Operating System And Computer Architecture

International Master Computer Networks & IoT Systems

Sami Taktak

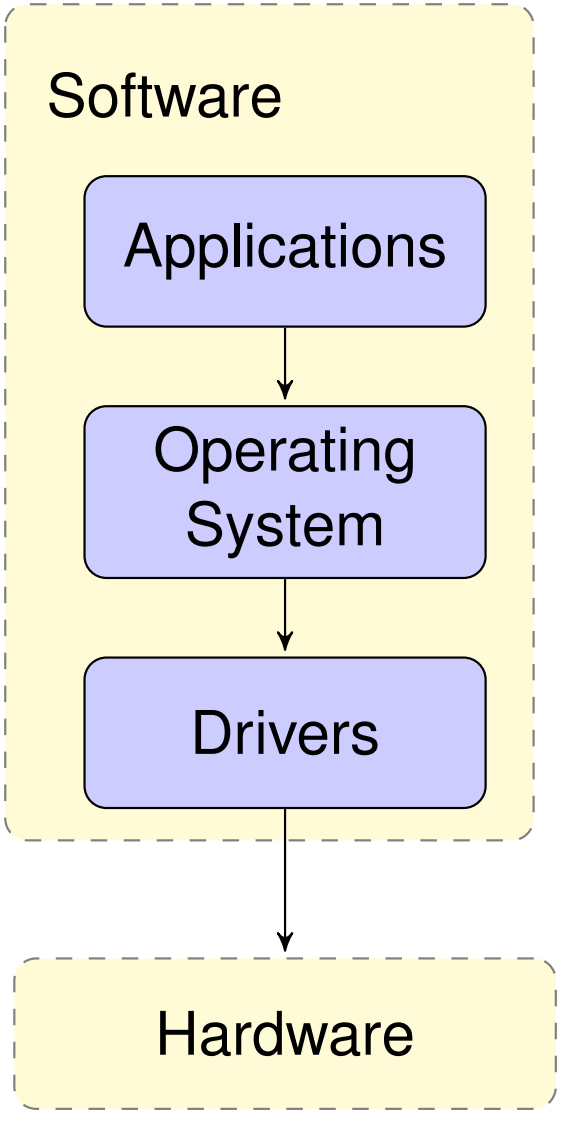
sami.taktak@cnam.fr

CEDRIC – Centre d'Étude et De Recherche en Informatique et Communications
CNAM – Conservatoire National des Arts et Métiers

October, 2023

What is a computer ?

Figure: An electronic system
[**kallernaComputer**]



Enter his world

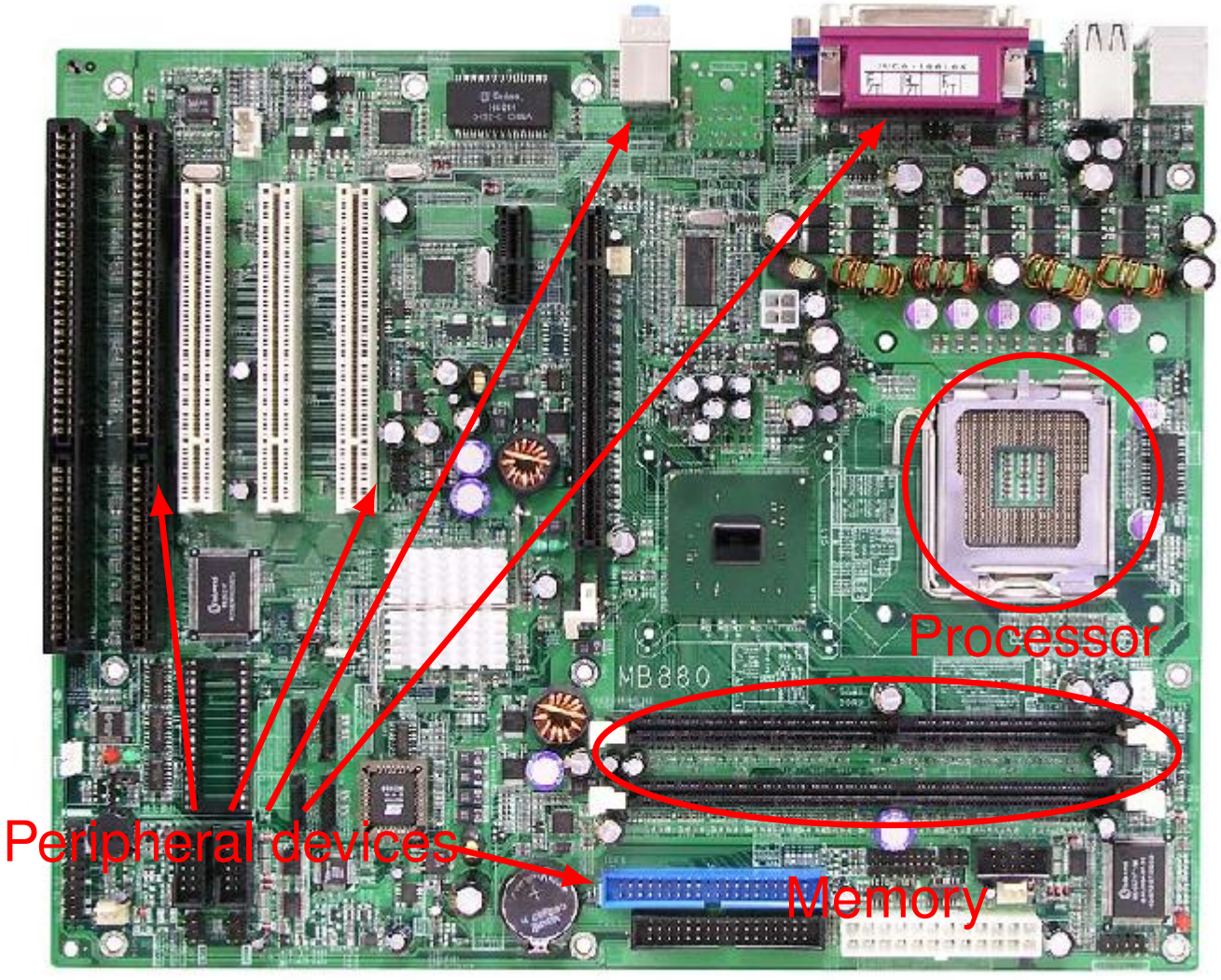
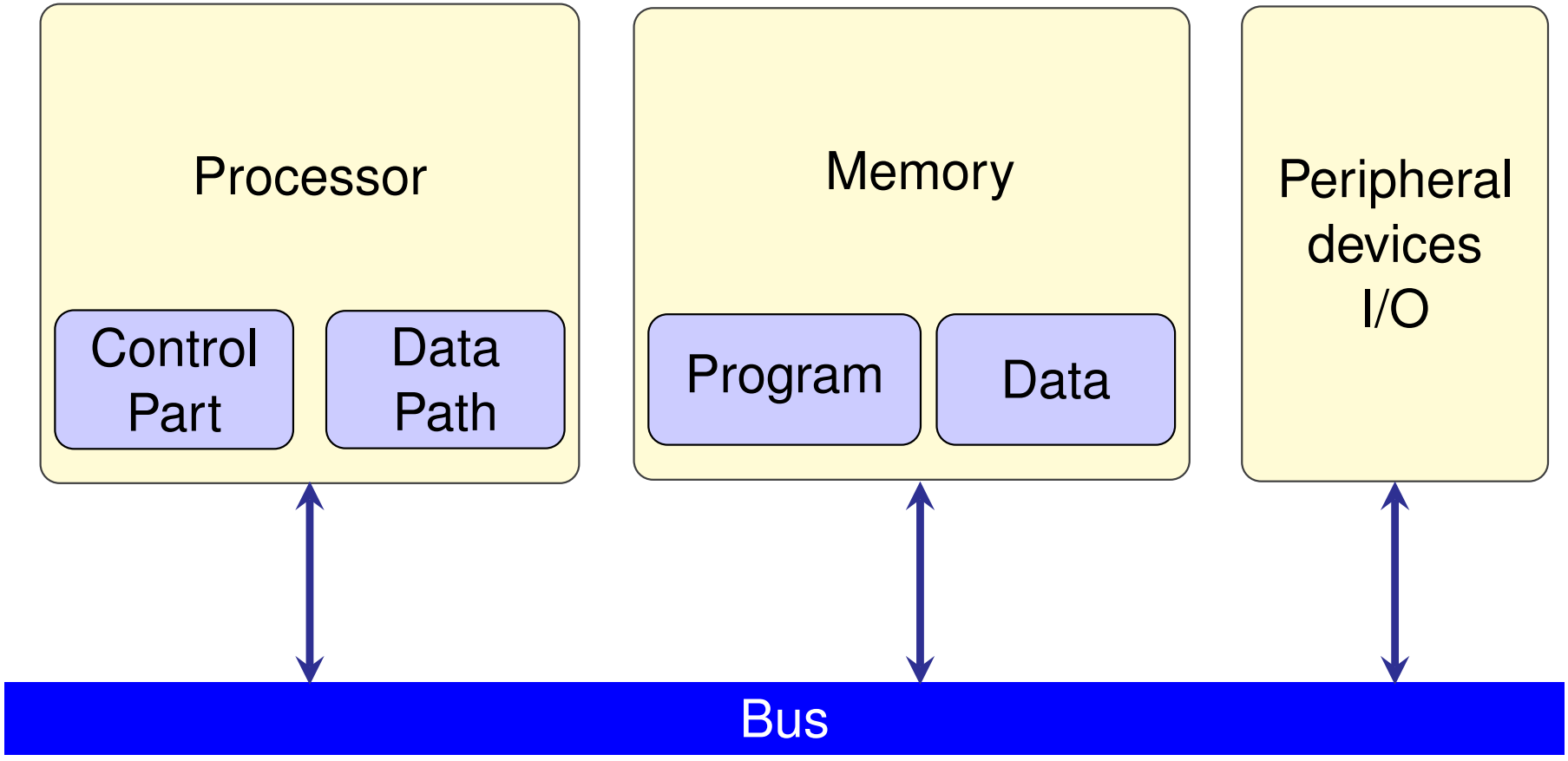


*Figure: Motherboard [**motherboard**]*

Von Neumann architecture



Inside the computer

Processor

Active part : follow the instructions

- Interpret and execute the instructions
- Read or write data from memory
- Communicate with the I/O

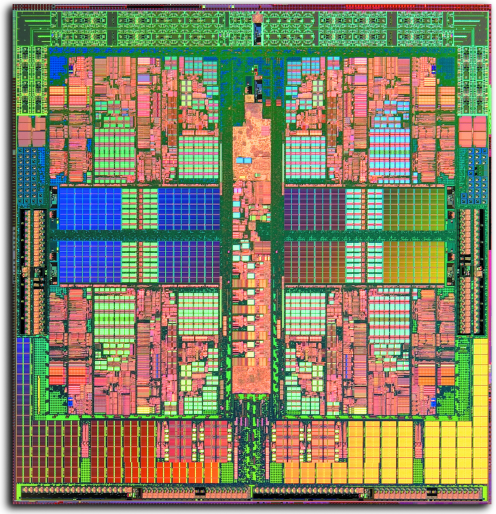


*Figure: Quad-Core AMD Opteron [**quad_core**]*

The Memory

Contains

- The running program
- Parts of the data needed

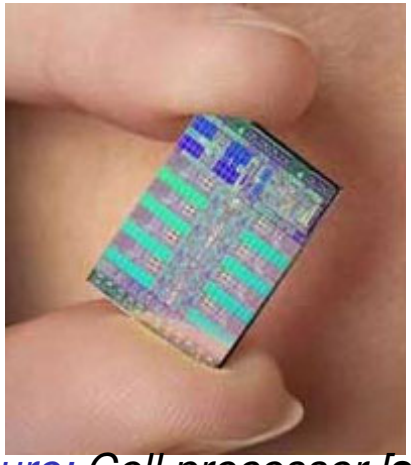


*Figure: Cell processor [**cell**]*

Inside the computer

Peripheral devices

I/O interfaces

Standard inputs

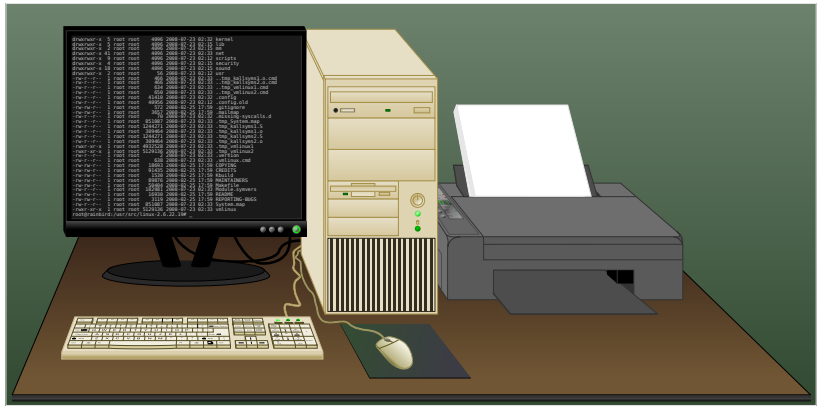
- Keyboard
- Mouse

Standard outputs

- Screen
- Printers

Standard inputs-outputs

- Network
- Disks (DVD, hard drive ...)



How to speak to an electronic device ?

Give electronic signals

- *on* (1) and *off* (0)
- The computer language is composed only by numbers in base 2.
- One letter is a bit.
- We give them instructions as a sequence of bits.

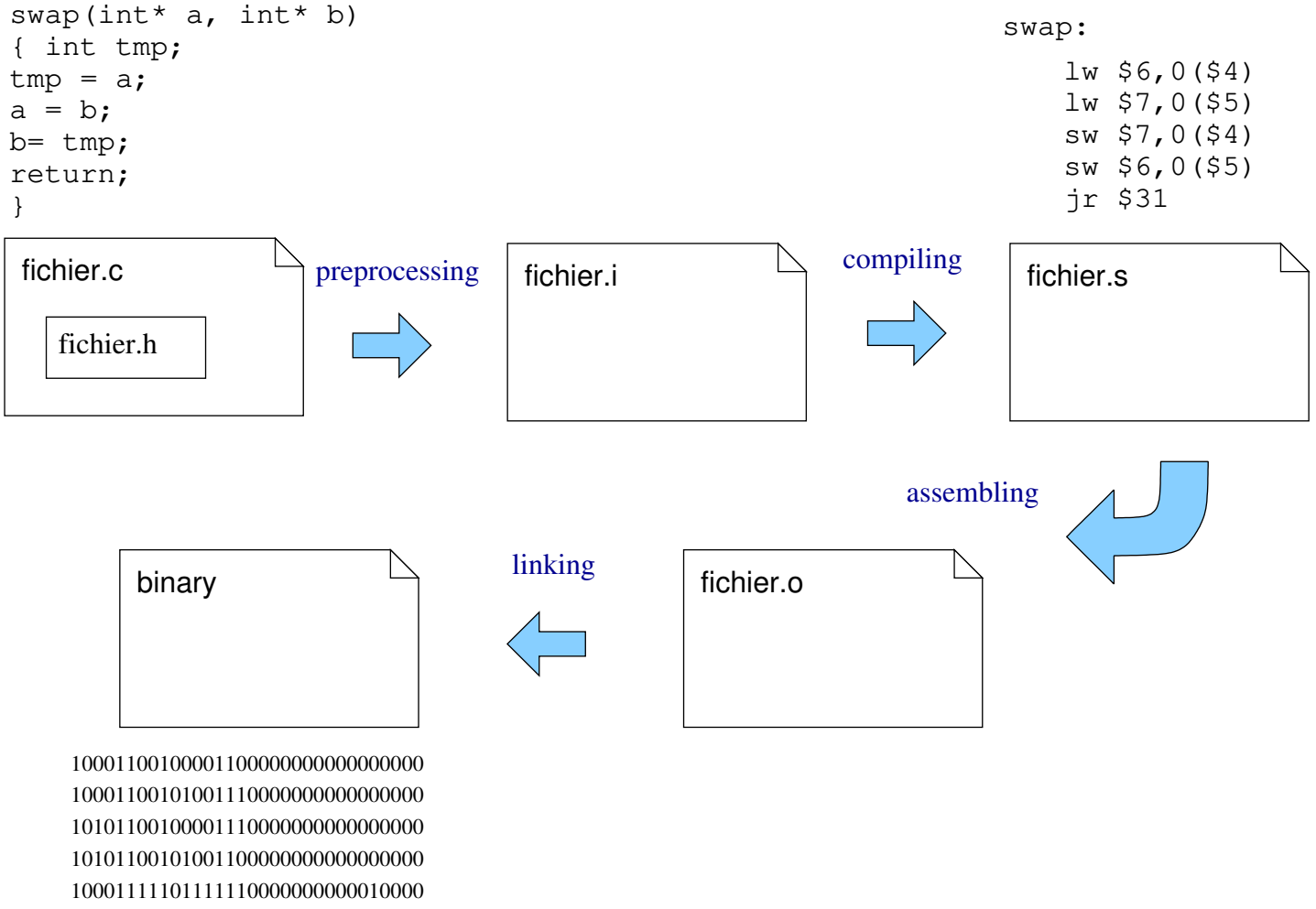
```
1 0 0 0 1 1 0 0 1 0 1 0 0 0 1 1 0 0 0 0 0
```

Command the addition of two numbers in machine language.

Assembly language

- Need more usable notation for programmers
- Using the computer to help program the computer

Compiler


swap(int* a, int* b)
{ int tmp;
tmp = a;
a = b;
b= tmp;
return;
}
swap:
lw $6,0($4)
lw $7,0($5)
sw $7,0($4)
sw $6,0($5)
jr $31
fichier.c
fichier.h
preprocessing
fichier.i
compiling
fichier.s
assembling
fichier.o
linking
binary
10001100100001100000000000000000
10001100101001110000000000000000
10101100100001110000000000000000
10101100101001100000000000000000
10001111101111110000000000010000

Human/Hardware Interface

Compiler

Translates a program from a (high-level) language to another language.

- Increase productivity.
- Independent from the system architecture and the assembly language.

Operating system (OS)

- Subroutines library for Input/Output handling
- Tasks scheduler

Data Representation

- Integer representation
  - Bounded Size Representation
  - Integer Expansion
  - Representation of Signed Numbers
  - Extension of Signed Number
- Representation of alphanumeric characters
  - ASCII Encoding
  - Words Encoding

0 and 1. . . Binary Representation

All information manipulated by computers are represented by words composed only by 0 and 1.

- *binary word* = a word using alphabets {0,1}
- *bit* = 0 or 1
- *byte* = binary word of 8 bits
- *quartet* = binary word of 4 bytes or 32 bits

Computation are done on binary representation of information

Information Encoding

- Information encoding = link between the external (natural) representation of information (character `'A'` or number `36`) and its binary representation (sequences of bits: 0010 0100)
- Context of information usage determines the used encoding or data type
- Necessity of coding rules for information processed by computers :
  - instruction : coding of instructions, depends of processor's type
  - numerical data ($\mathbb{N}$, $\mathbb{Z}$, etc.), alphanumeric characters or more complex (images) : standards encoding (two's complement, ASCII, UTF-8, RGB, ...).

How To Represent Number ?

Natural Numbers Representation

Different bases

A position system

- Base 2: $1011 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
- Base 8: $011 = 0 \times 8^2 + 1 \times 8^1 + 1 \times 8^0$
- Base 10: $2048 = 2 \times 10^3 + 0 \times 10^2 + 4 \times 10^1 + 8 \times 10^0$
- Base 16: $204C = 2 \times 16^3 + 0 \times 16^2 + 4 \times 16^1 + 12 \times 16^0$

Range

A n bits numbers can have a value between 0 and $2^n - 1$

Numeral System

- A *numeral system* (or *system of numeration*) describes how to represent numbers.
- In a system of base B with $B > 1$, numbers 0, 1, 2, ..., $B - 1$ are called digits.

Expression in base B

Any natural number N can be expressed as a sum of product of power of base B:

$$N = \sum_{i=0}^{n-1} a_i B^i = a_{n-1} \times B^{n-1} + ... + a_1 \times B^1 + a_0 \times B^0$$

with $\forall i, a_i < B$ and n the number of digits to represent N in base B.

- The compact notation of the natural number N in base B is: $a_{n-1}...a_1 a_0{}_B$

Notation For Common Base System

Notation with index

- 'b' for base 2 / binary : 111_b
- 'o' for base 8 / octal : 111_o
- 'd' for base 10 / decimal : 111_d
- 'h' for base 16 / hexadecimal : 111_h

Notation with prefix

- prefix "0b" for binary representation: 0b1110
- prefix "0o" or $\backslash$ for octal representation: 0o110 or $\backslash$110
- prefix "0x" for hexadecimal representation: 0x111, 0x23445, ...

Integer Representation

1 bit	0,1	$2^1 = 2$ states
2 bit	00,01,10,11	$2^2 = 4$ states
3 bit	000,001,010,011,100,101,110,111	$2^3 = 8$ states
n bit		2^n states

1 byte = 8 bits	Decimal	Binary
1 byte	1000^0B = 1 B	1024^0B = 1 B
1 kilobyte	1000^1B = 1 KB	1024^1B = 1 KiB
1 magebyte	1000^2B = 1 MB	1024^2B = 1 MiB
1 gigabyte	1000^3B = 1 GB	1024^3B = 1 GiB
1 terabyte	1000^4B = 1 TB	1024^4B = 1 TiB
1 petabyte	1000^5B = 1 PB	1024^5B = 1 PiB

Integer Representation

N expressed in base X:

$$N_X = a_n \ldots a_1 a_0$$

Most Significant Bit (MSB) Least Significant Bit (LSB)

N expressed in decimal:

$$N = a_n \times X^n + \ldots + a_1 \times X^1 + a_0 \times X^0$$

Base	Symbols	Example
X = 2	0,1	$110 = (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0)$
X = 10	0,1,2,3,4,5,6,7,8,9	$110 = (1 \times 10^2) + (1 \times 10^1) + (0 \times 10^0)$
X = 16	0,1,2,3,4,5,6,7,8,9, A,B,C,D,E,F	$110 = (1 \times 16^2) + (1 \times 16^1) + (0 \times 16^0)$

Common Bases

Base	Digits	Digits value in base 2
10	0,1,2,3,4,5,6,7,8,9	0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001
16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F	0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111
8	0,1,2,3,4,5,6,7	0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111

Examples:

$(AF2)_{16} = (1010\ 1111\ 0010)_2$

$0xAF2 = 0b1010\ 1111\ 0010$

$(72)_8 = (111\ 010)_2$

$0o072 = 0b111\ 010$

$\backslash 072 = 0b111\ 010$

Representation

Exemples:

- $1000_2 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 8_{10}$
- $1100_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 12_{10}$
- $12_{16} = 1 \times 16^1 + 2 \times 16^0 = 16 + 2 = 18_{10}$

Representation in base 2

- Information processed by a computer being binary words: natural number are represented in base 2 and digits are 0 and 1

Interpretation of natural numbers in binary system:

$$(a_{n-1}...a_1 a_0)_b = N_d = (\sum_{i=0}^{n-1} a_i 2^i)_d$$

Exemples

$N_d = 110011_b = (1 * 2^5 + 1 * 2^4 + 0 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0)_d$
$N_d = (2^5 + 2^4 + 2^1 + 2^0)_d = 51_d$

- Integer represented in binary $\Rightarrow$ huge number of bits
- Representation in base 16 is often preferred since it is more compact and conversion between hexadecimal and binary is easy

Hexadecimal Representation

- In base 16, symbols 0, 1, ..., 8, 9, A, B, C, D, E,F are used for the 16 digits:

Hexadecimal	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111

Hexadecimal	Decimal	Binary
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Interpretation of number in base 16

Interpretation of number in hexadecimal

$N_d = (a_{n-1}a_{n-2}a_1a_0)_h$

$N_d = (a_{n-1} \times 16^{n-1} + a_{n-2} \times 16^{n-2} + ... + a_1 \times 16^1 + a_0 \times 16^0)_d$

- four time less digits than in binary notation
- Attention: $1001_h \neq 1001_b$!

Computer Representation : bounded size !

Bounded size representation

Number are represented in fixed size words (8, 16, 32 or 64 bits)
$\Rightarrow$ all number cannot be represented

Representation interval

On p symbols in base B only natural numbers within interval $[0, B^p - 1]$ can be represented.

Example

- on 3 digits in decimal,
- on 3 bits in binary,
- on 3 digits in hexadecimal,

Computer Representation : bounded size !

Bounded size representation

Number are represented in fixed size words (8, 16, 32 or 64 bits)
$\Rightarrow$ all number cannot be represented

Representation interval

On p symbols in base B only natural numbers within interval $[0, B^p - 1]$ can be represented.

Example

- on 3 digits in decimal, interval [0, 999]
- on 3 bits in binary, interval [0,7]
- on 3 digits in hexadecimal, interval [0, 4095]

Expansion of a Natural Number Representation

Expansion for p to n symbols

For any base B, a natural number represented on p symbols can be extended on $n > p$ symbols by introducing 0 on symbols of ranks p to $n - 1$.

Expansion from 4 to 8 bits

- $0011_b \rightarrow$
- $1001_b \rightarrow$

Expansion of a Natural Number Representation

Expansion for p to n symbols

For any base B, a natural number represented on p symbols can be extended on $n > p$ symbols by introducing 0 on symbols of ranks p to $n - 1$.

Expansion from 4 to 8 bits

- $0011_b \rightarrow 00000011_b$
- $1001_b \rightarrow$

Expansion of a Natural Number Representation

Expansion for p to n symbols

For any base B, a natural number represented on p symbols can be extended on $n > p$ symbols by introducing 0 on symbols of ranks p to $n - 1$.

Expansion from 4 to 8 bits

- $0011_b \rightarrow 00000011_b$
- $1001_b \rightarrow 00001001_b$

Representing Negative Value

One's complement

Use the MSB as signe indicator :

- 124 representation:
  $64 + 32 + 16 + 8 + 4 = 2^6 + 2^5 + 2^4 + 2^3 + 2^2 = 01111100_2$
  $\Rightarrow +124 = 01111100_2$
- -124 representation:
  $124 = 01111100_2$
  $\Rightarrow -124 = 11111100_2$

Allow to represent on 8 bits, numbers

- from -127 (1111 1111) to 0 (1000 0000)
- and from 0 (0000 0000) to 127 (0111 1111)

$\Rightarrow$ Two representation for 0 !!
Additions and subtractions are complex

Representing Negative Value

Two's complement

Two's complement representation

1. Take the absolute value
2. Complement all the bits
3. Add 1

Positive number are represented as before.

n_{10}	n_2	$\bar{n}$	$-n_2 = \bar{n} + 1$	$-n_{10}$
1	0001	1110	1111	-1
5	0101	1010	1011	-5
6	0110	1001	1010	-6
0	0000	1111	0000	0

- Range value : $[-2^{n-1}; 2^{n-1} - 1]$
- The most significant bit represents the *sign bit*.

Two's complement representation

Use the MSB as signe indicator :

- 124 representation:
  $64+32+16+8+4 = 2^6+2^5+2^4+2^3+2^2 = 0111\ 1100_2$
  $\Rightarrow +124 = 0111\ 1100_2$
- -124 representation:
  $124 = 0111\ 1100_2$
  $-124 = \overline{124}+1 = 1000\ 0011_2+1 = 1000\ 0100_2$
  $\Rightarrow -124 = 1000\ 0100_2$

Allow to reprensent on 8 bits numbers

- from -1(1111 1111) to -128(1000 0000)
- and from 0 (0000 0000) to 127 (0111 1111)

Two's complement representation

Two's complement interpretation

Number are represented by two parts:

- a constant negative or null term
- a positive correction

Two's complement representation of $a_{n-1}...a_1 a_{0\,b}$

$(a_{n-1}...a_1 a_0)_b = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$

$(a_{n-1}...a_1 a_0)_b = -a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + ... + a_1 2^1 + a_0 2^0$

- $-a_{n-1}2^{n-1}$ negative or nul constant
- $\sum_{i=0}^{n-2} a_i 2^i$ correspond to the positive correction

Two's complement representation

Two's complement interpretation

Number are represented by two parts:

- a constant negative or null term
- a positive correction

Two's complement representation of $a_{n-1}...a_1 a_0{}_b$

$(a_{n-1}...a_1 a_0)_b = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$

$(a_{n-1}...a_1 a_0)_b = -a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + ... + a_1 2^1 + a_0 2^0$

- $-a_{n-1}2^{n-1}$ negative or nul constant
- $\sum_{i=0}^{n-2} a_i 2^i$ correspond to the positive correction

Two's complement representation

$$(a_{n-1}...a_1a_0)_b = (-a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + ... + a_1 2^1 + a_0 2^0)_d$$

- $-a_{n-1}2^{n-1}$ negative or nul constant
  $\Rightarrow a_{n-1}$ sign bit: give the signe value
- $\sum_{i=0}^{n-2} a_i 2^i$ correspond to the positive correction

ATTENTION

Do not miss interpret two's complement representation as the unsigned base 2 representation $\sum_{i=0}^{n-1} a_i 2^i$

Two's complement representation

$$(a_{n-1}...a_1a_0)_b = (-a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + ... + a_1 2^1 + a_0 2^0)_d$$

- $-a_{n-1}2^{n-1}$ negative or nul constant
  $\Rightarrow a_{n-1}$ sign bit: give the signe value
- $\sum_{i=0}^{n-2} a_i 2^i$ correspond to the positive correction

ATTENTION

Do not miss interpret two's complement representation as the unsigned base 2 representation $\sum_{i=0}^{n-1} a_i 2^i$

Two's complement representation

Examples

Example on 3 bits

Let $a_2 a_1 a_0$ be a 3 bit word, it's two's complement interpretation :

$$-a_2 \times 2^2 + a_1 \times 2 + a_0$$

- $N = (a_2 a_1 a_0)_b = -a_2 \times 2^2 + a_1 \times 2 + a_0$
- $N = 000_b = 0_d$
- $N = 101_b = -2^2 + 1 = -3$
- $N = 110_b = -2^2 + 2 = -2$
- $N = 111_b = -2^2 + 2 + 1 = -1$
- $N = 011_b = 2 + 1 = 3$
- Look at bit sign: for positive number, sign bit a_2 is null; for negative number, bit sign a_2 is 1

Two's complement representation

Examples

Example on 3 bits

Let $a_2 a_1 a_0$ be a 3 bit word, it's two's complement interpretation :

$$-a_2 \times 2^2 + a_1 \times 2 + a_0$$

- $N = (a_2 a_1 a_0)_b = -a_2 \times 2^2 + a_1 \times 2 + a_0$
- $N = 000_b = 0_d$
- $N = 101_b = -2^2 + 1 = -3$
- $N = 110_b = -2^2 + 2 = -2$
- $N = 111_b = -2^2 + 2 + 1 = -1$
- $N = 011_b = 2 + 1 = 3$
- Look at bit sign: for positive number, sign bit a_2 is null; for negative number, bit sign a_2 is 1

Two's complement representation

Examples

Example on 3 bits

Let $a_2 a_1 a_0$ be a 3 bit word, it's two's complement interpretation :

$$-a_2 \times 2^2 + a_1 \times 2 + a_0$$

- $N = (a_2 a_1 a_0)_b = -a_2 \times 2^2 + a_1 \times 2 + a_0$
- $N = 000_b = 0_d$
- $N = 101_b = -2^2 + 1 = -3$
- $N = 110_b = -2^2 + 2 = -2$
- $N = 111_b = -2^2 + 2 + 1 = -1$
- $N = 011_b = 2 + 1 = 3$
- Look at bit sign: for positive number, sign bit a_2 is null; for negative number, bit sign a_2 is 1

Two's complement representation

Examples

Example on 3 bits

Let $a_2 a_1 a_0$ be a 3 bit word, it's two's complement interpretation :

$$-a_2 \times 2^2 + a_1 \times 2 + a_0$$

- $N = (a_2 a_1 a_0)_b = -a_2 \times 2^2 + a_1 \times 2 + a_0$
- $N = 000_b = 0_d$
- $N = 101_b = -2^2 + 1 = -3$
- $N = 110_b = -2^2 + 2 = -2$
- $N = 111_b = -2^2 + 2 + 1 = -1$
- $N = 011_b = 2 + 1 = 3$
- Look at bit sign: for positive number, sign bit a_2 is null; for negative number, bit sign a_2 is 1

Two's complement representation

Examples

Example on 3 bits

Let $a_2 a_1 a_0$ be a 3 bit word, it's two's complement interpretation :

$$-a_2 \times 2^2 + a_1 \times 2 + a_0$$

- $N = (a_2 a_1 a_0)_b = -a_2 \times 2^2 + a_1 \times 2 + a_0$
- $N = 000_b = 0_d$
- $N = 101_b = -2^2 + 1 = -3$
- $N = 110_b = -2^2 + 2 = -2$
- $N = 111_b = -2^2 + 2 + 1 = -1$
- $N = 011_b = 2 + 1 = 3$
- Look at bit sign: for positive number, sign bit a_2 is null; for negative number, bit sign a_2 is 1

Two's complement representation

Examples

Example on 3 bits

Let $a_2 a_1 a_0$ be a 3 bit word, it's two's complement interpretation :

$$-a_2 \times 2^2 + a_1 \times 2 + a_0$$

- $N = (a_2 a_1 a_0)_b = -a_2 \times 2^2 + a_1 \times 2 + a_0$
- $N = 000_b = 0_d$
- $N = 101_b = -2^2 + 1 = -3$
- $N = 110_b = -2^2 + 2 = -2$
- $N = 111_b = -2^2 + 2 + 1 = -1$
- $N = 011_b = 2 + 1 = 3$
- Look at bit sign: for positive number, sign bit a_2 is null; for negative number, bit sign a_2 is 1

Two's complement representation

Examples

Example on 3 bits

Let $a_2 a_1 a_0$ be a 3 bit word, it's two's complement interpretation :

$$-a_2 \times 2^2 + a_1 \times 2 + a_0$$

- $N = (a_2 a_1 a_0)_b = -a_2 \times 2^2 + a_1 \times 2 + a_0$
- $N = 000_b = 0_d$
- $N = 101_b = -2^2 + 1 = -3$
- $N = 110_b = -2^2 + 2 = -2$
- $N = 111_b = -2^2 + 2 + 1 = -1$
- $N = 011_b = 2 + 1 = 3$
- Look at bit sign: for positive number, sign bit a_2 is null; for negative number, bit sign a_2 is 1

Extension of Signed Number

Extension of N with p bits : principle

Integer extension on n bits to $n > p$ bits :

- bits of rank 0 to $p-1$ remain unchanged
- bits of rank p to $n-1$ take the same value as the sign bit (rank $p-1$)

Extension : principle

If $N = a_{p-1}a_{p-2}...a_1a_0$ then on n bits
$N = a_{p-1}...(n-p\ times)...a_{p-1}a_{p-1}a_{p-2}...a_1a_0$

Extension of Signed Numbers

Signed Extension Examples

Extension from 4 to 8 bits

- $N_1 = 1001_b = 11111001_b$
- $N_2 = 0110_b = 00000110_b$

Extension for 16 to 32 bits in hexadecimal

- $N_3 = 90B2_h = FFFF90B2_h$
- $N_4 = 1110_h = 00001110_h$

Extension of Signed Numbers

Signed Extension Examples

Extension from 4 to 8 bits

- $N_1 = 1001_b = 11111001_b$
- $N_2 = 0110_b = 00000110_b$

Extension for 16 to 32 bits in hexadecimal

- $N_3 = 90B2_h = FFFF90B2_h$
- $N_4 = 1110_h = 00001110_h$

Extension of Signed Numbers

Signed Extension Examples

Extension from 4 to 8 bits

- $N_1 = 1001_b = 11111001_b$
- $N_2 = 0110_b = 00000110_b$

Extension for 16 to 32 bits in hexadecimal

- $N_3 = 90B2_h = FFFF90B2_h$
- $N_4 = 1110_h = 00001110_h$

Extension of Signed Numbers

Signed Extension Examples

Extension from 4 to 8 bits

- $N_1 = 1001_b = 11111001_b$
- $N_2 = 0110_b = 00000110_b$

Extension for 16 to 32 bits in hexadecimal

- $N_3 = 90B2_h = FFFF90B2_h$
- $N_4 = 1110_h = 00001110_h$

Extension of Signed Numbers

Signed Extension Examples

Extension from 4 to 8 bits

- $N_1 = 1001_b = 11111001_b$
- $N_2 = 0110_b = 00000110_b$

Extension for 16 to 32 bits in hexadecimal

- $N_3 = 90B2_h = FFFF90B2_h$
- $N_4 = 1110_h = 00001110_h$

Extension of Signed Numbers

Signed Extension Examples

Extension from 4 to 8 bits

- $N_1 = 1001_b = 11111001_b$
- $N_2 = 0110_b = 00000110_b$

Extension for 16 to 32 bits in hexadecimal

- $N_3 = 90B2_h = FFFF90B2_h$
- $N_4 = 1110_h = 00001110_h$

Extension of Signed Numbers

Signed Extension Examples

Extension from 4 to 8 bits

- $N_1 = 1001_b = 11111001_b$
- $N_2 = 0110_b = 00000110_b$

Extension for 16 to 32 bits in hexadecimal

- $N_3 = 90B2_h = FFFF90B2_h$
- $N_4 = 1110_h = 00001110_h$

Extension of Signed Numbers

Signed Extension Examples

Extension from 4 to 8 bits

- $N_1 = 1001_b = 11111001_b$
- $N_2 = 0110_b = 00000110_b$

Extension for 16 to 32 bits in hexadecimal

- $N_3 = 90B2_h = FFFF90B2_h$
- $N_4 = 1110_h = 00001110_h$

Representation of alphanumeric characters

Alphanumeric characters are used for natural language words (text, programs, commands, emails, . . .)

Characters encoding

- Characters on a keyboard are designed by their position (raw and column numbers)
- A coding table map each position (possibly with a modifier) to a character
- Many different encoding exist: ASCII, latin1, UTF8, . . .

ASCII Encoding

In that encoding, a character is encoded on 1 byte, a corresponding table give the encoding.
Only 7 of the 8 bits are used; the MSB is always 0.

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Extension of ASCII Encoding

Extension iso-latin1

- Extensions of ASCII encoding use the 8th to support more characters. Characters from 00_h to $7F_h$ remain the same for compatibility purpose. Characters add specifics of language (encoded from 80_h to FF_h)
- One extension for french characters (“accents”, “cédille”, . . .) is latin-1 (ISO 8859-1).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8																
9																
A		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Extension iso-latin

Words Coding

Word Coding

- A word is composed of letters
- An encoding of a word or a *character string* = encoding of each letter composing the word in sequence

ASCII encoding of a word

- "Lundi" = 0x4C ('L') 0x75 ('u') 0x6E ('n') 0x64 ('d') 0x69 ('i') et 0x00 ('\0' or end of string characters)
- "123" = 0x31 ('1') 0x32 ('2') 0x33 ('3') 0x00 ('\0' or end of string characters)

Attention

$123_d = 64 + 32 + 16 + 8 + 2 + 1$
$123_d = 01111011_b = 0x7B_h = 0x0000007B_h$
$123_d \neq$ "123"

Words Coding

Word Coding

- A word is composed of letters
- An encoding of a word or a *character string* = encoding of each letter composing the word in sequence

ASCII encoding of a word

- "Lundi" = 0x4C (’L’) 0x75 (’u’) 0x6E (’n’) 0x64 (’d’) 0x69 (’i’) et 0x00 (’\0’ or end of string characters)
- "123" = 0x31 (’1’) 0x32 (’2’) 0x33 (’3’) 0x00 (’\0’ or end of string characters)

Attention

$123_d = 64 + 32 + 16 + 8 + 2 + 1$
$123_d = 01111011_b = 0x7B_h = 0x0000007B_h$
$123_d \neq$ "123"

Words Coding

Word Coding

- A word is composed of letters
- An encoding of a word or a *character string* = encoding of each letter composing the word in sequence

ASCII encoding of a word

- "Lundi" = 0x4C ('L') 0x75 ('u') 0x6E ('n') 0x64 ('d') 0x69 ('i') et 0x00 ('\0' or end of string characters)
- "123" = 0x31 ('1') 0x32 ('2') 0x33 ('3') 0x00 ('\0' or end of string characters)

Attention

$123_d = 64 + 32 + 16 + 8 + 2 + 1$

$123_d = 01111011_b = 0x7B_h = 0x0000007B_h$

$123_d \neq$ "123"

Words Coding

Word Coding

- A word is composed of letters
- An encoding of a word or a *character string* = encoding of each letter composing the word in sequence

ASCII encoding of a word

- "Lundi" = 0x4C (’L’) 0x75 (’u’) 0x6E (’n’) 0x64 (’d’) 0x69 (’i’) et 0x00 (’\0’ or end of string characters)
- "123" = 0x31 (’1’) 0x32 (’2’) 0x33 (’3’) 0x00 (’\0’ or end of string characters)

Attention

$123_d = 64 + 32 + 16 + 8 + 2 + 1$

$123_d = 01111011_b = 0x7B_h = 0x0000007B_h$

$123_d \neq$ "123"

Words Coding

Word Coding

- A word is composed of letters
- An encoding of a word or a *character string* = encoding of each letter composing the word in sequence

ASCII encoding of a word

- "Lundi" = 0x4C ('L') 0x75 ('u') 0x6E ('n') 0x64 ('d') 0x69 ('i') et 0x00 ('\0' or end of string characters)
- "123" = 0x31 ('1') 0x32 ('2') 0x33 ('3') 0x00 ('\0' or end of string characters)

Attention

$123_d = 64 + 32 + 16 + 8 + 2 + 1$
$123_d = 01111011_b = 0x7B_h = 0x0000007B_h$
$123_d \neq$ "123"

Base Conversion

How to convert from base B_1 to base B_2 ?

- Conversion Algorithm
  - By using power table
  - By using successive divisions
- Simple correspondence between some bases:
  - 2 to 16
  - 16 to 2

  $\Rightarrow$ direct mapping

Conversion Algorithm by Successive Divisions

Conversion from base 10 to base $B > 1$ for a give numbers N:

- by successive division by B (Euclidean division)
- the remainder is the digit in B

Algorithm:

$i \leftarrow 0$
$Q \leftarrow 1$
while $Q > 0$ **do**
    $(Q, R) \leftarrow \frac{N}{B}$
    $a_i \leftarrow R$
    $N \leftarrow Q$
    $i \leftarrow i + 1$
end while
$a_i \leftarrow 0$
Return $\{a_{j,j\in[0,i]}\}$

Q: Quotient
R: Remainder

Euclidean division:

$$N = Q * B + R$$

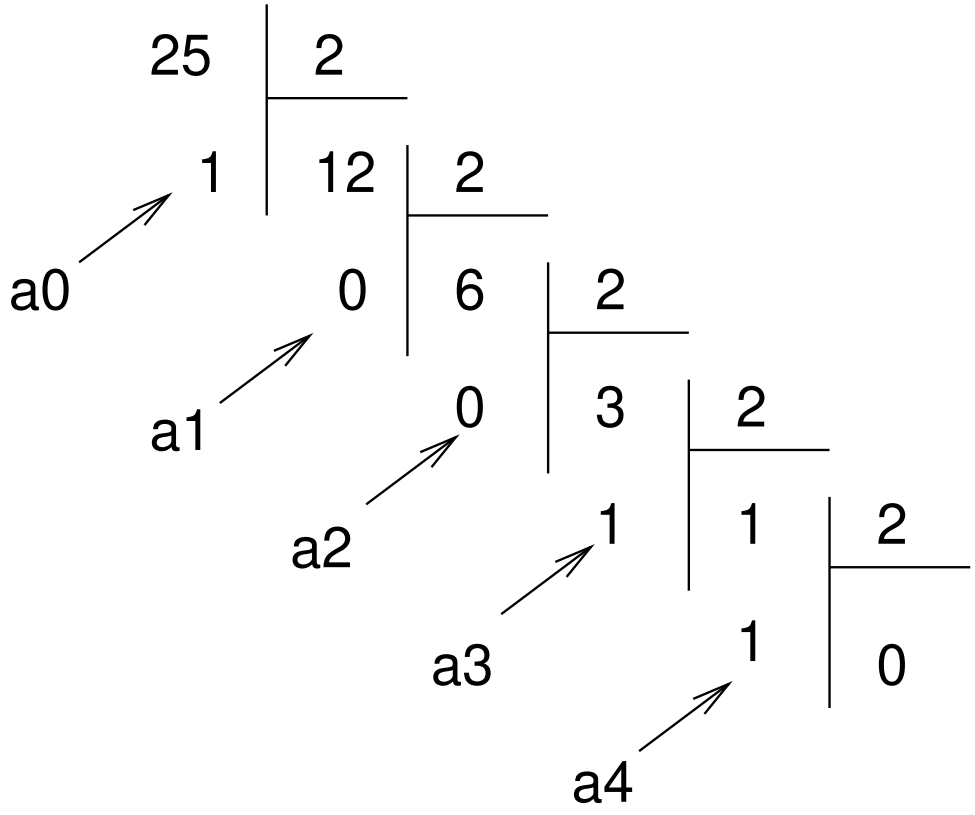
Result:

$$N_d = (a_i \dots a_0)_B$$

Conversion by Successive Division

Example

Convert 25d in binary



25d = 11001b = 2^4 + 2^3 + 2^0

Conversion from base 10 to base B

Conversion of a given number N to a base $B > 1$ on $(k+1)$ digits by searching for power multiples of B.

Algorithme

$i \leftarrow k$

$b_{i,i\in[0,k]} \leftarrow 0$

while $N \geq 0$ and $i \geq 0$ **do**

  find $\alpha \in [0, B-1]$ **such that**

    $N \geq \alpha.B^i$ and $N < (\alpha+1).B^i$

  $b_i \leftarrow \alpha$

  $N \leftarrow N - \alpha.B^i$

  $i \leftarrow i - 1$

end while

Return $\{b_{i,i\in[0,k]}\}$

Conversion from base 10 to base B

Exemple

Conversion of $N = 687_d$ to hexadecimal on k digits

- $16^3 = 4096$ and $16^2 = 256$. Initially $k = 3$ and $i = 3$
- For i = 3 : $N = 687_d \leq 16^3$ then $\alpha = b_3 = 0_h$
- For i = 2 : $687_d > 256_d$, $687_d > 2 * 256_d = 512_d$ and $687_d < 3 * 256_d = 768_d$, so $\alpha \leftarrow 2_h$ and $N \leftarrow 687_d - 512_d$
  $\Rightarrow b_2 = \alpha = 2_h$ and $N \leftarrow 175$
- For i = 1 : $175_d > 10 * 16_d = 160$ and $175_d < 11 * 16_d = 176_d$ then $\alpha \leftarrow 10_d = A_h$ and $N \leftarrow 175_d - 160_d$
  $\Rightarrow b_1 = \alpha = A_h$ and $N \leftarrow 15_d$
- For i = 0 : $b_o = 15_d = F_h$.
- Result : $687_d = (b_3 b_2 b_1 b_0)_h = 02AF_h$.

Conversion from base 10 to base B

Exemple

Conversion of $N = 687_d$ to hexadecimal on k digits

- $16^3 = 4096$ and $16^2 = 256$. Initially $k = 3$ and $i = 3$
- For i = 3 : $N = 687_d \leq 16^3$ then $\alpha = b_3 = 0_h$
- For i = 2 : $687_d > 256_d$, $687_d > 2 * 256_d = 512_d$ and $687_d < 3 * 256_d = 768_d$, so $\alpha \leftarrow 2_h$ and $N \leftarrow 687_d - 512_d$
  $\Rightarrow b_2 = \alpha = 2_h$ and $N \leftarrow 175$
- For i = 1 : $175_d > 10 * 16_d = 160$ and $175_d < 11 * 16_d = 176_d$ then $\alpha \leftarrow 10_d = A_h$ and $N \leftarrow 175_d - 160_d$
  $\Rightarrow b_1 = \alpha = A_h$ and $N \leftarrow 15_d$
- For i = 0 : $b_o = 15_d = F_h$.
- Result : $687_d = (b_3 b_2 b_1 b_0)_h = 02AF_h$.

Conversion from base 10 to base B

Exemple

Conversion of $N = 687_d$ to hexadecimal on k digits

- $16^3 = 4096$ and $16^2 = 256$. Initially $k = 3$ and $i = 3$
- For i = 3 : $N = 687_d \leq 16^3$ then $\alpha = b_3 = 0_h$
- For i = 2 : $687_d > 256_d$, $687_d > 2 * 256_d = 512_d$ and $687_d < 3 * 256_d = 768_d$, so $\alpha \leftarrow 2_h$ and $N \leftarrow 687_d - 512_d$
  $\Rightarrow b_2 = \alpha = 2_h$ and $N \leftarrow 175$
- For i = 1 : $175_d > 10 * 16_d = 160$ and $175_d < 11 * 16_d = 176_d$ then $\alpha \leftarrow 10_d = A_h$ and $N \leftarrow 175_d - 160_d$
  $\Rightarrow b_1 = \alpha = A_h$ and $N \leftarrow 15_d$
- For i = 0 : $b_o = 15_d = F_h$.
- Result : $687_d = (b_3 b_2 b_1 b_0)_h = 02AF_h$.

Conversion from base 10 to base B

Exemple

Conversion of $N = 687_d$ to hexadecimal on k digits

- $16^3 = 4096$ and $16^2 = 256$. Initially $k = 3$ and $i = 3$
- For i = 3 : $N = 687_d \leq 16^3$ then $\alpha = b_3 = 0_h$
- For i = 2 : $687_d > 256_d$, $687_d > 2 * 256_d = 512_d$ and $687_d < 3 * 256_d = 768_d$, so $\alpha \leftarrow 2_h$ and $N \leftarrow 687_d - 512_d$ $\Rightarrow b_2 = \alpha = 2_h$ and $N \leftarrow 175$
- For i = 1 : $175_d > 10 * 16_d = 160$ and $175_d < 11 * 16_d = 176_d$ then $\alpha \leftarrow 10_d = A_h$ and $N \leftarrow 175_d - 160_d$ $\Rightarrow b_1 = \alpha = A_h$ and $N \leftarrow 15_d$
- For i = 0 : $b_o = 15_d = F_h$.
- Result : $687_d = (b_3 b_2 b_1 b_0)_h = 02AF_h$.

Conversion from base 10 to base B

Exemple

Conversion of $N = 687_d$ to hexadecimal on k digits

- $16^3 = 4096$ and $16^2 = 256$. Initially $k = 3$ and $i = 3$
- For i = 3 : $N = 687_d \leq 16^3$ then $\alpha = b_3 = 0_h$
- For i = 2 : $687_d > 256_d$, $687_d > 2 * 256_d = 512_d$ and $687_d < 3 * 256_d = 768_d$, so $\alpha \leftarrow 2_h$ and $N \leftarrow 687_d - 512_d$
  $\Rightarrow b_2 = \alpha = 2_h$ and $N \leftarrow 175$
- For i = 1 : $175_d > 10 * 16_d = 160$ and $175_d < 11 * 16_d = 176_d$ then $\alpha \leftarrow 10_d = A_h$ and $N \leftarrow 175_d - 160_d$
  $\Rightarrow b_1 = \alpha = A_h$ and $N \leftarrow 15_d$
- For i = 0 : $b_o = 15_d = F_h$.
- Result : $687_d = (b_3 b_2 b_1 b_0)_h = 02AF_h$.

Conversion from base 10 to base B

Exemple

Conversion of $N = 687_d$ to hexadecimal on k digits

- $16^3 = 4096$ and $16^2 = 256$. Initially $k = 3$ and $i = 3$
- For i = 3 : $N = 687_d \leq 16^3$ then $\alpha = b_3 = 0_h$
- For i = 2 : $687_d > 256_d$, $687_d > 2 * 256_d = 512_d$ and $687_d < 3 * 256_d = 768_d$, so $\alpha \leftarrow 2_h$ and $N \leftarrow 687_d - 512_d$
  $\Rightarrow b_2 = \alpha = 2_h$ and $N \leftarrow 175$
- For i = 1 : $175_d > 10 * 16_d = 160$ and $175_d < 11 * 16_d = 176_d$ then $\alpha \leftarrow 10_d = A_h$ and $N \leftarrow 175_d - 160_d$
  $\Rightarrow b_1 = \alpha = A_h$ and $N \leftarrow 15_d$
- For i = 0 : $b_o = 15_d = F_h$.
- Result : $687_d = (b_3 b_2 b_1 b_0)_h = 02AF_h$.

Conversion from base 10 to base B

Exemple

Conversion of $N = 687_d$ to hexadecimal on k digits

- $16^3 = 4096$ and $16^2 = 256$. Initially $k = 3$ and $i = 3$
- For i = 3 : $N = 687_d \leq 16^3$ then $\alpha = b_3 = 0_h$
- For i = 2 : $687_d > 256_d$, $687_d > 2 * 256_d = 512_d$ and $687_d < 3 * 256_d = 768_d$, so $\alpha \leftarrow 2_h$ and $N \leftarrow 687_d - 512_d$
  $\Rightarrow b_2 = \alpha = 2_h$ and $N \leftarrow 175$
- For i = 1 : $175_d > 10 * 16_d = 160$ and $175_d < 11 * 16_d = 176_d$ then $\alpha \leftarrow 10_d = A_h$ and $N \leftarrow 175_d - 160_d$
  $\Rightarrow b_1 = \alpha = A_h$ and $N \leftarrow 15_d$
- For i = 0 : $b_o = 15_d = F_h$.
- Result : $687_d = (b_3 b_2 b_1 b_0)_h = 02AF_h$.

Conversion form binary to hexadecimal

Conversion base 2 $\rightarrow$ base 16

1. Split the binary number in quartet (block of 4 bits) from right to left
2. Convert each quartet in hexadecimal digit

Example

10110010 $\rightarrow$
001110110010 $\rightarrow$
000101010010 $\rightarrow$
010101000110 $\rightarrow$

Conversion from hexadecimal to binary

Conversion base 16 $\rightarrow$ base 2

1. Convert each hexadecimal digit in binary quartet:

Exemple

$2A_{16} \rightarrow$
$EC5_{16} \rightarrow$
$1256_{16} \rightarrow$
$1B39F_{16} \rightarrow$

Addition of natural number : en decimal

Principle

- Addition of digit of the same rang, begin from the right
- Rules 1: when the addition of two digits is greater than 10, a carry is propagated on the left
- Rules 2: the carry of the preceding rank is added to the current rank digits

Example

$$\begin{array}{rcc} & 7^{1} & 5 \\ + & 1 & 7 \\ \hline & 9 & 2 \end{array}$$

- $5 + 7 = 12$: keep 2 and 1 is the carry
- $7 + 1 + 1 = 9$ keep 9 and 0 is the carry

Addition in base 2

Principle (1)

Addition of bits of the same rank

Bits rule

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 0 = 1
- $1 + 1 = 2_d = 10_b$ (keep 0 and carry = 1)

Principle (2)

- When there is a carry, carry is summed with the bit on the left
- So, three digits needs to be added (instead of 2)

Addition of three 1: 1 + 1 + 1 = 3_d=11_b, 1 and carry = 1

Example

Example on 8 bits

$$\begin{array}{cccccccc} & 0^{1} & 1^{1} & 1 & 0 & 0 & 0 & 1 & 1 \\ + & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ \hline & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{array}$$

Example on 4 bits

$$\begin{array}{cccccc} & 1 & 1^{1} & 1 & 0 & 0 \\ + & 0 & 1 & 1 & 0 \\ \hline & 1 & 0 & 0 & 1 & 0 \end{array}$$

Carry on addition of the last rank: the sum cannot be represented on 4 bits

Overflow

Overflow in base 2

There is an *overflow* when the result of the sum of two natural numbers represented on n bits cannot be represented on n bits

Example on 4 bits

$$\begin{array}{cccccc} {\scriptstyle 1} & & {\scriptstyle 1}{\scriptstyle 1} & & & \\ & 1 & & 1 & 0 & 0 \\ + & 0 & & 1 & 1 & 0 \\ \hline {\scriptstyle 1} & 0 & & 0 & 1 & 0 \end{array}$$

Overflow in base B

There is an *overflow* when the result of the sum of two natural numbers represented on n digits cannot be represented on n digits. The result is greater than the maximal value that can be represented on n digits.

Addition in base 16

Principle

- Addition is done by summing digits of same rank
- Carry propagation when the result is greater than 16

Example

Addition in base 16

Principle

- Addition is done by summing digits of same rank
- Carry propagation when the result is greater than 16

Example

$$\begin{array}{cccc} & 0 & D & 2 \\ + & 5 & 2 & 9 \\ \hline & 5 & F & B \end{array} \qquad \begin{array}{cccc} & 1^{1} & F & A \\ + & 7 & 3 & 4 \\ \hline & 9 & 2 & E \end{array} \qquad \begin{array}{cccc} {}^{1} & 5 & 6^{1} & 5 \\ + & C & 2 & E \\ \hline 1 & 1 & 9 & 3 \end{array}$$

Subtraction in decimal

- Subtract digits of same rank starting from right:

$$\begin{array}{rcc} & 7 & 5 \\ - & 1^{1} & 7 \\ \hline & 5 & 8 \end{array}$$

- For units:"7 is greater than 5, take one to the tens"
- There is a carry to brink and subtract to the left
- For the tens: subtract the taken tens (the carry) "7 - 1 - 1"
- Two value to subtract instead of one
- Remarque: $A - B$ with A and B natural numbers is not defined if $B > A$

Subtraction in binary

Same principle than in decimal:

Rule on bits

- 0 - 0 = 0
- 1 - 0 = 1
- 1 - 1 = 0
- 0 - 1 = 2 - 1 = 1 with a carry to subtract on the left
  $\Rightarrow$ take one on the left, the carry, that will then be subtracted on the left

Binary subtraction

- For subtraction of b_i to a_i in the subtraction $A - B$, there is 2 values to subtract: the carry and the value b_i
- The subtraction of to bits has also two output: the result and the carry

Subtraction Examples

On 8 bits

$$\begin{array}{rcccccccc} & 0 & 1 & {}^{1}1 & {}^{1}0 & {}^{1}0 & 0 & 1 & 1 \\ - & 0 & 0^{1} & 1^{1} & 1^{1} & 1 & 0 & 0 & 0 \\ \hline & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{array}$$

On 4 bits

$$\begin{array}{rcccc} & {}^{1}1 & {}^{1}0 & 0 & 0 \\ - \ {}^{1} & 1^{1} & 1 & 0 & 0 \\ \hline & 1 & 1 & 0 & 0 \end{array}$$

Multiplication and division

Multiplication and division of 2 integers are complexes
$\Rightarrow$ need specific hardware (Floating-Point Unit FPU)

Multiplication and division by a power of 2 in binary

- Multiplier and divide by $2^n \equiv$ shift to the left or the right of n bits
  - $N_d = (a_n a_{n-1} ... a_1 a_0)_b \Rightarrow 2N_d = (a_{n-1} ... a_1 a_0 0)_b$
  - $N_d = (a_n a_{n-1} ... a_1 a_0)_b \Rightarrow (N/2)_d = (0 a_n a_{n-1} ... a_1)_d$.

Multiplication and division by a power of B in base B

- Shift to the left by 1 (respectively n) is multiplying by B (respectively by B^n)
- Shift to the right by 1 (respectively n) is dividing by B (respectively by B^n).

Shifting and multiplication/division

Example in base 10

- $1024/10 = 102$: shift to the right by 1 (dividing 10^1)
- $1024/100 = 10$: shift to the right by 2 (dividing 10^2)
- $1024 * 10 = 10240$: shift to the left de 1

Example in base 2

- $1100_b >> 1_d = 0110_b = 6_d = 12_d/2_d$
- $0110_b >> 2_d = 0001_b = 1_d = 6_d/4_d$
- $0011_b << 1_d = 0110_b = 6_d = 3_d * 2_d$
- $0011_b << 2_d = 1100_b = 12_d = 3_d * 4_d$

Boolean Logic

Definitions

- Boolean logic = formalization of reasoning based on statement which could be `true` or `false`.
- Let $\mathbb{B}$ the alphabet $\mathbb{B}$ = {`True`, `False`} = {`T`,`F`} = {`1`,`0`}.
- Order on the elements of $\mathbb{B}$: $0 < 1$
- *Boolean variable* = a variable that could be `true` or `false`
- *Boolean function* = function on $\mathbb{B}^n \rightarrow \mathbb{B}$
- *Truth table* = enumeration of values taken by a boolean fonction *f* depending on its parameters

Example of Boolean Function

Function g

$$
\begin{array}{lcl}
g : \mathbb{B}^2 & \rightarrow & \mathbb{B} \\
x, y & \rightarrow & \text{if } x = 0, y = 0 \rightarrow g = 1 \\
 & & \text{if } x = 0, y = 1 \rightarrow g = 0 \\
 & & \text{if } x = 1, y = 0 \rightarrow g = 0 \\
 & & \text{if } x = 1, y = 1 \rightarrow g = 1
\end{array}
$$

Truth table of g:

x	y	g
0	0	1
0	1	0
1	0	0
1	1	1

Logic Operations

- Complement or negation: `NOT` (unary function)
  If $a = 0$ then $\overline{a} = 1$ and if $a = 1$ then $\overline{a} = 0$
- Addition/Disjunction: `OR` (binary function)
  $a + b = max(a, b)$
- Multiplication/Conjonction: `AND` (binary function)
  $a.b = min(a, b)$

Truth Tables of Basic Operations

NOT

a	$\overline{a}$
0	1
1	0

AND

a	b	$a.b$
0	0	0
0	1	0
1	0	0
1	1	1

OR

a	b	$a+b$
0	0	0
0	1	1
1	0	1
1	1	1

Boolean Algebra (1)

$< \mathbb{B}, 0, 1, +, ., - >$ is a Boolean Algebra since:

- Addition and multiplication are associative:
  - $\forall x, y, z \in \mathbb{B}^3 \ (x + y) + z = x + (y + z)$
  - $\forall x, y, z \in \mathbb{B}^3 \ (x.y).z = x.(y.z)$
- Addition and Multiplication are commutative:
  - $\forall x, y \in \mathbb{B}^2 \ x + y = y + x$
  - $\forall x, y \in \mathbb{B}^2 \ x.y = y.x$
- Addition and multiplication are distributive with respect to each other (not the case for algebra on natural numbers !!!):
  - $\forall x, y, z \in \mathbb{B}^3 \ x.(y + z) = x.y + x.z$
  - $\forall x, y, z \in \mathbb{B}^3 \ x + (y.z) = (x + y).(x + z)$

Boolean Algebra (2)

$< \mathbb{B}, 0, 1, +, ., - >$ is a Boolean Algebra since:

- 0 is the neutral element for addition, 1 for multiplication:
  - $\forall x \in \mathbb{B}\ x + 0 = x$
  - $\forall x \in \mathbb{B}\ x.1 = x$
- Sum of an element and its complement is 1:
  - $\forall x \in \mathbb{B}\ x + \overline{x} = 1$
- Product of an element and its complement is 0:
  - $\forall x \in \mathbb{B}\ x.\overline{x} = 0$

Other Properties

- Morgan Laws:
  - $\overline{x + y} = \overline{x}.\overline{y}$
  - $\overline{x.y} = \overline{x} + \overline{y}$
- Simplification rules
  - Involution law : $\forall x \in \mathbb{B}, \quad \overline{\overline{x}} = x$
  - Absorbing element: $\forall x \in \mathbb{B}, \quad x.0 = 0$ and $x + 1 = 1$
  - Idempotence : $\forall x \in \mathbb{B}\ x.x = x$ and $x + x = x$

Algebraic Expression of a Boolean Function

- Any Boolean function f can be expressed only using constant `0` and `1`, name of Boolean parameters of f and operators `+`, `.` and $-$ of Boolean Algebra.
- Algebraic expression of a Boolean function f can be constructed from its truth table as:

(1) a disjunctive normal form (DNF)

(2) a conjunctive normal form (CNF)

Algebraic Expression of a Boolean Function

- Any Boolean function *f* can be expressed only using constant `0` and `1`, name of Boolean parameters of *f* and operators `+`, `.` and $-$ of Boolean Algebra.
- Algebraic expression of a Boolean function *f* can be constructed from its truth table as:

(1) a disjunctive normal form (DNF)

(2) a conjunctive normal form (CNF)

Disjunctive Normal Form

Definition (Disjunctive Normal Form)

A formula in Disjunctive Normal Form of a function f, is a logical formula which is a disjunction of conjunctive clauses representing f.
It is an OR of ANDs, a sum of products.

Construction

It can be constructed by the disjunction (OR) of terms where f holds 1 in its truth table. Each term is the product (AND) of variables names of f, negated if the variable is 0.

Example

- A
- $A + B$
- $A + (B.C)$
- $(\overline{A}.B) + (C.\overline{D})$

Disjunctive Normal Form

Definition (Disjunctive Normal Form)

A formula in Disjunctive Normal Form of a function f, is a logical formula which is a disjunction of conjunctive clauses representing f.
It is an OR of ANDs, a sum of products.

Construction

It can be constructed by the disjunction (OR) of terms where f holds 1 in its truth table. Each term is the product (AND) of variables names of f, negated if the variable is 0.

Example

- A
- $A + B$
- $A + (B.C)$
- $(\overline{A}.B) + (C.\overline{D})$

Conjunctive Normal Form

Definition (Conjunctive Normal Form)

A formula in Conjunctive Normal Form of a function f, is a logical formula which is a conjunction of disjunctive clauses representing f.
It is an ANDs of ORs, a product of sums.

Construction

It can be constructed by the conjunction (AND) of terms where f holds 0 in its truth table. Each term is the sum (OR) of variables names of f, negated if the variable is 1.

Example

- A
- $A.B$
- $A.(B + C)$
- $(\overline{A} + B).(C + \overline{D})$

Conjunctive Normal Form

Definition (Conjunctive Normal Form)

A formula in Conjunctive Normal Form of a function f, is a logical formula which is a conjunction of disjunctive clauses representing f.
It is an ANDs of ORs, a product of sums.

Construction

It can be constructed by the conjunction (AND) of terms where f holds 0 in its truth table. Each term is the sum (OR) of variables names of f, negated if the variable is 1.

Example

- A
- $A.B$
- $A.(B + C)$
- $(\overline{A} + B).(C + \overline{D})$

Equivalence between Boolean Functions or Expressions

- Two Boolean functions are equivalent if they have the same truth table
- Two arithmetic Boolean expressions are equivalent if they can be rewritten to a same third expression
- The Conjunctive Normal Form can be obtained from the Disjunctive Normal Form using the Involution law $\overline{\overline{f}} = f$

Example

$$(\overline{A}.B) + (C.\overline{D}) = \overline{\overline{(\overline{A}.B) + (C.\overline{D})}} \quad \text{Involution}$$

$$= \overline{\overline{(\overline{A}.B)}.\overline{(C.\overline{D})}} \quad \text{De Morgan's}$$

$$= \overline{(A + \overline{B}).(\overline{C} + D)} \quad \text{De Morgan's}$$

$$= \overline{(A.\overline{C}) + (A.D) + (\overline{B}.\overline{C}) + (\overline{B}.D)} \quad \text{Distributivity}$$

$$= \overline{(A.\overline{C})}.\overline{(A.D)}.\overline{(\overline{B}.\overline{C})}.\overline{(\overline{B}.D)} \quad \text{De Morgan's}$$

$$= (\overline{A} + C).(\overline{A} + \overline{D}).(B + C).(B + \overline{D}) \quad \text{De Morgan's}$$

Equivalence between Boolean Functions or Expressions

- Two Boolean functions are equivalent if they have the same truth table
- Two arithmetic Boolean expressions are equivalent if they can be rewritten to a same third expression
- The Conjunctive Normal Form can be obtained from the Disjunctive Normal Form using the Involution law $\overline{\overline{f}} = f$

Example

$$(\overline{A}.B) + (C.\overline{D}) = \overline{\overline{(\overline{A}.B) + (C.\overline{D})}} \quad \text{Involution}$$

$$= \overline{\overline{(\overline{A}.B)}.\overline{(C.\overline{D})}} \quad \text{De Morgan's}$$

$$= \overline{(A + \overline{B}).(\overline{C} + D)} \quad \text{De Morgan's}$$

$$= \overline{(A.\overline{C}) + (A.D) + (\overline{B}.\overline{C}) + (\overline{B}.D)} \quad \text{Distributivity}$$

$$= \overline{(A.\overline{C})}.\overline{(A.D)}.\overline{(\overline{B}.\overline{C})}.\overline{(\overline{B}.D)} \quad \text{De Morgan's}$$

$$= (\overline{A} + C).(\overline{A} + \overline{D}).(B + C).(B + \overline{D}) \quad \text{De Morgan's}$$

Equivalence between Boolean Functions or Expressions

- Two Boolean functions are equivalent if they have the same truth table
- Two arithmetic Boolean expressions are equivalent if they can be rewritten to a same third expression
- The Conjunctive Normal Form can be obtained from the Disjunctive Normal Form using the Involution law $\overline{\overline{f}} = f$

Example

$$(\overline{A}.B) + (C.\overline{D}) = \overline{\overline{(\overline{A}.B) + (C.\overline{D})}} \quad \text{Involution}$$

$$= \overline{\overline{(\overline{A}.B)}.\overline{(C.\overline{D})}} \quad \text{De Morgan's}$$

$$= \overline{(A + \overline{B}).(\overline{C} + D)} \quad \text{De Morgan's}$$

$$= \overline{(A.\overline{C}) + (A.D) + (\overline{B}.\overline{C}) + (\overline{B}.D)} \quad \text{Distributivity}$$

$$= \overline{(A.\overline{C})}.\overline{(A.D)}.\overline{(\overline{B}.\overline{C})}.\overline{(\overline{B}.D)} \quad \text{De Morgan's}$$

$$= (\overline{A} + C).(\overline{A} + \overline{D}).(B + C).(B + \overline{D}) \quad \text{De Morgan's}$$

Equivalence between Boolean Functions or Expressions

- Two Boolean functions are equivalent if they have the same truth table
- Two arithmetic Boolean expressions are equivalent if they can be rewritten to a same third expression
- The Conjunctive Normal Form can be obtained from the Disjunctive Normal Form using the Involution law $\overline{\overline{f}} = f$

Example

$$(\overline{A}.B) + (C.\overline{D}) = \overline{\overline{(\overline{A}.B) + (C.\overline{D})}} \quad \text{Involution}$$

$$= \overline{\overline{(\overline{A}.B)}.\overline{(C.\overline{D})}} \quad \text{De Morgan's}$$

$$= \overline{(A + \overline{B}).(\overline{C} + D)} \quad \text{De Morgan's}$$

$$= \overline{(A.\overline{C}) + (A.D) + (\overline{B}.\overline{C}) + (\overline{B}.D)} \quad \text{Distributivity}$$

$$= \overline{(A.\overline{C})}.\overline{(A.D)}.\overline{(\overline{B}.\overline{C})}.\overline{(\overline{B}.D)} \quad \text{De Morgan's}$$

$$= (\overline{A} + C).(\overline{A} + \overline{D}).(B + C).(B + \overline{D}) \quad \text{De Morgan's}$$

Equivalence between Boolean Functions or Expressions

- Two Boolean functions are equivalent if they have the same truth table
- Two arithmetic Boolean expressions are equivalent if they can be rewritten to a same third expression
- The Conjunctive Normal Form can be obtained from the Disjunctive Normal Form using the Involution law $\overline{\overline{f}} = f$

Example

$$(\overline{A}.B) + (C.\overline{D}) = \overline{\overline{(\overline{A}.B) + (C.\overline{D})}} \quad \text{Involution}$$

$$= \overline{\overline{(\overline{A}.B)}.\overline{(C.\overline{D})}} \quad \text{De Morgan's}$$

$$= \overline{(A + \overline{B}).(\overline{C} + D)} \quad \text{De Morgan's}$$

$$= \overline{(A.\overline{C}) + (A.D) + (\overline{B}.\overline{C}) + (\overline{B}.D)} \quad \text{Distributivity}$$

$$= \overline{(A.\overline{C})}.\overline{(A.D)}.\overline{(\overline{B}.\overline{C})}.\overline{(\overline{B}.D)} \quad \text{De Morgan's}$$

$$= (\overline{A} + C).(\overline{A} + \overline{D}).(B + C).(B + \overline{D}) \quad \text{De Morgan's}$$

Equivalence between Boolean Functions or Expressions

- Two Boolean functions are equivalent if they have the same truth table
- Two arithmetic Boolean expressions are equivalent if they can be rewritten to a same third expression
- The Conjunctive Normal Form can be obtained from the Disjunctive Normal Form using the Involution law $\overline{\overline{f}} = f$

Example

$$(\overline{A}.B) + (C.\overline{D}) = \overline{\overline{(\overline{A}.B) + (C.\overline{D})}} \quad \text{Involution}$$

$$= \overline{\overline{(\overline{A}.B)}.\overline{(C.\overline{D})}} \quad \text{De Morgan's}$$

$$= \overline{(A + \overline{B}).(\overline{C} + D)} \quad \text{De Morgan's}$$

$$= \overline{(A.\overline{C}) + (A.D) + (\overline{B}.\overline{C}) + (\overline{B}.D)} \quad \text{Distributivity}$$

$$= \overline{(A.\overline{C})}.\overline{(A.D)}.\overline{(\overline{B}.\overline{C})}.\overline{(\overline{B}.D)} \quad \text{De Morgan's}$$

$$= (\overline{A} + C).(\overline{A} + \overline{D}).(B + C).(B + \overline{D}) \quad \text{De Morgan's}$$

Equivalence between Boolean Functions or Expressions

- Two Boolean functions are equivalent if they have the same truth table
- Two arithmetic Boolean expressions are equivalent if they can be rewritten to a same third expression
- The Conjunctive Normal Form can be obtained from the Disjunctive Normal Form using the Involution law $\overline{\overline{f}} = f$

Example

$$(\overline{A}.B) + (C.\overline{D}) = \overline{\overline{(\overline{A}.B) + (C.\overline{D})}} \quad \text{Involution}$$

$$= \overline{\overline{(\overline{A}.B)}.\overline{(C.\overline{D})}} \quad \text{De Morgan's}$$

$$= \overline{(A + \overline{B}).(\overline{C} + D)} \quad \text{De Morgan's}$$

$$= \overline{(A.\overline{C}) + (A.D) + (\overline{B}.\overline{C}) + (\overline{B}.D)} \quad \text{Distributivity}$$

$$= \overline{(A.\overline{C})}.\overline{(A.D)}.\overline{(\overline{B}.\overline{C})}.\overline{(\overline{B}.D)} \quad \text{De Morgan's}$$

$$= (\overline{A} + C).(\overline{A} + \overline{D}).(B + C).(B + \overline{D}) \quad \text{De Morgan's}$$