Pearl 000 – Computer Architecture

Wouter van den Brink
w.vandenbrink@student.utwente.nl

Signals

A signal can be **analog** or **digital**. Analog signals have infinitely many possible levels, while digital signals have a limited number of levels, e.g. only 0 and 1. Most real-world quantities are analog, whilst most signals in computer science are digital.

An analog signal is subject to distortion by noise and imperfections, while a small disturbance to a digital signal does no harm to its reading.

Bits and bytes

A **bit** is a signal which can assume one of two levels: 0 (low) or 1 (high). 8 bits are called a **byte**. The number of possible values for *n* bits is 2^n. For example, 4 bits can produce 2^4 (16) values: 0000_2 (0_{10}) to 1111_2 (15).

Counting in binary is mostly done like we do in the decimal system: the least significant digit goes to the right, while the most significant digit goes to the left. Given the decimal number 13, one can calculate its value by using $1 \cdot 10^1 + 3 \cdot 10^0 = 13$. The same goes for binary. Given 1101_2, you can calculate the decimal value by using $1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 13$.

This system is called **positional notation**. It's also used in the hexadecimal system, which uses 16 digits: 0 – 9, then A – F. What follows is that the possible values for a hexadecimal number with *n* digits is 16^n.

Negative numbers

We can write negative binary numbers in three ways.

Use first bit for the minus sign

Counting negative numbers is just the same as for unsigned numbers, but the most significant bit now indicates if a number is positive (0) or negative (1). In this system, -3_{10} would be 1011_2.

1's complement

In this system, we invert all bits when negative. The first bit acts as a sign, just like in the previous system. Here, -3_{10} is 1100_2.

The problems with the previous two systems is that we have two ways of writing 0. Furthermore, you'll need special add and subtract devices to handle them.

2's complement

2's complement solves these problems. Here, we just continue counting like we would in an unsigned system. $0000_2 - 1_2$ results in 1111_2 because of an "overflow". That means that you can use 2's complement in an arithmetic system that only supports unsigned numbers. Calculating the value of a 2's complement number only requires you to negate the first "power", for example:

$$1101_2 = 1 \cdot -2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = -3$$

Gates and symbols

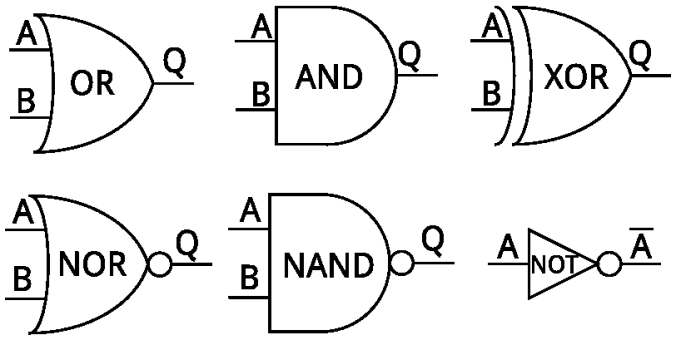
Below is a truth table of Boolean algebra.

A	B	$\bar{A}$ NOT	$A + B$ OR	$A \cdot B$ AND	$\overline{A + B}$ NOR	$\overline{A \cdot B}$ NAND	$A \oplus B$ XOR
0	0	1	0	0	1	1	0
0	1	1	1	0	0	1	1
1	0	0	1	0	0	1	1
1	1	0	1	1	0	0	0

There are a few formulas used in Boolean algebra:

Relationship	Dual form	Name
$AB = BA$	$A + B = B + A$	Commutative
$1A = A$	$0 + A = A$	Identity
$A\bar{A} = 0$	$A + \bar{A} = 1$	Complement
$A(B + C) = AB + AC$	$A + BC = (A + B)(A + C)$	Distributive
$\overline{AB} = \bar{A} + \bar{B}$	$\overline{A + B} = \bar{A} \cdot \bar{B}$	DeMorgan

Finally, the symbols for all the gates:



(Generously stolen from Google Images)

Programming

High-level programming

In this level, you have no details of the hardware you are working with. It's easier to program, but you'll end up with less efficient code.

Low-level programming

This requires more hardware knowledge. You have more control, and probably more efficient code.

Assembly

You'll need detailed hardware knowledge. This is about how powerful and efficient as it gets.

Processors

CPUs contain different elements, working together as a whole. Some important elements follow.

Combinatorial circuits
These devices perform operations on binary signals, to be used elsewhere in the CPU (or even outside of it). Operations in combinatorial circuits take some time.

Examples: ALU (arithmetic operations).

Switches
Switches can guide a signal to 1 of multiple possible sources or destinations. They can have several inputs and one output, or vice versa. They always have 1 or more control bits indicating which in- or output the switch should choose.

Registers
Registers store information presented at their inputs, when the control bit transitions from low to high. Their current value is always readable from their outputs.

Multiple registers together form a register bank. They are coordinated using a switch. So you end up with a device (the register bank) with control over the read address (which register to read from), the write address (self-explanatory by now), a control bit and data inputs and outputs.

Memory
Almost the same as a register, with the main difference being that you cannot read and write from and to a memory at the same time. A control bit indicates if you wish to read from or write to the memory. Memory is quite larger than the register bank.

Clock signal
Every t time, for example 16 000 000 times a second, a clock signal pulses through the CPU. This syncs every element and gives combinatorial circuits time to calculate. The lecture slides explain this way better than I would be able to through text.

Program
A CPU receives instructions (what to do at what moment) through a program. A program consists of multiple instructions, which indicate what element should do which action with what data. A program is stored in the program memory.

The program counter (PC) indicates at what “line” in the program we currently are. You can also modify the PC on-the-fly, making loops and such possible.