Systèmes d'exploitation et Architecture I

Représentation des nombres (1) - les entiers

Marie Duflot-Kremer présenté par Sovanna Tan

Licence 2 - Université Paris Est Créteil

Introduction

Dans un ordinateur, besoin de représenter les nombres :

- de manière compacte,
- pour avoir des calculs faciles.

Plusieurs types de nombres à représenter :

- Entiers positifs $\rightarrow$ écriture en base 2 (binaire)
- Entiers signés $\rightarrow$?
- Nombres réels $\rightarrow$?

Précision finie

Stockage en mémoire $\Rightarrow$ taille limitée pour chaque donnée.

Exemple : trois chiffres décimaux

On peut représenter les nombres de 000 à 999 mais pas :

- les nombres plus grands que 999
- les négatifs
- les réels

Problèmes de dépassement, de précision.

Retour sur l'exemple

588 + 657 = ? ? ?
003 - 005 = ? ? ?
005 / 002 = ? ? ?

Représentation des nombres

On représente les nombres à l'aide de symboles.

Attention

Bien faire la différence entre un nombre (par exemple dix) et sa (ses) représentations :

- 10 (décimal),
- 1010 (binaire),
- A (hexadécimal)
- sur les doigts (plusieurs représentations possibles)

Représentation - les bases

Pour représenter un entier, on choisit une base.
Les plus connues :

- 10 (décimal), la plus connue pour l'homme,
- 2 (binaire), pour l'ordi,
- 16 (hexadécimal), représentation plus compacte que le binaire.
- 1 (unaire), quand on compte sur ses doigts

D'autres plus rares :

- 12 (duodécimal) utilisé par les Egyptiens pour le compte en heures/mois,
- 20 (vigésimal) utilisée par les Aztèques, un peu en france (quatre-vingt)
- 60 (sexagésimal) mesure du temps, des angles, utilisée par les Babyloniens

Représentation - choix des symboles

- Les “chiffres arabes” utilisés pour le système décimal se composent de 10 symboles : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- Pour les bases 1 à 10 on utilise (une partie de) ces symboles
  - ex : en binaire 0 et 1
- Pour une base 11 ou plus il faut compléter :
  - ex : en hexa, on utilise les chiffres et A B C D E F
- Chez les babyloniens, deux symboles :
  - Un : , Dix : 
  - On les combine jusqu'à 60 :  $= 24$
  - Au-delà, on utilise la numération en base 60.

    ex :   $= 1 \times 60 + 24 = 84$

Comment ça marche ?

En base 10 : $\begin{aligned} 342 &= 300 + 40 + 2 \\ &= 3 \times 100 + 4 \times 10 + 2 \\ &= 3 \times 10^2 + 4 \times 10^1 + 2 \times 10^0 \end{aligned}$

Dans les autres bases c'est pareil :
En base b (≥ 2) on note $(n_t ... n_1 n_0)_b$ le nombre n qui vaut

$$n = n_t \times b^t + ... + n_1 \times b + n_0 \times 1 = \sum_{i=0}^{t} n_i \times b^i$$

Remarque :

On appelle cela un système de numération **positionnel** : la valeur d'un chiffre dépend de sa position dans la représentation. En base b, le $i^{\text{ème}}$ chiffre à partir de la droite correspond à b^{i-1}.
Un système de numération non positionnel : la numération romaine

Représentation binaire

Pourquoi le binaire ?

Facilement accepté par l'ordinateur : différence entre deux tensions.
Quelques ordinateurs fonctionnent en ternaire, mais très peu.

Base 2 $\rightarrow$ uniquement des 0 et des 1.
Choix arbitraire : on aurait pu prendre deux symboles quelconques, ex : α et β.

Rappel :

avec n symboles (=BInary digiTs = bits) on peut représenter en binaire un entier positif entre :

- tous les bits à 0 : $\sum_{i=0}^{n-1} 0 \times 2^i = 0$
- et tous les bits à 1 : $\sum_{i=0}^{n-1} 1 \times 2^i = 2^n - 1$

Exemples : $1101_2 = 13_{10}$
$7_{10} = 111_2$

Changements de base (1) - binaire → décimal

- Méthode 1 : addition de puissances successives.
- On utilise la formule ci-dessus.
- $n = (n_t ... n_1 n_0)_2 = n_t \times 2^t + ... n_1 \times 2^1 + n_0 \times 2^0$
- Rappel : comme on est en binaire, pour tout i on a $n_i \in \{0, 1\}$

Exemple

$$\begin{aligned} 10110_2 &= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\ &= 16 + 4 + 2 \\ &= 22_{10} \end{aligned}$$

Changements de base (1bis) - binaire → décimal

- Méthode 2 : multiplications par 2 et ajouts de 1.

Reprenons l'exemple

$$\begin{aligned}
10110_2 &= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\
&= (1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0) \times 2 + 0 \\
&= (1011_2 \times 2) + 0 \\
&= (((101_2 \times 2) + 1) \times 2) + 0 \\
&= (((((10_2 \times 2) + 1) \times 2) + 1) \times 2) + 0 \\
&= \ldots
\end{aligned}$$

- on part du 1 le plus à gauche,
- pour chaque nouveau chiffre on multiplie par 2,
- et si en plus le chiffre est 1 on ajoute 1.

Changements de base (1bis) - sur un exemple

- on part du 1 le plus à gauche,
- pour chaque nouveau chiffre on multiplie par 2,
- et si en plus le chiffre est 1 on ajoute 1.

Exemple :

$010101_2 = ?\ 21_{10}$

- on part du 1 le plus à gauche $\rightarrow 1$
- le chiffre suivant est un 0 $\rightarrow 1 \times 2 = 2$
- le chiffre suivant est un 1 $\rightarrow (2 \times 2) + 1 = 5$
- le chiffre suivant est un 0 $\rightarrow 5 \times 2 = 10$
- le chiffre suivant est un 1 $\rightarrow (10 \times 2) + 1 = 21$

Changements de base (2) - décimal $\rightarrow$ binaire

Méthode :

- Diviser le nombre choisi par deux,
- garder le quotient, noter le reste,
- diviser le quotient par 2, écrire le nouveau reste,
- continuer jusqu'à avoir un quotient $=0$.

Attention : Il faut écrire les restes de droite à gauche

Exemple : 20

$$
\begin{aligned}
&20 \;/\; 2 = 10 \text{ reste } 0\\
&10 \;/\; 2 = 5 \text{ reste } 0\\
&5 \;/\; 2 = 2 \text{ reste } 1\\
&2 \;/\; 2 = 1 \text{ reste } 0\\
&1 \;/\; 2 = 0 \text{ reste } 1
\end{aligned}
$$

Résultat : 10100

Représentation hexadécimale

- Base 16 $\leftrightarrow$ représentation hexadécimale.
- Besoin de 16 symboles :
  - 10 chiffres : 0 ... 9
  - plus 6 lettres : A B C D E F
- La façon dont est représenté le binaire car :
  - plus compacte,
  - plus lisible,
  - facile à convertir en binaire.

Changements de base (3) - hexa $\rightarrow$ binaire

hexa	$\longrightarrow$	binaire
1 symbole	$\longrightarrow$	4 bits

- chaque symbole hexadecimal a sa correspondance en binaire
- $C_{16} = 12_{10} = 1100_2$
- Pour traduire un nombre plus long on traduit les signes hexa un par un :

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

Exemple

$CAFE_{16} = 1100\ 1010\ 1111\ 1110_2$

Changements de base (4) - binaire $\longrightarrow$ hexa

binaire	$\longrightarrow$	hexa
4 bits	$\longrightarrow$	1 symbole

- Si besoin on complète avec des 0 à gauche pour avoir un multiple de 4 bits
- On traduit les bits par bloc de 4 en hexadécimal

$$\begin{array}{cccccccccccccccc}
0000 & 0001 & 0010 & 0011 & 0100 & 0101 & 0110 & 0111 & 1000 & 1001 & 1010 & 1011 & 1100 & 1101 & 1110 & 1111 \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & \mathrm{A} & \mathrm{B} & \mathrm{C} & \mathrm{D} & \mathrm{E} & \mathrm{F}
\end{array}$$

Exemple

$0111\ 0011\ 1010_2 = 73A_{16}$

Entiers non signés sur n bits

On a un entier sur k bits et on le veut sur $n \neq k$ bits. Comment faire ?

- Si $k < n$, il suffit de rajouter $n - k$ zéros à gauche.
- Si $k > n$ c'est plus compliqué :
  - Si on a au moins $k - n$ zéros à gauche on en enlève $k - n$.
  - Sinon c'est impossible

Exemple :

Représenter 8_{10} sur :

- 6 bits : 001000_2
- 4 bits : 1000_2
- 3 bits : impossible !

Entiers signés

Jusqu'à présent on a considéré des entiers non signés :

- pas besoin de signe (positif ou négatif)
- tous les bits ont servi à coder la valeur absolue

Pour calculer on va avoir besoin de nombres signés

- besoin de représenter le signe...
- de manière simple...
- tout en facilitant les calculs.

Représentation signe et valeur absolue

Principe

- Bit de gauche = signe (0 : positif, 1 : négatif).
- Les autres bits représentent la valeur absolue.
- Permet de représenter sur n bits un nombre allant de $-(2^{n-1}-1)$ à $2^{n-1}-1$.

Exemple

$00010_2 \longrightarrow 2$
$10010_2 \longrightarrow -2$

Représentation signe et valeur absolue (2)

Avantages :

- calculer l'opposé d'un nombre est facile,
- le bit de gauche donne le signe :
  - $0 \longrightarrow$ positif,
  - $1 \longrightarrow$ négatif.

Inconvénients :

- difficile d'additionner deux nombres de signes différents
- difficile de soustraire des nombres
- deux représentations de 0

Attention : dans la suite du cours (et donc à l'examen) on ne va PAS utiliser cette notation.

Représentation en complément à deux

Principe (sur n bits)

- Premier bit de poids négatif : -2^{n-1},
- les autres de poids positif : $+2^i$

$$(a_{n-1}...a_1 a_0)_2 \leftrightarrow -2^{n-1} \times a_{n-1} + \sum_{i=0}^{n-2} 2^i \times a_i$$

Exemples sur 8 bits

- $-45_{10} = -128 + 83 = -128 + 64 + 16 + 2 + 1$
  $= 11010011_2$
- $10111110_2 = -128 + 32 + 16 + 8 + 4 + 2 = -66_{10}$

Complément à deux (2) - valeurs extrêmes

Valeur minimale sur n bits quand :

- a_{n-1} vaut 1
- les a_i valent tous 0

Valeur maximale sur n bits quand :

- a_{n-1} vaut 0
- les a_i valent tous 1

Permet de représenter sur n bits des entiers de -2^{n-1} à $2^{n-1} - 1$.

Complément à deux (3)

Inconvénient :

- opposé d'un nombre (un peu) plus difficile à calculer (*cf.* plus loin)

Avantages :

- une seule représentation de 0,
- addition et soustraction + simples,
- le bit de gauche donne le signe :
  - $0 \rightarrow$ positif,
  - $1 \rightarrow$ négatif.

Remarque : Pour la suite du cours, on va utiliser uniquement la représentation en complément à deux.

Dépassement

Qu'appelle-t-on dépassement ?

Définition

On dit qu'il y a dépassement de capacité lors d'une opération arithmétique sur n bits lorsque le résultat ne peut pas être représenté sur n bits. L'opération donne alors un résultat incorrect.

On en verra dans ce cours dans :

- les décalages à gauche
- les additions
- les soustractions

Extension de signe

Entier signé sur n bits $\longrightarrow$ entier signé sur $n+k$ bits.

Méthode

On rajoute des bits à gauche de la même valeur que le bit de signe.

Exemples

- $5_{10} = 0101_2$ (sur 4 bits) $= 00000101_2$ (sur 8 bits).
- $-4 = 1100_2$ (sur 4 bits) $= 11111100_2$ (sur 8 bits).

Décalages (1) - à gauche

Méthode

- on enlève le bit de gauche,
- on décale les autres d'une position à gauche et
- on rajoute un 0 à droite.

Exemples

$$\begin{array}{ll} 00110011_2 & = 51_{10} \\ \downarrow & \\ 01100110_2 & = 102_{10} \end{array} \qquad \begin{array}{ll} 11110011_2 & = -13_{10} \\ \downarrow & \\ 11100110_2 & = -26_{10} \end{array}$$

Cela réalise une multiplication par 2.

Décalages (1bis) - dépassement

Il y a dépassement pour un décalage à gauche quand :

- les deux bits de gauche du nombre de départ sont différents,
- c'est à dire quand le bit de signe du résultat est différent de celui du nombre de départ.

Exemples

$00110 = 6_{10} \quad \rightarrow \quad 01100 = 12_{10}$ OK

$01100 = 12_{10} \quad \rightarrow \quad 11000 = -8_{10}$ dépassement

$10111 = -9_{10} \quad \rightarrow \quad 01110 = 14_{10}$ dépassement

$11101 = -3_{10} \quad \rightarrow \quad 11010 = -6_{10}$ OK

Décalages (2) - à droite

Méthode

- on enlève le bit en position 0 (le plus à droite),
- on décale les autres d'une position vers la droite et
- on recopie le bit de signe en position $n-1$ (à gauche).

Exemples

$$\begin{array}{ccc} 0110_2 = 6_{10} & 0101_2 = 5_{10} & 110010_2 = -14_{10} \\ \downarrow & \downarrow & \downarrow \\ 0011_2 = 3_{10} & 0010_2 = 2_{10} & 111001_2 = -7_{10} \end{array}$$

Cela revient à une division entière par deux.

Décalages (2bis) - les arrondis

- Pas de problème de dépassement,
- par contre on fait une division entière,
- arrondi à l'entier inférieur.

Attention aux arrondis

Qu'obtient-on si on décale n fois à droite un nombre sur n bits ?

- s'il est positif : $00...00_2 = 0$
- s'il est négatif : $11...11_2 = -1$! ! !

C'est normal : si on divise -1 par deux et qu'on arrondit à l'entier inférieur, on retrouve -1 !

En assembleur on verra deux décalages à droite :

- un logique (met un 0 à gauche) : entiers non signés, et
- un arithmétique (recopie du bit de signe) : entiers signés.

Addition

Méthode

- comme à l'école
- avec des retenues
- la différence ? 1+1= 0 et on retient 1

Exemple

$$
\begin{array}{rccccccl}
 & 0 & \overset{1}{0} & \overset{1}{1} & 1 & 0 & = & 6_{10} \\
+ & 0 & 0 & 1 & 1 & 1 & = & 7_{10} \\
\hline
 & 0 & 1 & 1 & 0 & 1 & = & 13_{10}
\end{array}
$$

Addition (2) - dépassement

Exemples sur 5 bits

Que donne l'algorithme d'addition sur 5 bits ?

$$\begin{array}{r} 1\,1\,1\,1\,1 \\ +\,1\,1\,1\,1\,1 \\ \hline 1\,1\,1\,1\,0 \end{array} \qquad \begin{array}{r} 0\,1\,1\,0\,1 \\ +\,0\,1\,0\,0\,0 \\ \hline 1\,0\,1\,0\,1 \end{array} \qquad \begin{array}{r} 1\,0\,1\,0\,1 \\ +\,1\,1\,0\,0\,0 \\ \hline 0\,1\,1\,0\,1 \end{array}$$

$$\begin{array}{r} 1\,0\,1\,0\,1 \\ +\,0\,1\,0\,0\,0 \\ \hline 1\,1\,1\,0\,1 \end{array}$$

Dépassement ? non,oui oui et non.

- Il peut y avoir dépassement lorsque les deux nombres à ajouter sont de signe identique.
- Le dépassement se voit au résultat de signe différent
  - par exemple $A > 0$, $B > 0$ et résultat < 0.

Opposé

But : calculer l'écriture (en compl. à deux) de $-A$ à partir de A.

Méthode

- On prend le complément booléen bit à bit (changer les 0 en 1 et les 1 en 0),
- on ajoute 1 au résultat obtenu.

Exemple

$$\begin{aligned} \mathrm{B} = 6 = &\quad 00110_2 \\ &\quad \ \downarrow \\ \bar{B} = &\quad 11001_2 \\ &\quad \ \downarrow \\ -B = \bar{B} + 1 = &\quad 11010_2 \end{aligned}$$

Vérification : 11010_2 = -16+8+2 = -6

Opposé (2) - dépassement

- Il y a dépassement lorsque A représentable mais pas -A
- Une seule valeur dans ce cas
- A $= -2^{n-1}$ sur n bits

Soustraction

Méthode

Comment calcule-t-on A-B ?

- A-B = A+ (-B)
- on calcule l'opposé -B,
- puis on l'ajoute à A.

$$A - B = A + \bar{B} + 1$$

Exemple

Calculer 5 - 12 (sur 6 bits) :

$$\begin{aligned} B &= 001100 \\ \bar{B} &= 110011 \\ -B &= 110100 \end{aligned} \qquad \begin{array}{r} 000101 \\ +110100 \\ \hline 111001 \end{array}$$

Résultat : -7

Soustraction (2) - dépassement

On calcule A-B. Quand y a-t-il dépassement ?

- Il peut y avoir dépassement lorsque les deux nombres à soustraire sont de signe différent.
- Le dépassement se voit au résultat de même signe que B
  - par exemple $A > 0$, $B < 0$ et résultat < 0.
  - ou encore $A < 0$, $B > 0$ et résultat > 0.

Retour sur les avantages

- un seul algorithme d'addition, pas de souci de signe
- la soustraction se fait en utilisant l'addition
- on a bien une seule représentation de 0
    - pour calculer l'opposé de 00000 ...
    - on prend le complément booléen bit à bit 11111 ...
    - puis on ajoute 1 ce qui donne 00000.

Et la multiplication ?

Méthode

Comment calcule-t-on A×B ?

- comme à l'école
- avec des décalages et des additions
- l'avantage ? on ne fait que des multiplications par 0 ou 1 !

$$
\begin{array}{rrrrrrrrr}
 & & & & 0 & 0 & 1 & 0 & 1 \\
 & & & \times & 0 & 0 & 0 & 1 & 1 \\
\hline
 & & & & 0 & 0 & 1 & 0 & 1 \\
 & & & 0 & 0 & 1 & 0 & 1 & . \\
 & & 0 & 0 & 0 & 0 & 0 & . & . \\
 & 0 & 0 & 0 & 0 & 0 & . & . & . \\
0 & 0 & 0 & 0 & 0 & . & . & . & . \\
\hline
 & & & & 0 & 1 & 1 & 1 & 1
\end{array}
$$

Cet algorithme sera vu/programmé en TD/TP

Pourquoi "décalage à gauche" = "multiplication par deux" ?

Lorsqu'il n'y a pas de dépassement bien sûr

Pas de dépassement = deux bits de gauche ont la même valeur.
Si A est positif :

- $A = 00a_{n-3}...a_1a_0$
- Quand on décale on a $0a_{n-3}...a_1a_00$
- on passe de $A = \sum_{i=0}^{n-3} a_i \times 2^i$ à $\sum_{i=0}^{n-3} a_i \times 2^{i+1}$=2A.

Si A est négatif :

- $A = (11a_{n-3}...a_1a_0)_2 = -2^{n-1} + 2^{n-2} + \sum_{i=0}^{n-3} a_i \times 2^i$.
- soit $A = -2^{n-2} + \sum_{i=0}^{n-3} a_i \times 2^i$
- quand on décale on a $A' = (1a_{n-3}...a_1a_00)_2 = -2^{n-1} + \sum_{i=0}^{n-3} a_i \times 2^{i+1}$.
- $A' = 2 \times (-2^{n-2} + \sum_{i=0}^{n-3} a_i \times 2^i) = 2A$

Pourquoi "décalage à droite" = "division par deux" ?

Division entière : arrondi à l'entier inférieur

- On va traiter le cas où A est pair
- l'écriture binaire a un 0 à droite
- le cas impair revient au même modulo l'arrondi

Si A est positif :

- on passe de $A = (0a_{n-2}...a_1 0)_2 = \sum_{i=1}^{n-2} a_i \times 2^i$
- à $(00a_{n-2}...a_1)_2 = \sum_{i=1}^{n-2} a_i \times 2^{i-1} = A/2$

Si A est négatif,

- on passe de $A = (1a_{n-2}...a_1 0)_2 = -2^{n-1} + \sum_{i=1}^{n-2} a_i \times 2^i$
- à $A' = (11a_{n-2}...a_1)_2 = -2^{n-1} + 2^{n-2} + \sum_{i=1}^{n-2} a_i \times 2^{i-1} = -2^{n-2} + \sum_{i=1}^{n-2} a_i \times 2^{i-1} = 1/2 \times (-2^{n-1} + \sum_{i=1}^{n-2} a_i \times 2^i) = A/2$

Pourquoi "$\bar{B}+1$"="$-B$" ?

- Il suffit de montrer que : $B + (\bar{B} + 1) = 0$.
- $\bar{B}$ est l'inverse bit à bit de B
- $\bar{B}$ a des 0 où B a des 1 et réciproquement
- et donc pour tout B, $B + \bar{B} = 11....11$
- et $11....11 + 00...01 = 00...00$