Architecture et Système

Stefan Schwoon

Cours L3, 2014/15, ENS Cachan

Latches and flip-flops

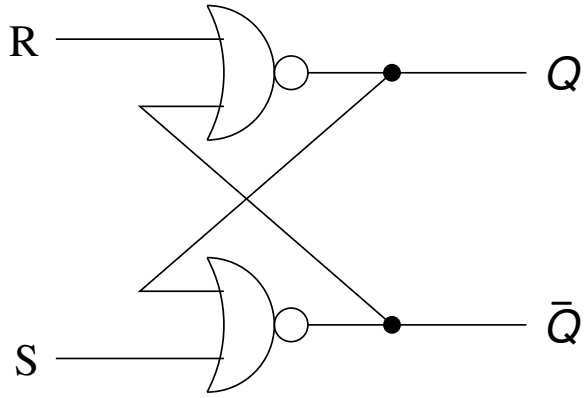
Circuits used for storing a bit of memory:

- unclocked circuits are called latches (*verrous*)
- clocked circuits are called flip-flops (*basculles*)

Typical design elements:

- controls (inputs): no effect when all controls are 0
- clock signal (only in flip-flops)
- two outputs Q and $\bar{Q}$, complements of each other

RS latch



controls R (reset) and S (set):

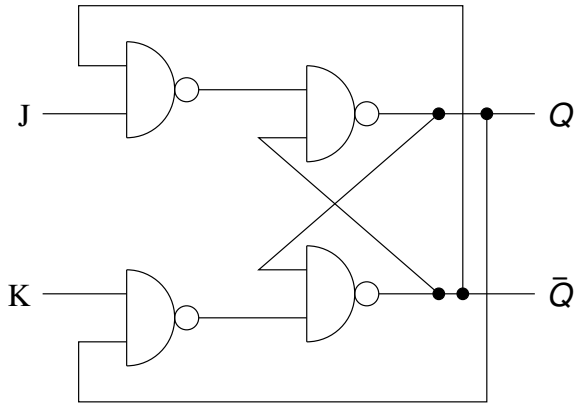
$R = S = 0$: outputs remain stable

$R = 1$, $S = 0$: output Q becomes 0

$R = 0$, $S = 1$: output Q becomes 1

$R = S = 1$: both outputs 0, undefined upon return to $R = S = 0$

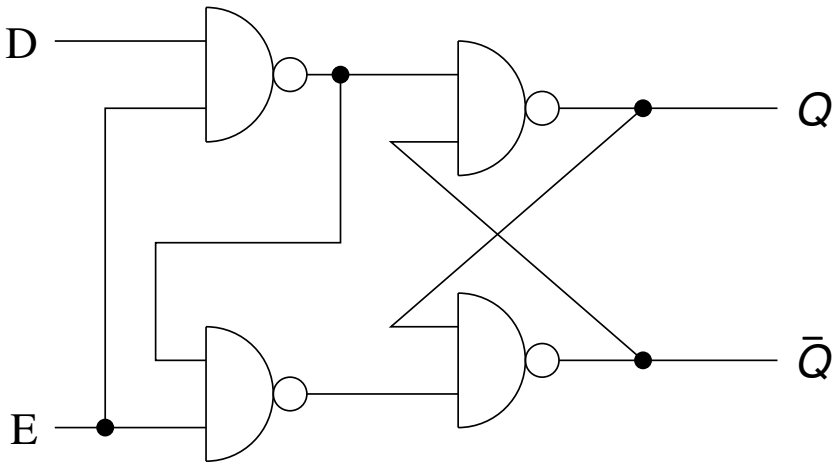
JK latch



controls J and K:

- $J = K = 0$: outputs remain stable
- $J = 0$, $K = 1$: output Q becomes 0
- $J = 1$, $K = 0$: output Q becomes 1
- $J = K = 1$: most recent rising signal taken into account

D latch

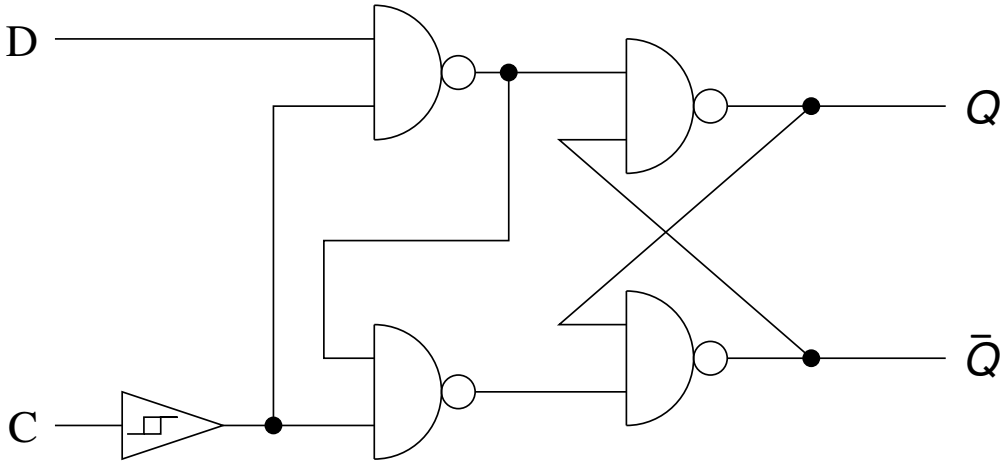


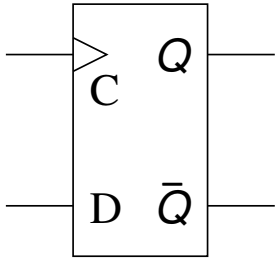
controls D (data) and E (enable):

- $E = 0$: outputs remain stable
- $E = 1$: output Q becomes D

Flip-flops

In clocked circuits, the *E* control becomes an oscillating signal *C* emitting 1 in regular (typically short) intervals. Thus, *D* is taken into account at specified moments.





In block diagrams, *C* is distinguished by a triangle.

When C is shared across all flip-flops in a system, this allows to synchronize computations.

The delays between the 1 signals need to be long enough so that all parallel computations can succeed; then, all results can be used (also as feedback) during the next clock cycle.

Also, the 1 signal needs to be short enough so that signals cannot “hop” across two circuits during one signal.

Programmable logic devices

Facilitate the use of logical circuits:

- circuitry is fixed, no need to assemble transistors and wiring manually
- certain connections can be turned on or off (= programming)

PLA (programmable logic array):

- n input bits, m output bits
- all input bits avaiable positively and in negated form
- AND-plane: $k \leq 2^n$ rows, programming can choose conjunctive clauses in each row
- OR-place: m columns, programming allows to choose clauses to “add”

ROM (read-only memory):

- special case of PLA: $k = 2^n$ and AND-plane is fixed
- programming limited to OR-plane
- realizes a memory of 2^n words of m bits

FPGA (field-programmable gate arrays):

- several lines of *logic cells*
- each logic cell can realize multiple functions on its input bits, selectable by multiplexer
- programming assigns functions to each cell and connections between them

Basic computer architecture

Let us discuss a basic design for computer architecture.

processor (CPU), consisting of control unit, arithmetic-logical unit (ALU), registers, microprogramming unit, and clock

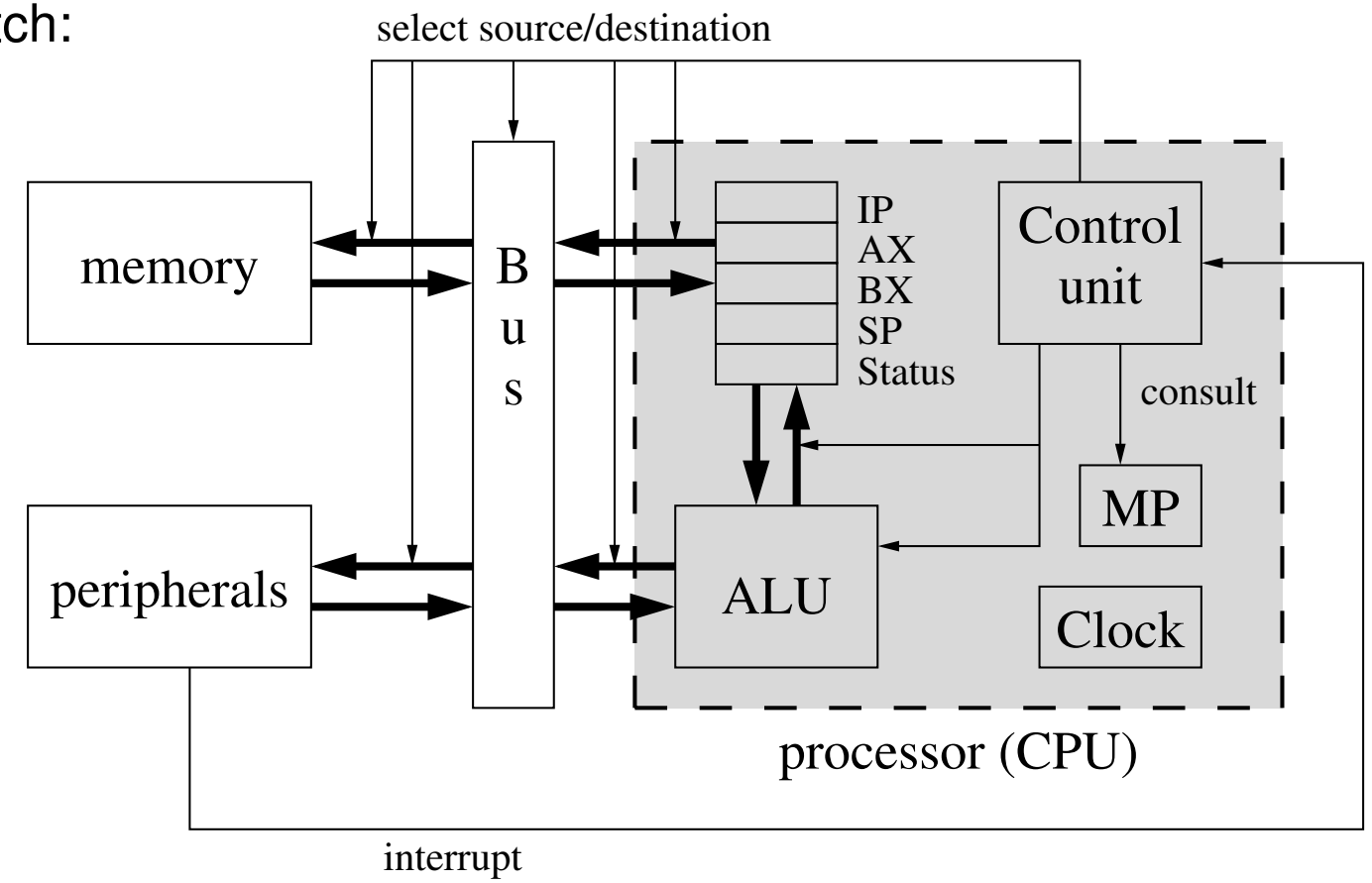
the main memory

the peripherals (external devices, input/output)

the bus, allows data transfer between the components

Basic computer architecture

A first sketch:



Memory

Stores both data and instructions ("von Neumann architecture")

Organised in 2^n words of m bits, each word stored at an *address* in $0..2^n - 1$

Typical values for m today: 32, 64; for n: 30..34 (= 1..16 GB)

Realizable, for instance, as D flip-flops (detailed discussion to follow).

Data exchange with CPU via bus:

- use multiplexer to choose word from a given address, transfer to bus
- use decoder to obtain a word from the bus, write to the correct address

The bus

Connects CPU, memory, peripherals: all components connected with input and output

Obtains a word from one source (using multiplexer), delivers it to a destination (using decoder)

One memory transfer at a time (e.g. cannot obtain two different words from memory at a time).

The CPU

Controls all operations. Its components are:

registers: short-term memory necessary for operations

ALU: used for *manipulating* data

control unit: coordinates operations, aided by clock and microprogramming unit

Registers

Registers exist for different purposes. The most common are:

general-purpose registers (accumulator, AX, eax, ...): store data used for computations

instruction pointer: points to next instruction in memory

instruction register: contains opcode of next instruction

status register: stores a number of bits indicating diverse states (result of comparison, overflow, division by zero, interrupt, ...)

stack pointer: points to a special area in memory used for remembering saved registers, return points for function calls, etc.

The control unit (CU)

Operates in phases, marked by clock signals

Each phase achieves one logical step; several phases may be needed to execute an instruction in memory.

For each phase, the CU must coordinate the transfers in the computer:

- provide the bus with the right control bits
- transfers between ALU and registers
- react to peripherals

Phases include: instruction fetch, decoding the operation, executing it

Control signals for each phase provided by microprogramming unit, e.g. a ROM

Phases

Phase 1 (Instruction fetch):

Make bus use value of IP as address selector, transfer to IR

Phase 2 (Decode):

Consult microprogramming unit what the instruction requires to do:
e.g. MU contains control bits for bus and register transfer

Phases 3 and others (Execute):

Using the control bits from the MU, organise the data transfer during the next phase(s).

Interrupts

When a peripheral needs to exchange data with the CPU, it raises an interrupt signal at the control unit.

When the CU is ready to execute the next instruction, it first checks whether at least one interrupt signal is present.

If so, it chooses the one with the highest priority.

The CPU then proceeds with an instruction that handles communication with the peripheral.