Architecture des machines

Jean-Baptiste.Yunes@univ-paris-diderot.fr
Université Paris Diderot
http://www.liafa.univ-paris-diderot.fr/~yunes/

Octobre 2012

Les bits

Beaucoup des supports numériques utilisent des bits pour coder de l'information :

- le passage ou non d'un courant (transistor),
- un trou ou non sur un support (carte perforée, DVD),
- l'orientation magnétique sur un support (Disque dur) ;

Un bit (binary digit) est un chiffre binaire.
Un simple bit n'est pas très utile, ils sont généralement regroupés par paquets.
Ces paquets sont interprétés pour représenter les éléments d'un ensemble fini, très souvent :

- ensemble de nombres ;
- lettres d'un alphabet.

8 bits forment un octet. les groupements les plus utilisés sont 16, 32, 64 et 128 bits, soit 2, 4, 8 et 16 octets.

Les nombres

Sont fréquemment employés trois codages pour représenter les nombres :

- le codage non signé pour représenter des ensembles de nombres entiers positifs ou nul ;
- le codage en complément à deux pour représenter des ensembles de nombres relatifs ;
- le codage à virgule flottante pour approximer des ensembles de réels.

Il est très important de noter que toutes les représentations couramment employées n'utilisent qu'un nombre borné de bits pour le codage. Ce qui conduit à ce que certaines opérations sur ces nombres produisent un dépassement de capacité (overflow). Les effets peuvent être surprenants. Par exemple, un programme en langage C a produit sur une certaine machine le résultat suivant :

$$100 * 200 * 300 * 400 = -1894967296$$

Les nombres

La plupart des propriétés mathématiques des nombres et des opérations afférentes sont conservées (mais pas toujours !).
Pour les entiers (signés ou non) l'addition et la multiplication sont commutatives et associatives.
Pour les nombres en virgule flottante (les flottants) ce n'est malheureusement pas le cas !

$$(1 + 1e20) - 1e20 \neq 1 + (1e20 - 1e20)$$

Attention aux dépassements de capacité.
L'étude des différentes représentations est importante, car elle permet d'être capable d'écrire des programmes qui se comportent correctement (en prenant en compte ces problèmes).

L'arithmétique des ordinateurs

La représentation des nombres

Dans n'importe quelle base b un nombre n s'exprime :

$$n = \sum_{i=0} c_i \times \mathrm{b}^i$$

où c_i est le chiffre de rang (en position) i.
Si p est le plus petit i tel que $n < p^i$ alors le nombre s'écrit (en notation positionnelle) :

$$n = c_{p-1} cp - 2 \ldots c_1 c_0$$

Par exemple, en base 8 le nombre 256 (noté 256_8) vaut $2 \times 8^2 + 5 \times 8^1 + 6 \times 8^0 = 2 \times 64 + 5 \times 8 + 6 = 128 + 40 + 6 = 152$ en base 10 (noté 152_{10}).
Trois bases sont couramment employées lorsqu'on pilote une machine à un niveau relativement bas :

- la base 2 avec les chiffres 0 et 1,
- la base 8 avec les chiffres 0, 1, 2, 3, 4, 5, 6, 7,
- la base 16 avec les chiffres 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

On note généralement :

- 1768 le nombre 1768_{10},
- 0x14AF le nombre $14AF_{16} = 5295_{10}$,
- 0354 le nombre $0354_8 = 236_{10}$.

Les bases 8 et 16 sont utiles car elles permettent d'écrire de façon plus concise des nombres binaires.

Byte-ordering

Les machines stockent les bits dans une mémoire (à plat), où chaque case mémoire peut contenir un octet. Les cases mémoires sont numérotées (par des entiers naturels).
Dans quel ordre une machine stocke t-elle les 4 octets représentant un entier de 32 bits (4 octets) ? Il n'y a chez nous pas de discussion concernant la notation positionnelle communément employée mais on pourrait écrire les nombres à l'envers par exemple.
Le problème est celui de l'écriture ! Est-ce qu'on écrit de droite à gauche ? De gauche à droite ? Autrement ?
Deux façons assez répandues sont utilisées dans les machines : big endian et little endian.
Soit un entier $b_{31}b_{30}\ldots b_1b_0$ donc logiquement découpé en $O_3O_2O_1O_0$ où

$$O_i = b_{i*8+7}b_{i*8+6}b_{i*8+5}b_{i*8+4}b_{i*8+3}b_{i*8+2}b_{i*8+1}b_{i*8}$$

Byte-ordering

Le stockage big-endian consiste stocker les octets dans l'ordre décroissant des puissances (octet de poids fort d'abord (big-end), le suivant juste derrière, etc). C'est l'écriture habituelle des nombres.
Le stockage little-endian procède à l'envers (octet de poids faible d'abord (little-end), le suivant derrière, etc).
La petite histoire veut que ces dénominations proviennent des ≪ Voyages de Gulliver ≫, livre dans lequel une guerre éclate entre deux fractions qui s'écharpent sur le problème de savoir si les œufs à la coque doivent être mangés en les ouvrant par la petite ou la grande pointe.
Si en général, la question du byte-ordering est invisible au programmeur, elle peut revenir à la surface à différents moments : lorsqu'on tente d'examiner le contenu d'une mémoire ou lorsqu'on désire transmettre des données d'une machine à l'autre...

Les nombres entiers

Les entiers naturels

On définit les entiers sur l bits de la manière suivante. On note $\vec{x}_\ell$ le mot de ℓ bits $[x_{\ell-1}, \ldots, x_0]$

$$B2U_\ell(\vec{x}) = \sum_{i=0}^{i=\ell-1} x_i 2^i$$

est la fonction qui associe aux mots de ℓ bits un nombre naturel correspondant. Ces nombres sont donc dans l'ensemble $[0, 2^\ell[$.

$$B2U_\ell : \{0,1\}^\ell \mapsto \{0, \ldots, 2^\ell - 1\}$$

Les nombres entiers

Les entiers relatifs

La fonction suivante est appelée représentation en complément à deux.

$$B2S_\ell(\vec{x}) = -x_{\ell-1}2^{\ell-1} + \sum_{i=0}^{i=\ell-2} x_i 2^i$$

est la fcontion qui associe aux mots de ℓ bits un nombre relatif correspondant. Ces nombres sont dans l'ensemble $[-2^{\ell-1}, 2^{\ell-1} - 1]$.

$$B2U_\ell : \{0,1\}^\ell \mapsto \{-2^{\ell-1}, \ldots, 2^{\ell-1} - 1\}$$

Dans ce codage, le bit de poids fort est appelé bit de signe.
On remarquera que l'intervalle n'est pas symétrique $|min_\ell| = |max_\ell| + 1$
Il existe deux autres alternatives : le complément à un et le codage du signe.

$$\text{comp à } 1 B2S_\ell(\vec{x}) = -x_{\ell-1}(2^{\ell-1} - 1) + \sum_{i=0}^{i=\ell-2} x_i 2^i$$

$$\text{signe} B2S_\ell(\vec{x}) = -1^{x_{\ell-1}}. \sum_{i=0}^{i=\ell-2} x_i 2^i$$

Inconvénient : deux 0 et additions et soustractions différentes

Les nombres entiers

En base 2, si l'on dispose de 16 bits, on peut représenter jusqu'à 65536 nombres. Donc (entre autres) les nombres de 0 à 65535 :

$$
\begin{aligned}
0000\,0000\,0000\,0000_2 &= 0_{10} \\
0000\,0000\,0000\,0001_2 &= 1_{10} \\
0000\,0000\,0000\,0010_2 &= 2_{10} \\
&\vdots \\
1111\,1111\,1111\,1101_2 &= 65533_{10} \\
1111\,1111\,1111\,1110_2 &= 65534_{10} \\
1111\,1111\,1111\,1111_2 &= 65535_{10}
\end{aligned}
$$

ou les nombres de -32768 à $+32767$:

$$
\begin{aligned}
0000\,0000\,0000\,0000_2 &= 0_{10} \\
0000\,0000\,0000\,0001_2 &= 1_{10} \\
0000\,0000\,0000\,0010_2 &= 2_{10} \\
&\vdots \\
0111\,1111\,1111\,1110_2 &= 32766_{10} \\
0111\,1111\,1111\,1111_2 &= 32767_{10} \\
1000\,0000\,0000\,0000_2 &= -32768_{10} \\
1000\,0000\,0000\,0001_2 &= -32767_{10} \\
&\vdots \\
1111\,1111\,1111\,1101_2 &= -3_{10} \\
1111\,1111\,1111\,1110_2 &= -2_{10} \\
1111\,1111\,1111\,1111_2 &= -1_{10}
\end{aligned}
$$

Les nombres entiers

Cette représentation s'appelle la complémentation à deux car la somme de x et $-x$ est une puissance de 2 (complément à 2^n). Pour la complémentation à 1 la somme de x et $-x$ est le vecteur rempli de 1, soit $2^n - 1$).
De plus on peut remarquer que seuls les nombres négatifs possèdent un bit de poids fort à 1. On l'appelle alors le bit de signe. On a donc :

$$d_{15} \times (-2^{15}) + d_{14} \times 2^{14} + \ldots + d_1 \times 2^1 + d_0 \times 2^0$$

Les nombres entiers

L'extension du signe

Pour passer d'une représentation à n bits à une représentation à $m > n$ bits il suffit de recopier le bit de signe.

Ex :

Le nombre					s'écrit
$S2B_{16}(2)$		0000	0000	0000	0010
$S2B_{20}(2)$	0000	0000	0000	0000	0010
$S2B_{16}(-2)$		1111	1111	1111	1110
$S2B_{20}(-2)$	1111	1111	1111	1111	1110

Les nombres entiers

L'addition et la soustraction

Ajoutons 13 et 7 :

$$\begin{array}{rcccc} & 0000 & 0000 & 0000 & 1101 & 13_{10} \\ + & 0000 & 0000 & 0000 & 0111 & 7_{10} \\ \hline & 0000 & 0000 & 0001 & 0100 & 20_{10} \end{array}$$

Pour soustraire on peut le faire directement :

$$\begin{array}{rcccc} & 0000 & 0000 & 0000 & 1101 & 13_{10} \\ - & 0000 & 0000 & 0000 & 0111 & 7_{10} \\ \hline & 0000 & 0000 & 0000 & 0110 & 6_{10} \end{array}$$

ou en utilisant la somme et la complémentation à deux :

$$\begin{array}{rcccc} & 0000 & 0000 & 0000 & 1101 & 13_{10} \\ + & 1111 & 1111 & 1111 & 1001 & -7_{10} \\ \hline & 0000 & 0000 & 0000 & 0110 & 6_{10} \end{array}$$

Remarquons que dans la dernière opération, un bit a été perdu (dernière retenue à gauche). Il s'agit du dépassement de capacité (overflow). Ceci peut se produire à tout moment, que l'on additionne des nombres positifs ou négatifs, que l'on soustrait...

Les flottants

La notation scientifique : $0,56 \times 10^{-23}$ ou 0.56E-23.
La notation scientifique normalisée : la mantisse appartient à l'intervalle $[1, 10[$.
$0,56 \times 10^{-23}$ n'est pas normalisé.
Sa normalisation est 5.6×10^{-24}.

Les flottants

IEEE 754

La représentation IEEE 754 sur 32 bits (le ≪ flottant simple précision ≫) :

31	30	...	23	22	...	0
s	exp.			mantisse		

La représentation IEEE 754 sur 64 bits (le ≪ flottant double précision ≫) :

63	62	...	52	51	...	0
s	exp.			mantisse		

Comme la représentation IEEE 754 suppose que les nombres sont normalisés, le premier 1 est sous entendu, la précision est donc de 24 bits ou 53 bits pour la mantisse.
On a donc :

$$(-1)^s \times (1 + \mathit{mantisse}) \times 2^{\mathit{exp}}$$

Pour permettre de simplifier les tests l'exposant est représenté en forme biaisée. Le véritable exposant est égal au nombre non signé représenté moins le biais (127 en simple précision, 1023 en double). Donc la valeur représentée est en fait :

$$(-1)^s \times (1 + \mathit{signifiant}) \times 2^{\mathit{exp}-\mathit{biais}}$$

On notera aussi que les IEEE 752 normalisés ont un exposant non nul.

Les flottants

Ex : Représentez le nombre -.75_{10} en simple précision IEEE 754.

$$\begin{aligned} -0.75_{10} &= -\frac{3_{10}}{4_{10}} = -\frac{11_2}{100_2} \\ &= -0,11_2 = -1,1 \times 2^{-1} \end{aligned}$$

donc

$$(-1)^1 \times (1 + 0,10000\ldots) \times 2^{-1} = (-1)^1 \times (1 + 0,10000\ldots) \times 2^{126-127}$$

la représentation est donc
1011 1111 0100 0000 0000 0000 0000 0000.
Le nombre zéro est représenté de façon particulière (il existe des flottants non normalisés) : tout à zéro.
Il existe aussi trois valeurs spéciales ($+\infty$, $-\infty$ et NaN).

Codes de gray

Générer les sous-ensembles d'un ensemble à n éléments est équivalent à générer l'ensemble des mots de n-bits. Générer l'ensemble des mots de n-bits peut consister à générer l'ensemble des entiers de 0 à $2^n - 1$ par incrémentation ordinaire. Mais l'incrémentation d'un entier en base 2 nécessite de modifier plusieurs bits en général.

Il existe un codage permettant d'énumérer tous les mots de n-bits de sorte que le passage d'un mot au suivant ne comporte qu'une seule modification ($0 \rightarrow 1$ ou $1 \rightarrow 0$).

Ces codes sont connus sous le nom de codes de Gray.

Une façon standard d'obtenir un code de Gray est la suivante.

Si on suppose que $G_n = (G_0, G_1, \ldots, G_{2^n-1})$ est une séquence de Gray de mots de n-bits alors $G_{n+1} = (0G_0, 0G_1, \ldots, 0G_{2^n-1}, 1G_{2^n-1}, \ldots, 1G_1, 1G_0)$ est une séquence de Gray de mots de $n+1$-bits.

Cette version particulière du code de Gray est appelée code binaire réflexif.

Tableaux de Karnaugh

Les tableaux de Karnaugh sont utilisés pour tenter de simplifier l'expression de fonctions Booléennes.
L'idée est de représenter la fonction de sorte que l'on visualise plus facilement certains termes de la fonction dont l'expression sera simple.
Pour une fonction booléenne à n variables, le tableau aura $2^{\lfloor \frac{n}{2} \rfloor}$ colonnes et $2^{\lceil \frac{n}{2} \rceil}$ lignes.
La colonne i (respectivement la ligne) aura pour étiquette le i-ème mot du code binaire réflexif de longueur $\lfloor \frac{n}{2} \rfloor$ (resp. $\lceil \frac{n}{2} \rceil$).
Chaque case du tableau aura pour valeur celle de la fonction considérée pour la valeur des variables indiquée en tête de la colonne et ligne.
On fabrique ensuite des termes par regroupements (des rectangles les plus grands possibles contenant 2^k cases pour lesquelles la fonction vaut 1). La fonction s'exprime alors comme somme de simple produits (les produits étant réduits aux variables - ou leur négation - qui ne changent pas de valeur dans le code binaire).

Tableaux de Karnaugh

Prenons la fonction majorité (la fonction qui vaut 1 lorsque pour plus de la moitié les valeurs d'entrées valent 1). La table de vérité pour la majorité à quatre variables est :

x	y	z	t	f(x,y,z,t)
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1

x	y	z	t	f(x,y,z,t)
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

On rappelle que toute fonction Booléenne peut s'exprimer en FND (forme normale disjonctive). Ainsi son expression en forme normale disjonctive (somme de produits) est :

$$\begin{aligned} f(x,y,z,t) = & (\neg x.y.z.t) + (x.\neg y.z.t) + \\ & (x.y.\neg z.t) + (x.y.z.\neg t) + \\ & (x.y.z.t) \end{aligned}$$

Tableaux de Karnaugh

Le tableau (ou l'un des) de Karnaugh associé à la fonction majorité à quatre variables est :

		xy			
		00	01	11	10
zt	00	0	0	0	0
	01	0	0	1	0
	11	0	1	1	1
	10	0	0	1	0

		xy			
		00	01	11	10
zt	00	0	0	0	0
	01	0	0	1	0
	11	0	1	1	1
	10	0	0	1	0

La fonction peut alors s'écrire sous la forme :

$$f(x, y, z, t) = (y.z.t) + (x.z.t) + (x.y.t) + (x.y.z)$$

On oubliera pas que le code de Gray est cyclique et que le tableau est un tore.

La multiplication selon Karatsuba

Si A et B sont deux nombres de taille n, on peut découper ces deux nombres de la façon suivante :
A : $A_h A_b$ tel que $A = A_h.2^{\frac{n}{2}} + A_b$
B : $B_h B_b$ tel que $B = B_h.2^{\frac{n}{2}} + B_b$
Le produit de A par B s'exprime donc :

$$A.B = (A_h.2^{\frac{n}{2}} + A_b).(B_h.2^{\frac{n}{2}} + B_b)$$

donc

$$A.B = A_h B_h.2^n + (A_h B_b + A_b B_h)2^{\frac{n}{2}} + A_b B_b$$

Pour gagner en performances on peut faire l'observation suivante (découverte par Karatsuba et Ofman) :

$$\begin{aligned} A.B = {} & (A_h B_h)2^n + \\ & ((A_h + B_b).(A_b + A_h) - A_h B_h - A_b B_b)2^{\frac{n}{2}} + \\ & A_b B_b \end{aligned}$$

où l'on observe qu'il n'y a que trois produits (moins de produits mais plus de sommes).

Thèmes de réflexion

- retracer l'histoire de la base 10,
- retrouver le principe physique utilisé pour stocker l'information pour les divers supports connus,
- retrouver la représentation des nombres utilisés par le langage Java (normalisé pour toutes les plateformes),
- reconstruire le tableau de correspondance entre les chiffres de la base 16 et leur écriture en base 8 et 2 (ainsi qu'en décimal) (attention : n'utiliser que des codages de longueur fixe !),
- écrire un programme permettant d'obtenir le tableau précédent,
- sachant que la taille standard d'un entier signé est de 4 octets, retrouver la plus petite valeur représentable en complément à deux et la plus grande, quelles seraient ces valeurs si l'on utilisait un codage sur 8 octets,
- retrouver quel intérêt il pourrait y avoir à écrire les nombres de gauche à droite...
- quels nombres relatifs sont représentés par les mots de 4 bits à travers la fonction $B2S$? Constater que le bit de poids fort code bien le signe.
- que code un vecteur rempli de 1 ?
- générer le code binaire réflexif des mots de 4-bits.

Thèmes de réflexion

- comment convertir un entier relatif codé sur n bits en le même entier relatif codé sur $m > n$ bits ?
- pourrait-on utiliser la même technique avec la base 10 pour enlever le signe ? comment ?
- faire faire des calculs qui débordent...
- jouer avec des bases négatives
- avec la représentation des entiers, que devient l'opération de division par 2 ou de multiplication par 2 ? généraliser aux bases quelconques.
- généraliser l'écriture binaire avec la présence d'une virgule.
- calculer la représentation IEEE 754 de $3/16$, celle de $1/3$?
- comprendre que les flottants ne sont pas répartis également sur la droite réelle.
- combien de bits sont modifiés au plus lorsqu'on passe de la représentation sur m bits d'un entier n à celle de $n + 1$?
- étudier le rapport entre la série $S_n = \sum_{k=0}^{n} \frac{1}{2^k}$, sa limite, et le nombre $0,11111111111\ldots$; rappeler qu'en base 10 aussi on a $0,99999\ldots = 1$
- simplifier diverses fonctions Booléennes en utilisant les tableaux de Karnaugh

ASCII

PDF : fr en	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
001	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
002	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
003	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
004	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
005	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
006	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
007	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

ISO 8859-1

ISO-8859-1																
	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1x	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2x	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
8x	PAD	HOP	BPH	NBH	IND	NEL	SSA	ESA	HTS	HTJ	VTS	PLD	PLU	RI	SS2	SS3
9x	DCS	PU1	PU2	STS	CCH	MW	SPA	EPA	SOS	SGCI	SCI	CSI	ST	OSC	PM	APC
Ax	NBSP	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
Bx	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Dx	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
Fx	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

- Andrew Tanenbaum, *Architecture de l'ordinateur*, InterEditions,1991.
- David A. Patterson & John L. Hennessy, *Computer Organization & Design : The hardware/software interface*. Morgan Kaufmann Publishers, Inc, 1994.
- David A. Patterson & John L. Hennessy, *Computer Architecture : A quantitative approach*, Morgan Kaufmann Publishers, Inc, 1990.
- William Barden, *The Z80 Microcomputer Handbook*, Howard W. Sams & Co., Inc., 1978.
- *Éléments pour une histoire de l'informatique*, Traduction d'œuvres choisies de Donald E. Knuth. Traduction dirigée par Patrick Cégielski. Société Mathématique de France - CSLI Publications, Stanford, California. 2011. isbn : 978-1-57586-622-2
- Philippe Breton, *Une histoire de l'informatique*, collection ≪ Points Sciences ≫ (réimpr. 1990), 261 p. (ISBN 2020123487)
- Georges Ifrah, *Histoire Universelle des Chiffres*, Robert Laffont. 1994. Deux tomes.
- Raúl Rojas, Ulf Hashagen (Eds.). *The First Computers : History and Architectures*. 2000. MIT Press.