A.O. Partie 2: Architecture de Von Neuman (UCT/CPU, bus, mémoires, …)

http://www.phmartin.info/cours/ao/ (→ TDs, QCMs, corrigés, ...)

1. Composants de haut-niveau (architecture de Von Neuman)

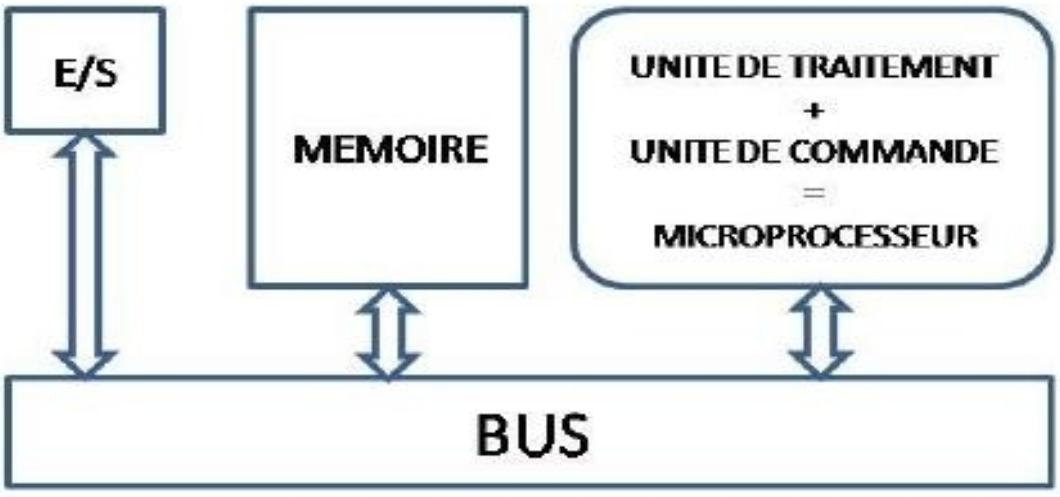
Ordinateur: unité centrale + unités d'entrées/sorties.

Micro-ordinateur: ordinateur dont l'unité centrale est sur une micro-plaquette (puce).

Unités d'entrées/sorties (E/S) [Input/Output (I/O)]: transfèrent des informations entre l'unité centrale et les unités périphériques qui sont beaucoup plus lentes; e.g., le(s) DMA(s) [Direct Memory Access processor(s)].

Unités périphériques: mémoires "permanentes" (alias "de masse": disques, bandes, ...) et autres unités (écran, clavier, imprimante, modem, ...);
à chaque catégorie d'unités périphériques est associé un contrôleur de périphériques qui gère ces unités.

Unité centrale (la partie la plus importante d'un ordinateur):
unité centrale de traitement (UCT/CPU) + mémoire centrale (alias "principale").



1. Composants de haut-niveau (architecture de Von Neuman)

Unité centrale (la partie la plus importante d'un ordinateur):
unité centrale de traitement (UCT/CPU) + mémoire centrale (alias "principale").

UCT ou CPU [Central Processing Unit (CPU)] ou **processeur**:
unité de commande/contrôle (UC) + unité de traitement/calcul (UT ou UAL);
exécute les instructions d'un programme.

UC: dirige le fonctionnement de l'UAL, de la mémoire et des E/S, e.g., va chercher, une par une, des instructions en mémoire (et les données qu'elles utilisent), décode chaque instruction, et envoie un signal à l'UAL pour déclencher l'exécution de l'instruction.

UT ou UAL: unité arithmétique et logique; effectue les opérations des programmes et donc aussi des entrées-sorties de données avec la mémoire.

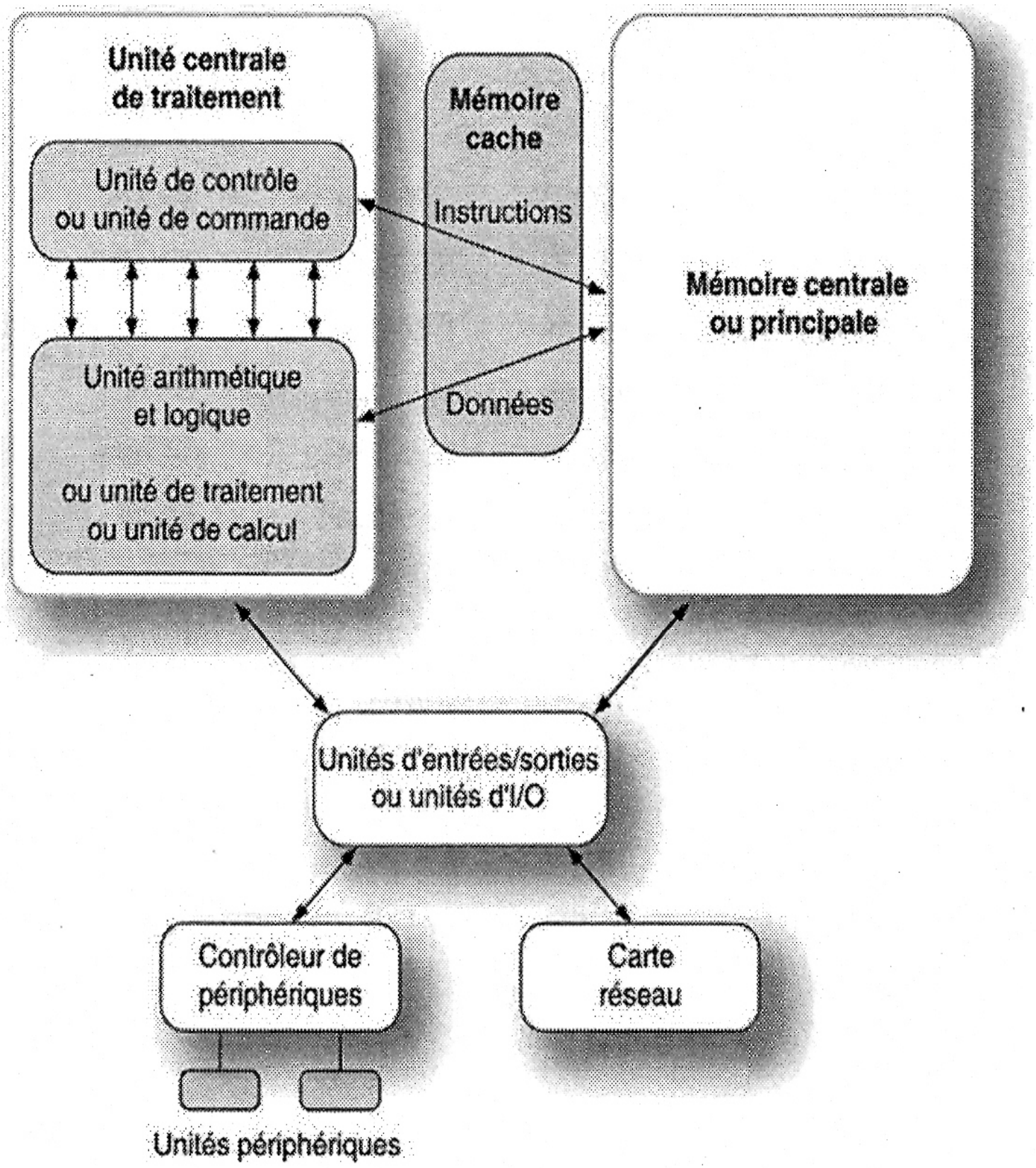
Mémoire (sous entendu "non permanente et externe à l'UCT/CPU") [memory]:
mémoire cache (alias "**antémémoire**"; quelques Mo -> très rapide d'accès)
+ mémoire centrale (alias "**principale**").

Mémoire centrale ou principale (MP [main memory]):
longue suite séquentielle de "mots mémoire" ayant chacun une adresse;
c'est une mémoire à semi-conducteurs;
elle contient une partie du système d'exploitation de l'ordinateur;
lorsqu'un programme s'exécute, tout ou partie du programme et des données y sont chargés.

Une mémoire cache stocke les données les plus récemment accédées.
Elle peut être entre l'UCT/CPU et la MP, entre l'UCT/CPU et un disque, entre l'UCT/CPU et un autre cache, … Quelques Mo -> rapide mais peu de données.

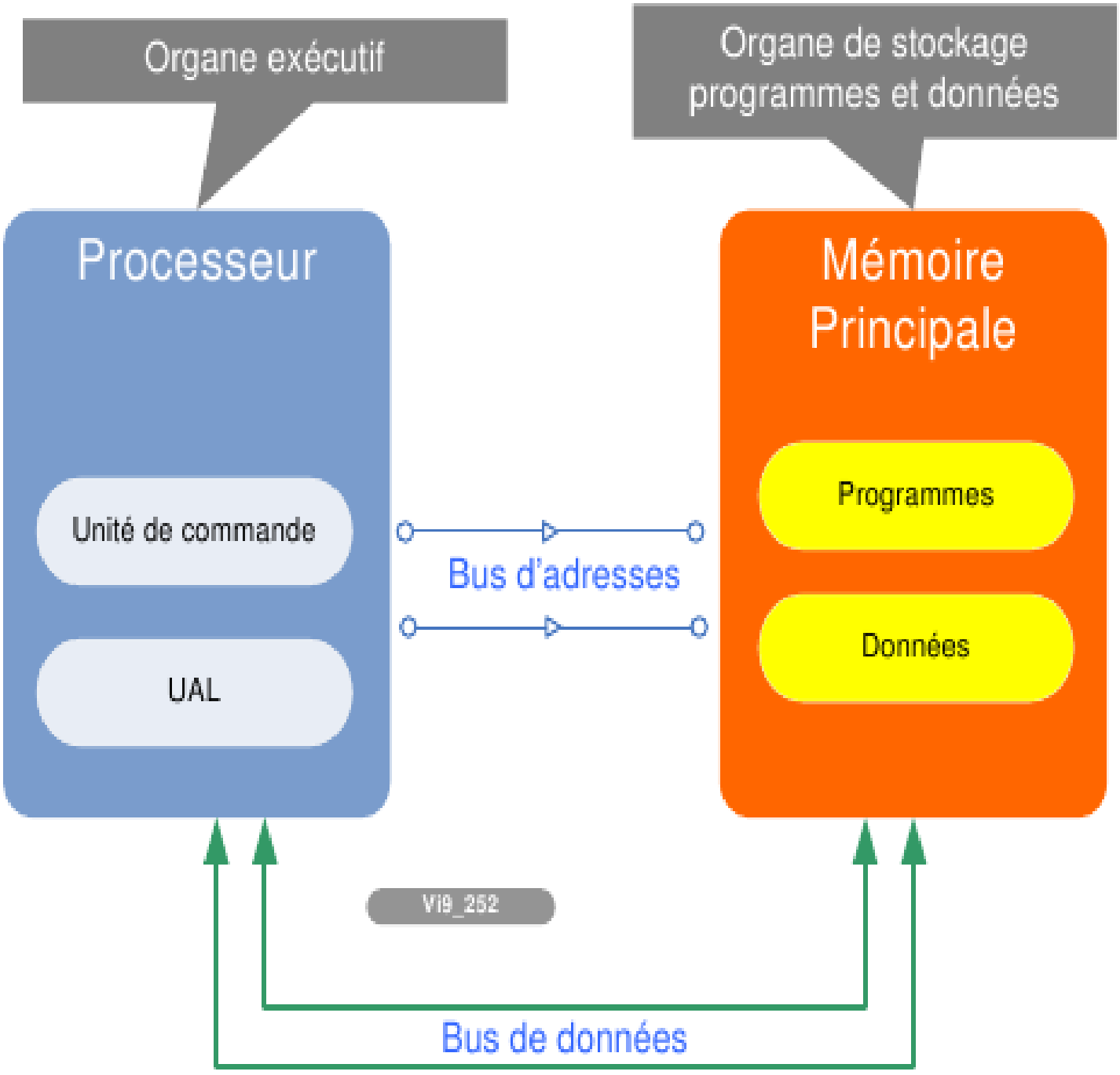
QW *(allez sur "ao**2**wooclap" sur la page Moodle du cours*
pour une ou deux questions d'évaluation relative à cette page).

1. Composants de haut-niveau (architecture de Von Neuman)



1. Composants de haut-niveau (architecture de Von Neuman)

Exemple encore plus simple de vue interne et fonctionnelle de l'unité centrale:



2. Mémoires - définitions

Registres: mémoire ***interne à l'UCT/CPU*** (donc pas dans la mémoire cache ou la MP mais, comme elles, un registre est une mémoire non permanente).

Registre: registre mot ou registre adresse.

Registre mot: contient le contenu d'un mot (unité d'information adressable);
a la taille d'un mot (longueur classique actuelle: 32 ou 64 bits).

Registre adresse: contient l'adresse d'un mot;
longueur: $\log_2$(Nombre_de_mots_en_MP).

Rappel:
Mémoire (sous entendu "non permanente et ***externe à l'UCT/CPU"***) [memory]:
mémoire cache (alias "**antémémoire**"; quelques Mo -> très rapide d'accès)
\+ **mémoire centrale** (alias "**principale**"; définition déjà donnée).

Mémoire de masse (alias **permanente**):
mémoire "primaire" (alias, "de stockage"; -> disques durs) *ou* "secondaire".

Mémoire secondaire (alias "auxiliaire" ou "périphérique"): pour archivage à long terme -> supports magnétiques (disques, cartouches, bandes), supports magnéto-optiques (disques) ou optiques (CD-ROM, CD-RW, DVD).

Stockage en/hors ligne: "en ligne" [on-line] (quasi-instantané -> disques durs), "(via un) réseau", "quasi en ligne" [near line] (via un jukebox et un robot), "hors-ligne" [off-line] (sur des étagères; accès manuel).

Stockage en réseau SAN [Storage Area Network]: stockage sur disques via un réseau rapide spécifique reliant des disques à des serveurs de fichiers.

Stockage en réseau NAS [Network Attached Storage]: stockage sur disques via un serveur de fichiers accessible via un réseau local.

2. Mémoires - définitions

Mot mémoire [word]: ensemble de bits pouvant être lus ou écrits simultanément. C'est donc aussi "l'unité d'information adressable".
Sa longueur classique actuelle: 32 ou 64 bits.
La MP comprend de nombreuses puces (éléments de mémoire) réparties sur des cartes mémoires de différentes manières:
- 1 bit/carte: un mot-mémoire est composé par les bits ayant la même adresse sur les cartes; l'adresse se compose du numéro de la puce et de l'adresse à l'intérieur de la puce;
- 1 bit/puce: un mot-mémoire est composé par les bits ayant la même adresse dans les puces;
- plusieurs bits/puce: un mot-mémoire est composé par plusieurs bits d'une même puce.

Ces différentes structures n'influent ni sur la capacité de la MP ni sur la longueur des adresses.

Capacité (taille) d'une mémoire: nombre de bits, d'octets [bytes] (1 octet = 8 bits) ou de mots qu'elle peut contenir, e.g., 300 Go pour certains disques magnétiques et 512 Mmots de 64 bits pour certaines MPs.

Temps d'accès mémoire: temps pour la lecture/écriture d'un mot mémoire; quelques nanosecondes à centaines de nanosecondes.

(Temps de) Cycle mémoire: temps minimal entre 2 accès mémoire; temps d'accès + temps des opérations de maintien, stabilisation, synchronisation, etc.

Débit: nombre d'informations lues ou écrites par seconde, e.g., 12 Mo/s pour une cartouche magnétique.

Temps d'accès registre: environ 10 fois plus court que le temps d'accès mémoire -> nécessité d'une mémoire entrelacée (voir page suivante), d'une mémoire cache ou de nombreux registres.

2. Mémoires - définitions

Mémoire entrelacée: divisée en blocs possédant chacun son propre registre d'adresse et son registre mot-mémoire; l'**entrelacement** consiste à placer les mots se trouvant à des adresses successives dans des blocs différents; ceci permet à l'UCT/CPU de lancer successivement des opérations d'accès à des blocs différents sans attendre la fin des transferts → parallélisme

Mémoire à accès aléatoire [Random Access Memory (RAM)]:
le temps d'accès y est identique pour chaque mot de la MP
car chaque mot mémoire est associé à une adresse unique. E.g.:
0 → "bleu" (à l'adresse 0 se trouve la chaîne de caractères "bleu"), 1 → "jaune"
=> temps d'accès constant.

Mémoire associative (e.g., mémoire cache): mémoire adressable
"par le contenu", i.e., via une clé et non via un index numérique. E.g.:
"mer" → "bleu" (à la clé "mer" est associé "bleu"), "soleil" → "jaune"
=> temps d'accès constant.
Note: cet accès par un index (-> utilisation d'une table de hachage ou d'une autre structure de donnés) est parfois décrit comme permettant "une recherche en parallèle sur toutes les cases de la mémoire associative" mais il n'y a pas d'accès en parallèle au même sens que dans une mémoire entrelacée.

Quelques types d'accès à la mémoire:
* "par le contenu" (pour les mémoires associatives, e.g., les mémoires cache)
* "aléatoire" (e.g., pour la MP): via une adresse
* "direct" (pour les disques, CDs, ...):
  accès à un bloc de données (contenant celle recherchée) via son adresse
  puis déplacement séquentiel jusqu'à la donnée recherchée
  → temps d'accès variable
* "séquentiel" (e.g., pour bandes magnétiques, fichier séquentiel)
  - lecture par déplacement séquentiel jusqu'à la donnée recherchée
  - écriture en fin de bande/fichier
* "semi-séquentiel" (e.g., pour un disque magnétique, l'accès au cylindre est direct et l'accès au secteur est séquentiel).

2. Mémoires - définitions

Mémoire vive [Read Write Memory]: pouvant être lue et (souvent) écrite.
Inverse: **Mémoire ~morte (ROM [Read Only Memory] or Read-mostly)**:
pouvant être lue mais *pas (ou peu de fois) écrite;*
elle est donc essentiellement programmée par le fabricant;
une MP peut contenir un peu de ROM pour stocker le noyau d'un système d'exploitation; le reste est mémoire vive et volatile (RAM).

Mémoire volatile (inverse: permanente): perd son contenu lorsque l'on coupe le courant, e.g., une mémoire à semi-conducteurs (sans pile associée). En pratique, c'est aussi une mémoire vive (sinon à quoi servirait elle ?).

Mémoire dynamique: mémoire volatile qui en plus doit être rafraîchie périodiquement, e.g., environ 100 fois par seconde car le condensateur se décharge; avantages: compactes et pas chères; utilisée dans les MPs).

Mémoire statique: mémoire volatile non dynamique, e.g., utilisation de 4 transistors pour stocker un bit; rapides (temps d'accès de quelques ns) mais chères et de faible capacité (quelques Mo) → pour mémoires cache.

Mémoire RAM [Random Access Memory (RAM)]
(mémoire volatile; statique/dynamique):

* **Mémoire RAM dynamique : DRAM**
* **Mémoire RAM statique**
  - **SRAM**
  - **MRAM** [Magnetic RAM]

Flash (basée sur une technologie EEPROM rapide mais dont le temps d'écriture est similaire à celui d'un disque dur) :
- SSD (disque statique à semi-conducteurs) [*solid-state drive] :* utilise des portes NAND flash ou bien, anciennement, de la RAM dynamique sans condensateur (note : pour les QCMs, toutes les RAM dynamiques sont supposées utiliser des condensateurs) ;
- SecureDigital (SD), MemoryStick (MS), CompactFlash (CF), … utilisée dans les appareils photos, téléphones, …

QW

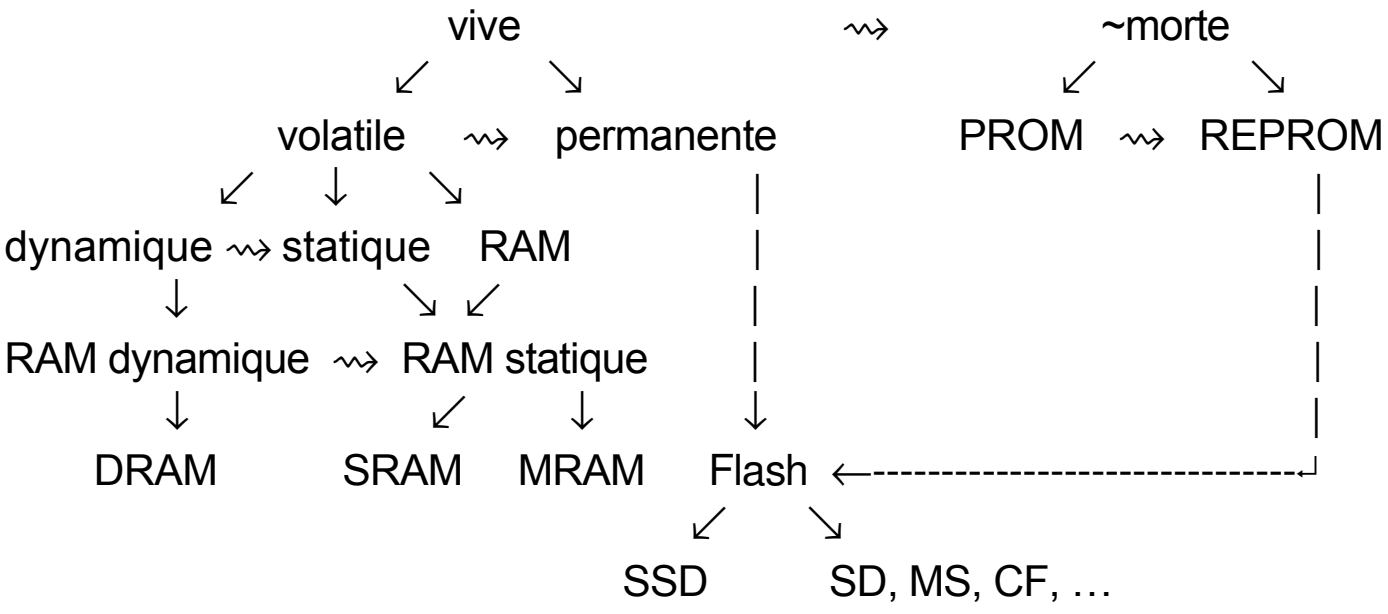
Mémoire ~morte (ROM [Read Only Memory] or Read-mostly).

Mémoire PROM [Programmable ROM]: pouvant être écrite une seule fois par l'utilisateur.

Mémoire REPROM [REProgrammable ROM]: pouvant être écrite un certain nombre de fois par l'utilisateur, e.g.,
- EPROM [Erasable PROM] (effacement par exposition aux ultraviolets),
- EA(P)ROM [Electrically Alterable (P)ROM],
- EE(P)ROM [Electrically Erasable (P)ROM] (effacement par tension électrique)

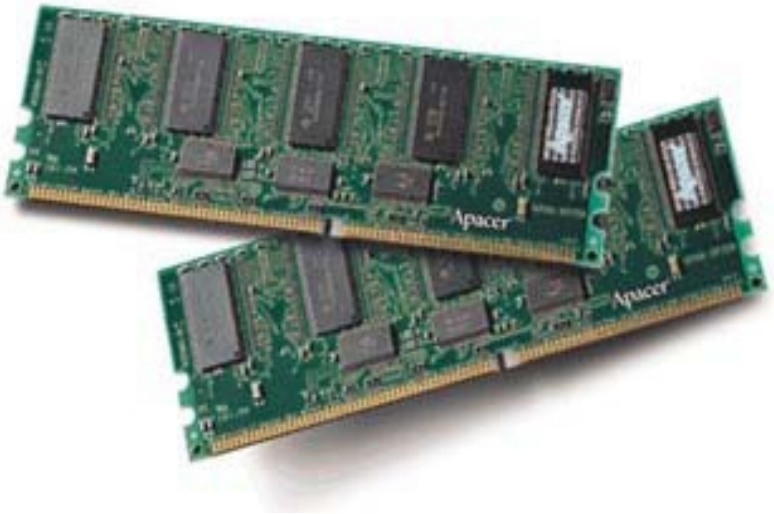
(- Flash : ~EE(P)ROM mais effacement par bloc, pas octet par octet).
Pages intéressantes mais qu'il n'est pas obligatoire de lire : 1, 2

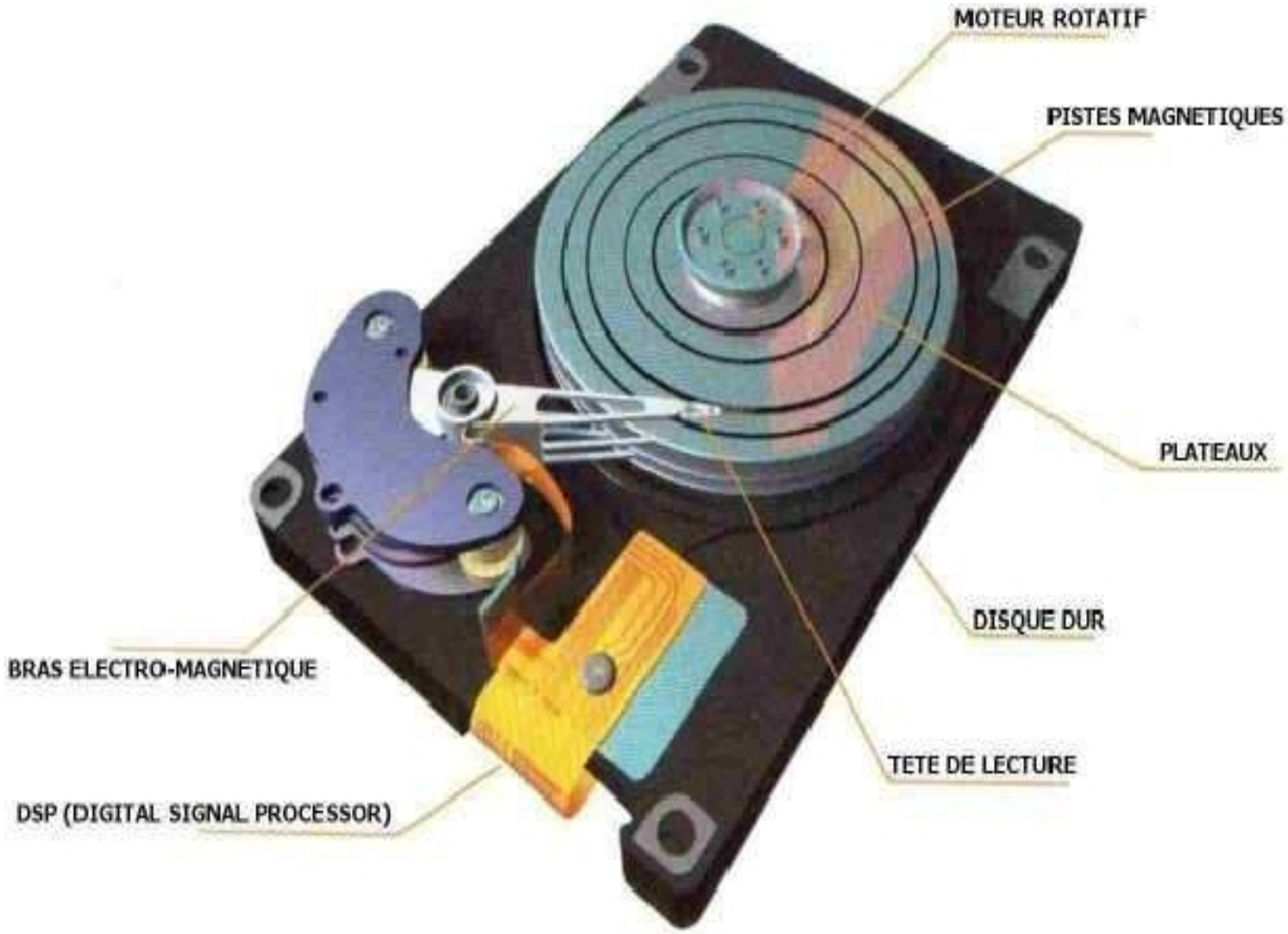
Résumé graphique de cette page et de la précédente :



Plus on s'éloigne du processeur, plus la capacité et le temps d'accès augmentent et plus le prix diminue.

2. Mémoires - exemples de disques durs et de barrettes mémoires





3. UCT/CPU (bus, UAL, UC) - définitions et exemples

3.1. Le bus

3.2. Définitions relatives à l'UAL - jeu d'instruction et registres

3.3. Définitions relatives à l'unité de commande

3.4. Exemples de vision interne et fonctionnelle de l'unité centrale

3.5. Synchronisation: cycles instruction/recherche/commande/CPU

3.6. Séquenceur microprogrammé

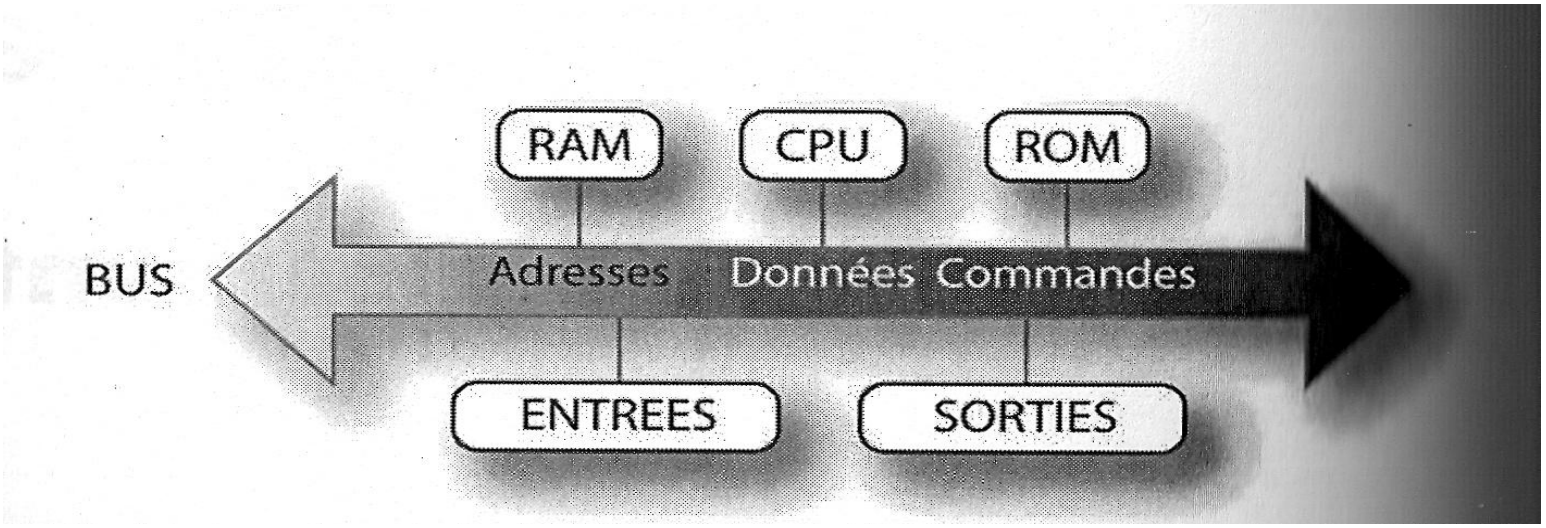
3.7.-3.18. Exemples *(non/peu détaillés oralement -> à examiner plus attentivement par soi-même, pour culture générale)*

3.1. Le bus - définitions

Bus: câbles (lignes de transport d'information: adresses, données et commandes) exploitées par les unités qui y sont rattachées; il peut y avoir un bus unique ou plusieurs.

- bus **d'adresses seulement** ou bien de **données seulement** ou bien de **commandes seulement** ou bien **un mix** de ces derniers;
- - bus **interne** (bus système): bus très rapide (+ de 1 Go/s) permettant à l'UCT/CPU d'accéder la MP (RAM), ou bien
  - bus **local** (bus d'entrées/sorties, bus d'extension): permet l'ajout de périphériques ou de cartes d'extension; relié au bus système par un "pont" [bridge], ou bien
  - bus **externe**: reliant l'ordinateur à des périphériques externes; connecté au bus local; e.g., le bus USB [Universal Serial Bus] qui peut connecter 127 périphériques;
- bus **bidirectionnel** (-> transfert des informations) ou bien **unidirectionnel** (-> lecture en mémoire);
- bus **série** (un seul fil) ou bien **parallèle** (toutefois, à des fréquences élevées, les lignes génèrent des interférences électro-magnétiques qui perturbent les signaux; les lignes parallèles, difficiles à isoler, sont donc peu utilisées).

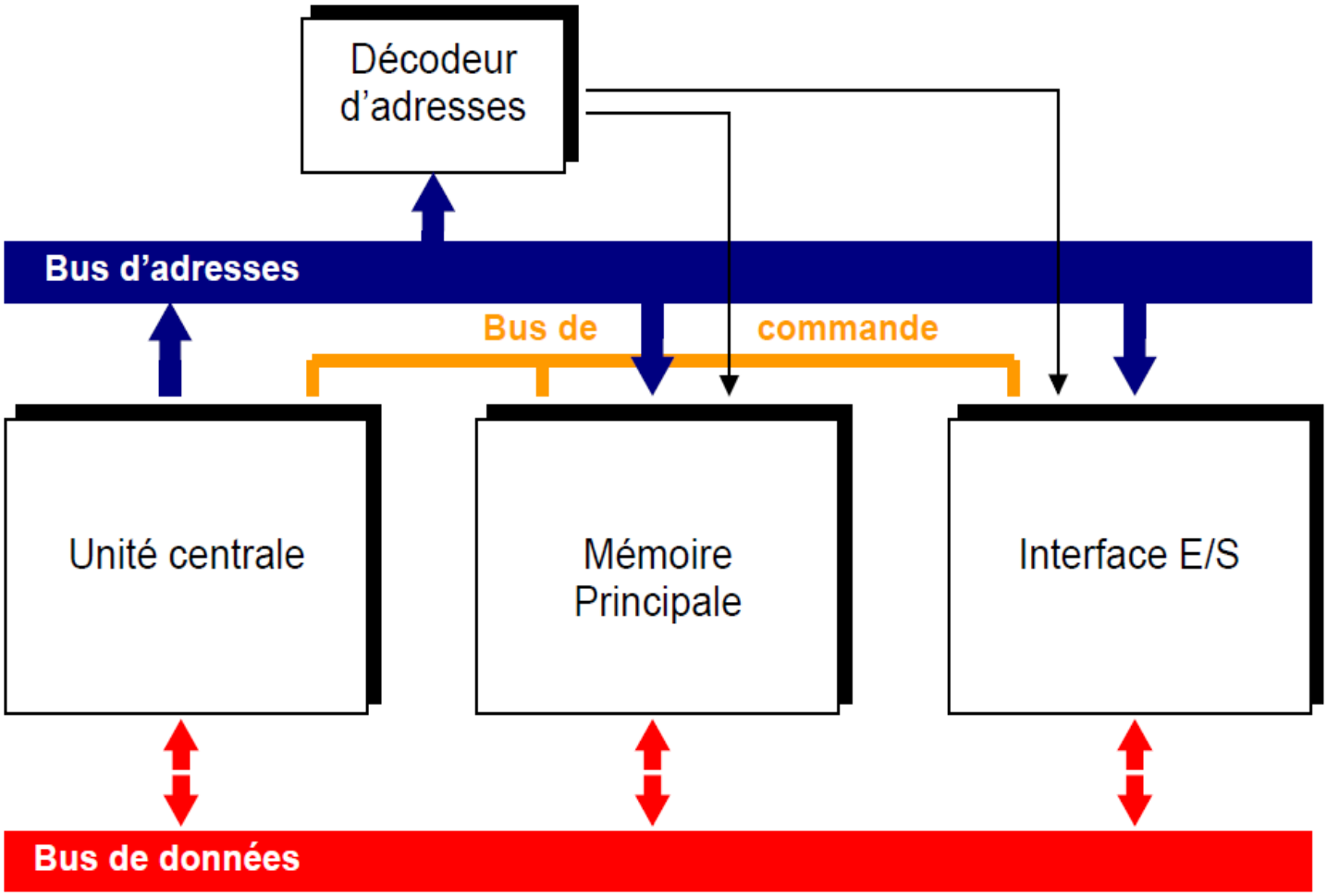
Exemple de bus unique:



3.1. Le bus - le décodeur d'adresses

Décodeur d'adresses: compte-tenu de l'adresse envoyée sur le bus, il envoie un signal d'activation à l'unité destinataire
-> il y a ainsi une seule unité sélectionnée à la fois.

Exemple:



3.2. Définitions relatives à l'UAL - Jeu d'instruction

Jeu d'instruction: ensemble des instructions (opérations) effectuables par l'UAL; e.g., plus de 100 sur des architectures CISC [Complex Instruction Set Code] et 35 sur des architectures RISC [Reduced Instruction Set Code] (détails dans un chapitre ultérieur).
Six groupes:

- **Transfert de données**: Load, Move, Store, transfert de registre/mémoire à registre, etc.
- **Opérations arithmétiques**: les 4 opérations en virgule fixe/flottante et en simple/multiple précision.
- **Opérations logiques**: NOT, AND, OR, XOR, etc.
- **Contrôle de séquence**: branchement impératif/conditionnel, boucle, appel de procédure, etc.
- **Entrées/sorties**: Read, Write, Print, etc.
- **Manipulations diverses**: décalages de bits, conversions de format, incrémentation/décrémentation de registre, etc.

3.2. Définitions relatives à l'UAL - Registres

Registres de l'UAL (ils sont accessibles au programmeur,
contrairement aux registres de l'UC):

- **Registres arithmétiques**: pour les opérations arithmétiques (+, -, *, /, %, complément_à_1, ...) ou logiques (et, ou, ou exclusif);
  les plus utilisés: l'**accumulateur** (ACC) et,
  son extension (pour doubler sa taille), le registre Q
- **Registres de base et d'index**: pour stocker la base ou bien l'index d'un tableau de données et ainsi calculer des adresses dans ce tableau
- **Registre d'état (registre condition)** [Program Status Word (PSW): chacun de ses bits est un "drapeau" [flag] qui indique un état, e.g.,
  le bit C indique un dépassement de capacité dans l'ACC,
  le bit Z si le résultat de l'opération effectuée est égal à 0 (ce bit est utilisé par l'instruction "jump on zero")
- **Registre pointeur (du sommet) d'une pile** [Stack Pointer (SP)]
- **Registres généraux (banalisés)**: pour diverses opérations, e.g., stocker des résultats intermédiaires
- **Registres spécialisés (pour certaines opérations)**, e.g., pour les décalages [shift register], les opérations arithmétiques à virgule flottante [floating point register], etc.

(QW)

3.3. Définitions relatives à l'unité de commande

UC: dirige le fonctionnement de l'UAL, de la mémoire et des E/S, e.g., va chercher, une par une, des instructions en mémoire (et les données qu'elles utilisent), décode chaque instruction, et envoie un signal à l'UAL pour déclencher l'exécution de l'instruction.

Décodeur (d'instructions): a en entrée le **registre d'instruction (RI)** qui contient l'instruction à exécuter; chaque instruction contient
- 1 champ code-opération [opcode (video)] (e.g., le code de l'instruction ADD)
- de 0 à 4 champs opérande (i.e., les paramètres de l'instruction, généralement 1 seul champ).

Pendant que les adresses des champs opérande sont utilisées pour aller chercher les opérandes, le décodeur décode le champ code opération pour indiquer au séquenceur quelle opération (parmi celles du jeu d'instruction) doit être effectuée.
Voir aussi le "cycle de recherche" et le "cycle d'exécution" en section 3.5.

Séquenceur (bloc logique de commande): sous l'impulsion de l'horloge,
- génère les signaux de commande (déclenchant et donc synchronisant l'exécution des différentes unités participant à l'exécution d'une instruction),
- met à jour le **Compteur ordinal (CO)** - alias **compteur de programme (PC)** - qui contient l'adresse de la prochaine instruction.

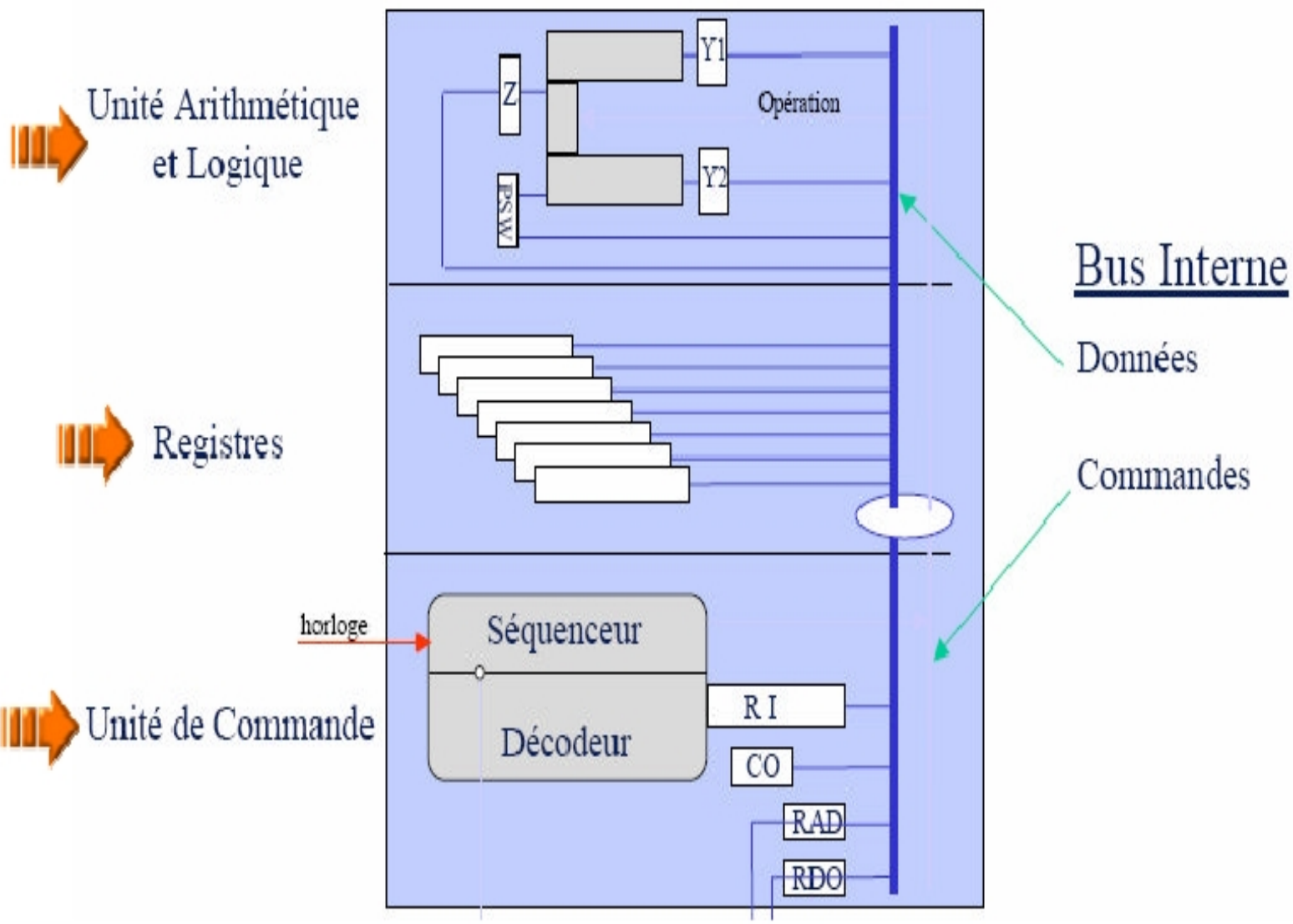
Lorsque les instructions sont exécutées séquentiellement, le CO est augmenté de 1 à chaque cycle de l'UCT/CPU (chaque signal de son horloge);
sinon, un branchement/saut [jump] est fait en mettant une nouvelle adresse dans le CO; le CO est alors utilisé pour chercher l'instruction en mémoire et la mettre dans le RI.

Séquenceur câblé: circuit complexe composé d'un sous-circuit différent (activé par le décodeur) pour chaque instruction.

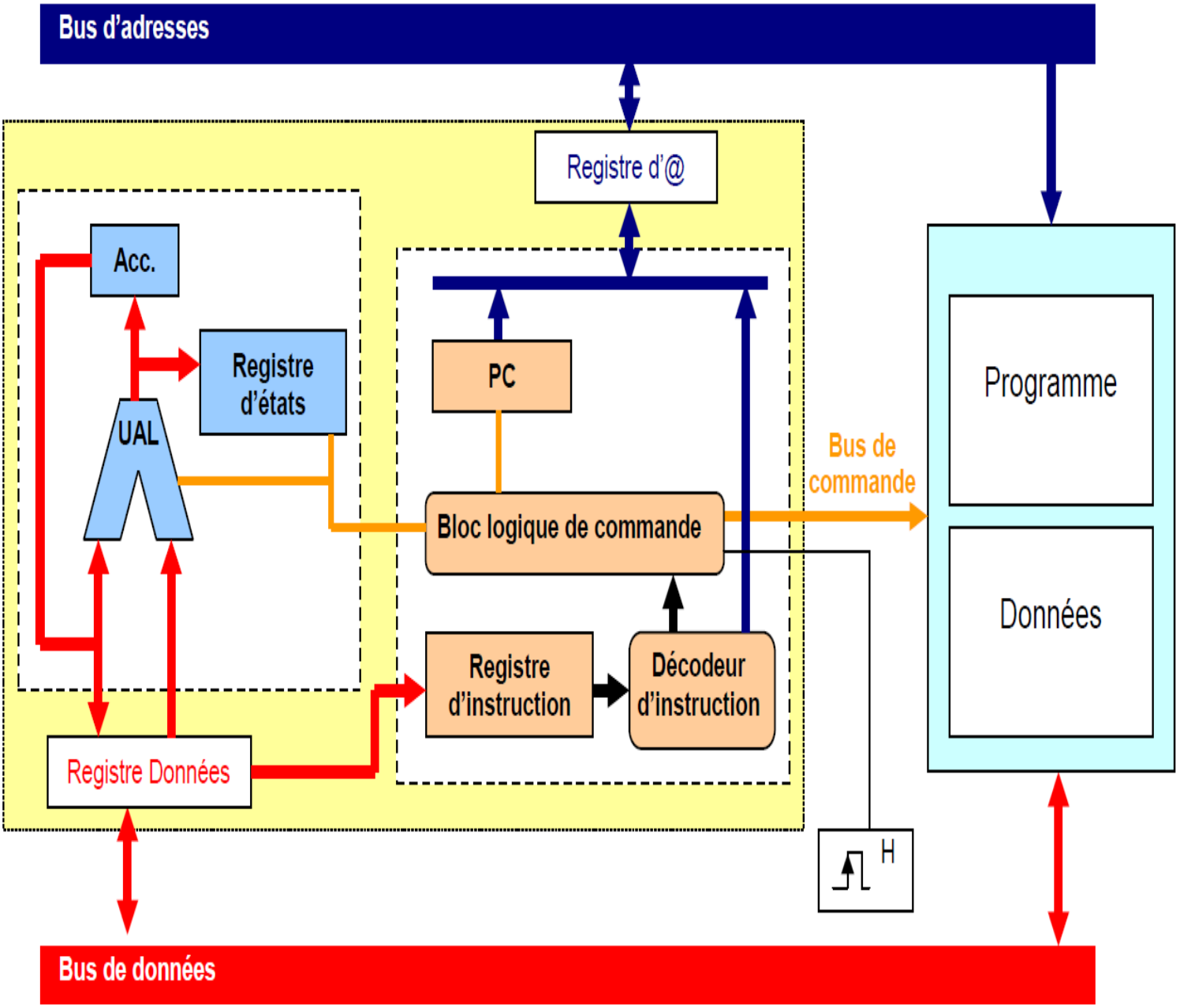
QW

3.4. Exemples de vision interne et fonctionnelle de l'unité centrale

Tout d'abord, juste l'UCT/CPU:

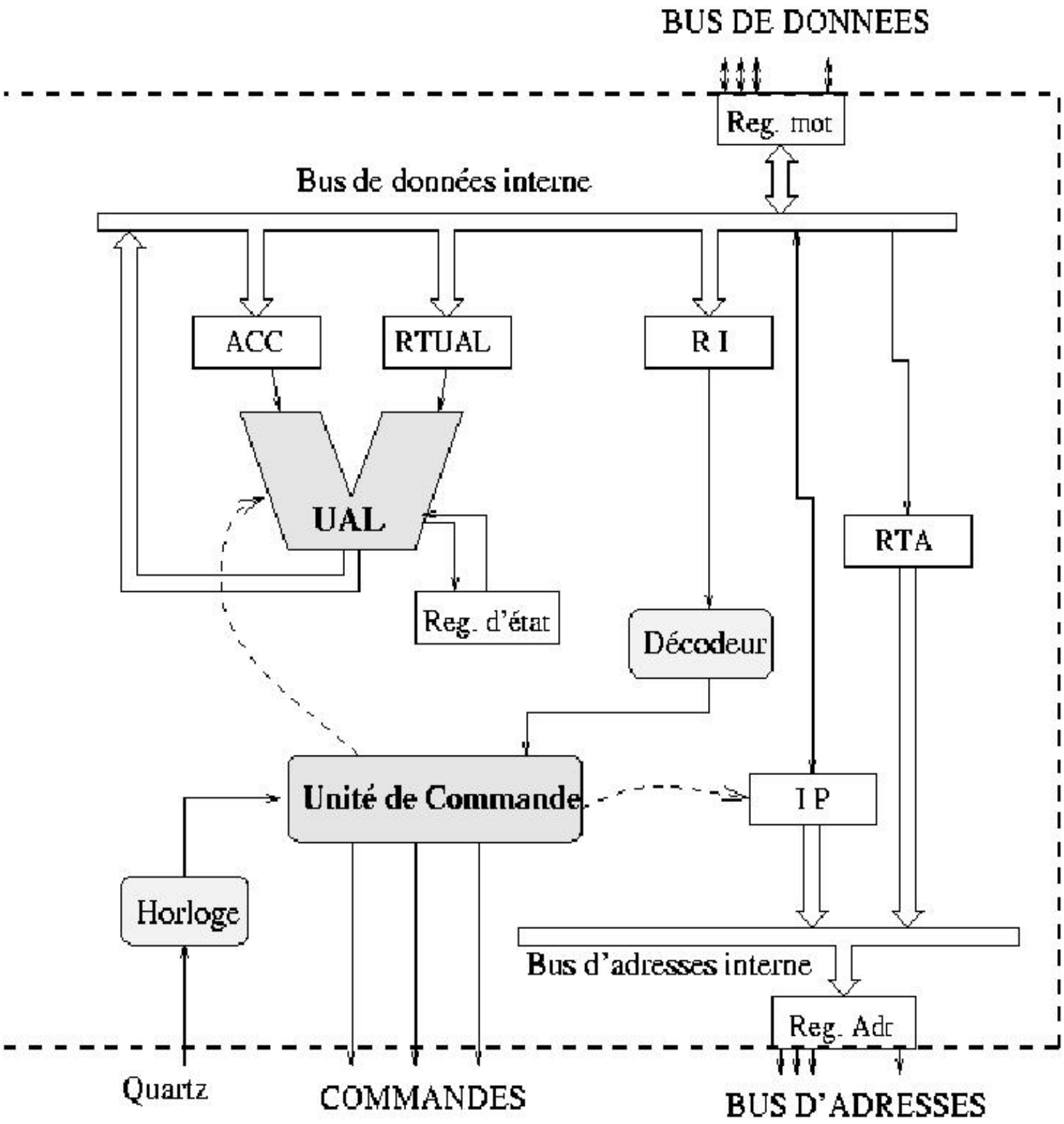


3.4. Exemples de vision interne et fonctionnelle de l'unité centrale

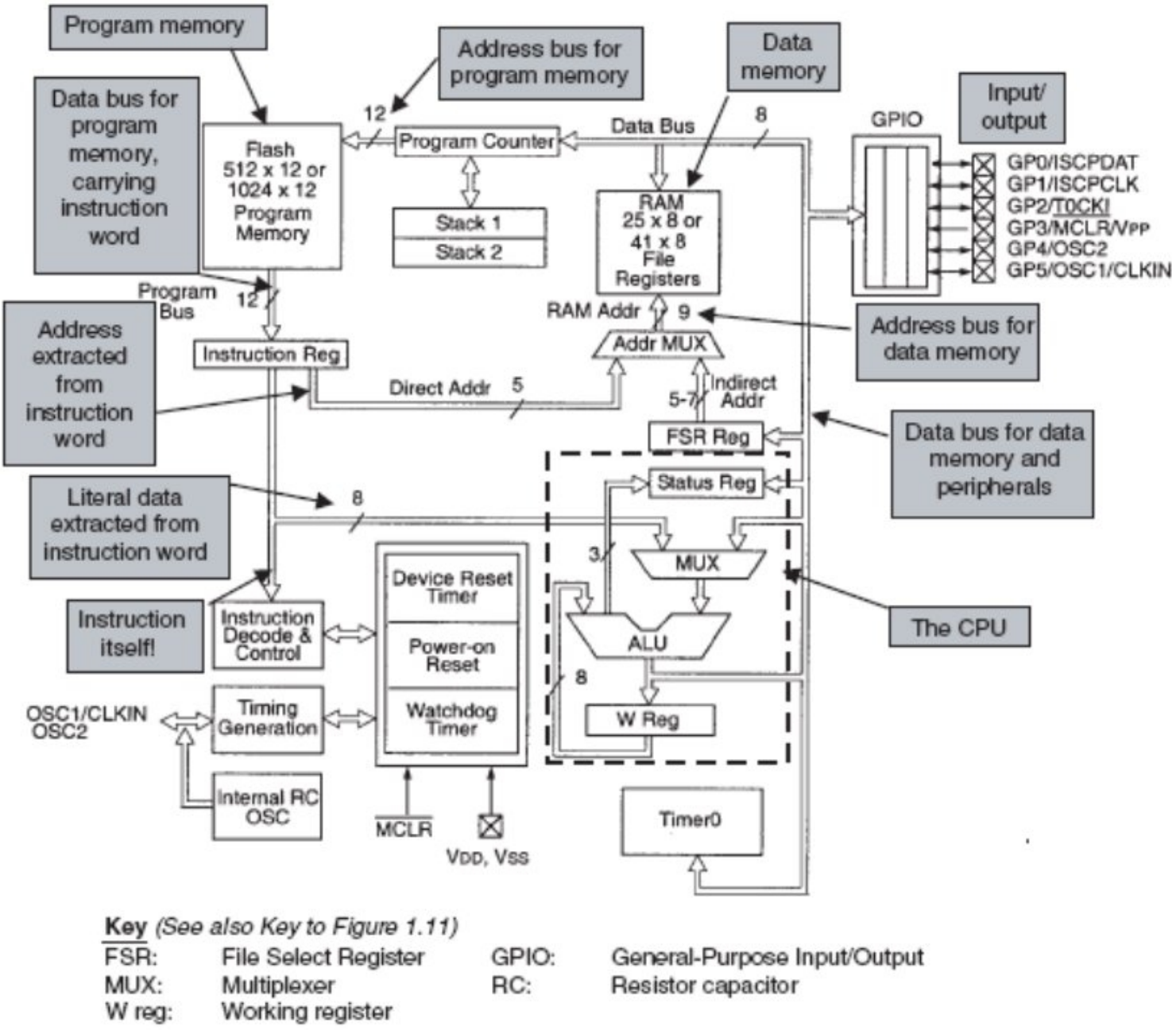


3.4. Exemples de vision interne et fonctionnelle de l'unité centrale

Juste l'UCT/CPU sur cette page ; plus de détails sur la page suivante.

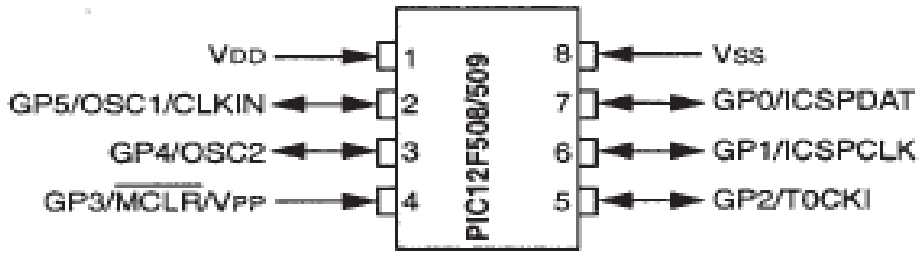


Ci-dessous, le "block diagram" du micro-controlleur PIC12F508/509.
Comme dans la plupart des micro-controlleurs, la mémoire de donnée et la mémoire de programme y sont séparées : c'est l'architecture Harvard, pas l'architecture Von Neumann.



QW : **L'architecture d'un micro-processeur (ou micro-controlleur) doit-elle dépendre de la façon dont les données sont encodées (-> en base 2 pour des supports à 2 états, en base 3 pour des supports à 3 états, …) ? (ne pas tenir compte de la taille des mémoires, bus, …)**
A) non, et que l'implémentation utilise de l'électricité/lumière/gaz/eau/…
B) nécessairement C) oui D) les 2 dernières E) aucune des 4 dernières

Les 8 pattes [pins] du PIC12F508/509 :



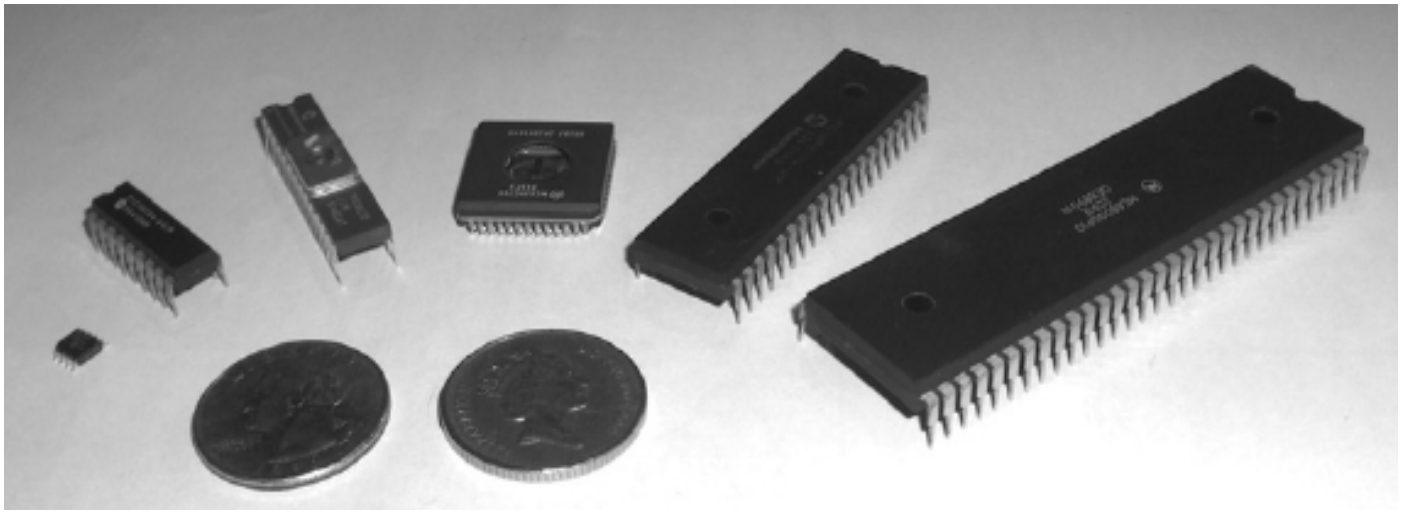
Key

V_{DD}:	Power supply	V_{SS}:	Ground
V_{PP}:	Programming voltage input	MCLR:	Master clear
OSC1, OSC2:	Oscillator pins	CLKIN:	External clock input
GP0 to GP5:	General-Purpose input/output pins (bidirectional except GP3)		
CSPDAT:	In-Circuit Serial Programming™ data pin.		
CSPCLK:	In-Circuit Serial Programming™ clock pin.		

Comparaison de familles de micro-controlleurs PIC :

PIC family	Stack size (words)	Instruction word size	Number of instructions	Interrupt vectors
12CXXX/12FXXX	2	12- or 14-bit	33	None
16C5XX/16F5XX	2	12-bit	33	None
16CXXX/16FXXX	8	14-bit	35	1
17CXXX	16	16-bit	58, including hardware multiply	4
18CXXX/18FXXX	32	16-bit	75, including hardware multiply	2 (prioritised)

Quelques micro-processeurs et micro-controlleurs ; de gauche à droite :
PIC 12F508, PIC 16F84A, PIC 16C72, Motorola 68HC05B16,
PIC 16F877, Motorola 68000.



3.5. Cycles instruction/recherche/commande/CPU

Cycle machine (alias "cycle de base" [clock cycle]): un cycle des signaux périodiques générés par l'horloge.

Cycle instruction: *cycle de recherche* suivi du *cycle d'exécution*;
s'étend sur un ou plusieurs cycles machines.

Cycle de recherche (en résumé ; plus de détails en page suivante):
1) lecture en mémoire – à l'adresse contenue dans le CO – de la prochaine instruction à exécuter et stockage de cette instruction dans le RI,
2) travail du décodeur,
3) travail du séquenceur.

Cycle d'exécution (en résumé ; plus de détails après la page suivante): sous le contrôle du séquenceur, le contenu des champs opérande sont copiés de la mémoire vers et depuis l'UAL ou, dans le cas d'un saut, vers le CO.

Cycle UCT/CPU: temps d'exécution de l'instruction la plus courte (du jeu d'instruction) ou la durée d'une action élémentaire provoquant un changement d'état.

Cycle mémoire: un accès à la mémoire principale;
beaucoup + long que le cycle UCT/CPU ->
- "mémoires entrelacées" permettant le recouvrement des cycles,
- mémoires caches (antémémoires) pour anticiper les transferts vers l'UCT/CPU.

QW

* *"Why not make one big CPU core instead of having multiple cores?"*

3.6. Séquenceur microprogrammé

Séquenceur câblé: circuit complexe composé de sous-circuits différents pour chaque instruction (et séparément activés par le décodeur).

Séquenceur microprogrammé: composé de
- "mémoire de microprogrammation" (très rapide, seulement pour la lecture) associant chaque code opération à un "microprogramme" [firmware] composé de micro-instructions, chacune associant des entrées à une sortie comme dans une table de vérité, donc simulant un circuit logique, et
- un mécanisme pour l'exécution séquentielle - et les branchements conditionnels - des microinstructions.

Microprogrammation horizontale (théorique): les microinstructions sont longues, chaque bit correspond à une commande.

Microprogrammation verticale (théorique): chaque microinstruction est courte et correspond à une commande -> travail de décodage pour générer un signal de commande.

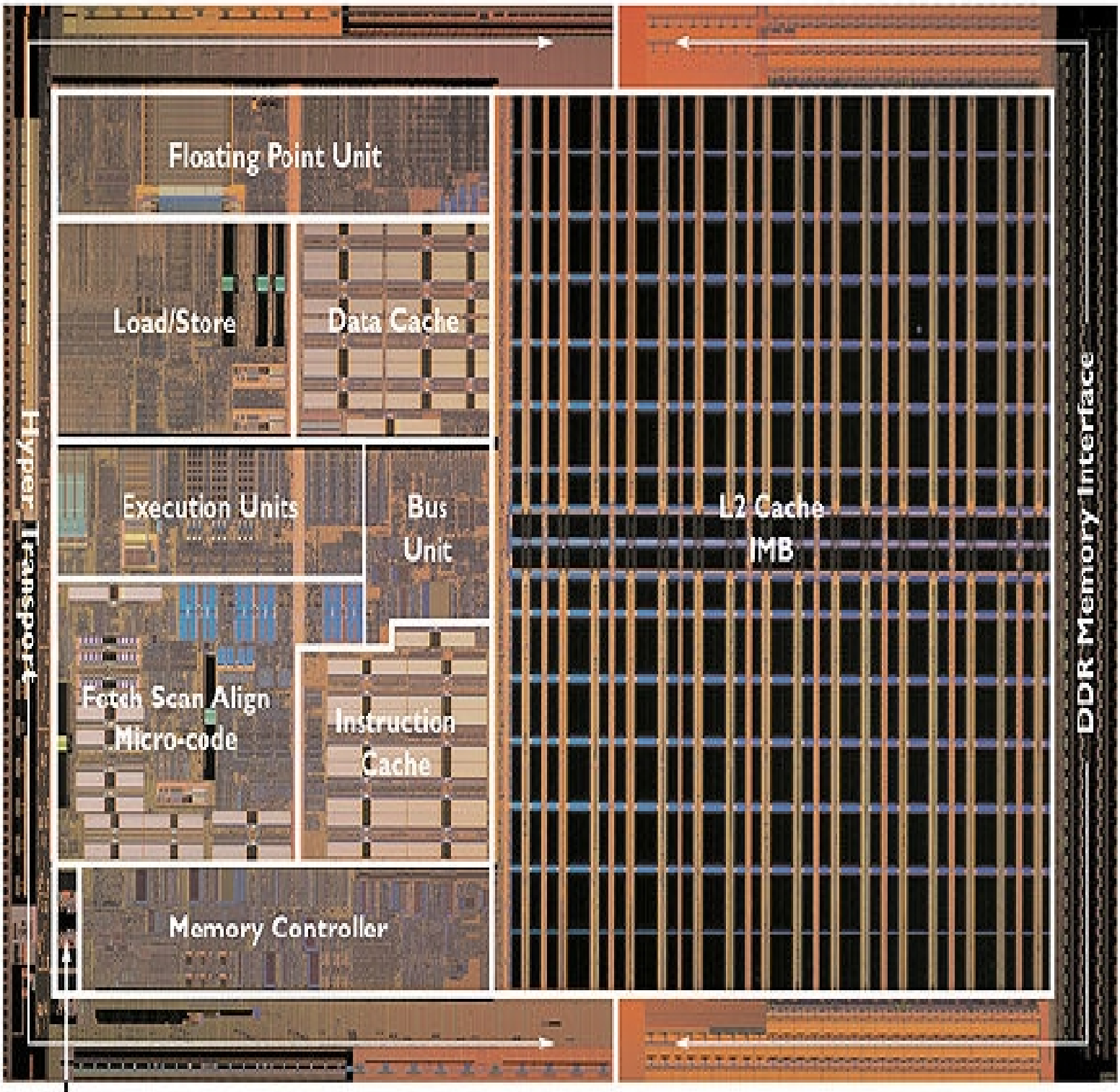
Codage par champs (un compromis pratique): plusieurs unités fonctionnelles sont contrôlées par une seule microinstruction.

Codage de type instruction (autre compromis pratique): chaque microinstruction a elle-même un code opération -> **séquenceur nanoprogrammé**.

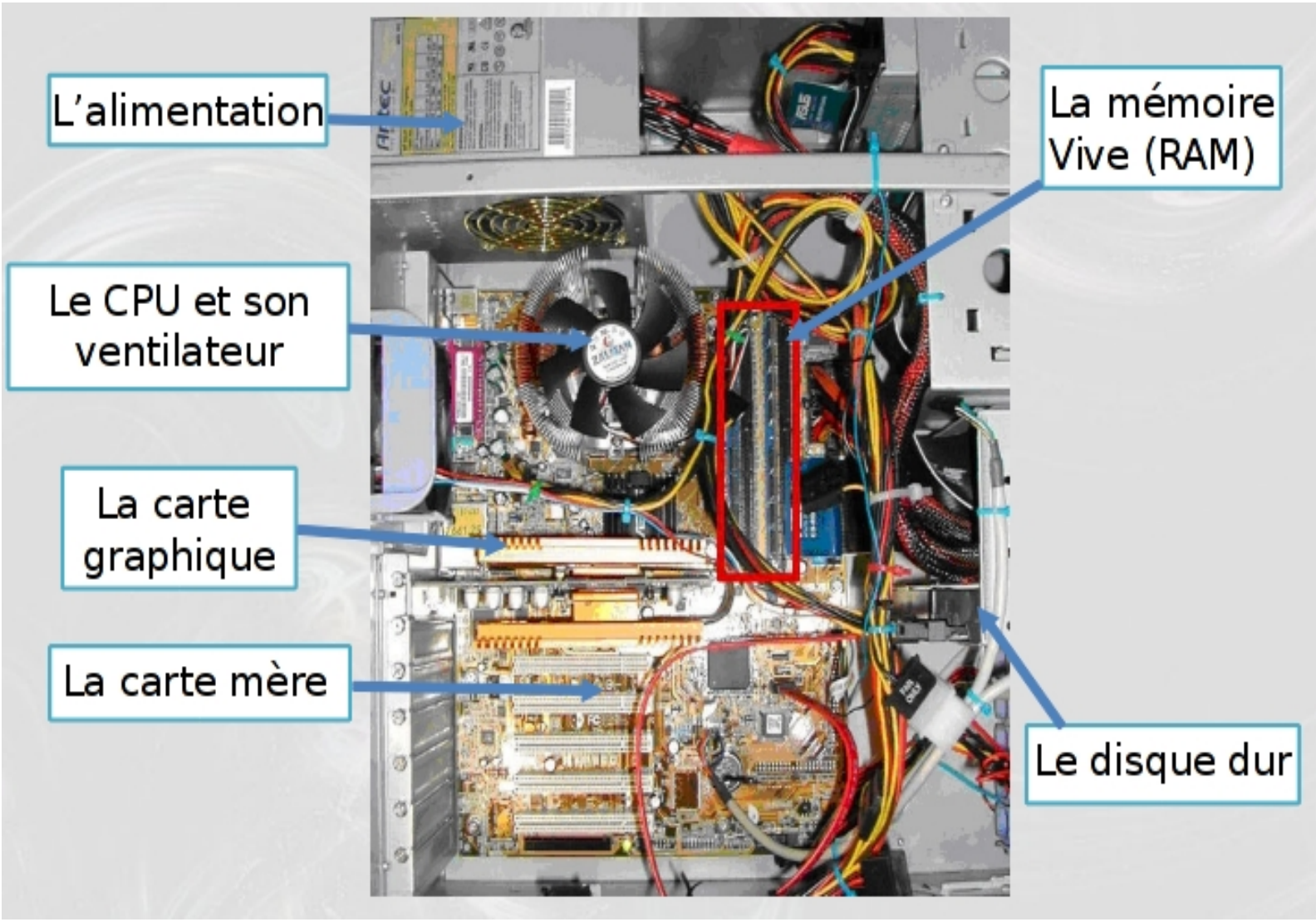
Machine microprogrammable: dont la microprogrammation est accessible au programmeur.

Vous devriez maintenant être capable de comprendre ces pages sur "l'intérêt ou non d'un système d'exploitation de 128 bits" (au lieu de 64 bits) : 1 (lisez en particulier le texte de Wim ten Brink) et 2.

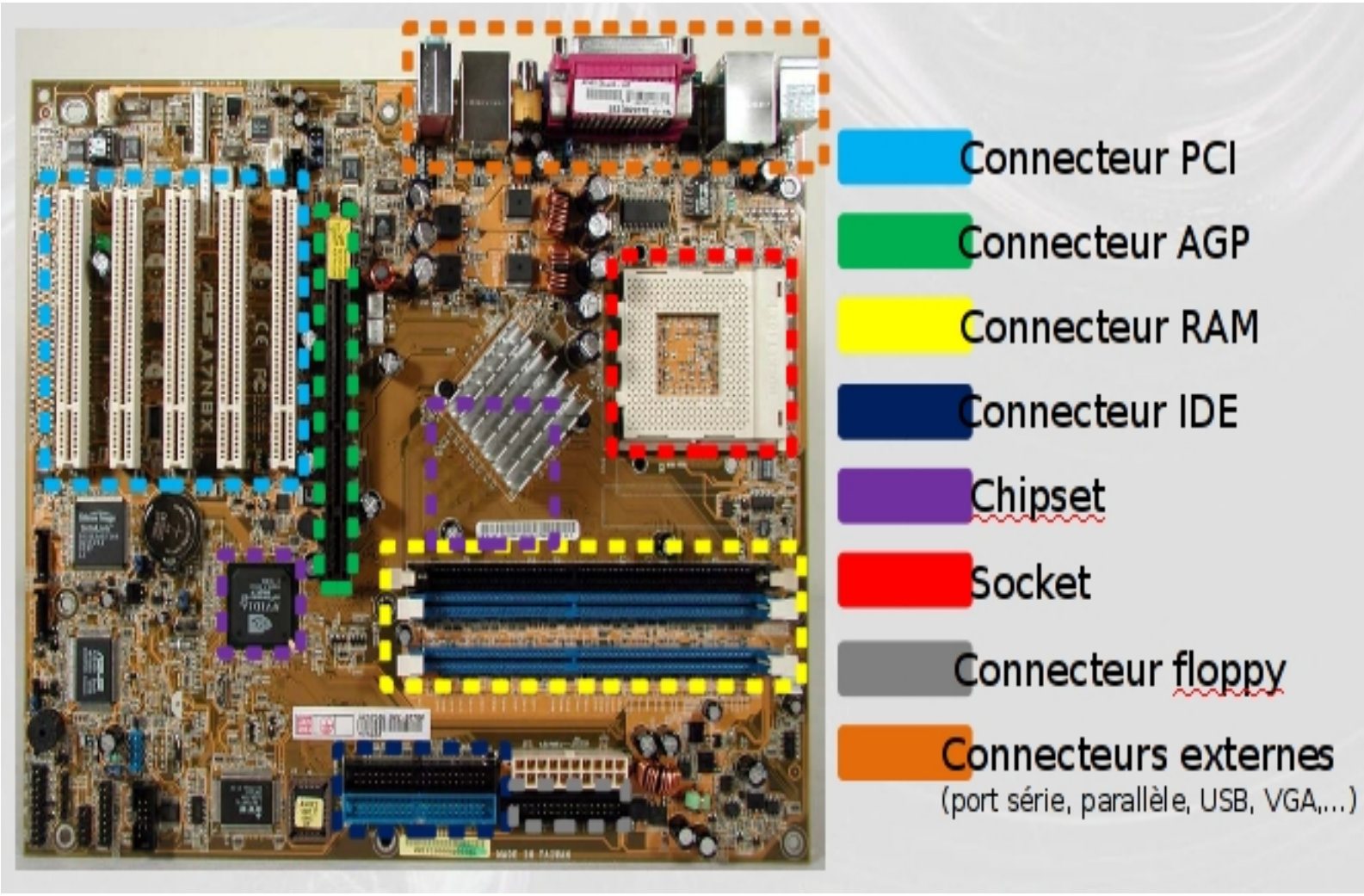
3.7. Exemple de processeur avec mémoire cache



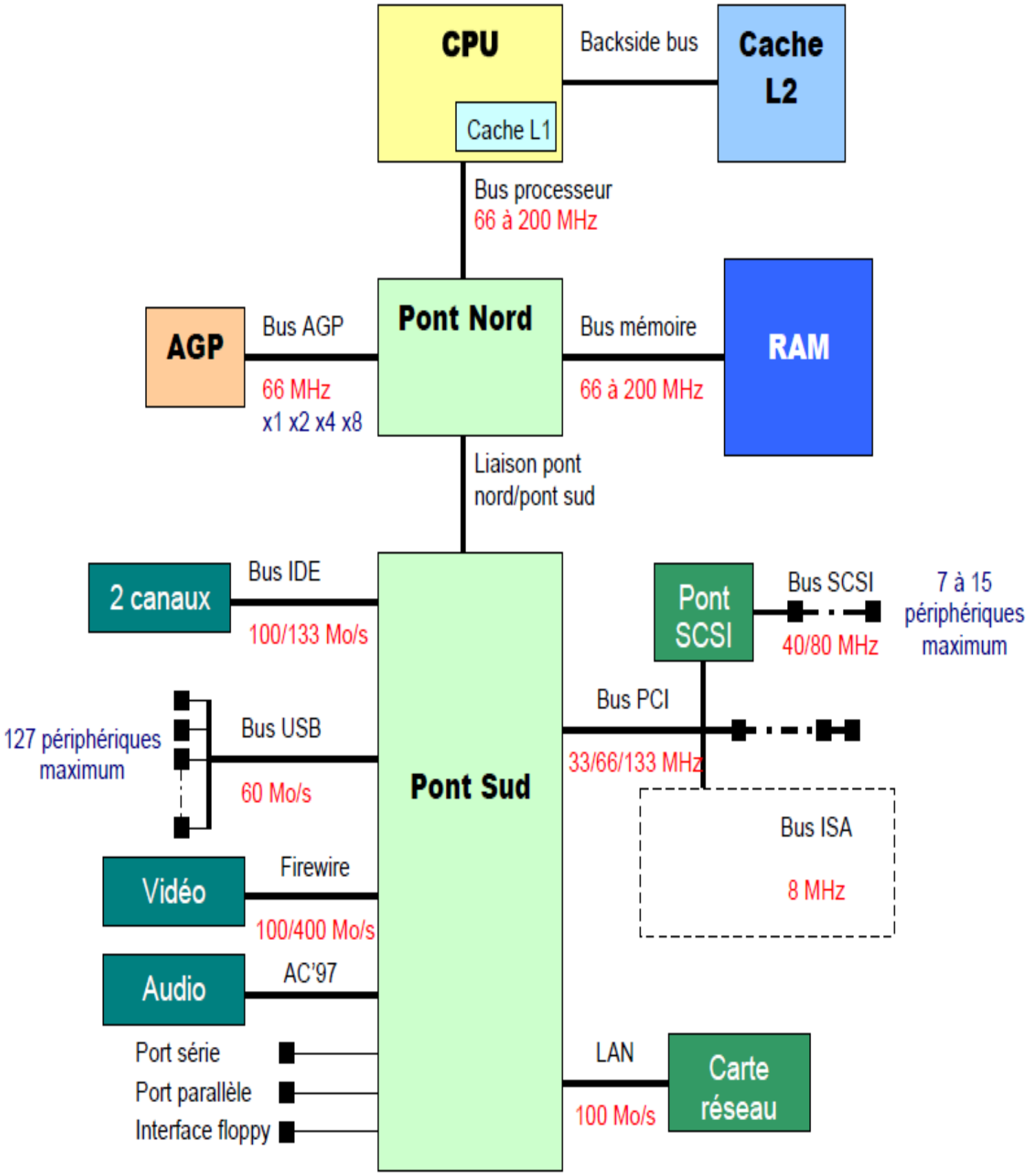
3.8. Exemple de composants d'ordinateur



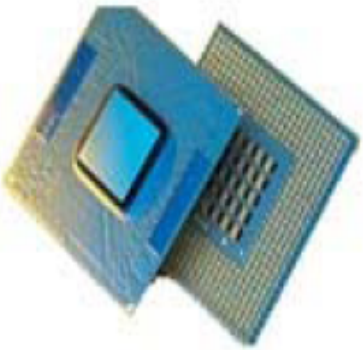
3.9. Exemple de carte mère



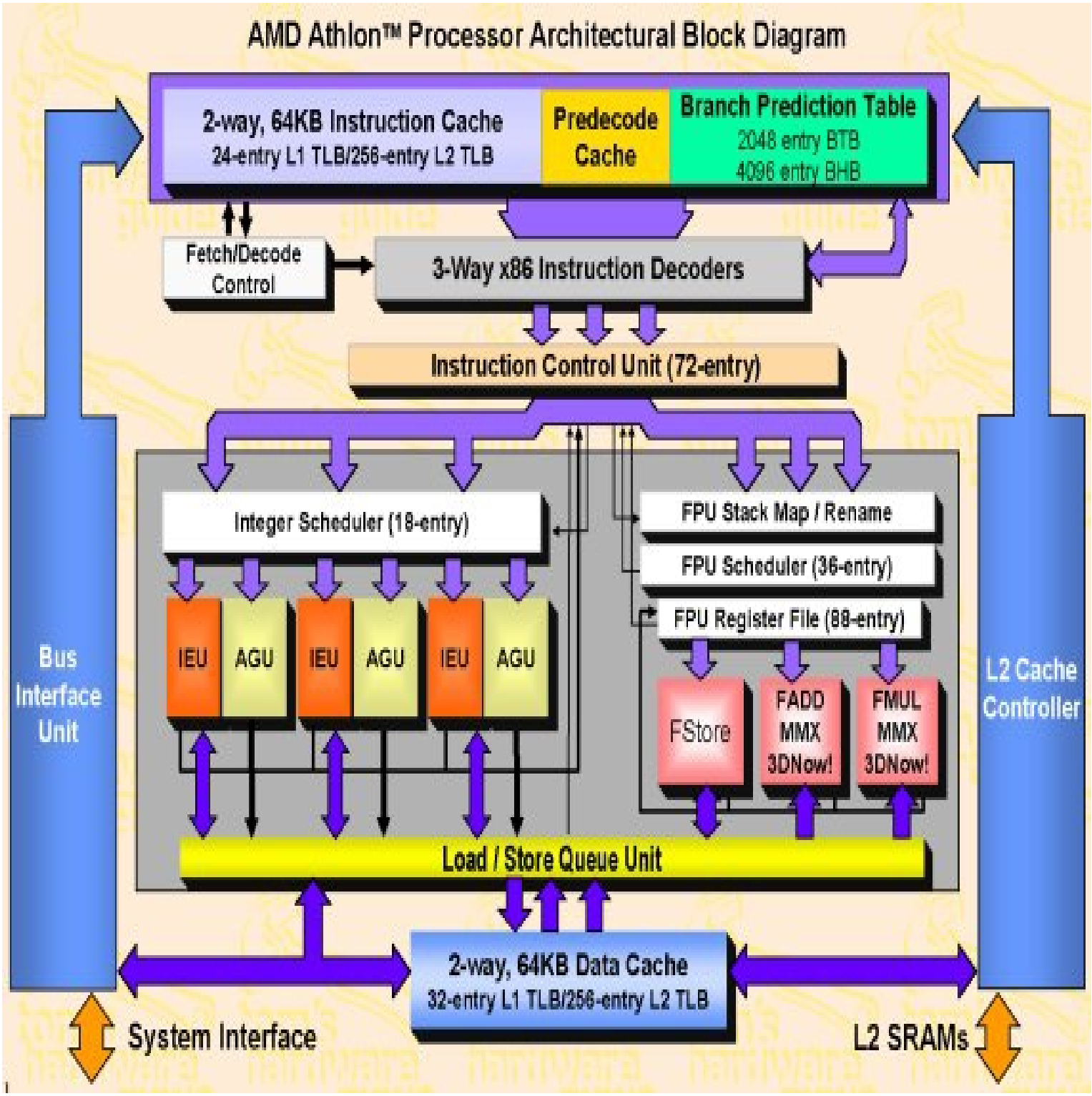
3.10. Exemple d'architecture d'une carte mère



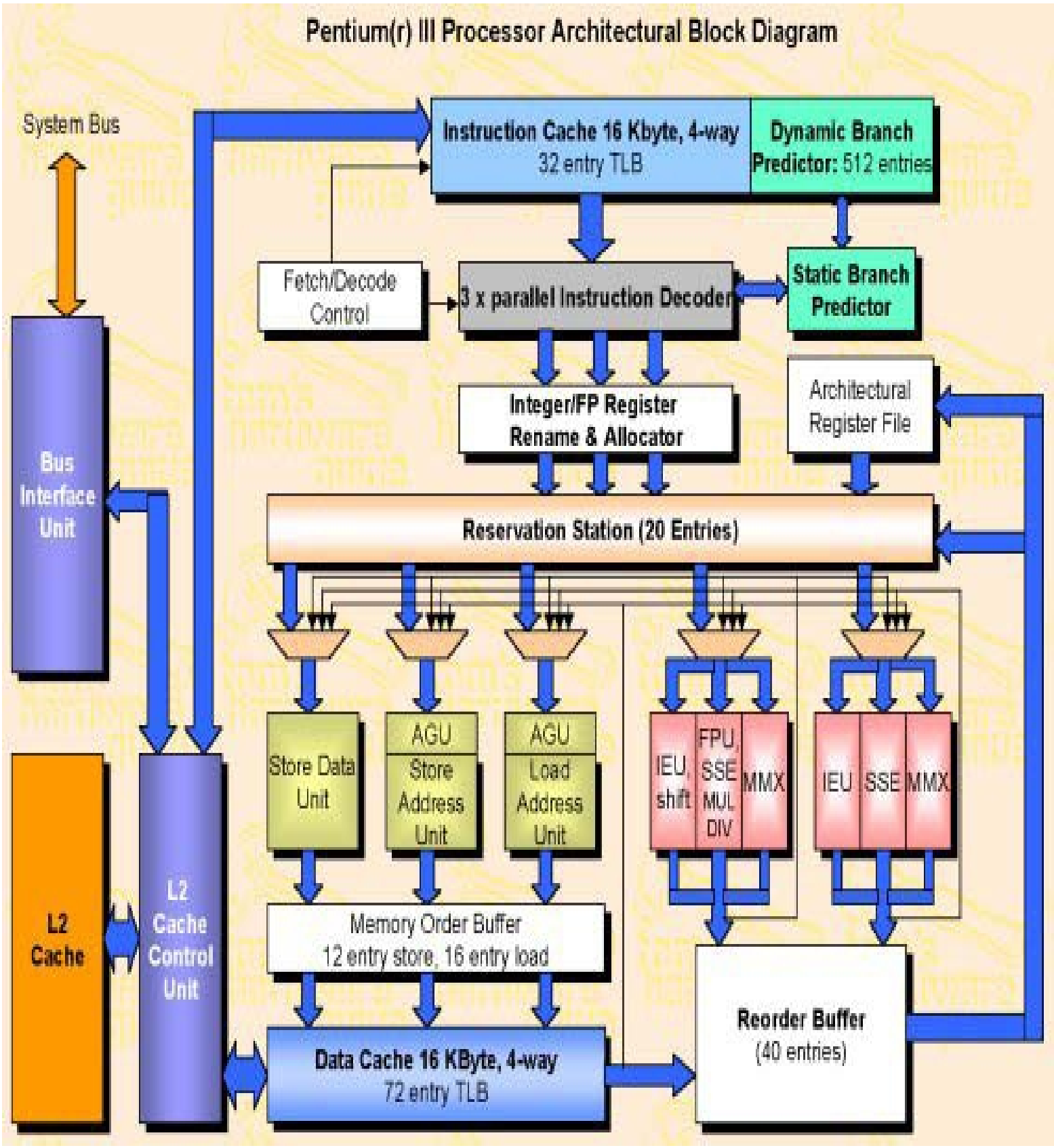
3.11. Exemple de microprocesseurs

			
Référence	**Athlon 64 4000 +**	**Pentium 4 3.4GHz Extreme Edition**	**Pentium M 2GHz**
Support	Socket 939	Socket 478	Socket 478 (portable)
Fréquence	2400 MHz	3400 MHz	2000 MHz
Bus processeur	200 MHz	200 MHz quad pumped	100 MHz quad pumped
Finesse gravure	0.13 µm	0.13 µm	0.09 µm
Cache L1	128 ko	8 ko	32 ko
Cache L2	1024 ko	512 ko	2048 ko
Fréquence cache L2	2400 MHz	3400 MHz	2000 MHz
Architecture	AMD K8	Intel NetBurst	Intel Dothan

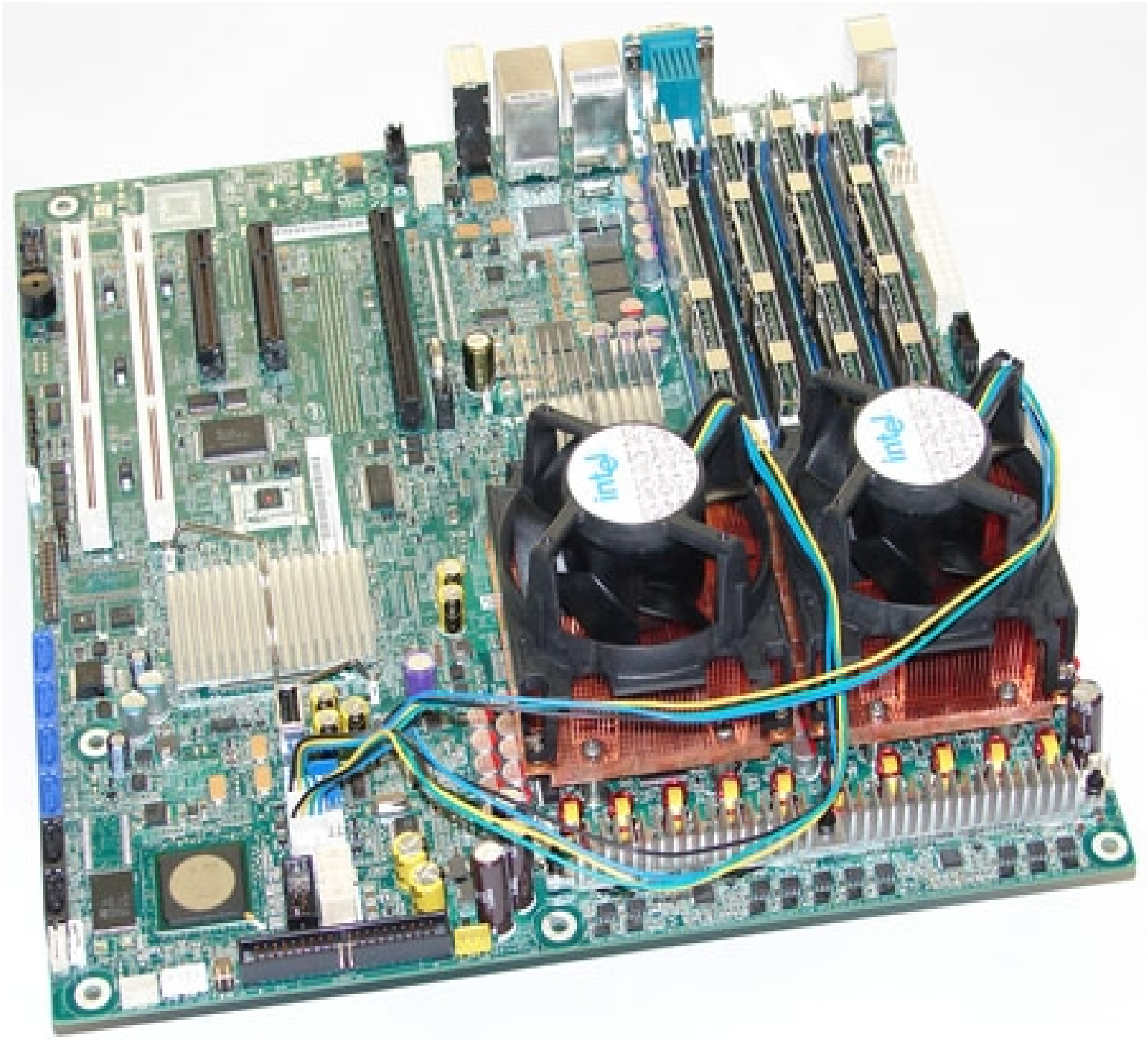
3.12. Exemple d'UCT/CPU



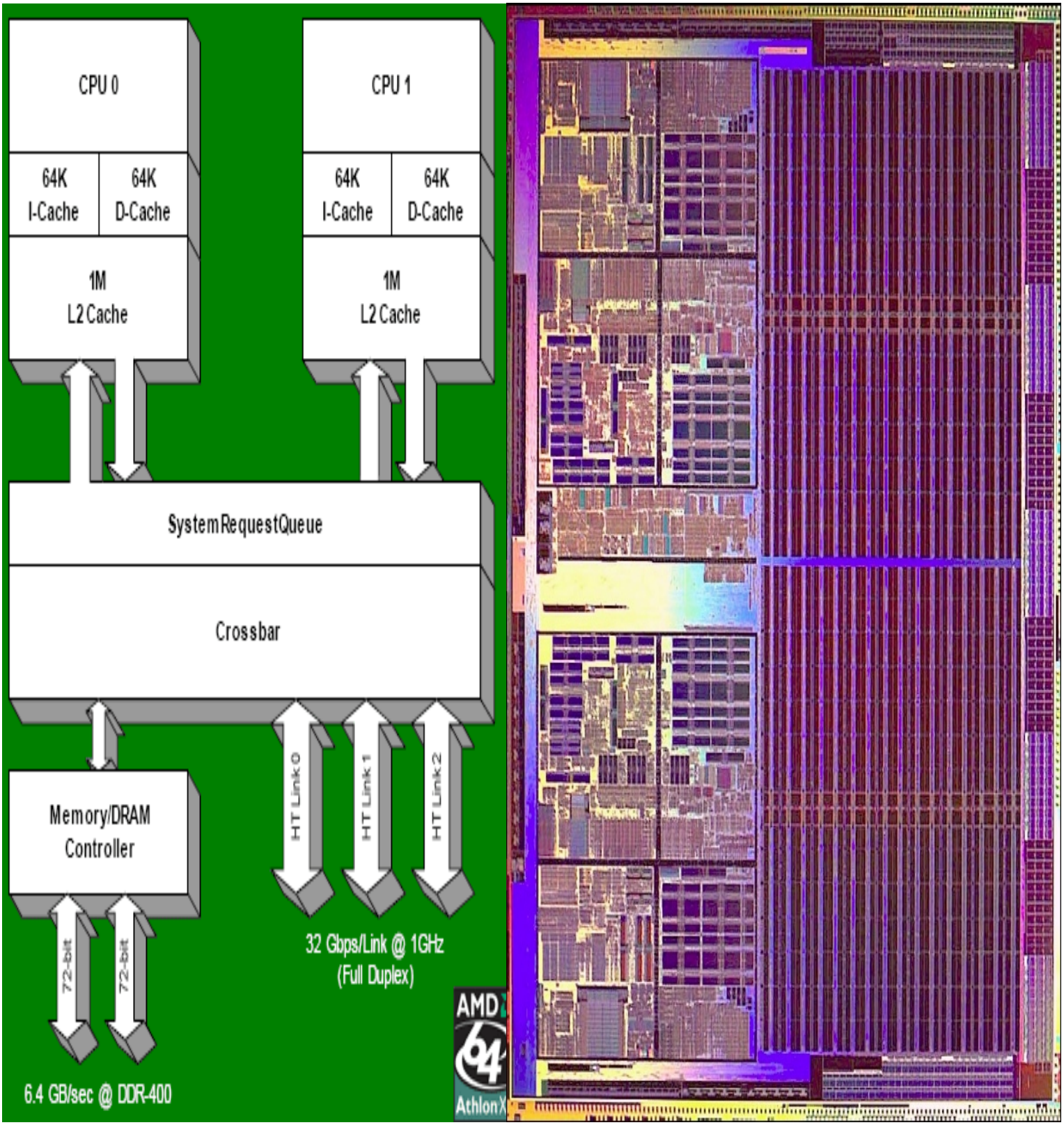
3.13. Autre exemple d'UCT/CPU



3.14. Exemple de carte mère à double UCT/CPU

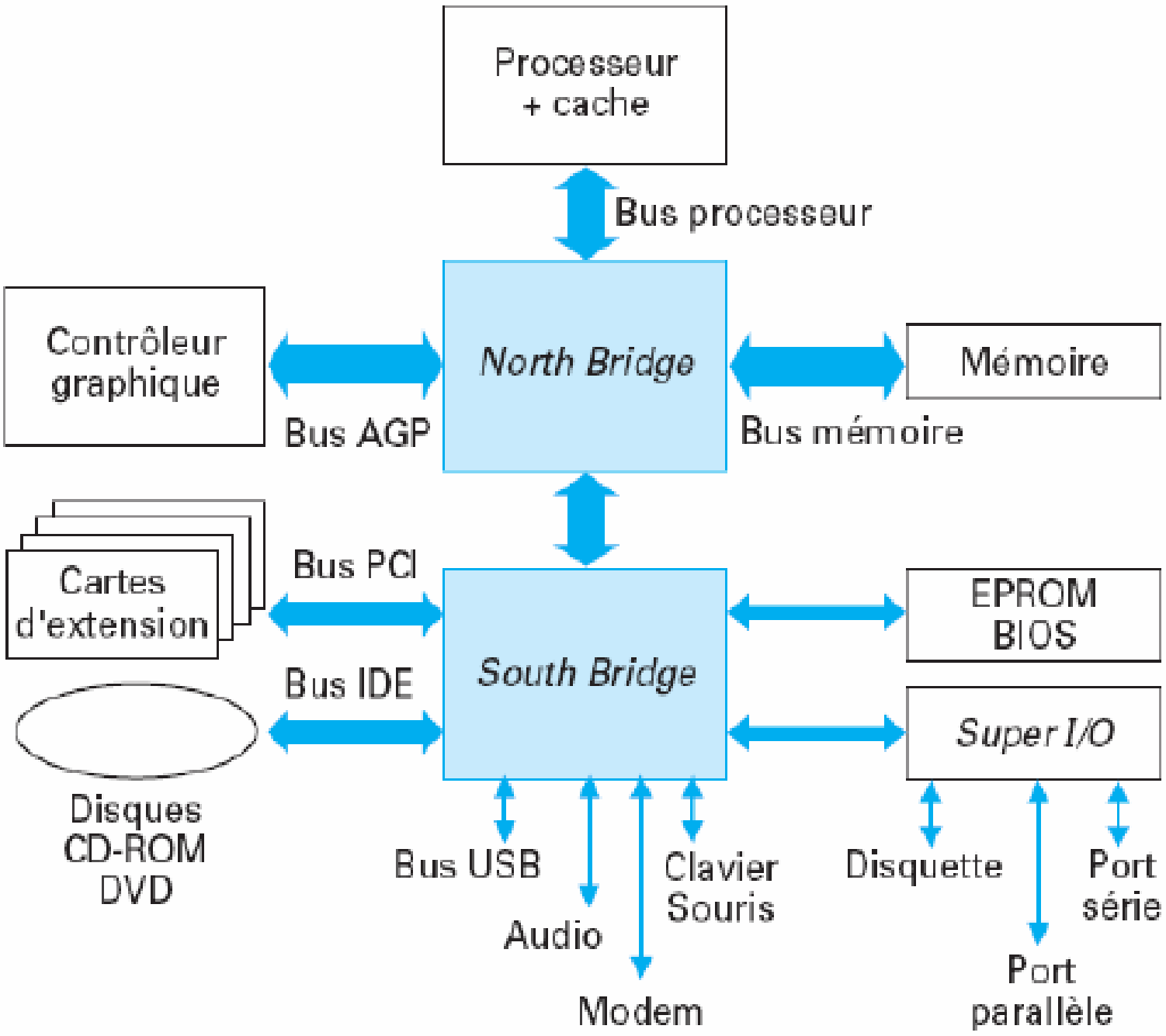


3.15. Exemple de microprocesseur double cœur

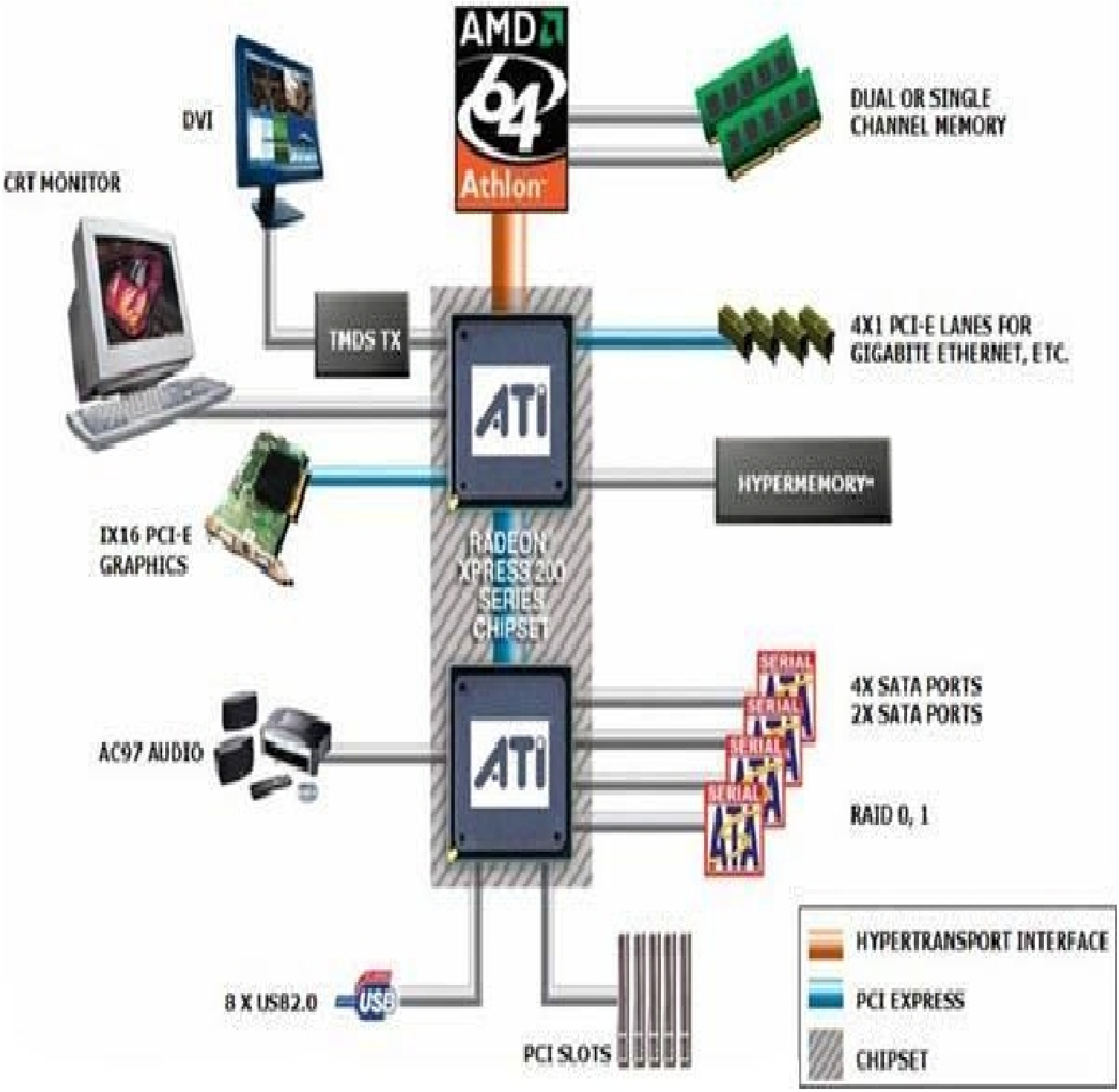


3.16. Exemple de microprocesseur à deux ponts

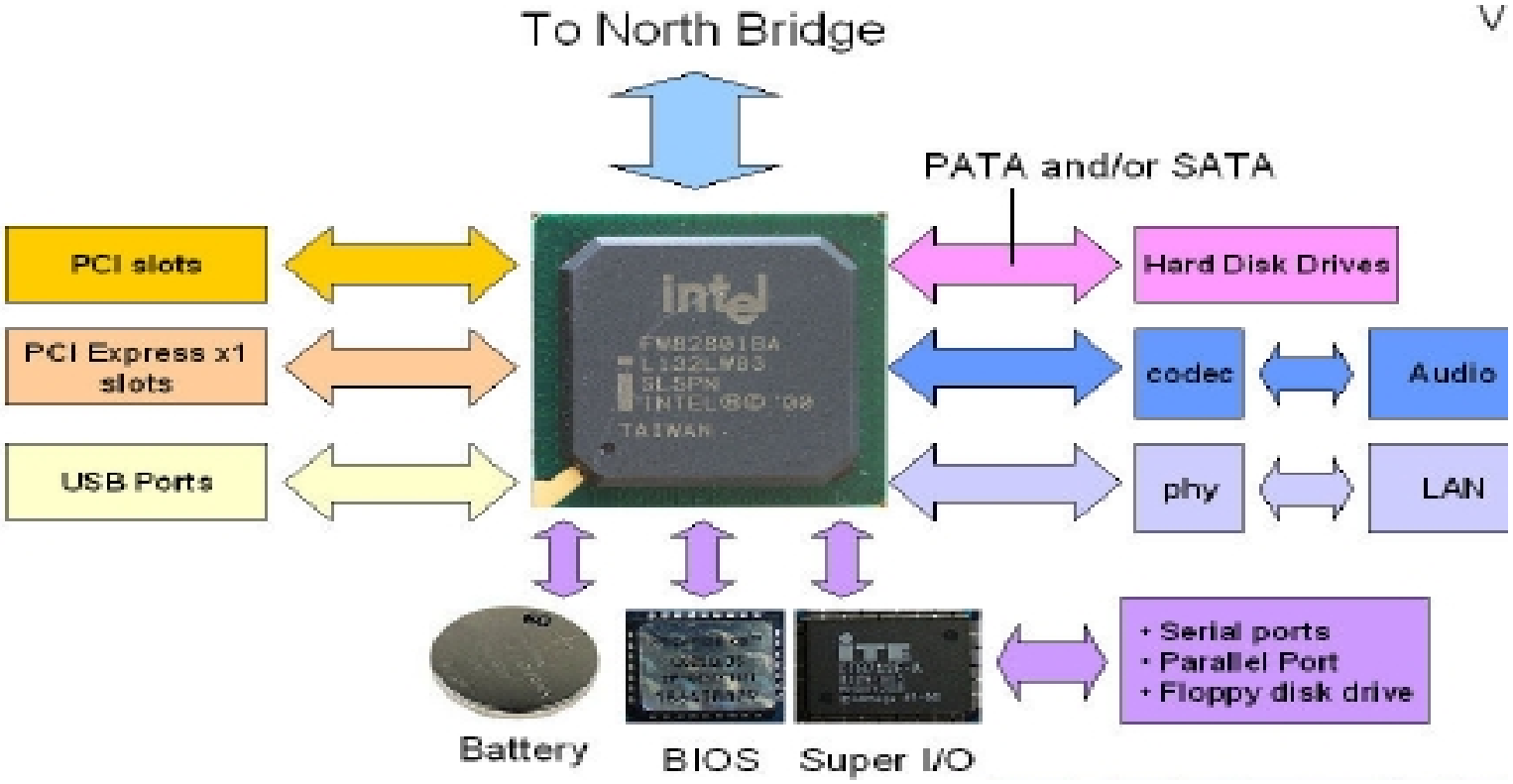
Ci-dessous, un schéma. La carte entière est à la page suivante.

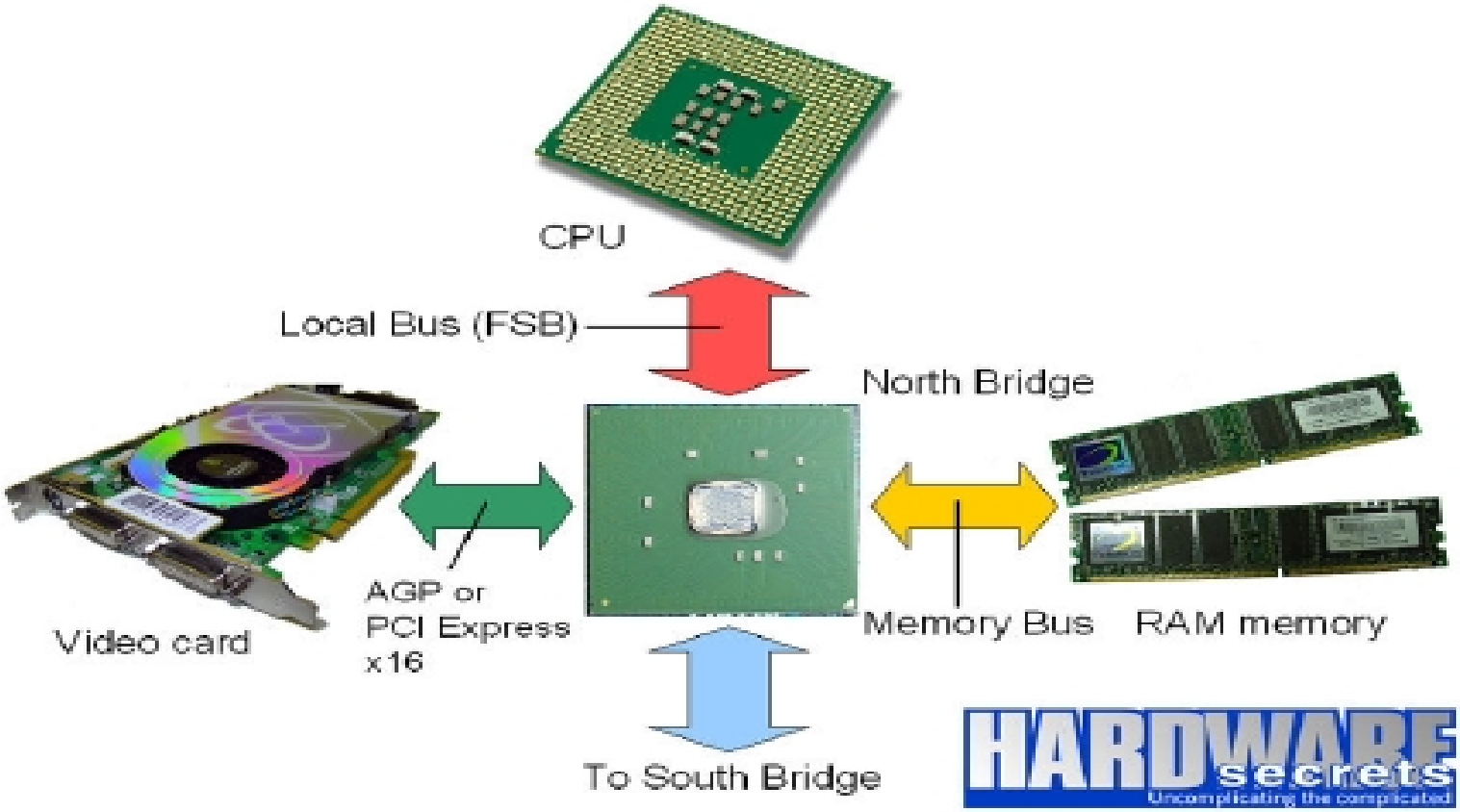


3.17. Exemple de microprocesseur à deux ponts



3.18. Exemple de microprocesseur Intel





4. Instructions et programmation - niveaux

Langage machine (binaire): jeu d'instruction (machine) 'directement-compréhensible | exécutable' par la machine.
Code machine|objet|binaire: programme (quasi-)exécutable par la machine.
Code source|non-binaire: code écrit dans un langage de programmation.

Langage **interprété** (e.g. Julia, Python) **/ compilé** (e.g. C, C#, C++, Rust, Java).

Compilateur: traducteur d'un code source en langage machine ou bien dans une "machine virtuelle de bas niveau" [Low Level Virtual Machine (LLVM)] qui traduit en langage machine – C++, Rust, Scala et Java bytecode utilisent cette seconde solution ; le compilateur de Java génère du Java bytecode qui est *interprété* sur une "machine virtuelle Java" [Java virtual machine (JVM)].
Compilateur "à l'avance" [ahead of time (AOT)|statique] → classique.
Compilateur "à la volée | juste-à-temps (JAT)" [just in time (JIT) | dynamic | run-time] → avec V8, JS et WebAssembly code sont JAT-compilés dans du bytecode V8.

Interpréteur: comme un compilateur (-> vérification syntaxique, ...) **mais**
1) **ne génère** le code machine **que pour** les parties qu'il ***exécute (en +** donc)*,
2) **ne "vérifie" pas la cohérence interne (-> la sémantique** (!= la syntaxe)) d'un programme : il s'arrête à la première erreur syntaxique (instruction incorrecte) ou d'exécution (e.g., dépassement de capacité); c'est de toute façon inutile pour du bytecode (qui, rappel, est toujours interprété).

Éditeur de liens: combine des codes objets en 1 code chargeable+exécutable.

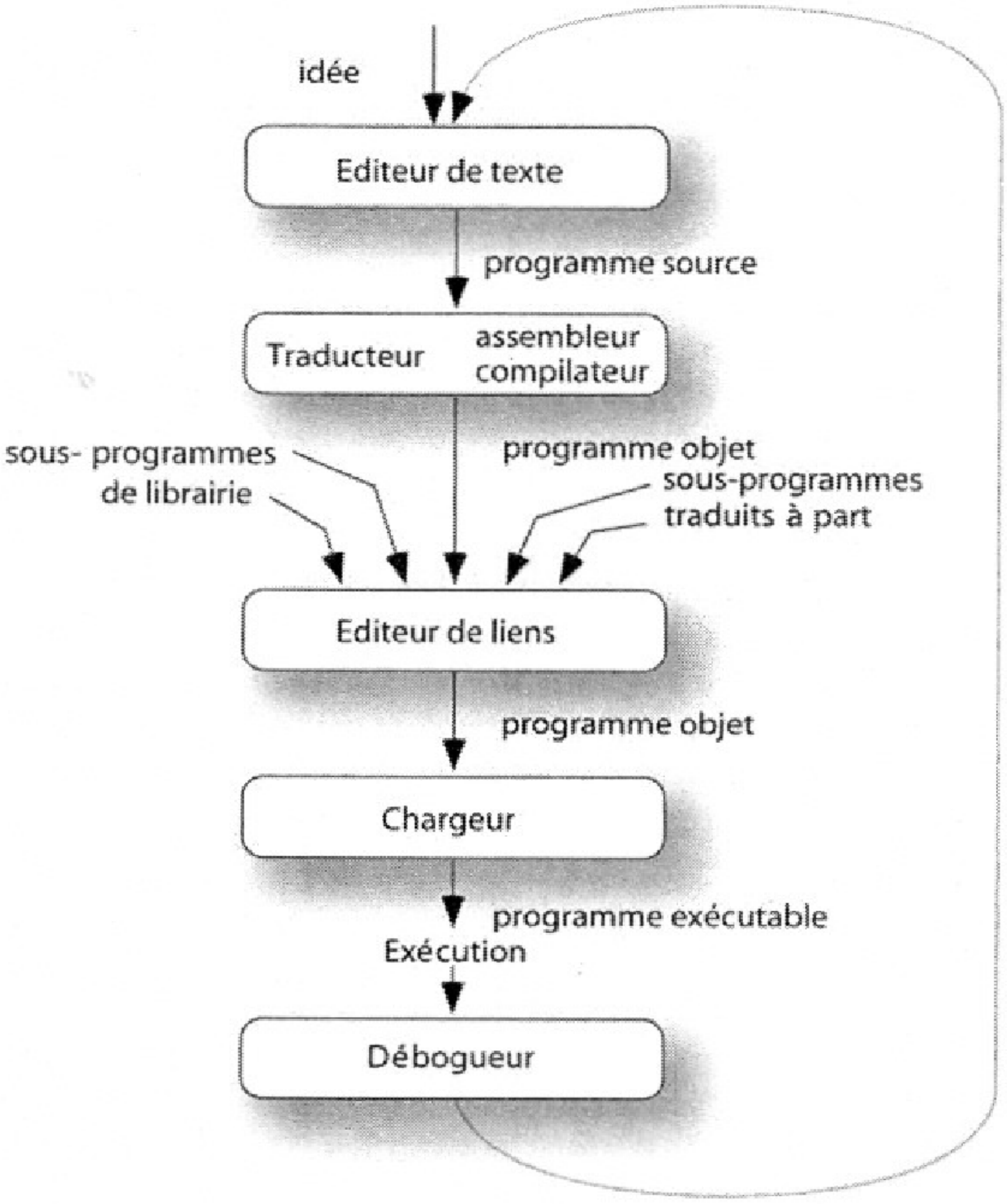
Chargeur: charge le code objet final en mémoire principale pour exécution.

Assembleur: langage de programmation structurellement proche du langage machine (~ 1 mnémonique pour chaque commande du jeu d'instruction) donc très facilement compilable ou interprétable.

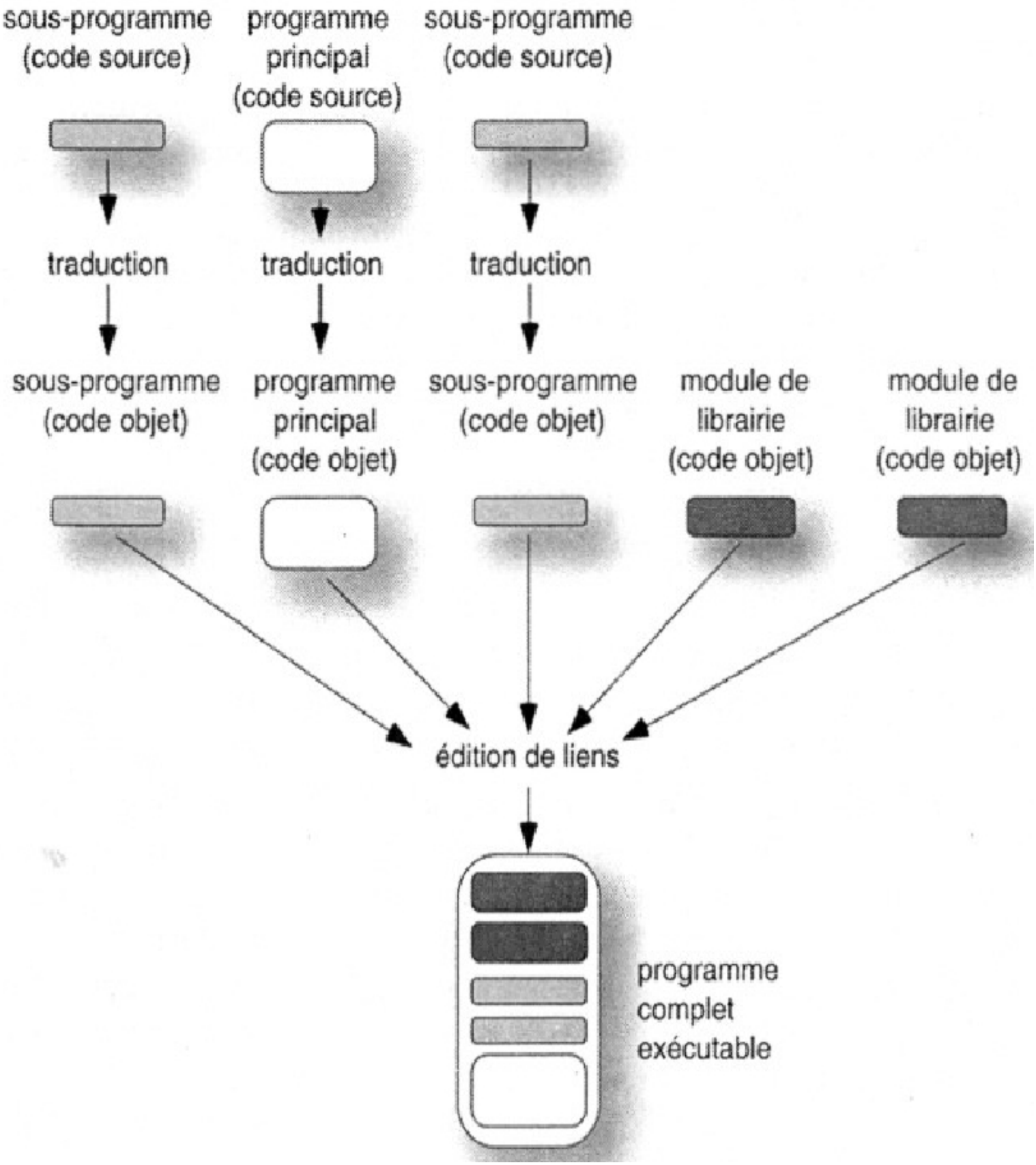
Langage évolué: langage de programmation qui n'est pas un assembleur.

Langage évolué de bas niveau: langage évolué dont le jeu d'instructions permet *aussi* d'effectuer des opérations sur des bits (ET, OU, décalage, ..) e.g., le langage C contrairement au langage Pascal. **QW**

4. Instructions et programmation - chaîne de développement



4. Instructions et programmation - édition de liens



* Exemples de codes pour le programme "Hello World" dans divers langages

4. Instructions et programmation - structure des instructions

Exemple de champs opérande pour une instruction:
- adresse du résultat (inutile si l'ACC est toujours utilisé)
- adresse d'un paramètre d'entrée (inutile si une pile est utilisée)
- adresse où chercher l'instruction suivante (inutile si le CO est incrémenté).

Machine à 0/1/2/... opérandes (alias, machine à 0/1/2/... adresses):
machine dont les instructions ont au maximum 0/1/2/... opérandes.

Exemple de programmation de "A = E – (F / G)" avec un seul opérande:

```
LOAD F   //ACC <- F    (i.e., l'accum. reçoit le contenu de F)
DIV  G   //ACC <- F/G  (plus littéralement: ACC <- ACC / G)
STA  T1  //T1  <- F/G  (plus littéralement: T1  <- ACC)
LOAD E   //ACC <- E
SUB  T1  //ACC <- E – F/G  (plus littéralement: ACC <- ACC – T1)
STA  A   //A   <- E – F/G  (plus littéralement: A   <- ACC )
```

Exemple de programmation de "A = E - F/G" avec 0 opérande,
via une pile dans la MP:

```
LOAD E   //pile={E}
LOAD F   //pile={E,F}
LOAD G   //pile={E,F,G}
DIV      //pile={E,F/G}
SUB      //pile={E-F/G}
STA  A   //pile={}
```

CompilerExplorer vous permet de voir la traduction de code C/C++ dans divers assembleurs, avec différents compilateurs et options de compilation.

Obsolète en 2023 :si vous souhaitez explorer les arcanes d'un microprocesseur et d'un assembleur, utilisez le simulateur de microprocesseur (pour étudiants, sur Windows uniquement) à http://www.softwareforeducation.com/sms32v50/

Note: les fonctionnalités avancées d'un langage – e.g. cf. Higher-Order Perl – n'ont pas de traduction directe et sont les plus dures+importantes à apprendre.

4. Instructions et programmation - structure des instructions

Exemple de programmation de "i= 0; do { i++; } while (i != 10);"
avec plusieurs opérandes :

```
     STORE i, 0      //i= 0;
     LOAD R1, 10     //R1= 10; //R1 et R2 sont des registres
     LOAD R2, i      //R2= i;  //un registre est plus rapide
                               // à utiliser que la variable i
NI:  ADD R2, R2, 1   //do{ R2= R2 + 1;
     SUB R3, R1, R2  //    R3= R1 - R2;  //would be equivalent to
                                         // "R3= 10-i" if i was used
     JNZ NI          //  }while (R3 != 0); //Jump to NI if the result
                                          // of the last operation is Not Zero
```

Pour culture, voici quelques exemples de différents types d'adressage
avec plusieurs opérandes :

```
LOAD R4, R3       //R4 <- la valeur dans R3;    //adressage registre
LOAD R4, 3        //R4 <- 3;                    //adressage immédiat
LOAD R4, (0011)   //R4 <- la valeur à l'adresse 0011
                  // adressage direct
LOAD R4, (R3)     //R4 <- la valeur à l'adresse dans R3
                  // adressage indirect par registre
LOAD R4, @(0011) //R4 <- la valeur à l'adresse située à l'adresse 0011
                  // adressage indirect par mémoire
LOAD R4, (R3)100 //R4 <- la valeur à l'adresse "100 + contenu de R3"
                  // adressage avec déplacement
```

Voici quelques exemples de différents types d'adressage
avec un seul opérande (exemples utilisés en TD) :

```
LOAD 1000, IMM;    //ACC <- 1000   ("IMM" veut dire "immédiat")
LOAD 3000, XR1;    //<=>  LOAD 3000+1  <=> LOAD 5,IMM
                   //=>   ACC <- 5
LOAD 0, I;  //<=>   LOAD 1000   <=>   LOAD 0,IMM
            //=>   ACC <- 0 ;    en effet, "I" veut dire "indirect"
```

* Quelques raisons pour lesquelles une toute petite partie d'un système d'exploitation est généralement écrite en assembleur plutôt qu'en C.

4. Instructions et programmation - adressage mémoire

Soit ADR le nombre de bits dans le champs d'une adresse
(i.e., le nombre de bits qui peut être utilisé par un bus adresse), et 2^n la taille de la MP.

- Si ADR = n, la MP est accessible dans sa totalité; toutes les méthodes d'adressage peuvent être utilisées.

- Si ADR < n, ADR ne permet pas d'adresser toutes les positions de mémoire de la MP. Toutefois, si le registre de base a n bits,
  la MP peut être divisée en blocs de taille 2^n et être totalement accessible par "adressage basé":
  - l'adresse du 1er mot d'un de ces blocs est d'abord mise dans le "registre de base",
  - il ne reste plus qu'à transmettre sur le bus adresse la différence entre ce début de bloc et l'adresse à atteindre.

  Par exemple, si le registre d'adresse XR1 contient le nombre 6, l'instruction LOAD 3000,XR1 va chercher le contenu du mot mémoire à l'adresse 3006 et va mettre ce contenu dans l'accumulateur. Ceci sera détaillé en TD.

- Si ADR > n, ADR peut adresser plus de positions de mémoire qu'il n'en existe pas dans la MP. Ceci offre la possibilité de déborder sur d'autres mémoires (e.g., les disques) -> notion/techniques de "mémoire virtuelle".

5. Comparaison avec les réseaux neuronaux

Cerveau**:** environ 10^{12} neurones (~ portes logiques complexes pouvant "commuter" des centaines de fois par seconde, au mieux).
Chaque neurone peut être connecté à 10^4 neurones; différentes zones spécialisées (vue, ouïe, …). Énorme redondance/distribution de l'information.
Le cerveau a un **fonctionnement parallèle et statistique ->** c'est une machine non "programmable" mais, via son câblage (ses connections), il peut effectuer/découvrir/reconnaître des associations entre certaines informations (images, sons, concepts, ...) et peut ainsi apprendre de nouvelles associations, i.e., de nouvelles (combinaisons de) connections.
Ainsi, le cerveau est très efficace en reconnaissance (d'images, de sons, ...) mais très inefficace en application de procédures/règles et,
de plus, le cerveau commet beaucoup d'erreurs arithmétiques/logiques/...

> **Ordinateur classique**: 1 ou plusieurs microprocesseurs, chacun composé de millions/milliards de transistors (~ portes logiques simples pouvant commuter des milliards de fois par seconde). Il y a peu de redondance (la perte/modification d'un seul bit peut être fatale). Un ordinateur classique a un **fonctionnement procédural (et donc essentiellement séquentiel) ->** il n'apprend que s'il applique un programme d'apprentissage; il est peu efficace en reconnaissance (d'image, de son, ...) mais très efficace en application de procédures/règles; il ne commet pas d'erreurs d'application de programme (mais des bits peuvent se perdre ou le programme peut avoir des erreurs).

Réseau (classique) de neurones (artificiels): quelques dizaines/.../milliers de neurones artificiels (simplification de neurones naturels), chacun connecté à une ou plusieurs dizaines de neurones (cf. ces progrès matériels en 2022).
Un tel réseau a un **fonctionnement parallèle et statistique ->** utile pour la reconnaissance (d'images, sons, ...) -> mêmes problèmes que pour le cerveau.

QW

Modèle de language *de grande taille* : modèle statistique (proche de ceux de l'Apprentissage Profond [Deep Learning] et donc des réseaux neuronaux ; modélisant la distribution statistique de séquences de mots/lettres/... et donc, pour une séquence donnée, stockant différentes suites à cette séquence et leurs probabilités (cf. théorème de Bayes) ; majoritairement utilisé pour le traitement automatique des langues ou la reconnaissance automatique de la parole) modélisant des millions/milliards de "paramètres" (critères/charactéristiques utilisés pour représenter puis analyser une séquence donnée et donc *reconnaître* la meilleure suite à une séquence donnée), e.g. :

- BERT ("Bidirectional Encoder Representations from Transformers") de Google : 340 millions de paramètres en 2020.
- GPT-3 ("Generative Pre-trained Transformer 3") de OpenAI (lié à Microsoft entre autres): 175 milliards de paramètres en 2020 (apprentissage non-supervisé", essentiellement, sur une partie du contenu du WWW, pendant des mois). Il est utilisable via l'interface Web de ChatGPT ou son API (voir sa documentation, dont sa page "technique" dite "pour les chercheurs").
- MT-NLG ("Megatron-Turing Natural Language Generation") de Microsoft : 530 milliards de paramètres en 2021.

Ces modèles|réseaux statistiques ne stockent et n'exploitent pas des représentations de connaissances. I.e., ils ne "représentent" pas le monde réel via des formules logiques, ne font pas d'inférences logiques au sens classique (ils exploitent des statistiques, pas des règles logiques), et ne sont donc pas capable "d'expliquer au sens classique" leurs résultats. **Toutefois**, ils (au moins GPT) sont capables d'effectuer des tâches i) dont le résultat souvent *semble* correct ou cohérent, et ii) qui, pour un tel résultat, requièrent usuellement un raisonnement logique, e.g., répondre à des questions d'une manière qui semble cohérente, calculer, résumer un texte, traduire ou compléter des phrases, répondre correctement à des questions, etc. Ces outils peuvent donc être utilisés comme guides (générateurs de suggestions, e.g., de vérification) mais leur faire confiance est dangereux (bien plus qu'à des programmes classiques bien débogués ou dont le fonctionnement correct a été prouvé formellement), e.g., actuellement la plupart de ces outils statistiques ignorent ou ne traitent pas correctement les négations dans leurs textes sources (ils peuvent donc conseiller de faire le contraire de ce qu'il est souhaitable de faire). Étant "statistiques", ils imitent aussi de mauvaises habitudes [1, 2, 3] et ont des "hallucinations". Il y a différentes façons de combiner des modèles|réseaux statistiques avec des outils classiques [4, 5, …].

6. TD pour cette partie 2 du cours d'A.O.

I. Exercices sur la partie "mémoires"

1. Listez les différences entre des mémoires volatile, dynamique et statique.
2. On considère une mémoire centrale de 2 MBytes où chaque octet est adressable séparément (→ adresse = "nombre d'octets depuis 0") :
   a) Calculer l'adresse, en octal, du sixième élément d'un tableau dont l'adresse du premier élément est 77_8, et dont tous les éléments sont composés de 16 bits ;
   b) calculer, en décimal, le nombre d'octets précédant l'adresse 77_8 ;
   c) calculer la taille de cette mémoire en l'exprimant en mots de 16 bits puis en mots de 32 bits.
3. Si le registre d'adresse d'une mémoire comporte 32 bits, calculer:
   a) le nombre de mots adressables si 1 mot = 1 byte;
   b) la plus haute adresse possible pour ces mots de 1 byte;
   c) le nombre de mots adressables si 1 mot = 32 bits;
   d) la plus haute adresse possible pour ces mots de 32 bits.
4. Répondre par "oui" ou "non" aux questions suivantes concernant une recherche dans une mémoire associative (note : souvent, quand une partie d'une question n'a pas de sens, la bonne réponse est "non") :
   a) la valeur à la position X est-elle supérieure à celle à la position Y ?
   b) y a-t-il une position contenant la valeur A ?
   c) la valeur C est-elle stockée à la position X ?
   d) la valeur de la position X est-elle égale a la valeur de la position Y ?
5. Répondre par oui ou non aux questions suivantes concernant une mémoire entrelacée (donc divisée en blocs) :
   a) il y a 1 registre d'adresse et 1 registre mot-mémoire pour toute la mémoire ?
   b) il y a 1 registre d'adresse et 1 registre mot-mémoire par bloc de mémoire ?
   c) cette mémoire entrelacée est forcément utilisée comme antémémoire ?
   d) chacun des blocs contient un programme différent ?
   e) les instructions d'un même programme sont réparties sur plusieurs blocs ?

6. Calcul du temps moyen de transfert en mémoire centrale (M.C.) d'un fichier séquentiel stocké sur disque. Les échanges entre l'unité de disque et la M.C. s'effectuent par l'intermédiaire d'un canal.
   Une zone de mémoire tampon, associée au disque, reçoit les caractères lus sur le disque. Le transfert vers la M.C. s'effectue quand la zone tampon est pleine ou quand le fichier a été entièrement lu.
   Les caractéristiques de l'unité de disque, du fichier et du canal sont les suivantes: 1) le temps moyen de positionnement de la tête de lecture sur une piste est de 20 ms ; 2) vitesse de rotation du disque : 6000 tours par minute ; 3) chaque secteur contient 1024 bits dont 64 sont réservés pour un pointeur vers le secteur suivant (du fichier) ; 4) le disque est composé de 128 cylindres ; 5) une piste comporte 32 secteurs; 6) tous les secteurs d'un fichier sont répartis dans un même cylindre ; 7) le fichier contient 1500 caractères ; 8) chaque caractère est codé sur 7 bits + 1 bit de parité ;
   9) la vitesse de transfert du canal est de 10 MBytes/seconde ;
   10) la taille de la zone tampon est de 4096 caractères. Déterminer :
   a) le temps moyen pour lire un secteur (en supposant que la tête de lecture soit sur la bonne piste);
   b) le nombre de secteurs nécessaires au stockage du fichier;
   c) le temps moyen de transfert de ce fichier entre le disque et la M.C. ;
   d) quels sont les facteurs limitatifs de ce transfert.

7. Soit une mémoire centrale de 1 Mmots de 32 bits réalisée avec des puces de 16Kbits. Cette mémoire peut être organisée suivant plusieurs principes. Considérez les trois principes suivants :
   - un bit par puce : un mot est constitué de 32*1 bit provenant chacun d'une puce différente, donc 32 puces sont nécessaires pour réaliser un mot;
   - 16 bits par puce : un mot est constituée de 2*16 bits; deux puces sont donc nécessaires pour former un mot de 32 bits;
   - 32 bits par puce : un mot est constitué de 1*32 bits.

   Calculer :
   a) le nombre de bits nécessaires pour adresser toute la mémoire dans chacun des cas;
   b) le nombre de pattes (de chaque puce) utilisées pour l'adressage et pour les données dans chacun des cas.

II. Exercices sur les parties "bus", "registres" et "CPU"

1. Qu'est-ce qu'un bus ?

2. Quelles sont les différences fondamentales entre les langages machine et les langages évolués ?

3. Décrire les cycles de recherche et d'exécution d'une instruction.

4. Quelle est la différence entre un séquenceur câblé et un séquenceur micro-programmé ?

5. Réaliser un programme qui calcule l'expression suivante dans une machine à 1 adresse et dans une machine à 0 adresse :

   R = A - (B / (C+D)) + (E*F) - G

6. Etant donné les contenus des registres et des mémoires suivants:

   XR1 = 1; XR2 = 2; B1 = 1000; B2 = 2000;

   ^1000 = 0; //en mémoire, à l'adresse 1000, il y a le nombre 0

   ^1001 = 1; ^2000 = 2; ^2001 = 3; ^3000 = 4; ^3001 = 5; ^0 = 1000;

   a) Trouver, comme dans les 3 premiers exemples ci-dessous, la valeur qui est mise dans l'accumulateur (le registre ACC) à la fin des 4 dernières opérations ci-dessous:

   LOAD 1000, IMM; // ACC <- 1000 ("IMM" veut dire "immédiat")

   LOAD 3000, XR1; // <=> LOAD 3000+1 <=> LOAD 5,IMM

   // => ACC <- 5

   LOAD 0, I; // <=> LOAD 1000 //car "I" veut dire "indirect"

   // <=> LOAD 0,IMM // => ACC <- 0

   LOAD 1000, B1; LOAD 1, B2; LOAD 999, XR2; LOAD 0,I

   b) Quelle est la valeur de F après l'exécution du programme suivant :

   LOAD 3000; ADD 2000,XR1; SUB 2001,B1; MPY 1001,B2 ;

   DIV 2000; ADD 1000,IMM; SUB 0,I,XR1; STORE F

7. Soit une machine dotée d'une mémoire centrale de 512 Kmots de 32 bits. Sachant que l'instruction-type occupe un mot-mémoire, quelles tailles proposeriez-vous pour les registres CO et RI ?

7. Exemples de questions de QCM pour cette partie 2

Voyez http://www.phmartin.info/cours/ao/ pour plus d'exemples de questions.

Un exemple d'unité d'entrée/sortie est ...
A) un bus B) un canal C) un DMA D) les 3 dernières réponses
E) aucune des 4 dernières réponses

Un exemple d'unité périphérique est ...
A) une mémoire "permanente" B) un disque C) un modem
D) les 3 dernières réponses E) aucune des 4 dernières réponses

Qu'est-ce que le "temps d'accès mémoire" ?
A) le temps minimal entre 2 accès mémoire
B) le temps pour la lecture/écriture d'un mot mémoire
C) le temps de lecture/écriture d'un registre mémoire
D) les 3 dernières réponses E) aucune des 4 dernières réponses

En règle générale, plus un composant est éloigné du processeur ...
A) plus sa capacité est grande B) plus son temps d'accès est grand
C) moins son prix est grand D) les 3 dernières réponses E) aucune des 4

Le bus local ... A) est le bus interne
B) permet l'ajout de périphériques ou de cartes d'extension;
C) est un bus de commandes D) les 3 dernières réponses E) aucune des 4

Un cycle instruction ...
A) est suivi d'un cycle d'exécution B) est précédé d'un cycle de recherche
C) est un cycle machine D) les 3 dernières réponses E) aucune des 4

Dans la microprogrammation horizontale ...
A) plusieurs unités fonctionnelles sont contrôlées par une seule microinstruction
B) une microinstruction est courte et correspond à une commande
C) une microinstruction est longue, chaque bit correspond à une commande
D) chaque microinstruction a elle-même un code opération
E) aucune des 4 dernières réponses

Un interpréteur ...

A) traduit (ou conduit à une traduction de) un code source en langage machine
B) exécute certaines parties du code C) vérifie la syntaxe de certaines parties
D) les 3 dernières réponses E) aucune des 4 dernières réponses

Pour effectuer A= (B / C) - D, le programme ci-dessous est ...

```
LOAD D
LOAD C
LOAD B
DIV
SUB
STA  A
```

A) entièrement correct sur une machine à 0 opérande
B) entièrement correct sur une machine à 1 opérande
C) syntaxiquement correct sur une machine à 0 opérande
mais ce n'est pas l'expression ci-dessus qui est effectuée
D) syntaxiquement correct sur une machine à 1 opérande
mais ce n'est pas l'expression ci-dessus qui est effectuée
E) aucune des 4 dernières réponses

Si N est le nombre de bits d'un registre d'adresse, la taille de la mémoire qu'adresse ce registre est *nécessairement* ...

A) 2*N bits B) (2^N - 1) bits C) 2^N bits D) 2^X bits, avec X= 2^N E) aucune

Si le registre d'adresse d'une mémoire comporte 8 bits, quelle est la plus haute adresse possible avec des mots de 16 bits ?

A) 255 B) 256 C) 1023 D) 1024 E) aucune des 4 dernières réponses

Si la mémoire principale est plus grande que ce que peut adresser un registre d'adresse, le programmeur peut-il néanmoins adresser toute la mémoire principale ?

A) non B) oui, via toutes les méthodes d'adressage
C) oui, via une combinaison d'adressages immédiats
D) oui, via un adressage basé E) oui, via des techniques de "mémoire virtuelle"

Par rapport à un ordinateur classique, un réseau de neurones ...

A) est plus efficace pour appliquer des procédures
B) a plus de redondances et de parallélisme
C) doit suivre le rythme d'une horloge
D) les 3 dernières réponses E) aucune des 4 dernières réponses