Computer Arithmetic

Written by HADIOUCHE Azouaou.

Disclaimer

This document follows the slides and lessons given by Mr.OUDJIDA, written in a mathematical textbook format, freedoms are taken when writing the course.

> To separate the contents of the course to actual additions or out of context information, a black band will be added by its side like the globing this comment.

USE IT AT YOUR OWN RISK!

Contents

Chapter 1

Classical logic

The most basic part of executing a computation on a machine is to describe the most basic information, which is true/false, and then compose them into a statement or a proposition.

Informally, given a sentence, it is said to be a *statement* if
- it is declarative, either affirmative or negative.
- it is possible truth values are true or false.
- it is verifiable in reality.

On those statements, we have some rules to give them truth values
- Law of identity: $A = A$ is a true statement.
- Law of non-contradiction: $\neg(A \wedge \neg A)$ is false statement.
- Law of excluded middle: either $\neg A$ or A is true statement.

Now we will formalize calculations on boolean variables which is known as Boolean algebra, it will help us analyze and create circuits later on, also simplifying them to have less components.

1.1. Boolean Algebra

Let $\mathbb{B} := \{0, 1\}$ denote the set of boolean values, which can be represented too with true/false. Any variable ta

Definition 1.1.1 (Boolean Variable): *Let x be a variable, x is said to be a* boolean variable *if it can assume values in $\mathbb{B}$. Let $f : \mathbb{B}^n \to \mathbb{B}$ a map, f is called a* boolean function.

Boolean functions will be the main study of Boolean algebra, how they can be written, expressed and modified without altering its values, the following operations will be useful for operating on boolean variables and construct functions.

Definition 1.1.2 (Boolean Operations): *Let x, y be two boolean variables, we define the operations $+, \cdot, \overline{}$ to be the logical or, and, not respectively, which have the following truth tables.*

y	x	$\overline{x}$	$x + y$	$x \cdot y$
0	0	1	0	0
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

there are some other operations that are as follows
- $x \mid y = \overline{x \cdot y}$.
- $x \otimes y = x \cdot \overline{y} + \overline{x} \cdot y$.
- $x \Rightarrow y = \overline{x} + y$.

Proposition 1.1.3 (Boolean Identites):

$\overline{\overline{x}} = x$	
$x + x = x$	$x \cdot x = x$
$x + 0 = x$	$x \cdot 1 = x$
$x + 1 = 1$	$x \cdot 0 = 0$
$x + y = y + x$	$x \cdot y = y \cdot x$
$x + (y + z) = (x + y) + z$	$x \cdot (y \cdot z) = (x \cdot y) \cdot z$
$x(y + z) = xz + yz$	$x + yz = (x + y) \cdot (x + z)$
$\overline{x + y} = \overline{x} \cdot \overline{y}$	$\overline{x \cdot y} = \overline{x} + \overline{y}$
$x + \overline{x} = 1$	$x \cdot \overline{x} = 0$

Definition 1.1.4 (Duality): *Let $f : \mathbb{B}^n \to \mathbb{B}$ be a boolean function, we define the dual of f as the map $(x_1, ..., x_n) \mapsto \overline{f(\overline{x}_1, ..., \overline{x}_n)}$. We can obtain the dual of a function f by swapping $+$ with $\cdot$, 0 with 1 and keep the variables unchanged.*

Definition 1.1.5 (Literal/Minterm/Maxterm): *Let $x_1, ..., x_n$ be boolean variables and $y_1, ..., y_n$ such that $\forall i \in [\![1, n]\!], y_i = x_i \vee y_i = \overline{x_i}$.*
- *A* literal *is a proposition in the form of x or $\overline{x}$ with x a boolean variable.*
- *A* minterm *of $x_1, ..., x_n$ is the product $y_1 \cdot y_2 \cdots y_n$.*
- *A* maxterm *of $x_1, ..., x_n$ is the sum $y_1 + y_2 ... + y_n$.*

Definition 1.1.6 (Conjunctive/Disjunctive Normal Form): *Let $f : \mathbb{B}^n \to \mathbb{B}$ be a boolean function, $x_1, ..., x_n$ boolean variables.*
- *DNF: $f(x_1, ..., x_n) = \sum_{i=1}^{k} X_i$ where X_i are minterms.*
- *CNF: $f(x_1, ..., x_n) = \prod_{i=1}^{k} X_i$ where X_i are maxterms.*

Proposition 1.1.7:
1. *the dual of a DNF is a CNF and vice versa.*
2. *Every boolean function can be written with only the defined connectives.*
3. *Every boolean function can be expressed only using one of those sets $\{+, —\}, \{\cdot, —\}, \{|\}$, we call them a complete set of connectives.*
4. *Any boolean function can be written in the CNF or DNF.*

Proof.

1. A minterm is of the form $y_1 ... y_n$ then its dual is $y_1 + ... + y_n$ which is a maxterm, now if f is in a DNF then it is the sum of minterms, the dual will become a product of maxterms which is a CNF, its easy to verify the rest.

2. Let $f : \mathbb{B}^n \to \mathbb{B}$ a boolean function, we define $g : \mathbb{B}^n \to \mathbb{B}$ as follows $(x_1, ..., x_n) \mapsto \sum_{i=1}^{2^n} \sigma_i(x_1, ..., x_n) \cdot f(x_1, ..., x_n)$ where $\sigma_i(x_1, ..., x_n)$ is defined as if we take the i in base 2, $i = \overline{i_n i_{n-1} ... i_1 i_0}^2$ then $\sigma_i(x_1, ..., x_n) = y_1 ... y_n$ and if $i_j = 1$ then $y_j = x_j$ otherwise $y_j = \overline{x_j}$. Notice that $\sigma_i(x_1, ..., x_n) = 1$ if and only if $\overline{x_n ... x_1}^2 = i$. So we have for $(x_1, ..., x_n) \in \mathbb{B}^n$, $g(x_1, ..., x_n) = f(x_1, ..., x_n)$ thus f can written only with the connectives.

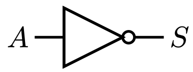
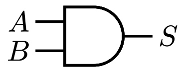
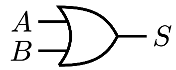
3. To prove this statement, we just need to use the fact that any function can be written using only $+, \cdot, —$.
   - We have by De Morgans laws that $x \cdot y = \overline{\overline{x} + \overline{y}}$ thus $+, —$ is enough to express every function.
   - Same can be used to express $x + y = \overline{\overline{x} \cdot \overline{y}}$.
   - Now we can use the NAND to write everything, notice that $x | x = \overline{x}$ and $(x|y)|(x|y) = x \cdot y$ thus we use the previous statement and we get that $\{|\}$ is a complete set of connectives.

4. Notice that in the first statement we proved that any boolean funtion can be written in the DNF, using the same function σ_i we can construct a DNF, it will be of the form $g : (x_1, ..., x_n) \mapsto \prod_{i=1}^{2^n} \overline{f(x_1, ..., x_n)} \cdot \overline{\sigma_i(x_1, ..., x_n)}$.

□

1.2. Logic Gates & Digital Circuits

To be able to use and/or/not in circuits, we introduce the most basic circuit components called logic gates, as the table below shows

Not	And	Or
		

Not:

A	S
0	1
1	0

And:

A	B	S
0	0	0
0	1	1
1	0	1
1	1	1

Or:

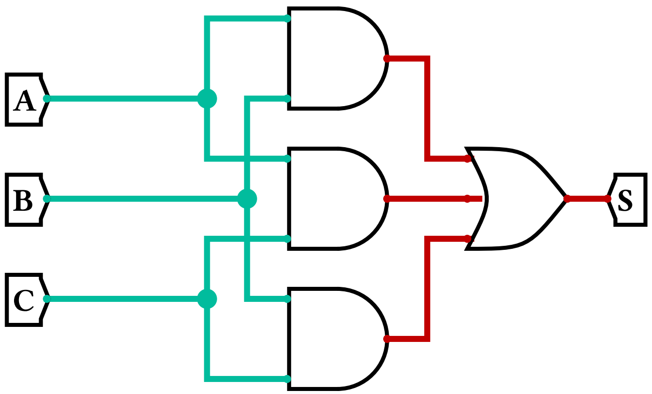
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

Thus we can use these to represent some circuits that behave as logical circuits called digital circuits.

Example:
- A commitee of n individuals decide issues for an organization. Each individual votes either yes or no for each proposal that arises. The proposal is passed if it receives at least p votes. Its easy to notice that the solution is equal to

$$y = \sum_{1 \leq i_1 < \cdots < i_p \leq n} x_{i_1} x_{i_2} \ldots x_{i_p}$$

where $x_1, \ldots, x_n$ represent the votes of the individuals and y the proposal passing. In case, $n = 3$ and $p = 2$ we get



1.3. Transistors

One of the biggest advancements in our modern world is the creation of a transistor, in principle the idea is simple, a transistor is simply an electrically controlled switch. We use this to materialize the logic gates to be able to use them directly.

1.3.1. Circuit Simplification & Reduction Methods

1. **Kernaugh Maps**
2. **Quine-McCluskey Method**

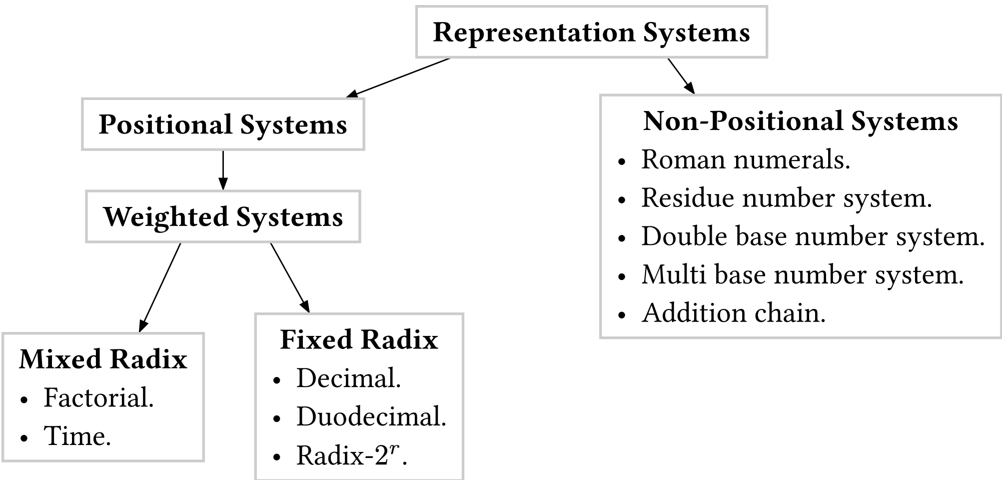
Chapter 2

Arithmetics

Computer arithmetic is the process of using algorithms for doing basic operations on numbers like addition multiplication... etc. To be able to do those operations, a representation of the numbers is needed, which will be the first part of the course.

2.1. Positive Integer Representations

Multiple numeral systems were devlopped throughout history, the roman numerals were the oldest that are still in use, but not in any complicated systems due to the difficulty of doing arithmetic operations on them.



2.1.1. Weighted Representation System

Let $I = [\![0, n-1]\!]$, and $\{D_i\}_{i \in I}$ be sets of digits, the digit vector $x = (x_{n-1}, ..., x_0) \in D_{n-1} \times \cdots \times D_0$ and let w a weight vector $w = (w_n, ..., w_0) \in \mathbb{Z}^n$, we have that the integer mapping of x in this weighted system is $w^t x = \sum_{i=0}^{n-1} x_i w_i$. The number of combinations possible is $\prod_{i=0}^{n-1} \# D_i$ but notice that there representations are not necessarily unique.

2.1.2. Mixed Radix Representation System

The mixed radix representation is the same as weighted but has a different way to get the weight vector, we consider a radix vector $r = (r_{n-1}, ..., r_0)$, $w_0 = 1$ and $w_i = w_{i-1} \cdot r_{i-1}$ and by using the representation from the weighted system we get $w^t x = \sum_{i=0}^{n-1} x_i \prod_{j=0}^{i-1} r_i$.

- The factorial system is an example of such a system, where we take the radix vector to be $r = (n, n-1, ..., 2, 1)$ then we get that the weight vector would be $w = (n!, (n-1)!, ..., 2!, 1!)$ and we take $D_i = \{0, 1, ..., i\}$, thus we get a unique representation for each integer, and using this represetation and the previous formula we get that the number of permutations is $\prod_{i=0}^{n-1} \# D_i = \sum_{i=0}^{n-1} i \cdot (i-1)! = (n+1)! - 1$.

2.1.3. Fixed Radix Representation System

The fixed radix represetation is just taking the mixed radix represetation with the vector to be all the same constant r and $D_i = D$ and thus we get that the representation is $\prod_{i=0}^{n-1} x_i r^i$.

- For $r = 10$, $D = \{0, ..., 9\}$, it is the decimal system.
- For $r = 2$, $D = \{0, 1\}$, it is the binary system.

We get a redundant representation since we can represent a number with multiple representations. Let r, s be the indices of two unbalanced positional number systems, n_r and n_s represent the number of digits required in radices r and s, then we get that $n_r / n_s = \log(s) / \log(r)$.

2.2. Integer Representations In Binary

We went through the way we represent positive integers in multiple systems. Now we will consider the binary system and the goal is to represent negative integers too, for that we will see 3 typical ways to represent them. Consider w the number of bits we will use for the representations and $x = (x_{w-1}, ..., x_0) \in \mathbb{B}^w$.

Definition 2.2.1 (Sign & Magnitude):

- *Value:* $X = (-1)^{x_{w-1}} \cdot \sum_{i=0}^{w-2} x_i 2^i$
  - x_{w-1} *is called the* sign bit
  - $(x_{w-2}, ..., x_0)$ *is the magnitude.*
- *Range:* $[\![-2^{w-1}+1; 2^{w-1}+1]\!]$.

Pros	*Cons*
• *Simple conceptually.* • *Easy to negate the values by flipping the sign bit.* • *Range is balanced evenly around 0 like the unsigned integers with one less bit.*	• *Duplicate value for zero.* • *Arithmetic operations are done with different circuits.* • *Overflow detection is more complicated because of the duplicate zero and sign bit.*

Definition 2.2.2 (One's Complement):

- *Value:* $X = -x_{w-1} \cdot (2^{w-1} - 1) + \sum_{i=0}^{w-2} x_i 2^i$.
- *Range:* $[\![-2^{w-1}+1; 2^{w-1}+1]\!]$.

Pros	*Cons*
• *Better for arithmetic with similar circuits to unsigned integers.* • *Easy to negate the values by inverting all the bits.*	• *Duplicate value for zero.* • *The range of the representation is asymmetric.*

Definition 2.2.3 (Two's Complement):

- *Value:* $X = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i 2^i$.
- *Range:* $[\![-2^{w-1}; 2^{w-1}-1]\!]$.

Pros	*Cons*
• *Single zero representation.* • *Arithmetic circuits use the exact same as unsigned integers.*	• *The range is asymmetric.* • *The negation requires extra circuitry.*

An overflow may happen in all representations, we will give the rules for detecting overflow in addition. Notice that if we add two numbers of different signs then there is no possible overflow. Thus, we will verify it only for numbers with the same sign.

Proposition (Properties Of Two's Complement):

Let $A = a_n a_{n-1} ... a_0$ *and* $B = b_n b_{n-1} ... b_0$ *two numbers represented in the two's complement representation.*

1. $A + B = c_n c_{n-1} ... c_0$ *overflows if and only if* $a_n = b_n$ *and* $c_n \neq a_n$.
2. A *can be represented in* $n + d$ *bits in two's complement with the representation* $A = \underbrace{a_n ... a_n}_{d} a_n ... a_0$.

Proof.

1. We consider the value of $A + B$

$$A + B = -a_n \cdot 2^n \sum_{i=0}^{n-1} a_i - b_n \cdot 2^n \sum_{i=0}^{n-1} b_i \cdot 2^i$$
$$= -(a_n + b_n) \cdot 2^n + \sum_{i=0}^{n-1} (a_i + b_i) 2^i$$

1. Consider the value of $A' = a_{n+d} a_{n+d-1} ... a_{n+1} a_n ... a_0, \forall i \in [\![1, d]\!], a_{n+i} = a_n$

$$A' = -a_{n+d} \cdot 2^{n+d} + \sum_{i=0}^{n+d-1} a_i \cdot 2^i$$
$$= -a_n \cdot 2^{n+d} + a_n \sum_{i=0}^{d-1} 2^{n+i} + \sum_{i=0}^{n-1} a_i \cdot 2^i = -a_n \cdot \sum_{i=0}^{d} 2^{n+i} + \sum_{i=0}^{n-1} a_i \cdot 2^i$$
$$= -a_n 2^n \cdot \frac{1 - 2^{d+1}}{1 - 2} + \sum_{i=0}^{n-1} a_i \cdot 2^i = -a_n 2^n \cdot (2^{d+1} - 1) + \sum_{i=0}^{n-1} a_i \cdot 2^i$$

by truncating the first n elements we get $-a_n 2^n \cdot (2 - 1) + \sum_{i=0}^{n-1} a_i \cdot 2^i = A$.□

2.3. Decimal Representations In Binary

After representing integers with binary, we will represent numbers with decimal digits. So we consider the following representations, we take

Definition 2.3.4 (Fixed Point Representation):
Let $x = x_{w-1}x_{w-2}...x_0x_{-1}...x_{-m} \in \mathbb{B}^{w+m}$.
- *Value:* $-x_{w-1} \cdot 2^{w-1} + \sum_{i=-m}^{w-2} x_i \cdot 2^i$
- *Range:* $[\![-2^{n-1}; 2^{n-1} - 1]\!]/2^m$.

We denote the values represented in this fixed point system as $Q_{n.m}$ *where* n *is the number of bits for integers and* m *the number of bits for the decimal part.*