

Registers and memories

Lecturer: Guillaume Beslon
(Lecture adapted from Lionel Morel)

Computer Science and Information Technologies - INSA Lyon

Fall 2026

Content

Sequential behaviors

Registers

n-bit Registers

Addressable memories

Memory hierarchy

Why Sequential circuits?

- ▶ All the circuits we have built so far are combinational. They combine their (binary) inputs to compute their (binary) outputs. **i.e.** they implement **functions** (in the mathematical sense of the term):

$$\text{Outputs} = f(\text{Inputs})$$

For a given input value, such circuits **always** produce the same output, whatever input values it received in the past.

- ▶ But combinational circuits cannot store information. To store information, we need **sequential circuits** that implement behaviors of the form:

$$\text{Outputs}(t + 1) = f(\text{Inputs}(t), \text{Outputs}(t))$$

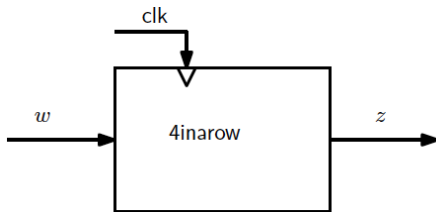
⇒ Today, we will learn how to build sequential circuits...

Foreword: Running Example

Let's design a circuit with input w , and output z .

- ▶ z becomes true whenever w was true for “long enough”
- ▶ z becomes true whenever w was true 4 “instants” in a row

⇒ need memory to “remember” occurrences of w .



Sequential circuits

A sequential element keeps a state.

So a sequential circuit is not a function:

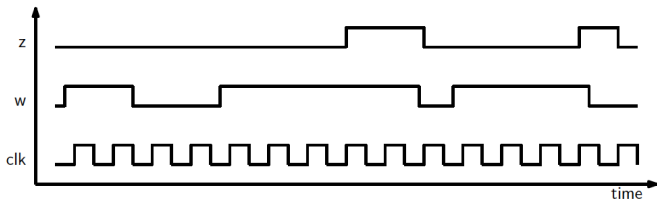
- ▶ for a given input value, the output value depends on:
 - ▶ the **current** input value
 - ▶ the **sequence of input values** that have been **received in the past**

⇒ In general, representing a sequential circuit with a truth table is impossible

Representing sequences of values

Two forms:

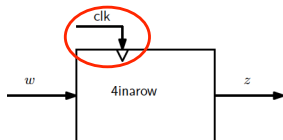
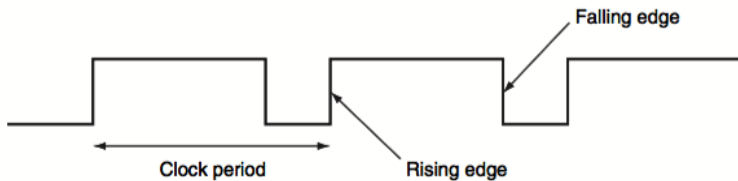
- ▶ time-diagrams, called **chronograms**
 - Chronograms describe graphically a **specific** sequence of input/output values
 - A chronogram is only a **representative example** of the circuit's behavior
 - Example: 4-in-a-row



- ▶ State machines (FSM):
 - Describe the full system's behavior **formally**
 - Next lecture

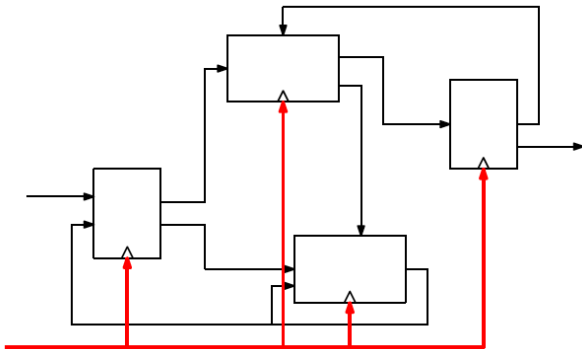
Clocks

- ▶ 99% of existing processors are **synchronous**
- ▶ A synchronous sequential circuit receives a **clock signal**



Synchronous circuits

- ▶ 99% of existing processors are **synchronous**
- ▶ A synchronous sequential circuit receives a **clock signal**
- ▶ signals can then be synchronized, either on rising or falling edges of the clock
- ▶ clock ticks are supposed to be long enough to mask propagation delays.



Content

Sequential behaviors

Registers

n-bit Registers

Addressable memories

Memory hierarchy

Building a very simple 1 bit memory

The most basic sequential element is the 1-bit memory.

- ▶ If $\text{Keep} = 1$ then $\text{Output}(t + 1) = \text{Output}(t)$
- ▶ If $\text{Keep} = 0$ then $\text{Output}(t + 1) = \text{Input}(t)$

This simple behavior can be represented by a truth table:

Keep	Input(t)	Output(t)	Output($t + 1$)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



$$O(t + 1) = \overline{K}.I(t).\overline{O(t)} + \overline{K}.I(t).O(t) + K.\overline{I(t)}.O(t) + K.I(t).O(t)$$

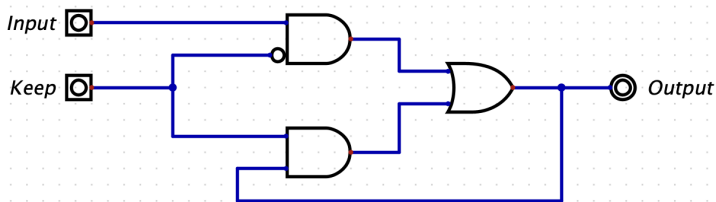
Building a very simple 1 bit memory

$$O(t+1) = \overline{K}.I(t).\overline{O(t)} + \overline{K}.I(t).O(t) + K.\overline{I(t)}.O(t) + K.I(t).O(t)$$

⇓

$$O(t+1) = \overline{K}.I(t) + K.O(t)$$

⇓

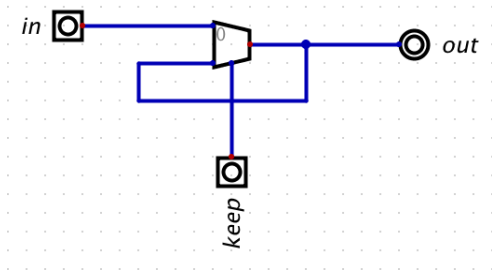


! Be sure that you really understand the behavior of this circuit !

Building a very simple 1 bit memory

But one could also notice that $O(t+1) = \bar{K}.I(t) + K.O(t)$ is the Boolean equation of a **2-to-1 multiplexer**

Hence the circuit can be further simplified:



This circuit is a **Latch** (un verrou); More precisely a **D-Latch**¹...

¹A **latch** is a generic memory circuit that stores one bit of data. A **D-latch** is a specific type of latch that stores its Data input when enabled and holds its previous value when disabled.

We need more

A latch's output changes **whenever the keep signal is up...**

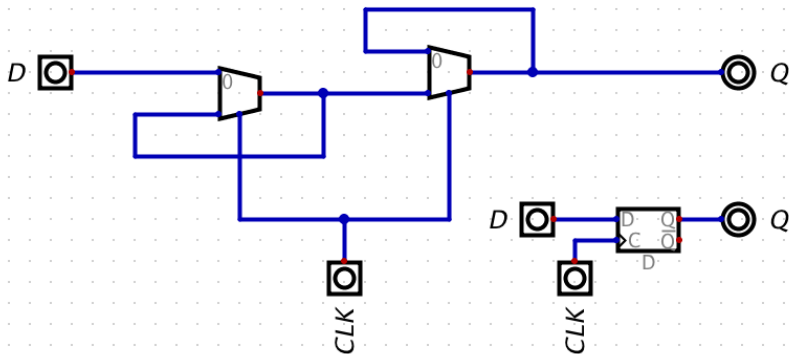
This is not accurate: signal is up for while!

We need a memory element that:

- ▶ is driven by a clock
- ▶ changes on edges

This can be implemented using TWO D-Latches...

D-Flip-flop (bascule synchrone)



- ▶ The D-FlipFlop stores the “D” (Data) bit when its CLK input goes high
- ▶ Beware: there are various types of Flip-Flop (T-FlipFlop, RS-FlipFlop, JK-FlipFlop) which behave differently

We need more

A D-Flip-Flop stores the D input **On each rising edge of the clock signal...**

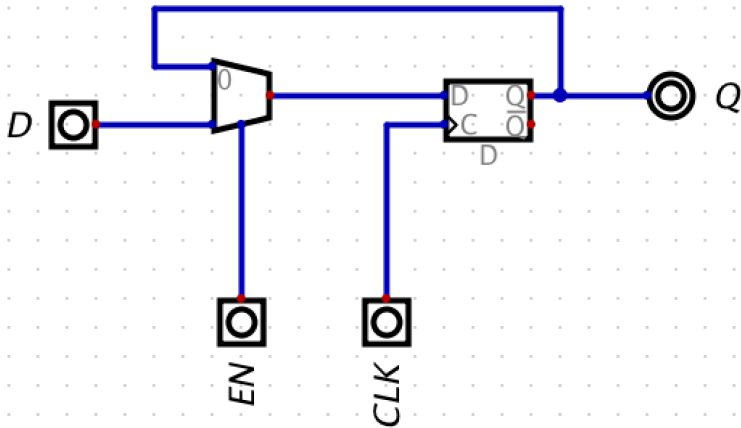
Hmm... memory content may changes at each time step!

We need a memory element that:

- ▶ is driven by a clock
- ▶ changes on edges
- ▶ can change or not depending on a WRITE-ENABLE (or ENABLE or “EN”) signal

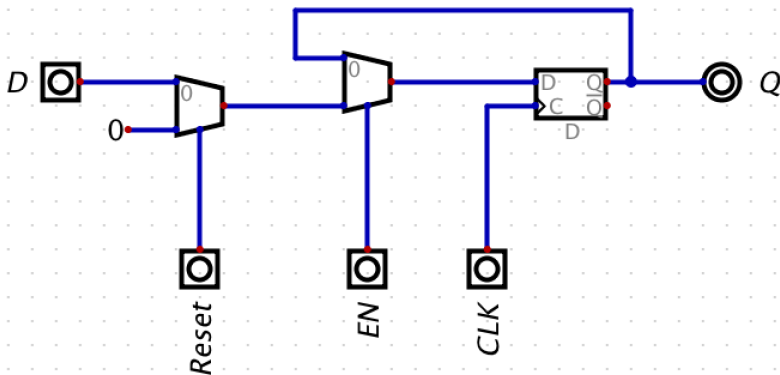
This can be implemented by adding a MUX to the Flip-Flop...

Write-enabled flip-flop



Reset command

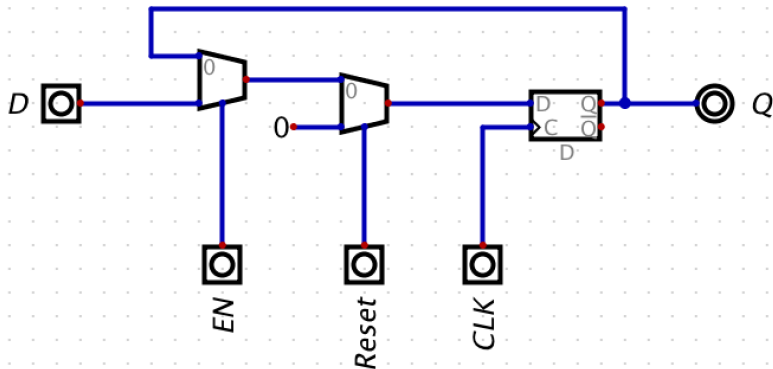
With the same kind of idea, one can add a RESET command to the Flip-Flop



WARNING: This circuit implements a **Synchronous** reset but many D-FlipFlop (including the one of the Digital simulator) are **Asynchronous**.

Reset command

Note that the exact behavior of the circuit depends on the relative positions of the two multiplexers



Here the Reset signal is effective whatever the state of the EN signal...

Content

Sequential behaviors

Registers

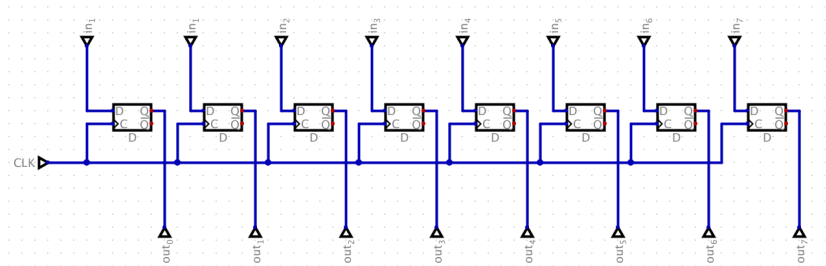
n-bit Registers

Addressable memories

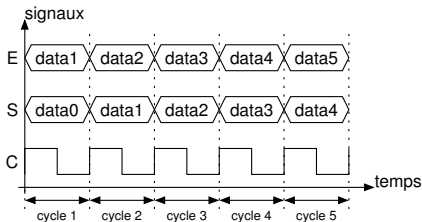
Memory hierarchy

n-bits register (registre n-bits)

The simplest possible n -bits register is made of n D-flip-flops in parallel (all flip-flops having the same clock signal):



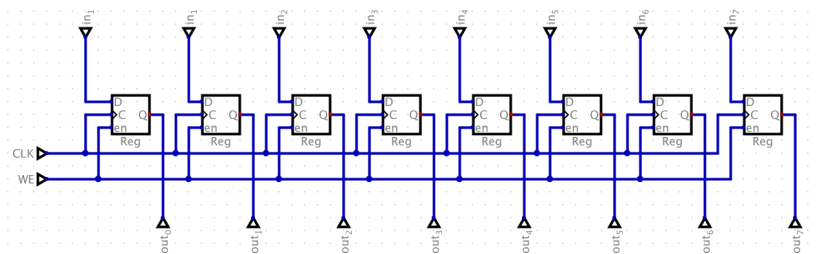
Chronogramme:



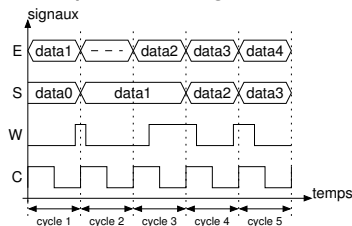
- ▶ The input is stored at the CLK rising edge; the output is stable during the whole CLK cycle
- ▶ Between two CLK rising edges, all variations of the input signal are ignored

Write-Enable n-bits register

To memorize a data over several cycles, we add a signal *WE* (Write Enable, sometimes simply called "EN"):



example chronogramme:



At the end of each cycle:

- ▶ If $WE=1 \rightarrow$ the input data is stored and made available on the output;
- ▶ if $WE=0 \rightarrow$ the output is kept constant.

Content

Sequential behaviors

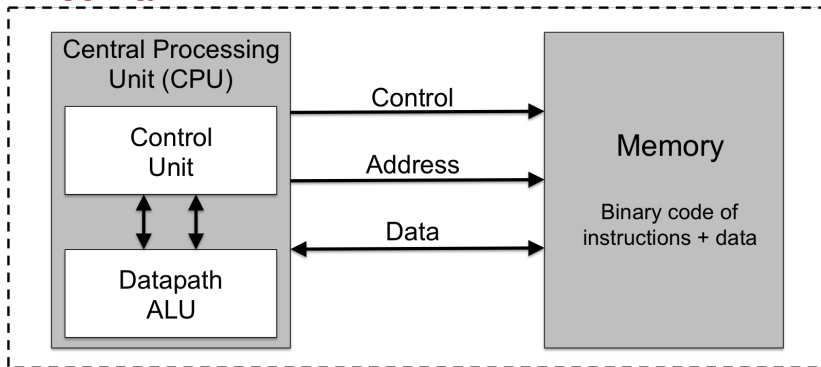
Registers

n-bit Registers

Addressable memories

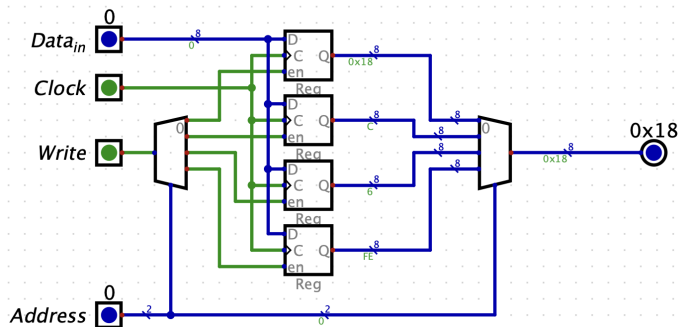
Memory hierarchy

Von Neumann



- ▶ In a von Neumann machine, memories are used to store data and instructions
- ▶ Memories contain several words (actually a lot!) identified by their **address**
- ▶ One word can be read/written at a time depending on the **control** signal

From registers to memories



Such memory is called an **SRAM** (Static Random Access Memory²). It can be viewed as a collection of memory cells, each identified by a number called its **address**.

Each cell stores a fixed number of bits (typically 8 bits).

²“Random Access” means that any memory location can be accessed directly, regardless of its position in memory.

Why Is an Address Important?

- ▶ The concept of **address** is central to almost every use of a computer.
- ▶ An **address** is “simply” the number that identifies a memory cell.
- It tells the computer **where** an information is stored, so that it can be found later.

Where vs. What

An address tells us **where** to find the information; the value stored at that address tells us **what** the information is.

Why does this matter?

Whenever a computer needs to use an information, it must know **where** that information is stored.

Losing the “where” means losing the ability to find the information. Exactly as if the information had been erased.

The pro/cons of SRAM

- ▶ SRAM uses essentially the same technology as computation circuits, *i.e.*, logic circuits.
 - This makes SRAM extremely fast (basically as fast as computation circuits)
- ▶ But SRAM is **expensive**: each bit requires many transistors.

SRAM is **very fast**, but it is too expensive and power-hungry to be used for large amounts of memory.

⇒ We'll need other kind of technologies to build large memories...

Dynamic RAM (DRAM³)

- ▶ **Dynamic RAM (DRAM)** uses capacitors to store bits, with a single transistor controlling access to each cell.
 - ▶ **Advantage:** memory cells are small and cheap, allowing us to build very large memories.
 - ▶ **Drawback:** the charge stored in the capacitor leaks slowly.
 - The memory cells must therefore be **refreshed periodically** (typically on the order of tens of milliseconds).
 - ▶ **Drawback:** access is significantly slower than with SRAM.
 - A DRAM cell can be accessed in **several tens of ns**, compared with a few ns for SRAM.
 - But accessing **large memories** also involves slow data transfer operations
 - As a result, the effective latency of large DRAM access can be **hundreds of times larger** than that of a SRAM access.

The trade-off: DRAM is **slower than SRAM**, but its cells are much **smaller and cheaper**, making it suitable for very large memories.

³Warning: SDRAM (Synchronous DRAM) is a DRAM, not an SRAM!

Capacity/Latency trade-off

- ▶ a fast memory is small / a big memory is slow
⇒ Memory-Wall problem

This problem is as old as computer science and but it is not (not only) a technological problem: it is due to the laws of physics and so cannot be overcome!

We need to bypass it → Memory Hierarchy

Ideally one would desire an indefinitely large memory capacity such that any particular [...] word would be immediately available. [...] We are [...] forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible.

*John von Neumann,
Preliminary Discussion of the Logical Design
of an Electronic Computing Instrument, 1946*

Content

Sequential behaviors

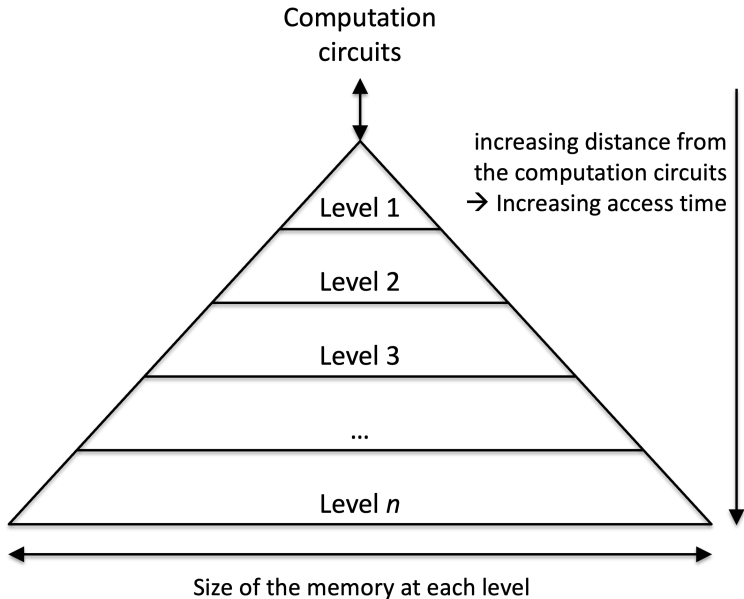
Registers

n-bit Registers

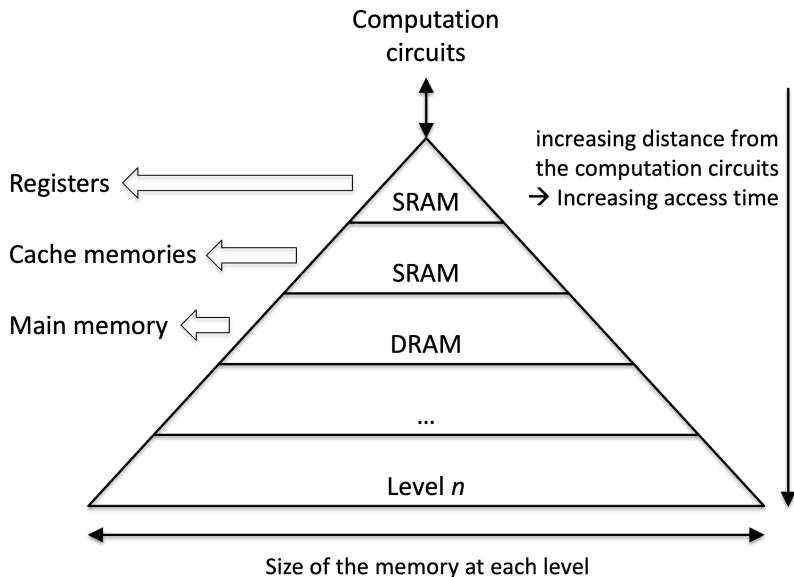
Addressable memories

Memory hierarchy

Memory Hierarchy



Memory Hierarchy



Registers

What is a register?

A **register** is a small memory element that stores a single word of data. Registers are located at the same place than the computing machinery (i.e. inside the CPU) and are directly connected to the computing circuits.

- ▶ Registers are built from **flip-flops**, just like the storage elements used to build SRAM.
- ▶ They can be accessed **extremely quickly**, typically in a single computation cycle.
- ▶ A computer contains a relatively **small number of registers** compared with the amount of data stored in memory.

The key idea: Registers are the **fastest storage** available to the computing machinery, but they are very limited in number and capacity.

Cache Memories

Small SRAM elements that exploit the principle of **data locality** to accelerate access to frequently used data.

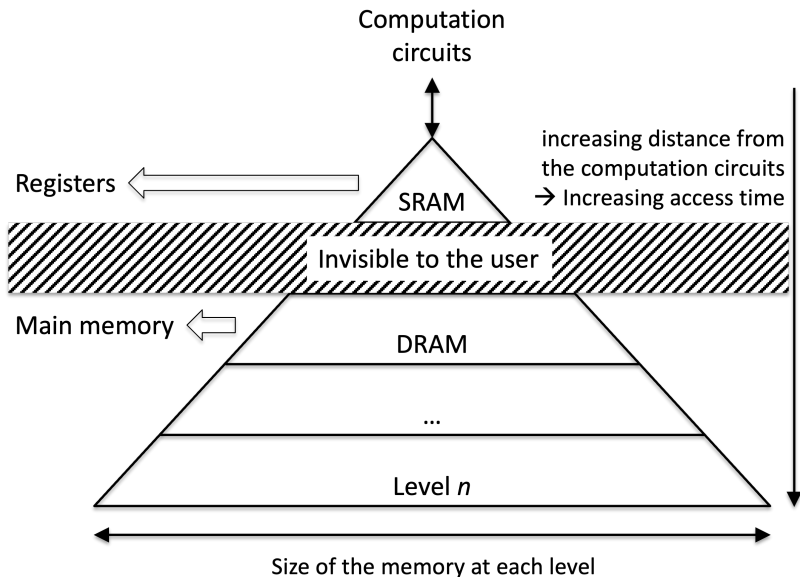
- ▶ **Temporal locality:** If a piece of data is accessed once, it is likely to be accessed again soon.
- ▶ **Spatial locality:** If a piece of data is accessed, nearby data in memory is likely to be accessed soon.

Modern computers typically have **three levels of cache**. L1 is usually divided into separate instruction and data caches:

- ▶ **L1** – very small and very fast
 - ▶ There are two L1: L1-I (for instructions) and L1-D (for data)
 - ▶ Typically tens of KB per core, access time $\approx 1\text{--}2$ ns
- ▶ **L2** – larger, but slower
 - ▶ Typically hundreds of KB per core, access time $\approx 3\text{--}10$ ns
- ▶ **L3** – much larger, but slower still
 - ▶ Typically several MB, often shared, access time $\approx 10\text{--}30$ ns

Cache memories are managed automatically by the hardware. They are normally invisible to the programmer, but have a major impact on computer performance ($\rightarrow$ cache miss).

Memory Hierarchy



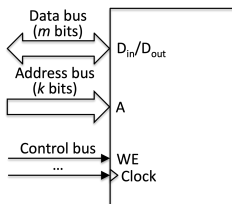
Main Memory

Main memory, or **RAM (Random Access Memory)**, is the memory used to store all the information that is currently being used by the computer (*i.e.* programs and data).

- ▶ Main memory is directly accessible by the computation machinery.
- ▶ It is typically implemented using **DRAM**:
 - ▶ much denser and cheaper than SRAM;
 - ▶ large enough to store programs and large amounts of data;
 - ▶ but much slower than the CPU (hence the use of cache memories).
- ▶ Each memory element has an **address** that allows the CPU to read or write its contents.
- ▶ Main memory is **volatile**:
 - ▶ its contents are lost when the computer is powered off.

Its role in the memory hierarchy is to provide a good compromise between **capacity, cost, and access speed**.

Main memory in practice



the main memory is a digital circuit that receives three inputs:

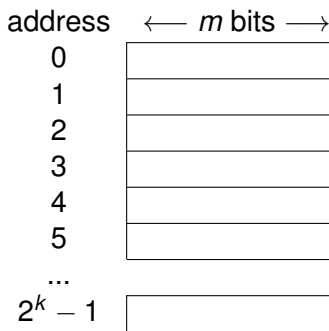
- ▶ Data bus (D_{in}/D_{out}) to send/receive data on m bits (m being generally a multiple of 8). **m is the size of the words stored in this memory.**
- ▶ Address bus (A). The address is an integer encoded on k bits (k being generally a multiple of 8). **The maximum number of words in the memory is 2^k .**
- ▶ Control bus. Its content is architecture dependent but it contains at least a WE (Write Enable) signal and a $Clock$.

if $WE=1$, the input data is written in the memory at address A .

If $WE=0$, the word stored at address A is written on the data bus.

Theoretical memory organization

In theory, a memory is a **Vector** of 2^k elements of m bits each.
With k address bits we should have:



But in practice, such memories are difficult to implement...

- ▶ Memories are generally made of 8-bits cells
- ▶ Addresses correspond to 8-bits elements but several ($l = m/8$) elements are read simultaneously

Actual Memory Organization

Suppose that the memory uses k -bit addresses and stores **4-byte words** ($l = 4$, $4 \times 8 = 32$ bits per word)

	$\leftarrow 8 \rightarrow$	$\leftarrow 8 \rightarrow$	$\leftarrow 8 \rightarrow$	$\leftarrow 8 \rightarrow$
address	byte 0	byte 1	byte 2	byte 3
0				
4				
8				
$\vdots$				
$2^k - 4$				

- ▶ It is **byte-addressable**: every byte has its own address.
- ▶ A 4-byte word therefore occupies four consecutive addresses: 0–3, 4–7, 8–11, ...
- ▶ The starting address of each word is a multiple of 4. These are called **word-aligned addresses**.

Why is it important?

- ▶ k indirectly specify the maximum memory capacity of a machine:

$$k \text{ address bits} \Rightarrow 2^k \text{ addressable bytes} \Rightarrow \frac{2^k}{I} \text{ words.}$$

- ▶ Example:

- ▶ Modern PCs use 64-bit addresses such that there are $2^{64} = 18\,446\,744\,073\,709\,551\,616$ possible addresses.

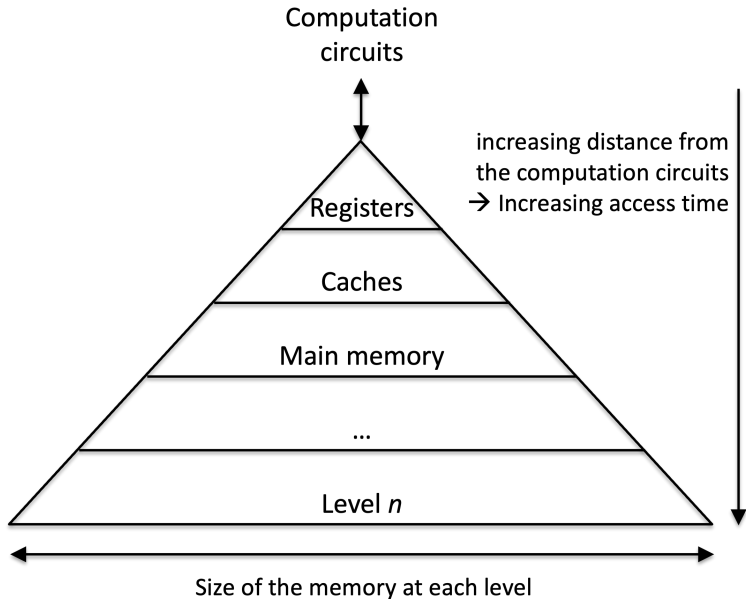
- ▶ For byte-addressable memory it gives:

$$2^{64} \text{ bytes} = 16 \text{ EiB (exbibyte)} \quad (1 \text{ EiB} = 1.15 \times 10^9 \text{ GiB})$$

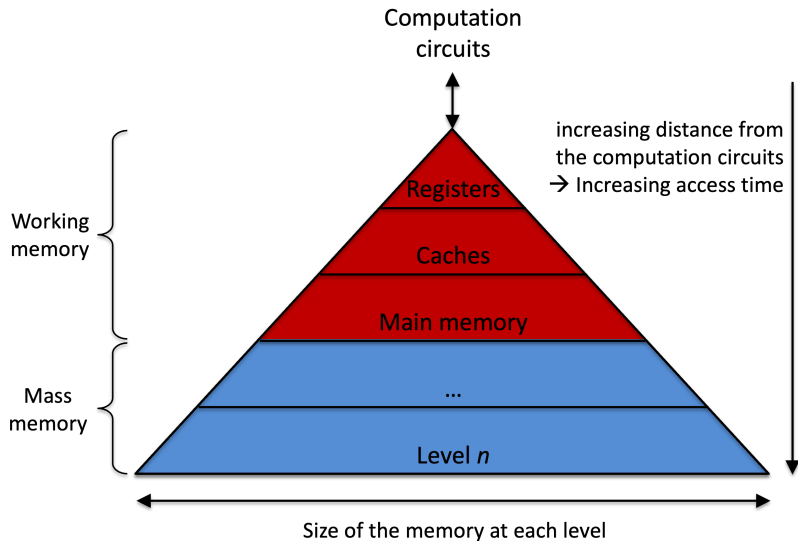
- ▶ But a 64-bit PC does **not** need 16 EiB of memory (actually this is much larger than the amount of memory that can actually be installed). So machines use only part of this address space (typically 8, 16 or 32 GiB).

But the maximum memory capacity can still be an issue for smaller machines using 16 or 32 address bits...

Memory hierarchy so far...



Memory hierarchy so far...



Working Memory vs. Mass Memory

Working memory

Used **directly and actively** while programs are running.

- ▶ **Registers**

- ▶ Very small and extremely fast

- ▶ **Cache memories**

- ▶ SRAM
 - ▶ Small, very fast

- ▶ **Main memory (RAM)**

- ▶ DRAM
 - ▶ Much larger, but slower

Volatile: contents are lost when power is removed.

Mass memory

Memory used to store **large amounts of information** for long periods of time.

- ▶ **SSD**

- ▶ “Flash” memory
 - ▶ Fast (but slower than RAM)

- ▶ **Hard disk (HDD)**

- ▶ Mechanic/magnetic storage
 - ▶ Large and inexpensive

Non-volatile: contents are preserved without power.

A fundamental trade-off: **Working memory**: fast, small, volatile
Mass memory: slower, large, persistent

Hard Disk Drives (HDD)

A **hard disk drive (HDD)** stores information magnetically on one or more rotating disks called **platters**.

A movable **read/write head** accesses the data on the surface of the platters.



Hard Disk Drives (HDD)

A **hard disk drive (HDD)** stores information magnetically on one or more rotating disks called **platters**.

A movable **read/write head** accesses the data on the surface of the platters.

- ▶ **Non-volatile:** data is preserved when the power is off.
- ▶ **Very high capacity:** several TB can be stored at a relatively low cost.
- ▶ **Mechanical:** the platters must rotate and the read/write head must move to the required position.
- ▶ **Slow random access:**
 - ▶ The head must physically move to the appropriate location.
 - ▶ Typical access latency is on the order of **milliseconds**.
- ▶ **Good sequential access:**
 - ▶ Once the head is positioned, consecutive data can be read relatively efficiently.

HDDs provide **large, inexpensive, non-volatile storage**, but are much **slower** than memory because they rely on **mechanical movement**.

SSD and Flash Memory

Flash memory

Flash memory stores information using electrical charge trapped inside specially designed transistors.

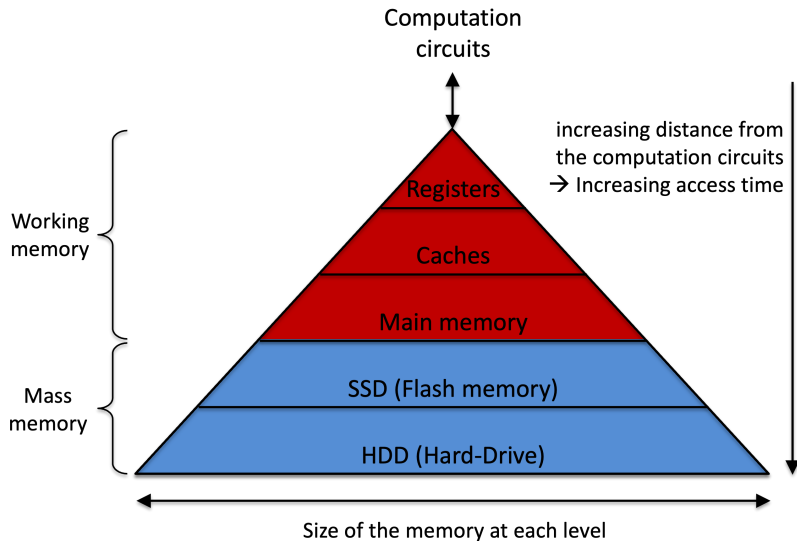
- ▶ **Non-volatile:** data is preserved without power (unlike SRAM and DRAM).
- ▶ **High density:** many bits stored in a small physical space.
- ▶ **No moving parts:** unlike hard disks.
- ▶ **Slower than RAM:** accessing Flash is much slower than accessing DRAM (but faster than HDD)
- ▶ **Limited write endurance:** Flash cells can only be erased and rewritten a finite number of times.

SSD = Solid-State Drive

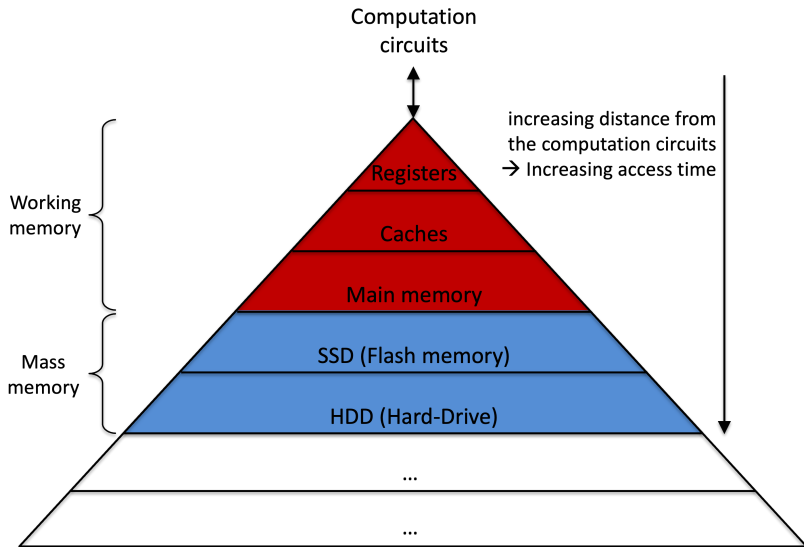
An SSD is a storage device built primarily from **Flash memory**, together with a controller that manages the Flash chips (including write endurance) and presents them to the computer as a storage device. SSD are faster than HDD because they don't rely on mechanical movement.

Conclusion

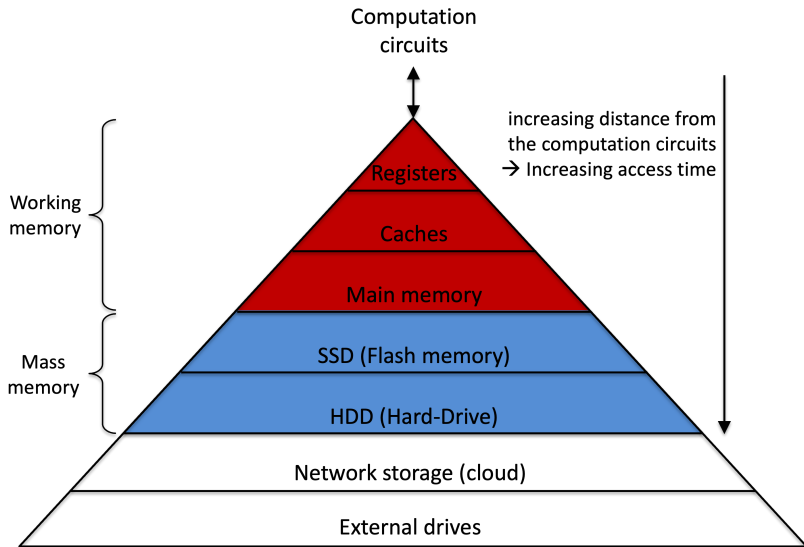
A memory hierarchy is typically composed of 5 different levels:



Are there lower levels?



Yes... But that's another story...



Memory hierarchies in practice

In modern systems, a typical memory hierarchy is generally composed of 6+ different levels with three main groups:

1. Registers ($\sim 1\text{KB}$, access time $< 1\text{ ns}$)
2. Cache memories (SRAM)
 - ▶ L1 (16KB - 128KB per core, $\sim 1\text{ ns}$)
 - ▶ L1-D
 - ▶ L1-Ins
 - ▶ L2 (256KB - 1MB per core, $\sim 3\text{--}10\text{ ns}$)
 - ▶ L3 (2MB to 32MB (shared), $\sim 10\text{--}30\text{ ns}$)
3. Main memory (DRAM) ($\sim 8\text{--}32\text{ GB}$, $\sim 50\text{--}150\text{ ns}$)
4. Flash Drive ($\sim 200\text{ GB} - 1\text{ TB}$, $\sim 10\text{--}100\text{ }\mu\text{s}$)
5. Hard Disk ($\sim 1\text{--}5\text{ TB}$, $\sim 5\text{--}15\text{ ms}$)
6. Network/Internet storage, external storage (virtually unlimited but access time depends on the network and the storage system)

The exact sizes and access times vary significantly between systems. Try `lscpu` on Linux or `sysctl` on macOS.

A library analog



Orders of magnitude...

Imagine that the machine is a student looking for a book in a library. The closer the information is to the student, the faster it can be accessed.

Computer	Library analogy	Access time
Registers	Book opened in your hands	same as reading
L1 cache	Book ready on your desk	one second
L2 cache	Book on a trolley next to you	a few seconds
L3 cache	Book on a nearby shelf	tens of seconds
DRAM	Book somewhere in the library	several minutes
SSD	Book in another library	days
HDD	Book in another country	weeks/months

Key ideas: (1) Fast access requires keeping information close to the user. (2) Mass storage devices are too slow to be used as working memory. (3) Information is moved on demand so that frequently needed information is quickly accessible.