# Codage des informations

## SIN1 - Cours 1

J. Villemejane - julien.villemejane@u-pec.fr

IUT Créteil-Vitry
Département GEII
Université Paris-Est Créteil

Année universitaire 2013-2014

# Plan du cours

# Le codage : pourquoi et comment ?

**Différents types d'informations dans le monde du numérique**

Texte, nombres, sons, instructions, images...

Une information numérique peut être de *deux types* :

- une **instruction**, qui représente une opération réalisée par un organe de calcul (un microprocesseur par exemple) ;
- une **donnée** :
  - non numérique (caractère alphanumérique) ;
  - numérique :
    - entiers naturels (0 ; 1 ; 315 ...)
    - entiers relatifs (-1578 ; -15 ; -1 ...)
    - réels (3.1415 ; 4587.598 ...)

Comment **coder** ces informations électriquement ?

# Le codage : pourquoi et comment ?

## Différents types d'informations dans le monde du numérique

Texte, nombres, sons, instructions, images...

Une information numérique peut être de *deux types* :

- une **instruction**, qui représente une opération réalisée par un organe de calcul (un microprocesseur par exemple) ;
- une **donnée** :
  - non numérique (caractère alphanumérique) ;
  - numérique :
    - entiers naturels (0 ; 1 ; 315 ...)
    - entiers relatifs (-1578 ; -15 ; -1 ...)
    - réels (3.1415 ; 4587.598 ...)

Comment **coder** ces informations électriquement ?

# Le codage : pourquoi et comment ?

## Différents types d'informations dans le monde du numérique

Texte, nombres, sons, instructions, images...

Une information numérique peut être de *deux types* :

- une **instruction**, qui représente une opération réalisée par un organe de calcul (un microprocesseur par exemple) ;
- une **donnée** :
  - non numérique (caractère alphanumérique) ;
  - numérique :
    - entiers naturels (0 ; 1 ; 315 ...)
    - entiers relatifs (-1578 ; -15 ; -1 ...)
    - réels (3.1415 ; 4587.598 ...)

Comment **coder** ces informations électriquement ?

# Le codage : pourquoi et comment ?

## Différents types d'informations dans le monde du numérique

Texte, nombres, sons, instructions, images...

Une information numérique peut être de *deux types* :

- une **instruction**, qui représente une opération réalisée par un organe de calcul (un microprocesseur par exemple) ;
- une **donnée** :
  - non numérique (caractère alphanumérique) ;
  - numérique :
    - entiers naturels (0 ; 1 ; 315 ...)
    - entiers relatifs (-1578 ; -15 ; -1 ...)
    - réels (3.1415 ; 4587.598 ...)

## Comment **coder** ces informations électriquement ?

# Le codage : pourquoi et comment ?

## Représentation de l'information

Les informations numériques sont transmises par des **signaux électriques**.

Afin d'avoir un *langage universel*, ces données sont représentées sous forme **binaire**, c'est à dire une suite de **0** et de **1**.

L'information élémentaire est appelé **BIT** (*BInary digiT*).

Un **mot binaire** de $n$ bits permet d'obtenir $2^n$ combinaisons différentes.

### Combinaisons

| | | |
|---|---|---|
| 1 bit | 2 combinaisons | $\{0, 1\}$ |
| 2 bits | 4 combinaisons | $\{00, 01, 10, 11\}$ |
| 3 bits | 8 combinaisons | $\{000, 001, 010, 011, 100, 101, 110, 111\}$ |

**1 octet = 8 bits = 256 combinaisons**

# Le codage : pourquoi et comment ?

## Représentation de l'information

Les informations numériques sont transmises par des **signaux électriques**.

Afin d'avoir un *langage universel*, ces données sont représentées sous forme **binaire**, c'est à dire une suite de **0** et de **1**.

L'information élémentaire est appelé **BIT** (*BInary digiT*).

Un **mot binaire** de $n$ bits permet d'obtenir $2^n$ combinaisons différentes.

### Combinaisons

| | | |
|---|---|---|
| 1 bit | 2 combinaisons | $\{0, 1\}$ |
| 2 bits | 4 combinaisons | $\{00, 01, 10, 11\}$ |
| 3 bits | 8 combinaisons | $\{000, 001, 010, 011, 100, 101, 110, 111\}$ |

**1 octet = 8 bits = 256 combinaisons**

# Le codage : pourquoi et comment ?

## Représentation de l'information

Les informations numériques sont transmises par des **signaux électriques**.

Afin d'avoir un *langage universel*, ces données sont représentées sous forme **binaire**, c'est à dire une suite de **0** et de **1**.

L'information élémentaire est appelé **BIT** (*BInary digiT*).

Un **mot binaire** de $n$ bits permet d'obtenir $2^n$ combinaisons différentes.

### Combinaisons

| | | |
|---|---|---|
| 1 bit | 2 combinaisons | $\{0, 1\}$ |
| 2 bits | 4 combinaisons | $\{00, 01, 10, 11\}$ |
| 3 bits | 8 combinaisons | $\{000, 001, 010, 011, 100, 101, 110, 111\}$ |

**1 octet = 8 bits = 256 combinaisons**

# Le codage : pourquoi et comment ?

## Représentation de l'information

Les informations numériques sont transmises par des **signaux électriques**.

Afin d'avoir un *langage universel*, ces données sont représentées sous forme **binaire**, c'est à dire une suite de **0** et de **1**.

L'information élémentaire est appelé **BIT** (*BInary digiT*).

Un **mot binaire** de $n$ bits permet d'obtenir $2^n$ combinaisons différentes.

### Combinaisons

| | | |
|---|---|---|
| 1 bit | 2 combinaisons | {0, 1} |
| 2 bits | 4 combinaisons | {00, 01, 10, 11} |
| 3 bits | 8 combinaisons | {000, 001, 010, 011, 100, 101, 110, 111} |

**1 octet = 8 bits = 256 combinaisons**

# Le codage : pourquoi et comment ?

## Représentation de l'information

Les informations numériques sont transmises par des **signaux électriques**.

Afin d'avoir un *langage universel*, ces données sont représentées sous forme **binaire**, c'est à dire une suite de **0** et de **1**.

L'information élémentaire est appelé **BIT** (*BInary digiT*).

Un **mot binaire** de $n$ bits permet d'obtenir $2^n$ combinaisons différentes.

### Combinaisons

| | | |
|---|---|---|
| 1 bit | 2 combinaisons | {0, 1} |
| 2 bits | 4 combinaisons | {00, 01, 10, 11} |
| 3 bits | 8 combinaisons | {000, 001, 010, 011, 100, 101, 110, 111} |

**1 octet = 8 bits = 256 combinaisons**

# Le codage : pourquoi et comment ?

## Codage des caractères

**Comment coder des caractères alphanumériques ?**

Nécessité de **transmettre des messages** "rapidement"

code morse (1835)

Apparition de **nouveaux codes** avec l'informatique

- ASCII *American Standard Code for Information Interchange* (7 bits) ;
- UNICODE UTF-16 et UTF-32 (16 et 32 bits)

# Le codage : pourquoi et comment ?

## Codage des caractères

**Comment coder des caractères alphanumériques ?**

Nécessité de **transmettre des messages** "rapidement"

code morse (1835)

Apparition de **nouveaux codes** avec l'informatique

- ASCII *American Standard Code for Information Interchange* (7 bits) ;
- UNICODE UTF-16 et UTF-32 (16 et 32 bits)

# Le codage : pourquoi et comment ?

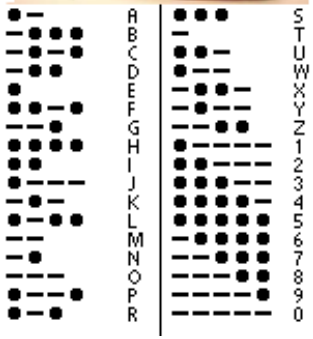
## Codage des caractères

**Comment coder des caractères alphanumériques ?**

Nécessité de **transmettre des messages** "rapidement"

Apparition de **nouveaux codes** avec l'informatique

- ASCII *American Standard Code for Information Interchange* (7 bits) ;
- UNICODE UTF-16 et UTF-32 (16 et 32 bits)




code morse (1835)

# Le codage : pourquoi et comment ?

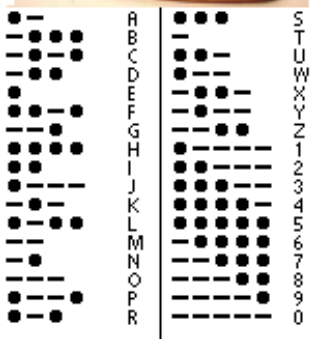
## Codage des caractères

**Comment coder des caractères alphanumériques ?**

Nécessité de **transmettre des messages** "rapidement"

Apparition de **nouveaux codes** avec l'informatique

- ASCII *American Standard Code for Information Interchange* (7 bits) ;
- UNICODE UTF-16 et UTF-32 (16 et 32 bits)




code morse (1835)

# Le codage : pourquoi et comment ?

## Codage des caractères - ASCII

Normalisé en 1967, il propose un jeu de **128 caractères** (7 bits).

| | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|
| 0000 | NUL | DLE | SP | 0 | @ | P | ‘ | p |
| 0001 | SOH | DC1 | ! | 1 | A | Q | a | q |
| 0010 | STX | DC2 | “ | 2 | B | R | b | r |
| 0011 | ETX | DC3 | # | 3 | C | S | c | s |
| 0100 | EOT | DC4 | $ | 4 | D | T | d | t |
| 0101 | ENQ | NAK | % | 5 | E | U | e | u |
| 0110 | ACK | SYN | & | 6 | F | V | f | v |
| 0111 | BEL | ETB | ’ | 7 | G | W | g | w |
| 1000 | BS | CAN | ( | 8 | H | X | h | x |
| 1001 | HT | EM | ) | 9 | I | Y | i | y |
| 1010 | LF | SUB | * | : | J | Z | j | z |
| 1011 | VT | ESC | + | ; | K | [ | k | { |
| 1100 | FF | FS | , | ¡ | L | \ | l | — |
| 1101 | CR | GS | - | = | M | ] | m | } |
| 1110 | SO | RS | . | ¿ | N | ^ | n | ~ |
| 1111 | SI | US | / | ? | O | _ | o | DEL |

**Limitation à la langue anglaise**

Dépourvue de toute décoration (accent, cédille, tilde...)
Non adaptée à tous les alphabets (chinois, cyrillique...)

# Le codage : pourquoi et comment ?

## Codage des caractères - ASCII

Normalisé en 1967, il propose un jeu de **128 caractères** (7 bits).

| | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|
| 0000 | NUL | DLE | SP | 0 | @ | P | ‘ | p |
| 0001 | SOH | DC1 | ! | 1 | A | Q | a | q |
| 0010 | STX | DC2 | “ | 2 | B | R | b | r |
| 0011 | ETX | DC3 | # | 3 | C | S | c | s |
| 0100 | EOT | DC4 | $ | 4 | D | T | d | t |
| 0101 | ENQ | NAK | % | 5 | E | U | e | u |
| 0110 | ACK | SYN | & | 6 | F | V | f | v |
| 0111 | BEL | ETB | ’ | 7 | G | W | g | w |
| 1000 | BS | CAN | ( | 8 | H | X | h | x |
| 1001 | HT | EM | ) | 9 | I | Y | i | y |
| 1010 | LF | SUB | * | : | J | Z | j | z |
| 1011 | VT | ESC | + | ; | K | [ | k | { |
| 1100 | FF | FS | , | ¡ | L | \ | l | — |
| 1101 | CR | GS | - | = | M | ] | m | } |
| 1110 | SO | RS | . | ¿ | N | ^ | n | ~ |
| 1111 | SI | US | / | ? | O | _ | o | DEL |

**Limitation à la langue anglaise**

Dépourvue de toute décoration (accent, cédille, tilde...)
Non adaptée à tous les alphabets (chinois, cyrillique...)

# Le codage : pourquoi et comment ?

## Codage des caractères - Unicode / ISO/IEC 10646

**Unicode** (1991) a été développée dans le but de remplacer l'utilisation de pages de code nationales (tel que l'ASCII).

### Unicode version 6.0 (janvier 2012)

- 137 468 caractères à usage privé ;
- 109 242 lettres ou syllabes, chiffres ou nombres, symboles divers, signes diacritiques (accent...) et signes de ponctuation ;
- plusieurs centaines de caractères de contrôle ou modificateurs spéciaux.

Unicode est mis à jour régulièrement : 5.2 (2009), 6.0 (2011), 6.1 (2012), 6.3 (2013)

Dernière mise à jour : Mandaic, Batak, Ethiopic Extended-A, Brahmi, Playing Cards, **Emoticons**, Alchemical Symbols

# Le codage : pourquoi et comment ?

## Codage des caractères - Unicode / ISO/IEC 10646

**Unicode** (1991) a été développée dans le but de remplacer l'utilisation de pages de code nationales (tel que l'ASCII).

### Unicode version 6.0 (janvier 2012)

- 137 468 caractères à usage privé ;
- 109 242 lettres ou syllabes, chiffres ou nombres, symboles divers, signes diacritiques (accent...) et signes de ponctuation ;
- plusieurs centaines de caractères de contrôle ou modificateurs spéciaux.

Unicode est mis à jour régulièrement : 5.2 (2009), 6.0 (2011), 6.1 (2012), 6.3 (2013)

Dernière mise à jour : Mandaic, Batak, Ethiopic Extended-A, Brahmi, Playing Cards, **Emoticons**, Alchemical Symbols

# Le codage : pourquoi et comment ?

## Codage des caractères - Unicode / ISO/IEC 10646

**Unicode** (1991) a été développée dans le but de remplacer l'utilisation de pages de code nationales (tel que l'ASCII).

### Unicode version 6.0 (janvier 2012)

- 137 468 caractères à usage privé ;
- 109 242 lettres ou syllabes, chiffres ou nombres, symboles divers, signes diacritiques (accent...) et signes de ponctuation ;
- plusieurs centaines de caractères de contrôle ou modificateurs spéciaux.

Unicode est mis à jour régulièrement : 5.2 (2009), 6.0 (2011), 6.1 (2012), 6.3 (2013)

Dernière mise à jour : Mandaic, Batak, Ethiopic Extended-A, Brahmi, Playing Cards, **Emoticons**, Alchemical Symbols

# Le codage : pourquoi et comment ?

## Codage des caractères - ASCII/Unicode

ASCII/8859-1 Text

| | |
|---|---|
| A | 0100 0001 |
| S | 0101 0011 |
| C | 0100 0011 |
| I | 0100 1001 |
| I | 0100 1001 |
| / | 0010 1111 |
| 8 | 0011 1000 |
| 8 | 0011 1000 |
| 5 | 0011 0101 |
| 9 | 0011 1001 |
| - | 0010 1101 |
| 1 | 0011 0001 |
| | 0010 0000 |
| t | 0111 0100 |
| e | 0110 0101 |
| x | 0111 1000 |
| t | 0111 0100 |

Unicode Text

| | |
|---|---|
| A | 0000 0000 0100 0001 |
| S | 0000 0000 0101 0011 |
| C | 0000 0000 0100 0011 |
| I | 0000 0000 0100 1001 |
| I | 0000 0000 0100 1001 |
| | 0000 0000 0010 0000 |
| 天 | 0101 1001 0010 1001 |
| 地 | 0101 0111 0011 0000 |
| | 0000 0000 0010 0000 |
| س | 0000 0110 0011 0011 |
| ل | 0000 0110 0100 0100 |
| ا | 0000 0110 0010 0111 |
| م | 0000 0110 0100 0101 |
| | 0000 0000 0010 0000 |
| α | 0000 0011 1011 0001 |
| ⋦ | 0010 0010 0111 0000 |
| γ | 0000 0011 1011 0011 |

*www.unicode.org*

# Codage des entiers naturels

## Rappel sur l'addition et la multiplication en binaire

| | | |
|---|---|---|
| $0 + 0 = 0$ | | $0 \times 0 = 0$ |
| $0 + 1 = 1$ | | $0 \times 1 = 0$ |
| $1 + 0 = 1$ | | $1 \times 0 = 0$ |
| $1 + 1 = 0$ | (et une retenue) | $1 \times 1 = 1$ |

# Codage des entiers naturels

## A partir de l'ASCII...

$$\begin{array}{rcl} 8 & \rightarrow & 00111000 \\ +\ 5 & \rightarrow & +\ 00110101 \\ \hline 13 & \neq & 01101101 \end{array} \quad = \text{'m'}$$

## Codage alphanumérique non prévu pour le calcul

Nécessité d'un **codage des nombres** différent pour permettre le calcul

# Codage des entiers naturels

## A partir de l'ASCII...

$$\begin{array}{rcl} 8 & \rightarrow & 0\,0\,1\,1\,1\,0\,0\,0 \\ +\ 5 & \rightarrow & +\ 0\,0\,1\,1\,0\,1\,0\,1 \\ \hline 13 & \neq & 0\,1\,1\,0\,1\,1\,0\,1 \end{array} \quad = \quad \text{'m'}$$

## Codage alphanumérique non prévu pour le calcul

Nécessité d'un **codage des nombres** différent pour permettre le calcul

# Codage des entiers naturels

## A partir de l'ASCII...

$$\begin{array}{rcl} 8 & \rightarrow & 0\,0\,1\,1\,1\,0\,0\,0 \\ +\ 5 & \rightarrow & +\ 0\,0\,1\,1\,0\,1\,0\,1 \\ \hline 13 & \neq & 0\,1\,1\,0\,1\,1\,0\,1 \end{array} \quad = \quad \text{'m'}$$

## Codage alphanumérique non prévu pour le calcul

Nécessité d'un **codage des nombres** différent pour permettre le calcul

# Codage des entiers naturels

## A partir de l'ASCII...

$$\begin{array}{rcl} 8 & \rightarrow & 0\,0\,1\,1\,1\,0\,0\,0 \\ +\ 5 & \rightarrow & +\ 0\,0\,1\,1\,0\,1\,0\,1 \\ \hline 13 & \neq & 0\,1\,1\,0\,1\,1\,0\,1 \end{array} \quad = \quad \text{'m'}$$


## Codage alphanumérique non prévu pour le calcul

Nécessité d'un **codage des nombres** différent pour permettre le calcul

# Codage des entiers naturels

## A partir de l'ASCII...

$$\begin{array}{rcl} 8 & \rightarrow & 0\,0\,1\,1\,1\,0\,0\,0 \\ +\ 5 & \rightarrow & +\ 0\,0\,1\,1\,0\,1\,0\,1 \\ \hline 13 & \neq & 0\,1\,1\,0\,1\,1\,0\,1 \quad = \quad \text{'m'} \end{array}$$


## Codage alphanumérique non prévu pour le calcul

Nécessité d'un **codage des nombres** différent pour permettre le calcul

# Codage des entiers naturels

## Base décimale (Base 10)

La base décimale (dite base 10) possède 10 chiffres $\{0,1,...,9\}$.

Un **nombre entier** est une somme de termes où chacun est une **puissance de dix** multipliée par un chiffre.
Chacun de ces termes correspond à un *poids* : unité, dizaine, centaine...

| | Centaine | | Dizaine | | Unité |
|---|---|---|---|---|---|
| $345 =$ | $3$ | $+$ | $4$ | $+$ | $5$ |
| | $\times$ | | $\times$ | | $\times$ |
| Rang | $10^2$ | | $10^1$ | | $10^0$ |

# Codage des entiers naturels

## Base décimale (Base 10)

La base décimale (dite base 10) possède 10 chiffres $\{0,1,...,9\}$.

Un **nombre entier** est une somme de termes où chacun est une **puissance de dix** multipliée par un chiffre.
Chacun de ces termes correspond à un *poids* : unité, dizaine, centaine...

| | Centaine | | Dizaine | | Unité |
|---|---|---|---|---|---|
| $345 =$ | $3$ | $+$ | $4$ | $+$ | $5$ |
| | $\times$ | | $\times$ | | $\times$ |
| Rang | $10^2$ | | $10^1$ | | $10^0$ |

# Codage des entiers naturels

## Base décimale (Base 10)

La base décimale (dite base 10) possède 10 chiffres $\{0,1,...,9\}$.

Un **nombre entier** est une somme de termes où chacun est une **puissance de dix** multipliée par un chiffre.
Chacun de ces termes correspond à un *poids* : unité, dizaine, centaine...

| | Centaine | | Dizaine | | Unité |
|---|---|---|---|---|---|
| $345 =$ | $3$ | $+$ | $4$ | $+$ | $5$ |
| | x | | x | | x |
| Rang | $10^2$ | | $10^1$ | | $10^0$ |

# Codage des entiers naturels

## Base décimale (Base 10)

La base décimale (dite base 10) possède 10 chiffres $\{0,1,...,9\}$.

Un **nombre entier** est une somme de termes où chacun est une **puissance de dix** multipliée par un chiffre.
Chacun de ces termes correspond à un *poids* : unité, dizaine, centaine...

| | Centaine | | Dizaine | | Unité |
|---|---|---|---|---|---|
| $345 =$ | $3$ | $+$ | $4$ | $+$ | $5$ |
| | x | | x | | x |
| Rang | $10^2$ | | $10^1$ | | $10^0$ |

# Codage des entiers naturels

## Base décimale (Base 10)

La base décimale (dite base 10) possède 10 chiffres $\{0,1,...,9\}$.

Un **nombre entier** est une somme de termes où chacun est une **puissance de dix** multipliée par un chiffre.
Chacun de ces termes correspond à un *poids* : unité, dizaine, centaine...

| | Centaine | | Dizaine | | Unité |
|---|---|---|---|---|---|
| $345 =$ | $3$ | $+$ | $4$ | $+$ | $5$ |
| | $\times$ | | $\times$ | | $\times$ |
| Rang | $10^{2}$ | | $10^{1}$ | | $10^{0}$ |

# Codage des entiers naturels

## Base décimale (Base 10)

La base décimale (dite base 10) possède 10 chiffres $\{0,1,...,9\}$.

Un **nombre entier** est une somme de termes où chacun est une **puissance de dix** multipliée par un chiffre.
Chacun de ces termes correspond à un *poids* : unité, dizaine, centaine...

| | Centaine | | Dizaine | | Unité |
|---|---|---|---|---|---|
| $345 =$ | $3$ | $+$ | $4$ | $+$ | $5$ |
| | x | | x | | x |
| Rang | $10^{2}$ | | $10^{1}$ | | $10^{0}$ |

# Codage des entiers naturels

## Base binaire (Base 2)

La base binaire (dite base 2) possède 2 symboles $\{0,1\}$.

### Dynamique de codage des naturels

| Nb bits | Dynamique de codage |
|---|---|
| 4 | 0 à 15 |
| 8 | 0 à 255 |
| 16 | 0 à 65 535 |
| 32 | 0 à 4 294 967 295 |

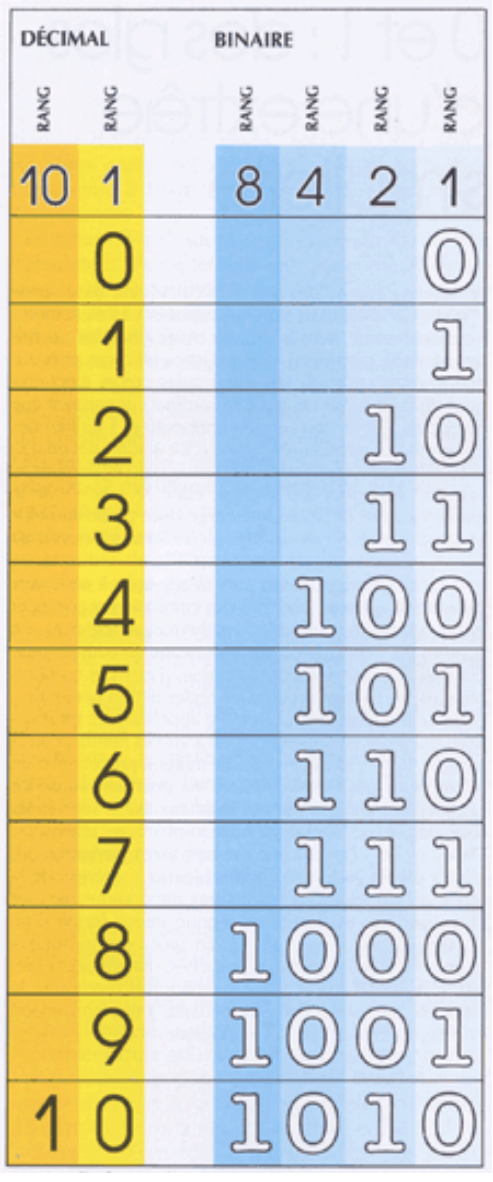
| $1001 =$ | $1$ | $+$ | $0$ | $+$ | $0$ | $+$ | $1$ |
|---|---|---|---|---|---|---|---|
| | x | | x | | x | | x |
| Rang | $2^3$ | | $2^2$ | | $2^1$ | | $2^0$ |
| Puissance | 3 | | 2 | | 1 | | 0 |

*Par exemple*

$(1001)_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$

$(1001)_2 = 1 \cdot 8_{10} + 0 \cdot 4_{10} + 0 \cdot 2_{10} + 1 \cdot 1_{10}$

$(1001)_2 = (9)_{10}$



| DÉCIMAL | | BINAIRE | | | |
|---|---|---|---|---|---|
| RANG | RANG | RANG | RANG | RANG | RANG |
| 10 | 1 | | 8 | 4 | 2 | 1 |
| | 0 | | | | | 0 |
| | 1 | | | | | 1 |
| | 2 | | | | 1 | 0 |
| | 3 | | | | 1 | 1 |
| | 4 | | | 1 | 0 | 0 |
| | 5 | | | 1 | 0 | 1 |
| | 6 | | | 1 | 1 | 0 |
| | 7 | | | 1 | 1 | 1 |
| | 8 | | 1 | 0 | 0 | 0 |
| | 9 | | 1 | 0 | 0 | 1 |
| 1 | 0 | | 1 | 0 | 1 | 0 |

# Codage des entiers naturels

## Base binaire (Base 2)

La base binaire (dite base 2) possède 2 symboles $\{0,1\}$.

### Dynamique de codage des naturels

| Nb bits | Dynamique de codage |
|---|---|
| 4 | 0 à 15 |
| 8 | 0 à 255 |
| 16 | 0 à 65 535 |
| 32 | 0 à 4 294 967 295 |

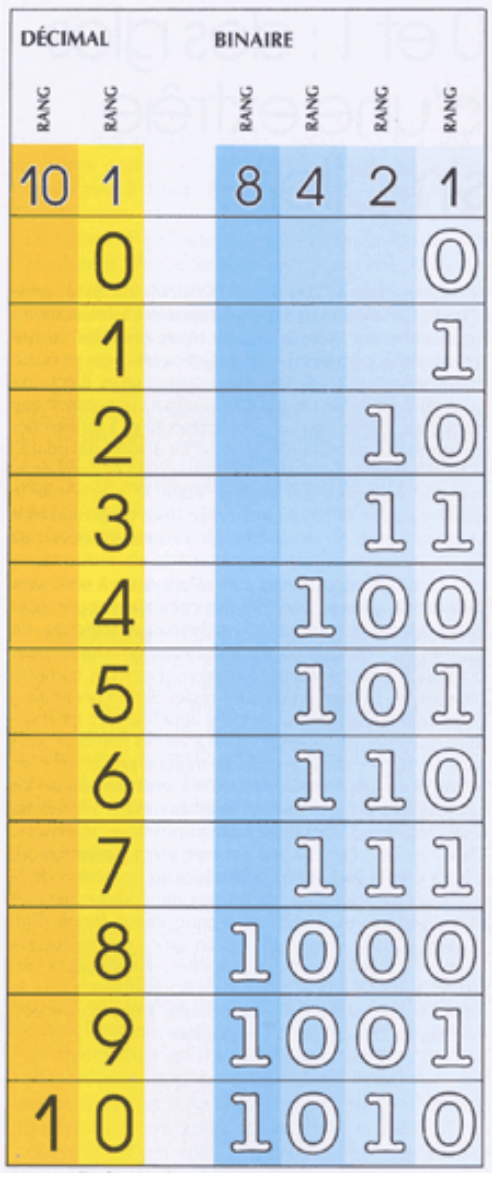
| $1001 =$ | $1$ | $+$ | $0$ | $+$ | $0$ | $+$ | $1$ |
|---|---|---|---|---|---|---|---|
| | x | | x | | x | | x |
| Rang | $2^3$ | | $2^2$ | | $2^1$ | | $2^0$ |
| Puissance | 3 | | 2 | | 1 | | 0 |

*Par exemple*

$(1001)_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$

$(1001)_2 = 1 \cdot 8_{10} + 0 \cdot 4_{10} + 0 \cdot 2_{10} + 1 \cdot 1_{10}$

$(1001)_2 = (9)_{10}$



| DÉCIMAL | | BINAIRE | | | |
|---|---|---|---|---|---|
| RANG | RANG | RANG | RANG | RANG | RANG |
| 10 | 1 | | 8 | 4 | 2 | 1 |
| | 0 | | | | | 0 |
| | 1 | | | | | 1 |
| | 2 | | | | 1 | 0 |
| | 3 | | | | 1 | 1 |
| | 4 | | | 1 | 0 | 0 |
| | 5 | | | 1 | 0 | 1 |
| | 6 | | | 1 | 1 | 0 |
| | 7 | | | 1 | 1 | 1 |
| | 8 | | 1 | 0 | 0 | 0 |
| | 9 | | 1 | 0 | 0 | 1 |
| 1 | 0 | | 1 | 0 | 1 | 0 |

# Codage des entiers naturels

## Base binaire (Base 2)

La base binaire (dite base 2) possède 2 symboles $\{0,1\}$.

### Dynamique de codage des naturels

| Nb bits | Dynamique de codage |
|---|---|
| 4 | 0 à 15 |
| 8 | 0 à 255 |
| 16 | 0 à 65 535 |
| 32 | 0 à 4 294 967 295 |

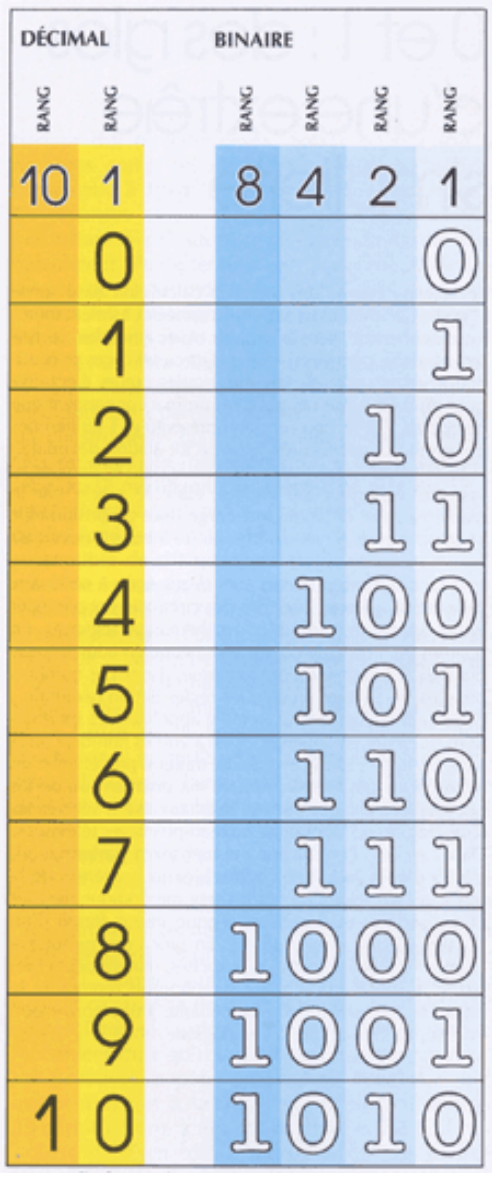
| $1001 =$ | $1$ | $+$ | $0$ | $+$ | $0$ | $+$ | $1$ |
|---|---|---|---|---|---|---|---|
| | x | | x | | x | | x |
| Rang | $2^3$ | | $2^2$ | | $2^1$ | | $2^0$ |
| Puissance | 3 | | 2 | | 1 | | 0 |

*Par exemple*

$(1001)_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$

$(1001)_2 = 1 \cdot 8_{10} + 0 \cdot 4_{10} + 0 \cdot 2_{10} + 1 \cdot 1_{10}$

$(1001)_2 = (9)_{10}$



| DÉCIMAL | | BINAIRE | | | |
|---|---|---|---|---|---|
| RANG | RANG | RANG | RANG | RANG | RANG |
| 10 | 1 | | 8 | 4 | 2 | 1 |
| | 0 | | | | | 0 |
| | 1 | | | | | 1 |
| | 2 | | | | 1 | 0 |
| | 3 | | | | 1 | 1 |
| | 4 | | | 1 | 0 | 0 |
| | 5 | | | 1 | 0 | 1 |
| | 6 | | | 1 | 1 | 0 |
| | 7 | | | 1 | 1 | 1 |
| | 8 | | 1 | 0 | 0 | 0 |
| | 9 | | 1 | 0 | 0 | 1 |
| 1 | 0 | | 1 | 0 | 1 | 0 |

# Codage des entiers naturels

## Base hexadécimale (Base 16)

La base hexadécimale (dite base 16) possède 16 symboles $\{0,1,...,9,A,B,C,D,E,F\}$ (où A=10, B=11,... F=15).

| $D7A =$ | $D$ | $+$ | $7$ | $+$ | $A$ |
|---|---|---|---|---|---|
| | x | | x | | x |
| Rang | $16^2$ | | $16^1$ | | $16^0$ |
| Rang | $256_{10}$ | | $16_{10}$ | | $1_{10}$ |
| Puissance | 2 | | 1 | | 0 |

*Par exemple*

$$(D7A)_{16} = D \cdot 16^2 + 7 \cdot 16^1 + A \cdot 16^0$$
$$(D7A)_{16} = 13 \cdot 256_{10} + 7 \cdot 16_{10} + 10 \cdot 1_{10}$$
$$(D7A)_{16} = 3328_{10} + 112_{10} + 10_{10}$$
$$(D7A)_{16} = (3450)_{10}$$

| Décimal | | Hexadécimal | |
|---|---|---|---|
| RANG | RANG | RANG | RANG |
| 10 | 1 | 16 | 1 |
| | 0 | | 0 |
| | 1 | | 1 |
| | 2 | | 2 |
| | 3 | | 3 |
| . | . | . | . |
| | 9 | | 9 |
| 1 | 0 | | A |
| 1 | 1 | | B |
| 1 | 2 | | C |
| 1 | 3 | | D |
| 1 | 4 | | E |
| 1 | 5 | | F |
| 1 | 6 | 1 | 0 |
| 1 | 7 | 1 | 1 |

# Codage des entiers naturels

## Base hexadécimale (Base 16)

La base hexadécimale (dite base 16) possède 16 symboles {0,1,...,9,A,B,C,D,E,F} (où A=10, B=11,... F=15).

| $D7A =$ | $D$ | $+$ | $7$ | $+$ | $A$ |
|---|---|---|---|---|---|
| | x | | x | | x |
| Rang | $16^{2}$ | | $16^{1}$ | | $16^{0}$ |
| Rang | $256_{10}$ | | $16_{10}$ | | $1_{10}$ |
| Puissance | 2 | | 1 | | 0 |

*Par exemple*

$$(D7A)_{16} = D \cdot 16^{2} + 7 \cdot 16^{1} + A \cdot 16^{0}$$
$$(D7A)_{16} = 13 \cdot 256_{10} + 7 \cdot 16_{10} + 10 \cdot 1_{10}$$
$$(D7A)_{16} = 3328_{10} + 112_{10} + 10_{10}$$
$$(D7A)_{16} = (3450)_{10}$$

| Décimal | | Hexadécimal | |
|---|---|---|---|
| RANG | RANG | RANG | RANG |
| 10 | 1 | 16 | 1 |
| | 0 | | 0 |
| | 1 | | 1 |
| | 2 | | 2 |
| | 3 | | 3 |
| . | . | . | . |
| | 9 | | 9 |
| 1 | 0 | | A |
| 1 | 1 | | B |
| 1 | 2 | | C |
| 1 | 3 | | D |
| 1 | 4 | | E |
| 1 | 5 | | F |
| 1 | 6 | 1 | 0 |
| 1 | 7 | 1 | 1 |

# Codage des entiers naturels

## Base hexadécimale (Base 16)

La base hexadécimale (dite base 16) possède 16 symboles {0,1,...,9,A,B,C,D,E,F} (où A=10, B=11,... F=15).

| $D7A =$ | $D$ | $+$ | $7$ | $+$ | $A$ |
|---|---|---|---|---|---|
| | x | | x | | x |
| Rang | $16^{2}$ | | $16^{1}$ | | $16^{0}$ |
| Rang | $256_{10}$ | | $16_{10}$ | | $1_{10}$ |
| Puissance | 2 | | 1 | | 0 |

*Par exemple*

$$(D7A)_{16} = D \cdot 16^{2} + 7 \cdot 16^{1} + A \cdot 16^{0}$$
$$(D7A)_{16} = 13 \cdot 256_{10} + 7 \cdot 16_{10} + 10 \cdot 1_{10}$$
$$(D7A)_{16} = 3328_{10} + 112_{10} + 10_{10}$$
$$(D7A)_{16} = (3450)_{10}$$

| Décimal | | Hexadécimal | |
|---|---|---|---|
| RANG | RANG | RANG | RANG |
| 10 | 1 | 16 | 1 |
| | 0 | | 0 |
| | 1 | | 1 |
| | 2 | | 2 |
| | 3 | | 3 |
| . | . | . | . |
| | 9 | | 9 |
| 1 | 0 | | A |
| 1 | 1 | | B |
| 1 | 2 | | C |
| 1 | 3 | | D |
| 1 | 4 | | E |
| 1 | 5 | | F |
| 1 | 6 | 1 | 0 |
| 1 | 7 | 1 | 1 |

# Codage des entiers naturels

## Transcodage - Base B $\rightarrow$ 10

Transcodage = Passage d'une base à une autre

### Base $B$ vers 10

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $N_B =$ | $a_n$ | + | $a_{n-1}$ | + ... + | $a_1$ | + | $a_0$ |
| | x | | x | | x | | x |
| Rang | $B^{\mathbf{n}}$ | | $B^{\mathbf{n-1}}$ | ... | $B^{\mathbf{1}}$ | | $B^{\mathbf{0}}$ |
| Puissance | n | | n-1 | ... | 1 | | 0 |

avec $a_i \in \{0, 1, ...p-1\}$ et $a_n \neq 0$

$B = 16$ (hexadécimal)

$(A5E)_{16} = 10 \cdot 16^2 + 5 \cdot 16^1 + 14 \cdot 16^0 = 2560_{10} + 80_{10} + 14_{10}$

$(A5E)_{16} = (2654)_{10}$

$B = 2$ (binaire)

$(1010\ 1100)_2 = 1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0$

$(1010\ 1100)_2 = (172)_{10}$

# Codage des entiers naturels

## Transcodage - Base B $\rightarrow$ 10

Transcodage = Passage d'une base à une autre

### Base $B$ vers 10

| $N_B =$ | $a_n$ | + | $a_{n-1}$ | + ... + | $a_1$ | + | $a_0$ |
|---|---|---|---|---|---|---|---|
| | x | | x | | x | | x |
| Rang | $B^{\mathbf{n}}$ | | $B^{\mathbf{n-1}}$ | ... | $B^{\mathbf{1}}$ | | $B^{\mathbf{0}}$ |
| Puissance | n | | n-1 | ... | 1 | | 0 |

avec $a_i \in \{0, 1, ...p-1\}$ et $a_n \neq 0$

$B = 16$ (hexadécimal)

$(A5E)_{16} = 10 \cdot 16^2 + 5 \cdot 16^1 + 14 \cdot 16^0 = 2560_{10} + 80_{10} + 14_{10}$

$(A5E)_{16} = (2654)_{10}$

$B = 2$ (binaire)

$(1010\ 1100)_2 = 1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0$

$(1010\ 1100)_2 = (172)_{10}$

# Codage des entiers naturels

## Transcodage - Base B → 10

Transcodage = Passage d'une base à une autre

### Base $B$ vers 10

| $N_B =$ | $a_n$ | + | $a_{n-1}$ | + ... + | $a_1$ | + | $a_0$ |
|---|---|---|---|---|---|---|---|
| | x | | x | | x | | x |
| Rang | $B^{\mathbf{n}}$ | | $B^{\mathbf{n-1}}$ | ... | $B^{\mathbf{1}}$ | | $B^{\mathbf{0}}$ |
| Puissance | n | | n-1 | ... | 1 | | 0 |

avec $a_i \in \{0, 1, ...p-1\}$ et $a_n \neq 0$

$B = 16$ (hexadécimal)

$(A5E)_{16} = 10 \cdot 16^2 + 5 \cdot 16^1 + 14 \cdot 16^0 = 2560_{10} + 80_{10} + 14_{10}$

$(A5E)_{16} = (2654)_{10}$

$B = 2$ (binaire)

$(1010\ 1100)_2 = 1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0$

$(1010\ 1100)_2 = (172)_{10}$

# Codage des entiers naturels

## Transcodage - Base 10 → B

### Base 10 vers $p$

Division successive par $p$

$$\begin{array}{r|l} N & B \\ \hline a_0 & N_1 \end{array} \rightarrow \begin{array}{r|l} N_1 & B \\ \hline a_1 & N_2 \end{array} \rightarrow \ldots \begin{array}{r|l} N_n & B \\ \hline a_n & 0 \end{array}$$

### Base 10 vers 2

| | |
|---|---|
| 172 | 2 |
| 0 | 86 |
| | 2 |
| 0 | 43 |
| | 2 |
| 1 | 21 |
| | 2 |
| 1 | 10 |
| | 2 |
| 0 | 5 |
| | 2 |
| 1 | 2 |
| | 2 |
| 0 | 1 |
| | 2 |
| 1 | 0 |

# Codage des entiers naturels

## Transcodage - Base 10 → B

### Base 10 vers $p$

Division successive par $p$

$$\begin{array}{c|c} N & B \\ \hline a_0 & N_1 \end{array} \rightarrow \begin{array}{c|c} N_1 & B \\ \hline a_1 & N_2 \end{array} \rightarrow \ldots \begin{array}{c|c} N_n & B \\ \hline a_n & 0 \end{array}$$

### Base 10 vers 2

| | |
|---|---|
| 172 | 2 |
| 0 | 86 |
| | 2 |
| 0 | 43 |
| | 2 |
| 1 | 21 |
| | 2 |
| 1 | 10 |
| | 2 |
| 0 | 5 |
| | 2 |
| 1 | 2 |
| | 2 |
| 0 | 1 |
| | 2 |
| 1 | 0 |

# Codage des entiers naturels

## Transcodage - Base 2 $\leftrightarrow$ 16

### Binaire vers hexadécimal (et inversement)

| Binaire | | | | Hexa |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 1 | 1 | 3 |
| 0 | 1 | 0 | 0 | 4 |
| 0 | 1 | 0 | 1 | 5 |
| 0 | 1 | 1 | 0 | 6 |
| 0 | 1 | 1 | 1 | 7 |

| Binaire | | | | Hexa |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 8 |
| 1 | 0 | 0 | 1 | 9 |
| 1 | 0 | 1 | 0 | A |
| 1 | 0 | 1 | 1 | B |
| 1 | 1 | 0 | 0 | C |
| 1 | 1 | 0 | 1 | D |
| 1 | 1 | 1 | 0 | E |
| 1 | 1 | 1 | 1 | F |

*Par exemple*

| Hexadécimal | $D$ | $7$ | $A$ |
|---|---|---|---|
| Binaire | $1101$ | $0111$ | $1010$ |

**8 bits = 1 octet = 2 hexadécimaux**

# Codage des entiers naturels

## Transcodage - Base 2 $\leftrightarrow$ 16

### Binaire vers hexadécimal (et inversement)

| Binaire | | | | Hexa |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 1 | 1 | 3 |
| 0 | 1 | 0 | 0 | 4 |
| 0 | 1 | 0 | 1 | 5 |
| 0 | 1 | 1 | 0 | 6 |
| 0 | 1 | 1 | 1 | 7 |

| Binaire | | | | Hexa |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 8 |
| 1 | 0 | 0 | 1 | 9 |
| 1 | 0 | 1 | 0 | A |
| 1 | 0 | 1 | 1 | B |
| 1 | 1 | 0 | 0 | C |
| 1 | 1 | 0 | 1 | D |
| 1 | 1 | 1 | 0 | E |
| 1 | 1 | 1 | 1 | F |

*Par exemple*

| Hexadécimal | $D$ | $7$ | $A$ |
|---|---|---|---|
| Binaire | $1101$ | $0111$ | $1010$ |

**8 bits = 1 octet = 2 hexadécimaux**

# Codage des entiers naturels

## Transcodage - Base 2 $\leftrightarrow$ 16

### Binaire vers hexadécimal (et inversement)

| Binaire | | | | Hexa |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 1 | 1 | 3 |
| 0 | 1 | 0 | 0 | 4 |
| 0 | 1 | 0 | 1 | 5 |
| 0 | 1 | 1 | 0 | 6 |
| 0 | 1 | 1 | 1 | 7 |

| Binaire | | | | Hexa |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 8 |
| 1 | 0 | 0 | 1 | 9 |
| 1 | 0 | 1 | 0 | A |
| 1 | 0 | 1 | 1 | B |
| 1 | 1 | 0 | 0 | C |
| 1 | 1 | 0 | 1 | D |
| 1 | 1 | 1 | 0 | E |
| 1 | 1 | 1 | 1 | F |

*Par exemple*

| Hexadécimal | $D$ | $7$ | $A$ |
|---|---|---|---|
| Binaire | $1101$ | $0111$ | $1010$ |

**8 bits = 1 octet = 2 hexadécimaux**

# Codage des entiers relatifs

## Problématique

**Comment coder les nombres négatifs ?**

Rajout d'un bit de signe

- 0 pour un nombre positif
- 1 pour un nombre négatif

### Dynamique de codage des entier relatifs

| Nb bits | Dynamique de codage |
|---|---|
| 4 | -8 à +7 |
| 8 | -128 à +127 |
| 16 | -32 768 à 32 767 |
| 32 | -2 147 483 648 à 2 147 483 647 |

- Utilisation du code **complément à deux**

# Codage des entiers relatifs

## Problématique

**Comment coder les nombres négatifs ?**

Rajout d'un bit de signe

- 0 pour un nombre positif
- 1 pour un nombre négatif

### Dynamique de codage des entier relatifs

| Nb bits | Dynamique de codage |
|---|---|
| 4 | -8 à +7 |
| 8 | -128 à +127 |
| 16 | -32 768 à 32 767 |
| 32 | -2 147 483 648 à 2 147 483 647 |

- Utilisation du code **complément à deux**

# Codage des entiers relatifs

## Problématique

**Comment coder les nombres négatifs ?**

Rajout d'un bit de signe

- 0 pour un nombre positif
- 1 pour un nombre négatif

### Dynamique de codage des entier relatifs

| Nb bits | Dynamique de codage |
|---|---|
| 4 | -8 à +7 |
| 8 | -128 à +127 |
| 16 | -32 768 à 32 767 |
| 32 | -2 147 483 648 à 2 147 483 647 |

- Utilisation du code **complément à deux**

# Codage des entiers relatifs

## Problématique

**Comment coder les nombres négatifs ?**

Rajout d'un bit de signe

- 0 pour un nombre positif
- 1 pour un nombre négatif

### Dynamique de codage des entier relatifs

| Nb bits | Dynamique de codage |
|---|---|
| 4 | -8 à +7 |
| 8 | -128 à +127 |
| 16 | -32 768 à 32 767 |
| 32 | -2 147 483 648 à 2 147 483 647 |

- Utilisation du code **complément à deux**

# Codage des entiers relatifs

## Complément à 1 et complément à 2 - Algorithme

**Si nécessaire**, rechercher le nombre minimal de bits $p$ pour coder le nombre $n$ en appliquant la relation :

$$p \geq \log_2 |n| + 1$$

- **si** $n \geq 0$ (nombres de $0$ à $2^{p-1} - 1$), le code est strictement le code binaire naturel étendu à $p$ bits (en complétant à gauche par des $0$). Le bit de poids fort est égal à $0$.
- **si** $n < 0$ (nombres de $-2^{p-1}$ à $-1$) :
  1. coder $|n|$ en binaire en complétant à gauche par des $0$ pour obtenir un code sur $p$ bits ;
  2. inverser tous les bits de la représentation binaire (**complèment à un** ou C1) ;
  3. ajouter 1 au résultat (**complèment à deux** ou C2)

# Codage des entiers relatifs

## Complément à 1 et complément à 2 - Algorithme

**Si nécessaire**, rechercher le nombre minimal de bits $p$ pour coder le nombre $n$ en appliquant la relation :

$$p \geq \log_2 |n| + 1$$

- **si** $n \geq 0$ (nombres de $0$ à $2^{p-1} - 1$), le code est strictement le code binaire naturel étendu à $p$ bits (en complétant à gauche par des $0$). Le bit de poids fort est égal à 0.
- **si** $n < 0$ (nombres de $-2^{p-1}$ à $-1$) :
  1. coder $|n|$ en binaire en complétant à gauche par des $0$ pour obtenir un code sur $p$ bits ;
  2. inverser tous les bits de la représentation binaire (**complèment à un** ou C1) ;
  3. ajouter 1 au résultat (**complèment à deux** ou C2)

# Codage des entiers relatifs

## Complément à 1 et complément à 2 - Algorithme

**Si nécessaire**, rechercher le nombre minimal de bits $p$ pour coder le nombre $n$ en appliquant la relation :

$$p \geq \log_2 |n| + 1$$

- **si** $n \geq 0$ (nombres de $0$ à $2^{p-1} - 1$), le code est strictement le code binaire naturel étendu à $p$ bits (en complétant à gauche par des $0$). Le bit de poids fort est égal à 0.
- **si** $n < 0$ (nombres de $-2^{p-1}$ à $-1$) :
  1. coder $|n|$ en binaire en complétant à gauche par des $0$ pour obtenir un code sur $p$ bits ;
  2. inverser tous les bits de la représentation binaire (**complèment à un** ou C1) ;
  3. ajouter $1$ au résultat (**complèment à deux** ou C2)

# Codage des entiers relatifs

## Complément à 1 et complément à 2 - Algorithme

**Si nécessaire**, rechercher le nombre minimal de bits $p$ pour coder le nombre $n$ en appliquant la relation :

$$p \geq \log_2 |n| + 1$$

- **si** $n \geq 0$ (nombres de $0$ à $2^{p-1} - 1$), le code est strictement le code binaire naturel étendu à $p$ bits (en complétant à gauche par des $0$). Le bit de poids fort est égal à 0.
- **si** $n < 0$ (nombres de $-2^{p-1}$ à $-1$) :
  1. coder $|n|$ en binaire en complétant à gauche par des $0$ pour obtenir un code sur $p$ bits ;
  2. inverser tous les bits de la représentation binaire (**complèment à un** ou C1) ;
  3. ajouter $1$ au résultat (**complèment à deux** ou C2)

# Codage des entiers relatifs

## Complément à 1 et complément à 2 - Algorithme

**Si nécessaire**, rechercher le nombre minimal de bits $p$ pour coder le nombre $n$ en appliquant la relation :

$$p \geq \log_2 |n| + 1$$

- **si** $n \geq 0$ (nombres de $0$ à $2^{p-1} - 1$), le code est strictement le code binaire naturel étendu à $p$ bits (en complétant à gauche par des $0$). Le bit de poids fort est égal à 0.
- **si** $n < 0$ (nombres de $-2^{p-1}$ à $-1$) :
  1. coder $|n|$ en binaire en complétant à gauche par des $0$ pour obtenir un code sur $p$ bits ;
  2. inverser tous les bits de la représentation binaire (**complèment à un** ou C1) ;
  3. ajouter $1$ au résultat (**complèment à deux** ou C2)

# Codage des entiers relatifs

## Complément à 1 et complément à 2 - Algorithme

**Si nécessaire**, rechercher le nombre minimal de bits $p$ pour coder le nombre $n$ en appliquant la relation :

$$p \geq \log_2 |n| + 1$$

- **si** $n \geq 0$ (nombres de $0$ à $2^{p-1} - 1$), le code est strictement le code binaire naturel étendu à $p$ bits (en complétant à gauche par des $0$). Le bit de poids fort est égal à 0.
- **si** $n < 0$ (nombres de $-2^{p-1}$ à $-1$) :
  1. coder $|n|$ en binaire en complétant à gauche par des $0$ pour obtenir un code sur $p$ bits ;
  2. inverser tous les bits de la représentation binaire (**complèment à un** ou C1) ;
  3. ajouter $1$ au résultat (**complèment à deux** ou C2)

# Codage des réels

## Problématique

Dans un intervalle donné, il y a :

- un **nombre limité d'entiers**
- mais un **nombre infini de réels**.

Quelque soit le codage envisagé, il n'y a qu'un nombre fini de valeurs possibles entraînant une **imprécision systématique** du codage.

L'imprécision ne sera jamais plus grande que l'écart entre 2 codes consécutifs, écart qui est une valeur caractéristique du code choisi (souvent appelé *EPSILON*).

# Codage des réels

## Problématique

Dans un intervalle donné, il y a :

- un **nombre limité d'entiers**
- mais un **nombre infini de réels**.

Quelque soit le codage envisagé, il n'y a qu'un nombre fini de valeurs possibles entraînant une **imprécision systématique** du codage.

L'imprécision ne sera jamais plus grande que l'écart entre 2 codes consécutifs, écart qui est une valeur caractéristique du code choisi (souvent appelé *EPSILON*).

# Codage des réels

## Problématique

Dans un intervalle donné, il y a :

- un **nombre limité d'entiers**
- mais un **nombre infini de réels**.

Quelque soit le codage envisagé, il n'y a qu'un nombre fini de valeurs possibles entraînant une **imprécision systématique** du codage.

L'imprécision ne sera jamais plus grande que l'écart entre 2 codes consécutifs, écart qui est une valeur caractéristique du code choisi (souvent appelé *EPSILON*).

# Codage des réels

## Virgule fixe

On considère un dénominateur *implicite* $2^p$ commun à toute la représentation. Le code équivalent est généralement appelé $Q_p$.

La méthode de codage d'un réel $x$ sur $n$ bits en code $Q_p$ peut être décrite par :

1. multiplier $x$ par $2^p$ ;
2. ne conserver que la partie entière du résultat précédent et la coder en code C2.

**Représentation d'un réel $x$ sur un nombre $n$ de bits en code $Q_p$**

| $x =$ | $a_n$ | $+$ | $a_{n-1}$ | $+ \ldots +$ | $a_1$ | $+$ | $a_0$ |
|---|---|---|---|---|---|---|---|
| | x | | x | | x | | x |
| Rang | $p^{n-p-1}$ | | $p^{n-p-2}$ | ... | $2^{-p+1}$ | | $2^{-p}$ |
| Puissance | n-p-1 | | n-p-2 | ... | -p+1 | | -p |

L'erreur est inférieure à $2^{-p}$

Cette méthode de codage est souvent utilisée dans les applications de traitement audio ou vidéo, la majorité des processeurs de traitement du signal étant basée sur cette technique.

# Codage des réels

## Virgule fixe

On considère un dénominateur *implicite* $2^p$ commun à toute la représentation. Le code équivalent est généralement appelé $Q_p$.

La méthode de codage d'un réel $x$ sur $n$ bits en code $Q_p$ peut être décrite par :

1. multiplier $x$ par $2^p$ ;
2. ne conserver que la partie entière du résultat précédent et la coder en code C2.

### Représentation d'un réel $x$ sur un nombre $n$ de bits en code $Q_p$

| $x =$ | $a_n$ | $+$ | $a_{n-1}$ | $+ \ldots +$ | $a_1$ | $+$ | $a_0$ |
|---|---|---|---|---|---|---|---|
| | x | | x | | x | | x |
| Rang | $p^{n-p-1}$ | | $p^{n-p-2}$ | ... | $2^{-p+1}$ | | $2^{-p}$ |
| Puissance | n-p-1 | | n-p-2 | ... | -p+1 | | -p |

L'erreur est inférieure à $2^{-p}$

Cette méthode de codage est souvent utilisée dans les applications de traitement audio ou vidéo, la majorité des processeurs de traitement du signal étant basée sur cette technique.

# Codage des réels

## Virgule fixe

On considère un dénominateur *implicite* $2^p$ commun à toute la représentation. Le code équivalent est généralement appelé $Q_p$.

La méthode de codage d'un réel $x$ sur $n$ bits en code $Q_p$ peut être décrite par :

1. multiplier $x$ par $2^p$ ;
2. ne conserver que la partie entière du résultat précédent et la coder en code C2.

### Représentation d'un réel $x$ sur un nombre $n$ de bits en code $Q_p$

| $x =$ | $a_n$ | $+$ | $a_{n-1}$ | $+ \ldots +$ | $a_1$ | $+$ | $a_0$ |
|---|---|---|---|---|---|---|---|
| | x | | x | | x | | x |
| Rang | $p^{\mathbf{n-p-1}}$ | | $p^{\mathbf{n-p-2}}$ | ... | $2^{\mathbf{-p+1}}$ | | $2^{\mathbf{-p}}$ |
| Puissance | n-p-1 | | n-p-2 | ... | -p+1 | | -p |

L'erreur est inférieure à $2^{-p}$

Cette méthode de codage est souvent utilisée dans les applications de traitement audio ou vidéo, la majorité des processeurs de traitement du signal étant basée sur cette technique.

# Codage des réels

## Virgule fixe

On considère un dénominateur *implicite* $2^p$ commun à toute la représentation. Le code équivalent est généralement appelé $Q_p$.

La méthode de codage d'un réel $x$ sur $n$ bits en code $Q_p$ peut être décrite par :

1. multiplier $x$ par $2^p$ ;
2. ne conserver que la partie entière du résultat précédent et la coder en code C2.

### Représentation d'un réel $x$ sur un nombre $n$ de bits en code $Q_p$

| $x =$ | $a_n$ | + | $a_{n-1}$ | + ... + | $a_1$ | + | $a_0$ |
|---|---|---|---|---|---|---|---|
| | x | | x | | x | | x |
| Rang | $p^{n-p-1}$ | | $p^{n-p-2}$ | ... | $2^{-p+1}$ | | $2^{-p}$ |
| Puissance | n-p-1 | | n-p-2 | ... | -p+1 | | -p |

L'erreur est inférieure à $2^{-p}$

Cette méthode de codage est souvent utilisée dans les applications de traitement audio ou vidéo, la majorité des processeurs de traitement du signal étant basée sur cette technique.

# Codage des réels

## Virgule fixe - Exemple

Codage de $0,893_{10}$ en code $Q_7$ sur 8 bits

1. $0,893_{10} \times 2^7 = 114,304_{10} \approx 114_{10}$
2. $114_{10} = 0111\ 0010_{C2}$

Vérification

$0111\ 0010_{C2} = 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4} + 0 \cdot 2^{-5} + 1 \cdot 2^{-6} + 0 \cdot 2^{-7}$

$0111\ 0010_{C2} = 0,5 + 0,25 + 0,125 + 0,015625$

$0111\ 0010_{C2} = 0,890625_{10}$

Erreur de $0,893 - 0,890625 = 0,002375 < 2^{-7} = 0,0078125$

# Codage des réels

## Virgule fixe - Exemple

Codage de $0,893_{10}$ en code $Q_7$ sur 8 bits

1. $0,893_{10} \times 2^7 = 114,304_{10} \approx 114_{10}$
2. $114_{10} = 0111\ 0010_{C2}$

Vérification

$0111\ 0010_{C2} = 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4} + 0 \cdot 2^{-5} + 1 \cdot 2^{-6} + 0 \cdot 2^{-7}$

$0111\ 0010_{C2} = 0,5 + 0,25 + 0,125 + 0,015625$

$0111\ 0010_{C2} = 0,890625_{10}$

Erreur de $0,893 - 0,890625 = 0,002375 < 2^{-7} = 0,0078125$

# Codage des réels

## Virgule fixe - Exemple

Codage de $0,893_{10}$ en code $Q_7$ sur 8 bits

1. $0,893_{10} \times 2^7 = 114,304_{10} \approx 114_{10}$
2. $114_{10} = 0111\ 0010_{C2}$

Vérification

$0111\ 0010_{C2} = 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4} + 0 \cdot 2^{-5} + 1 \cdot 2^{-6} + 0 \cdot 2^{-7}$

$0111\ 0010_{C2} = 0,5 + 0,25 + 0,125 + 0,015625$

$0111\ 0010_{C2} = 0,890625_{10}$

Erreur de $0,893 - 0,890625 = 0,002375 < 2^{-7} = 0,0078125$

# Codage des réels

## Virgule fixe - Exemple

Codage de $0,893_{10}$ en code $Q_7$ sur 8 bits

1. $0,893_{10} \times 2^7 = 114,304_{10} \approx 114_{10}$
2. $114_{10} = 0111\ 0010_{C2}$

Vérification

$0111\ 0010_{C2} = 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4} + 0 \cdot 2^{-5} + 1 \cdot 2^{-6} + 0 \cdot 2^{-7}$

$0111\ 0010_{C2} = 0,5 + 0,25 + 0,125 + 0,015625$

$0111\ 0010_{C2} = 0,890625_{10}$

Erreur de $0,893 - 0,890625 = 0,002375 < 2^{-7} = 0,0078125$

# Codage des réels

## Virgule flottante / Norme IEEE 754

Un nombre est représenté par des chiffres significatifs (ou mantisse) et par un exposant.

Avantages :

- plus grande plage de valeurs pouvant être codées ;
- débordements de capacité moins fréquents.

Inconvénient : complexité des opérateurs permettant les opérations de base

### Règles de codage IEEE 754 - simple précision (32 bits)

1 bit pour le signe S, 8 bits pour l'exposant E et 23 bits pour la mantisse F

| Bit | 31 | 30..23 | 22..0 |
|---|---|---|---|
| Dénomination | S (signe) | E (exposant) | F (mantisse) |

La valeur décimale vaut alors : $v = (-1)^S \cdot 2^{E-127} \cdot 1, F$ avec $0 < E, F < 255$

Cas spéciaux

- si $E = 255$ et $F \neq 0$, alors $v$ n'est pas un nombre (code illicite) ;
- si $E = 255$ et $F = 0$, alors $v$ vaut $\pm \inf$ (selon S) ;
- si $E = 0$ et $F \neq 0$, alors $v$ est un nombre dénormalisé dont la valeur est : $v = (-1)^S \cdot 2^{E-126} \cdot 0, F$

# Codage des réels

## Virgule flottante / Norme IEEE 754

Un nombre est représenté par des chiffres significatifs (ou mantisse) et par un exposant.

Avantages :

- plus grande plage de valeurs pouvant être codées ;
- débordements de capacité moins fréquents.

Inconvénient : complexité des opérateurs permettant les opérations de base

### Règles de codage IEEE 754 - simple précision (32 bits)

1 bit pour le signe S, 8 bits pour l'exposant E et 23 bits pour la mantisse F

| Bit | 31 | 30..23 | 22..0 |
|---|---|---|---|
| Dénomination | S (signe) | E (exposant) | F (mantisse) |

La valeur décimale vaut alors : $v = (-1)^S \cdot 2^{E-127} \cdot 1, F$ avec $0 < E, F < 255$

Cas spéciaux

- si $E = 255$ et $F \neq 0$, alors $v$ n'est pas un nombre (code illicite) ;
- si $E = 255$ et $F = 0$, alors $v$ vaut $\pm \inf$ (selon S) ;
- si $E = 0$ et $F \neq 0$, alors $v$ est un nombre dénormalisé dont la valeur est : $v = (-1)^S \cdot 2^{E-126} \cdot 0, F$

# Autres codages
## BCD

Chaque chiffre de la représentation décimale est codé en binaire naturel sur 4 bits.

*Par exemple*

| Décimal | 2 | 6 | 9 |
|---|---|---|---|
| BCD | 0010 | 0110 | 1001 |

Encore utilisé pour communiquer avec des dispositifs décimaux, comme certains afficheurs.

# Autres codages
## BCD

Chaque chiffre de la représentation décimale est codé en binaire naturel sur 4 bits.

*Par exemple*

| Décimal | 2 | 6 | 9 |
|---|---|---|---|
| BCD | 0010 | 0110 | 1001 |

Encore utilisé pour communiquer avec des dispositifs décimaux, comme certains afficheurs.

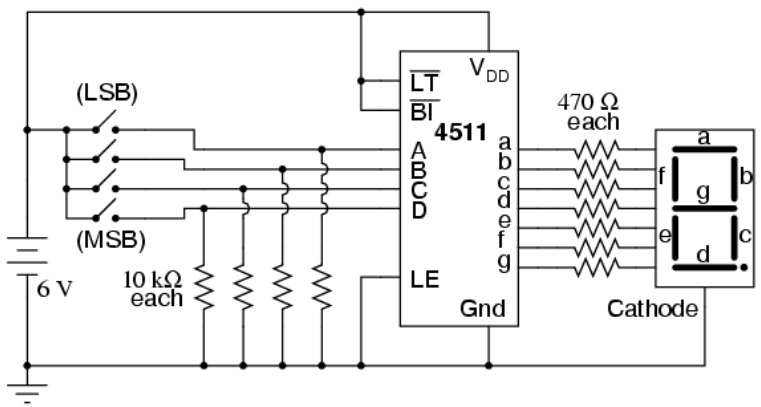
# Autres codages
## BCD

Chaque chiffre de la représentation décimale est codé en binaire naturel sur 4 bits.

*Par exemple*

| Décimal | 2 | 6 | 9 |
|---|---|---|---|
| BCD | 0010 | 0110 | 1001 |

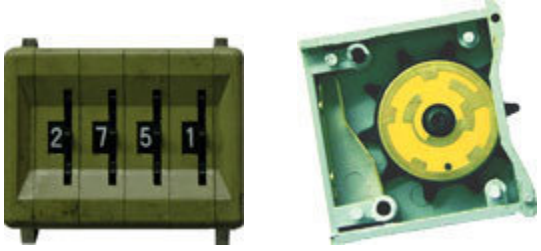
Encore utilisé pour communiquer avec des dispositifs décimaux, comme certains afficheurs.



# Autres codages

## Gray

Le code **Gray** ou *binaire réfléchi* a été initialement créé pour résoudre les problèmes liés aux codeurs de position absolue.

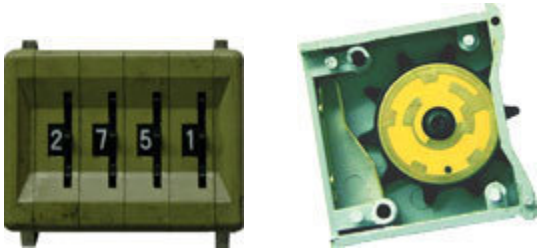



Passage d'une position à l'autre par **changement d'un seul bit**, évitant ainsi des codes intermédiaires fugitifs (dûs aux temps de réaction des capteurs, par exemple).

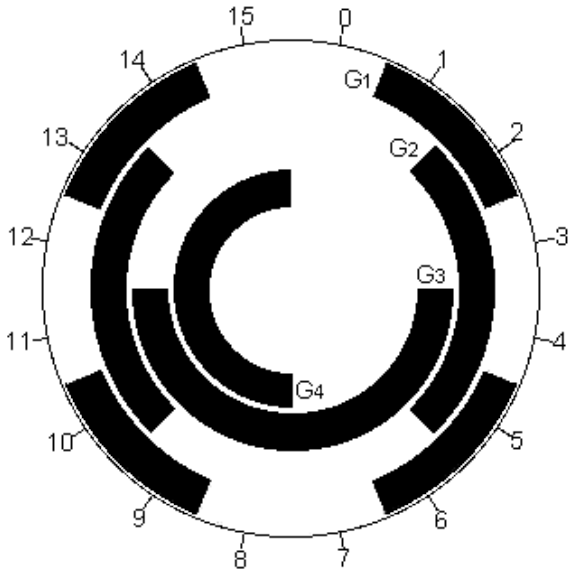
# Autres codages

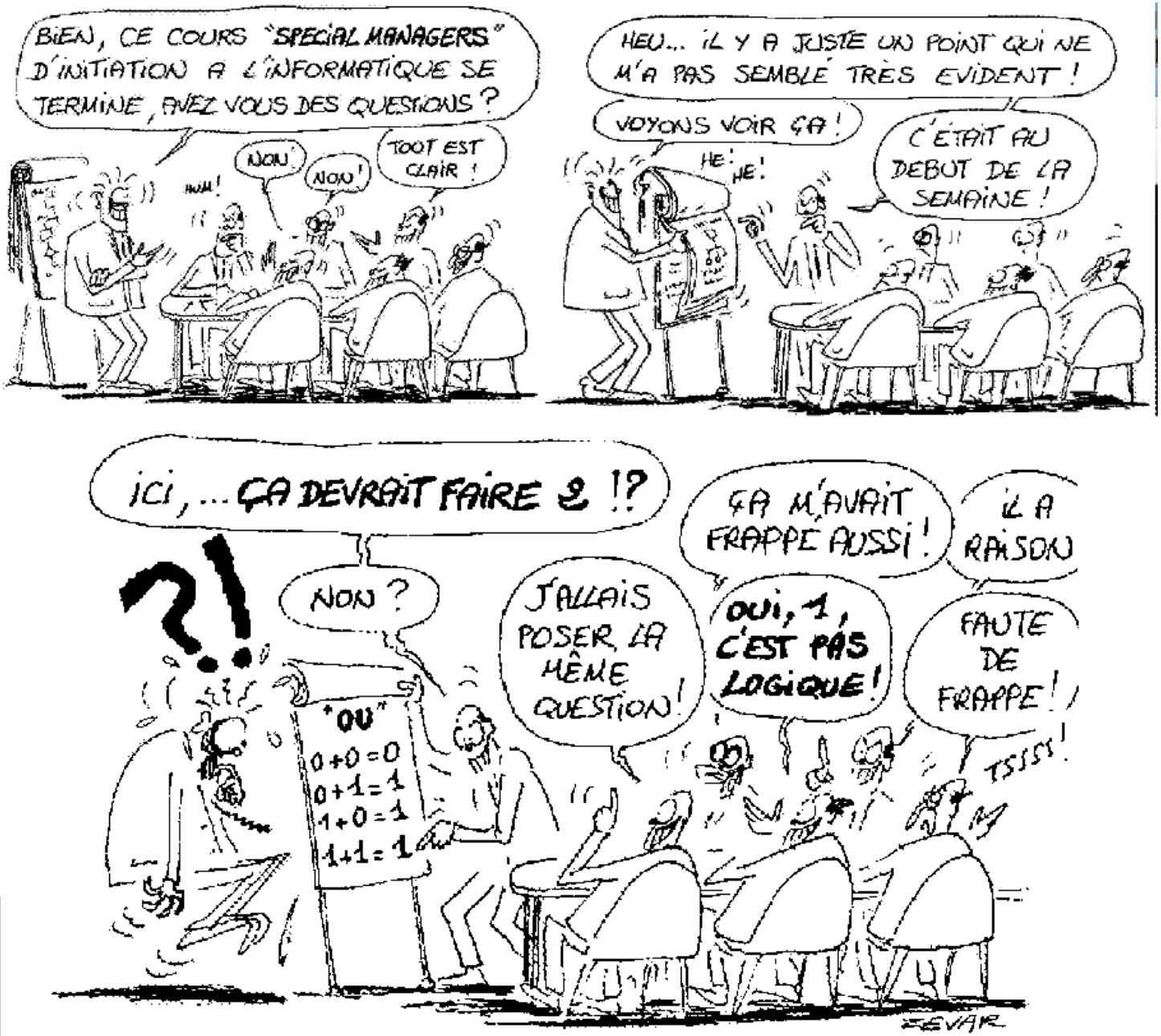
## Gray

Le code **Gray** ou *binaire réfléchi* a été initialement créé pour résoudre les problèmes liés aux codeurs de position absolue.




Passage d'une position à l'autre par **changement d'un seul bit**, évitant ainsi des codes intermédiaires fugitifs (dûs aux temps de réaction des capteurs, par exemple).




BIEN, CE COURS "SPÉCIAL MANAGERS" D'INITIATION À L'INFORMATIQUE SE TERMINE, AVEZ VOUS DES QUESTIONS ?
NON !
NON !
TOUT EST CLAIR !
HEU... IL Y A JUSTE UN POINT QUI NE M'A PAS SEMBLÉ TRÈS ÉVIDENT !
VOYONS VOIR ÇA !
HE HE !
C'ÉTAIT AU DÉBUT DE LA SEMAINE !
ICI, ... ÇA DEVRAIT FAIRE 2 !?
?!
NON ?
"OU"
0+0=0
0+1=1
1+0=1
1+1=1
J'ALLAIS POSER LA MÊME QUESTION !
ÇA M'AVAIT FRAPPÉ AUSSI !
OUI, 1, C'EST PAS LOGIQUE !
IL A RAISON
FAUTE DE FRAPPE !
TSISS !
EVAR