Introduction à l'architecture des ordinateurs

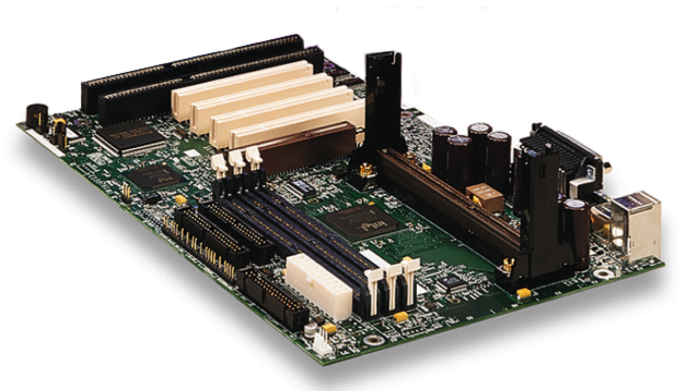
Bernard GIRAU

LORIA - bureau C042

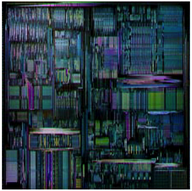
contact : Bernard.Girau@loria.fr

Introduction à l'architecture des ordinateurs
Objectifs

Comprendre comment



fonctionne.

en insistant sur 

Organisation du cours

- 18h CM
- 15h TD
- 12h TP
- UE 303 (Architecture des ordinateurs - Introduction aux bases de données), coeff 0,5
    - note E1 (projet, TP noté, ou ...) : coeff 0,2
    - note E2 (examen écrit, ou ...) : coeff 0,3

- Chap. 0 : Arithmétique des ordinateurs (rappels)
- Chap. 1 : Logique combinatoire (rappels) et séquentielle
- Chap. 2 : Structure interne des ordinateurs
- Chap. 3 : Les instructions machine et le langage d'assemblage
- Chap. 4 : Chemins de données et unité de contrôle
- Chap. 5 : Mémoire hiérarchique
- Chap. 6 : Pipeline d'instructions

0.1 - Introduction

Généralités

- codage des informations sur ordinateurs : entiers, réels, textes, sons, images, vidéos, ...
- base commune : codage des entiers

codage binaire : système de position

- cf nombres décimaux
- permet manipulations algorithmiques
- 2 chiffres (binaires)

0.2 - Codage des entiers

0.2 - Codage des entiers

Entiers naturels

Décomposition en base $B > 1$

$$x = x_0 + x_1.B + x_2.B^2 + x_3.B^3 + \dots$$

existence et unicité de la représentation si les x_i sont entre 0 et $B - 1$

notation habituelle : chiffres de poids faible (x_0) à droite

Exemple

Base 10 vers base 2 :
$1234_{10} = 10011010010_2$

0.2 - Codage des entiers

Entiers naturels

Binaire/octal/hexadécimal

- $B = 2$ / $B = 8$ / $B = 16$
- chiffres hexadécimaux : 0, 1, 2, . . . , 9, A, B, C, D, E, F
- on passe du binaire à l'octal/hexadécimal et regroupant les chiffres par 3/4 à partir des poids faibles
- utilisé pour écrire de manière plus compacte et lisible des contenus binaires

Exemple

```
1234=0b10011010010
=0o(.10)(011)(010)(010)=0o2322
=0x(.100)(1101)(0010)=0x4D2
```

Exercices

1. Convertir 2009_{10} en binaire, en hexadécimal et en octal.
2. Convertir 0x0053 en décimal en passant par le binaire.
3. Convertir 0b0000001001010101 en décimal en passant par l'hexadécimal.
4. Ecrire l'algorithme de codage en base 2 d'un entier positif.
5. Ecrire l'algorithme de décodage d'un entier écrit en base 2, les différents chiffres binaires étant stockés dans un tableau.

Addition en base 2

Propagation de retenue c_i (poids faibles en tête) :

- $c_0 = 0$
- calcul récurrent : trouver c_{i+1} et s_i dans $\{0, 1\}$ tels que $a_i + b_i + c_i = 2c_{i+1} + s_i$ (FA : full adder)

Exemple

$1234 + 567 = 1801$

			1	1	1	1		1	1		
	1	0	0	1	1	0	1	0	0	1	0
+		1	0	0	0	1	1	0	1	1	1
=	1	1	1	0	0	0	0	1	0	0	1

Taille

entier de taille n + entier de taille n = entier de taille $n+1$
$\Longrightarrow$ dépassements de capacité éventuels (overflow)

Algorithme (addition)

c $\leftarrow$ 0
pour i de 0 à $n-1$
  $(s_i, c) \leftarrow FA(a_i,b_i,c)$
fpour

Full Adder

s_i : parité de $a_i + b_i$+c
c : $a_i + b_i$+c ≥ 2 ?

Exercice

- Additionner 1000101 et 111, puis 1011111 et 10011.

Soustraction en base 2

Propagation de retenue c_i (poids faibles en tête) :

- $c_0 = 0$
- calcul récurrent similaire : trouver c_{i+1} et s_i dans $\{0, 1\}$ tels que $2c_{i+1} + a_i - b_i - c_i = s_i$
- algorithme similaire

Exemple

$1234 - 567 = 667$

	1	0	0	1	1	0	1	0	0	1	0
–		1	0	0	0	1	1	0	1	1	1
	$_1$				$_1$	$_1$	$_1$	$_1$	$_1$	$_1$	
=		1	0	1	0	0	1	1	0	1	1

Exercices

- Soustraire 111 de 11000001, puis 1111 de 110000.

Multiplication en base 2

- addition et décalage (cf base 10)
- additions au fur et à mesure en base 2
- optimisation : on ignore les multiplications par 0
- taille : produit de deux nombres de taille $n \longrightarrow$ taille $2n$

Exemple

$26 \times 13 = 338$

				1	1	0	1	0
x					1	1	0	1
				1	1	0	1	0
		1	1	0	1	0	.	.
	1	0	0	0	0	0	1	0
	1	1	0	1	0	.	.	.
1	0	1	0	1	0	0	1	0

Algorithme (multiplication)

p $\leftarrow$ 0

m $\leftarrow$ a

pour i de 0 à $n-1$

  si $b_i = 1$ *alors* $p \leftarrow p{+}m$

  fsi

  $m \leftarrow m{<}{<}1$

fpour

Exercice

- Multiplier 111011 par 1001, puis 11010110 par 111.

Division en base 2

- soustraction et décalage (cf base 10)
- quotient et reste
- taille : quotient de la taille du 1er opérande, reste de la taille du 2ème

Exemple

$107/6 = 17$ et $107\%6 = 5$

1	1	0	1	0	1	1	1	1	0		
1	1	0									
		0	1	0	1	1	1	0	0	0	1
				1	0	1					

Algorithme (division)

q $\leftarrow$ 0
r $\leftarrow$ 0
pour i de $n-1$ à 0
    $r \leftarrow 2r{+}a_i$
    $q \leftarrow 2q$
    si $r \geq b$ alors
        $q \leftarrow q{+}1$
        $r \leftarrow r{-}b$
    fsi
fpour

Exercice

- Diviser 1000100 par 1001, puis 1010101 par 111.

Gestion du signe

- entiers non signés : de 0 à $2^n - 1$
- entiers signés : de $-(2^{n-1} - 1)$ à $2^{n-1} - 1$
- complément à deux : de -2^{n-1} à $2^{n-1} - 1$

$$x = -x_{n-1}.2^{n-1} + x_0 + x_1.2 + x_2.2^2 + \ldots + x_{n-2}.2^{n-2}$$

Cà2 : code les nombres négatifs par $2^n - |x|$ en non signé

$\longrightarrow$ nom complet : complément à deux puissance n

Cà2 : opposé

$-x$ s'obtient :

- en prenant le complémentaire de x, puis en ajoutant 1
- ou en gardant les bits de poids faible de x jusqu'au premier '1', puis en complémentant tous les suivants

Attention : il faut toujours savoir la taille du codage, sinon le codage Cà2 n'a pas de sens, et les opérateurs doivent travailler sur des opérandes de même taille

Exercices

On utilisera une représentation sur 16 bits en complément à 2.

1. Convertir 0b1111111111111111 en décimal.
2. Convertir -600_{10} en binaire, en hexadécimal et en octal.
3. Convertir 0xFF2C en décimal en passant par le binaire.

Cà2 : addition

- on ignore la dernière retenue
- détection de dépassement : les deux dernières retenues c_{n-1} et c_n doivent être identiques

Cà2 : soustraction

- on ajoute l'opposé
- détection de dépassement : idem

Cà2 : changement de taille de codage

Pour passer de n_1 à $n_2 > n_1$ bits, il suffit de rajouter des bits en poids forts tous égaux à x_{n_1-1}.
Pour passer de n_1 à $n_2 < n_1$ bits, il suffit de supprimer les $n_1 - n_2$ bits de poids forts à condition qu'il soient tous égaux à x_{n_2-1}.

Exercices

1. Additionner 87 et 45 en utilisant un codage en complément à 2 sur 1 octet.
2. Soustraire 27 de 45 en utilisant un codage en complément à 2 sur 2 octets.

0.3 - Codage des réels

0.3 - Codage des réels

Virgule fixe

Principes du codage en virgule fixe

- codage entier + position virgule
- si f bits en partie fractionnaire, alors $n - f$ bits en partie entière
- codage de x = codage entier de $x.2^f$ (Cà2 ou signé)

Exercices

1. Que représente 11.0111 dans un codage binaire en virgule fixe ?
2. En utilisant un codage binaire en virgule fixe sur 8 bits dont 5 en partie fractionnaire, représenter 5.25
3. Quelle est la précision de ce codage ?

Opérateurs

- opérateurs arithmétiques : utilisation des opérateurs entiers
- multiplication :
  - taille de la partie entière/fractionnaire = somme des tailles des parties entières/fractionnaires
  - troncature en fonction de la position de la virgule

Exemple 2

nombre de taille 2.6 $\times$ nombre de taille 2.6 $=$ nombre de taille 4.12
que se passe-t-il si on suppose maintenir un codage en taille 2.6

								1	1.	0	1	1	1	0	0
							x	0	0.	0	1	0	1	0	0
						1	1	0	1	1	1	0	0	.	.
				1	1	0	1	1	1	0	0	.	.	.	.
0	0	0	1.	0	0	0	1	0	0	1	1	0	0	0	0

Exercices

1. Additionner 10.0111 et 0.101.
2. Multiplier 11.1011 par 10.01 (codage binaire en virgule fixe sur 8 bits dont 5 en partie fractionnaire).
3. Additionner 101.0101 et 11.1 (codage Cà2 en virgule fixe sur 8 bits dont 5 en partie fractionnaire).

0.3 - Codage des réels

Virgule flottante

Norme IEEE 754

- norme adoptée en 1985
- définit la simple et la double précision
- principe général :

  signe

  mantisse de taille m (simple : 23, double : 52)

  exposant de taille k (simple : 8, double : 11)

$$x = (-1)^s \, x_0.x_1 x_2 \dots x_{m-1} \, 2^e$$

Virgule flottante, norme IEEE 754

Mantisse normalisée

- on impose (sauf cas particulier) $x_0 = 1$
- il devient inutile de l'expliciter dans la mantisse

$$x = (-1)^s \, 1.x_0 x_1 x_2 \ldots x_{m-1} \, 2^e$$

Exposant biaisé

- pour ne pas avoir de signe, on code e+biais
- biais$= 2^{k-1} - 1$
- valeur min réservée pour le codage de 0
- valeur max réservée pour le codage des infinis et de `NaN`
- simple précision, biais$= 127$, exposant biaisé va de 1 pour $e = -126$ à 254 pour $e = 127$
- double précision, biais$= 1023$, exposant biaisé va de 1 pour $e = -1022$ à 2046 pour $e = 1023$

0.2 - Codage des réels

Virgule flottante, norme IEEE 754

Exercices

1. Coder 12,5 en simple précision.
2. Coder -0,390625 en simple précision.
3. En codage flottant simple précision, quel réel représente 1 10000101 00100000000000000000000 ?

Codage de 0

- exposant biaisé min $= 000\ldots000$
- mantisse nulle $= 000\ldots000$
- signe 0 ou 1, pour 0^+ et 0^-
- attention : $\sqrt{0^-} = 0^-$

Codage des infinis

- exposant biaisé max $= 111\ldots111$
- mantisse nulle $= 000\ldots000$
- signe 0 ou 1, pour $+\infty$ et $-\infty$

0.3 - Codage des réels

Virgule flottante, norme IEEE 754

NaN

- "not a number"
- symbolise un résultat invalide ou indéterminé
- exposant biaisé max = 111...111
- mantisse non nulle
- signe 0 ou 1

Exemples

- $+\infty \; - \; +\infty$
- $0 \times \infty$
- $0/0$
- $+\infty \; / \; +\infty$

NaN (suite)

- propagation des NaN
- résultats des opérateurs logiques pas toujours intuitifs

Exemples

- $x + \mathtt{NaN} = \mathtt{NaN}$
- $\mathtt{NaN} - \mathtt{NaN} = \mathtt{NaN}$
- si $x = \mathtt{NaN}$ alors $x == x$ est faux et $x! = x$ est vrai
- si $x = \mathtt{NaN}$ ou $y = \mathtt{NaN}$ alors $x < y$, $x <= y$, $x == y$, $x >= y$ et $x > y$ sont faux

0.3 - Codage des réels

Virgule flottante, norme IEEE 754

Nombres dénormalisés

- exposant minimal
- mantisse non normalisée (pas de '1' implicite)
- but : échantillonner mieux autour de 0

Exemples

0 00000000 00100000000000000000000 $= 2^{-129}$

1 00000000 10000000000000000000000 $= -2^{-127}$

0 00000000 00000000000000000000001 $= 2^{-149}$

Arrondis

Notion d'arrondi :

- ensemble des réels échantillonné par valeurs représentables en machine
- déterministe
- 4 modes possibles :
  - au plus proche (arrondi "pair" si on est au milieu)
  - par excès (vers $+\infty$)
  - par défaut (vers $-\infty$)
  - vers zéro

Arrondis (suite)

Notion d'arrondi "exact" ou "correct" :

- principe : le système doit se comporter comme si le calcul était fait en précision infinie sur les opérandes exactes, puis arrondi
- comportement prévisible et reproductible
- indispensable pour portabilité des logiciels, reproductibilité des calculs, interopérabilité des systèmes
- difficile à satisfaire pour les opérations élémentaires

Arrondis, cas des fonctions élémentaires

- si on a m bits de mantisse, on calcule d'abord une approximation de précision n bits, avec $m < n$, puis on arrondit
- question fondamentale : quel n faut-il pour que l'approximation obtenue soit la même que l'arrondi du résultat exact ?
- pour les opérateurs arithmétiques, $n = m + 3$ suffit toujours
- théorème/algo de 1975 : pour calculer *log* et *exp* en double précision, il faut $n = 10^{244}$! ! !
- théorème/algo de 1995 : on se ramène à environ $n = 1\,000\,000$ bits
- optimal ? ? ?

Opérations arithmétiques : cas de l'addition $(s_x, e_x, m_x) + (s_y, e_y, m_y)$

- traiter les cas particuliers : $x_1 = NaN$ ou $x_2 = NaN$, $x_1 = 0$ ou $x_2 = 0$, $x_1 = +/-\infty$ ou $x_2 = +/-\infty$

addition	$-\infty$	$x \in \mathbb{F}^*_-$	0^-	0^+	$x \in \mathbb{F}^*_+$	$+\infty$	NaN
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	NaN	NaN
$y \in \mathbb{F}^*_-$	$-\infty$	$x+y$ ou $-\infty$	y	y	$x+y$	$+\infty$	NaN
0^-	$-\infty$	x	0^-	0^+	x	$+\infty$	NaN
0^+	$-\infty$	x	0^+	0^+	x	$+\infty$	NaN
$y \in \mathbb{F}^*_+$	$-\infty$	$x+y$	y	y	$x+y$ ou $+\infty$	$+\infty$	NaN
$+\infty$	NaN	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

0.3 - Codage des réels

Virgule flottante, norme IEEE 754

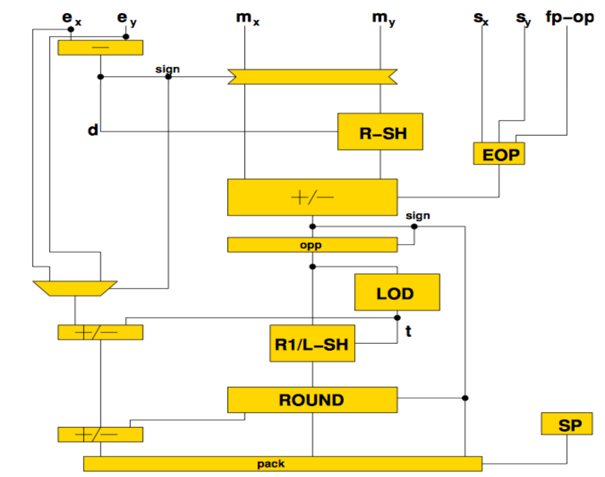
Opérations arithmétiques : cas de l'addition $(s_1, e_1, m_1) + (s_2, e_2, m_2)$

- différence des exposants : $d = e_x - e_y$
- échange des valeurs : $x \longleftrightarrow y$ et $d = -d$ si $d < 0$
- alignement des mantisses : $m_y = m_y >> d$
- calcul du type d'opération :
  $op = +$ si $(add, s_x = s_y)$ ou $(sub, s_x \neq s_y)$
  $op = -$ si $(add, s_x \neq s_y)$ ou $(sub, s_x = s_y)$
- addition/soustraction : $m_r = m_x \ op \ m_y$
- correction du signe : $m_r = -m_r$ si $m_r < 0$

Opérations arithmétiques : cas de l'addition $(s_1, e_1, m_1) + (s_2, e_2, m_2)$

- détection du 1 de poids fort ($t =$indice MSO, most significant one)
- normalisation (et correction exposant) : $m_r = m_r << t$ et $e_r = e_r + t$
- dénormalisation
- arrondi (en fonction des 3 bits de garde)
- renormalisation si l'arrondi provoque une propagation de retenue

Opérations arithmétiques : cas de l'addition



Exercices

1. En codage flottant simple précision, additionner
   1 00000101 00100010001000100010001
   et 0 00000011 10101010101010101010101.
2. Multiplier ces deux mêmes nombres.