# 4. Codage de nombres entiers naturels

## Principes de fonctionnement des ordinateurs

Jonas Lätt

Centre Universitaire d'Informatique

UNIVERSITÉ DE GENÈVE
CENTRE UNIVERSITAIRE D'INFORMATIQUE

*Trouvé une erreur sur un transparent? Envoyez-moi un message*

- *sur Twitter @teachjl ou*
- *par e-mail jonas.latt@unige.ch*

# Contenu du cours

## Partie I: Introduction

## Partie II: Codage de l'information

## Partie III: Circuits logiques

## Partie IV: Architecture des ordinateurs

1. Introduction
2. Histoire de l'informatique
3. Information digitale et codage de l'information
4. **Codage des nombres entiers naturels**
5. Codage des nombres entiers relatifs
6. Codage des nombres réels
7. Codage de contenu média
8. Portes logiques
9. Circuits logiques combinatoires et algèbre de Boole
10. Réalisation d'un circuit combinatoire
11. Circuits combinatoires importants
12. Principes de logique séquentielle
13. Réalisation de la bascule DFF
14. Architecture de von Neumann
15. Réalisation des composants
16. Code machine et langage assembleur
17. Réalisation d'un processeur
18. Performance et micro-architecture
19. Du processeur au système

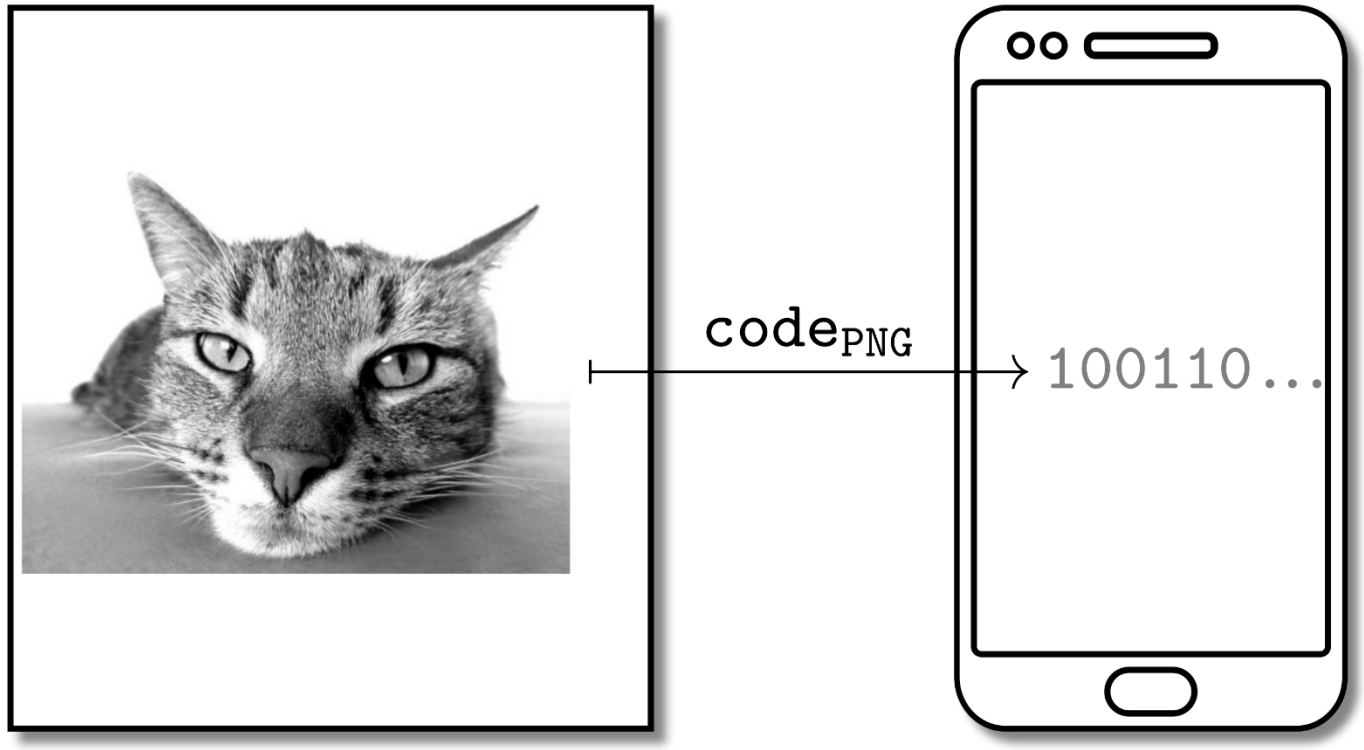
# 7. Codage de nombres entiers naturels

“Il existe 10 genres de personnes:
Celles qui comprennent le système binaire et celles qui ne le comprennent pas.”

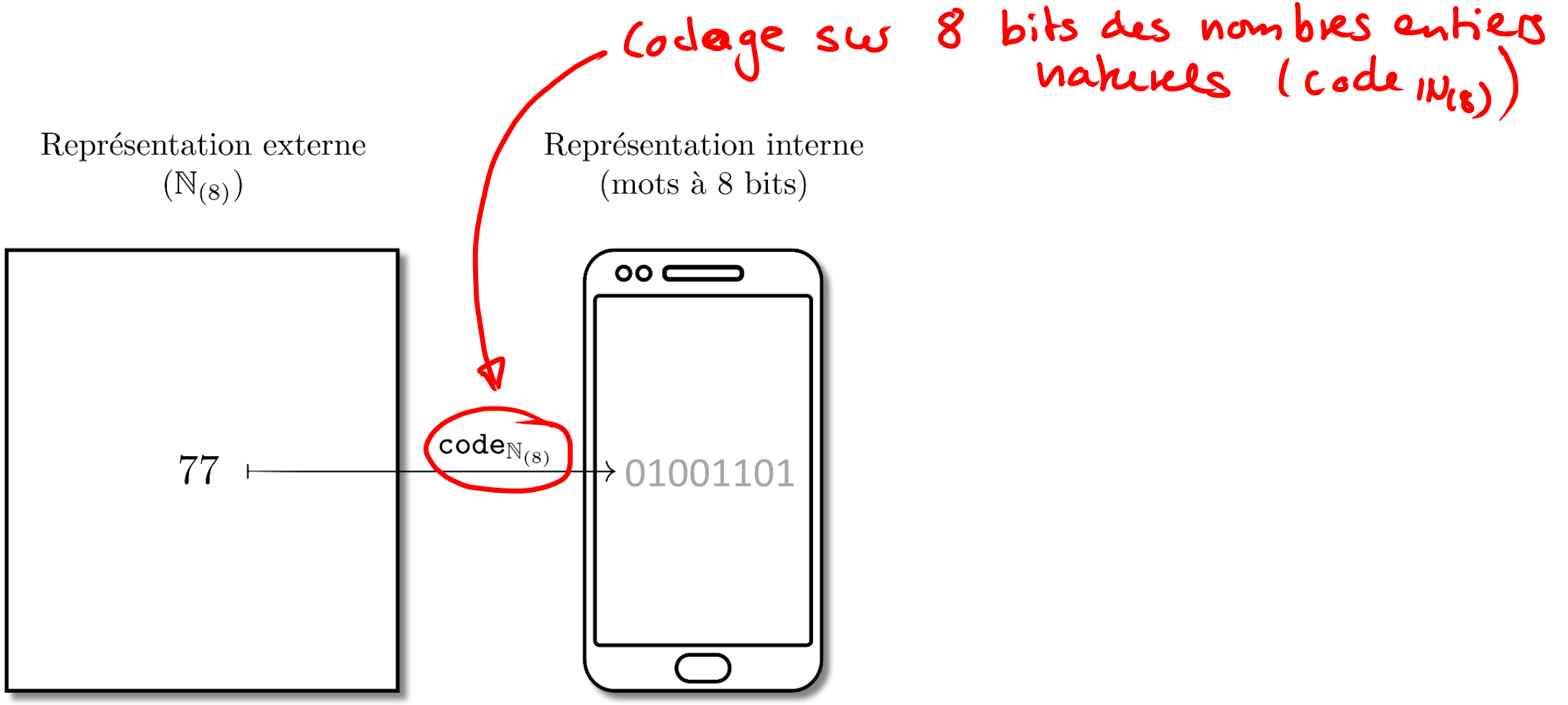
# Rappel: codage de l'information

Représentation externe
("Monde réel")

Représentation interne
(Espace des états binaires)



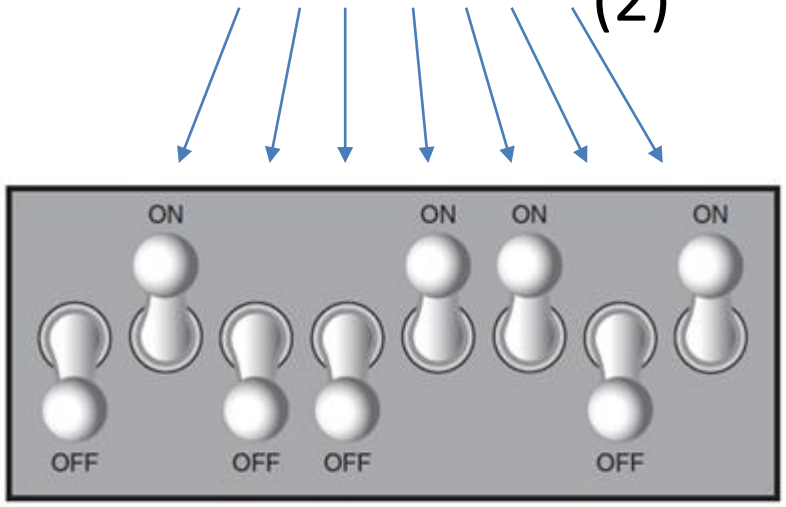
# Aujourd'hui: codage de nombres entiers



# Représentation d'entiers: idée

$77_{(10)}$

$1001101_{(2)}$

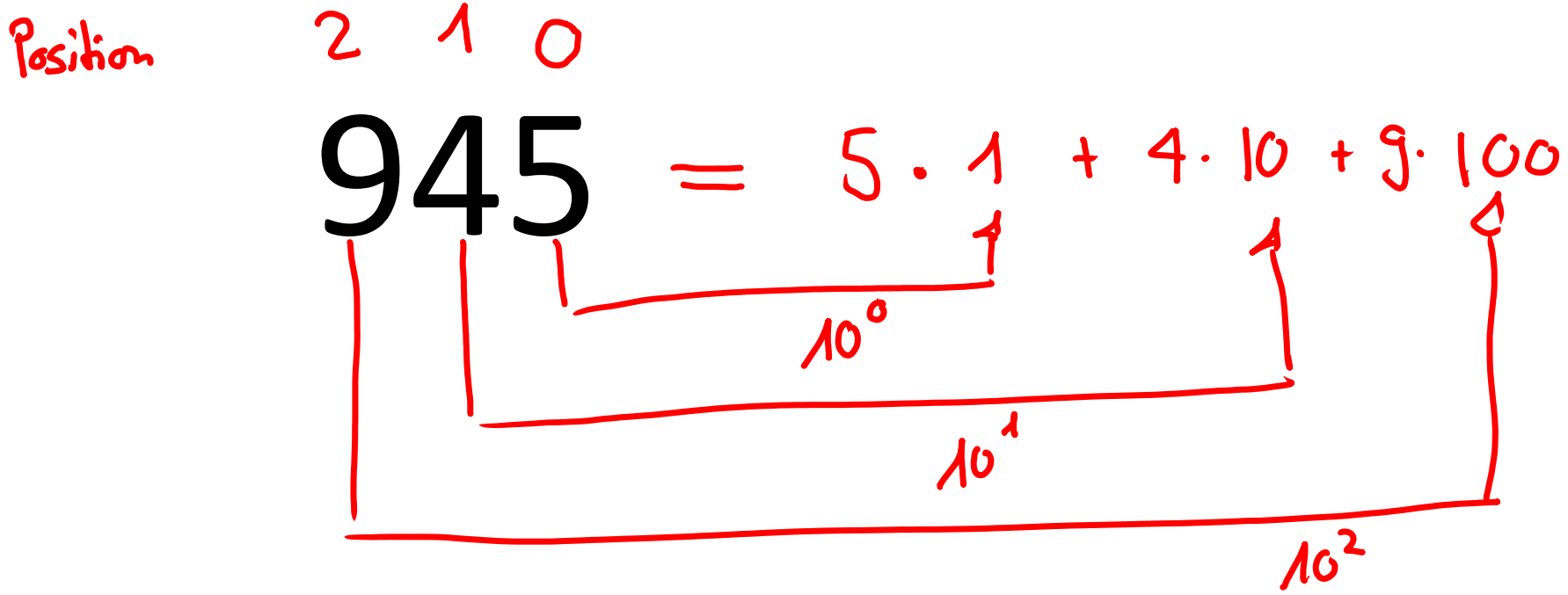



Un nombre exprimé en base décimale

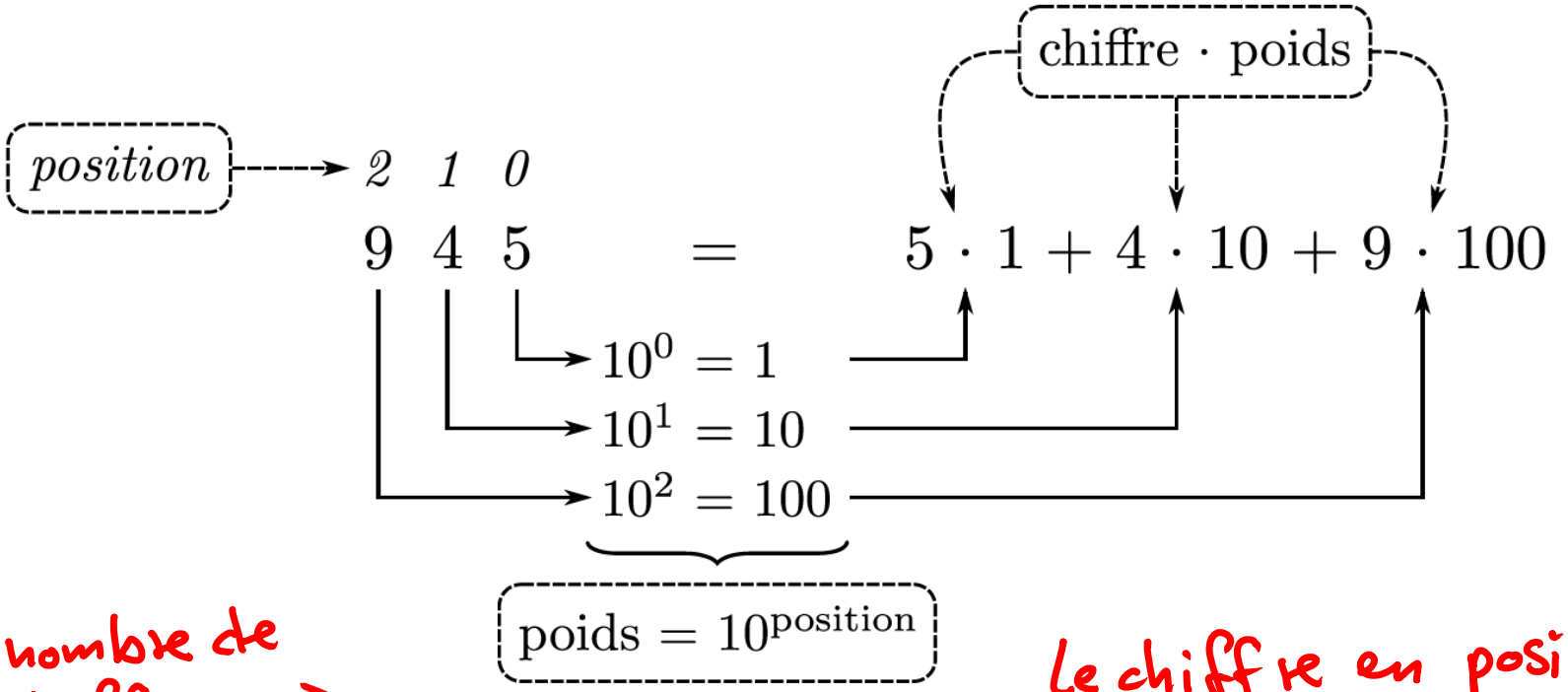
Le même nombre en base binaire

Chaque chiffre du nombre, exprimé en base binaire, est représenté par un état binaire dans l'ordinateur.

# La notation positionnelle décimale



# Notation positionnelle




N : nombre de chiffres

le chiffre en position $i$

Cas général (decimal): $x = \sum_{i=0}^{N-1} a_i \cdot 10^i$

# Cas général: n'importe quelle base

En base quelconque: $x = \sum_{i=0}^{N-1} a_i \cdot b^i$

Base: b entier $\geq 2$

Conversion d'une base quelconque vers la base décimale: appliquez simplement la definition ci-dessus.

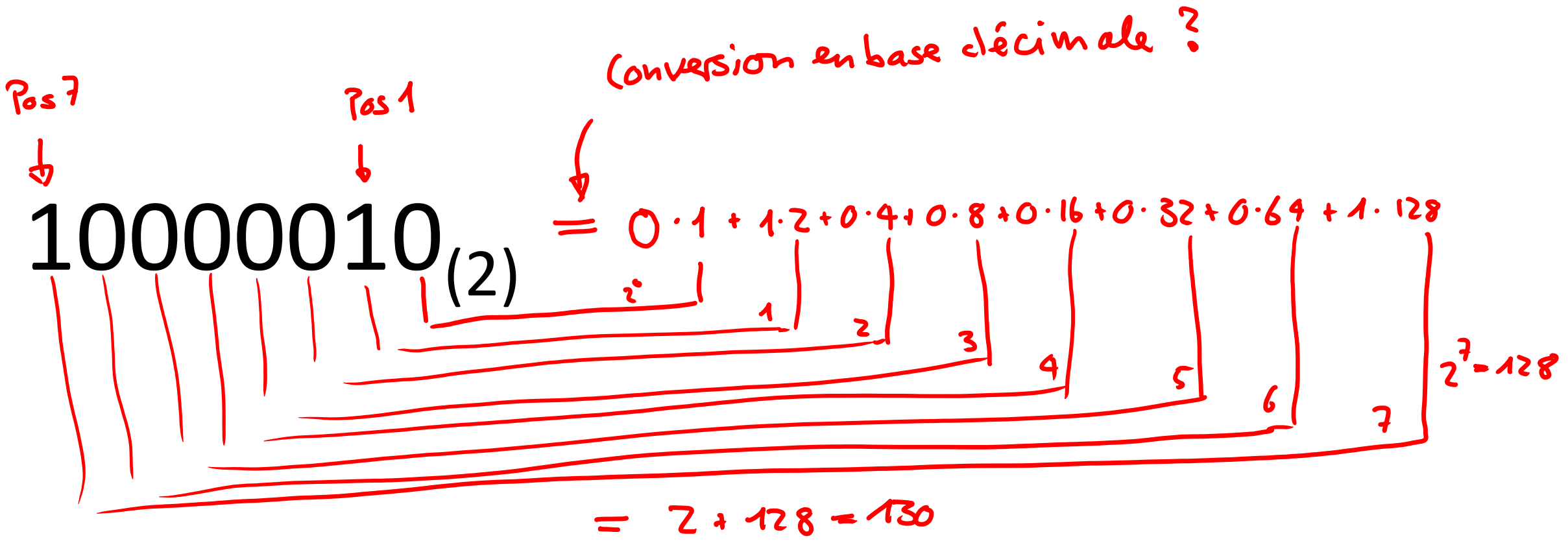
Tout en base décimale

Position 2 1 0

$523_{(6)} = 3 \cdot 1 + 2 \cdot 6 + 5 \cdot 36 = 3 + 12 + 180 = 195_{(10)}$

Poids : $6^0 = 1$ $6^1 = 6$ $6^2 = 36$

# Autre exemple: base binaire




En base 2: sommez simplement les poids des positions de chiffre 1.

# Conversion de base décimale vers base b

- On divise successivement par la base b
- Le reste de la division devient un chiffre du résultat, à écrire de droite à gauche
- Exemple: conversion de la base 10 vers la base 6.

$195_{(10)} = 523_{(6)}$

$195 / 6 = 32$ + 'un reste' 3 ; $6 \cdot 32 = 192$ ; $195 - 192 = 3$

$32 / 6 = 5$ R 2

$5 / 6 = 0$ R 5

R 3 : chiffre de droite

chiffre de gauche

# A vous de jouer!

Conversion vers la base 3:

votamatic.unige.ch

code d'accès

GRXL

$$77_{(10)} = ?_{(3)}$$ = 2212 (3)

77 / 3 = 25 R 2

25 / 3 = 8 R 1

8 / 3 = 2 R 2

2 / 3 = 0 R 2

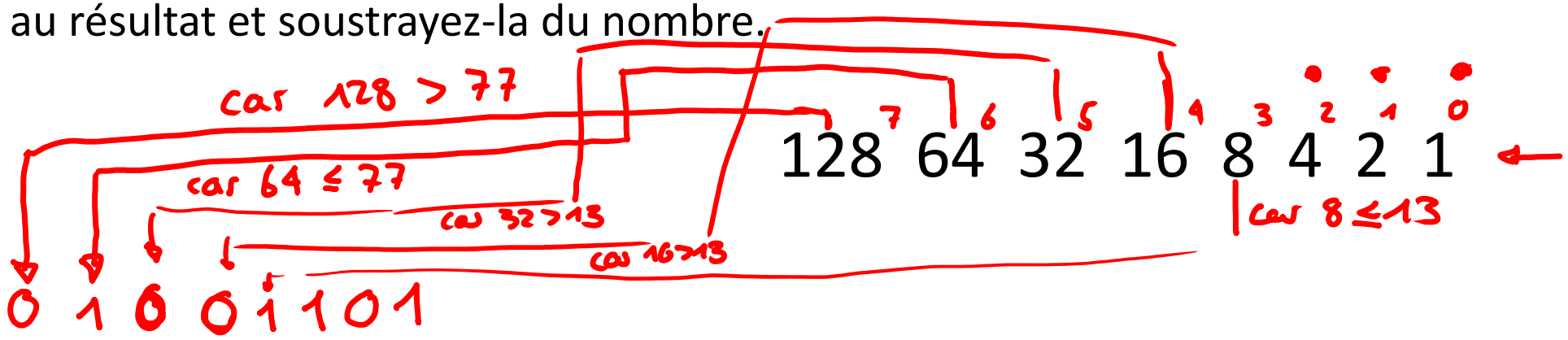
# Conversion de base 10 vers base 2

## Méthode de décomposition en puissances de 2

La méthode de conversion présentée ci-dessus fonctionne aussi pour la base 2. Mais voici une autre méthode qui est plus pratique pour une conversion manuelle.

Listez toutes les puissances de 2 de $2^0$ à $2^{k-1}$. De manière répétée

- Choisissez la plus grande puissance inférieure ou égale au nombre
- Ajoutez-la au résultat et soustrayez-la du nombre.




$77_{(10)}$ =

77 - 64 = 13
13 - 8 = 5
5 - 4 = 1

# Commentaire sur l'ordre des chiffres

En français, on écrit et on lit un nombre de gauche à droite, donc en commençant par le chiffre de poids élevé.

945

En mathématiques, on développe une série en commençant par le terme d'indice faible.

$$5 \cdot 10^0 + 4 \cdot 10^1 + 9 \cdot 10^2$$

Les deux choix sont arbitraires

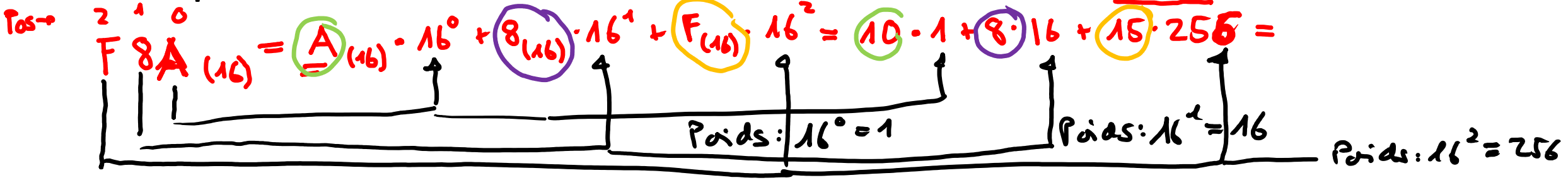
# La base hexadécimale (16)

# La base hexadécimale (16) est particulière

Les chiffres 0-9 ne suffisent pas! Il faut des chiffres pour 10-15. On utilise:

| |
|---|
| $10_{(10)} = A_{(16)}$ |
| $11_{(10)} = B_{(16)}$ |
| $12_{(10)} = C_{(16)}$ |
| $13_{(10)} = D_{(16)}$ |
| $14_{(10)} = E_{(16)}$ |
| $15_{(10)} = F_{(16)}$ |

Exemple: convertissons le nombre hexadécimal F8A en décimal:



# Conversion binaire -> hexadécimal

| Décimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Binaire** | **0000** | **0001** | **0010** | **0011** | **0100** | **0101** | **0110** | **0111** | **1000** | **1001** | **1010** | **1011** | **1100** | **1101** | **1110** | **1111** |
| **Hexa** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |

Pour convertir un nombre binaire en hexadécimal, il suffit de le traduire par groupes de quatre chiffres binaires, et vice-versa.

$\underbrace{1110}_{E}\ \underbrace{0010}_{2}{}_{(2)} = E2_{(16)}$car

$1110_{(2)} = E_{(16)}$, et

$0010_{(2)} = 2_{(16)}$

# Exercice

votamatic.unige.ch

code d'accès

MGJX

Convertissez le nombre $F37_{(16)}$ en base binaire.

$F_{(16)} = 1111_{(2)}$

$3_{(16)} = 0011_{(2)}$

$7_{(16)} = 0111_{(2)}$

$F37_{(16)} = 1111\ 0011\ 0111_{(2)}$

$0011\ 0111_{(2)} = 37_{(16)}$

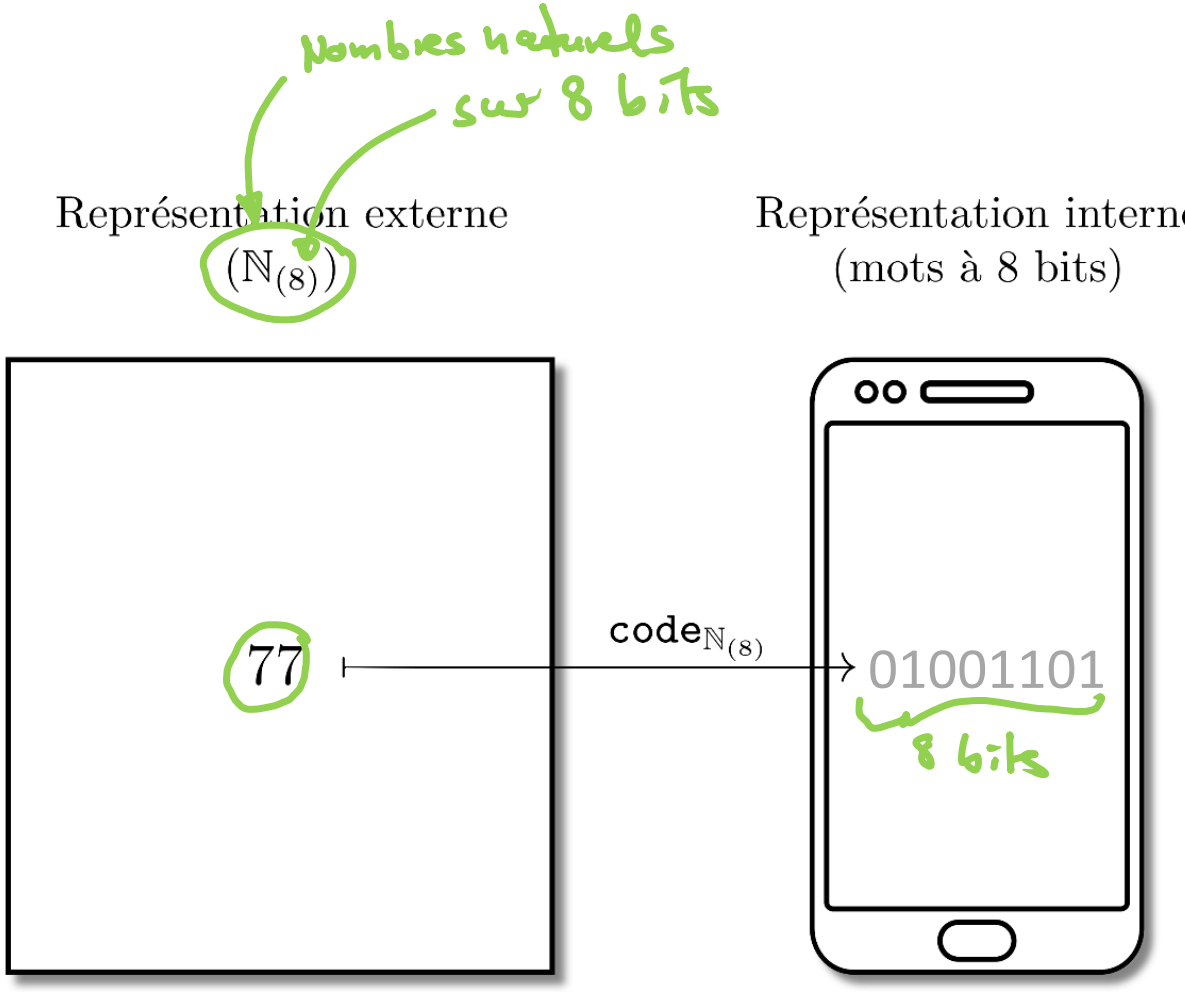
(3 7)

| Décimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Binaire | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
| Hexa | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |

# 4. Codage de nombres entiers naturels

## Codage à taille fixe pour les nombres entiers

# Codage de nombres entiers




**Idée:**

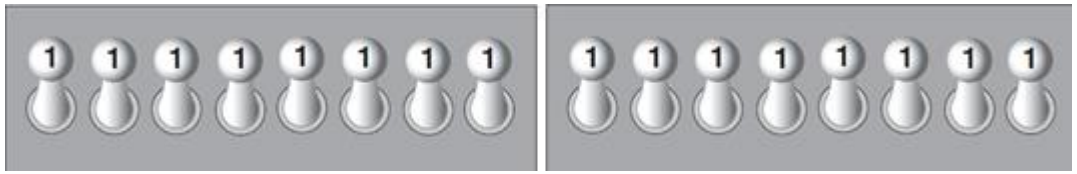
1. On fixe un nombre de bits (ici: 8)
2. On convertit l'entier naturel en base binaire, et on assigne les chiffres 0/1 aux états binaires de la représentation interne.

**Notation:**

$\mathbb{N}_{(k)}$ ← Nbre de bits des codage

# Les mots

La plupart du temps, les ordinateurs regroupent l'information par **mots** (anglais: **word**), des séquences de bits de longueur fixe (8 bits, 16 bits, 32 bits, 64 bits, etc.)



Exemple: Un **mot** à 16 bits.

L'ordinateur applique ses opérations (p.ex. les opérations arithmétiques) à des *mots* entiers.

La fonction de codage a donc typiquement la taille d'un mot.

# Le codage $\texttt{code}_{\mathbb{N}_{(k)}}$

- Le codage $\texttt{code}_{\mathbb{N}_{(k)}}$ accepte des entiers naturels dans $\mathbb{N}_{(k)}$ et les convertit en séquences binaires.
- Principe: le nombre est exprimé en base binaire, et chaque chiffre correspond à un état.
- Le codage dépend de la taille du mot.

Exemple en $\mathbb{N}_{(16)}$:

- $0_{(10)} \xrightarrow{\texttt{code}_{\mathbb{N}_{(16)}}}$ 0000 0000 0000 0000
- $2_{(10)} \xrightarrow{\texttt{code}_{\mathbb{N}_{(16)}}}$ 0000 0000 0000 0010
- $65535_{(10)} \xrightarrow{\texttt{code}_{\mathbb{N}_{(16)}}}$ 1111 1111 1111 1111

# Plus grand nombre en *k* bits

Quel est le plus grand nombre entier naturel qu'on peut représenter à l'aide d'un *mot* à 16 bits?

- Il s'agit du nombre n = $1111\ 1111\ 1111\ 1111_{(2)}$ • •
- Il suffit de realiser que n + 1 = $1\ 0000\ 0000\ 0000\ 0000_{(2)} = 2^{16}$.
- Donc, n = $2^{16} - 1 = 65535$

$n = 2^0 + 2^1 + \dots 2^{15} = 65\,535$

autre méthode

$n+1 = 1\ 0000\ 0000\ 0000\ 0000_{(2)} = 2^{16} = 65\,536$

Position 16

$n = 65\,536 - 1 = 65\,535$

# Le sous-ensemble $\mathbb{N}_{(k)}$ des nombres naturels

- Un **mot** ne représente qu'un nombre limité de valeurs entières naturelles.
- Un **mot** de $k$ bits peut représenter un nombre $a$ compris entre 0 et $2^k$-1. Nous appelons $\mathbb{N}_{(k)}$ ce sous-ensemble des nombres naturels:

$$\mathbb{N}_{(k)} = \{x | \ 0 \leq x < 2^k\}.$$

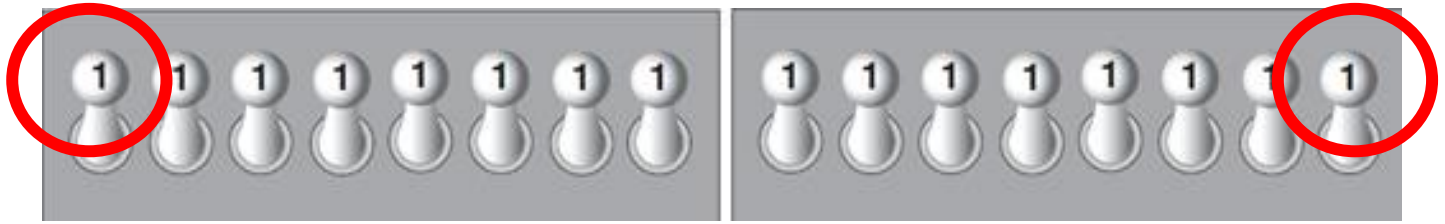
| Ensemble | N'ombre d'octets | Plus grand nombre (en base binaire) | Plus grand nombre (en base hexadécimale) | Plus gd nbre (décimal) |
|---|---|---|---|---|
| $\mathbb{N}_{(8)}$ | 1 | 1111 1111 | FF | 255 |
| $\mathbb{N}_{(16)}$ | 2 | 1111 1111 1111 1111 | FF FF | 65535 |
| $\mathbb{N}_{(32)}$ | 4 | … | FF FF FF FF | ~$4.3*10^9$ |
| $\mathbb{N}_{(64)}$ | 8 | … | FF FF FF FF FF FF FF FF | ~$1.8*10^{19}$ |

Le cours recommence à 14:15

# Notation: bits de poids fort et faible

- Bit de poids fort
- Most significant bit
- MSB

- Bit de poids faible
- Least significant bit
- LSB



# Notation pour les états binaires en mémoire

Pour la suite de ce cours, pour décrire le contenu d'un **mot**, on ne s'embêtera pas à écrire des longues séquences de 0 et 1 comme

```
1111 0010 0001 0000
```

On utilisera plutôt une notation hexadécimale plus concise:

```
0xF210
```

"ce qui suit est une séquence binaire décrite en notation hexadécimale"

veut dire

Donc:

$$61968_{(10)} \xrightarrow{\texttt{code}_{\mathbb{N}_{(16)}}} \texttt{0xF210}$$

# 4. Codage de nombres entiers naturels

## Débordements

# Débordements (*overflow*)

**Problème:** l'ensemble $\mathbb{N}$ est infini, alors que le sous-ensemble $\mathbb{N}_{(k)}$ est de taille limitée. On ne peut pas représenter tous les nombres!

$\mathbb{N}_{(8)} = \{0, 1, 2, \ldots, 255\}$

**Exemple:** En $\mathbb{N}_{(8)}$, donc en utilisant des mots à 8 bits, effectuons l'addition

$$130 + 170 = 300$$

Résultat plus grand que 255 !

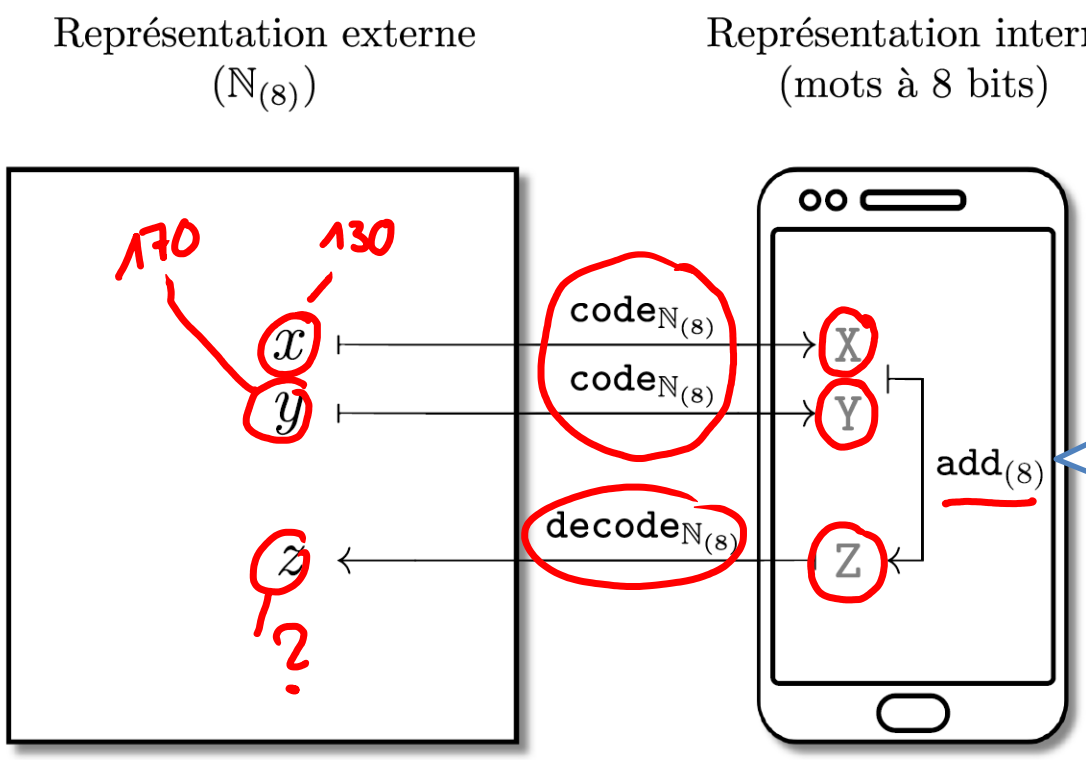
**Question:** que faire?

**Réponse:** il n'y a rien à faire. Mais, on aimerait un comportement bien défini, reproduisible.

**Gestion des débordements**

# Exemple: débordement lors d'additions

## Le circuit additionneur add(k)




La fonction $\text{add}_{(8)}$ est une **fonction logique**. Elle travaille en représentation interne: c'est un opérateur qui applique une séquence de bits sur une autre séquence de bits.

Dans un ordinateurs, les fonctions logiques sont réalisées à l'aide de **circuits logiques** (voir chapitre 9).

# Débordements

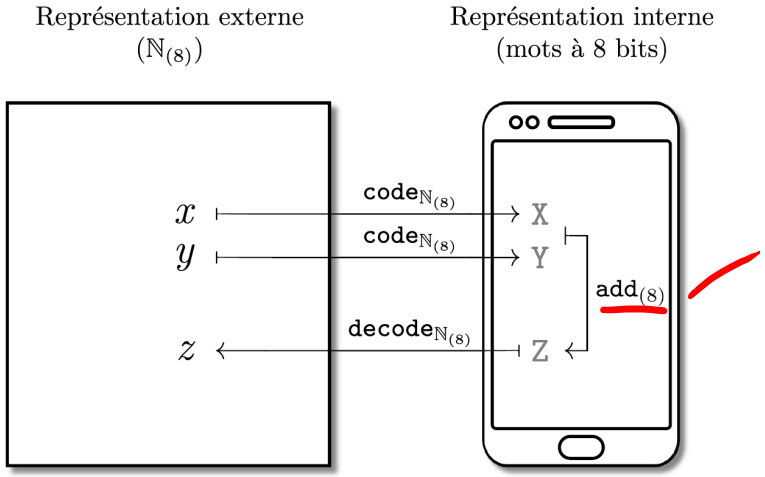
En base binaire, il est évident que le résultat est trop grand, car on «déborde à gauche»:

| | | |
|---|---|---|
| | 130 | 1 0 0 0 0 0 1 0 |
| + | 170 | + 1 0 1 0 1 0 1 0 |
| = | 300 | = 1 0 0 1 0 1 1 0 0 |

Comportement adopté: on ignore la retenue qui déborde

Le bit ignoré correspondait à une valeur de 256 (ou $2^k$). On soustrait donc 256 (ou $2^k$) du résultat.

# Débordements




Rappel: $\text{add}_{(8)}$ est une fonction logique sur 8 bits: Elle lit des mots à 8 bits et elle produit des mots à 8 bits

$x = 130$
$y = 170$

$\xrightarrow{\text{code}_{\mathbb{N}_{(8)}}}$

X: 0x82
Y: 0xAA

$\xrightarrow{\text{add}_{(8)}}$

Z: 0x2C

$\xrightarrow{\text{decode}_{\mathbb{N}_{(8)}}}$

44

# Débordements en $\mathbb{N}_{(k)}$: Représentation externe

Lors d'une addition ou soustraction de deux nombres en $\mathbb{N}_{(k)}$, un **débordement** désigne une situation dans laquelle le résultat ne se trouve pas en $\mathbb{N}_{(k)}$. En représentation externe, les règles de débordement s'énoncent par:

$$\text{Addition en } \mathbb{N}_{(k)}: \quad x, y \longmapsto \begin{cases} x+y & \text{si } x+y < 2^k \\ x+y-2^k & \text{si } x+y \geq 2^k \end{cases}$$

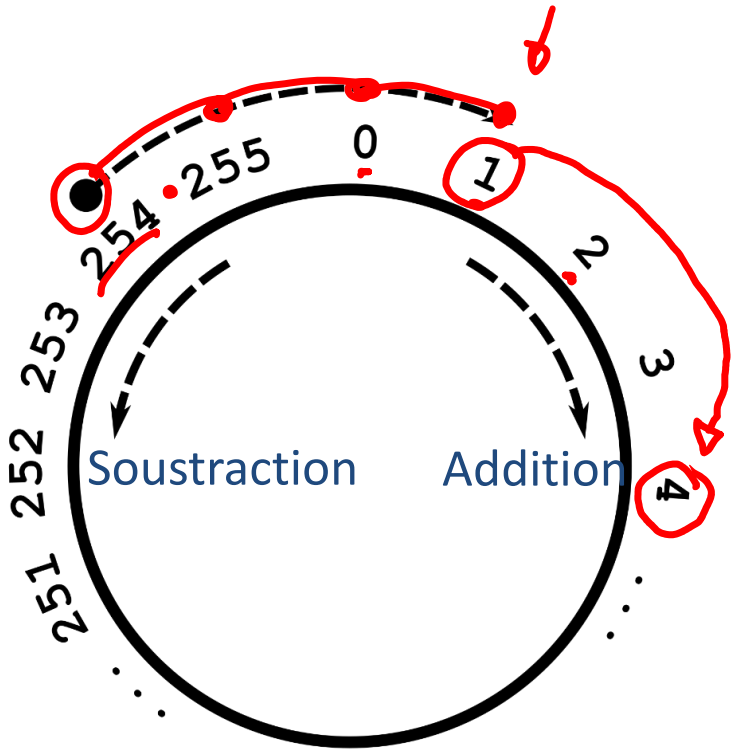
$$\text{Soustraction en } \mathbb{N}_{(k)}: \quad x, y \longmapsto \begin{cases} x-y & \text{si } x-y \geq 0 \\ x-y+2^k & \text{si } x-y < 0 \end{cases}$$

# Exemple: débordement dans un octet (k=8):

| | Calcul | Résultat z |
|---|---|---|
| **Addition sans débordement** | $x = 20$<br>$y = 30$<br>$z = \texttt{decode}_{\mathbb{N}_{(8)}}(\texttt{add}_{(8)}(\texttt{X}, \texttt{Y}))$ | 20 + 30 = 50 |
| **Addition avec débordement** | $x = 150$<br>$y = 200$<br>$z = \texttt{decode}_{\mathbb{N}_{(8)}}(\texttt{add}_{(8)}(\texttt{X}, \texttt{Y}))$ | 150 + 200 − 256 = 94 |
| **Soustraction avec débordement** | $x = 150$<br>$y = 200$<br>$z = \texttt{decode}_{\mathbb{N}_{(8)}}(\texttt{sub}_{(8)}(\texttt{X}, \texttt{Y}))$ | 150 − 200 + 256 = 206 |

# Interprétation: gestion cyclique des débordements

- La règle de débordement équivaut à un traitement *cyclique* l'intervalle $\mathbb{N}_{(k)}$.
- Si on dépasse la valeur maximale, on recommence à zéro, et vice-versa.




1 + (3) = 4
↳ une rotation de 3 positions

254 + 3 = 257
↓ Règles de débordement
Z = 254 + 3 − 256 = 1