CENTRE UNIVERSITAIRE ABDELHAFID BOUSSOUF-MILA

MODULE : STRUCTURE MACHINE 1

CHAPITRE 3

LA REPRÉSENTATION DE L'INFORMATION

ENSEIGNÉ PAR : BOUMASSATA MERIEM

2022/2023

I. Représentation des nombres

1. Nombres entiers

- Problématique :

- La représentation (ou codage) des nombres est nécessaire afin de les stocker et manipuler par un ordinateur.
- Dans un ordinateur, un nombre est représenté par une succession de bits.
- Le principal problème est la limitation de la taille du codage : la valeur d'un nombre peut être arbitrairement grande, mais comme nous travaillons dans un environnement "fini" (les différentes mémoires des ordinateurs sont finies en taille),
- Le codage dans l'ordinateur doit s'effectuer sur un nombre de bits fixé. Les valeurs minimales et maximales des nombres manipulés par les machines sont donc limitées.

1. Nombres entiers (Suite)

a) Les entiers naturels (non-signés)

- Les entiers naturels (positifs ou nuls) sont codés sur un nombre d'octets fixé (rappel : un octet est un groupe de 8 bits).
- On rencontre habituellement des codages sur 1, 2 ou 4 octets, plus rarement sur 64 bits (8 octets).
- Un codage sur n bits permet de représenter tous les nombres naturels compris entre 0 et $2^n - 1$.
- Par exemple sur 1 octet, on pourra coder les nombres de 0 à 255 = $2^8 - 1$.

1. Nombres entiers (Suite)

a) Les entiers naturels (Suite)

- La représentation en machine est effectuée de la façon suivante : On représente le nombre en base 2 et on range les bits dans les cases mémoires binaires contiguës correspondant à leur poids binaire, de la droite vers la gauche. Si nécessaire, on complète à gauche par des zéros (bits de poids fort), ce qui évidemment ne change pas la valeur de l'entier.
- Exemple : l'entier n = $(144)_{10}$ est représenté par $(0000000010010000)_2$ en base deux sur 2 octets (16 bits).

  En effet, $(144)_{10} = (10010000)_2$, puis on complète à gauche par des zéros.
- Remarque : ce nombre peut être écrit sur 8 bits mais pas sur moins de 8 bits.

1. Nombres entiers (Suite)

b) Les entiers relatifs (signés)

- Les nombres entiers relatifs peuvent être positifs ou négatifs. Ils possèdent donc un signe. En machine nous devons également représenter le signe de ces nombres.
- Problème: Comment indiquer à la machine qu'un nombre est négatif ou positif ?

➢ Il existe 3 méthodes pour représenter les nombres négatifs :

– Signe/Valeur absolue.

– Complément à 1(complément restreint).

– Complément à 2 (complément à vrai).

1. Nombres entiers (Suite)

b) Les entiers relatifs (Suite)

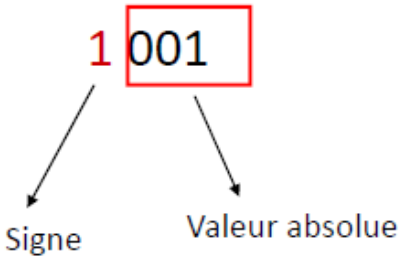
❖ Représentation Signe / Valeur absolue (S/VA)

- Si on travaille sur n bits, alors le bit du poids fort est utilisé pour indiquer le signe:

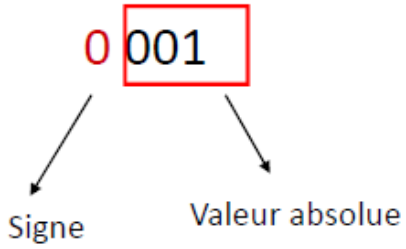
  1 : signe négatif.

  0 : signe positif.

- Les autres bits (n-1) désignent la valeur absolue du nombre.
- Exemple : Si on travaille sur 4 bits.



1001 est la représentation de -1



0001 est la représentation de + 1

1. Nombres entiers (Suite)

b) Les entiers relatifs (Suite)

- Représentation Signe / Valeur absolue (Suite)

- Sur 3 bits on obtient :

signe	VA	valeur
0	00	+ 0
0	01	+ 1
0	10	+ 2
0	11	+ 3
1	00	- 0
1	01	- 1
1	10	- 2
1	11	- 3

Les valeurs sont comprises entre -3 et +3

$-3 \leq N \leq +3$

$-(4-1) \leq N \leq +(4-1)$

$-(2^2 -1) \leq N \leq +(2^2 -1)$

$-(2^{(3-1)} -1) \leq N \leq +(2^{(3-1)} -1)$

- Sur n bits, l'intervalle des valeurs qu'on peut représenter en système en valeur absolue : $-(2^{(n-1)} -1) \leq N \leq +(2^{(n-1)} -1)$.

1. Nombres entiers (Suite)

b) Les entiers relatifs (Suite)

- Représentation Signe / Valeur absolue (Suite)
  - Avantages et inconvénients :

- Représentation assez simple .
- Le zéro possède deux représentations +0 et -0 ce qui conduit à des difficultés au niveau des opérations arithmétiques.
- Pour les opérations arithmétiques il nous faut deux circuits : l'un pour l'addition et le deuxième pour la soustraction .
- L'idéal est d'utiliser un seul circuit pour faire les deux opérations, puisque :

$$X - Y = X + (-Y)$$

1. Nombres entiers (Suite)

b) Les entiers relatifs (Suite)

- Représentation en complément à 1 (complément restreint / CA1)

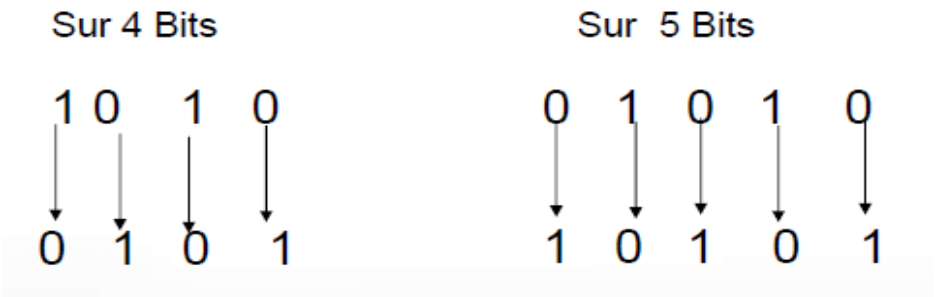
- On appel complément à un d'un nombre N un autre nombre N' tel que :

$$N + N' = 2^n - 1$$

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

- 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.

- Exemple 1 :



1. Nombres entiers (Suite)

b) Les entiers relatifs (Suite)

❖ Représentation en complément à 1 (Suite)

- Dans cette représentation , 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 2 :

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

1. Nombres entiers (Suite)

b) Les entiers relatifs (Suite)

❖ Représentation en complément à 1 (Suite)

- Sur n bits , l'intervalle des valeurs qu'on peut représenter en système en CA1 :

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

- Exemple : Sur 3 bits les valeurs sont comprises entre -3 et +3 :

  $-3 \leq N \leq +3$

  $-(4-1) \leq N \leq +(4-1)$

  $-(2^2-1) \leq N \leq +(2^2-1)$

  $-(2^{(3-1)}-1) \leq N \leq +(2^{(3-1)}-1)$

Valeur en CA1	Valeur en binaire	Valeur 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

- On remarque que dans cette représentation le zéro possède aussi une double représentation (+0 et –0) .

1. Nombres entiers (Suite)

b) Les entiers relatifs (Suite)

- Représentation en complément à 2 (complément à vrai / CA2)

- Soit **X** un nombre sur n bits alors :

$$X + 2^n = X \text{ modulo } 2^n$$

- Le résultat sur **n** bits ⇒ la même valeur que **X** :

$$X + 2^n = X$$

- Si on prend deux nombres entiers **X** et **Y** sur **n** bits , on remarque que la soustraction peut être ramener à une addition :

$X - Y = X + (-Y)$ ⇒ trouver une valeur équivalente à -Y ?

$$X - Y = (X + 2^n) - Y = X + (2^n - 1) - Y + 1$$

On a $Y + CA1(Y) = 2^n - 1$ donc $CA1(Y) = (2^n - 1) - Y$

On obtient :

$$X - Y = X + CA1(Y) + 1$$

1. Nombres entiers (Suite)

b) Les entiers relatifs (Suite)

- Représentation en complément à 2 (Suite)

- La valeur CA1(Y)+1 s'appelle le complément à deux de y :

CA1(Y)+1 = CA2(Y)

- Et enfin on va obtenir :

X - Y = X+ CA2(Y) ➡ transformer la soustraction en une addition.

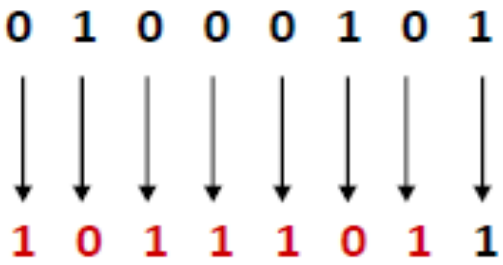
1. Nombres entiers (Suite)

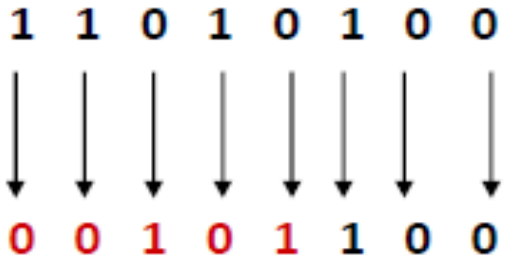
b) Les entiers relatifs (Suite)

- Représentation en complément à 2 (Suite)

- Pour trouver le complément à 2 d'un nombre : il faut parcourir les bits de ce nombre à partir du poids faible et garder tous les bits avant le premier 1 et inverser les autres bits qui viennent après.

- Exemples :





1. Nombres entiers (Suite)

b) Les entiers relatifs (Suite)

❖ Représentation en complément à 2 (Suite)

- Dans cette représentation, aussi , le bit du poids fort nous indique le signe (0 : positif , 1 : négatif).
- Le complément à deux du complément à deux d'un nombre est égale au nombre lui-même.

CA2 (CA2 (N)) = N

- Exemple : Quelle est la valeur décimale représentée par la valeur 101010 en complément à deux sur 6 bits ?

  Le bit poids fort indique qu'il s'agit d'un nombre négatif.

  Valeur = -CA2 (101010) = -$(010110)_2$= -(22)

1. Nombres entiers (Suite)

b) Les entiers relatifs (Suite)

- Représentation en complément à 2 (Suite)

- Une autre méthode consiste à rajouter 1 au complément à 1 du nombre.
- Exemple : Trouver le complément à 2 de 01000101 sur 8 bits ?

CA2 (01000101) = CA1 (01000101) + 1

= (10111010) + 1

= (10111011)

1. Nombres entiers (Suite)

b) Les entiers relatifs (Suite)

- Représentation en complément à 2 (Suite)

- Sur n bits , l'intervalle des valeurs qu'on peut représenter en CA2 :

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

- Exemple : Sur 3 bits les valeurs sont comprises entre -4 et +3 :

  $-4 \leq N \leq +3$

  $-(4) \leq N \leq +(4 - 1)$

  $-(2^2) \leq N \leq +(2^2 - 1)$

  $-(2^{(3-1)}) \leq N \leq +(2^{(3-1)} - 1)$

- On remarque que le zéro n'a pas une double représentation.

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

1. Nombres entiers (Suite)

b) Les entiers relatifs (Suite)

- Représentation en complément à 2 (Suite)

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

1. Nombres entiers (Suite)

c) Opérations arithmétiques en CA2

- Exemple 1 : Effectuer les opérations suivantes sur 5 Bits , en utilisant la représentation en CA2 :

+ 9	0 1 0 0 1
+ 4	+ 0 0 1 0 0
+ 13	0 1 1 0 1

Le résultat est positif

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

+ 9	0 1 0 0 1
- 4	+ 1 1 1 0 0
+ 5	1 0 0 1 0 1

Report

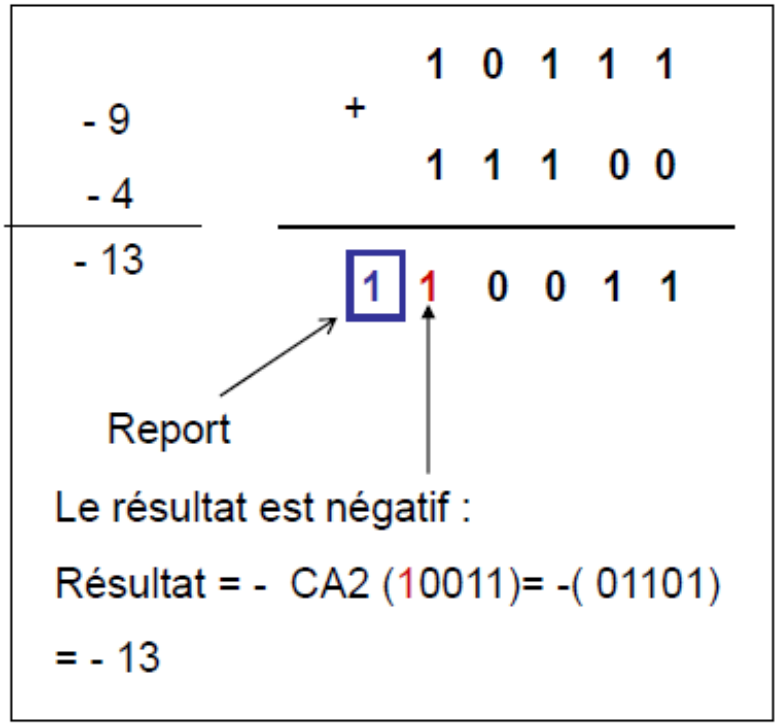
Le résultat est positif

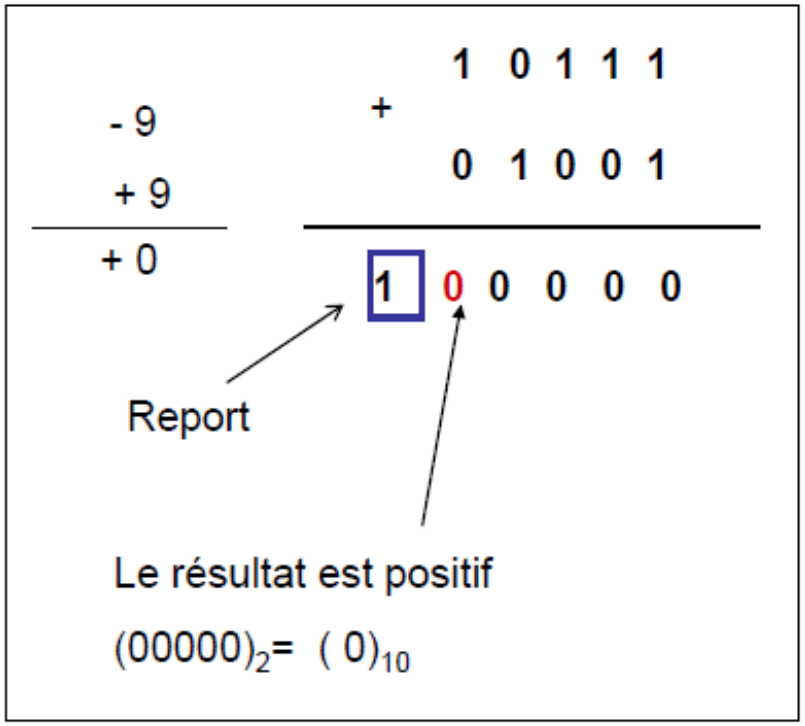
$(00101)_2 = (5)_{10}$

1. Nombres entiers (Suite)

c) Opérations arithmétiques en CA2 (Suite)

- Exemple 2 : Effectuer les opérations suivantes sur 5 Bits , en utilisant la représentation en CA2 :





1. Nombres entiers (Suite)

c) Opérations arithmétiques en CA2 (Suite)

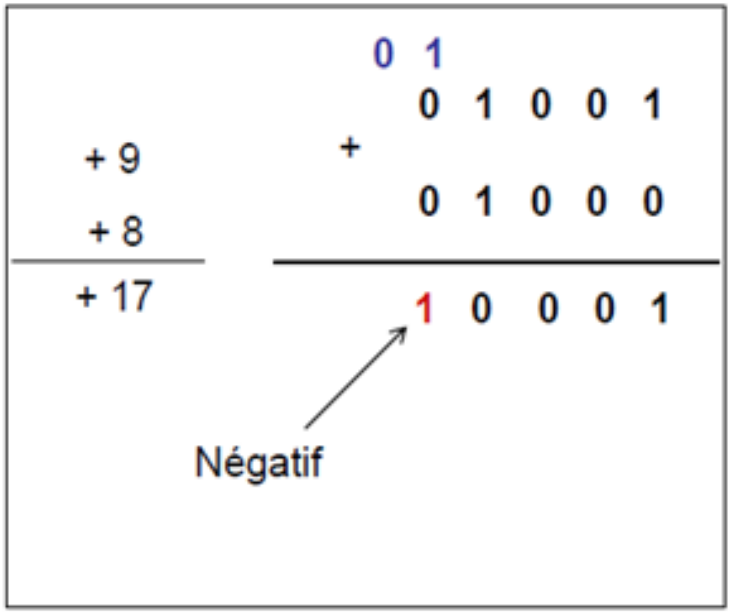
- La retenue et le débordement

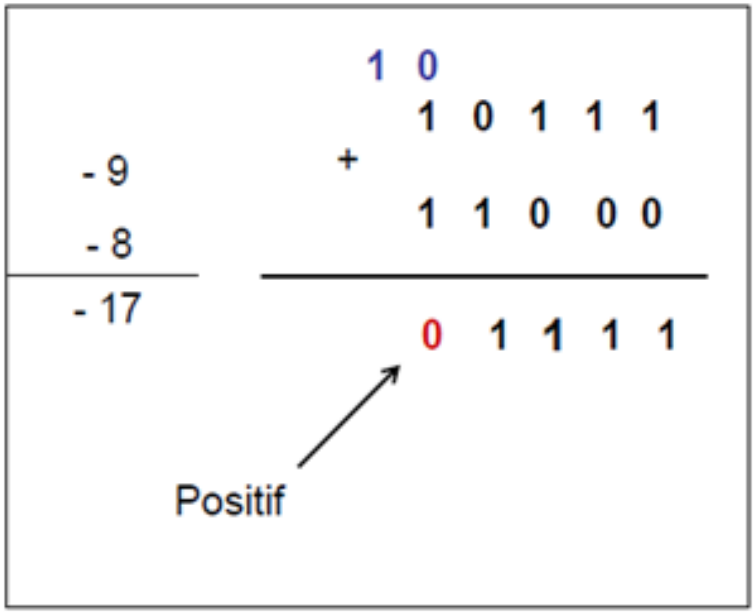
- On dit qu'il y a une retenue si une opération arithmétique génère un report.
- On dit qu'il y a un débordement (Over Flow) ou dépassement de capacité si le résultat de l'opération sur n bits est faux :
  - Le nombre de bits utilisés est insuffisant pour contenir le résultat.
  - Autrement dit le résultat dépasse l'intervalle des valeurs sur les n bits utilisés.
- Si les deux opérandes sont de même signe, le dépassement de capacité se produit quand la retenue de l'addition des bits de signes et la retenue de l'addition des bits juste avant les bits de signe sont différentes.
- Il n'y a jamais un dépassement de capacité si les deux nombres sont de signes opposés.

1. Nombres entiers (Suite)

c) Opérations arithmétiques en CA2 (Suite)

- La retenue et le débordement (Suite)
- Exemple : cas de débordement :





2. Nombres réels (fractionnaires)

- 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 nombres réels :

1) Virgule fixe : la position de la virgule est fixe.

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

2. Nombres réels (Suite)

a) Représentation en virgule fixe

- Dans cette représentation la partie entière est représentée sur n bits et la partie fractionnelle sur p bits , en plus un bit est utilisé pour le signe.
- Exemple : si n=3 et p=2 on va avoir les valeurs suivantes :

Signe	P.E	P.F	valeur
0	000	00	+ 0,0
0	000	01	+ 0,25
0	000	10	+ 0,5
0	000	11	+ 0,75
0	001	00	+ 1,0
.	.	.	.
.	.	.	.

Dans cette représentation les valeurs sont limitées et nous n'avons pas une grande précision

2. Nombres réels (Suite)

b) Représentation en virgule flottante

- L'idée est de ne pas fixer la position de la virgule : on représente un nombre par deux entiers, l'un appelé mantisse reprend les chiffres significatifs du réel sans virgule, l'autre l'exposant, donne la position de la virgule.
- En base dix, le fait de multiplier un nombre à virgule par 10^n permet de décaler la virgule de n places vers la droite. Par exemple :

  $232,56 = 23,256 \times 10^1$

  $232,56 = 2325,6 \times 10^{-1}$

- En base deux, c'est exactement la même chose. Par exemple :

  $(11,0101)_2 = (1,10101)_2 \times 2^1$

  $(11,0101)_2 = (11010,1)_2 \times 2^{-3}$

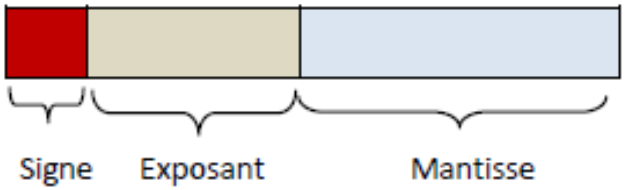
2. Nombres réels (Suite)

b) Représentation en virgule flottante (Suite)

- Normalisation

- D'une façon générale, un nombre à virgule flottante est représenté dans les ordinateurs comme une succession de bits décomposée en trois zones:

- Bit de signe
- Exposant
- Mantisse



- Par exemple pour le nombre $-(101,011)_2 = (1,01011 \times 2^2)_2$, le signe est - , la mantisse est le nombre situé après la virgule, soit 01011, et l'exposant est 2.
- En informatique, une norme s'est imposée pour la représentation des nombres flottants. C'est la norme IEEE 754.

2. Nombres réels (Suite)

b) Représentation en virgule flottante (Suite)

- **La norme IEEE 754**

- L'IEEE 754 (IEEE signifie : Institute of Electrical and Electronics Engineers) 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. Nombres réels (Suite)

b) Représentation en virgule flottante (Suite)

- **La norme IEEE 754 (Suite)**

2. La deuxième E désigne l'exposant , il est codé sur les bits consécutifs au signe.

3. La troisième M désigne la mantisse (les bits situés après la virgule) sur les bits restants.

- Remarque : Pour éviter d'avoir des exposants négatifs, on utilise un exposant dit décalé (biaisé) au lieu de l'exposant simple. La valeur de cet exposant décalé est égale à la valeur de l'exposant réel, additionnée de la valeur de décalage. La valeur du décalage doit être suffisamment grande pour pouvoir décaler tous les exposants de valeur négative.

2. Nombres réels (Suite)

b) Représentation en virgule flottante (Suite)

- **La norme IEEE 754 (Suite)**

- La norme IEEE 754 définit trois formats de représentations des nombres flottants :

1. Simple précision sur 32 bits

Signe mantisse	Exposant décalé	Mantisse normalisée
1 bit	8 bits	23 bits

2. Double précision sur 64 bits

Signe mantisse	Exposant décalé	Mantisse normalisée
1 bit	11 bits	52 bits

3. Précision étendue sur 80 bits

Signe mantisse	Exposant décalé	Mantisse normalisée
1bit	15 bits	64 bits

2. Nombres réels (Suite)

b) Représentation en virgule flottante (Suite)

- **La norme IEEE 754 (Suite)**

- Pour écrire un nombre sous la forme IEEE 754 il faut :

  1. Convertir le nombre en binaire.
  2. Ecrire le nombre sous la forme normalisée :

$$N = (+/-)\ (1,m)_2 * 2^{ER}$$

(ER : Exposant réel)

  3. Calculer l'exposant décalé : Par exemple, dans la simple précision, l'exposant est décalé de $2^{8-1}-1$:

$$Eb = ER+127,$$

(Eb : Exposant biaisé)

- Remarque : Le 1 précédant la virgule n'est pas codé en machine (appelé bit cache).

2. Nombres réels (Suite)

b) Représentation en virgule flottante (Suite)

- **La norme IEEE 754 (Suite)**

- Exemple 1: Traduisons en binaire, en utilisant la norme IEEE 754 simple précision (sur 32 bits), le nombre : -25,8125.

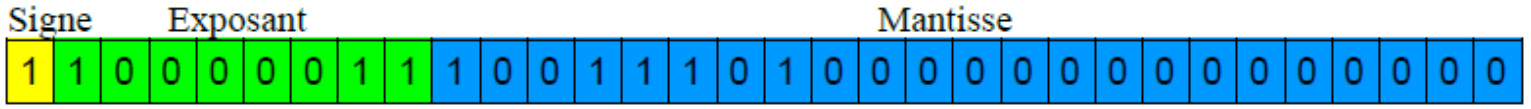
  $(25,8125)_{10} = (11001,1101)_2$

  $11001,1101 = 1,10011101 * 2^4$ (2^4 décale la virgule de 4 chiffres vers la droite).

  La mantisse, étendue sur 23 bits, est : 100 1110 1000 0000 0000 0000

  $Eb = 127 + 4 = (131)_{10} = (10000011)_2$

  Signe = 1, ce qui indique un nombre négatif.



En binaire : 11000001110011101000000000000000

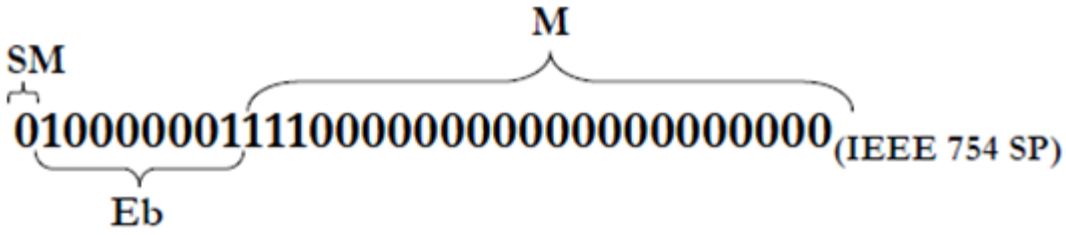
En hexadecimal : C1CE8000.

2. Nombres réels (Suite)

b) Représentation en virgule flottante (Suite)

- **La norme IEEE 754 (Suite)**
  - Exemple 2:

Conversion IEEE754 - Décimale
(Evaluation d'un réel)



S = 0, donc nombre positif

Eb = 129, donc exposant = Eb-127 = 2

1,M = 1,111

$+ 1{,}111_{(2)} \cdot 2^2 = 111{,}1_{(2)} = 7{,}5_{(10)}$

2. Nombres réels (Suite)

b) Représentation en virgule flottante (Suite)

- **La norme IEEE 754 (Suite)**

- Remarque 1 :

  Il est a noter que l'on n'écrit pas le 1 devant la virgule, car il existe forcement, sauf pour le nombre 0, qui est un cas particulier. Il y a d'ailleurs deux 0 : un négatif (1 suivi de 31 zéros) et un positif (32 zéros).

- Remarque 2 :

  L'avantage de la virgule flottante par rapport à la virgule fixe est que l'intervalle des valeurs possibles est plus étendu (elle est la plus utilisée par les ordinateurs).

II. Représentation des caractères

1. Introduction

- Les caractères, appelés **symboles alphanumériques**, incluent les lettres majuscules et minuscules, les symboles de ponctuation (&, . ; # " -etc), et les chiffres.
- Les caractères, comme les nombres, doivent être représentés en vue d'être exploités par les ordinateurs.
- Il faut donc que l'on soit en mesure de les coder sous forme binaire.
- Le codage des caractères est fait par une table de correspondance indiquant la configuration binaire représentant chaque caractère.
- Évidemment, pour pouvoir échanger des informations entre ordinateurs, il faut que cette correspondance soit la même sur toutes les machines.
- Voici les systèmes de codage de caractères les plus utilisés.

2. Le code EBCDIC

- EBCDIC (Extended Binary Coded Decimal Interchange Code).
- C'est un mode de codage des caractères sur 8 bits créé par IBM à l'époque des cartes perforées.
- Il dérive du codage 6 bits décimal codé binaire, utilisé dans cartes perforées et dans la plupart des périphériques IBM de la fin des années 1950 et au début du 1960.
- EBCDIC est encore utilisé dans les systèmes AS/400 d'IBM ainsi que sur les mainframes sous MVS (aujourd'hui z/OS), VM ou DOS/VSE.

3. Le codage ASCII

- ASCII : American Standard Code for Information Interchange.
- La norme ASCII (on prononce généralement «aski») établit une correspondance entre une représentation binaire des caractères de l'alphabet latin et les symboles, les signes, qui constituent cet alphabet.
- La quasi-totalité des ordinateurs personnels et des stations de travail utilisent l'encodage ASCII.
- Le code ASCII de base représentait les caractères sur 7 bits (c'est-à-dire 128 caractères possibles, de 0 a 127).
- Toutefois, les ordinateurs travaillant presque tous sur un multiple de huit bits (multiple d'un octet) depuis les années 1970, chaque caractère d'un texte en ASCII est souvent stocké dans un octet dont le 8e bit est 0.

3. Le codage ASCII (Suite)

- **Table des codes ASCII**

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

- Pour lire le tableau, on associe au caractère le code binaire suivant : n° de ligne_n° de colonne. Par exemple, le caractère a est situé dans la case correspondant à la 7e ligne, 2e colonne. Sa représentation binaire est : 110_0001 soit sous la forme d'un octet : 01100001. Ce qui correspond au caractère 97 en décimal.
- Pour faciliter la lecture des codes ascii par un humain, il est plus commode de coder les numéros de ligne et de colonne en hexadécimal, et la lettre 'a' aura alors pour code 61 en hexadécimal.

3. Le codage ASCII (Suite)

- Les codes 0 à 31 ne sont pas des caractères. On les appelle caractères de contrôle car ils permettent de faire des actions telles que :
- retour a la ligne (Carriage return) ;
- bip sonore (Audible bell).
- Les codes 65 à 90 représentent les majuscules.
- Les codes 97 à 122 représentent les minuscules (il suffit de modifier le 6ème bit pour passer de majuscules a minuscules, c'est-à-dire ajouter 32 au code ASCII en base décimale).
- Le code ASCII a été mis au point pour la langue anglaise, il ne contient donc pas de caractères accentués, ni de caractères spécifiques à une langue. Le code ASCII a donc été étendu à 8 bits pour pouvoir coder plus de caractères (on parle d'ailleurs de code ASCII étendu).

3. Le codage ASCII (Suite)

❖ Le codage ASCII étendu

- La nécessité de représenter des textes comportant des caractères non présents dans la table ASCII tels ceux de l'alphabet latin utilisés en français comme le 'à', le 'é' ou le 'ç' impose l'utilisation d'un autre codage que l'ASCII.
- Afin de représenter plus de langues et notamment des langues occidentales comme la France, on a étendu le code de 7 à 8 bits.
- Mais les 8 bits de l'octet vont être utilisés. Cela permet de coder $2^8 = 256$ caractères, soit **128 caractères supplémentaires.**
- L'ISO, organisation internationale de normalisation, propose de son côté plusieurs variantes de codages adaptées aux différentes langues. La plus utilisée concerne les langues européennes occidentales. Il s'agit de l'ISO-8859-1, aussi nommé ISO-Latin1.

4. Le codage UNICODE

- Actuellement il existe un autre code sur 16 bits , se code s'appel UNICODE.
- UNICODE16 bits -> 65 536 caractères.
- Ce code contient en plus de tous les caractères européens, 42000 caractères asiatiques.
- Le code ASCII est contenu dans les 128 premiers caractères d'UNICODE.

5. Le codage UFT-8

- Unicode, dans la théorie, c'est très bien, mais dans la pratique, pour chaque lettre il occupe 2 octets (16 bits).
- C'est du gaspillage : par exemple plusieurs langues partagent les mêmes lettres (français, anglais,…) notamment les lettres de l'alphabet français non accentués. Pour optimiser cela, on utilise 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".

III. Codage binaire

1. Le codage binaire pur

- Le binaire pur est aussi qualifié de binaire naturel.
- Ce codage a déjà été abordé dans le cadre du chapitre 2 (les systèmes de numération).
- En effet, dans ce codage on associe à chaque entier positif la valeur qui lui correspond selon le système de numération binaire.
- Exemple :

Sur 6 bits : $(35)_{10} = (100011)_2$

2. Le code binaire réfléchi (code DE GRAY)

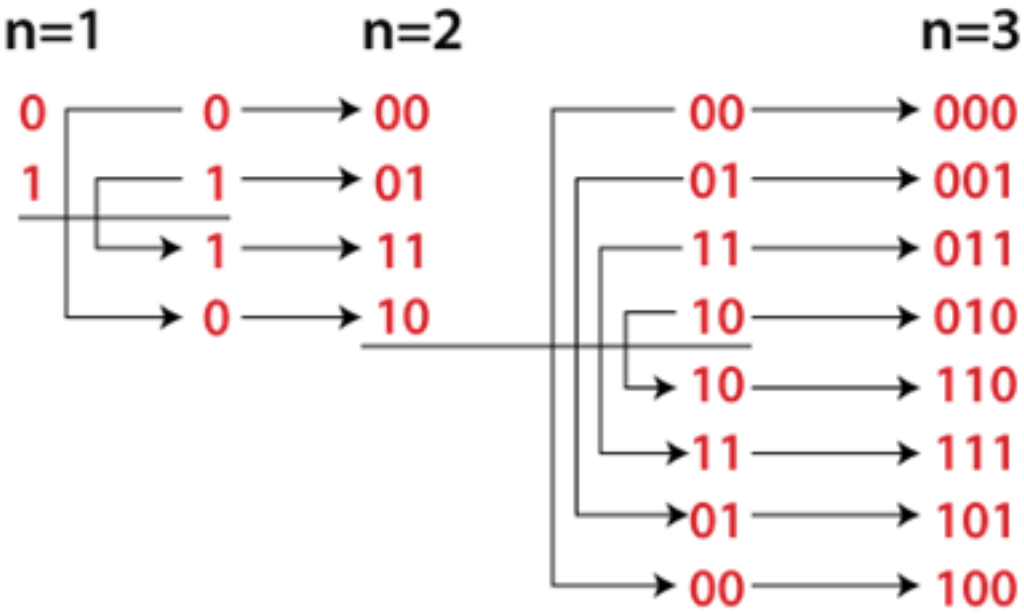
- Dans le code binaire pur le passage d'une combinaison à l'autre entraîne parfois le changement simultané de plusieurs bits. C'est par exemple le cas pour la transition de l'équivalent décimal 3 à l'équivalent décimal 4 pour laquelle les bits de poids 1 et 2 passent de 1 à 0 et le bit de poids 3 passe de 0 à 1. (Le passage de 011 à 100 implique la modification de 3 bits).
- Pour éviter cet inconvénient, il est nécessaire d'imaginer des codes pour lesquels le passage d'une combinaison à la suivante n'implique que la modification d'un bit et d'un seul (les transitions s'effectuent sans ambiguïté, éliminant les risques d'aléas). De tels codes sont appelés "codes réfléchis". Parmi ceux-ci, le code de **Gray** est le plus employé.
- Tout comme le binaire naturel, le binaire réfléchi peut coder n'importe quel nombre entier naturel.
- Remarque : Un code réfléchi est un code non pondéré ne peut être utilisé pour les opérations arithmétiques.

2. Le code binaire réfléchi (code DE GRAY) (Suite)

- Le code Gray est utilisé lorsque l'on désire une progression numérique binaire sans parasite transitoire. Il sert également dans les tableaux de Karnaugh utilisés lors de la conception de circuits logiques.
- Il est construit de telle façon qu'à partir du chiffre 0 chaque nombre consécutif diffère du précédent immédiat d'un seul digit (bit).
- Une des méthodes de construction du code Gray s'appelle la méthode du code binaire réfléchi ou code REFLEX. Voici son principe:
  1. On établi un code de départ : zéro est codé 0 et un est codé 1.
  2. Puis, à chaque fois qu'on a besoin d'un bit supplémentaire, on symétrise les nombres déjà obtenus (comme une réflexion dans un miroir).
  3. et on rajoute un 1 au début des nouveaux nombres et un zéro sur les anciens.

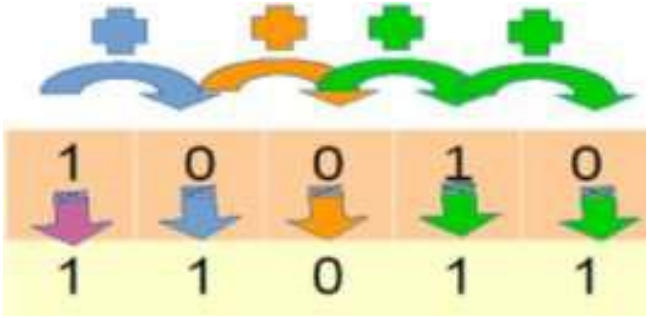
2. Le code binaire réfléchi (code DE GRAY) (Suite)

- Construction d'une table de code de gray par réflexion :



2. Le code binaire réfléchi (code DE GRAY) (Suite)

- Il existe d'autres méthodes pour calculer code de gray :



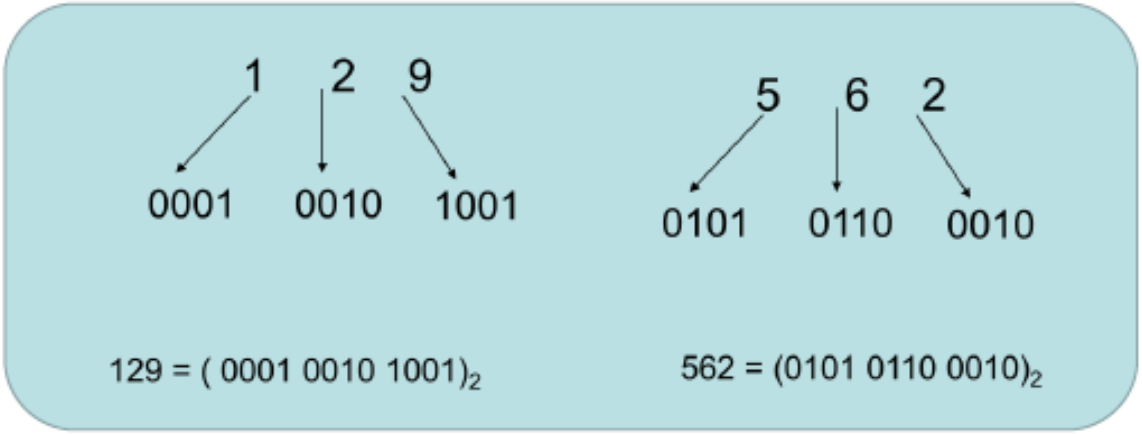
- Le premier Bit à gauche reste le même,
- Puis, de gauche à droite faire la somme des bits adjacents sans retenue.
- Exemple : $(10010)_2=(11011)_{Gray}$

3. Le code BCD (Binary Coded Decimal)

- Les informations traitées par l'ordinateur ne sont pas toujours converties selon le système de numération binaire.
- Ces informations peuvent être manipulées sous forme décimale codée binaire, c'est-à-dire, que des codes (en binaires) sont associés à des nombres décimaux sans pour autant faire la conversion traditionnelle décimal → binaire.
- Parmi les codes définis à cet effet : Le code BCD (binary coded decimal) ou DCB (décimal codé binaire) ou encore code 8421

3. Le code BCD (Suite)

- Le BCD (Binary Coded Decimal), ou Décimal Codé en Binaire en français est le code décimal le plus utilisé en électronique.
- Il contient des mots-code qui sont la traduction en binaire naturel (sur 4 bits) de chacun des dix chiffres du système décimal.
- Son principe : Associer un code binaire à chaque chiffre décimal.
- Exemple 1 : Le nombre 512 en décimal équivaut à $(1000000000)_2$. En code BCD ce nombre sera codé comme suit $(0101\ 0001\ 0010)_{BCD}$
- Autres exemples :



3. Le code BCD (Suite)

- Dans le code BCD, il y a des configurations binaires non exploitées.
- En effet, sachant que le nombre de chiffres dans la base 10 est justement 10 et sachant par ailleurs que le nombre de configurations que l'on peut représenter avec 4 chiffres est 16, on déduit qu'il y a 6 configurations non utilisées.

Chiffre Décimal	Code BCD	
0	0000	
1	0001	
2	0010	
3	0011	
4	0100	
5	0101	
6	0110	
7	0111	
8	1000	
9	1001	
/	1010	Non utilisés
/	1011	
/	1100	
/	1101	
/	1110	
/	1111	

3. Le code BCD (Suite)

❖ Addition BCD

- Pour additionner correctement deux chiffres BCD, on utilise un additionneur binaire à 4 bits et une retenue.
- Il y a deux cas possibles :
  - La somme est un chiffre BCD valide (0 à 9); ou,
  - La somme est égale ou supérieure à dix.
- Dans le premier cas, aucune action n'est requise.
- Dans le deuxième cas, il faut ajouter $+6= +(0110)_2$ à la somme obtenue. La combinaison de la retenue et de la somme finale produit alors un résultat correct.

3. Le code BCD (Suite)

- Addition BCD (Suite)
- Exemples :

Exemple : addition 3 + 4 en BCD :

```
  3      0 0 1 1
 +4     +0 1 0 0
 07    0 | 0 1 1 1   ← somme inférieure à dix, résultat correct
                      (retenue 0, somme 7)
```

Exemple : addition 7 + 8 en BCD :

```
  7      0 1 1 1
 +8     +1 0 0 0
 15    0 | 1 1 1 1   ← somme supérieure à 9, il faut ajouter +6
      +0 1 1 0      + 6
      1 | 0 1 0 1   ← résultat correct, retenue de 1, somme de 5
```

Exemple : addition 9 + 8 en BCD :

```
  9      1 0 0 1
 +8     +1 0 0 0
 17    1 | 0 0 0 1   ← somme supérieure à 9, il faut ajouter +6
      +0 1 1 0      + 6
      1 | 0 1 1 1   ← résultat correct, retenue de 1, somme de 7
```

3. Le code BCD (Suite)

- Addition BCD (Suite)

- Un autre exemple :

$18_{10} + 8_{10} = 0001\ 1000 + 0000\ 1000$

```
   0001  1000
+  0000  1000
   0001 10000     On ajoute 6 car c'est > 9
+         0110
   0001 10110     Le cinquième bit est passé comme retenue au digit de gauche
   0010  0110   = 26(10)
```

4. Le code excédent 3 (BCD+3)

- Ce code ressemble beaucoup au code BCD.
- Son principe est basé sur le fait d'associer à chaque chiffre décimal son équivalent binaire additionné de 3.
- Exemple :

  512 en décimal vaut (0101 + 0011) (0001 + 0011) (0010 + 0011) ce qui donne : (1000 0100 0101) Excédent 3
- Le tableau ci-dessus donne l'équivalent Excédent 3 des chiffres décimaux.

Chiffre Décimal	Code BCD	Code excédent 3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

- Remarque : Dans le code Excess 3 : Pour l'addition, s'il y a une retenue on doit y ajouter 3 $(0011)_2$ sinon on soustrait 3 $(0011)_2$.