

Les mémoires

Eric Cariou

*Université de Pau et des Pays de l'Adour
UFR Sciences Pau - Département Informatique*

Eric.Cariou@univ-pau.fr

Mémoire

- ◆ Mémoire
 - ◆ Dispositif capable d'enregistrer, de conserver et de restituer des informations
 - ◆ Informations binaires pour un ordinateur
- ◆ On classe les mémoires selon
 - ◆ Caractéristiques : capacité, débit ...
 - ◆ Type d'accès : séquentiel, direct ...

Organisation de l'information

- ◆ Unité de base : bit
 - ◆ Le plus petit élément de stockage
- ◆ Octet (ou *byte*) : groupe de 8 bits
- ◆ Le caractère (7, 8 ou 16 bits)
 - ◆ Codage selon un standard (ASCII, Unicode ...)
- ◆ Mot : groupement d'octets (8, 16, 32, 64 ...)
 - ◆ Unité d'information adressable en mémoire
- ◆ Enregistrement : bloc de données
- ◆ Fichier : ensemble d'enregistrements

Caractéristiques des mémoires

- ◆ Adresse
 - ◆ Valeur numérique référençant un élément de mémoire
- ◆ Capacité ou taille
 - ◆ Nombre d'informations que peut contenir la mémoire
 - ◆ S'exprime en nombre de mots ou plus généralement en octets
 - ◆ 512 Ko, 8 Mo, 500 Go, 2 To ...
- ◆ Temps d'accès
 - ◆ Temps s'écoulant entre le lancement d'une opération de lecture/écriture et son accomplissement

Caractéristiques des mémoires

- ◆ Cycle mémoire
 - ◆ Temps minimal entre 2 accès successifs à la mémoire
 - ◆ Cycle > temps d'accès
 - ◆ Car besoin d'opérations supplémentaires entre 2 accès (stabilisation des signaux, synchronisation ...)
- ◆ Débit
 - ◆ Nombre d'informations lues ou écrites par seconde
 - ◆ Exemple : 300 Mo/s
- ◆ Volatilité
 - ◆ Conservation ou disparition de l'information dans la mémoire hors alimentation électrique de la mémoire

Méthodes d'accès

- ◆ Accès séquentiel
 - ◆ Pour accéder à une information on doit parcourir toutes les informations précédentes
 - ◆ Accès lent
 - ◆ Exemple : bandes magnétiques (K7 vidéo)
- ◆ Accès direct
 - ◆ Chaque information a une adresse propre
 - ◆ On peut accéder directement à chaque adresse
 - ◆ Exemple : mémoire centrale d'un ordinateur

Méthodes d'accès

- ◆ Accès semi-séquentiel
 - ◆ Intermédiaire entre séquentiel et direct
 - ◆ Exemple : disque dur
 - ◆ Accès direct au cylindre
 - ◆ Accès séquentiel au secteur sur un cylindre
- ◆ Accès associatif/par le contenu
 - ◆ Une information est identifiée par une clé
 - ◆ On accède à une information via sa clé
 - ◆ Exemple : mémoire cache

Types de mémoire

- ◆ 2 grandes familles
 - ◆ Mémoires non volatiles : ROM (Read Only Memory)
dites mémoires mortes
 - ◆ Leur contenu est fixe (ou presque ...)
 - ◆ Conservé en permanence
 - ◆ Mémoires volatiles : RAM (Random Access Memory)
dites mémoires vives
 - ◆ Leur contenu est modifiable
 - ◆ Perte des informations hors alimentation électrique
 - ◆ Random : à prendre dans le sens « accès sans contraintes »
(et non pas aléatoire)

Mémoires non volatiles (ROM)

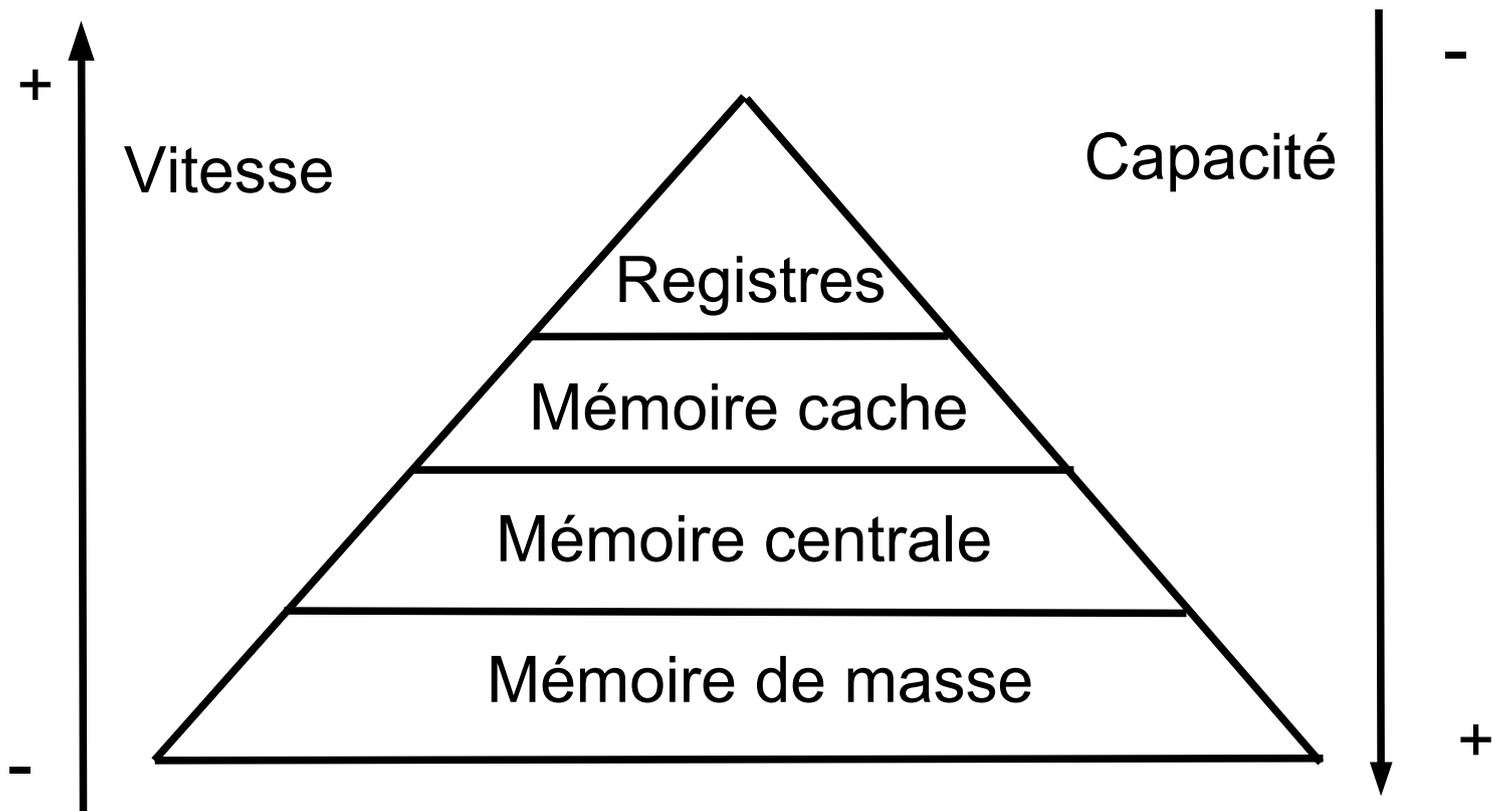
- ◆ ROM
 - ◆ « Câblage en dur » de l'information
 - ◆ Premier type de mémoire morte, on a gardé son nom pour toute cette famille
- ◆ PROM : mémoire programmable une seule fois
- ◆ EPROM : mémoire reprogrammable (via des ultra-violets)
- ◆ EEPROM : mémoire reprogrammable (électriquement)
 - ◆ Exemple : BIOS d'un ordinateur

Mémoires volatiles (RAM)

- ◆ 2 grands types de RAM
 - ◆ DRAM : Dynamic RAM
 - ◆ Dynamique : nécessite un rafraîchissement périodique de l'information
 - ◆ Peu coûteuse
 - ◆ SRAM : Static RAM
 - ◆ Statique : ne nécessite pas de rafraîchissement
 - ◆ Beaucoup plus rapide que la DRAM
 - ◆ Mais beaucoup plus chère

Hiérarchie mémoire

- ◆ Dans un ordinateur, plusieurs niveaux de mémoire



Registres

- ◆ Se trouvent intégrés dans le CPU
- ◆ Servent à gérer l'exécution des opérations
 - ◆ Spécialisés, ex : adresse de la prochaine instruction en mémoire
 - ◆ Généraux : résultats intermédiaires des calculs
- ◆ Taille globale faible
 - ◆ Jusqu'à quelques dizaines/centaines d'une taille de quelques octets
- ◆ Très rapides (vitesse du CPU)
- ◆ *Voir le cours sur les processeurs*

Mémoire cache

- ◆ Mémoire intermédiaire entre le processeur et la mémoire centrale
- ◆ Mémoire cache est intégrée dans le processeur et est cadencée à la même fréquence
- ◆ But de la mémoire cache
 - ◆ Débit de la mémoire centrale très lent par rapport au débit requis par le processeur
 - ◆ On accélère la vitesse de lecture des informations par le CPU en les plaçant (en avance) dans le cache
- ◆ Mémoire associative
- ◆ De type SRAM car doit être rapide
- ◆ Taille : jusqu'à quelques dizaines de Mo

Mémoire centrale

- ◆ Taille : jusqu'à plusieurs (dizaines de) Go
- ◆ Accès direct
- ◆ De type DRAM car moins cher
- ◆ Vitesse relativement lente

Exemple de comparaison vitesse cache/centrale

- ◆ Mémoire : SDRAM-DDR 2100
- ◆ Processeur : AMD Athlon XP 2200+ (1.8 Ghz)
- ◆ Dans les 2 cas : lecture de mots de 64 bits
- ◆ Mémoire
 - ◆ Fréquence de 133 Mhz et mots de 64 bits
 - ◆ 2 accès par cycle horloge (DDR = Double Data Rate)
 - ◆ Débit théorique maximum de 2,1 Go/s (moins en pratique)
- ◆ Processeur
 - ◆ Cache L1 du processeur : débit mesuré de 18 Go/s
 - ◆ Cache L2 du processeur : débit mesuré de 5,6 Go/s

source : <http://www.x86-secret.com>

Mémoire de masse

- ◆ Mémoire de grande capacité : de quelques Go à plusieurs To
- ◆ Mémoire non volatile
 - ◆ Stockage de programmes, données, fichiers en tout genre
- ◆ Très lente
- ◆ Exemples
 - ◆ Disque dur
 - ◆ Clés USB
 - ◆ DVD

Hiérarchie mémoire : conclusion

- ◆ Organisation de façon à ce que le CPU accède le plus rapidement possible aux données utiles
 - ◆ Instructions des programmes en cours d'exécution
 - ◆ Données manipulées par ces programmes
 - ◆ Ces instructions et données sont chargées en mémoire centrale à partir d'une mémoire de masse
- ◆ Hiérarchie
 - ◆ Mémoire cache : rapide et petit
 - ◆ Mémoire centrale : moins rapide et plus gros
 - ◆ Mémoire de masse : lent et très gros
- ◆ Plus une mémoire est rapide, plus elle est chère

Réalisation de mémoires

◆ Pour les DRAM

- ◆ Un bit = un transistor et un condensateur
- ◆ Le condensateur stocke la valeur binaire
- ◆ Doit être rafraîchi régulièrement pour conserver la valeur stockée dans le condensateur
 - ◆ Ralentit la vitesse d'accès à la mémoire

◆ Pour les SRAM

- ◆ Un bit = 4 transistors = 2 portes NOR
- ◆ Peut notamment les construire à partir de bascules RS

Mémoires à partir de bascules RS

- ◆ Une bascule stocke un bit
- ◆ Un mot = un groupe de bits (8, 16 ...)
 - ◆ Correspond à un groupe de 8, 16 ... bascules
- ◆ Accès aux informations/bascules
 - ◆ Lecture/écriture de tous les bits d'un mot en parallèle et en même temps
 - ◆ Accès mot par mot et non pas bit par bit

Mémoires à partir de bascules RS

- ◆ Principe général pour accès aux données
 - ◆ Bits de contrôle pour déclencher opération lecture/écriture
 - ◆ X bits de données à écrire et X bits de données lus
 - ◆ X = taille du mot d'accès à la mémoire
- ◆ Bits de contrôle
 - ◆ Un bit d'écriture et un bit de lecture par mot
 - ◆ Si à 1, sélectionne le mot en écriture/lecture
- ◆ Bits de données
 - ◆ Un bit d'écriture en entrée par bit
 - ◆ Si bit d'écriture du mot à 1 : valeur du bit à écrire
 - ◆ Un bit de lecture en sortie par bit
 - ◆ Si bit de lecture du mot à 1 : contient la valeur du bit lu

Mémoires à partir de bascules RS

◆ Opération d'écriture

- ◆ Le bit d'écriture du mot à écrire est positionné à 1 (les autres à 0)
- ◆ Les X bits d'écriture de bits sont positionnés et seront écrits dans le mot sélectionné

◆ Opération de lecture

- ◆ Le bit de lecture du mot à lire est positionné à 1 (les autres à 0)
- ◆ Les X bits de lecture de bits contiennent les bits du mot sélectionné

Mémoires à partir de bascules RS

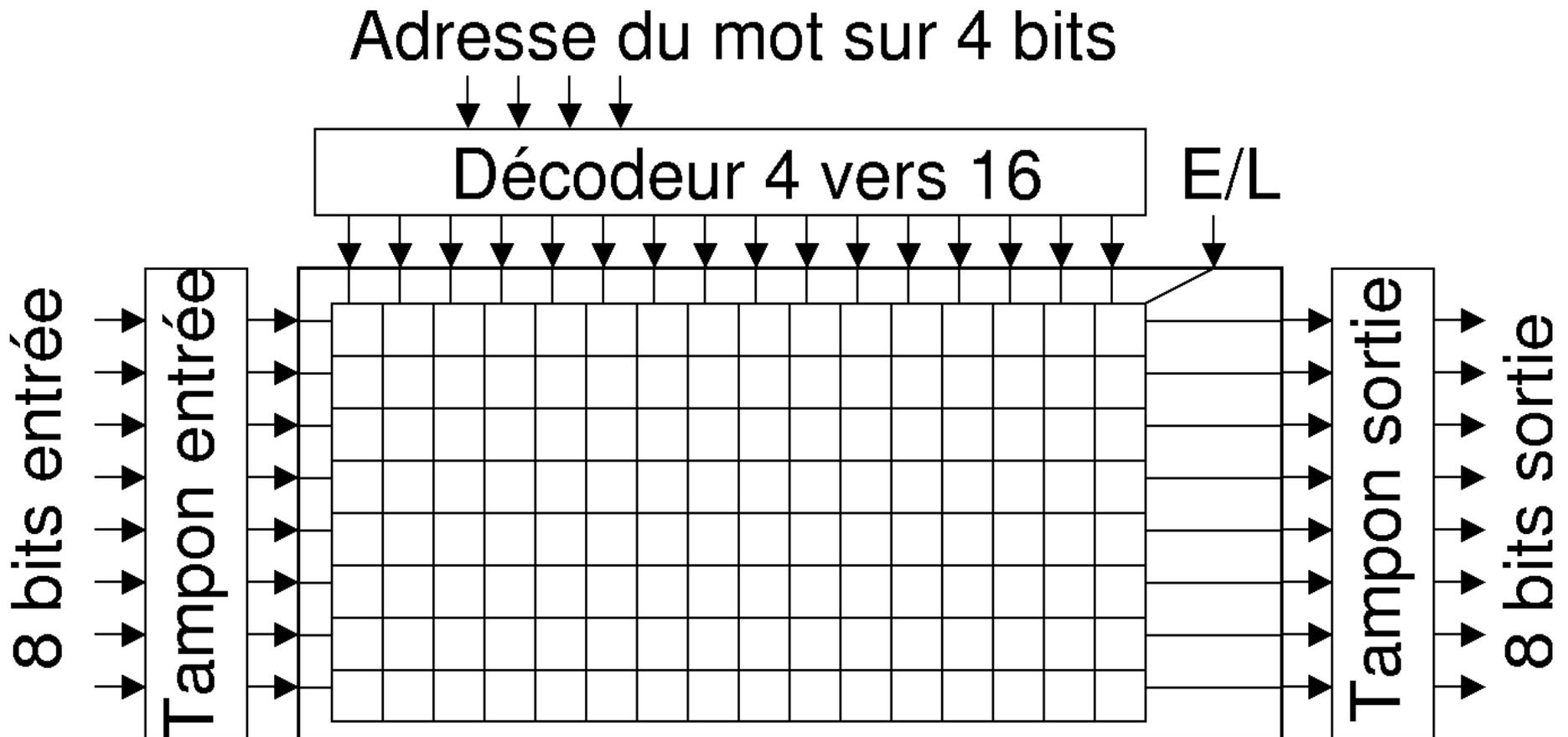
- ◆ Pour un bloc mémoire de 16 mots de 8 bits
- ◆ En entrée du bloc mémoire
 - ◆ Le contenu du mot à écrire : 8 bits
 - ◆ La sélection des mots à lire/écrire : 16 bits
 - ◆ Un bit pour différencier l'opération d'écriture ou de lecture. Écriture = 1, Lecture = 0
- ◆ En sortie du bloc mémoire
 - ◆ Le contenu du mot lu : 8 bits
- ◆ Pour sélection du mot à lire/écrire
 - ◆ Peut utiliser un décodeur sur 4 bits : 4 vers 16

Mémoires à partir de bascules RS

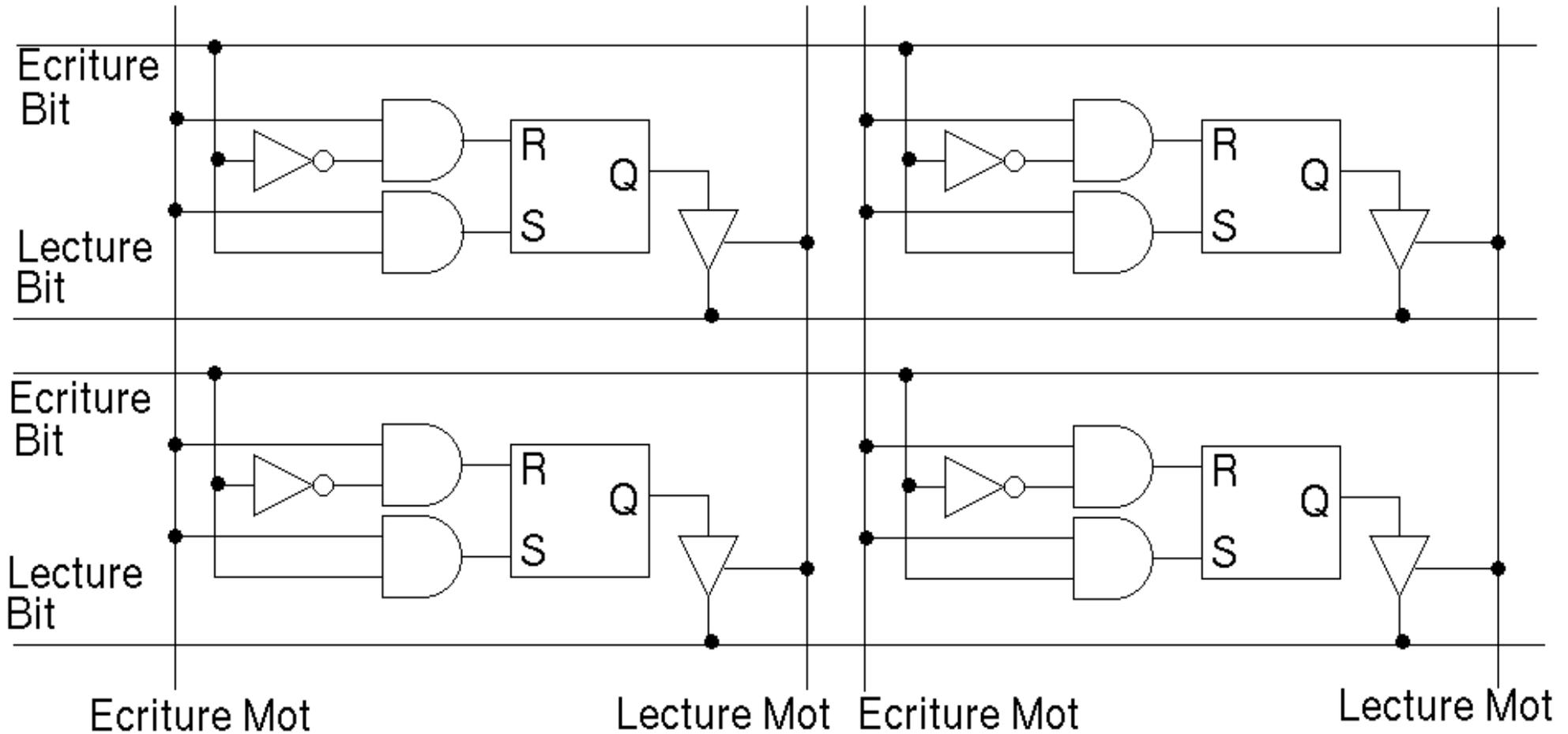
- ◆ Bloc mémoire général
 - ◆ Constitué de blocs élémentaires
 - ◆ Chaque bloc élémentaire stocke un bit d'un mot
 - ◆ Au total : matrice de 8 x 16 blocs élémentaires
- ◆ Pour un bloc élémentaire
 - ◆ En entrée
 - ◆ Me : bit d'écriture de mot
 - ◆ MI : bit de lecture de mot
 - ◆ Be : valeur du bit à écrire
 - ◆ En sortie
 - ◆ BI : valeur du bit lu

Mémoire à partir de bascules RS

- ◆ Schéma général d'une matrice 16 x 8 bits



Mémoires à partir de bascules RS



4 blocs élémentaires

Mémoire à partir de bascules RS

◆ Pour l'écriture, en entrée de la bascule

◆ $R = Me\overline{Be}$ et $S = MeBe$

◆ Table de vérité

Me	Be		R	S		Q+
0	0		0	0		Q
0	1		0	0		Q
1	0		1	0		0
1	1		0	1		1

◆ $Me = 0$: pas d'écriture = maintien de la valeur mémorisée dans la bascule

◆ $Me = 1$: on a bien $Q+ = Be =$ le bit à écrire

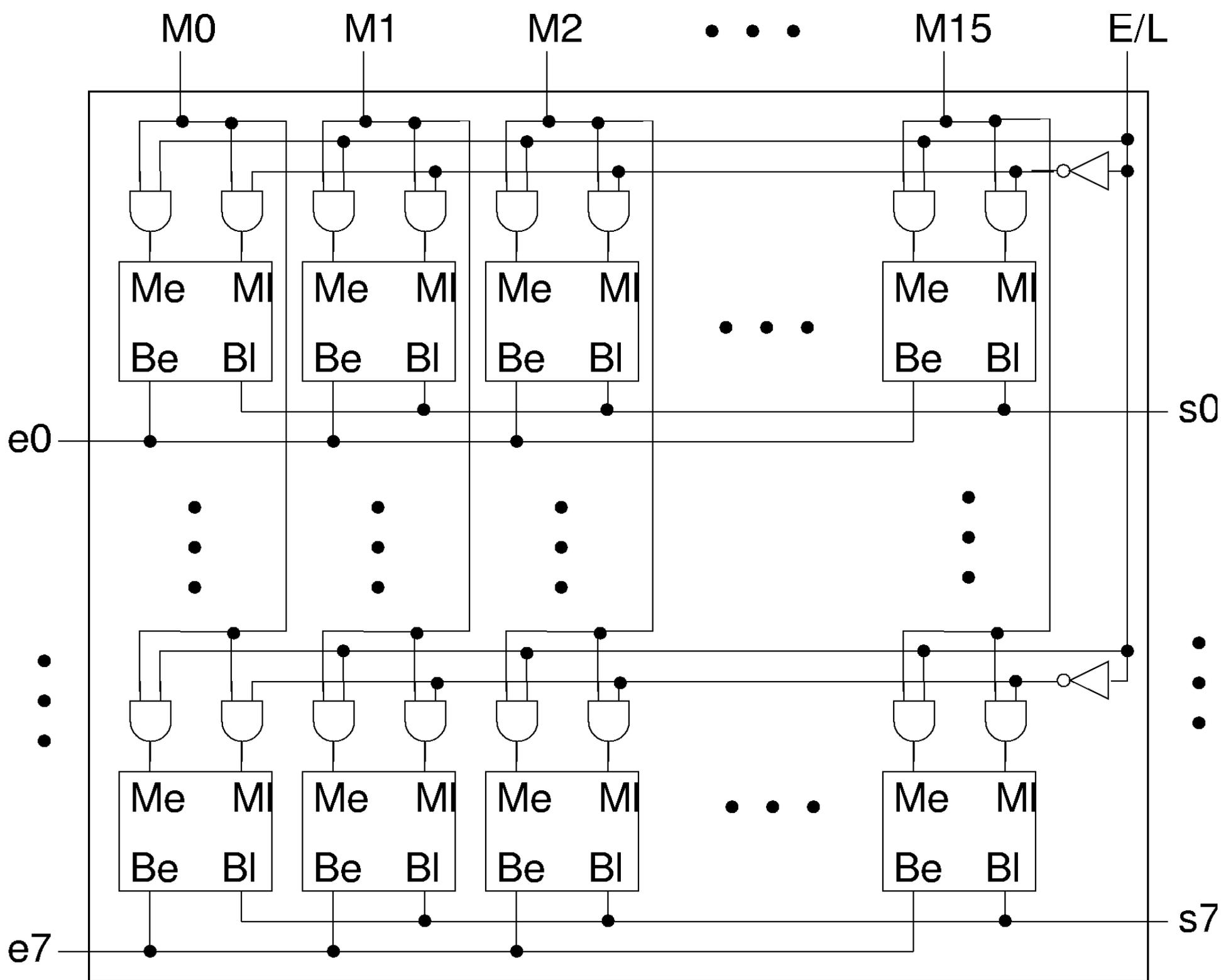
Mémoire à partir de bascules RS

- ◆ Pour la lecture : portes logique à 3 états
 - ◆ $MI = 0$, pas de lecture
 - ◆ $MI = 1$, lecture du bit
 - ◆ Le bit lu est placé dans BI

◆ Tables de vérité

Ml	Q		Bl
0	0		X
0	1		X
1	0		0
1	1		1

Ml		Bl
0		X
1		Q



La mémoire cache

Mémoire cache

- ◆ Processeur a besoin d'un débit soutenu en lecture d'instructions et de données
 - ◆ Pour ne pas devoir attendre sans rien faire
- ◆ Problème
 - ◆ Mémoire centrale qui stocke ces instructions et données est beaucoup trop lente pour assurer ce débit
- ◆ Idée
 - ◆ Utiliser une mémoire très rapide intermédiaire entre la mémoire centrale et le processeur
 - ◆ Mémoire cache (ou cache tout court)

Mémoire Cache

◆ Problèmes

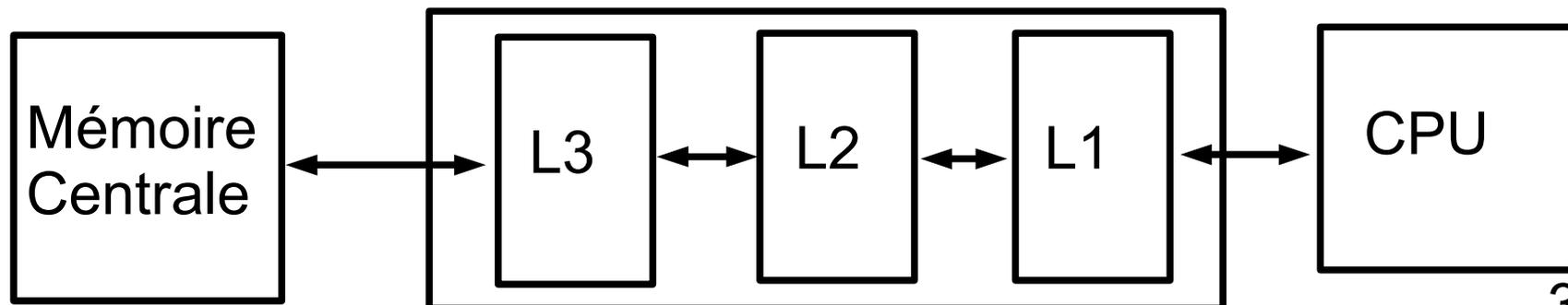
- ◆ Mémoire cache doit être petite (quelques centaines de Ko à quelques dizaines de Mo) pour être efficace en terme de débit
- ◆ Ne peut donc pas y stocker tous les programmes en cours d'exécution et leurs données

◆ Solutions

- ◆ Algorithmes pour « deviner » et mettre dans le cache les données/instructions avant que le CPU en ait besoin
- ◆ Recherche bon compromis entre tailles, types de cache (données/instructions), niveaux de cache, techniques d'accès au cache ... pour meilleures performances

Organisation en niveaux

- ◆ Cache est organisé en plusieurs niveaux
 - ◆ Jusqu'à 3 niveaux (L1, L2 et L3)
 - ◆ De rares processeurs avec un L4
 - ◆ L3 généralement utilisé pour l'échange de données entre cores dans un processeur multi-core
 - ◆ L1 et L2 généralement spécifique à un core
 - ◆ Cache L1 généralement scindé en 2 parties : un dédié aux instructions et l'autre aux données
 - ◆ Cache L_{i+1} joue le rôle de cache pour le niveau L_i
 - ◆ Cache L_{i+1} plus grand que L_i mais moins rapide en temps d'accès aux données



Organisation en niveaux

- ◆ Relation entre les niveaux de cache
 - ◆ Cache inclusif
 - ◆ Le contenu du niveau L1 se trouve aussi dans L2
 - ◆ Cache exclusif
 - ◆ Le contenu des niveaux L1 et L2 sont différents
- ◆ Cache inclusif
 - ◆ Taille globale du cache : celle du cache L2
- ◆ Cache exclusif
 - ◆ Taille globale du cache : taille L1 + taille L2

Organisation en niveaux

- ◆ Avantages et inconvénients de chaque approche
 - ◆ Exclusif
 - ◆ Cache plus grand au total
 - ◆ L2 de taille quelconque
 - ◆ Mais doit gérer la non duplication des données : prend du temps, L2 moins performant
 - ◆ Inclusif
 - ◆ Cache L2 plus performant
 - ◆ Mais taille totale plus faible
 - ◆ Et contraintes sur la taille de L2 (ne doit pas être trop petit par rapport à L1)
 - ◆ Cache inclusif : historiquement le plus utilisé
 - ◆ Mais trouve aussi des CPU à cache exclusif (AMD)

Performance

- ◆ Évaluation de la performance d'une mémoire cache
 - ◆ Taux de succès le plus élevé possible
 - ◆ Succès : la donnée voulue par le CPU est dans le cache L1
 - ◆ Ou la donnée voulue par le cache L1 est dans le cache L2 ...
 - ◆ Échec : la donnée voulue n'y est pas
 - ◆ Doit la charger à partir du niveau de cache suivant ou de la mémoire centrale
 - ◆ A prendre en compte également
 - ◆ Temps d'accès à la mémoire, latences : nombre de cycle d'horloges requis pour
 - ◆ Lire une donnée dans le cache en cas de succès
 - ◆ Aller la récupérer en cas d'échec au niveau supérieur
 - ◆ Latences dépendent de la taille des niveaux et de l'organisation des données dans les niveaux

Performance : niveaux

- ◆ Choix de taille du cache et du nombre de niveaux
 - ◆ Augmentation de la taille de la mémoire cache
 - ◆ Augmente le taux de succès
 - ◆ Mais ne gagne pas forcément en performance car augmentation du temps d'accès proportionnellement à la taille
 - ◆ Augmentation du nombre de niveaux (plus de 3)
 - ◆ Pas de gain supplémentaire vraiment significatif
- ◆ Cache inclusif
 - ◆ Le cache L2 doit être bien plus grand que le cache de niveau L1 car sinon on stocke peu de données supplémentaires dans L2 par rapport à L1 et donc taux de succès de L2 faible
- ◆ Ne pas oublier le coût élevé de ce type de RAM

Localité et pre-fetching

◆ Localité temporelle

- ◆ Une donnée référencée à un temps t aura de très fortes chances d'être référencée dans un futur proche

◆ Localité spatiale

- ◆ Si une donnée est référencée à un temps t , alors il y a de très fortes chances que les données voisines le soient dans un futur proche
- ◆ `for (i=0; i < N; i++)`
 `somme += A[i];`
- ◆ Localité spatiale : $A[i]$ ($A[i+1]$, $A[i+2]$...)
- ◆ Localité temporelle : N , A , i

Localité et pre-fetching

- ◆ Pour améliorer le taux de succès du cache
 - ◆ *Pre-fetching* : chargement en avance des données dont le CPU devrait avoir besoin
- ◆ Algorithmes de pre-fetching sont basés sur les principes de localité
 - ◆ Localité temporelle
 - ◆ Garder en mémoire cache les dernières données référencées par le programme
 - ◆ Localité spatiale
 - ◆ Charger en avance les données/instructions contiguës à une donnée/instruction référencée

Accès par lignes mémoire

- ◆ *Note : dans les transparents qui suivent, pour simplifier les explications, on considère que l'on a qu'un seul niveau de cache*
- ◆ Mémoire cache = sous-partie de la mémoire centrale
 - ◆ Comment est déterminée cette sous-partie ?
- ◆ Principe
 - ◆ Les échanges d'informations entre mémoire de niveaux différents se font par blocs d'adresses consécutives
 - ◆ Une ligne mémoire
 - ◆ Car pas beaucoup plus coûteux de lire une ligne qu'un seul mot
 - ◆ Et s'adapte bien au principe de localité spatiale

Accès par lignes mémoire

- ◆ Pour le processeur
 - ◆ La présence de la mémoire cache est transparente
 - ◆ Le CPU demande toujours à accéder à une adresse en mémoire centrale (pour lecture ou écriture)
- ◆ Adressage d'un mot mémoire
 - ◆ Adressage direct : adresse du mot en mémoire
 - ◆ Adressage via ligne : index de la ligne et déplacement dans la ligne

index	déplacement
<i>n bits</i>	<i>m bits</i>
 - ◆ Correspondance entre les 2 adresses
 $adresse\ mémoire = index \times taille_ligne + déplacement$

Cohérence données cache/mémoire

- ◆ Lecture/écriture en mémoire centrale via cache
 - ◆ Le cache contient une copie d'une partie de la mémoire centrale : nécessité de cohérence mutuelle
 - ◆ Opérations de lecture/écriture sont effectuées dans le cache avec répercussion sur la mémoire centrale
- ◆ Opération de lecture
 - ◆ Pas de modification de valeurs : toujours cohérent
- ◆ Opération d'écriture
 - ◆ Modification du contenu du cache : la partie équivalente en mémoire centrale n'est plus cohérente

Cohérence données cache/mémoire

- ◆ Cohérence cache/mémoire en écriture, 2 techniques
 - ◆ Écriture simultanée (*write-through*)
 - ◆ Quand écrit un mot d'une ligne du cache, on écrit simultanément le mot de la même ligne en mémoire centrale
 - ◆ Écriture plus longue car accès mémoire centrale
 - ◆ Écriture différée ou réécriture (*write-back*)
 - ◆ On n'écrit le mot que dans la ligne du cache
 - ◆ Quand cette ligne est supprimée du cache, on l'écrit en mémoire centrale

Cohérence données cache/mémoire

- ◆ Intérêt du write-back
 - ◆ Limitation des écritures en mémoire centrale
- ◆ Inconvénient du write-back
 - ◆ Problème si d'autres éléments accèdent à la mémoire en écriture
 - ◆ Périphériques en mode DMA (Direct Memory Access)
 - ◆ Multi-processeurs avec mémoire commune
 - ◆ Processeurs multi-cores
 - ◆ Nécessite alors des algorithmes supplémentaires pour gérer la cohérence

Correspondance lignes cache/mémoire

- ◆ Mémoire cache
 - ◆ Contient des lignes de mots de la mémoire centrale
- ◆ Trois méthodes pour gérer la correspondance entre une ligne dans le cache et une ligne de la mémoire centrale
 - ◆ Correspondance directe
 - ◆ Correspondance associative totale
 - ◆ Correspondance associative par ensemble
- ◆ Exemples avec
 - ◆ Lignes de 32 octets
 - ◆ Mémoire cache de 512 Ko : 16384 lignes
 - ◆ Mémoire centrale de 128 Mo : doit être gérée via les 512 Ko de cache et ses 16384 lignes

Correspondance lignes cache/mémoire

- ◆ Correspondance directe (direct mapped)
 - ◆ L lignes en cache
 - ◆ La ligne d'adresse j en mémoire centrale sera gérée par la ligne i en cache avec
 - ◆ $i = j \bmod L$
 - ◆ A partir de l'adresse d'une ligne en mémoire on sait directement dans quelle ligne du cache elle doit se trouver
- ◆ Exemple
 - ◆ Chaque ligne du cache correspond à
 - ◆ $128 \times 1024 \times 1024 / 16384 = 8192$ octets = 256 lignes
 - ◆ Une ligne i du cache contiendra à un instant donné une des 256 lignes j de la mémoire tel que $i = j \bmod 16384$
 - ◆ La ligne 56871 de la mémoire centrale ne peut être placée que dans la ligne 7719 du cache

Correspondance lignes cache/mémoire

- ◆ Correspondance directe (direct mapped)
 - ◆ Avantage
 - ◆ On sait immédiatement où aller chercher la ligne
 - ◆ Accès très rapide à la ligne (latence d'1 cycle d'horloge)
 - ◆ Inconvénient
 - ◆ Devra parfois décharger et charger souvent les mêmes lignes alors que d'autres lignes sont peu accédées
 - ◆ Peu efficace en pratique
 - ◆ Taux de succès entre 60 et 80%

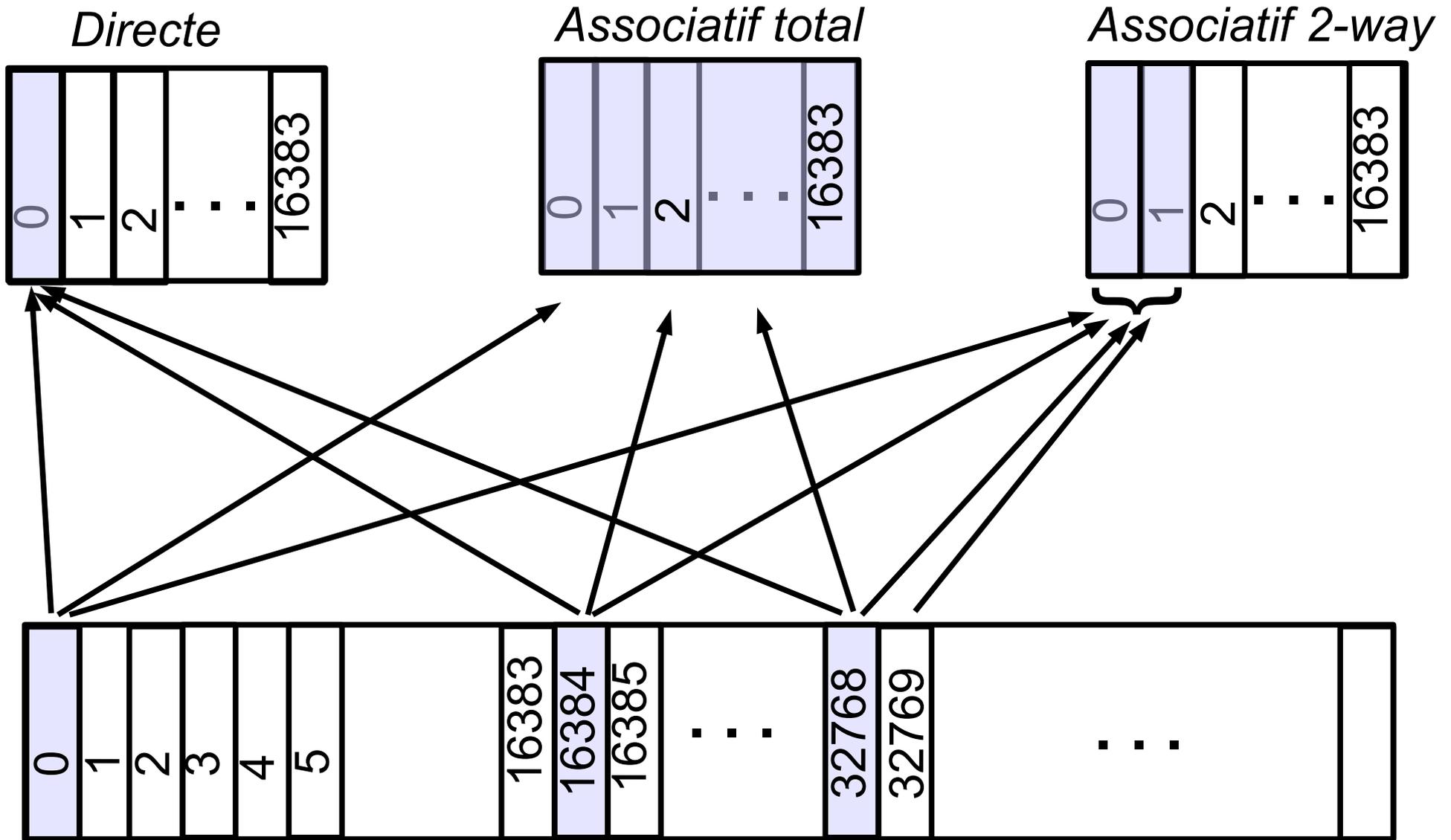
Correspondance lignes cache/mémoire

- ◆ Correspondance associative totale (fully associative)
- ◆ Une ligne de la mémoire centrale peut être placée dans n'importe quelle ligne de la mémoire cache
- ◆ Avantages
 - ◆ Très souple et efficace pour gérer les lignes de manière optimale en terme de succès d'accès
- ◆ Inconvénient
 - ◆ Doit au pire parcourir toutes les lignes du cache pour savoir si la ligne cherchée s'y trouve ou pas
- ◆ Au final
 - ◆ Taux de succès de l'ordre de 90 à 95%
 - ◆ Mais temps d'accès de l'ordre de $(L / 2)$ cycles d'horloge

Correspondance lignes cache/mémoire

- ◆ Correspondance associative par ensemble (N-way associative)
 - ◆ Solution intermédiaire aux 2 précédentes
 - ◆ On regroupe les lignes du cache en ensembles de N lignes
 - ◆ On applique le modulo pour trouver la ligne cible dans le cache mais peut placer la ligne dans n'importe quelle ligne de l'ensemble contenant cette ligne cible
- ◆ Avantages
 - ◆ Plus souple et efficace que la correspondance directe
 - ◆ Temps d'accès court = 2.5 cycles pour $N = 4$
 - ◆ Taux de succès : 80% à 90%
- ◆ Méthode utilisée en pratique car meilleur compromis

Correspondance lignes cache/mémoire



Remplacement des lignes

- ◆ Le cache contient une partie des lignes de la mémoire centrale
 - ◆ Si lecture ou écriture par le CPU dans une ligne qui n'est pas dans le cache
 - ◆ On doit charger en cache cette ligne
 - ◆ Nécessité d'enlever des lignes pour y mettre d'autres lignes
- ◆ 3 méthodes pour choisir la ligne à remplacer
 - ◆ Aléatoire
 - ◆ La plus ancienne
 - ◆ La moins utilisée

Remplacement des lignes

- ◆ Remplacement aléatoire : une ligne choisie au hasard
 - ◆ Simple à mettre en œuvre
 - ◆ Peu efficace car peut supprimer des lignes très accédées
- ◆ Remplacement de la plus ancienne
 - ◆ LRU : Last Recently Used
 - ◆ Nécessite des compteurs associés aux lignes
- ◆ Remplacement de la moins utilisée
 - ◆ LFU : Least Frequently Used
 - ◆ Nécessite également des compteurs

Remplacement des lignes

- ◆ Avec correspondance directe
 - ◆ Pas besoin de choisir : on ne peut forcément remplacer qu'une seule ligne
- ◆ Avec les 2 autres types de correspondance
 - ◆ Pourra utiliser une des 3 méthodes de remplacement
- ◆ Avec une associativité par ensemble, on peut avoir une approche intermédiaire simple (si $N > 2$) souvent utilisée en pratique
 - ◆ Entre remplacement aléatoire et LRU : pseudo-LRU
 - ◆ Dans chaque ensemble de N lignes du cache, on marque la ligne dernièrement accédée
 - ◆ Un bit suffit au lieu de compteurs
 - ◆ Quand besoin de remplacer une ligne dans l'ensemble, on prend au hasard une des $N - 1$ lignes non marquées⁵²

Conclusion sur mémoire cache

- ◆ Performances globales de la mémoire cache dépendent de plusieurs facteurs corrélés
 - ◆ Taille de la mémoire cache et organisation des niveaux
 - ◆ Méthode d'association des lignes entre mémoire centrale et cache
 - ◆ Rapidité d'accès à la bonne ligne dans le cache
 - ◆ Méthode de remplacement des lignes
 - ◆ Algorithmes de *pre-fetching*
 - ◆ Chargement en avance dans le cache de données/instructions dont le processeur devrait avoir besoin
- ◆ But : un taux de succès global qui doit être le plus élevé possible tout en assurant un temps d'accès bas
- ◆ Généralement autour de 90% de nos jours