

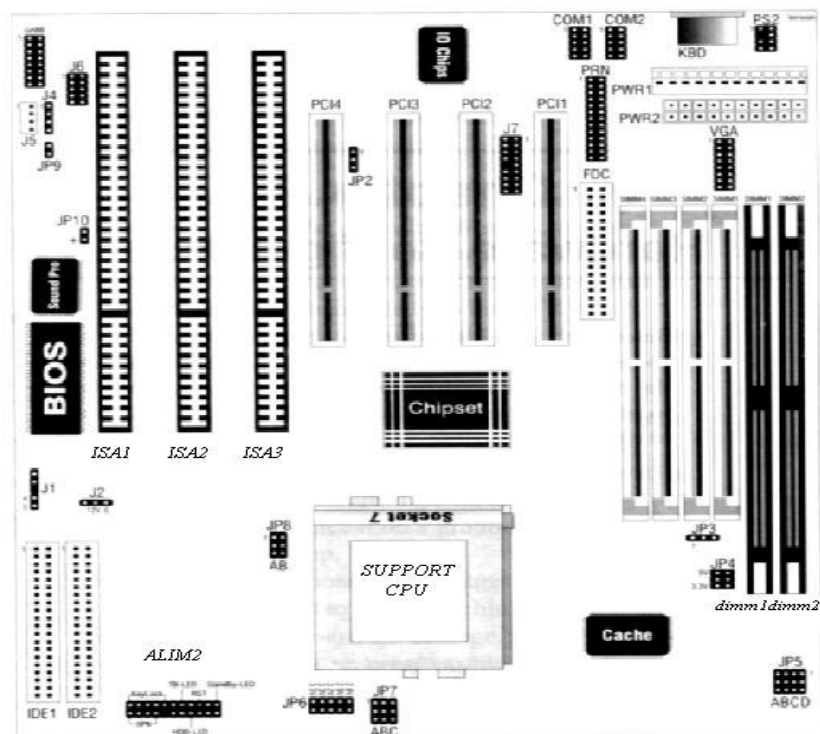
HAMDI HOCINE

NOTIONS DE STRUCTURE MACHINE

COURS & EXERCICES CORRIGES

NOTIONS DE STRUCTURE MACHINE

COURS & EXERCICES CORRIGES



Les éditions de l'université Mentouri Constantine

- Avant Propos -

Cet ouvrage est destiné en premier lieu aux élèves techniciens supérieurs des écoles publiques ou privées (universités, centres de formation professionnelle, écoles privées d'informatique), et en second lieu aux élèves ingénieurs (grandes écoles et universités). Il peut être également utilisé avec profit par toutes les personnes non spécialistes, souhaitant entreprendre une action de formation continue.

Il a pour but de leur donner les bases pour la compréhension du fonctionnement interne d'un ordinateur, en vue de son exploitation et de sa maintenance éventuelle.

Ce que nous proposons n'est pas un cours classique. En effet, la compréhension de la structure interne d'un ordinateur nécessite des connaissances aussi bien en informatique qu'en électronique. Si cela ne pose pas de problèmes aux élèves ingénieurs (compte tenu de leurs acquis antérieurs à l'étude de cette matière), par contre les élèves techniciens supérieurs (en première année informatique) n'ont pas le minimum (ni en électronique ni en informatique) facilitant la compréhension. C'est pour cela que nous avons essayé de simplifier au maximum, en exploitant surtout le volet pratique avec un recours minimum à la théorie. Ainsi certains paragraphes qui sont normalement de nature très technique, ont été présentés sous un aspect descriptif avec des exemples pratiques. C'est pour cela qu'au lieu de détailler « technologiquement » la constitution des composants, nous avons préféré nous limiter à un aspect purement fonctionnel tout en donnant les principales caractéristiques avec des ordres de grandeur des valeurs utilisées dans la pratique. Nous espérons ainsi toucher un large public, et que ce document serve de référence à consulter en cas de besoin.

Cet ouvrage est-il la traduction fidèle de nos intentions pédagogiques ? Nous le souhaitons vivement, et remercions par avance toutes les personnes qui auront l'amabilité de nous faire part de leurs critiques ou de leurs conseils, afin d'en améliorer le contenu dans les prochaines éditions.

Dr Hamdi Hocine

*Chargé de cours Au département d'Electronique de l'Université de
Constantine*

- Table des matières - PARTIE 1 : COURS

CHAPITRE I : STRUCTURE D'UN ORDINATEUR	PAGE
-1- Schéma fonctionnel de base de l'ordinateur	1
-2- Les périphériques	2
-3- Interfaces d'entrée et de sortie (e/s)	
-4- Les mémoires	3
-5- Les bus	5
-6- L'unité centrale de traitement ou cpu	7
-7- Le système d'exploitation	
-8- Horloges et alimentations	8
-9- Notions de bus système et bus local	
-10- Notion de carte mère (pour pc)	9
 CHAPITRE II : REPRESENTATION DES INFORMATIONS EN MEMOIRE ET CODAGE	
-1- Rappels sur les systèmes de numération	
-1-Principe	12
-2- Changement de base : nombres entiers et nombres fractionnaires	
-2- Représentation ou codage binaire des nombres en mémoire	15
-1-Nombres entiers : codage des nombres négatifs en complément à 2	
-2-Nombres fractionnaires : techniques de la virgule fixe et de la virgule flottante	17
-3- Codage des caractères	18
-1-Code CCITT n° 2 (code Baudot) ou code télégraphique	
-2-Code CCITT n°5 (ou code ISO) et code ASCII	20
-3-Code EBCDIC	22
 CHAPITRE III : LES MEMOIRES	
-1- La mémoire centrale	
-1-Définitions	25
-2-Principales caractéristiques	
-3-Espace adressable	29
-4-Hiérarchie de mémoire	32
-5-Réalisation physique de mémoire	35
-2- Les mémoires de masse	
-1-Le disque souple ou disquette (floppy disk)	37
-2-Le disque dur (hard disk)	41
-3-Autres procédés de stockage	44
 CHAPITRE IV : LE PROCESSEUR OU CPU	
-1- Architecture standard d'un CPU	
-1- Notion d'architecture	58
-2-Principe du processeur	
-3- Structure et organisation interne du processeur	59
-4- Les registres	61
-5-Les signaux d'entrée sortie du processeur	64
-2- Fonctionnement du processeur	
-1-Structure d'instruction et adresse mémoire	66
-2-Notions de cycle et état	68
-3- Le logiciel du cpu	
-1- Pile et sous-programme	70
-2-Les indicateurs et leur utilisation	74
-3-Modes d'adressage	75

- Table des matières - PARTIE 2 : EXERCICES CORRIGES

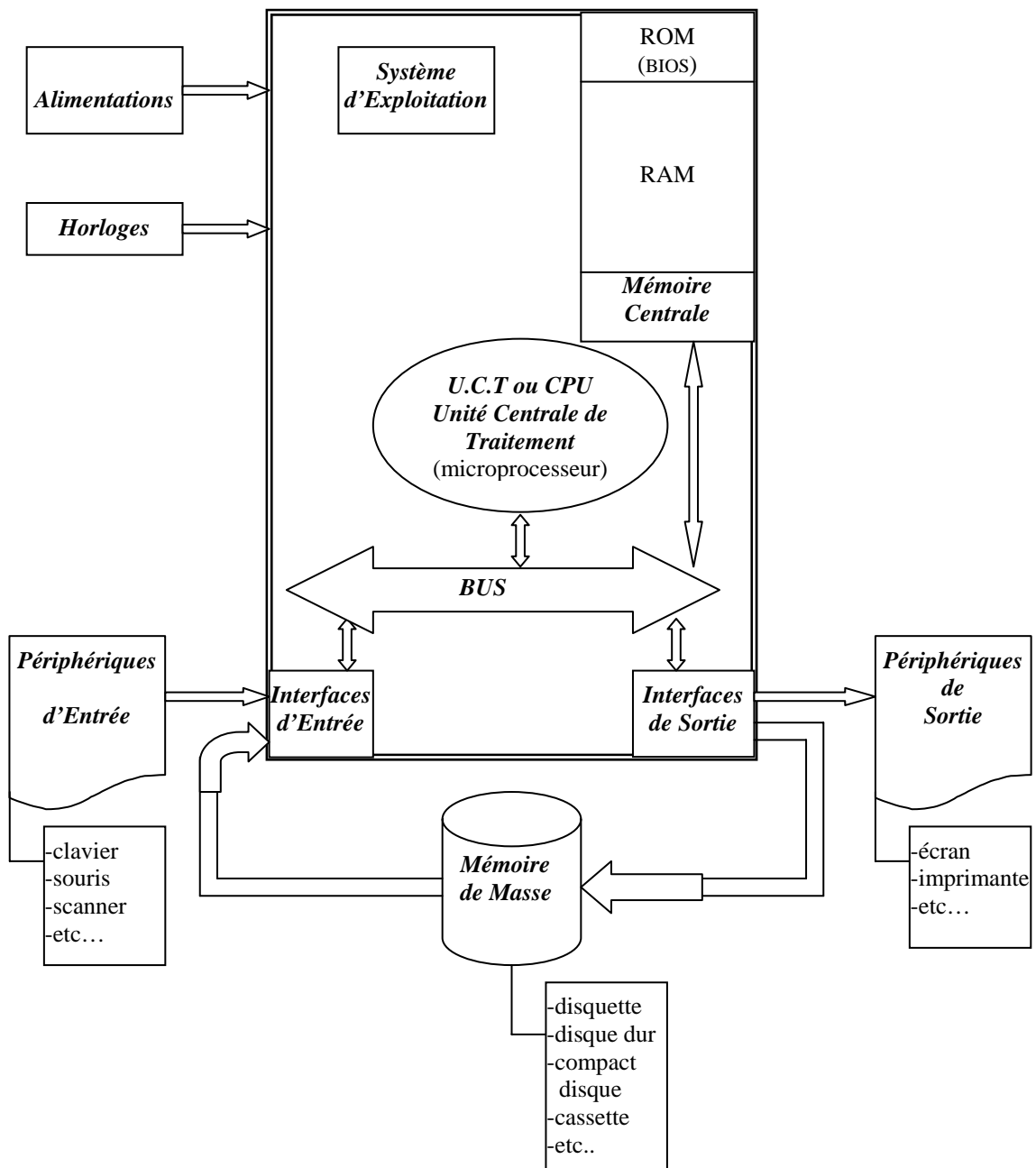
CHAPITRE 1 : LOGICIELS ET ENVIRONNEMENT DE PROGRAMMATION	PAGE
-I-Terminologie de base	83
Informatique, hardware et software, système d'exploitation, processeur, carte mère, bus, entrées sorties, interfaces.	
-II-Langages et environnement de programmation	85
Les langages, les traducteurs, environnement de programmation.	
-III-Les logiciels	61
Editeurs, traitements de texte, tableurs, gestionnaires de bases de données, ...	
CHAPITRE 2 : REPRESENTATION DES INFORMATIONS EN MEMOIRE ET CODAGE	
Ex 1 & 2 : Rappel sur les systèmes de numération et conversions	94
Rappels de cours (notions de base, bit, quartet, octet, mot) ; codage, décodage, transcodage de nombres entiers et fractionnaires	
Ex 3 & 4 : Représentation des nombres en mémoire	99
Codage des entiers naturels, des entiers relatifs, complément à 2, nombres fractionnaires en virgule fixe, nombres fractionnaires en virgule flottante	
Ex 5, 6, 7 : Opérations élémentaires en arithmétique non signée	101
Additions binaire, hexadécimale, BCD ; soustraction en complément à 2 ; multiplication et division binaires.	
Ex 8 & 9 : Codage, transmission et impression	104
Codage de messages en code baudot, iso, ascii, ebcdic ; transmission : bits de parité, start, stop ; impression : caractères imprimables et caractères de contrôle.	
CHAPITRE 3 : LES MEMOIRES	
Ex 1, 2, 3 : Bus d'adresses, bus de données, taille de l'espace adressable	107
Ex 4 : Stockage de nombres et de caractères	108
Ex 5 & 6 : Mémoire organisée en octets et mémoire organisée en mots, techniques de stockage « big endian » et « little endian »	108
Ex 7 : Mémoire segmentée	110
Ex 8 & 9 : Réalisation physique de mémoires	110
Ex 10 : Synthèse des notions précédentes	115
Ex 11&12&13&14 : Antémémoire, mémoire paginée, et mémoire virtuelle	118
Ex 15 & 16 : Unité de disque dur : pistes, secteurs, cylindre, capacité, taux de transfert	121
CHAPITRE 4 : LE PROCESSEUR OU CPU	
Partie 1 : Rappels de cours	125
Ex1 à Ex5 : Notions de CPU, UC, UAL ; Registres PC, ACC, RI, PSW, SP	
Partie 2 : Exercices	
Ex6 : Adresses symboliques et adresses physiques, langage assembleur de mimosa	127
Ex 7 & 8 : Assemblage et désassemblage	
Ex9 : Programmation en langage assembleur	129
Ex10 : Fetch et exécution	
Ex11 : Exercice de synthèse : assembleur, fetch et exécution	
Ex12 : Pile et sous programme	132
Ex13 : Utilisation des bits ou indicateurs zéro (Z), carry (C), signe (N)	133
Ex14 : Algorithmie et programmation en assembleur	135
Partie 3 : Contrôle des connaissances	
Ex3-1 : Adressage relatif sur Mimosa	137
Ex3-2 : Pile et sous programme (mémoire organisée en octets)	
Ex3-3 : Désassemblage et exécution de programme.	138
Ex3-4 : Programmation en assembleur et assemblage	139

CHAPITRE I: STRUCTURE D'UN ORDINATEUR

Un ordinateur est une machine de traitement de l'information dans la logique binaire (celle de l'algèbre de BOOLE). L'unité élémentaire d'information est le bit qui peut prendre pour valeurs 0 ou 1.

Le traitement de l'information se fait par une unité de calcul, d'où le nom de *calculateur* communément attribué à l'ordinateur.

-I- SCHEMA FONCTIONNEL DE BASE DE L'ORDINATEUR



Eléments de la structure de base simplifiée d'un ordinateur

-II-LES PERIPHERIQUES

On désigne sous le terme de périphérique tout élément situé à la périphérie de l'ordinateur, qui n'est pas nécessaire à son fonctionnement, mais dont l'utilisateur a besoin pour communiquer avec l'ordinateur.

On distingue deux catégories de périphériques : les périphériques d'entrée et les périphériques de sortie. Les termes entrée et sortie désignent respectivement un transfert d'information de l'extérieur vers l'ordinateur (éventuellement le processeur) ou de l'ordinateur vers l'extérieur.

Parmi les périphériques d'entrée on peut citer le clavier, la souris, le crayon optique, le scanner etc... Le périphérique standard d'entrée (stdin) est le clavier.

Parmi les périphériques de sortie on peut citer l'écran et l'imprimante. Le périphérique standard de sortie (stdout) est l'écran.

-III-INTERFACES D'ENTRÉE ET DE SORTIE (E/S)

On les appelle également coupleurs ou organes d'entrée-sortie. Ce terme d'interface (masculin ou féminin) désigne un dispositif (matériel ou logiciel) qui permet à deux entités (matérielles ou logicielles) de communiquer entre elles. Ils permettent de faire de l'adaptation d'information, qu'elle porte sur la forme (transformation des bits disposés en parallèle vers des bits disposés en série ou l'inverse), sur le code (transcodage) ou sur la vitesse (cas du transfert vers une imprimante par exemple).

Les interfaces d'E/S les plus courantes sont les interfaces *série* (ports *COM*) et *parallèle* (ports *LPT* : *LPT1=PRN* pour le premier port d'imprimante sous DOS). L'interface série a tendance à être remplacée par le port *USB* (Universal Synchronous Bus) qui est beaucoup plus rapide mais dont l'utilisation n'est pas encore généralisée.

-IV-LES MEMOIRES

On distingue différentes catégories de mémoire, qui sont appelées mémoires primaire ou principale et secondaire ou auxiliaire. La mémoire principale correspond à la mémoire centrale, et la mémoire auxiliaire correspond à la mémoire cache et aux mémoires de masse.

-1-La mémoire centrale

Elle est constituée de la RAM et de la ROM.

1-a- RAM (random access memory)

La ram ou mémoire à accès aléatoire, est appelée ainsi car l'accès se fait en *lecture et écriture*. Elle est également appelée *mémoire vive* car les informations qui s'y trouvent ont besoin d'alimentation pour "vivre": dès qu'on coupe l'alimentation, le contenu de cette mémoire s'évapore (mémoire volatile).

Par conséquent c'est une zone de *travail* et de *stockage temporaire* de l'information pour l'utilisateur et le système d'exploitation. Pour ne pas perdre les données, stockées temporairement en mémoire ram, il faut effectuer une *sauvegarde* en mémoire de masse.

La taille de la ram s'exprime généralement en mégaoctets, un octet correspondant au regroupement de 8 bits.

Remarque:

La *ram-cmos* est une petite partie de mémoire ram indépendante de la zone de travail, et fabriquée dans la technologie cmos. A l'extinction de l'ordinateur, son rôle est de conserver les informations sur la configuration de l'ordinateur (périphériques disponibles, interfaces etc..), ainsi que sur l'environnement de travail de l'utilisateur: clavier français (azerty) ou anglais (qwerty), date dans le format français ou anglais, etc...Son contenu est sauvegardé en permanence par une pile ou un accumulateur.

1-b- ROM (read only memory)

La rom est une mémoire où l'accès s'effectue *uniquement en lecture*. Les informations qui sont contenues dans cette mémoire y ont été écrites par le constructeur de l'ordinateur. On y trouve généralement quelques programmes de gestion de l'ordinateur, dont une petite partie du système d'exploitation appelée BIOS (Basical Input Output System).

Généralement l'utilisateur n'a pas besoin d'accéder à la rom. C'est pourquoi dans le langage parlé quand on parle de mémoire, on sous entend la mémoire ram.

-2-Mémoires auxiliaires

On classe dans cette catégorie tout ce qui est en plus de la mémoire centrale (ou principale). On distingue deux types: la mémoire cache et les mémoires de masse.

-2-a-Les mémoires de masse

Comme la mémoire ram, l'accès à la mémoire de masse se fait en lecture et en écriture. Mais contrairement à la ram, c'est une unité de *sauvegarde* donc de stockage permanent. Cette mémoire est beaucoup moins rapide que la ram.

Les unités de sauvegarde les plus utilisées sont le lecteur de disque souple ou disquette (de taille standard 1,44 mégaoctets), le lecteur de disque dur (constitué de plateaux d'aluminium) dont la taille s'exprime en gigaoctets, le lecteur de disque compact ou disque optique numérique (écriture par rayon laser et non par procédé magnétique) de taille standard 650 mégaoctets, le lecteur de cartouches magnétiques de tailles variées: 1 Mo, 2 Mo, 120 Mo.

-2-b-La mémoire cache

Comme la mémoire ram est beaucoup moins rapide que le processeur, elle le ralentit dans ses traitements. De plus la mémoire de masse étant encore moins rapide, l'accès aux données en mémoire de masse est très pénalisant (en terme de vitesse) pour le processeur. C'est pourquoi on utilise de la mémoire ram additionnelle, qui est beaucoup plus rapide que la ram de la mémoire centrale. Au lieu de travailler en ram ou d'accéder à la mémoire de masse pour lire des données, le processeur travaille dans cette mémoire (transparente à l'utilisateur donc *cachée* pour lui) et préfère y stocker les données souvent utilisées. Ainsi à chaque fois qu'il a besoin d'accéder à des données, le processeur recherche d'abord si elles sont présentes en mémoire cache. Ce n'est que lorsqu'elles sont absentes qu'il accède en RAM ou en mémoire de masse.

Par conséquent plus la taille de la mémoire cache est grande, plus c'est intéressant pour le processeur (gain en vitesse des traitements). Comme elle est plus rapide que la ram, elle est donc plus chère et il faut trouver un compromis prix-taille.

processeur	mémoire cache de niveau 1 (processeur)	mémoire cache de niveau 2 (ram cache)	mémoire centrale (ram)	disque dur	disque compact	disquette	Impri- mante
------------	---	--	------------------------------	---------------	-------------------	-----------	-----------------

Classification de quelques éléments par ordre de vitesse décroissante

-V-LES BUS

Ce sont les moyens de transport de l'information à l'intérieur de l'ordinateur. On peut faire l'analogie entre le bus de transport en commun et le bus d'ordinateur de la manière suivante : - *unité élémentaire d'information* : la personne \leftrightarrow le bit,

- *unité de transport*: la place assise \leftrightarrow le fil,

- *capacité de transport ou taille du bus*: en nombre de places assises \leftrightarrow nombre de bits ou fils.

On distingue trois catégories de bus.

-1-Le bus de données

Il transporte les données. Sa taille est en général égale à la taille des mots que traite le processeur, c'est à dire le nombre de bits utilisé pour coder les données.

Si on utilise par exemple un processeur 32 bits (comme le pentium), la taille des données traitées est de 32 bits, et il est alors préférable d'avoir un bus de données à 32 bits.

-2-Le bus d'adresses

Il transporte l'adresse à laquelle on veut aller lire ou écrire une donnée.

Si le bus d'adresses comporte n fils, on dira que c'est un **bus à n bits**, et sa **capacité d'adressage est de 2^n** , c'est à dire que le nombre d'emplacements mémoire que l'on peut adresser (nombre d'adresses que l'on peut fabriquer) est égal à 2^n .

-3-Le bus de commande

C'est un ensemble de lignes (fils) de transport des ordres (ou commandes) destinés aux différents éléments de l'ordinateur pour effectuer les opérations.

Dans une opération d'écriture ou de lecture en mémoire centrale, on sollicite les trois bus à la fois.

Exemple1 : Pour écrire la donnée 33 à l'emplacement mémoire qui a pour adresse 55, la succession des opérations est la suivante :

- dépôt de la donnée (33) sur le bus de données,
- dépôt de l'adresse (55) sur le bus d'adresses,
- activation de la ligne WM (write memory) du bus de commande. La donnée 33 s'écrit alors dans l'emplacement mémoire qui a pour adresse 55.

Exemple2 : Pour lire le contenu de l'emplacement mémoire qui a pour adresse 54 (et qui contient la valeur 42), la succession des opérations est la suivante :

- dépôt de l'adresse (54) sur le bus d'adresses,
- activation de la ligne RM (read memory) du bus de commande,
- la donnée (42) contenue dans l'emplacement mémoire qui pour adresse 54 se retrouve alors automatiquement sur le bus de données.

Adresses Mémoire	Contenu des emplacements Mémoire
50	---
51	---
---	---
---	---
54	42
55	33
---	---

Structure de la mémoire ram

-VI-L'UNITE CENTRALE DE TRAITEMENT ou CPU

L'unité centrale de traitement ou *processeur* ou CPU (Central Processing Unit) est le cerveau de l'ordinateur. Elle comprend deux parties essentielles: l'unité arithmétique et logique (ALU) et l'unité ou organe de commande (CU), ainsi qu'un ensemble de registres de travail, et de la mémoire interne auxiliaire.

L'ALU effectue les traitements c'est à dire toutes les opérations arithmétiques et logiques.

L'unité de commande, comme son nom l'indique, a pour rôle de commander l'ALU et les autres éléments internes de l'ordinateur (principalement la mémoire). En vérité c'est le *chef d'orchestre* de l'ordinateur: c'est elle qui coordonne le travail des différents éléments. Ainsi par exemple pour effectuer l'addition de deux nombres, c'est elle qui ordonne au séquenceur de déposer les adresses des nombres sur le bus d'adresses, ordonne à la mémoire de sortir les données sur le bus de données, ordonne à l'ALU d'effectuer l'addition puis de déposer le résultat sur le bus de données.

Parmi les fabricants de processeurs les plus célèbres on peut citer Intel (processeurs 80x et pentium), Motorola (processeurs 680xx), AMD (processeurs K6 et K7), Cyrix.

Le terme *microprocesseur* fait référence à une miniaturisation très poussée, il désigne un processeur avec un fort taux d'intégration de composants (plusieurs millions de transistors).

-VII-LE SYSTEME D'EXPLOITATION

C'est un ensemble de programmes de gestion (exploitation) de l'ordinateur. Le système d'exploitation accomplit principalement deux fonctions :

- il gère les ressources partagées par les différents utilisateurs ou les différents programmes d'un même utilisateur (mémoire, entrées sorties, etc...);
 - il fournit un ensemble de services pour faciliter le travail de l'utilisateur, par l'intermédiaire d'une interface appelée interface utilisateur. Le langage de commande (à ne pas confondre avec un langage de programmation) est l'élément essentiel de cette interface.
- Chaque système d'exploitation possède son propre langage de commande.***

Le système d'exploitation est organisé en couches ou en modules ayant chacun une fonction particulière : gestion mémoire, gestion des E/S, gestion horloges, interface utilisateur, etc... En général une petite partie (le BIOS) est en mémoire rom et le reste en mémoire de masse. Parmi les plus courants on peut citer VMS, UNIX, MSDOS, WINDOWS_9X, SYSTEME 8, LINUX.

-VIII-HORLOGES ET ALIMENTATIONS

Les éléments de l'ordinateur utilisent des tensions de travail différentes; un bloc d'alimentation (boîtier) fournit les différentes tensions nécessaires (+5V,-5V,+12V,-12V, ...).

Les différents éléments travaillant à des vitesses différentes, les horloges fournissent les fréquences nécessaires à chaque élément, pour que chacun travaille à son propre rythme. La fréquence de base est celle du bus système (autour de 100Mhz), les autres sont des sous multiples ou des multiples de cette fréquence, la plus élevée étant celle du processeur.

-IX-NOTIONS DE BUS SYSTÈME ET BUS LOCAL

-1-Bus système

Il regroupe les bus de données, d'adresses, de commande, ainsi que les lignes (fils) d'alimentations et d'horloges. Il est disponible sur la carte mère sous forme de *slots d'extension* (connecteurs dupliqués) pour « enficher » les cartes d'extension ou *cartes filles*.

Dans le monde du PC, à l'origine il s'appelait bus IBM. Puis avec le développement de la technologie et l'apparition d'autres constructeurs, des bus normalisés sont apparus et les plus courants sont: le bus *ISA* (Industry Standard Architecture), le bus *EISA* (Extended ISA) moins courant et destiné aux ordinateurs serveurs, *MCA* (Micro Channel Architecture) pour les ordinateurs PS/2 d'IBM, *PCMCIA* sur les ordinateurs portables.

-2-Bus local

C'est un bus relié « *directement* » au processeur pour accélérer les transferts d'information. Les plus courants dans le monde du PC sont : le bus *PCI* (Peripheral Component Interconnect) qui sert à connecter des cartes filles et joue donc le même rôle que le bus *ISA*, les bus *VESA* (Video Electronics Standards Association) et *AGP* (Accelerated Graphics Port) dédiés au transport rapide des informations graphiques, et qui servent à connecter des cartes graphiques (communément appelées cartes vidéo).

-X-NOTION DE CARTE MERE (pour PC)

C'est la partie vitale de l'ordinateur. C'est sur cette carte que viennent se fixer les composants électroniques indispensables (processeurs, cartes mémoire, circuits d'horloges, etc...) et les *cartes filles* d'extension correspondant chacune à une application déterminée.

Le schéma de la carte mère d'un compatible PC, par exemple, reprend la structure de base d'un ordinateur donnée au paragraphe I. On y remarque un certain nombre de *slots* ou bus pour y "enficher" des cartes filles d'extension, et des *connecteurs* à relier à des prises externes par des nappes de fils pour connecter des périphériques. Sur certaines cartes mères (Xcel2000_ATX par exemple) les nappes ont tendance à être supprimées, et les prises externes sont directement montées sur la carte mère.

Il n'y a qu'un nombre limité de constructeurs de cartes mères dans le monde, et la technologie des cartes évolue très rapidement. On remarque ainsi que plusieurs fonctions assurées auparavant par des cartes filles d'extension, sont devenues des standards et sont maintenant intégrées à la carte mère:

- la fonction de traitements graphiques (carte graphique ou vidéo). Sur certaines cartes mères on rajoute de la mémoire Ram spécifique (8Mégaoctets sur la BXpro par exemple) appelée mémoire vidéo, destinée aux traitements graphiques. Sur d'autres cartes cette mémoire vidéo est prélevée sur la mémoire centrale;
- la fonction "multimédia": carte son compatible avec la norme Soundblaster, et joystick pour les simulateurs et les jeux;
- la fonction modem (carte fax modem) pour la téléphonie et la fonction réseau (en général au standard ethernet), ne sont pas encore intégrées à 100% à l'heure actuelle. Même si l'essentiel de ces fonctions est intégré, il subsiste quand même de petites cartes enfichées directement sur la carte mère dans des ports spécifiques.

Les programmes *drivers* (pilotes) nécessaires à la configuration de la carte mère pour l'utilisation de toutes ses fonctions, sont fournis par le constructeur sur un disque compact qui accompagne la documentation de la carte mère.

-1-BUS D'EXTENSION

Les slots ou bus d'extension que l'on peut rencontrer sur la carte mère d'un PC sont:

ISA: de couleur noire, c'est une extension du bus système pour connecter des cartes filles d'extension. Il peut être à 8 bits ou 16 bits de données, et travaille à la fréquence de 8 mégahertz. La fréquence du bus d'extension est différente de celle du bus système.

EISA: identique au précédent, il arrive à 11 mégahertz, mais présente l'avantage majeur de travailler avec un bus de données à 32 bits, d'où un fort gain en vitesse.

PCI: de couleur blanche, il joue le même rôle que le bus ISA, avec l'avantage de travailler à la fréquence du bus système de la carte mère (comprise entre 66 et 133 MHz).

BANK RAM ou BUS MEMOIRE: de couleur noire, ce sont les slots du bus mémoire, pour "enficher" les barrettes de mémoire ram. Ces slots sont différents en fonction de la technologie de la mémoire: Simm, Dimm, Edo, Rdr. Par exemple la carte mère TXpro2 possède des slots pour les mémoires Simm et Dimm.

VESA et AGP: de couleur marron, ils permettent de connecter des cartes d'applications graphiques.

SOCKET OU BUS PROCESSEUR: On rencontre généralement trois types de supports (ou *sockets*) pour enficher le processeur dont la fréquence varie aujourd'hui entre 500 MHz et 1GHz, certaines cartes mères en possédant deux de types différents:

-le **socket 7:** support carré et de couleur marron, il est destiné aux processeurs "bas de gamme" (486DX, pentium, cyrix, amd586...). On le rencontre généralement sur des cartes mères dont le bus système a une fréquence de 66MHz;

-le **socket SLOT1:** de couleur marron également, il a une forme longitudinale et reçoit "la carte processeur", sur laquelle se trouvent le processeur, de la mémoire cache de niveau 1 et des composants d'adaptation. Il a été inventé spécialement pour les processeurs pentium II et PIII. Il équipe généralement les cartes mères dont le bus système a une fréquence de 100MHz.

-le **socket PGA 370:** également de forme carrée il a une couleur blanche qui le distingue du socket 7. Il est destiné à recevoir des processeurs Intel-Celeron et des AMD_K6. Actuellement les recherches d'Intel s'orientent vers ce support pour en faire un bus processeur universel. Certains processeurs PIII de nouvelle génération sont destinés à être enfichés sur ce support.

-2-CONNECTEURS D'INTERFACE

Les connecteurs disponibles en fonction du type ou de la technologie de carte mère sont:

PWR1 et **PWR2**: connecteurs au format AT et ATX (cf schéma de la TXPRO2) à relier au boîtier d'alimentation qui fournit les différentes tensions pour la carte mère.

Alim2: ensemble de connecteurs sur la carte mère pour alimenter les boutons (marche-arrêt, reset), le haut parleur, ainsi que les voyants lumineux, disposés sur la carcasse de l'ordinateur.

IDE ou EIDE (EIDE: Enhanced Integrated Drive Electronics ou électronique de commande intégrée améliorée): connecteurs d'interface à relier aux mémoires de masses HD (disque dur), CD (compact disk) et DVD (digital versatile disk). Si on utilise deux mémoires de masse sur le même connecteur, l'une devra être configurée en maître (master) et l'autre en esclave (slave) à l'aide de cavaliers ("jumpers") disponibles directement sur l'extérieur du boîtier.

SCSI (Small Computer Systems Interface ou interface pour les systèmes de faible taille): de même nature que l'interface IDE, elle permet cependant de gérer jusqu'à sept périphériques différents, et de s'adapter automatiquement à la vitesse de chacun. De plus c'est une interface plus rapide. Elle a été utilisée sur les ordinateurs personnels pour la première fois par "Apple".

FDC (floppy disk controller): connecteur à relier au lecteur de disquette.

COM: connecteurs pour les ports d'interface série: com1(9 broches) et com2 (25 broches).

USB: connecteur pour le port d'interface série USB (ultra synchronous bus), beaucoup plus rapide que le port com. Il a tendance à remplacer les ports Com et Centronics.

PRN: connecteur pour le port d'interface parallèle au standard centronics, utilisé principalement pour connecter une imprimante.

VGA: connecteur d'interface graphique au standard VGA (video graphics adapter) pour connecter l'écran.

KBD: connecteur pour relier le clavier.

SND: connecteur "multimédia" pour le son (sound) avec trois prises: une pour le micro (Mic), une pour les haut parleurs (Line Out), une pour une autre entrée de signal (Line In).

JOY ou GAMES: connecteur pour un "Joystick" ou une manette de jeux.

PS2: connecteurs pour relier un clavier ou une souris au format ps2 (inventé par IBM).

Schéma synoptique de la carte mère TXPRO II (INTEL)

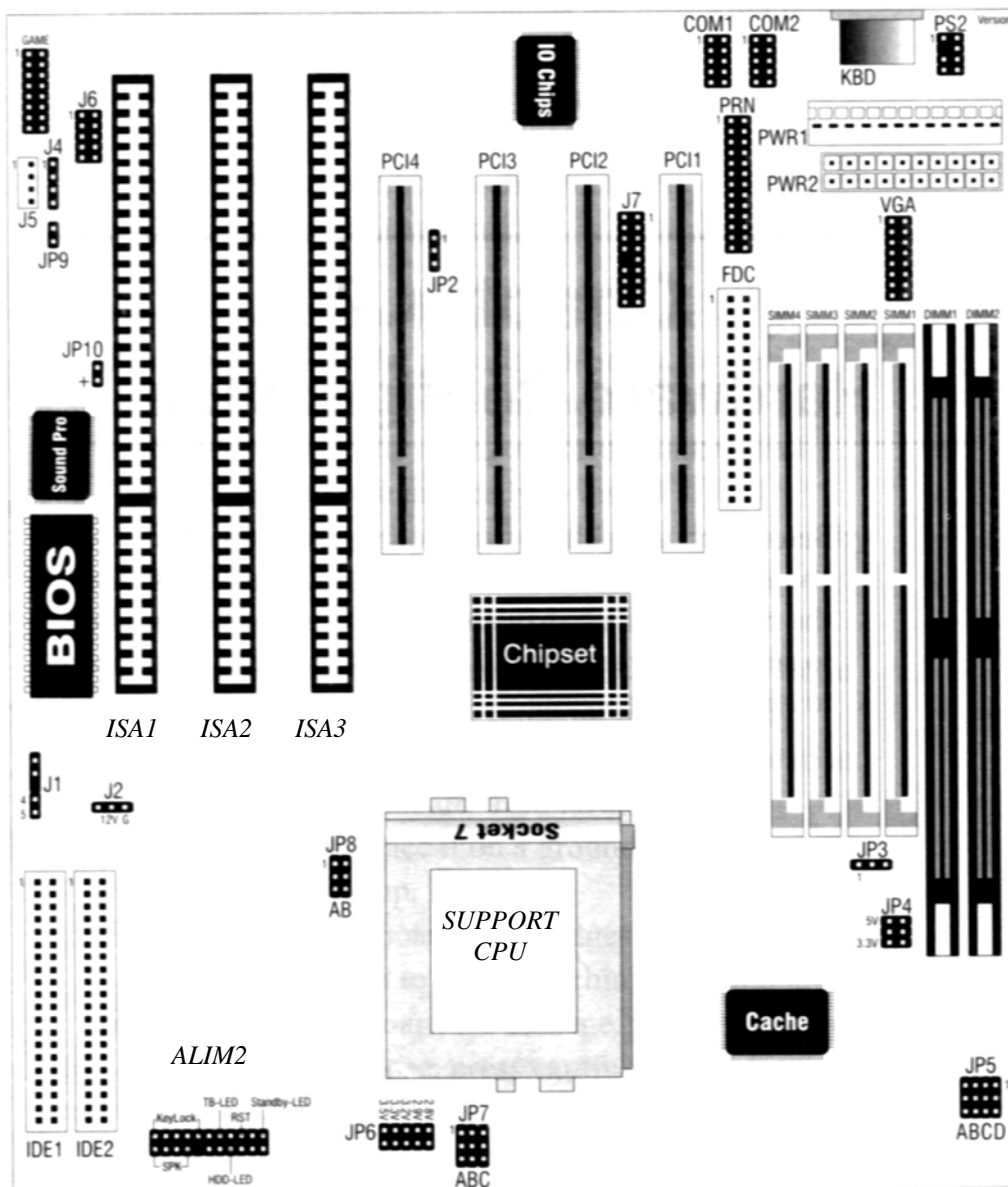


Schéma synoptique de la carte mère XCEL2000AT (INTEL)

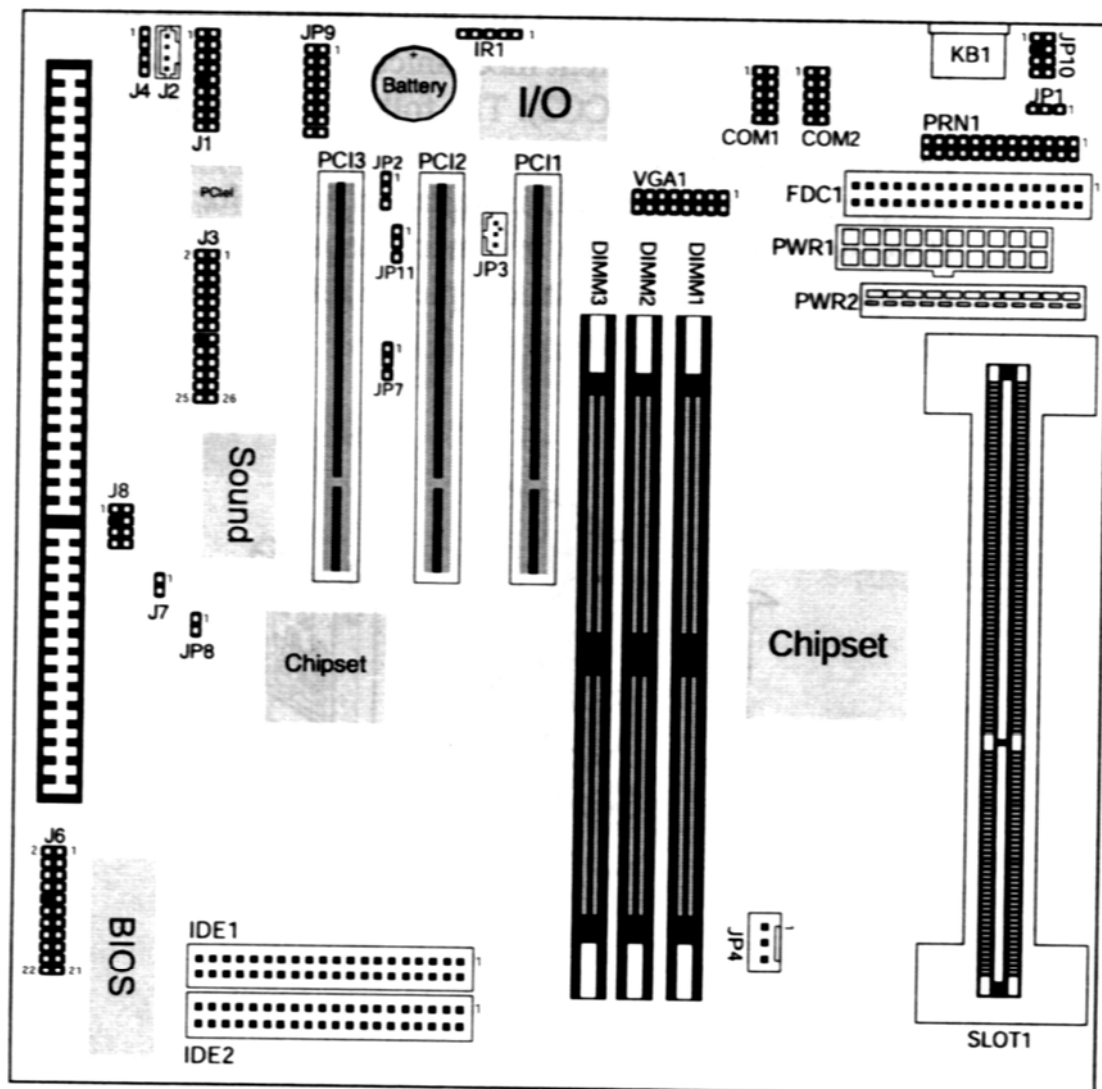
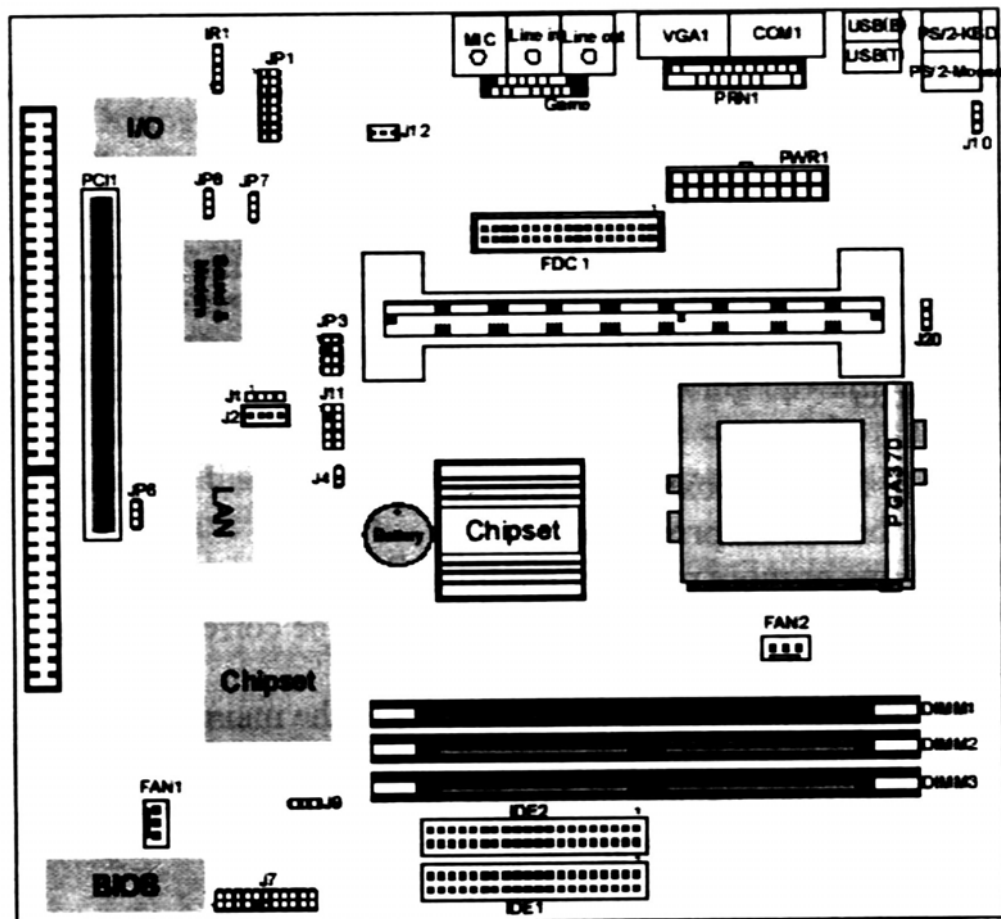


Schéma synoptique de la carte mère XCEL2000ATX (INTEL)

Mainboard Components



CHAPITRE II: REPRESENTATION DES INFORMATIONS EN MEMOIRE ET CODAGE

-I- RAPPEL SUR LES SYSTEMES DE NUMERATION.

-I-1- Principe

Tout nombre N écrit dans une base b peut se mettre sous la forme:

$$N(b) \equiv (N)_b = a_n b^n + a_{n-1} b^{n-1} + a_{n-2} b^{n-2} + \dots + a_0 b^0 + a_{-1} b^{-1} + a_{-2} b^{-2} \dots + a_{-n} b^{-n}$$

$$\underbrace{\hspace{10em}}_{\text{partie entière}} \quad \underbrace{\hspace{10em}}_{\text{partie fractionnaire}}$$

-I-2- Changement de base.

-2-1- Nombres entiers

-2-1-a- Codage: il s'agit de transcrire un nombre N exprimé en base dix dans une base b quelconque. La méthode consiste à procéder par divisions successives par la base, on s'arrête quand le dernier quotient est nul. Le résultat sera donné par les restes successifs obtenus dans l'ordre, en les écrivant de droite à gauche en fonction des puissances croissantes de la base, selon la formule suivante (où n est le nombre de divisions et les R_i les restes successifs) :

$$(N)_b = R_n b^{n-1} + R_{n-1} b^{n-2} + R_{n-2} b^{n-3} + \dots + R_1 b^0$$

Exemples: - Conversion de $(44)_{10}$ en base 2. On effectue 6 divisions successives (n=6) et on aura donc 6 restes :

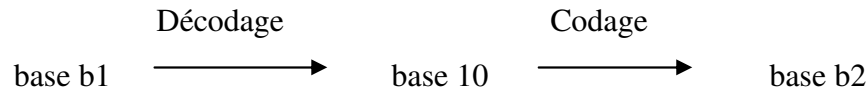
$$\begin{array}{rcl} & & R_6 \ R_5 \ R_4 \ R_3 \ R_2 \ R_1 \\ (44)_{10} & \rightarrow & (1 \ 0 \ 1 \ 1 \ 0 \ 0)_2 \\ (543)_{10} & \rightarrow & (1037)_8 \end{array}$$

-2-1-b- Décodage: cela consiste à écrire un nombre N exprimé dans une base b dans la base dix. Le résultat est obtenu en appliquant directement la formule donnée dans le principe.

Exemple: $(300)_8 = (3 \times 8^2 + 0 \times 8^1 + 0 \times 8^0)_{10} = 3 \times 64 = (192)_{10}$

-2-1-c- Transcodage : il consiste à transcrire un nombre N écrit dans une base b_1 vers une autre base b_2 , b_1 et b_2 étant quelconques.

*Cas général: cette méthode est valable quelles que soient les bases b_1 et b_2 . Un transcodage se fera d'abord par décodage puis par codage.



*Cas particuliers: Cette technique s'applique particulièrement quand la base b_2 correspond à la base 2, et la base b_1 de départ est une puissance de 2 ($b_1 = 2^n$). Si on regarde le plus grand chiffre que l'on peut écrire dans la base b_1 , on se rend compte qu'il faut en effet n bits pour coder ce chiffre en binaire. Donc **chaque chiffre du nombre N sera exprimé sous la forme d'un nombre binaire ayant n bits**.

Exemple :

$$\left. \begin{array}{l} (3)_8 = (011)_2 \\ (5)_8 = (101)_2 \\ (7)_8 = (111)_2 \end{array} \right\} (357)_8 = (011\ 101\ 111)_2$$

Remarque : Pour passer du binaire vers une base b_1 qui est une puissance de 2 ($b_1 = 2^n$), on regroupe les bits n par n pour obtenir N codé dans la base b_1 .

Exemple : $(011\ 101\ 111)_2 = (357)_8$

*Cas du BCD : un cas particulier intéressant c'est quand la base b_1 est égale à dix. Dans ce cas si on applique la même méthode que précédemment, chaque chiffre du nombre N sera codé sur 4 bits. Le résultat obtenu n'est pas du binaire mais du décimal codé en binaire (**BCD** pour Binary Coded Decimal). Pour marquer la différence avec le binaire et éviter les erreurs d'interprétation, on utilise soit une mise entre parenthèses avec une indication de la base, soit on sépare chaque groupe de 4 bits par un point.

Exemple : $(10)_{16} = (00010000)_2$ et $(10)_{10} = (00010000)_{BCD} \equiv 0001.0000$

On remarque que les nombres $(10)_{10}$ et $(10)_{16}$ qui correspondent à deux valeurs décimales différentes (10 et 16 respectivement) donnent la même suite de bits, d'où l'intérêt de les différencier.

-2-2-Nombres fractionnaires

-2-2-a-Décodage

Pour passer d'un nombre fractionnaire N écrit dans une base b vers la base 10, il suffit d'appliquer la formule donnée par le principe général (cf -I-1-).

Exemples : $(10,11)_2 = 0 \times 2^0 + 1 \times 2^1 + 1 \times 2^{-1} + 1 \times 2^{-2} = 0 + 2 + 0,5 + 0,25 = (2,75)_{10}$
 $(70,40)_8 = 0 \times 8^0 + 7 \times 8^1 + 4 \times 8^{-1} + 0 \times 8^{-2} = 0 + 56 + 4/8 + 0 = (56,50)_{10}$

-2-2-b-Codage

Pour coder un nombre fractionnaire N écrit en base 10 dans une base b quelconque, il faut traiter séparément les parties entière et fractionnaire.

Pour la partie entière il faut procéder par divisions successives par la base comme pour le codage des nombres entiers.

Pour la partie fractionnaire, il faut au contraire procéder par multiplications successives par la base. Les parties entières successives obtenues constituent le résultat, écrit dans l'ordre des puissances croissantes de la base.

On arrête les multiplications quand la précision voulue est obtenue. La précision dans la base d'arrivée ne peut pas être supérieure à celle de la base départ (base 10).

Exemple : coder $(4,345)_{10}$ en base deux avec une précision de 2^{-5} (5 chiffres après la virgule).

Partie entière : $(4)_{10} = (100)_2$

Partie fractionnaire : $(0,345)_{10} = (01011)_2$.

En effet en procédant par multiplications successives on trouve :

$$0,345 \times 2 = 0,690 \rightarrow \text{partie entière du résultat égale 0}$$

$$0,69 \times 2 = 1,38 \rightarrow \text{partie entière du résultat égale 1}$$

$$0,38 \times 2 = 0,76 \rightarrow \text{partie entière du résultat égale 0}$$

$$0,76 \times 2 = 1,52 \rightarrow \text{partie entière du résultat égale 1}$$

$$0,52 \times 2 = 1,04 \rightarrow \text{partie entière du résultat égale 1}$$

Par conséquent $(4,345)_{10} = (100,01011)_2$.

-II-REPRESENTATION OU CODAGE BINAIRE DES NOMBRES EN MEMOIRE

-II-1-Nombres entiers

On dit qu'un ordinateur traite des mots de n bits s'il **utilise n bits pour coder les données**.

Dans ce cas le plus grand nombre non signé (ou plus grande valeur absolue) que l'on peut coder est égal à $2^n - 1$.

-1-1-Technique valeur absolue plus signe

Si on utilise des nombres signés (positifs ou négatifs), la technique la plus simple est de réserver un bit, sur les n disponibles, pour coder le signe. Ce bit le plus significatif (bit de poids le plus fort) est appelé **bit de signe**. Si le nombre est positif on met le bit de signe à zéro, et s'il est négatif on le met à un. Ainsi par exemple avec un codage sur 4 bits, +7 s'écrit 0111 et -7 s'écrit 1111 : 1 bit pour le signe et 3 bits ($n-1=3$) pour coder la valeur absolue.

Dans ce cas les plus grands nombres que l'on peut coder sont $+(2^{n-1} - 1)$ et $-(2^{n-1} - 1)$.

L'inconvénient de cette technique de codage est que le nombre zéro est codé deux fois : + 0 et - 0. De plus lors des opérations arithmétiques, il peut y avoir débordement sur le bit de poids le plus fort, ce qui entraîne des difficultés d'interprétation du bit de signe.

-1-2- Technique du complément à 2

Pour remédier à ces problèmes, on utilise la technique du complément à 2 pour coder les nombres négatifs. Le complément à 2 est obtenu par addition de un au complément à un (obtenu en inversant les bits de la valeur absolue : les zéros deviennent des uns et vice versa). Le complément à 2 est également appelé complément vrai, et le complément à un appelé complément restreint.

$$C_2(-N) = C_1(N) + 1 \text{ (où } N \text{ correspond à la valeur absolue).}$$

Les plus grands nombres que l'on peut coder sont alors $+ 2^{n-1} - 1$ et $- 2^{n-1}$.

Remarques : * le complément à 2 d'un nombre positif est lui même. La technique du complément à deux permet d'économiser un circuit soustracteur dans la conception des UAL. Ainsi l'opération $A - B$ sera considérée comme une addition: $A + (-B)$. Il faut donc rajouter à A le complément à 2 de $(-B)$;

* dans la technique du complément à deux, le zéro est codé une seule fois, ce qui libère un code (celui du - 0) pour coder un nombre négatif supplémentaire.

-1-3-Conversion du complément à deux vers le décimal

Le bit de poids le plus fort permet de déterminer le signe et la valeur de l'équivalent décimal du nombre. S'il est égal à zéro, le nombre est positif et il suffit de convertir directement en décimal le code binaire. S'il est égal à un, le code binaire correspond au complément à 2 d'un nombre qui est négatif. Pour obtenir la valeur absolue décimale du nombre, il faut donc appliquer une nouvelle fois le complément à 2 à la valeur binaire, puis faire la conversion en décimal, et enfin lui affecter le signe moins.

Exemple1 : nombres codés sur quatre bits.

(0011) : le bit de poids le plus fort est égal à 0, donc cette série de bits est le complément à deux d'un nombre positif. Il s'agit de +7.

(1101) : le bit de poids le plus fort est égal à 1, donc cette série de bits est le complément à deux d'un nombre négatif. Calculons d'abord le complément à 2 : $C2(1101) = (0011) = 3$. Donc la suite de bits donnée est le complément à deux de -3.

Exemple 2: le tableau qui suit donne pour chaque série de bits, sa signification dans les trois techniques de codage des nombres entiers, avec un codage sur 4 bits.

Série de bits	Equivalent décimal Techn.valeur absolue	Equivalent décimal Tech.val.abs.+ signe	Equivalent décimal Tech.complément à 2
0000	0	+0	+0
0001	1	+1	+1
0010	2	+2	+2
----	--	--	--
----	--	--	--
0111	7	+7	+7
1000	8	-0	-8
1001	9	-1	-7
1010	10	-2	-6
----	--	--	--
----	--	--	--
1110	14	-6	-2
1111	15	-7	-1

-II-2-Nombres fractionnaires

-2-1-Technique de la virgule fixe

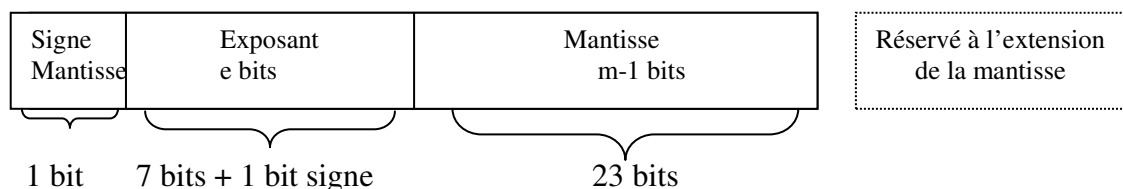
On réserve un nombre de bits fixe pour les parties entière et fractionnaire. Par exemple 24 bits et 8 bits respectivement, pour un nombre codé sur 32 bits. On dira alors que la précision est de 2^{-8} .

Les deux inconvénients majeurs de cette technique sont que d'une part la précision est toujours la même (2^{-8} dans l'exemple), d'autre part le plus grand nombre que l'on peut coder est limité ($\pm(2^{24-1} - 1) = \pm(2^{23} - 1)$ dans l'exemple). C'est pourquoi cette technique est encore utilisée uniquement sur certaines calculatrices.

-2-2- Technique de la virgule flottante

Comme son nom l'indique, cette technique est inspirée de l'écriture décimale avec les puissances de 10, où l'on peut déplacer à sa guise la virgule et modifier en conséquence l'exposant de la puissance de 10. Ainsi par exemple 123,5 peut s'écrire $12,35 \cdot 10^1 = 1,235 \cdot 10^2 = 0,1235 \cdot 10^3$. On dit alors que 1235 représente la mantisse et 3 représente l'exposant.

Sur les n bits utilisés pour le codage des nombres, on réserve m bits pour la mantisse et son signe (**mantisse en général purement fractionnaire**), et e bits pour l'exposant et son signe (**exposant en général codé en complément à deux**).



Ainsi dans l'exemple précédent, la précision est égale à 2^{-23} (car mantisse purement fractionnaire), alors que l'exposant en complément à 2 codé sur 8 bits varie de -128 ($-2^{(8-1)}$) à $+127$ ($+2^{(8-1)} - 1$). Ce qui donne pour valeurs extrêmes des nombres $N_{\min} = (-0,111\dots1) \cdot 2^{-128}$ et $N_{\max} = (+0, \underbrace{111\dots1}_{23\text{fois}}) \cdot 2^{+127}$.

L'intérêt de la technique de la virgule flottante est donc double : d'une part pour le même nombre de bits que pour la virgule fixe, on augmente de manière considérable la précision, d'autre part le nombre le plus élevé que l'on peut coder est beaucoup plus grand.

-III- CODAGE DES CARACTERES

Le codage de l'information est effectué pour faciliter la communication et la transmission de l'information, soit à l'intérieur d'un ordinateur, soit entre plusieurs ordinateurs, soit entre un ordinateur et des périphériques. Les codes les plus utilisés sont le code Baudot et le code ASCII.

-III-1- Code CCITT n° 2 (code Baudot) ou code télégraphique

Ce code inventé par Baudot a été normalisé par le CCITT (Comité Consultatif International du Télégraphe et du Téléphone). Il est destiné aux communications télégraphiques (d'où le nom de code télégraphique). Il est aussi connu sous le nom de code à cinq moments car les caractères (chiffres, lettres, symboles etc...) sont codés sur 5 bits.

A chaque caractère alphanumérique on associera donc un code sur 5 bits. Avec cinq bits on peut obtenir $2^5 = 32$ codes possibles. Sur les trente-deux codes deux sont réservés pour des caractères de contrôle: l'un ($1B_{\text{hex}}$) pour indiquer que ce qui suit est une série de chiffres, et l'autre ($1F_{\text{hex}}$) pour indiquer les lettres. Les trente codes qui restent sont attribués 2 fois : une fois pour les chiffres et une fois pour les lettres. Avec cette technique on peut donc coder soixante caractères car un même code peut désigner une lettre ou un chiffre. Par exemple le code 01_{hex} peut désigner le chiffre 5 ou la lettre T, et $1C_{\text{hex}}$ désigne le chiffre 7 ou la lettre U. Ainsi pour la transmission de la chaîne de caractères « T O T O 5 9 O U 7 9 », la série de codes à transmettre est : $1F\ 01\ 03\ 01\ 03\ 1B\ 01\ 03\ 1F\ 03\ 1C\ 1B\ 1C\ 03$ (on remarquera les deux caractères de contrôle avant chaque passage de lettre à chiffre et vice versa).

Lors de la transmission chaque groupe de 5 bits (représentant l'information utile d'un caractère) est encadré par deux bits : un bit de début de caractère (*start*) et un bit de fin de caractère (*stop*).

Remarque : quand on écrit les 32 codes hexadécimaux en binaire, on remarque que les 3 bits de poids les plus forts sont toujours à zéro. On peut donc les supprimer pour obtenir un codage sur 5 bits.

Bien que ce code fut révolutionnaire au moment de son invention, et présente l'avantage de générer un faible volume d'information à transmettre (en nombre de bits), il présente cependant trois inconvénients majeurs :

- le nombre de caractères à coder est très limité,
- on ne prend en compte que les caractères majuscules,
- ce code est peu structuré (l'ordre logique d'attribution des numéros de code est difficile à comprendre) donc difficile à apprendre.

LETTRES	CHIFFRES	CODE BAUDOT	CODE ASCII	
			code Lettres	code Chiffres
A	-	18	41	2D
B	?	13	42	3F
C	:	0E	43	3A
D	♣	12	44	--
E	3	10	45	33
F	E	16	46	--
G	%	0B	47	25
H		05	48	--
I	8	0C	49	38
J		1A	4A	--
K	(1E	4B	28
L)	09	4C	29
M	.	07	4D	2E
N	,	06	4E	2C
O	9	03	4F	39
P	0	0D	50	30
Q	1	1D	51	31
R	4	0A	52	34
S	'	14	53	27
T	5	01	54	35
U	7	1C	55	37
V	=	0F	56	3D
W	2	19	57	32
X	/	17	58	2F
Y	6	15	59	36
Z	+	11	5A	2B
CR (retour chariot)		02	0D	--
LF (saut de ligne)		08	0A	--
SP (espace)		04	20	--
Initialisation des communications		00	--	--
Code chiffres		1B	--	--
Code lettres		1F	--	--

Tableau des codes Baudot et ASCII

-III-2- Code CCITT n°5 (ou code ISO) et code ASCII

Pour faciliter les échanges internationaux d'information, l'ISO (International Standard Organization) a normalisé la proposition américaine ASCII (American Standard Code Information Interchange), ce qui a donné le code ISO ou version internationale de référence.

C'est un code à 7 bits (chaque caractère est codé sur 7 bits). Le huitième bit peut être utilisé soit comme bit de parité pour la protection contre les erreurs de transmission, soit pour l'extension des caractères graphiques (codage de 128 caractères supplémentaires).

On peut donc coder $2^7 = 128$ caractères. Ces caractères se répartissent en deux groupes principaux : les caractères de commande ou contrôle forment le premier groupe (codes variant de 00_{hex} à $1F_{\text{hex}}$). Les caractères « nationaux » ainsi que les caractères alphanumériques et les symboles graphiques constituent le groupe 2 (codes variant de 20_{hex} à $7F_{\text{hex}}$).

-2-1-Code ISO et code ASCII

Dans le tableau de base international certaines positions ont été figées, d'autres ont été laissées au choix entre deux possibilités, et enfin dix positions ont été réservées aux caractères nationaux. Le code ISO correspond au tableau de base international dans lequel un choix a été effectué concernant les positions libres ou au choix (cf tableau). Le code ASCII diffère du code ISO (pour le groupe 2) uniquement par trois codes, pour lesquels le choix des caractères fait par l'ISO est différent du choix américain. Il y a donc une différence de 6 caractères.

Code Hexadécimal	Tableau de base international	Code ISO	Code ASCII	Version française	Signification du symbole
23	£ ou #	#		£	
24	\$ ou ₤	¤	\$	\$	monétaire
40	Libre	@		à	
5B	Libre	[°	degré
5C	Libre	\		ç	
5D	Libre]		§	paragraphe
5 ^E	Ev.libre	^		^	
60	Ev.Libre	`		μ	micro
7B	Libre	{		é	
7C	Libre			ù	
7D	Libre	}		è	
7 ^E	Ev.libre	~	~	¨	tréma

-2-2- Caractères de commande ou de contrôle

Les caractères de contrôle ou de commande (groupe 1) sont des caractères non imprimables dont les codes varient entre 00_{hex} et $1F_{\text{hex}}$. Ils se répartissent en cinq catégories ou séries.

-2-2-a-Série TC (Commandes de Transmission) : on trouve ici dix caractères réservés au contrôle des transmissions, dont ceux marquant le début (SOH) et la fin (ETB) du bloc, le début (STX) et la fin (ETX) du texte, la fin de transmission (EOT) et le caractère d'échappement (DLE : Data Link Escape).

-2-2-b-Série FE (mise en page) : six caractères LF (Line Feed ou saut de ligne), CR (Carriage Return ou retour au début de ligne), FF (Form Feed ou saut de page), HT, VT, (tabulations horizontale et verticale), BS (Back Space ou retour arrière d'un caractère).

-2-2-c-Série DC (commandes de périphérique) : 4 caractères DC1 à DC4.

-2-2-d-Série IS (Séparateurs d'Information) : 4 caractères IS1 à IS4.

-2-2-e-Commandes particulières : au nombre de 8, ces caractères sont : ESC (escape), NUL, SO (Shift Out ou hors code), SI (Shift In ou en code), BEL (sonnerie), CAN (cancel ou annulation), EM (End of Medium ou fin de support), SUB (substitute).

Remarque : le caractère de commande DEL (delete) a pour code 7F.

-2-3- Bits de parité, start et stop

Les codes ISO et ASCII sont des codes à 7 bits, le 8^obit (bit de poids le plus fort) peut être utilisé comme *bit de parité* : il indique le nombre de bits à un dans le code du caractère. Ce bit peut servir à la détection locale (au niveau caractère) des erreurs de transmission.

La parité peut être choisie de type paire ou impaire. Dans la parité paire, si le nombre de bits à 1 est un nombre pair, le bit de parité (bp) est mis à zéro, et il est mis à un si ce nombre est impair. Dans la parité impaire on effectue l'inverse (un et zéro respectivement).

Pour la transmission les bits de chaque caractère sont encadrés par un *bit de start* et un ou deux *bits de stop*. Ces bits ont une valeur fixe prédéterminée. On transmet d'abord le bit de start, puis les 7 (ou 8) bits du code en commençant par le bit de poids le plus faible, et enfin le(s) bit(s) de stop.

Remarque : quand la ligne de communication est au repos, elle est à l'état logique 1. Ainsi on peut détecter une coupure de la ligne de transmission si elle reste à l'état logique zéro pendant longtemps. En général le bit de start est à l'état logique zéro ($b_{start} = 0$) pour permettre la détection du début de la transmission (changement d'état par rapport au repos). Quant au bit de stop il est à l'état logique un ($b_{stop} = 1$) pour ramener la ligne de transmission à son état de repos (état logique un), et détecter ainsi le début de la transmission d'un nouveau caractère.

Dans tous les cas, l'émetteur et le récepteur décident d'un commun accord du mode de communication choisi : avec ou sans bit de parité, un ou 2 bits de stop. **La transmission asynchrone** (caractère par caractère) **standard utilise 8 bits, sans parité, 1 bit start et 1 bit stop.**

Exemple : Donner la série de bits transmise lors de la transmission des caractères A et C codés en code ISO, avec 1 bit de parité (impaire), 1 bit start et 2 bits stop.

Caractère A : code hexadécimal 41 → série de bits correspondante : 100 0001.

Le nombre de bits à 1 est pair, comme on utilise la parité impaire → bit de parité = 1.

La série de 8 bits du caractère A sera alors 1100 0001.

Caractère C : code hexadécimal 43 → série de bits correspondante : 100 0011

Le nombre de bits à 1 est impair, comme on utilise la parité impaire → bit de parité = 0

La série de 8 bits du caractère C sera alors 0100 0011.

Transmission : on encadre les 8 bits (7bits + bit de parité) de chaque caractère par un bit de start (égal à zéro) et deux bits de stop (égaux à 1), puis on transmet les bits dans l'ordre de droite à gauche.

Disposition des bits du caractère : bit stop . bit stop . bit parité . code à 7 bits . bit start

Bits transmis pour le caractère A : 1 1 1 1000001 0 .

Bits transmis pour le caractère C : 1 1 0 1000011 0 .

-III-3- Code EBCDIC

C'est un code à 8 bits dérivé du code BCD (Extended BCD Interchange Code), donc sa logique de numérotation est celle du BCD. Inventé par IBM en 1964 pour ses ordinateurs, il a été très longtemps utilisé par les autres constructeurs d'ordinateurs. Mais de nos jours il n'est guère utilisé et on lui préfère le code ASCII .

TD CHAPITRE 2 : « REPRESENTATION DES INFORMATIONS EN MEMOIRE ET CODAGE »
-I-RAPPELS SUR LES SYSTEMES DE NUMERATION : CONVERSIONS
1°Rappels de cours

-1°-1- Définition d'une base

-1°-2-Que désignent les termes bit, quartet, octet, mot ?

2°Conversions
2°-1-Nombres entiers
Solutions

-a- $(102)_3 \rightarrow (?)_{10}$

$(11)_{10}$

-b- $(321)_{10} \rightarrow (?)_4$

$(11001)_4$

-c- $(543)_{10} \rightarrow (?)_8$

$(1037)_8$

-d- $(22)_{16} \rightarrow (?)_8$

$(42)_8$

-e- $(16)_7 \rightarrow (?)_3$

$(111)_3$

-f- $(7F)_{16} \rightarrow (?)_8$ (de deux méthodes différentes) solution $(177)_8$

-g- $(177)_8 \rightarrow (?)_{16}$

$(7F)_{16}$

-h- $(7F)_{16} \rightarrow (?)_{BCD}$

$(0001.0010.0111)_{BCD} = (127)_{10}$

2°-2-Nombres fractionnaires
Solutions

-a-Décimal \rightarrow Binaire : $(9,375) \rightarrow (?)$

$(1001,011)$

$(4,345)_{10} \rightarrow (?)$

$(100,01011)_2$

-b-Binaire \rightarrow Décimal : $(101,101) \rightarrow (?)$

$(5,625)$

-II-REPRESENTATION DES NOMBRES EN MEMOIRE
-3°Nombres entiers

-3°-1-Codage : En utilisant un codage sur 5 bits, donner l'équivalent binaire des valeurs décimales suivantes, pour chacun des trois codes étudiés (Valeur Absolue Sans Signe, Valeur Absolue plus Signe, Complément à 2) : +7,+9,+13,-7,-9,-13

-3°-2-Décodage : Considérons chacune des chaînes de bits suivantes. Si elle correspond à un codage d'un nombre dans chacun des trois codes précédemment cités, donner pour chaque chaîne les trois valeurs décimales dont c'est le code. (00111), (10000), (11111), (10001)

-4-Nombres fractionnaires

-4°1-Technique de la virgule fixe : Si on utilise 16 bits pour coder les nombres, dont 12 pour la partie entière et 4 pour la partie fractionnaire, donner la précision, et les valeurs Nmin et Nmax des nombres que l'on peut coder.

-4-2°-Technique de la virgule flottante : Si on utilise 16 bits pour coder les nombres (4 pour l'exposant et 12 pour la mantisse), donner la précision, les valeurs Nmin et Nmax des nombres que l'on peut coder.

-III-OPERATIONS ELEMENTAIRES EN ARITHMETIQUE NON SIGNEE

-5°Addition -5°-1-Binaire: définir les notions de Carry (retenue) et Overflow (débordement)

-5°-2-Hexadécimal : Faire l'addition de $(23)_{16}$ et $(E9)_{16}$ de 2 méthodes différentes (directement et en passant par le binaire).

-5°-3-BCD : Additionner 0010.1001 et 0001.0101, puis donner l'équivalent décimal du résultat.

-6°-Soustraction binaire par la méthode du complément à 2: effectuer en binaire les opérations $13 - 5$ et $5 - 13$.

-7°-Multiplication et division binaire par une puissance de 2: montrer qu'on peut le faire par décalages successifs à gauche ou à droite.

-IV-CODAGE , TRANSMISSION ET IMPRESSION

-8°-1-Donner la chaîne de bits correspondant au message suivant : « TZ 58K », lors de sa transmission en code Baudot.

-8°-2-Mêmes questions que pour l'exercice 8°-1, si le codage s'effectue en code ISO, avec pour la transmission : b.p. impaire + 1b.start + 2b.stop.

-9°-Considérons le message suivant à envoyer d'un micro vers une imprimante :

AID SP LF 98 SP CR 1 SP A SP 2 SP MILLIONS LF PAR SP MOUTON? CR SP SP ON SP DELIRE ?

SP = space (espace) ; LF = Line Feed (saut de ligne) ; CR = Carriage Return (retour chariot).

9°-1-Donner pour chacun des 4 codes connus, le nombre de bits total nécessaire pour coder ce message.

9°-2-Si le message est transmis en mode asynchrone non standard (bp paire + 1b.start + 2b.stop), donner le nombre de bits total reçu par l'imprimante pour chacun des 4 codes.

9°-3-Donner la forme du message tel qu'il apparaît sur papier après impression par l'imprimante.

CHAPITRE III : LES MEMOIRES

-I-LA MEMOIRE CENTRALE

-1-Définitions

La mémoire centrale est une suite d'*emplacements* ou de *positions mémoire*, ayant tous le même nombre de bits (8, 16, 24 , 32). C'est la plus petite partie de mémoire à laquelle on peut accéder pour une opération de lecture ou d'écriture. Chaque emplacement mémoire est repéré par un numéro appelé *adresse mémoire*.

Une mémoire peut être *organisée en octets* (bytes) *ou en mots*. La taille d'un mot varie d'un ordinateur à un autre et dépend étroitement de la taille des données traitées par le processeur. La *capacité* ou *taille* d'une mémoire sera donc exprimée en octets ou en mots, selon son mode d'organisation.

Exemple : comparer la taille en bits de deux mémoires, l'une de 4 mégaoctets et l'autre de 512 kilomots de 64 bits chacun.

Il est à remarquer au préalable qu'en binaire 1 kilo vaut 1024 et non pas 1000.

$$1K \equiv 1\text{kilo} = 1024 = 2^{10} \approx 10^3 \qquad 1\text{Mega} \equiv 1M = 1K \times 1K = 2^{20} \approx 10^6 .$$

$$4Mo = 4 M \times 8 \text{ bits} = 2^2 \times 2^{20} \times 2^3 \text{ bits} = 2^{25} \text{ bits}$$

$$512 \text{ kilomots de } 64 \text{ bits} = \frac{1}{2} K \times K \times 64 \text{ bits} = 2^{-1} \times 2^{10} \times 2^{10} \times 2^6 \text{ bits} = 2^{25} \text{ bits}.$$

Les deux mémoires ont donc la même taille exprimée en bits.

-2-Principales caractéristiques

-a-Stockage des données

La ram étant une zone de travail et de stockage temporaire, on peut donc y trouver les programmes à exécuter (ils sont chargés de la mémoire de masse dans la ram pour y être exécutés) ainsi que les données à traiter (nombres, caractères, chaînes de caractères), le tout codé en binaire.

En général on utilise 8 bits pour coder les caractères, 16 à 32 bits pour les nombres entiers, et de 32 à 64 bits pour les nombres en virgule flottante. Une mémoire organisée en octets est donc bien adaptée pour stocker des caractères, alors que la mémoire organisée en mots est mieux adaptée pour le stockage des nombres, chacune pouvant et devant bien entendu stocker les deux types de données. C'est pourquoi les ordinateurs dits « scientifiques » qui sont destinés à faire presque exclusivement des calculs ont une mémoire

organisée en mots, alors que les ordinateurs « généralistes » ont une mémoire organisée en octets.

Il existe deux techniques de stockage et de lecture des informations en mémoire : « *big endian* » et « *little endian* ».

Pour stocker une donnée dans la technique big endian, on commence par l'écriture des bits de poids les plus forts de la donnée dans les positions des bits de poids les plus forts de l'emplacement mémoire.

Dans la seconde technique on fait l'inverse, c'est à dire qu'on traite les bits de poids les plus faibles en premier.

Exemple : on désire stocker successivement en mémoire le caractère « a », le nombre entier « b », et la chaîne de caractères « fgh ». Chaque caractère sera codé sur 8 bits, et le nombre entier codé sur 32 bits. Les bits du caractère a seront notés a_1 à a_8 , ceux de b de b_1 à b_{32} , etc... Donner la chaîne de bits dans les 2 techniques de stockage.

@	NUMEROS DES BITS							
	8	7	6	5	4	3	2	1
00	--	--	--	--	--	--	--	--
01	a_8	a_7	--	--	--	a_3	a_2	a_1
02	b_8	b_7	--	--	--	b_3	b_2	b_1
03	b_{16}	b_{15}	--	--	--	b_{11}	b_{10}	b_9
04	b_{24}	b_{23}	--	--	--	b_{19}	b_{18}	b_{17}
05	b_{32}	b_{31}	--	--	--	b_{27}	b_{26}	b_{25}
06	f_8		--	--	--	f_3	f_2	f_1
07	g_8	g_7	--	--	--	g_3	g_2	g_1
08	h_8	h_7	--	--	--	h_3	h_2	h_1
09	--	--	--	--	--	--	--	--

Technique Little Endian

@	NUMEROS DES BITS							
	8	7	6	5	4	3	2	1
00	--	--	--	--	--	--	--	--
01	a_8	a_7	--	--	--	a_3	a_2	a_1
02	b_{32}	b_{31}	--	--	--	b_{27}	b_{26}	b_{25}
03	b_{24}	b_{23}	--	--	--	b_{19}	b_{18}	b_{17}
04	b_{16}	b_{15}	--	--	--	b_{11}	b_{10}	b_9
05	b_8	b_7	--	--	--	b_3	b_2	b_1
06	f_8		--	--	--	f_3	f_2	f_1
07	g_8	g_7	--	--	--	g_3	g_2	g_1
08	h_8	h_7	--	--	--	h_3	h_2	h_1
09	--	--	--	--	--	--	--	--

Technique Big Endian

Disposition des bits pour une mémoire organisée en octets

On remarque que pour la mémoire organisée en octets, le stockage d'un caractère ou d'une chaîne de caractères donne le même résultat dans les 2 techniques.

On remarque également que dans le cas d'une mémoire organisée en octets, on est amené à regrouper plusieurs emplacements pour représenter un nombre ou une chaîne de caractères. L'adresse du nombre ou de la chaîne est celle du premier emplacement qu'elle occupe. On dira ainsi que l'adresse du nombre b est 02 et l'adresse de la chaîne « fgh » est 06.

@	NUMEROS DES BITS			
	32---25	24---17	16 ----9	8 ---- 1
00	----	----	----	----
01				$a_8-----a_1$
02	$b_{32}----b_{25}$	$b_{24}----b_{17}$	$b_{16}----b_9$	$b_8-----b_1$
03		$h_8-----h_1$	$g_8-----g_1$	$f_8-----f_1$
04	----	----	----	----

Technique Little Endian

@	NUMEROS DES BITS			
	32---25	24---17	16 ----9	8 ---- 1
00	----	----	----	----
01	$a_8-----a_1$			
02	$b_{32}----b_{25}$	$b_{24}----b_{17}$	$b_{16}----b_9$	$b_8-----b_1$
03	$f_8-----f_1$	$g_8-----g_1$	$h_8-----h_1$	
04	----	----	----	----

Technique Big Endian

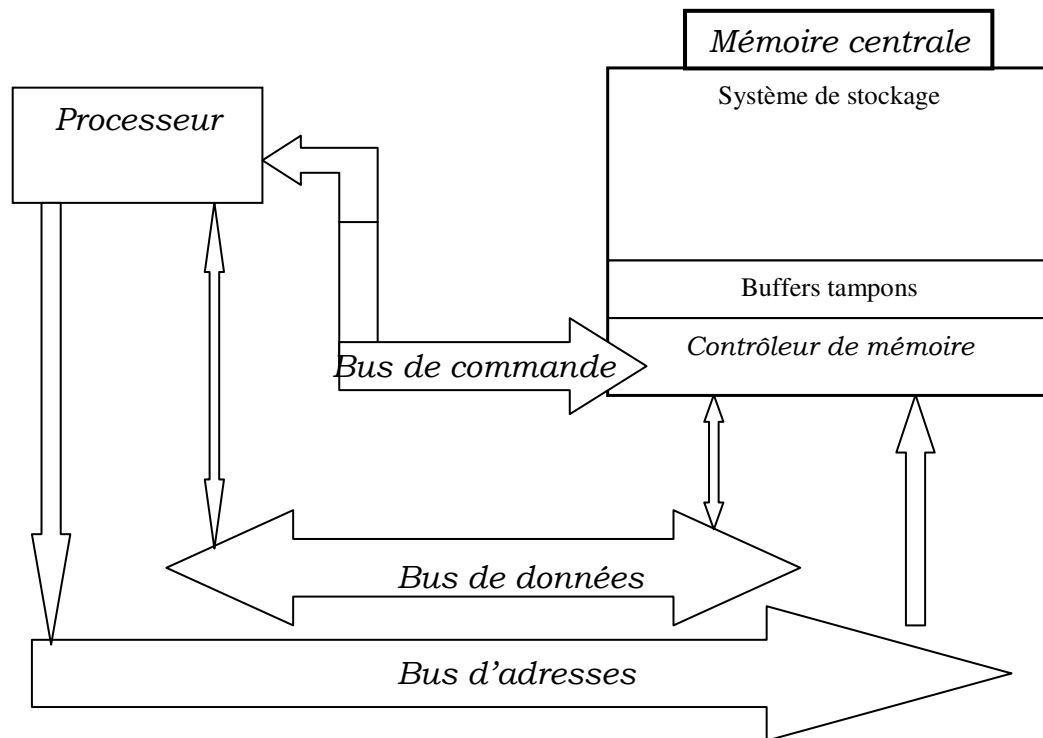
Disposition des bits pour une mémoire organisée en mots de 32 bits

On remarque ici également, que pour le nombre entier codé sur 32 bits, dans le cas d'une mémoire organisée en mots de 32 bits, le résultat est le même dans les 2 techniques de codage. Cela confirme bien que chaque organisation de mémoire est mieux adaptée pour un type de données.

On observe enfin que dans le cas d'une mémoire organisée en mots, il y a une perte d'espace et un ***fractionnement de la mémoire***: dès qu'on écrit une donnée dans un emplacement mémoire, même si la donnée ne fait qu'un bit, les bits (de l'emplacement) inutilisés sont perdus et on ne peut rien y écrire d'autre.

-b-Unité d'information échangée et contrôleur de mémoire

L'unité d'échange d'informations entre le processeur et la mémoire est l'emplacement mémoire. Quand le processeur lit (ou écrit en mémoire), on transfère soit le contenu d'un emplacement mémoire (si la donnée occupe un seul emplacement mémoire), soit le contenu de plusieurs emplacements mémoire (si la donnée occupe plusieurs emplacements). Il faut dans ce dernier cas indiquer à la mémoire le nombre d'emplacements que l'on doit transférer. D'où la nécessité d'un contrôleur de mémoire (en plus du système de stockage) qui s'occupe de ces « problèmes de gestion ».



Dans le cas d'une donnée occupant plusieurs emplacements mémoire, l'adresse transmise au contrôleur est l'adresse du premier emplacement. D'autres signaux indiquent, si nécessaire, la taille de la donnée (en nombre d'emplacements mémoire).

-c-Temps de cycle de la mémoire

Les performances de l'ordinateur sont conditionnées par la vitesse de la mémoire centrale qui est très faible devant celle du processeur. La vitesse de la mémoire dépend surtout de sa technologie de fabrication (ttl, cmos, bicmos, ecl, rdr, etc...).

Temps d'accès à la mémoire : c'est le temps qui s'écoule entre le moment où le contrôleur déclenche une demande d'accès (en lecture ou en écriture) au système de stockage, et l'instant où cette opération est terminée. Ce temps varie entre 6 et 200 nanosecondes.

Temps de cycle mémoire : c'est le temps entre deux demandes d'accès consécutives du processeur à la mémoire. Il dépend également de la vitesse du bus mémoire et de l'architecture générale de l'ordinateur. Ce temps de cycle est de l'ordre de 1 à 3 fois le temps d'accès.

Notion de bus mémoire : c'est l'ensemble des lignes d'alimentation de la mémoire, des bus de données et d'adresses, et des lignes du bus de commande destinées à la mémoire. On parle parfois du temps de cycle du bus (Bus Cycle Time) où tous les éléments interviennent, en particulier la taille du bus de données qui joue un rôle important. Pour accélérer les traitements, on rajoute des « *buffers* » *tampons* (registres intermédiaires) au contrôleur de mémoire.

-3-Espace adressable

Si le bus d'adresses comporte n bits, on peut donc adresser 2^n emplacements mémoire (chaque emplacement faisant un octet ou un mot). L'espace adressable est alors de 2^n adresses.

-a-Espace d'adresses physique et capacité de la mémoire centrale

Le processeur dépose sur le bus d'adresses l'adresse physique (ou réelle) de la donnée. Cette adresse correspond soit à un emplacement en mémoire centrale, soit à une adresse d'unité d'entrée/sortie.

Si le bus d'adresses est réservé uniquement à la communication entre le processeur et la mémoire centrale, toute adresse sur le bus est une adresse mémoire. Si d'autres unités sont connectées sur le même bus on distingue deux cas:

- soit c'est la valeur même de l'adresse qui indique s'il s'agit d'une adresse mémoire ou d'une adresse externe (par exemple adresses hautes pour E/S et basses pour la mémoire),
- soit une ligne spéciale du bus indique la nature de l'adresse transmise: adresse mémoire ou adresse d'entrée/sortie (par exemple bit = 0 pour adresse E/S et 1 pour adresse mémoire).

Espace de mémoire		Espace d'entrée/sortie
←→		←→
limite de l'espace adressable	←→	Premier cas : toutes les adresses sur le bus sont destinées à la mémoire centrale. On utilise d'autres lignes pour les unités d'E/S.
limite de l'espace adressable	←→	Deuxième cas : les adresses peuvent concerner la mémoire centrale ou les E/S.

-b-Adresse physique et adresse logique

Une adresse physique est l'adresse d'un emplacement qui a une existence réelle (physique). Dans les programmes on utilise des noms symboliques pour les emplacements mémoire contenant des instructions. Ces *adresses symboliques ou logiques* s'appellent *étiquettes* (« label »). Les noms de *variables* (x, y, a, b) sont aussi des noms symboliques qui désignent des emplacements mémoire dont le contenu peut varier pendant l'exécution du programme.

n°ligne	Label ou étiquette	Mnémonique	opérande ou adresse	Signification
1	a_prêt :	SAUT,E	lire_a	Si E est vrai, aller à lire_a
2		SAUT	a_prêt	Aller à a_prêt
3	lire_a :	ENTREE		Lire a sur le périphérique d'entrée
4		STOCKER	a	Mettre a en mémoire
5		ENTREE		Lire b sur le périphérique d'entrée
6		STOCKER	b	Mettre b en mémoire
7		FIN		

Exemple de programme en langage assembleur de l'ordinateur virtuel « mimosa »

Les lignes 3 et 4 du programme de l'exemple précédent permettent de lire la valeur de la variable « a » dans le tampon d'entrée, et de stocker cette valeur dans un emplacement mémoire, appelé justement « a » (adresse symbolique ou logique). La ligne 1 teste si la variable E est vraie (égale à 1), dans le cas positif on saute à la ligne d'instruction ayant pour nom d'étiquette « lire_a » (adresse symbolique).

Au cours de la traduction d'un programme source en un programme exécutable (par un assembleur dans le cas du programme en langage assembleur précédent), on affectera des adresses mémoire (emplacements mémoire) à toutes les références symboliques désignant une variable ou une adresse d'instruction (étiquette). La traduction ou *conversion d'adresse logique en adresse physique* (« adress translation ») définit le *mappage mémoire* (« mapping memory »), c'est-à-dire une correspondance entre l'adresse logique ou *virtuelle* et la mémoire physique. Le mécanisme retenu pour réaliser ce mappage constitue un aspect important de l'architecture d'un ordinateur et de son système d'exploitation.

Remarque : Des circuits intégrés (MMU: memory management unit) ont même été développés pour améliorer (faciliter et accélérer) la gestion mémoire.

-c-Structure de l'espace adressable et notion de segment de programme

Lorsque l'espace adressable est constitué d'une seule suite d'emplacements dont les adresses logiques sont consécutives, on dit que l'on a un *espace adressable linéaire*. Si cet espace est découpé en pages qui ont toutes la même taille (valeur qui varie de 512 octets à 4 kilooctets), on parle alors de *mémoire paginée*. L'adresse logique d'une donnée est alors constituée de deux parties : un numéro de page virtuelle et un déplacement à l'intérieur de la page. Il en est de même pour la mémoire physique qui est également découpée en pages de même longueur. Le mécanisme de traduction d'adresses (correspondance entre adresses logique et physique) passe par une *table des pages* dont chaque ligne correspond à une page et donne une description de cette page : numéro de page physique, indicateur d'état (page présente ou absente en ram), indicateurs de protection (autorisations d'accès en lecture, écriture,...).

Lorsque l'espace adressable est *segmenté*, il est alors constitué d'un ensemble de *segments de longueur variable*. Une *adresse logique* comprend alors deux parties: un *numéro de segment* est un déplacement à l'intérieur du segment ou *offset*. En général dans un programme on rencontre le *segment de code* (réservé aux instructions) et le *segment de données*. On peut même découper un programme en procédures ou sous-programmes, chacun de ces derniers correspondant à un segment de programme qui lui même peut être constitué de segment de code et de segment de données. D'où la nécessité de protéger la mémoire contre les intrusions : un programme ne peut accéder qu'à son propre segment de données (zone privée) ou à celui qu'il partage avec d'autres programmes (zone publique).

numéros offsets	emplacements mémoire	numéros de pages (ou segments)
0		0
1		
0		1
1		
2	5F	
3		
0		2
1		
2		
---	----	

L'adresse de l'emplacement mémoire qui contient la donnée 5F est (1,2) : numéro de page (ou segment) suivi de l'offset.

-d-Objectifs du mappage mémoire

L'allocation mémoire aux données et aux programmes est effectuée par le système d'exploitation et des dispositifs matériels tels que le MMU.

En plus de la traduction des adresses logiques en adresses physiques, le mappage mémoire a deux autres objectifs: rendre possible une utilisation optimale de la mémoire physique, et donner une assistance dans la protection de la mémoire. Le mécanisme de traduction d'adresses peut rencontrer deux types de problèmes:

-Espace adressable inférieur à la taille physique de la mémoire centrale: c'est le cas posé par les ordinateurs PC à cause de la compatibilité ascendante qui a nécessité de garder le même nombre de bits (20) sur le bus d'adresses, ce qui limite la taille de la ram à 1M emplacements (= 1 Moctets), alors que la taille réelle de la mémoire physique installée est de plus en plus grande (au delà de la centaine de mégaoctets). D'où la nécessité d'installer un mécanisme d'**extension d'adresses**, pour utiliser la mémoire supplémentaire disponible, qui peut être de type « hard » (matériel : on utilise un registre d'extension d'adresses ou on rajoute des lignes au bus d'adresses) ou soft (logiciel: par exemple programme EMM386. exe pour msdos).

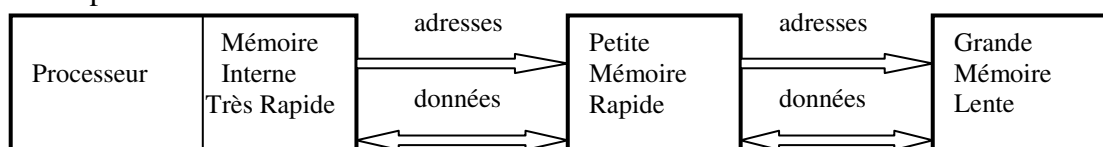
-Espace adressable supérieur à la taille de la mémoire centrale: c'est le cas que l'on rencontre avec les gros programmes qui ne peuvent pas être exécutés sur la mémoire physique installée (cas des premiers PC avec une ram de 64 Ko). A la technique la plus simple appelée recouvrement de segment ou **overlay** (qui optimise l'utilisation de la mémoire centrale en ne chargeant en mémoire que le segment de programme nécessaire à un instant donné) a succédé la technique plus compliquée de la **mémoire virtuelle** (on utilise une partie du disque comme mémoire ram virtuelle).

Il est à remarquer que dans les deux cas les opérations s'effectuent de manière transparente pour l'utilisateur.

-4-Hiérarchie de mémoire

-a-Notion de hiérarchie de mémoire

Dans un ordinateur on rencontre différentes catégories de mémoire. Plus on se rapproche du processeur, plus la vitesse de la mémoire augmente et plus l'importance de cette mémoire augmente (elle « gravit » un échelon dans la *hiérarchie*). Il faut donc trouver un compromis entre la vitesse et la capacité de chaque mémoire à cause du prix élevé des mémoires rapides.



-b-Antémémoire ou mémoire cache

Pour accélérer les traitements du processeur on insère entre ce dernier et la mémoire ram de la mémoire de même nature en petite quantité appelée *antémémoire* ou *mémoire cache*. Cette dernière est de cinq à dix fois plus rapide que la ram ordinaire (ordres de grandeurs des temps de cycles : cpu : 2ns, antémémoire : 100ns, ram : 500ns).

On utilise la mémoire cache pour les données fixes auxquelles on accède souvent. Quand le processeur émet une requête pour accéder à une donnée en mémoire centrale, on regarde d'abord si le bloc contenant l'information se trouve dans l'antémémoire. Si oui l'accès se fait uniquement à l'antémémoire. Sinon le bloc contenant l'information est transféré de la mémoire centrale vers la mémoire cache. Ainsi si la prochaine fois on a besoin d'accéder à la donnée suivante, il y a une grande probabilité qu'elle appartienne au même bloc, car les données sont souvent stockées de manière séquentielle (les unes à la suite des autres).

De la même manière, quand le processeur écrit des résultats en mémoire, ils sont écrits en mémoire cache. Comme l'utilisateur ne voit pas cette antémémoire qui est cachée pour lui, quand il demande une sauvegarde (sur disque) des résultats, en fait il va sauvegarder le contenu de la RAM. Il faut donc au préalable transférer le contenu de l'antémémoire dans la ram, il y a deux manières de transférer ce contenu : *l'écriture immédiate* (write through) et *l'écriture différée* dans le temps (write back), c'est à dire quand le processeur est libre.

L'utilisation de la mémoire cache (comme auxiliaire de la ram) augmente la vitesse des traitements, et on peut quantifier cette amélioration des performances. Ainsi si par exemple la ram et l'antémémoire ont un temps de cycle respectivement de 500ns et 100ns, et si on suppose un taux de réussite moyen de 80% (c'est à dire quand le processeur veut accéder à une information en mémoire, il la trouve dans 80% des cas dans l'antémémoire, et pour les 20% restants il ne la trouve pas et doit en plus accéder à la ram), le nouveau temps de cycle de la mémoire ou *temps de cycle apparent* est de : $80\% \times 100 + 20\% \times (500 + 100) = 80 + 120 = 200\text{ns}$. Le gain relatif en vitesse est de $(|200 - 500|) / 500 = 0,6$ soit 60%.

Remarque :

*Pendant le temps où les données s'échangent entre la mémoire RAM et la mémoire cache, le processeur n'est pas interrompu (l'exécution du programme continue), car ce temps de transfert est très court.

*L'inconvénient de l'écriture différée est qu'en cas de coupure secteur, les informations en mémoire cache seront perdues, alors que si l'utilisateur a déjà demandé une sauvegarde, en fait il n'a sauvegardé que le contenu de la ram, et non pas les dernières valeurs de ses données qui n'ont pas encore été transférées de l'antémémoire vers la ram.

-c-Mémoire Virtuelle

Cette technique permet de résoudre le problème de taille physique de mémoire ram inférieure à l'espace d'adresses logiques (cas des gros programmes et de petite ram).

On place tout ou une partie de l'espace adressable sur disque. Les programmes de l'utilisateur ne sont plus limités par la taille de la ram, mais par l'espace disponible en mémoire de masse. On peut dire que la ram se comporte comme une antémémoire pour la *mémoire virtuelle sur disque*.

Associée à la *pagination*, la technique de la mémoire virtuelle consiste à échanger dynamiquement des pages entre la *mémoire virtuelle (espace d'adresses logique)* et la *mémoire centrale (espace physique)*, et cela en fonction des besoins du programme en cours d'exécution. Ces échanges de pages sont effectués par un programme appelé *gestionnaire de pages*.

La stratégie d'appel de page à la demande s'appelle *swapping*. A chaque appel à une adresse logique du programme, on vérifie d'abord si la page qui contient l'adresse physique correspondante se trouve en mémoire centrale, en faisant appel au buffer ou *tampon de traduction d'adresses*. Si oui on peut continuer l'exécution du programme. Si non, on dit qu'il s'est produit un *défaut de page* (page fault). Le système de mappage mémoire envoie un signal d'*interruption* au processeur (le processeur s'arrête de travailler), et l'exécution du programme est alors suspendue, en attendant que le gestionnaire de pages ramène la page demandée de la mémoire virtuelle vers la ram. Le processeur est ensuite de nouveau activé pour qu'il continue l'exécution du programme.

Contrairement au cas de la mémoire cache, la nouvelle mémoire (mémoire virtuelle) est ici plus lente que la mémoire ram. Par conséquent on a un ralentissement des traitements.

On peut quantifier le ralentissement introduit par la technique de mémoire virtuelle (associée à la pagination). Considérons par exemple une mémoire ram de temps de cycle 500ns, et un tampon de traduction d'adresses de temps de cycle 100ns, et un disque dur de temps de cycle 20 ms. Si on suppose que le taux de réussite est de 80%, c'est à dire que 8 fois sur 10, quand le processeur veut accéder à une donnée, il accède au tampon où la traduction de l'adresse logique lui fournit une adresse physique en ram ; il accède alors à la donnée en ram. Dans les 20% des cas qui restent, l'adresse logique ne possède pas d'équivalent physique en ram (la donnée n'est pas en ram). Il accède alors au disque dur (mémoire virtuelle) pour transférer vers la ram le bloc de données (une page) contenant la donnée cherchée (voir détails du mécanisme à l'exercice 13 du chapitre 3).

Ainsi le temps d'accès moyen à une donnée en mémoire centrale ou temps de cycle apparent sera égal à : $0,8 (100 + 500) + 0,2 (100 + 20000 + 500) = 4600$ ns. On remarque que la vitesse s'est dégradée considérablement et la perte relative de vitesse est égale à 8,2 soit 820% ($(500 - 4600) / 500 = 8,2$). La vitesse a été divisée par un coefficient supérieur à 8.

Remarque : les objectifs de l'antémémoire et de la mémoire virtuelle (ainsi que la pagination) sont différents. On utilise la mémoire cache quand la vitesse des traitements est primordiale (on obtient une accélération car le temps de cycle apparent de la mémoire centrale est plus faible), alors que la seconde est plutôt utilisée quand on a des problèmes de taille mémoire (on augmente virtuellement la taille de la ram au détriment de la vitesse qui se dégrade).

-5-Réalisation physique de mémoire

A l'origine les boîtiers mémoire (de capacité 16, 32, 64, 128 kilooctets) s'enfichaient dans des endroits prédéfinis sur la carte mère. Pour ne pas laisser de trous dans les adresses et éviter ainsi les problèmes d'adressage, on prenait la précaution de commencer par les endroits correspondant aux adresses physiques les plus basses.

Aujourd'hui avec les nouvelles technologies (dimm, simm, edo, rdr, etc...) des groupes de boîtiers sont disponibles sous forme de **barrette de mémoire** (de capacité allant jusqu'à 128 Mo) qui s'enfichent directement sur le bus mémoire. On dispose en général de trois ou quatre **slots** (ou **banque** ou **bus mémoire**) pour enficher les barrettes, en respectant toujours la règle d'utilisation des adresses basses en premier.

La deuxième évolution importante concerne la vitesse. En effet on trouve aujourd'hui sur les ordinateurs domestiques des mémoires ram travaillant à des fréquences de 133 mégahertz (à comparer avec les 8 mhz du processeur des débuts du PC).

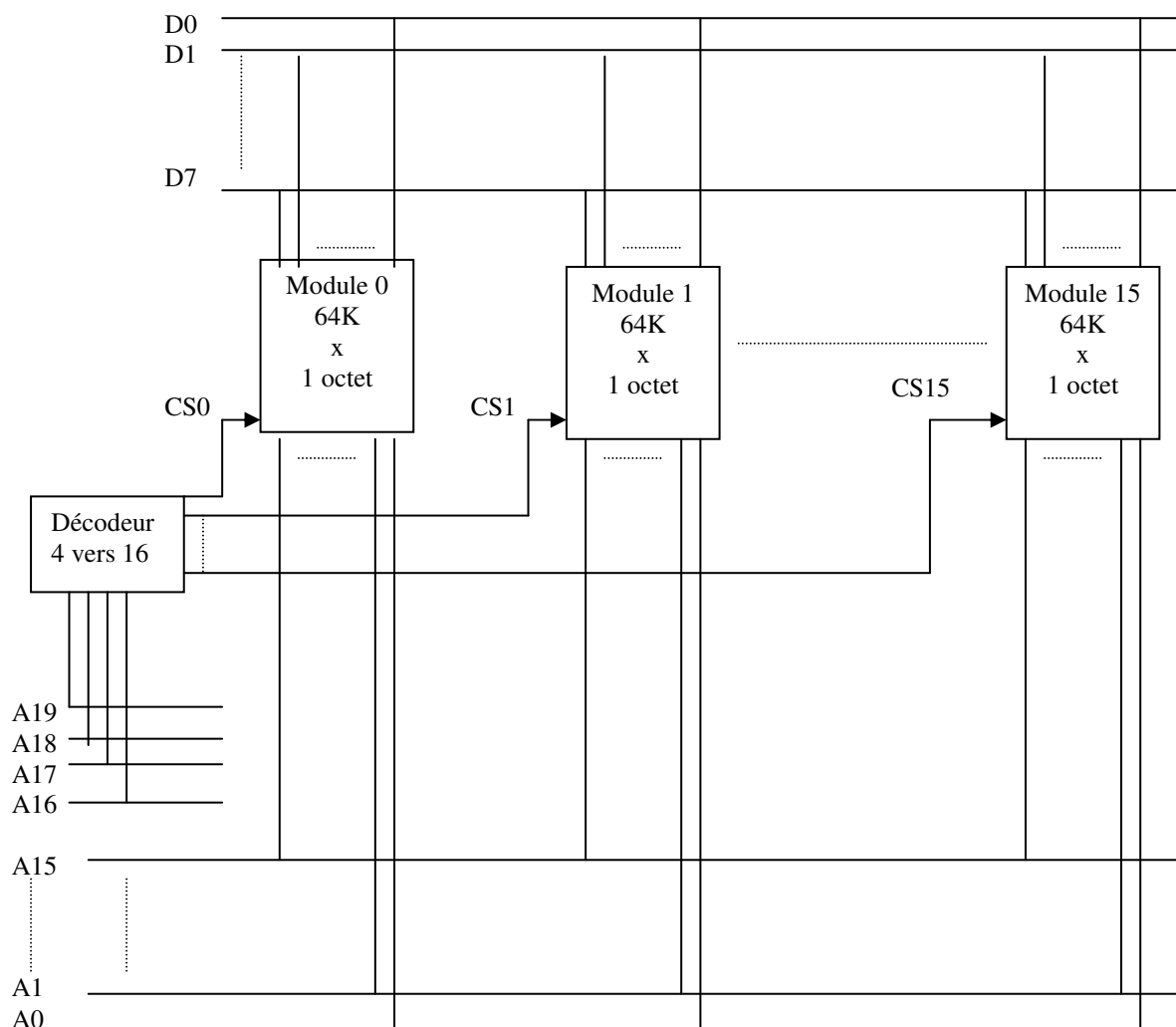
De manière tout à fait générale la mémoire centrale est réalisée à l'aide de boîtiers de mémoire vive. Chaque cellule de mémoire de un bit est réalisée soit à l'aide d'une bascule (**mémoire statique**), soit à l'aide du condensateur d'un transistor MOS (**mémoire dynamique**). Les mémoires dynamiques qui nécessitent un **rafraîchissement** périodique (pour ne pas perdre les informations car les condensateurs se déchargent) sont moins chères car moins rapides que les mémoires statiques.

Exemple : dans le schéma qui suit, on a réalisé une mémoire de un mégaoctets à l'aide de 16 modules ou boîtiers mémoire de 64k x 1 octet (64 k emplacements mémoires de 1 octet chacun), chaque boîtier étant muni d'une ligne de sélection CS (Chip Select).

Comme la mémoire est organisée en octets on a utilisé un bus de données à 8 bits (lignes D0 à D7).

Comme l'espace physique adressable est de 1 Mo (1M emplacements mémoire), le bus d'adresses aura 20 bits ($1\text{ M} = 2^{20}$), dont 4 passent par un décodeur 4 vers 16 pour la sélection des boîtiers (lignes A16 à A19).

Chacune des 8 lignes du bus de données est reliée à tous les boîtiers. Il en est de même pour les 16 lignes du bus d'adresses (lignes A0 à A15) qui permettent d'adresser les 64K emplacements mémoire de chaque boîtier ($64\text{K} = 2^{16}$).



Exemple de câblage de boîtiers mémoire pour réaliser 1 module de 1 Mo

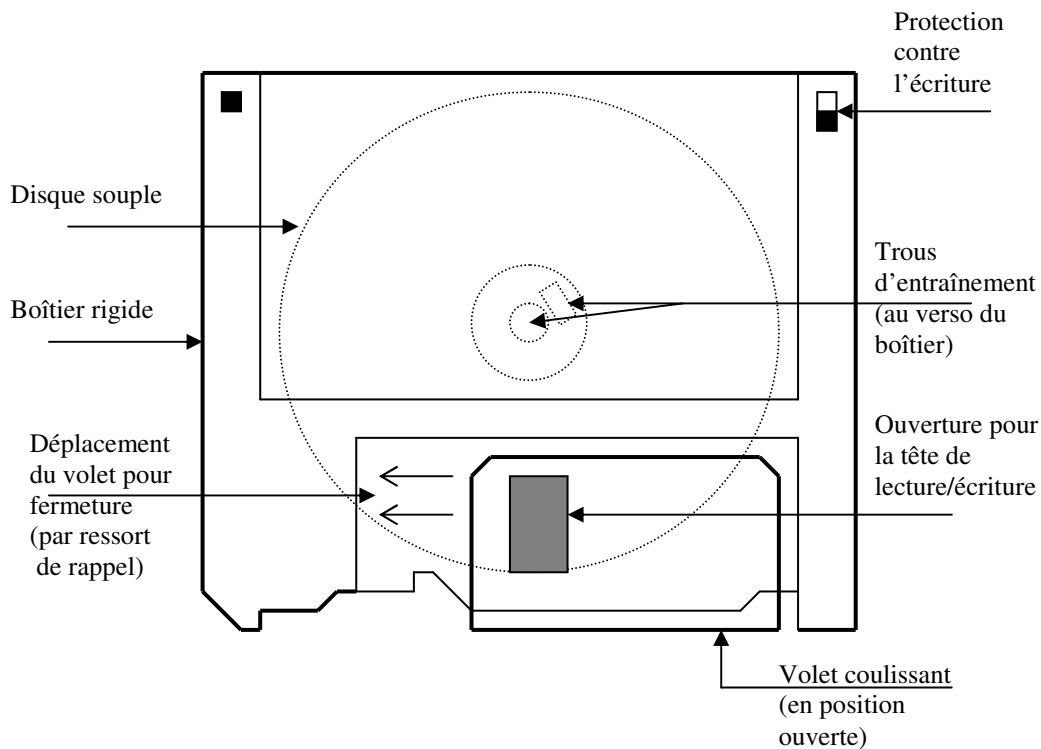
-II-LES MEMOIRES DE MASSE

-1-Le disque souple ou disquette (floppy disk)

-a-Support et conditionnement

Le support est constitué d'un disque en matière souple, le plus souvent du mylar, recouvert d'une fine couche de matière magnétisable. Si les deux surfaces du disque sont utilisables, on parle de disquette *double face* (il faut alors deux têtes de lecture/écriture).

On fait souvent suivre le terme générique (disquette) du diamètre exprimé en pouces : 8", 5" $\frac{1}{4}$, 3" $\frac{1}{2}$. Le format standard le plus répandu est le 3" $\frac{1}{2}$.



Disquette 3" $\frac{1}{2}$

-b-Formatage

La surface de la disquette est divisée en cercles concentriques constituant des *pistes* (track), dont le nombre moyen avoisine la centaine. Les pistes sont numérotées et possèdent donc une adresse physique. La *densité radiale* s'exprime en pistes par pouce : *tpi* (track per inch). Les valeurs courantes sont 48, 96 et 135 tpi. Précisons toutefois que les pistes ne sont pas matérialisées physiquement, il s'agit simplement d'un *partage logique*; c'est pourquoi on peut reformater une disquette.

Chaque piste est constituée du même nombre de cellules mémoire. Ce nombre dépend de la **densité longitudinale** exprimée en bits par pouce : **bpi** (bits per inch). Cette valeur varie de 300 à 10000 bpi. L'enregistrement se fait en simple, double ou quadruple densité. Le format le plus courant est la haute densité (HD).

Le début des pistes coïncide avec le rayon passant par le **trou d'index**, ce trou est détecté par un capteur optique.

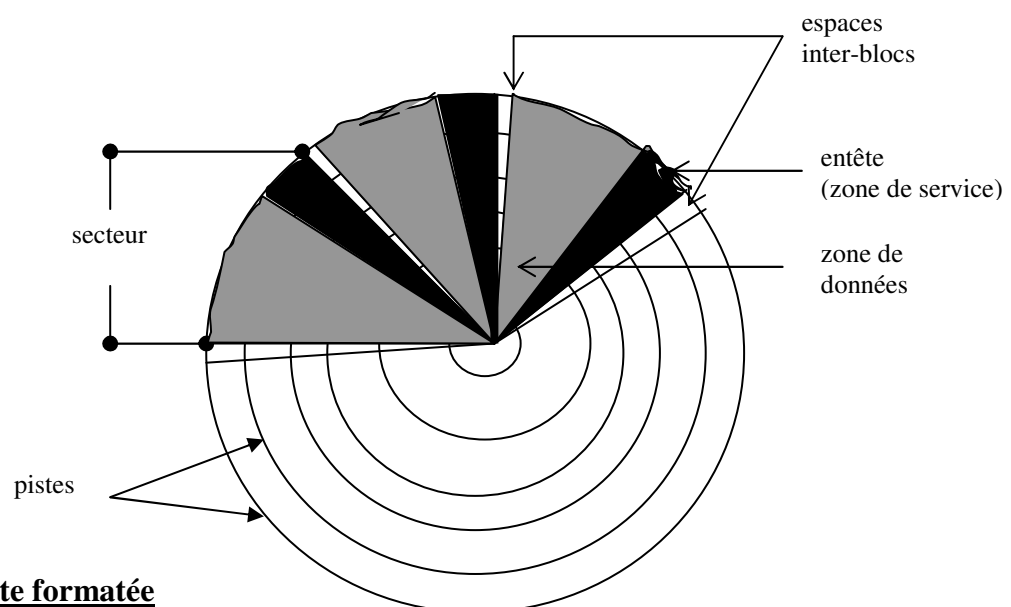
Chaque piste est divisée en **secteurs** ou **blocs**. Les secteurs d'une même piste sont donc numérotés, ce qui permet de définir leurs adresses. Deux secteurs consécutifs sont toujours séparés par un espace appelé **espace inter-blocs** (zone blanche sur la figure), dont le contenu ne peut pas être confondu avec une donnée. Le contenu d'un secteur est constitué de deux parties : un **entête** qui contient des informations de service (adresse de la piste, adresse du secteur à l'intérieur de la piste) et les **données** proprement dites. La taille d'un secteur varie généralement de 128 à 512 octets. Le nombre de secteurs par piste varie de 8 à 28.

Le nombre de pistes par surface, la longueur moyenne d'une piste, et la densité longitudinale, déterminent la capacité brute ou non formatée d'une disquette.

La **capacité brute ou non formatée** d'une disquette est le produit :

Capacité brute/face = Nombre pistes /face x Nombre secteurs /piste x Capacité d'un secteur

Le **formatage** consiste donc à inscrire magnétiquement sur la surface vierge du disque, la structuration en pistes et secteurs. Il s'agit essentiellement d'écrire les espaces inter-blocs et des informations de service dans les entêtes. En raison des espaces inter-blocs et des informations de service, la **capacité formatée** est donc inférieure à la capacité brute ou non formatée d'une disquette.



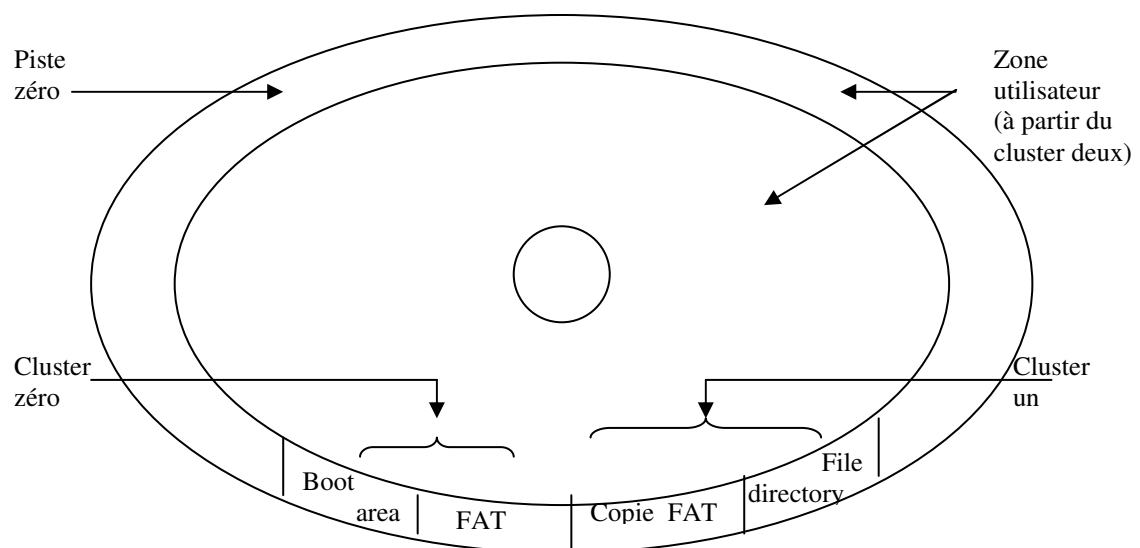
Disquette formatée

Exemple : cas du système d'exploitation ms-dos

Densité	Nombre de pistes/face	Nombre de secteurs/piste	Taille d'un secteur (en octets)	Densité radiale (en tpi)	Capacité disquette
Simple	40	8	512		
Double	40	9	512	48	360 Ko
Double:3 1/2	80	9	512	96	720 Ko
Haute :5 1/4	80	15	512		1,2 Mo
Haute :3 1/2	90	16	512	135	1,44 Mo

Ainsi une disquette standard au format 3 1/2 aura pour capacité brute :

$$90 \times 16 \times \frac{1}{2} \text{ Ko} \times 2 \text{ faces} = 1440 \text{ Ko} = 1,44 \text{ Mo.}$$



La notion de secteur ou bloc trouve son importance dans le fait que c'est la plus petite unité d'échange de données. Pour les fichiers l'unité d'allocation et de stockage est appelée **cluster**. Un cluster peut contenir 1, 2, 4 ou 8 secteurs.

D'un point de vue logique, les données enregistrées sur une disquette sont organisées en fichiers, auxquels s'ajoutent des informations de service :

- nom de la disquette (volume), date de première utilisation, nombre de secteurs libres,
- répertoire des fichiers résidents sur la disquette,
- pointeurs sur les secteurs appartenant à un même fichier.

Lors de la recherche d'information, la conversion entre la structure logique (organisation en fichiers) et la structure physique (pistes et secteurs) est prise en charge par le gestionnaire de fichiers du système d'exploitation.

-c-Principales caractéristiques d'un lecteur de disquettes

Une unité ou lecteur de disquettes se caractérise par un certain nombre de paramètres. Parmi ceux là certains concernent le formatage du support (disquette), d'autres concernent les caractéristiques physiques du lecteur proprement dit. Dans ce qui suit on donne les paramètres les plus importants avec des indications sur les valeurs moyennes et les standards couramment utilisés.

1. Le diamètre ou format ou type des disquettes (le standard est 3 1/2")
2. Le nombre de têtes de lecture/écriture (2 pour double face)
3. La densité radiale ou le nombre de pistes par face qu'on peut lire (standard 135 tpi)
4. La densité longitudinale de 3600 à 12000 bpi
 - Le nombre de secteurs par piste (standard en haute densité 16 secteurs)
 - La capacité d'un secteur (standard 512 octets par secteur) et la capacité utile
5. Capacité utile (formatée) par face: environ 650 Ko (dans standard 1, 44 Mo non formatée)
6. Vitesse de transfert ou débit nominal (0,5 Mbit/s) et utile (40 Ko/s)
7. Temps de déplacement de piste à piste (6 ms en moyenne) et temps d'accès moyen à une piste (160 ms)
8. Temps nécessaire pour déposer la tête de lecture/écriture sur le support : entre 0 et 20 ms
9. Vitesse de rotation : 360 tpm (tours par minute) soit 168 ms pour faire un tour.

On admet généralement que quand on veut lire une donnée sur une disquette, pour *trouver la bonne piste il faut parcourir 1/3 des pistes*. De même une fois qu'on a trouvé la piste, pour *accéder au bon secteur, on parcourt en moyenne la moitié de la piste (1/2 tour)*. Une fois que c'est fait, il faut alors déposer la tête de lecture/écriture sur le support.

Par conséquent le temps de positionnement moyen de la tête sur une donnée (ou ***temps de lecture d'un secteur***) est égal à la somme de trois termes:

Temps d'accès moyen à la piste + temps de recherche du secteur + temps de dépôt de la tête.

Exemple.: Si on utilise les valeurs moyennes, et qu'on désigne par A, B, C les 3 termes, on désire calculer le temps de transfert (lecture) d'un fichier de 1920 caractères codés sur 8 bits.

A = Temps d'accès moyen à la piste:

$A = 1/3 \times 80$ (nombre de pistes / face) $\times 6$ ms (temps déplacement piste à piste) = 160 ms

B = Temps de recherche du bon secteur:

$B = 1/2$ tour = $(60s / 360 \text{ tours}) / 2 = 84$ ms

C = Temps de dépôt de la tête:

$C = 16$ ms (temps moyen compris entre 0 et 20 ms)

Le **temps moyen de lecture d'un secteur** sur la disquette sera donc la somme des trois termes : $160 + 84 + 16 = 260$ ms.

Le **débit utile** moyen donne le nombre de bits lus ou écrits par seconde.

1 secteur = $1/2$ Ko en 260 ms \rightarrow en 1 seconde on aura $(1/2 \text{ Ko} / 260 \cdot 10^{-3} \text{ s}) \approx 2 \text{ Ko} / \text{s}$.

Le temps de transfert ou de lecture du fichier sera de :

$(1s / (2 \text{ K} \times 8)) \times (1920 \times 8) \approx 0.96 \text{ s} = 960$ ms.

Le **débit théorique ou nominal** est égal au produit de la vitesse de rotation (en tours par seconde) par la capacité d'une piste (en octets).

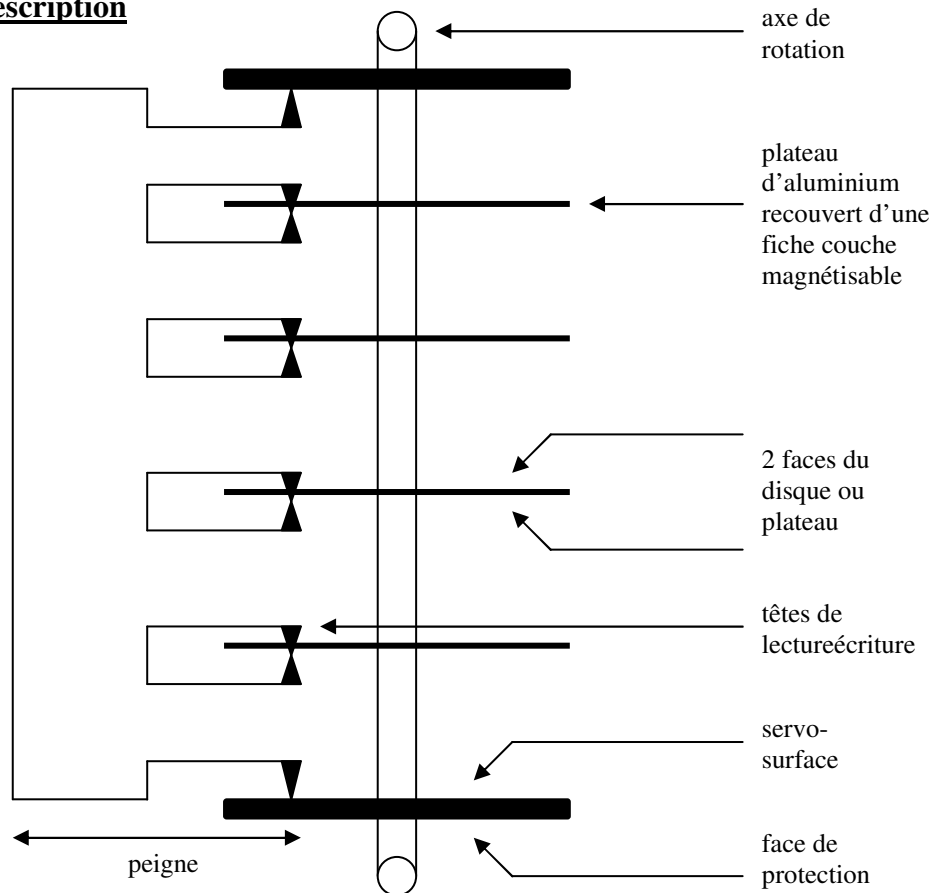
$V_r = 360 \text{ tr/mn} \leftrightarrow 6 \text{ tr/s}$	} Débit théorique = $6 \times 16 \times 512 = 48 \text{ Ko/s}$
Capacité d'une piste: 16 (secteurs) $\times 512$ (octets)	

Remarque:

On observe que ce débit théorique, généralement mis en avant par les publicités commerciales, est très loin du débit réel ou utile qui tient compte de tous les paramètres (recherche de la piste et du secteur, et dépôt de la tête sur les pistes).

-2-Le disque dur (hard disk)

-a-Description



Représentation schématique d'un disque dur

L'une des faces appelée servo-surface est généralement réservée à des informations destinées au guidage du dispositif de lecture/écriture. Les deux faces externes des deux plateaux du haut et du bas ne sont pas utilisées pour le stockage mais comme faces de protection.

Le même principe que pour les disquettes est adopté pour le formatage. La seule notion supplémentaire est celle de ***cyindre*** : c'est l'ensemble des pistes qui se trouvent simultanément sous les têtes de lecture/écriture. Dans le cas le plus simple d'une tête par face, le nombre de cylindres est égal au nombre de pistes par face, et chaque cylindre comporte autant de pistes qu'il y a de faces.

La notion de cylindre a été introduite car elle correspond à l'ensemble des informations auxquelles il est possible d'accéder sans aucun mouvement du ***peigne*** ou dispositif de lecture/écriture. L'accès aux informations sera par conséquent optimisé si des informations contiguës sont enregistrées sur un même cylindre (donc sur des pistes de faces différentes) et non pas séquentiellement sur une même face.

-b-Principales caractéristiques d'un disque dur

Si un utilisateur ordinaire ne s'intéresse qu'à la capacité du disque en gigaoctets, et à sa « vitesse » ou débit (en mégaoctets par seconde), il existe cependant d'autres caractéristiques techniques intéressantes pour l'utilisateur averti. La technologie des techniques de stockage évoluant très rapidement, il est très difficile de donner des valeurs moyennes qui seront très vite dépassées (en microinformatique ce qui date de deux à trois ans est déjà considéré comme vieux). Les valeurs qui suivent sont données uniquement à titre indicatif pour fixer les ordres de grandeur.

1. Le diamètre ou format du disque : les diamètres les plus répandus sont le 5" ¼ (pour les PC de bureau) et le 3" ½ (sur les ordinateurs portables)
2. Le nombre de têtes de lecture/écriture par face (entre 16 et 255)
3. La densité radiale ou nombre de cylindres (de 200 à 10.000)
4. La densité longitudinale de 3600 à 12000 bpi. Concrètement on préfère parler de :
 - nombre de secteurs par piste (de 20 à 64) ;
 - capacité d'un secteur (256 à 1024 octets par secteur) et sa capacité utile.
5. La capacité non formatée du disque : de quelques dizaines de mégaoctets à quelques dizaines de gigaoctets. La capacité a été multipliée par 1000 en l'espace de 10 ans. On peut également parler de nombre de clusters (65505 par exemple) et du nombre de secteurs par cluster (64 par exemple), ce qui donne un disque dur de 4 gigaoctets.
6. La vitesse de transfert ou débit nominal et débit utile (de 0,5 Mo/s à 4 Mo/s)
7. Le temps de déplacement de piste à piste (de 2 à 15 ms) et temps d'accès moyen à une piste (de 15 à 100 ms)
8. Le temps nécessaire pour déposer la tête de lecture/écriture sur le support: entre 0 et 20 ms
10. La vitesse de rotation : de 2400 à 7000 tpm (tours par minute).
11. Le type d'interface (IDE, EIDE, SCSI, USB, PCMCIA)

De la même manière que pour un lecteur de disquettes, on peut calculer le temps moyen de lecture des données ou débit moyen d'un lecteur de disque dur.

Exemple 1 : considérons une unité de disque dur tournant à la vitesse 6000 tpm, avec 4 plateaux simple face, 600 pistes par face, 20 secteurs par piste et 512 octets par secteur.

La capacité de stockage du disque sera de : $4 \times 600 \times 20 \times \frac{1}{2} \text{ Ko} = 24000 \text{ Ko} \approx 24 \text{ Mo}$.

Vitesse de 6000 tpm \Leftrightarrow 100 tr/s; 1 piste = 1 tour = 20 secteurs = 10 Ko.

Le taux de transfert théorique ou débit approximatif est égal à la vitesse de rotation multipliée par la capacité d'une piste en octets. ➔ Taux = $100 \times 10 \text{ Ko/s} = 1000 \text{ Ko/s} \approx 1 \text{ Mo/s}$.

Exemple 2 : considérons un fichier de 1920 caractères codés sur 8 bits, stocké sur un disque dur tournant à la vitesse de 6000 tpm. Tous les secteurs du fichier sont stockés sur un même cylindre, et chaque piste comprend 40 secteurs. Chaque secteur a une taille de 1024 bits dont 64 bits sont réservés à des informations de gestion (un pointeur vers le secteur suivant).

- a- Le nombre de secteurs nécessaire (utiles) au stockage du fichier est de :
 $(1920 \times 8) / (1024 - 64) = 16$ secteurs.
- b- $V_r = 6000 \text{ tr/mn} = 100 \text{ tr/s} \rightarrow$ il faut 10 ms pour faire un tour, soit 10 ms pour lire 40 secteurs \rightarrow il faut 0.25 ms pour lire un secteur.
- c- Le temps moyen de positionnement sur le bon secteur (une fois que la piste est trouvée) correspond en moyenne au temps pour faire $\frac{1}{2}$ tour (on parcourt la moitié des secteurs de la piste), soit 5 ms.
- d- Le temps moyen de transfert d'un secteur est égal au temps de recherche du secteur plus le temps de lecture du secteur, soit 5.25 ms.
- e- Comme tous les secteurs sont sur un même cylindre, on n'a pas besoin de changer de piste donc de cylindre. La recherche du cylindre ou de la piste se fait donc une seule fois, sa durée est égale à 20 ms.
- f- La durée de transfert des 16 secteurs est égale à $16 \times 5.25 = 84$ ms.
- g- La durée totale de transfert du fichier est égale à $20 + 84 = 104$ ms.

Remarques :

-1-On observe que pour le transfert d'un secteur de données, on passe 20 fois plus de temps à trouver le bon secteur qu'à lire le secteur proprement dit (5ms contre 0,25 ms). Il est donc intéressant de ***stocker les données d'un fichier qui sont sur une même piste sur des secteurs contigus***.

-2-La recherche d'un cylindre (ou d'une piste) prend 4 fois plus de temps que la recherche d'un secteur (20 ms contre 4 ms). Il est donc plus intéressant de ***stocker les données d'un fichier sur un minimum de cylindres, pour économiser les déplacements entre pistes***.

-3-On remarque que pour le même fichier stocké à peu près dans les mêmes conditions sur un lecteur de disquettes, la transfert du disque dur est environ dix fois plus rapide par rapport à la disquette (durée 10 fois plus courte).

-4- Grâce aux progrès technologiques, les densités radiale et longitudinale sont de plus en plus grandes, et on a tendance à diminuer le nombre de plateaux (de 2 à 4) et à augmenter le nombre de têtes par face (jusqu'à 32), ce qui augmente les taux de transfert.

-3-Autres procédés de stockage

La technologie des supports de stockage évolue très rapidement, et on atteint des capacités de stockage phénoménales et inimaginables il y a à peine quelques années, en raison principalement de deux facteurs : l'augmentation de la densité et l'utilisation de techniques de compression de données (Zip, Jpeg et Mpeg principalement). Par ailleurs l'amélioration des interfaces et l'utilisation croissante de mémoire cache au niveau de l'électronique intégrée aux lecteurs, permettent des vitesses de transfert de plus en plus grandes.

Un élément important qui influence grandement les performances est le type d'interface utilisée. Le taux d'occupation du processeur variable (de 1% à 90 %) et les débits très éloignés font qu'on obtient en bout de course des taux de transfert très différents. En effet l'interface parallèle même configurée dans le bios en mode EPP (enhanced parallel port), plus rapide que le mode ECP (enhanced capabilities port), autorise un taux de transfert limité (de 300 Ko/s à 1,5 Mo/s), alors que l'interface Scsi va à plus de 5 Mo/s. De même l'interface usb a un débit limité à 1,5 Mo/s (en attendant la version 2.0 qui va de 45 Mo/s à 60 Mo/s, soit 30 à 40 fois celui de la version 1.1) alors que l'EIDE autorise un débit allant de 5 Mo/s à 33 Mo/s.

Mais quelles que soient les améliorations technologiques, les deux techniques principalement utilisées pour l'enregistrement des données sont soit de type optique (laser) soit de type magnétique (ou une combinaison des deux).

-a-lecteurs et supports magnétiques

a-1-Le dérouleur de bandes ou streamer :

Il y a encore très peu de temps, c'était encore le seul moyen d'archivage (backup de disques durs) digne de ce nom, utilisé pour les gros systèmes informatiques des grandes entreprises.

La bande magnétique entraînée par un moteur asservi passe d'une bobine émettrice vers une bobine réceptrice. La largeur de bande (1/4", 1/2", 1 pouce), le nombre de pistes, ainsi que le mode d'enregistrement varient beaucoup d'un modèle à l'autre.

On utilise souvent l'appellation anglaise de « streamer » à cause du mode d'enregistrement des données qui est continu (stream). Il consiste à insérer les données sans tenir compte de la structure en fichiers et blocs, ce qui permet de gagner en capacité utile (pas d'espace interbloc comme sur les disques) et d'augmenter la vitesse des transferts. Cependant cela présente l'inconvénient de ne pas permettre l'accès à un fichier isolé ou un bloc particulier.

Certaines unités récentes permettent l'exploitation aussi bien en mode continu, qu'en accès sélectif aux données enregistrées. Par ailleurs l'évolution technologique aidant, on a de plus en plus tendance à remplacer les bandes isolées par des cartouches de plus petite taille, où les deux bobines et la bande sont enfermées dans un support en plastique comme pour les cassettes audio.

Compte tenu de l'évolution technologique et de la capacité de plus en plus grande des supports de stockage sur disque amovible du type disque compact (de l'ordre de la vingtaine de gigaoctets), on s'achemine à terme vers le remplacement de la bande magnétique par le support en plastique car le stockage des données est plus fiable en raison de l'utilisation du laser (pas de contact entre la tête de lecture et le support), et la durée de stockage garantie peut aller jusqu'à 50 ans.

a-2-Les lecteurs de cartouches :

Toutes les solutions proposées sont des solutions « propriétaires », c'est à dire que ce sont des technologies propres à chaque constructeur et qu'il n'y a pas de norme standard. Cependant quatre « acteurs » principaux (Fujitsu, Castlewood, Iomega et Imation) émergent et se battent pour imposer leur technologie comme standard. Toutefois, en raison de leurs capacités, la plupart des solutions sont destinées à un usage personnel (pour remplacer le lecteur de disquettes de 1,44 Mo), et bien peu sont destinées à un archivage intensif de données au sein d'un groupe de travail.

La *technologie magnétique* est utilisée par IOMEGA, depuis déjà plusieurs années, avec ses lecteurs de cartouche ZIP et JAZ. Les cartouches ZIP sont lues par des lecteurs différents en fonction de la capacité (100 ou 250 Mo). Quant aux cartouches JAZ, elles sont plus grandes (format 5" 1/4) et permettent de stocker jusqu'à 2 Go. Enfin les cartouches Click ! renferment un mini-disque métallique (50 x 55 mm) de 40 Mo, qui est une solution pratique pour l'informatique mobile (portables, appareil photo, caméscope...). **Aucun de ces lecteurs ne permet de lire les disquettes aux anciens formats (1,44 Mo).**

Comme les disques durs, les appareils de chez ORB utilisent la *technologie magnéto-résistive*, avec des supports pouvant atteindre une capacité de 2,2 Go.

La *technologie laser-servo* utilisée par Imation permet de stocker 120 Mo sur une cartouche (ce qui est largement suffisant pour une sauvegarde ponctuelle), tout en conservant la **compatibilité avec les supports existants de 1,44 Mo ou 720 Ko**. Elle utilise une mécanique d'origine Matsushita qui supporte des médias au format disquette classique ou cartouche.

Au chapitre des produits annoncés et prochainement disponibles, Sony et bien d'autres fabricants comptent commercialiser un lecteur de cartouches (HiFD 200) de 200 Mo, également compatible avec les disquettes traditionnelles.

La *technologie magnéto-optique* conçue par Fujitsu (et vendue en OEM aux autres constructeurs) permet de stocker sur des cartouches allant de 128 Mo à 1,3 Go en usage personnel, et de 2,6 à 5,2 Go par face pour les modèles destinés à l'archivage. Cette technologie est improprement appelée magnétique car les informations sont écrites et lues par un laser. D'un point de vue technique les données sont stockées sur un disque magnéto-optique (enfermé dans une cartouche) en changeant la polarité de la couche d'enregistrement. Tourné dans une direction, le pôle nord de la charge magnétique équivaut à un « 1 » ; tourné dans l'autre il équivaut à un « 0 ». Si la fiabilité des données est inégalée et les disques certifiés pour une longévité supérieure à 30 ans (car l'enregistrement se fait au moyen d'un laser contrairement à l'enregistrement magnétique où la tête de lecture est en contact direct avec la cartouche), les performances sont cependant inférieures à celles des dispositifs magnétiques.

Le lecteur de disque optique numérique effaçable PD de Matsushita est très peu différent du lecteur magnéto-optique de Fujitsu. Apparu il y a environ cinq ans, ce lecteur enregistreur utilisait des cartouches de 250 à 640 Mo. Il avait la particularité de pouvoir lire également les cd-rom classiques de taille 640 Mo.

-b-Lecteurs et supports optiques

b-1-Le disque compact ou cd

Ce disque en plastique de 12 cm de diamètre a une capacité standard de 650 Mo. On augmente le volume des données stockées par l'utilisation de techniques de compression. Les techniques principales utilisées sont zip (pour les données), mp3 (pour les fichiers son), mpeg-1 et mpeg-2 (pour les images).

Au niveau vitesse on utilise comme unité le 1X correspondant à un taux de transfert de 150 Ko/s. Le temps d'accès moyen aux données se situe autour de 60 ms pour les lecteurs, et 120 ms pour les graveurs (lecteurs enregistreurs). Ainsi un appareil avec la mention 6X/4X/24X signifie que la gravure des cd-R se fait en 6 vitesses (6 x 150 Ko/s), celle des cd-RW en 4 vitesses, et la lecture des cd-rom en 24 vitesses. La vitesse des lecteurs atteint 50X alors que celle des graveurs ne dépasse pas les 8X en écriture et 24 X en lecture.

CD-DA ou CD-audio : ancêtre de tous les cd, il permet d'enregistrer 74mn de musique en stéréo, avec une résolution 16 bits, à une fréquence d'échantillonnage maximale de 44,1 khz.

CD-texte : il s'agit d'une amélioration apportée par sony au format cd-audio. En plus des pistes de son, diverses informations (les titres des morceaux, le nom des interprètes, etc..) sont enregistrées. Ce format n'est pas encore reconnu par tous les graveurs, ni par tous les logiciels, et peu de platines de salon sont en mesure de le lire.

CD-extra : la première partie (ou *session*) contient de la musique et peut donc être lue par un lecteur de cd-audio. La deuxième partie qui contient des données informatiques (paroles de chansons, photos,...) nécessite un lecteur de cd-rom pour être lue.

Une session sur un cd est un ensemble de données (informatiques, audio ou vidéo) délimité par une plage (un espace sans données) de début et une plage de fin. Ces plages gaspillent 15 Mo d'espace environ.

CD-rom xa : ce format permet de mélanger son, séquences vidéo et données informatiques. Il est à l'origine de l'explosion du multimédia dans le monde du PC. Il permet de stocker au maximum 650 Mo de données (au format ISO9660), étendu après jusqu'à 800 Mo.

CD-vidéo : il contient environ 75 mn de séquences vidéo comprimées au format MPEG-1 (352 x 288 pixels, 25 images par seconde). Au format vidéo MPEG-2, on peut stocker une douzaine de minutes d'images de haute qualité (mais nécessite d'une carte de décodage).

CD-R et CD-RW : à la différence des cd-R (compact disc recordable) qui ne peuvent être gravés qu'une fois, les cd-RW (rewritable) peuvent être gravés jusqu'à 1000 fois. Un graveur de cd-rw peut écrire des cd-R (alors que l'inverse n'est pas vrai). Les CD-R doivent être gravés avec une vitesse inférieure à 6X, et les CD-RW avec une vitesse inférieure à 6X. Au delà il faut utiliser des CD prévus pour la gravure à grande vitesse.

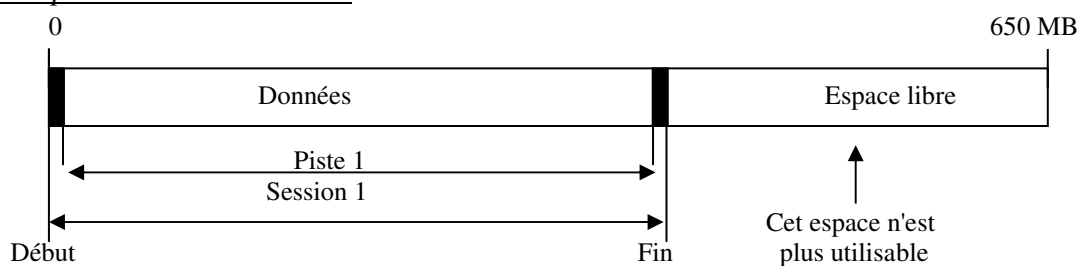
CD-photo : format inventé par kodak permettant de stocker jusqu'à 100 photos de très haute résolution (3072x2048 pixels).

b-2-Modes d'écriture des CD

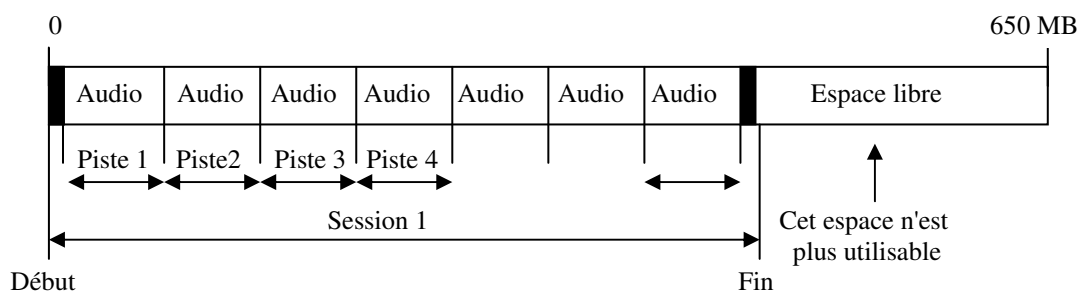
Disc-at-once (DAO)

Ce mode est utilisé lors de l'écriture d'un disque complet en une seule fois et sans pause. Aucune donnée ne peut être ajoutée par la suite, même si la capacité complète du disque vierge n'a pas été utilisée.

Exemple : CD-ROM courant



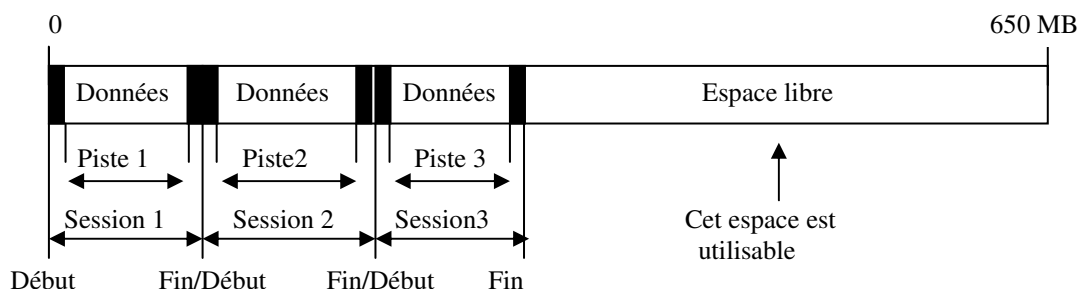
Exemple: CD audio



Remarque: les zones de début et de fin ne font pas partie des données, mais contiennent des informations sur la session.

Track-at-once (TAO)

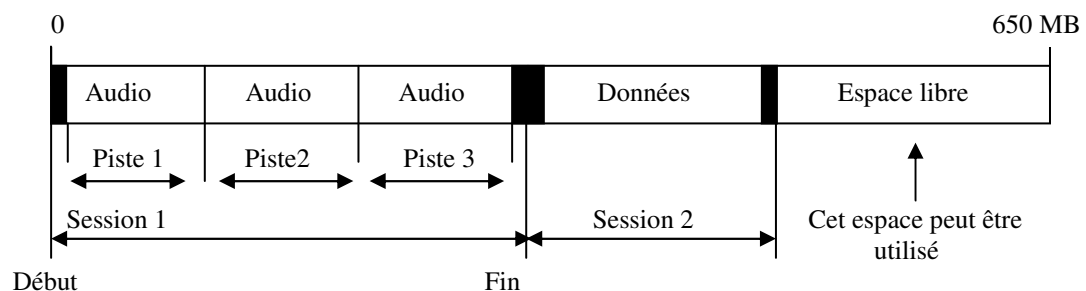
Ce mode est utilisé lors de l'écriture de données sur un disque piste par piste. D'autres piste peuvent être ajoutées par la suite s'il reste suffisamment d'espace sur le disque. C'est pourquoi le mode track-at-once est quelquefois appelé **Multisession**.



Remarque: Seule la première session d'un disque multisession peut être lue sur un lecteur de Cd audio conventionnel (lecteur de salon ou balladeur). Les disques *monosession* sont lus intégralement.

Session-at-once (SAO)

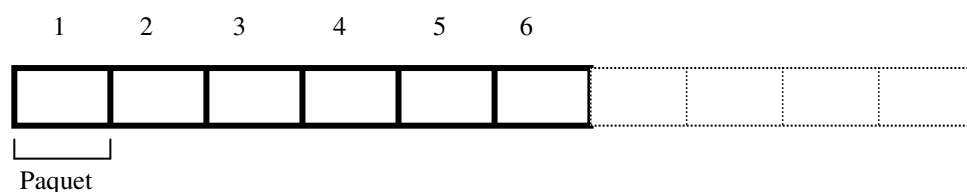
Ce mode est utilisé lors de l'écriture de chaque session en une seule fois et sans pause. D'autres sessions peuvent être ajoutées par la suite s'il reste suffisamment d'espace sur le disque.



Remarque: avec les CD-Extra, l'audio est écrit lors de la première session, et les données lors de la deuxième. 99 pistes au maximum peuvent être écrites.

Packet Writing

Ce mode est utilisé lors de l'écriture, sur la piste d'un disque, de petits blocs de données appelés *paquets*, comme pour les disquettes ou les disques durs. Ce mode est utile lorsqu'on fait de petites sauvegardes incrémentales de données. Il faudra pour cela utiliser un logiciel supportant expressément le mode écriture par paquets (comme le Direct CD de la firme Adaptec par exemple).



Remarque: les disques doivent être formatés avant de pouvoir être utilisés pour l'écriture par paquets. Avec les CD-RW, les données effacées peuvent être remplacées tant que le disque n'est pas plein. Avec les CD-R, l'espace utilisé par les données effacées ne peut être réutilisé, et il est masqué de manière à ne plus être visible.

b-3-Le dvd (digital versatile disc ou disque digital multi-usage) :

De même taille et de même technologie de base (le laser) que le cd, la fréquence d'enregistrement est double et la capacité de 6 à huit fois plus grande. Pour les dvd rom et vidéo les standards sont aujourd'hui stabilisés, en raison de la réunion de 1997 où les ténors mondiaux (Hitachi, Matsushita, Jvc, Sony, Pioneer, Philips, Thomson,...) se sont entendus sur un standard, et ont créé le « dvd forum » comme organe de consultation et de normalisation. Par contre pour les dvd réinscriptibles il n'existe aucun standard qui se dégage, si ce n'est qu'on s'achemine vers la coexistence de trois normes : dvd-ram, dvd+rw et dvd-rw.

Quant à la vitesse, le taux de transfert unitaire (1X) diffère par rapport à celui des cd en raison du volume de données à transférer. Ici 1X signifie 1,4 Mo/s (contre 0,15 Mo/s).

Lecteurs de dvd :

Les lecteurs de deuxième ou troisième génération sont « multiread », c'est à dire qu'ils permettent de lire les dvd-rom et dvd-vidéo, ainsi que les cd-rom et cd-audio. Ils sont presque tous à la norme atapi (standard de transfert de données pour les périphériques, qui sont automatiquement reconnus par l'ordinateur) et peuvent donc être branchés sur un connecteur EIDE, SCSI, USB ou parallèle d'un ordinateur. De la même manière que pour les cd-rw, la compatibilité avec le format UDF (universal disc format) permet de lire et d'écrire des données sur un média par simple « glisser/déplacer » de la souris.

La décompression d'un fichier mpeg-2 est effectuée par un circuit spécialisé qui reconstruit les séquences d'image et les convertit en signaux vidéo. Il faut donc, en plus du lecteur de dvd, une carte de décompression mpeg-2 à brancher sur un bus PCI de l'ordinateur. Toutes les cartes mpeg-2 sont munies d'une entrée vidéo (à relier à la sortie svga de la carte graphique de l'ordinateur), d'une sortie svga (à relier au moniteur) et d'une sortie s-vidéo (à relier à un téléviseur). Quant à la partie son, elle dispose également d'une entrée marquée cd (à relier à la sortie son du lecteur de dvd), et d'une sortie marquée audio (à relier à l'entrée audio de la carte son).

Dvd-rom : en simple ou en double face, il permet une capacité de 4,7 à 17 Go. On peut y stocker tout document numérisé (logiciels, texte, vidéo compressée au format mpeg-2).

Dvd-vidéo : il permet de stocker des séquences vidéo numérisées au format mpeg-2, qui est le premier procédé de compression vidéo numérique en qualité « broadcast » (conforme aux exigences de la qualité de diffusion tv). On peut y stocker au moins un film de 135 mn avec la bande son en stéréo en plusieurs langues, ou la bande son «dvd », c'est à dire simultanément en trois formats différents : stéréo de base (accessible à toutes les cartes son

standards), mpeg-2, et enfin AC3. Le format AC3 autorise huit bandes son distinctes par séquence, car on utilise la technique de codage du son dite multivoies (ou norme 5.1), où chacun des huit canaux numériques est indépendant des 7 voisins. On peut ainsi reconstituer la bande son en « qualité cinéma », par l'intermédiaire de six (5+1) hauts parleurs : gauche, droit, central, surround gauche, surround droit, et enfin le caisson de graves extrêmes pour les effets sonores complexes (chocs et explosions).

Lecteurs enregistreurs de dvd :

Le dvd ram est fabriqué depuis au moins 3 ans alors que les autres dvd commencent à peine à apparaître sur le marché. Il est à remarquer que pour les dvd +rw l'appellation dvd a été refusée par le « dvd forum ».

Si aujourd'hui chaque groupe de constructeurs propose ses dvd avec une capacité propre, dans une année les dvd réinscriptibles disponibles sur le marché auront une capacité normalisée de 4,7 Go par face (soit 9,4 Go en double face).

DVD R : il est conçu pour être enregistré seulement une fois. C'est le successeur naturel du cd-R. Actuellement de capacité 3,95 Go, les dvd de 4.7 Go attendent leur normalisation. Le premier fabricant à avoir proposé ce type de support à la fin de l'année 1999 est Pioneer.

DVD RAM : soutenu par trois constructeurs japonais (hitachi, toshiba et matsushita) réunis sous la marque panasonic, ce support a une capacité de 2, 6 Go par face en simple couche et 5,2 Go en double couche. Un dvd ram est constitué de deux couches de métal recouvertes d'une enveloppe en plastique, le tout enfermé dans une cartouche en plastique rigide. L'écriture d'une donnée se fait par un laser qui chauffe le métal à environ 600°C, un deuxième laser plus faible chauffe la surface à 200°C pour modifier les données.

C'est une technologie particulière dans la mesure où les disques sont enfermés dans une cartouche et que l'on utilise la technique dite de changement de phase (semblable à celle des disques magnéto optiques ?). Les cartouches de type II (2,6 Go) permettent l'extraction du disque de sa cartouche pour être lu sur un lecteur de dvd rom de la prochaine génération. Actuellement un dvd ram ne peut pas être lu sur un simple lecteur de dvd rom ou dvd vidéo, en raison de la faible réflectivité due au changement de phase, ainsi qu'à la structure physique du média. La prochaine génération de lecteurs de dvd rom est prévue pour lire les dvd ram.

Les lecteurs enregistreurs de dvd ram permettent de lire tous les disques antérieurs : Cd rom, cd vidéo, cd r/w, cd audio, dvd rom, dvd vidéo, et cartouches PD de Matsushita.






































































DVD +RW : Bien que non reconnu par le dvd forum, ce format qui propose actuellement des disques de 3 Go (bientôt 4,7 Go) est soutenu par un groupe d'industriels puissants et influents (Sony, Philips, Mitsubishi, Hewlett-Packard, Yamaha, Ricoh ...), qui contrôlent les trois quarts du marché du Cd-RW. Ce support a été conçu comme successeur du cd-RW.

DVD RW : proposé par pioneer, ce format a été retenu par le dvd forum (il y a un an et demi) comme successeur possible du dvd ram. Non encore disponible sur le marché, il aura une capacité de 4,7 Go par face et sera compatible avec le parc de lecteurs existant.

Remarque : autres moyens de stockage

Les solutions de stockage exposées jusqu'ici sont de loin les plus répandues sur le marché. Cependant l'évolution des technologies des semi-conducteurs aidant, on peut maintenant trouver des solutions de stockage de plus en plus miniaturisées destinées aux ordinateurs portables et à l'électronique « nomade » (console de jeu, appareil photo, téléphone cellulaire, caméra,...). Mais le plus souvent elles nécessitent un périphérique spécifique pour être lues sur un ordinateur.

On peut citer à titre indicatif la PC card flash qui se présente sous la forme d'une carte PCMCIA et abrite de la mémoire flash de 48 Mo, le datapak de Kingston qui a une capacité de 2 Go (au format pc card de type III), le microdisque dur d'IBM de 340 Mo enchâssé dans un boîtier de 42,8 mm x 36,4 mm et 55 mm d'épaisseur, et qui ne pèse pas plus de 20 grammes. Son unique plateau de 16 cm² est plus petit qu'une pièce de 2 francs (le 1Go serait en préparation). On peut citer également le compact flash, le SmartMedia, le memory stick de Sony, qui sont très utilisés dans les appareils photos, et dont les capacités atteignent 64 Mo.

TABLEAU DES COMPATIBILITES CD & DVD		Disques	DVD- Rom	DVD– Vidéo	DVD-R	DVD- Ram	DVD- RW	CD- Rom	CD- Audio	CD-R	CD-RW	CD- Vidéo	CD-Extra
		Capacité	4,7 à 17	4,7 à 17	7,9	5,2	3	0,65	0,65	0,65	0,65	0,65	0,65
		Contenu	Données en lecture	Vidéo Mpeg-2 en lecture	Données en gravure simple	Données en gravure multiple	Données en gravure multiple	Données en lecture	Audio en lecture	Données en gravure simple	Données en gravure multiple	Vidéo Mpeg-1 en lecture	Audio et données en lecture
Lecteurs	Taux de transfert	Temps d'accès											
DVD-Rom 2X	4,7Mo/s	150ms											
DVD-Rom 8X	4,7Mo/s	150 ms											
DVD–Vidéo	9,8 Mo/s	150 ms			rarement								
DVD-R	11 Mo/s ?	630 ms ?											
DVD-Ram	5 Mo/s	120ms											
DVD-RW	n.c	n.c											
DVD-Audio	n.c	n.c											
CD-Rom 50X	7,5 Mo/s	70 ms								Rareme nt			
CD-Audio	150 Ko/s	70 ms											
CD-R (8 X)	1,2 M0/s	120 ms											
CD-RW (4 X)	600 Ko/s	120 ms											

TD CHAPITRE 3 : « LES MEMOIRES »

EXERCICE 1:

Si un ordinateur possède une mémoire de 1Mo, quelle la taille de sa M.C. exprimée :
1° en mots de 32 bits? 2° en mots de 64 bits ?

EXERCICE 2:

Trouver le nombre de bits nécessaire pour adresser une mémoire de 512 mots de 16 bits chacun.

EXERCICE 3:

On considère une mémoire de 65536 mots de 25 bits chacun. Donner le nombre de fils nécessaires sur le bus d'adresses et sur le bus de données.

EXERCICE 4:

Les adresses d'une mémoire désignent des bytes (octets). Quelle est l'adresse du quatrième élément d'une table qui commence à l'adresse (0019)_H et qui est constituée d'éléments de 16 bits ?

EXERCICE 5:

Un emplacement mémoire de longueur 16, a le contenu suivant (stocké dans la technique Big Endian):

0011 1000 0011 0101. Quelle est sa signification s'il s'agit:

1° d'un entier en représentation binaire? 2° d'une chaîne de caractères en code ISO?

EXERCICE 6:

La mémoire d'un ordinateur est organisée en mots de 32 bits. On veut stocker la chaîne de caractères "EPSIMA" et le nombre entier (14389)₁₀. Donner la disposition des chaînes de bits correspondantes (en code ISO) dans les deux techniques "Big Endian" et "Little Endian".

EXERCICE 7:

Quelle est l'adresse effective correspondant à une adresse de base de code hexadécimal (0019)_H et un déplacement de (0016)_H ?

EXERCICE 8:

On dispose de boîtiers mémoire de capacité 8Kx1byte, chacun étant muni d'une broche de sélection. Montrer comment on peut construire un module de mémoire de 64Kbytes, en faisant clairement ressortir le câblage des bus d'adresses et de données.

EXERCICE 9:

1°Mêmes questions que pour l'exercice 8 avec boîtier = 4Kx4bits, et taille mémoire = 8Ko.

2°Mêmes questions que pour l'exercice 8 avec boîtier = 8Kx8bits, et taille mémoire = 64Kmots de 16 bits.

3°Mêmes questions que pour l'exercice 8 avec boîtier = 4Kx2bits, et taille mémoire = 8Ko.

EXERCICE 10 : (de consolidation !) *Toutes les questions sont indépendantes.*

Considérons deux ordinateurs : A avec une mémoire organisée en octets, et B avec une mémoire organisée en mots de 32 bits.

Soit la chaîne de caractères « DIMANCHEsp23spJANVsp00? » (*sp* signifie space = le caractère espace).

1°) Donner le nombre d'**emplacements-mémoire** nécessaires pour stocker la chaîne de caractères codée en ASCII, dans les cas des ordinateurs A et B.

2°) Si le bus d'adresses est à 8 fils, quelle est la taille de l'**espace adressable** dans chaque cas ?

3° Si le bus de données est à 8 bits, quel est dans chaque cas le nombre d'**opérations d'échange** entre le processeur et le contrôleur de la mémoire, pour stocker en mémoire la chaîne de caractères codée en ASCII ?

4°) Si le bus de données est à 32 bits, même question que la question 3°).

5°) On veut stocker le message codé en ASCII en mémoire de l'ordinateur A, donner la disposition en mémoire des chaînes de bits correspondantes dans les deux techniques "Big Endian" et "Little Endian".

6°) Pour l'ordinateur B, même question que 12°)

7°) Donner pour les codes ISO et Baudot, le nombre de bits total nécessaire pour coder ce message.

8°) Si le message est transmis en mode asynchrone (1bit parité, 1bit start, 1bits stop), donner dans chacun des 2 cas (ISO et Baudot) le nombre de bits total reçus par l'imprimante.

EXERCICE 11:

On considère un ordinateur disposant d'une antémémoire de temps de cycle 100ns. Sa mémoire centrale a un temps de cycle de 600ns.

1° Quel est le temps de cycle apparent de la mémoire si on suppose un taux de réussite de 80%?

2° Quel est le gain relatif par rapport à une architecture sans antémémoire?

EXERCICE 12:

On considère un système de pagination, la table des pages est en M.C. qui a un temps de cycle de 750ns. Le temps d'accès au tampon de traduction d'adresses est de 50ns. La probabilité de trouver l'adresse cherchée dans le tampon (taux de réussite) est de 80%.

1° Quel est le temps d'accès moyen à une donnée se trouvant en M.C. ?

2° Quelle est la dégradation (perte de vitesse) par rapport à un système sans pagination.

EXERCICE 13 :

On considère un système de mémoire virtuelle basé sur la pagination. La table des pages est en mémoire centrale (ram). De plus l'ordinateur possède un tampon de traduction d'adresses. Donner les différentes séquences d'accès aux mémoires qui peuvent être nécessaires, sachant que :

- si une traduction d'adresse de page est dans le tampon de traduction, alors la page est obligatoirement présente en mémoire centrale ;
- dans la table des pages ne se trouvent que les adresses de pages virtuelles se trouvant en ram.

EXERCICE 14 :

On considère maintenant un système de mémoire virtuelle basé sur la pagination. La table des pages est en mémoire centrale (ram). De plus l'ordinateur possède un tampon de traduction d'adresses et une mémoire cache. Donner les différentes séquences d'accès aux mémoires qui peuvent être nécessaires, ainsi que quelques séquences qui ne se produiront jamais, sachant que :

- si une traduction d'adresse de page est dans le tampon de traduction, alors la page est présente en mémoire centrale ;
- si un bloc de mémoire est dans la mémoire cache, alors la traduction d'adresse de sa page est obligatoirement dans le tampon de traduction ;
- dans la table des pages se trouvent toutes les adresses de pages virtuelles se trouvant en ram.

EXERCICE 15:

Une unité de disque dur possède 4 plateaux (disques) ayant les caractéristiques suivantes: une tête de lecture/écriture par disque, une seule tête fonctionne à un instant donné, 600 pistes par disque, 512 octets par secteur, vitesse de rotation de 3000 tours par minute.

- 1° Quelle est la capacité de stockage de l'unité de disque?
- 2° Quel est son taux de transfert?
- 3° Quel est le temps de transfert d'un secteur ?

EXERCICE 16:

On veut calculer le temps moyen de transfert en mémoire centrale d'un fichier séquentiel stocké sur disque dur. Les échanges entre l'unité de disque et la M.C. s'effectuent par l'intermédiaire d'un canal. Une zone de mémoire tampon associée au disque reçoit les caractères lus sur le disque; le transfert vers la M.C. s'effectue soit quand la zone tampon est pleine, soit quand le fichier a été entièrement lu. Les caractéristiques sont les suivantes: Temps moyen de positionnement de la tête de lecture/écriture sur une piste: 20 ms; *Temps de dépôt ou de soulèvement de la tête de lecture écriture: 5 ms ; * Vitesse de rotation du disque: 6000 tr/mn; *Vitesse de transfert du canal: 10 Mo/s; * Taille du tampon: 5000 octets; * Taille du fichier 3000 caractères; *Codage d'un caractère: 8 bits; *Tous les caractères du fichier sont sur un même cylindre; *Nombre de cylindres: 128; *Nombre de secteurs par piste: 50; *Taille d'un secteur: 1024bits dont 24 sont réservés pour un pointeur vers le secteur suivant.

- 1° Quel est le nombre de secteurs nécessaire au stockage du fichier?
- 2° Quel est le temps moyen pour lire un secteur, en supposant que la tête est sur la bonne piste?
- 3° Quel est le temps moyen de transfert de ce fichier entre le disque dur et la M.C. ?
- 4° Même question que 3° si le fichier a une taille de 6000 caractères.
- 5° Même question que 3° si le fichier de 3000 caractères est stocké sur 3 cylindres différents
- 6° Même question que 3° si le fichier de 3000 caractères est toujours stocké sur 1 seul cylindre, et la taille du tampon est de 500 octets.

CHAPITRE IV : L'UNITE CENTRALE DE TRAITEMENT ou CPU**-I- ARCHITECTURE STANDARD D'UN CPU****1° Notion d'architecture**

On désigne par ce terme tous les aspects concernant :

- la structure : interconnexion entre les divers composants matériels,
- l'organisation : gestion des unités fonctionnelles et leur interaction dynamique,
- la réalisation : détail de la construction de chaque élément,
- le fonctionnement : décrit le comportement du système complet, y compris ses performances.

L'utilisateur ordinaire ne s'intéresse qu'au fonctionnement du processeur, et à un degré moindre à l'organisation interne des différents composants quand il doit effectuer une programmation en assembleur. Seul l'utilisateur devant effectuer une conception ou une réalisation matérielle peut s'intéresser aux deux autres aspects.

2° Principe du processeur

L'unité centrale est à la fois le cœur et le cerveau de l'ordinateur. Elle est constituée du processeur central de traitement (CPU : central processing unit) et de la mémoire centrale. Les données à traiter et les programmes à exécuter par le processeur sont stockés en mémoire centrale. Le processeur est capable d'interpréter (décoder) et d'exécuter les instructions des programmes.

Compte tenu des progrès de la technologie VLSI (very large scale integration) qui a atteint des degrés d'intégration très élevés (plusieurs millions de transistors sur un centimètre carré), on a pu intégrer dans le processeur beaucoup de composants qui auparavant étaient réalisés à part (coprocesseur arithmétique, mémoire cache de niveaux 1 et 2, module de calcul 3D,...). On obtient donc en bout de course une très forte miniaturisation, d'où le nom de *microprocesseur*.

On a l'habitude de distinguer à l'intérieur du cpu plusieurs unités fonctionnelles qui n'ont pas obligatoirement une existence physique. On distingue trois entités (ou groupes d'éléments) :

-**l'unité arithmétique et logique** (ALU) qui effectue les différents traitements : opérations de test, opérations arithmétiques, opérations logiques ;

-**l'unité de contrôle** ou de commande (CU) qui est le véritable chef d'orchestre de l'ordinateur. C'est elle qui va chercher successivement les instructions dans la mémoire, les interprète (décode), et selon le résultat de cette interprétation, ordonne à l'alu et aux autres organes (mémoire centrale, unités d'entrée sortie, éléments du processeur) d'effectuer chacun le travail qui le concerne (en leur envoyant des signaux par activation des lignes physiques nécessaires), pour obtenir le résultat final qui est la bonne exécution de l'instruction.

Il est à remarquer que l'utilisateur n'a pas du tout à se préoccuper du travail de l'unité de commande;

-**les registres** qui constituent la mémoire locale et privée de niveau zéro du processeur (la mémoire cache interne du processeur constituant le niveau 1). Ce sont des unités de mémorisation très rapides et de capacité limitée et figée appelée longueur du registre. Ils servent essentiellement à la mémorisation temporaire des informations, et certains comme le compteur ordinal (PC : Program Counter) ou le pointeur de pile (SP : stack pointer) ont une fonction bien précise. Nous n'étudierons que les registres accessibles à l'utilisateur par programmation.

3° Structure et organisation interne du processeur

Les informations circulent à l'intérieur du processeur sur des bus internes. Ces derniers sont reliés aux bus de données et d'adresses de l'ordinateur par le biais de deux registres particuliers appelés **buffers de données et d'adresses**, qui ont la même taille que les bus respectifs auxquels ils sont reliés.

Les registres sont regroupés sur le dessin (figure 4-1) de manière fonctionnelle. D'un côté les éléments qui interviennent directement dans les opérations arithmétiques ou logiques : l'unité arithmétique et logique (**ALU**), l'accumulateur (**ACC**) et le registre des indicateurs (**PSW**). De l'autre côté on trouve les éléments qui interviennent directement dans la lecture et le décodage des instructions du programme : l'unité de commande (**CU**) et le registre d'instructions (**RI**). Au milieu on trouve les registres auxiliaires (**R1 à Rn**) et tous les registres qui ont une fonction spécifique: le compteur ordinal (**PC**), le pointeur de pile (**SP**), le registre d'index (**X**), le **multiplexeur** (qui est relié aux différents chemins de données et d'adresses).

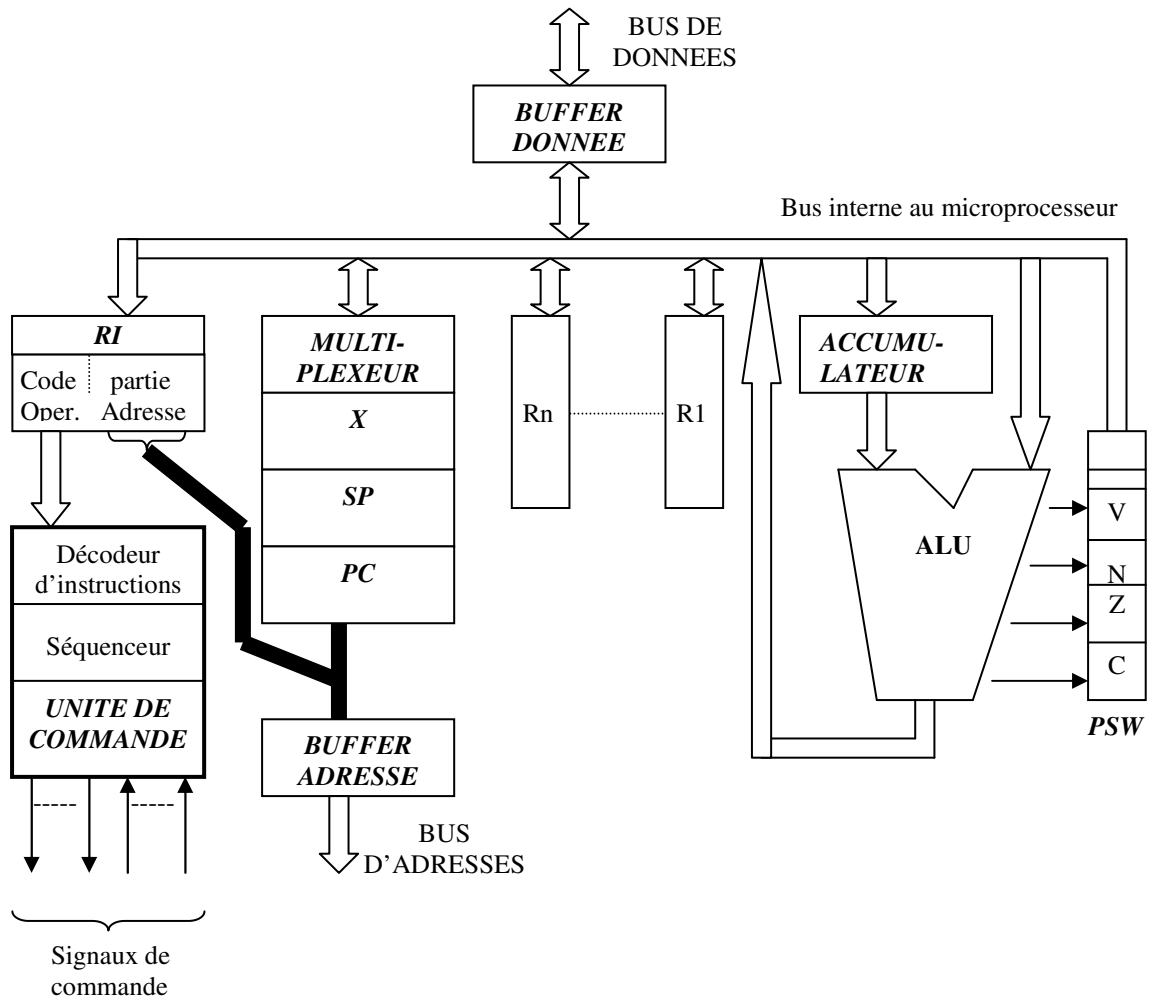


Figure 4-1 : architecture standard d'un microprocesseur

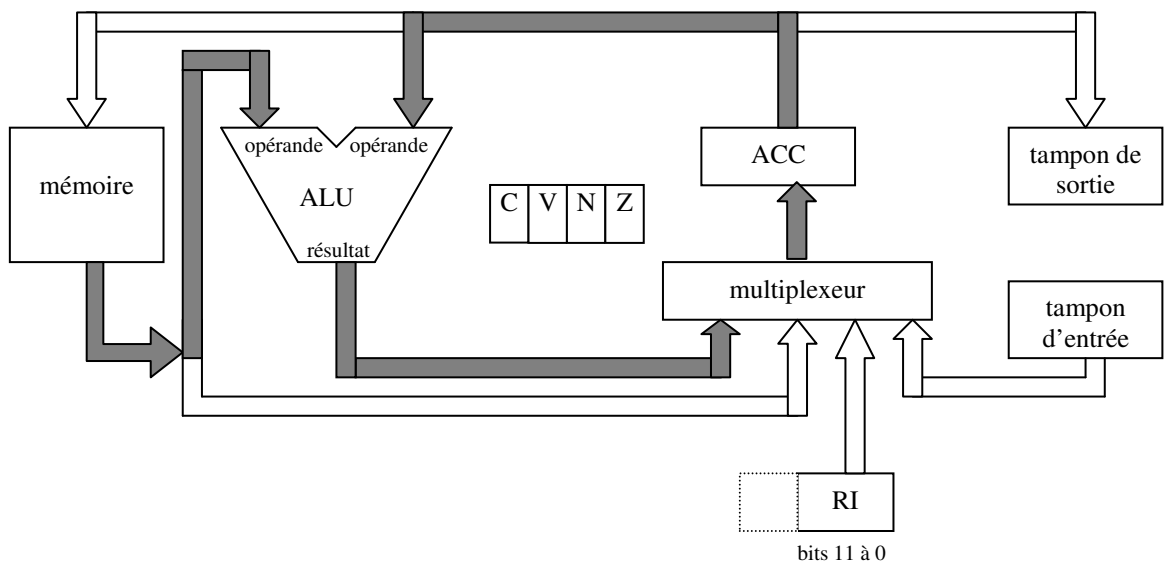


Figure 4-2 : chemins de données de Mimosa

4° Les registres

Les registres constituent la seule partie du processeur accessible à l'utilisateur, on peut y accéder par des instructions machine. On distingue les registres visibles et les registres non visibles. Pour expliquer la structure et l'organisation du processeur telles qu'elles sont vues par le programmeur, il suffit de décrire les registres visibles. Ce n'est que dans le cas où l'on veut étudier la réalisation et le fonctionnement interne du processeur que l'on s'intéresse aux registres non visibles.

Un registre a une longueur (capacité) comprise entre 1 et 128 bits, avec des valeurs typiques de 8, 16, 32 ou 64 bits.

4°a- L'accumulateur

Il sert pour toutes les opérations arithmétiques et la plupart des opérations logiques. Il contient un opérande au début de l'opération, puis le résultat à la fin de l'opération. Sur la figure 4-2 on peut voir en gras le chemin des données pour une opération arithmétique. Un des opérandes arrive à l'ALU par un bus interne, en venant de l'accumulateur. Le deuxième opérande vient de la mémoire par le biais du bus de données de l'ordinateur, rentre dans le processeur à travers le buffer de données, pour atteindre enfin l'ALU à travers un bus interne. Une fois que l'ALU a effectué le travail, le résultat arrive à l'accumulateur via un bus interne et le multiplexeur.

Il est à remarquer que toutes ces opérations s'effectuent sous le contrôle de l'unité de commande, qui envoie (dans un ordre précis) les signaux pour chaque élément afin que cette opération arithmétique s'effectue correctement.

L'accumulateur est également un passage obligé (zone de transit) pour toutes les opérations entre la mémoire et les unités d'entrée sortie (cf figure 4-2).

Enfin il peut servir de compteur ou de décompteur, car son contenu peut être incrémenté ou décrétementé par des instructions spécifiques.

Remarque : Comme l'accumulateur est appelé à recevoir des données, il a la même taille que les autres registres de travail. Cette taille est en général égale à celle du bus de données. Compte tenu de l'importance de ce registre, certains processeurs possèdent deux accumulateurs (voire plus).

4°b-Le multiplexeur

Le multiplexeur est relié aux différents chemins de données et d'adresses. Il permet d'effectuer un aiguillage pour la sélection du chemin que doivent emprunter les informations, compte tenu du fait que tous les registres sont reliés aux bus (cf figure 4.3).

4°c-Les registres auxiliaires

Ils sont plus ou moins nombreux selon le processeur. Ce sont des zones de stockage temporaire de l'information pour l'utilisateur.

4°d-Le compteur ordinal : PC

Il contient l'adresse de la prochaine instruction à exécuter.

Son contenu est donc envoyé au registre adresse de la mémoire de l'ordinateur en empruntant le chemin : bus interne, multiplexeur, buffer d'adresses, et enfin le bus d'adresses de l'ordinateur (cf figures 4-1 et 4-3). Une fois que le contenu du PC est déposé sur le bus d'adresses, l'unité de commande donne l'ordre de lecture à la mémoire. Le contenu de l'adresse (qui correspond en fait au code d'une instruction) se retrouve automatiquement dans le registre d'instruction.

L'exécution d'une instruction mémoire (dont l'adresse est passée par le PC) incrémente automatiquement le PC (appelé également compteur de programmes).

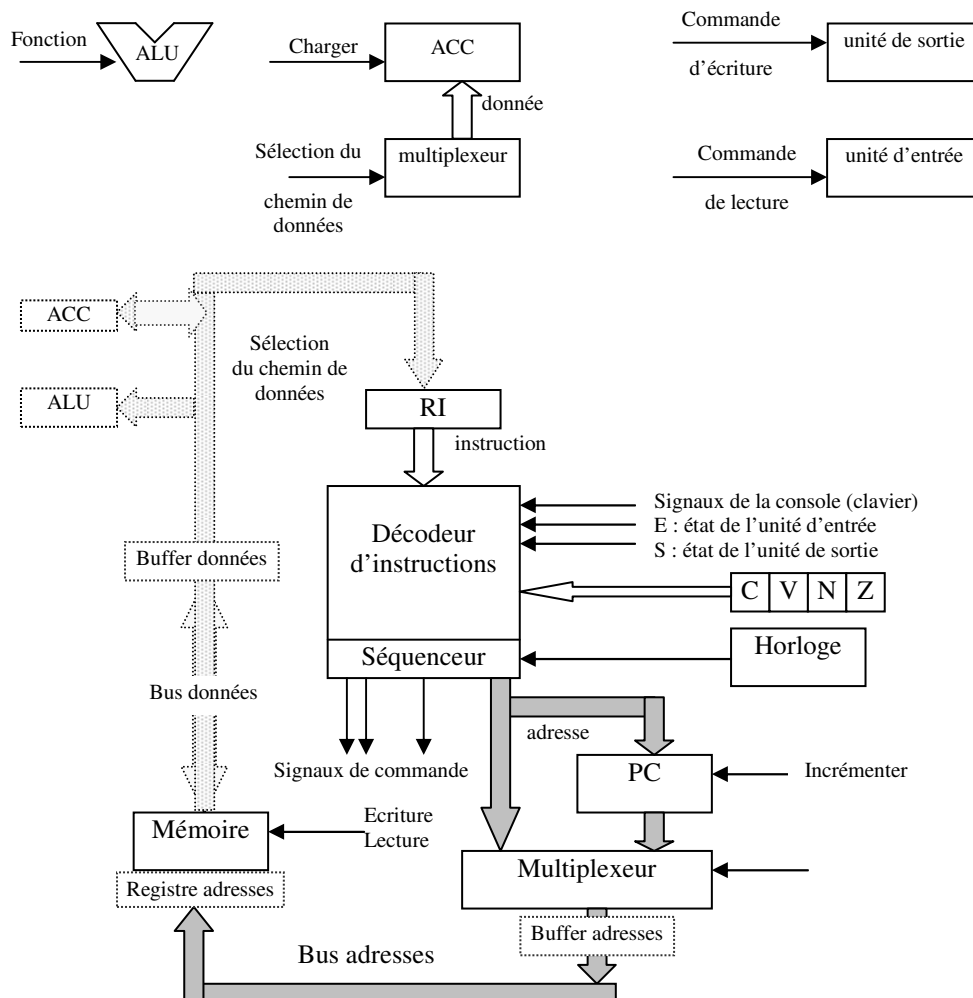


Figure 4-3 : chemins d'adresses, lignes de commande et d'état de Mimosa

4°e-Le registre d'instructions : RI

Une fois que l'instruction arrive au registre d'instruction RI, le décodeur décode (ou interprète) l'instruction. Le décodage permet de reconnaître l'opération à effectuer (addition, soustraction, test,...) et des signaux de commande seront alors activés par l'unité de commande (cf figure 4-3) pour permettre l'exécution de l'instruction. En cas de besoin le processeur (en réalité l'unité de commande) retournera en mémoire pour chercher un autre opérande, comme par exemple dans le cas des instructions de lecture mémoire ou d'addition du contenu de l'accumulateur avec le contenu d'une adresse mémoire (cf exemple figure 4-6).

Remarque : ce registre est souvent considéré comme faisant partie de l'unité de commande.

4°f-Le registre index : X

Il permet d'effectuer l'adressage indexé (cf paragraphe III-3-4). Certains processeurs peuvent avoir plusieurs registres d'index.

4°g-Le registre des indicateurs PSW (processor status word)

Ce registre rassemble un ensemble de bits ou indicateurs (cf paragraphe III-2) qui donnent des informations sur le résultat de l'exécution d'une instruction. Parmi ses bits les plus utilisés on trouve le bit C (carry), le bit de débordement V (overflow), le bit Z (zéro) et le bit N (signe).

4°h-Le pointeur de pile : SP (stack pointer)

Il donne l'adresse de la prochaine position mémoire libre sur la pile. La pile est une petite partie de la mémoire RAM utilisée pour sauvegarder temporairement le contenu des registres, lorsque cela est nécessaire (par exemple pour sauvegarder l'adresse de retour lors d'un appel de sous programme ou d'une interruption). L'adresse de la prochaine position mémoire à utiliser pour cette sauvegarde est donnée par le registre SP (cf paragraphe III-1).

Remarque : il existe des piles lifo (last in first out) où la dernière information écrite en mémoire est la première à sortir, et des piles fifo (first in first out) où la première information écrite en mémoire est la première à en sortir.

5° Les signaux d'entrée sortie du processeur

Toutes les lignes des trois bus (données, adresses et commande), ainsi que des lignes d'alimentation et d'horloge sont raccordées au processeur. Contrairement à la figure 4-3 où même des lignes internes d'état et de commande sont représentées, sur la figure 4-4 seules certaines lignes (accessibles à l'utilisateur) nécessaires à la compréhension du fonctionnement du processeur sont représentées.

Parmi les lignes du bus de commande on peut remarquer que certaines partent du processeur pour envoyer des ordres, alors que d'autres arrivent au processeur soit pour la réception d'information soit pour recevoir des requêtes.

Quand le processeur reçoit une demande d'interruption par la ligne INTerrupt-REQuest, il termine d'abord le travail en cours et, s'il est programmé pour répondre aux interruptions, il autorise l'interruption par la ligne INT-ACKnowledge, avant d'exécuter la procédure prévue en cas d'interruption. Il est à remarquer que les processeurs possèdent plusieurs lignes d'interruption.

Pour accélérer certains transferts entre les unités périphériques et la mémoire, certaines unités d'entrée sortie accèdent directement à la mémoire sans passer par le processeur. Pour cela elles émettent une demande d'accès direct à la mémoire (Direct Memory Access) en activant la ligne DMA-REQ. Si le processeur accepte il active la ligne DMA-ACK.

La ligne Reset permet de réinitialiser le processeur, par écriture d'une valeur initiale dans chaque registre du processeur. Cette valeur vaut 00 pour la plupart des registres d'où le terme de remise à zéro.

Remarque: la réinitialisation hard (ou à froid) de l'ordinateur consiste à réinitialiser le processeur et à écrire la même valeur (hexadécimale 00 ou FF) dans tous les emplacements mémoire de la ram.

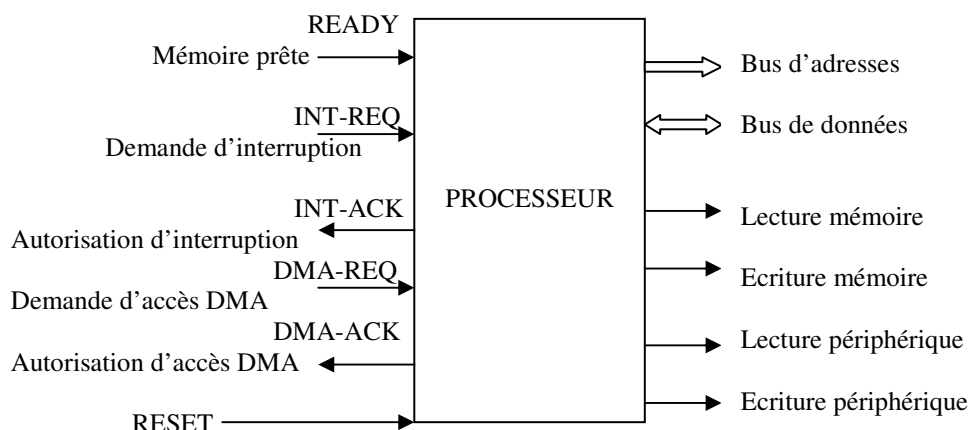


Figure 4-4 : les signaux d'entrée sortie du processeur

ORDINATEUR VIRTUEL MIMOSA

Bus d'adresses : 12 bits

Code opération : 4 ou 8 bits selon que l'instruction nécessite un opérande explicite ou non

Registre d'instructions : 16 bits

Accumulateur : 16 bits

4 Registres auxiliaires 16 bits : R0 , R1 , R2 , R3

Bus de données : 16 bits

Pile : adresses de (000)₁₆ à (0FF)₁₆

Langage assembleur de Mimosa

Code opération	Mnémonique	Opérande ou Adresse	Signification	Indicateurs modifiés
0	SAUTSP	aaa	(Appel de S/P) PC:= aaa ; SP:= (SP) + 1	
1	NOMBRE	aaa	ACC := aaa *	N , Z
2	CHARGER	aaa	Acc := m(aaa) **	N , Z
3	DEC	aaa	Décrémenter le contenu de l'adresse aaa	N,Z,C
4	STOCKER	aaa	m(aaa) := (ACC) **	
5	ET	aaa	ACC := (ACC) and (m(aaa))	Z
6	ADD	aaa	ACC := (ACC) + m(aaa)	N,Z,C,V
7	SUB	aaa	ACC := (ACC) – m(aaa)	N,Z,C,V
8	SAUT	aaa	PC := aaa	
9	SAUT, N	aaa	Si N = 1 alors PC := aaa ***	
A	SAUT , Z	aaa	Si Z = 1 alors PC := aaa ***	
B	INC	aaa	Incrémenter le contenu de l'adresse aaa	C,V,Z
C	SAUT , C	aaa	Si C = 1 alors PC := aaa ***	
D	SAUT , E	aaa	Si E = 1 alors PC := aaa ***	
E	SAUT , S	aaa	Si S = 1 alors PC := aaa ***	
F0	ROG		Rotation à gauche : décalage à gche de (ACC) et mise du Carry dans le bpp faible de ACC	Z,C,V
F1	NEG		ACC := - (ACC)	N,Z
F2	ENTREE		ACC := (tampon d'entrée)	N,Z,E
F3	SORTIE		tampon de sortie := (ACC)	N,Z,S
F4	NOOP		Aucune opération à faire	
F5	HALTE		Fin ; le processeur se met en attente	
F6	RETSP		Retour de sous programme	
F7	PSHA		Pile := (ACC)	
F8	EFFACEAC		Mettre à zéro l'accumulateur	Z
F9	EFFACERN		Mettre à zéro l'indicateur N	N
FA	EFFACERZ		Mettre à zéro l'indicateur Z	Z
FB	ROD		Rotation à dte : décalage à dte de (ACC) et mise du Carry dans le bpp fort de ACC	Z,C
FC	EFFACERC		Mettre à zéro l'indicateur C	C
FD	DECD		Décalage à droite de (ACC)	Z,C
FE	DECG		Décalage à gauche de (ACC)	Z,C
FF	PULA		ACC := (Haut de la Pile)	N,Z

* « := » signifie assignation d'une valeur au contenu d'une adresse ou d'un registre, par exemple PC := 1 ou m(aaa) := 1, signifie mise de la valeur 1 dans le PC ou dans l'adresse aaa

** « () » la mise entre parenthèses d'un registre ou d'une adresse signifie contenu du registre ou de l'adresse

*** En réalité les instructions de saut conditionnel sont des instructions à adressage relatif. Voir chapitre IV-3-3 pour les différents modes d'adressage (page 75)

Remarque importante: Le tableau mentionne les indicateurs d'état qui peuvent être modifiés par l'exécution des instructions. Nous ferons l'hypothèse que les indicateurs N,Z,C, et V ne sont jamais mis à zéro automatiquement. Ainsi par exemple, si l'indicateur N vaut 1 avant l'exécution d'une instruction CHARGER, il vaudra encore 1 à la fin de celle-ci, même si la valeur affectée à l'accumulateur a un signe positif. Il convient donc de faire attention et de mettre, si nécessaire, explicitement les indicateurs à zéro par les instructions EFFACER.

Figure 4-5 : programmation de l'ordinateur virtuel mimosa

-II- FONCTIONNEMENT DU PROCESSEUR

Pour illustrer le fonctionnement du processeur, on se servira du processeur virtuel mimosa (et de l'ordinateur virtuel de même nom) dont la structure est conforme à celle donnée figure 4-1, et dont le langage assembleur est décrit à la figure 4-5. Il possède un bus d'adresses 12 bits. Quant au bus de données et à l'accumulateur (ainsi que l'ALU), ils auront une taille de 8 ou 16 bits en fonction des besoins pédagogiques (donc de la partie du cours à traiter). Les autres registres gardent la même taille indiquée sur la figure 4.5 .

1°Structure d'instruction et adresse mémoire

Partons d'un exercice simple donné sous forme de phrases, et essayons de le transcrire en langage assembleur de mimosa, puis de le stocker en mémoire. On va supposer qu'on a un bus de données 8 bits (donc un ACC 8bits, une ALU qui traite des mots de 8 bits).

1°a-Exemple de programme

- 1-Charger (dans l'accumulateur) la valeur (35)h
- 2-L'additionner avec le contenu de l'adresse mémoire (B4F)h
- 3-Stocker le résultat à l'adresse mémoire (36E)h
- 4-Stop

N°de ligne d'instruction	INSTRUCTION		Code hexadécimal de l'instruction	Commentaires
	Mnémonique	Argument ou Opérande		
1	NOMBRE	35	135	Mettre (35)h dans ACC
2	ADD	B4F	6B4F	Additionner contenu de ACC avec contenu de l'adresse B4F
3	STOCKER	36E	436E	Stocker le résultat (contenu dans ACC) à l'adresse 36E
4	HALTE		F5	Arrêt du programme

1°b-Stockage du programme en mémoire

Pour stocker les données en mémoire, mimosa (comme le processeur 68000 de MOTOROLA) traite les bits de poids le plus fort en premier. Certains processeurs (comme le 8080 et toute la série 80xx de INTEL) traitent les bits de poids le plus faible en premier.

On va supposer que la mémoire est organisée en octets, il s'agit alors de décrire le contenu de la mémoire si le programme est stocké à partir de l'adresse 200.

Adresse mémoire (hexadécimal)	Contenu	
200	13	1°instruction
201	5-	
202	6B	2°instruction
203	4F	
204	43	3°instruction
205	6E	
206	F5	4°instruction

Remarque : on observe que les instructions occupent des nombres d'emplacements mémoire différents: certaines en occupent deux alors que d'autres en occupent un seul. Sur certains processeurs ce nombre varie de 1 à 4.

1°c-Contenu des adresses mémoire affectées aux données

Adresse mémoire	Contenu avant exécution	Contenu après exécution
36E	21	87
B4F	52	52
ACC		87

1°d-Exécution du programme

Essayons de suivre les différentes phases d'exécution de chaque instruction, en précisant à chaque fois les contenus des bus et des différents registres.

Numéro Instruction	Phase de l'instruction	Ligne <u>write</u> memory	Ligne Read memory	PC	Bus adresses	Bus données	RI	ACC
1	1			200	200			
	2		1	201	200	13	13--	
	3			201	201		13--	
	4		1	202	201	5-	135-	
	5			202		5-	135-	35
2	1			202	202			35
	2		1	203	202	6B	6B--	35
	3			203	203			35
	4		1	204	203	4F	6B4F	35
	5			204	B4F			35
	6		1		B4F	52		35
	7							87
3	1			204	204			
	2		1	205	204	43	43--	
	3			205	205		43--	
	4		1	206	205	6E	436E	
	5				36E	87		
	6	1						
4	1		1	207	206	F5	F5--	

Remarques :

Les lignes read et write memory ne fonctionnent pas sur des niveaux mais sur des fronts : ce n'est pas la valeur 0 ou 1 qui provoque un changement d'état, mais l'activation correspond au passage de l'état logique 0 à 1 (pour read) et de 1 à 0 (pour write).

On a décomposé chaque instruction en un certain nombre de phases élémentaires. Même si ce nombre a été exagéré pour plus de clarté, cette décomposition existe effectivement dans la réalité, et le nombre de phases est différent d'une instruction à l'autre.

2°Notions de cycle et état2°a-Définition

Un cycle (ou cycle processeur ou cycle machine) est le temps nécessaire pour effectuer une opération de lecture ou d'écriture.

Chaque cycle comprend deux phases : une phase recherche (fetch) et une phase exécution. Chaque phase comprend plusieurs états qui correspondent chacun à une période de l'horloge de commande. Pour un processeur de fréquence 100 Mhz, la période est donnée par : $T=1/f=10.10^{-9} = 10 \text{ ns}$.

Pour simplifier on a l'habitude de parler de cycle d'instruction, et on dit que chaque cycle comprend deux étapes : une étape recherche de l'instruction et une étape exécution de l'instruction. En fait on devrait parler de micro-instructions, car dans la plupart des cas, chaque instruction du programme comprend plusieurs cycles élémentaires (appelés cycles machine), et chacun d'eux est effectivement constitué d'une phase recherche et d'une phase exécution (cf figure 4-6). Le nombre de cycles machine est différent d'une instruction à l'autre.

2°b-Déroulement d'une instruction : fetch et exécution

Reprenons l'instruction « STOCKER 36E » de l'exemple précédent, et décomposons la en phases recherche et exécution en fonction des périodes d'horloge (figure 4-6).

La première phase recherche amène le code opération de l'instruction (4) dans le registre d'instructions RI. Après décodage, l'unité de commande reconnaît l'instruction d'écriture mémoire, et sait qu'elle doit effectuer une autre phase recherche pour aller chercher la partie basse de l'adresse d'écriture (6E), puis une autre phase recherche pour déposer l'adresse contenue dans le registre RI (36E) sur le bus d'adresses et sortir le contenu de l'accumulateur (87) sur le bus de données, et enfin une phase exécution pour l'écriture mémoire (écriture du contenu du bus de données à l'adresse mémoire indiquée par le bus d'adresses).

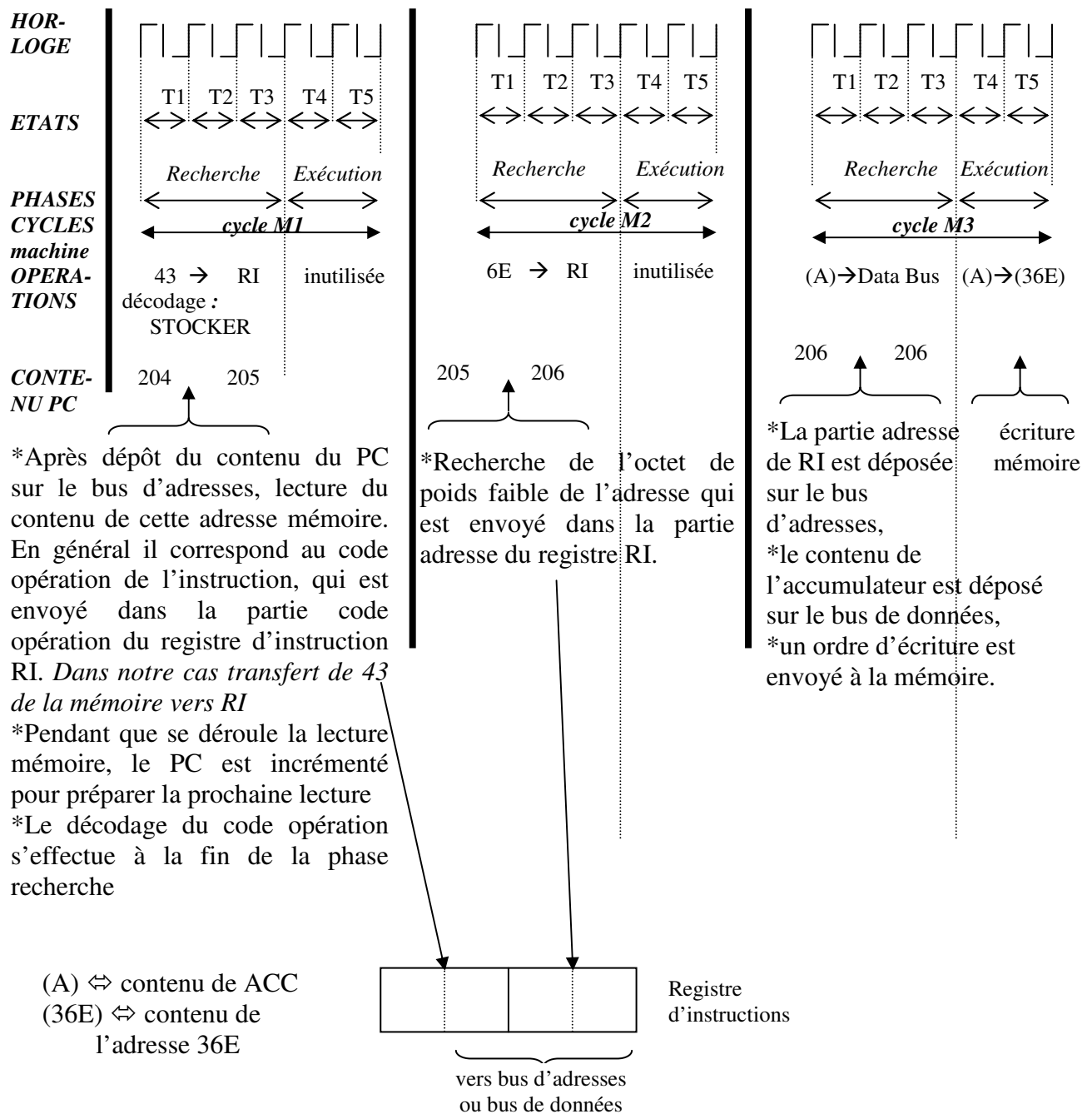


Figure 4-6 :détail de l'exécution de l'instruction « STOCKER 36E » de mimosa

-III- LE LOGICIEL DU CPU

1° Pile et sous-programme

1-a Définition

Une partie de programme exécutée n fois sera extraite du programme principal et utilisée comme sous-programme. Un sous-programme est donc utilisé pour des tâches répétitives.

SAUTSP : est le mnémonique utilisé par mimosa pour effectuer un saut vers le sous-programme,

RETSP: placé à la fin du sous-programme, il permet de revenir au programme qui a effectué l'appel (en général le programme principal).

-1-b-Pile

La pile est une partie de la mémoire ram utilisée pour sauvegarder les contenus des registres, lorsque cela est nécessaire (par exemple pour sauvegarder l'adresse de retour et le contenu du registre d'état, lors d'un appel de sous-programme ou d'une interruption).

L'adresse de la prochaine position mémoire à utiliser pour sauvegarder des données (ou des adresses) est à tout instant donnée par le pointeur de pile: SP (stack pointer).

Il existe des piles LIFO (« last in first out ») et des piles FIFO (« first in first out »). La pile de Mimosa est de type lifo, c'est à dire que la dernière information écrite sur la pile est la première à être lue.

Le pointeur de Pile est décrémenté par une écriture dans la Pile, et incrémenté par une lecture. Les instructions d'écriture et de lecture de Mimosa sont PSHA et PULA.

1-c- exécution de SAUTSP et adresse de retour

L'adresse de retour est mise dans la pile en même temps que s'exécute l'instruction d'appel de sous-programme. L'adresse de retour, qui est celle de la prochaine instruction à exécuter du programme principal (et qui se trouve par conséquent dans le PC à la fin de la phase recherche) est **automatiquement** mise dans la pile.

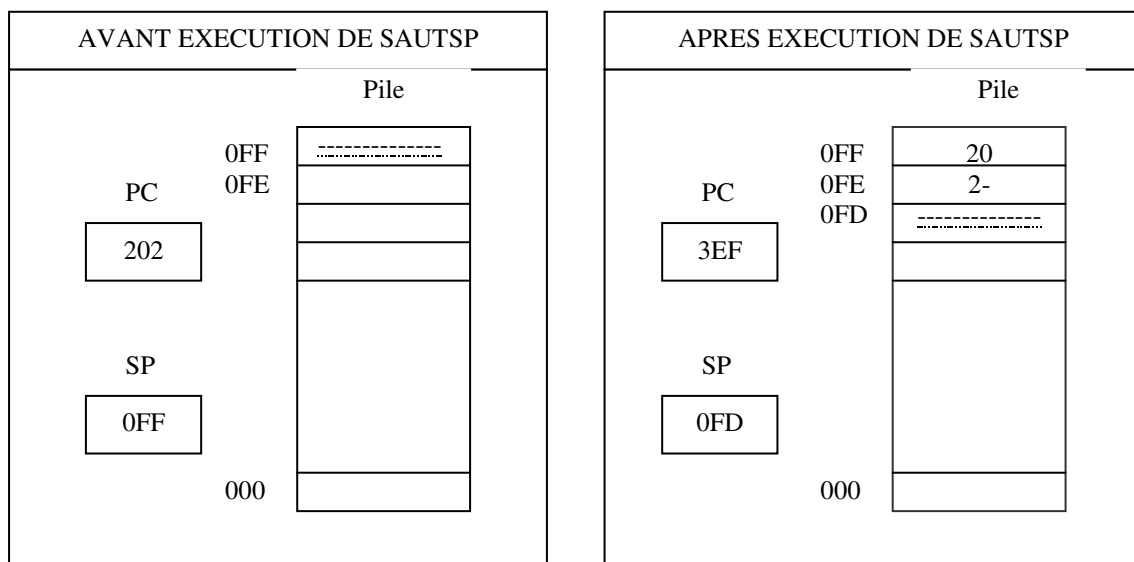
Remarques :

- Comme le SP est appelé à recevoir une adresse, sa taille est égale à celle de PC, soit 12 bits pour mimosa.
- Mimosa traite les octets de poids fort en premier quand on procède à une écriture dans la pile. Quand on lit de la pile on traite les octets de poids faible en premier (car la pile est de type lifo).
- La mémoire de mimosa est organisée en 16 pages de 256 emplacements mémoire chacune (bus d'adresses 12 bits) . La pile de mimosa occupe la page 0, c'est-à-dire les adresses mémoire 000 jusqu'à 0FF.

Exemple : (supposons que Mimosa possède une mémoire organisée en octets)

Soit à exécuter dans un programme l'instruction « SAUTSP 3EF ». Cette instruction est stockée en mémoire à partir de l'adresse 200, et occupe deux emplacements mémoire (càd deux octets). Ce sont les adresses 200 (contenu 03) et 201 (contenu EF). La prochaine instruction est donc stockée à partir de l'adresse 202.

Essayons de représenter les contenus de SP, PC, et la pile, avant et après l'exécution de l'instruction de saut au sous programme.

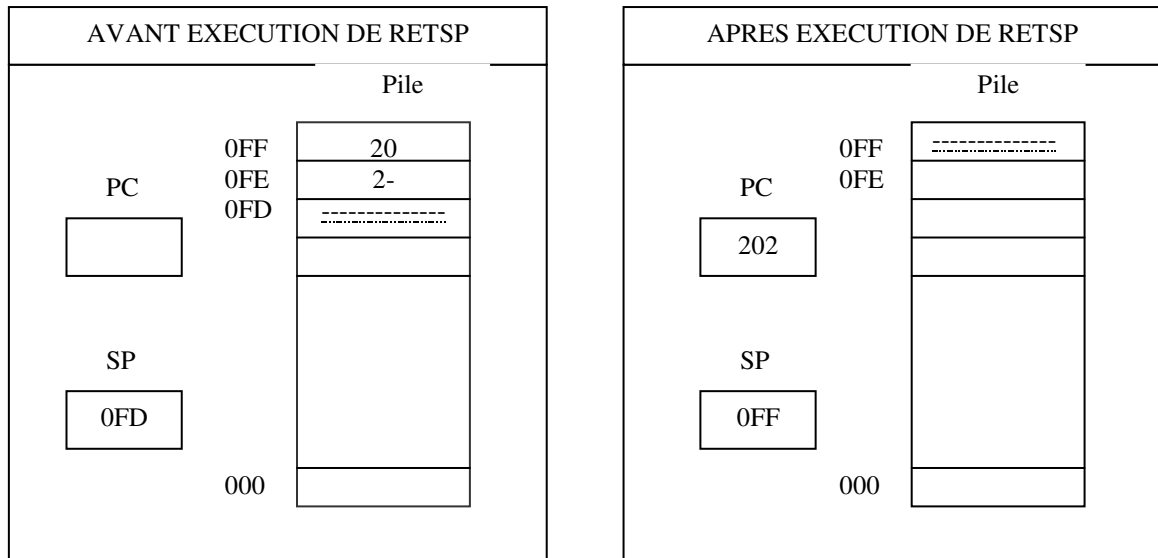


L'adresse de retour , qui est naturellement l'adresse de la prochaine instruction à exécuter du programme principal, et qui se trouve dans le PC, est automatiquement mise dans la pile. On remarque également que le pointeur de pile est décrémenté deux fois (deux écritures dans la pile). Ce qui fait que la prochaine position de libre sur la pile est l'adresse 0FD, et elle est donnée correctement par le pointeur de pile.

1-d- exécution de RETSP

L'exécution de RETSP ramène dans le PC l'adresse stockée dans la pile, lors de l'appel du sous programme, et modifie donc le contenu de SP en conséquence.

Lors de la lecture de la pile, on traite la partie basse de l'adresse en premier.

-1-e-Gestion de la Pile

La gestion de la pile est automatique lors de l'appel ou du retour d'un sous-programme. Mis à part le PC, aucun autre registre n'est sauvegardé automatiquement dans la pile.

Les instructions de Mimosa de sauvegarde et de lecture sont: PSHA et PULA.

-1-f-Exemple d'application

Considérons le programme suivant pour Mimosa (dont la mémoire est supposée organisée en octets). Donner le contenu de la PILE, de SP, de PC et RI, à la fin de la phase recherche et à la fin de la phase exécution de chaque instruction. L'état initial correspond à (PC) = 200; (SP) = 050.

PROGRAMME PRINCIPAL

Numéro Instruction	Adresse mémoire	Mnémonique	Adresse
	200	-----	---
	---	-----	---
1	25C	NOMBRE	2F5
2	25E	PSHA	
3	25F	SAUTSP	400
8	261	PULA	---
	262	-----	---
	---	-----	---
9	350	HALTE	

SOUS PROGRAMME 1

Numéro Instruction	Adresse mémoire	Mnémonique	Adresse
4	400	NOOP	---
	---	-----	---
5	420	SAUTSP	600
	422	NOOP	
	---	-----	---
7	500	RETSP	

SOUS PROGRAMME 2

Numéro Instruction	Adresse mémoire	Mnémonique	Adresse
	600	NOOP	---
	---	-----	---
6	620	RETSP	

SOLUTION :

<u>REGISTRES</u>				<u>PILE</u> (- - signifie l'adresse est vide)							
N°instruction et phase	Contenu PC	Contenu RI	Contenu SP	Adr 050	Adr 04F	Adr 04 ^E	Adr 04D	Adr 04C	Adr 04B	Adr 04A	Adr 049
<i>Etat Initial</i>	200		050	--	--	--	--	--	--	--	--
1 recherche	25E	12F5	050	--	--	--	--	--	--	--	--
1 exécution			050	--	--	--	--	--	--	--	--
2 recherche	25F	F7--	050	--	--	--	--	--	--	--	--
2 exécution			04E	2F	5-	--	--	--	--	--	--
3 recherche	261	0400	04E	2F	5-	--	--	--	--	--	--
3 exécution	400		04C	2F	5-	26	1-	--	--	--	--
4 recherche	401	F4--		2F	5-	26	1-	--	--	--	--
4 exécution				2F	5-	26	1-	--	--	--	--
5 recherche	422	0600		2F	5-	26	1-	--	--	--	--
5 exécution	600		04A	2F	5-	26	1-	42	2-	--	--
6 recherche	621	F6--		2F	5-	26	1-	42	2-	--	--
6 exécution	422		04C	2F	5-	26	1-	--	--	--	--
7 recherche	501	F6--		2F	5-	26	1-	--	--	--	--
7 exécution	261		04E	2F	5-	--	--	--	--	--	--
8 recherche	262	FF--		2F	5-	--	--	--	--	--	--
8 exécution			050	--	--	--	--	--	--	--	--
9 recherche	351	F5--	050	--	--	--	--	--	--	--	--
9 exécution				--	--	--	--	--	--	--	--

-2-Les indicateurs et leur utilisation

Parmi les bits du registre des indicateurs PSW (Processor Status Word), les bits les plus intéressants (car les plus utilisés) sont le Carry (C), le Zéro (Z) et le Signe (N).

Les indicateurs ne sont pas relatifs à l'accumulateur, mais à la dernière instruction exécutée qui a positionné les indicateurs. Il est à remarquer que certaines instructions ne modifient aucun indicateur.

Les indicateurs sont positionnés à un automatiquement, mais ils ne sont jamais positionnés à 0 automatiquement. Il faut donc pour les mettre à 0 utiliser des instructions spécifiques de remise à 0 (cf tableau des mnémoniques de Mimosa figure 4-6).

-2-a-L'indicateur C : CARRY (ou Report ou Retenue)

Il permet d'indiquer par sa valeur logique 1 :

- Que le résultat d'une addition de nombres **non signés** a provoqué un dépassement de capacité : il ne peut tenir sur 8 (ou 16) bits pour une addition 8 (ou 16) bits. C'est alors un report .
- Que la soustraction $N1 - N2$ de deux nombres **non signés** n'est pas possible, ce qui signifie que $N1$ est inférieur à $N2$. C'est alors une retenue.
- La valeur logique d'un bit d'état à tester. Pour ce faire, le bit d'état est transféré dans le Carry par un ou plusieurs décalages du contenu du registre contenant le bit d'état à tester.

-2-b-L'indicateur de signe N

Les nombres signés sont généralement exprimés en complément à 2, le bit le plus significatif étant le bit de signe. Sa valeur est à 0 pour un nombre positif et à 1 pour un nombre négatif.

La valeur 1 de l'indicateur de signe N indique donc qu'il s'agit d'un nombre négatif.

On utilise l'indicateur de signe N pour indiquer que le bit le plus significatif (poids fort) d'un nombre est à 1.

-2-c-L'indicateur de débordement V (ou Overflow)

Cet indicateur est mis à un lorsqu'il y a un dépassement de capacité pour des opérations arithmétiques en **complément à 2**. Ainsi pour des nombres de 8 bits par exemple, le résultat d'une opération arithmétique ne doit pas dépasser -128 ou $+127$. Du point de vue fonction logique, l'indicateur V est le OU-exclusif du CARRY et du SIGNE : $V = C \oplus N$.

-2-d-L'indicateur zéro Z

Lorsque le résultat d'une opération (arithmétique, logique, de chargement...) est nul, cet indicateur prend la valeur logique 1.

Ainsi dans le cas de la comparaison de deux nombres N1 et N2, il suffit de faire la soustraction $N1 - N2$. Si les deux nombres sont égaux, le résultat est nul et l'indicateur Z se met à 1.

Par ailleurs si l'on tient compte de ce qui a été dit sur le carry, l'égalité $N1 = N2$ se traduit par $C=0$ et $Z=1$. Si $C=1$ et $Z=0$, cela signifie que $N1 < N2$. Par contre Si $C=0$ et $Z=0$, alors $N1 > N2$. Il est à remarquer que la situation $C=1$ et $Z=1$ à la suite d'une opération de soustraction est impossible.

Mais l'utilisation la plus fréquente de l'indicateur zéro est le test de fin de boucle : si une opération ou un ensemble d'instructions doit être exécuté n fois, ce nombre sera chargé dans un registre qui sera utilisé en compteur qu'on va décrémenter à chaque exécution de la boucle. Ainsi lorsque Z sera égal à 1, cela signifie que le contenu du compteur est nul, et donc que l'on a exécuté la boucle n fois. Une instruction spéciale (de branchement conditionnel) qui teste le bit Z permet de sortir de la boucle.

-3-Modes d'adressage

Les modes d'adressage caractérisent la façon dont est obtenue l'adresse du deuxième opérande d'une instruction, le premier opérande étant contenu dans l'accumulateur. Nous n'étudierons pas les modes d'adressage indirect qui sont nombreux et peu utilisés, pour nous concentrer sur les modes d'adressage les plus courants, et qui existent dans mimosa.

On appelle **adresse effective (EA)** l'adresse où se trouve réellement la donnée à lire, ou bien l'adresse d'écriture de la donnée.

-3-a-Adressage implicite ou adressage registre

format de l'instruction : « Code_Opération »

Le deuxième opérande est contenu dans un registre défini par le code opération de l'instruction elle même, il n'y a donc pas d'adresse à préciser.

Les instructions utilisant ce mode d'adressage sont celles portant sur les bits du registre processeur status, d'incrémention et de décrémention de registres internes, transfert de données entre registres, etc...

Pour Mimosa toutes les instructions de la deuxième partie du tableau, c'est-à-dire les instructions dont le code opération commence par F sont des instructions à adressage implicite (comme par exemple PSHA, HALTE, NOOP). Ces instructions occupent un octet.

-3-b-Adressage absolu ou direct :

format de l'instruction : « Code_Opération addrH addrL »

Le code opération de l'instruction est suivi d'une adresse exprimée en hexadécimal. C'est la façon la plus courante de faire un adressage pour le processeur, c'est-à-dire d'aller lire ou écrire une donnée en mémoire.

Remarque :

Dans ce mode d'adressage, pour Mimosa l'instruction occupe deux octets.

Exemple: Instruction « charger 4EB »: adresse effective: EA = 4EB.

-3-c-Adressage immédiat :

format de l'instruction : « Code_Opération # (ou \$) Donnée »

L'opérande est directement fourni, il n'y a donc pas d'adresse. C'est-à-dire qu'au lieu de fournir l'adresse où aller écrire ou lire une donnée, on donne directement la donnée en question. C'est la façon la plus simple de manipuler des constantes.

L'opérande est précédé du symbole # ou \$ qui est caractéristique de l'adressage immédiat.

Ce type d'adressage sert uniquement quand on veut comparer, charger dans un registre, ou tester des valeurs connues. C'est également la manière la plus rapide pour initialiser un registre à une valeur donnée.

Remarque:

Pour Mimosa nous avons supprimé le symbole (# ou \$) caractéristique de l'adressage immédiat, car nous avons utilisé une **instruction unique** (« NOMBRE ») pour effectuer de l'adressage immédiat. C'est pour cela qu'elle ressemble à un adressage absolu.

-3-d-Adressage indexé : *format de l'instruction : « Code_Opération Donnée, R »*

L'adresse effective est la somme de deux termes : le contenu d'un registre d'index R, et un nombre D précisé dans l'instruction et appelé déplacement (offset) : $EA = (R) + D$.

Le nombre D correspond à une adresse : si cette adresse se trouve en page 0 on parle alors d'adressage indexé page 0, sinon on parle d'adressage absolu indexé.

L'adressage indexé est très utile lorsqu'il s'agit d'adresser l'une après l'autre de nombreuses positions mémoire situées à des cases adjacentes, car le registre peut être incrémenté ou décrémenté comme un compteur.

L'adressage indexé est fondamental dans la gestion de tables, c'est-à-dire de blocs de données ou d'adresses.

Remarque :

Pour Mimosa, dans ce mode d'adressage, l'instruction occupe deux octets : 4 bits pour le code instruction (mimosa ne possède qu'un seul registre d'index appelé X) et 12 bits pour la donnée D (qui est une adresse).

Exemple : « CHARGER 120,X » Adresse effective: $EA = 120 + (X)$
(par exemple si $(X) = 5$, $EA = 125$).

-3-e-Adressage relatif : *format de l'instruction : « Code_Opération Donnée »*

Cet adressage est spécifique des instructions de branchements conditionnels.

La nouvelle adresse où le programme doit éventuellement se brancher (si le test est positif) n'est pas exprimée par l'adresse absolue, mais par un déplacement par rapport au contenu du compteur ordinal (adresse de l'instruction en cours).

L'adresse effective est donnée par : $EA = (PC) + i + D$, où :

(PC) : représente l'adresse du premier octet de l'instruction de branchement,

i : est le nombre d'emplacements mémoire occupés par l'instruction de branchement,

(PC) + i : est l'adresse de la prochaine instruction en séquence,

D : est le déplacement exprimé en hexadécimal et en complément à 2.

Exemple : instructions « SAUT,S Donnée » et « SAUT,E Donnée »

Le processeur teste le bit de signe (bit de poids le plus fort) du déplacement pour savoir s'il s'agit d'un déplacement positif ou négatif. Si ce bit =0, il s'agit d'un déplacement positif, c'est à dire que l'adresse de saut ou de branchement est supérieure à l'adresse de l'instruction en cours contenue dans le PC. Si ce bit =1, il s'agit d'un déplacement négatif, c'est à dire que l'adresse de saut ou de branchement est inférieure à l'adresse de l'instruction en cours contenue dans le PC.

L'avantage de l'adressage relatif est de ne pas comporter d'adresse absolue. Ainsi un programme où tous les branchements sont réalisés en adressage relatif, est facilement translatable d'une zone mémoire à une autre, sans devoir modifier les adresses.

Remarque :

Pour Mimosa, dans ce mode d'adressage l'instruction occupe deux octets : 4 bits pour le code instruction et 12 bits pour la donnée D.

Comme mimosa utilise 12 bits pour coder la donnée D, la valeur de D exprimée en complément à 2 ne dépasse pas $+2^{12-1} - 1$ et -2^{12-1} , soit $2^{12-1} - 1$ (= +2047 en décimal) et -2^{12-1} (= -2048 en décimal).

Exemple d'application :

Supposons que la mémoire de mimosa est organisée en octets (donc $i = 2$ et l'instruction occupe deux octets). Supposons également que l'instruction occupe les emplacements mémoire d'adresses 200 et 201. Considérons les deux cas d'un déplacement positif (pour lequel EA=210) puis négatif (pour lequel EA=1F6). Il s'agit de chercher dans les deux cas la valeur du déplacement pour l'instruction « SAUT,S Donnée ».

-a-Déplacement positif: EA = 210, calculons D.

$i=2 \rightarrow EA = (PC) + 2 + D \rightarrow 210 = 200 + 2 + D \rightarrow D = 210 - 202 = 00E$ (soit 14 en décimal). L'instruction s'écrira alors SAUT,S 00E .

-b-Déplacement négatif: EA = 1F6, calculons D.

$i=2 \rightarrow EA = (PC) + 2 + D \rightarrow 1F6 = 200 + 2 + D \rightarrow D = 1F6 - 202 = FF4$ (soit -12 en décimal). En effet en décimal cela donne $D = -12$ qui est négatif \rightarrow son complément à deux exprimé sur 12 bits donne FF4

L'instruction s'écrira alors SAUT,S FF4.

TD CHAPITRE 4 : LE PROCESSEUR OU CPU
PARTIE 1 : RAPPELS DE COURS

1°Quels sont les principaux éléments de l'unité centrale et du processeur central d'un ordinateur ?

2°Que signifient les sigles CPU, UAL, ALU, UC, PC, PSW, SP ?

3°Quelle est l'interprétation qu'on doit donner aux contenus des registres PC, RI, et ACC ? Conclure quant à leur taille.

4°Comment peut-on amener l'ordinateur à exécuter un programme dont la première instruction se trouve à l'emplacement mémoire d'adresse $(01F)_{16}$?

5°Supposons qu'une instruction se trouve à l'adresse n de la mémoire. Que se passe-t-il si après avoir exécuté cette instruction le compteur de programme prend toujours la valeur n ?

PARTIE 2 : EXERCICES
EXERCICE 6 : adresses symboliques et langage assembleur de mimosa

(mémoire organisée en mots de 16 bits)

Substituer à toutes les adresses symboliques (adresses d'instruction et d'opérande) de l'exemple, des adresses numériques hexadécimales. Montrer en même temps comment il faut placer le programme en mémoire, si on suppose que la première adresse à utiliser est (100)h, et que les variables a et b seront stockées respectivement aux adresses (110)h et (111)h.

LIGNE	LABEL	MNEMONIQUE	ADRESSE	SIGNIFICATION
1	aprêt :	SAUT,E	lirea	Si E vrai, aller à lirea
2		SAUT	aprêt	Aller à aprêt
3	Lirea :	ENTREE		Mettre ds Acc contenu du tampon d'entrée
4		STOCKER	a	Mettre (Acc) à l'adresse mémoire de a
5	bprêt :	SAUT,E	lireb	Si E vrai, aller à lireb
6		SAUT	bprêt	
7	Lireb :	ENTREE		
8		STOCKER	b	$m(b) := (Acc)$
9		ADD	a	$Acc := (Acc) + (m(a))$
10	Res_prêt :	SAUT,S	Res_prêt	Si S vrai, aller à Res_prêt
11		SORTIE		Mettre (Acc) dans Tampon de sortie
12		HALTE		

EXERCICE 7 : langage assembleur et assemblage de programme

(mémoire organisée en mots de 16 bits)

Réécrire le programme précédent en remplaçant chaque mnémonique du langage assembleur de mimosa par son code. Donner pour chaque adresse mémoire son contenu en hexadécimal puis en binaire.

EXERCICE 8 : désassemblage

L'ordinateur mimosa possède en mémoire le contenu indiqué par le tableau.

Adresse mémoire (hexa)	Contenu
100	0001 0000 0001 1001
101	0100 0001 0000 0101
102	0001 0000 0010 0000
103	0110 0001 0000 0101

104	1111 0101 indifférent
-----	-----------------------

1°Réécrire le même contenu en hexadécimal.

2°Réécrire le programme en code mnémonique puis donner sa signification.

EXERCICE 9 : programmation en assembleur

On suppose que l'ordinateur mimosa ne possède pas d'instruction de soustraction SUB. Ecrire un programme de calcul de $a - b$. On supposera que la valeur de a se trouve déjà dans l'accumulateur, que b occupe l'emplacement mémoire d'adresse $(10A)_{16}$, et que le résultat du programme (la différence) occupera l'accumulateur. Le programme devra être stocké en mémoire à partir de l'adresse $(100)_{16}$, et les emplacements mémoire d'adresses $(10B)_{16}$ et $(10C)_{16}$ peuvent être utilisés comme espace de travail.

EXERCICE 10 : fetch et exécution

En supposant que chaque instruction du programme de l'exercice 8 prend un cycle machine, et que chaque cycle est partagé en 2 phases (recherche et exécution), donner dans un tableau le contenu des registres PC, RI, ACC et de l'emplacement mémoire $(105)_{16}$, au cours du déroulement de l'exécution de ce programme.

EXERCICE 11 : assembleur + fetch + exécution

On veut prendre la valeur absolue d'un nombre qui se trouve à l'adresse mémoire $(109)_{16}$. Pour cela on prend le nombre et on teste s'il est négatif. Si oui, on le rend positif et on le range à la même adresse. Si non, on le range tel quel à la même adresse.

1° Ecrire le programme correspondant en langage assembleur de mimosa.

2° Donner le contenu des emplacements mémoire si le programme est stocké à partir de l'adresse $(100)_{16}$ (on supposera que le bus de données, l'accumulateur, et les emplacements mémoire ont tous une capacité de 16 bits).

3° En supposant que le nombre est positif, donner comment le programme est exécuté dans le temps, en précisant les contenus de : Bus de données, Bus d'adresses, PC, RI, ACC, $m(109)$.

4° Même question que 3° en supposant cette fois que le nombre est négatif.

EXERCICE 12 : pile et sous programme

On donne le programme suivant pour mimosa où chaque adresse est exprimée sur 12 bits, et où un emplacement mémoire vaut 16 bits, et chaque instruction occupe un emplacement mémoire.

Donner pour chacune des instructions 1 à 6, le contenu de la pile, de SP et de PC. Ces contenus correspondent à chaque fois à l'état juste après l'exécution de l'instruction.

L'état 0 correspond à l'état après la phase recherche de l'instruction SAUTSP 200, et juste avant la phase exécution de cette instruction. Dans cet état on a donc $(PC) = 120$ et $(SP) = 056$. La pile occupe les adresses de 000 à 0FF.

N° Instr	Adr	Instruction
	100	----
	---	----
	---	----
1	11F	SAUTSP 200
	120	----
	---	----
	---	----
	130	HALTE
Programme principal		

N° instr	adr	Instruction
	200	----
	---	----
	---	----
2	24F	SAUTSP 300
	250	----
	---	----
	---	----
6	270	RETSP
Sous programme 1		

N° instr	adr	Instruction
	300	----
	---	----
3	310	PSHA
	---	----
4	320	PULA
	---	----
	---	----
5	350	RETSP
Sous programme 2		

EXERCICE 13 : utilisation des indicateurs Carry, Signe, Zero

Il s'agit de tester un bit du registre d'état d'un organe d'E/S. Si le bit est à 1, le périphérique est prêt et on peut faire le transfert d'E/S. S'il est égal à zéro, le périphérique est occupé et il faut attendre jusqu'à ce qu'il soit prêt. Donner pour une lecture sur le périphérique, le programme en langage assembleur dans les trois cas suivants :

1°cas : l'état *prêt* est indiqué par le bit 2^0 du registre d'état

Après lecture du registre d'état, il est pratique d'envoyer le bit d'état dans le carry par une instruction de décalage, et de tester le *bit carry* (la lecture du registre d'état correspond à un transfert de son contenu dans l'accumulateur Acc).

2°cas : l'état *prêt* est indiqué par le bit 2^7 du registre d'état

Après lecture du registre d'état (transfert dans Acc), il suffit de tester l'*indicateur de signe* (en supposant que la lecture du registre positionne le bit N), qui sera à 1 si le bit numéro 7 du nombre lu est à 1.

3°cas : l'état *prêt* est indiqué par le bit 2^3 du registre d'état

Le moyen le plus simple de tester l'état *prêt* est de faire un masquage, c'est à dire de mettre à zéro tous les bits sauf le *bit prêt*, de sorte que l'*indicateur Z* sera positionné ($Z=1$) si *prêt* = 0, et Z sera nul dans le cas contraire. Le masquage se fera donc par un ET logique avec la valeur binaire 00001000 (08 en hexa). On utilisera l'adresse 200 comme adresse de travail et de stockage temporaire.

Remarque : l'adresse du registre d'état PSW est 5FF.

EXERCICE 14 : algorithmie et programmation en assembleur de mimosa

Dans une table constituée de nombres logiques, on cherche à déterminer l'élément qui a la plus petite valeur. Ecrire l'algorithme puis le programme en langage assembleur de mimosa.

On suppose que la table commence à l'adresse 300 et que sa longueur est égale à 256. La valeur min devra être stockée en fin de programme à l'adresse 400. Le programme est stocké en mémoire à l'adresse 100. On utilisera comme zones de travail 3 registres correspondant chacun à une adresse :

- R0 : contient la valeur min provisoire
- R1 : contient l'adresse du i^{ème} élément
- R2 : sert de compteur

PARTIE 3 : CONTROLE DES CONNAISSANCES - DUREE 1H30 -

EXERCICE 3-1 : Adressage relatif sur Mimosa

Si la mémoire de mimosa est **organisée en octets**, considérons l'instruction de branchement conditionnel en adressage relatif « SAUT,S donnée », où donnée exprime le déplacement D. Cette instruction occupe les emplacements mémoire d'adresses 310 et 311.

1° Si D = 00A, quelle est la valeur de l'adresse effective EA ?

2° Donner la valeur de D si : a°) on veut se brancher à l'adresse 313 ; b°) on veut se brancher à l'adresse 30F.

EXERCICE 3-2 : Pile et sous programme (mémoire organisée en octets)

Considérons le programme suivant pour Mimosa. Donner le contenu de la PILE, de SP, de PC et RI, à la fin de la phase recherche et à la fin de la phase exécution de chaque instruction. L'état initial correspond à (PC) = 200 ; (SP) = 050 ; (Acc) = 2FB3.

PROGRAMME PRINCIPAL

Numéro Instructio	Adresse Mémoire	Mnémonique	Adresse
	200	-----	---
	---	-----	---
1	220	PUSHA	
	---	-----	---
2	22F	SAUTSP	400
	---	-----	---
8	235	PULA	
	---	-----	---
9	250	HALTE	

SOUS PROGRAMME

Numéro Instructio	Adresse Mémoire	Mnémonique	Adresse
	400	-----	---
	---	-----	---
3	420	NOMBRE	435
4	422	PSHA	
	---	---	---
	---	-----	---
5	44F	NOP	---
6	450	PULA	
	---	-----	---
	---	-----	---
7	500	RETSP	

EXERCICE 3-3 : Désassemblage et exécution de programme.

Le contenu de la mémoire de MIMOSA (**organisée en octets**) est donné par le tableau.

Adresse mémoire	Contenu	Adresse mémoire	contenu
300	21	307	72
301	F2	308	F5
302	61	309	A3
303	F3	30A	00
304	41	30B	42
305	F2	30C	F4
306	FD	30D	F5

1° Désassembler le programme qui se trouve en mémoire.

2° Donner après l'exécution du programme, le contenu des emplacements mémoire d'adresses : 1F2, 1F3, 2F4, 2F5. Avant l'exécution, ces emplacements contenaient respectivement les valeurs hexadécimales suivantes : 07, 05, 0E, 04 .

3° Quelle est l'instruction qu'on a oubliée dans ce programme et pourquoi ?

EXERCICE 3-4 : Programmation en assembleur et assemblage

On désire écrire un programme qui permet de lire deux nombres a et b sur un périphérique d'entrée (clavier par exemple), de les stocker aux adresses respectives 300 et 301, puis de faire leur comparaison, et en fonction du résultat effectuer certains traitements :

- si a = b, stocker le nombre « a » (ou « b ») à l'adresse 400, puis envoyer ce nombre sur le périphérique de sortie (imprimante par exemple) et s'arrêter;

- si b < a, calculer la moyenne $(a+b)/2$, puis stocker le résultat à l'adresse 400, puis envoyer le résultat sur le périphérique de sortie et s'arrêter;

- si b > a, retourner obligatoirement saisir deux nouveaux nombres a et b.

1° Ecrire le programme correspondant en langage assembleur de Mimosa. Ce programme sera stocké en mémoire à partir de l'adresse 200, et la mémoire est organisée en mots de 16 bits.

2° Donner le contenu des emplacements mémoire en hexadécimal

- Table des matières - PARTIE 2 : EXERCICES CORRIGES

CHAPITRE 1 : LOGICIELS ET ENVIRONNEMENT DE PROGRAMMATION	PAGE
-I-Terminologie de base	83
Informatique, hardware et software, système d'exploitation, processeur, carte mère, bus, entrées sorties, interfaces.	
-II-Langages et environnement de programmation	85
Les langages, les traducteurs, environnement de programmation.	
-III-Les logiciels	61
Editeurs, traitements de texte, tableurs, gestionnaires de bases de données, ...	
CHAPITRE 2 : REPRESENTATION DES INFORMATIONS EN MEMOIRE ET CODAGE	
Ex 1 & 2 : Rappel sur les systèmes de numération et conversions	94
Rappels de cours (notions de base, bit, quartet, octet, mot) ; codage, décodage, transcodage de nombres entiers et fractionnaires	
Ex 3 & 4 : Représentation des nombres en mémoire	99
Codage des entiers naturels, des entiers relatifs, complément à 2, nombres fractionnaires en virgule fixe, nombres fractionnaires en virgule flottante	
Ex 5, 6, 7 : Opérations élémentaires en arithmétique non signée	101
Additions binaire, hexadécimale, BCD ; soustraction en complément à 2 ; multiplication et division binaires.	
Ex 8 & 9 : Codage, transmission et impression	104
Codage de messages en code baudot, iso, ascii, ebcdic ; transmission : bits de parité, start, stop ; impression : caractères imprimables et caractères de contrôle.	
CHAPITRE 3 : LES MEMOIRES	
Ex 1, 2, 3 : Bus d'adresses, bus de données, taille de l'espace adressable	107
Ex 4 : Stockage de nombres et de caractères	108
Ex 5 & 6 : Mémoire organisée en octets et mémoire organisée en mots, techniques de stockage « big endian » et « little endian »	108
Ex 7 : Mémoire segmentée	110
Ex 8 & 9 : Réalisation physique de mémoires	110
Ex 10 : Synthèse des notions précédentes	115
Ex 11&12&13&14 : Antémémoire, mémoire paginée, et mémoire virtuelle	118
Ex 15 & 16 : Unité de disque dur : pistes, secteurs, cylindre, capacité, taux de transfert	121
CHAPITRE 4 : LE PROCESSEUR OU CPU	
Partie 1 : Rappels de cours	125
Ex1 à Ex5 : Notions de CPU, UC, UAL ; Registres PC, ACC, RI, PSW, SP	
Partie 2 : Exercices	
Ex6 : Adresses symboliques et adresses physiques, langage assembleur de mimosa	127
Ex 7 & 8 : Assemblage et désassemblage	
Ex9 : Programmation en langage assembleur	129
Ex10 : Fetch et exécution	
Ex11 : Exercice de synthèse : assembleur, fetch et exécution	
Ex12 : Pile et sous programme	132
Ex13 : Utilisation des bits ou indicateurs zéro (Z), carry (C), signe (N)	133
Ex14 : Algorithmie et programmation en assembleur	135
Partie 3 : Contrôle des connaissances	
Ex3-1 : Adressage relatif sur Mimosa	137
Ex3-2 : Pile et sous programme (mémoire organisée en octets)	
Ex3-3 : Désassemblage et exécution de programme.	138
Ex3-4 : Programmation en assembleur et assemblage	139

**Td chapitre 1 : NOTIONS SUR LES LOGICIELS
ET
L'ENVIRONNEMENT DE PROGRAMMATION**

-I-TERMINOLOGIE DE BASE

-1-Informatique

L'informatique est le traitement (par une machine) automatisé de l'information.

Les différentes formes de l'information sont : écrite (texte), sonore (parole et musique) et visuelle (image fixe ou animée). Un ordinateur multimédia permet de traiter ces trois formes.

-2-Hardware (matériel) et software (logiciel)

Le mot *hardware* (littéralement ce qui est dur) désigne tout ce qui est matériel (cartes, clavier, processeur...). Quant au *software* (ce qui est mou) il correspond au produit de l'esprit humain : les programmes. Il désigne aussi bien le système d'exploitation que les programmes achetés ou développés par l'utilisateur.

-3-Système d'exploitation

C'est un ensemble de programmes de gestion (exploitation) de l'ordinateur. Il est organisé en couches ou en modules ayant chacun une fonction particulière : gestion mémoire, gestion des E/S, gestion horloges, interface utilisateur, etc... On peut citer parmi les plus courants VMS, UNIX, DOS, WINDOWS 9X, SYSTEME 8, LINUX.

Chaque système d'exploitation possède son propre langage de commande.

-4-PC ou Compatible PC

PC est l'abréviation de « Personal Computer », terme désignant une famille d'ordinateurs d'IBM, qui porte en elle un nouveau concept et a révolutionné l'utilisation de l'outil informatique (bien qu'IBM ne soit pas l'inventeur de ce concept). Un ordinateur « compatible » est un ordinateur qui a la même architecture de base qu'un IBM-PC et utilise le même système d'exploitation (MS-dos). Il est désigné par « compatible IBM-PC », ou l'association des mots composant cette phrase : compatible, PC, compatible IBM, etc...

-5-Processeur ou CPU

C'est le cerveau de l'ordinateur. Il est constitué principalement d'une unité arithmétique et logique (pour effectuer les traitements), d'une unité de commande (qui coordonne le travail des différents éléments du processeur et de l'ordinateur), d'un ensemble de registres de travail, et de mémoire interne auxiliaire. Parmi les fabricants de processeurs les plus célèbres on peut citer Intel (processeurs 80x et pentium), Motorola (processeurs 680xx), AMD et Cyrix.

Le terme *microprocesseur* fait référence à une miniaturisation très poussée, il désigne un processeur avec un fort taux d'intégration de composants (plusieurs millions de transistors).

-6-Carte mère

C'est la partie vitale de l'ordinateur. C'est sur cette carte que viennent se fixer les composants électroniques indispensables (processeurs, cartes mémoire, circuits d'horloges, etc..) et les *cartes filles* d'extension correspondant chacune à une application déterminée.

-7-Bus

Ce sont les moyens de transport de l'information à l'intérieur de l'ordinateur. On peut faire l'analogie entre le bus de transport en commun et le bus d'ordinateur de la manière suivante : - *unité élémentaire d'information* : la personne \leftrightarrow le bit,

- *unité de transport*: la place assise \leftrightarrow le fil,

- *capacité de transport ou taille du bus*: nombre de places assises \leftrightarrow en nombre de bits ou fils.

Bus système : il regroupe les bus de données, d'adresses, de commande, ainsi que les lignes (fils) d'alimentations et d'horloges. Il est disponible sur la carte mère sous forme de *slots* (rails ou connecteurs) pour « enficher » les cartes d'extension ou *cartes filles*.

Dans le monde du PC, les plus courants sont le bus *ISA* (Industry Standard Architecture), et à un degré moindre le bus *EISA* (Extended ISA) sur certains serveurs, *MCA* (Micro Channel Architecture) pour les ordinateurs PS/2 d'IBM, et *PCMCIA* (sur les portables).

Bus local : Bus relié « directement » au processeur pour accélérer les transferts d'information. Les plus courants dans le monde du PC sont : le bus *PCI* (Peripheral Component Interconnect) qui sert à connecter des cartes filles et joue donc le même rôle que le bus *ISA*, le bus *VESA* (Video Electronics Standards Association) pour connecter des cartes graphiques, et le bus *AGP* (Accelerated Graphics Port) pour accélérer le transport des informations graphiques.

-8-Entrées/Sorties (E/S)

Elles désignent respectivement un transfert d'information de l'extérieur vers le processeur ou du processeur vers l'extérieur.

-9-Interface

Terme (masculin ou féminin) désignant un dispositif (matériel ou logiciel) qui permet à deux entités (matérielles ou logicielles) de communiquer entre elles. Il permet de faire de l'adaptation d'information : de forme (parallèle vers série ou l'inverse), de code (transcodage) ou de vitesse (cas d'un du transfert vers une imprimante par exemple).

Les interfaces d'E/S les plus courantes sont les interfaces *série* (ports *COM*) et *parallèle* (ports *LPT* : *LPT1=PRN* pour le port d'imprimante sous DOS). Elles ont tendance à être remplacées par le port *USB* (Universal Synchronous Bus) qui est beaucoup plus rapide mais dont l'utilisation n'est pas encore généralisée.

-10-Drivers ou pilotes

Ce sont des programmes de gestion (*pilotage*) des cartes d'interfaces du matériel (on parle de drivers de carte graphique, drivers du cdrom, etc...).

-II- LANGAGES ET ENVIRONNEMENT DE PROGRAMMATION

-II-1-Les langages (définitions cf encyclopédie encarta 1999)

Le **langage de programmation** est un langage informatique composé d'une série d'instructions pouvant être traitées, interprétées et exécutées par un ordinateur. Ces instructions, qui sont frappées au clavier, se composent de caractères, de symboles, et de règles permettant de les assembler.

Il existe différents types de langages, allant du plus rudimentaire au plus complexe, que l'on classe généralement en deux familles: les langages de bas niveau et les langages évolués. On y ajoute parfois une autre catégorie, les langages de quatrième génération.

-1-a-Langages de bas niveau

Les langages de bas niveau sont des langages proches de la machine, ou des langages offrant peu d'instructions et de types de données

Langage machine

Le langage machine représente le langage dans lequel s'exprime le résultat final d'une compilation de langage assembleur ou d'un langage de haut niveau quelconque. Constitué de « 0 » et de « 1 », ce langage est chargé et exécuté par le **microprocesseur**. Appelé également code machine, il constitue le seul langage réellement «compris» par l'ordinateur, tous les autres langages correspondant à des formes de structuration du langage humain.

Langage assembleur

Le langage assembleur est un langage de programmation de très bas niveau, où chaque instruction correspond à une instruction machine unique. Le jeu d'instructions d'un tel langage est donc associé à un certain type de processeur. Ainsi, les programmes écrits en langage assembleur pour un processeur particulier doivent être réécrits pour tourner sur un ordinateur équipé d'un processeur différent. Après écriture d'un programme en langage assembleur, le programmeur fait alors appel à « l'**assembleur** » spécifique du processeur, qui traduit ce programme en instructions machine. Le langage assembleur peut être préféré à un langage de haut niveau lorsque le programmeur recherche une vitesse d'exécution élevée ou un contrôle étroit de la machine. En effet, les programmes écrits dans ce type de langage tournent plus vite et occupent moins de place que ceux produits par un compilateur. En outre, ils donnent au programmeur la possibilité d'agir directement sur le matériel (processeur, mémoire, affichage et connexion d'entrées / sorties).

-1-b-Langages évolués

Les langages évolués, dits aussi de haut niveau, sont des langages informatiques offrant un certain niveau d'abstraction par rapport au langage de la machine, et manipulant des structures syntaxiques telles que les déclarations, les instructions de contrôle, etc... Usuellement, le terme désigne tout langage de niveau supérieur à celui du langage assembleur.

Les langages évolués sont classés en trois grandes familles : les langages procéduraux, les langages orientés-objets et les langages orientés-listes. On retrouve ainsi dans la famille des langages **procéduraux** le FORTRAN, le COBOL, le BASIC, l'ADA, le PASCAL et le C, dans la famille des langages **orientés-objets**, le C++, et dans la famille des langages **orientés-listes**, le LISP et PROLOGUE.

Les langages procéduraux sont des langages où la procédure (suite d'instructions) constitue l'élément de base. La plupart des langages évolués sont des langages procéduraux.

Les langages orientés-objets sont des langages adaptés à la programmation orientée-objet, type de programmation où chaque programme est considéré comme un ensemble d'objets distincts, ces objets constituant eux-mêmes des ensembles de structures de données et de procédures intégrées. Dans de tels langages, chaque objet appartient à une classe qui définit les structures de données et les procédures associées à cet objet.

Les langages orientés-listes peuvent être apparentés aux langages orientés-objets, à la différence près qu'ils manipulent non pas des objets mais des listes, c'est-à-dire des structures de données multi-éléments à organisation linéaire.

-1-c-Langages de quatrième génération

Les langages de quatrième génération (L4G en abrégé), conçus pour l'interaction avec le programmeur, qualifient souvent les langages propres aux **bases de données** relationnelles. Se situant un cran au-dessus de langages tels que le Pascal ou le COBOL, ils se composent d'un jeu d'instructions s'apparentant à des *macro-instructions*, séquences d'instructions prédéfinies auxquelles on accède par une commande très simple. Toutefois, ces langages conservent un aspect hybride, dérivant le plus souvent des langages évolués. Le plus connu d'entre eux est SQL.

Acronyme de *Structured Query Language*, le langage SQL n'est pas un véritable langage comme le C ou le Pascal, pouvant être plutôt considéré comme un langage d'interrogation, de mise à jour et de gestion des bases de données relationnelles. Dérivé du projet de recherche d'IBM qui avait conduit à l'élaboration du langage SEQUEL (*Structured English Sequel Language*), le langage SQL est une norme mondialement reconnue et répandue. Il peut être employé pour formuler des questions de manière interactive, mais aussi inséré dans un programme sous forme d'instructions de manipulation de données. Il est d'une utilisation très simple, le destinant ainsi aux professionnels comme aux novices en informatique.

-II-2-Les traducteurs

Un ordinateur ne comprend qu'un seul langage, le langage machine qui se présente comme une suite de « 0 » et de « 1 » (forme binaire). C'est pourquoi tout programme écrit dans un langage (de bas ou de haut niveau) doit être décodé et traduit en langage machine, avant de pouvoir être exécuté. Ce processus de conversion du **code source** (frappé au clavier) **au code objet** (assimilable par l'ordinateur) est assuré par un programme capable de traduire un jeu de symboles en un autre jeu, par application de règles de syntaxe et de sémantique. Suivant la nature du langage de programmation employé, ce programme s'appelle un assembleur, un compilateur ou un interpréteur.

-2-a-Assembleur et désassembleur

L'assembleur est un programme de traduction : il traduit un programme (source) écrit en langage assembleur en un programme (cible) en langage machine.

Le désassembleur effectue le travail inverse : il traduit un programme en langage machine en un programme en langage assembleur.

L'assembleur et le désassembleur sont spécifiques d'un langage assembleur, donc correspondent à un processeur particulier.

-2-b-Langages compilés

Les langages dits compilés sont des langages où toutes les instructions sont d'abord traduites en code objet dans une première étape, puis dans une seconde étape en code exécutable. Cette conversion s'effectue au moyen d'un **compilateur** avant de faire appel à un **éditeur de liens** (« *linker* » en anglais).

Compilation :

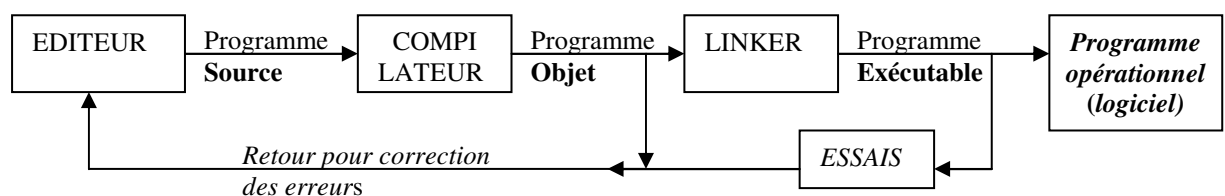
Le code du programme à compiler, ou code « source », est généralement écrit dans un langage de haut niveau (par exemple le C ou le Pascal). Or, l'ordinateur ne peut pas exécuter directement ce code. Le but de la compilation, justement, est de le traduire en séquence d'instructions lisibles par le *microprocesseur*. La compilation produit alors un code « objet » qui est exprimé en langage machine de l'ordinateur, dans lequel il serait peu pratique d'écrire directement. Cette tâche de traduction est réalisée par un **programme appelé compilateur**.

Au sens le plus large, un compilateur traduit un ensemble de symboles en un autre ensemble selon différentes règles logiques. Les compilateurs procèdent généralement en deux phases. Durant la première, il s'agit de comprendre le code source, c'est-à-dire d'en créer une représentation cohérente en mémoire. Pour cela, on effectue successivement une analyse lexicale, qui reconnaît les symboles du langage source, puis une analyse syntaxique, qui identifie la structure du programme selon la grammaire du langage, et enfin une analyse sémantique, qui vérifie les autres règles du langage et détecte les éventuelles erreurs. Lors de la seconde phase, on procède à la traduction proprement dite : la structure du programme évolue selon les exigences du langage d'arrivée, pour produire finalement un code objet qui soit performant.

Edition des liens :

Comme le propre d'un langage évolué est d'être indépendant de la machine, par conséquent le code objet produit par la compilation n'est en général pas directement exécutable. Il faut donc effectuer une phase d'éditions des liens, qui *transforme les adresses logiques en adresses physiques*. Par exemple quand une application est écrite sous forme de programme principal et de sous programmes, il faut effectuer une correspondance entre les noms logiques des sous programmes, utilisés par le programme principal pour effectuer les appels, et les adresses physiques de ces sous programmes. Le **linker** (ou éditeur de liens) est le programme qui effectue cette édition des liens entre les différents programmes pour produire un seul programme exécutable.

Remarque : Dans certains environnements de programmation, le linker est intégré (comme dernière phase) au compilateur, et on parle alors d'un compilateur trois passes (effectue trois passages : analyse syntaxique et sémantique, traduction, édition des liens) qui produit directement du code exécutable.



Phases de développement d'un logiciel

Debugger (ou déboggeur)

C'est un outil (programme) *d'aide à la mise au point* de programmes. Il est utilisé pendant la phase de développement pour détecter et corriger les erreurs de programmation (qui ne sont ni syntaxiques ni sémantiques et ne sont donc pas détectables par le compilateur). La compilation et l'exécution se font sous le contrôle du **debugger**, et on peut insérer différents points d'arrêt dans le programme. A l'exécution le programme s'arrête aux points prédéfinis, et on peut alors vérifier les valeurs des variables, des registres du processeur, forcer des valeurs etc..., avant de poursuivre l'exécution du programme.

-2-c-Langages interprétés

Les langages dits interprétés sont des langages décodés et exécutés instruction par instruction à l'aide d'un programme appelé **interpréteur**. Le plus connu des langages interprétés est le langage BASIC, bien que la plupart des versions actuelles en permettent ou en imposent la compilation.

Un programme écrit dans un langage interprété est directement exécutable, et ne nécessite pas une phase préalable de traduction. Il est décodé et traduit ligne par ligne au moment de son exécution.

-II-3-Environnement de programmation

-3-a-Environnement

Il désigne l'ensemble des éléments physiques et matériels constituant l'environnement dans lequel on développe une application ou un programme. Ce sont en premier lieu la machine (processeur) et son système d'exploitation, puis les outils logiciels tels que le compilateur (et donc le langage d'écriture des programmes), le linker, et éventuellement le debugger.

On distingue les environnements classiques ou séparés, dans lesquels chaque élément existe sous forme de programme à part, et les **environnements intégrés**, où le développement de programmes est beaucoup plus souple. En effet l'utilisateur exécute un seul programme pour accéder à un environnement avec des fenêtres. Le passage d'une fonction à une autre (édition, compilation, exécution, etc...) se fait par simple changement de fenêtre.

-3-b-Compatibilité

La notion d'ordinateur compatible avec l'IBM-PC suppose une compatibilité à deux niveaux : hardware et software.

En premier lieu il faut avoir la même architecture physique que l'IBM-PC. En second lieu il faut utiliser le même système d'exploitation, et les programmes qui s'exécutent sur l'un doivent pouvoir s'exécuter sur l'autre.

Si les deux niveaux sont satisfaits, on parle alors d'ordinateur 100% compatible.

-3-c-Portabilité

La notion de compatibilité engendre celle de portabilité : tout programme développé sur un ordinateur donné peut être « porté » sur un autre ordinateur, où il s'exécutera directement sans aucune modification. On dira de ce programme ou de ce logiciel qu'il est portable.

-III-LES LOGICIELS

Un logiciel est un programme ou un ensemble de programmes destinés à une application particulière. On regroupe généralement les applications par groupes ou familles d'applications.

III-1-Editeurs

Editer un fichier consiste à l'ouvrir soit en lecture, soit en lecture et écriture (pour modifier son contenu). S'il n'existe pas il est souvent automatiquement créé à l'ouverture.

Une fois qu'on a terminé de travailler sur un fichier, on le ferme puis on le sauvegarde en mémoire de masse. Le format de sauvegarde est le format *texte* ou *ascii* (sous forme de chaînes de caractères alphanumériques codés en *ascii*), et l'extension du fichier est généralement « .txt ». Avec un éditeur de texte on dispose généralement d'une seule police de caractères, la taille des fichiers est limitée, le nombre d'opérations (de manipulation du texte) est également limité.

On distingue les éditeurs ligne (qui affichent une seule ligne à la fois) et les éditeurs pleine page qui permettent une saisie et une modification aisées du texte.

-III-2-Traitements de texte

Le principe de fonctionnement est le même que celui de l'éditeur, avec cependant quatre différences importantes :

- une plus grande variété de polices de caractères, avec différentes formes et tailles,
- on peut effectuer toutes les opérations imaginables sur le texte,
- la taille du fichier est théoriquement illimitée (limitée en fait par la taille de la mémoire RAM),
- en plus du format texte, chaque logiciel de traitement de texte possède son propre format de sauvegarde des données. Certains (comme winword) permettent même de lire et de sauvegarder dans différents formats de données (ceux des autres traitements de texte). Ils permettent également d'intégrer au texte des dessins, des graphes et des tableaux.

-III-3-Tableurs

Un tableur est un logiciel qui permet de réaliser des tableaux, appelés feuilles de calcul. Une feuille est constitué d'un ensemble de cellules, où chacune est l'intersection d'une ligne et d'une colonne. Les lignes sont en général indexées (numérotation) par des chiffres, et les colonnes par les lettres de l'alphabet. Par exemple D2 signifie la cellule du tableau constituée par l'intersection de la ligne 2 et de la colonne D (quatrième colonne).

Le contenu d'une cellule peut être de type numérique, caractère, formule mathématique, date, etc...

Au niveau fonctionnel, on dispose des mêmes possibilités et fonctionnalités que pour un traitement de texte.

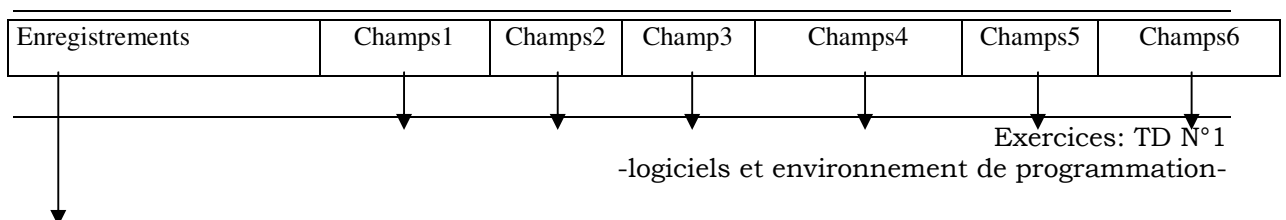
Comme exemples de tableurs pour PC on peut citer excel et multiplan.

-III-4-Gestionnaires de bases de données

Une base de données (BDD) est un ensemble de données ordonnées, donc structurées d'une certaine manière. L'intérêt de structurer les informations est de faciliter la recherche et la modification .

Pour créer une BDD, on commence par définir la structure des données, c'est à dire les différents champs et leurs types. Ensuite on procède à l'instanciation de la base c'est à dire à son remplissage par des enregistrements.

Les gestionnaires les plus courants dans le monde du PC sont dbase, paradox et access.



Champs →	N°matricule	Nom	Prénom	Date_naissance	Adresse	Note_contrôle
Types →	Nombre-entier	Caractère-16	Caractère-16	Date	Caractère-30	Nombre-décimal
Enregistrement1	123456	Benali	Ali	1/1/2000	1 rue du bug 2000 Tera nova	10.25
Enregistrement2	123502	---	---	---	---	---
Enregistrement3	123479	---	---	---	---	---
---	---	---	---	---	---	---

Exemple de base de données

-III-5-Logiciels intégrés

C'est un logiciel qui intègre principalement les fonctions de traitement de texte, tableur, et gestionnaire de base de données.

Pour le PC les plus courants sont 1-2-3 de Lotus et Works de Microsoft.

-III-6-Logiciels de dessin

On peut les séparer en deux catégories : ceux qui sont destinés à faire des graphes à partir d'un ensemble de points (destinés surtout aux scientifiques et de plus en plus intégrés dans d'autres logiciels), et ceux qui permettent de faire du dessin proprement dit (MSDRAW et Paint).

-III-7-Logiciels utilitaires

Comme leur nom l'indique ce sont des outils qui facilitent le travail de l'utilisateur. On peut les classer en deux catégories principales : ceux qui font du traitement de fichiers (copie, sauvegarde, vérification du contenu, compactage, etc...) comme « ptools » ou « norton utilities » sur le PC, et les antivirus qui permettent de nettoyer et d'immuniser des disques contre les virus.

-III-8-Logiciels dédiés

C'est un logiciel destiné à une application particulière et qui ne fait pas partie de l'une des familles déjà citées.

SOLUTIONS DU TD CHAPITRE 2
« REPRESENTATION DES INFORMATIONS EN MEMOIRE ET CODAGE »
-I-RAPPELS SUR LES SYSTEMES DE NUMERATION : CONVERSIONS
1°Rappels de cours
-1°-1- Notions de base et système de numération

Quelle que soit la base b , tout chiffre exprimé dans cette base varie de 0 à $b-1$. Ainsi par exemple dans la base 10 les chiffres varient de 0 à 9, dans la base 8 ils varient de 0 à 7, et dans la base deux ils prennent uniquement deux valeurs : 0 et 1 (d'où l'appellation de système binaire).

Les bases les plus utilisées sont 10, 8, 16 et 2. Ces bases ont donné naissance aux différents systèmes de numération respectifs: décimal, octal, hexadécimal et binaire. Ainsi si on effectue des opérations arithmétiques ou logiques dans la base 10 ou la base 2, on dit que l'on travaille respectivement en décimal ou en binaire.

Principe :

Tout nombre N écrit dans une base b peut se mettre sous la forme:

$$N(b) \equiv (N)_b = \underbrace{a_n b^n + a_{n-1} b^{n-1} + a_{n-2} b^{n-2} + \dots + a_0 b^0}_{\text{partie entière}} + \underbrace{a_{-1} b^{-1} + a_{-2} b^{-2} + \dots + a_{-n} b^{-n}}_{\text{partie fractionnaire}}$$

L'écriture du nombre N sous la forme d'une suite de coefficients a_i revient à écrire le nombre N dans la base b . Pour trouver son équivalent décimal, il suffit de développer l'expression précédente.

-1°-2-Que désignent les termes bit, quartet, octet, mot ?

Dans le système binaire la base $b=2$, il n'y a donc que deux valeurs possibles pour les chiffres : 0 et 1. Chacune de ces valeurs possibles est appelée **bit**. Les ordinateurs travaillant en binaire, le bit représente donc l'information élémentaire que traite un ordinateur. Cette information est vraie (égale à 1) ou fausse (égale à zéro). Il est alors facile de faire des opérations logiques sur les bits : ET, OU, Complémentation, Ou exclusif etc...

Un **quartet** est un regroupement de 4 bits. Il est surtout utilisé dans le transcodage de la base 16 vers le binaire (base 2) ou inversement.

Un **octet** est un regroupement de 8 bits. Il représente une unité très pratique pour la représentation, le codage et le stockage des données.

Le **mot** représente l'entité élémentaire utilisée par l'ordinateur pour coder et manipuler les données. Cette unité dépend de la taille (en nombre de bits) des données que traite le processeur de l'ordinateur (plus précisément l'unité arithmétique et logique). Elle correspond en général également à l'unité (en nombre de bits) utilisée pour le stockage des informations en mémoire. On parlera d'ordinateur 8, 16 ou 32 bits, pour signifier que le processeur utilise 8, 16 ou 32 bits pour coder et traiter les données.

2°Conversions

Enoncés

Effectuer les conversions des nombres suivants:

2°-1-Nombres entiers

Solutions

$$(102)_3 \rightarrow (?)_{10}$$

$$(11)_{10}$$

$$(321)_{10} \rightarrow (?)_4$$

$$(11001)_4$$

$$(543)_{10} \rightarrow (?)_8$$

$$(1037)_8$$

$$(22)_{16} \rightarrow (?)_8$$

$$(42)_8$$

$$(16)_7 \rightarrow (?)_3$$

$$(111)_3$$

$$(7F)_{16} \rightarrow (?)_8 \text{ (de deux méthodes différentes) solution } (177)_8$$

$$(177)_8 \rightarrow (?)_{16}$$

$$(7F)_{16}$$

$$(7F)_{16} \rightarrow (?)_{BCD}$$

$$(0001.0010.0111)_{BCD} = (127)_{10}$$

-2°-2-Nombres fractionnaires

Solutions

$$\text{-a-Décimal} \rightarrow \text{Binaire} : (9,375)_{10} \rightarrow (?)_2$$

$$(1001,011)_2$$

$$(4,345)_{10} \rightarrow (?)_2$$

$$(100,01011)_2$$

$$\text{-b-Binaire} \rightarrow \text{Décimal} : (101,101) \rightarrow (?)_{10}$$

$$(5,625)_{10}$$

Solutions

2°-1-Nombres entiers

2-1-a-Codage : il s'agit de transcrire un nombre N exprimé en base dix dans une base b quelconque. La méthode consiste à procéder par divisions successives par la base, on s'arrête quand le dernier quotient est nul. Le résultat sera donné par les restes successifs obtenus dans l'ordre, en les écrivant de droite à gauche en fonction des puissances croissantes de la base, selon la formule suivante (où n est le nombre de divisions et les R_i les restes successifs) :

$$(N)_b = R_n b^{n-1} + R_{n-1} b^{n-2} + R_{n-2} b^{n-3} + \dots + R_1 b^0$$

Conversion de $(32)_{10}$ en base 4. On effectue 5 divisions successives ($n = 5$) et on aura donc 5 restes :

$$\begin{array}{rcl}
 & & R_6 \ R_5 \ R_4 \ R_3 \ R_2 \ R_1 \\
 (32)_{10} & \rightarrow & (1 \ 1 \ 0 \ 0 \ 1)_4 \\
 (543)_{10} & \rightarrow & (1 \ 0 \ 3 \ 7)_8 \\
 (44)_{10} & \rightarrow & (1 \ 0 \ 1 \ 1 \ 0 \ 0)_2 \\
 (127)_{10} & \rightarrow & (7 \ F)_{16}
 \end{array}$$

-2-1-b- Décodage : cela consiste à écrire un nombre N exprimé dans une base b dans la base dix. Le résultat est obtenu en appliquant directement la formule donnée dans le principe.

$$(300)_8 = (3 \times 8^2 + 0 \times 8^1 + 0 \times 8^0)_{10} = 3 \times 64 = (192)_{10}$$

$$(102)_3 \rightarrow (11)_{10}$$

$$(16)_7 \rightarrow (13)_{10}$$

-2-1-c- Transcodage : il consiste à transcrire un nombre N écrit dans une base b1 vers une autre base b2, b1 et b2 étant quelconques.

*Cas général: cette méthode est valable quelles que soient les bases b1 et b2.

Un transcodage se fera d'abord par décodage puis par codage.

$$\begin{array}{ccccc}
 & \text{Décodage} & & \text{Codage} & \\
 \text{base b1} & \longrightarrow & \text{base 10} & \longrightarrow & \text{base b2} \\
 (111)_3 & \rightarrow & (13)_{10} & \rightarrow & (16)_7 \\
 (7F)_{16} & \rightarrow & (127)_{10} & \rightarrow & (177)_8 \\
 (177)_8 & \rightarrow & (127)_{10} & \rightarrow & (0111 \ 1111)_2
 \end{array}$$

*Cas particuliers:

** b2 = 2 et b1 = 2ⁿ

Si la base b2 correspond à la base 2, et la base b1 de départ est une puissance de 2 ($b1 = 2^n$), et si on regarde le plus grand chiffre que l'on peut écrire dans la base b1, on se rend compte qu'il faut n bits pour coder ce chiffre en binaire. Donc chaque chiffre du nombre N sera exprimé sous la forme d'un nombre binaire ayant n bits.

$$\left. \begin{array}{l} (3)_8 = (011)_2 \\ (5)_8 = (101)_2 \\ (7)_8 = (111)_2 \end{array} \right\} (357)_8 = (011 \ 101 \ 111)_2$$

** b1 = 2 et b2 = 2ⁿ

Pour passer du binaire vers une base b2 qui est une puissance de 2 ($b2 = 2^n$), on regroupe les bits n par n pour obtenir N codé dans la base b1.

$$(011\ 101\ 111)_2 = (357)_8$$

**Généralisation: b1 = 2ⁿ et b2 = 2^m

Si les bases de départ et d'arrivée sont des puissances de 2, on utilise la base 2 comme base intermédiaire pour faire le transcodage.

base b1	→	base 2	→	base b2
(22) ₁₆	→	(0010 0010) ₂	→	(42) ₈
(7F) ₁₆	→	(0111 1111) ₂	→	(177) ₈
(1257) ₈	→	(001 010 101 111) ₂	→	(2AF) ₁₆

*Cas du BCD: un cas particulier intéressant c'est quand la base b1 est égale à dix. Dans ce cas si on applique la même méthode que précédemment, chaque bit du nombre N sera codé sur 4 bits. Le résultat obtenu n'est pas du binaire mais du décimal codé en binaire (**BCD** pour Binary Coded Decimal). Pour marquer la différence avec le binaire et éviter les erreurs d'interprétation, on utilise soit une mise entre parenthèses avec une indication de la base, soit on sépare chaque groupe de 4 bits par un point.

base b1	→	base 10	→	BCD
(42) ₈	→	(34) ₁₀	→	(0011 0100) _{BCD}
(7F) ₁₆	→	(127) ₁₀	→	(0001. 0010. 0111)
(10) ₁₆	→	(16) ₁₀	→	(0001 0110) _{BCD}
(10) ₁₀	→	(10) ₁₀	→	(0001 0000) _{BCD}

Remarque: $(10)_{16} = (00010000)_2$ et $(10)_{10} = (00010000)_{BCD} \equiv 0001.0000$

On remarque que les nombres $(10)_{10}$ et $(10)_{16}$ qui correspondent à deux valeurs décimales différentes (10 et 16 respectivement) donnent la même suite de bits, d'où l'intérêt de les différencier.

-2°-2-Nombres fractionnaires

-2°-2-a-Codage décimal-binaire

La partie entière est traitée par divisions successives. La partie fractionnaire par multiplications successives, les parties entières successives obtenues constituent le résultat écrit dans l'ordre des puissances croissantes de la base. On arrête les multiplications quand la précision voulue est obtenue.

Codage de $(4,345)_{10}$ en base deux avec une précision de 2^{-5} (5 chiffres après la virgule) :

Partie entière : $(4)_{10} = (10)_2$

Partie fractionnaire : $(0,345)_{10} = (01011)_2$.

En effet en procédant par multiplications successives on trouve :

$$0,345 \times 2 = 0,690 \rightarrow \text{partie entière du résultat égale 0}$$

$$0,69 \times 2 = 1,38 \rightarrow \text{partie entière du résultat égale 1}$$

$$0,38 \times 2 = 0,76 \rightarrow \text{partie entière du résultat égale 0}$$

$$0,76 \times 2 = 1,52 \rightarrow \text{partie entière du résultat égale 1}$$

$$0,52 \times 2 = 1,04 \rightarrow \text{partie entière du résultat égale 1}$$

Par conséquent $(4,345)_{10} = (10,01011)_2$.

Codage de $(9,375)_{10}$ en base deux avec une précision de 2^{-3} (3 chiffres après la virgule) :

Partie entière : $(9)_{10} = (1001)_2$

Partie fractionnaire : $(0,375)_{10} = (011)_2$.

En effet en procédant par multiplications successives on trouve :

$$0,375 \times 2 = 0,750 \rightarrow \text{partie entière du résultat égale 0}$$

$$0,75 \times 2 = 1,5 \rightarrow \text{partie entière du résultat égale 1}$$

$$0,5 \times 2 = 1,00 \rightarrow \text{partie entière du résultat égale 1}$$

Par conséquent $(9,375)_{10} = (1011,011)_2$.

-2°-2-b-Codage Base b \rightarrow décimal

Il suffit d'appliquer la formule donnée par le principe général (cf -I-1-).

$$(10,11)_2 = 0 \times 2^0 + 1 \times 2^1 + 1 \times 2^{-1} + 1 \times 2^{-2} = 0 + 2 + 0,5 + 0,25 = (2,75)_{10}$$

$$(101,101)_2 = 1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = 1 + 4 + 0,5 + 0,125 = (5,625)_{10}$$

$$(70,40)_8 = 0 \times 8^0 + 7 \times 8^1 + 4 \times 8^{-1} + 0 \times 8^{-2} = 0 + 56 + 4/8 + 0 = (56,50)_{10}$$

-II-REPRESENTATION DES NOMBRES EN MEMOIRE

-3°-Nombres entiers

Enoncé

-3°-1-Codage : En utilisant un codage sur 5 bits, donner l'équivalent binaire des valeurs décimales suivantes, pour chacun des trois codes étudiés (Valeur Absolue Sans Signe, Valeur Absolue plus Signe, Complément à 2) : +7,+9,+13,-7,-9,-13

-3°-2-Décodage : Considérons chacune des chaînes de bits suivantes. Si elle correspond à un codage d'un nombre dans chacun des trois codes précédemment cités, donner pour chaque chaîne les trois valeurs décimales dont c'est le code. (00111), (10000), (11111), (10001)

Solution

-3°-1-Codage binaire en mémoire sur 5 bits

Nombre décimal	Technique Valeur absolue sans signe	Technique Valeur absolue + signe	Technique Complément à 2
+7	00111	00111	00111
+9	01001	01001	01001
+13	01101	01101	01101
-7	00111	10111	11001
-9	01001	11001	10111
-13	01101	11101	10011

-3°-2-Décodage d'une série de bits

Série de bits	Equivalent décimal Tech.val.abs.ss signe	Equivalent décimal Tech.val.abs.+ signe	Equivalent décimal Tech.complément à 2
00111	7	+7	+7
10000	16	-0	-16
11111	31	-15	-1
10001	17	-1	-15

4°Nombres fractionnaires

Enoncé

-4°1-Technique de la virgule fixe : Si on utilise 16 bits pour coder les nombres, dont 12 pour la partie entière et 4 pour la partie fractionnaire, donner la précision, et les valeurs Nmin et Nmax des nombres que l'on peut coder.

-4-2°-Technique de la virgule flottante : Si on utilise 16 bits pour coder les nombres (4 pour l'exposant et 12 pour la mantisse), donner la précision, les valeurs Nmin et Nmax des nombres que l'on peut coder.

Solution**-4°-1-Technique de la virgule fixe**

Partie fractionnaire : on utilise 4 bits (soit 4 chiffres après la virgule), donc la précision est de 2^{-4} ($= 6,25.10^{-2}$ en décimal). Par conséquent la valeur min de la partie fractionnaire est 0.0000 et la valeur max est de 0.1111.

Partie entière : on utilise 11 bits pour le codage plus un bit de signe. Les valeurs extrêmes que l'on peut obtenir pour la partie entière correspondent aux 11 bits égaux à 1. les valeurs correspondantes en décimal sont $+(2^{11} - 1)$ et $-(2^{11} - 1)$, soit ± 2047 .

Par conséquent les valeurs extrêmes des nombres qu'on peut obtenir avec un codage sur 16 bits dont 12 pour la partie entière et 4 pour la partie fractionnaire sont :

$$\text{ExposantMax} = + (111\dots 1, 1111) \approx +2047,9375$$

$$\text{ExposantMin} = - (111\dots 1, 1111) \approx -2047,9375$$

-4°-2-Technique de la virgule flottante

On a utilisé 12 bits pour la mantisse dont un bit de signe. Comme la mantisse est purement fractionnaire, la précision est donc de 2^{-11} . Par conséquent la valeur min de la mantisse est de $-0,11\dots 1$ et la valeur max est de $+0,11\dots 1$ (séries de 11 fois la valeur 1).

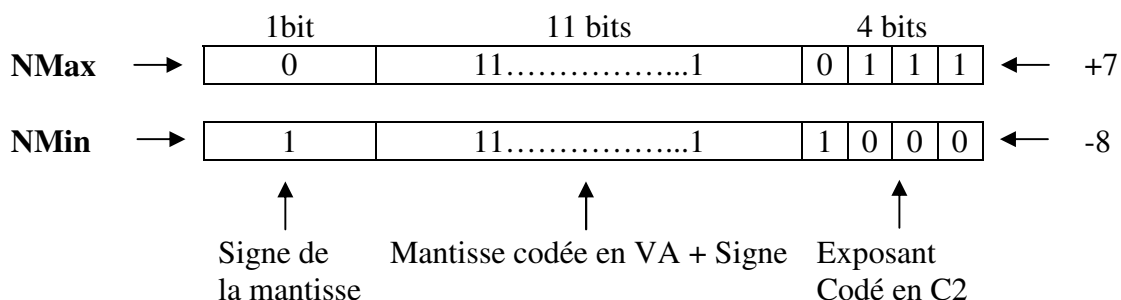
Pour coder l'exposant on utilise 4 bits, par conséquent les valeurs extrêmes de l'exposant codé en complément à deux sont :

$$\text{ExposantMmax} = (+2^{(n-1)} - 1) = (+2^{(4-1)} - 1) = +7 \quad \text{Exposantmin} = -2^{(n-1)} = -8.$$

Par conséquent les valeurs extrêmes des nombres qu'on peut obtenir sont

$$N_{\text{max}} = +(0,11\dots 1). 2^{+7} \quad N_{\text{min}} = -(0,11\dots 1). 2^{-8}.$$

Remarque : on a fait un mauvais choix dans la répartition des bits pour l'exposant et la mantisse. En effet par rapport à la technique de la virgule fixe, si on a gagné en précision (passage de 2^{-4} à 2^{-11}), par contre on a perdu en ce qui concerne la plus grande valeur qu'on peut coder (de 2^{11} à 2^7). En fait il suffit de mettre 10 bits pour la mantisse et 6 bits pour l'exposant pour gagner sur les deux tableaux.



-III-OPERATIONS ELEMENTAIRES EN ARITHMETIQUE NON SIGNEE

-5°Addition

Enoncé

-5°-1-Binaire : définir les notions de Carry (retenue) et Overflow (débordement)

-5°-2-Hexadécimal : Faire l'addition de $(23)_{16}$ et $(E9)_{16}$ de 2 méthodes différentes (directement et en passant par le binaire).

-5°-3-BCD : Additionner 0010.1001 et 0001.0101, puis donner l'équivalent décimal du résultat.

Solution

-5°-1-Binaire

$0 + 0 = 0$; $0 + 1 = 1$; $1 + 0 = 1$; $1 + 1 = 0$ et retenue = 1. La retenue est appelée *Carry*.

Quand le résultat de l'addition de deux nombres codés sur n bits ne peut pas tenir sur n bits, il y a un dépassement de capacité et on parle alors de *débordement* ou *overflow*.

-5°-2-Hexadécimal

On peut faire l'addition de 2 manières : soit directement, soit en passant par le codage en binaire.

$$\begin{array}{r}
 (23)_{16} \quad 0010 \ 0011 \\
 (E9)_{16} \quad 1110 \ 1001 \\
 \hline
 (10C)_{16} \quad \underbrace{1}_{1} \underbrace{0000}_{0} \underbrace{1100}_{C}
 \end{array}$$

-5°-3-BCD

Pour faire l'addition on procède comme en binaire, puis si le résultat est supérieur à 9 (1001 en binaire), on rajoute 6 (0110 en binaire) au résultat pour sauter les six positions binaires interdites en BCD, soit 1010,1011,...1111 (correspondant aux valeurs décimales comprises entre 10 et 15).

$$\begin{array}{rcl}
 & 0010 \ 1001 & \text{bcd} \quad (29)_{10} \\
 + & \underline{0001 \ 0101} & \text{bcd} \quad + (15)_{10} \\
 & 0011 \ 1110 & \text{binaire} \quad (44)_{10} \\
 & \underbrace{}_{\text{binaire} > (9)_{10} \Rightarrow \text{donc on doit corriger le résultat en rajoutant 6.}} & \\
 + & \underline{0000 \ 0110} & \text{bcd} \\
 & \underbrace{0100} \ \underbrace{0100} & \text{bcd} \\
 & 4 \ 4 & (44)_{10}
 \end{array}$$

6°Soustraction binaire en complément à 2**Enoncé**

Effectuer en binaire les opérations $13 - 5$ et $5 - 13$.

Solution

$$A - B = A + (-B) = A + C_2(-B)$$

6°-1-Soustraction $(13 - 5)_{10}$

$$\begin{array}{r}
 B \quad 0101 \\
 C1(B) \quad 1010 \\
 + \quad 1 \quad 0001 \\
 \hline
 C2(-B) \quad 1011 \\
 + \quad A \quad 1101 \\
 \hline
 1 \ 1000 \leftrightarrow (8)_{10} \\
 \uparrow
 \end{array}$$

retenue égale à 1 \rightarrow la retenue est à ignorer ;
le résultat est positif ($A > B$). Donc le résultat est $(8)_{10}$.

6°-2-Soustraction $(5 - 13)_{10}$

$$\begin{array}{r}
 B \quad 1101 \\
 C1(B) \quad 0010 \\
 + \quad 1 \quad 0001 \\
 \hline
 C2(-B) \quad 0011 \\
 + \quad A \quad 0101 \\
 \hline
 1000 \\
 \uparrow
 \end{array}$$

retenue = 0 \rightarrow le résultat obtenu est le complément à 2 d'un nombre négatif (cela signifie $A < B$). Pour le trouver il suffit calculer le complément à 2 du résultat :

$$C2(1000)_2 = (1000)_2. \text{ Donc le résultat est } (-8)_{10}.$$

7°Multiplication et division binaires d'un nombre exprimé sur n bits**Enoncé**

Montrer comment on peut effectuer la multiplication et la division par une puissance de deux à l'aide de décalages successifs.

Solution

7°-1-Multiplication par 2

Elle s'obtient par un décalage à gauche des bits. S'il y a débordement de capacité (bit overflow $V = 1$), il faut donc rajouter 2^n au résultat (nombres codés sur n bits). Pour retrouver le véritable résultat quel que soit V, il faut rajouter $V \cdot 2^n$ au résultat intermédiaire.

7°-1-a-Exemple avec $V = 0$

$$\begin{array}{r}
 \boxed{0} \quad 0 \ 1 \ 1 \ 1 \quad \boxed{0} \\
 V \quad \quad (7)_{10} \quad C
 \end{array}
 \quad \text{Décalage} \rightarrow \quad
 \begin{array}{r}
 \boxed{0} \quad 1 \ 1 \ 1 \ 0 \quad \boxed{0} \\
 V \quad \quad (14)_{10} \quad C
 \end{array}$$

$V = 0 \rightarrow$ le résultat est correct

7°-1-b-Exemple avec $V = 1$

$$\begin{array}{ccc}
 \boxed{0} & 1 & 0 & 0 & 1 & \boxed{0} \\
 V & & & & & C \\
 & & & & & (9)_{10}
 \end{array}
 \xrightarrow{\text{Décalage}}
 \begin{array}{ccc}
 \boxed{1} & 0 & 0 & 1 & 0 & \boxed{0} \\
 V & & & & & C \\
 & & & & & (2)_{10}
 \end{array}$$

$$V = 1 \rightarrow \text{résultat final} = (2)_{10} + 1 \cdot 2^4 = (18)_{10}$$

7°-2-Division par 2

Elle s'obtient par un décalage à droite des bits. Si le bit carry est nul ($C=0$) le résultat est entier. Si $C=1$ (le bit qui tombe dans le carry est un 1) le véritable résultat est un nombre fractionnaire, il faut donc rajouter 0,5 au résultat. Pour retrouver le véritable résultat quel que soit C , il faut rajouter $C \cdot 2^{-1}$ au résultat intermédiaire.

$$\begin{array}{ccc}
 1 & 0 & 1 & 0 & \boxed{} \\
 & & & & \text{carry} \\
 & & & & (10)_{10}
 \end{array}
 \xrightarrow{\text{Décalage}}
 \begin{array}{ccc}
 0 & 1 & 0 & 1 & \boxed{0} \\
 & & & & \text{carry} \\
 & & & & (5)_{10}
 \end{array}$$

Carry = 0 \rightarrow le résultat est un nombre entier.

$$\begin{array}{ccc}
 1 & 1 & 0 & 1 & \boxed{} \\
 & & & & \text{carry} \\
 & & & & (13)_{10}
 \end{array}
 \xrightarrow{\text{Décalage}}
 \begin{array}{ccc}
 0 & 1 & 1 & 0 & \boxed{1} \\
 & & & & \text{carry} \\
 & & & & (6)_{10}
 \end{array}$$

Carry = 1 \rightarrow le résultat est un nombre fractionnaire \rightarrow résultat final = $(6,5)_{10}$.

-IV-CODAGE ET TRANSMISSION

-8°-Codes Baudot et Iso

Enoncé

-8°-1-Donner la chaîne de bits correspondant au message suivant : « TZ 58K », lors de sa transmission en code Baudot.

-8°-2-Mêmes questions que pour l'exercice 8°-1, si le codage s'effectue en code ISO, avec pour la transmission : b.p. impaire + 1b.start + 2b.stop.

Solution

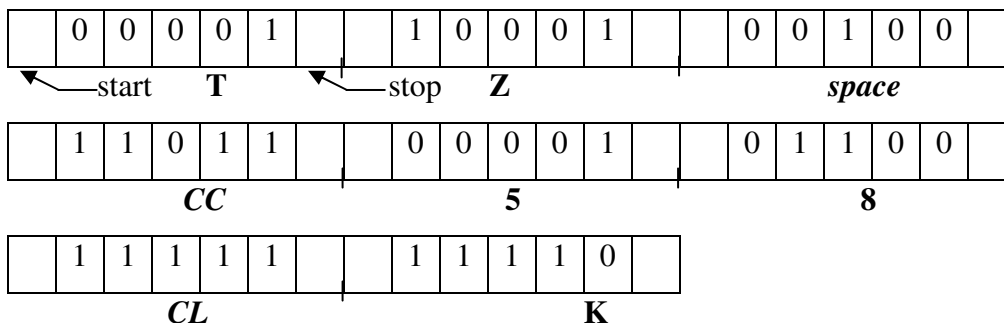
8°-1-Code Baudot

Appelons *CC* et *CL* les caractères de contrôle correspondant au code chiffre (1B) et au code lettre (1F). Les caractères circulant sur la ligne lors de la transmission en code Baudot du message « TZ 58K » sont :

Caractères : T Z space CC 5 8 CL K

Codage : (01)₁₆ (11)₁₆ (04)₁₆ (1B)₁₆ (01)₁₆ (0C)₁₆ (1F)₁₆ (1E)₁₆

Transmission : codage sur 5bits, et on rajoute un bit start=0 et un bit stop=1 pour chaque caractère.

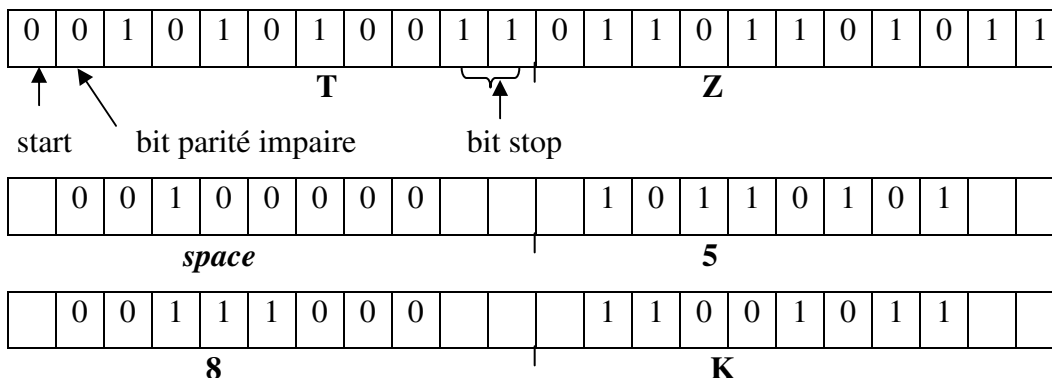


8°-2-Code Iso

Caractères : T Z space 5 8 K

Codage : (54)₁₆ (5A)₁₆ (20)₁₆ (35)₁₆ (38)₁₆ (4B)₁₆

Transmission : codage sur 7 bits plus: 1bit start, 2 bits stop, 1 bit de parité impaire



9°Codage , transmission et impression

Enoncé

Considérons le message suivant::

AID SP LF 98 SP CR 1 SP A SP 2 SP MILLIONS LF PAR SP MOUTON? CR SP SP ON SP DELIRE ?

SP = space (espace) ; LF = Line Feed (saut de ligne) ; CR = Carriage Return (retour chariot).

9°-1-Donner pour chacun des 4 codes connus, le nombre de bits total nécessaire pour coder ce message.

9°-2-Si le message est transmis en mode asynchrone non standard (7bits + bp paire + 1bit start + 2bits stop), donner le nombre de bits total reçu par l'imprimante pour chacun des 4 codes.

9°-3-Donner la forme du message tel qu'il apparaît sur papier après impression par l'imprimante.

Solution

Il y a 48 caractères dans le message.

9°-1-Codage

9°-1-a-Baudot : codage sur 5 bits

Le nombre de caractères d'échappement nécessaires est de 9. Par conséquent le nombre total de caractères est $48+9=57$ caractères. Le nombre de bits nécessaires pour le codage est : $57 \times 5 = 285$ bits.

9°-1-b-Iso et Ascii : codage sur 7 bits

Le nombre de bits est : $48 \times 7 = 336$ bits.

9°-1-c-Ebcdic : codage sur 8 bits

Le nombre de bits est : $48 \times 8 = 384$ bits.

9°-2-Transmission

Dans cette transmission en mode asynchrone (non standard) on a rajouté 4 bits par caractère : 1start, 1parité, 2 stop.

9°-2-a-Baudot

On doit rajouter $57 \times 2 = 114$ bits. Le nombre total de bits est de $285+114=399$.

9°-2-b-Iso et Ascii

On doit rajouter $48 \times 4 = 192$ bits. Le nombre total de bits est de $336+192=528$.

9°-2-c-ebcdic

On doit rajouter $48 \times 4 = 192$ bits. Le nombre total de bits est de $384 + 192 = 576$ bits.

9°-3-Impression

La forme du message tel qu'imprimé par l'imprimante est la suivante (le symbole X désignant l'espace) :

A I D X
9 8 X
1 X A X 2 X M I L L I O N S
P A R X M O U T O N ?
X X O N X D E L I R E ?

Remarque : l'espace (sp) à la fin de la 2^o ligne est inutile, car le caractère suivant est un retour chariot (CR).

SOLUTIONS DU TD CHAPITRE 3
« LES MEMOIRES »**EXERCICE 1****Enoncé**

Si un ordinateur possède une mémoire de 1Mo, quelle la taille de sa M.C. exprimée :

1° en mots de 32 bits? 2° en mots de 64 bits ?

Solution

a°) 1 Mo = 2^{20} octets = $2^{20} \times 2^3$ bits = 2^{23} bits = $[(2^{23}) / (32)]$ mots de 32 bits = $(2^{23} / 2^5)$ mots de 32 bits = 2^{18} mots de 32 bits = $2^8 \times 2^{10}$ mots de 32 bits = 256 Kmots de 32 bits.

b°) 1 Mo = 2^{20} octets = $2^{20} \times 2^3$ bits = 2^{23} bits = $[(2^{23}) / (64)]$ mots de 64 bits = $(2^{23} / 2^6)$ mots de 64 bits = 2^{17} mots de 64 bits = $2^7 \times 2^{10}$ mots de 64 bits = 128 Kmots de 64 bits.

EXERCICE 2**Enoncé**

Trouver le nombre de bits nécessaire pour adresser une mémoire de 512 mots de 16 bits chacun.

Solution

Pour l'adressage, on n'a pas besoin de connaître le nombre de bits dans un mot. Ce qui est important c'est le nombre de mots ou d'emplacements mémoire, car on adresse des emplacements mémoire et non des bits.

Avec un bus d'@ à n bits ou n fils, on peut fabriquer 2^n adresses, donc adresser 2^n emplacements.

Taille mémoire = 512 mots de 16 bits = 2^9 emplacements de 16 bits → il faut 9 bits pour adresser 512 mots.

EXERCICE 3**Enoncé**

On considère une mémoire de 65536 mots de 25 bits chacun. Donner le nombre de fils nécessaires sur le bus d'adresses et sur le bus de données.

Solution

1° 65536 mots de 25 bits = 64×1024 mots = $2^6 \times 2^{10} = 2^{16}$ mots → il faut 16 bits sur le bus d'adresses pour adresser les 2^{16} emplacements.

2° 1 emplacement mémoire = 1 mot de 25 bits → nécessité de 25 fils sur le bus de données si on veut effectuer les transferts avec la mémoire en une seule opération.

EXERCICE 4**Enoncé**

Les adresses d'une mémoire désignent des bytes (octets). Quelle est l'adresse du quatrième élément d'une table qui commence à l'adresse $(0019)_H$ et qui est constituée d'éléments de 16 bits ?

Solution

adresse	emplacements mémoire	N° élément de la table
0019		} 1° élément
001A		
001B		} 2° élément
001C		
001D		} 3° élément
001E		
001F		} 4° élément
0020		

Un élément = 16 bits = 2 x 8 bits = 2 octets
= 2 emplacements mémoire

L'adresse du 4° élément de la table est $(001F)_{16}$
On donne l'adresse de début de l'information, c'est-à-dire l'@ du 1° emplacement mémoire occupé par la donnée.

EXERCICE 5**Enoncé**

Un emplacement mémoire de longueur 16, a le contenu suivant (stocké dans la technique Big Endian): 0011 1000 0011 0101. Quelle est sa signification s'il s'agit:

1° d'un entier en représentation binaire? 2° d'une chaîne de caractères en code ISO?

Solution

1° 0011 1000 0011 0101 \longleftrightarrow (3 8 3 5)₁₆
(3835)₁₆ = (14389)₁₀

2° (38 35)₁₆ \longleftrightarrow (85)_{ISO} . En effet (38)₁₆ = code du caractère « 8 » en ISO,
et (35)₁₆ = code du caractère « 5 » en ISO.

EXERCICE 6**Enoncé**

La mémoire d'un ordinateur est organisée en mots de 32 bits. On veut stocker la chaîne de caractères "EPSIMA" et le nombre entier $(14389)_{10}$. Donner la disposition des chaînes de bits correspondantes (en code ISO) dans les deux techniques "Big Endian" et "Little Endian".

Solution

Chaîne de caractères : E P S I M A

Codes iso correspondants : 45 50 53 49 4D 41

$$(14389)_{10} = (3835)_{16}$$

1 emplacement mémoire fait 32 bits.

1° Big Endian

@	NUMEROS DES BITS			
	31 24	23 16	15 8	7 0
000E	E 0100 0101	P 0101 0000	S 0101 0011	I 0100 1001
000F	M 0100 1101	A 0100 0001	----	----
0010	3 8 0011 1000	3 5 0011 0101	----	----
0011				

-a-La chaîne de caractères fait 6 caractères, comme chacun est codé sur 8 bits, on aura donc 48 bits à stocker. On aura besoin pour cela de 2 emplacements mémoire.

On stocke les bits de poids les plus forts de chaque caractère de la chaîne, dans les bits de poids les plus forts de l'emplacement mémoire. Mais on fait attention pour que les caractères de la chaîne gardent le même ordre.

On remarque que les bits 0 à 15 du 2° emplacement sont inutilisés et perdus.

-b- $(3835)_{16}$: c'est de l'hexadécimal qu'on doit coder en binaire → chaque « digit » sera codé sur 4 bits. De plus on utilise les bits de poids le plus fort en premier : aussi bien pour les bits du nombre que ceux de l'emplacement mémoire. On remarque ici également que les bits 0 à 15 du 3° emplacement sont inutilisés et perdus.

2° Little Endian

@	NUMEROS DES BITS			
	31 24	23 16	15 8	7 0
000E	I 0100 1001	S 0101 0011	P 0101 0000	E 0100 0101
000F	----	----	A 0100 0001	M 0100 1101
0010	----	----	3 8 0011 1000	3 5 0011 0101
0011				

Le principe est le même que pour la technique Big Endian, la seule différence étant qu'on traite ici les bits de poids les plus faibles en premier, aussi bien pour les données que pour les emplacements mémoire.

On remarque que les bits inutilisés et « gaspillés » au niveau des emplacements mémoire sont les bits de poids les plus forts (ce qui est normal car on utilise ceux de poids le plus faible en premier).

EXERCICE 7**Enoncé**

Quelle est l'adresse effective correspondant à une adresse de base de code hexadécimal (0019)_H et un déplacement de (0016)_H ?

Solution

$$@effective = @segment + offset = (0019)_H + (0016)_H = (002F)_H$$

EXERCICE 8**Enoncé**

On dispose de boîtiers mémoire de capacité 8Kx1byte, chacun étant muni d'une broche de sélection. Montrer comment on peut construire un module de mémoire de 64Kbytes, en faisant clairement ressortir le câblage des bus d'adresses et de données.

Solution

Nombre de boîtiers = $(64 \text{ K} \times 8 \text{ bits}) / (8 \text{ K} \times 8 \text{ bits}) = 8$ boîtiers
 mémoire de 64 K x 8 bits → mémoire organisée en mots de 8 bits → on prendra un bus de données à 8 bits.

Taille d'un mot mémoire = 8 bits

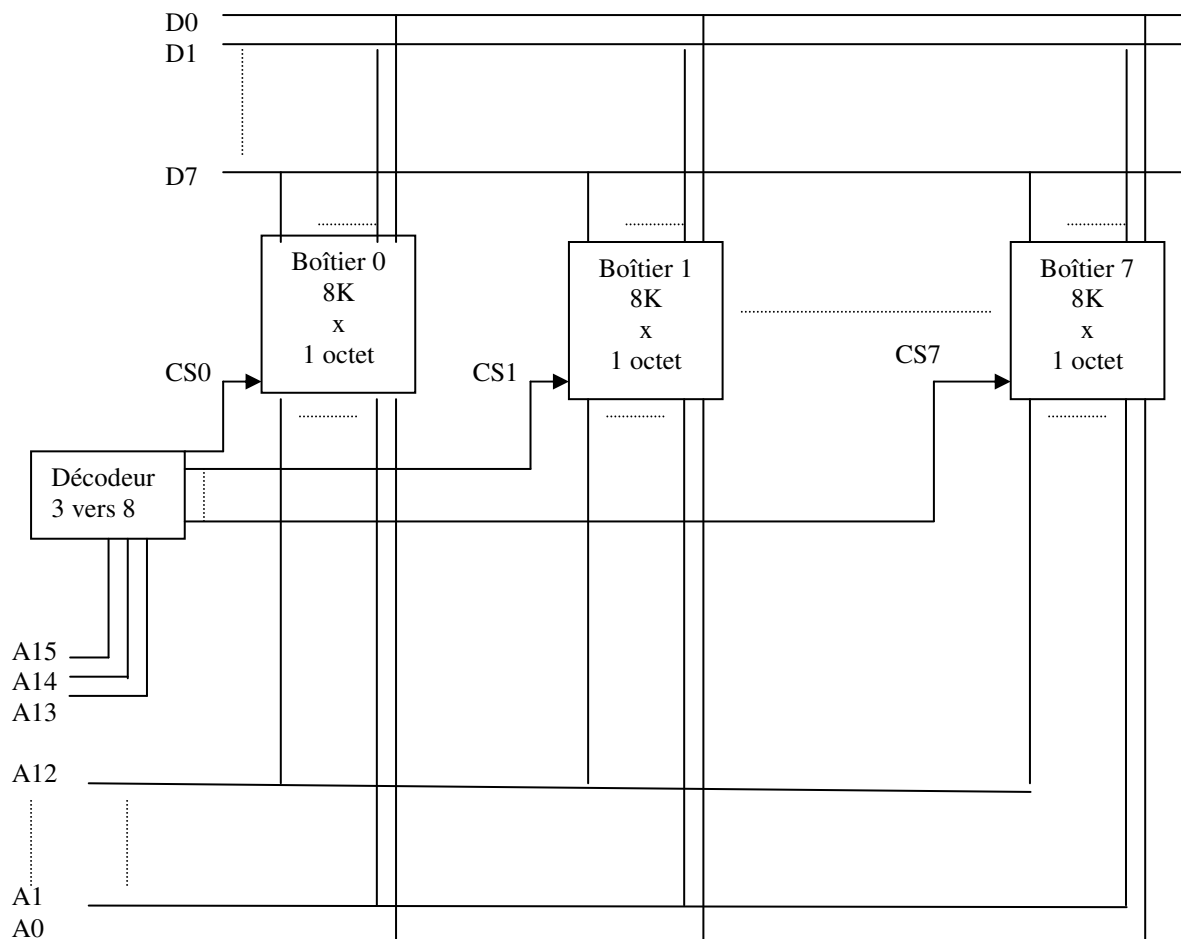
Taille d'un emplacement mémoire d'un boîtier = 8 bits

→ taille d'un mot mémoire = taille d'un emplacement du boîtier = 8 bits

Taille d'un boîtier = 8K emplacements de 8 bits = 2^{13} emplacements mémoire → il faut 13 bits pour adresser les 2^{13} E.M.

Comme on a 8 boîtiers, pour adresser les 2^3 boîtiers il faut 3 fils supplémentaires pour la sélection des boîtiers.

On aura donc au total 16 fils d'adresses : 13 pour adresser les emplacements et 3 pour la sélection des boîtiers. Ce nombre est conforme au nombre de fils d'adressage nécessaire d'après la taille mémoire. En effet une mémoire de 64 Kilo-octets = 2^{16} emplacements mémoire de 8 bits → il faut 16 fils pour adresser ces 2^{16} emplacements.



Exercice 8: Câblage de boîtiers mémoire de 8K x 1 octet pour réaliser 1 module de 64 Ko

EXERCICE 9**Enoncé**

1°Mêmes questions que pour l'exercice 8 avec boîtier = 4Kx4bits, et taille mémoire = 8Ko.

2°Mêmes questions que pour l'exercice 8 avec boîtier = 8Kx8bits, et taille mémoire = 64K mots de 16 bits.

3°Mêmes questions que pour l'exercice 8 avec boîtier = 4Kx2bits, et taille mémoire = 8Ko.

Solution

1°Nombre de boîtiers = taille mémoire / taille d'un boîtier = 8 K x 8 bits / 4 k x 4 bits = 4 boîtiers mémoire de 8 K octets → mémoire organisée en octets → on prendra un bus de données à 8 bits.

Taille d'un mot mémoire = 8 bits }

Taille d'un emplacement mémoire d'un boîtier = 4 bits

→ taille d'un mot mémoire / taille d'un emplacement du boîtier = 8 bits / 4 bits = 2 → il faut assembler les boîtiers 2 par 2 pour former des mots de 8 bits : le premier relié aux 4 bits de poids les plus faibles du bus de données, et le 2° aux 4 bits de poids les plus forts.

Taille d'un boîtier = 4K emplacements de 4 bits = 2^{12} emplacements mémoire → il faut 12 bits pour adresser les 2^{12} E.M.

Comme les boîtiers sont groupés deux par deux → on aura donc $(4 \text{ boîtiers} / 2) = 2^1$ groupes de boîtiers. Pour les adresser il faut 1 fil supplémentaire pour la sélection des groupes de boîtiers.

On aura donc au total 13 fils d'adresses : 12 pour adresser les emplacements et 1 pour la sélection des boîtiers. Ce nombre est conforme au nombre de fils d'adressage nécessaire d'après la taille mémoire. En effet mémoire de 8 K octets = 2^{13} emplacements mémoire d'un octet → il faut 13 fils pour adresser ces 2^{13} emplacements.

2°Nombre de boîtiers = $(64 \text{ K} \times 16 \text{ bits}) / (8 \text{ K} \times 8 \text{ bits}) = 16$ boîtiers mémoire de 64 K mots x 16 bits → mémoire organisée en mots de 16 bits → on prendra un bus de données à 16 bits.

Taille d'un mot mémoire = 16 bits }

Taille d'un emplacement mémoire d'un boîtier = 8 bits

→ taille d'un mot mémoire / taille d'un emplacement du boîtier = 16 bits / 8 bits = 2 → il faut assembler les boîtiers 2 par 2 pour former des mots de 16 bits.

Taille d'un boîtier = 8K emplacements de 8 bits = 2^{13} emplacements mémoire → il faut 13 bits pour adresser les 2^{13} E.M.

Comme les boîtiers sont groupés deux par deux → on aura donc $(16 \text{ boîtiers} / 2) = 8$ groupes de boîtiers. Pour adresser les 2^3 groupes il faut 3 fils supplémentaires pour la sélection des groupes de boîtiers.

On aura donc au total 16 fils d'adresses : 13 pour adresser les emplacements et 3 pour la sélection des boîtiers. Ce nombre est conforme au nombre de fils d'adressage nécessaire d'après la taille mémoire. En effet une mémoire de 64 K mots de 16 bits = 2^{16} emplacements mémoire de 16 bits → il faut 16 fils pour adresser ces 2^{16} emplacements.

3° Nombre de boîtiers = taille mémoire / taille d'un boîtier = $8 \text{ K} \times 8 \text{ bits} / 4 \text{ k} \times 2 \text{ bits} = 8$ boîtiers mémoire de 8 K octets → mémoire organisée en octets → on prendra un bus de données à 8 bits.

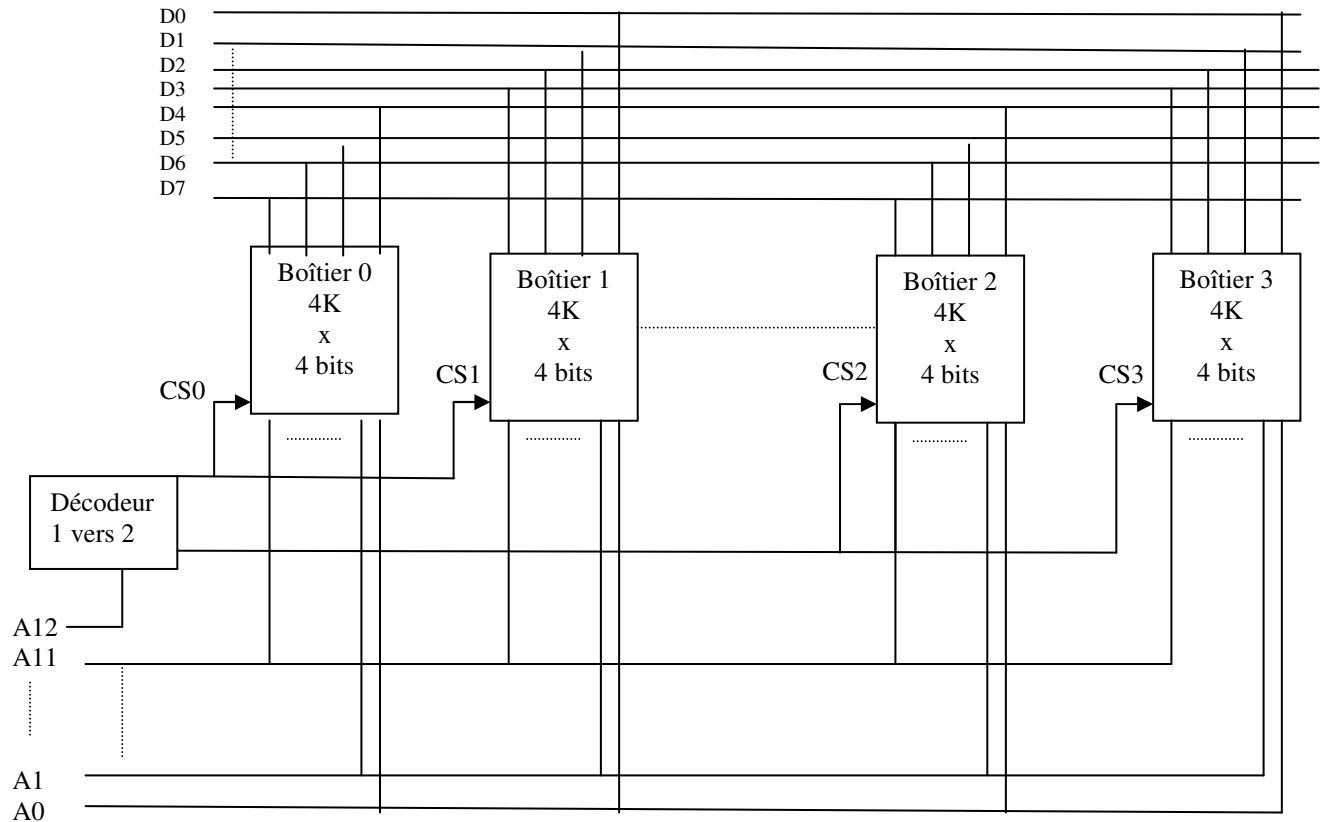
Taille d'un mot mémoire = 8 bits
Taille d'un emplacement mémoire d'un boîtier = 2 bits

→ taille d'un mot mémoire / taille d'un emplacement du boîtier = $8 \text{ bits} / 2 \text{ bits} = 4$ → il faut assembler les boîtiers 4 par 4 pour former des mots de 8 bits.

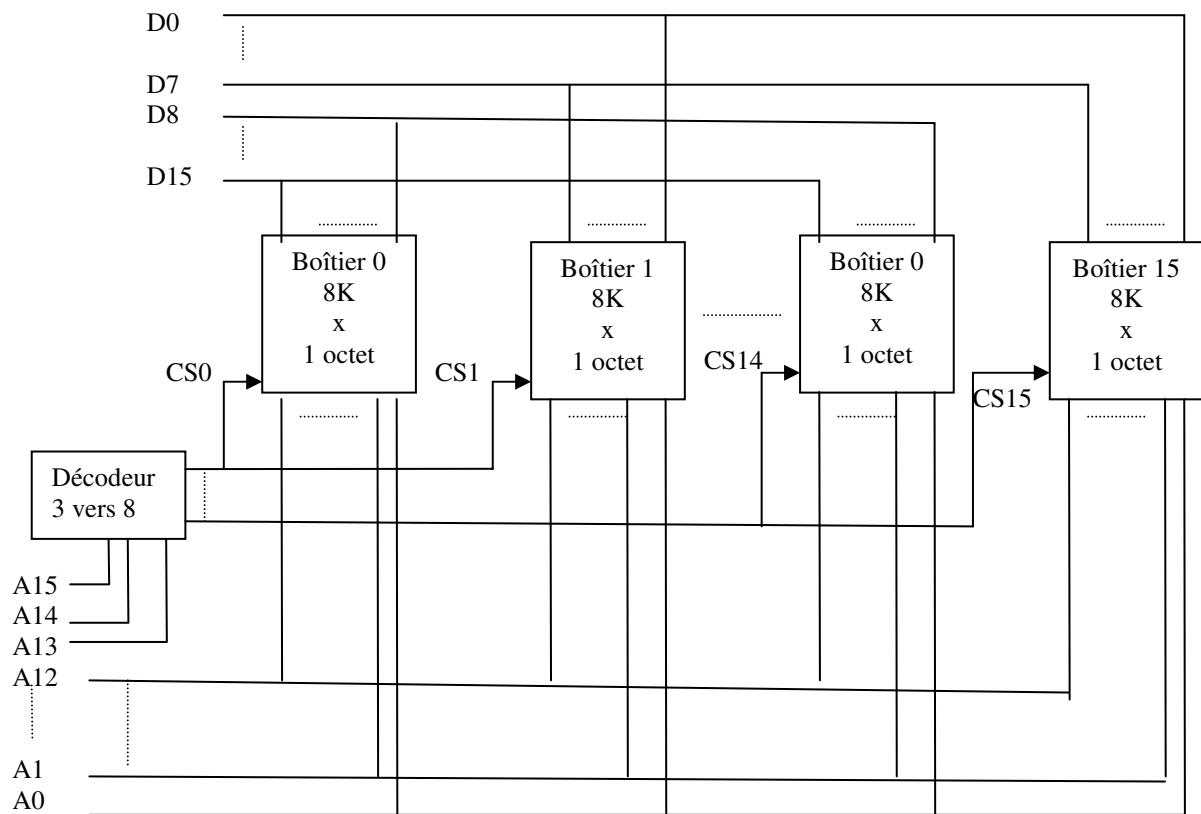
Taille d'un boîtier = 4K emplacements de 2 bits = 2^{12} emplacements mémoire → il faut 12 bits pour adresser les 2^{12} E.M.

Comme les boîtiers sont groupés quatre par quatre → on aura donc $(8 \text{ boîtiers} / 4) = 2^1$ groupes de boîtiers. Pour les adresser il faut 1 fil supplémentaire pour la sélection des groupes de boîtiers.

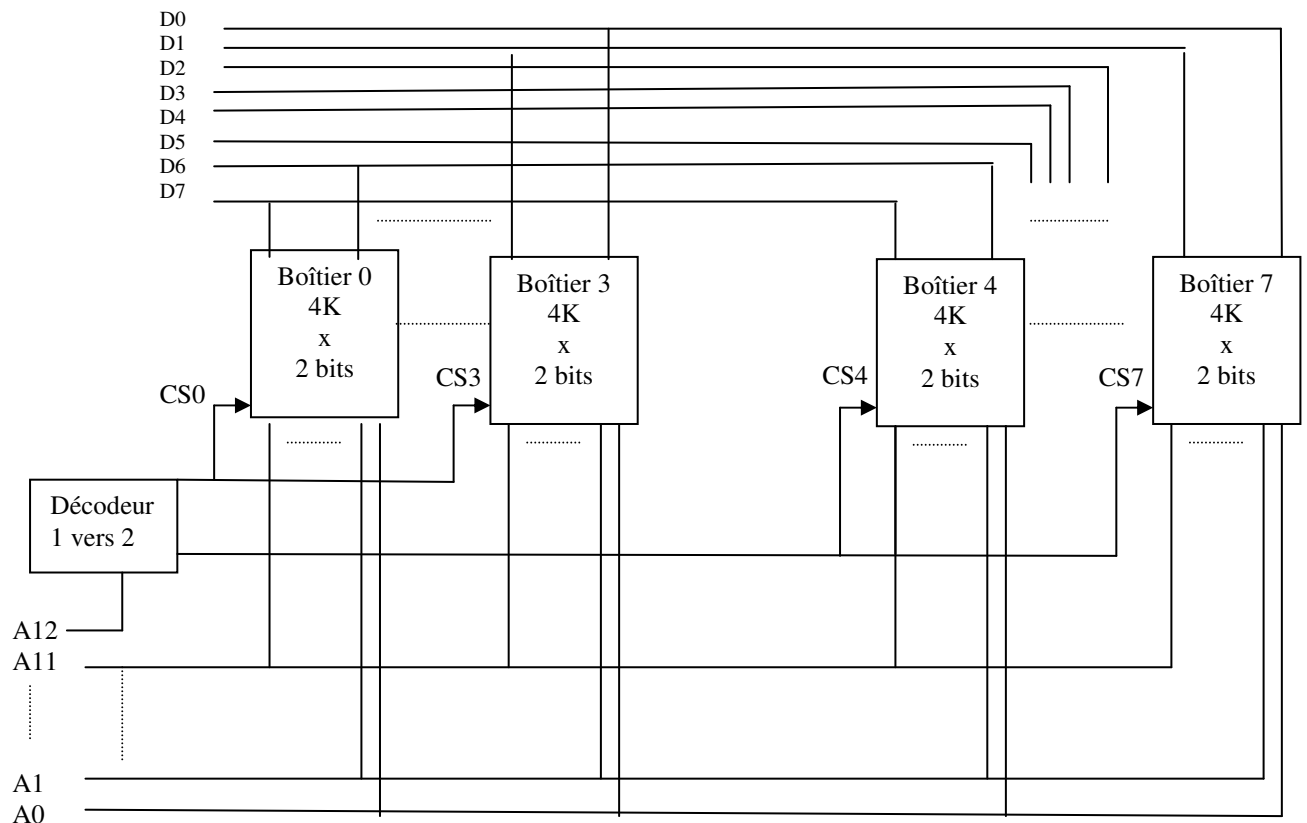
On aura donc au total 13 fils d'adresses : 12 pour adresser les emplacements et 1 pour la sélection des boîtiers. Ce nombre est conforme au nombre de fils d'adressage nécessaire d'après la taille mémoire. En effet une mémoire de 8 K octets = 2^{13} emplacements mémoire d'un octet → il faut 13 fils pour adresser ces 2^{13} emplacements.



Exercice 9-1 : Câblage de boîtiers mémoire de 4K x 4 bits pour réaliser 1 module de 8 Ko



Exercice 9-2 : module mémoire de de 64 Kmots de 16 bits avec des boîtiers de 8K x 1 octet



Exercice 9-3 : Câblage de boîtiers mémoire de 4K x 2 bits pour réaliser 1 module de 8 Ko

EXERCICE 10 (de consolidation !) *Toutes les questions sont indépendantes.*

Enoncé

Considérons deux ordinateurs : A avec une mémoire organisée en octets, et B avec une mémoire organisée en mots de 32 bits.

Soit la chaîne de caractères « DIMANCHEsp23spJANVsp00? » (*sp* signifie space = le caractère espace).

1°) Donner le nombre d'**emplacements-mémoire** nécessaires pour stocker la chaîne de caractères codée en ASCII, dans les cas des ordinateurs A et B.

2°) Si le bus d'adresses est à 8 fils, quelle est la taille de l'**espace adressable** dans chaque cas ?

3°) Si le bus de données est à 8 bits, quel est dans chaque cas le nombre d'**opérations d'échange** entre le processeur et le contrôleur de la mémoire, pour stocker en mémoire la chaîne de caractères codée en ASCII ?

4°) Si le bus de données est à 32 bits, même question que la question 3°).

5°) On veut stocker le message codé en ASCII en mémoire de l'ordinateur A, donner la disposition en mémoire des chaînes de bits correspondantes dans les deux techniques "Big Endian" et "Little Endian".

6°) Pour l'ordinateur B, même question que pour la question 5°)

7°) Donner pour les codes ISO et Baudot, le nombre de bits total nécessaire pour coder ce message.

8°) Si le message est transmis en mode asynchrone (1bit parité, 1bit start, 1bits stop), donner dans chacun des 2 cas (ISO et Baudot) le nombre de bits total reçus par l'imprimante.

Solution

1° Dans le stockage des caractères en mémoire, chaque caractère est codé en ascii sur 8 bits. La chaîne fait 20 caractères.

-a-Cas A : mémoire organisée en octets. Chaque caractère occupe un E.M. (emplacement mémoire). Donc on aura besoin de 20 E.M.

-b-Cas B : mémoire organisée en mots de 32 bits. Dans chaque E.M. on peut stocker $32/8 = 4$ caractères. Donc le nombre d'E.M. nécessaires est de $20/4 = 5$ E.M.

2° Si le bus d'adresses est à 8 fils, la taille de l'espace adressable est $2^8 = 256$ adresses (ou E.M.), et cela quel que soit le mode d'organisation de la mémoire, donc quelle que soit la taille d'un emplacement mémoire.

3° Bus de données 8 bits. Donc à chaque opération d'échange entre le processeur et la mémoire, on ne peut transporter qu'un caractère à la fois, et cela quel que soit le mode d'organisation de la mémoire ou la taille d'un E.M. Donc pour stocker le fichier on aura besoin de 20 opérations d'échange.

4° Bus de données de 32 bits.

-a-Cas A : comme un E.M. ne fait que 8 bits, on ne peut transférer qu'un octet (donc un caractère) à la fois. Le nombre d'opérations d'échange est donc égal à 20.

-b-Cas B : Un E.M. comprend 32 bits, et a la même capacité que le bus de données. Donc pour chaque opération d'échange, on transfère 32 bits (4 caractères) d'un seul coup. Le nombre d'opérations d'échange pour transférer le fichier est égal à $20/4 = 5$ opérations.

5° Cas A : mémoire organisée en octets.

On obtient exactement le même résultat dans les 2 cas Big et Little Endian :

Caractère et code hexa	Code Binaire
D 44	0100 0100
I 49	0100 1001
M 4D	0100 1101
A 41	0100 0001
N 4 ^E	0100 1110
C 43	0100 0011
H 48	0100 1000
E 45	0100 0101
sp 20	0010 0000
2 32	0011 0010
3 33	0011 0011
Sp 20	0010 0000
J 4A	0100 1010
A 41	0100 001
N 4E	0100 1110
V 56	0101 0110
Sp 20	0010 000
0 30	0011 0000
0 30	0011 0000
? 2F	0010 1111

6° Cas B : mémoire organisée en mots de 32 bits.**Technique Big Endian**

@	Code Binaire			
	32---25	24---17	16 ----9	8 ---- 1
00	D	I	M	A
01	N	C	H	E
02	Sp	2	3	Sp
03	J	A	N	V
04	Sp	0	0	?

Technique Little Endian

@	Code Binaire			
	32---25	24---17	16 ----9	8 ---- 1
00	A	M	I	D
01	E	H	C	N
02	Sp	3	2	Sp
03	V	N	A	J
04	?	0	0	sp

7°

-a-Code ISO : chaque caractère est codé sur 7 bits ➔ nombre de bits : $20 \times 7 = 140$ bits

-b-Code Baudot : On doit rajouter 3 caractères de contrôle pour les passages de chiffre à lettre et vice versa. Donc au total on aura 23 caractères. Chacun est codé sur 5 bits ➔ nombre de bits total : $23 \times 5 = 115$ bits.

8°

Pour la transmission on rajoute à chaque caractère 3 bits. Donc en code iso on va rajouter $20 \times 3 = 60$ bits, et en code baudot on rajoute $23 \times 2 = 46$ bits. La nombre total de bits reçus par l'imprimante est : en code iso : $140 + 60 = 200$, et en code baudot : $115 + 46 = 161$.

EXERCICE 11**Enoncé**

On considère un ordinateur disposant d'une antémémoire de temps de cycle 100ns. Sa mémoire centrale a un temps de cycle de 600ns.

1° Quel est le temps de cycle apparent de la mémoire si on suppose un taux de réussite de 80%?

2° Quel est le gain relatif par rapport à une architecture sans antémémoire?

Solution

Taux de réussite de 80 % signifie que dans 80 % des cas, quand on accède en mémoire cache on y trouve la donnée cherchée, et dans les 20% restants, on est obligé d'accéder en plus en ram pour trouver la donnée.

Le temps de cycle apparent : $t_{ca} = 80\% \times t_{c_memoire_cache} +$

$$20\% \times (t_{c_memoire_cache} + t_{c_ram}) = 0,8 \times 100ns + 0,2 (100 + 600) = 220 \text{ ns}$$

$t_{ca} < t_{c_ram} \rightarrow$ on a un gain en vitesse. $G = (t_{c_ram} - t_{ca}) / t_{c_ram} = |220 - 600| / 600$

$$= 380/600 = 0,63 \leftrightarrow 63\%$$

Remarque : le but de l'antémémoire est d'accélérer les traitements.

EXERCICE 12**Enoncé**

On considère un système de pagination, la table des pages est en M.C. qui a un temps de cycle de 750ns. Le temps d'accès au tampon de traduction d'adresses est de 50ns. La probabilité de trouver l'adresse cherchée dans le tampon (taux de réussite) est de 80%.

1° Quel est le temps d'accès moyen à une donnée se trouvant en M.C. ?

2° Quelle est la dégradation (perte de vitesse) par rapport à un système sans pagination.

Solution

Dans 80% des cas : accès au tampon pour traduction de l'adresse logique en adresse physique + accès en ram pour lire la donnée

Dans 20%des cas : accès au tampon sans succès (l'équivalent physique de l'adresse logique fournie ne se trouve pas dans le tampon) + accès à la table des pages (en ram) pour lire l'adresse physique + accès en ram pour lire la donnée.

Le temps de cycle apparent : $t_{ca} = 80\% (50 + 750) + 20\% (50 + 750 + 750) = 950 \text{ ns}$

$t_{ca} > t_{c_ram} \rightarrow$ perte de vitesse. $Perte = (t_{ca} - t_{c_ram}) / t_{c_ram} = |950 - 750| / 750$

$$= 0,26 \leftrightarrow 26\%.$$

Remarque : La pagination est très souvent associée au mécanisme de mémoire virtuelle. Le but est alors d'augmenter la taille mémoire, au détriment de la vitesse (dégradation). Pour calculer le temps de cycle apparent, ici nous n'avons tenu compte que du mécanisme de pagination et avons supposé que la donnée est toujours en mémoire RAM. En réalité la dégradation est beaucoup plus importante car souvent la donnée ne se trouve pas en RAM mais en mémoire virtuelle. Il faut alors rajouter l'accès à la mémoire virtuelle (mémoire de masse) qui prend beaucoup plus de temps qu'un accès à la ram.

EXERCICE 13

Enoncé

On considère un système de mémoire virtuelle basé sur la pagination. La table des pages est en mémoire centrale (ram). De plus l'ordinateur possède un tampon de traduction d'adresses. Donner les différentes séquences d'accès aux mémoires qui peuvent être nécessaires, sachant que :

- si une traduction d'adresse de page est dans le tampon de traduction, alors la page est obligatoirement présente en mémoire centrale ;
- dans la table des pages ne se trouvent que les adresses de pages virtuelles se trouvant en ram.

Solution

On considère un système de mémoire virtuelle basé sur la pagination, avec l'utilisation d'un buffer ou tampon de traduction d'adresses.

La table des pages est en mémoire centrale (ram). C'est une table de correspondance: pour une adresse logique (adresse de page virtuelle) elle donne son équivalent physique (adresse en RAM). Il faut remarquer que dans la table des pages ne se trouvent que les adresses de pages virtuelles dont l'équivalent physique se trouve en ram, en d'autres termes les adresses des données se trouvant en ram.

Le buffer ou tampon de traduction d'adresses donne pour une adresse logique son équivalent physique en RAM. Par conséquent si une traduction d'adresse de page est dans le tampon de traduction, alors la page est obligatoirement présente en mémoire centrale.

Les différentes séquences d'accès aux mémoires qui peuvent être nécessaires pour lire une donnée sont:

On commence par accéder au buffer. Deux cas peuvent se présenter:

-a- **adresse logique présente** → on trouve son équivalent physique en ram → on accède en ram pour lire la donnée. Donc au total: *accès au buffer + accès en ram pour lire la donnée.*

-b-adresse **logique absente** → on ne peut pas trouver son équivalent physique → on accède en ram pour rechercher l'adresse dans la table des pages. Deux cas peuvent se présenter:

-b-1-Après accès à la table des pages en ram, l'**adresse logique est présente** → on trouve son équivalent physique en ram → on accède à nouveau en ram pour lire la donnée.

Donc au total *accès au buffer + accès en ram pour trouver l'adresse physique + accès en ram pour lire la donnée.*

-b-2-Après accès à la table des pages en ram, l'**adresse logique est absente** → on accède à la mémoire virtuelle pour charger la page demandée, et donc lire la donnée. Donc au total *accès au buffer + accès en ram + accès en mémoire virtuelle (dique dur) pour transférer la page recherchée et donc pour lire la donnée.* Il est à remarquer que dans ce dernier cas tous les temps d'accès sont négligeables devant celui de l'accès au disque dur: en effet il se compte en millisecondes alors que les autres sont en nanosecondes.

EXERCICE 14

Enoncé

On considère maintenant un système de mémoire virtuelle basé sur la pagination. La table des pages est en mémoire centrale (ram). De plus l'ordinateur possède un tampon de traduction d'adresses et une mémoire cache. Donner les différentes séquences d'accès aux mémoires qui peuvent être nécessaires, ainsi que quelques séquences qui ne se produiront jamais, sachant que :

- si une traduction d'adresse de page est dans le tampon de traduction, alors la page est présente en mémoire centrale ;
- si un bloc de mémoire est dans la mémoire cache, alors la traduction d'adresse de sa page est obligatoirement dans le tampon de traduction ;
- dans la table des pages se trouvent toutes les adresses de pages virtuelles se trouvant en ram.

Solution

Les séquences suivantes sont possibles :

1. Accès au tampon de traductions (succès) + Accès à l'antémémoire (succès)
(traduction c'est l'adresse physique présente) (bloc de mémoire présent → lecture de la donnée)
2. Accès au tampon de traductions (succès) + Accès à l'antémémoire (échec) + Accès à la ram
(traduction c'est l'adresse physique présente) (bloc de mémoire absent) (lecture de la donnée)
3. Accès au tampon de traductions (échec)
 + Accès à la table des pages en ram (succès) + Accès à la ram
(page présente) (lecture de la donnée)

4. Accès au tampon de traductions (échec)

+ Accès à la table des pages en ram (échec) + Accès au disque

(page absente)

(lecture de la page donc de la donnée)

Il faut remarquer que, si la traduction de page est présente dans le tampon, on n'accède jamais à la mémoire virtuelle sur disque, car si une traduction d'adresse de page est dans le tampon de traduction, alors la page est obligatoirement présente en mémoire centrale. De même si elle est absente, il est inutile de consulter l'antémémoire (mémoire cache) car l'antémémoire ne contient qu'une copie des données se trouvant en mémoire centrale (ram) auxquelles on a accédé récemment. Par conséquent si le bloc de mémoire se trouvait dans la mémoire cache, alors la traduction d'adresse de sa page se trouverait obligatoirement dans le tampon de traduction.

On peut donc citer comme séquences impossibles:

- tampon de traduction (succès) + table des pages en ram avec succès (page présente) + disque
- tampon de traduction (échec) + mémoire cache (échec) + ram...
- tampon de traductions d'adresses (succès) + table des pages en ram (succès) + antémémoire
- tampon de traductions d'adresses (échec) + antémémoire (échec) + disque.

EXERCICE 15 :**Enoncé**

Une unité de disque dur possède 4 plateaux (disques) ayant les caractéristiques suivantes: une tête de lecture/écriture par disque, une seule tête fonctionne à un instant donné, 600 pistes par disque, 20 secteurs/piste, 512 octets par secteur, vitesse de rotation de 3000 tours par minute.

1° Quelle est la capacité de stockage de l'unité de disque?

2° Quel est son taux de transfert?

3° Quel est le temps de transfert d'un secteur ?

Solution

1°Capacité de stockage :

$$4 \times 600 \times 20 \times \frac{1}{2} \text{ Ko} = 24000 \text{ Ko}$$

$$V_r = 3000 \text{ tpm} = 3000 \text{ tr/1mn} = 3000 \text{ tr} / 60\text{s} = \mathbf{50 \text{ tr/s}}$$

2°Taux de transfert :

a°) première méthode : **Taux de transfert** = Vitesse de rotation x capacité d'une piste en octets:

$$50 \times (20 \times \frac{1}{2} \text{ Ko}) = \mathbf{500 \text{ Ko/s}} = 0,5 \text{ Mo/s}$$

b°deuxième méthode : directe

$$1 \text{ tour} = \text{Une piste} = 20 \text{ secteurs} \times \frac{1}{2} \text{ Ko} = 10 \text{ Ko}$$

$$\text{Vitesse} : 50 \text{ tr/s} = 50 \times 10 \text{ Ko} / \text{s} = 500 \text{ Ko/s}$$

3° Temps de transfert d'un secteur :

$$\begin{array}{ll}
 500 \text{ Ko} & \rightarrow 1 \text{ s} \\
 1 \text{ secteur} = \frac{1}{2} \text{ Ko} & \rightarrow x \text{ seconde}
 \end{array}
 \left. \vphantom{\begin{array}{l} 500 \text{ Ko} \\ 1 \text{ secteur} = \frac{1}{2} \text{ Ko} \end{array}} \right\} \rightarrow x = (1 \text{ s} \times 0,5 \cdot 10^3) / 0,5 \cdot 10^6 = 1 \cdot 10^{-3} \text{ s} = \mathbf{1 \text{ ms}}$$

EXERCICE 16 :**Enoncé**

On veut calculer le temps moyen de transfert en mémoire centrale d'un fichier séquentiel stocké sur disque dur. Les échanges entre l'unité de disque et la M.C. s'effectuent par l'intermédiaire d'un canal. Une zone de mémoire tampon associée au disque reçoit les caractères lus sur le disque; le transfert vers la M.C. s'effectue soit quand la zone tampon est pleine, soit quand le fichier a été entièrement lu. Les caractéristiques sont les suivantes:

Temps moyen de positionnement de la tête de lecture/écriture sur une piste: 20 ms; *Temps de dépôt ou de soulèvement de la tête de lecture écriture: 5 ms ; * Vitesse de rotation du disque: 6000 tr/mn; *Vitesse de transfert du canal: 10 Mo/s; * Taille du tampon: 5000 octets; * Taille du fichier 3000 caractères; *Codage d'un caractère: 8 bits; *Tous les caractères du fichier sont sur un même cylindre; *Nombre de cylindres: 128; *Nombre de secteurs par piste: 50; *Taille d'un secteur: 1024bits dont 24 sont réservés pour un pointeur vers le secteur suivant.

1° Quel est le nombre de secteurs nécessaire au stockage du fichier?

2° Quel est le temps moyen pour lire un secteur, en supposant que la tête est sur la bonne piste?

3° Quel est le temps moyen de transfert de ce fichier entre le disque dur et la M.C. ?

4° Même question que 3° si le fichier a une taille de 6000 caractères.

5° Même question que 3° si le fichier de 3000 caractères est stocké sur 3 cylindres différents

6° Même question que 3° si le fichier de 3000 caractères est toujours stocké sur 1 seul cylindre, et la taille du tampon est de 500 octets.

Solution

1° Nombre de secteurs théorique (nécessaire) :

$$= \text{taille fichier} / \text{taille d'un secteur} = (3000 \text{ caractères} \times 8 \text{ bits}) / 1024 = 23,44 = 24 \text{ secteurs}$$

$$\text{Capacité utile d'un secteur} = 1024 - 24 = 1000 \text{ bits}$$

$$\text{Nombre de secteurs utiles (nécessaire)} = (3000 \times 8) / 1000 = 24 \text{ secteurs}$$

2°-a) Temps de lecture ou de transfert d'un secteur :

1° méthode : en passant par le taux de transfert

$$\text{Taux de transfert} = V_r \times \text{capacité d'une piste}$$

$$V_r = 6000 \text{tpm} = 6000 \text{tr/1mn} = 6000 \text{tr/60s} = 100 \text{ tr/s}$$

Capacité d'une piste = nombre de secteurs/piste x taille d'un secteur

$$\text{Taux de transfert} = 100 \text{ tr/s} \times 50 \times 1024 \text{ bits} = 5000 \text{ K bits/s} = 5 \text{ Mbits/s}$$

$$\begin{array}{l} 1 \text{seconde} \rightarrow 5000 \text{ Kbits} \\ x \text{ seconde} \rightarrow 1 \text{ secteur} = 1 \text{Kbit} \end{array} \quad \left. \vphantom{\begin{array}{l} 1 \text{seconde} \\ x \text{ seconde} \end{array}} \right\} \rightarrow \text{temps de lecture d'un secteur:} \\ x = (1 \text{s} \times 1 \text{K}) / 5000 \text{K} = 0,2 \cdot 10^{-3} \text{s} = \mathbf{0,2 \text{ ms}}$$

2° méthode: directement par la vitesse

1 tour équivaut à une piste. La vitesse de rotation est de 100 tr/s donc 100 pistes / seconde. Pour faire un tour il faut: $1 \text{s} / 100 = 10 \cdot 10^{-3} \text{s} = 10 \text{ ms}$.

$$\begin{array}{l} \text{Donc: } 1 \text{ tr} = 50 \text{ secteurs} \rightarrow 10 \text{ ms} \\ 1 \text{ secteur} \rightarrow x \text{ seconde} \end{array} \quad \left. \vphantom{\begin{array}{l} 1 \text{ tr} \\ 1 \text{ secteur} \end{array}} \right\} \rightarrow \text{temps de lecture d'un secteur: } x = 10 \text{ms} / 50 = \mathbf{0,2 \text{ ms}}$$

2°-b) Le temps moyen de positionnement sur le bon secteur: on admet généralement qu'une fois qu'on est sur la bonne piste, il faut faire un demi tour pour trouver le bon secteur.

$$\begin{array}{l} 1 \text{tour} \rightarrow 10 \text{ ms} \\ \frac{1}{2} \text{ tour} \rightarrow x \text{ ms} \end{array} \quad \left. \vphantom{\begin{array}{l} 1 \text{tour} \\ \frac{1}{2} \text{ tour} \end{array}} \right\} \rightarrow x = 10 / 2 = \mathbf{5 \text{ ms}}.$$

Le temps moyen de lecture d'un secteur est égal au temps de positionnement sur le bon secteur + le temps de lecture ou de transfert du secteur.

$$\mathbf{T_{moy} = 5 \text{ ms} + 0,2 = 5,2 \text{ ms}.$$

3° -Tout le fichier sur un même cylindre \rightarrow on n'a pas besoin de déplacer le dispositif des têtes de lecture/écriture d'une piste à une autre, puisque toutes les pistes sont lues en même temps.

-De plus Taille fichier < taille du tampon \rightarrow **tout le fichier lu sur le disque est transféré à la mémoire tampon d'un seul coup.**

Par conséquent le temps de transfert du fichier du disque vers la mémoire tampon est la somme de 4 termes:

A= temps moyen de positionnement de la tête de lecture sur le bon cylindre (ou piste) +

B= temps de dépôt de la tête sur la piste +

C= temps de lecture des 24 secteurs (càd du fichier) +

D= temps de transfert du tampon vers la mémoire centrale.

A= temps moyen de positionnement sur la bonne piste ou le bon cylindre = **20 ms** (d'après l'énoncé).

B= temps de dépôt de la tête sur la piste = 5 ms .

C= nombre de secteurs x temps moyen de lecture d'un secteur = $24 \times 5,2 = 124,8 \text{ ms}$.

Calcul de D:

Vitesse de transfert ou débit du canal: $10 \text{ Mo/s} = 10 \cdot 10^6 \text{ } \emptyset/\text{s}$.

Taille du fichier : 3000 caractères = 3000 \emptyset .

$$\left. \begin{array}{l} 1 \text{ seconde} \rightarrow 10 \cdot 10^6 \emptyset \\ x \text{ seconde} \rightarrow 3000 \emptyset \end{array} \right\} \rightarrow x \equiv D = 3000 / 10 \cdot 10^6 = 0,33 \text{ ms.}$$

Temps de transfert du fichier = A + B + C + D = 20 + 5 + 124,8 + 0,33 = 150,13 ms.

Remarques :

-1-Le temps de transfert à travers le canal (mémoire tampon vers ram) est négligeable par rapport au temps de transfert global du fichier.

-2-Le temps de recherche de la bonne piste (le bon cylindre) représente à peu près 25% du temps global. Si on utilise plusieurs cylindres pour stocker le fichier, cette durée sera multipliée par le nombre de cylindres.

-3-Dans le temps moyen de lecture d'un secteur, la durée de lecture proprement dite est très inférieure au temps de recherche du bon secteur.

En conclusion, sur une unité de disque dur, ce sont les phases de recherche de la piste et du secteur (et donc les phases mécaniques) qui sont les plus coûteuses en temps.

4° Attention! Ce n'est pas parce que la taille du fichier a doublé que le temps de transfert va doubler. En effet les termes A et B restent inchangés, seuls les termes C et D vont doubler.

$$C = 124,8 \times 2 = 249,6 \text{ ms}$$

$$D = 0,33 \times 2 = 0,66 \text{ ms}$$

Temps de transfert du fichier = A + B + C + D = 20+5+249,6+0,66 = 275,26ms

5° Si le fichier est stocké sur 3 cylindres, les deux termes qui vont changer sont A et B. A sera multiplié par 3. De plus le temps de dépôt de la tête de lecture sur la piste sera multiplié par 5. En effet à chaque déplacement d'un cylindre à un autre on est obligé de relever la tête, puis une fois sur le bon cylindre déposer la tête. On aura au total 3 dépôts des têtes et 2 soulèvements des têtes. $A = 20 \times 3 = 60 \text{ ms}$

$$B = 6 \times 5 = 30 \text{ ms}$$

$$C = 124,8 \text{ ms}$$

$$D = 0,33 \text{ ms}$$

6° La taille de la mémoire tampon n'aura aucune influence sur la durée globale de transfert du fichier. Le seul inconvénient c'est qu'on sera obligé de solliciter le processeur à plusieurs reprises (6 fois), pour transférer à chaque fois 500 octets au lieu de transférer les 3000 octets d'un seul coup.

SOLUTIONS DU TD CHAPITRE 4 LE PROCESSEUR OU CPU

PARTIE 1 : RAPPELS DE COURS

ENONCES

1°Quels sont les principaux éléments de l'unité centrale et du processeur central d'un ordinateur ?

2°Que signifient les sigles CPU, UAL, ALU, UC, PC, PSW, SP ?

3°Quelle est l'interprétation qu'on doit donner aux contenus des registres PC, RI, et ACC ? Conclure quant à leur taille.

4°Comment peut-on amener l'ordinateur à exécuter un programme dont la première instruction se trouve à l'emplacement mémoire d'adresse $(01F)_{16}$?

5°Supposons qu'une instruction se trouve à l'adresse n de la mémoire. Que se passe-t-il si après avoir exécuté cette instruction, le compteur de programme prend toujours la valeur n ?

SOLUTIONS

EXERCICE 1

1°L'unité centrale se compose du processeur central et de la mémoire centrale.

2°Le processeur central se compose essentiellement de l'unité arithmétique et logique, l'unité de commande, et les registres.

EXERCICE 2

-**CPU** : Central Processing Unit. C'est l'unité Centrale de traitement, appelée encore processeur central ou tout simplement processeur.

-**UAL** : Unité Arithmétique et Logique, est la traduction française du sigle anglais ALU.

-**ALU** : Arithmetic and Logic Unit.

-**UC** : Unité de commande. Pour éviter la confusion avec l'abréviation de l'unité centrale, on utilisera de préférence l'abréviation anglaise CU: Command Unit.

-**PSW** : Processor Status Word. C'est le mot d'état du processeur. Cela désigne le registre des indicateurs ou registre d'état.

-**PC** : Program Counter. C'est le compteur de programme ou compteur ordinal.

-**SP** : Stack Pointer. Pointeur de pile.

Les trois dernières abréviations désignent des registres.

EXERCICE 3

1°Contenus de PC, RI, et ACC

PC : Il contient l'adresse de la prochaine instruction à exécuter. Son contenu correspond à l'adresse d'un emplacement mémoire.

RI : Il contient le code binaire d'une instruction. Ce code comprend une partie code opération de l'instruction, et une partie adresse.

ACC : Il contient généralement le code (binaire) d'une valeur numérique (que cette dernière soit le résultat d'une opération arithmétique ou logique, ou une adresse).

Par conséquent les trois registres contiennent des données différentes sur le plan sémantique.

2°Tailles des registres

PC : son contenu est une adresse. Donc sa taille est en général égale au nombre de bits du bus d'adresses (rien n'empêche que sa taille soit supérieure).

RI : son contenu est le code d'une instruction. Comme chaque instruction occupe un ou plusieurs emplacements mémoire, par conséquent sa taille est égale à celle d'un ou de plusieurs mots mémoire.

ACC : son contenu est une valeur numérique. Comme on peut effectuer des opérations de transfert entre l'accumulateur et la mémoire, donc sa taille est en général égale à celle d'un mot mémoire.

EXERCICE 4

Comme le compteur de programme donne l'adresse de la prochaine instruction à exécuter, pour exécuter notre programme dont la première instruction est stockée à l'adresse (01F), il suffit donc de mettre dans le PC (compteur ordinal) la valeur (01F).

EXERCICE 5

Compte tenu du rôle du PC qui donne toujours l'adresse de la prochaine instruction en séquence, le processeur exécutera constamment et sans s'arrêter cette même instruction qui se trouve à l'emplacement mémoire d'adresse n. On dit qu'on entre alors dans une boucle infinie (le programme ne s'arrête pas).

PARTIE 2 : EXERCICES

EXERCICE 6 : adresses symboliques et langage assembleur de mimosa

(Mémoire organisée en mots de 16 bits)

Enoncé

Substituer à toutes les adresses symboliques (adresses d'instruction et d'opérande) de l'exemple, des adresses numériques hexadécimales. Montrer en même temps comment il faut placer le programme en mémoire, si on suppose que la première adresse à utiliser est (100)h, et que les variables a et b seront stockées respectivement aux adresses (110)h et (111)h.

LIGNE	LABEL	MNEMONIQUE	ADRESSE	SIGNIFICATION
1	aprêt :	SAUT,E	lirea	Si E vrai, aller à lirea
2		SAUT	aprêt	Aller à aprêt
3	lirea :	ENTREE		Mettre ds ACC contenu du tampon d'entrée
4		STOCKER	a	Mettre (ACC) à l'adresse mémoire de a
5	bprêt :	SAUT,E	lireb	Si E vrai, aller à lireb
6		SAUT	bprêt	
7	lireb :	ENTREE		
8		STOCKER	b	m(b) := (ACC)
9		ADD	a	ACC := (ACC) + (m(a))
10	Res_prêt :	SAUT,S	Res_prêt	Si S vrai, aller à Res_prêt
11		SORTIE		Mettre (ACC) dans Tampon de sortie
12		HALTE		

Solution : Correspondance adresse logique et adresse physique

On suppose que la mémoire est organisée en mots de 16bits. Donc chaque instruction occupera un emplacement mémoire.

ADRESSE MEMOIRE	MNEMONIQUE	OPERANDE (valeur du champ adresse)
100	SAUT, E	102
101	SAUT	100
102	ENTREE	
103	STOCKER	110
104	SAUT, E	106
105	SAUT	104
106	ENTREE	
107	STOCKER	111
108	ADD	110
109	SAUT, S	109
10A	SORTIE	
10B	HALTE	
110	Cet emplacement contiendra la valeur de la variable "a" après l'exécution de l'instruction « stocker 110 »	
111	Cet emplacement contiendra la valeur de la variable "b" après l'exécution de l'instruction « stocker 111 »	

EXERCICE 7 : langage assembleur, assemblage de programme et stockage en mémoire*(Mémoire organisée en mots de 16 bits)***Enoncé**

Réécrire le programme précédent en remplaçant chaque mnémonique du langage assembleur de mimosa par son code. Donner pour chaque adresse mémoire son contenu en hexadécimal puis en binaire.

Solution

L'assemblage consiste à remplacer chaque mnémonique du programme précédent par son code opération, donc chaque instruction sera traduite en langage machine (binaire), puis stockée en mémoire. Les instructions du programme seront stockées séquentiellement à partir de l'adresse 100.

On rappelle que mimosa traite la partie haute de l'instruction en premier (stockage mémoire en big endian). On supposera comme précédemment que la mémoire est organisée en mots de 16 bits.

-7-a-ASSEMBLAGE			-7-b-STOCKAGE EN MEMOIRE		
			<i>Remarque</i> : le tiret (« - ») signifie valeur non définie, sa valeur est indifférente.		
Adresse Mémoire	Code Opération	Adresse ou Opérande	Adresse Mémoire	Contenu des emplacements mémoire en Hexadécimal	Contenu des emplacements mémoire en Binaire
100	D	102	100	D102	1101 0001 0000 0010
101	8	100	101	8100	1000 0001 0000 0000
102	F2		102	F2--	1111 0010 -----
103	4	110	103	4110	0100 0001 0001 0000
104	D	106	104	D106	1101 0001 0000 0110
105	8	104	105	8104	1000 0001 0000 0100
106	F2		106	F2--	1111 0010 -----
107	4	111	107	4111	0100 0001 0001 0001
108	6	110	108	6110	0110 0001 0001 0000
109	E	109	109	E109	1110 0001 0000 1001
10A	F3		10A	F3--	1111 0011 -----
10B	F5		10B	F5--	1111 0101 -----

EXERCICE 8: Désassemblage**Enoncé**

L'ordinateur mimosa possède en mémoire le contenu indiqué par le tableau.

Adresse mémoire (hexa)	Contenu
100	0001 0000 0001 1001
101	0100 0001 0000 0101
102	0001 0000 0010 0000
103	0110 0001 0000 0101
104	1111 0101 indifférent

1° Réécrire le même contenu en hexadécimal.

2° Réécrire le programme en code mnémonique puis donner sa signification.

Solution: c'est le programme d'addition de $(25)_{10}$ et $(32)_{10}$

Adresse mémoire	Contenu Hexa	Code Mnémonique	valeur opérande	Signification
100	1019	NOMBRE	019	$ACC := (19)_{16}$ (mettre dans ACC la valeur $(19)_{16}$ càd $(25)_{10}$)
101	4105	STOCKER	105	$m(105) := (ACC)$ (mettre dans @ mémoire $(105)_{16}$ le contenu de ACC)
102	1020	NOMBRE	020	$ACC := (020)_{16}$ (mettre dans Acc la valeur $(020)_{16}$ càd $(32)_{10}$)
103	6105	ADD	105	$ACC := ACC + m(105)$ (ajouter au contenu de ACC celui de l'@ 105
104	F5--	HALTE	--	Arrêt du programme

EXERCICE 9 : programmation en assembleur

Enoncé

On suppose que l'ordinateur mimosa ne possède pas d'instruction de soustraction SUB. Ecrire un programme de calcul de $a - b$. On supposera que la valeur de a se trouve déjà dans l'accumulateur, que b occupe l'emplacement mémoire d'adresse $(10A)_{16}$, et que le résultat du programme (la différence) occupera l'accumulateur. Le programme devra être stocké en mémoire à partir de l'adresse $(100)_{16}$, et les emplacements mémoire d'adresses $(10B)_{16}$ et $(10C)_{16}$ peuvent être utilisés comme espace de travail.

Solution

Adresse mémoire	Code Mnémonique	valeur de ADR	Signification
100	STOCKER	10B	Sauvegarder le contenu de l'accumulateur (la valeur de a)
101	CHARGER	10A	Charger dans l'accumulateur le nombre b
102	NEG		Calculer l'opposé de b
103	STOCKER	10C	Et stocker (-b) à l'adresse $(10C)_{16}$
104	CHARGER	10B	Récupérer la valeur de a dans l'accumulateur
105	ADD	10C	Faire l'addition du contenu de ACC avec le contenu de l'@ $(10C)_{16}$ (càd $a + (-b)$)
106	HALTE		

EXERCICE 10 : fetch et exécution

Enoncé

En supposant que chaque instruction du programme de l'exercice 8 prend un cycle machine, et que chaque cycle est partagé en 2 phases (recherche et exécution), donner dans un tableau le contenu des registres PC, RI, ACC et de l'emplacement mémoire $(105)_{16}$, au cours du déroulement de l'exécution de ce programme.

Solution

Numéro du cycle	Phase de recherche	Phase d'exécution	PC	RI	ACC	m(105)
1	x	x	101	1019	non défini	non défini
			101	1019	$(019)_{16}$	non défini
2	x	x	102	4105	****	non défini
			102	4105	****	$(019)_{16}$
3	x	x	103	1020	****	non défini
			103	1020	$(020)_{16}$	non défini
4	x	x	104	6105	****	non défini
			104	6105	$(039)_{16}$	non défini
5	x	x	105	F5--	****	non défini
			105	F5--	****	non défini

EXERCICE 11 : assembleur + fetch + exécution**Enoncé**

On veut prendre la valeur absolue d'un nombre qui se trouve à l'adresse mémoire $(109)_{16}$. Pour cela on prend le nombre et on teste s'il est négatif. Si oui, on le rend positif et on le range à la même adresse. Si non, on le range tel quel à la même adresse.

1°Ecrire le programme correspondant en langage assembleur de mimosa.

2°Donner le contenu des emplacements mémoire si le programme est stocké à partir de l'adresse $(100)_{16}$ (on supposera que le bus de données, l'accumulateur, et les emplacements mémoire ont tous une capacité de 16 bits).

3° En supposant que le nombre est positif, donner comment le programme est exécuté dans le temps, en précisant les contenus de : Bus de données, Bus d'adresses, PC, RI, ACC, m(109).

4°Même question que 3° en supposant cette fois que le nombre est négatif.

Solution

1°

Adresse mémoire	Code Mnémonique	Valeur ADR	Signification
100	EFFACERN	--	N := 0
101	CHARGER	109	ACC:=m(109)
102	SAUT,N	104	Si N=1 alors PC := 104
103	SAUT	105	PC := 105
104	NEG	--	ACC := - ACC
105	STOCKER	109	m(109):=ACC
106	HALTE	--	

2°

Adresse mémoire	Contenu hexadécimal
100	F9--
101	2109
102	9104
103	8105
104	F1--
105	4109
106	F5--

3° Cas où le nombre est positif

N°cycle	Phase recherche	Phase exécution	PC	Bus @	RI	Bus data	ACC	m(109)	Observations
1	x x x	x	100	---	----	----	----	nbre a	@début→PC
			100	100	----	----	----	'''	(PC)→bus@
			101		F9--	F9--	----	'''	M(100)→RI et PC=PC+1 Et décodage F9
			101		F9--	F9--	----	'''	Bit N = 0
2	x x	x	101	101	F9--		----	'''	(PC)→bus@
			102	101	2109	2109	----	'''	m(100)→ACC et PC=PC+1 et décodage
			102	109	2109	2109	----	'''	Partie @de RI →Bus @
			102	109	2109	nbre a	nbre a	'''	m(109)→ACC
3	x x	x	102	102	2109		'''	'''	(PC)→bus@
			103		9104	9104	'''	'''	m(102)→RI et décodage
			103		9104		'''	'''	N=0 (→instruction suivante ne rien faire)
4	x x	x	103	103			'''	'''	(PC)→bus@
			104		8105	8105	'''	'''	m(103)→RI et décodage
			105				'''	'''	Saut à l'@105
5	x x	x	105	105			'''	'''	(PC)→bus@
			106	105	4109	4109	'''	'''	m(105)→RI et décodage
			106	109	4109		'''	'''	Partie basse de RI →Bus @
			106	109		nbre a	'''	'''	m(109)→ACC
6	x x	x	106	106			'''	'''	(PC)→bus@
			107	106	F5--	F5--	'''	'''	m(106)→RI et décodage
			107	106	F5--	F5--	'''	'''	Arrêt

4° Cas où le nombre est négatif

N°cycle	Phase recherche	Phase exécution	PC	Bus @	RI	Bus data	ACC	m(109)
1	x x x	x	100	---	----	----	----	nbre a
			100	100	----	----	----	'''
			101		F9--	F9--	----	'''
			101		F9--	F9--	----	'''
2	x x	x	101	101	F9--		----	'''
			102	101	2109	2109	----	'''
			102	109	2109	2109	----	'''
			102	109	2109	nbre a	nbre a	'''
3	x x	x	102	102	2109		'''	'''
			103		9104	9104	'''	'''
			104		9104		'''	'''

4	x x	x	104	104	9104		''''	'''
			105		F1--	F1--	''''	'''
			105		F1--	F1--	- (nbre a)	'''
5	x x	x x	105	105	F1--		''''	'''
			106	105	4109	4109	''''	'''
			106	109	4109		''''	'''
			106	109	4109	- (nbre a)	''''	- (nbre a)
6	X x	x	106	106	4109		''''	''''
			107	106	F5--	F5--	''''	''''
			107	106	F5--	F5--	''''	''''

EXERCICE 12 : pile et sous programme

Enoncé

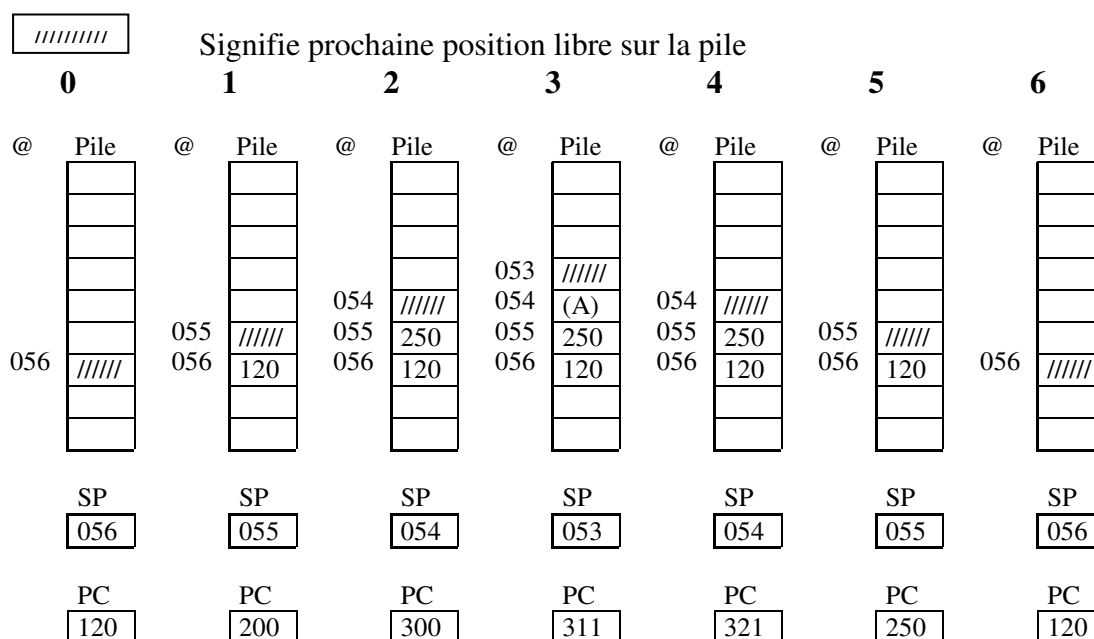
On donne le programme suivant pour mimosa où chaque adresse est exprimée sur 12 bits, chaque instruction occupe un emplacement mémoire, et un emplacement mémoire vaut 16 bits,.

Donner pour chacune des instructions 1 à 6, le contenu de la pile, de SP et de PC. Ces contenus correspondent à chaque fois à l'état juste après l'exécution de l'instruction.

L'état 0 correspond à l'état après la phase recherche de l'instruction SAUTSP 200, et juste avant la phase exécution de cette instruction. Dans cet état on a donc :

(PC) = 120 et (SP) = 056. La pile occupe les adresses de 000 à 0FF.

Solution



Détaillons les phases 2 et 5, c'est à dire avant et après exécution des instructions correspondantes.

Instruction 2 : SAUTSP 300

-1°-Contenu de PC = 24F -2°-Dépôt du contenu de PC sur le bus d'@ -3°-Lecture mémoire (le contenu de l'@24F c'ad le code de l'instruction SAUTSP 300 est mis dans le registre RI) et incrémentation du PC ($PC := PC + 1 = 250$) -4°-Décodage du contenu de RI c'ad interprétation de SAUTSP 300 -5°-Execution de l'instruction c'ad :

- a°)écriture du contenu de PC dans la pile pour sauvegarder l'@ de retour → écriture de 250 sur le haut de la pile ;
- b°)décrémentation de SP → $SP := SP - 1 = 055 - 1 = 054$;
- c°)dépôt du contenu de la partie adresse du registre RI dans le PC (@300 qui est l'adresse du sous programme) : $(PC) := 300$.

Instruction 5 : RETSP

Phase recherche :

- 1°-(PC)=350 -2°- Bus @ := (PC) = 350 -3°-Lecture mémoire et en parallèle incrémentation du PC: $(PC) = (PC) + 1 = 351$ -4°- $m(350)$ = code de RETSP est mis dans RI -5°- Décodage de l'instruction RETSP

Phase Exécution:

- 6°-Lecture de l'adresse de retour à partir de la pile → mise du haut de la pile dans le PC → $PC = 250$ -7°-La lecture de la pile → incrémentation du pointeur de pile : $(SP) = (SP) + 1 = 054 + 1 = 055$.

EXERCICE 13 : indicateurs Carry, Signe (N), Zero

Enoncé

Il s'agit de tester un bit du registre d'état d'un organe d'E/S. Si le bit est à 1, le périphérique est prêt et on peut faire le transfert d'E/S. S'il est égal à zéro, le périphérique est occupé et il faut attendre jusqu'à ce qu'il soit prêt. Donner pour une lecture sur le périphérique, le programme en langage assembleur dans les trois cas suivants :

1°cas : l'état **prêt** est indiqué par le bit 2^0 du registre d'état

Après lecture du registre d'état, il est pratique d'envoyer le bit d'état dans le carry par une instruction de décalage, et de tester le **bit carry** (la lecture du registre d'état correspond à un transfert de son contenu dans l'accumulateur ACC).

2°cas : l'état **prêt** est indiqué par le bit 2⁷ du registre d'état

Après lecture du registre d'état (transfert dans ACC), il suffit de tester *l'indicateur de signe* (en supposant que la lecture du registre positionne le bit N), qui sera à 1 si le bit numéro 7 du nombre lu est à 1.

3°cas : l'état **prêt** est indiqué par le bit 2³ du registre d'état

Le moyen le plus simple de tester l'état **prêt** est de faire un masquage, c'est à dire de mettre à zéro tous les bits sauf le **bit prêt**, de sorte que *l'indicateur Z* sera positionné (Z=1) si **prêt** = 0, et Z sera nul dans le cas contraire. Le masquage se fera donc par un ET logique avec la valeur binaire 00001000 (08 en hexa). On utilisera l'adresse 200 comme adresse de travail et de stockage temporaire, et l'adresse 5FF comme adresse du registre d'état PSW .

Solution1°cas : état prêt = bit 2⁰ du registre d'état

Adresse	Mnémonique Opérnde	Signification
100	CHARGER 5FF	Mettre le contenu du registre d'état dans ACC → lecture du mot d'état
101	DECD	Le bit 0 du mot d'état est mis dans le carry par un décalage à droite
102	SAUT,C 104	Si C=1 → périphérique prêt → aller à l'@104 pour l'opération d'E (lecture)
103	SAUT 100	Sinon C=0 → périphérique non prêt → retour à l'@100 pour relire le mot d'état
104	ENTREE	Lecture de la donnée sur le périphérique d'entrée → ACC := (tampon d'entrée)
105	HALTE	Arrêt du programme

2°cas : état prêt = bit 2⁷ du registre d'état

Adresse	Mnémonique Opérnde	Signification
100	EFFACERN	Mettre à zéro le bit de signe (N) du registre des indicateurs (PSW)
101	CHARGER 5FF	Mettre le contenu du registre d'état dans ACC → le bit 2 ⁷ du mot d'état correspond au bit de signe → si bit 2 ⁷ = 1 → N= 1 ; sinon N reste à zéro
102	SAUT,N 104	Si N=1 → périphérique prêt → aller à l'@104 pour l'opération d'E (lecture)
103	SAUT 101	Sinon N=0 → périphérique non prêt → retour à l'@101 pour lire le mot d'état
104	ENTREE	Lecture de la donnée sur le périphérique d'entrée → ACC := (tampon d'entrée)
105	HALTE	Arrêt du programme

3°cas : état prêt = bit 2³ du registre d'état

Adresse	Mnémonique	Adresse	Signification
100	NOMBRE	08	Mettre le nombre 8 (↔ valeur binaire 00001000) dans ACC
101	STOCKER	200	m(200) := (ACC) (le nombre 8 est stocké à l'@200)
102	CHARGER	5FF	Lecture du registre d'état (↔ mettre mot d'état dans ACC)
103	ET	200	ET logique du contenu de ACC avec contenu de l'adresse 200, et le résultat se trouve dans l'accumulateur. Compte tenu de la valeur binaire contenue dans l'@200, cela revient à mettre à zéro tous les bits du mot d'état sauf le bit 2 ³
104	SAUT,Z	102	Si Z=1 → le résultat du ET logique est nul → périphérique non prêt → retour à l'@102 pour relire le mot d'état
105	ENTREE		Sinon Z=0 → Lecture de la donnée sur le périphérique d'entrée → ACC := (tampon d'entrée)
106	HALTE		Arrêt du programme

EXERCICE 14 : recherche du minimum dans une table**Enoncé**

Dans une table constituée de nombres logiques, on cherche à déterminer l'élément qui a la plus petite valeur. Ecrire l'algorithme puis le programme en langage assembleur de mimosa.

On suppose que la table commence à l'adresse 300 et que sa longueur est égale à 256. La valeur min devra être stockée en fin de programme à l'adresse 400. Le programme est stocké en mémoire à l'adresse 100. On utilisera comme zones de travail 3 registres correspondant chacun à une adresse :

- R0 : contient la valeur min provisoire
- R1 : contient l'adresse du i^{ème} élément
- R2 : sert de compteur

Solution**Algorithme**

```

i := 1 ;
compteur := n ;
min := element(i) ;
Boucle
    compteur :=compteur-1 ;
    Si compteur = 0 Alors
        sortie de boucle ;
    FinSi ;
    i := i+1;
    Si element (i) < min Alors
        Min := element(i) ;
    FinSi ;
FinBoucle ;
FIN.

```

Programme

Etiquette	Adresse mémoire	Mnémonique	Opé- rande	Signification
	100	NOMBRE	00	ACC := (00) ₁₆
	101	STOCKER	R0	(R0) := (ACC) ↔ (R0) := 0 ↔ valeur min = 00
	102	NOMBRE	300	ACC := @ de début de la table
	103	STOCKER	R1	R1 := @ de début de la table
	104	NOMBRE	256	ACC := 256 (longueur de la table)
	105	STOCKER	R2	R2 := ACC
	106	CHARGER	R1	ACC := le 1 ^{er} élément de la table
	107	STOCKER	R0	R0 := le 1 ^{er} élément de la table
boucle :	108	EFFACERZ		Mettre à zéro le bit Z
	109	DEC	R2	R2=R2-1 (décrémenter le nbre d'éléments lus)
	10A	SAUT,Z	sortie	Si Z=1 → on a lu tous les éléments de la table → sortir de boucle
	10B	INC	R1	R1=R1+1 (dans R1 l'@ du prochain élément de la table)
	10C	EFFACERC		Remettre à zéro le bit carry C

Etiquette	Adresse mémoire	Mnémonique	Adresse	Signification
	10D	SUB	R1	$ACC := ACC - (R1) \leftrightarrow ACC := \min - (R1)$
	10E	ZAUT,C	boucle	Si $C=1 \leftrightarrow \min < (R1) \rightarrow$ on garde le min
	10F	CHARGER	R1	Sinon $C=0 \leftrightarrow \min > (R1) \rightarrow$ le nouveau min est celui dont l'adresse est donnée par R1 \rightarrow mettre le nouveau min dans ACC $ACC := (R1)$
	110	STOCKER	R0	Stockage du nouveau min dans R0
	111	SAUT	boucle	
sortie :	112	CHARGER	R0	$ACC := (R0)$ (charger le min dans ACC)
	113	STOCKER	400	$m(400) := ACC$ (stockage du min à l'adresse 400)
	114	HALTE		Arrêt du programme

PARTIE 3 : CONTROLE DES CONNAISSANCES (durée 1h30)**EXERCICE 3-1 : Adressage relatif sur Mimosa****Enoncé**

Si la mémoire de mimosa est **organisée en octets**, considérons l'instruction de branchement conditionnel en adressage relatif « SAUT,S donnée », où donnée exprime le déplacement D. Cette instruction occupe les emplacements mémoire d'adresses 310 et 311.

1° Si D = 00A, quelle est la valeur de l'adresse effective EA ?

2° Donner la valeur de D si : a°) on veut se brancher à l'adresse 313 ; b°) on veut se brancher à l'adresse 30F.

Solution

Question 1 : adresse effective EA = (PC)+i+D=310+2+0A=31C

(i=2 car l'instruction occupe deux emplacements mémoire)

Question 2 : a°) Déplacement D = EA – [(PC) +2]=313-312=01

b°) Déplacement D = 30F –312 = -3. Le déplacement est négatif, il faut donc l'exprimer en complément à 2. Compl2 (-3) = FD. Donc Déplacement D = FD

EXERCICE 3-2 : Pile et sous programme (mémoire organisée en octets)**Enoncé**

Considérons le programme suivant pour Mimosa. Donner le contenu de la PILE, de SP, de PC et RI, à la fin de la phase recherche et à la fin de la phase exécution de chaque instruction. L'état initial correspond à (PC) = 200 ; (SP) = 050 ; (ACC) =2FB3.

PROGRAMME PRINCIPAL

Numéro Instructio	Adresse Mémoire	Mnémonique	Adresse
	200	-----	---
	---	-----	---
1	220	PUSHA	
	---	-----	---
2	22F	SAUTSP	400
	---	-----	---
8	235	PULA	
	---	-----	---
9	250	HALTE	

SOUS PROGRAMME

Numéro instructio	Adresse mémoire	Mnémonique	Adresse
	400	-----	---
	---	-----	---
3	420	NOMBRE	435
4	422	PSHA	
	---	---	---
	---	-----	---
5	44F	NOP	---
6	450	PULA	
	---	-----	---
	---	-----	---
7	500	RETSP	

Solution

(Par souci de clarté et pour économiser l'espace, la structure de la RAM est basculée à droite par une rotation de 90° à droite)

<u>REGISTRES</u>				<u>CONTENU PILE</u> (-- signifie l'adresse est vide)							
N°instruction et phase	Contenu PC	Contenu RI	Contenu SP	Adr 50	Adr 4F	Adr 4E	Adr 4D	Adr 4C	Adr 4B	Adr 4A	Adr 49
<i>Etat Initial</i>	200		50	--	--	--	--	--	--	--	--
1 recherche	221	F7--	50	--	--	--	--	--	--	--	--
1exécution	221		4E	2F	B3	--	--	--	--	--	--
2 recherche	231	0400	4E	2F	B3	--	--	--	--	--	--
2exécution	400		4C	2F	B3	23	1 -	--	--	--	--
3 recherche	422	1435	4C	2F	B3	23	1 -	--	--	--	--
3exécution	422		4C	2F	B3	23	1 -	--	--	--	--
4 recherche	423	F7--	4C	2F	B3	23	1 -	--	--	--	--
4exécution	423		4A	2F	B3	23	1 -	43	5 -	--	--
5 recherche	450	F4--	4A	2F	B3	23	1 -	43	5 -	--	--
5exécution	450		4A	2F	B3	23	1 -	43	5 -	--	--
6 recherche	451	FF--	4A	2F	B3	23	1 -	43	5 -	--	--
6exécution	451		4C	2F	B3	23	1 -	--	--	--	--
7 recherche	501	F6--	4C	2F	B3	23	1 -	--	--	--	--
7exécution	231		4E	2F	B3	--	--	--	--	--	--
8 recherche	236	FF--	4E	2F	B3	--	--	--	--	--	--
8exécution	236		50	--	--	--	--	--	--	--	--
9 recherche	251	F5--	50	--	--	--	--	--	--	--	--
9exécution	251		50	--	--	--	--	--	--	--	--

EXERCICE 3-3 : Désassemblage et exécution de programme.**Enoncé**

Le contenu de la mémoire de MIMOSA (**organisée en octets**) est donné par le tableau suivant :

Adresse mémoire	Contenu	Adresse mémoire	contenu
300	21	307	72
301	F2	308	F5
302	61	309	A3
303	F3	30A	00
304	41	30B	42
305	F2	30C	F4
306	FD	30D	F5

1°Désassembler le programme qui se trouve en mémoire.

2°Donner après l'exécution du programme, le contenu des emplacements mémoire d'adresses : 1F2, 1F3, 2F4, 2F5. Avant l'exécution, ces emplacements contenaient respectivement les valeurs hexadécimales suivantes : 07, 05, 0E, 04 .

3°Quelle est l'instruction qu'on a oubliée dans ce programme et pourquoi ?

Solution

Numéro Ligne	Mnémonique	Opérande
1	CHARGER	1F2
2	ADD	1F3
3	STOCKER	1F2
4	DECD	
5	SUB	2F5
6	SAUT,Z	300
7	STOCKER	2F4
8	HALTE	

CONTENU DES ADRESSES APRES EXECUTION DU PROGRAMME			
Adr 1F2	Adr 1F3	Adr 2F4	Adr 2F5
12	5	2	4

InstructionManquante :EFFACERZ

Position de l'instruction manquante : avant l'instruction de soustraction SUB

Explications : quand le résultat d'une opération n'est pas nul, ➔ le bit Z n'est pas modifié. S'il était à 1, il reste à 1 ➔ risque d'erreur ➔ avant l'opération arithmétique il faut le mettre à zéro par l'instruction EFFACERZ.

EXERCICE 3-4 : Programmation en assembleur et assemblage**Enoncé**

On désire écrire un programme qui permet de lire deux nombres a et b sur un périphérique d'entrée (clavier par exemple), de les stocker aux adresses respectives 300 et 301, puis de faire leur comparaison, et en fonction du résultat effectuer certains traitements :

- si a = b, stocker le nombre « a » (ou « b ») à l'adresse 400, puis envoyer ce nombre sur le périphérique de sortie (imprimante par exemple) et s'arrêter;
- si b < a, calculer la moyenne (a+b)/2, puis stocker le résultat à l'adresse 400, puis envoyer le résultat sur le périphérique de sortie et s'arrêter;
- si b > a, retourner obligatoirement saisir deux nouveaux nombres a et b.

1°Ecrire le programme correspondant en langage assembleur de Mimosa. Ce programme sera stocké en mémoire à partir de l'adresse 200, et la mémoire est organisée en mots de 16 bits.

2°Donner le contenu des emplacements mémoire en hexadécimal.

Solution

Adresse mémoire	Mnémonique	Opérande
200	ENTREE	
201	STOCKER	300
202	ENTREE	
203	STOCKER	301
204	EFFACERZ	
205	EFFACERC	
206	SUB	300
207	SAUT, Z	20A

Adresse mémoire	Contenu hexadécimal
200	F2--
201	4300
202	F2--
203	4301
204	FA--
205	FC--
206	7300
207	A20A

Adresse mémoire	Mnémonique	Opérande		Adresse mémoire	Contenu hexadécimal
208	SAUT, C	20C		208	C20C
209	SAUT	200		209	8200
20A	CHARGER	300		20A	2300
20B	SAUT	20F		20B	820F
20C	CHARGER	300		20C	2300
20D	ADD	301		20D	6301
20E	DECD			20E	FD--
20F	STOCKER	400		20F	4400
210	SORTIE			210	F3--
211	HALTE			211	F5--