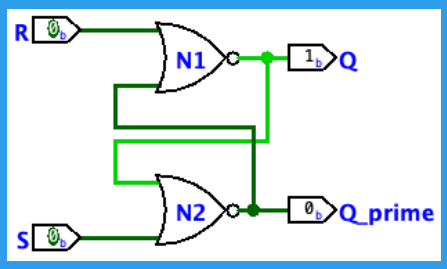
Computer Systems



Sequential Circuits

CS51 – Fall 2026

Today we will continue seeing circuits, but this time sequential circuits.

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**.

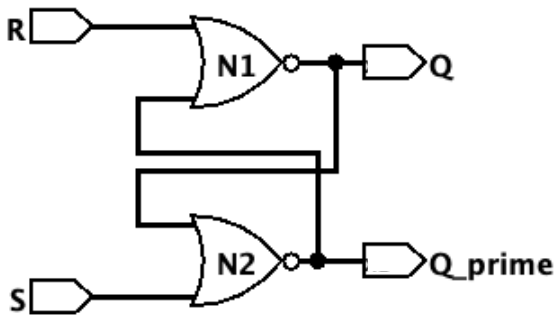
2

All circuits we have seen so far are 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 which not only depend on current input but have memory components that allow them to utilize prior inputs, too. And to establish some vocabulary regarding memory: Storing data into memory is called writing while retrieving data from memory is called reading.

SR-Latches

Let's see the simplest memory circuit we can build, an SR-latch.

Think

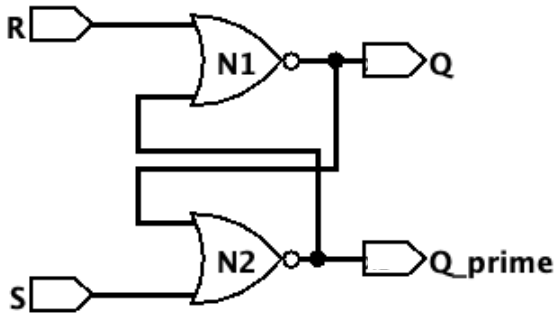


Anything unusual about this circuit?

4

This is a diagram of an SR-latch. Do you notice anything unusual about this circuit?

SR-latch



There are two cross-coupled NOR gates

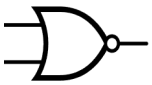
5

There are two cross-coupled NOR gates!

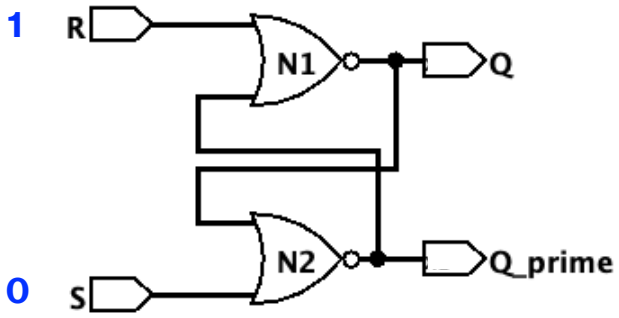
Think: $R = 1, S = 0$

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



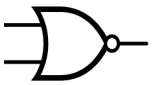
6

For inputs S=0 and R=1, what will the Q and Q' outputs be? As a help, this is the NOR truth table (the negation of OR)

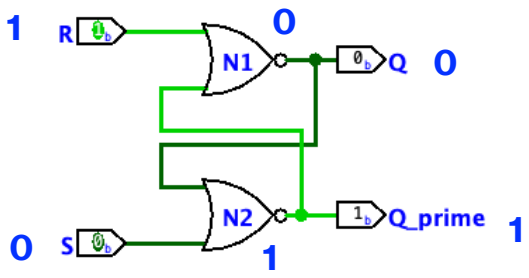
$R = 1, S = 0$

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



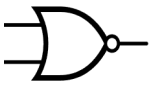
7

We get 0 and 1 respectively

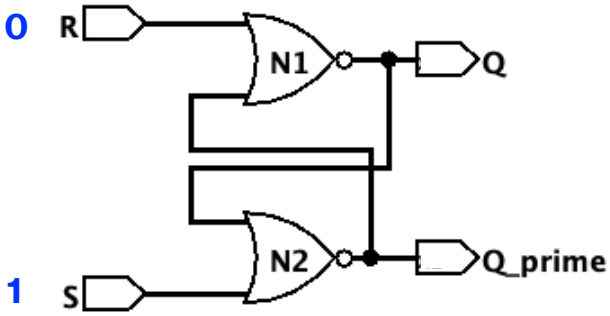
Think: $R = 0, S = 1$

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



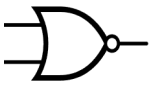
8

What if the input is S=1 and R=0?

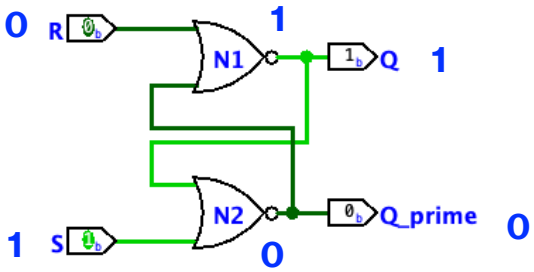
$\boldsymbol{R = 0, S = 1}$

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



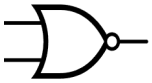
9

We see the outputs are also reversed to 1 and 0

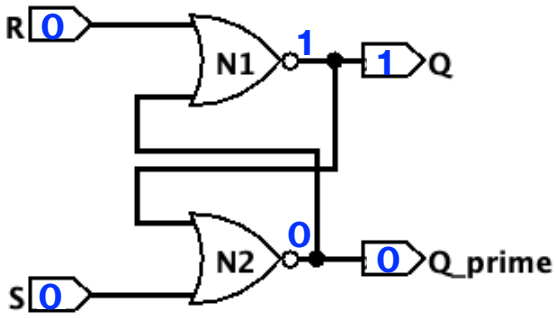
Think: $R = 0, S = 0, N1 = 1, N2 = 0$

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



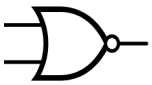
10

What if both R and S are 0 and N1 and N2 are 1 and 0?

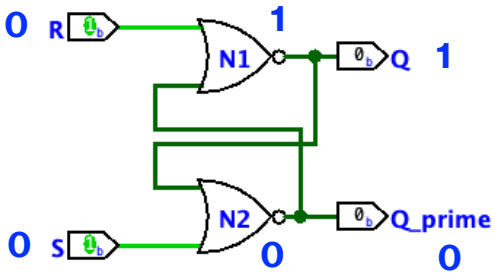
$\boldsymbol{R = 0, S = 0, N1 = 1, N2 = 0}$

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

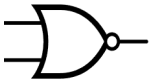
11

Hmm, nothing seems to change

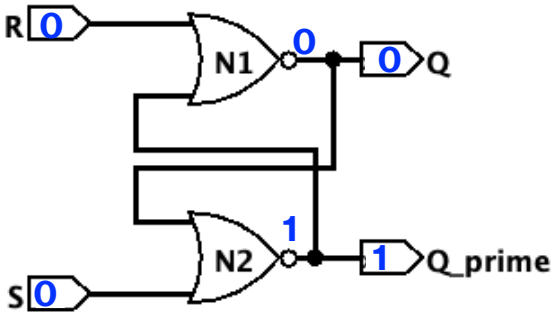
Think: $R = 0, S = 0$, N1 $= 0$, N2 $= 1$

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



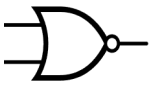
12

What if though for the same input N1 and N2 are reversed to 0 and 1?

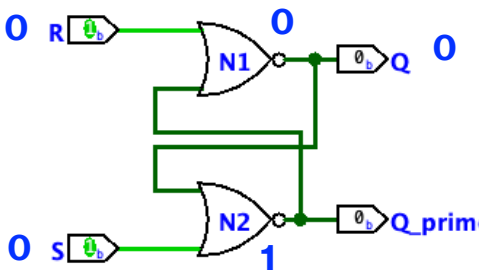
$R = 0, S = 0, N1 = 0, N2 = 1$

What will be the output?

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

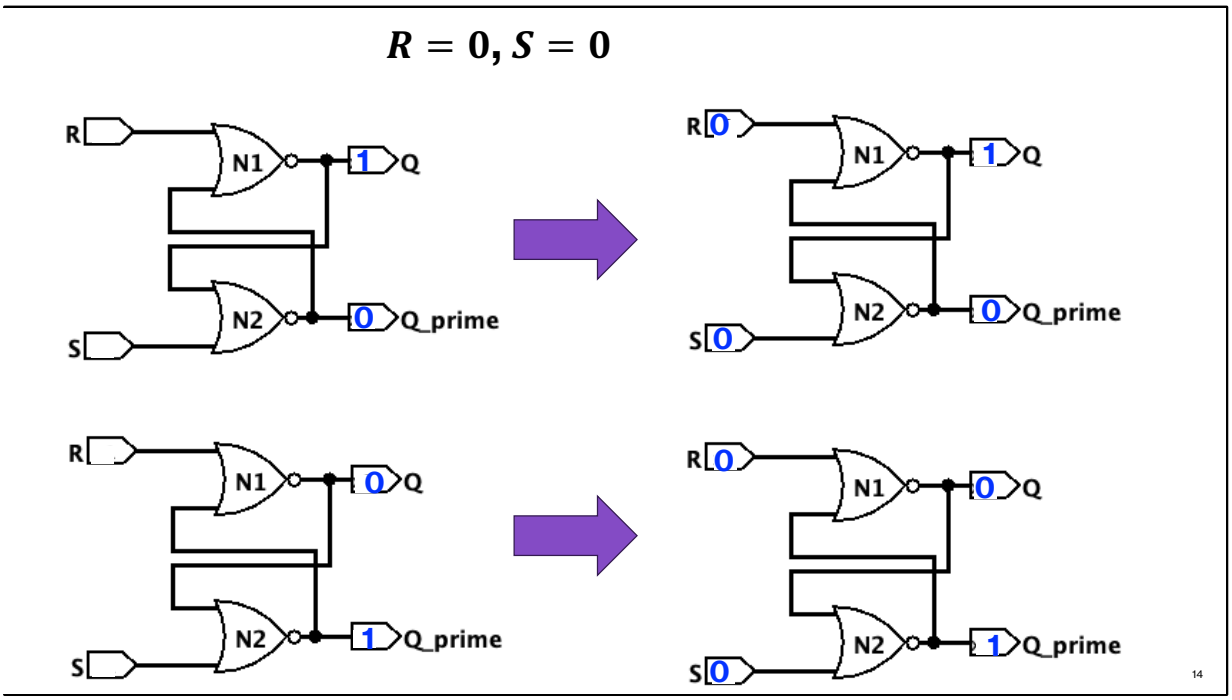


X	Y	X ⊽ Y
0	0	1
0	1	0
1	0	0
1	1	0



13

Again, nothing seems to change



Well look at that! when we have inputs R=0 and S=0 then the circuit latches, that is remembers and holds the previous state of what Q and Q' were! This is what we mean by building memory of 1 bit.

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

?

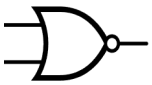
15

So we can either set (Q to 1), reset (Q to 0) or hold the previous state. Let's look at what happens if both S and R are equal to 1.

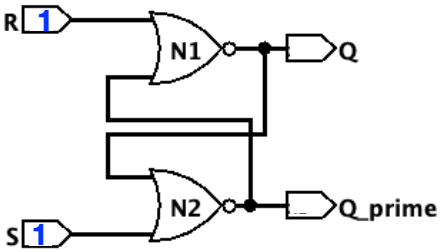
Think: $R = 1, S = 1$

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



16

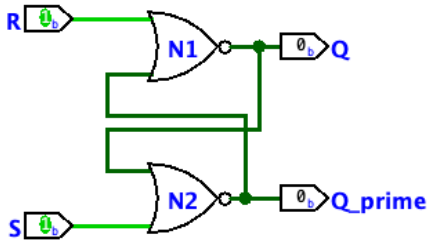
Last case, R=0, S=0. What will happen here?

$R = 1, S = 1$

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



17

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. This is considered to be illegal output so it's not uncommon to see this row on the table displaying illegal instead of 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.

18

One latch can store one 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 even 64 bits and are built directly into a CPU.
Remember that they store a relatively small number of bits and are extremely fast.

Memory

Next, we will talk about memory in general.

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

20

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 as shown in this table.

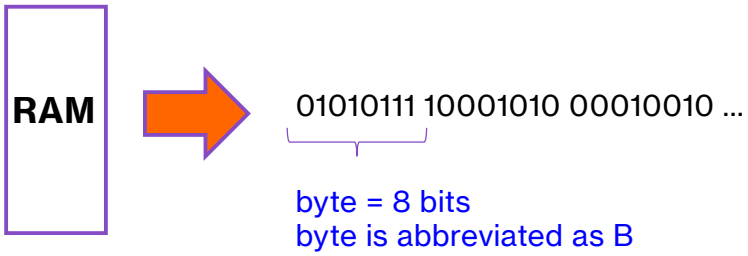
Units of memory

- **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

21

Let's see what are the most common units of measuring the size of memory. We will often use the base 10 and base 2 definitions interchangeably.

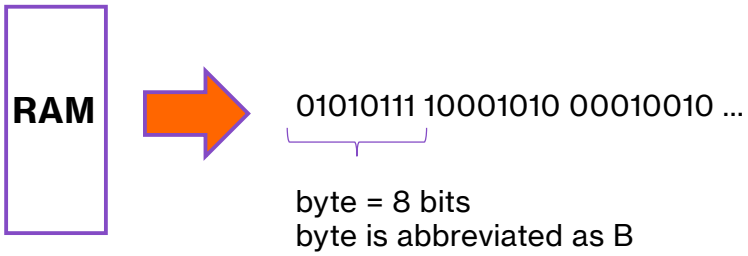
Memory



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

If my computer has 64 GB of memory, how many bits is that?

Memory

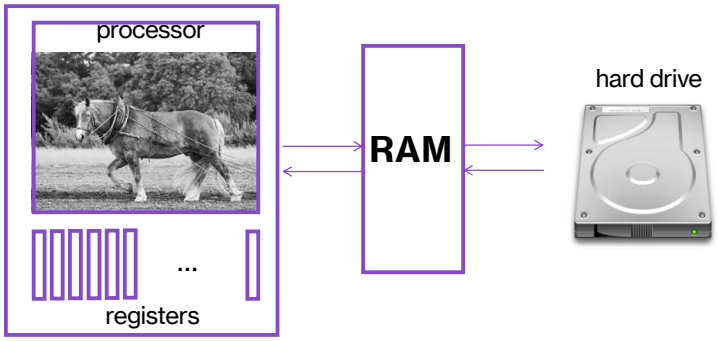


byte = 8 bits
byte is abbreviated as B

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

64 billion bites or 64*8=512 billion bits

Memory hierarchy

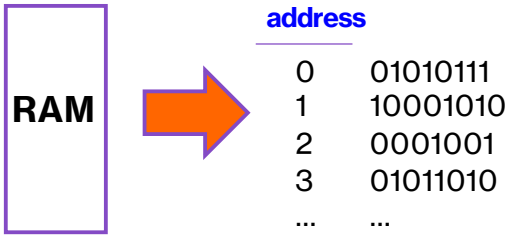


Capacity: bytes Gigabytes Terabytes

24

If we were to think of our computer, our CPU holds a some relatively small number of bytes between registers, the main memory holds a couple dozen gigabytes, and hard drives or external storage holds terabytes.

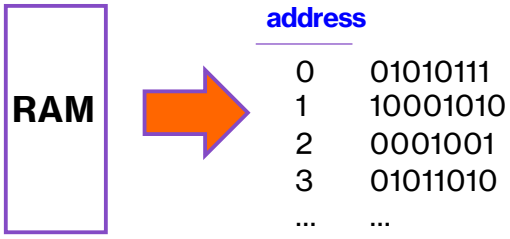
Memory



Memory is byte addressable

If we were to look into main memory, we would be able to address it by byte.

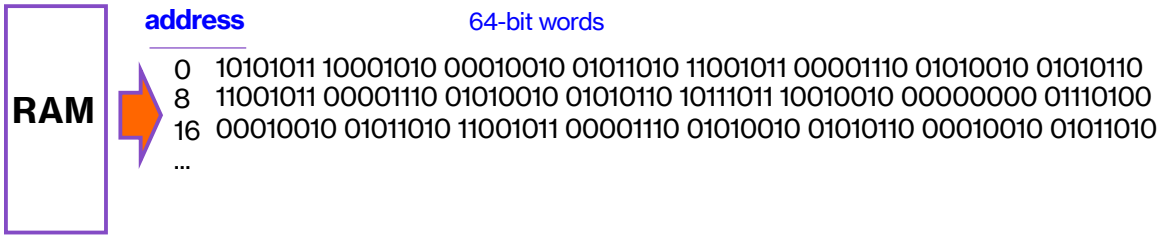
Memory



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

Specifically, we organize memory into words.

Memory



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

As we've discussed before, most modern computers use 64 bit words which corresponds to 8 byte words. This is an example of a 64-bit word memory

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?

28

If we have a 32-bit processor accessing memory locations using a 32-bit number, what is the largest a 32-bitbyte-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

29

It would be 2^32 unique numbers which is about 4 GB.

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?

30

What if we had 64 bit words?

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

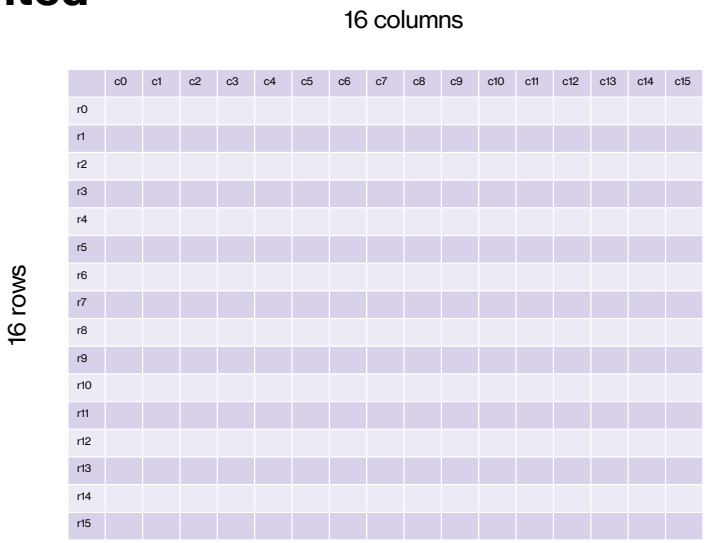
16 Exabytes = 16 Million Terabytes

31

2^64 unique numbers are about 16 million terabytes! This is a huge number and practically there are physical limitations

Memory implemented

Memory is generally implemented as a grid/matrix of bytes



32

In general, memory is implemented as a matrix of bytes. Here for example we have a 16x16 memory array.

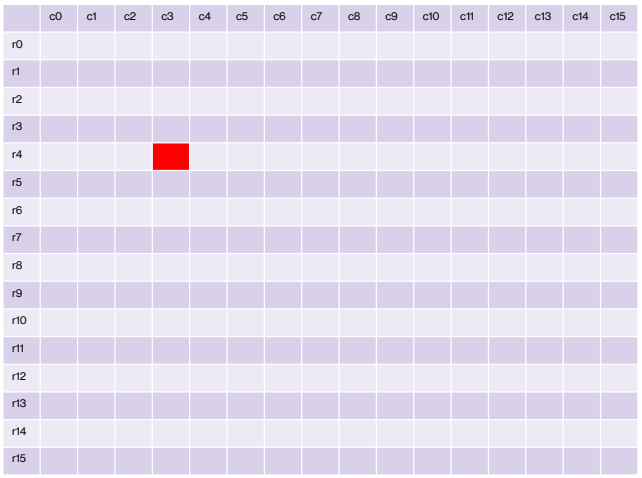
Memory implemented

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

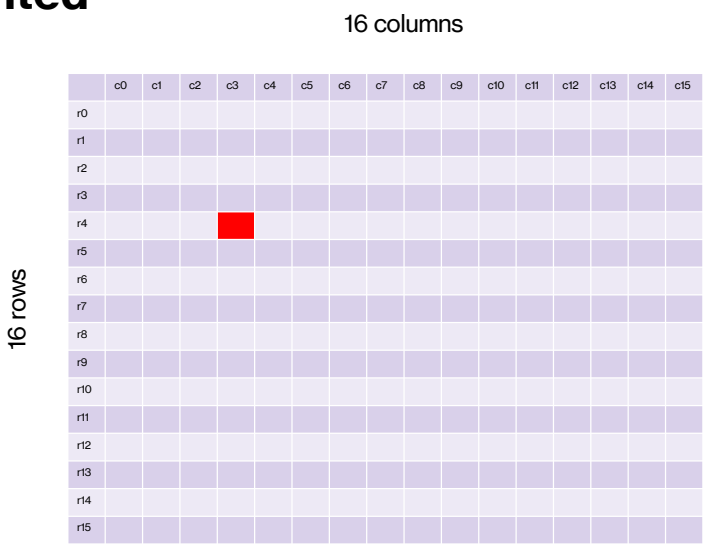


33

Memory implemented

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

row = 4, column = 3



34

row would be 4 and column 3

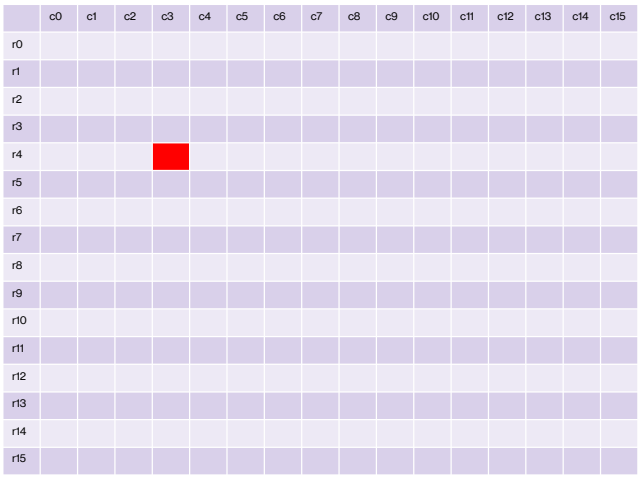
Memory implemented

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



35

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

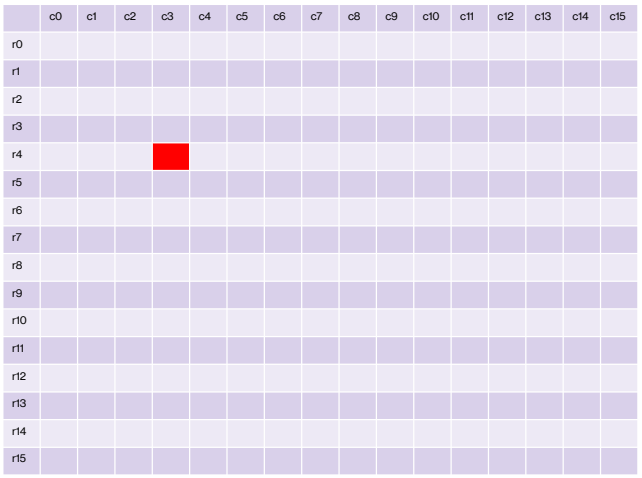
4 bits each for a total of 8 bits

<8-bit memory address>

4 bits for row 4 bits for column

16 columns

16 rows



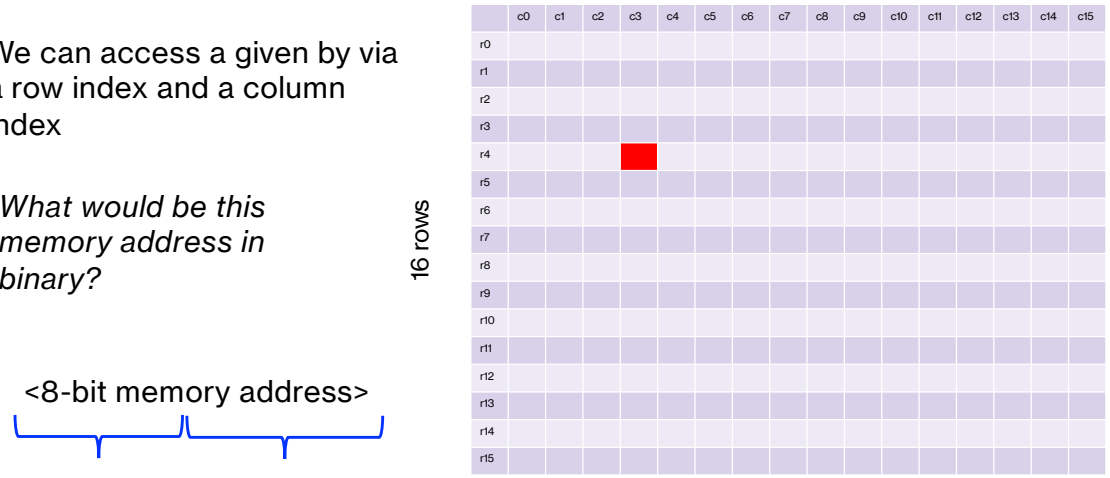
36

8 bits in total, 4 for row and 4 for column

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

37

Going back to this example, what would the memory address be here?

Memory implemented

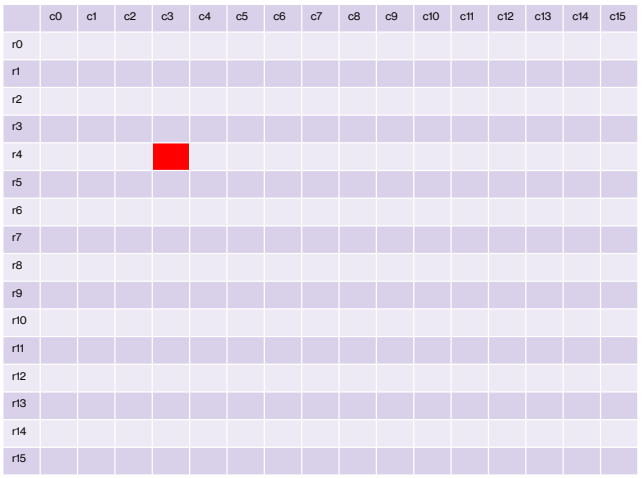
We can access a given by via a row index and a column index

0100 0011

row column

16 columns

16 rows



38

0100=4 and 0011 = 3

Memory implemented

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

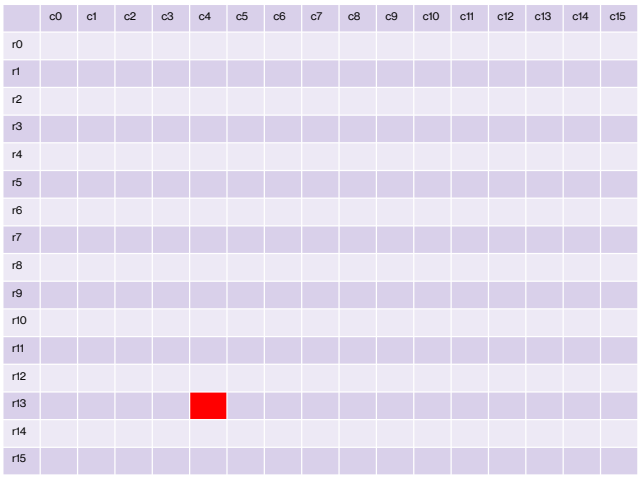
What would be this memory address in binary?

<8-bit memory address>

4 bits for row 4 bits for column

16 columns

16 rows



39

Let's do one more

Memory implemented

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

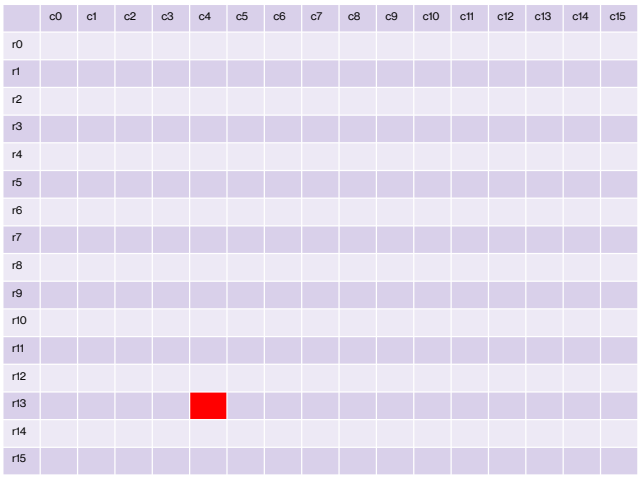
Note we use unsigned numbers

1101 0100

row column

16 columns

16 rows

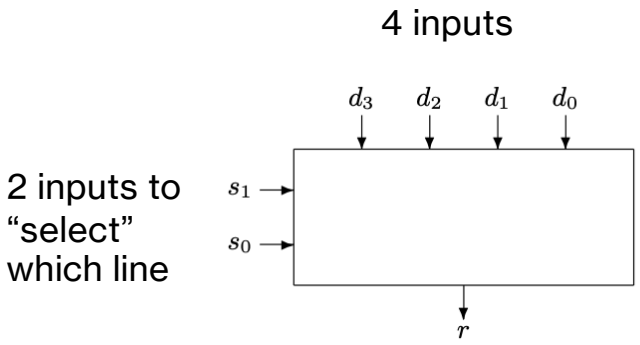


40

11010100 for row 13 and column 4. Notice that we use unsigned numbers

Quick aside: multiplexer

4:1 multiplexer

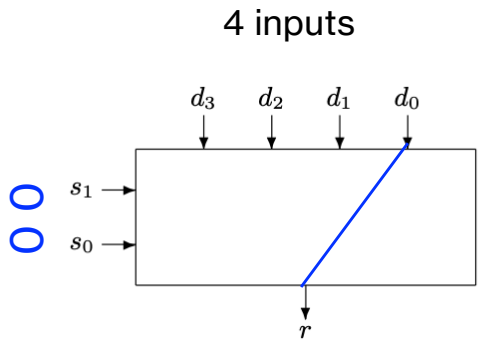


41

We have seen decoders. Now we will see another circuit called multiplexer. Specifically let's see a 4:1 multiplexer which takes 4 inputs and log4=2 inputs to select a line.

Quick aside: multiplexer

4:1 multiplexer

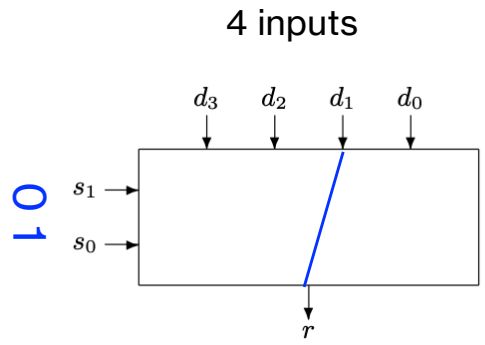


42

For example, if our inputs are 0 0 then the input of d0 is propagated to r.

Quick aside: multiplexer

4:1 multiplexer



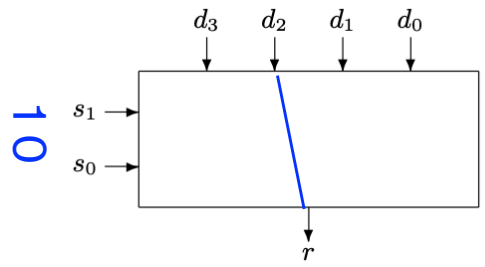
43

if our inputs are 0 1 then the input of d1 is propagated to r.

Quick aside: multiplexer

4:1 multiplexer

4 inputs



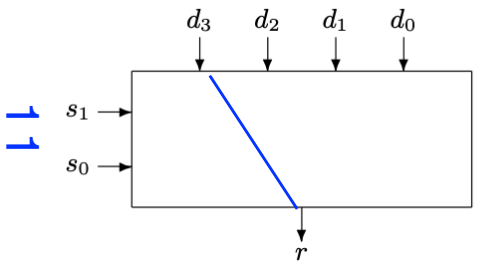
44

if we have 10 then it's d2

Quick aside: multiplexer

4:1 multiplexer

4 inputs



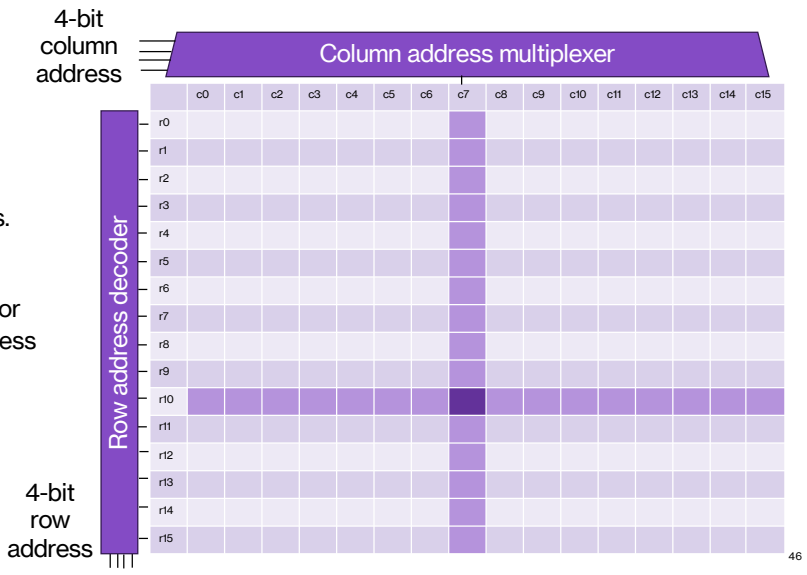
45

and finally, if we have 11 then it's d2

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.



To convert from an address to the actual memory cell that contains the desired bit, we will utilize one multiplexer and one decoder.
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.

47

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.

48

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.

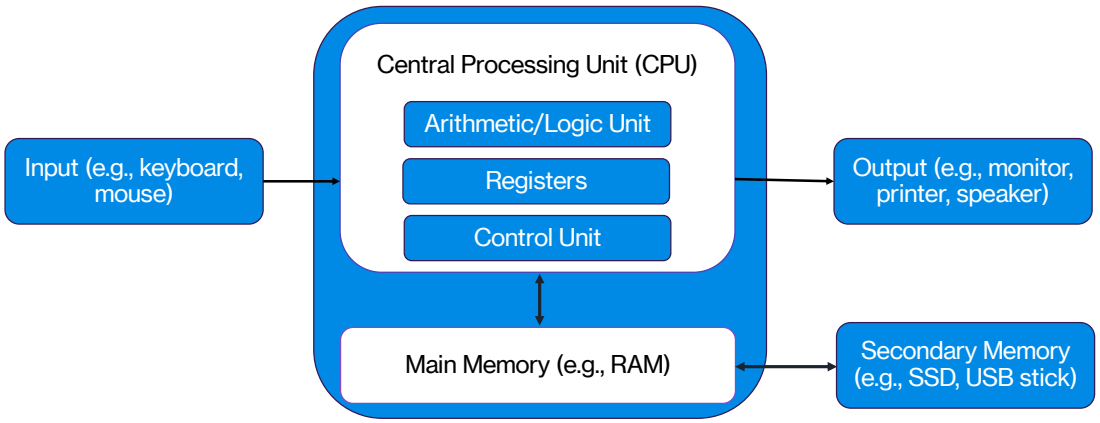
It was named random, because any data could be accessed as easily as any other regardless of where it was located.

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



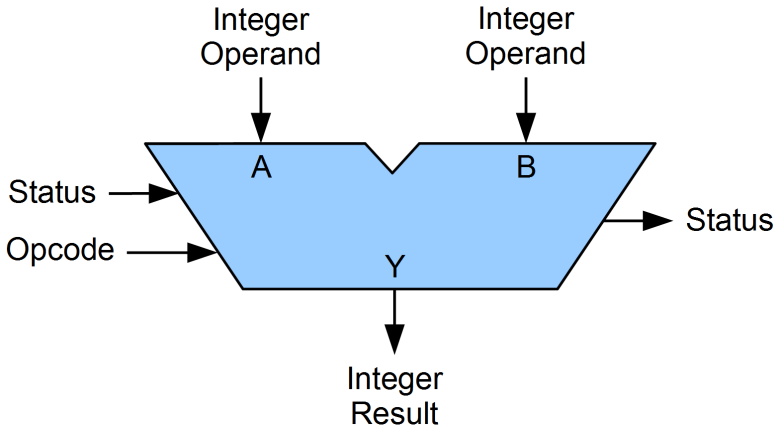
49

A simplified view of a computer would show that it comprises mostly of two components. A central processing unit, i.e. the CPU, and memory (also known as primary or main memory). You can think of the CPU as the brains of your computer. It consists of three main components. An arithmetic/logic unit, known as the ALU, that performs arithmetic and logic operations, processor registers that supply information to the ALU and store the results that the ALU outputs, and a control unit that orchestrates the fetching of data from the main memory and which coordinates the ALU and the registers. The main memory stores current data, programs, and instructions that can be accessed directly by the CPU. Any data that should not be changed is stored in a special type of main memory called ROM (read-only memory). For example, the programs that your operating system needs when it boots are stored in ROM. In contrast, RAM (random-access memory) is like our computer's short-term memory for data and programs that need to be read and written but is volatile, that is its information is wiped if power is removed. Secondary memory stores programs and data permanently, without having to worry about what happens when we power off our computer. Secondary memory comes mostly in the form of solid-state drives (SSDs) these days. They are smaller, faster, and less noisy than the older hard-disk drives (HDDs). Other examples of secondary memory are usb

drives, memory cards, CDs/DVDs. All this exchange of information can be supplemented by input that the user provides through their keyboard, mouse, etc, and it can be directed to output devices such as the monitor, printers, speakers, etc.

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.



50

If we zoom into the CPU, we would see the ALU which 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. The upcoming assignment will familiarize you with circuits like this!