Registers and memories

Lecturer: Guillaume Beslon
(Lecture adapted from Lionel Morel)

Computer Science and Information Technologies - INSA Lyon

Fall 2023

Content

Why Sequential circuits?

A combinatorial circuit implements a function (in the mathematical sense of the term):

- For a given input value, it **always** produces the same output, whatever input values it received in the past.

Question

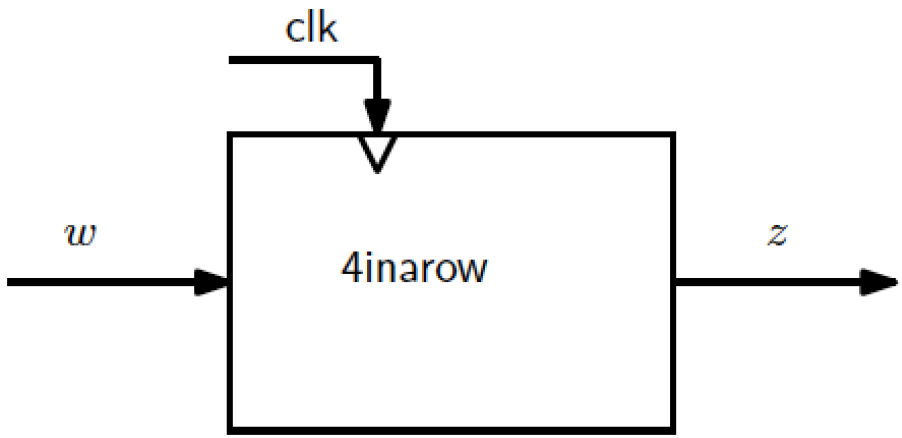
Sometimes, we'd like to keep some information within a circuit, when inputs change.

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

$\Rightarrow$ need memory to "remember" occurrences of w.



Sequential circuits

A sequential element keeps a state.
So a sequential circuit is not a function:

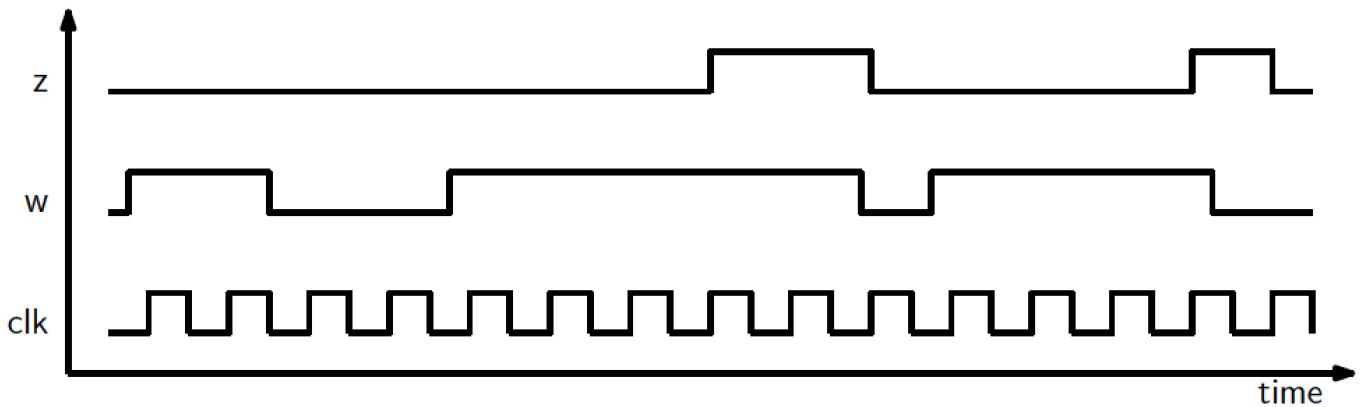
- for a given input value, the output value depends on:
  - the **current** input value
  - the **sequence of input values** that have been **received in the past**

$\Rightarrow$ Representing a sequential circuit with a truth table is impossible

Representing sequences of values

Two forms:

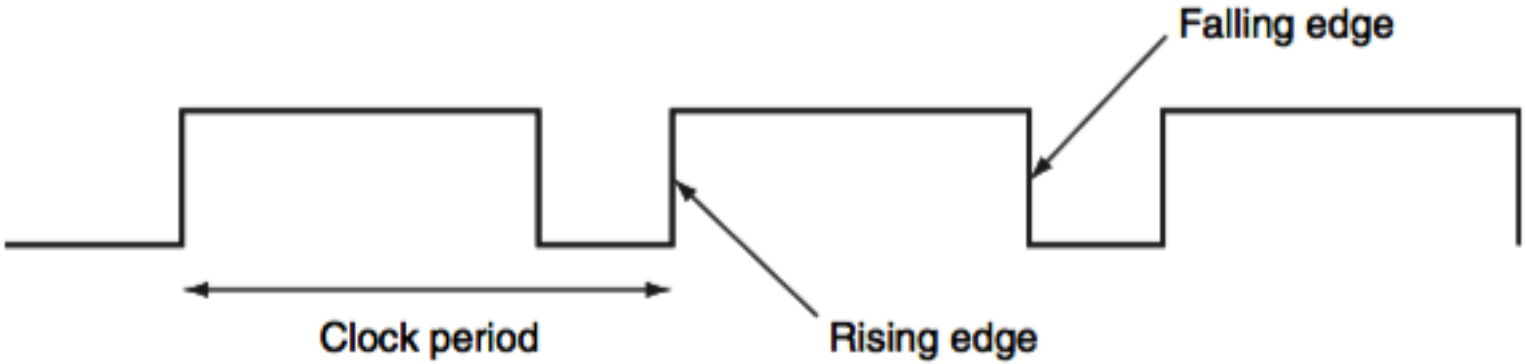
- time-diagrams, called **chronograms**
  - → Chronograms describe graphically a **specific** sequence of input/output values
  - → A chronogram is only a **representative example** of the circuit's behavior
  - → Example: 4-in-a-row

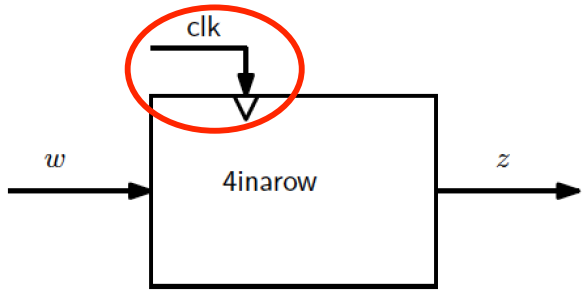


- State machines (FSM):
  - → Describe the full system's behavior **formally**
  - → Next lecture

Clocks

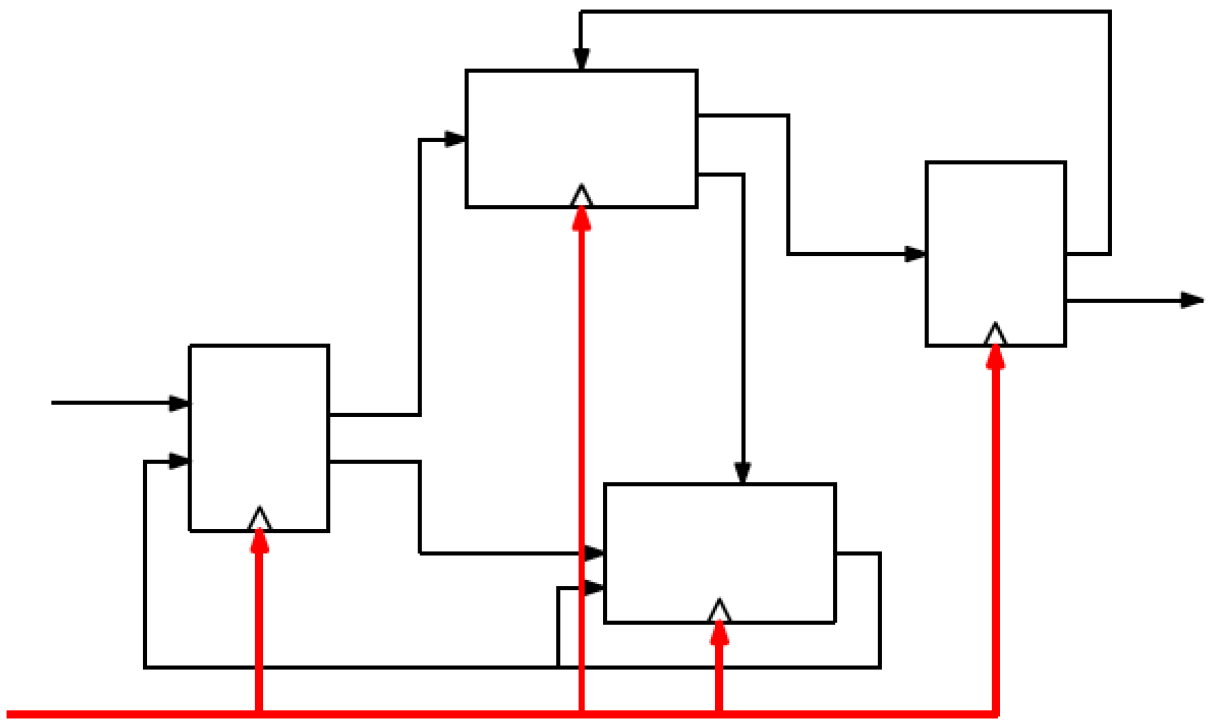
- 99% of existing processors are **synchronous**
- A synchronous sequential circuit receives a **clock signal**





Synchronous circuits

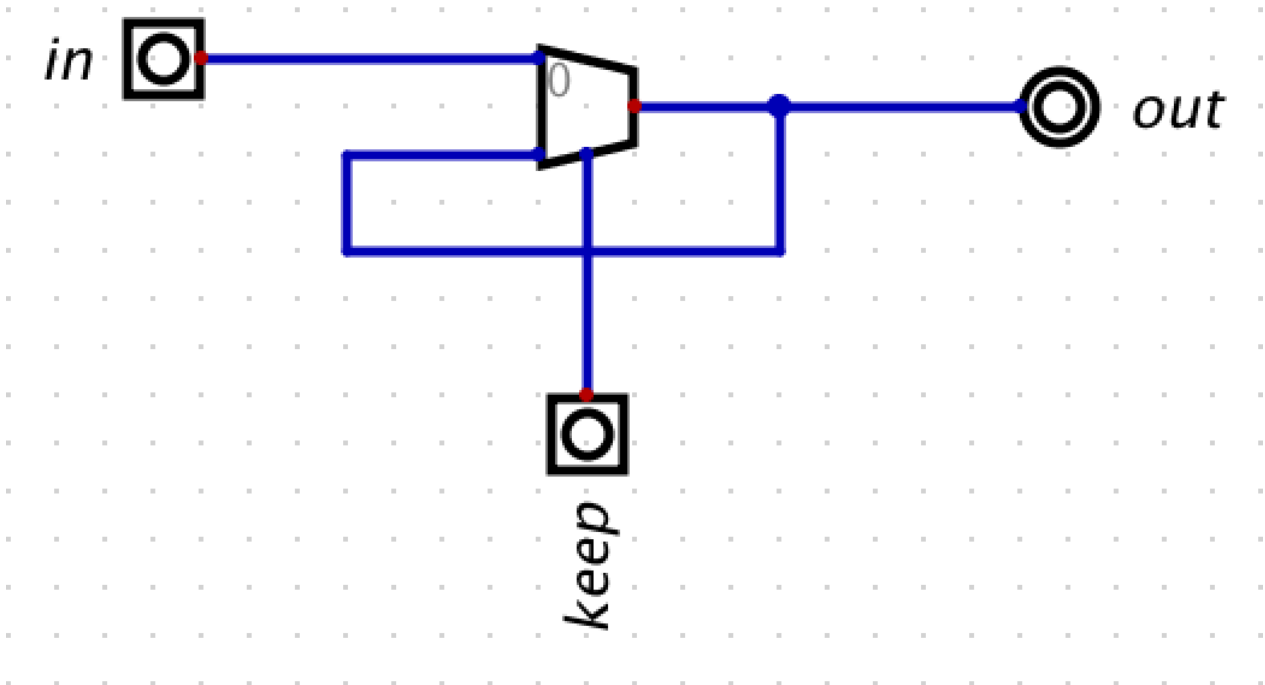
- 99% of existing processors are **synchronous**
- A synchronous sequential circuit receives a **clock signal**
- signals can then be synchronized, either on rising or falling edges of the clock
- clock ticks are supposed to be long enough to mask propagation delays.



Content

D-Latch (Verrou, bascule asynchrone)

- Specifications:
  - When *Keep* $= 0$, copy IN to OUT
  - When *Keep* $= 1$, stay unchanged
- Can easily be implemented using a MUX (multiplexer)



! Care you really understand the behavior of this circuit !

We need more

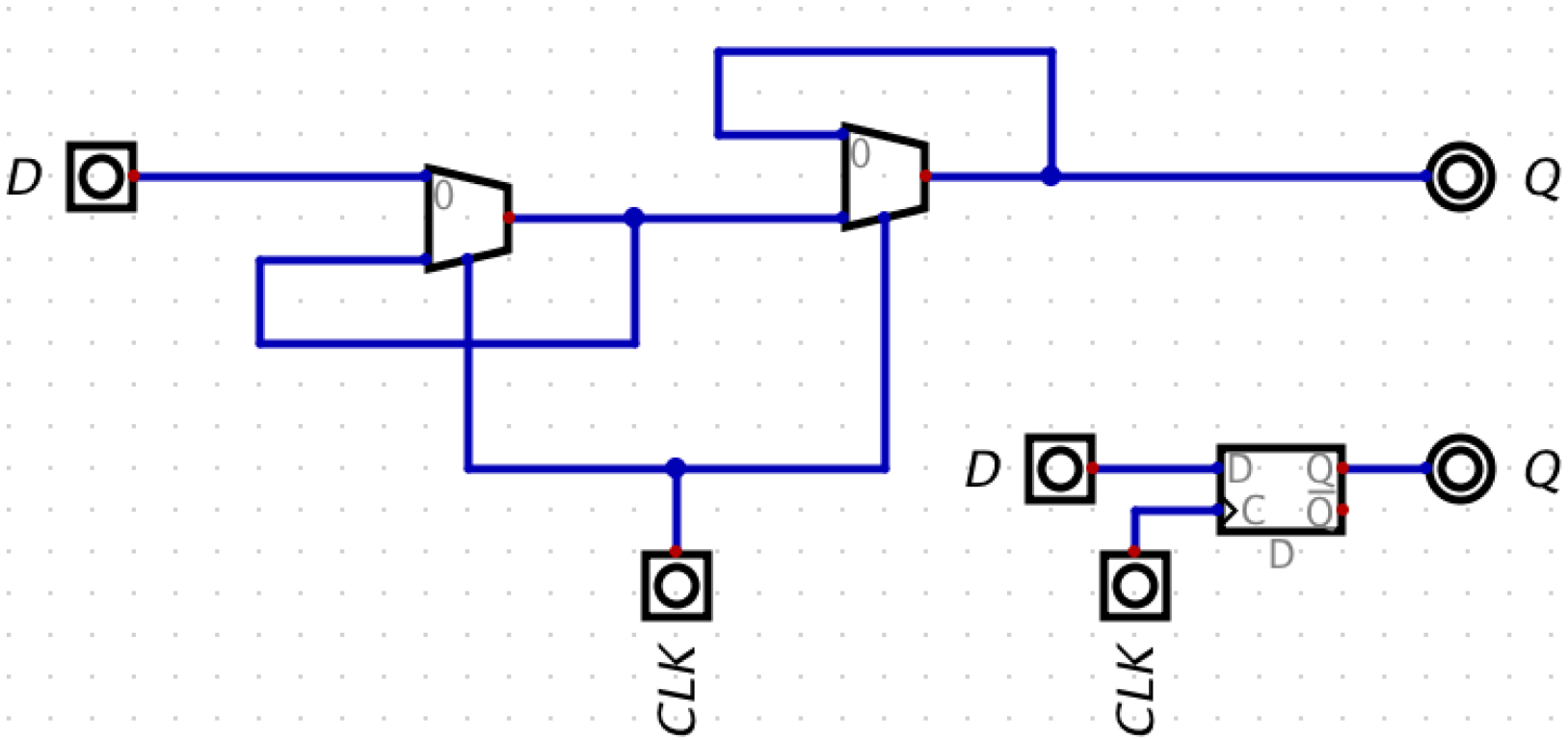
A latch's output changes **whenever the keep signal is up**...

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-FlipFlip) which behave differently

We need more

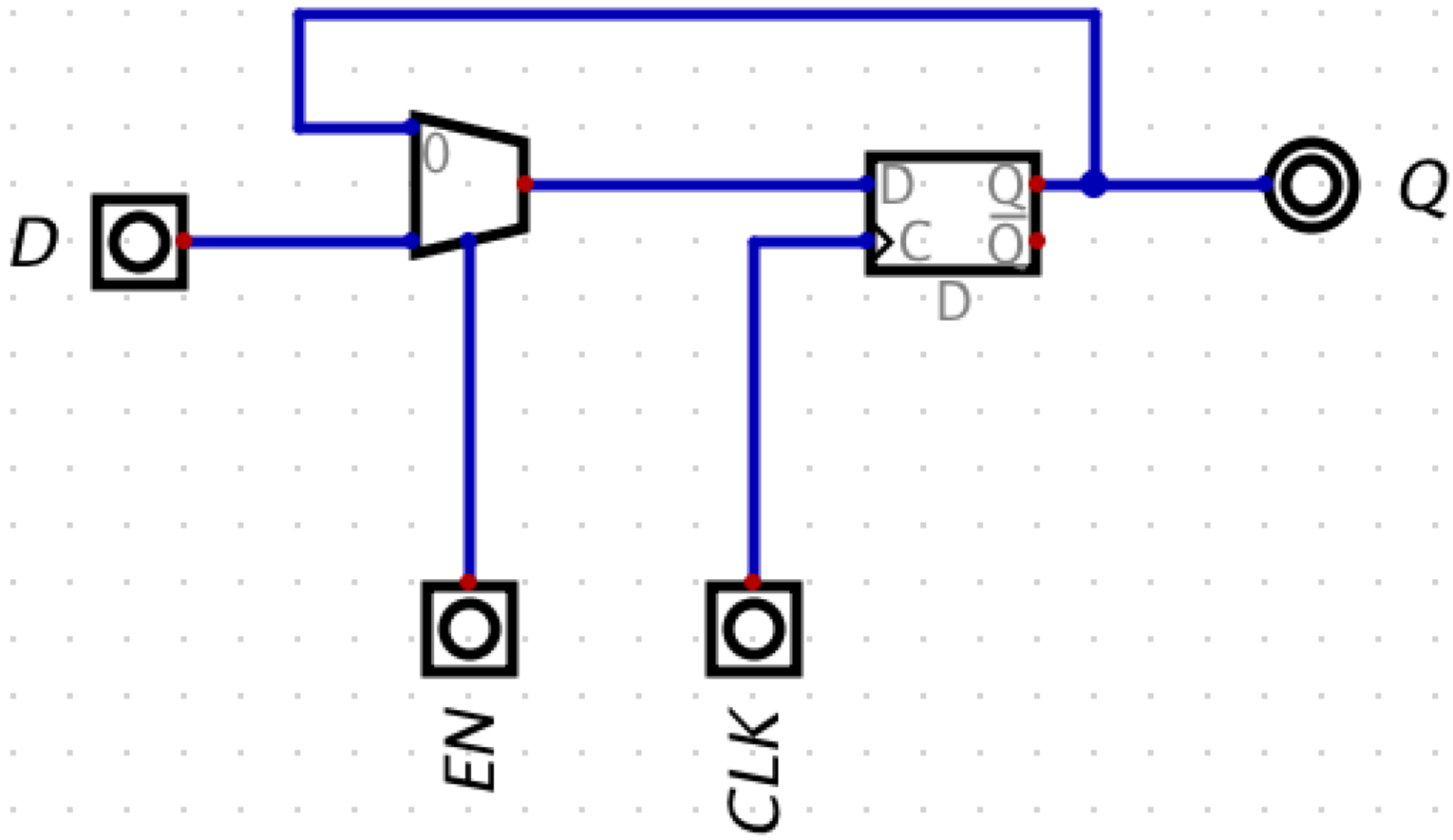
A D-Flip-Flop stores the D input **On each rising edge of the clock signal**...

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

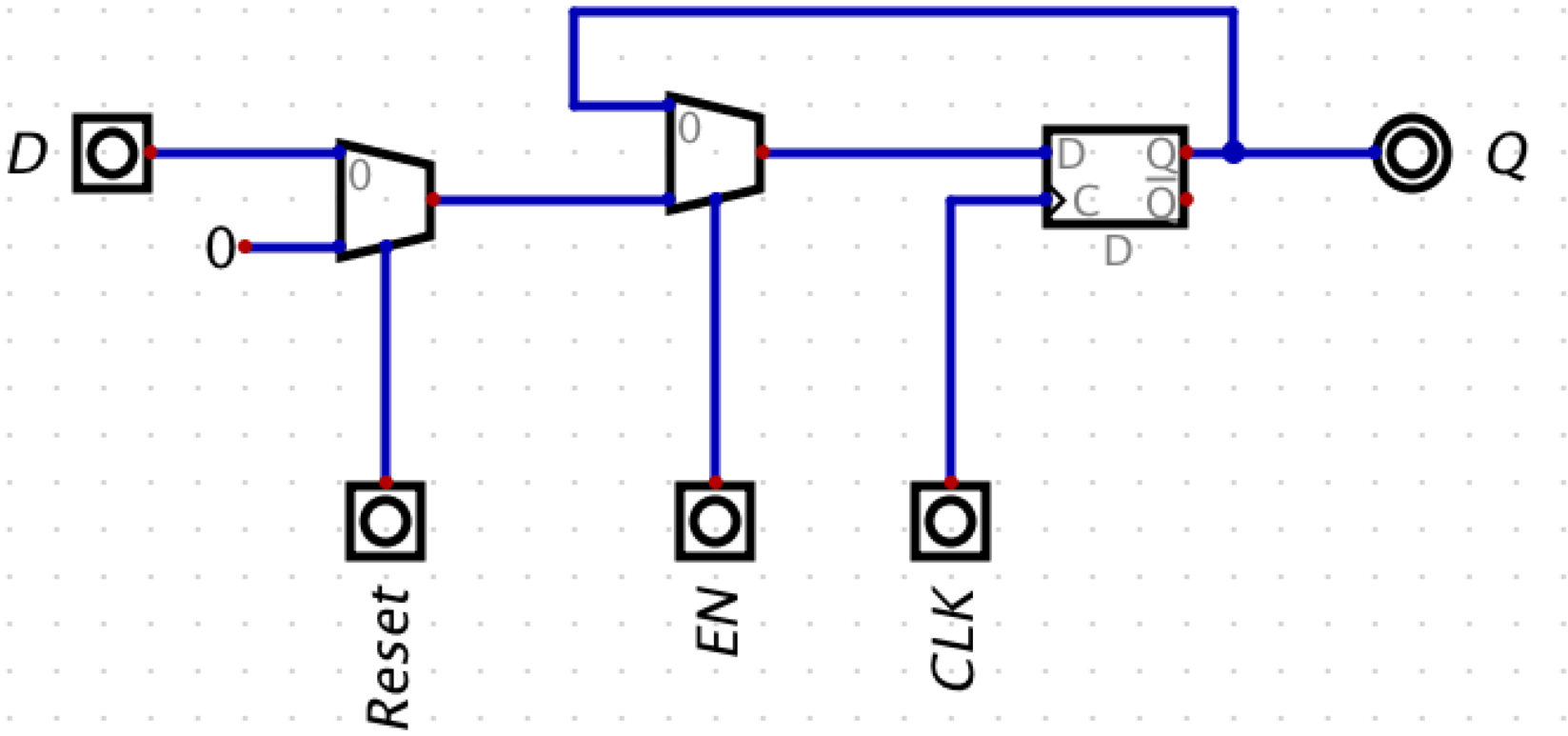
This can be implemented by adding a MUX to the Flip-Flop...

Write-enabled flip-flop



Reset command

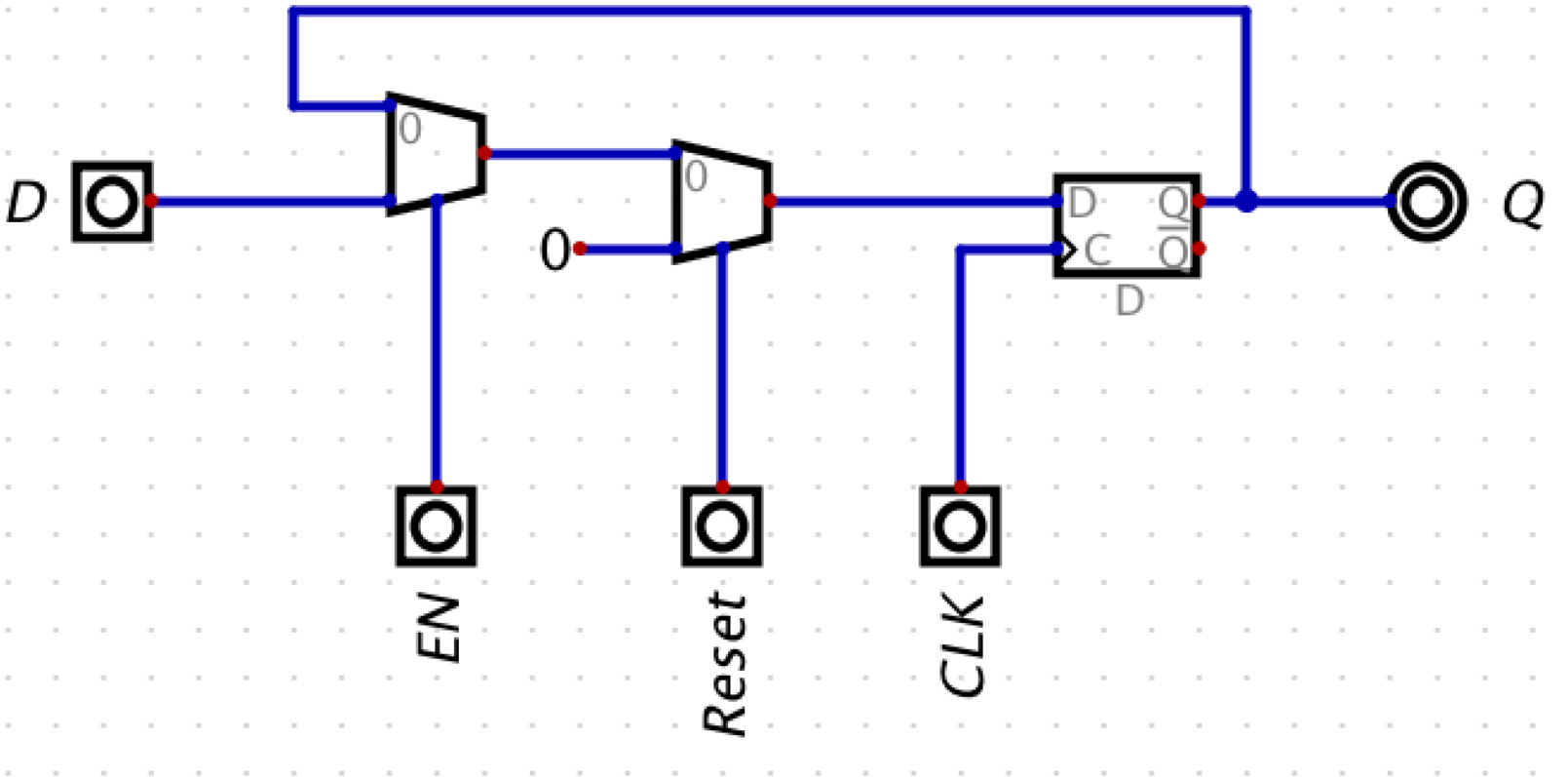
With the same kind of idea, one can add a RESET command to the Flip-Flop



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

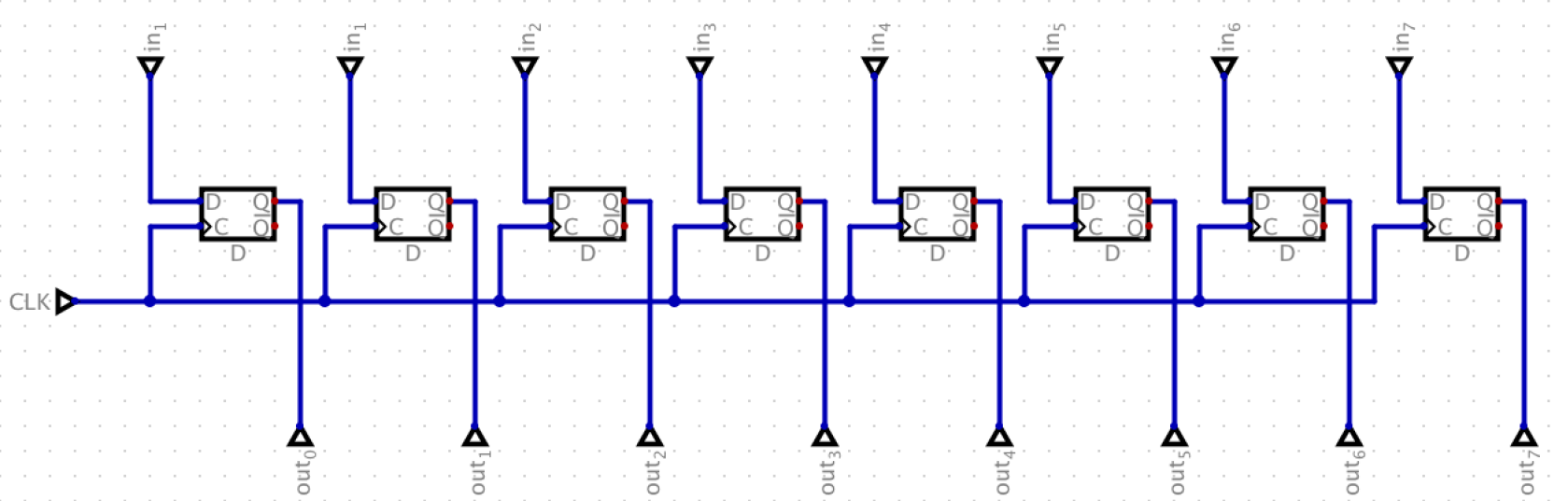


Here the Reset signal is effective whatever the state of the EN signal...

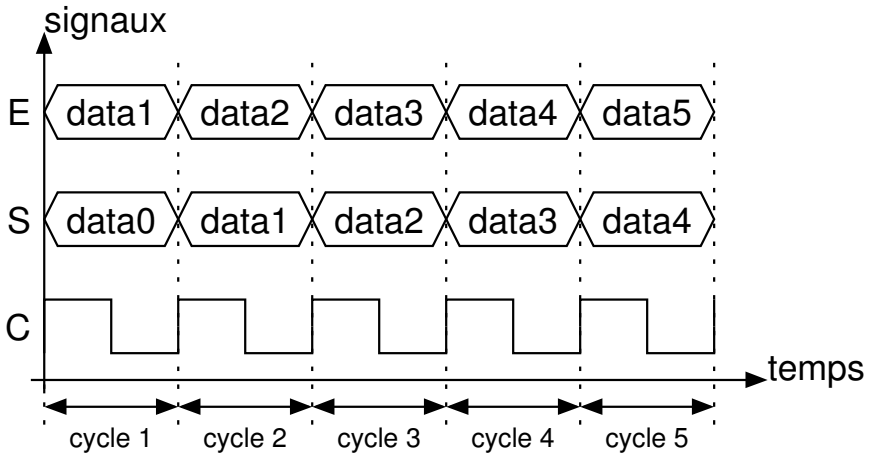
Content

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):



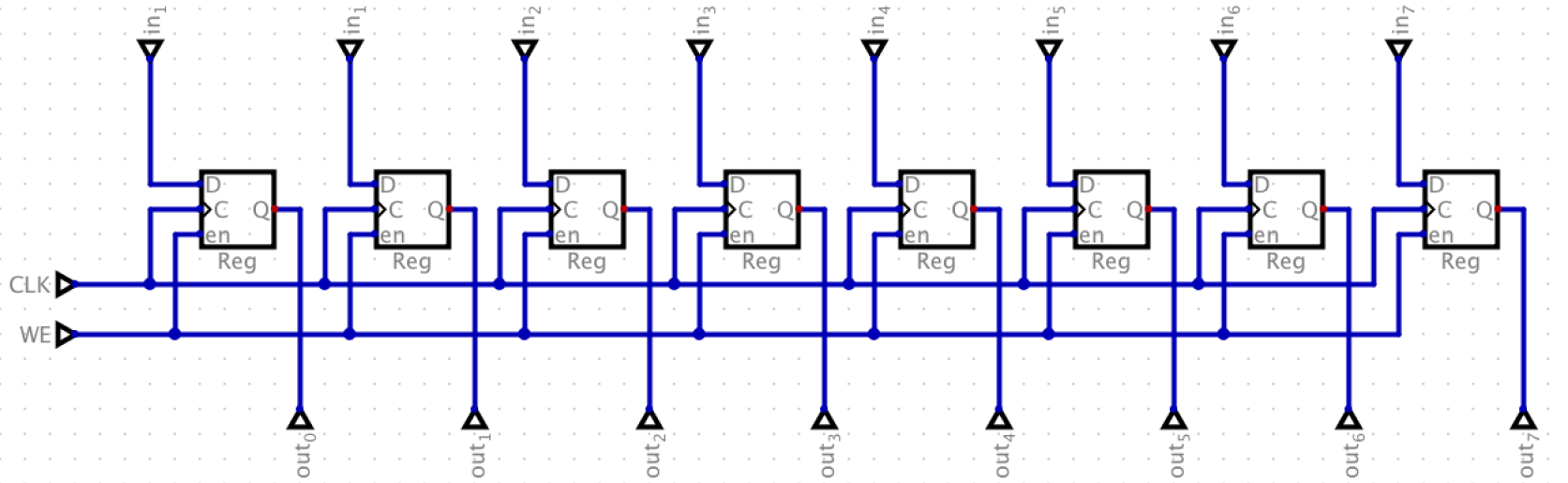
Chronogramme:



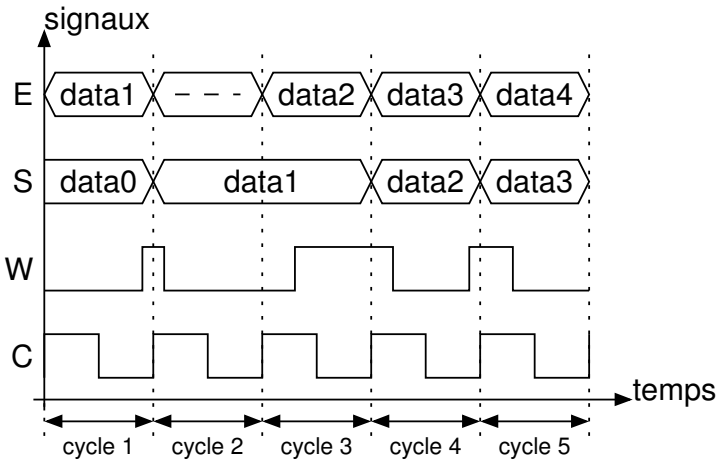
- The input is stored at the CLK rising hedge; the output is stable during the whole CLK cycle
- Between two CLK rising hedges, all variations of the input signal are ignored

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:

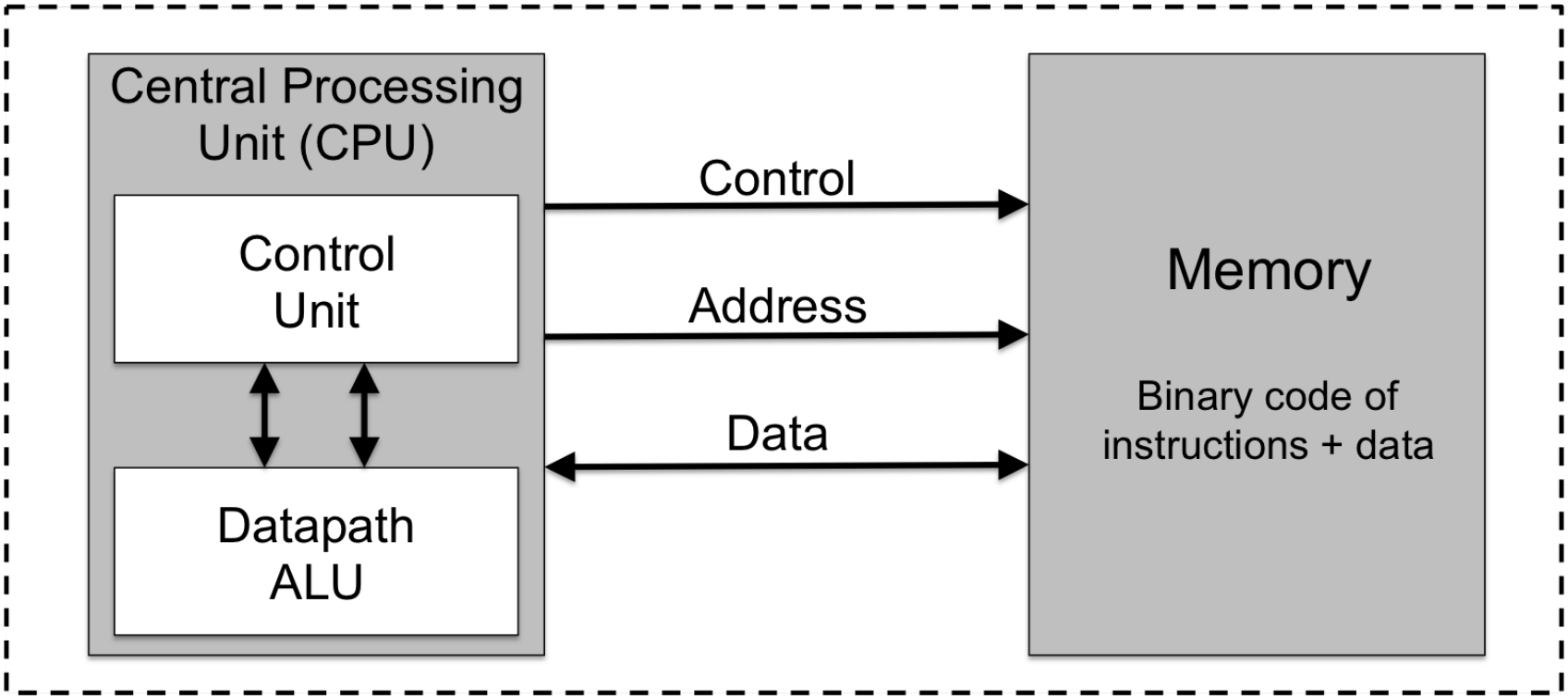


At the end of each cycle:

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

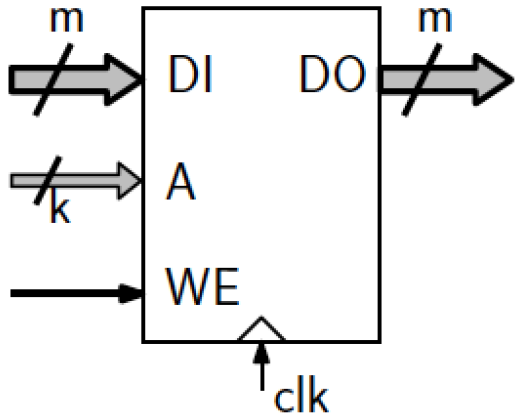
Content

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

Memory in general



Interface

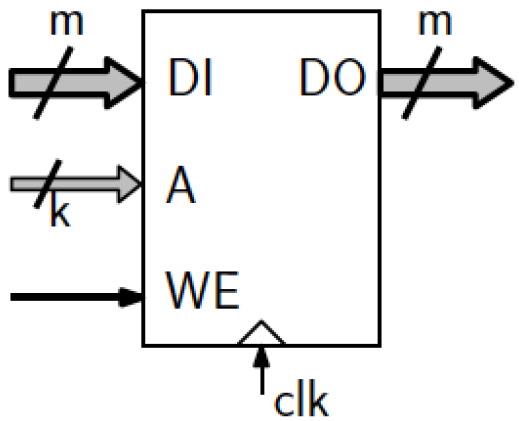
A: addresses on k bits for a memory of 2^k words

DI: Data Input, on m bits (generally a multiple of 8). The input will be written to word (aka memory cell) of address A is WE is true

DO: Data Output, always outputs the m bits contained in the memory cell of address A.

WE: Write Enable

REMEMBER



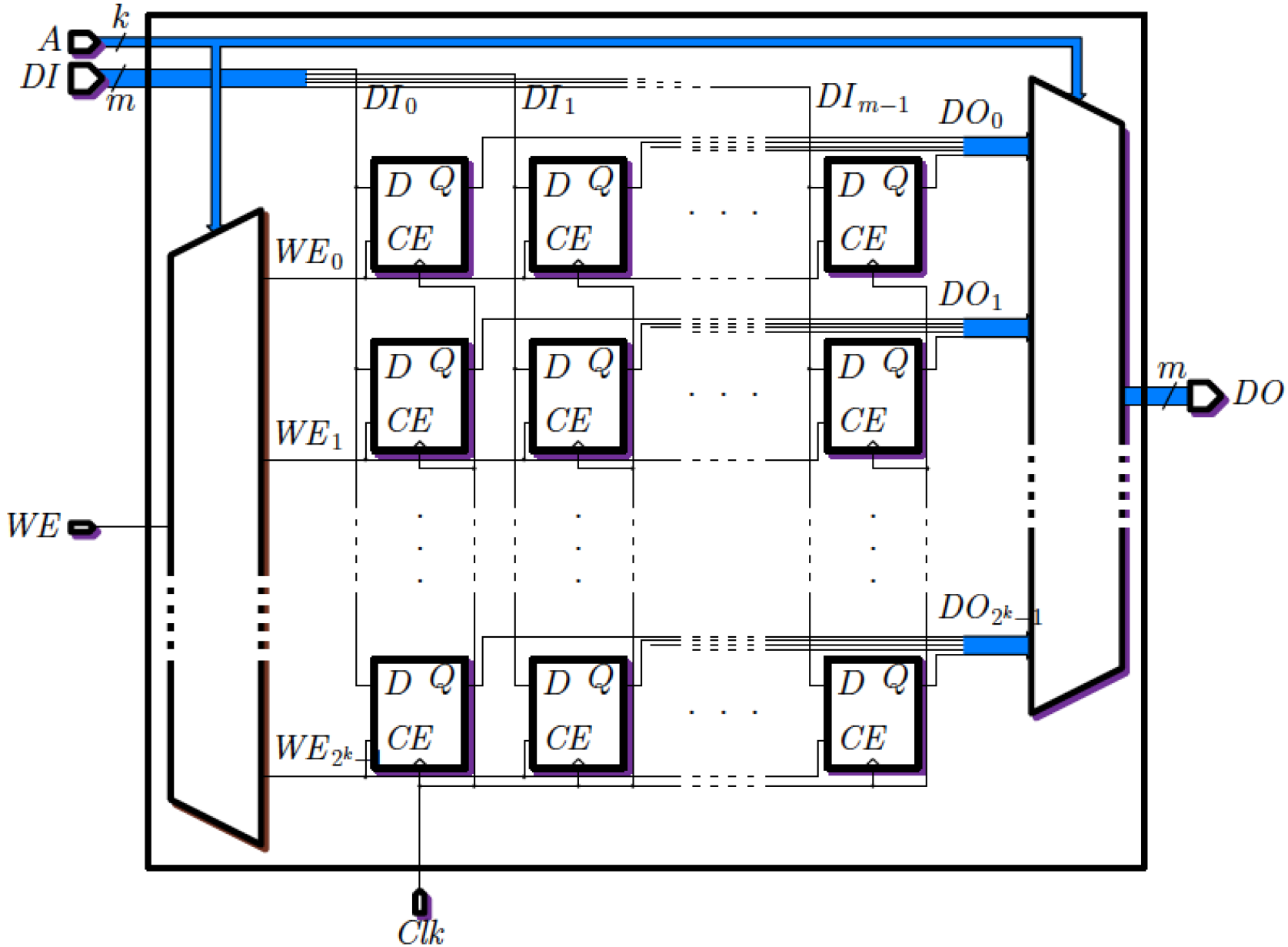
Size of the memory

k specifies the number of words (2^k)

m is the size of the words (in bits).

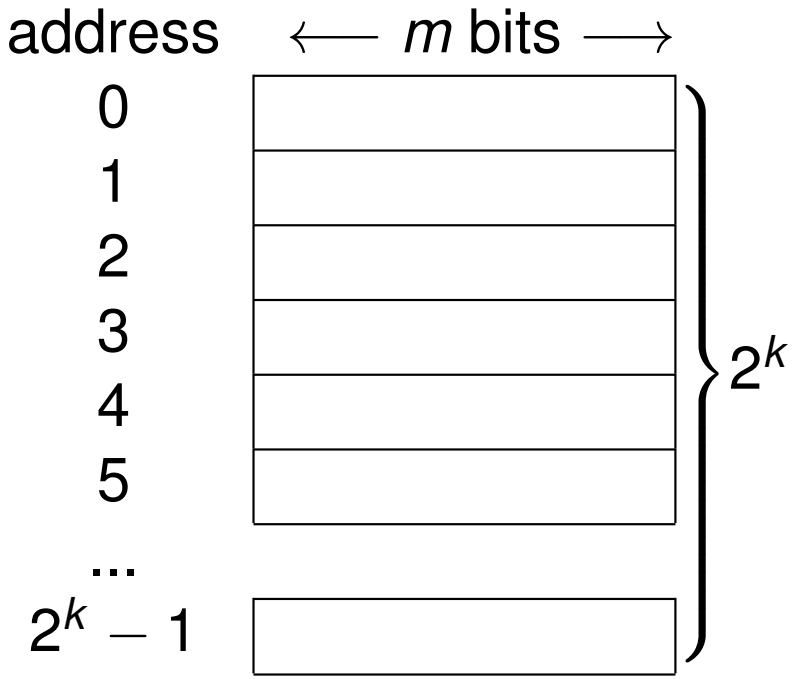
- The capacity of the memory is $m \times 2^k$ bits
- If $m = l \times 8$, the capacity of the memory is $l \times 2^k$ bytes

Memory: possible implementation



Memory as seen from Software

A memory is a **Vector** of 2^k m bits elements.
With k address bits we have:

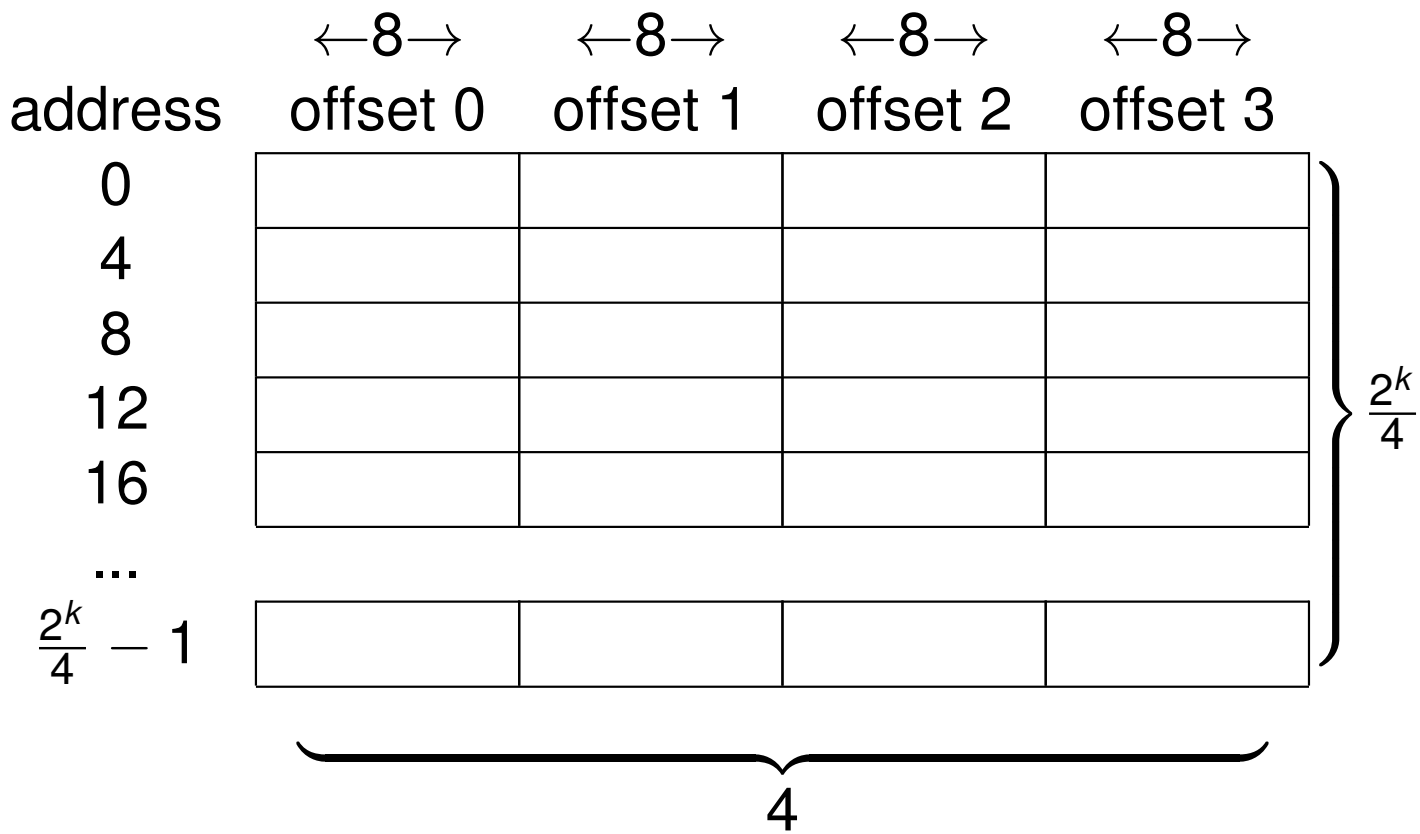


But in practice, such memories are difficult to implement...

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

Hardware Organization

With k address bits and $l = 4$:



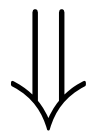
Von Neumann on Memory

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.

Preliminary Discussion of the Logical Design of an Electronic Computing Instrument, 1946

Memory Hierarchy

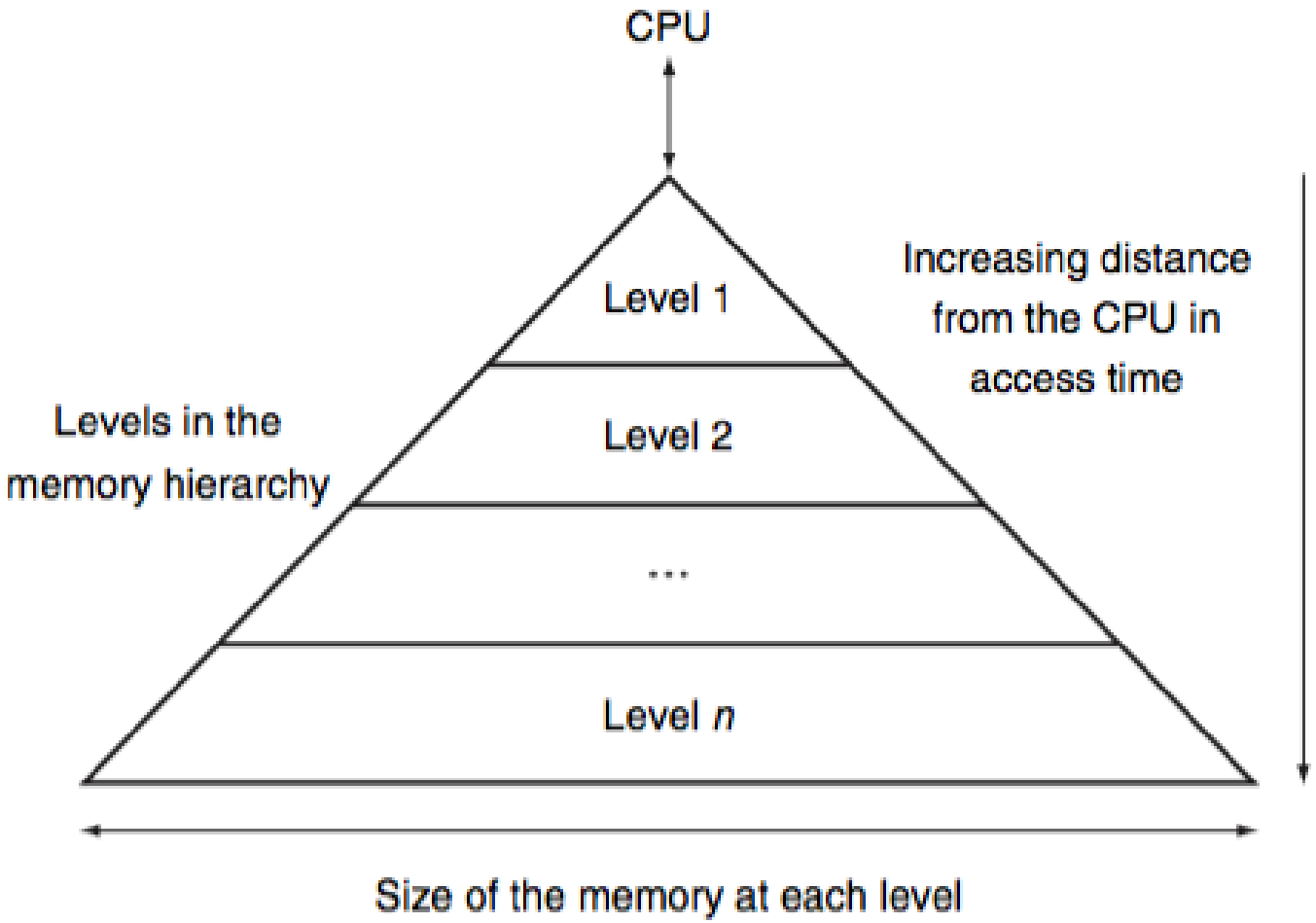
Capacity/Latency trade-off



- a fast memory is small
- a big memory is slow

This is due to the laws of physics and cannot be overcomed!

Memory Hierarchy

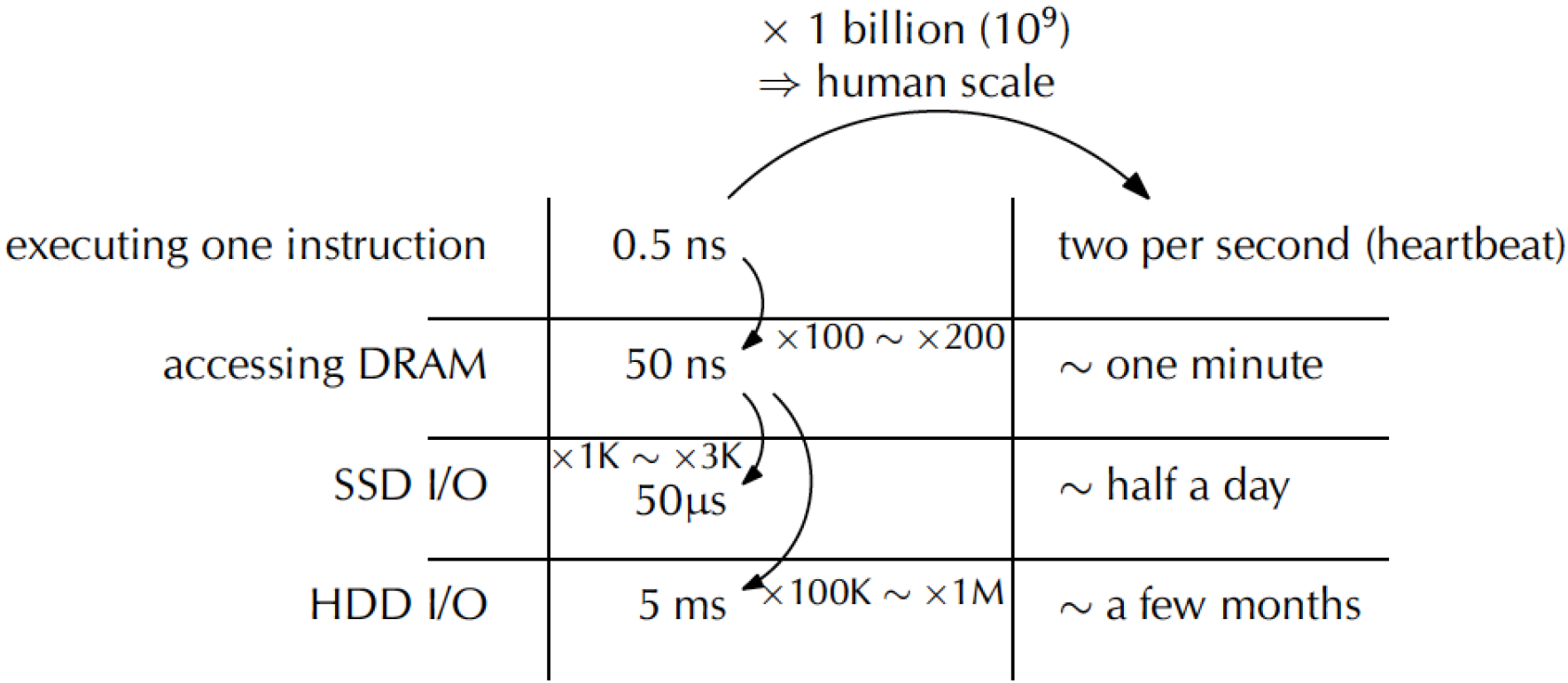


Memory Hierarchy[1]

Memory technology	Typical access time	$ per GiB in 2012
SRAM semiconductor memory	0.5–2.5 ns	$500–$1000
DRAM semiconductor memory	50–70 ns	$10–$20
Flash semiconductor memory	5,000–50,000 ns	$0.75–$1.00
Magnetic disk	5,000,000–20,000,000 ns	$0.05–$0.10

[1] Numbers as of 2012

Latencies — order of magnitude



Memory hierarchies in practice

In “real computers”, a typical memory hierarchy is generally composed of 5 different levels:

1. Registers
2. Cache memories (managed by the CPU)
   - L1
     - L1-D
     - L1-Ins
   - L2
   - L3
3. Main memory (RAM)
4. Storage devices (ROM, Hard Drives, Removable Drives...)
   $\rightarrow$ Virtual memory (managed by the Operating System)
5. Network/Internet storage