pollev.com/kakiryan

COMP311: *COMPUTER ORGANIZATION!*

Lecture 20: R-Type & I-Type Instructions!

tinyurl.com/comp311-fa25

Quiz Topics!

There will be a seating chart!

- **Pipelining**
  - Performance Analysis of Single-Cycle vs. Multi-Stage Pipelined Implementations
- The Register File!
  - Conceptually, what it is, how it works, how to read/write to one.
- RISC-V Assembly
  - Given an instruction in hex or binary, translate it into assembly.
  - Given an instruction in assembly, translate it into machine code
  - Be able to fill in the R-Format instruction templates we have seen in class.
  - Be able to trace through the execution of an assembly instruction. Making correct updates to the register file. See the in-class problems we have done together the past 3-4 classes!

RISC-V ASSEMBLY! ☺ YAY YAY YAY FUN FUN

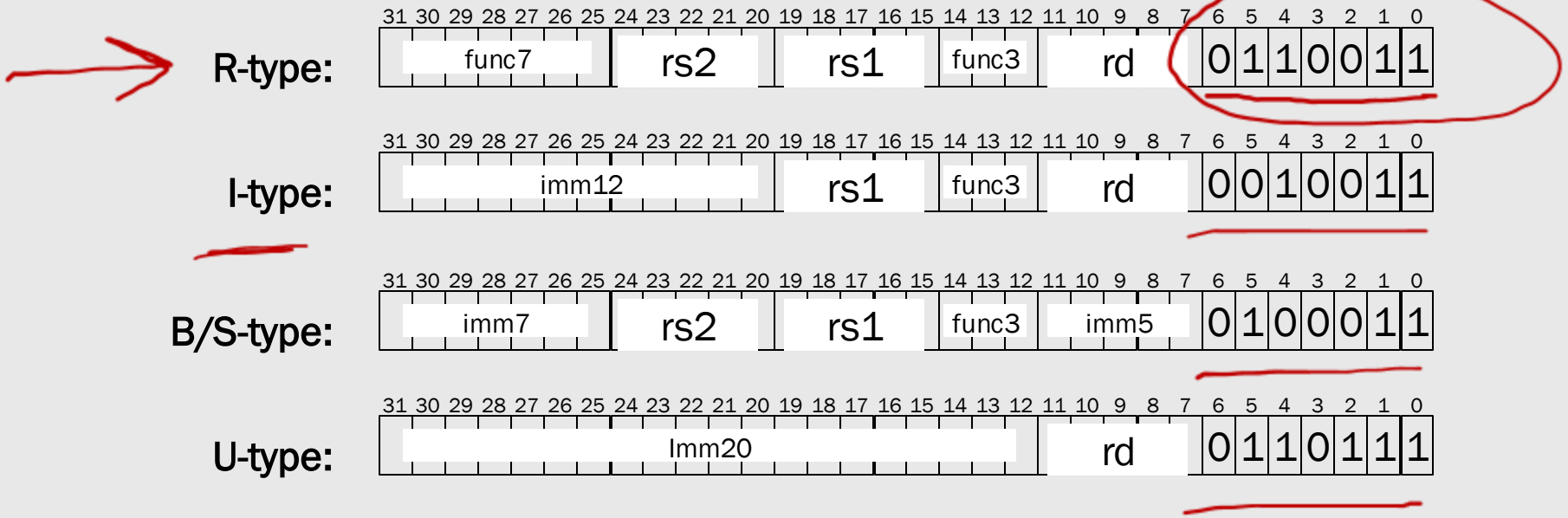
Reminder: How are programs turned into machine code?

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-type Data Processing

ALU instructions with register operands

Rd - register file write address
Rn, Rm - register source operands
Shift or Rs - Optional shift of Rm

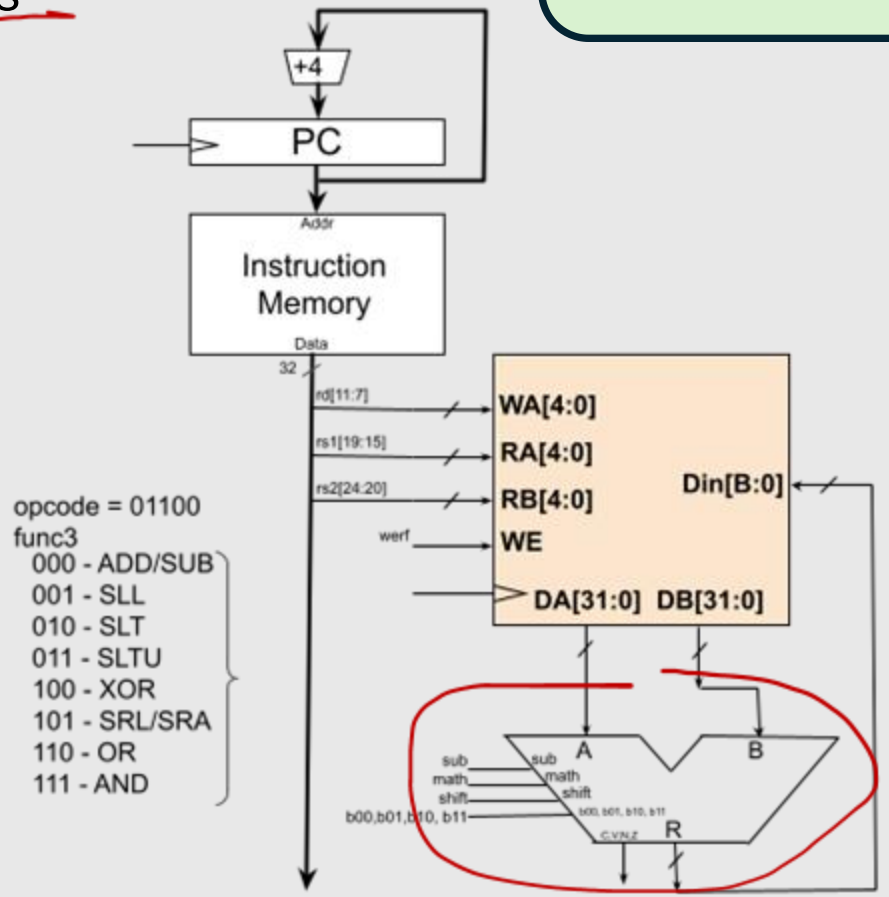
LA - direction and type of shift

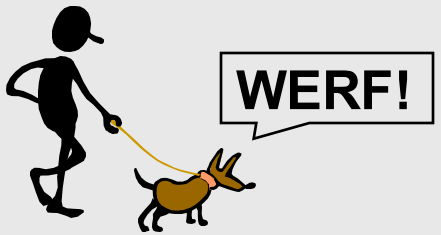
S-bit - controls update of PSR

Func decoding from ALU lecture

Register write back controlled by WERF logic

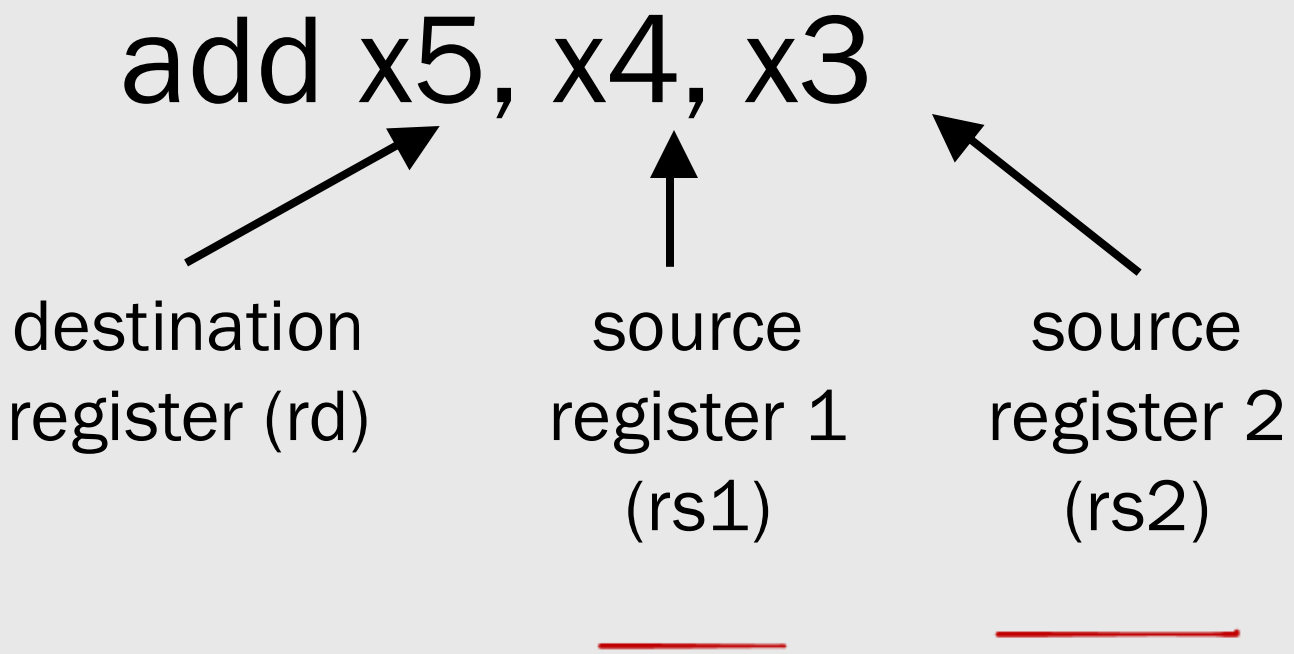
For R-Type and I-Type Instructions, the RegWrite signal in our data path will be 1.





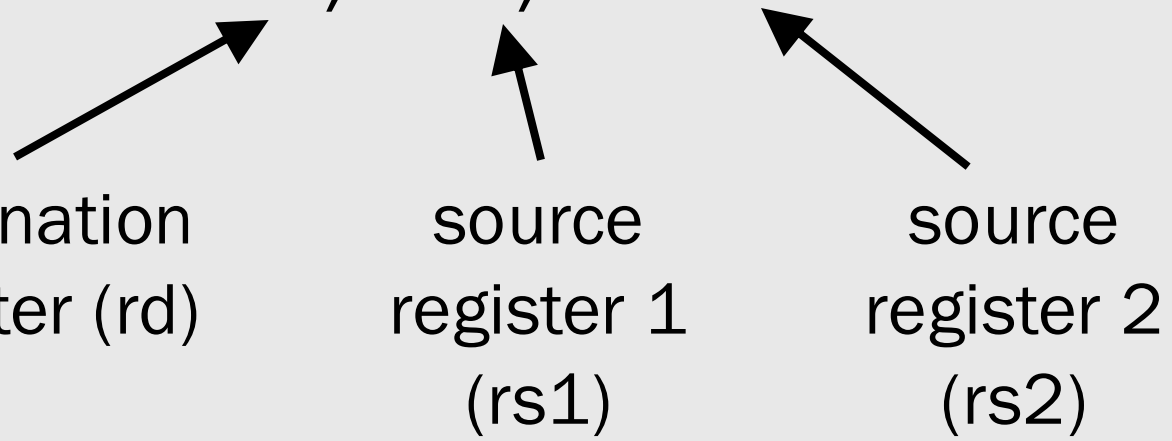
Syntax

register 5 = register 4 + register 3



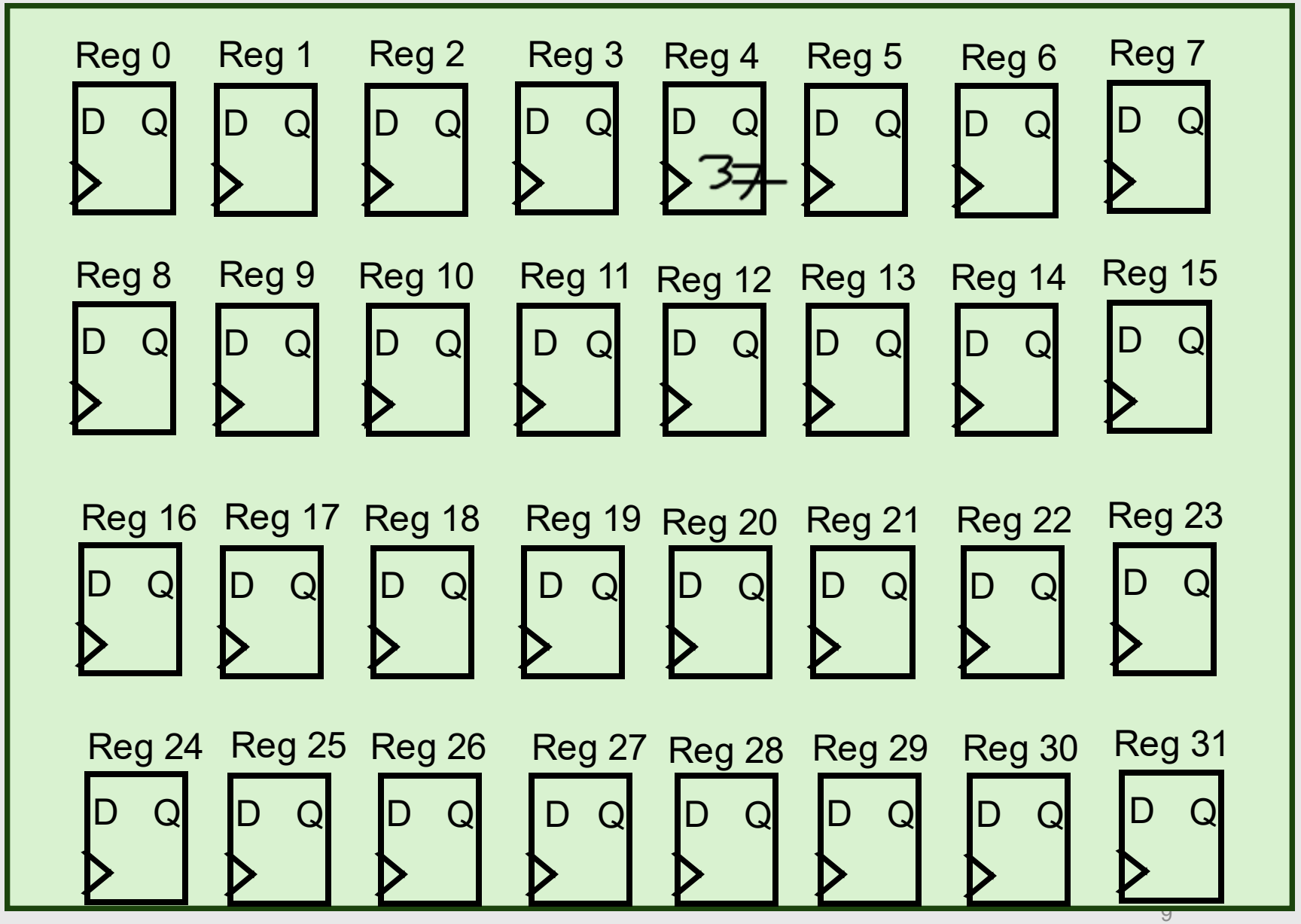
R-Format Syntax

```
add x5, x4, x3
```

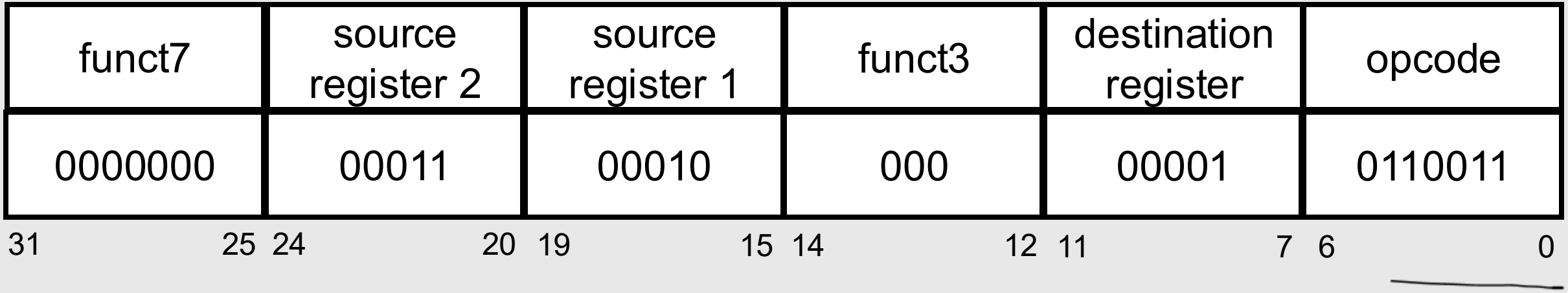


funct7	source register 2	source register 1	funct3	destination register	opcode
0000000	00011	00100	000	00101	0110011
31 25	24 20	19 15	14 12	11 7	6 0

Register File



r1 = r2 + r3



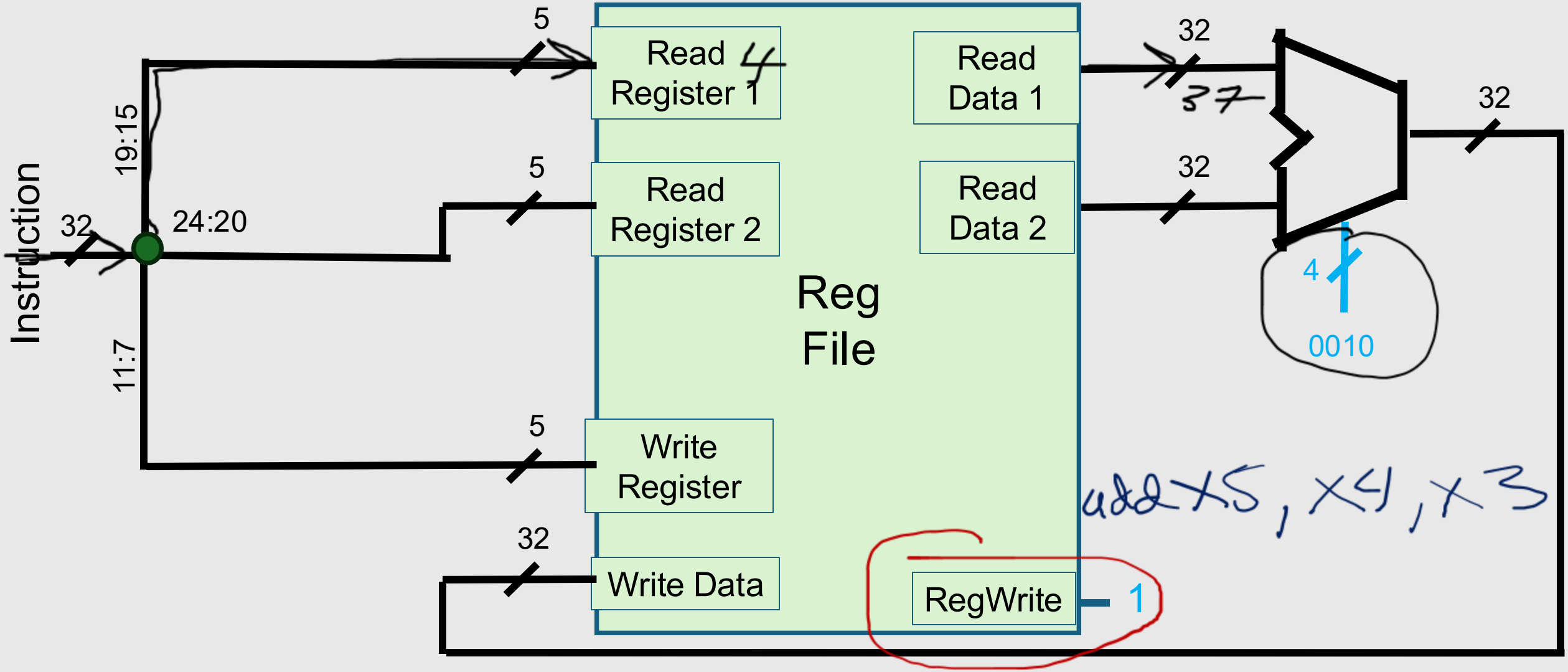
00000000001100010000000010110011

Opcode is always 0110011 for integer R-type ALU instructions

funct3 is always the primary operation code group (e.g. 000 = add/sub. 100 = xor, 111 = and)

funct7 is used to further specify the operation (e.g. distinguishes add vs. sub. srl vs. sra)

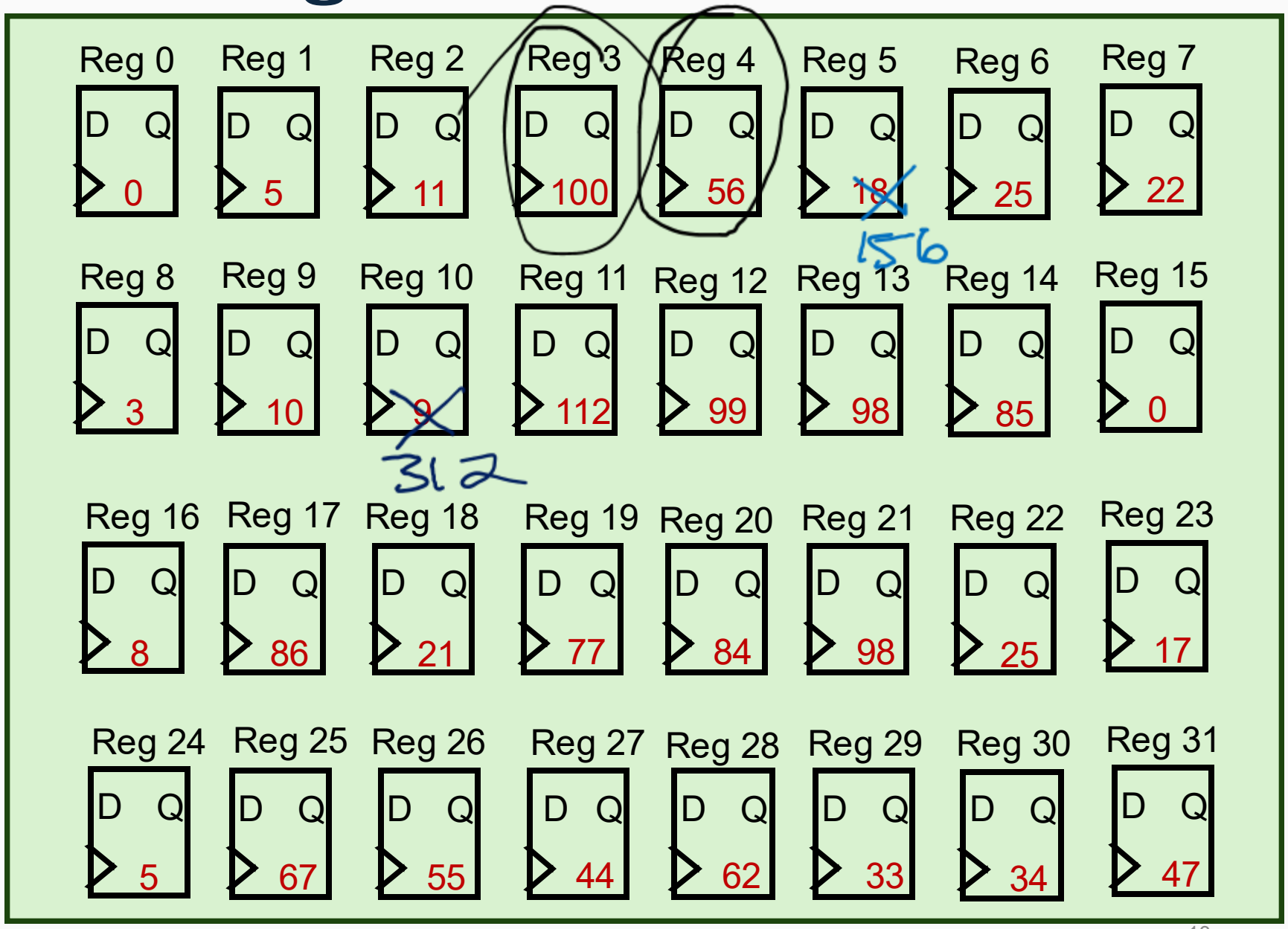
funct7	rs2	rs1	funct3	rd	opcode
0000000	00011	00010	000	00001	0110011
31 25	24 20	19 15	14 12	11 7	6 0



What are the contents of the register file after the following instructions are executed?

```
0x003202b3
0x00528533
0x00EA6020
```

Contents of RegFile



1 hex - 4 bits

What are the contents of the register file after the following instructions are executed?

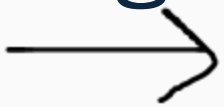
0x003202b3

B ⇒ 1011

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

0000 0000 0011 0010 0000 0010 1011 0011

What are the contents of the register file after the following instructions are executed?



`0x003202b3`

`0b00000000001100100000000101010110011`

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

x5 = x4 + x3

x5 = 56 + 100

x5 = 156

add x5, x4, x3

funct7 + funct3 all zero ⟹ add

What are the contents of the register file after the following instructions are executed?

0x00528533

funct7	source register 2	source register 1	funct3	destination register	opcode
0	00101	00101	000	01010	0110011
31 — 25	24 — 20	19 — 15	14 — 12	11 — 7	6 — 0

x5 x5 x10

0000000 01010 00101 000 01010 0110011

add x10, x5, x5

What are the contents of the register file after the following instructions are executed?

0x00528533

0b0000000 00101 00101 000 01010 0110011

funct7	source register 2	source register 1	funct3	destination register	opcode
0000000	00101	00101	000	01010	0110011
31 25	24 20	19 15	14 12	11 7	6 0

x10 = x5 + x5

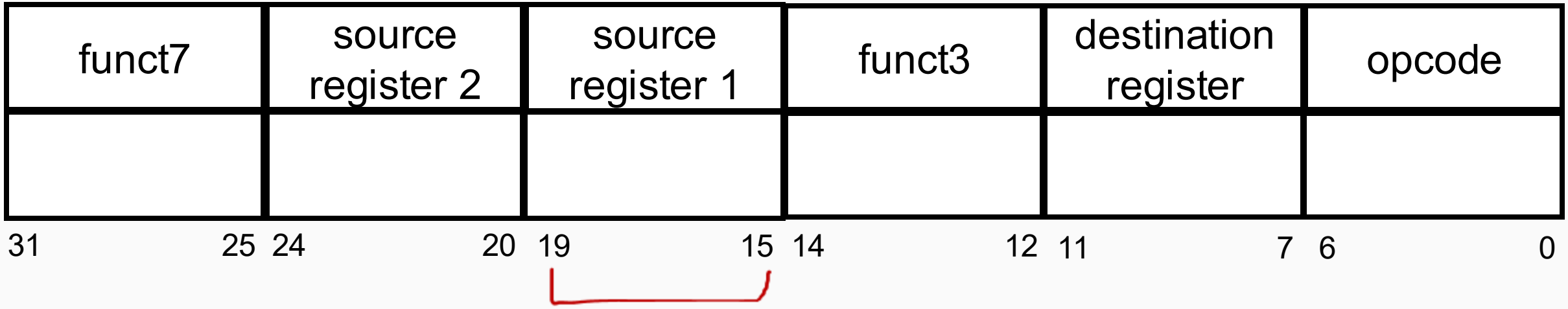
x10 = 156 + 156

x10 = 312

add x10, x5, x5

What are the contents of the register file after the following instructions are executed?

`0x00a38633`



What are the contents of the register file after the following instructions are executed?

→ **0x00a38633**

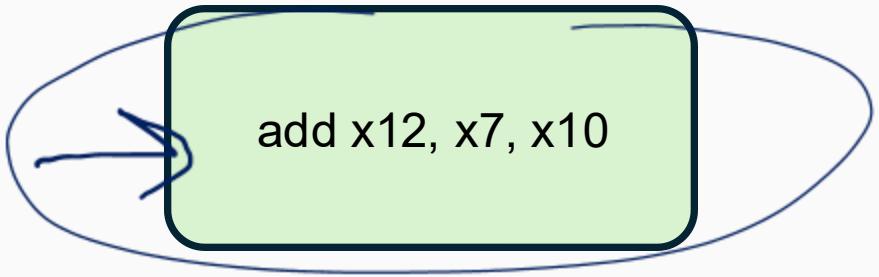
0b000000001010**00111**000**01100**0110011

funct7	source register 2	source register 1	funct3	destination register	opcode
0000000	01010	00111	000	01100	0110011
31 25	24 20	19 15	14 12	11 7	6 0

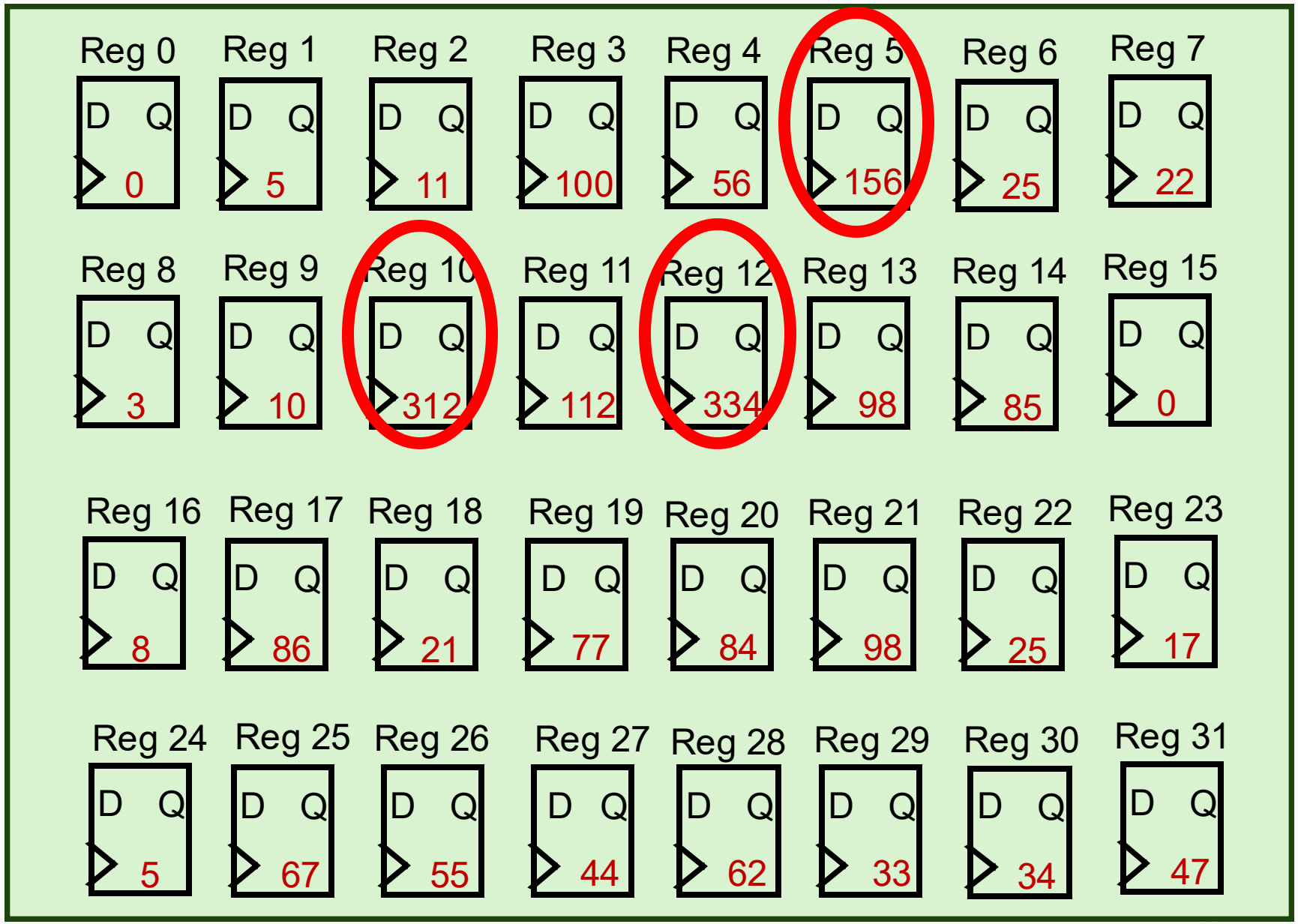
x12 = x7 + x10

x12 = 22 + 312

x12 = 334



Final Contents of RegFile



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)

Subtract Instruction

sub rd, rs1, rs2

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

value of the register rd

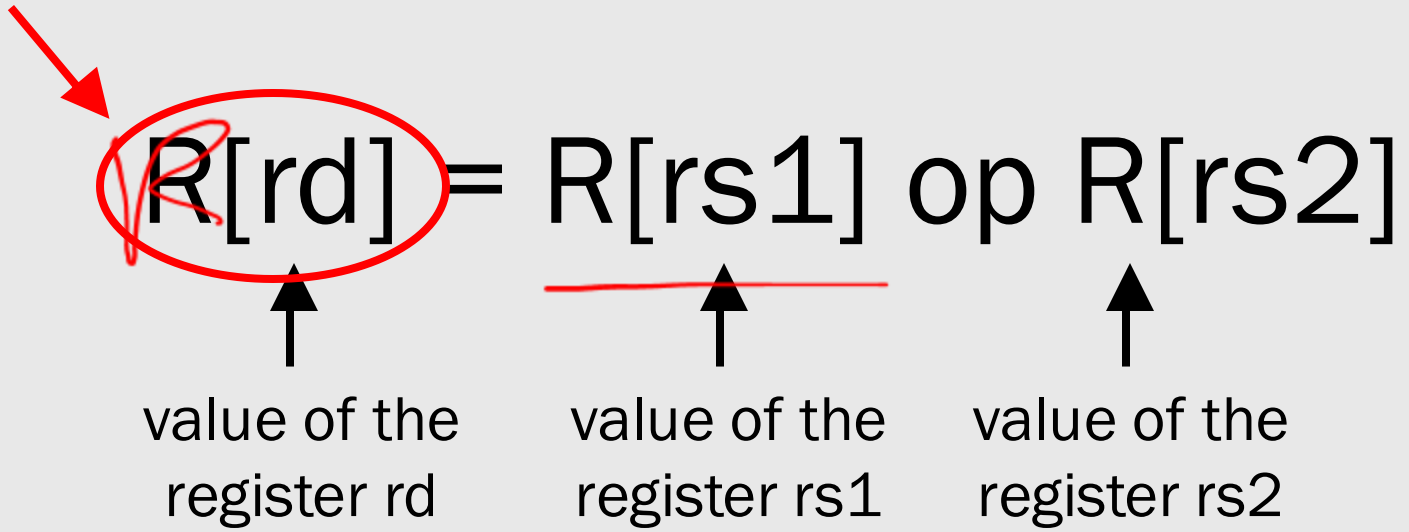
value of the register rs1

value of the register rs2

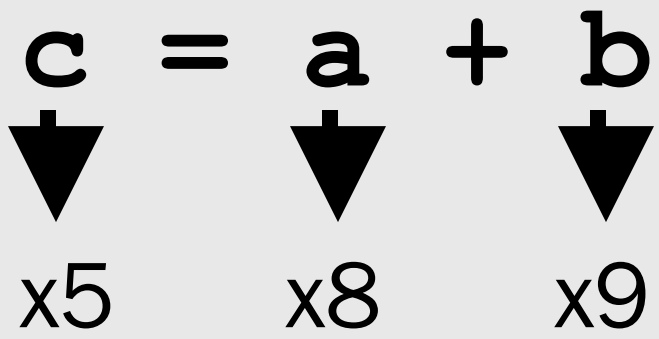
R-format Syntax

op rd, rs1, rs2

This notation means RegisterFile[rd]
(i.e. the contents of register rd)



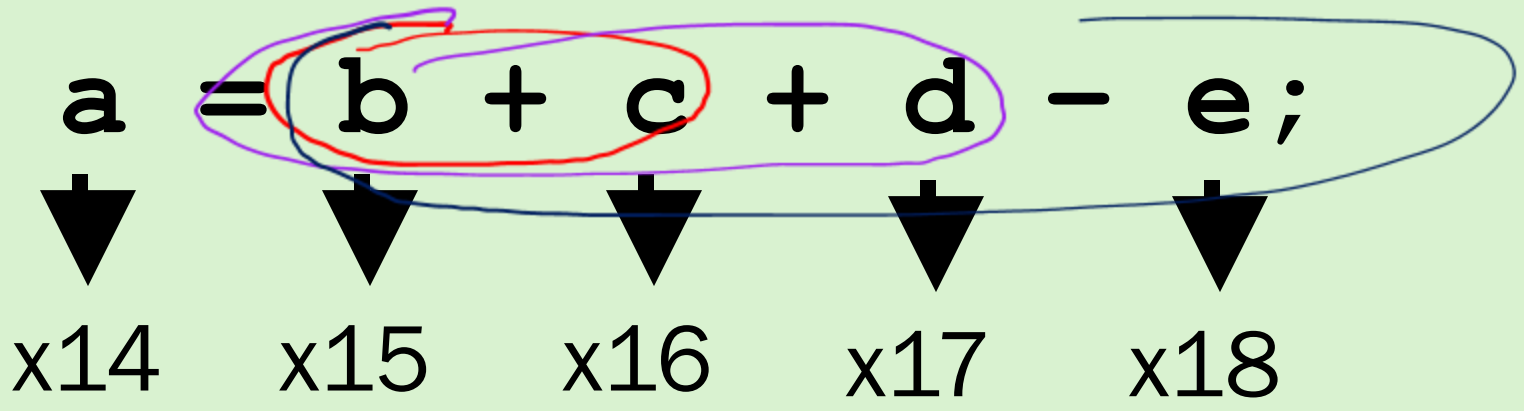
Translating from C to RISC-V




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

Translate this line from C to RISC-V

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



You may use additional registers in your solution

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

x15 := x15 + x16

x15 := x15 + 17

x14 := x15 - x18

Translate this line from C to RISC-V

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

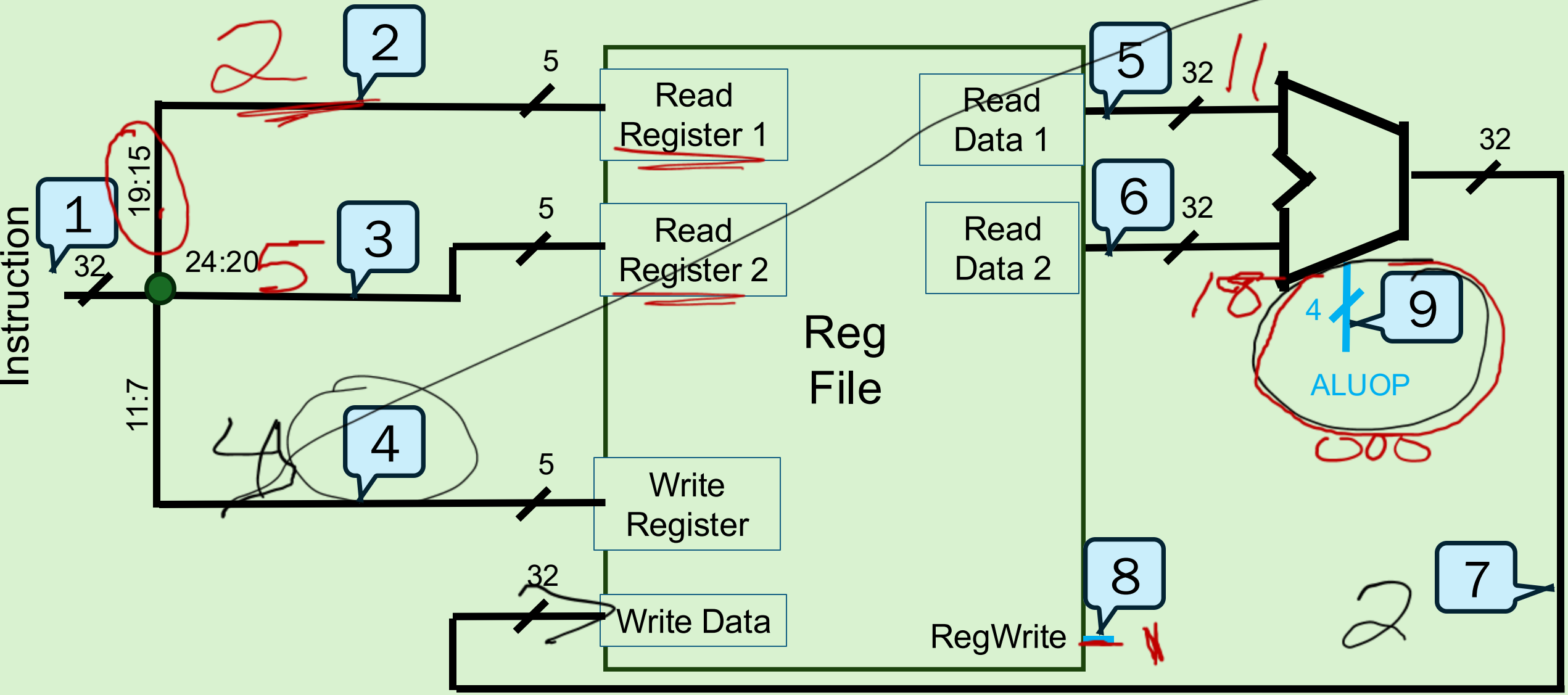
x14 x15 x16 x17 x18

```
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
```

What is the value of each signal when executing: **`and x4 x2 x5`**



WS Soln

and x4, x2, x5

1. 00000000 00101000 10 111 01000 1110011

2.

7. 11 & 18

⟹ 01011

& 10010

⟹ 00010 ⟹ 2

Original Contents of RegFile