**3ième partie (Architecture : le langage machine)**

(Utilise l’ouvrage ISN en terminale S de Gilles Dowek chez Eyrolles)



La mémoire est composée de plusieurs milliards de circuits mémoires un bit. Ces circuits sont organisés en agrégats de huit, seize, trente deux, soixante quatre bits, et parfois davantage, que l'on appelle des *cases mémoires*, et qui peuvent donc mémoriser des mots de huit, seize, trente deux, soixante quatre bits, *etc*. Le nombre de ces cases définit la taille de la mémoire de l'ordinateur. En général on indique la taille de la mémoire en indiquant le nombre d'octets c'est-à-dire de mots de huit bits qui peuvent être mémorisés. Ainsi une mémoire de 4 gigaoctets (binaires), contient 4 ⨯ 230 ⨯ 8 = 34 359 738 368 circuits mémoire un bit. Si la mémoire est organisée en mots de soixante-quatre bits, ces circuits sont répartis en 536 870 912 cases permettant de mémoriser un mot chacune. Comme il faut distinguer ces cases les unes des autres, on donne à chacune un numéro : son *adresse*. La mémoire contient les *données* sur lesquelles on calcule et le *programme* qui décrit le calcul effectué, donné sous la forme d'un ensemble d'*instructions*.

Le processeur de son côté n'a qu'un très petit nombre de cases mémoires que l'on appelle des *registres*, on peut imaginer, par exemple, qu'il ne contient que deux registres, que l'on appelle *A* et *B*. Les registres peuvent contenir des données, mais aussi des adresses de cases mémoire. Lorsque l'on parle de *processeurs 32 bits* ou *64 bits*, on fait référence à la taille de ces registres.

Pour échanger des données avec la mémoire, le processeur utilise deux procédés qui permettent l'un de transférer l'état d'un registre dans une case mémoire et l'autre de transférer l'état d'une case mémoire dans un registre.

Pour transférer le contenu du registre *A* dans la case mémoire d'adresse *n*, le processeur met les différents fils qui composent le bus d'adresses dans un état qui correspond à l'expression en base deux du nombre *n* et il met les différents fils qui composent le bus de données dans un état qui correspond au contenu du registre. Au signal d'horloge, chaque case de la mémoire compare son propre numéro au numéro arrivé sur le bus d'adresse, seule la case numéro *n* se reconnaît, elle met alors ses différentes entrées *S* (voir le chapitre XXX) dans l'état 1, de manière à enregistrer le mot arrivant le sur le bus de données. Le procédé symétrique permet au processeur de récupérer une valeur précédemment enregistrée : les informations circulent toujours du processeur vers la mémoire sur le bus d'adresses, mais elles circulent dans l'autre sens sur le bus de données, c'est la case *n* qui connecte sa sortie au bus de données, et c'est le registre qui met ses entrées *S* à 1 de manière à enregistrer le mot qui arrive sur le bus de données.

Ces deux opérations qui s'appellent le stockage (STA) et le chargement (LDA) du contenu d'une case mémoire dans le registre *A* (ST pour *STore*, LD pour *LoaD*). Il y a bien entendu des opérations similaires pour le registre *B* (STB et LDB).

Une autre opération que peut exécuter le processeur est l'addition du contenu du registre *A* et du contenu du registre *B.* Et le résultat de l'opération peut être stocké dans le registre *A* (ADD *A*) ou dans le registre *B* (ADD *B*). De même, DEC *A* *décrémente* la valeur contenue dans le registre *A,* c'est-à-dire soustrait 1 à la valeur contenue dans le registre *A* et stocke la valeur ainsi obtenue dans le registre *A* et DEC *B* réalise le même calcul sur la valeur contenue dans le registre *B.*

Si, par exemple, le processeur effectue successivement les opérations

LDA 7

LDB 8

ADD *A*

LDB 9

ADD *A*

LDB 10

ADD *A*

STA 11

et que, dans l'état initial la case 7 de la mémoire contient le nombre 42, la case 8 le nombre 68, la case 9 le nombre 47 et la case 10 le nombre 33, l'exécution des huit opérations ci-dessus a comme effet de

* LDA 7 : charger le contenu de la case 7, soit 42, dans le registre *A*,
* LDB 8 : charger le contenu de la case 8, soit 68, dans le registre *B*,
* ADD *A*: ajouter le contenu des registres *A* et *B* et mettre le résultat, 110, dans le registre *A*,
* LDB 9 : charger le contenu de la case 9, soit 47, dans le registre *B*,
* ADD *A*: ajouter le contenu des registres *A* et *B* et mettre le résultat, 157, dans le registre *A*,
* LDB 10 : charger le contenu de la case 9, soit 33, dans le registre *B*,
* ADD *A* : ajouter le contenu des registres *A* et *B* et mettre le résultat, 190, dans le registre *A*,
* STA 11 : stocker le contenu du registre *A*, soit 190, dans la case 11.

Au bout du compte, cette séquence d'opérations ajoute les quatre nombres présents dans les cases 7, 8, 9 et 10 de la mémoire et stocke le résultat dans la case 11.

Le langage machine

Un ordinateur doit être capable d'exécuter un programme. Il faut donc un moyen d'indiquer au processeur la séquence des opérations qu'il doit exécuter. Par exemple, la séquence LDA 7, LDB 8, ADD A, LDB 9, ADD *A*, LDB 10, ADD *A*, STA 11. Dans les premières machines, des cartes perforées ou un ruban perforé situé à l'extérieur de la machine indiquaient les opérations à effectuer, comme les cartes d'un orgue de barbarie indiquent les notes à jouer l'une après l'autre. Puis cette idée a été abandonnée au profit d'une autre : celle d'enregistrer le programme dans la mémoire avec les données. Ainsi on peut exprimer le programme ci-dessus en binaire en décidant par exemple que l'instruction LDA s'écrit 0, l'instruction LDB s'écrit 1, l'instruction STA s'écrit 2, l'instruction STB s'écrit 3 et l'instruction ADD s'écrit 4, DEC s'écrit 5. *A* s'écrit 0 et *B* s'écrit 1. Le programme ci-dessus s'écrit alors 0, 7, 1, 8, 4, 0, 1, 9, 4, 0, 1, 10, 4, 0, 2, 11, ce qui commence à devenir assez difficile à lire, même s'il est facile de passer d'une représentation à l'autre. On peut ensuite stocker ce programme dans la mémoire, en commençant, par exemple, à la case 100

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| adresse | .. | .. | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 | 112 | 113 | 114 | 115 | .. | .. |
| valeur |  | .. | 0 | 7 | 1 | 8 | 4 | 0 | 1 | 9 | 4 | 0 | 1 | 10 | 4 | 0 | 2 | 11 | .. | .. |

il suffit alors maintenant d'ajouter au processeur un nouveau registre, le *compteur de programme*, *PC*, qui débute à 100, et à chaque étape le processeur

* charge le contenu des cases mémoires d'adresses *PC* et *PC* + 1,
* décode le premier de ces nombres en une instruction (0 devient LDA, 1 LDB, *etc.*),
* exécute l'instruction en question,
* et ajoute 2 au registre *PC*.

Pour que le compteur de programme fonctionne correctement, chaque instruction doit utiliser exactement deux cases mémoire : une pour son nom et une pour son argument. L'argument des instructions ADD et DEC est le nom d'un registre. En revanche comme les instructions de chargement ont déjà un argument : l'adresse mémoire où aller chercher la donnée à charger, elle ne peuvent pas en avoir un second pour indiquer le registre utilisé. C'est pour cela que l'on a deux instructions LDA et LDB et de même deux instructions STA et STB.

Enregistrer les programmes en mémoire permet de faire très simplement des boucles et des tests. On ajoute aux instructions ci-dessus une instruction JMP (*jump*) telle que JMP *n* charge simplement le nombre *n*, ou plutôt le nombre *n* – 2 qui sera augmenté de 2 immédiatement après l'exécution du JMP, dans le registre *PC* pour détourner le programme de sa route et le forcer à continuer son exécution à l'adresse *n*. De même l'instruction JMZ (*jump if zero*), qui effectue un saut si le contenu du registre *A* est 0, permet de faire des tests. On ajoute enfin l'instruction END, qui termine le programme. En langage machine, on suppose que JMP s'écrit 6. JMPZs'écrit 7. END s'écrit 8 avec un argument puisqu'il en faut un : 0.

Pour construire une boucle ou un test avec ces nouvelles instructions, il faut tout d'abord trouver une façon de traduire la condition du test ou la condition d'arrêt de la boucle par un test d'égalité à zéro. Par exemple, pour effectuer un test comme x == 2, on peut placer la valeur de x dans le registre *A*, exécuter deux fois l'instruction DEC *A*, et enfin tester si le registre *A* contient 0. Ensuite, on écrit les séquences d'instructions qui correspondent aux différentes branches du test ou au corps de la boucle, et on utilise JMPZ et JMP pour diriger l'exécution du programme dans l'une ou l'autre de ces séquences. Par exemple, un programme qui lit une valeur *x* dans la case mémoire d'adresse 11, puis recopie la case mémoire d'adresse 12 dans la case mémoire d'adresse 20 si *x* vaut 2, ou la case mémoire d'adresse 13 dans la case mémoire d'adresse 30 dans le cas contraire peut s'écrire :

|  |  |  |
| --- | --- | --- |
| 100 : | 0 | LDA 11 |
| 101 : | 11 |
| 102 : | 5 | DEC A |
| 103 : | 0 |
| 104 : | 5 | DEC A |
| 105 : | 0 |
| 106 : | 7 | JMPZ 114 |
| 107 : | 114 |
| 108 : | 1 | LDB 13 |
| 109 : | 13 |
| 110 : | 3 | STB 30 |
| 111 : | 30 |
| 112 : | 6 | JMP 118 |
| 113 : | 118 |
| 114 : | 1 | LDB 12 |
| 115 : | 12 |
| 116 : | 3 | STB 20 |
| 117 : | 20 |
| 118 : | 8 | END |
| 119 : | 0 |

Les instructions précédentes ne permettent que d'accéder à un nombre limité de cases mémoire, dont les adresses sont les constantes entières qui servent d'arguments aux instructions LDA, STA, LDB et STB. Le calcul sur des structures de données plus complexes, comme des tableaux, nécessite d'autres instructions pour accéder à une case mémoire dont l'adresse est elle même une donnée, en particulier le résultat d'un calcul. Les processeurs réels ont bien d'autres instructions encore : des opérations booléennes, des opérations sur les nombres entiers, des opérations sur les nombres flottants, *etc.*

**Savoir-faire.** Savoir dérouler l'exécution d'une séquence d'instructions.

**Méthode**. Le principe est de suivre, instruction par instruction, l'évolution du programme en observant leur effet sur les valeurs contenues dans les registres, y compris le compteur de programme *PC*, et les valeurs contenues dans la mémoire, un peu comme on le ferait pour l'état de l'exécution d'un programme écrit en Java.

**Exercice corrigé.** Écrire une séquence d'instructions qui multiplie par 5 le nombre rangé dans la case mémoire d'adresse 10 et stocker le résultat dans la case mémoire d'adresse 11.

Pour multiplier par 5, on fait 4 additions, en accumulant le résultat dans le registre *A*. On note *x* le nombre rangé à l'adresse 10.

|  |  |  |  |
| --- | --- | --- | --- |
|  | A | B |  |
| LDA 10 | *x* |  | Charger le nombre rangé à l'adresse mémoire 10 dans *A* |
| LDB 10 | *x* | *x* | Charger le nombre rangé à l'adresse mémoire 10 dans *B* |
| ADD A | *x*+*x* | *x* | Additionner *A* et *B*, résultat dans *A* |
| ADD A | *x*+*x*+*x* | *x* | Additionner *A* et *B*, résultat dans *A* |
| ADD A | *x*+*x*+*x*+*x* | *x* | Additionner *A* et *B*, résultat dans *A* |
| ADD A | *x*+*x*+*x*+*x*+*x* | *x* | Additionner *A* et *B*, résultat dans *A* |
| STA 11 | *x*+*x*+*x*+*x*+*x* | *x* | Stocker le nombre contenu dans *A* à l'adresse mémoire 11 |

La séquence LDA 10, LDB 10, ADD *A*, ADD *A*, ADD *A*, ADD *A*, STA 11, s'écrit en langage machine 0, 10, 1, 10, 4, 0, 4, 0, 4, 0, 4, 0, 2, 11. On la stocke par exemple à partir de l'adresse 100 de la mémoire.

État de la mémoire avant l'exécution du programme

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| adresse | .. | 10 | 11 | .. | .. | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 | 112 | 113 | .. |
| valeur |  | *x* |  | .. | .. | 0 | 10 | 1 | 10 | 4 | 0 | 4 | 0 | 4 | 0 | 4 | 0 | 2 | 11 | .. |

État de la mémoire après l'exécution du programme

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| adresse | .. | 10 | 11 | .. | .. | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 | 112 | 113 | .. |
| valeur |  | *x* | 5*x* | .. | .. | 0 | 10 | 1 | 10 | 4 | 0 | 4 | 0 | 4 | 0 | 4 | 0 | 2 | 11 | .. |

**Exercice.** Expliquer ce que fait le programme suivant, écrit en langage machine, en supposant que le nombre *x* contenu dans la case mémoire d'adresse 10 est strictement positif. Que se passe-t-il si

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| adresse | .. | 10 | 11 | .. | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 | 112 | 113 | 114 | 115 | .. |
| valeur | .. | *x* | *y* |  | 0 | 10 | 1 | 10 | 7 | 112 | 5 | 0 | 4 | 1 | 6 | 104 | 3 | 11 | 8 | 0 |  |

**Exercice.** Écrire un programme lit deux valeurs *x* et *y* contenues respectivement dans les cases mémoires 11 et 12, calcule la différence *y* – *x*, et range le résultat dans l'adresse 13. On suppose que ces deux valeurs sont des nombres entiers positifs.

Compléter ce programme pour qu'il range la valeur 0 à l'adresse 15 si *x* est égal à *y*, ou la valeur *x* sinon.