Architecture des ordinateurs

2ème partie

Ordinateurs, Microprocesseurs, Mémoires, langage d'assemblage

Notes de cours

Option : Génie Informatique
DUT/BTS

NGAH ASSE
Institut Universitaire de technologie
Université de Douala

Année 2005

Ce cours a bénéficié de l'appui de la coopération française (Projet Comètes)

Table des matières

Introduction générale

Ce cours est une introduction succincte aux concepts de base de l'architecture et de l'organisation des ordinateurs. Elle a pour but de mieux en comprendre le fonctionnement. L'attention est portée sur les structures communes essentielles sans entrer dans les détails des diverses architectures. Ces notions doivent permettre de mieux analyser les performances d'un programme et d'aider à son optimisation.

Chapitre 1 Analyse multicouche de l'ordinateur

1.1 Introduction

Un ordinateur est une machine capable de résoudre des problèmes en appliquant des instructions préalablement définies. La suite des instructions décrivant la façon dont l'ordinateur doit effectuer un certain travail est appelé programme. Les circuits électroniques de chaque ordinateur ne pouvant reconnaître et exécuter qu'un nombre très limité d'instructions, tout programme doit être, avant son exécution, converti pour n'être exprimé qu'avec ces instructions.

L'ensemble des instructions exécutables directement par un ordinateur forme un langage qui permet aux opérateurs de communiquer avec cet ordinateur : c'est le langage machine.

Les langages machine sont si primitifs qu'il est extrêmement pénible et fastidieux de les utiliser. On est souvent amené à construire un nouveau jeu d'instructions plus pratique à utiliser que le langage machine L1 qu'on appellera langage machine L2.

. Il existe deux approches de construire L2 qui diffèrent suivant la manière dont les programmes écrits en L2 sont traités par l'ordinateur qui, en dernière analyse, ne peut exécuter que des programmes écrits dans son langage machine L1.

La première technique est la traduction qui consiste à remplacer chaque instruction du programme écrit en L2 par la suite d'instructions en L1 qui est équivalente. On obtient alors un nouveau programme entièrement écrit en L1 que l'ordinateur exécute.

La seconde façon consiste à écrire un programme en L1 capable, après avoir examiné chaque instruction d'un programme en L2, d'exécuter directement la séquence d'instructions en L1 équivalente : c'est l'interprétation. ici on n'a pas besoin de générer un programme tout un programme en L1. Le programme écrit en L1 qui examine et exécute chaque instruction du programme en L2 s'appelle un interpréteur (ou un interprète).

Traduction et interprétation se ressemblent beaucoup en ce que dans chacune de ces techniques toute instruction en L2 est finalement convertie en une suite équivalente d'instructions en L1. Elles restent tout de même différentes en ce que pour le cas de la traduction tout le programme en L2 est d'abord traduit en un programme en L1 puis le programme en L2 disparaît et c'est le programme en L1 qui est exécuté alors que pour l'interprétation chaque instruction en L2 est analysée puis exécutée directement.

La machine qui exécuterait les programmes directement en L2 est dite machine virtuelle de langage machine L2. Si une telle machine pouvait exister les gens écriraient leur programme directement en L2 qui seraient directement exécutés par la machine.

Pour que l'interprétation ou la traduction reste assez simple il faut que les langages L1 et L2 ne soient pas trop différents. Cela fait que le langage L2 bien que meilleur que L1 n'est pas le langage idéal pour les programmeurs. On peut donc être amené à définir un nouvel ensemble d'instructions plus proches de l'utilisateur final et donc moins dépendantes de la machine que celles de L2 : c'est le langage L3 à l'aide duquel on pourra écrire des programmes comme si une machine basée sur ce langage existait. Ces programmes seront traduits en L2 ou interprétés par un interpréteur écrit en L2.

On peut ainsi concevoir toute une série de langages, chacun étant un peu plus pratique que son prédécesseur, jusqu'à ce qu'on obtienne un jugé convenable. On peut voir un ordinateur alors comme un empilement de *couches* ou de *niveaux*.

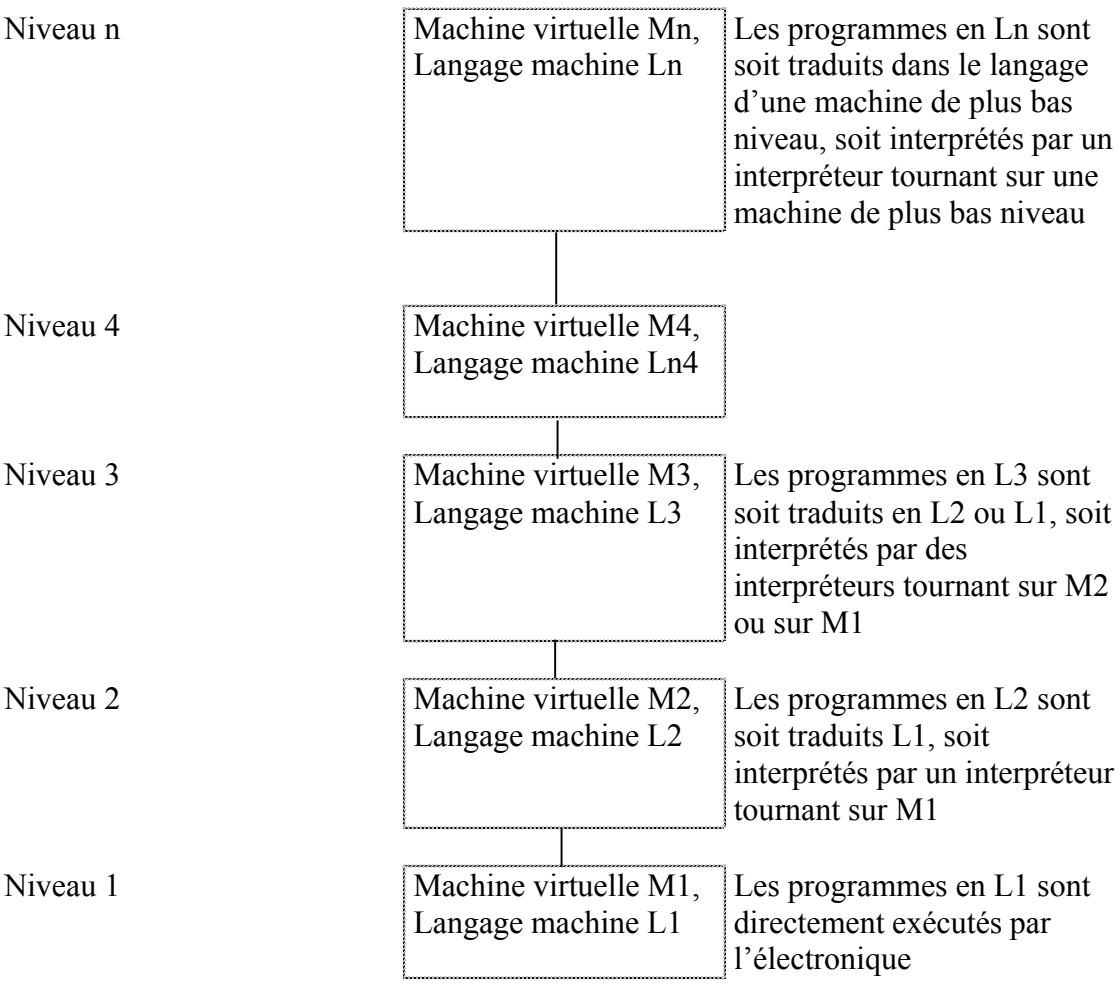


Fig. 1.1 Architecture d'une machine multi-niveaux

1.2 Langage et machines virtuelles

Tout machine a son langage machine formé de l'ensemble des instructions de base qu'elle peut exécuter. On peut donc dire qu'une machine définit un langage. Mais réciproquement un langage définit une machine, la machine qui peut exécuter tous les programmes écrits en ce langage. Cette machine peut être réalisable ou non.

Un ordinateur composé de *n* couches peut être vue comme *n* machines virtuelles distinctes, chaque machine virtuelle ayant son propre langage. Il n'y a que les programmes écrits en L1 qui puissent être véritablement traités par les circuits électroniques sans qu'il soit nécessaire d'envisager pour eux une quelconque traduction ou interprétation. Les programmes écrits en L2, L3, …, Ln, pour leur exécution, doivent être soit interprétés par un interpréteur d'un plus bas niveau soit traduits en langage d'un plus bas niveau.

1.3 Les machines multicouches Actuelles

La plupart des ordinateurs actuels ont au moins deux niveaux mais on trouve couramment des machines à 6 niveaux. Au niveau 0, à la base, on trouve le matériel. Les circuits électroniques exécutent les programmes en langage machine du niveau 1. On peut citer le niveau composant inférieur au niveau 0.

Le niveau 0 est aussi appelé le niveau physique, à ce niveau c'est des composants logiques (portes logiques) qui sont manipulées.

Le niveau suivant est le niveau 1 ou niveau du langage machine. Ce niveau comprend un programme appelé microprogramme dont le travail est d'interpréter les instructions de niveau 2. On appelle encore ce niveau le niveau microprogrammé. La plupart des instructions à ce niveau consistent à transférer des données ou à exécuter des tests très simples.

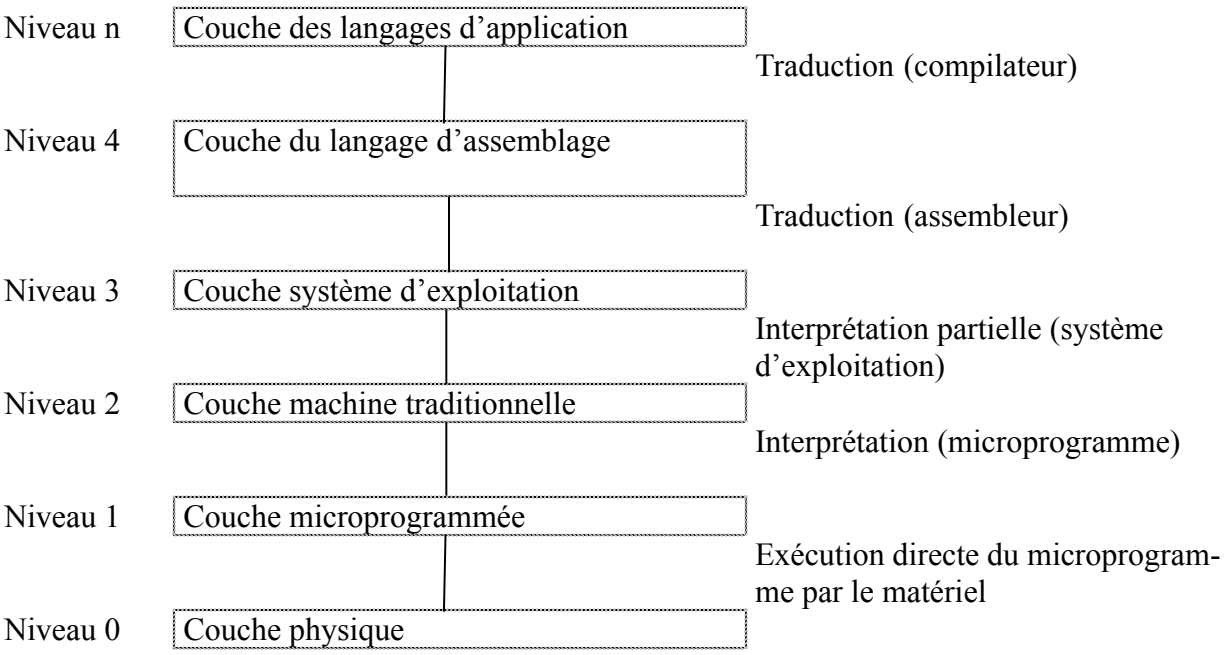


Fig. 1.2 Les six couches de la plupart des ordinateurs actuels.

Une machine de niveau 1 dispose d'au moins un microprogramme. Chaque microprogramme définit implicitement un langage de niveau 2 et une machine virtuelle dont le langage machine est ce langage. Toutes les machines de niveau 2 (niveau machine conventionnel) ont beaucoup de points communs, mêmes si ces machines sont de constructeurs différents.

Les constructeurs d'ordinateurs proposent pour chaque machine un manuel souvent intitulé « manuel de référence du langage machine ». Ces langages se rapportent au niveau 2.

Le jeu d'instructions machine qui y est décrit est l'ensemble des instructions interprétées par le microprogramme.

Le niveau 3 est souvent un niveau hybride. La plupart des instructions de son langage figurent également dans le langage de niveau2. Il existe plus de différences entre les machines de niveau 3 qu'entre les machines de niveau 1 ou 2.

Les services offerts par le niveau 3 sont pris en charge par un interpréteur s'exécutant au niveau 2 appelé *système d'exploitation*. Les instructions niveau 3 qui sont identiques aux instructions niveau 2 sont traitées directement par le microprogramme et non par le système d'exploitation. C'est pour cette raison que ce niveau est dit hybride.

Les niveaux 0, 1, 2 et 3 servent principalement à supporter les traducteurs et interpréteurs dont les niveaux supérieurs ont besoin et sont écrits par des *programmeurs système (spécialistes des machines virtuelles).* Les niveaux 4 et au-dessus concernent le programmeur d'applications qui a un problème concret à résoudre.

Les niveau 4, 5 et au-dessus sont souvent, mais pas toujours, associés à des traducteurs.

Les langages machine des niveaux 1, 2, et 3 sont numériques alors qu'à partir du niveau 4 on trouve des langages qui contiennent des mots beaucoup plus compréhensibles par les êtres humains.

Ce niveau 4, le niveau langage d'assemblage, est une forme symbolique d'un des langages sous-jacents. Ce niveau permet d'écrire des programmes pour les niveaux 1, 2 et 3 dans une forme moins déplaisante. Les programmes en langage d'assemblage sont d'abord traduits en langage de niveau 1, 2 ou 3 puis sont interprétés par la machine virtuelle ou réelle correspondante. Le programme qui réalise la traduction s'appelle un assembleur.

Au niveau 5 on trouve des langages conçus pour êtres utilisés par des programmeurs d'application. Ces langages, appelés langages de haut niveau sont nombreux : BASIC, C, COBOL, FORTRAN, LISP, MODULA-2, PASCAL, PROLOG, …

Les programmes écrits en l'un quelconque de ces langages sont souvent traduits en niveau 3 ou 4 par des traducteurs appelés *compilateurs.*

Les niveaux 6 et au-dessus sont des ensembles de programmes conçus pour créer des machines destinées à traiter des applications déterminées (gestion, éducation, informatique, ...)

L'ensemble des types de données, des opérations et des caractéristiques de chaque niveau s'appelle l'architecture de ce niveau.

1.4 Histoire des machines multicouches

Dans les années 40 les ordinateurs n'avaient que 2 niveaux : le niveau machine conventionnel, dans lequel on faisait les programmes, et le niveau physique qui exécutait ces programmes.

1951 : En UK, M. V. Wilkes eut l'idée d'une machine à 3 niveaux qui devait disposer d'un interpréteur interne destiné à exécuter les programmes du niveau machine conventionnel, le matériel devant exécuter des programmes dont le répertoire d'instructions était très limité. Ce qui devait permettre de réduire la complexité du matériel(époque des tubes à vide), le coût et d'augmenter la fiabilité. Il y eut la construction de quelques machines à 3 niveaux dans les années 50, 60.

Les assembleurs et les compilateurs furent créés dans les années 50 dans le but de faciliter la tâche des programmeurs

Années 60 : naissance des systèmes d'exploitation pour diminuer les pertes de temps exemple : FMS (Fortran Monitor System) tournait sur IBM 709.

Le traitement par lots indiquait l'organisation du travail qui découlait de la lecture des paquets de cartes et de l'écriture sur les imprimantes (travail qui demandait plusieurs heures). Ce sont les systèmes d'exploitation en temps partagés (suites des travaux des chercheurs du Dartmouth college, du MIT, etc.) qui ont permis aux programmeurs de communiquer directement avec leur machines au début des années 60(connexion des terminaux lourds via lignes de téléphone à l'ordinateur ; on pouvait taper le programme et obtenir immédiatement les résultats)

1.5 Matériel et logiciel

Les programmes écrits dans les langages de l'ordinateur (niveau 1) peuvent être directement exécutés par l'électronique (niveau 0) sans interprétation ni traduction. Cette électronique à laquelle on ajoute la mémoire et les unités d'entrée/sortie forme ce qu'on appelle le matériel. Le matériel c'est tout ce qui est tangible (circuits intégrés, cartes, câbles alimentation, mémoires, imprimantes, terminaux)

Par opposition le logiciel est intangible (abstrait) et est formé d'algorithmes et de leur représentation informatique (les programmes).

Toute opération effectuée par le logiciel peut l'être directement par le matériel et toute instruction exécutée par le matériel peu être simulé par le logiciel : matériel et logiciel sont donc logiquement équivalents. Seuls les paramètres tels que le coût de réalisation, la vitesse d'exécution nécessaire, la fiabilité requise ou la fréquence des changements attendue peuvent justifier le choix d'exécution par le matériel ou par le logiciel.

1.6 Les grandes étapes de l'architecture des ordinateurs

1.6.1 La génération zéro : les calculateurs mécaniques (1642-1945)

1642 : invention de la première machine mécanique par Blaise PASCAL (à 19 ans), opérations effectuées : addition et soustraction.

1672 : Leibniz adjoint la multiplication et la division à la machine de PASCAL

1822 Charles Babbage, professeur de mathématique (Université de Cambridge), construit la machine à différence (algorithmes des différences finies utilisant les polynômes), comme celle de PASCAL cette machine effectuait les opérations d'addition et de soustraction et en plus comme innovation, elle restituait les résultats en gravant un plateau de cuivre à l'aide d'un timbre en acier. .
Quelques années plus tard il construit la machine analytique (entièrement mécanique) qui comportait quatre parties : le magasin (mémoire de 1000 mots de 50 chiffres décimaux)), le moulin(unité de calcul), l'entrée (lecteur de cartes perforées) et la sortie(perforation ou impression). La machine analytique n'était pas réservée à une application particulière, elle lisait les instructions sur les cartes perforées et les traitait.
1944 : Howard Aikein construit le Mark I(72 mots de 23 chiffres décimaux, temps de cycle 6 secondes)

1.6.2 La première génération: les tubes à vide (1945-1955)

1943 : le gouvernement anglais construit COLOSSUS (premier calculateur électronique numérique) avec l'aide de Alan Turing, mathématicien anglais.
1946 : Construction par Mauchley et J. Presper Eckert (université de pennsylvanie) de l'ENIAC ou Electronic numerical Integrator And Computer (18000 tubes à vide, 1500 relais, poids 30 tonnes, puissance électrique 140 Kw, 20 registres de 10 chiffres décimaux, programmation par la manipulation de quelques 6000 commutateurs)
1949 : construction de l'EDSAC par Maurice Wilkes(Univerté de Cambridge en UK)
1952 : construction par Von Neumann et Herman Goldsteine(Institute of Advanced Study de Princeton) de l'IAS, premier ordinateur à programme enregistré ,4096 mots de
La machine de Von Neumann comportait 5 parties : la mémoire, l'unité arithmétique et logique, l'unité de contrôle et les dispositifs d'entées et de sorties.
Construction du Whirlwind I (mots de 16 bits)par les chercheurs du MIT qui fut le premier mini-ordinateur commercialisé.
1953 : IBM 701(2K mots de 36 bits, 2 instructions par mot)
1956 : IBM 704 (4K mots de 36 bits, dispositifs spécialisés de calcul flottant)
Gamma ET premier ordinateur à programme enregistré de Bull(modèle Von Neuman).

1.6.3 La deuxième génération: les transitors (1955-1965)

1948 : invention du transistor par John Bardeen, Walter brattain et William Shockley
construction du TX-0
1961 : construction du PDP-1 (4K mots de 18 bits, temps de cycle 5 microsecondes) par la société DEC
quelques années plus tard PDP-8(mots de 12 bits, bus unique)
1964 : 6600(architecture parallèle, 10 fois plus rapide que IBM 7094) de la société CDC.

1.6.4 La troisième génération: les circuits intégrés (1965-1980)

L'invention du circuit intégré a permis de placer des dizaines de transistors sur une seule puce de silicium, ce qui a conduit à des ordinateurs plus petits, plus rapides et moins chers. Nous allons voir les plus caractéristiques d'entre eux.
En 1964, IBM le numéro 1 en informatique, invente le System/360 modèles 30 et 75 comportant des circuits intégrés et destinés tout aussi bien aux applications scientifiques qu'à la gestion. Le System/360 avait de nombreuses innovations, dont l'une des plus importantes était la notion de gamme d'une demi-douzaine de machines de tailles et de puissances différentes, mais dotées du même langage d'assemblage. L'autre grande innovation était la *multiprogrammation* qui permet à plusieurs programmes de résider simultanément en mémoire, de sorte que lorsqu'un programme entre dans une phase d'attente d'entrée-sortie, l'UC passe à l'exécution d'une partie d'un autre programme. Le 360 fut le premier ordinateur capable d'émuler (de simuler) un autre ordinateur. Pour sortir du dilemme binaire-parallèle *vs* décimal-série, le 360 utilisait un compromis: il avait 16 registres de 32 bits pour l'arithmétique binaire.

Le 360 avait une taille énorme (pour l'époque) de l'espace d'adressage: adresses de 24 bits (16 Mo). Il reste vrai qu'à cette époque, 16 Mo avaient un goût d'infini.
La série 360 fut suivie de la série 370 puis des séries 4300, 3080 et
3090 qui toutes avaient la même architecture. Au milieu des années quatre-vingt, cette limite de 16 Mo commença à poser de tels problèmes qu'IBM dut abandonner partiellement la compatibilité en passant à des adresses de 32 bits, ce qui permet d'avoir des espaces mémoires de 4Go.
La troisième génération fut aussi l'occasion pour l'industrie mini-informatique de progresser, notamment avec l'introduction par DEC du PDP-11, successeur du PDP-8. Par certains côtés, le PDP-11 était le petit frère du 360, tout comme le PDP-1 avait été celui de la 7094. Le PDP-11 et le 360 avaient des registres orientés mot, une mémoire orientée octet et un rapport coût/performance très intéressant.

1.6.5 La quatrième génération: les ordinateurs personnels et VLSI (1980-XXXX)

Les *VLSI (Very Large Scale Integration)* permettent d'intégrer des dizaines de milliers, puis ensuite des millions de transistors sur une puce et donc d'avoir des ordinateurs toujours plus petits et plus rapides. Avant l'arrivée du PDP-1, les ordinateurs étaient si gros et si chers, que les entreprises, les administrations et les universités avaient créé, pour les accueillir, des départements spéciaux qu'on appelait souvent centre de calcul ou centre informatique. La mini-inforrnatique permit à des services d'acheter leur propre ordinateur. L'ère de l'informatique personnelle commence avec la chute des pris vers les années 80, il était désormais possible à une personne physique d'avoir un ordinateur.
Les ordinateurs personnels étaient dédiés à faire du traitement de texte, du calcul financier à l'aide de tableurs, et d'autres applications hautement interactives pour lesquelles les gros ordinateurs sont mal adaptés.

		Modèles		
30		40	50	65
Caractéristiques				
Performance relative	1	3.5	10	21
Temps de cycle (ns)	1000	625	500	250
Mémoire maximum (K)	64	256	256	512
Nb octets chargés ~ cycle	1	2	4	16
Nb max. canaux de données	3	3	4	6

Figure 1-3. L'offre initiale d'IBM en matière de Système 360

Depuis les années 80, comme le montre la figure 1-4, on pouvait ranger les ordinateurs en cinq catégories, en fonction de leur taille physique, de leurs performances ou de leurs domaines d'application.

Type MIPS		Mo	Exemple	Type d'utilisation
Ordinateur personnel	1	1	IBM PS/2	Traitement de textes
Miniordinateur	2	4	PDP – 11/84	Temps réel
Supermini	10	32	SUN - 4	Serveur de fichiers
Gros ordinateur	30	128	IBM 3090/300	Banque
Superordinateur 125		1024	Cray-2 Calculs	météo

Figure 1-4. Cinq types courants d'ordinateurs

Les gros ordinateurs sont les descendants de l'IBM 360 et du CDC 6600. La différence entre un gros ordinateur et un super-mini réside essentiellement dans les possibilités d'entrées-

sorties et les applications pour lesquelles on les utilise. Un super-mini a souvent un ou deux disques de plusieurs dizaines de Giga-octet. Un gros ordinateur peut en avoir une centaine. Les super-minis sont, en général, utilisés dans le cadre d'applications interactives, alors que les gros ordinateurs sont utilisés pour de gros travaux en traitement par lots ou pour des applications transactionnelles (banque, réservation de places d'avions, ...) dans lesquelles on accède à de très grosses bases de données. Viennent enfin les super-ordinateurs, spécialement conçus pour maximiser le nombre de *FLOPS (FLoating point OPerations per Second:* opérations en virgule flottante par seconde). N'avait droit au titre de super-ordinateur que des machines qui dépassaient le gigaflops. Pour atteindre ces performances, il est nécessaire d'utiliser une architecture spécifique, à base de parallélisme, ce qui ne convient qu'à une petite classe d'applications. Pendant des années on a associé le nom de Seymour Cray aux superordinateurs. Seymour Cray a, en effet, conçu le CDC 6600 et son successeur, le 7600. Puis il a fondé sa propre entreprise, Cray Research, pour construire le Cray-l et le Cray-2. En 1989, il a quitté Cray Research pour fonder une autre entreprise avec l'espoir de construire un Cray-3.

Conclusion

Nous d'achever l'analyse multicouche des ordinateurs et une brève connaissance de l'histoire de ces machines

Nous allons dans les chapitre suivant nous intéresser aux couches les plus basses de cette architecture couches physique, microprogramée et machine conventionnelle.

Pour ce faire le cours s'appuie sur le modèle en usage dans l'immense majorité des architectures aujourd'hui: le modèle de VON NEUMANN.

CHAPITRE 2: INTRODUCTION AUX MICROPROCESSEURS

2.1 Définitions

Un microprocesseur est circuit intégré à large échelle (LSI) programmable qui fonctionne comme l'unité de traitement d'un ordinateur digital.
Un circuit intégré est un composant électronique minuscule constitué de transistors interconnectés et pouvant exécuter des fonctions câblées ou programmées.
Un micro-ordinateur est un ensemble constitué d'un microprocesseur (CPU), d'une mémoire principale (MP), des circuits de communication (bus et interface) des unités d'entrée/sortie ou unité d'échange.

2.2 Fonctionnement du micro-ordinateur : modèle Von Neumann

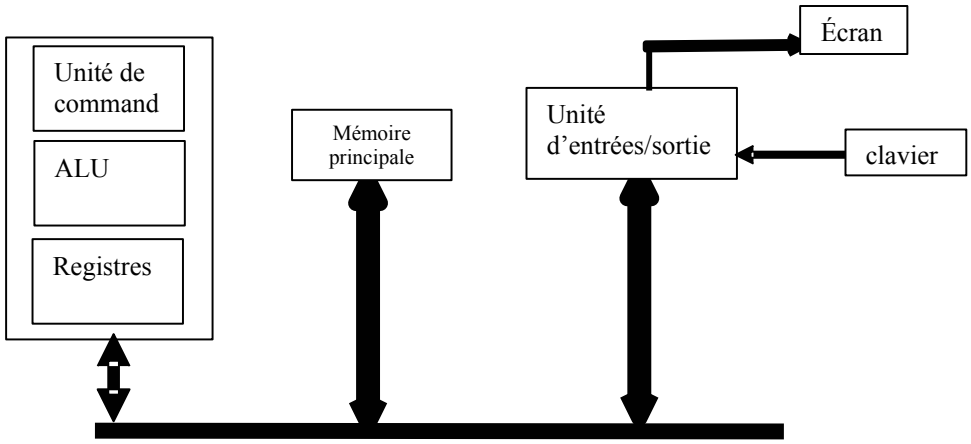


Fig. 1 Machine de Von Neumann

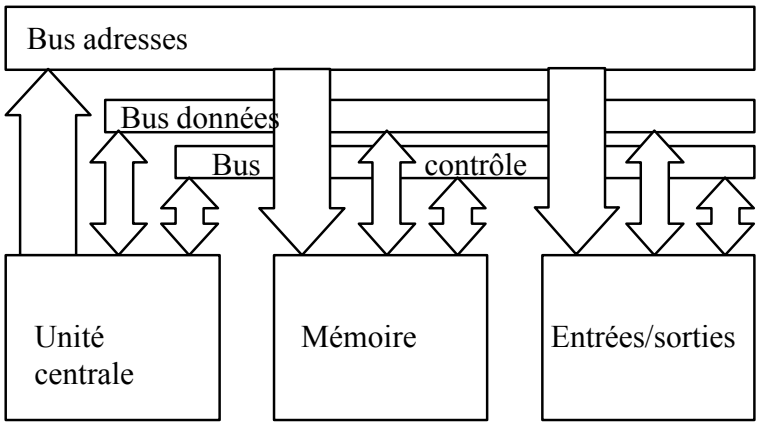


Fig.2 Bus détaillé

2.2.1 Rôles des composantes :

1° Unité centrale (UC) : contrôler toutes les composantes, en exécutant les instructions d'un programme ; effectuer des calculs arithmétiques et logiques ;
2° Mémoire (M) : garder le programme en cours d'exécution et les données associées ;
3° Entrées/sorties (E/S) : relier le système au monde externe à travers les unités périphériques (écran, clavier, disques, bandes magnétiques, réseaux, etc.) ;
4° Bus d'adresse (A) : véhiculer l'adresse mémoire ou d'unité E/S engendrée par l'UC (et dans certains cas par d'autres unités de contrôle) ;

5° Bus de données (D) : véhiculer l'information (instructions, données) entre l'UC, la mémoire et les unités E/S ;
6° Bus de contrôle (C) : véhiculer les signaux employés par l'UC pour le contrôle du système (Adresse mémoire valide, Adresse E/S valide, Lecture/écriture, Attente, Interruption, etc.).

2.2.2 Exécution d'une instruction

Selon le modèle de Von Neumann les données et les programmes (suite d'instructions) sont tous stockés en mémoire principale.
La machine doit donc avant l'exécution d'une instruction chercher celle-ci en mémoire principale.
L'exécution d'une instruction comprend trois phases : la recherche, le décodage, l'exécution.
Les étapes de l'exécution d'une instruction :
1°) Chargement de la prochaine instruction à exécuter depuis la mémoire principale jusque dans le registre d'instruction (RI) compris dans le groupe des registres de l'unité centrale ;
2°) modification du compteur de programme ou compteur ordinal pour qu'il pointe sur l'instruction suivante à exécuter ;
3°) décodage de l'instruction que l'on vient de charger par l'unité centrale ;
4°) localisation dans la mémoire des éventuelles données utilisées par l'instruction ;
5°) Chargement de données, si nécessaire, dans les registres internes de l'unité centrale ;
6°) exécution proprement dite de l'instruction ;
7°) stockage des résultats à leurs destination respectives ;
8°) retour à l'étape 1°) pour l'exécution de l'instruction suivante.

Cette suite d'étapes appelée cycle d'exécution ou de chargement-décodage-exécution est à la base de fonctionnement de tout ordinateur.
Les étapes de 1 à 2 constituent la phase de recherche, les étapes de 3 à 5 constituent la phase de décodage, 6,7 et 8 entrent dans la phase exécution.
L'ensemble des instructions dont dispose le programmeur à un niveau donné s'appelle *jeu d'instructions*

En d'autre termes l'exécution peut se résumer comme suit :

1° l'UC lit le code de l'instruction en mémoire : l'UC sort l'adresse de l'instruction sur le bus d'adresses et active les signaux "Adresse mémoire valide" et "Lecture" ; la mémoire sort sur le bus de données l'information se trouvant à l'adresse indiquée ; l'UC transfère cette information (code d'une instruction) du bus de données vers un registre interne (registre d'instruction) ;
2° l'UC décode le code de l'instruction, qui indique la suite d'opérations à effectuer ;
3° si la lecture d'une donnée se trouvant en mémoire est nécessaire :
l'UC calcule l'adresse de la donnée en mémoire, sort l'adresse sur le bus d'adresses et active les signaux "Adresse mémoire valide" et "Lecture" ; la mémoire sort sur le bus de données l'information se trouvant à l'adresse indiquée ; l'UC transfère ce code du bus de données vers un registre interne ;
si l'écriture d'une donnée en mémoire est nécessaire :
l'UC calcule l'adresse de la donnée en mémoire, sort l'adresse sur le bus d'adresses, transfère la donnée d'un registre interne vers le bus de données et active les signaux "Adresse mémoire valide" et "Ecriture" ; la mémoire inscrit à l'adresse indiquée l'information se trouvant sur le bus de données ;
si une opération arithmétique ou logique doit être effectuée :
l'UC récupère le(s) opérande(s) (en passant éventuellement par les étapes de lecture de données se trouvant en mémoire) et l(es) envoie à l'UAL (interne à l'UC) ; le résultat

de l'opération est stocké dans un registre interne ou en mémoire (passage par les étapes d'écriture d'une donnée en mémoire) ;

4° l'UC calcule l'adresse de l'instruction suivante.

2.2.3 Exécution des programmes

Les instructions d'un programme sont exécutées successivement, en suivant les branchements conditionnels ou inconditionnels et les appels de procédures.

Les programmes qui composent le système d'exploitation assurent la gestion des ressources (processeur, mémoire, E/S) et font la liaison entre les programmes d'application.

2.2.4 Architecture et performances

2.2.4.1 Temps d'exécution

Durée d'exécution d'un programme (heure début - heure fin) = temps passé par l'UC pour exécuter effectivement le programme (temps UC utilisateur) + temps passé pour exécuter des programmes du système d'exploitation + attente des résultats d'opérations d'E/S + temps passé pour exécuter d'autres programmes (fonctionnement "temps partagé", *time-sharing*).

Périodes d'horloge par instruction (*clock Cycles Per Instruction, CPI*) :

$$CPI = \frac{\text{temps UC utilisateur (en périodes d'horloge)}}{\text{nombre d'instructions du programme}}$$

.

Temps UC utilisateur = (période horloge) × (CPI) × (nombre instructions du programme).

Réduction du temps UC utilisateur :

1° augmentation de la fréquence de l'horloge (amélioration de la technologie, simplification des circuits) ;

2° réduction du *CPI* (structure UC mieux adaptée aux instructions, accès mémoire plus rapides) ;

3° réduction du nombre d'instructions (instructions plus puissantes, compilateurs optimisants).

2.2.4.2 Amélioration des accès mémoire

Hypothèses de proximité :

1° *Proximité temporelle* : un objet déjà référencé le sera à nouveau bientôt.

2° *Proximité spatiale* : les projets proches d'un objet référencé seront bientôt référencés.

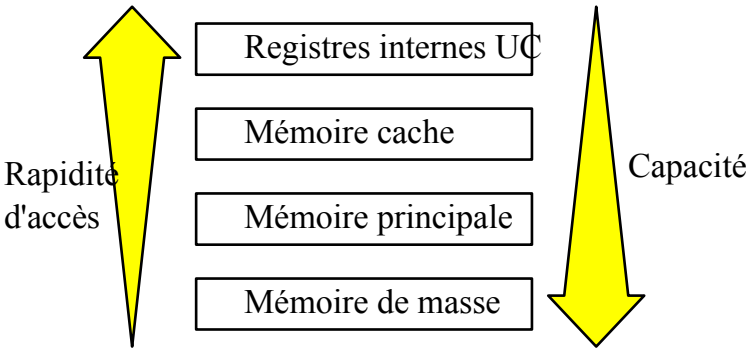


Fig.3 Hiérarchisation des mémoires

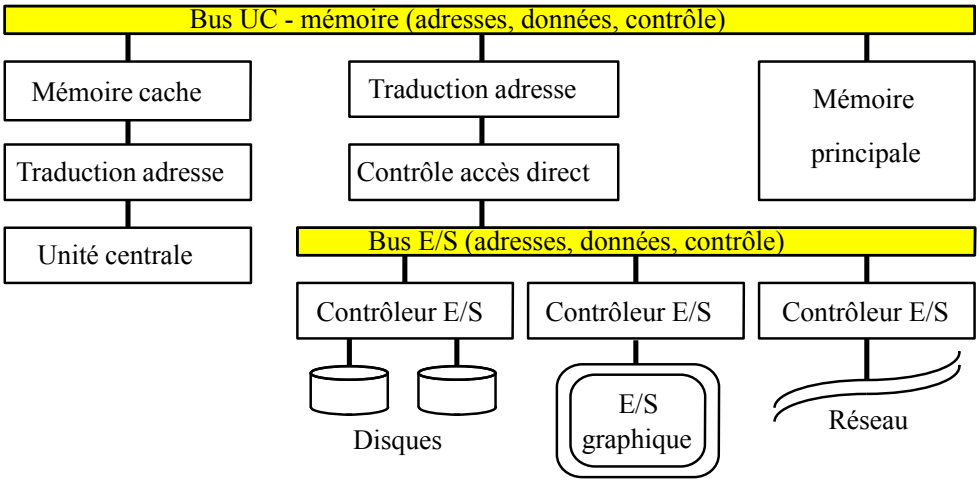


Fig 4. Structure générale d'un micro-ordinateur

2.2.4.3 Modèles d'exécution et réduction du temps d'exécution

Considérons l'addition A+B=C, avec A, B, C en mémoire. Nous avons plusieurs possibilités, parmi lesquelles :

1° A + B = C (l'instruction d'addition agit directement sur des opérandes en mémoire et retourne le résultat en mémoire)

2° A = (R) (une première instruction ramène A dans un registre UC)
(R) + B = C (l'instruction d'addition agit sur un opérande en registre et un autre en mémoire et retourne le résultat en mémoire)

3° A + B = (R) (l'instruction d'addition agit directement sur des opérandes en mémoire et retourne le résultat dans un registre UC)
(R) = C (une deuxième instruction envoie le résultat en mémoire)

4° A = (R1) (une première instruction ramène A dans un registre UC)
(R1) + B = (R2) (l'instruction d'addition agit sur un opérande en registre et un autre en mémoire et retourne le résultat dans un autre registre UC)
(R2) = C (une troisième instruction envoie le résultat en mémoire)

5° A = (R1) (une première instruction ramène A dans un registre UC)
(R1) + B = (R1) (l'instruction d'addition agit sur un opérande en registre UC et un autre en mémoire et retourne le résultat dans le même registre UC)
(R1) = C (une troisième instruction envoie le résultat en mémoire)

6° A = (R1) (une première instruction ramène A dans un registre UC)
B = (R2) (une deuxième instruction ramène B dans un autre registre UC)
(R1) + (R2) = (R3) (l'instruction d'addition agit sur des opérandes en registres UC et retourne le résultat dans un registre UC)
(R3) = C (une quatrième instruction envoie le résultat en mémoire)

Notons chaque type d'instruction d'addition par (*m*,*n*) : au maximum *m* objets en mémoire, *n* objets au total. La solution 1° correspond alors à (3,3), 2° et 3° à (2,3), 4° à (1,3), 5° à (1,2) et 6° à (0,3). Chaque UC est conçue pour qu'elle puisse employer certaines de ces possibilités. Voici la classification correspondante de quelques UC existantes :

(0, 2)	IBM RT-PC
(0, 3)	SPARC, MIPS, HP Precision Architecture
(1, 2)	PDP 10, Motorola 68000, IBM 360
(1, 3)	IBM 360 (instructions RS)
(2, 2)	PDP 11, NS 32x32, IBM 360 (instructions SS)
toutes VAX	

2.3 Structure d'un microprocesseur

2.3.1 Description

Un microprocesseur est un circuit intégré à grande échelle qui exécute les fonctions d'unité centrale d'ordinateur.
Il comporte les éléments suivants :

- une unité arithmétique et logique (ALU = Arithmétic and Logic Unit) qui réalise des opérations élémentaires (addition, soustraction, multiplication, comparaison, décalage, etc.)
- Un jeu de registres c'est-à-dire un ensemble de bascules dont toutes les entrées et les sorties sont accessibles et dans lesquels il est possible d'inscrire des informations comme un mot binaire. Pour un microprocesseur élémentaire on peut distinguer un accumulateur, un registre d'index, un pointeur de pile.
- Une unité de contrôle ou de commande qui contrôle et coordonne le traitement des données par les différentes unités de la machine, conformément au programme.

Les microprocesseurs comportent en général un jeu de registres internes plus important que celui-ci.
Un pointeur de pile, dispositif permettant de retrouver les données qui sont empilées dans les différents registres.
Un accumulateur est un registre temporaire dans lequel on emmagasine les données ainsi que les résultats des opérations arithmétiques et logiques.
On appelle registre d'index une liste des éléments contenus dans un registre avec les références qui permettent de localiser ces éléments.
On appelle mot la plus petite quantité d'information adressable.

L'unité de commande

L'unité de commande est l'ensemble des dispositifs coordonnant le fonctionnement de l'ordinateur afin de lui faire exécuter la suite d'opérations spécifiées dans les instructions du programme.
Elle comprend principalement :

- le compteur ordinal qui est un registre contenant l'adresse en mémoire de l'instruction à exécuter ;
- Le registre d'instruction reçoit l'instruction à exécuter ;
- Le décodeur qui détermine quelle opération doit être exécutée ;
- Le séquenceur qui génère les signaux de commande ;
- L'horloge qui émet des impulsions électroniques régulière, synchronisant ainsi toutes les actions de l'unité centrale

Le séquenceur

Le séquenceur est un automate générant les signaux de commande nécessaires à l'action et au contrôle des unités participant à l'exécution d'une instruction donnée. Ces signaux sont distribués aux différents points de commande des organes concernés selon un chronogramme tenant compte des temps de réponse des circuits sollicités.
Il peut être câblé ou microprogrammé.

Synchronisation des opérations

Les signaux périodiques générés par l'horloge définissent le cycle de base ou cycle machine (clock cycle), durée élémentaire régissant le fonctionnement de la machine

Registres de l'unité centrale

Le nombre et le type des registres implantés dans une unité centrale font partie de son architecture et ont une influence importante sur la programmation et les performances de la machine. Nous voudrions ici passer en revue les registres fondamentaux, que l'on retrouve sur toutes les machines ou presque.

Compteur ordinal (CO) : Ce registre (Program Counter : PC) contient l'adresse de la prochaine instruction à exécuter. Après chaque utilisation il est automatiquement incrémenté du nombre de mots correspondant à la longueur de l'instruction traitée : le programme est exécuté en séquence.
En cas de rupture de séquence (branchement conditionnel ou non, appel à une routine, etc.) il est chargé avec la nouvelle adresse. Le compteur ordinal, dont la taille dépend de l'espace adressable, n'est généralement pas accessible directement au programmeur.
Registre instruction (RI) : C'est le registre de destination dans lequel le CPU transfert l'instruction suivante à partir de la mémoire. Sa taille dépend du format des instructions machines. Le décodeur utilise le registre instruction pour identifier l'action (ou le microprogramme) à entreprendre ainsi que les adresses des opérandes, de destination ou de saut. Le programmeur n'a pas accès au registre instruction.
Accumulateur (ACC) : L'accumulateur est un registre de l'unité arithmétique et logique. Il a de nombreuses fonctions. Il peut contenir un des deux opérandes avant l'exécution et recevoir le résultat après. Cela permet d'enchaîner des opérations. Il peut servir de registre tampon pour les opérations d'entrées/sorties : dans certaines machines c'est le seul registre par lequel on peut échanger des données directement avec la mémoire. Sa taille est égale à la longueur des mots en mémoire. Il possède souvent une extension (Q), pour les multiplications, décalages, divisions, etc. Le registre ACC est accessible au programmeur et très sollicité. Certaines machines possèdent plusieurs accumulateurs.
Registres généraux ou banalisés : Ils permettent de limiter les accès à la mémoire, ce qui accélère l'exécution d'un programme. Ils peuvent conserver des informations utilisées fréquemment, des résultats intermédiaires, etc. Ils sont accessibles au programmeur.
Registres d'indice ou d'index : (XR) Ils peuvent être utilisés comme des registres généraux mais ils ont une fonction spéciale utilisée pour l'adressage indexé. Dans ce cas l'adresse effective d'un opérande est obtenue en ajoutant le contenu du registre d'index à l'adresse contenue dans l'instruction. Ce type d'adressage et de registre est très utile pour manipuler des tableaux. Le programmeur dispose alors d'instructions permettant l'incrémentation ou la décrémentation du registre d'index. En particulier les registres d'index peuvent être incrémentés ou décrémentés automatiquement après chaque utilisation. Dans certaines machines ces instructions sont applicables à tous les registres généraux, il n'y a alors pas de registre d'index spécifique.
Registre de base : A de très rares exceptions à l'intérieur d'un programme on ne fait référence qu'à des adresses relatives ou virtuelles. Par contre l'unité centrale a besoin de connaître les adresses physiques où se situent réellement instructions et données. Celles-ci dépendent de l'endroit où a été chargé le programme en mémoire, l'espace physique occupé par un programme pouvant ne pas être contigu. Le rôle des registres de base est de permettre le calcul des adresses effectives.
Un registre de base contient une adresse de référence, par exemple l'adresse physique correspondant à l'adresse virtuelle 0. L'adresse physique est obtenue en ajoutant au champ adresse de l'instruction le contenu du registre de base. Le registre de base est encore utilisé quand le nombre de bits du champ adresse ne permet pas d'accéder à toute la mémoire.
Registre d'état (Program Status Word : PSW) : Une partie des bits de ce registre, aussi appelé registre condition, constitue des drapeaux (flags) qui indiquent certains états particuliers. Par exemple à la fin de chaque opération on peut y trouver le signe du résultat (Négatif, Zéro ou Positif), ainsi qu'une éventuelle retenue (Carry) ou un dépassement de capacité (Overflow). Ces bits indicateurs peuvent être testés pour déterminer la suite du déroulement du programme : branchements conditionnels. On trouve également le mode de fonctionnement de l'unité centrale.
Deux modes sont possibles le mode utilisateur et le mode système ou superviseur. Dans le mode utilisateur certaines instructions sont interdites : elles provoquent un déroutement vers

le système d'exploitation. Un bit peut également indiquer un déroulement pas à pas : demande de trace (T).
Le registre peut aussi contenir le niveau de l'interruption en cours de traitement ou un masque des niveaux d'interruptions autorisés.
Registre pointeur de pile (PP) : Une pile est une zone mémoire dans laquelle les informations sont rangées de façon contiguë. Le pointeur de pile (Stack Pointer : SP) indique le sommet de la pile : la position de la dernière information enregistrée. Dans certaines machines le pointeur de pile indique la position où sera mémorisée la prochaine donnée. Le fonctionnement d'une pile est du type Dernier Entré Premier Sorti (LIFO : Last In First Out). Les deux principales opérations liées à la pile concernent l'ajout d'un élément dans la pile ou le retrait, souvent nommées respectivement PUSH et PULL. Lorsqu'une donnée est enregistrée dans la pile elle est placée à l'adresse qui suit celle du dernier mot stocké. Après l'opération le pointeur de pile est incrémenté.
Lorsque un mot est retiré de la pile il correspond à la dernière information qui y a été entrée. Après l'opération le pointeur est décrémenté. Une pile est réservée à l'usage de l'unité centrale, en particulier pour sauvegarder les registres et l'adresse de retour en cas d'interruption ou lors de l'appel d'une procédure. Le pointeur de pile est accessible au programmeur, ce qui est souvent source d'erreur. Certaines machines sont dotées de plusieurs pointeurs de piles.
Pour améliorer les performances d'un processeur il faut disposer du plus grand nombre de registres possibles. On réduit ainsi les accès à la mémoire. De plus, il est préférable d'éviter de les spécialiser. On évite ainsi des transferts entre registres, par exemple pour calculer la valeur d'un indice et utiliser ensuite cet indice pour modifier une case d'un tableau.

2.3.2 Structure interne d'un microprocesseur élémentaire

On peut représenter un microprocesseur par le diagramme simplifié suivant :

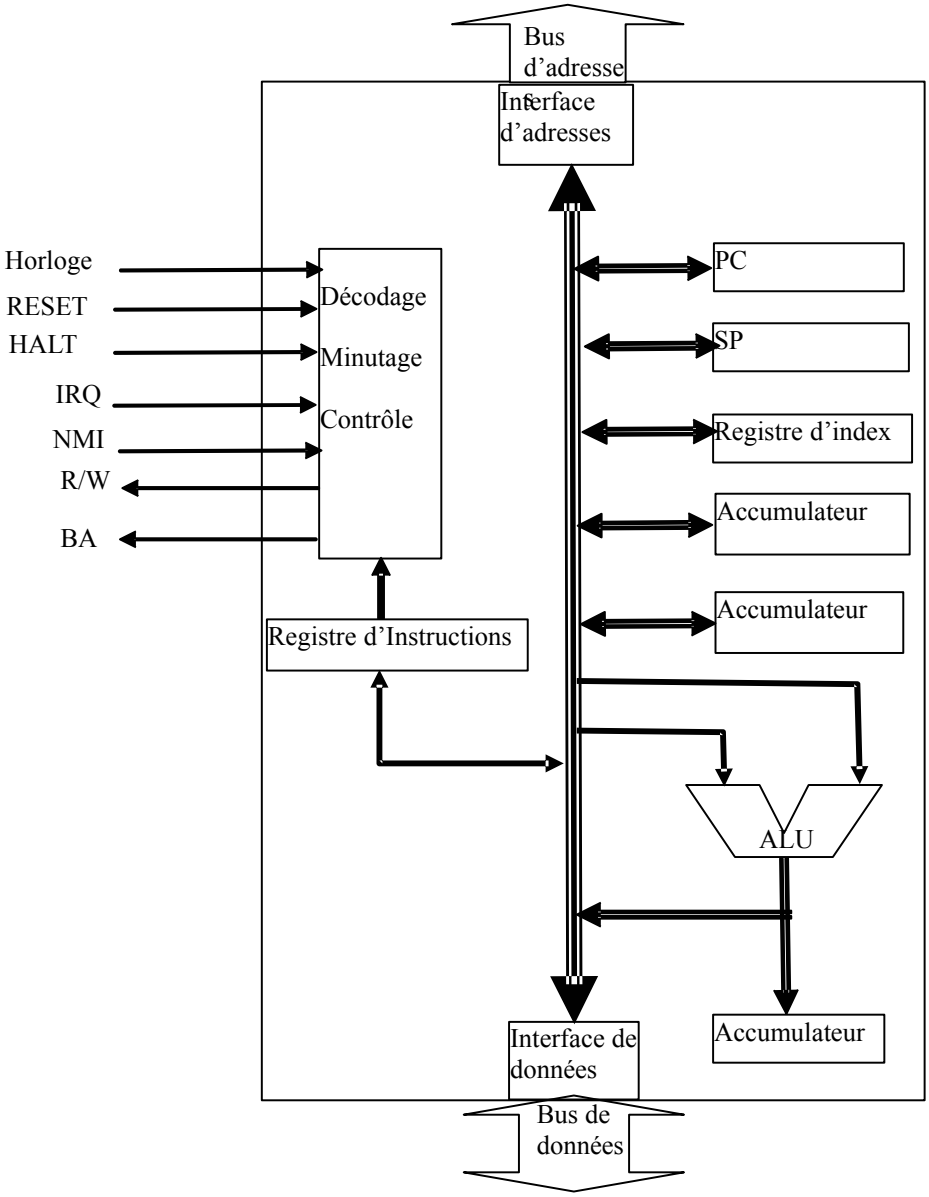


Fig.5 : Synoptique 1 d'un microprocesseur

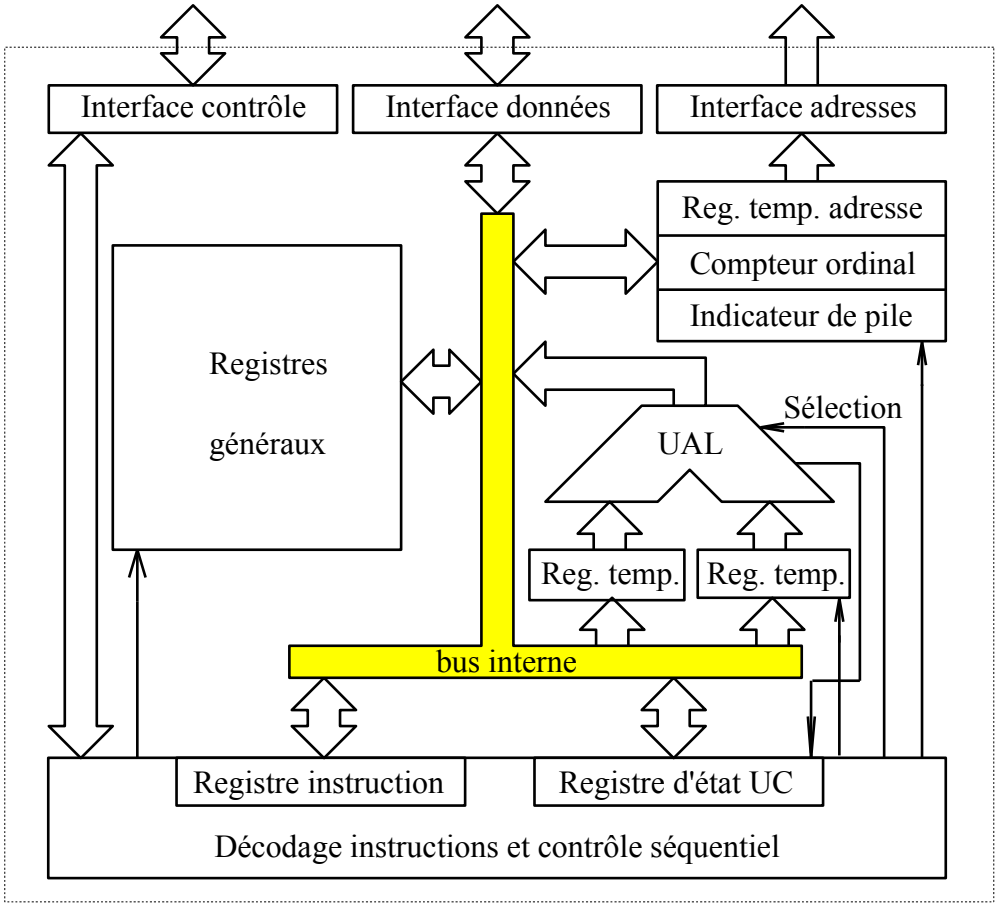


Fig.6 : Synoptique 2 d'un microprocesseur

Exemple d'exécution d'un programme simple
Le problème consiste à additionner 9 + 14(supposons codes hexa) en rangeant le résultat dans l'accumulateur.
Séquence des instructions :
1° chargement de 9 dans l'accumulateur
2° addition de14 au contenu de l'accumulateur (le résultat reste dans l'accumulateur)
Organigramme

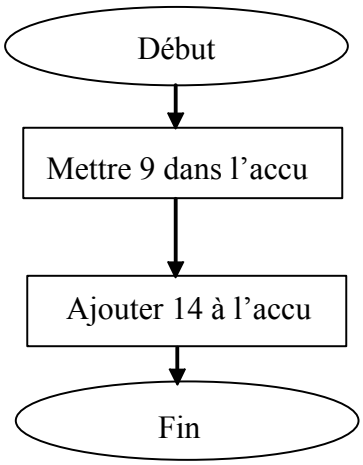


Fig.5 Organigramme du programme d'addition 9 + 12, résultat restant dans l'accumulateur
On suppose que le programme en mémoire commence à l'adresse 102.

On suivra l'exécution de du programme par les images des registres et de la mémoire

Instructions	Codes sur 2 octets
Mettre 9 dans l'accumulateur	3E 09
Additionner 14 à l'accumulateur	C6 14

Tableau des instructions du programme.

106
14 105
C6 104
09 103
3E 102
102
101

Comment le programme occupe la mémoire (mémoire de mot de 8 bits)

1° Situation au départ du programme

CO 0102

RI ? C6 104

Accu ?

3E	14	105
	09	103
	102	

2° Lecture du 1er octet d'instruction

CO 0103

RI 3E C6 104

Accu ?

3E	14	105
	09	103
	102	

3° Décodage et recherche de la donnée

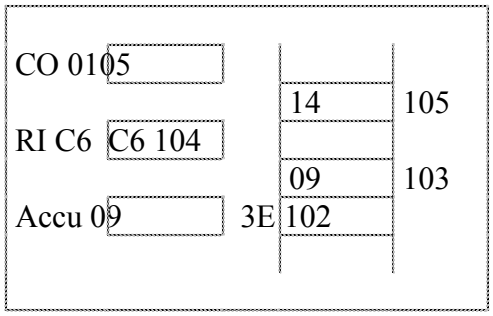
CO 0104

RI 3E C6 104

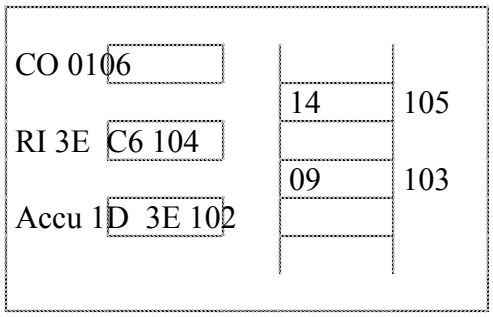
Accu 09

3E	14	105
	09	103
	102	

4° On passe à la 2ème instruction



5° Exécution de l'addition et fin



2.3.3 Le parallélisme

Le parallélisme consiste à exécuter simultanément sur des processeurs différents des instructions relatives à un même programme. Cela se traduit par le découpage d'un programme en plusieurs processus qui seront traités par des processeurs différents dans le but de gagner en temps d'exécution. Cela nécessite toutefois une communication entre les différents processus. C'est le même principe de fonctionnement que dans une entreprise: le travail est divisé en petits processus traités par des services différents et qui ne servent à rien si la communication entre les services ne fonctionne pas (ce qui est généralement le cas dans les entreprises...).

On peut distinguer le parallélisme interne, à l'intérieur d'un ordinateur classique et le parallélisme externe défini pour plusieurs processeurs travaillant en liaison plus ou moins étroites.

Le parallélisme interne peut se présenter sous trois formes :

Parallélisme par duplication : on peut réaliser une simultanéité réelle de certaines opérations ou actions, en multipliant les dispositifs affectés à ces tâches. Exemples : additionneur parallèle, bus multilignes pour transferts parallèles, mémoires à blocs indépendants entrelacés, additionneur ou multiplication supplémentaire.

Parallélisme par anticipation : on réalise un accroissement des performances en effectuant certaines actions susceptibles de réduire le temps d'attente d'une unité critique. Exemple : recouvrir dans le temps l'exécution d'une instruction avec la recherche de l'instruction suivante, doter le CPU d'une antémémoire, DMA envol de cycle, recouvrement d'actions dans un pipeline.

Parallélisme par multiplexage : lorsque plusieurs unités lentes sont servies par une unité rapide, on réalise une simultanéité apparente. Exemple : systèmes utilisés en temps partagé [*time-sharing*].

Pour le parallélisme externe on distingue, selon la classification de Flynn trois types d'architecture :

SISD [Single Instruction Single Data stream]: il s'agit tout simplement de la machine de Von Neumann, strictement séquentielle et dépourvue de parallélisme externe. Elle possède en fait

une seule unité de commande traitant une seule séquence d'instructions et une seule unité d'exécution traitant une unique séquence de données.
SIMD [Single Instruction Multiple Data stream] : il s'agit de machines ayant une unité de commande unique mais plusieurs unités d'exécution.
MIMD [Multiple Instruction Multiple Data stream] : Dans cette catégorie on retrouve les multiprocesseurs (processeurs partageant une même mémoire) les multiordinateurs. (processeurs ayant chacun leur propre mémoire locale et travaillant ensemble par l'entremise d'un réseau)

2.3.4 Le pipelining

Le pipelining est un principe simple à comprendre. Un programme comporte généralement des portions de code (plus ou moins grandes) qui sont traitées de nombreuses fois par le processeur. Le pipelining consiste donc à éviter d'avoir à réitérer de nombreuses fois des instructions que l'on a déjà traitées en fournissant directement le résultat!

2.3.5 L'architecture CISC

L'architecture CISC (Complex Instruction Set Computer, ce qui signifie "ordinateur avec jeu d'instructions complexes") est utilisée par tous les processeurs de type x86, c'est-à-dire les processeurs fabriqués par Intel, AMD, Cyrix, ...
Les processeurs basés sur l'architecture CISC peuvent traiter des instructions complexes, qui sont directement câblées sur leurs circuits électroniques, c'est-à-dire que certaines instructions difficiles à créer à partir des instructions de base sont directement imprimées sur le silicium de la puce afin de gagner en rapidité d'exécution sur ces commandes.
L'inconvénient de ce type d'architecture provient justement du fait que des fonctions supplémentaires sont imprimées sur le silicium, d'où un coût élevé.
D'autre part, les instructions sont de longueurs variables et peuvent parfois prendre plus d'un cycle d'horloge ce qui les rend lentes à l'exécution étant donné qu'un processeur basé sur l'architecture CISC ne peut traîter qu'une instruction à la fois!

2.3.6 L'architecture RISC

Contrairement à l'architecture CISC, un processeur utilisant la technologie RISC (Reduced Instruction Set Computer, dont la traduction est "ordinateur à jeu d'instructions réduit") n'a pas de fonctions supplémentaires câblées. Cela impose donc des programmes ayant des instructions simples interprétables par le processeur. Cela se traduit par une programmation plus difficile et un compilateur plus puissant. Cependant vous vous dîtes qu'il peut exister des instructions qui ne peuvent pas être décrites à partir des instructions simples...
En fait ces instructions sont tellement peu nombreuses qu'il est possible de les câbler directement sur le circuit imprimer sans alourdir de manière dramatique leur fabrication.
L'avantage d'une telle architecture est bien évidemment le coût réduit au niveau de la fabrication des processeurs l'utilisant. De plus, les instructions, étant simples, sont exécutées en un cycle d'horloge, ce qui rend l'exécution des programmes plus rapides qu'avec des processeurs basés sur une architecture CISC.
De plus, de tels processeurs sont capables de traîter plusieurs instructions simultanément en les traitant en parallèle.

2.3.7 CISC ou RISC

A comparer les spécificités des deux types d'architecture on pourrait conclure que les processeurs basé sur une architecture de type RISC sont les plus utilisés...
Cela n'est malheureusement pas le cas... En effet les ordinateurs construits autour d'une architecture RISC nécessitent une quantité de mémoire plus importante que les ordinateurs de type CISC

2.4 Types de microprocesseurs

On distingue les microprocesseur par la largeur de leurs bus de données c'est-à-dire le nombre de bits de données que ce dernier peut transmettre en parallèle. On peut alors distinguer les microprocesseur à mot de :

- 4 bits rares d'utilisation
- 8 bits utilisation courante
- 16 bits
- 32 bits
- 64 bits récents

Le nombre de bits exprime la longueur du mot de données utilisé dans le type de machine considéré.
Quelques exemples de microprocesseurs.

Mot de données (bits)	Microprocesseur Constructeur	
4 4004		Intel
8080/8085 Intel		
8 Z80		Zilog
6800/6802/6809 Motorola		
6502/65C02 MOS		Technology
8086/8088 Intel		
16 80186/80188;	80286 Intel	
68000 Motorola		
32 80386/80386SX;	80486 Intel	
6x86Mx Cyrix		
K5		AMD
68008/68010/68020/68030/68040		Motorola
64 Pentium(32/64) Intel		
K6		AMD
6x86(32/64) Cyrix		

2.5 Historique

2.5.1 Quatre générations informatiques de « minis »

Génération 1ère 2ème 3ème 4ème				
Nom de la machine	whirlwind	PDP-1	PDP-8/1	LSI-11
Composant	Tubes électroniques	Transistors Circuits intégrés		Microprocesseur
Année 1950		1960	1965	1976
encombrement	Un bâtiment	Armoires	Un rack	Une carte
Dimensions	15*15*6 m	2,4*0,75*1,8 m	60*60*60 cm	22*25*1,2 cm
Consommation(watts)	150 000	2 500	250	50
Nbre d'accès mémoire/s	80 000	200 000	600 000	900 000

2.5.2 Historique des microprocesseurs

Année	Nom	Registres/Bus données (bits)	Espace d'adressage (bits)	Constructeur	Observations
1969	4004	4/4	1k	Intel	Ens.de60000op./s
1972	8008	8/8	16k	Intel	
1974	8080	8/8	64k	Intel	64k d'adresses, 290.000op/s
6800	8/8			Motorola	
Z80	8/8			Zilog	
6502	8/8			Mos Technologies	
1976	8085	8/8	64k	Intel	8080 reconditionné
1978	8086	16/16	1M	Intel	
1979	68000	32/16	16M		
1980	8088	16/8	1M	Intel	
1981					PC sur 8088 par IBM
1982	80186	16/16	1M	Intel	8086 + E/S sur une puce
	80188	16/16	1M	Intel	8088 + E/S sur une puce
68008	32/8		4M	Motorola	
80286	16/16		16M	Intel	Espace d'adressage augmenté
1983	68010	32/16	16M	Motorola	Support mémoire virtuelle
68012	32/16		2G	Motorola Version	du 68010
1984	68020	32/32	4G	Motorola	Vrai UC 32 bits
1985	80386	32/32	4G	Intel	Vrai UC 32 bits
1987	68030	32/32	4G	Motorola	MMU sur le circuit
1988	80386SX	32/16	4G	Intel	80386 avec bus 80286
1989	80486	32/32	4G	Intel	Version du 80386 plus rapide
68040	32/32		4G	Motorola	Version plus rapide du 68030
1993	Pentium	32/64	4G	Intel	Processeur CISC orienté RISC
1995	Pentium pro	64/64	4G	Intel	5.5 millions de transistors
1997	Pentium® II	64/64	4G	Intel	7.5 millions de transistors
1999	Pentium® III	64/64	4G		
2001	Pentium® IV	64/64	4G		

2.6 Circuits du micro-ordinateur

Pour obtenir un micro-ordinateur performant câblé avec le microprocesseur, i l faut : une horloge à quartz , les puces de mémoire, des puces d'adaptation d'interface, des bus.

2.6.1 L'horloge à quartz

Circuit qui produit des signaux périodiques à partir d'un oscillateur à quart. Ces signaux se présentent sous forme d'impulsions de période T. Les circuits de commande et séquencement mettent les différentes sections du microprocesseur en service à tour de rôle par période de d'horloge. La cadence à laquelle l'horloge contraint le microprocesseur de fonctionner est sa fréquence de fonctionnement c'est-à-dire le nombre d'opérations élémentaires qu'il peut effectuer par seconde. Elle se mesure en MHz (1MHz = 10^6 Hz). Le temps d'exécution de chaque instruction élémentaire est indiqué par le constructeur du microprocesseur en périodes d'horloge.

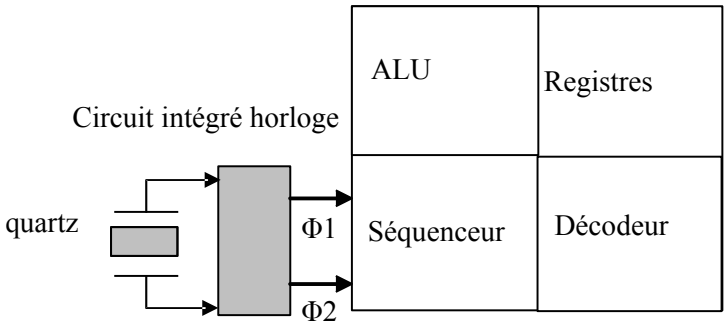


Fig.6 : Circuit d'horloge

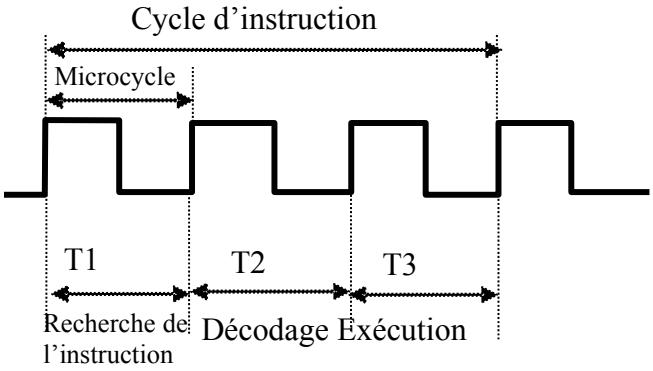


Fig.7 : Signal d'horloge et cycle d'instruction

Exemple : Soit une horloge de 50 MHz alors la période d'horloge T= 1/50 000 000 = 20 ns
Une instruction à trois microcycles sera complètement traitée en un temps t = 3x20 ns = 60 ns
On appelle :

microcycle la période d'horloge

cycle d'instruction l'ensemble des périodes nécessaires pour exécuter une instruction.

2.6.2 Mesure des performances des microprocesseurs

Il n'existe pas d'instrument capable de mesurer de façon rigoureuse les performances des ordinateurs afin de les comparer entre eux. Ce que l'on peu mesurer avec les microprocesseurs c'est :

La fréquence de travail (en Mhz), plus elle est élevée plus on a la chance de travailler vite.

La vitesse d'exécution des instructions. On applique au microprocesseur l'instruction NOP (no-operation) et l'on compte combien il en ingurgiter en une seconde.

Le nombre d'instructions exécutables par seconde se mesure en MIPS(millions d'instructions par seconde) .
On parle aussi de SPECfp ou SPECint(pour des entiers) pour les mesures de vitesse d'exécution d'instructions en virgule fixe ou en virgule flottante

2.6.3 Horloge système

Le circuit d'horloge système est destiné à fournir les signaux d'horloge de base au microprocesseur (exemples : 8284, 82284, 82384). Désormais ce circuit est intégré dans des ensembles incluant outre l'horloge, des circuits de logique pour le système. Le circuit dispose

d'une entrée RES qui reçoit le signal POWER GOOD de l'unité d'alimentation. Ce signal s'établit lorsque les tensions se sont stabilisées et autorise ainsi l'horloge à fonctionner, et par conséquent le système à démarrer.

2.6.4 Contrôleur de bus

Le contrôleur de bus (exemple : 82288) sert de relais entre le processeur et les bus. En fait le processeur n'est pas capable de fournir la puissance électrique nécessaire pour attaquer la charge que représentent les lignes des bus. Ainsi dépend-il de l'intervention de tels circuits tampons.

2.6.5 Horloge calendrier

Encore appelé RTC (Real Time Clock : horloge en temps réel) ce circuit en technologie CMOS (faible consommation d'énergie, exemple : MC 146818) permettant de conserver à la fois la date et les données de la configuration.

Caractéristiques d'un RTC :

- Une horloge sur 12 ou 24 heures
- Un pilotage par quartz
- Un calendrier
- Une zone mémoire indépendante de 50 octets à l'origine pour conserver les informations vitales (mémoire CMOS)
- Un calendrier programmé sur un siècle
- Faculté d'émettre une interruption

2.6.6 Les mémoires

Dans un ordinateur on distingue deux grand groupes de mémoires : les *mémoire centrales* et les *mémoires périphériques* ou mémoires de masse.

2.6.6.1 Les mémoires centrales

C'est la mémoire de travail du microprocesseur, c'est le lieu où programmes et données sont stockés avant exécution.

a) Typologie

Elles se divisent en deux grandes catégories : les mémoires vivent appelées RAM (Random Access Memory) à contenu volatile et les mémoires mortes appelées ROM (Read Only Memory) à contenu permanent.

b) Organisation

La mémoire centrale est organisée en bits, en cellules ou mot mémoire.

Le bit est la plus petite quantité d'information représentable(0 ou 1). Les bit sont regroupés en cellules ou mots mémoires ; chaque cellule porte un numéro de référence appelé adresse notée habituellement en hexadécimal. L'adresse d'une cellule est différente de son contenu.

2.6.6.2 Les mémoires de masse

a) Typologie

Encore appelées mémoire d'archivage elles servent à l'archivage des données. On distingue

- les mémoires magnétiques: les disques durs, les disquettes, les bandes magnétiques
- les mémoires laser: CD-ROM, DVD.
- Les disques flash

b) Organisation

Comme la mémoire centrale les mémoires de masse sont organisées en bits réalisés pour des disques et bandes magnétiques par l'orientation des charges magnétiques sur une mince pellicule recouvrant la surface du disque ou de la bande; pour les disques laser cela est réalisé par les trous ou les pleins pratiqués sur la surface du disque en sillon.

CHAPITRE 3: LA MÉMOIRE

C'est la partie de l'ordinateur dans laquelle programmes et données sont rangés.
On distingue les mémoires principales (à semi-conducteurs) des mémoires secondaires ou mémoires de masse (disques durs, disquettes, disques optique CD et DVD, bandes magnétiques, flash disk)

3.1 La mémoire centrale

3.1.1 Bit et cellule mémoire

Le bit est la plus petite unité d'information envisageable : un chiffre binaire (0 ou 1)
Une mémoire est formée d'un certain nombre de cellules contenant chacune une certaine quantité d'informations (bits) ; on parle aussi de mot mémoire.
La cellule mémoire est la plus petite quantité d'information adressable.
Le nombre de bits d'une cellule mémoire définit la taille ou la longueur du mot mémoire.

3.1.2 Les adresses mémoire

Chaque cellule a un numéro, son adresse, qui permet à un programme de la référencer. Pour une mémoire de constituée de n cellules les adresses vont de 0 à n-1.
Si le mot d'adresse est constitué de m bits alors on peut adresser jusqu'à 2^m cellules distinctes. La taille du mot d'adresse ne dépend que du nombre total de cellules et non de leur taille.

3.1.3 Ordonnancement des octets (indianisme)

Les octets d'un mot peuvent être numérotés de gauche à droite ou de droite à gauche : on parle respectivement de mémoire "gros-boutiste" ou "big endian" et de mémoire "petit-boutiste" ou "little indian".
Les microprocesseurs Intel sont "little-indian" alors que ceux de Motorola sont "big-indian"

Adresses mémoire

	N	N + 1	N + 2	N + 3	N + 4	N + 5
Gros-boutiste	A N 2 0	0 1				

	N + 5	N + 4	N + 3	N + 2	N + 1	N
Petit-boutiste	1 0 0 2	N A				

Figure 3.1 Représentation gros-boutiste et petit-boutiste

3.1.4 Organisation d'une mémoire

3.1.4.1 Organisation interne d'une mémoire

Une cellule est registre de n bits n étant la taille du mot mémoire. Un bit est réalisé par une bascule D.
Quelques signaux :
Les données: $D_{n-1} D_{n-2} \ldots\ldots D_1 D_0$
Les adresses: $A_{m-1} A_{m-2} \ldots\ldots A_1 A_0$
Les Commandes: *CS* Chip Select
$R/\overline{w}$ Read or write
OE Output enable
Équation d'activation de la sortie *:* $S = CS.\ R/\overline{w}.OE$

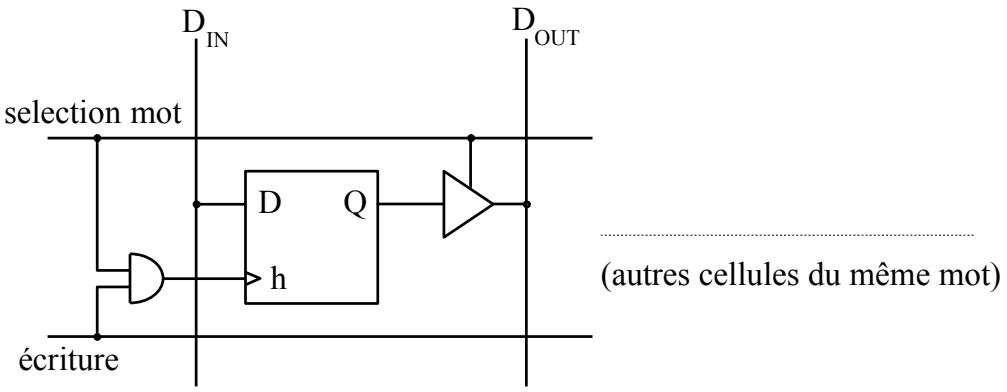
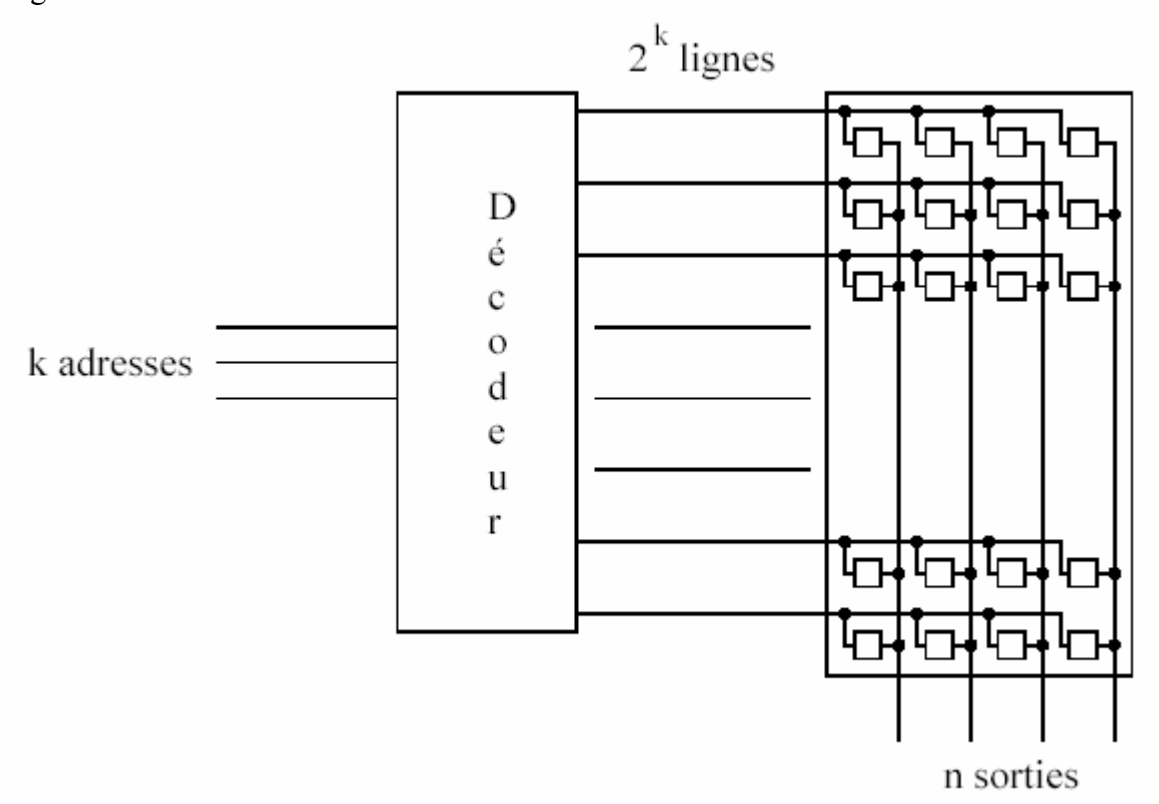
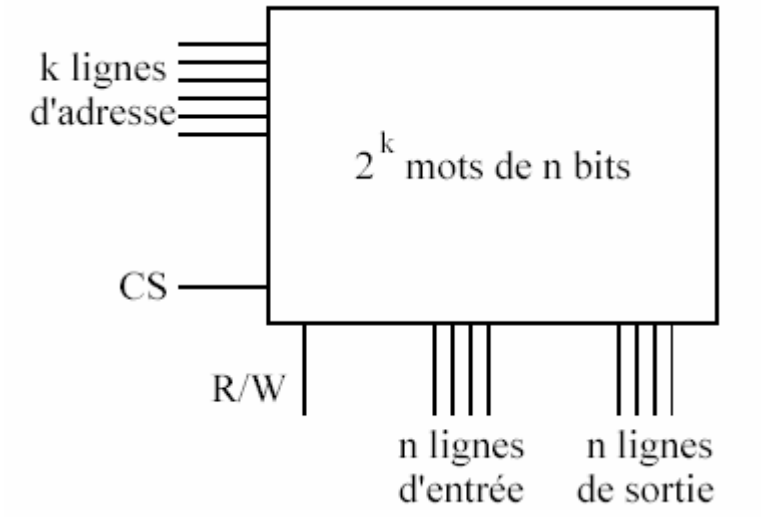
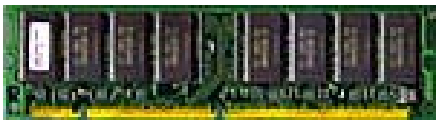

D IN
D OUT
selection mot
D
Q
h
écriture
(autres cellules du même mot)

Figure 3.2


2 k lignes
D
é
c
o
d
e
u
r
k adresses
n sorties


k lignes
d'adresse
2 k mots de n bits
CS
R/W
n lignes
d'entrée
n lignes
de sortie

3.1.4.2 Aspect extérieur des mémoires centrales
On distingue plusieurs aspects des mémoires centrales:
Circuits intégrés classiques: en boîtier DIP (Dual In-line Package, 24 broches)
Batterie de CI: regroupés en 8 ou 9, le 9ème circuit servant à contrôler la parité.
En barrettes SIMM (Single In-line Memory Module) à simple rangée de connexion ou DIMM (Dual In_line Momory Module) à double rangée de connexion.



3.1.5 Caractéristiques d'une mémoire
Une mémoire est caractérisée par sa capacité et son temps d'accès notés sous la forme : *NxT – t,* où N indique le nombre de cellules, *T* taille d'une cellule et *t* le temps d'accès à la mémoire. Exemple : 32K x 8 - 60 est une mémoire de 32 K mots de 8 bits chacun soit une capacité de 256 Kbits avec un temps d'accès de 60 ns. Cette mémoire peut s'organiser de plusieurs façons : 256Kbits = 32K x 8 bits = 16K x 16 bits = 8K x 32 bits.
La capacité est la quantité d'information que peut contenir cette mémoire (fonction du nombre de cellules et la taille d'une cellule).
Le temps d'accès indique le temps minimum pour lire l'information dans la cellule et la placer sur le bus.
Le temps d'attente ou *wait state* est le temps (en périodes d'horloge) supplémentaire accordé à la mémoire.
Exemple: pour un processeur cadencé à 50 Mhz (20 ns de période), la lecture ou l'écriture en mémoire impose 4 périodes d'horloge soit 4 x 20 ns = 80 ns.
Si le processeur est cadencé à 500Mhz (2 ns de période) , il faut 35 périodes d'horloge, soit 35 x 2 ns = 70 ns.
3.1.6 Typologie des mémoires
Elles se divisent en deux grandes catégories : les mémoires vives appelées RAM (Random Access Memory) et les mémoires mortes appelées ROM(Read Only Memory).
3.1.6.1 Les mémoires vives
Ce sont des mémoires à accès aléatoire, accessible en lecture et en écriture. Elles ont pour avantage d'être rapides et pour inconvénient d'être volatiles c'est-à-dire que leur contenu disparaît à la mise hors tension. La mémoire RAM stocke les programmes et les données. Un ordinateur comporte couramment aujourd'hui 8 ; 16 ; 32 ; 64 ; 128 Mo.
On distingue plusieurs variantes : DRAM (Dynamic RAM), SRAM (Static RAM), VRAM (Vidéo RAM), SDRAM (Synchronous DRAM), NVRAM, etc.
Elles se divisent en deux grandes familles : les RAM statiques et les RAM dynamiques.
Les RAM statiques disposent d'une structure interne basée sur des bascules D.
Les RAM dynamiques disposent d'une structure interne basée sur des condensateurs et la mémorisation d'un bit est matérialisée par l'absence ou la présence d'une charge électrique aux bornes du condensateur. Cette charge électrique a tendance à diminuer c'est pourquoi les RAM dynamiques doivent être rafraîchies régulièrement pour maintenir la mémorisation.
On appelle mémoire quasi-statique une mémoire dynamique disposant d'un dispositif de rafraîchissement interne.
La mémoire CMOS, accessible par le programme Setup, est une RAM dont le contenu est maintenu par une pile qui sert aussi d'alimentation de l'horloge.
3.1.6.2 Les mémoires mortes
ROM : Ce sont des mémoires à accès en lecture seulement dont un des avantages en dehors de leur rapidité est d'être non volatiles. Leur capacité dans un micro-ordinateur se chiffre en quelques Ko et stockent essentiellement le programme de démarrage BIOS(Basic Input Output System), le premier programme que lit l'ordinateur.

On distingue aussi plusieurs variantes : PROM(Programmable ROM), EPROM(erasable PROM), EEPROM(Electrically erasable PROM), EAPROM(Electrically alterable PROM), Flash.

3.1.7 Relation entre mémoires et microprocesseur

La mémoire comme le reste des éléments de l'ordinateur est mis en relation avec le microprocesseur par l'intermédiaire du bus général (trois bus spécialisés)

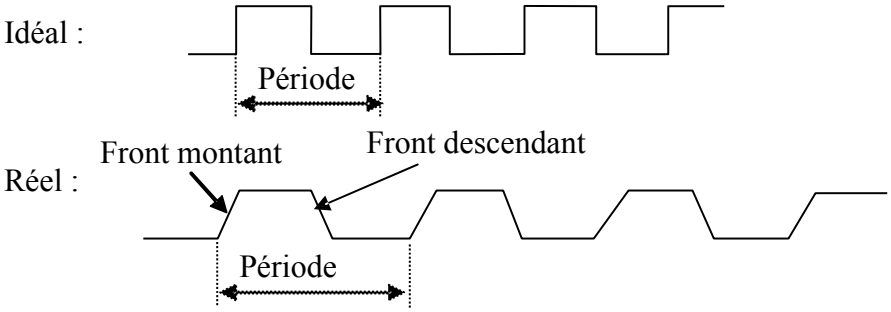
- les sorties d'adresse microprocesseur sont reliées aux entrées d'adresses de la mémoire ;
- les lignes de données et de commande du microprocesseur sont reliées aux lignes correspondantes de la mémoire ;

En pratique il existe plus d'un circuit mémoire.

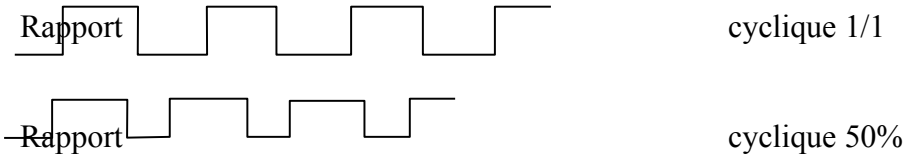
3.1.8 Chronogrammes types des mémoires

Dans toute transaction avec la mémoire il bien synchroniser les commandes et les adresses pour lire ou écrire les données.

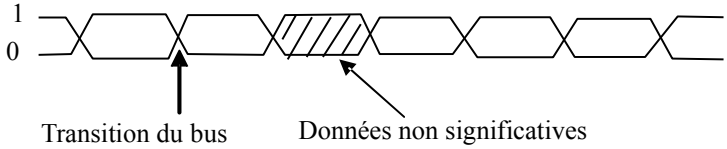
Impulsions d'horloge :



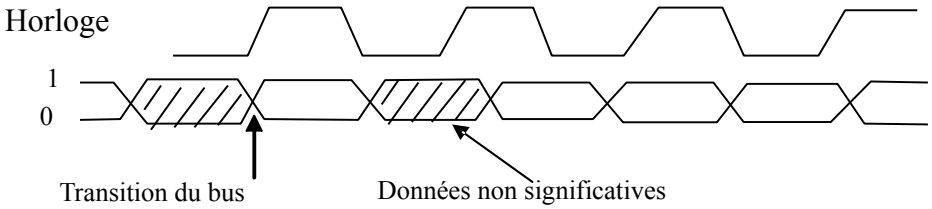
Une période ou cycle d'horloge est un cycle complet du signal c'est-à-dire deux alternances. Les deux alternances peuvent être de même durée ou de durées différentes ce qui est défini par le rapport cyclique ou rapport des alternances :



Chronogramme du bus



Chronogramme d'écriture en mémoire



Séquence d'écriture
1° le microprocesseur place une adresse sur le bus d'adresse
2° il met une donnée sur le bus de donnée ;
3° il émet une commande de lecture ou écriture sur le bus de commande ;
4° après un certain temps nécessaire à la stabilisation des signaux, il place une commande d'exécution sur le bus de commande.
Le cycle d'instruction : une instruction est exécutée en cycle d'instruction qui peut exiger plusieurs cycles de bus.
Un cycle de bus ou cycle machine représente une opération unique et complète de lecture ou d'écriture en mémoire. Les principaux cycle de bus sont :

Recherche d'une instruction par lecture en mémoire ;
Lecture d'une donnée en mémoire ;
Ecriture en mémoire.

Mode simple et mode rafale
En mode rafale ou salve , le processeur émet une adresse et lit en salve une série de cellule mémoire.
Exemple de lecture de données
Mode simple (8 cycles de bus):

Adresse	Donnée	Adresse	Donnée	Adresse	Donnée	Adresse	Donnée

Mode simple (5 cycles de bus):

Adresse	Donnée	Donnée	Donnée	Donnée

Multiplexage du bus
Dans certains cas on utilise un seul bus pour émettre aussi bien les adresses que les données. Dans ce cas on émet d'abord l'adresse , la mémoire la retient(la verrouille) puis on place la donnée en lecture ou en écriture.
Intérêt : économie de bus
Inconvénient : complication de la gestion du bus.

3.1.9 Principe du décodage d'adresse

Exemple : mémoire de 64 Ko en 8 modules de 8 Ko.
Nombre de lignes d'adresse : 16 ($A_{15} A_{14} \ldots\ldots A_1 A_0$)
Chaque module de 8 Ko (2^{13} octets) peut être adressé par 13 bits d'adresse($A_{12} A_{11} \ldots\ldots A_1 A_0$) ; les trois bits $A_{15} A_{14} A_{13}$ passent par un décodeur 3 vers 8pour sélectionner un des 8 circuits($8 = 2^3$)

Carte d'adressage

Module adressé	$A_{15}A_{14}A_{13}$ Binaire Hexadécimal			
de à	de			à
0 000 000	00000000000000	0001111111111111	0000	1FFF
1 001 001	00000000000000	0011111111111111	2000	3FFF
2 010 010	00000000000000	0101111111111111	4000	5FFF
3 011 011	00000000000000	0111111111111111	6000	7FFF
4 100 100	00000000000000	1001111111111111	8000	9FFF
5 101 101	00000000000000	1011111111111111	A000	BFFF
6 110 110	00000000000000	1101111111111111	C000	DFFF
7 111 111	00000000000000	1111111111111111	E000	FFFF

3.2 Les mémoires dans les PC

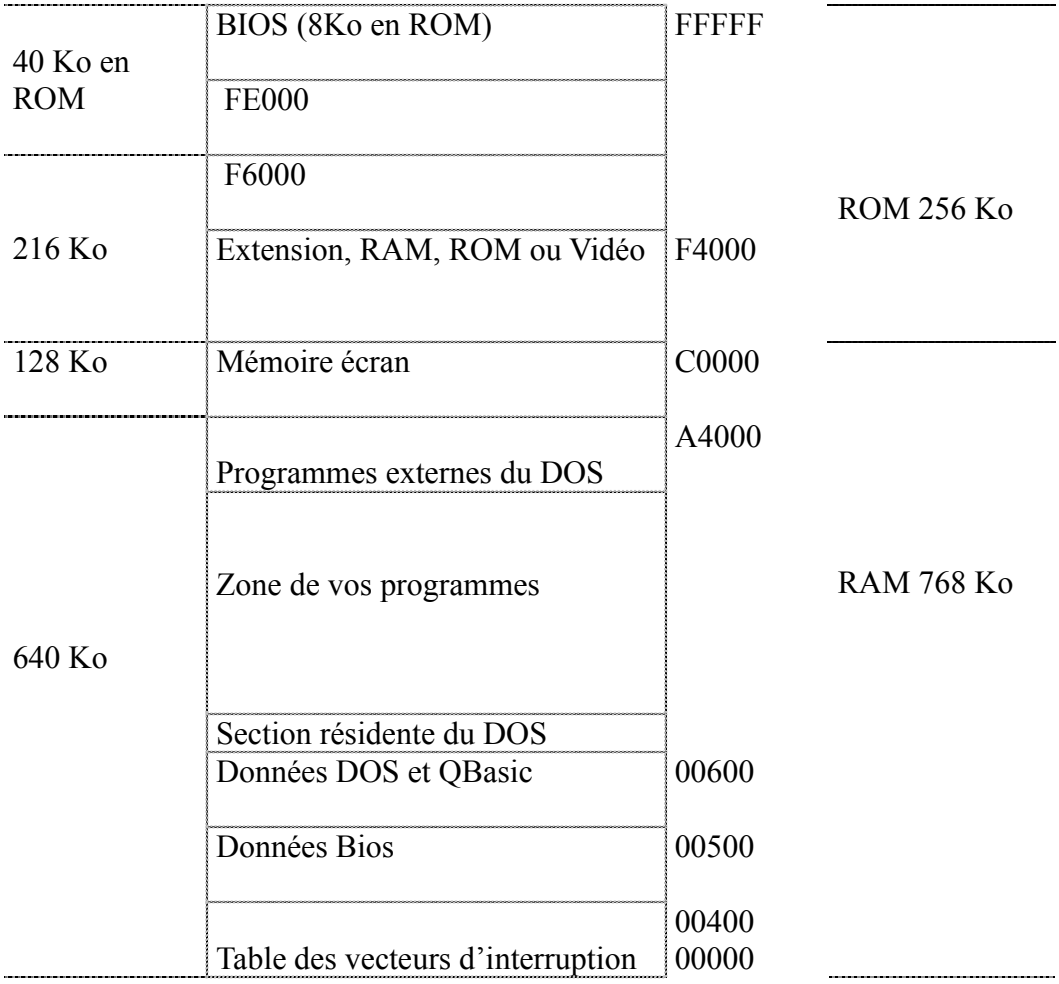


Figure 3.3 Exemple classique d'organisation d'une mémoire centrale de micro-ordinateur de PC d'origine IBM pour son premier méga-octet

3.2.1 La barrière des 640 Ko et catégories des mémoires

Le 8086/8088 a un espace mémoire adressable de 1 M mots(20 lignes d'adresse, soit 1 Mo si la taille du mot mémoire est de 8 bits, 1 octet) utilisé par le DOS comme suit :

640 Ko pour les programmes de l'utilisateur ;

384 Ko (situé entre 640 Ko et 1 Mo) pour la gestion interne de la machine.

La barrière des 640 Ko a pu être brisée à partir des processeurs 80286(24 lignes d'adresse, 16 Mo), 80386 et 80486(32 lignes d'adresse, 4 Go) et le pentium. La rupture s'est aussi faite de manière logique par Windows. Mais la contrainte de compatibilité ascendante a fait conserver ces considérations.

Catégories logiques des mémoires centrales :

La mémoire conventionnelle : constituée par les 640 Ko maximum ;

La mémoire supérieure ou UMA (Upper Mémory Area) : espace des 384 Ko au-dessus des 640 Ko réservé aux besoins du DOS ; les blocs libres de cet espace se nomment UMB (Upper Memory Blocks)

La mémoire paginée ou expansée : mémoire qu'on peut ajouter à la mémoire conventionnelle et supérieure, mais qui ne peut pas être directement exploitée par le processeur ;

La mémoire étendue : mémoire supplémentaire situé au delà du 1 ^{er} méga-octet et adressée directement par les processeurs 286, 386, 486 et pentium fonctionnant en mode « protégé »

3.2.1.1 La mémoire paginée et les EMM

La mémoire paginée est gérée par des utilitaires spécifiques appelées EMM(Expanded Memory Manager) chargés dans la mémoire de l'ordinateur par le fichier système config.sys. Le programme EMM se trouve d'abord un espace de 64 Ko (cadre de page) dans la zone des 384 Ko réservé au DOS puis le cadre de page est scindé en 4 pages de 16 Ko. Les données situées dans la mémoire supplémentaire (pages logiques) sont alors appelées au fur et à mesure dans les 4 pages physiques.

Pour fonctionner le programme EMM et la mémoire supplémentaire doivent être conçus l'un pour l'autre ; une association au hasard risque de ne pas fonctionner. le programme d'application doit savoir faire appel aux pages de 16 Ko situées dans cette mémoire, faute de quoi celles-ci resteront inexploitées.

3.2.1.2 Les EMS

Les spécifications des EMM sont définies par les « EMS » ou Epanded Memory Spécification.

Quelques EMS :

EMS LIM : produit de Lotus Intel et Microsoft (définition d'un bloc contigu de 64 Ko divisé en pages de 16 Ko), il peut adresser jusqu'à 8 Mo de mémoire supplémentaire(suffisant pour les données mais insuffisant pour faire transiter le code des programmes)

EEMS (enhanced Epanded Memory Spécification): produit de ATS, permet l'emploi de plus de 4 pages qui peuvent être logées n'importe où dans le 1er Mo.

EMS version 4.0 : produit de Lotus, Intel , Microsoft et ATS, les pages peuvent être non contiguës et se situer dans le 1 ^{er} Mo leur taille peut atteindre jusqu'à 1 Mo. On peut accéder à 32 Mo de mémoire expansée. Le programme d'application peut modifier la taille de la mémoire expansée.

EMM386 : gestionnaire de mémoire expansée sous DOS et Windows, fait appel à la mémoire étendue afin de simuler de la mémoire paginée.

Les émulateurs sur disques : ce sont des simulateurs de mémoire expansée non plus sur la mémoire physique centrale mais sur un disque. On brise la barrière de 64 Ko de cadre de page mais le temps d'accès au disque pénalise la vitesse de travail.

3.2.1.3 La mémoire étendue

En mode protégé les processeurs 286, 386, 486 et pentium peuvent gérer directement un espace d'adressage supérieur à 1 M. Ces processeurs peuvent fonctionner sous trois modes différents.

Mode réel : mode exclusif des 86/88, l'adresse ne porte que sur 1Mo (20 bits d'adresse). Dans ce mode même un processeur plus évolué fonctionne comme exactement comme le 8086/8088 ;
Le mode protégé : grâce au passage par des registres du microprocesseur appelés descripteurs, l'espace adressable est suffisamment étendu (24 bits soit 16 Mo pour le 286, 32 bits soit 4 Go pour les 386DX et 486)
Mode virtuel : sous mode du mode protégé, ne s'applique qu'à partir du 386. il permet l'exécution des conçus pour le mode réel. ; chaque programme se voit attribuer un espace de 1 Mo dans l'espace du mode protégé.

3.2.1.4 La mémoire haute ou HMA

En mode réel et sous DOS le 286 et le 386 sont capables de gérer sous Dos et en mode réel la zone haute de la mémoire ou UMA(High Memory Area), un espace de 64 Ko situé au delà du 1er méga octet. Cette possibilité fait suite à un bogue du 286 qui, fonctionnant en mode réel génère un « 1 » sur la ligne d'adresse A_{20} alors qu'elle doit rester à « 0 »
L'exploitation de cette erreur impose qu'on ait de la mémoire physique à cet emplacement (64 K au-dessus du 1 er Mo). Le DOS peut alors se loger dans ce cet espace et libérer de la mémoire conventionnelle pour des programmes résidents et des drivers , bien sûr à condition qu'on ait insérer au préalable les lignes suivantes dans le fichier config.sys :

DIVISE = C:\DOS\HIMEM.SYS

DOS = HIGH

Un gestionnaire virtuel le driver WINA20.386 ramène la ligne A20 à « 0 » lorsque l'adresse dépasse FFFFF en mode réel et permet de résoudre d'éventuels conflits entre DOS et Windows lorsque tous deux tentent d'accéder simultanément aux HMA en mode 386 étendu.

3.2.2 Adressage des mémoires

L'adressage des mémoires par le processeur s'effectue en mode "segmenté", l'espace mémoire étant divisé en segments de 64 Ko maximum.
Les processeurs 386 et supérieurs disposent, de plus d'un mode "paginé" qui gère la mémoire par "pages" de 4 Ko

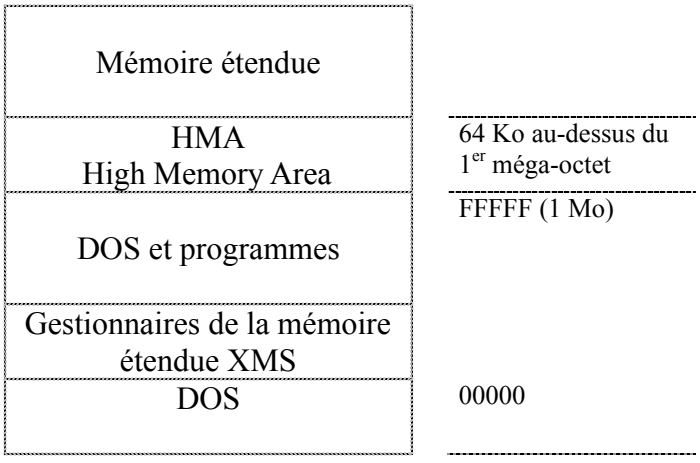


Figure 3. 4 Positionnement de la zone HMA en mémoire. La mémoire étendue est la partie de la mémoire excédant le 1er méga-octet.

3.2.2.1 Segmentation de la mémoire

Un segment est une portion de mémoire de 64 Ko au maximum(selon 86/88). Le microprocesseur dispose d'un registre de segment qui pointe un segment en mémoire via une adresse de 16 bits sur les 20bits nécessaires pour un espace de 1 méga.

En hexadécimal on multiplie cette adresse par 16 (10H), soit un décalage de 4 bits (un chiffre hexadécimal) pour obtenir l'adresse du segment sur 20 bits. Pour référencer une cellule mémoire on spécifie l'adresse de segment, l'adresse de la cellule dans le segment ou "déplacement" par rapport au début du segment. La syntaxe est la suivante:

Segment: Déplacement

Exemple 2BB5:0110, spécifie le segment 2BB5 (16 bits) ou 2BB50 (20 bits) avec un déplacement de 0110(16 bits).

La segmentation permet de scinder les programmes en partitions logiques et leur appliquer des protections efficaces. Elle oblige cependant que le segment soit rangé de manière contiguë et en totalité en mémoire physique centrale.

Le concept de segmentation est efficace mais très peu souple. On lui associe souvent le concept de pagination.

3.2.2.2 Pagination de la mémoire

Le contenu d'un segment est fractionné en sections linéaires de 4 Ko : ce sont des pages qui peuvent être manipulée individuellement grâce à l'unité de pagination du microprocesseur. C'est ce concept qui permet au DOS de loger en mémoire supérieure (zone des 384 Ko au-dessus des 640 premiers Ko) des programmes résidants et des drivers avec les processeurs 386 et 486.

Quelques réservations:

128 Ko pour la RAM vidéo aux adresses 0A0000 – 0BFFFF

128 Ko pour DOS aux adresses 0E0000 – 100000

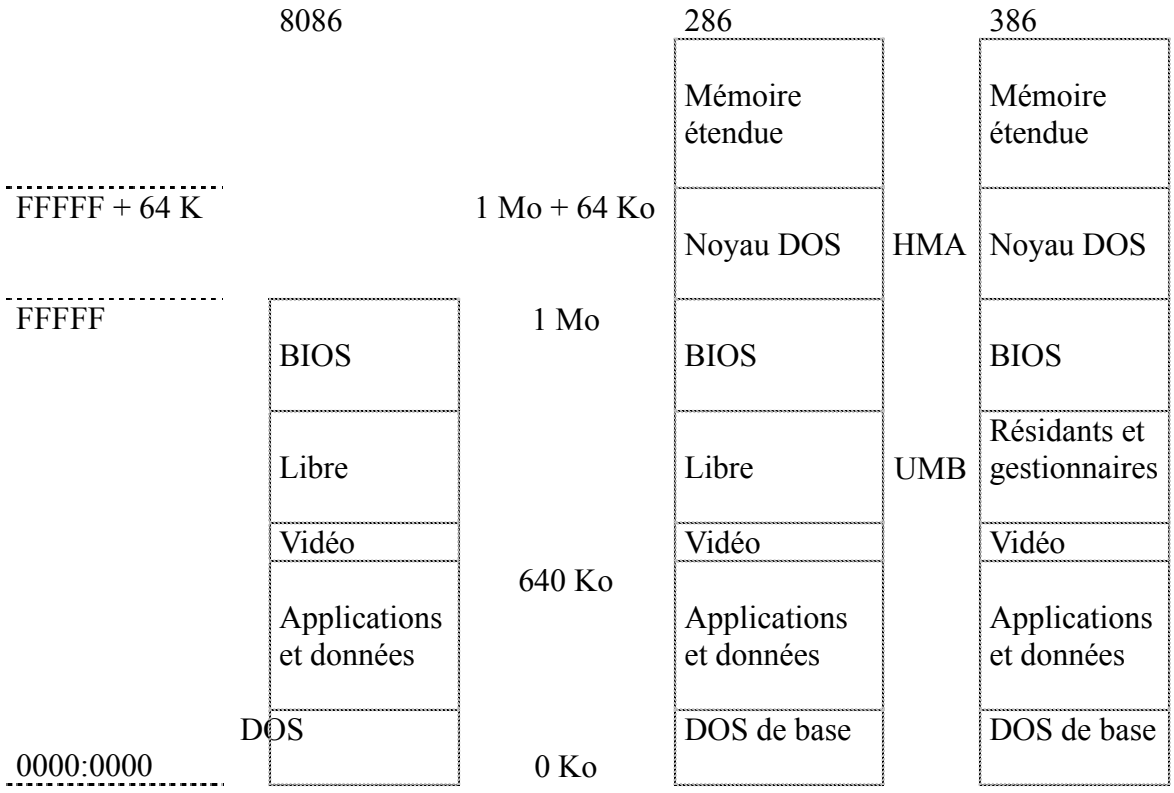


Figure 3. 5 Organisation de la mémoire en fonction du processeur (avec DOS 6)

3.2.2.3 Les blocs de mémoire supérieure (UMB)

On nomme par UMB (Upper Memory Blocks ou blocs de mémoire supérieure les espaces libres de la zone mémoire réservée au système (espace situé entre les 1ers 640 Ko et le 1er Mo). On peut y loger des pilotes de périphériques et des gestionnaires divers (attention aux conflits avec le système, paralysie possible de l'ordinateur).

La commande *MEMMAKER* permet de calculer et de monter automatiquement en UMB les séquences qui s'y prêtent.
Les UMB sont gérées par le gestionnaire du DOS *EMM386* qui doit alors être inséré au fichier config.sys après l'appel de *HIMEM.SYS*

3.2.3 La mémoire virtuelle

C'est un espace disque qui simule de la mémoire centrale.
Cette méthode offre l'avantage de disposer d'une mémoire centrale apparente considérablement accrue et économique.
Elle présente l'inconvénient du ralentissement des applications à cause des raisons suivantes:

- Le disque est beaucoup moins rapide que la mémoire centrale;
- Les échanges s'opèrent dans les deux sens entre le disque dur et la mémoire centrale: on parle du *swapping*.

Windows applique ce concept lorsqu'on travail en mode 386 étendu.

3.2.4 Les gestionnaires de mémoire et analyse des mémoires

Dans DOS 6 et Windows on peut trouver les gestionnaires suivants:

- HIMEM.SYS, conforme à la norme XMS 2.0, il donne accès à la mémoire étendue et aux HMA;
- EMM386.EXE, repend la mémoire étendue gérée par HIMEM.SYS afin d'émuler de la mémoire expansée et/ou fournir des UMB;
- RAMDRV.SYS, crée un disque virtuel,
- SMARTDRV.EXE, crée un cache pour les disques.

Analyse de la mémoire par le BIOS et le DOS
L'autotest d'initialisation de la machine par le BIOS fournit la 1 ère analyse de la mémoire. Ce test ne peut détecter que des pannes matérielles, auquel cas un code position du circuit incriminé est indiqué. Avec le DOS avec la commande MEM on peut connaître le contenu et l'organisation générale des mémoires.

3.2.5 La mémoire CMOS

Tout micro-ordinateur doit être configuré par déclaration de son organisation. Cela se fait par un écran d'initialisation, le setup(par positionnement des cavaliers sur la carte mère dans l'ancien temps) ou par des méthodes automatiques du "plug and play" ou PnP. Les informations fournies dans le setup sont mémorisées dans la mémoire CMOS (RAM dont le contenu est maintenu par une pile. Ce circuit porte aussi l'horloge.

3.3 Les modes d'adressage des mémoires

A chaque interaction avec la mémoire le microprocesseur doit spécifier la cellule mémoire à laquelle il s'adresse: c'est l'adressage des mémoires.
Les adresses spécifiées par le microprocesseur concernent aussi bien les mémoires centrales les circuits d'entrées/sorties que les registres internes.
Il existe plusieurs modes d'adressage des mémoires centrales dont les modes:

- Implicite;
- Inhérent;
- Immédiat;
- À registre;
- Direct et absolu;
- Indirect par registre;
- Indirect par mémoire;
- Indexé;
- Relatif;
- Par base.

On peut aussi ajouter des compositions de ces modes.
Format général d'une instruction:

Code opération	opérande

Selon le cas on peut avoir 0 ou plusieurs (1 à 8) octets pour l'opérande alors que le code opération est généralement sur 1 octet.

3.3.1 Adressage implicite

Dans ce mode d'adressage l'instruction fait implicitement référence à l'un des registres internes du microprocesseur.
Format de l'instruction:

Code opération

Exemple: L'instruction *DAA(correction* d'erreu*r d'addition en complément à 2)* fait directement référence à l'accumulateur.

3.3.2 Adressage inhérent ou à registre

C'est une variante de l'adressage implicite, mais dans ce mode l'instruction spécifie nommément un registre. Exemple: INCB (incrémenter le registre B).
Format de l'instruction:

Code opération

3.3.3 Adressage immédiat

La donnée sur laquelle porte l'instruction se trouve dans l'instruction elle-même.
Format de l'instruction:

Code opération	Donnée immédiate

Code opération	Valeur haute	Valeur basse

ou

Donnée immédiate(2 octets)

On parle indifféremment de donnée immédiate d'opérande immédiate ou d'instruction immédiate.

3.3.4 Adressage de registre

A la différence de l'adressage inhérent le registre spécifié ne fait plus partie du code opération mais apparaît dans un octet à part.
Format de l'instruction:

Code opération	Registre	Donnée

Code opération	Registre

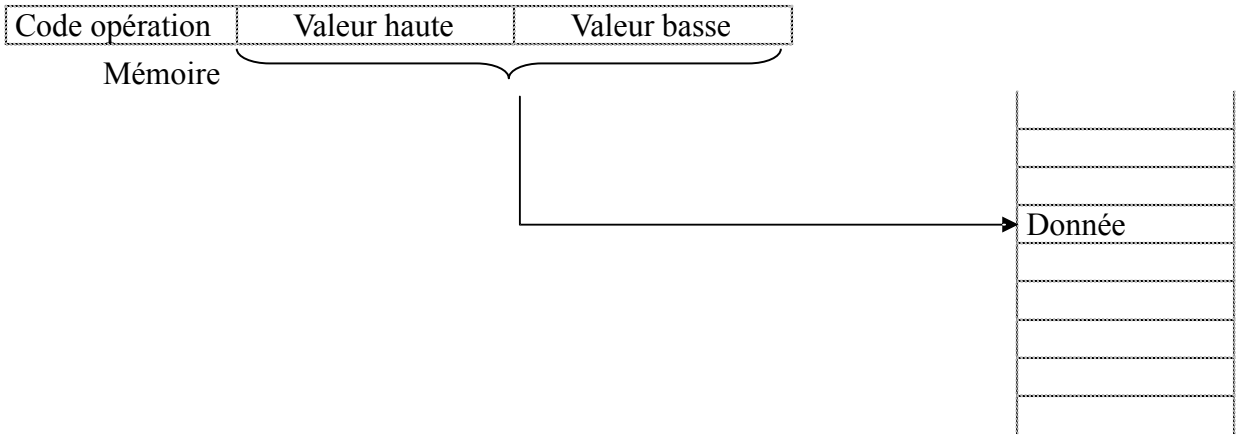
ou

Exemple: ADD C, 5 pour additionner 5 à C(le résultat reste dans C)

3.3.5 Adressage direct et absolu

L'opérande est l'adresse de la cellule dans laquelle se trouve la donnée.

Format de l'instruction:



La donnée est obtenue par lecture de la cellule mémoire dont l'adresse est passée par l'instruction; l'adresse de la donnée joue le rôle d'opérande.
La lecture de l'instruction demandera trois accès mémoire.
Motorola propose deux variantes d'adressage direct :

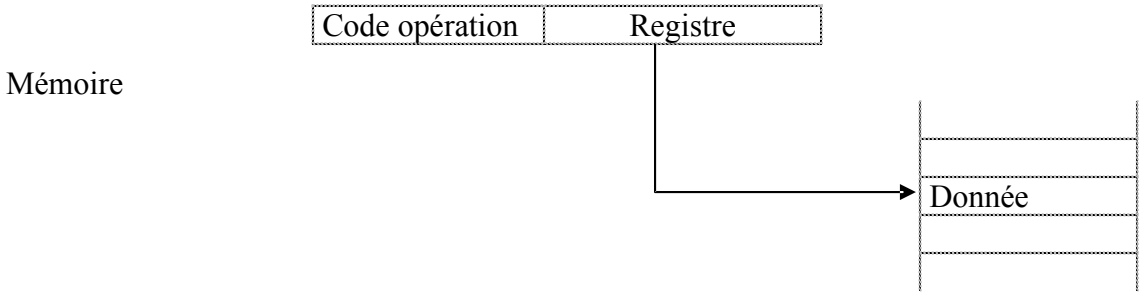
L'adressage direct restreint : utilisation d'un seul octet pour une adresse 16 bits;
L'adressage direct étendu : utilisation du format d'adresse correspondant à la largeur du bus d'adresse (16 bits pour un microprocesseur 16 bits d'adresse).

Variantes de l'adressage absolu:

L'adressage court: l'adresse est définie sur deux octets;
L'adressage long: l'adresse est définie sur quatre octets.

3.3.6 Adressage indirect par registre

Format de l'instruction

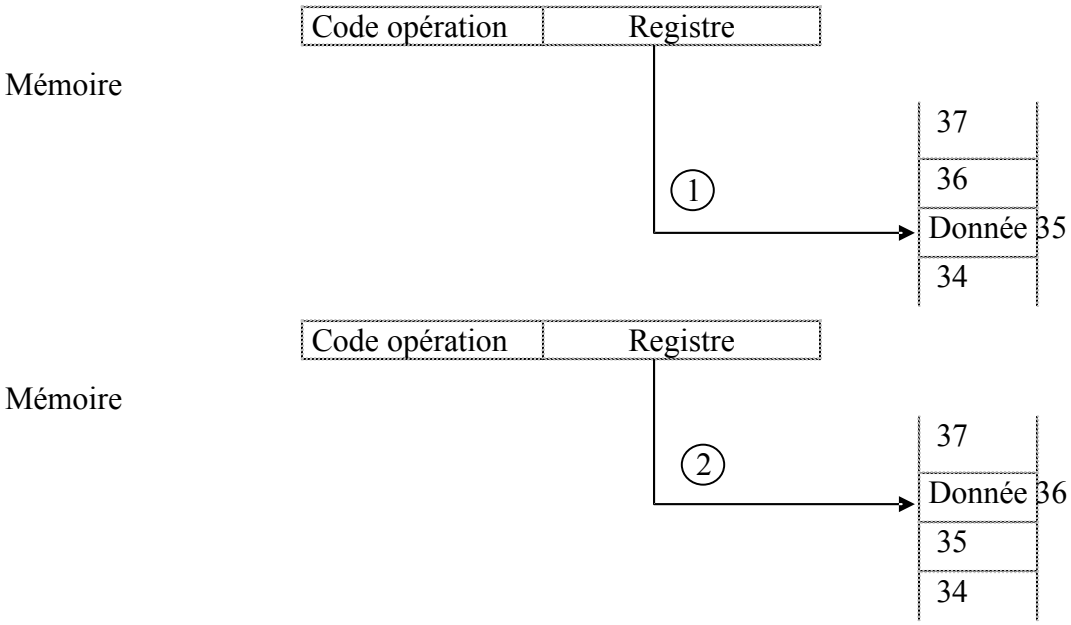


L'adresse de la donnée est contenu dans un registre spécifié par l'instruction. L'adressage de la donnée s'effectue en trois opérations:

Lecture de l'instruction;
Extraction de l'adresse de la donnée;
Adressage de la cellule de cette donnée.

Exemple MOV AX, [BX], range dans le registre AX le contenu de la cellule mémoire dont l'adresse est dans le registre BX.
NB: Les crochets chez Intel indiquent l'adressage indirect.

Adressage indirect par registre avec incrémentation ou décrémentation
Généralement on lit oui écrit dans des cellules mémoires successives. Ainsi pour passer d'une cellule à la suivante il faut augmenter ou diminuer de "1" l'adresse; on dit incrémenter ou décrémenter.
On peut aussi décrémenter ou incrémenter de toute autre mais en l'absence de toute spécification la valeur est "1".



Pré ou post-incrémentation/décrémentation

Le registre d'adresse peut être incrémenté avant ou après l'exécution de l'instruction (lecture de l'opérande.

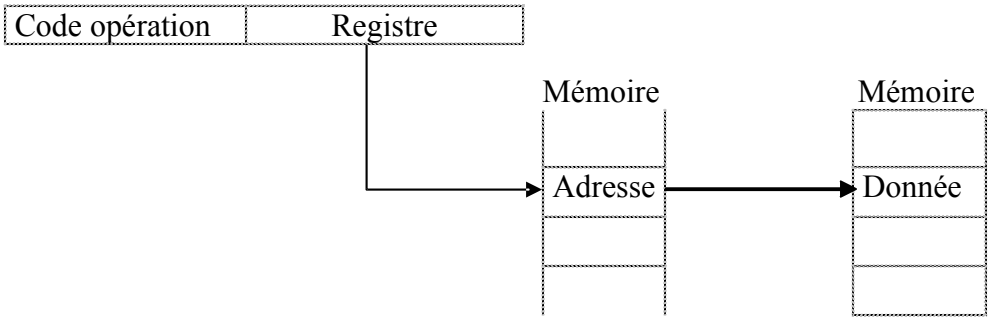
Pré-incrémentation/décrémentation
1°- lecture de l'instruction
2°- elle pointe le registre d'adresse
3°- le registre d'adresse pointe l'opérande
4°- Lecture de l'opérande
5°- Le registre d'adresse est modifié
6°- Le registre d'adresse pointe la cellule suivante

Post-incrémentation/décrémentation
1°- lecture de l'instruction
2°- elle pointe le registre d'adresse
3°- Le registre d'adresse est modifié
4°- le registre d'adresse pointe l'opérande
5°- Lecture de l'opérande
6°- Le registre d'adresse pointe la cellule suivante

3.3.7 Adressage indirect par mémoire

L'instruction porte l'adresse de l'adresse de l'opérande

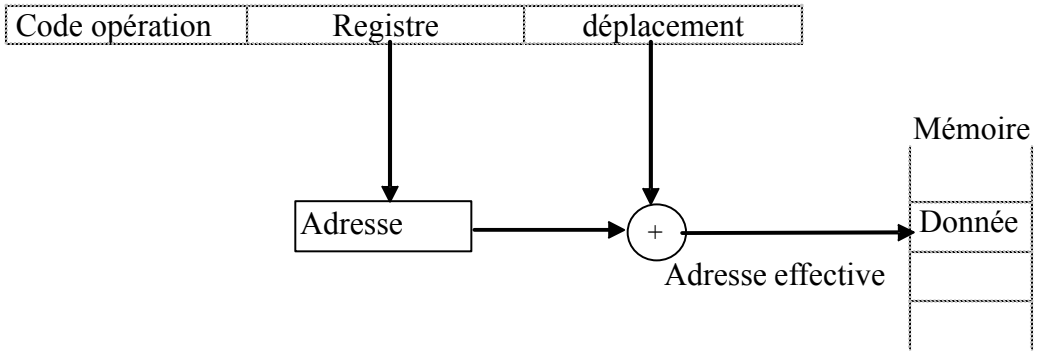
Cheminement
1°- lecture de l'instruction
2°- elle pointe une adresse x en mémoire
3°- l'adresse x contient une adresse y qui pointe l'opérande
4°- Lecture de l'opérande



3.3.8 Adressage relatif

3.3.8.1 Adressage relatif

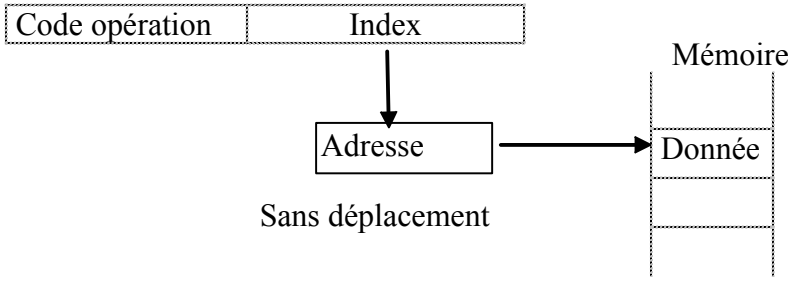
On utilise un adressage indirect par registre plus un déplacement.



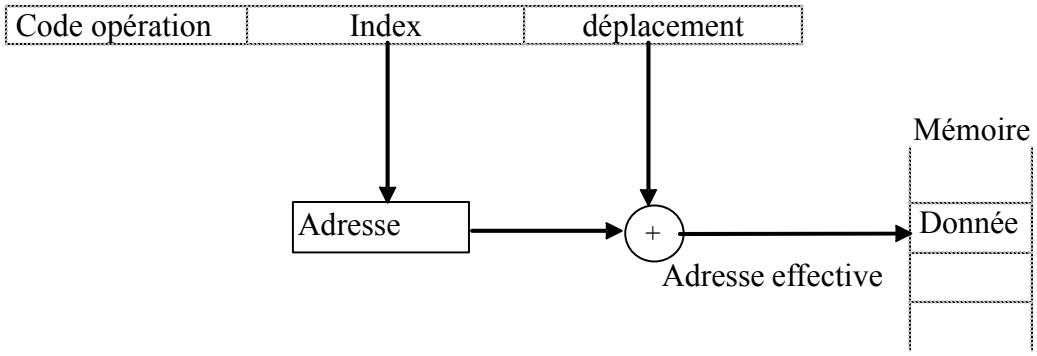
Le déplacement noté en complément à 2 parcourt [-128, +127]
Cheminement
1°- lecture de l'instruction
2°- elle pointe le registre d'adresse (ou d'index)
3°- le registre d'adresse fournit une adresse
4°- le déplacement est ajouté à l'adresse
5°- la somme fournit l'adresse effective
6°- lecture de l'opérande

3.3.8.2 Adressage relatif au contenu du compteur ordinal
Le compteur ordinal est un registre spécifique qui s'auto-incrémente d'une instruction à la suivante.
Exemple: supposons CO= 4432
Il ramène une instruction de trois cellules consécutives de la mémoire
A chaque fois il est incrémenté
Après lecture de l'instruction complète CO = 4432+3 = 4435
Si le déplacement codé dans l'instruction est +24 l'adresse effective sera 4435 + 24 = 4459 et non 4456.

3.3.9 Adressage indexé
Ce mode d'adressage fait intervenir des registres spécifiques ,les registres d'indexe. Le processus est une forme de l'adressage indirect par registre.



Cheminement
1°- lecture de l'instruction
2°- elle pointe le registre d'index
3°- le registre d'index pointe l'opérande
4°- lecture de l'opérande



Cheminement
1°- lecture de l'instruction
2°- elle pointe le registre d'index
3°- le registre d'index fournit une adresse
4°- ajout du déplacement pour donner l'adresse effective
5°- lecture de l'opérande

3.3.10 Adressage par registre de base
Il s'opère de la même manière que l'adressage indirect à la seule différence que le registre de a des propriétés spécifiques.

NB: On peut aussi combiner plusieurs de ces modes d'adressage pour autant qu'ils soient compatibles pour l'obtention de formules panachées.

3.3.11 Segmentation de la mémoire
Intel a fractionné la mémoire en segments, il s'agissait d'adresser une mémoire de 1 Mo (20 bits) par utilisation des registres limités à 16 bits.
Quelques explications:

Adresse effective (AE): est une adresse logique.
L'adresse logique peut résulter de l'addition d'une adresse de segment et d'un déplacement (offset).
Le mécanisme de segmentation transforme une adresse logique en une adresse linéaire.
L'adresse logique se confond avec l'adresse virtuelle. Elle peut adresser de la mémoire virtuelle sur disque.
L'adresse physique est l'adresse réelle, absolue de la cellule mémoire.

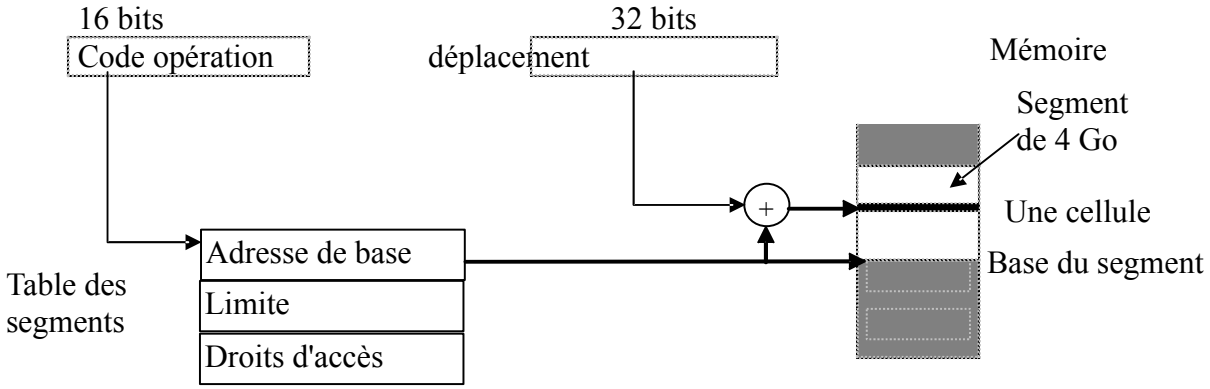
La gamme couverte par les adresses logiques peut être énorme puisqu'elle englobe l'adressage virtuel.
Exemple du 80486: adresses logiques sur 46bits pour des segments de 32 bits d'adresse

3.3.11.1 Principe de la segmentation avec les processeurs 16 bits
Cas d'Intel: la mémoire centrale peut être débitée par tranche de 64 K(16 bits) cellules, tranches pouvant se retrouver n'importe où et parfois se recouvrir partiellement ou totalement. La seule contrainte c'est que l'adresse de départ de segment soit divisible par 16. on peut adresser(sur 16 bits seulement) n'importe quelle cellule d'un segment d'adresse de départ connue.
On s'appuie sur deux types de registre:

Un registre de segment contenant l'adresse de d'un segment;
Un registre d'adresse effective (adresse logique) contenant une adresse au sein de ce segment.

Calcul de l'adresse physique:

1° [registre de segment (sur 16bits)] x 16 = adresse (sur 20 bits)
ou [registre de segment(4 chiffres hexadécimal)] x 10H = adresse(5 chiffres hexadécimal)
2° adresse (20 bits) + [registre d'adresse effective] = adresse physique (20 bits)
Segmentation avec le 486
Ce qui vient d'être présentation concerne des processeurs Intel d'espace mémoire de 1 Mo (20 bits) au maximum.
Pour le 486 ou le pentium le bus d'adresse compte 32 bits soit un espace mémoire de capacité totale 2^{32} adresses (4 Go) d'où la gestion des segments de longueur variable mais toujours inférieure à 4 Go.
Une tâche (une application) peut disposer de 16381 segment soit un maximum de 64 Téra-octets y compris la mémoire virtuelle.



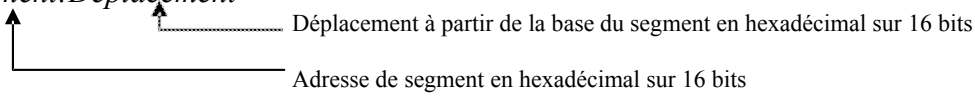
Mode de calcul de l'adresse physique d'une cellule mémoire avec l'adressage segmenté du 486
Le sélecteur est l'équivalent du registre de segment, mais adresse une table contenant essentiellement l'adresse de base, la limite supérieure du segment, les droit d'accès de ce segment.

3.3.11.2 Protection offerte par les segments
Les droits d'usage d'un segment sont associés aux registres de segment. Pour les processeurs Intel il y a 4 niveaux de protection ou privilèges.
Le niveau 0 est affecté de la plus grande protection (noyau du système d'exploitation avec les services généraux orientés microprocesseur: gestion de la mémoire, isolement des tâches, gestion multitâche, communication entre tâches, contrôle des entrées-sorties, etc.)
Le privilège 1 concerne les services du système avec des fonctions de haut niveau: cadencement des accès aux fichiers, gestion des périphériques, communication, affectation des ressources, etc. Ces éléments sont ainsi isolés des applications.
Le privilège 2 regroupe les extensions au système d'exploitation: gestion des bases de données, services d'accès aux fichiers, etc.
Les privilèges 0, 1, 2 relèvent du système.
Le privilège 3 est le niveau des applications. Ainsi une application n'a pas le droit de toucher aux ressources système. Les droits d'accès sont sévèrement réglementés.
3.3.11.3 Séparation des segments
Il est de bonne coutume de séparer en segments individuels les données relatives à un programme. On définit alors habituellement 4 segments qui sont:
1° le segment pour les instructions du programme
2° le segment pour les données (les variables)
3° le segment pour la pile de sauvegarde (pour usage avec les interruptions, etc.)
4° un segment supplément pour tout usage.
Pour les processeurs Intel depuis le 8086//8088, on trouve des registres spécialisés pointant l'adresse de ces segments:
CS: segment du code du programme

DS: segment des données (Data Segment)
SS: segment de pile de sauvegarde (Stack Segment)
ES: le segment libre (Extra-Segment)

3.3.11.4 Désignation d'une adresse

C'est l'existence des segments qui a défini le mode de désignation d'une cellule en mémoire.
Syntaxe : S*egment:Déplacement*



Exemple: F000:FFF0

3.3.11.5 Contraintes de la segmentation

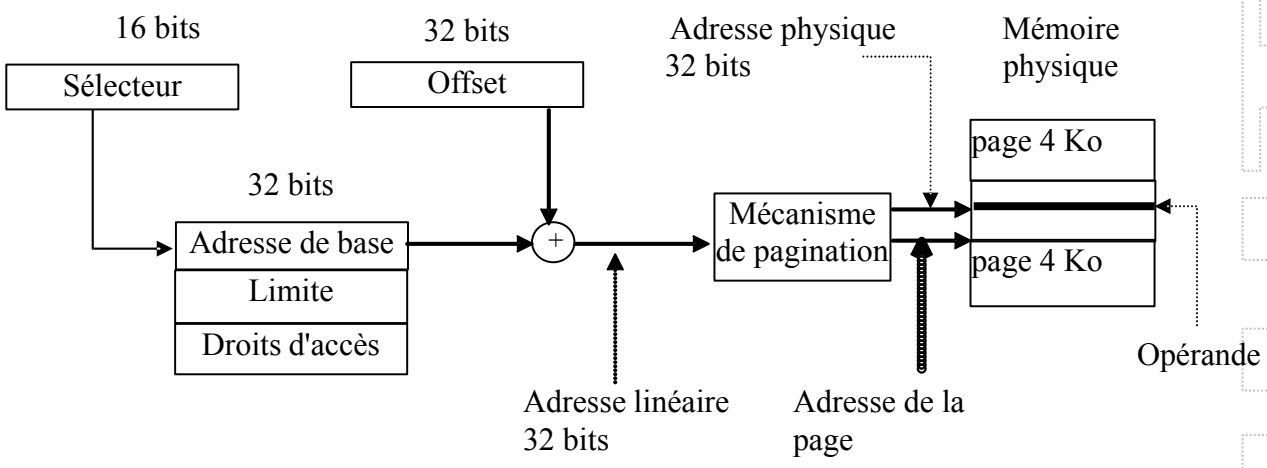
Avec la segmentation de la mémoire, les tâches sont scindées en partions logiques. Les segments sont translatables dans la mémoire. Ainsi le système d'exploitation peut allouer dynamiquement la mémoire physique à l'application, c'est-à-dire récupérer la mémoire libérée par une application pour l'allouer à une autre qui en a besoin. Ce principe permet la gestion de la mémoire virtuelle, les segments pouvant se prolonger sur le disque quand la mémoire physique est insuffisante. Le code de description du segment comporte souvent un bit spécifiant s'il se trouve en mémoire physique ou en mémoire virtuelle. Cela engendre les obligations suivantes:

- un segment doit être chargé en totalité dans la mémoire physique(ce qui peut entraîner une gêne si le segment est de grande taille);
- un segment doit être rangé dans une zone continue de la mémoire centrale, sans trou(2ème gêne possible).

Ainsi la segmentation est efficace mais pas toujours souple, elle ne permet pas l'optimisation de l'usage de la mémoire centrale.
Pour remédier à cela on lui associe le concept de pagination.

3.3.12 Pagination de la mémoire

La pagination fractionne le contenu d'un segment en sections linéaires d'une longueur fixe de 4 Ko, par convention : c'est une page.
Les adresses linéaires issues de la segmentation deviennent des adresses logiques pour la pagination, les pages de 4 Ko pouvant se loger n'importe où en mémoire.
Le mécanisme de pagination transforme l'adresse linéaire issue de la segmentation en adresse de page.



Pour une meilleure gestion de la mémoire une unité de gestion de gestion de la mémoire ou MMU Memory Management Unit) est associée au processeur intégré dans ce dernier ou dans un circuit séparé.

3.4 Accès direct en mémoire

Certains besoins tel que le transfert des données du disque dur vers la mémoire peuvent prendre énormément du temps au processeur et inutilement.
Une solution à cela est le transfert direct du disque vers la mémoire sans intervention active du processeur. D'autres unités en dehors des disques peuvent demander pareils services: on parle d'accès direct en mémoire ou *DMA* (Direct Mémory Acces) qui consiste à court-circuiter le processeur et dialoguer directement avec la mémoire.

L'échange en DMA reste toujours initialisé par le processeur qui, une fois l'échange initialisé, se déconnecte des bus internes du systèmes(bus de données et bus d'adresses) afin que l'échange direct s'opère. La déconnexion est rendue possible par les circuits trois états.
Le circuit demandeur de DMA le signifie sa demande au processeur par le signal HOLD (suspension, déconnexion)
Processeur répond par la ligne HoldA (Hold Acknowledge = acquisition de la demande de Hold) autorisant un autre circuit à prendre la maîtrise des bus et à exécuter un transfert en DMA.
Le transfert en DMA applique trois stratégies différentes, par ordre de complexité croissante:
1° Halte du microprocesseur
2° vol de cycle
3° multiplexage

3.4.1 DMA par halte (arrêt) du processeur

C'est la méthode la plus simple à appliquer. Elle fait intervenir un circuit spécialisé programmable appelé contrôleur de DMA qui se substitue au processeur lors d'un échange et gouverne cet échange.

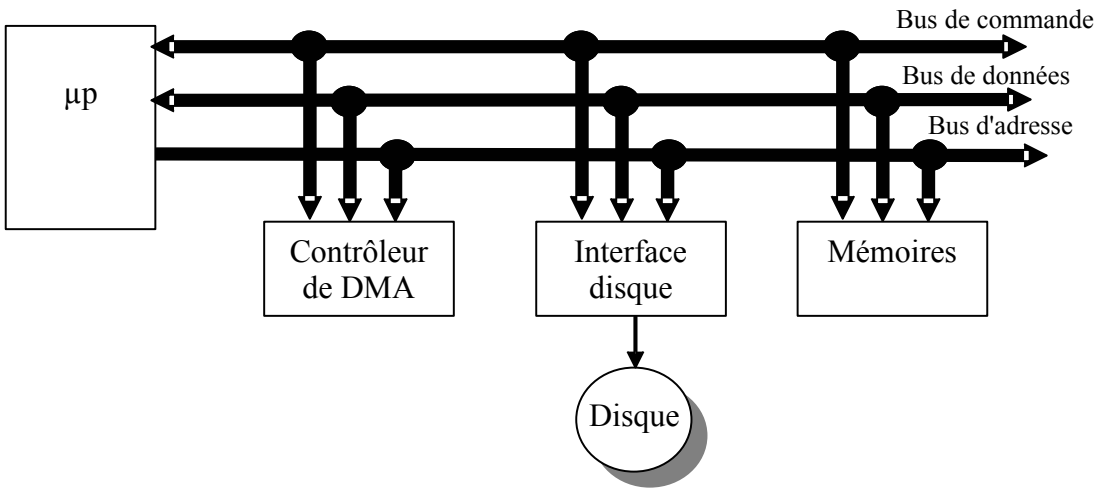


Figure 3.7 Contrôleur DMA dans le montage de l'ordinateur
Un exemple de contrôleur de DMA est le 8237 d'Intel.
Ce circuit comporte un ensemble de registres spécialisés dont:

Un registre d 'adresse de base : reçoit du programme une adresse initiale destinée au registre d'adresse courante.

Un registre d 'adresse courante : chargé par le registre d'adresse de base, il est incrémenté ou décrémenté à chaque transfert d'un octet pour suivre l'adresse de la position courante en mémoire.

Un registre de compte d'octets de base : il est chargé par la valeur du nombre total d'octets à transférer.
Un registre de compte courant : reçoit une copie du registre de compte d'octets de base et est en suite décrémenté octet après octet.
Un registre de commande servant à la programmation du circuit.
Un registre de mode déterminant le mode de fonctionnement.
Un registre temporaire : qui stocke l'octet à transférer.

Lorsqu'un transfert concerne deux éléments adressables le contrôleur doit disposer de deux jeux d'adresses et fait appelle à deux voies de transmission désignées par :
1° Canal 0 pour l'adresse source;
2° Canal 1 pour l'adresse cible.
L'adresse source permet de lire un octet via le canal 0 , l'octet est rangé provisoirement dans le registre temporaire puis basculé sur le canal 1 et transféré à la cible.

3.4.1.1 Organisation d'un échange

Le programme loge dans le registre de compte de base le nombre total d'octet à transférer en DMA, et dans les registres d'adresse les adresses de départ en mémoire à partir desquelles ces octets doivent être lus ou rangés. La procédure de transfère est alors la suivante:

Le contrôleur de DMA reçoit d'un périphérique une demande de DMA;
Il la transmet au processeur en lui demandant l'autorisation de prendre le contrôle des bus;
Le processeur termine l'instruction en cours d'exécution et émet une autorisation de DMA
Le processeur se déconnecte simultanément des bus de données et d'adresses;
Le contrôleur de DMA en prend la direction et lance les échanges. Pour chacun d'eux :
Il place l'adresse courante sur le bus d'adresse;
Il transfère un octet;
Il décrémente ou incrémente l'adresse;
Il décrémente le nombre d'octet à transmettre.
Lorsque le compte des octets arrive à 0 l'échange est achevé;
Le contrôleur lève la demande de DMA en remettant cette ligne à 0;
Le microprocesseur supprime l'autorisation de transfert en DMA et reprend le contrôle des bus;
Le processeur poursuit le programme en cours.

3.4.1.2 Avantages et inconvénients

Pendant le processus de DMA le processeur se déconnecte et se reconnecte sans sauvegarde ni restitution de ses registre ce qui favorise un gain de temps.
En revanche le processeur entre en état de sommeil et le programme principal voit son exécution totalement arrêtée. Une succession de DMA peut empêcher l'exécution de programme.
Le processus de DMA ne peut être lancé que si le processeur a terminé l'exécution de l'instruction en cours.

3.4.2 DMA par vol de cycle

Le principe du vol de cycle consiste à détourner une impulsion d'horloge du microprocesseur pour l'appliquer au transfert en DMA.
Les impulsions de l'horloge sont destinées toutes au microprocesseur mais les circuits spécialisés peuvent détourner une de ces impulsions et l'appliquer au circuit de DMA, il s'opère alors un transfert en DMA pendant ce cycle, et les cycles suivants reviennent au processeur comme si de rien n'était: c'est pourquoi on parle de vol de cycle(impulsion).
Le vol de cycle peut porter su un ou plusieurs cycles et être répétitif selon diverses cadences.
Avantages et inconvénients

Processeur n'est pas arrêté mais ralenti;
Il n'a plus besoin de finir l'instruction en cours; le DMA est donc pris immédiatement en compte;
Le transfert en DMA est moins rapide qu'en DMA par halte du processeur;

Il peut se faire en arrière-plan

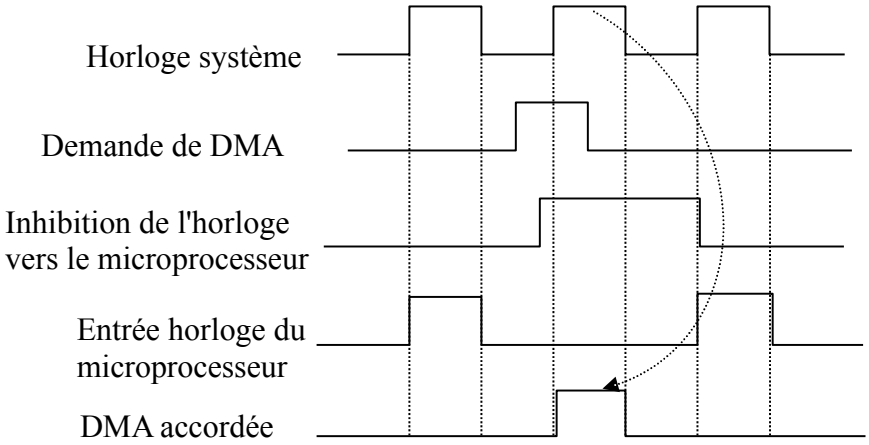


Figure 3.8 DMA par vol de cycle

3.4.3 DMA multiplexé

C'est la méthode la plus complexe : elle recourt aux «temps morts» des bus du système. C'est-à-dire les moments pendant lesquels les bus sont inoccupés. C'est le cas lorsque le processeur exécute des traitements internes ou des opérations arithmétiques. Dans ce cas les circuits de transfert en DMA détectent la disponibilité des bus, en prennent le contrôle et procèdent rapidement à un transfert.
Ce principe impose un second jeu de bus desservant le DMA avec un environnement logique très complexe. Des circuits tampons trois états séparent les deux jeux de bus. Le processeur attaque un jeu de bus et les circuits DMA l'autre. Ce système aboutit à un meilleur compromis de vitesses globales de travail, traitement par le processeur et DMA. Le processeur n'est ni stoppé ni ralenti. En revanche la technique est plus difficile à mettre en œuvre avec les processeurs avancés. (mode pipeliné, prérecherche)

3.4.4 Le contrôleur de DMA: 8237

Le contrôleur de DMA servant de référence est le 8237(circuit 40 broches). On utilise aussi ses variantes.
Un seul circuit permet l'exploitation de 4 voies de DMA, mais en cascadant deux unités on peut exploiter jusqu'à 7 lignes de DMA.
Les transferts se font sur un octet pour les voies de 0 à 3 et sur 2 octets pour les voies 4 à 7.

3.5 La mémoire cache et sa gestion

Le mémoire cache ou antémémoire permet d'adapter la vitesse de la mémoire principale à la vitesse du processeur. C'est une mémoire intermédiaire se situant entre le processeur et la mémoire centrale. C'est une mémoire RAM très rapide mais limité en capacité en raison de son coût.
La mémoire cache stocke une petite partie des données et/ou des instructions (lues par le processeur ou lues par le cache par anticipation) se trouvant dans la mémoire centrale.
Lorsque le processeur a besoin d'une donnée il la cherche d'abord dans le cache;

S'il la trouve on parle de succès (gain de temps)
S'il ne la trouve pas, il va alors la chercher dans la mémoire principale on parle d'échec (perte de temps relative)

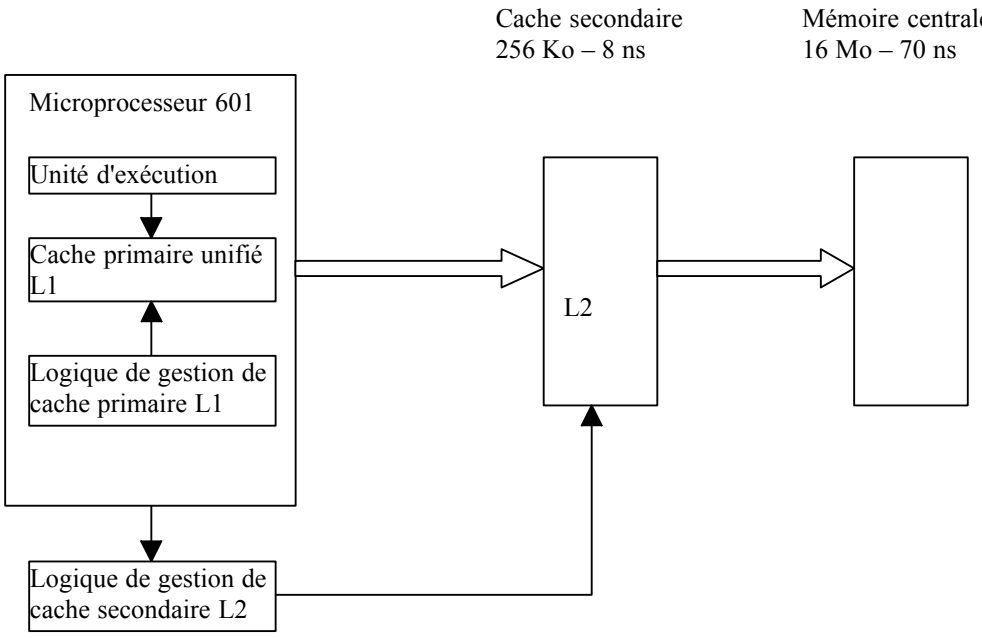
Les statistiques montrent que le taux de réussite peut atteindre jusqu'à 90% et même davantage (70% avec un cache de 8 Ko, 90% avec un cache de 64 Ko). Tout dépend cependant des conditions générales de fonctionnement et l'application.

3.5.1 Mode de fonctionnement

Les données lues en mémoire centrale sont chargées dans le cache selon plusieurs stratégies. Dans le cas où le processeur charge en mémoire centrale une donnée avec toute les suivantes pour autant qu'il y ait de la place disponible, on parle de *cache en lecture* . Ici le processeur cherche ses données en priorité dans l'antémémoire.
Le processeur peut aussi enregistrer directement ses données dans l'antémémoire et faire des mises à jour de la mémoire centrale plus tard (transfert à intervalles réguliers, transfert après cumul d'une certaine quantité ou exploitation de la disponibilité des bus). Il s'agit de *cache en écriture* dont un inconvénient est l'existence de période critique où la mémoire centrale n'est pas à jour.
Lorsque la fonction de cache en écriture est interdite les données sont émises vers le cache qui les transfère aussi directement vers la mémoire centrale: on parle alors de *cache transparent en écriture* ou *cache en lecture seulement*.
On utilise la terminologie suivante:
Write-back: en écriture seulement,
Write-through: il est transparent en écriture, les informations le traversant librement sur la base d'une page (4 Ko) ou d'un bloc.

3.5.2 Les niveaux de cache

Les caches peuvent exister sur plusieurs niveaux. Souvent on distingue:
Un *cache primaire* ou cache de niveau 1 noté L1 implanté dans le microprocesseur. C'est le plus efficace mais sa capacité est forcement limitée (8 à 64 Ko).
Un *cache secondaire* ou cache de niveau 2 noté L2 extérieur au microprocesseur. Sa capacité est plus importante (> 100 Ko)



3.5.3 Caches unifiés et caches diversifiés

On peut concevoir deux architectures pour cache:

- Un seul cache qui regroupe instructions et données: on parle de cache unifié, c'est le plus simple à réaliser mais ce n'est pas la formule optimale.

- Deux caches spécialisés l'un pour les instructions ou *I cache* l'autre pour les données ou d cache: on parle caches diversifiés ou séparés ou architecture de havard. c'est plus difficile à réaliser mais plus efficace(élimination de l'une des causes d'embouteillages d'accès à l'antémémoire observés avec les caches unifiés).

3.5.4 Cohérence et MESI

Cache et mémoire centrale doivent constamment être mis à jour afin qu'il n'y ait ni divergence ni contradiction. Cette mise en harmonie relève de la cohérence. Elle est assurée sous commande du processeur et applique différentes stratégies dont la plus courante est basée sur le protocole MESI (Modified, Exclusive, Shared, Invalid) avec ses quatre états qui se traduisent par la position d'indicateurs spécifiques pour chaque ligne de donnée de l'antémémoire:
1 Modifié (modified) : le bloc dans le cache a été modifié par rapport au contenu de la mémoire. La donnée dans le cache est valide mais celle de la position mémoire correspondante est périmée;
2 Exclusif (Exclusive): la ligne de cache contient des données identiques à celles de la mémoire aux mêmes adresses. Aucun autre cache ne dispose de ces mêmes données;
3 Partagé (Shared): la ligne de cache contient des données identiques à celles de la mémoire aux mêmes adresses. Au moins un autre cache dispose des mêmes données;
4 Invalide (Invalid): la ligne du cache ne contient pas des données valides.

En tenant compte de la position des indicateurs, donnée après donnée, le processeur peut vérifier leur conformité et apporter les corrections nécessaires.
La vérification du cache par rapport au contenu de la mémoire centrale ou au contenu d'autres caches est garantie par la logique de Snooping, une logique de recherche.

3.5.5 Architecture des caches

On distingue plusieurs architectures de caches:
1 Cartographiée ou *à adressage direct (Direct Mapped Memory):* la mémore RAM est reproduite adresse pour adresse dans le cache à l'image de ce qui se passe dans la mémoire centrale (facile et rapide, pas efficace).
2 Associatif: les informations en provenance de la RAM centrale sont placées n'importe où dans le cache, sans adresse fixe. La recherche d'une donnée se fait par comparaison: le contrôleur de cache reçoit l'adresse demandée et vérifie si la même adresse ne se trouve pas dans le cache en les parcourant une à une (recherche de donnée très longue).
3 Associatif multigroupe ou *mutltivoie:* c'est une association des élément des deux précédentes méthodes, le cache est divisé en deux, quatre ou huit groupes dont chacun enregistre un bloc de données. Ainsi un bloc de la mémoire centrale peut se trouver dans n'importe quel bloc du cache.

Questions et réponses :

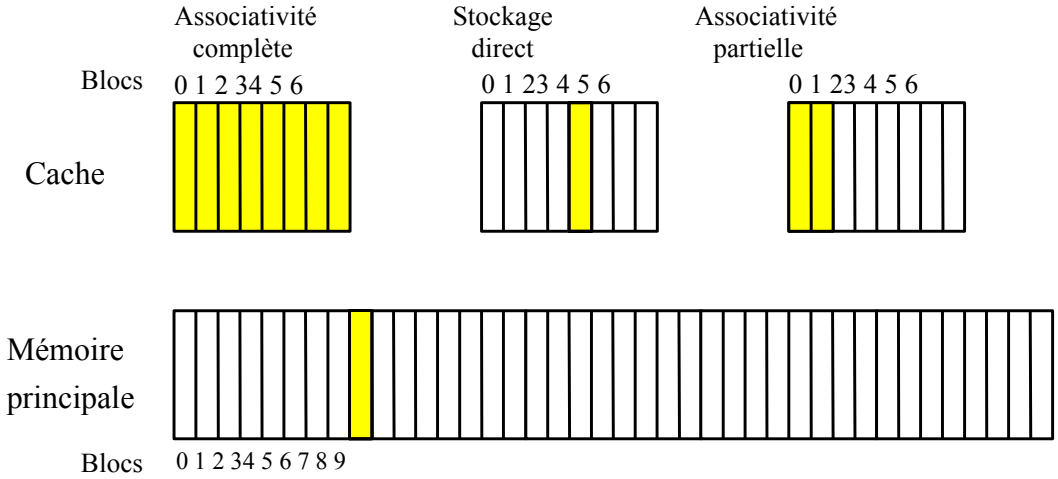
Question 1 : Où peut-on placer un bloc ?
Réponse 1 : Plusieurs possibilités :

1° stockage direct : le bloc n de la mémoire principale peut se retrouver seulement dans le bloc $m = (n \text{ modulo } s_b)$ de la mémoire cache, s_b étant la taille en nombre de blocs de la mémoire cache ;

2° tockage complètement associatif : un bloc quelconque de la mémoire principale peut se retrouver dans n'importe quel bloc de la mémoire cache ;

3° associativité partielle (*set associative*) : séparation de la mémoire cache en groupes de blocs et associativité complète dans un groupe, c'est-à-dire le bloc n de la mémoire principale peut se retrouver dans n'importe quel bloc du groupe $g = (n \text{ modulo } s_g)$ de la mémoire cache, s_g étant le nombre total de groupes de blocs dans la

mémoire cache.



Question 2 : Comment retrouve-t-on un bloc ?
Réponse 2 : Chaque bloc du cache a un descripteur associé, qui contient son numéro de bloc en mémoire principale.

1° Si le stockage est direct, l'adresse d'un bloc dans la cache est obtenue par la troncation de son numéro de bloc en mémoire principale. Le descripteur associé est ensuite comparé avec le numéro du bloc en mémoire principale ; si le résultat est négatif, le bloc recherché par l'UC est ramené de la mémoire principale (l'UC est mise en attente) ; si le résultat est positif, le transfert UC cache a lieu normalement.

2° Si le stockage est complètement associatif, la composante "numéro du bloc en mémoire principale" de l'adresse sortie par l'UC est comparée simultanément avec les descripteurs de tous les blocs de la cache ; si aucune coïncidence n'est obtenue, le bloc recherché est absent de la cache et doit donc être ramené de la mémoire principale.

3° Si le stockage est partiellement associatif, la combinaison correspondante des deux méthodes antérieures est employée.

Question 3 : Quel bloc doit être remplacé dans le cache en cas d'absence du bloc référencé (*miss*) ?
Response 3 : Si le stockage est direct, il n'y a pas d'alternatives. Si un degré d'associativité est présent, deux stratégies de base (résultats des statistiques meilleurs pour la deuxième stratégie, mais sensiblement proches) :

1° Choix aléatoire du bloc à remplacer.

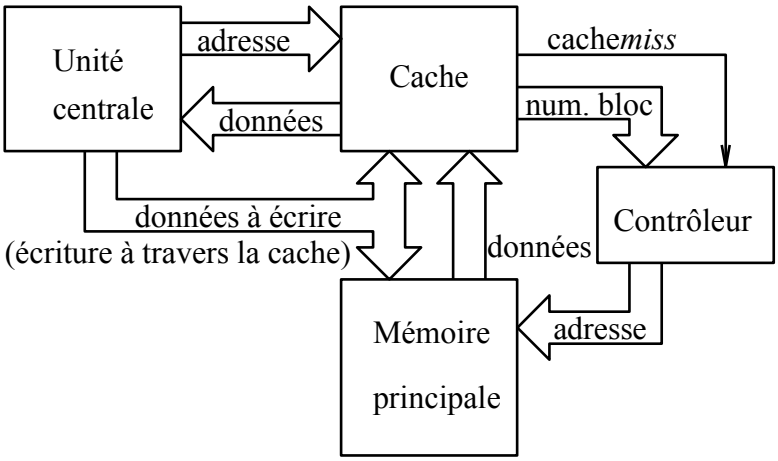
2° Remplacement du bloc le moins récemment utilisé (principes de localité).

Question 4 : Comment fonctionne l'écriture ?
Réponse 4 : Premier constat (statistique) : pour les UC de type RISC, les écritures en mémoire représentent environ 10% des accès mémoire.
Possibilités :

1° Ecriture à travers la cache : toute écriture est effectuée simultanément en mémoire cache et en mémoire principale ; le remplacement d'un bloc dans la cache ne produit pas d'écriture en mémoire principale.

2° Ecriture au remplacement du bloc : l'UC n'écrit que dans la cache, mais tout bloc se trouvant dans la cache et ayant été modifié par l'UC est réécrit en mémoire principale au moment où il doit être remplacé dans la cache par un autre bloc.

Problème principal : les dispositifs d'entrée/sortie effectuent des transferts directement avec la mémoire principale, donc des incohérences peuvent apparaître entre les informations gardées en mémoire cache et en mémoire principale. Les solutions combinent en général des composantes matérielles et logicielles.
Introduction de la mémoire cache dans un système :



Comment assurer un débit élevé pour les transferts cache mémoire centrale (condition nécessaire pour l'efficacité de la cache) :

1° Utilisation d'un bus de données de taille supérieure entre la cache et la mémoire principale (64 bits, 128 bits).

2° Séparation de la mémoire principale en bancs et multiplexage sur le bus de données des informations destinées aux bancs différents.

3° Utilisation de modes spécifiques d'accès pour les circuits de RAM dynamiques qui composent la mémoire principale (par ex., accès page : l'adresse de ligne est présentée une seule fois et ensuite plusieurs colonnes différentes sont lues successivement).

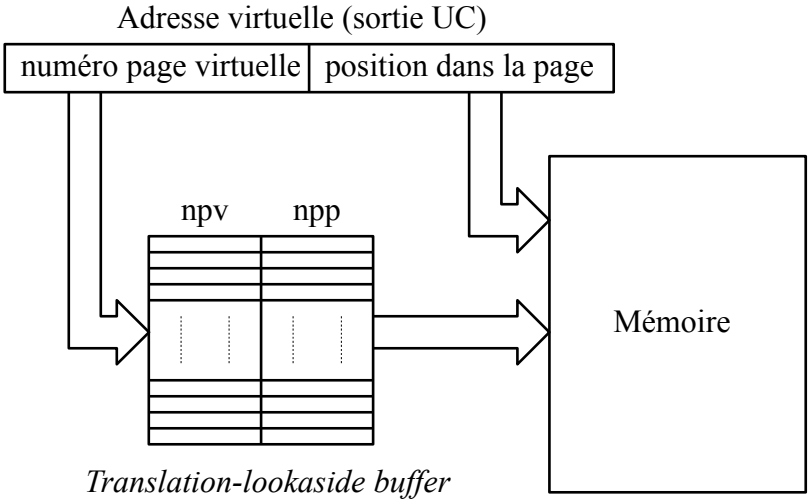
3.5.6 Remplacement du contenu de l'antémémoire

L'algorithme de remplacement des informations des antémémoires est le LRU (Least Recently used = la moins récemment utilisée). Selon cet algorithme la donnée la plus ancienne est supprimée du cache et remplacée par la plus récente lorsque le cache est plein.
On rencontre aussi un autre algorithme plus spécifique pour des actions prédictives. On utilise la notion de la données la plus récemment utilisée ou MRU (Most Recently Used).

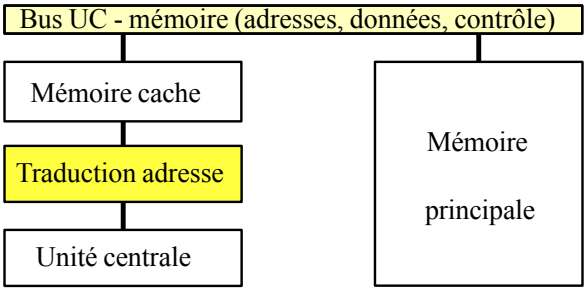
3.5.7 Mémoire principale et mémoire virtuelle

La mémoire principale permet d'adapter la vitesse de la mémoire de masse (disques magnétiques) à la vitesse de l'UC. Les blocs sont dans ce cas appelés couramment "pages".
L'adresse mémoire produite par l'unité centrale est appelée adresse virtuelle, et l'espace d'adressage correspondant "mémoire virtuelle". Une adresse reçue par la mémoire principale est appelée adresse physique, et l'espace d'adressage correspondant "espace physique". La mémoire principale (physique) a en général une taille nettement inférieure à la mémoire virtuelle. L'association (partie de) mémoire virtuelle espace physique est assurée par des mécanismes de gestion de la mémoire (appelés "gestion de la mémoire virtuelle"). La gestion de la mémoire virtuelle n'est pas assurée principalement par le *hardware*, comme pour la mémoire cache, mais principalement par des composantes du système d'exploitation.
Principe : l'espace virtuel et l'espace physique sont composés de "pages" (de même taille), à chaque instant un certain nombre de pages virtuelles correspondant au nombre de pages dans l'espace physique se trouvent en mémoire principale, les autres pages virtuelles étant seulement dans la mémoire de masse ; quand une nouvelle page virtuelle est nécessaire, elle est chargée de la mémoire de masse en mémoire principale ou elle remplace une autre page virtuelle.
Des mécanismes de protection entre programmes (et entre programmes et zones de données) sont rajoutés à la gestion de la mémoire virtuelle : chaque entrée dans la table de correspondances contient aussi une description des accès permis à cette page.
Les questions et les réponses :

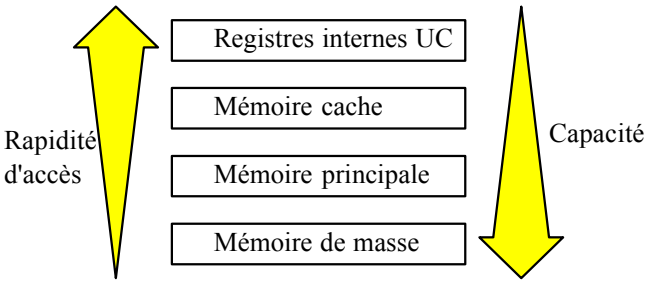
Question 1 : Où peut-on placer une page virtuelle en mémoire principale ?
Réponse 1 : Une page virtuelle peut être placée dans n'importe quelle page physique (stockage complètement associatif). Ce choix assure le nombre le plus bas de transferts de pages virtuelles entre la mémoire principale et la mémoire de masse (ces transferts étant extrêmement coûteux).
Question 2 : Comment retrouve-t-on une page virtuelle ?
Réponse 2 : Une table de correspondances est employée : le numéro de la page virtuelle (une partie de l'adresse produite par l'UC) est l'adresse dans la table, et à cette adresse on trouve le numéro de la page physique correspondante (ou l'indication qu'elle est absente de l'espace physique), ainsi que des information concernant la protection de la page. Si aucune page physique ne correspond à a la page virtuelle recherchée, cette page est chargée de la mémoire de masse. La table de correspondances a souvent une taille importante et seulement une partie (en général la plus récemment utilisée) se trouve, à un instant donné, dans un circuit de traduction rapide spécifique (*translation-lookaside buffer, TLB*), le reste étant gardé en mémoire.



Question 3 : Quelle page doit être remplacée en mémoire en cas d'absence de la page référencée (*page fault*) ?
Réponse 3 : En général c'est la page la moins récemment référencée (consistant avec les principes de localité), mais d'autres choix sont possibles et la stratégie choisie par le système d'exploitation peut dépendre du contexte.
Question 4 : Comment fonctionne l'écriture ?
Réponse 4 : En raison du coût excessif d'une écriture en mémoire de masse, tous les systèmes existants utilisent l'écriture au remplacement de la page : l'UC n'écrit que dans la mémoire principale, mais toute page modifiée par l'UC est réécrite en mémoire de masse au moment où elle doit être remplacée en mémoire principale par une autre page.
La traduction de l'adresse virtuelle en adresse physique a lieu *avant* l'entrée de l'adresse dans la mémoire cache !



3.5.8 Hiérarchie des mémoires



Hypothèses de localité :
1°*Localité temporelle* : un objet déjà référencé le sera à nouveau bientôt.
2°*Localité spatiale* : les objets proches d'un objet référencé seront référencés bientôt.
Ces hypothèses de localité sont confirmées par des statistiques de fonctionnement ; cela assure l'utilisation efficace de chaque niveau intermédiaire de la hiérarchie comme un tampon de taille limitée entre l'unité centrale et le niveau suivant, plus lent, de la hiérarchie.
Unité minimale d'information qui peut être soit présente soit absente (dans la mémoire cache ou dans la mémoire principale) = bloc. Toute adresse a donc deux composantes :

numéro du bloc	position dans le bloc

Questions à prendre en compte (pour la mémoire cache et pour la mémoire principale) :
1° Où peut-on placer un bloc ?
2° Comment retrouve-t-on un bloc ?
3° Quel bloc doit être remplacé en cas d'absence du bloc référencé (*miss*) ?
4° Comment fonctionne l'écriture ?

Caractéristiques* :

Composante Registres UC	Cache Mémoire	principale	Disques
Taille typique	< 1 Koctet	< 512 Koctets	< 1 Goctet	> 1 Goctet
Temps d'accès	*10 ns*	*20 ns*	*70 ns*	*20 000 000 ns*
Débit typique	800 Moctets/s	200 Moctets/s	133 Moctets/s	4 Moctets/s
Gérée par	Compilateurs	*Hardware*	Syst. exploitation	Syst. exploit.

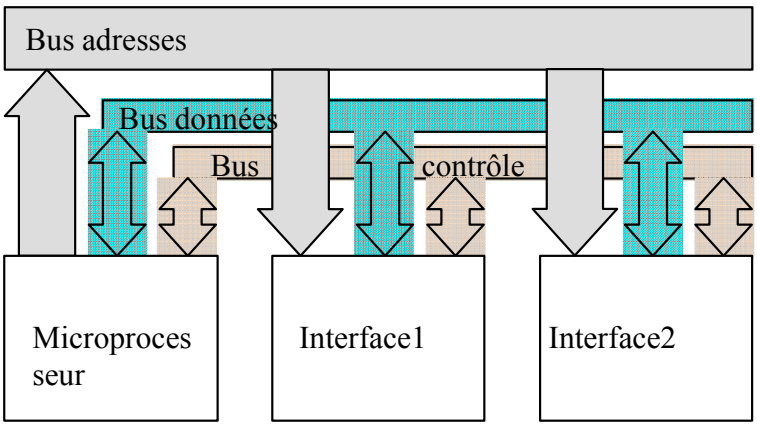
*année de référence 2000

CHAPITRE 4: BUS ET INTERFACES

Le microprocesseur dialogue avec les périphériques via des circuits dits d'interface, encore appelés entrées-sorties.
Chaque périphérique reçoit une adresse au même titre que les adresses des emplacements mémoire mais le microprocesseur ne doit pas le confondre avec ces derniers.

4.1 Distribution des interfaces

Le microprocesseur dialogue avec les périphériques par l'intermédiaire des circuits d'interface. Ces circuits peuvent se situés sur la carte système ou faire l'objet des cartes additionnelles enfichées dans les connecteurs d'extension.



Ces interfaces peuvent être assimilées à des "ports d'attache" pour des périphériques.
Les entrées sorties sont désignées par *E/S* ou *I/O*.
Chaque interface dispose d'adresses qui lui sont propres (une ou plusieurs selon qu'elle possède une ou plusieurs registres séparément).
L'adressage s'effectue selon deux stratégies : la méthode cartographique et l'adressage sur espaces distincts.
La méthode cartographique : on répartit les adresses disponibles entre les mémoires et les interfaces. Pour éviter les confusions on dresse une carte géographique montrant à qui les adresses appartiennent.
On peut remarquer qu'avec cette méthode

- le champ d'adressage total ne varie pas ;
- L'espace des adresses pour les mémoires est légèrement réduit (quelques dizaines d'adresses d'entrées/sorties dans un énorme espace d'adresses.) ;
- Toutes les instructions disponibles avec les mémoires sont également disponibles pour les entrées/sorties ;
- Le mot d'adresse reste long par rapport au nombre d'entrées/sorties ;
- Tous les modes d'adressage disponibles pour les mémoires sont aussi applicables.

Adressage sur espaces distincts : le microprocesseur dispose d'instructions spéciales pour les entrées/sorties différentes de celles qu'il utilise pour adresser la mémoire (IN pour une entrée et OUT pour une sortie). On peut donc utiliser les mêmes adresses sans risque de confusion car les espaces sont distincts. Les circuits décodeurs se chargent de bien aiguiller les adresses.

- La même adresse peut servir dans un cas, avec IN et OUT, aux commandes d'entrées/sorties et dans un autre cas, avec les autres instructions, à adresser les mémoires.
- L'espace d'adressage des mémoires n'est pas réduit
- Il n'est pas nécessaire d'établir la carte des ressources partagées des en adresses
- Seules les instructions IN et OUT sont utilisables directement avec les entrées/sorties
- Seuls les modes d'adressage direct et absolu sont disponibles

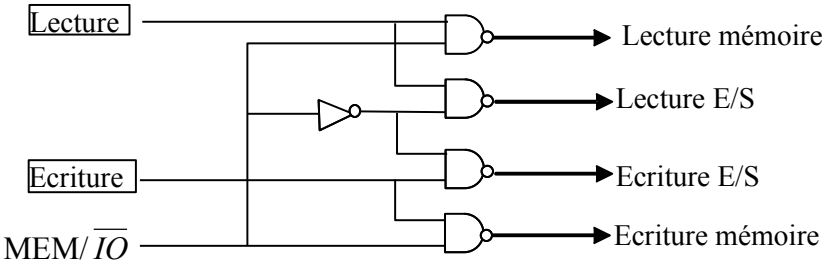
La longueur du mot d'adresse peut être réduite à un octet(moins de 256 E/S habituellement) ; les instructions peuvent être alors plus compactes, les programmes plus courts occupant moins de place en mémoire

Discrimination entre mémoire et entrées/sorties en mode d'espaces distincts.

Pour différencier les entrées/sorties des adresses mémoire le microprocesseur émet une commande spéciale MEM/ $\overline{IO}$.

Si MEM/ $\overline{IO}$ = 1 les mémoires sont adressées

Si MEM/ $\overline{IO}$ = 0 c'est les entrées/sorties qui sont adressées



Sélection mémoire ou E/S

4.2 Les bus d'extension du microprocesseur

Le microprocesseur communique avec ses périphériques par le biais des interfaces. Les cartes d'extension sont enfichées dans des connecteurs dont les normes ou les standards ont évolué avec la technologie.

Les principales normes sont :

Le bus ISA (Industrie Standard Architecture) ou bus du PC-AT

Le bus EISA (Extended ISA), un développement du précédent. Le bus EISA peut recevoir des cartes ISA. En voie d'abandon.

Le bus MCA (Micro Channel Architecture) de IBM : à l'abandon en raison du succès relatif.

Le bus local VESA(Video Electronic Standard Association) ou bus VLB(Vesa Local Bus)

4.2.1 Le bus ISA

Le bus ISA (Industries Standard Architecture), est le bus standard du PC-AT

Le bus ISA se compose de :

1 la section d'origine XT, sur 8 bits avec un bus de deux fois 31 broches soit 62 broches

2 son extension 16 bits avec un complément sur deux fois 18 contacts, soit 36 contacts

4.2.2 Les extensions PCMIA

Le connecteur PCMIA (*Personal Computer Memory International Association*) est un connecteur majoritairement externe et de très faible encombrement. La taille d'une carte ici est de la taille d'une carte de crédit (85,6 x 54 mm). Il sert aujourd'hui à connecter des cartes mémoires RAM, flash ou autres, des disques durs, des modems, des cartes réseau, etc.

On distingue trois types de carte PCMIA avec une même interface physique(connecteur 68 broches).

1 type I : épaisseur des cartes 3,3 mm

2 type II : épaisseur de 8 mm

3 Type III : épaisseur 10,5 mm

NB : Eviter des multiples insertions-extractions sous le risque de dégradation des contacts

4.2.3 Le bus PCI

Le bus PCI (Peripheral Component Interconnect) est un bus de 32 bits développé par Intel, de bande passante de 132 Mo/s à 33MHz

Indépendant du processeur, il intègre un buffer entre le processeur et les périphériques (contraire du bus VESA qui peut se connecter directement au processeur)

C'est un bus ouvert dont une extension devrait porter à 64 bits avec une bande passante de 264 Mo/s

4.2.4 Le bus SCSI (Small Computer System Interface)

C'est un bus intelligent multimaître , c'est-à-dire possédant son propre microprocesseur pour le gérer et pouvant relier jusqu'à 8 dispositifs (dont le micro-ordinateur lui-même. On ne peut donc relier que 7 autres dispositifs dont les disques durs, les imprimantes, les scanners, les lecteurs de CD-ROM, etc.) dont chacun peut jouer le rôle de maître.

Ce bus fonctionne aussi bien en *mode synchrone* (les octets sont expédiés en salve avec un débit de l'ordre de 4 Mo/s, un accusé de réception suivant chaque salve) qu'en *mode asynchrone* (chaque octet transféré est suivi d'un accusé de réception, débit moyen 1,5 Mo/s). Le câble SCSI de base comporte 50 lignes. Il doit être terminé à chaque extrémité par une terminaison présentée sous forme d'un bloc connecteur appelé "terminateur".
Types de connecteurs SCSI,

1 broches au pas de 2.54 mm
2 broches au pas de 2.16 mm (connecteur centronics 50 points, verrouillable et robuste)

Les variantes du bus SCSI

1 SCSI 1 : Norme de référence 8 bits, vitesse maximale de transmission des données 2,5 Mo/s en mode asynchrone et 5 Mo/s en mode synchrone, longueur maximale du câble 16 m; nombre d'unités connectées 8, largeur de connecteur 50 broches
2 SCSI 2 : 8/16/32 bits, disposant d'un jeu de commandes plus développé, il permet d'atteindre un débit de 40 Mo/s, nombre d'unités connectées 8, largeur de connecteur 50/68 broches
3 SCSI 3 : 8/16/32 bits, débit moyen 40 Mo/s, nombre d'unités connectées 8/16/32, largeur de connecteur 50/68 broches.
4 SCSI différentiel: utilisation des tension différentielles, ce qui fait une bonne immunité au bruit, la longueur de câble est portée à 25 m
5 Wide SCSI, largeur mot de données 8, 16 ou 32 bits, nombre de conducteurs du câbles 68. Vitesse de transmission 5 Mo/s en 8 bits, 10 Mo/s en 16 bits et 20 Mo/s en 32 bits.
6 Fast SCSI: la vitesse est doublée par rapport au SCSI, soit un vitesse de 10 Mo/s en mode synchrone.

4.2.5 Le bus IDE (Itegrated Drive Electronic)

C'est un bus spécialement conçu pour les disques durs. Le bus IDE est aussi connu sous le nom de ATA avec ses déclinaisons ATA1, ATA2 et ATA3 supportant les systèmes 32 bits.

4.2.6 Arbitrage des bus

La gestion des bus ou arbitrage des bus est assurée par du matériel, des circuits de logique spéciale connus sous le nom de ASIC (Application specific Integrated Circuits). L'arbitre de bus gère les niveaux de priorité pour résoudre d'éventuels conflits. Généralement des circuits de rafraîchissement des mémoires DRAM et d'accès direct en mémoire bénéficient des plus hautes priorités.

Bus Année		Données (bits)	Fréquence (MHz)	Débit Max (Mo/s)	Débit type (Mo/s)
PC 1981		8	8,33 8,33 0,5		
ISA 1985		16	8,33 16,66 2		
EISA 1988		32 8,33 33 8			
NuBus 1987		32 10 40 25			
1993		32	20	80 25	
MCA 1987		32 10 40 8			
1992		64	10	80	

1994		64	20	160	
VESA 1991		32 25-33	132 67		
1994		32	25-50	200	
1994		64	25-50	400	
PCI 1993		32	25-33	132 65	
1994		64	25-33	264	
Sbus		32			
		64			
Industrypack	1989	16	8 ou 32	16	8
	1994	32	8 ou 32	64	

CHAPITRE 5: INDICATEURS D'ETATS

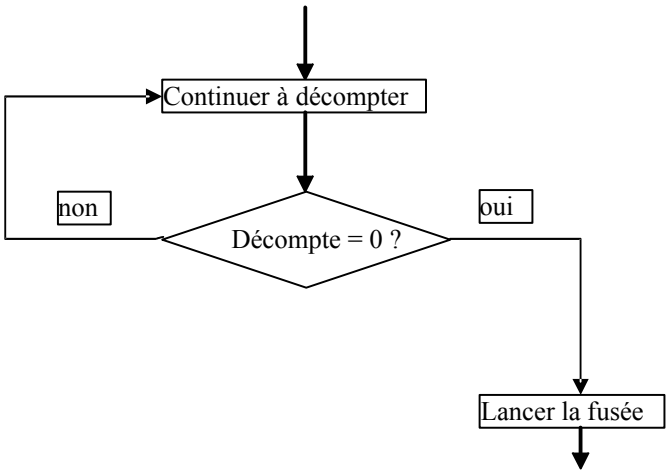
5.1 Indicateurs d'états et branchements conditionnels

Le microprocesseur incorpore un certain nombre de bits appelés indicateurs d'état (ou *flag* en anglais), regroupés en des registres appelés registres d'état , chargés de l'alerter lorsqu'une situation particulière s'établit.

Un indicateur est un bit qui reste au repos lorsque tout se passe bien et passe à l'état actif lorsque la situation qu'il est chargé de détecter surgit. Certains programmes (donc le programmeur) peuvent y passer des tests et en décider de ce qu'il y a lieu de faire: on parle de branchements conditionnels.

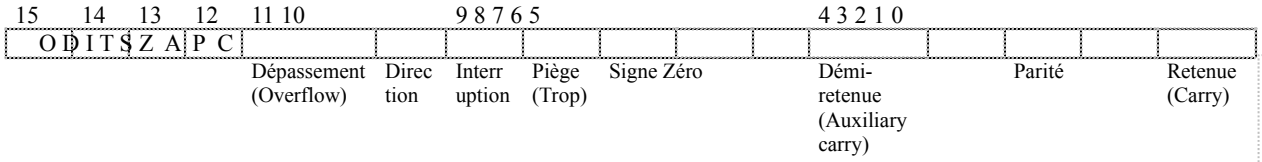
On dit qu'il y a branchement conditionnel lorsque la voie choisie dépend d'une condition (la position de l'indicateur d'état concerné.

Exemple: lancement d'une fusée

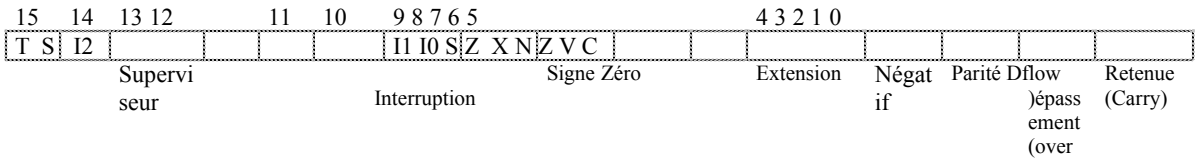


Un mot d'états regroupe plusieurs indicateurs d'états. Il est stocké dans un registre appelé registre d'états, registre de mot d'états, registre de conditions (ou CR = Condition Register), Program Status Word (PSW), Registre de contrôle (ou CR = Control Register) ou registre d'état machine

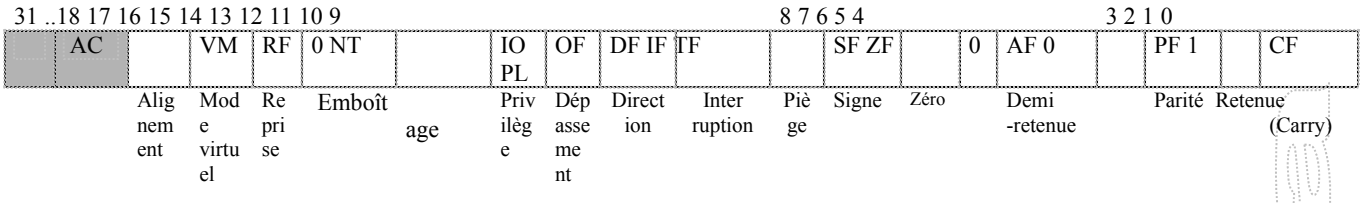
Exemple: Mot d'état du 8086/8088



Mot d'état du 68000



Mot d'état du 80486



5.2 Etude de quelques indicateurs d'états

5.2.1 Indicateur de zéro

Noté généralement Z, l'indicateur de zéro surveille le résultat d'opération arithmétiques et logiques principalement.
Il comporte deux états : un état de repos (0 par exemple) et un état d'indication du zéro d'un résultat d'opérations (1 par exemple)

5.2.2 Indicateur de retenue

Souvent noté par c, l'indicateur de retenue retient et contient lors d'une opération la retenue engendrée par les bits de poids le plus fort.
Les opérandes en double précision tiennent sur des mots de 2 octets. Leurs opérations d'addition se réalisent donc en deux temps dans les machines 8 bits :

- On additionne les octets de faible poids (LSB). S'il y a retenue elle positionne l'indicateur de retenue à 1
- On additionne les octets de fort poids (MSB), plus le contenu de l'indicateur de retenue.

Ce qui précède montre qu'il y a deux opérations d'addition, « avec prise en compte de la retenue » et « sans prise en compte de la retenue », réalisées par deux instructions bien distinctes.
Exemple de programme :

```
MOV A, 82H      ; introduire 1000 0010 dans l'accumulateur A
ADD A, 80H      ; ajouter 1000 0000 au contenu de l'accumulateur A sans retenue
MOV B, A        ; transférer le contenu de l'accumulateur A dans le registre B
MOV A, 2H       ; mettre 0000 0010  dans A
ADD A, 8H       ; ajouter 000 1000 au contenu de l'accumulateur A avec la retenue
                  précédente
```

Traduction

Retenue	1	
Cumulande	+ 0000 0010	1000 0010
Cumulateur	+ 0000 1000	1000 0000
Somme	= 0000 1011	0000 0010

5.2.3 Indicateur de signe

L'indicateur de signe intervient lorsqu'on travail en logique binaire signé en recopiant le bit de signe, représenté par le bit de poids le plus fort (bit 7 pour un octet et bit 15 pour un mot de 16 bits).

5.2.4 Indicateur de dépassement

L'indicateur de dépassement intervient pour indiquer si le résultat d'une opération nécessite plus de bit que le registre réservé à cet effet n'en dispose. On parle de dépassement de format ou de capacité. Ce qu'on remarque, en complément à deux, lorsque la somme de deux nombres positifs donne un nombre négatif ou la somme de deux nombres négatifs donne un nombre positif.

5.2.5 Indicateur de parité

L'indicateur de parité contrôle la parité ou l'imparité du mot reçu et se positionne en conséquence.

5.2.6 Indicateur de demi retenue

Lorsqu'on effectue des opérations en DCB trois types d'erreurs :

- un quartet peut générer un code non valide
- un quartet peut générer un report pour le quartet suivant
- le quartet de poids le plus fort peut générer une retenue

L'indicateur de demi-retenue ou indicateur de retenue auxiliaire est chargé détecter la propagation du quartet de faible poids vers le quartet de poids le quartet de fort poids.
La correction automatique de cette erreur est réalisée si on insère directement l'instruction "DAA" après l'instruction Addition "ADD"en langage assembleur.

```
Exemple :   MOV A, 9    ; mettre 1001 dans l'accumulateur
            ADD A, 9    ; additionner 1001 au contenu de A
            DAA         ; corriger immédiatement ce calcul qu'on veut en DCB
```

5.2.7 Indicateur d'interruption

Une interruption est un événement qui vient perturber le travail du microprocesseur. Elle l'interrompe pour lui faire exécuter une autre tâche urgente. Le microprocesseur fait appel à plusieurs indicateurs d'interruption.
Et lorsqu'un événement externe demande au processeur d'interrompre son travail pour exécuter une nouvelle tâche, le processeur va interroger ces indicateurs à fin de savoir s'il en a l'autorisation. C'est le programme qui décide à un moment ou à un autre de faire basculer ces indicateurs dans la position souhaitée.

5.2.8 Indicateur de pas à pas ou "Trap"

Lorsqu'on vient de rédiger un programme, ce programme peut comporter des erreurs appelées bogues. Le programme ne fonctionne pas alors comme il était prévu ou ne fonctionne pas du tout. Pour détecter les bogues, une méthode consiste à faire tourner le programme concerné instruction par instruction.
Il existe à cet effet un indicateur spécial appelé piège ou pas à pas (Trap, Step ou Single Step … en anglais)

5.2.9 Indicateur de reprise

Un point d'arrêt est une marque introduite dans le déroulement du programme pour le stopper. Le programme s'arrête chaque fois qu'il rencontre un point d'arrêt. Cela permet au programmeur d'examiner le déroulement du déroulement du programme et savoir si l'exécution jusqu'à ce point est conforme aux espérances ou non.
L'indicateur de reprise ou *resume flag* est testé par le programme avant que ce dernier n'exécute un point d'arrêt.
Cet indicateur permet de prendre en compte ou d'ignorer certaines erreurs.

5.2.10 Indicateur de direction

Permet de choisir les adresses croissantes ou les adresses décroissantes lors du transfert des octets en mémoire.

Indicateur de mode virtuel

Les microprocesseurs 8086/8088 ne savent travailler dans un mode dit *mode réel*, un processeur évolué d'intel peut émuler ce mode en restant dans son mode protégé, on parle de mode virtuel.
C'est un indicateur spécial, l'indicateur de mode virtuel qui autorise ou interdit ce mode de fonctionnement.

Chapitre 6 Programmation en langage assembleur

6.1 Introduction générale

L'assembleur permet de contrôler directement la CPU. Cela permet d'avoir une total maîtrise du système et surtout permet de faire des programmes rapides par rapport aux langages de haut niveau (C++, Basic, ...). En effet, bien que ces langages permettent de faire des programmes facilement et rapidement, ils n'optimisent pas le code d'exécution. Cela engendre donc des programmes (beaucoup) plus volumineux. Notons que l'on peut insérer de l'assembleur dans certain langage (Pascal et C par exemple) pour accélérer l'exécution du programme. Mais avant d'aller plus loin, rappelons brièvement la fonction de l'unité centrale (CPU).

Comme nous l'avons vu dans le précédent chapitre, la CPU (Central Processing Unit) charge, analyse et exécute les instructions présentes en mémoire de façon séquentielle, c'est-à-dire une instruction à la suite de l'autre. Elle contient une unité de calculs arithmétiques et logiques (UAL), d'un bus interne, d'un bus externe se connectant au système, d'un décodeur d'instructions qui décode l'instruction en cours, et des registres pour mémoriser des résultats temporairement.

Ce sont les registres qui permettent la communication entre le programme et la CPU. Ils sont 'l'interface' de la CPU. En effet, pratiquement, toutes les données qui passent par la CPU, pour être traitées par celle-ci, doivent se trouver dans les registres de cette dernière. Ces cases mémoire sont les plus rapides de tout le système.

Il existe différents types de registres dans une CPU: les registres de traitements, les registres d'adressages et les registres d'état. Les Registres de traitement sont des registres destinés au traitement des valeurs contenues dans celle-ci; par exemple on peut effectuer une addition d'un registre de traitement avec un autre registre, effectuer des multiplications ou des traitements logiques. Les Registres d'adressage permettent de pointer un endroit de la mémoire; ils sont utilisés pour lire ou écrire dans la mémoire. Les registres d'état (ou volet : FLAG en anglais) sont de petits registres (de 1 Bit) indiquant l'état du processeur et 'le résultat' de la dernière instruction exécutée. Les plus courants sont le Zero Flag (ZF) qui indique que le résultat de la dernière opération est égale a zéro (après une soustraction par exemple), le Carry Flag (CF) qui indique qu'il y a une retenue sur la dernière opération effectuée, Overflow Flag (OF) qui indique un dépassement de capacité de registre, etc.

Pour gérer ces registres, on fait appel aux instructions du processeur. Ces instructions permettent d'effectuer des tâches très simples sur les registres. Elles permettent de mettre des valeurs dans les registres, d'effectuer des traitements logiques, des traitements arithmétiques, des traitements de chaîne de caractères, etc.

Ces instructions sont formées à l'aide du code binaire dans le programme. Or pour nous, ils est beaucoup plus facile d'utiliser des symboles à la place de ces codes binaires. C'est pour cela que l'on fait appel au compilateur qui permet de transformer un programme assembleur fait avec des mots clés compréhensibles pour nous (mais incompréhensible pour la machine), en un programme exécutable compréhensible par le processeur.

Ces mots clés, ou mnémoniques, sont souvent la compression d'un mot ou d'une expression en anglais présentant l'action de l'instruction. Par exemple sur les processeurs 8086

l'instruction MUL permet la multiplication (MULtiply), sur Z80 l'instruction LD permet de charger une valeur dans un registre ou dans la mémoire (Load)

Les instructions sont souvent suivies d'opérandes permettant d'indiquer sur quel(s) registre(s) on veut effectuer le traitement, quelle valeur on veut utiliser.

Exemple: Pour additionner 2 registres (2 registres 16 bits AX et BX) sur 8086: ADD AX,BX
Cette instruction correspond en gros a: AX=AX+BX

La même chose sur 68000: ADD.W D1,D0
Cette instruction correspond en gros a D0=D0+D1

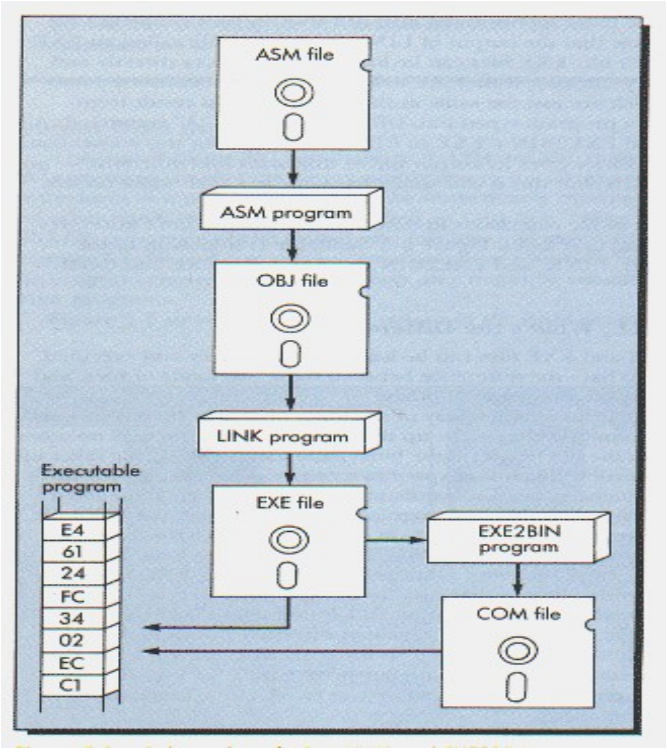
Sur cet exemple, nous pouvons remarquer la différence de syntaxe entre deux processeurs différents: lorsqu'un constructeur conçoit un processeur, il détermine aussi son assembleur. C'est pour cela que l'assembleur n'est pas universel car il est différent sur chaque processeur (ou plutôt concepteur). Et même si par hasard une instruction est identique entre 2 CPU différentes, elle ne sera pas compatible sur chaque processeur car le code binaire correspondant sera différent. Ceci est l'un des plus gros inconvénients de la programmation en assembleur car il faut alors reprogrammer de fond en comble le logiciel si on veut le porter sur un autre processeur. Chaque processeur comporte un nombre plus ou moins important d'instructions.

Dans ce chapitre, nous allons nous restreindre au langage assembleur du microprocesseur intel 8086.

6.2 L'assemblage

Un programme écrit en langage d'assemblage doit subir un certain nombre de transformations avant de pouvoir être exécuté. La figure suivante présente les différentes étapes du traitement d'un programme en langage d'assemblage.

La première phase de traitement est l'assemblage (TASM). Le programme source est traduit en langage machine et le programme objet ainsi obtenu est rangé dans un fichier. La seconde phase de traitement est l'édition des liens. La troisième phase du traitement est l'exécution du programme proprement dite qui peut utiliser des données et qui produit des résultats.



L'assembleur assigne aux instructions qu'il traduit des adresses dites relatives car elles sont attribuées de façon séquentielle à partir du début du programme.

6.3 Edition de liens

Le fichier .OBJ contient donc le produit binaire de l'assemblage. Mais ce fichier est inutilisable tel quel. Le DOS ne peut pas le charger et encore moins l'exécuter. Pour ce faire, nous avons encore deux étapes à franchir. La première est celle de l'édition de liens (LINK, c'est-à-dire "lier" en anglais).
L'utilitaire LINK ou TLINK continue le travail commencé par l'assembleur en lisant le fichier .OBJ pour créer un fichier "exécutable" (.EXE). On peut "lier" plusieurs fichiers .OBJ en un seul et unique fichier .EXE, et c'est pourquoi on le nomme "éditeur" de liens. On pourrait être plus précis en disant : un éditeur automatique de liens.

6.4 Chargement du programme

On ne peut exécuter un programme que s'il se trouve en mémoire centrale. L'assembleur range le programme objet en mémoire secondaire et l'éditeur de lien en fait un produit exécutable: mais avant de pouvoir exécuter ce programme, il faut le charger en mémoire centrale. Chaque instruction du programme possède une adresse relative qui lui a été attribuée par l'assembleur; il serait simple de placer le programme en mémoire de façon à ce que les adresses relatives correspondent aux adresses réelles des instructions (appelées adresses absolues). Ceci n'est pas possible car une partie de la mémoire centrale est occupée en permanence par le système d'exploitation; le programme ne pourra alors être placé en mémoire à partir de l'adresse zéro ou de l'adresse 100h. Si la zone de mémoire disponible pour le programme commence à l'adresse physique 2C8F0 (2C8F:0000), la première instruction du programme (d'adresse relative zéro) sera placée à l'adresse 2C8F0 pour un programme .exe et l'adresse 2C9F0 pour un programme.com (d'adresse relative 100h) et les instructions suivantes aux adresses qui suivent: on dit alors que le chargeur a translaté le programme.

6.5 Les registres

Un registre est un élément de mémoire interne du microprocesseur. Sa capacité est très réduite, son rôle n'est pas de stocker de données, mais plutôt des adresses ou les résultats intermédiaires de calculs. La plupart des commandes de l'assembleur utilisent les registres. Il est donc important de connaître leurs utilités respectives. Un registre n'est qu'une zone de stockage de chiffres binaires, à l'intérieur même du processeur. Le 8086 dispose d'un certain nombre de registres qui sont:

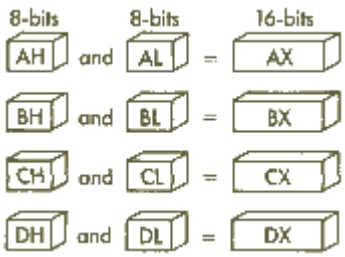
AX	accumulateur
BX, CX, DX	registres banalisés
DI, SI, BP	registres d'index
IP	pointeur d'instructions
SP	pointeur de pile
CS	registre de segment de code
DS	registre de segment de données
SS	registre de segment de pile
ES	extra segment
FLAGS	registre d'état.

Tous ces registres sont des registres 16 bits. Néanmoins les registres AX, BX, CX, DX peuvent être subdivisées en deux registres de huit bits supérieurs et inférieurs, et être référencés sous *x*H et *x*L: par exemple AH et AL, pour AX. En d'autre termes AX=256*AH+AL.

6.5.1 Registres de données

Il existe quatre registres de 16 bits : AX, BX, CX, DX. Ils servent pour le calcul et le stockage. Cependant, chacun d'eux peut être scindé en deux registres de 8 bits. L'octet de gauche d'un registre 16 bits est dit de poids fort, il est le plus significatif tandis que celui de droite (le moins significatif) est dit de poids faible. Ainsi, AX peut être décomposé en deux registres: AH et AL. II en va de même des registres BX, CX et DX. D'une manière générale, les programmes en assembleur placent dans ces registres des nombres issus de la mémoire, effectuent des opérations sur ces registres et replacent ensuite les résultats en mémoire.

Peu de programmes s'écartent de ce schéma de base. Quant aux registres autres que AX, BX, CX et DX, ils sont souvent utilisés pour les déplacements de nombres entre la mémoire et ces 4 registres. Ces registres sur 16 bits peuvent être utilisés sous la forme de 2 registres 8 bits chacun. Les registres 8 bits correspondant à AX, BX, CX et DX se nomment AL, AH, BL, BH, CL, CH, DL, DH.



Par conséquent, on peut inscrire une valeur quelconque sur 8 bits dans le registre BL, par exemple, sans que cela n'affecte le contenu du registre BH. Les registres de données sont en général interchangeables : malgré leurs fonctions, ils sont en effet d'un usage général. Cependant, on utilise souvent AX pour des opérations simples avec des instructions qui adressent implicitement ce registre sans qu'il soit nécessaire de le spécifier.

- AX est l'accumulateur;
- BX, le registre de " Base ", peut parfois être utilisé comme base d'indexage.
- CX est le registre " Compteur ". Vous vous rendrez compte que de nombreuses opérations de comptage sont effectuées directement avec CX par le processeur ;
- DX est le registre propre aux manipulations des " Données ". Dans les opérations sur 32 bits, avant l'apparition du processeur INTEL 386, les 16 bits de poids fort sont placés dans CX et ceux de poids faible, dans DX.

Ce sont donc les tâches spécifiques des registres CX, DX, AX et BX. Mais rappelons-le : ils sont interchangeables !

6.5.2 Registres d'index

Les registres d'index DI et SI et les registres "pointeurs de pile" (BP et SP) sont des registres spécifiques au langage assembleur.

- SI est le registre Source (Source Index) et DI le registre de destination (Destination Index) dans les opérations d'indexage.
- BP est le registre de base de la pile. C'est lui qui pointe sur l'adresse de base de la pile (nous verrons cela plus tard).
- SP (Stack pointer), est le pointeur de pile qui pointe sur le sommet de la pile. On dit de ces registres d'indexage qu'ils "pointent" sur un emplacement mémoire. Ainsi, si SI contient F000h, on dira qu'il pointe sur l'offset F000h.

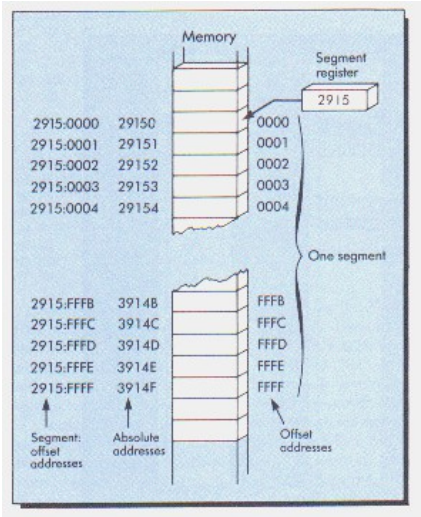
6.5.3 Registres de segment

Dans l'environnement du système d'exploitation DOS, nous devons donc être capables de représenter des nombres allant de 0 à 1 048 575. Pour générer les adresses nécessaires à une mémoire de 1 méga-octet il faut donc avoir 20 bits. Le processeur x86 utilise des mots de 20 bits pour adresser la mémoire réelle. Le jeu complet des 1 048 576 adresses différentes est appelé "espace adressable de 1 méga-octet". Cependant l'unité centrale du x86 ne comporte aucun registre de 20 bits puisque les registres sont de 16 bits. Un registre unique ne peut donc référencer que 64K de mémoire ($2^{16} = 65\,536$). II est donc nécessaire de combiner deux registres pour accéder à la mémoire. Le x86 en mode réel combine un des neuf registres généraux avec l'un des quatre registres de segments pour former une adresse de 20 bits. Les registres de segments sont appelés

- segment code (CS) ;
- segment données (DS) ;
- segment extra (ES) ;
- segment pile (SS).

À chaque type d'accès en mémoire correspond par défaut un registre de segment et parfois un registre général. Par exemple, CS et IP sont combinés pour accéder à la prochaine instruction à exécuter sous la forme CS :IP. SS et SP sont combinés en SS:SP pour toutes les opérations concernant la pile. II est parfois possible d'utiliser un registre de segment autre que celui utilisé par défaut en le spécifiant explicitement.

Si nous mettons bout à bout deux registres de 16 bits, nous obtenons une adresse sur 32 bits. Nous pourrions ainsi adresser 4 gigaoctets (2^{32} = 4 294 967 296). Pour adresser 1 méga-octet, nous n'avons besoin que de 4 bits en plus des 16 bits d'un registre général. Cette combinaison est obtenue en décalant le registre de segment de 4 bits et en ajoutant le résultat au contenu du registre général pour obtenir l'adresse sur 20 bits. Le résultat est appelé adresse effective (EA). L'adresse effective est placée sur le bus d'adresses afin de pouvoir accéder à l'emplacement mémoire correspondant. On peut donc considérer l'adresse effective comme constituée de deux parties. Le registre de segment définit une zone mémoire de 64K, et le registre général spécifie un déplacement à partir de l'origine de ce segment de 64K (c'est-à-dire une adresse sur 16 bits à l'intérieur de ce segment). L'adresse effective est exprimée en donnant le registre segment et le registre général séparés par deux points. Une adresse mémoire peut donc être symboliquement représentée par: DS:SI ou explicitement par : 2915:0004. Le segment de 64K est défini par le registre DS et le déplacement dans ce segment est contenu dans le registre SI. L'adresse du segment est 2915 et le déplacement dans le segment est 0004 en hexadécimal. L'adresse effective est donc 29154 hexadécimal. Notez que le décalage de l'adresse du segment est obtenu en ajoutant un zéro à droite. Ainsi, 2915:0004 devient 29150 + 0004, soit 29154, qui est l'adresse effective. Le même calcul se fait pour le cas de 2915:FFFB; 29150 + FFFB = 3914B.



6.5.4 Pointeur d'instructions

Le registre IP (Instruction pointer) est le pointeur d'instructions. Il indique la prochaine instruction à exécuter par le processeur. Dans certains autres processeurs, on l'appelle aussi le "Program Counter" - le compteur d'instruction de programme). Il a la responsabilité de guider le processeur lors de ses déplacements dans le programme en langage machine, instruction par instruction. Le registre IP est constamment modifié après l'exécution de chaque instruction

afin qu'il pointe sur l'instruction suivante. Le x86 dépend entièrement du registre IP pour connaître l'instruction suivante.

6.5.5 Registre Drapeau

Le registre Drapeau est un ensemble de 16 bits. La valeur représentée par ce nombre de 16 bits n'a aucune signification en tant qu'ensemble: ce registre est manipulé bit par bit, certains d'entre eux influenceront le comportement du programme. Les bits de ce registre sont appelés "indicateurs", on peut les regrouper en deux catégories:

6.6 Indicateurs d'état

Bit	signification	abréviation	TDEBUG
0	"Carry" ou Retenue	CF	c
2	Parité	PF	p
9	Retenue auxiliaire	AF	a
6	Zéro	ZF	z
7	Signe	SF	s
11	débordement (overflow)	OF	o

6.6.1 Indicateurs de contrôle

Bit	signification	abréviation	TDEBUG
8	trap	TF	
9	interruption	IF	i
10	direction	DF	d

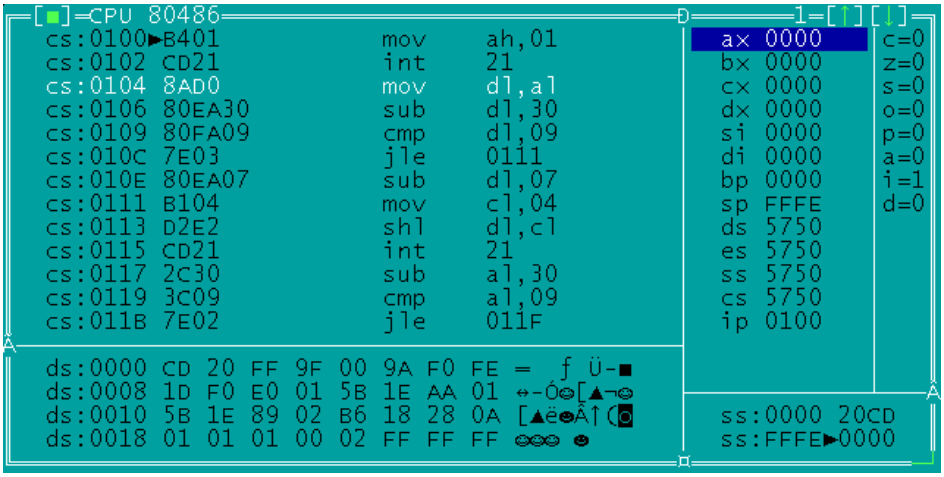
Les bits l, 5, 12, 13,14, IS de ce registre FLAG de 16 bits ne sont pas utilisés. Les indicateurs sont en relation directe avec certaines instructions: Les instructions arithmétiques, logiques et de comparaison modifient la valeur des indicateurs.

```
DEC AX        ; affecte les indicateurs  CF, ZF, SF, OF, AF
CMP AX,1      ; affecte les indicateurs  CF, ZF, SF, OF, AF
```

Les instructions conditionnelles testent la valeur des indicateurs et agissent en fonction du résultat.

```
JA    - teste les indicateurs CF et ZF
JNE   - teste l'indicateur ZF
JS    - teste l'indicateur SF
```

Lors d'une instruction non arithmétique ou logique, les indicateurs CF et OF sont remis à 0. Nous verrons que dans certains cas les indicateurs ne sont pas modifiés, dans certains cas ils le sont et, parfois, ils sont indéfinis...



Le registre Drapeau dans TDEBUG se situe en haut à droite des autres registres

6.6.2 Signification des principaux indicateurs:

L'indicateur CF (CARRY ou retenue): Il sera mis à 1 s'il y a eu retenue lors de la dernière instruction arithmétique.

L'indicateur OF (OVERFLOW ou débordement) : Il sera mis à 1 si le résultat d'une addition de 2 nombres positifs donne un nombre négatif et inversement. En règle générale, cet indicateur est activé quand le signe change alors qu'il ne devrait pas. En arithmétique non signée, la valeur de cet indicateur n'aura pas de signification.

L'indicateur ZF (ZERO): Il sera mis à 1 (activé, set) si le résultat d'une instruction arithmétique a donné zéro et il le restera jusqu'à la prochaine instruction arithmétique.

L'indicateur SF (SIGN ou signe) Il sera mis à 1 si le résultat d'une instruction a donné un nombre négatif (bit de poids fort =1), il sera mis à 0 dans le cas contraire.

L'indicateur DF (DIRECTION) Il est modifié par une instruction CLD ou STD et pas par le résultat d'une instruction (cf. manipulation de chaînes de caractères).

L'indicateur PF (PARITÉ) L'indicateur est mis à 1 si le résultat d'une opération contient un nombre pair de bits 1.

6.6.3 Comparaisons préalables au branchement

Pour effectuer un branchement, il faut dans bien des situations faire une comparaison. Il existe une seule instruction de cette catégorie permettant de comparer deux registres ou emplacements de mémoire: CMP . Les opérations de comparaison sont toujours suivies d'une instruction de branchement conditionnel car elles affectent les indicateurs du registre Drapeau. Le résultat de la comparaison est indiqué par les indicateurs. Les cinq formes disponibles de cette instruction sont:

CMP reg, imm	CMP AX, 0Ah
CMP reg, mem	CMP AX, [BX]

CMP mem, imm	CMP [BX], 0Ah
CMP mem, reg	CMP [BX], AX
CMP reg, reg	CMP AX, BX

"reg" étant un registre de 8 ou 16 bits (sauf un registre de segment);
"mem" étant une adresse (ou un identificateur);
"imm" est une valeur immédiate (constante).

Note: une opération de comparaison est en fait une soustraction qui n'affecte aucune opérande (cf. instructions arithmétiques):

CMP A,B <==> (A-B)

Dans le cas de comparaison de caractères, le 8086 effectue la soustraction des codes ASCII des caractères.

6.7 Conditions de branchements

Il existe deux catégories d'instructions pour tester les conditions de branchements. La première peut s'exercer sans le recours de comparaisons préalables. On peut employer JNZ après une comparaison mathématique du type DEC ou INC.

6.7.1 Premier groupe

JO 7O JUMP IF O=1 (si débordement) ; JNO 71 JUMP IF O=0 (si pas de débordement)
JC 72 JUMP IF C=1 (si "carry"); JNC 73 JUMP IF C=0 (si pas de carry)
JZ 74 JUMP IF Z=1 (si zéro); JNZ 75 JUMP IF Z=0 (si pas de zéro)
JS 78 JUMP IF S=1 (si signe) ; JNS 79 JUMP IF S=0 (si non signé)
JP 7A JUMP IF P=1 (si parité) ; JNP 7B JUMP IF P=0 (si pas de parité)

6.7.2 Deuxième groupe

JBE 76 JUMP IF(C=1) OR (Z=1); JA 77 JUMP IF(C=0) AND (Z=0)
JL 7C JUMP IF S <> 0; JGE 7D JUMP IF S = 0
JLE 7E JUMP IF((S XOR 0) OR Z)= 1; JG 7F JUMP IF((S XOR 0) OR Z)= 0

Le deuxième groupe comprend des tests constitués de combinaisons d'indicateurs que l'on utilise généralement en relation avec CMP (comparer) pour construire des boucles. Cette instruction de comparaison est capable de calculer des relations plus complexes entre deux valeurs : pour savoir si un nombre est inférieur ou égal à un autre, par exemple.

6.7.3 Codage de quelques instructions :

Regardons maintenant le code assembleur nécessaire pour exécuter quelques instructions fréquentes.

IF THEN ELSE ... En assembleur, on obtient :

```
Si (ax==1)              If:   CMP AX, 1
    bx = 10;                  JNZ   Else
```

```
Sinon {                          Then:    MOV BX,10
        bx = 0;                           JMP   EndIf
        cx  =10;                  Else:   MOV BX,0
      }                                   MOV CX,10
EndIf:
```

La boucle FOR ...

```
 bx = 0 ;                                MOV BX, 0
 Pour K = 0 jusqu'à 10                  MOV CX,0
     bx  = bx + K ;                For:  CMP CX,10
                                         JA EndFor
                                         ADD BX, CX
                                         INC CX
                                         JMP For
                                    EndFor:
```

La boucle while .

```
   Bx = 5 ;                              MOV BX,5
   Tant que bx > 0                 While :  CMP BX,0
      faire bx = bx-1                       JLE EndWhile
                                            DEC BX
                                            JMP While
                                   EndWhile:
```

La boucle repeat

```
 bx =  10 ;                                MOV BX,10
 Répéter                            Repeat1:   DEC BX
    bx = bx – 1 ;                             CMP BX,0
 jusqu'à bx <= 0                              JG Repeat1
                                    Endrepeat1 :
```

Si le nombre de répétitions est connu et au moins égal à 1, on pourra procéder comme suit:

```
            MOV BX, 0
            MOV CX, 0Ah
Repeat2:
             ADD BX, CX
             LOOP Repeat2; LOOP décrémente automatiquement CX jusqu'à atteindre 0
EndRepeat2:
```

Le test à plusieurs alternatives...

```
Switch(BX){                              CMP BX,1
 case 1: ax =1 ; break;                  JNZ case2
 case  2: ax = 5 ; break ;               MOV AX,1
 case  3: ax  = 10; break;               JMP endswitch
 default: ax = 0;                  case2: CMP BX,2
 }                                        JNZ case3
                                          MOV AX,5
                                          JMP endswitch
                                   case 3: CMP BX, 3
```

```
        JNZ default
        MOV AX,10
        JMP endswitch
default:
        MOV AX,0
endswitch:
```

6.8 Le compteur d'instructions (IP)

Puisque le pointeur d'instruction est le registre servant à exécuter les instructions, il est nécessairement modifié par un saut conditionnel. Si la condition du test est remplie, on place une nouvelle valeur dans le compteur du programme. Si, par contre, les conditions du test ne sont pas remplis (comme dans le cas d'une instruction JNZ (Jump if Not Zero) avec l'indicateur Z activé), alors le pointeur d'instruction suit son cours normalement, et le programme se déroule comme si aucun saut conditionnel n'avait été rencontré. Si les conditions du test sont remplies, l'octet de données (c'est-à-dire l'octet qui suit l'instruction de test) est additionné l'IP (Instruction Pointer). Par exemple, si l'IP contient 102h, si le processeur se rend compte que l'indicateur Z est désactivé (en lisant l'octet complémentaire, disons 10h, de JNZ), la nouvelle valeur de l'IP après une instruction JNZ 10 sera le résultat l'addition : 10h plus 102h. L'instruction qui sera exécutée immédiatement après le JNZ est celle qui se trouve à l'adresse 122h (102h+10h).

6.9 Structure d'un programme assembleur

Comme dans tout programme le fichier source doit être saisi de manière rigoureuse. Chaque définition et chaque instruction doivent ainsi s'écrire sur une nouvelle ligne (pour que l'assembleur puisse différencier les différentes instructions) Le fichier source contient:

1. Un nom du programme sous TITLE sui vie d'un nom. Cette partie est facultative.
2. Une partie pour déclarer une pile qui est définie dans le segment de pile délimité par les directives SEGMENT STACK et ENDS
3. Des définitions de données déclarées par des directives. Celles-ci sont regroupées dans le segment de données délimité par les directives SEGMENT et ENDS
4. Puis sont placées les instructions (qui sont en quelque sorte le coeur du programme), la première devant être précédée d'une étiquette, c'est-à-dire par un nom qu'on lui donne. Celles-ci sont regroupées dans le segment d'instructions délimité par les directives SEGMENT et ENDS
5. Enfin, le fichier doit être terminé par la directive END suivi du nom de l'étiquette de la première instruction (pour permettre au compilateur de connaître la première instruction à exécuter (Les points-virgules marquent le début des commentaires, c'est-à-dire que tous les caractères situés à droite d'un point virgule seront ignorés) Voici à quoi ressemble un fichier source (fichier .ASM):

```
TITLE  nomprogramme     ;cette directive permet de nommer votre programme

Pile     SEGMENT  STACK; segment de pile dont le nom est Pile (ou tout autre nom)

        ; déclarer la pile et sa taille

 Pile   ENDS
```

```
Donnees  SEGMENT; voici le segment de données dont l'étiquette est Donnees
                 ; (ou tout autre nom)

        ;Placez ici les déclarations de données

Donnees         ENDS; ici se termine le segment de donnees

Lecode   SEGMENT  ; voici le segment d'instructions dont l'étiquette est Lecode
                  ; (ou tout autre nom)

          ASSUME DS:donnee, CS: Lecode


          debut:        ; placez ici votre première instruction (son étiquette est nommée debut)

                        ; placez ici le reste de vos instructions

Lecode   ENDS; fin du segment d'instructions

          END debut; fin du programme suivie de l'étiquette de la première instruction
```

6.10 Déclaration d'un segment

Pour l'instant, oublions le segment de pile. Les données sont regroupées dans une zone de la mémoire appelée segment de données, tandis que les instructions se situent dans un autre segment qui est le segment d'instructions. Le registre DS (Data Segment) contient le segment de données, tandis que le registre CS (Code Segment) contient le segment d'instructions. C'est la directive ASSUME qui permet d'indiquer à l'assembleur où se situe le segment de données et le segment de code. Puis il s'agit d'initialiser le segment de données:

```
MOV AX, nom_du_segment_de_donnees
MOV DS, AX
```

On fera pareil quand viendra le temps où le segment de pile va être utilisé.

Bibliographie

[1] Ronald J. Tocci *Circuits numériques : théorie et applications* 2ème édition Dunod 1992

[2] Alfred Strohmeier *Le matériel informatique : concepts et principes* Presses polytechniques romandes 1994

[3] Andrew Tanenbaum *Architecture de l'ordinateur* 3ème édition InterEditions 1996