Chapitre 6
Adressage des opérandes

Bien que nous ayons parlé d'adressage des opérandes et que nous ayons utilisé des opérandes dans ce qui précède, la question de l'adressage et des opérandes doit être examinée d'un point de vue plus global. C'est ce que nous proposons de faire ici. Notons qu'en langage de programmation évolué cette question n'est pas considérée, car le compilateur s'en occupe automatiquement. Nous avons déjà mentionné l'adressage des opérandes au chapitre 3 (section 3.7). Nous y avons en particulier fait la différence entre l'adresse relative et l'adresse absolue d'un opérande.

Rappelons ici que l'assembleur attribue aux instructions et aux données du programme des *adresses relatives*; ces adresses sont ainsi appelées car elles sont relatives au début du programme, l'adresse attribuée par l'assembleur au premier élément (instruction ou donnée) du programme étant égale à zéro et les adresses suivantes étant attribuées de façon séquentielle. Ainsi, si nous assemblons successivement trois programmes (ou sous-programmes) différents qui doivent fonctionner ensemble, chaque programme sera assemblé à partir de l'adresse zéro. Lorsqu'on veut charger les trois programmes en mémoire, en supposant même que la mémoire soit libre à partir de l'adresse zéro, on se rend compte qu'au moins deux des programmes ne pourront être chargés à partir de l'adresse 0. Les adresses réelles que leurs instructions et leurs données occuperont sont appelées leurs *adresses absolues*. Donc, pour au moins deux de nos programmes, les adresses absolues seront différentes des adresses relatives. En général, ceci sera vrai pour tout programme chargé en mémoire, car les premières adresses de la mémoire centrale sont toujours occupées par une partie du système d'exploitation.

Le travail du chargeur sera plus ou moins complexe selon l'ordinateur et le système d'exploitation considéré. Nous avons vu en effet que de nombreux ordinateurs sont tels que les adresses des opérandes sont situées dans les instructions elles-mêmes. On voit donc qu'au moment du chargement toutes les adresses relatives apparaissant dans des instructions doivent être modifiées par addition d'une valeur fixe qui est l'adresse de chargement du début du programme. Par contre, pour les anciens ordinateurs Macintosh, basés sur le processeur Motorola MC68000, le système d'exploitation utilisait un système d'adressage différent; les instructions ne comprenaient que les adresses relatives des opérandes et ce n'était qu'au moment de l'exécution d'une instruction que l'adresse relative de son opérande était transformée en adresse absolue.

Quels que soient les ordinateurs utilisés, les programmes objets produits par les assembleurs sont faits pour fonctionner une fois chargés à n'importe quel endroit de la mémoire: on dit que ce sont des programmes *translatables*.

L'adresse mémoire effective est l'adresse qui est effectivement utilisée au moment de l'exécution de l'instruction. Quelquefois cette adresse est la même que celle qui apparaît dans l'instruction, mais ce n'est généralement pas le cas. Cette adresse peut être modifiée de plusieurs façons au moment de l'exécution selon les divers modes d'adressage utilisés qui sont présentés ci-dessous.

6.1 Adressage immédiat

Dans ce mode d'adressage un opérande se trouve dans l'instruction elle-même. On obtient l'opérande sans avoir à faire un accès supplémentaire à la mémoire puisque ce dernier fait partie intégrante de l'instruction, d'où le qualificatif *immédiat*. Les instructions ci-dessous utilisent ce mode d'adressage :

```
LDA     456,i    ; placer la valeur décimale 456 dans le registre A
CPA     '*',i    ; comparer A au code ASCII d'un astérisque
```

où la valeur à déplacer ou à utiliser fait partie intégrante de l'instruction, comme le montre la figure 6.1.

Figure 6.1 Instruction avec adressage immédiat

6.2 Adressage direct

L'adressage direct est sans doute le mode d'adressage le plus souvent utilisé et est très simple. Dans ce mode, le champ opérande de l'instruction contient l'adresse effective de l'opérande et on doit faire un accès à la mémoire pour obtenir ce dernier. C'est le mode d'adressage le plus répandu; il n'est cependant et à toute fin pratique pas utilisé avec des adresses absolues, sauf dans des cas très particuliers où l'on doit nécessairement utiliser des adresses absolues, comme par exemple:

```
LDA  0x2340,d ; prendre le contenu des octets à l'adresse hexa 2340
```

instruction illustrée par la figure 6.2. Les adresses absolues utilisées doivent être inférieures à l'adresse maximum possible pour l'ordinateur utilisé et on doit savoir ce qu'elles signifient! Malheur à vous, si l'adresse n'est pas la bonne! En fait, l'utilisation normale est plutôt semblable à ce qui suit, où le symbole est remplacé dans l'instruction machine par son adresse :

```
LDA  val,d     ; prendre la valeur de la variable val
```

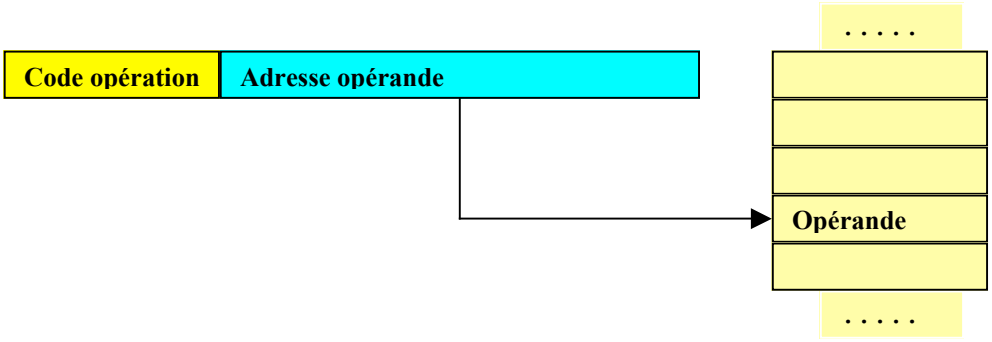

Figure 6.2 Instruction avec adressage direct

 ©2009 Ph. Gabrini

6.3 Adressage basé

Le mode d'adressage basé est un mode d'adressage relatif: dans les instructions les opérandes sont repérés par des adresses relatives. Ces dernières sont utilisées au moment de l'exécution *avec* le contenu d'un registre de base pour déterminer les adresses effectives des opérandes.

L'adresse effective de l'opérande est calculée au moment de l'exécution de l'instruction par addition du contenu du registre de base à l'adresse relative de l'opérande située dans l'instruction (souvent appelée *déplacement*):

adresse effective = adresse de base + déplacement

Le calcul de l'adresse effective est fait dynamiquement à l'exécution de chaque instruction ayant un opérande mémoire. Il importe alors que le registre de base contienne bien l'adresse de base correspondant à la zone de données du programme.

Selon les ordinateurs on notera qu'un registre de base peut devoir être nécessairement un registre adresse, ou comme sur PEP 8 il n'y a pas de registre de base et où l'adressage basé est en fait similaire à l'adressage indexé (voir plus bas). La figure 6.3 illustre le calcul de l'adresse effective d'un opérande repéré par la quantité hexadécimale 01AC sur un ordinateur Macintosh (version 68000). Selon les conventions de ce Macintosh le registre de base est le registre d'adresse A5, s'il contient la valeur C6454, l'adresse effective est alors:

C6454 + 01AC = C6600

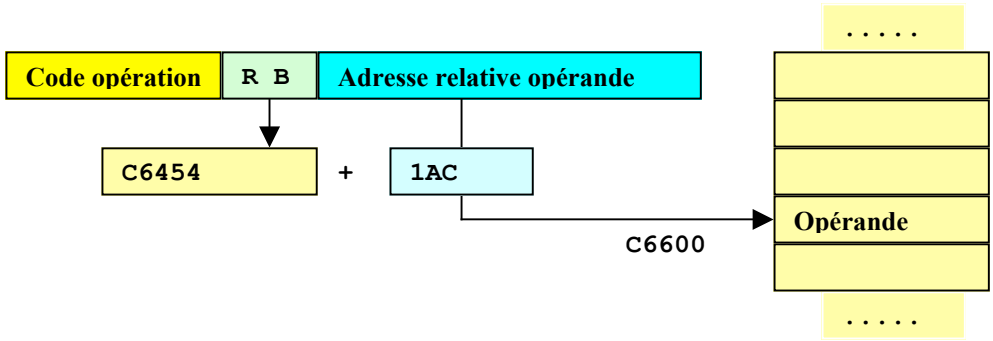

Figure 6.3 Calcul de l'adresse effective d'un opérande avec registre de base

Le mode d'adressage basé présente, par rapport aux autres modes d'adressage, des avantages et des inconvénients. L'inconvénient principal est que ce mode d'adressage est plus difficile à apprendre; un second inconvénient est que le contenu des registres de base est essentiel au bon fonctionnement du programme et que toute modification intempestive de ces registres conduit à la catastrophe. Parmi les avantages, notons le fait que les programmes écrits dans ce mode sont automatiquement *translatables*, on peut les transférer d'un endroit de la mémoire à un autre sans pour autant modifier les adresses des opérandes, seuls les contenus des registres de base devront être modifiés. On notera aussi comme avantage le fait que le couple base-déplacement peut être défini avec un nombre de bits inférieur à une adresse complète. Enfin si l'on doit effectuer des modifications d'adresse, ceci se fait facilement par modification du registre de base correspondant.

6.4 Adressage indexé

Ce mode d'adressage est semblable au mode d'adressage basé. Le calcul de l'adresse effective de l'opérande d'une instruction consiste encore à ajouter à l'adresse de l'opérande le contenu d'un registre d'index. Dans ce cas le calcul de l'adresse effective peut alors être résumé par:

adresse effective = adresse de base + contenu du registre d'index.

L'adresse de base est située dans l'instruction. Contrairement au registre de base dont la valeur reste généralement constante au cours de l'exécution d'un programme, le registre d'index prend des valeurs diverses au cours de l'exécution. On s'en sert en fait surtout comme d'un indice pour accéder aux valeurs d'une table.

La figure 6.4 illustre le calcul de l'adresse effective de l'opérande pour l'instruction PEP 8:

```
LDA  Vecteur,x    ; prendre Vecteur[x]
```

qui charge dans A le contenu d'un opérande défini avec indexation par le registre X que l'on supposera contenir la valeur décimale 22A (obtenue par `LDX  0x22A,i`) et par l'adresse de base `Vecteur` (3246) L'adresse effective de l'opérande sera donc de:

```
3246 + 22A = 3470
```

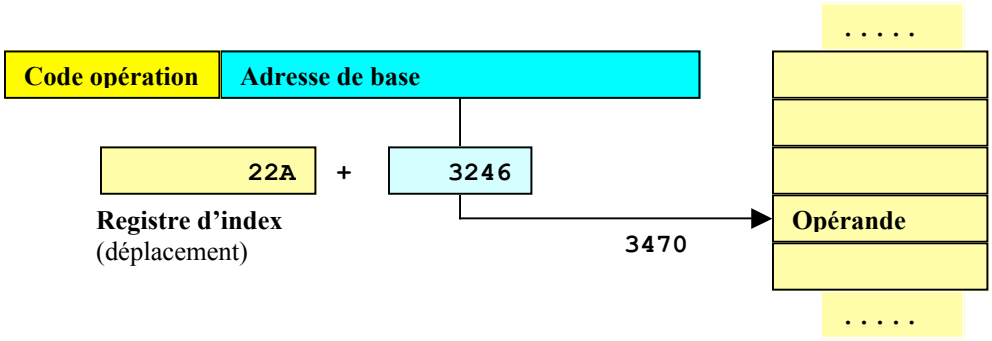

Figure 6.4 Calcul de l'adresse effective de l'opérande

6.5 Adressage indirect

Dans ce mode d'adressage, le champ adresse de l'instruction ne contient plus l'adresse de l'opérande, mais l'adresse d'une position mémoire qui, elle, contient l'adresse de l'opérande. Certains ordinateurs possèdent un mécanisme d'indirection, c'est le cas de PEP 8; d'autres n'en possèdent pas, encore qu'une indirection soit possible par l'intermédiaire des registres. Dans le cas des ordinateurs qui permettent l'indirection, certains sont tels que chaque cellule mémoire possède un indicateur d'indirection, ce qui rend possible les indirections à plusieurs niveaux.

Les indirections à plusieurs niveaux peuvent être comparées à ce qui arrive lorsque vous arrivez dans une grosse compagnie et que vous cherchez à rencontrer Monsieur Aristide; vous vous adressez à la réception qui vous envoie au bureau de Monsieur Xénophon, le bureau de Monsieur Xénophon vous renvoie au bureau de Monsieur Yves, le bureau de Monsieur Yves vous renvoie au bureau de Monsieur Zacharie où vous trouvez Monsieur Aristide.

La figure 6.5 illustre d'une part l'indirection simple et d'autre part une indirection à deux niveaux.

 ©2009 Ph. Gabrini

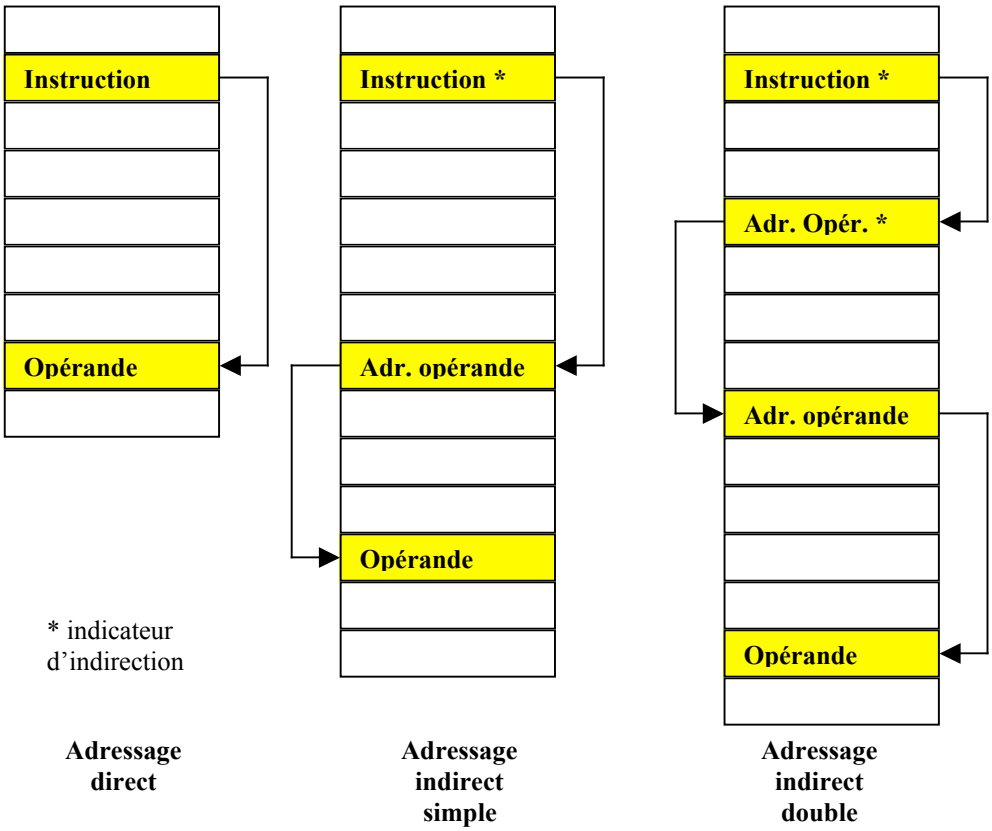

Figure 6.5 Adressage direct et indirect

La figure 6.6 représente l'adressage indirect simple de Pep 8. L'indicateur d'indirection dans une instruction est la lettre n, comme par exemple :

```
LDA     pointeur,n      ; pointeur contient l'adresse de l'opérande
```

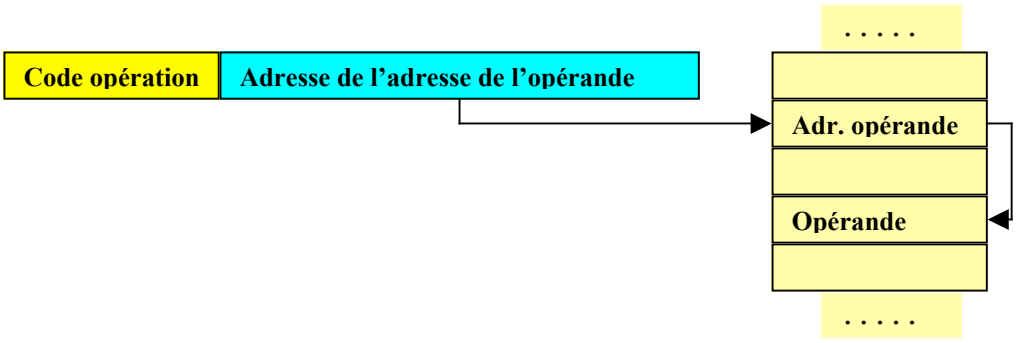

Figure 6.6 Adressage indirect

Supposez que vous disposiez d'une variable pointeur qui comprenne l'adresse d'une variable dynamique entière obtenue par un appel à l'allocateur `new`. En C++ vous auriez écrit les instructions suivantes pour déclarer le pointeur, obtenir l'espace mémoire pour la variable dynamique et accéder à la valeur de la variable ainsi repérée.

```
int *pointInt = new int;
.......
*pointInt = 31415;
```

Pour faire la même chose en assembleur, vous devez disposer d'un sous-programme **new** (voir le chapitre 9) et utiliser l'indirection pour atteindre la variable dynamique repérée par le pointeur.

```
pointInt:  .BLOCK  2       ; variable pointeur
      .......
      LDA   taille,i       ;    [taille mémoire requise]
      STA   -2,s           ;
      LDA   pointInt,i     ;    [adresse du pointeur]
      STA   -4,s           ;
      SUBSP 4,i            ;
      CALL  new            ;  place adresse dans pointInt
      .......
      LDA   31415,i        ;  valeur à affecter
      STA   pointInt,n     ; *pointInt = 31415
```

6.6 Adressage sur la pile

Il existe une structure de mémoire spéciale utilisée par les divers logiciels fonctionnant sur un processeur donné. Il s'agit de la **pile** : c'est un bloc de mémoire alloué spécifiquement par le système d'exploitation et utilisé par les logiciels. Au niveau de la programmation en assembleur, la pile est utilisée essentiellement pour gérer les appels à des sous-programmes et aussi pour l'allocation de mémoire aux sous-programmes. Ceci sera vu en détail au chapitre 9 qui présente les sous-programmes, car, en l'absence de sous-programmes, il n'y a aucune nécessité d'utiliser la pile.

6.7 Exercices

6.7.1 Si le registre A5 d'un Macintosh contient la valeur hexadécimale 201AE, calculer le déplacement nécessaire pour composer l'adresse relative du mot d'adresse absolue hexadécimale 20592 en utilisant le registre A5 comme registre de base.

6.7.2 Si la variable Vecteur est située à l'adresse 125A et si le registre X contient la valeur hexadécimale 123, quelle sera l'adresse effective de l'opérande de l'instruction suivante?

```
LDA      Vecteur,x
```

6.7.3 L'adressage indirect multiple n'existe pas dans les ordinateurs PEP 8; on peut cependant le réaliser en utilisant le registre X. En supposant que la variable Point contienne l'adresse de l'opérande (premier niveau) de la figure 6.5 (adressage indirect double), donner les instructions permettant de placer l'opérande dans le registre A.

 ©2009 Ph. Gabrini