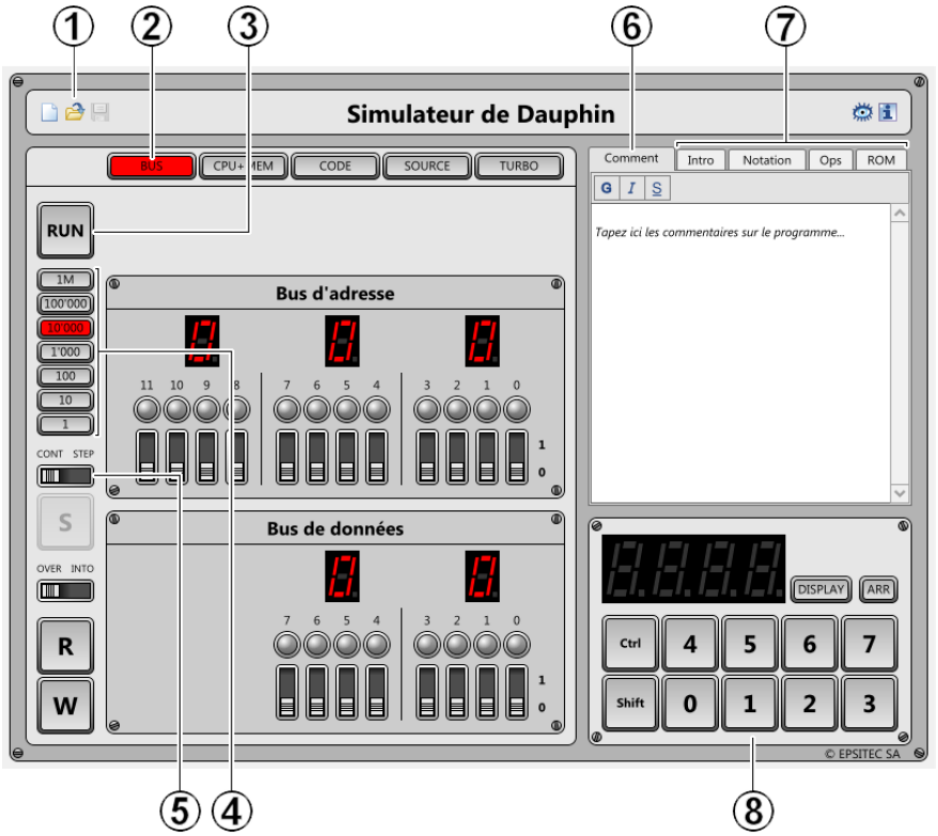
1 Dans les entrailles d'un ordinateur avec le simulateur de Dauphin

1.1 Présentation du logiciel

Le simulateur de Dauphin simule le fonctionnement du Dauphin, mini-ordinateur crée en 1977 par le professeur Jean-Daniel Nicoud, une sorte d'ancêtre du Raspberry Pi.
Ce simulateur réalisé par la société Epsitec, est librement téléchargeable avec sa documentation depuis cette page : `http://www.epsitec.com/dauphin/`.
La plupart des commentaires et exercices ci-après sont tirées de cette documentation.
Le logiciel n'étant pas installé sur le réseau et ne fonctionnant pas sous Linux, le plus simple est de l'installer chez vous sur une clef USB. Vous pourrez ensuite lancer l'exécutable `Dolphin.exe` depuis la clef et obtenir une fenêtre avec l'interface graphique suivante :



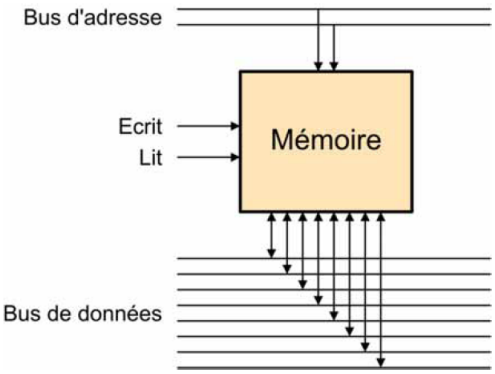
1. Icônes pour effacer, ouvrir ou enregistrer un programme.
2. Choix des panneaux affichés en dessous. [BUS] affiche les bus d'adresse et de données. [CPU+MEM] affiche le microprocesseur et la mémoire. [CODE] affiche le code avec les adresses mémoire et [SOURCE] affiche la traduction du code en langage d'assemblage.
3. Bouton principal [RUN/STOP] pour démarrer ou arrêter le processeur.
4. Choix de la vitesse du processeur, en nombre d'instructions par secondes.
5. Commutateur [CONT STEP] pour choisir le mode continu ou « pas à pas » (step by step).
6. Commentaires.
7. Résumé des instructions du microprocesseur.
8. Clavier et affichage du Dauphin, activés par des accès bus et certaines instructions.

1.2 La mémoire

La **mémoire physique** (ou mémoire vive) d'un ordinateur stocke les données et les programmes. On parle aussi de RAM pour Random Access Memory puisque chaque cellule peut être adressée pour recevoir des données ou des instructions. Elle supporte deux opérations élémentaires, **écriture** et **lecture**.

Par exemple une mémoire de 4 octets (soit 32 bits) peut être vue comme une commode ayant 4 tiroirs (ou cellules mémoires). Les tiroirs sont numérotés de 0 à 3, et chaque tiroir contient 8 bits. Dans la figure ci-dessous, le tiroir numéro 1 (donc le deuxième depuis le haut)contient la valeur binaire 00100110, c'est-à-dire H'26 en hexadécimal.

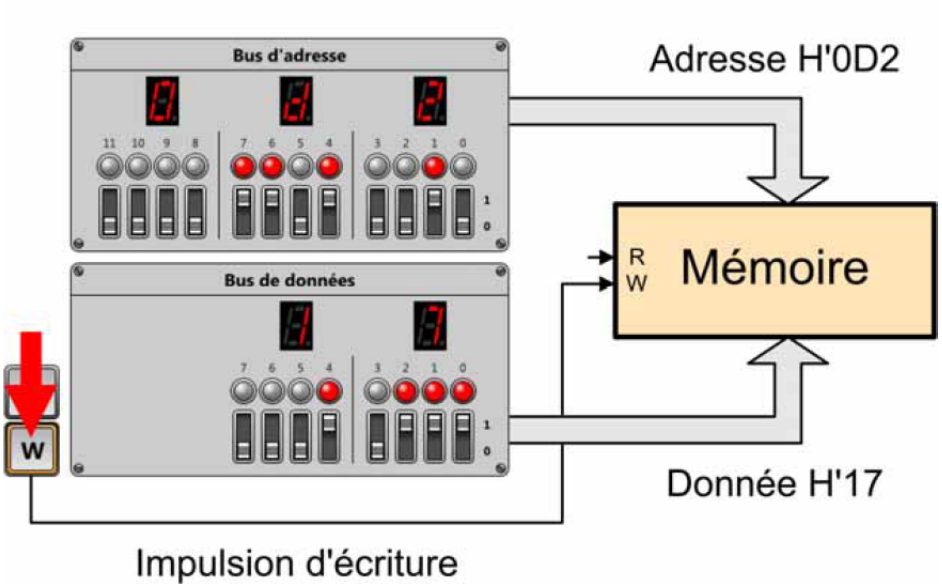
Pour écrire un octet, il faut donner deux informations : la donnée à écrire et l'adresse mémoire (ou numéro du tiroir). Ainsi une mémoire de 4×8 bits est reliée à l'unité centrale de l'ordinateur (le microprocesseur) par 8 connections (8 bits de données par cellule mémoire) et 2 connections pour les adresses ($2^2 = 4$ cellules mémoire ou tiroirs).



Un ensemble de connexions est appelé **bus**. La mémoire est donc accessible à l'unité centrale (microprocesseur) par un **bus d'adresse** et un **bus de données**.
Deux autres connexions permettent de sélectionner mode d'accès à la mémoire (en lecture ou en écriture).
La mémoire du Dauphin comporte ainsi 2 048 cellules mémoires (soit 16 384 bits), sélectionnées grâce à un bus d'adresse de 12 bits.

Exercice 1 *Ecrire ou Lire dans la mémoire du Dauphin*

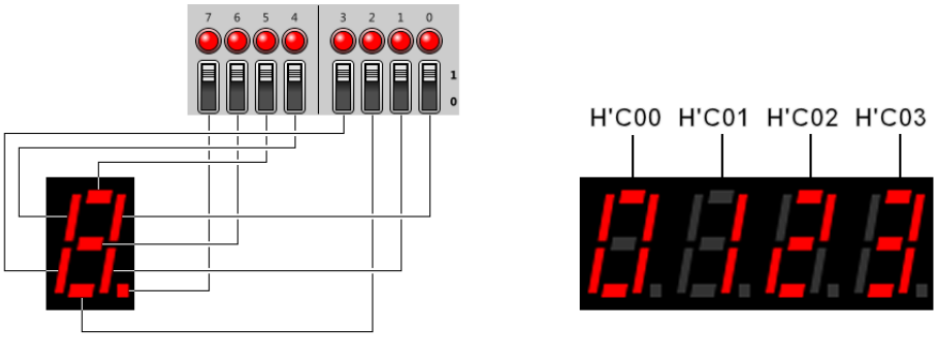
1. Pour écrire la valeur H'17 à l'adresse H'0D2 :
   - **a.** Sélectionner l'adresse H'0D2 (0000 1101 0010) avec les interrupteurs du bus d'adresse.
   - **b.** Sélectionner la donnée H'17 (0001 0111) avec les interrupteurs du bus de données.
   - **c.** Presser sur le bouton poussoir [W] (write = écrire) en bas à gauche. Pendant que le bouton est maintenu pressé, les petites lampes et les deux écrans du bus de données affiche les valeurs qui y sont déposées et l'impulsion d'écriture est transmise à la mémoire.



2. Pour lire la valeur H'17 à l'adresse H'0D2 :
   a. Sélectionnez à nouveau l'adresse H'0D2 (0000 1101 0010).
   b. Pressez sur le bouton poussoir [R] (read = lire).

1.3 Les périphériques

Le dialogue entre l'utilisateur et l'ordinateur s'effectue à travers des **périphériques** d'entrée (souris,clavier,scanner) ou de sortie (écran,imprimante).
Le Dauphin dispose d'un périphérique d'entrée (clavier) et de deux périphériques de sortie (écran de 4 afficheurs digitaux et écran bitmap).
Les périphériques sont accessibles par le biais ce cellules mémoires spécifiques. Par exemple , les quatre afficheurs de l'écran digital sont accessibles aux adresses H'C00, H'C01, H'C02 et H'C03. Chaque bit du bus de données correspond à un segment de l'afficheur.



Exercice 2 *Affichage sur l'écran digital*

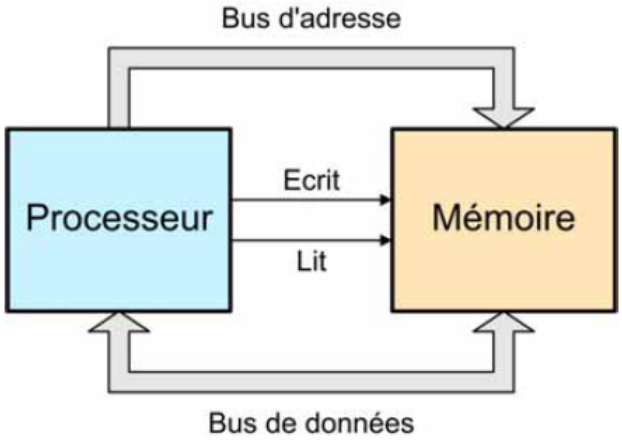
1. Sélectionner l'adresse H'C00 (1100 0000 0000) avec les interrupteurs du bus d'adresse.
2. Sélectionner la donnée H'07 (0000 0111) avec les interrupteurs du bus de données.
3. Pressez sur le bouton poussoir [W] (write = écrire) en bas à gauche. Une sorte de « J » apparaît sur l'afficheur de gauche.

1.4 Le microprocesseur

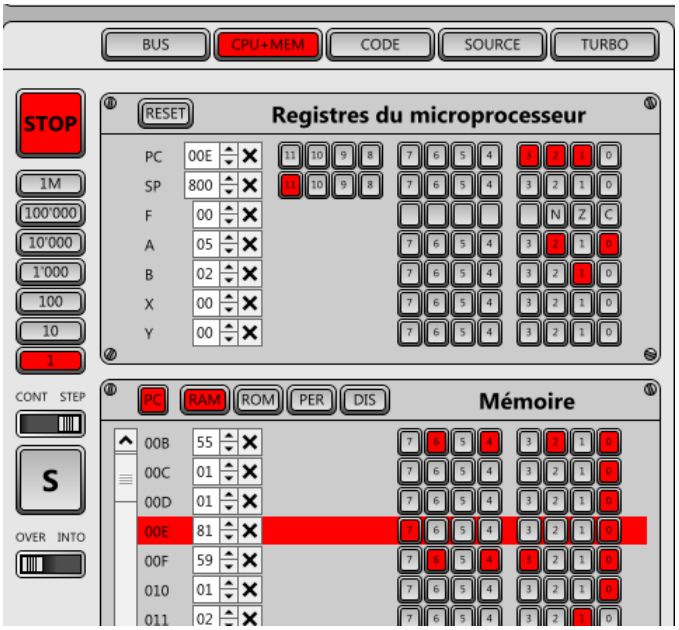
Le microprocesseur est l'unité centrale de l'ordinateur.
Pour simplifier on peut considérer que le microprocesseur exécute une seule instruction à la fois (mais il existe désormais des microprocesseurs à plusieurs coeurs qui travaillent en parallèle).
Dans l'ordre :

- il lit la prochaine instruction à réaliser dans la mémoire ;
- il décode cette instruction (**l'Unité de commande** qui décompose une instruction en une suite de micro-instructions préprogrammées dans le « dur ») ;
- il l'exécute (opérations arithmétiques et logiques réalisés par **l'Unité Arithmétique et Logique**)

Les instructions et les données sont stockées dans la mémoire (il faut bien les ranger et réserver suffisamment de place pour que le code compilé ne recouvre pas les données). Le microprocesseur possède un accès privilégié à la mémoire, il contrôle le bus de données et le bus d'adresses.



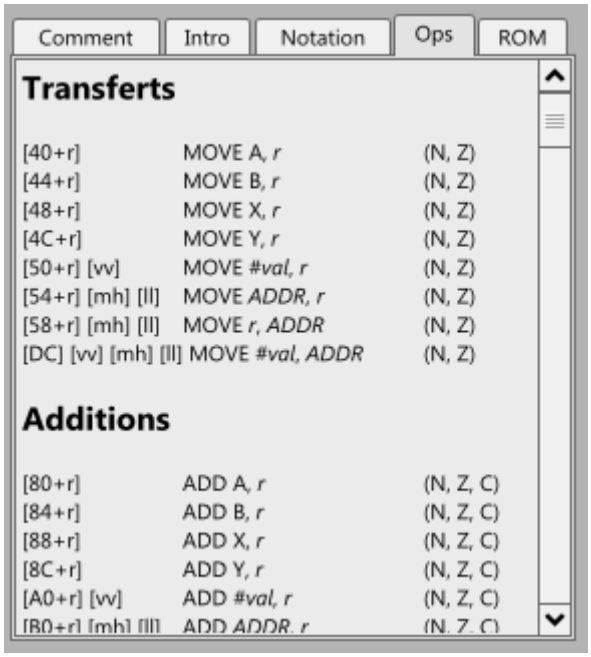
Le microprocesseur abrite un certain nombre de zones mémoires d'accès très rapides appelées **registres** .
Un registre important est le compteur ordinal (ou Programm Counter) qui contient l'adresse mémoire de la prochaine instruction à réaliser.
Le microprocesseur du Dauphin contient un registre PC de 12 bits (car le bus d'adresses est de largeur 12 bits) et des registres de données de 8 bits A et B par exemple (puisque les cellules mémoires contiennent exactement 8 bits et que le bus de données a pour largeur 8 bits).
Par exemple sur la figure, le registre PC pointe vers l'adresse mémoire H'00E (en hexadécimal soit $2^3+2^2+2^1$ en décimal). La cellule mémoire d'adresse H'00E contient la valeur H'81 en hexadécimal (soit $2^3 \times 16 + 1$ en décimal. On peut remarquer qu'un chiffre hexadécimal est codé par 4 bits (valeur entre 0 et $2^4 - 1 = 15$ soit F en hexa).



Si on clique sur l'icône nouveau programme et qu'on lance le micro-processeur du Dauphin avec Run, on peut remarquer qu'il parcourt toutes les adresses à la recherche d'une instruction. On peut choisir les modes Cont ou Step (pas à pas). Le contenu H'00 d'une cellule mémoire est décodé comme l'instruction NOP soit « ne rien faire et passer à l'instruction à suivante »). On peut remarquer que le microprocesseur agit très vite (il effectue une tâche élémentaire à la fois mais peut en réaliser des millions par seconde) et que le registre PC est incrémenté par défaut de 1 (on dispose d'instructions de type JUMP pour faire des sauts en avant ou en arrière et modifier le flux normal des instructions, dans un langage de programmation celà correspond à des boucles ou des instructions conditionnelles).

1.5 Jeu d'instructions élémentaires

Le microprocesseur du Dauphin peut exécuter un certain nombres d'instructions élémentaires dont la syntaxe est détaillée dans le guide de présentation et dans le menu activé en cliquant sur l'onglet ops à droite.



Voici quelques instructions que nous utiliserons dans des programmes élémentaires :

Syntaxe	Interprétation
MOVE H'601 , B	Déplace la valeur de l'adresse mémoire H'A01 (sur 12 bits en hexadécimal) vers le registre A
MOVE A , H'601	Déplace la valeur du registre A vers l'adresse mémoire H'A01 (sur 12 bits en hexadécimal)
MOVE #12 , A	Déplace la constante 12 (en décimal sur 8 bits) vers le registre A
MOVE #12 , H6A01	Déplace la constante 12 (en décimal sur 8 bits) vers l'adresse mémoire H'A01
ADD A , B	Ajoute le contenu du registre A au registre B
ADD #12 , A	Ajoute la valeur décimale 12 (sur 8 bits) au registre A
JUMP H'601	Saut inconditionnel vers l'instruction à l'adresse mémoire H'601
JUMP , ZS H'601	Saut vers l'instruction à l'adresse mémoire H'601 si la valeur de la dernière instruction était 0
SUB #01 , A	Retrancher la constante 1 au registre A
SUB A , B	Retrancher la valeur du registre A à celle du registre B

Comme le microprocesseur ne sait manipuler que des bits on peut distinguer deux niveaux dans l'écriture d'une suite d'instructions ou programme qu'on veut lui faire exécuter :

- Rédaction d'un code source dans un **langage d'assemblage** utilisant les instructions précédentes plus quelques directives supplémentaires ;
- Traduction de chaque instruction en séquence d'un ou plusieurs octets et implémentation du code dans la mémoire de l'ordinateur à partir d'une adresse mémoire précisée dans le code source par la directive `.LOC 0A1` si par exemple on veut implémenter le code source à partir de l'adresse mémoire H'0A1. Pour exécuter le programme il ne reste plus qu'à faire pointer le registre PC vers H'0A1.

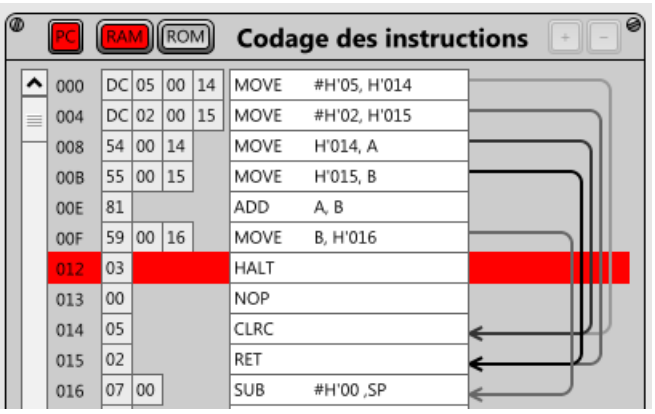
On pourrait coder manuellement les instructions dans la mémoire mais il est plus simple pour un humain de manipuler des mots que des bits, le programmeur écrira donc le code source et c'est un programme préenregistré, **l'assembleur**, qui traduira ce code source en instructions compréhensibles par le microprocesseur.

Exemple 1

Voici un premier exemple de code source en langage d'assemblage. Il s'agit d'un programme qui définit deux constantes, les stocke aux adresses mémoires H'100 et H'101, les charge dans les registres A et B, les additionne puis stocke le résultat à l'adresse mémoire H'102.
On peut remarquer la possibilité de définir des constantes et des étiquettes (Start et End). Les étiquettes seront utiles pour remplacer les adresses mémoires dans les JUMP.
Lorsqu'on effectue des transferts de valeurs entre registres et adresses mémoires les adresses sont notées en hexadécimal, les registres sont désignés par leur nom et les valeurs constantes sont préfixées du symboles # (on parle de valeurs immédiates).
On peut insérer des commentaires après un point virgule.

```
1     .LOC 0   ; localisation du code
2
3  Constantes :
4
5     NUMBER1 = H'05  ; premiere constante
6     NUMBER2 = H'02  ; seconde constante
7
8
9  Start:
10    MOVE #NUMBER1, H'014 ; placer NUMBER1 dans la cellule d'adresse H'014
11    MOVE #NUMBER2,H'015 ; placer NUMBER2 dans la cellule d'adresse H'015
12
13    MOVE H'014 , A   ; deplacer le contenu de la cellule memoire H'014 dans le registre A
14    MOVE H'015 , B    ; deplacer le contenu de la cellule memoire H'015dans le regsitre B
15    ADD A , B       ; ajouter le contenu du registre A au registre B
16    MOVE B , H'016   ; deplacer le contenu du registre B dans la cellule memoire H'016
17    HALT     ; fin du programme
```

Après exécution du programme, on peut obtenir la figure ci-dessous en cliquant sur l'onglet Code.
Le code du programme est implémenté des adresses H'000 à H'012. Les adresses mémoires H'014, H'015 et H'016 contiennent les données (traduites en instructions ce qui n'a pas de sens ici). Il faut noter que l'instruction ADD A , B est codé sur un octet (81 en hexadécimal) alors que l'instruction MOVE H'014 , A est codée sur trois octets (54 00 14 en hexadécimal).



Python est un langage interprété de haut niveau. On peut faire apparaître une traduction en langage d'assemblage d'un code intermédiaire entre le langage de haut niveau et le bytecode manipulé par l'interpréteur Python. Le module `dis` permet cette manipulation appelée désassemblage. Voici un exemple de désassemblage du code permettant l'échange des contenus de deux variables. Des structures de pile sont utilisées pour stocker les noms ou les constantes.

```
>>> import dis
>>> dis.dis('a=2 ; a=3 ; a, b = b, a')
  1           0 LOAD_CONST               0 (2)
              3 STORE_NAME               0 (a)
              6 LOAD_CONST               1 (3)
              9 STORE_NAME               0 (a)
             12 LOAD_NAME                1 (b)
             15 LOAD_NAME                0 (a)
             18 ROT_TWO
             19 STORE_NAME               0 (a)
             22 STORE_NAME               1 (b)
             25 LOAD_CONST               2 (None)
             28 RETURN_VALUE
```

Exercice 3

Compléter le programme ci-dessous entre les étiquettes START et END pour qu'il échange les valeurs NUMBER1 et NUMBER2 stockées aux adresses H'100 et H'101.

```
    .LOC 0 ; debut du code source

CONSTANTES:
    NUMBER1 = H'05 ; premiere valeur
    NUMBER2 = H'06 ; deuxieme valeur
    ADDR1 = H'100;
    ADDR2 = H'101;

INITILISATION :
    MOVE #NUMBER1, ADDR1
    MOVE #NUMBER2, ADDR2

START:
    ......

END:
    HALT
```

1.6 Programmes avec saut conditionnel ou inconditionnel

Dans un programme on a souvent besoin de changer le flux normal d'instructions (le registre PC pointe vers l'instruction suivante). Les instructions de type JUMP permettent d'effectuer un saut vers une instruction située en amont ou en aval dans le code source. Ce saut peut dépendre de la réalisation d'une condition (par exemple la valeur de la dernière instruction est 0) ou s'effectuer dans tous les cas (saut inconditionnel).
On peut ainsi réaliser des structures de choix alternatifs (`if ... else ...`) ou des boucles (`while ...`).

Exercice 4 *une boucle avec un seul saut*

1. Editer puis compiler le code source suivant.

```
1      .LOC    0
2
3  LOOP:
4      NOP
5      NOP
6      JUMP    LOOP
```

2. Exécuter le programme et décrire l'algorithme implémenté.
3. Traduire en Python un programme du même type et désassembler le code avec le module `dis`.

Exercice 5 *une boucle avec deux sauts*

1. Editer puis compiler le code source suivant.

```
1      .LOC 0    ; debut du code source
2
3  CONSTANTES:
4      NUMBER1 = H'05 ; constante égale au multiplicateur
5      NUMBER2 = H'0A ; constante égale au facteur maximal + 1
6      ADDR1 = H'A1
7
8  INITIALISATION:
9      MOVE #00, A
10     MOVE #NUMBER2, B
11     MOVE #NUMBER1, ADDR1
12
13 START:
14     SUB #01,B
15     JUMP,ZS END ; saut vers la fin si B contient 0
16     ADD ADDR1 , A
17     JUMP START
18
19 END:
20     HALT
```

2. Exécuter le programme et décrire l'algorithme implémenté (on s'intéressera au contenu du registre A lorsque le programme est terminé).

   Que se passe-t-il si on échange les lignes 14 et 16 ?

Exercice 6

Compléter puis compiler le code source suivant pour qu'à la fin du programme le registre B contienne $\sum_{k=0}^{n} k$ où n est un entier stocké dans le registre A.

```
    .LOC 0

Constantes:
    NUMBER1 = H'0B

Initialisation :
    MOVE #NUMBER1 , A
    MOVE #00 , B

Start:
    ....

End:
    HALT
```

Exercice 7

1. Editer puis compiler le code source suivant.

```
    .LOC  0

Constantes :
    EXPOSANT = H'005 ; exposant n+1 pour le calcul de 3^n
    ADDR1 = H'100 ; adresse de stockage de l'exposant

Initialisation:
    MOVE #EXPOSANT , A
    MOVE  #01 , B

Start :
    SUB #01 , A
    JUMP , ZS End
    MOVE A , ADDR1
    MOVE B , A
    ADD  A , B
    MOVE ADDR1 , A
    JUMP Start

End :
    HALT
```

2. Exécuter le programme et décrire l'algorithme implémenté (on s'intéressera au contenu du registre B lorsque le programme est terminé).

3. Ecrire un programme qui calcule 3^n où n est un entier naturel défini comme constante avant le corps du programme.

Exercice 8

1. Ecrire un programme qui calcule le terme de rang n (où n est une constante définie avant le corps du programme) de la suite (u_n) définie par

$$\begin{cases} u_0 = 1 \\ u_{n+1} = 2u_n + n \end{cases}$$

2. Ecrire un programme qui calcule le terme de rang n (où n est une constante définie avant le corps du programme) de la suite de Fibonacci (F_n) définie par

$$\begin{cases} F_0 = F_1 = 1 \\ F_{n+2} = F_{n+1} + F_n \end{cases}$$

2 Modèle de Von Neuman

2.1 Unité centrale, mémoire et bus

En 1946, **John Von Neuman**, mathématicien, a proposé un modèle de machine d'usage universel munie d'une mémoire unique contenant les programmes et les données et reliée à un circuit capable de lire un programme en mémoire, et de l'exécuter en utilisant les données en mémoire. Ce **modèle de Von Neuman** est à la base de l'architecture de tous les ordinateurs actuels, il est principalement composé de trois éléments :

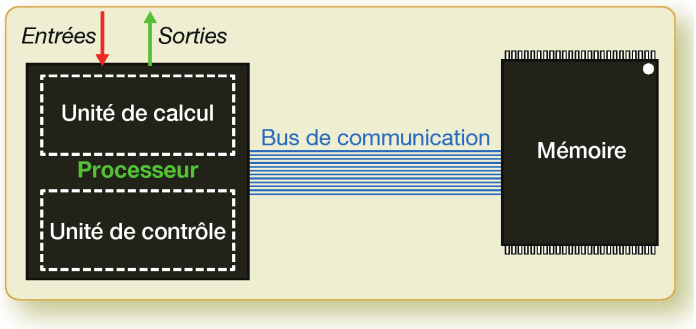
- L'**unité centrale** représentée par le **microprocesseur** (ou **Central Processing Unit**) qui est composé essentiellement d'une **Unité de Contrôle** (ou de Commande) qui lit en mémoire le programme et décode les instructions et d'une **Unité de Calcul** qui exécute la séquence d'instructions. Dans l'unité de calcul, une **Unité Arithmétique et Logique** (UAL) peut réaliser des opérations simples (addition, soustraction, opérations logiques). Le CPU dispose de plusieurs mémoires appelées **registres** qui sont d'un accès très rapide puisque situées en son coeur. Certains registres permettent de stocker des données, d'autres servent à stocker des adresses mémoires comme le **compteur ordinal** (ou Programme Conter) qui pointe vers l'adresse de la prochaine instruction ;

- La **mémoire physique** qui se décline en deux types principaux. La **mémoire vive** ou **RAM** pour `Random Access Memory` contient les **données** et les **programmes** , elle accessible en lecture et en écriture mais elle est volatile (elle s'efface lorsque le courant est coupé). La **mémoire morte ROM** (`Read Only Memory`) non volatile est accessible en lecture seule, elle contient les paramètres d'usine et les informations nécessaires au démarrage.

  La mémoire vive est constituée de cellules physiques (bascules D) pouvant coder chacune 1 bit. On peut voir la mémoire vive comme une immense matrice constitués de registres montés en parallèles où chaque cellule mémoire est repérée par sa ligne et sa colonne : le signal d'horloge est envoyé vers la cellule mémoire (ou registre) souhaité en écriture et un multiplexeur permet de sélectionner la cellule souhaitée en lecture. Les cellules mémoire sont organisées en **octets** (8 bits) et en **mots mémoire** (32 ou 64 bits sur les architectures récentes). **Chaque mot mémoire de la RAM est directement accessible en écriture et en lecture**, on parle de **mémoire à accès direct**. Lorsque le processeur veut accéder à une cellule mémoire, il dépose son adresse sur un bus mémoire, celle-ci est réceptionnée par un contrôleur qui sélectionne la bonne cellule à l'aide d'un multiplexeur. Néanmoins si les accès mémoire se font sur les mots mémoire de 32 ou 64 bits, les adresses mémoire sont comptées en octets et non en mots.

- Les **périphériques** d'entrée ou de sortie qui permettent au processeur de communiquer avec des dispositifs externes (mémoire de masse comme un disque dur, clavier,écran) pour **la lecture** ou **l'écriture** d'informations.

  Des circuits électroniques comme la puce *Direct Memory Access* peuvent aider le microprocesseur en permettant un accès direct des périphériques à la mémoire physique (mais si elle soulage le processeur,la puce DMA est moins rapide que lui).



Le processeur,la mémoire et les périphériques communiquent par le biais de **bus** :

- le **bus de données** transfèrent données et instructions entre l'unité centrale et la mémoire, il est bidirectionnel ;

- le **bus d'adresses** véhicule les adresses des instructions à charger dans le registre d'instruction ou dans un registre particulier, il est unidirectionnel du processeur vers la mémoire ;

- le **bus de commandes** transmet les micro-instructions vers les différents composants du microprocesseur. Ces micro-instructions résultent du décodage de l'instruction qui se trouve dans le registre d'instruction par un composant appelé séquenceur : une instruction se traduisant par plusieurs actions ou micro-instructions.

De plus, le processeur communique avec les périphériques (clavier, souris, écran ...) par le biais de **bus d'entrées et de sorties**.

2.2 La carte mère

Le processeur, la mémoire et tous les composants principaux sont fixés sur la **carte mère** . Ils sont fixés ou soudés sur la **carte mère** de l'ordinateur qui est alimentée par un bloc d'alimentation chargé de convertir la tension alternative du secteur en tension continue.
La carte mère supporte les bus qui relient les différents composants entre eux. Le microprocesseur est fixé sur un support appelé **socket**.
Tous ces composants et le microprocesseur en particulier diffusent de la chaleur et des ventilateurs et autres radiateurs sont utilisés pour faire circuler l'air et dissiper la chaleur.
Une **horloge** cadence le déroulement des tâches effectuées par les composants de la carte mère en leur envoyant des signaux. (cadences comprises entre 1 MegaHertz et 1 GigaHertz).

Les cellules mémoires de la mémoire vive sont des **transistors** utilisées comme des **condensateurs**, ils se déchargent ce qui nécessite un rafraîchissement de la mémoire toutes les 15 ns environ. On parle de DRAM pour **RAM dynamique**.La mémoire RAM étant **volatile**, pour conserver des données et des programmes lorsque l'ordinateur est éteint, on utilise des **mémoires de masse** comme des disques durs (support magnétique).
Dans les mémoires de masse, les données sont organisées en une arborescence de répertoires et de fichiers par des **systèmes de fichiers** comme `FAT` ou `NTFS` sous Windows ou `Ext2` sous Linux. L'accès au système de fichiers comme aux différentes ressources matérielles (processeur, mémoire ...) est géré par un programme chargé en mémoire vive au démérrage et qui s'appelle **système d'exploitatio**n.
Il faut garder à l'esprit **la différence de temps d'accès** entre une **mémoire vive** (de l'ordre de 60 ns) et la **mémoire de masse** (de l'ordre de 10 ms) soit l'équivalent d'un temps d'accès de 1 s en mémoire vive pour 12 jours en mémoire de masse.
Le microprocesseur possède des **registres généraux**, où il peut stocker des résultats intermédiaires. Ces registres sont de petite taille mais d'un accès beaucoup plus rapide que la RAM. La taille de ces registres généraux caractérise l'architecture 32 bits ou processeur 64 bits du micro-processeur : un des registres, appelé **Program Counter**, pointe vers l'adresse de la prochaine instruction lue en mémoire, il a donc la même largeur en bits que le bus d'adresse.
Par ailleurs le microprocesseur possède en plus des registres (de 32 ou 64 bits) des espaces de **mémoire cache** de niveau L1 (entre 8 et 128 Ko et), L2 (entre 128 Ko et 1 Mo) : il s'agit de **RAM statique** (pas besoin de rafraîchissement), d'**accès très rapide** (10 ns).
Il existe aussi des mémoires caches de niveaux L3 ou L4 (de 1 à plusieurs Mo) à proximité du processeur sur la carte mère. Il est difficile de réaliser des RAM statiques avec une grande capacité de stockage et elles coûtent très chers.
Enfin, un ordinateur contient des mémoires inscrites dans le « dur » de la carte mère, il s'agit de mémoires persistantes appelées **ROM** (en lecture seule) comme le **CMOS** qui contient les informations du programme **BIOS** (Basic Input Output System) gérant le démarrage de la machine (auto-test, lecture du premier secteur de disque,lancement du système d'exploitation).
Le BIOS joue un rôle essentiel :

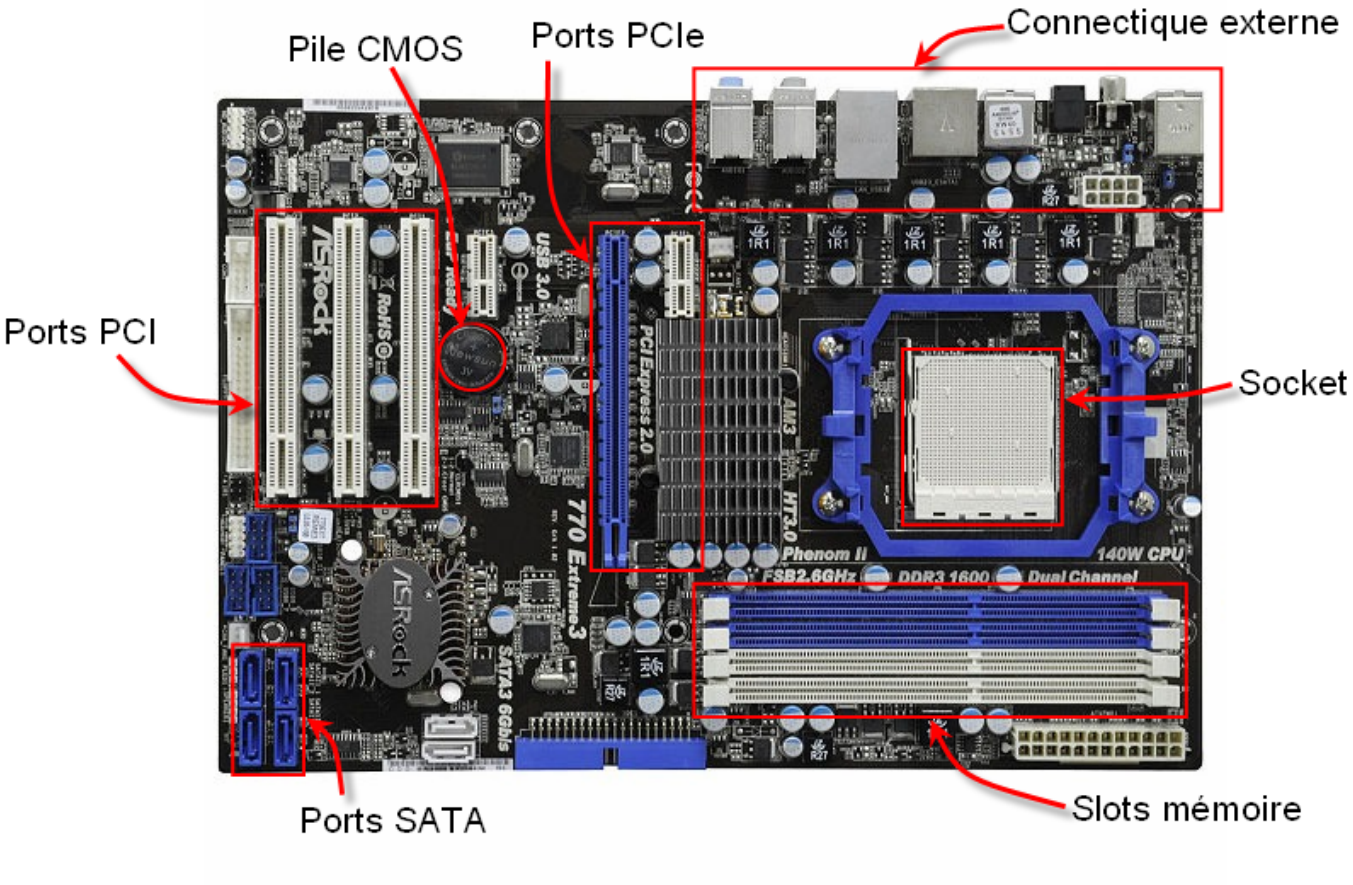
- il initialise et identifie tous les composants de la carte mère et les périphériques qui lui sont connectés (phase POST pour Power-On Self-Test) ;
- il recherche le chargeur d'amorçage (ou **bootloader**) du système d'exploitation présent sur le premier périphérique bootable défini dans l'ordre d'amorçage (phase de boot)

Un composant particulier, le **chipset joue le rôle de gare de triage (le processeur étant le chef de gare)** et fait coopérer les autres composants. Il est constitué de deux parties :

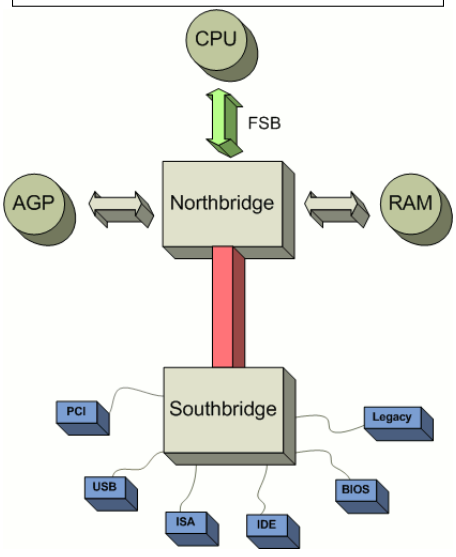
- le **north-bridge** est chargé de contrôler les composants rapides (processeur, mémoire RAM, mémoire video, bus graphiques AGP ou PCI 16x) ;
- le **south-bridge** est chargé de contrôler les composants lents (ports parallèles ou séries de périphériques comme les anciennes imprimantes ou souris, bus USB, bus reliés aux contrôleurs PCI des cartes réseaux, ou aux contrôleurs IDE ou SATA des disques durs) ;

Comme complément d'information on pourra consulter la page web dont est issue la photo ci-après :
`http://www.siteduzero.com/tutoriel-3-555196-la-carte-mere.html`

Carte-mère (image extraite du Site du Zéro)



Chipset (image extraite de Wikipedia)



Exercice 9

Sur un systèmes Linux, on pourra utiliser les commandes suivantes en mode console pour obtenir quelques informations sur son matériel :

1. `free -mt` pour afficher les quantités de mémoire vive et Swap libres et utilisées ;
2. `df` pour afficher la quantité d'espace disque disponible et occupée dans les systèmes de fichiers montés ;
3. `du` ou `ls -l` pour afficher l'espace occupé par un répertoire. (dans les 3 cas précédents l'option `-h` permet un affichage en kilooctets ou megaoctets plutôt qu'en octets) ;
4. `lshw` (pour Hardware Lister) permet de recueillir des informations très complètes sur l'ensemble des composants du système. Pour récupérer ces données dans un fichier materiel.html, on peut utiliser la syntaxe : `lshw -html > /home/materiel.html`.

Exercice 10

1. La largeur en bits des registres du processeur et du bus d'adresses détermine la taille de la mémoire adressable. Longtemps l'architecture 32 bits a dominé, actuellement, les ordinateurs vendus sont d'architecture 64 bits.
   - Déterminer la taille en Go de la mémoire physique adressable dans un registre de 32 bits ? et dans un registre de 64 bits ?
   - Si on suppose que la loi de Moore continue à se vérifier et que la taille de la mémoire physique disponible double tous les 2 ans, dans combien de temps les registres 64 bits seront-ils insuffisants ?
2. Le débit du bus de données reliant le processeur (via le chipset) à la mémoire RAM est égal au produit de la fréquence du bus par sa largeur. Le temps de latence (ou temps d'accès) de la mémoire est l'inverse du débit.
   Calculer le débit en `Mo/s` et le temps de latence en `ns` offerts par une mémoire présentant les caractéristiques suivantes :
   `SDRAM - 64 MO - Bus 64 bits - DIMM 168 broches/100 Mhz`

2.3 Système d'exploitation

En pratique, lorsqu'un programme est exécuté, un **processus** est créé et un espace est alloué en mémoire vive pour stocker, son **texte** (ou code),ses **données** (les variables) et sa **pile** (paramètres et adresses de retour des fonctions successivement appelées).
De nos jours les utilisateurs utilisent plusieurs programmes en même temps mais comment est-ce possible avec un processeur qui n'exécute qu'une instruction à la fois ?
Un énorme programme appelé **système d'exploitation** fait l'interface entre les programmes et le matériel. Le **noyau du système d'exploitation** est chargé en mémoire vive au démarrage de la machine.
Les premiers ordinateurs fonctionnaient sans sytème d'exploitation. De nos jours, il existe deux grandes familles de systèmes d'exploitation : Unix (Mac OS/X, Solaris, Free Bsd, GNU/Linux) et Windows. A ce sujet, on pourra consulter avec profit les pages 192 et 193 du manuel ISN de **Gilles Dowek** ou l'introduction de l'ouvrage « Systèmes d'exploitation » d'**Andrew Tanenbaum**.
Le système d'exploitation remplit principalement deux tâches :

- il offre aux programmeurs d'application une interface de programmation qui représente une **abstraction** du matériel (UC, mémoire), par exemple un fichier est une abstraction de la mémoire.
- il gère le partage des ressources matérielles entre les différents programmes et utilisateurs : on parle de **multiplexage temporel** pour le partage du temps d'utilisation du microprocesseur et de **multiplexage spatial** pour le partage de la mémoire.

  Un **ordonnanceur** ou *scheduler* répartit ainsi le temps de processeur entre les différents processus, qui s'exécutent à tour de rôle par quantum de quelques millisecondes. L'illusion du parallélisme est ainsi créée.

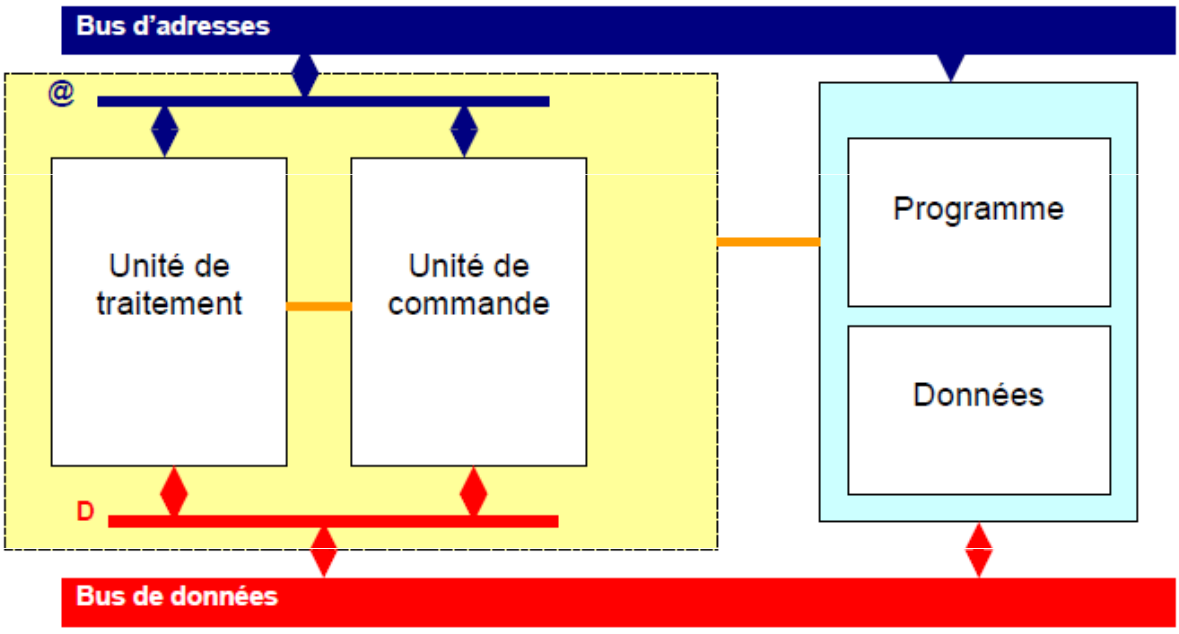
Lorsqu'un programme s'exécute, le processeur est en **mode utilisateur**. Si le programme veut accéder aux périphériques ou à la mémoire, un **appel système** est lancé qui fait passer le processeur en **mode noyau**. Le système d'exploitation prend alors la main et décide si le programme peut accéder à la ressource demandée.
Le système protège ainsi l'accès aux ressources et permet la multiprogrammation.

Pour la gestion de la mémoire, une méthode de **la mémoire virtuelle** est couvent adoptée. Un programme n'a pas besoin d'être chargé entièrement en mémoire pour être exécuté. Chaque programme dispose de son propre espace d'adressage mémoire découpé en entités appelés `pages`.Un programme peut ainsi adresser plus que la mémoire physique. Un dispositif matériel adjacent au processeur et appelé **Memory Management Unit** transcrit dynamiquement les adresses mémoire virtuelles en adresses mémoires physiques.

Les pages utilisées par le processus en cours sont mappées en mémoire vive par le système d'exploitation. Lorsque des pages ne sont plus utilisées ou que la mémoire vive est saturée, le système d'exploitation déplace des pages de la mémoire vive vers une mémoire de masse (disque). Ces échanges entre mémoire vive et mémoire de masque s'appellent le **swapping** . Le système d'exploitation Linux utilise ainsi une partition de **swap** sur une mémoire de masse pour stocker les pages qui font le va-et-vient entre disque et mémoire vive.

Les images suivantes sont extraites d'un diaporama sur l'architecture des ordinateurs réalisée par une professeur de l'INSA (Véronique Eglin) et qui m'a été présenté pendant ma formation ISN en 2011.

Ordinateur et architecture



Principe d'organisation du matériel





Unité centrale

Unité de calcul

Exécution des instructions

Unité de commande

Lecture/décodage instructions

Bus

Périphériques

Unité de calcul - terminologie:

-- *unité de traitement (u.t) qui exécute et organise les travaux réalisés par le processeur*

-- *d'unité arithmétique et logique (u.a.l.) spécialisée dans les calculs simples (+, -, ...)*

-- *d'unité mathématique spécialisée dans les calculs plus complexes : c'est la calculatrice scientifique de l'unité de traitement.*

Mémoire centrale

Instructions

+

Données

Code instructions

+

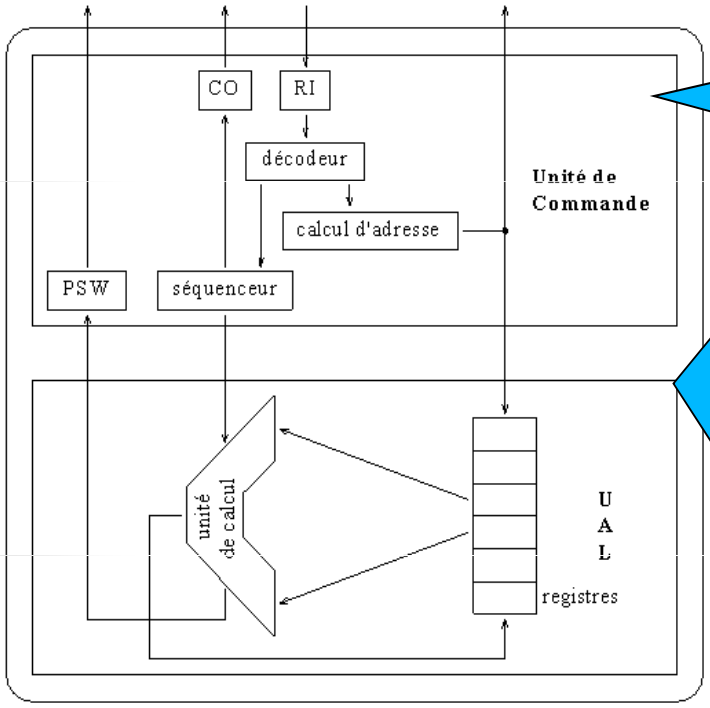
Données binaires

Petites « barrettes » enfichables

Interface de gestion des communications = connections électriques

INSA LYON

L'unité centrale: DEUX GRANDES PARTIES



L'unité de commande:

- décode des *instructions*
- calcule des *adresses* des données à traiter.
- joue le rôle de *séquenceur* qui contrôle le fonctionnement des circuits

L'UAL (ou de traitement):

- exécute les instructions
- utilise des registres pour stocker des données à traiter, des résultats intermédiaires, des informations de commande, des adresses

➢ gestion des opérations arithmétiques ou logiques,

➢ registre instruction (RI) qui contient l'instruction à exécuter,

➢ compteur ordinal (CO) qui pointe sur la prochaine instruction,

➢ registre d'état (PSW : Processor Status Word) qui contient des informations sur l'état du système

Les mémoires: deux grandes catégories

- On appelle « **mémoire** » tout composant électronique capable de stocker temporairement des données.

- On distingue ainsi **deux grandes catégories de mémoires** :

  - La mémoire centrale :
    - mémorise temporairement les données
    - réalisée à l'aide de micro-conducteurs
    - appelée aussi **mémoire vive**

  - La mémoire de masse:
    - stocke des informations à long terme
    - stockage magnétiques (disque dur, CD ROM..)
    - appelée aussi **mémoire physique**

Les mémoires : hiérarchies

Les mémoires sont ordonnées en fonction des critères suivants

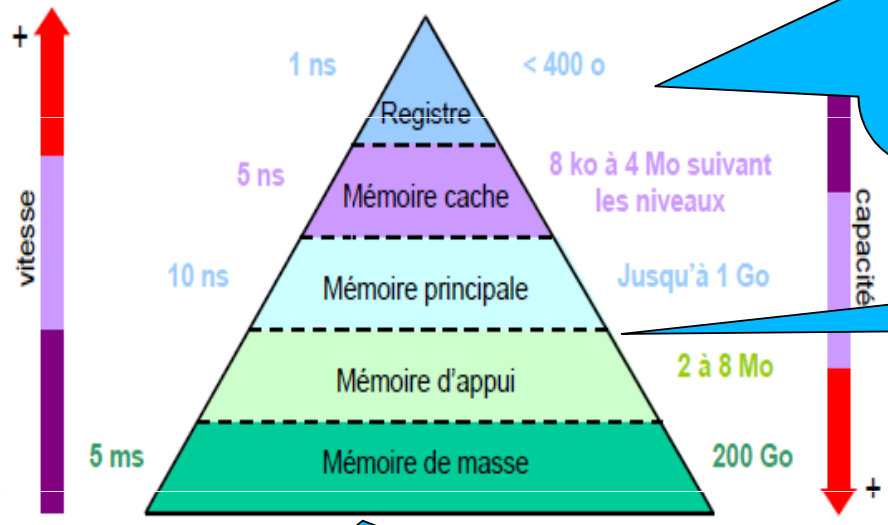
- Adresse (référence du mot mémoire)
- Type et temps d'accès
- Capacité ou taille (nombre de mots)

=> Différencier une mémoire dynamique (DRAM, volatile à rafraîchir) d'une mémoire statique (SRAM, mémoire cache très rapide);

=> Différencier une mémoire vive (RAM) d'une mémoire morte (ROM)

=> Evaluer quantitativement l'organisation des mémoires

Les mémoires : hiérarchie



La mémoire cache:

- données les plus récemment utilisées.
- contient une petite partie de la mémoire centrale
- si l'information désirée se trouve dans le cache, pas besoin d'aller en mémoire centrale (gain de temps, 2 à 5 fois)
- de type SRAM

« faire gagner du temps au processeur »

Mémoire centrale
De type DRAM, moins cher, plus lente

La mémoire d'appui : rôle de mémoire cache, intermédiaire entre la mémoire centrale et les mémoires de masse.

La mémoire de masse: grande capacité utilisée pour le stockage permanent ou la sauvegarde des informations.