UNIVERSITE DE GABES
École Nationale d'Ingénieurs de Gabès

ENIG

ENIG

Département GCR

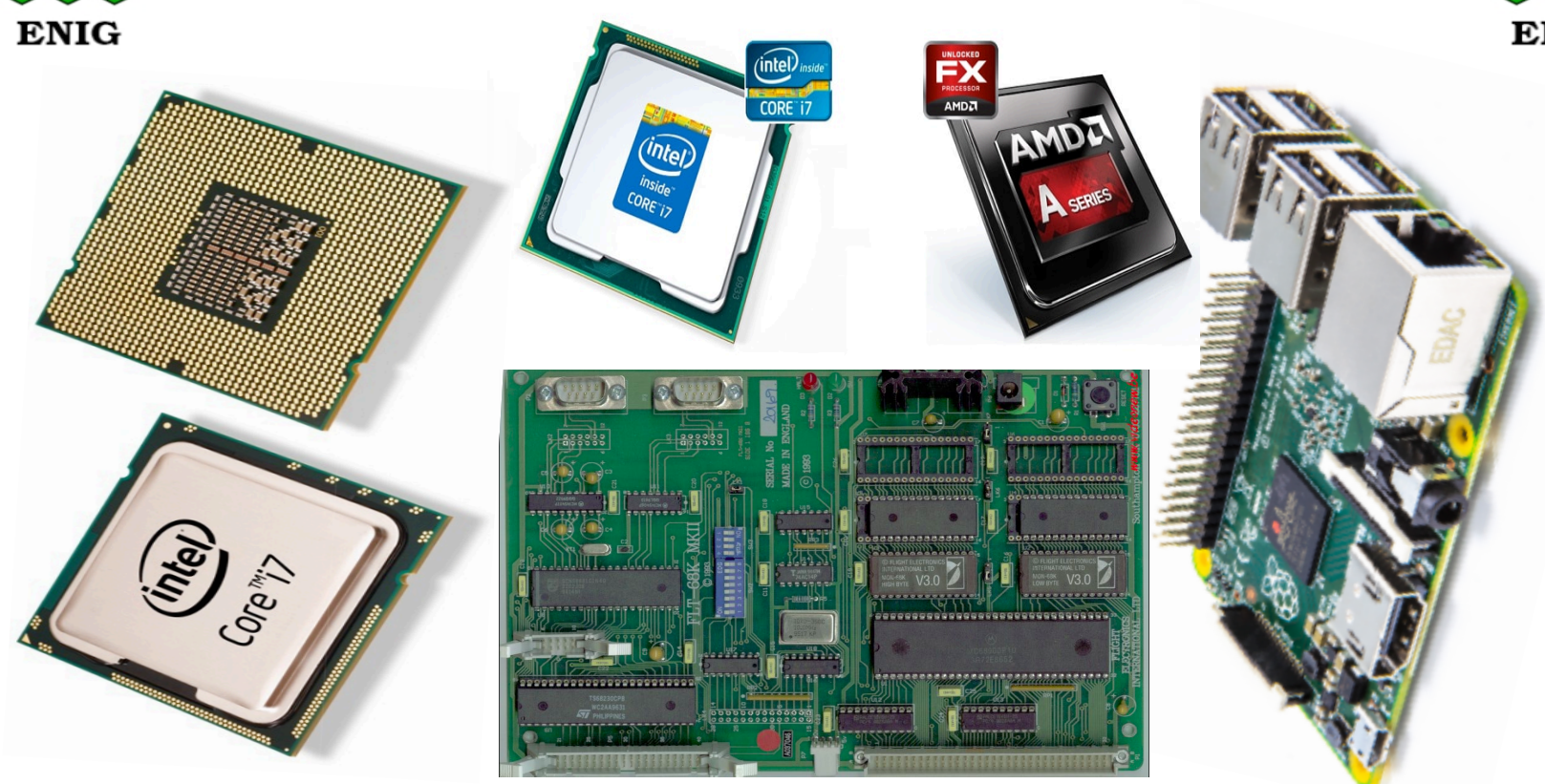
Support de cours

Architecture des ordinateurs et microprocesseurs

Elaboré par

Hamdi Ridha

A.U. 2020-2021



Architecture des ordinateurs et microprocesseurs

A.U. 2020-2021



Architecture des ordinateurs et microprocesseurs

Ch01 : Généralités
- Représentation de l'information
- Circulation de l'information dans un calculateur

Ch02 : Architecture d'un ordinateur
- Les différents composants d'une carte mère
- Les microprocesseurs
- Périphériques et mémoires

Ch03 : Microprocesseurs : Etude détaillée d'un exemple (le 68000 de Motorola)
- Organisation interne du 68000
- Jeux d'instructions et modes d'adressage
- Interfaces d'entrées-sorties (étude d'un exemple : le 68230)

Ch04 : Structure d'un systèmes d'exploitation
- Structure d'un OS
- Noyau et appels systèmes
- Chaine de compilation et d'exécution

Ch05 : Gestion de mémoire
- Mémoire virtuelle et MMU
- Segmentation et pagination

Représentation de l'information

On veut stocker

- des entiers positifs (12, 7850, ...)
- des entiers négatifs (-3, ...)
- des caractères ('A', 'r', ';', ...)
- des chaînes de caractères ("Hello", ...)
- des réels (7.873, ...)
- des instructions
- ...

Mais la mémoire
ne contient que des bits (0 ou 1)!

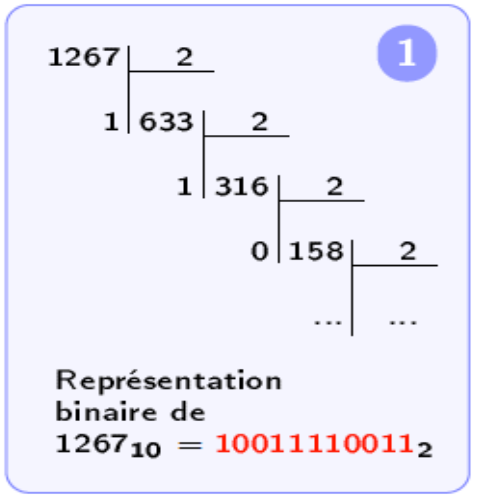


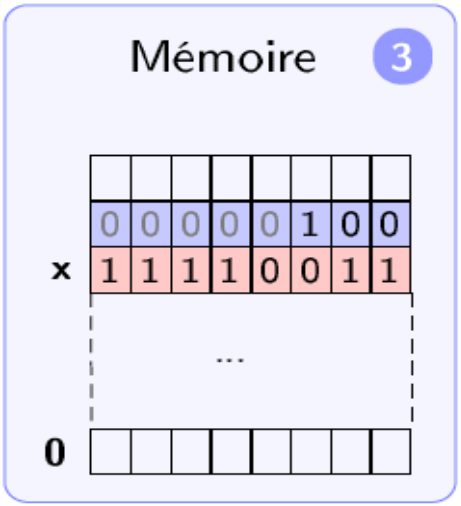
Tout coder sous forme d'entiers positifs en binaire

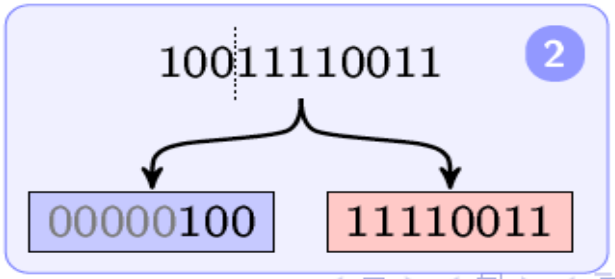
Représentation de l'information

Méthode pour stocker un **entier positif (non-signé)** en mémoire:

1. Représentation du nombre en binaire
2. Découpage de cette représentation en octets
3. Stockage de chaque octet consécutivement







Représentation de l'information

Cas des entiers signés

Complément à 1:
Le bit de poids fort correspond au signe:

- 0 = positif
- 1 = négatif

Le complément à 1 est obtenu en complémentant bit à bit sa valeur absolue avec 1

Exemple: représentation de -23_{10} ?

- $|-23_{10}| = 23_{10} = 00010111_2$
- Complément à 1 de $|-23_{10}|$: (inversion des bits)

$$\begin{array}{rr} & 11111111 \\ - & 00010111 \\ \hline = & 11101000 \end{array}$$

– Deux représentations pour 0: 00000000_2 et 11111111_2
– Plage de nombres représentés sur 8 bits: -127 à $+127$

Représentation de l'information

Cas des entiers signés

Complément à 2:

Le bit de poids fort correspond au signe:

- 0 = positif
- 1 = négatif

Le complément à 2 est obtenu en ajoutant 1 au complément à 1 de sa valeur absolue (et inversement)

Exemple: représentation de -23_{10} ?

- $|-23_{10}| = 23_{10} = 00010111_2$
- Complément à 1 de $|-23_{10}| = 11101000_2$
- Ajout de 1: $11101000_2 + 1 = 11101001_2$

– Une seule représentation pour 0: 00000000_2

– Plage de nombres représentés sur 8 bits: -128 à $+127$

Représentation de l'information

Cas des caractères

Plusieurs formats de représentation binaire:

- EBCDIC (*Extended Binary-Coded Decimal Interchange Code*)
  - Représentation sur 8 bits (256 caractères possibles)
  - Utilisé autrefois sur les mainframes IBM
- ASCII (*American Standard Code for Information Interchange*)
  - représentation sur 7 bits (pas d'accents)
  - ASCII étendu : sur 8 bits (mais pas normalisé) (ex: OEM, ANSI)
- Unicode : encodage sur 16 bits (65536 possibilités) pour représenter tous les caractères de toutes les langues

Représentation de l'information

Cas des fractionnaires

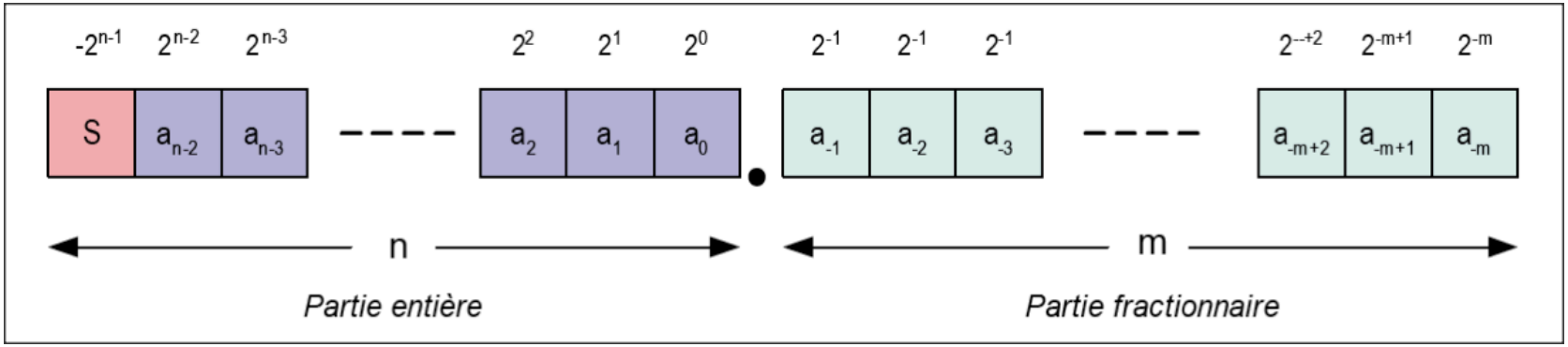
Représentation en virgule fixe

En se limitant à la représentation binaire en complément à 2, les nombres fractionnaires peuvent être représentés par la relation

$$A = -2^{n-1}S + \sum_{i=-m}^{n-2} a_i 2^i$$

avec n : nombre de bits pour exprimer la partie entière du nombre

m : nombre de bits pour exprimer la partie fractionnaire du nombre

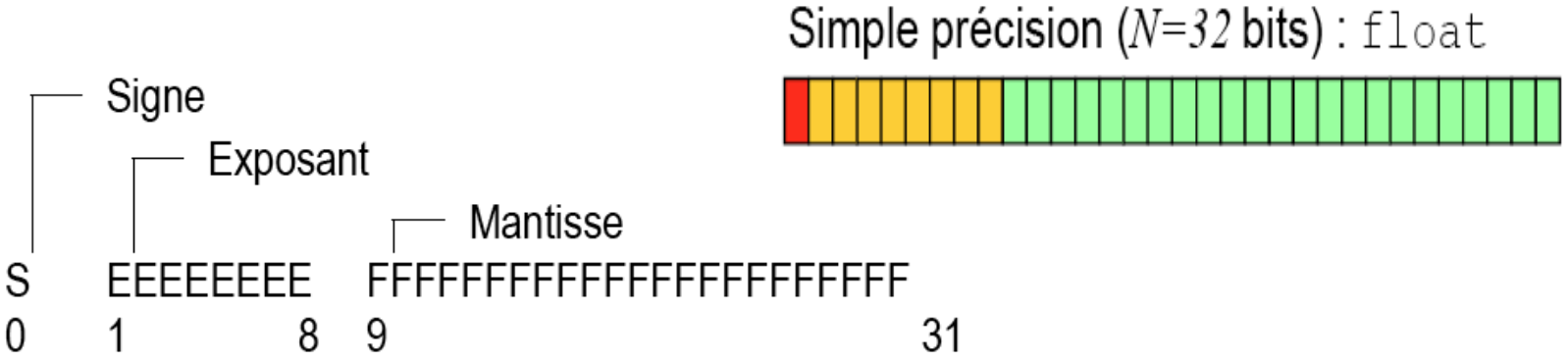


Représentation de l'information

Cas des fractionnaires

Représentation en virgule fixe - Simple précision:

Le standard IEEE 754 pour les nombres en virgules flottantes en simple précision est de 32 bits. Le premier bit S (le plus à gauche) représente le signe, les 8 suivants (E) l'exposant et les 23 derniers (F) la représentation fractionnaire du nombre en format 0.23, appelé mantisse



Représentation de l'information

Cas des fractionnaires

Représentation en virgule fixe - Simple précision:

La valeur V représentée par le mot codé en 32 bits peut être déterminé de la manière suivante :

- E=255 F≠0 V=NaN : *le résultat n'est pas un nombre*
- E=255 F=0 S=1 V= – Infinity
- E=255 F=0 S=0 V= + Infinity
- 0<E<255 $V=(-1)^S 2^{(E-127)} \cdot (1.F)$

  où (1.F) est prévu pour représenter un nombre binaire fractionnaire 0.23 précédé d'un 1 soit un format fractionnaire global de 2.23 avec le signe toujours positif (0) et le bit de poids fort égal à 1, soit 01.ϕϕϕϕϕϕϕ…

- E=0 F≠0 $V=(-1)^S 2^{(-126)} \cdot (0.F)$

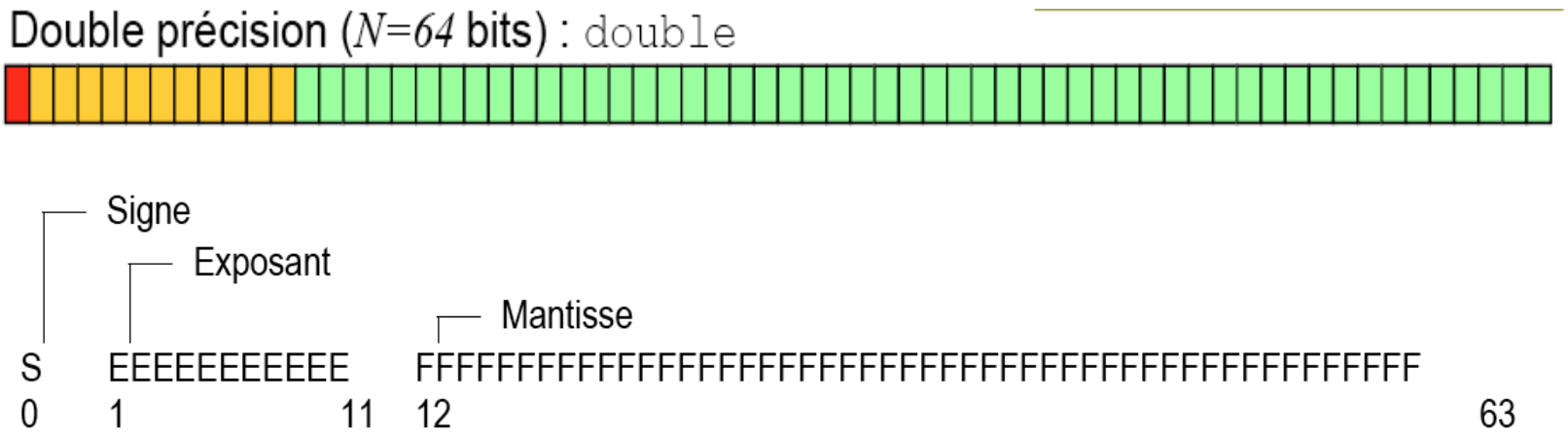
  V correspond à une valeur non normalisée
- E=0 F=0 S=1 V= – 0
- E=0 F=0 S=0 V= + 0

Représentation de l'information

Cas des fractionnaires

Représentation en virgule fixe - Double précision:

La virgule flottante double précision, selon le standard IEEE est constituée d'un mot de 64 bits. Le premier bit S (le plus à gauche) représente le signe, les 11 suivants (E) l'exposant et les 52 derniers (F) la représentation fractionnaire du nombre en format 0.52, appelée mantisse

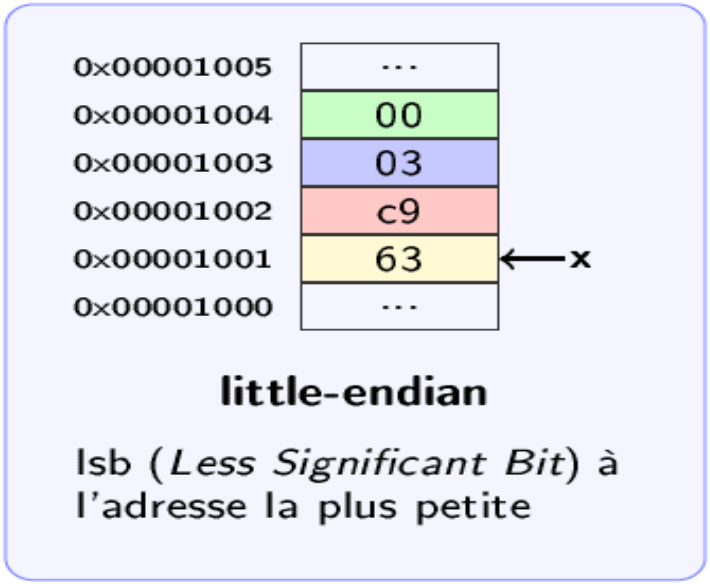


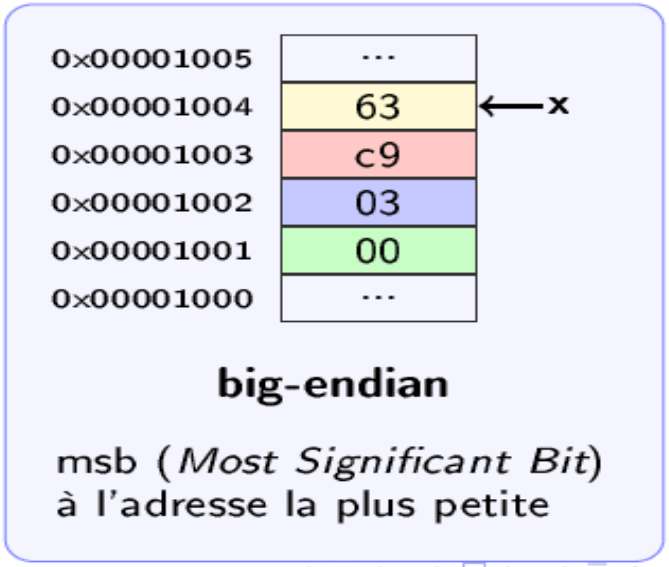
Représentation de l'information

Ordre d' "empilement" lorsque stockage sur plusieurs octets ?
Exemple:

$$248163_{10} = 11|11001001|01100011_2$$

00	03	C9	63





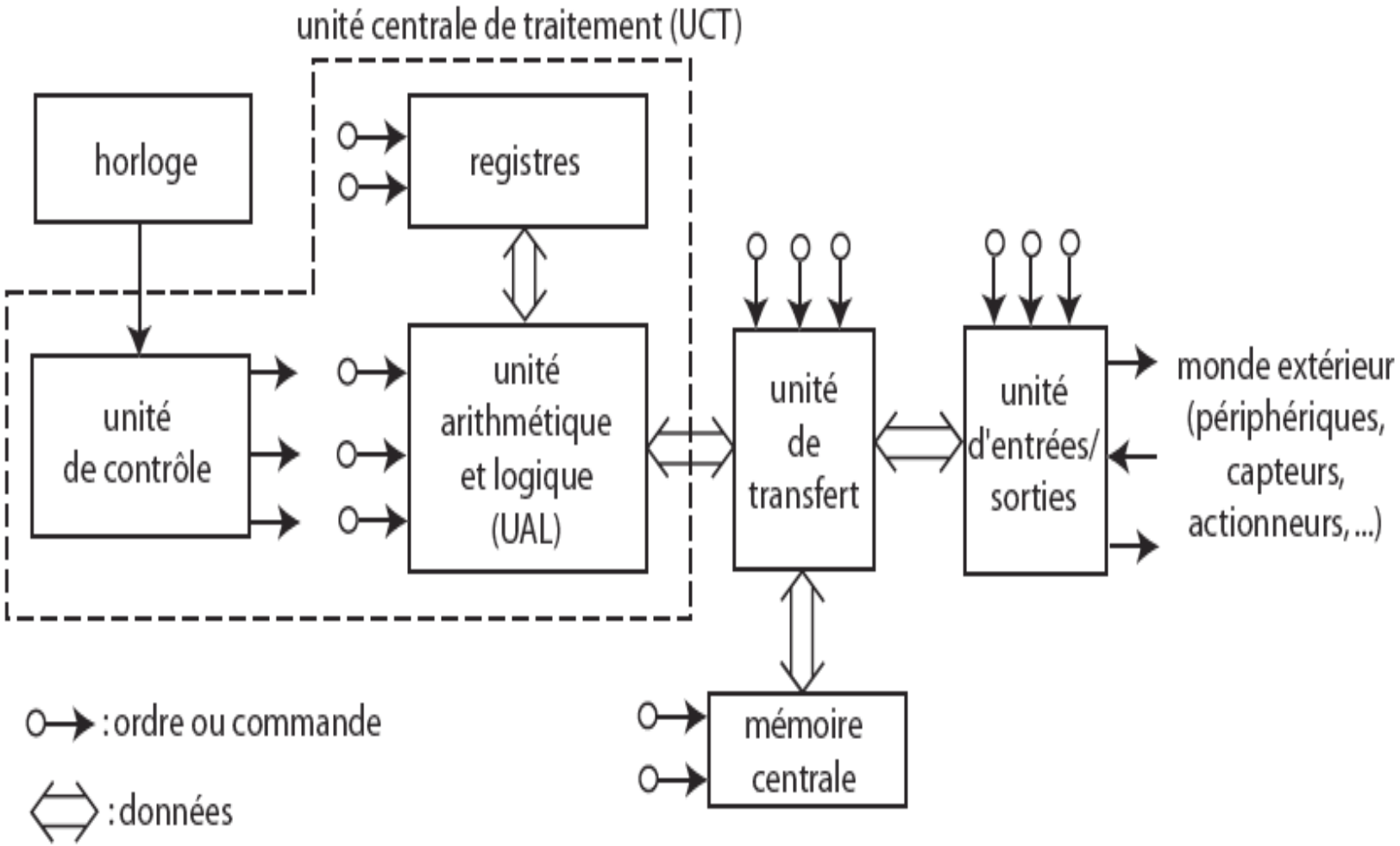
GÉNÉRALITÉS

Les unités d'évaluation de la capacité mémoire ou de flux binaire

- **Ko :** Un KiloOctet = 2^{10} octets =1024 octets soit 1024x8 = 8192 bits
- **Mo :** Un MegaOctet = 2^{20} octets = 1024 Ko soit 1024x1024 = 1 048 576 octets
- **Go :** Un Go = 1024 Mo soit
- **To :** Un To = 1024 Go

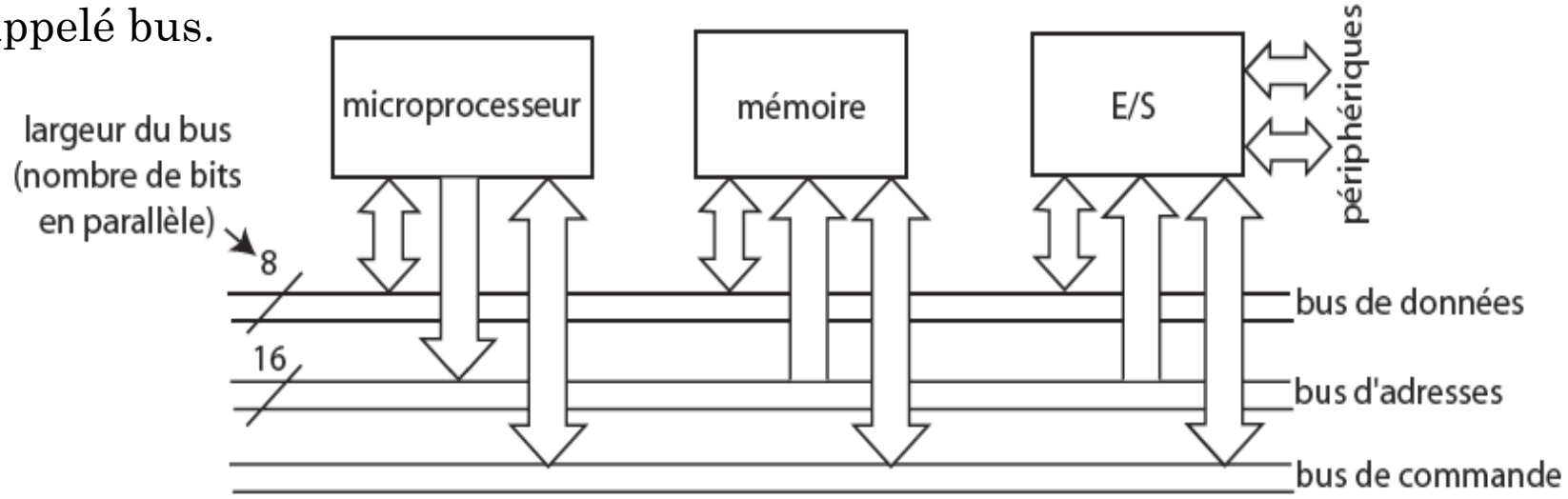
	bit	octet	Ko	Mo	Go	To
bit	1					
octet	8	1				
Ko	8 192	1 024	1			
Mo	8 388 608	1 048 576	1 024	1		
Go	8 589 934 592	1 073 741 824	1 048 576	1 024	1	
To	8 796 093 022 208	1 099 511 627 776	1 073 741 824	1 048 576	1 024	1

Structure d'un calculateur



Circulation de l'information dans un calculateur

Le microprocesseur échange des informations avec la mémoire et l'unité d'E/S, sous forme de mots binaires, au moyen d'un ensemble de connexions appelé bus.



Bus d'adresses : permet au microprocesseur de spécifier l'adresse de la case mémoire à lire ou à écrire

Bus de données : permet les transferts entre le microprocesseur et la mémoire ou les E/S

Bus de commande : transmet les ordres de lecture et d'écriture de la mémoire et des E/S.

GÉNÉRALITÉS

Un microprocesseur se présente sous la forme d'un circuit intégré muni d'un nombre généralement important de broches.

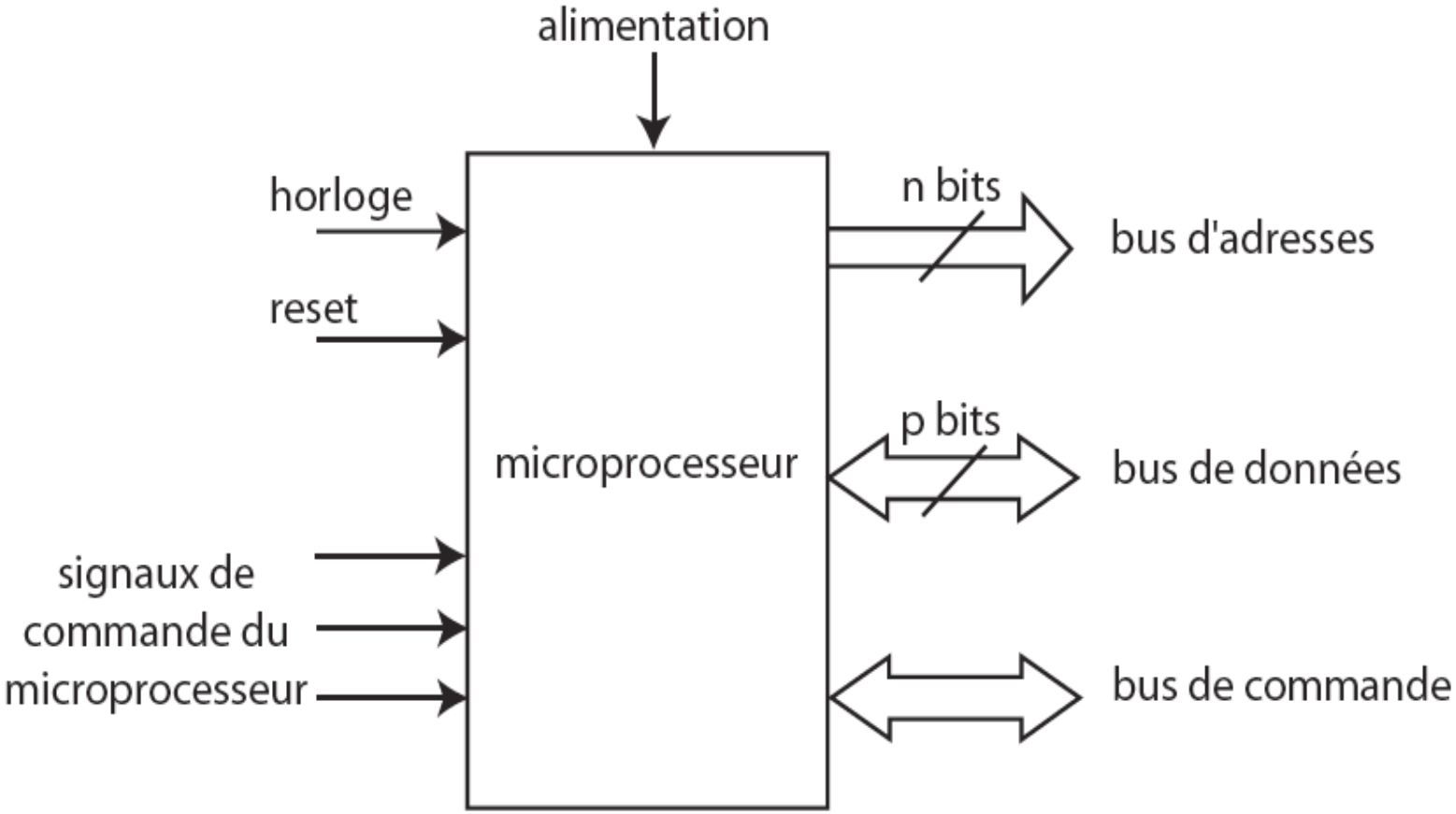
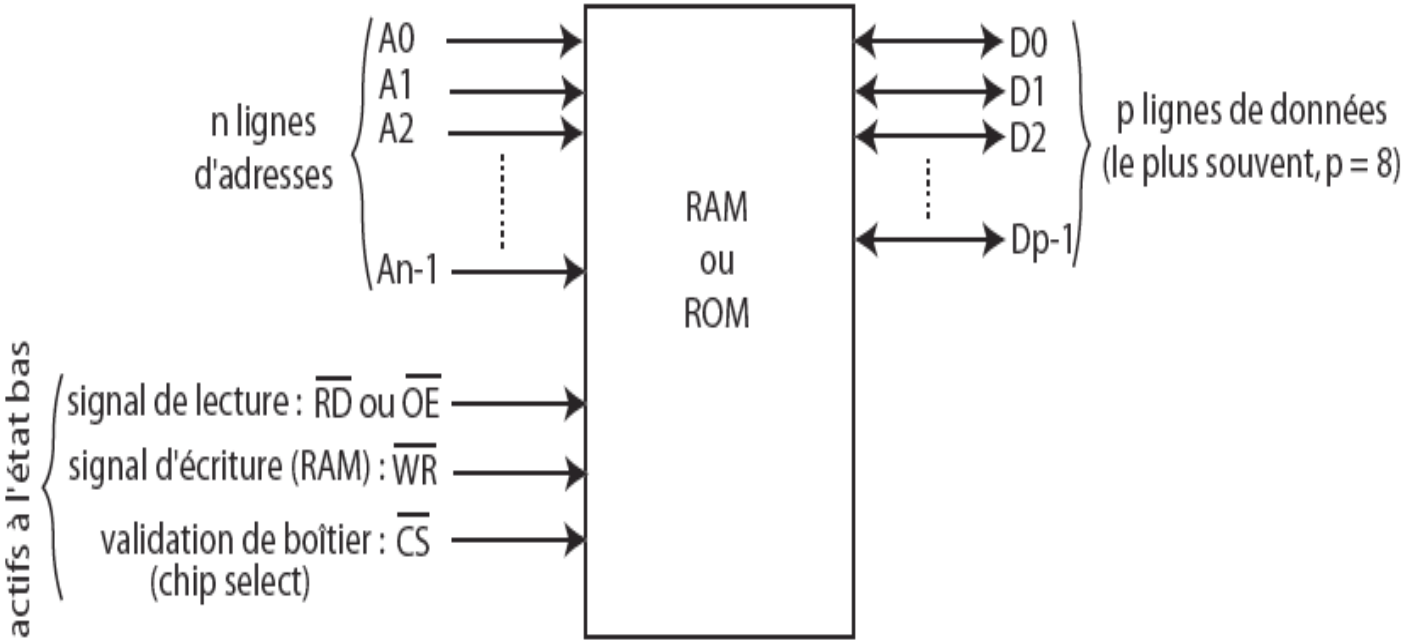
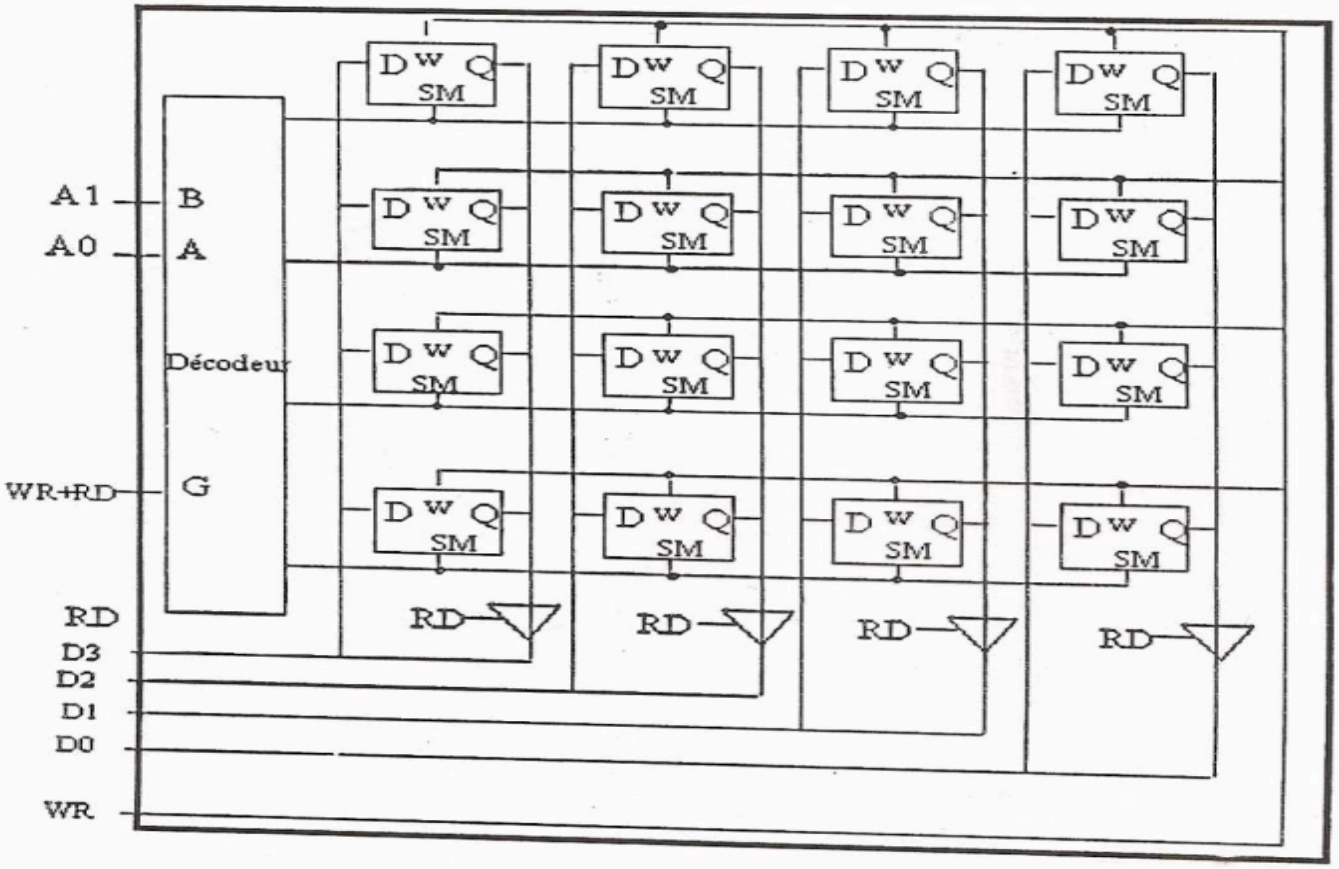


Schéma fonctionnel d'une mémoire

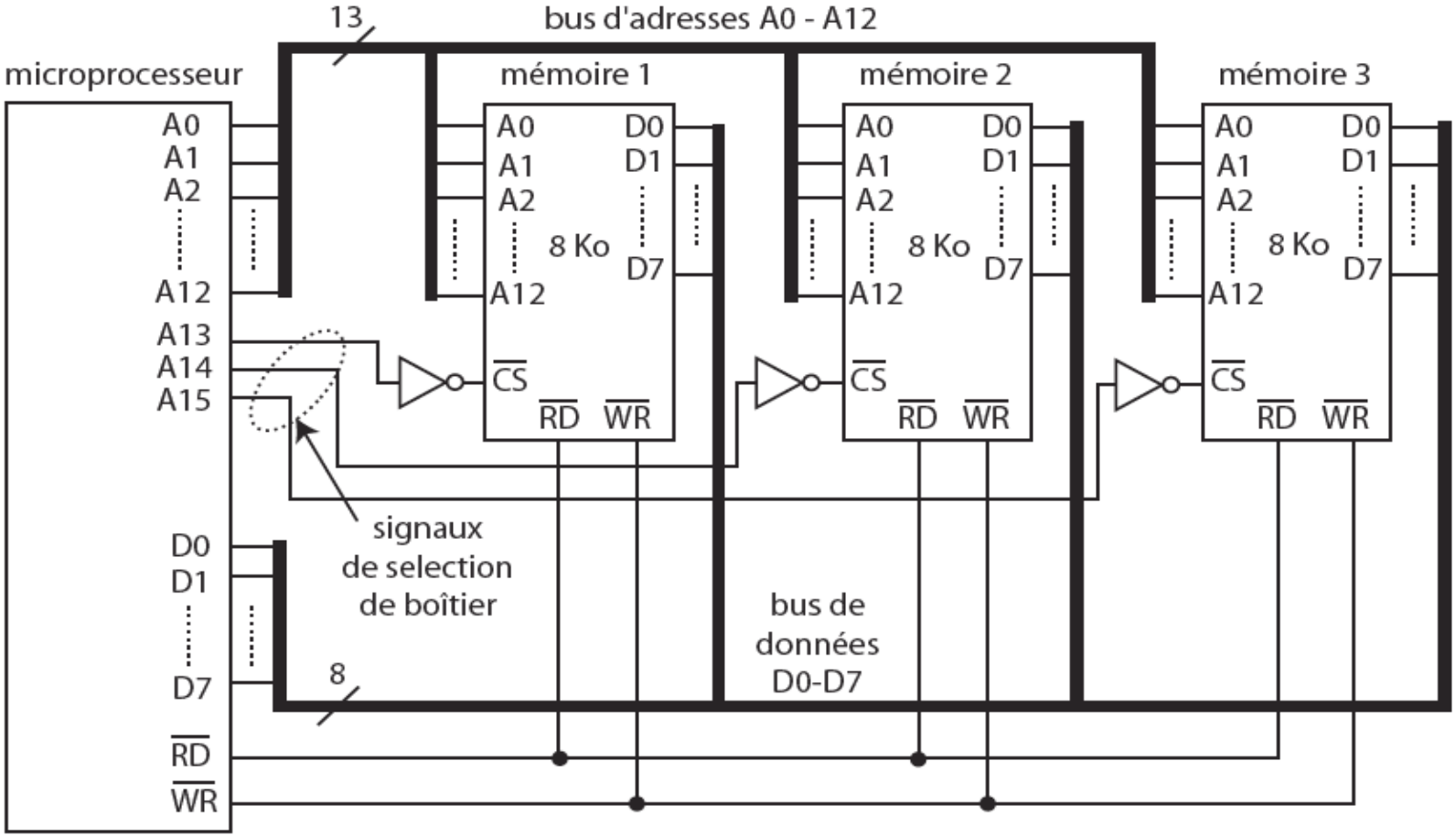


Capacité de la mémoire : n lignes d'adresses permettent d'adresser 2^n cases mémoire

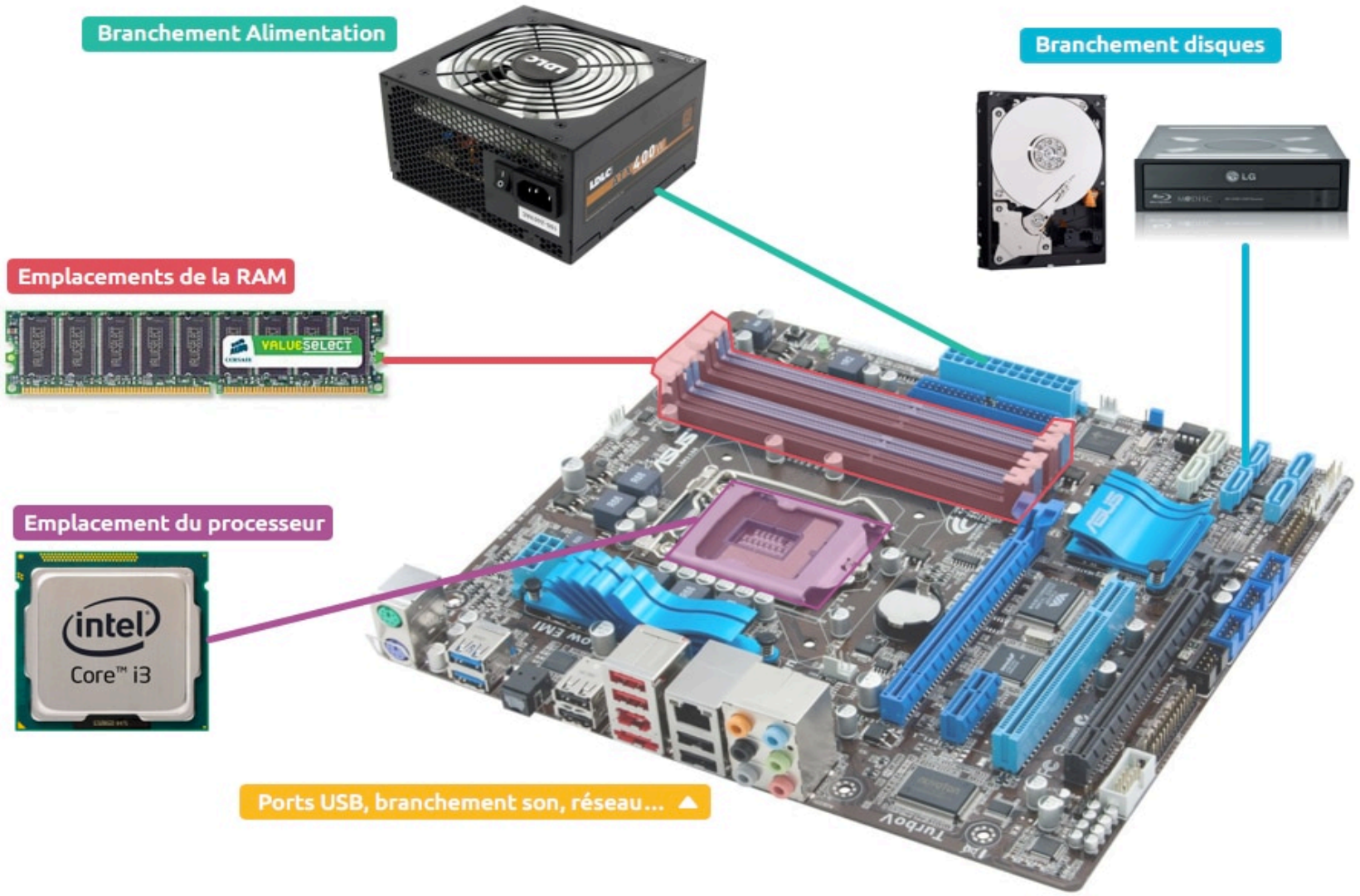
Fonction de mémorisation -exemple



Interfaçage microprocesseur/mémoire



ARCHITECTURE D'UN ORDINATEUR



ARCHITECTURE D'UN ORDINATEUR

La carte mère

La carte mère est l'un des éléments essentiels d'un ordinateur. Elle assure la connexion physique des différents composants (processeur, mémoire, carte d'entrées/sorties, ...) par l'intermédiaire de différents bus (adresses, données et commande).
On retrouve toujours sur une carte mère :

Le chipset : c'est une interface d'entrée/sortie. Elle est constituée par un jeu de plusieurs composants chargé de gérer la communication entre le microprocesseur et les périphériques. C'est le lien entre les différents bus de la carte mère.
le chipset est généralement composé de deux composants baptisé Pont Nord et Pont Sud.
Le pont Nord s'occupe d'interfacer le microprocesseur avec les périphériques rapides (mémoire et carte graphique) nécessitant une bande passante élevée alors que le pont sud s'occupe d'interfacer le microprocesseur avec les périphériques plus lents (disque dur, CDROM, lecteur de disquette, réseau, etc…).

ARCHITECTURE D'UN ORDINATEUR

Le BIOS (Basic Input Ouput Service) : c'est un programme responsable de la gestion du matériel : clavier, écran, disques durs, liaisons séries et parallèles, etc... Il est sauvegardé dans une mémoire morte (EEPROM) et agit comme une interface entre le système d'exploitation et le matériel.

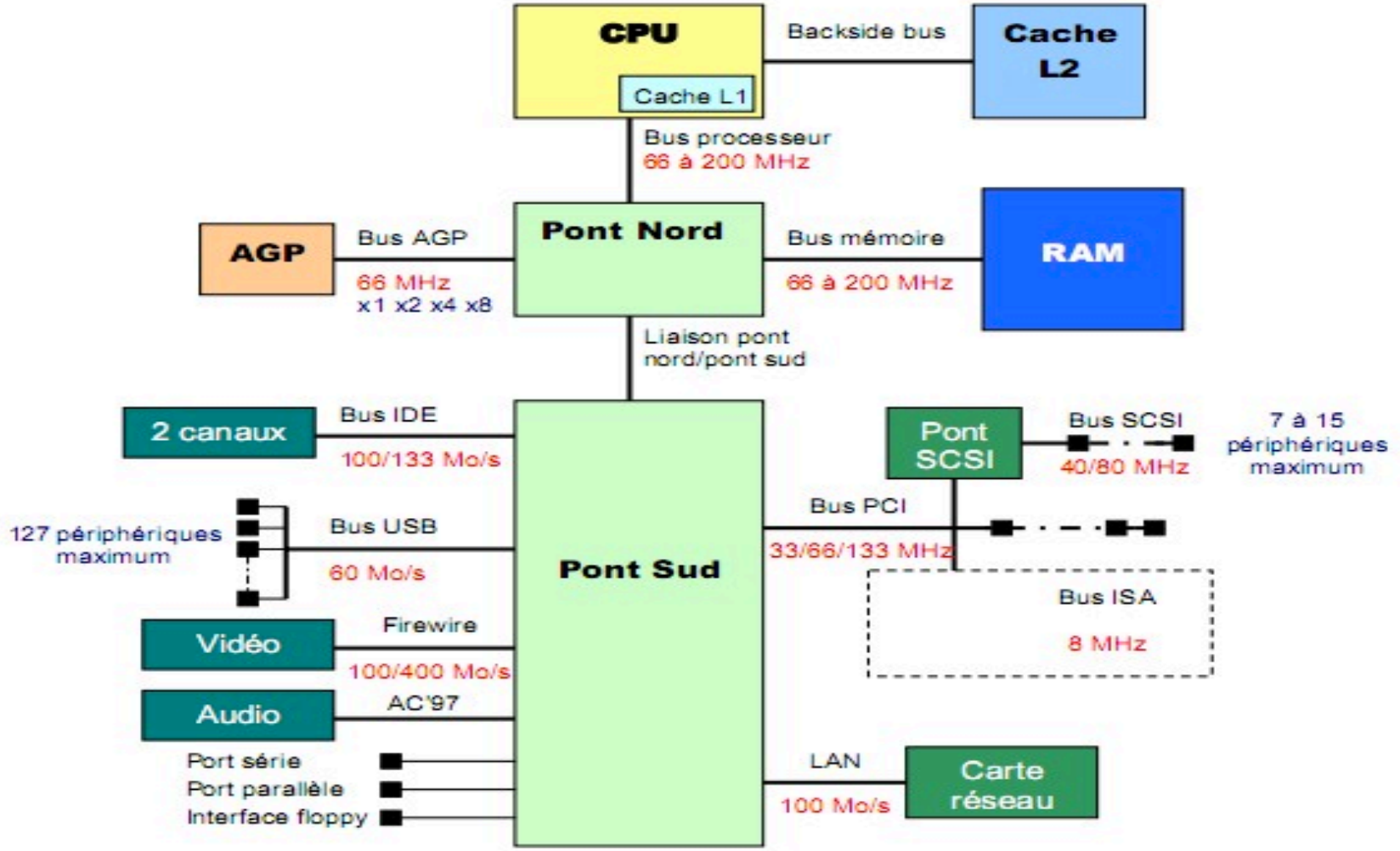
L'horloge : elle permet de cadencer le traitement des instructions par le microprocesseur ou la transmission des informations sur les différents bus.

Les ports de connexion : ils permettent de connecter des périphériques sur les différents bus de la carte mère. Il existe des ports « internes » pour connecter des cartes d'extension (PCI, ISA, AGP) ou des périphériques de stockage (SCSI, IDE, Serial ATA) et des ports « externes » pour connecter d'autres périphériques (série, parallèle, USB, firewire, etc …)

Le socket : c'est le nom du connecteur destiner au microprocesseur. Il détermine le type de microprocesseur que l'on peut connecter.

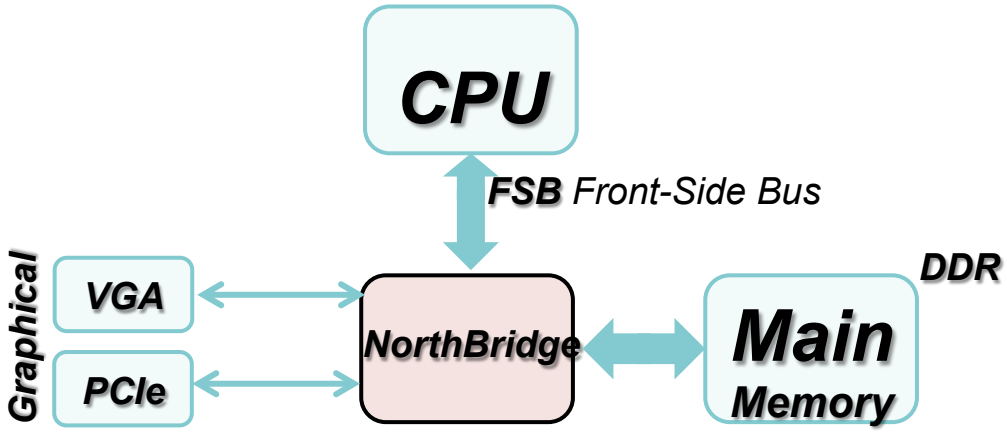
ARCHITECTURE D'UN ORDINATEUR

Architecture d'une carte mère



Architecture d'une carte mère

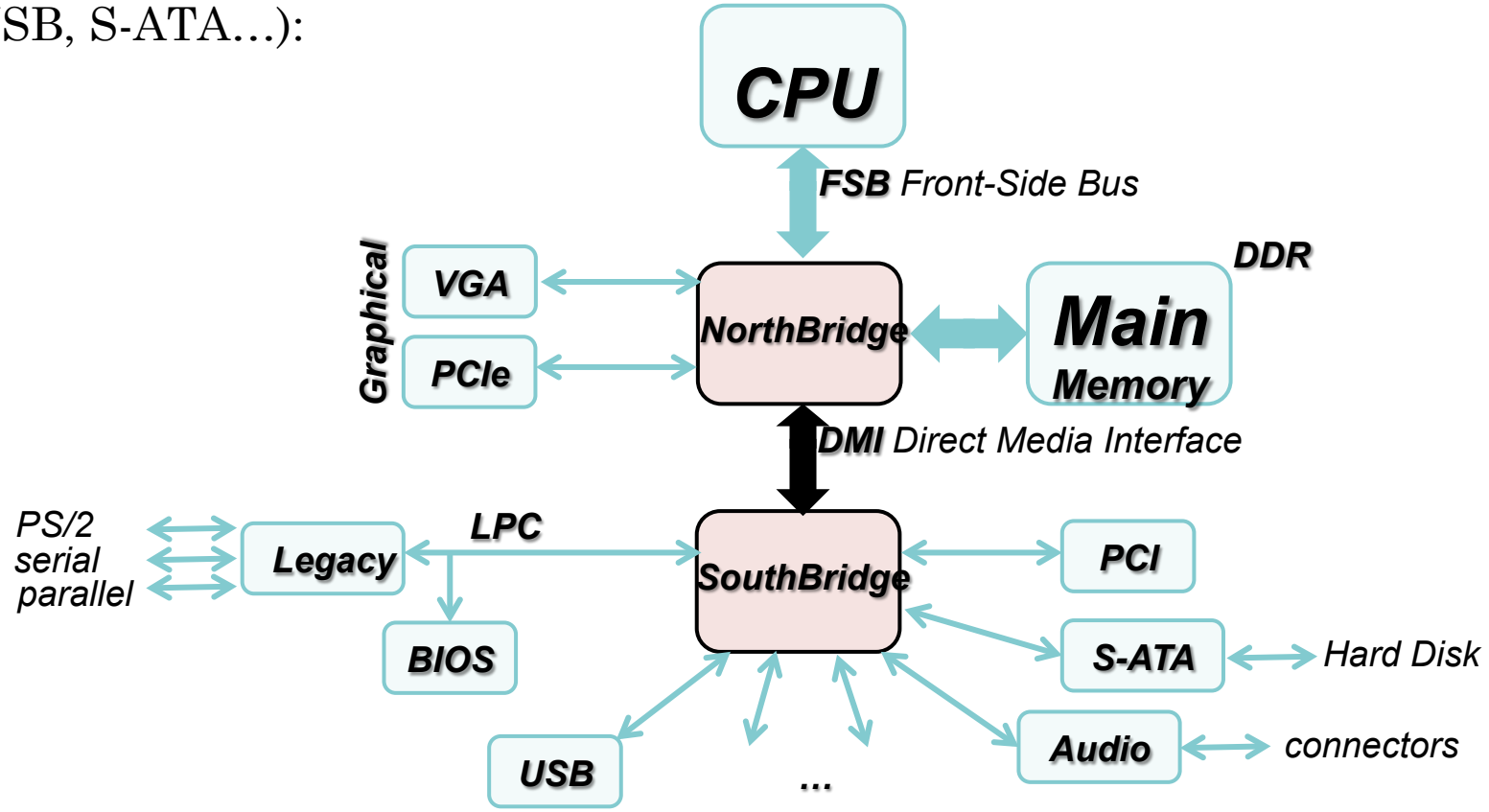
Le **NorthBridge (Pont Nord)** est un composant matériel servant de passerelle entre CPU et fonctions matérielles rapides (mémoire principale et vidéo):



Les fonctions interfacées étant exigeantes en bande passante, les bus de communications utilisés sont des bus parallèles relativement larges (DRR bus, FSB, PCIe x16).

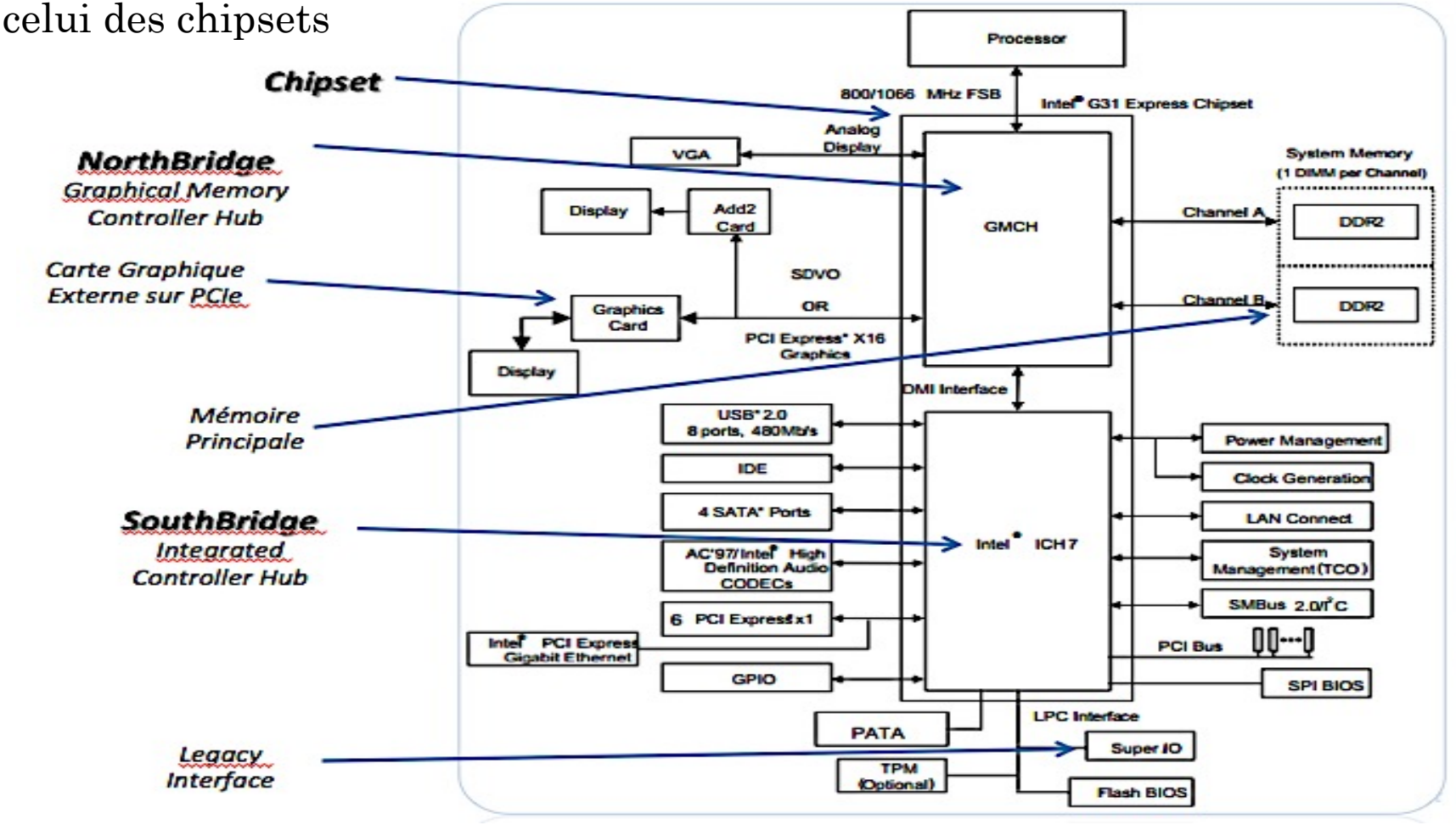
Architecture d'une carte mère

Le **SouthBridge (Pont Sud)** est un composant matériel servant de passerelle entre northBridge et fonctions matérielles "plus lentes" (PCI, USB, S-ATA…):

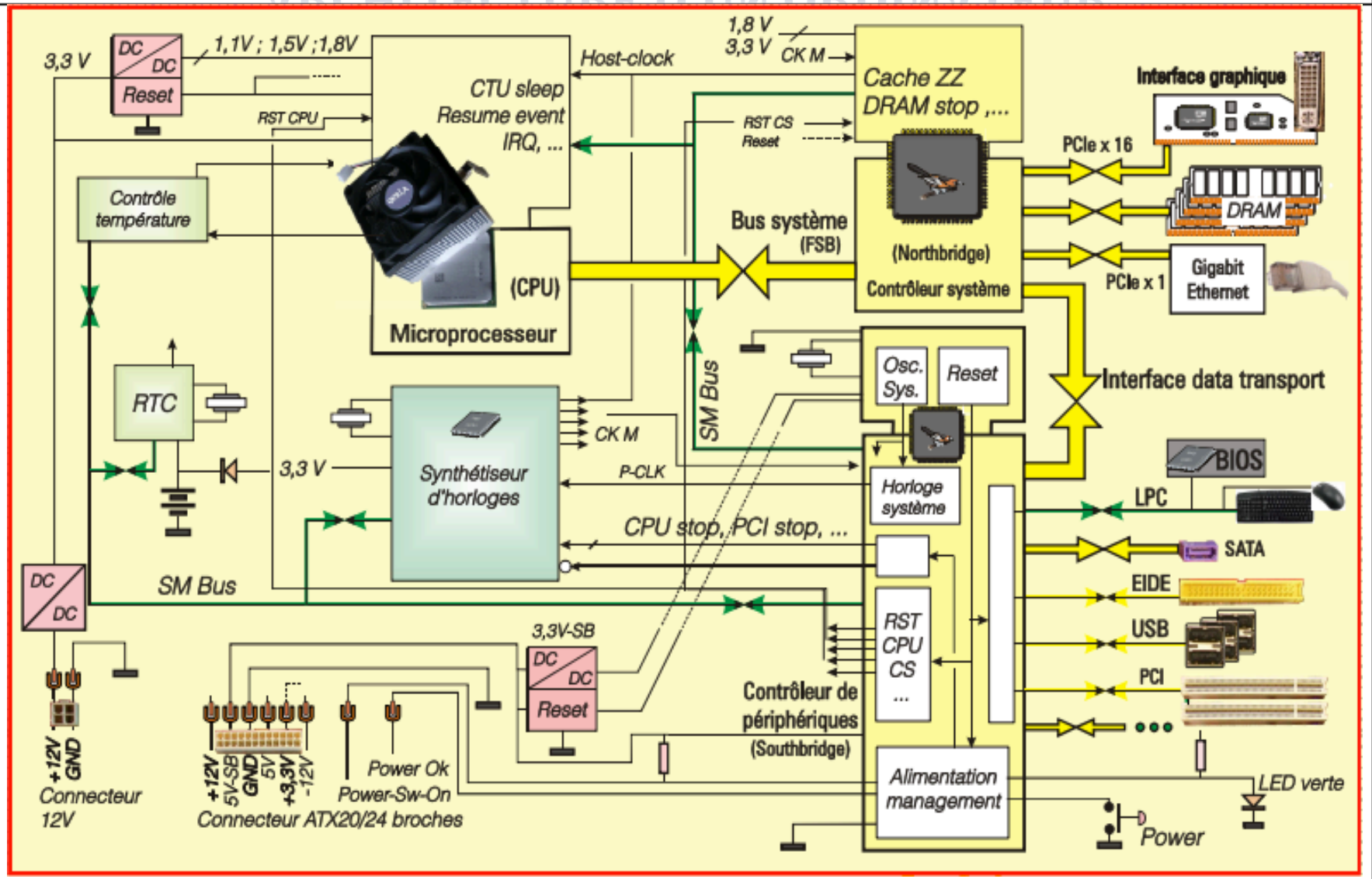


ARCHITECTURE D'UN ORDINATEUR

On appel **Chipset** (jeu de composants) l'ensemble northBridge et southBridge. Intel étant leader sur le marché des GPP, il l'est également sur celui des chipsets



ARCHITECTURE D'UN ORDINATEUR



ARCHITECTURE D'UN ORDINATEUR

Bus processeur : on l'appelle aussi bus système ou FSB (Front Side Bus). Il relie le microprocesseur au pont nord puis à la mémoire. C'est un bus 64 bits.

Bus IDE : il permet de relier au maximum 2 périphériques de stockage interne par canal (disque dur ou lecteur DVDROM/CDROM). Son débit est de 133 Mo/s. Lorsque 2 périphériques sont reliés sur le même canal, un doit être le maître (prioritaire sur la prise du bus) et l'autre l'esclave.

Bus PCI (Peripheral Component Interconnect) : Il a été créé en 1991 par Intel. Il permet de connecter des périphériques internes. C'est le premier bus à avoir unifier l'interconnexion des systèmes d'entrée/sortie sur un PC et à introduire le système plug-and-play. Il autorise aussi le DMA. C'est un bus de 32 bits. On retrouve une révision du bus PCI sur les cartes mères de serveur ayant une largeur de bus de 64 bits et une fréquence de 133 MHz.

Bus AGP (Accelered Graphic Port) : Il a été créé en 1997 lors de l'explosion de l'utilisation des cartes 3D qui nécessitent toujours plus de bandes passantes pour obtenir des rendus très réalistes. C'est une amélioration du bus PCI. Il autorise en plus le DIME (DIrect Memory Execution) qui permet au processeur graphique de travailler directement avec les données contenues dans la RAM sans passer par le microprocesseur à l'instar d'un DMA. C'est un bus 32 bits et son débit maximum est de 2 Go/s (en x8).

Bus ISA (Industry Standard Architecture) : C'est l'ancêtre du bus PCI. On ne le retrouve plus sur les nouvelles générations de cartes mères.

Bus SCSI (Small Computer System Interface) : c'est un bus d'entrée/sortie parallèle permettant de relier un maximum de 7 ou 15 périphériques par contrôleur suivant la révision du protocole utilisée. C'est une interface concurrente à l'IDE qui présente l'avantage de pouvoir connecter plus de périphériques pour des débits supérieurs. En outre, ces périphériques peuvent partager le bus lors d'un dialogue contrairement à l'IDE. Mais son coût reste très élevé… elle est utilisée pour les serveurs.

Bus USB (Universal Serial Bus) : c'est un bus d'entrée/sortie plug-and-play série. Dans sa deuxième révision (USB 2.0), il atteint un débit de 60 Mo/s. Un de ces avantages est de pouvoir connecter théoriquement 127 périphériques. Il supporte de plus le hot plug-and-play (connexion ou déconnexion de périphériques alors que le PC fonctionne).

Bus firewire : c'est un bus SCSI série. Il permet de connecter jusqu'à 63 périphériques à des débits très élevés (100 à 400 Mo/s). Ces applications sont tournées vers la transmission de vidéos numériques.

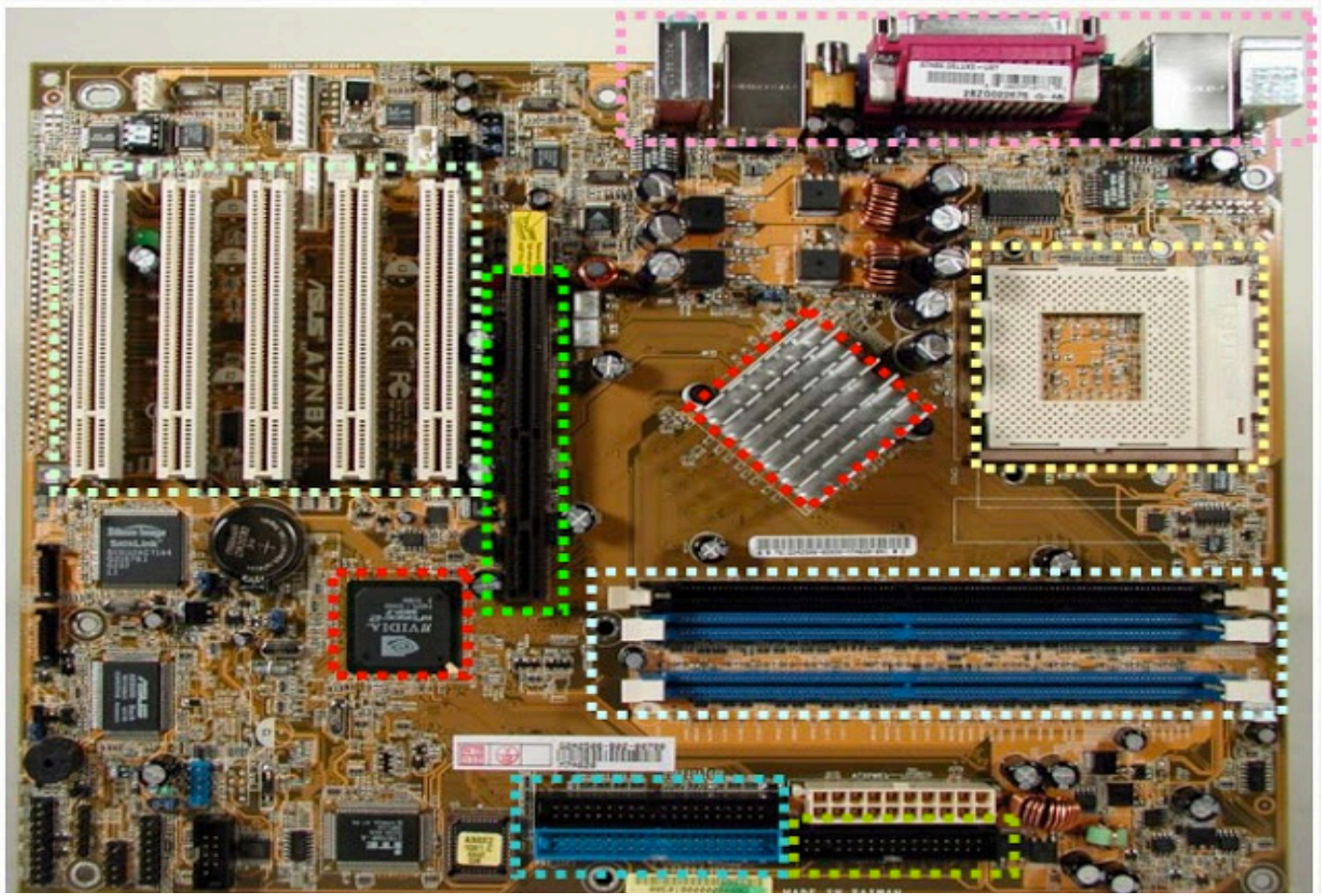
Liaison pont nord/pont sud : ses caractéristiques dépendent du chipset utilisé. Chaque fabricant a en effet développé une solution propriétaire pour connecter les deux composants de leur chipset. Pour Intel, c'est Intel Hub Architecture (IHA) dont les débits atteignent 533 Mo/s. Pour Nvidia (en collaboration avec AMD), c'est l'HyperTransport qui atteint des débits de 800 Mo/s.

Le Serial Ata, remplaçant du bus IDE, présente des débits de 150 Mo/s qui passeront bientôt à 300 Mo/s dans la prochaine révision du bus. Il permet de connecter des disques durs ou des lecteurs optiques.

Le PCI Express, remplaçant des bus PCI et AGP, permet d'atteindre des débits de 250 Mo/s dans sa version de base qui peuvent monter jusqu'à 8Go/s dans sa version x16 destinée à des périphériques nécessitant des bandes passantes très élevées (application graphique).

ARCHITECTURE D'UN ORDINATEUR

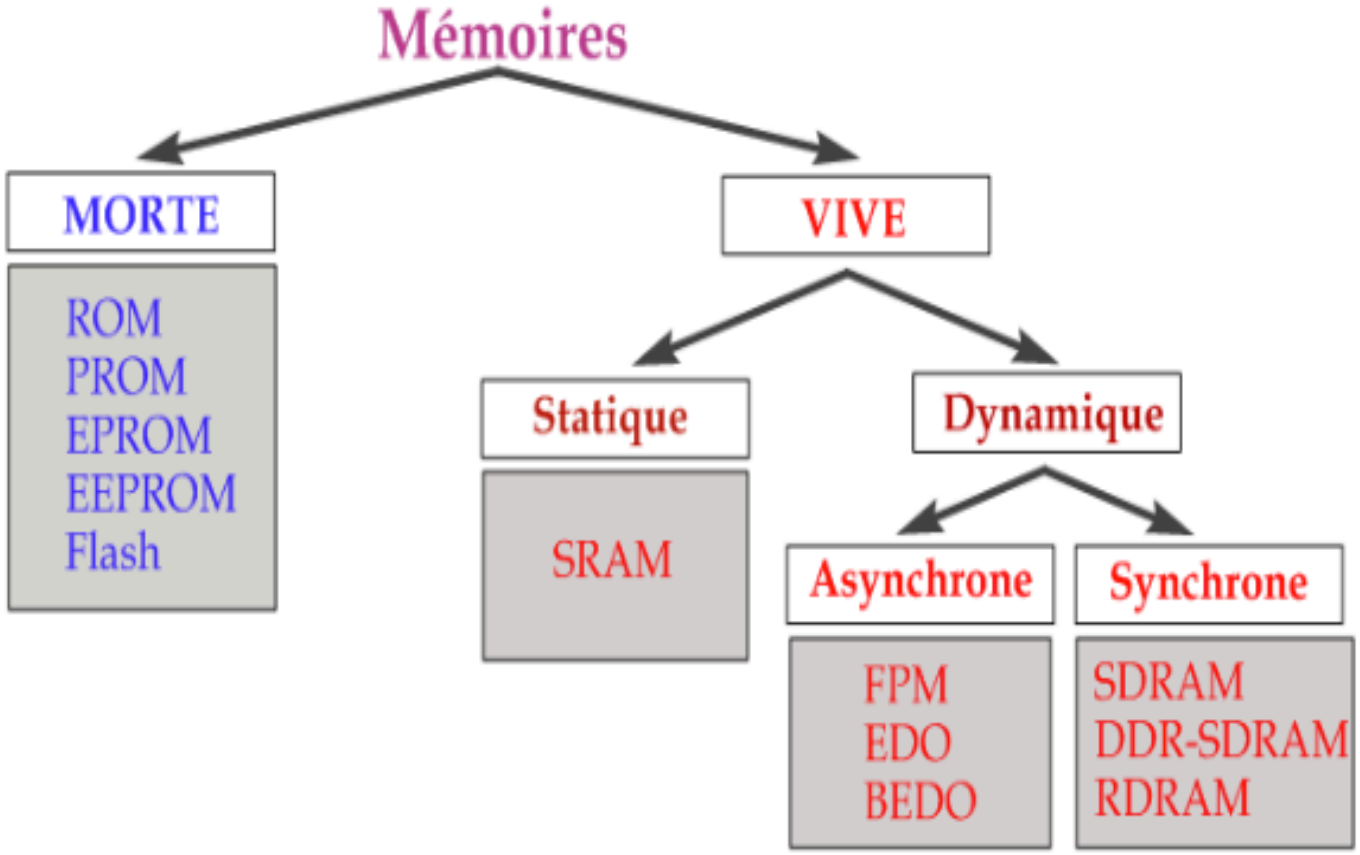
Exemple :
Carte mère
ASUS A7N8X



Connecteur PCI
Connecteur AGP
Connecteur RAM
Connecteur IDE
Chipset
Socket
Connecteurs Externes (port série, parallèle, firewire, USB, etc...)
Connecteur floppy

Les **mémoires**

Les différentes mémoires peuvent être classées comme indiqué sur le schéma ci-dessous :

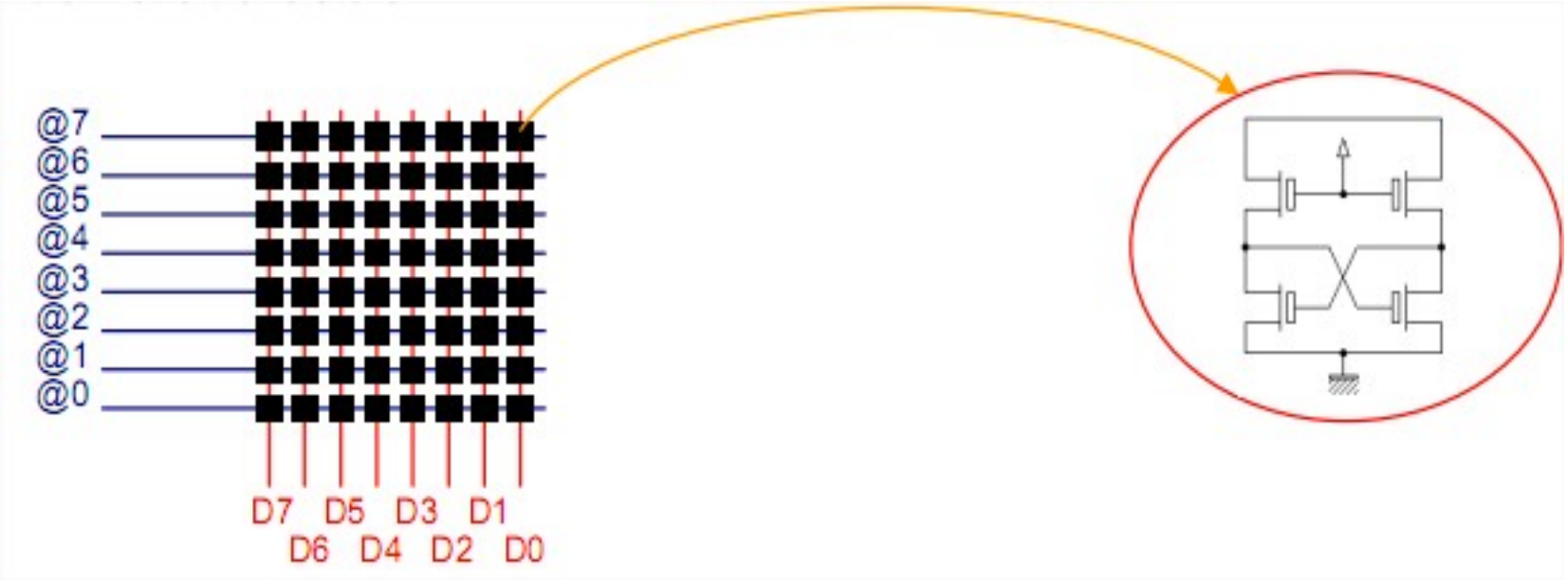


Les **mémoires**

Les mémoires vives (RAM) (Random Acces Memory)

Les RAM statiques

Le bit mémoire d'une RAM statique (SRAM) est composé d'une bascule. Chaque bascule contient entre 4 et 6 transistors

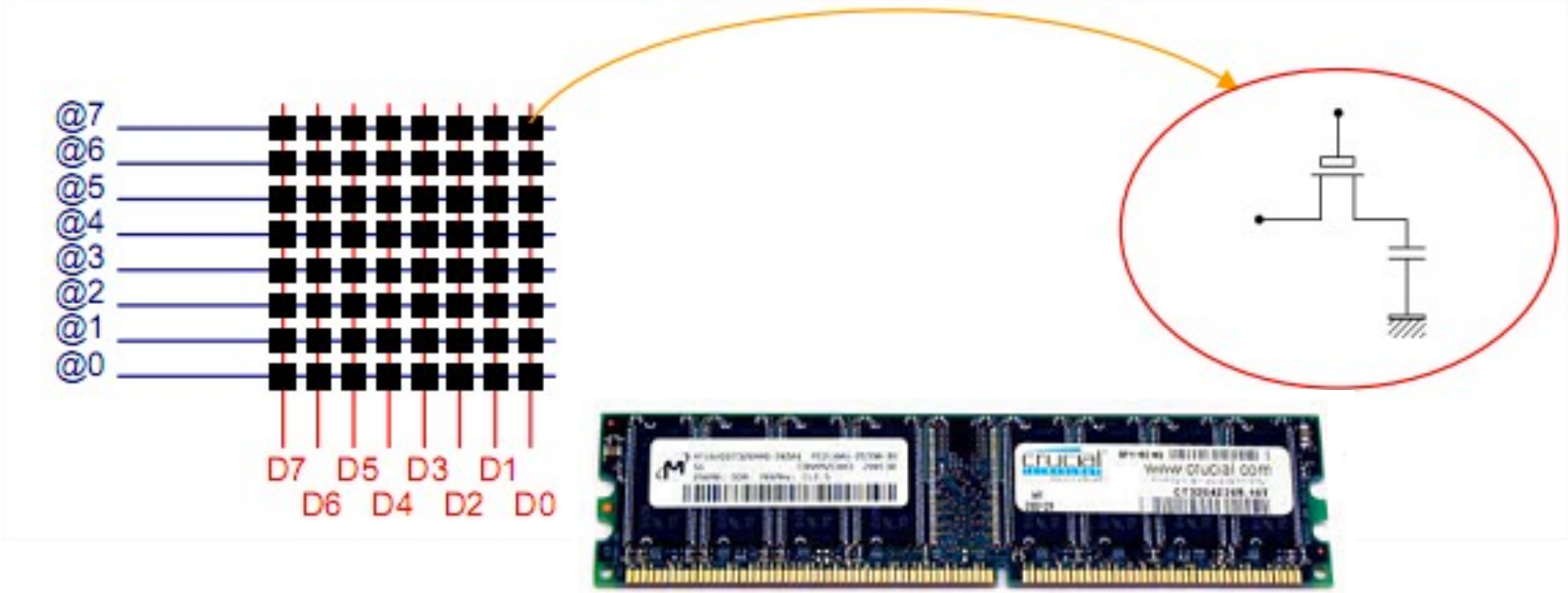


Les **mémoires**

Les mémoires vives (RAM)

Les RAM dynamiques

Dans les RAM dynamiques (DRAM), l'information est mémorisée sous la forme d'une charge électrique stockée dans un condensateur (capacité grille substrat d'un transistor MOS).



Les mémoires vives (RAM)

Avantages de RAM dynamiques :

Cette technique permet une plus grande densité d'intégration, car un point mémoire nécessite environ quatre fois moins de transistors que dans une mémoire statique. Sa consommation s'en retrouve donc aussi très réduite.

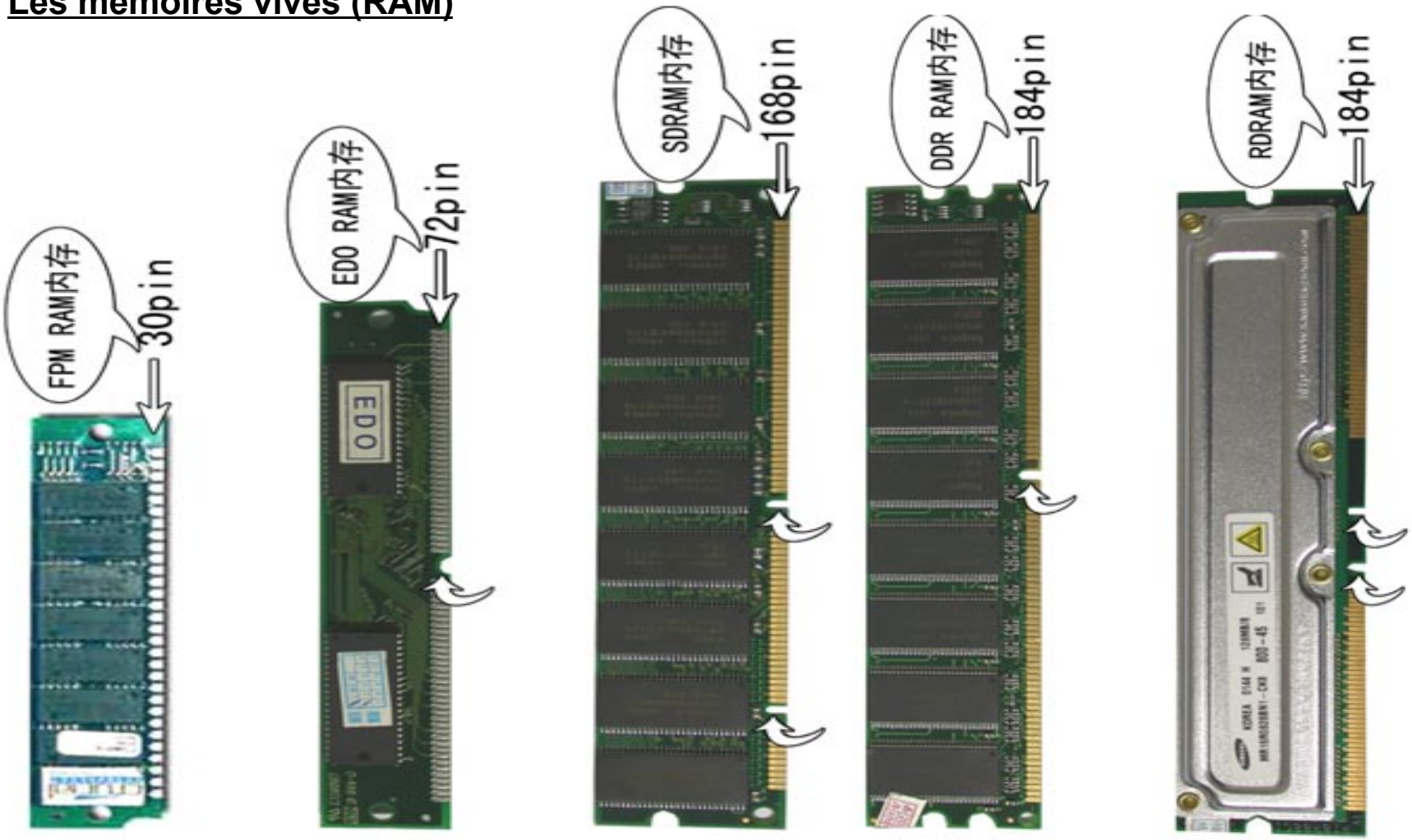
Inconvénients de RAM dynamiques :

La présence de courants de fuite dans le condensateur contribue à sa décharge. Ainsi, l'information est perdue si on ne la régénère pas périodiquement (charge du condensateur). Les RAM dynamiques doivent donc être rafraîchies régulièrement pour entretenir la mémorisation : il s'agit de lire l'information et de la recharger. Ce rafraîchissement indispensable a plusieurs conséquences :

- il complique la gestion des mémoires dynamiques car il faut tenir compte des actions de rafraîchissement qui sont prioritaires.
- la durée de ces actions augmente le temps d'accès aux informations.

D'autre part, la lecture de l'information est destructive. En effet, elle se fait par décharge de la capacité du point mémoire lorsque celle-ci est chargée. Donc toute lecture doit être suivie d'une réécriture.

Les mémoires vives (RAM)



Les mémoires vives (RAM)

Evolution des DRAM

Année	Désignation	Type	Largeur Bus
1987	Fast Page Mode (FPM)	Asynchronous	4 ou 8 octets
1990	Extended Data Out (EDO)	Asynchronous	4 ou 8 octets
1997	SDRAM PC 66	Synchronous	8 octets
1998	SDRAM PC 100	Synchronous	8 octets
1999	SDRAM PC 133	Synchronous	8 octets
1999	Rambus PC600/700/800	Synchronous	2 octets
2000	DDR-SDRAM PC 1600	Synchronous	8 octets
2001	DDR-SDRAM PC 2100	Synchronous	8 octets
2002	DDR-SDRAM PC 2700/3200	Synchronous	8 octets
2002	Rambus PC 1066	Synchronous	2 octets
2003	DDR-SDRAM PC 3500/4000/4400	Synchronous	8 octets
2003	DDR2-SDRAM PC 4200	Synchronous	8 octets
2007	DDR3	Synchronous	8 octets

Les mémoires vives (RAM)

RAM FPM (asynchrone)
la mémoire FPM (Fast Page Mode) fut utilisée sur les processeurs 80386 et 80486 d'Intel. Elle permet d'améliorer la vitesse de transfert des données en accelérant le transfert de données contiguës.

RAM EDO (asynchrone)
La mémoire EDO (Extended Data Out) est technologiquement identique à la FPM mais consiste en une amélioration logicielle qui permet l'envoi de données pendant le décodage de la prochaine adresse mémoire à accéder.

Mémoire SDRAM (synchrone)
La mémoire SDRAM (Synchronous DRAM) est capable d'échanger des données en se synchronisant avec le signal d'horloge pour des fréquences de 66, 100 ou 133 Mhz. La mémoire est découpée en deux bancs ce qui permet d'accéder à l'un pendant la configuration du second. Cette mémoire est utilisable en mode rafale 2,4,8 ou 512 octets.

Les mémoires vives (RAM)

Mémoire DDR-SDRAM

La Double Data Rate SDRAM est un standard mis au point par Nec, Samsung et Toshiba. Elle double le taux de transfert des actuelles DRAM en permettant de lire les données sur le front montant et sur le front descendant du signal d'horloge. La DDR SDRAM fonctionne à 100, 133, 150, 166, 200 ou 233 Mhz et est capable d'atteindre des vitesses de transfert de l'order de 1.6 à 4,2 Go/s

Mémoire RDRAM ou DRDRAM

La Direct Rambus DRAM développée par la société Rambus grâce au soutien d'Intel représente une amélioration de la DRAM basée sur une architecture à bus très rapide permettant de transférer 16 bits à 800 Mhz. Ce qui lui permet d'atteindre un taux de transfert de 1,6 Go/s si on utilise la technologie Dual-channel. La dernière évolution de la RDRAM, la PC 1200 permet quant à elle d'atteindre une bande passante théorique de 4,8 Go/s. C'est malheureusement une technologie propriétaire qui oblige à verser des royalties à RAMBUS et Intel. D'autre part sa conception est nettement plus coûteuse que celle de la DDR-SDRAM.

Les mémoires vives (RAM)

Mémoire DDR2-SDRAM

La mémoire DDR2 est une évolution de la DDR. Ces deux types de mémoires possèdent des voltages ainsi que des connecteurs différents ce qui les rend incompatibles entre elles.

La mémoire DDR vieillissante est peu à peu remplacée par la DDR2 qui offre une bande passante supérieure.

Les mémoires vives (RAM)

Fréquence et débit mémoire

Le débit théorique d'une mémoire d'un ordinateur dépend de trois facteurs : la fréquence du bus FSB (Front Side Bus) reliant le CPU à la mémoire, la largeur du bus en nombre de bits, un coefficient multiplicateur qui dépend de la technologie utilisée (normal = 1, DDR = 2, QDR = 4).

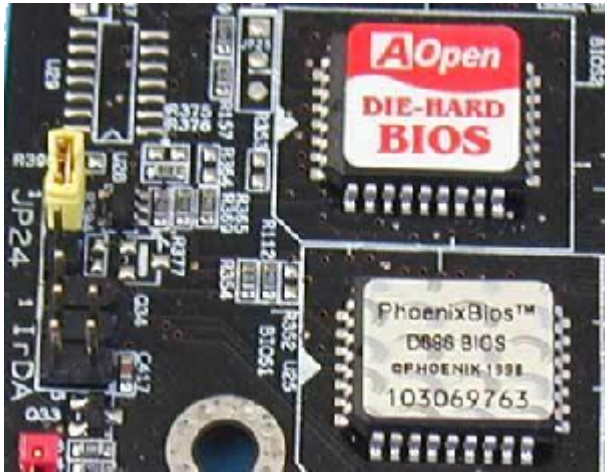
Le débit mémoire est donc donné par la formule suivante :

$$debit = Frequence \times Largeur \times Coef$$

Exemple : Un ordinateur doté d'un FSB à 133 Mhz de largeur 64 bits utilisant la technologie DDR possède un débit mémoire théorique de :
133 × 64 × 2 bits/s soit 2,128 Go/s.

Exemple : Un Pentium 4 doté d'un FSB à 100 Mhz quad pumped de largeur 64 bits utilisant de la DRDRAM possède un débit mémoire théorique de : 100 × 64 × 4 bits/s soit 3,2 Go/s.

La mémoire morte : ROM (Read Only Memory)



Les mémoires mortes ou ROM (Read Only Memory) sont des mémoires non volatiles qui sont vouées à être accédées en lecture en fonctionnement normal. Elles contiennent du code et des données qui ne sont pas amenés à changer souvent. Les ROM contiennent généralement les routines d'accès de base aux périphériques.

ROM classiques

L'information contenue dans ces mémoires est enregistrée de manière irréversible lors de la fabrication du circuit. Le principe de réalisation de ces mémoires est le réseau de diodes.

PROM

La mémoire PROM (Programmable ROM) est une ROM à enregistrement irréversible qui peut être programmée par un utilisateur grâce à un dispositif appelé programmateur de PROM.

La mémoire morte : ROM (Read Only Memory)

EPROM et EEPROM

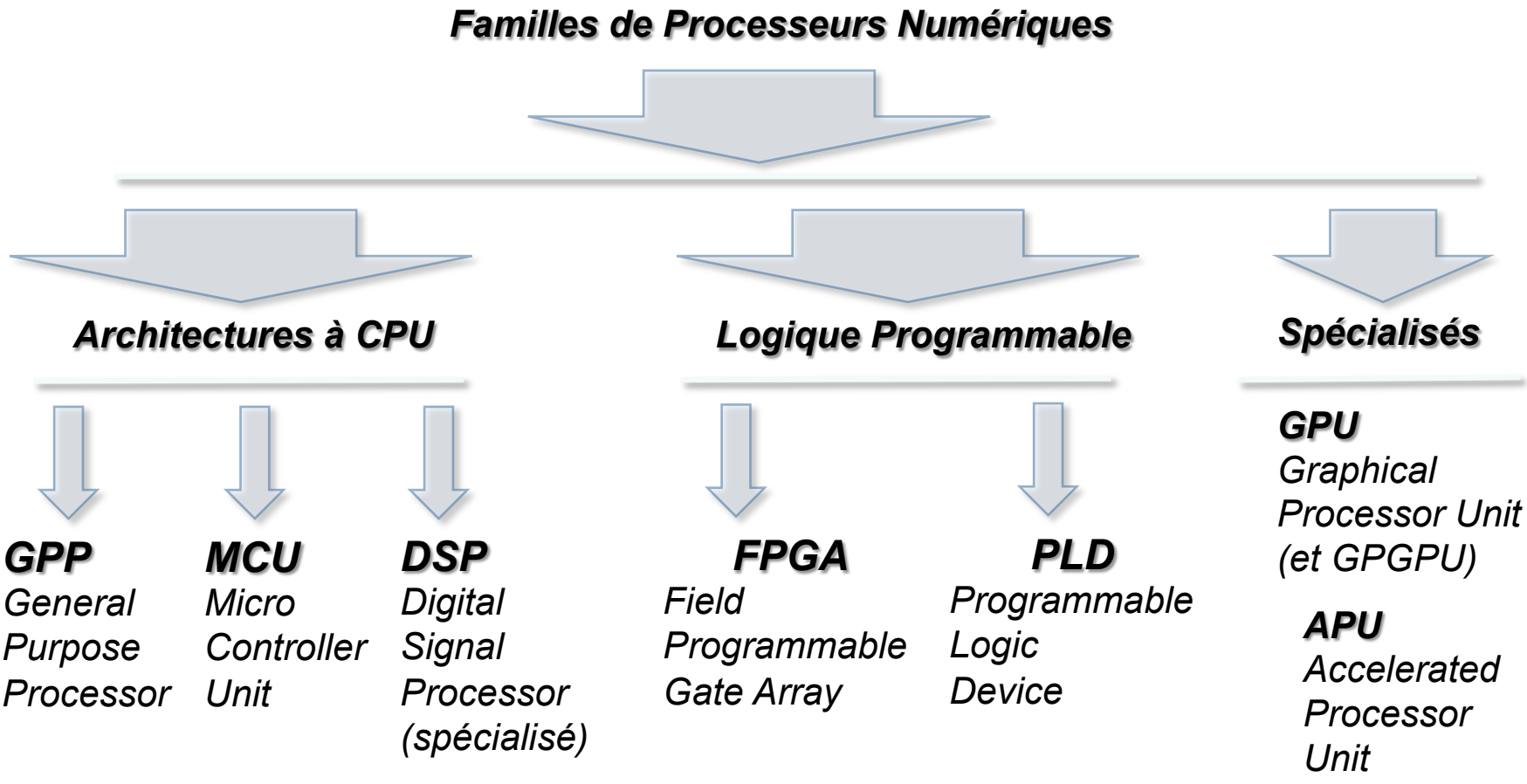
Les EPROM (Erasable Programmable ROM) présentent l'avantage de pouvoir être effacées et réécrites. Pour effacer une EPROM il faut la soumettre à un rayonnement ultraviolets pendant 30 minutes. Elles sont facilement reconnaissables en raison de la présence sur leur face supérieure d'une fenêtre de quartz obturée par un adhésif afin de les soustraire aux rayonnements ultraviolets naturels (soleil, néon).

Les EEPROM (Electrically EPROM) quant à elles utilisent un procédé électrique pour l'effacement des données. Elles sont donc bien plus pratiques que les EPROM.

Flash

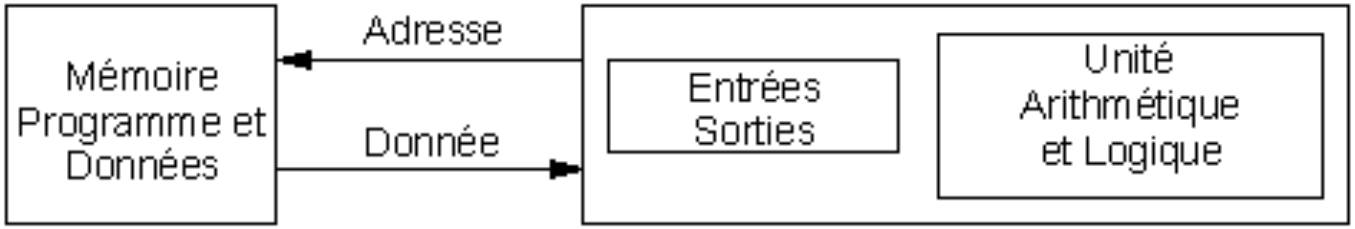
Les mémoires Flash réalisent la synthèse des technologies EEPROM et SRAM. Elles sont rapides, ne nécessitent pas de rafraîchissement. Elle sont effaçables et reprogrammables rapidement par blocs de 64 Ko. Malheureusement son prix de revient est assez élevé et on ne peut la reprogrammer qu'un nombre limité de fois (environ 100.000).

Le microprocesseur

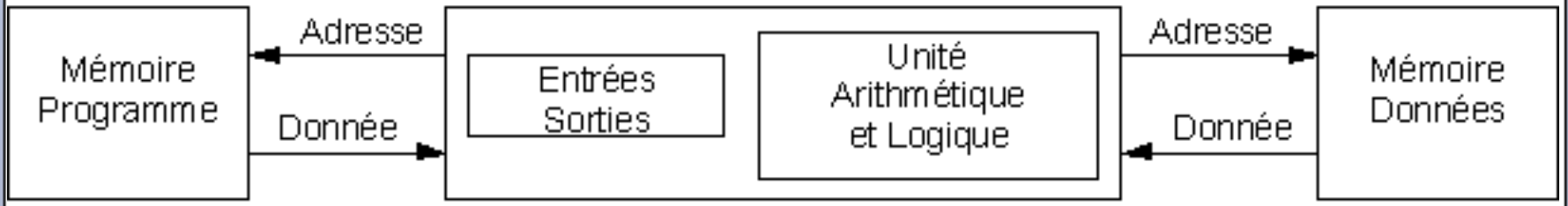


ARCHITECTURE D'UN ORDINATEUR

Architecture de type Von Neuman



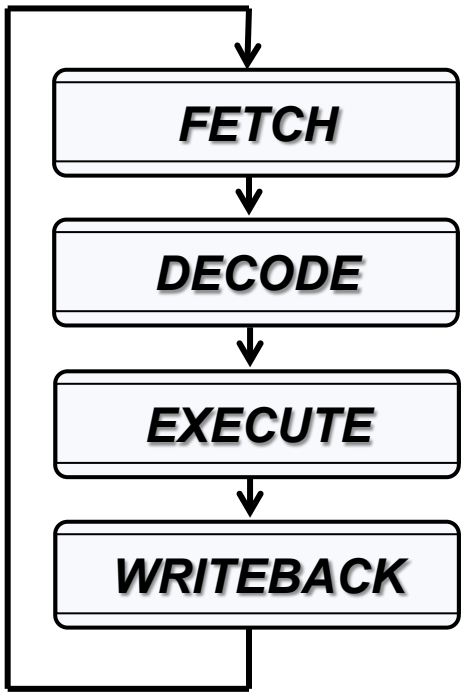
Architecture de type Harvard



Présentation globale des deux principales structures de microprocesseurs

ARCHITECTURE D'UN ORDINATEUR

Tout CPU effectue séquentiellement les traitements présentés ci-dessous :



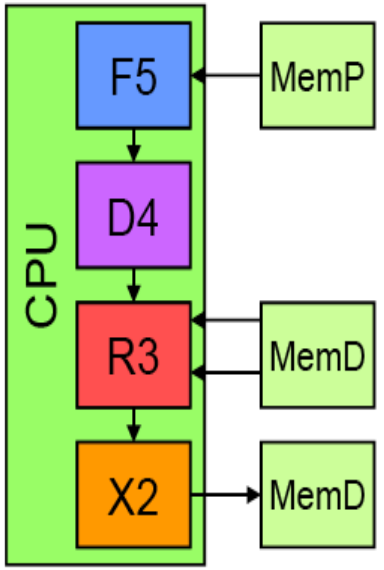
- **FETCH** : Aller chercher le code binaire d'une instruction en mémoire programme. Beaucoup de CPU récents sont capables d'aller chercher plusieurs instructions durant la phase de fetch.
- **DECODE** : décodage du ou des opcodes des instructions précédemment fetchées.
- **EXECUTION** : Exécution de ou des instructions précédemment décodées. Cette opération est réalisée par les EU's (Execution Unit).
- **WRITEBACK** : Ecriture du résultat en mémoire ou dans un registre (typiquement dans le CPU)

ARCHITECTURE D'UN ORDINATEUR

Exécution avec pipeline

	t1	t2	t3	t4	t5	t6	t7	t8
Fetch	F1	F2	F3	F4	F5	…	…	…
Decode		D1	D2	D3	D4	D5	…	…
Read			R1	R2	R3	R4	R5	…
Execute				X1	X2	X3	X4	X5

Amorçage du pipeline



- Instructions segmentées en étages
- Exécution entrelacée de plusieurs instructions
  - Chacune à une étage différent
- Augmentation de la fréquence d'horloge
  - Etages plus simples donc plus rapides
- Géré par le séquenceur

Les jeux d'instructions peuvent être classés en 2 grandes familles, **RISC et CISC**, respectivement Reduce et Complex Instruction Set Computer. Les architectures RISC n'implémentent en général que des instructions élémentaires (CPU's ARM, MIPS, 8051, PIC18 …). A l'inverse, les architectures CISC (CPU's x86-64, 68xxx …) implémentent nativement au niveau assembleur des traitements pouvant être très complexes (division, opérations vectorielles, opérations sur des chaînes de caractères …).

En 2012, la frontière entre ces deux familles est de plus en plus fine. Par exemple, le jeu d'instructions des DSP RISC-like TMS320C66xx de TI compte 323 instructions. Néanmoins, les architectures compatibles x86-64 sont des architectures CISC. Nous allons rapidement comprendre pourquoi.

Avantages architecture CISC :

- Empreinte mémoire programme faible, donc plus d'instructions contenues en cache. Néanmoins sur CPU CISC, plus de 80% des instructions compilées sont de types RISC.
- Compatibles x86-64, rétrocompatibilité des applications développées sur anciennes architectures.

Inconvénients architecture CISC :

- Architecture CPU complexe (mécanismes d'accélération matériels, décodeurs, Execution Units …), donc moins de place pour le cache.
- Jeu d'instructions mal géré par les chaînes de compilation (mécanismes d'optimisation)

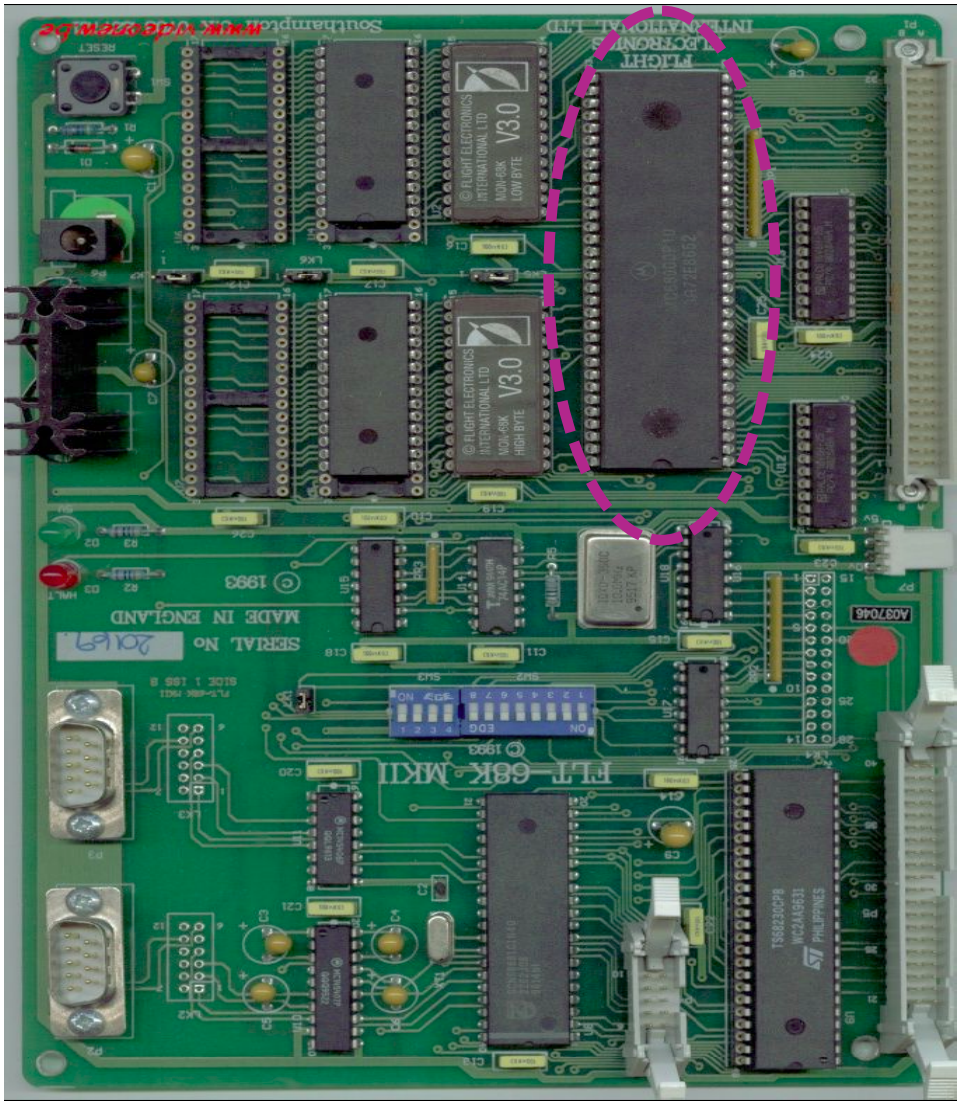
Inconvénients architecture RISC :

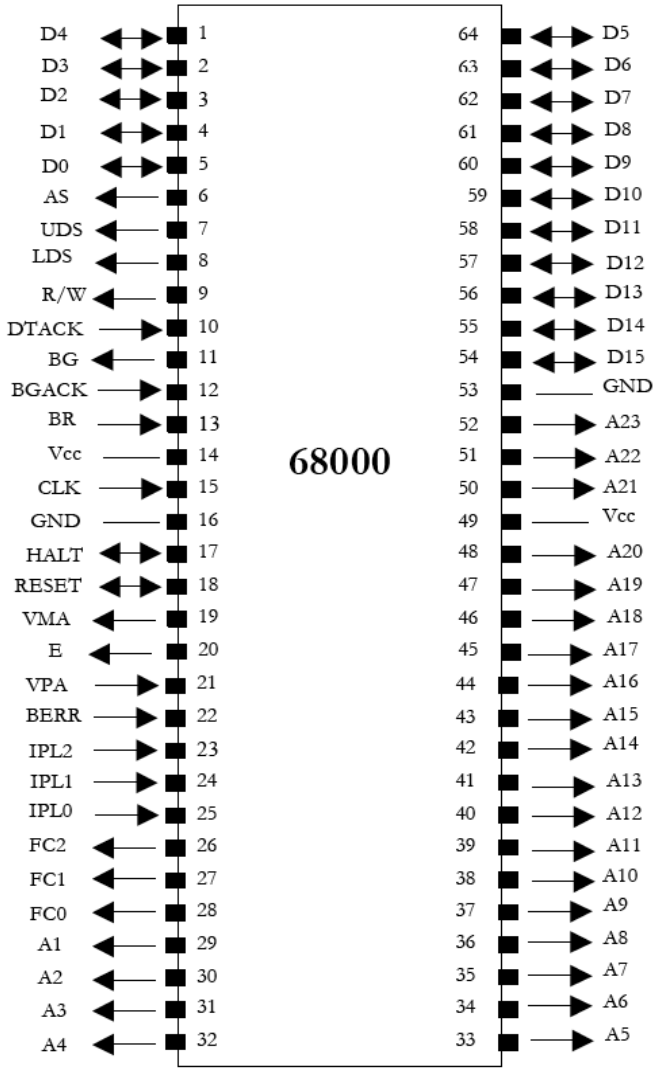
- Empreinte mémoire programme élevée, donc moins d'instructions contenues en cache et mémoire principale.

Avantages architecture RISC :

- Architecture CPU moins complexe (mécanismes d'accélération matériels, décodeurs, Execution Units …).
- En général, tailles instructions fixes et souvent exécution en un ou deux cycles CPU.
- Jeu d'instructions plus simple à appréhender pour le développeur et donc le compilateur. Jeu d'instructions très bien géré par les chaînes de compilations (mécanismes d'optimisation). Beaucoup d'architectures RISC récentes, travaillent avec de nombreux registres de travail généralistes. Facilite le travail du compilateur.

MICROPROCESSEURS MOTOROLA (EXEMPLE : LE 68000)





MICROPROCESSEURS MOTOROLA (EXEMPLE : LE 68000)

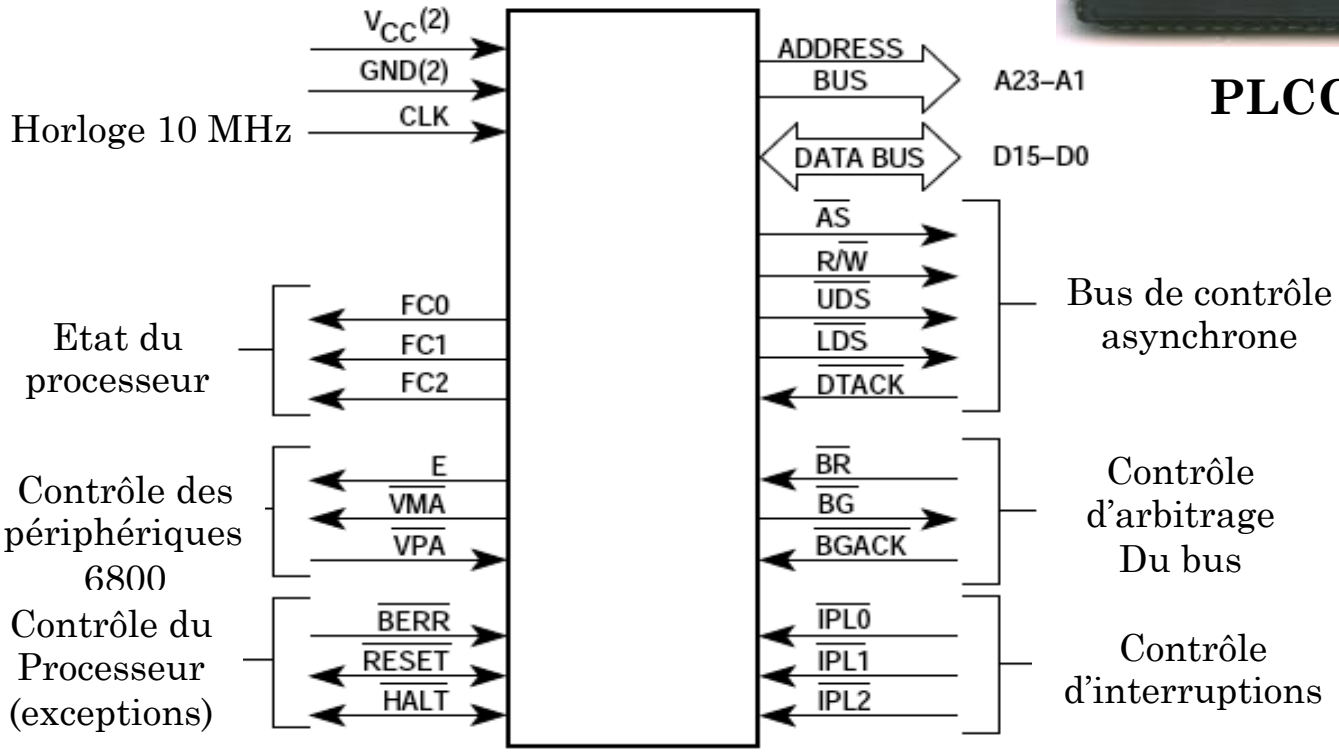
Bien que désuet aujourd'hui, l'architecture de 68000 et sa programmation reste une référence dans l'étude des microprocesseurs.



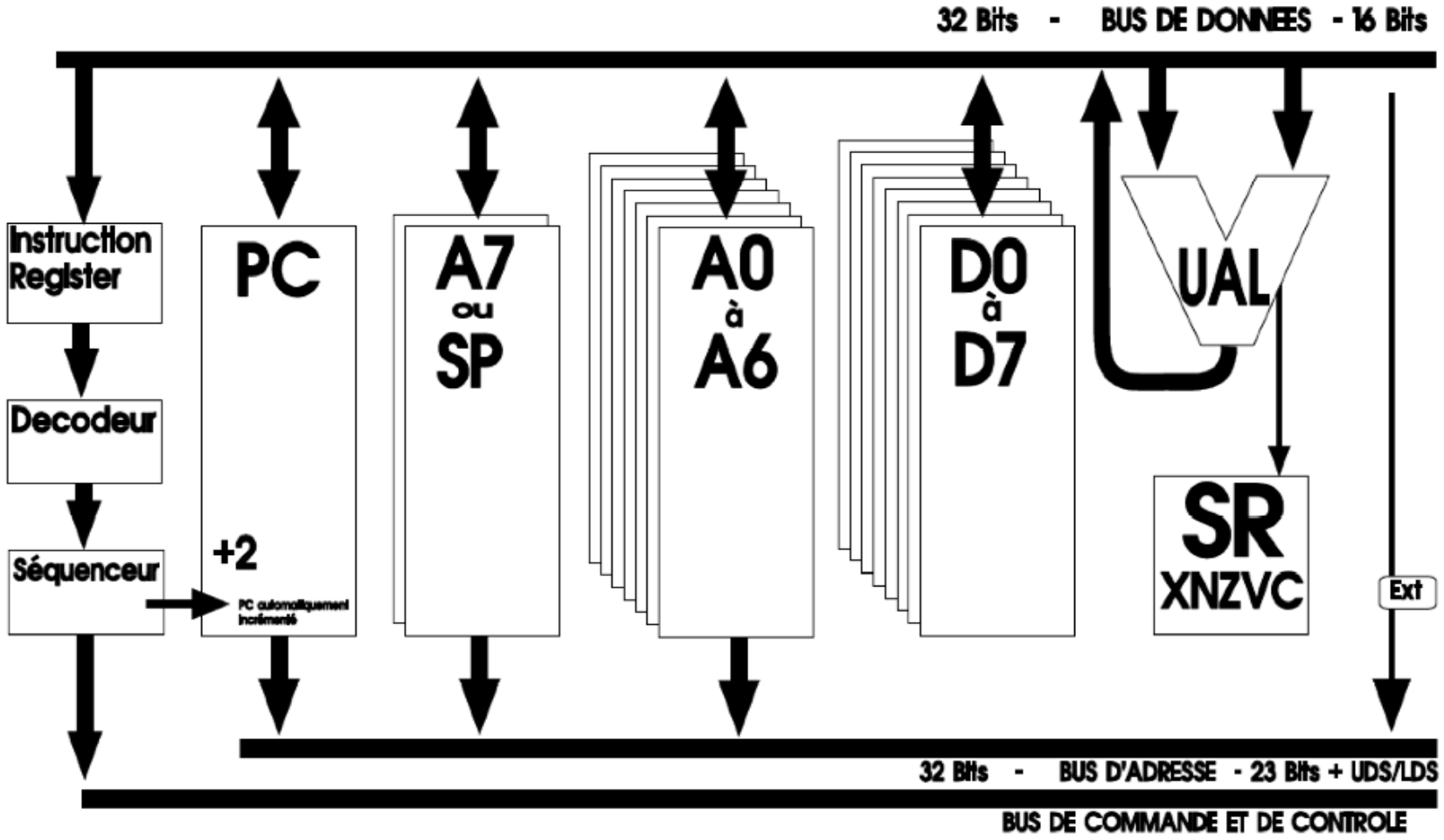
PLCC



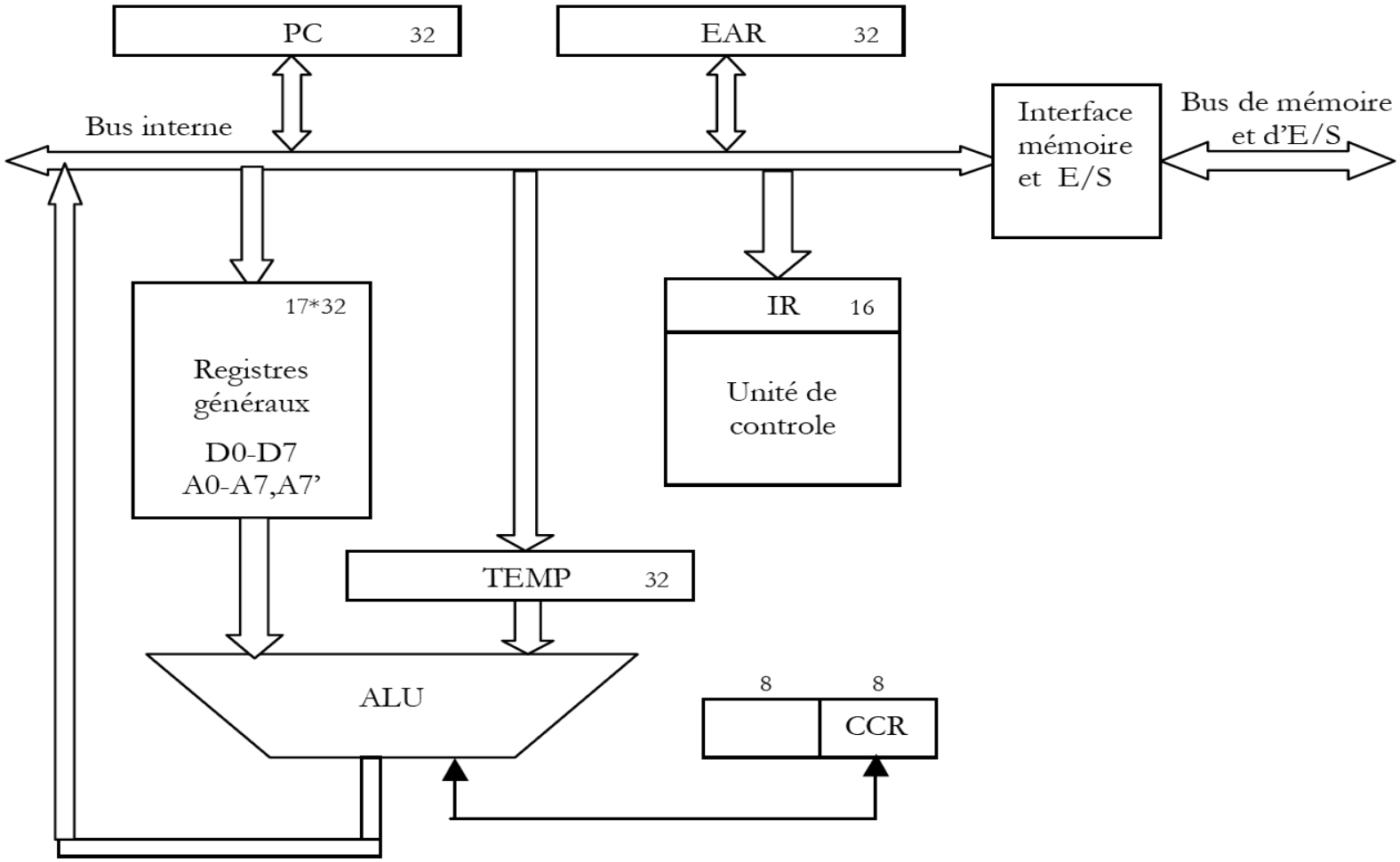
DIP



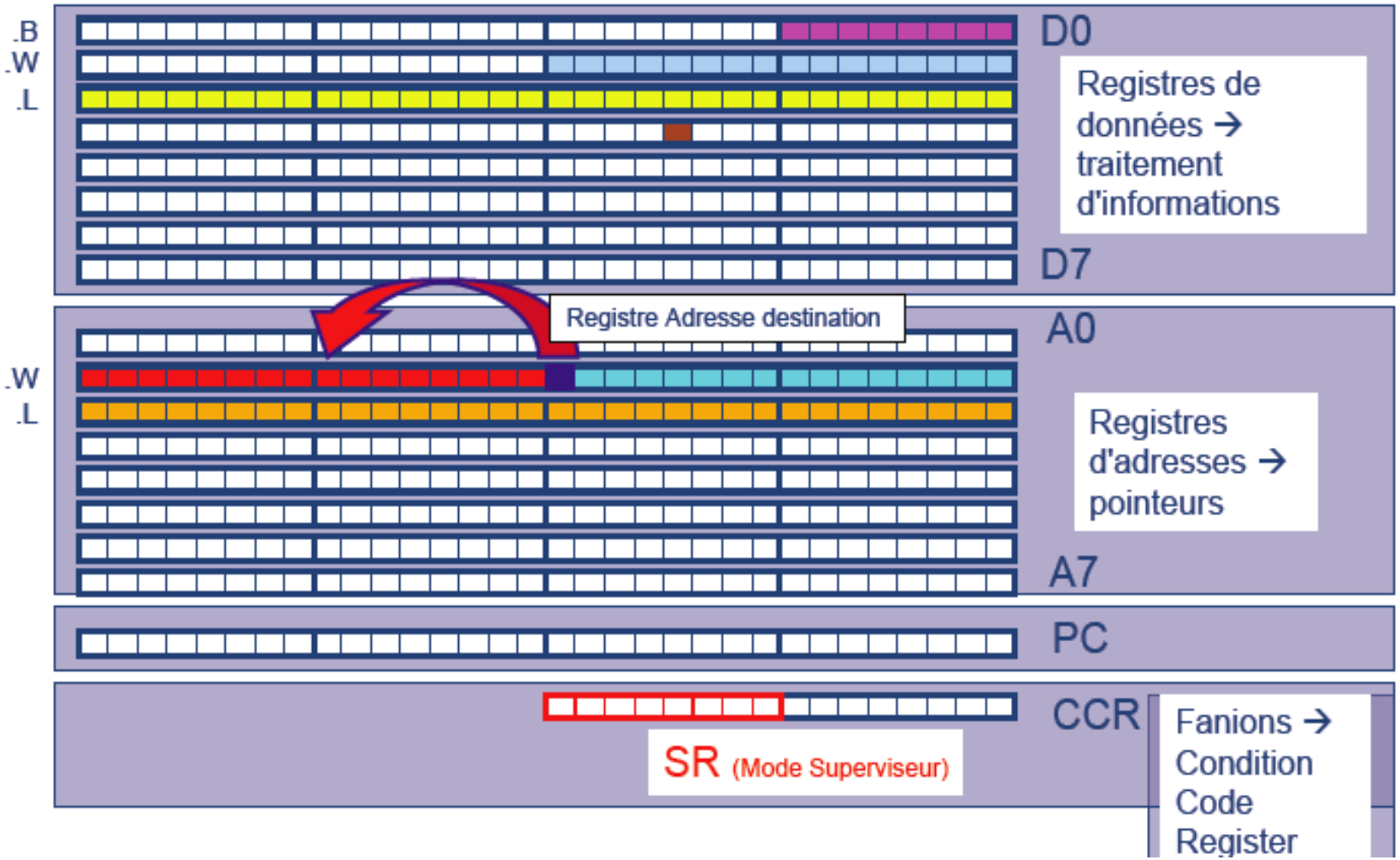
Organisation interne du MC68000



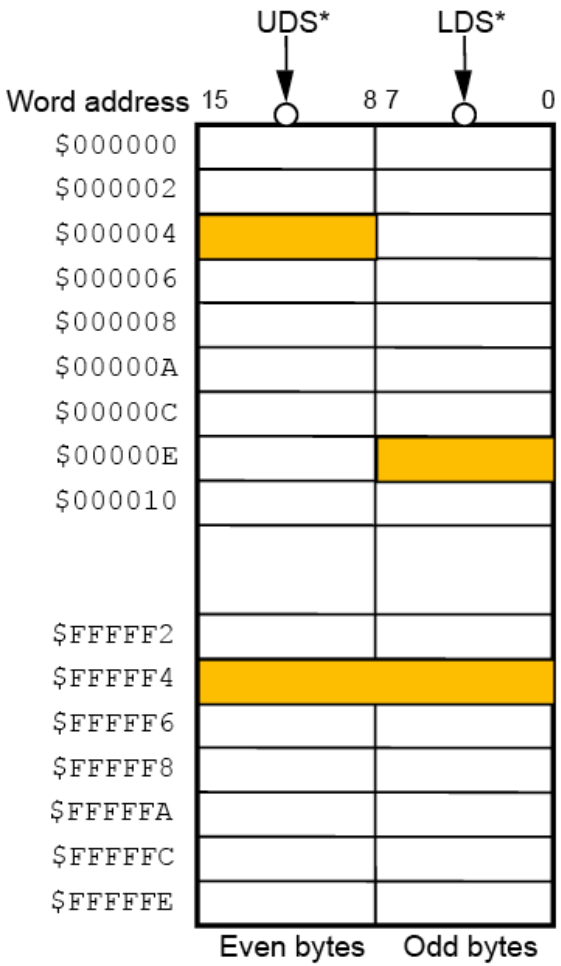
Organisation interne du MC68000

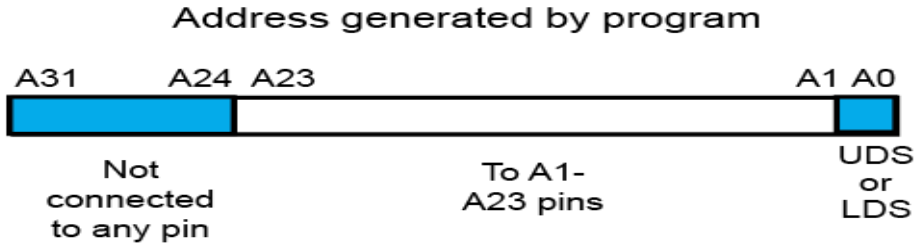


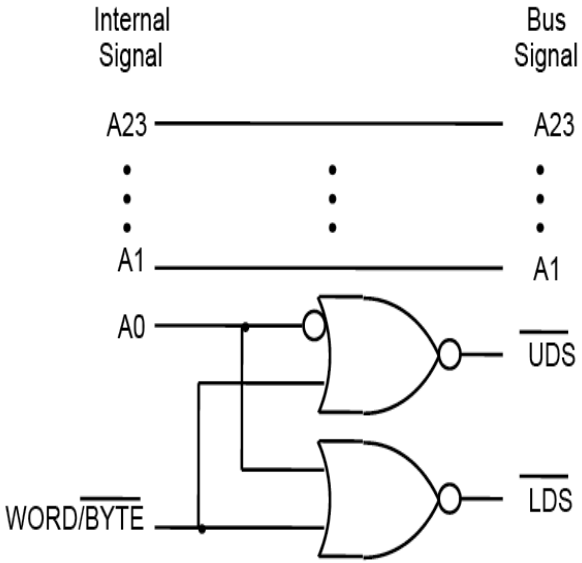
Registres du processeur



MICROPROCESSEURS MOTOROLA (EXEMPLE : LE 68000)

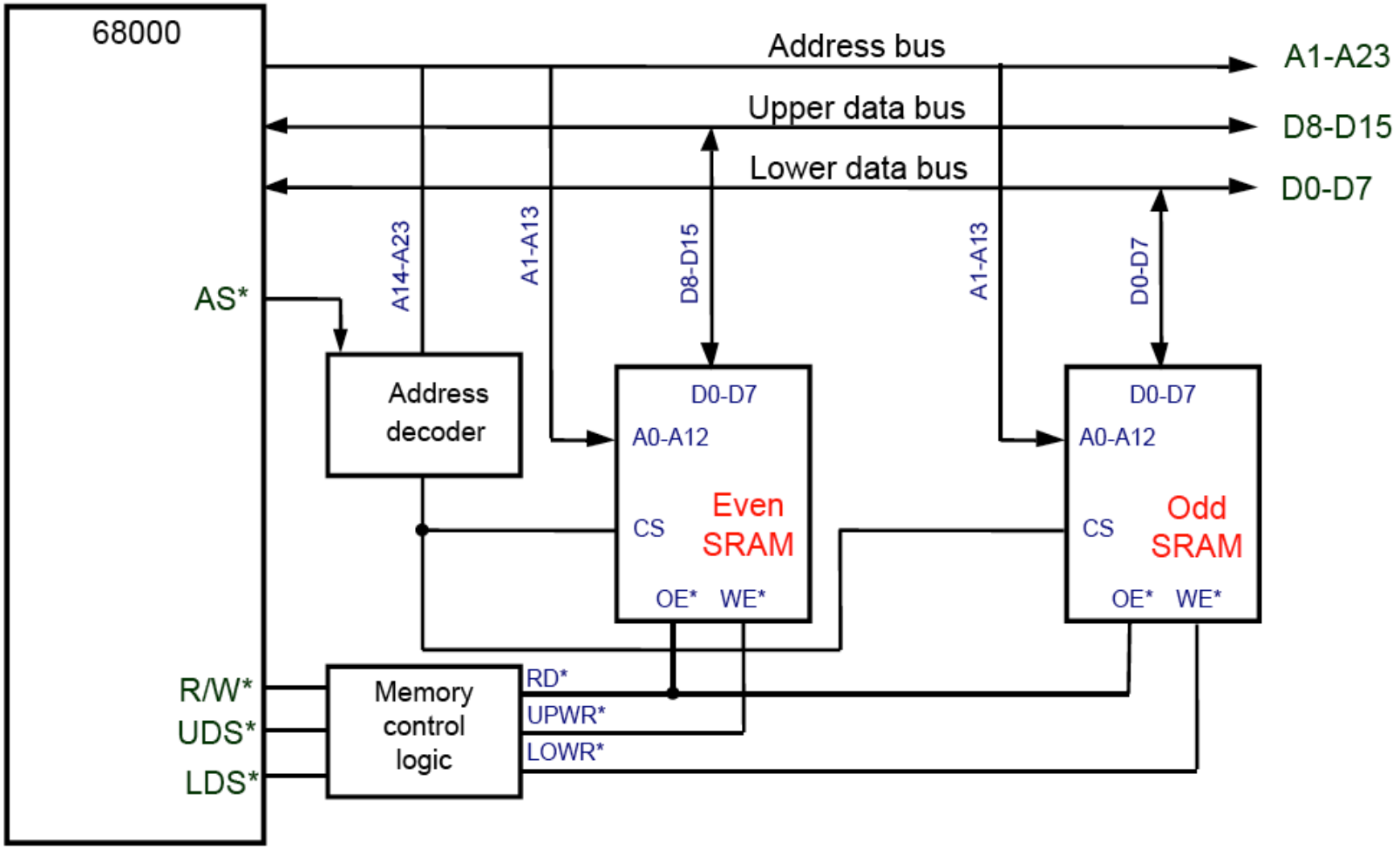






WORD/BYTE	A0	UDS	LDS
1	X	0	0
0	0	0	1
0	1	1	0

MICROPROCESSEURS MOTOROLA (EXEMPLE : LE 68000)



Exemple: Transfert à l'adresse 0x100 (adresse paire)

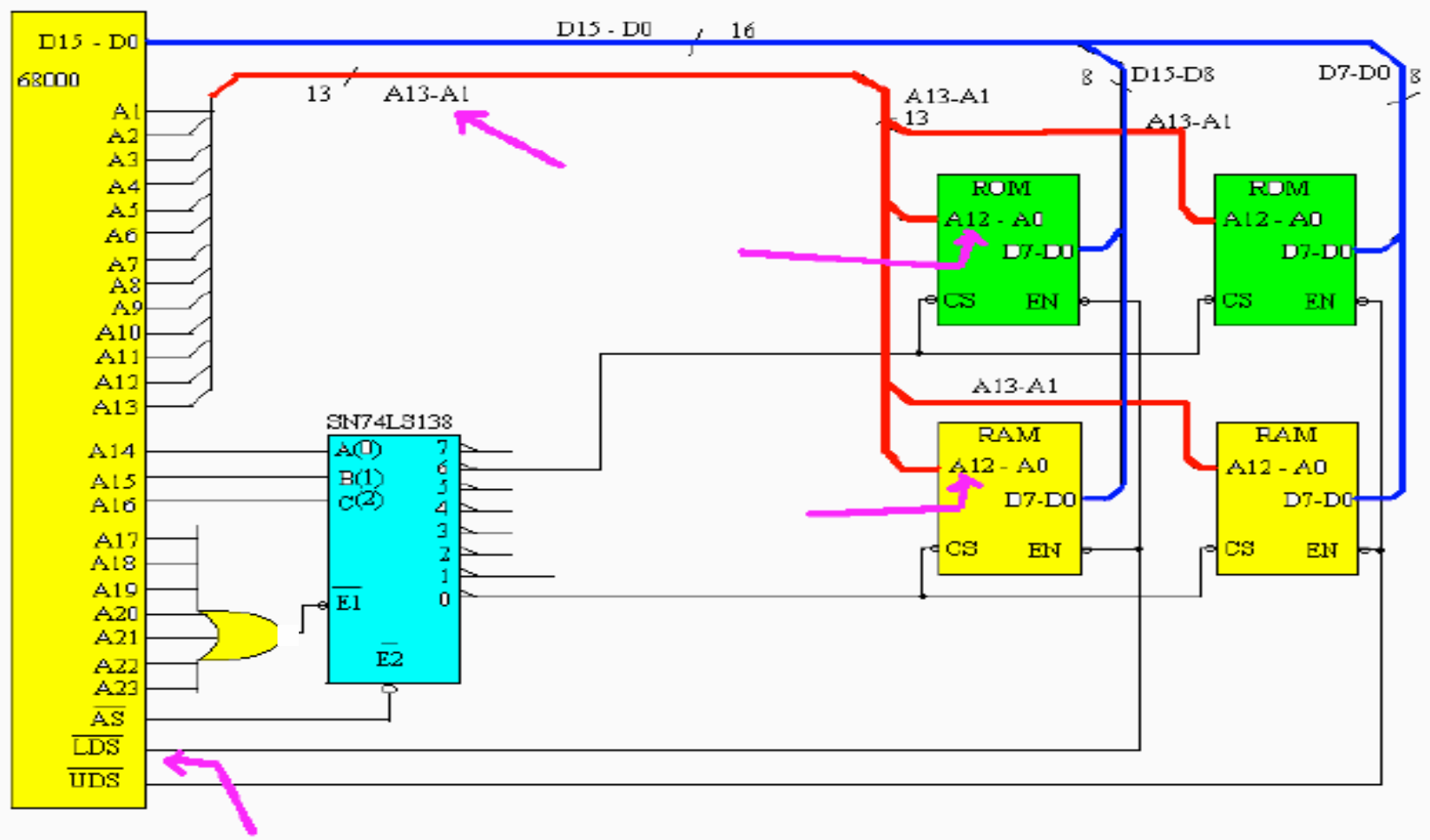
Adresse (dans le programme)	0	0	0	1	0	0	0	0	0	0	0	0
id en hexadécimal	1				0				0			
Broches du bus d'adresse	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	Uds/Lds
sur le bus pour Opérande Octet	0	0	0	1	0	0	0	0	0	0	0	0/1
sur le bus pour Opérande Mot	0	0	0	1	0	0	0	0	0	0	0	0/0
broches d'adresse de la mémoire	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
	0			8				0				

Exercice1 :

On voudrait permettre à un µp 68000 d'adresser une zone mémoire (RAM) de 64K mots (ou 128 KB) avec une adresse de début fixée à $480000.

Réalisez le MAP de mémoire

Exercice2 : Déterminez le MAP de mémoire associé à la figure suivante



Exercice 4 :

Le µp 68000, qui fait le cœur de la carte de commande, permet d'adresser une zone mémoire dont l'organisation (MAP de mémoire) est décrite par le tableau suivant :

	A23	A22	A21	A20	A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Réservé	0	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
ROM1/2	0	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Réservé	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
RAM1/2	0	1	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
RAM3/4	0	1	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
RAM5/6	0	1	0	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
RAM7/8	0	1	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Réservé	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Int E/S	1	1	1	1	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

X : 0 ou 1 **Int E/S** : Interfaces d'Entrées- Sorties

MICROPROCESSEURS MOTOROLA (EXEMPLE : LE 68000)

1- En se référant au tableau précédent complétez les informations relatives aux :

ROM : Adresse de début (hex) : ..... Adresse de fin (hex) : .....

Taille totale : ..... Taille de chaque circuit EPROM : .....

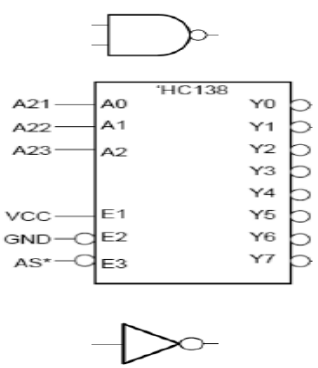
RAM : Adresse de début (hex) : ..... Adresse de fin (hex) : .....

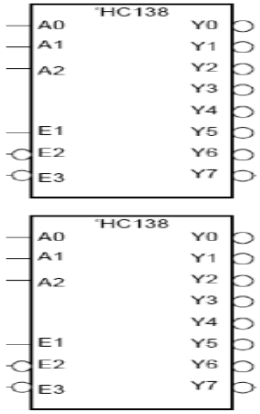
Taille totale : ..... Taille de chaque circuit RAM : .....

Interfaces d'E/S : Adresse de début (hex) : ..... Adresse de fin (hex) : .....

Taille totale : .....

2- Complétez le circuit de décodage d'adresse associé à la MAP proposée par le constructeur





$\overline{CS}$ (EPROM1/2)

$\overline{CS}$ (RAM1/2)

$\overline{CS}$ (RAM3/4)

$\overline{CS}$ (RAM5/6)

$\overline{CS}$ (RAM7/8)

$\overline{CS}$ (Int E/S)

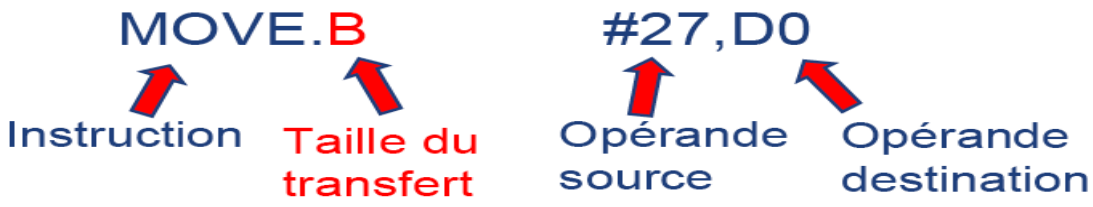
3- Quel est le rôle du circuit HC138 ?

.....

4- Quel est le rôle de la ligne AS ?

.....

Format d'une instruction



Types de données

bits		
bytes	*8 bits*	*.B*
words	*16 bits*	*.W*
long words	*32 bits*	*.L*

- Jeu d'instructions
  - Transferts
  - Arithmétiques et logiques
  - Branchements
  - Spéciales

Données mémoires

- ***DS : data store***

  Réservation d'un espace mémoire.

  étiquette *DS.W* *4*

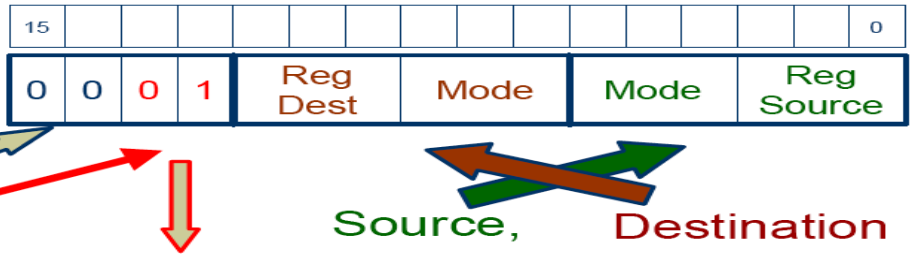
  Cette instruction réserve 4×2 octets, auquel on pourra faire référence avec *étiquette*.

- ***DC : define constant***

  Définition d'une constante au format *byte*, *word*, ou *long*.

  étiquette *DC.W* *1000*

Codage des instructions



–	0	0
.B octet	0	1
.W doublet	1	1
.L quadlet	1	0



0	D0 ou A0
1	D1 ou A1
2	D2 ou A2
3	D3 ou A3
4	D4 ou A4
5	D5 ou A5
6	D6 ou A6
7	D7 ou A7

000	Dn	**Registre** de donnée	
001	An	**Registre** d'adresse (MOVEA)	
010	(An)	**Indirect par registre**	
011	(An)+	Ind. par reg., **post-auto incrémenté**	
100	-(An)	Ind. par reg., **pré-auto décrémenté**	
101	(d16,An)	Ind. par reg, avec déplacement	
110	(d8,An,Xn)	Ind. par reg indexé avec déplacement	
111	xxx.W	**Absolu** court	Reg **000**
111	xxx.L	**Absolu** long	Reg **001**

Instructions de transferts (Move)

L'instruction MOVE transfert la source vers la destination, en écrasant le contenu de cette dernière. Par défaut, le format de l'instruction MOVE est .W.

Les modes d'adressages spécifient comment accéder à l'information

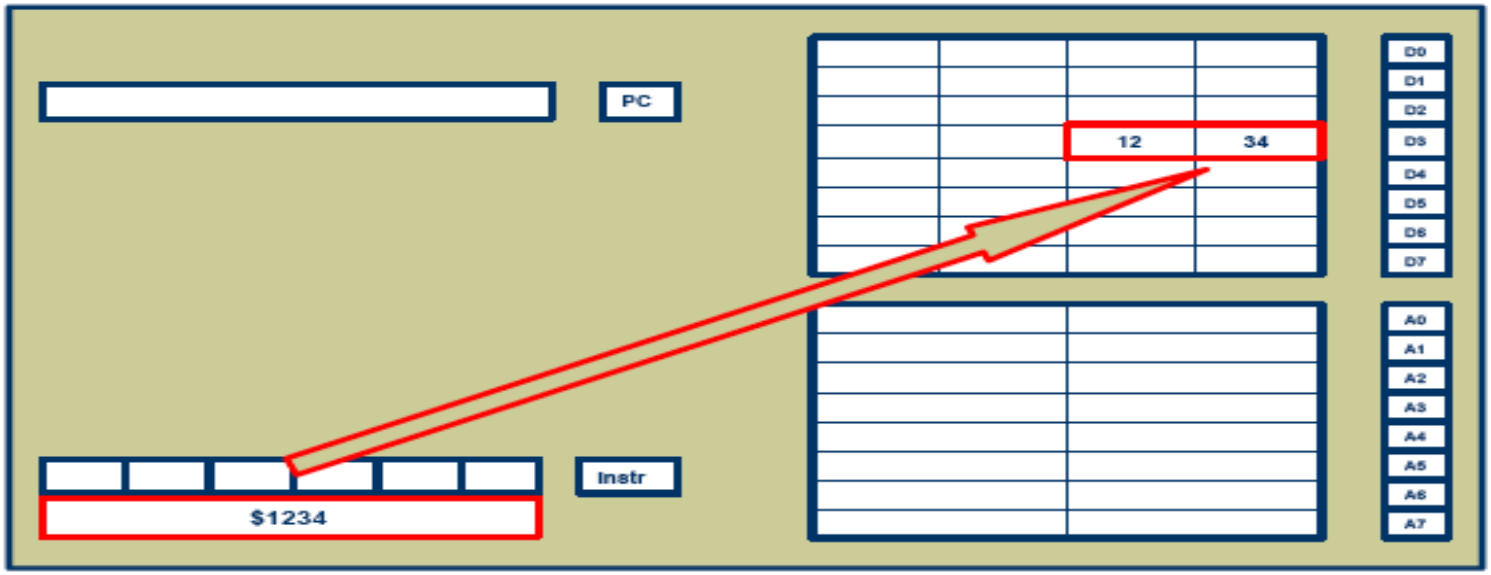
- Transfert simple :
  - MOVE.*(B,W,L)* *<data>, <data->* ***[N,Z,V=0,C=0]***
  - ***Transfert de n'importe quel mode d'adressage colonne <data> → colonne <data->***
  - ***Ex: MOVE.W #35, D3***
- Fanions :
  - ***N, Z*** mis à jour selon la valeur finale de la destination
  - ***V=0, C=0*** V et C toujours à 0 après cette instruction
  - ***X*** pas modifié

- Transfert vers registre d'adresse :
  - MOVEA.*(W,L)* <all>, Aj [-]
  - ***Transfert de n'importe quel mode d'adressage colonne <all> → Aj : registre d'adresse A0..A7***
  - ***Ex: MOVEA.W #$8000, A3***
- Fanions :
  - ***N, Z, V, C, X*** pas modifié [-]
- Transferts multiples
  - MOVEM.*(W,L)* <list>,<ctrl-> [-]
  - MOVEM.*(W,L)* <ctrl>,<list> [-]
  - <list> : A0-A3/D0/D4 → A0, A1, A2, A1, D0, D4
- **Transferts avec la pile (Stack)**
  - MOVEM.*(W,L)* <list>, -(Aj) [-]
  - MOVEM.*(W,L)* (Ai)+, <list> [-]

Modes d' adressage

Les modes d'adressages spécifient comment accéder à l'information

Mode d'adressage Immédiat



Move.w #$1234, D3

→ immédiat

$ → Hexa

Immédiat →registre

Pas modifié

MOVE.B #$12,D0



MOVE.W #$1234,D0

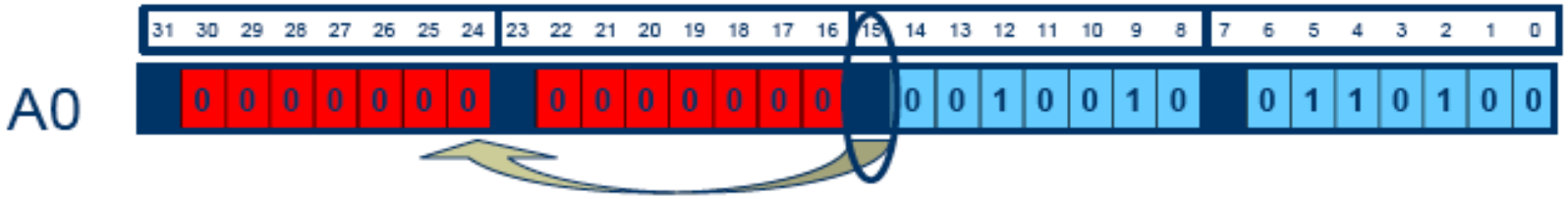


MOVE.L #$12345678,D0



Immédiat →registre d'adresses

MOVEA.W #$1234,A0



MOVEA.W #$9234,A0



MOVEA.L #$12345678,A0

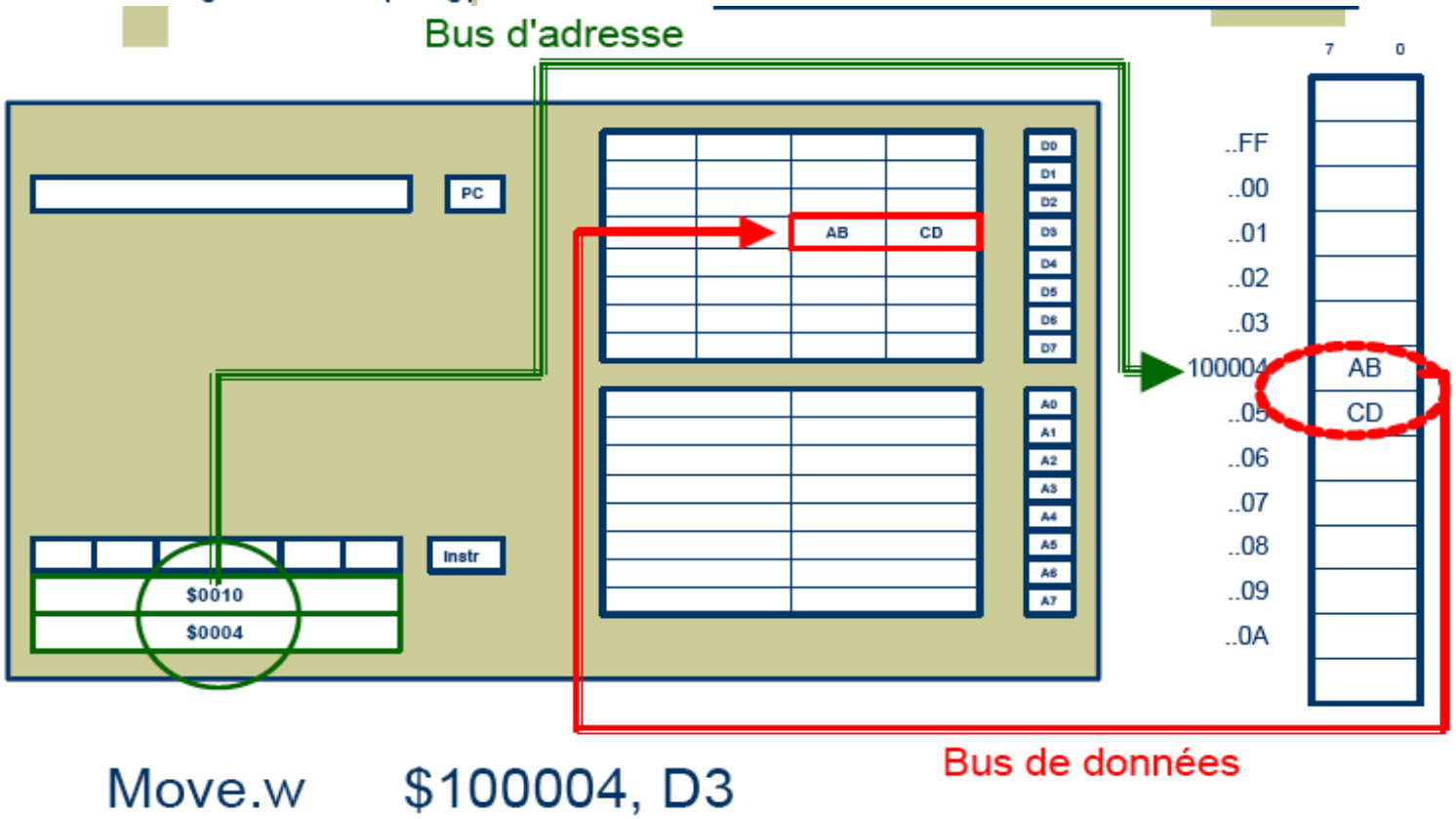


Adressage direct des registres : Dx ou Ax

On opère directement sur les registres soit en lecture, soit en écriture.

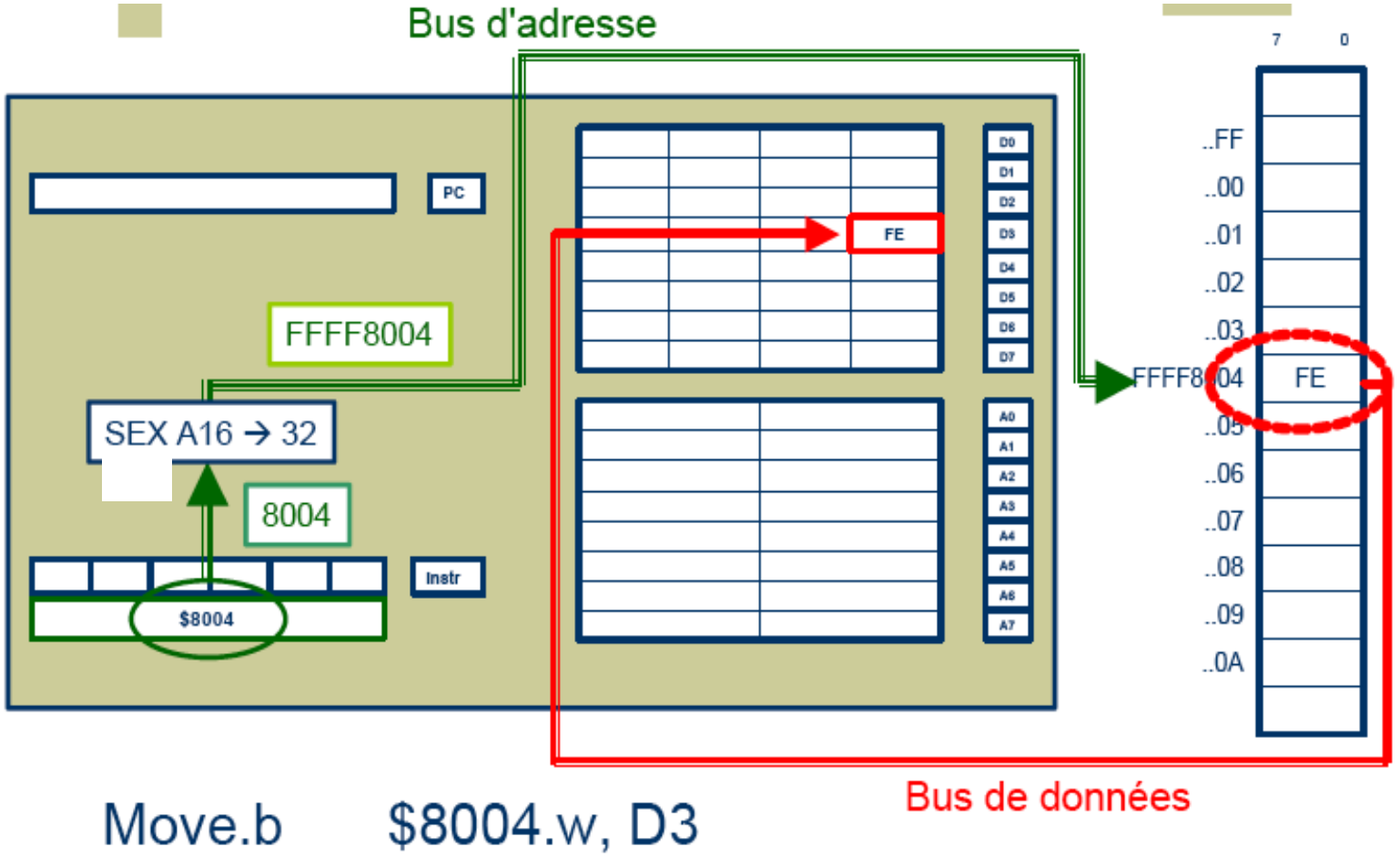
```
MOVE.B    D0,$2000
MOVE.W    A0,A1
MOVE.L    $2000,D0
MOVE.L    A1,$2000
```

Mode d'adressage Absolu (long)



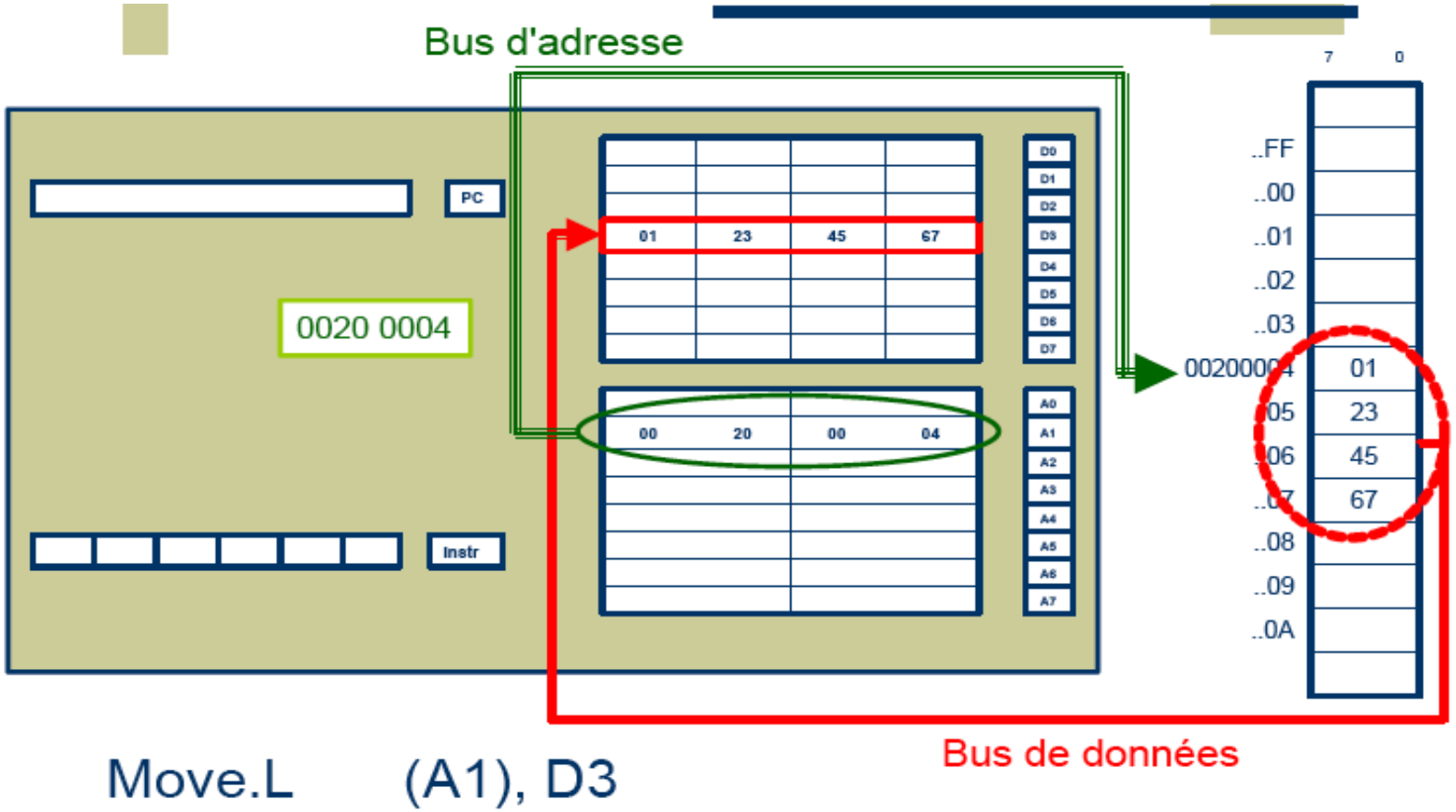
Le contenu de la case $100004 (32 bits) est mis dans D3

MICROPROCESSEURS MOTOROLA (EXEMPLE : LE 68000)



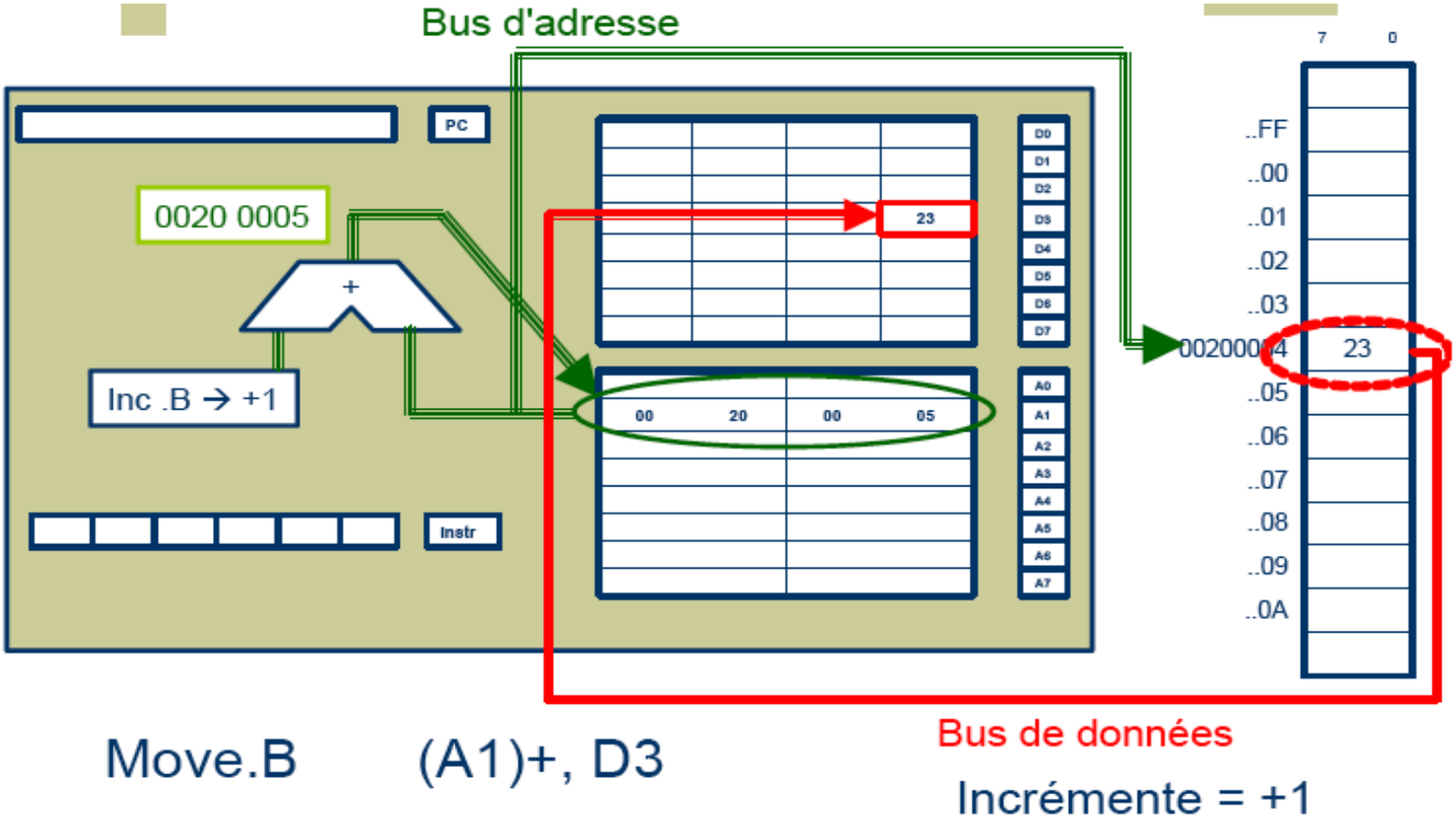
Le contenu de la case $8004 (16 bits) est mis dans D3

L' adresse effective est le contenue du registre d' adresse Ax.



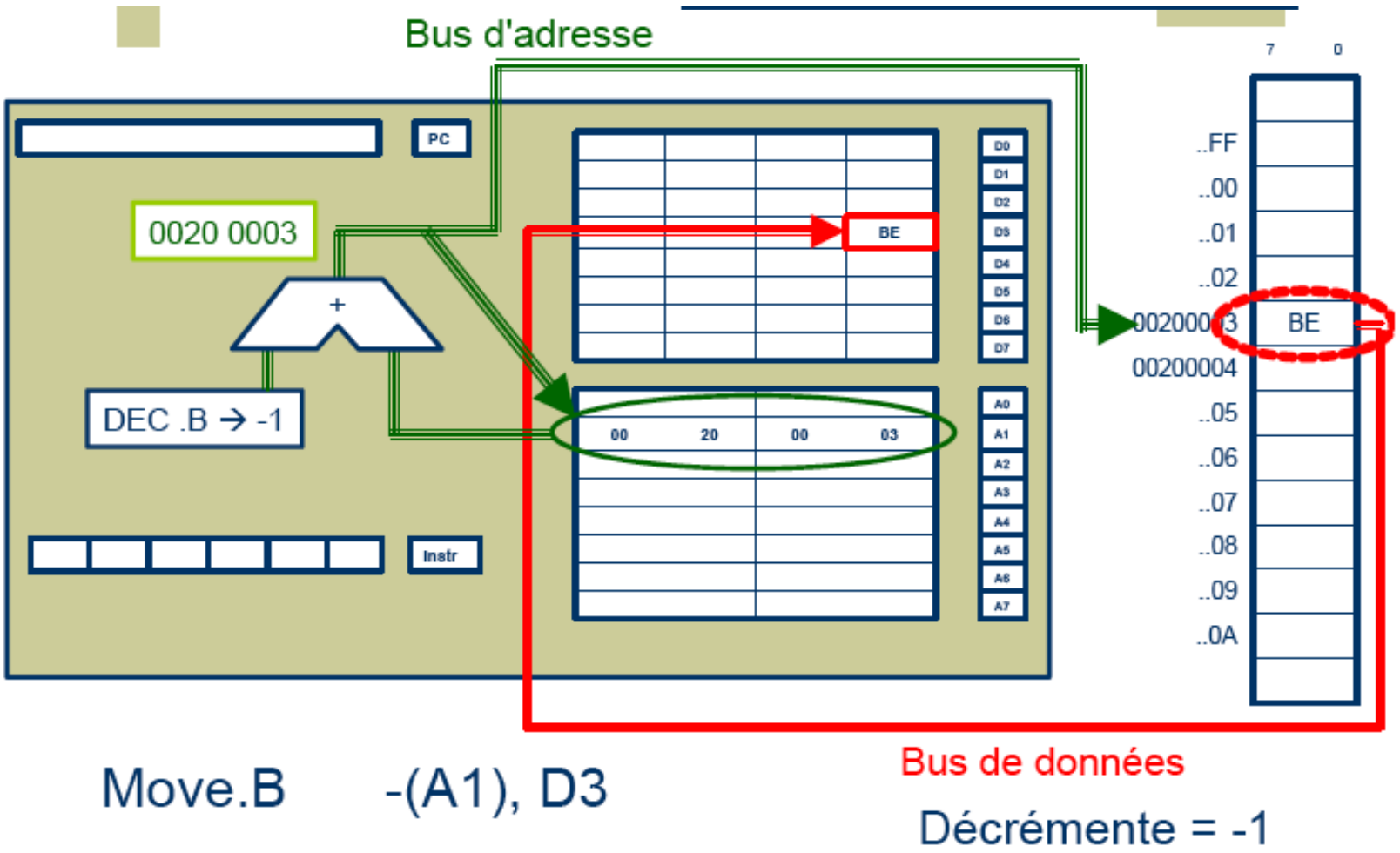
Le contenu de la case mémoire dont l'adresse se trouve dans A1 est mis dans D3

MICROPROCESSEURS MOTOROLA (EXEMPLE : LE 68000)



Le contenu de la case mémoire dont l'adresse se trouve dans A1 est mis dans D3. Le registre A1 est ensuite incrémenté

MICROPROCESSEURS MOTOROLA (EXEMPLE : LE 68000)



Le registre A1 est ensuite décrémenté et le contenu de la case mémoire dont l'adresse se trouve dans A1 est mis dans D3.

Adressage à base : Adressage registre indirect avec déplacement

l'adressage indirect avec déplacement ne modifie pas l'adresse de base Ax

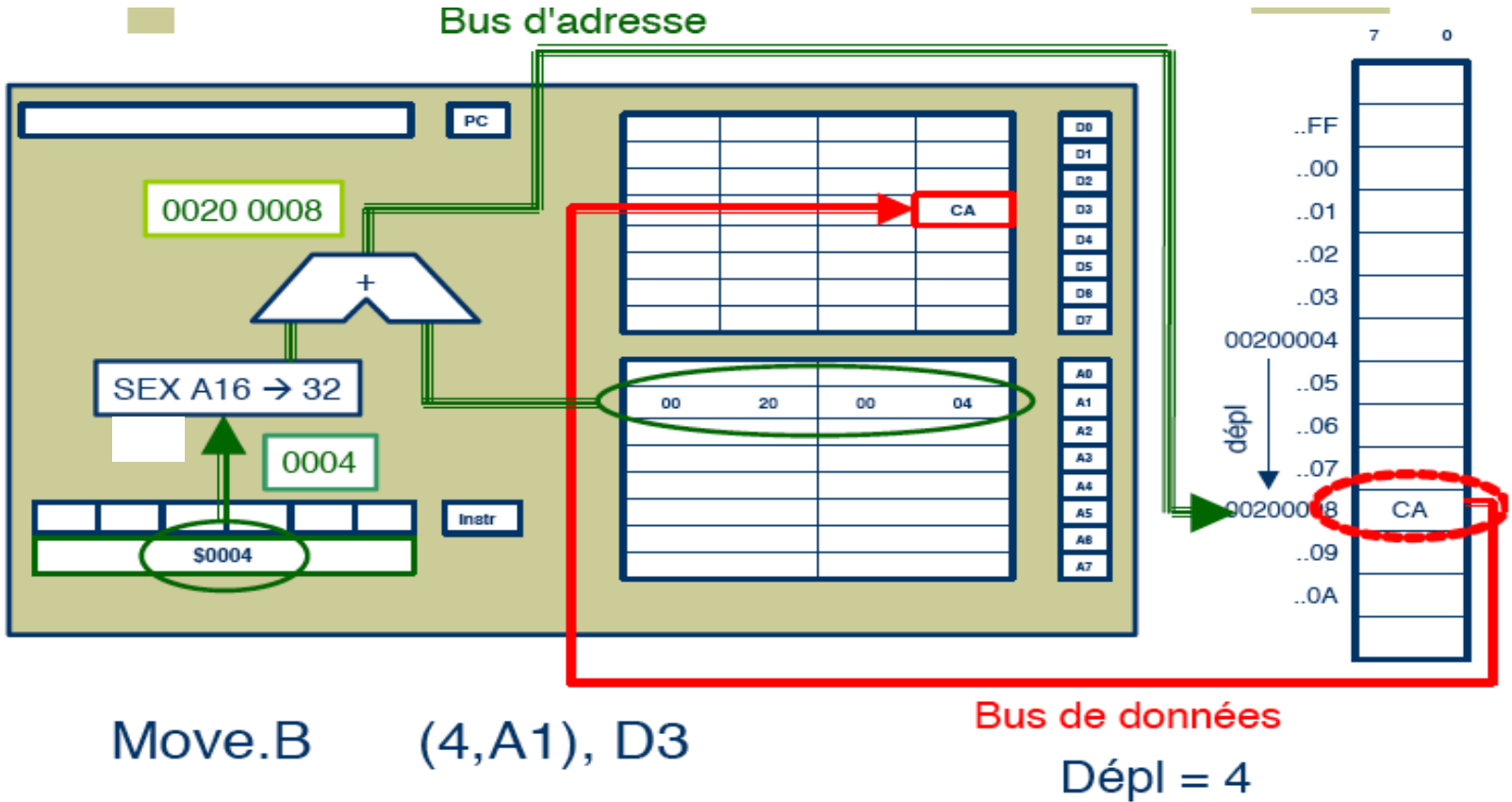
1- Adressage à base (*Adressage registre indirect avec déplacement*) d_{16} (A_x)

L'adresse effective est la somme de (Ac) et de d_{16}, un déplacement signée sur 16 bits compris entre -17784 et +17783.

2- Adressage à base avec index (court/long) (*Adressage registre indirect avec déplacement sur 8 bits signés et index*) d_8 ($A_{x,,}$ X)

L'adresse effective est calculée à partir de l'adresse de base Ax ajouté d'un déplacement signé sur 8 bits et de la valeur de l'index X (un registre de donnée ou un registre d'adresse).

- Exemple (Adressage à base avec index)



Le contenu de la case mémoire dont l'adresse est indiquée par A1+4 est mis dans D3. Le registre A1 est ensuite incrémenté

-Adressage relatif

Dans ce mode d'adressage, l'adresse effective est la somme d'un déplacement fixe dans l'instruction et la valeur courante du compteur de programme.

Ce mode d'adressage ressemble au mode d'adressage à base avec index où les registre de base est PC

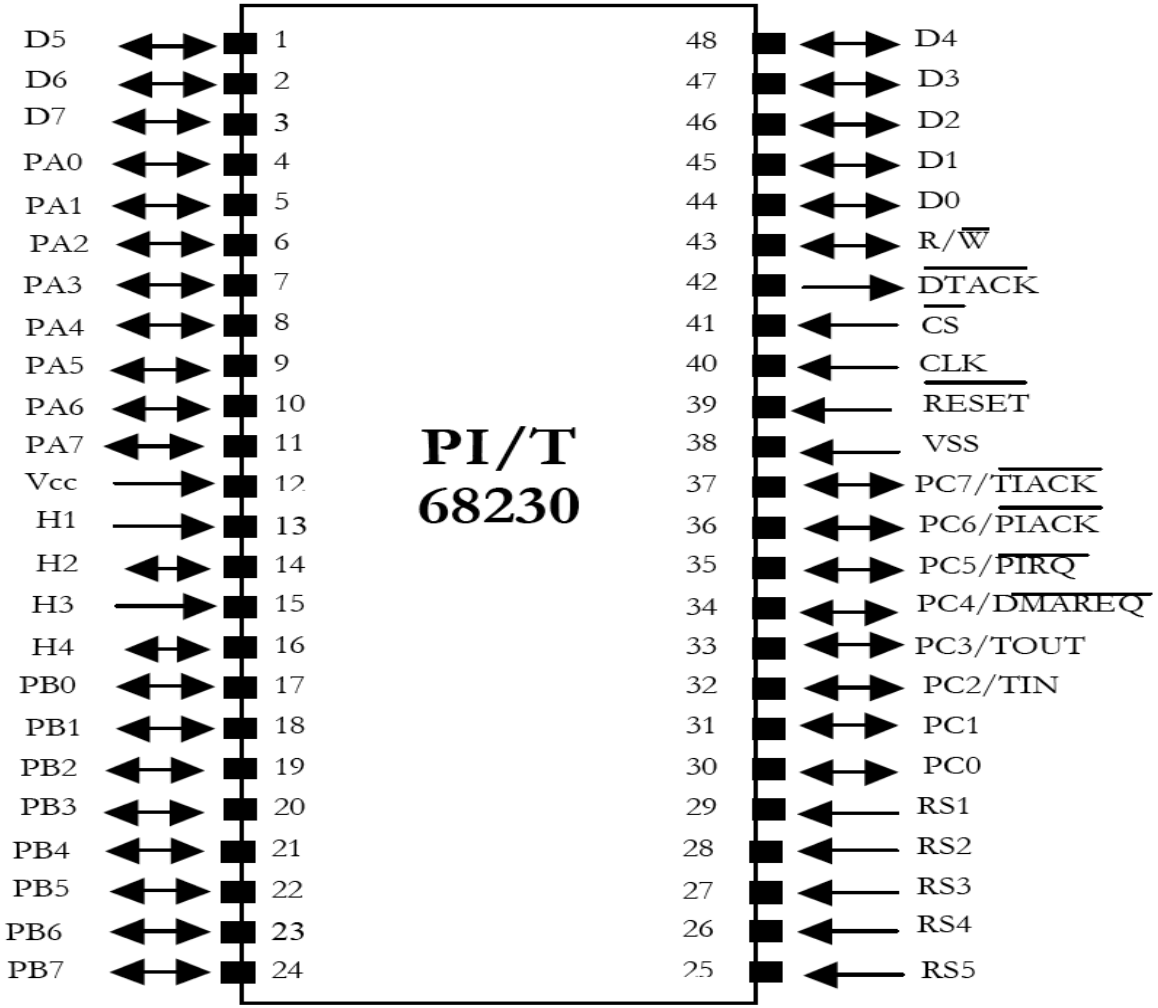
Exemples

BNE Boucle	L'adresse est indiquée par PC + Boucle
MOVE Label(PC), D0	Le contenu de la case mémoire dont l'adresse est donnée par PC+Label est mis dans D0

Exemple : le PIT 86230



86230



MICROPROCESSEURS MOTOROLA (EXEMPLE : LE 68000)

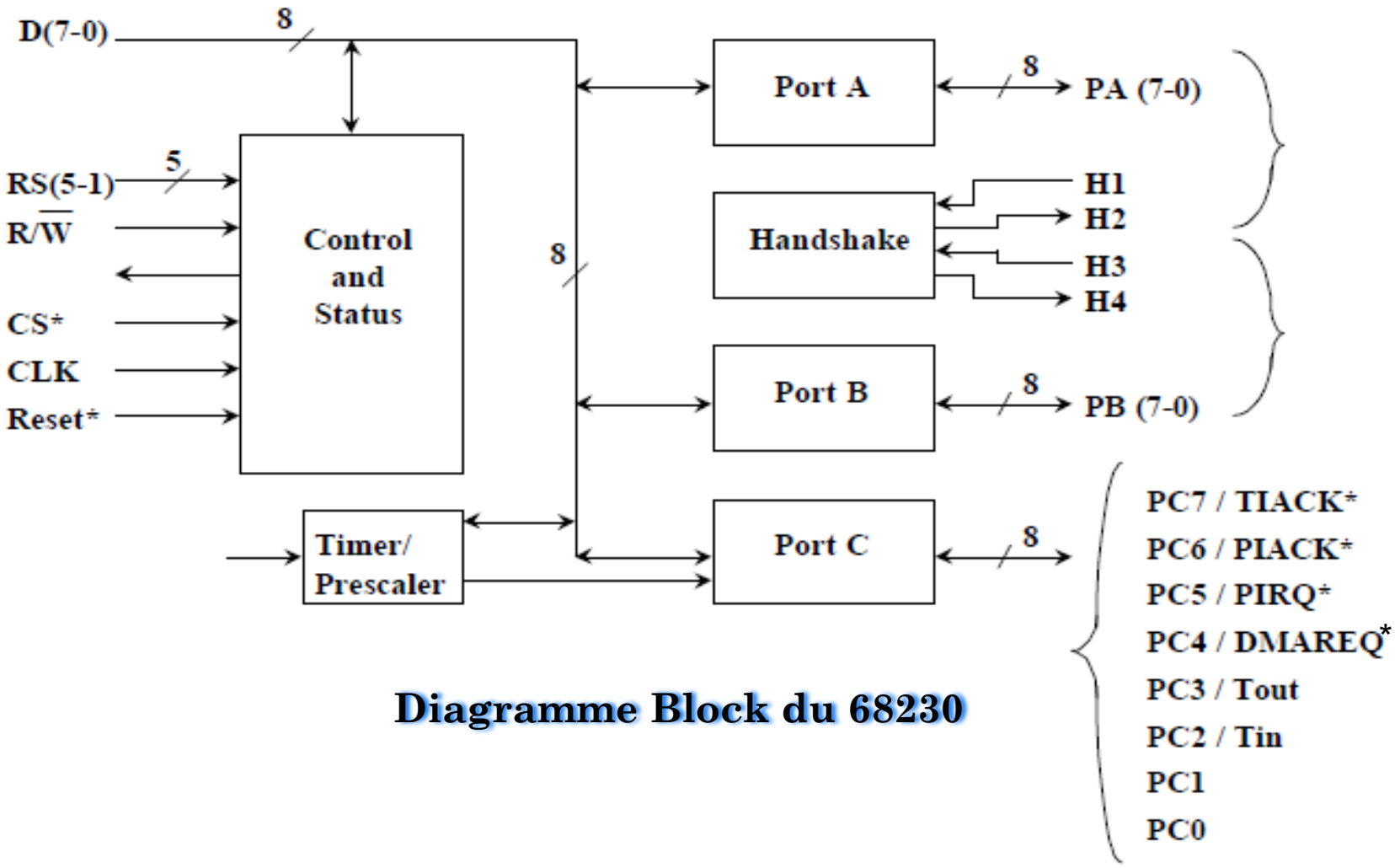


Diagramme Block du 68230

Fonctions du PIT 86230

Fonction PORT

- 24 lignes individuellement programmables en Entrée et en Sortie, groupées autour de 3 ports de 8 lignes (A,B,C).
- Les ports A et B peuvent opérer suivant 4 modes :

- Mode 0 : Double tampons d'entrée 8 bits.
  Double tampons de sortie 8 bits.
  Simples lignes d'entrée/sortie.
- Mode 1 : Double tampon d'entrée 16 bits.
  Double tampon de sortie 16 bits.
- Mode 2 : Simples lignes d'E/S 8 bits.
  Direction déterminée par le matériel.
  Double tampon d'E/S 8 bits.
- Mode 3 : Direction déterminée par le matériel.
  Double tampon d'E/S 16 bits.

- Le port C est un port multifonction (Port et Timer).
- 4 lignes de dialogue.
- Interruptions autovectorisées H1, H2, H3, H4 et vectorisées.

Fonction TIMER

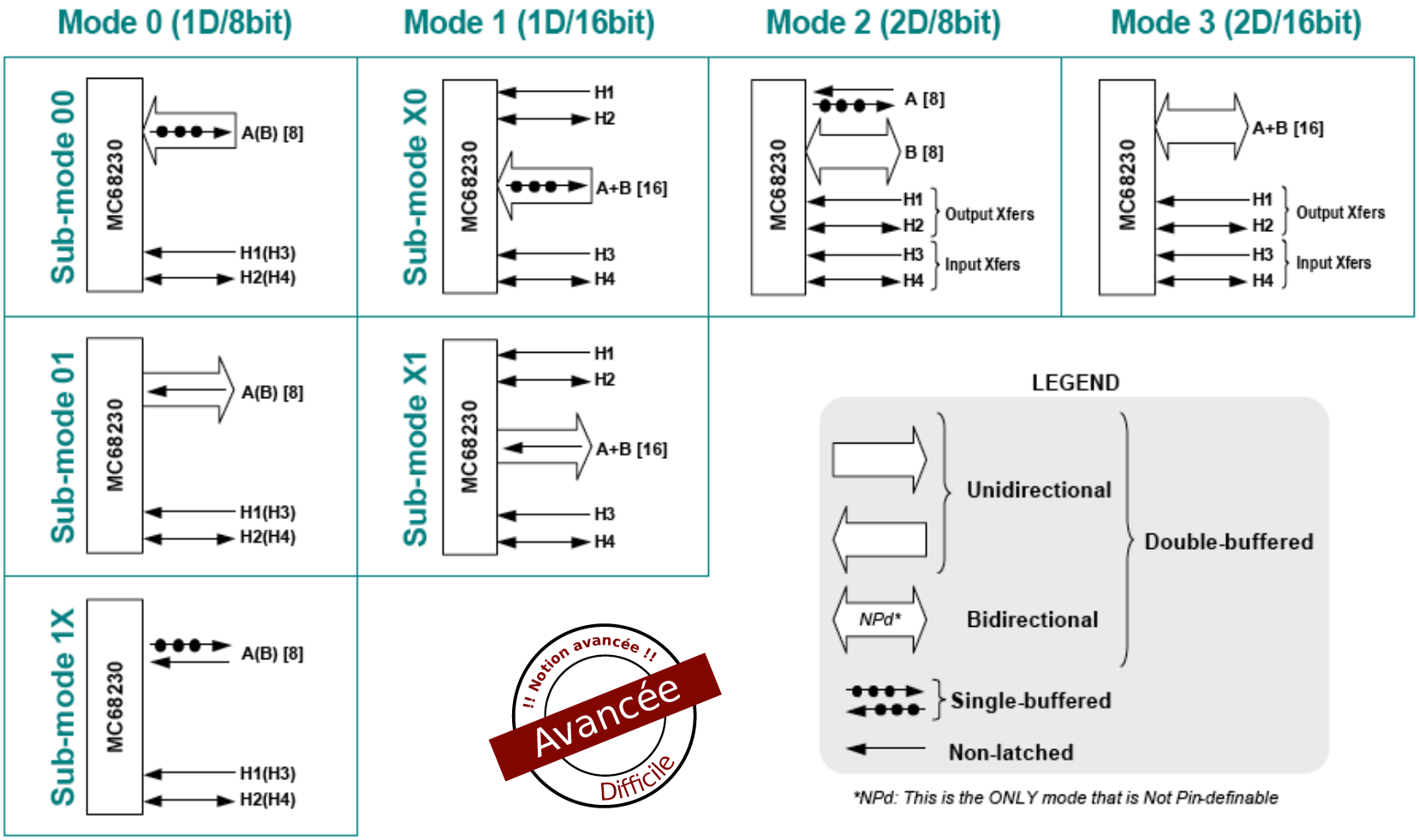
- Temporisateur de 24 bits qui permet de :
- générer périodiquement des interruptions.
- générer une interruption après un délai,
- générer des signaux carrés.
- mesurer un délai écoulé.
- Se comporter en chien de garde.
- Interruptions autovectorisées ou vectorisées.
- Horloge interne prédivisée par 32 (5 bits de prédivision)
- Horloge externe avec ou sans prédivision par 32.

Registres du PIT 86230

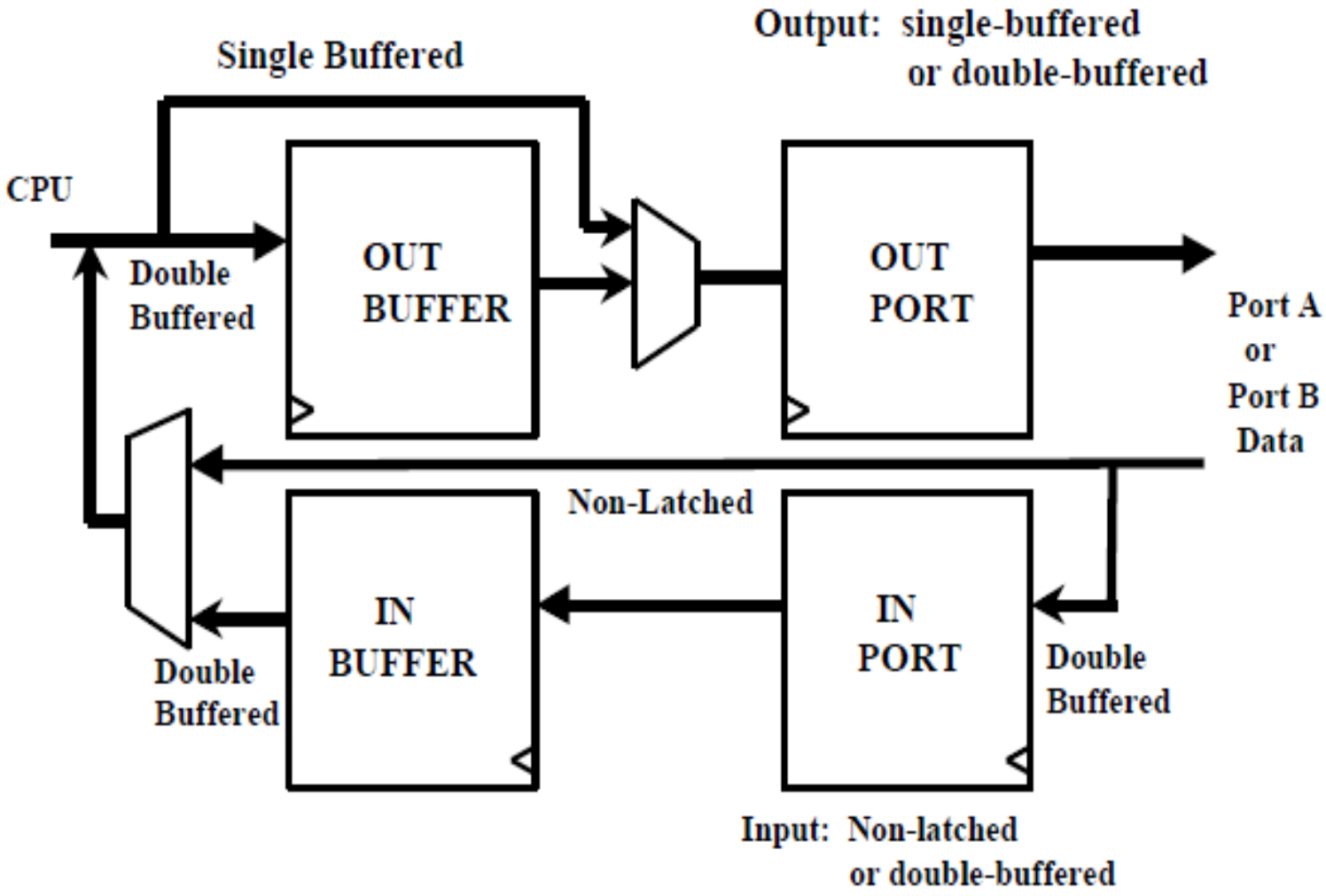
Address	Port	Function
$800001	PGCR	Port general control
$800003	PSRR	Port service request
$800005	PADDR	Port A data direction
$800007	PBDDR	Port B data direction
$800009	PCDDR	Port C data direction
$80000B	PIVR	Port Interrupt vector
$80000D	PACR	Port A Control
$80000F	PBCR	Port B Control
$800011	PADR	Port A Data
$800013	PBDR	Port B Data
$800015	PAAR	Port A Alternate
$800017	PBAR	Port B Alternate

Address	Port	Function
$800019	PCDR	Port C Data
$80001B	PSR	Port register
$800021	TCR	Timer control
$800023	TIVR	Timer interrupt vector
$800027	CPRH	Counter preload - high
$800029	CPRM	Counter preload - medium
$80002B	CPRL	Counter preload - low
$80002F	CNTRH	Count register - high
$800031	CNTRM	Count register - medium
$800033	CNTRL	Count register - low
$800035	TSR	Timer Status

Modes d'adressage du PIT 86230



Modes d'adressage du PIT 86230



Registre de contrôle général du port (PGCR)

- **PGCR7-PGCR6**
  - Select the operating mode of the PI/T
- **PGCR5-PGCR4**
  - Enables the handshake pairs H3-H4 and H1-H2. These bits have to be set before we can make use of the control inputs and outputs. Doing this avoids spurious operation of the handshake lines before the PI/T has been fully configured
- **PGCR3-PCGR0**
  - Determine the *sense* of the four handshake lines. These control lines can be programmed to be active-low or active-high

Bit	PGCR7	PGCR6	PGCR5	PGCR4	PGCR3	PGCR2	PGCR1	PGCR0
Function	Port mode control		H34 enable	H12 enable	H4 sense	H3 Sense	H2 Sense	H1 Sense

PGCR7	PGCR6	Port mode control
0	0	Mode 0
0	1	Mode 1
1	0	Mode 2
1	1	Mode 3

PGCR5	H3, H4 control
0	H34 disable
1	H34 enable

PGCR4	H1, H2 control
0	H12 disable
1	H12 enable

PGCR3-0	Sense (Assertion Level)
0	LOW
1	HIGH

Registre de contrôle général du port (PGCR)

PGCR7 PGCR6	PGCR5	PGCR4	PGCR3	PGCR2	PGCR1
P Port Mode Control 00 (Mode 0) 01 (Mode 1) 10 (Mode 2) 11 (Mode 3)	H34 Enable	H12 Enable	H4 sense	H3 sense	H2 sense	H1 sense
	0 Disable 1 Enable		0 Active low 1 Active high		

Example:

PGCR = %00 11 0000

Means:

Mode 0, Unidirectional 8-bit, separate PA & PB
Both H34 and H12 handshaking enabled
H4-H4 active low

Registre (PACR) en Mode 0

PACR7 PACR6	PACR5 PACR4 PACR3	PACR2	PACR1 PACR0
Submode: 00 submode 0 01 submode 1 10 submode 1x	H2 Control 0xx Edge-sensitive input 100 output- negated 101 output - asserted 110 output - interlocked Handshake 111 Output - pulsed handshake	H2 Interrupt 0 Disabled 1 Enabled	H1 Control 0X H1 interrupt disabled 10 H1 interrupt enabled XX

Example: PACR = %00000000
PADDR = %00000000

Means: Port A is used as an input port
Submode 0 (Double Buffered input)
H2 Edge-sensitive
H2 interrupt disabled
H1 interrupt disabled

Registre (PBCR) en Mode 0

PBCR7 PBCR6	PBCR5 PBCR4 PBCR3	PBCR2	PBCR1 PBCR0
Submode: 00 submode 0 01 submode 1 10 submode 1x	H4 Control 0xx Edge-sensitive input 100 output- negated 101 output - asserted 110 output - interlocked Handshake 111 Output - pulsed handshake	H4 Interrupt 0 Disabled 1 Enabled	H3 Control 0X H3 interrupt disabled 10 H3 interrupt enabled XX

Example: PBCR = %00000000
PBDDR = %11111111

Means: Port B is used as an output port
Submode 1 (Double Buffered output)
H4 Edge-sensitive
H4 interrupt disabled
H3 interrupt disabled

Exercice 6

Donnez un programme assembleur 68000 permettant de décomposer le mot sauvegardé à l'adresse $E01050 comme suit :

- les bits 0-5 sont copiés dans le mot mémoire d'adresse $E01060,
- les bits 6-7 et 11-12 sont copiés dans le mot mémoire d'adresse $E01070,
- les bits 13-15 sont copiés dans le mot mémoire d'adresse $E01080,
- le reste des bits sont copiés dans le mot mémoire d'adresse $E01090.

  Le programme met à zéro le reste des bits des mots mémoire de destination.

Exercice 7

Donnez un programme 68000 permettant de donner le minimum (un octet) d'un tableau de $100 octets débutant à l'adresse $400500. Ce programme enregistre le minimum à l'adresse $400600

Exercice 8

On se propose de commander un feu de carrefour à l'aide d'un système à base du µP 68000 de Motorola. C'est un carrefour possède 2 voies (A et B). Chaque voie est munie de trois feux (V, O et R) pour les automobiles.

Le tableau suivant donne l'affectation des lampes du feu de carrefour aux lignes du port PA du PI/T.

PA1	Rouge	Panneau A
PA2	Orangée	Panneau A
PA3	Vert	Panneau A
PA5	Rouge	Panneau B
PA6	Orangée	Panneau B
PA7	Vert	Panneau B

1- En se servant du tableau d'adresse des registres du PI/T suivant, donnez un programme assembleur 68000 qui donne un allumage normal des feux (vert puis orangé puis rouge), sachant qu'un feu rouge dure 40 secondes, un feu vert dure 30 secondes et un feu orangé dure 10 secondes.

MICROPROCESSEURS MOTOROLA (EXEMPLE : LE 68000)

Registre	Abréviation	Adresse	Offset
Registre de contrôle général du port	PGCR	$800001	$00
Registre de demande de service du port	PSRR	$800003	$02
Registre de direction du port A	PADDR	$800005	$04
Registre de direction du port B	PBDDR	$800007	$06
Registre de direction du port C	PCDDR	$800009	$08
Registre vecteur d'interruption du port	PIVR	$80000B	$0A
Registre de contrôle du port A	PACR	$80000D	$0C
Registre de contrôle du port B	PBCR	$80000F	$0E
Registre de données du port A	PADR	$800011	$10
Registre de données du port B	PBDR	$800013	$12
Registre alterné du port A	PAAR	$800015	$14
Registre alterné du port B	PBAR	$800017	$16
Registre de données du port C	PCDR	$800019	$18
Registre d'état du port	PSR	$80001B	$1A
Registre de contrôle du Timer	TCR	$800021	$20
Registre vecteur d'interruption du Timer	TIVR	$800023	$22
Registre de préchargement du compteur Haut	CPRH	$800025	$26
Registre de préchargement du compteur Moyen	CPRM	$800027	$28
Registre de préchargement du compteur Bas	CPRL	$80002B	$2A
Registre de comptage Haut	CNTRH	$80002F	$2E
Registre de comptage Moyen	CNTRM	$800031	$30
Registre de comptage Bas	CNTRL	$800033	$32
Registre d'état du Timer	TSR	$800035	$34

Exercice 2 :

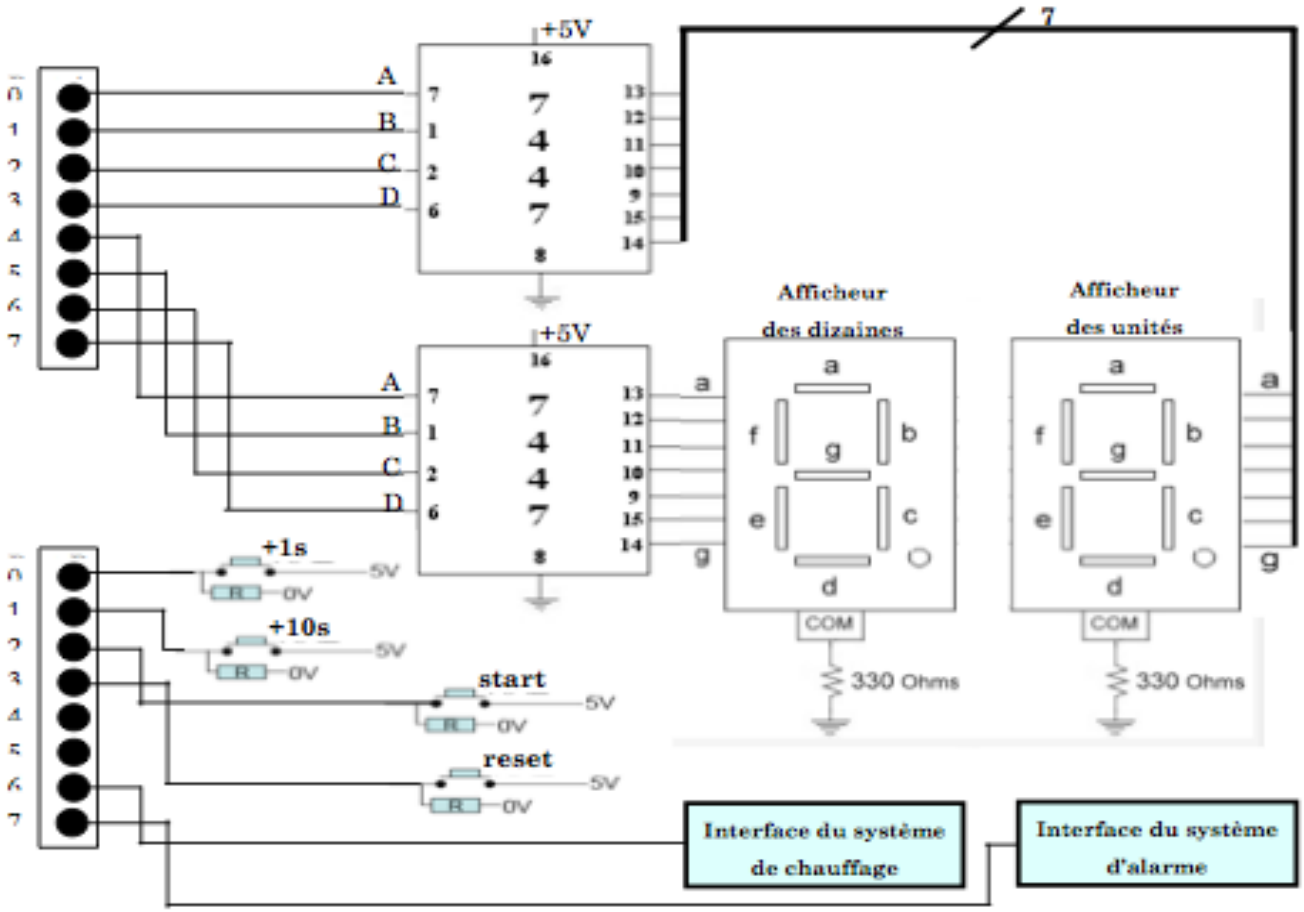
Il s'agit de réaliser un système destiné à la préparation de quelques produits biochimiques et chimiques au laboratoire. On se limitera dans cet exercice à l'étude de dispositif de chauffage. Ce dispositif est commandé par un système à base du microprocesseur 68000 à travers le circuit d'interface parallèle du type PI/T 68230.

Avec ce dispositif, l'utilisateur sélectionne un temps de chauffage exprimé en secondes, à l'aide de 2 boutons '***+1s***' et '***+10s***'. Puis, après l'appui d'un bouton '***start***', le chauffage démarre tandis que le temps restant à s'écouler est affiché sur une minuterie constituée de 2 afficheurs 7 segments. Si le temps choisi par l'utilisateur est écoulé, un petit système d'alarme est déclenché pendant 2 secondes. Un bouton '***reset***' permet d'arrêter et de remettre la minuterie à zéro.

Les caractéristiques de cette application sont les suivantes :

- Les broches 0 à 3 du port A (de PI/T 68230) sont connectées au premier afficheur 7 segments (afficheur des unités) par l'intermédiaire d'un décodeur BCD/7 Segments, le IC7447.
- Les broches 4 à 7 du port A sont connectées au deuxième afficheur 7 segments (afficheur des dizaines) par l'intermédiaire d'un autre décodeur de même type.
- Les broches 0 à 3 du port B sont connectées à 4 boutons permettant à l'utilisateur de programmer le temps de cuisson, de démarrer la cuisson ou d'interrompre celle-ci. Ils sont à l'état logique 0 au repos et à 1 lorsqu'ils sont appuyés.
- Les broches 6 et 7 du port B commandent respectivement le système de chauffage et celui d'alarme (0 : arrêter, 1 : Activer).

Voici un schéma de synthèse du circuit, utilisé par ce dispositif.



Le programme principal appelle un sous-programme d'initialisation des registres de direction et de contrôle de deux ports A et B de PI/T. Donnez ce sous-programme qui débute à l'adresse '*Init*' en se referant au tableau suivant.

```
.....
.....
.....
.....
.....
.....
.....
.....
```

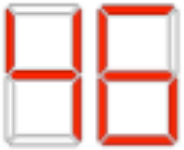
Registre	Abréviation	Adresse	Offset
Registre de contrôle général du port	PGCR	$800001	$00
Registre de direction du port A	PADDR	$800005	$04
Registre de direction du port B	PBDDR	$800007	$06
Registre de contrôle du port A	PACR	$80000D	$0C
Registre de contrôle du port B	PBCR	$80000F	$0E
Registre de données du port A	PADR	$800011	$10
Registre de données du port B	PBDR	$800013	$12
Registre alterné du port A	PAAR	$800015	$14
Registre alterné du port B	PBAR	$800017	$16

ENIG

Dans tous ce qui suit, on suppose que les registres de données D0 et D1 reflètent respectivement les valeurs affichées ou qui seront affichées par l'afficheur des unités et celui des dizaines.

c-à-d :

Si D0 contient \$06 et D1 contient \$04, la minuterie affiche la valeur suivante



Proposez un sous-programme qui débute à l'adresse *'Display'* permettant d'afficher sur la minuterie les valeurs des unités et des dizaines lus respectivement à partir de D0 et D1. Ce sous-programme fait appel à un autre sous-programme de temporisation d'une seconde qui débute à l'adresse *'Delay'* pour pouvoir visualiser chaque changement. Donnez aussi ce sous-programme de temporisation.

Display.....
.....
.....
.....
.....
.....
.....

Delay.....
.....
.....
.....
.....
.....
.....

Complétez le programme principal, suivant, qui débute à l'adresse $400400, en tenant compte des indications suivantes :

* Vous pouvez éventuellement changer la partie du programme donnée si elle ne correspond pas à votre propre choix des registres.
* Si l'utilisateur appuie sur le bouton '***+1s***' et que l'afficheur des unités affiche la valeur 9, celui ci est mise à zéro tandis que l'afficheur des dizaines est incrémenté de 1 tant que ce dernier est inférieure à 9. Si la minuterie contient la valeur 99, tout appuis sur le bouton '***+1s***' sera ignoré par le microprocesseur.
* Si l'utilisateur appuie sur le bouton '***+10s***' et que l'afficheur des dizaines affiche la valeur 9, cet appuie sera ignoré par le microprocesseur.

```
                Org $400400
                BSR Init
                CLR.B D0
                CLR.B D1
Test_Reset      BSR  Display
                MOVE.B $12(A0),D3
                BTST.B   #03,D3
                BEQ      Test_Plus1
                CLR.B D0
                CLR.B D1
                BCLR   #06, $12(A0)
Test_Plus1      BSR  Display
.....
Test_Plus10     BSR  Display
.....
Test_Start      BSR  Display
.....
Loop            BRA   Test_Reset
```

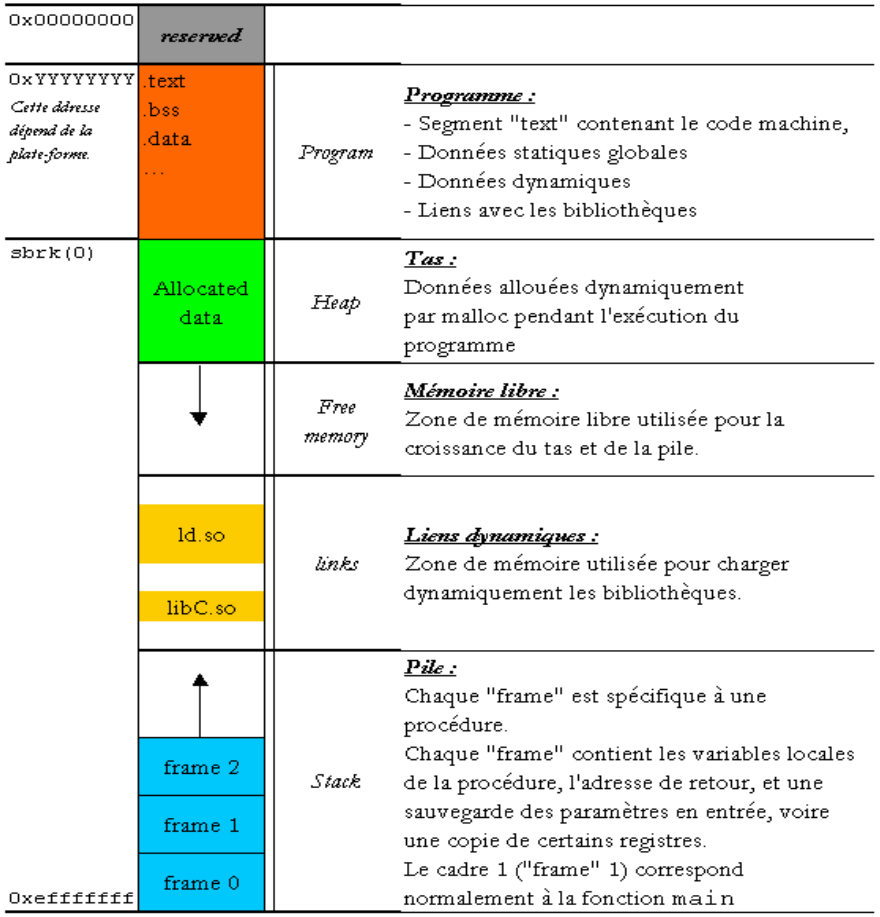
Quelques approfondissements sur le processeur 68000

États du processeurs - code fonction

FC2	FC1	FC0	Etat	Domaine Mémoire
1	1	0	Non utilisé	
0	0	1	Utilisateur	Données
0	1	0	Utilisateur	Programme
0	1	1	Non utilisé	
1	0	0	Non utilisé	
1	0	1	Superviseur	Données
1	1	0	Superviseur	Programme
1	1	1	Reconnaissance d'interruption	

(FC0, FC1, FC2 : Function Code, Logique trois états) : Ces trois sorties sont positionnées par le processeur dès le début d'un cycle bus. Elles communiquent à la périphérie l'état du processeur. Elles indiquent en particulier si le processeur fonctionne en mode SUPERVISEUR (FC2=1) ou en mode UTILISATEUR (FC2=0), si l'information en cours d'exécution est du type donnée ou du type programme ou si le processeur se trouve en reconnaissance d'interruption

utilisation de la mémoire pour chaque processus

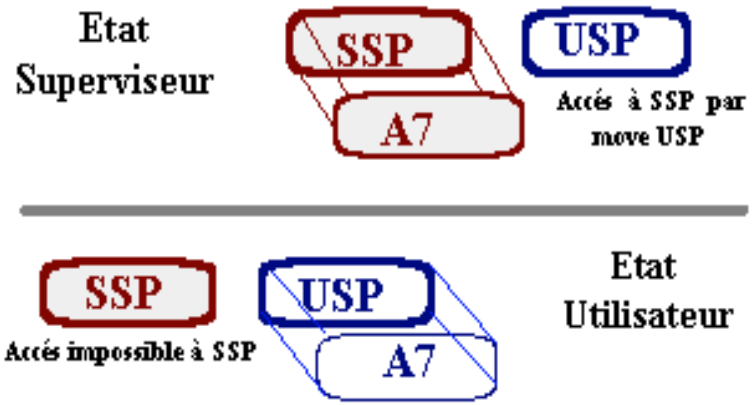


Quelques approfondissements sur le processeur 68000

Le registre A7

La pile est une zone mémoire accessible en lecture-écriture qui permet de sauvegarder un contexte. Le processeur dispose d'un pointeur de pile utilisateur USP et d'un pointeur de pile superviseur SSP.

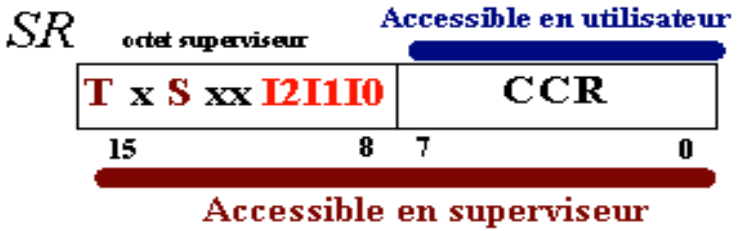
Le registre A7 contient le pointeur de pile correspondant à l'état courant du processeur. Il est impossible d'accéder à l'autre registre pointeur de pile par A7. (En superviseur l'accès à USP est toutefois possible grâce à l'instruction privilégiée move USP).



Quelques approfondissements sur le processeur 68000

Le registre d'état SR

Le registre comprend 16 bits.



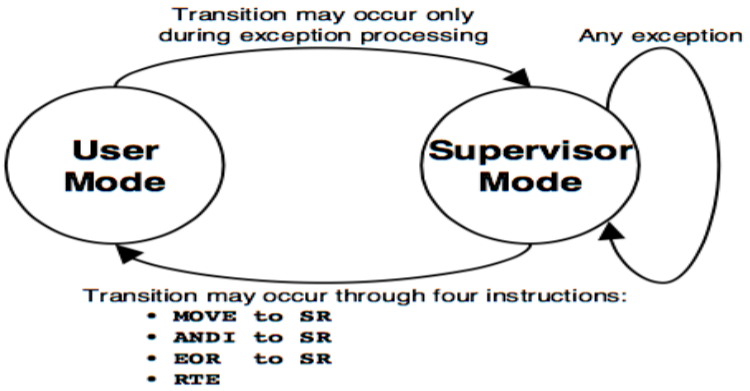
La partie basse, contient les drapeaux (Z, N, C, ...). Elle nommée "CCR" et parfois improprement appelée "octet utilisateur". Elle est accessible dans les deux états.

La partie haute (bits 15 à 8) du registre d'état, dite "octet superviseur", est accessible (modifiable) que dans l'état Superviseur. Les bits de cette partie du SR sont liés aux interruptions.

- Le bit **T**, s'il est à un "1", à chaque instruction exécutée par le processeur, une interruption est créée. Elle est utilisée par les outils de mise au point des programmes.
- Le bit **S** renvoie l'état actuel du processeur : S="0" USER et S="1" SUPER. Il permet aussi de basculer l'état du processeur en USER par une opération d'écriture S <= "0". Cette opération est seulement possible lorsque le processeur est en superviseur.
- Les bits **I2 à I0** permettent de masquer les niveaux d'interruptions matérielles.

Quelques approfondissements sur le processeur 68000

	User mode	Supervisor mode
Enter the mode by	Clear S bit in status register	Executing TRAP instruction, privilege violation, reset, interrupt
FC2 pin on device (same as S bit)	0	1
Default Stack Pointer	USP	SSP
Access to SR	Read whole SR Write to CCR only	Read whole SR Write whole SR
Usable instructions	All instructions, except: **RESET** **RTE** **STOP** **ANDI #d, SR** **EORI #d, SR** **ORI #d, SR** **MOVE <ea>SR** **MOVE An,USP**	All instructions



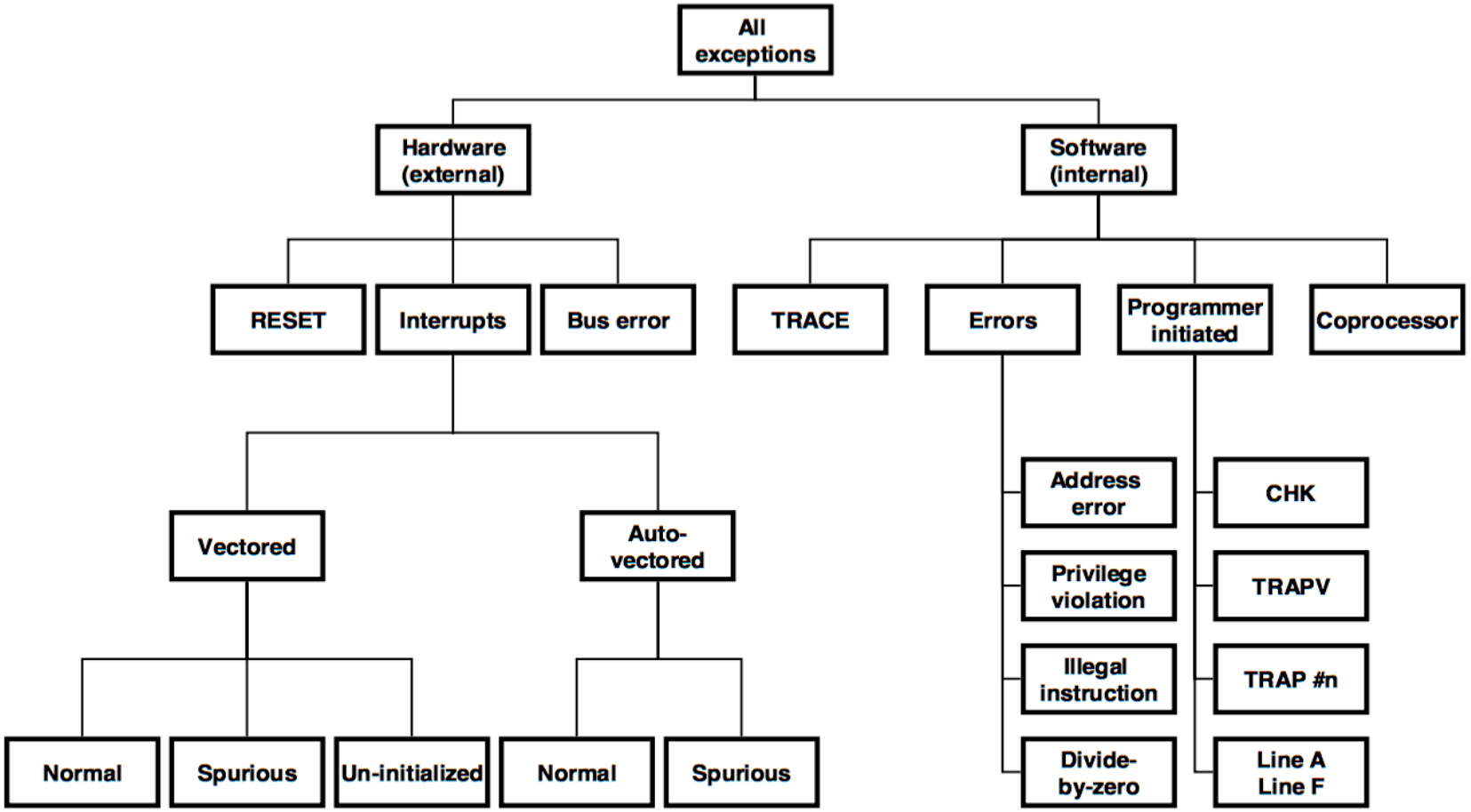
Jump to an exception handler

```
[temp_register]     ← [SR]
[S]                 ← [1]
[T]                 ← [0]
get_vector_number
exception_vector    ← vector_number×4
handler_address     ← [M(exception_vector)]
[SSP]               ← [SSP]-4
[M(SSP)]            ← [PC]
[SSP]               ← [SSP]-2
[M(SSP)]            ← [temp_register]
[PC]                ← handler_address
```

Return from an exception handler

```
[SR]                ← [M(SSP)]
[SSP]               ← [SSP]+2
[PC]                ← [M(SSP)]
[SSP]               ← [SSP]+4
```

Quelques approfondissements sur le processeur 68000



Quelques approfondissements sur le processeur 68000

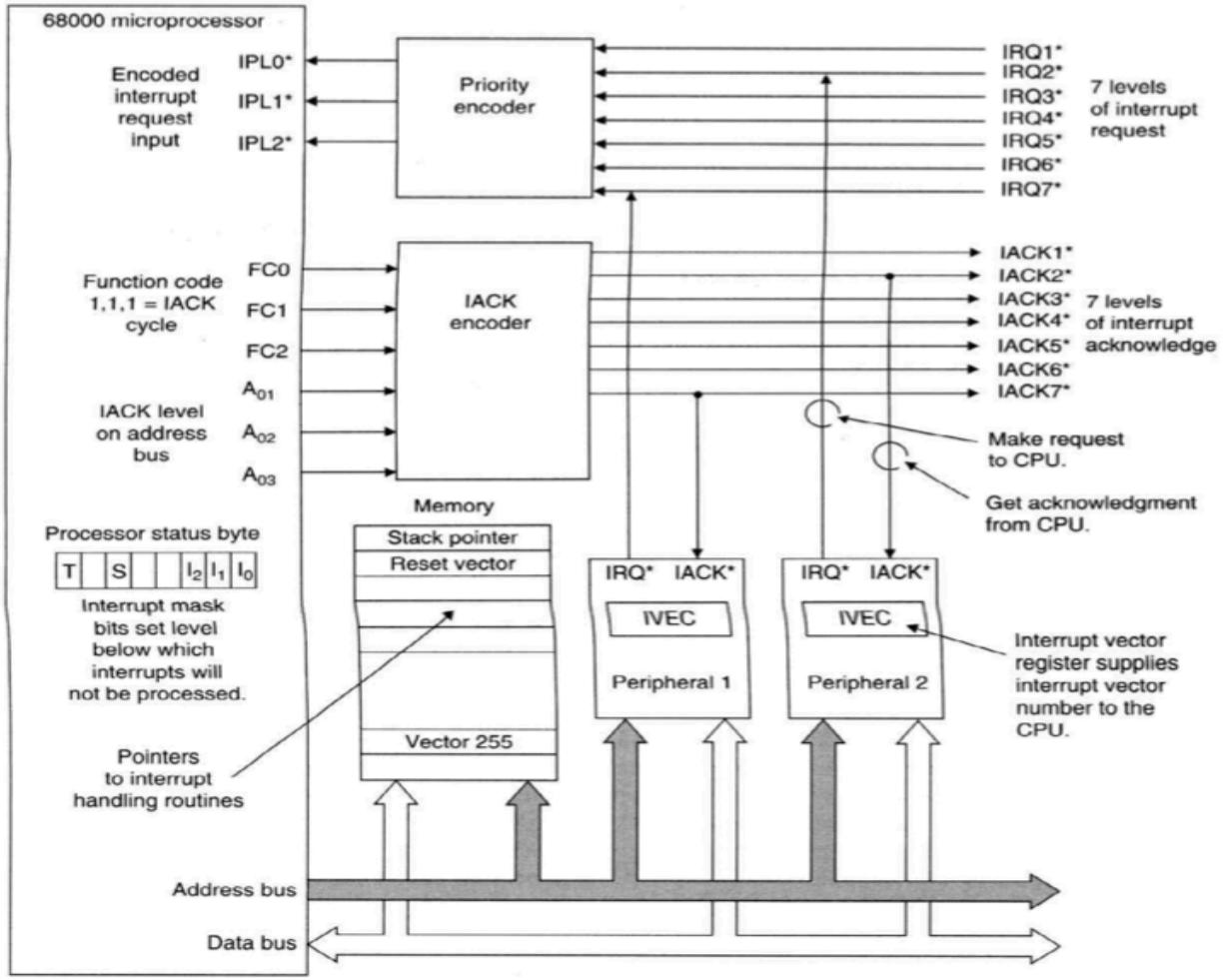
Les interruptions

- L'interruption initialisation, Vecteur 0 et 1. Elle se produit quand on active la borne RESTE du processeur. Il se rend à la position mémoire 0h charger la valeur du pointeur de pile superviseur puis à l'adresse 4h charger l'adresse de la première instruction à exécuter. Il se trouve à l'issue de cette opération dans l'état superviseur.
- L'interruption erreur de bus, Vecteur 2. Elle se produit quand on active la borne BEER. Cette borne est liée au chien de garde qui surveille le bon fonctionnement du protocole asynchrone à l'occasion d'un transfert de données.
- L'interruption erreur d'adresse, Vecteur 3. Elle est crée par le processeur lorsqu'il rencontre une instruction qui présente un défaut d'alignement entre les adresses et la taille des données. Ceci se produit, par exemple, qu'en on transfère un objet de 16 bits à une adresse impaire.
- L'interruption TRAP #i, Vecteurs 32 à 47. C'est une interruption qui apparaît quand le processeur exécute cette instruction. On dispose ainsi de 16 interruptions logiciel TRAP #0 à TRAP #15. Ces interruptions sont utilisées par les commandes du système d'exploitation.
- L'interruption violation de privilège, Vecteur 8. Elle se produit quand le processeur exécute une instruction privilégiée alors qu'il est dans l'état Utilisateur.
- Les interruptions demandées par les composants périphériques, Vecteurs 25 à 31 et 64 à 255. Les demandes sont réalisées sur les bornes IPL du processeur. Pour les vecteurs 25 à 31, la valeur est calculée par le processeur à partir du niveau d'interruption demandé sur les bornes IPL (Fonctionnement dit" Autovectorisé"). Par contre pour les vecteurs supérieurs à 64, la valeur est fournie au processeur sur le bus données bas (Fonctionnement dit" Vectorisé").

N ° Vecteur	Vecteur	
0	$000	Reset (Pointeur de pile - SSP)
1	$0004	Reset (Pointeur de programme - PC)
2		Erreur bus
3	$00C	Erreur d'adresse
4	$010	Instruction illégale
5	$014	Division par 0
6	$018	Instruction CHK
7	$01C	Instruction TRAPV
8	$020	Violation de privilège
9	$024	Trace
10	$028	Émulation 1010
11	$02C	Émulation 1111
12 à 14	$030 à $038	Non affecté, ne pas utiliser (68000)
15	$03C	Interruption non initialisée (valeur mise par défaut dans le registre VR des composants périphériques à l'issue du Reset matériel)
16 à 23	$040 à $05C	Non affecté, ne pas utiliser (68000)
24	$060	Interruption parasite
25 à 31	$064 à $07C	Auto-vecteur Niveau 1 à Niveau 7 pour les iT matérielles
32 à 47	$080 à $0BC	TRAP 0 à TRAP 15
48 à63	$0C0 à $0FC	Non affecté, ne pas utiliser (68000)
64 à 255	$100 à $3FC	Interruptions matérielles

Quelques approfondissements sur le processeur 68000

Les interruptions



ASSIGNMENT	RAM ADDRESS
Autovector #1	$400016
Trap #0	$400040
Trap #1	$400046
Trap #2	$40004C
Trap #3	$400052
Trap #4	$400058
Trap #5	$40005E
Trap #6	$400064
Trap #7	$40006A
Trap #8	$400070
Trap #9	$400076
Trap #10	$40007C
User vector #64	$400082
User vector #65	$400088
User vector #66	$40008E
User vector #67	$400094
User vector #68	$40009A
User vector #69	$4000A0
User vector #70	$4000A6
User vector #71	$4000AC



Quelques approfondissements sur le processeur 68000

- **The sequence of operations during a vectored interrupt request is the following**
  - A peripheral requires attention by asserting its interrupt request output (IRQ*)
  - The priority encoder produces a 3-bit code with the highest IRQ* line active and passes it to the 68000 on the IPL0*-IPL3* inputs
  - The 68000 compares the level of the interrupt with the interrupt mask flag ($I_2I_1I_0$) in the SR.
  - If the requested input is greater than ($I_2I_1I_0$), the interrupt is serviced, otherwise it is ignored
  - If the 698000 decides to service the interrupt:
    - The code 111 is placed on ($FC_2FC_1FC_0$) to inform the system that an interrupt is about to be serviced
    - The priority of the interrupt is placed on ($A_3A_2A_1$)
    - ($FC_2FC_1FC_0$) and ($A_3A_2A_1$) are passed to an interrupt acknowledge decoder which asserts one of the seven IACK* lines
  - The asserted IACK* line informs the interrupting device that it is about to be serviced
  - The peripheral whose interrupt level matches the asserted IACK* will "know" that it is going to be serviced
  - The peripheral then writes the IVEC vector onto the data bus (D_7D_0) and asserts the DTACK* line (DTACK stands for Data Transfer Acknowledge)
  - the active DTACK* terminates the IACK cycle and the 68000 will execute the interrupt handler pointed by the vector fetched from (D_7D_0)

Quelques approfondissements sur le processeur 68000

Port Service Request Register

- **PSSR6-PSSR5 (service request)**
  - Determines whether the PI/T generates an interrupt or a DMA request
- **PSSR4-PSSR3 (operation select)**
  - Determines whether two of the dual-function pins belong to port C or perform special-purpose functions
- **PSSR2-PSSR0 (interrupt-priority control)**

Bit	PSRR7	PSRR6	PSRR5	PSRR4	PSRR3	PSRR2	PSRR1	PSRR0
Function	×	SRVRQ (DMA control)		Interrupt control		Port interrupt priority		

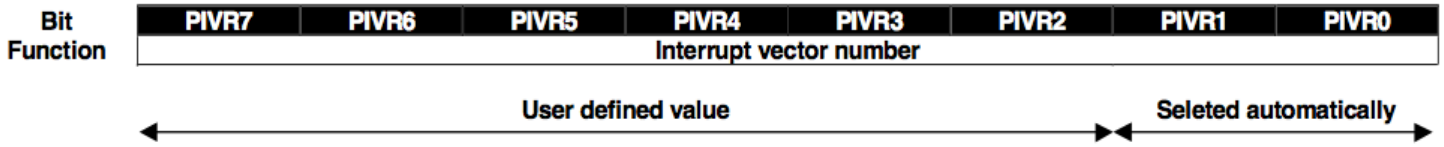
PSRR6	PSRR5	Interrupt pin function	
0	×	PC4/DMAREQ* = PC4	DMA not used
1	0	PC4/DMAREQ* = DMAREQ*	Associated with double-buffered transfers controlled by H1 H1 does not cause interrups in this mode
1	1	PC4/DMAREQ* = DMAREQ*	Associated with double-buffered transfers controlled by H3 H3 does not cause interrups in this mode

PSRR4	PSRR3	Interrupt pin function	
0	0	PC5/PIRQ* = PC5	No interrupt support
		PC6/PIACK* = PC6	No interrupt support
0	1	PC5/PIRQ* = PIRQ*	Autovectored interrupt supported
		PC6/PIACK* = PC6	Autovectored interrupt supported
1	0	PC5/PIRQ* = PC5	
		PC6/PIACK* = PIACK*	
1	1	PC5/PIRQ* = PIRQ*	Vectored interrupt supported
		PC6/PIACK* = PIACK*	Vectored interrupt supported

PSRR2	PSRR1	PSRR0	Order of priority interrupt Higuest ← → Lowest			
0	0	0	H1S	H2S	H3S	H4S
0	0	1	H2S	H1S	H3S	H4S
0	1	0	H1S	H2S	H4S	H3S
0	1	1	H2S	H1S	H4S	H3S
1	0	0	H3S	H4S	H1S	H2S
1	0	1	H3S	H4S	H2S	H1S
1	1	0	H4S	H3S	H1S	H2S
1	1	1	H4S	H3S	H2S	H1S

Port Interrupt Vector Register

- **When the parallel port section executes a vectored interrupt, it supplies the 68000 with one of four possible interrupt vector numbers**
  - Each of these numbers is associated with a specific interrupt cause
  - The upper-order six bits must be supplied by the programmer
  - The two least-significant bits are determined by the interrupt source
- **This arrangement has been implemented to avoid the need for four separate vector number registers**



PIVR1	PIVR0	Interrupt source
0	0	H1
0	1	H2
1	0	H3
1	1	H4

Quelques approfondissements sur le processeur 68000

```
*-----------------------------------------------------------------------------------------------
* Title          : Example of 68k harware interrupt and monitor system calls
* Written by     : Dr. Hamdi Ridha
* Date           : January 13th 2020
* Description    : This is a simple example, which explains some of the basics of 68k harware interrupt
*                  and monitor system calls
*-----------------------------------------------------------------------------------------------
LF    equ     $0A                 ; ASCII definitions for Linefeed and
CR    equ     $0D                 ; Carriage Return
      org     $400400
      MOVE.l  #$800001,A0
      MOVE.b  #$10,(A0)           ; Enable mode 0, Enable H1/2 , Sens of H2 is High to Low
      MOVE.b  #$18,2(A0)          ; Enable vectored interrupt via PIRQ/PIACK
      MOVE.b  #64,$A(A0)          ; Set PIVR with vector number
      MOVE.b  #4,$C(A0)           ; Enable sub-mode 00, Enable H2 as edge sensitive input
      MOVE    #$4EF9,$400088      ; JUMP + absolute long opcode
      MOVE.l  #$400450,$40008A
      ANDI    #$FBFF,SR           ; Set interrupt mask to L3 : allows interrupts of level4 and above
      BRA     *                   ; endless loop
      ORG     $400450
      LEA     text,A6
      trap    #11                 ; Monitor system call
      DC.w    5                   ; Output string pointed by A6 and terminated by $00
      trap    #11
      DC.w    0                   ; Exit user program and re-enter monitor
text  DC.b    'Hello this is an'  ; string to print, string ends with NULL
      DC.b    'example of interrupt!',LF,LF,0
```



Architecture d'un système embarqué

Motorola 68000 CPU Opcodes

Mnemonic	Size	Single Effective Address Operation Word	Data
ORI to CCR	B	0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0	B I
ORI to SR	W	0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0	W I
ORI	B W L	0 0 0 0 0 0 0 0 S M Xn	I
ANDI to CCR	B	0 0 0 0 0 0 1 0 0 0 1 1 1 1 0 0	B I
ANDI to SR	W	0 0 0 0 0 0 1 0 0 1 1 1 1 1 0 0	W I
ANDI	B W L	0 0 0 0 0 0 1 0 S M Xn	I
SUBI	B W L	0 0 0 0 0 1 0 0 S M Xn	I
ADDI	B W L	0 0 0 0 0 1 1 0 S M Xn	I
EORI to CCR	B	0 0 0 0 1 0 1 0 0 0 1 1 1 1 0 0	B I
EORI to SR	W	0 0 0 0 1 0 1 0 0 1 1 1 1 1 0 0	W I
EORI	B W L	0 0 0 0 1 0 1 0 S M Xn	I
CMPI	B W L	0 0 0 0 1 1 0 0 S M Xn	I
BTST	B L	0 0 0 0 1 0 0 0 0 0 M Xn	B N
BCHG	B L	0 0 0 0 1 0 0 0 0 1 M Xn	B N
BCLR	B L	0 0 0 0 1 0 0 0 1 0 M Xn	B N
BSET	B L	0 0 0 0 1 0 0 0 1 1 M Xn	B N
BTST	B L	0 0 0 0 Dn 1 0 0 M Xn	B N
BCHG	B L	0 0 0 0 Dn 1 0 1 M Xn	B N
BCLR	B L	0 0 0 0 Dn 1 1 0 M Xn	B N
BSET	B L	0 0 0 0 Dn 1 1 1 M Xn	B N
MOVEP	W L	0 0 0 0 Dn 1 D S 0 0 1 An	W D
MOVEA	W L	0 0 S An 0 0 1 M Xn	
MOVE	B W L	0 0 S Xn M M Xn	
MOVE from SR	W	0 1 0 0 0 0 0 0 1 1 M Xn	
MOVE to CCR	B	0 1 0 0 0 1 0 0 1 1 M Xn	
MOVE to SR	W	0 1 0 0 0 1 1 0 1 1 M Xn	
NEGX	B W L	0 1 0 0 0 0 0 0 S M Xn	
CLR	B W L	0 1 0 0 0 0 1 0 S M Xn	
NEG	B W L	0 1 0 0 0 1 0 0 S M Xn	
NOT	B W L	0 1 0 0 0 1 1 0 S M Xn	
EXT	W L	0 1 0 0 1 0 0 0 1 S 0 0 0 Dn	
NBCD	B	0 1 0 0 1 0 0 0 0 0 M Xn	
SWAP	W	0 1 0 0 1 0 0 0 0 1 0 0 0 Dn	
PEA	L	0 1 0 0 1 0 0 0 0 1 M Xn	
ILLEGAL		0 1 0 0 1 0 1 0 1 1 1 1 1 1 0 0	
TAS	B	0 1 0 0 1 0 1 0 1 1 M Xn	
TST	B W L	0 1 0 0 1 0 1 0 S M Xn	
TRAP		0 1 0 0 1 1 1 0 0 1 0 0 Vector	
LINK	W	0 1 0 0 1 1 1 0 0 1 0 1 0 An	W D
UNLK		0 1 0 0 1 1 1 0 0 1 0 1 1 An	
MOVE USP	L	0 1 0 0 1 1 1 0 0 1 1 0 D An	
RESET		0 1 0 0 1 1 1 0 0 1 1 1 0 0 0 0	
NOP		0 1 0 0 1 1 1 0 0 1 1 1 0 0 0 1	
STOP		0 1 0 0 1 1 1 0 0 1 1 1 0 0 1 0	W I

Mnemonic	Size	Single Effective Address Operation Word	Data
RTE		0 1 0 0 1 1 1 0 0 1 1 1 0 0 1 1	
RTS		0 1 0 0 1 1 1 0 0 1 1 1 0 1 0 1	
TRAPV		0 1 0 0 1 1 1 0 0 1 1 1 0 1 1 0	
RTR		0 1 0 0 1 1 1 0 0 1 1 1 0 1 1 1	
JSR		0 1 0 0 1 1 1 0 1 0 M Xn	
JMP		0 1 0 0 1 1 1 0 1 1 M Xn	
MOVEM	W L	0 1 0 0 1 D 0 0 1 S M Xn	W M
LEA	L	0 1 0 0 An 1 1 1 M Xn	
CHK	W	0 1 0 0 Dn 1 1 0 M Xn	
ADDQ	B W L	0 1 0 1 Data 0 S M Xn	
SUBQ	B W L	0 1 0 1 Data 1 S M Xn	
Scc	B	0 1 0 1 Condition 1 1 M Xn	
DBcc	W	0 1 0 1 Condition 1 1 0 0 1 Dn	W D
BRA	B W	0 1 1 0 0 0 0 0 Displacement	W D
BSR	B W	0 1 1 0 0 0 0 1 Displacement	W D
Bcc	B W	0 1 1 0 Condition Displacement	W D
MOVEQ	L	0 1 1 1 Dn 0 Data	
DIVU	W	1 0 0 0 Dn 0 1 1 M Xn	
DIVS	W	1 0 0 0 Dn 1 1 1 M Xn	
SBCD	B	1 0 0 0 Xn 1 0 0 0 0 M Xn	
OR	B W L	1 0 0 0 Dn D S M Xn	
SUB	B W L	1 0 0 1 Dn D S M Xn	
SUBX	B W L	1 0 0 1 Xn 1 S 0 0 M Xn	
SUBA	W L	1 0 0 1 An S 1 1 M Xn	
EOR	B W L	1 0 1 1 Dn 1 S M Xn	
CMPM	B W L	1 0 1 1 An 1 S 0 0 1 An	
CMP	B W L	1 0 1 1 Dn 0 S M Xn	
CMPA	W L	1 0 1 1 An S 1 1 M Xn	
MULU	W	1 1 0 0 Dn 0 1 1 M Xn	
MULS	W	1 1 0 0 Dn 1 1 1 M Xn	
ABCD	B	1 1 0 0 Xn 1 0 0 0 0 M Xn	
EXG	L	1 1 0 0 Xn 1 M 0 0 M Xn	
AND	B W L	1 1 0 0 Dn D S M Xn	
ADD	B W L	1 1 0 1 Dn D S M Xn	
ADDX	B W L	1 1 0 1 Xn 1 S 0 0 M Xn	
ADDA	W L	1 1 0 1 An S 1 1 M Xn	
ASd	B W L	1 1 1 0 0 0 0 D 1 1 M Xn	
LSd	B W L	1 1 1 0 0 0 1 D 1 1 M Xn	
ROXd	B W L	1 1 1 0 0 1 0 D 1 1 M Xn	
ROd	B W L	1 1 1 0 0 1 1 D 1 1 M Xn	
ASd	B W L	1 1 1 0 Rotation D S M 0 0 Dn	
LSd	B W L	1 1 1 0 Rotation D S M 0 1 Dn	
ROXd	B W L	1 1 1 0 Rotation D S M 1 0 Dn	
ROd	B W L	1 1 1 0 Rotation D S M 1 1 Dn	

Mode	Register List Mask															
Postincrement	A7	A6	A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
Predecrement	D0	D1	D2	D3	D4	D5	D6	D7	A0	A1	A2	A3	A4	A5	A6	A7

Brief Extension Word					
M	Xn	S	0	0	0	Displacement
---	---	---	---	---	---	---

Architecture d'un système embarqué

Motorola 68000 CPU Opcodes

Addressing Mode	Format	M	Xn
Data register	Dn	0 0 0	reg
Address register	An	0 0 1	reg
Address	(An)	0 1 0	reg
Address with Postincrement	(An)+	0 1 1	reg
Address with Predecrement	-(An)	1 0 0	reg
Address with Displacement	(d_{16}, An)	1 0 1	reg
Address with Index	(d_8, An, Xn)	1 1 0	reg
Program Counter with Displacement	(d_{16}, PC)	1 1 1	0 1 0
Program Counter with Index	(d_8, PC, Xn)	1 1 1	0 1 1
Absolute Short	(xxx).W	1 1 1	0 0 0
Absolute Long	(xxx).L	1 1 1	0 0 1
Immediate	#imm	1 1 1	1 0 0

Data Type	Letter
Immediate	I
Bit Index	N
Displacement	D
Optional Displacement	D
Register List Mask	M

Data Size	Letter
Byte	B
Word	W
Long	L
Any	

Direction	D	D
Register to memory	0	1
Memory to register	1	0

Direction	D
Dn ♦ <ea> → Dn	0
<ea> ♦ Dn → <ea>	1

Operation Size	Suffix	S	S	S
Byte	.b	0 0	/	0 1
Word	.w	0 1	0	1 1
Long	.l	1 0	1	1 0

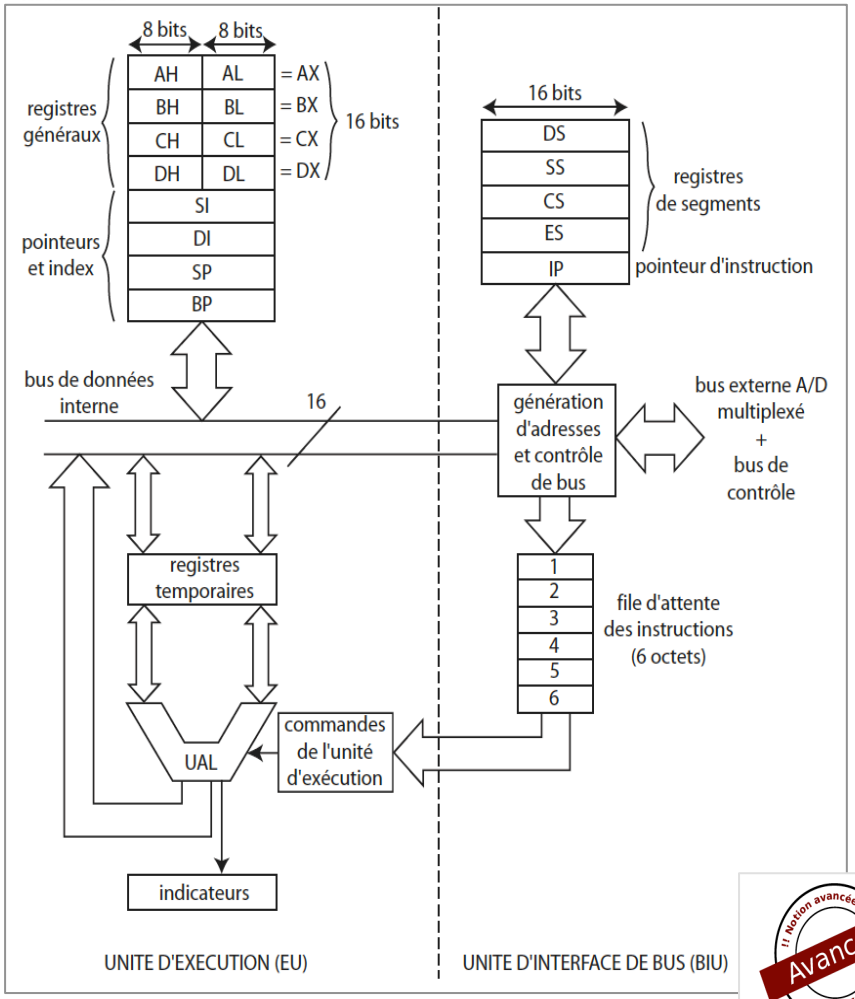
Direction	d	D
Right	R	0
Left	L	1

Rotation	M
Immediate	0
Register	1

Mode	M
Dn	0
-(An)	1

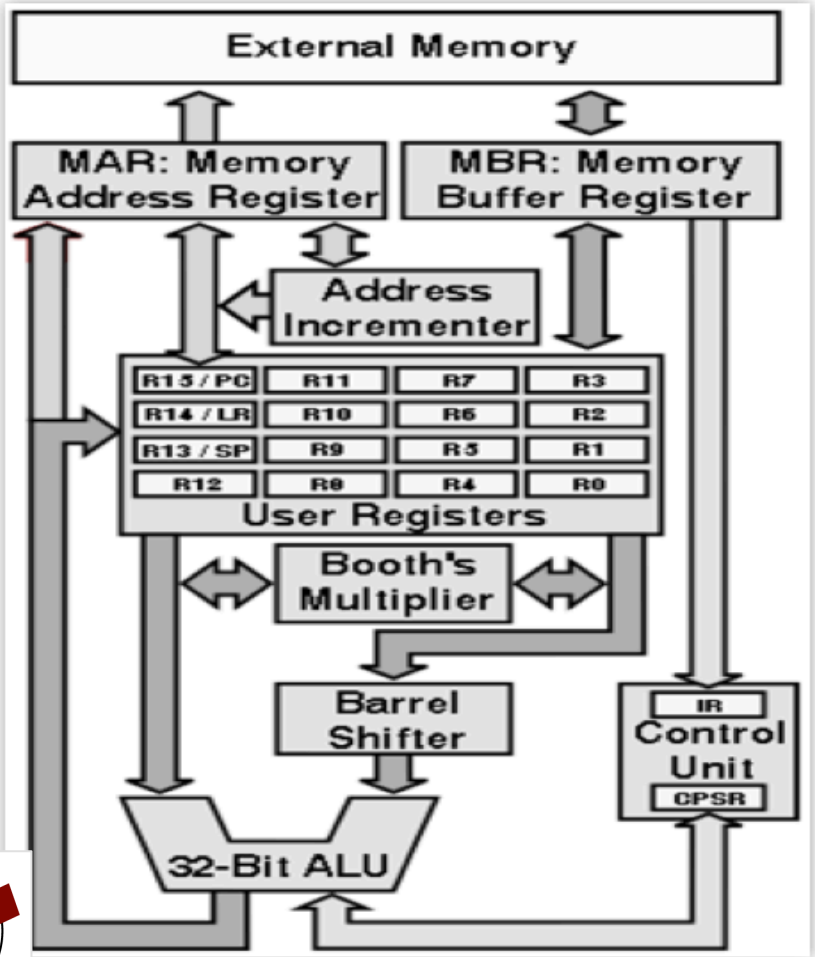
Condition	Mnemonic	Cond
True	T	0 0 0 0
False	F	0 0 0 1
Higher	HI	0 0 1 0
Lower or Same	LS	0 0 1 1
Carry Clear	CC	0 1 0 0
Carry Set	CS	0 1 0 1
Not Equal	NE	0 1 1 0
Equal	EQ	0 1 1 1
Overflow Clear	VC	1 0 0 0
Overflow Set	VS	1 0 0 1
Plus	PL	1 0 1 0
Minus	MI	1 0 1 1
Greater or Equal	GE	1 1 0 0
Less Than	LT	1 1 0 1
Greater Than	GT	1 1 1 0
Less or Equal	LE	1 1 1 1

Comparaison avec d'autres architecture



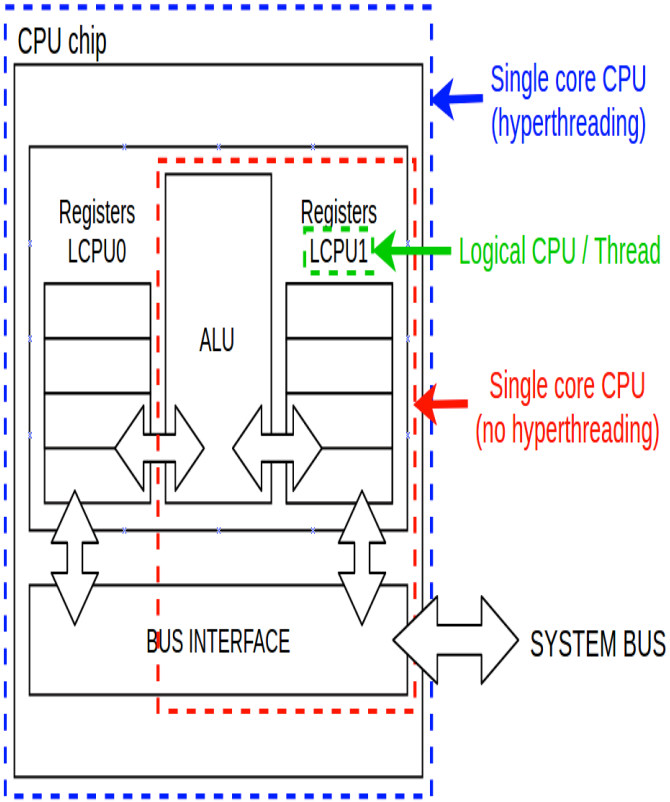
Architecture 8086 (Intel)





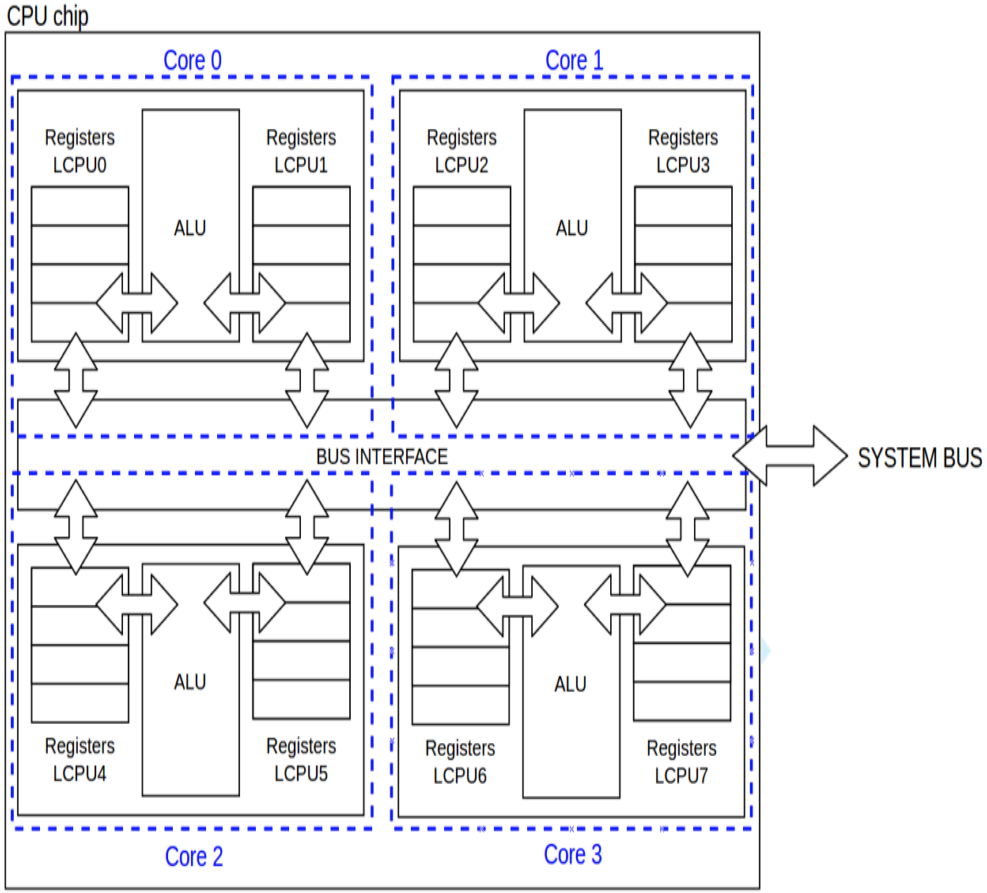
Architecture ARM

Comparaison avec d'autres architecture



Single core hyperthreading CPU
(2 logical CPU's / threads)

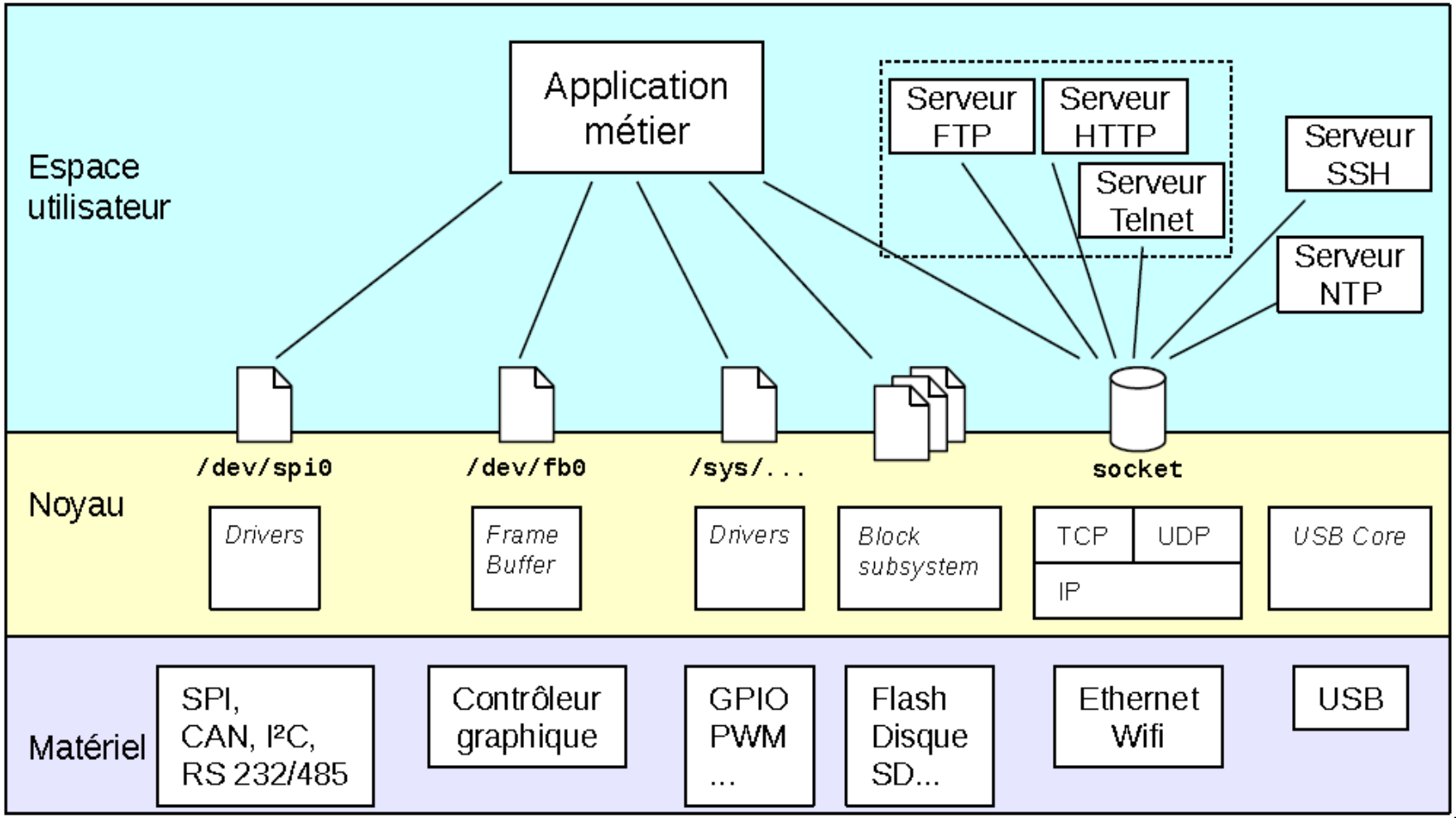




Quad-core hyperthreading CPU

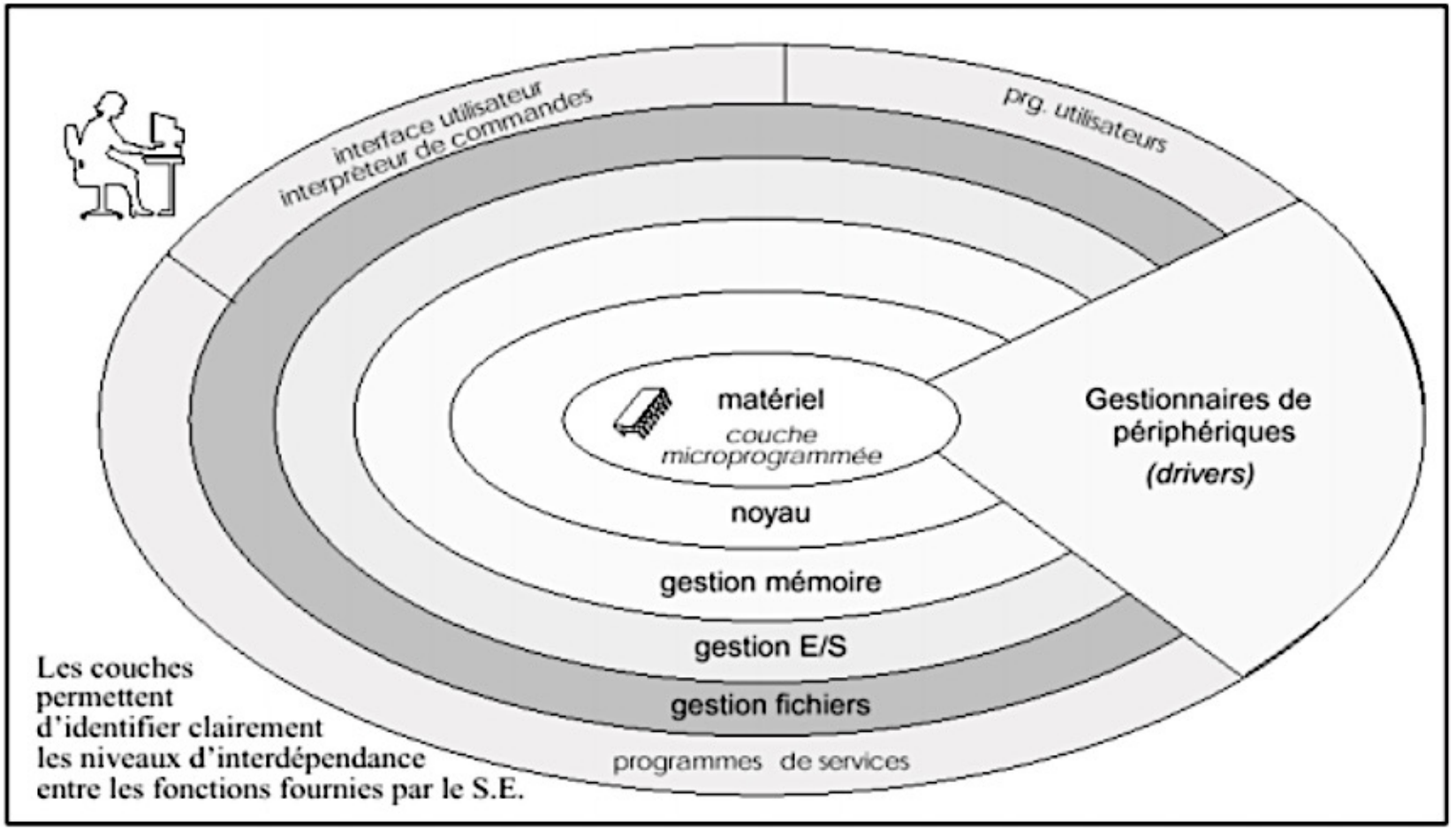
Système d'exploitation

Masquer la complexité matérielle / simplifier les accès au matériel



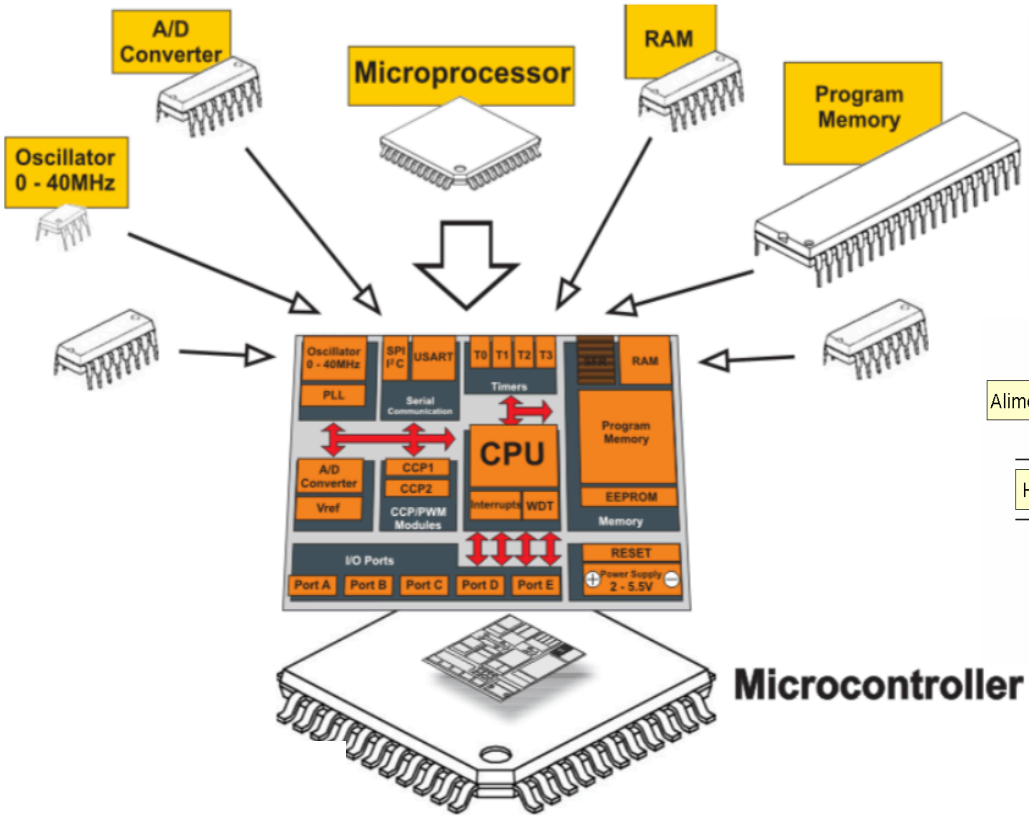
Système d'exploitation

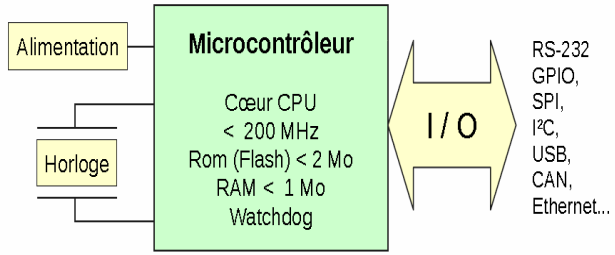
Masquer la complexité matérielle / simplifier les accès au matériel

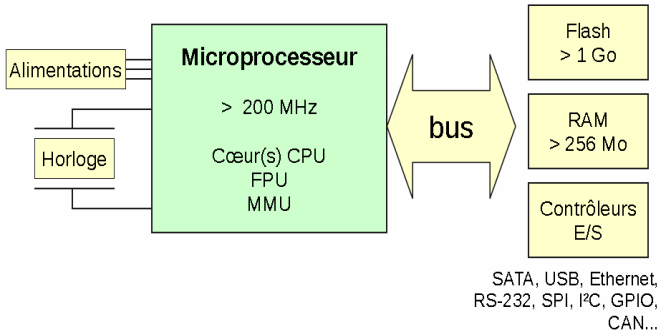


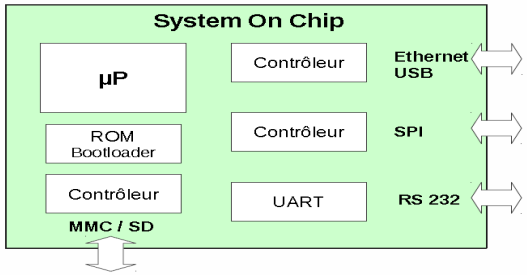
Système d'exploitation

Microprocesseur vs Microcontrôleur

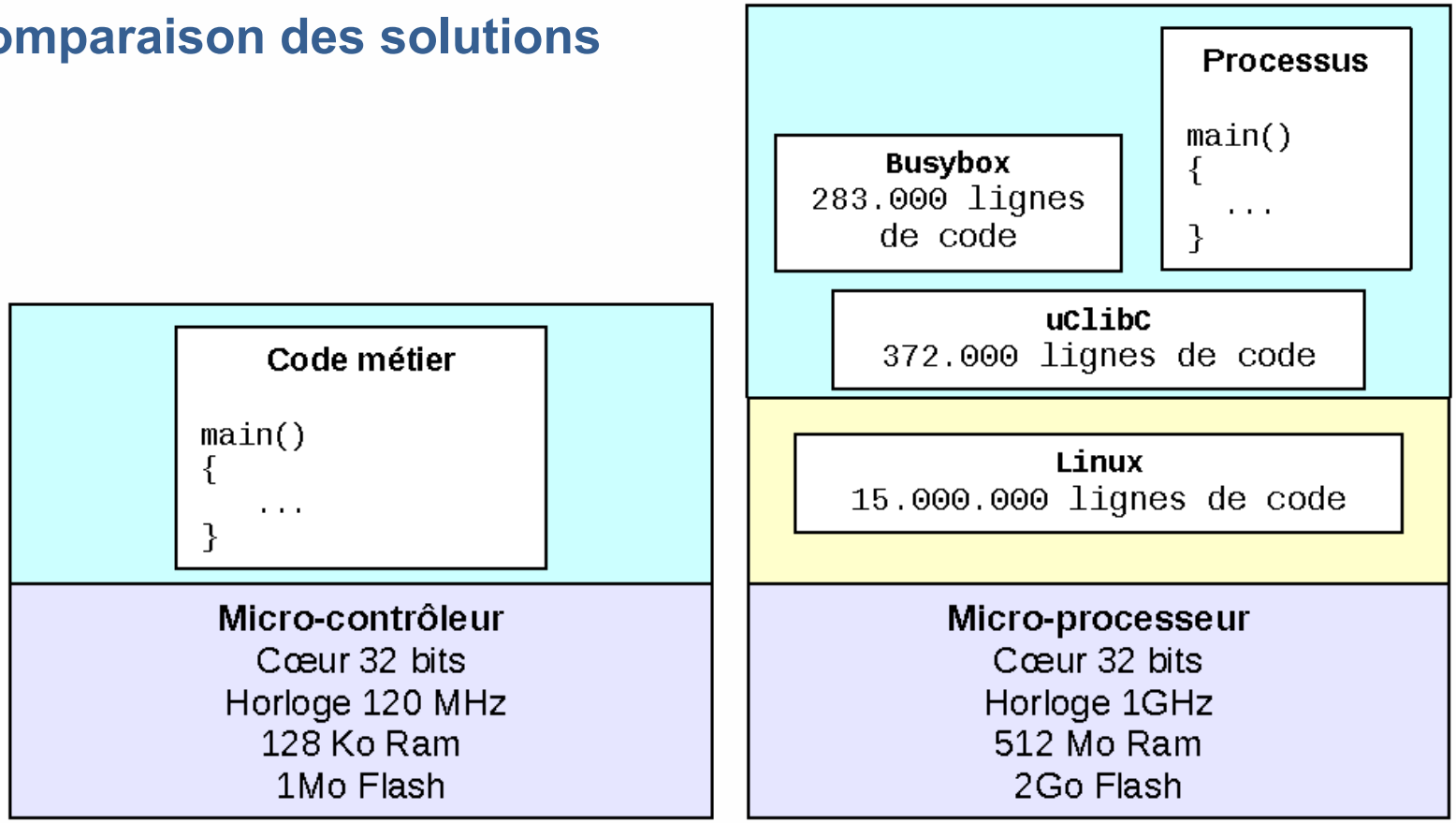








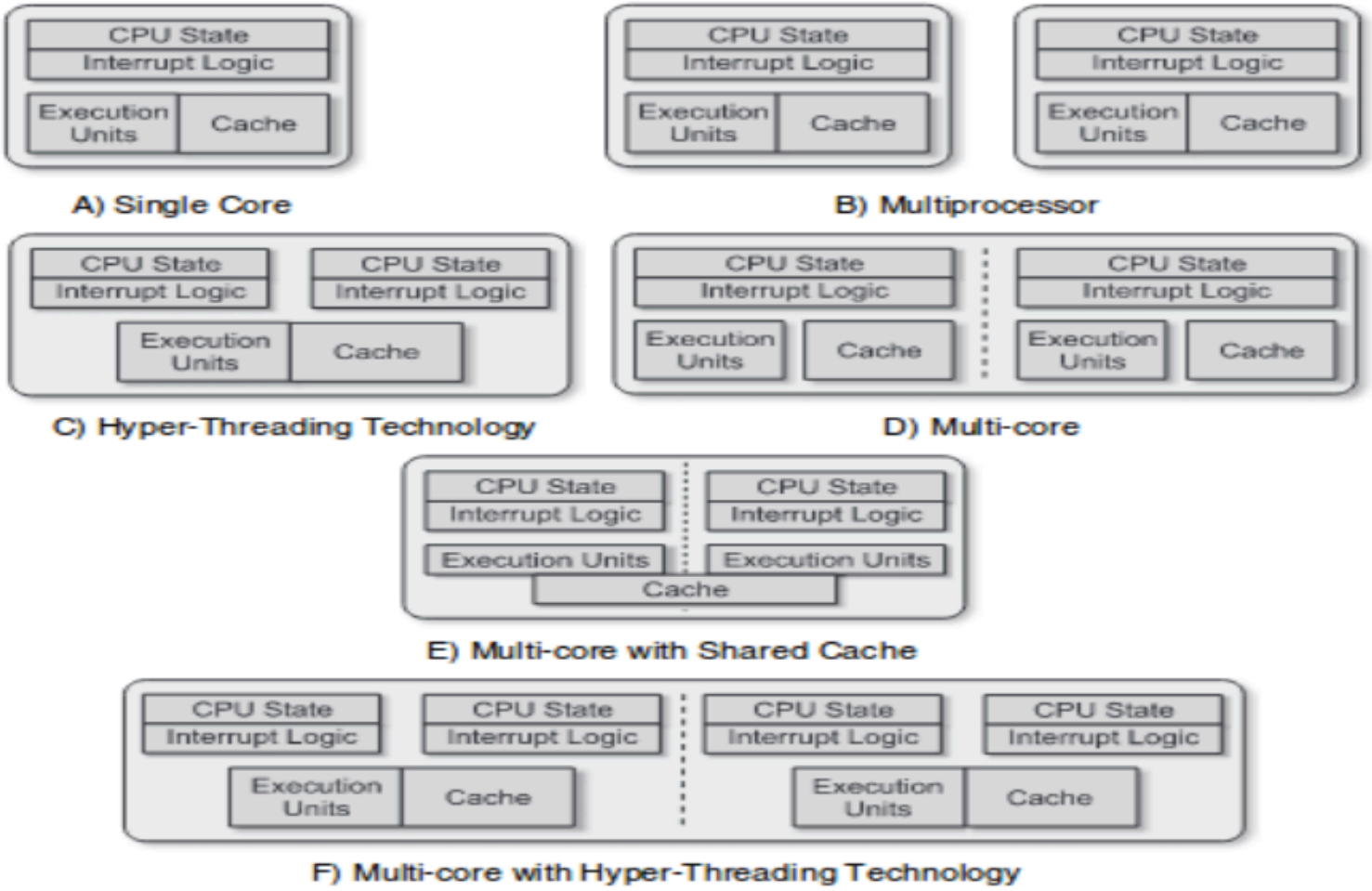
Comparaison des solutions



Système à micro-contrôleur (STM32F2)

Système à micro-processeur (T.I. AM3359 de BeagleBone Black)

Système d'exploitation



Sans système d'exploitation

Un seul et unique programme dans l'ordinateur se charge de tout faire, y compris manipuler le matériel. Et bien évidemment, un programmeur qui souhaite se passer de système d'exploitation doit tout programmer, y compris les parties du programme en charge de la gestion des périphériques, des ports d'entrée-sortie, de la mémoire...

De plus, ce que le programmeur aura programmé sera difficilement portable sur d'autres ordinateurs, qui n'auront pas le même matériel : on ne gère pas les périphériques et/ou la mémoire de la même façon sur tous les ordinateurs. Autant vous dire que pour rendre notre programme compatible avec un autre ordinateur, il faudrait reprogrammer une grande partie du programme ainsi conçu.

Avec un système d'exploitation

Plutôt que de créer un seul et unique programme informatique chargé de tout gérer, on segmente ce programme en plusieurs programmes séparés. Ces programmes peuvent être divisés en deux types :

Programmes systèmes	Programmes applicatifs
Ils sont en charge de la gestion du matériel de l'ordinateur. Ils gèrent : • la gestion de la mémoire ; • les périphériques et ports d'entrée-sortie ; • l'exécution des autres programmes (ceux qui ne sont pas des programmes systèmes) exécutés sur l'ordinateur.	Ces programmes applicatifs sont des programmes comme les autres qui font exécuter directement leurs instructions par le processeur, mais qui : • Ne peuvent pas gérer les périphériques et ports d'entrée-sortie. • Ne peuvent manipuler qu'une portion de la mémoire que le(s) programme(s) système(s) leur a(ont) attribué. Lorsqu'ils ont besoin de le faire, ceux-ci délèguent les taches à effectuer aux programmes systèmes.

Généralement, les programmes systèmes (et quelques programmes applicatifs de base) sont tous regroupés dans ce qu'on appelle un **système d'exploitation**, aussi appelé OS (*operating system* en anglais). En plus de ces programmes systèmes, le reste de l'OS est composé de programmes applicatifs qui permettent par exemple d'afficher une interface qui permet à l'utilisateur de pouvoir utiliser l'ordinateur comme il le souhaite.

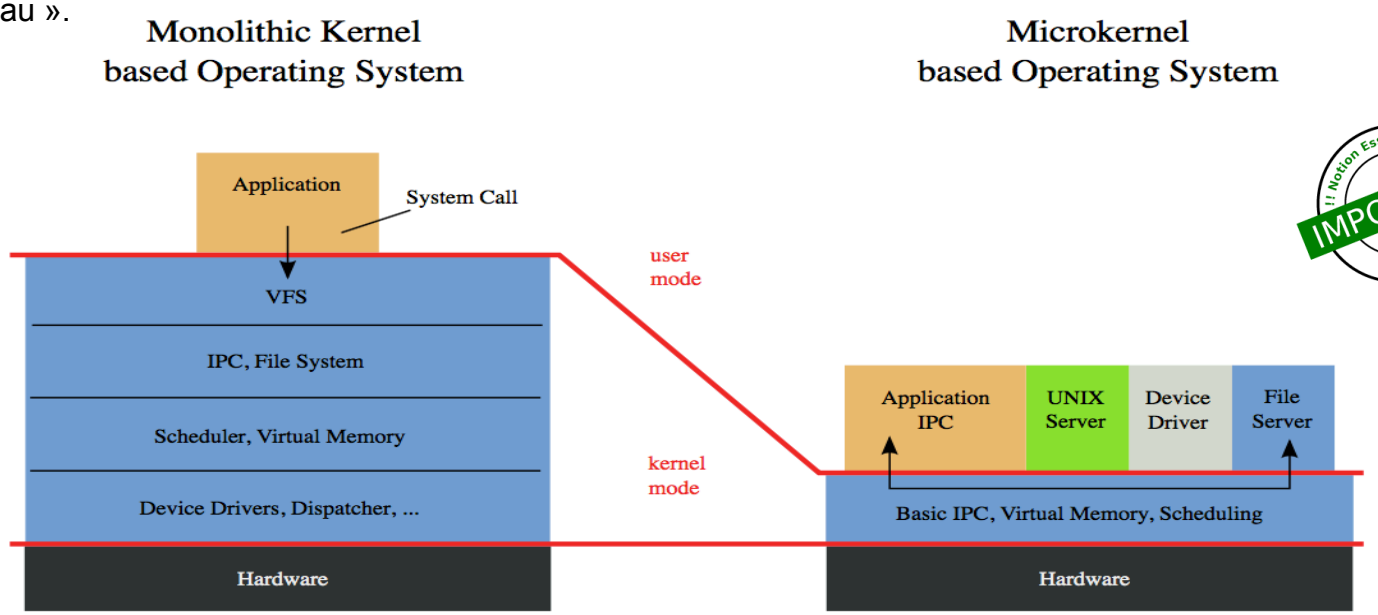
Système d'exploitation

Noyau monolithique

Le système d'exploitation est composé d'un seul programme : le noyau. Celui-ci est typiquement organisé en couches. La construction monolithique est la plus courante, appliquée à la plupart des Unix

Systèmes à micro-noyaux

Les systèmes à micro-noyaux cherchent à minimiser les fonctionnalités dépendantes du noyau en plaçant la plus grande partie des services du système d'exploitation à l'extérieur de ce noyau, c'est-à-dire dans l'espace utilisateur. Ces fonctionnalités sont alors fournies par de petits serveurs indépendants possédant leur propre espace d'adressage Un petit nombre de fonctions fondamentales est conservé dans un noyau minimaliste appelé « micronoyau ».



Comparaison entre noyau monolithique ou micronoyau

Système d'exploitation

Noyaux monolithiques modulaires

Dans ce type de noyau, seules les parties fondamentales du système sont regroupées dans un bloc de code unique (monolithique). Les autres fonctions, comme les pilotes matériels, sont regroupées en différents modules qui peuvent être séparés tant du point de vue du code que du point de vue binaire.
La modularité du noyau permet le chargement à la demande de fonctionnalités et augmente les possibilités de configuration.

Noyaux hybrides

La construction hybride ressemble à une construction microkernel, cependant certaines fonctions ont été placées dans le noyau pour des raisons d'efficacité

Noyaux temps réel

Les noyaux temps réel peuvent adopter en théorie n'importe quelle architecture précédemment listée. Ils fournissent souvent deux interfaces séparées, l'une spécialisée dans le temps réel et l'autre générique. Les applications temps réel font alors appel à la partie temps réel du noyau.

Machine Virtuelle (VM)

Une machine virtuelle, ou VM (Virtual Machine), est un environnement d'application ou de système d'exploitation installé sur un logiciel qui imite un matériel dédié. Côté utilisateur final, l'interaction avec une machine virtuelle est la même qu'avec un matériel dédié.



Système d'exploitation



- Un système d'exploitation est dit **multi-tâches** quand il permet l'exécution simultanée de plusieurs programmes. Tous les système d'exploitation actuels sont multi-tâches.
- Il est dit **multi-utilisateurs** quand il est conçu pour être utilisé simultanément par plusieurs usagers, souvent à travers un réseau informatique (notion de serveurs). Ils sont multi-tâches et en général sécurisés, c'est-à-dire qu'il vont refuser d'exécuter toute opération pour laquelle l'usager n'a pas préalablement reçu une permission.
- Il est dit **multi-processeurs** quand il est conçu pour exploiter un ordinateur équipé de plusieurs processeurs. Dans de tels systèmes d'exploitation, plusieurs programmes sont exécutés simultanément par les différents processeurs.
- Il est dit **temps réel** quand il garantit que les opérations seront effectuées en respectant des délais stricts, et ce quelles que soient les conditions d'utilisation (charge du système). De tels systèmes d'exploitation sont utilisés dans l'industrie, l'aéronautique ou l'électronique pour créer des systèmes temps réel (souvent embarqué)

Système d'exploitation

- Un programme qui s'exécute est appelé un **processus**.
- Un processus comporte du code machine exécutable, une zone mémoire (données allouées par le processus), une pile ou *stack* (pour les variables locales des fonctions et la gestion des appels et retour des fonctions) et un tas ou *heap* pour les allocations dynamiques (fonctions `malloc` ou `new`).
- Un processus est créé avec 3 flux standards : un flux d'entrée (***stdin***) relié par défaut au clavier, un flux de sortie (***stdout***) relié par défaut à l'écran et un flux d'erreur (***stderr***).
- Un **flux ou flot** est un **canal recevant ou fournissant de l'information**.
- Ce processus est une entité qui, de sa création à sa mort, est identifié par l'OS par une valeur numérique : le **PID** (*Process IDentifier*).
- Les commandes `ps` et `top` listent les processus sous UNIX et, sous Windows on utilisera le gestionnaire de tâches (`taskmgr.exe`).

Chaine de compilation et d'exécution

```c
int main()
{
    int a=3;
    a++;
    int b=a+2;

    char texte[]="j aime l informatique";

    return 0;
}
```

Code (C)

(=langage humain)

Compilateur

```
b8af 0860 0055 482d a808 6000 4883 f80
4889 e577 025d c3b8 0000 0000 4885 c07
f45d bfa8 0860 00ff e00f 1f80 0000 0000
b8a8 0860 0055 482d a808 6000 48c1 f803
4889 e548 89c2 48c1 ea3f 4801 d048 89c6
48d1 fe75 025d c3ba 0000 0000 4885 d274
f45d bfa8 0860 00ff e20f 1f80 0000 0000
803d 4104 2000 0075 1155 4889 e5e8 7eff
ffff 5dc6 052e 0420 0001 f3c3 0f1f 4000
4883 3d08 0220 0000 741b b800 0000 0048
85c0 7411 55bf 9006 6000 4889 e5ff d05d
e97b ffff ffe9 76ff ffff 9090 5548 89e5
c745 fc03 0000 0083 45fc 018b 45fc 83c0
0289 45f8 c745 e06a 2061 69c7 45e4 6d65
206c c745 e820 696e 66c7 45ec 6f72 6d61
c745 f074 6971 7566 c745 f465 00b8 0000
0000 5dc3 9090 9090 9090 9090 9090 9090
4889 6c24 d84c 8964 24e0 488d 2d77 0120
004c 8d25 6801 2000 4889 5c24 d04c 896c
24e8 4c89 7424 f04c 897c 24f8 4883 ec38
4c29 e541 89ff 4989 f648 c1fd 0349 89d5
31db e829 feff ff48 85ed 741a 0f1f 4000
4c89 ea4c 89f6 4489 ff41 ff14 dc48 83c3
0148 39eb 75ea 488b 5c24 0848 8b6c 2410
4c8b 6424 184c 8b6c 2420 4c8b 7424 284c
```

gcc (GCC) http://gcc.gnu.org/

(C, C++,Obj-C,Fortran,Ada,Go)

GNU Compiler Collection

Le + répendu sous Linux

D'autres existent:

Nwcc
Clang
Quick C (Microsoft)
Turbo C (Embarcadero)
XL C (IBM)

Fichier executable

(=binaire)

(Windows: .exe
Linux: nom quelconque)

Note:
Compilateur C++ compatibles

BorlandC++
Intel C++ Compiler
Visual Studio
Turbo C++
ProDev
Solaris Studio

Chaine de compilation et d'exécution

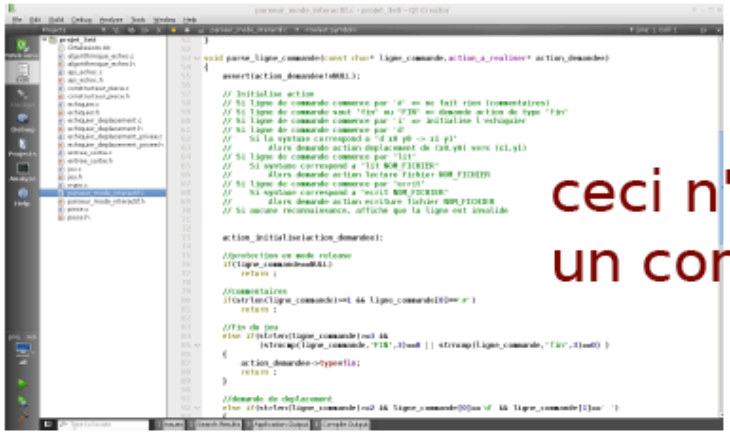
Ne pas confondre **IDE** et **Compilateur**



éditer du texte/code

- Emacs
- QtCreator
- MS Visual C++
- Dev C++
- Eclipse
- Code::Blocks
- Anjuta
- KDevelop

génère le binaire



ceci n'est pas un compilateur!

ENIG

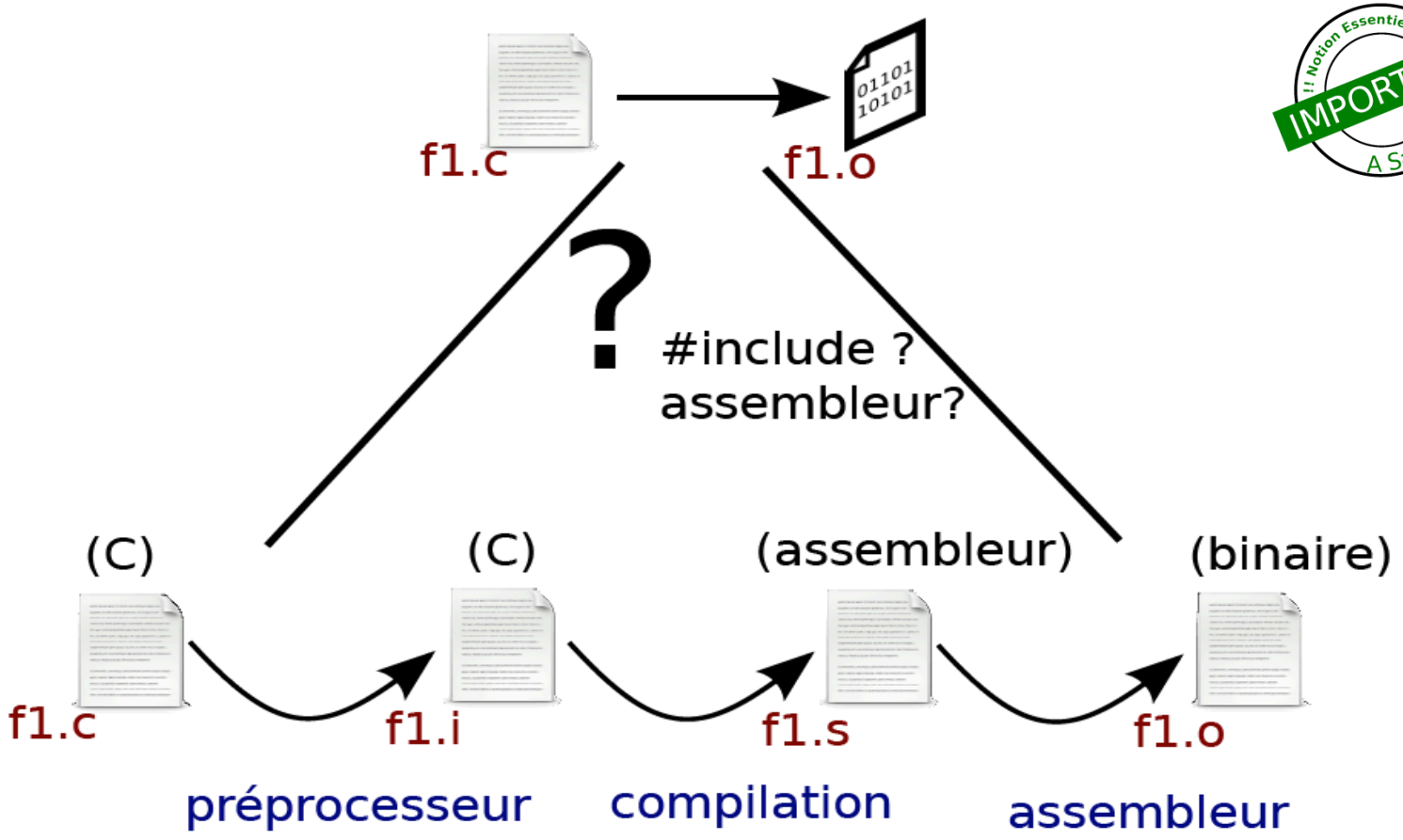
ENIG

Chaine de compilation et d'exécution



Langage	Type	Compilé/Interprété	Domaine d'application
Basic	Non structuré	Interprété	Apprentissage
Cobol	Structuré	Compilé	Gestion
Fortran	Structuré	Compilé	Calculs
PL1	Structuré	Compilé	Industrie
C	Structuré	Compilé	Programmation système
Pascal	Structuré (il existe une version objet)	Compilé	Enseignement
ADA	Structuré (il existe une version objet)	Compilé	Militaire, aérospatiale
LISP	Structuré	Interprété et compilé	Intelligence artificielle
Visual Basic	Orienté objet	Interprété ou compilé	Interfaces homme-machine
C++	Orienté objet	Compilé	Industrie
Java	Orienté objet	Interprété et compilé	Industrie – Internet
Perl	Orienté objet	Interprété	Industrie– administration réseau
PHP	Orienté objet	Interprété	Web

Compilation d'un programme écrit en C





Compilation d'un programme écrit en C

Pour comprendre ce qui se passe à chaque étape, prenons comme exemple un fichier `bonjour.c` contenant le code suivant :

```
#include <stdio.h>

int main(void)
{
    puts("Bonjour !");
    return 0;
}
```

Première étape : le pré-processeur

Le pré-processeur (qui s'appelle `cpp` sous Unix) est un module de pré-traitement qui effectue des transformations sur le code écrit avant qu'il ne soit traité par le compilateur.

Il faut comprendre que le pré-processeur effectue des remplacements textuels (et non sémantiques) : le pré-processeur n'est pas le compilateur, il n'interprète pas ce qu'il manipule. On peut faire le rapprochement avec la fonction Remplacer d'un traitement de texte.

Dans le cas d'une directive `#define NOM val`, le pré-processeur remplace donc tout simplement chaque occurence de la chaine `NOM` par la valeur `val` dans tout le texte du fichier. Dans le cas d'une directive `#include <fichier.h>`, le pré-processeur fait simplement un copier-coller du contenu du fichier `fichier.h` dans le fichier contenant la macro. Ce remplacement est récursif : si le fichier inclut contient lui-même des `#include`, les fichiers qu'il inclut seront aussi inclut dans le fichier initial.

Compilation d'un programme écrit en C

Dans le cas de notre exemple, si on tape la commande `cpp bonjour.c bonjour.i`, le pré-processeur génère un fichier `bonjour.i` qui contient, en plus du code initial que l'on a écrit, tout le contenu du fichier `stdio.h` (plus tout le contenu de tous les fichiers d'en-tête inclus dans `stdio.h`, récursivement).

Deuxième étape : le compilateur

Le compilateur (le programme `cc` sous Unix) a pour rôle de traduire le code C en code assembleur.
Le compilateur C a pour particularité de faire de la compilation séparée : cela signifie que chaque fichier source est compilé sans regarder les autres fichiers, même s'ils font partie du même programme.
Pour compiler un fichier C, le compilateur a juste besoin de connaitre les prototypes de toutes les fonctions utilisées dans ce fichier, mais il n'a pas besoin de d'avoir à sa disposition le code de ces fonctions externes.
Les prototypes des fonctions sont définies dans des fichiers .h, qui sont inclus par le pré-processeur dans le fichier C qui les utilise via la directive `#include`.
Dans le cas de notre exemple, si on appelle la commande `cc -S bonjour.i` sur le fichier généré par le pré-processeur, on obtient un fichier `bonjour.s` qui contient le même programme mais écrit en langage assembleur

Troisième étape : l'assembleur

L'assembleur (appelé `as` sous Unix) prend un fichier écrit en langage assembleur et le traduit en code binaire (langage machine).
Dans le cas de notre exemple, si vous tapez la commande `as -o bonjour.o bonjour.s`, l'assembleur produit un fichier objet `bonjour.o` qui contient le même programme que `bonjour.s`, mais en langage machine (inutile de chercher à lire ce fichier, c'est du code machine, totalement illisible pour un humain !).

Compilation d'un programme écrit en C

Quatrième étape : l'éditeur de liens

L'éditeur de lien (`ld` sous Unix) a pour rôle de fusionner les différents fichiers objets pour produire l'exécutable final. En pratique, cela veut dire qu'il va fusionner :

- les codes contenus dans les différentes fonctions de tous les fichiers objets du programme, pour obtenir une seule grosse zone de code ;
- les données statiques allouées dans tous les fichiers objets du programme, pour obtenir une grosse zone de données statiques commune à tout le programme.

Même dans le cas de notre exemple simpliste, un programme écrit en C utilise toujours d'autres fichiers objets qui font partie de la bibliothèque C standard (appelée en général `libc.a`, un fichier .a étant une archive contenant plusieurs fichiers .o) : dans notre exemple, on utilise le fichier objet contenant le code de la fonction `puts`, ainsi que d'autres fichiers objets permettant de rendre le fichier objet exécutable et qu'on ne détaillera pas (ce type de fichiers s'appellent souvent `crt0.o` sous Unix).

Comme le compilateur C fait de la compilation séparée, l'assembleur a laissé un trou dans le code objet produit, à l'endroit de l'appel à `puts` : ce trou est une indication laissée à l'éditeur de liens pour lui dire : « ici, tu dois insérer un véritable appel à la fonction `puts` que tu trouveras dans un autre fichier objet ».



Compilation d'un programme écrit en C

Mais beaucoup plus simplement ...

En pratique, il serait assez pénible d'avoir à effectuer à la main toutes ces étapes pour obtenir son exécutable...
La chaine de compilation GCC est en fait beaucoup plus simple à utiliser : elle fourni un programme générique (appelé simplement `gcc`) qui est capable d'appeler chacun des programmes nécessaires dans le bon ordre jusqu'à produire l'exécutable.

Le programme `gcc` se base simplement sur l'extension du fichier qu'on lui donne pour déterminer quels programmes il doit appeler. Par exemple, si le fichier se termine par :

- .c : il appelle dans l'ordre : le pré-processeur, le compilateur, l'assembleur et l'éditeur de lien ;
- .i : il appelle dans l'ordre : le compilateur, l'assembleur et l'éditeur de liens ;
- .s : il appelle dans l'ordre : l'assembleur et l'éditeur de liens ;
- .o : il appelle simplement l'éditeur de liens

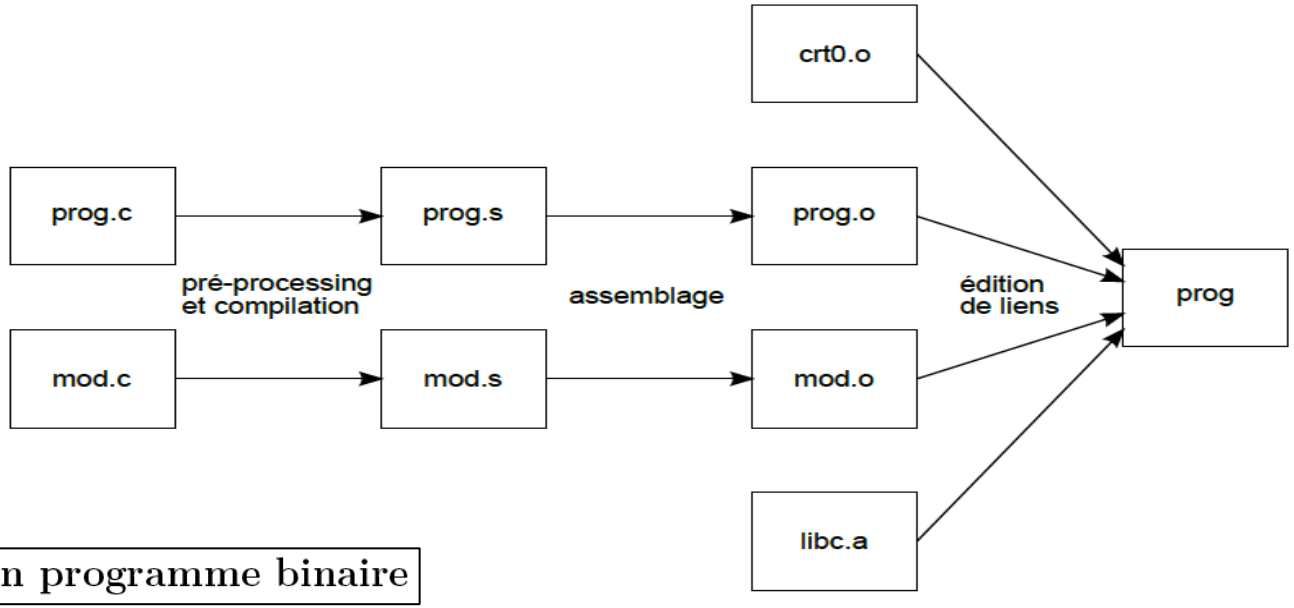
On peut demander à `gcc` de produire les fichiers intermédiaires manipulés par les différents programmes qu'il appelle si on veut les observer :

- `gcc -E -o bonjour.i bonjour.c` produit le fichier en sortie du pré-processeur ;
- `gcc -S bonjour.c` produit le fichier en sortie du compilateur (après avoir appelé aussi le pré-processeur) ;

Compilation d'un programme écrit en C

— `gcc -c bonjour.o` produit le fichier en sortie de l'assembleur (après avoir appelé aussi le pré-processeur et le compilateur) ;
— `gcc -o bonjour bonjour.c` appelle tous les programmes nécessaires pour produit directement le binaire `bonjour`.

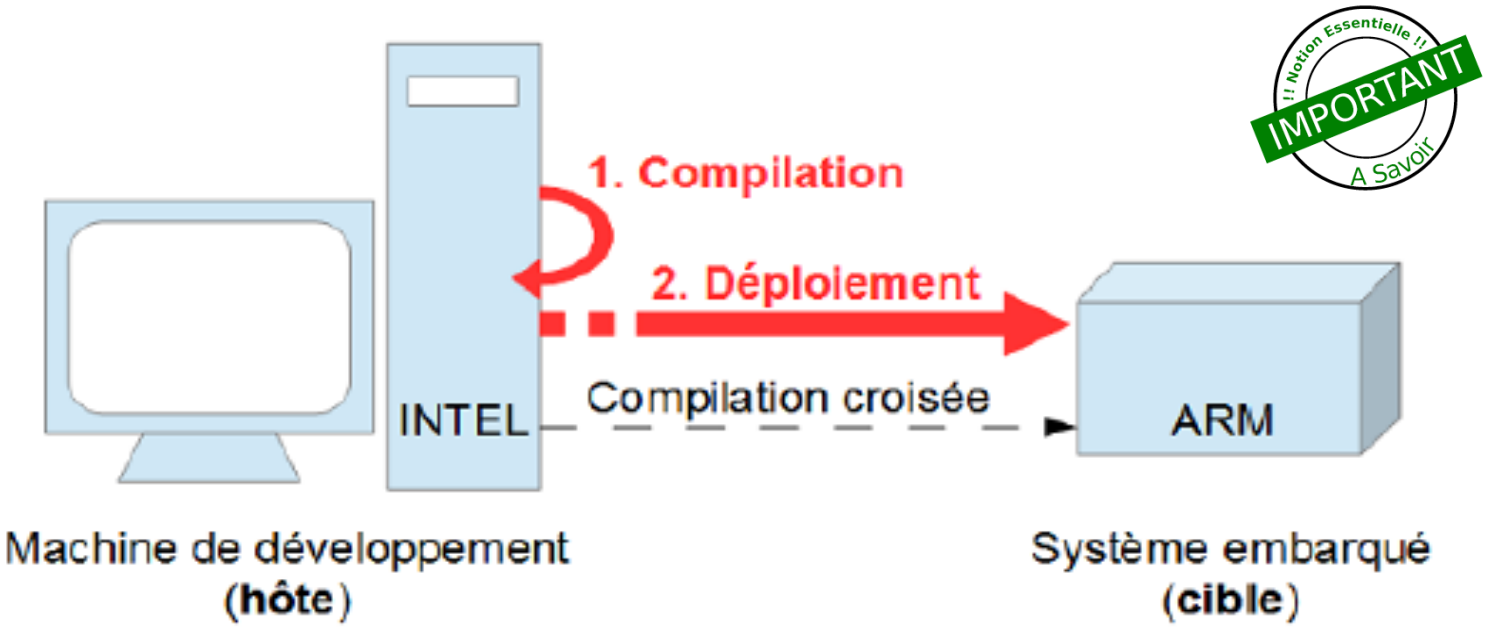
Le schéma ci-dessous résume le processus de compilation dans le cas d'un programme composé d'un programme principal `prog.c` qui utilise un module `mod.c` :



Exécution d'un programme binaire

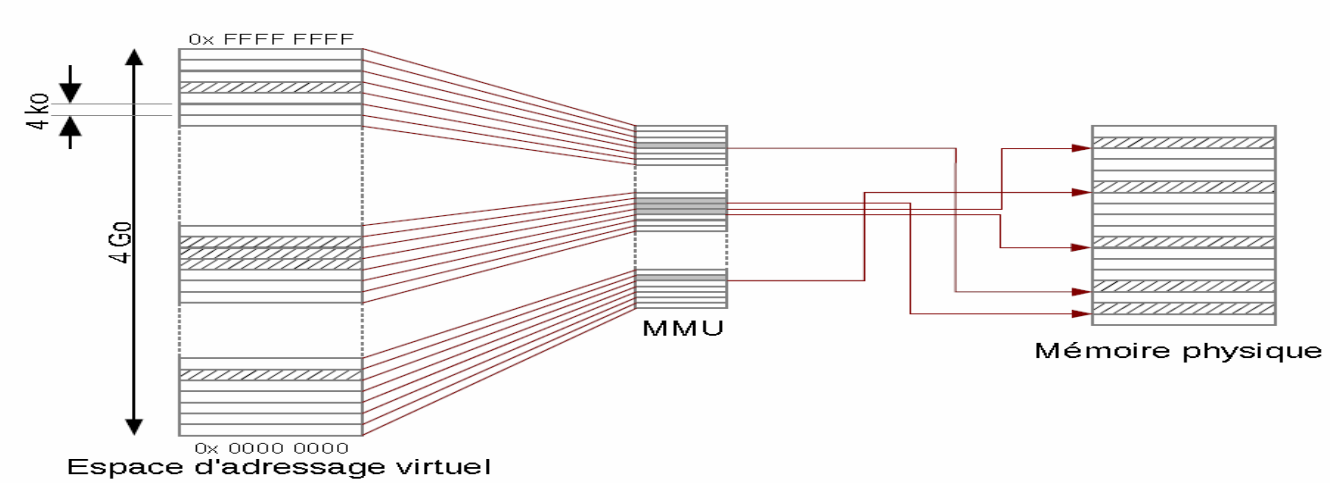
Une fois qu'on a généré le binaire de notre programme, on peut l'exécuter en tapant la commande `./bonjour` dans un terminal.

Chaîne de compilation croisée



La compilation croisée fait référence aux chaînes de compilation capables de traduire un code source en code objet dont l'architecture processeur diffère de celle où la compilation est effectuée.

Mémoire virtuelle et MMU : Fonctions assurées





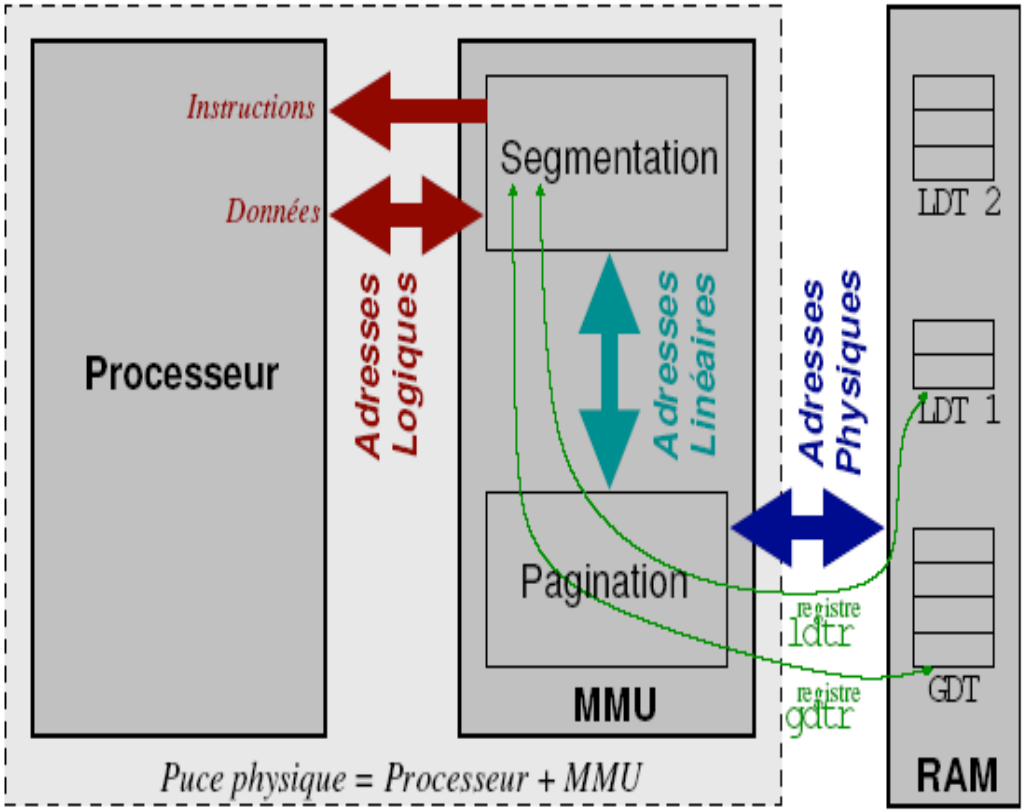
- **Translation d'adresses** :
  - **Segmentation** : subdivision des espaces d'adressage d'après leur fonctionnalités.
  - **Pagination** : subdivision des espaces d'adressage des tâches en petites tranches de taille fixe
- **Protection** : Chaque programme reste confiné dans son espace mémoire
- **Mémoire virtuelle** : création d'un espace d'adressage important incluant la mémoire physique et une partie de la mémoire secondaire (>mémoire physique)
- **Swapping** : des portions de la mémoire sont rapatriées et envoyées vers l'espace de stockage secondaire.

MMU : Origine

Dans les premiers microprocesseurs, il s'agissait d'un circuit intégré dédié.

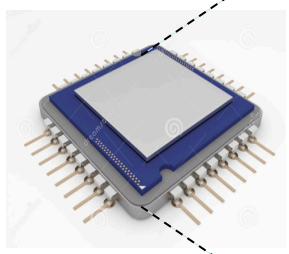
Puis le MMU a été intégré aux microprocesseurs à partir du 80386 (Intel) et le 68030 (Motorola)

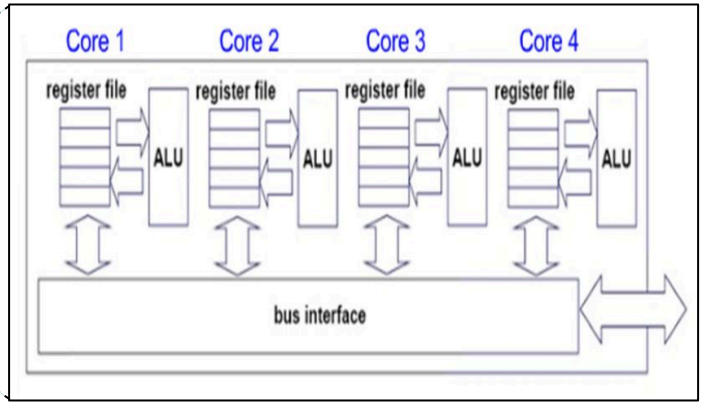


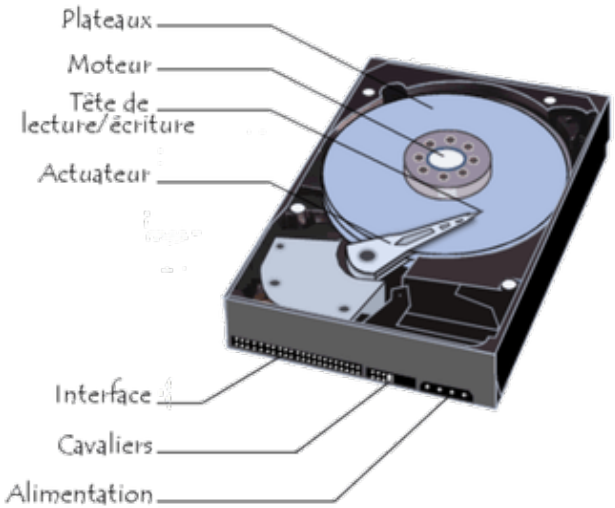


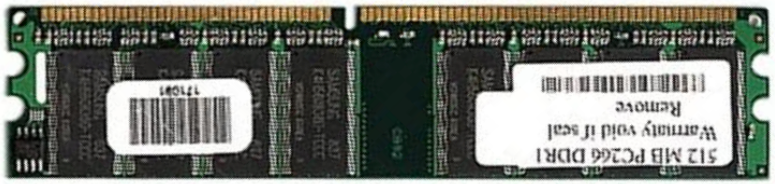
MMU : Origine

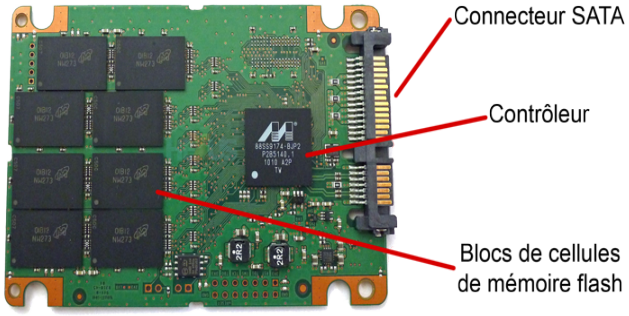
Rappel







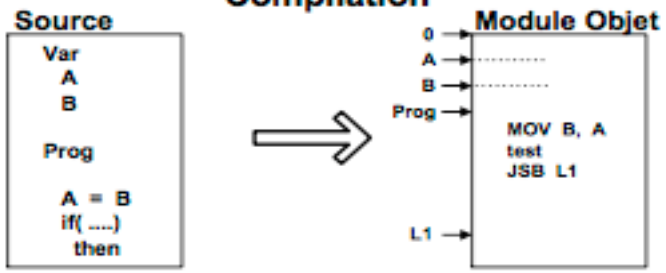




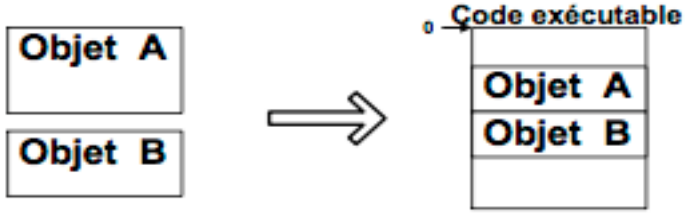
MMU : Origine

TRANSLATION D'ADRESSE

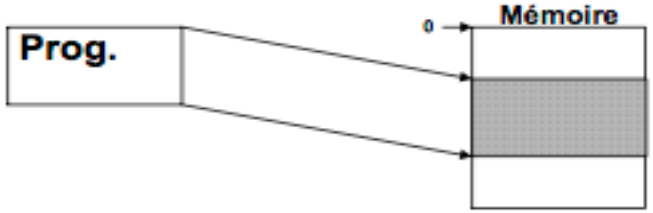
Compilation



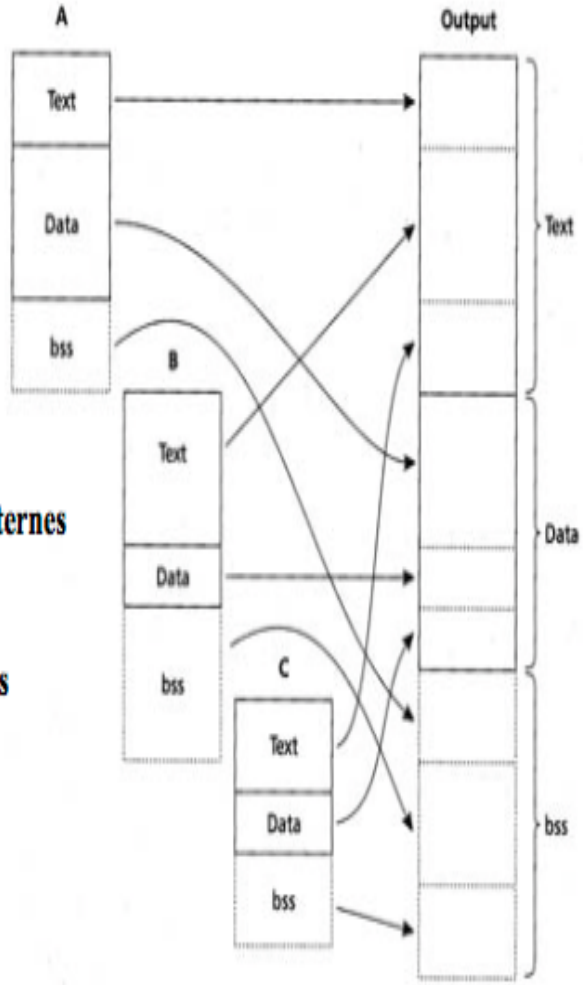
ÉDITEUR DE LIENS



CHARGEMENT EN MÉMOIRE



Édition des liens



Résoudre les références externes

L'assignation mémoire

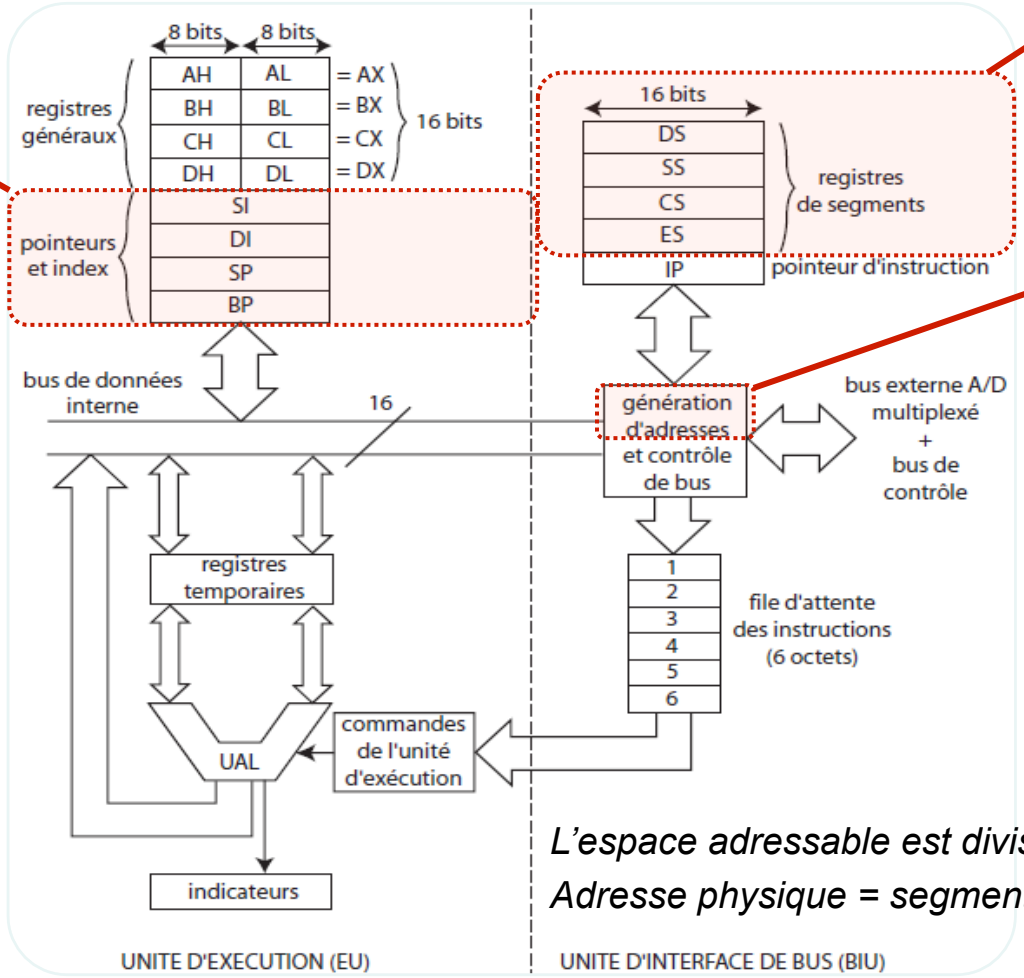
Relocalisation des symboles

Segmentation – Principe (origine - 8086)

Segments Registers or Segments Selectors

Index Registers and Stack manipulation registers

Segmentation Unit (physical address generation)



!! Notion Essentielle !! IMPORTANT A Savoir

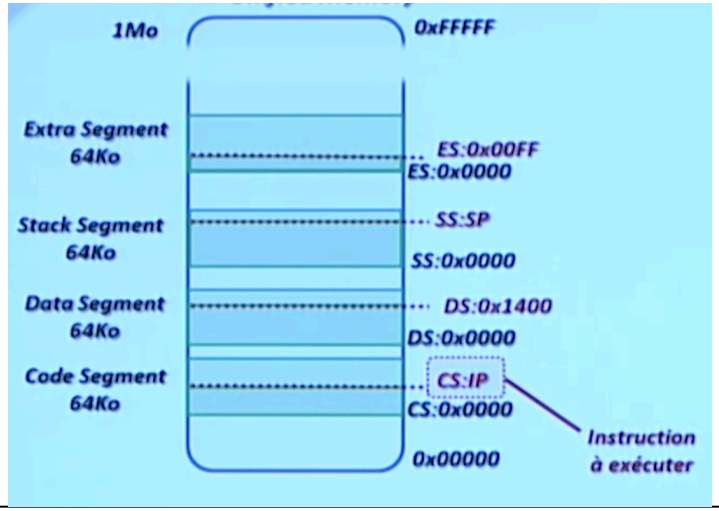
L'espace adressable est divisé en segments de 64Ko

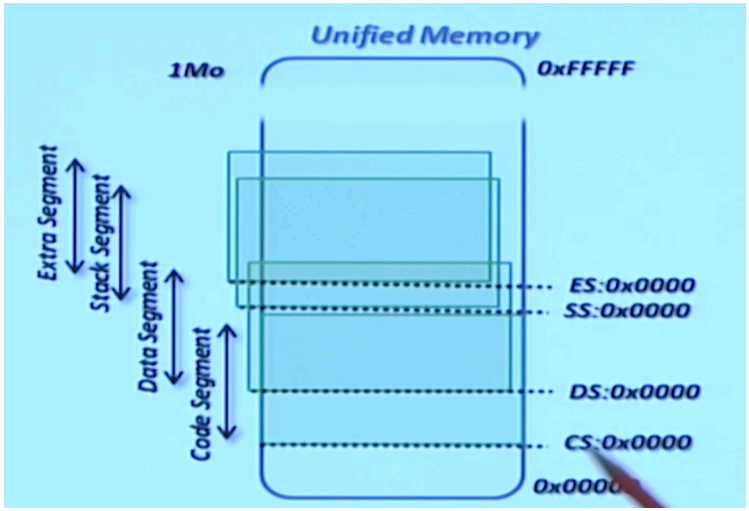
Adresse physique = segment << 4 + offset

Segmentation – Principe (origine - 8086)

Adresse physique = segment << 4 + offset

DS (Data Segment)	Segment:offset	Physical Address
0xFB0A	DS:0x0005	0xFB0A5
0xF000	DS:0xB0A5	0xFB0A5
0xF203	DS:0x9075	0xFB0A5
...	...	...





Segmentation – Principe (origine - 8086)

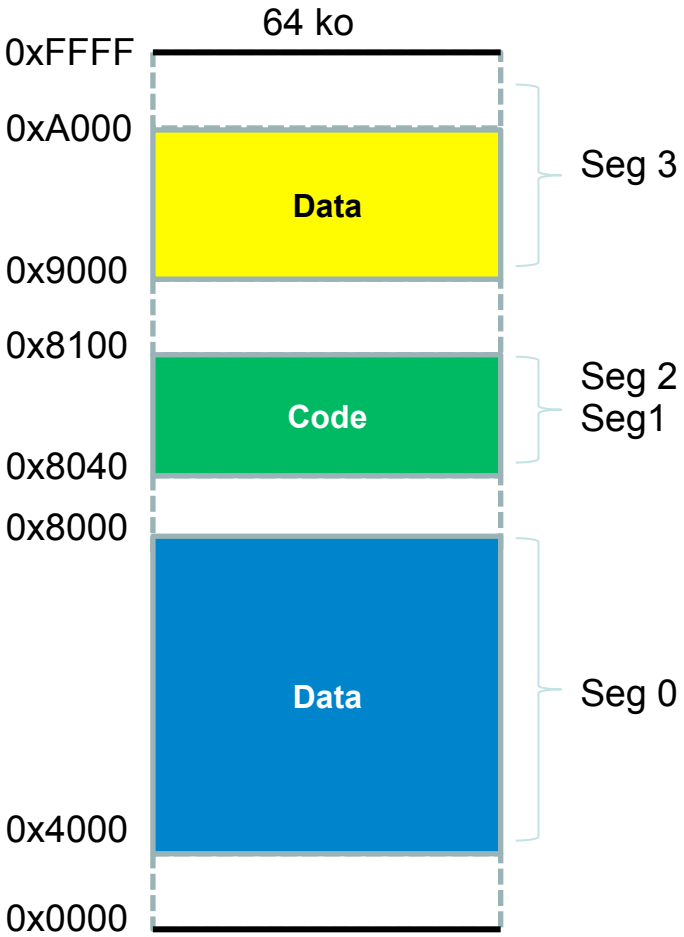


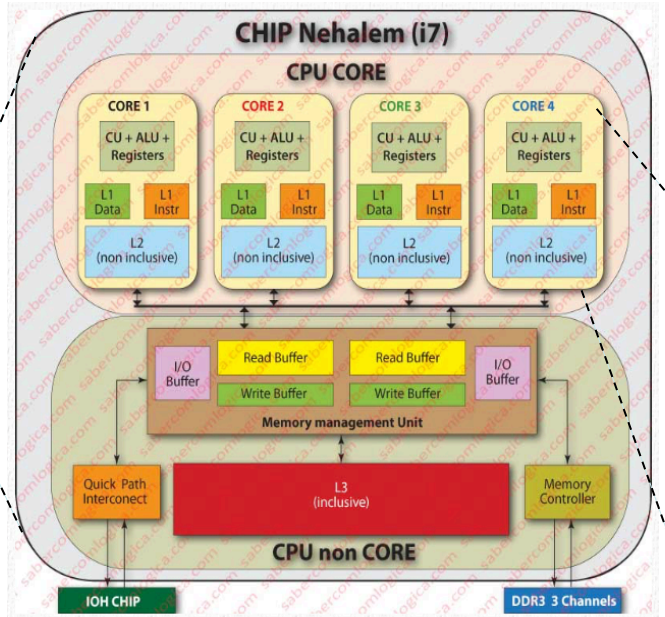
Table des segments du processus 1

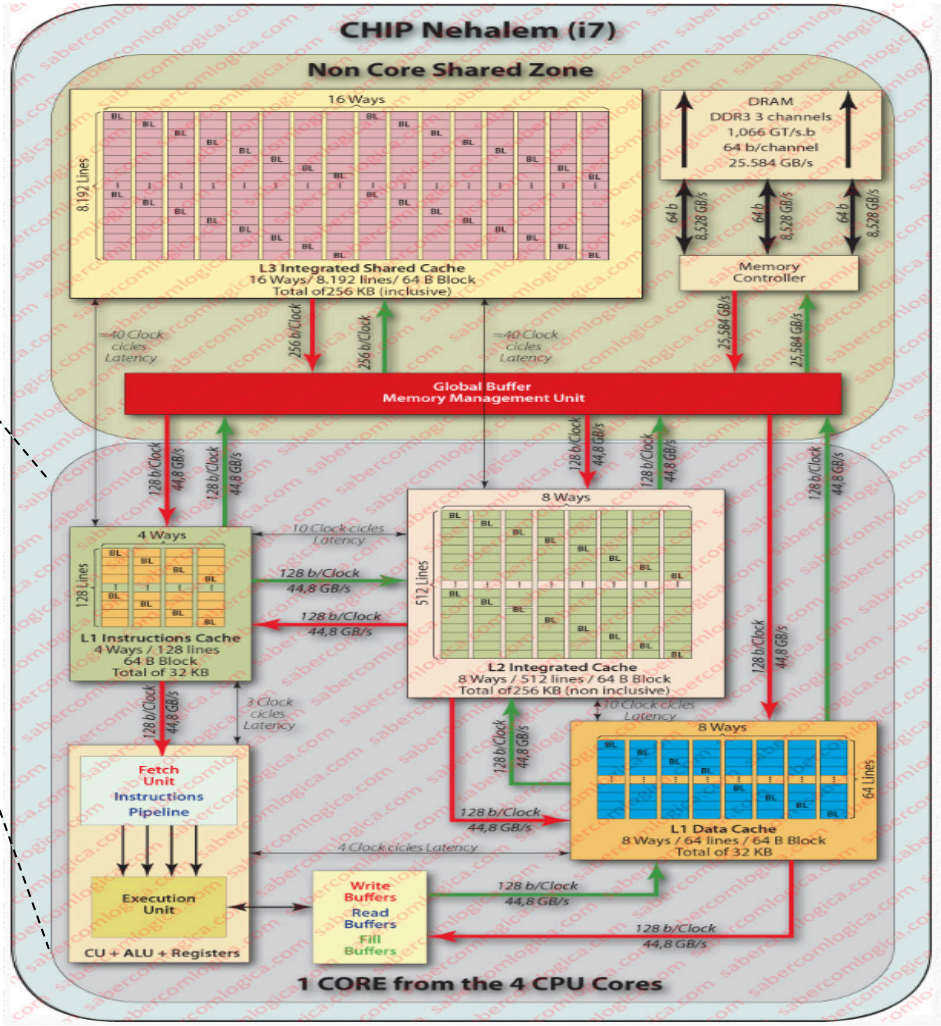
Index	Base	Limite	Type	Priv.
0	0x4000	0x3FFF	2	3
1	0x8040	0x00BF	10	3

Table des segments du processus 2

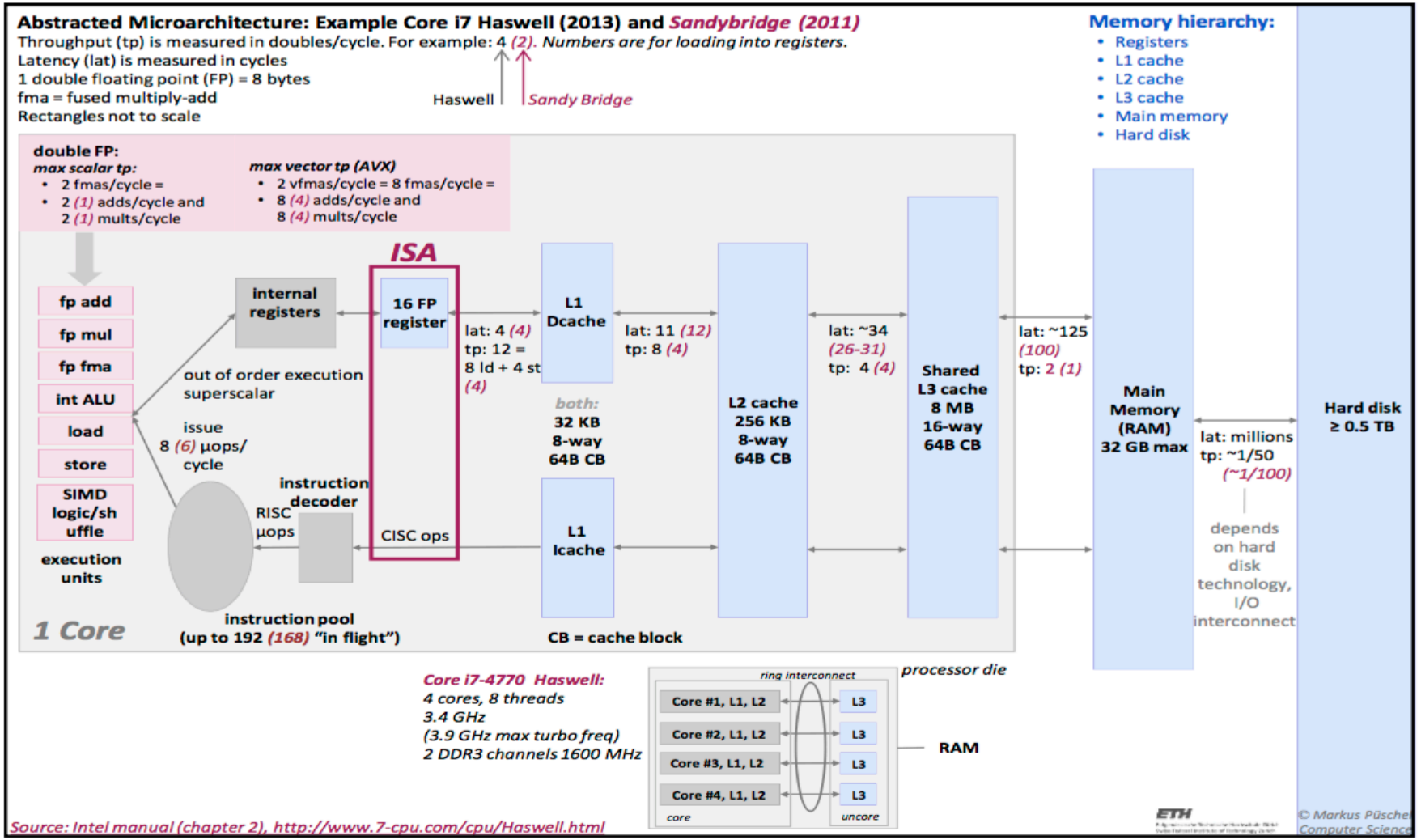
Index	Base	Limite	Type	Priv
2	0x8040	0x00BF	10	3
3	0x9000	0x0FFF	2	3

MMU (dans les processeurs modernes)



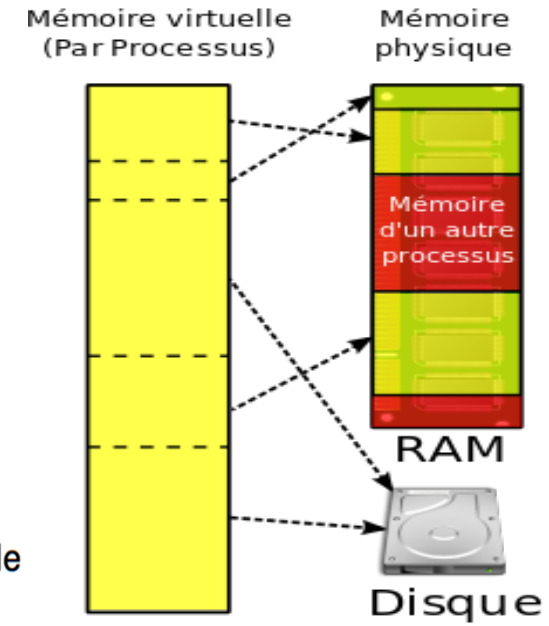


MMU (microarchitecture du processeur i7)



Mémoire virtuelle (MV)

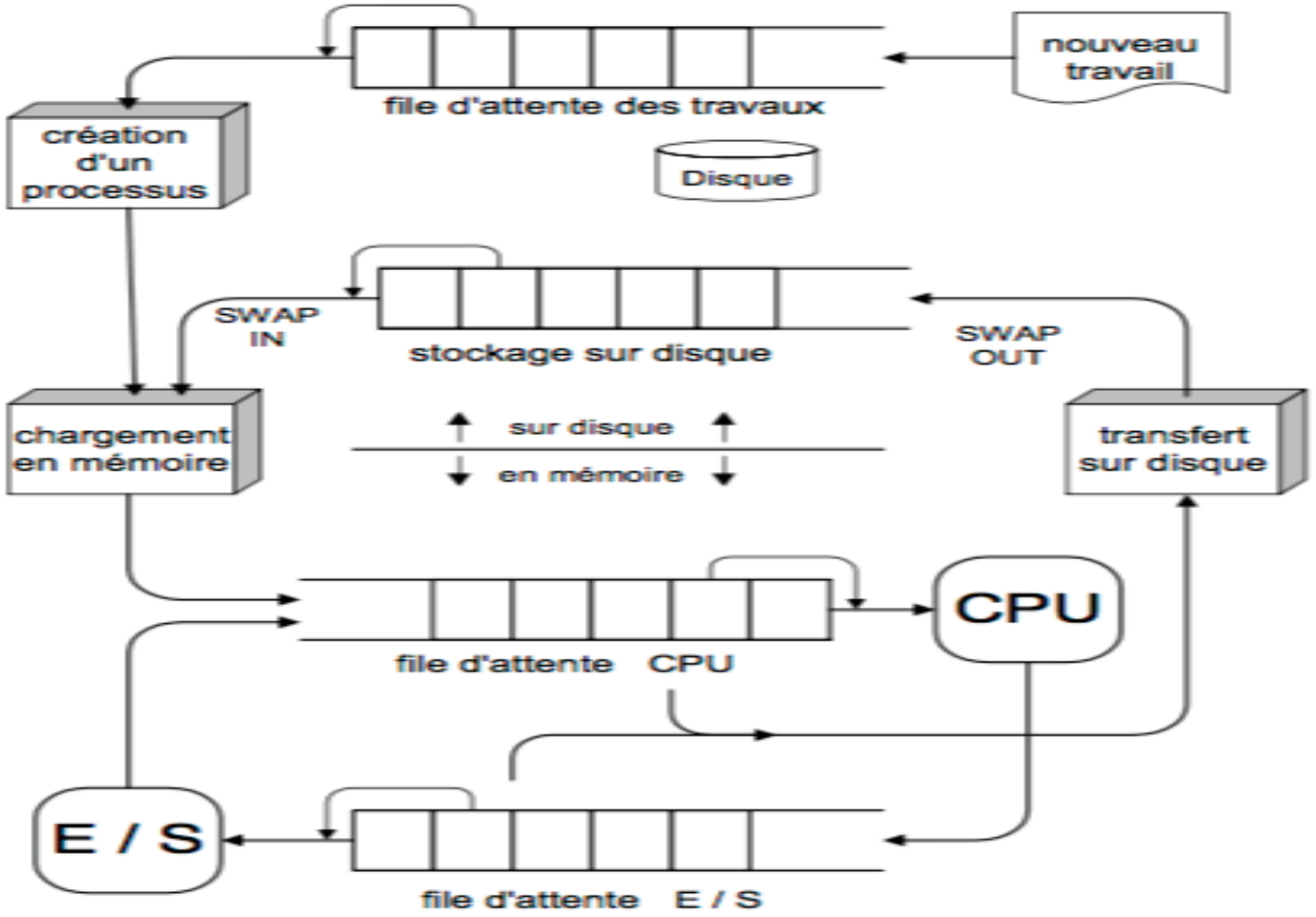
- Principe de la **mémoire virtuelle**: la taille de l'ensemble formé par le programme, les données et la pile **peut dépasser la capacité disponible de mémoire physique.**
- Le SE conserve les parties de programme en cours d'utilisation dans la mémoire principale et le reste sur le disque.
- Séparation entre la mémoire logique de l'utilisateur et la mémoire physique.
- Seulement une partie d'un programme a besoin d'être dans la mémoire pour l'exécution.
- Les programmes peuvent être plus grands que la mémoire physique.
- Rend plus facile la tâche de programmation: le programmeur n'a pas besoin de se préoccuper de la quantité de mémoire physique disponible.
- Système de gestion de la mémoire (*Memory manager*): partie du SE qui gère la hiérarchie de stockage
  - Suivre les parties de la mémoire qui sont utilisées ou non utilisées.
  - Allouer/libérer espace mémoire aux processus.
  - Contrôler le *swapping* entre la mémoire principale et le disque.



Implantation

- Pagination à la demande
- Segmentation à la demande

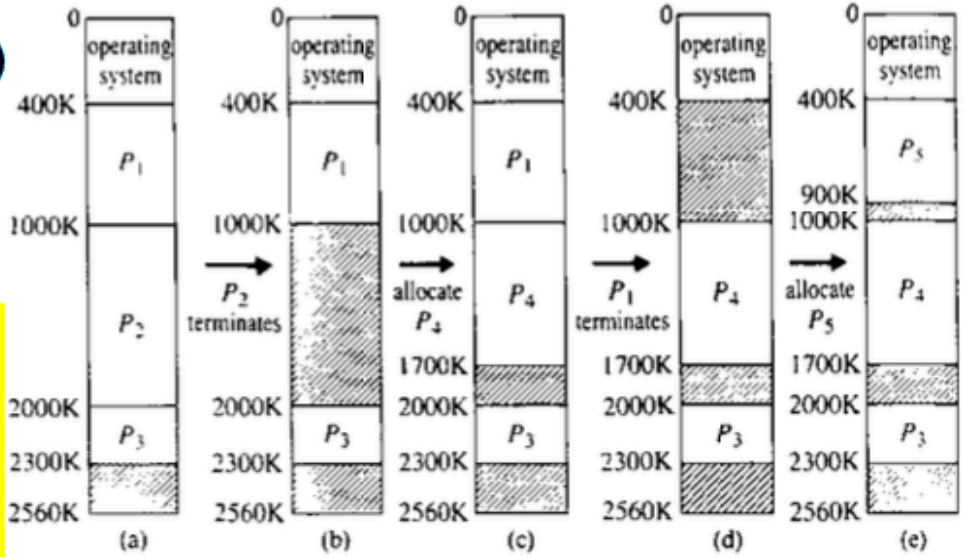
SE, MV et MMU : Allocation

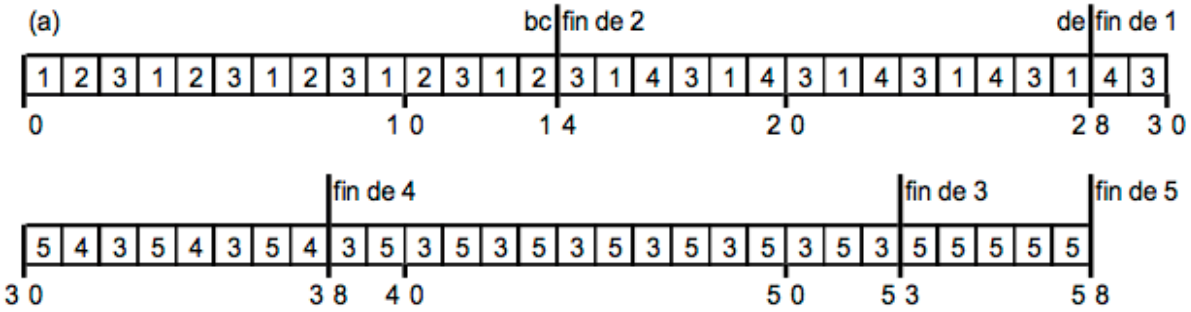


SE, MV et MMU : Allocation

- chargement FCFS (allocation des travaux)
- allocation mémoire First-Fit
- allocation du CPU Round-Robin

travaux	mémoire	temps cpu
1	600 K	10
2	1000K	5
3	300 K	20
4	700 K	8
5	500 K	15



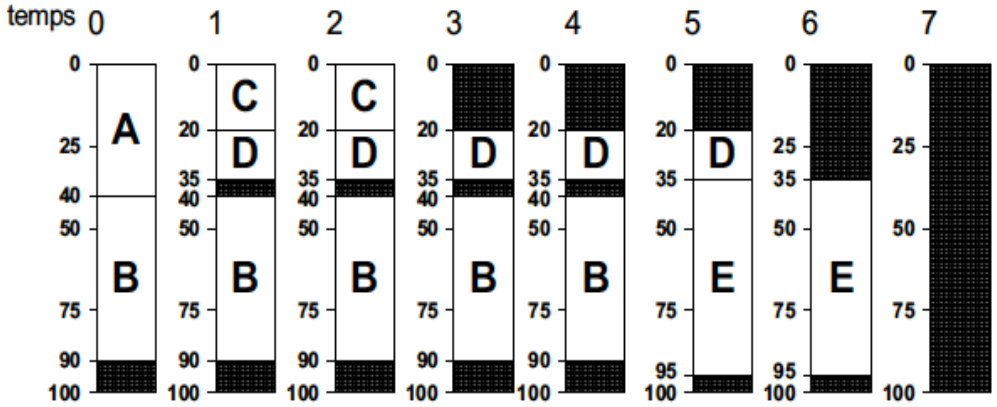


SE, MV et MMU : Allocation

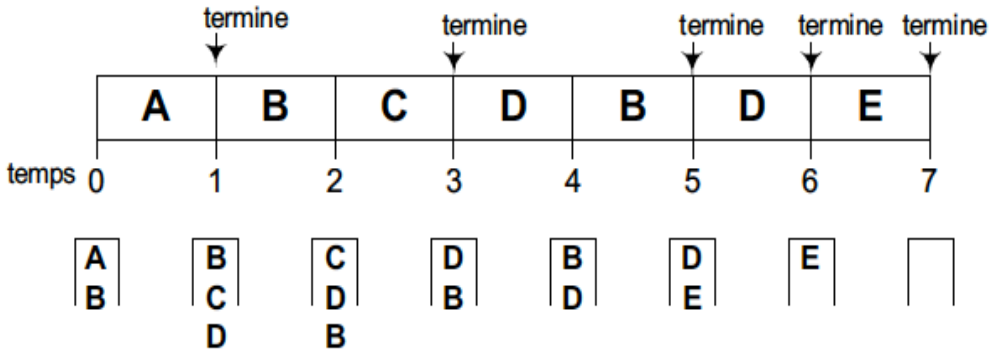
+ chargement FCFS
+ allocation de la mémoire avec First-Fit
+ allocation du CPU avec Round-Robin (Q=1)

travail	mémoire	CPU
A	40 K	1 ut
B	50 K	2 ut
C	20 K	1 ut
D	15 K	2 ut
E	60 K	1 ut

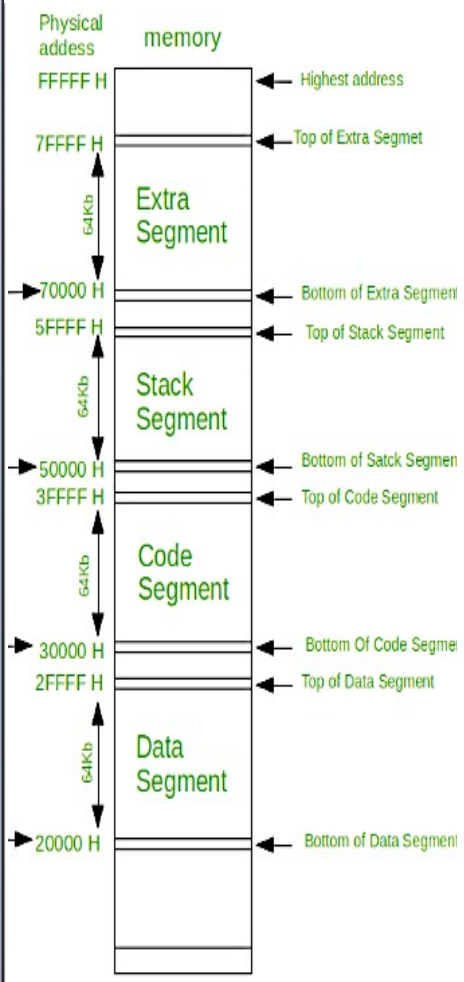
Allocation de la mémoire



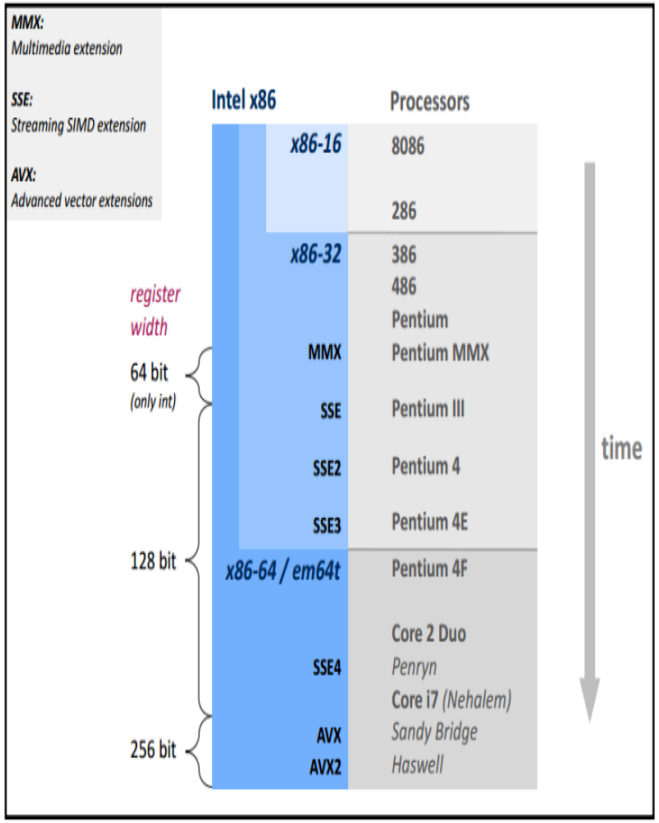
Allocation du CPU



Mécanisme de segmentation (dans les processeurs x86)

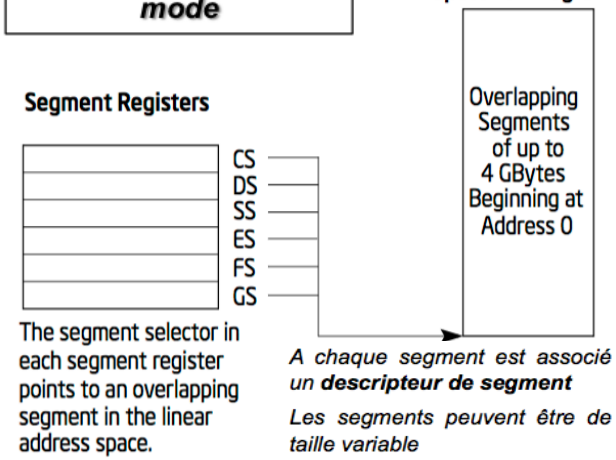


Real addressing mode



Sur processeurs x86-64 modernes, les sélecteurs de segment font toujours 16 bits, néanmoins de nouveaux sélecteurs ont été ajoutés (FS et GS).

Segmented adressing mode



*A chaque segment est associé un **descripteur de segment***

Les segments peuvent être de taille variable

Flat adressing mode

Ce modèle inhibe la génération d'exceptions via l'unité de segmentation, néanmoins il offre une meilleure flexibilité, minimise l'utilisation de ressources hardware (MMU) et optimise la vitesse d'exécution.

*Ce modèle autorise la pagination et est celui implémenté par exemple par **Linux***

Mécanisme de segmentation : Limites et solutions

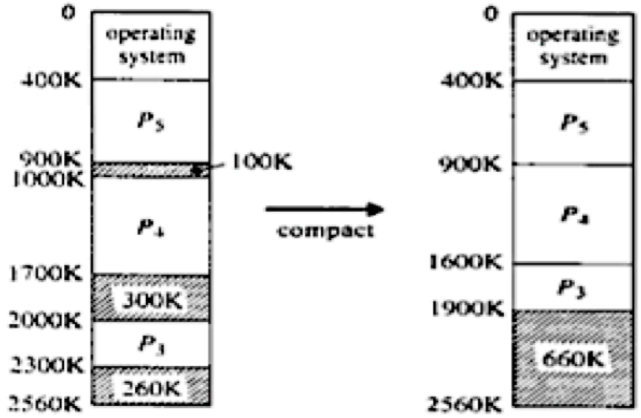
Real addressing mode → Taille des segments fixes (46 Ko) → **fragmentation interne**

Segmented adressing mode → Taille des segments variables → **fragmentation externe**

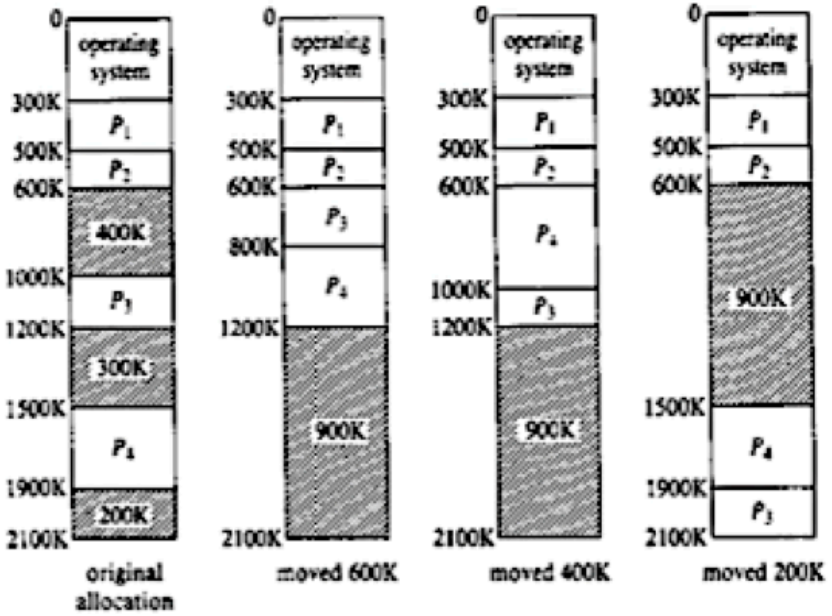
- Fragmentation → perte de mémoire
- Défragmentation → perte de temps à l'exécution

Solutions curatives **compactage**

Décaler les blocs alloués pour les ramener en début de mémoire, et ainsi regrouper toutes les zones libres en une seule (oblige à interrompre les processus concernes pendant ce temps).



Efficacité → minimiser les déplacements

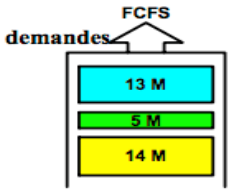


Mécanisme de segmentation : Limites et solutions

Solutions préventives

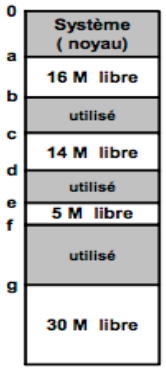
Algorithme FIRST-FIT

Choisi le **PREMIER** espace libre pouvant contenir le processus à partir du **DÉBUT** de la liste des espaces libres



Liste des espaces libres

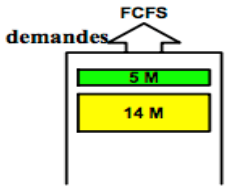
adresse de début	taille
a	16 M
c	14 M
e	5 M
g	30 M



Nous utilisons
- une liste ordonnée selon l'adresse
- reprise de la recherche au début de la liste

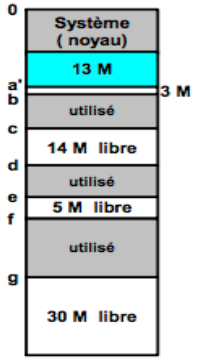
? autres critères

Choisi le **PREMIER** espace libre pouvant contenir le processus à partir du **DÉBUT** de la liste des espaces libres



Liste des espaces libres

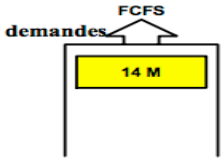
adresse de début	taille
a'	3 M
c	14 M
e	5 M
g	30 M



Nous utilisons
- une liste ordonnée selon l'adresse
- reprise de la recherche au début de la liste

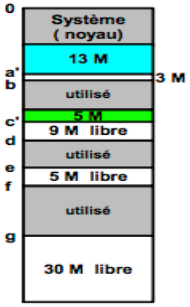
Algorithme FIRST-FIT

Choisi le **PREMIER** espace libre pouvant contenir le processus à partir du **DÉBUT** de la liste des espaces libres



Liste des espaces libres

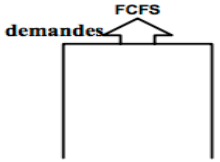
adresse de début	taille
a'	3 M
c'	9 M
e	5 M
g	30 M



Nous utilisons
- une liste ordonnée selon l'adresse
- reprise de la recherche au début de la liste

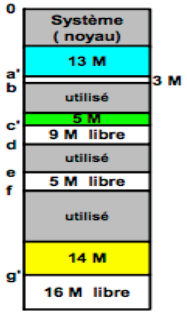
Algorithme FIRST-FIT

Choisi le **PREMIER** espace libre pouvant contenir le processus à partir du **DÉBUT** de la liste des espaces libres



Liste des espaces libres

adresse de début	taille
a'	3 M
c'	9 M
e	5 M
g'	16 M



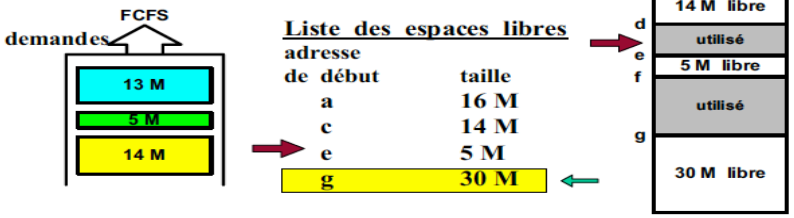
Nous utilisons
- une liste ordonnée selon l'adresse
- reprise de la recherche au début de la liste

Mécanisme de segmentation : Limites et solutions

Solutions préventives

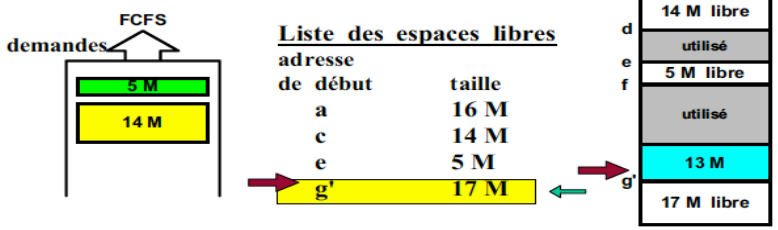
Algorithme NEXT-FIT

Choisi le PREMIER espace libre pouvant contenir le processus SUIVANT la dernière insertion dans la liste des espaces libres



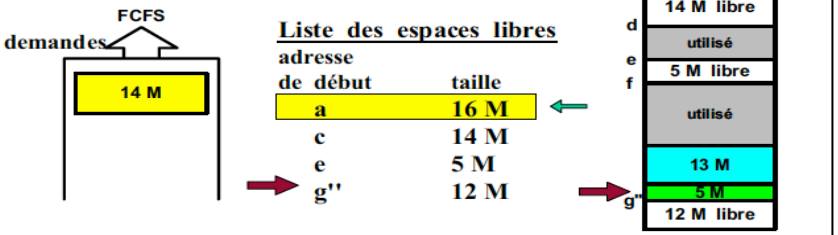
Nous utilisons
- une liste ordonnée selon l'adresse
- reprise de la recherche à la suite de la dernière insertion

Choisi le PREMIER espace libre pouvant contenir le processus SUIVANT la dernière insertion dans la liste des espaces libres



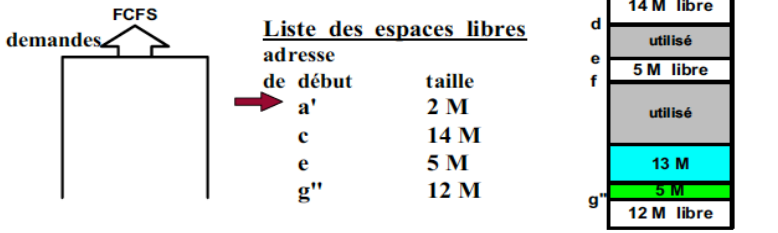
Nous utilisons
- une liste ordonnée selon l'adresse
- reprise de la recherche à la suite de la dernière insertion

Choisi le PREMIER espace libre pouvant contenir le processus SUIVANT la dernière insertion dans la liste des espaces libres



Nous utilisons
- une liste ordonnée selon l'adresse
- reprise de la recherche à la suite de la dernière insertion

Choisi le PREMIER espace libre pouvant contenir le processus SUIVANT la dernière insertion dans la liste des espaces libres



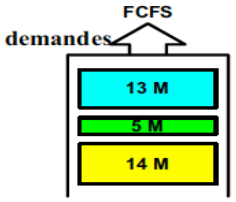
Nous utilisons
- une liste ordonnée selon l'adresse
- reprise de la recherche à la suite de la dernière insertion

Mécanisme de segmentation : Limites et solutions

Solutions préventives

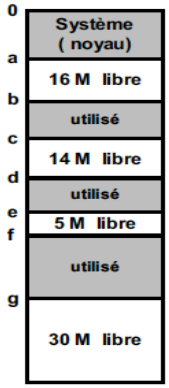
Algorithme BEST-FIT

Choisi le PLUS PETIT espace libre pouvant contenir le processus dans la liste des espaces libres



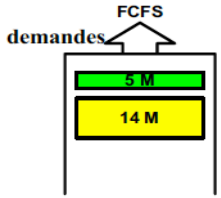
Liste des espaces libres

adresse de début	taille
e	5 M
c	14 M
a	16 M
g	30 M



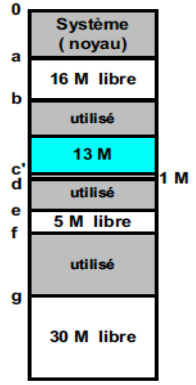
Nous utilisons une liste ordonnée selon la taille la plus petite

Choisi le PLUS PETIT espace libre pouvant contenir le processus dans la liste des espaces libres



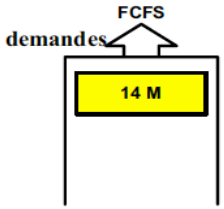
Liste des espaces libres

adresse de début	taille
c'	1 M
e	5 M
a	16 M
g	30 M



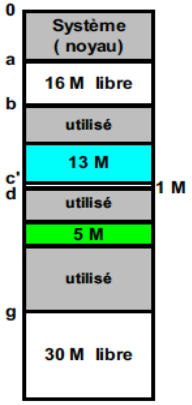
Nous utilisons une liste ordonnée selon la taille la plus petite

Choisi le PLUS PETIT espace libre pouvant contenir le processus dans la liste des espaces libres



Liste des espaces libres

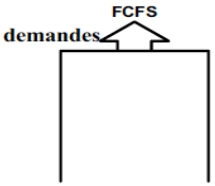
adresse de début	taille
c'	1 M
a	16 M
g	30 M



Nous utilisons une liste ordonnée selon la taille la plus petite

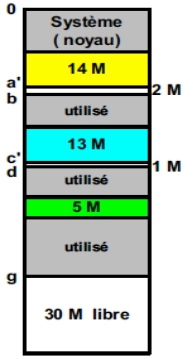
Algorithme BEST-FIT

Choisi le PLUS PETIT espace libre pouvant contenir le processus dans la liste des espaces libres



Liste des espaces libres

adresse de début	taille
c'	1 M
a'	2 M
g	30 M



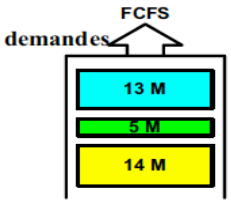
Nous utilisons une liste ordonnée selon la taille la plus petite

Mécanisme de segmentation : Limites et solutions

Solutions préventives

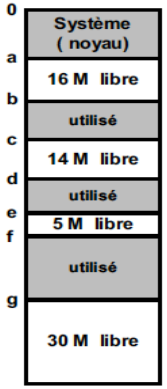
Algorithme WORST-FIT

Choisi le **PLUS GRAND** espace libre pouvant contenir le processus dans la liste des espaces libres



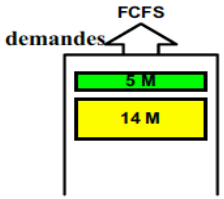
Liste des espaces libres

adresse de début	taille
g	30 M
a	16 M
c	14 M
e	5 M



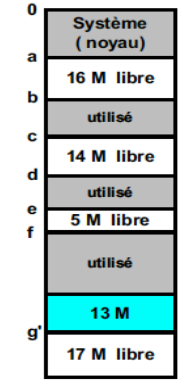
Nous utilisons une liste ordonnée selon la taille la plus grande

Choisi le **PLUS GRAND** espace libre pouvant contenir le processus dans la liste des espaces libres



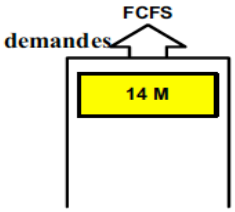
Liste des espaces libres

adresse de début	taille
g'	17 M
a	16 M
c	14 M
e	5 M



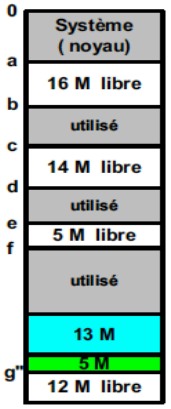
Nous utilisons une liste ordonnée selon la taille la plus grande

Choisi le **PLUS GRAND** espace libre pouvant contenir le processus dans la liste des espaces libres



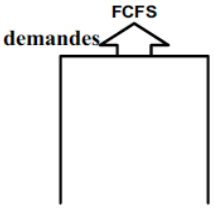
Liste des espaces libres

adresse de début	taille
a	16 M
c	14 M
g''	12 M
e	5 M



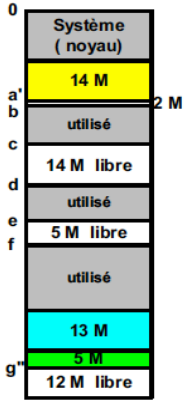
Nous utilisons une liste ordonnée selon la taille la plus grande

Choisi le **PLUS GRAND** espace libre pouvant contenir le processus dans la liste des espaces libres



Liste des espaces libres

adresse de début	taille
c	14 M
g''	12 M
e	5 M
a'	2 M



Nous utilisons une liste ordonnée selon la taille la plus grande

Mécanisme de segmentation : Limites et solutions

Buddy system (blocs compagnons)

Le principe est d'éviter les blocs de tailles trop différentes. Pour cela on arrondit les demandes d'allocation mémoire à la puissance de 2 supérieure (au dessus d'une valeur minimale raisonnable, par exemple 1K octet).

Pour chaque taille, on maintient une liste de blocs : L_{10} (blocs de $2^{10} = 1024$ octets), L_{11} (2048 octets), etc. qui commencent tous a une adresse qui est un multiple de leur taille. Par exemple les blocs de L_{11} peuvent commencer à l'adresse 0, 2K, 4K, 6K, etc.

Pour allouer un espace de 3000 octets, la demande est arrondie à la puissance de 2 supérieure, soit $2^{12} = 4096$. On prend donc un bloc libre dans la liste L_{12} des blocs libres de 4096 octets (si il n'y en a pas, on prendra dans la liste du dessus L_{13}, etc.) commençant par exemple à l'adresse $12288 = 3 \ 2^{12}$.

L'excédent par rapport à la demande est de 4096 - 3000 = 1096 : on peut en retirer un bloc de 1024 octets qui est remis dans L_{10}. Le bloc réellement alloué est de 4096 - 1024 = 3072 octets.

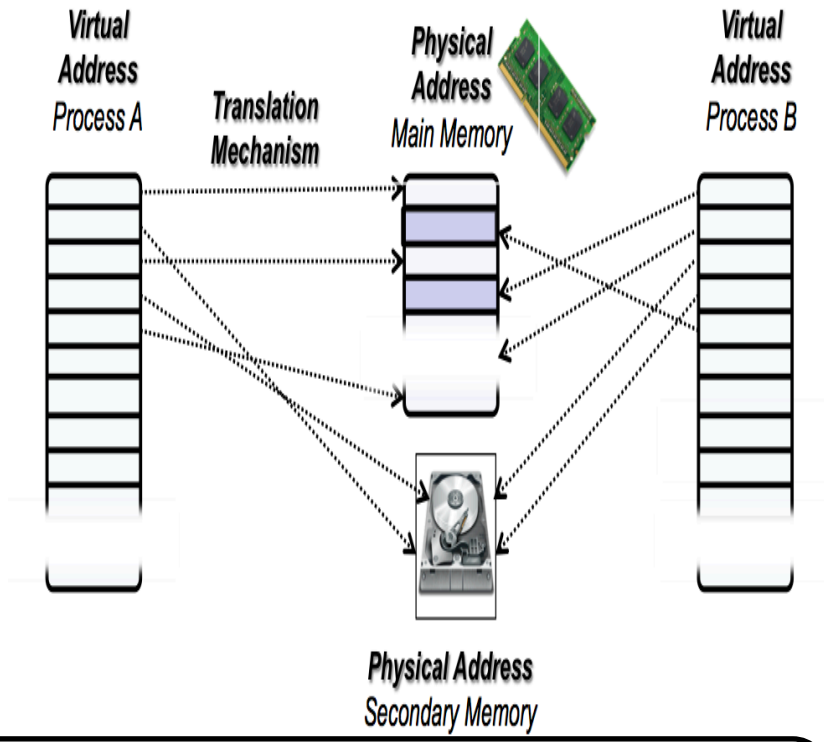
Pour restituer ce bloc de 3072 octets, il est fragmente en un bloc de taille 2048 (2^{11}) commençant à l'adresse $12288 = 3 \times 2^{12} = 6 \times 2^{11}$ qui est remis dans L_{11}, et un bloc de 1024 (2^{10}) commençant en $12288 + 2048 = 3 \times 2^{12} + 2^{11} = 14 \times 2^{10}$ destiné à L_{10}.
Or ce dernier bloc est la moitié d'un bloc deux fois plus grand. Si l'autre moitié (son bloc compagnon, qui commence en 15×2^{10}) est libre, on peut les fusionner en un bloc libre de 2^{11} qui ira dans L_{11}.

Mécanisme de Pagination

La plupart des implémentations de mémoire virtuelle divisent un espace d'adressage virtuel en blocs d'adresses de mémoire virtuelle contiguës, appelées ***pages***, qui ont généralement une taille de ***4 Ko***.

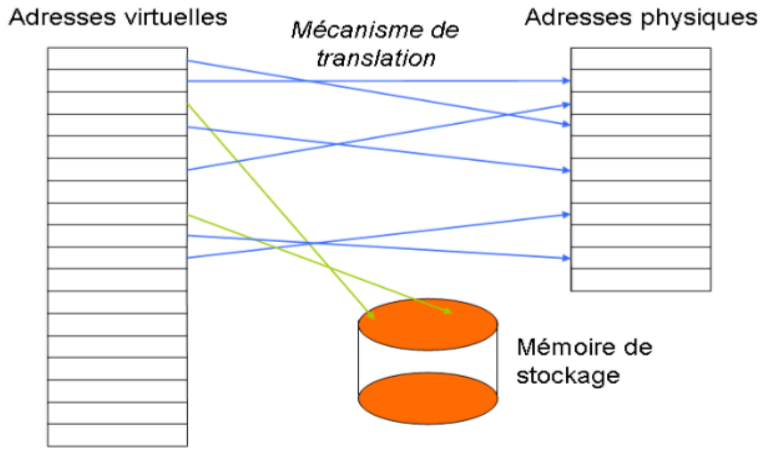
Afin de traduire les adresses virtuelles d'un processus en adresses de mémoire physique utilisées par le matériel pour réellement traiter les instructions, la MMU utilise ce que l'on appelle une table de pages, c'est-à-dire une structure de données gérée par le système d'exploitation qui stocke les mappages entre les adresses virtuelles et physiques .

Concrètement, la MMU stocke un cache des mappages récemment utilisés parmi ceux stockés dans l'ensemble de la table de pages du système d'exploitation, qui est appelé ***Translation Lookaside Buffer (TLB)***.

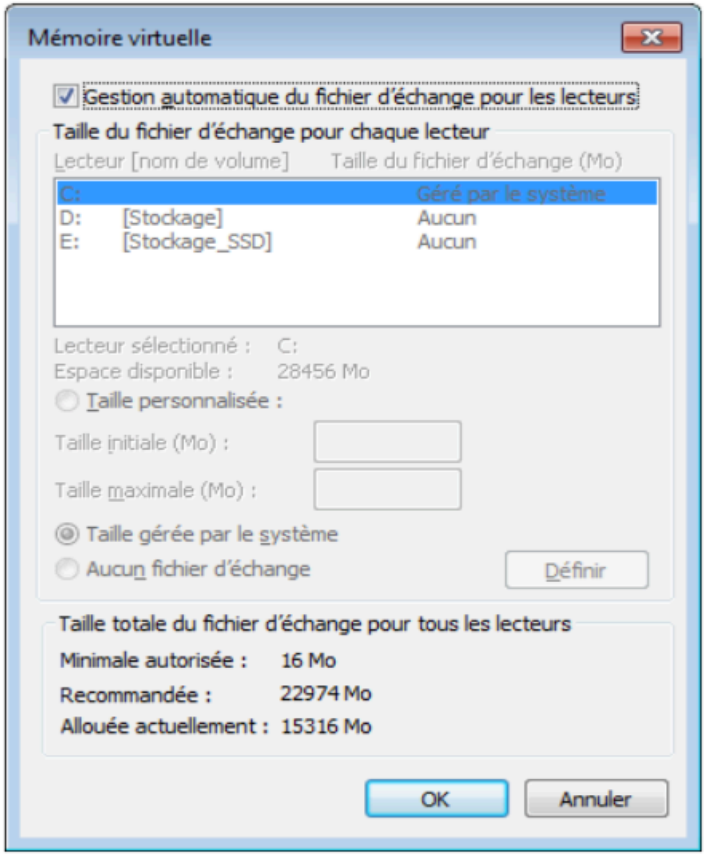


- Une mémoire paginée est découpée en pages fixant ainsi une granularité de la mémoire.
- La mémoire principale est découpée en frames (cadres) de même taille, chaque cadre contenant une page.
- Il peut y avoir plus de pages que de cadres
- Les architectures Intel modernes supportent des pages de différentes tailles : 4Ko, 2Mo, 4Mo ou 1Go (dépend du mode de pagination)

Mécanisme de Pagination

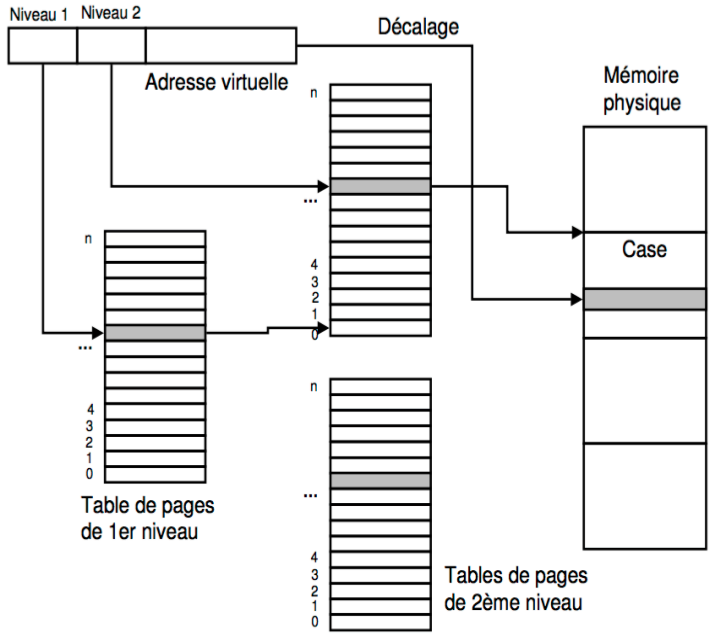


- Une mémoire paginée est découpée en pages fixant ainsi une granularité de la mémoire.
- La mémoire principale est découpée en frames (cadres) de même taille, chaque cadre contenant une page.
- Il peut y avoir plus de pages que de cadres
- Le stockage des pages sur le disque dur se fait dans un fichier temporaire appelé fichier **SWAP** quand la mémoire vive n'est plus suffisante.
- Les architectures Intel modernes supportent des pages de différentes tailles : 4Ko, 2Mo, 4Mo ou 1Go (dépend du mode de pagination)



Mécanisme de Pagination

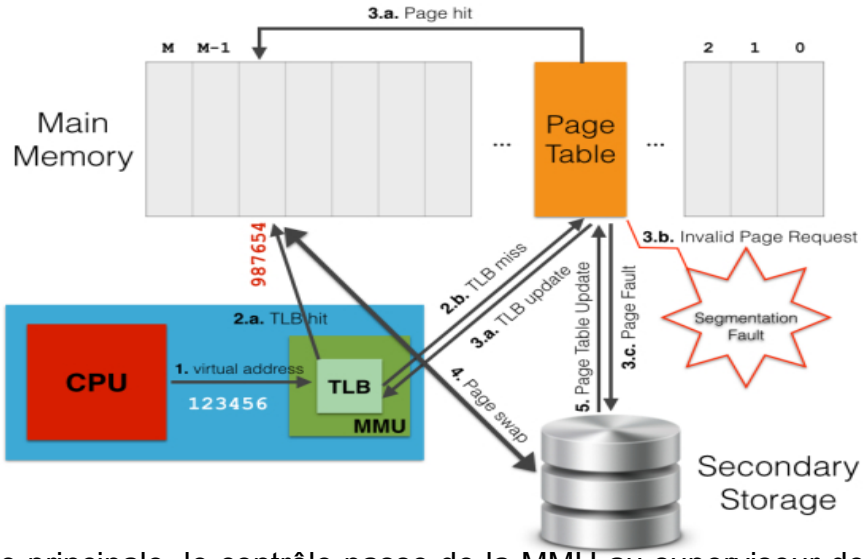
- La taille de la table de pages peut être très grande : par exemple, on au- rait plus de 1 million d'entrées (2^{20}) pour un adressage virtuel sur 32 bits et des pages de 4 Ko. Pour éviter d'avoir des tables trop grandes en mémoire, de nombreux ordinateurs utilisent des tables des pages à plusieurs niveaux.
- Par exemple, une table de pages à deux niveaux, pour un adressage sur 32 bits et des pages de 4 Ko, est composée de 1024 tables de 1 Ko. Il est ainsi possible de charger uniquement les tables de 1 Ko nécessaires.
- L'accès à la table de pages (qui se trouve en mémoire) peut se faire via un registre du MMU qui pointe sur la table de pages. Ceci est une solution lente, car elle nécessite des accès mémoire.



- La solution consiste à doter le MMU d'un dispositif appelé TLB (sorte de mémoire associative), qui est composé d'un petit nombre d'entrées, normalement entre 8 et 64 (la plupart des programmes ont tendance à faire un grand nombre de références à un petit nombre de pages). Le TLB contient des circuits spéciaux pour accéder aux adresses qui sont hautement référencées.

Mémoire virtuelle et MMU : Pagination

Lorsqu'une adresse virtuelle doit être traduite en une adresse physique, la MMU la recherche d'abord dans le cache TLB. Si une correspondance est trouvée (*TLB hit*) alors l'adresse physique est retournée et le calcul continue simplement. Inversement, s'il n'y a pas de correspondance pour l'adresse virtuelle dans le cache TLB (*TLB miss*), la MMU recherche une correspondance sur la table de page entière. Si cette correspondance existe sur la table des pages, elle est en conséquence écrite dans le cache TLB. Ainsi, la traduction d'adresse est redémarrée afin que la MMU puisse trouver une correspondance sur le TLB mis à jour.



Si la page demandée n'est pas chargée dans la mémoire principale, le contrôle passe de la MMU au superviseur de page (dans le noyau du système d'exploitation). qui va détecté que la page demandée doit être récupérée à partir du stockage secondaire (le disque) où elle est actuellement stockée. Ainsi, le superviseur de page accède au disque, stocke dans la mémoire principale la page correspondant à l'adresse virtuelle à l'origine du défaut de page, met à jour la table de pages et le TLB avec un nouveau mappage entre l'adresse virtuelle et l'adresse physique où la page a été stockée, et enfin dit à la MMU de relancer la requête afin qu'un hit TLB ait lieu.

En fait, la tâche ci-dessus fonctionne tant qu'il y ait suffisamment de place dans la mémoire principale pour stocker les pages du disque. Cependant, lorsque toute la mémoire physique est épuisée, le superviseur de page doit également libérer une page dans la mémoire principale pour permettre le stockage de la page entrante du disque. Pour déterminer correctement la page à déplacer de la mémoire principale vers le disque, le superviseur de pagination peut utiliser plusieurs algorithmes de remplacement de page, tels que le moins récemment utilisé (LRU).

Mémoire virtuelle et MMU : Pagination

- A la suite d'un défaut de page, le système d'exploitation doit ramener en mémoire la page manquante à partir du disque. S'il n'y a pas de cadres libres en mémoire, il doit retirer une page de la mémoire pour la remplacer par celle demandée. Si la page à retirer a été modifiée depuis son chargement en mémoire, il faut la réécrire sur le disque.
- Le choix de la page à remplacer peut se limiter aux pages du processus qui ont provoqué le défaut de page (allocation locale) ou à l'ensemble des pages en mémoire (allocation globale).
- En général, l'allocation globale produit de meilleurs résultats que l'al- location locale. Des algorithmes de remplacement de page ont été proposés.
- Soit la suite de références de pages virtuelles suivante : 2, 1, 0, 7, 1, 6, 1, 5, 7, 6, 2, 6, 7, 0, 7, 1, 0, 2, 1, 7
  On désire comparer les performances des algorithmes de gestion de remplacement de pages

1)

L'algorithme OPT (OPTimal) consiste à choisir comme victime la page qui sera référencée le plus tardivement dans le futur

Page	2	1	0	7	1	6	1	5	7	6	2	6	7	0	7	1	0	2	1	7
Cadre 0	**2**	2	2	2	2	**6**	6	6	6	6	6	6	6	**0**	0	0	0	0	1	1
Cadre 1	-	**1**	1	1	1	1	1	**5**	5	5	**2**	2	2	2	2	2	2	2	2	2
Cadre 2	-	-	**0**	**7**	7	7	7	7	7	7	7	7	7	7	7	**1**	7	7	7	7
Déf. de page	O	O	O	O	N	O	N	O	N	N	O	N	N	O	N	O	N	N	N	N



Mémoire virtuelle et MMU : Pagination

2)

L'algorithme FIFO (First In First Out) consiste à choisir comme victime la page la plus ancienne en mémoire

Page	2	1	0	7	1	6	1	5	7	6	2	6	7	0	7	1	0	2	1	7
Cadre 0	**2**	2	2	**7**	7	7	7	**5**	5	5	**2**	2	2	2	2	**1**	1	1	1	1
Cadre 1	-	**1**	1	1	1	**6**	6	6	**7**	7	7	7	7	**0**	0	0	0	**2**	2	2
Cadre 2	-	-	**0**	0	0	0	**1**	1	1	**6**	6	6	6	6	**7**	7	7	7	7	7
Déf. de page	O	O	O	O	N	O	O	O	O	O	O	N	N	O	O	O	N	O	N	N

3)

L'algorithme LRU (Least Recently Used/la moins récemment utilisée) consiste à choisir comme victime la page qui n'a pas été référencée depuis le plus longtemps.

Page	2	1	0	7	1	6	1	5	7	6	2	6	7	0	7	1	0	2	1	7
Cadre 0	**2**	2	2	**7**	7	7	7	**5**	5	5	**2**	2	2	**0**	0	0	0	0	0	**7**
Cadre 1	-	**1**	1	1	1	1	1	1	1	**6**	6	6	6	6	**1**	1	1	1	1	1
Cadre 2	-	-	**0**	0	0	**6**	6	6	**7**	7	7	7	7	7	7	7	7	**2**	2	2
Déf. de page	O	O	O	O	N	O	N	O	O	O	O	N	N	O	N	O	N	O	N	O

Mémoire virtuelle et MMU : Pagination

Influence de l'écriture des programmes

En Java, comme en C, les tableaux bidimensionnels sont rangés en mémoire ligne par ligne. Soit la déclaration suivante :

```
int [][] tab = new int[1024][1024];
```

et les deux extraits de programmes :

```
for (int i = 0; i < 1024; i++)
    for(int j = 0; j < 1024; j++)
        tab[i] [j] = 1;
```

```
for(int j = 0; j < 1024; j++)
    for (int i = 0; i < 1024; i++)
        tab[i] [j] += 2;
```

Rappel : en Java les int sont stockés sur 4 octets.

Premier jet

Les lignes des tableaux étant allouées de façon contiguë, chaque ligne tient dans une page ($4Ko = 1024 \times 4o$) et on peut affirmer que la première version du programme, travaillant ligne par ligne, va accéder successivement au 1024 pages stockant le tableau.

Dans la deuxième version, le programme accède au tableau colonne par colonne et va donc dans la première itération accéder successivement à toutes les 1024 pages, puis à nouveau dans la 2^e itération, ... Si le tableau ne tient pas en mémoire, on va faire 1024×1024 défaut de pages.

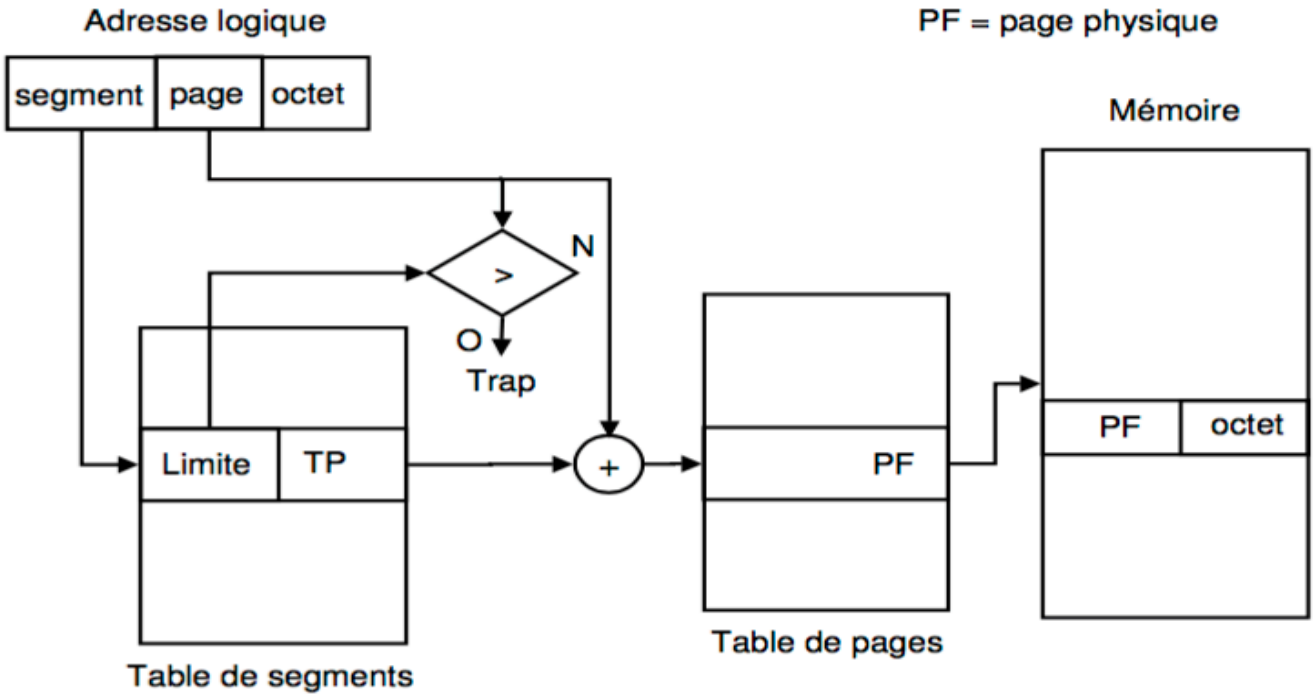
Après réflexion

1. Pas de problème, le tableau tient entier dans la RAM de nos machines.
2. Sinon, notre calcul ne tient pas compte du stockage de la longueur du tableau (en Java les tableaux sont des objets (voir `http://java.sun.com/docs/books/jvms/second_edition/html/Concepts.doc.html#16446`).
3. Mais ce que l'on a raconté tient pour la gestion du cache de nos processeurs...

Segmentation paginée

La segmentation peut être combinée avec la pagination. Chaque segment est composé d'un ensemble de pages. Les adresses générées par les compilateurs et les éditeurs de liens, dans ce cas, sont alors des triplets :

<numéro du segment, numéro de page, déplacement dans la page>



Mémoire virtuelle et MMU : Pagination

Segmentation et ***pagination*** sont 2 mécanismes permettant d'atteindre le même objectif : ***Séparer l'espace d'adressage physique des processus***

La ***segmentation*** assigne un espace linéaire diffèrent à chaque processus

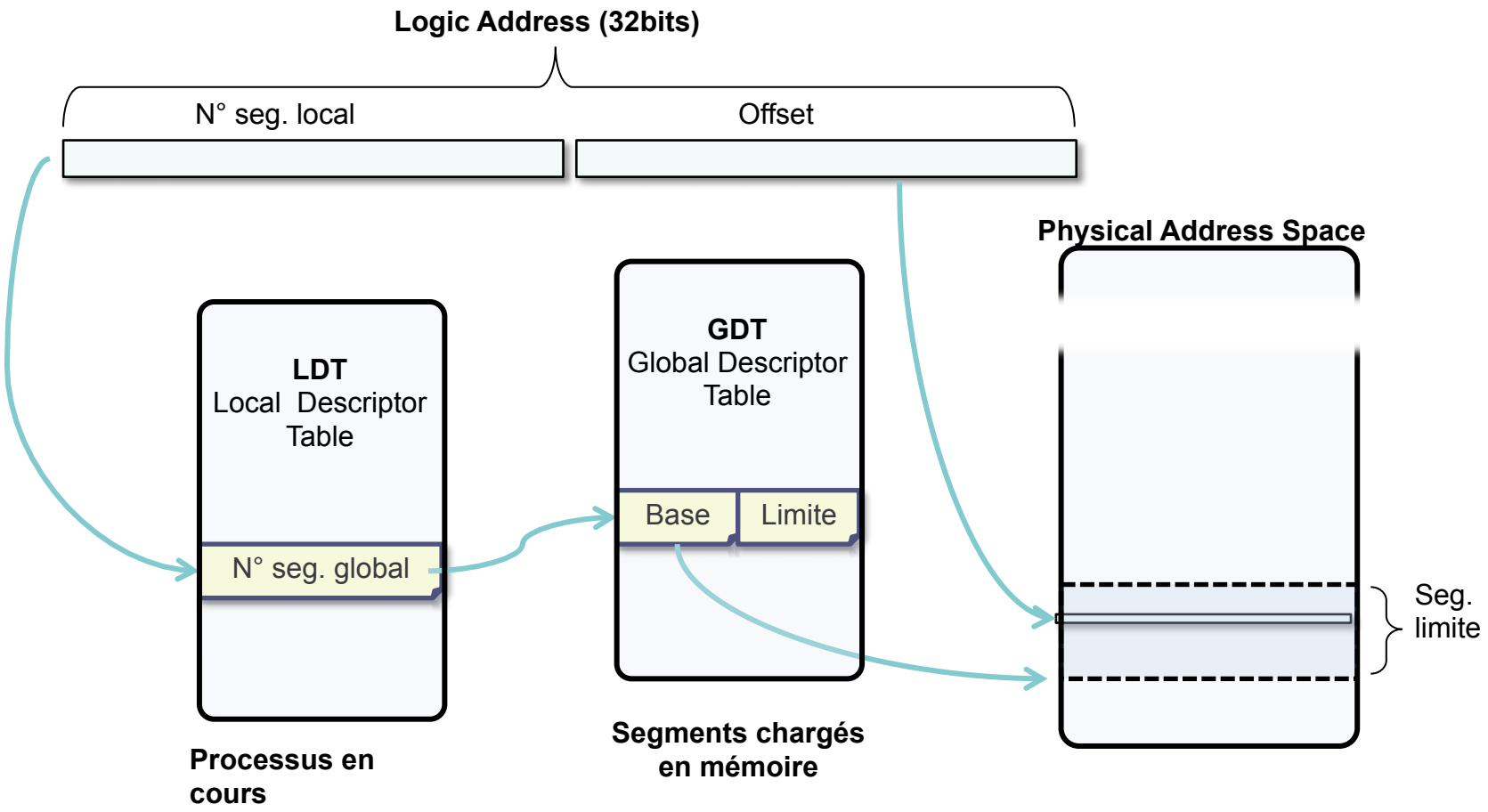
La ***pagination*** permet d'associer un espace linéaire à différents espaces physiques

L'avantage pour le programmeur est qu'il n'a pas à connaître la quantité de mémoire dont dispose l'ordinateur sur laquelle son programme va s'exécuter, il demande seulement une quantité de mémoire.
Un autre avantage est que l'on peut modifier la quantité de mémoire physique de l'ordinateur sans que cela impacte le fonctionnement des programmes.

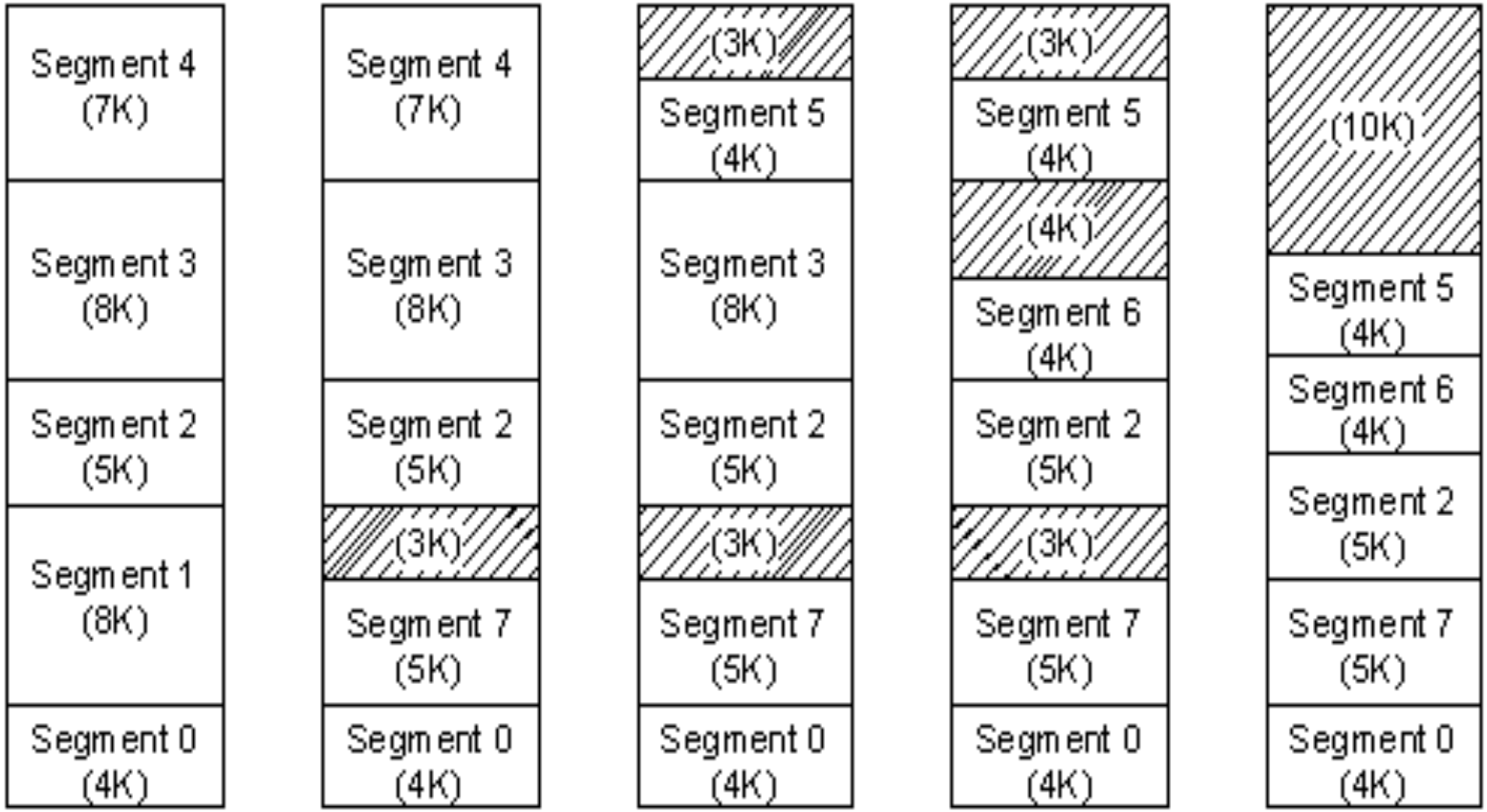
Linux préfère la ***pagination*** :

- La gestion des processus est plus facile sur un unique espace linéaire
- La segmentation n'est pas disponible sur tous les processeurs

Mémoire virtuelle et MMU : Segmentation - Tables locale/globale des segments



Mémoire virtuelle et MMU : Segmentation - Fragmentation



L'inconvénient majeur de la segmentation est la fragmentation externe de la mémoire physique.

Mémoire virtuelle et MMU : Segmentation – Cas de Linux

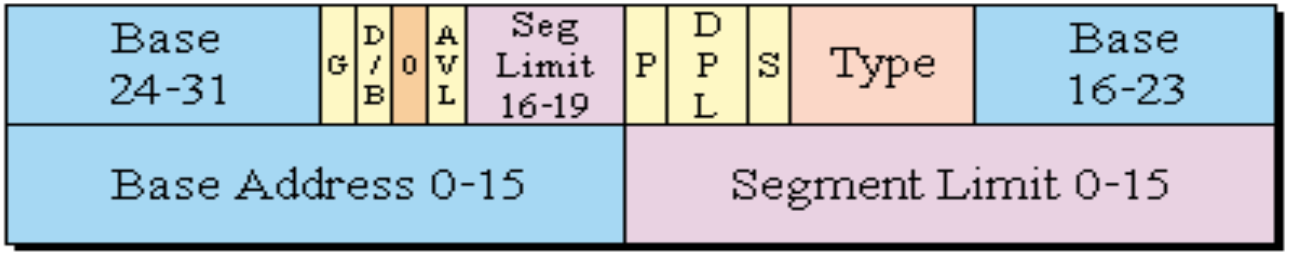
Linux utilise très peu la technique de segmentation :

- Le travail de la MMU est grandement simplifié si les processus partagent le même espace de mémoire linaire
- Linux est multiplateformes, or de nombreuses architectures RISC ont un usage très limités de la segmentation.

Sous Linux, tous les processus évoluant dans l'espace user partagent les mêmes segments de code et de data (Flat memory model). Idem pour les processus noyau évoluant en mode kernel.

Segment	***Base***	***G***	***Limit*** *4Gb size*	***S***	***Type*** *Code/data*	***P*** *in memory*	***DPL*** *privilège*	***D/B*** *32bits code/ data*
User code	*0x00000000*	*1*	*0xFFFFF*	*1*	*10*	*1*	*3*	*1*
User data	*0x00000000*	*1*	*0xFFFFF*	*1*	*2*	*1*	*3*	*1*
Kernel code	*0x00000000*	*1*	*0xFFFFF*	*1*	*10*	*1*	*0*	*1*
Kernel data	*0x00000000*	*1*	*0xFFFFF*	*1*	*2*	*1*	*0*	*1*

Mémoire virtuelle et MMU :
Segmentation – Descripteur de segment (Intel 80x86)





AVL - Available for use by the operating system
BASE - Segment Base Address
D / B - Default Segment Size (16 / 32 bits)
DPL - Descriptor Privilege Level
G - Granularity
LIMIT - Segment Limit
P - Present Bit
S - Descriptor Type (System / Application)
TYPE - Segment Type

Type bits for Data segments

bit 3	Data/Code	0 (data)
bit 2	Expand-down	0 (normal) 1 (expand-down)
bit 1	Writable	0 (read-only) 1 (read-write)
bit 0	Accessed	0 (hasn't been accessed) 1 (has been accessed)

Type bits for Code segments

bit 3	Data/Code	1 (code)
bit 2	Conforming	0 (non-conforming) 1 (conforming)
bit 1	Readable	0 (execute-only) 1 (executable and readable)
bit 0	Accessed	0 (hasn't been accessed) 1 (has been accessed)