I REPRESENTATION DES NOMBRES ENTIERS

I.1 Représentation d'un entier naturel

Un entier naturel est un nombre entier positif ou nul. Le choix à faire (c'est-à-dire le nombre de bits à utiliser) dépend de l'intervalle des nombres que l'on désire utiliser. Pour coder des nombres entiers naturels compris entre 0 et 255, il nous suffira de 8 bits (un octet) car 2^8=256. D'une manière générale un codage sur n bits pourra permettre de représenter des nombres entiers naturels compris entre 0 et 2^{n-1}.

Exemples : 9 = $(00000101)_2$, 128 = $(10000000)_2$

I.2 Représentation d'un entier relatif

Un entier relatif est un entier pouvant être négatif. Il faut donc coder le nombre de telle façon que l'on puisse savoir s'il s'agit d'un nombre positif ou d'un nombre négatif.

- **Problème :** Comment indiquer à la machine qu'un nombre est négatif ou positif ????

Il existe 3 méthodes pour représenter les nombres négatifs :

I.2 .1 Signe et valeur absolue (S/VA)

Si on travail sur n bits , alors le bit du poids fort est utilisé pour indiquer le signe (1 : signe négatif, 0 : signe positif) et les autres bits (n -1) désignent la valeur absolue du nombre.

- Exemple : Si on travail sur 4 bits

  $(-5)_{10}$= $(1\ 101)_2$ $(+5)_{10}$ = $(\ 0\ 101)_2$

Signe	VA	Valeurs
0	**00**	**+ 0**
0	01	+1
0	10	+2
0	11	+3
1	**00**	**-0**
1	01	-1
1	10	-2
1	11	-3

Sur n bits, l'intervalle de valeurs qu'on peut représenter en S/VA :

$$-(2^{(n-1)} - 1) \leq N \leq +(2^{(n-1)}-1)$$

Essayons de calculer 2 + (-2) = 0

	Binaire					Décimal	
	0	0	1	0		2	
+	1	0	1	0	+	−2	
=	1	1	0	0	=	−4	

Ce n'est pas ce que nous voulions :-4 au lieu de 0 !!

Avantages : C'est une représentation assez simple.

Inconvénient : le zéro possède deux représentations +0 et -0

I.2 .2 Le complément à un

On appelle complément à un (c.à.1) d'un nombre N un autre nombre N' tel que : $N+N'=2^n-1$

n : est le nombre de bits de la représentation du nombre N .

Exemple :

Soit N=1010 sur 4 bits donc son complément à un de N : $N'= (2^4 - 1)-N$

$N'=(16-1)-(1010)_2= (15)_{10} - (1010)_2 = (1111)_2 – (1010)_2 = 0101$

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

Pour trouver le complément à un d'un nombre, il suffit d'inverser tous les bits de ce nombre : Si le bit est un 0 mettre à sa place un 1 et si c'est un 1 mettre à sa place un 0 .

Exemples

$(10010011) = (01101100)_{c.à.1}$

$(11001100) = (00110011)_{c.à.1}$

En Complément à un, le bit du poids fort nous indique le signe (0 : positif , 1 : négatif).

Le complément à un du complément à un d'un nombre est égale au nombre lui même.

CA1(CA1(N))= N

Exemple :

Quelle est la valeur décimale représentée par la valeur 101010 en complément à 1 sur 6 bits ?

- Le bit poids fort indique qu'il s'agit d'un nombre négatif.
- Valeur = - CA1(101010)

$= - (010101)_2 = - (21)_{10}$

Si on travaille sur 3 bits

CA1	Binaire	Décimal
000	000	**+ 0**
001	001	+ 1
010	010	+ 2
011	011	+ 3
100	- 011	- 3
101	- 010	- 2
110	- 001	- 1
111	- 000	**- 0**

Inconvénient : Dans cette représentation le zéro possède une double représentation.

Si on travaille sur n bits, l'intervalle des valeurs qu'on peut représenter en CA1 :

$$-(2^{(n-1)}-1) \leq N \leq +(2^{(n-1)}-1)$$

I.2 .3 Le complément à deux

Les nombres positifs sont codés de la même manière qu'en binaire pure alors qu'un nombre négatif est codé en ajoutant la valeur 1 à son complément à 1. Le bit le plus significatif est utilisé pour représenter le signe du nombre.

La représentation en complément à deux est la représentation la plus utilisée pour la représentation des nombres négatifs dans la machine.

Le complément à deux du complément à deux d'un nombre est égal au nombre lui-même.

CA2(CA2(N))= N

CA2	binaire	valeur
000 001 010 011	000 001 010 011	**+ 0** + 1 + 2 + 3
100 101 110 111	- 100 - 011 - 010 - 001	- 4 - 3 - 2 - 1

Avantage : On remarque que le zéro n'a pas une double représentation

Si on travail sur n bits, l'intervalle des valeurs qu'on peut représenter en CA2 :

$$-(2^{(n-1)}) \leq N \leq +(2^{(n-1)} - 1)$$

Exemple

On désire coder la valeur −19 sur 8 bits. Il suffit :

1. d'écrire 19 en binaire : 00010011

2. d'écrire son complément à 1 : 11101100

3. et d'ajouter 1 : 11101101

La représentation binaire de −19 sur 8 bits est donc 11101101.

- **Opérations arithmétique en complément à deux**

+9	01001
+4	00100
+13	**0**1101

(le bit de signe = 0 ➔ le nombre est positif)

$(01101)_2 = (13)_{10}$

+9	01001
−4	11100
+5	100101

On remarque que le résultat est sur 6 alors qu'on travaille sur 5 bits dans ce cas on ignore le sixième bit (appelé report) de ce fait le résultat de l'addition sera 00101(Résultat positif)

$(00101)_2 = (5)_{10}$

−9	10111
−4	11100
−13	110011

On ignore le report le résultat est 110011 (nombre négatif)

Résultat = - CA2 (10011)= -(01101)

= - 13

−9	10111
+9	01001
0	100000

Le résultat est positif

$(00000)_2 = (0)_{10}$

➢ La retenue et le débordement

- ✓ On dit qu'il y a une retenue si une opération arithmétique génère un report.
- ✓ On dit qu'il y a un débordement (Over Flow) ou dépassement de capacité: si le résultat de l'opération sur n bits **et** faux.
- ✓ Le nombre de bits utilisés est insuffisant pour contenir le résultat, autrement dit le résultat dépasse l'intervalle des valeurs sur les n bits utilisés.

➢ Cas de débordement

$$\begin{array}{rr} +9 & 01001 \\ \underline{+8} & \underline{01000} \\ +17 & 10001 \end{array}$$

Le résultat est négatif alors qu'il doit être positif ➔ Débordement !!!

$$\begin{array}{rr} -9 & 10111 \\ \underline{-8} & \underline{11000} \\ -17 & 01011 \end{array}$$

Le résultat est positif alors qu'il doit être négatif ➔ Débordement !!!

Nous avons un débordement si la somme de deux nombres positifs donne un nombre négatif, ou la somme de deux nombres négatifs donne un Nombre positif

Il n'y a jamais un débordement si les deux nombres sont de signes différents.

II. LA REPRESENTATION DES NOMBRES REELS

Un nombre réel est constitué de deux parties : la partie entière et la partie fractionnelle (les deux parties sont séparées par une virgule)

Problème : comment indiquer à la machine la position de la virgule ?

Il existe deux méthodes pour représenter les nombre réel :

II.1 Nombres à virgule fixe

Possède une partie 'entière' et une partie 'décimale' séparés par une virgule, utilisé parles premières machines. La position de la virgule est fixe d'où le nom.

Exemple : $(11.01)_2$, $(75.23)_8$, $(E7,A4)_{16}$

II.2 Nombres à virgule flottante

Les nombres à virgule flottante (nombres flottants) (floating-point numbers), sont en général des nombres non entiers dont on écrit les chiffres uniquement après la virgule et auquel on ajoute un exposant pour préciser de combien de positions la virgule doit être déplacées.

Par exemple en notation décimale, on écrira :

le nombre 31, 41592 sous la forme : $0.3141592 * 10^2$

le nombre -0,01732 sous la forme : $0.1732 * 10^{-1}$

En informatique, une norme s'est imposée pour la représentation des nombres flottants. C'est la norme IEEE 754.

➢ La norme IEEE 754

L'IEEE 754 est un standard pour la représentation des nombres à virgule flottante en binaire. Il est le plus employé actuellement pour le calcul des nombres à virgule flottante dans le domaine informatique,

Dans la norme IEEE 754, un nombre flottant est toujours représenté par un triplet (S,E,M)

1. La première composante S détermine le signe du nombre représenté, ce signe valant 0 pour un nombre positif, et 1 pour un nombre négatif , le signe est représenté par un seul bit, le bit de poids fort (Celui le plus à gauche)
2. la deuxième E désigne l'exposant est codé sur 8 bits consécutifs au signe
3. la troisième M désigne la mantisse (les bits situés après la virgule) sur les 23 bits restants

Ainsi le codage se fait sous la forme suivante :

S	E	E	E	E	E	E	E	E	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M

IEEE 754 Simple précision :

Signe (1 bit)	Exposant (8 bits)	Mantisse (23 bits)

IEEE 754 Double précision :

Signe (1 bit)	Exposant (11 bits)	Mantisse (52 bits)

Certaines conditions sont toutefois à respecter pour les exposants :

- ✓ L'exposant 00000000 est interdit
- ✓ L'exposant 11111111 est interdit. On s'en sert toutefois pour signaler les erreurs, on appelle alors cette configuration NaN (Not a Number)
- ✓ Il faut ajouter 127 à l'exposant (cas de la simple précision) pour une conversion de décimal vers un nombre réel binaire. Les exposants peuvent ainsi aller de -256 à 255.

La formule d'expression des nombres réels est ainsi la suivante :

$$(-1)^s \cdot 2^{(E-127)} \cdot 1,M$$

Remarque Le 127 du (E-127) vient de $2^{\text{nbre bit de l'exposant -1}} - 1$

Exemple 1 : Trouver la représentation IEEE 754 simple précision du nombre

$(35.5)_{10}$ Le nombre est positif ➔ S=0

$(35.5)_{10} = (100011.1)_2$ ……. Virgule fixe

$= 1.000111 * 2^5$ ..... Virgule flottante (M= 000111)

Exposant : E-127 = 5 ➔ E= 132 = $(10000100)_2$

Donc

0	10000100	00011100000000000000000
S	E	M

Exemple 2: Trouver la représentation IEEE 754 simple précision du nombre (-525.5)$_{10}$ Le nombre est négatif ➔ S=1

$(525.5)_{10} = (1000001101.1)_2$ ……. Virgule fixe

$= 1.0000011011 * 2^9$ ..... Virgule flottante (M= 0000011011)

Exposant : E-127 = 9 ➔ E= 136 = $(10001000)_2$

Donc

1	10001000	00000110110000000000000
S	E	M

Exemple 3 : Trouver la représentation IEEE 754 simple précision du nombre

$(-0.625)_{10}$ Le nombre est négatif ➔ S=1

$(0.625)_{10} = (0.101)_2$ ……. Virgule fixe

$= 1.01 * 2^{-1}$ ..... Virgule flottante (M= 01)

Exposant : E-127 = -1 ➔ E= 126 = $(1111110)_2$

Donc

1	01111110	01000000000000000000000
S	E	M

Exemple 4 : trouver le nombre flottant ayant la représentation IEEE754 suivante :

0	10000001	11100000000000000000000

S =0 ➔ Le nombre est positif

E = $(10000001)_2 = 129$ ➔ E-127 = 129 -127 =2

1.M = 1.111

➔ $1.111 * 2^2 = (111,1)_2 = (7.5)_{10}$

LA NOTATION DCB (DECIMAL CODE BINAIRE)

Une notation fréquemment utilisée dans les ordinateurs est la notation BCD (Binary Coded Decimal). Dans cette notation, chaque chiffre du nombre décimal est représenté en binaire, soit sur 4 bits (BCD compacté), soit sur 8 bits.

Par exemple, en BCD compacté, $(128)_{10}$ s'écrit donc :

0001 0010 1000

1 2 8

À l'exception de certaines calculettes, peu d'ordinateurs utilisent cette notation, car elle prend plus d'espace mémoire que le binaire. En effet, toutes les combinaisons de bits supérieures à 1001 (9) sont inutilisées. Cependant, c'est une étape intermédiaire indispensable quand on veut passer du décimal au binaire ou du binaire au décimal.

Voici comment on peut effectuer l'addition en BCD. Si l'on additionne en arithmétique binaire les codes BCD correspondant à des chiffres dont la somme ne dépasse pas 9, on aura évidemment un résultat donnant la représentation correcte de la somme DCB. Par exemple, 45 + 31 donnent bien 76 :

45	0100 0101
+31	+0011 0001
76	0111 0110

Cependant, dès que la somme dépasse 9 dans un motif de 4 bits, il faut apporter une correction. Par exemple, si on additionne 8 + 6, on obtient 1110, qui excède l'intervalle de définition d'un digit BCD. On voit cependant que si on ajoute 6 à ce résultat, on obtient bien la représentation BCD de 14:

8		1000	
+6	=	+0110	
14		1110	Résultat > 9,
		+0110	on ajoute 6
		10100	= 0001 0100 = 14_{BCD}

On effectue donc l'addition en motifs de 4 bits. Si le résultat dépasse 9 pour l'un ou plusieurs de ces motifs, on leur ajoute 6 pour forcer une retenue et on obtient le résultat escompté en BCD :

```
 76    0111 0110      76    0111 0110      26    0010 0110
+45    0100 0101     +43    0100 0011     +45    0100 0101
---   ----------     ---   ----------     ---   ----------
121    1011 1011     119    1011 1001      71    0110 1011
      +0110 0110           +0110                     +0110
      ----------           ----------     ---   ----------
      1 0010 0001          1 0001 1001          0111 0001
```

REPRESENTATIONS DES CARACTERES

C'est quoi l'ASCII, l'UNICODE, l'UTF-8 ?

1.L'ASCII (*American Standard Code for Information Interchange*).

L'ordinateur est une machine à calculer, il est capable d'effectuer des calculs sur des **nombres**. Il est incapable de comprendre le texte.

Il faut donc faire un choix : par quel nombre on prend pour représenter la lettre 'A' ? Et pour les signes de ponctuation, quels nombres utiliser ?

Il existe différentes conventions (ou codes). L'un des plus connus est le code **ASCII** (*American Standard Code for Information Interchange*). C'est un standard américain, mais c'est l'un des plus utilisés, en particulier sur la plupart des ordinateurs.

Le code ASCII définit précisément la correspondance entre symboles et nombres jusqu'au nombre 127:

Dec.	Car.	Dec.	Car.	Dec.	Car.	Dec.	Car.
0	NUL	32	SP	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

Il faut donc utiliser le nombre 97 pour représenter un '**a**' minuscule. Pour représenter un '**?**', il faut utiliser le code 63.
Certains codes (ceux inférieurs à 32) sont des codes de contrôle (il ne sont pas faits pour être affichés). Par exemple le code 10 permet d'aller à la ligne, le code 7 fait biper l'ordinateur, etc.

Mais on remarque qu'il n'y a aucun caractère accentué (Car c'est un code fait par les americains)

Il faut donc trouver un code plus pratique. Il existe : c'est l'**UNICODE**.

2. L'Unicode

Au lieu d'utiliser seulement les codes 0 à 127, il utilise des codes de valeur bien plus grandes.

Le code UNICODE permet de représenter tous les caractères spécifiques aux différentes langues. De nouveaux codes sont régulièrement attribués pour de nouveaux caractères: caractères latins (accentués ou non), grecs, arabes, allemands, japonais, chinois (L'alphabet Chinois *Kanji* comporte à lui seul 6879 caractères).

L'Unicode définie donc une correspondance entre symboles et nombres.

Voici une toute petite partie des tables UNICODE (les nombres sont présentés en notation hexadécimal):

Char	Code	Char	Code	Char	Code	Char	Code	Char	Code	Char	Code	Char	Code	Char	Code
	160	Ą	161	˘	162	Ł	163	¤	164	Ľ	165	Ś	166	§	167
¨	168	Š	169	Ş	170	Ť	171	Ź	172		173	Ž	174	Ż	175
°	176	ą	177	˛	178	ł	179	´	180	ľ	181	ś	182	ˇ	183
¸	184	š	185	ş	186	ť	187	ź	188	˝	189	ž	190	ż	191
Ŕ	192	-	193	Â	194	Ă	195	Ä	196	Ĺ	197	Ć	198	Ç	199
Č	200	É	201	Ę	202	Ë	203	Ě	204	Í	205	Î	206	Ď	207
Đ	208	Ń	209	Ň	210	Ó	211	Ô	212	Ő	213	Ö	214	×	215
Ř	216	Ů	217	Ú	218	Ű	219	Ü	220	Ý	221	Ţ	222	ß	223
ŕ	224	á	225	â	226	ă	227	ä	228	í	229	ć	230	ç	231
č	232	é	233	ę	234	ë	235	ě	236	í	237	î	238	ď	239
đ	240	ń	241	ň	242	ó	243	ô	244	ő	245	ö	246	÷	247
ř	248	ů	249	ú	250	ű	251	ü	252	ý	253	ţ	254	˙	255

3. Unicode dans la pratique: UTF-8

Généralement en Unicode, un caractères prend **2 octets**. Autrement dit, le moindre texte prend **deux fois plus de place qu'en ASCII**.

De plus, si on prend un texte en français, la grande majorité des caractères utilisent seulement le code ASCII. Seuls quelques rares caractères nécessitent l'Unicode, la solution proposé est d'utiliser: l'**UTF-8**.

Un texte en UTF-8 est simple: il est partout en ASCII, et dès qu'on a besoin d'un caractère appartenant à l'Unicode, on utilise un caractère spécial signalant "*attention, le caractère suivant est en Unicode*".

Par exemple, pour le texte "Bienvenue à l'université d'Annaba !", seul le "**à** " et le "**é** " ne font pas partie du code ASCII.

L'UTF-8 rassemble le meilleur de deux mondes: l'efficacité de l'ASCII et l'étendue de l'Unicode. D'ailleurs l'UTF-8 a été adopté comme norme pour l'encodage des fichiers XML. La plupart des navigateurs récents supportent également l'UTF-8 et le détectent automatiquement dans les pages HTML.