Cours 1 - Système de numération et arithmétique binaire

Halim Djerroud

révision : 1.0

Introduction

Pour manipuler, afficher ou transmettre des nombres en utilisant des circuits électroniques, il est nécessaire de représenter chaque symbole par un état différent du circuit.

- Il faut un circuit à dix états pour représenter les symboles de la base dix, de 0 à 9

Problème :

- Il est difficile et excessivement cher de fabriquer des circuits à dix états

Implication :

- Il faut chercher à utiliser un système de numération avec peu de symboles

Pourquoi on compte jusqu'à 10 ?

- Deux mains, dix doigts

Les symboles de la base 10

- Symboles de la base 10 $= \underbrace{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}_{10 \text{ symboles}}$

Exemple :

0	10	.	90	100	1000
1	11	.	91	101	1001
2	12	.	92	102	1002
3	13	.	93	...	...
4	14		94	199	
5	15		95	200	
6	16		96	...	
7	17		97	999	
8	18		98		
9	19		99		

La base décimale

Exemple :

$$(6810)_{10} = (6_3 8_2 1_1 3_0)_{10} = 6 \times 10^3 + 8 \times 10^2 + 1 \times 10^1 + 3 \times 10^0$$

Formule :

$$(nombre)_b = \sum_{i=0}^{n-1} a_i \times b^i$$

- La formule permet de convertir un nombre de n'importe quelle base en base dix

Base Binaire

En informatique le nombre binaire s'écrit : b101010 (le nombre commence par b)

- Symboles de la base binaire = $\underbrace{0, 1}_{2 \text{ symboles}}$

Exemple :

0	1001
1	1010
10	1011
11	1100
100	1101
101	1110
110	...
111	
1000	

Octet

- Un octet est formé de 8 bits
- Le bit le plus à droite est appelé bit du poids faible
- Le bit le plus à gauche est appelé bit du poids fort
- L'octet est représenté avec le signe Ø

poids fort $1_7 0_6 1_5 0_4 0_3 1_2 1_1 1_0$ poids faible

Les unités de capacité

- Le bit (noté petit b)
- L'octet = 2^3 bits = 8 bits. (noté 1 $\emptyset$)
- Le Kilo-octet = 2^{10} octets = 1024 $\emptyset$ (noté 1 $K\emptyset$)
- Le Mega-octet = 2^{20} octets = 1024^2 $\emptyset$ (noté 1 $M\emptyset$)
- Le Giga-octet = 2^{30} octets = 1024^3 $\emptyset$ (noté 1 $G\emptyset$)
- Le Tera-octet = 2^{40} octets = 1024^4 $\emptyset$ (noté 1 $T\emptyset$)

Conversion de la base binaire en base 10

Exemple :

$$(1011)_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

Formule :

$$(nombre)_b = \sum_{i=0}^{n-1} a_i \times b^i$$

$$(1011)_2 = (11)_{10}$$

Conversion de la base binaire en base 10

Exercice :

Exprimer en base décimale les nombres binaires suivants :

- $(1100)_2 = (?)_{10}$
- $(10101010)_2 = (?)_{10}$
- $(1010101)_2 = (?)_{10}$
- $(1111111)_2 = (?)_{10}$
- $(110011001100)_2 = (?)_{10}$

Base Octale

En informatique le nombre octal s'écrit : 04532 (le nombre commence par O (zero))

Les symboles de la base Octale ou Base 8

- Symboles de la base Octale (8) $= \underbrace{0, 1, 2, 3, 4, 5, 6, 7}_{8 \text{ symboles}}$

Exemple :

0	11
1	12
2	13
3	14
4	15
5	16
6	17
7	20
10	...

Conversion de la base Octale en base 10

Exemple :

$$(04752)_8 = 4 \times 8^3 + 7 \times 8^2 + 5 \times 8^1 + 2 \times 8^0$$

Formule :

$$(nombre)_b = \sum_{i=0}^{n-1} a_i \times b^i$$

$$(04752)_8 = (?)_{10}$$

Conversion de la base Octale en base 10

Exercice :

Exprimer en base décimale les nombres octaux suivants :

- $(02457)_8 = (?)_{10}$
- $(0421561)_8 = (?)_{10}$
- $(0101010)_8 = (?)_{10}$
- $(0441243)_8 = (?)_{10}$
- $(010011001100)_8 = (?)_{10}$

Base Hexadécimale

En informatique le nombre Hexadécimal s'écrit : 0x8A5F3 (le nombre commence par 0x (zero x))

Les symboles de la base Hexadécimale ou Base 16

- Symboles de la base Hexadécimale (16) $= \underbrace{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}_{16 \text{ symboles}}$

Exemple :

0	E
...	F
9	10
A	11
B	...
C	
D	

Conversion de la base Hexadécimale en base 10

Exemple :

$$(0x1D5B)_{16} = 1 \times 16^3 + 13 \times 16^2 + 5 \times 16^1 + 11 \times 16^0$$

Formule :

$$(nombre)_b = \sum_{i=0}^{n-1} a_i \times b^i$$

$$(0x1D5B)_{16} = (?)_{10}$$

Conversion de la base Hexadécimale en base 10

Exercice :

Exprimer en base décimale les nombres Hexadécimaux suivants :

- $(A4BC)_{16} = (?)_{10}$
- $0x12345 = (?)_{10}$
- $0x1BD4FC3 = (?)_{10}$
- $0xCCCCCC = (?)_{10}$
- $0xDFA34CDFFF = (?)_{10}$

Conversion entre bases

Nous allons utiliser une méthode simple et rapide pour résoudre les problèmes de conversion.
Supposons que nous voulons convertir en base b un nombre N donnée en base a.
La méthode consiste à utiliser une succession de divisions en base a où $N_i(i = 1, 2, \cdots, n)$ représente le quotient de la division, b le diviseur et $R_j(j = 0, 2, \cdots, n)$ le reste de la division.

Conversion entre bases

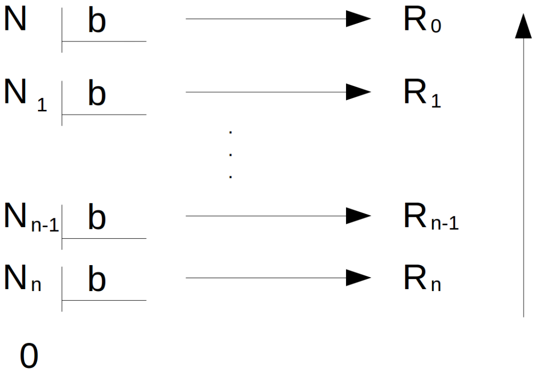

FIGURE – Conversion

Conversion de la base Décimale à la base Binaire

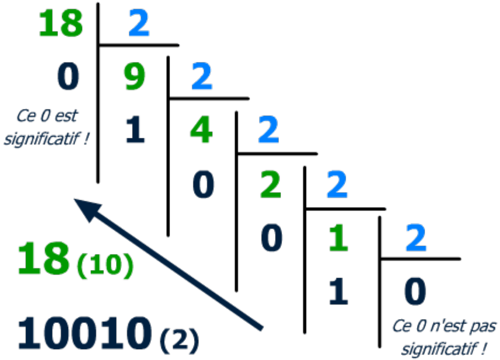

FIGURE – Conversion décimal - binaire

Astuce : Conversion entre bases

1er cas : la base origine n'est pas une puissance de la base cible, par exemple convertie de la base hexadécimal vers la base décimale :

- Effectuer des divisions successives

2er cas : la base origine est une puissance de la base origine, par exemple convertie de la base hexadécimal vers la base binaire. Ici on remarque la que la base hexadécimale (16) peut s'écrire sous la forme $2^{\underline{4}} = 16$, dans ce cas :

- Il suffi de convertir chaque nombre de la base origine dans la base cible individuellement sur 4 bits

$$Exemple : 0xA5 => (1010\,0101)_2 = (A\,5)_{16}$$

Les unités de capacité

- Le bit (noté petit *b*)
- L'octet = 2^3 bits = 8 bits. (noté 1 $\varnothing$)
- Le Kilo-octet = 2^{10} octets = 1024 $\varnothing$ (noté 1 $K\varnothing$)
- Le Mega-octet = 2^{20} octets = 1024^2 $\varnothing$ (noté 1 $M\varnothing$)
- Le Giga-octet = 2^{30} octets = 1024^3 $\varnothing$ (noté 1 $G\varnothing$)
- Le Tera-octet = 2^{40} octets = 1024^4 $\varnothing$ (noté 1 $T\varnothing$)

Les unités de capacité

Les préfixes métriques dans système international d'unités (SI) k, M, G signifiaient :

- $k = 10^3$
- $M = 10^6$
- $G = 10^9$

Or en informatique $1k = 1024\emptyset \ldots$

A partir de 1998, le SI a clarifié la différence :

- $k = 1000$, $ki = 1024$
- $M = 1000000$, $Mi = 1048576$
- $G = 1000000000$, $Gi = 1073741824$

Addition binaire

$$\begin{array}{rr} & 0 \\ + & 0 \\ \hline 0 & 0 \end{array} \qquad \begin{array}{rr} & 1 \\ + & 0 \\ \hline 0 & 1 \end{array}$$

$$\begin{array}{rr} & 0 \\ + & 1 \\ \hline 0 & 1 \end{array} \qquad \begin{array}{rr} & 1 \\ + & 1 \\ \hline 1 & 0 \end{array}$$

Addition binaire

Exemple : Additionnez les deux nombres binaire :

$11101101 + 10111001 = ?$

		1	1	1	1			1		
		1	1	1	0	1	1	0	1	(237)
+		1	0	1	1	1	0	0	1	(185)
	1	1	0	1	0	0	1	1	0	(422)

Exercice

Exercice : Additionnez les deux nombres binaires suivants :

- $10101010 + 10101010$
- $11111111 + 10000000$
- $11111111 + 00111111$
- $01111111 + 01111111$

Soustraction binaire

La soustraction en binaire est faite de la même manière qu'en base décimale. En additionnant un nombre positif à un nombre négatif. Le nombre dont la valeur absolue est plus petite est soustrait du nombre dont la valeur absolue est plus grande et le signe de plus grand nombre est affecté au résultat.

Soustraction binaire

Exemple : Soustraire les deux nombres binaire :
$11001010 - 10101001 = ?$

	1	1	$_1$0	0	1	0	1	$_1$0	(202)
-	1	$_1$0	1	0	1	0	$_1$0	1	(169)
	0	0	1	0	0	0	0	1	(33)

Attention : Cette méthode est à titre explicatif, ce n'est pas la méthode utilisée par la machine.

Avantages et problèmes des nombres négatifs

Pour implémenter les nombres négatifs, il faut représenter le signe (-), or dans un ordinateur, nous pouvons uniquement écrire des 0 ou des 1. Donc la solution consiste à désigner le bit le plus significatif comme étant le bit du signe : 1 pour (-) et 0 pour (+).

Avantages et problèmes des nombres négatifs

Avantage :

- Si on peut représenter les nombres négatifs alors on peut transformer les soustraction en addition, par exemple $10 - 30$ peut ainsi s'écrire $10 + (-30)$
- Un ordinateur ne distingue pas entre une addition et une soustraction. Ainsi, le circuit utilisé pour l'opération d'addition peut être utilisé pour l'opération de soustraction grâce au complément à 2.

Avantages et problèmes des nombres négatifs

Problème :

- Si l'ordinateur propose de mettre un bit pour distinguer les nombres positifs et négatifs alors le nombre Zéro 0 aura deux représentations -0 et $+0$
- Les nombres doivent avoir le même nombre de bits
- Il y a un risque de débordement de capacité

Solution : complément à 2

Complément à 2 d'un nombre positif est lui même : Donc rien à faire

---

Complément à 2 d'un nombre négatif est :

Complément à 2

$$Complment\ \ 2\ (N) = Complment\ \ 1(N) + 1$$

Complément à 1

$$Complment\ \ 1(N) = inverse(N)$$

Complément : Inverser les bits : 1 => 0 et 0 => 1

Complément à 2

Exemple :

Trouvez le complément à 2 de : $N = 10101001$

C1 :

$$
\begin{array}{cccccccccc}
\sim & & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
\hline
 & & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0
\end{array}
$$

C2 :

$$
\begin{array}{cccccccccc}
 & & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\
+ & & & & & & & & & 1 \\
\hline
 & & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1
\end{array}
$$

Complément à 2

Exemple de soustraction : 10 − 12 sur 5 bits

-Donc on peut écrire : $10 + (-12)$

-Pour effectuer ce calcul il suffi d'additionner 10 avec le C2 de 12

10 en binaire sur 5 bits = 01010

-12 en binaire sur 5 bits "$C2(12) = C1(12) + 1$" :

12 s'écrit 01100

$C1(12)$ s'écrit 10011

$C2(12)$ s'écrit 10100

$$
\begin{array}{rccccc}
 & 0 & 1 & 0 & 1 & 0 \\
+ & 1 & 0 & 1 & 0 & 0 \\
\hline
 & 1 & 1 & 1 & 1 & 0
\end{array}
$$

Complément à 2

Donc :

$$\begin{array}{cccccc} & 0 & 1 & 0 & 1 & 0 \\ + & 1 & 0 & 1 & 0 & 0 \\ \hline & 1 & 1 & 1 & 1 & 0 \end{array}$$

Ici on remarque le résultat est négatif car le dernier bit est égale à 1

Pour trouver le résultat sans le signe, il faut appliquer au résultat un complément à 2

Donc :

$C2(11110) = 00001 + 1 = 00010 = (2)_{10}$

Résultat en décimal : $10 - 12 = -2$

Complément à 2

- Le calcul en complément à 2 se fait sur n bits, si une retenue est générée à la fin de l'addition (par exemple le résultat de l'addition est sur $n+1$ bits) alors la retenue est tout simplement ignorée

Débordement

Les opérations signées utilisent le dernier bit pour représenter le signe du nombre, donc pour n bits, il est possible de représenter les valeurs suivantes :

- Nombres signés sur n bits : $[-2^{n-1}, 2^{n-1} - 1]$

Les opérations signées peut engendre des débordement de capacité (noté V pour *overflow*), il est possible de détecter le débordement en appliquant la formule suivante :

$$Pour\ l'operation\ :\ s = a + b$$

$$V = \bar{a}_{n-1} * \bar{b}_{n-1} * s_{n-1} + a_{n-1} * b_{n-1} * \bar{s}_{n-1}$$

V : est une expression booléen, si V=1 alors i y a débordement

Exercice complément à 2

Exercice : Effectuez les opérations suivantes en binaire (sur 8 bits) :

- $125 - 117$
- $0xA5 - 0xA4$
- $b10100101 + b0001111$
- $054 - 015$

Codage des caractères

Les différents type codages :

- Code BCD (Binary Coded Decimal)
- Code 7 segments
- Code ASCII (American Standard Code for Information Interchange)
- UTF

Code ASCII

	30	40	50	60	70	80	90	100	110	120
0:		(	2	<	F	P	Z	d	n	x
1:		)	3	=	G	Q	[	e	o	y
2:		*	4	>	H	R	\	f	p	z
3:	!	+	5	?	I	S	]	g	q	{
4:	"	,	6	@	J	T	^	h	r	|
5:	#	-	7	A	K	U	_	i	s	}
6:	$	.	8	B	L	V	`	j	t	~
7:	%	/	9	C	M	W	a	k	u	DEL
8:	&	0	:	D	N	X	b	l	v	
9:	'	1	;	E	O	Y	c	m	w	