

## NUMÉRATION LOGIQUE


C4 : codage des entiers  
 Nicole VINCENT

## Codage binaire de l'information

- L'architecture actuelle des ordinateurs nécessite une représentation en binaire de toute information :  
 $\{0, 1\}$ , {faux, vrai}, {éteint, allumé}, {noir, blanc}, ...
- En Chine (premier millénaire av. JC), système binaire lié au {Yin, Yang} ou {actif, passif}
- Leibniz en 1703 : représentation des nombres en binaire.
- Boole (1815-1864) crée une algèbre avec 2 valeurs numériques : 0 et 1  
 C'est l'**algèbre de Boole** ou **calcul booléen**.

## Codage des nombres entiers en machine?

- Représenter les entiers relatifs
- Les opérations arithmétiques  $+$ ,  $-$ ,  $\times$  et  $/$  doivent être faciles à effectuer.
- Contrainte : architecture du matériel, la taille des **mots mémoire** est limitée : 16, 32, 64, ... bits.
- Objectif : représenter l'information de la façon compacte  
 Ne pas produire de dépassements de capacité (overflow)

 Mots mémoire de taille suffisamment grande

Problème des lanceurs Ariane 4 et 5

## Codage des entiers en binaire

- On va présenter quatre codages :
  - Codage binaire naturel signé
  - Décimal codé binaire
  - Codage "complément à deux"
  - Codage "complément à un"

## Codage binaire naturel signé

- On fixe la longueur des mots
- Le bit de poids fort est réservé au signe

Exemple : sur 4 bits,  $+6$  est codé par 0110  
 $-6$  est codé par 1110

- |   |   |
|---|---|
| <ul style="list-style-type: none"> <li>• Avantages           <ul style="list-style-type: none"> <li>+ Codage/décodage très facile</li> <li>+ Représentation des entiers négatifs</li> </ul> </li> </ul> | <ul style="list-style-type: none"> <li>• Inconvénients           <ul style="list-style-type: none"> <li>- Il y a deux représentations de 0</li> <li>- Les opérations arithmétiques ne sont pas faciles</li> </ul> </li> </ul> |
|---|---|

## Décimal codé binaire

Exemple : L'entier positif 19032  
 Ancien système, binary coded decimal (BCD)  $0_{10} = 0000$  ;  $1_{10} = 0001$  ; ... ;  
 chaque chiffre est codé en binaire sur 4 bits  $8_{10} = 1000$  ;  $9_{10} = 1001$   
 codé : 0001 1001 0000 0011 0010

- |   |   |
|---|---|
| <b>Avantages</b> <ul style="list-style-type: none"> <li>+ Codage/décodage facile</li> <li>+ Pas de limitation des grandeurs représentées</li> </ul> | <b>Inconvénients</b> <ul style="list-style-type: none"> <li>- On gâche de l'espace : 6 combinaisons sont non utilisées</li> <li>- Opérations arithmétiques compliquées</li> </ul> |
|---|---|

Représentations utilisées dans des systèmes de bases de données (SGBD)

- Stocker de grands nombres
- Utilisées dans des systèmes basiques avec affichage digital

## Complément à 2 : CA2

- On fixe le nombre d'octets pour stocker des nombres et faire des opérations : 1,2,3,4,... octets
- Quantité d'informations que l'on peut coder dans ce système
  - 8 bits (1 octet) : On peut coder  $2^8 = 256$  nombres
  - 16 bits (2 octets) : On peut coder  $2^{16} = 65536$  nombres
  - ...
- Codage des nombres positifs et négatifs :
  - 1 octet : 256 nombres  $\rightarrow$  de -128 à +127
  - 2 octets : 512 nombres  $\rightarrow$  de -32768 à +32767
  - ...

## Codage des entiers relatifs par complément à 2

- Les entiers positifs : séquence de bits commençant par 0 et écriture binaire signée
- Les entiers négatifs : séquence commençant par 1
  - n est codé par le codage binaire de  $2^k - n$
- Le 1<sup>er</sup> bit est naturellement appelé "bit de signe"

0 0 0 0 0 0 0 0  $\rightarrow$  0  
 0 0 0 0 0 0 0 1  $\rightarrow$  1  
 ...  
 0 1 1 1 1 1 1 1  $\rightarrow$  127

## Codage des entiers relatifs par complément à 2

- Les entiers négatifs : séquence commençant par 1
    - n est codé par le codage binaire de  $2^k - n$
- Sur 8 bits
- 1 :  $(2^8 - 1)_{10} = (100000000)_2 - 1 = (11111111)_2$   
 -2 :  $(2^8 - 2)_{10} = (11111110)_2$
- 0 0 0 0 0 0 0 0  $\rightarrow$  0  
 0 0 0 0 0 0 0 1  $\rightarrow$  1  
 ...  
 0 1 1 1 1 1 1 1  $\rightarrow$  127  
 1 0 0 0 0 0 0 0  $\rightarrow$  -128  
 ...  
 1 1 1 1 1 1 1 0  $\rightarrow$  -2  
 1 1 1 1 1 1 1 1  $\rightarrow$  -1
- 127 :  $(2^8 - 127)_{10} = (2^8 - 2^7 + 1) = (10000001)_2$
- |   |                   |
|---|-------------------|
|   | 1 0 0 0 0 0 0 0 1 |
| - | 1 0 0 0 0 0 0 0 0 |
|   | 1 0 0 0 0 0 0 0 1 |

## Codage des entiers relatifs par complément à 2

- Les entiers négatifs : séquence commençant par 1
  - n est codé par le codage binaire de  $2^k - n$
- Réciproquement sur 4 bits  
 si le code CA2  $m = (s_3 s_2 s_1 s_0)_2$  avec  $s_3 = 1$  m représente un nombre négatif  $m = -n$   
 $m$  est le code de  $(2^4 - n)_{10} = (2^4 + nb)_{10} = (m)_{10}$   
 $(nb)_{10} = (m)_{10} - 2^4$

## Complément à deux sur k bits

- Sur k bits, on code les entiers de  $\{-2^{k-1}, \dots, 2^{k-1}-1\}$
- Le complément à deux du complément à deux d'un entier  $n \leq 2^{k-1}$  est l'entier lui-même
 
$$2^k - (2^k - n) = n$$
- Le complément à deux de zéro est zéro  
 $2^k - 0 = 2^k$ , restriction à la taille k bits fixée  $k=8$  1 0000 0000
- Si l'entier n est codé sur k bits en complément à deux :  $n = s_{k-1} s_{k-2} \dots s_3 s_2 s_1 s_0$

$$(n)_{10} = -s_{k-1} 2^{k-1} + \sum_{i=0}^{k-2} s_i 2^i$$

En complément à deux, le bit de signe  $s_{k-1}$  a comme poids  $-2^{k-1}$

## Algorithme de codage

- Soit un nombre  $m \in \mathbb{Z}$   $m \in \{-2^{k-1}, \dots, 2^{k-1}-1\}$ ,
- Si  $m \in \{0, \dots, 2^{k-1}-1\}$  positif, écriture binaire et on rajoute des 0 devant pour former k bits.
  - Si  $m = -n$  avec  $n \in \{0, \dots, 2^{k-1}-1\}$ , il faut "calculer"  $2^k - n$  en binaire
- Remarque :  $[1111111111] = \sum_{i=0}^{k-1} 2^i$  et  $\sum_{i=0}^{k-1} 2^i + 1 = 2^k$
- $n + x = [1111111111]$  ; x s'obtient en inversant les bits  $n + x + 1 = 2^k$
- Le code est  $2^k - n$  Soit x + 1
- Ecrire n en binaire sur k bits
  - Inverser bit à bit
  - Ajouter 1 en binaire

## Codage exemple

$m = 27$  en CA2s

$27 = 16 + 8 + 2 + 1 = (11011)_2$ , donc le codage est  $[00011011]$

$m = -27$  en CA2s

- Ecrire  $n$  en binaire sur  $k$  bits  $27 = (00011011)_2$
- Inverser bit à bit  $[00011011]$  devient  $[11100100]$
- Ajouter 1 en binaire  $[11100100]$  devient  $[11100101]$

Codage :  $[11100101]$

12

## Décodage

- Si le code commence par 0, on fait la conversion binaire  $\rightarrow$  décimal

Exemple :  $[00011011]$  on a le code de  $27 = (11011)_2$

- Si le code commence par 1, nombre du type  $-n$ . Pour trouver  $n$ , deux méthodes :

Exemple :  $[11100101]$

- $(k-1)$  bits à droite du 1  $(1100101)$
- On RETRANCHE 1  $(1100100)$
- On inverse bit à bit  $(0011011)$
- On convertit du binaire au décimal  $16+8+2+1$
- $(k-1)$  bits à droite du 1  $(1100101)$
- On inverse bit à bit  $(0011010)$
- On AJOUTE 1  $(11011)$
- On convertit du binaire au décimal  $n=27$

On a le code de -27

## Complément à 2

- Avantages et inconvénients
  - + Codage/décodage facile
  - + Représentation unique de zéro
  - + Opérations arithmétiques faciles
- Taille mémoire fixée.

13

## Qu'est-ce qu'un bon système d'addition?

- Facilement compatible avec les opérations, addition et soustraction
- Taille fixée une addition de nombres peut sortir de la portée  
 $110 + 80 = 190$ , ou  $-110 - 80 = -190$ ,  
 l'addition de deux nombres en CA2s peut sortir de l'intervalle  $\{-127, \dots, 128\}$
- Addition de deux nombres de signes opposés codés en CA2s est codable en CA2s  
 $110 + (-80) = 30$  et  $-110 + 80 = -30$
- Addition de deux nombres de même signe, le système doit  
 - donner le résultat si il est dans le bon intervalle  
 $110 + 24 = 154$ ;  $-110 - 24 = -154$   
 - indiquer le problème

14

## Complément à deux : addition, signes opposés

Le résultat est toujours représentable

Exemple sur un octet,  $k = 8$  :  $+63 + (-63)$

	0	0	1	1	1	1	1	1
+	1	1	0	0	0	0	0	1
retenue	1	1	1	1	1	1	1	1
	1	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0

On ne tient pas compte de la retenue.

On a effectué  $+63 + (256 - 63) - 2^8$

15

## Complément à deux : addition, signes opposés

Exemples sur un octet,  $k = 8$

$s = -63 + 28$   $(63)_{10} = (1111111)_2$

-63	1	1	0	0	0	0	0	1
+28	0	0	0	1	1	1	0	0
retenue	1	1	0	1	1	1	0	1
	1	1	0	1	1	1	0	1
	0	1	0	0	0	1	1	1

Il n'y a pas de retenue, on lit le résultat directement

$$s = -(1+2+32) = -35$$

16

$+63 - 28$

'inverse' de 28 :  $11100011$

+63	0	0	1	1	1	1	1	1
-28	1	1	1	0	0	1	0	0
retenue	1	1	1	1	1	1	1	1
	1	0	0	1	0	0	0	1
	1	0	0	0	1	1	1	1

Il y a une retenue, on la néglige, i.e. on retranche  $2^8$

$$s = 1+2+32 = +35$$

## Complément à deux : addition, même signe

- L'addition de deux nombres de même signe peut donner lieu à un **dépassement de capacité**
- Cas de deux entiers de signe positif  
On a un dépassement de capacité quand la retenue est distincte du bit de signe

+35	0	0	1	0	0	1	1
+65	0	1	0	0	0	0	1
retenue					1	1	
	0	1	1	0	0	1	0

$$S=4+32+64=100$$

Dépassement de capacité

## Complément à deux : addition, même signe

- Cas de deux entiers de signe négatif.  
On a toujours une retenue, que l'on oublie.  
En effet, on calcule  $(2^k - n_1) + (2^k - n_2) - 2^k$ .  
On a un dépassement de capacité quand la retenue est distincte du bit de signe).

-35	1	1	0	1	1	1	0	1
-65	1	0	1	1	1	1	1	1
retenue	1	1	1	1	1	1	1	1
	1	1	0	0	1	1	1	0

$$1100011+1=1100100=4+32+64=100$$

Dépassement de capacité

$$S=-100$$

## Résumé pour l'addition en "complément à deux"

- 2 nombres de signes opposés
  - Le résultat est représentable avec le nombre de bits fixés, pas de dépassement de capacité
  - S'il y a une retenue, on l'oublie
  - On lit directement le résultat codé en CA2
- 2 nombres de même signe
  - Il y a dépassement de capacité si la retenue est distincte du bit de signe
  - S'il y a une retenue on l'oublie si on a vérifié qu'il n'y avait pas overflow
  - On lit directement le résultat codé en CA2
- Ce codage est donc souvent choisi en pratique

## Complément à 1 CA1<sub>k</sub>

- Entiers positifs : commencent par 0  $\Rightarrow$  code = écriture binaire
- Entiers négatifs : commencent par 1

Sur 8 bits

$$00000000 \rightarrow 0$$

$$00000001 \rightarrow 1$$

pour les nombres négatifs  
Inversion bit à bit

$$01111111 \rightarrow 127$$

$$10000000 \rightarrow -127$$

$$11111110 \rightarrow -1$$

## Complément à 1

- Avantages
  - Plus facile de coder/décoder les nombres négatifs : On inverse simplement bit à bit
- Inconvénients
  - Opérations arithmétiques plus compliquées
  - Détection de l'overflow plus compliquée
  - 2 manières de coder le "0", donc 2 tests pour voir si un nombre est nul
  - On code un nombre de moins : -127  $\rightarrow$  127 au lieu de -128  $\rightarrow$  127

## Conclusion

- La représentation des nombres entiers est limitée par la taille du mot mémoire qui leur est affectée
- Le code le plus souvent utilisé est le code complément à deux. Il n'a qu'une seule représentation du zéro, les opérations arithmétiques et la détection de dépassement de capacité sont faciles à effectuer.
- Dans tous les cas et en fonction des architectures d'ordinateurs il y aura toujours des opérations dont le résultat n'est pas représentable. Sans précautions, elles engendrent des résultats aberrants ou empêchent la poursuite des calculs.