

Architecture des ordinateurs

Chapitre 4 : Architecture matérielle - La Mémoire

* *

Clément MOREAU, Olivier PLOTON

{clement.moreau, olivier.ploton}@univ-tours.fr

Université de Tours ~ Département informatique de Blois

Licence 2 - Informatique

Sommaire

- 1 Généralités
- 2 Logique séquentielle
- 3 Banc à registres et RAM
- 4 Mémoire à code correcteur d'erreurs
- 5 Optimisations mémoire

Généralités

Nous sommes partis du constat que l'on pouvait conserver en mémoire des données et résultats.

Mais comment fait-on pour doter une machine d'une *mémoire* ?

On l'a vu jusqu'ici, il existe de multiples instances de stockage au sein d'un ordinateur :

- Registre,
- Cache,
- Mémoire principale / morte,
- etc.

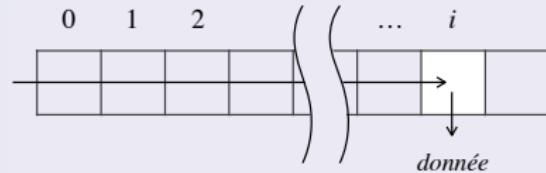
Généralités

Soit une mémoire constituée de n adresses, on souhaite accéder à la donnée de l'adresse i .

Définition (Accès séquentiel)

L'accès séquentiel est caractérisé par une lecture linéaire des adresses, dès lors, la complexité d'accès est en $O(n)$.

On retrouve ce mode d'accès dans les bandes magnétiques par exemple.

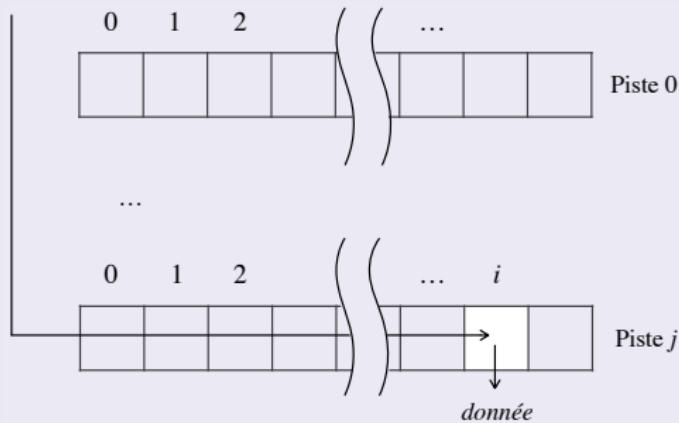


Généralités

Définition (Accès semi-direct)

L'accès semi-direct se caractérise par une décomposition de la mémoire en p pistes de $\frac{n}{p}$ cellules. Toute piste permet un accès direct, puis une lecture séquentielle est effectuée pour l'accès à la donnée. La complexité d'accès est alors en $O(n/p)$.

On retrouve ce mode d'accès dans les disques durs et les disques électroniques.



Généralités

- *Disque Dur ou HDD (Hard Drive Disk)*

- Fonctionnement mécanique : Ensemble de plateaux.
- Chaque plateau est constitué d'un disque en aluminium, verre ou céramique, recouvert d'une surface magnétique permettant le stockage des données et d'une tête de lecture/écriture.
- Pour écrire, la tête modifie la polarité du disque en sa surface en fonction du courant électrique qui la traverse.
- Pour lire, c'est la polarité du disque qui engendre aux bornes de la tête un potentiel électrique qui permet de lire un 1 ou un 0.



FIGURE 3 - Intérieur d'un boîtier de disque dur Seagate Medalist® avec trois plateaux.

Généralités

- *Disque électroniques ou SSD (Solid State Disk)*

- Fonctionnement électrique : Utilise la mémoire flash à semi-conducteurs qui possède les mêmes caractéristiques que la mémoire vive mais dont les données ne disparaissent pas lors de la mise hors tension.
- Une cellule de base est représentée à l'aide d'un transistor MOS, une mémoire est donc représentée par un ensemble de transistors.
- L'écriture et l'effacement des données s'effectuent par l'application de différentes tensions aux points d'entrée de la cellule.
- Possibilité d'erreurs de stockage dues aux pertes de tension ou décharges ⇒ gestion à l'aide de codes correcteurs.
- Gestion de la répartition par des algorithmes de *wear levelling* chargés de répartir les écritures de manière uniforme sur l'ensemble de la mémoire.



FIGURE 4 - Support SSD
avec plusieurs mémoires
flash Intel®

Généralités

Avantages

- HDD
 - Bon rapport coût/capacité de stockage
 - Fiabilité de stockage,
 - Capacité de ré-écriture importante.
- SSD
 - Rapidité d'accès,
 - Résistance au choc,
 - Faible consommation électrique,
 - Absence de fragmentation.

Inconvénients

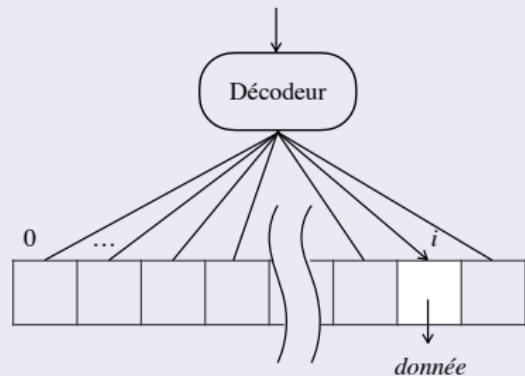
- HDD
 - Fragilité des disques,
 - Lenteur d'accès et latency importante,
 - Consommation électrique forte.
- SSD
 - Coût plus important,
 - Sensibilité aux corruptions logiques.

Définition (Accès direct (ou aléatoire))

L'accès direct (ou accès aléatoire) permet d'accéder à l'adresse de la donnée immédiatement via un décodeur d'adresse. À partir de la valeur numérique de l'adresse, le décodeur sélectionne la cellule mémoire correspondante.

Ainsi, la complexité d'accès est constante en $O(1)$ et le temps d'accès est principalement conditionné par le décodage et le temps de circulation de l'information dans les bus.

On utilise ce mode d'accès pour la mémoire principale (RAM ou Random Access Memory) et les caches.



Généralités

Qu'est-ce qu'une mémoire à semi-conducteurs ?

Pour faire simple, une mémoire à semi-conducteurs est une mémoire qui utilise des technologies issues de l'électricité et des matériaux semi-conducteurs.

On a vu que la mémoire Flash et la RAM possédaient des caractéristiques techniques proches car elles reposent toutes deux sur la technologie des *mémoires à semi-conducteurs*.

En fonction des usages, il existe différents types de mémoires électroniques, qui ont chacun leurs caractéristiques et leur place dans l'ordinateur.

- RAM (Random Access Memory) ou mémoire vive.
- ROM (Read Only Memory) ou mémoire morte.
- PROM (Programmable ROM).
- EPROM (Erasable PROM).
- EEPROM (Electrically EPROM). Dont fait partie les mémoires flash.

Généralités

| Type | Accès | Modifications | Initialisation | Volatile |
|----------------|-------------------|---------------|----------------|----------|
| RAM | R/W | Électrique | Électrique | Oui |
| ROM | R | Non | Masque | Non |
| PROM | R | Non | Électrique | Non |
| EPROM | R ⁺ /W | Ultraviolet | Électrique | Non |
| EEPROM (Flash) | R ⁺ /W | Électrique | Électrique | Non |

TABLEAU 1 - Différents types de mémoires à semi-conducteurs

Généralités

Si elles ne peuvent être modifiées, à quoi servent les ROM et PROM ?

À mémoriser des informations figées de façon permanente. Par exemple :

- Les microprogrammes souches de la machine (Décodeur / Séquenceur).
- Programme de démarrage pour de charger depuis un disque dur le reste du système d'exploitation.
- Plus généralement, toute instruction primordiale d'affichage ou d'E/S et dépendante du matériel est stockée de manière morte dans une partie de la mémoire principale, généralement une PROM.

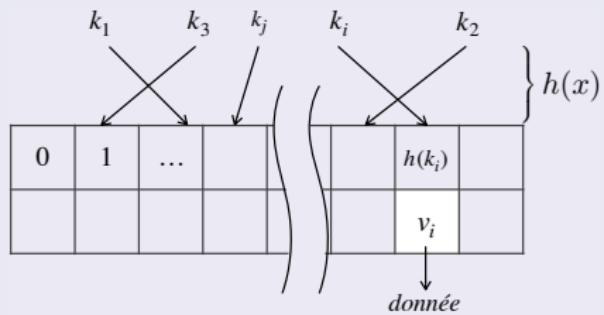
Généralités

Définition (Accès associatif)

Les mémoires à *accès associatif* enregistrent des couples type clé-valeur (k_i, v_i) souvent à partir d'une table de hachage. Cela permet de mémoriser des correspondances quelconques de valeurs. On définit alors une fonction de hachage h qui associe à la clé k_i une valeur d'indexe $h(k_i)$ appelée alvéole (ou bucket).

Plusieurs mécanismes de mémoire cache ou de mémoire virtuelle font un usage intensif de tableaux associatifs. La complexité d'accès est constante en $O(1)$.

Cependant, les fonctions de hachage ne sont pas injectives : $\exists (k, k'), k \neq k'$ et $h(k) = h(k')$, la complexité d'accès au pire des cas est alors en $O(n)$.



Généralités

Pour résumer :

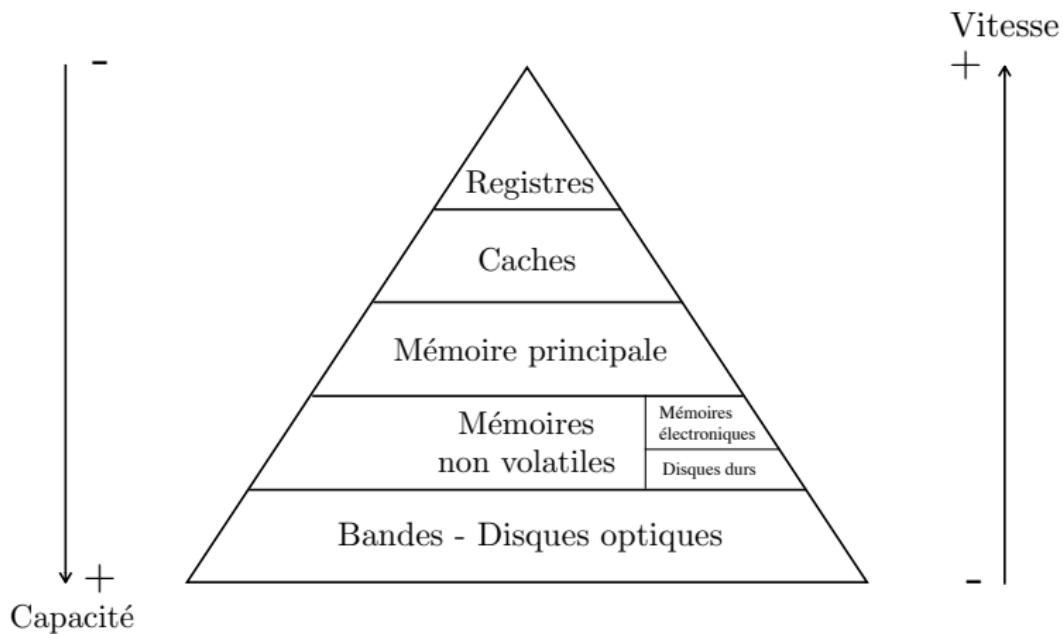
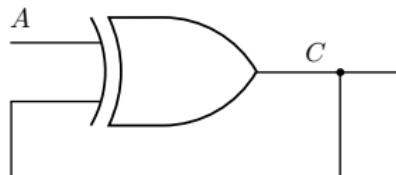


FIGURE 5 - Hiérarchie mémoire

Logique séquentielle

Comment implémenter une mémoire à l'aide de composant logique ?



| A | C avant | C après |
|-----|-----------|-----------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

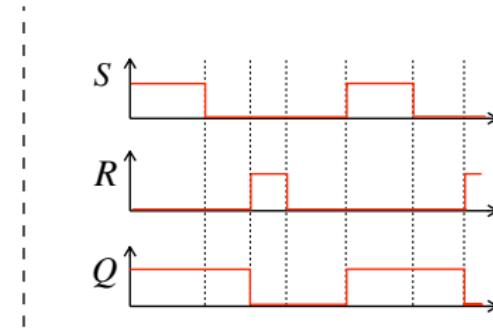
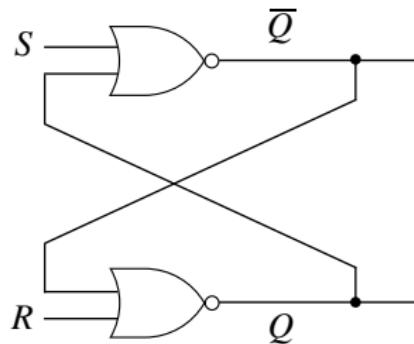
Mais si $A = 1$ et $C = 1$ alors le circuit n'arrête pas d'osciller !

⇒ Problème lié à la rétro-action.

Logique séquentielle

On a besoin d'un circuit qui puisse conserver l'information jusqu'à ce que nous le décidions de redémarrer.

Ce type de circuit existe et est appellé *Bascule RS Asynchrone*(Reset-Set).



L'entrée $(R, S) = (1, 1)$ est interdite sur les bascules RS !

Logique séquentielle

On a la table de vérité suivante :

| R | S | Q_{n+1} |
|-----|-----|-----------|
| 0 | 0 | Q_n |
| 0 | 1 | 1 |
| 1 | 0 | 0 |

| R/S | 00 | 01 | 11 | 10 |
|-------|----|----|-----------|----|
| Q_n | 0 | 1 | φ | 0 |
| | 1 | 1 | φ | 0 |

On a l'équation $Q_{n+1} = S \vee (\neg R \wedge Q_n)$

Bascule RS

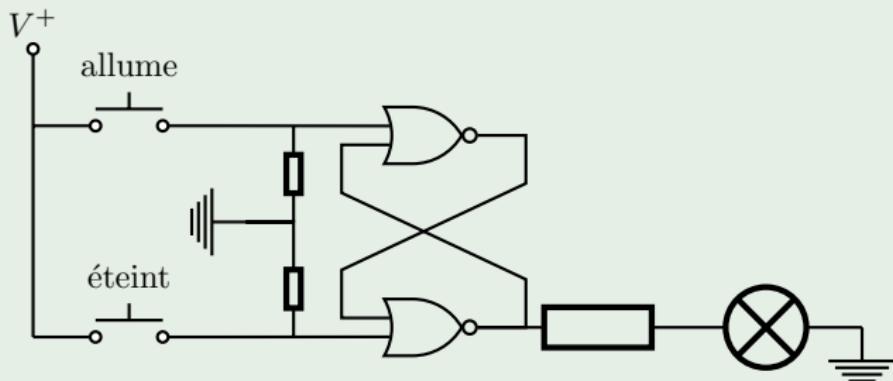
Un verrou RS ne s'utilise que pour les entrées $(R, S) \in \{(0, 0), (0, 1), (1, 0)\}$

- L'entrée $(R, S) = (0, 1)$ met à 1 la sortie du verrou $Q_{n+1} = 1$.
- L'entrée $(R, S) = (1, 0)$ met à 0 la sortie du verrou $Q_{n+1} = 0$.
- L'entrée $(R, S) = (0, 0)$ laisse la sortie du verrou dans son état. Il s'agit d'un état de mémorisation.

Logique séquentielle

Interrupteur

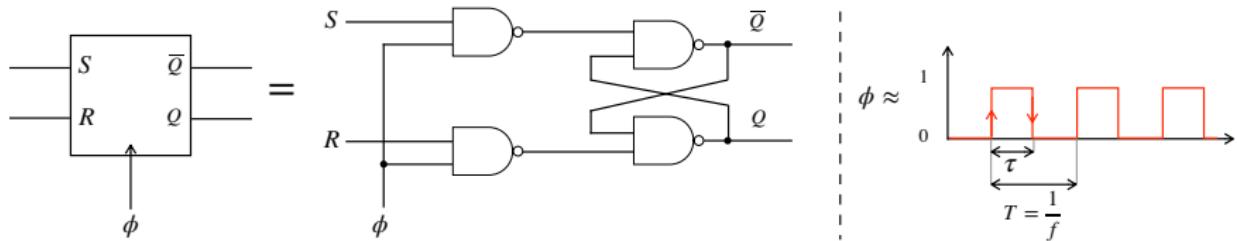
Avec une bascule R/S qu'on peut contrôler le flux électrique avec deux boutons “allumé” et “éteint”. On peut maintenir la lumière allumée même lorsque le bouton “allumé” est relâché. On a donc bien mémorisé l'état allumé.



Logique séquentielle

Le problème est le délai de stabilisation.

On va introduire une horloge ϕ afin d'espacer les délais de stabilisation et coordonner les signaux.

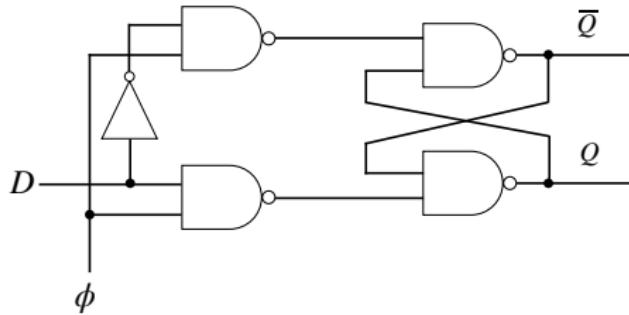


Ainsi, quelque soit le signal d'entrée des porte AND, celui-ci n'atteindra la bascule si et seulement si le signal ϕ est sur front montant à 1. On appelle cette structure une bascule RS *synchrone*.

Logique séquentielle

On peut avoir une unique entrée dans un verrou.

On a la table de vérité suivante :



| D | Q_{n+1} |
|---|-----------|
| 0 | 0 |
| 1 | 1 |

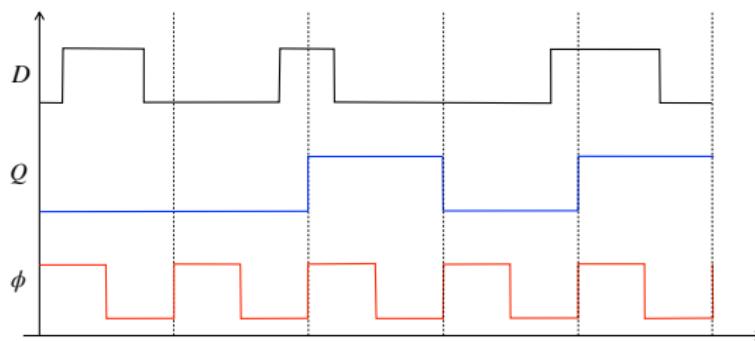
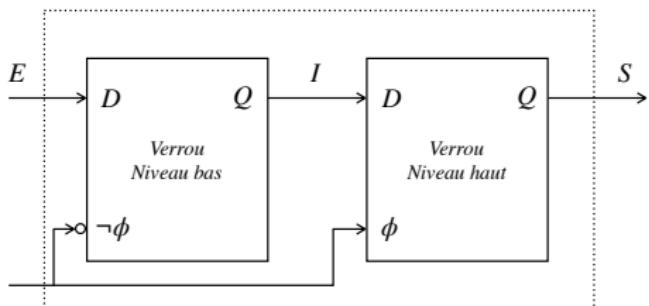
On a l'équation $Q_{n+1} = D$

Ce verrou stocke D lorsque ϕ est à son niveau haut. On dit qu'il s'agit d'un verrou régi par le *niveau haut de l'horloge*.

Ce type de bascule est appellé *Bascule D*.

Logique séquentielle

En plaçant en série un verrou régi par le niveau bas de l'horloge, puis un autre régi par le niveau haut, on obtient une bascule :



La bascule mémorise
l'entrée E sur le *front montant* de l'horloge ϕ .

Logique séquentielle

À quoi sert la logique séquentielle ?

⇒ À plein de choses ! Mais entre autres à fabriquer des registres !

Nos registres sont fabriqués à partir de bascules régie par un front d'horloge.
Cette convention fixe la manière d'utiliser les bascules :

- sur un cycle d'horloge on calcule un résultat r_i , qui est mémorisé sur l'entrée D de la bascule à la fin du cycle.
- au cours du cycle suivant :
 - le résultat du cycle précédent r_i est disponible et constant sur la sortie Q .
 - on calcule un nouveau résultat r_{i+1} mémorisé à la fin du cycle.

Logique séquentielle

Le registre le plus simple que l'on puisse imaginer consiste simplement à monter des bascules D en parallèle, en les connectant au même signal d'horloge.

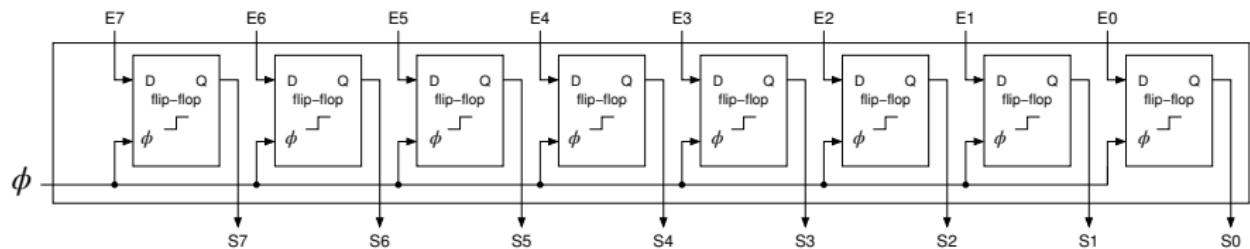


FIGURE - Circuit séquentiel d'un registre de 8 bits

Logique séquentielle

C'est quand même un peu compliqué la logique séquentielle...

Oui, c'est vrai que les circuits séquentiels sont plus complexes que les circuits combinatoires.

Exemple

On souhaite construire un circuit séquentielle d'un compteur 2 bits ($b_1 b_2$) modulo 4 qui s'incrémenté si son entrée $e = 1$ et qui conserve sa valeur si $e = 0$.

La construction d'un tel circuit sans préalable est plus ardue, c'est pourquoi nous va introduire le concept d'automate qui permet d'abstraire (et de simplifier) le comportement d'un circuit.

Logique séquentielle

Pour les circuit synchrones (i.e. en présence d'une horloge), on peut abstraire un circuit à l'aide d'un automate de Moore.

Définition

Un *automate de Moore* est un 6-uplet $M = (Q, \Sigma, \Gamma, \delta, \lambda, q_0)$ avec :

- Q , un ensemble fini d'états, $q_0 \in Q$ un état initial.
- Σ est l'alphabet d'entrée et Γ l'alphabet de sortie.
- $\delta : Q \times \Sigma \rightarrow Q$, une fonction de transition.
- $\lambda : Q \rightarrow \Gamma$, une fonction de sortie pour chaque état.

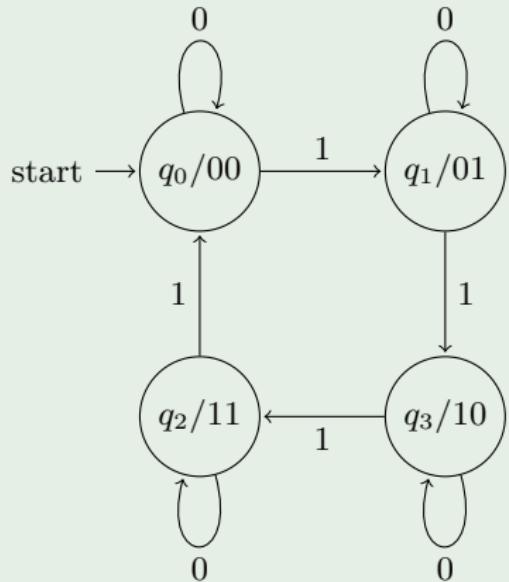
La sortie de M en réponse à une séquence d'entrée $a_1a_2...a_n \in \Sigma^n$ est $\lambda(q_0)\lambda(q_1)... \lambda(q_n)$ où $q_0, ..., q_n$ est la séquence des états tels que $\delta(q_{i-1}, a_i) = q_i$ pour $1 \leq i \leq n$.

Un automate de Moore retourne la sortie $\lambda(q_0)$ pour toute entrée ε .

Logique séquentielle

Exemple

On a donc $\Sigma = \{0, 1\}$ et $\Gamma = \{(00), (01), (10), (11)\}$.



| $b_1^{(n)}$ | $b_0^{(n)}$ | e | $b_1^{(n+1)}$ | $b_0^{(n+1)}$ |
|-------------|-------------|-----|---------------|---------------|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 |

On déduit que :

- $b_1^{(n+1)} = b_1^{(n)} \oplus (b_0^{(n)} \wedge e)$
- $b_0^{(n+1)} = b_0^{(n)} \oplus e$

Logique séquentielle

Un automate fini synchrone de Moore peut être implanté en matériel de la façon suivante :

Un registre à n bits stocke l'état courant de l'automate et k signaux codant l'entrée courante e .

La fonction combinatoire Φ déduite de l'automate de la fonction de transition δ et la fonction de sortie λ .

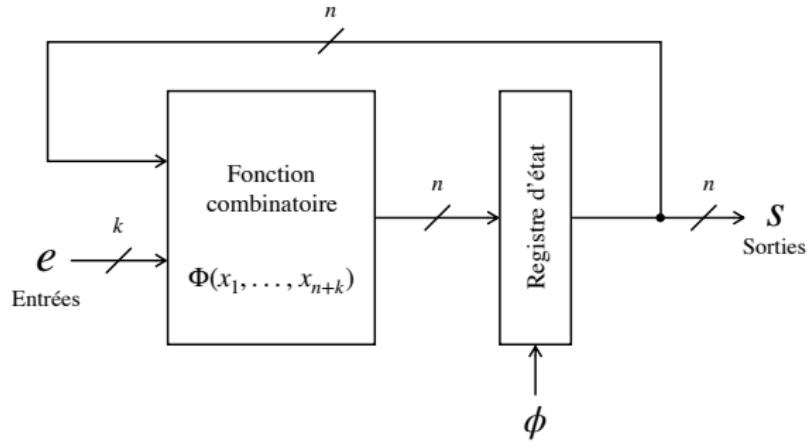


FIGURE - De l'automate au circuit séquentiel

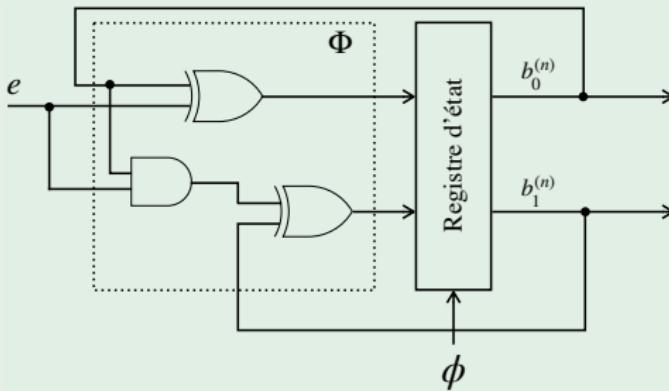
Logique séquentielle

Exemple

Pour revenir au compteur binaire...

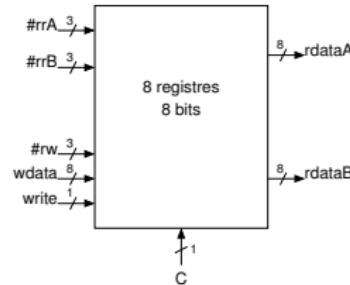
Si on note :

- $\Phi_1(b_1^{(n)}, b_0^{(n)}, e) = b_1^{(n+1)}$
- $\Phi_0(b_1^{(n)}, b_0^{(n)}, e) = b_0^{(n+1)}$



Banc à registres et RAM

Voici par exemple un banc de 8 registres 8 bits, avec deux ports de lecture :



Pour implanter un tel banc de registres, on peut utiliser :

- 8 registres 8 bits ;
- 2 multiplexeurs 64 vers 8 pour sélectionner parmi les registres :
 - rdataA en fonction de #rrA,
 - rdataB en fonction de #rrB.
- un décodeur 3 vers 8 pour adresser le registre à écrire d'après #rw.

Banc à registres et RAM

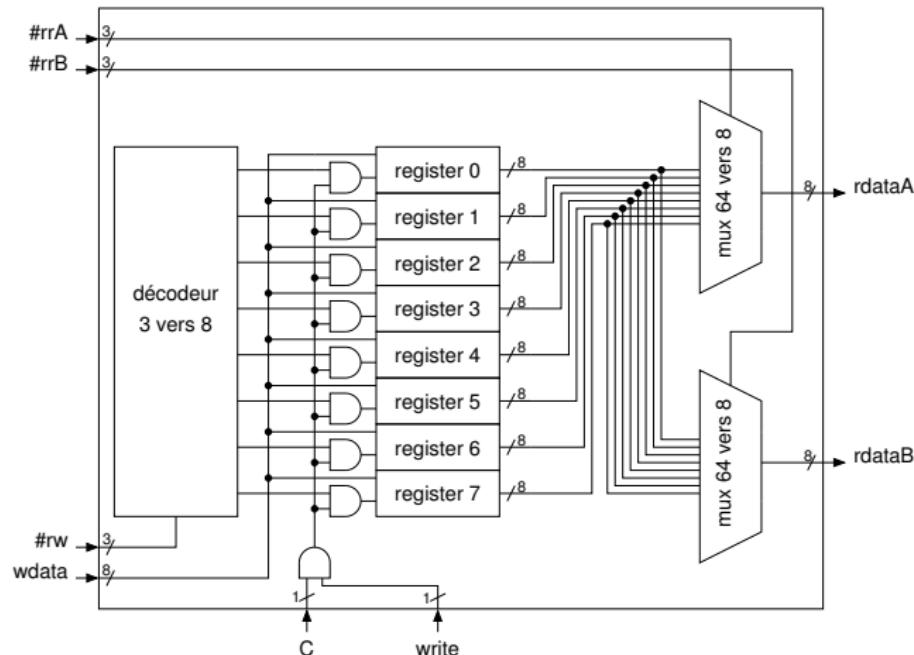


FIGURE - Un banc à registres 64 bits.

Banc à registres et RAM

Les registres et les bancs de registres fournissent les éléments de base pour la construction de petites mémoires.

Pour la construction de mémoire plus grandes, comme la mémoire centrale d'un ordinateur, on doit recourir à d'autres technologies. On distingue essentiellement deux types de mémoires RAM :

- Dans une *SRAM* (Static RAM), les bits sont stockés par des verrous : les données stockées se conservent tant que l'ordinateur est sous tension.
- Dans une *DRAM* (Dynamic RAM), les bits sont stockés à l'aide de condensateurs, dont il faut rafraîchir la charge à intervalles de temps réguliers : ils ne conservent leur charge que pendant quelques ms.

Banc à registres et RAM

Qu'est-ce qui distingue la SRAM et la DRAM ?

A capacité équivalente :

- le temps d'accès à une DRAM est 5 à 10 fois plus long qu'avec une SRAM.
- une SRAM présente un coût beaucoup plus élevé qu'une DRAM.

La SRAM et la DRAM jouent des rôles différents dans la hiérarchie mémoire :

- la SRAM est utilisée pour les niveaux de cache du processeur (\approx Mio),
- la DRAM compose la mémoire centrale (\approx Gio).

Certaines mémoires RAM sont asynchrones, ce qui signifie qu'elles ne sont pas cadencées par une horloge. Mais il existe aussi des DRAM synchrones :

- *SDRAM* (Synchronous DRAM) : opérations sur un front d'horloge.
- *DDR SDRAM* (Double Data Rate SDRAM) : sur les deux fronts d'horloge.

Banc à registres et RAM

Ces mémoires sont souvent plus sujettes aux erreurs. Quelles sont les conséquences ?

Les conséquences d'une erreur mémoire peuvent être très variées.

- Dans les systèmes sans code correcteur d'erreurs, une erreur peut conduire à une panne de l'ordinateur ou à la corruption des données,
- Dans les centres de données importants, les erreurs de mémoire sont une des causes de panne de matériel les plus fréquentes,
- Une erreur de mémoire peut n'avoir aucune conséquence : si elle affecte une partie de la mémoire qui n'est pas utilisée ou si elle affecte une partie d'un programme sans en changer le fonctionnement.

Une *mémoire à code correcteur d'erreurs* (ou *Error-Correcting Code Memory*) est un type de mémoire contenant un code correcteur permettant de détecter et de corriger les types les plus courants de corruption de données. Ce type de mémoire est particulièrement utilisé dans les ordinateurs où la corruption de données ne peut être tolérée en aucun cas, comme pour les calculs scientifiques ou financiers par exemple.

Mémoire à code correcteur d'erreurs

Comme on l'a précisé, les mémoires aujourd'hui sont majoritairement électroniques.

Un des problèmes majeurs des supports de stockage actuels est leur sensibilité aux corruptions logiques.

| Support mémoire | Taux d'erreurs |
|------------------|--|
| Disquette | $\approx 10^{-9}$: à 5 Mo/s, 3 bits erronés par min |
| CD-ROM optique | $\approx 10^{-5}$: 7ko erronés sur un CD de 700 Mo |
| HDD | $\approx 10^{-4}$: environ 10^6 erreur pour 10 Go |
| Flash et SSD | $\approx 10^{-5}$ |
| Semi-conducteurs | $\approx 10^{-9}$ |

TABLEAU 2 - Ordre de grandeur du taux d'erreur sur des supports mémoire

Mémoire à code correcteur d'erreurs

Il existe plusieurs algorithmes de détection et de correction des erreurs. Parmi les plus connus, on retrouve :

- *Les codages de parité,*
- *Codage de Hamming,*
- BCH,
- Codage de Reed-Solomon.

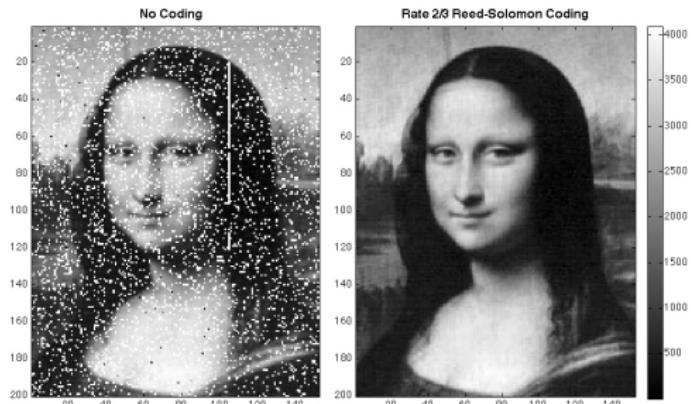


FIGURE 6 - Correction par l'algorithme de Reed-Solomon d'une image corrompue

Mémoire à code correcteur d'erreurs

Définition (Codage par blocs)

Soit $x \in \{0, 1\}^k$. Un *codage par blocs* transforme alors une séquence $x = x_1x_2\dots x_k$ en un bloc de $n = k + r$ bits, $\varphi(x) = y_1y_2\dots y_ky_{k+1}\dots y_n$ qui comporte r symboles de redondance.

L'ensemble $C_\varphi = \{\varphi(x) | x \in \{0, 1\}^k\}$, image par φ de $\{0, 1\}^k$ (où φ est injective) est appelé un code(n, k).

Ainsi, une erreur est détectée si l'on constate une suite de bits x' telle que $x' \notin C_\varphi$.

Le rendement R d'un code(n, k) est tel que $R = \frac{k}{n}$.

Mémoire à code correcteur d'erreurs

Définition (Distance de Hamming)

On appelle *poids de Hamming* d'une séquence $x = x_1 \dots x_k$ avec $x_i \in \{0, 1\}$ le nombre $w(x) = \sum_{i=1}^k x_i$.

Soit $(x, y) \in \{0, 1\}^k \times \{0, 1\}^k$. On note également $x \oplus y$, l'opérateur ou-exclusif bit à bit (ou addition)

La *distance de Hamming* est le nombre de bits pour lesquels x et y diffèrent, soit :

$$d_{\mathcal{H}}(x, y) = w(x \oplus y)$$

Exemple

- $d_{\mathcal{H}}(10010, 01010) = 1 \oplus 0 + 0 \oplus 1 + 0 \oplus 0 + 1 \oplus 1 + 0 \oplus 1 = 2$
- $d_{\mathcal{H}}(00110, 10101) = 3$

Mémoire à code correcteur d'erreurs

La distance de Hamming permet de caractériser le nombre d'erreurs que peut corriger un code C_φ de longueur n . En effet, si $x \in C_\varphi$, une séquence de n bits et soit x' la séquence considérée en mémoire, supposée différente de x , alors $d_H(x, x') > 0$.

Pour pouvoir corriger correctement x' , il faut que :

- ① $x' \notin C_\varphi$
- ② x soit la séquence la plus proche de x' : $\forall y \in C_\varphi$, si $y \neq x$ alors $d_H(y, x') > d_H(x, x')$

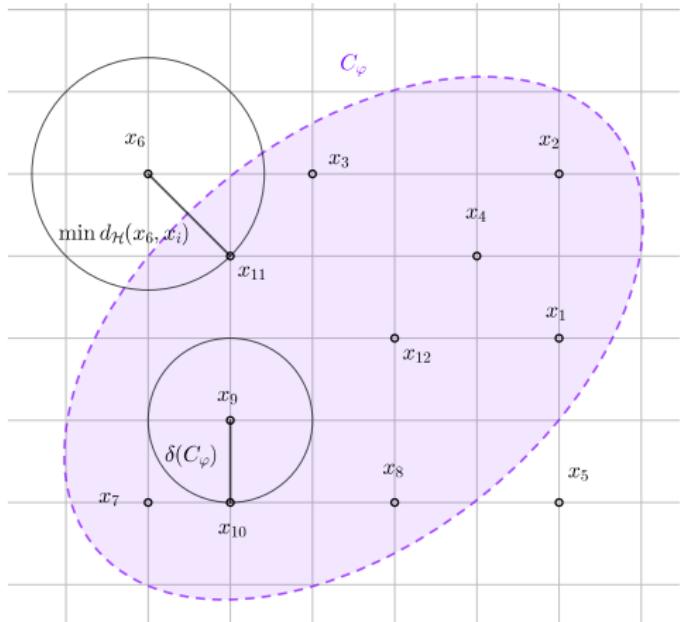


FIGURE 7 - Espace de codage C_φ

Mémoire à code correcteur d'erreurs

Définition (Contrôle de parité (*checksum*))

Soit une séquence $x = x_1x_2\dots x_k$ avec $x_i \in \{0, 1\}$. On pose $\varphi(x) = x_1\dots x_k, x_{k+1}$ où $x_{k+1} = w(x) \bmod 2$.

Cette méthode est appelé *contrôle par bit de parité*.

Exemple

- $\varphi(01110) = 011101$
- $\varphi(11110) = 111100$

Corollaire

Toute séquence $x \in C_\varphi$ est telle que $w(x) \equiv 0 \bmod 2$.

C'est-à-dire que toute séquence codée comporte un nombre pair de 1.

Mémoire à code correcteur d'erreurs

Définition (Parité longitudinale et transversale)

Soit une séquence $x = x_1x_2\dots x_k$ avec $x_i \in \{0, 1\}$. On construit un code(n, k_1k_2) avec $k = k_1 \times k_2$. On considère X la matrice représentative de x de dimension $k_1 \times k_2$ et telle que $x_{ij} = x_{(i-1) \times k_2 + j}$. Le code est alors une matrice $M = \varphi(X)$ de taille $(k_1 + 1) \times (k_2 + 1)$ telle que :

$$\varphi(X) = \begin{cases} m_{ij} = w(X_i) \bmod 2 & \text{si } j = k_2 + 1 \\ m_{ij} = w({}^t X_j) \bmod 2 & \text{si } i = k_1 + 1 \\ m_{ij} = w(x) \bmod 2 & \text{si } i = k_1 + 1 \text{ et si } j = k_2 + 1 \\ m_{ij} = x_{ij} & \text{sinon} \end{cases}$$

Avec :

- X_i , le vecteur ligne représentant la i -ème ligne extraite de la matrice X .
- X_j , le vecteur colonne représentant la j -ème colonne extraite de la matrice X .

Cette méthode est appelée codage par *parité longitudinale et transversale*.

Mémoire à code correcteur d'erreurs

Exemple

Soit $x = 0101\ 0010\ 0010\ 0100\ 0000\ 0$. Il vient que

$$X = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \text{ Dès lors}$$

$$\varphi(X) = \varphi \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{0} & 1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 \end{pmatrix}$$

Mot erronné : $\begin{matrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & \textcolor{red}{1} & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \textcolor{blue}{0} & 0 & 0 & 1 \end{matrix}$ $\xrightarrow{\text{correction}}$ $\begin{matrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{matrix}$

Mémoire à code correcteur d'erreurs

Définition (Code t —{déTECTeur, CORRECTEUR})

Un code(n, k) est dit t —déTECTeur (resp. t —CORRECTEUR) s'il permet de détECTer (resp. corriger) toute erreur sur t bits ou moins d'une séquence de n bits.

Exemple

- L'ajout d'un bit de parité est pour l'exemple donné précédemment un code(6, 5). Il est 1—déTECTeur et 0—CORRECTEUR avec un rendement de 83,333%.
- Le contrôle de parité longitudinale et transversale sur 21 bits est un code(32, 21). Il est 1-CORRECTEUR avec un rendement 65,625%

Mémoire à code correcteur d'erreurs

Définition (Distance minimale δ)

Soient $x_1 \in \{0, 1\}^n$ et $x_2 \in \{0, 1\}^n$, on note $\varphi(x_1) = c_1$ et $\varphi(x_2) = c_2$.

On appelle *distance minimale* $\delta(C_\varphi)$ de la fonction de codage φ , la distance définie telle que $\delta(C_\varphi) = \inf_{c_1 \neq c_2} d_H(c_1, c_2)$, c'est-à-dire la plus petite distance séparant deux éléments de l'ensemble C_φ .

Théorème

Soit φ un code de longueur n . Les propriétés suivantes sont équivalentes et impliquent que φ est t -correcteur :

- ① $\forall x \in \{0, 1\}^n, \#\{c | c \in C_\varphi, d_H(x, c) \leq t\} \leq 1$
- ② $\forall (c_1, c_2) \in C_\varphi^2, c_1 \neq c_2 \Rightarrow d_H(c_1, c_2) > 2t$
- ③ $\delta(C_\varphi) \geq 2t + 1$

Optimisations mémoire

Malgré l'amélioration des performances et la relative 'bonne' vitesse des mémoires électriques, la mémoire demeure plus lente que le processeur ! (Près de 10 fois plus lente)

- Plusieurs instructions en une nanoseconde.
- Quelques dizaines de nanosecondes pour récupérer une donnée en mémoire.

Et les mémoires rapides coûtent très cher.

Une solution serait de créer une mémoire rapide, pas trop grosse et qui contienne les informations utiles pour le processeur au moment t .

⇒ Principe de la *mémoire cache* multi-niveaux (ou *antémémoire*).

- Utilisation de mémoire petite et d'un algorithme d'accès de complexité efficace.
- Algorithme prédictif de *pre-fetching* (ou prélecture) permettant de charger en avance les données utiles au CPU.

Optimisations mémoire

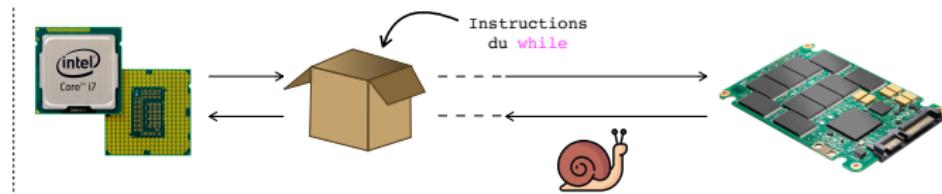
Les algorithmes de *pre-fetching* se basent sur les principes de localité :

Définition (Principe de localité spatiale 1)

Les données sont stockées en mémoire de manière contigüe :

⇒ les informations voisines à une information demandée ont toutes les chances d'être demandées également.

```
while(reste != 0) {  
    a = b;  
    b = reste;  
    reste = a % b;  
}  
pgcd = b;
```

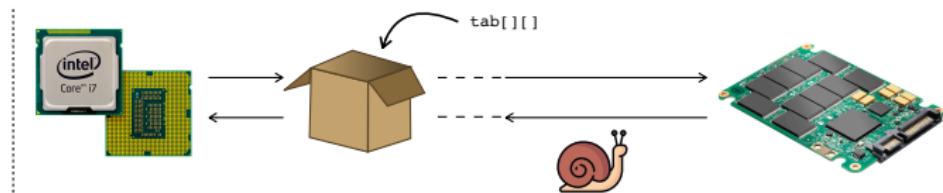


Optimisations mémoire

Définition (Principe de localité spatiale 2)

Les instructions sont stockées en mémoire de manière contigüe :
⇒ les instructions voisines à une instruction demandée ont toutes les chances d'être demandées également.

```
for(i = 0; i < n; i++) {  
    for(j = 0; j < m; j++) {  
        tab[i][j] = i * j;  
    }  
}
```



Optimisations mémoire

Définition (Principe de localité temporelle)

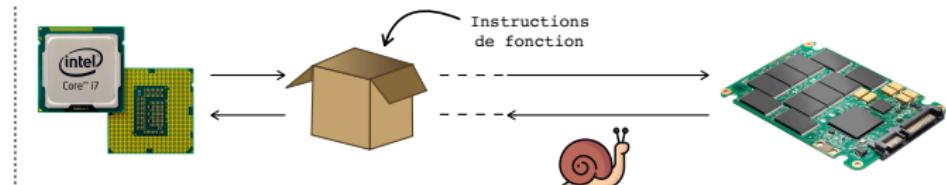
Traitements répétitifs dans un programme :

- boucles **for**,
- boucles **while**,

codes des fonctions appelées dans de telles boucles.

⇒ Un groupe d'instructions récemment exécutées est susceptible de l'être à nouveau dans un avenir proche.

```
for(i = 0; i < n ; i++) {  
    fib = fibo(i);  
    printf("fib(%d) = %d",  
        i, fib);  
}
```



Optimisations mémoire

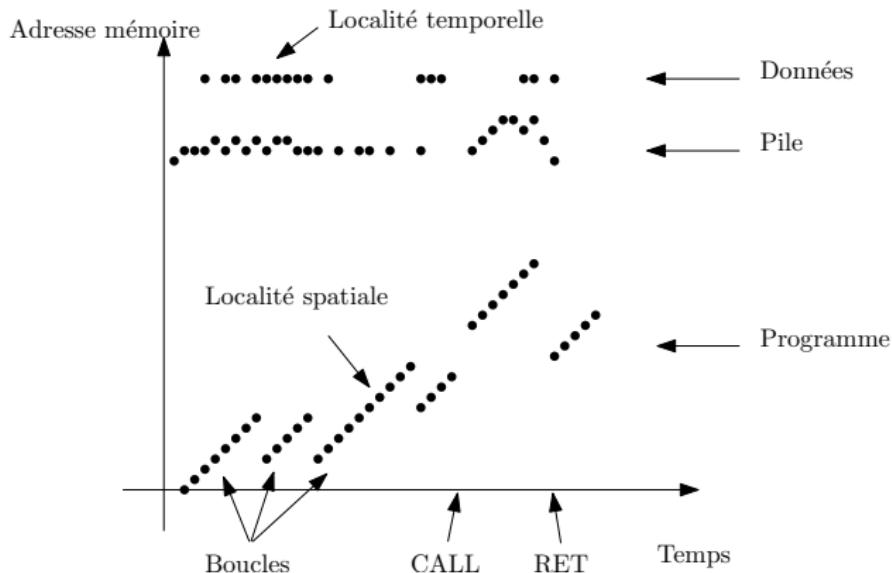


FIGURE - Illustration des principes de localité spatiale et temporelle

Optimisations mémoire

Comment placer les adresses dans un cache ?

Il existe trois méthodes principales de gestion des données dans l'antémémoire :

- ① *Cache direct* – Basé sur une fonction de hachage h ,
- ② *Cache associatif total* – On charge quand on peut les données et on les évacue selon un algorithme de file d'attente (FIFO, LRU, etc.),
- ③ *Cache mixte* (ou associatif par ensemble) – Mélange les deux approches précédentes.

Optimisations mémoire

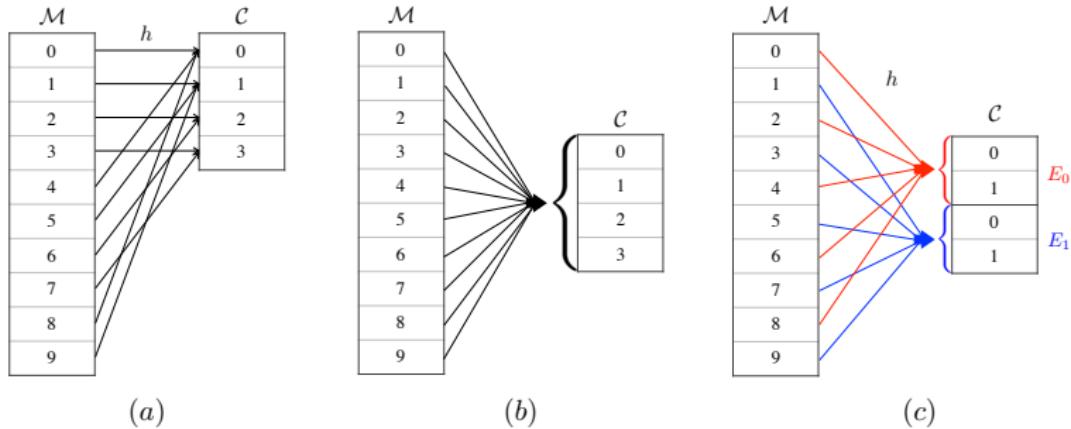


FIGURE - Les différentes techniques de rangement (a) Cache direct (b) Cache associatif total (c) Cache mixte

Conclusion

Pour terminer cette partie, j'aimerais revenir sur quelques aspects fondamentaux de la mémoire :

- Il existe plusieurs méthodes d'accès à la mémoire : *séquentielle, semi-directe, aléatoire et associatif.*
 - Ces méthodes d'accès conditionnent la rapidité de la mémoire par la *complexité d'accès.*
- Les technologies utilisées sont également un paramètre important de la rapidité de la mémoire.
 - Aujourd'hui, les technologies *flash* et *mémoires à semi-conducteurs* sont de plus en plus utilisées : *SSD, mémoire vive, etc...*
- La mémoire est implémentée de façon logique grâce aux circuits séquentiels.
 - On parle alors de verrous ou de bascules. Il en existe plein et ces circuits peuvent être synchrones ou asynchrones.
 - Un abstraction des circuits synchrones peut-être obtenu à l'aide des automates à états finis de Moore.

Conclusion

- Les mémoires à semi-conducteurs issues ou électroniques sont particulièrement sensibles aux *corruptions logiques*.
 - Certaines mémoires implémentent des *codes correcteurs* d'erreurs afin de prévenir ces aléas : les détecter et/ou les corriger.
 - Une des techniques les plus utilisées est celle des *bits de parité*.
- La mémoire principale est près de 10 fois plus lente que le processeur.
 - Création d'une petite mémoire de proximité rapidement accessible et contenant les informations essentielles au temps t : *Caches*.
 - Basée sur les *principes de localité spatiale et temporelle*.