

# Architecture des ordinateurs

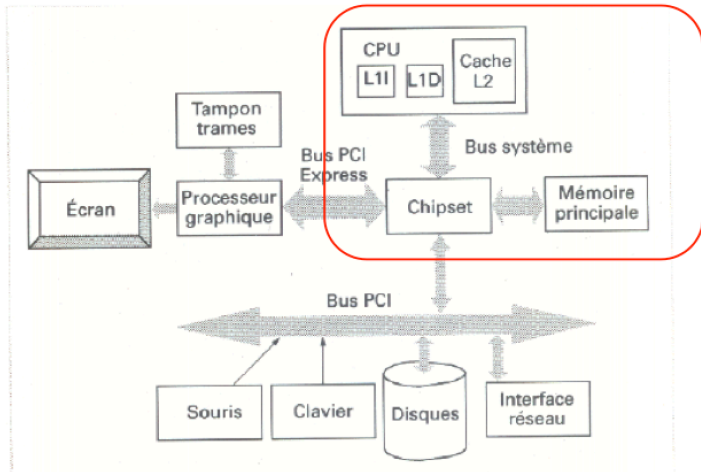
## Cours 1

Principe de fonctionnement d'un ordinateur  
Représentation des entiers naturels

# Plan du cours 1

- 1 Architecture générale d'un ordinateur
- 2 Logique booléenne et algèbre de Boole
- 3 Introduction à la représentation en machine
- 4 Représentation des entiers naturels  $\mathbb{N}$
- 5 Changement de base

# Architecture générale d'un ordinateur (type PC)



# Vue abstraite d'un ordinateur (type PC)



Un ordinateur, dont l'architecture abstraite est représentée ci-dessus, comprend

- un processeur (ou CPU)
- une mémoire centrale (ou RAM ou mémoire vive)
- un bus
- des périphériques

# Architecture générale d'un ordinateur



- Le **processeur** (ou CPU) est l'unité de traitement de l'information (instructions et données). Il exécute des programmes (suite d'instructions qui définissent un traitement à appliquer à des données).
- La **mémoire centrale** (ou RAM ou mémoire vive) est une unité de stockage temporaire des informations nécessaires à l'exécution d'un programme. Externe au processeur, elle stocke en particulier les instructions du programme en cours d'exécution ou à exécuter et les données du programme (nombre, caractères alphanumériques, adresses mémoire, ...).
- Le **bus** est le support physique des transferts d'information entre les différentes unités.
- Les **périphériques** sont des unités connexes permettant de communiquer avec l'ensemble processeur-mémoire : clavier, écran, disque dur, réseau, imprimante/scanner, ...

# Représentation d'un programme : instructions et données

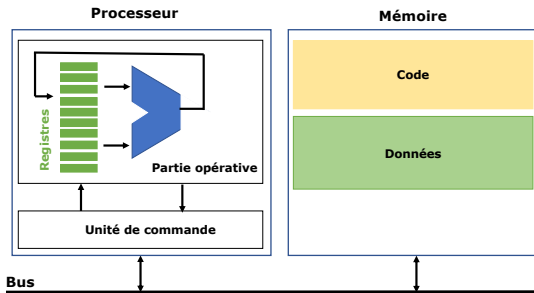
- Un programme définit un traitement à appliquer à des données
- Deux parties :
  - les données
  - le traitement qui est une suite d'opérations sur les données

## Représentation en machine

- Les données sont représentées en binaire avec un codage associé à leur nature (entiers relatifs, caractères,...) et stockées en mémoire
- Le traitement à réaliser est traduit en une suite d'instructions compréhensibles par le processeur cible : ces instructions sont dites en langage machine
- Elles sont codées en binaire et stockées en mémoire

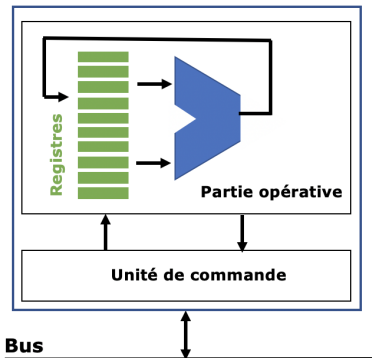
# Stockage des informations

- Dans la mémoire sont stockées les données et les instructions du programme en cours d'exécution
- Dans le processeur, toute donnée (ou information) est stockée dans un **registre** : instruction en cours d'exécution + des données ou valeurs temporaires
- Transfert des informations entre la mémoire et le processeur via le bus



# Architecture d'un processeur séquentiel

- **Unité de commande** : récupère les instructions dans la mémoire, les analyse puis séquence les actions élémentaires pour leur réalisation.
- **Partie opérative** : au service de l'unité de commande, contient les outils pour réaliser les actions élémentaires ordonnées par l'unité de commande. Comprend notamment une **unité arithmétique et logique (ALU)** et des **registres** (éléments de mémorisation temporaire).





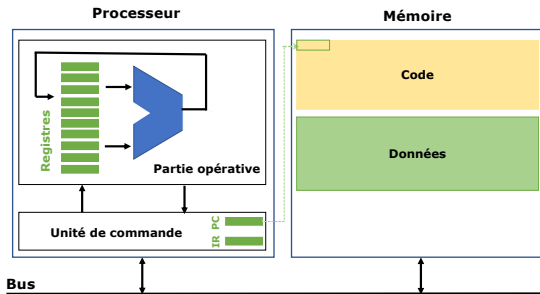
# Boucle d'exécution réalisée par processeur

Deux registres particuliers dans le processeur

- PC = Program Counteur
- IR = Instruction Register

Le processeur exécute sans fin la suite des opérations suivantes

- 1 Lire une instruction en mémoire (mise dans IR)



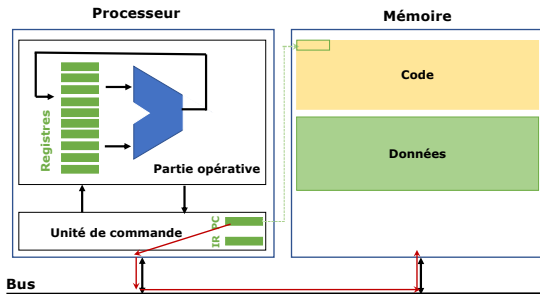
# Boucle d'exécution réalisée par processeur

Deux registres particuliers dans le processeur

- PC = Program Counteur
- IR = Instruction Register

Le processeur exécute sans fin la suite des opérations suivantes

- 1 Lire une instruction en mémoire (mise dans IR)



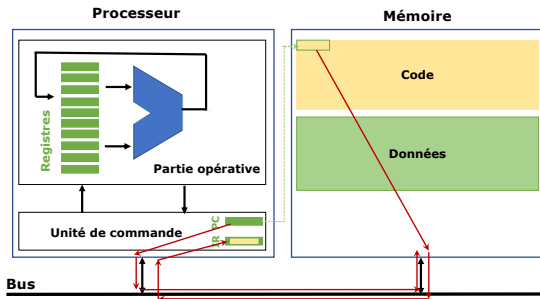
# Boucle d'exécution réalisée par processeur

Deux registres particuliers dans le processeur

- PC = Program Counteur
- IR = Instruction Register

Le processeur exécute sans fin la suite des opérations suivantes

- 1 Lire une instruction en mémoire (mise dans IR)



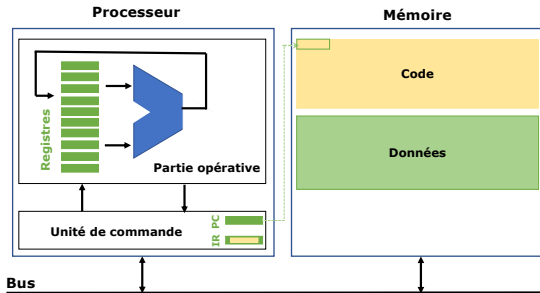
# Boucle d'exécution réalisée par processeur

Deux registres particuliers dans le processeur

- PC = Program Counteur
- IR = Instruction Register

Le processeur exécute sans fin la suite des opérations suivantes

- 1 Lire une instruction en mémoire (mise dans IR)
- 2 Décoder l'instruction



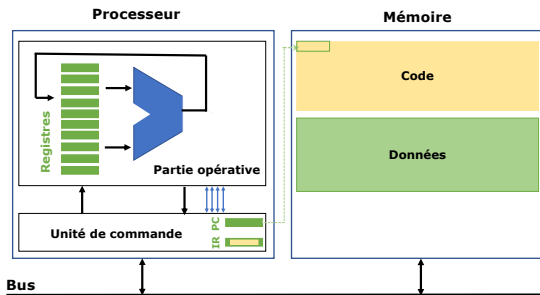
# Boucle d'exécution réalisée par processeur

Deux registres particuliers dans le processeur

- PC = Program Counteur
- IR = Instruction Register

Le processeur exécute sans fin la suite des opérations suivantes

- 1 Lire une instruction en mémoire (mise dans IR)
- 2 Décoder l'instruction
- 3 Exécuter l'instruction



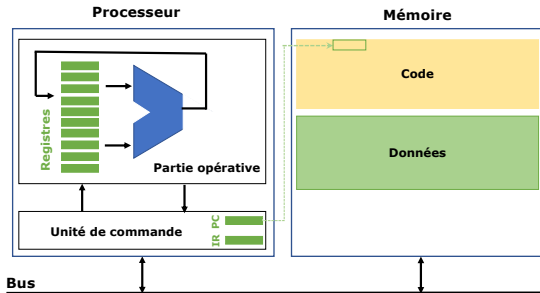
# Boucle d'exécution réalisée par processeur

Deux registres particuliers dans le processeur

- PC = Program Counteur
- IR = Instruction Register

Le processeur exécute sans fin la suite des opérations suivantes

- 1 Lire une instruction en mémoire (mise dans IR)
- 2 Décoder l'instruction
- 3 Exécuter l'instruction
- 4 Calculer l'adresse de l'instruction suivante (mise à jour PC)



# Réalisation physique des éléments d'un ordinateur

- La réalisation physique des éléments d'un ordinateur est électronique (mécanique pour certains périphériques).
- Les grandeurs manipulées au sein des composants sont des tensions électriques
  - qui sont stockées dans des éléments mémorisants, ou
  - qui transitent sur des fils et traversent des portes combinant les tensions et réalisant des fonctions logiques.
- On distingue deux niveaux de tension (par exemple 0V et 1.5V) qui représentent deux valeurs distinctes nommées 0 et 1.
- C'est pourquoi toutes les informations manipulées sont représentées par des mots composés de 0 et de 1.

# Des 0 et des 1... représentation binaire

...Toutes les informations manipulées sont représentées par des mots composés de 0 et de 1.

- **mot binaire** = un mot formé sur l'alphabet  $\{0,1\}$
- **bit** = 0 ou 1
- **octet** = mot binaire composé de 8 bits
- **quartet** = mot binaire composé de 4 bits
- **mot MIPS** = mot de 32 bits = un mot de 4 octets

Soit  $M$  un mot binaire de  $n$  bits,  $M$  est décrit par la suite de bits  $m_{n-1}m_{n-2}\dots m_0$ , chaque bit  $m_i$  est identifié par sa position  $i$  dans le mot, de 0 (bit de poids faible) à  $n - 1$  (bit de poids fort).

L'information représentée par un mot binaire dépend de son interprétation : instruction, donnée (entiers, chaîne de caractères, entête descripteur de fichier, adresse mémoire, ...)

Les traitements réalisés par l'ordinateur sont faits sur la représentation binaire des informations, en calculant des fonctions logiques sur des mots binaires.



# Logique booléenne

- **Logique booléenne** : formalisation des raisonnements basés sur des éléments qui peuvent être soit vrais, soit faux.
- Soit  $\mathcal{B}$  l'alphabet  $\mathcal{B} = \{\text{FAUX}, \text{VRAI}\} = \{F, V\} = \{0, 1\}$ .
- Ordre sur les éléments de  $\mathcal{B}$  :  $0 < 1$
- **Variable booléenne** : une variable pouvant contenir soit vrai, soit faux.
- **Fonction booléenne** : une fonction de  $\mathcal{B}^n \rightarrow \mathcal{B}$
- **Table de vérité** : énumération ligne à ligne des valeurs prises par une fonction  $f$  en fonction de la valeur de ses paramètres

# Opérations logiques

- Le **complément** / NON / NOT et noté par le surlignage est une fonction unaire. Si  $a = 0$  alors  $\bar{a} = 1$  et si  $a = 1$  alors  $\bar{a} = 0$
- L'**addition** / OU / OR et notée  $+$  est une fonction binaire et définie par  $a + b = \max(a, b)$
- La **multiplication** / ET / AND et notée  $.$  est une fonction binaire et définie par  $a.b = \min(a, b)$ .
- Leur table de vérité :

a	$\bar{a}$
0	1
1	0

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

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

Toute fonction Booléenne peut être décrite par composition des fonctions élémentaires ET, OU, NON, car  $\langle \mathcal{B}, 0, 1, +, ., - \rangle$  forme une algèbre (cf. séances 8 et suivantes).

# Extension des opérateurs booléens sur des mots binaires

Soient  $A = a_{n-1}..a_0$  et  $B = b_{n-1}..b_0$  deux mots binaires de  $n$  bits,

On définit les opérations NON, ET, OU sur les mots de  $n$  bits :

- $C = \bar{A}$  le mot binaire sur  $n$  bits tel que :  $C = c_{n-1}..c_0$  et  $\forall i \in [0, n-1] : c_i = \bar{a}_i$
- $D = A.B$  le mot binaire sur  $n$  bits tel que :  $D = d_{n-1}..d_0$  et  $\forall i \in [0, n-1] : d_i = a_i.b_i$
- $E = A + B$  le mot binaire sur  $n$  bits tel que :  $E = e_{n-1}..e_0$  et  $\forall i \in [0, n-1] : e_i = a_i + b_i$

Et par extension, toute fonction Booléenne  $f$  de  $\mathcal{B}^p$  vers  $\mathcal{B}$  peut être étendue sur des mots de  $n$  bits en appliquant  $f$  sur les  $p$  bits opérands de même position.

## Exemples

Soient  $A = 0111$  et  $B = 1010$

- $\bar{A} = 1000$  et  $\bar{B} = 0101$
- $A.B = 0010$
- $A + B = 1111$

# Opérations de décalage logique sur des mots binaires

Soient  $A = a_{n-1}..a_0$  un mot binaire de  $n$  bits,

On définit les opérations de décalage logique de  $p$  bits :

- $B = A \ll p$  le mot binaire sur  $n$  bits tel que :  $B = b_{n-1}..b_0$  et  $\forall i \in [0, p-1] : b_i = 0$  et  $\forall i \in [p, n-1] : b_i = a_{i-p}$
- $C = A \gg p$  le mot binaire sur  $n$  bits tel que :  $C = c_{n-1}..c_0$  et  $\forall i \in [p, n-1] : c_i = 0$  et  $\forall i \in [0, p-1] : c_i = a_{i+p}$

## Exemples

Soient  $A = 0111$

- $A \ll 1 = 1110$
- $A \gg 2 = 0001$

# Représentation des informations en machine

- La représentation binaire est facile à réaliser (2 états d'équilibre) et les opérations élémentaires sont relativement simples à effectuer sous forme de circuit logique .
- Différents types d'informations (instructions, données) dans un ordinateur, mais toutes représentées sous forme binaire.
- L'information élémentaire = le bit
- Les informations (instructions, caractère, nombre, ...) sont représentées avec un ensemble de bits en utilisant un codage.

# Codage des informations

- Le codage d'une information = correspondance entre la représentation externe de l'information (caractère A ou nombre 36) et sa représentation binaire (suite de bits).
- C'est l'utilisation d'une information qui en détermine le type (décodage appliqué, lieu d'utilisation...)
- Besoin de codage pour les informations traitées par le processeur :
  - les instructions : les instructions et leur codage dépend du (type de) processeur
  - les données numériques (N, Z, etc.), alphanumériques ou plus complexes (image) : codage suivant des normes (complément à deux, ASCII, UTF-8, RGB).

# Système de numération

- Un **système de numération** fait correspondre à un nombre  $N$  un certain formalisme écrit et oral permettant de le décrire de façon unique.
- Dans un système de base unaire, le nombre  $N$  est représenté par autant de symbole unaire que la valeur qu'il représente ( $N=5$  est représenté par ●●●●●).
- Dans un système à base  $B > 1$ , les symboles unaires (●) sont regroupés en paquets de  $B$  éléments, eux même regroupés en paquets de  $B$  éléments (donc des paquets de  $B^2$  éléments), et ainsi de suite. les symboles  $0, 1, 2, \dots, B - 1$  sont appelés **chiffres**.

## Expression d'un entier $N$ dans une base $B$ (représentation par position pondérée)

Tout entier naturel  $N$  peut être exprimé comme une somme de multiples de puissance de la base  $B$ , les multiples étant des chiffres (donc  $< B$ )

$$N = \sum_i a_i B^i = a_n B^n + a_{n-1} B^{n-1} + \dots + a_1 B + a_0 \text{ avec } \forall i \ a_i < B$$

- La notation condensée de l'entier naturel  $N$  dans la base  $B$  est  $a_n a_{n-1} \dots a_1 a_0_B$ .
- Cette notation est pondérée : chaque position  $i$  correspond à la puissance de la base  $B^i$  et il y a un chiffre pour chaque position (éventuellement 0).

# Notations

## Notation avec indice pour les systèmes de base

- $b$  pour la base 2/le binaire :  $111_b$
- $d$  pour la base 10/le décimal :  $111_d$
- $h$  pour la base 16/l'hexadécimal :  $111_h$ ,

## Notation avec préfixe réservée aux mots binaires (voir plus loin)

- le préfixe  $0x$  pour la représentation hexadécimale d'un mot binaire :  $0x111$ ,  $0x23445$ , ...
- le préfixe  $0b$  pour la représentation d'un mot binaire :  $0b1110$



# Représentation en base 2

- Les informations manipulées par un ordinateur étant des mots binaires : les nombres entiers sont représentés en base 2 et les chiffres sont 0 et 1

## Interprétation des entiers naturels représentés en binaire

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

## Exemples

$$N_d = 111_b = (1 * 2^2 + 1 * 2^1 + 1 * 2^0)_d$$

$$N_d = 111_b = 7_d$$

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

$$N_d = (2^5 + 2^4 + 2^1 + 2^0)_d = 51_d$$

- Nombres entiers exprimés en binaire  $\Rightarrow$  grand nombre de bits
- Représentation en base 16/en hexadécimal préférée car simple conversion hexadécimal depuis/vers binaire et notation plus dense

# Représentation en hexadécimal

- En base 16, on utilise les symboles 0, 1, ..., 8, 9, A, B, C, D, E, F pour les 16 chiffres avec la correspondance suivante :

Hexadécimal	Décimal	Binaire
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111

Hexadécimal	Décimal	Binaire
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

## Interprétation d'un naturel représenté en hexadécimal

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

$$N_d = (a_{n-1}16^{n-1} + a_{n-2}16^{n-2} + \dots + a_116 + a_0)_d$$

- Il y a quatre fois moins de symboles que dans la notation en binaire.
- Attention  $1001_h \neq 1001_b$  !

# Représentation des valeurs des mots

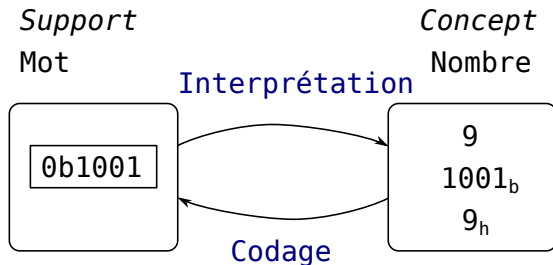
- Un mot binaire peut être interprété de **différentes façons**, pas uniquement comme un nombre non signé
- $\Rightarrow$  Pour décrire la suite de bits composant un mot binaire – par exemple le contenu d'un registre – on utilise une notation **avec préfixe** :
  - Le préfixe *0x* pour la représentation hexadécimale : *0x11*, *0x0445*, *0x1234ABCD*
  - Le préfixe *0b* pour la représentation binaire : *0b1110*, *0b00010011*
- Cette notation décrit le contenu du mot **sans l'interpréter** :
  - Ne dit pas s'il s'agit d'un nombre positif, d'un nombre négatif, d'un caractère, d'une instruction...
  - Décrit juste la valeur des bits 1 à 1, du bit de **poids faible** (le plus à droite) au bit de **poids fort** (le plus à gauche)
- Les 0 en tête sont généralement écrits pour indiquer la taille du mot
- La taille de représentation (taille du mot en bits) doit être connue : c'est normalement toujours le cas

# Codage entiers naturels

## Codage entiers naturels

- Pour encoder un entier naturel sur un mot, on utilise la représentation binaire du nombre
- Le bit de poids faible (bit de rang 0) est pondéré par  $2^0$ , le bit de poids fort (bit de rang  $N - 1$ ) par  $2^{N-1}$
- Ce codage s'appelle le codage **entiers naturels**

# Codage entiers naturels



## Exemple

- Sur 4 bits, le mot 0b1001 s'interprète comme la valeur 9 selon le codage *entiers naturels*, et  $9 = 1001_b$
- Néanmoins, ce sont **deux choses différentes** : le même mot pourrait être interprété d'une façon différente, en utilisant un autre codage
- Le nombre 3 se code 0b0011 sur un mot de 4 bits, et  $3 = 11_b$

# Représentation en machine : taille bornée

## Taille de représentation bornée

Les nombres sont représentés sur des mots de taille bornée (32 ou 64 bits)  
⇒ tous les nombres ne sont pas représentables

## Intervalle de représentation

- Sur  $p$  symboles en base  $B$ , représentation possible des entiers naturels compris dans l'intervalle  $[0, B^p - 1]$
- Sur un mot de  $n$  bits, représentation possible de l'intervalle  $[0, 2^n - 1]$  avec le codage **entiers naturels**

## Exemples

- Sur 3 chiffres en décimal, intervalle  $[0, 999]$
- Sur 3 symboles en hexadécimal, intervalle  $[0, 4095 = \text{FFF}_h]$
- Sur 8 bits, intervalle  $[0, 255 = 11111111_b]$

# Extension de la représentation d'un entier naturel

## Extension de $p$ à $n$ bits

- Soit un mot de  $p$  bits contenant une valeur  $v$ , interprété selon le codage entier naturel par le nombre  $d$
- Pour que le mot  $v'$  de  $n$  bits ( $n > p$ ) encode également le nombre  $d$ , il faut :
  - Copier les bits de poids faible de  $v$  dans  $v'$
  - Mettre les bits de poids fort restant à 0

## Preuve

Soient  $v = v_p v_{p-1} \dots v_0$  et  $v' = v'_n v'_{n-1} \dots v'_p \dots v'_0$  avec  $v'_i = 0$  pour  $i \in [p, n-1]$  et  $v'_i = v_i$  pour  $i \in [0, p-1]$ , et si  $\sum_{i=0}^{p-1} v_i * 2^i = d$  est l'interprétation de  $v$  selon le codage entiers naturels,

l'interprétation de  $v'$  dans ce codage est

$$\sum_{i=0}^{n-1} v'_i * 2^i = \sum_{i=0}^{p-1} v_i * 2^i + \sum_{i=p}^{n-1} 0 * 2^i = d.$$

## Extension de 4 à 8 bits

- 3 sur 4 bits se code 0b0011 → 3 sur 8 bits se code 0b0000 0011
- 9 sur 4 bits se code 0b1001 → 9 sur 8 bits se code 0b0000 1001

# Changement de base

Comment passer d'une représentation dans la base  $B_1$  vers une base  $B_2$  ?

- Algorithmes de conversion
  - Par divisions successives
  - Par tableau de puissance
- Correspondance simple entre certaines bases (2 vers 16, 16 vers 2)



# Algorithme de conversion par divisions successives

Conversion de la base 10 à une base  $B > 1$  pour un nombre  $N$  donné possible par divisions successives.

En base  $B$  :

- Le symbole  $a_0$  est le reste de la division euclidienne de  $N$  par  $B$  car :

$$\begin{aligned} N &= a_{n-1}B^{n-1} + \dots + a_iB^i + \dots + a_2B^2 + a_1B + a_0 \\ &= (a_{n-1}B^{n-2} + \dots + a_iB^{i-1} + a_2B^1 + a_1)B + a_0 \end{aligned}$$

- Le symbole de rang 1 ( $B^1$ ) est le reste de la division euclidienne de  $\frac{N}{B}$  par  $B$  car :

$$\begin{aligned} \frac{N}{B} &= a_{n-1}B^{n-2} + \dots + a_iB^{i-1} + \dots + a_2B + a_1 \\ &= (a_{n-1}B^{n-3} + \dots + a_iB^{i-2} + \dots + a_2)B + a_1 \end{aligned}$$

- Le symbole de rang  $i$  est le reste de la division euclidienne de  $\frac{N}{B^i}$  par  $B$  car :

$$\frac{N}{B^i} = a_{n-1}B^{n-1-i} + \dots + a_{i+1}B + a_i$$

## Algorithme

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

# Algorithme de conversion par divisions successives

Conversion de la base 10 à une base  $B > 1$  pour un nombre  $N$  donné possible par divisions successives.

En base  $B$  :

- Le symbole  $a_0$  est le reste de la division euclidienne de  $N$  par  $B$  car :

$$\begin{aligned} N &= a_{n-1}B^{n-1} + \dots + a_iB^i + \dots + a_2B^2 + a_1B + a_0 \\ &= (a_{n-1}B^{n-2} + \dots + a_iB^{i-1} + a_2B^1 + a_1)B + a_0 \end{aligned}$$

- Le symbole de rang 1 ( $B^1$ ) est le reste de la division euclidienne de  $\frac{N}{B}$  par  $B$  car :

$$\begin{aligned} \frac{N}{B} &= a_{n-1}B^{n-2} + \dots + a_iB^{i-1} + \dots + a_2B + a_1 \\ &= (a_{n-1}B^{n-3} + \dots + a_iB^{i-2} + \dots + a_2)B + a_1 \end{aligned}$$

- Le symbole de rang  $i$  est le reste de la division euclidienne de  $\frac{N}{B^i}$  par  $B$  car :

$$\frac{N}{B^i} = a_{n-1}B^{n-1-i} + \dots + a_{i+1}B + a_i$$

## Algorithme

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

# Algorithme de conversion par divisions successives

Conversion de la base 10 à une base  $B > 1$  pour un nombre  $N$  donné possible par divisions successives.

En base  $B$  :

- Le symbole  $a_0$  est le reste de la division euclidienne de  $N$  par  $B$  car :

$$\begin{aligned} N &= a_{n-1}B^{n-1} + \dots + a_iB^i + \dots + a_2B^2 + a_1B + a_0 \\ &= (a_{n-1}B^{n-2} + \dots + a_iB^{i-1} + a_2B^1 + a_1)B + a_0 \end{aligned}$$

- Le symbole de rang 1 ( $B^1$ ) est le reste de la division euclidienne de  $\frac{N}{B}$  par  $B$  car :

$$\begin{aligned} \frac{N}{B} &= a_{n-1}B^{n-2} + \dots + a_iB^{i-1} + \dots + a_2B + a_1 \\ &= (a_{n-1}B^{n-3} + \dots + a_iB^{i-2} + \dots + a_2)B + a_1 \end{aligned}$$

- Le symbole de rang  $i$  est le reste de la division euclidienne de  $\frac{N}{B^i}$  par  $B$  car :

$$\frac{N}{B^i} = a_{n-1}B^{n-1-i} + \dots + a_{i+1}B + a_i$$

## Algorithme

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

# Algorithme de conversion par divisions successives

Conversion de la base 10 à une base  $B > 1$  pour un nombre  $N$  donné possible par divisions successives.

En base  $B$  :

- Le symbole  $a_0$  est le reste de la division euclidienne de  $N$  par  $B$  car :

$$\begin{aligned} N &= a_{n-1}B^{n-1} + \dots + a_iB^i + \dots + a_2B^2 + a_1B + a_0 \\ &= (a_{n-1}B^{n-2} + \dots + a_iB^{i-1} + a_2B^1 + a_1)B + a_0 \end{aligned}$$

- Le symbole de rang 1 ( $B^1$ ) est le reste de la division euclidienne de  $\frac{N}{B}$  par  $B$  car :

$$\begin{aligned} \frac{N}{B} &= a_{n-1}B^{n-2} + \dots + a_iB^{i-1} + \dots + a_2B + a_1 \\ &= (a_{n-1}B^{n-3} + \dots + a_iB^{i-2} + \dots + a_2)B + a_1 \end{aligned}$$

- Le symbole de rang  $i$  est le reste de la division euclidienne de  $\frac{N}{B^i}$  par  $B$  car :

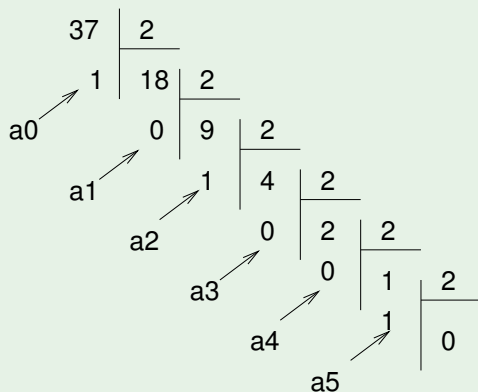
$$\frac{N}{B^i} = a_{n-1}B^{n-1-i} + \dots + a_{i+1}B + a_i$$

## Algorithme

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

# Exemple de conversion par divisions successives

Conversion de 37d en binaire



$$37d = 100101b = 2^5 + 2^2 + 2^0$$

# Conversion de la base 10 vers la base $B$ sur $(k+1)$ symboles

Conversion d'un nombre  $N$  donné dans une base  $B > 1$  sur  $(k + 1)$  symboles en cherchant les multiples des puissances de  $B$  en commençant par le symbole de poids fort.

## Algorithme

```
 $i \leftarrow k$   
 $b_{i,i \in [0,k]} \leftarrow 0$   
while  $N \geq 0$  and  $i \geq 0$  do  
  if  $N \geq \alpha \cdot B^i$  and  $N < (\alpha + 1) \cdot B^i, \alpha \in [0, B - 1]$  then  
     $b_i \leftarrow \alpha$   
     $N \leftarrow N - \alpha \cdot B^i$   
  end if  
   $i \leftarrow i - 1$   
end while  
Return  $b_{i,i \in [0,k]}$ 
```

$\Rightarrow$  réalisable à la main avec un tableau des puissances  $B^i$ , en cherchant le plus grand multiple de  $B^i$  inférieur ou égal à  $N$

# Conversion en binaire avec tableau de puissances

## Conversion de $39_d$ en binaire

	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
	$a_6$	$a_5$	$a_4$	$a_3$	$a_2$	$a_1$	$a_0$
$39 < 2^6 \implies a_6 = 0$	0	?	?	?	?	?	?
$39 > 2^5 \implies a_5 = 1$	0	1	?	?	?	?	?
$39 - 2^5 = 7 < 2^4 \implies a_4 = 0$	0	1	0	?	?	?	?
$7 < 2^3 \implies a_3 = 0$	0	1	0	0	?	?	?
$7 > 2^2 \implies a_2 = 1$	0	1	0	0	1	?	?
$7 - 2^2 = 3 > 2^1 \implies a_1 = 1$	0	1	0	0	1	1	?
$3 - 2^1 = 1 \geq 2^0 \implies a_0 = 1$	0	1	0	0	1	1	1

# Conversion de la base 2 à la base 16

## Conversion base 2 $\rightarrow$ base 16

- 1 Séparer le nombre binaire en quartet (paquet de 4 bits) en partant de la droite
- 2 Convertir chaque quartet en son symbole hexadécimal

Exemple  $N = 0b1101001010010111$

$N = 0b \quad 1101 \quad 0010 \quad 1001 \quad 0111$

Exemple  $N = 0b0011110011100101$

$N = 0b \quad 0011 \quad 1100 \quad 1110 \quad 0101$

—



# Conversion de la base 2 à la base 16

## Conversion base 2 → base 16

- 1 Séparer le nombre binaire en quartet (paquet de 4 bits) en partant de la droite
- 2 Convertir chaque quartet en son symbole hexadécimal

### Exemple $N = 0b1101001010010111$

$$\begin{array}{rcllcl} N & = & 0b & 1101 & 0010 & 1001 & 0111 \\ & = & 0x & D & 2 & 9 & 7 \end{array}$$

$$\rightarrow N = 0xB297$$

### Exemple $N = 0b0011110011100101$

$$\begin{array}{rcllcl} N & = & 0b & 0011 & 1100 & 1110 & 0101 \\ & = & 0x & 3 & C & E & 5 \end{array}$$

$$\rightarrow N = 0x3CE5$$

# Conversion de la base 16 à la base 2

## Conversion base 16 $\rightarrow$ base 2

- 1 Convertir chaque symbole hexadécimal en quartet/mot de 4 bits

Exemple  $N = 0xAFDB$

$N = 0x \quad A \quad F \quad D \quad B$

Exemple  $N = 0b854F$

$N = 0x \quad 8 \quad 5 \quad 4 \quad F$

# Conversion de la base 16 à la base 2

## Conversion base 16 $\rightarrow$ base 2

- 1 Convertir chaque symbole hexadécimal en quartet/mot de 4 bits

### Exemple $N = 0xAFDB$

$$\begin{array}{rcllclcl} N & = & 0x & A & F & D & B \\ & = & 0b & 1010 & 1111 & 1101 & 1011 \end{array}$$

$$\rightarrow N = 0b1010111111011011$$

### Exemple $N = 0b854F$

$$\begin{array}{rcllclcl} N & = & 0x & 8 & 5 & 4 & F \\ & = & 0b & 1000 & 0101 & 0100 & 1111 \end{array}$$

$$\rightarrow N = 0b1000010101001111$$

# Conclusion

On a vu

- Architecture générale d'un ordinateur
- Quelques éléments de Logique booléenne
- Représentation en machine, représentation des entiers naturels et algorithmes de conversion de base de représentation

Important :

- Moodle 2025 :  
<https://moodle-sciences-25.sorbonne-universite.fr/>

## Conversion de $N = 687_d$ en hexadécimal avec un tableau de puissances (exemple supplémentaire)

- On suppose que  $k$  vaut 3 et on rappelle que  $16^3 = 4096$  et  $16^2 = 256$ .

$16^3$	$16^2$	$16^1$	$16^0$
$a_3$	$a_2$	$a_1$	$a_0$
?	?	?	?

- $N = 687_d$
- $687 < 16^3$   
 $\rightarrow a_3 = 0$

# Conversion de $N = 687_d$ en hexadécimal avec un tableau de puissances (exemple supplémentaire)

- On suppose que  $k$  vaut 3 et on rappelle que  $16^3 = 4096$  et  $16^2 = 256$ .

$16^3$	$16^2$	$16^1$	$16^0$
$a_3$	$a_2$	$a_1$	$a_0$
?	?	?	?

$16^3$	$16^2$	$16^1$	$16^0$
$a_3$	$a_2$	$a_1$	$a_0$
0	?	?	?

- $N = 687_d$
- $687 < 16^3$   
 $\rightarrow a_3 = 0$
- $N = 687_d$
- $2 \times 16^2 \geq 687 < 3 \times 16^2$   
 $\rightarrow a_2 = 2$

# Conversion de $N = 687_d$ en hexadécimal avec un tableau de puissances (exemple supplémentaire)

- On suppose que  $k$  vaut 3 et on rappelle que  $16^3 = 4096$  et  $16^2 = 256$ .

$16^3$	$16^2$	$16^1$	$16^0$
$a_3$	$a_2$	$a_1$	$a_0$
?	?	?	?

$16^3$	$16^2$	$16^1$	$16^0$
$a_3$	$a_2$	$a_1$	$a_0$
0	?	?	?

$16^3$	$16^2$	$16^1$	$16^0$
$a_3$	$a_2$	$a_1$	$a_0$
0	2	?	?

- $N = 687_d$
- $687 < 16^3$   
 $\rightarrow a_3 = 0$
- $N = 687_d$
- $2 \times 16^2 \geq 687 < 3 \times 16^2$   
 $\rightarrow a_2 = 2$
- $N = 687 - 2 \times 16^2 = 175_d$
- $10 \times 16 < 175 < 11 \times 16$   
 $\rightarrow a_1 = A$

# Conversion de $N = 687_d$ en hexadécimal avec un tableau de puissances (exemple supplémentaire)

- On suppose que  $k$  vaut 3 et on rappelle que  $16^3 = 4096$  et  $16^2 = 256$ .

$16^3$	$16^2$	$16^1$	$16^0$
$a_3$	$a_2$	$a_1$	$a_0$
?	?	?	?

$16^3$	$16^2$	$16^1$	$16^0$
$a_3$	$a_2$	$a_1$	$a_0$
0	?	?	?

$16^3$	$16^2$	$16^1$	$16^0$
$a_3$	$a_2$	$a_1$	$a_0$
0	2	?	?

$16^3$	$16^2$	$16^1$	$16^0$
$a_3$	$a_2$	$a_1$	$a_0$
0	2	A	?

- $N = 687_d$
- $687 < 16^3$   
 $\rightarrow a_3 = 0$
- $N = 687_d$
- $2 \times 16^2 \geq 687 < 3 \times 16^2$   
 $\rightarrow a_2 = 2$
- $N = 687 - 2 \times 16^2 = 175_d$
- $10 \times 16 < 175 < 11 \times 16$   
 $\rightarrow a_1 = A$
- $N = 175 - 10 \times 16^1 = 15_d$   
 $\rightarrow a_0 = F$



# Conversion de $N = 687_d$ en hexadécimal avec un tableau de puissances (exemple supplémentaire)

- On suppose que  $k$  vaut 3 et on rappelle que  $16^3 = 4096$  et  $16^2 = 256$ .

$16^3$	$16^2$	$16^1$	$16^0$
$a_3$	$a_2$	$a_1$	$a_0$
?	?	?	?

$16^3$	$16^2$	$16^1$	$16^0$
$a_3$	$a_2$	$a_1$	$a_0$
0	?	?	?

$16^3$	$16^2$	$16^1$	$16^0$
$a_3$	$a_2$	$a_1$	$a_0$
0	2	?	?

$16^3$	$16^2$	$16^1$	$16^0$
$a_3$	$a_2$	$a_1$	$a_0$
0	2	A	?

$16^3$	$16^2$	$16^1$	$16^0$
$a_3$	$a_2$	$a_1$	$a_0$
0	2	A	F

- $N = 687_d$
- $687 < 16^3$   
 $\rightarrow a_3 = 0$
- $N = 687_d$
- $2 \times 16^2 \leq 687 < 3 \times 16^2$   
 $\rightarrow a_2 = 2$
- $N = 687 - 2 \times 16^2 = 175_d$
- $10 \times 16 < 175 < 11 \times 16$   
 $\rightarrow a_1 = A$
- $N = 175 - 10 \times 16^1 = 15_d$   
 $\rightarrow a_0 = F$
- $687_d = 02AF_h$