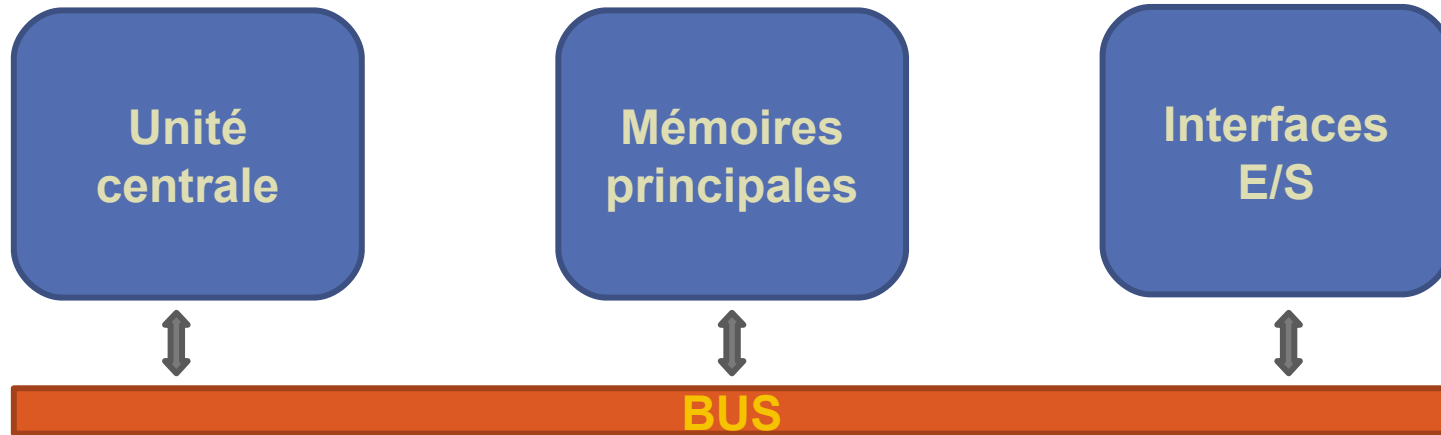


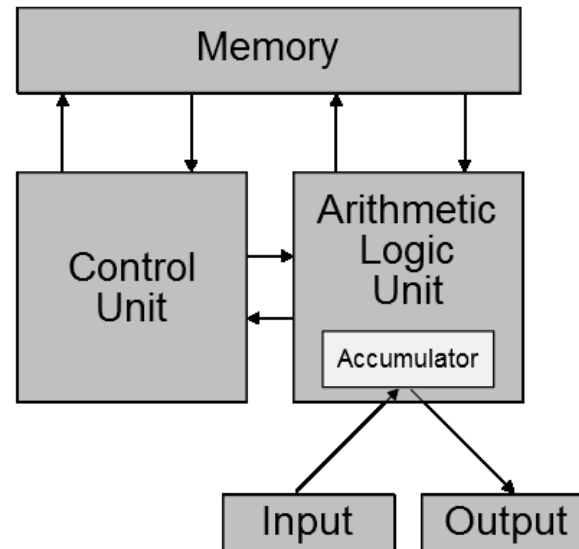
ARCHITECTURE

ISN 2012

vue schématique de l'architecture de Von Neumann



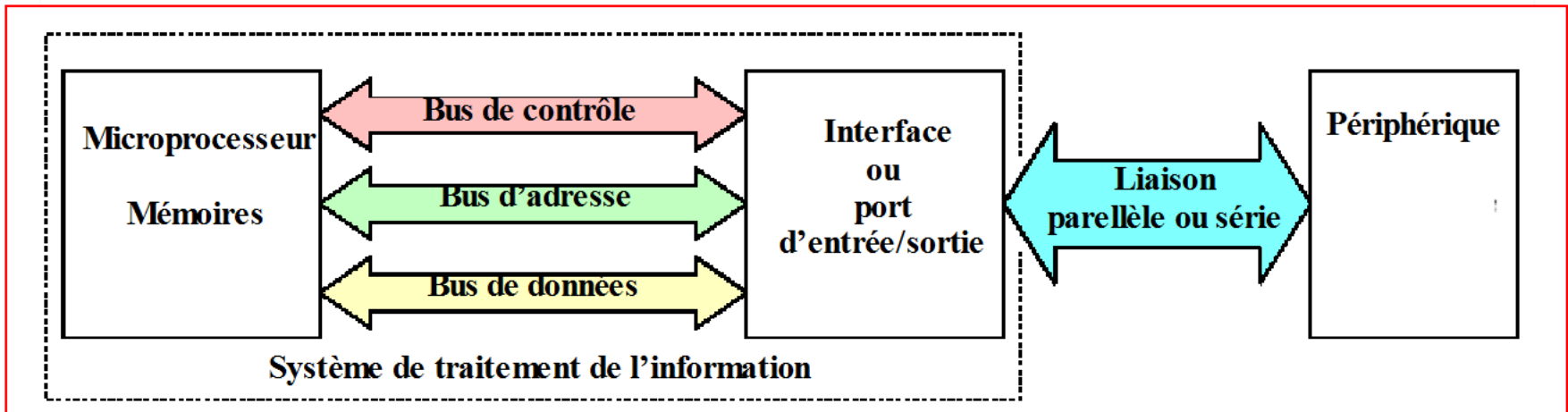
Les différents organes du système sont reliés par des voies de communication appelées **bus**.



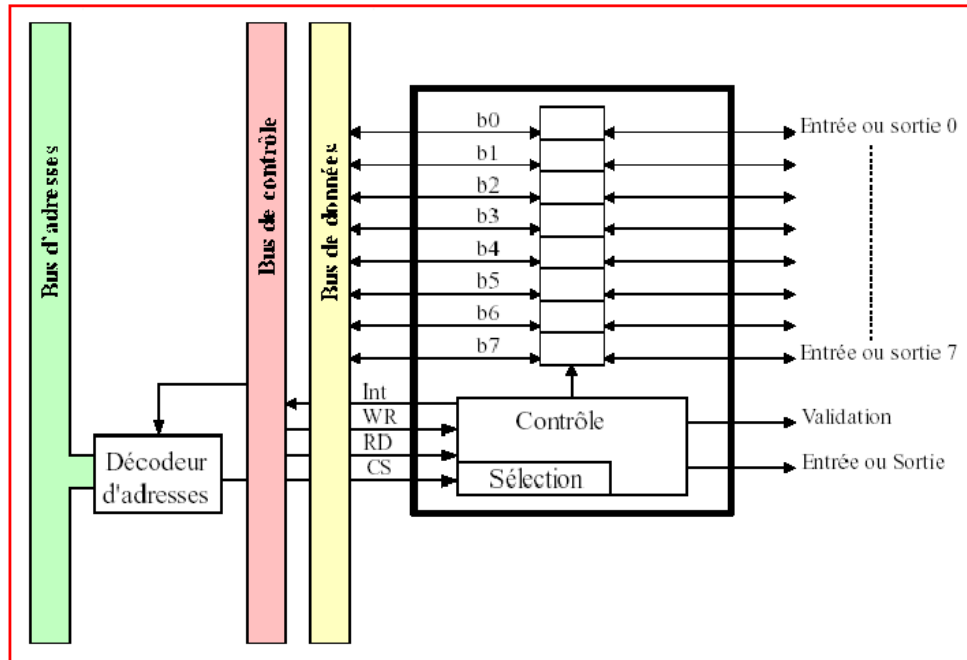
La même architecture vue par Wikipédia (la notion de bus est moins présente).

Les interfaces D'entrées/sorties

Elles permettent d'assurer la communication entre le microprocesseur et les périphériques. (capteur, clavier, moniteur ou afficheur, imprimante, modem, etc...).



Interface parallèle

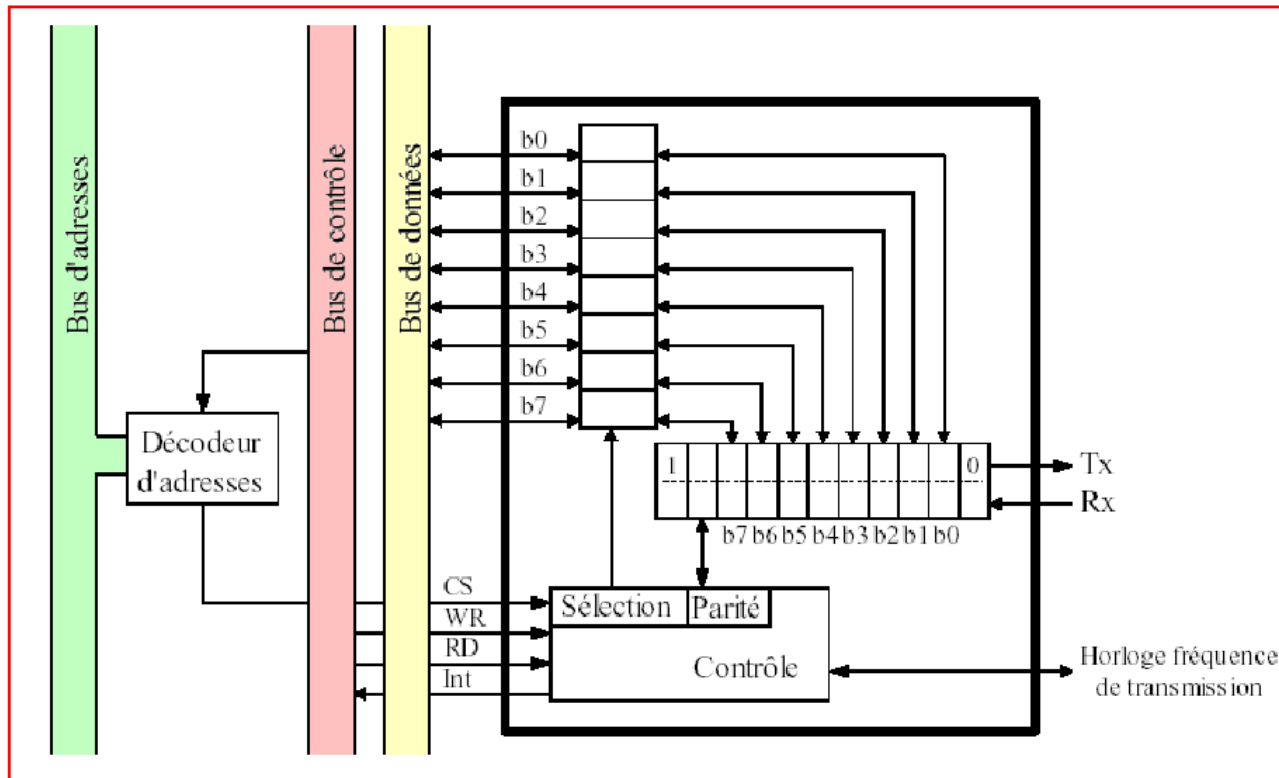


Ce type d'interface (PIO : Parallel Input Output) permet de connecter habituellement une imprimante, une mémoire de masse externe, une commande de moteur, etc...

Les "n" bits de la donnée à transmettre entre le système et le périphérique sont envoyés simultanément. Le câble de transmission nécessite un nombre important de conducteurs (n bits + la masse + des lignes de contrôle).

Le temps de transmission d'un mot est très court, mais ne permet pas de couvrir des distances très importantes.

Interface Série



Les "n" bits de la donnée à transmettre entre le système et le périphérique sont envoyés les uns après les autres (en série). Le câble de transmission nécessite un nombre réduit de conducteurs: Tx transmission, Rx réception, la masse + des lignes de contrôle (3 fils minimum). Le temps de transmission d'un mot est plus important (10 fois plus) qu'une liaison parallèle, mais permet de couvrir des distances importantes.

La mémoire principale

Elle contient les instructions du ou des programmes en cours d'exécution et les données associées à ce programme.

Physiquement, elle se décompose souvent en :

- une mémoire morte (ROM = Read Only Memory) chargée de stocker le programme. C'est une mémoire à lecture seule.
- une mémoire vive (RAM = Random Access Memory) chargée de stocker les données intermédiaires ou les résultats de calculs. On peut lire ou écrire des données dedans, ces données sont perdues à la mise hors tension.

Remarque : Les disques durs, disquettes, CDROM, etc... sont des périphériques de stockage et sont considérés comme des mémoires secondaires.

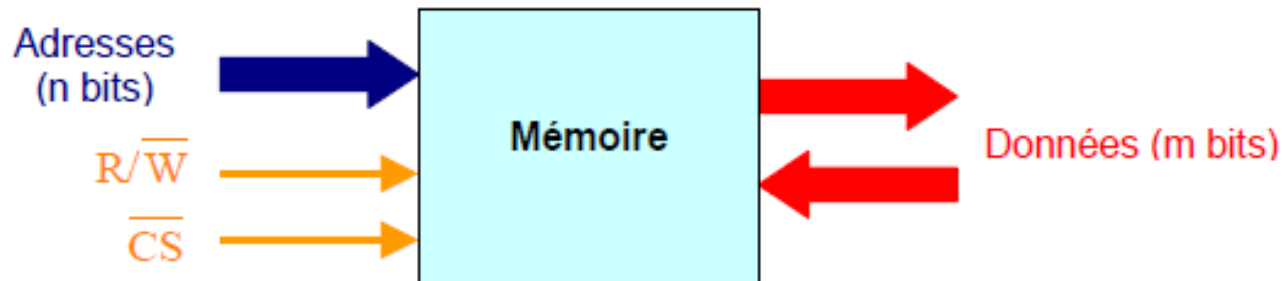
Organisation d'une mémoire

Une mémoire peut être vue comme une armoire où chaque tiroir contient une et une seule **donnée**.

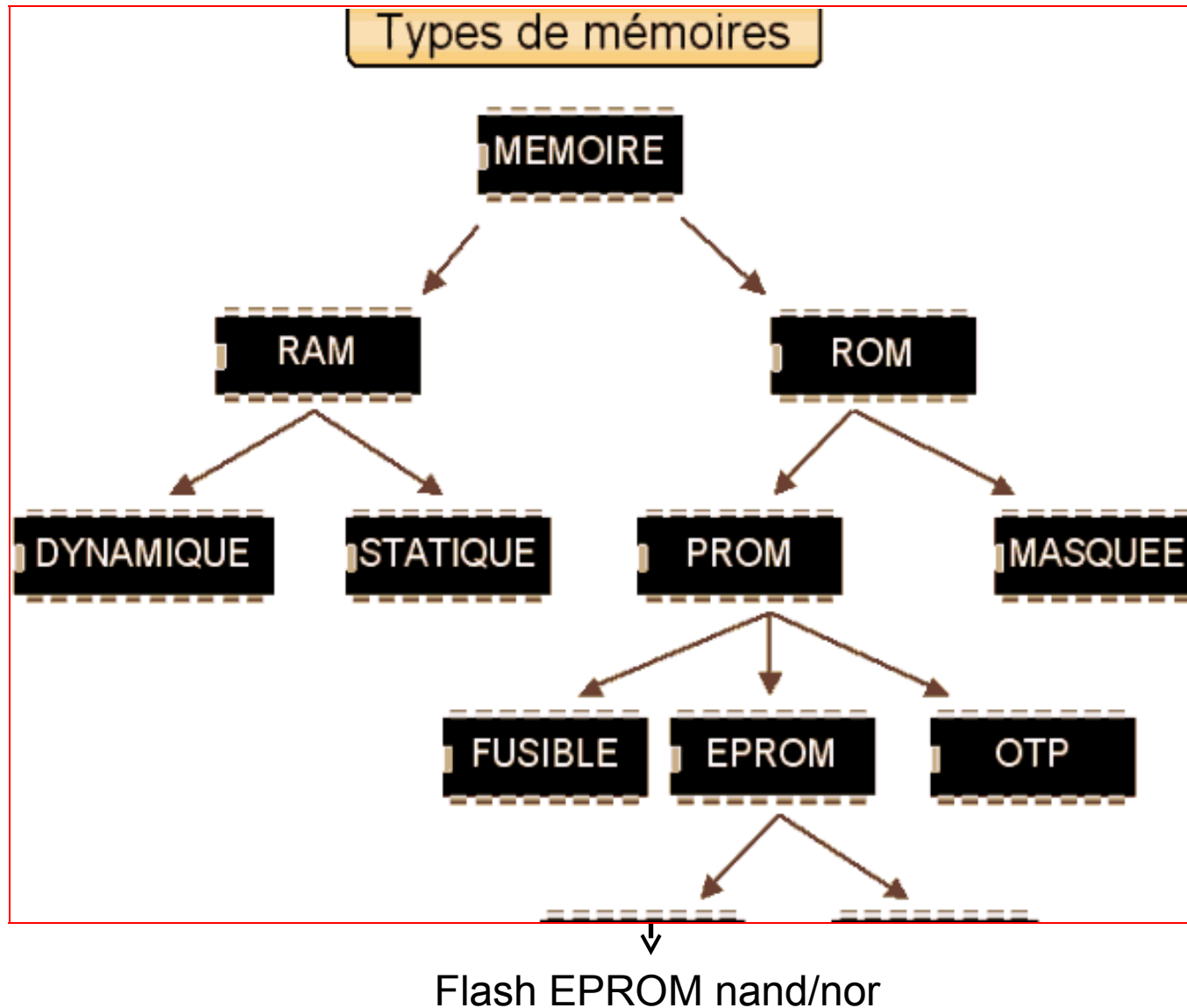
Chaque tiroir est repéré par une **adresse**.

Le nombre de fils d'adresses d'un boîtier mémoire définit donc le nombre de cases mémoire que comprend le boîtier. Le nombre de fils de données définit la taille des données que l'on peut sauvegarder dans chaque case mémoire.

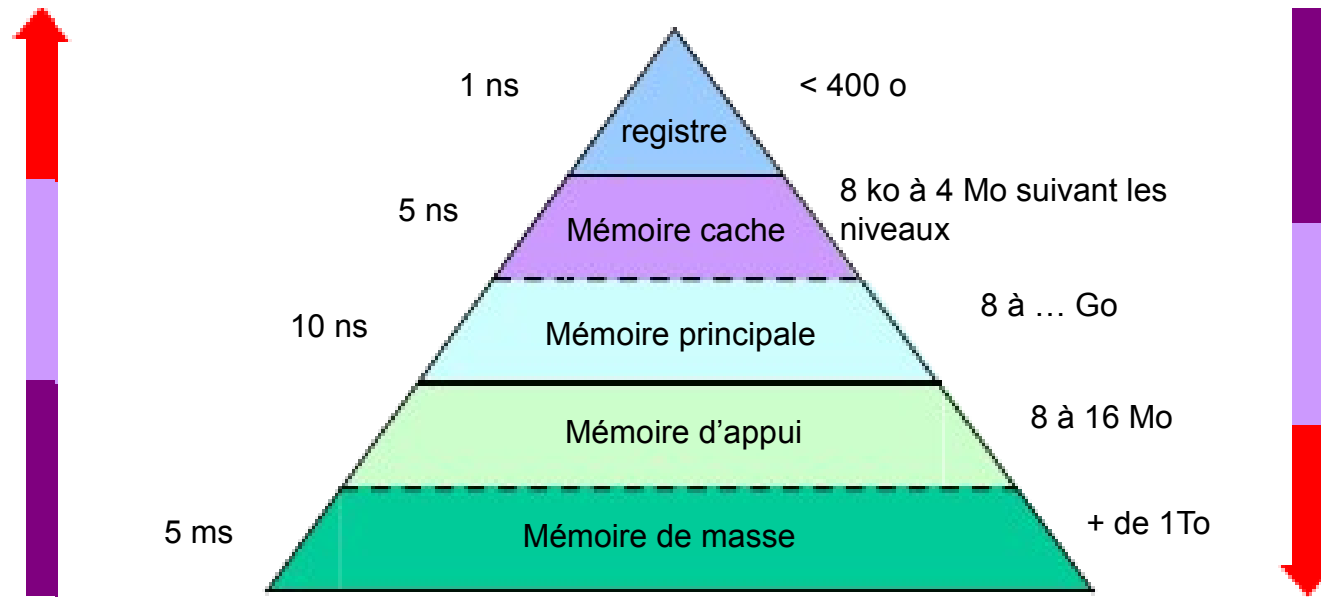
En plus du bus d'adresses et du bus de données, un boîtier mémoire comprend une entrée de commande qui permet de définir le type d'action que l'on effectue avec la mémoire (lecture/écriture) et une entrée de sélection qui permet de mettre les entrées/sorties du boîtier en haute impédance.



Différents type de mémoires



Notion de hiérarchie mémoire



Les registres sont les éléments de mémoire les plus rapides. Ils sont situés au niveau du processeur et servent au stockage des opérandes et des résultats intermédiaires.

La mémoire cache est une mémoire rapide de faible capacité destinée à accélérer l'accès à la mémoire centrale en stockant les données les plus utilisées.

La mémoire principale est l'organe principal de rangement des informations. Elle contient les programmes (instructions et données) et est plus lente que les deux mémoires précédentes.

La mémoire d'appui sert de mémoire intermédiaire entre la mémoire centrale et les mémoires de masse. Elle joue le même rôle que la mémoire cache.

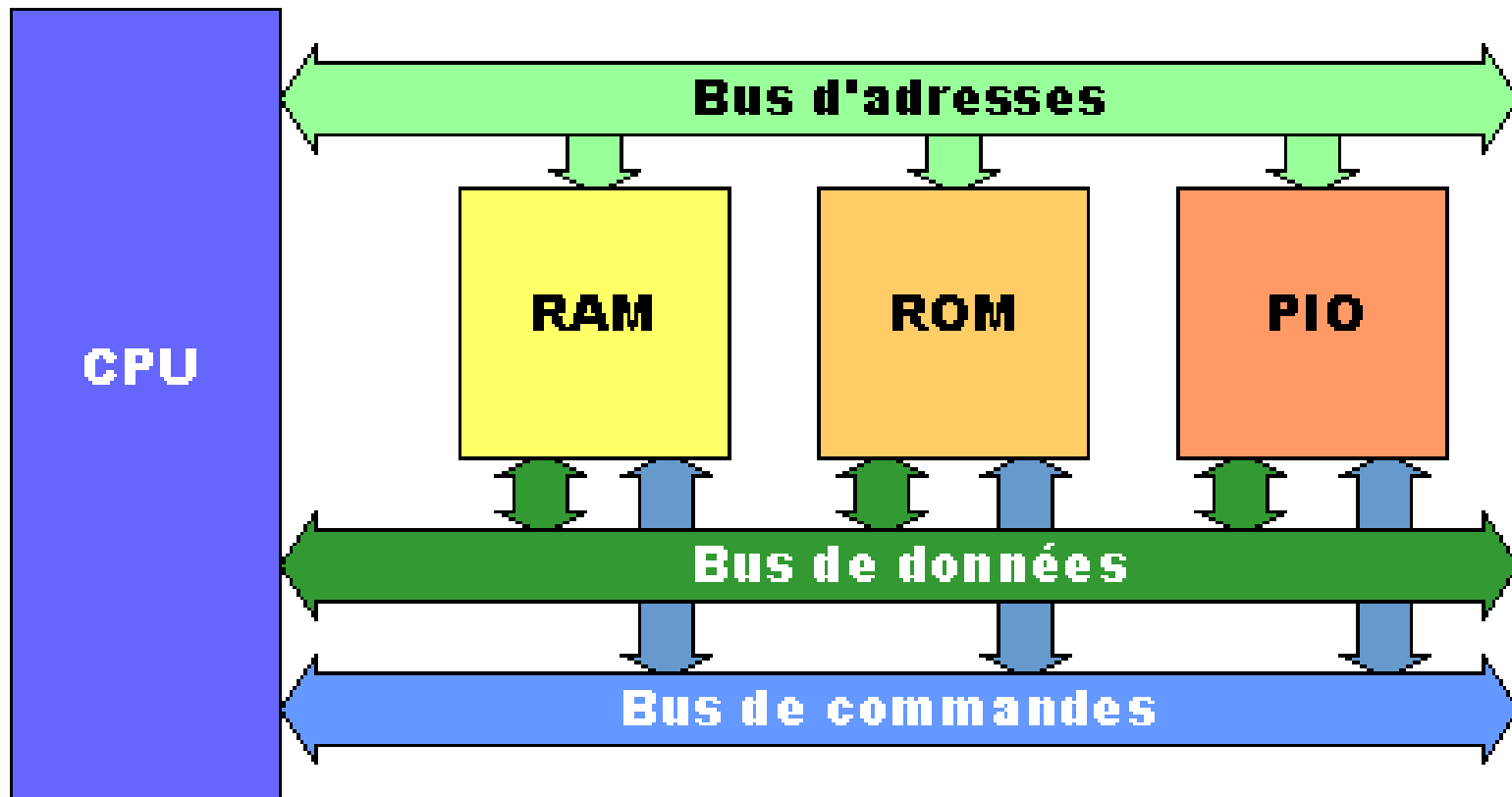
La mémoire de masse est une mémoire périphérique de grande capacité utilisée pour le stockage permanent ou la sauvegarde des informations. Elle utilise pour cela des supports magnétiques ou optiques.

Les bus

Un bus est un ensemble de fils qui assure la transmission du même type d'information. On retrouve trois types de bus véhiculant des informations en parallèle dans un système de traitement programmé de l'information :

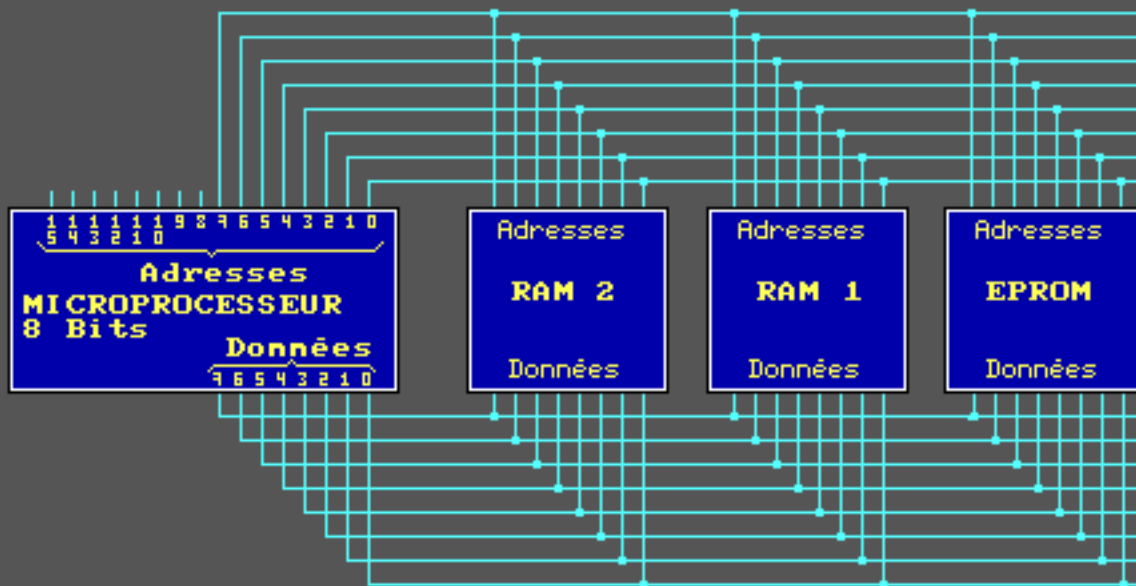
- un bus de données : bidirectionnel qui assure le transfert des informations entre le microprocesseur et son environnement, et inversement. Son nombre de lignes est égal à la capacité de traitement du microprocesseur.
- un bus d'adresses: unidirectionnel qui permet la sélection des informations à traiter dans un *espace mémoire* (ou *espace adressable*)
- un bus de commande: constitué par quelques conducteurs qui assurent la synchronisation des flux d'informations sur les bus des données et des adresses.

Les bus



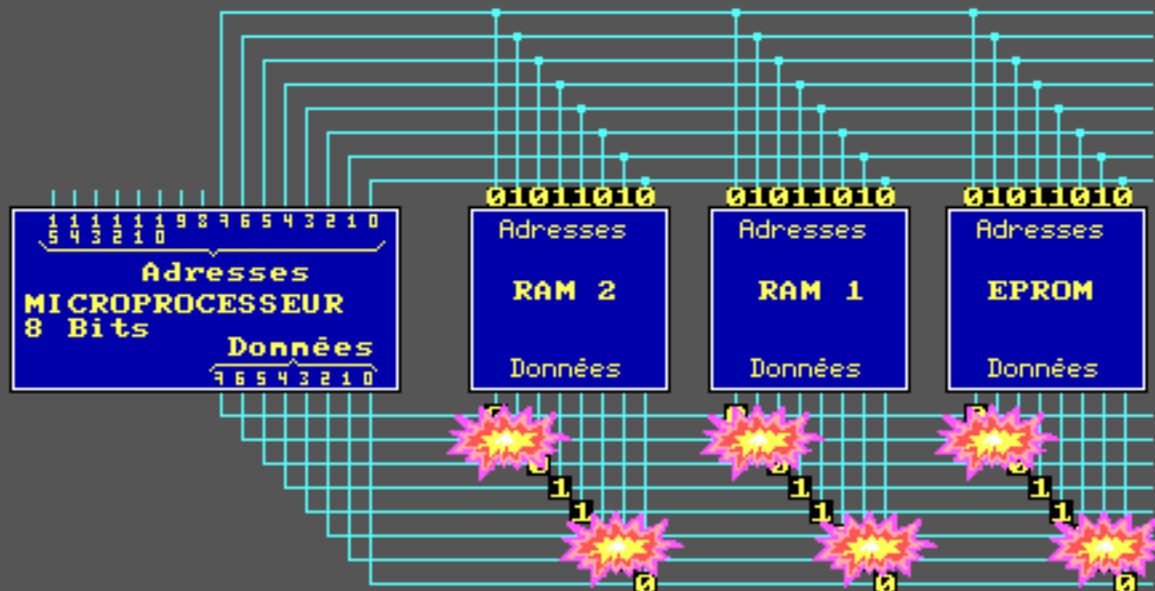
Le décodage d'adresses

En réalité toutes les mémoires sont raccordées en parallèle sur le bus d'adresse et de données (mettons de côté le bus de commande).



Le décodage d'adresses

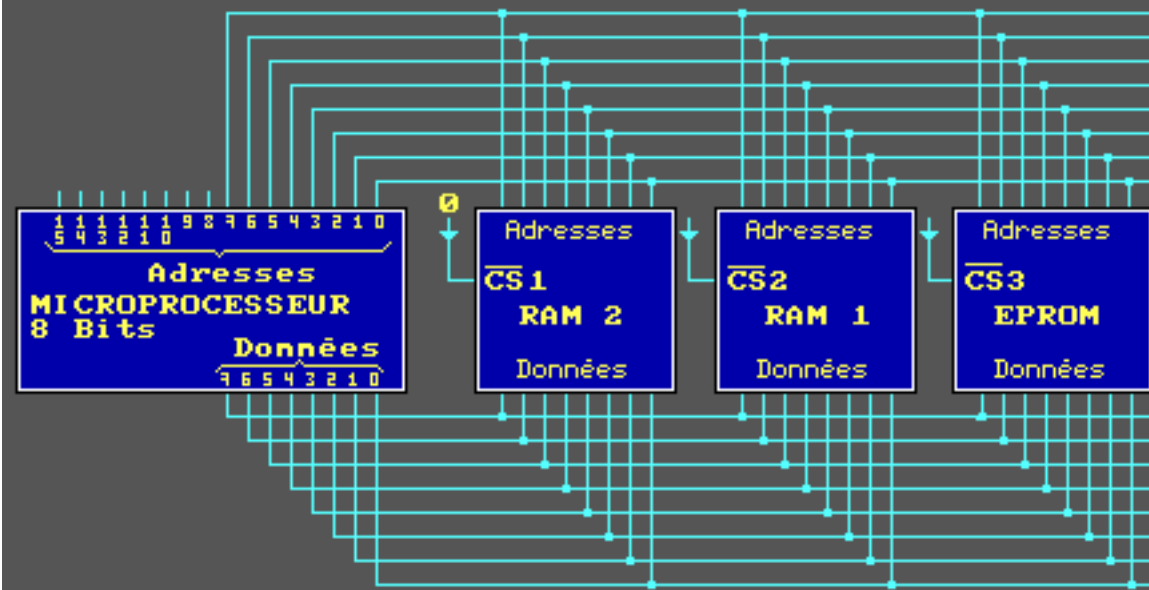
En réalité toutes les mémoires sont raccordées en parallèle sur le bus d'adresse et de données (mettons de côté le bus de commande).



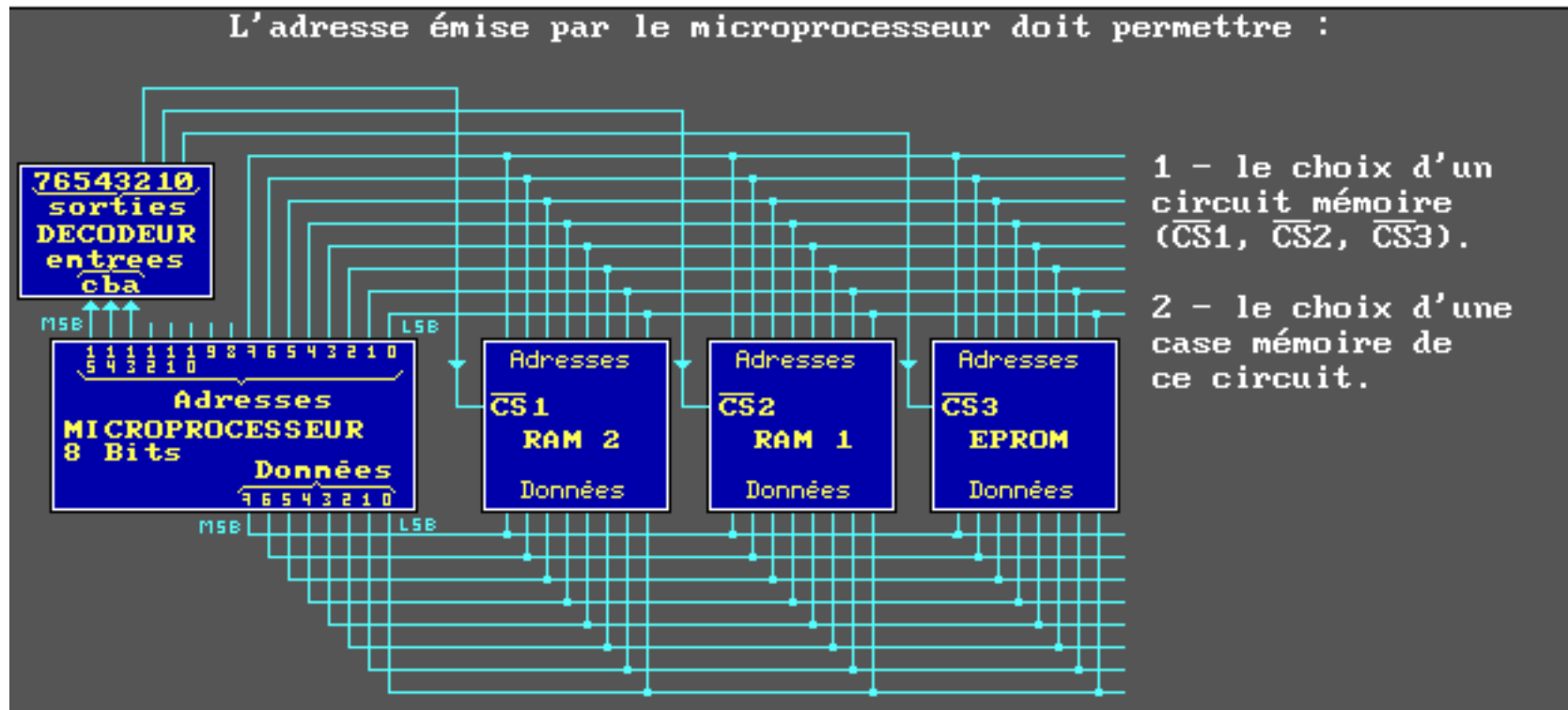
Problème:
dans ce cas
nous obtenons
un "conflit"
d'informations".

Le décodage d'adresses

D'où l'utilité d'insérer une commande \overline{CS} (Chips Select) pour qu'une seule mémoire puisse dialoguer à la fois.

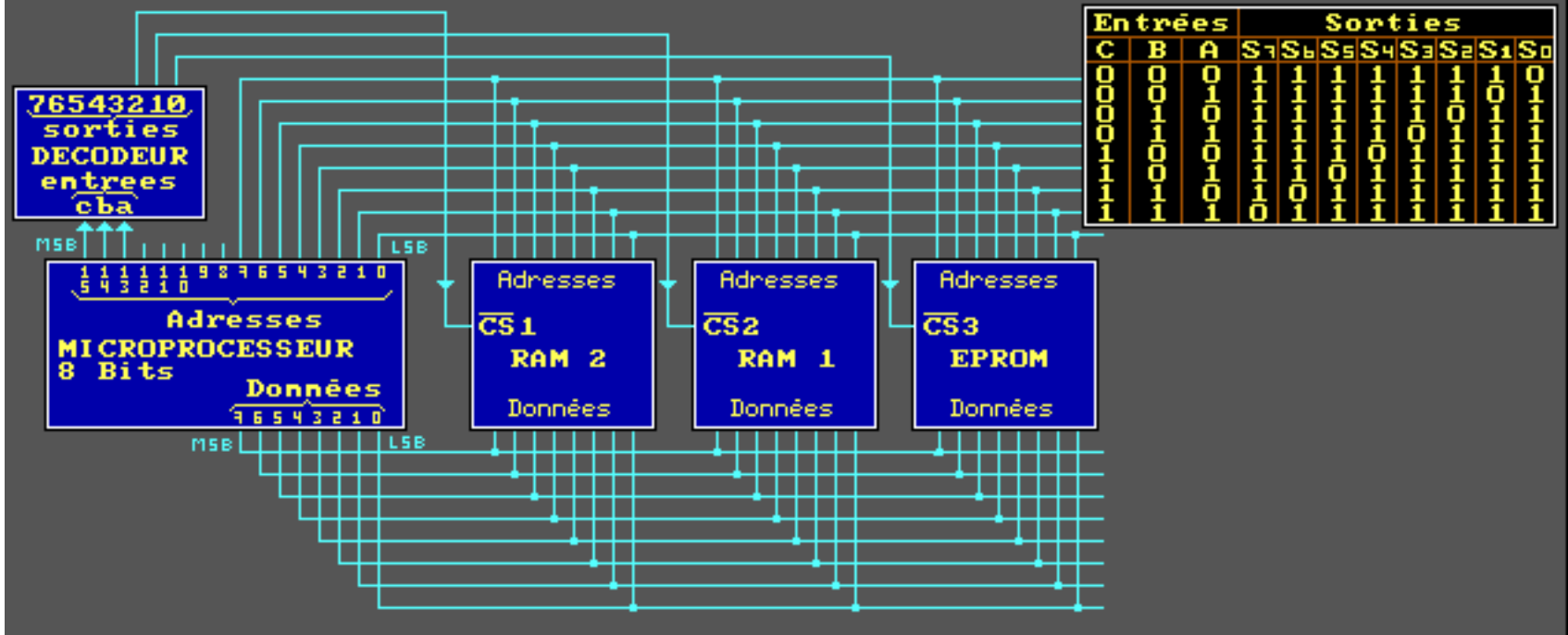


Le décodage d'adresses



Le décodage d'adresses

Rappelons brièvement la table de vérité du décodeur.



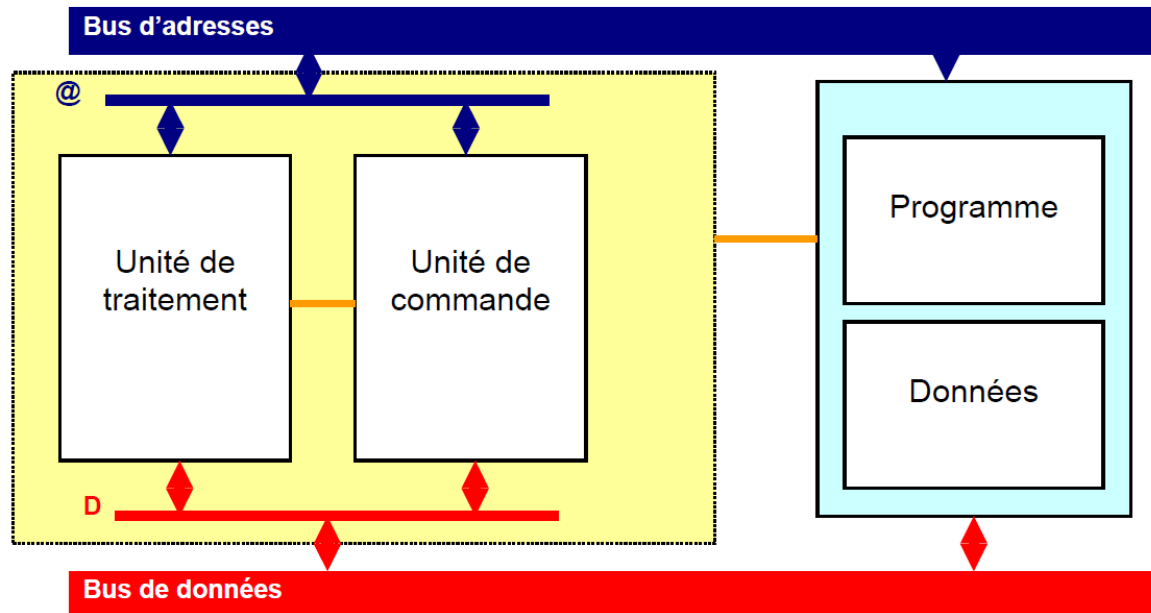
L'unité centrale

Elle est composée par le microprocesseur qui est chargé d'interpréter et d'exécuter les instructions d'un programme, de lire ou de sauvegarder les résultats dans la mémoire et de communiquer avec les unités d'échange. Toutes les activités du microprocesseur sont cadencées par une horloge.

On caractérise le microprocesseur par :

- sa fréquence d'horloge.
- le nombre d'instructions par secondes qu'il est capable d'exécuter : en **MIPS**
- la taille des données qu'il est capable de traiter : en **bits**

Architecture de base d'un microprocesseur

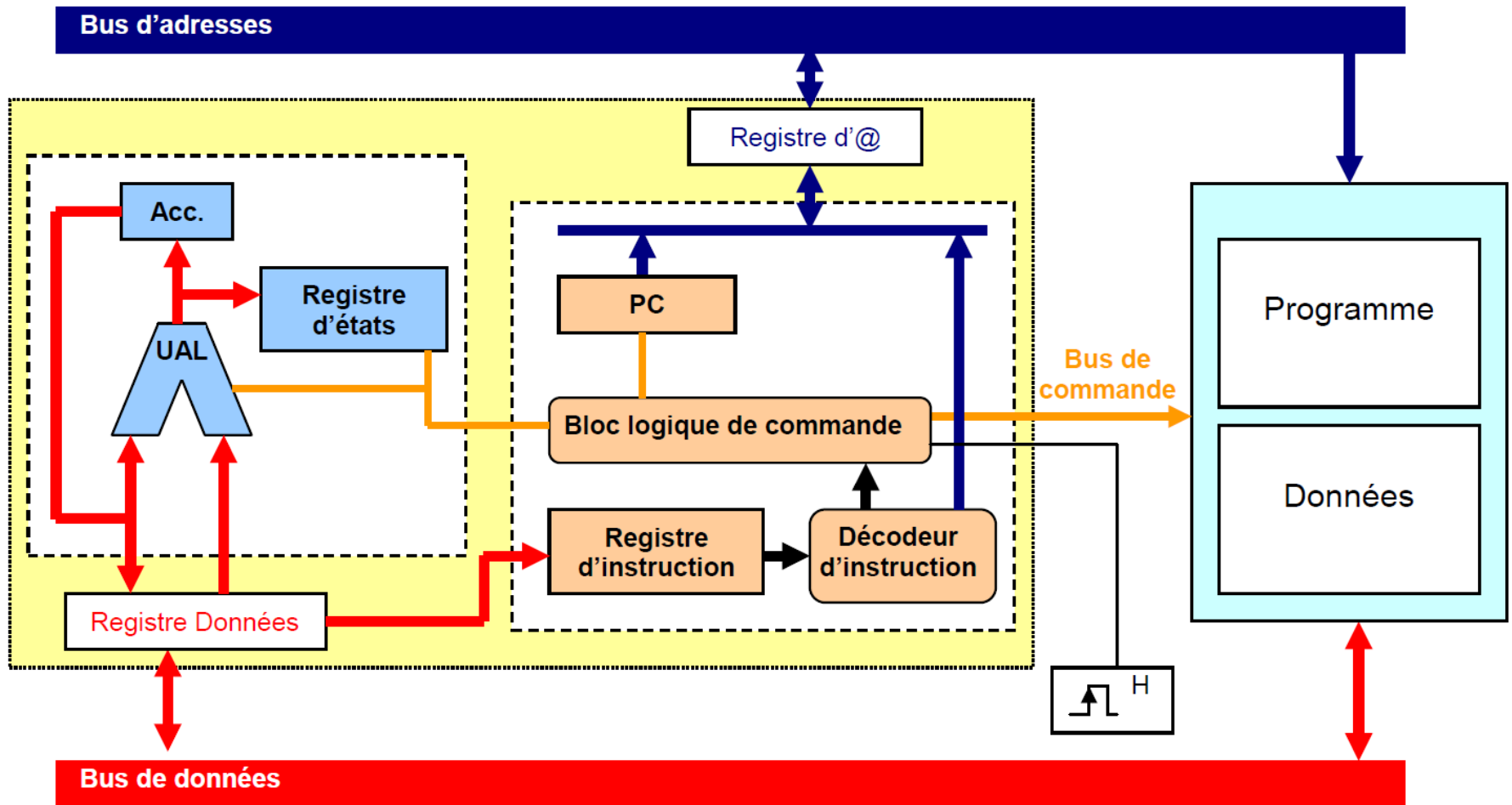


Un microprocesseur est construit autour de deux éléments principaux :

- Une unité de commande
- Une unité de traitement

associés à des registres chargés de stocker les différentes informations à traiter. Ces trois éléments sont reliés entre eux par des bus interne permettant les échanges d'informations.

Architecture détaillée d'un microprocesseur



L'unité de commande

Elle permet de séquencer le déroulement des instructions. Elle effectue la recherche en mémoire de l'instruction. Comme chaque instruction est codée sous forme binaire, elle en assure le décodage pour enfin réaliser son exécution puis effectue la préparation de l'instruction suivante. Pour cela, elle est composée par :

- **le compteur de programme** constitué par un registre dont le contenu est initialisé avec l'adresse de la première instruction du programme. Il contient toujours l'adresse de l'instruction à exécuter.
- **le registre d'instruction et le décodeur d'instruction** : chacune des instructions à exécuter est rangée dans le registre instruction puis est décodée par le décodeur d'instruction.
- **Bloc logique de commande (ou séquenceur)** : Il organise l'exécution des instructions au rythme d'une horloge. Il élabore tous les signaux de synchronisation internes ou externes (bus de commande) du microprocesseur en fonction des divers signaux de commande provenant du décodeur d'instruction ou du registre d'état par exemple.

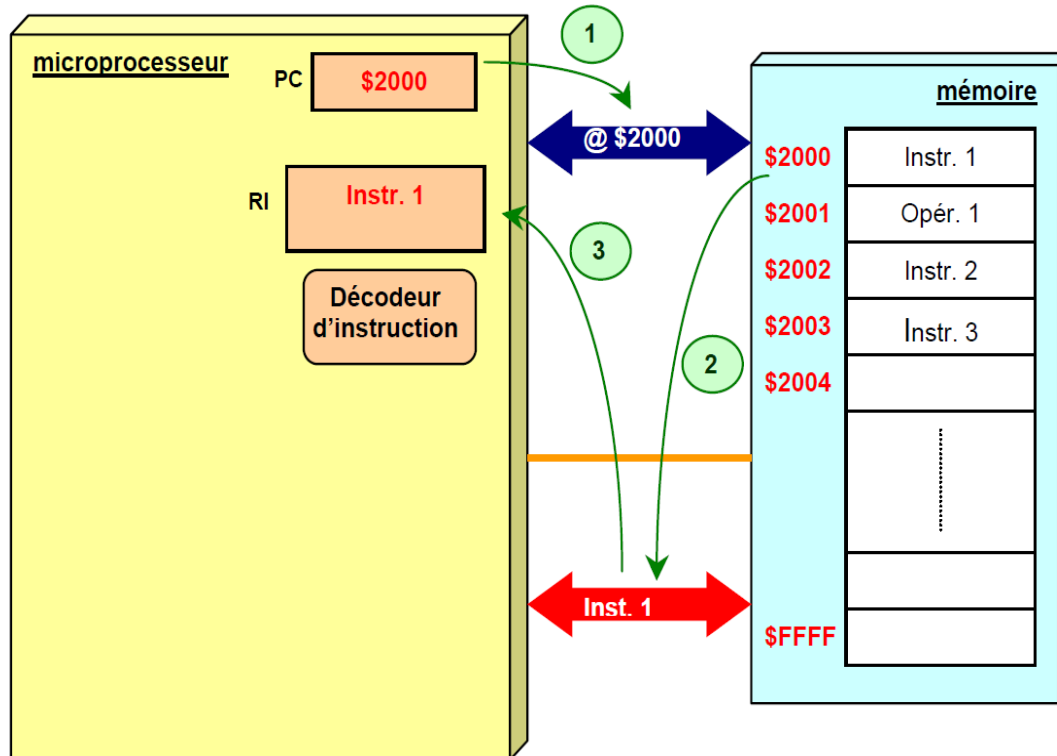
L'unité de traitement

C'est le cœur du microprocesseur. Elle regroupe les circuits qui assurent les traitements nécessaires à l'exécution des instructions :

- **L'Unité Arithmétique et Logique (UAL)** est un circuit complexe qui assure les fonctions logiques (ET, OU, Comparaison, Décalage , etc...) ou arithmétique (Addition, soustraction).
- **Le registre d'état** est généralement composé de 8 bits à considérer individuellement. Chacun de ces bits est un indicateur dont l'état dépend du résultat de la dernière opération effectuée par l'UAL. On les appelle *indicateur d'état* ou *flag* ou *drapeaux*. Dans un programme le résultat du test de leur état conditionne souvent le déroulement de la suite du programme. On peut citer par exemple les indicateurs de :
 - retenue (carry : C) signe (Sign : S)
 - débordement (overflow : OV ou V)
 - zéro (Z)
 - parité (Parity : P)
- **Les accumulateurs** sont des registres de travail qui servent à stocker une opérande au début d'une opération arithmétique et le résultat à la fin de l'opération.

Cycle d'exécution d'une instruction (phase 1)

Recherche de l'instruction à traiter



1. Le PC contient l'adresse de l'instruction suivante du programme. Cette valeur est placée sur le bus d'adresses par l'unité de commande qui émet un ordre de lecture.

2. Au bout d'un certain temps (temps d'accès à la mémoire), le contenu de la case mémoire sélectionnée est disponible sur le bus des données.

3. L'instruction est stockée dans le registre instruction du processeur.

Cycle d'exécution d'une instruction (phase 2)

Décodage de l'instruction et recherche de l'opérande

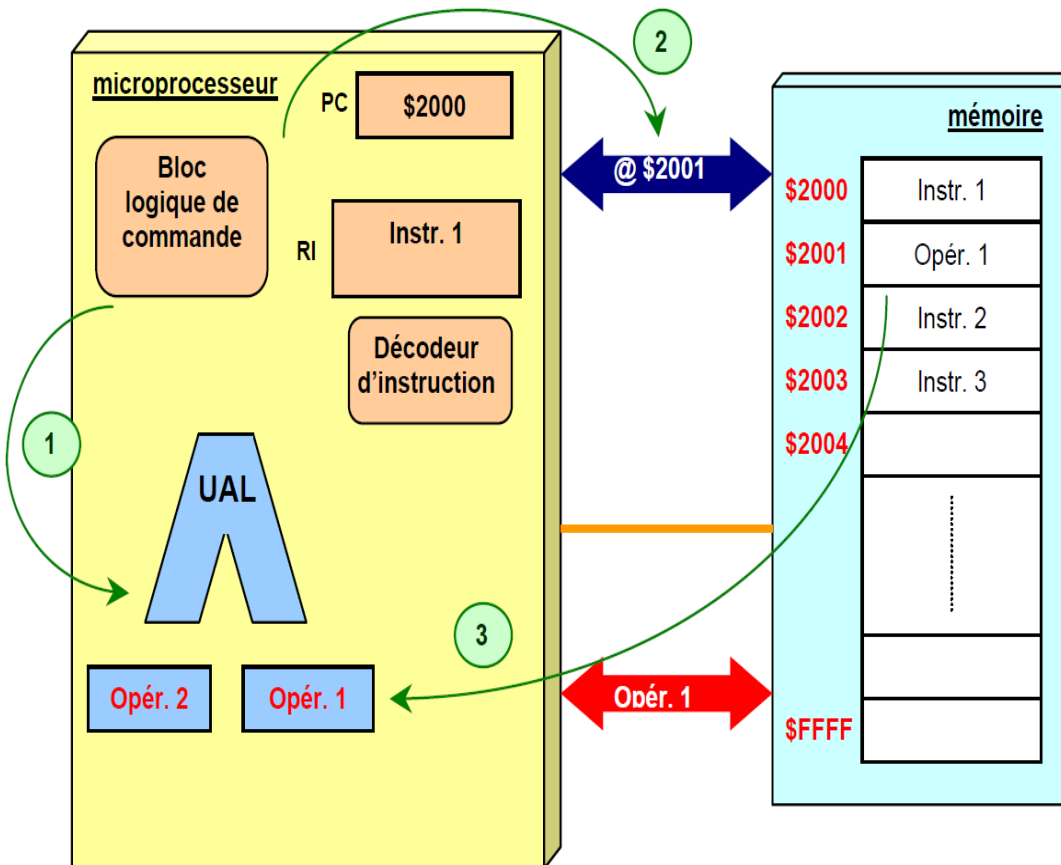
Le registre d'instruction contient maintenant le premier mot de l'instruction qui peut être codée sur plusieurs mots. Ce premier mot contient le code opératoire qui définit la nature de l'opération à effectuer (addition, rotation,...) et le nombre de mots de l'instruction.

1. L'unité de commande transforme l'instruction en une suite de commandes élémentaires nécessaires au traitement de l'instruction.

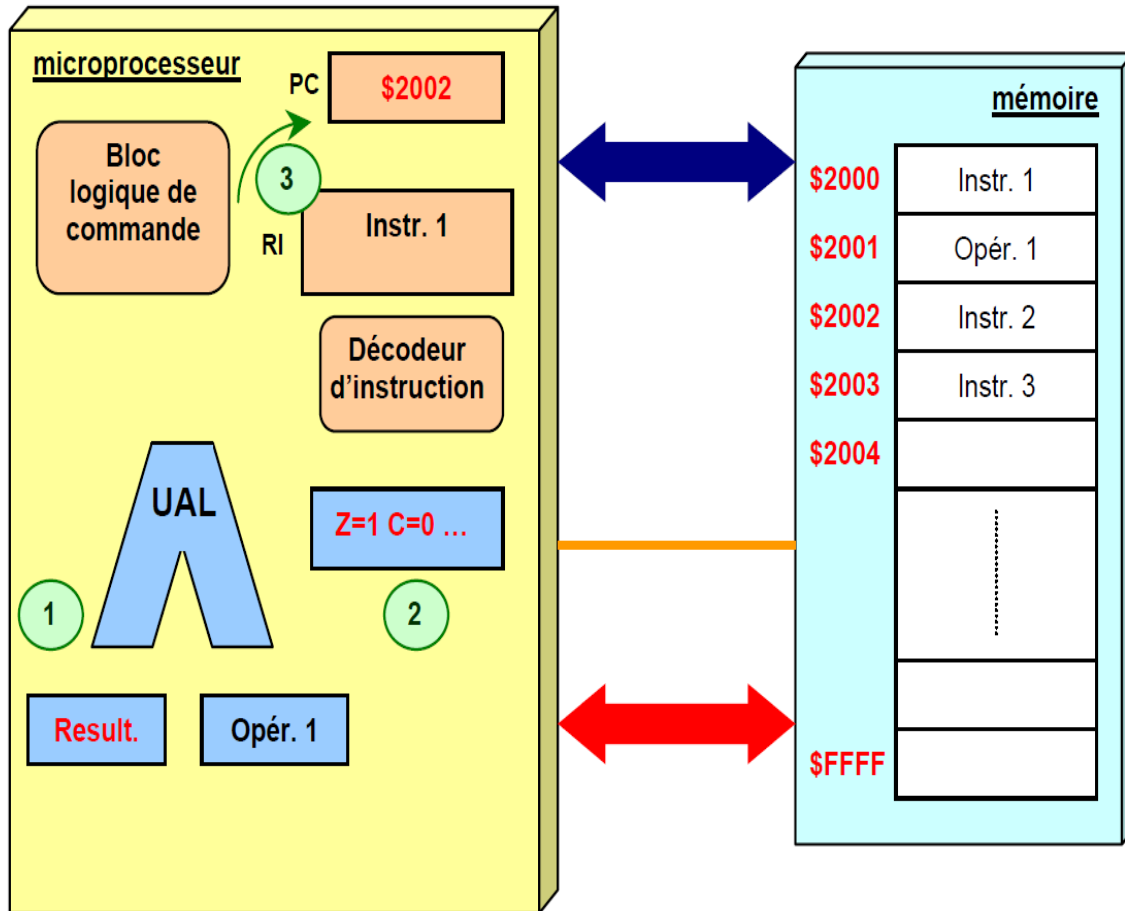
2. Si l'instruction nécessite une donnée en provenance de la mémoire, l'unité de commande

recupère sa valeur sur le bus de données.

3. L'opérande est stockée dans un registre.



Cycle d'exécution d'une instruction (phase 3)



Exécution de l'instruction

1. Le micro-programme réalisant l'instruction est exécuté.

2. Les drapeaux sont positionnés (*registre d'état*).

3. L'unité de commande positionne le PC pour l'instruction suivante.

Jeu d'instructions

- Il décrit l'ensemble des opérations élémentaires que le microprocesseur pourra exécuter.
- Les instructions que l'on retrouve dans chaque microprocesseur peuvent être classées en 4 groupes :
 - **Transfert de données** pour charger ou sauver en mémoire, effectuer des transferts de registre à registre, etc...
 - **Opérations arithmétiques** : addition, soustraction, division, multiplication
 - **Opérations logiques** : ET, OU, NON, NAND, comparaison, test, etc...
 - **Contrôle de séquence** : branchement, test, etc...
- Chaque instruction nécessite un certain nombre de cycles d'horloges pour s'effectuer : c'est le temps d'exécution. Il est dépendant de la complexité de l'instruction.

Jeu d'instructions : codage

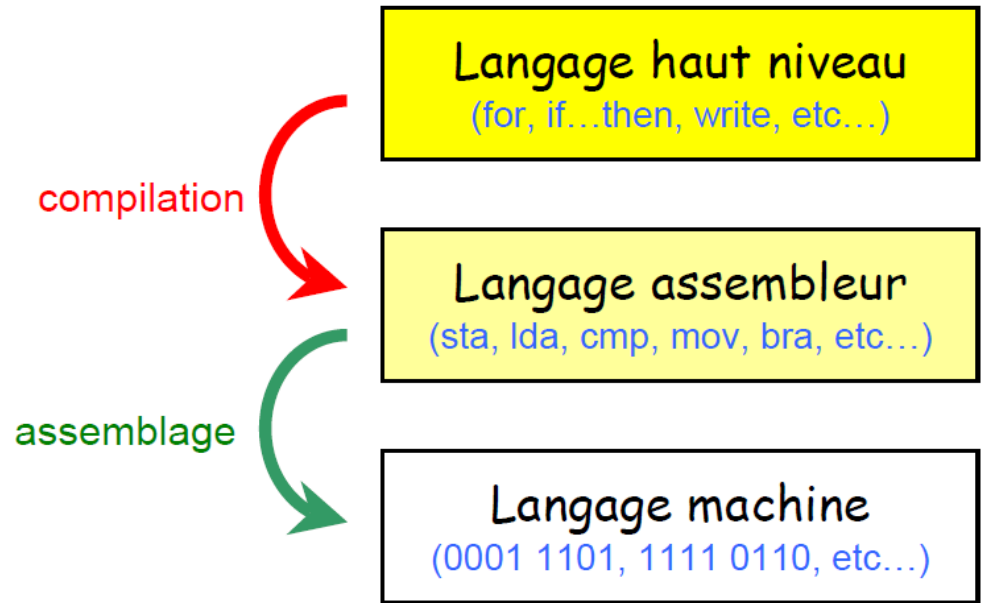
- Les instructions et leurs opérandes (paramètres) sont stockés en mémoire principale. La taille totale d'une instruction (nombre de bits nécessaires pour la représenter en mémoire) dépend du type d'instruction et aussi du type d'opérande. Une instruction est composée de deux champs :
 - **Le code instruction**, qui indique au processeur quelle instruction réaliser
 - **Le champ opérande** qui contient la donnée, ou la référence à une donnée en mémoire (son adresse).

Code instruction	Code opérande
1001 0011	0011 1110

Langage de programmation

Le langage **machine** est le langage compris par le microprocesseur.

Le langage **assembleur** est le langage le plus « proche » du langage machine. Il est composé par des instructions en général assez rudimentaires que l'on appelle des **mnémoniques**.



Code machine (68HC11)		Assembleur (68HC11)	Langage C
@00	C6 64	LDAB #100	A=0 ; for (i=1 ; i<101 ; i++) A=A+i ;
@01	B6 00	LDAA #0	
@03	1B	ret ABA	
@04	5A	DECB	
@05	26 03	BNE ret	

Améliorations de l'architecture de base

BUT : - Diminuer le temps d'exécution d'un programme.

COMMENT : - Augmenter la fréquence de l'horloge



- Augmentation de la température.

- Augmenter la taille du refroidisseur.

- Diminuer la tension d'alimentation.

- Diminuer le nombre moyen de cycles d'horloge nécessaire à l'exécution d'une instruction :

- Optimiser le compilateur dans un langage de haut niveau (voir diapositive précédente).

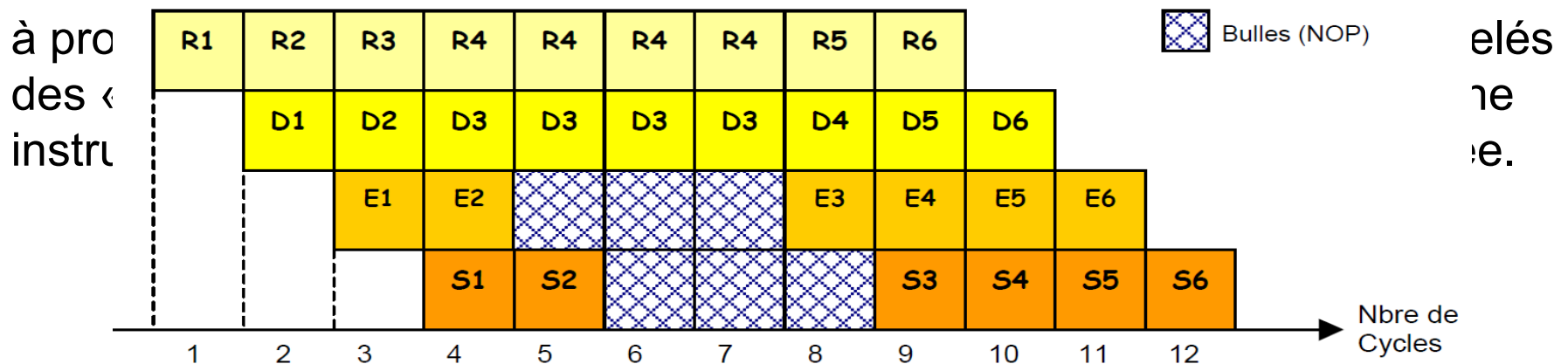
- Utiliser une architecture de microprocesseur qui réduise le nombre de cycles par instruction.

Problème du pipeline

Plus le pipeline est long, plus le nombre de cas où il n'est pas possible d'atteindre la performance maximale est élevé. Il existe 3 principaux cas, appelés **aléas**, où la performance d'un processeur pipeliné peut être dégradée :

- **aléa structurel** qui correspond au cas où deux instructions ont besoin d'utiliser la même ressource du processeur,
- **aléa de données** qui intervient lorsqu'une instruction produit un résultat et que l'instruction suivante utilise ce résultat avant qu'il n'ait pu être écrit dans un registre,
- **aléa de contrôle** qui se produit éventuellement chaque fois qu'une instruction de branchement est exécutée.

Lorsqu'un aléa se produit, cela signifie qu'une instruction ne peut continuer



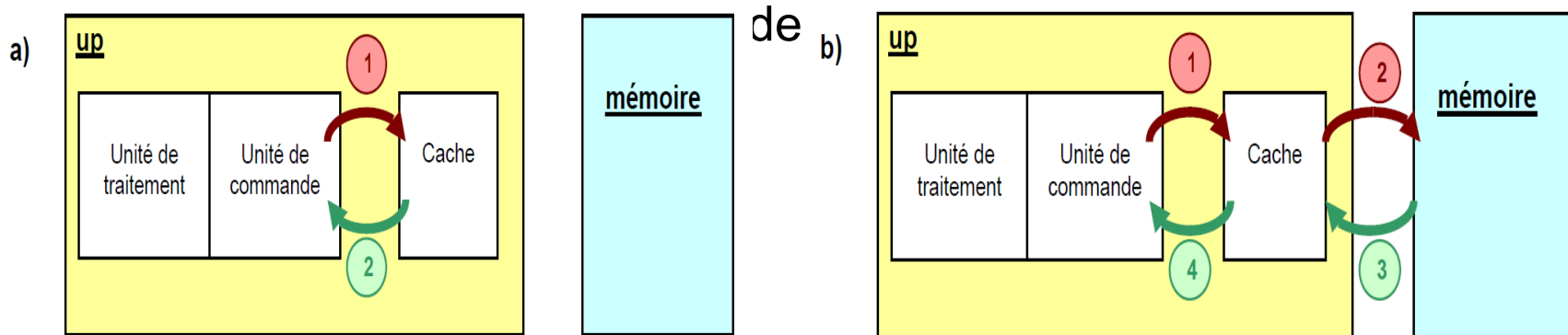
Notion de cache mémoire

le temps de cycle processeur décroît plus vite que le temps d'accès mémoire entraînant un goulot d'étranglement.

Le principe de cache est très simple : le microprocesseur n'a pas conscience de sa présence et lui envoie toutes ses requêtes comme s'il s'agissait de la mémoire principale :

Soit la donnée ou l'instruction requise est présente dans le cache et elle est alors envoyée directement au microprocesseur. On parle de **succès** de cache. (a)

soit la donnée ou l'instruction n'est pas dans le cache, et le contrôleur de cache envoie alors une requête à la mémoire principale. Une fois l'information récupérée, il la renvoie au microprocesseur tout en la



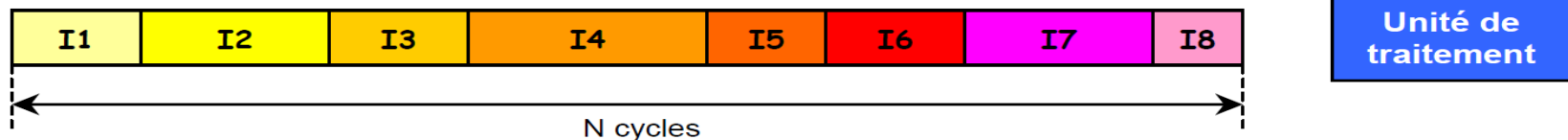
Architecture superscalaire

Une autre façon de gagner en performance est d'exécuter plusieurs instructions en même temps. L'approche superscalaire consiste à doter le microprocesseur de plusieurs unités de traitement travaillant en parallèle. Les instructions sont alors réparties entre les différentes unités d'exécution. Il faut donc pouvoir soutenir un flot important d'instructions et pour cela disposer d'un cache performant.

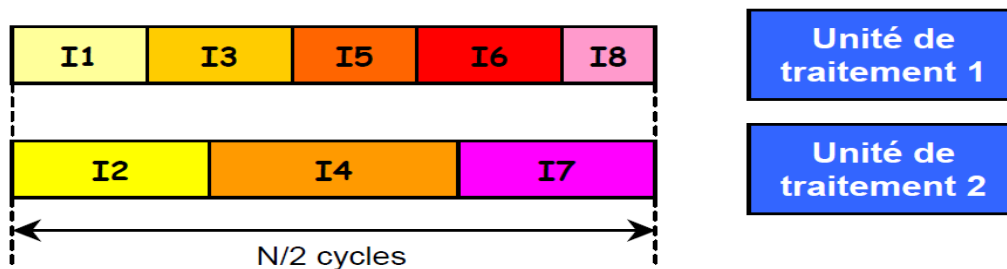
On distingue :

- La technologie **HyperThreading** : plusieurs unités logiques dans une

Architecture scalaire :

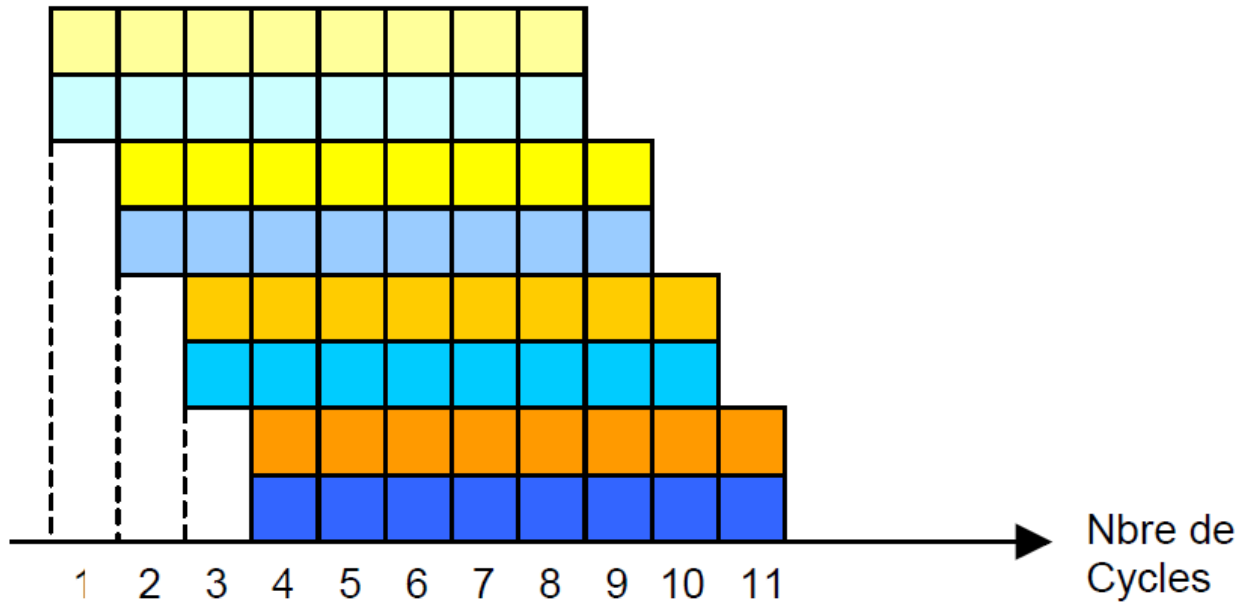
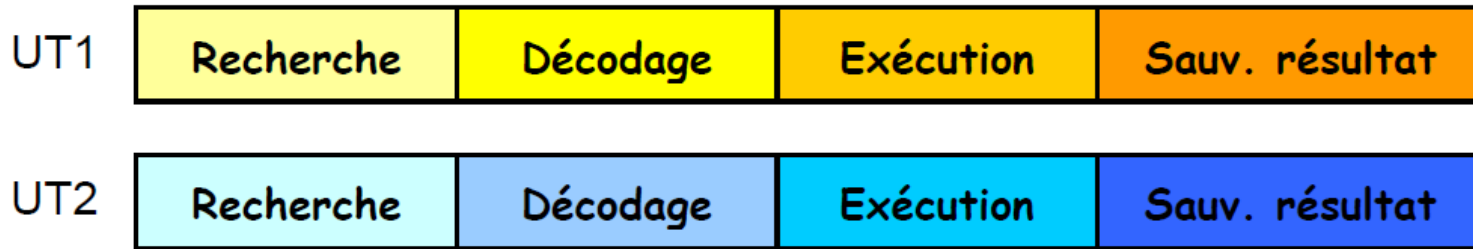


Architecture superscalaire :



superscalaire

Le principe est d'exécuter les instructions de façon pipelinée dans chacune des unités de traitement travaillant en parallèle.



EXERCICES

Mini assembleur utilisé

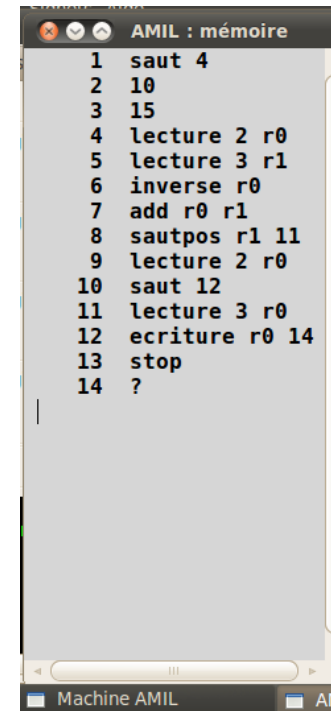
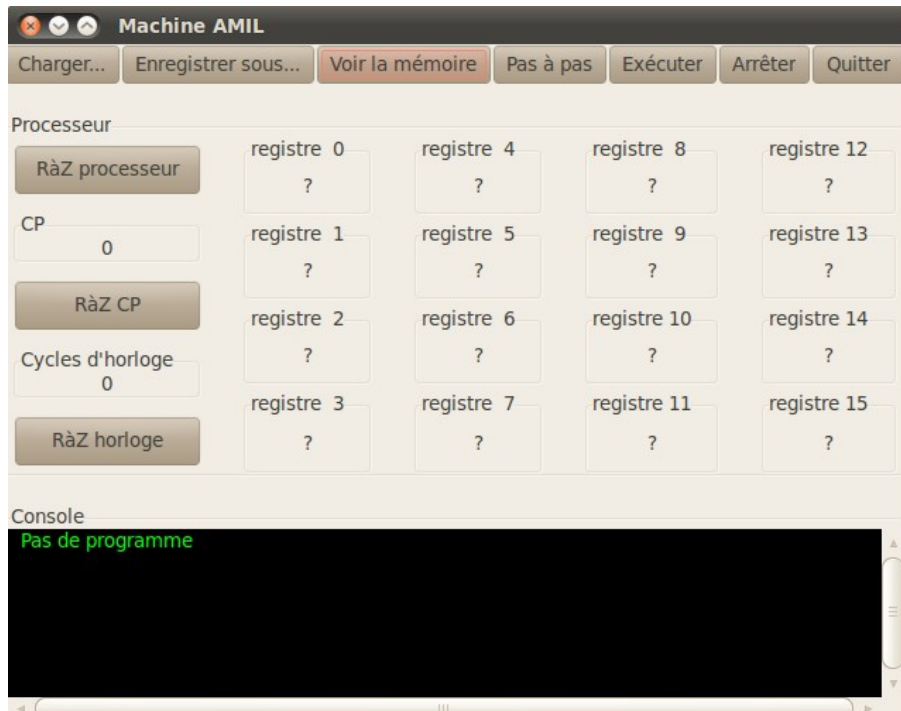
AMIL : <http://www-lipn.univ-paris13.fr/~boudes/spip.php?rubrique27>

Version WEB : <http://www-lipn.univ-paris13.fr/~boudes/amilweb/>

Version Améliorée : <http://www2.lifl.fr/~mailliet/isn/archi/amil/>

Version ZIP : <http://www2.lifl.fr/~mailliet/isn/archi/amil/amilweb.zip>

Version Linux utilisant GTK :



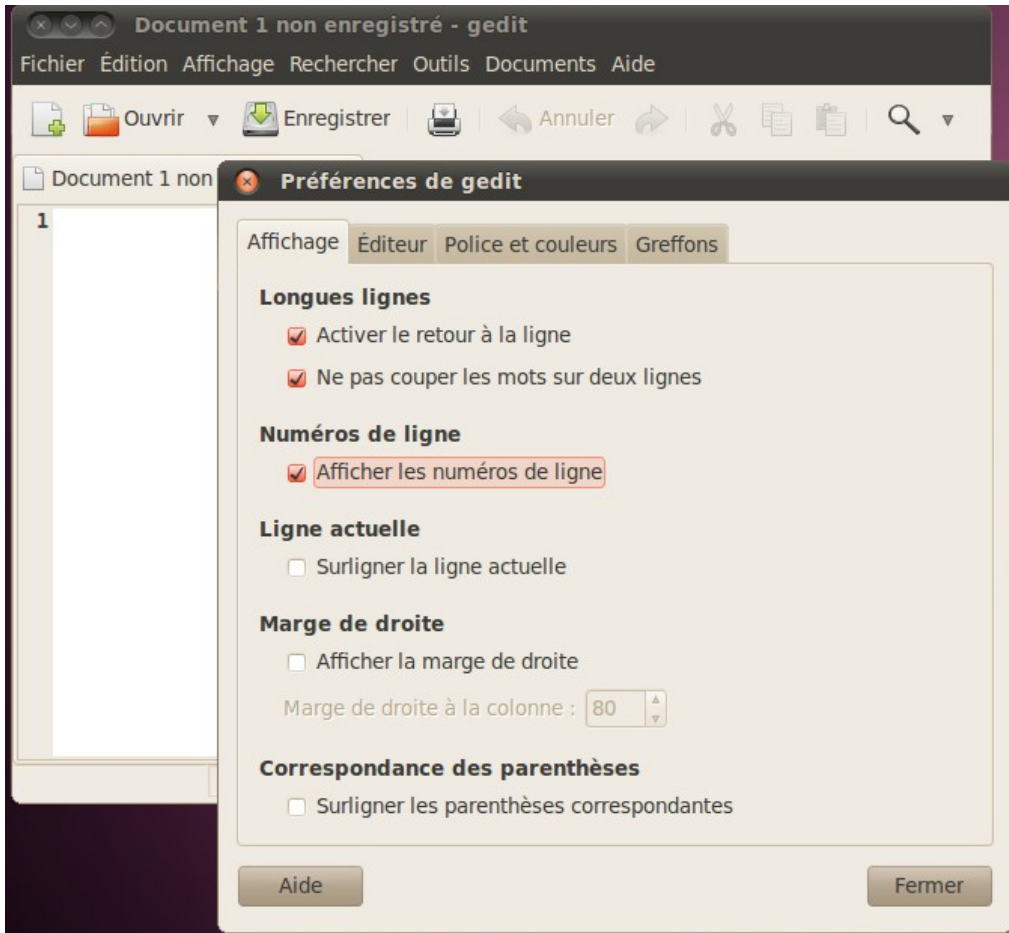
Jeu d'Instructions (initial)

Mnémonique	détail	action
stop	Arrête l'exécution du programme.	
noop	N'effectue aucune opération.	
saut i	Met le compteur ordinal à la valeur i.	$PC \leftarrow i$
sautpos ri j	Si la valeur contenue dans le registre i est positive ou nulle, met le compteur ordinal à la valeur j.	si $ri \geq 0$ $PC \leftarrow j$ sinon $PC \leftarrow PC+1$
valeur x ri	Initialise le registre i avec la valeur x.	$ri \leftarrow x$
lecture i rj	Charge, dans le registre j, le contenu de la mémoire d'adresse i.	$rj \leftarrow \text{men}(i)$
lecture *ri rj	Charge, dans le registre j, le contenu de la mémoire dont l'adresse est la valeur du registre i.	$rj \leftarrow \text{men}(ri)$
écriture ri j	Écrit le contenu du registre i dans la mémoire d'adresse j.	$ri \rightarrow \text{men}(j)$
écriture ri *rj	Écrit le contenu du registre i dans la mémoire dont l'adresse est la valeur du registre j.	$ri \rightarrow \text{men}(rj)$
inverse ri	Inverse le signe du contenu du registre i	$ri \leftarrow -ri$
add x rj	Ajoute x au contenu du registre j.	$ri \leftarrow ri + x$
add ri rj	Ajoute la valeur du registre i à celle du registre j.	$rj \leftarrow rj + ri$
soustr, mult, div, et	Même syntaxe que pour add mais pour la soustraction, multiplication, la division entière, le et bit à bit.	$rj \leftarrow rj$ (-, *, /, and) ri ou $ri \leftarrow ri$ (*, /, and) x

Jeu d'Instructions (ajout)

Mnémonique	détail	action
lecture ri rj	Écrit le contenu du registre i dans le registre j.	$rj \leftarrow ri$
sautnul ri j	Si la valeur contenue dans le registre i est nulle, met le compteur ordinal à la valeur j.	si $ri = 0$ $PC \leftarrow j$ sinon $PC \leftarrow PC+1$
sautnonnul ri j	Si la valeur contenue dans le registre i est non nulle, met le compteur ordinal à la valeur j.	si $ri \neq 0$ $PC \leftarrow j$ sinon $PC \leftarrow PC+1$
appel i	Appel de sous-programme à l'adresse i	$PC \leftarrow i$ (l'adresse de retour est empilée)
retour	retour de procédure à l'appelant.	$PC \leftarrow$ (haut de pile)
empiler ri	Place la valeur contenue dans le registre i en haut de la pile (la même pile que pour les adresses).	$ri \rightarrow \text{Haut_de_Pile}$
depiler rj	Place la valeur en haut de la pile (la même pile que pour les adresses) dans le registre rj.	$rj \leftarrow \text{Haut_de_Pile}$

Convention d'écriture



Pour écrire vos programmes, utiliser **GEDIT** (ou Notepad++ sous Windows).

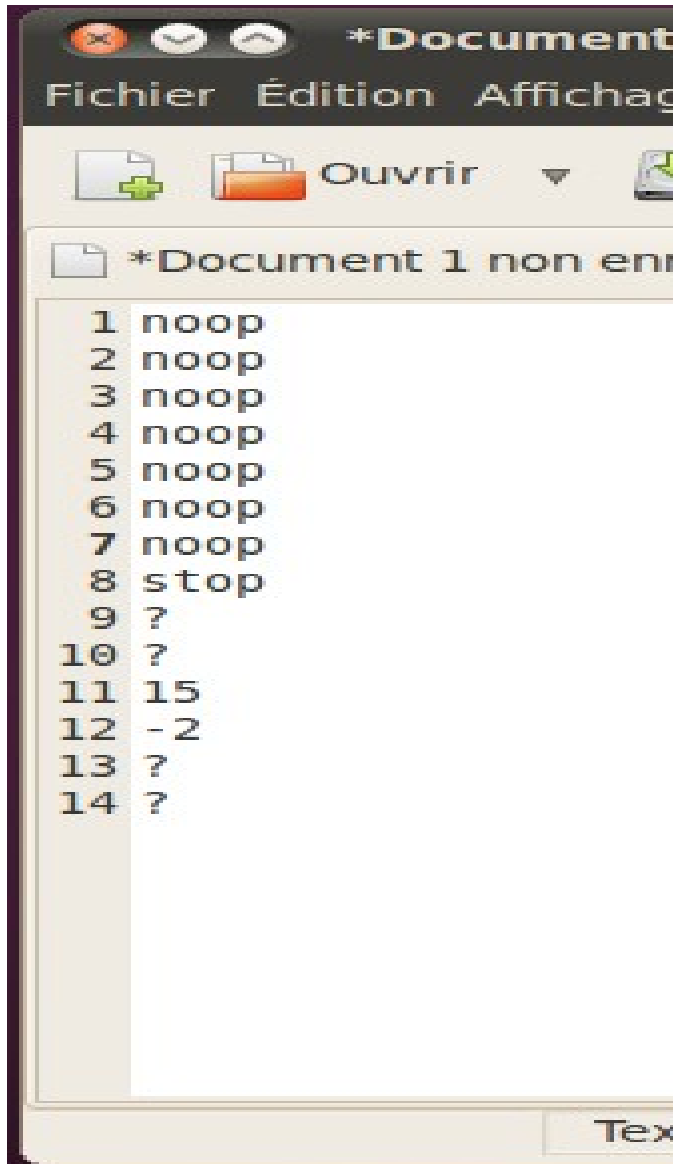
Dans les préférences de GEDIT cocher 'Afficher les numéros de ligne'.

Chaque numéro de ligne correspond à une adresse mémoire (une instruction avec ses données).

Chaque programme se termine par l'instruction '**stop**'.

Les instructions d'un programme sont consécutives (jusqu'au '**stop**').

Convention d'écriture



```
1 noop
2 noop
3 noop
4 noop
5 noop
6 noop
7 noop
8 stop
9 ?
10 ?
11 15
12 -2
13 ?
14 ?
```

Ce programme (fort intéressant) **est composé :**

- De 14 lignes
- Les lignes de 1 à 8 contiennent les instructions du programme.
- Les lignes de 9 à 14 sont des cases mémoires que vous pouvez utiliser et modifier.
- Par convention les lignes comprenant un nombre (11 et 12) sont en entrée (saisie du clavier, mais rien n'empêche de les modifier), les lignes comprenant un ? (9,10,13,14) sont en sortie (moniteur).

exercices

Les exos de 1 à 4 sont à faire avec le jeu d'instructions initial d'AMIL

Exercice 1 :

Ecrire un programme qui lit un nombre en entrée et le restitue en sortie.

Exercice 2 :

Lire dans l'ordre la valeur de 'a', 'b' et 'x', et rendre ensuite le résultat $y = ax + b$.

Exercice 3 (simulation du if) :

Ecrire 3 (x) à l'emplacement mémoire 12, lire un nombre placé en mémoire 10, si ce nombre est plus grand ou égal à 3 (x) l'écrire à l'emplacement mémoire où le 3 (x) est écrit.

Exercice 4 (if / else) :

Ecrire le même programme mais qui répond 0 (faux) ou 1 (vrai) à la condition. Tester ce pg avec la valeur 3, cela répond-t-il à l'exigence de l'exo3 ? Corriger le problème en utilisant les instructions étendues d'AMIL.

Exercice 5 : (Tant Que, tableau)

Ecrire un programme qui, lit l'adresse i du premier indice d'un tableau (i est donc donné) de longueur quelconque, qui contient une suite de nombre positif et qui se termine par un nombre négatif, chaque valeur de $\text{mem}(x)$ étant recopiée en sortie (sauf le nb négatif).

```
I = 20
P
R
O
G
Stop
L20 : 12
L21 : 5
L22 : 15
L23 : 6
L24 : 85
L25 : -1
```



```
I = 20
P
R
O
G
Stop
L20 : 12
L21 : 5
L22 : 15
L23 : 6
L24 : 85
L25 : -1
L30 : 12
L31 : 5
L32 : 15
L33 : 6
L34 : 85
```


exercices

Exercice 6 : (pour, tableau)

Ecrire le même programme mais cette fois l'indice de fin est connu (le -1 n'a plus lieu d'exister)

```
I = 20
j = 24
P
R
O
G
Stop

L20 : 12
L21 : 5
L22 : 15
L23 : 6
L24 : 85
```



```
I = 20
J = 24
P
R
O
G
Stop

L20 : 12
L21 : 5
L22 : 15
L23 : 6
L24 : 85

L30 : 12
L31 : 5
L32 : 15
L33 : 6
L34 : 85
```

Rappel : réalisation d'un 'pour' à l'aide d'un 'Tant Que' :
for (i = 1 ; i < 9 ; i++) { suite instructions } peut être remplacé par :

```
i = 1
TQ ( i < 9 ) {
    la même suite d'instructions
    i++
}
```

Exercice 7 :

mettre 2 valeurs dans r0 et r1 et échanger leur valeurs en se servant de r2 , puis en se servant de la mémoire au lieu de r2, puis de la pile.

Echanger 2 valeurs en mémoire sans transférer ces valeurs dans les registres (r0 et r1) mais grâce à leurs adresses.

Refaire les mêmes exercices, mais n'échangez que si la première valeur est supérieure à la seconde.

Exercice 8 : (tri à bulles)

Ecrire un programme de tri à bulles effectuant le résultat suivant :

Rappel (ou pas) ; l'algorithme du tri à bulles est le suivant.

PROCEDURE Tri_bulle (Tableau a[1:n])

Booleen permut = VRAI;

TANT QUE permut = VRAI

permut = FAUX

POUR i VARIANT DE 1 à N-1 FAIRE

SI a[i] > a[i+1] ALORS

échanger a[i] et a[i+1]

permut = VRAI

FIN SI

FIN POUR

FIN TANT QUE

FIN PROCEDURE

```
L20 : 12
L21 : 5
L22 : 15
L23 : 6
L24 : 85
```



```
L30 : 5
L31 : 6
L32 : 12
L33 : 15
L34 : 85
```

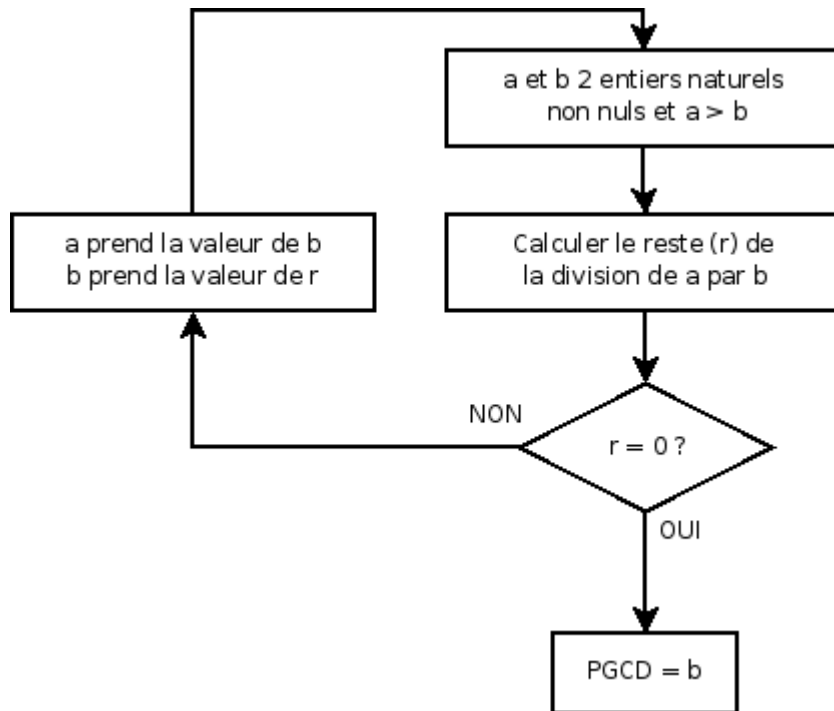
exercices

Exercice 9 :

Ecrire un programme lisant deux nombres a et b , remplaçant a par b dans le cas où $a < b$ et retournant a , b et $a \bmod b$ (reste de la division entière de a par b).

Exercice 10 : PGCD de 2 nbs

Ecrire un programme qui retourne le PGCD de 2 nombres (Algorithme d'Euclide).



Exercice 11 :

Sachant que $\text{PGCD}(a,b) \cdot \text{PPCM}(a,b) = a \cdot b$, écrire un programme (ou ajouter au pg précédent) qui rend le PPCM de 2 nbs a et b .

Exercice 12 : sous-programmes

Ecrire un programme comprenant un sous programme capable de retourner $y = 2x + 3$. Essayez-le avec au moins 3 valeurs de x que vous lirez en intercalant une série aléatoire de « noop ».

Exercice 13 : factorielle

Ecrire un programme calculant $n!$ (n étant donné en mémoire) par la méthode récursive.

exercices

Exercice 14 :

Ecrire un programme qui partant de trois données entrées en mémoire (v, d, l) va mettre v en mémoire à partir de l'adresse d et ce pour l adresses consécutives.

Exercice 15 :

Ecrire un programme qui partant de trois données entrées en mémoire (v, d, l) va mettre 1 en mémoire à partir de l'adresse d et ce pour l adresses consécutives. mettre 0 dans les adresses en $d+2v, d+3v \dots d+iv$.

Exercice 16 :

En se servant des 2 exercices précédents, programmer le crible d'Eratostène.

[http://www.fil.univ-lille1.fr/~](http://www.fil.univ-lille1.fr/~wegrzyno/portail/API1/Doc/TP/TP-Tableaux/tp-tableaux002)

[wegrzyno/portail/API1/Doc/TP/TP-Tableaux/tp-tableaux002](http://www.fil.univ-lille1.fr/~wegrzyno/portail/API1/Doc/TP/TP-Tableaux/tp-tableaux002)

[http://www.fil.univ-lille1.fr/~](http://www.fil.univ-lille1.fr/~wegrzyno/portail/API1/Doc/TP/TP-Tableaux/New_Animation)

[wegrzyno/portail/API1/Doc/TP/TP-Tableaux/New_Animation](http://www.fil.univ-lille1.fr/~wegrzyno/portail/API1/Doc/TP/TP-Tableaux/New_Animation)
(pour l'animation)

Le crible d'Eratosthene

Eratosthene , mathématicien grec du IIIème siècle avant JC, a établi une méthode connue sous le nom de crible d'Eratosthene permettant de déterminer par exclusion tous les nombres premiers.

Cette méthode consiste à lister tous les nombres entiers depuis 2 jusqu'à une valeur limite n que l'on se fixe, puis à barrer successivement ces nombres

L'algorithme procède par élimination : il s'agit de supprimer d'une table tous les multiples des entiers de 2 à n, n étant un entier que l'on se fixe.

On commence par les multiples de 2, puis à chaque fois on raye les multiples du plus petit entier restant jusqu'à ce que le carré de celui-ci soit supérieur au plus grand entier de la liste.

On peut s'arrêter lorsque le carré du plus petit entier est supérieur au plus grand entier, car dans ce cas, s'il existait des non-premiers, ils auraient déjà été rayés précédemment.

À la fin du processus, tous les entiers qui n'ont pas été rayés sont les nombres premiers inférieurs à n.