

10/21/26

COMP 311:

Computer Organization!

Lecture 17: Building the ALU



The University
of North Carolina
at Chapel Hill

<https://tinyurl.com/comp311-fa26>

Logistics Update



- Lab 2 due tomorrow!

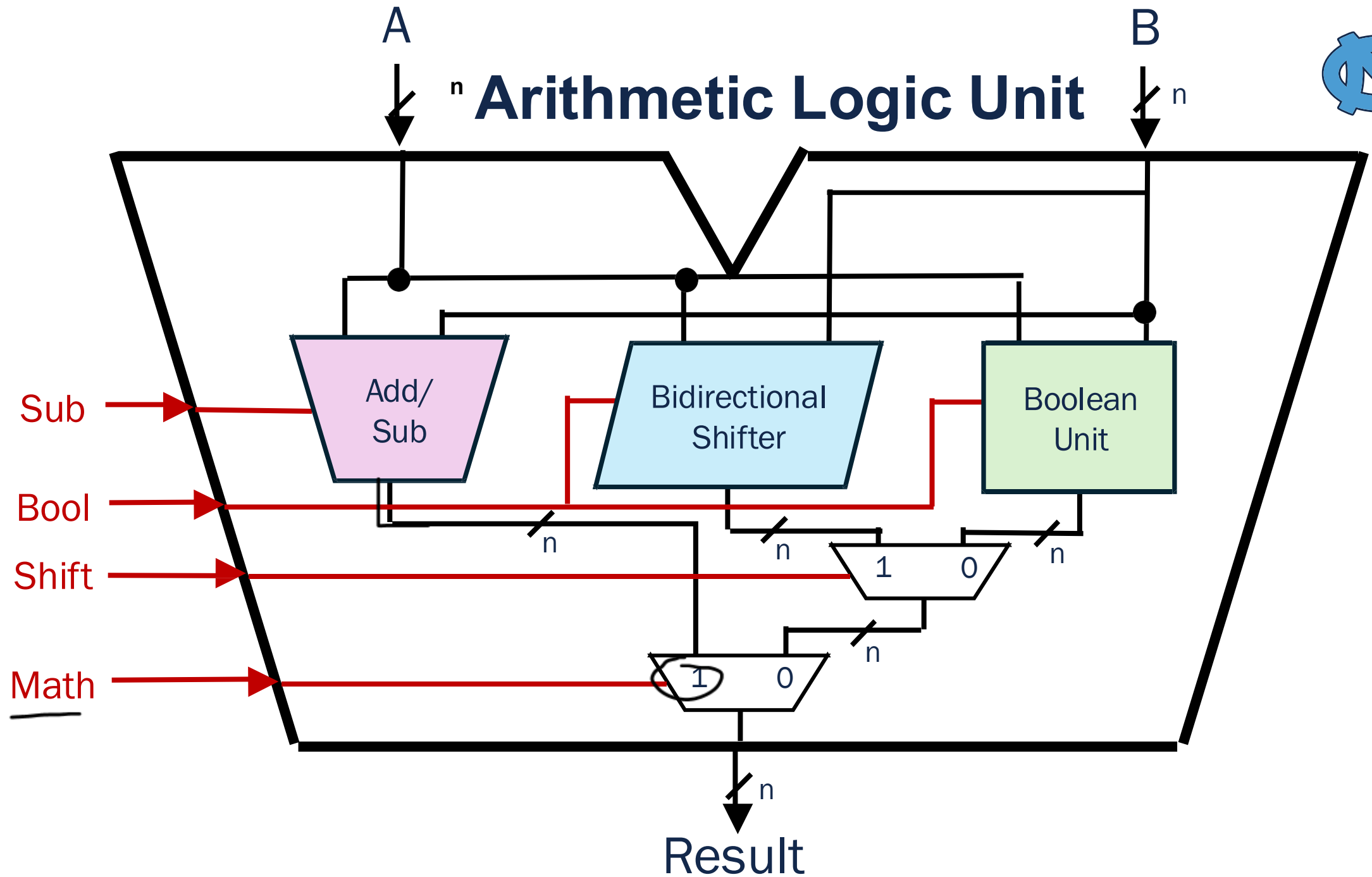
Plan for Class Today



- 4:15-4:50: ALU + Register File
- If time, intro to timing analysis

Guiding Question ☺

How do the ALU and
Register File work
together?



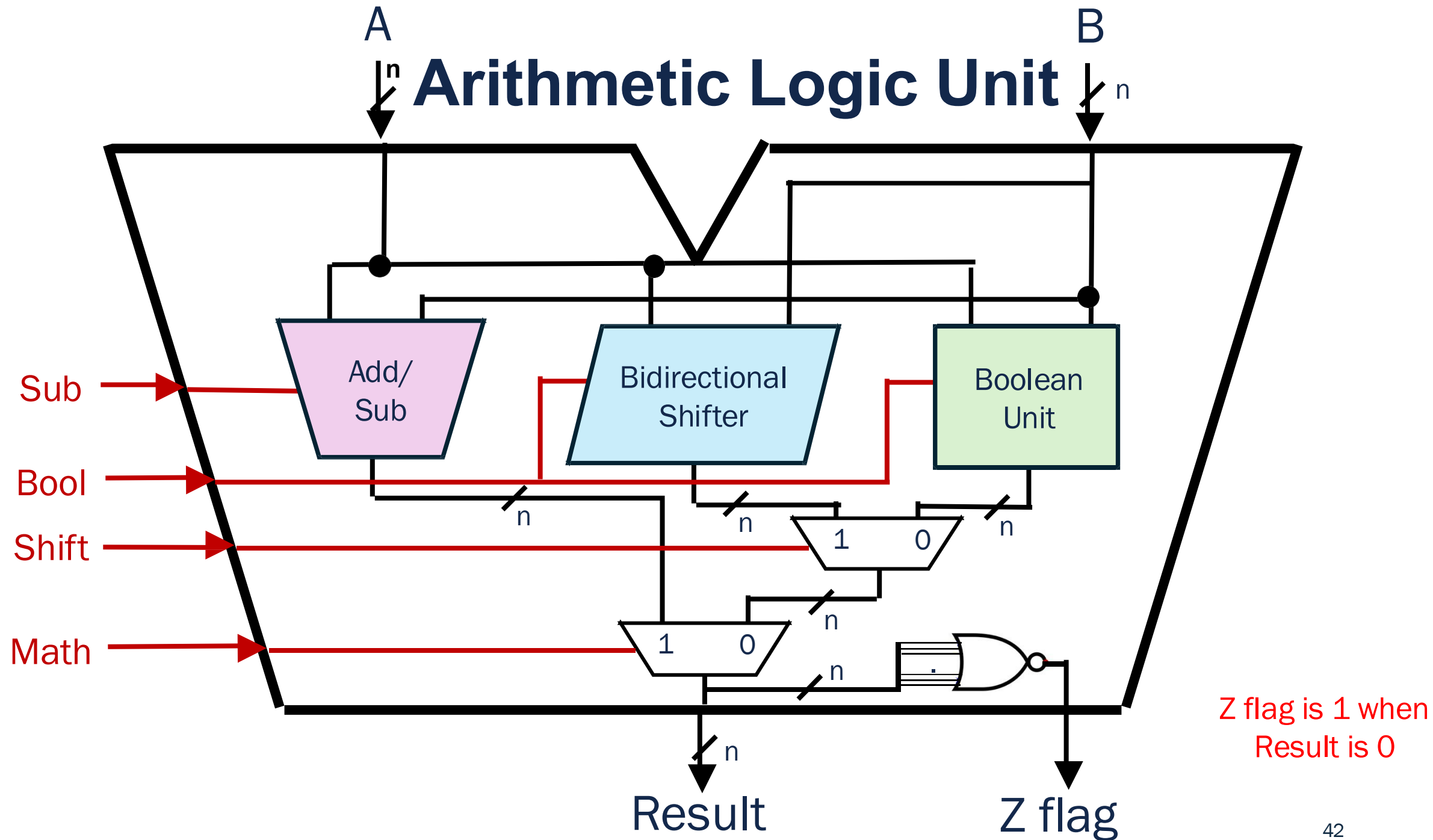
ALU Control Bits

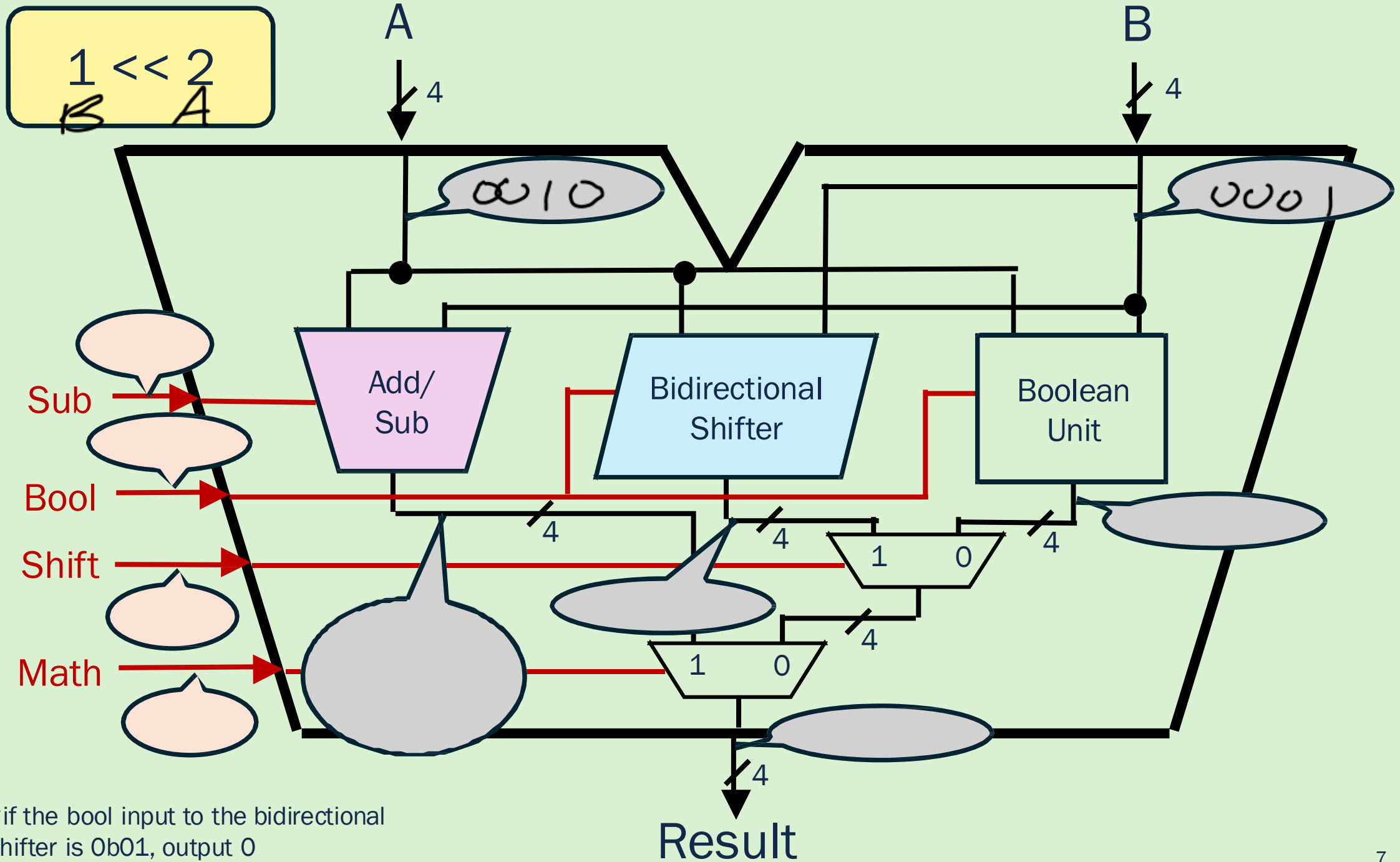
Operation	Expression	Sub
Add	$A + B$	0
Sub	$A - B$	1

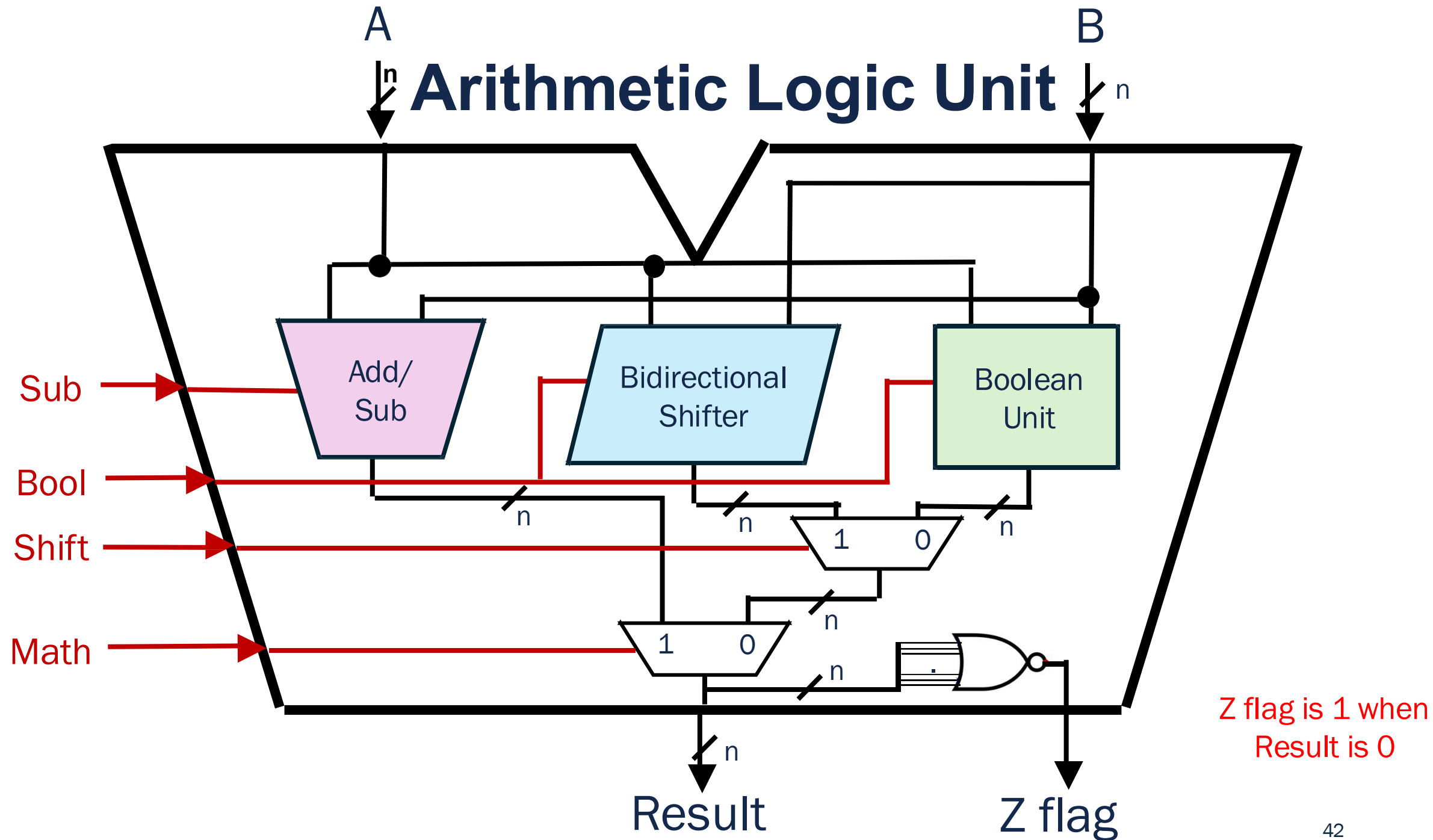
Operation	Expression	Bool
Left Shift	$B \ll A$	00
Logical Right Shift	$B \gg A$	10
Arithmetic Right Shift	$B \ggg A$	11

Operation	Expression	Bool
AND	$A \times B$	00
OR	$A + B$	01
XOR	$A \oplus B$	10
NOR	$\overline{A + B}$	11

Operation	Sub	Bool	Shift	Math
$A + B$	0b0	0bXX	0bX	0b1
$A - B$	0b1	0bXX	0bX	0b1
$B \ll A$	0bX	0b00	0b1	0b0
$B \gg A$	0bX	0b10	0b1	0b0
$B \ggg A$	0bX	0b11	0b1	0b0
$A \text{ AND } B$	0bX	0b00	0b0	0b0
$A \text{ OR } B$	0bX	0b01	0b0	0b0
$A \text{ XOR } B$	0bX	0b10	0b0	0b0
$A \text{ NOR } B$	0bX	0b11	0b0	0b0







Some Takeaways

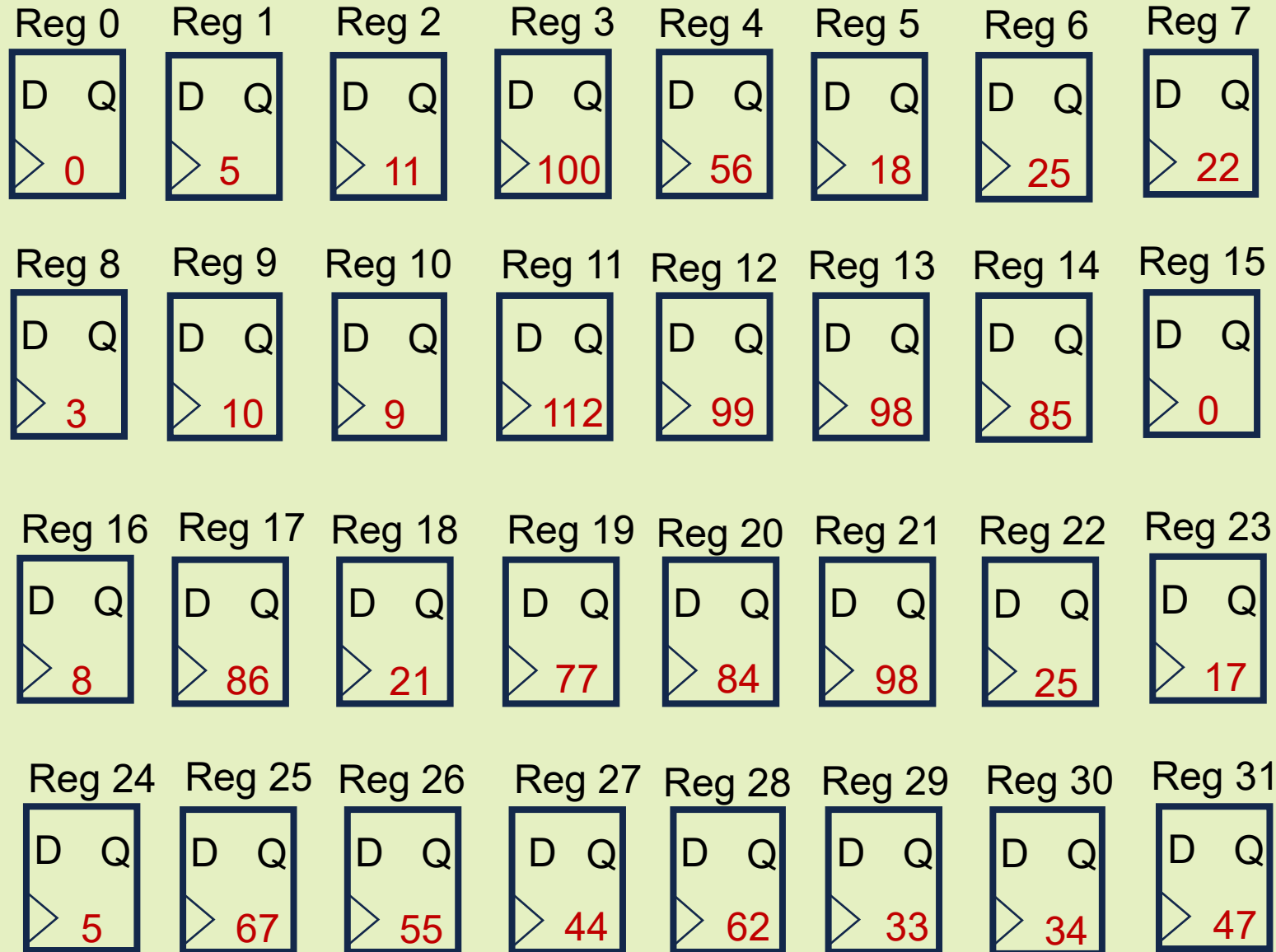


- This is ***one possible*** ALU implementation based on MIPS-style instruction set
- The building blocks (addition, subtraction, shifting, Boolean operations, and multiplexers) are also used in RISC-V processors
 - You will build in Lab 3!! 😊
- Bits in the control table are implementation choices, not specified directly by the ISA
- **The ISA specifies what an instruction does. The ALU design determines how the hardware implements it.**



Some Register File Details

Register File

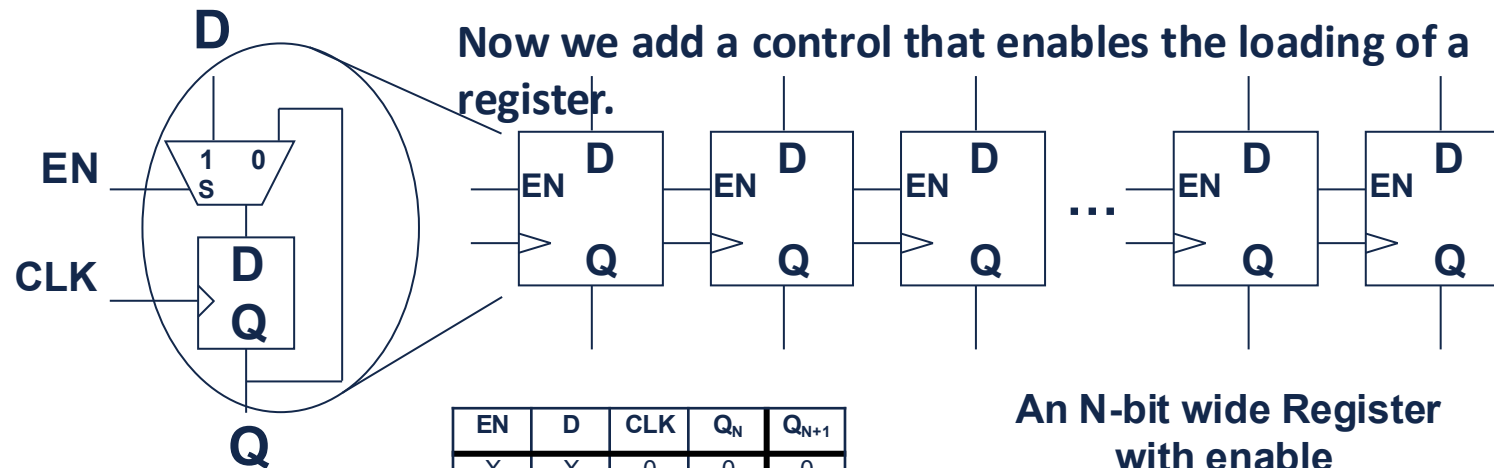


- Contains the register state of the computer
- Any register can be read or written, just specify the number of the register

Implementation Details on the Register File!

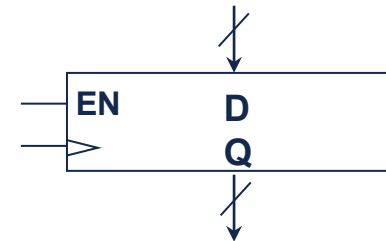


Thus far, our building blocks units have focused on logical and arithmetic functions. We'll also need functional units for storing intermediate results. By now, we are used to the notion of building wide registers.



EN	D	CLK	Q_N	Q_{N+1}
X	X	0	0	0
X	X	0	1	1
X	X	1	0	0
X	X	1	1	1
0	X	↑	0	0
0	X	↑	1	1
1	0	↑	X	0
1	1	↑	X	1

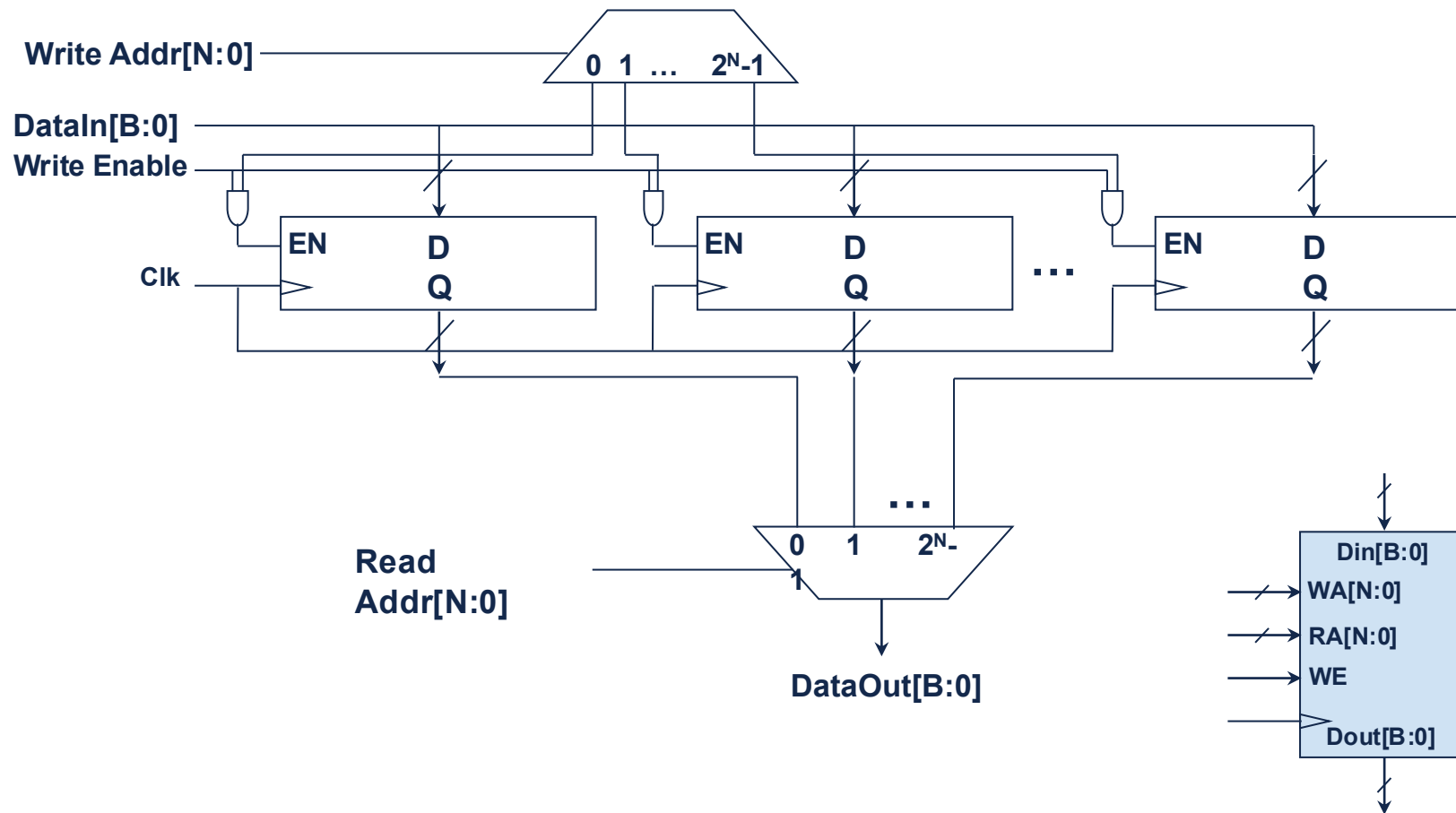
An N-bit wide Register with enable



A Register File



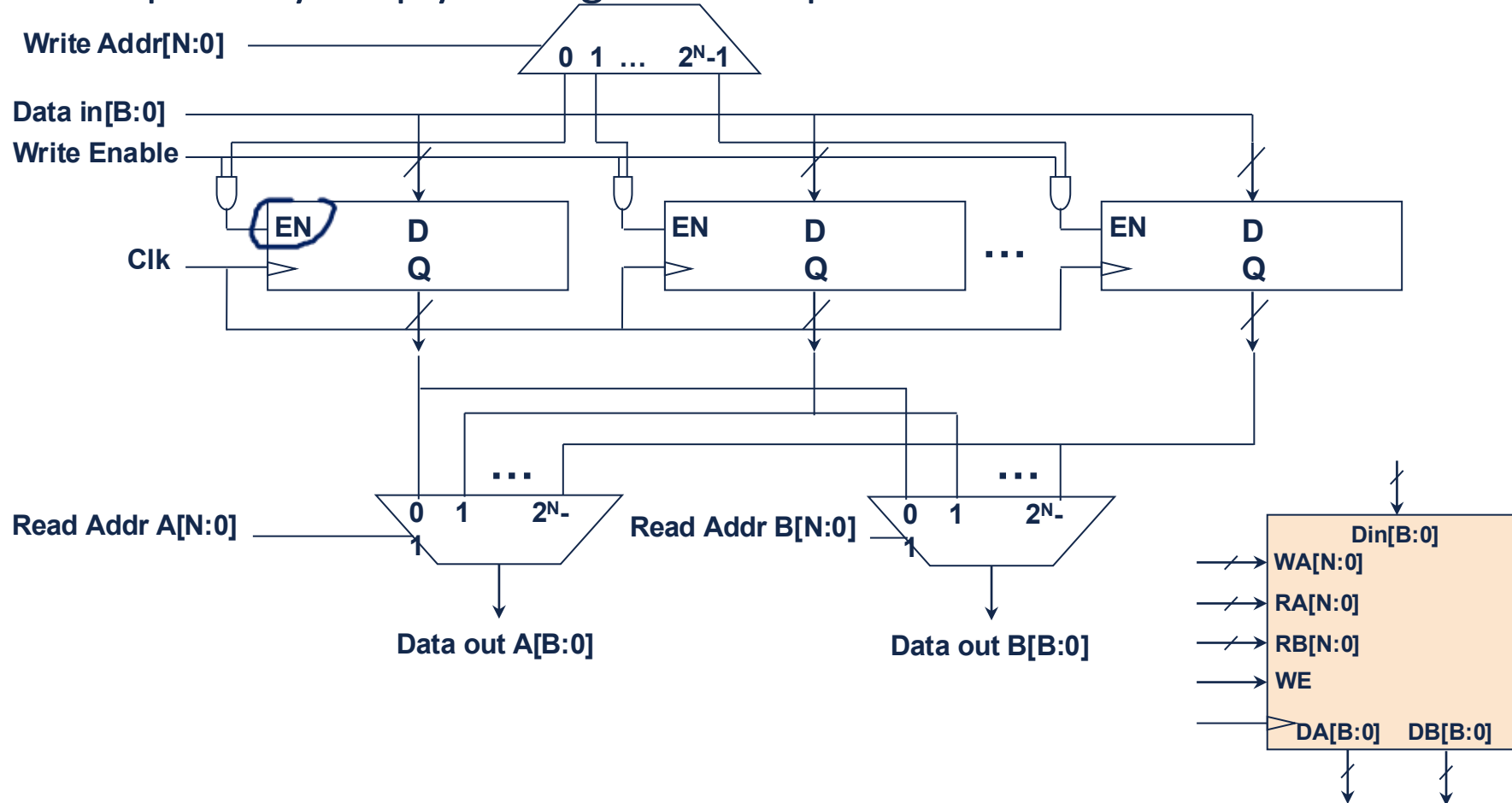
- We can also construct an addressable array of registers



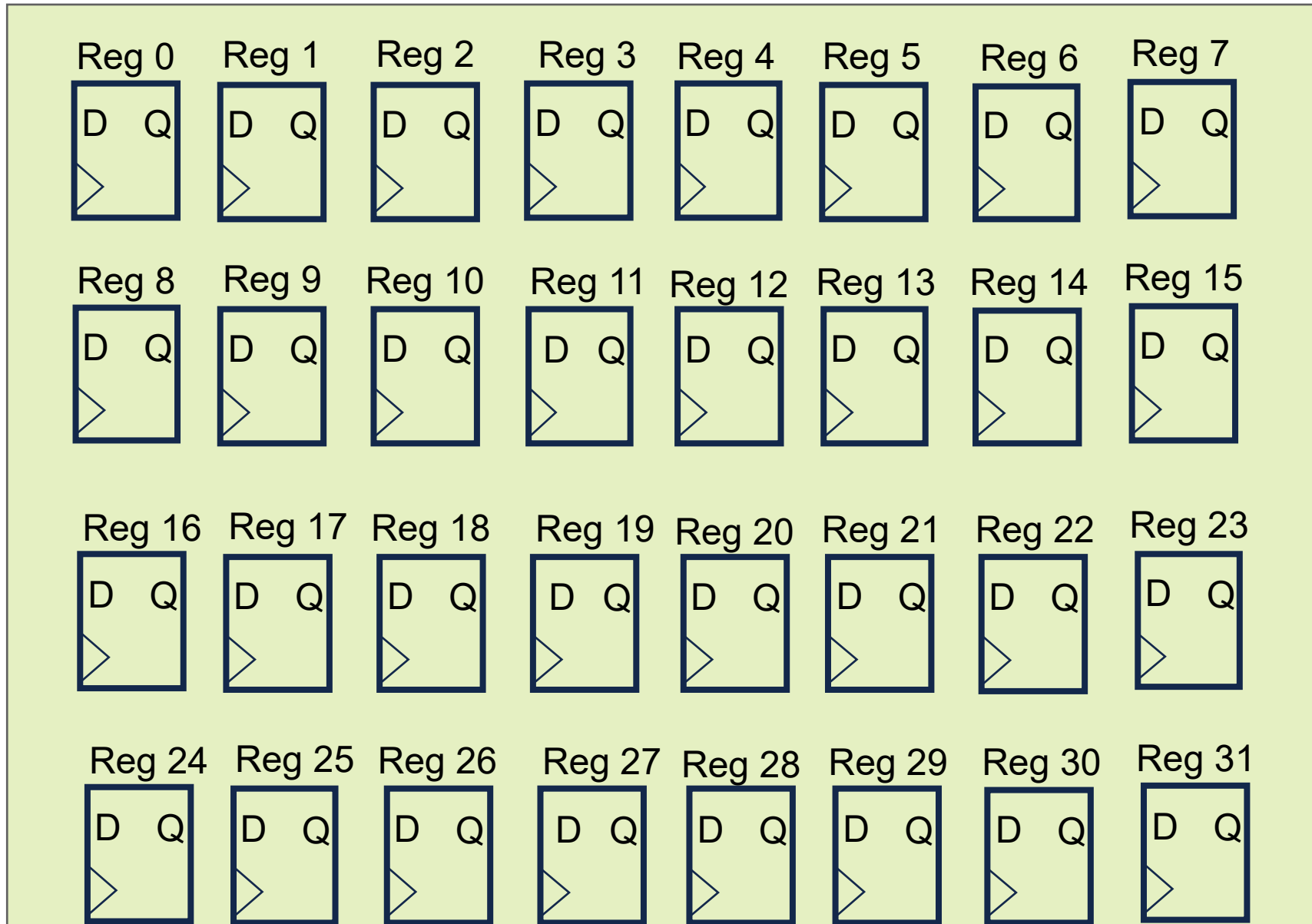
A Multi-Ported Register File



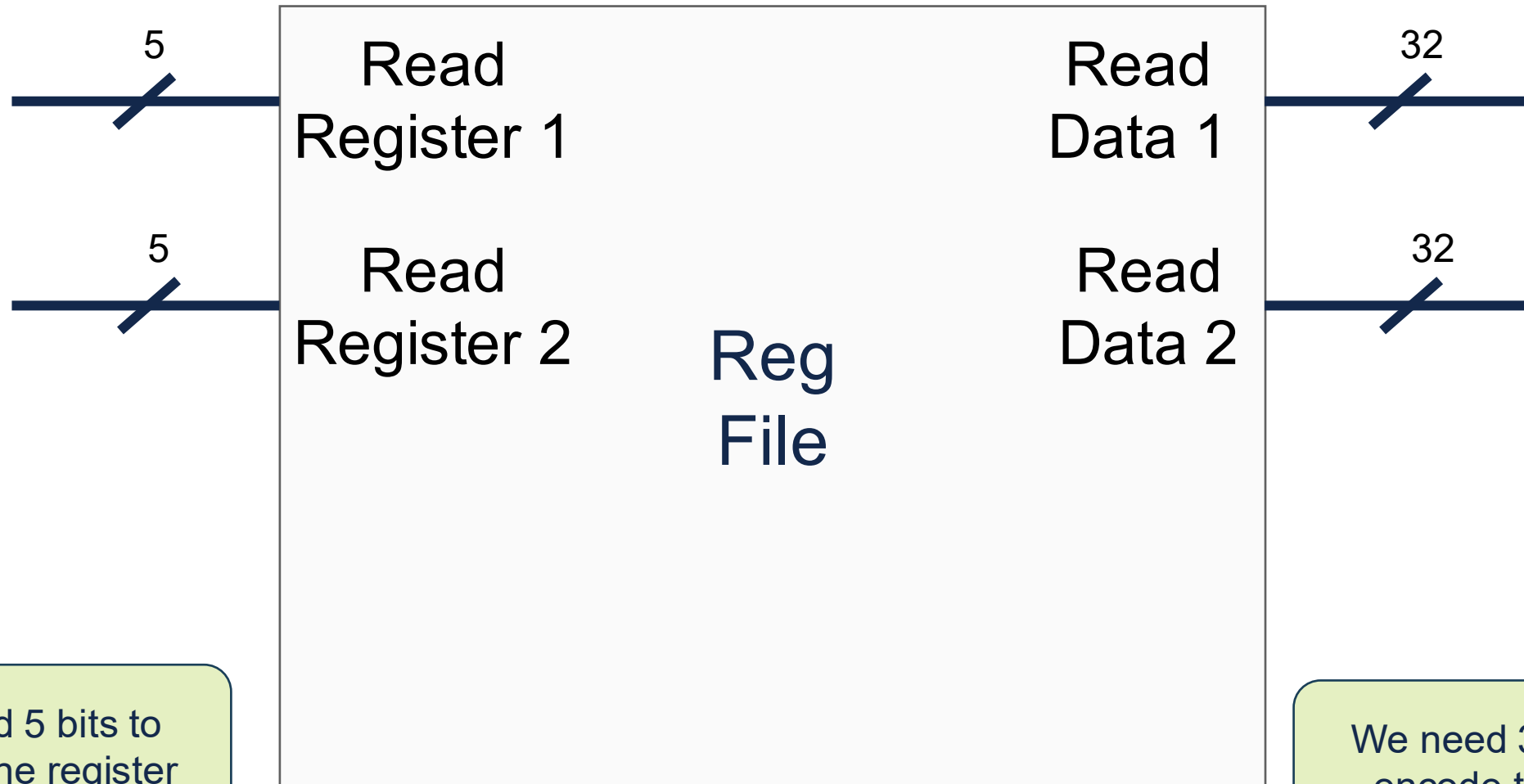
- Multiple read ports by simply adding more output MUXs



Register File



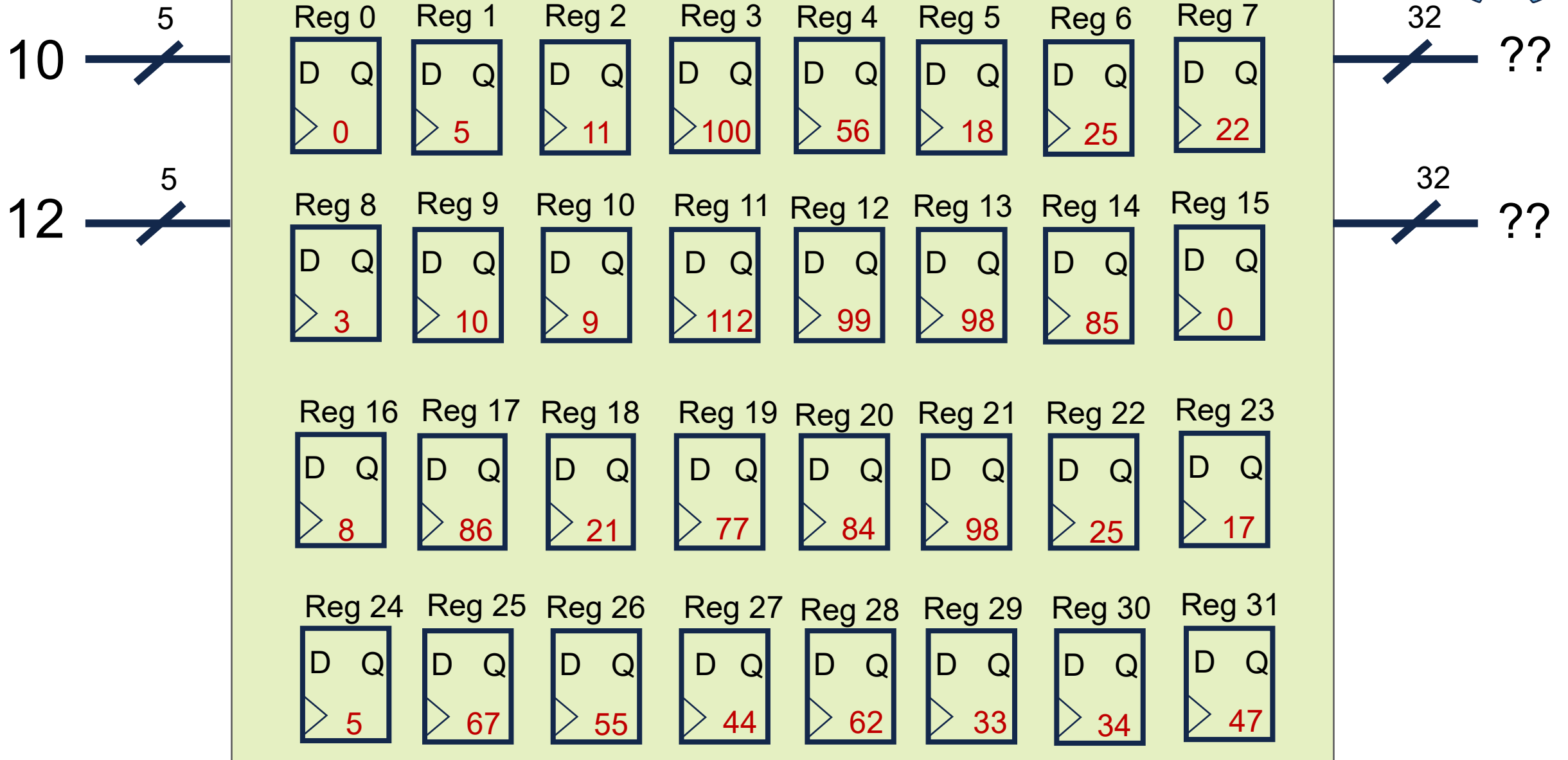
Reading from the Register File



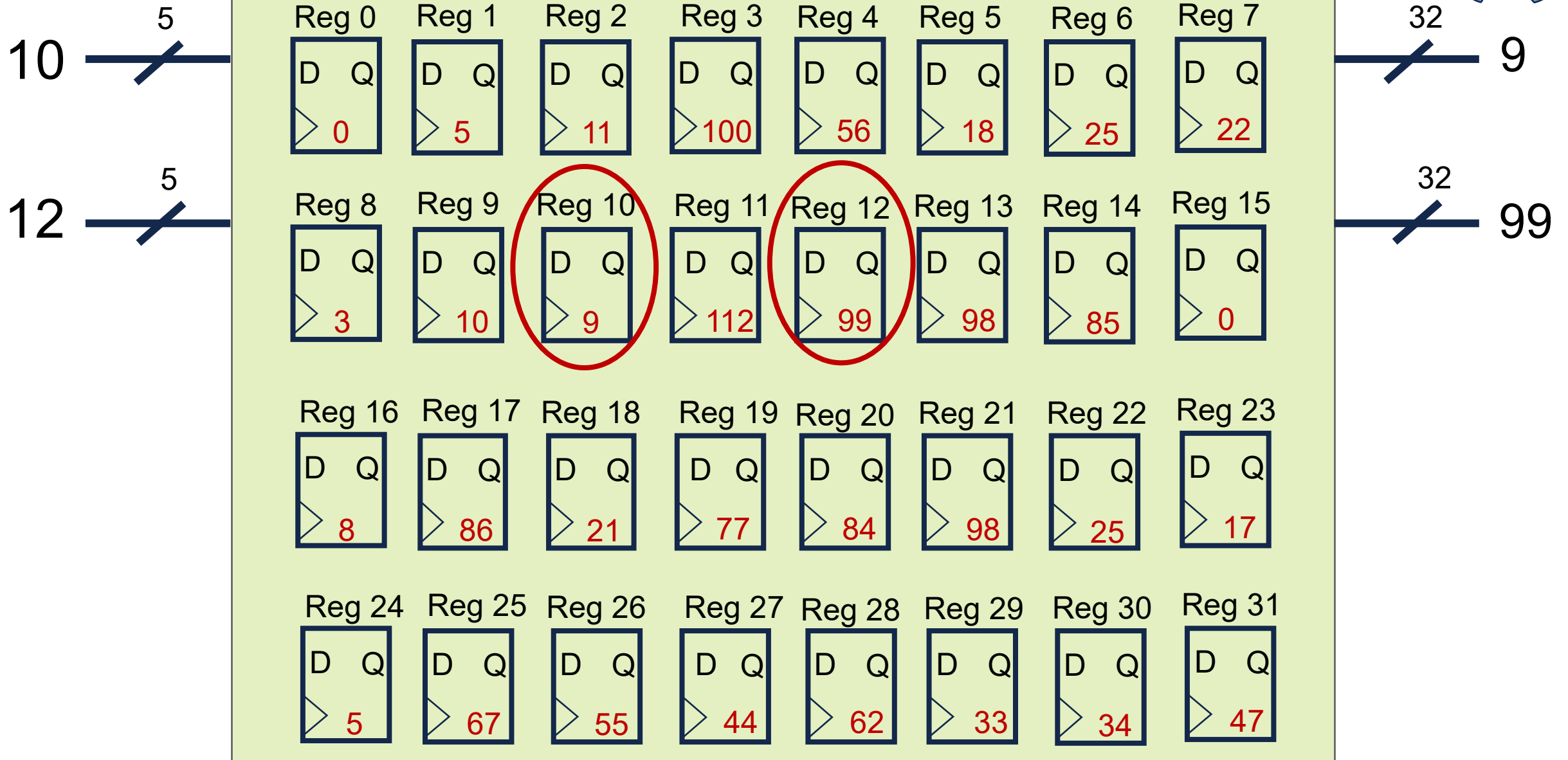
We need 5 bits to encode the register

We need 32 bits to encode the data

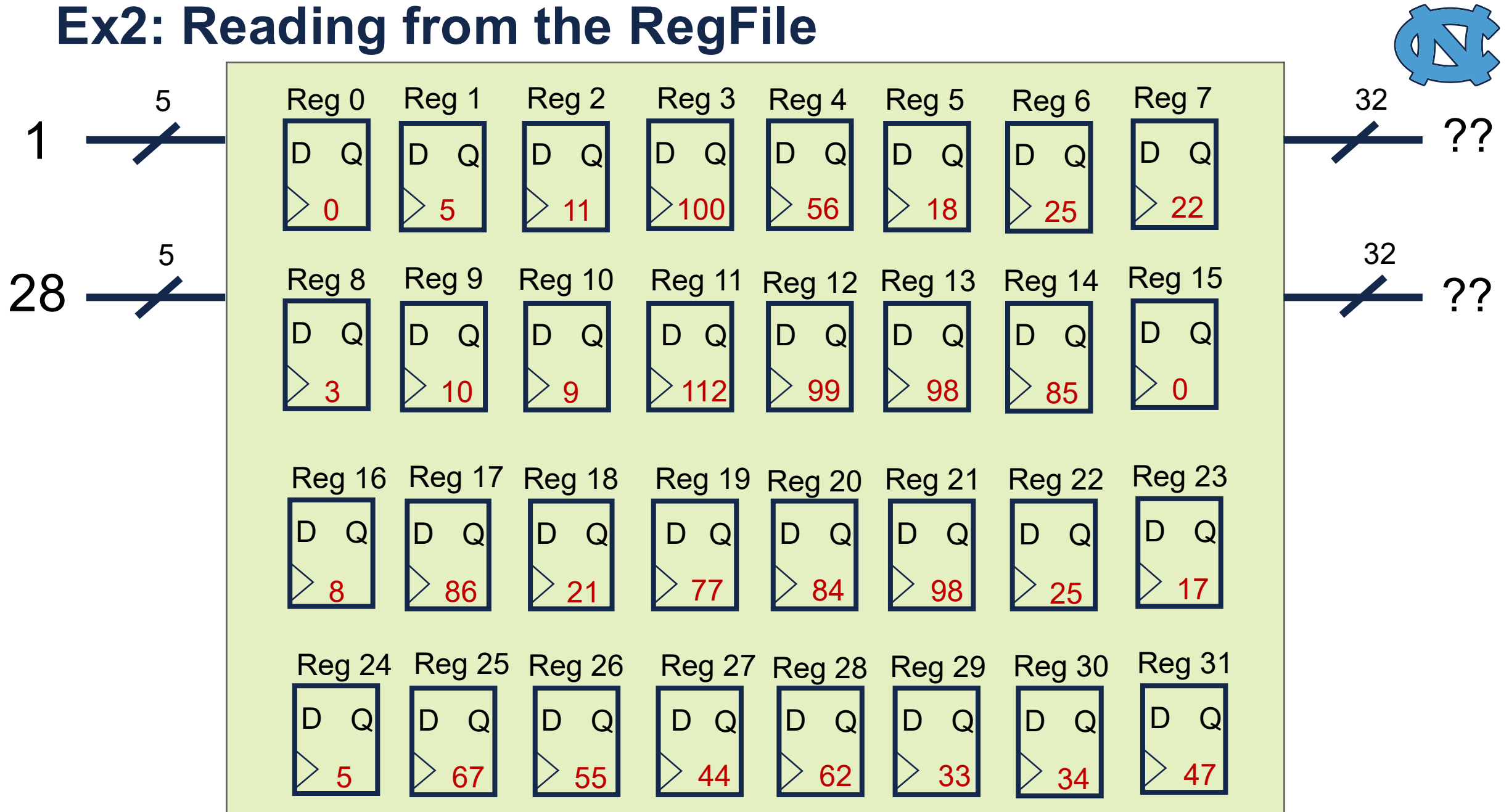
Ex1: Reading from the RegFile



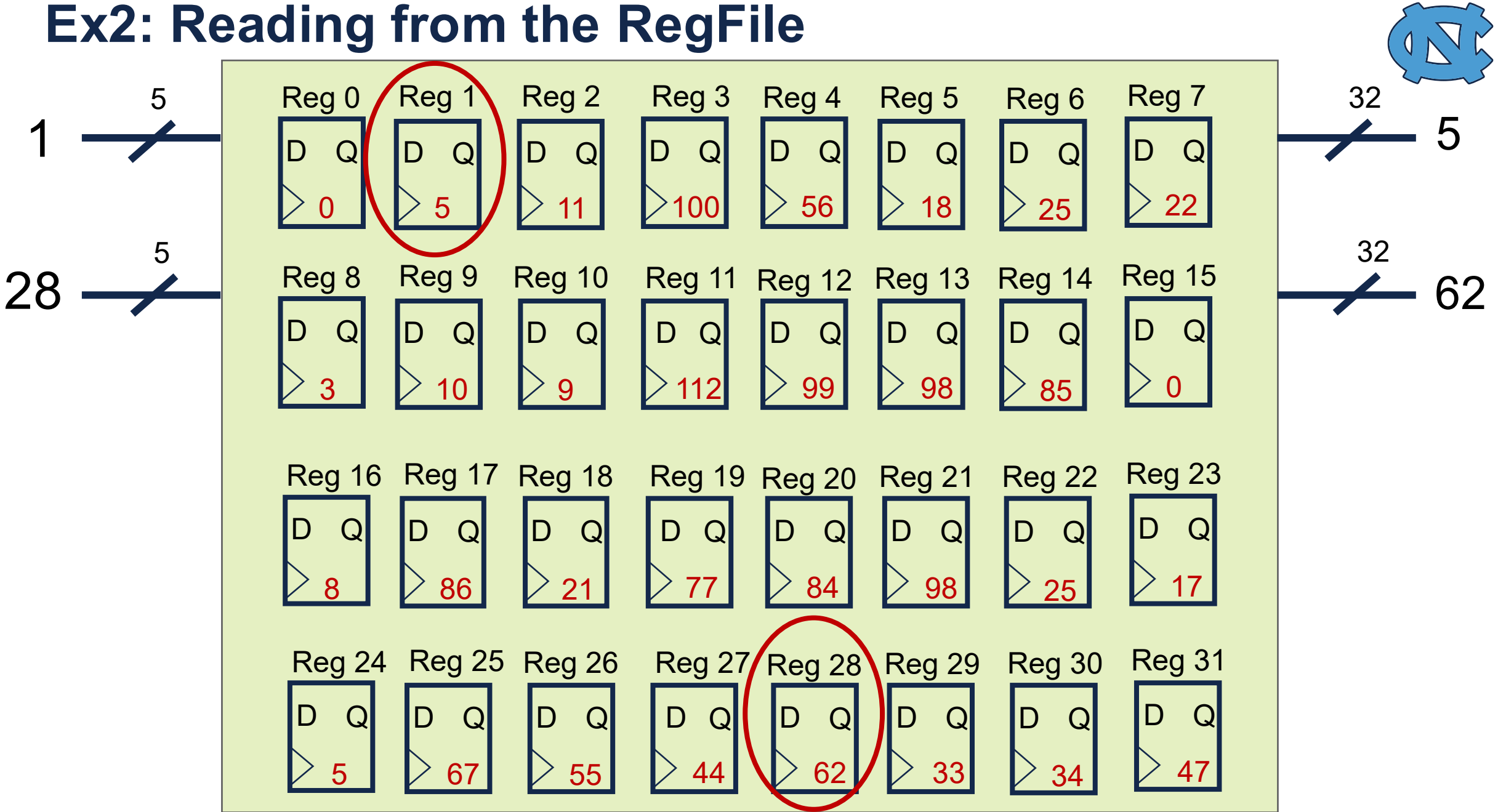
Ex1: Reading from the RegFile



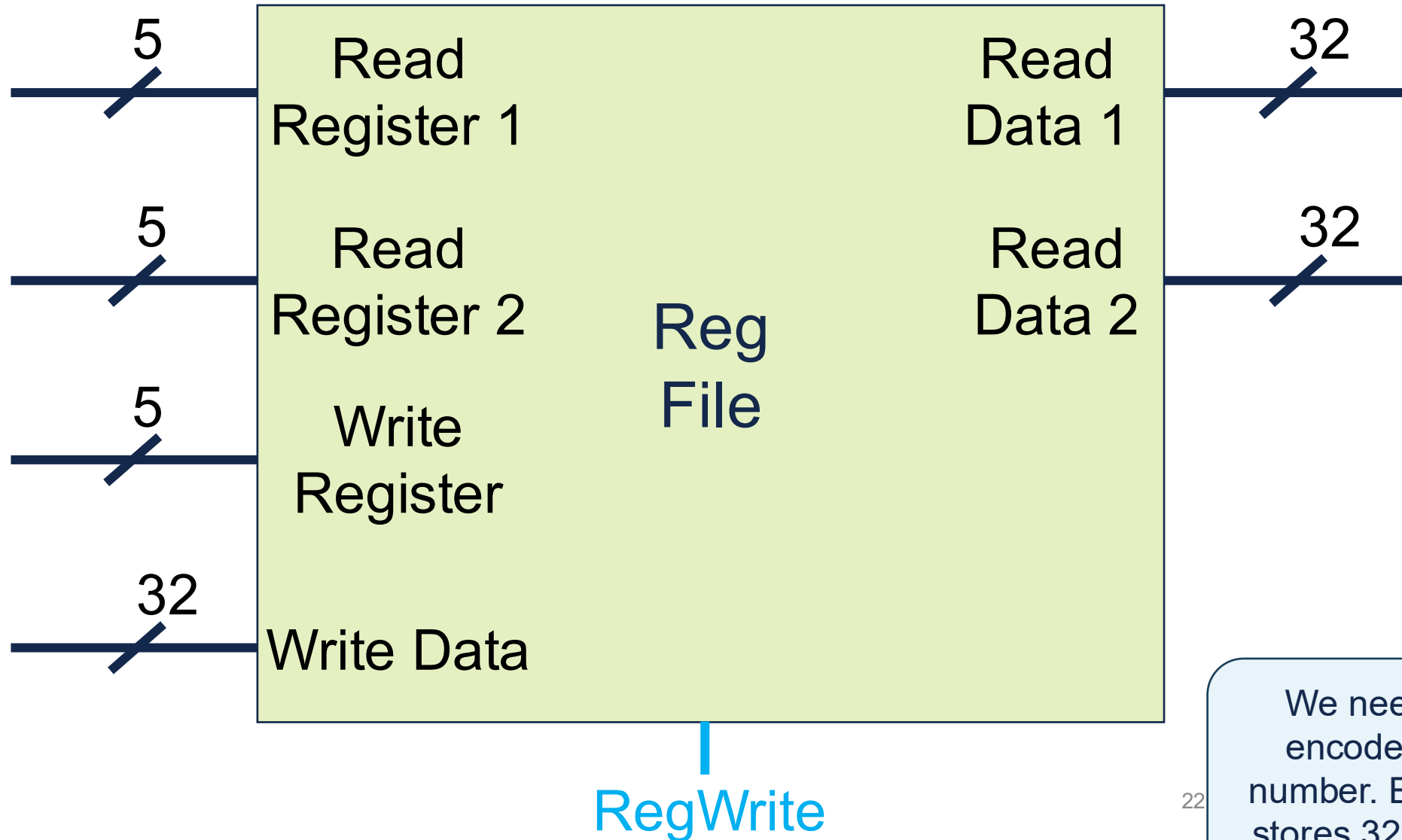
Ex2: Reading from the RegFile



Ex2: Reading from the RegFile



Writing to the Register File



22

We need 5 bits to encode a register number. Each register stores 32 bits of data!

Writing to the Register File



- To write to a register inside of the register file, set the following signals
 - **Write register**: The number of the register you are writing to
 - **Write data**: The data you want to store inside the register
 - **RegWrite**: This is a control signal whose value needs to be 1 to write to a register
 - When we are not writing to a register, this value should be 0



Piecing Things Together: The ALU + Register File

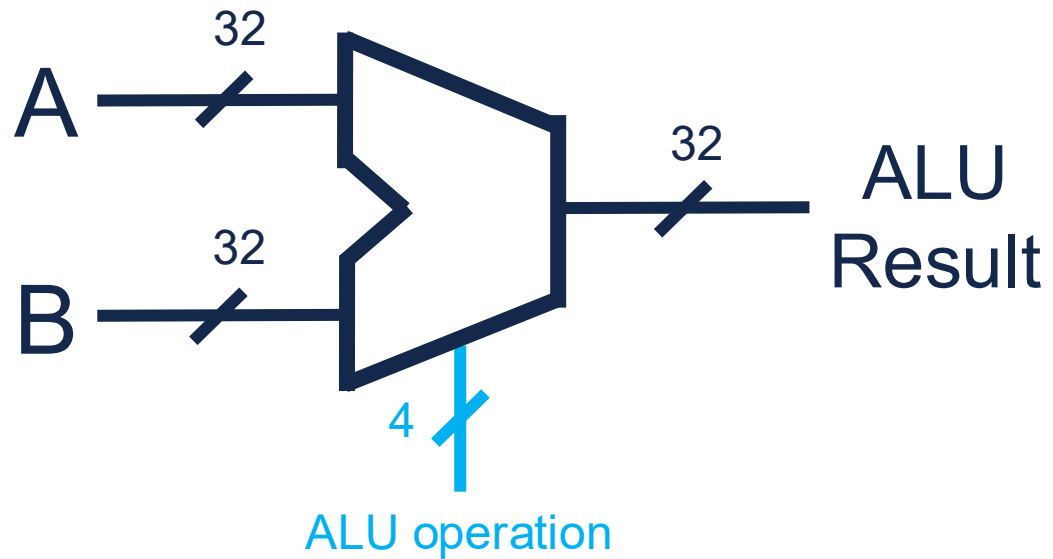
$$x1 = x3 + x2$$

funct7	source register 2	source register 1	funct3	destination register	opcode
0000000	00011	00010	000	00001	0110011
31	25 24	20 19	15 14	12 11	7 6
					0

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)

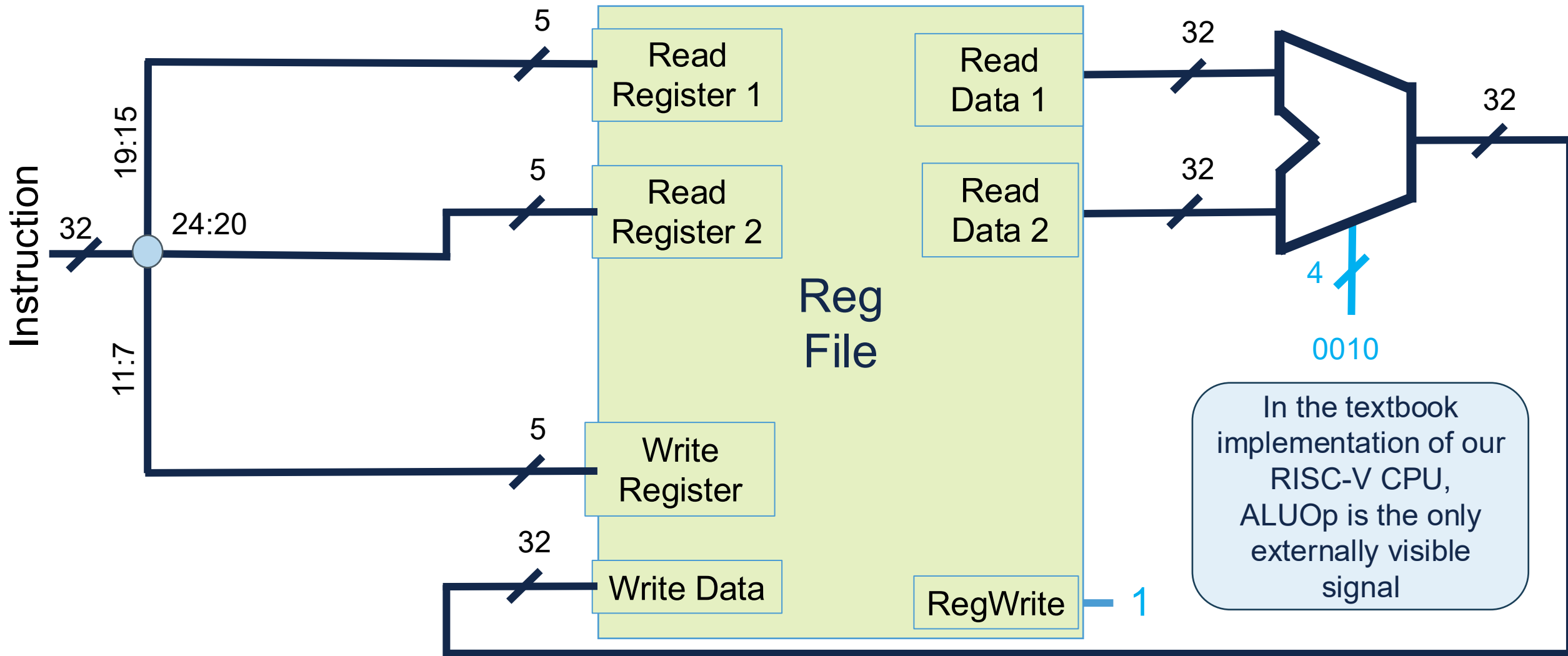
RISC-V 32I ALU



Function	ALU Operation
AND	0b0000
OR	0b0001
ADD	0b0010
SUB	0b0110
set on less than (SLT)	0b0111

Control Signal Input

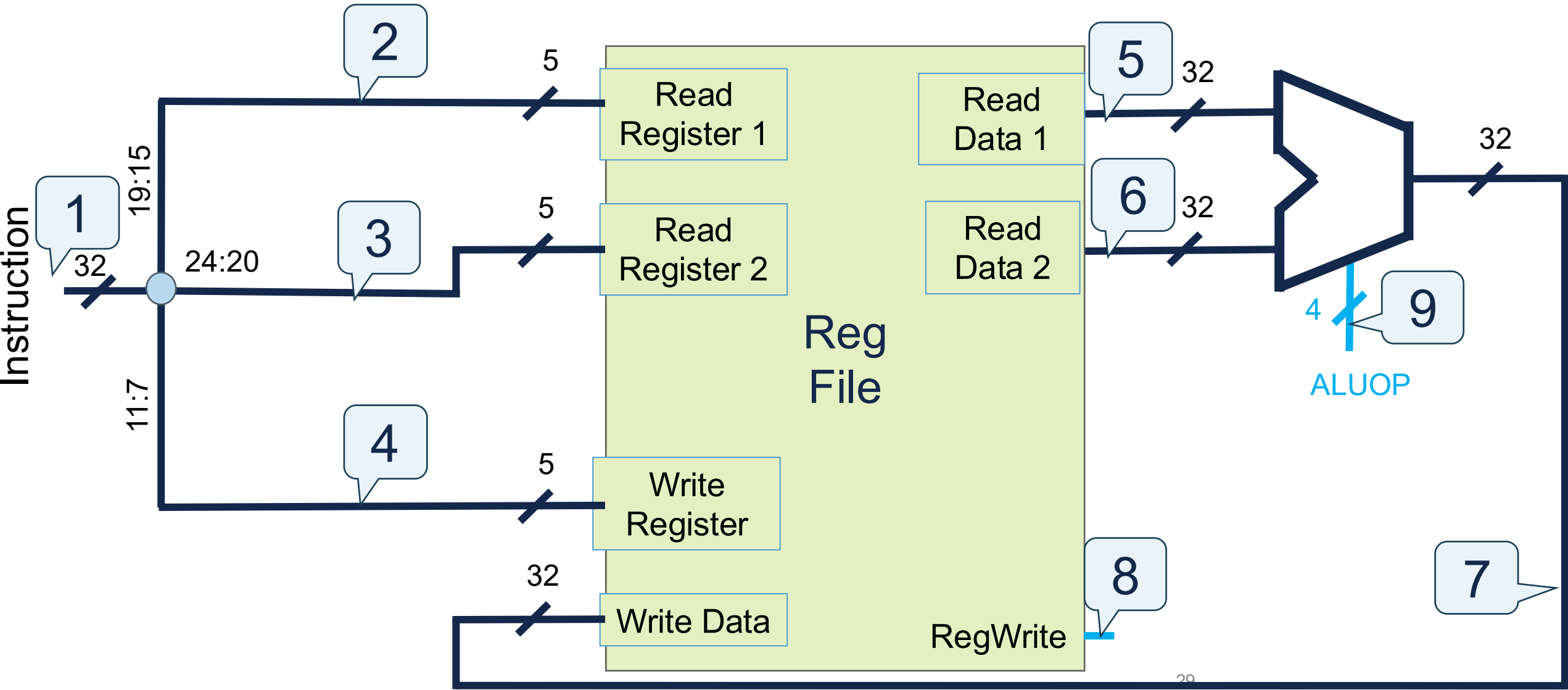
funct7	rs2	rs1	funct3	rd	opcode
0000000	00011	00010	000	00001	0110011
31	25 24	20 19	15 14	12 11	7 6
					0



Contents of RegFile

Reg 0	Reg 1	Reg 2	Reg 3	Reg 4	Reg 5	Reg 6	Reg 7
D Q > 0	D Q > 5	D Q > 11	D Q > 100	D Q > 56	D Q > 18	D Q > 25	D Q > 22
Reg 8	Reg 9	Reg 10	Reg 11	Reg 12	Reg 13	Reg 14	Reg 15
D Q > 3	D Q > 10	D Q > 9	D Q > 112	D Q > 99	D Q > 98	D Q > 85	D Q > 0
Reg 16	Reg 17	Reg 18	Reg 19	Reg 20	Reg 21	Reg 22	Reg 23
D Q > 8	D Q > 86	D Q > 21	D Q > 77	D Q > 84	D Q > 98	D Q > 25	D Q > 17
Reg 24	Reg 25	Reg 26	Reg 27	Reg 28	Reg 29	Reg 30	Reg 31
D Q > 5	D Q > 67	D Q > 55	D Q > 44	D Q > 62	D Q > 33	D Q > 34	D Q > 47

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





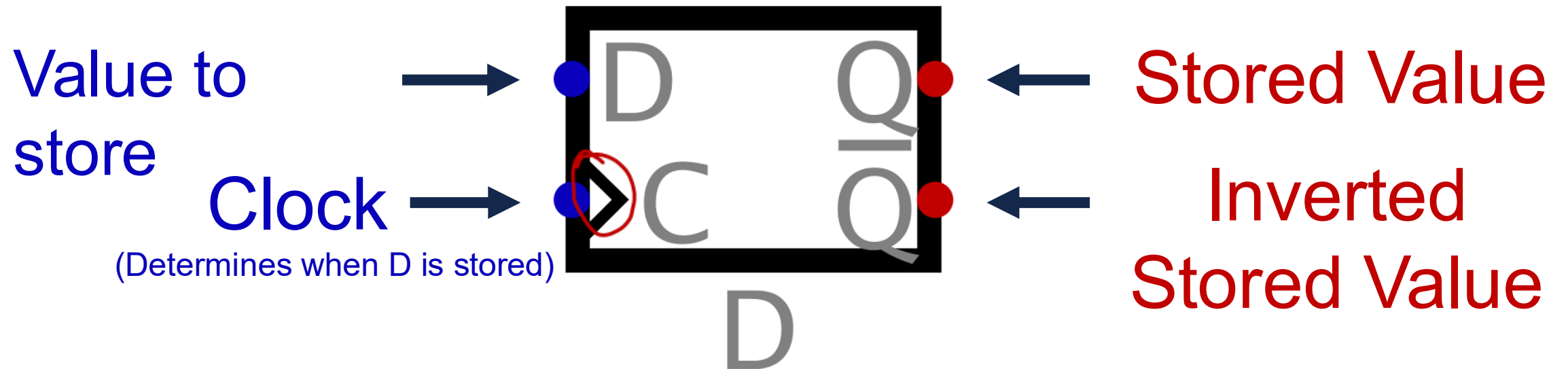
Flip Flops And Registers

Combinational vs. Synchronous Logic



- Combinational logic:
 - All the circuits we've seen so far!
 - Every element in the circuit is *recomputed* as soon as a change is detected on an input
 - Result dependent only on current input (no memory)
- Synchronous logic:
 - Circuits ~synced~ with a clock
 - Output depends on current input && past state
 - This past state is stored in a register!

D Flip-Flop



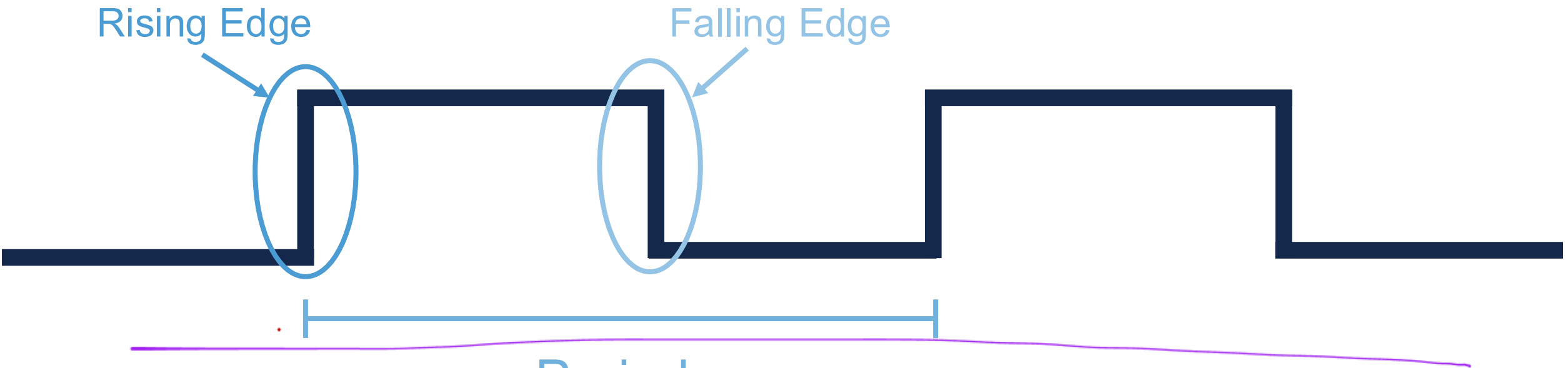
Clock



Rising Edge

Falling Edge

Period



Clock



- The clock is a one-bit signal that oscillates (or toggles) back and forth between 0 and 1
- The purpose of the clock is to ensure that our circuits stay in sync
- The period of the clock is the time between one rising edge to the next. The clock spends an equal time in low and high states.
 - The period determines how fast our circuit runs
 - The shorter the period – or in other words the higher the frequency – the faster our circuit will run

D Flip-Flop



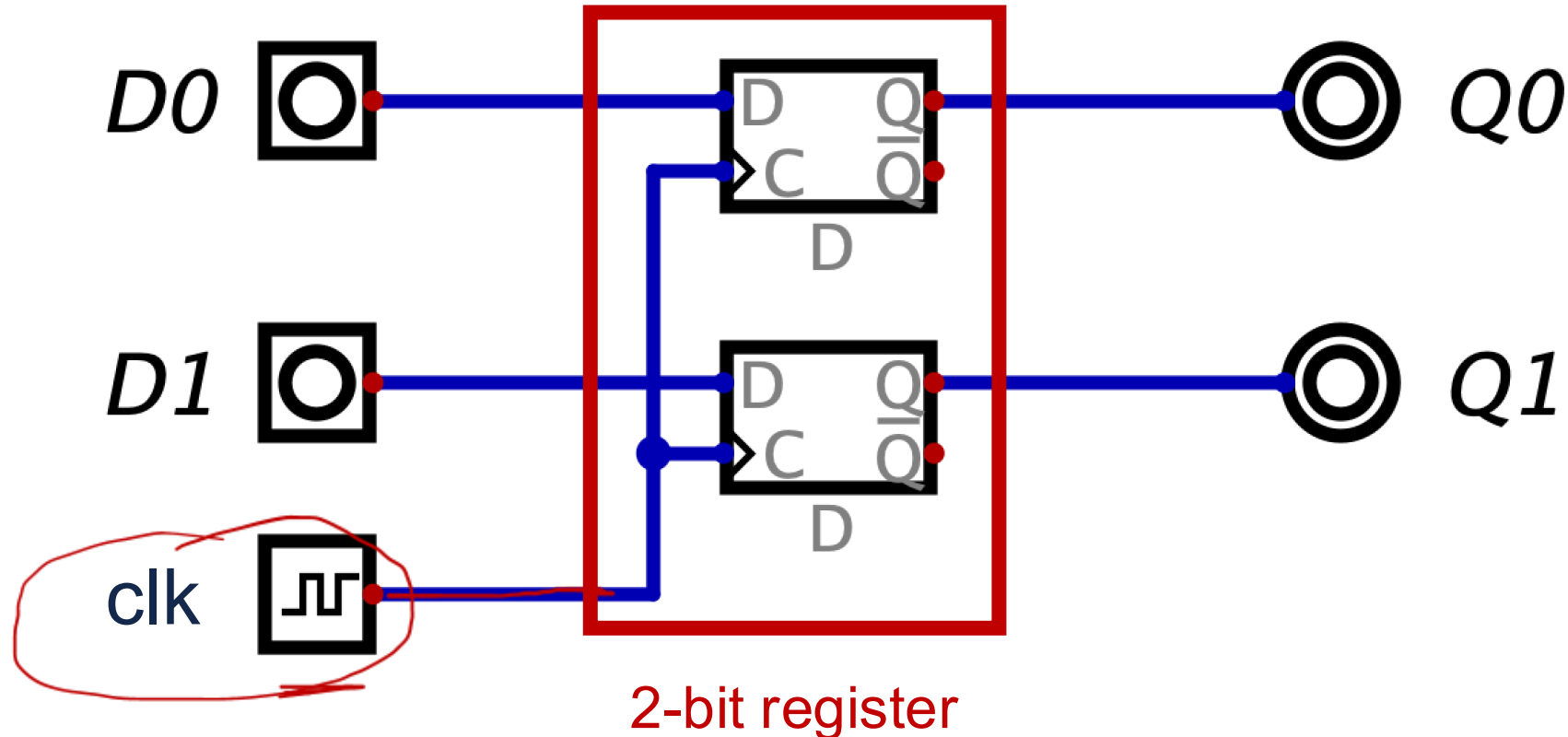
- Positive-edge triggered
 - Stores D when the clock goes from 0 to 1
- Negative-edge triggered
 - Stores D when the clock goes from 1 to 0



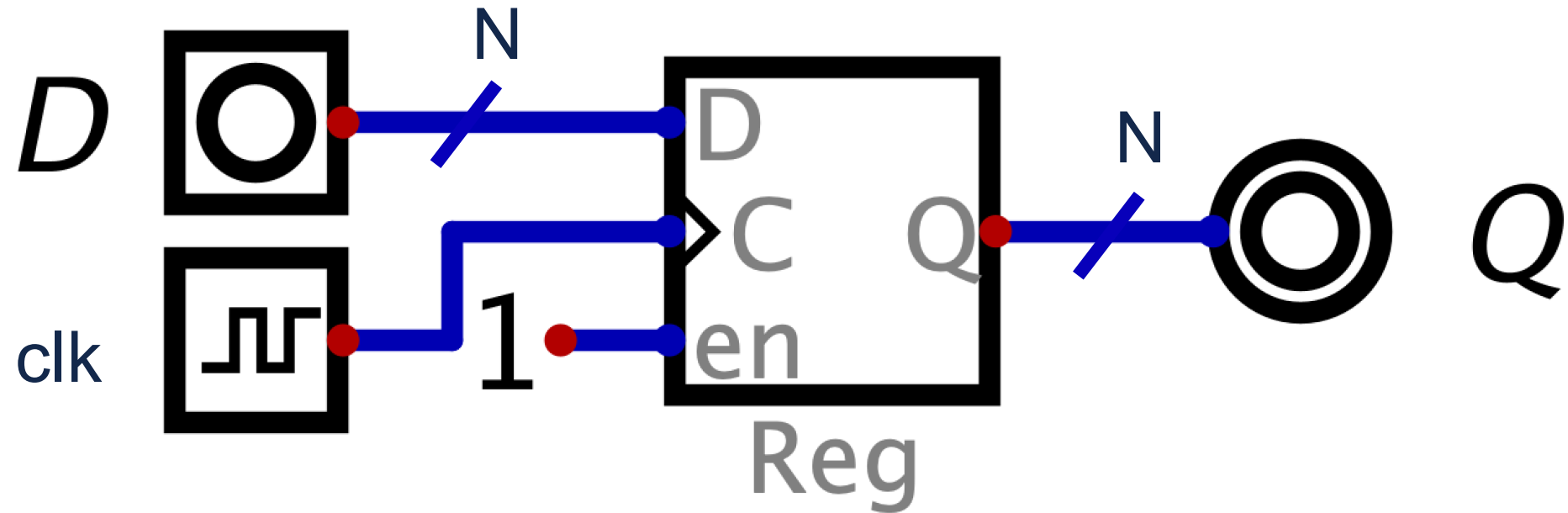
Registers



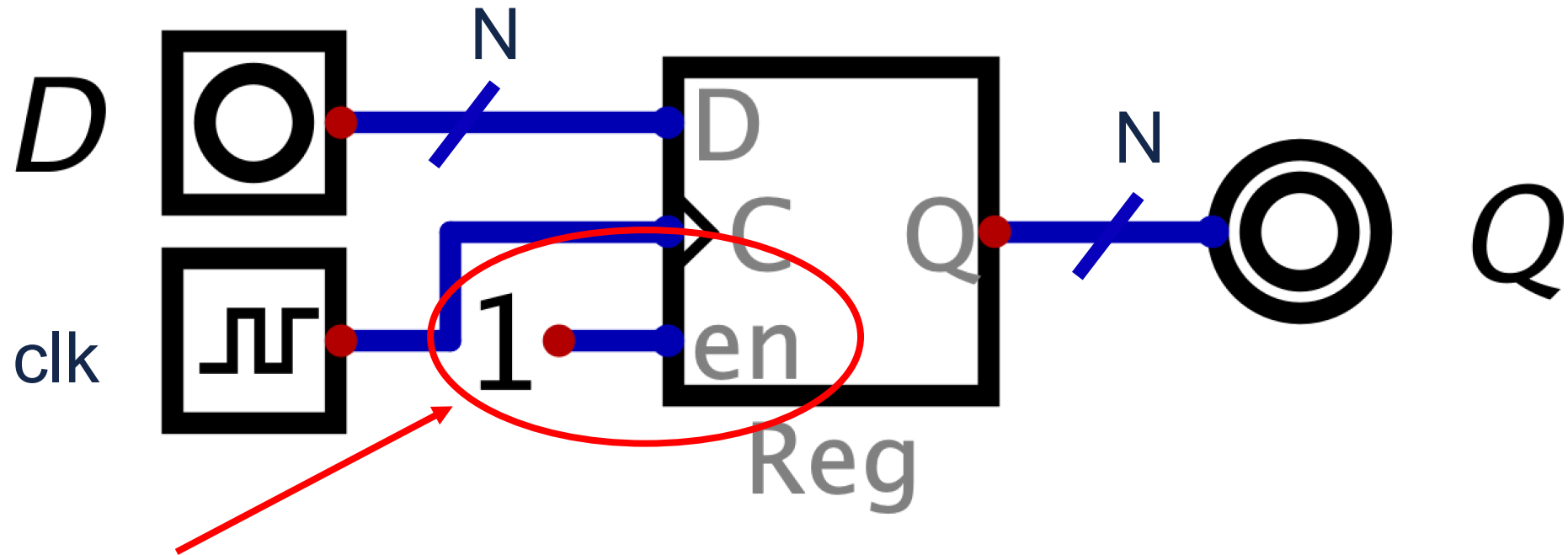
- To store multi-bit values, we can combine multiple FFs together into what is known as a register



N-bit Positive Edge Triggered Register



N-bit Positive Edge Triggered Register



Enable

- If 1, the register will read D on the rising edge of the clock
- If 0, the register will not read D

Registers Store the State & Results!



Operations to

Perform

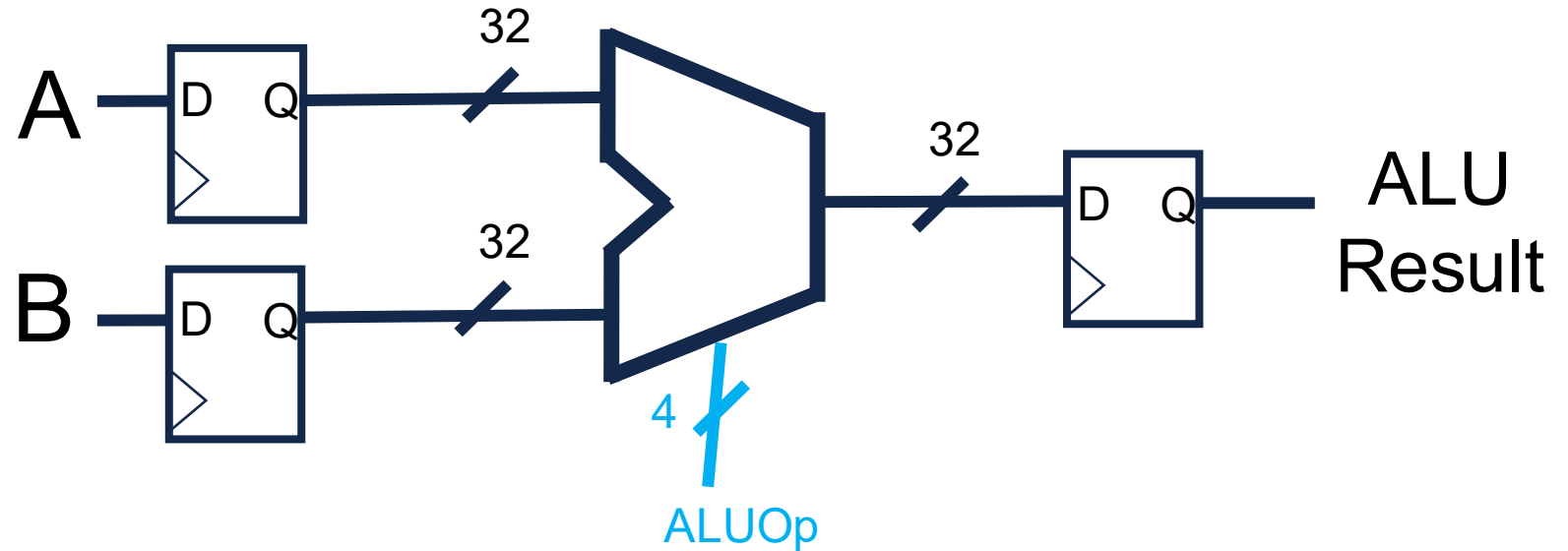
$3 + 4$

$5 - 2$

$9 \text{ OR } 4$

$6 \text{ AND } 8$

$5 + 9$



Assume all registers are connected to the same clock (not shown)

ALU



- The ALU will perform one operation per clock cycle
- To perform an operation, we have three inputs to set: A, B, and ALUOp
 - (ALU Op (ALU Control line from the textbook) is also synchronized with a register, but that is abstracted away for now)

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set less than
1100	NOR

	Operation	A	B	ALUOp
Cycle 0	3 + 4			
Cycle 1	5 - 2			
Cycle 2	9 OR 4			
Cycle 3	6 AND 8			
Cycle 4	5 + 9			

ALU



- The ALU will perform one operation per clock cycle
- To perform an operation, we have three inputs to set: A, B, and ALUOp
 - (ALU Op (ALU Control line from the textbook) is also synchronized with a register, but that is abstracted away for now)

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set less than
1100	NOR

	Operation	A	B	ALUOp
Cycle 0	3 + 4	3	4	0b0010
Cycle 1	5 - 2	5	2	0b0110
Cycle 2	9 OR 4	9	4	0b0001
Cycle 3	6 AND 8	6	8	0b0000
Cycle 4	5 + 9	5	9	0b0010

Dependent Operations

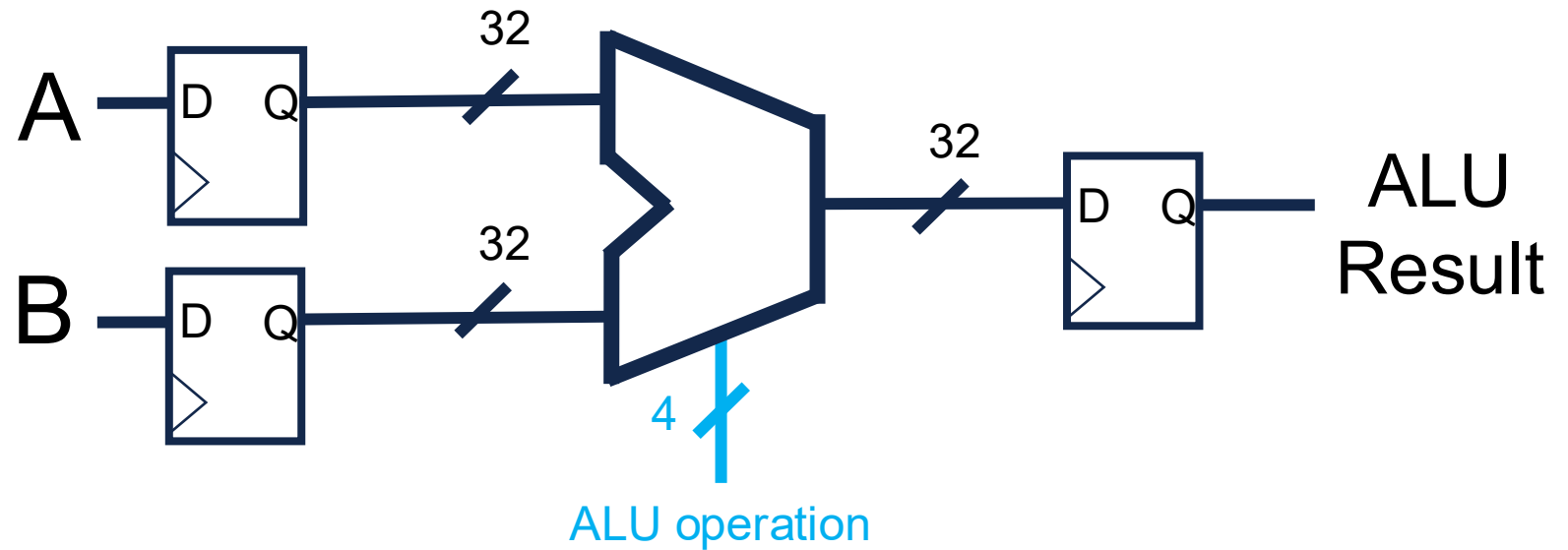


Operations to
Perform

$$a = 5 + 4$$

$$b = 7 + 8$$

$$c = a + b$$



Assume all registers are connected to the same clock (not shown)