

Introduction aux systèmes informatiques

Représentation interne des informations

Michel Salomon

IUT de Belfort-Montbéliard
Département d'informatique

Un ordinateur traite des informations de différents types

- Instructions → `mov AH,09h` \Leftrightarrow 101101000001001 \Leftrightarrow B409 ; etc.
- Nombres → -23 ; 3,1415 ; etc.
- Images → formats PNG, PNM, JPG, GIF, etc.
- etc.

Toute information est représentée sous **forme binaire**

- Suite de chiffres binaires → 2 valeurs possibles : **0** ou **1**
- Un chiffre binaire ou **binary digit** → information élémentaire

Codage / format d'une information

Définit la correspondance entre représentations externe \leftrightarrow interne

- Convention / règle à suivre pour représenter une information
- Exemple : représentation de la quantité dix → dix, 10, X, etc.

En interne, un ordinateur traite deux types d'informations

- ① les instructions (`mov AH,09h` → 8086 - 16 bits);
 - Représentent les opérations effectuées par un ordinateur;
 - elles sont composées de deux champs :
 - 1 - le code de l'opération ($B4_{16} = 10110100_2$);
 - 2 - les opérandes de l'opération ($09_{16} = 00001001_2$)
 - elles constituent le langage machine;
 - le langage le plus proche du langage machine est le langage d'assemblage ou **Assembleur**
- ② les données élémentaires

En interne, un ordinateur traite deux types d'informations

- ① les instructions ;
- ② les données élémentaires
 - Ce sont les opérandes et les résultats d'opérations ;
 - elles sont de deux types :
 - 1 - numériques ;
 - nombres entiers relatifs : -23 ; -1 ; 0 ; 1 ; 12 ; 234 ; etc.
 - nombres fractionnaires : -0,125 ; 3,1415 ; etc.
 - nombres en notation scientifique : $3,1 \times 10^5$; etc.
 - 2 - non numériques
 - caractères alphanumériques : A, B, ..., Z,
a, b, ..., z, 0, 1, ..., 9
 - caractères spéciaux : ?, #, \$, etc.

Assembleur 16 bits dans MS-DOS - bonjour.asm

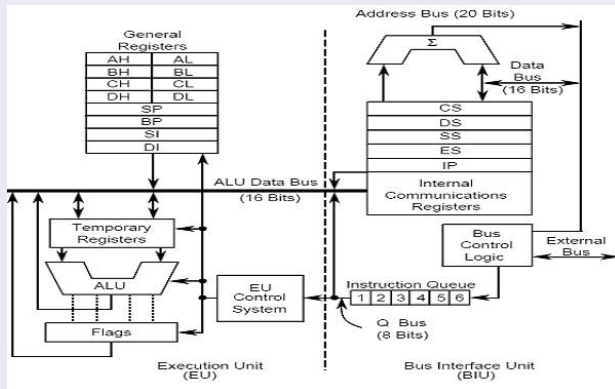
```
bits 16 ; indique qu'on produit du code 16 bits
org 0x100 ; directive de localisation mém. (COM)
section .data
    MessageBonjour: db 10,'Bonjour, monde...',10,'$'
section .text
    mov DX,MessageBonjour
    mov AH,09h
    int 21h
    mov AH,4Ch
    int 21h
```

Caractéristiques du 8086

- Bus d'adresses de 20 bits, mais registres d'adresses de 16 bits
- Registres de travail de 16 bits (AX=AH AL, BX, CX et DX)

Langage machine - Illustration avec le 8086

Structure globale du 8086 - Bus Interface Unit et Execution Unit



Langage machine - Illustration avec le 8086

Assembleur 16 bits dans MS-DOS - bonjour.asm

```
bits 16 ; indique qu'on produit du code 16 bits
org 0x100 ; directive de localisation mém. (COM)
section .data
    MessageBonjour: db 10,'Bonjour, monde...',10,'$'
section .text
    mov DX,MessageBonjour
    mov AH,09h
    int 21h
    mov AH,4Ch
    int 21h
```

Dump mémoire du programme

```
00000000 BA0C01 mov dx,0x10c
00000003 B409 mov ah,0x9
00000005 CD21 int 0x21
00000007 B44C mov ah,0x4c
00000009 CD21 int 0x21
```

Assembleur 32 bits dans linux - bonjour.asm

```
bits 32
section .data
    MessageBonjour: db 10,'Bonjour, monde...',10
    MessageBonjourLong: equ $-MessageBonjour
section .text
global _start
_start: ; Affichage de la chaine de caracteres
    mov EAX,4
    mov EBX,1
    mov ECX,MessageBonjour
    mov EDX,MessageBonjourLong
    int 80h
    ; Terminaison du programme
    mov EAX,1
    mov EBX,0
    int 80h
```

Processus d'assemblage (dans un terminal)

- Compilation → `nasm -f elf bonjour.asm`
- Édition des liens → `ld -s bonjour.o -o bonjour (??)`

Un format est une convention (normalisée) de représentation ou stockage d'un type de données tel que texte, image, son, etc.

Codage des images matricielles

- Une image est un ensemble de points lumineux → pixel
- Une image est un tableau 2D (matrice) → contient les pixels
- Profondeur de codage / couleur → nombre de bits par pixel
 - 1 bits → $2^1 = 2$ couleurs (noir et blanc)
 - 8 bits → $2^8 = 256$ couleurs (avec palette) ou niveaux de gris
 - 24 bits → $2^{24} = 16$ millions de couleurs (codage RVB)
- Représentation des couleurs sur écran → codage RVB
 - 1 pixel logique correspond à 3 pixels physiques à l'écran
 - chaque pixel d'un triplet physique émet une couleur primaire

Le bit (0 ou 1) est la plus petite unité de mesure

Toutes les autres unités sont des regroupements de bits

- un quartet est un groupement de 4 bits ;
- un octet (*Byte*) est un groupement de 8 bits ;
- un mot (*Word*) est un groupement d'octets (2 ou plus)
 - unité de base manipulée par un processeur (registres, etc.)

Unités de mesure basées sur l'octet

- Principalement utilisées pour parler du stockage de données
- Deux types d'unités : en puissances de 2 ; en puissances de 10
- Issues de recommandations officielles (IEC 60027-2 - 1998)
- Historiquement, utilisation des dénominations officielles des puissances de 10 pour les puissances de 2

Le bit (0 ou 1) est la plus petite unité de mesure

Unités de mesure basées sur l'octet

- Principalement utilisées pour parler du stockage de données

<i>Puissances de 2</i>			
1 Kibioctet	Kio ou KiB	1024 octets	2^{10}
1 Mébioctet	Mio ou MiB	1024 Kio	2^{20}
1 Gibioctet	Gio ou GiB	1024 Mio	2^{30}
1 Tébioctet	Tio ou TiB	1024 Gio	2^{40}

<i>Puissances de 10</i>			
1 Kiloctet	ko ou kB	1000 octets	10^3
1 Mégaoctet	Mo ou MB	1000 ko	10^6
1 Gigaoctet	Go ou GB	1000 Mo	10^9
1 Téraoctet	To ou TB	1000 Go	10^{12}

- 1 Kibioctet = 1 “Kilo binaire octet”

Usage encore confus des deux types d'unités

- Au niveau matériel
 - Octets comptés en puissances de 2
 - Souvent en utilisant les dénominations historiques (puissances de 10)
 - Exemple : barrette mémoire de 8 Go à la place de 8 Gio
- Au niveau logiciel
 - Octets pouvant être comptés dans l'une ou l'autre des unités
 - Usage fréquent des dénominations historiques pour les puissances de 2
 - Exemples (quelques commandes à lancer dans un terminal)
 - Puissances de 2 ; bonnes dénominations → `/sbin/ifconfig`
 - Puissances de 2 et 10 ; dénominations "historiques" → `df`
 - Puissances de 2 ; dénominations "historiques"
→ `cat /proc/meminfo`

Principes

- Un système de numération écrit les nombres à partir
 - d'une liste finie de symboles appelés chiffres ;
 - de règles définissant comment combiner ces chiffres
- Dans un système de base $b > 1$, un nombre entier naturel N est représenté par la suite de chiffres

$$c_n c_{n-1} \dots c_1 c_0$$

tels que :

- $c_i \in \{0, 1, \dots, b-2, b-1\}$;
- $c_n \neq 0$

et vérifiant

$$N = c_n \cdot b^n + c_{n-1} \cdot b^{n-1} + \dots + c_1 \cdot b^1 + c_0 \cdot b^0 = \sum_{i=0}^n c_i \cdot b^i$$

Principes (suite)

$$N = c_n \cdot b^n + c_{n-1} \cdot b^{n-1} + \dots + c_1 \cdot b^1 + c_0 \cdot b^0 = \sum_{i=0}^n c_i \cdot b^i$$

- À chaque position est associée un poids : la puissance de b
 - c_n est le chiffre de poids fort ;
 - c_0 est le chiffre de poids faible

Exemple - écriture de 2017 en base 10

- Utilisation de $b = 10$ chiffres $\rightarrow c_i \in \{0, 1, 2, \dots, 9\}$
- Nombre écrit comme une addition de puissances de $b = 10$

$$\begin{aligned} 2017_{10} &= 2 \times 10^3 + 0 \times 10^2 + 1 \times 10^1 + 7 \times 10^0 \\ &= 2000 + 0 + 10 + 7 \\ &= 2017 \end{aligned}$$

Numération en base b

Conversion d'une base b quelconque vers la base 10

- Nombre en base 3 $\rightarrow c_i \in ?$

$$(201)_3 = 201_3 = ?$$

- Nombre en base 11 $\rightarrow c_i \in ?$

$$(6A)_{11} = 6A_{11} = ?$$

Systèmes de numération usuels en informatique

- Système binaire, $b = 2$ et $c_i \in \{0, 1\}$;
- Système octal, $b = 8$ et $c_i \in \{0, \dots, 7\}$;
- Système hexadécimal, $b = 16$ et $c_i \in \{0, \dots, 9, A, B, \dots, F\}$
- Intérêt des systèmes octal et hexadécimal
 - représentation aisée des nombres ayant beaucoup de bits ;
 - conversion avec le système binaire simple (puissances de 2)

Numération en base b

Conversion d'une base b quelconque vers la base 10

- Nombre en base 3 $\rightarrow c_i \in \{0, 1, 2\}$

$$(201)_3 = 201_3 = (2)_{10} \times 3^2 + (0)_{10} \times 3^1 + (1)_{10} \times 3^0 = 2 \times 9 + 1 \times 1 = 19$$

- Nombre en base 11 $\rightarrow c_i \in \{0, 1, \dots, 8, 9, A\}$

$$(6A)_{11} = 6A_{11} = (6)_{10} \times 11^1 + (A)_{10} \times 11^0 = 6 \times 11 + 10 \times 1 = 76$$

Systèmes de numération usuels en informatique

- Système binaire, $b = 2$ et $c_i \in \{0, 1\}$;
- Système octal, $b = 8$ et $c_i \in \{0, \dots, 7\}$;
- Système hexadécimal, $b = 16$ et $c_i \in \{0, \dots, 9, A, B, \dots, F\}$
- Intérêt des systèmes octal et hexadécimal
 - représentation aisée des nombres ayant beaucoup de bits ;
 - conversion avec le système binaire simple (puissances de 2)

Correspondance entre les systèmes de numération usuels

Décimal	Binaire	Octal	Hexadécimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

Passage indirect d'un système de numération à un autre

- 1 Conversion de la base de départ b_d vers la base 10
- 2 Conversion de la base 10 vers la base d'arrivée b_a

Exemple 1 - Conversion de $6BC_{16}$ en base 12

- 1 Conversion de la base 16 vers la base 10

$$\begin{aligned}6BC_{16} &= (6)_{10} \times 16^2 + (B)_{10} \times 16^1 + (C)_{10} \times 16^0 \\6BC_{16} &= 6 \times 16^2 + 11 \times 16^1 + 12 \times 16^0 \\6BC_{16} &= 6 \times 256 + 11 \times 16 + 12 \times 1 = 1724_{10}\end{aligned}$$

- 2 Conversion de la base 10 vers la base 12

$$\begin{aligned}1724/12 &= 143 \text{ reste } 8 &\leftrightarrow & 1724 = 143 \times 12 + 8 \\143/12 &= 11 \text{ reste } 11 &\leftrightarrow & 143 = 11 \times 12 + 11 \\11/12 &= 0 \text{ reste } 11 &\leftrightarrow & 11 = 0 \times 12 + 11\end{aligned}$$

On obtient donc $6BC_{16} = 1724_{10} = BB8_{12}$

Changement de base

Exemple 2 - Conversion de 22_{10} en base 2

<i>Quotients successifs</i>	22	0
	11	1
	5	1
	2	0
	1	1
	0	1

*Restes
successifs*

22		2
0		11
		2
1		5
		2
		1
		2
		2
		0
		1
		1
		2
		0

Conversions usuelles en informatique

- binaire ; octal ; hexadécimal → décimal
 - Additionner, respectivement, des puissances de 2 ; 8 ; 16
- décimal → binaire ; octal ; hexadécimal
 - 1 Divisions entières successives, respectivement par 2 ; 8 ; 16, tant que le quotient est non nul ;
 - 2 Lecture des restes du dernier vers le premier

Conversions octal ou hexadécimal \rightarrow binaire

- Chaque chiffre octal ou hexadécimal est éclaté en son équivalent binaire sur respectivement 3 ou 4 bits
- Exemples :
 - $22_8 = 010\ 010_2 = 10010_2$;
 - $8A_{16} = 1000\ 1010_2 = 10001010_2$

Conversion binaire \rightarrow octal ou hexadécimal

- On remplace de droite à gauche chaque groupe de 3 ou 4 bits par son équivalent octal ou hexadécimal
- Si le nombre de bits du nombre binaire n'est pas un multiple de 3 ou 4, on complète à gauche avec des bits à 0
- Exemples :
 - $010101_2 = 010\ 101_2 = 25_8$;
 - $10101_2 = 0001\ 0101_2 = 15_{16}$

Un système informatique travaille sur des longueurs fixes de bits

des **mots (Word)** - assembleur x86 → 1 word = 16 bits = 2 octets

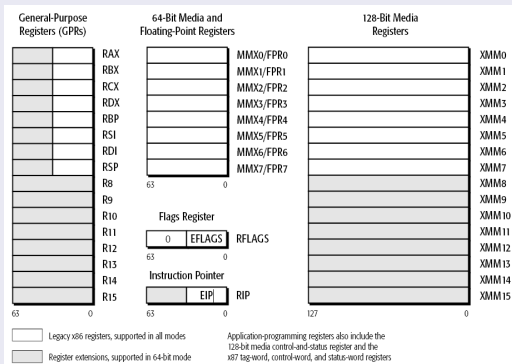
- Unité de base manipulée par un processeur (registres, etc.)
- Taille liée à la micro-architecture du processeur (registres)
 - processeur 16 bits → mots de 16 bits = 2 octets (dw)
 - processeur 32 bits → mots de 32 bits = 4 octets (dd)
 - processeur 64 bits → mots de 64 bits = 8 octets (dq)

Problématiques générales de représentation des nombres

- Problème de capacité
→ on ne peut représenter qu'un nombre fini de nombres
- Problème du signe
→ différencier les nombres négatifs des positifs ?
- Problème de perte de précision avec les nombres réels
→ gestion des arrondis ?

Représentation des adresses et données

Illustration avec “des” registres de l'architecture x86-64

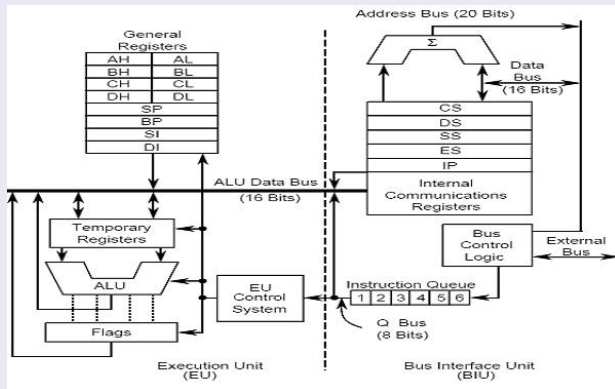


- Registres *de données* (nombres entiers) → RAX, RBX, RCX, RDX
- Registres *contenant* des offsets → RBP, RSI, RDI, RSP, RIP

Espace d'adressage virtuel sur au plus 64 bits

Représentation des adresses et données

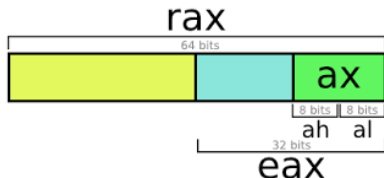
Structure globale du 8086 - Bus Interface Unit et Execution Unit



Représentation des adresses et données

Accessibilité des registres généraux

- 64 bits → RAX
- 32 bits → EAX
- 16 bits → AX
- 8 bits → AH et AL



Adressage de la mémoire lié au mode dans lequel est le processeur

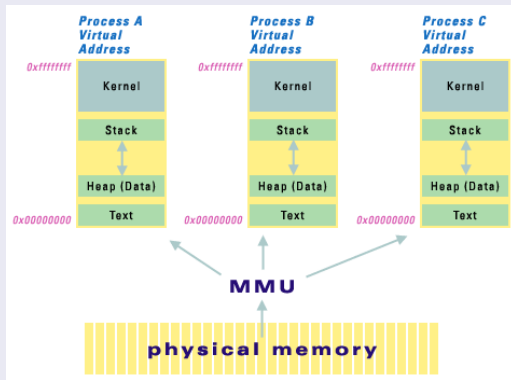
- *Legacy mode* (x86) → mode réel (au boot) et mode protégé
 - Systèmes d'exploitation / processeurs 16 bits ou 32 bits
 - Adresses virtuelles gérées par le SE d'au plus 32 bits en protégé
 - Mode réel (8086) → (au plus 1 Mio de mémoire)

@ physique (20 bits) = @ segment (16 bits) $\times 2^4$ + offset (déplacement sur 16 bits)

- *Long mode* (x86-64)
 - Systèmes d'exploitation / processeurs 64 bits
 - Adresses virtuelles gérées par le SE faisant au plus 64 bits

Représentation des adresses et données

Un processus est l'image mém. d'un programme en cours d'exéc.



- SE 32 bits → bloc / segment de mém. virtuelle de 4 Gio
- Adresse virtuelle “mappée” en physique par le noyau → **MMU**

Taille de l'espace d'adressage virtuel / physique “réel” fixé par le processeur

Représentation des adresses et données

Illustration avec les types de données de base en Java

- `char` → 1 octet ⇒ code UNICODE (cf. fin du cours)
- `byte` → 1 octet ⇒ entiers relatifs $\in \{-2^7, \dots, 2^7 - 1\}$
- `short` → 2 octets ⇒ entiers relatifs $\in \{-2^{15}, \dots, 2^{15} - 1\}$
- `int` → 4 octets ⇒ entiers relatifs $\in \{-2^{31}, \dots, 2^{31} - 1\}$
- `long` → 8 octets ⇒ entiers relatifs $\in \{-2^{63}, \dots, 2^{63} - 1\}$
- `float` → 4 octets ⇒ des réels entre $-3,4 \times 10^{38}$ et $3,4 \times 10^{38}$
- `double` → 8 octets ⇒ des réels entre $-1,7 \times 10^{308}$ et $1,7 \times 10^{308}$

Illustration avec les types de données de base en C / C++

- `signed char` → 1 octet ⇒ entiers relatifs $\in \{-2^7, \dots, 2^7 - 1\}$
- `unsigned char` → 1 octet ⇒ entiers nat. $\in \{0, \dots, 2^8 - 1\}$
- `short int` → 2 octets ⇒ entiers relatifs $\in \{-2^{15}, \dots, 2^{15} - 1\}$
- etc.

Endianess ou boutisme

Un ordre dans lequel les octets d'un nombre peuvent être :

- stockés en mémoire ;
- transmis à travers un réseau

Little endian vs. big endian - Stockage des octets

- **Little endian** → le moins significatif stocké en premier (LSB)
- **Big endian** → le plus significatif stocké en premier (MSB)

Exemple avec le nombre 0x12345678 (32 bits) en mémoire

Little endian	0000:	78
	0001:	56
	0002:	34
	0003:	12

Big endian	0000:	12
	0001:	34
	0002:	56
	0003:	78

Représentation des données numériques - entiers

Problématiques générales de représentation des entiers

- Problème de capacité
 - Codage sur un nombre fixe n de bits $\rightarrow 2^n$ nombres différents
 - Représentation dite en champ fixe
- Problème de représentation des nombres négatifs
 - Exemple : 1 bit pour coder le signe \rightarrow existence de 2 zéros

Représentation des entiers naturels en binaire naturel (ou pur)

- Nombres représentés en base 2, bits rangés dans l'ordre de leur poids, en complétant éventuellement à gauche avec des 0
- n bits permettent de représenter les nombres entre 0 et $2^n - 1$
- Ils peuvent être stockés dans des registres de n bits
 - Taille des registres d'un processeur fixée par le constructeur
 - Un ordinateur ne peut représenter qu'un nombre fini de valeurs
- Un calcul peut donner un résultat supérieur au nombre maximum représentable \rightarrow débordement (*overflow*)

Représentation en binaire naturel (ou pur) (suite) - Exemple

- 8 bits permettent de représenter l'ensemble $\{0, 1, \dots, 255\}$;
- en code binaire naturel, 108 a pour représentation sur 8 bits

	<i>MSB</i>						<i>LSB</i>
	0	1	1	0	1	1	0 0
position	7						0
poids	2^7	2^6	2^5	2^4	2^3	2^2	2^1 2^0

Most Significant Bit => bit le plus significatif

Least Significant Bit => bit le moins significatif

On a $108_{10} = 01101100_2$ sur 8 bits

Opérations arithmétiques binaires

- Principe identique à l'arithmétique décimale
- Addition et soustraction
 - Chiffre par chiffre ;
 - des poids faibles vers les poids forts ;
 - en propageant la retenue
- Multiplication et division correspondent respectivement à une série d'additions, soustractions

Tables d'addition et de soustraction

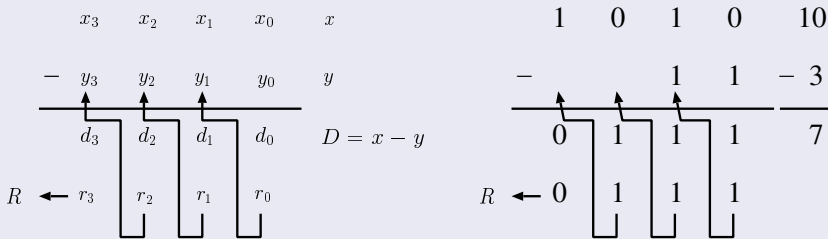
Opérandes		Addition		Soustraction	
x	y	retenue	x+y	retenue	x-y
0	0	0	0	0	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	0	0	0

Opérations arithmétiques binaires

Exemple de soustraction entre nombres codés sur 4 bits

- Débordement (*overflow*) si le résultat de la soustraction entre deux nombres nécessite plus de bits ;
- c'est-à-dire si la dernière retenue est égale à 1

Soustraction de 3 à 10



On distingue différentes représentations des entiers signés

- Représentation avec le signe et la valeur absolue
- Représentations complémentées
- Représentations biaisées (ou excédentaires)

Représentation avec le signe et la valeur absolue

- Bit le plus à gauche utilisé pour le signe

$0 \rightarrow +$ et $1 \rightarrow -$

- n bits permettent de coder les entiers N tels que

$$-(2^{n-1} - 1) \leq N \leq +(2^{n-1} - 1)$$

- Exemple

- 4 bits permettent de représenter $\{-7, \dots, 0, \dots, 7\}$;
- on a notamment $+5 = 0101$ et $-5 = 1101$

- Remarques (inconvenients)

- 1 Existence de deux zéros ($000 \dots 0$ et $100 \dots 0$)
- 2 Tables d'addition et de multiplication compliquées

Représentation en complément à 1

- Conservation du bit de signe
- Représentation d'un entier négatif $-X$ obtenue en inversant chaque bit de la représentation de l'entier positif $+X$
 - 0 est remplacé par 1 ;
 - 1 est remplacé par 0
- n bits permettent de coder les entiers N tels que

$$-(2^{n-1} - 1) \leq N \leq +(2^{n-1} - 1)$$

- Exemple
 - 4 bits permettent de représenter $\{-7, \dots, 0, \dots, 7\}$;
 - on a notamment $+5 = 0101$ et $-5 = 1010$
- Remarques
 - Existence de 2 zéros ($000\dots 0$ et $111\dots 1$)
 - On parle également de complément logique

Représentation des données numériques - entiers relatifs

Exemple de représentation en complément à 1

- Codage sur 5 bits
 - Bit de **Signe** → bit de poids fort
 - Nombres représentables (31 nombres)

$$\begin{aligned} -(2^4 - 1) &\leq N \leq +(2^4 - 1) \\ -15 &\leq N \leq +15 \end{aligned}$$

- Représentation de $+7 \rightarrow 7_{10} = 0111_2$ sur 4 bits

$$\begin{array}{c} \underline{0 \quad 1 \quad 1 \quad 1} \\ \text{position} \quad \begin{array}{cccc} 4 & 3 & 2 & 1 & 0 \\ S & & & & \end{array} \end{array}$$

- Représentation de $-11 \rightarrow 11_{10} = 1011_2$ sur 4 bits

$$\begin{array}{c} \underline{0 \quad 1 \quad 0 \quad 1 \quad 1} \rightarrow \underline{1 \quad 0 \quad 1 \quad 0 \quad 0} \\ \text{position} \quad \begin{array}{cccccc} 4 & 3 & 2 & 1 & 0 & \\ S & & & & & S \end{array} \\ \text{Rep. de } +11 \qquad \qquad \text{Rep. de } -11 \end{array}$$

- $\forall a \geq 0, a + (-a) = 11111_2 = 31 = 2^5 - 1$

Représentation en complément à 2

- Principaux objectifs
 - L'addition d'un nombre et de son opposé donne zéro
 - Suppression du double zéro
 - Quasiment autant de positifs que de négatifs
- Principe de calcul des nombres négatifs
 - 1 On calcule le complément à 1 du nombre ;
 - 2 on additionne 1 à la valeur du complément à 1
- n bits permettent de coder les entiers N tels que

$$-2^{n-1} \leq N \leq +(2^{n-1} - 1)$$

- Exemple
 - 4 bits permettent de représenter $\{-8, \dots, 0, \dots, 7\}$;
 - on a notamment $+5 = 0101$ et $-5 = 1011$
- Remarques
 - Opérations arithmétiques intéressantes (soustraire un nombre se ramène à l'addition de son complément à 2)
 - On parle également de complément arithmétique

Représentation biaisée ou excédentaire (plus de bit de signe)

- Code les nombres de façon décalé en fixant le binaire de zéro
→ autant de nombres positifs (avec zéro) que de négatifs
- Principe de calcul
 - 1 Ajouter une valeur de biais (ou d'excès);
 - 2 utiliser le code binaire naturel du résultat
- Sur n bits, généralement le biais = 2^{n-1} → représente 0
- Un nombre négatif $-X$ est représentable si $-X \geq -\text{biais}$
- n bits permettent de coder les entiers N tels que

$$-2^{n-1} \leq N \leq +(2^{n-1} - 1)$$

- Exemple
 - 4 bits peuvent représenter $\{-8, \dots, 0, \dots, 7\}$; biais = $2^3 = 8$;
 - on a notamment $-8 = 0000$ et $5 = 1101$ (binaire de 13)
- Remarque
 - On parle également de code excédent-*valeur de biais*;
 - ou de représentation sur n bits en excédent à *valeur de biais*

Exemple de représentation biaisée ou excédentaire

- Codage sur 5 bits (code excédent-16)
 - **Plus de bit de signe** ; **biais** = $2^4 = 16 \rightarrow$ fixe le binaire pour 0
 - Nombres représentables (32 nombres)

$$-2^4 \leq N \leq +(2^4 - 1)$$

$$-16 \leq N \leq +15$$

- Représentation de $+7 \rightarrow 7 + 16 = 23_{10} = 10111_2$ sur 5 bits

$$\begin{array}{cccccc} & & & & \underline{1} & \underline{0} & \underline{1} & \underline{1} & \underline{1} \\ \text{position} & & & & 4 & 3 & 2 & 1 & 0 \end{array}$$

- Représentation de $0 \rightarrow 0 + 16 = 16_{10} = 10000_2$ sur 5 bits

- Représentation de $-11 \rightarrow -11 + 16 = 5_{10} = 00101_2$ sur 5 bits

$$\begin{array}{cccccc} & & & & \underline{0} & \underline{0} & \underline{1} & \underline{0} & \underline{1} \\ \text{position} & & & & 4 & 3 & 2 & 1 & 0 \end{array}$$

Représentation d'entiers signés sur 16 bits

Décimal	Bit de signe	Complément à 1	Complément à 2
+32767	011...111	011...111	011...111
+32766	011...110	011...110	011...110
...
+1	000...001	000...001	000...001
+0	000...000	000...000	000...000
-0	100...000	111...111	—————
-1	100...001	111...110	111...111
...
-32766	111...110	100...001	100...010
-32767	111...111	100...000	100...001
-32768	—————	—————	100...000

- Rationnels et réels sont des nombres comportant une virgule
- Représenter tous les rationnels et les réels est impossible
 - Tout nombre est codé sur un nombre fini de bits
- Prise en compte de la virgule
- On distingue différentes représentations des réels :
 - la représentation en virgule fixe ;
 - la représentation en virgule flottante - norme IEEE 754
 - simple précision sur 32 bits (`float` en C / C++);
 - double précision sur 64 bits (`double` en C / C++);
 - précision étendue sur 80 bits (`long double` en C / C++)

Institute of Electrical and Electronics Engineers

<http://www.ieee.org>

Représentation en virgule fixe

- Similaire à celle des nombres entiers, avec une virgule virtuelle
⇒ gestion de la virgule par le programmeur...
- Exemple - sur 8 bits
 - Un choix possible :
 - 4 bits pour la partie entière ;
 - 4 bits pour la partie décimale
 - ainsi 01011100 représente $0101,1100_2 = 101,11_2$
- Changement de base
 - binaire ⇒ décimal (base $b \rightarrow$ remplacer 2 par b)
 - partie entière \rightarrow addition de puissances de 2 positives ;
 - partie décimale \rightarrow addition de puissances de 2 négatives

$$\begin{aligned}101,11_2 &= 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} \\ &= 2^2 + 2^0 + 2^{-1} + 2^{-2} \\ &= 4 + 1 + 0,5 + 0,25 = 5,75\end{aligned}$$

Représentation en virgule fixe (suite)

- Changement de base (suite)
 - décimal \Rightarrow binaire (base $b \rightarrow$ remplacer 2 par b)
 - partie entière \rightarrow conversion d'un nombre entier en base 2
 - partie décimale
 - Multiplications successives par 2 de nombres décimaux ;
 - arrêt quand la partie décimale est nulle ou que le nombre de bits correspond à la taille du registre
 - Le nombre binaire cherché s'obtient en lisant les parties entières de la première vers la dernière

On a ainsi $0,5625_{10} = 0,1001_2$ car

$$0,5625 \times 2 = 1,125 (= 1 + 0,125)$$

$$0,125 \times 2 = 0,25 (= 0 + 0,25)$$

$$0,25 \times 2 = 0,5 (= 0 + 0,5)$$

$$0,5 \times 2 = 1,0 (= 1 + 0)$$

Donner la valeur décimale représentée par $1011,10101_2$

$$1011,10101_2 = ?$$

Donner le binaire représentant $\frac{1}{7}$ sur 8 bits (7 bits pour la partie décimale)

$$\frac{1}{7} = ?$$

Représentation des données numériques - réels

Donner la valeur décimale représentée par $1011,10101_2$

$$\begin{aligned}1011,10101_2 &= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} \\ &\quad + 0 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5} \\ &= 2^3 + 2^1 + 2^0 + 2^{-1} + 2^{-3} + 2^{-5} \\ &= 8 + 2 + 1 + \frac{1}{2^1} + \frac{1}{2^3} + \frac{1}{2^5} = 11 + \frac{1}{2} + \frac{1}{8} + \frac{1}{32} \\ &= 11 + 0,5 + 0,125 + 0,03125 = \mathbf{11,65625}\end{aligned}$$

Donner le binaire représentant $\frac{1}{7}$ sur 8 bits (7 bits pour la partie décimale)

$$\frac{1}{7} = ?$$

Représentation des données numériques - réels

Donner le binaire représentant $\frac{1}{7}$ sur 8 bits (7 bits pour la partie décimale)

$$\frac{1}{7} = 0,142857142857 \dots \rightarrow \text{partie entière} = \mathbf{0}$$

$$0,142857 \times 2 = 0,285714 \rightarrow \mathbf{0}$$

$$0,285714 \times 2 = 0,571428 \rightarrow \mathbf{0}$$

$$0,571428 \times 2 = 1,142856 \rightarrow \mathbf{1}$$

$$0,142856 \times 2 = 0,285712 \rightarrow \mathbf{0}$$

$$0,285712 \times 2 = 0,571424 \rightarrow \mathbf{0}$$

$$0,571424 \times 2 = 1,142848 \rightarrow \mathbf{1}$$

$$0,142848 \times 2 = 0,285696 \rightarrow \mathbf{0}$$

$$\dots = \rightarrow \text{partie décimale} = \mathbf{0010010} (\text{sur 7 bits})$$

$$\text{d'où } \frac{1}{7} \approx \mathbf{0,0010010_2} \approx 0,140625$$

Représentation en virgule flottante - Notation scientifique

- Un nombre réel N est représenté sous la forme

$$N = + \text{ ou } - M \times B^E$$

- M = mantisse en base B ;
- B = base (2, 8, 10, 16, etc.) ;
- E = exposant

tels que :

- la mantisse M , aussi appelée significande, est un nombre comportant une virgule ;
 - l'exposant E est un entier relatif
- Exemples
 - $123000 = 1,23 \times 10^5$;
 - $-0,002 = -0,02 \times 10^{-1}$;
 - $1011_2 = 1,011 \times 2^3$

Représentation en virgule flottante - Notation scientifique

- **Problème** → de nombreuses représentations possibles

- La représentation dépend de la position de la virgule
- Exemple

$$\begin{aligned}3,141592 &= 0,3141592 \times 10^1 \\ &= 3,141592 \times 10^0 \\ &= 31,41592 \times 10^{-1} \\ &= \dots\end{aligned}$$

- **Solution** → une représentation normalisée

- Normalisation en imposant la forme de la mantisse
 - Tronquer ou arrondir la mantisse peut être nécessaire, en raison de la taille fixe pour la stocker
- Exemple
 - 1 seul chiffre significatif (non zéro) avant la virgule

$$3,141592 \times 10^0$$

Représentation en virgule flottante - Notation scientifique

- Nombre fini de valeurs → utilisation de valeurs approchées
- Densité des nombres variable → plus on s'éloigne de 0, plus la distance entre les nombres représentés s'accroît
 - Exemple
 - notation scientifique sur 4 chiffres en base 10 ;
 - mantisse à 2 chiffres, normalisée sous la forme $0,nc$ ($n \neq 0$) ;
 - exposant sur 2 chiffres

$$\begin{array}{c} \text{Nb de mantisses} \qquad \text{Nb d'exposants} \\ \underbrace{(2 \times 9 \times 10)} \times \underbrace{((2 \times 10 \times 10) - 1)} = \\ 35820 \text{ nombres possibles} \end{array}$$

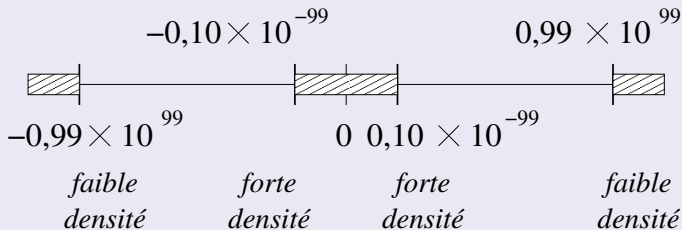
Représentation des données numériques - réels

Représentation en virgule flottante - Notation scientifique

- Nombre fini de valeurs → utilisation de valeurs approchées
- Densité des nombres variable → plus on s'éloigne de 0, plus la distance entre les nombres représentés s'accroît
- Exemple

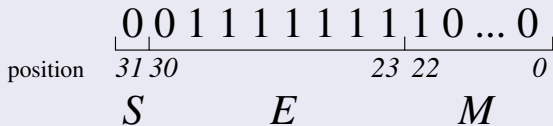
→ entre $0,11 \times 10^0$ et $0,12 \times 10^0$ écart de $0,01$

→ entre $0,11 \times 10^{10}$ et $0,12 \times 10^{10}$ écart de $10^8 = 100000000$



Représentation en virgule flottante - Norme IEEE 754

- Format des nombres flottants simple précision sur 32 bits
 - 1 bit de signe ;
 - 8 bits d'exposant ;
 - 23 bits de mantisse



- Format des nombres flottants double précision sur 64 bits
 - 1 bit de signe ;
 - 11 bits d'exposant ;
 - 52 bits pour la mantisse

Représentation en virgule flottante - Simple précision (32 bits)

- Le bit le plus à gauche est utilisé pour le **Signe** :
 - 0 → + ;
 - 1 → -
- L'**Exposant** est représenté sur 8 bits en excédent à 127

$$E_{\text{réel}} \in \{-126, -125, \dots, 126, 127\}$$

- car il y a deux valeurs réservées :
 - -127 → tous les bits à 0 ;
 - 128 → tous les bits à 1
- La **Mantisse** M est normalisée sous la forme suivante :

$$1, cc \dots$$

- où $c \in \{0, 1\}$;
- le 1 avant la virgule n'est pas représenté parmi les 23 bits de la mantisse → bit caché (*hidden bit*)

Représentation des données numériques - réels

Représentation en virgule flottante - Simple précision (32 bits)

- Exemple 1 - Codage de 1,5

$$1,5_{10} = 1,1_2 = 1,1_2 \times 2^0 \text{ (notation scientifique normalisée en base 2)}$$

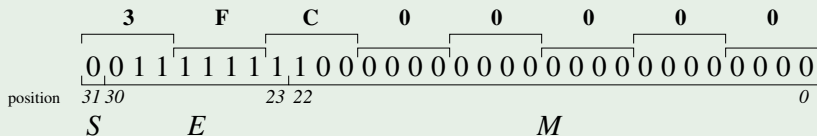
- Signe positif $\rightarrow S = 0_2$
- Exposant réel égal à 0 \rightarrow exposant biaisé valant

$$E_{\text{biaisé}} = E_{\text{réel}} + 127 = 0 + 127 = 127 = 01111111_2$$

- Mantisse égale à $1,1_2 \rightarrow M = 10\dots0_2$

On a ainsi $1,5_{10}$ qui est codé par $3FC00000_{16}$

Codage IEEE 754 simple précision de 1,5₁₀



Représentation des données numériques - réels

Représentation en virgule flottante - Simple précision (32 bits)

- Exemple 2 - Valeur décimale du réel $BF270000_{16}$

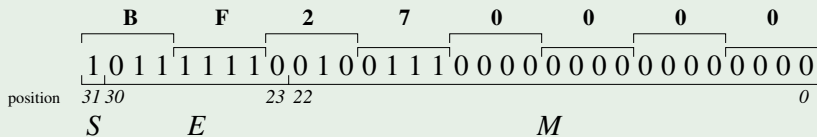
- $S = 1_2 \rightarrow$ signe négatif
- Exposant biaisé égal à 126 \rightarrow exposant réel valant

$$E_{\text{réel}} = E_{\text{biaisé}} - 127 = (01111110_2 = 126) - 127 = -1$$

- $M = 01001110 \dots 0_2 \rightarrow$ mantisse égale à $1,0100111_2$

$$-1,0100111_2 \times 2^{-1} = -0,10100111_2 = -0,65234375_{10}$$

Codage IEEE 754 simple précision d'un réel



Représentation en virgule flottante - Simple précision (32 bits)

- Exceptions
 - ① Tous les bits de E sont à 0 ($E_{\text{réel}} = -127$)
 - Tous les bits de M à zéro $\rightarrow \pm 0$
 - M différent de zéro \rightarrow mantisse dénormalisée
 - ② Tous les bits de E sont à 1 ($E_{\text{réel}} = 128$)
 - Tous les bits de M à zéro $\rightarrow \pm\infty$
 - M différent de zéro \rightarrow **Not A Number**
- Arrondir les nombres flottants
 - Au plus près (méthode d'arrondi par défaut)
 - Tout nombre compris entre deux est arrondi à la valeur la plus proche avec un bit de poids faible à 0
 - Vers zéro
 - Vers $-\infty$
 - Vers $+\infty$

Représentation des données non numériques

Chaque symbole est représenté par une valeur binaire

Les données non numériques sont codées via une table de correspondance qui donne pour chaque caractère son code

Pourquoi faut-il une convention de codage ?

Pour assurer la cohérence des données lors des Entrées / Sorties

- Dans un ordinateur → affichage / saisie au clavier ; etc.
- Entre ordinateurs → échange d'informations

Code **American Standard Code for Information Interchange** (1)

- Norme codant chaque caractère sur 7 bits → 128 caractères
 - Codes 0x00 à 0x1F → dialogues avec les périphériques
 - Codes 0x20 à 0x2F → caractères de ponctuation
 - etc.
- Suffisant pour langue anglaise → pas de caractères accentués

Représentation des données non numériques

Code *American Standard Code for Information Interchange* (2)

- L'ordinateur gère des octets = 8 bits → codes supplémentaires
- Codes 0x80 à 0xFF en plus → usage non normalisé
 - Caractères accentués (Latin-1)
 - Symboles mathématiques
 - etc.

Code *International Standard Organisation 8859-1* (ASCII étendu)

Norme codant chaque caractère sur 8 bits → 256 caractères

- 128 premiers caractères identiques à l'ASCII
 - Codes 0x00 à 0x1F → caractères non affichables
 - Codes 0x30 à 0x39 → les chiffres (les 4 bits de poids faible donnent la valeur en binaire du chiffre)
 - etc.
- 128 derniers caractères varient avec le numéro de page (1, 2, ...)
 - 8859-1 → langues romanes (Latin-1)
 - 8859-2 → langues slaves d'origines latines (hongrois, etc.)

Représentation des données non numériques

Code **UNICODE** - Objectif

Donner à chaque caractère de tout système d'écriture un nom / ID numérique unique, indépendamment du matériel et du logiciel

Standard informatique développé par un consortium

- Unicode 11.0 a été publié en juin 2018 (<http://www.unicode.org>)
 - 139 systèmes d'écritures encodant 136755 symboles différents
- Lié à la norme ISO/CEI 16046 (Unicode est un sur-ensemble)
 - Travail → en parallèle et synchronisé
 - Unicode → classement alphabétique, composition des caractères, etc.
- Conçu pour offrir un espace de codage organisé en couches
 - 1114112 codes différents (nombre entier ou *point de code*)
 - divisés en 17 zones (ou plans) de 65536 points de codes
 - notation globale d'un point de code (xxxx va de 4 à 6 chiffres)
 - U+0000 à U+FFFF → plan multilingue de base
 - U+10000 à U+FFFFF → 15 plans suivants (F0000-FFFFF = privé)
 - U+100000 à U+10FFFF → dernier plan (privé)

Représentation des données non numériques

Unicode sépare la définition d'un caractère de son codage physique

- Unité de base du codage (ou *codet*) - 1 car. occupe au plus 21 bits
 - 1 octet (mot de 8 bits) → UTF-8 ⇒ 1 caractère = de 1 à 4 mot(s)
 - 2 octets (mot de 16 bits) → UTF-16 ⇒ 1 car. = 1 ou 2 mot(s)
 - 4 octets (mot de 32 bits) → UTF-32 ⇒ 1 caractère = 1 mot
- La couche *Formalisme de codage des caractères* précise
 - la taille de l'unité de codage en nombre d'unités de base
 - comment retrouver le point de code correspondant

Universal Transformation Format

- Transformation universelle pour représenter un point de code
- Longueur d'encodage variable (nombre de codets)
- UTF-8
 - 128 premiers caractères identiques à l'ASCII
 - encode près de 90% des textes dans le Web

Codage des caractères en UTF-8

Binary format of bytes in sequence

1st Byte	2nd Byte	3rd Byte	4th Byte	Number of Free Bits	Maximum Expressible Unicode Value
0xxxxxxx				7	007F hex (127)
110xxxxx	10xxxxxx			(5+6)=11	07FF hex (2047)
1110xxxx	10xxxxxx	10xxxxxx		(4+6+6)=16	FFFF hex (65535)
11110xxx	10xxxxxx	10xxxxxx	10xxxxxx	(3+6+6+6)=21	10FFFF hex (1,114,111)

- Espace occupé par un caractère → varie avec le point de code
 - Code allant de 0 à 127 → 1 octet
 - Code allant de 128 à 2047 → 2 octets
 - etc.
- Illustration avec le caractère é de code 233 = 11101001₂
 - Point de code occupant 2 octets
 - Point de code = 11000011 10101001₂ = C3 A9₁₆
- Exemple d'affichage en ISO 8859-1 d'un texte en UTF-8
 - Saisi → Un petit exemple de texte qui a été écrit
 - Affiché → Un petit exemple de texte qui a Ã©tÃ© Ã©crit