What is a microprocessor?

Session 1

PH435

… being a review of Session 0, and a preview of things to come

Highlights of Session 0

We discussed evolution from

Digital circuits (gate based)

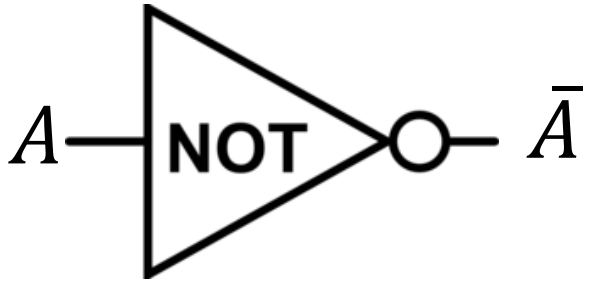
to

FPGA

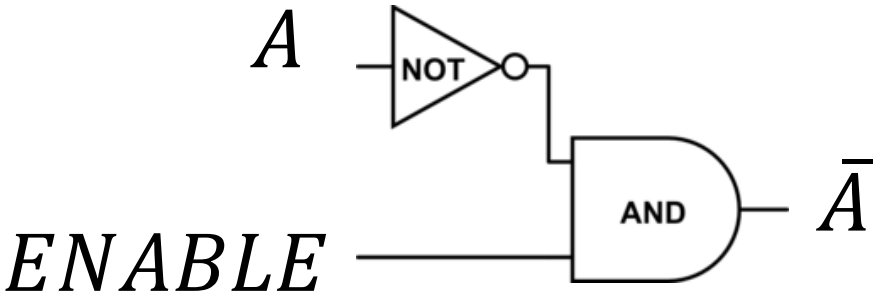
to

Microprocessors

Digital circuits



… always



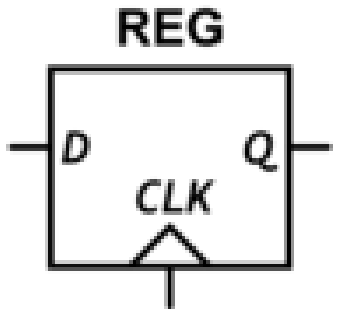
… when ENABLE=1

Primitive textbook level ‘ideal’ combinational circuits: their outputs are completely determined by truth table

They **have no memory**

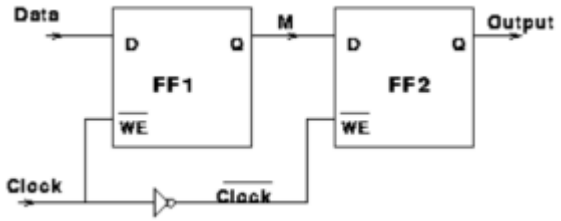
Sequential Circuits

Make a quantum leap over combinational circuits by introducing the concept of **memory**

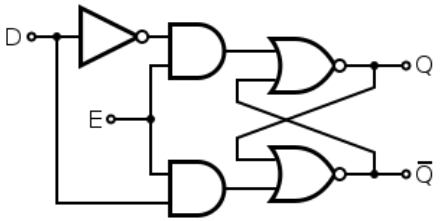


$Q = D$ at clock edge, and remains unchanged until next clock edge

Note: “edge” trigger here is done by cascading 2 D latches



Where each D latch *looks* like a combinational circuit



By introducing feedback, you introduce ‘memory’
Note: this is equally true of an analog feedback circuit!

FPGA
Field Programmable Gate Array

Practically, a large number of gates on a chip whose interconnections can be set (i.e.) programmed

Conceptually, FPGA's function as state machines, globally synchronized by a clock.

hence they implicitly require memory – this memory is amorphously distributed throughout the chip, being a bunch of (for eg.) D registers, each consisting of 2 NOT, 4 NAND and 4 NOR gates interconnected as shown in previous slide

FPGA in practice

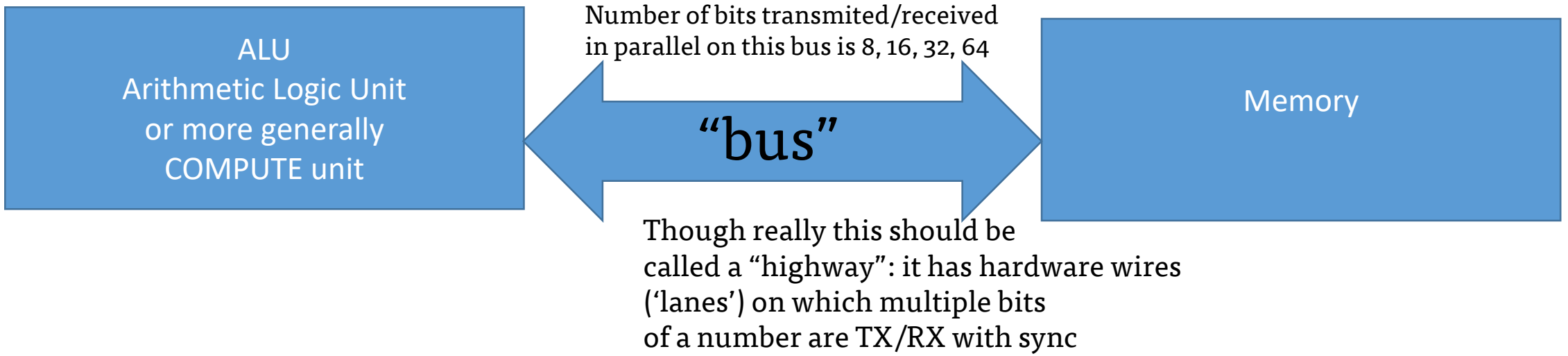
Very good at doing
input → compute → output
in a massively parallel format

Compute step may include multiple input combinations and intermediate storage / states before output is produced

Eg: Game of Life:
2-dim matrix (memory) stores state Dead/Alive of object at each site.
At each clock, new state of each site is computed by ruleset that requires access to state of near-neighbor sites (but *not* all sites)

Microprocessors

Next big jump in digital computing comes by functionally separating the ‘compute’ and ‘memory’ functionality into two separate blocks



Microprocessors

Memory is functionally and practically of different types

Functional defn

RAM

Random **A**ccess **M**emory

Every object stored here has an address – you can pull up any object at random if you know its address

Functional defn

Program Memory

ALU reads 'what to do' from here. Note: functionally program instructions (called 'opcodes') are also data!

Functional defn

Data Memory

ALU reads/writes data input and compute outputs here

Memory

Practical defn

ROM

Read **O**nly **M**emory

If you want to protect some data from being changed you put it here. Written once – then gate connections removed – stores values forever

Practical defn

EPROM

Eraseable **P**rogrammable **R**ead **O**nly **M**emory

Can be erased and re-written (i.e. gate connections can be re-arranged) - like an FPGA, but only functions as memory

Practical def

DRAM

Dynamic RAM

frequent read/write

Practical def

Flash RAM

(also NV RAM)

faster to r/w than EPROM

Microcontrollers

"bare-bones" version of microprocessor

Has essentially the same architecture as a microprocessor – usually much simplified (reduced ALU, reduced memory) with specific sets of applications in mind.

Applications usually involve **control** of physical systems – hence microprocessor → microcontroller

Microcontrollers

MAIN FUNCTIONAL DIFFERENCE

Microcontroller (generally) runs a single program.
At power on, the ALU is wired to access to program memory location `00`, decode and execute that opcode and proceed as per sequential set of opcodes till the program ends, then jump back to location `00` & loop

Microprocessors can generally run multiple programs.
Note: NOT IN PARALLEL! The programs multiplex. Some clock cycles are used to read opcodes for program 1 from program memory XX, then program 2 from program memory YY etc
An OPERATING SYSTEM (Linux, Win, MacOS) does the co-ordination between different programs)

Keywords to discuss next

Firmware v/s software

Architecture– von Neumann,
Princeton, Harvard etc

RISC (Reduced InStruCtion Set)

In-memory computing (Neuromorphic)