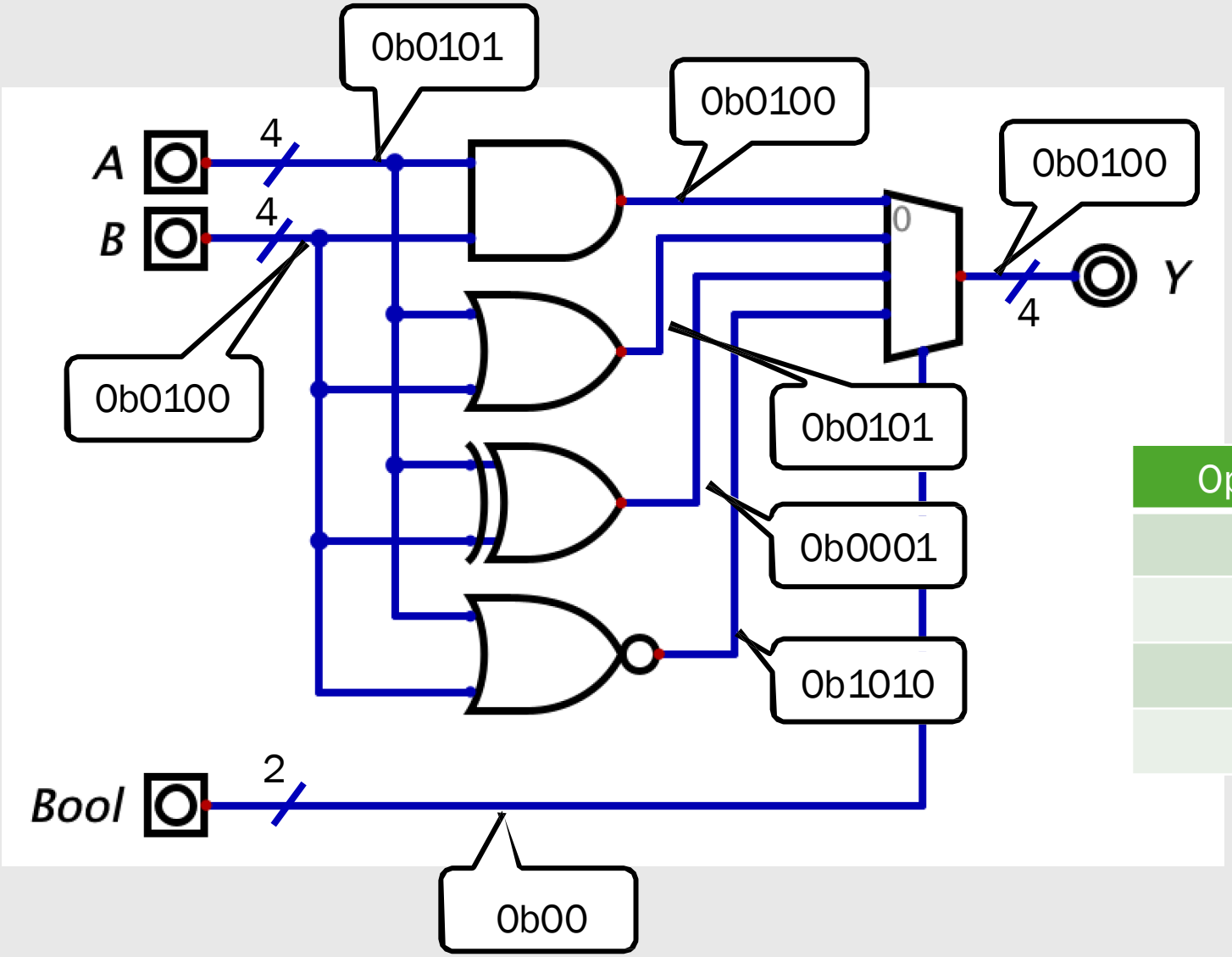
Operation	Expression	Bool
AND	A x B	0b00
OR	A + B	0b01
XOR	A + B	0b10
NOR	$\overline{A + B}$	0b11

Let's say n = 4. I want to perform Y = A AND B where A is 5 and B is 4.



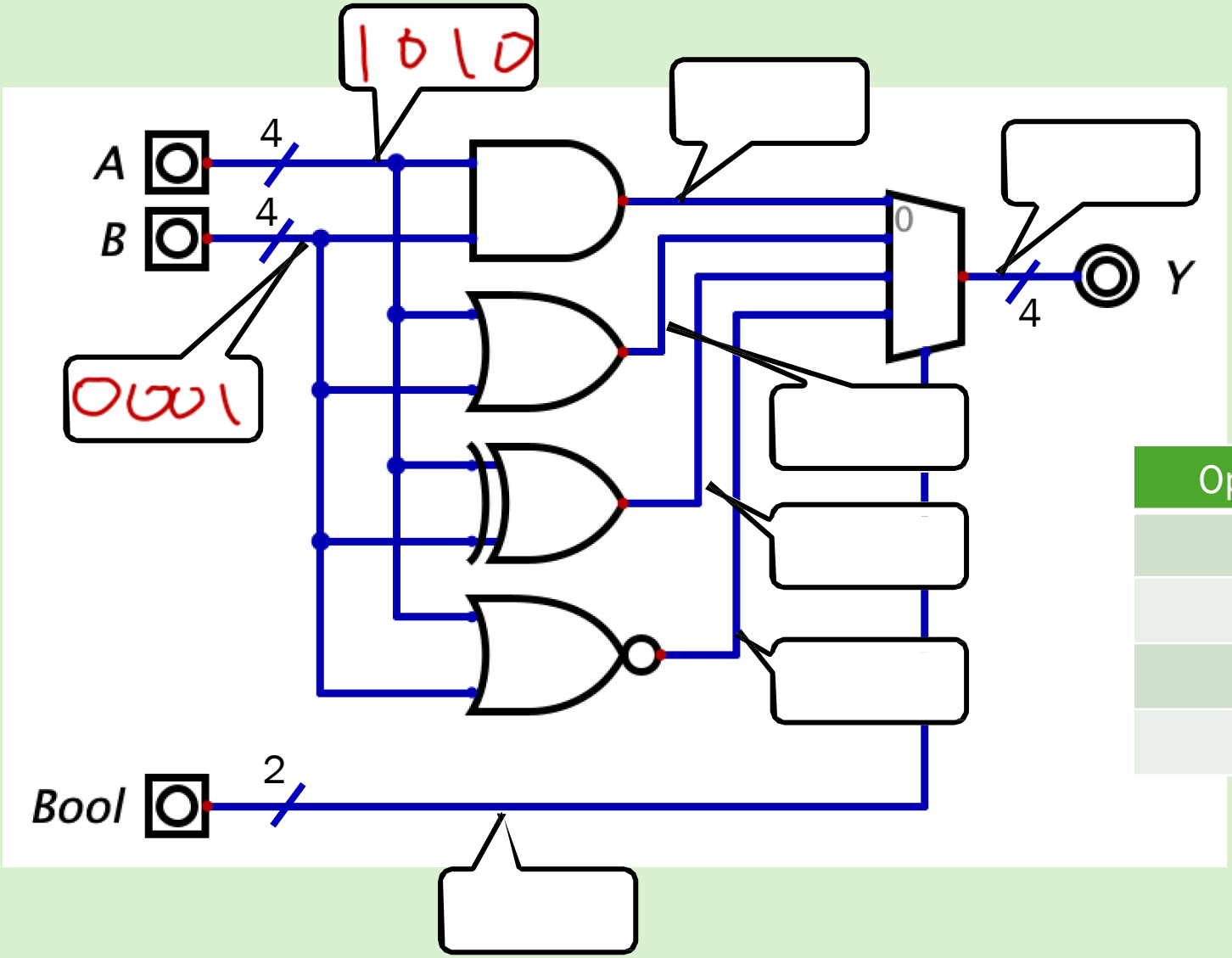
Operation	Expression	Bool
AND	A x B	0b00
OR	A + B	0b01
XOR	$A \oplus B$	0b10
NOR	$\overline{A + B}$	0b11

Let's say n = 4. I want to perform Y = A AND B where A is 5 and B is 4.



Operation	Expression	Bool
AND	A x B	0b00
OR	A + B	0b01
XOR	A⊕B	0b10
NOR	$\overline{A + B}$	0b11

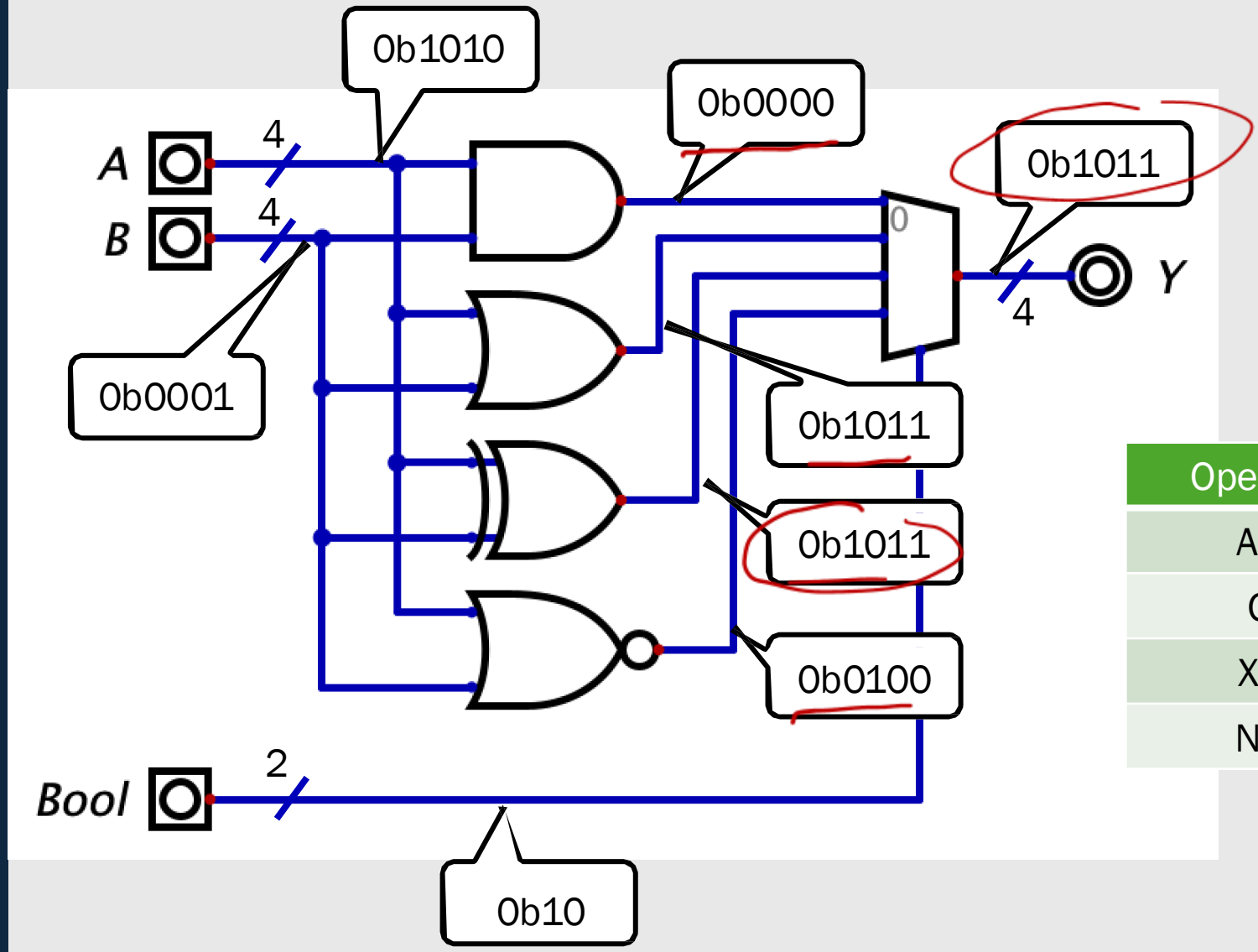
What are the values on each signal when we want to perform Y = A XOR B where A is 0b1010 and B is 0b0001?



instruction

Operation	Expression	Bool
AND	A x B	0b00
OR	A + B	0b01
XOR	$A \oplus B$	0b10
NOR	$\overline{A + B}$	0b11

What are the values on each signal when we want to perform Y = A XOR B where A is 0b1010 and B is 0b0001?



Operation	Expression	Bool
AND	A x B	0b00
OR	A + B	0b01
XOR	A⊕B	0b10
NOR	$\overline{A + B}$	0b11