JFA - 31

Les nombres réels

Vu la constitution interne des systèmes informatiques, les données sont représentées par des "mots" d'un certain nombre fixé de bits. 8 bits (rare) ou 16 ou 32 , 64, et même 80 bits (10 octets). Les 32 bits (en 'simple précision') et 64 bits (en 'double précision') s'imposent chez les constructeurs.

Pour représenter les nombres réels en binaire, on pourrait réserver un espace :

- Pour le signe
- Pour la partie entière du nombre et
- Pour la partie fractionnaire,

S	Partie Entière							V	Partie Fractionnaire						
1	1	0	1	0	1	1	0	0	,	1	0	1	0	1	0	1

Pour représenter les nombres réels ***négatifs*** en binaire, on fait le complément à 1, **et on ajoute 1 à droite de la partie fractionnaire** :

En virgule fixe (13.3) :

```
2,5 = 0000000000010,100
```

$\overline{2{,}5} = 1111111111101{,}011$

$\overline{2{,}5} + 0{,}001 = 1111111111101{,}100$

```
=> - 2,5 = 1111111111101,100
```

JFA - 32

Les nombres réels : virgule fixe

- C'est la ***représentation en virgule fixe*** (fixed point). Elle est utilisée quand le processeur de l'ordinateur n'a aucune unité de calcul en virgule flottante ou quand une virgule fixe permet d'augmenter la vitesse d'exécution ou d'améliorer l'exactitude des calculs. La plupart des processeurs à faible coût (ex. : microcontrôleurs) ne disposent pas d'unité de calcul en virgule flottante. Ce mode de représentation n'a que l'avantage de la simplicité !

- Elle est peu employée, vu ses inconvénients :

  L'espace réservé à la partie fractionnaire limite le nombre de bits réservés à la partie entière.

  - Ce qui est gênant pour représenter de très grands nombres, pour lesquels la partie fractionnaire est généralement peu significative.
  - Inversement, la précision sur de très petits nombres est limitée par le manque d'espace dans la partie fractionnaire alors que pour ces nombres, la partie entière ne contient que des zéros.

D'où un espace mal utilisé dans les deux cas.

JFA - 33

Les nombres réels : virgule flottante

D'où le codage en virgule flottante qui permet une bien meilleure préservation des chiffres significatifs autant pour les grands que pour les petits nombres.

On va représenter le nombre sous la forme :

$$N = (-1)^S . 1,M . 2^E$$

Avec :

- S : Signe : 0 signe +, 1 signe -
- M : Mantisse dont la partie entière n'est pas exprimée,
- E : Exposant.

- Exemple :

$$-2,5_{10} \Rightarrow (-1)^1 . 2,5_{10} \Rightarrow (-1)^1 . 10,1_2 = (-1)^1 . 1,01 . 2^1$$

Avec :

- S : 1
- M : 01
- E : 1

- Le 0 , +/- l'infini, NAN (Not A Number) sont codés de façon spéciale,

iNFO IUT GRAND OUEST NORMANDIE R 1.03

JFA - 34

Les normes de représentation

Il existe plusieurs Normes de représentation des nombres en virgule flottante :

- ***IBM en 32 bits :***

31 — 24	23	22 — 0
Exposant + 127	signe	Mantisse

- ***IEEE754 en 32 / 64 / 80 bits :***

Type	Signe	Exposant	Mantisse
Simple précision 32 bits	31	30 — 23	22 — 0
	1 bit	8 bits, e + 127	23 bits
Double précision 64 bits	63	62 — 52	51 — 0
	1 bit	11 bits, e + 1023	52 bits
Précision étendue 80 bits	79	78 — 64	63 — 0
	1 bit	15 bits, e + 16383	64 bits

Sur 80 bits : $3,4.10^{-4932} \rightarrow 1,2.10^{4932}$

iNFO IUT GRAND OUEST NORMANDIE R 1.03

JFA - 35

Les normes de représentation

➢ ***Exemple de PI sur 32 bits en IEEE754 Simple précision :***

Type	Signe	Exposant	Mantisse
Simple précision 32 bits	31	30 23	22 0
	1 bit	8 bits, e + 127	23 bits

$X = 3{,}14159265385_{10}$

$X = 11.00100100001111111011010101000010_2$

$X = 1{,}10010010000111111011010.\ 2^1{}_2$

$X = (-1)^0.1{,}10010010000111111011010.\ 2^1{}_2$

Donc :

→ s = 0
→ m = 10010010000111111011010
→ e = 1+127 = 128 = 10000000
→ 0 10000000 10010010000111111011010
➔ 0100 0000 0100 1001 0000 1111 1101 1010
➔ x = 0x40490FDA

Sur 80 bits, l'étendue est de : $3{,}4.10^{-4932} \rightarrow 1{,}2.10^{4932}$

JFA - 36

Les codes

DUT Informatique – Semestre 1

Ressource R 1.03

Responsable : Jean-François ANNE

13/09/2025

JFA - 37

Les Codes

- Plusieurs codes sont utilisés pour représenter les données, avec leurs avantages et leurs inconvénients :
  - Code 8421 (Base 2),
  - Le code Binaire réfléchi (GRAY)
  - Le code DCB (BCD) ,
  - Codes ASCII
  - Code UTF 8
- Un ordinateur ne cesse de transférer des données : vers la mémoire, le disque, un périphérique, un autre ordinateur. Chaque transfert peut altérer les données, spécialement en analogique. Il faut donc être capable de détecter ces erreurs, voire de les corriger…
- On utilise donc des codes avec détection d'erreur :
  - Codes avec contrôle de parité
- Ou des codes avec auto correction d'erreurs,

IUT GRAND OUEST NORMANDIE iNFO R 1.03

JFA - 38

Le Binaire réfléchi ou code GRAY

- En binaire réfléchi, ***un seul chiffre change entre deux nombres consécutifs***.
- Il est utilisé lorsque l'on désire supprimer les erreurs d'états transitoires (glitchs). Il est fréquemment utilisé dans les capteurs angulaires ou de positionnement, mais aussi lorsque l'on désire une progression numérique binaire sans parasite transitoire.
- Le code Gray sert également dans les tableaux de Karnaugh utilisés lors de la conception de circuits logiques.
- ***Démonstration : En réalisant un comptage en binaire pur, on peut lire par erreur les étapes intermédiaires du comptage :***

En binaire :
7 (0111) -> 8 (1000)
0111 -> 0110 -> 0100 -> 0000 -> 1000

En code Gray, ***un seul chiffre change*** entre deux nombres successifs :
7 (0100) -> 8 (1100)

Ceci permet de limiter les parasites et les erreurs.

IUT GRAND OUEST NORMANDIE iNFO R 1.03

JFA - 39

Construction du code Binaire réfléchi

- ***Méthode de construction par miroir :***
  On le construit par blocs miroirs :
  - On pose 0 1 en vertical, on met un miroir (barre bleue),
  - on recopie les chiffres placés avant le miroir après le miroir,
  - puis on met des 0 à gauche des nombres avant le miroir,
  - et des 1 à gauche des nombres après le miroir, et ainsi de suite …

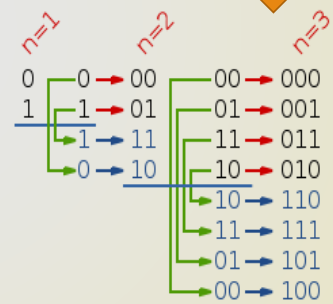
- ***Méthode de construction à partir du binaire pur :*** il suffit d'effectuer l'opération suivante :

$$n = \frac{N \oplus 2N}{2}$$

soit N = 0111,
2N = 1110
On effectue un OU Exclusif entre N et 2N :

$$\begin{array}{r} 0111 \\ \oplus\ 1110 \\ \hline 1001 \end{array}$$

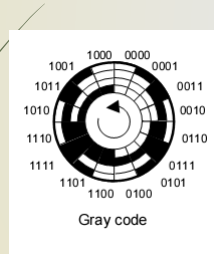
Puis nous divisons par 2 : 1001 / 2 = 0100
N = 0111 => n = 0100 en code Gray.

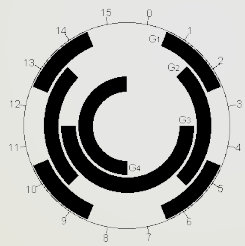


JFA - 40

Application du code GRAY

- On trouve des applications du code GRAY dans la vie de tous les jours :
  - Figure 1 :Programmateur de machine à laver (ne pas avoir l'essorage en même temps que le lavage !)
  - Figure 2 :Encodage de l'angle de rotation d'un lecteur de disquette, CDROM, Disque dur :





❖ ***Figure 1 :***
16 positions angulaires détectées en 4 bits

❖ ***Figure 2***

http://www.positron-libre.com/cours/electronique/logique-combinatoire/codage-information/code-gray.php

JFA - 41

Le code DCB (Décimal codé en Binaire) ou BCD Binary Coded Decimal)

- C'est la traduction en binaire **de chaque chiffre décimal**. Chaque chiffre est exprimé séparément par un quartet (4 bits). L'utilisation de ce code est importante, car chaque fois que l'on veut visualiser un nombre sur des afficheurs, il suffit d'écrire ce nombre en D.C.B. et de le relier sur les afficheurs. Ce code est donc utilisé dans tous les systèmes d'affichage de chiffres décimaux (Multimètres, Fréquencemètres Horloges...).
- ***Exemple : 15987_{10} en DCB :***

1 5 9 8 7
0001 0101 1001 1000 0111

Le code DCB du décimal 15987 est une séquence de 20 bits 0001 0101 1001 1000 0111

Décimal	*B.C.D*
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	0001 0000
11	0001 0001
12	0001 0010
13	0001 0011
14	0001 0100
15	0001 0101
16	0001 0110

JFA - 42

Additions en code DCB

- On peut effectuer des additions en code DCB, mais pour obtenir un résultat correct, il faut ajouter 6 si le quartet (4 bits) est supérieur à 9, et/ou si on a eu une retenue vers le bloc de 4 bits de gauche et ainsi de suite. :

Exemples :

```
  18        0001 1000
+  5      + 0000 0101
  23        0001 1101
          +      0110
            0010 0011
              2    3
```

```
  265                0010   0110   0101
+ 975              + 1001   0111   0101
  1240               1011   1101   1010
                                  + 0110
                            1110   0000
                          + 0110
                            0100
                  1100
                + 0110
              1   0010
  ------------------------------------
    0001      0010   0100   0000
      1         2      4      0
```

JFA - 43

Représentation des caractères

- L'ordinateur ne sachant faire que travailler sur des nombres, Il faut donc faire un choix :
  - Quel nombre prend-on pour représenter la lettre 'A' ?
  - Et pour les signes de ponctuation, quels nombres utiliser ?
  - Et pour les alphabets des différentes langues ?
  - ….

- Il existe donc différents codages (codes) pour représenter les caractères, chiffres, alphabets et autres ponctuations.
- Le code ASCII,
- L'EBCDIC,
- Pages de code : 1 page de code par langue, et par constructeur (Obsolète)
- L'Unicode,
- L'UTF 8, UTF 16, UTF32

IUT GRAND OUEST NORMANDIE iNFO R 1.03

JFA - 44

Code ASCII

- L'American Standard Code for Information Interchange (Code américain normalisé pour l'échange d'information) : **ASCII**, est une norme informatique de codage de caractères apparue dans les années 1960. C'était la norme de codage de caractères la plus influente, elle est en cours de remplacement par les normes UTF aujourd'hui.

- L'ASCII définit 128 caractères numérotés de 0 à 127, (7 bits suffisent). Toutefois, les ordinateurs travaillant sur un multiple de huit bits (un octet) depuis les années 1970, chaque caractère d'un texte en ASCII est stocké dans un octet dont le 8e bit est 0. Aujourd'hui encore, certains systèmes de messagerie électronique et de SMS fonctionnent avec des mots composés de seulement sept bits.

- Les caractères de numéro 0 à 31 et le 127 ne sont pas affichables ; ils correspondent à des commandes de contrôle informatique. Le caractère numéro 127 est la commande pour effacer. Le caractère numéro 32 est l'espace. Les autres caractères sont les chiffres arabes, les lettres latines majuscules (de 65 à 90) et minuscules sans accent (de 97 à 122), des symboles de ponctuation, des opérateurs mathématiques et quelques autres symboles.
- En modifiant le 6ème bit, nous passons de majuscules à minuscules, c'est-à-dire en ajoutant 32_{10} au code ASCII.

IUT GRAND OUEST NORMANDIE iNFO R 1.03

JFA - 45

Code ASCII sur 7 bits : 128 caractères

➢ ***Les codes de contrôle ASCII***

NUL Null : pas de caractère
BEL Bell : sonnerie
BS Backspace : arrière caractère
HT TAB : Tabulation horizontale
LF Line Feed : saut de ligne
VT Vertical tabulation : tab. verticale
FF Form feed : page suivante
CR Carriage return : retour à la ligne
CAN Cancel : annulation
ESC Escape : Echappement
SP Space : espace
DEL Delete : suppression

Exemple :
Y = 101 1001
Y = 59 (hexadecimal)

ACK = 06

LSB \ MSB		0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	0000	NUL	DLE	SP	0	@	P	`	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	-	=	M	]	m	{
E	1110	SO	RS	.	>	N	^	n	~
F	1111	SI	US	/	?	O	_	o	DEL

JFA - 46

Code ASCII étendu sur 8 bits (256 caractères)

➢ Maintenant, nous utilisons le code ASCII étendu. Il permet le codage de caractères sur 8 bits, soit 256 caractères possibles.

➢ ***Exemple d'une table de code étendu :***

LSB \ MSB		0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111
0	0000	NUL	DLE	SP	0	@	P	`	p	Ç	É	á	▒	└	ð	Ó	-
1	0001	SOH	DC1	!	1	A	Q	a	q	ü	æ	í	▒	┴	Ð		±
2	0010	STX	DC2	"	2	B	R	b	r	é	Æ	Ó	▓	┬	Ê	ô	_
3	0011	ETX	DC3	#	3	C	S	c	s	â	ô	ú	|	├	Ë	ò	¾
4	0100	EOT	DC4	$	4	D	T	d	t	ä	o	ñ	┤	─	È	õ	¶
5	0101	ENQ	NAK	%	5	E	U	e	u	à	ò	Ñ	Á	┼	I	Õ	§
6	0110	ACK	SYN	&	6	F	V	f	v	å	û	ª	Â	ã	í	µ	÷
7	0111	BEL	ETB	'	7	G	W	g	w	ç	ù	º	À	Ã	î	þ	¸
8	1000	BS	CAN	(	8	H	X	h	x	ê	ÿ	¿	©	╚	ï	Þ	°
9	1001	HT	EM	)	9	I	Y	i	y	ë	Ó	®	╣	╔	┘	Ú	¨
A	1010	LF	SUB	*	:	J	Z	j	z	è	Ú	¬	║	╩	┌	Û	·
B	1011	VT	ESC	+	;	K	[	k	{	ï	ø	½	╗	╦	█	Ù	¹
C	1100	FF	FS	,	<	L	\	l	|	î	£	¼	╝	╠	▄	ý	³
D	1101	CR	GS	-	=	M	]	m	{	ì	Ø	¡	¢	═	¦	Ý	²
E	1110	SO	RS	.	>	N	^	n	~	Ä	×	«	¥	╬	ì	¯	■
F	1111	SI	US	/	?	O	_	o	DEL	Å	ƒ	»	┐	¤	▀	´	

JFA - 47

L'UNICODE : L'Universal Code (ISO/CEI 10646)

- Devise de l'Unicode : ***"un seul nombre (codage) pour chaque caractère, quelque soit la plate-forme, le programme ou le langage,"***.
- Au lieu d'utiliser une plage réduite de caractères disponibles, on va utiliser un code permettant une plus grande quantité de signes (245 000 caractères). Il permet de représenter tous les caractères des différentes langues. Des mises à jour sont effectuées régulièrement pour ajouter de nouveaux caractères:
  - caractères latins (accentués ou non), grecs, cyrilliques, arméniens,
  - hébreux, thaï, hiragana, Katakana
  - ...
  - L'alphabet Chinois Kanji comporte à lui seul 6879 caractères.
- La dernière version, Unicode 9.0, a été publiée le 21 juin 2016
- Il sert à l'interopérabilité de logiciels, et permet de copier des textes utilisant des caractères de différents alphabets entre des logiciels différents, même n'ayant pas été spécifiquement conçus pour eux.
- Même si l'UNICODE est bien conçu, il reste assez peu utilisé. Pour les programmeurs, il n'est pas très facile à manipuler.
- Ce standard se développe de plus en plus. Les langages Java, .Net (C#) et Python, comme la plupart des systèmes d'exploitation (Windows, Linux, MacOS X...) supportent l'Unicode.

IUT GRAND OUEST NORMANDIE – iNFO – R 1.03

JFA - 48

L'UNICODE (ISO/CEI 10646)

- La mise au point et l'adoption de l'Unicode ont été très lentes, car le codage des caractères posait de gros problèmes techniques et politiques ; et il a fallu beaucoup de temps pour s'accorder sur une définition commune des caractères pour toutes les langues :
  - c'est quoi un caractère ? Le ss en allemand est-il un caractère ou un double s ?
- La représentation sous 2 octets de l'Unicode doublait la taille des documents
- Un autre problème : les différents constructeurs ne rangeaient pas les octets dans la mémoire dans le même ordre. On peut ranger les octets de poids forts en premier ou en dernier dans la mémoire.
- On a donc défini deux standards dits :
  - ***Little Endian (petit-boutiste ou petit-boutien)*** :
    - Poids Faible en début d'adresse mémoire
  - ***Big Endian (gros-boutiste ou grand-boutien)*** :
    - Poids Fort en début d'adresse mémoire,
- *Exemple :*
  - Microsoft Word utilise ***Little Endian,***
  - pour l'Internet , c'est ***Big Endian*** qui est recommandé.

IUT GRAND OUEST NORMANDIE – iNFO – R 1.03

JFA - 49

Définition de L'UNICODE

Unicode est défini suivant un modèle en couches présentées en partant de la plus haute (la plus éloignée de la machine) :

- **Répertoire des caractères abstraits :** La couche la plus élevée est la définition du jeu de caractères. Unicode leur donne des noms. Dresser la liste des caractères et leur donner des noms est donc la première couche d'Unicode. Par exemple, Latin-1 a un jeu de 256 caractères, et le caractère Ç est nommé « lettre majuscule latine c cédille ».
- **Jeu de caractères codés :** Après on ajoute à la table précédente, un index numérique (Nombre entier) associé à chaque caractère, appelé ***point de code***. L'espace de codage de ces nombres est divisé en 17 zones de 65 536 points de codes. Ces zones sont appelées plans.
  - Le point de code est noté U+xxxx où xxxx est en hexadécimal, et comporte 4 à 6 chiffres :
  - 4 chiffres pour le premier plan, appelé plan multilingue de base (donc entre U+0000 et U+FFFF);
  - 5 chiffres pour les 15 plans suivants (entre U+10000 et U+FFFFF);
  - 6 chiffres pour le dernier plan (entre U+100000 et U+10FFFF).
  - Ainsi, le caractère nommé « Lettre majuscule latine c cédille » (Ç) a un index de U+00C7. Il appartient au premier plan.

iNFO IUT GRAND OUEST NORMANDIE R 1.03

JFA - 50

Définition de L'UNICODE (Suite)

- **Formalisme de codage des caractères :** Puis le codage des caractères (représentation physique en mémoire) :
  - Cette couche spécifie quelle unité de codage (*code units*), ou codet, va représenter un point de code (caractère) : octet, Word (mot de 16 bits), ou Dword (mot de 32 bits).
- **Mécanisme de sérialisation des caractères :** Cette couche s'occupe de sérialiser les suites d'unités de codage définies par la couche précédente en suites d'octets. C'est ici que se choisit **l'ordre des octets** entre le gros-boutien (Big-Endian) et petit-boutien (Little-Endian). Il est possible d'ajouter un indicateur d'ordre des octets (ou BOM, byte order mark), qui permet d'indiquer en début de fichier ou de flot de données s'il est en gros-boutien ou en petit-boutien.
- **Surcodage de transfert :** Une dernière couche optionnelle où peut intervenir les mécanismes de compression ou de chiffrement.

iNFO IUT GRAND OUEST NORMANDIE R 1.03