Ecole Nationale des Sciences Appliquées

Kénitra

Cycle Préparatoire

Architecture des ordinateurs et Systèmes d'exploitation

Pr. N. EL HAMI

Année universitaire : 2020/2021

Chapitre 3 :

Systèmes de numération : Partie 2

1. Représentation des nombres réels à virgule fixe: 3
1.1 Représentation des nombres réels 3
1.2 Conversion d'un nombre réel du décimale vers le binaire : 4
1.3 Opérations arithmétiques 5

2- Nombres réels représentation en virgule flottante 7

3- Décimaux codés en binaire (BCD) 10

4- Données alphanumériques 11
4.1 Code ASCII standard : 7 bits 11
4.2 Code ASCII étendu : 8 bits 12

1. Représentation des nombres réels à virgule fixe:

Pour représenter les nombres réels il faut définir l'emplacement d'une virgule séparant la partie entière et la partie décimale.

Comment indiquer à la machine la position de la virgule ?

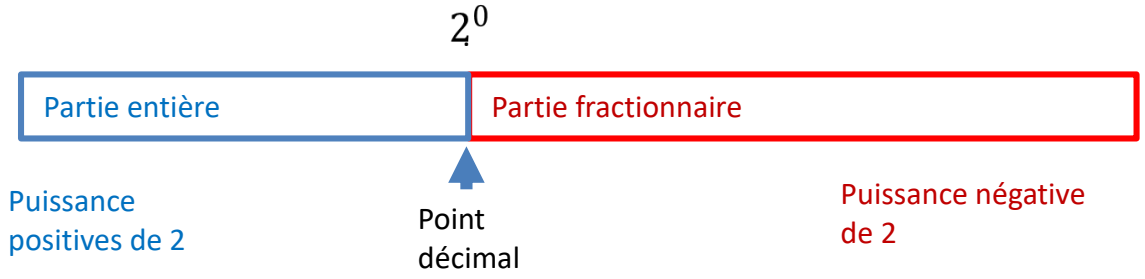
Il existe deux méthodes pour représenter les nombres réels :

Virgule fixe : la position de la virgule est fixe

Virgule flottante : la position de la virgule change (dynamique)

1.1 Représentation des nombres réels

Notation positionnelle avec les puissances positives et négatives de la base 2 nombre à virgule



Binaire → décimal

$\mathbf{0,101_2}$

1	0	1

0

1 x $\mathbf{2^{-1}}$ + 0 x $\mathbf{2^{-2}}$ + 1 x $\mathbf{2^{-3}}$

= 0,5 + 0 + 0, 125

= 0,625

On représente **la partie entière** sur **n bits** et la **partie fractionnelle** sur **p bits** + le bit du signe.

Exemple : si n=5 et p=4 $\mathbf{21,3125_{10}}$

21= 1 x $\mathbf{2^4}$ + 1 x $\mathbf{2^2}$ + 1 x $\mathbf{2^0}$

$\mathbf{0,3125_{10}}$ = 0 x $\mathbf{2^{-1}}$ + 1 x $\mathbf{2^{-2}}$ + 1 x $\mathbf{2^{-4}}$

= 0,25 + 0,0625

$\mathbf{21,3125_{10}}$ == 101010101

1.2 Conversion d'un nombre réel du décimale vers le binaire :

R est un réel = 0,abc

a	b	c

Binaire

Multiplier la partie fractionnaire par 2		Extraire la partie entière
	R = $a \times 2^{-1} + b \times 2^{-2} + c \times 2^{-3}$	
x 2	$a \times 2^{0} + b \times 2^{-1} + c \times 2^{-2}$	-----> a
x 2	$b \times 2^{0} + c \times 2^{-1}$	-----> b
x 2	$c \times 2^{0}$	-----> c

Continuer tant que la partie fractionnaire restante ≠ 0

$\mathbf{0,875_{10}}$ == binaire ?

0,875	x2	= 1,75
0,875	x2	= 1,5
0,875	x2	= 1,0 Fin

1	1	1

$\mathbf{0,875_{10}} = \mathbf{0,111_{2}}$

Conversion de la base décimale à la base 6

$\mathbf{45,125_{10}}$ == convertir $?_{6}$

45	6	7	6	1	6	
42	7	6	1	0	0	113
3		1		1		

0,125 x 6 = 0,75
0,75 x 6 = 4,5 0,043
0,5 x 6 = 3,0 $\mathbf{45,125_{10}} = \mathbf{113,043_{6}}$

Conversion de la base octale à la base hexadécimale

$\mathbf{53,25_8}$ == convertir $\mathbf{?_{16}}$

53,25 = 101 011 , 010 101

= 0010 1011 , 0101 0100

= 2 B , 5 4

= $\mathbf{2B,54_{16}}$

$\mathbf{45,125_{10}}$ = $\mathbf{2B,54_{16}}$

Conversion de la base binaire à la base 4

$\mathbf{111101,100011_2}$ == convertir $\mathbf{?_4}$

111101,100011 = 11 11 01 , 10 00 11

= 3 3 1, 2 0 3

= $\mathbf{331,203_4}$

$\mathbf{111101,100011_2}$ = $\mathbf{331,203_4}$

Conversion de la base binaire à la base 6

$\mathbf{1111,1011_2}$ == convertir $\mathbf{?_6}$

$\mathbf{1111,1011}$ = 1 x $\mathbf{2^3}$ + 1 x $\mathbf{2^2}$ + 1 x $\mathbf{2^1}$ + 1 x $\mathbf{2^0}$ + 1 x $\mathbf{2^{-1}}$ + 0 x $\mathbf{2^{-2}}$ + 1 x $\mathbf{2^{-3}}$ + 1 x $\mathbf{2^{-4}}$

= 8 + 4 + 2 +1 + 0,5 + 0,125 + 0,625

= $\mathbf{15,6875_{10}}$

15	6	2	6	
12	2	0	0	23
3		2		

0,6875 x 6 = 4,125

0,125x6=0,75 0,4043

0,75 x 6 =4,5

0,5x6=3,0

$\mathbf{1111,1011_2}$ = $\mathbf{23,4043_6}$

1.3 Opérations arithmétiques

1 1 0 0 1 1 , 1 1 0 1 + 1 0 1 , 1 0 1 = ?

1 1 1

1 1 0 0 1 1 , 1 1 0 1

\+ 1 0 1 , 1 0 1

---

1 1 1 0 0 1 , 0 1 1 1

Division de 1 1 0 1 1 0 0 1, 1 1 0 1 par 1 0 1, 1

```
1 1 0 1 1 0 0 1 1 , 1 0 1   |  1 0 1 1
1 0 1 1                     |-----------------
0 0 1 0 1 0 0               | 1 0 0 1 1 1 , 1 0 0
    1 0 1 1                 |
  0 1 0 0 1 1               |
    1 0 1 1                 |
  0 1 0 0 0 1               |
      1 0 1 1               |
    0 0 1 1 0 1             |
      1 0 1 1               |
      0 0 1 0 0 1           |
```

Division de 1 0 0 1 1 1 1 0 1 1 , 1 1 0 1 par 1 1, 1 0 1

```
1 0 0 1 1 1 1 0 1 1  1 1 0 , 1   |  1 1 1 0 1
  1 1 1 0 1                      |-----------------
  0 1 0 1 0 1 0                  | 1 0 1 0 1 1 1 1 , 0 1 1
    1 1 1 0 1                    |
    0 1 1 0 1 1 1                |
      1 1 1 0 1                  |
      1 1 0 1 0 1                |
       1 1 1 0 1                 |
       1 1 0 0 0 1               |
        1 1 1 0 1                |
        1 0 1 0 0 0              |
         1 1 1 0 1               |
         0 1 0 1 1 1 0           |
           1 1 1 0 1             |
           1 0 0 0 1             |
```

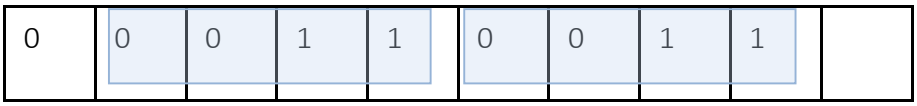
Exemple :

Convertir $\mathbf{0,1_{10}}$ == binaire ?

0,1 x 2 = 0,2
0,2 x 2 = 0,4
0,4 x 2 = 0,8
0,8 x 2 = 1,6
0,6 x 2 =1,2
0,2 x 2 = 0,4
0,4 x 2 = 0,8
0,8 x 2 = 1,6
0,6 x 2 =1,2

$\mathbf{0,1_{10}}$ n'a pas une représentation binaire exacte sur n bit (n fini).

Remarque : On fait des approximations pour représenter des nombres décimaux en binaire.



Il n'existe pas de virgule au niveau de la représentation dans l'ordinateur : les nombres fractionnaires sont gérés comme des entiers et c'est le programmeur qui doit gérer la virgule (qui n'apparaît pas dans le stockage du nombre) et faire évoluer sa place au cours des opérations (On parle de **virgule virtuelle**).

La virgule fixe n'est pas efficace pour gérer les calculs avec le maximum de précision possible car il faut sans cesse apprécier les ordres de grandeur des résultats intermédiaires en gardant le maximum de chiffres significatifs tout en évitant les débordements de capacité. C'est pourquoi on préfère en général utiliser **l'arithmétique en virgule flottante**.

2- Nombres réels représentation en virgule flottante

$$N = \pm 1, M \times 2^{E}$$

1, M s'appelle la Mantisse du nombre.
E : l'exposant.
On appelle la partie fractionnelle M la pseudo mantisse.
Il y a **deux** principaux types de nombres en virgule flottante :

1- Simple précision sur 32 bits
   pseudo mantisse 23 bits (2^{-1} … … $\mathbf{2^{-23}}$) un bit de signe et 8 bits d'exposant.
2- Double précision sur 32 bits.
   pseudo mantisse 52 bits ($\mathbf{2^{-1}}$ … … $\mathbf{2^{-52}}$) un bit de signe et 11 bits d'exposant.

La norme IEEE 754 définit la façon de coder un nombre réel. Cette norme se propose de coder le nombre sur 32 bits et définit trois composantes :

• le signe est représenté par un seul bit, le bit de poids fort

• l'exposant est code sur les 8 bits consécutifs au signe

• la pseudo mantisse (les bits situes après la virgule) sur les 23 bits restants.

Représentation 1

$$N = \pm 1, M \times 2^{E}$$

Représentation 2

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

$$(-1)^S \times 2^{E-127} \times (1+F)$$

- S est le bit de signe et l'on comprend alors pourquoi 0 est positif (-1^0=1),
- E est l'exposant auquel on doit bien ajouter 127 pour obtenir son équivalent code,
- F est la partie fractionnaire.

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 des erreurs, on appelle alors cette configuration du nombre NaN, ce qui signifie ≪ Not a number ≫.
- il faut rajouter 127 (01111111) a l'exposant pour une conversion de décimal vers un nombre réel binaire.

Méthode 1 :

Traduisons en binaire format flottant simple précision 32 bits (float) le nombre :

X = $(-8,625)_{10}$

Etape 1 : Représentons la valeur sous le format N = ± 1, M × 2^E

$8,625_{décimal} = 1000,1010_{binaire}$

Etape 2 : Décale la virgule de 3 chiffres vers la gauche

$1000,1010 = 1,0001010$

Etape 3 : La partie fractionnaire sur 23 bits = **0001010 0000 0000 0000 0000**

(on ne mémorise pas le 1 implicite d'avant la virgule)

Etape 4 : Le décalage IEEE en simple précision 8 bits : 2^{8-1} - 1 = 127

L'exposant : exposant = 3 + décalage = 3 + 127 = 130

$130_{10} = 10000010_2$

bit de signe	exposant	partie fractionnaire 23 bits
1	10000010	0001010 0000 0000 0000 0000

Méthode 2 :

X = $(-8,625)_{10}$

Etape 1 : Représentons la valeur sous le format N = ± 1, M × 2^E

$8,625_{10} = 1,078125 \times 2^3{}_{10}$

Etape 2 : Représenté 0,078125

0,078125 × 2 = 0,15625 0

0,15625 × 2 = 0,3125 0
0,3125 × 2 = 0,625 0
0,625 × 2 = 1,25 1
0,25 × 2 = 0,5 0
0,5 × 2 = 1 1 0001010

Etape 3 : La partie fractionnaire sur 23 bits = **0001010 0000 0000 0000 0000**

Etape 4 : Le décalage IEEE en simple précision 8 bits : 2^{8-1} - 1 = 127

L'exposant : exposant = 3 + décalage = 3 + 127 = 130

$130_{10} = 10000010_2$

bit de signe	exposant	partie fractionnaire 23 bits
1	10000010	0001010 0000 0000 0000 0000

$(-1)^S \times 2^{E-127} \times (1+F)$

Décalage (ou biais) = 2^{n-1}

8 bits : 2^{8-1} - 1 = 127

Exemple 1 : convertir -32,75 ?

- S = 1
- 0,75 × 2 = 1,5

0,5 × 2 = 1

|-32,75| = 32,75 = 10 0000,11

- 32,75= $(1{,}0000011)_2 \times 2^5$

M = $00000110 \ldots 0_2$ et e = 5

- E = e + biais = 5 + 127 = 4 + 128

E = $1000\ 0100_2$

- -32,75 → 1 10000100 0000011 0000000000000000

Exemple 2 : convertir 18,125?

- S = 0
- 0,125 × 2 = 0,25

0,25 × 2 = 0,5

0,5 × 2 = 1

|18,125| = 18,125 = 1 0010,0012

- 18,125 = $(1{,}0010001)_2 \times 2^4$

M = $00100010 \ldots 0_2$ et e = 4

- E = e + biais = 4 + 127 = 3 + 128

E = $\mathbf{1000\ 0011_2}$

• 18,125 → 0 10000011 00100010000000000000000

3- Décimaux codés en binaire (BCD)

BCD (Binary Coded Decimal)

Parfois au lieu d'utiliser une arithmétique binaire, certaines machines, comme par exemple certaines calculatrices de poche, utilisent une arithmétique décimale. Dans ce cas, les opérations sont effectuées directement sur la représentation décimale des nombres. Chaque chiffre (de 0 à 9) composant le nombre est codé à l'aide de bits. Il existe différents codes :

• BCD (Binary Coded Decimal),
• Excédent-3,
• 2 dans 5,
• etc.

BCD	Décimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9

Décimal => 59
Binaire => 111011
BCD => 0101 1001

4- Données alphanumériques

Codes usuels :

ASCII (American Standard Code for Information Interchange)

- standard : 7 bits
- étendu : 8 bits

ANSI (American National Standard Institute)

- basés sur le code ASCII
- utilisé par Windows Utilisation d'un code page pour designer les extensions

EBCDIC (Extended Binary Coded Decimal Interchange Code):

8 bits similaire à ASCII, utilisé essentiellement par IBM.

4.1 Code ASCII standard : 7 bits

Le code ASCII de base représentait les caractères sur 7 bits (c'est-à-dire 128 caractères possibles, de 0 à 127). Le huitième bit est un bit de signe.

MSB / LSB		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

Exemple : de code ASCII

W = 57 Hexadécimale

W = 101 0111 Binaire

W = 87 Décimale

Les codes 0 à 31 sont des caractères de contrôle car ils permettent de faire des actions telles que :

CR: le retour à la ligne

BEL: un Bip sonore

4.2 Code ASCII étendu : 8 bits

LSB \ MSB		8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111
0	0000	Ç	É	á	▒	└	ð	Ó	-
1	0001	ü	æ	í	▓	┴	Ð		±
2	0010	é	Æ	Ó	▓	┬	Ê	ô	_
3	0011	â	ô	ú	│	├	Ë	ò	¾
4	0100	ä	ö	ñ	┤	─	È	ő	¶
5	0101	à	ò	Ñ	Á	┼	I	Ő	§
6	0110	å	û	ª	Â	ã	í	µ	÷
7	0111	ç	ù	º	À	Ã	î	þ	¸
8	1000	ê	ÿ	¿	©	╚	ï	Þ	°
9	1001	ë	Ö	®	╣	╔	┘	Ú	¨
A	1010	è	Ü	¬	║	╩	┌	Û	·
B	1011	ï	ø	½	╗	╦	▉	Ù	¹
C	1100	î	£	¼	╝	╠	▄	ý	³
D	1101	ì	Ø	¡	¢	=	¦	Ý	²
E	1110	Ä	×	«	¥	╬	Ì	¯	■
F	1111	Å	ƒ	»	┐	¤	▀	´	

Le code ASCII étendu. Il permet le codage de caractères sur 8 bits, soit 256 caractères possibles.

Différence entre :

Nombre 12 = $\mathbf{12_{10}}$

Texte ″12″ ou ‘12’ = $\mathbf{49\ 50_{10}}$

Il faut différencier entre

Nombre 12 = $\mathbf{12_{10}}$

= 0000 0000 0000 $\mathbf{1100_2}$

Texte ″12″ ou ‘12’ = $\mathbf{49\ 50_{10}}$

3 1 3 $\mathbf{2_{16}}$

= 0011 0001 0011 $\mathbf{0010_2}$