15 octobre

```cpp
#include <iostream>

int main() {
    int x ; // x est un entier signé (complément à deux)
    unsigned int y ; // y est un entier non négatif
    char z ; // z est un caractère

    // Toutes ces variables tiennent sur 32 bits

    // On place la même valeur de 32 bits dans les 3 variables
    // avec le bit de poids fort valant 1
    // 1000 0000 0000 0000 0000 0000 0010 1111

    x = 0b10000000000000000000000000101111 ;
    y = 0b10000000000000000000000000101111 ;
    z = 0b10000000000000000000000000101111 ;

    // L'affichage des trois valeurs est différent
    // L'affichage tient compte du type: ce sont les
    // instructions qui donnent leur sémantique aux
    // valeurs représentées en binaire.

    std::cout << x << std::endl ;

    std::cout << y << std::endl ;

    std::cout << z << std::endl ;

}
```

3 Variables x, y, z

Etant donné une représentation binaire, quelle information représente-t-elle ?

```
-2147483601
2147483695
/
```

← x

← y

← 2

?

0

Chapitre 3: Organisation

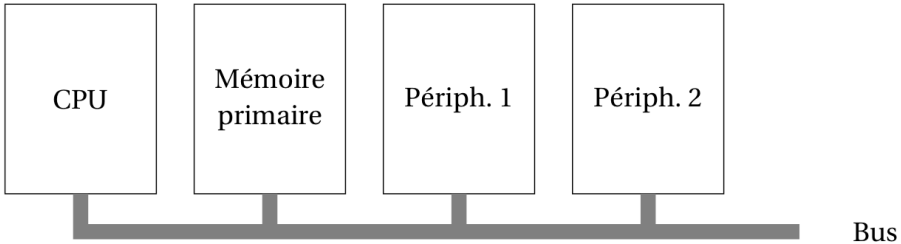


FIGURE 3.1. – Un modèle simple de l'organisation d'un ordinateur.

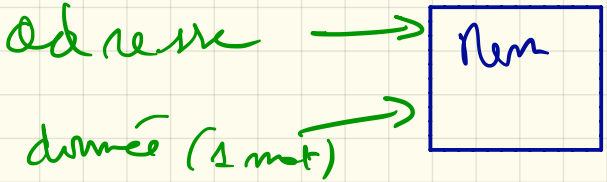
Mémoire (principale / primaire) vRAM

Stockage à court terme de programmes en cours d'exécution et des données.

- La mémoire est découpée en cases (cellules) qui ont chacune une taille fixe (1 octet)
- Chaque case possède un numéro unique (0, 1, 2...) qu'on appelle adresse.
- Les lectures / écritures en mémoire se font par mots de taille fixe (4 / 8 cases)
  - 32 bits
  - 64 bits.

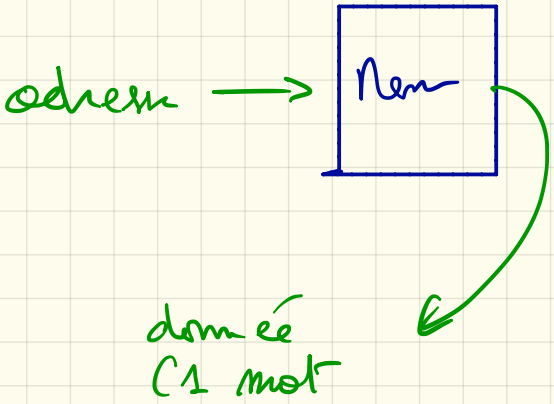
Opérations :

Ecriture



la mémoire stocke la donnée à partir de l'adresse

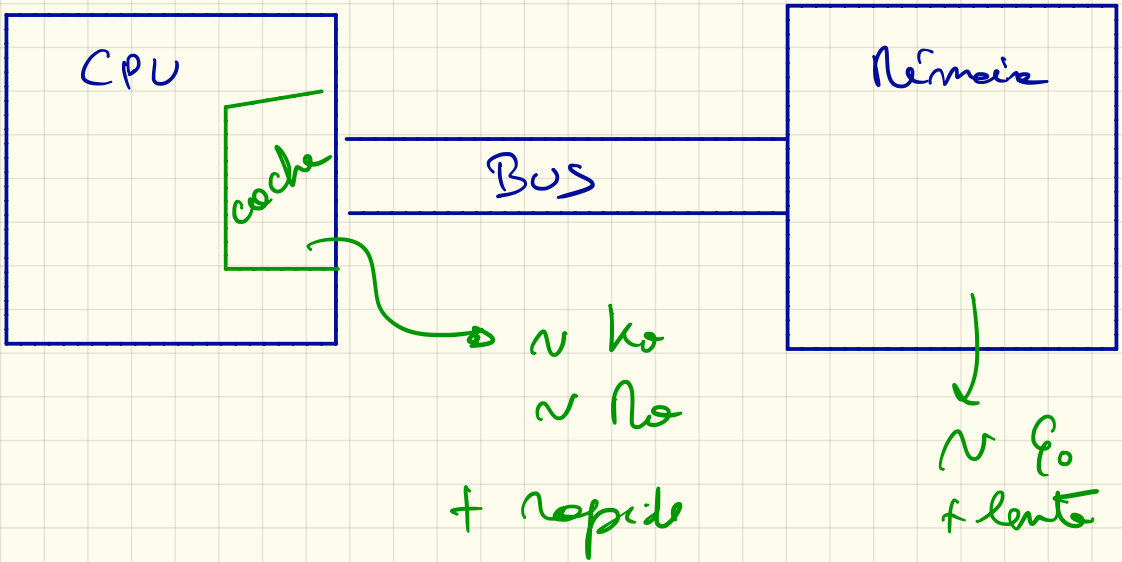
Lecture



la mémoire renvoie la donnée apid l'adresse

la vitesse du bus est un facteur qui ralentit l'accès à la mémoire

Mémoire Cache



lecture en mémoire ?

① le CPU interroge le cache.

② Si la donnée s'y trouve → cache hit

Sinon → on interroge la mémoire et on essaie de stocker cette donnée dans le cache

cache miss

Ecriture : idem : on peut écrire dans le cache plutôt que dans la mémoire

Si la cache est pleine → choisir une victime pour faire de la place.

si cette victime a été modifiée, il faut recopier la modification en mémoire !

algos complexes

CPU

3 parties :
- l'unité de contrôle
- l'ALU
- registres

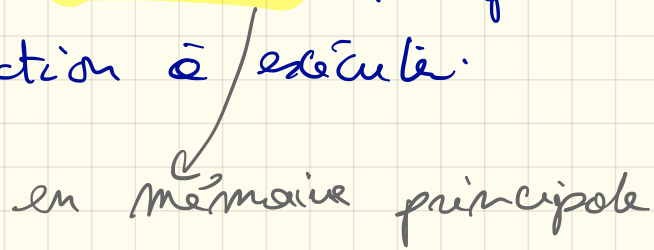
Registres : mémoire très petite mais très rapide dans le CPU.

2 types
- de travail : contiennent les données des instructions
- de contrôle : nécessaires pour le fct. du CPU.

Registres de Contrôle:

1) le registre d'instruction (IR)
   Contient l'instruction à exécuter.

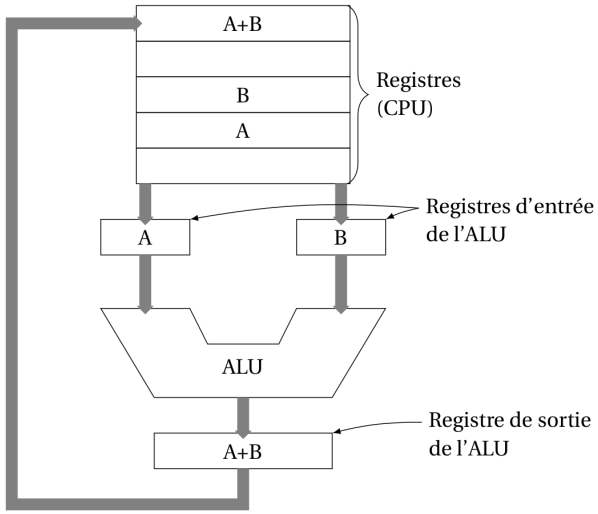
2) le pointeur d'instruction (PC)
   contient l'adresse de la prochaine instruction à exécuter.



Chemin des données

Registres → ALU → Registres

Cycle



URE 3.7 – Le chemin des données. Les flèches grises symbolisent les bus (internes au

Boucle d'interprétation (fetch-decode-execute)

dans machine :

1. Charger, dans le registre IR, l'instruction située en mémoire (M) à l'adresse donnée par PC : IR ← M[PC]
2. Incrémenter PC : PC ← PC+1
3. Analyser l'instruction dans IR
4. Exécuter l'instruction (ceci peut modifier PC)
5. Aller en 1.

Voir simulateur !

Remarque : <u>taille des registres</u> 32 bits ~ 64 bits

⚠ taille de la mémoire

PC ~ 32 bits → au maximum

2^{32} adresses ≠

Si les cases font 1 octet

→ limite de 4 Go sur la mémoire.

Amélioration de l'efficacité du CPU

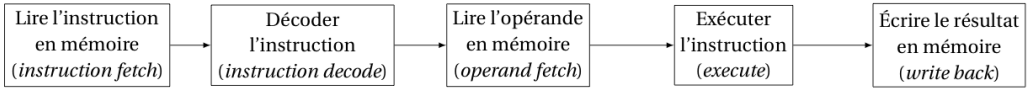


FIGURE 3.10. – Illustration d'un *pipeline* à cinq étages.

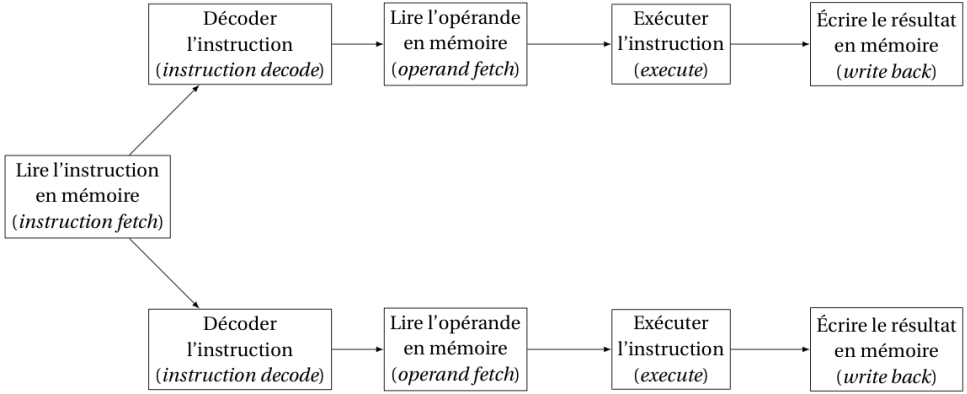
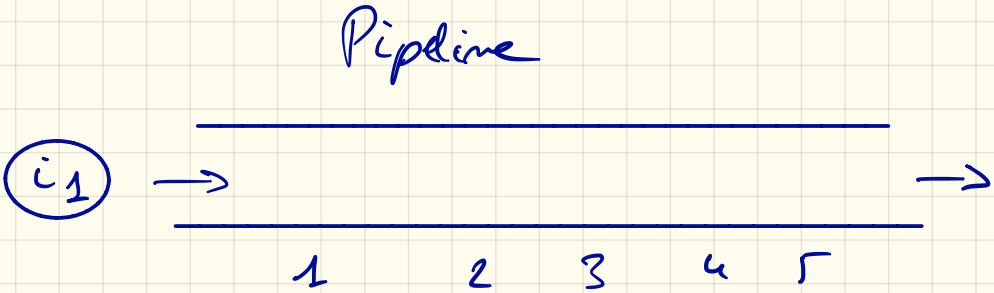
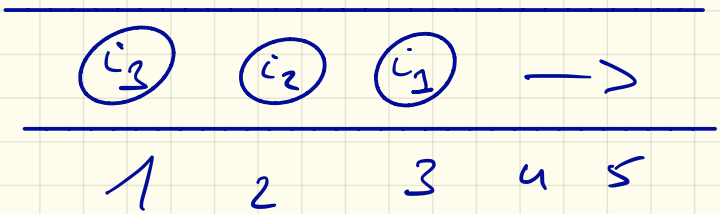
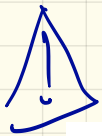


FIGURE 3.11. – Un *pipeline* superscalaire à cinq étages. L'étage « *instruction fetch* », qui est typiquement très rapide, est unique. Les instruction récupérées en mémoire par cet étage sont réparties sur deux *pipelines*, qui peuvent exécuter deux instructions en parallèle.





eax, ebx registres



```
1    add eax, ebx ; place dans eax la somme eax+ebx
2    add ecx, eax ; place dans ecx la somme ecx+eax
```

la 2e instruction a besoin du résultat de la 1e !