Introduction à l'informatique

pour les mathématiques, la physique et les sciences computationnelles

Yann Thorimbert

Chapitre 5
Architecture des ordinateurs

Yann Thorimbert

Chapitres du cours

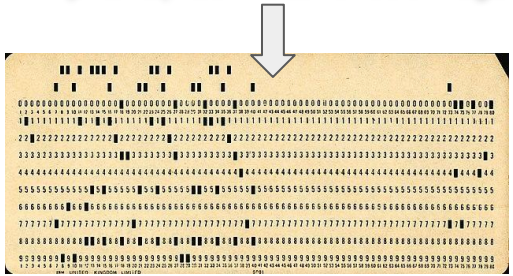
1. Origines des ordinateurs et des réseaux informatiques
2. Codage des nombres
3. Codage des médias
4. Circuits logiques
5. **Architecture des ordinateurs** ←
6. Conception et exécution de programmes
7. Algorithmique, programmation et structures de données

Hardware et software

- Hardware : partie matérielle de la machine (circuits électroniques)
- Software : partie logicielle (instructions à exécuter)

Hardware

```
if x < 0:
    print("le nombre est négatif")
```



Software

Ordinateurs "généralistes" VS "dédiés"

- Les ordinateurs n’ont pas toujours été conçus pour exécuter n’importe quels types de programmes avec la même aisance.

- Il est tentant de concevoir la partie matérielle spécifiquement, de façon à lui faire épouser la partie logicielle.

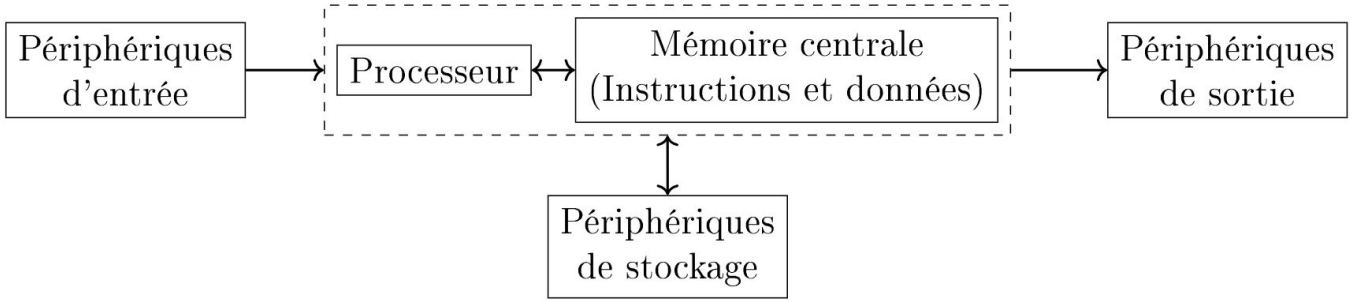
- Exemple moderne d’architecture **dédiée** au calcul parallèle sur de nombreuses données : carte graphique (GPU).

Architecture de von Neumann

- Il n’a pas toujours été évident que les parties matérielles et logicielles devaient être clairement séparées.
- Dès les années 1950, premiers ordinateurs (notamment EDVAC) à proposer un hardware "immuable", séparé d’un un software facile à changer.
- Concept de **programme stocké** : les instructions sont bien séparées du hardware. À la base de l’**architecture** dite de "**von Neumann**".
- Cela dit, le programme est toujours stocké d’une façon ou d’une autre, quelle que soit la machine.

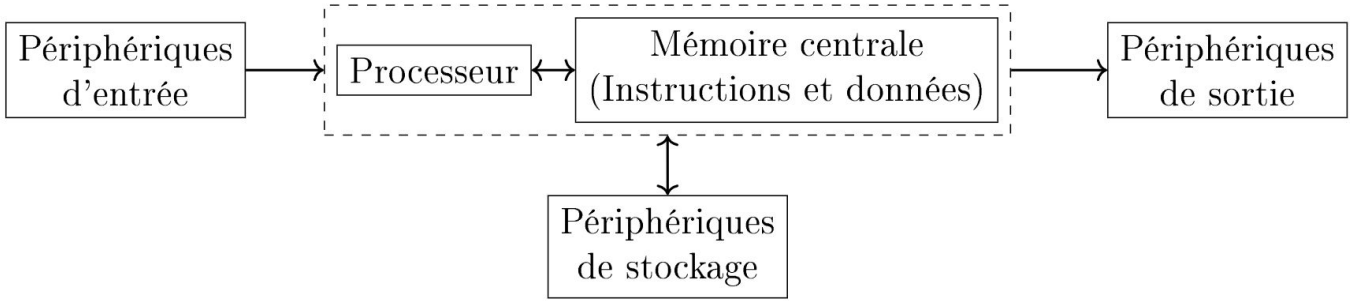
Architecture de von Neumann (programme stocké)

- Dans les grandes lignes, architecture encore utilisée sur les ordinateurs modernes.



Architecture de von Neumann (programme stocké)

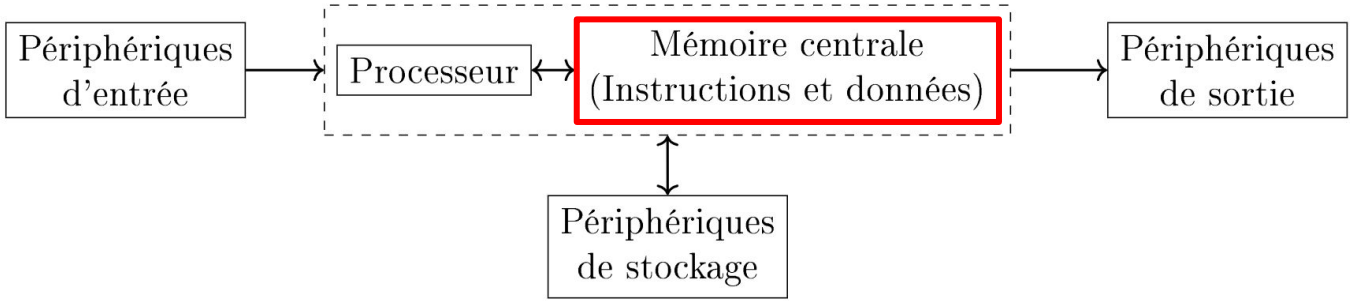
- Coeur de l'ordinateur : **processeur** et **mémoire centrale**
- Dispositifs "non essentiels" : périphériques



- Les flèches représentent le sens de circulation de l'information.

La mémoire centrale

- Mémoire de travail de l'ordinateur.
- À ne pas confondre avec les périphériques de stockage auxiliaires.



La mémoire centrale

- Mémoire de travail de l'ordinateur.

- À ne pas confondre avec les périphériques de stockage auxiliaires ou mémoires de masse :
  Disque dur magnétique, mémoire flash SSD

- Typiquement :

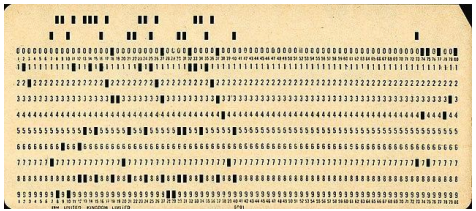
	Persistance	**Capacité**	**Accès**
Mémoire centrale	Non (volatile)	Petite	Rapide
Stockage auxiliaire	Oui	Grande	Lent

Aparté sur les types de mémoires persistantes

- Nombreuses évolutions à travers le temps.
- Mécanique : modification de la forme du support (carte perforée)
- Magnétique : modification de l'aimantation (disquette, disque dur, …)

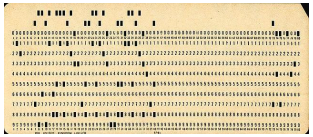
Aparté sur les types de mémoires persistantes

- Mécanique : modification de la forme du support (carte perforée)
- Magnétique : modification de l'aimantation (disquette, disque dur, …)
- Optique : modification des propriétés réflectives (CD, DVD, Blu-Ray)
- Flash : transistors spéciaux permettant de piéger des électrons au sein d'une petite zone isolante (clé USB, carte SSD).



(Pas à l'échelle)

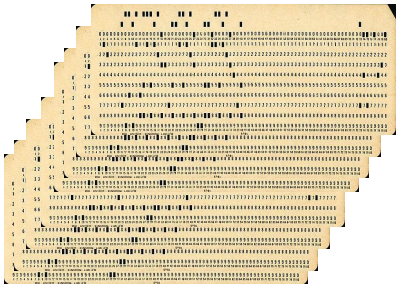
Aparté sur les types de mémoires persistantes



Type de support	Exemple	Capacité typique approximative
Mécanique	Carte perforée	Plusieurs dizaines d'octets
Magnétique	Disquette	1.44 MB (3.5 pouces)
Magnétique	Disque dur	Quelques TB
Magnétique	Bande magnétique	Plusieurs TB
Optique	CD	700 MB
Optique	DVD	4.7 GB (simple couche)
Optique	Blu-Ray	Jusqu'à 100 GB
Flash	SSD	Quelques TB
Flash	Clé USB	Centaines de Gb à 2 TB

Aparté sur les types de mémoires persistantes

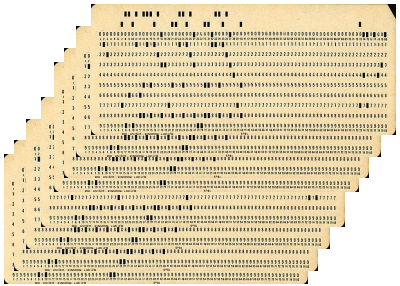
2 MB ~ 200’000 cartes perforées

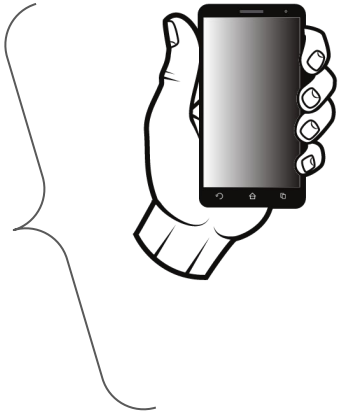


(~ 2 km de haut)

Aparté sur les types de mémoires persistantes

1 TB ~ 100 milliards de cartes perforées



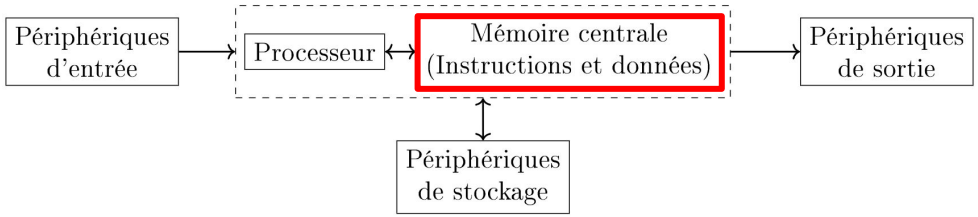


La mémoire centrale | **Terminologie**

- Aussi nommée "mémoire vive" ou "RAM" pour des raisons historiques.
- Random Access Memory : accès dans un **ordre quelconque** aux données, par opposition aux mémoires qui nécessitent de consulter les données dans un ordre précis.
- Mémoire "vive" car opposé aux mémoires "mortes", accessibles en lecture seulement et souvent nommées ROM (Read Only Memory).
- Nous préférons le terme "mémoire centrale".

La mémoire centrale | **Données et instructions**

- Les instructions sont des données "comme les autres", stockées dans la mémoire centrale de façon binaire, par mots entiers (typiquement 32 ou 64 bits à la fois).

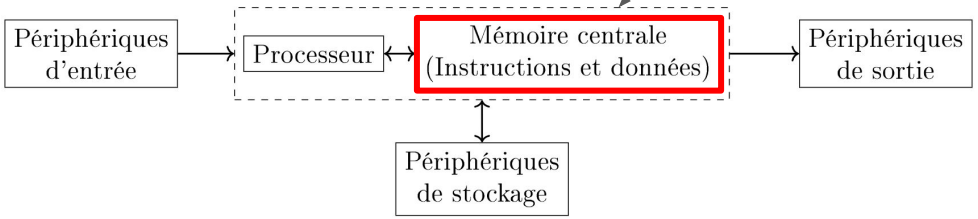


La mémoire centrale | **Données et instructions**

- Les instructions sont des données "comme les autres", stockées dans la mémoire centrale de façon binaire, par mots entiers (typiquement 32 ou 64 bits à la fois).

- Les données sont adressées de façon ordonnée (portent un "numéro" pour qu'on les retrouve).

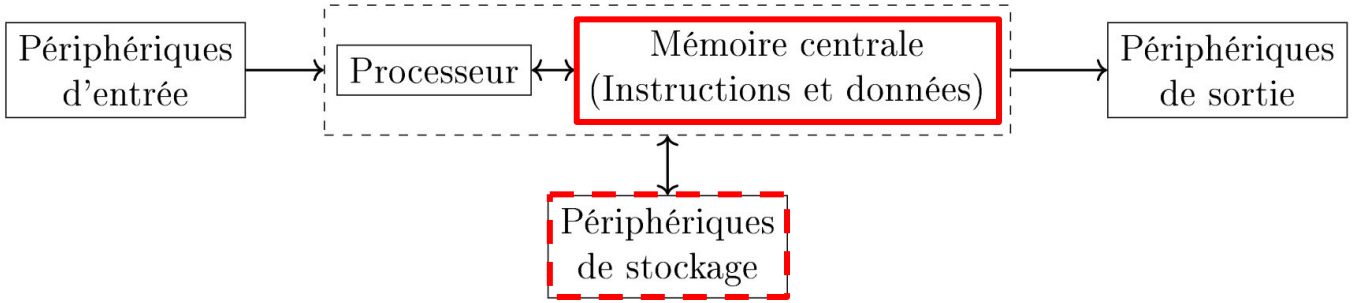
Adresse	0	1	2	3	...
Mot	10011001	11011001	00011000	00010001	...



La mémoire centrale | **Note sur l'amorçage**

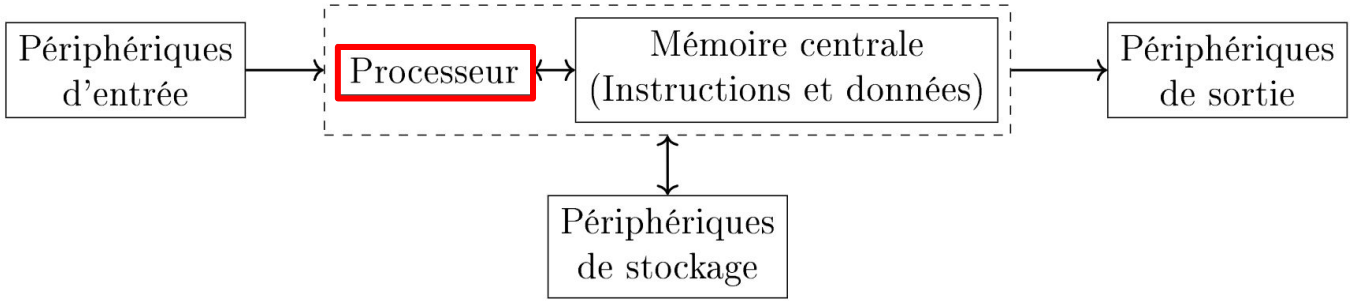
- La mémoire centrale étant volatile, un certain **minimum vital** est stocké dans une mémoire morte à proximité afin de d'amorcer le démarrage ("**boot**").

- **BIOS** (Basic Input/Output System) est un programme fixe (*firmware*) intégré au sein du matériel pour une interaction minimale avec la machine.



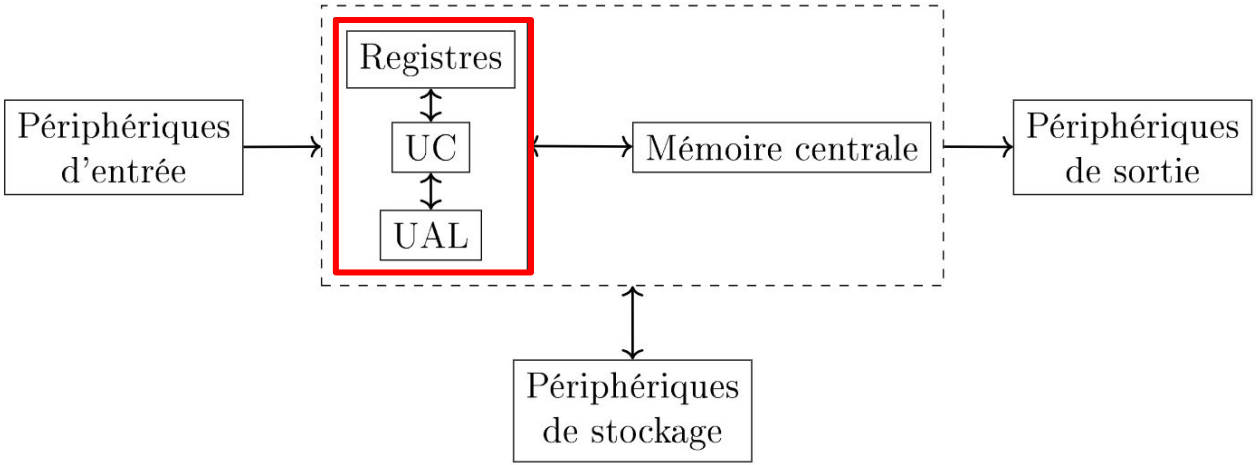
Le processeur

- Partie de la machine où l'information est **traitée**.
- Central Processing Unit (**CPU**) en anglais.
- Est composé de plusieurs sous-parties.



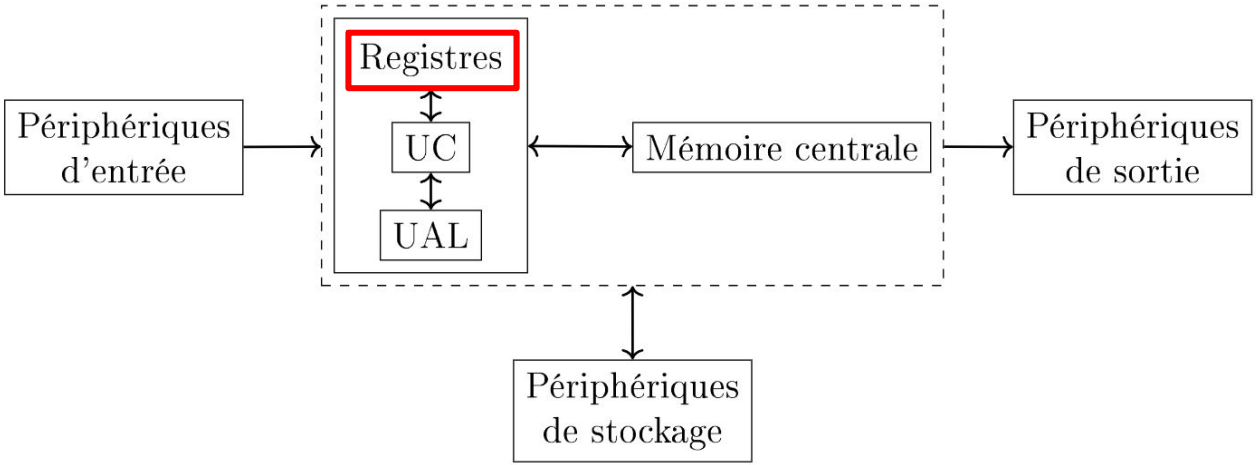
Le processeur

- **Registres** : unités de mémoire (données en attente de lecture/écriture).
- **Unité de contrôle (UC)** : décode les instructions, dirige le tout.
- **Unité arithmétique et logique (UAL)** : effectue les opérations.



Le processeur | **Les registres**

- Unités de mémoire extrêmement **rapides** et **restreintes**.
- Stockage **temporaire** de valeurs intermédiaires et données en attente de transfert vers d'autres éléments.
- Regroupées au sein d'un **banc de registres**.



Le processeur | **Les registres**

- Si les registres sont si performants, pourquoi toute la mémoire centrale n’est-elle pas faite de registres ?

Le processeur | **Les registres**

- Si les registres sont si performants, pourquoi toute la mémoire centrale n’est-elle pas faite de registres ?

- Parce que c’est grâce au fait qu’il y en a peu (quelques dizaines) qu’ils sont rapides !

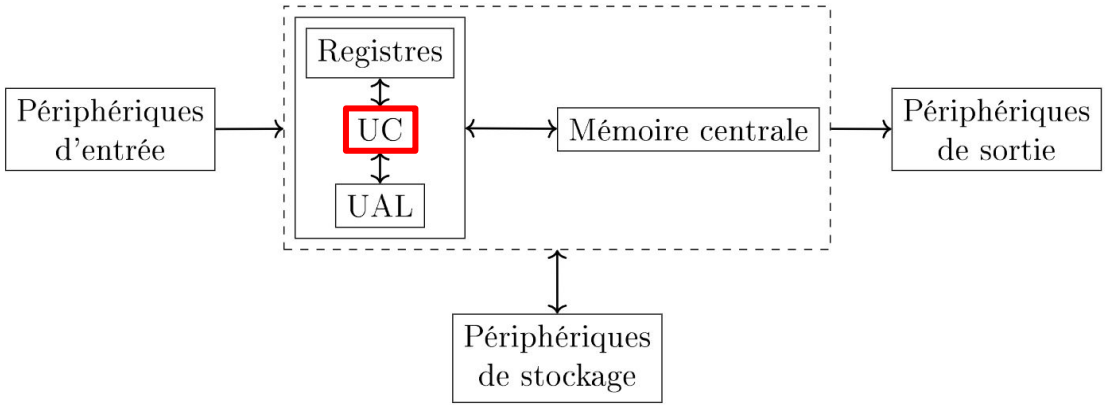
- Des **multiplexeurs** sont utilisés pour **adresser** les registres. Plus il y a de registres, plus le signal doit traverser de portes logiques pour coder/décoder une adresse.

- Autres raisons : coût, proximité du processeur, stabilité du signal électrique.

Le processeur | **L'unité de contrôle**

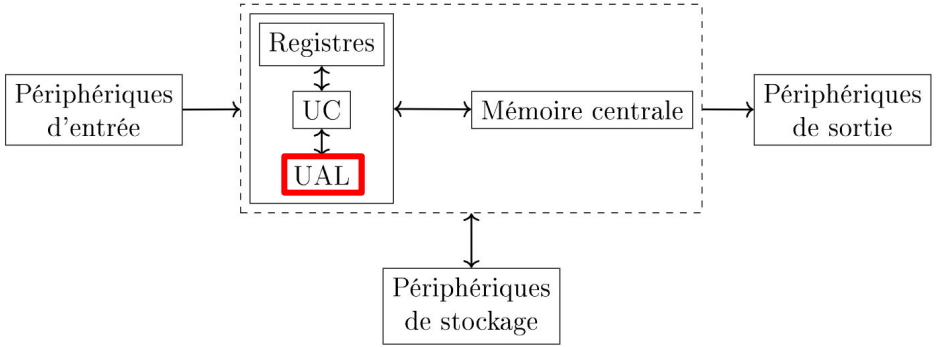
- Pilote le CPU et son interaction avec les autres composants en envoyant des **signaux**. Exemple : spécifier à l'UAL le type d'opération à effectuer.

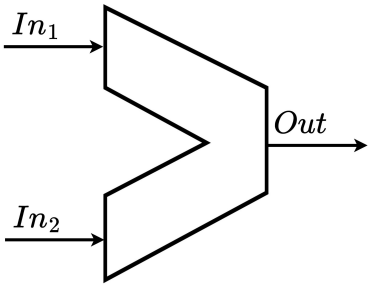
- Synchronise le traitement (les choses doivent se faire dans un ordre précis).



Le processeur | **L'unité arithmétique et logique**

- Circuit combinatoire effectuant les opérations arithmétiques (add, sub, mult, div, …) et logiques (NOT, AND, OR, …)
- Prend toujours deux mots en entrée et donne un mot en sortie.
- Arithmetic Logic Unit (**ALU**) en anglais.

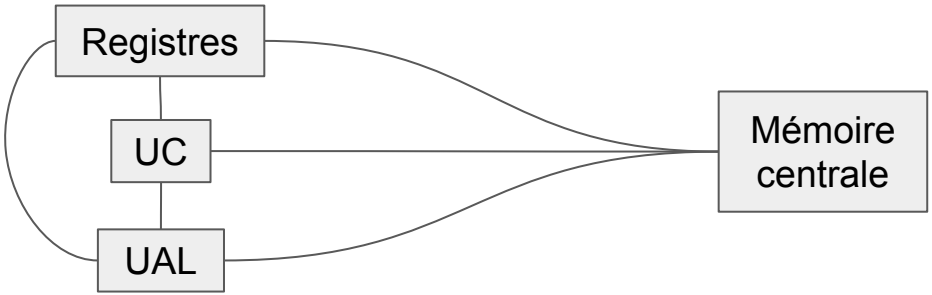




Symbole de l'UAL

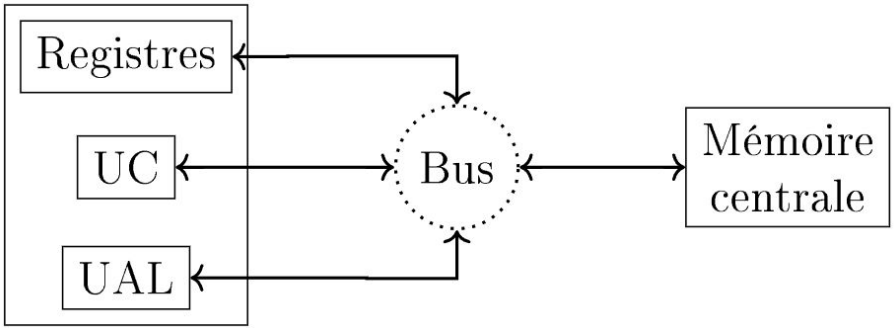
Flux de l’information dans le processeur

- Les composants **communiquent** entre eux en envoyant des données dans le **bus système**.
- Le bus système est un ensemble de connexions.



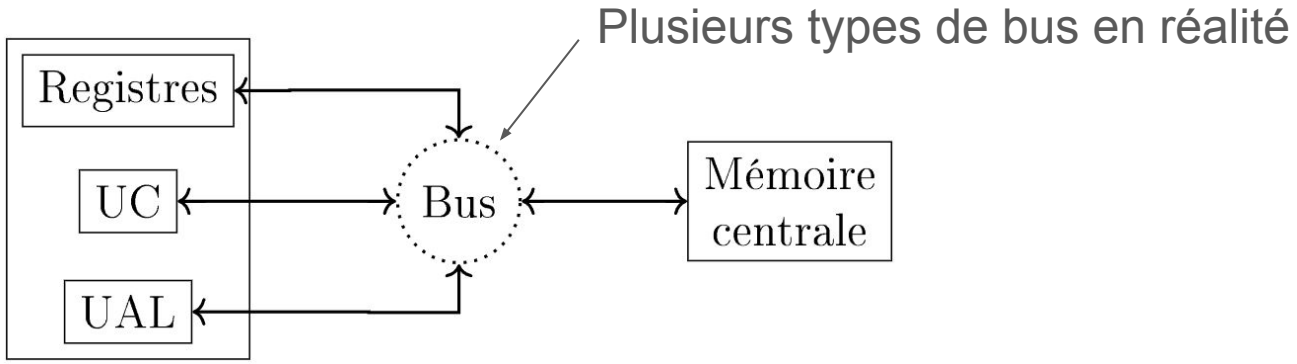
Flux de l’information dans le processeur

- Les composants **communiquent** entre eux en envoyant des données dans le **bus système**.
- Le bus système est un ensemble de connexions.
- Représentation abstraite :



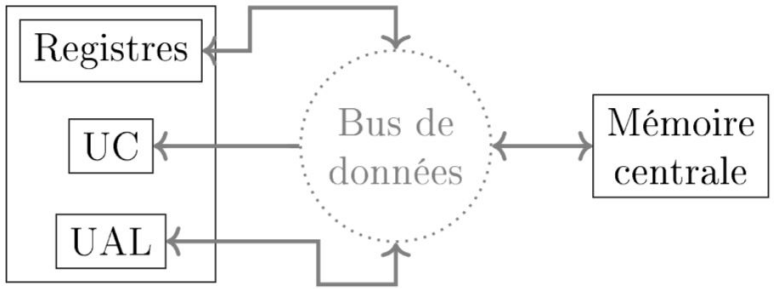
Types de signaux

- Il y a trois types de signaux et trois types de bus :
  - données
  - adresses
  - contrôle



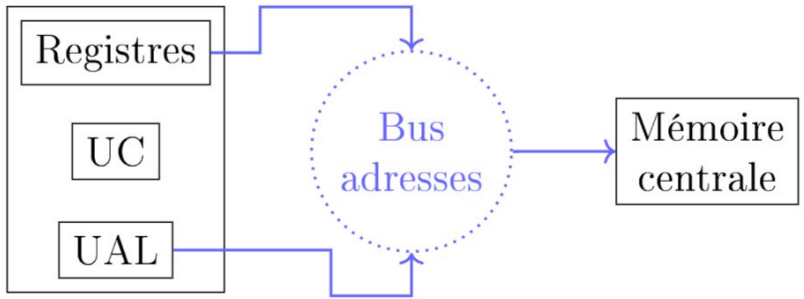
Flux de données

- **Données** lues en mémoire centrale (MC) et déposées dans registres.
- Calculs effectués sur les données dans l'UAL.
- **Résultats** déposées dans registres et écrits en MC.
- Attention : les **instructions** elles-mêmes sont des données ! Même l'UC doit donc en recevoir.



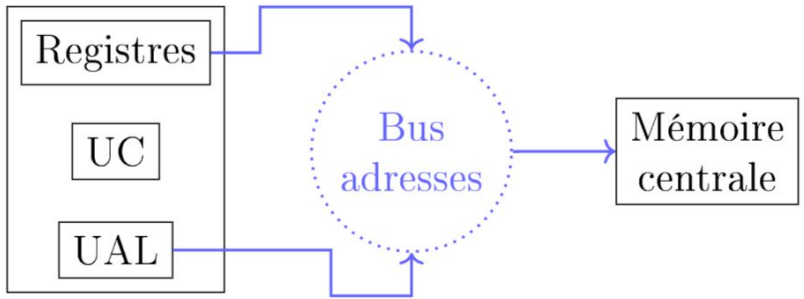
Flux d’adresses

- La MC reçoit des adresses et répond éventuellement par des données.
- Les registres contiennent très souvent des adresses attendant d’être envoyées vers la MC.
- L’UAL est amenée à calculer des adresses.



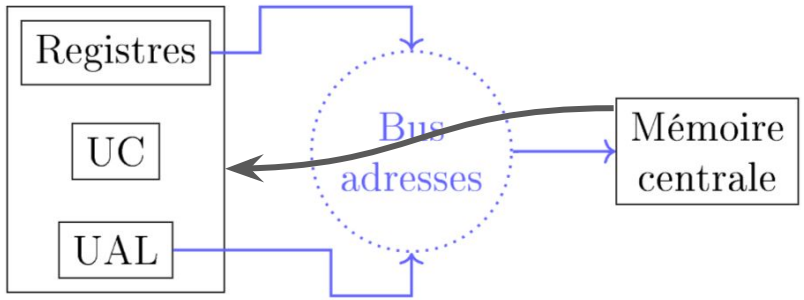
Flux d’adresses

- Mais alors, d’où viennent les adresses, si ce n’est pas de la MC ?



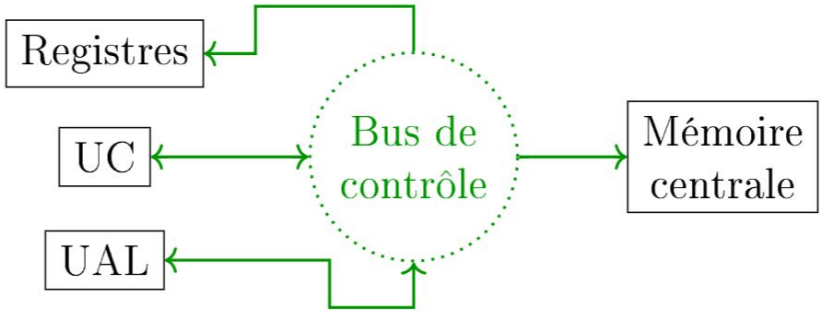
Flux d’adresses

- Mais alors, d’où viennent les adresses, si ce n’est pas de la MC ?
- Réponse : Certaines données (sous-partie d’une instruction, résultats de calcul) peuvent entrer dans un registre en tant que données et en sortir en tant qu’adresses.



Flux de contrôle

- MC consulte le signal pour savoir si les adresses sont à **lire** ou à **écrire**.
- Registres consultent également le signal pour se **synchroniser** correctement.
- UAL a besoin de savoir le **type d'opération** à effectuer.
- UAL doit également informer du résultat des opérations de **branchement**.



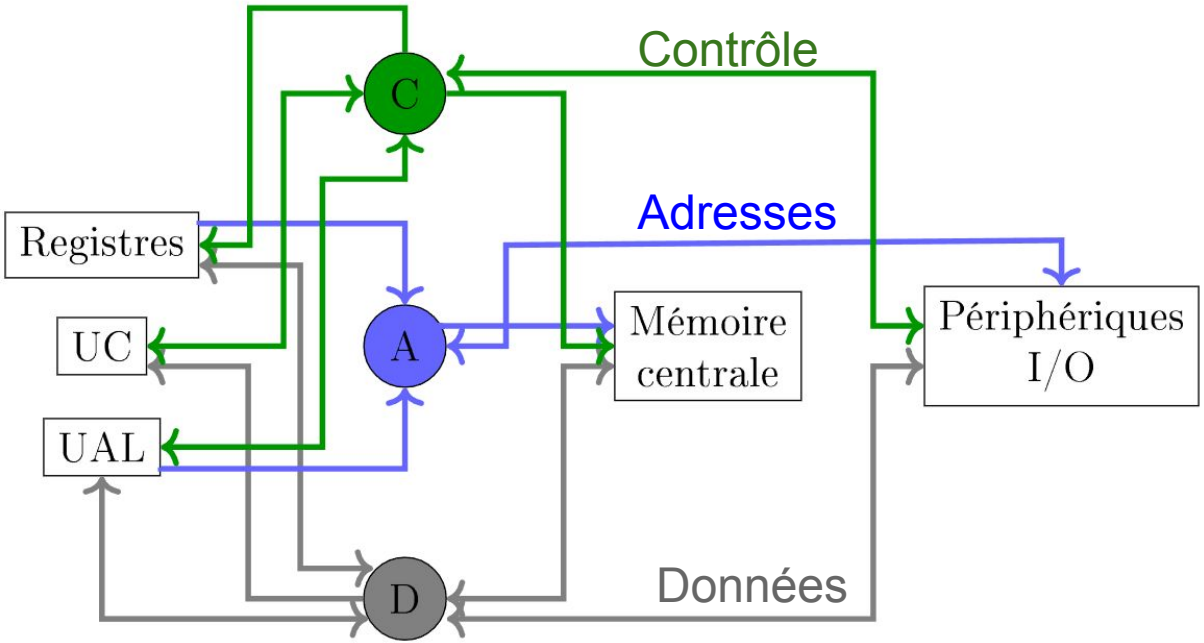
Flux d’information et périphériques

- Quand un périphérique a besoin de l’attention du processeur, un signal d’**interruption** est envoyé.

- **Synchronisation** des autres éléments : attendre le bon moment pour parler/écouter un périphérique.

- Certains périphériques utilisent des bus qui leurs sont propres (par exemple USB).

Flux d'information à travers les bus | **Résumé**



(Diapositive hors champ)

Le cycle Fetch-Decode-Execute

- Pour des raisons logiques et électriques, les circuits traitent l'information de façon **synchrone**, à chaque **pulsation** d'une horloge (cf. circuits séquentiels).

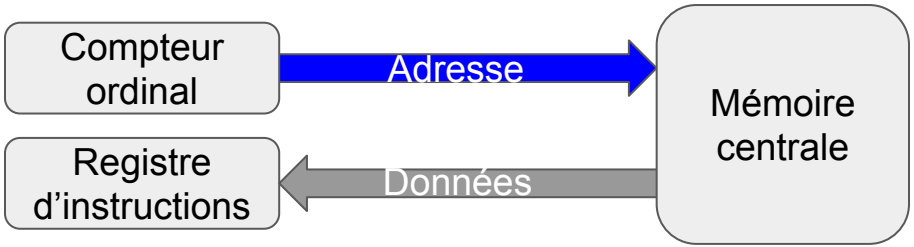
- L'information est traitée par **cycles** de trois étapes qui se suivent en boucle, dans l'ordre : *fetch*, *decode* et *execute*. Une instruction "atomique" est réalisée en un cycle au minimum.

- CPU modernes : des milliards de cycles par seconde (GHz) !
  Hertz signifie 1 / s, c'est une fréquence.

- L'unité de contrôle est responsable du bon fonctionnement du cycle.

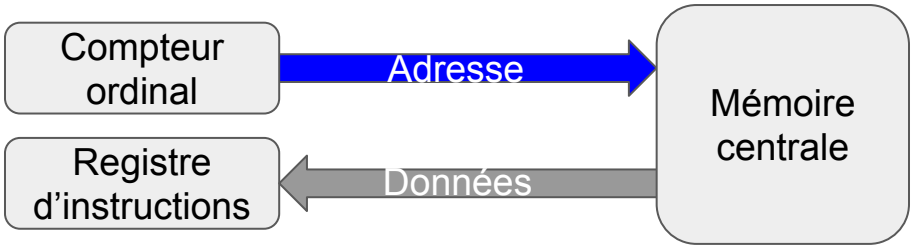
Le cycle Fetch-Decode-Execute | **Étape *fetch***

- Fetch ("aller chercher") : lecture de la **prochaine instruction** en MC.
- Il existe un registre spécial nommé **compteur ordinal** ou **pointeur d'instruction** : il contient l'adresse de la prochaine instruction à lire.
- Autre registre spécial: **registre d'instruction**, contient les données de l'instruction elle-même.



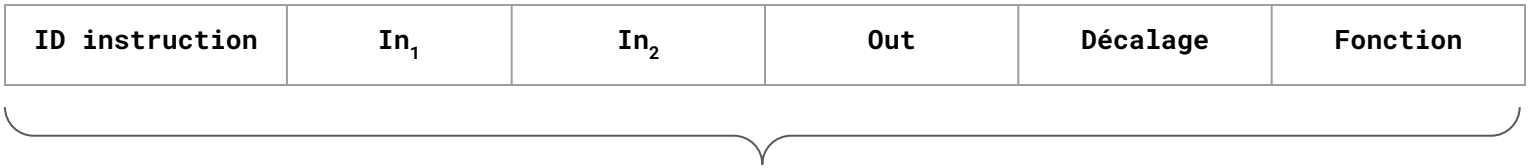
Le cycle Fetch-Decode-Execute | **Étape *fetch* - 3 cas**

- **Séquence** d'instructions : compteur ordinal incrémenté du nombre d'octets correspondant à la taille d'une instruction.
- **Saut** d'instruction : compteur ordinal incrémenté d'une valeur spécifique.
- **Branchement** : compteur ordinal mis à jour en fonction du résultat fourni par l'UAL.



Le cycle Fetch-Decode-Execute | **Étape *decode***

- Étape centrée sur l'UC, qui **interprète l'instruction** du registre d'instruction.
- Détermination du type d'opération, des registres concernés, et envoi de signaux de contrôle dans le bus.
- L'UC analyse chaque sous-partie de l'instruction :



Taille d'un mot (par exemple 32 bits)
Cf. prochain chapitre.

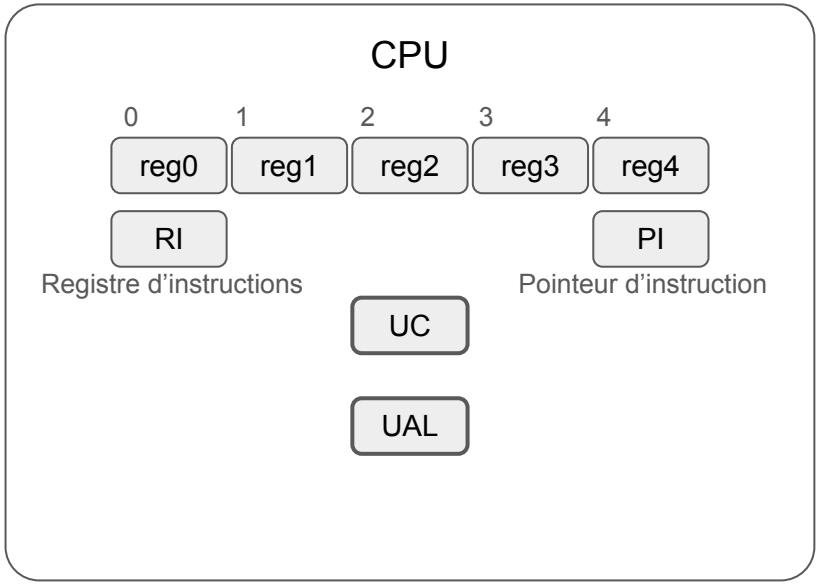
Le cycle Fetch-Decode-Execute | **Étape *execute***

- Les éléments concernés par les signaux de contrôle de l'étape précédente effectuent leur travail.

- Exemples :
  - Calcul au sein de l'UAL
  - Saut inconditionnel modifiant le compteur ordinal
  - Accès à la MC

Exemple de cycle *fetch-decode-execute*

Soustraction** des données des registres **4** et **2** à ranger dans le registre **3

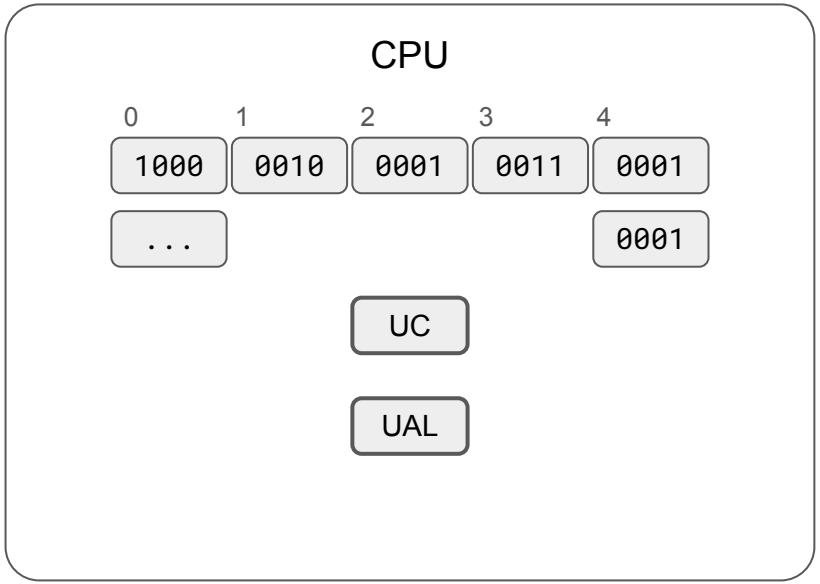


Mémoire centrale

addr	Opcode	In1	In2	Out	Shift	Fct
0000	100001	10101	10101	00101	10000	100010
0001	000000	00100	00010	00011	00000	100010
0010	100000	00000	11111	00100	00000	100010
0011	000001	00111	10111	00100	00000	100010
0100	100001	10101	10001	00100	00000	100010

Étape fetch

Soustraction** des données des registres **4** et **2** à ranger dans le registre **3

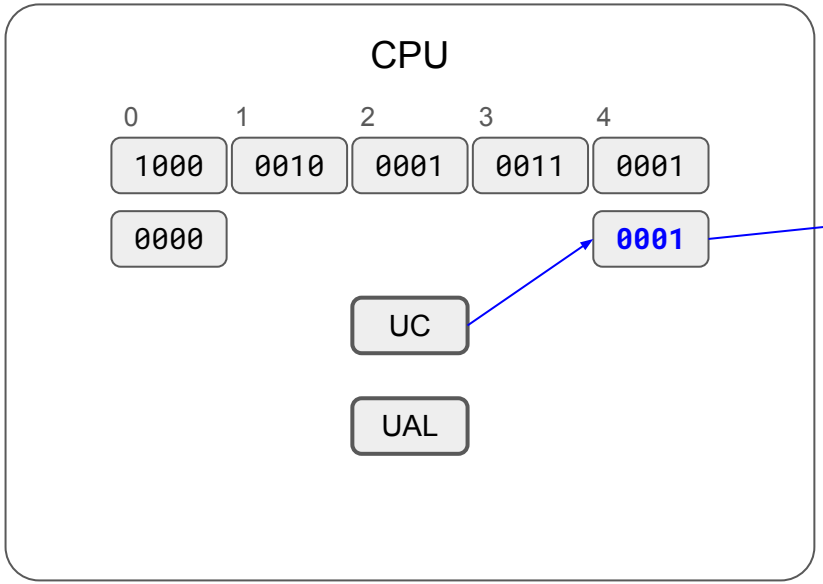


Mémoire centrale

addr	Opcode	In1	In2	Out	Shift	Fct
0000	100001	10101	10101	00101	10000	100010
0001	000000	00100	00010	00011	00000	100010
0010	100000	00000	11111	00100	00000	100010
0011	000001	00111	10111	00100	00000	100010
0100	100001	10101	10001	00100	00000	100010

Étape fetch

Soustraction** des données des registres **4** et **2** à ranger dans le registre **3

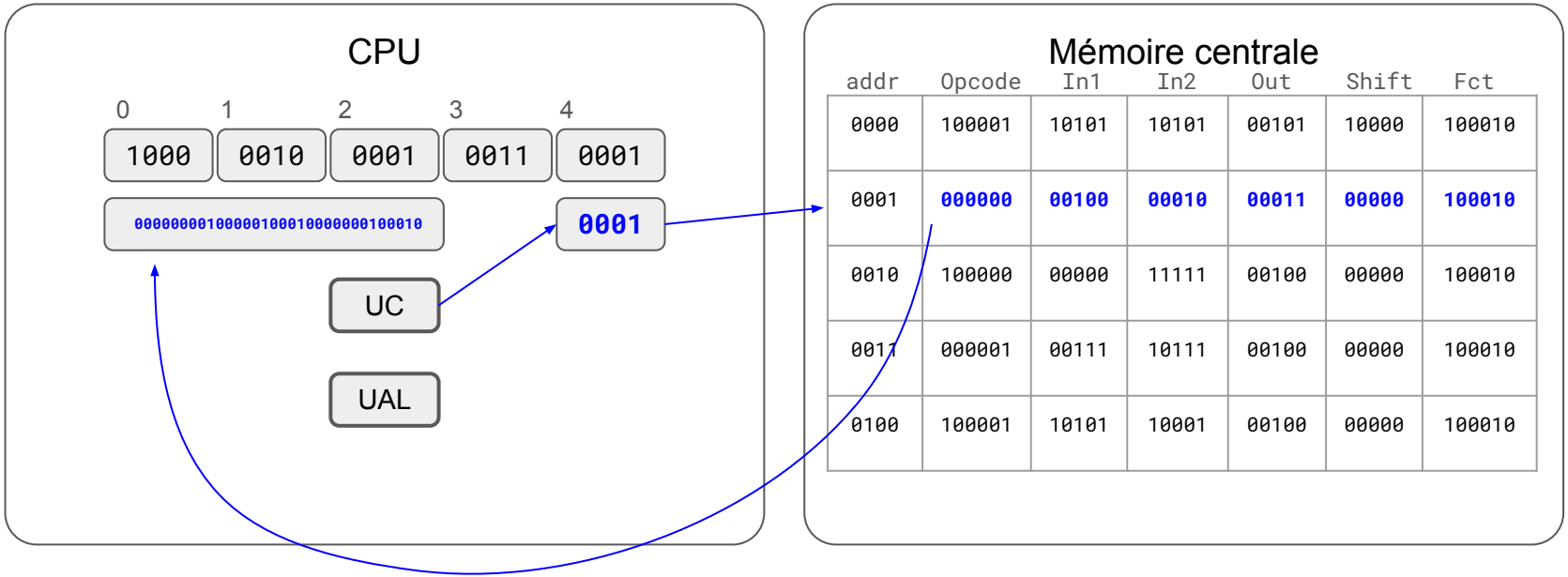


Mémoire centrale

addr	Opcode	In1	In2	Out	Shift	Fct
0000	100001	10101	10101	00101	10000	100010
0001	000000	00100	00010	00011	00000	100010
0010	100000	00000	11111	00100	00000	100010
0011	000001	00111	10111	00100	00000	100010
0100	100001	10101	10001	00100	00000	100010

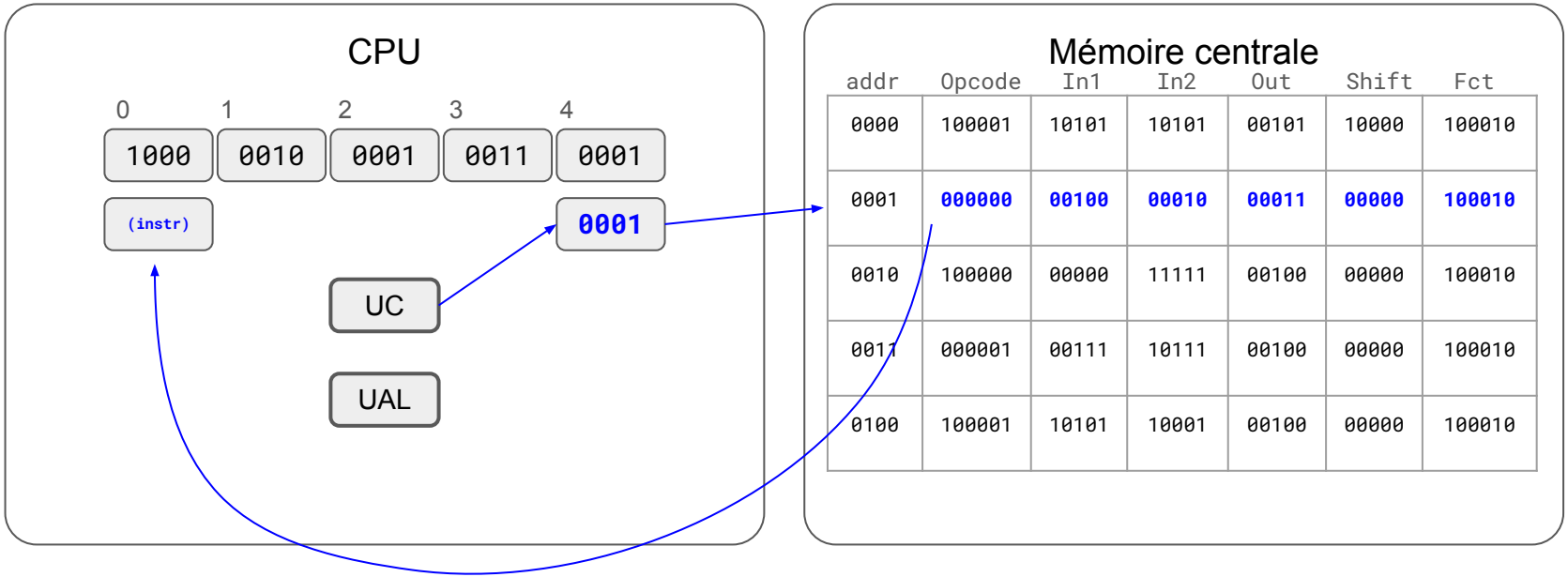
Étape fetch

Soustraction** des données des registres **4** et **2** à ranger dans le registre **3



Étape fetch

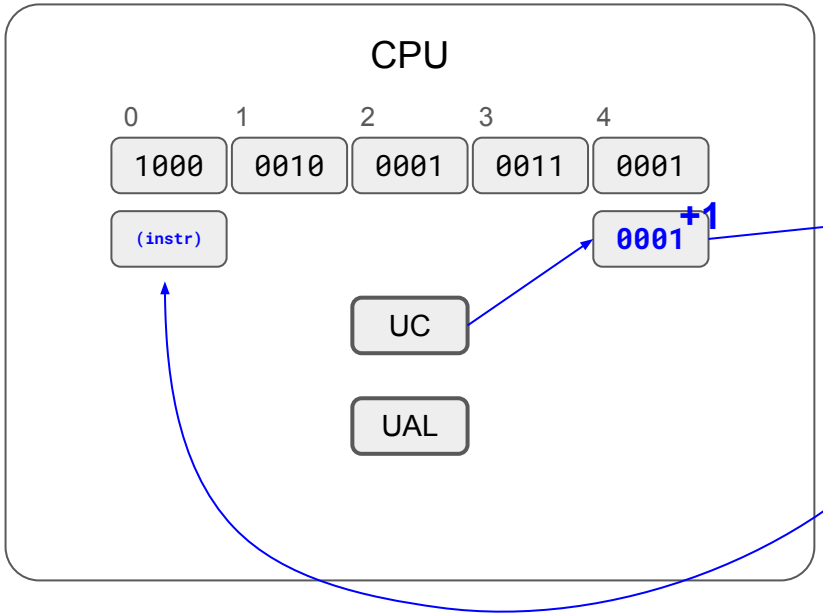
Soustraction** des données des registres **4** et **2** à ranger dans le registre **3



addr	Opcode	In1	In2	Out	Shift	Fct
0000	100001	10101	10101	00101	10000	100010
0001	**000000**	**00100**	**00010**	**00011**	**00000**	**100010**
0010	100000	00000	11111	00100	00000	100010
0011	000001	00111	10111	00100	00000	100010
0100	100001	10101	10001	00100	00000	100010

Étape fetch

Soustraction** des données des registres **4** et **2** à ranger dans le registre **3

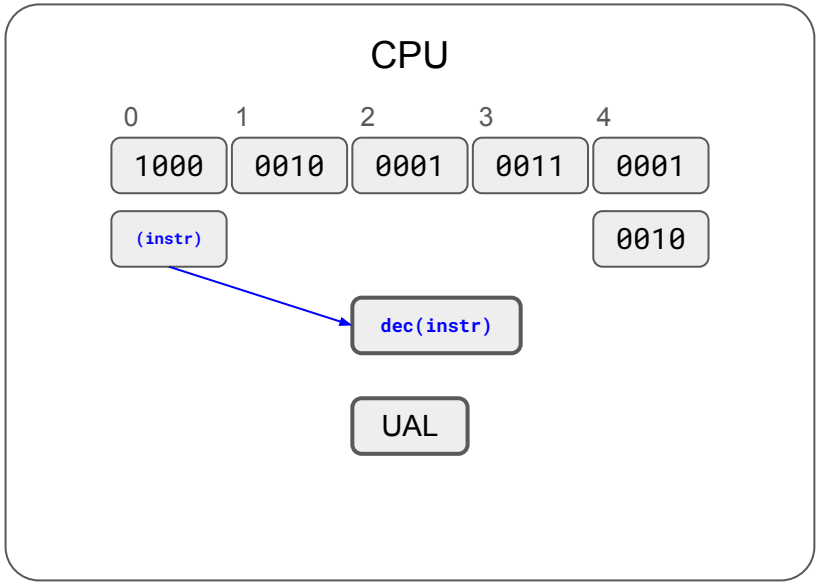


Mémoire centrale

addr	Opcode	In1	In2	Out	Shift	Fct
0000	100001	10101	10101	00101	10000	100010
0001	**000000**	**00100**	**00010**	**00011**	**00000**	**100010**
0010	100000	00000	11111	00100	00000	100010
0011	000001	00111	10111	00100	00000	100010
0100	100001	10101	10001	00100	00000	100010

Étape decode

__Soustraction__ des données des registres __4__ et __2__ à ranger dans le registre __3__

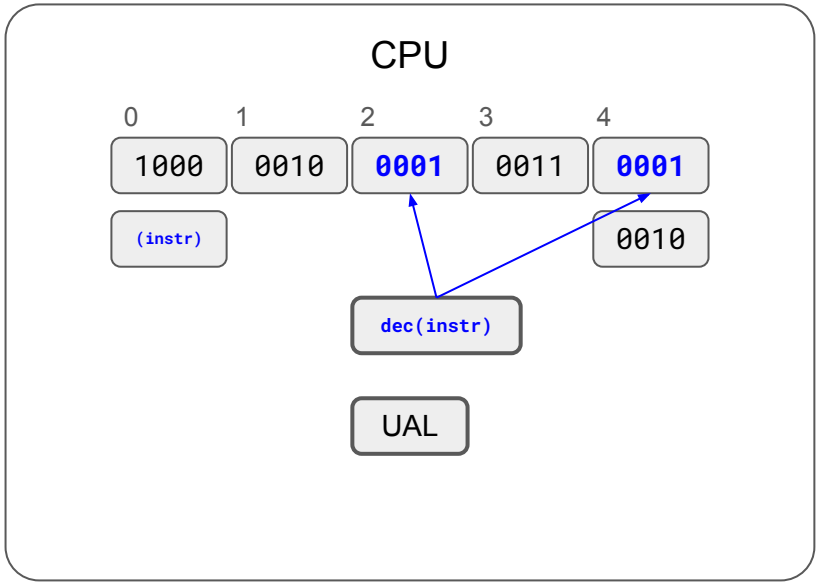


Mémoire centrale

addr	Opcode	In1	In2	Out	Shift	Fct
0000	100001	10101	10101	00101	10000	100010
0001	000000	00100	00010	00011	00000	100010
0010	100000	00000	11111	00100	00000	100010
0011	000001	00111	10111	00100	00000	100010
0100	100001	10101	10001	00100	00000	100010

Étape decode

Soustraction** des données des registres **4** et **2** à ranger dans le registre **3

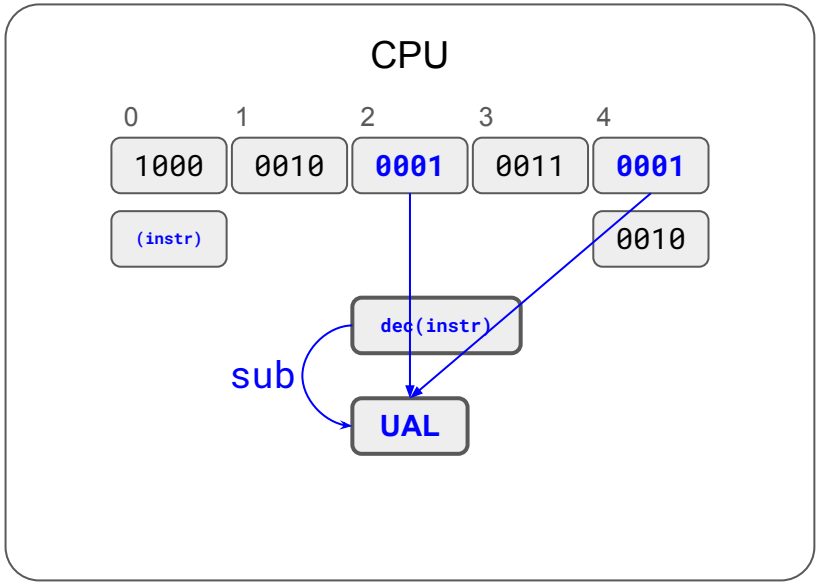


Mémoire centrale

addr	Opcode	In1	In2	Out	Shift	Fct
0000	100001	10101	10101	00101	10000	100010
0001	000000	00100	00010	00100	00000	100010
0010	100000	00000	11111	00100	00000	100010
0011	000001	00111	10111	00100	00000	100010
0100	100001	10101	10001	00100	00000	100010

Étape execute

Soustraction** des données des registres **4** et **2** à ranger dans le registre **3

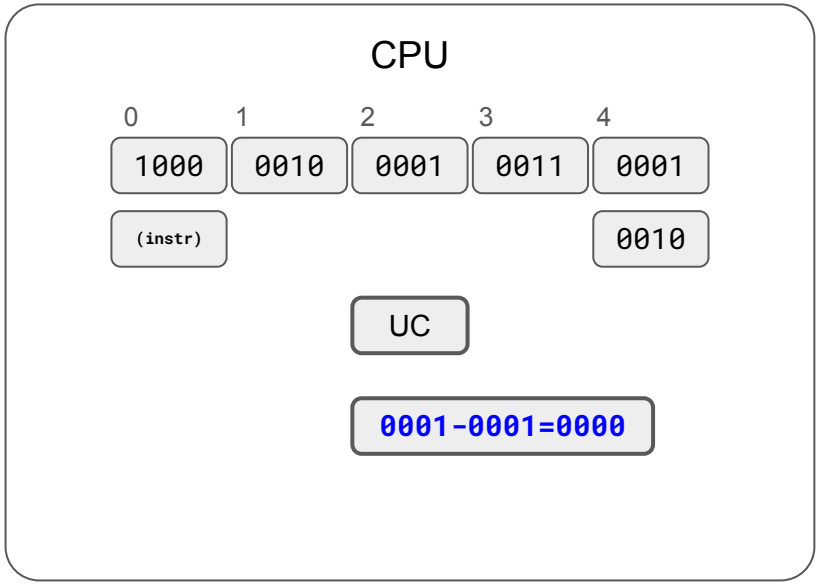


Mémoire centrale

addr	Opcode	In1	In2	Out	Shift	Fct
0000	100001	10101	10101	00101	10000	100010
0001	000000	00100	00010	00011	00000	100010
0010	100000	00000	11111	00100	00000	100010
0011	000001	00111	10111	00100	00000	100010
0100	100001	10101	10001	00100	00000	100010

Étape execute

Soustraction** des données des registres **4** et **2** à ranger dans le registre **3

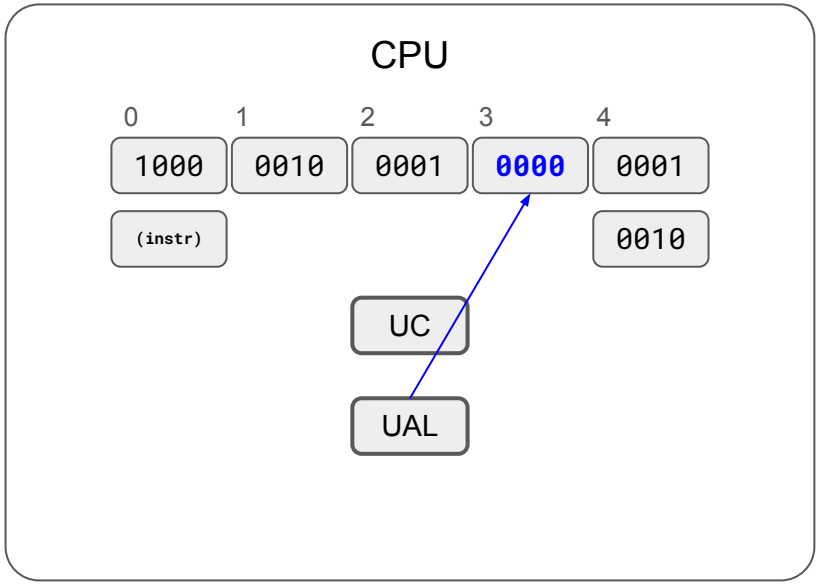


Mémoire centrale

addr	Opcode	In1	In2	Out	Shift	Fct
0000	100001	10101	10101	00101	10000	100010
0001	000000	00100	00010	00011	00000	100010
0010	100000	00000	11111	00100	00000	100010
0011	000001	00111	10111	00100	00000	100010
0100	100001	10101	10001	00100	00000	100010

Étape execute

__Soustraction__ des données des registres __4__ et __2__ à ranger dans le registre __3__

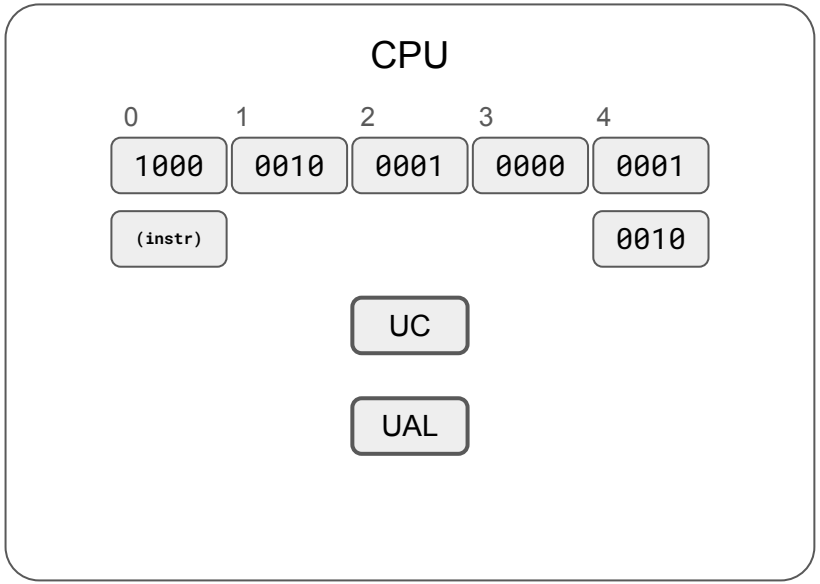


Mémoire centrale

addr	Opcode	In1	In2	Out	Shift	Fct
0000	100001	10101	10101	00101	10000	100010
0001	000000	00100	00010	00011	00000	100010
0010	100000	00000	11111	00100	00000	100010
0011	000001	00111	10111	00100	00000	100010
0100	100001	10101	10001	00100	00000	100010

Fin du cycle

Soustraction** des données des registres **4** et **2** à ranger dans le registre **3



Mémoire centrale

addr	Opcode	In1	In2	Out	Shift	Fct
0000	100001	10101	10101	00101	10000	100010
0001	000000	00100	00010	00011	00000	100010
0010	100000	00000	11111	00100	00000	100010
0011	000001	00111	10111	00100	00000	100010
0100	100001	10101	10001	00100	00000	100010

Le cycle Fetch-Decode-Execute | **Étape *execute***

- NB : le fait de charger des données dans les registres depuis la MC (load), ou à l'inverse d'écrire en MC (store), constitue un type d'instruction.

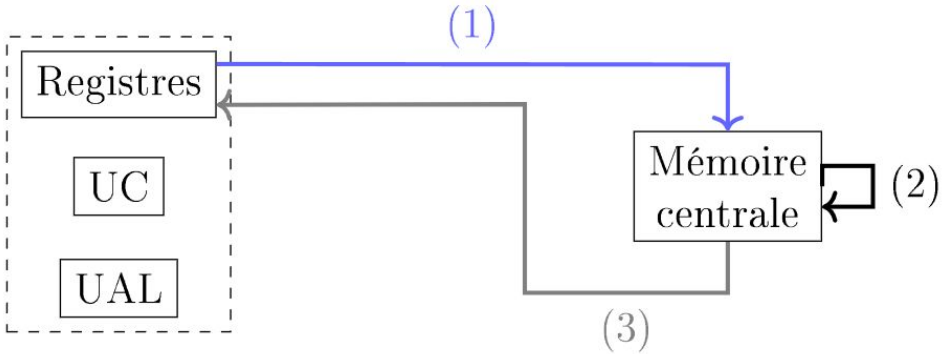
- Dans l'exemple précédent, des instructions précédentes avaient chargé les bonnes valeurs dans les registres.

- Et si le travail à effectuer prend plusieurs cycles ?

Le cycle Fetch-Decode-Execute | **Gestion des cycles**

- Et si le travail à effectuer prend plusieurs cycles ?

- Exemple : accès à la mémoire, où chaque étape peut prendre de nombreux cycles CPU.



Le cycle Fetch-Decode-Execute | **Gestion des cycles**

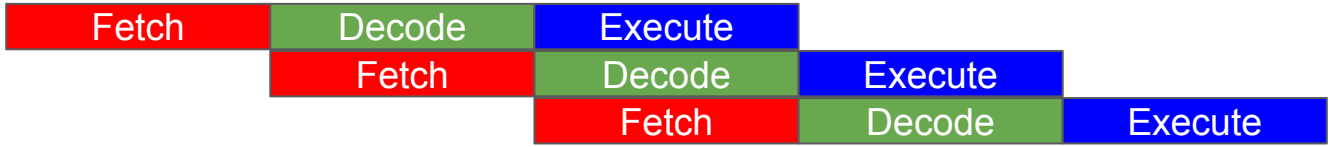
- Et si le travail à effectuer prend plusieurs cycles ?
- Processeurs modernes mettent en place plusieurs stratégies pour anticiper le travail à faire et éviter d’attendre "inutilement" :

Le cycle Fetch-Decode-Execute | **Gestion des cycles**

- Et si le travail à effectuer prend plusieurs cycles ?
- Processeurs modernes mettent en place plusieurs stratégies pour anticiper le travail à faire et éviter d'attendre "inutilement" :
  - **Spéculation** sur les opérations à venir → **Pipelining**

Le cycle Fetch-Decode-Execute | **Gestion des cycles**

- Et si le travail à effectuer prend plusieurs cycles ?
- Processeurs modernes mettent en place plusieurs stratégies pour anticiper le travail à faire et éviter d'attendre "inutilement" :
  - **Spéculation** sur les opérations à venir → **Pipelining**



Le cycle Fetch-Decode-Execute | **Gestion des cycles**

- Et si le travail à effectuer prend plusieurs cycles ?
- Processeurs modernes mettent en place plusieurs stratégies pour anticiper le travail à faire et éviter d'attendre "bêtement" :
  - **Spéculation** sur les opérations à venir → **Pipelining**
  - Exécution dans le **désordre** si possible (instructions indépendantes).
  - **Multithreading** : un autre bout du programme (indépendant de celui-ci) peut être exécuté.
- Nous reviendrons sur les différentes formes de traitement parallèle des données au moment d'aborder les systèmes d'exploitation.

Retour au sujet de la mémoire

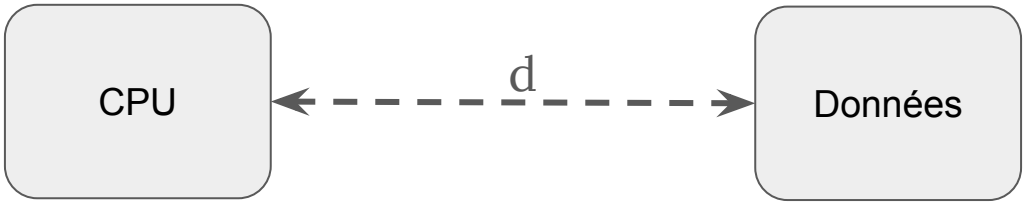
- Le transfert des données n'étant pas instantané, l'attente est inévitable.
- Cependant, en choisissant judicieusement la façon des stocker les données à **proximité** du CPU, on peut limiter la casse.
- Il est donc tentant d'avoir plusieurs types de mémoire.

Note sur la nécessité de différents types de mémoire

- Jusqu'ici, on a parlé de différents éléments stockant des états binaires :
  - Les registres (au sein même du processeur).
  - La mémoire centrale (voisine directe du processeur).
  - Le stockage auxiliaire ou mémoire de masse (périphérique).

- En pratique :
  - **Adressage** impacte le temps d'attente des données ("**latence**").
  - **Technologie** utilisée (transistors ? condensateurs ? autre ?) impacte latence et **prix**.
  - **Distance** au CPU impacte latence, même si les données transitent à une vitesse très élevée.

Note sur le temps de latence et la fréquence du CPU



- Si le signal est transmis à vitesse v et le chemin à parcourir est environ $2d$, alors le temps de latence est $t = 2d \,/\, v$.
- Par conséquent, au-delà de la fréquence $f = 1/\, t = v \,/\, (2d)$, le signal ne peut pas "suivre" le CPU.
- Supposons $d = 10$ cm et $v = ⅔\ c$, alors : $f = 2 \cdot 10^8 \,/\, 0.2 \approx 1\ \text{GHz}$.

Note sur la nécessité de différents types de mémoire

- Rappel :
  - Les registres (au sein même du processeur).
  - La mémoire centrale (voisine directe du processeur).
  - Le stockage auxiliaire (périphérique).

- Dans l'absolu, un seul type de mémoire suffirait, mais pour les raisons pratiques discutées il est extrêmement souhaitable de **hiérarchiser** les mémoires : registres, MC et éventuellement mémoire de masse (auxiliaire).

- Sur les ordinateurs modernes, on va même plus loin encore…

Temps de latence et capacité

- Latence : durée nécessaire pour accéder à une donnée en mémoire.
- Capacité : quantité de données stockable en mémoire.
- Problème : grand écart de valeurs entre registres et MC.

Type d'accès	Latence	Latence [Cycles CPU]	Capacité
Registre	0.3 ns	1	1 mot
Écart à combler, en latence et en capacité !			
Mémoire centrale	120 ns	400	$\sim$ 10 GB
Mémoire flash SSD	$\sim$ 100 μs	160'000	$\sim$ 1 TB
Disque dur magnétique	1-10 ms	3'000'000	$\sim$ 1 TB

D'après Brendan Gregg. Systems Performance : Enterprise and the Cloud. Prentice Hall, oct 2013.

Hiérarchie de mémoires

- Différents éléments stockant des états binaires :
  - Les registres (au sein même du processeur).
  - ???  mi-chemin entre registres et mémoire centrale
  - La mémoire centrale (voisine directe du processeur).
  - Le stockage auxiliaire (périphérique).

Hiérarchie de mémoires

- On peut ajouter des mémoires à différents niveaux hiérarchiques :
  - Les registres (au sein même du processeur).
  - **Mémoires cache.** ⇦
  - La mémoire centrale (voisine directe du processeur).
  - Le stockage auxiliaire (périphérique).

Type d'accès	Latence	Latence [Cycles CPU]	Capacité
Registre	0.3 ns	1	1 mot
Cache L1	0.9 ns	3	~ 10 kB
Cache L2	2.8 ns	9	~ 500 kB
Cache L3	12.9 ns	43	~ 4 MB
Mémoire centrale	120 ns	400	~ 10 GB
Mémoire flash SSD	~ 100 μs	160'000	~ 1 TB
Disque dur magnétique	1-10 ms	3'000'000	~ 1 TB

D'après Brendan Gregg. Systems Performance : Enterprise and the Cloud. Prentice Hall, oct 2013. (sauf encadré)

Mémoires cache

- La mémoire cache est typiquement divisée en plusieurs niveaux (**L**evels): L1, L2 et L3.
- L1 est plus petite et plus rapide, L3 plus grande et plus lente d'accès.
- L1 et L2 souvent incluses **au sein** même du **CPU**.
- L3 est souvent partagée par plusieurs UAL dans les processeurs multicoeurs.

Mémoires cache | **Heuristique de fonctionnement**

- Comment la mémoire cache permet-elle d'améliorer les performances ? Grâce à une **heuristique** sur le fonctionnement habituel des algorithmes et de leur implémentation sous forme de programmes.

- Une donnée accédée **à un instant** le sera sûrement à nouveau bientôt.

- Une donnée accédée **à un endroit** verra sûrement ses voisines accédées bientôt.

- Ainsi, des allers-retours avec la mémoire centrale sont économisés.

Mémoires cache | **Heuristique, exemple de code**

```
# NB : Mouvement rectiligne uniforme juste pour l'exemple
pos = [0, 0, 0] # position 3D d'un mobile
vel = [12, -3, 4] # vitesse 3D d'un mobile
dt = 0.001 # petit pas de temps
for t in range(1000):
    pos[0] = pos[0] + vel[0] * dt
    pos[1] = pos[1] + vel[1] * dt
    pos[2] = pos[2] + vel[2] * dt
```

Mémoires cache | **Heuristique, exemple de code**

```
pos = [0, 0, 0] # position 3D d'un mobile
vel = [12, -3, 4] # vitesse 3D d'un mobile
dt = 0.001 # petit pas de temps
for t in range(1000):
    pos[0] = pos[0] + vel[0] * dt
    pos[1] = pos[1] + vel[1] * dt
    pos[2] = pos[2] + vel[2] * dt
```

Accédé 2000 fois en peu de temps

Mémoires cache | **Heuristique, exemple de code**

```
pos = [0, 0, 0] # position 3D d'un mobile
vel = [12, -3, 4] # vitesse 3D d'un mobile
dt = 0.001 # petit pas de temps
for t in range(1000):
    pos[0] = pos[0] + vel[0] * dt
    pos[1] = pos[1] + vel[1] * dt
    pos[2] = pos[2] + vel[2] * dt
```

Accédé 2000 fois en peu de temps

Et les voisins également !

Mémoires cache

- Réalisées avec des **transistors** (cf. circuits séquentiels, à venir). Typiquement SRAM (*Static* RAM)
- Sont une **copie de la mémoire centrale** mais à "portée de main" du CPU.
- *Dirty bit* utilisé pour indiquer si le cache est **synchronisé** avec la MC.
- Normalement **invisible pour le programmeur**. Géré au niveau hardware pour savoir si les données doivent être synchronisées avec la véritable MC.
- **Complexifie la compréhension** des performances du code !

Mémoires cache

- Normalement **invisible pour le programmeur**.

- **Compactifier** les données en MC augmente les chances de *cache hit*, c'est-à-dire d'utilisation d'une donnée dans la mémoire cache.
  ⇒ Peut impacter indirectement le travail du programmeur dans les codes critiques en performance, dans certains cas très spécifiques.

- À l'inverse un *cache miss* désigne une donnée qu'il faut aller chercher dans la mémoire centrale car elle n'est pas déjà en cache.

- À retenir : ne concerne habituellement pas le programmeur scientifique.

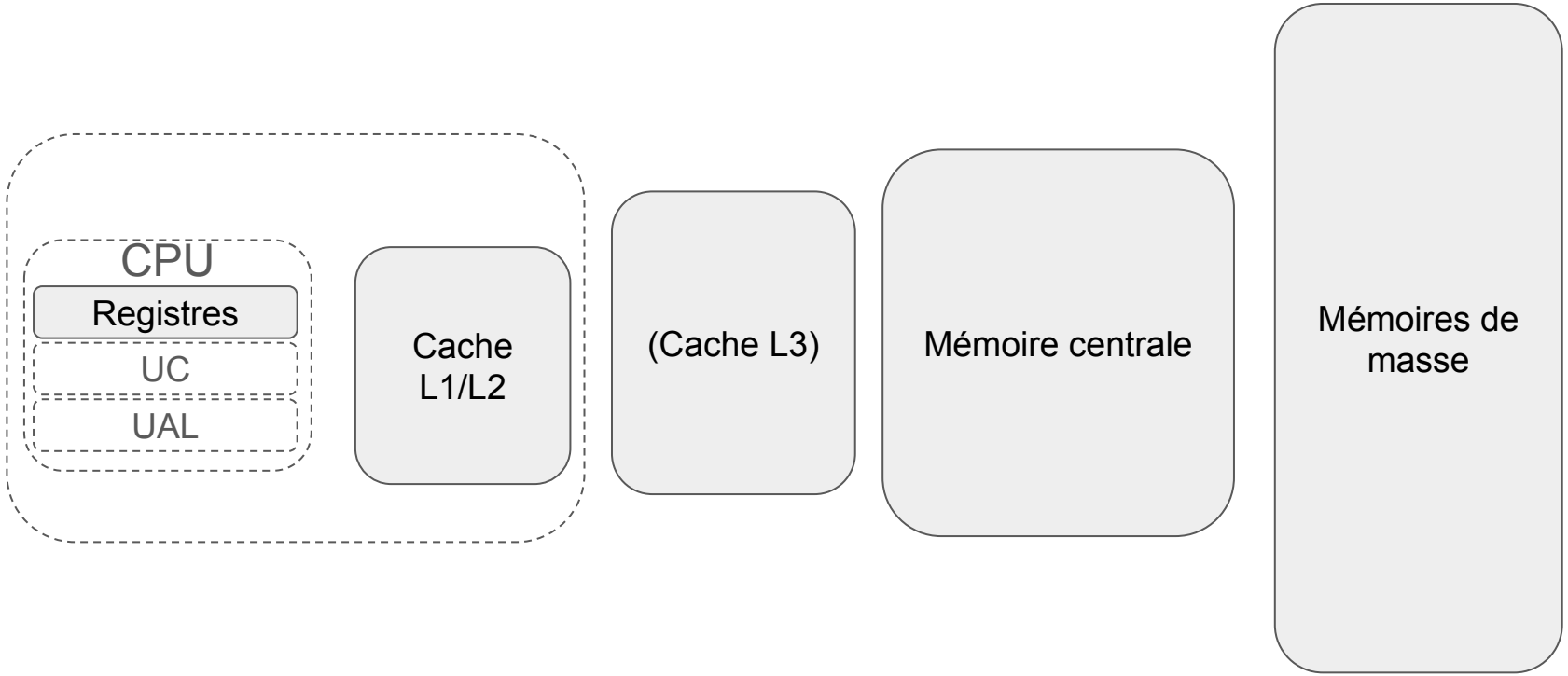
Mémoire centrale

- Souvent réalisées avec des **condensateurs**.
  Typiquement DRAM (*Dynamic* RAM).

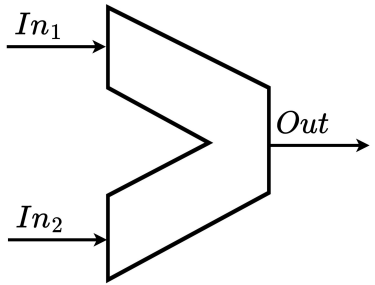
- Utilisation de condensateurs pour garder des charges électriques permet de produire une mémoire plus dense pour le même prix.

- Condensateurs se déchargent (notamment quand on les lit) ⇒ complexifie et ralentit l'accès aux données par rapport à la SRAM.

Mémoires | **Résumé**



Retour au sujet de l’unité arithmétique et logique



Fonctionnement de l’UAL

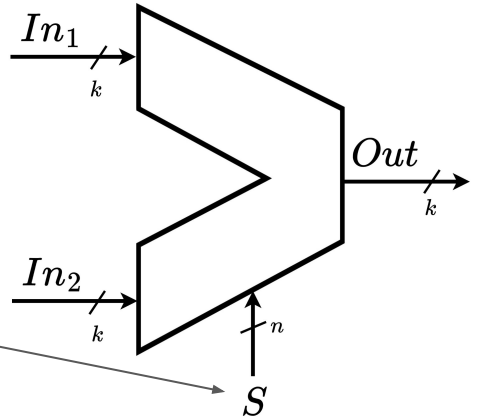
- UAL basiques peuvent n’implémenter que l’addition/soustraction, et exprimer toutes les autres opérations arithmétiques en fonction de cela.

- UAL d’ordinateurs modernes possèdent souvent des circuits spéciaux pour certaines opérations supplémentaires.

- Incluent également des circuits pour les opérations logiques.

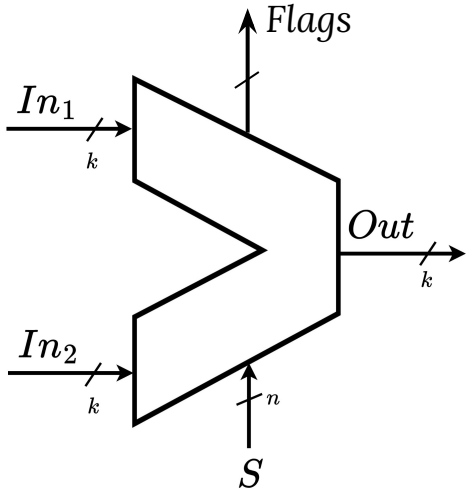
Fonctionnement de l'UAL

- Prend également les bits de sélection de l'opération à effectuer.



Symbole d'UAL travaillant avec des inputs de *k* bits et des codes d'opération de *n* bits.

Fonctionnement de l'UAL



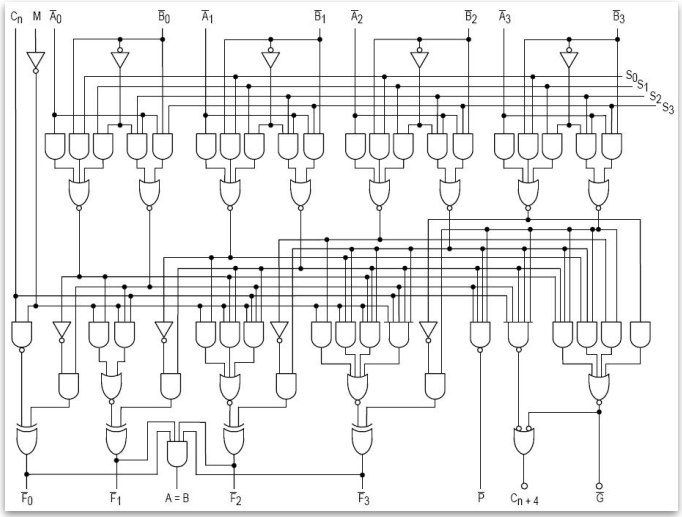
- Prend également les bits de sélection de l'opération à effectuer.

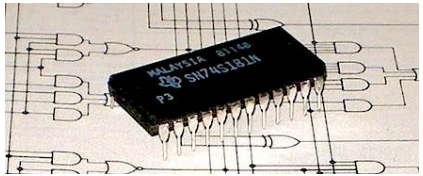
- En plus de l'output, *flags* pour :
  - Débordement
  - Résultat nul
  - Résultat négatif

  Utilisé par exemple pour les comparaisons : si a < b, alors a - b est négatif.

Fonctionnement de l’UAL | **Exemple de l’Intel 74181**

- Première UAL "complète" sur une seule puce.
- Implémente l’addition et la soustraction, mais pas la multiplication et la division.
- Implémente les fonctions logiques classiques.
- Travaille avec des séquences de 4 bits.





Jeu d’instructions d’un processeur

- Une UAL donnée ne peut effectuer "atomiquement" toutes les opérations. Exemple : calcul d’une racine carrée, software VS hardware (les deux sont possibles).

- Approche 1 : le processeur connaît beaucoup d’instructions différentes.

Performance  Économie/simplicité

- Approche 2 : le processeur connaît peu d’instructions différentes.

Performance  Économie/simplicité

Jeu d’instructions d’un processeur

- Une UAL donnée ne peut effectuer "atomiquement" toutes les opérations. Exemple : calcul d’une racine carrée, software VS hardware (les deux sont possibles).

- Approche 1 : le processeur connaît beaucoup d’instructions différentes. **CISC** (Complex Instruction Set) - Exemple : x86 (ordinateurs de bureau)

- Approche 2 : le processeur connaît peu d’instructions différentes. **RISC** (Reduced Instruction Set) - Exemple : ARM (téléphones, tablettes)

Jeu d’instructions d’un processeur

- Approche 1 : le processeur connaît beaucoup d’instructions différentes.
  CISC (Complex Instruction Set) - Exemple : x86 (ordinateurs de bureau)
  Haute consommation d’énergie.
  Instructions de longueur variable.

- Approche 2 : le processeur connaît peu d’instructions différentes.
  RISC (Reduced Instruction Set) - Exemple : ARM (téléphones, tablettes)
  Basse consommation d’énergie.
  Instructions de longueur fixe.

En quoi consiste une instruction ?

- Comment formuler une instruction ?
- Comment la traduire sous une forme compréhensible pour la machine ?
- Réponses dans le prochain chapitre.

Sondage

votamatic.unige.ch : SGGX

À votre avis, où se situe la majorité des transistors au sein des CPU modernes ?

- UAL
- UC
- Cache
- Registres