ENSEIRB

DEPARTEMENT ELECTRONIQUE
E2
- STAGE 68000 -
Cours 2 : Instructions Arithmétiques

Patrice KADIONIK

email : kadionik@enseirb.fr
http : www.enseirb.fr/~kadionik

SOMMAIRE

1. OBJECTIFS :

Etude des instructions ASM 68000 pour les opérations arithmétiques

Le jeu d 'instructions 68000 comporte 8 groupes :

1. Instructions de transfert de données (MOVE, MOVEM…)
2. Instructions d 'arithmétique entière
3. Instructions logiques
4. Instructions de décalage et de rotation
5. Instructions de manipulations de bits
6. Instructions BCD (*Binary Coded Decimal*)
7. Instructions de contrôle d 'exécution (CMP, TST, TAS…)
8. Instructions de contrôle système : instructions privilégiées

2. REPRESENTATION DES NOMBRES ENTIERS :

Le 68000 ne traite que des nombres entiers. Pour les nombres réels, on a recours à :
- un coprocesseur mathématique (FPU), par exemple le 68881
- une bibliothèque d 'émulation logicielle par Développement Limité

Le format d 'un nombre réel est normalisé : IEEE 754 (simple ou double précision)

Nombres entiers Non Signés (NS) :

- 8 bits	: $00 à $FF (0 à 255)	.B Byte
- 16 bits	: $0000 à $FFFF (0 à 65535)	.W Word
- 32 bits	: $00000000 à $FFFFFFFF (0 à 4294967295)	.L Long

Nombres entiers Signés (S) :
On utilise la complémentation à 2.
Si MSB=1, nombre <0
Si MSB=0, nombre >=0

- 8 bits	: MSB=0 $00 à $7F (0 à 127)	
	MSB=1 $80 à $FF (-128 à -1)	.B Byte
- 16 bits	: MSB=0 $0000 à $7FFF (0 à 32767)	
	MSB=1 $8000 à $FFFF (-32768 à -1)	.W Word
- 32 bits	: MSB=0 $00000000 à $7FFFFFFF	
	MSB=1 $80000000 à $FFFFFFFF	.L Long

Représentation binaire :

nombre entier de 8 bits :

$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$
1 0 0 0 0 0 0 1 soit $2^7 + 2^0 = 128 + 1 = 129$

Représentation hexadécimale :

On considère la représentation binaire et l 'on regroupe 4 bits consécutifs que l 'on code en base 16. On utilise le symbole $xx (Motorola) ou xxH (HP) pour indiquer la représentation hexadécimale d 'un nombre entier.

0=$0 … 9=$9 10=$A 11=$B 12=$C 13=$D 14=$E 15=$F

$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$
1 0 0 0 0 0 0 1
$8 $1 soit $81 = 129 en hexadécimal (81H en notation HP)

C 'est au programmeur de savoir à priori le type des nombres entiers qu 'il manipule et non le processeur !!!

Le processeur ne sait pas reconnaître qu 'il manipule un nombre NS ou S. On aura ainsi des indicateurs (*flags*) positionnés dans les 2 cas à chaque instruction exécutée (ou presque…). C 'est à la charge ensuite du programmeur de tester les bons flags.

Le jeu d 'instructions du 68000 possède des instructions pour les nombres NS et S (instructions de test, comparaison)

REGISTRE D 'ETAT SR (*Status Register*)

Le 68000 possède :
- des registres de données : D0-D7
- des registres d 'adresses : A0-A7, A7 est le pointeur de pile
- le registre compteur programme : PC
- le registre d 'état : SR, 16 bits

SR :
- octet fort : octet superviseur
- octet faible : octet utilisateur ou CCR (*Code Condition Register*)

T : mode trace : T=1 pour debug en pas à pas
S : état processeur : S=1 mode superviseur, S=0 mode utilisateur
I2 à I0 : masque d 'interruption

X : eXtension
N : Negativ
Z : Zero
V : oVerflow
C : Carry

-> 5 bits d 'état pour arithmétique NS et S qui fournissent des infos sur le résultat de la dernière instruction exécutée

BIT 0 : C
Dépassement de capacité de représentation du résultat pour arithmétique NS. Mis à 1 si une opération d 'addition, de soustraction génère une retenue, 0 sinon

BIT 1 : V
Dépassement de capacité de représentation du résultat pour arithmétique S. Mis à 1 si une opération d 'addition de 2 nombres de même signe ou de soustraction de 2 nombre de signe opposé génère un résultat dont la représentation dépasse la capacité de représentation (en complément à 2), 0 sinon

BIT 2 : Z
Mis à 1 si le résultat de l 'opération est nul, 0 sinon

BIT 3 : N
Mis à 1 si le résultat est négatif (MSB=1), 0 sinon

BIT 4 : X
Bit d 'extension fonctionnant comme C permettant des opérations en précision multiple (64 bits). Cf après...

Le 68000 possède des instructions de branchement conditionnel qui testent le ou les flags X,C,V,N,Z. Si la condition testée est fausse, on passe à l ' instruction suivante sinon déroutement du programme à une autre adresse.

Les flags sont affectés par chaque instruction exécutée (sauf opération sur Ax comme CMPA par exemple)

<u>Bilan :</u>
Arithmétique NS : Z, C
Arithmétique S : N,Z,V

4. INSTRUCTIONS POUR ARITHMETIQUE ENTIERE :

Le 68000 possède 56 instructions (mnémoniques) de base.

Instructions pour arithmétique entière :
- addition : ADD, ADDA, ADDQ, ADDX, ADDI
- soustraction : SUB, SUBA, SUBQ, SUBX, SUBI
- multiplication : MULU, MULS
- division : DIVU, DIVS
- mise à 0 : CLR
- comparaison : CMP, CMPA, CMPI, CMPM
- extension du bit de signe : EXT
- test : TST
- cycle RMW (*Read Modify Write*) : TAS

INSTRUCTIONS D 'ADDITION :

On a 5 instructions pour additionner 2 nombres entiers.

Addition :

ADD <EA>, Dn

Dn <- Dn + (<EA>)

ADD Dn,<EA>

<EA> <- Dn + (<EA>)

EA : adresse effective (*Effectiv Address*) calculée à partir du mode d 'adressage utilisé par l 'instruction

Taille des opérandes : B,W,L
Modes d 'adressage : tous
Flags du CCR : affectés comme précédemment

RAPPELS SUR LES MODES D 'ADRESSAGE DU 68000 :

- direct registre : Dn, An	MOVE.B D0,D1
- indirect : (An)	MOVE.B (A0),D1
- indirect avec post/prédécrémentation : (An)+, -(An)	MOVE.B D0,-(A7)
- indirect avec déplacement : d(An)	MOVE.B 8(A0),D0
- indirect avec déplacement et index : d(An,X)	MOVE.B (A0,D0),D1
- absolu : Abs.W, Abs.L	MOVE.B $2000,D0
- immédiat : #	MOVE.B #1,D0
- indirect avec déplacement/index par rapport à la valeur courante du PC : d(PC), d(PC,X)	MOVE.B 8(PC),D0

RAPPELS SUR LA COMPLEMENTATION :

Soit N, nombre entier sur n bits

1. Complément vrai :

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

2. Complément à 1 :

On inverse les bits 1 à 1. C 'est le nombre $\tilde{N}$ tel que :
$N + \tilde{N} = 2^n - 1$
$\tilde{N} = (2^n - 1) - N$
ex : 11 $0B 00001011 soit 11110100 soit $F4 (nombre négatif)

3. Complément à 2 :

Complément à 1 et ajout de 1. **C 'est l 'opposé de N !!!**

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

ex : 11 $0B soit $F4+1=$F5=-$0B soit -11 CFQD

Cela suppose que l 'on représente les nombres entiers en complément à 2, soit le MSB comme bit de signe et pour 0 tous les bits à 0 (représentation unique)

-N = compl2 (N)

La représentation en complément à 2 permet de manipuler à la fois des nombres S et NS soit une seule ALU pour le processeur au lieu de 2 !

Principe d 'affectation des flags du CCR sur ADD

Cela reste valable pour les autres instructions si l 'on se reporte aux équations logiques d 'affectations des bits du CCR.

Ici :

1. On se reporte aux équations logiques pour ADD

ex : V = Sn.Dn.Rn/ + S n/ + Dn/.Rn

Sn : registre source

Dn : registre destination

Rn : résultat

ou

2. $V = C_n$ xor C_{n-1}

$C = C_n$

Le calcul à la main de la valeur courante de C et V posent problème. Le calcul des autres flags du registre CCR est immédiat.

Exemple 1 : $36 (>0)+ $47 (>0)

```
MOVE.B #$36,D0
MOVE.B #$47,D1
ADD.B D0,D1
```

C_n C_{n-1} (n=8)

```
      1 1
      0 0 1 1 0 1 1 0
+     0 1 0 0 0 1 1 1
      0 1 1 1 1 1 0 1   =$7D (>0)
```

C_n=0 C_{n-1}=0

d 'où C=C_n=0, V=C_n xor C_{n-1}=0 xor 0 = 0, N=0,Z=0,X=C=0

soit CCR=$00

Exemple 2 : $81 (<0) + $7F (>0)

```
        MOVE.B #$81,D0
        MOVE.B #$7F,D1
        ADD.B D0,D1
```

C_n C_{n-1} (n=8)

```
  1 1 1 1 1 1 1 1
    1 0 0 0 0 0 0 1
+   0 1 1 1 1 1 1 1
  ---------------
    0 0 0 0 0 0 0 0   =$00
```

C_n=1 C_{n-1}=1

d'où C=C_n=1, V=C_n xor C_{n-1}=1 xor 1 = 0, N=0,Z=1,X=C=1
soit CCR=$15

Exemple 3 : -$01 (<0) + -$02 (<0)

```
        MOVE.B #$FF,D0
        MOVE.B #$FE,D1
        ADD.B D0,D1
```

C_n C_{n-1} (n=8)

```
  1 1 1 1 1 1 1
    1 1 1 1 1 1 1 1
+   1 1 1 1 1 1 1 0
  ---------------
    1 1 1 1 1 1 0 1   =-$03 (<0)
```

C_n=1 C_{n-1}=1

d'où C=C_n=1, V=C_n xor C_{n-1}=1 xor 1 = 0, N=1,Z=0,X=C=1
soit CCR=$19

Addition d'adresses :

ADDA <EA>,An
An <- An + (<EA>)

Taille des opérandes : W,L
Modes d'adressage : tous
Flags du CCR : non affectés

ex :

```
        A0=$2000
        ADDA.L      #$100,A0
        A0=$2100
```

Addition avec prise en compte du bit d 'extension :

ADDX Dx,Dy
Dy <- Dx + Dy + X
ADDX -(Ax),-(Ay)
Ax <- Ax - N
Ay <- Ay - N
(Ay) <- (Ax) + (Ay) + X

Taille des opérandes : B,W,L
Modes d 'adressage : Dn, -(An)
Flags du CCR : affectés
ex : addition 64 bits (D1,D0) + (D3,D2)
ADD.L D0,D2
ADDX.L D1,D3

Addition avec adressage immédiat :

ADDI #<data>,<EA>
<EA> <- <data> + (<EA>)

Taille des opérandes : B,W,L
Modes d 'adressage : Dn,An,(An),(An)+,-(An),D(An),d(An,X),absolu
Flags du CCR : affectés

ex :
D0.L=$2000
ADDI.L #$100,D0
D0.L=$2100

Addition avec adressage immédiat rapide :

ADDQ #<data>,<EA>
<EA> <- <data> + (<EA>)
<data>=0, ajout de 1…<data>=7, ajout de 8

Taille des opérandes : B,W,L
Modes d 'adressage : Dn,An,(An),(An)+,-(An),D(An),d(An,X),absolu
Flags du CCR : affectés

Temps d 'exécution optimisé (ex : faire i++ en C)
ex :
D0.L=$2000
ADDQ.L #0,D0
D0.L=$2001

INSTRUCTIONS DE SOUSTRACTION :

On a 5 instructions pour soustraire 2 nombres entiers.
SUB, SUBA, SUBX, SUBI, SUBQ

Ces 5 instructions fonctionnent comme leur équivalent pour l 'addition : (ADD devient SUB…). On change le signe + en - dans les équations de l 'opération réalisée.

exemple : Soustraction :

SUB <EA>, Dn
Dn <- Dn - (<EA>)
SUB Dn,<EA>
<EA> <- (<EA>) - Dn

Taille des opérandes : B,W,L
Modes d 'adressage : tous
Flags du CCR : affectés

Une soustraction revient à l 'addition du complément à 2 du nombre.

On reviendra aux équations logiques générales pour calculer à la main la valeur des bits du registre CCR.

INSTRUCTIONS DE NEGATION :

On a 2 instructions pour calculer l 'opposé d ' un nombre entier.
NEG, NEGX

Cela revient à calculer 0 - N. On se ramène donc à une soustraction.

NEG <EA>
<EA> <- 0 - (<EA>)
NEGX <EA>
<EA> <- 0 - (<EA>) - X

Taille des opérandes : B,W,L
Modes d 'adressage : tous
Flags du CCR : affectés

INSTRUCTIONS DE MULTIPLICATION/ DIVISION :

On a 2 instructions de multiplication :
- nombres S : MULS
- nombre NS : MULU

On réalise la multiplication de 2 mots de 16 bits, le résultat est sur 32 bits dans un registre de donnée.

On a 2 instructions de division :
- nombres S : DIVS
- nombre NS : DIVU

On divise un nombre de 32 bits stocké (dividende) dans un registre de donnée par un diviseur de 16 bits stocké en mémoire ou dans un registre. Le résultat de la division remplace le dividende si l 'opération est licite avec dans le mot de poids fort le reste et dans le mot de poids faible le quotient.
C 'est bien sûr une division entière…

Si le diviseur est nul, le dividende est inchangé et le processeur exécute l 'exception No 5 de la table des vecteur dont le point d 'entrée de la routine est précisée à l 'adresse $000014.
Dans le cas de la division, il y a dépassement si le quotient de tient pas sur 16 bits (pas de modifications du dividende) :
- division NS : quotient > 65536
division S : quotient > 32767 ou quotient <-32768

MULS <EA>,Dn
Dn <- Dn x (<EA>)

MULU <EA>,Dn
Dn <- Dn x (<EA>)

DIVS <EA>,Dn
Dn <- quotient,reste de : Dn : (<EA>)

DIVU <EA>,Dn
Dn <- quotient,reste de : Dn : (<EA>)

INSTRUCTION D 'EXTENSION DU BIT DE SIGNE :

Cette instruction étend le bit de signe (MSB) d 'un entier contenu par un registre Dn d 'un octet à un mot ou d 'un mot à un long mot.
Cele permet de garder la même représentation des nombres (S).

EXT Dn

Taille des opérandes : W,L

ex :

D0.L=$80 soit -128 sur 8 bits
EXT.W D0
D0.L=$FF80 soit toujours -128 sur 16 bits.

On fera toujours des initialisations de registres sur 32 bits pour éviter les effets de bords.

INSTRUCTION DE MISE A 0 :

Cette instruction permet d 'initialiser à 0 le contenu d 'une adresse effective.

CLR <EA>
<EA> <- 0

Il faudra faire attention au temps d 'exécution de CLR par rapport à MOVEQ si l 'on doit optimiser les temps d 'exécution.

4. INSTRUCTIONS POUR ARITHMETIQUE BCD :

Les instructions BCD ressemblent aux instructions d 'arithmétique avec prise en compte du bit X. Elles ne travaillent que sur des octets (2 chiffres BCD)

Codage BCD :

0	0000
1	0001
9	1001
19	00011001 soit un octet

On a 3 instructions ABCD, SBCD et NBCD équivalentes à ADDX, SUBX et NEGX

Addition BCD :

ABCD Dx,Dy
Dy <- Dx_{10} + Dy_{10} + X
ABCD -(Ax),-(Ay)
Ax <- Ax - 1
Ay <- Ay - 1
(Ay) <- $(Ax)_{10}$ + $(Ay)_{10}$ + X

Taille des opérandes : B
Modes d 'adressage : Dn, -(An)
Flags du CCR : affectés

Soustraction BCD :

SBCD Dx, Dy
Dy <- Dy_{10} - Dx_{10} - X
SBCD -(Ax),-(Ay)
Ax <- Ax - 1
Ay <- Ay - 1
(Ay) <- $(Ay)_{10}$ - $(Ax)_{10}$ - X

Taille des opérandes : B
Modes d 'adressage : Dn, -(An)
Flags du CCR : affectés

Opposé d 'un nombre BCD :

NBCD <EA>
<EA> <- 0 - (<EA>)$_{10}$ - X

5. INSTRUCTIONS DE COMPARAISON ET DE TEST :

INSTRUCTIONS DE COMPARAISON

Un programme n 'exécute pas toujours des instructions de façon séquentielle : sauts, branchements, boucles, appels de sous-programmes, interruptions…

On utilise au préalable des instructions de comparaison avant un branchement conditionnel pour affecter les bits du registre CCR. Cela permet ensuite aux instructions de contrôle d 'effectuer ou non le branchement.

On a à disposition 4 instructions de comparaison qui se ramènent à des instructions de soustraction dont on ne sauvegarde par le résultat final.
CMP, CMPA, CMPI, CMPM (comparables à SUB, SUBA, SUBI)

On soustrait l 'opérande source à l 'opérande destination pour affecter les bits du registre CCR sans sauvegarder le résultat final.

Ex :

```
        MOVE.B #20,D0
        ...
        CMP.B #10,D0      * D0.B - #10 ?
        BEQ EGAL          * branchement à l 'adresse de l 'étiquette EGAL si 10
        …                 * D0 est différent de 10
```

On retrouvera toujours la paire instruction de comparaison - instruction de branchement conditionnel

Comparaison :

CMP <EA>,Dn
Dn - (<EA>) ?

Taille des opérandes : B,W,L
Modes d 'adressage : tous
Flags du CCR : affectés

Comparaison d 'adresse :

CMPA <EA>,An
An - (<EA>) ?

Taille des opérandes : W,L
Modes d 'adressage : tous
Flags du CCR : affectés

Comparaison immédiate :

CMPI #<data>,<EA>
(<EA>) - <data> ?

Taille des opérandes : B,W,L
Modes d 'adressage : comme SUBI
Flags du CCR : affectés

Comparaison mémoire :

CMPM (Ay)+,(Ax)+
(Ax) - (Ay) ?
Ax <- Ax + N
Ay <- Ay + N

Taille des opérandes : B,W,L
Modes d 'adressage : (An)+
Flags du CCR : affectés

Instruction spécialisée pour comparer élément par élément 2 chaînes de caractères ou 2 tableaux.

INSTRUCTIONS DE TEST :

Ce sont des instructions de comparaison spécialisées : comparaison d 'un opérande avec la valeur immédiate 0 :
TST, TAS

L 'instruction TST soustrait 0 à l 'opérande et affecte les bits du registre CCR sans sauvegarder le résultat final de la soustraction. Les bits du registre CCR sont affectés comme pour l 'instruction CMP.

Ex :

```
        MOVE.B #20,D0
        ...
        TST.B D0        * D0.B - #0 ?
        BEQ EGAL        * branchement à l 'adresse de l 'étiquette EGAL si 0
        …               * D0 est différent de 0
```

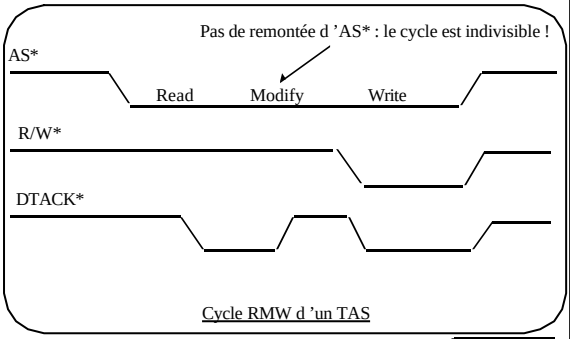

INSTRUCTION DE *TEST AND SET* (TAS) :

Instruction de test et de mise à 1 du MSB d 'un octet.

TAS est similaire à TST : comparaison de l 'opérande avec 0, affectation du CCR et mise à 1 du MSB de l 'octet opérande MAIS :
Le cycle est **indivisible non interruptible : cycle RMW (*Read Modify Write Cycle*)**
inhibition du cycle RETRY sur un BUS ERROR pendant la phase R du cycle RMW

TAS <EA>

Taille des opérandes : B
Modes d 'adressage : tous sauf immédiat
Flags du CCR : affectés



Cycle RMW d 'un TAS

BUT : permettre de mettre en place un environnement multimaître (ex : multiprocesseur 68000). On a besoin d 'accéder de façon indivisible à des variables partagées, mémoire partagée en assurant leur intégrité.

Le maître qui exécute un TAS est le même durant la lecture de la valeur courante de la variable partagée et l 'écriture de la nouvelle valeur d 'où l 'intégrité. L 'accès à la variable partagée se fait donc de façon indivisible par TAS.

TAS permet de mettre en place l 'objet informatique **sémaphore binaire** indispensable dans les environnements multimaîtres/multiprocesseurs mais aussi dans un système d 'exploitation multitâche généralement monoprocesseur-multimaître.

Le bus d 'adresses/données est une ressource partagée dans un environnement multiprocesseur 68000. Le processeur 68000 possède un arbitre de bus intégré qui permet de gérer les demandes de transfert de données :

signaux BR* (*Bus Request*), BG* (*Bus Grant*), BGACK*

Le processeur 68000 est le premier processeur conçu pour être mise en œuvre facilement par les informaticiens :

- mise en œuvre d 'un OS aisé avec 2 niveaux d 'utilisation : bit S
  exemple : S=0 processus UNIX, S=1 mode kernel quand appel système par le processus.
- TAS : mise en place de sémaphores, gestion mémoire partagée…
- arbitre de bus.
- cycle RETRY (voir TP micro UV Libre).

6. REPRESENTATION DES NOMBRES REELS : INTRODUCTION

Le 68000 ne traite que des nombres entiers. Pour les nombres réels, on a recours à :

- un coprocesseur mathématique (FPU), par exemple le composant 68881.
- une bibliothèque d 'émulation logicielle par Développement Limité.

Le format de représentation d 'un nombre réel à virgule flottante est normalisé : IEEE 754 (simple ou double précision)

Il faut discerner les nombres réels :

- à virgule fixe.
- à virgule flottante.

NOMBRES REELS FRACTIONNAIRES A VIRGULE FIXE :

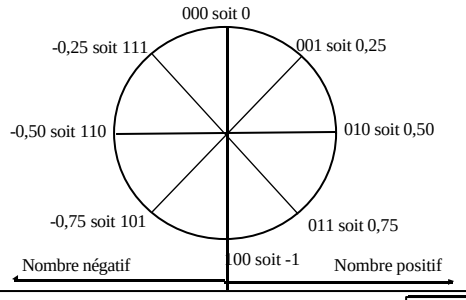
La plupart des applications de calcul scientifique (DSP) utilise des nombres fractionnaires dont les valeurs sont comprises entre -1,0 et +0,999…

Le codage des nombres fractionnaires (à virgule fixe) sur n bits reprend celui des nombres entiers en complément à 2.

Un nombre fractionnaire F sur n bits se représente par :
$F = -2^{0}.b_{n-1} + 2^{-1}.b_{n-2} + \ldots + 2^{-(n-2)}.b_1 + 2^{-(n-1)}.b_0$

b_{n-1} est le bit de poids fort qui correspond au bit de signe.
b_0 est le bit de poids faible.

Exemple : cas n=3



DSP MOTOROLA 56000 :

n=24

b_{23}	b_{22}	b_{21}	…	b_3	b_2	b_1	b_0
-2^{0}	-2^{-1}	-2^{-2}		-2^{-20}	-2^{-21}	-2^{-22}	-2^{-23}

-1<= F <= 0,9999998

bit de signe virgule

NOMBRES REELS MIXTES A VIRGULE FIXE :

Un nombre mixte est un nombre réel composé d 'une partie entière comprenant un bit de signe et une partie fractionnaire, la virgule fixe étant située entre les 2 parties.

Le codage des nombres mixtes à virgule fixe sur n bits reprend celui des nombres entiers en complément à 2.

Un nombre fractionnaire M sur n bits avec i bits pour la partie entière et q bits pour la partie entière (n=i+q) se représente par :
$M = -2^{i-1}.b_{i-1} + 2^{i-2}.b_{i-2} + \ldots + 2^{0}.b_{0} + 2^{-1}.b_{-1} + 2^{-q}.b_{-q}$

b_{i-1} est le bit de poids fort qui correspond au bit de signe.
b_{-q} est le bit de poids faible.

Exemple : cas n=16, i=4, q=12

Soit M = -3,66211. On considère M '=3,66211 et l 'on prendra son complément à 2 :

M ' = 3,66211 0011,101010011000
$Compl_1$(M ') = 1100,010101100111
+1 + 0000,000000000001
M = $Compl_2$(M ')= 1100,010101101000 = -3,66211

NOMBRES REELS A VIRGULE FLOTTANTE :

Un besoin de normalisation des nombres à virgule flottante est apparu très tôt car chaque fondeur de processeur DSP avait sa norme d 'où la norme IEEE 754/854 en 1985.

Cette norme définit 2 formats de nombres « flottants » :
- réel simple précision sur 32 bits (*float* en langage C).
- réel double précision sur 64 bits (*double* en langage C).

NOMBRE REEL SIMPLE PRECISION

31 30 23 22 0
| Signe | 8 bits pour l 'exposant e | 23 bits pour la partie fractionnaire f |
| s | e_7 e_6 … e_1 e_0 | f_{22} f_{21} … f_2 f_1 f_0 |

NOMBRE REEL DOUBLE PRECISION

63 62 52 51 0
| Signe | 11 bits pour l 'exposant e | 52 bits pour la partie fractionnaire f |
| s | e_{10} e_9 … e_1 e_0 | f_{51} f_{50} … f_2 f_1 f_0 |

NB : non à l 'échelle

La partie fractionnaire f est codée sur 23 ou 52 bits.

L 'exposant e est codé sur 8 ou 11 bits. Il est décalé ou biaisé de 127 ou 1023 : E = e-127 ou E = e-1023. E est l 'exposant non biaisé.

Le bit de signe S est sur 1 bit.

L 'exposant e détermine le type de nombre codé. 5 types sont définis dans la norme :
- les nombres normalisés dont la mantisse est >= 1 et < 2.
- les nombres dénormalisés dont la mantisse est >= 0 et < 1.
- les nombres +infini et -infini.
- les nombres indéfinis NaN (*Not a number*).
- le nombre 0.

NOMBRE REEL NORMALISE :

Le nombre réel normalisé R est représenté par : $R = (-1)^S.2^E.1,f$ (1,f : mantisse)

E = e-127 ou E = e-1023
On a : 1 <= e <= 254 ou 1 <= e <= 2046 soit :
-126 <= E <= 127 ou -1022<= E <= 1023

La mantisse est 1.0 + la partie fractionnaire soit 1,f

Exemple : $+10 = (-1)^0.2^3.1,25$ soit :
- en simple précision sur 32 bits : E =3, e=E+127=130
soit le format IEEE 754 binaire : 0 10000010 01000000000000000000000
soit en hexadécimal sur 32 bits : $41200000
- en double précision sur 64 bits : $4024000000000000

NOMBRE REEL NORMALISE :

Valeur la plus grande positive sur 32 bits : S=0, E=127, e=254, f=1…1 soit $3,4.10^{+38}$ soit exactement $(2-2^{-23}).2^{127}$
Valeur la plus petite positive sur 32 bits : S=0, E= -126, e=1, f=0…1 soit $1,1.10^{-38}$ soit exactement $(1-2^{-23}).2^{-126}$

Valeur la plus grande positive sur 64 bits : S=0, E=1023, e=2046, f=1…1 soit $1,8.10^{+308}$ soit exactement $(2-2^{-52}).2^{1023}$
Valeur la plus petite positive sur 64 bits : S=0, E= -1022, e=1, f=0…1 soit $2,2.10^{-308}$ soit exactement $(1-2^{-52}).2^{-1022}$

NOMBRE REEL DENORMALISE :

Le nombre réel dénormalisé R est représenté par : $R = (-1)^S.2^{Emin}.\underline{0.f}$

Emin = -126 ou Emin = -1022 soit e=0

Les nombres dénormalisés ont une faible amplitude comprise entre $2^{-126}.(1-2^{-23})$ et $2^{-126}.2^{-23}$ sur 32 bits.

AUTRES NOMBRES REELS :

- Zéro : s=0, e=0, f=0…0, mantisse=0,f soit sur 32 bits : $00000000
- +infini : s=0, e=255, f=0…0, mantisse=0,f soit sur 32 bits $7F800000
- -infini : s=1, e=255, f=0…0, mantisse=0,f soit sur 32 bits $FF800000

- NaN : c 'est par exemple 0 multiplié par l 'infini.
- NaN : s=0, e=255, f=1…1, mantisse=1,f soit sur 32 bits $7FFFFFFF