NOTES de COURS : Pré-requis

Chapitre 1 : Organisation générale d'un calculateur numérique

Un système informatique moderne est composé de quatre parties complémentaires dont les trois premières sont indispensables pour que le calculateur rende le moindre service. Ce sont :

1) Le matériel dont le cœur comprend le processeur et la mémoire centrale,
2) Le logiciel composé de programmes destinés à être exécutés par le matériel,
3) Le Système d'Exploitation, ensemble de programmes permettant d'exploiter le calculateur,
4) Les progiciels, ensemble de logiciels permettant de mettre en machine une application sans avoir à écrire de programmes.

1.1 Le matériel

Le matériel d'un ordinateur est composé d'une Unité Centrale (UC) contenant le processeur chargé d'exécuter les instructions présentes en mémoire centrale, de plusieurs dispositifs d'entrée-sortie capables d'établir des échanges avec le monde extérieur (clavier-écran, imprimantes ...) et de mémoires auxiliaires (disques et bandes) permettant de ranger de façon permanente les programmes et les données que le calculateur devra traiter. La figure 1.1 montre ces composants principaux.

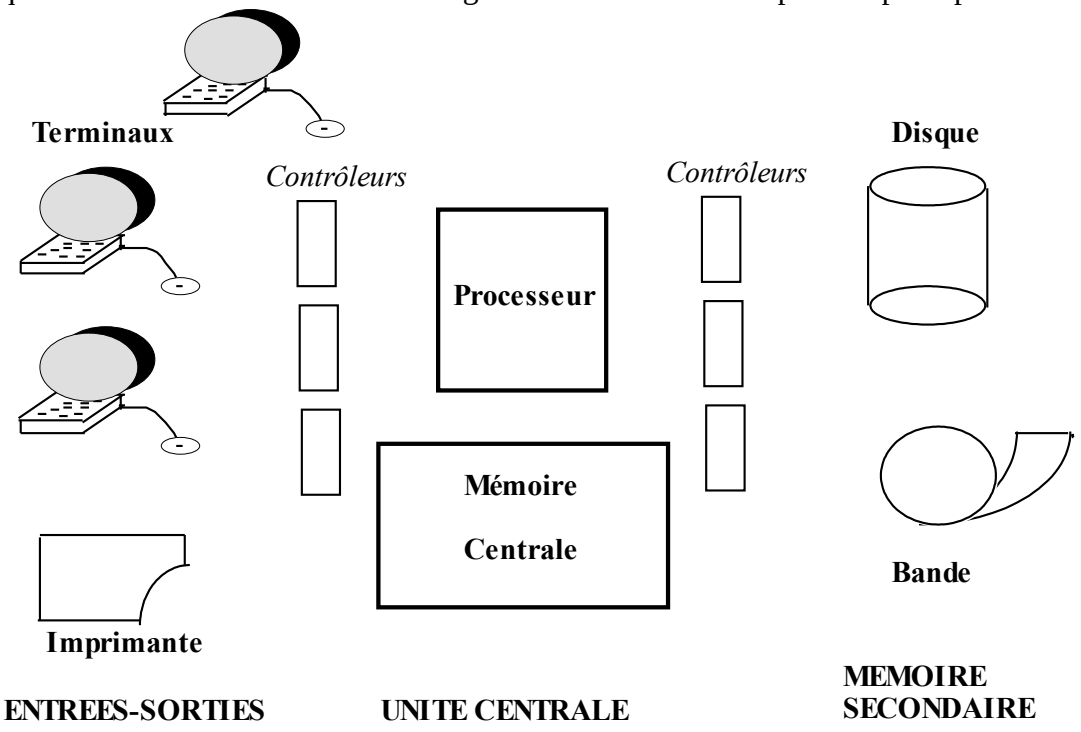

Figure 1.1 Structure d'un calculateur numérique

Un programme exécutable est une suite d'instructions que le processeur doit interpréter les unes après les autres ; ce programme, stocké sur le disque, est transféré en mémoire centrale pour être exécuté. Le processeur contient un Compteur Programme (CP) qui repère le numéro de l'instruction en cours d'exécution ; au lancement du programme ce compteur à la valeur 0 puis sa valeur augmente d'une unité à chaque instruction exécutée.

Parmi ces instructions, certaines font appel à d'autres suites d'opérations dont le rôle est de gérer les échanges entre un dispositif physique extérieur (disque, terminal ...) et le processeur central, à travers un processeur spécialisé appelé contrôleur (CTRL).

Ex: Soit le programme suivant :

0- Lire au clavier la valeur donnée à une variable x
1- Lire au clavier la valeur donnée à une variable y
2- Exécuter l'opération $z = x + y$
3- Ecrire sur l'écran la valeur de z

L'instruction 0 activera une suite d'opérations exécutées par le contrôleur du clavier qui viendra lire les chiffres tapés par l'utilisateur pour composer un nombre qui sera transmis au processeur central puis affecté à la variable x, ce qui terminera l'exécution de la première instruction. Le compteur programme sera ensuite incrémenté d'une unité pour aller chercher l'instruction suivante.

L'instruction 1 s'exécutera de façon identique à l'instruction 0, le nombre lu étant affecté à la variable y.

Le processeur possède alors les valeurs de x et de y.

L'instruction 2 réalise la somme de ces 2 valeurs rangées dans l'unité centrale, soit dans le processeur lui-même (on parle alors de registre), soit en mémoire centrale, dans une zone de stockage nommée z.

L'instruction désignée par la valeur 3 du compteur programme consistera à lire la zone nommée z puis à transmettre au contrôleur la valeur contenue dans cette zone ; celui-ci compose la suite de chiffres correspondant à ce nombre pour l'écrire sur l'écran.

L'unité centrale est le cœur de la machine informatique : elle est formée de circuits logiques généralement reliés entre eux par des "bus" sur lesquels circule l'information échangée entre les cartes. (carte mère) (Figure 1.2)

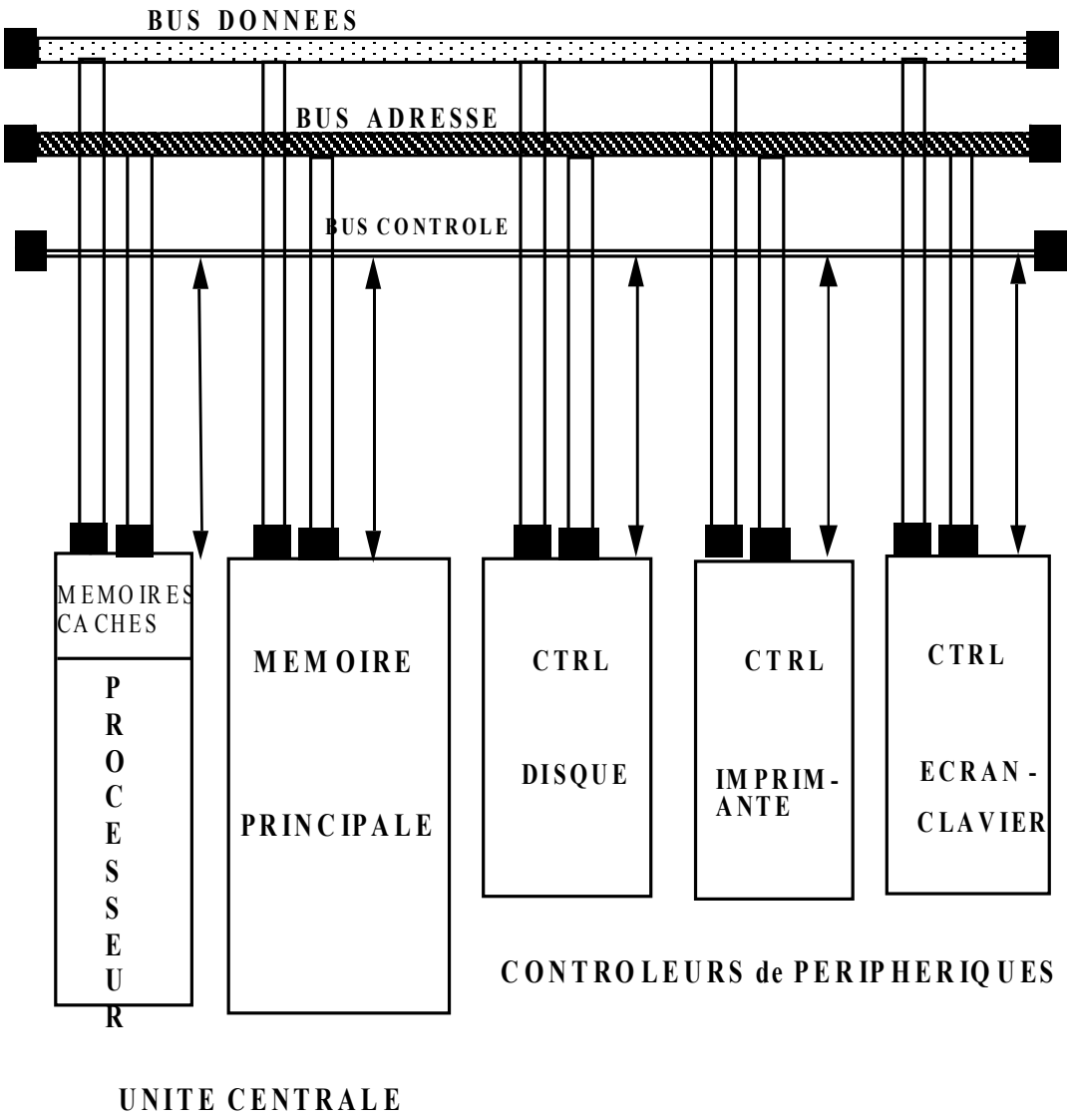

Figure 1.2 Composants fonctionnels d'une unité centrale

1.1.1 Le processeur

Les processeurs sont aujourd'hui des circuits intégrés pouvant contenir plusieurs millions de transistors. La performance du processeur s'exprime alors en millions d'instructions par seconde (MIPS).

Le calcul de performance s'établit à partir de la formule suivante :

$$T = N * C * Tc$$

T = Temps d'exécution total d'une suite d'instructions

N = Nombre d'instructions exécutées

C = Nombre de cycles d'horloge nécessaires en moyenne pour exécuter une instruction

Tc = Temps de cycle de l'horloge de base de la machine.

Par exemple un processeur dont la fréquence d'horloge est de 3 Ghz capable d'exécuter 2,2 instructions par cycle en moyenne aura exécuté 6,6 milliards d'instructions en 1 seconde d'unité centrale. (N = T / N*Tc)

(La nanoseconde (ns) soit 10^{-9} s représente le temps séparant deux impulsions d'une horloge à 1 Ghz).

Par le passé, les constructeurs affichaient la performance des processeurs en Millions d'Instructions par Secondes (instructions entières ou flottantes). Vu la complexité des processeurs actuels et notamment leur capacité d'exécution simultanée de plusieurs instructions, ces mesures ne sont plus significatives.

L'étude de performance s'établit actuellement par comparaison à partir de programmes de tests (appelés benchmarks) significatifs des types d'applications que le processeur exécute. Pour le calcul, les références sont des benchmarks sur des entiers (SPECint) et sur des flottants (SPECfloat : il s'agit d'opérations arithmétiques sur des nombres représentés en flottant (voir chapitre 2 paragraphe 2.1).

Le tableau fournissant le SPECint du processeur Itanium (Architecture IA64 de HP et Intel mise sur le marché en Mai 2001) se lit ainsi :
Le SPECint2000 est de 314 et le SPECint_base2000 est aussi de 314. Ceci signifie que les performances crêtes obtenues sont identiques aux performances nominales. Ceci s'explique par la jeunesse du processeur en juin 2001 qui ne disposait pas encore des outils logiciels pour optimiser le code produit à partir du programme test.
La première colonne donne le nom du programme test. La seconde (Reference Time) est le temps de référence actuellement obtenu sur une architecture classique. La troisième (Base runtime) est le temps obtenu en utilisant une version de base du calculateur sans chercher à améliorer les outils de base, la troisième (Base Ratio) est le rapport (*100) entre le temps de référence et le temps de base ; ces valeurs sont aussi produites en essayant d'optimiser l'exécution du test (colonnes Runtime et Ratio).
La fin de la fiche décrit le matériel, le système d'exploitation et les options de compilation utilisées pour produire le code qui a engendré ce résultat.

En regardant précisément la fiche du processeur Alpha21264 on remarque que les deux valeurs sont différentes : Le SPECint2000 est de 533 et le SPECint_base2000 est de 511.
Ceci signifie qu'il est possible d'améliorer le rendement de la machine en utilisant des options de compilation (et peut-être des options du système d'exploitation). Remarquez par exemple que le programme 253permblk est deux fois plus rapide dans la version optimisée que la version nominale). C'est l'expérience acquise sur ce type de processeur et ce type de système qui permet d'engendrer un code plus efficace.

Les processeurs qui équipent les unités centrales des calculateurs actuels tendent à se standardiser alors que quelques années auparavant chaque constructeur développait une ou plusieurs unités centrales spécifiques. Le processeur Intel Pentium ® définit une gamme de processeurs, actuellement commercialisés dans les

calculateurs PC (Pentium, PentiumIII, Pentium Pro, Pentium IV), évolutions successives des architectures 80x86. Le dual core est le dernier de la gamme sorti des usines d'Intel qui exploite le parallélisme de flots pour améliorer encore les performances ; une nouvelle architecture en rupture avec l'architecture des Pentium équipe certains calculateurs notamment chez Hewlett Packard et Intel. (Itanium, architecture IA64).

Les autres principaux constructeurs de processeurs sont :

Advanced Micro Devices (Athlon, Opteron compatible Intel), Hewlett-Packard (HP9000), Silicon Graphics Inc (R14000), IBM (RS6000), Sun (Blade) (comparaisons.)

1.1.2 La mémoire centrale

Définitions

1) Dans un calculateur numérique, l'information est représentée par des variables qui ne prennent qu'un nombre limité de valeurs discrètes ; en prenant 10 valeurs représentées par les symboles 0 à 9 on définit le système décimal ; les calculateurs numériques utilisent le système binaire qui contient seulement deux valeurs discrètes notées 0 et 1 appelées BIT (BInary digiT).

2) Un groupement de 8 positions binaires est appelé un octet.

3) On note 1K. (1 Kilo) = 2^{10} soit 1.024 et 1M.(1 Méga) = 2^{20} soit 1.048.576

La mémoire centrale du calculateur est formée à partir de circuits intégrés contenant 1, 4, 16 ou 64 Méga positions mémoire par boîtier.

Une position mémoire est identifiée par une adresse et permet de mémoriser un bit, un octet ou deux octets. L'information mémorisée dans une position binaire de la mémoire représente donc soit la valeur 0 soit la valeur 1 en base 2 (Figure 1.3 a).

Une adresse mémoire est donc un nombre représenté par un ensemble de positions binaires : par exemple, pour sélectionner un mot parmi un Méga mot de mémoire il faut donc 20 positions binaires ; dans les calculateurs actuels les adresses sont codées en utilisant de 32 à 64 bits.

Considérons un boîtier mémoire possédant par exemple 4.194.304 adresses (4Méga) mémorisant un bit à chaque adresse. En montant en parallèle 32 de ces boîtiers on forme ainsi des mots de 32 bits permettant de stocker chaque information sur 32 positions binaires (Figure 1.3 b). Si une information est un nombre entier non signé on peut ainsi représenter tous les entiers de l'intervalle [0 .. 2^{32} - 1]. On obtient alors une mémoire centrale de 4 Méga mots de 32bits ; un accès à une telle mémoire fourni donc un mot sur 32 positions binaires.

Par exemple, la carte mémoire IBM contient 256 Méga octets répartis sur 2 ensembles de 4 composants, chacun fournissant 2 octets par adresse. Une valeur de l'adresse est comprise entre 0 et 16 Méga -1 pour chaque composant. Cette mémoire est donc composée de 32 Méga mots de 8 octets chacun.

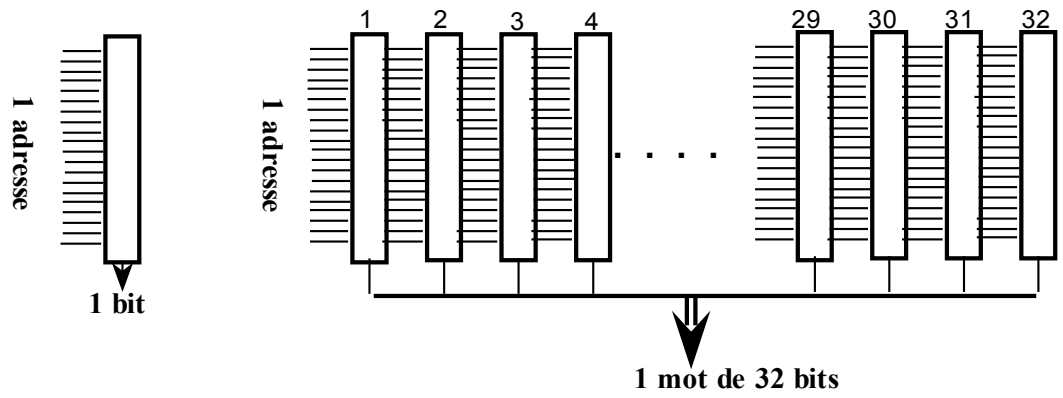

Figure 1.3a) Un boîtier mémoire de 1M.bits

Figure 1.3b) Une mémoire de 1M.mots de 32 bits

La taille du mot mémoire a évolué au cours du temps selon les capacités technologiques de 8, 16, 32, quelquefois 36 ou 64 bits pour être aujourd'hui de 32 ou 64 bits dans la plupart des calculateurs ; le nombre de mots mémoire dans une machine est très variable, fonction du prix de vente recherché et des performances attendues ; il se situe dans l'intervalle 4M.mots à 256M.mots selon les machines. La capacité mémoire des calculateurs croît de façon continue grâce à l'évolution technologique des circuits intégrés et la standardisation de la taille des adresses mémoire sur 32 bits qui permettraient d'implanter jusqu'à 2^{32} adresses soit plus de 4 milliards de mots mémoire.

1.1.3 Les contrôleurs de périphériques

Un contrôleur est un seul circuit intégré (pour un écran-clavier par exemple) ou une carte logique (pour le disque par exemple) chargé d'analyser les ordres fournis par le processeur central pour commander les dispositifs électro-mécaniques des périphériques contrôlés.

A chaque périphérique est associé un contrôleur et un programme spécifique qui fait partie du système d'exploitation de la machine fourni par le constructeur, chargé de piloter l'échange.

Parmi les contrôleurs les plus classiques on distingue les circuits d'E/S (Entrées/Sorties) séries qui échangent les informations bit par bit, les circuits d'E/S parallèles qui échangent 1, 2 ou 4 octets simultanément, les contrôleurs de disques (disques durs, disquettes) ou les contrôleurs d'écrans de visualisation.

Les circuits d'E/S séries permettent de connecter des terminaux clavier-écran, des imprimantes, des chaînes d'acquisitions de données spécifiques. Il existe deux familles principales : les circuits asynchrones (UART : Universal Asynchronous Receiver Transmitter) et les circuits synchrones/asynchrones (USART : Universal Synchronous Asynchronous Receiver Transmitter).

Définitions

Dans une transmission asynchrone de caractères, l'intervalle de temps séparant la transmission de deux caractères est quelconque (Figure 1.4a) ; le récepteur ne connaît donc pas l'instant d'émission du caractère ; chaque caractère doit alors disposer d'une marque identifiant le début de la séquence de bits le composant et d'un repère de fin pour permettre des contrôles sur l'échange (Figure 1.4b).

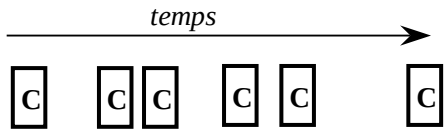

Figure 1.4a Transmission asynchrone

Figure 1.4b Format d'un caractère

En mode synchrone, les bits d'un bloc de données sont transmis sans discontinuité grâce à une synchronisation préalable des horloges de l'émetteur et du récepteur (Figure 1.5). Chaque bloc contenant généralement plusieurs milliers de bits est encadré par une séquence de caractères de synchronisation des horloges en début de transmission et des bits supplémentaires en fin de bloc permettant un contrôle d'erreur. Des caractères spéciaux permettent de repérer les débuts et les fins de blocs. Dans ce mode, la transmission entre un émetteur et un récepteur est permanente dès qu'elle est établie ; lorsqu'une absence d'information apparaît le contrôleur insère des caractères spéciaux de synchronisation pour maintenir l'échange.

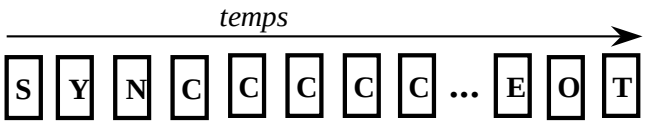

Figure 1.5 Transmission synchrone

L'UART assure la transmission caractères par caractères en encadrant chaque caractère d'un ou deux bits de début, d'un bit de fin et d'un bit de parité pour vérifier la transmission. La vitesse de transmission (entre 1.200 et 57.344 bits/s.), le format des caractères (5 ou 8 bits), le format des bits de contrôle (1 bit de début, 1 ou 2 bits de fin) et la direction des données sont programmables ce qui rend ce circuit universel pour l'adapter à n'importe quel type de transmission série.

Ainsi l'émission d'un caractère codé sur 8 bits, encadré d'un bit START et de 2 bits STOP entraîne l'envoi de 11 bits sur la ligne ; une vitesse de transmission de 57.344 bits/s. correspond à un débit maximum de 57.344 / 11 soit 5213 caractères/s.

L'USART peut assurer la fonctionnalité d'une UART et permet aussi d'effectuer des échanges synchrones en gérant des procédures de transmission particulières. Les échanges entre calculateurs interconnectés par un réseau de transmission de données s'établissent par des circuits de ce type qui permettent d'atteindre des vitesses de transmission beaucoup plus importantes qu'en mode asynchrone (plus de 10 M.bits/s).

Pour résumer, à titre d'exemple, la figure 1.6 décrit la carte mère d'un micro-ordinateur Intel. Le bus reliant le processeur au contrôleur de mémoire (82810E) est le bus système (FSB Front Side Bus) cadencé à 133, 266 ou 333 Mhz pour une largeur de 8 octets (Pentium III) ce qui fournit un débit crête de 2.128 GO/s. Pour le Pentium 4 le processeur enchaîne 4 transferts successifs par cycle à 100 Mhz ce qui fournit une bande passante de 3.2 GO/s.

Le bus mémoire relie le contrôleur mémoire à la mémoire centrale composée de SDRAM (Synchronous Dynamic Random Access Memory) de 133 Mhz. à 666 Mhz (ou de DDRAM (Double Data Rate DRAM à 236 ou 333 Mhz). Le bus d'accès à la mémoire fait quatre octets ce qui donne un débit de 533 MO/s pour la SDRAM et 1,066 MO/s. pour la DDRAM. Toutefois pour la DDRAM la transmission de données peut s'effectuer deux fois par cycle. De même des composants mémoire à 800 Mhz (Rambus DRAM) permettent d'atteindre un débit crête de 3.2 GO/s.

Le standard présent pour les interfaces graphiques est le bus AGP-4X (Accelerator Graphic Port). Il relie le contrôleur de mémoire à la carte graphique qui contient de la mémoire vidéo (SDRAM ou DDR-SDRAM). Le bus GP est cadencé à 66 Mhz, avec quatre accès maximum par cycle pour AGP-4X ou 8 accès par cycle AGP-8X ; pour une largeur de quatre octets la bande passante pour le 4X est de 1,066 Go/s. et sera pour le 8X de 2,132 Go/s. La nouvelle norme est le bus PCI Express (qui n’a rien à voir avec le bus PCI) est un bus série qui peut exister en plusieurs versions (de 1 à 32 voies) permettant d’obtenir des débits jusqu’à 8 Go/s soit 4 fois le débit du port AGP.

Après le contrôleur mémoire est installé le contrôleur d'entrées-sorties. Pour les disques le standard PC est le bus IDE/UDMA-ATA100. (2 canaux sur le schéma). Le bus IDE n'est pas synchronisé par une horloge ; le protocole d'échange est établi à partir de signaux asynchrones. Sa largeur est de 2 octets et la norme ATA100 permet d'atteindre 100 MO/s.

Pour les liaisons séries, le standard est le bus USB (Universal Serial Bus ; 4 ports et 2 canaux sur le schéma). Le bus est cadencé à 12 Mhz et donc un débit crête de 1.5 MO/s. (le standard USB2.0 est une extension du bus USB cadencé à 480 Mhz pour un débit de 48 MO/s.) Sur ce bus se connectent des claviers, des souris, des modems permettant l'accès à des réseaux.

Des cartes d'extensions série (SIO) permettent de connecter d'autres dispositifs séries.

D'autres contrôleurs de périphériques peuvent se connecter par l'intermédiaire de connecteurs enfichés sur des récepteurs femelles disposés sur la carte qui définissent un standard de bus : le PCI (Peripheral Component Interface). Ce bus est cadencé à 33 Mhz sur une largeur de 4 octets ce qui procure un débit crête de 133 Mo/s. Une version à 66 Mhz sur 8 octets fournit un débit de 512 Mo/s. AMD et Intel soutiennent deux évolutions concurrentes du bus d'E/S pour améliorer la bande passante actuellement disponible. Le bus PCI Express est en passe de supplanter le bus PCI pour l'interconnexion de cartes d'E/S ; les cartes SCSI (Small Computer System Interface) permettent de connecter des disques supplémentaires (640 Mo/s.).

Le circuit Flash Bios est le contrôleur de configuration et de démarrage. Il contient une mémoire PROM (Programmable Read Only Memory) qui stocke le programme de démarrage. Ce programme débute par la lecture de la configuration du calculateur rangé dans une mémoire Flash (mémoire programmable électriquement et rémanente) pour déterminer la présence des contrôleurs installés et leur fournir des valeurs d'initialisation. Le démarrage se poursuit jusqu'à l'identification d'un programme, généralement résident sur disque (le bootstrap), qui sera chargé en mémoire centrale pour démarrer le système d'exploitation du calculateur.

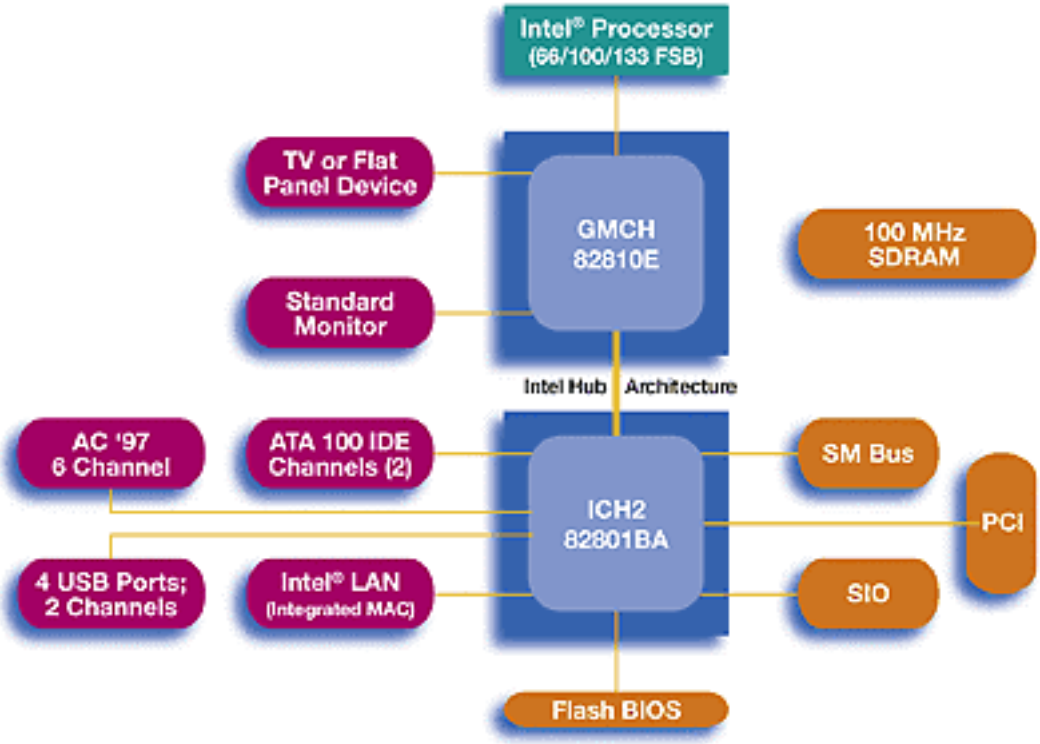

Fig.1.6 La carte mère d'un micro-ordinateur Intel ®

Et pour en savoir (ou en faire) plus…

1.2 Le logiciel

Le logiciel est l'ensemble des programmes composant une application qui devront être exécutés par le processeur. Un programme est une suite d'instructions s'appliquant sur des données d'entrée pour produire des données de sorties ; l'instruction est interprétée par le processeur et commande les dispositifs matériels du calculateur.

L'ensemble des instructions constitue un langage. Il existe plusieurs types de langages de programmation pour construire un programme, mais le calculateur ne peut exécuter que son langage machine ; en effet, le processeur a été conçu pour reconnaître le codage d'un certain nombre d'instructions sous la forme d'une séquence de bits particulière stockée dans un mot mémoire. La définition du langage machine est donc établie par le concepteur du processeur. Chaque processeur a son langage machine, certain d'une même famille pouvant être compatible entre eux (architecture 80x86).

Pour écrire un programme, le programmeur peut donc utiliser le langage machine, une représentation symbolique du langage machine appelé langage d'assemblage ou des langages évolués.

Considérons le programme qui réalise l'opération $b := a + b$ qui exprime que le contenu de la mémoire d'adresse *a* est ajouté au contenu de la mémoire d'adresse *b* et le résultat est écrit dans la mémoire d'adresse *b*, devant être exécuté sur un calculateur 16 bits (la taille du mot mémoire vaut 16 bits, les valeurs d'adresses étant aussi codées sur 16 bits)

1.2.1 Le langage machine

Le programmeur "devrait écrire" en mémoire centrale le programme suivant :

Adresse mémoire	*Contenu de la mémoire*	
0000000000000000	0 0 0 1 0 1 1 1 1 1 0 0 0 0 1 0	instruction 1
0000000000000001	0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0	adresse de l'opérande 1
0000000000000010	0 1 1 0 0 0 0 0 1 0 0 1 1 1 1 1	instruction 2
0000000000000011	0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1	adresse de l'opérande 2
0000000000000100	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	instruction 3
.	.	
.	.	
.	.	
0000000100000000	0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 1	valeur de l'opérande 1
0000000100000001	1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1	valeur de l'opérande 2

Le compteur programme (CP) est mis à 0 et l'ordre d'exécution est transmis au processeur ; la configuration binaire de la première instruction sera interprétée ainsi par le processeur :

- lire le contenu du mot mémoire dont l'adresse se trouve dans le mot mémoire suivant l'instruction,
- ranger cette valeur dans le registre N°2 du processeur (La valeur 2 provient du codage 0 0 1 0 des 4 derniers bits du mot).

Le compteur programme sera incrémenté de 2 pour lire l'instruction suivante ; la configuration binaire de la seconde instruction sera interprétée ainsi :

- lire le contenu du mot mémoire dont l'adresse se trouve dans le mot mémoire suivant l'instruction,
- ajouter la valeur lue au contenu du registre N°2,
- ranger le résultat dans l'adresse mémoire lue précédemment.

Le compteur programme sera à nouveau incrémenté de 2 pour lire l'instruction suivante ; la configuration binaire de la troisième instruction sera interprétée comme l'arrêt du fonctionnement du processeur.

Après l'exécution de ce programme seul le contenu du mot mémoire d'adresse 0000000100000001 sera modifié et contiendra la valeur du résultat soit la valeur

0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0.

(Voir le chapitre suivant sur la représentation des nombres).

Un analyseur d'état logique piégeant les sorties de la mémoire permet de "voir" ces configurations, la valeur 1 étant représentée par une tension positive (2V. à 5 V.) et la valeur 0 par une tension de 0 V.

On convient aisément du manque de lisibilité d'un tel programme et de la difficulté de programmer ainsi ! Un premier effort pour rendre ces informations plus lisibles a consisté à opérer un changement de base. Des transcodages très simples sont possibles en choisissant la base 8 ou la base 16 ; en base 8 il suffit de regrouper 3 positions binaires de la base 2 et utiliser les symboles de 0 à 7 ; en base 16 on regroupe 4 positions binaires de la base 2 et on utilise 16 symboles 0 à 9 puis A, B, C, D, E et F.

Le programme précédent se lit en octal (base 8) :

Adresse mémoire	*Contenu de la mémoire*	
000000	0 1 3 7 0 2	0 (001) (011) (111) (000) (010)
000001	0 0 4 0 0 0	0 (000) (000) (100) (000) (000)
000002	0 6 0 2 3 7	0 (110) (000) (010) (011) (111)
000003	0 0 0 4 0 1	0 (000) (000) (100) (000) (001)
000004	0 0 0 0 0 0	
.	.	
.	.	.
004000	0 0 0 1 2 3	0 (000) (000) (001) (010) (011)
004001	1 7 7 7 7 3	1 (111) (111) (111) (111) (011)

Ce même programme se lit en hexadécimal (base 16) :

Adresse mémoire	*Contenu de la mémoire*	
0000	1 7 C 2	(0001) (0111) (1100) (0010)
0001	0 1 0 0	(0000) (0001) (0000) (0000)
0002	6 0 9 F	(0110) (0000) (1001) (1111)
0003	0 1 0 1	(0000) (0001) (0000) (0001)
0004	0 0 0 0	
.	.	
.	.	
.	.	
0100	0 0 5 3	(0000) (0000) (0101) (0011)
0101	F F F B	(1111) (1111) (1111) (1101)

Bien que la représentation octale ou hexadécimale soit plus concise que la représentation binaire, il demeure très fastidieux de programmer ainsi. La première amélioration fût donc de concevoir une représentation symbolique de ces instructions et de traduire par programme cette représentation symbolique dans le code machine précédent.

1.2.2 Le langage d'assemblage

Il s'agit d'un langage qui permet d'exprimer les opérations que peut réaliser un processeur de façon plus expressive que le langage machine en utilisant des symboles pour coder les instructions et des directives à l'attention du programme traducteur.

Par exemple le programme précédent s'écrit :

```
        .ORG 0        Mettre l'origine du programme à l'adresse 0 de la mémoire
  MOVE MA,R2          Transférer le mot mémoire appelé MA dans le registre N°2
  ADD  R2,MB          Lire le contenu du mot mémoire appelé MB, ajouter la valeur
                      contenue dans le registre N°2 et re-écrire le résultat  dans MB
  HALT                 Arrêt d'exécution du programme

        .ORG 256      Ranger les données à partir de l'adresse 256 de la mémoire
MA:     .WORD  83     Mettre la valeur 83 dans le mot appelé MA
MB:     .WORD  -5     Mettre la valeur -5 dans le mot appelé MB

        .END          Fin du programme à traduire
```

Il y a bijection entre les instructions du langage d'assemblage et les instructions machines ; un programme analyse la syntaxe puis la sémantique de chaque instruction symbolique pour produire l'instruction machine équivalente. Ce programme traducteur s'appelle l'assembleur.

L'assembleur utilise un certain nombre de directives qui permettent de définir par exemple les adresses d'implantation du programme, d'initialiser des mots avec des données ou de réserver des zones mémoires nécessaires à l'exécution du programme.

Les lignes du programme précédent commençant par un **.** sont des directives ; les autres lignes sont des instructions. Noter la différence entre l'instruction HALT qui arrête le processeur et la directive .END qui marque la fin des lignes à assembler et stoppe donc le programme assembleur.

Bien que cette formulation du programme soit plus lisible, elle impose la connaissance du jeu d'instructions du processeur par le programmeur ; celui-ci doit connaître toutes les opérations que peut réaliser le processeur et gérer l'implantation des instructions et des données en mémoire. D'autre part un programme écrit pour un processeur donné ne peut s'exécuter que sur des machines ayant le même processeur ou un processeur compatible. Un programme écrit en langage d'assemblage doit donc être re-écrit pour pouvoir s'exécuter sur une machine différente.

Ces considérations ont conduit à la création de langages évolués de programmation dont les instructions ne manipulent pas directement les ressources de la machine.

1.2.3 Les langages évolués de programmation

La manière d'exprimer les instructions ne correspond plus à la représentation machine des opérations, mais aux objets abstraits définis par le programmeur. Pour cela un langage évolué est composé d'instructions et de méthodes de représentations des données orientées vers la classe de problèmes que ce langage doit pouvoir exprimer.

En langage C par exemple, le programme précédent s'écrit :

```
int main somme();
{
  int ma,mb ;
  ma = 83;
  mb = -5;
  mb = mb+ma ;
}
```

Un tel programme sera traduit en langage d'assemblage par un compilateur, puis assemblé pour produire le langage machine équivalent à ce programme.

Il existe une foule de langages évolués (certain en aurait recensé plus de 2.000!) représentant plusieurs styles de programmation contenus dans quatre grandes classes : les langages impératifs, les langages à objets, les langages fonctionnels et les langages déclaratifs.

La classe des langages impératifs contient les langages les plus anciens, les plus nombreux et les plus utilisés ; ils permettent de définir des actions opérant sur des données d'entrée pour produire des données de sortie : les plus courants (qui sont aussi les plus anciens) sont FORTRAN (FORmula TRANslating, COBOL (COmmon

Business Oriented Language), BASIC (Beginner's All-purpose Symbolic Instruction Code), C, Pascal et Ada.

Chacun de ces langages a historiquement un domaine d'application privilégié que l'on découvre dans l'extension de l'acronyme du nom. Le langage PASCAL a été créé pour l'enseignement des langages de programmation, le langage C pour l'écriture du système UNIX et le langage Ada à l'initiative du Département d'Etat de la Défense américain en vue de normaliser tous les développements de programmes pour les applications militaires.

Les langages à objets constituent une évolution des langages précédents en introduisant la notion d'objets sur lesquels s'appliquent des opérations. Dans ces langages le choix des objets précède celui des actions (contrairement aux langages précédents). Les objets sont des composants qui contiennent des attributs (données) et des méthodes (actions) qui décrivent le comportement de l'objet. La communication entre objets se fait par envoi de messages, qui donne l'accès à un attribut ou bien qui déclenche une méthode. Les principaux langages à objets sont Eiffel, C++, Java (et quelques autres qui n'ont d'objets que le nom !).

Les langages fonctionnels utilisent comme composant de base la notion de fonction. Un programme écrit dans un de ces langages est une suite de déclarations de fonctions indépendantes les unes des autres. Une fonction s'exécutera lorsque ses arguments auront été évalués. La notion précédente d'instructions opérant sur des données n'existe plus. Les langages LISP (LISt Processing), ML (Caml en France) sont les plus significatifs de cette classe.

Les langages déclaratifs fondés sur la logique permettent d'exprimer des relations permanentes entre des entités au moyen de règles. Une application est alors décrite par une base de règles et une base de faits ; exécuter un programme consiste à interpréter une requête en réalisant des inférences sur les faits au moyen des règles pour atteindre un but. Le langage PROLOG (PROgramming in LOGic) est le langage déclaratif par excellence.

Les méthodes de conception de programmes diffèrent fortement suivant la classe de langages évolués choisie pour réaliser la programmation.

1.3 Le Système d'Exploitation d'un calculateur

Le Système d'Exploitation (SE ou OS pour Operating System) est un ensemble de programmes associé à une machine ayant deux fonctions fondamentales et complémentaires :

La première, rendre la machine informatique accessible aux utilisateurs pour pouvoir développer des programmes et exécuter des applications.

Ensuite, rentabiliser l'utilisation des ressources du calculateur (processeur, mémoire, entrée-sortie) en enchaînant l'exécution de programmes et en partageant ces ressources entre plusieurs utilisateurs. Ces programmes systèmes constituent un logiciel indispensable à l'utilisation de la machine ; leurs fonctions essentielles

consistent à combler le fossé qui existe entre les besoins exprimés par l'utilisateur et les potentialités de l'architecture matérielle.

Il existe différents types de SE, fonction de l'environnement dans lequel se trouve le calculateur et donc adaptés aux problèmes à résoudre ; on peut distinguer cinq grandes classes de Système d'Exploitation.

(On emploiera aussi le terme *système* pour évoquer l'ensemble matériel plus logiciel d'un système informatique à ne pas confondre avec le système d'exploitation, ensemble de logiciels devant gérer la machine pour des programmes d'application).

1.3.1 Les systèmes de traitements par lots

L'utilisateur soumet des travaux que le calculateur exécutera les uns après les autres dans un ordre pré-établi fonction de paramètres définis par le concepteur du SE tels la catégorie de l'utilisateur (privilégié, commun), la priorité des travaux demandés sur une échelle variant de 0 à 7 par exemple, ou la disponibilité des ressources par rapport aux besoins du programme (place en mémoire centrale, temps d'occupation du processeur...) ; c'est un programme système, l'ordonnanceur, qui choisit la chronologie d'exécution des travaux soumis à la machine ; dans un temps indéterminé (de quelques minutes à plusieurs heures) le calculateur rend les résultats d'exécution du programme, sous forme d'un listing sur l'imprimante ou d'un fichier résultat sur le disque.

Ce sont les premiers systèmes montés sur les calculateurs qui correspondent à l'utilisation d'un lecteur de cartes perforées pour entrer les programmes et les données et d'une imprimante pour sortir les résultats. Dans les systèmes actuels, la liste des travaux à traiter sera mémorisée dans un fichier disque ; cette liste sera évaluée au bon gré du système d'exploitation dans les moments d'inactivité du calculateur. Les résultats seront produits soit sur l'imprimante soit sur un fichier disque pour exploitation ultérieure par l'utilisateur.

1.3.2 Les systèmes multiprogrammés

La multiprogrammation est une technique d'exploitation du calculateur qui tend à occuper le plus possible la ressource unité centrale dans le but initial de rentabiliser l'utilisation de cette ressource.

En effet, ce type de système est issu de deux constatations primordiales :

En premier lieu, le coût excessif du matériel. Dans les années 60, date d'apparition des premiers systèmes multiprogrammés le prix d'achat du matériel et le coût de sa maintenance étaient des postes très importants. Il importait donc de faire fonctionner la machine continuellement en exploitant le mieux possible la ressource unité centrale.

En second lieu, l'unité centrale n'est effectivement occupée qu'une faible partie du temps total d'exécution d'un programme dans un mode traitement par lots compte tenu de la disparité des temps de traitement à l'intérieur de cette unité centrale et des

temps d'entrée-sortie. On convient qu'il existe au moins un rapport 1.000 entre le temps de cycle d'une entrée ou sortie d'information et le temps d'exécution d'une instruction dans l'unité centrale.

Prenons par exemple une unité centrale qui exécute un programme en 2.000.000 de cycles machines à 2,5 Ghz puis affiche dix pages de texte sur un écran alphanumérique de 24 lignes et 80 colonnes connecté par une ligne série à 19.600 bits/s.

Le programme s'exécutera en 0,8 ms alors que l'impression nécessitera environ 10s. Si ce programme doit afficher une page tous les 200.000 cycles machine, alors dans la séquence exécution - impression, l'unité centrale est inactive et en attente de fin d'impression de la page 99,992 % du temps total d'exécution. N'oublions pas que cette unité centrale était initialement le composant le plus onéreux du système.

Donc, l'objectif du système multiprogrammé supportant l'exécution de ce programme sera de rentabiliser la ressource unité centrale en cherchant à prendre en compte l'exécution d'un autre programme au lieu d'attendre sans rien faire la fin de l'impression de la page.

Dès que le programme en cours d'exécution dans l'unité centrale :

* attend la libération d'une ressource (imprimante, disque, ...) ou
* attend l'arrivée d'une information (caractère d'un contrôleur de terminal, signal de fin d'impression d'une ligne ...) ou
* atteint une limite de temps imparti pour demeurer possesseur de l'unité centrale (limite choisie par le concepteur du système)

alors son exécution est suspendue au profit d'un autre programme dont il ne manquait que la ressource unité centrale pour s'exécuter.

Ces systèmes contiennent de nombreux programmes dont les algorithmes sont souvent complexes et qui peuvent contenir plusieurs centaines de milliers de lignes pour assurer notamment:

* la gestion de l'occupation de la mémoire centrale et de la liste des programmes prêts à être exécutés rangés sur le disque,
* le choix du programme à activer en l'amenant en totalité ou en partie en mémoire centrale,
* l'allocation des ressources (imprimantes, disques) de la machine aux différents programmes demandeurs en évitant les inter-blocages,
* l'intégrité des informations présentes dans la machine.

De par leur complexité, ces systèmes sont devenus très coûteux ; simultanément, le coût du matériel a diminué fortement et les coûts de développement, maintenance puis exploitation de ces systèmes multiprogrammés ont considérablement augmentés.

Cette multiprogrammation a donc évolué pour considérer non seulement le partage de ressources entre programmes mais aussi le partage d'informations entre plusieurs programmes d'un utilisateur ou entre plusieurs utilisateurs.

Sont apparus alors une troisième classe de systèmes : les systèmes interactifs.

1.3.3 Les systèmes interactifs

Contrairement au système de traitement par lots, ce type de système doit fournir une réponse immédiate (à l'échelle de temps d'un utilisateur) à une demande de traitement. L'utilisateur se trouve maintenant "en prise directe" avec le calculateur par l'intermédiaire d'un terminal de type écran-clavier.

Ces systèmes possèdent les caractéristiques des systèmes multiprogrammés avec en outre la contrainte supplémentaire du temps de réponse ; l'utilisateur doit patienter le moins possible devant son terminal après avoir transmis un travail à la machine. Ils peuvent être subdivisés en deux types différents :

1.3.3.1 Le système interactif transactionnel

Le travail soumis à un tel système est du type :

interrogation -> traitement -> réponse

Le système d'exploitation se doit de traiter chaque demande dans des temps raisonnables, fixés par le cahier des charges de l'application, et de veiller à l'intégrité des informations accédées sur les plans de confidentialité et de sécurité.

Le calculateur est programmé pour n'exécuter qu'une seule application, souvent très complexe, mais accessible par un grand nombre d'utilisateurs. Il s'agit des systèmes de réservation aérienne, des systèmes de gestion de ventes par correspondance ou des serveurs Internet par exemple.

1.3.3.2 Le système interactif temps partagé

Le système à la charge de partager la ressource machine entre tous les utilisateurs connectés de telle façon que chaque utilisateur ait l'impression d'avoir à sa disposition toutes les ressources de la machine ; pendant que vous tapez 10 caractères au clavier pour composer un ordre à donner à la machine par exemple, celle-ci est capable d'exécuter au moins 5 milliards d'instructions machine !

Elle peut donc exécuter plusieurs millions d'instructions composant des programmes appartenant à d'autres utilisateurs ; le système alloue donc l'unité centrale à chaque utilisateur pendant un quantum de temps, ce qui interdit à un programme de monopoliser l'unité centrale, et en gérant au mieux l'occupation de cette unité centrale.

Dès qu'un programme est en exécution, les autres programmes sont donc inactifs ; le quantum de temps étant de l'ordre de quelques dizaines de millisecondes, ces commutations de programmes n'apparaissent pas à l'utilisateur. Toutefois, si le nombre d'utilisateurs augmente, si les programmes sollicitent fortement l'unité centrale pour réaliser des calculs importants et si les entrées-sorties sont peu nombreuses alors les temps d'attente peuvent être visibles pour le programmeur.

Pour une machine de la puissance d'un IBM/POWER avec 1 Giga Octets de mémoire par exemple, et des programmes d'application de type tableurs, traitement de textes, jeux... le nombre d'utilisateurs acceptables est de l'ordre d'une trentaine ; au delà, même si la ressource unité centrale n'est pas saturée, les conflits d'accès aux ressources partagés, notamment le disque, entraînent des dégradations de performances très importantes.

Ces systèmes sont donc multi-utilisateurs, chaque utilisateur disposant d'un terminal écran-clavier connecté sur la machine, ou mono-utilisateur mais multitâche, l'utilisateur pouvant lancer plusieurs programmes qui s'exécutent en temps partagé ; c'est le mécanisme de multi-fenêtrage existant sur les stations de travail actuelles qui permet à un utilisateur de faire exécuter plusieurs programmes concurremment.

1.3.4 Les systèmes temps-réel

Il s'agit des systèmes d'exploitation mis en œuvre sur les machines devant contrôler et commander des processus (pilote automatique, surveillance de centrales nucléaires, autocommutateurs, robots autonomes ...) en faisant l'acquisition d'informations provenant du monde extérieur (températures, pressions, vitesses, positions ...) pour produire des données qui seront exploitées immédiatement par un pilote du système (alarmes, courbes d'évolution de données captées...), ou bien enregistrées pour exploitation ultérieure (télémesures, images ...) ou même qui provoqueront une action immédiate sur l'environnement (déclenchement du pistolet de peinture après positionnement du bras du robot devant la tôle à peindre, action de contre-mesure après une détection radar, fermeture de vannes ...).

Ces applications sont constituées de tâches de traitement dont les durées et les échéances font partie de la spécification du système ; par exemple, la commande du pistolet de peinture doit débuter 2 s. après le positionnement et être maintenue pendant 20 s. La validité de l'application doit être démontrée dans son domaine logique, ce qui est aussi le cas des applications s'exécutant sur les systèmes précédents, mais de plus dans son domaine temporel : une tâche lancée trop tard ou trop tôt peut provoquer des conséquences catastrophiques.

Ces systèmes doivent donc assurer une continuité de service (notion de tolérance aux pannes) et le respect des échéances et des durées des tâches qui composent l'application.

1.3.5 Les systèmes répartis et les systèmes distribués

Dans les premiers systèmes informatiques le coût du matériel d'un système installé était plus important que celui du logiciel nécessaire pour faire fonctionner le système ; rentabiliser l'utilisation du système revenait donc à partager ses ressources. Actuellement le coût du matériel est presque négligeable par rapport à celui du logiciel (une station de travail de 1 000 € peut supporter l'exécution de plusieurs logiciels standard dont la licence coûte de 2 000 € à 10 000 €, et le développement d'un logiciel spécifique peut rapidement coûter un million d'euros.) ; la logique

économique entraîne donc de dupliquer les installations (sans les pirater !!) de produits logiciels sur des machines différentes pour augmenter les séries de ces logiciels et mieux amortir leur coût.

En outre, il devient aussi rentable pour l'utilisateur de répartir les traitements de ses applications informatiques en des sites différents et récupérer localement toute la puissance de traitement pour une tâche donnée.

C'est ainsi que sont apparus les systèmes répartis, assurant une répartition des traitements sur différentes machines par l'intermédiaire d'un réseau de communication.

Pour un utilisateur, il s'agit donc de plusieurs systèmes d'exploitation (un par machine) capables de communiquer pour échanger des informations ; par exemple, dans une entreprise commerciale, la gestion de stock s'exécutera sur une machine située au magasin, la facturation sur une seconde située aux points de vente, la comptabilité et la gestion du personnel sur une troisième située à l'administration, chaque machine travaillant sur ses fichiers propres et mettant en œuvre des programmes spécifiques puis communiquant en fin de journée des états récapitulatifs permettant de mettre à jour certaines informations partagées par plusieurs services (la facturation et la comptabilité ou la gestion des stocks et le service achat par exemples).

Les services du réseau sont devenus plus complets et plus performants ; on a alors vu apparaître les systèmes distribués. Dans un tel système, l'ensemble des ordinateurs interconnectés par le réseau se comporte comme un seul système informatique, géré par un seul système d'exploitation ; les utilisateurs peuvent alors se connecter depuis n'importe quel terminal et accéder à n'importe quel programme ; de plus ils peuvent partager l'utilisation de toutes les ressources installées sur le réseau (imprimantes, traceurs, bases de données, calculateurs vectoriels...) par l'intermédiaire de calculateurs spécialisés appelés serveurs.

Parmi ces services du système distribué s'appuyant sur un réseau de communication, on peut citer la répartition des fichiers d'une application qui peuvent se trouver sur n'importe quel disque et qui peuvent être accédés depuis n'importe quelle machine : une machine A qui possède ou non un disque peut créer un fichier sur le disque de la machine B et lire les informations d'un fichier se trouvant sur une troisième machine C en indiquant seulement les noms des fichiers concernés ; le logiciel NFS (Network File System) de la norme OSF par exemple, assure cette gestion de la répartition des fichiers.

De même, les traitements peuvent être répartis entre plusieurs unités centrales, permettant ainsi un fonctionnement en parallèle de plusieurs calculateurs participant à l'exécution de la même application ; un programme de calcul d'images de synthèse par exemple (très gourmand en ressource mémoire et calcul) peut être découpé en plusieurs sous-programmes lancés par un calculateur initial sur plusieurs machines du réseau qui rendront leurs résultats de calculs au calculateur initial qui réalisera ensuite l'affichage de l'image calculée ; il existe plusieurs techniques pour gérer ce partage reposant sur une bibliothèque d'échanges de messages (MPI Message Passing Interface), sur des extensions de langages évolués (Java RMI) ou sur des protocoles d'interconnexion généraliste (CORBA).

Ces systèmes distribués ou répartis gèrent donc des réseaux d'ordinateurs ; ces réseaux peuvent être locaux, constitués alors d'un support permettant des débits importants d'information sur des câbles coaxiaux ou des fibres optiques : la norme Ethernet permet des échanges entre calculateurs hétérogènes ; la bande passante initiale de cette norme s'étend de 100 Mb./s à 10 Gb./s pour atteindre prochainement 100 Gb./s).

Les réseaux distants permettent de communiquer entre calculateurs éloignés sur un même territoire ou sur des continents différents : dans ce cas, ce sont les infrastructures des communications téléphoniques qui sont utilisées (câble, satellite) ou des réseaux spécialisés (Transpac, Numéris en France) ; les réseaux téléphoniques supportent des débits d'information jusqu'à 56K.bit/s et les réseaux spécialisés jusqu'à 144 K.b/s (Numéris) ; des progrès spectaculaires, via l'ADSL sont récemment apparus améliorant les débits des réseaux de transmissions pour pouvoir supporter les échanges d'images fixes, d'images animées ou de fichiers sons.

Il devient désormais possible par exemple, de lancer un calcul sur un super calculateur installé à Paris ou Londres et accessible par réseaux, pour visualiser, par exemple, les efforts subis entre deux pièces mécaniques mobiles et afficher en temps-réel les images sur un écran graphique couleur dans votre laboratoire. (Grid Computing, GLOBUS)

En conclusion, cette brève description des différents types de systèmes d'exploitation a pour but de montrer l'importance capitale de ces logiciels et de situer les contextes dans lesquels ils s'exécutent ; en fait, c'est le système d'exploitation qui spécialise un ordinateur pour une utilisation particulière.

Cet ensemble de programmes est donc étroitement lié aux ressources du calculateur puisqu'il en assure la gestion et à l'utilisateur puisqu'il lui permet de développer ses propres programmes et d'exploiter la machine.

Des programmes spécifiques, réalisés pour un type d'application donné, s'exécutent sous le contrôle du système d'exploitation et permettent de tirer parti d'un ordinateur sans avoir à développer des programmes : ce sont les progiciels.

1.4 Les progiciels

Un progiciel est un programme d'application, s'exécutant sous un système d'exploitation, permettant de traiter des données d'un problème sans avoir à développer de programmes. Ils sont composés d'une interface utilisateur généralement réalisée sous forme de menus présentant toutes les possibilités du programme et d'un ensemble de sous-programmes se retrouvant dans un ou plusieurs fichiers sur la mémoire de masse de l'ordinateur ; ce sont par exemple, les traitements de textes, les tableurs, les gestionnaires de bases de données, les outils XAO (tout processus Assisté par Ordinateur : D, Dessin ; CF, Conception-Fabrication ; S, Spécification ; E, Enseignement ; P, Publication ...).

Chapitre 2 : Représentation de l'information en machine

Toute information traitée par une machine informatique est représentée par une séquence de positions binaires ; il existe deux classes d'information manipulées par l'ordinateur : les instructions qui composent le langage machine et les données qui sont interprétées, modifiées ou produites par les instructions.

Le langage machine fait l'objet du chapitre suivant.

Les données sont des nombres ou des caractères alphanumériques : par exemple le nombre 121 ou la suite des 3 caractères "1" "2" "1" ; avec le nombre il sera possible d'effectuer toutes les opérations arithmétiques usuelles et sur la suite de caractères il sera possible de réaliser des permutations, des juxtapositions, des insertions et des impressions.

2.1 Représentation des nombres en machine

Une quantité d'objets N, dans un système de numération de base b, est représentée par la formule suivante :

$$N = a_q b^q + a_{q-1} b^{q-1} + ... + a_0 b^0 + ... + a_{-p} b^{-p}$$

dans laquelle les coefficients a_i sont des symboles élémentaires pré définis (dans le système décimal on utilise les symboles de 0 à 9) ; ce coefficient a_i exprime le nombre de fois que la valeur b^i est contenue dans le nombre ; il faut donc b-1 symboles pour représenter les a_i auxquels il faut rajouter le 0 pour signifier qu'aucune quantité b^i n'est à considérer pour le rang i.

Tous les a_i étant alors représentés, on conviendra d'écrire le nombre sous la forme :

$$N = a_q\, a_{q-1} .... a_0, ... a_{-p}$$

La position i occupée par le symbole a_i représente alors le poids attaché à ce symbole dans la base considérée.

Exemple : dans le système décimal à 10 symboles (0-9)

$$123{,}92 = 1*10^2 + 2*10^1 + 3*10^0 + 9*10^{-1} + 2*10^{-2}$$

Dans un calculateur numérique digital un chiffre est représenté dans le système binaire à 2 symboles (0 et 1).

Exemple : $101010{,}01 = 1*2^5 + 0*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 + 0*2^{-1} + 1*2^{-2}$

Pour simplifier l'écriture on utilise le système octal à 8 symboles (0-7) ou hexadécimal à 16 symboles (0-F) ; on symbole octal représente alors 3 symboles binaires et un symbole hexadécimal correspond à 4 symboles binaires.

Exemple : $101010{,}01 = (52{,}2)_8$
$= (2A{,}4)_{16}$

Les codes octal ou hexadécimal ne sont qu'une simplification d'écriture du même nombre binaire; ils sont donc utilisés pour représenter toute information circulant dans l'ordinateur ; par exemple le contenu suivant d'un registre de 8 bits 11111111 sera noté 377 en octal ou FF en hexadécimal.

2.1.1. Représentations des nombres négatifs

Un nombre arithmétique est représenté par un signe, une partie entière et une partie décimale. Le signe ne pouvant prendre que 2 valeurs (+ ou -), il sera représenté sur une position binaire ; on convient de donner la valeur 1 pour un nombre négatif et 0 pour un nombre positif.

La représentation d'un nombre algébrique nécessite donc n+1 positions : 1 position de signe et n positions de valeur absolue ; par convention, le bit de signe est situé après le bit de plus fort poids du nombre.

Exemple : 1 00011011 - 27
0 00000011 +3

Il existe plusieurs solutions pour représenter les nombres algébriques ; suivant ces choix de représentations les algorithmes qui traduisent les opérations arithmétiques de base sont différents.

<u>Première solution : signe et valeur absolue</u>

C'est la représentation de l'exemple précédent. L'algorithme des opérations arithmétiques est alors le suivant :
Comparer les signes des nombres à traiter.
S'ils sont égaux alors ajouter les valeurs absolues et donner au résultat le signe des opérandes.
S'ils sont différents alors déterminer la plus grande valeur absolue des deux nombres et soustraire la plus petite de la plus grande ; le signe du résultat est alors celui de la plus grande valeur absolue.
Cet algorithme est relativement compliqué et donc peu efficace.

Deuxième solution : Complément à 2^{n+1}

On cherche à trouver une écriture des nombres algébriques telle que lorsqu'une opération doit être effectuée sur ces nombres, alors une addition soit systématiquement effectuée sur leurs représentations considérées comme des nombres positifs en numération binaire. De plus, nous voudrions trouver une écriture telle que, l'addition effectuée, le résultat obtenu soit la représentation du résultat sans autre modification. Une telle écriture existe : c'est l'écriture en complément à 2^{n+1}.

Elle est fondée sur le fait que le nombre de positions binaires pour écrire un nombre est fixe et que l'on ne considérera pas la retenue provenant éventuellement de l'addition des bits de plus forts poids.

Dans cette écriture :

1) On dispose de n+1 cellules binaires pour représenter un nombre entier X
2) $0 \leq | X | \leq 2^n - 1$
3) a) si $X \geq 0$ alors $X = \sum_0^n x_i 2^i$ et du fait de 2) $x_n = 0$

Donc, pour tout nombre positif $\leq 2^n - 1$, l'écriture de X revient à écrire le nombre en binaire pur.

b) si $X < 0$ alors $\sum_0^n x_i 2^i = 2^{n+1} - | X |$ et du fait de 2) $x_n = 1$

Donc, pour tout nombre négatif, l'écriture du nombre revient à prendre la représentation en complément à 2^{n+1} de $|X|$.

Pour effectuer ce calcul, on remarque ($0 \leq X \leq 2^n$) :

$$2^{n+1} - X = (2^{n+1} - 1 - X) + 1$$

Or $2^{n+1} - 1 = 111...111$ sur n positions binaires et donc

$(2^{n+1} - 1 - X) = \overline{X}$ (le complément bit à bit de X) ; donc

$$2^{n+1} - X = \overline{X} + 1$$

Dans cette représentation la position de forts poids x_n indique le signe du nombre : 0 pour les nombres positifs et 1 pour les nombres négatifs ; de plus le 0 n'a qu'une seule représentation, tous les bits à 0.

Exemples : Considérons un registre 8 bits et les nombres algébriques suivants -128, +127, 5, -5

Représentation de -128 :

|-128| =	10000000
Complément bit à bit =	01111111
+	00000001
-128 =	10000000

(Remarquons que +128 n'est pas représentable sur 8 bits)

Représentation de +127 :

+127 = 01111111

Représentation de +5 :

+5 = 00000101

Représentation de -5 :

Complément bit à bit =	11111010
+	00000001
-5 =	11111011

Soit X et Y deux nombres algébriques, X* et Y* leurs écritures en complément à 2^{n+1} et +* l'opérateur d'addition arithmétique modulo 2^{n+1} on montre que

$$(X+Y)^* = X^* +^* Y^*.$$

Donc toute opération d'addition ou de soustraction sur des nombres représentés ainsi se résume à une addition, le résultat étant exprimé dans cette représentation.

2.1.2 Dépassement de capacité numérique

Tout nombre devant être représenté par un nombre fixe de positions binaires, certaines opérations arithmétiques peuvent fournir un résultat qui n'est pas représentable ; par exemple, sur 8 positions binaires, l'opération 120 + 136 donne un résultat qui ne peut être représenté que sur au moins 9 positions binaires ; il apparaît alors un signal, engendré par l'opérateur d'addition, indiquant que le résultat est incorrect ; ce signal est généralement récupéré par le système d'exploitation pour prévenir l'utilisateur qu'un résultat incorrect a été produit.

Un dépassement de capacité numérique est engendré lors d'une opération d'addition sur des nombres de signes identiques : si l'addition de deux nombres négatifs conduit à un nombre dont le signe est positif ou de façon symétrique, si l'addition de deux nombres positifs conduit à un nombre dont le signe est négatif alors le résultat est incorrect. Un circuit logique simple réalise cette fonction sur tous les opérateurs arithmétiques.

2.1.3 Représentation des nombres fractionnaires

La représentation d'un nombre fractionnaire nécessite une convention complémentaire pour placer la virgule : en choisissant un emplacement fixe on impose de gérer explicitement la position de la virgule lors d'opérations arithmétiques ; pour éviter cet écueil et pour pouvoir représenter de grands nombres ou des nombres très précis on choisit de représenter ces nombres à l'aide de la notation scientifique de la forme :

$$N = M * B^e$$

où M est la mantisse, B la base de numération et e l'exposant

La version informatique de cette représentation est l'expression en virgule flottante

2.1.4 Nombres en virgule flottante

Cette représentation permet de dissocier l'intervalle des nombres utilisés de la précision. L'intervalle est représenté par le nombre de chiffres de l'exposant et la précision par le nombre de chiffres de la mantisse.

2.1.3.1 Remarques préliminaires

Un nombre non entier ayant une écriture finie en base B1, n'aura généralement pas d'écriture finie en base B2. Par exemple

0,6 s'écrit 0,100110011001100.....

et donc toute représentation informatique de cette valeur n'est qu'une valeur approchée du nombre 0,6.

Le nombre n de positions binaires disponibles pour coder le nombre N est fixe ; une partie est alors affectée à la représentation de la mantisse et une partie à la représentation de l'exposant ; plus la taille de la mantisse est importante et plus les calculs sont précis mais en contre partie la taille de l'exposant est faible et l'étendue des nombres est réduite.

Les nombres en virgule flottante sont utilisés pour modéliser les nombres réels ; la discrétisation de la représentation des nombres fait que tous les réels ne peuvent être représentés exactement ; il s'ensuit donc des distorsions dans les calculs qui peuvent conduire à des erreurs si la précision est insuffisante.

Pratiquement on trouve des représentations des nombres flottants sur 32, 64 ou même 80 bits.

2.1.3.2 Représentation

Des normes internationales ont fixé la représentation des nombres flottants ; le format simple précision de la norme IEEE 754 contient 32 bits répartis ainsi :

Bit 31 = bit de signe du nombre : s
Bits 30 à 23 (8 bits) = exposant
Bits 22 à 0 (23 bits) = mantisse.

Selon cette norme la valeur est définie ainsi :

1) Si e=0 et M ≠ 0 ou bien e=255 alors erreur de calcul (NaN, Not a Number)

2) Pour 0 < e < 255 alors le nombre représente la valeur $(-1)^s * (1+M/(2^{23}))* 2^{e-127}$

3) Si e=0 et M=0 alors la valeur est 0.

Lors d'opérations arithmétiques ou de conversion, un processus de normalisation transforme le nombre obtenu en multipliant par 2 la mantisse jusqu'à amener un 1 en position forts poids et diminuant d'autant la valeur de l'exposant pour augmenter le nombre de chiffres significatifs de la mantisse et donc la précision des nombres.

Cette norme définit quatre mode d'arrondi vers infini négatif, infini positif, vers 0 ou au plus près. L'arrondi vers 0 est une troncature de la mantisse (arrondi par défaut pour les nombres >0 mais par excès pour les nombres <0). L'arrondi vers l'infini positif est une troncature pour les négatifs et une troncature suivie d'une incrémentation pour les positifs. L'arrondi vers l'infini négatif est une troncature pour les positifs et une troncature suivie d'une décrémentation pour les négatifs. L'unité flottante des processeurs comprend un registre d'état qui mémorise le mode d'arrondi.

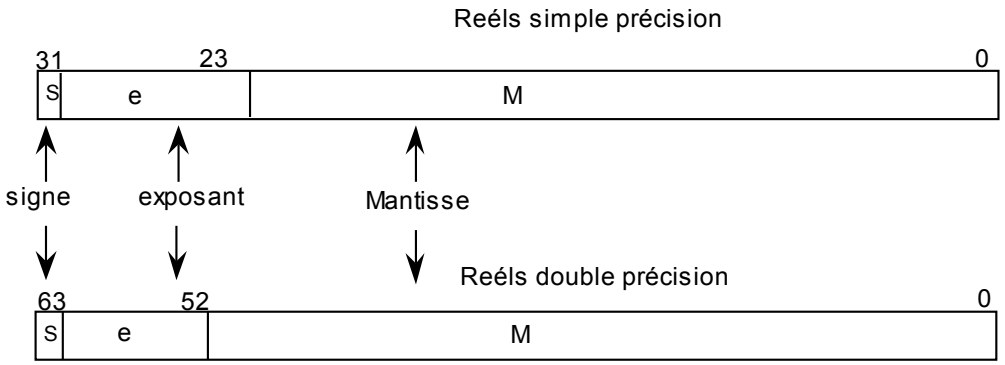

Fig2.1 Format à la norme IEEE 754 de la représentation des nombres flottants

	En simple précision	En double précision
Nombre positif maximum :	$3,4 * 10^{38}$	$18 * 10^{307}$
Nombre positif minimum :	$1,2 * 10^{-38}$	$2.2 * 10^{-308}$
Précision :	$0,12 * 10^{-6}$	$0.25 * 10^{-15}$

2.2 Conversions

2.2.1 Décimal en binaire

Deux méthodes permettent de convertir un nombre décimal en binaire : la première consiste à soustraire du nombre décimal la plus grande puissance de 2 qui lui est inférieure, puis de répéter le processus sur la différence ; le nombre binaire est composé en mettant un 1 dans les positions correspondant aux puissances de 2 utilisées et un 0 pour les positions inutilisées.

Exemples : 145 = 128 + 17 = 128 + 16 +1 s'écrit 1 0 0 1 0 0 0 1
145,75 = 128 + 16 +1 + 1/2 +1/4 s'écrit 1 0 0 1 0 0 0 1 , 1 1

Dans la seconde méthode on sépare la partie réelle de la partie fractionnaire.
Pour convertir la partie réelle on divise le nombre par 2, puis les quotients obtenus jusqu'à obtenir le quotient 0 ; la suite des restes des différentes divisions représente le nombre binaire, le faible poids étant le premier reste obtenu.

Exemple :

quotient	reste
145	
72	1
36	0
18	0
9	0
4	1
2	0
1	0
0	1

Pour convertir la partie fractionnaire on multiplie par 2 cette partie ; on réitère ensuite sur les parties fractionnaires des nombres obtenus jusqu'à la précision voulue ou la valeur 0 ; la suite des parties entières représente le nombre binaire, le fort poids étant la première partie entière obtenue.

Exemples :

	partie fractionnaire	partie entière		
a)	0,75	1	,5	
	0,5	1	,0	0,75 = 1 1
b)	0,56	1	,12	
	0,12	0	,24	
	0,24	0	,48	
	0,48	0	,96	
	0,96	1	,92	
	0,92	1	,84	
	0,84	1	,68	
	0,68	1	,36	

... .

0,56 = 1 0 0 0 1 1 1 1 ....

2.2.1 Binaire en décimal

De façon évidente on opère la somme des puissances de 2 représentées par les positions à 1 du nombre binaire.

Exemple : 1 0 1 0 1 1 est le nombre décimal 1 + 2 + 8 + 32 = 43

2.2.2 Décimal en complément à 2

On rappelle qu'il faut n+1 positions pour représenter un nombre, la position de forts poids représentant le signe.

Pour un nombre positif, on convertit le nombre en binaire. Pour un nombre négatif on prend le complément à 2^{n+1} du nombre positif obtenu en ajoutant 1 au nombre représenté par le complément bit à bit du nombre initial ; plus simplement, cette représentation s'obtient en complémentant tous les bits du nombre initial à partir du premier 1 non compris et en commençant par les faibles poids.

Exemple : - 43 +43 = 0 1 0 1 0 1 1
Complément = 1 0 1 0 1 0 0
+1 = 1 0 1 0 1 0 1 -43 = 1 0 1 0 1 0 1

2.2.3 Décimal en flottant

Il faut trouver la valeur binaire du signe s, de la mantisse M et de l'exposant e.

Dans la norme IEEE 754 on opère ainsi : s = 0 si N ≥ 0, s = 1 si N < 0.

On cherche la première puissance de 2 inférieure au nombre et on convertit cette valeur en binaire; on rajoute 127 à cette valeur pour obtenir l'exposant.

On obtient la mantisse en effectuant la division du nombre à convertir par la puissance de 2 trouvée : cette valeur est donc comprise entre 1 et 2.

Exemple : 432
432 = 256 * 1,6875
$256 = 2^8$ donc e = 01111111 + 00001000 = 10000111
0,6875 = 0,5 + 0,125 + 0,0625 donc M = 101100000000...0

donc 432 s'écrit en flottant IEEE simple précision

432 = 0 10000111 10110000000000000000000

D'autres représentations des nombres flottants existent selon les constructeurs de machine conduisant à des précisions et des dynamiques différentes.

2.3 Autres représentations des nombres

Suivant le type d'opérations qui sera effectué sur les nombres, il sera plus intéressant d'utiliser d'autres représentations des nombres. En effet, la représentation flottante permet par exemple de manipuler de grands nombres avec précision ; par contre tout traitement sur ces nombres nécessite des procédures de conversion qui sont relativement complexes et donc prennent un temps machine qui peut devenir considérable.

En informatique de gestion on imprime des résultats avec généralement de deux à quatre chiffres après la virgule et on est amené à effectuer de nombreuses entrées-sorties de chiffres. Certains constructeurs proposent alors le format DCB (Décimal Codé Binaire) dans lequel chaque symbole décimal est codé par sa représentation binaire. Pour coder 10 symboles il faut donc 4 positions binaires ; un nombre en DCB sera donc une suite de quartets précédée par un bit de signe.

Par exemple le nombre 192 sera codé 0001 1001 0010.

On pourra coder le signe sous la forme d'un bit ou bien en utilisant une combinaison sans objet de la représentation des chiffres (1111 par exemple). En utilisant un tel code les conversions pour les entrées sorties sont immédiates. De nombreux constructeurs d'unités centrales proposent le format DCB et des instructions arithmétiques associées pour manipuler ces nombres.

2.4 Représentation des caractères alphanumériques

L'ensemble des caractères alphanumériques couramment utilisé est composé des 10 chiffres décimaux (0-9), des 26 lettres de l'alphabet en majuscules et minuscules (A..Z, a..z), des caractères spéciaux tels + - , ? ;. / : = % $ * ^ ( ) & etc...., et de certains caractères de contrôle qui n'ont aucune représentation graphique.

Le code alphanumérique standard est le code ASCII (American Standard Code for Information Interchange) qui code toute information alphanumérique sur 7 positions binaires et donc permet de représenter 128 caractères (Fig2.2).

Un caractère est représenté sur un octet ; le huitième bit, bit de forts poids de l'octet, est utilisé comme bit de parité et la comparaison entre la valeur transmise et la valeur calculée par le récepteur permet de déceler éventuellement des erreurs de transmissions lors d'échanges de caractères entre des dispositifs d'entrées-sorties et l'unité centrale.

Ce bit est mis à 1 lorsque le nombre total de bits de l'octet, bit de parité compris, est pair ; il est mis à 0 dans le cas contraire.

Si, lors d'une transmission, un nombre impair de positions binaires ont été modifiées alors une erreur de transmission est décelée par le récepteur.

Exemples :	Codes ASCII sur 7 bits	Codes lus ou transmis sur 8 bits
A	1000001	01000001
B	1000010	01000010
C	1000011	11000011
0	0110000	00110000
1	0110001	10110001
2	0110010	10110010
3	0110011	00110011

Remarque : Pour convertir en binaire un chiffre codé en ASCII et, par exemple, rentré dans l'unité centrale par l'intermédiaire d'un clavier, il faut alors prendre les quatre bits de faible poids de l'octet.

Il existe d'autres codages pour résoudre le problème de la représentation de caractères non prévus en ASCII (é, ö, #, …) tel Unicode qui spécifie un numéro unique pour chaque caractère, quelle que soit la plate-forme, quel que soit le logiciel et quelle que soit la langue.

Des codes plus élaborés, utilisés lorsque la répétition du message transmis est impossible, ou lorsqu'un niveau de sécurité important est nécessaire, permettent de corriger le message après détection d'une ou plusieurs erreurs.

Le principe de la détection d'erreur consiste à former à partir d'une suite de (n-k) bits un message de n bits dont k sont fonctions de la valeur des autres.

2.5 Un exemple de code correcteur d'erreur simple : le code de Hamming

Nous évoquons ce code à titre d'exemple, ce problème étant traité dans des études plus générales sur la théorie des codes.

Supposons une information initiale sur 4 bits. Hamming proposa de créer 3 bits de parité portant sur trois groupes différents de 3 des 4 bits initiaux de la sorte :
Soit P la fonction parité, telle qu'elle a été définie précédemment (§2.4).

$$a4 = P(a0,a1,a2)$$
$$a5 = P(a1,a2,a3)$$
$$a6 = P(a2,a3,a4)$$

Le message alors obtenu est composé des bits a0,a1,a2,a3,a4,a5,a6.

On suppose qu'une seule erreur au plus est possible lors de la transmission.

Soit b0,b1,b2,b3,b4,b5,b6 le message reçu. On forme b'4,b'5 et b'6 avec la fonction parité sur les bits contrôlés ; s'il n'y a pas eu d'erreurs b'j= bj pour j valant 4,5 ou 6.

On forme alors

s4 = P(b'4, b4) = P [P(b0,b1,b2),b4)]
s5 = P(b'5, b5) = P [P(b1,b2,b3),b5)]
s6 = P(b'6, b6) = P [P(b2,b3,b4),b6)]

Le fait qu'une quantité sj vaille 1 indique que le bit erroné appartient au sous ensemble des bits contrôlés par la fonction parité.

De façon générale, il y a n+1 situations possibles : soit l'un des n bits est transmis de façon erronée soit il n'y a pas d'erreur. Puisqu'on dispose de k variables pour distinguer tous les cas il faut $2^k \geq n+1$. (Dans l'exemple précédent n = 7 et k = 3).

Une erreur sur a0 correspond à s6 = 0, s5 = 0, s4 = 1
Une erreur sur a1 correspond à s6 = 0, s5 = 1, s4 = 1
Une erreur sur a2 correspond à s6 = 1, s5 = 1, s4 = 1
Une erreur sur a3 correspond à s6 = 1, s5 = 1, s4 = 0

Le décodage de la valeur de s4s5s6 permet donc de retrouver le bit erroné et de modifier sa valeur.

oct	dec	hex	caractère		oct	dec	hex	caractère
0	0	0	NUL	^@	40	32	20	space
1	1	1	SOH	^A	41	33	21	!
2	2	2	STX	^B	42	34	22	”
3	3	3	ETX	^C	43	35	23	#
4	4	4	EOT	^D	44	36	24	$
5	5	5	ENQ	^E	45	37	25	%
6	6	6	ACQ	^F	46	38	26	&
7	7	7	BEL	^G	47	39	27	’
10	8	8	BS	^H	50	40	28	(
11	9	9	TAB	^I	51	41	29	)
12	10	A	LF	^J	52	42	2A	*
13	11	B	VT	^K	53	43	2B	+
14	12	C	FF	^L	54	44	2C	’
15	13	D	CR	^M	55	45	2D	-
16	14	E	SO	^N	56	46	2E	.
17	15	F	SI	^O	57	47	2F	/
20	16	10	DLE	^P	60	48	30	0
21	17	11	DC1	^Q	61	49	31	1
22	18	12	DC2	^R	62	50	32	2

23	19	13	DC3	^S	63	51	33	3
24	20	14	DC4	^T	64	52	34	4
25	21	15	NAK	^U	65	53	35	5
26	22	16	SYN	^V	66	54	36	6
27	23	17	ETB	^W	67	55	37	7
30	24	18	CAN	^X	70	56	38	8
31	25	19	EM	^Y	71	57	39	9
32	26	1A	SUB	^Z	72	56	3A	:
33	27	1B	ESC	^[	73	59	3B	;
34	28	1C	FS	^|	74	60	3C	<
35	29	1D	GS	^]	75	61	3D	=
36	30	1E	RS	^^	76	62	3E	>
37	31	1F	US	^_	77	63	3F	?

oct	dec	hex	character	oct	dec	hex	character
_							
100	64	40	@	140	96	60	’
101	65	41	A	141	97	61	a
102	66	42	B	142	98	62	b
103	67	43	C	143	99	63	c
104	68	44	D	144	100	64	d
105	69	45	E	145	101	65	e
106	70	46	F	146	102	66	f
107	71	47	G	147	103	67	g
110	72	48	H	150	104	68	h
111	73	49	I	151	105	69	i
112	74	4A	J	152	106	6A	j
113	75	4B	K	153	107	6B	k
114	76	4C	L	154	108	6C	l
115	77	4D	M	155	109	6D	m
116	78	4E	N	156	110	6E	n
117	79	4F	O	157	111	6F	o
120	80	50	P	160	112	70	p
121	81	51	Q	161	113	71	q
122	82	52	R	162	114	72	r
123	83	53	S	163	115	73	s
124	84	54	T	164	116	74	t
125	85	55	U	165	117	75	u
126	86	56	V	166	118	76	v
127	87	57	W	167	119	77	w
130	88	58	X	170	120	78	x
131	89	59	Y	171	121	79	y
132	90	5A	Z	172	122	7A	z
133	91	5B	[	173	123	7B	{
134	92	5C	\	174	124	7C	|
135	93	5D	]	175	125	7D	}
136	94	5E	^	176	126	7E	-
137	95	5F	_	177	127	7F	del

Fig.2.2 La Table des codes ASCII

Chapitre 3 : LES SERVICES DU SYSTÈME D'EXPLOITATION (SE)

Un système d'exploitation est une collection de programmes qui fournit un environnement d'exécution pour les utilisateurs de la machine leur permettant de développer des programmes et d'exploiter les services fournis par le matériel. Un constructeur d'ordinateur propose donc un matériel et un système d'exploitation fournissant un certain nombre de services.

Dans tous systèmes d'exploitation on retrouvera toujours deux classes de services complémentaires fournis par des programmes qui permettent :

1) d'exploiter et de programmer un ordinateur,
2) de partager les ressources de la machine et les informations qu'elle mémorise entre plusieurs programmes utilisateurs.

Les services de la première classe sont accessibles à l'utilisateur par l'intermédiaire d'une liste de commandes donnant accès aux fichiers, aux programmes en cours d'exécution (les processus) et aux périphériques.

Les programmes utilisateurs s'adressent au système d'exploitation pour utiliser une ressource de la machine ; il supervise l'utilisation des ressources machines (mémoire centrale, processeur, entrées-sorties) pour fournir les ressources nécessaires à chaque programme de la façon la plus efficace.

4.1 Les principaux services d'exploitation d'un calculateur

Ces services, exprimés par l'intermédiaire de commandes ayant une syntaxe et une sémantique propres à chaque constructeur, sont proposés dans le langage de commande.

Il s'agit du langage SHELL dans le système UNIX, du JCL (Job Control Language) dans le monde IBM, CSI (Command String Interpreter) pour d'autres ou DOS (Disk Operating System) pour les IBM/PC. Les systèmes actuels Windows ou Mac-OS permettent d'accéder à ces services par des interfaces graphiques et des menus plus conviviaux.

Dans tous systèmes d'exploitation on retrouvera les fonctions suivantes, exprimées par des commandes (malheureusement !) différentes.

4.1.1 La manipulation de fichiers

Le SE permet l'utilisation d'un type d'information abstrait, le fichier, unité logique de mémorisation d'informations, stocké sur une unité physique (généralement un disque ou une bande magnétique).

Grâce à ces programmes du SE, qui implémentent cette abstraction, l'utilisateur "voit" un fichier comme une collection ordonnée d'informations de même nature ; les opérations d'accès à un fichier manipulent cette abstraction.

Les commandes essentielles sur les fichiers sont alors :

a) Création

Un fichier est créé par copie d'un fichier existant, ou pour un fichier devant contenir du texte, par un programme spécifique, l'éditeur de textes.

Les créations de fichiers sont généralement le résultat de l'exécution d'autres commandes ou de programmes. Par exemple, l'assembleur crée le fichier du programme binaire lorsqu'il traduit les instructions du langage d'assemblage contenu dans un fichier texte.

b) Destruction

Cette opération consiste à supprimer un fichier du support sur lequel il se trouve et à récupérer la place qu'il occupait pour une utilisation ultérieure ; le fichier détruit n'a donc plus d'existence logique ; sur certains systèmes, il peut encore avoir une existence physique et donc être récupéré par une autre commande permettant de restituer l'accès au fichier ; en fait, dans ce cas, ce ne sont que les informations de référence sur le fichier qui sont masquées : les données ne sont pas effacées.

Par exemple, sur le MacOS ou sous Windows, le fait de glisser un fichier dans la poubelle ne supprime pas ce fichier mais seulement l'accès depuis le dossier ; ce n'est que le vidage de la poubelle qui détruit physiquement le fichier et restitue l'espace occupé.

Exemple sous le système UNIX :

rm fich2 suppression du fichier fich2

c) Recopie

Une copie strictement identique du fichier identifié est transférée dans un nouveau fichier qui sera créé avec le nom donné dans la commande. Si ce nom avait déjà été donné à un fichier, alors cet ancien fichier sera remplacé par le nouveau.

Ex: cp fich1 fich2 recopie de fich1 dans un fichier de nom fich2

d) Renommage

Cette commande permet de modifier le nom d'un fichier sans en faire une copie.

Ex: mv fich1 fich2 renommage de fich1 en fich2

e) Affichage du contenu

Tout fichier contenant du texte doit pouvoir être imprimé sur un écran ou un terminal imprimant. Les fichiers textes sont composés de caractères alphanumériques codés en ASCII ; les dispositifs d'entrées-sorties reconnaissent ce code qui est alors traduit pour produire la calligraphie des caractères que l'on connaît.

Des fichiers contenant du format binaire, résultat d'assemblage par exemple, ne pourront donc être affichés.

Ex: lpr prog.asm impression du contenu du fichier prog.asm

f) Tri et recherches d'informations

Les fichiers de données doivent être exploités suivant différents critères (ordre alphabétique, chronologique, par type d'information...) qui nécessitent d'opérer des tris sur le fichier initial ; ces programmes sont essentiels en informatique de gestion.

D'autre part, lorsqu'on développe de nombreux programmes il est nécessaire d'avoir des outils pour analyser les fichiers sources en recherchant des informations ou comparant les contenus par exemple ; le système Unix fournit pour cela un certain nombre de commandes.

Ex: sort -r noms
Trie, par ordre alphabétique inverse, le fichier noms

cmp -l fich1 fich2
Fournit l'emplacement et la valeur des caractères différents dans la comparaison de fich1 et fich2.

grep -n langage fich1
Recherche le mot langage dans le fichier fich1

4.1.2 L'édition de textes

Les éditeurs de textes permettent de composer et de modifier des fichiers contenant du texte.

Il existe plusieurs types d'éditeurs de textes dont l'existence est liée à l'évolution technologique des dispositifs d'entrée d'information : cartes perforées, téléimprimeur, écran alphanumérique ou écran bit-map.

Les éditeurs lignes gèrent des lignes de textes, soit en les numérotant, soit par rapport à la ligne courante ; ils contiennent ensuite des commandes pour positionner le curseur sur le caractère recherché. Ils sont d'un emploi peu commode et correspondent à des dispositifs d'entrée peu évolués. ED sous UNIX est un éditeur ligne ne pouvant traiter que des fichiers de 128 K. octets.

Les éditeurs vidéo gèrent une page de texte et autorisent l'accès au texte par déplacement du curseur sur l'écran ; ils permettent aisément de rajouter, supprimer, modifier une ligne ou un caractère dans une page de texte ; leur emploi est relativement

simple (après avoir étudié l'ensemble des commandes proposées !) et ils correspondent à l'utilisation de terminal vidéo.

Il existe de nombreux éditeurs de textes, proposés par les constructeurs de calculateurs et donc spécifiques (EDIT sous HP, Textedit sous Sunview des SUN par exemple) ou liés à un système d'exploitation standard que l'on retrouve sur de nombreuses machines, EMACS ou VI par exemple.

L'éditeur de textes constitue le service à utiliser pour composer du texte dans un fichier et pour le modifier.

4.1.3 Les outils de développement de programmes

Le rôle d'un calculateur est donc d'exécuter des programmes pour le compte d'un utilisateur ; un programme est écrit, soit en langage d'assemblage, soit en langage évolué; le système d'exploitation fournit les programmes capables de transformer ces programmes en binaire et de les charger dans la mémoire centrale en vue d'être exécutés par le processeur.

a) l'assembleur

L'assembleur transforme un programme écrit en langage d'assemblage en code binaire; ce programme interprète les directives qu'il contient pour permettre la génération des instructions machine ; par la suite, il transforme chaque instruction symbolique en sa configuration binaire correspondante et produit les adresses absolues des mots mémoire référencés ; il existe donc un assembleur par machine.

b) le compilateur

Le compilateur analyse les instructions du langage évolué pour produire la suite d'instructions machine dont l'exécution réalisera l'opération spécifiée par le programmeur. Ce programme réalise donc les analyses lexicales, syntaxiques et sémantiques du programme source pour pouvoir produire un programme en instructions machine strictement équivalent ; un programme, une fois compilé, peut donc s'exécuter sans nécessiter la présence du programme source dont il est issu puisque c'est la représentation en instruction machine qui sera interprétée par l'unité centrale ; il existe donc un compilateur par langage évolué.

c) l'interpréteur

Par contre, un interpréteur est un programme qui interprète directement une forme intermédiaire du programme source, sans transformation préalable de ce programme en langage spécifique d'une machine (c'est le cas de l'interprétation du langage Java dont un analyseur transforme le texte original en un "byte-code" indépendant de tout système). Il est composé d'un ensemble de modules qui réalisent la lecture du texte à interpréter, l'analyse syntaxique de l'instruction puis l'interprétation immédiate de l'ordre reconnu ; l'exécution d'un programme écrit avec un langage interprété nécessite donc la présence du texte source pour être exécuté.

Un interpréteur est moins performant qu'un compilateur, car à chaque exécution du programme, il se produit une analyse et une reconnaissance des instructions intermédiaires par des sous-programmes de l'interpréteur alors qu'un compilateur n'effectue ces opérations qu'une seule fois. Par contre, l'interpréteur permet une exécution très interactive des programmes qui peuvent être testés au fur et à mesure de leur composition ; en outre ces interpréteurs sont généralement plus simple à développer que des compilateurs. La principale caractéristique des langages interprétés est donc la portabilité puisqu'il suffit de disposer de l'interprète de la forme intermédiaire du langage source pour exécuter un programme sur un calculateur.

d) la mise au point de programmes

Le développement de programmes nécessite un outil de mise au point ("debug") qui permet de suivre pas à pas l'exécution d'un programme pour déceler des erreurs de programmation.

Lors d'une programmation en langage d'assemblage, cet outil permet de suivre l'exécution du programme directement en binaire, instruction par instruction et de visualiser le contenu d'un mot mémoire ou d'un registre de l'unité de calcul après l'exécution de chaque instruction.

Lors d'une programmation en langage évolué compilé il est indispensable de disposer d'un outil qui permet de retrouver les instructions et les variables dans leur forme symbolique puisque, après compilation, c'est un programme différent qui s'exécute.
Une commande de compilation particulière permet d'inclure dans le fichier résultat le nom de chaque symbole utilisé par le programmeur et leur correspondance avec les instructions machine engendrées ; ainsi la référence à une information binaire lors de l'exécution pourra être présentée sous la forme du symbole initial. Le programme ainsi testé est donc une version de mise au point du programme final.
La compilation sans cette option engendre donc un programme différent du précédent : il n'est alors pas exclu que des erreurs subsistent puisque l'objet qui était en test est différent de l'objet final !

L'existence d'une industrie de développement de programmes a provoqué des besoins supplémentaires en outils de développement qui vont bien au-delà des outils de mise au point évoqués. Ces besoins conduisent à la réalisation d'atelier logiciel pour la production de logiciel ; ils ne font pas partie de l'ensemble des programmes du système d'exploitation mais sont considérés comme des progiciels spécifiques dans un processus de production particulier.

4.1.4 Les outils de génération de programmes

a) l'éditeur de liens

Le développement de projets logiciels nécessite l'écriture de plusieurs milliers de lignes d'instructions et parfois plusieurs millions ; il est donc fondamental de pouvoir structurer clairement la programmation de telles applications et de disposer d'outils du SE pour gérer cette structuration.

Pour cela on utilisera trois niveaux hiérarchiques comprenant :

- des programmes
- des modules
- des sous-programmes et des bibliothèques

Une application est d'abord découpée en programmes ; un programme correspond à une ou plusieurs fonctions importantes de la conception de l'application ; des données peuvent être produites par un programme pour servir de données d'entrées à d'autres programmes ; cette communication est généralement réalisée par fichiers explicitement créés par le programmeur ou implicitement par le SE.

Un programme est encore une entité trop importante pour être appréhendé dans sa totalité par un programmeur ou peut nécessiter des fonctions très générales développées par ailleurs (bibliothèque de fonctions mathématiques, graphiques ou autres). On définit alors la notion de modules, unités de compilation pouvant faire référence à des modules compilés séparément.

Par exemple, trois programmeurs peuvent développer trois modules différents d'un même programme aujourd'hui, vouloir les compiler et tester leur exécution, puis développer de nouveaux modules demain. Le test s'effectue en simulant les interactions entre les modules. Se pose alors le problème de l'intégration de ces différents modules en un seul programme exécutable : l'éditeur de liens est un programme du SE chargé de relier entre eux les modules compilés séparément.

Les références à des modules compilés séparément sont notés "externes" dans le programme et ne seront donc pas compilés ; l'éditeur de liens reprend toutes les références externes et crée un lien vers le module correspondant ; après avoir satisfait toutes les références externes de tous les modules il engendre un programme binaire qui sera la réunion de tous les modules initiaux.

Ce programme pourra ensuite être chargé en mémoire pour être exécuté sous le contrôle su SE.

Cet outil est très important en phase de développement de programmes car, d'une part il permet d'utiliser des fonctions dans des bibliothèques de programmes déjà existantes, et surtout, la modification d'un module lors de la phase de mise au point du programme entraîne seulement la recompilation de celui-ci et non celle de tout le programme.

Un module peut être amené à exécuter plusieurs fois la même séquence d'instructions. (Par exemple la lecture des valeurs de plusieurs matrices). On définit alors la notion de sous-programme, séquences d'instructions qui sera appelée depuis un module par une instruction particulière du type :

call nom_de_sous-programme

et qui se terminera par l'instruction :

return

pour continuer l'exécution du programme à l'instruction suivant l'instruction call.

Cette notion de sous-programme évite de dupliquer en plusieurs endroits du programme la même séquence d'instructions. Dans la mesure où cette séquence est locale à un module donné, il n'est pas judicieux d'en faire un module indépendant ; par contre, si cette fonction est utilisée depuis plusieurs modules alors, pour éviter de dupliquer ce code dans ces modules, il est intéressant d'en faire un module à part entière. Des modules indépendants peuvent alors être réunis dans un seul fichier pour former une bibliothèque de sous-programmes qui sera reliée tout entière avec les modules utilisant ses services.

b) le chargeur

Une fois le programme sous sa forme exécutable, il reste à l'amener en mémoire pour l'exécuter. Les adresses des instructions et des données sont calculées par le compilateur puis l'éditeur de liens à partir de l'adresse 0 du début du programme. Le mécanisme de gestion de la mémoire centrale et les procédures du SE associées devront transformer les adresses mémoires logiques contenues dans les instructions en adresses physiques émises par le processeur vers la mémoire.

Ces opérations sont fonctions de la politique de gestion de la mémoire centrale, implémentée par des programmes système qui gèrent la hiérarchie mémoire associée à un processeur particulier.

4.2 Les services de gestion des ressources de l'ordinateur

Cette deuxième classe de services correspond à des primitives systèmes qui vont gérer les accès à toutes les ressources physiques de l'ordinateur à partir des ordres donnés dans les programmes utilisateurs.

Par exemple, l'imprimante sera utilisée par tous les utilisateurs mais un seul à la fois évidemment. Les programmes devant faire une impression s'adresseront donc à un programme système qui gèrera toutes les requêtes et qui assurera la séquentialisation des différents accès à ce terminal.

Tous les éléments actifs d'un ordinateur sont donc ainsi gérés par des primitives systèmes : le processeur, la mémoire centrale, les périphériques d'entrées-sorties, les espaces disque etc....Ils constituent l'essentiel d'un système d'exploitation totalement invisible pour les programmeurs d'application et a fortiori des utilisateurs. Ces primitives sont généralement écrites en langage C et certaines en langage d'assemblage. Elles constituent le noyau du système d'exploitation.

Les mécanismes de gestion des ressources dans un système d’exploitation seront abordés dans ce cours.

BIBLIOGRAPHIE

A. TANENBAUM : "Architecture de l'ordinateur " 3ième Édition 1991

J. H HENNESY & D.A PATTERSON : " Computer Architecture :A quantitative approach" Morgan Kaufmann Second Edition 1995

RIFFLET J.M. : "La programmation sous Unix" McGraw-Hill

BOURNE S. : Le système Unix, Interédition

A. TANENBAUM : Les Systèmes d'exploitation des ordinateurs, Interédition

SILBERSCHATZ, GALVIN, GAGNE : "Operating System Concepts "7th edition", John Wiley & sons Inc. 2005