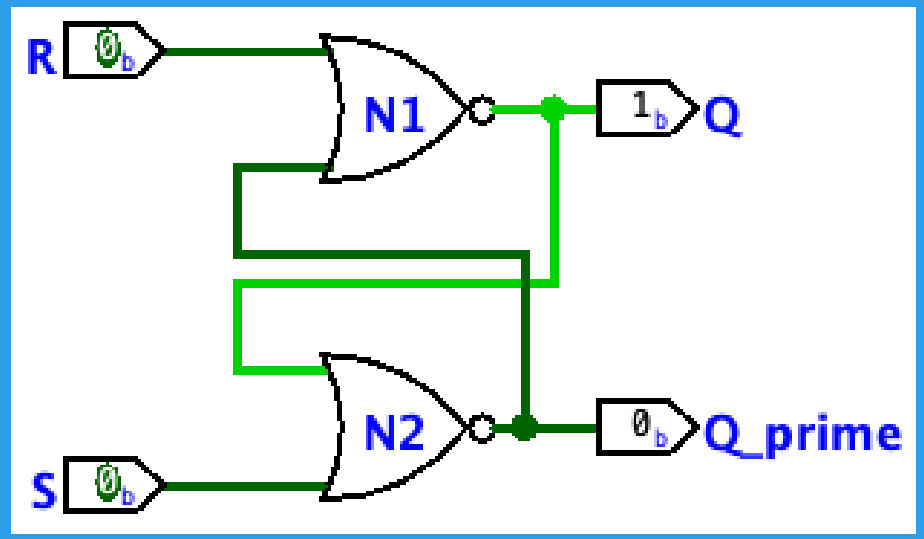
Computer Systems



Sequential Circuits

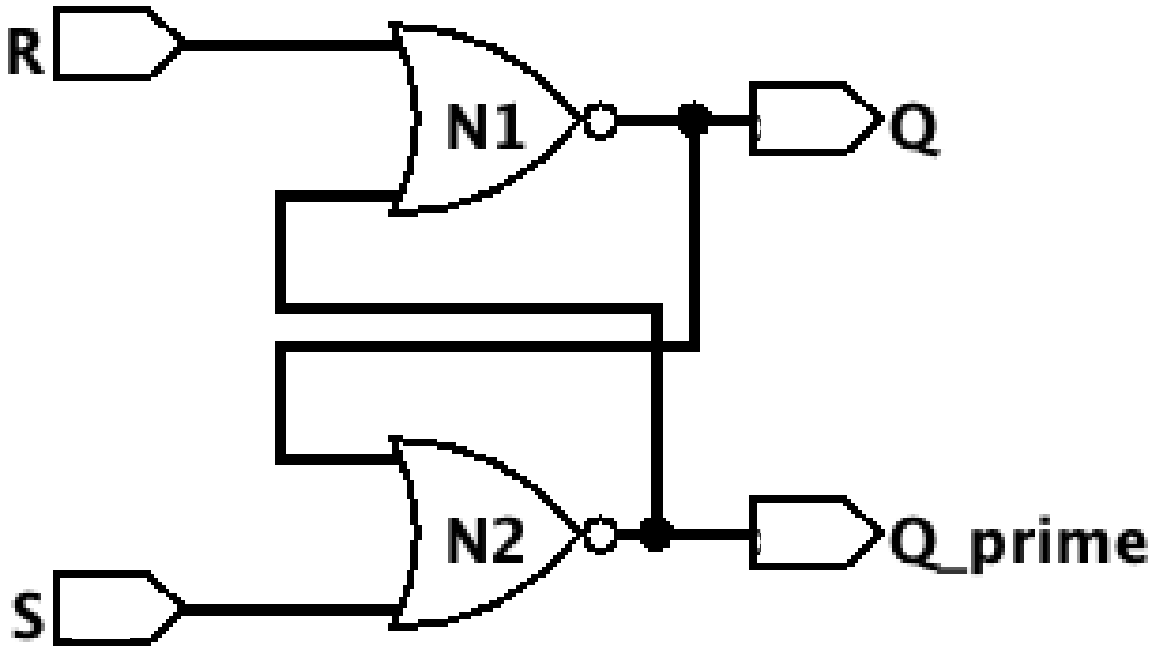
CS51 – Fall 2026

Sequential circuits

- Last lecture, all circuits we saw were **combinational**: their output depends on the present inputs and there is no notion of memory of past inputs.
- Beyond combinational circuits, there are also **sequential circuits**.
- Sequential circuits not only depend on current input but have memory components that allow them to utilize prior inputs, too.
- Storing data into memory is called **writing** while retrieving data from memory is called **reading**.

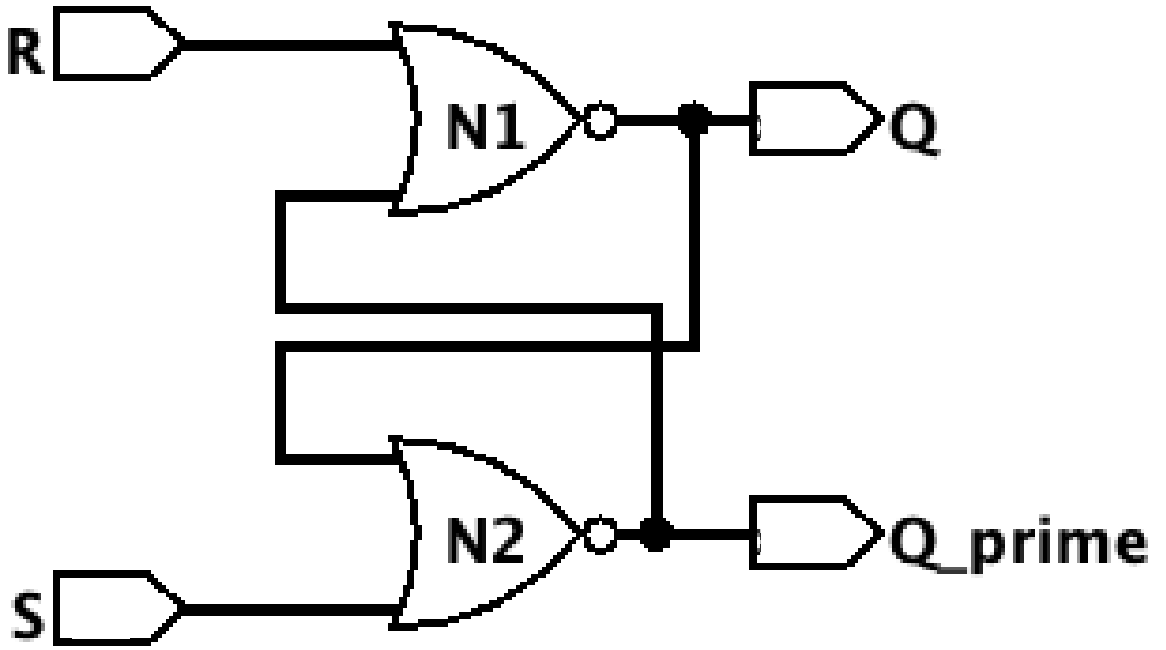
SR-Latches

Think



Anything unusual about this circuit?

SR-latch

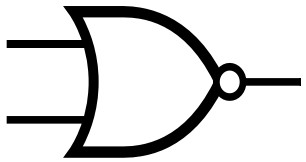


There are two cross-coupled NOR gates

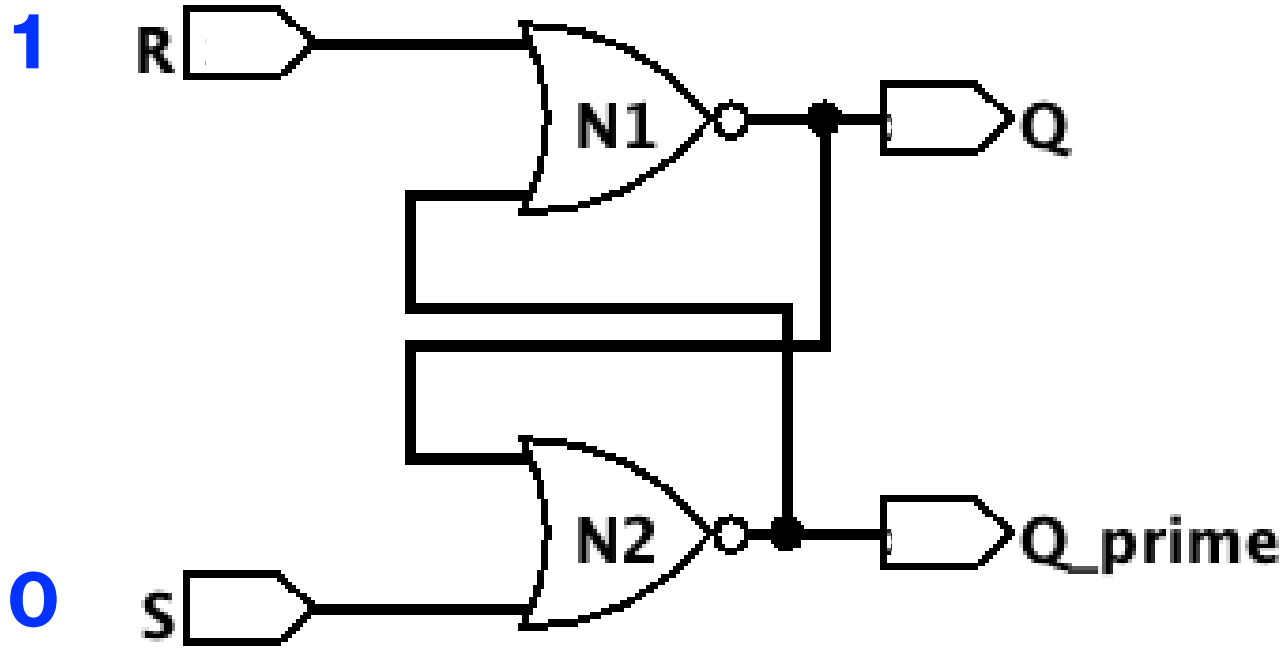
Think: $R = 1, S = 0$

What will be the output?

Input		Output	
S	R	Q	Q'
0	0		
0	1		
1	0		
1	1		



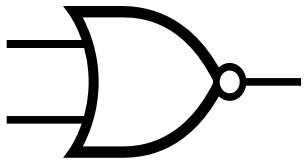
X	Y	X ⊽ Y
0	0	1
0	1	0
1	0	0
1	1	0



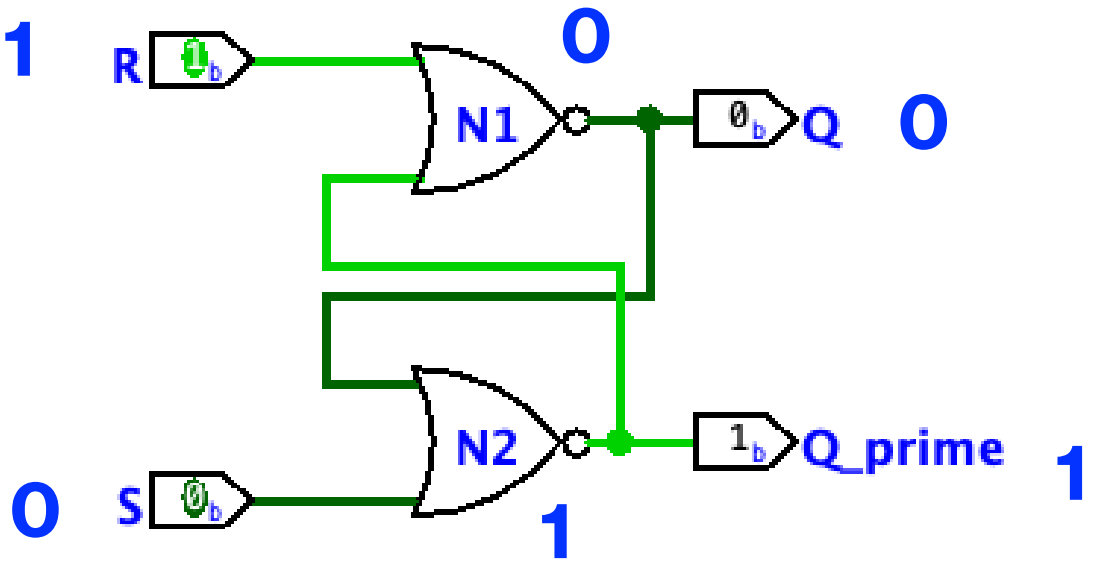
$$R = 1, S = 0$$

What will be the output?

Input		Output	
S	R	Q	Q'
0	0		
0	1	0	1
1	0		
1	1		



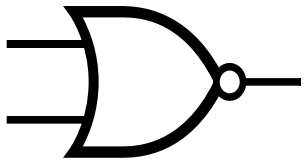
X	Y	X ⊽ Y
0	0	1
0	1	0
1	0	0
1	1	0



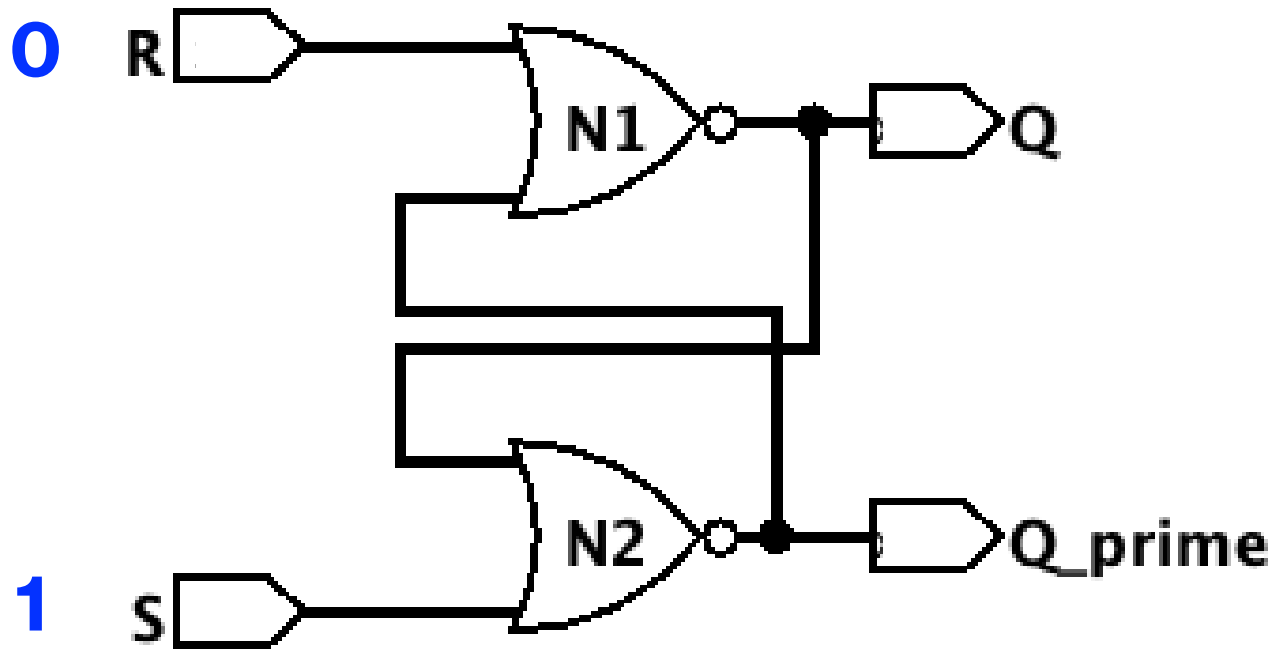
Think: $R = 0, S = 1$

What will be the output?

Input		Output	
S	R	Q	Q'
0	0		
0	1	0	1
1	0		
1	1		



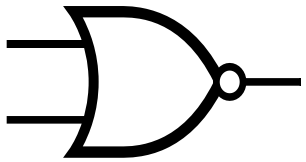
X	Y	X ⊽ Y
0	0	1
0	1	0
1	0	0
1	1	0



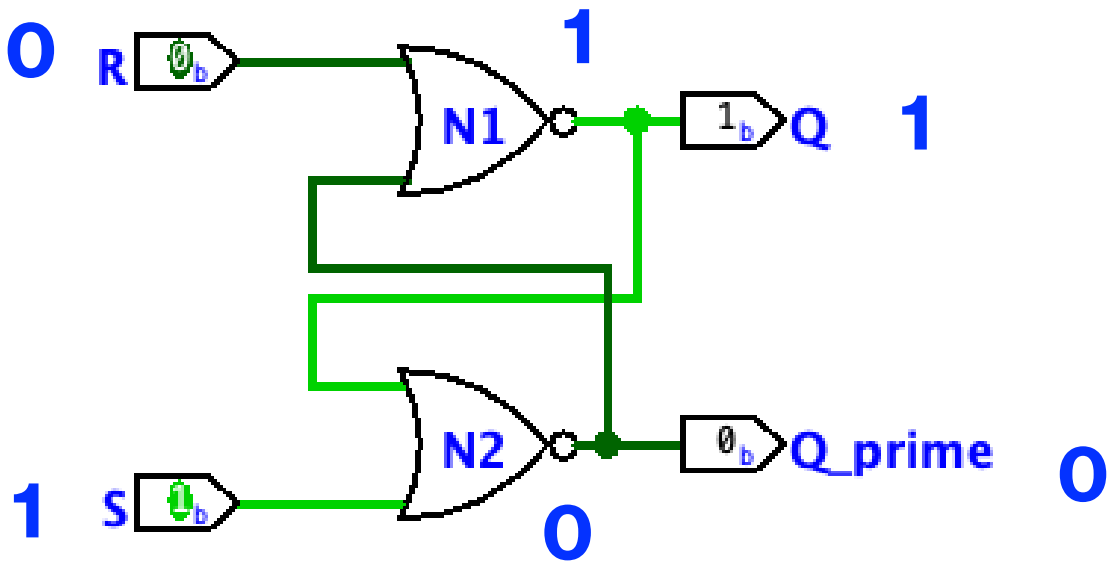
$$R = 0, S = 1$$

What will be the output?

Input		Output	
S	R	Q	Q'
0	0		
0	1	0	1
1	0	1	0
1	1		



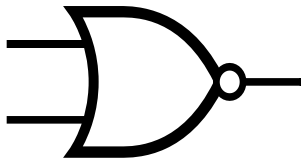
X	Y	X ⊽ Y
0	0	1
0	1	0
1	0	0
1	1	0



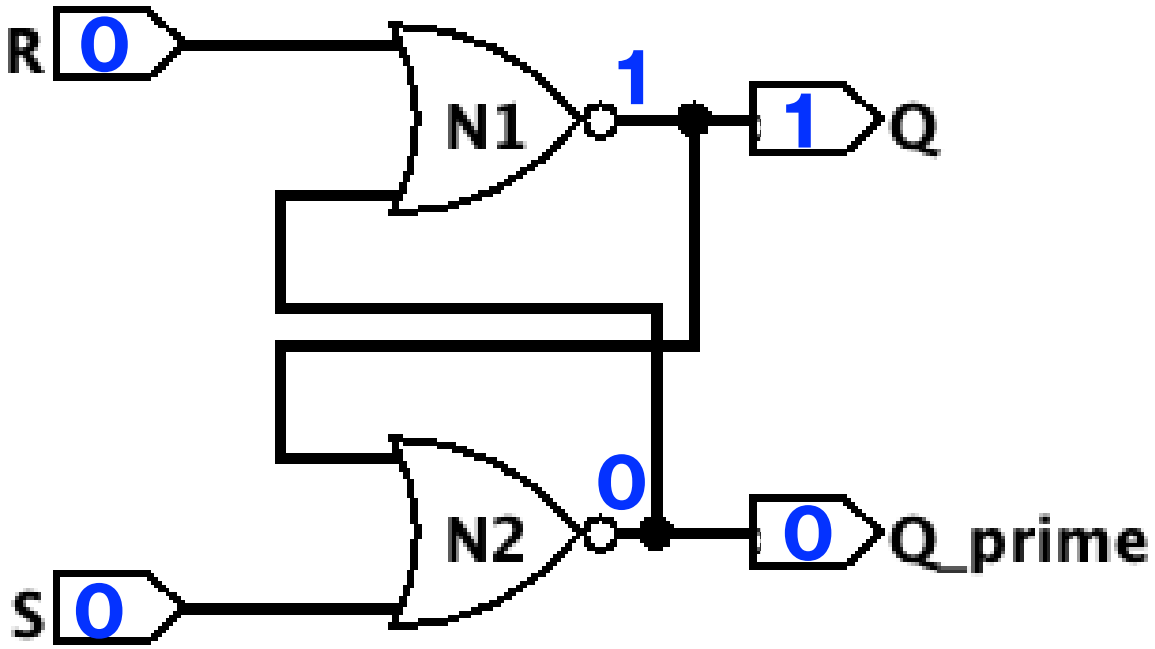
Think: $R = 0, S = 0, \mathrm{N1} = 1, \mathrm{N2} = 0$

What will be the output?

Input		Output	
S	R	Q	Q'
0	0		
0	1	0	1
1	0	1	0
1	1		



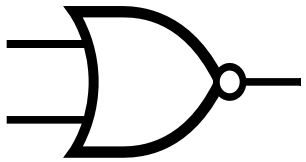
X	Y	X ⊽ Y
0	0	1
0	1	0
1	0	0
1	1	0



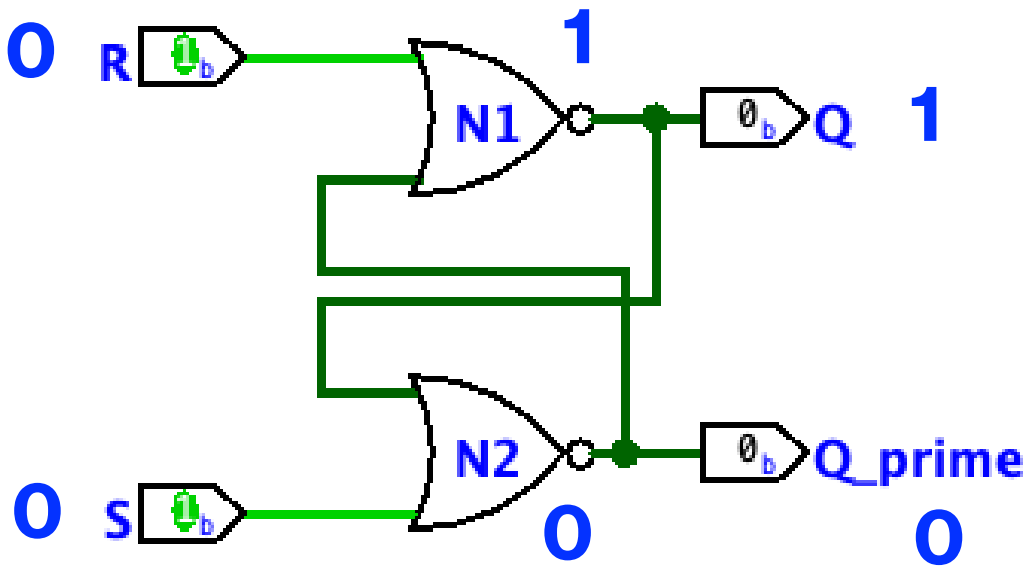
$$R = 0, S = 0, \mathrm{N1} = 1, \mathrm{N2} = 0$$

What will be the output?

Input		Output	
S	R	Q	Q'
0	0		
0	1	0	1
1	0	1	0
1	1		



X	Y	X ⊽ Y
0	0	1
0	1	0
1	0	0
1	1	0

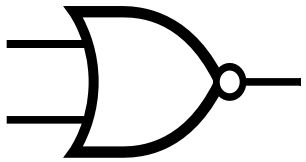


nothing changes, Q remains 1 and Q' remains 0

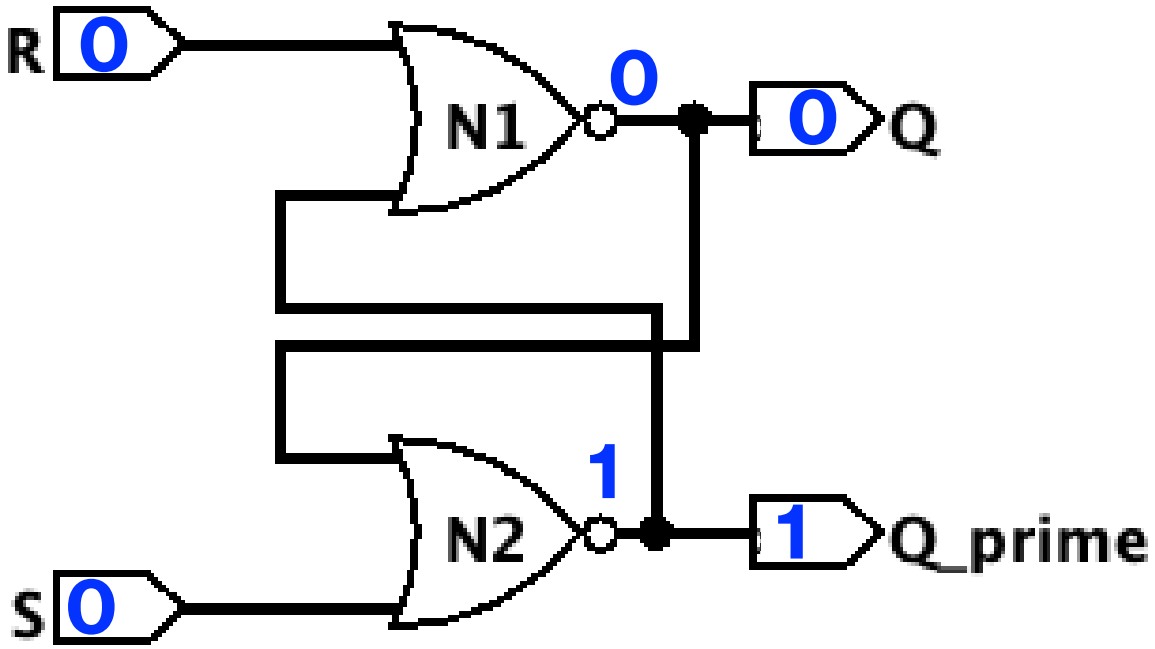
Think: $R = 0, S = 0, \mathrm{N1} = 0, \mathrm{N2} = 1$

What will be the output?

Input		Output	
S	R	Q	Q'
0	0		
0	1	0	1
1	0	1	0
1	1		



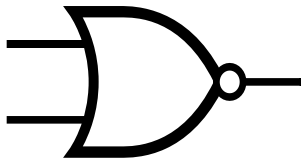
X	Y	X ⊽ Y
0	0	1
0	1	0
1	0	0
1	1	0



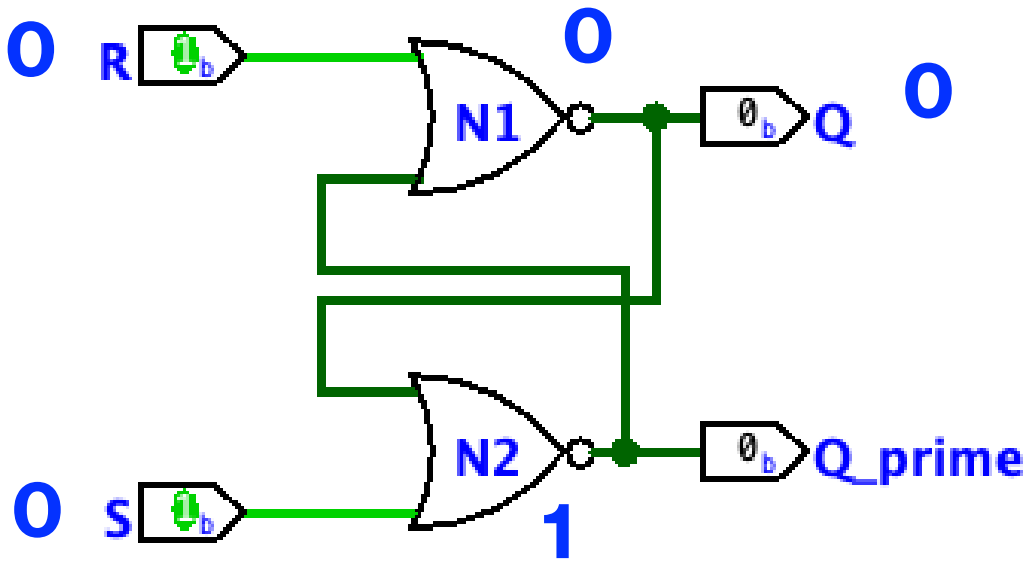
$$R = 0, S = 0, \mathrm{N1} = 0, \mathrm{N2} = 1$$

What will be the output?

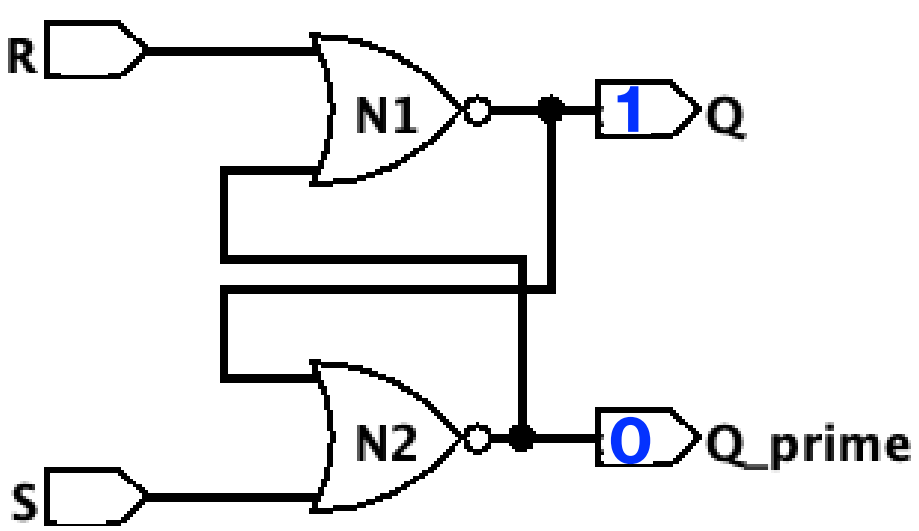
Input		Output	
S	R	Q	Q'
0	0		
0	1	0	1
1	0	1	0
1	1		

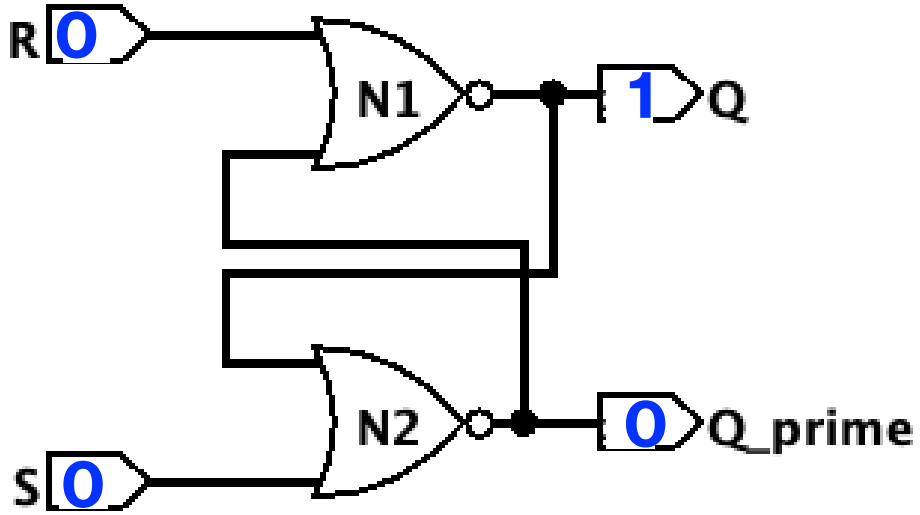


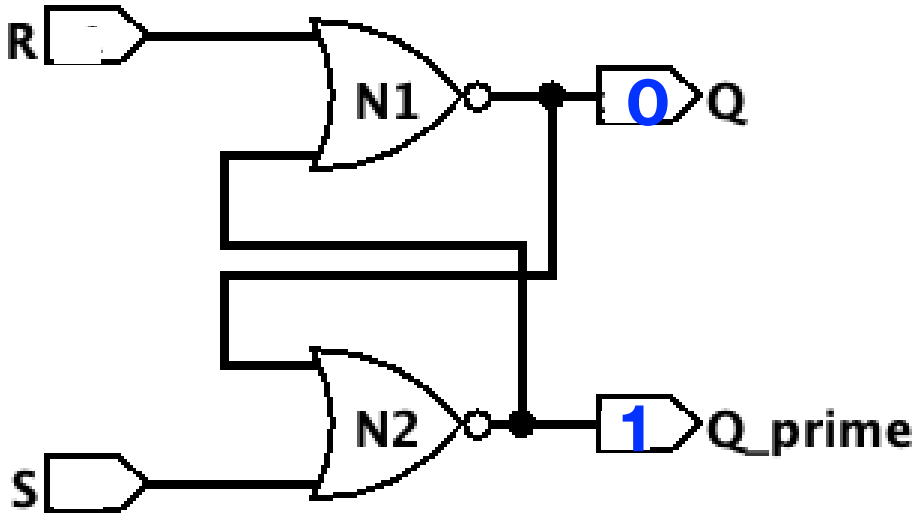
X	Y	X ⊽ Y
0	0	1
0	1	0
1	0	0
1	1	0

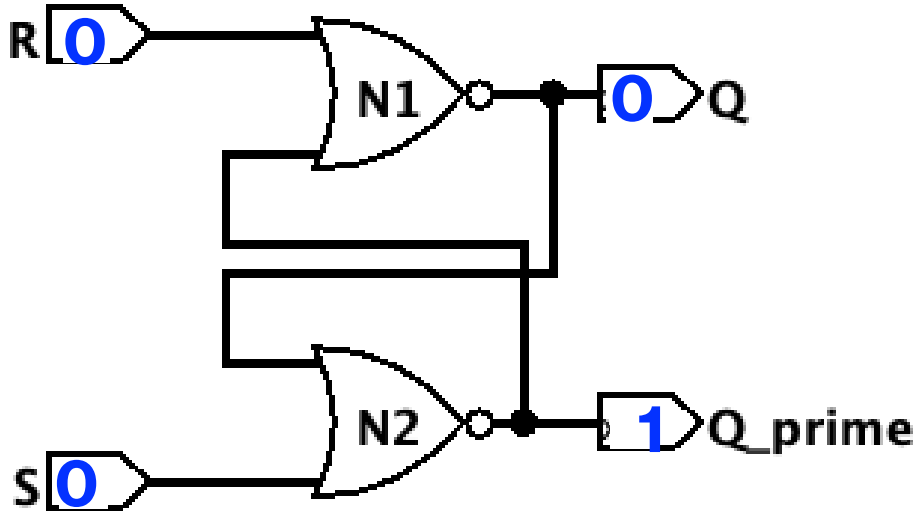


$$R = 0, S = 0$$


R
N1
1
Q
N2
0
Q_prime
S


R
0
N1
1
Q
N2
0
Q_prime
S
0


R
N1
0
Q
N2
1
Q_prime
S


R
0
N1
0
Q
N2
1
Q_prime
S
0

SR-latch

Input		Output	
S	R	Q	Q'
0	0	$Q_previous$	$Q'_previous$
0	1	0	1
1	0	1	0
1	1		

Hold previous state

Reset: Q = 0

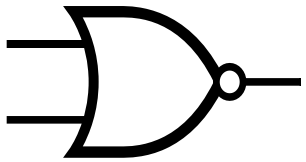
Set: Q = 1

?

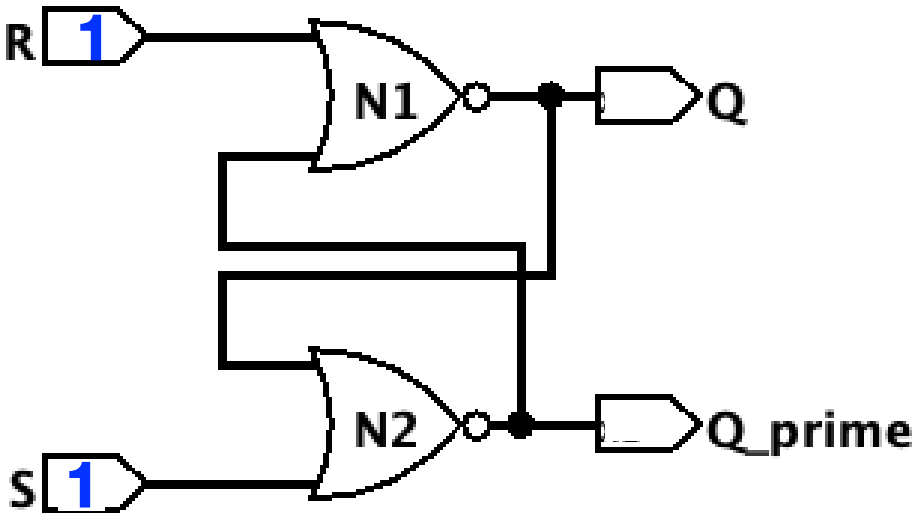
Think: $R = 1, S = 1$

What will be the output?

Input		Output	
S	R	Q	Q'
0	0	$Q_previous$	$Q'_previous$
0	1	0	1
1	0	1	0
1	1		



X	Y	X ⊽ Y
0	0	1
0	1	0
1	0	0
1	1	0

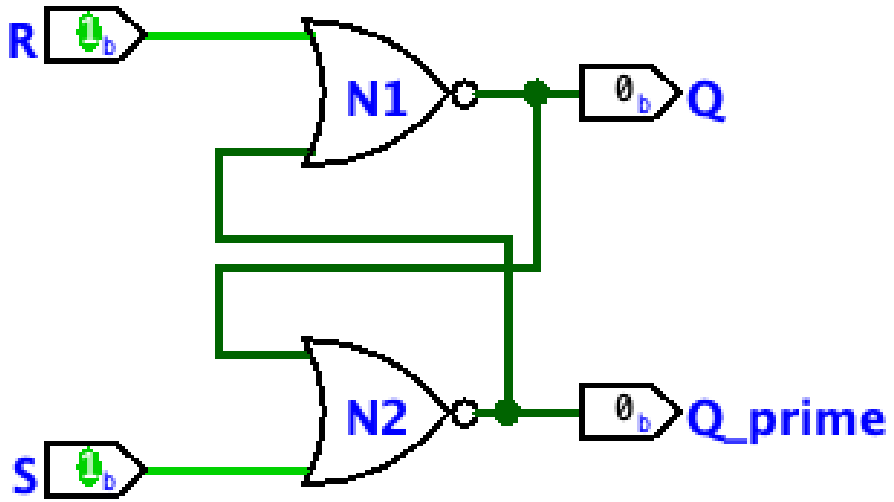


$R = 1, S = 1$

Input		Output	
S	R	Q	Q'
0	0	$Q_previous$	$Q'_previous$
0	1	0	1
1	0	1	0
1	1	0	0

- Both NOR gates N_1 and N_2 receive at least one input being 1 (R and S, respectively), so they can immediately produce 0 for Q and Q' respectively.
- Illegal state

X	Y	X ⊽ Y
0	0	1
0	1	0
1	0	0
1	1	0



From latches to registers

- One latch can store 1 bit of information.
  - There are more advanced circuits that store 1 bit of information, such as D-latches and flip flops.
- If we group together n such circuits that store 1 bit we have a **register**.
- The **width** of the register is the n bits we can store which corresponds to n underlying circuits that each stores 1 bit of information.
- Registers are the key building block of sequential circuits.
- Registers today are 32 or 64 bits and are built directly into a CPU.
- Remember that they store a relatively small number of bits and are extremely fast.

Memory

When you need more memory

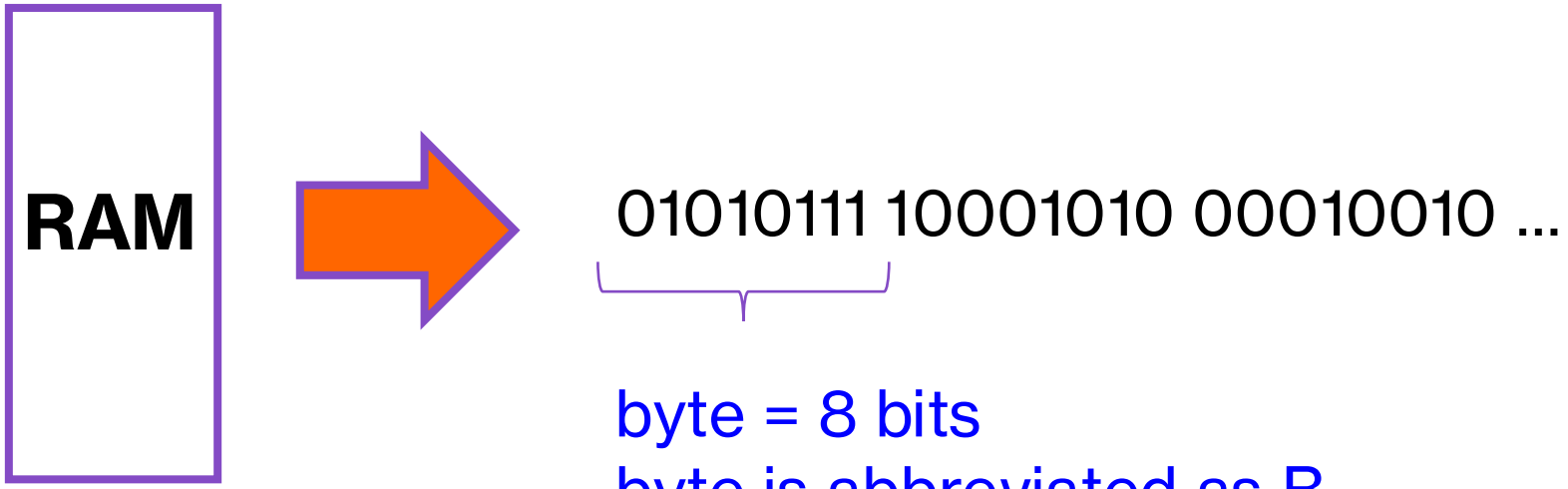
- Sequential circuits like SR-latches have memory components that can store small amounts of data from a single to a relatively small number of bits.
- Computers require a lot more memory and they do so efficiently by utilizing **memory arrays**, two-dimensional arrays of **memory cells**, each storing one bit of information:

0	1	0	1	1	1	0	1
1	1	0	0	0	0	1	1
0	1	1	1	1	0	0	1
1	1	0	1	1	1	1	1
0	1	1	0	0	0	0	1
1	1	1	0	1	1	1	1
1	0	1	1	1	1	1	1
0	1	0	1	1	0	1	0

Units of memory

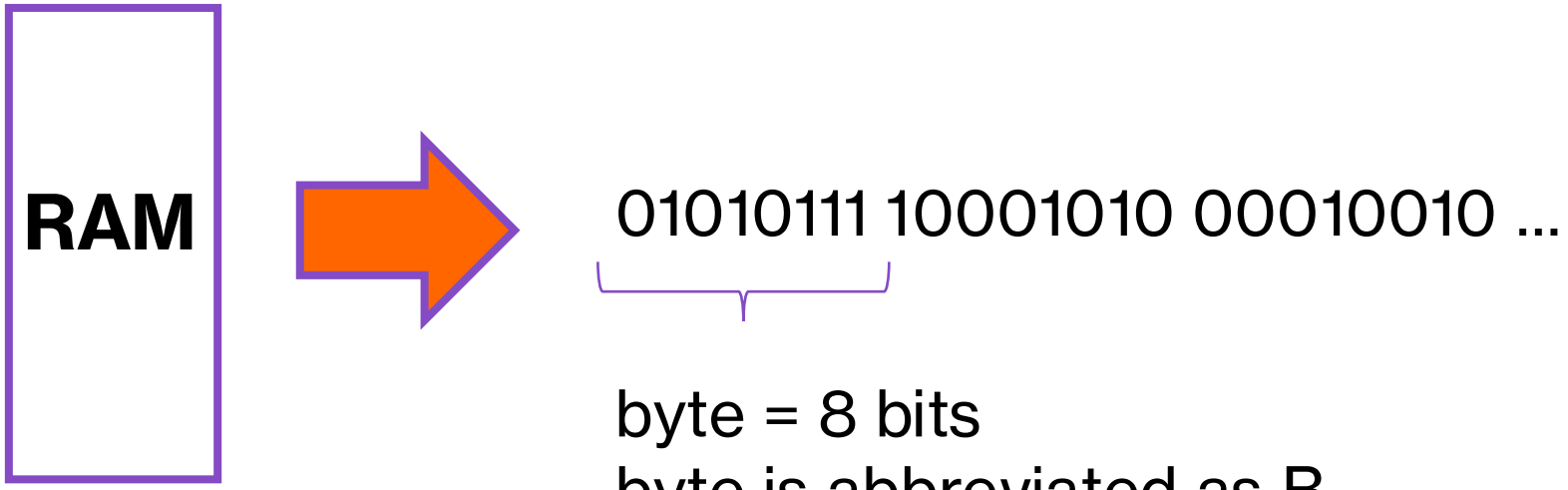
- **Bit**: the smallest unit of memory.
- **Nibble**: 4 bits, half a byte
- **Byte**: 8 bits, the basic unit of measuring memory. Notation is B, e.g., 8B is 8 bytes or 64 bits.
- **Kilobyte**: 10^3 bytes $= 1{,}000$ bytes. Notation is KB.
  - In base 2, 1KiB (kibibyte) = 2^{10} bytes
- **Megabyte**: 10^3 Kilobytes = $1{,}000{,}000$ bytes. Notation is MB.
  - In base 2, 1MiB (mebibyte) = 2^{20} bytes
- **Gigabyte**: 10^3 Megabytes = $1{,}000{,}000{,}000$ bytes. Notation is GB.
  - In base 2, 1GiB (gibibyte) = 2^{30} bytes
- **Terabyte**: 10^3 Gigabytes = $1{,}000{,}000{,}000{,}000$ bytes. Notation is TB.
  - In base 2, 1TiB (tebibyte) = 2^{40} bytes

Memory



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

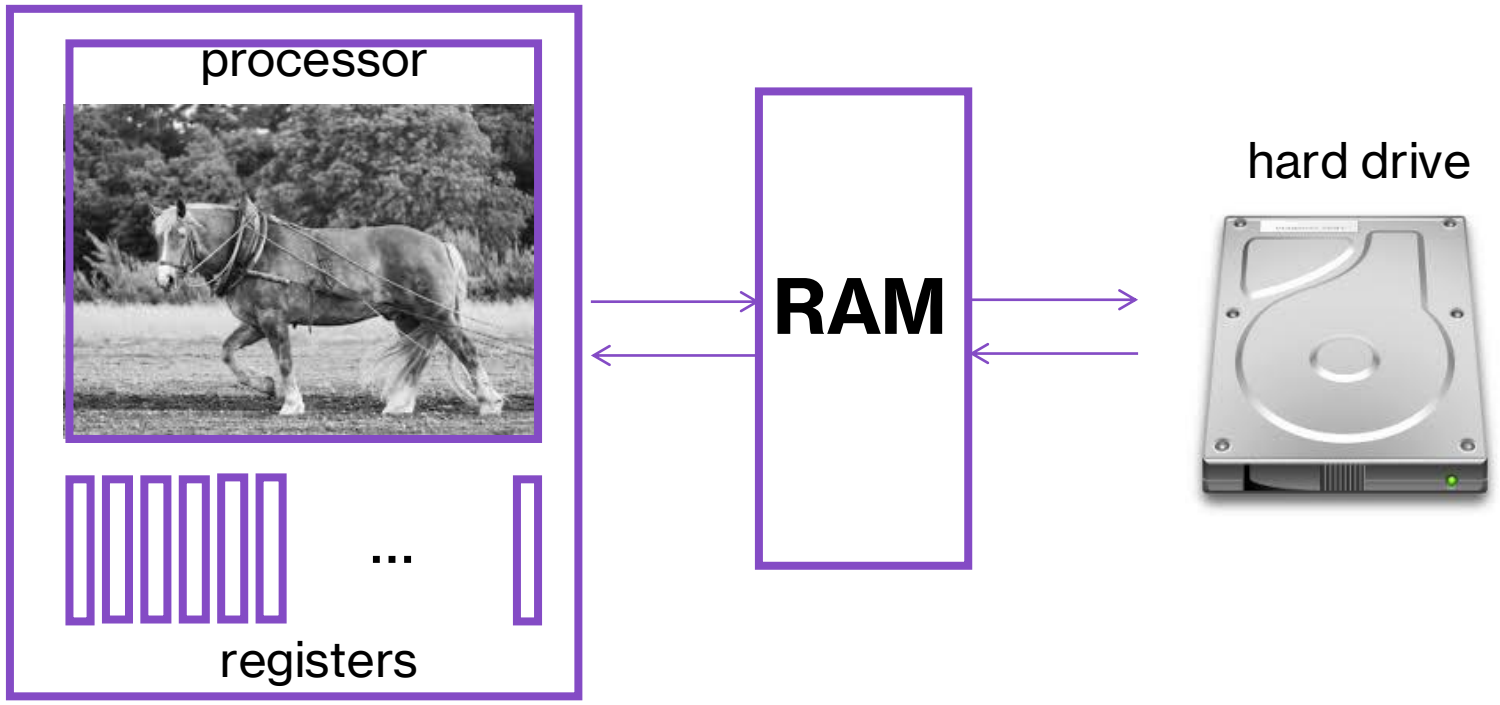
Memory



byte = 8 bits
byte is abbreviated as B

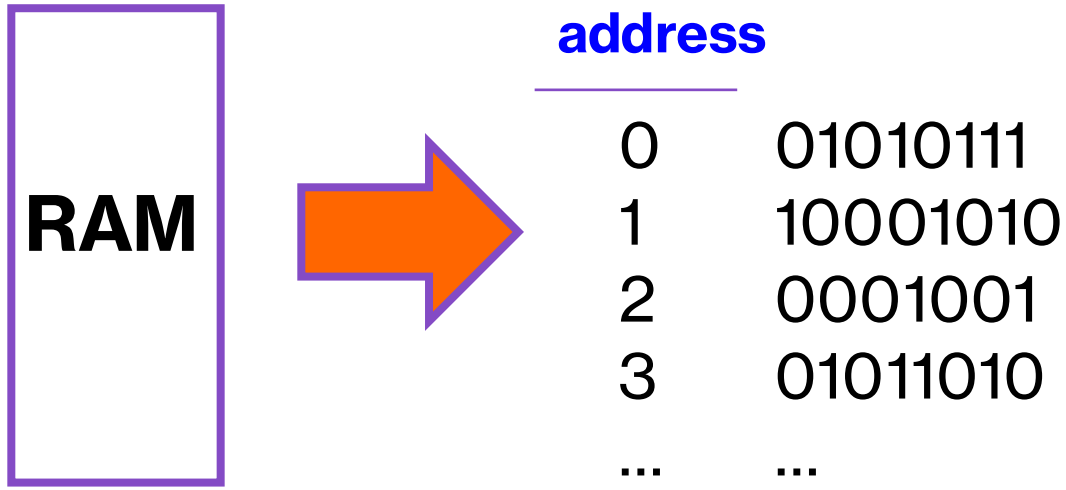
64GB = 64,000,000,000 = 64 billion bytes = 512 billion bits

Memory hierarchy



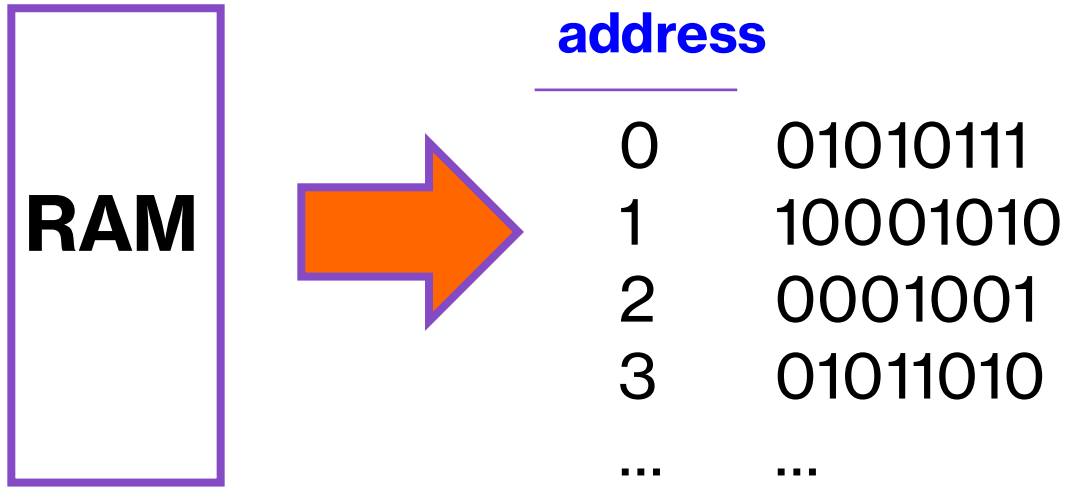
Capacity: bytes Gigabytes Terabytes

Memory



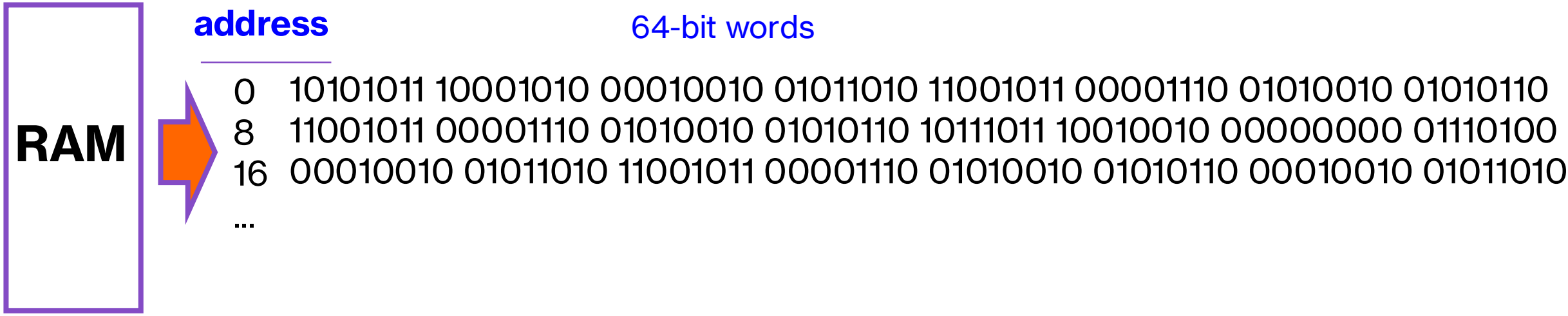
Memory is byte addressable

Memory



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

Memory



Most modern computers use 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

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

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^{64} 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 Million Terabytes

Memory implemented

Memory is generally implemented as a grid/matrix of bytes

16 columns

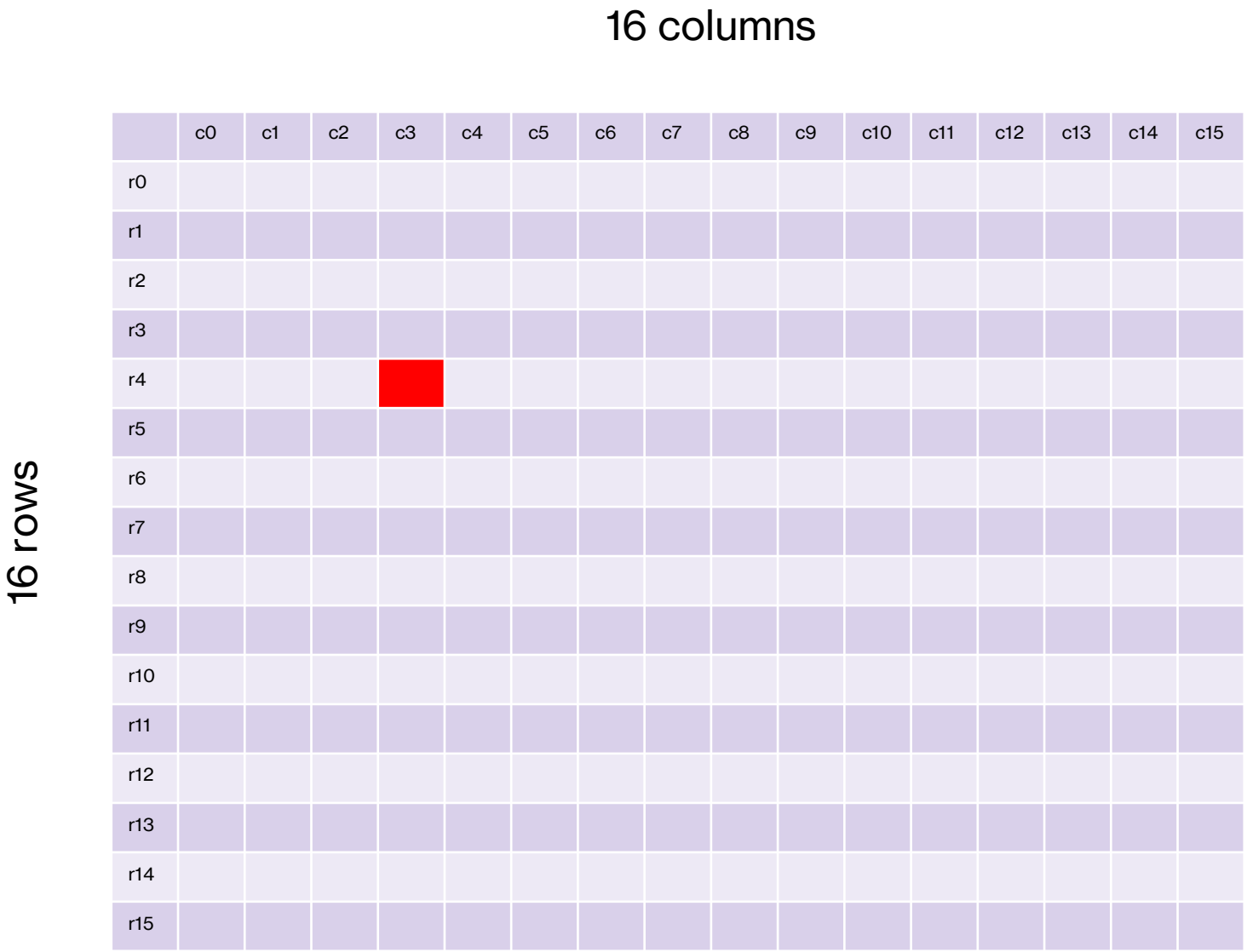
16 rows

	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

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

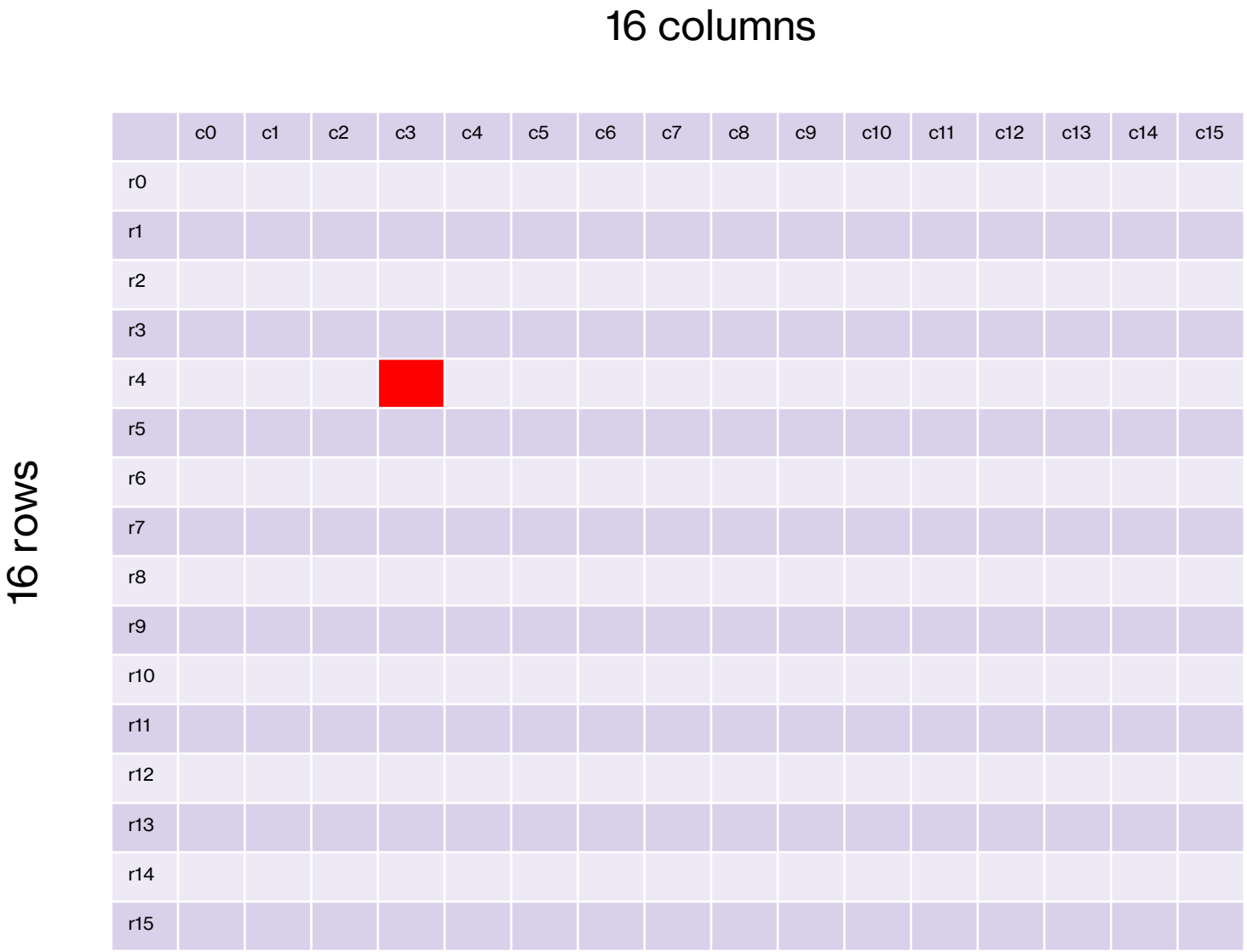
What row/column would this entry be? (Assuming indices start at 0)



Memory implemented

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

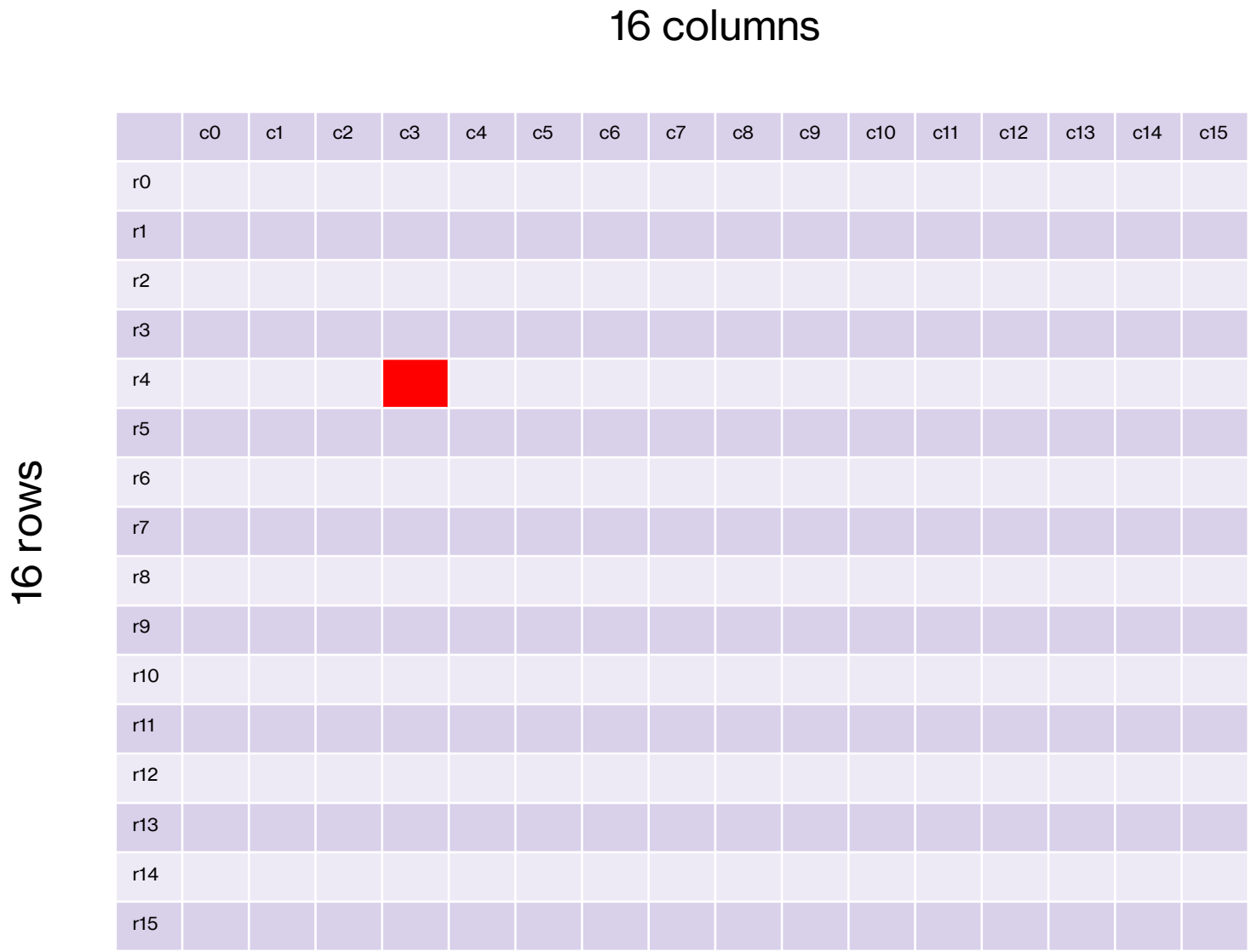
row = 4, column = 3



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?



Memory implemented

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

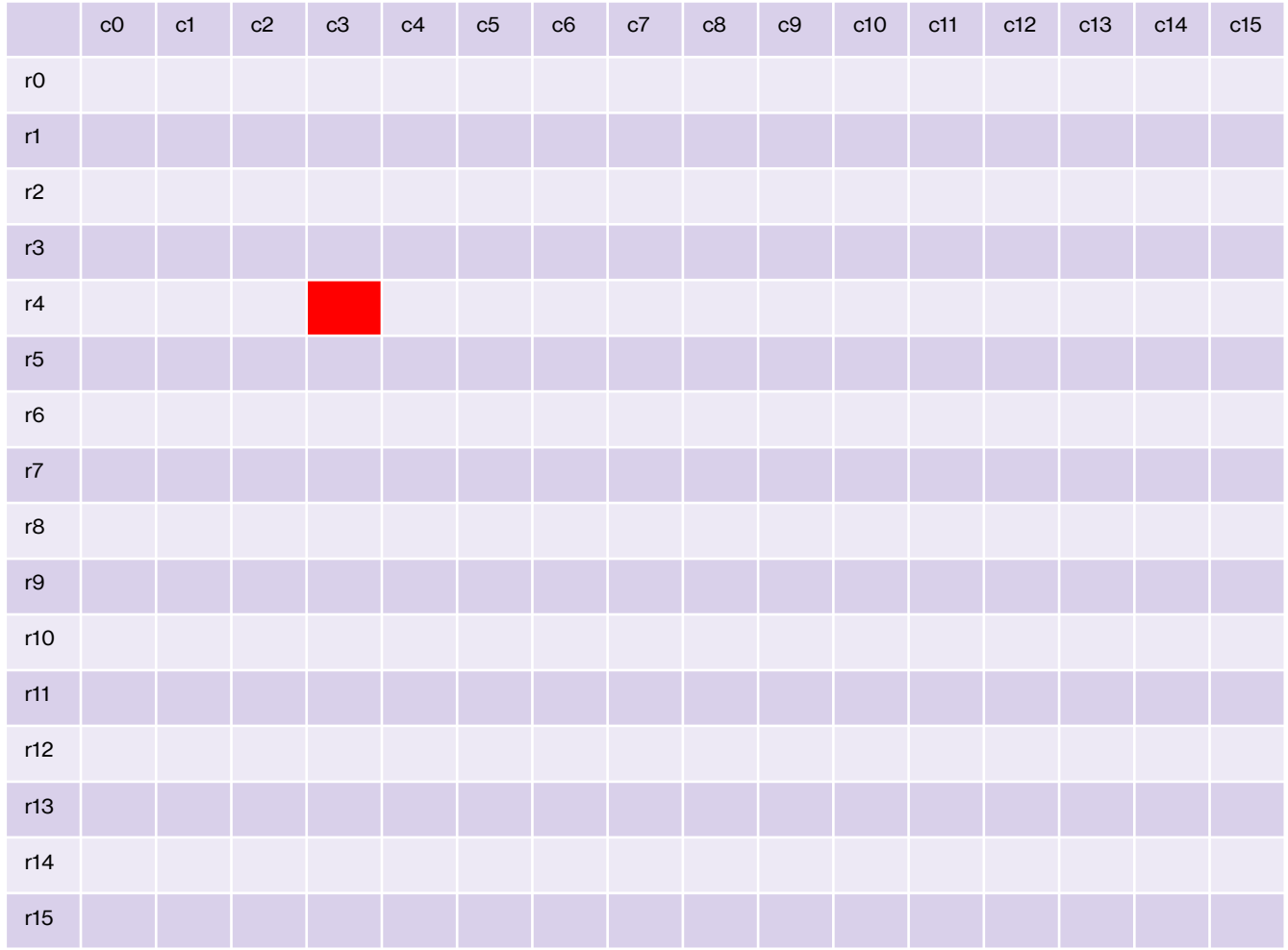
4 bits each for a total of 8 bits

<8-bit memory address>

4 bits for row 4 bits for column

16 columns

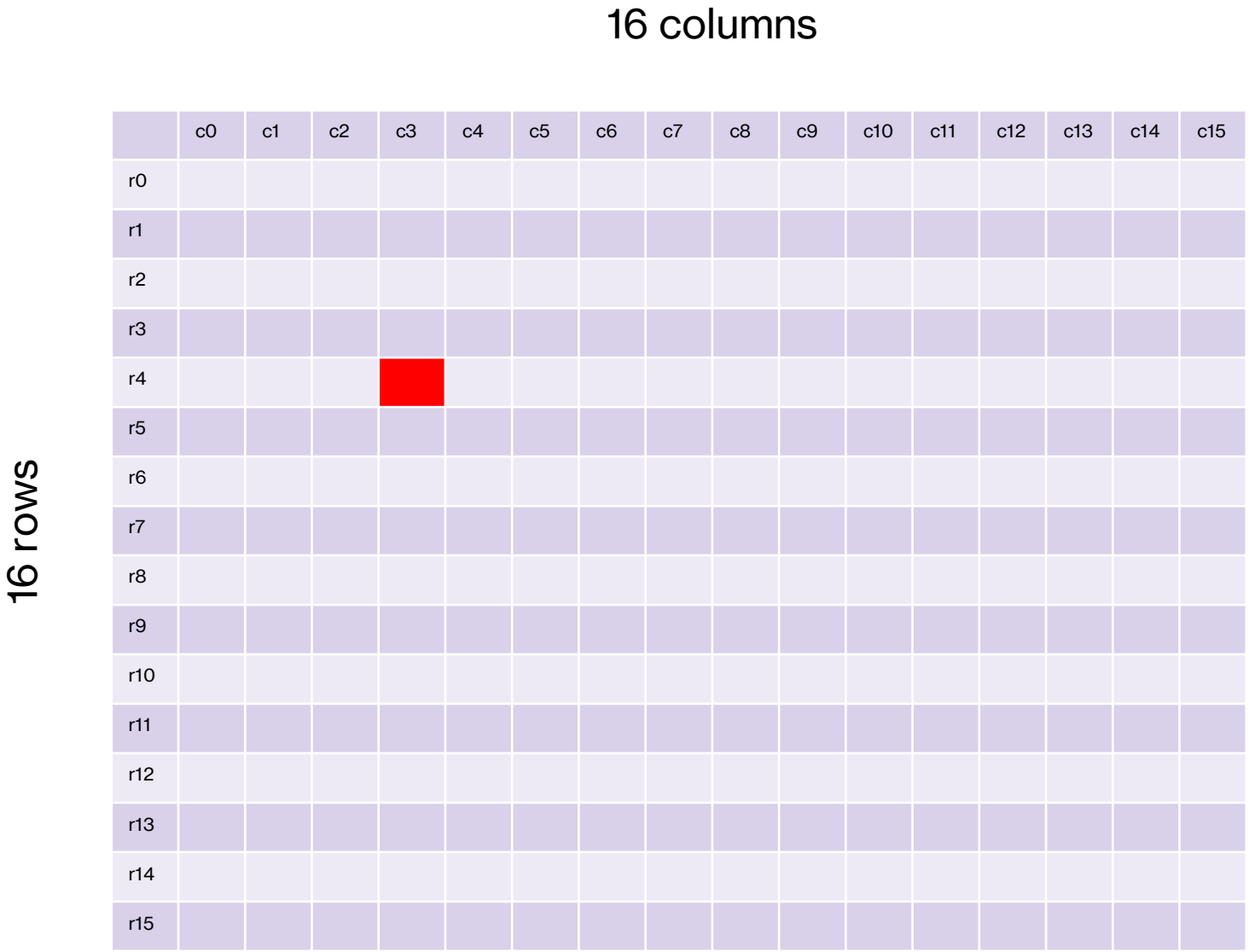
16 rows



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

Memory implemented

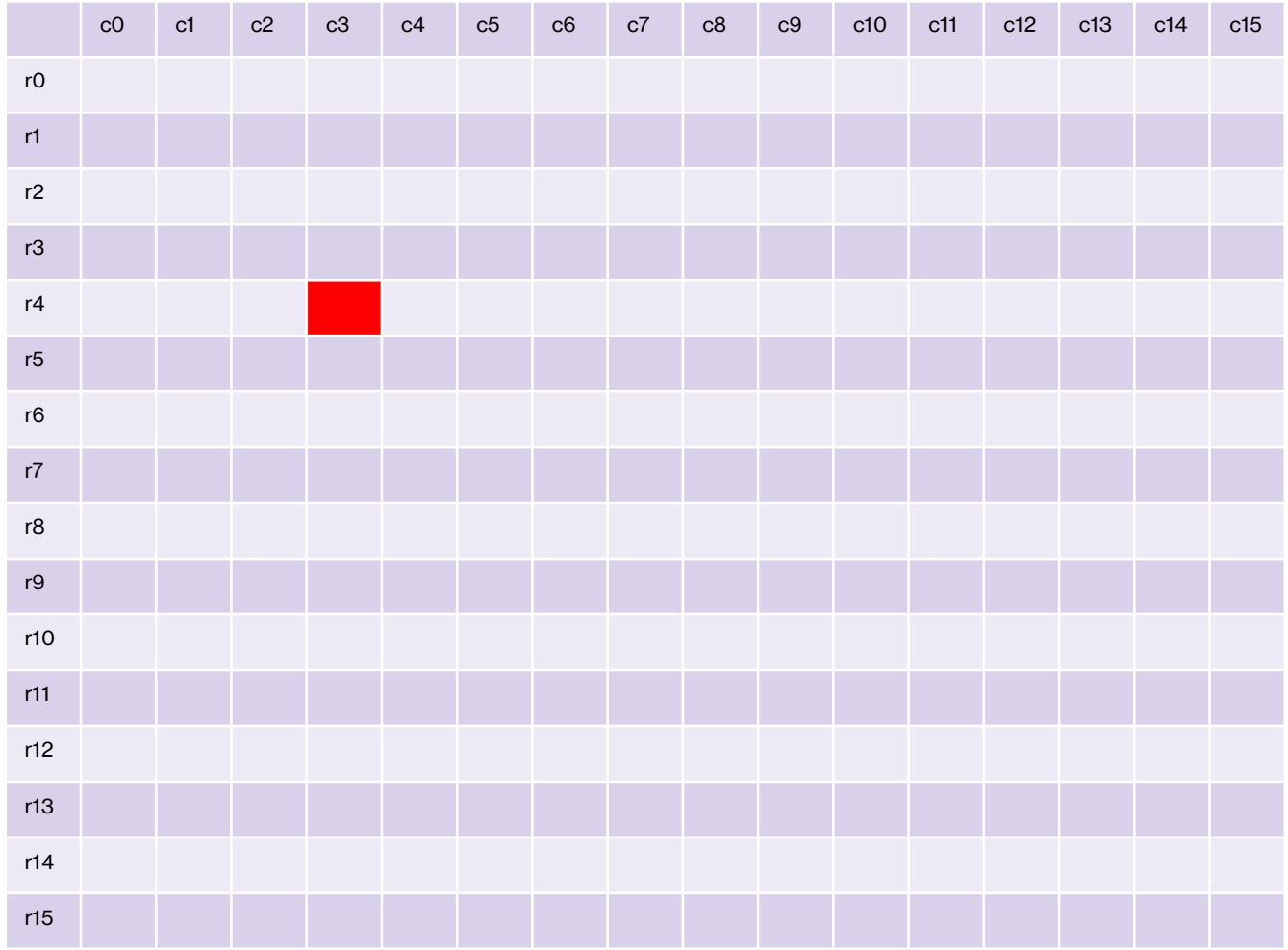
We can access a given by via a row index and a column index

0100 0011

row column

16 columns

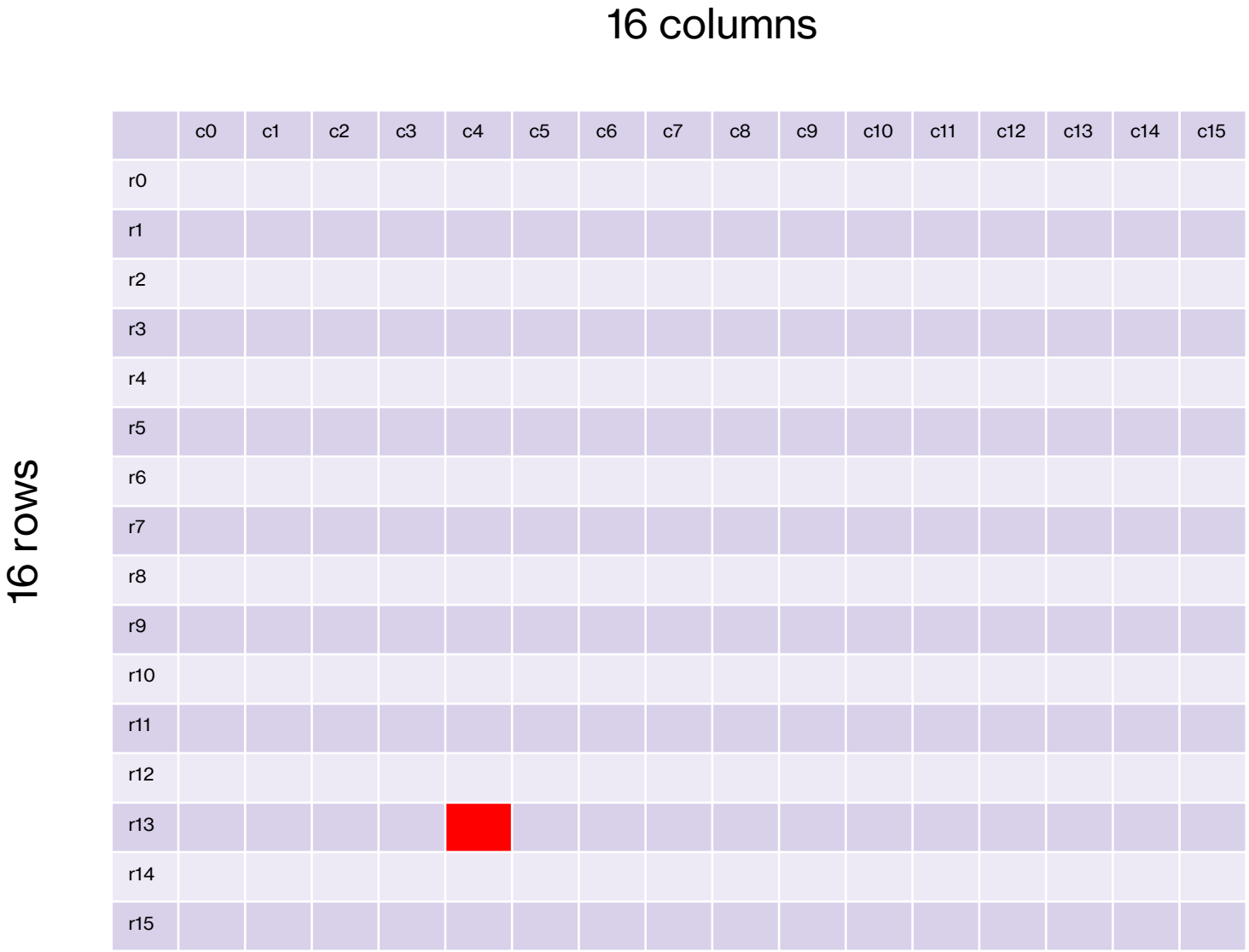
16 rows



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

Memory implemented

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

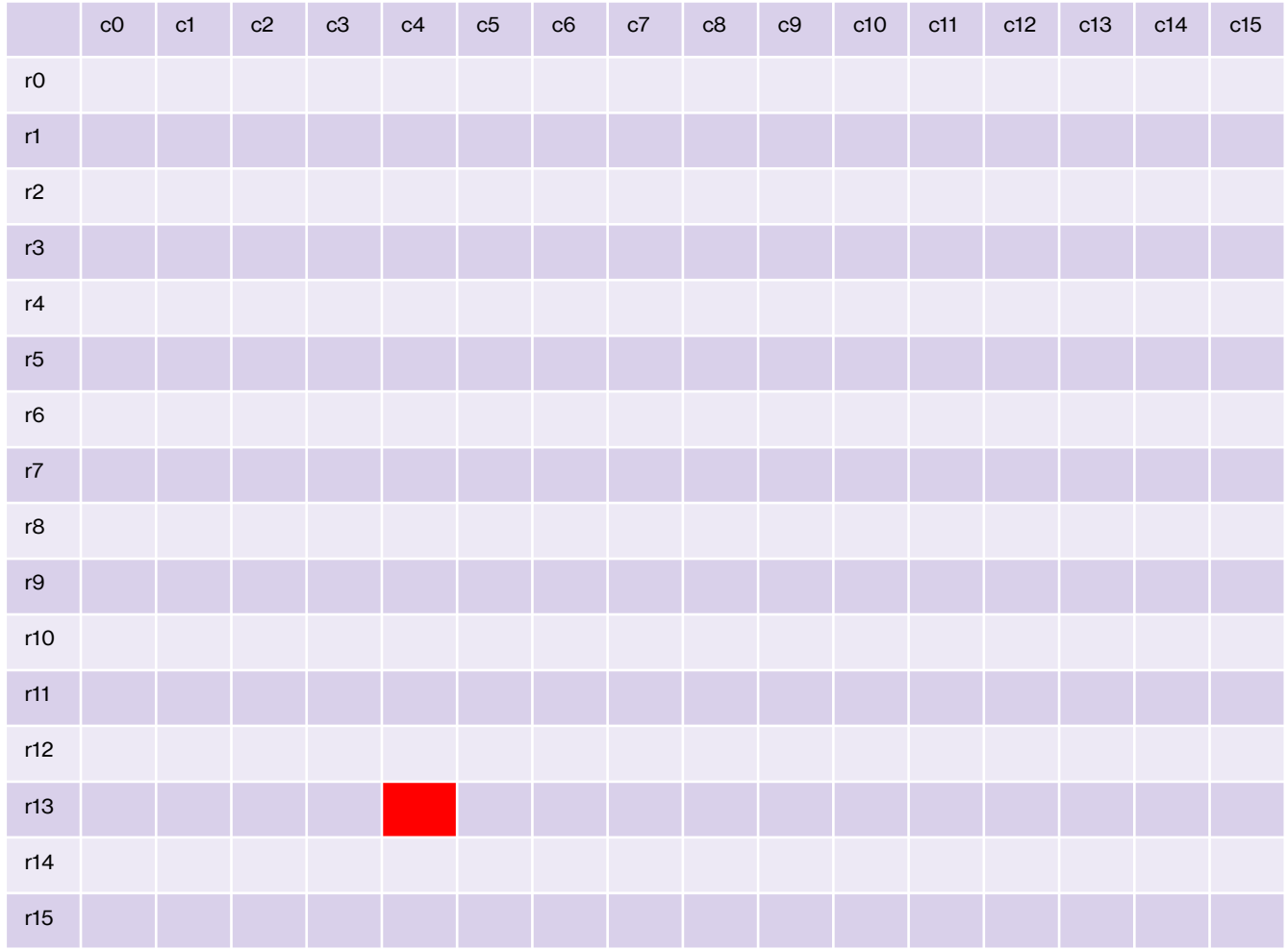
Note we use unsigned numbers

1101 0100

row column

16 columns

16 rows

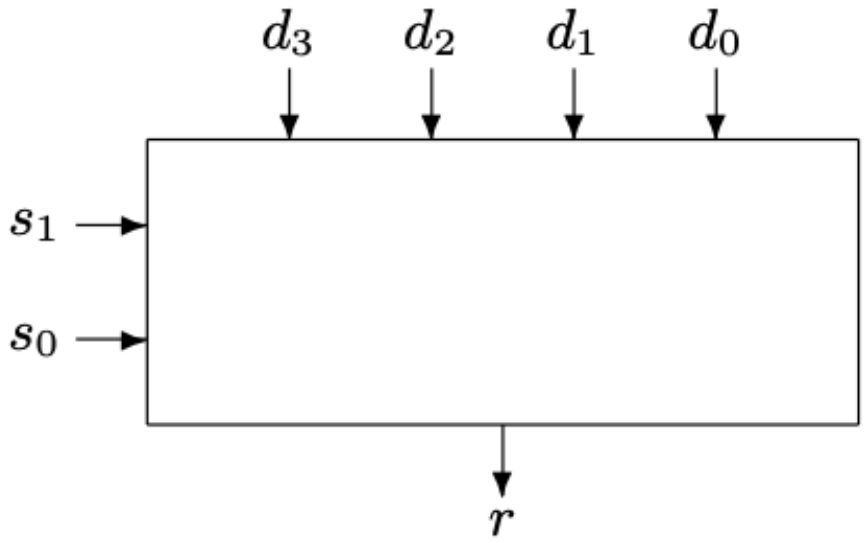


Quick aside: multiplexer

4:1 multiplexer

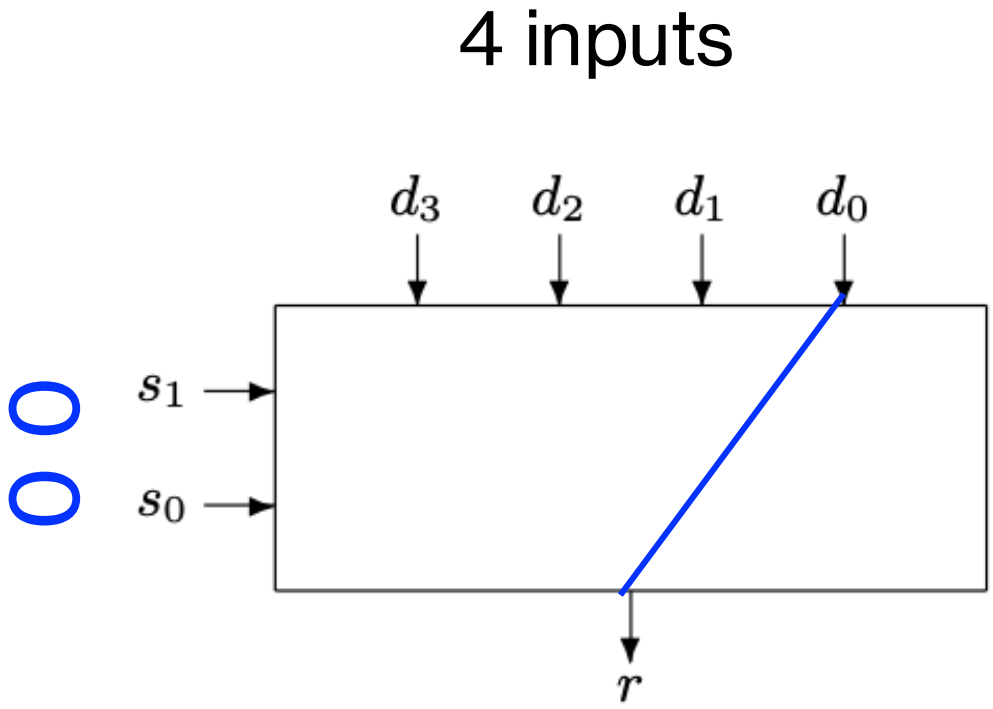
4 inputs

2 inputs to "select" which line



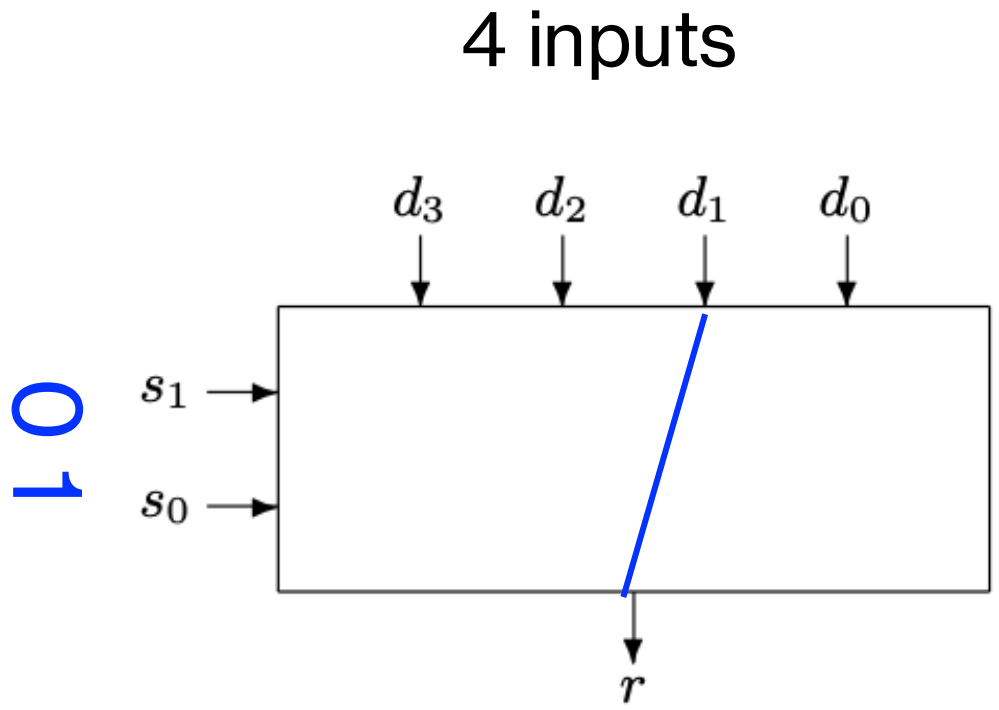
Quick aside: multiplexer

4:1 multiplexer



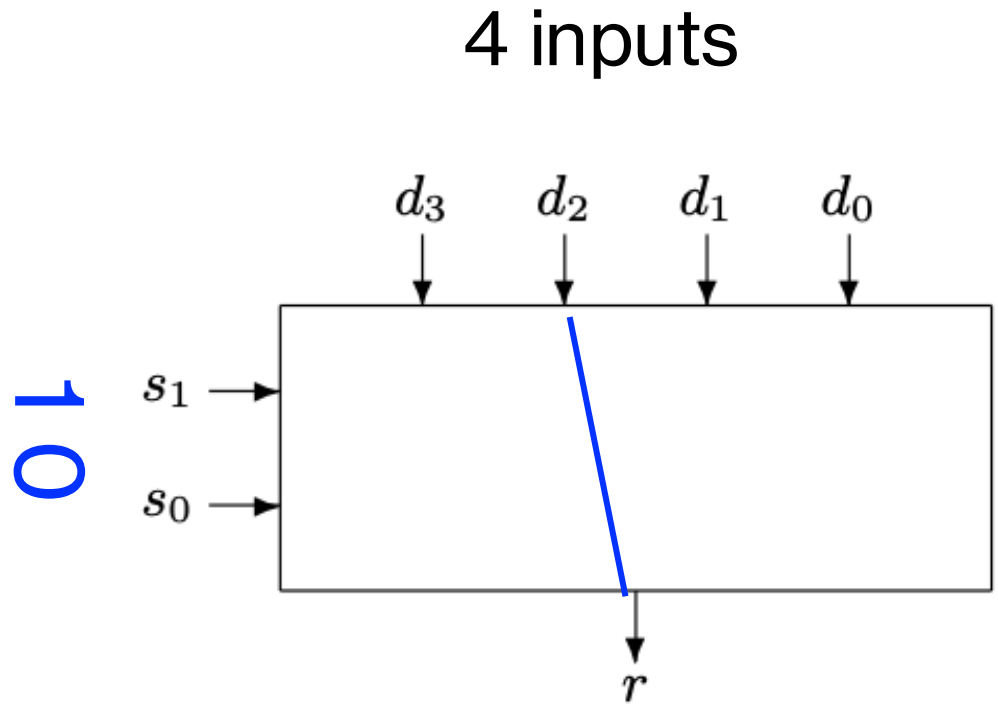
Quick aside: multiplexer

4:1 multiplexer



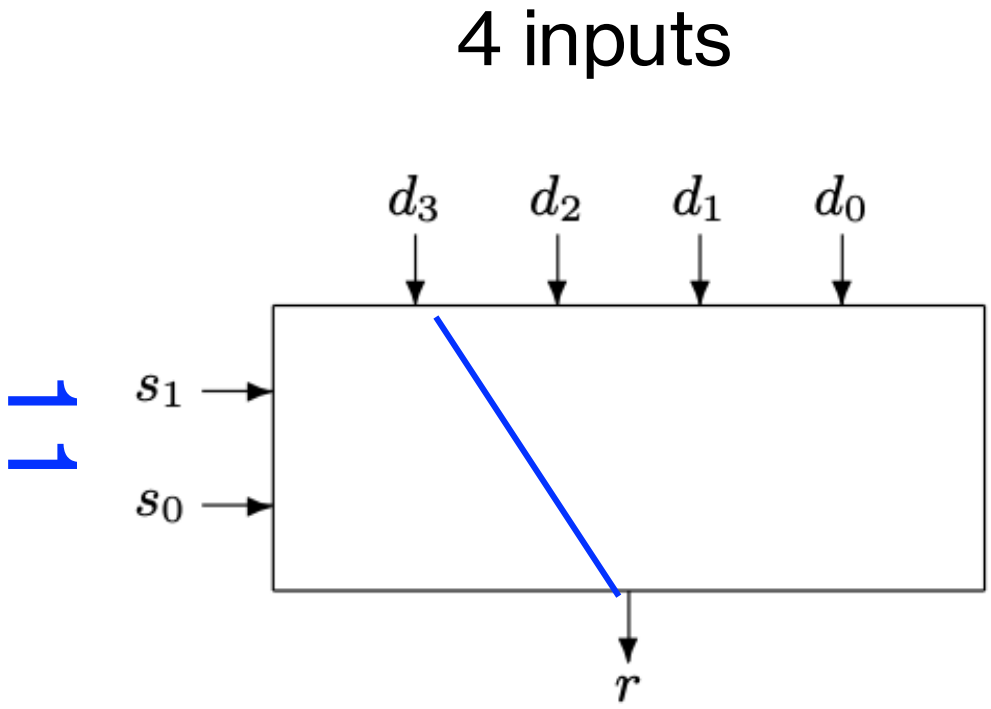
Quick aside: multiplexer

4:1 multiplexer



Quick aside: multiplexer

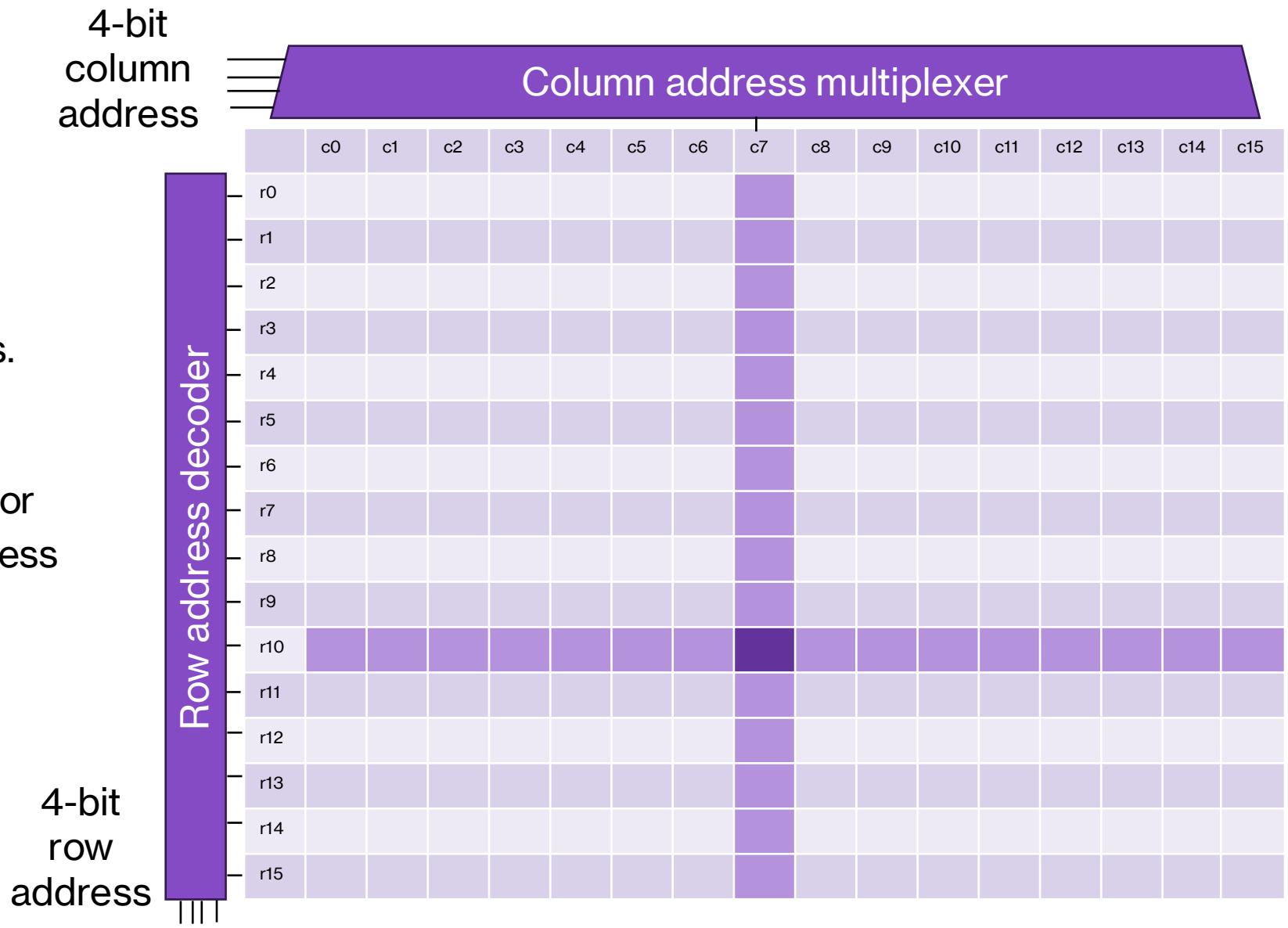
4:1 multiplexer



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.



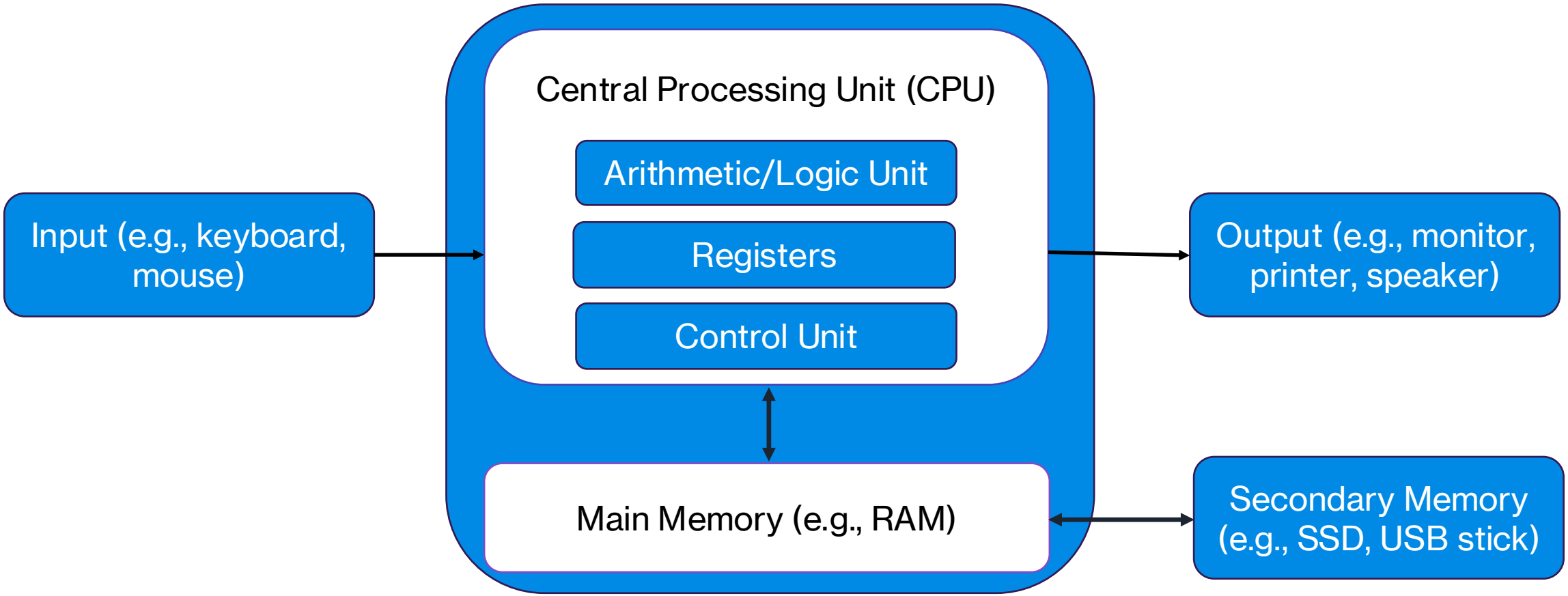
Memory core

- What we just built (with some simplifications) is one memory core.
- Memory cores are combined in bigger arrays to build increasingly bigger memory that can reach GBs.
- The actual physical component in each matrix cell can differ across implementations and affects how fast, compact, and expensive the resulting memory is.
- Some memories are built that way using a latch for each cell while others use capacitors to store the binary value of the cell.

Types of memory

- Memory can be divided in **RAM** (Random-Access Memory) and **ROM** (Read-Only Memory).
- RAM is considered **volatile**. This means that it loses its data when the power is turned off. It provides quick read and write access and is the main memory of your computer.
  - It was named random, because any data could be accessed as easily as any other regardless of where it was located.
- ROM is **non-volatile** and it retains its data even if the power is turned off. Although it is quick to read, it can either not be written (historically was written only at manufacturing time) or is very slow.
  - Memory in cameras, game consoles, laser printers is often ROM.
  - Your computer ROM is responsible for storing the BIOS (Basic I/O System) that instructs how to start your computer and loads the operating system into the RAM.

Revisiting the simplified view of a computer



Arithmetic Logic Unit (ALU)

- Performs arithmetic and logical operations.
- Arithmetic operations like adding A and B (ripple-carry adder!), increment/decrement of A by 1, etc.
- Logical operations, like bitwise AND/OR of A and B or logical and arithmetic shifting.

