Module 5: Mémoire (suite)

1 Introduction

Nous avons vu dans la leçon précédente, les caractéristiques de la mémoire à savoir la localisation, l'adressage, la capacité et les méthodes d'accès. Nous avons également présenté comment les données sont stockées dans la mémoire. Nous avons fini la leçon précédente par présenter les performances de la mémoire.

Dans cette leçon, nous allons présenter les différents types de mémoires et la technologie utilisée dans la fabrication de ces mémoires. Nous allons aborder la programmation en mémoire et comment les espaces mémoires des programmes sont gérés au niveau de la mémoire.

2 Mémoires à semi-conducteurs

Les mémoires à accès rapide sont composées de cellules mémoire fondées sur un support semi-conducteur. On distingue donc plusieurs types de mémoires qui ont chacun leurs caractéristiques [1] :

RAM (Read Access Memory, Mémoire à accès aléatoire) :

Il s'agit de la mémoire vive (principale) de l'ordinateur comme l'illustre la figure 1.

1. Des parties de cette section sont tirées du livre de l'architecture de l'ordinateur d'Emmanuel Lazard.

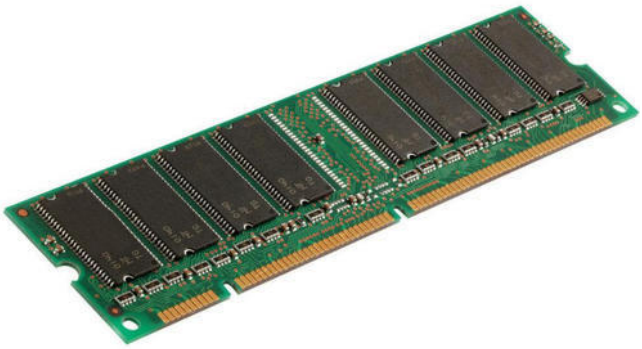

FIGURE 1 – Mémoire à accès aléatoire (image tirée de Google image).

Elle est composée de cellules mémoire standard volatiles. Elle sert à mémoriser toutes les instructions et les données nécessaires au bon fonctionnement des programmes. L'initialisation et les modifications de l'information se font grâce au courant électrique.

ROM (Read Only Memory, Mémoire à lecture seule) :

À l'inverse de la RAM, la ROM, appelée aussi mémoire morte, a un contenu figé lors de la fabrication du circuit intégré. La figure 2 présente un exemple de ROM.

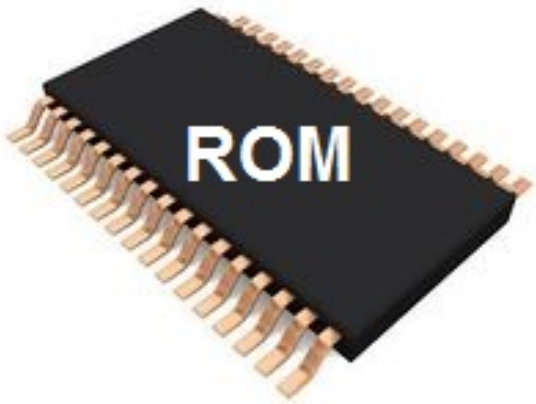

FIGURE 2 – Mémoire à lecture seule (image tirée de Google image).

PROM (Programmable ROM, Mémoire à lecture seule programmable) :

une PROM est dotée lors de sa fabrication d'une liaison dans chaque cellule. Cette liaison est en fait un fusible que l'utilisateur peut détruire. Une PROM est vendue vierge (elle a donc un coût plus faible puisque tous les circuits fabriqués sont identiques) et un appareillage électrique simple permet d'éliminer les fusibles dans certaines cases mémoire suivant le choix de l'utilisateur. La figure 3 présente un exemple de PROM.

FIGURE 3 – Mémoire à lecture seule programmable (image tirée de Google image).

La PROM pourrait être programmée une seule fois par l'utilisateur, ce qui est un processus irréversible.

EPROM (Erasable PROM, PROM effaçable) :

Un bit est maintenant stocké sous la forme de charges électriques localisées dans une cellule mémoire, isolées du reste du circuit. En raison de cette isolation, la mémoire est non volatile : les charges ne disparaissent pas lorsque l'alimentation est coupée. Mais on peut les enlever manuellement, ce qui est impossible sur les ROM et les PROM. La figure 4 présente un exemple de EPROM.

FIGURE 4 – PROM effaçable (image tirée de Google image).

EEPROM (Electrically Erasable PROM, PROM électriquement effaçable) :

La mémoire EEPROM permet un effacement électrique des informations : un certain type de courant permet l'écriture de l'information par stockage de charges tandis qu'un autre type les fait disparaître. À l'inverse de L'EPROM qui nécessite d'effectuer des manipulations pour effacer l'information et la reprogrammer : il faut extraire le boîtier de son support, l'exposer à un rayonnement ultraviolet, le placer sur l'appareil de programmation avant de le réinstaller. L'EEPROM n'a plus besoin d'être sortie de son logement pour subir les modifications souhaitées. La figure 5 présente un exemple de EEPROM.

FIGURE 5 – PROM électriquement effaçable (image tirée de Google image).

2.1 Technologie de la mémoire centrale

Nous avons mentionné précédemment que le temps de latence et la bande passante sont parmi les paramètres les plus importants pour la mesure de performance de la mémoire. Le temps de latence (qui affecte directement la mémoire cache) représente la principale préoccupation de la mémoire cache, tandis que la bande passante (qui affecte le processeur et les entrées/sorties) représente la principale préoccupation des multi-processeurs et les entrées/sorties.

Bien que les mémoires caches bénéficient grandement des mémoires à faible latence, il est quand même plus facile d'améliorer la bande passante de la mémoire avec des nouvelles organisations que de réduire le temps de latence.

Depuis 1975, pratiquement tous les ordinateurs utilisent les mémoires dynamiques, appelées DRAM pour la mémoire principale et les mémoires statiques, appelées SRAM pour les mémoires caches. Ces deux types de technologies sont définis comme suit :

2.1.1 DRAM

Les DRAM (Dynamic Read Access Memory) sont un type de mémoire vive compacte et moins cher comparativement aux mémoire SRAM (voir plus bas). Les mémoires DRAM utilisent un seul transistor pour stocker un bit. La lecture de ce bit entraîne la destruction de l'information, donc cette information doit être récupérée. Ceci est une des raisons pourquoi le temps de cycle de la mémoire DRAM est traditionnellement plus long que le temps d'accès. Récemment, les DRAM ont introduit plusieurs banques qui permettent au processus de réécriture d'être caché. De plus, pour éviter la perte d'information si un bit n'est pas lu ou écrit, le bit doit être rafraîchi. Le contrôleur de la mémoire inclut un matériel qui assure le rafraîchissement périodiques des DRAM.

La cellule des mémoires DRAM garde un bit sous forme de charges électriques dans un minuscule condensateur associé à un transistor de commande. Cette construction permet donc de réduire fortement la taille de la cellule (un transistor au lieu de quatre à six) et d'augmenter considérablement la capacité maximales des boîtiers mémoire. L'inconvénient de cette technologie est qu'une fuite de courant décharge le condensateur en quelques millisecondes, ce qui entraîne la perte de l'information.

Pour assurer un système balancé, il a été suggéré que la capacité de la mémoire devrait croître linéairement avec la vitesse du processeur. Par exemple, un processeur avec 1000 MIPS (mega instructions per second) devrait avoir une mémoire de capacité de 1000 Mo.

2.1.2 SRAM

Les SRAM (Static Read Access Memory) sont un type de mémoire largement utilisé dans la fabrication des mémoires caches. Ces mémoires sont par contre plus chers que les DRAM. À la différence des DRAM où les données nécessitent d'être réécrites après une lecture, les SRAM n'ont pas cette dynamique, d'où la différence dans le temps d'accès et le temps du cycle. Les SRAMs n'ont pas

besoin de rafraîchissement, par conséquent le temps d'accès est très proche au temps du cycle.

Aujourd'hui, toutes les mémoires caches sont intégrées dans le processeur. La plus grande capacité des mémoires caches de troisième niveau (on va voir ça dans le cours sur les mémoires caches) atteint 12 Mo, alors que la capacité des mémoires principales est généralement entre 4 Go et 16 Go pour les DRAM. Le temps d'accès des caches (SRAM) reste toujours plus rapide, trois à cinq fois plus rapide que les DRAM.

Les cellules de la mémoire SRAM sont construites comme une bascule élémentaire qui permet de mémoriser un seul bit de façon permanente tant que l'alimentation électrique est présente. L'avantage de cette technologie est que le temps d'accès est très faible car le bit est stocké de manière active dans la cellule. Cependant, l'inconvénient principale des SRAM est qu'elles requirent de 4 à 6 transistors par cellule pour construire la bascule, ce qui explique la capacité plus faible des mémoires SRAM comparativement aux mémoires DRAM.

Quelque soit le type de mémoire (statique ou dynamique), chaque cellule de mémorisation possède une ligne de donnée permettant de lire le bit stocké ou de l'écrire, une ligne de commande indiquant une lecture ou une écriture, une ligne de sélection permettant d'activer la cellule comme illustré dans la figure 6 [2].

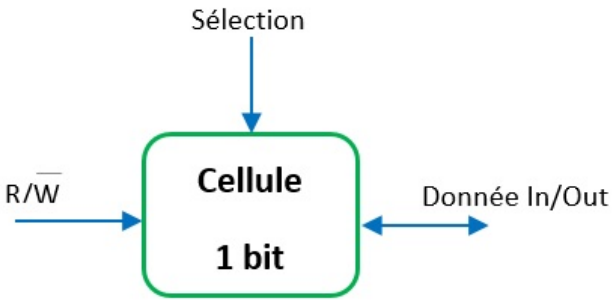

FIGURE 6 – Commande d'une cellule mémoire RAM.

Pour effectuer une lecture, on envoie le signal correspondant ($R/\overline{W} = 1$), on sélectionne la cellule (sélection = 1) et on récupère la donnée en sortie. Pour effectuer une écriture, on met le bit à mémoriser sur la ligne de donnée, on positionne le signal d'écriture ($R/\overline{W} = 0$) et on sélectionne la cellule.

3 Programmation en mémoire

Nous avons mentionné précédemment que la mémoire est utilisée par les programmes pour stocker les codes exécutables ainsi que leurs données. Chaque

2. Figure tirée du livre d'architecture de l'ordinateur d'Emmanuel Lazard.

langage a sa propre manière de stocker les informations pendant leur utilisation, mais les découpages de l'espace mémoire sont standard.

3.1 Espace mémoire d'un programme

Un programme a plusieurs choses à stocker à savoir le code exécutable, et les variables globales et locales. Rappelons que les variables globales peuvent exister pendant toute l'exécution d'un programme, alors que les variables locales n'existent que pendant l'appel d'une fonction.

Les variables globales sont stockées dans une zone mémoire qui s'appelle "tas" ou "heap", alors que les variables locales sont stockées dans une pile "stack".

Dans ce qui suit, nous allons voir des exemples de stockage de variables dans deux langages différents à savoir le langage C et Java [3].

3.1.1 Langage C

En plus des variables globales, on trouve aussi dans le tas les allocations mémoires crées à l'aide de la fonction **malloc()**. Cette zone évolue donc au fur et à mesure que le programme alloue et libère de la mémoire. À son tour, la pile contient les variables locales d'une fonction, ses paramètres ainsi que l'adresse de retour. Lors d'un appel de fonction, la fonction appelante empile les paramètres de l'appel et l'adresse de retour. De cette façon, le contrôle est donné à la fonction appelée, qui empile ses variables locales. À la fin de l'exécution de la fonction, les variables locales, paramètres et adresse de retour sont dépilées.

Prenons l'exemple du code C suivant :

```
1 void fonctionC(int i){
2 char *pointeur = malloc(100);
3 ..
4 }
```

Lors de l'appel de la fonction $fonctionC$, en plus de l'adresse de retour, 8 octets sont réservés sur la pile : 4 pour le paramètre i, et 4 pour le pointeur *pointeur*. Ce dernier contient l'adresse d'une zone de 100 octets réservée sur le tas.

Notez bien que la zone crée sur le tas ne pourra être détruite qu'avec un appel explicite de la fonction $free()$ qu'il ne faut surtout pas l'oublier pour éviter la saturation de la mémoire.

3.1.2 Langage Java

Pour le langage Java, le tas correspond à la zone mémoire où sont créés tous les objets avec leurs variables d'instance déclarés dans une classe et non dans une méthode. Considérons le code Java suivant :

```
1 public class ClasseJava{
2 int taille;
3
```

3. Exemples tirés du livre d'architecture de l'ordinateur.

```
4       public void fonctionJava(){
5           objet obj = new Objet(0);
6       }
7   }
```

À sa création, un objet de la classe *ClasseJava* est placé sur le tas avec sa variable d'instance *taille*. Lors de l'appel de la méthode $fonctionJava()$, la variable locale *obj* est empilée avec le code $fonctionJava$. *obj* est une référence, équivalent à un pointeur dans le langage C, à un objet de la classe *Objet* créé sur le tas.

À l'inverse du langage C, il n'existe pas en Java de destruction explicite des objets. Le mécanisme de ramasse-miettes (garbage collector) supprime régulièrement du tas ceux qui ne sont plus référencés par une variable.