L'ABC DE LA MICRO-INFORMATIQUE

LE JEU D'INSTRUCTIONS DU 8088

Nous poursuivons aujourd'hui notre présentation du jeu d'instructions du 8088 avec les instructions arithmétiques, suite classique des instructions logiques vues le mois dernier.

Nous n'allons pas revenir sur la représentation des nombres en mémoire car nous y avons consacré assez de temps dans un précédent article de cette série ; article auquel vous voudrez bien vous reporter en cas de besoin. Nous allons seulement mettre en lumière une particularité du 8088 qui peut travailler avec trois types de représentation : le binaire pur (un octet peut alors représenter n'importe quel nombre compris entre – 128 et + 127), le BCD non compacté (un octet représente alors un chiffre en codage BCD, c'est-à-dire un chiffre entre 0 et 9 ; en fait, seuls les quatre bits de poids faible sont utilisés) et le BCD compacté (un octet représente deux chiffres de 0 à 9 en exploitant les deux fois 4 bits de l'octet).

Ces diverses formes de représentation conduisent le programmeur du 8088 à faire appel à certaines instructions spéciales dites d'ajustement afin que les opérations arithmétiques exécutées soient exactes quel que soit le type de représentation choisi. Nous préciserons cela sur des exemples dans un instant.

La première instruction arithmétique est l'addition qui existe sous deux formes : ADD (addition de deux opérandes) et ADC (addition de deux opérandes avec retenue). La syntaxe et les règles d'adressage sont communes aux deux instructions à savoir :

ADD opérande 1, opérande 2
ou ADC opérande 1, opérande 2.

Dans le premier cas, l'opérande 2 est ajouté à l'opérande 1 et le résultat est mis à la place de l'opérande 1 qui est donc détruit. Dans le deuxième cas, le bit de retenue CF du registre d'état du 8088 est ajouté au résultat, ce qui permet de réaliser des opérations en multiple précision.

Les deux instructions, après avoir fait l'addition, positionnent le bit de retenue CF du registre d'état en fonction du résultat ainsi que, si nécessaire, les bits PF, ZF, SF et OF en fonction de leur signification propre (revoir la présentation du registre d'état si nécessaire).

Tous les modes d'adressage sont autorisés pour l'opérande 2 tandis que, pour l'opérande 1, seul le mode immédiat est interdit. C'est tout à fait logique puisque l'opérande 1 est le destinataire du résultat. Par ailleurs, il est bien évident que deux adressages simultanés en mémoire centrale sont interdits comme dans le cas de l'instruction MOV vue le mois dernier.

Les deux opérandes doivent être de même taille (octet ou mat de 16 bits) ; par contre, l'opérande 2 peut être un octet alors que l'opérande 1 peut être un mot. L'inverse est évidemment interdit car le résultat de l'opération ne pourrait être codé sur un octet. Enfin, dernière restriction, les registres de segments ne doivent pas être utilisés pour contenir les opérandes.

OPERATIONS EN DOUBLE PRECISION

L'instruction ADD permet d'ajouter deux octets ou deux mots de 16 bits. Compte tenu des possibilités de codage binaire, cela limite les capacités arithmétiques du 8088 à des résultats inférieurs ou égaux à 65535, ce qui est assez peu. Fort heureusement, grâce à ADC, il est possible de manipuler des nombres en double, voire en multiple précision. Nous allons voir un exemple en double précision qui permet de manipuler des mots de 32 bits et, donc, d'atteindre des valeurs de l'ordre de 4 milliards.

Supposons que nous souhaitions ajouter le mot de 32 bits contenu à l'adresse 100 au mot de 32 bits contenu à l'adresse 200 et que le résul-

tat soit à placer dans la paire de registres AX et BX avec les poids faibles dans AX. Le petit programme suivant va s'en charger :

```
MOV AX, [100]
MOV BX, [102]
ADD AX, [200]
ADC BX, [202]
```

En effet, la première instruction charge les poids faibles du premier mot de 32 bits dans AX. La seconde fait de même, mais avec les poids forts (rappelons que le 8088 range les nombres en mémoire avec les poids faibles et les poids forts aux adresses les plus fortes). L'instruction ADD ajoute les poids faibles de nos deux mots de 32 bits et positionne, si nécessaire, le bit de retenue CF. L'instruction ADC additionne alors les poids forts en tenant compte de la retenue éventuelle provenant de la somme des poids faibles. L'addition globale est donc parfaitement exacte.

LES ADDITIONS EN BCD

Les additions ADD et ADC ne sont pas directement utilisables si les nombres sur lesquels on travaille sont codés en BCD ; en effet, ADD et ADC font des additions en binaire pur. En voici un exemple concret qui suppose que l'on souhaite ajouter les deux nombres décimaux 25 et 47. On peut écrire :

```
MOV AL,25
MOV BL,47
ADD AL,BL
```

et l'on constatera avec surprise que AL contient 6C et non le 72 auquel nous nous attendions. En effet, ADD a fait une addition en binaire pur et cette dernière n'est donc exacte que si les nombres sont exprimés en hexadécimal. Pour que notre addition devienne exacte en décimal il faut utiliser l'instruction DAA qui signifie Decimal Adjust for Addition (ajustement décimal, pour addition). Cette instruction s'utilise sans opérande car elle n'agit que sur le contenu du registre AL qu'elle transforme de binaire pur en BCD compacté. Pour que l'exemple vu ci-avant soit correct, il suffit donc de lui adjoindre DAA de la façon suivante :

```
MOV AL,25
MOV BL,47
ADD AL,BL
DAA
```

pour trouver 72 dans AL et non plus 6C.
Cette instruction présente cependant un inconvénient majeur qui est de ne positionner correctement que le bit CF du registre d'état. Il ne faut donc en aucun cas utiliser les autres bits de ce registre (pour des instructions de branchement conditionnel par exemple) suite à un DAA car cela n'aurait aucun sens.
Une autre instruction d'ajustement pour addition existe sous le mnémonique AAA (ASCII Adjust for Addition). Elle est l'équivalent de DAA, mais pour les nombres représentés en BCD non compacté. Son utilisation est assez peu courante car ce mode de représentation gaspille beaucoup de place en mémoire. En revanche, elle permet de transformer les nombres de 0 à 9 codés en ASCII en leur représentation binaire. La seule précaution à prendre est de placer au préalable ces nombres dans AL puisque, comme DAA, AAA n'agit que sur le contenu de AL. Ainsi, si AL contient 37 (code ASCII de 7), le fait de faire un AAA laissera 7 dans AL.

LES INSTRUCTIONS DE SOUSTRACTION

Elles sont le complément exact des instructions d'addition vues ci-avant puisque l'on trouve :

SUB, équivalent de ADD
SBB, équivalent de ADC
DAS, équivalent de DAA

et enfin AAS, équivalent de AAA.
Nous n'allons pas à nouveau détailler l'utilisation et le rôle exact de toutes ces instructions car tout ce que nous avons dit pour les instructions d'addition reste valable. Précisons seulement, pour le lecteur non habitué, que SBB signifie SuBstract with Borrow, c'est-à-dire soustraction avec retenue (contrairement au français, la retenue de la soustraction ne porte pas, en américain, le même nom que celle de l'addition). C'est bien évidemment le bit de retenue CF qui est utilisé pour SBB, comme c'était le cas pour ADC.
Les exemples d'addition de deux mots de 32 bits et d'addition en BCD avec ajustement décimal ensuite sont immédiatement transposables avec la soustraction en échangeant les mnémoniques compte tenu des équivalences vues ci-avant. Nous vous laissons le soin de le réaliser.

POUR AJOUTER OU SOUSTRAIRE 1 ET POUR CHANGER DE SIGNE

Lorsque l'on veut augmenter ou diminuer d'une ou deux unités le contenu d'un registre ou d'une adresse mémoire, il n'est pos nécessaire d'utiliser ADD ou SUB car deux instructions sont spécifiquement prévues pour cela ; ce sont INC et DEC. La première augmente de un le contenu d'un registre ou d'une mémoire (octet ou mot de 16 bits) et la seconde le diminue de un. Tous les modes d'adressage, sauf l'adressage immédiat qui n'aurait aucun sens, sont utilisables. En outre, les registres de segments ne doivent pas être modifiés par ces instructions. La syntaxe est très simple :
INC opérande ou DEC opérande, où l'opérande est un registre du 8088 ou une adresse mémoire.
Lorsque l'on veut obtenir l'opposé d'un nombre, il est possible de le soustraire à 0 mais c'est long et peu élégant car il existe l'instruction NEG qui sait faire cela très bien. Cette instruction ne fait appel qu'à un opérande sous la forme :
NEG opérande
L'opérande est alors changé en son opposé, c'est-à-dire en 0-opérande. Tous les modes d'adressage hors le mode immédiat peuvent être utilisés et il est interdit de modifier ainsi un registre de segment.

LE 8088 CONNAIT SA TABLE DE MULTIPLICATION...

Instruction réservée il y a encore quelques années aux microprocesseurs de haut de gamme, la multiplication se « démocratise » et se rencontre maintenant sur de nombreux microprocesseurs ordinaires tels le 6809 de Motorola ou le 8088 qui nous intéresse aujourd'hui. Comble de raffinement, le 8088 nous offre deux instructions de multiplication : MUL et IMUL. La première est une multiplication non signée alors que IMUL considère que les opérandes sont signés. La syntaxe des deux instructions est identique :
MUL opérande ou IMUL opérande
Le multiplicande se trouve toujours dans AX pour un mot de 16 bits ou dans AL pour un octet. Le seul opérande qui

apparaît dans le corps de l'instruction est donc le multiplicateur. Tous les modes d'adressage, hors l'adressage immédiat, sont autorisés. Dans le cas de multiplication de deux octets, le résultat (un mot de 16 bits donc) est placé dans AX. Dans le cas de mots de 16 bits (résultat sur 32 bits donc), le résultat est stocké dans AX et DX avec les poids forts dans ce dernier.

En ce qui concerne MUL, les indicateurs CF et OF du registre d'état sont positionnés correctement après exécution de l'instruction. Pour IMUL, ce n'est pas le cas et il faut donc y prendre garde.

Ces deux instructions manipulent évidemment des nombres en binaire pur ou signé mais une possibilité, restreinte il est vrai, de multiplication décimale existe grâce à AAM (ASCII Adjust for Multiply). Cette instruction, comme AAA et AAS, ne sait traiter que les nombres décimaux en représentation BCD non compactée. De ce fait, elle n'est utilisable qu'après des multiplications d'octets non signés et le résultat de la multiplication est donc supposé être dans AX. Voici un exemple d'emploi :

```
MOV AL,8
MOV BL,6
MUL BL
AAM
```

Le contenu de AX est alors 0408, c'est-à-dire 48 en représentation BCD non compactée (un chiffre décimal par octet, rappelons-le).

... ET IL SAIT AUSSI DIVISER

Malgré les progrès de la micro-informatique, la division reste une instruction peu courante sur les microprocesseurs 8/16 bits, et le 8088 fait figure d'original en proposant non pas une mais deux instructions de division : DIV et IDIV. Vu les mnémoniques, vous avez deviné que DIV était une division non signée tandis que IDIV était signée. DIV et IDIV s'utilisent de la façon suivante :

DIV opérande ou IDIV opérande, où opérande peut être un mot de 16 bits ou un octet qui représente le diviseur et qui peut être un registre ou une adresse mémoire. Le dividende est de longueur double de celle du diviseur. Il est alors contenu dans AX si le diviseur est un octet et dans DX et AX si le diviseur est un mot de 16 bits. Le quotient de la division est stocké dans AL ou AX selon sa taille (octet ou mot respectivement) et le reste est placé dans AH ou DX (idem).

Le registre d'état n'est pas positionné par les instructions de division ; par contre, si le quotient dépasse la capacité du registre destinataire, le 8088 génère une interruption de type 0 (nous verrons un peu plus tard ce que cela signifie) qu'il vous appartient donc de traiter si vous ne voulez pas que votre programme se mette à faire n'importe quoi dans ce cas.

Comme pour la multiplication, une forme restreinte de l'ajustement vu pour addition et soustraction existe. Le mnémonique de l'instruction est AAD (ASCII Adjust for Divide) mais le mode d'emploi est différent de ce que nous avons vu jusqu'à présent ; l'ajustement doit en effet se faire sur le dividende AVANT l'exécution de l'opération. De plus, comme il est uniquement possible de travailler sur des octets, le dividende est stocké sous forme étendue dans AX (dizaines dans AH et unités dans AL), le quotient est placé dans AL et le reste dans AH. Voici un exemple d'utilisation :

```
MOV AX,0407
AAD
MOV BL,6
DIV BL
```

Ce qui donne comme contenu final de AX 0507 puisque le quotient est 07 et le reste 05.

LES EXTENSIONS DE SIGNES

Nous avons vu que toutes les instructions arithmétiques pouvaient travailler sur des octets ou des mots de 16 bits. Pour que cela puisse donner des résultats corrects dans tous les cas, et compte tenu du principe de la représentation en binaire signé, il est nécessaire de disposer d'instructions d'extension de signe pour pouvoir passer d'un octet signé à un mot signé ou d'un mot signé à un double mot signé. En effet, rappelons que le bit de poids fort d'un mot binaire est son signe (0 pour un nombre positif et 1 pour un nombre négatif). Pour transformer un octet signé en mot de 16 bits signé, il faut donc recopier ce bit de signe dans les 8 bits de poids fort du mot de 16 bits ainsi constitué. Une telle opération est réalisée par l'instruction CBW (Convert Byte to Word) qui s'utilise sans opérande car elle transforme l'octet contenu dans AL en un mot de 16 bits placé dans AX.

Si AL contient 0010 1001, après un CBW, AX contiendra 0000 0000 0010 1001. Par contre, si AL contient 1001 1101, après un CBW, AX contiendra 1111 1111 1001 1101. Dans les deux cas, il y a eu extension de signe de AL dans AX au travers de AH.

L'instruction CWD s'utilise de la même façon mais pour transformer un mot de 16 bits en long mot de 32 bits (Convert Word to Double word). Le mot doit être contenu dans AX et le double mot obtenu est dans AX et DX avec les poids forts dans DX.

NE VOUS AFFOLEZ PAS !

Tout ce que nous venons de voir aujourd'hui peut sembler un peu complexe pour qui n'est pas habitué aux microprocesseurs. C'est tout à fait normal car, comme nous l'avons expliqué au début de cette série, les possibilités arithmétiques de ces derniers sont assez limitées. Dans la pratique, et hormis quelques cas particuliers où une très grande rapidité de calcul est nécessaire, la programmation en assembleur est très peu utilisée pour faire du calcul pur ; on lui préfère un langage évolué qui facilite grandement de telles opérations. En toute logique, vous aurez donc rarement à manipuler de telles instructions dans des programmes en assembleur et, lorsque ce sera le cas, ce sera presque toujours pour des opérations arithmétiques relativement limitées.

Si les règles d'utilisation des diverses instructions présentées vous semblent nombreuses et délicates, relisez calmement cet article, vous constaterez alors qu'elles sont tout à fait cohérentes car elles se retrouvent identiques à elles-mêmes pour quasiment toutes les instructions.

CONCLUSION

Nous terminerons cette présentation le mois prochain et nous pourrons alors écrire quelques exemples de programmes et continuer notre voyage au cœur de la micro-informatique en abordant d'autres sujets tout aussi intéressants.

C. TAVERNIER