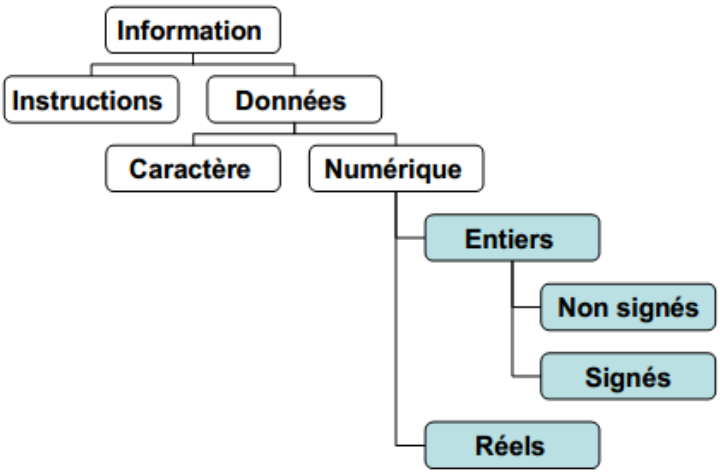
Chapitre 2 : Représentation de l'information dans la machine

Introduction



I. Représentation des nombres entiers

Il existe deux types d'entiers :

– les entiers non signés (positifs) ; et

– les entiers signés (positifs ou négatifs)

• **Problème** : Comment indiquer à la machine qu'un nombre est négatif ou positif ?

1. Codage des entiers positifs

1.1. Utilisation du code binaire pur :

- L'entier naturel (positif ou nul) est représenté en base 2,

- Les bits sont rangés selon leur poids, on complète à gauche par des 0

Exemple : sur un octet, $(10)_{10}$ se code en binaire pur? $(0\ 0\ 0\ 0\ 1\ 0\ 1\ 0)_2$

1.2. Etendu du codage binaire pur

- Codage sur n bits : représentation des nombres de 0 à $2^n - 1$

- sur 1 octet (8 bits): codage des nombres de 0 à $2^8 - 1 = 255$

- sur 2 octets (16 bits): codage des nombres de 0 à $2^{16} - 1 = 65535$

- sur 4 octets (32 bits) : codage des nombres de 0 à 2^{32}- 1 = 4 294 967 295

2. Codage des entiers relatifs (signés)

• Il existe 3 méthodes pour représenter les nombres négatifs :

– Code binaire signé (par signe et valeur absolue)

– Complément à 1(complément restreint)

– Complément à 2 (complément à vrai)

2.1. Représentation Binaire signé

- Si on travail 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 : Sur 8 bits, codage des nombres -24 et -128 en (bs)

-24 est codé en binaire signé par : $1\ 0\ 0\ 1\ 1\ 0\ 0\ 0_{(bs)}$

-128 hors limite nécessite ⟶ 9 bits au minimum car $(128)_{10}= (10000000)_2$

Etendu de codage : Avec n bits, on code tous les nombres entre $-(2^{n-1}-1)$ et $(2^{n-1}-1)$

Exemple : Avec 4 bits : -7 et +7

Limitation :

- Deux représentations du zéro : + 0 et - 0 Sur 4 bits : +0 = 0000(bs), -0 = 1000(bs) qui conduit à des difficultés au niveau des opérations arithmétiques

- Multiplication et l'addition sont moins évidentes

2.2. Représentation en complément à un (Complément restreint)

- les nombres positifs sont codés de la même façon qu'en binaire pure.

- un nombre négatif est codé en inversant chaque bit de la représentation de sa valeur absolue

- Le bit le plus significatif est utilisé pour représenter le signe du nombre :

- si le bit le plus fort = 1 alors nombre négatif
- si le bit le plus fort = 0 alors nombre positif

Exemple : -24 en complément à 1 sur 8 bits

- |-24|en binaire pur ⟶ $0\,0\,0\,1\,1\,0\,0\,0_{(2)}$ puis
- on inverse les bits ⟶ $1\,1\,1\,0\,0\,1\,1\,1_{(cà1)}$

Remarque

• 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.

$$CA1(CA1(N))= N$$

Limitation :

- deux codages différents pour 0 (+0 et -0)
- Sur 8 bits : +0=$0\,0\,0\,0\,0\,0\,0\,0_{(cà1)}$et -0=$1\,1\,1\,1\,1\,1\,1\,1_{(cà1)}$
- Multiplication et l'addition sont moins évidentes.

2.3. Représentation en complément à 2(Complément à vrai)

- les nombres positifs sont codés de la même manière qu'en binaire pure.

- un nombre négatif est codé en ajoutant la valeur 1 à son complément à 1

- Le bit le plus significatif est utilisé pour représenter le signe du nombre

Exemple : -24 en complément à 2 sur 8 bits

24 est codé par $0\,0\,0\,1\,1\,0\,0\,0_{(2)}$

-24⟶ $1\,1\,1\,0\,0\,1\,1\,1_{(cà1)}$

donc -24 est codé par $1\,1\,1\,0\,1\,0\,0\,0_{(cà2)}$

- Un seul codage pour 0. Par exemple sur 8 bits :

- +0 est codé par $00000000_{(cà2)}$
- -0 est codé par $11111111_{(cà1)}$
- Donc -0 sera représenté par $00000000_{(cà2)}$

Etendu de codage :

- Avec n bits, on peut coder de $-(2^{n-1})$ à $(2^{n-1}-1)$
- Sur 1 octet (8 bits), codage des nombres de -128 à 127

  +0 = 00000000 -0=00000000

  +1 = 00000001 -1=111111111

  … …

  +127= 01111111 -128=10000000

Remarque 2

• Dans cette représentation, 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

-**Exercices**-

- Coder $100_{(10)}$ et $-100_{(10)}$par complément à 2 sur 8 bits

$100_{(10)}= 01100100_{(Cà2)}$

$-100_{(10)}= 10011010_{(Cà2)}$

- Décoder en décimal 11001001(Cà2) et 01101101(Cà2)

$11001001_{(Cà2)}= -55_{(10)}$

$01101101_{(Cà2)}= 109_{(10)}$

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

2.4. Opérations arithmétiques en CA2

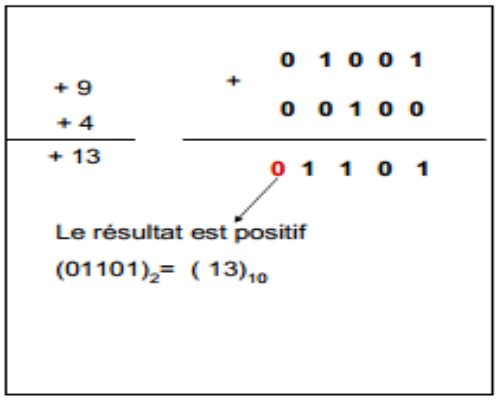
Principe :

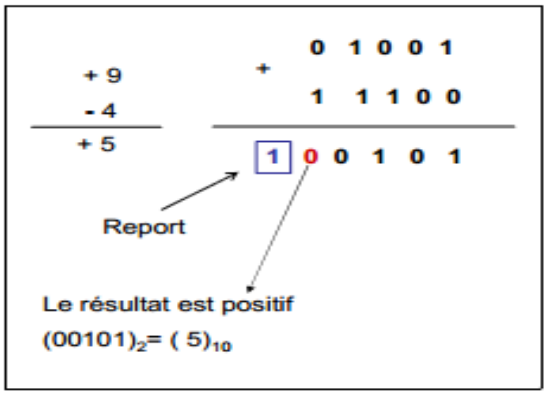
- dans l'addition une retenue générée par le bit de signe sera ignorée.

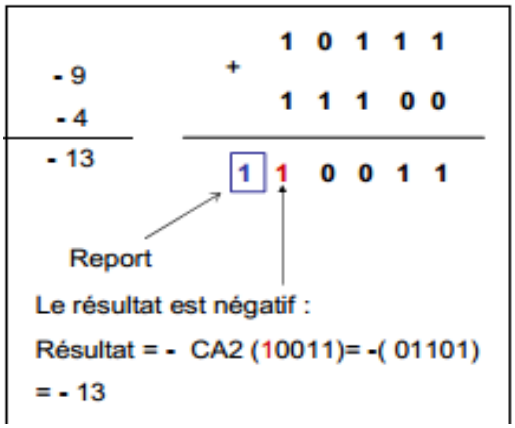
Une condition nécessaire pour accepter le résultat ainsi obtenu, est qu'une retenue doit être aussi générer par le bit juste avant signe.

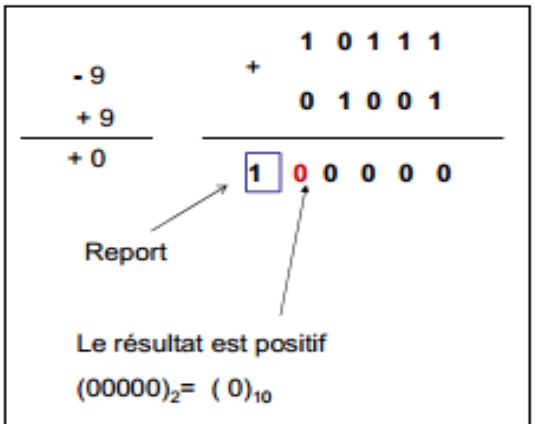
- Si une retenue est générée exclusivement par les bits de signe ou par le bit juste avant signe, l'addition est impossible a réalisée. Une erreur de dépassement de capacités est déclarée.

Effectuer les opérations suivantes sur 5 Bits, en utilisant la représentation en CA2









2.5. La retenue et le débordement

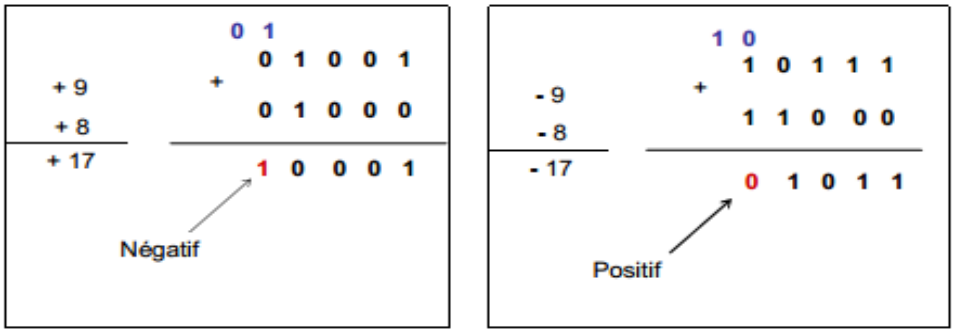
• On dit qu'il y a une retenue si une opération arithmétique génère un report.

• 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.

Exemple



•Nous avons un débordement si la somme de deux nombres positifs donne un nombre négatif.

•Ou la somme de deux nombres négatifs donne un Nombre positif

• Il n'y a jamais un débordement si les deux nombres sont de signes différents.

II. Représentation des nombres réels

• Les formats de représentations des nombres réels sont :

Format virgule fixe

- utilisé par les premières machines
- possède une partie 'entière' et une partie 'décimale' séparés par une virgule. La position de la virgule est fixe d'où le nom.
- Exemple : $54{,}25_{(10)}$; $10{,}001_{(2)}$; A1, $F0B_{(16)}$

Format virgule flottante (utilisé actuellement sur machine)

- défini par : ± m .b^e
  - ✓ un signe + ou –
  - ✓ une mantisse m (en virgule fixe)

- ✓ un exposant e (un entier relative)
- ✓ une base b (2,8,10,16,…)
- ✓ Exemple : $0,5425 . 10^{2}_{(10)}$; $10,1 . 2^{-1}_{(2)}$; A0, $B4.16^{-2}_{(16)}$

1. Codage en Virgule Fixe

Etant donné une base b; un nombre x est représenté par :

- $x = a_{n-1}a_{n-2}\ldots a_1a_0,a_{-1}a_{-2}\ldots a_{-p}$ (b)
- a_{n-1} est le chiffre de **poids fort**
- a_{-p} est le chiffre de **poids faible**
- n est le nombre de chiffre avant la virgule
- p est le nombre de chiffre après la virgule
- La valeur de x en base 10 est : $x = \sum_{-p}^{n-1} a_i b^i$

- Exemple :

$101,01_{(2)}=1.2^2+0.2^1+1.2^0+0.2^{-1}+1.2^{-2} = 5,25_{(10)}$

Conversion de la base 10 à la base b :

- Partie entière est codée sur p bits (est transformée en effectuant des divisions successives par b)
- Partie décimale est codée sur q bits en multipliant par b successivement jusqu'à ce que la partie décimale soit nulle ou le nombre de bits q est atteint. Exemple : $35,625=(?)_2$

Exemple : $35,625=(?)_2$

P.E= 35 = $(100011)_2$

PF= 0,625 = $(?)_2$

0,625 * 2 = 1 ,25
0,25 * 2 = 0 ,5
0,5 * 2 = 1 ,0

$(0,625)=(0,101)_2$

Donc $35,625=(100011,101)_2$

- **Exemple 2:** $(0,6)_{10}=(?)_2$

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

  ⟶ $(0,6)= (0,1001)_2$

Remarque :

Le nombre de bits après la virgule va déterminer la précision .

Exemple 3 : $4,25_{(10)}= ?_{(2)}$ format virgule fixe

- $4_{(10)}= 100_{(2)}$
- 0,25 x 2= 0,5 ⟶ 0
- 0,5 x 2 = 1,0 ⟶ 1
- donc $4,25_{(10)}= 100,01_{(2)}$

2. Représentation en virgule flottante

• Chaque nombre réel peut s'écrire de la façon suivante :

$N= \pm M * b^e$

– M : mantisse,

– b : la base ,

– e : l'exposant

• **Exemple :**

$15,6 = 0,156 * 10^{+2}$

$- (110,101)_2= - (0,110101)_2 * 2^{+3}$

$(0,00101)_2= (0,101)_2 * 2^{-2}$

Le codage en base 2, format virgule flottante, revient à coder le signe, la mantisse et l'exposant.

Exemple : Codage en base 2, format virgule flottante, de (3,25)

$3,25_{(10)}= 11,01_{(2)}$(en virgule fixe)

$= 1,101 . 2^{1}_{(2)}$

$= 110,1 . 2^{-1}_{(2)}$

Pb : différentes manières de représenter E et M ⟶ **Normalisation**

Remarque :

On dit que la mantisse est normalisée si le premier chiffre après la virgule est différent de 0 et le premier chiffre avant la virgule est égale à 0.

2.1. Normalisation

Dans cette représentation sur **n** bits :

– La mantisse est sous la forme signe/valeur absolue

- 1 bit pour le signe
- et k bits pour la valeur.

– L'exposant (positif ou négatif) est représenté sur p bits

Signe mantisse	Exposant	Mantisse normalisée
1 bit	p bits	k bits

- Normalisé : virgule est placé après le bit à 1 ayant le poids fort

2.2. Représentation de l'exposant en complément à deux

• On veut représenter les nombres $(0,015)_8$ et $-(15, 01)_8$ en virgule flottante sur une machine ayant le format suivant :

Signe mantisse	Exposant en CA2	Mantisse normalisée
1 bit	4 bits	8 bits

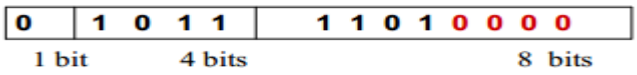
Donc :

$(0,015)_8=(0,000001101)_2= 0,1101 * 2^{-5}$

Signe mantisse : positif (0)

Mantisse normalisé : 0,1101

Exposant = -5 → utiliser le complément à deux pour représenter le -5 Sur 4 bits
CA2(0101)=1011



$-(15,01)_8 = -(001101,000001)_2 = -0,1101000001 * 2^4$

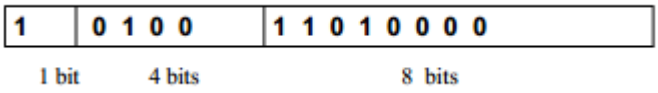
Signe mantisse : négatif (1)

Mantisse normalisée : 0,1101000001

Exposant = 4 , en complément à deux il garde la même valeur (0100)

On remarque que la mantisse est sur 10 bits (1101 0000 01), et sur la machine seulement 8 bits sont utilisés pour la mantisse.

Dans ce cas on va prendre les 8 premiers bits de la mantisse



Remarque :

si la mantisse est sur k bits et si elle est représenté sur la machine sur k' bits tel que k>k' , alors la mantisse sera tronquée : on va prendre uniquement k' bits → perdre dans la précision .

2.3. Représentation de L'Exposant décalé (biaisé)

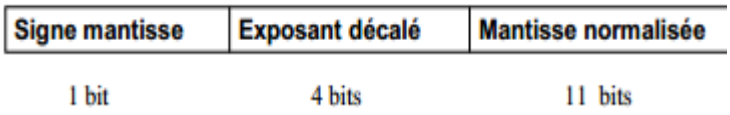
Avec l'exposant biaisé on a transformé les exposants négatifs à des exposants positifs en rajoutons à l'exposant la valeur 2^{p-1}.

La valeur 2^{p-1} s'appelle le biais ou le décalage

Exposant Biaisé = Exposant réel + Biais

Exemple

• On veut représenter les nombres $(0,015)_8$ et $-(15, 01)_8$ en virgule flottante sur une machine ayant le format suivant :



$(0,015)_8=(0,000001101)_2= 0,1101 * 2^{-5}$

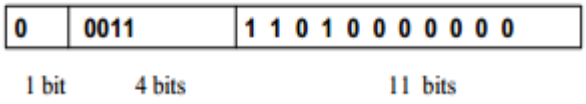
Signe mantisse : positif (0)

Mantisse normalisé : 0,1101

Exposant réel = -5

Calculer le biais : $b= 2^{4-1} = 8$

Exposant Biaisé = -5 + 8 = +3 = $(0011)_2$



- $(15,01)_8=(001101,000001)_2= 0,1101000001 * 2^4$

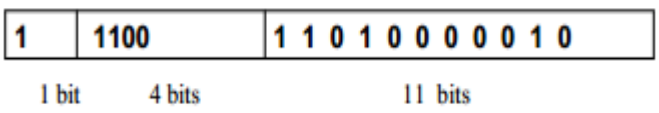
Signe mantisse : négatif (1)

Mantisse normalisée : 0,1101000001

Exposant réel = + 4

Calculer le biais : $b= 2^{4-1}= 8$

Exposant Biaisé = 4 + 8 = +12 = $(1100)_2$



2.4. Opérations arithmétiques en virgule flottante

• Soit deux nombres réels N1 et N2 tel que

$N1=M1*b^{e1}$

et $N2=M2*b^{e2}$

• On veut calculer N1+N2 ?

• Deux cas se présentent :

– Si e1 = e2 alors $N3= (M1+M2)\ b^{e1}$

– Si e1 <> e2 alors élevé au plus grand exposant et faire l'addition des mantisses et par la suite normalisée la mantisse du résultat.

Exemple

• Effectuer l'opération suivante : $(0,15)_8+ (1,5)_8=(?)$:

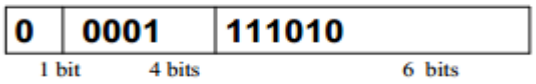
$(0,15)_8= (0,001101) = 0,1101 *2^{-2}$

$(1,5)_8= (001, 1\ 01) = 0,1101 *2^1$

$(0,15)_8+ (1,5)_8= 0,1101 *2^{-2}+ 0,1101 *2^1$

$= 0,0001101 *2^1+ 0,1101 *2^1$

$= 0, 1110101 * 2^1$



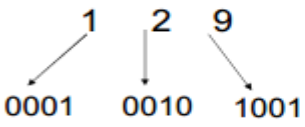
III. Le codage BCD (Binary Coded Decimal)

•Pour passer du décimal au binaire, il faut effectuer des divisions successives. Il existe une autre méthode simplifiée pour le passage du décimal au binaire.

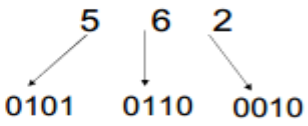
• Le principe consiste à faire des éclatements sur 4 bits et de remplacer chaque chiffre décimal par sa valeur binaire correspondante.

• Les combinaisons supérieures à 9 sont interdites

Exemple



$129 = (0001\ 0010\ 1001)_2$

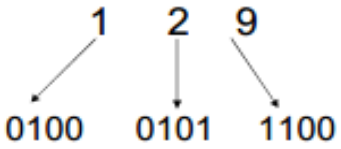


$562 = (0101\ 0110\ 0010)_2$

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

Le codage EXCESS3 (BCD+3)

Décimal	BCD+3	Binaire
0	3	0011
1	4	0100
2	5	0101
3	6	0110
4	7	0111
5	8	1000
6	9	1001
7	10	1010
8	11	1011
9	12	1100



IV. Représentation des Caractères

Les caractères, appelés symboles alphanumériques, incluent les lettres majuscules et minuscules, les symboles de ponctuation (&~ , . ; # " - etc...), et les chiffres.

Les caractères sont des données non numériques (addition n'a pas de sens) ; Par contre, il est souvent utile de comparer deux caractères, par exemple pour les trier dans l'ordre alphabétique.

Le codage des caractères est fait par une table de correspondance indiquant la configuration binaire représentant chaque caractère.

Les deux codes les plus connus sont l'EBCDIC de IBM (en voie de disparition) et le code ASCII (American Standard Code for Information Interchange).

1. Code (ou Table) ASCII (American Standard Code for Information Interchange)

- 7 bits pour représenter 128 caractères (0 à 127)
- 48 à 57 : chiffres dans l'ordre (0,1,…,9)
- 65 à 90 : les alphabets majuscules (A,…,Z)
- 97 à 122 : les alphabets minuscule (a,…z)

2. Table ASCII Etendu

- 8 bits pour représenter 256 caractères (0 à 255)
- Code les caractères accentués : à, è,…etc.
- Compatible avec ASCII

3. Code Unicode (mis au point en 1991)

- 16 bits pour représenter 65 536 caractères (0 à 65 535)
- Compatible avec ASCII
- Code la plupart des alphabets : Arabe, Chinois, ….
- On en a défini environ 50 000 caractères pour l'instant