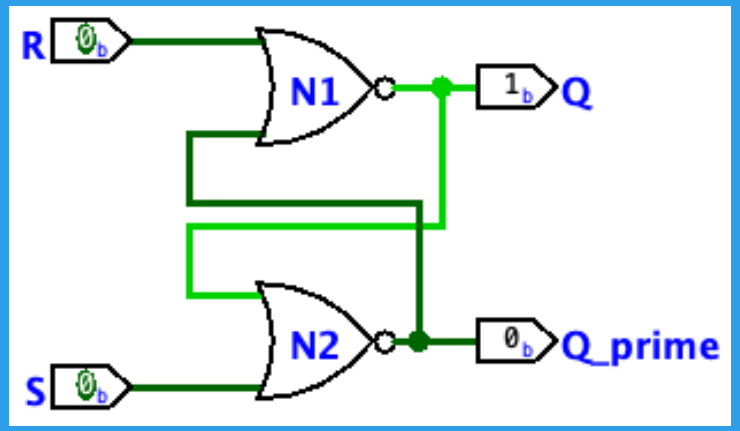
Computer Systems



Memory

CS51 – Spring 2026

Administrative

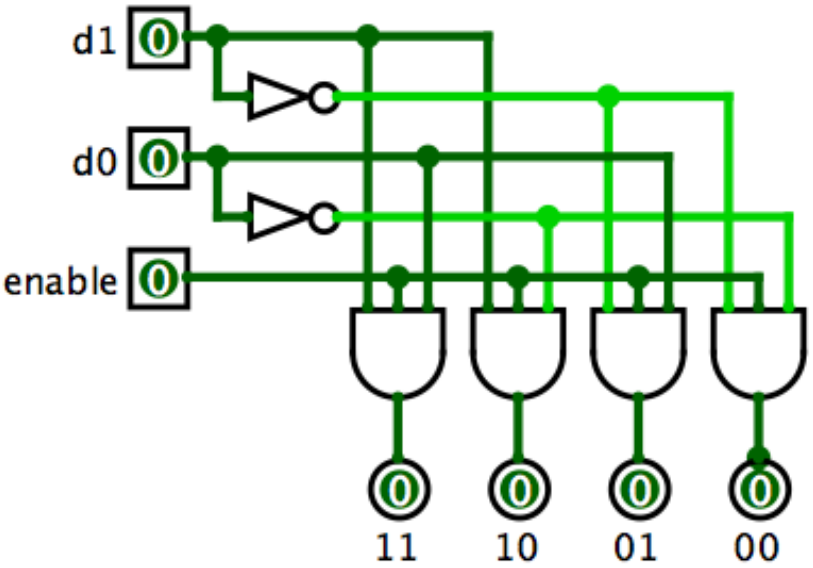
Assignment 3

Assignment 4

- Released
- Don’t forget to do pre-lab

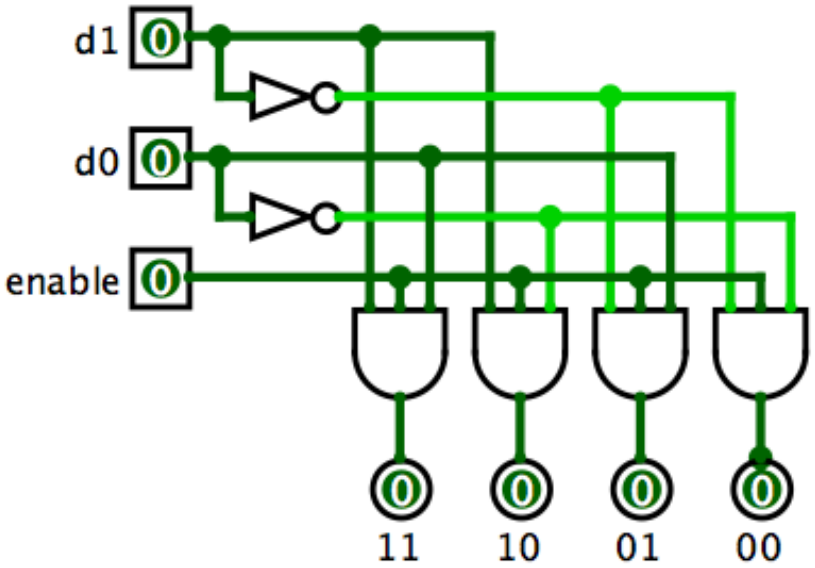
Lab tomorrow

2-bit decoder recap



What does a 2-bit decoder do?

2-bit decoder recap

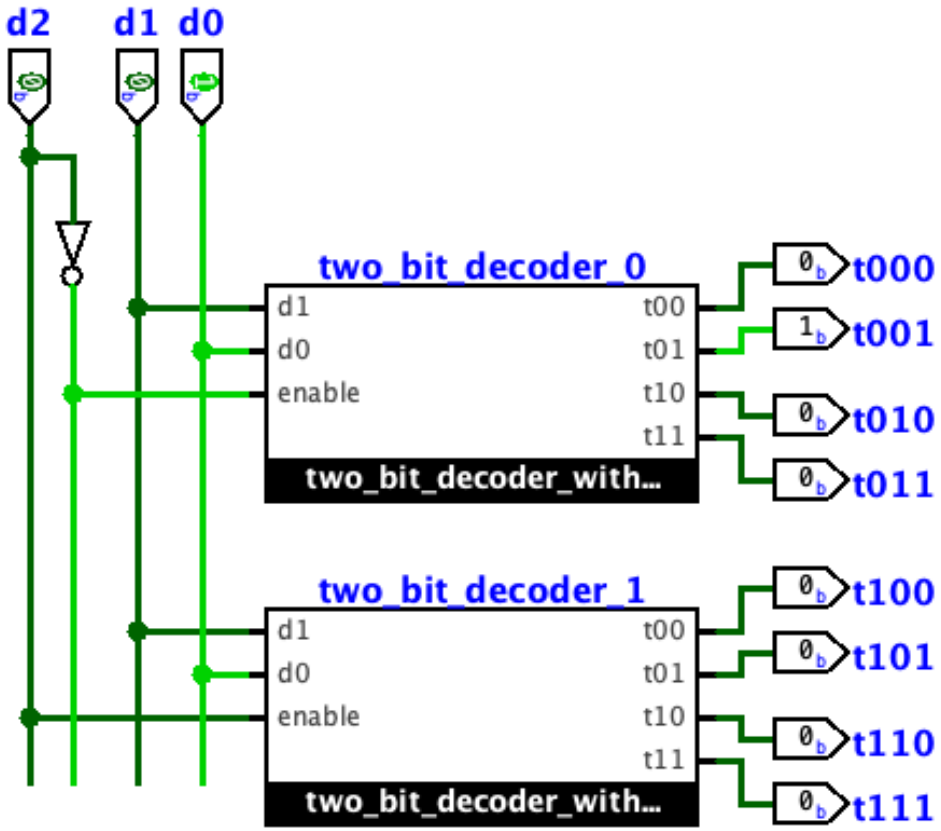


d_1	d_0	out_0	out_1	out_3	out_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Sends ‘1’ along one input line according to d_1 d_0

enable=0 turns off all lines

3-bit decoder



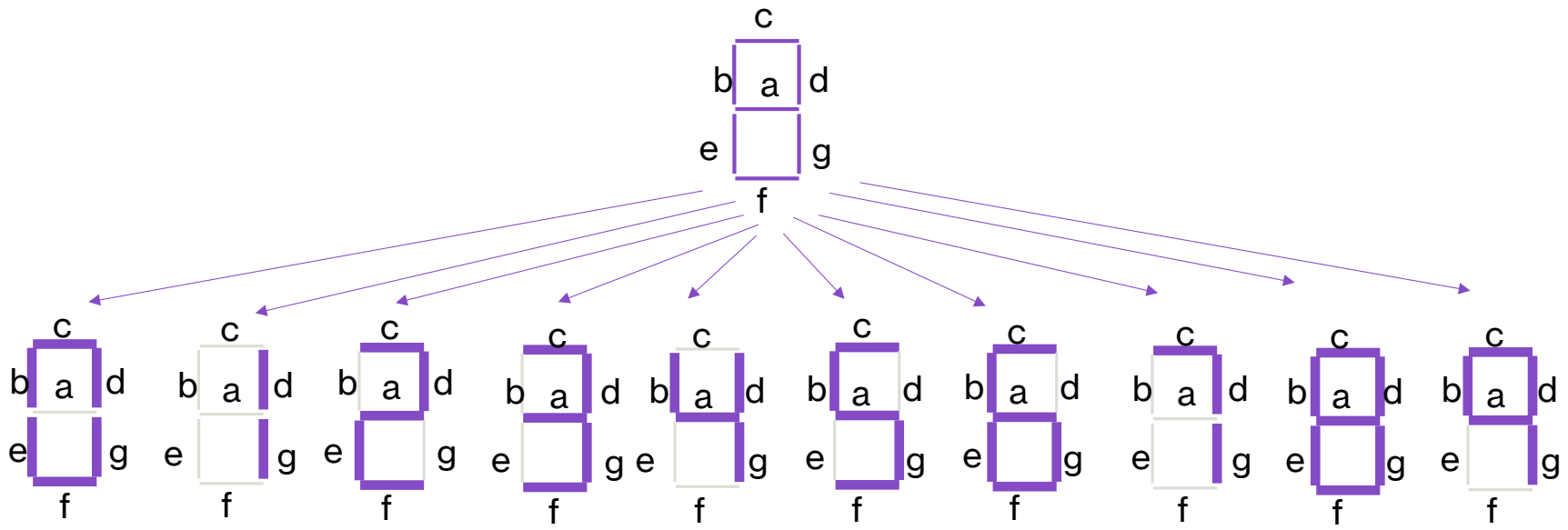
Inputs			Outputs							
d_2	d_1	d_0	t_{000}	t_{001}	t_{010}	t_{011}	t_{100}	t_{101}	t_{110}	t_{111}
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

- d_2 acts as the enable bit to two decoders.
- When d_2 is 0, the top decoder is activated, sending output to one of t0xx.
- When d_2 is 1, the bottom decoder is activated, sending output to one of t1xx.

Decoder application example

How do clocks like this work?

Decoding 7-segment displays

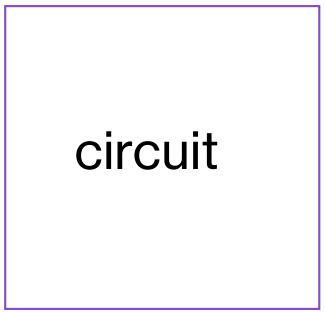


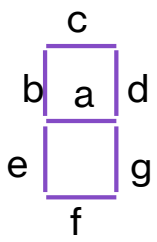
7 segments, labeled a to g

Different combinations of these segments, generate the digits 0-9

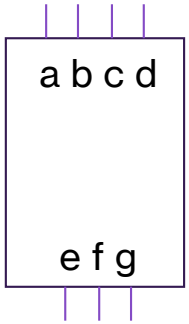
7-segment display

Number to display





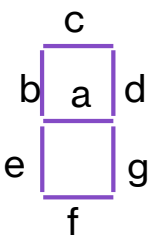
hardware component
with 7 inputs



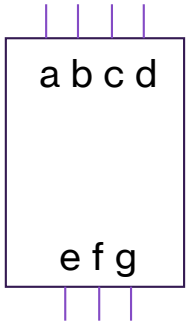
7-segment display

Number to display

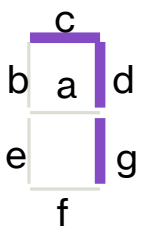
circuit



hardware component
with 7 inputs

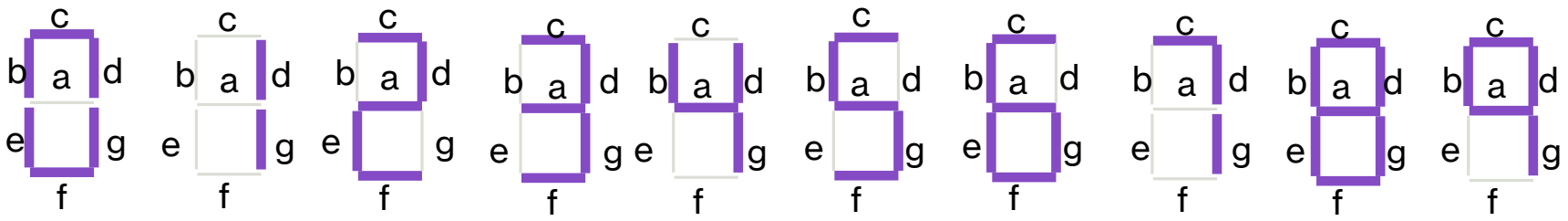


7



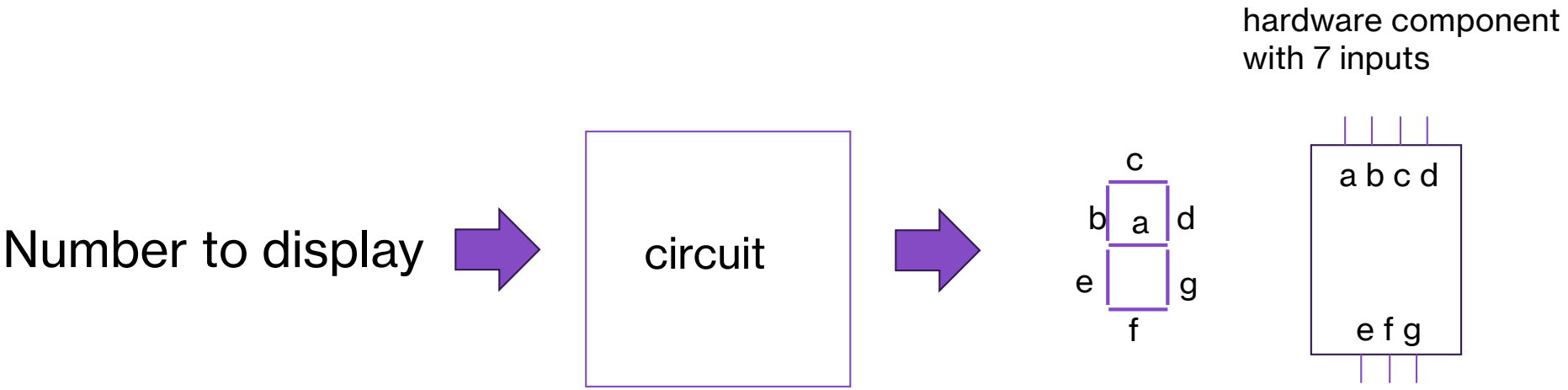
a = 0
b = 0
c = 1
d = 1
e = 0
f = 0
g = 1

7-segment displays in clocks



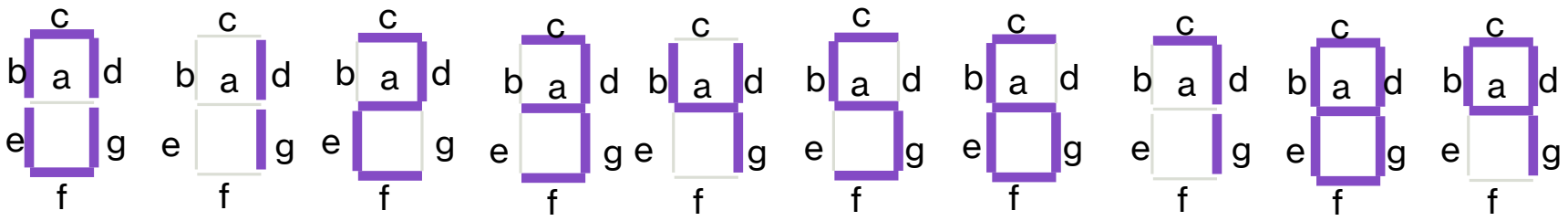
Digit	Illuminated segments						
	a	b	c	d	e	f	g
0		x	x	x	x	x	x
1				x			x
2	x		x	x	x	x	
3	x		x	x		x	x
4	x	x		x			x
5	x	x	x			x	x
6	x	x	x		x	x	x
7			x	x			x
8	x	x	x	x	x	x	x
9	x	x	x	x			x

7-segment display



How many binary inputs do we need for the numbers 0-9?

Decoding 7-segment displays in clocks



Input	Digit	Illuminated segments						
		a	b	c	d	e	f	g
0000	0		x	x	x	x	x	x
0001	1				x			x
0010	2	x		x	x	x	x	
0011	3	x		x	x		x	x
0100	4	x	x		x			x
0101	5	x	x	x			x	x
0110	6	x	x	x		x	x	x
0111	7			x	x			x
1000	8	x	x	x	x	x	x	x
1001	9	x	x	x	x			x

Our friend the decoder

2-bit decoder: 4 output lines
3-bit decoder: 8 output lines

4-input bits

How many output lines?

Our friend the decoder

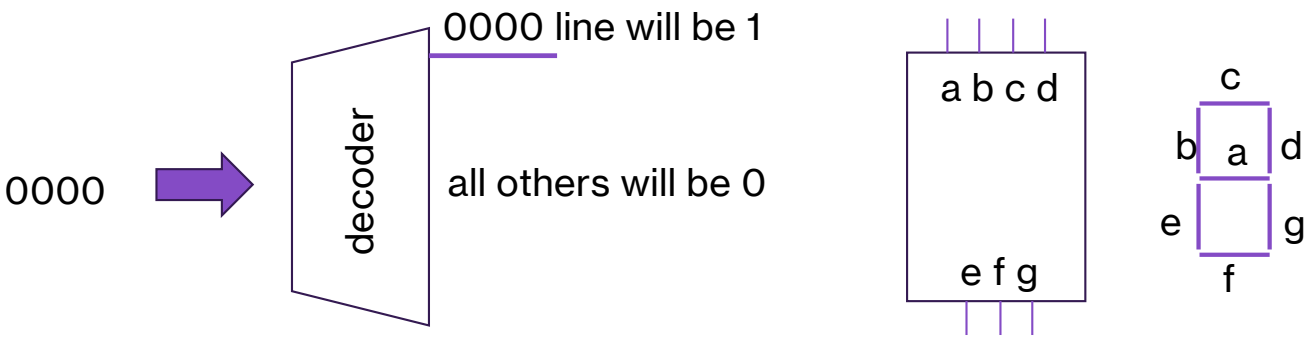
2-bit decoder: 4 output lines
3-bit decoder: 8 output lines

4-input bits

16 output lines: exactly 1 will be activated depending on the input

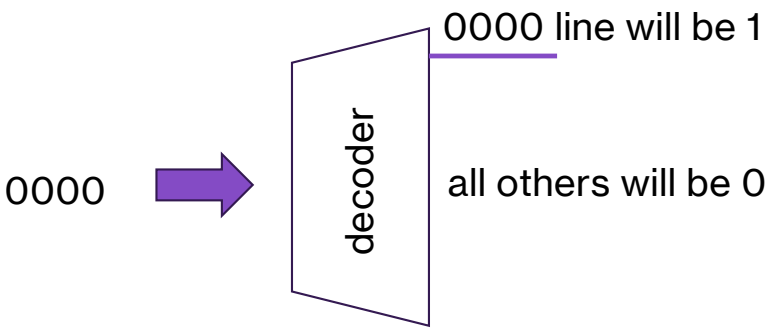
Our friend the decoder

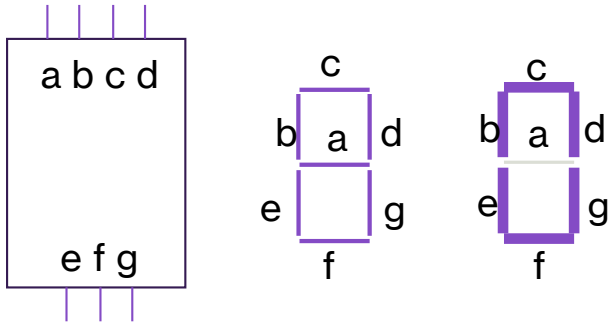
Input	Digit	Illuminated segments						
		a	b	c	d	e	f	g
0000	0		x	x	x	x	x	x



Our friend the decoder

Input	Digit	Illuminated segments						
		a	b	c	d	e	f	g
0000	0		x	x	x	x	x	x

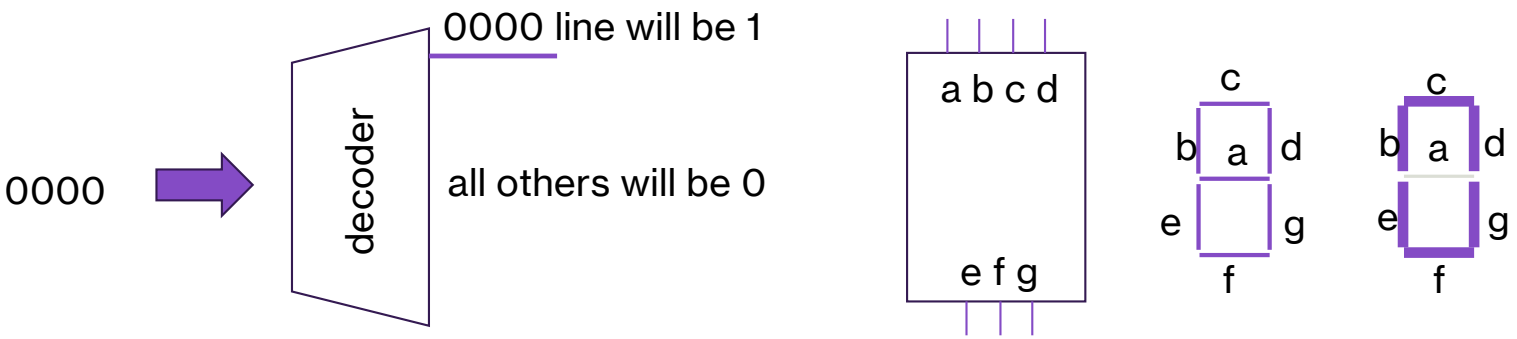




How do we get it to display a 0?

Our friend the decoder

Input	Digit	Illuminated segments						
		a	b	c	d	e	f	g
0000	0		x	x	x	x	x	x



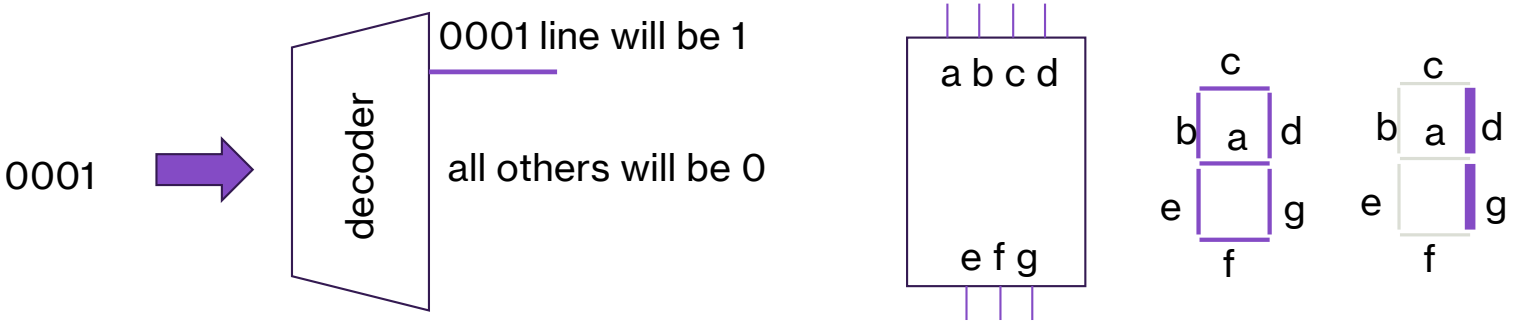
Connect the 0000 line to: b c d e f g

When the line is active those segments will illuminate

When it's not active, no display

Our friend the decoder

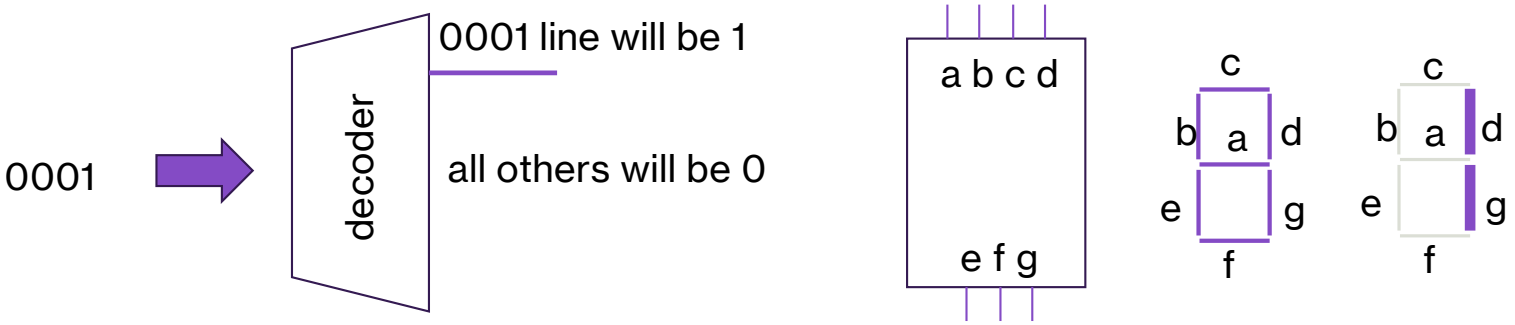
Input	Digit	Illuminated segments						
		a	b	c	d	e	f	g
0001	1				x			x



How do we get it to display a 1?

Our friend the decoder

Input	Digit	Illuminated segments						
		a	b	c	d	e	f	g
0001	1				x			x



Connect the 0001 line to: d g

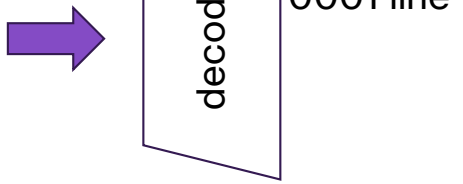
When the line is active those segments will illuminate

When it's not active, no display

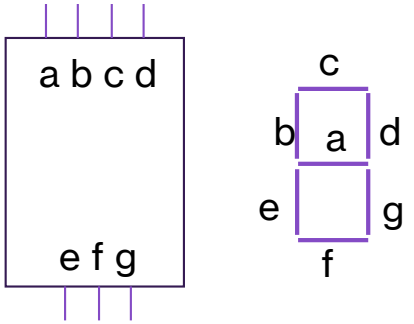
Our friend the decoder

Input	Digit	Illuminated segments						
		a	b	c	d	e	f	g
0000	0		x	x	x	x	x	x
0001	1				x			x

4-input bits



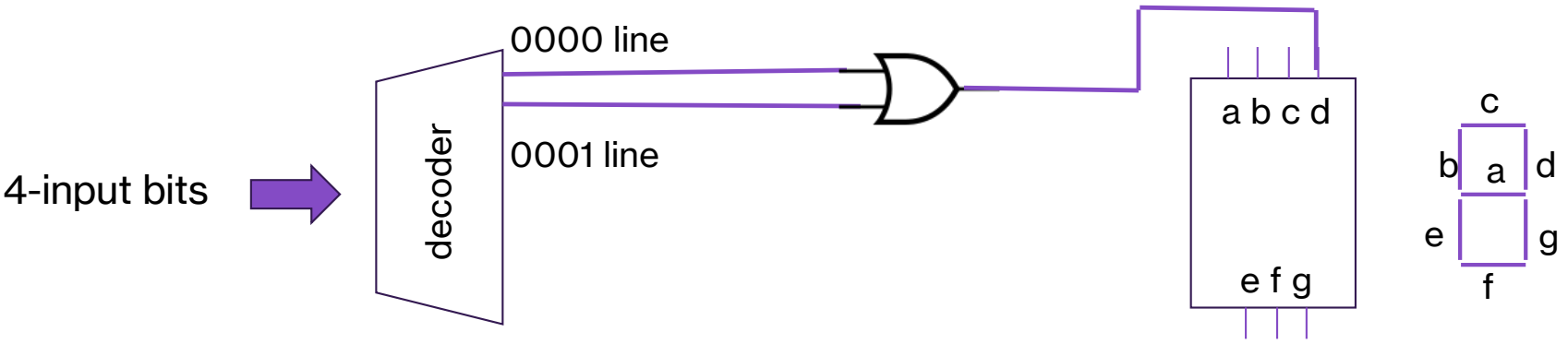
Both of these lines want to connect to d



How do we do this?
Hint: gate??

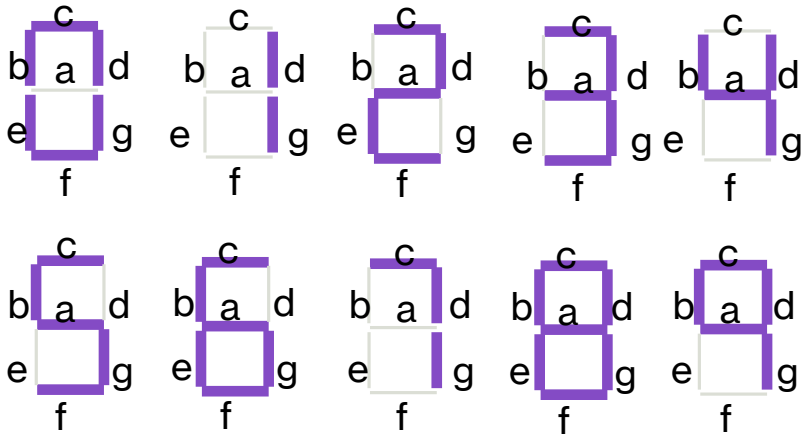
Our friend the decoder

Input	Digit	Illuminated segments						
		a	b	c	d	e	f	g
0000	0		x	x	x	x	x	x
0001	1				x			x

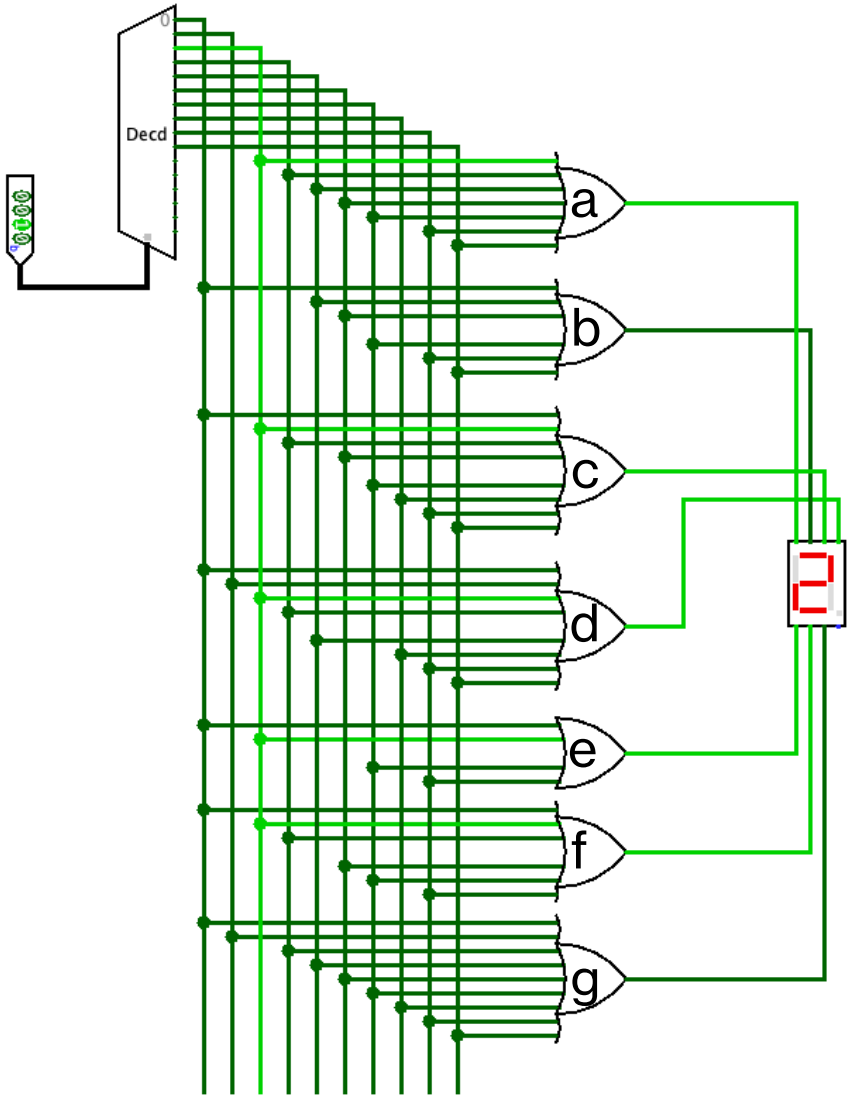


Since only one line will be active at a time, use OR gate

A 7-segment display in Logisim



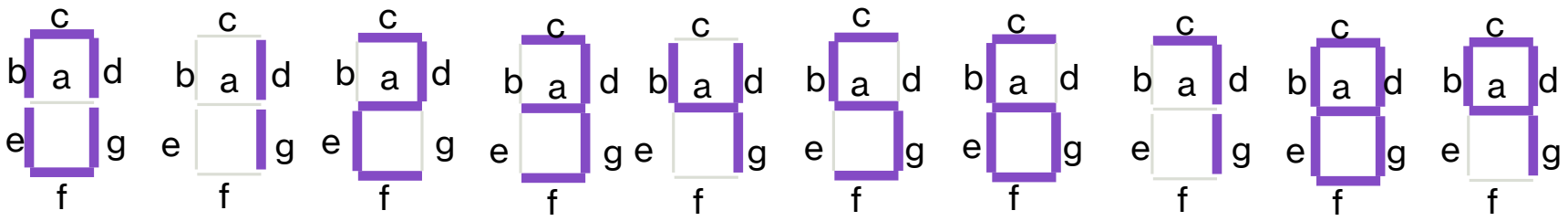
Input	Digit	Illuminated segments						
		a	b	c	d	e	f	g
0000	0		x	x	x	x	x	x
0001	1				x			x
0010	2	x		x	x	x	x	
0011	3	x		x	x		x	x
0100	4	x	x		x			x
0101	5	x	x	x			x	x
0110	6	x	x	x		x	x	x
0111	7			x	x			x
1000	8	x	x	x	x	x	x	x
1001	9	x	x	x	x			x



A 7-segment display in Logisim

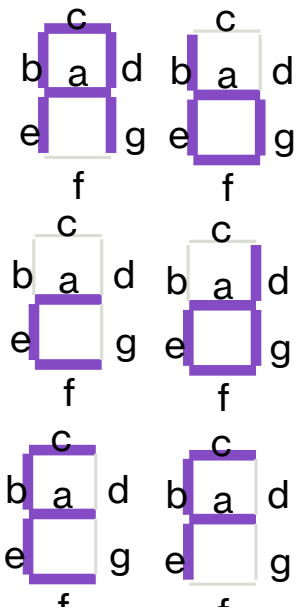
Check it out!

PRACTICE TIME – Expanding to 16 characters



Imagine that the 7-segment display we have built so far is expanded to include 16 characters, the existing 10 digits, and A-F, as depicted here to differentiate among them.

How will the design of the circuit change on Logisim-Evolution?

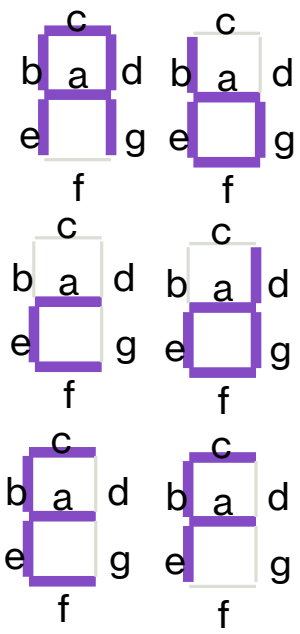


Input	Character	Illuminated segments						
		a	b	c	d	e	f	g
0000	0		x	x	x	x	x	x
0001	1				x			x
0010	2	x		x	x	x	x	
0011	3	x		x	x		x	x
0100	4	x	x		x			x
0101	5	x	x	x			x	x
0110	6	x	x	x		x	x	x
0111	7			x	x			x
1000	8	x	x	x	x	x	x	x
1001	9	x	x	x	x			x

ANSWER– Expanding to 16 characters

Expanded table:

Circuit in Logisim-Evolution is expanded to include all outputs of decoders connected to 6 additional or gates



Input	Character	Illuminated segments						
		a	b	c	d	e	f	g
0000	0		x	x	x	x	x	x
0001	1				x			x
0010	2	x		x	x	x	x	
0011	3	x		x	x		x	x
0100	4	x	x		x			x
0101	5	x	x	x			x	x
0110	6	x	x	x		x	x	x
0111	7			x	x			x
1000	8	x	x	x	x	x	x	x
1001	9	x	x	x	x			x
1010	A	x	x	x	x	x		x
1011	B	x	x			x	x	x
1100	C	x				x	x	
1101	D	x			x	x	x	x
1110	E	x	x	x		x	x	
1111	F	x	x	x		x		

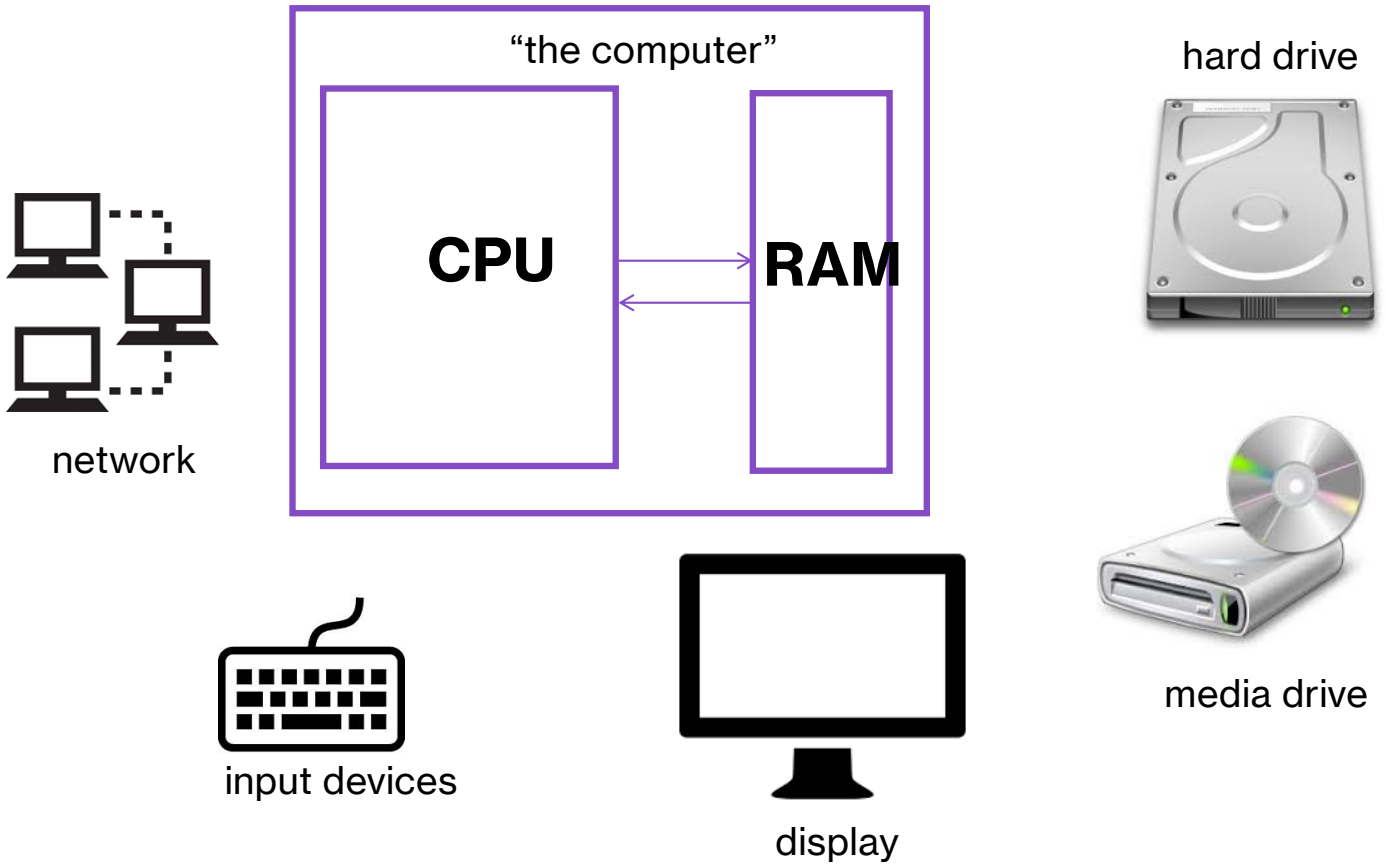
Combinational circuits

All circuits we have seen have been **combinational**: their output depends on the present inputs and there is no notion of memory of past inputs.

combinational circuits perform **arithmetic and logical operations**, **transmit data**, or **convert code**. So far, we have seen for

- *Arithmetic and logical operations*: logic circuits and half/full/and ripple-carry adders
- Transmit data: decoders (and we will see multiplexers in the next assignment)
- Convert code: 7-segment display decoders

Computer simplified

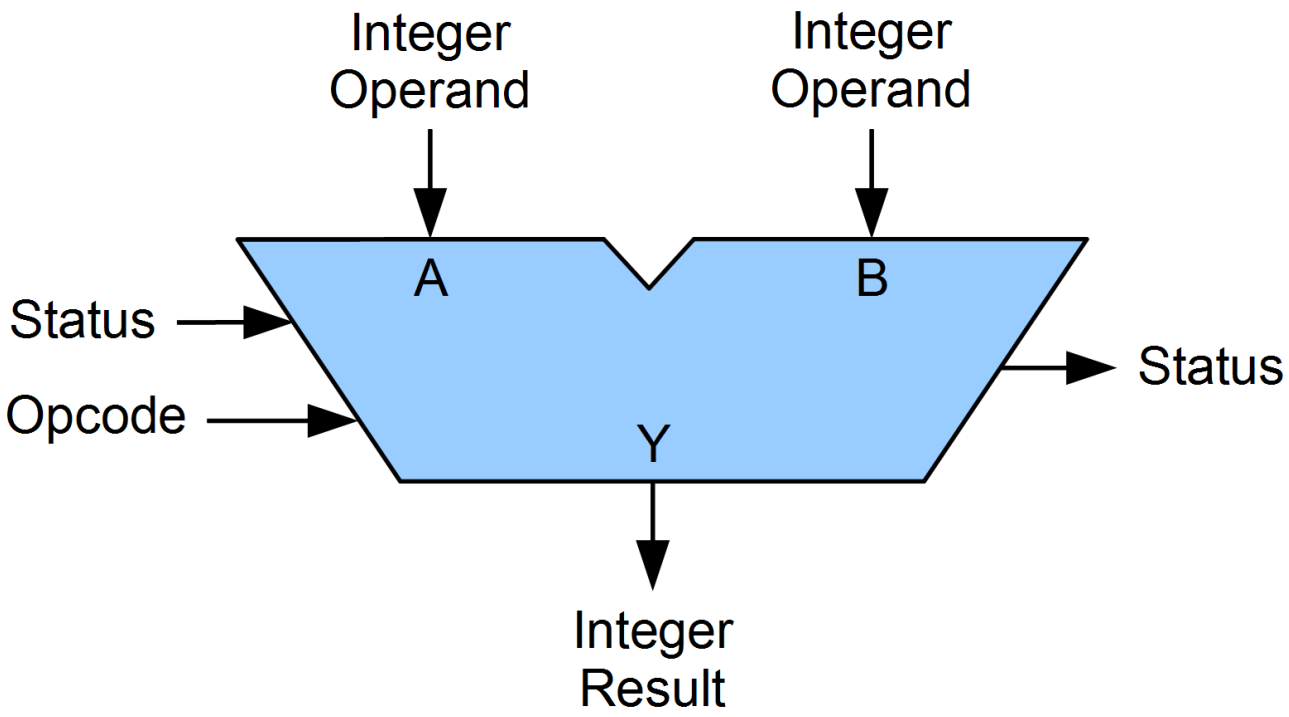


Inside the CPU: Arithmetic Logic Unit (ALU)

Performs arithmetic and logical operations

Arithmetic: add, subtract, multiply, ...

Logical: AND, OR, shift

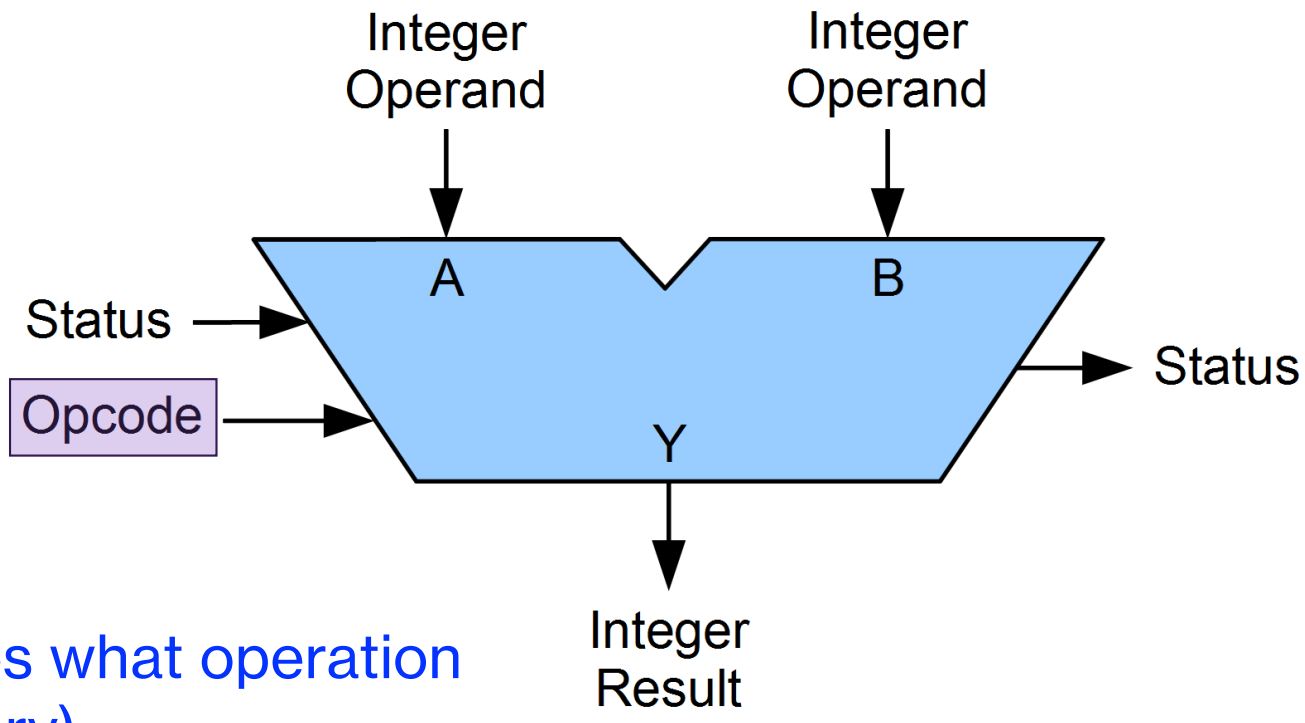


Inside the CPU: Arithmetic Logic Unit (ALU)

Performs arithmetic and logical operations

Arithmetic: add, subtract, multiply, ...

Logical: AND, OR, shift



Opcode specifies what operation (encoded in binary)

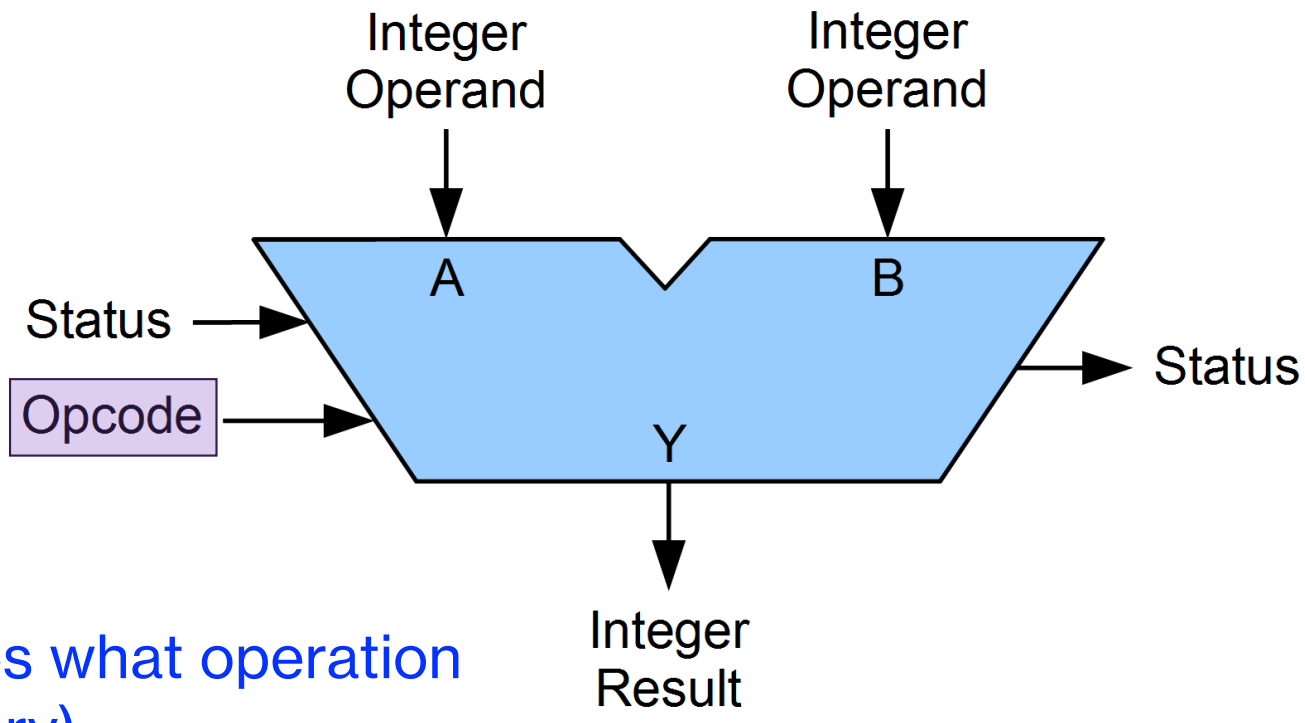
Inside the CPU: Arithmetic Logic Unit (ALU)

Where do the numbers come from?

Performs arithmetic and logical operations

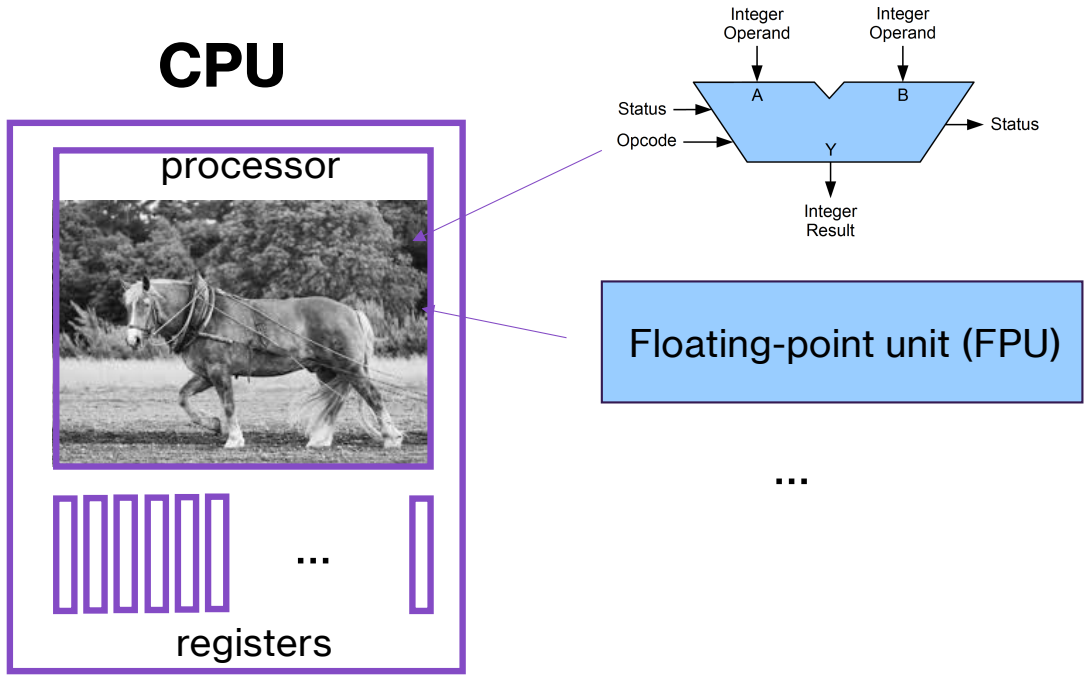
Arithmetic: add, subtract, multiply, ...

Logical: AND, OR, shift



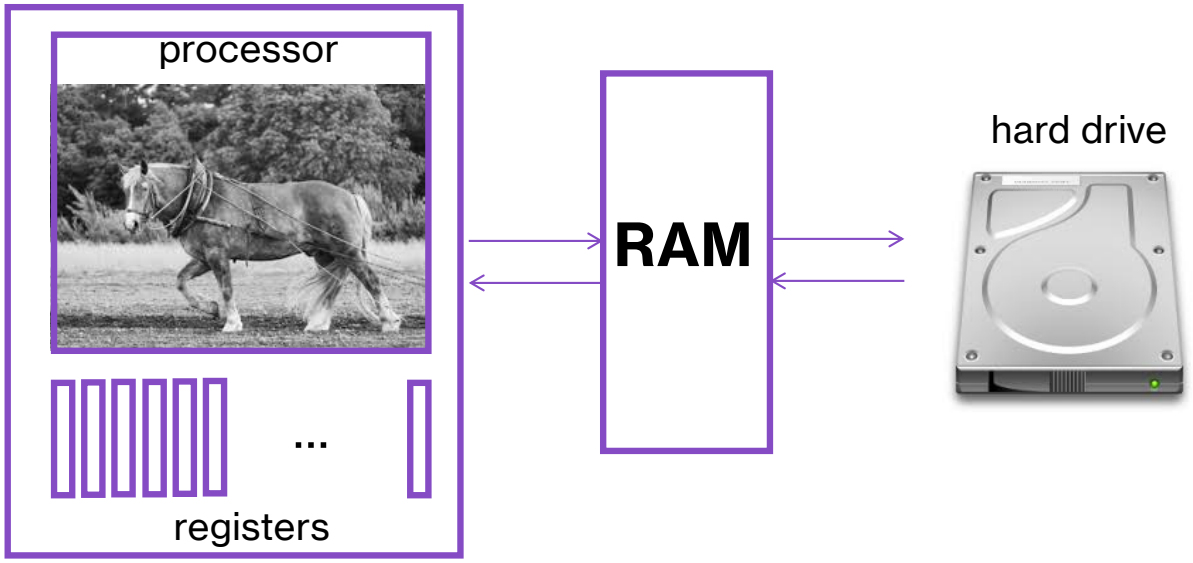
Opcode specifies what operation (encoded in binary)

Inside the CPU



registers: local, fast memory slots

Memory hierarchy

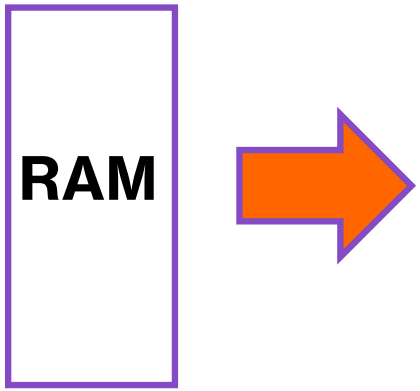


Why do we have these different types of memory?

Memory speed

operation	access time	times slower than register access	for comparison...
register	0.3 ns	1	1 s
RAM	120 ns	400	6 min
Hard disk (SSD)	1ms	~million	1 month
Hard disk (HDD)	10ms	~million	10 months
google.com	0.4s	~billion	30 years

Memory

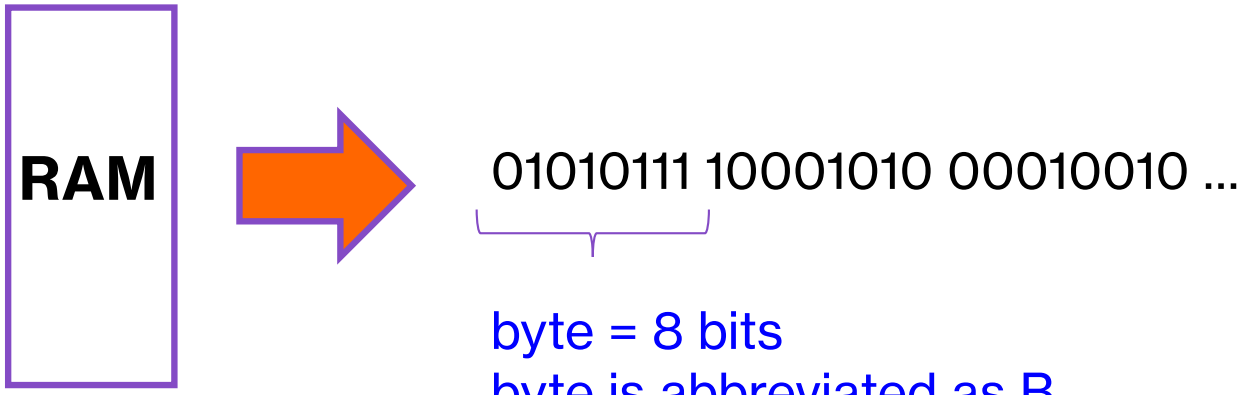


01010111100010100010010 ...

What is a byte?

?

Memory



byte = 8 bits
byte is abbreviated as B

My laptop has 64GB (gigabytes) of memory. How many bits is that?

Units of memory

Bit:

Nibble:

Byte: 8 bits

Kilobyte (KB):

Megabyte (MB)

Gigabyte (GB):

Terabyte (TB):

Units of memory

Bit: the smallest unit of memory.

Nibble: 4 bits, half a byte

Byte: 8 bits, the basic unit of measuring memory. Notation is B, e.g., 8B is 8 bytes or 64 bits.

Kilobyte (KB): 10^3 bytes $= 1{,}000$ bytes. (thousand bytes)

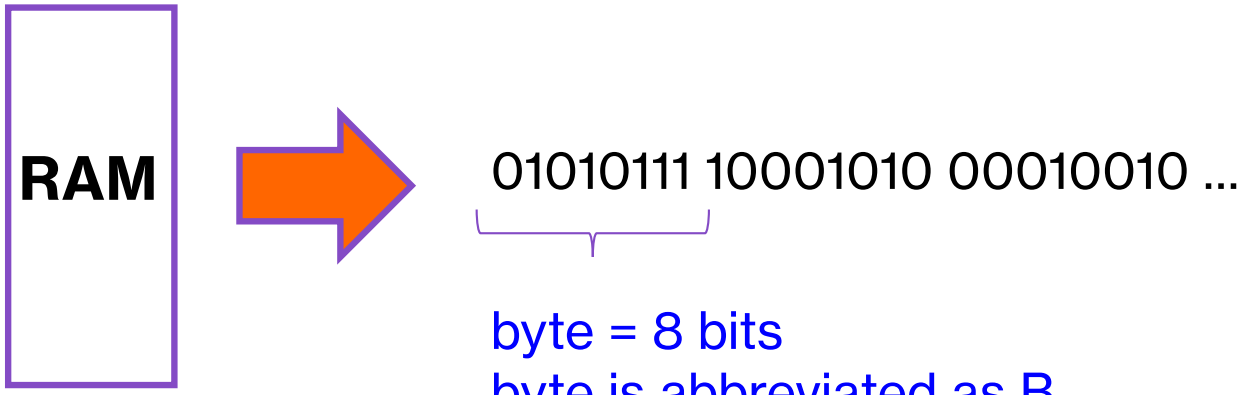
- Kibibyte = 2^{10} bytes = 1024 bytes (approximately a thousand bytes)

Megabyte (MB): 10^3 Kilobytes = $1{,}000{,}000$ bytes. (million bytes)

Gigabyte (GB): 10^3 Megabytes = $1{,}000{,}000{,}000$ bytes. (billion bytes)

Terabyte (TB): 10^3 Gigabytes = $1{,}000{,}000{,}000{,}000$ bytes. (quadrillion bytes)

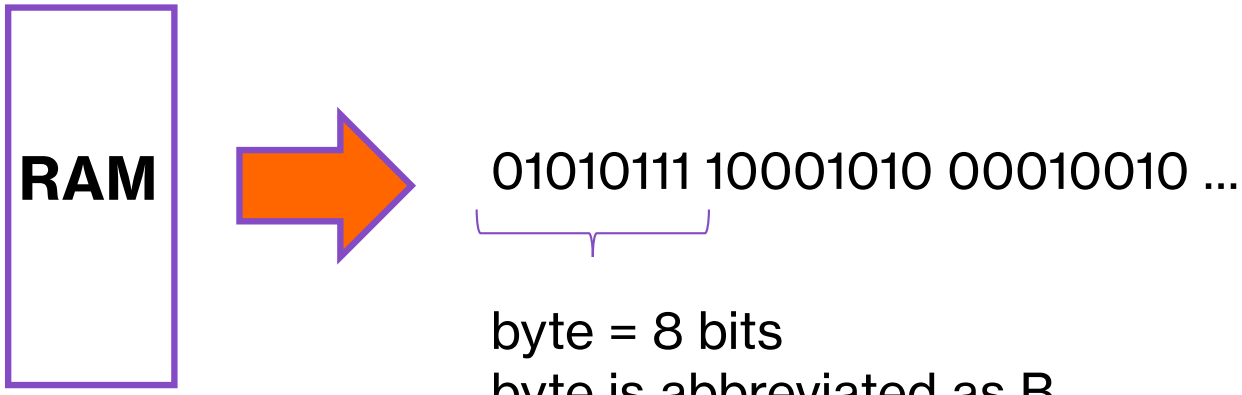
Memory



byte = 8 bits
byte is abbreviated as B

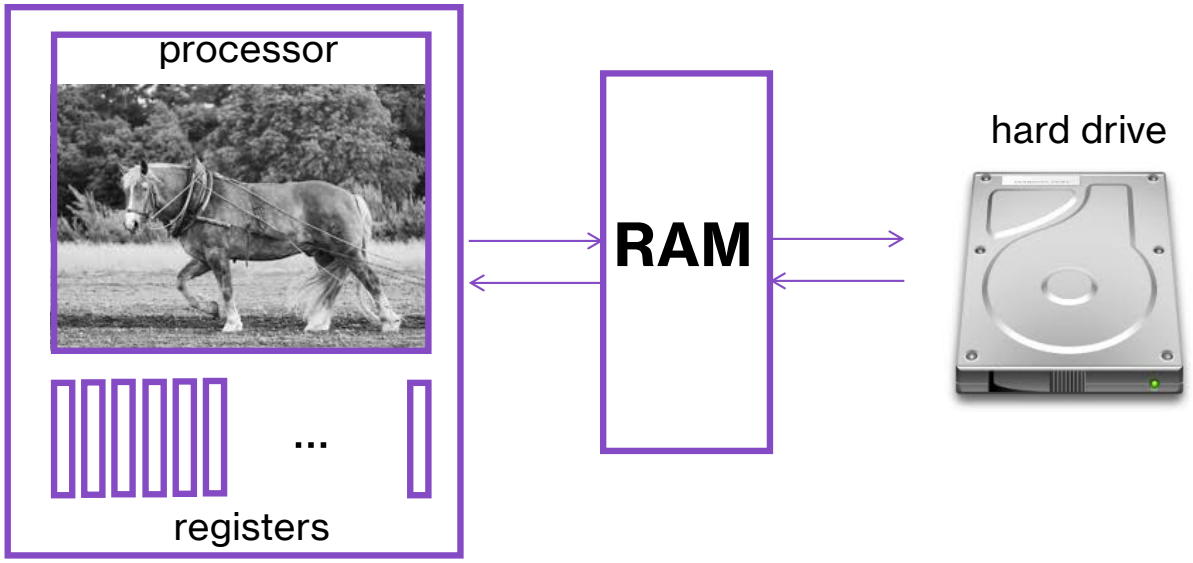
My laptop has 64GB (gigabytes) of memory. How many bits is that?

Memory



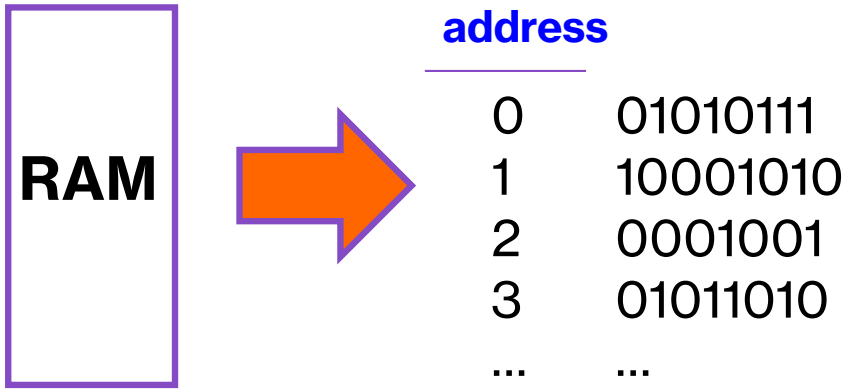
64GB = 64,000,000,000 = 64 billion bytes = 512 billion bits

Memory hierarchy



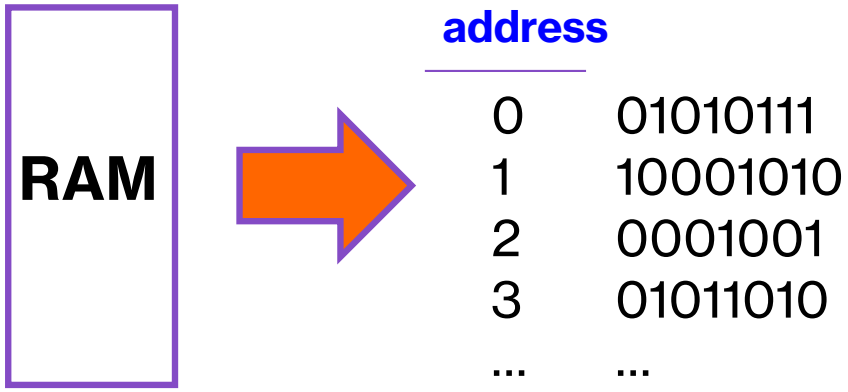
Capacity: hundreds of registers Gigabytes Terabytes

Memory



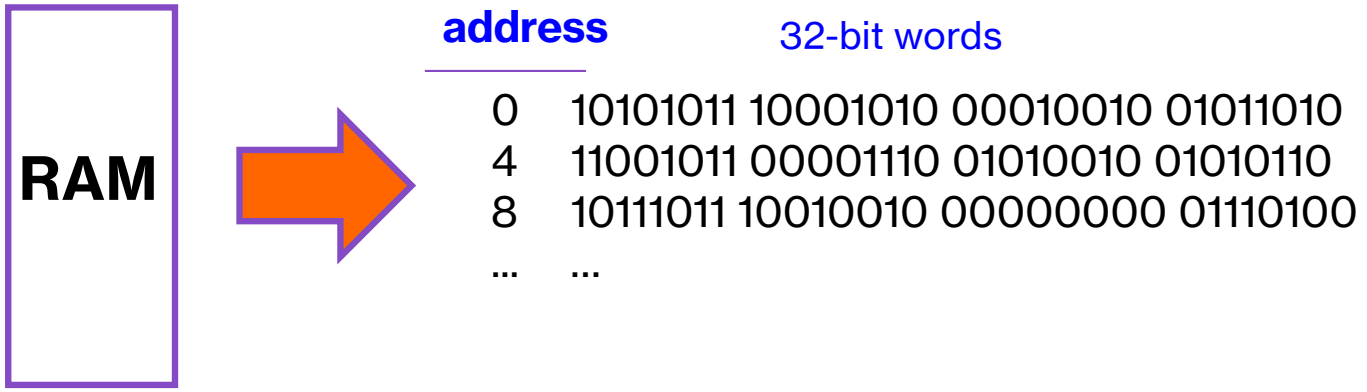
Memory is byte addressable

Memory



Memory is organized into “words”, which is the most common functional unit

Memory



Most modern computers use 32-bit (4 byte) or 64-bit (8 byte) words

Memory capacity

Memory is commonly byte addressable (i.e., we can ask for a specific byte)

32-bit processors access memory locations using a 32-bit number

What is the largest a 32-bit, byte-addressable memory can be?
Hint: how many unique addresses are possible?

Memory capacity

With 32 bits, we can represent 2^{32} unique numbers.

$$2^{32} = 2^2 2^{10} 2^{10} 2^{10} \approx 4 * 10^3 10^3 10^3 = 4 * 10^9$$

We could therefore address about 4 billion bytes, which is 4GB

Memory capacity

64-bit processors access memory locations using a 64-bit number

What is the largest a 64-bit, byte-addressable memory can be?
Hint: how many unique addresses are possible?

Memory capacity

With 64 bits, we can represent 2^{32} unique numbers.

$$2^{64} = 2^4 2^{10} 2^{10} 2^{10} 2^{10} 2^{10} 2^{10} \approx 16 * 10^{18}$$

KB, GB, TB, Petabyte, Exabyte

16 Exabytes = 16 * 10^6 TB = 16 Million Terabytes

Memory implemented

Memory is generally implemented as a grid/matrix of bytes

16 columns

16 rows

	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	c15
r0																
r1																
r2																
r3																
r4																
r5																
r6																
r7																
r8																
r9																
r10																
r11																
r12																
r13																
r14																
r15																

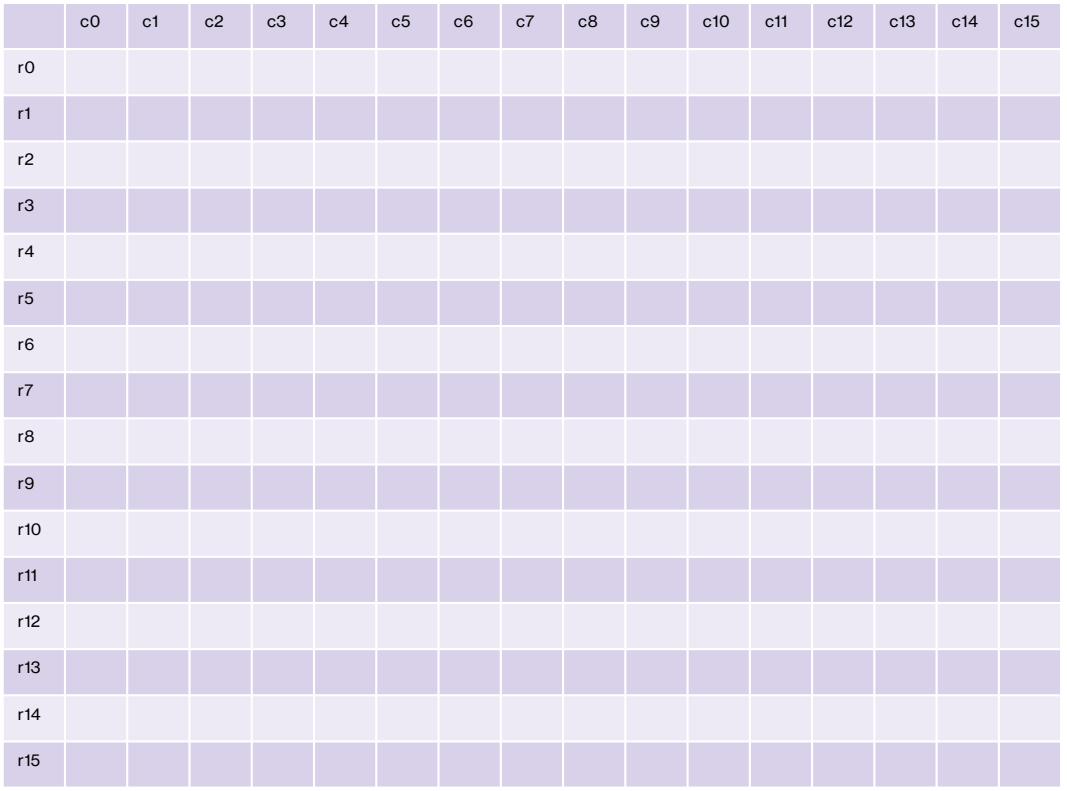
Memory implemented

Memory is generally implemented as a grid/matrix of bytes

How much memory is this?

16 columns

16 rows



Memory implemented

Memory is generally implemented as a grid/matrix of bytes

$16 * 8 = 2^4 * 2^4 = 2^8$ = 256 bytes

16 columns

16 rows

	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	c15
r0																
r1																
r2																
r3																
r4																
r5																
r6																
r7																
r8																
r9																
r10																
r11																
r12																
r13																
r14																
r15																

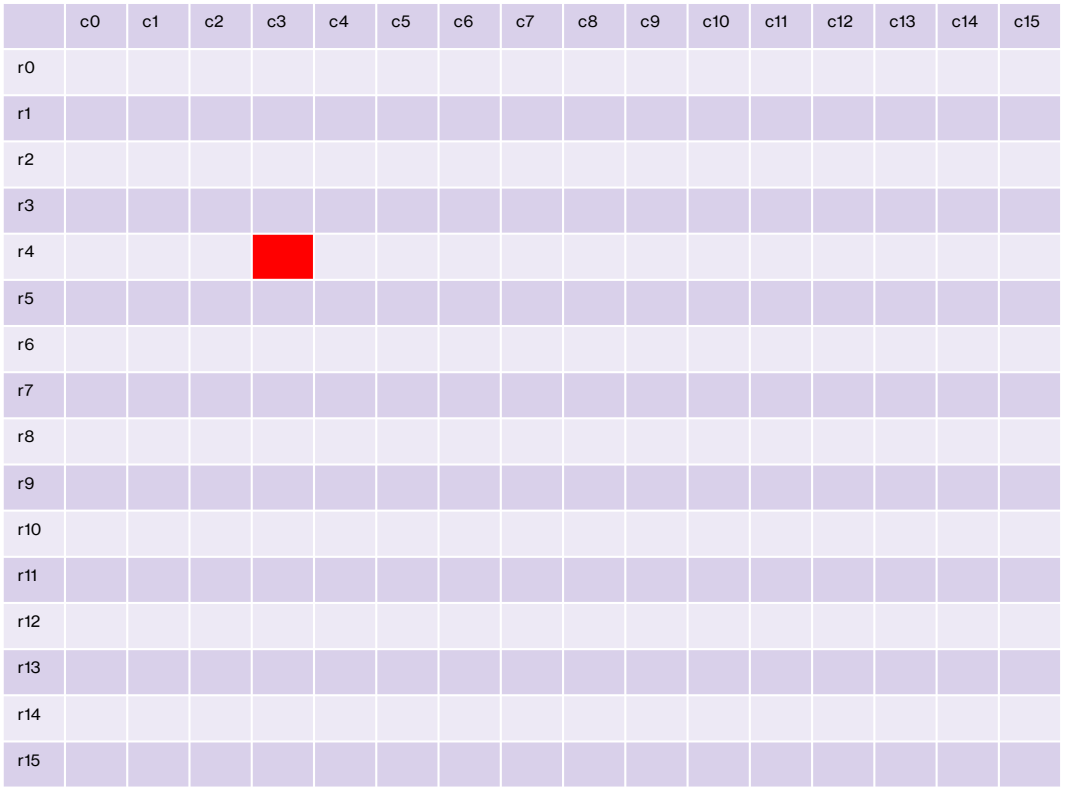
Memory implemented

We can access a given by via a row index and a column index

What row/column would this entry be? (Assuming indices start at 0)

16 columns

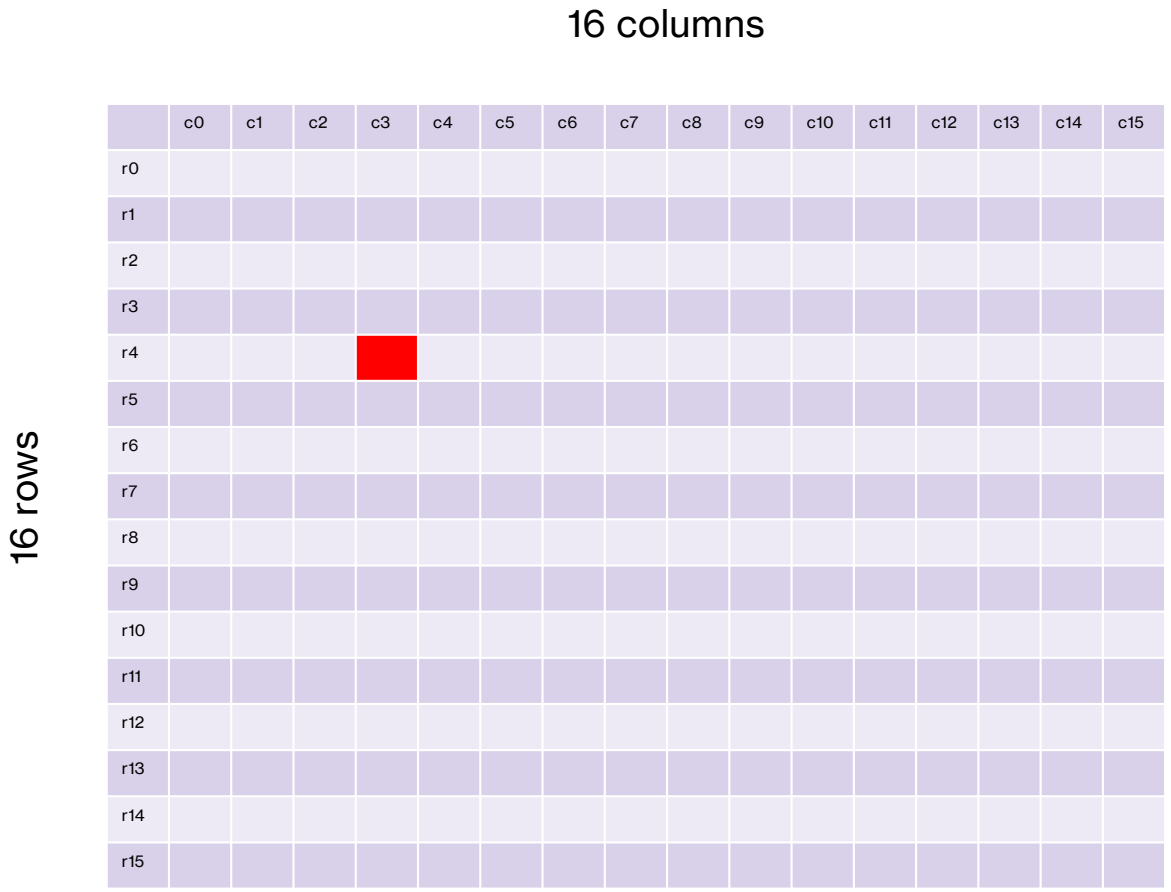
16 rows



Memory implemented

We can access a given by via a row index and a column index

row = 4, column = 3



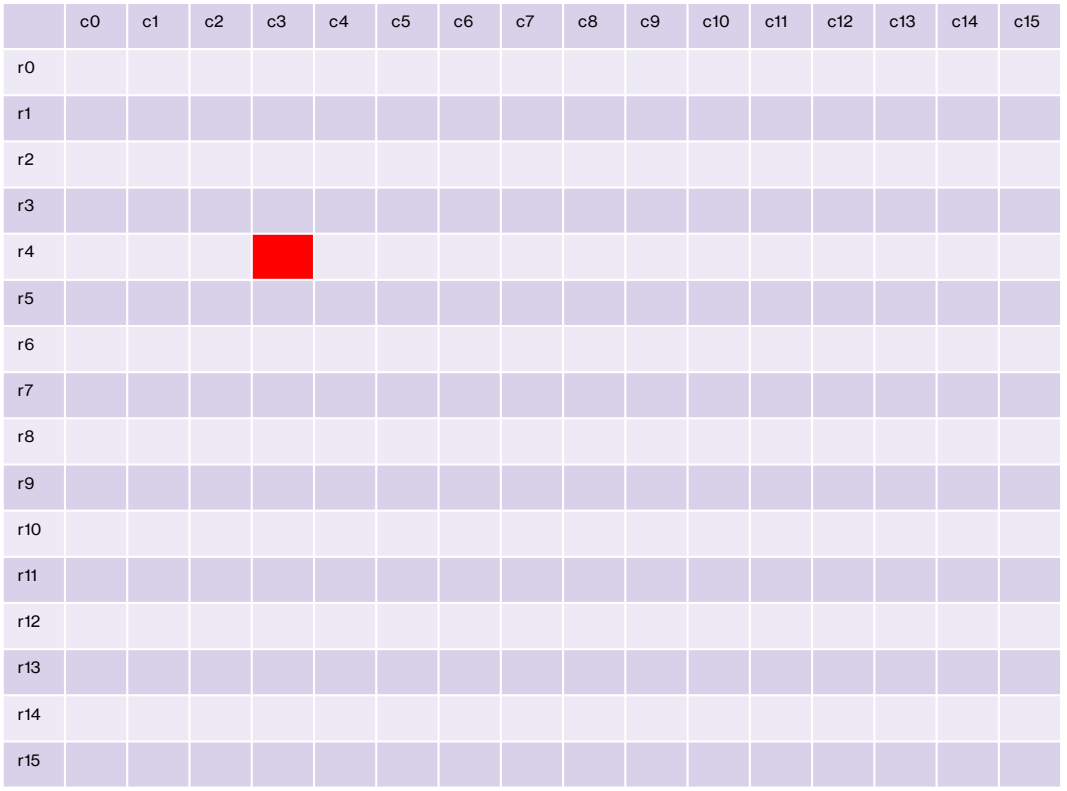
Memory implemented

We can access a given by via a row index and a column index

Everything is binary: how many bits do we need to specify the row and column?

16 columns

16 rows



Memory implemented

We can access a given by via a row index and a column index

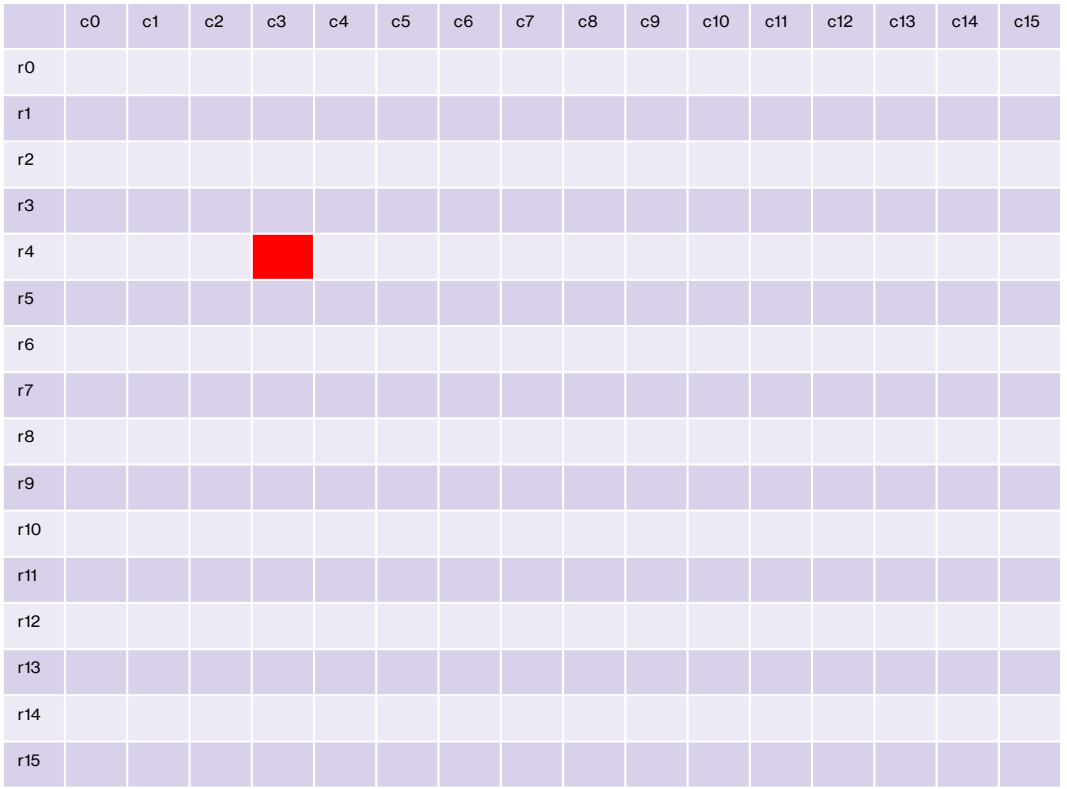
4 bits each for a total of 8 bits

<8-bit memory address>

4 bits for row 4 bits for column

16 columns

16 rows



Memory implemented

We can access a given by via a row index and a column index

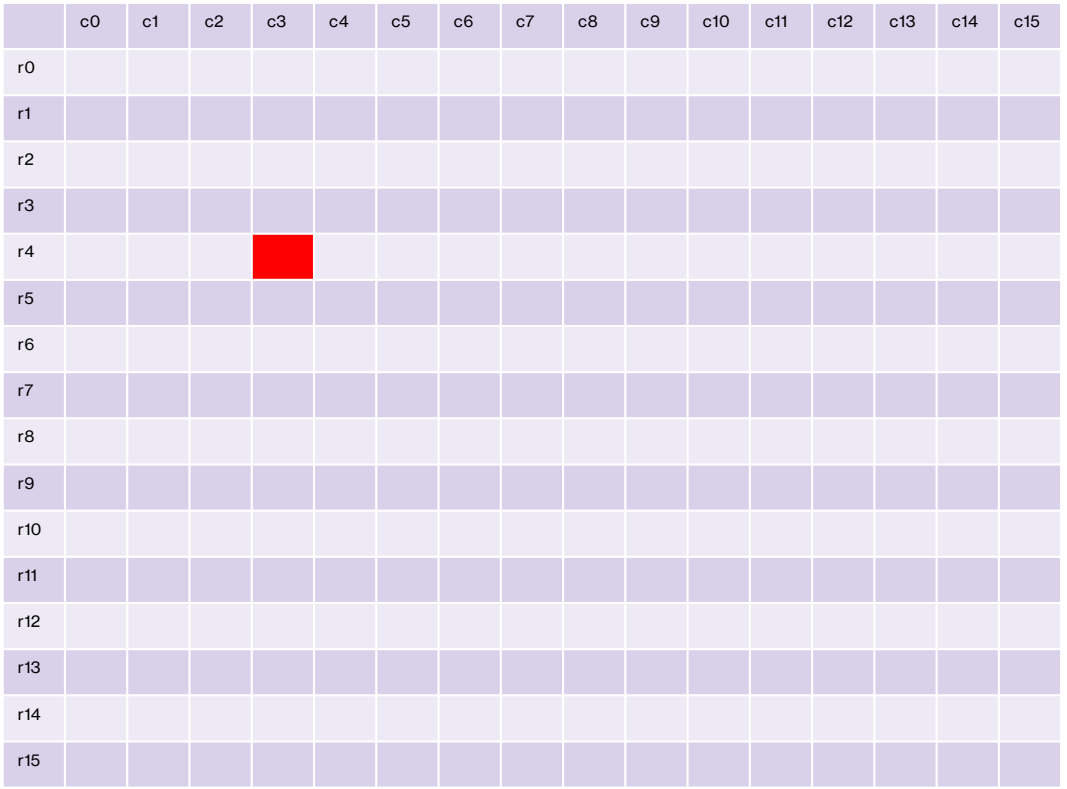
What would be this memory address in binary?

<8-bit memory address>

4 bits for row 4 bits for column

16 columns

16 rows



Memory implemented

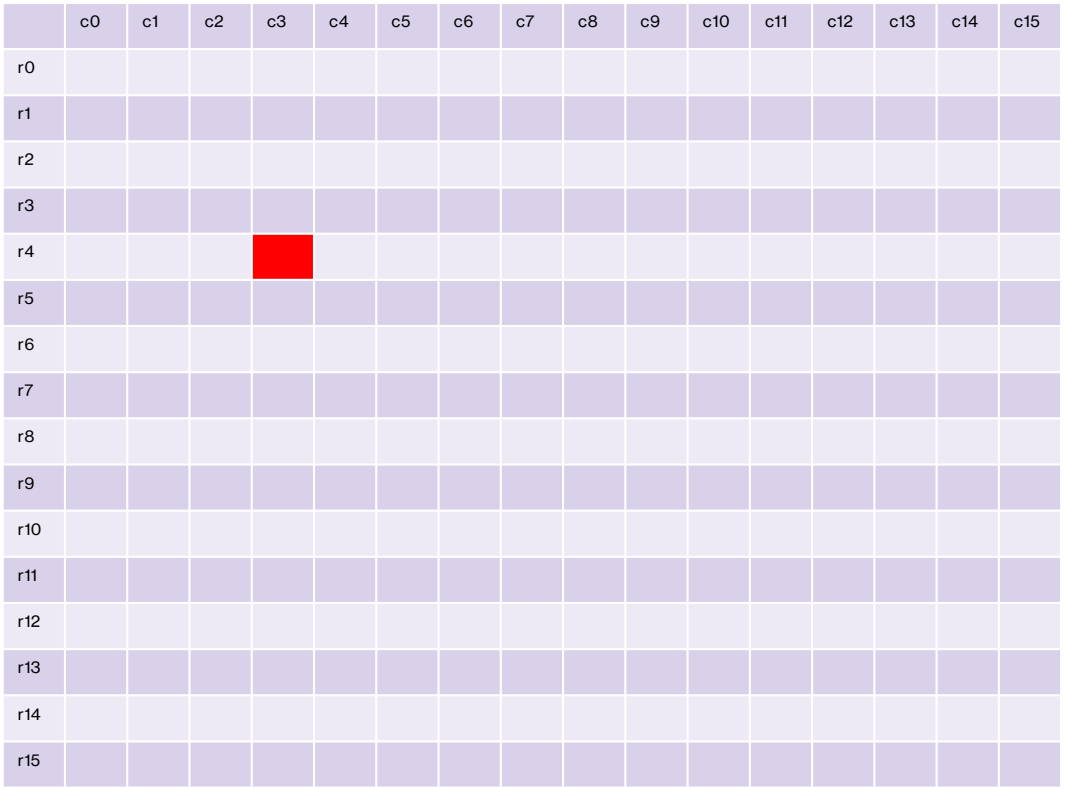
We can access a given by via a row index and a column index

0100 0011

row column

16 columns

16 rows



Memory implemented

We can access a given by via a row index and a column index

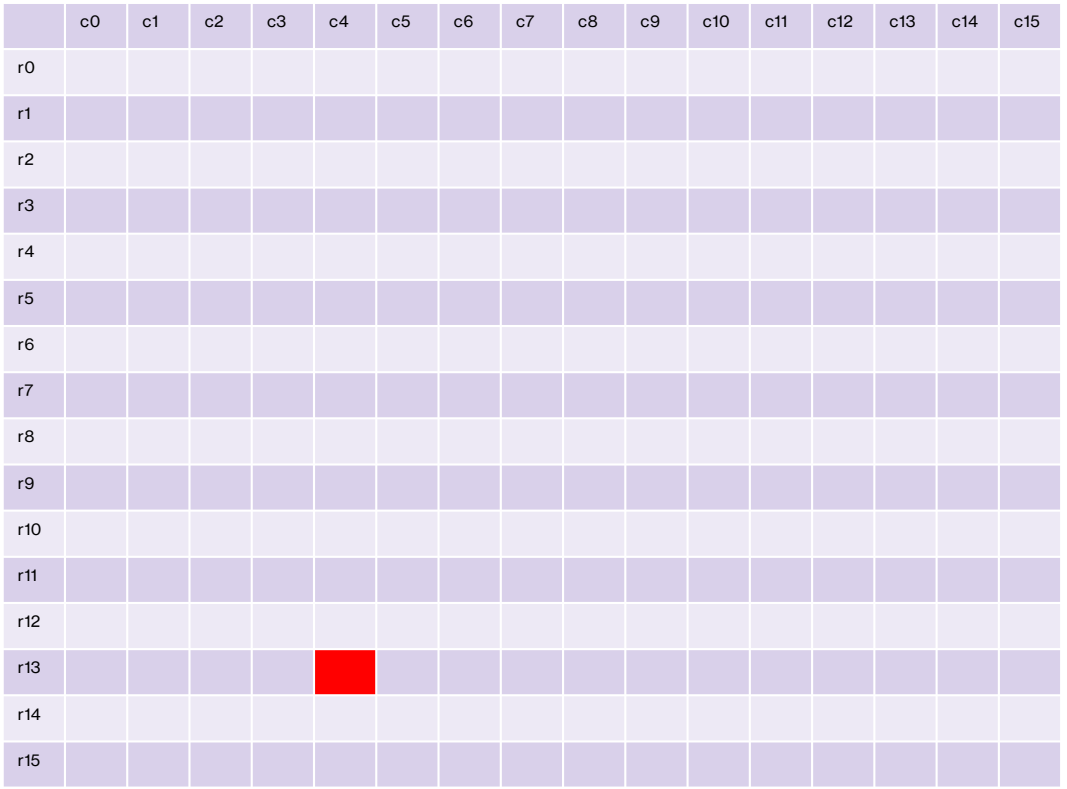
What would be this memory address in binary?

<8-bit memory address>

4 bits for row 4 bits for column

16 columns

16 rows



Memory implemented

We can access a given by via a row index and a column index

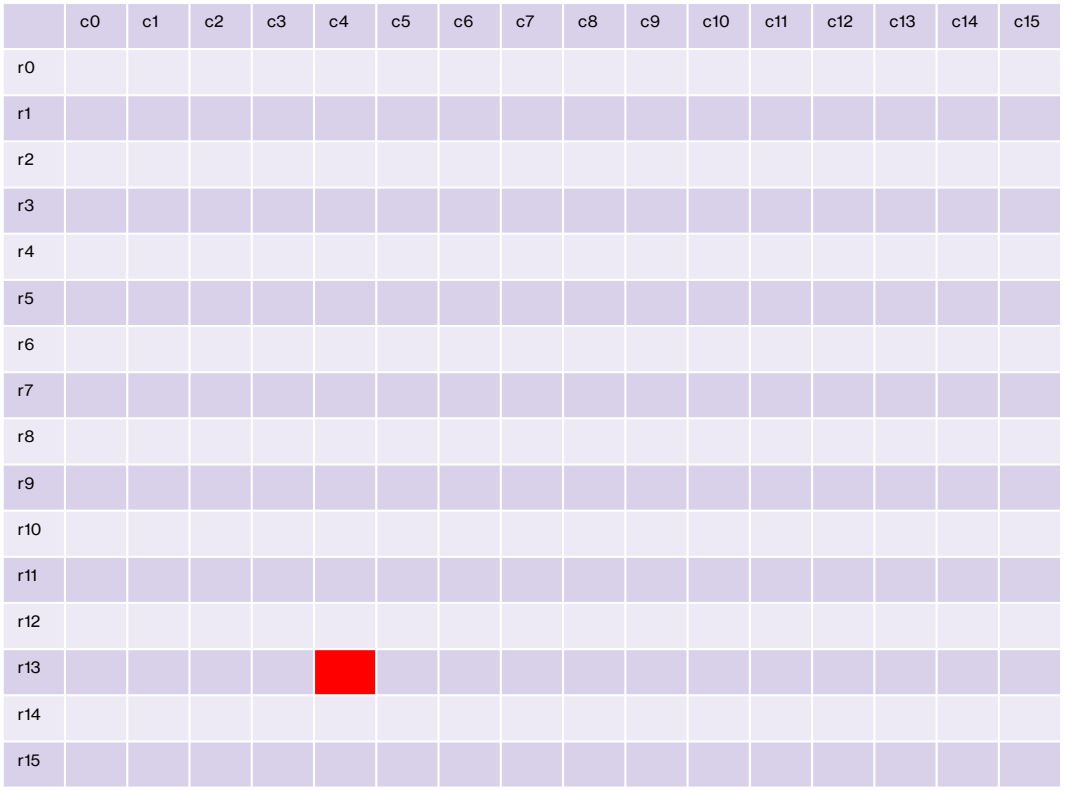
Note we use unsigned numbers

1101 0100

row column

16 columns

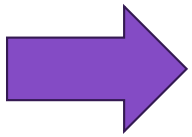
16 rows

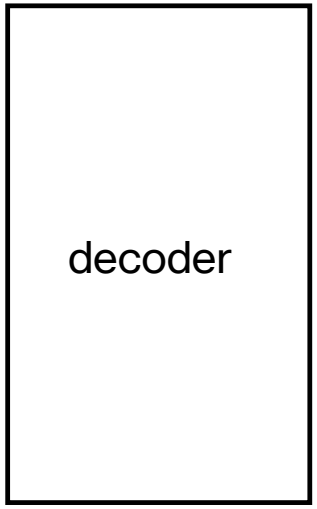


Quick aside

4-bit decoder

4 bits

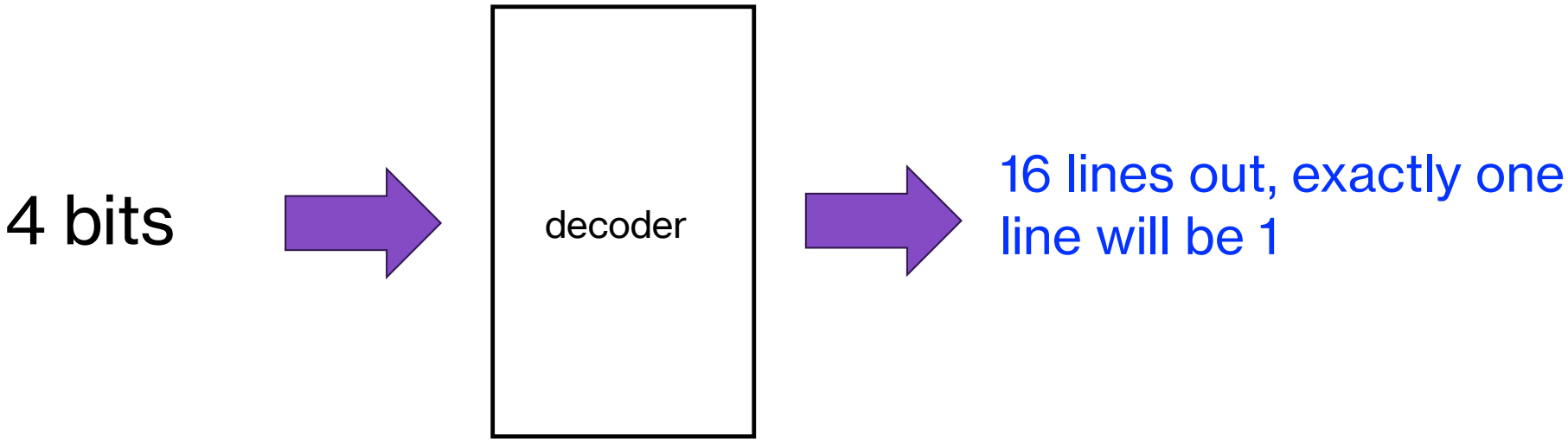




How many lines out?

Quick aside

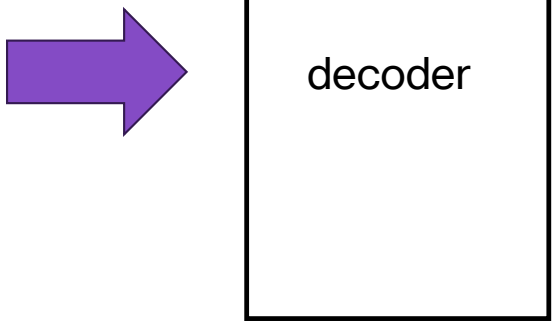
4-bit decoder



Quick aside: multiplexer

16:1 multiplexer

4 bits



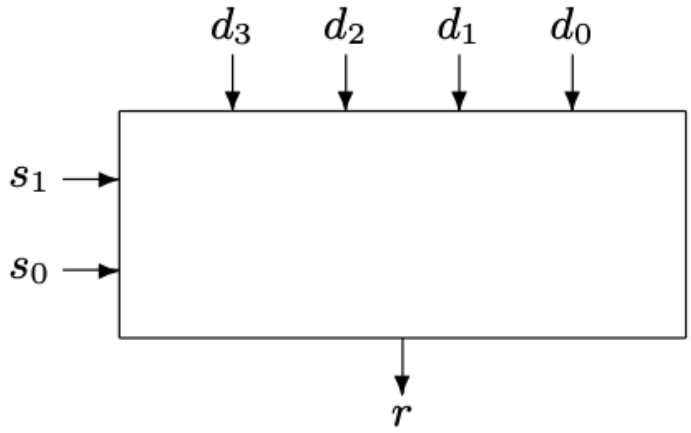
How many lines out?

Quick aside: multiplexer

4:1 multiplexer

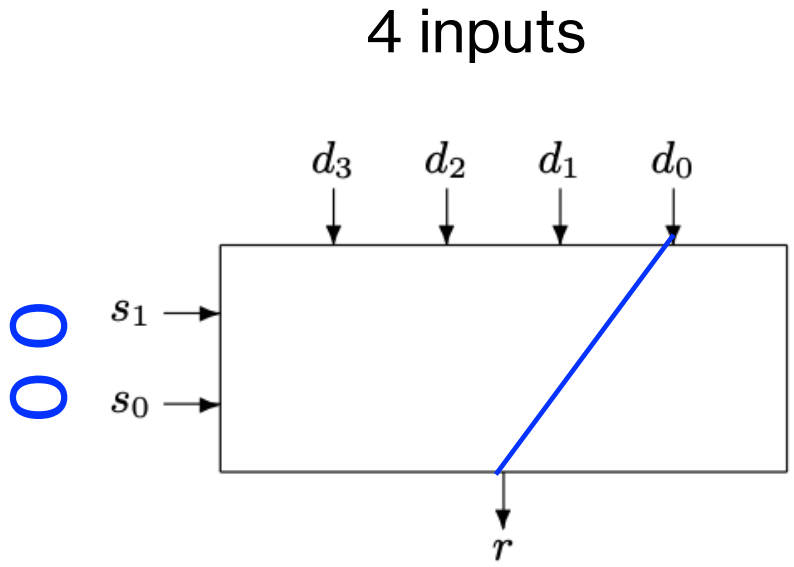
4 inputs

2 inputs to "select" which line



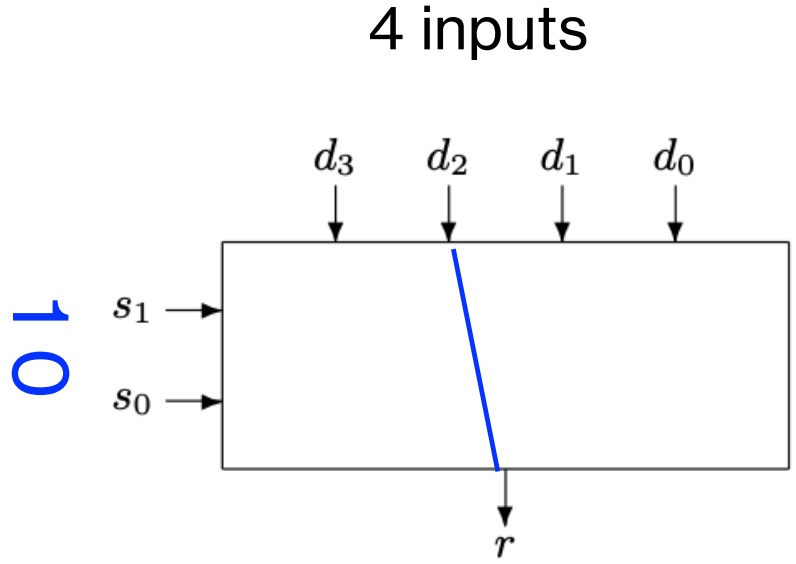
Quick aside: multiplexer

4:1 multiplexer



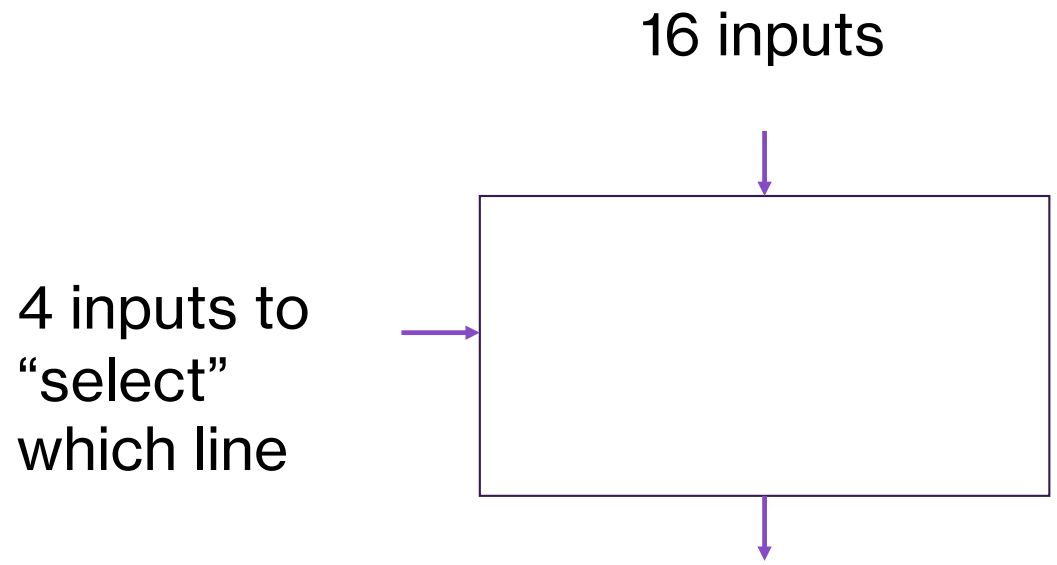
Quick aside: multiplexer

4:1 multiplexer



Quick aside: multiplexer

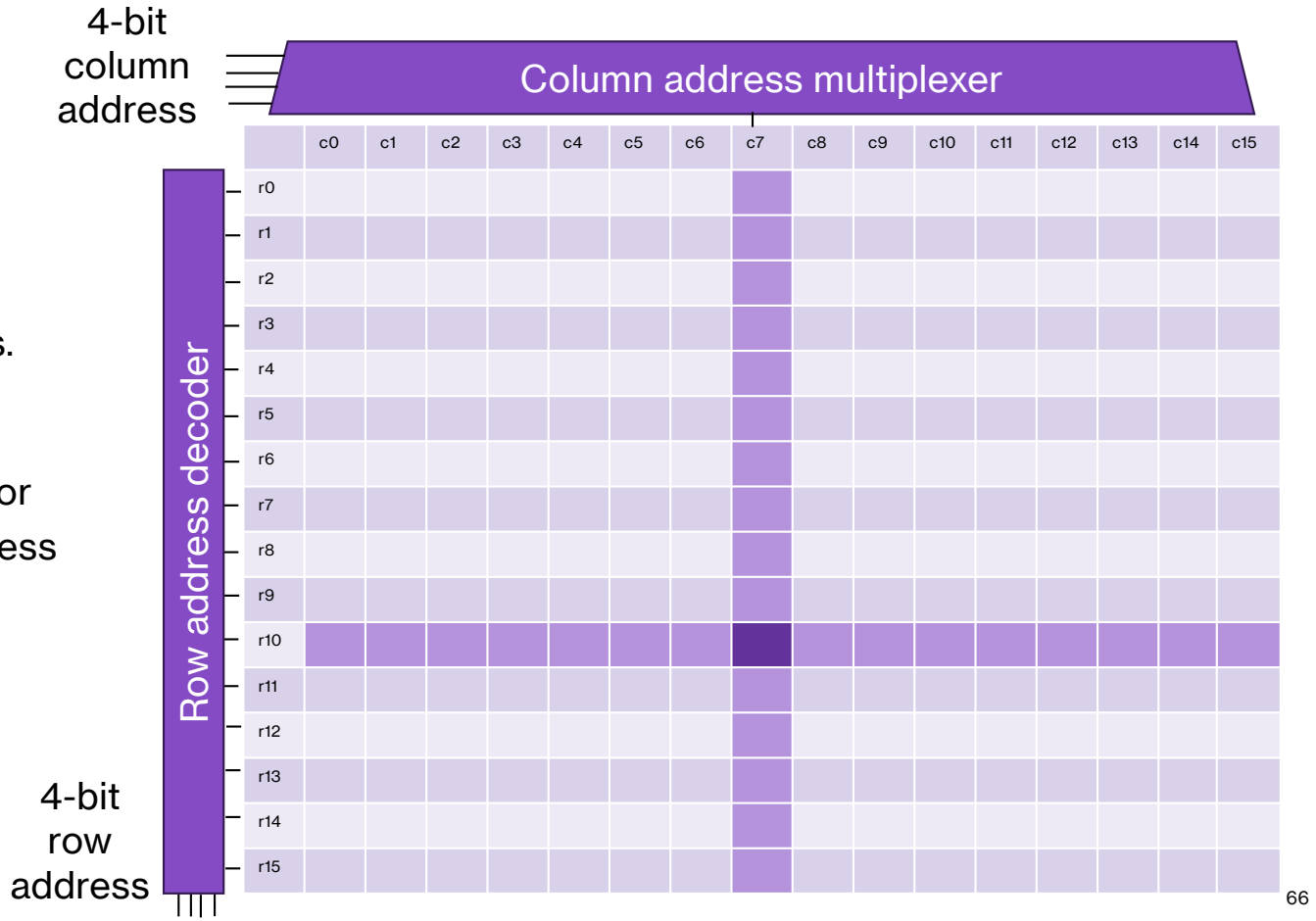
16:1 multiplexer



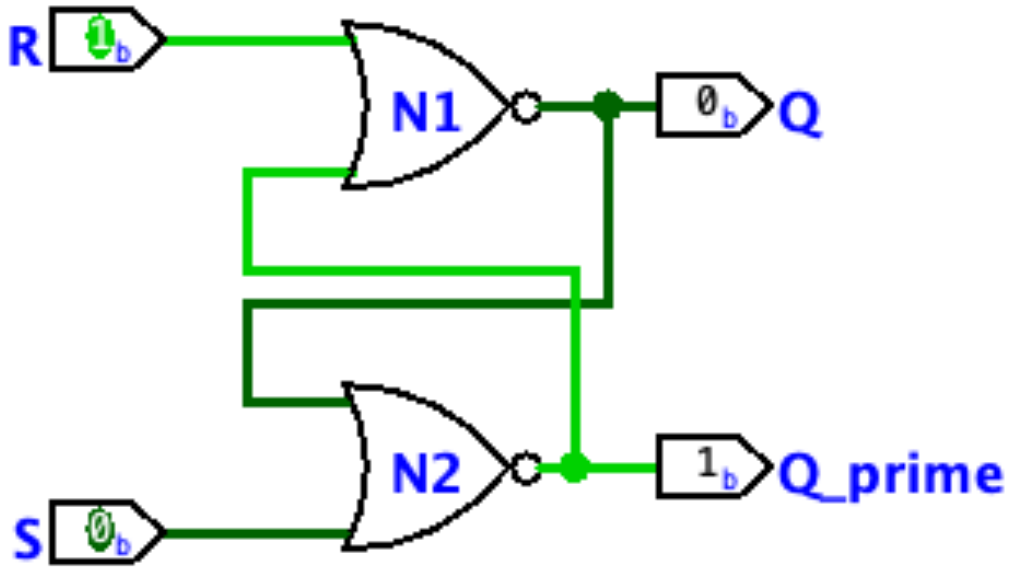
From addresses to actual memory cells

The decoder will be responsible for taking the 4-bits of the row address and mapping it to one of the 16 rows.

The multiplexer will be responsible for taking the 4-bits of the column address as selection lines and to map to a single column.



How do we store bits?

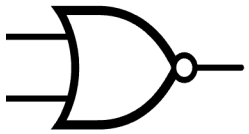


Anything unusual about this circuit?

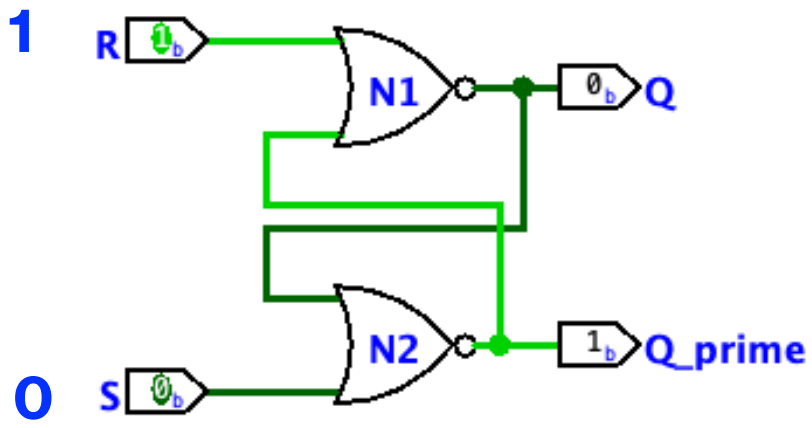
Case I: $R = 1, S = 0$

What will be the output?

Case	Input		Output	
	S	R	Q	Q'
IV	0	0		
I	0	1		
II	1	0		
III	1	1		



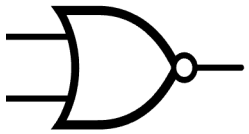
X	Y	X ⊽ Y
0	0	1
0	1	0
1	0	0
1	1	0



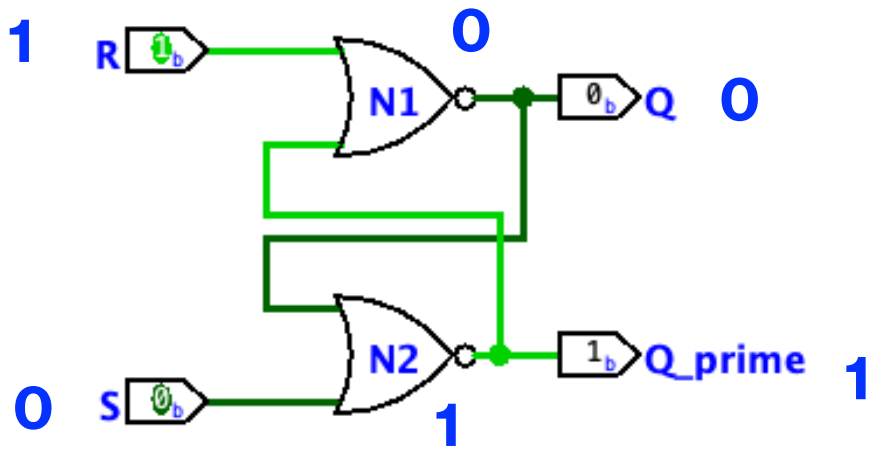
Case I: $R = 1, S = 0$

What will be the output?

Case	Input		Output	
	S	R	Q	Q'
IV	0	0		
I	0	1	0	1
II	1	0		
III	1	1		



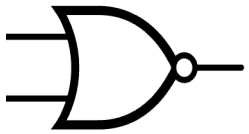
X	Y	X ⊽ Y
0	0	1
0	1	0
1	0	0
1	1	0



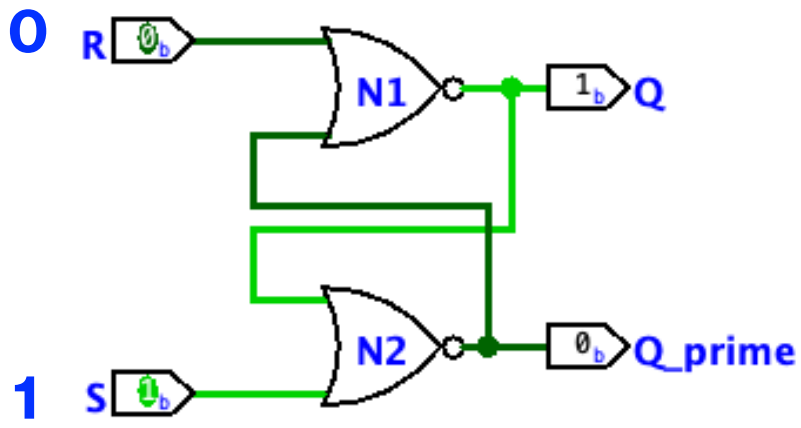
Case II: $R = 0, S = 1$

What will be the output?

Case	Input		Output	
	S	R	Q	Q'
IV	0	0		
I	0	1	0	1
II	1	0		
III	1	1		



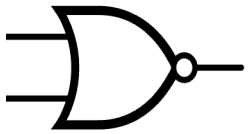
X	Y	X ⊽ Y
0	0	1
0	1	0
1	0	0
1	1	0



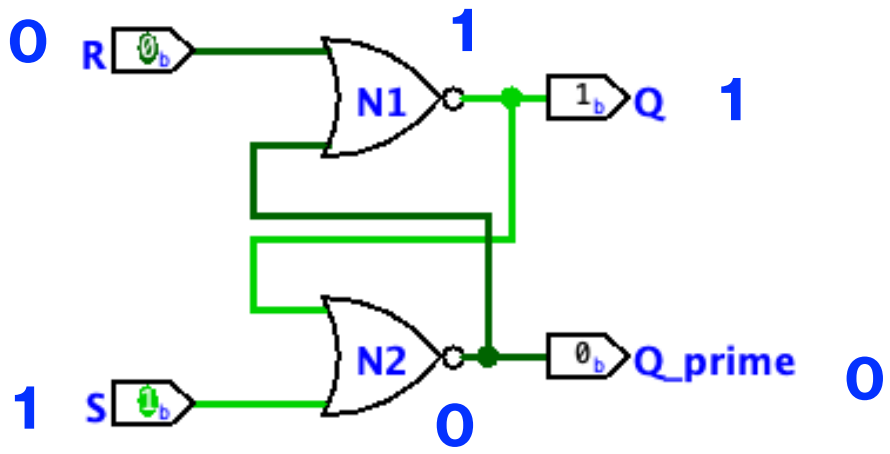
Case II: $R = 0, S = 1$

What will be the output?

Case	Input		Output	
	S	R	Q	Q'
IV	0	0		
I	0	1	0	1
II	1	0	1	0
III	1	1		



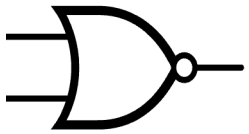
X	Y	X ⊽ Y
0	0	1
0	1	0
1	0	0
1	1	0



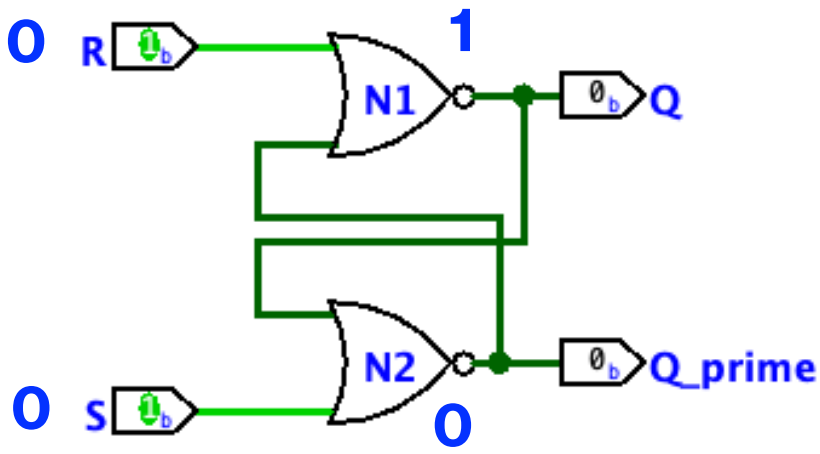
Case IV: $R = 0, S = 0$

What will be the output?

Case	Input		Output	
	S	R	Q	Q'
IV	0	0		
I	0	1	0	1
II	1	0	1	0
III	1	1		



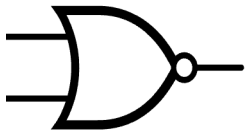
X	Y	X ⊽ Y
0	0	1
0	1	0
1	0	0
1	1	0



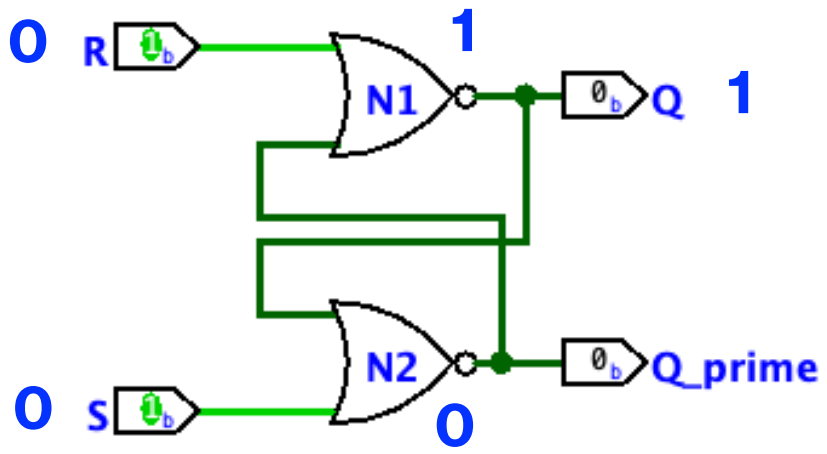
Case IV: $R = 0, S = 0$

What will be the output?

Case	Input		Output	
	S	R	Q	Q'
IV	0	0		
I	0	1	0	1
II	1	0	1	0
III	1	1		



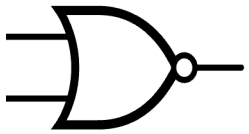
X	Y	X ⊽ Y
0	0	1
0	1	0
1	0	0
1	1	0



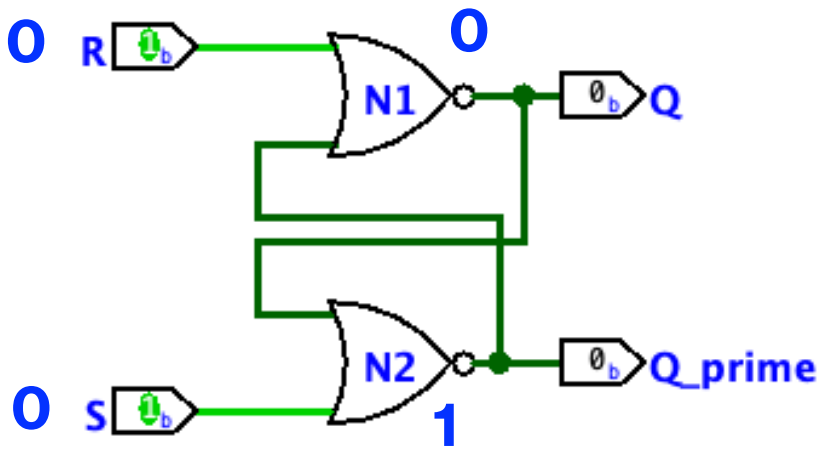
Case IV: $R = 0, S = 0$

What will be the output?

Case	Input		Output	
	S	R	Q	Q'
IV	0	0		
I	0	1	0	1
II	1	0	1	0
III	1	1		



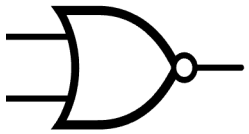
X	Y	X ⊽ Y
0	0	1
0	1	0
1	0	0
1	1	0



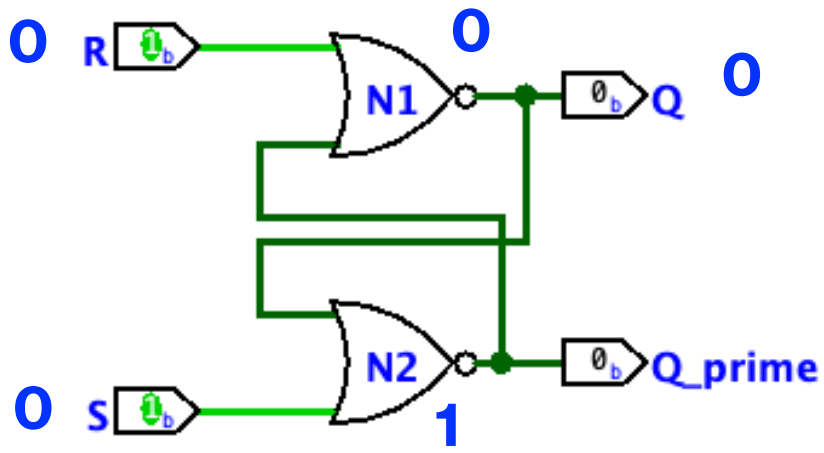
Case IV: $R = 0, S = 0$

What will be the output?

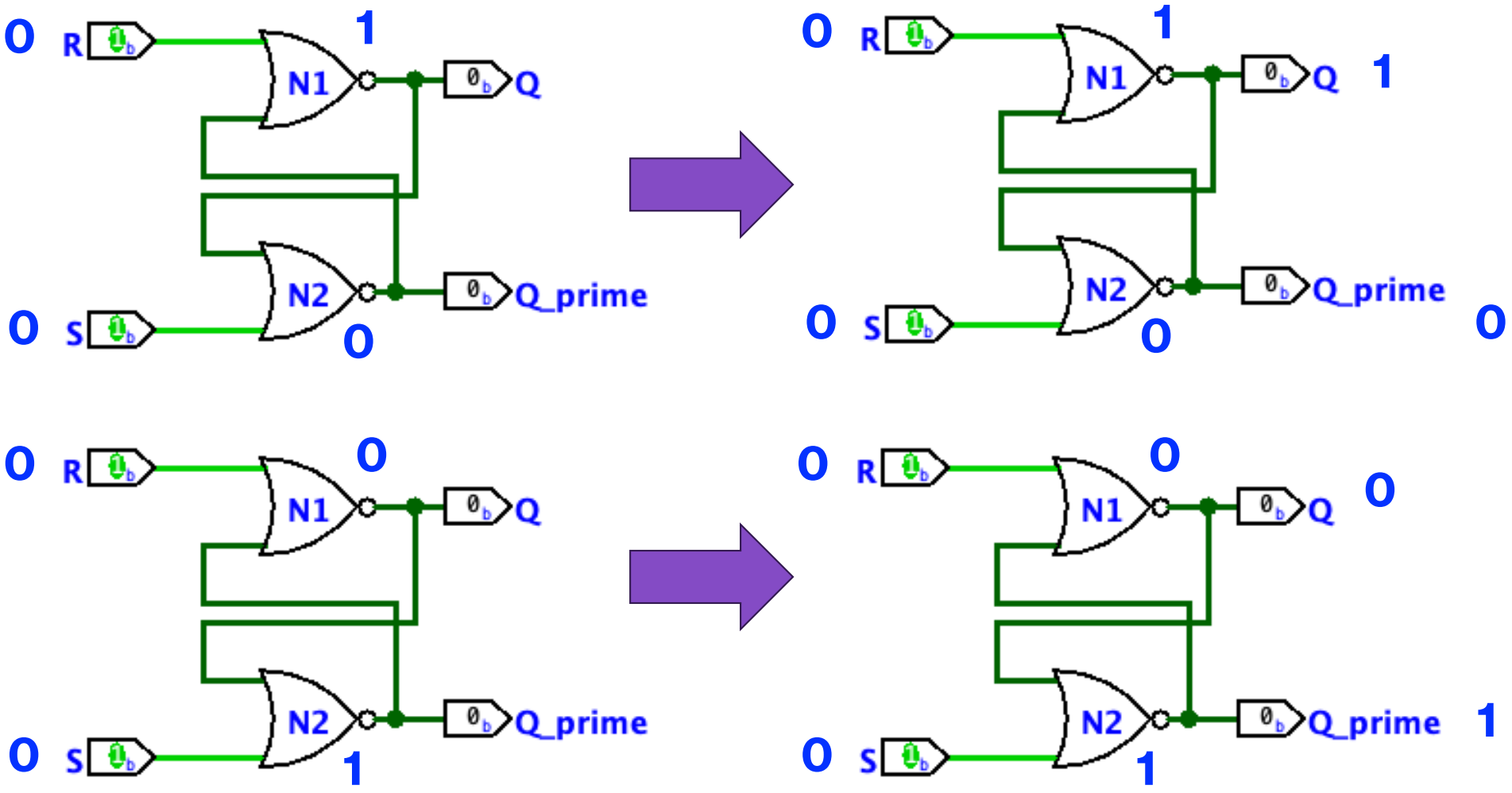
Case	Input		Output	
	S	R	Q	Q'
IV	0	0		
I	0	1	0	1
II	1	0	1	0
III	1	1		



X	Y	X ⊽ Y
0	0	1
0	1	0
1	0	0
1	1	0



Case IV: $R = 0, S = 0$


0
R
1
N1
Q
1
0
N2
Q_prime
0
S
0
0
R
1
N1
Q
1
N2
Q_prime
0
S
0
0
R
0
N1
Q
0
1
N2
Q_prime
1
S
0
0
R
0
N1
Q
0
N2
Q_prime
1
S
1

SR-latch

Case	Input		Output	
	S	R	Q	Q'
IV	0	0	$Q\_previous$	$Q'\_previous$
I	0	1	0	1
II	1	0	1	0
III	1	1		

Hold current state

Reset: Q = 0

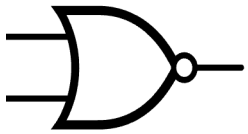
Set: Q = 1

messy... not allowed

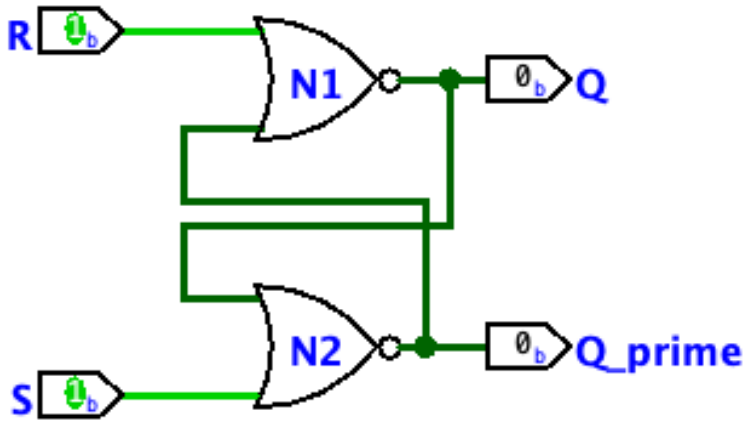
Case III: $R = 1, S = 1$

What will be the output?

Case	Input		Output	
	S	R	Q	Q'
IV	0	0	$Q\_previous$	$Q'\_previous$
I	0	1	0	1
II	1	0	1	0
III	1	1		



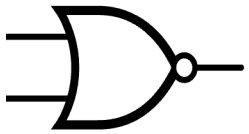
X	Y	X ⊽ Y
0	0	1
0	1	0
1	0	0
1	1	0



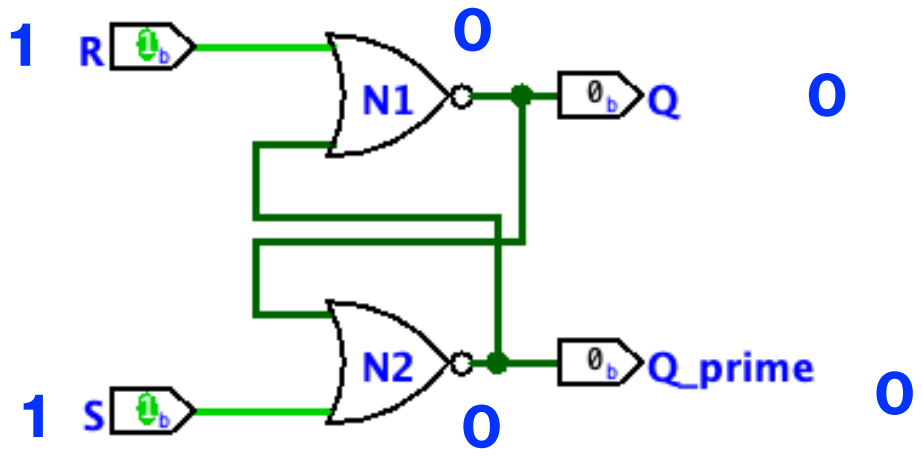
Case III: $R = 1, S = 1$

What will be the output?

Case	Input		Output	
	S	R	Q	Q'
IV	0	0	$Q\_previous$	$Q'\_previous$
I	0	1	0	1
II	1	0	1	0
III	1	1	0	0



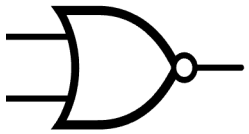
X	Y	X ⊽ Y
0	0	1
0	1	0
1	0	0
1	1	0



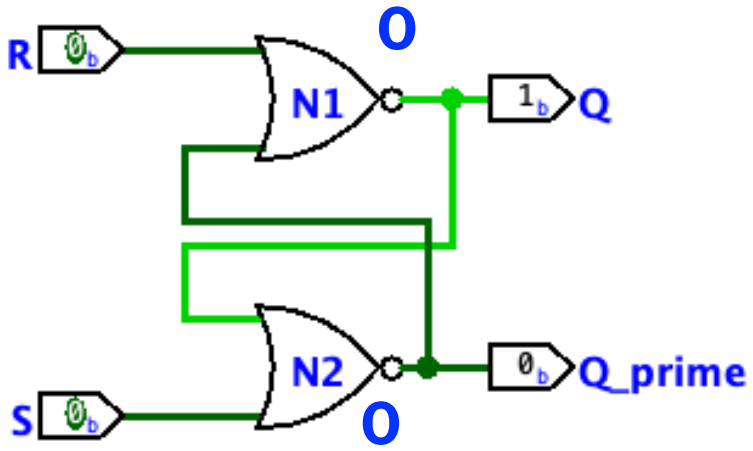
Case I: $R = 0, S = 0$

What will be the output?

Case	Input		Output	
	S	R	Q	Q'
IV	0	0		
I	0	1	0	1
II	1	0	1	0
III	1	1	0	0

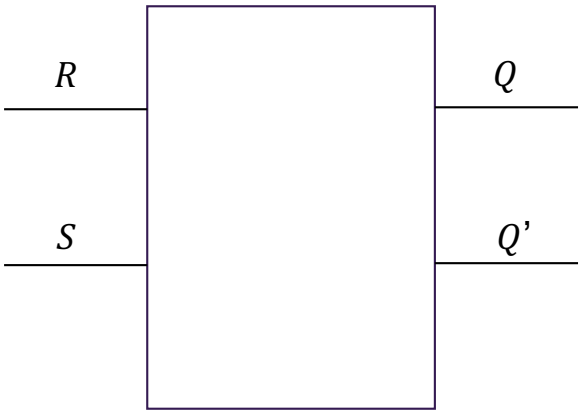


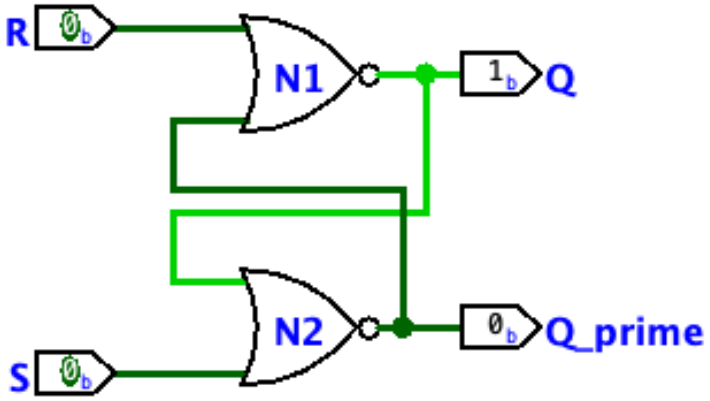
X	Y	X ⊽ Y
0	0	1
0	1	0
1	0	0
1	1	0



SR-latch - Abstraction

- As an application of abstraction, instead of seeing the full circuit on the right, you might encounter the following symbol for SR-latches:





Latches for memory

Latches can store 1 bit

Other latches exists (e.g., D latch)

Other circuits that store 1 bit (e.g., flip-flops)

Can combine these to make memory (registers, RAM)

Latches for memory

Latches can store 1 bit

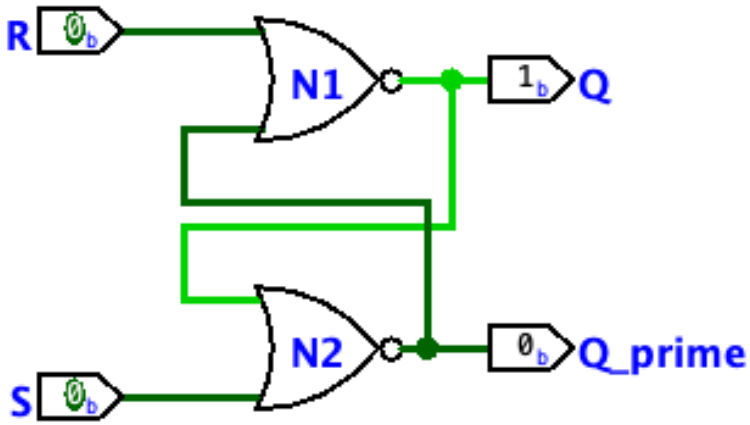
Other latches exists (e.g., D latch)

Other circuits that store 1 bit (e.g., flip-flops)

Can combine these to make memory (registers, RAM)

Why do we have hard drives?

Latches for memory



When power goes off, we lose what we stored!