Initiation à l'Assembleur (5)

par André C. et tous ceux qui voudront bien y participer...

RAPPELS SUR LES NOMBRES SIGNES

Avant d'aborder la délicate question des opération arithmétiques, je voudrais faire une parenthèse sur les nombres signés. Un octet s'écrit toujours de #00 à #FF. Mais il peut être vu comme un nombre entier (donc de 0 à 255 en décimal) ou comme un nombre signé. Dans ce dernier cas, son b7 sert de drapeau de signe, tandis que les 7 autres bits décrivent la valeur du nombre.

Ceci est aussi valable pour les grands nombres, qui sont écrits sur plusieurs octets, sachant que c'est le b7 de l'octet de poids fort qui représente le signe. Rappel : En langage machine, l'écriture des octets constituant les grands nombres se fait dans l'ordre inverse, celui de poids le plus faible en tête et celui de poids le plus fort en dernier. Par exemple, un nombre sur deux octets s'écrit LLHH. Ainsi le nombre #0100 (256 en décimal) sera découpé en #00 #01 pour être utilisé en langage machine. Pour simplifier, dans tout ce qui suit, on ne prendra en considération que des nombres sur un octet.

Les nombres signés positifs

Ces nombres ne posent aucun problème de lecture. Leur b7=0 et les bits restant indiquent directement la valeur du nombre. Puisque ce nombre est codé sur 7 bits au lieu de 8, il ne pourra prendre qu'une valeur de 000 0000 à 111 1111 en binaire soit de #00 à #7F en hexadécimal et de 0 à 127 en décimal.

Les nombres signés négatifs

Là, la situation est un peu plus compliquée, car non seulement leur b7=1 indique qu'ils sont négatifs, mais les bits restants, qui indiquent la valeur du nombre sont codés en 'complément à deux'. Ah que la vie est compliquée ! Le nombre décimal -3 ne peut pas s'écrire 1000 0011, ce qui aurait été trop simple. En effet si vous essayez d'ajouter +5 et -3 en binaire, vous allez obtenir -8 !

Evidemment, il a donc fallu trouver un truc. Je passerai sur la démonstration de la chose, sachez seulement que les nombres négatifs doivent être représentés en **'complément à 2'**. Sans entrer dans les détails voici la recette pour coder un nombre négatif : **Inverser chacun des 8 bits du nombre positif correspondant. Chaque 0 est transformé en 1 et chaque 1 en 0. Ajouter ensuite 1.**

Exemple: 5 s'écrivant 0000 0101, pour obtenir -5, inversez tous les bits, ce qui donne 1111 1010, puis ajoutez 1, ce qui donne 1111 1011.

Inversement pour interpréter un nombre signé négatif, il faut partir du complément à 2 pour retrouver ce nombre. Exemple pour interpréter le nombre signé négatif 1111 1011, il faudrait retirer 1 puis inverser les 8 bits (c'est à dire les opérations habituelles effectuées dans l'ordre inverse). Mais curieusement on peut aussi d'abord inverser les 8 bits puis ajouter 1, ce qui est étonnant, mais plus simple ! C'est la beauté de ce truc de complément à 2 ! Démonstration (sachant que pour retirer 1, il faut ajouter le complément à 2 de 1 soit 1111 1111) :

```
Soit          1111 1011      Ou encore    1111 1011
inversion    +0000 0100      +(-1)       +1111 1111
+(1)          0000 0001      ----------------------
----------------------                   =1111 1010 (et C=1)
On a bien(5)=0000 0101      inversion     0000 0101 (c'est à dire 5)
```

On peut vérifier les additions suivantes (valeur décimale entre parenthèse) :

```
 (3)          0000 0011        (3)         0000 0011
+(5)         +0000 0101      +(-5)        +1111 1011
------------------           ----------------
=(8)         =0000 1000      =(-2)        =1111 1110
```

Le dernier résultat 1111 1110 représente bien -2 (complément à 2 de 000 0010). Pour interpréter le résultat d'une addition dont le b7 est à 1, il faut donc en calculer le complément à 2.

```
Résultat     1111 1110      Ou encore    1111 1110
inversion   +0000 0001      +(-1)       +1111 1111
+ (1)        0000 0001      ----------------
------------------                       =1111 1101 (et C=1)
Soit (2)    =0000 0010      inversion    0000 0010 (c'est à dire 2)
```

On voit encore que pour interpréter un nombre signé négatif binaire, le plus simple est d'en calculer le complément à 2 en respectant l'ordre les deux opérations 'inversion des bits' puis 'incrémentation'. Le complément à 2 d'un complément à 2 redonne le nombre d'origine.

Voici un tableau, pris dans "Programmation du 6502" de Rodnay Zaks (Sybex 1980), qui s'intitule "Table des compléments" et qui donne le complément à 2 de quelques nombres compris entre +127 et -128. Notez bien que les nombres positifs ont exactement la même représentation que les nombres non signés (normaux) et que leur attribuer un "complément à 2" est un peu abusif. La seule différence est que leur b7 étant obligatoirement à 0, on ne peut les utiliser que dans l'intervalle de 0 à +127.

+	CODE EN COMPLEMENT A 2	−	CODE EN COMPLEMENT A 2
+ 127	01111111	− 128	10000000
+ 126	01111110	− 127	10000001
+ 125	01111101	− 126	10000010
		− 125	10000011
+ 65	01000001	− 65	10111111
+ 64	01000000	− 64	11000000
+ 63	00111111	− 63	11000001
+ 33	00100001	− 33	11011111
+ 32	00100000	− 32	11100000
+ 31	00011111	− 31	11100001
+ 17	00010001	− 17	11101111
+ 16	00010000	− 16	11110000
+ 15	00001111	− 15	11110001
+ 14	00001110	− 14	11110010
+ 13	00001101	− 13	11110011
+ 12	00001100	− 12	11110100
+ 11	00001011	− 11	11110101
+ 10	00001010	− 10	11110110
+ 9	00001001	− 9	11110111
+ 8	00001000	− 8	11111000
+ 7	00000111	− 7	11111001
+ 6	00000110	− 6	11111010
+ 5	00000101	− 5	11111011
+ 4	00000100	− 4	11111100
+ 3	00000011	− 3	11111101
+ 2	00000010	− 2	11111110
+ 1	00000001	− 1	11111111
+ 0	00000000	− 0	00000000

Retenue (C pour Carry) et débordement (V pour oVerflow)

Nous verrons que les commandes du 6502 permettent d'effectuer toutes les opérations arithmétiques nécessaires, sans qu'il soit utile de pratiquer soit-même toute cette cuisine indigeste. Tout ça, c'est bien beau, mais il faut savoir comment interpréter le résultat qui atterri dans l'accumulateur A ! En effet, ce registre A ne contient qu'un octet, c'est à dire un nombre sur 8 bits. Alors, le résultat, c'est quoi?

Nous avons déjà évoqué les drapeaux C et V. Ce dernier vous semble bien mystérieux ? Il n'y a pas de quoi, car C et V ont une grande parenté. C reflète la retenue qui passe de b7 à un b8 fictif pour les opérations portant sur les nombres non signés. V 'reflète' la retenue qui passe de b6 à b7 pour les opérations portant sur les nombres signés (lesquels sont codés sur 7 bits, le 8e n'étant qu'un drapeau de signe). En fait, ce drapeau V est 'corrigé'. Je me permets de reprendre mot à mot ce que j'avais écrit sur le drapeaux V:

«Drapeau V : o**V**erflow c'est l'indicateur le plus complexe et le moins utilisé. Il sert à savoir s'il y a eu dépassement de capacité lors d'une opération arithmétique. Lorsque les nombres non signés (de 0 à 255), codés sur 8 bits dépassent 255, une retenue passe de b7 à 'b8'. Cette retenue est gardée dans Carry. Vous imaginerez facilement que la chose est plus compliquée avec les nombres signés. Les nombres signés (de -128 à +127) étant en fait codés sur 7 bits, il y a overflow lorsqu'ils dépassent +127 ou lorsqu'ils sont < -128. Mais à cause des soustractions, il n'était pas facile de trouver un indicateur qui marche dans tous les cas, c'est pourquoi il est très compliqué : L'indicateur V est obtenu en effectuant un OU exclusif (XOR) entre la retenue lors du passage de b6 à b7 et la retenue lors du passage de b7 à 'b8' (carry). Ainsi si l'une de ces 2 retenues (mais pas les 2) est à 1, alors V est à 1.**»**

Il ne me semble pas utile de faire la démonstration théorique de manière dont V est élaboré. Cela me demanderai plusieurs pages et je ne suis même pas sûr d'en être capable. Mais par contre, je peux résumer comment on utilise C et V et donner quelques exemples.

Calculs avec les nombres non signés

Là, pas de problème, on ne s'occupe que de C. Si C passe à 1 à l'issue d'une addition, c'est qu'au

résultat contenu dans l'accumulateur A, il faut ajouté un 8e bit. C'est très pratique sur les grands nombres (dont l'écriture nécessite plusieurs octets) car il faut reporter cette retenue C dans l'addition des octets suivants. C'est pour ça que la commande s'appelle ADC (addition avec retenue). Pour ça aussi qu'au départ il faut mettre la retenue à 0 avec la commande CLC. Nous verrons plus tard le détail de la chose.

Calculs avec les nombres signés

Dans l'arithmétique avec les nombres signés, on ignore la retenue C (elle est en fait 'incorporée' dans V). Par contre si V passe à 1, alors le résultat est faux.

Exemples d'addition de nombres signés positifs :

```
 0000 0110  ( +6)          0111 1111  (+127)
+0000 1000  ( +8)         +0000 0001  (  +1)
----------------          ----------------
=0000 1110  (+14)         =1000 0000  (-128)
Avec V=0 et C=0           Avec V=1 et C=0
Résultat exact            Résultat faux
```

Exemples d'addition de nombres signés négatifs :

```
 1111 1010  ( -6)          1000 0001  (-127)
+1111 1000  ( -8)         +1100 0010  ( -62)
----------------          ----------------
=1111 0010  (-14)         =0100 0011  ( +67)
Avec V=0 et C=1           Avec V=1 et C=1
Résultat exact            Résultat faux
```

Exemples d'addition de nombres signés positif + négatifs avec résultat positif :

```
 0000 0100  ( +4)
+1111 1110  ( -2)
----------------
=1111 0010  ( +2)
Avec V=0 et C=1
Résultat exact
```

Exemples d'addition de nombres signés positif + négatifs avec résultat négatif :

```
 0000 0010  ( +2)
+1111 1100  ( -4)
----------------
=1111 1110  ( -2)
Avec V=0 et C=0
Résultat exact
```

Les nombres signés devant être compris entre -128 et +127, toute opération dont le résultat sort de cet intervalle est illégitime. L'addition de deux grand nombres signés positifs positifs (résultat supérieur à 127) ou celle de deux grands nombres signés négatifs positifs (résultat inférieur à -128) est donc risquée. Il en est de même pour la soustraction d'un grand nombre positif à un grand nombre négatif ou celle d'un grand nombre négatif à un grand nombre positif.

Nombres codés sur plusieurs octets

Vous savez tous qu'avec 2 octets on peut écrire des nombres compris entre #0000 et # FFFF, c'est à dire entre 0 et 65535. Si on veut aller plus loin, il faut utiliser des nombres à 3, 4, ... , N octets.

Et les nombres signés ? Le principe est le même : le b7 de l'octet de poids le plus fort est utilisé pour le signe. Tous les autres bits servent à coder la valeur. Sans entrer dans les détails, voici quelques exemple de nombres signés sur 2 octets :

`00000000 00000000` représente 0
`00000000 00000001` représente 1
`01111111 11111111` représente +32767 (limite supérieure des nombres signés positifs sur 2 octets)
`11111111 11111111` représente -1
`11111111 11111110` représente -2
`10000000 00000000` représente -32768 (limite inférieure des nombres signés négatifs sur 2 octets)

à suivre...