Architecture des ordinateurs : Codage binaire et hexadécimal Arithmétique des processeurs

(4TIN304U)

F. Pellegrini
Université de Bordeaux

Notation positionnelle (1)

- La notation positionnelle représente un nombre sous la forme d'une séquence de chiffres
  - Chaque chiffre représente le multiple d'une puissance d'un nombre appelé base
  - Les puissances croissent à partir de zéro, de la droite vers la gauche
- Nous utilisons couramment la base 10, avec les chiffres de « 0 » à « 9 »
  - $123 = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$

Notation positionnelle (2)

- La notation positionnelle présente de nombreux avantages :
  - Utilise toujours les mêmes chiffres
    - À la différence de l'écriture en chiffres romains : I = 1, V = 5, X = 10, L = 50, C = 100, D = 500, M = 1000, ...
  - Permet d'écrire facilement de très grands nombres

Notation binaire (1)

- Les ordinateurs encodent l’information en utilisant les états de systèmes physiques
- Les systèmes physiques les plus simples possèdent deux états :
  - Tension électrique / pas de tension
  - Orientation nord ou sud d’un aimant
  - Etc.
- La notation binaire est naturelle pour représenter les états de la mémoire d'un ordinateur

Notation binaire (2)

- La notation binaire utilise la base 2 et les chiffres « 0 » et « 1 »
  - $101011 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
- Pour lever toute ambigüité, on indique parfois la base (en décimal) à la fin d'un nombre
  - $101011_2 \neq 101011_{10}$
  - $101011_2 = 43_{10}$

Numération en binaire (1)

- En binaire, on compte comme dans toute autre base
- Lorsque, dans une colonne, on est arrivé au plus grand chiffre :
  - On remet la colonne à zéro
  - On incrémente la puissance supérieure

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

Numération en binaire (2)

- On énumère les nombres binaires en appliquant ce principe, en partant de 0

0	100	1000	1100
1	101	1001	1101
10	110	1010	1110
11	111	1011	1111

- Etc.

Notation hexadécimale (1)

- La notation binaire est fastidieuse !
  - Même les nombres les plus courants peuvent être longs à écrire
- Il faut trouver une notation plus économe en place

Notation hexadécimale (2)

- Il faut trouver une base qui :
  - Se convertisse facilement en une écriture binaire
    - Donc une puissance de 2
  - Soit plus grande que 2, mais pas trop grande
    - Retenir 32 chiffres ou plus serait plutôt pénible...
  - Permette d'écrire facilement des octets
    - Donc dont le $\log_2$ soit un diviseur de 8 : base 4 ou 16
- Choix : base 16, ou « codage hexadécimal »
  - $16 = 2^4$, $\log_2(16) = 4 = 8 / 2$

Notation hexadécimale (3)

- Les chiffres hexadécimaux vont de « 0 » à « 9 », puis de « A » à « F »

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

  - $B0A_{16} = 11 \times 16^2 + 0 \times 16^1 + 10 \times 16^0 = 2826_{10}$

- Dans de nombreux langages, on préfixe les nombres hexadécimaux par « 0x… » ou « 0X… »

Arithmétique entière (1)

- Avec n bits, on dispose de 2^n combinaisons possibles, qui permettent de représenter les nombres entiers naturels de 0 à $2^n - 1$
- Les règles classiques de l'addition s'appliquent

		1	1		1				
	0	0	1	1	0	1	0	1	53
+	1	0	0	1	0	1	0	0	148
	1	1	0	0	1	0	0	1	201

Arithmétique entière (2)

- Pour représenter des nombres négatifs, on peut transformer le bit de poids le plus fort en bit de signe, pour coder 2^{n-1} nombres entiers positifs et 2^{n-1} nombres entiers négatifs
- Lorsque le bit de signe est à 0, on considère que le nombre est positif, et on code les entiers naturels de 0 à $2^{n-1} - 1$

Bit de signe

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1
0	1	1	1	1	1	1	1	127

Arithmétique entière (3)

- Lorsque le bit de signe est à 1, on considère que le nombre est négatif
- Plusieurs moyens sont envisageables pour coder les entiers négatifs avec les (n-1) bits restants

Arithmétique entière (4)

- Codage des nombres négatifs au format naturel
  - Même codage des n-1 bits restants que pour les nombres positifs

1	0	0	0	0	0	0	1	-1

  - Problèmes :
    - On a deux zéros (gaspillage d'une configuration)

0	0	0	0	0	0	0	0	+0
1	0	0	0	0	0	0	0	-0

    - Nécessité d'un circuit spécifique pour la soustraction

Arithmétique entière (5)

- Pour éviter les problèmes du codage précédent, il faut un codage des nombres négatifs tel que :
  - Le bit de signe soit à 1
  - Il n'y ait qu'un seul zéro
  - On puisse utiliser la méthode d'addition standard pour additionner nombres positifs et négatifs

Arithmétique entière (6)

- En particulier, avec les contraintes précédentes, on veut :

	0	0	0	0	0	0	0	1	1
+	1	.	.	.	.	.	.	.	-1
	0	0	0	0	0	0	0	0	0

- La seule solution possible est donc :

1	1	1	1	1	1	1	1	-1

qui génère une retenue en sortie (« *carry* »), perdue car elle ne peut pas être stockée

Arithmétique entière (7)

- Pour représenter l'opposé d'un nombre entier, on prend son complément bit à bit, auquel on ajoute 1
- Cette notation est appelée « complément à deux »

	0	0	0	0	0	1	0	1	5
Complément bit à bit	1	1	1	1	1	0	1	0	
+1	1	1	1	1	1	0	1	1	-5

Arithmétique entière (8)

- Ajouter un nombre à son opposé en complément à deux donne toujours zéro car :
  - Ajouter un nombre à son complément bit à bit donne toujours un vecteur constitué uniquement de 1
  - Ajouter 1 à ce vecteur donne un vecteur constitué uniquement de 0, après perte de la retenue

		0	0	0	0	0	1	0	1	5
+		1	1	1	1	1	0	1	1	-5
	~~1~~	0	0	0	0	0	0	0	0	0

Arithmétique entière (9)

- Ce principe s'étend à toute addition entre entiers signés

	0	0	1	0	1	1	1	0	46
+	1	1	0	0	1	0	1	1	-53
	1	1	1	1	1	0	0	1	-7

Arithmétique entière (10)

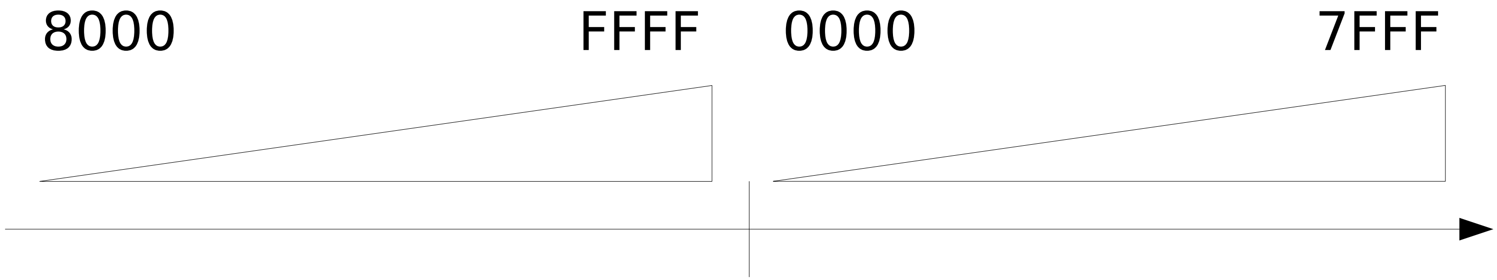
- Principales valeurs en complément à deux pour un nombre sur 8 bits

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1
1	1	1	1	1	1	1	1	-1
0	1	1	1	1	1	1	1	127
1	0	0	0	0	0	0	0	-128

Le domaine de validité d'un nombre entier signé sur n bits est donc $[-2^{n-1}, 2^{n-1}-1]$

Arithmétique entière (11)

- Les nombres signés sont organisés de façon croissante en deux sous-blocs considérés de façon non signée
- Par exemple, sur 16 bits :



Arithmétique entière (12)

- Il y a débordement arithmétique (« *overflow* ») lorsque le résultat attendu n'est pas représentable dans le système choisi

	0	1	1	0	1	1	1	0	110
+	0	0	1	0	1	0	1	1	43
	1	0	0	1	1	0	0	1	-103 (et non 153)

- Il peut y avoir débordement sans perte de retenue, ou perte de retenue sans débordement

Arithmétique flottante (1)

- Dans de nombreux calculs, il n'est pas possible d'utiliser des nombres entiers, et le domaine des nombres manipulés est très grand
  - Masse de l'électron : 9×10^{-28} grammes
  - Masse du soleil : 2×10^{33} grammes
  - Le domaine dépasse les 10^{60}
- Nécessité de trouver un format adapté pour représenter de tels nombres avec un petit nombre de bits (32 ou 64 en pratique)

Arithmétique flottante (2)

- Comme le domaine à représenter est infini, il faut l'échantillonner de façon représentative
- On représentera donc un nombre à virgule sous la forme scientifique

$$n = f \times 10^{e}$$

  - f : fraction, ou mantisse
  - e : exposant, sous la forme d'un entier signé

Arithmétique flottante (3)

- Par exemple :
  - $3.14 = 0.314 \times 10^{1} = 3.140 \times 10^{0}$
  - $0.00001 = 0.01 \times 10^{-3} = 1.000 \times 10^{-5}$
- Le domaine dépend de la taille maximale de l'exposant
- La précision dépend du nombre maximal de chiffres significatifs de la mantisse

Arithmétique flottante (4)

- Il existe plusieurs représentations possibles du même nombre
- On privilégie toujours la forme normalisée, telle que le premier chiffre de la mantisse soit significatif, c'est-à-dire différent de zéro
- Cette forme maximise l'utilisation des chiffres significatifs de la mantisse, et donc la précision
  - $f = 0$ ou $f \in [1.0 \,;\, 10.0\,[$

$$f = \boxed{10^0} \,.\, \boxed{10^{-1}}\boxed{10^{-2}}\boxed{10^{-3}}\boxed{10^{-4}}\boxed{10^{-5}}\boxed{10^{-6}} \ldots$$

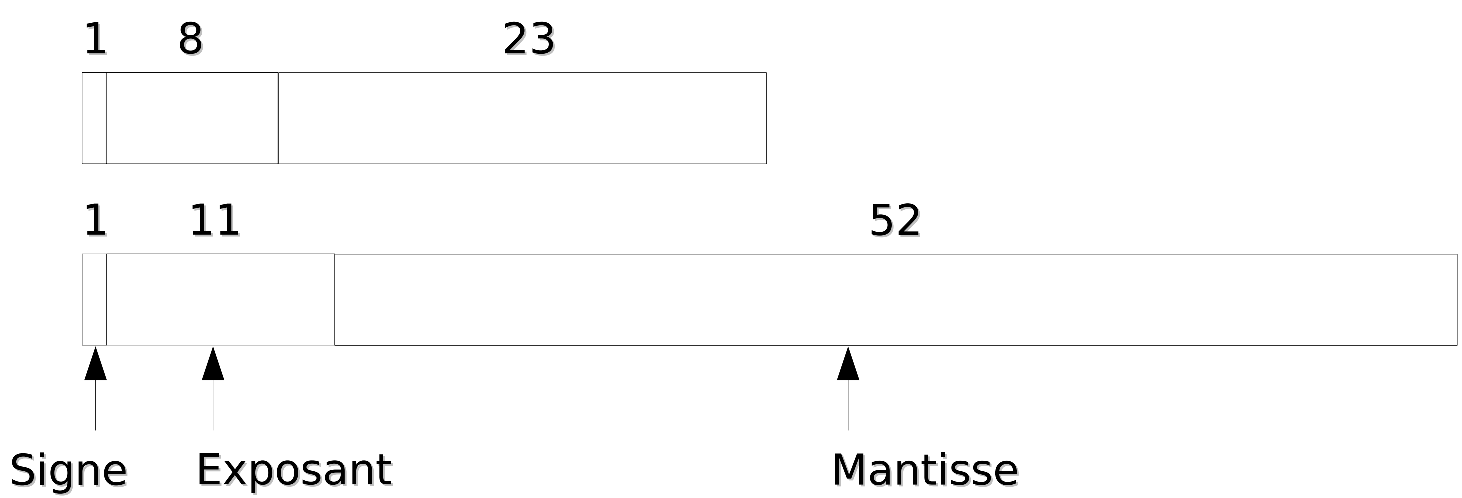
Norme IEEE 754 (1)

- Ce standard définit trois formats de nombres à virgule flottante
  - Simple précision (32 bits)
  - Double précision (64 bits)
  - Précision étendue (80 bits)
    - Utilisé pour stocker les résultats intermédiaires de calculs au sein des coprocesseurs arithmétiques
- Utilise la base 2 pour les mantisses et le codage par excès pour les exposants

$$f = \boxed{2^0} \, . \, \boxed{2^{-1}}\boxed{2^{-2}}\boxed{2^{-3}}\boxed{2^{-4}}\boxed{2^{-5}}\boxed{2^{-6}} \ldots$$

Norme IEEE 754 (2)

- Format des nombres
  - Commencent par un bit de signe (0 : positif)
  - Exposants définis par excès (127 pour la simple précision et 1023 pour la double précision)
    - Valeurs minimum (0) et maximum (255 ou 2047) réservées pour des codages spéciaux



Norme IEEE 754 (3)

- Une mantisse normalisée est constituée d'un chiffre 1, de la virgule, et du reste de la mantisse
- Comme le 1 de tête doit toujours être présent, il n'est pas nécessaire de le stocker
- La pseudo-mantisse de la norme IEEE 754 est donc constituée implicitement d'un 1 et de la virgule, suivis des 23 ou 52 bits effectifs
  - On parle aussi de « significande »
  - Le significande code des valeurs dans [1;2[

Norme IEEE 754 (4)

- Exemple : représentation en simple précision du nombre 0.75_{10} :
  - $0.75_{10} = 1.1_2 \times 2^{-1}$
  - Le significande est donc : .1000...0
  - L'exposant est donc : $-1 + 127 = 126 = 01111110_2$
- Le codage du nombre sur 32 bits est donc :

0	01111110	10000000000000000000000

$3F400000_{16}$

Norme IEEE 754 (5)

- Un des problèmes principaux avec les nombres à virgule flottante est la gestion des erreurs numériques telles que :
  - Débordements (« *overflow* ») : le nombre est trop grand pour être représenté
  - Débordements inférieurs (« *underflow* ») : le nombre est trop petit pour être représenté
  - Résultat qui n'est pas un nombre (« *not-a-number* », ou NaN), comme par exemple le résultat d'une division par 0

Norme IEEE 754 (6)

- En plus des nombres normalisés classiques, la norme IEEE 754 définit donc quatre autres types numériques :
  - Not-a-number : résultat impossible
  - Infini : infinis positif et négatif, pour le débordement
  - Zéro : zéros positif et négatif, pour le débordement inférieur (« *underflow* »)
  - Nombres dénormalisés, pour les valeurs trop petites pour être représentables de façon normalisée

Norme IEEE 754 (7)

- Tableau récapitulatif

	Simple précision	Double précision
Taille totale	32	64
Bit de signe	1	1
Bits d'exposant	8	11
Bits de significande	23	52
Domaine décimal	$\simeq 10^{-38}$ à 10^{+38}	$\simeq 10^{-308}$ à 10^{+308}