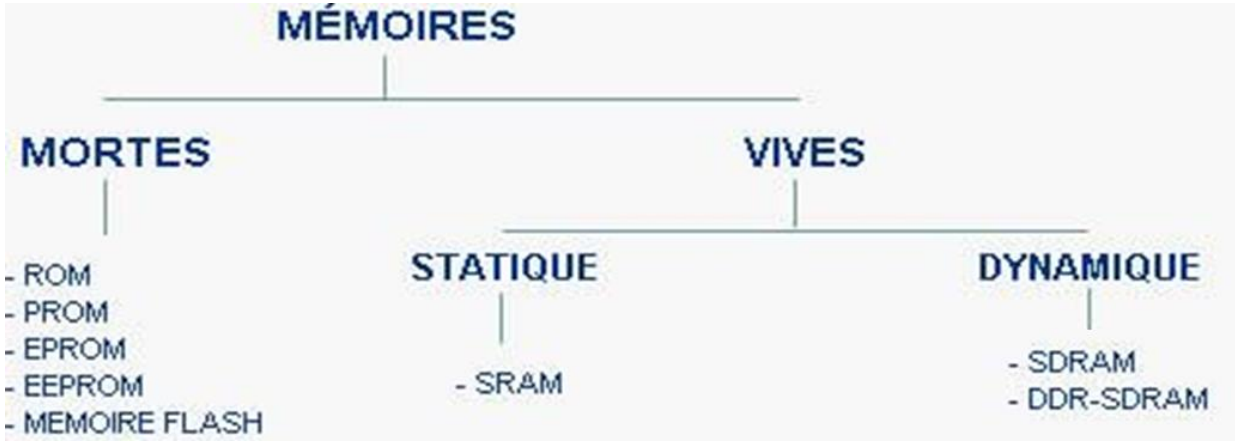
Interfaçage avec la mémoire et les E/S

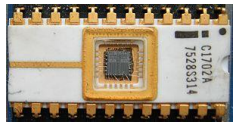
La mémoire

- Deux principaux types
  - Mémoire vive
    - Mémoire **volatile**: son contenu est perdu si elle n'est plus alimentée
  - Mémoire morte
    - Originalement, ce terme était utilisé pour une mémoire **non volatile** dont le contenu était fixé lors de sa programmation et ne pouvait plus être modifié d'où le terme « morte » (Read Only en anglais)
    - Avec les évolutions technologiques, ce terme est utilisé pour toute mémoire non volatile même si son contenu est modifiable



La mémoire

- Mémoire morte: Read Only Memory (ROM)
  - ROM : le contenu de la mémoire est créé à la fabrication
  - PROM (Programmable ROM): programmée qu'une seule fois par l'utilisateur. Une fois programmée, elle devient une mémoire morte (ROM).
  - EPROM (Erasable PROM): Pour effacer la mémoire EPROM, il faut la retirer du circuit et la soumettre à un rayonnement UV. La programmation de l'EPROM nécessite de la positionner physiquement sur un appareil dédié à cette opération.

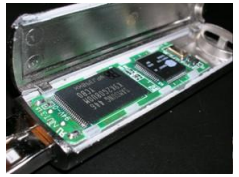


La mémoire

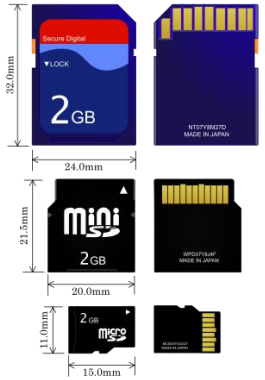
- Mémoire morte: Read Only Memory (ROM)

  - EEPROM (Electrically Erasable PROM): peut être effacée électriquement, sans qu'il soit nécessaire de la retirer de l'appareil qui l'utilise.

  - FLASH: C’est un type d’EEPROM qui permet la modification de plusieurs espaces mémoires en une seule opération. La mémoire flash est donc plus rapide lorsque le système doit écrire à plusieurs endroits en même temps.



USB



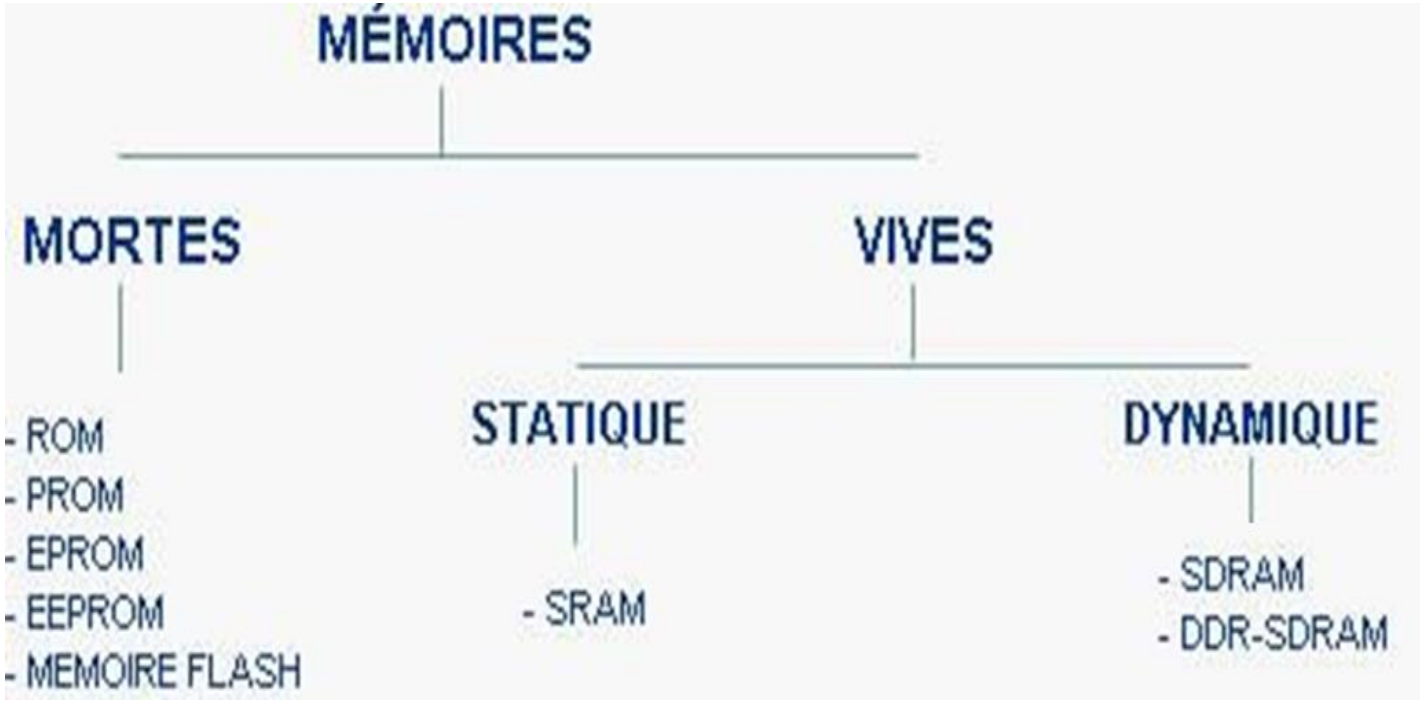
SD

Compact Flash

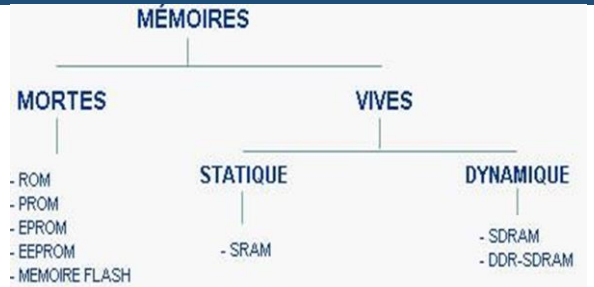
Disque SSD

La mémoire

- Mémoire vive: RAM: Random Access Memory
  - Le terme Random-Access dans l'appellation anglaise vient du fait que cette mémoire permet un accès direct à la donnée par opposition aux mémoires séquentielles



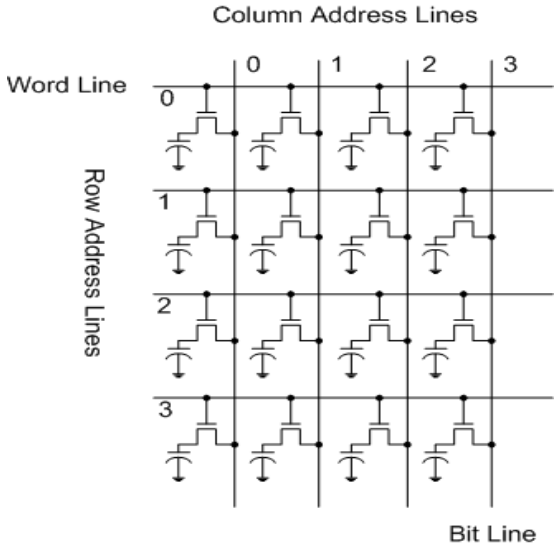
La mémoire



- Mémoire vive: RAM: Random Access Memory

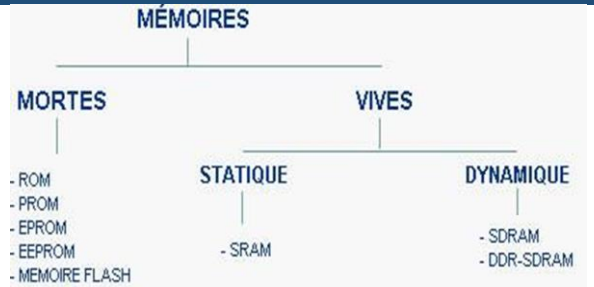
1) Dynamique

- DRAM: un condensateur et un transistor pour un bit → peu de ressources pour un bit → on peut avoir des mémoires DRAM à une densité élevée de petites tailles
  - Utilisation d'un condensateur → l'information disparaît à moins que la charge des condensateurs ne soit rafraichie avec une période de quelques ms ➔ Mémoire dynamique
  - A besoin d'être rafraichie périodiquement sinon elle perd son contenu



- Plus lente que la mémoire SRAM (statique)
  - Temps nécessaire pour charger et décharger un condensateur
  - Temps de rafraichissement des condensateurs

La mémoire



- Mémoire vive: RAM: Random Access Memory

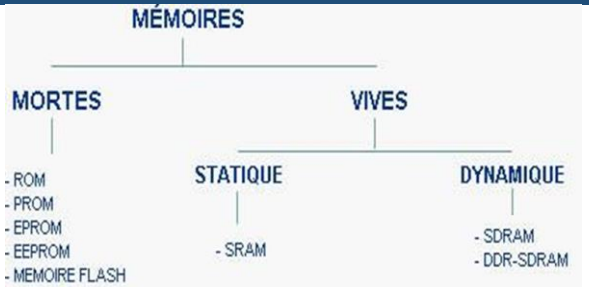
1) Dynamique

  - SDRAM ou Synchronous D*RAM :* interface de communication synchrone.
    - Jusqu'à son apparition, les mémoires DRAM étaient asynchrones (ne sont pas synchronisées avec le signal de l'horloge du bus)

  - DDR-SDRAM ou DDR (Double Data Rate): une meilleure bande passante que l'ordinaire SDRAM en transférant les données à la fois sur le front montant et sur le front descendant des impulsions d'horloge, (une vitesse double en lecture et en écriture)

  - DDR2, DDR3 et DDR4: la différence est la fréquence qui est de plus en plus grande

La mémoire

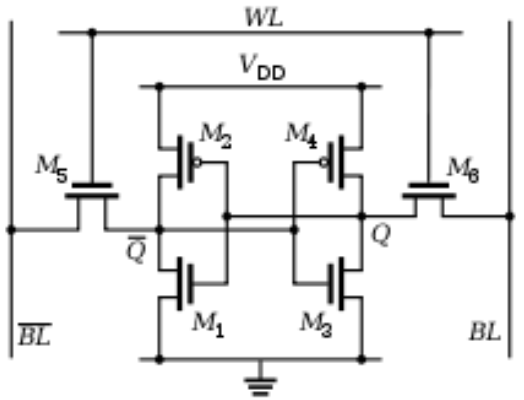


- Mémoire vive: RAM: Random Access Memory

2) Statique

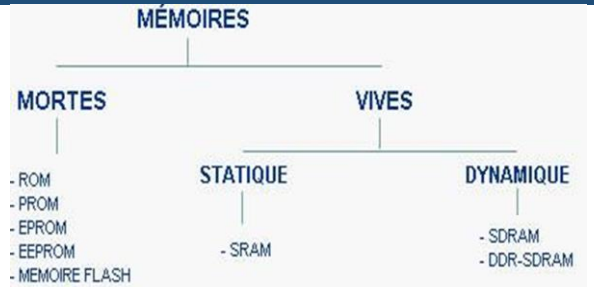
- SRAM: 6 transistors pour 1 bit
  - Moins de densité que les mémoires dynamiques (pour une même surface, la SRAM contient moins de cases mémoire)
  - Elle n'a pas besoin de rafraîchir périodiquement son contenu

➔ mémoire statique



La mémoire

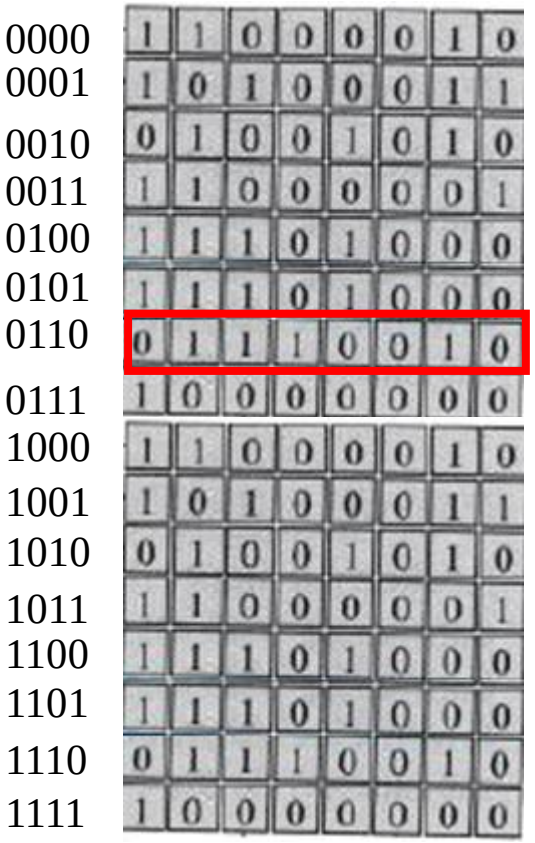
- Mémoire vive: RAM: Random Access Memory

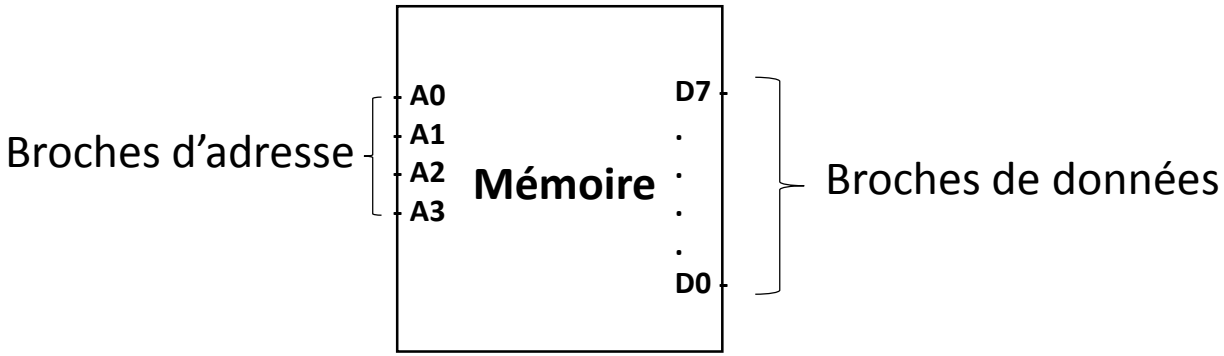


Mémoire dynamique	**Mémoire statique**
Grande densité d'intégration	Petite densité d'intégration
Bon marché	Chère
Lente	Rapide
Besoin de rafraîchissement	Pas de rafraichissement nécessaire

La mémoire

- Les pins d'adresse et de données
  - Exemple: On a 16 mots dans la mémoire
    - On a 16 emplacements mémoires différents → 16 adresses différentes
    - On a besoin de 4 bits pour écrire 16 adresses différentes (de 0 à 15)= log2(16)





Si le processeur veut lire le contenu de l'adresse 6, il écrit la valeur 0110 sur les broches d'adresse, en réponse la mémoire écrit le contenu de cette case (01110010) sur les broches de données

La mémoire

- Les pins d'adresse et de données
  - Exemple: Si on a une mémoire de 2Go avec des mots mémoires de 16 bits.

    1) Combien de mots mémoire cette mémoire contient-elle?

    2) Combien de pins d'adresse cette mémoire contient-elle?

La mémoire

- Les pins d'adresse et de données
  - Exemple: Si on a une mémoire de 2Go avec des mots mémoires de 16 bits, combien de pins d'adresse cette mémoire contient-elles?

1) Combien de mots mémoire cette mémoire contient-elle?

   Nombre de mots mémoire= capacité /taille de mot mémoire=2Go/16bits= 2Go/2octet= 2^{30}

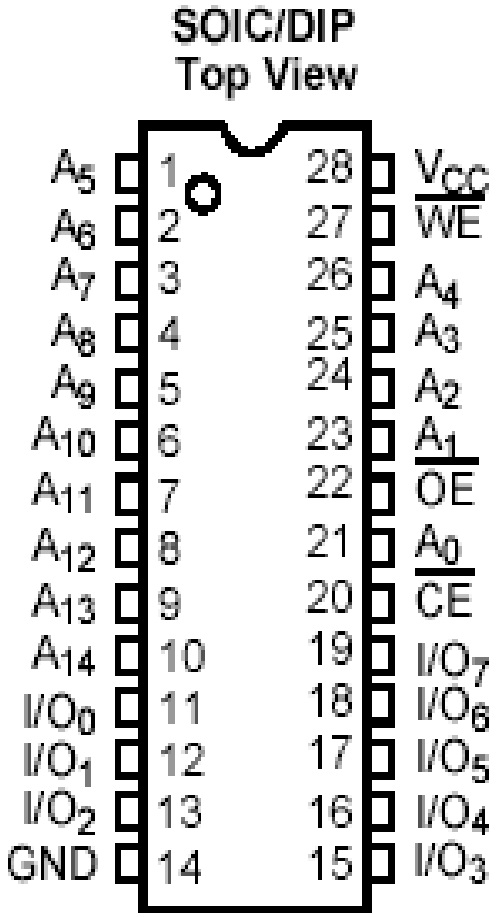
2) Combien de pins d'adresse cette mémoire contient-elle?

   Nombre de pins adresse= log2(nombre de mots mémoire)

   ➔Nombre de pins adresse= log2(2^{30})=**30**

Communication processeur/mémoire

- Interface d'une mémoire

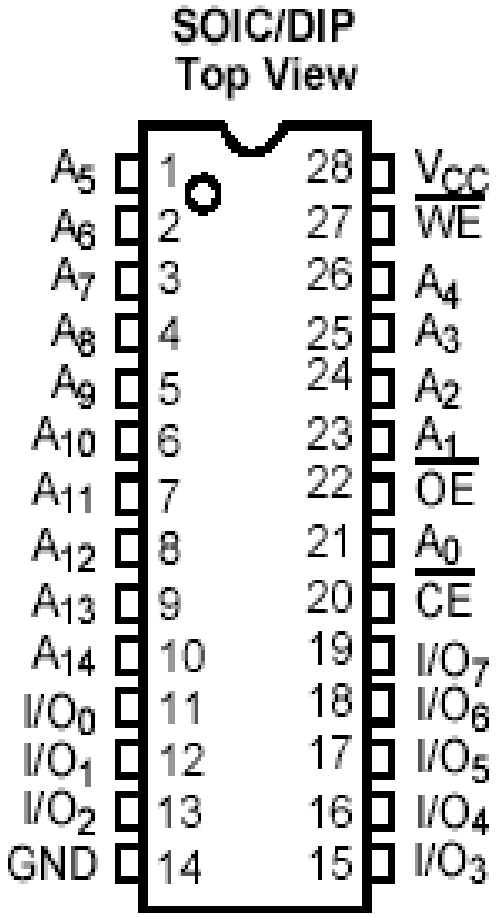


- Les pins d'adresses qui seront connectés au bus d'adresse (A0 à A14)
- Les pins d'entrées/sorties qui sont connectés au bus de données (I/O_0 à I/O_7)
- **WE**: Write Enable qui active l'écriture
- **OE**: Output Enable qui active la lecture
- **CE/CS**: Chip Enable/ Chip Select qui active la mémoire (si $\overline{CE}$=1 la mémoire est inactive et on ne peut faire ni lecture ni écriture),
- VCC et GND pour l'alimentation et la masse

- **Quelle est la capacité de cette mémoire?**

Communication processeur/mémoire

- Interface d'une mémoire

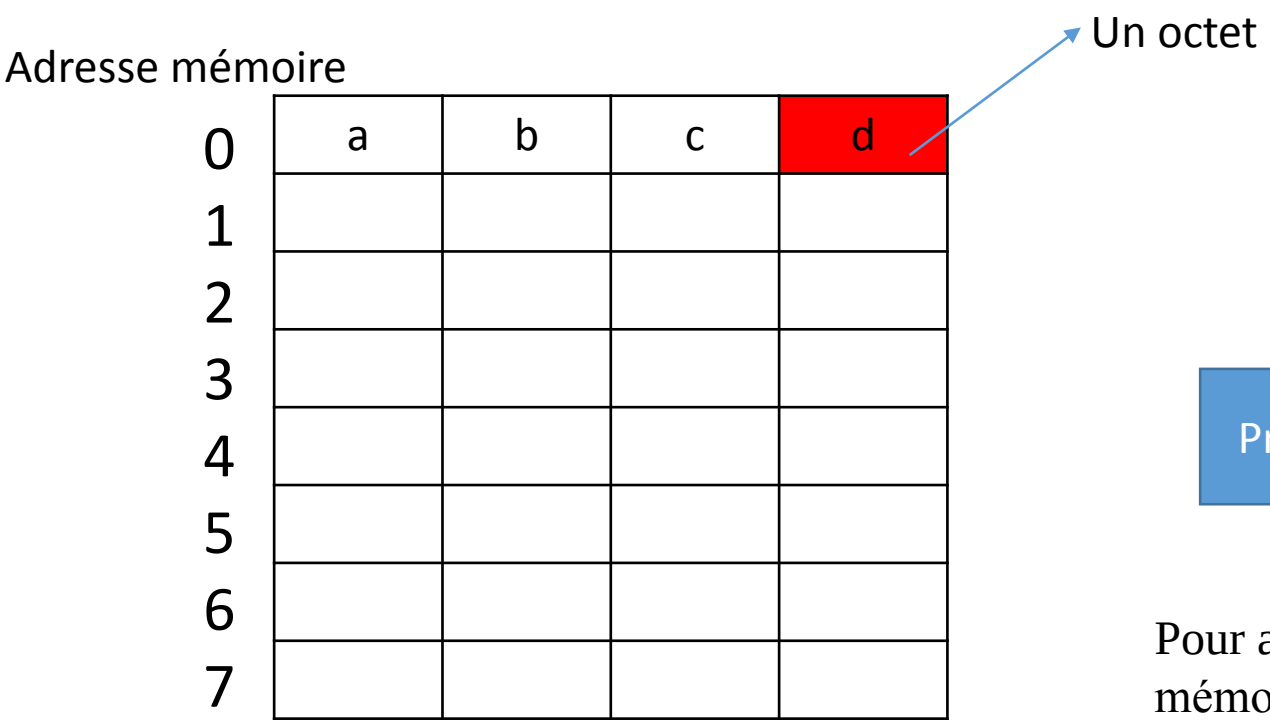


- **Quelle est la capacité de cette mémoire?**
  - 15 pins d'adresses → 2^{15} adresses différentes= 32K mots mémoire
  - 8 pins de données → 1 mot mémoire= 8 bits
  - La capacité= 32K x 8 bits =32Ko

Accès mémoire

- La plupart des processeurs utilisent l'adressage de l'ordre de l'octet (byte-addressing: chaque octet a sa propre adresse)
- Par contre la mémoire est adressable de l'ordre de mot mémoire: chaque mot mémoire a une adresse

- Exemple: une mémoire avec des mots mémoires de 4 octets




```
char a, b, c, d;

……
d=10;
```

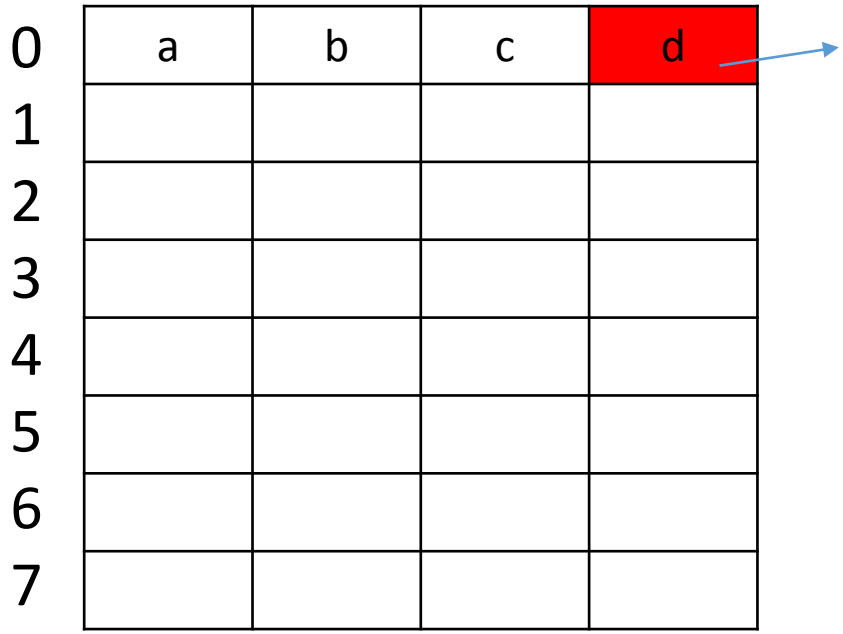
Pour accéder à la variable d, le processeur envoie au contrôleur mémoire l'adresse 3, le contrôleur fait la correspondance et déduit qu'il s'agit du mot mémoire d'adresse 0

Accès mémoire

- L'accès mémoire se fait en écrivant/lisant **un mot mémoire entier**

  - Exemple: une mémoire avec des mots de 4 octets
    - Si le processeur veut modifier le contenu de la variable d, il faut suivre les étapes suivantes:
      1) Le contrôleur mémoire lit tout le mot mémoire (a,b,c,d) d'adresse 0 et le met dans un buffer
      2) Le contrôleur mémoire modifie le quatrième octet de ce buffer
      3) Le contrôleur mémoire écrit dans la mémoire le nouveau contenu du buffer

Adresse mémoire



Mémoire avec des mots de 32 bits

```
char a, b, c, d;

……
d=10;
```

Accès mémoire

- Attention:
  - 1 **mot processeur** correspond à la taille des données que le processeur peut manipuler en une seule instruction machine (la taille de ses registres génériques)
    - Exemple: un processeur 32-bits a un mot processeur de 32 bits
  - 1 **mot mémoire** correspond à la taille des données que la mémoire peut lire ou écrire à la fois
    - Exemple: si un processeur 64-bits veut lire un mot de 64 bits à partir d'une mémoire 32 bits, cette opération se fait en deux temps
  - Un processeur « **byte-addressable** »: c'est le type du processeur le plus commun dans les ordinateurs
    - Chaque adresse envoyée par le processeur correspond à un octet de la mémoire
    - Pour cette raison, un processeur 32-bits ne peut pas adresser plus de 4GB indépendamment de la taille du mot mémoire

Accès mémoire

- Exercice:
  - Soit le code suivant :

```
#include<stdio.h>
void main(){
int T[5]={1,2,3,4,5};
int i;
for(i=0;i<5;i++)
printf("adresse de T[%d]=%x\n",i,&T[i]);
}
```

  - Sachant que la première ligne retournée par ce code est : adresse deT[0]=dfbfb3d0 et qu'un entier sur cette machine prend 32 bits, donner le reste des lignes retournées par ce code.

Accès mémoire

- Solution:
  - Soit le code suivant :

```
#include<stdio.h>
void main(){
int T[5]={1,2,3,4,5};
int i;
for(i=0;i<5;i++)
printf("adresse de T[%d]=%x\n",i,&T[i]);
}
```

- Sachant que la première ligne retournée par ce code est : adresse deT[0]=dfbfb3d0 et qu'un entier sur cette machine prend 32 bits, donner le reste des lignes retournées par ce code.

- La première case du tableau est à l'adresse dfbfb3d0, la case suivante se trouve donc à cette adresse + 4. Le reste des lignes affichées par le programme sont :
  - adresse deT[1]=dfbfb3d4
  - adresse deT[2]=dfbfb3d8
  - adresse deT[3]=dfbfb3dc
  - adresse deT[4]=dfbfb3e0

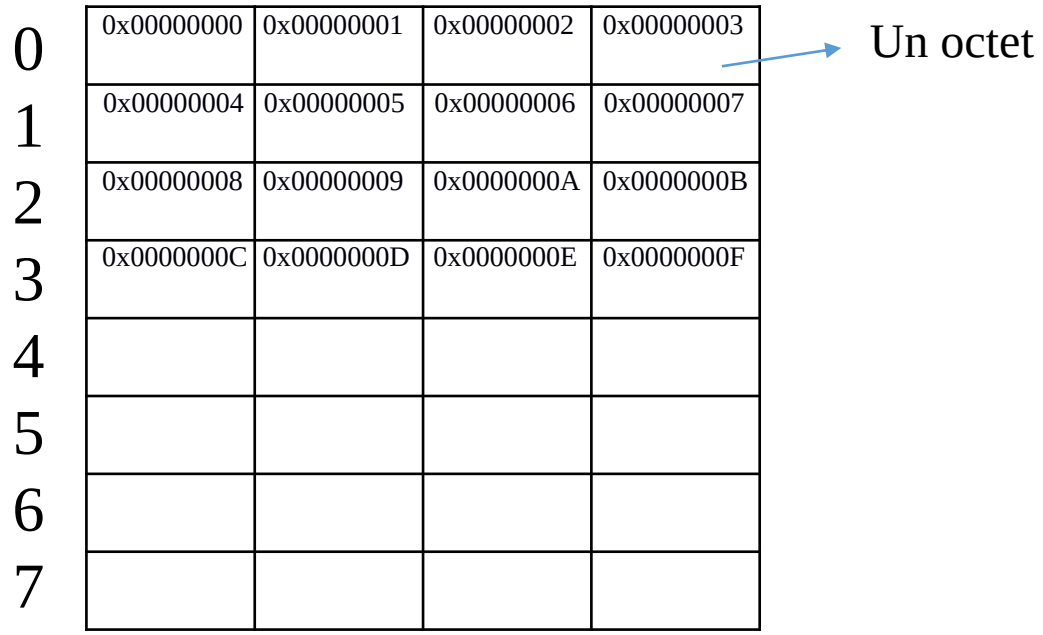
Accès mémoire

- Exercice:
  - Soit une mémoire avec des mots mémoire sur 32 bits connectée à un processeur. Le processeur veut accéder à l'octet d'adresse 0x0000000F. Quelle est l'adresse du mot mémoire qui contient cet octet ?

Accès mémoire

- Solution:
  - Soit une mémoire avec des mots mémoire sur 32 bits connectée à un processeur. Le processeur veut accéder à l'octet d'adresse 0x0000000F. Quelle est l'adresse du mot mémoire qui contient cet octet ?
  - Le processeur voit les adresses des octets, l'octet d'adresse 0x0000000F est donc le dernier octet du quatrième mémoire soit le mot mémoire d'adresse 3 (0x0000000F / 4= 0x00000003).

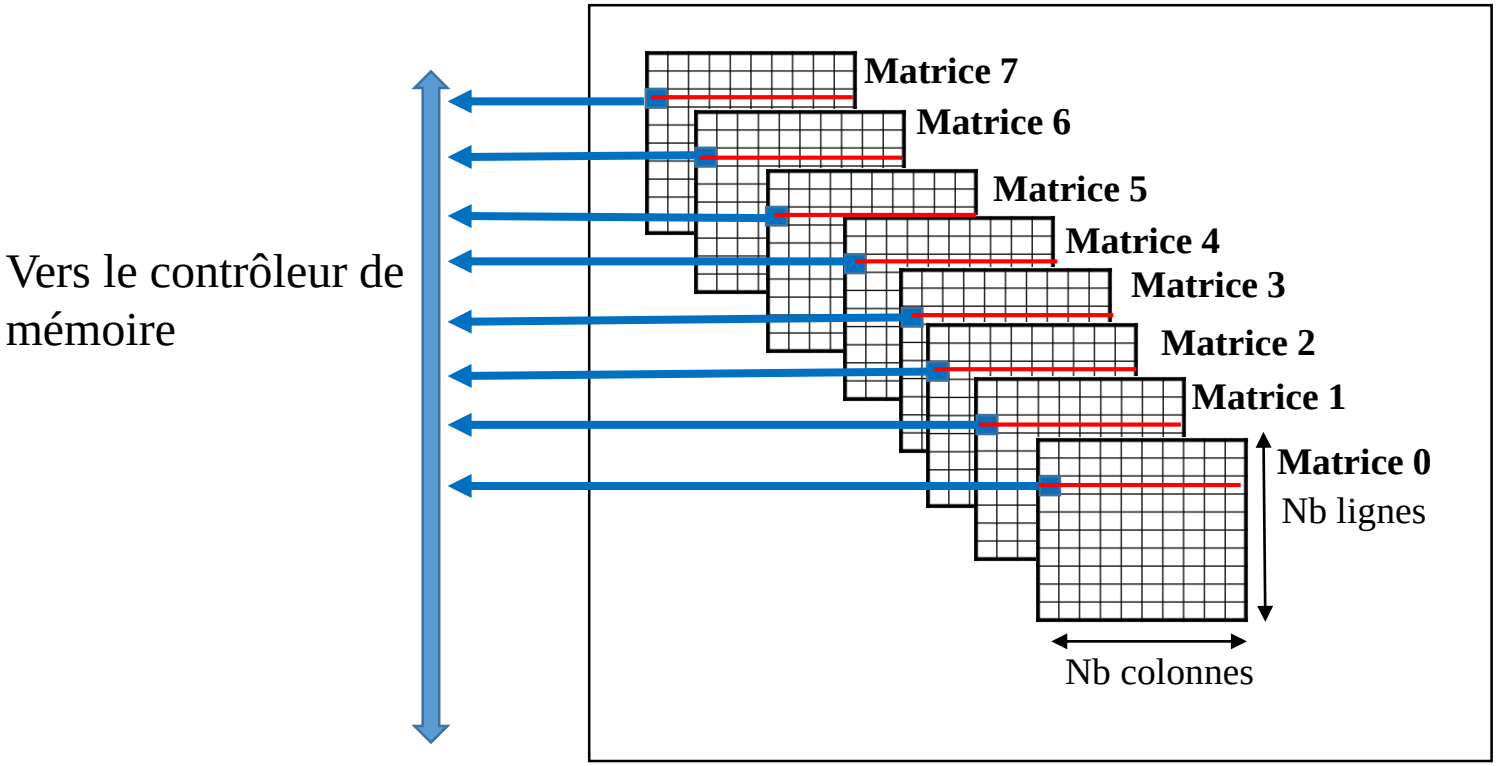
Adresse mémoire



Mémoire avec des mots de 32 bits

Accès mémoire

- La mémoire DRAM est organisée sous forme de matrices. Pour accéder à un mot mémoire, il faut spécifier une ligne puis une colonne.
- En spécifiant une ligne, elle sera activée sur toutes les matrices de la mémoire.
- En spécifiant la colonne, on active un seul bit de chaque ligne activée.
- La combinaison des bits provenant de chaque matrice constitue un mot mémoire



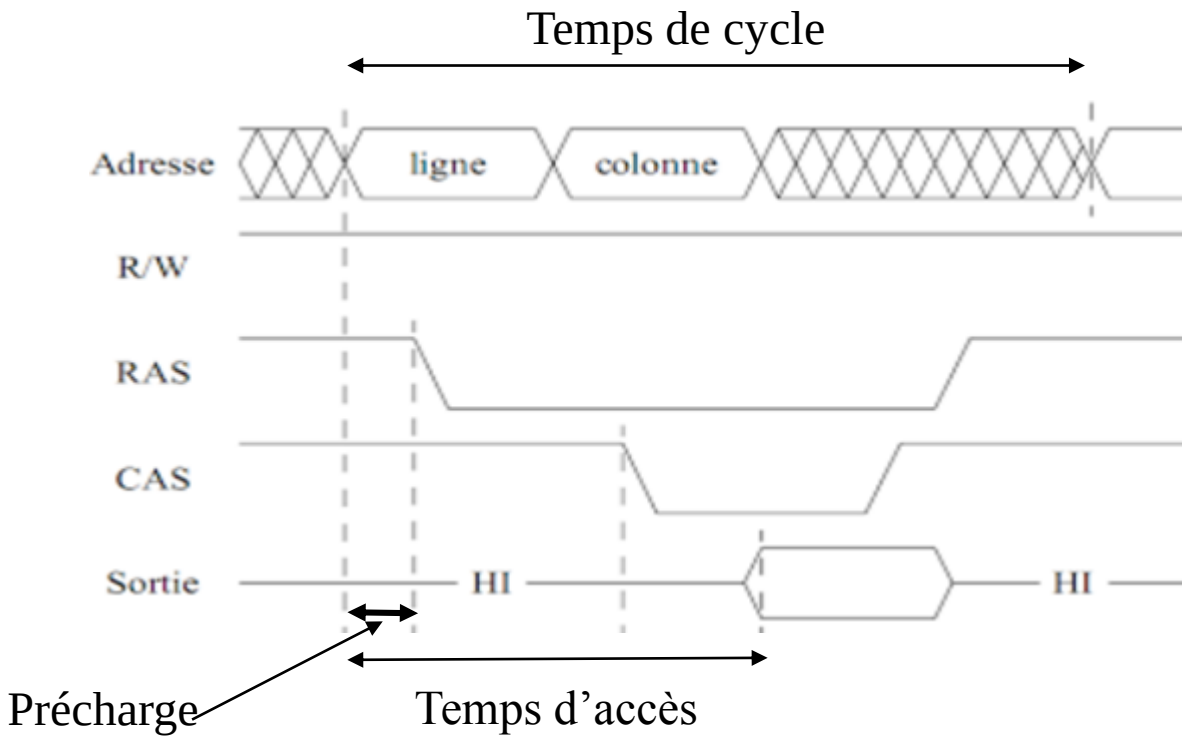
- Exemple:
  - Mémoire de 8 x 8 x 8 bits
  - Un mot mémoire= 8 bits
  - Le mot mémoire qu'on a sélectionné correspond à la ligne 2 et la colonne 0

Paramètres temporels d'une mémoire

- La mémoire est caractérisée par deux paramètres principaux:
  - Le temps d'accès: le temps qui sépare une demande de lecture de l'obtention de la donnée
  - Le temps de cycle: le temps minimum qui sépare deux demandes successives de lecture ou d'écriture

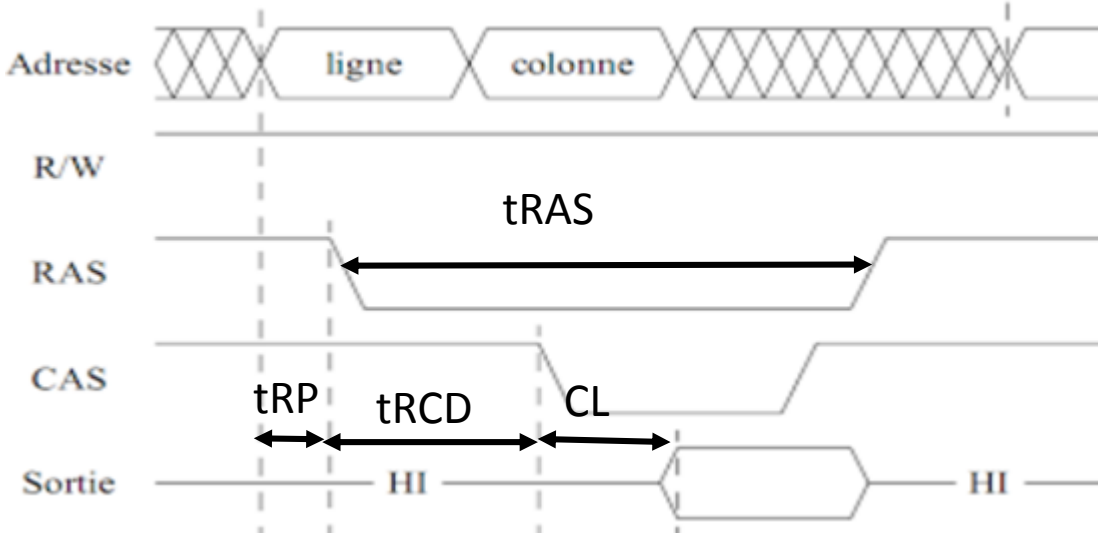
Le temps d'accès d'une RAM

- La mémoire vive doit être vue comme une matrice (lignes et colonnes).
  - La mémoire précharge la bonne ligne (Précharge), puis doit l'activer (RAS), et ensuite sélectionner la bonne colonne (CAS). Une fois ceci fait, elle peut commencer son opération de lecture ou d'écriture.



Le temps d'accès d'une RAM

- Le timing d'une RAM est généralement écrit sous forme de 4 nombres (en nombre de cycles)
  - CL – tRCD – tRP – tRAS
    - CL: CAS Latency. Il signifie Column Access Strobe Latency et c'est le timing le plus connu. C'est le temps d'accès à partir de l'activation de la colonne
    - tRCD: Il veut dire RAS to CAS Delay. Temps entre activation de ligne et activation de colonne.
    - tRP: Row Precharge Time, le temps d'attente pour pouvoir sélectionner la ligne
    - tRAS: Row Active Time, à partir du moment où une ligne devient active, on ne peut plus changer de ligne avant un certain nombre de cycles.



Le temps d'accès d'une RAM

- La latence= latence en cycles (CL) / fréquence mémoire

  Exemple: Pour la G.Skill Trident X Series 8 Go, DDR3 2400 MHz, les timings sont composés de 4 chiffres qui sont 10-12-12-31

  La fréquence 2400 MHz affichée par les constructeurs est généralement doublée (car c'est du double rate), la vraie fréquence de la mémoire est 1200 MHz

  ➔Latence en seconde = CL / fréquence
  =10/1200MHz= 8,33 ns

Relation entre la fréquence mémoire et la latence

- La latence= latence en cycles (CL) / fréquence mémoire

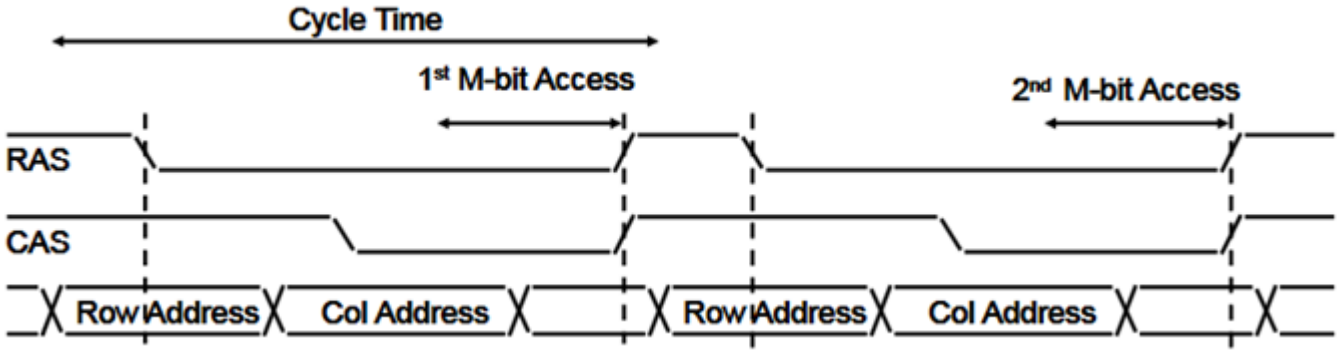
  - Exemples de mémoires DDR2:

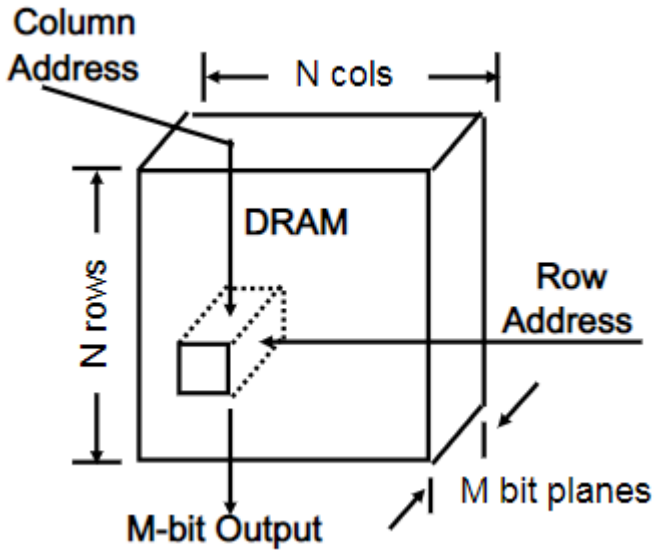
Fréquence mémoire	Fréquence commerciale (double rate)	Cycle d'horloge	Latence en cycles	Latence en s
1GHz	2GHz	1ns	10	10ns
1GHz	2GHz	1ns	18	18ns
500MHz	1GHz	2ns	8	16ns

- La latence en s est une fonction de la fréquence et de nombre de cycles → avoir une fréquence plus grande ne veut pas forcément dire une latence plus petite

Le temps d'accès d'une RAM

- Accès en mode normal

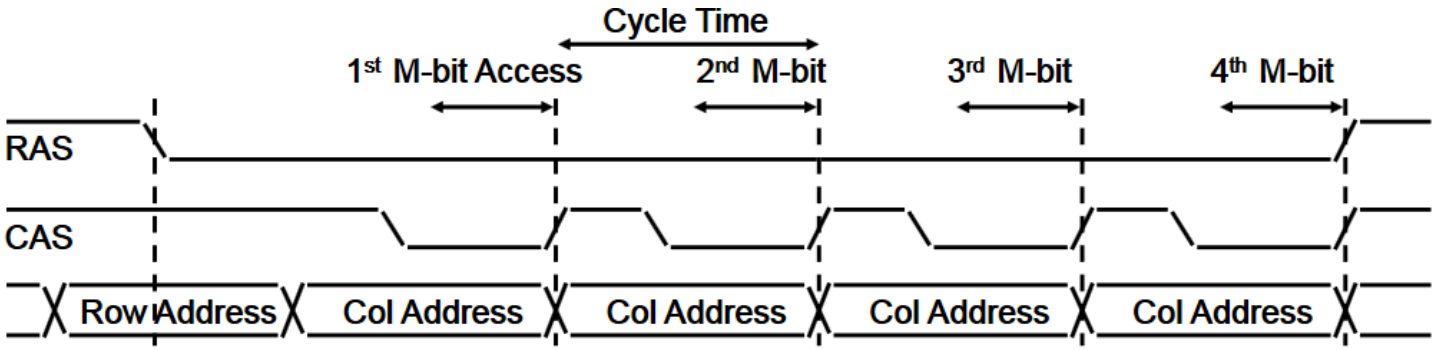


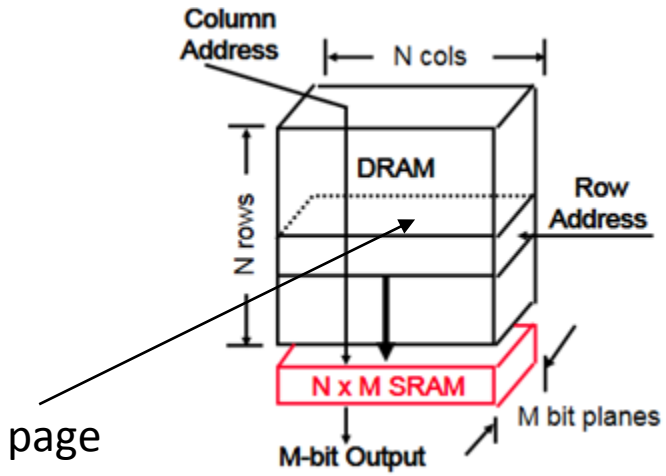


A chaque requête, on spécifie le numéro de la ligne et de la colonne

Le temps d'accès d'une RAM

- Accès en mode page





- On peut accéder à plusieurs colonnes de la même ligne plus rapidement en sélectionnant la ligne une seule fois

- Ce mode permet d'accéder rapidement à des blocs de données contiguës

Espace d'adressage d'un processeur

- C'est la capacité mémoire maximale qu'il peut adresser
- Elle est donnée par la taille de son bus d'adresse
- Les processeurs sont « byte-adressable »

➔ Un processeur ayant 16 bits d'adresse a un espace d'adressage de 2^{16} = 64KB
Un processeur ayant 32 bits d'adresse a un espace d'adressage de 2^{32} = 4GB

- L'espace d'adressage contient les adresses de tous les composants adressables:
  - Mémoires RAM
  - Mémoire BIOS ROM/flash
  - Périphériques d'E/S: clavier, souris, etc.

Espace d'adressage d'un processeur

- Les processeurs 32-bits et 64-bits
  - Les processeurs 32-bits ont généralement:
    - Des registres à 32 bits
    - Un bus de données de 32 bits
    - Un bus d'adresse de 32 bits
    - Un espace d'adressage de: 2^{32} octets=4GB
    - Ils ne peuvent pas contenir des composants dont la somme de la mémoire excède 4GB
    - **Attention**: si on met une mémoire RAM de 4 GB pour un processeur 32-bits, il y a une partie de cette mémoire qui ne sera pas utilisée, puisqu'il y a une partie des adresses de l'espace d'adressage qui doit être attribuée aux E/S

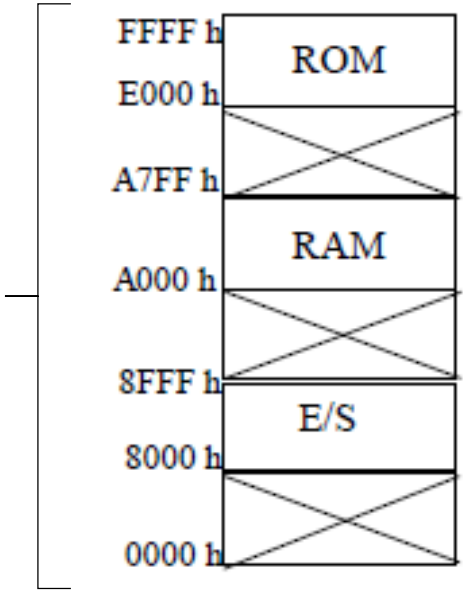
Espace d'adressage d'un processeur

- Les processeurs 32-bits et 64-bits
  - Les processeurs 64-bits ont généralement:
    - Des registres à 64 bits
    - Un bus de données de 64 bits
    - Un bus d'adresse de 64 bits
    - Un espace d'adressage de: 2^{64} octets=16 Exaoctets ≈16 millards GB

Communication processeur/mémoire

- L'espace adressable est partagé entre les mémoires et les E/S du système
- A chaque mémoire ou E/S, on affecte une plage mémoire

Espace adressable

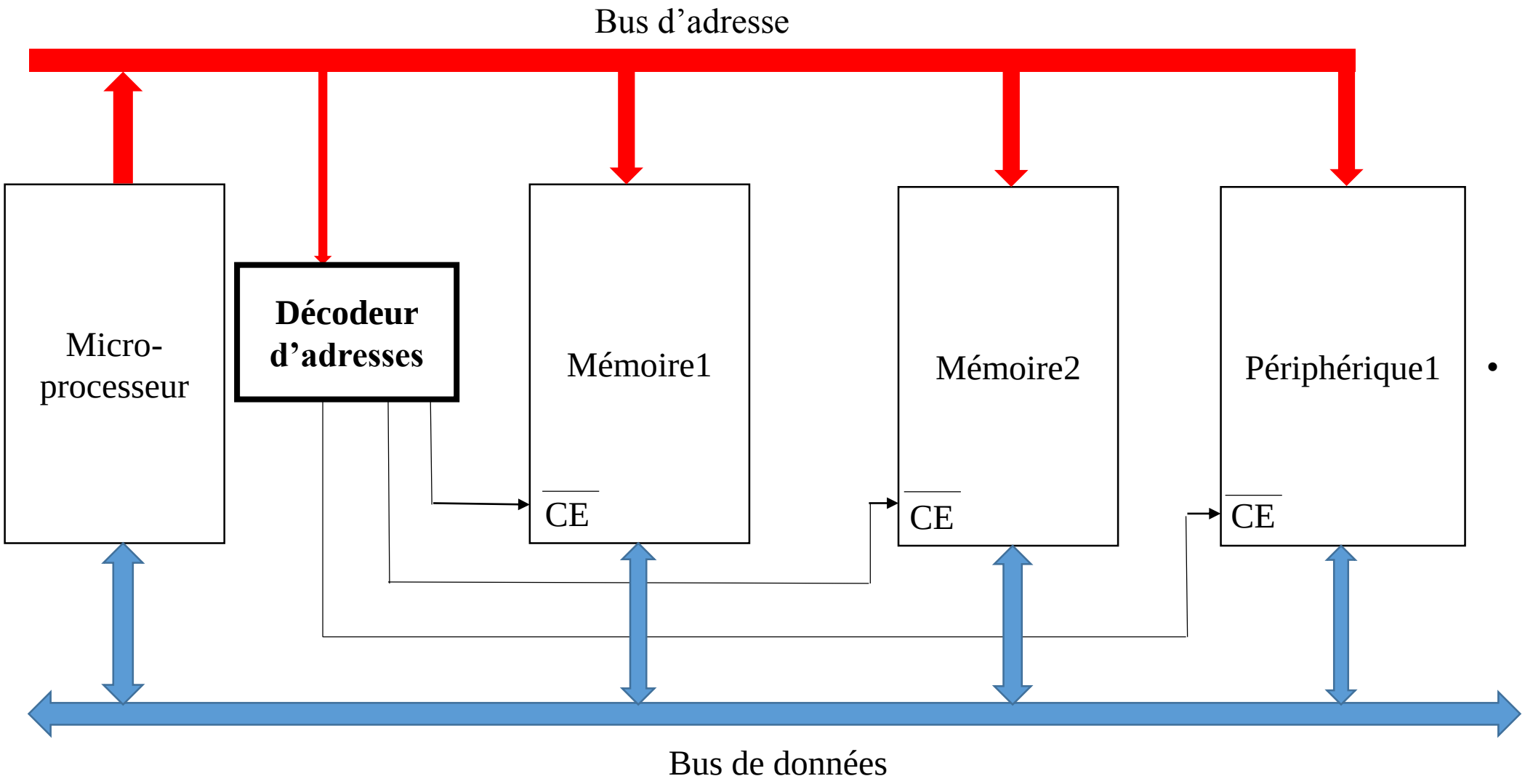


C'est un plan d'adresses géré par le décodeur d'adresses
→ Selon la valeur de l'adresse reçue sur le bus d'adresse, il détermine le composant qui contient les données se trouvant à cette adresse et il l'active

- Le processeur ne doit activer que le composant auquel il veut accéder à un moment donné
- Sinon, tous les composants vont écrire sur le bus de données → collision de données
- Pour n'activer que le composant concerné, on utilise un décodeur d'adresses

Communication processeur/mémoire

- Le décodeur d'adresses



- Le décodeur d'adresses permet de sélectionner le composant avec lequel le processeur veut communiquer en examinant une partie de l'adresse envoyée sur le bus d'adresse
- A chaque valeur de l'entrée du décodeur correspond une sortie qui n'active qu'un seul CE

Communication processeur/mémoire

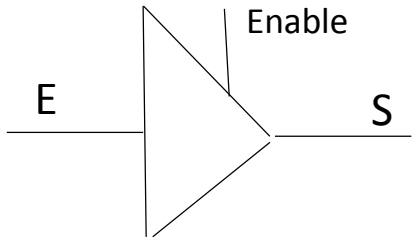
- Le décodeur d'adresse
  - Attention: le décodeur d'adresse est différent d'un décodeur mémoire
  - Le décodeur mémoire se trouve dans la mémoire et permet d'activer les lignes et les colonnes
  - Le décodeur d'adresses se trouve à l'extérieur des composants et permet d'activer un composant (mémoire/ES) parmi plusieurs

Communication processeur/mémoire

- Comment n'activer que le composant auquel le processeur veut accéder et déconnecter les autres alors qu'ils sont tous connectés au bus de données?
  - Utilisation des buffers tri-state (un circuit logique qui joue le rôle d'interrupteur ouvert ou fermé selon l'état de son entrée de contrôle)
  - Ces buffers sont utilisés en pairs pour implémenter deux chemins des données (lecture et écriture)

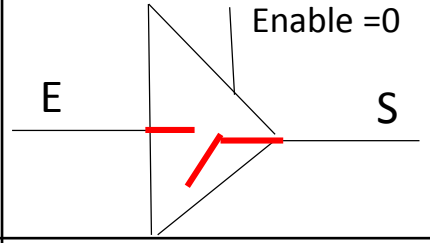
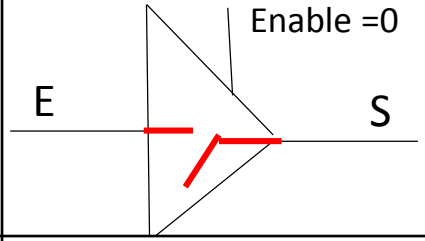
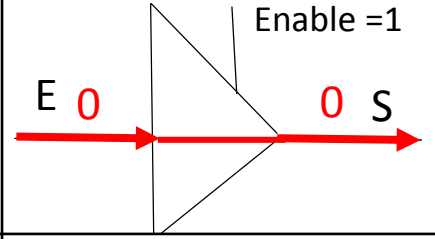
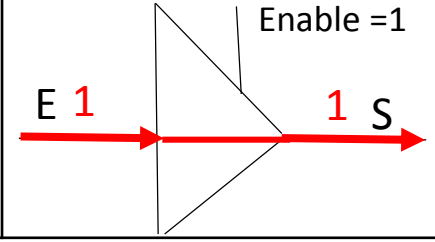
Communication processeur/mémoire

- Les tri-state



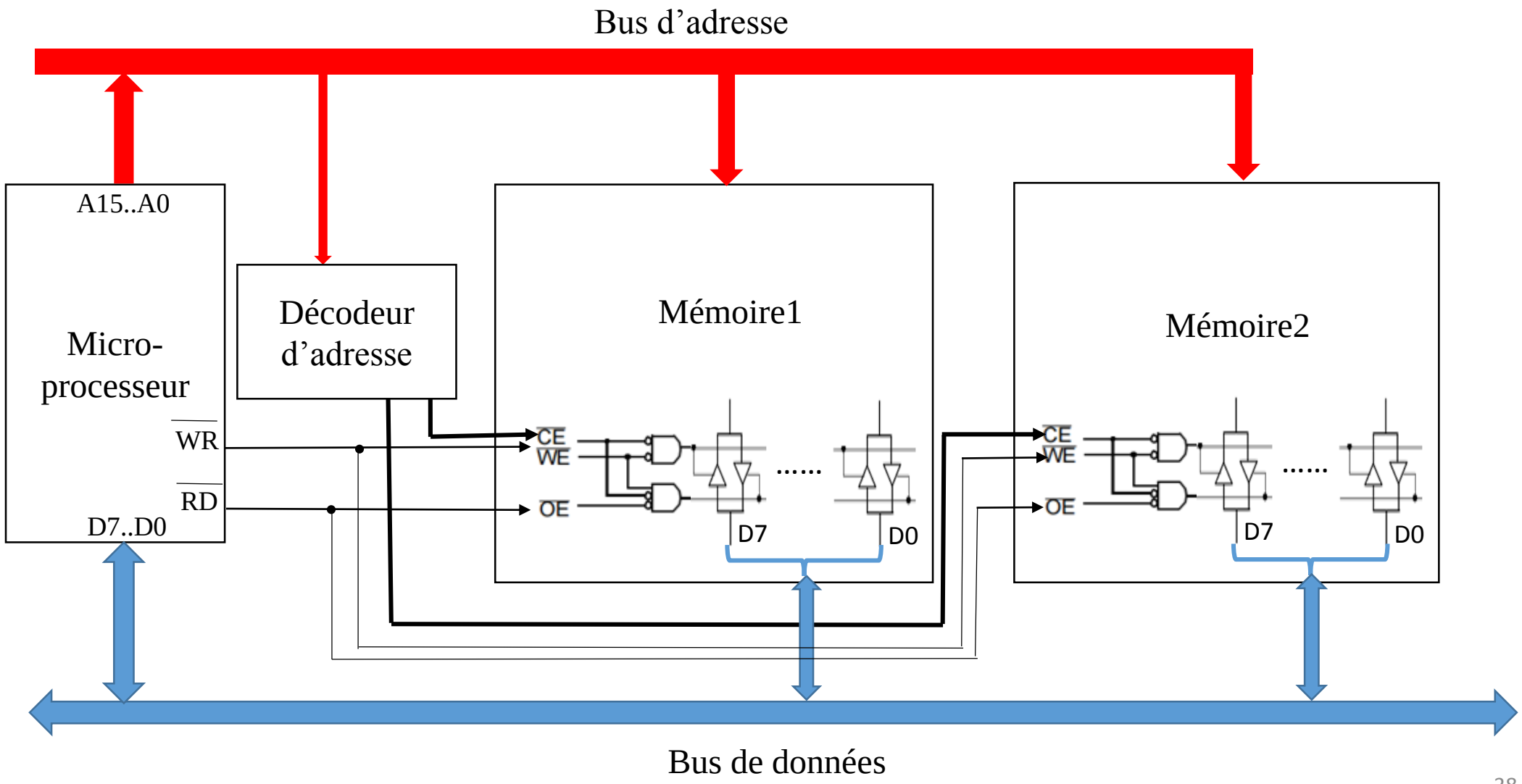
Si Enable=0 → S=Z

Si Enable=1 → S=E

Enable	Entrée	Sortie	Schéma	Equivalence
0	0	Z (haute impédance)		Circuit ouvert /résistance infinie
0	1	Z (haute impédance)		Circuit ouvert /résistance infinie
1	0	0		Circuit fermé /résistance nulle ou très faible
1	1	1		Circuit fermé /résistance nulle ou très faible

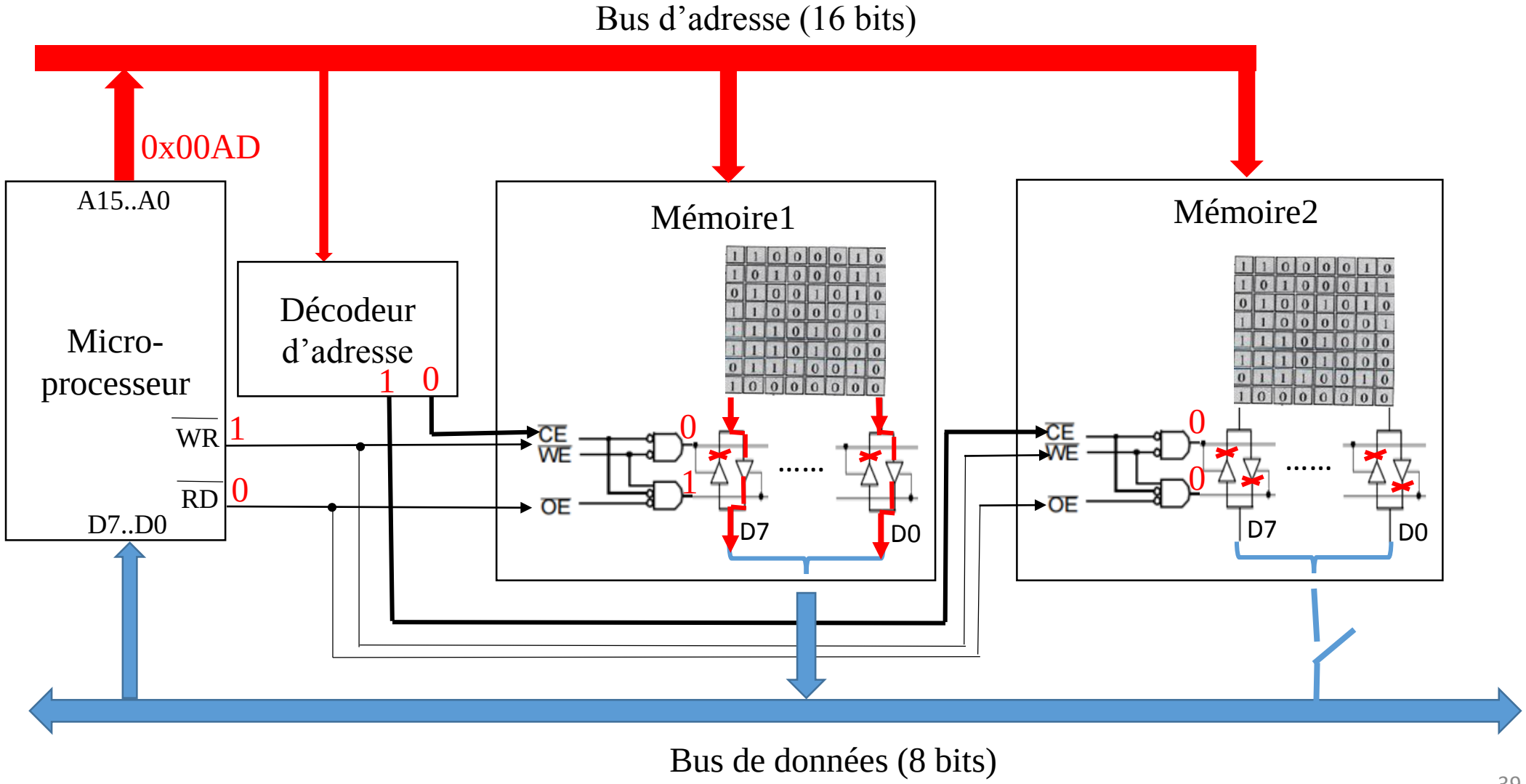
Communication processeur/mémoire

- L'utilisation des tri-states



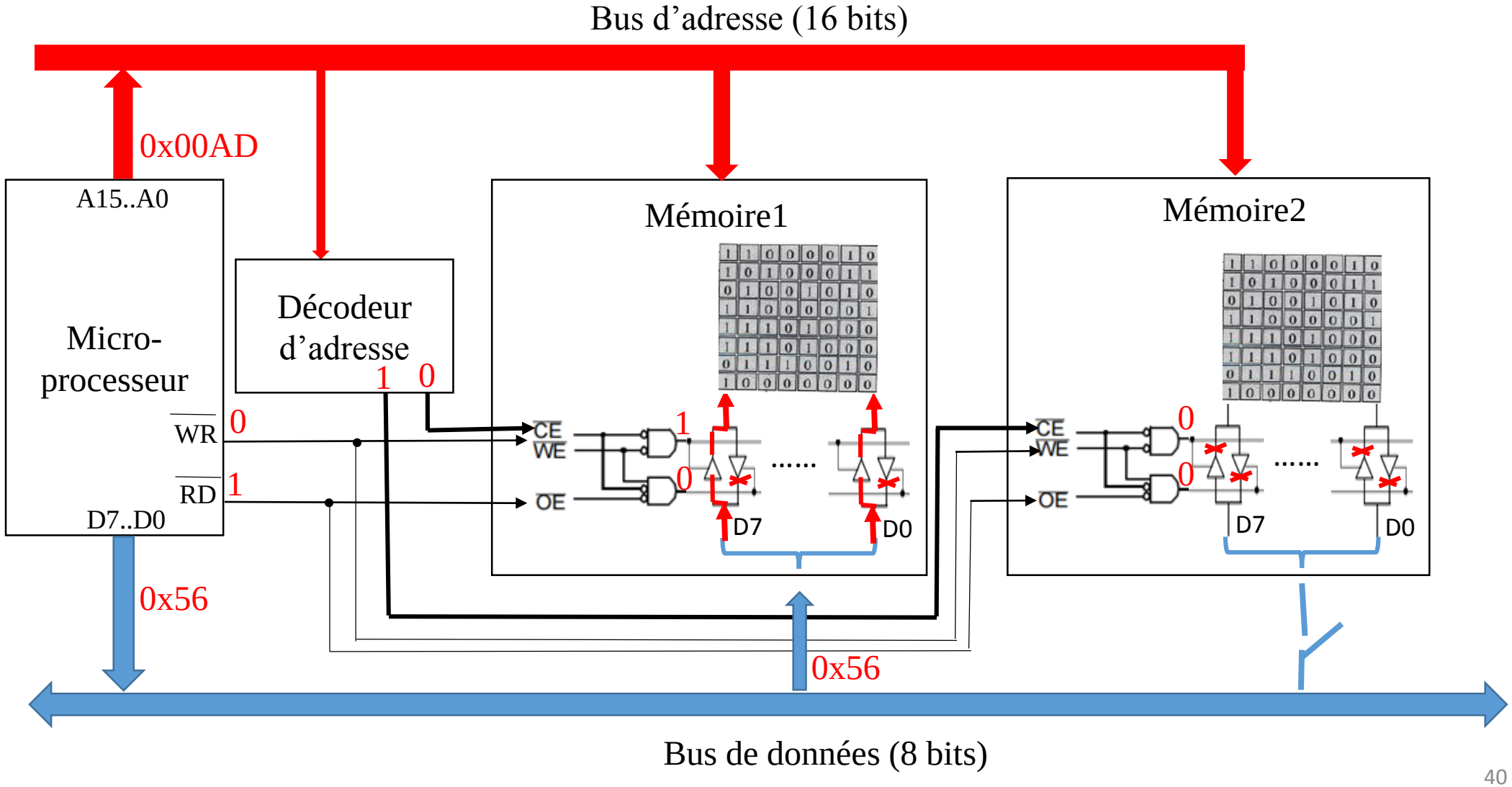
Communication processeur/mémoire

- L'utilisation des tri-states
  - Exemple: lecture du contenu de l'adresse 0x00AD qui se trouve dans la mémoire 1



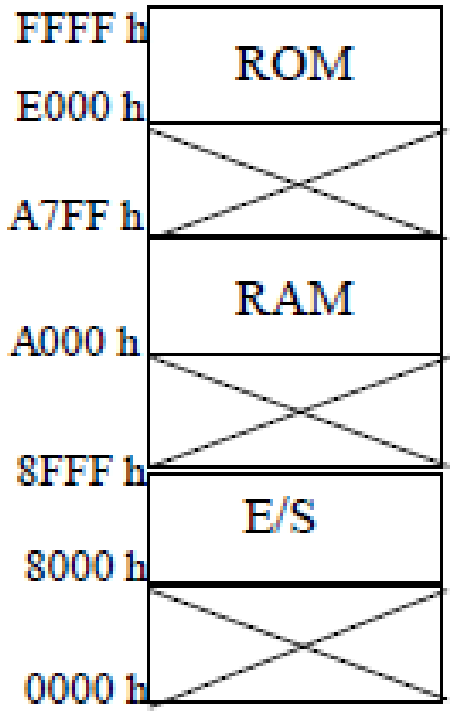
Communication processeur/mémoire

- L'utilisation des tri-states
  - Exemple: écriture de la valeur 0x56 à l'adresse 0x00AD qui se trouve dans la mémoire 1



Plan mémoire

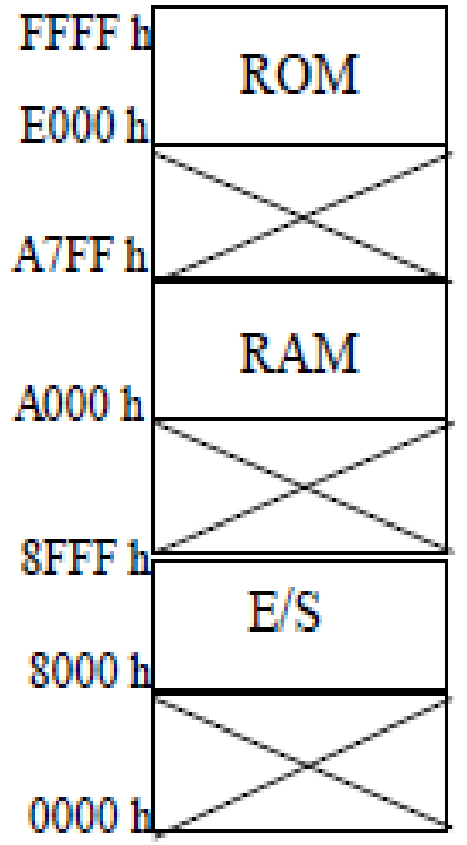
- Calcul des plages mémoires
  - Pour un processeur byte-addressable ayant 16 bits d'adresse, les adresses des composants qui y sont connectés vont de 0000h à FFFFh
  - Le processeur associe à la ROM les adresses entre E000h et FFFFh, à la RAM A000h à A7FFh et à l'E/S 8000h à 8FFFh
  - En déduire les tailles de ces composants
    - Taille de la ROM= FFFFh –E000h+1=1FFFh +1=2000h=2^{13} octets=8Ko
    - Taille de la RAM= A7FFh –A000h+1=07FFh+1=0800h=2^{11} octets=2Ko
    - Taille de la plage E/S= 8FFFh –8000h+1=0FFFh+1=1000h=2^{12} octets=4Ko



Plan mémoire

- Déterminer les équations du décodeur d'adresse

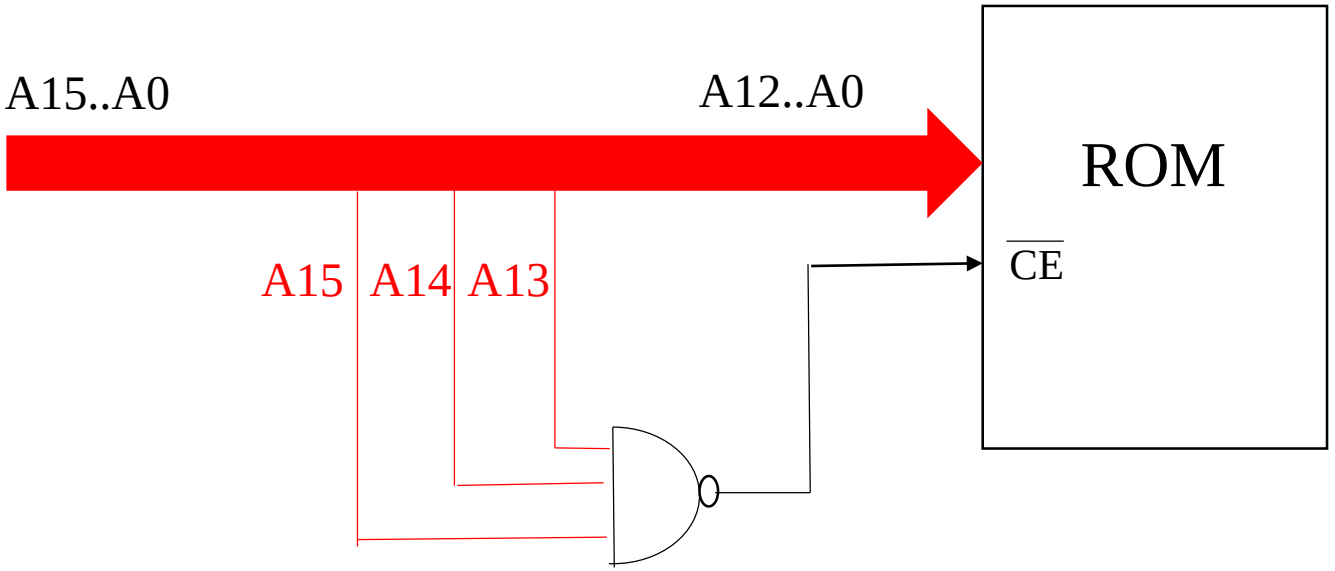
		Bus d'adresse															
		A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
ROM	fin	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	début	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
LIBRE	fin	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
	début	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
RAM	fin	1	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1
	début	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
LIBRE	fin	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
	début	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
E/S	fin	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
	début	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LIBRE	fin	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	début	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



- Pour sélectionner ROM : il faut avoir A15=1, A14=1 et A13=1 ➔ $\overline{CE_{ROM}} = \overline{A15 \cdot A14 \cdot A13}$
- Pour sélectionner RAM: il faut avoir A15=1, A14=0, A13=1, A12=0 et A11=0 ➔ $\overline{CE_{RAM}} = A15 \cdot \overline{A14} \cdot A13 \cdot \overline{A12} \cdot \overline{A11}$
- Pour sélectionner E/S: il faut avoir A15=1, A14=0, A13=0 et A12=0 ➔ $\overline{CE_{E/S}} = \overline{A15 \cdot \overline{A14} \cdot \overline{A13} \cdot \overline{A12}}$

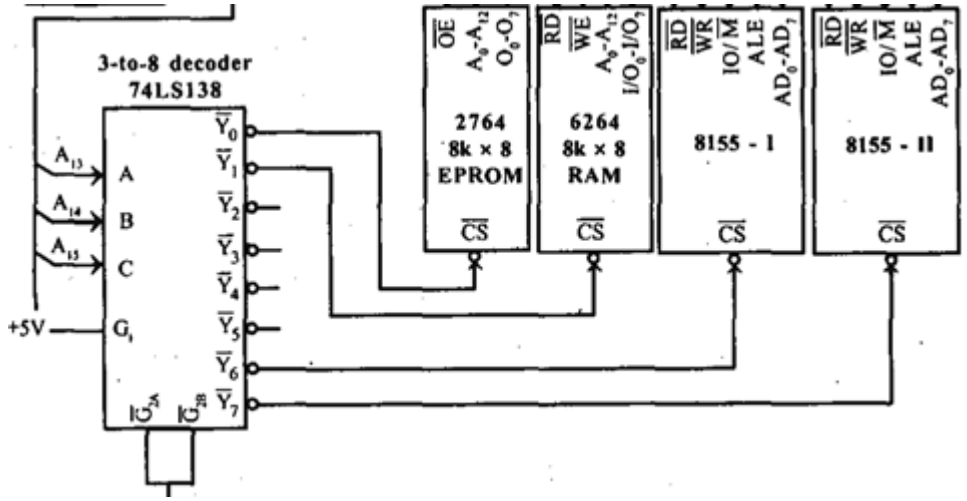
Plan mémoire

- Implémentation du décodeur d'adresse
  - Le décodeur peut se baser sur des portes logiques et/ou des décodeurs binaires



Décodage d'adresse en utilisant une porte logique

$$\overline{CE_{ROM}} = \overline{A15 \cdot A14 \cdot A13}$$

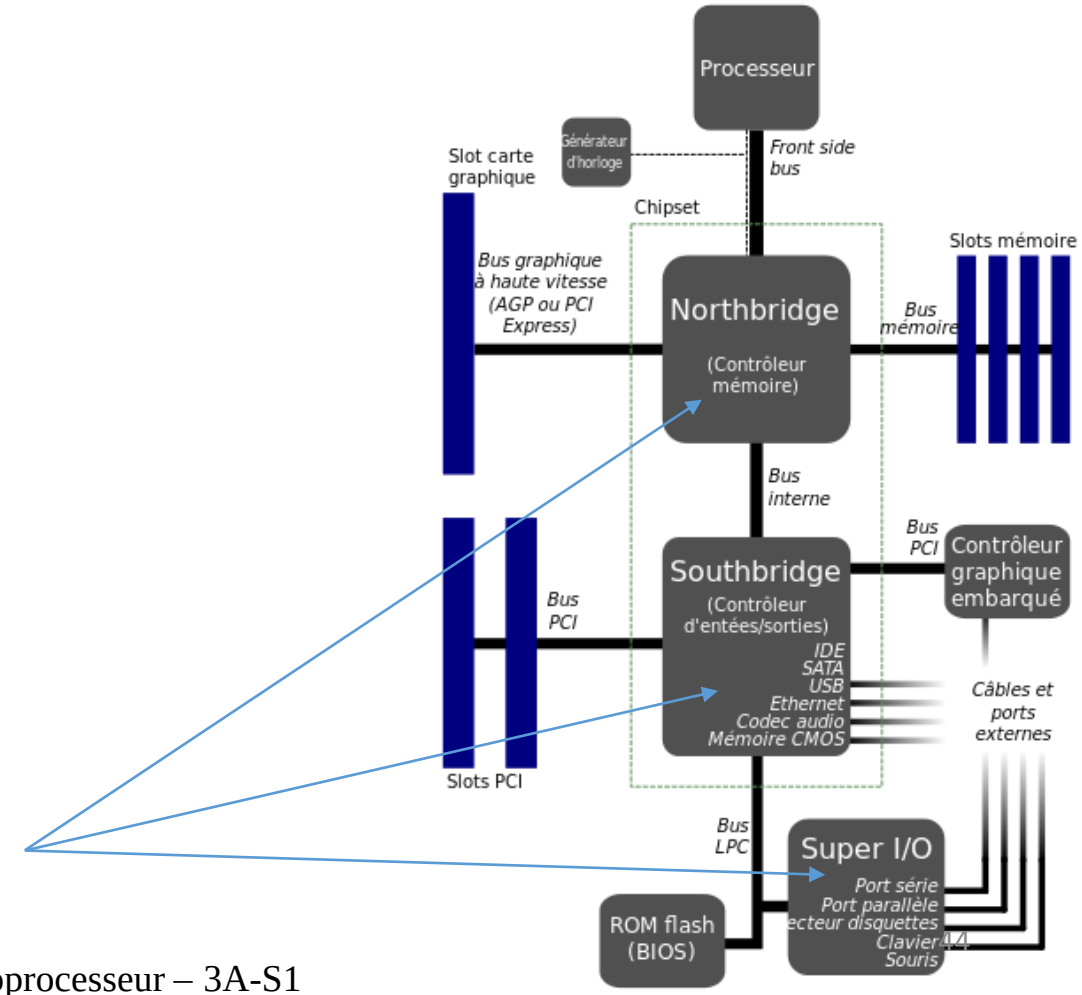


Décodage d'adresse en utilisant un décodeur binaire 3-vers-8

Plan mémoire

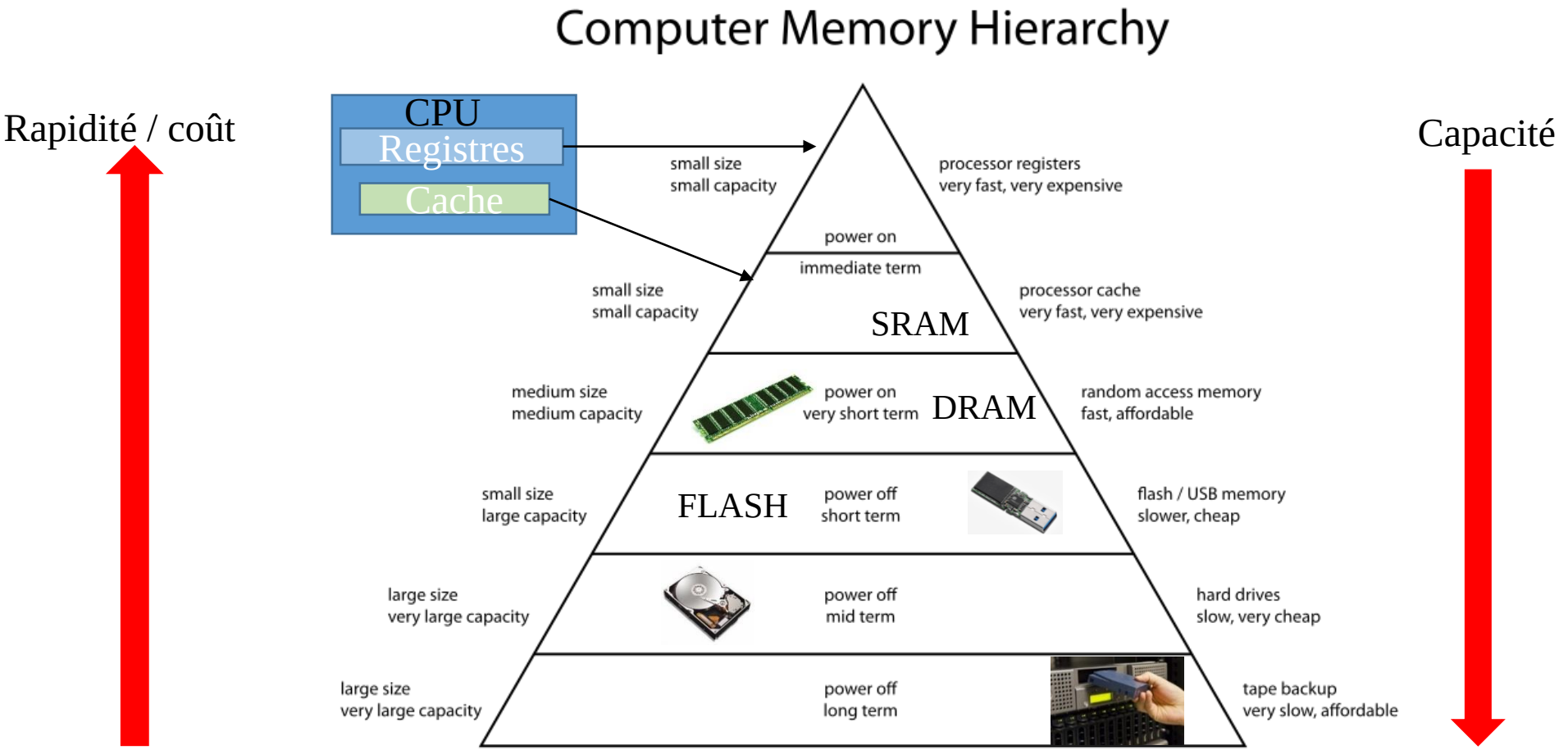
- Implémentation du décodeur d'adresse

  - Dans les ordinateurs, le décodage suit une arborescence, selon l'endroit où on met un composant dans la carte mère on a une plage d'adresses différentes
    - Exemple: les barrettes mémoire sur le canal 0 ont des plages différentes de celles sur le canal 1, l'emplacement de la carte son a sa propre plage d'adresses, etc.

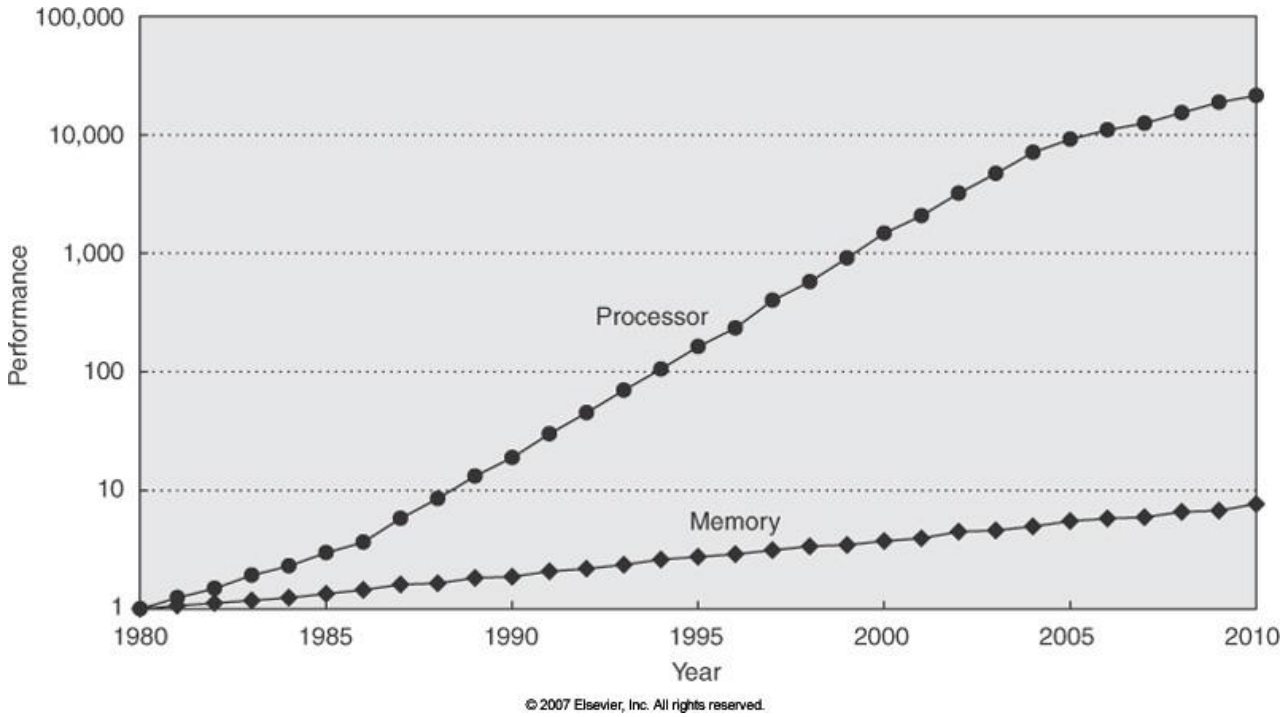


Décodage

Comparaison des supports mémoire



Evolution des performances des CPU et DRAM



- La performance (rapidité d'accès) de la mémoire RAM n'évolue pas au même rythme que le processeur

- En effet, si on veut implémenter une mémoire qui est aussi rapide que le processeur, elle sera beaucoup plus coûteuse que les mémoires qu'on utilise couramment

→ Pour être économiquement acceptables, les mémoires évoluent lentement

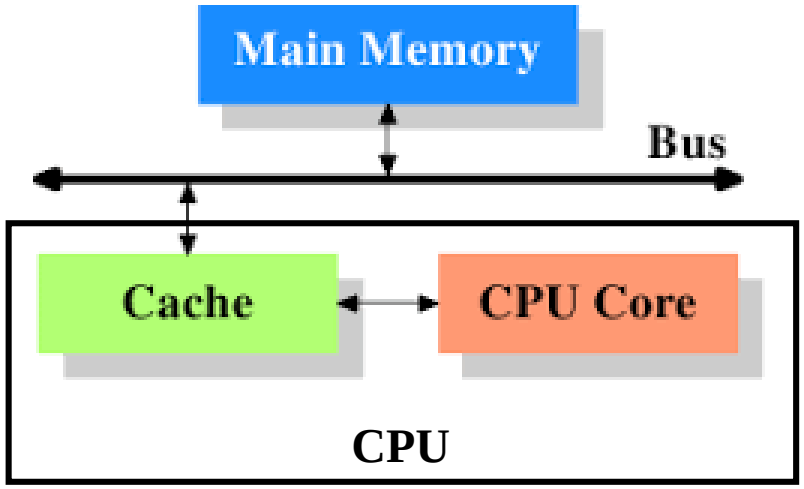
- Le processeur étant plus rapide perd beaucoup de temps en attente de la réponse de la mémoire pour lire une donnée

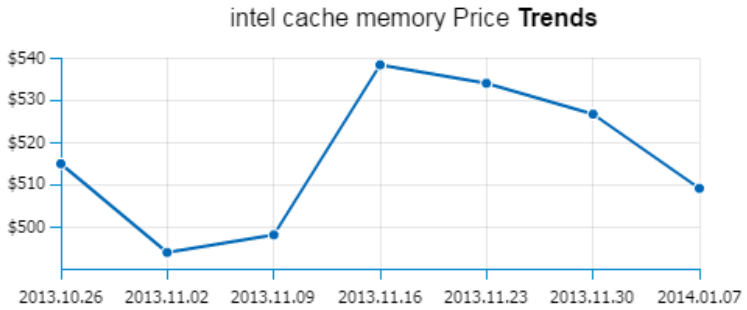
→ il faut trouver une solution qui est économiquement acceptable tout en profitant de la performance du processeur

- La mémoire cache du processeur permet de trouver un compromis entre coût et rapidité

La mémoire cache

- C’est une mémoire de petite taille qui se trouve souvent dans la même puce du processeur
- C’est une mémoire plus rapide que la mémoire principale mais plus chère
- La mémoire cache charge tout au long de l’exécution des parties de la mémoire principale, afin de diminuer le temps d’accès du processeur à ces données.
- Le cache utilise des algorithmes pour garder les données les plus utilisées par le processeur à l’intérieur du cache pour ne pas perdre du temps à les chercher plusieurs fois dans la mémoire principale





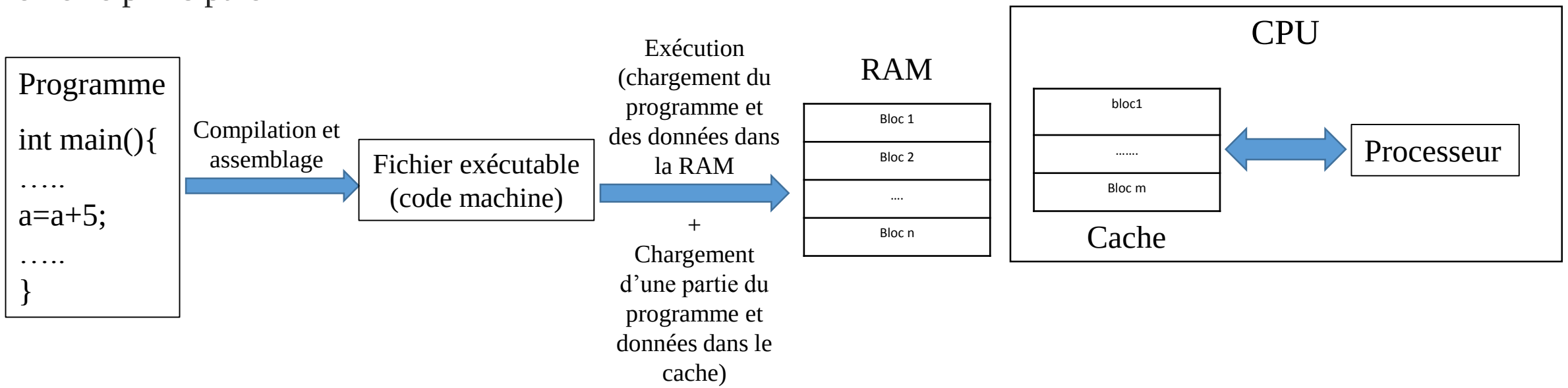
https://www.aliexpress.com/price/intel-cache-memory_price.html

La mémoire cache

- Pour exécuter un programme (une suite d'instructions), le processeur doit accéder à la mémoire principale (RAM) d'une manière séquentielle pour récupérer à chaque fois les données nécessaires à l'instruction en cours

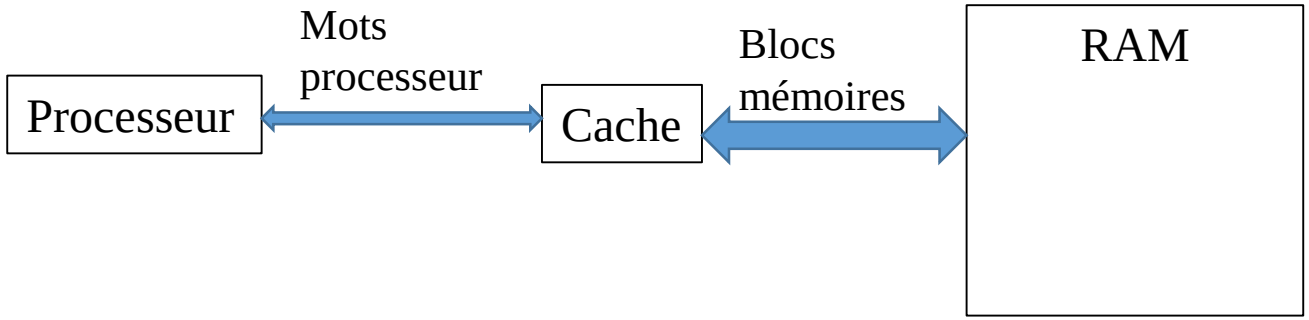
- Au lieu d'accéder aux données dans la RAM d'une manière séquentielle, ce qui prend du temps, on charge d'un seul coup des blocs mémoire qui sont susceptibles d'être utilisés, tout de suite après, par le processeur et on les charge dans le cache

- Le processeur accède aux données par la suite à partir du cache, qui a un temps d'accès plus court que la mémoire principale



La mémoire cache

- Comment le cache permet de diminuer le temps d'accès aux données par le processeur?
- L'accélération de l'accès est garantie par deux facteurs:
  1) Le fait de charger tout un bloc de données de la mémoire principale prend moins de temps que la récupération des données une par une
  2) La mémoire cache est plus rapide que la mémoire RAM
     a. Elle est implémentée par la technologie SRAM et pas DRAM
     b. Elle est plus proche du processeur



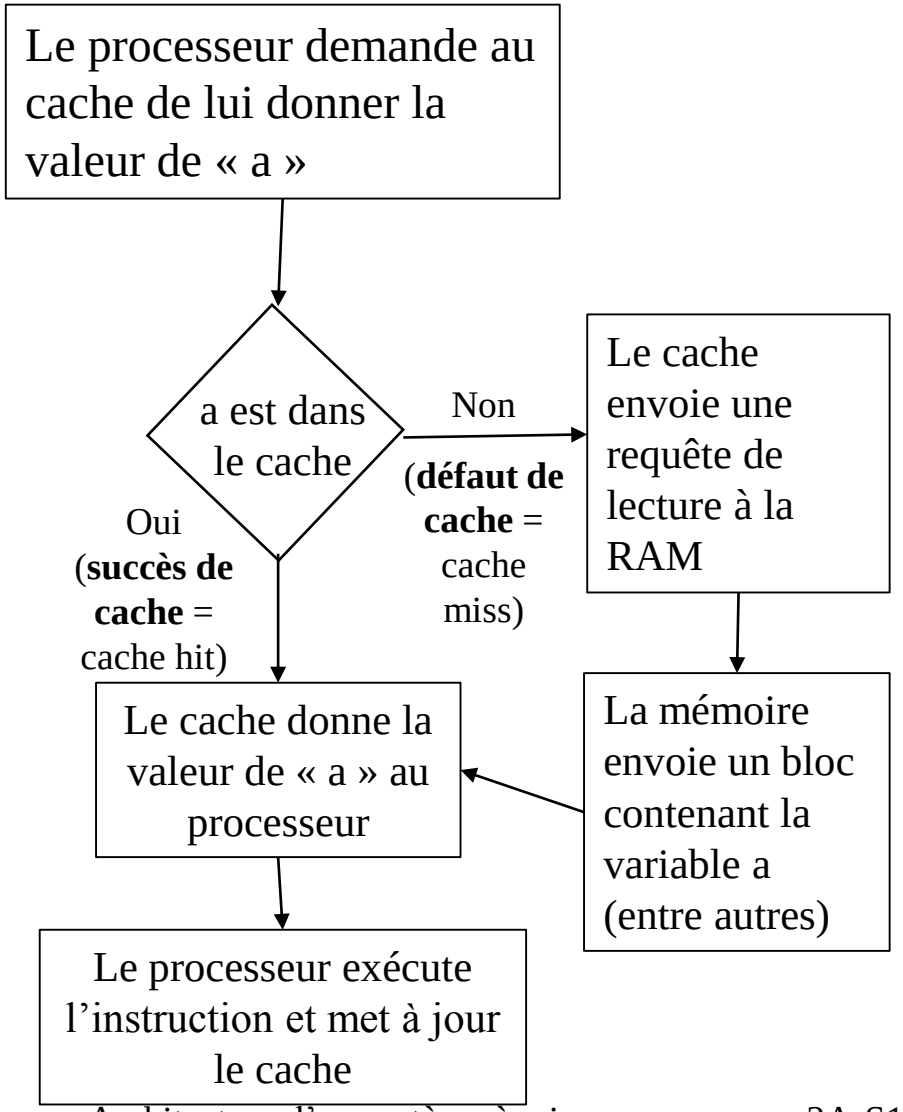
La mémoire cache

- Exécution des programmes en utilisant le cache

Programme

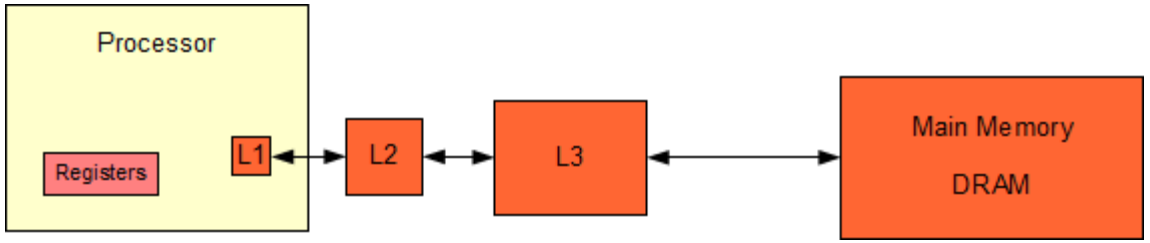
```
int main(){
…..
a=a+5;
…..
}
```

Exécution de l'instruction: a=a+5;



La mémoire cache

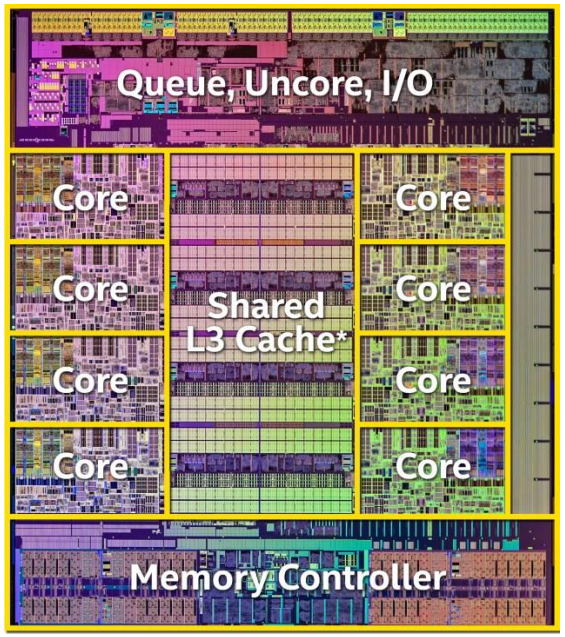
- Dans les microprocesseurs, on différencie plusieurs niveaux de caches, souvent au nombre de trois :
  - Le cache de premier niveau (L1), plus rapide et plus petit (cache de données pouvant être séparé du cache d'instructions) ;
  - Le cache de second niveau (L2), moins rapide et plus gros ;
  - Le cache de troisième niveau (L3), encore moins rapide et encore plus gros ;
- Ces derniers caches peuvent être situés dans la puce du CPU ou à l'extérieur (les CPUs modernes ont des caches intégrés)



- Exemple: le Core i5-4210U a un cache L1 de 8KB, un cache L2 de 2x256KB et un cache L3 de 3MB

```
wmic cpu get L2CacheSize,L3CacheSize

L2CacheSize  L3CacheSize
256          3072
```



Exemple: Le CPU Haswell-E d'Intel

La mémoire cache

- En utilisant plusieurs niveaux de cache, on essaie de trouver un compromis entre coût, capacité et temps d'accès

- Les différents niveaux de cache ne sont pas implémentés de la même façon
  - Le niveau L1 est le plus rapide mais le plus cher par unité de stockage
  - Le L2 est un peu moins rapide et moins cher mais plus grand
  - La même logique est utilisée par les autres niveaux de cache

La mémoire cache

- Si on a un cache plus grand → le nombre de défauts de cache diminue → moins d'accès à la RAM → exécution plus rapide → Pourquoi de ne pas avoir des caches avec des dizaines/centaines de MB

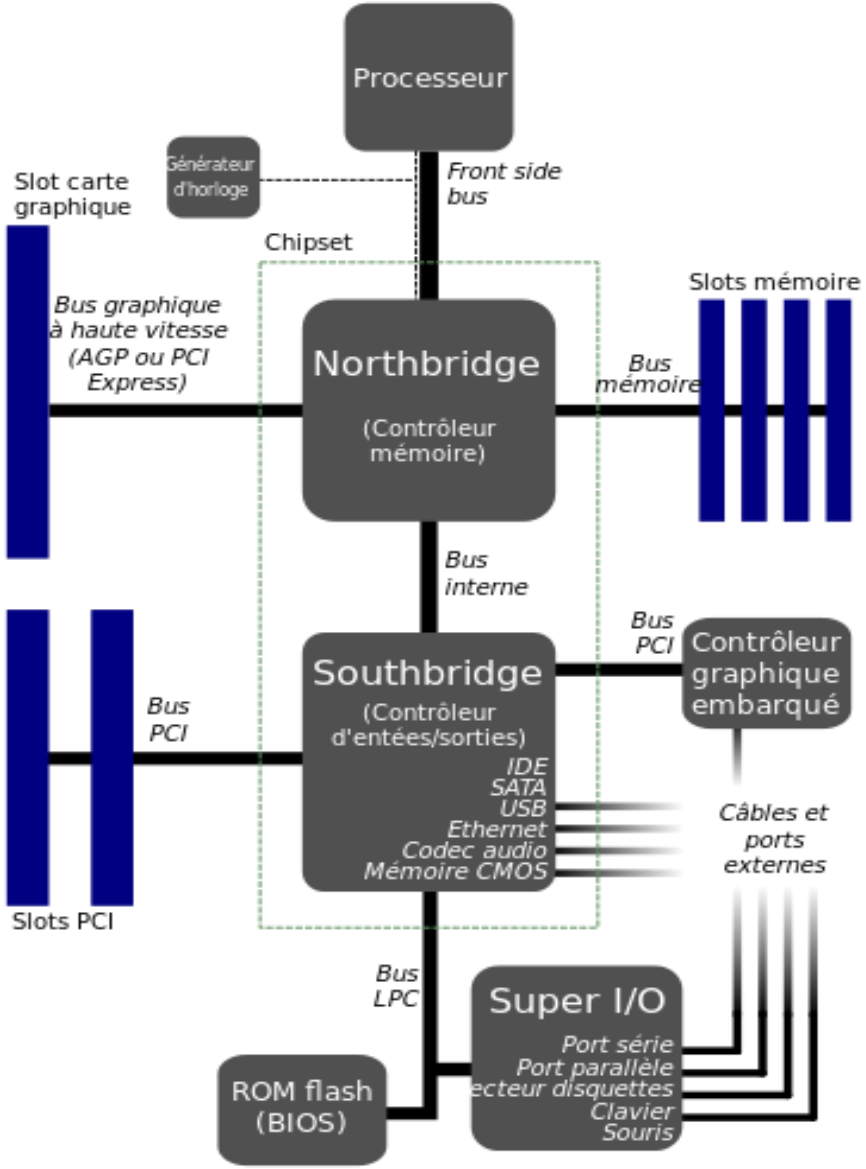
- Le problème est que la taille du cache est limitée par certains facteurs
  - Le prix: la technologie SRAM est plus chère que la DRAM
  - L'efficacité en temps d'accès: le temps nécessaire pour chercher une donnée dans le cache devient trop grand si on a un grand cache
  - L'efficacité en termes de ressources: le nombre de portes logiques nécessaires pour chercher une donnée dans le cache et pour référencer toutes les données qui y sont explose avec la capacité du cache
  - La possibilité d'intégration dans le CPU: Il devient plus compliqué d'intégrer le cache dans le CPU si sa taille est très importante

Bus

- Dans un ordinateur, il n'y a pas un seul bus mais une architecture de bus adaptée à la vitesse des composants connectés

- Ces bus sont connectés à un chipset qui joue le rôle de pont entre les bus et gère le trafic
- Le chipset détermine la compatibilité de tous les composants qui y sont connectés (CPU, mémoires, disque dur, carte graphique, périphériques, etc.): un chipset peut être limité à un seul type de mémoire (DDR2, DRR3, etc.)

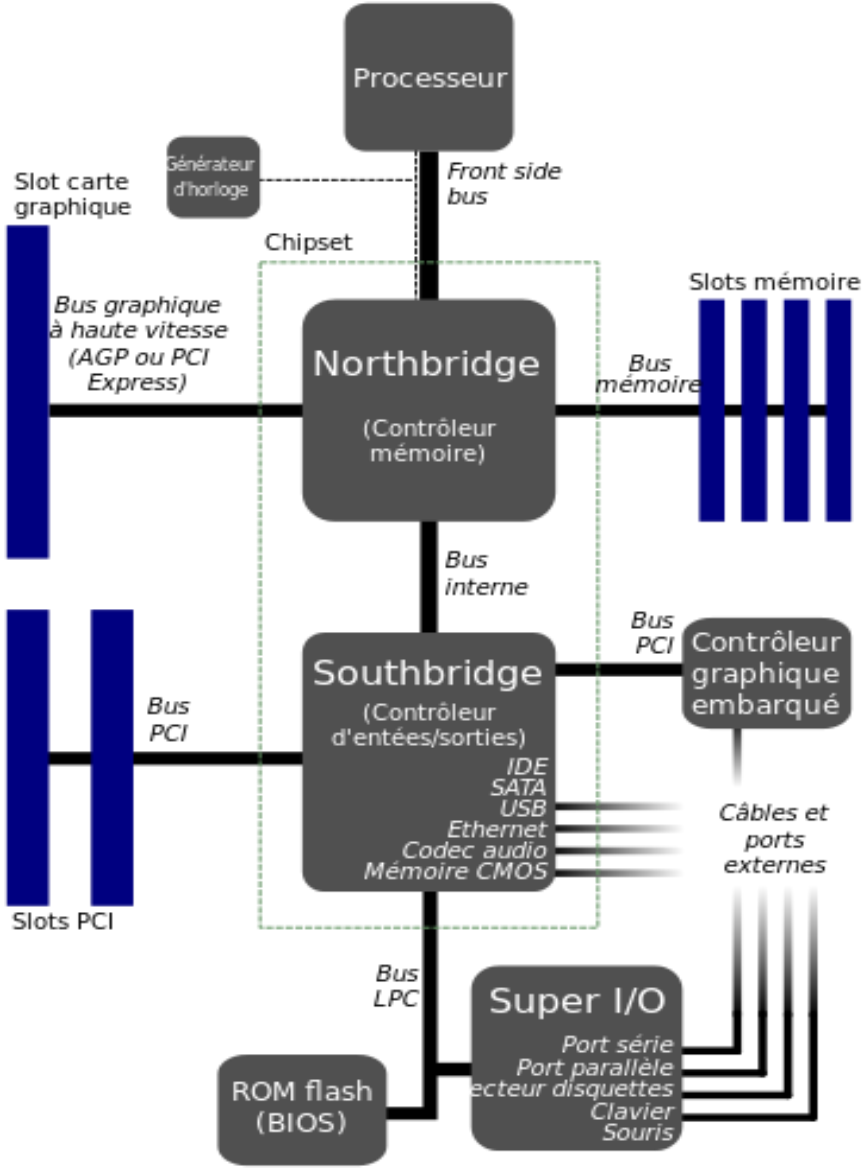
- Le chipset contient traditionnellement deux ponts qui sont le pont nord (northbridge) et le pont sud (southbridge)



Bus

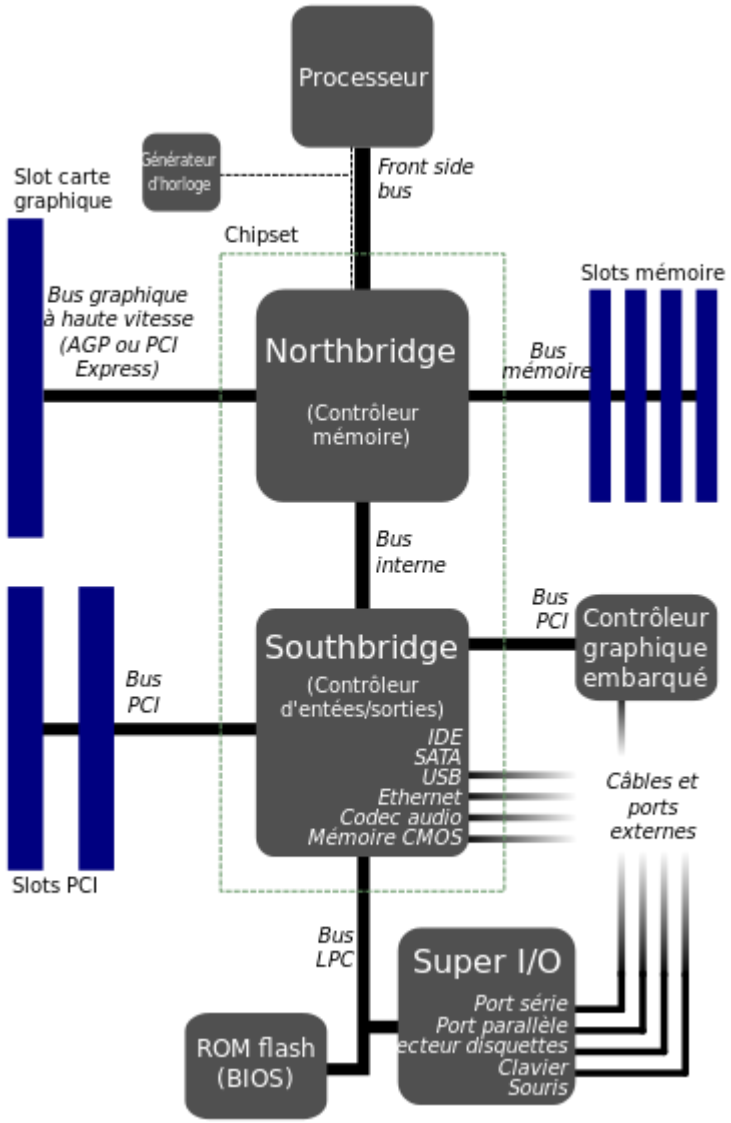
- Le **front side bus** (**FSB**), aussi appelé **bus système**, est traditionnellement le bus informatique qui relie le processeur au « Northbridge » et qui gère les échanges avec les périphériques proches du CPU et notamment avec la mémoire vive.

- Le « pont nord » gère des communications entre le microprocesseur et les périphériques rapides tel que :
  - la mémoire vive ;
  - le bus AGP pour carte graphique ;
  - les bus PCI Express pour les périphériques externes rapides dont les cartes graphiques ;
  - Le « pont sud »



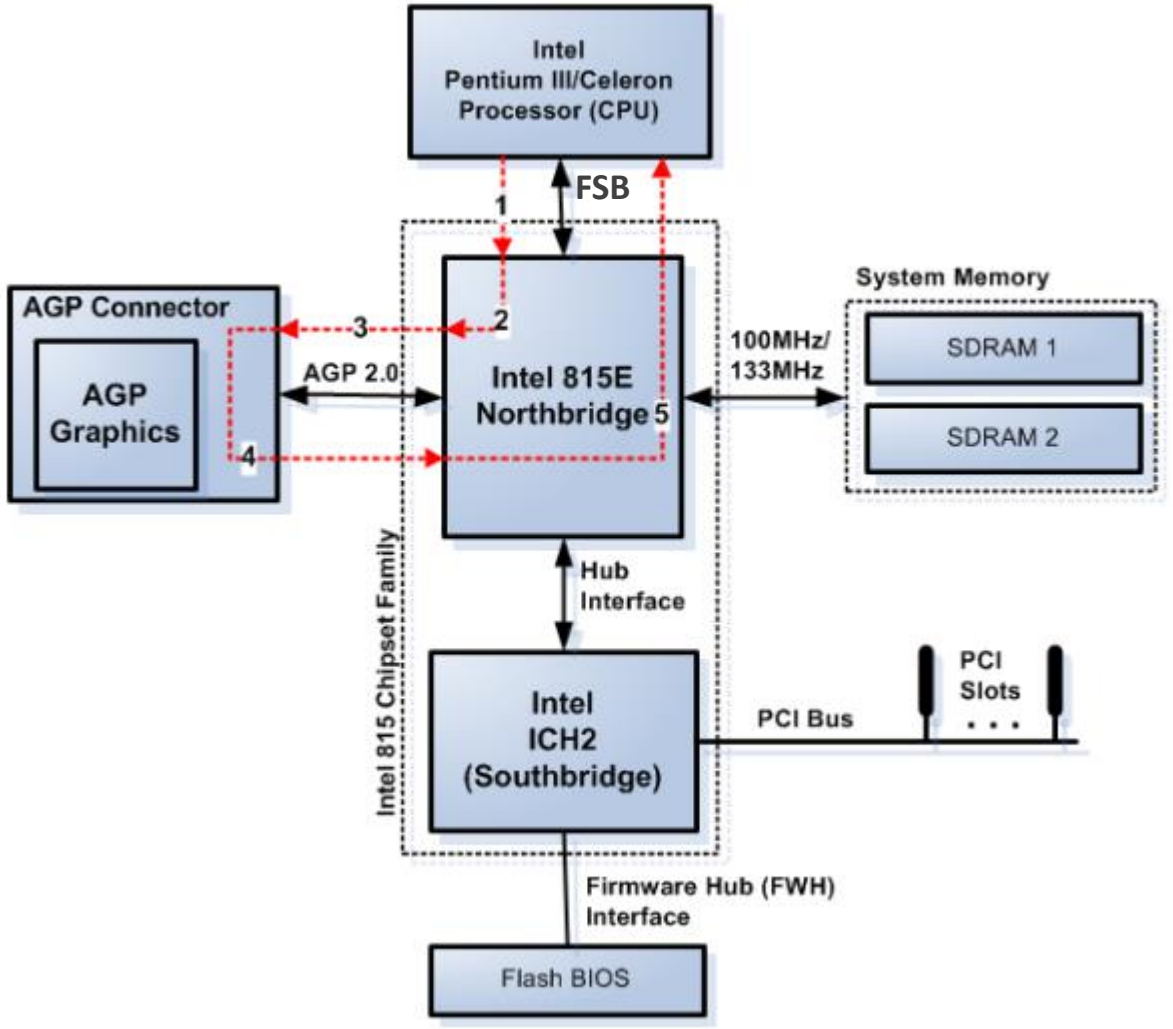
Bus

- La puce southbridge connecte les composants plus lents que ceux pris en charge par le northbridge:
  - bus PCI,
  - l'interface Ethernet, USB et firewire
  - l'interface PS/2 pour le clavier et la souris,
  - le port série,
  - le port parallèle
  - Certaines de ces fonctions sont souvent prises en charge par un contrôleur secondaire d'I/O et, dans ce cas, le southbridge fournit une interface à ce contrôleur



Bus

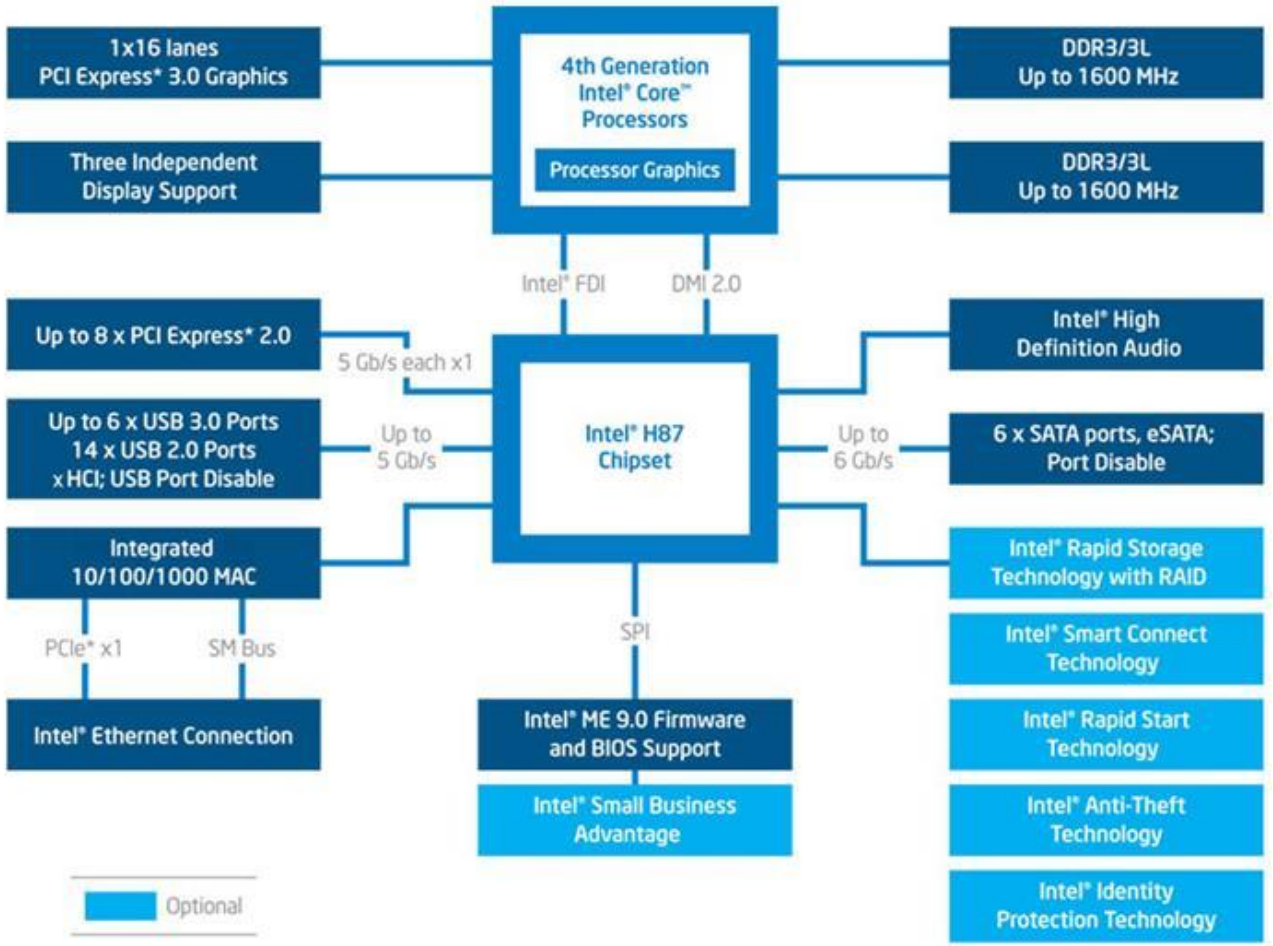
- Structure du northbridge
  - Décodeur d'adresse
  - Routeur des requêtes
  - Contrôleurs mémoire
  - Contrôleurs AGP/PCI Express



http://resources.infosecinstitute.com/system-address-map-initialization-in-x86x64-architecture-part-1-pci-based-systems/#gref

Bus

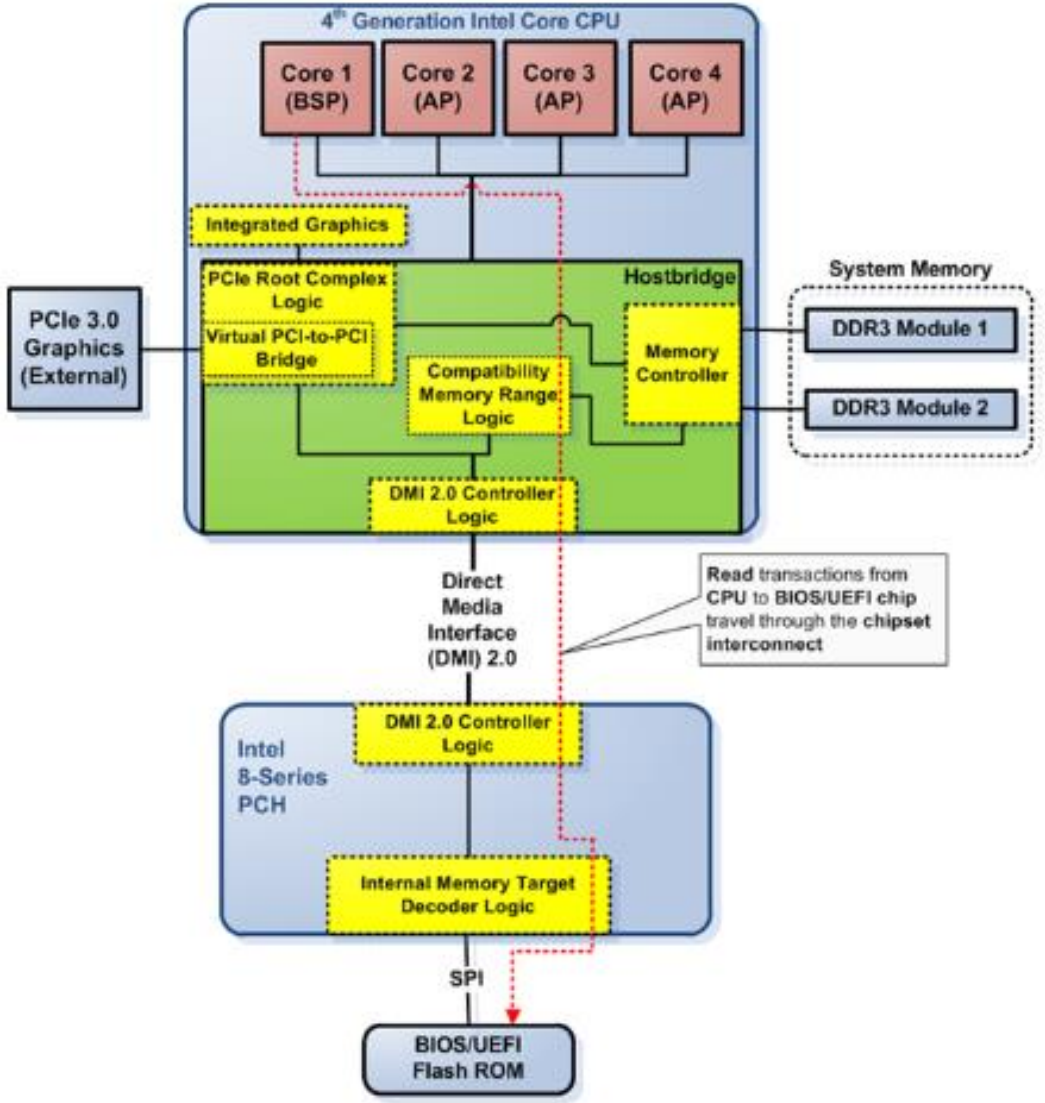
- Dans les ordinateurs modernes, le northbridge est intégré dans le CPU et s'appelle Hostbridge
- Le southbridge est remplacé par le PCH (Platform Controller Hub, H87 dans l'exemple)
- Le CPU communique directement avec la mémoire et les dispositifs PCI Express, ce qui permet d'accélérer cette communication
- Le fait que la communication entre CPU et le reste des périphériques ne passe que par le PCH permet de l'accélérer



http://resources.infosecinstitute.com/system-address-map-initialization-x86x64-architecture-part-2-pci-express-based-systems/#gref

Bus

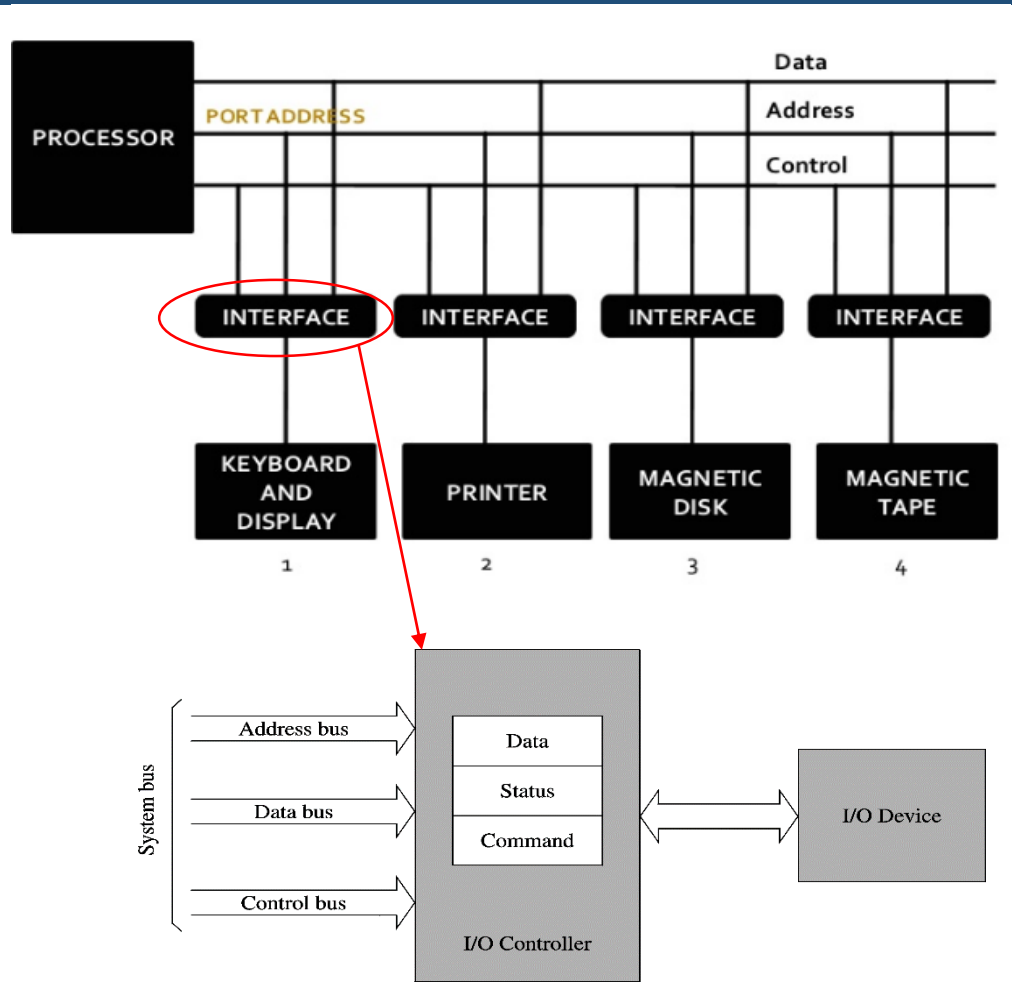
- Le hostbridge reprend les mêmes fonctionnalités du northbridge
- Le PCH reprend celles du southbridge



http://resources.infosecinstitute.com/system-address-map-initialization-x86x64-architecture-part-2-pci-express-based-systems/#gref

Les E/S

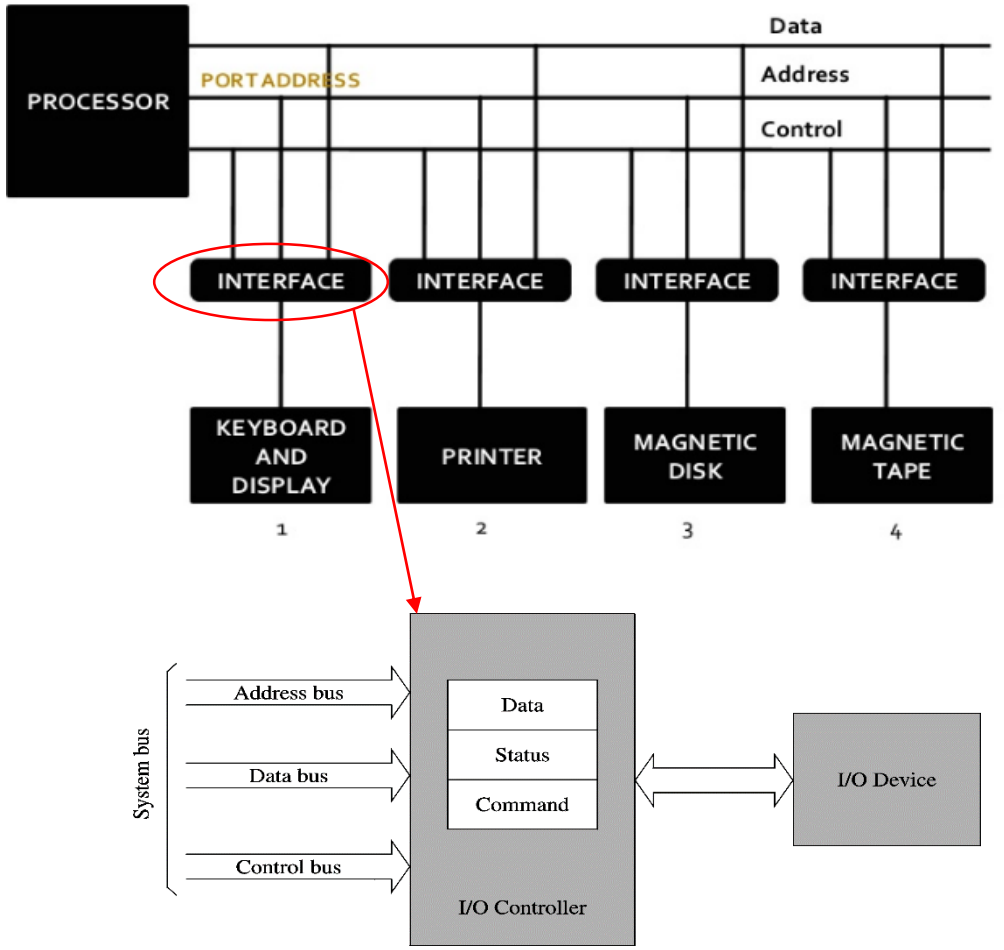
- Les contrôleurs des E/S
  - Fonction
    - Agissent comme des interfaces entre le bus et les périphériques
    - Epargnent au processeur la gestion des détails de communication bas/niveau
    - Gèrent les interfaces électriques
  - Composition
    - 3 types de registres
    - Données: les données échangées entre le bus et le périphériques transitent par ces registres
    - Etat: indique l'état du périphérique (réception d'une donnée, la transmission a pris fin, etc.)
    - Commande/contrôle: permettent de configurer et de contrôler les périphériques (activer la réception/transmission, configurer la vitesse de transmission, etc.)
  - Emplacement
    - Dans le northbridge/southbridge
    - Intégrés dans les E/S



Les E/S

- Les contrôleurs des E/S
  - Communication avec le processeur
    - Du point du processeur, un périphérique est un ensemble de registres (données, état et contrôle) qu'il peut lire ou écrire dedans pour communiquer

    - Exemple: la communication avec le clavier nécessite le plus souvent deux registres:
      - Un registre de données qui contient le code ASCII du dernier bouton appuyé
      - Un registre d'état/contrôle qui indique entre autres si un nouveau caractère vient d'être tapé. Le bit correspondant à cet évènement sera mis à 1 si on tape un nouveau caractère et il se remet automatiquement à 0 lorsque le processeur lit le caractère du registre de données



Les E/S

- Les protocoles de communication avec les E/S
  - Par scrutation (polling)
  - Par interruption
  - Par DMA

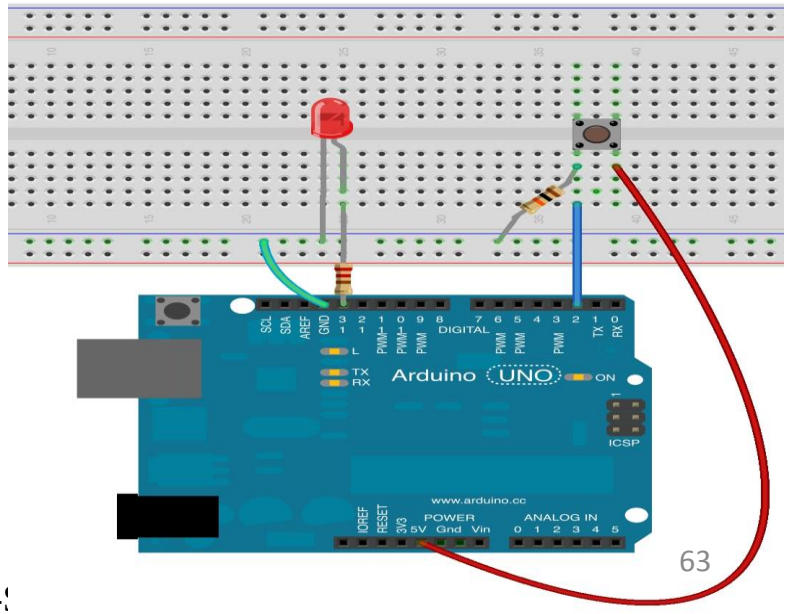
Les E/S

- Les protocoles de communication avec les E/S

1) La scrutation (polling)
   - Scruter l'état de l'E/S
   - Exemple: Pour savoir si l'utilisateur a tapé un caractère, le processeur surveille le registre d'état du clavier
   - Exemple sur Arduino: programme qui allume la LED quand le bouton est relâché et l'éteind quand le bouton est appuyé

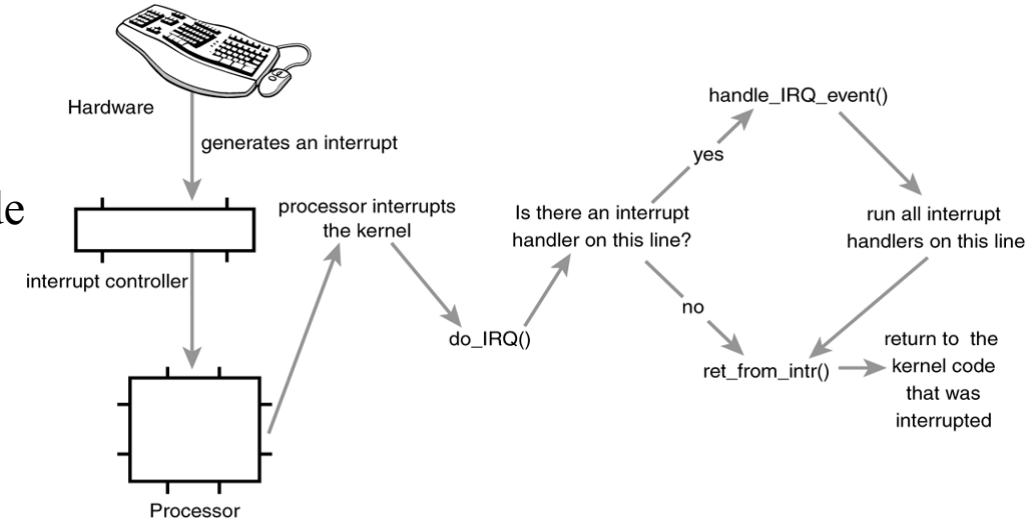
```
void loop() {
if(digitalRead(bouton)==LOW)
{
  digitalWrite(led,HIGH);
}
else
{
  digitalWrite(led,LOW);
}
}
```

Scrutation de l'état du bouton dans une boucle



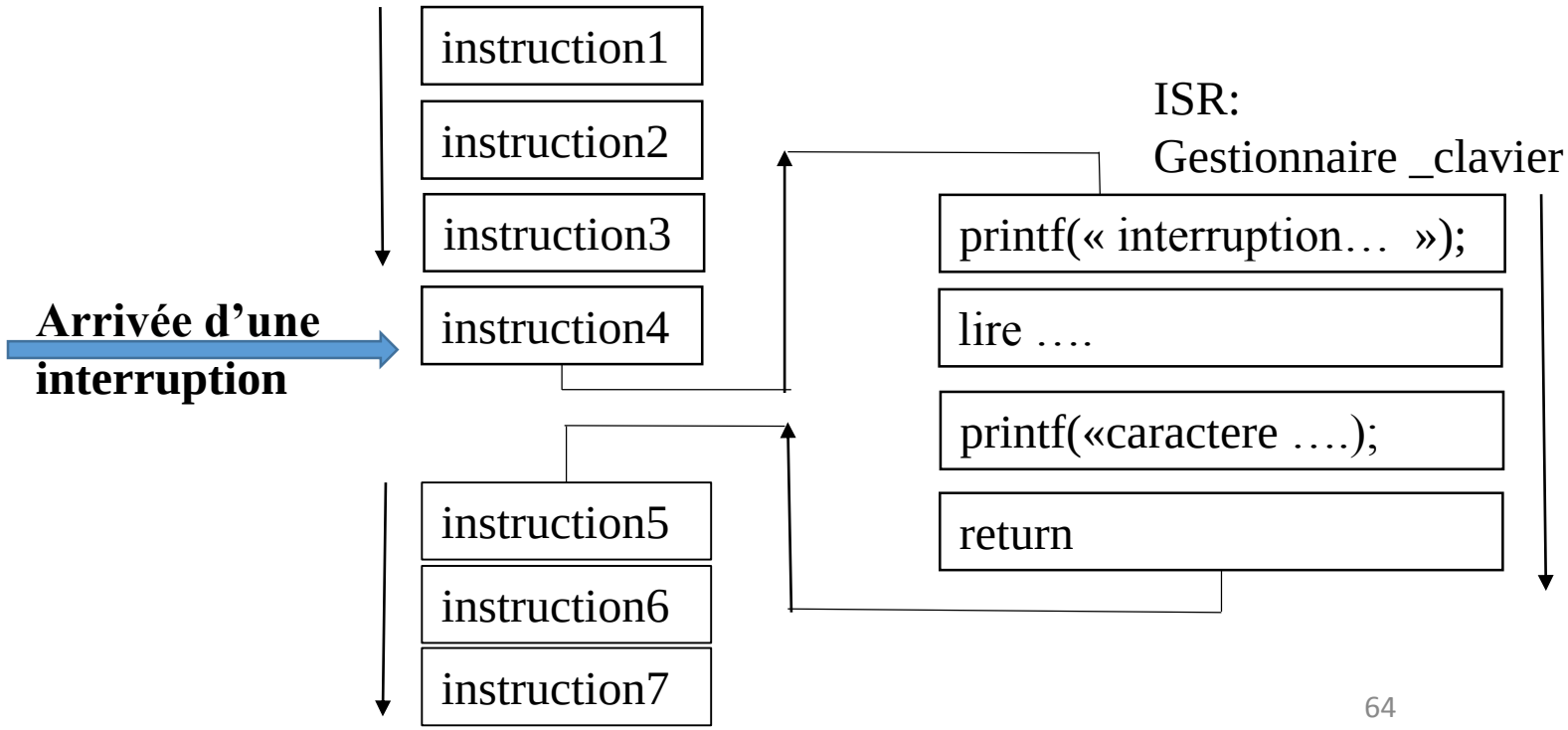
Les E/S

- Les protocoles de communication avec les E/S
  2) Les interruptions
     - L'E/S génère une interruption
     - Le contrôleur d'interruption met le processeur au courant de l'interruption
     - Le processeur interrompt ce qu'il était en train d'exécuter
     - S'il y a une routine d'interruption: Interrupt Service Routine (ISR), le processeur l'exécute
     - A la fin de l'exécution de l'ISR, le processeur reprend ce qu'il était en train d'exécuter

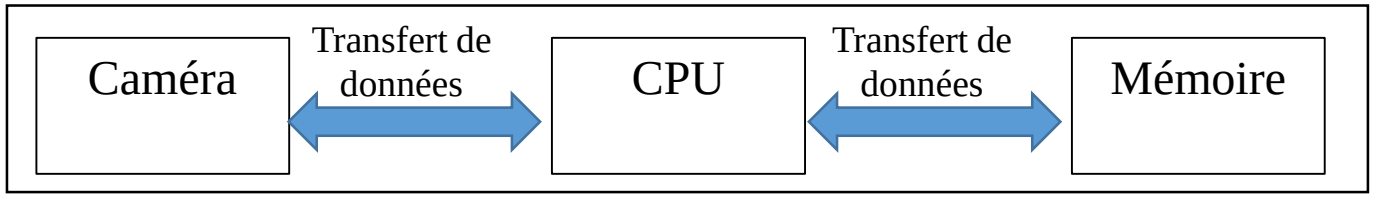



```
int main(){
Instruction1;
Instruction2;
Instruction3;
Instruction4;
Instruction5;
Instruction6;
Instruction7;
....
}

//ISR exécutée quand un bouton est appuyé
int gestionnaire_clavier()
{printf ("interruption clavier");
char c= Lire(registre des donnees du clavier);
printf("caractere lu=%c\n",c);
}
```



- Les protocoles de communication avec les E/S
  3) Le DMA (Direct Memory Access)
     - Ce mode de communication permet à l'E/S d'accéder directement à la mémoire sans passer par le processeur
     - L'objectif est d'accélérer le programme en libérant le processeur de cette tâche
     - Exemple:
       - Le processeur veut récupérer les images de la caméra (en streaming) et y appliquer un traitement (filtrage, tracking d'objets, etc.)
       - 1ère solution: sans DMA
         - Le processeur doit envoyer des commandes à l'E/S, récupérer les images et les stocker dans la mémoire



- Les protocoles de communication avec les E/S

3) Le DMA (Direct Memory Access)

- 2ème solution: avec DMA
  - Le processeur lance une commande DMA au contrôleur DMA
  - Le contrôleur DMA récupère les images de la caméra et les stocke directement dans la mémoire sans « déranger le processeur »
  - Le processeur entre temps peut accéder à la mémoire pour appliquer des traitements sur les images précédentes