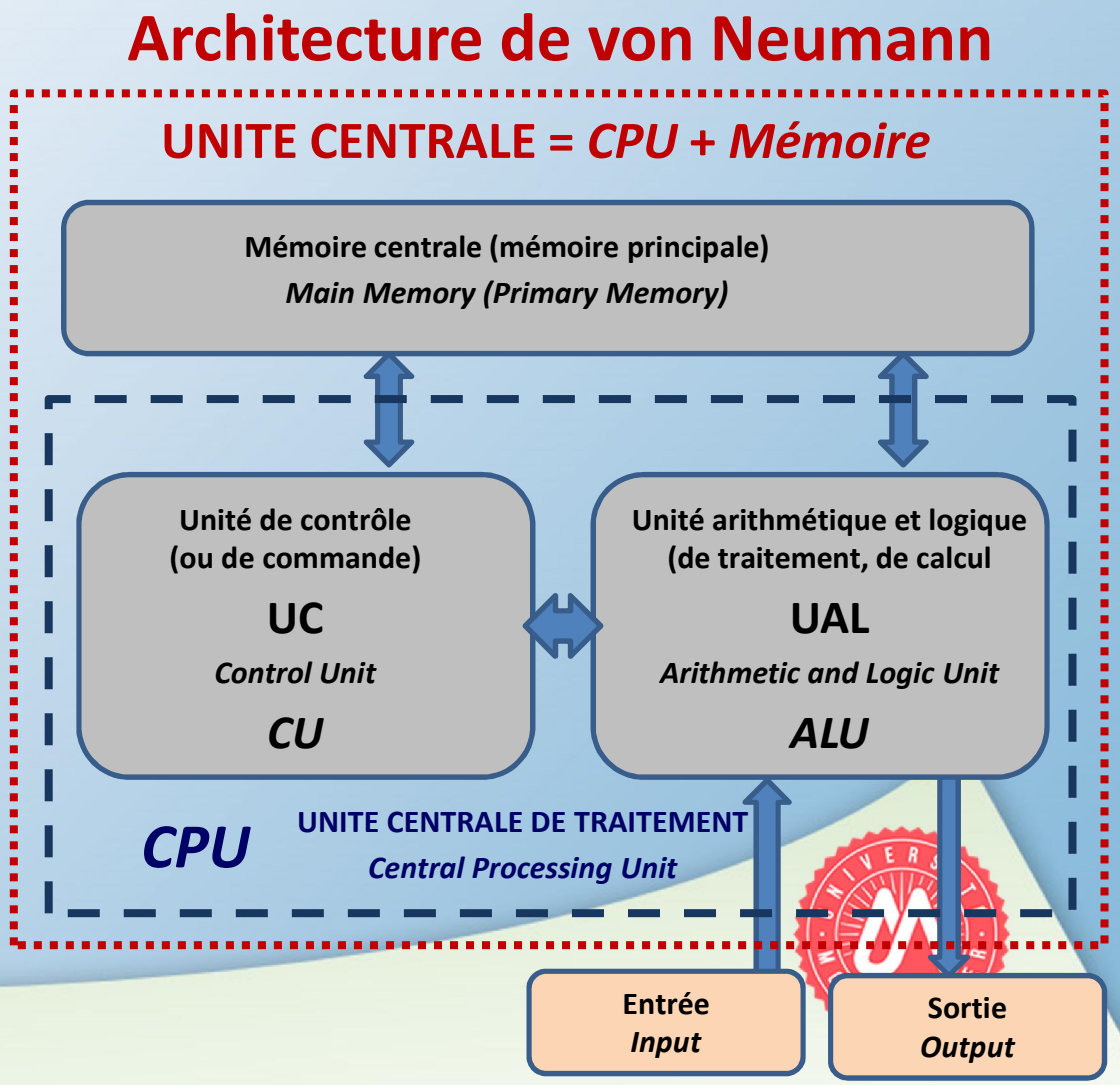
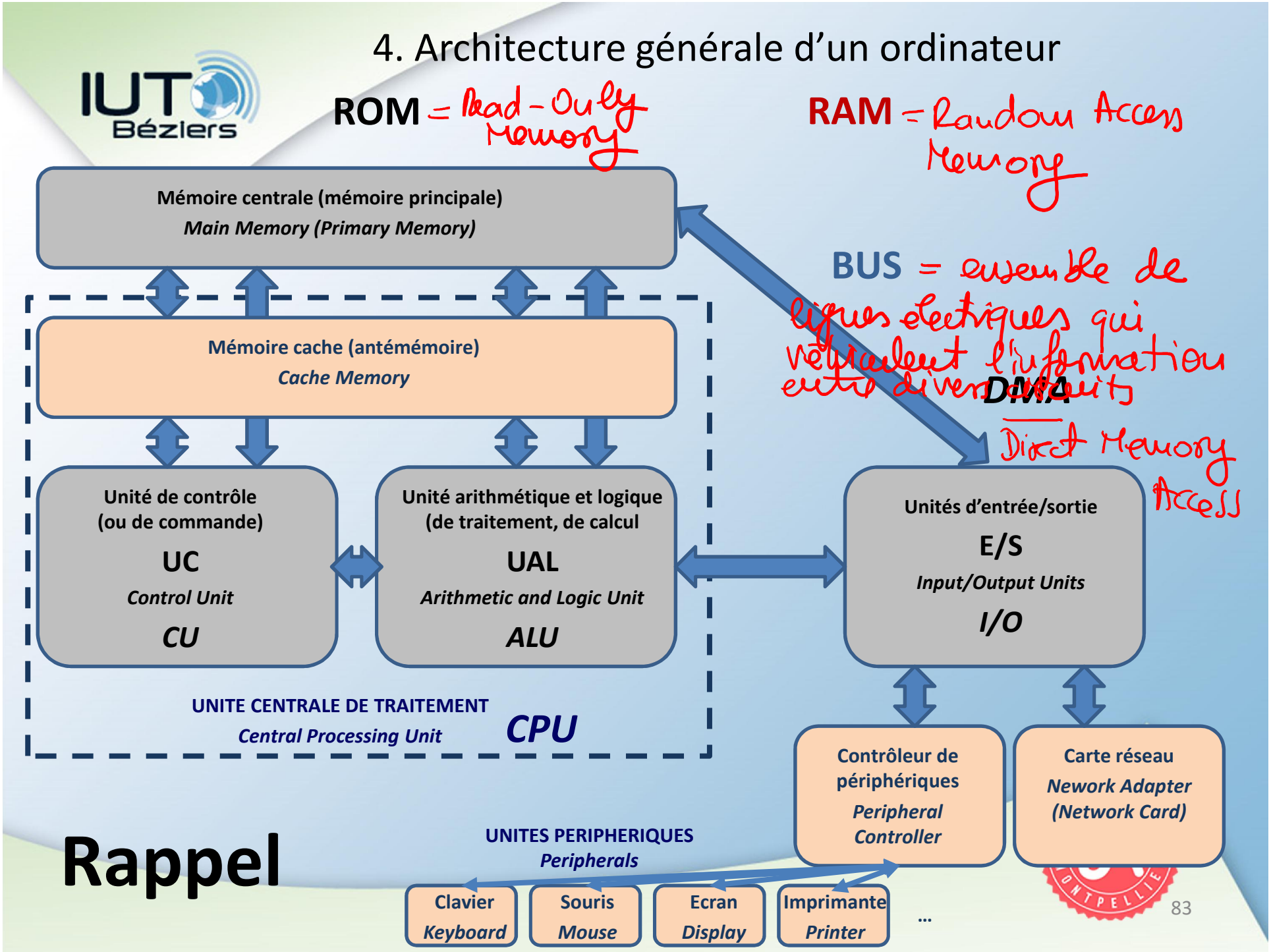
4. Architecture générale d'un ordinateur

Rappels

- Tous les ordinateurs modernes sont basés sur le modèle de **machine** universelle **de von Neumann**
- Ils contiennent **en plus** des composants pour améliorer la vitesse de circulation des données et l'interopérabilité des circuits (mémoire cache, « bus », contrôleurs de périphériques, accès direct à la mémoire…)



4. Architecture générale d'un ordinateur



Rappel

Rôle du cache (antémémoire)

- La **vitesse (fréquence) du CPU >>** la vitesse (fréquence) de la **RAM** (jusqu'à **x 10** selon le type de RAM! !)

Ex. RAM **DDR3** : f ~ **1 GHz**, µP **Core i7** : f ~ **3 GHz**

- Lorsque le CPU essaie d'écrire d'accéder à la RAM, il y a donc des « **temps morts** »
- Le **cache** a donc pour but de **stocker temporairement les instructions et/ou des données afin qu'elles puissent être retrouvées rapidement par le processeur**
- Le cache est donc une **mémoire** rapide :
  - **Plus rapide que la RAM**
  - **Moins rapide que le processeur**

Les niveaux de cache dans l'unité centrale

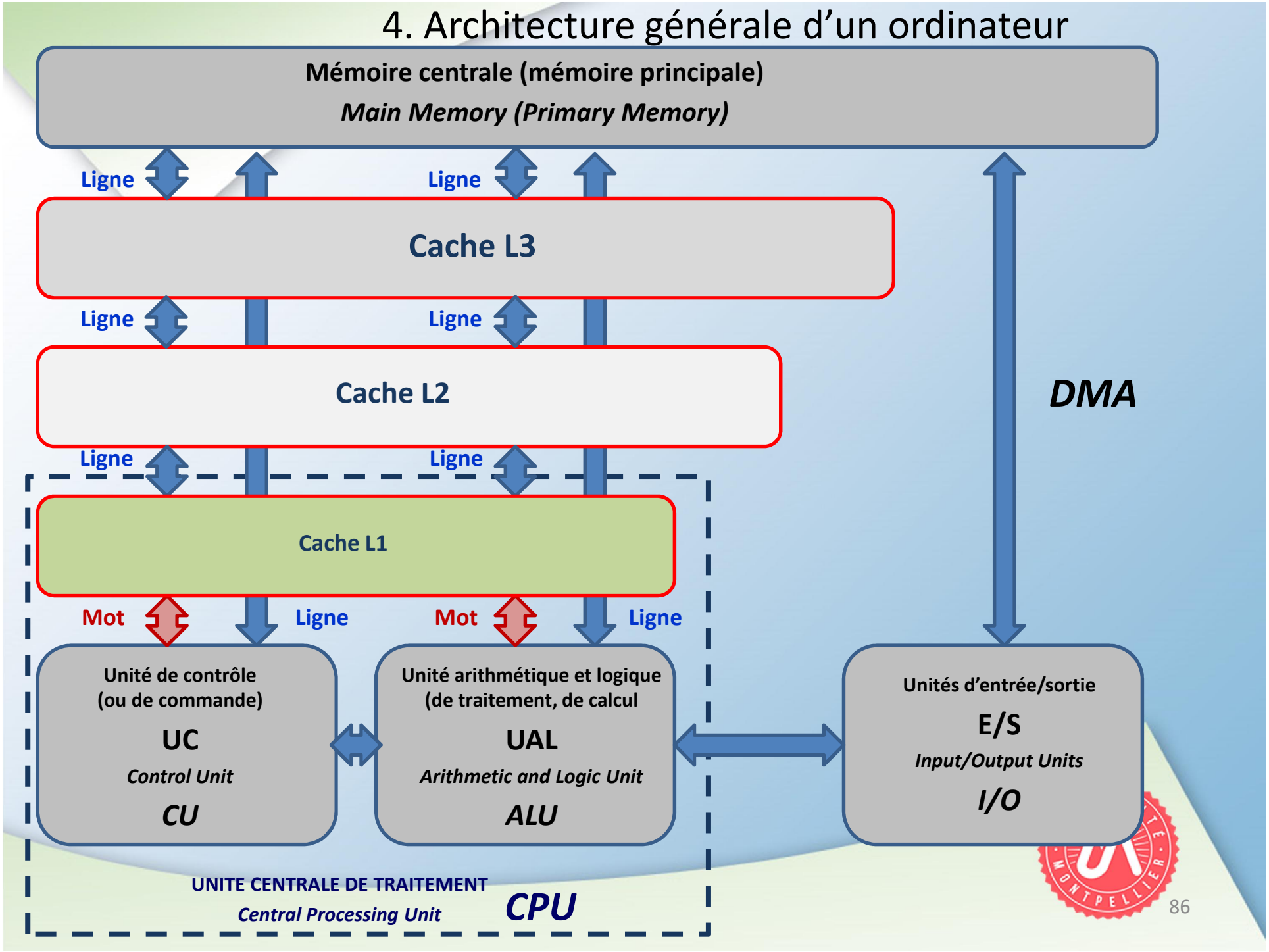
- Cache de niveau **L1** : le **plus rapide et plus petit** (cache de données pouvant être séparé du cache d'instructions)
- Cache de niveau **L2** : moins rapide et plus gros
- Cache de niveau **L3** : encore moins rapide et encore plus gros

- Le cache de niveau **L1** est situé **dans le CPU**
- Les caches de niveau **L2** et **L3** peuvent être situés **à l'intérieur ou à l'extérieur du CPU – sur les machines récentes, tous les caches sont dans le CPU**

Mot = le plus petit élément de données qui peut être transféré entre le **processeur** et la **mémoire cache**

Ligne = le plus petit élément de données qui peut être transféré entre la mémoire **cache** et la **mémoire de niveau supérieur**

4. Architecture générale d'un ordinateur



Fonctionnement du cache (antémémoire)

1) L'élément demandeur (**CPU**) **demande une information**
2) Le **cache vérifie s'il possède** cette information :
   - s'il la possède, il la retransmet à l'élément demandeur – on parle alors de succès de cache (« **cache hit** »)
   - s'il ne la possède pas, il la demande à l'élément fournisseur (la RAM) – on parle alors de défaut de cache (« **cache miss** »)
3) L'élément fournisseur (RAM) **traite la demande et renvoie la réponse au cache**
4) Le cache la **stocke pour utilisation ultérieure** et la retransmet à l'élément demandeur au besoin

Les 2 principes de fonctionnement du cache

- Déduits suite à des études du comportement des programmes

1) **Localité spatiale**
   L'accès à une donnée située à une adresse X va probablement être suivi d'un accès à une zone très proche de X :
   cas des **instructions exécutées en séquence**

2) **Localité temporelle**
   L'accès à une zone mémoire à un instant donné a de fortes chances de se reproduire dans la suite immédiate du programme : cas des **boucles de quelques instructions** seulement

La mémoire centrale

- **A pour rôle de contenir :**
  - **les instructions (les programmes)**
  - **les données**
- Les instructions sont stockées sous forme de « code machine »
  (ex. code d'une instruction d'addition d'un µP 8086 sur 8 bits : 00010010)
- Les données sont stockées sous d'autres formes de codes
  (ex. codage nb. entier, virgule flottante, code ASCII…)
- La **mémoire** centrale est **divisée** physiquement en **cellules**
- **Chaque cellule** contient un **mot mémoire** (WORD)
  et possède une **adresse propre**

La mémoire centrale

- A chaque **mot mémoire** on associe donc :
  - Une **adresse** (position du mot en mémoire)
  - Un **contenu** (instruction ou donnée)
- La **longueur d 'un mot mémoire** a beaucoup varié dans le temps :
  1, 2, 4, 8, 12, 24, ***32***, 48, 60, ***64 bits***

  Valeurs actuelles
- **Capacité d'une mémoire** = fonction du :
  - Nombre de mots
  - Nombre de bits (octets) / mot
- Une **cellule mémoire** peut être **adressée** :
  - Pour **lire le mot** qu'elle contient (**READ** - « ***R*** »)
  - Pour **écrire** un nouveau mot (**WRITE** – « ***W*** »)
    en écrasant l'ancien

Les registres du processeur

- **Registres** = **cellules mémoire situées dans le processeur, très rapides**
- Il existe 2 types de **registres**:
  - D'**adresse** (contient l'adresse d'un mot) **RA**
  - De **données = registres « mot »** (contiennent un mot mémoire) **RM**
- **Tailles** des registres :
  - Registre mot **RM** : taille d'un **mot mémoire**
  - Registre d'adresses **RA** : en général, taille d'un **mot mémoire, ou moins**
- **Le registre d'adresses doit permettre d'adresser TOUTES les cellules de la MEMOIRE → la taille maximale de mémoire que le CPU peut gérer dépend UNIQUEMENT de la taille du registre d'adresses**

Les registres du processeur : mémoire adressable

Exemples :

1) Soit un µP à registre d'adresses de 8 bits. Combien de cellules mémoire peut-il adresser ? Quelle sera la taille maximale de la mémoire utilisable si la taille d'un mot mémoire est de 1 octet ? De 2 octets ?

2) On dispose d'un µP à registre d'adresses de 32 bits et de barrettes de mémoire avec des cellules de mots de 64 bits. Combien de Go au maximum peut adresser (utiliser) le µP ?

Multiples de l'octet

Traditionnellement, *les préfixes « kilo », « méga », « giga »*, etc. dans le monde informatique, ne représentaient pas une puissance d'un nombre en base 10 (10^3 = 1 000), mais *une puissance d'un nombre en base 2* (2^{10} = 1 024).
Cet usage reste largement en vigueur chez les professionnels comme le grand public, c'est donc celui employé dans ce cours.

Nom	*Symbole*	*Valeur (usage courant)*	*Valeur si le multiple était exprimé selon les préfixes du SI d'unités (non utilisée)*
kilooctet	ko	2^{10}	10^3
mégaoctet	Mo	2^{20}	10^6
gigaoctet	Go	2^{30}	10^9
téraoctet	To	2^{40}	10^{12}
pétaoctet	Po	2^{50}	10^{15}
exaoctet	Eo	2^{60}	10^{18}
zettaoctet	Zo	2^{70}	10^{21}
yottaoctet	Yo	2^{80}	10^{24}

La mémoire centrale

- **Lecture** d'un mot (**READ**) :
  - L'UC du CPU inscrit l'adresse du mot à lire dans le **RA**
  - L'UC demande une opération à l'UAL
  - La copie du mot situé à l'adresse inscrite dans le RA est transféré dans un **RM**

- **Ecriture** d'un mot (**WRITE**) :
  - **RM** ← mot à écrire dans la mémoire
  - **RA** ← adresse où le mot sera écrit
  - **mémoire [RA] ← RM**

Le contenu précédent de mémoire [RA] est effacé !!!

La mémoire centrale

- **Temps d'accès = temps nécessaire à la lecture ou à l'écriture d'un mot mémoire (1 ns à qq. centaines de ns)**
- **Si le temps d'accès est identique pour tous les mots de la mémoire centrale → mémoire à accès aléatoire = Random Access Memory (RAM)**

- **Question : comment sait-on si un mot mémoire correspond à une instruction ou à une donnée ?**

L'unité centrale de traitement (CPU)

- **Composée de :**
  - **l'Unité de Contrôle (UC)**
  - **l'Unité Arithmétique et Logique (UAL)**
- L'**UC** gère l'exécution du **programme**
- L'**UAL** réalise les **opérations** arithmétiques et logiques (**+ – × ÷ AND OR XOR**…)
- Toutes les opérations s'exécutent dans les **registres du CPU**

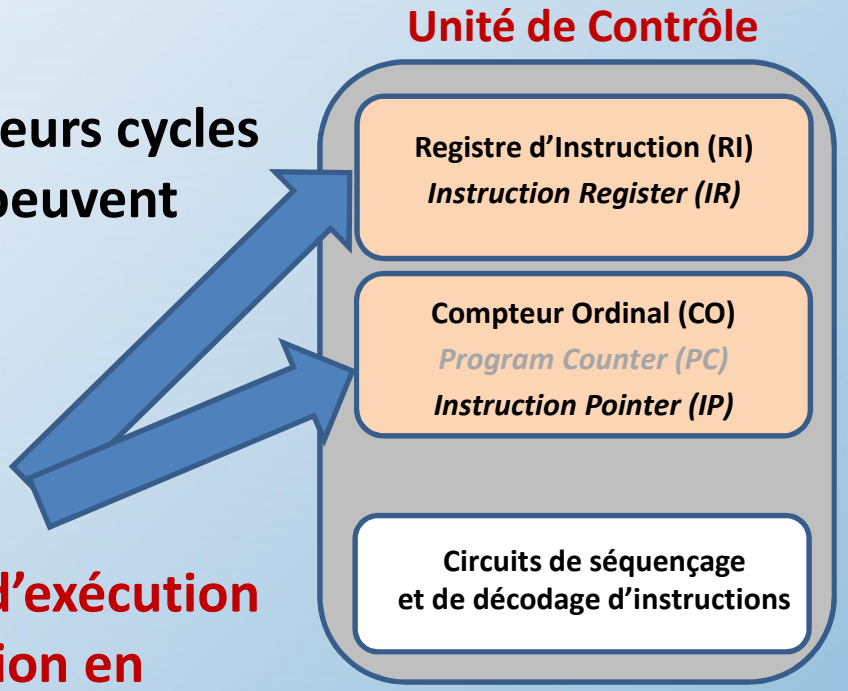
Le CPU

L'Unité de Contrôle (UC) : gère l'exécution des programmes

1) **Extrait l'instruction de la mémoire**
2) **Analyse l'instruction**
3) **Recherche dans la mémoire les données concernées par l'instruction**
4) **Déclenche l'opération adéquate soit sur l'UAL, soit sur les E/S**
5) **Range le résultat dans la mémoire**

- **Le « top départ » de chaque opération est donné par l'horloge**
  = circuit qui génère des signaux de séquence régulière de « 0 » et « 1 »

L'UC



- **Une instruction peut nécessiter plusieurs cycles d'horloge, ou plusieurs instructions peuvent être exécutées sur un seul cycle (ex. 4 FLOPS/cycle)**

- **L'UC contient 2 registres importants**

- le RI contient l'instruction en cours d'exécution

- le CO contient l'adresse de l'instruction en cours (ou l'adresse de l'instruction suivante, selon l'archi du µP)

- **Une fois l'instruction chargée dans le RI, le CO est incrémenté pour désigner (« pointer ») l'instruction suivante**

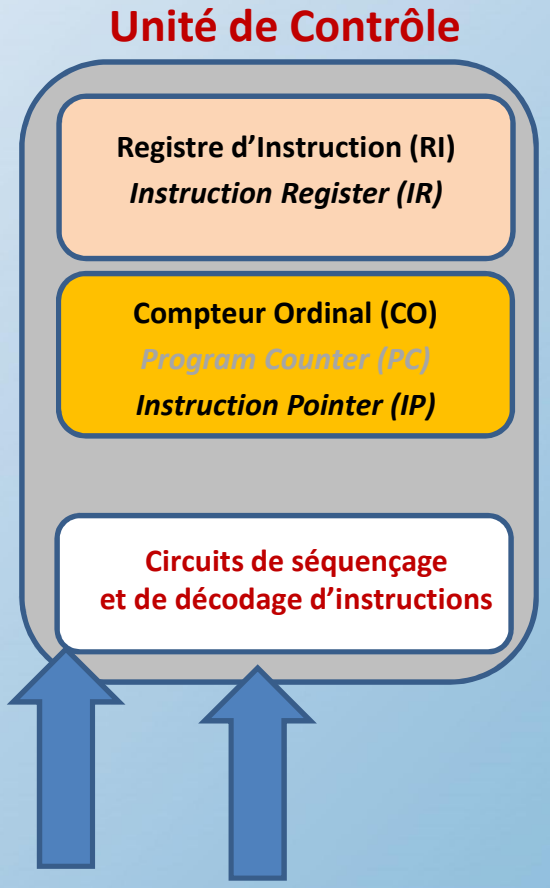
[CO] ← [CO] + 1

L'UC

- Les instructions de saut et d'appel au sous-programmes permettent de modifier le contenu du CO pour le faire pointer à une adresse précise

- Les programmeurs ne peuvent pas avoir accès aux registres de l'UC

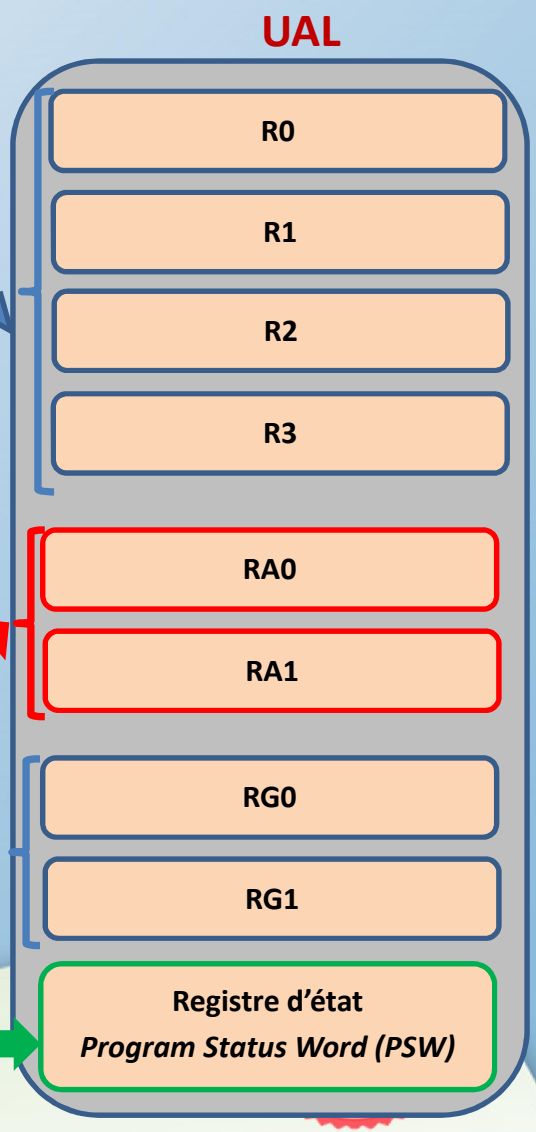
- L'UC contient aussi
  - un dispositif de décodage des instructions
  - un « séquenceur » = circuit qui active les différents composants électroniques nécessaires à l'exécution de l'instruction en cours



L'Unité Arithmétique et Logique (UAL)

- **Intègre des registres qui doivent contenir les données nécessaires aux opérations arithmétiques et logiques (+ - × ÷ ET OU …), à savoir :**
  - les opérands = valeurs sur lesquelles s'effectuent les opérations
  - les résultats
  - les valeurs intermédiaires

- **Groupes de registres de l'UAL**
  - registres arithmétiques (pour les opérations)
  - registres d'adresse « de base » et « d'index »
  permettant de calculer des adresses par rapport d'une valeur de base ou d'un index
  - registres généraux (banalisés), qu'on peut utiliser pour des opérations intermédiaires
  - le registre d'état



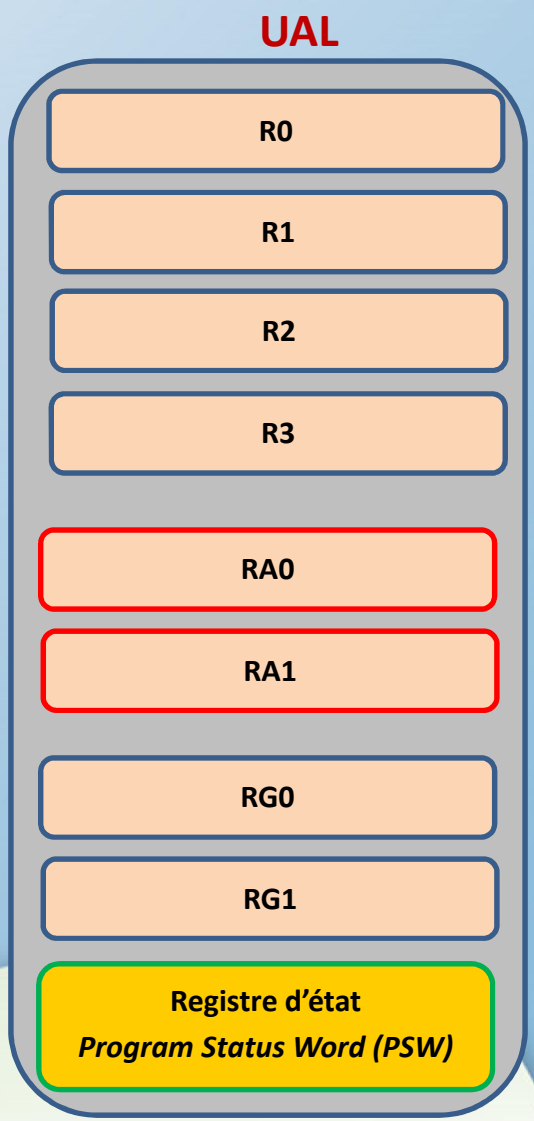
L'UAL

- **Le registre d'état (« Program Status Word »)** contient des bits = drapeaux (« flags ») ou indicateurs d'état, qui informent sur la dernière opération effectuée par le processeur, tels que :

  - l'existence d'une retenue lors de la dernière opération arithmétique (CARRY FLAG, CF=1 si retenue)
  - dépassement de capacité (OVERFLOW FLAG, OF=1 si dépassement
  - nullité du résultat (ZERO FLAG, ZF=1 si résultat nul)
  - parité (PARITY FLAG, PF=1 si résultat pair)

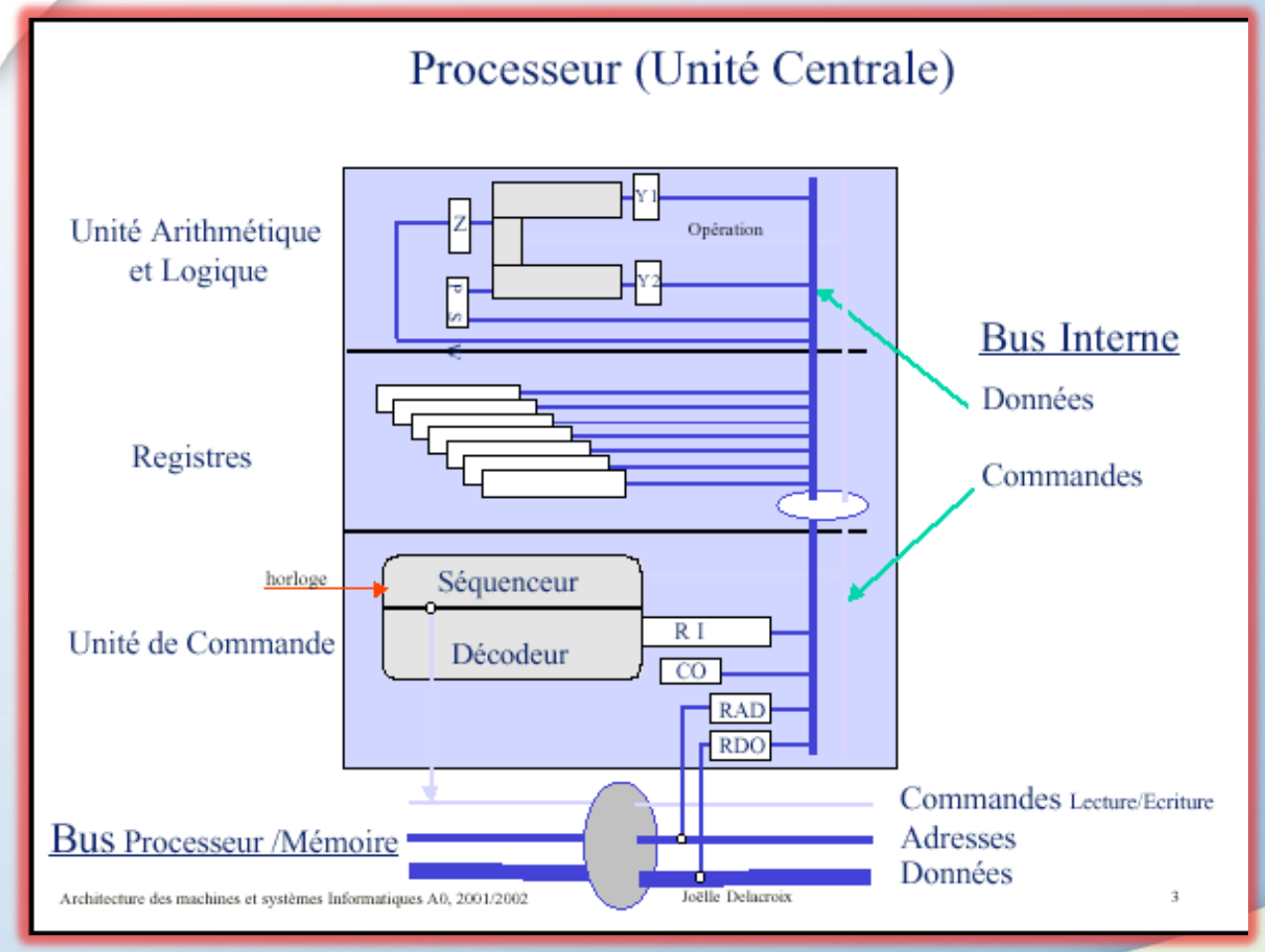
  But :

- **Le PSW est interprété bit par bit pour décider lors de sauts conditionnels**



4. Architecture générale d'un ordinateur

Le CPU



Les unités d'Entrée/Sortie

(« E/S », « I/O » ou « unités d'échange »)

- Servent d'interface entre le CPU et les périphériques
- Les opérations associées sont fonction du périphérique
- Fonctionnement similaire à la mémoire (registre d'adresses du périphérique, registre d'échange de données…)
- Le CPU travaille ++ vite que les périphériques → pour ne pas faire attendre le CPU, on utilise des processeurs spécialisés pour gérer certains périphériques (ex. carte graphique)
- Certaines périphériques peuvent avoir besoin d'un accès direct à la mémoire → DMA gérée par des processeurs spécialisés

Les unités périphériques

- **Deux types :**
  - **Unités permettant l'échange de données avec l'extérieur (écran, clavier, souris, imprimantes, cartes réseau…)**
  - **Mémoires auxiliaires = mémoires non volatiles, permettant de stocker les données de façon permanente (disques durs, clés USB, CD, DVD…) = MEMOIRES DE MASSE**

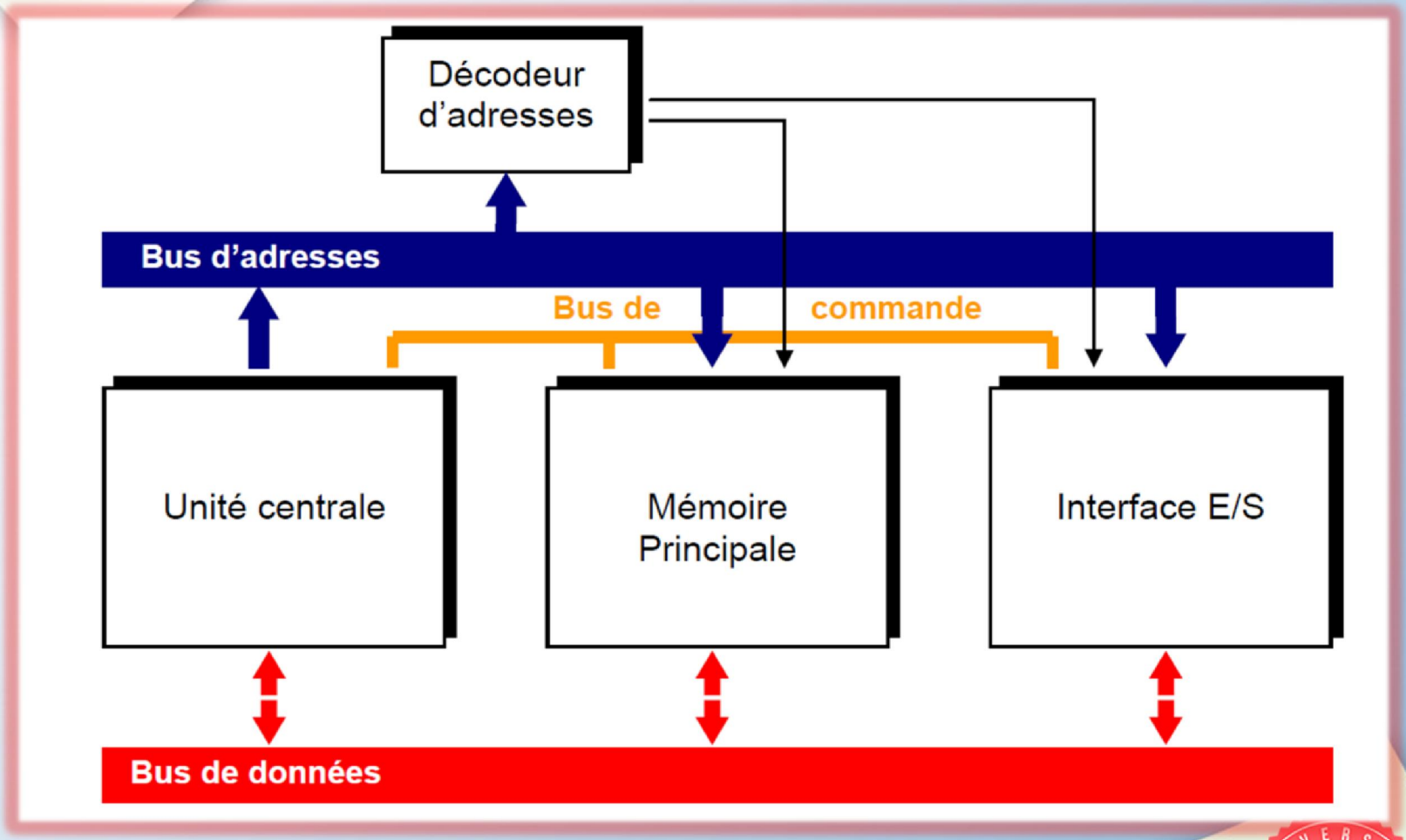
- **Pour chaque catégorie de périphériques, il existe un contrôleur de périphériques qui gère ces unités et l'interface avec les unités d'E/S**

Les bus

- **Ensemble de fils électriques sur lesquels transitent les signaux**
- **Relient les différents composants entre eux**
- **3 types de bus :**
  - **de données (échange de données)**
  - **d'adresses (transmission de l'adresse de la donnée à échanger ou de l'instruction à exécuter)**
  - **de contrôle (spécifie le type d'opération (R/W) et donne le « top départ » de l'opération)**
- **Largeur d'un bus = nb. de fils constituant le chemin = nb. de bits (octets) pouvant être véhiculés en même temps**
- **Les bus peuvent être vus comme des unités d'E/S :**
  la CPU met des données sur un bus, ou elle lit les données présentes sur un bus

4. Architecture générale d'un ordinateur

Les bus



Le jeu d'instructions

- **Ensemble des instructions exécutables par le processeur**
- **Les instructions portent sur les données et les adresses = opérandes**
- **Il existe différents formats d'instructions, en fonction du nombre de parties réservées aux opérandes, ex. :**

| **opération** | opérande | (*format 1 adresse*)
| **opération** | opérande 1 | opérande 2 | (*format 2 adresses*)

Exemples d'instructions génériques format 1 adresse :

lirePériph nomPériph
écrirePériph nomPériph
chargerAcc adresseMémoire
mémoAcc adresseMémoire
testZéro adresseMémoire

additionner adresseMémoire
soustraire adresseMémoire
multiplier adresseMémoire
diviser adresseMémoire
stop

Structure des instructions niveau machine

- **Toutes les opérations s'exécutent dans les registres du µP**
- **Les µP sont capables de faire un certain nombre d'opérations simples, par ex. :**
  - + – × ÷ AND OR XOR
  - Tester le signe d'une valeur numérique
  - Copier le contenu d'un registre dans un autre registre
  - Stocker en mémoire le résultat d'une opération
- **Une instruction machine doit fournir au CPU toutes les informations pour déclencher une opération élementaire**
- **Elle doit contenir le code opération (= mnémonique) et une ou plusieurs adresses**
- **Donc, une instruction = code opération + 0 à 4 champs d'adresse**
- **1 seul champ d'adresse → instruction à 1 adresse**

Structure des instructions niveau machine

| **opération** | opérande | → ***format 1 adresse***

| **opération** | opérande 1 | opérande 2 | → *format 2 adresses*

- **Aujourd'hui, on prefère les formats à 1 adresse**
  (pour éviter l'augmentatation des tailles des registres avec l'augmentation de la capacité de la mémoire...)
- **Pour réduire de 2 à 1 les adresses des opérandes, on utilise un registre spécial = registre accumulateur :**
  - **Le 2ème opérande se trouve déjà dans le registre accumulateur, chargé par l'instruction précédente**
  - **L'opération se fait implicitement dans le registre accumulateur à l'aide d'une instruction qui porte seulement sur le 1er opérande**
  - **Le résultat est stocké dans ce même accumulateur**
- On peut aussi envisager des **instructions à zéro adresse** → utilisation d'une **pile** (liste LIFO) =
  zone de mémoire ou les opérandes se trouvent dans les deux positions supérieures et le résultat au sommet

Le jeu d'instructions

- **Exemple**

Tâches à effectuer	Instructions génériques	Exécution du programme par la CPU (4 phases/cycle CPU)
• **Acquisition au clavier**	**lirePériph** Clavier	H_0 : $RA \leftarrow PC$ H_1 : $RI \leftarrow RM$, $PC \leftarrow PC + 1$ H_2 : $RSP \leftarrow RI_p$ H_3 : $ACC \leftarrow RE$
• **Addition de la valeur lue avec une valeur en mémoire**	**additionner** 163	H_0 : $RA \leftarrow PC$ H_1 : $RI \leftarrow RM$, $PC \leftarrow PC + 1$ H_2 : $RA \leftarrow RI_p$ H_3 : $ACC \leftarrow ACC + RM$
• **Affichage du résultat à l'écran**	**ecrirePériph** Ecran	H_0 : $RA \leftarrow PC$ H_1 : $RI \leftarrow RM$, $PC \leftarrow PC + 1$ H_2 : $RSP \leftarrow RI_p$ H_3 : $RE \leftarrow ACC$

RA=registre d'adresses, PC=Program Counter (Compteur Ordinal), RI=registre d'instructions, RM=registre mot
RSP=registre de sélection du périphérique (E/S), RE=registre d'échange de données (E/S), ACC=registre accumulateur

Le jeu d'instructions

- **Représentation des instructions en mémoire : séquence de bits découpée en champs**

  Exemple
  0010 0010 0000 1100

- **Représentation symbolique**
  (ou en langage bas niveau = assembleur) :
  - les codes des instructions sont représenté par des abréviations = mnémoniques (*LOAD*, *STORE*, *ADD*…)
  - les registres sont représentés par leur nom (*AX*, *BX*, *DX*…)
  - les adresses mémoire sont représentées par leur valeur ou un nom de variable
  - des symboles particuliers existent pour les contenus, les modes d'adressage, … (@, [ ], ()…)

Le jeu d'instructions

- **Exemple**

Adresse mémoire	Contenu
0100	0010 0010 0000 1100
0101	0001 0010 0000 1101
0110	0001 0010 0000 1110
0111	0011 0010 0000 1111
1100	0000 0000 0000 0010
1101	0000 0000 0000 0011
1110	0000 0000 0000 0100
1111	0000 0000 0000 0000

Adresse mémoire	Contenu
0100	*LOAD* (1100)
0101	*ADD* (1101)
0110	*ADD* (1110)
0111	*STORE* (1111)
1100	0002
1101	0003
1110	0004
1111	0000

Le jeu d'instructions

- **En langage bas niveau (assembleur), *X = X + Y* se traduit par :**

Format 2 adresses

LOAD X,R1
ADD R1, Y
STORE R1, X
CMP R1,X

Format 1 adresse
(implicitement, les opérations se font dans R1)

LOAD X
ADD Y
STORE X
CMP X

4. Architecture générale d'un ordinateur
Exemples d'instructions du 80x86

ADD	**Add**	Ajoute deux entiers	Le résultat remplace le premier opérande
AND	Logical **AND**	Effectue un ET logique des opérandes	Le résultat remplace le premier opérande
CALL	**Call** Procedure	Appelle une procédure	
CBW	**C**onvert **B**yte to **W**ord	Convertit un octet en mot	Le registre AL est étendu à AX
CLC	**Cl**ear **C**arry Flag	Met le drapeau de retenue à zéro	
CLD	**Cl**ear **D**irection Flag	Met le drapeau de direction à zéro	
CLI	**Cl**ear **I**nterrupt Flag	Met le drapeau d'interruption à zéro	
CMC	**Com**plement **C**arry Flag	Inverse le drapeau de retenue	
CMP	**Comp**are	Compare deux entiers (de façon arithmétique)	Positionne les drapeaux en fonction de la différence entre les opérandes
CMPSzz	**Comp**are **S**trings	Compare un octet/mot de deux chaînes	Mnémoniques : CMPS, CMPSB, CMPSW
CWD	**C**onvert **W**ord to **D**oubleword	Convertit un mot en double mot	Le registre AX est étendu à DX:AX
DAA	**D**ecimal **A**djust AL after **A**ddition	Ajuste le registre AL après addition (mode décimal)	Utilisé avec le codage BCD compacté
DAS	**D**ecimal **A**djust AL after **S**ubtraction	Ajuste le registre AL après soustraction (mode décimal)	Utilisé avec le codage BCD compacté
DEC	**Dec**rement by 1	Décrémente un entier	Ôte un de l'opérande
DIV	Unsigned **Div**ide	Divise par un entier non signé	Le dividende est le registre AX/DX:AX, le quotient est écrit dans AL/AX et le reste dans AH/DX
ESC	**Esc**ape		Utilisé avec l'unité de calcul en virgule flottante
HLT	**Halt**	Entre en état d'arrêt jusqu'à réception d'une interruption	Permet de réduire la consommation de puissance du processeur. Alias : HALT

Le jeu d'instructions

- **Une instruction en langage de haut niveau (ex. C) équivaut à plusieurs instructions en langage de bas niveau (assembleur)**
  → rôle important du compilateur !

- **Le jeu d'instructions du processeur doit être suffisamment expressif pour permettre de coder toute instruction d'un langage de haut niveau**

Compilateur

- **Compilateur = logiciel** qui transforme un programme (« code ») source écrit dans un langage de programmation (le langage source) en un autre langage informatique (le langage cible)

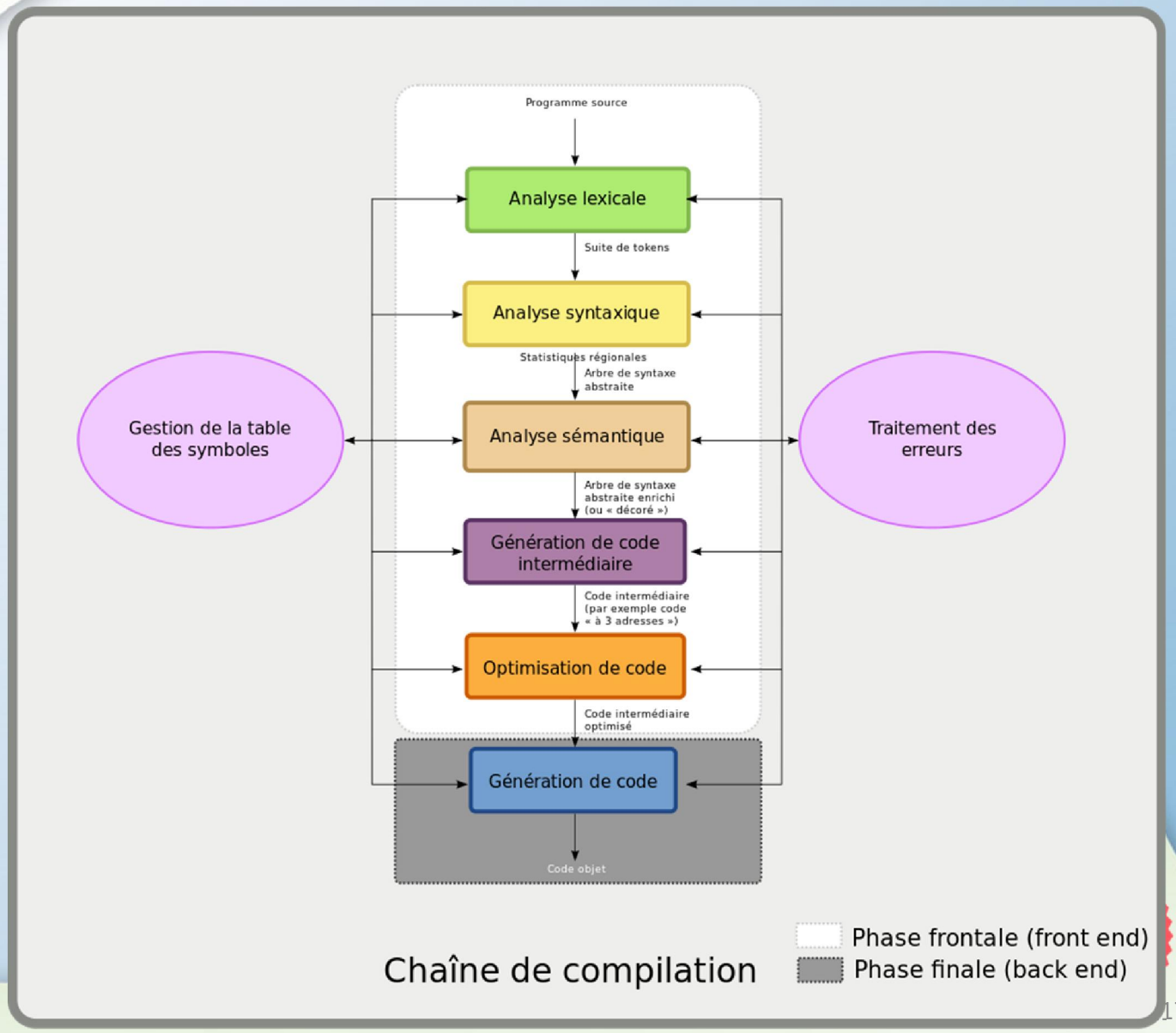
- Il s'agit en général de la traduction d'un programme écrit un **langage de haut niveau**, facilement compréhensible par l'humain (C, Java…), **vers** un langage de plus bas niveau = **assembleur ou langage machine**

- Dans le cas de langage semi-compilé (ou semi-interprété), le code source est traduit en un langage intermédiaire, sous forme binaire (code objet ou bytecode), avant d'être lui-même interprété ou compilé

- **Interpréteur = ?**

4. Architecture générale d'un ordinateur

Compilateur



Chaîne de compilation