Module 2: Représentation des données (suite)

1 Introduction

Nous avons vu dans la leçon précédente, comment représenter des nombres à savoir les nombres entiers, les nombres fractionnaires et les nombres négatifs. Nous avons également abordé le concept de changement de base et nous avons données des exemples de passage d'une base quelconque à la base 10 et inversement.

Dans cette leçon, nous allons continuer avec la représentation des données. Nous allons aborder la représentation des caractères et la représentation des nombres réels. Nous finirons cette leçon avec la présentation des opérations arithmétiques binaires à savoir l'addition, la multiplication et la soustraction.

2 Représentation des caractères

Un caractère peut désigner un chiffre (1, 2, 3, etc.), une lettre en minuscule (a, b, f) ou majuscule (A, B, Q), et un symbole de ponctuation ou mathématique ($:, ; , !, ?, , >, =, etc.$). Un texte par exemple est représenté par une suite de caractères.

Il existe des standards pour coder les caractères à savoir le code américain normalisé pour l'échange d'information ASCII (American Standard Code for Information Interchange)[1] qui permet le codage de caractères sur 8 bits (version étendue).Le code ASCII est le code le plus utilisé. Il définit 128 (2^7) caractères numérotés de 0 (0000 0000) à 127 (1111 1111), donc chaque caractère est représenté avec 7 bits. Vu que les ordinateurs fonctionnent des représentations en puissance de 2, on ajoute un 0 au code de 7 bits pour avoir 8 bits.

Chaque plage de valeurs représentent des catégories particulières de caractères. Par exemple:

- Les codes de 0 à 31 et le code 127 ce sont des codes non affichables. Ils sont réservés pour des opérations de contrôle d'où leur nom (caractères de contrôle) à savoir le retour chariot (CR). Le code 127 représente la commande pour effacer.
- Les codes de 65 à 90 représentent les lettres en ordre alphabétique en majuscules.

[1] `https://fr.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange`

Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

Table 1: Les codes ASCII des différents caractères.

- Les codes de 97 à 122 représentent les lettres en ordre alphabétique en minuscules. Notez que le passage du majuscule au minuscule s'effectue en replaçant le cinquième bit, cela veut dire qu'on ajoute 32 au code ASCII décimal. Par exemple, le code 65 représente la lettre A en majuscule, et le code $(65 + 32 = 97)$ représente la lettre a en minuscule.
- Les codes de 48 à 57 représentent les chiffres dans un ordre croissant de 0 à 9. Par exemple, 52 représente le chiffre 4 avec la représentation binaire $(0110100)_2$.

Un tableau montrant les codes ASCII est présenté dans la figure suivante[2]:

[2]Figure tirée des images de Google.

3 Représentation des nombres réels

La norme IEEE exige une normalisation lors de la représentation en base 2 de telle sorte que la représentation puisse s'écrire selon la formule suivante:

$$(-1)^{Signe}.Mantisse.2^{\pm Exposant} \tag{1}$$

Cette formule est appelée aussi la notation scientifique. Cette formule est composée de trois parties:

1. Le signe du nombre, représenté par un seul bit (1 pour le + et 0 pour le -)
2. La mantisse
3. L'exposant

On distingue deux représentations selon la norme IEEE 754, soit une représentation en simple précision (sur 32 bits) et une représentation en double précision (sur 64 bits).

Exemple 3.1. Si on veut représenter le nombre $(-18,75)_{10}$ en binaire avec précision simple 32 bits on procède comme suit[3] :

- $(18)_{10} = (10010)2$
- $(0,75)_{10} = (0,110)2$
- Donc: $(18,75)_{10} = (10010,110)_b$
- On décale la virgule à gauche jusqu'au dernier bit valant 1. Dans ce cas, on compte 4 bits pour arriver au dernier bit valant 1. Par conséquent, l'exposant dans ce cas égal à 4. Chaque décalage vers la gauche représente une division par 2.
- On convertit l'exposant, qui est égal à 4 en décimal, en binaire ce qui donne $(4)_{10} = (100)_2$.
- Une fois toutes les composantes sont disponibles, on peut maintenant appliquer la formule: $(10010,110)_b = 1,0010110_b \times 2^{100_b}$. Les trois composantes sont donc:
  1. Signe: Négatif (1)
  2. Exposant: 4 (100_b). On faut toujours ajouter 127 (01111111) à l'exposant pour une conversion de décimal vers un nombre réel binaire. Dans ce cas: $4 + 127 = (10000011)_2$.
  3. Mantisse: 0010110. Pour avoir le nombre total de bits (32 bits dans ce cas), on complète la mantisse par des 0. Cela donne: 001 0110 0000 0000 0000 0000 (23 bits pour la mantisse).

[3] Exemple tiré du `http://php.iai.heig-vd.ch/~lzo/micro/ex/Corrige_2.pdf`

Donc, pour résumer: -18,75 s'écrit en nombre binaire avec précision simple comme:

1(signe) 10000011(exposant) 001 0110 0000 0000 0000 0000(mantisse).

Avec une précision double (64 bits), le signe s'écrit toujours avec 1 bit, l'exposant s'écrit avec 11 bits, et la mantisse avec 52 bits (total donne 64 bits).

Le tableau suivant présente le nombre de bits dans chaque composante avec les différents niveaux de précision.

Précision	Taille	Signe	Exposant	Mantisse
Simple	32 bits	1 bit	8 bits	23 bits
Double	64 bits	1 bit	11 bits	52 bits
Quadruple	128 bits	1 bit	15 bits	112 bits

Table 2: Tableau récapitulatif des nombres de bits dans chaque composante par niveau de précision.

- Pour obtenir l'exposant avec une simple précision on ajoute 127 (01111111_2), et pour une double précision, on ajoute 1023 (01111111111_2).
- Pour obtenir la mantisse, on garde seulement les bits après la virgules et on complète à droite par des 0.
- L'exposant 00000000 est interdit.
- L'exposant 11111111 est interdit.

4 Arithmétique binaire

Nous avons vu dans les sections précédentes comment coder les nombres en binaire. Dans cette section, nous allons présenter quelques exemples sur comment effectuer des opérations arithmétiques comme l'addition, la soustraction et la multiplication avec des nombres binaires.

4.1 Addition

L'addition des nombres binaires s'effectue de la même façon que l'addition des nombres décimaux qu'on connaît habituellement. La simplicité de l'addition des nombres binaires provient du fait qu'on manipule uniquement deux nombres 0 et 1 et donc nous avons uniquement 4 opérations possibles résumées comme suit:

1. $0 + 0 = 0$.
2. $0 + 1 = 1$.
3. $1 + 0 = 1$.
4. $1 + 1 = 0$ avec une retenue égale à 1.

L'addition s'effectue bit à bit de droite à gauche. Les retenues sont reportées toujours à gauche de l'opération courante.

Exemple 4.1. Si on veut additionner les nombres $(10101001)_2$ et $(11001011)_2$, on procède comme suit:

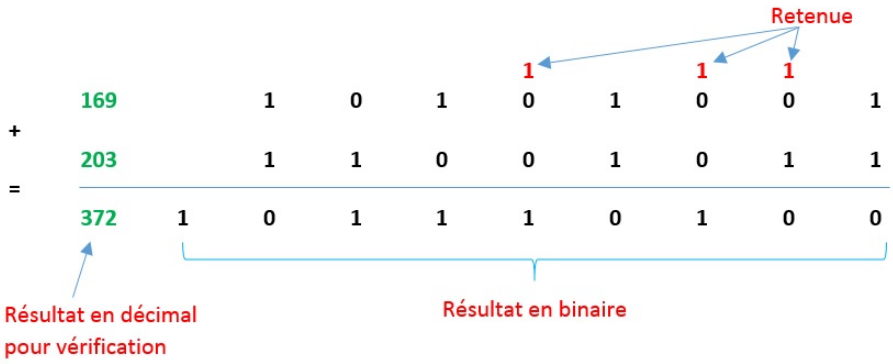

FIGURE 1 – Addition de deux nombres binaires.

Pour vérifier les résultats, nous pouvons faire l'addition des nombres décimaux correspondant aux nombres binaires soit 169 + 203 = 372.

4.2 Soustraction

L'opération de soustraction est identique à celle d'addition de deux nombres binaires. Nous avons donc 4 opérations sur les nombres binaires résumées comme suit:

1. 0 - 0 = 0.
2. 0 - 1 = 1 avec avec une retenue 1 dans la colonne suivante (à gauche).
3. 1 - 0 = 1.
4. 1 - 1 = 0.

Exemple 4.2. Si on veut soustraire le nombre $(10101001)_2$ du nombre $(11001011)_2$, on procède comme suit:

Pour vérifier les résultats, nous pouvons faire la soustraction des nombres décimaux correspondant aux nombres binaires soit 203 - 169 = 34.

La soustraction présente certaines difficultés avec les nombres binaires négatifs. C'est à dire, dans l'exemple précédent, si on soustrait le nombre $(11001011)_2$ du $(10101001)_2$ le résultat sera un nombre binaire négatif (-34 en décimal). Pour effectuer ce genre de soustraction, nous procéderons comme suit:

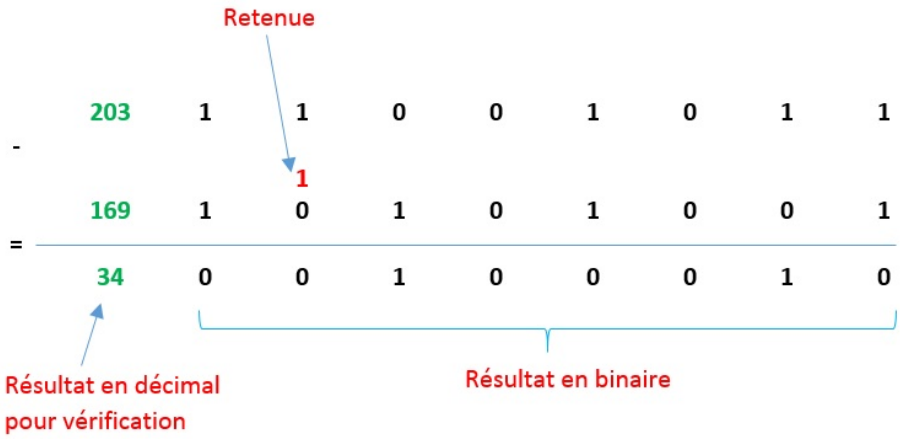

FIGURE 2 – Soustraction de deux nombres binaires.

- Calculer tout d'abord le complément à 2 du nombre binaire diminuteur $(11001011)_2$. Donc, on calcule le complément à 1 en inversant les bits (0 devient 1 et 1 devient 0), puis on ajoute 1 au résultat pour avoir le complément à 2. Donc, le complément à 1 du nombre $(11001011)_2$ donne $(00110100)_2$. On ajout 1 pour avoir le complément à 2 qui donne $(00110101)_2$.
- La soustraction revient donc à additionner le nombre binaire $(10101001)_2$ avec $(10101001)_2$ qui donne $(11011110)_2$ qui vaut -34 en décimal.

4.3 Multiplication

La multiplication binaire est encore plus simple que celle avec des nombres décimaux. Nous avons également 4 opérations possibles résumées comme suit:

1. 0 x 0 = 0.
2. 0 x 1 = 0.
3. 1 x 0 = 0.
4. 1 x 1 = 1.

L'exemple suivant présente comment la multiplication de deux nombres binaires se fait.

Exemple 4.3. Si on prend l'exemple précédent avec les deux nombres $(10101001)_2$ et $(11001011)_2$, le résultat de multiplication sera comme suit:

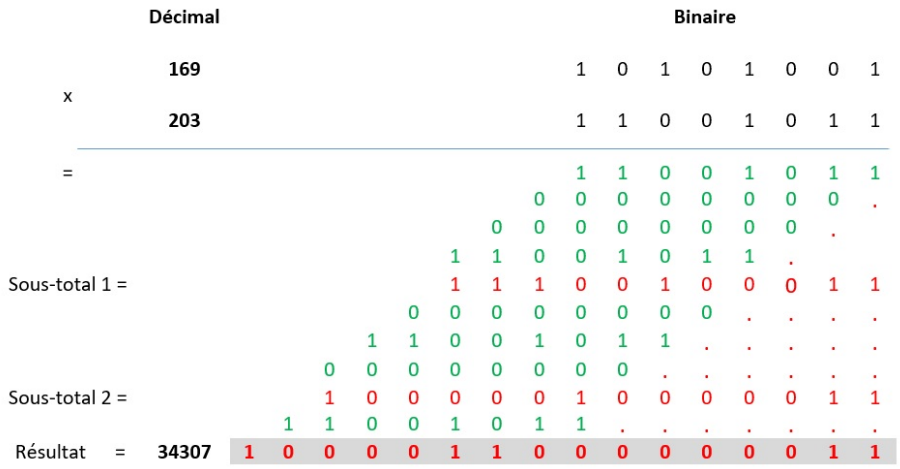

FIGURE 3 – Multiplication de deux nombres binaires.

4.4 Division

La division binaire se fait de la même façon qu'en décimal. Nous avons 2 opérations possibles résumées comme suit:

1. $0 \div 1 = 0.1 \div 1 = 1$.
   L'exemple suivant présente comment la division de deux nombres binaires se fait.

Exemple 4.4. Si on veut diviser le nombre $(1010)_2$ par le nombre $(10)_2$, on procède comme suit:

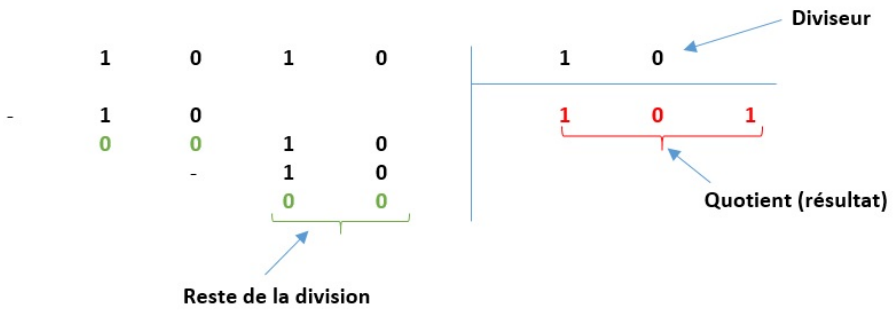

FIGURE 4 – Division de deux nombres binaires.

Comme on peut le constater à partir de la figure 4, si le diviseur est plus grand que le nombre divisé ($001 > 10$), on place un 0 à droite du quotient et on

se déplace à droite pour le reste des bits du nombre divisé un bit à la fois. Le reste de la division dans l'exemple précédent égale à 0. On peut donc vérifier le résultat de la division en décimal: $10(1010) \div 2(10) = 5(101)$ et le reste de la division est 0.