Quatrième partie

Le langage machine du microprocesseur 8088 : instructions fondamentales

Chapitre 13

Les instructions de transferts

Nous avons vu quelques instructions de transfert dans le chapitre précédent de façon à pouvoir donner un premier programme concret. Cependant toutes les possibilités de transfert n'ont pas été explorées. Nous allons poursuivre l'exploration des transferts possibles dans ce chapitre, sans en voir toutes les possibilités (en particulier sur l'adressage indexé, que nous aborderons au chapitre 23). Commençons, avant cela, par le principe du stockage des mots.

13.1 Principe du stockage des mots chez *Intel*

Un mot (de seize bits) est constitué de deux octets : un appelé **octet de poids fort** (**MSB** pour l'anglais *Most Significant Byte*) et l'autre **octet de poids faible** (**LSB** pour l'anglais *Least Significant Byte*). La dénomination provient de ce que, lorsque ce mot représente un entier naturel, on veut que :

$$val(w) = 256 \times val(MSB) + val(LSB).$$

Comment coder w : par MSB LSB ou par LSB MSB ? Il s'agit d'une vieille discussion entre informaticiens, la bataille des petits boutiens (*little endian* en anglais) contre les grands boutiens (*great endian* en anglais), par référence aux *Voyages de Gulliver* de Daniel Defoe dans lesquels on discute comment casser un œuf pour le gober : par le grand bout ou le petit bout.

Cela dépend des concepteurs de microprocesseur. *Intel* a choisi l'ordre LSB MSB, qui n'est pas nécessairement l'ordre auquel on pourrait s'attendre.

Il y a donc deux façons de stocker un mot de deux octets, disons `AB`, à l'adresse n :

— La *technique petitboutienne* place `B` à l'adresse n et `A` à l'adresse $n+1$.

— La *technique grandboutienne* place `A` à l'adresse n et `B` à l'adresse $n+1$.

Tous les microprocesseurs *Intel* et plusieurs mini-ordinateurs, tels que les VAX de Digital, utilisent la technique petitboutienne. Les microprocesseurs Motorola (utilisés dans les premiers Macintosh) et les grands systèmes utilisent la technique grandboutienne.

Remarque.- Cette différence dans le codage des mots pose un problème pour traduire un logiciel d'un système à l'autre.

13.2 Autres instructions de transfert

13.2.1 Classification des instructions de transfert

La modélisation des ordinateurs que nous avons donnée ci-dessus nous conduit à distinguer quatre types d'instructions de transfert :

- l'**initialisation d'un registre** qui permet de donner directement une valeur à un registre ;
- les **transferts entre registres** qui permettent de transférer une valeur d'un registre dans un autre ;
- les **accès à la mémoire vive** qui comprennent, d'une part, l'initialisation d'un élément de mémoire vive et, d'autre part, le transfert d'une valeur d'un registre vers un élément de mémoire vive, ou *vice-versa* ;
- les **entrées-sorties** qui permettent de transférer une valeur d'un registre vers un **périphérique**, ou *vice-versa*.

Nous avons déjà vu l'initialisation d'un registre général, le transfert entre registres et l'initialisation d'un élément de mémoire vive. Nous étudierons les entrées-sorties dans le chapitre suivant. Voyons dans ce chapitre les autres types de transfert.

13.2.2 Stockage du contenu de l'accumulateur dans un élément de mémoire vive

Langage symbolique.- L'instruction de stockage du contenu de l'accumulateur dans un élément de mémoire vive s'écrit :

```
MOV [decalage], A
```

où `A` désigne l'un des registres `AL` ou `AX` et `decalage` le décalage de l'adresse, qui est un entier de seize bits. Le segment par défaut est le segment de données.

Sémantique.- La signification de cette instruction est la copie du contenu de l'accumulateur dans l'élément de mémoire vive d'adresse spécifiée.

Langage machine.- Ce transfert se code sur trois octets :

```
| opcode | faible | fort |
```

où `faible` est l'octet de poids faible du décalage de l'adresse, `fort` l'octet éventuel de poids fort et `opcode` est :

```
1010 001w
```

avec, évidemment, `w` égal à 0 pour `AL` et à 1 pour `AX`, soit A2h ou A3h.

Exercice corrigé.- *Traduire l'instruction* :

```
MOV [A900h], AX
```

en langage machine.

Nous prendrons à partir de maintenant la convention `debug` pour le langage symbolique, qui consiste à écrire tous les entiers en hexadécimal. D'après ce que nous venons de dire, cette instruction se traduit par les trois octets :

```
| 1010 0011 | 0000 000 | 1010 1001 |
```

en binaire, soit A3 00 A9h.

Vocabulaire.- Il est traditionnel de parler d'**adressage direct**, d'**adressage absolu** ou d'**adressage étendu** dans le cas de copie entre accumulateur et mémoire.

13.2.3 Chargement d'un élément de mémoire vive dans l'accumulateur

Langage symbolique.- L'instruction de chargement d'un élément de mémoire vive dans l'accumulateur s'écrit :

```
MOV A, [decalage]
```

où `A` désigne l'un des registres `AL` ou `AX` et `decalage` le décalage de l'adresse, qui est un entier de seize bits. Le segment par défaut est le segment de données.

Sémantique.- La signification de cette instruction est la copie du contenu de l'élément de mémoire vive d'adresse spécifiée vers l'accumulateur.

Langage machine.- Ce transfert se code sur trois octets :

```
| opcode | faible | fort |
```

où `faible` est l'octet de poids faible du décalage de l'adresse, `fort` l'octet éventuel de poids fort et `opcode` est :

```
1010 000w
```

avec, évidemment, `w` égal à 0 pour `AL` et à 1 pour `AX`, soit A0h ou A1h.

Exercice corrigé.- *Traduire l'instruction* :

```
MOV AX, [A900]
```

en langage machine.

D'après ce que nous venons de dire, cette instruction se code par les trois octets :

```
| 1010 0001 | 0000 000 | 1010 1001 |
```

en binaire, soit A1 00 A9h.

13.2.4 Transfert entre registres ou registre et case mémoire

Langage symbolique.- L'instruction de transfert entre registres ou registre et case mémoire s'écrit :

```
MOV destination, source
```

ce qui ne nous apprend pas grand chose.

Langage machine.- Cette instruction se code par deux octets dans le cas de registres et par deux ou quatre octets dans le cas d'une case mémoire :

```
| 1000 10dw | mod reg r/m |           |           |
```

— Dans le premier octet d'opcode, le bit `d` (pour *Destination*) est égal à 1 si la destination est un registre général : celui-ci est alors spécifié par le champ `reg` du deuxième octet d'opcode. Le bit `w` est égal à 0 pour le transfert d'un octet et à 1 pour le transfert de deux octets (un mot).

— Le deuxième octet du code opération, que nous retrouverons souvent, spécifie le mode d'adressage et le ou les registres concernés :

  — Les trois bits du champ `reg` spécifient le registre dans le cas d'un transfert entre registre et case mémoire et le registre de destination dans le cas d'un transfert entre registres. La signification de ce champ est déterminée par le tableau, déjà rencontré, de la figure 12.1.

  — les deux bits du champ `mod` et les trois bits du champ `r/m` spécifient le mode d'adressage :

    — Si `mod = 11`, il s'agit d'un transfert entre registres généraux : le champ `r/m` spécifie le second registre suivant le tableau de la figure 12.1.

    — Si `mod = 00`, on a un transfert entre registre et case mémoire (rappelons que le sens est déterminé par le bit `d` du premier octet), l'**adresse effective** AE de la case mémoire étant spécifiée par `r/m` suivant le tableau de la figure 13.1. Seul le cas `r/m = 110` nous intéressera pour l'instant. Dans ce cas, le déplacement (sur 16 bits) doit être spécifié par deux octets supplémentaires. Nous étudierons les autres cas plus tard ; contentons-nous de dire que, par exemple, `[BX]` signifie le contenu du registre `BX`.

r/m	AE
000	[BX] + [SI]
001	[BX] + [DI]
010	[BP] + [SI]
011	[BP] + [DI]
100	[SI]
101	[DI]
110	déplacement
111	[BX]

FIGURE 13.1 – Modes d'adressage du 8088 lorsque `mod = 00`

— Les modes `mod = 01` et `mod = 10` ne nous intéresseront pas pour l'instant : le décalage est la somme de contenus de registres et d'un déplacement spécifié sur un ou deux octets.

Exercice corrigé 1.- *Traduire l'instruction* :

```
MOV BX, AX
```

en langage machine.

D'après ce que nous venons de dire, cette instruction se code par les deux octets représentés en binaire par :

```
| 1000 1011 | 11 011 000 |
```

soit 8B D8h.

Vocabulaire.- Il est traditionnel de parler d'**adressage implicite** ou d'**adressage de registre** dans le cas de copie entre registres. Le nom d'adressage *implicite* provient de ce que les opérandes (les deux registres concernés) ne sont pas indiqués par un octet distinct : c'est le code d'opération qui contient (qui implique) les noms des registres concernés.

Exercice corrigé 2.- *Traduire l'instruction* :

```
MOV BX, [10h]
```

en langage machine.

D'après ce que nous venons de dire, cette instruction se code par les quatre octets représentés en binaire :

```
| 1000 1011 | 00 011 110 | 0001 000 | 0000 0000 |
```

soit 8B 1D 10 00h.

13.2.5 Choix du segment

Nous venons de voir comment transférer le contenu de l'accumulateur en mémoire. On spécifie le décalage et, par défaut, le segment est le segment DS des données. Il s'agit effectivement du segment le plus approprié. Si ce n'est pas ce que l'on désire, il faut l'indiquer explicitement.

Langage symbolique.- L'instruction générale de transfert de l'accumulateur à un élément de mémoire vive s'écrit :

```
MOV SEG:[decalage], A
```

où `SEG` est l'un des registres de segment `CS` ou `SS`, `A` désigne l'un des registres `AL` ou `AX` et `decalage` le décalage de l'adresse, qui est un entier de seize bits.

Langage machine.- Dans ce cas, il faut faire précéder l'instruction par un octet spécifiant le segment désiré, appelé **préfixe de changement de segment** (*override prefix* en anglais) :

```
001 sreg 110
```

où les deux bits du champ `sreg` désignent le registre de segment conformément au tableau, déjà rencontré, de la figure 12.2.

Exercice corrigé.- *Traduire l'instruction* :

```
MOV CS:[A900h], AX
```

en langage machine.

Il suffit d'ajouter le préfixe `001 01 110` à l'instruction que nous venons de voir, ce qui donne 2E A3 00 A9h.

13.3 Historique

Stephen Morse regrette dans [Edw-08] que, lors de la conception du jeu d'instructions du 8086, il n'ait pas pu changer le fait de placer l'octet de poids faible avant l'octet de poids fort, par compatibilité avec le 8080. Il explique l'origine de la technique petitboutienne du microprocesseur 8008 et de ses descendants : un des buts du 8008 est de mimer le comportement d'un processeur conçu par *Datapoint* dont les bits arrivent en série, et non en parallèle. Dans un tel processeur, on doit voir les bits de poids faible avant les bits de poids fort de façon à traiter efficacement les problèmes de retenue.

13.4 Bibliographie

[Edw-08] Benj Edwards, *Stephen Morse : Father of the 8086 Processor*, **PCWorld**, Jun 17, 2008 7 :00 am. Disponible en ligne :

`http://www.pcworld.com/article/146917/stephen_morse_father_of_the_8086_processor.html`