ARCHITECTURE DES SYSTÈMES INFORMATIQUES

Michel MEYNARD

3 Parties :
– Architecture ;
– Assembleur ;
– Systèmes d'exploitation.

Architecture :
analyse de la structure des ordinateurs et du logiciel de base.

Assembleur :
étude précise d'un langage d'assemblage : instructions, structures de contrôle et de données.

Systèmes d'exploitation :
systèmes de fichiers et gestion des processus.

1. Introduction

1.1 Informatique / ASI ?

Une définition de l'informatique :
science et **techniques** du traitement de l'information

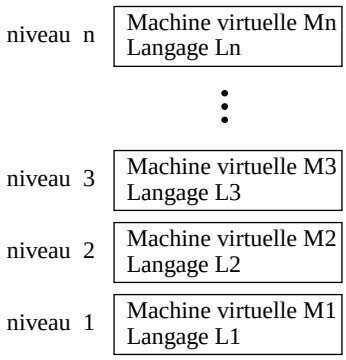
- **discipline scientifique :**
  - fortement liée au mathématiques historiquement et conceptuellement ;
  - théorie des graphes, modélisation, maths discrètes, algorithmique, calculabilité, complexité ;

- **discipline technologique :**
  - matériel : fortement liée à l'électronique, architecture des ordinateurs, évolution des composants (SSI → VLSI), multiprocesseurs ;
  - logiciel : "art de la programmation", ateliers de génie logiciel, programmation orientée objet, parallèle, systèmes d'exploitation.

Objectif du cours :
acquérir une vision cohérente de l'architecture matérielle et logicielle des « machines informatiques » traitant et stockant l'information.

1.2 Principe de décomposition

Architecture multi-niveaux (ou couches)
Afin d'améliorer les performances, et en raison des impératifs technologiques, le jeu d'instructions des machines réelles est limité et primitif. On construit donc au-dessus, une série de couches logicielles permettant à l'homme un dialogue plus aisé.



Les programmes en L_i sont :
- soit **traduits** (compilés) en L_{i-1} ou L_{i-2} ou … L_1,
- soit **interprétés** par un interpréteur tournant sur L_{i-1} ou L_{i-2} ou … L_1

1.2.1 Organisation multi-niveaux

Le principe conceptuel du découpage en plusieurs couches de complexités croissantes est omniprésent en informatique.

Exemples :

- Les systèmes informatiques (matériel + logiciel).

- La conception des systèmes d'exploitation (noyau et couches Unix)

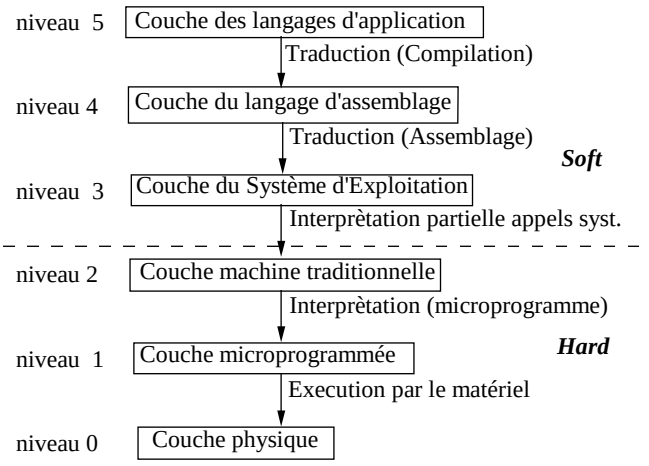
- Les réseaux informatiques : les 7 couches de la norme Open System Interconnection de l'ISO
- Les méthodes de conception de systèmes d'information (méthode Merise)

- Les compilateurs (code source, intermédiaire, objet)

- Les Types Abstraits de Données et les langages Orientés-Objet

- Les systèmes transactionnels multi-niveaux
- ...

1.2.2 Décomposition des SI



niveaux composant (électronique) puis atomique (physique solide)

Vocabulaire : niveau ou langage ou machine

- 0 portes logiques, circuits combinatoires, à mémoire ;
- 1 une instruction machine (code binaire) interprétée par son microprogramme ;
- 2 suite d'instructions machines du **jeu d'instructions**
- 3 niveau 2 + ensemble des services offerts par le S.E. **(appels systèmes)** ;
- 4 langage d'assemblage symbolique traduit en 3 par le programme assembleur ;
- 5 langages évolués (de haut niveau) traduits en 3 par compilateurs ou alors interprétés par des programmes de niveau 3.

1.3 Matériel et Logiciel

Matériel (Hardware)
Ensemble des composants mécaniques et électroniques de la machine : processeur(s), mémoires, périphériques, bus de liaison, alimentation…

Logiciel (Software)
Ensemble des programmes, de quelque niveau que ce soit, exécutables par un ou plusieurs niveaux de l'ordinateur. Un programme = mot d'un langage.
Le logiciel est immatériel même s'il peut être stocké physiquement sur des supports mémoires.

Matériel et Logiciel sont conceptuellement équivalents

Toute opération effectuée par logiciel peut l'être directement par matériel et toute instruction exécutée par matériel peut être simulée par logiciel.

Le choix est facteur du coût de réalisation, de la vitesse d'exécution, de la fiabilité requise, de l'évolution prévue (maintenance), du délai de réalisation du matériel…

Exemples de répartition matériel/logiciel

- premiers ordinateurs : multiplication, division, manip. de chaînes, commutation de processus … par logiciel : actuellement descendus au niveau matériel ;
- à l'inverse, l'apparition des processeurs micro-programmés à fait remonter d'un niveau les instructions machines ;
- les processeurs RISC à jeu d'instructions réduit ont également favorisé la migration vers le haut ;
- machines spécialisées (Lisp, bases de données) ;
- Conception Assistée par Ordinateur : prototypage de circuits électroniques par logiciel ;
- développement de logiciels destinés à une machine matérielle inexistante par simulation (contrainte économique fondamentale).

La frontière entre logiciel et matériel est très mouvante et dépend fortement de l'évolution technologique (de même pour les frontières entre niveaux).

A chaque niveau, le programmeur communique avec une **machine virtuelle** sans se soucier des niveaux inférieurs.

1.4 Plan du cours d'architecture

1. Introduction
2. Représentation des données
3. Structure des ordinateurs
4. La Couche Physique
5. La Couche Microprogrammée
6. La Couche Machine
7. Perspectives et Conclusion

Bibliographie

[Cazes 2003] Architecture des machines et des systèmes informatiques, Dunod

[Tanenbaum 1988] Architecture de l'ordinateur, InterEditions

[De Blasi 1990] Computer Architecture, Addison-Wesley

[Krakowiak 1982] Logiciel de base, Université de Grenoble

2. Représentation de l'information

2.1 Introduction

La technologie passée et actuelle a consacrée les circuits mémoires (électroniques et magnétiques) permettant de stocker des données sous forme **binaire**.

Remarque :
des chercheurs ont étudié et continuent d'étudier des circuits ternaires et même décimaux…

le bit :
abréviation de *binary digit*, le bit constitue la plus petite unité d'information et vaut soit 0, soit 1.

les bits sont généralement stockés séquentiellement et sont conventionnellement numérotés de la façon suivante :

b_{n-1}	b_{n-2}	…	b_2	b_1	b_0
1	0	…	0	1	1

On regroupe ces bits par paquets de n qu'on appelle des **quartets** (n=4), des **octets** (n=8) « *byte* », ou plus généralement des **mots** de n bits « *word* ».

Poids fort et faible
La longueur des mots étant la plupart du temps paire (n=2p), on parle de demi-mot de poids fort (ou le plus significatif) pour les p bits de gauche et de demi-mot de poids faible (ou le moins significatif) pour les p bits de droite.

Exemple : mot de 16 bits

b_{15} b_{14} … b_8	b_7 … b_0
octet le plus significatif	octet le moins signif.
Most Signifant Byte	*Least Significant Byte*

Unités multiples
Ces mots sont eux-mêmes groupés et on utilise fréquemment les unités multiples de l'octet suivantes :

1 Kilo-octet = 2^{10} octets = 1024 octets noté 1 **Ko**

1 Méga-octet = 2^{20} octets = 1 048 576 octets noté 1 **Mo**

1 Giga-octet = 2^{30} octets ≅ 10^9 octets noté 1 **Go**

1 Téra-octet = 2^{40} octets ≅ 10^{12} octets noté 1 **To**

2.2 Représentation des entiers positifs

2.2.1 Représentation en base 2

Un mot de n bits permet de représenter 2^n configurations différentes. En base 2, ces 2^n configurations sont associées aux entiers positifs x compris dans l'intervalle [0, 2^n-1] de la façon suivante :

$x = b_{n-1} * 2^{n-1} + b_{n-2} * 2^{n-2} + \ldots + b_1 * 2 + b_0$

Ainsi, un quartet permet de représenter l'intervalle [0, 15], un octet [0, 255], un mot de 16 bits [0, 65535].

Exemples :
00…0 représente 0
000…001 représente 1
0000 0111 représente 7 (4+2+1)
0110 0000 représente 96 (64+32)
1111 1110 représente 254 (128+64+32+16+8+4+2)
0000 0001 0000 0001 représente 257 (256+1)
100…00 représente 2^{n-1}
111…11 représente 2^n-1

Par la suite, cette convention sera notée Représentation Binaire Non Signée (RBNS).

2.2.2 Représentation en base 2^p

La représentation d'un entier x en base 2, $b_{n-1}b_{n-2}\ldots b_0$, est lourde en écriture. Aussi lui préfère-t-on une représentation plus compacte en base 2^p. On obtient cette représentation en découpant le mot $b_{n-1}b_{n-2}\ldots b_0$ en tranches de p bits à partir de la droite (la dernière tranche est complétée par des 0 à gauche si n n'est pas multiple de p). Chacune des tranches obtenues est la représentation en base 2 d'un chiffre de x représenté en base 2^p.

Usuellement, p=3 (représentation **octale**) ou p=4 (représentation **hexadécimale**).

En représentation hexadécimale, les valeurs 10 à 15 sont représentées par les symboles A à F. Généralement, on suffixe le nombre hexa par un H. De plus, on le préfixe par un 0 lorsque le symbole le plus à gauche est une lettre.

Exemples :
x=200 et n=8
en binaire : 11001 000 (128+64+8)
en octal : 3 1 0 (3*64+8)
en hexadécimal : C 8 (12*16+8) : 0C8H

2.2.3 Opérations

Les opérations arithmétiques +, -, /, * et de comparaison < , >, = sur les nombres binaires représentables sur machine sont des prolongements des opérations usuelles sur N ou sur Z. Cependant, des dépassements de capacité surviennent sur les intervalles considérés …

Addition binaire sur n bits
L'addition binaire de 2 mots de n bits est réalisée en ajoutant successivement de droite à gauche les bits de même poids des deux mots ainsi que la retenue éventuelle de l'addition précédente. En binaire non signé, la dernière retenue ou report (*carry)*, représente le coefficient de poids 2^n et est donc synonyme de dépassement de capacité. Cet indicateur de Carry est situé dans le registre d'état du processeur.

exemple sur 8 bits :

```
  1         1 1 1 1
    1 1 1 0 0 1 0 1    0 E5H
+   1 0 0 0 1 0 1 1    +8BH
---------------------------------- ---------
(1) 0 1 1 1 0 0 0 0      170H    368₁₀ > 255
```

Pour les autres opérations, on va tout d'abord définir une représentation des entiers négatifs.

2.3 Représentation des entiers relatifs

Quatre façons d'utiliser les nombres négatifs en machine ont été employées. Actuellement et en pratique, la méthode du "complément à 2" est la plus utilisée.

2.3.1 La valeur absolue signée

Dans cette représentation, le bit de poids n-1 indique le signe (0 → positif, 1 → négatif) tandis que les bits n-2 ... 0 représentent la valeur absolue de l'entier négatif dans la Représentation Binaire Non Signée (RBNS).

Intervalle de définition : $[-2^{n-1}+1, 2^{n-1}-1]$

Exemples sur un octet :

3	0000 0011	-3	1000 0011
127	0111 1111	-127	1111 1111
0	0000 0000	-0	1000 0000

Inconvénients :
- 2 représentations distinctes de 0
- opérations arithmétiques peu aisées :
  3 + -3 = -6 !

2.3.2 Le complément à 1 (C1)

(ou complément restreint)

Dans cette représentation, les entiers positifs sont en RBNS tandis qu'un négatif -|x| est obtenu par inversion de tous les bits de la RBNS de |x|. Ici encore, le bit de poids n-1 indique le signe (0 → positif, 1 → négatif).

Intervalle de définition : $[-2^{n-1}+1, 2^{n-1}-1]$

Exemples sur un octet :

3	0000 0011	-3	1111 1100
127	0111 1111	-127	1000 0000
0	0000 0000	0	1111 1111

Inconvénients :
• 2 représentations distinctes de 0
• opérations arithmétiques peu aisées : 3 + -3 = 0 (1111 1111) mais 4 + -3 = 0 (00…0) !

Le second problème est résolu si l'on ajoute 1 lorsqu'on additionne un positif et un négatif :
3+1+ -3=0 (00…0) et 4 +1+ -3=1 (00…01)

D'où l'idée de la représentation en Complément à 2.

2.3.3 Le complément à 2 (C2)

En C2 les entiers positifs sont en RBNS tandis que les négatifs sont obtenus par inversion de leur valeur absolue (C1) puis addition binaire de 1. Ici encore, le bit de poids n-1 indique le signe (0 → positif, 1 → négatif).
Une autre façon d'obtenir le C2 d'un entier relatif x consiste à écrire la RBNS de la somme de x et de 2^n.

Intervalle de définition : $[-2^{n-1}, 2^{n-1}-1]$

Exemples sur un octet [-128, +127] :

3	0000 0011	-3	1111 1101
127	0111 1111	-127	1000 0001
0	0000 0000	-128	1000 0000

Inconvénient :
• Intervalle des négatifs non symétrique des positifs ;
• Le C2 de -128 est -128 !

Avantage fondamental :
• l'addition binaire fonctionne correctement !
-3+3=0 : 1111 1101+0000 0011=(1) 0000 0000
-3 + -3 = -6 : 1111 1101+1111 1101=(1) 1111 1010

Remarquons ici que le positionnement du Carry à 1 n'indique pas un dépassement de capacité !

Dépassement de capacité en C2

127+127=-2 : 0111 1111+0111 1111=(0) 1111 1110
-128+-128=0 : 1000 0000+1000 0000=(1) 0000 0000
-127+-128=1 : 1000 0001+1000 0000=(1) 0000 0001

Lors de la 1° addition, la somme de deux grands positifs aboutit à un résultat négatif (retenue entrante sur le bit 7) et le Carry n'est pas positionné (pas de retenue sortante du bit 7). Dans les deux autres exemples, la somme de deux grands négatifs donne un positif (retenue sortante du bit 7, Carry=1, mais pas de retenue entrante).

La condition pour qu'une addition de deux entiers relatifs en C2 sur n bits soit correcte est que les retenues entrante et sortantes du bit n-1 soient identiques.

La plupart des processeurs possèdent un indicateur d'**Overflow** (dépassement de capacité) dans leur registre d'état qui permet au programmeur de tester l'incorrection d'une opération en C2.

Overflow = Carry XOR Retenue$_{n-2 \rightarrow n-1}$

Remarquons que l'addition d'un positif et d'un négatif ne peut jamais donner lieu à overflow.

2.3.4 L'excédent à 2^{n-1}

Dans cette représentation, l'ensemble des nombres représentables est le même qu'en C2. Tout nombre x de cet ensemble est représenté par $x+2^{n-1}$ en RBNS. Attention, le bit de poids n-1 indique le signe ($0 \rightarrow$ négatif, $1 \rightarrow$ positif).

L'excédent à 2^{n-1} se déduit du C2 en inversant le bit de signe !

Intervalle de définition : $[-2^{n-1}, 2^{n-1}-1]$

Exemples sur un octet :

3	1000 0011	-3	0111 1101
127	1111 1111	-127	0000 0001
0	1000 0000	-128	0000 0000

Avantage :
- représentation uniforme des entiers relatifs

Inconvénients :
- représentation des positifs différente de la RBNS
- opérations arithmétiques peu aisées 3+-3 =-128 !

2.3.5 Opérations en RBNS et C2

Le C2 étant la représentation la plus utilisée, nous allons étudier les opérations arithmétiques en non signé et en C2.

2.3.5.1 Addition

Rappelons qu'en RBNS et en C2, l'addition binaire (ADD réalisée par toutes les UAL de processeur) donne un résultat cohérent tant que celui-ci reste dans l'intervalle représenté ($[0, 2^n-1]$ et $[-2^{n-1}, 2^{n-1}-1]$)

En RBNS, le dépassement de capacité est signalé par le positionnement à 1 du Carry tandis qu'en C2, c'est l'indicateur d'Overflow qui est mis à 1.

2.3.5.2 Soustraction

La soustraction x-y pourrait être réalisée par inversion du signe de y puis addition avec x (sauf pour -128). L'inversion de signe (en C2) est généralement fournie par le matériel (NEG).

Exemple : R1 := 5 - 8
R1 := 5; R2 := 8; NEG R2; ADD R1,R2

L'instruction de soustraction SUB est généralement câblée par le matériel.

2.3.5.3 Multiplication et Division

La multiplication x*y peut être réalisée par y additions successives de x tandis que la division peut être obtenue par soustractions successives et incrémentation d'un compteur tant que le dividende est supérieur à 0 (pas efficace).

Cependant, la plupart des processeurs fournissent des instructions MUL et DIV directement câblées et efficaces.

Cas particulier
Lorsqu'une multiplication ou une division a comme opérande 2^n, on peut la simuler par décalage (*Shift*) à gauche ou à droite de n positions de l'autre opérande.

Exemples :
13*8 : 0000 1101 décalé 3 fois à gauche :
0110 1000 c'est-à-dire 104 ;
126/16 : 0111 1110 décalé 4 fois à droite :
0000 0111 c'est-à-dire 7.

En C2, il faut également réintroduire le bit de signe lors du décalage. Des instructions de décalage arithmétique sont généralement fournies par le processeur.

2.4 Représentation des décimaux

On s'intéresse ici à la catégorie des nombres décimaux, D, et non pas des réels, R. Les types de données **real** sont trompeurs : pour s'en convaincre, il suffit sur de nombreux calculateurs numériques non symboliques d'exécuter l'opération 1/3 puis de multiplier par 3 le résultat.

2.4.1 Décimal Codé Binaire *BCD*

Utilisé principalement en comptabilité pour sa précision, ce codage utilise un quartet pour chaque chiffre décimal :
$0 \rightarrow 0000$; $1 \rightarrow 0001$; $2 \rightarrow 0010$; …; $9 \rightarrow 1001$

Les quartets de code hexa 0AH à 0FH sont inutilisés. Chaque octet permet donc de stocker 100 combinaisons différentes représentant les entiers de 0 à 99.

Les opérations de l'arithmétique binaire sur ces entiers provoque des incohérences de représentation dans certaines situations :
33+58 : 0011 0011 + 0101 1000 = 1000 1011 =8BH

Aussi, les processeurs possèdent des instructions spécialisées d'ajustement des résultats à exécuter à la suite d'opérations binaires sur des opérandes DCB. Ici, il faut rajouter 06H au résultat !

Codage DCB des nombres à virgule

Une suite d'octets permet de coder un nombre décimal signé comme suit :
- 1 octet d'en-tête donne (en RBNS) le nombre total n de quartets utilisés.
- 1 octet spécifie la position (en RBNS) de la virgule.
- 1 quartet indique le signe du nombre (0H:+; 0FH:-).
- n quartets représentant les chiffres en DCB.

Exemple :
0000 0011 0000 0010 0000 0010 0010 0001
3 2 + 2 2 1

représente le nombre 2,21

Inconvénients
- format de longueur variable
- taille mémoire utilisée importante
- opérations arithmétique lentes : Ajustements nécessaires
- décalage des nombres nécessaires avant opérations pour faire coincider la virgule.

Avantage
- Résultats absolument corrects : pas d'erreurs de troncatures ou de précision d'où son utilisation en comptabilité.

2.4.2 Virgule flottante

On exprime généralement les nombres décimaux à l'aide de la notation scientifique en virgule flottante :

$$x=m*b^e$$

où m est la **mantisse**, b la **base** et e l'**exposant**.

Exemple :
$pi = 0{,}0314159*10^2 = 31{,}4159*10^{-1}=3{,}14159$

Il existe une représentation **normalisée** de la mantisse consistant à positionner un seul chiffre différent de 0 à gauche de la virgule et tous les autres à droite de la virgule. On obtient ainsi :
$b^0 \leq m < b^1$.

Exemple :
$pi = 3{,}14159*10^0$

On a la même forme de représentation en virgule flottante à mantisse normalisée pour le binaire (b=2)

Exemple : $7{,}25_{10} \rightarrow 111{,}01_2 \rightarrow 1{,}1101*2^2$
4+2+1+0,25=(1+0,5+0,25+0,0625)*4

Remarques :
- $2^0 = 1 \leq m < 2^1 = 2$
- Les puissances négatives de 2 sont : 0,5; 0,25; 0,125; 0,0625; 0,03125; 0,015625; 0,0078125; …
- La plupart des nombres à partie décimale finie n'ont pas de représentation binaire finie : (0,1; 0,2; …).
- Par contre, tous les nombres finis en virgule flottante en base 2 s'expriment de façon finie en décimal car 10 est multiple de 2.
- Réfléchir à la représentation en base 3.

Il faut bien comprendre que cette représentation binaire en virgule flottante, quel que soit le nombre de bits de mantisse et d'exposant, ne fait qu'approcher la plupart des nombres décimaux. Et cela, avant quelqu'opération que ce soit !

Algorithme de conversion de la partie décimale
On applique à la partie décimale des *multiplications successives* par 2, et on range, à chaque itération, la partie entière du produit dans le résultat.

Exemples :
0,375 à coder en binaire virgule flottante
0,375*2=**0**,75*2=**1**,5; 0,5*2=**1**,0 → 0,**011**

0,23*2=**0**,46*2=**0**,92*2=**1**,84*2=**1**,68*2=**1**,36*2=**0**,72*2=**1**,44*2=**0**,88……
$0{,}23_{10}$ sur 8 bits de mantisse : 0,**00111010**

2.4.3 Virgule Flottante en machine

Standardisation :
– portabilité entre machines, langages
– reproductibilité des calculs et « exactitude »
– communication de données via les réseaux
– représentation de nombres spéciaux
– procédures d'arrondi

La norme IEEE-754 (1985)
– simple précision sur 32 bits (float)
– double précision sur 64 bits (double)

Norme IEEE-754 flottants en **simple précision**
– signe : 1 bit (0 : +, 1 : -)
– exposant : 8 bits en excédent 127 [-127, 128]
– mantisse : 23 bits en RBNS ; normalisé sans représentation du 1 de gauche ! La mantisse est arrondie !

soit 4 octets ordonnés : signe, exposant, mantisse.

Valeur d'un float : $(-1)^s * 2^{(e-127)} * 1,m$

Exemple :
33,0 = +100001,0 = +1,00001 2^5
représenté par : 0 1000 0100 0000 100...
c'est-à-dire : 42 04 00 00H

Exemple :
-5,25 = -101,01 = -1,0101 2^2
représenté par : 1 1000 0001 0101 000...
c'est-à-dire : 0C0 A8 00 00H

Attention, certains codes spéciaux servent à représenter des nombres **exceptionnels** !
– 0 : e=0H et m=0H (s donne le signe)
– infini : e=0FFH et m=0H (s donne le signe)
– NaN (Not a Number) : e=0FFH et m qcq
– dénormalisés : e=0H et m≠0H ; dans ce cas, il n'y a plus de bit caché et la mantisse doit être décalée d'un cran à gauche : très petits nombres (de 2^{-149}=1.4 10^{-45} à 2^{-126}-2^{-149}=1.18 10^{-38})

En normalisé :
Plus grande valeur absolue représentable
MAX = $2^{127} * (2-2^{-23}) \approx 3.4\ 10^{38}$
PAS = $2^{127} * 2^{-23} \approx 2.03\ 10^{31}$
soit 7 chiffres significatifs

Plus petite valeur absolue représentable (sauf 0)
MIN = $2^{-126} * (1) \approx 1.18\ 10^{-38}$
PAS = $2^{-126} * 2^{-23} \approx 1.4\ 10^{-45}$
soit 7 chiffres significatifs

Quelques règles de calcul
-0=+0 ; 1/+0=+∞ ; 1/-0=-∞ ; x/∞=0 ;
NaN=∞/∞= 0/0=$(-|x|)^{1/n}$; x+∞=∞
règles d'arrondi extrêmement complexes ...

Remarques :

Il existe, entre deux nombres représentables, un intervalle de réels non exprimables. La taille de ces intervalles (pas) croît avec la valeur absolue :

proche et ≠ 0 : 2^{-149}=1.4 10^{-45}
proche de ∞ : 2^{124}=2.03 10^{31}

Cependant, si l'on exprime la distance entre un nombre et son "successeur" en pourcentage de ce nombre, ce pourcentage reste constant :

$2^{-149}/2^{-126} = 2^{-23} \approx$ **1,19 10^{-7}** $\approx 2^{104}/2^{127} = 2^{-23}$

Ainsi l'erreur relative due à l'arrondi de représentation est approximativement la même pour les petits nombres et les grands !

Le nombre de chiffres décimaux significatifs varie en fonction du nombre de bits de mantisse, tandis que l'étendue de l'intervalle représenté varie avec le nombre de bits d'exposant :

double
– 1 bit de signe
– 11 bits d'exposant
– 52 bits de mantisse (16 chiffres significatifs)

2.5 Représentation des caractères

Symboles
– alphabétiques,
– numériques,
– de ponctuation et autres (semi-graphiques, ctrl)

Utilisation
– entrées/sorties
– représentation externe (humaine) des programmes (sources) et données y compris les données numériques

Code ou jeu de car.
Ensemble de caractères associés **aux mots binaires** les représentant. La quasi-totalité des codes ont une taille fixe (7 ou 8 ou 16 bits) afin de faciliter les recherches dans les mémoires machines.

– ASCII (7) ;
– EBCDIC (8) ;
– ISO 8859-1 ou ISO Latin-1 (8) ;
– UniCode (16) ;
– UTF8 : 1 caractère codé sur 1 à 4 octets.

2.5.1 Jeux de caractères

2.5.1.1 Code ASCII

American Standard Code for Information Interchange

Ce très universel code à 7 bits fournit 128 caractères (0..127) divisé en 2 parties : 32 caractères de fonction (de contrôle) permettant de commander les périphériques (0..31) et 96 caractères imprimables (32..127).

Codes de contrôle importants :
0 Null <CTRL> <@>; 3 End Of Text <CTRL> <c>;
4 End Of Transmission <CTRL> <d>; 7 Bell
8 Backspace 10Line Feed 12 Form Feed
13 Carriage Return 27 Escape <CTRL> <z>

Codes imprimables importants :
20HEspace; 30H..39H "0".."9";41H..5AH"A".."Z"
61H..7AH"a".."z"

Le code ASCII est normalisé par le CCITT (Comité Consultatif International Télégraphique et Téléphonique) qui a prévu certaines variantes nationales : {} aux USA donnent éè en France …

La plupart du temps, l'unité mémoire étant l'octet, le 8° bit est utilisé :
- soit pour détecter les erreurs de transmission (bit de parité)
- soit pour définir des caractères spéciaux (semi-graphiques, accents) dans un code ASCII "étendu".

ASCII

Hexa MSD	0	1	2	3	4	5	6	7
LSD	Bin.	000	001	010	011	100	101	110	111
0	0000	NUL	DLE	espace	0	@	P	‘	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	^	n	~
F	1111	SI	US	/	?	O	_	o	DEL

ISO-8859-1

Hexa	LSD															
MSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

2.5.1.2 Code EBCDIC

Extended Binary Coded Decimal Interchange Code

Ce code à 8 bits d'IBM est également beaucoup utilisé sur les machines du constructeur. Cependant, certaines variantes existent en fonction du type de matériel. Des programmes de transcodage ASCII⇔EBCDIC existent sur la plupart des mini et gros ordinateurs.

2.5.1.3 ISO-8859-1 ou ISO Latin-1

Code 8 bits très utilisé en Unix, il permet de gérer les lettres accentuées, y compris majuscules, du français et d'autres langues latines.

2.5.1.4 Unicode

Code 16 bits supportant la plupart des langues écrites : anglais, français, chinois, japonais, ...

Les caractères de \u0020 (20H) à \u007E (7EH) correspondent à ceux de l'ASCII et de Latin-1

Les caractères de \u00A0 à \u00FF correspondent à ceux de Latin-1 sur [0A0H, 0FFH]

Ce jeu est utilisé pour le codage interne Java des char et des String. Il est difficilement utilisable car peu d'outils permettent de représenter l'intégralité des caractères (police de 2^{16} caractères).

2.5.2 Erreurs et compression

Dans la machine, l'information ne cesse de circuler, via les bus internes, entre les divers composants (U.C., M.C.). De plus, elle est également transmise aux périphériques, voire aux autres machines distantes à travers des réseaux de communication. Ces communications nombreuses ont deux inconvénients majeurs : premièrement, des erreurs surviennent pendant le transport en raison de parasites ou de pannes; deuxièmement, le temps de communication est généralement très grand devant les temps de traitement.

Pour éviter le premier écueil, des algorithmes de codage permettent la détection, et certains même, la correction des erreurs de transmission. Quant à la vitesse de communication, d'autres algorithmes dits de compression (ou compactage) réduisent la taille des données à transmettre. Le nombre d'algorithmes réalisant ces objectifs étant très importants, nous ne ferons qu'observer ceux qui sont parmi les plus simples et les plus répandus.

2.5.2.1 Contrôle de parité

Cette méthode simple permet uniquement de **détecter** une erreur dans un caractère. Elle consiste à rajouter un bit, dit de parité, au n bits du caractère à transmettre. En **parité paire**, le bit rajouté au caractère est calculé de façon à ce que le nombre total de bits à 1 dans le caractère, y compris le bit de parité, soit pair. En **parité impaire**, le nombre total de bits à 1, y compris le bit de parité, doit être impair. On rajoute généralement le bit de parité sur le poids le plus fort du caractère (en ASCII sur b_7).

Exemples :
en ASCII, le "A" se code 100 0001 soit 41H
en parité paire, "A" devient : **0**100 0001 soit 41H
en parité impaire, "A" devient : **1**100 0001 = 0C1H

en ASCII, le "z" se code 111 1010 soit 7AH
en parité paire, "z" devient : **1**111 1010 soit 0FAH
en parité impaire, "z" devient : **0**111 1010 = 7AH

Remarquons que l'altération d'un **unique** et **quelconque** bit durant la transmission est facilement détectée à l'arrivée. Cependant, on ne peut pas déterminer sa position. De plus, si un nombre pair de bits sont inversés, le contrôle de parité devient incohérent puisqu'il masque des erreurs !

2.5.2.2 Code de Hamming

Proposé par R. Hamming (1952), ce code autocorrecteur (détection et localisation de l'**unique** erreur) consiste à rajouter k bits de parité à chaque caractère codé sur n bits. Le nouveau mot ainsi constitué de n+k bits contient donc n bits d'information et k bits de contrôle positionnés à l'intérieur du mot sur les indices correspondant à des puissances de 2. Chaque bit de parité b_i contrôle ainsi les bits d'information dont les indices contiennent i en Binaire Non Signé.

Exemple : ASCII 7 bits + 4 bits de contrôle

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}
1	0	**0**	1	**1**	**0**	**0**	0	**0**	**0**	**0**

Ci-dessus le code de l'espace (20H) avec un code de Hamming en **parité paire** :
b_1 contrôle b_1, b_3, b_5, b_7, b_9, b_{11} : 1+0+1+0+0+0=2
b_2 contrôle b_2, b_3, b_6, b_7, b_{10}, b_{11} : 0+0+0+0+0+0=0
b_4 contrôle b_4, b_5, b_6, b_7 : 1+1+0+0=2
b_8 contrôle b_8, b_9, b_{10}, b_{11} : 0+0+0+0=0

Exemple du "b" (62H)

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}
0	0	**1**	1	**1**	**0**	**0**	1	**0**	**1**	**0**

Détection et correction
L'avantage du code de Hamming provient du fait que chaque bit d'information est contrôlé par au moins deux bits de parité : b_5 par b_4 et b_1, b_{11} par b_8, b_2 et b_1. Ainsi, lorsqu'un bit d'information est erroné, plusieurs bits de parité deviennent incohérent !

Exemple : "b" erroné sur le bit 7

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}
0	0	**1**	1	**1**	**0**	***1***	1	**0**	**1**	**0**

b_1 : parité incorrecte : 0+1+1+***1***+0+0=3 **impair**
b_2 : parité incorrecte : 0+1+0+***1***+1+0=3 **impair**
b_4 : parité incorrecte : 1+1+0+***1***=3 **impair**
b_8 : parité correcte : 1+0+1+0=2 pair

p_i=b_i est erroné
1=(p_1 xor p_3 xor p_5 xor p_7 xor p_9 xor p_{11}) and (p_2 xor p_3 xor p_6 xor p_7 xor p_{10} xor p_{11}) and (p_4 xor p_5 xor p_6 xor p_7) and not p_8 and not p_9 and not p_{10} and not p_{11} → p_7=1

Remarquons que le calcul du bit erroné revient à additionner tous les indices des bits de parité incorrects (ici 1+2+4=7) ! Ce code fonctionne également lorsque c'est un bit de contrôle qui a été inversé. Rappelons que le code de Hamming repose sur l'hypothèse fondamentale qu'un bit **au plus** est corrompu pendant la transmission !

2.5.2.3 Code de Huffman

Le codage de Huffman (1952) est une technique de **compression** de données permettant de stocker un message (un fichier) ou de le transmettre grâce à un minimum de bits afin d'améliorer les performances. Ce codage à **taille variable** suppose l'indépendance des caractères du message et on doit connaître la distribution probabiliste de ceux-ci à une position quelconque dans ce message. Plus la probabilité d'occurrence d'un caractère dans un fichier est grande, plus son code doit avoir une taille réduite.

On code chaque caractère par un mot de longueur variable de façon à ce qu'**aucun mot ne soit le préfixe d'un autre**. La propriété principale des codes de Huffman consiste en ce que la longueur moyenne du codage d'un caractère dans un fichier soit minimale.

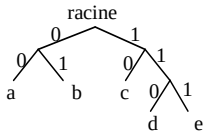
Pour calculer le code d'un alphabet et de sa distribution, on construit un arbre binaire étiqueté par des 0 et des 1, les feuilles de cet arbre représentant les caractères tandis que les chemins issus de la racine constituent les codes correspondants.

Exemple :

Soit l'alphabet {a,b,c,d,e} associé à la distribution probabiliste suivante :

Car.	a	b	c	d	e
Proba.	0,3	0,25	0,20	0,15	0,10

arbre binaire :



code de Huffman correspondant :

Car.	a	b	c	d	e
Code	00	01	10	110	111

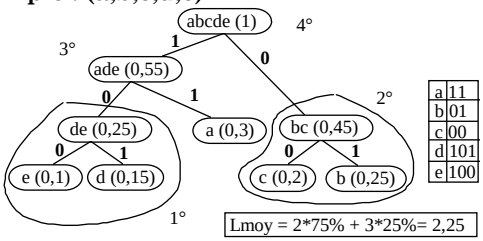
Remarquons que ce code n'est pas unique (il suffit de permuter les 0 et les 1). On calcule la longueur moyenne de codage comme suit :

75% des caractères nécessitent 2 bits (a,b,c)
25% des caractères nécessitent 3 bits (d,e)
L_{moy}=75%*2 + 25%*3 = 2,25 bits

Remarquons qu'avec un code de taille fixe, chaque caractère aurait nécessité 3 bits, soit une perte de 33 %.

Algorithme de Huffman
L'algorithme consiste à construire l'arbre en partant des deux feuilles "les plus basses" (plus faibles probabilités *ici d et e*). On leur associe un père, sorte de nœud virtuel dont la probabilité d'apparition du préfixe associé est égale à la somme des probabilités de ses deux fils. On réitère le processus en choisissant à nouveau les deux sommets sans père de plus faible probabilité jusqu'à la construction de la racine.

Exemple : (a,b,c,d,e)



Remarque
Lors de transmission de données, le message reçu est analysé bit par bit jusqu'à ce qu'on reconnaisse une configuration valable. Ce codage est très sensible aux erreurs (un bit altéré interdit la compréhension de la fin du message) ! Par conséquent, il est fréquent de découper le message en mots de n bits et d'y associer un code autocorrecteur (Hamming).

2.6 Représentation des images et des sons

Les images et les sons sont souvent codés sur des supports **analogiques** (continu). Il faut les **numériser** (digitaliser, discrétiser) pour les stocker sur des supports numériques.

On **échantillonne** l'image ou le son en le représentant par une suite finie de points : plus la taille de l'échantillon est grande, plus précise est la conversion analogique-numérique.

La conversion numérique-analogique inverse est bien sur possible : modem (**mo**dulateur-**dém**odulateur)

Les **images** peuvent être représentées dans divers formats :
– bitmap ou pixmap : chaque point de l'image est représentée par un bit (0 : blanc, 1 : noir) ou par un mot (1 couleur parmi 2^n)
– vectoriels : une figure géométrique (rectangle, cercle, ...)

Ces formats donnent lieu à de nombreux codes alliant souvent **formatage** et **compression**
bitmap : MS bitmap (.bmp), GIF (.gif), jpeg (.jpg), ..
vectoriel : DXF autocad (.dxf), EPS postscript (.eps)
son : MP3, OGG Vorbis, …

2.7 Structures de données

A partir des représentations élémentaires précédentes (nombres et caractères), on peut construire des structures de données beaucoup plus complexes pouvant être homogènes (tableaux, liste, pile…) ou hétérogènes (articles).

Deux méthodes de rangement existent pour les structures de données de **taille variable**. Soit la taille de la structure est conservée explicitement et dynamiquement (fichiers, chaînes pascal), soit une marque de fin de structure indique la terminaison (lignes d'un fichier texte (CR-LF MS-DOS, LF UNIX), chaînes C).

Exemple :

Chaîne pascal

17	e	x	e	m	p	l	e		d	e		c	h	a	î	n	e

Avantage : taille connue permettant des calculs rapides (accès i^{ème}, concaténation, …).
Inconvénient : taille maximum fixe= 255 octets.

Chaîne C

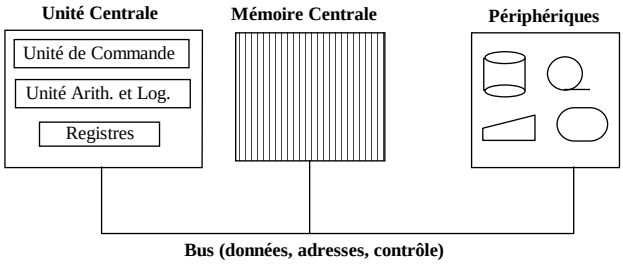
e	x	e	m	p	l	e		d	e		c	h	a	î	n	e	\0

Avantage : taille limitée seulement par la mémoire.
Inconvénient : parcours longs…

3. Structure des ordinateurs

Le modèle d'architecture de la plupart des ordinateurs actuels provient d'un travail effectué par John **Von Neumann** en 1946.

Le modèle de Von Neumann



Principes du modèle de programmation :
- code = **séquence** d'instructions en MC ;
- données stockées en MC ;

Actuellement, d'autres types d'architecture (5° génération, machines systoliques, …) utilisant massivement le **parallélisme** permettent d'améliorer notablement la vitesse des calculs.

On peut conjecturer que dans l'avenir, d'autres paradigmes de programmation **spécifiques** à certaines applications induiront de nouvelles architectures.

3.1 L' Unité Centrale (UC)

ou processeur (*Central Processing Unit CPU)*

Cerveau de l'ordinateur, l'UC exécute séquentiellement les instructions stockées en Mémoire Centrale. Le traitement d'une instruction se décompose en 3 temps : **chargement, décodage, exécution**. C'est l'unité de commande qui ordonnance l'ensemble, tandis que l'UAL exécute des opérations telles que l'addition, la rotation, la conjonction…, dont les paramètres et résultats sont stockés dans les registres (mémoires rapides).

3.1.1.1 Les registres

Certains registres spécialisés jouent un rôle particulièrement important. Le Compteur Ordinal (CO) *Instruction Pointer IP, Program Counter PC* pointe sur la prochaine instruction à exécuter; le Registre Instruction (RI) contient l'instruction en cours d'exécution; le registre d'état *Status Register, Flags, Program Status Word PSW* contient un certain nombre d'indicateurs (ou drapeaux ou bits) permettant de connaître et de contrôler certains états du processeur; Le pointeur de pile *Stack Pointer* permet de mémoriser l'adresse en MC du sommet de pile (structure de données *Last In First Out LIFO* indispensable pour les appels procéduraux); Des registres d'adresse (index ou bases) permettent de stocker les adresses des données en mémoire centrale tandis que des registres de travail permettent de stocker les paramètres et résultats de calculs.

3.1.1.2 L'Unité de Commande

Celle-ci exécute l'algorithme suivant :

répéter
1. **charger** dans RI l'instruction stockée en MC à l'adresse pointée par le CO;
2. CO:=CO+taille(instruction en RI);
3. **décoder** (RI) en micro-instructions;
4. (localiser en mémoire les données de l'instruction;)
5. (charger les données;)
6. **exécuter** l'instruction (suite de micro-instructions);
7. (stocker les résultats mémoires;)

jusqu'à l'infini

Lors du démarrage de la machine, CO est initialisé soit à l'adresse mémoire 0 soit à l'adresse correspondant à la fin de la mémoire (2^m-1). A cette adresse, se trouve le moniteur en mémoire morte qui tente de charger l'amorce "*boot-strap*" du système d'exploitation.

Remarquons que cet algorithme peut parfaitement être simulé par un logiciel (interprèteur). Ceci permet de tester des processeurs matériels avant même qu'il en soit sorti un prototype, ou bien de simuler une machine X sur une machine Y (émulation).

3.1.1.3 L'Unité Arith. et Log.

Celle-ci exécute des opérations :
- **arithmétiques** : addition, soustraction, C_2, incrémentation, décrémentation, multiplication, division, décalages arithmétiques (multiplication ou division par 2^n).
- **logiques** : et, ou, xor, non, rotations et décalages.

Selon le processeur, certaines de ces opérations sont présentes ou non. De plus, les opérations arithmétiques existent parfois pour plusieurs types de nombres (C2, DCB, virgule flottante) ou bien des opérations d'ajustement permettent de les réaliser.

Enfin sur certaines machines (8086) ne possédant pas d'opérations en virgule flottante, des co-processeurs arithmétiques (8087) peuvent être adjoint pour les réaliser.

Les opérations arithmétiques et logiques positionnent certains indicateurs d'état du registre PSW. C'est en testant ces indicateurs que des branchements conditionnels peuvent être exécutés vers certaines parties de programme.

Pour accélérer les calculs, on a intérêt à utiliser les registres de travail comme paramètres, notamment l'accumulateur (AX pour le 8086).

3.2 La Mémoire Centrale (MC)

La mémoire centrale de l'ordinateur est habituellement constituée d'un ensemble ordonné de 2^m cellules (cases), chaque cellule contenant un mot de n bits. Ces mots permettent de conserver programmes et données ainsi que la pile d'exécution.

3.2.1 Accès à la MC

La MC est une mémoire électronique et l'on accède à n'importe laquelle de ses cellules au moyen de son adresse comprise dans l'intervalle [0, 2^m-1]. Les deux types d'accès à la mémoire sont :
- la **lecture** qui transfère sur le bus de données, le mot contenu dans la cellule dont l'adresse est située sur le bus d'adresse.
- l'**écriture** qui transfère dans la cellule dont l'adresse est sur le bus d'adresse, le mot contenu sur le bus de données.

La taille n des cellules mémoires ainsi que la taille m de l'espace d'adressage sont des caractéristiques fondamentales de la machine. Le mot de n bits est la plus petite unité d'information transférable entre la MC et les autres composants. Généralement, les cellules contiennent des mots de 8, 16 ou 32 bits.

3.2.2 Contenu/adresse (valeur/nom)

Attention à ne jamais confondre le contenu d'une cellule, mot de n bits, et l'adresse de celle-ci, mot de m bits même lorsque n=m.

Parfois, le bus de données a une taille multiple de n ce qui permet la lecture ou l'écriture de plusieurs mots consécutifs en mémoire. Par exemple, le microprocesseur 8086 permet des échanges d'octets ou de mots de 16 bits (appelés "mots").

Exemple (cellules d'un octet)

Adresse	Contenu bin								hexa.
0	0	1	0	1	0	0	1	1	53H
1	1	1	1	1	1	0	1	0	0FAH
2	1	0	0	0	0	0	0	0	80H
…									
32	0	0	1	0	0	0	0	0	20H (32_{10})
…									
2^m-1	0	0	0	1	1	1	1	1	1FH

Parfois, une autre représentation graphique de l'espace mémoire est utilisé, en inversant l'ordre des adresses : adresses de poids faible en bas, adresses fortes en haut. Cependant, pour le 8086, lorsqu'on range un mot (16 bits) à l'adresse mémoire i, l'octet de poids fort se retrouve en i+1. Il est aisé de s'en souvenir mnémotechniquement via la gravité dans les liquides de densités différentes.

3.2.3 RAM et ROM

Random Access Memory/ Read Only Memory

La RAM est un type de mémoire électronique **volatile** et **réinscriptible**. Elle est aussi nommée mémoire vive et plusieurs technologies permettent d'en construire différents sous-types : **statique**, **dynamique** (rafraîchissement). La RAM constitue la majeure partie de l'espace mémoire puisqu'elle est destinée à recevoir programme, données et pile d'exécution.

La ROM est un type de mémoire électronique **non volatile** et **non réinscriptible**. Elle est aussi nommée mémoire morte et plusieurs technologies permettent d'en construire différents sous-types (ROM, PROM, EPROM, EEPROM (Flash), …). La ROM constitue une faible partie de l'espace mémoire puisqu'elle ne contient que le moniteur réalisant le chargement du système d'exploitation et les Entrées/Sorties de plus bas niveau. Sur les PCs ce moniteur s'appelle le *Basic Input Output System*. Sur les Macintosh, le moniteur contient également les routines graphiques de base. C'est toujours sur une adresse ROM que le Compteur Ordinal pointe lors du démarrage machine.

DDR SDRAM : Double Data Rate Synchronous Dynamic RAM est une RAM dynamique (condensateur) qui a un *pipeline* interne permettant de synchroniser les opérations R/W.

3.3 Les périphériques

Les périphériques, ou organes d'Entrée/Sortie (E/S) *Input/Output (I/O)*, permettent à l'ordinateur de **communiquer** avec l'homme ou d'autres machines, et de **mémoriser massivement** les données ou programmes dans des fichiers. La caractéristique essentielle des périphériques est leur **lenteur** :
- Processeur cadencé en Giga-Hertz : instructions éxécutées chaque nano-seconde (10^{-9} s) ;
- Disque dur de temps d'accès entre 10 et 20 ms (10^{-3} s) : rapport de 10^7 !
- Clavier avec frappe à 10 octets par seconde : rapport de 10^8 !

3.3.1 Communication

L'ordinateur échange des informations avec l'**homme** à travers des terminaux de communication homme/machine : clavier ←, écran →, souris ←, imprimante →, synthétiseur (vocal) →, table à digitaliser ←, scanner ←, crayon optique ←, lecteur de codes-barres ←, lecteur de cartes magnétiques ←, terminaux consoles stations ↔ …
Il communique avec d'autres machines par l'intermédiaire de **réseaux** ↔ locaux ou longue distance (via un modem).

3.3.2 Mémorisation de masse
ou mémorisation secondaire

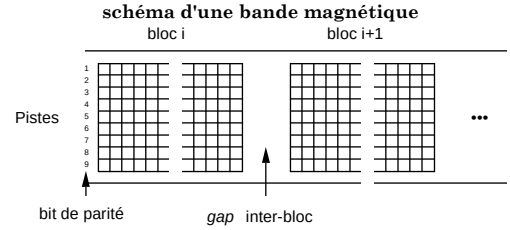
Les mémoires électroniques étant chères et soit volatiles (non fiables) soit non réinscriptibles, le stockage de masse est réalisé sur d'autres supports. Ces autres supports sont caractérisés par :
- non volatilité et réinscriptibilité
- faible prix de l'octet stocké
- lenteur d'accès et modes d'accès (séquentiel, séquentiel indexé, aléatoire, …)
- forte densité
- parfois amovibilité
- *Mean Time Between Failures* plus important car organes mécaniques → **stratégie de sauvegarde**

Supports Optiques
Historiquement, les **cartes 80 colonnes** ont été parmi les premiers supports mais sont complètement abandonnées aujourd'hui. Les **rubans perforés**, utilisés dans les milieux à risque de champ magnétique (Machines Outils à Commande Numérique), sont leurs descendants directs dans le cadre des supports optiques.

Supports Magnétique
Aujourd'hui (années 90), les supports magnétiques constituent la quasi-totalité des mémoires de masse des ordinateurs à usage général.

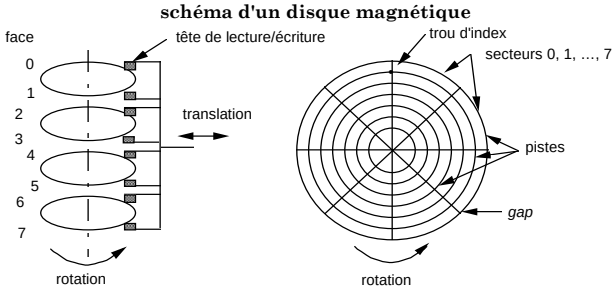
Bandes magnétiques
Ce sont des supports à accès **séquentiel** particulièrement utilisés dans la sauvegarde. On les utilise de plus en plus sous forme de **cassettes**. Les dérouleurs de cassettes sont appelés *streamers*. Les densités et vitesses sont variables : quelques milliers de *bytes per inch (bpi)* et autour d'un Mo/s. Le temps d'accès dépendant de la longueur de la bande …

schéma d'une bande magnétique



Disques magnétiques
Ils constituent la majorité des mémoires secondaires. Durs ou souples, fixes ou amovibles, solidaires (Winchester) ou non (dispack) de leurs têtes de lecture/écriture, il en existe une très grande diversité. Les disques durs ont des capacités variant entre quelques dizaines à quelques centaines de Mo, des vitesses de transfert autour du Mo/s et des temps d'accès approchant les 10 ms.

schéma d'un disque magnétique



Composé de faces, de pistes concentriques, de secteurs *"soft sectored"*, la densité des disques est souvent caractérisée par le nombre de *"tracks per inch" (tpi)* . Sur les disques durs, un cylindre est constitué d'un ensemble de pistes de même diamètre.
Un contrôleur de disque (carte) est chargé de transférer les informations entre un ou plusieurs secteurs et la MC. Pour cela, il faut lui fournir : le sens du transfert *(R/W)*, l'adresse de début en MC (Tampon), la taille du transfert, la liste des adresses secteurs (n° face, n° cylindre, n° secteur). La plus petite **unité de transfert** physique est **1 secteur**.

Pour accéder à un secteur donné, le contrôleur doit commencer par translater les bras mobiles portes-têtes sur le bon cylindre, puis attendre que le bon secteur passe sous la tête sélectionnée pour démarrer le transfert. Le **temps d'accès** moyen caractérise la somme de ces deux délais moyens.

Exemple : TA_{moyen} et transfert d'1 secteur ?
disque dur 8 faces, 50 cylindres, 10 secteurs/piste d'1 Ko, tournant à 3600 tours/mn, ayant une vitesse de translation de 1 m/s et une distance entre la 1° et la dernière piste de 5 cm.

TA_{moyen} = (5 cm/2)/1 m/s + (1/60 t/s)/2= 25ms+8,3 ms
Transfert d'1 secteur = (1/60)/10 = 1,66 ms

Supports optiques

Supports optiques

- unités de lecture fonctionnant au moyen d'un faisceau laser ;
- densités de stockage supérieures au magnétique : de 10^2 à 10^4 fois plus ;
- temps d'accès plus longs : archivage de masse.

CDROM : disques compacts (même format que les CDs audio) pré-enregistrés par pressage en usine et non réinscriptibles : (logiciels, annuaires, encyclopédies, …).

CDR : inscriptibles une seule fois ;

CDRW : réinscriptibles (1000 fois)

DVDR, DVDRW : idem CD mais avec des capacités plus importantes : 4,7 Go contre 700 Mo, double couches ...

Magnéto-optiques : combinant la technologie optique (laser) et magnétique (particules orientées), ils sont réinscriptibles. Amovibles, plus denses que les disques magnétiques mais moins rapides, ils constituent un compromis pour les archivages et les fichiers rarement accédés.

3.4 Contrôleur d'E/S et IT

A l'origine, l'UC gérait les périphériques en leur envoyant une requête puis en attendant leur réponse. Cette **attente active** était supportable en environnement monoprogrammé.

Actuellement, l'UC délègue la gestion des E/S aux processeurs situés sur les cartes contrôleur (disque, graphique, …) :

1. l'UC transmet la requête d'un processus à la carte contrôleur ;
2. l'UC « endort » le processus courant et exécute un processus « prêt » ;
3. le contrôleur exécute l'E/S ;
4. le contrôleur prévient l'UC de la fin de l'E/S grâce au mécanisme **d'interruption** ;
5. l'UC désactive le processus en cours d'exécution puis « réveille » le processus endormi qui peut continuer à s'exécuter.

Grâce à ce fonctionnement, l'UC ne perd pas son temps à des tâches subalternes !

Généralement, plusieurs **niveaux d'interruption** plus ou moins prioritaires sont admis par l'UC

3.5 Le(s) Bus

Le **bus de données** est constitué d'un ensemble de lignes bidirectionnelles sur lesquelles transitent les bits des données lues ou écrites par le processeur. (Data 0-31)

Le **bus d'adresses** est constitué d'un ensemble de lignes unidirectionnelles sur lesquelles le processeur inscrit les bits formant l'adresse désirée.

Le **bus de contrôle** est constitué d'un ensemble de lignes permettant au processeur de signaler certains événements et d'en recevoir d'autres. On trouve fréquemment des lignes représentant les **signaux** suivants :

V_{cc} et GROUND : tensions de référence ←
$\overline{\text{RESET}}$: réinitialisation de l'UC ←
R/$\overline{\text{W}}$: indique le sens du transfert →
MEM/$\overline{\text{IO}}$: adresse mémoire ou E/S →

Technologiquement, les bus de PC évoluent rapidement (Vesa Local Bus, ISA, PCI, PCI Express, ATA, SATA, SCSI, …)

3.6 Améliorer les performances

3.6.1 Hiérarchie mémoire et Cache

Classiquement, il existe 3 niveaux de mémoire ordonnés par vitesse d'accès et prix décroissant et par taille croissante :
- Registres ;
- Mémoire Centrale ;
- Mémoire Secondaire.

Afin d'accélérer les échanges, on peut augmenter le nombre des niveaux de mémoire en introduisant des **CACHE** ou antémémoire de Mémoire Centrale et/ou Secondaire : Mémoire plus rapide mais plus petite, contenant une copie de certaines données :

Fonctionnement :

Le demandeur demande à lire ou écrire une information ;
si le cache possède l'information, l'opération est réalisée, sinon, il récupère l'info. depuis le fournisseur puis réalise l'op.

Le principe de **séquentialité** des instructions et des structures de données permet d'optimiser le chargement du cache avec des segments de la mémoire fournisseur. Stratégie de remplacement est généralement LRU (Moins Récemment Utilisée).

3.6.1.1 Niveaux et localisation des caches

- **Cache Processeur réalisé en SRAM**

  - Niveau 1 (L1) : séparé en 2 caches (instructions, données), situé dans le processeur, communique avec L2 ;
  - Niveau 2 (L2) : unique (instructions et données) situé dans le processeur ;
  - Niveau 3 (L3) : existe parfois sur certaines cartes mères.

Par exemple, Pentium 4 ayant un cache L2 de 256 Ko.

- **Cache Disque**

De quelques Méga-octets, ce cache réalisé en DRAM est géré par le processeur du contrôleur disque. Il ne doit pas être confondu avec les tampons systèmes stockés en mémoire centrale (100 Mo). Intérêts de ce cache :

- Lecture en avant (arrière) du cylindre ;
- Synchronisation avec l'interface E/S (IDE, SATA, …)
- Mise en attente des commandes (SCSI, SATA)

3.6.2 Pipeline

Technique de conception de processeur avec plusieurs petites unités de commande placées en série et dédiées à la réalisation d'une tâche spécifique. Plusieurs instructions se chevauchent à l'intérieur même du processeur.

Par exemple, décomposition simple d'une instruction en 4 étapes :

6. Fetch : chargement de l'instruction depuis la MC ;
7. Decode : décodage en micro-instructions ;
8. Exec : exécution de l'instruction (UAL) ;
9. Write Back : écriture du résultat en MC ou dans un registre

Soit la séquence d'instruction : i1, i2, i3, ...

sans pipeline :

i1F, i1D, i1E, i1W, i2F, i2D, i2E, i2W, i3F, ...

avec pipeline à 4 étages

i1F	i1D	i1E	i1W	
	i2F	i2D	i2E	I2W
		i3F	i3D	I3E
			i4F	i4D

Chaque étage du pipeline travaille « à la chaîne » en répétant la même tâche sur la série d'instructions qui arrive.

Si la séquence est respectée, et s'il n'y a pas de conflit, le débit d'instructions (*throughput*) est multiplié par le nombre d'étages !

Problèmes et solutions :
- Rupture de séquence : vidage des étages !
- Dépendance d'instructions : mise en attente forcée

Architecture superscalaire :
Plusieurs pipeline (2) dans le même processeur travaillent en parallèle (→instons indépendantes).

Exemple :
Pentium 4 : selon ses versions, de 20 à 31 étages !

3.6.3 SIMD

Single Instruction Multiple Data désigne un ensemble d'instructions vectorielles permettant des opérations scientifiques ou multimédia. Par exemple, l'AMD 64 possède 8 registres 128 bits et des instructions spécifiques utilisables pour le streaming, l'encodage audio ou vidéo, le calcul scientifique.

3.6.4 DMA et BUS Mastering

L'accès direct mémoire ou DMA (Direct Memory Access) est un procédé informatique où des données circulant de ou vers un périphérique (port de communication, disque dur) sont transférées directement par un contrôleur adapté vers la mémoire centrale de la machine, sans intervention du microprocesseur si ce n'est pour initier et conclure le transfert. La conclusion du transfert ou la disponibilité du périphérique peuvent être signalés par interruption.

La technique de Bus Mastering permet à n'importe quel contrôleur de demander et prendre le contrôle du bus : le maître peut alors communiquer avec n'importe lequel des autres contrôleurs sans passer par l'UC.
Cette technique implémentée dans le bus PCI permet à n'importe quel contrôleur de réaliser un DMA. Si l'UC a besoin d'accéder à la mémoire, elle devra attendre de récupérer la maîtrise du bus. Le contrôleur maître lui vole alors des cycles mémoires.

4. La Couche Physique

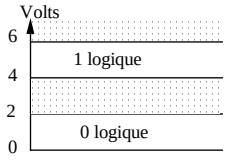
La couche physique est constituée de circuits électroniques complexes interconnectés. Ces circuits sont conçus à partir d'un nombre peu importants de **circuits de base** relativement simples. Seuls quelques uns de ces circuits seront décrits fonctionnellement ici.

4.1 Portes logiques et Algèbre booléenne

Nous ne nous intéresserons qu'aux caractéristiques essentielles des circuits et non à l'implantation électronique des réseaux de transistors sur les supports semi-conducteurs !

Une porte logique *gate* a un comportement binaire, "il faut qu'une porte soit ouverte ou bien fermée", et on lui associera la valeur 0 ou 1 selon le potentiel de sa sortie.

Exemple :



Remarque
Ce découpage des tensions de sortie des circuits en **deux plages** représentant le 0 et le 1 est tout à fait arbitraire et on a déjà envisagé d'autres découpages en 3, 4 et même 10 plages de potentiels. Avec 10 plages représentant les chiffres de 0 à 9, on obtient une machine travaillant directement en décimal ! Cependant ces architectures restent du domaine de la recherche en raison :
1. de leur **coût** : il faut développer tous les circuits composant la machine …
2. de leur **fiabilité** plus faible : les 10 plages étant plus "minces" et plus proches les unes des autres, des parasites peuvent plus facilement provoquer des erreurs.

Technologie
2 familles de transistors sont utilisés :
- les transistors à jonctions (techno. bipolaire) utilisés en TTL (Transistor Transistor Logic) et en ECL (Emitter Coupled Logic).
  - TTL rapide, consommation élevée
  - ECL très rapide 10*TTL, consommation élevée
- Les transistors à effet de champ (techno. unipolaire) utilisés en PMOS, **CMOS**, NMOS …
  - MOS lent TTL/10, consommation faible (portables)
  - HMOS, XMOS rapide 1*TTL, consommation faible

4.2 Algèbre booléenne

Variable booléenne
– un nom x, y, z
– 2 valeurs possibles vrai (1), faux (0)

Opérations booléennes
– non noté $\bar{x}$

x	0	1
$\bar{x}$	1	0

– et noté x.y ou plus simplement xy

x\y	0	1
0	0	0
1	0	1

– ou inclusif noté x+y

x\y	0	1
0	0	1
1	1	1

– ou exclusif noté x ⊕ y

x\y	0	1
0	0	1
1	1	0

– équivalence noté x ≡ y

x\y	0	1
0	1	0
1	0	1

4.3 Propriétés des opérations

– **commutativités** du et, du ou :
xy=yx x+y=y+x

– **associativités** du et, du ou :
x(yz)=(xy)z=xyz x+(y+z)=(x+y)+z=x+y+z

– **distributivités**
x(y+z)=xy+xz avec la convention . prioritaire sur +
x+(yz)=(x+y)(x+z)

– lois de **Morgan**
$\overline{(x+y)} = \bar{x} \,.\, \bar{y}$ $\overline{(x.y)} = \bar{x} + \bar{y}$

– **double négation** : $\bar{\bar{x}} = x$

– **absorption** : x.(x+y)=x=x+(x.y)

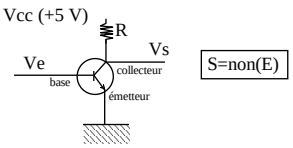
– **réduction** : (x.y)+($\bar{x}$.y)=y=(x+y).($\bar{x}$+y)

– **complétude** du {non, et} ou du {non, ou}
toute fonction logique n-aire peut être réalisée par des combinaisons de ces 2 opérations.

Exemple : implication binaire, ou ternaire ...

4.4 Circuits logiques de base

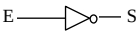
4.4.1 L'inverseur *not*



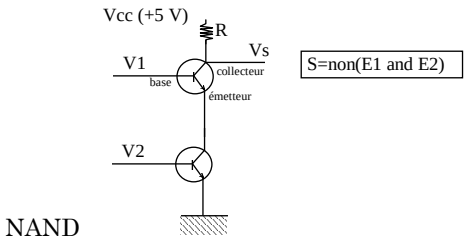
Lorsque V_e est inférieur à la valeur critique du transistor, celui-ci est bloqué et est équivalent à un interrupteur **ouvert** : V_s est donc proche de V_{cc}. Lorsque V_e est supérieur à la tension critique, le transistor bascule (la porte se **ferme**) et équivaut à une résistance quasi-nulle : V_s est donc proche de 0.

Remarquons que l'inversion de l'entrée sur la sortie n'est pas instantanée : selon les technologies d'intégration, elle tourne autour des **quelques nano-secondes** (10^{-9} s).

On schématise une porte non *not* (inverseur) de la façon suivante :



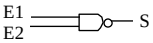
4.4.2 Portes non-et, non-ou *nand, nor*



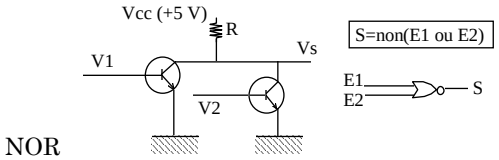
NAND

Si $V_1 < V_{critique}$ ou $V_2 < V_{critique}$ alors $V_s=V_{cc}$
sinon $V_s=0$

On schématise une porte non-et *nand* de la façon suivante :



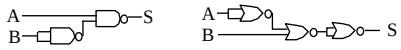
En plaçant deux transistors en série, on obtient une porte **nor** schématisée de la façon suivante :



NOR

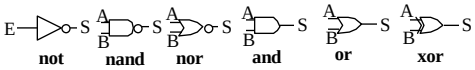
Complétude
Les portes NAND et NOR sont dites **complètes** car on peut câbler avec l'une ou l'autre n'importe quelle fonction booléenne n-aire.

Exemple : Implication (A implique B)
nonA ou B =non(A et nonB)=non(non(nonA ou B))



Rappel des lois de Morgan :
non(A ou B)=nonA et non B
non(A et B)=nonA ou non B

Schémas :
On schématise de la façon suivante les circuits logiques usuels réalisés à partir des portes de base :



4.4.3 Composants de base

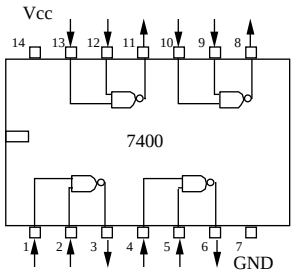
Les circuits intégrés logiques (puce, *chip*) permettent d'utiliser plusieurs fonctions logiques sur une plaquette de silicium (5mm*5mm). Différents procédés technologiques permettent cette intégration à plus ou moins forte densité. La densité est calculée en nombre de portes ou de transistors par mm^2 ou par circuit. Ensuite,

chaque plaquette est encapsulée dans un boîtier (noir) rectangulaire en plastique ou en céramique, d'où sortent des broches (pattes) de connexion.

Il existe 4 classes de produits classées selon leur densité d'intégration :

SSI (Small Scale Integration) 1 à 10 portes/circuit
MSI (Medium " " ") 10 à 100 " "
LSI (Large " " ") 100 à 100 000 "
VLSI (Very Large " ") plus de 100 000 "

Exemple : SSI 7400 TTL (V_{cc}=5V) Texas Instr.



Ce circuit très utilisé permet de câbler 4 nands binaires.

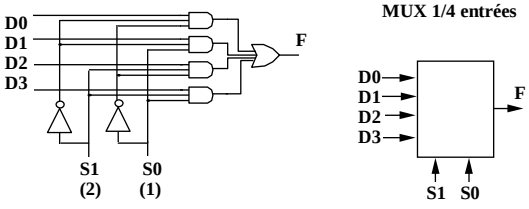
4.5 Circuits logiques combinatoires

Ces circuits plus ou moins complexes sont caractérisés par le fait que leur sortie s'exprime **uniquement** par une fonction logique de leurs entrées. Ils sont différentiables des circuits mémoires dont la sortie dépend également de leur état antérieur.

4.5.1 Multiplexeur (Mux)

C'est un cas typique de circuit MSI permettant d'**aiguiller** une entrée parmi 2^n sur son unique sortie grâce à n lignes de sélection.

Exemple : Mux à 4 entrées, 1 sortie

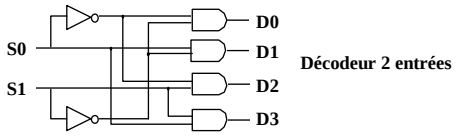


Ce type de circuit est notamment utilisé pour multiplexer le bus d'adresses et le bus de données du 8086. Associé à un compteur, il permet également la conversion **parallèle/série**. Le **démultiplexeur** réalise la fonction inverse en aiguillant une entrée unique vers l'une des 2^n lignes de sorties (série/parallèle).

4.5.2 Décodeur

Ce MSI dispose de n lignes d'entrées et de 2^n lignes de sortie. Selon le nombre x en RBNS présent sur les n entrées, seule la sortie d'indice égal à x est sélectionnée (active) tandis que les autres sont au repos. Selon les circuits, l'état actif peut être représenté par 1 ou bien par 0 (actif à l'état bas).

Exemple : décodeur 2 entrées actif à l'état 1



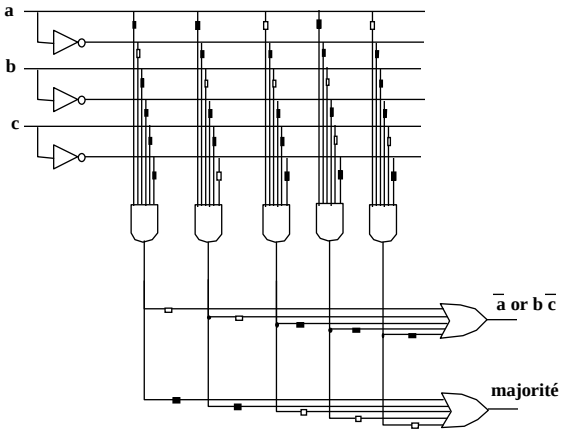
Remarques
Ce type de circuit permet de sélectionner les différents circuits mémoires de la MC. En effet, on utilise certaines lignes de poids forts du bus d'adresses comme entrées du décodeur et les sorties comme sélecteurs de boîtiers (*Chip Select* $\overline{\text{CS}}$).

Les circuits décodeurs du commerce sont de type 4 vers 16, 3 vers 8, ou contiennent plusieurs 2 vers 4. Il existe aussi des circuits spéciaux 4 vers 10 pour le DCB. Le circuit inverse nommé **codeur** permet de donner l'indice de l'unique ligne d'entrée active.

4.5.3 Autres circuits

Bien entendu, une multitude d'autres types de circuits combinatoires sont nécessaires. Afin de permettre la réalisation de fonctions logiques spécifiques, des circuits programmables existent. Les **réseaux logiques programmables** (*Programmable Logic Array* ou *PLA*) sont constitués d'un réseau de portes Not, And et Or, dont les entrées sont des fusibles. La programmation d'un PLA consiste à faire "griller" un certain nombre de ces fusibles afin de réaliser la fonction logique désirée.

Exemple :
PLA 3 entrées (3 not), 2 sorties (2 or), 5 and
1° sortie : $\overline{a}$ or b $\overline{c}$; 2° sortie : majorité



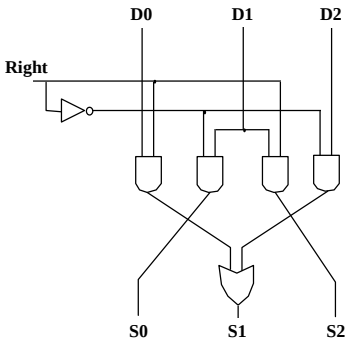
4.6 Circuits de calcul

Ces circuits MSI permettent d'effectuer les opérations arithmétiques et logiques.

4.6.1 Décaleur

Ce circuit permet de décaler un registre du processeur d'un ou plusieurs bits vers la droite ou la gauche. Il existe des décaleurs logiques (introduction de 0 sur le bit entrant), des décaleurs à droite arithmétiques (recopie du bit le plus significatif), et des "rotateurs" (recopie du bit sortant sur le bit entrant).

Exemple : Décaleur 1 position sur 3 bits

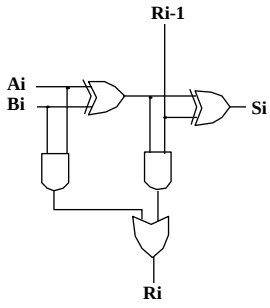


Remarque :
Décalage à gauche = multiplication par 2
Déc. arithmétique à droite = division par 2

4.6.2 Additionneur

Un demi-additionneur est constitué d'un XOR (même table de vérité addition) associé à un AND pour la retenue de sortie. Pour obtenir un additionneur, il faut rajouter un autre demi-additionneur pour la retenue d'entrée.

Exemple : additionneur 1 bit



Remarque :
l'additionneur est le circuit de base pour l'arithmétique binaire signée et non signée. En effet, toutes les autres opérations peuvent être obtenues logiciellement si on a un additionneur et un inverseur.

Question : OR de la retenue sortante R_i peut il être remplacé par un XOR? (oui)

4.6.3 U.A.L.

A partir des circuits de calcul précédents et de quelques circuits combinatoires, tels que des décodeurs, il est aisé d'imaginer la construction d'Unité Arithmétique et Logique ayant

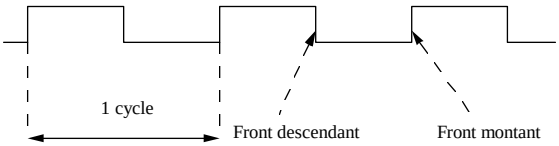
- en entrée : 2 mots de données de n bits, un mot de sélection de k bits
- en sortie, un mot de données de n bits résultant de l'une des 2^k opérations exécutée sur les 2 mots d'entrée ou sur un seul.

Exercice :
Construisez (schéma de câblage) une UAL réalisant l'addition, la soustraction, les décalages 1 bit, le C2, l'inversion sur des mots de 2 bits.

4.7 Horloge

L'horloge est la base de temps de l'UC. Celle-ci émet une suite d'**impulsions** calibrées. L'origine de ces impulsions est un **quartz** qui, soumis à une différence de potentiel, bat à une fréquence de quelques dizaines de MégaHertz (MHz). Le **temps de cycle** ou période de l'horloge varie donc entre 10 ns (100 Mhz) et 1 µs (1 MHz).

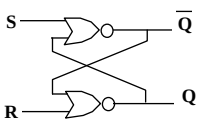
Exemple : diagramme temporel d'une horloge



Il est parfois impératif de diviser le cycle de temps de l'horloge principale en plusieurs intervalles. On utilise alors d'autres horloges déphasées par rapport à l'horloge principale (circuit de retardement). Le plus souvent, ces horloges délivrent des signaux symétriques et de même cycle (période).
Le cadencement des actions à l'intérieur de l'UC est toujours déclenché soit sur **front montant**, soit sur **front descendant**.

4.8 Circuits de mémorisation

Des circuits mémoires sont nécessaires dans toutes les parties d'une machine (UC, MC, périph.). A l'inverse d'un circuit combinatoire, la sortie d'un circuit mémoire ne dépend pas que de ses entrées mais aussi de son état précédent. Tous les circuits mémoires sont des dérivations de la **bascule RS** dont voici le schéma :



NOR

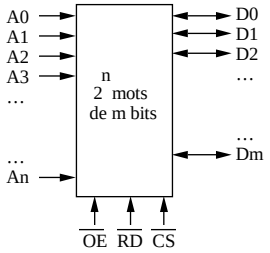
S	A	$\overline{Q}$
0	0	1
0	1	0
1	0	0
1	1	0

On remarque que les deux sorties Q et $\overline{Q}$ sont rarement (S=R=1) simultanément égales et que leur état est stable (bascule) tant que les deux entrées ne varient pas. L'activation de S (Set) à 1 provoque le positionnement de Q à 1. Ensuite, les variations de S sont inopérantes. Inversement, l'activation de R (Reset) bascule Q à 0 . La bascule RS se "souvient" donc de la dernière activation d'une entrée.

Des **améliorations** de ce circuit permettent de construire des bascules stables et sans ambiguïtés de fonctionnement (bascules JK, D). En associant n bascules, on construit des registres n bits.

4.8.1 Organisation de la mémoire

L'évolution technologique a provoqué une augmentation importante du nombre de bits stockés par boîtier. Années 70 : 1Kbit puis 16Kbits,…actuellement : 1Mbit. On peut schématiser un boîtier mémoire (RAM) de la manière suivante :



$\overline{OE}$ *Output Enable* active les lignes de sortie $D_{0...m}$, tandis que $\overline{RD}$ sélectionne la lecture ou l'écriture et que $\overline{CS}$ *Chip Select* sélectionne le boîtier. $\overline{OE}$ est nécessaire car lors d'une **lecture**, il synchronise la recopie du mot sélectionné sur les lignes $D_{0...m}$.

Une mémoire 16 Kbits peut être constituée de 2 boîtiers de 8 Kbits ou de 4 * 4 Kbits ou de 16 * 1Kbits …

5. La Couche Microprogrammée

La couche microprogrammée dépend **complètement** de la couche physique sous-jacente. En fonction de l'architecture interne de l'UC, elle permet de coder chaque **instruction du niveau Machine** en une **suite de micro-instructions** élémentaires.

Chaque micro-instruction est codée sur un certain nombre de champs, chaque champ indiquant l'activité d'un **signal interne** à l'UC. Dans l'UC, une **mémoire de commande** (ROM) (invisible à l'utilisateur) contient le texte correspondant à toutes les instructions machine. L'exécution d'une micro-instruction nécessite l'existence de plusieurs sous-cycles permettant de synchroniser les différentes actions. Ces sous-cycles sont obtenus grâce à des circuits retards internes.

L'enchaînement des micro-instructions est semblable à l'enchaînement des instructions machine ! La règle générale est la séquence mais des ruptures conditionnelles ou non peuvent intervenir. Une micro-instruction spéciale de fin permet au micro-séquenceur d'exécuter l'instruction suivante.

6. La Couche Machine

Elle constitue le niveau **le plus bas** auquel l'utilisateur a accès.

Types d'ordinateurs (évolution constante !)

- les ordinateurs personnels (PC ou Macintosh) ;
  - les ordinateurs de bureau ;
  - les ordinateurs portables ;
  - les assistants personnels (ou PDA) ;

- les moyens systèmes (midrange) (ex IBM AS/400-ISeries, RISC 6000...)

- les mainframes (serveurs centraux) (ex. : IBM zSeries 64 bits, Siemens SR2000 et S110 ...) ;

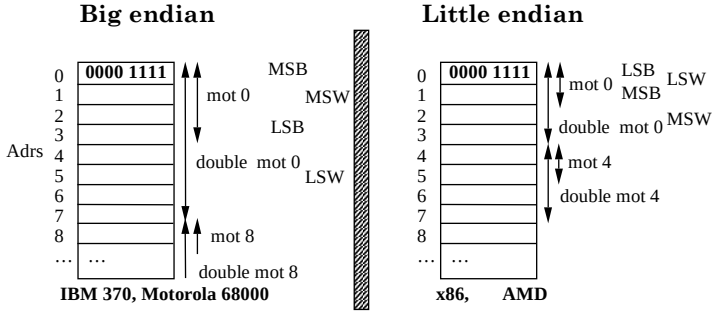
- les superordinateurs (Blue Gene machine IBM utilisant 65536 Power PC réalisant 136,8 TFlops);

- les stations de travail (PC puissants pour CAO, …) ;

6.1 Exemples

Pentium 4 (x86 CPU)
- 8 registres 32 bits (AL, AH, AX, EAX);
- Espace adrs : 4 Go ;
- Mémoire virtuelle segmentée (CS code, DS data, SS stack) avec un sélecteur de segment 16 bits + adrs virtuelle 32 bits ;
- (x87) coprocesseur arithmétique flottante (FPU) 32, 64 ou 80 bits

AMD 64
- 16 registres 64 bits (AL, AH, AX, EAX, RAX)
- Espace adrs : 256 To
- Mémoire virtuelle non segmentée sur 64 bits
- SIMD avec registres 128 bits



En Little endian, mnémotechniquement, la "gravité" est orientée vers le bas de la mémoire (adresses supérieures). Les Power PC d'IBM sont bi-endian.

6.2 Format des instructions

Programme = suite d'**instructions** machines

Une instruction est composée de plusieurs champs :
• 1 champ obligatoire : le **code opération** désigne le type d'instruction. Sa longueur est souvent **variable** afin d'optimiser la place et la vitesse utilisée par les instructions les plus fréquentes (Huffman).
• 0 à n champs optionnels : les **opérandes** désignent des données immédiates ou stockées dans des registres ou en MC. Le type de désignation de ces données est nommé **mode d'adressage**.

Exemple :

Code Opération	$Opérande_1$	$Opérande_2$

De plus, la taille des cases et mots mémoires doivent être des multiples de la taille d'un caractère afin d'éviter le **gaspillage de place** ou des **temps** de recherche prohibitifs. Les codes alphanumériques usuels étant sur 8 bits (EBCDIC) ou 7+1 bits (ASCII), c'est la raison du découpage des MC en octets.

Enfin, des adresses devant également être stockées en MC, c'est la raison pour laquelle la taille de l'espace d'adressage est généralement un multiple de 2^8 octets (64 Ko, 16 Mo, 4 Go). Sauf lorsque la technique d'adressage utilise des segments recouvrants : 8086 : 1Mo = 2^{20} o; 1 adrs = 2 mots.

6.3 Modes d'adressage

Remarque : pour simplifier, nous utiliserons dorénavant la notation mnémonique des instructions machines, notation alphanumérique qui correspond à la couche 4 du langage d'assemblage.

Types de donnée représentés par les opérandes :
- donnée **immédiate** stockée dans l'instruction machine
- donnée dans un **registre** de l'UC
- donnée à une **adresse** en MC

Nombres d'opérandes :
Nous supposerons un nombre maximal de 2 opérandes, ce qui représente le cas général. Souvent, un opérande implicite n'est pas désigné dans l'instruction : c'est l'**accumulateur** qui est un registre de travail privilégié; le Z80 a au maximum un opérande explicite (et A comme opérande implicite).

Source et Destination :
Lorsque 2 opérandes interviennent dans un transfert ou une opération arithmétique, l'un est source et l'autre destination de l'instruction. L'ordre d'apparition varie suivant le type d'UC :
IBM 370, 8086 : ADD DST, SRC *DST := DST+SRC*
PDP-11, 68000 : ADD SRC, DST *DST := DST+SRC*

6.3.1 Adressage immédiat

La valeur de la donnée est stockée dans l'instruction. Cette valeur est donc copiée de la MC vers l'UC lors de la phase de chargement (*fetch*) de l'instruction.

Avantage : pas d'accès supplémentaire à la MC
Inconvénient : taille limitée de l'opérande

Exemples : Z80
(1) ADD A, <n> ; Code Op. 8 bits, n sur 8 bits en C2
(2) LD <Reg>, <n> ; Code Op 5 b., Reg 3 b., n 8 b.

6.3.2 Adressage registre

La valeur de la donnée est stockée dans un registre de l'UC. La désignation du registre peut être explicite (2) ou implicite (1) (A sur Z80).

Avantage : accès rapide % MC
Inconvénient : taille limitée de l'opérande

Selon le nombre de registres de travail de l'UC, la taille du code des instructions varie :
8 registres nécessitent 3 bits (Z80)
16 registres nécessitent 4 bits(68000)

6.3.3 Adressage direct

La valeur de la donnée est stockée à une adresse en MC. C'est cette adresse qui est représentée dans l'instruction.

Avantage : taille quelconque de l'opérande
Inconvénient : accès mémoire supplémentaire, taille importante de l'instruction

Selon le type d'implantation mémoire requis par l'UC, plusieurs adressages directs peuvent coexister.

Exemples : 8086
MOV AL, <dis> ; déplacement intra-segment (court) transfère dans le registre AL, l'octet situé à l'adresse dis dans le segment de données : dis est codé sur 16 bits, *Data Segment* est implicite.

ADD BX, <aa> ; adresse absolue aa= S,D (long) ajoute à BX, le mot de 16 bits situé à l'adresse D dans le segment S : D et S sont codés sur 16 bits.

L'IBM 370 n'a pas de mode d'adressage direct, tandis que le 68000 permet l'adressage direct court (16 bits) et long (32 bits).

6.3.4 Adressage indirect

La valeur de la donnée est stockée à une adresse m en MC. Cette adresse m est stockée dans un registre r ou à une adresse m'. C'est r ou m' qui est codé dans l'instruction (m' est appelé un **pointeur**). L'adressage indirect par registre est présent dans la totalité des UC, par contre, l'adressage indirect par mémoire est moins fréquent. Il peut cependant être simulé par un adressage direct dans un registre suivi d'un adressage indirect par registre.
Peu de machines disposent de mode d'adressage indirect à plusieurs **niveaux d'indirection** !

Avantage : taille quelconque de l'opérande
Inconvénient : accès mémoire supplémentaire(s), taille importante de l'instruction (pas par registre)

Exemples : Z80 indirection seulement par HL
ADD A, (HL) ; adressage indirect par registre
codage de l'instruction sur 8 bits (code op.) : A et HL sont désignés implicitement

LD (HL), <reg> ; adressage indirect par registre
code op. sur 5 bits et reg sur 3 bits; HL implicite

6.3.5 Adressage indexé (ou basé)

Il est parfois nécessaire d'accéder à des données situées à des adresses consécutives en MC. En adressage indexé, on charge un **registre d'index** avec l'adresse de début de cette zone de données, puis on spécifie dans l'instruction, le déplacement à réaliser à partir de cet index. L'adresse réelle de la donnée accédée est donc égale à l'adresse de l'index ajoutée au déplacement.
Certaines UC exécutent automatiquement l'incrémentation ou la décrémentation de leurs registres d'index ce qui permet, en bouclant, de réaliser des transferts ou d'autres opérations sur des zones (chaînes de caractères).

Avantage : taille importante de la zone (256 octets si déplacement sur 8 bits)
Inconvénient : taille importante de l'instruction (si codage du déplacement)

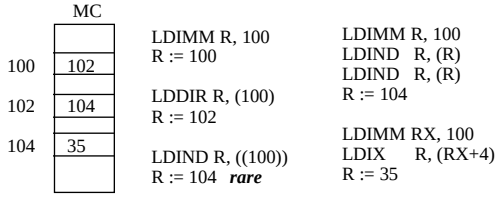
Exemple : Z80 indexation par IX et IY
LD (IX+<dépl>), <reg>; adressage indexé par IX
codage de l'instruction : code op. sur 12 bits, IX sur 1 bit, reg sur 3 bits, dépl sur 8 bits.

Sur le 8086, MOVSB (MOVe String Byte) permet de transférer l'octet en (SI) vers (DI) puis d'incrémenter ou décrémenter SI et DI. Remarquons que l'indexation sans déplacement équivaut à l'indirection.

6.3.6 Remarques

Dans une instruction, lorsque deux opérandes sont utilisés, deux modes d'adressages interviennent. Souvent certains modes sont incompatibles avec d'autres pour des raisons de taille du code instruction ou de temps d'accès : par exemple en 8086, jamais deux adressages directs. Par contre, un opérande fait parfois appel à la conjonction de deux modes d'adressage : mode indexé et basé + déplacement du 8086 !

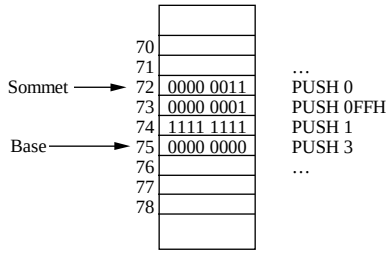
Schéma des différents modes d'adressage



Toute indirection à n niveaux peut être simulée dès lors qu'on possède une instruction à indirection simple.

6.3.7 Adressage par pile

La **pile d'exécution** de l'UC est constituée d'une zone de la MC dans laquelle sont transférés des mots selon une stratégie Dernier Entré Premier Sorti (*Last In First Out* ***LIFO***). Le premier élément entré dans la pile est placé à la **base** de la pile et le dernier élément entré se situe au **sommet** de la pile.



Les instructions d'entrée et de sortie de pile sont **PUSH** <n> et **POP** <reg>. Selon le type d'UC, la pile remonte vers les adresses faibles (voir schéma) ou bien descend vers les adresses fortes.

L'adresse du sommet de pile est toujours conservée dans un registre nommé **pointeur de pile** (*Stack Pointer SP*). Sur certaines machines, un registre général peut servir de SP. Certaines UC utilisent également un **pointeur de base de pile** (*Base Pointer BP*).

6.3.8 Utilisations de la pile

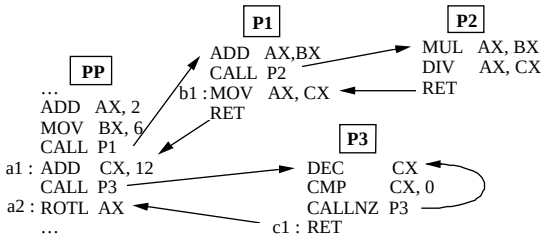
Avantages :
• structure de données LIFO et opérations de manipulation physiquement implantées : vitesse
• instructions courtes car opérandes implicites

Inconvénients :
• pas toujours dans une zone MC protégée
• sa cohérence nécessite égalité du nombre d'empilages et de dépilages (programmeur).
• capacité souvent limitée (par exemple 1 segment)

L'appel procédural

L'intérêt de l'utilisation de sous-programmes nommés **procédures** lors de l'écriture de gros programmes a été démontré : **concision**, **modularité**, **cohérence**, … Le programme principal (PP) fait donc appel (*CALL*) à une procédure P1 qui exécute sa séquence d'instructions puis rend la main, retourne (*RET*) à l'instruction du PP qui suit l'appel. Cette rupture de séquence avec retour doit également pouvoir être réalisée dans le code de la procédure appelée P1, soit récursivement, soit vers une autre procédure P2. Ces appels imbriqués, en nombre quelconque, rendent impossible l'utilisation d'une batterie de registres qui sauveraient les valeurs de retour du CO !

Exemple d'appels procéduraux imbriqués

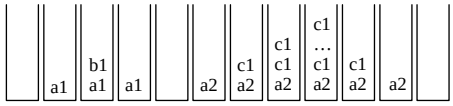


Les appels procéduraux imbriqués nécessitent l'utilisation de la pile de la manière suivante :

CALL <adrs> génère automatiquement (couche 1) :
1. PUSH CO ; le compteur ordinal pointe toujours sur l'instruction suivante
2. JMP <adrs> ; jump = goto

RET génère automatiquement :
1. POP CO ; branchement à l'adresse de retour

Exemple d'évolution de la pile



Attention aux appels récursifs mal programmés qui provoquent des débordements de pile (*Stack Overflow*) ! P3 : 2^n appels maximum (mots de n bits).

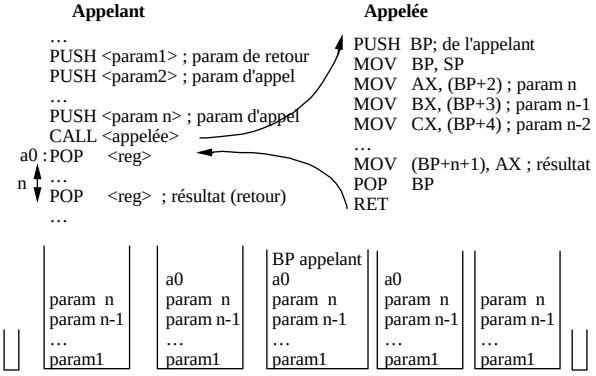
Paramètres et Variables locales

La plupart des langages de programmation évolués (Pascal, c, …) permettent le **passage de paramètres** et l'utilisation de **variables locales** aux procédures. Les paramètres sont passés soit **par valeur**, soit **par adresse**. Les variables locales sont créées à l'activation de la procédure et détruites lors de son retour (**durée**). D'autre part, leur **visibilité** est réduite aux instructions de la procédure.

Passage de paramètre

Dans les programmes écrits en langage machine, la gestion des paramètres d'appel et de retour est à la charge du programmeur (registres, MC, pile). Par contre, un **compilateur** doit fournir une gestion générique des paramètres quel que soit leur nombre et leur mode de passage. La plupart du temps, la pile est utilisée de la façon suivante :
Dans l'appelant, juste avant l'appel (CALL), le compilateur génère des instructions d'empilage (PUSHs) des paramètres d'appel et de retour. Juste après le CALL, il génère le même nombre de dépilage afin de nettoyer la pile. Dans le corps de la procédure appelée, la première opération consiste à affecter à un registre d'index ou de base (BP) la valeur du sommet de pile ± taille adresse de retour. Par la suite, les références aux paramètres sont effectuées via ce registre (BP±0..n) !

Exemple de passage de paramètres :



Variables locales

L'implémentation de l'espace dédié aux variables locales est réalisé dans la pile d'exécution. Les premières instructions machines correspondant à la compilation d'une procédure consistent toujours à positionner les variables locales dans la pile, juste au dessus de l'adresse de retour. Dans l'exemple précédent, il suffit de rajouter autant de PUSH, après le MOV BP,SP, que nécessaires. Ces variables locales seront ensuite accédées via des adressages (BP-i) générés par le compilateur. Ce type d'implémentation permet l'appel procédural ainsi que la récursivité.

Paramètres et variables locales

Finalement, chaque instance d'appel de procédure possède un espace d'adressage local composé :
- d'une part, des paramètres d'appel et de retour localisés dans la pile à (BP+2..n);
- d'autre part, des variables locales localisées dans la pile à (BP+1..m).

Exemple simple

Nous considérerons une procédure récursive simple n'utilisant que des paramètres et variables locales codés sur un mot machine. La fonction **mult** est une procédure à un paramètre de retour réalisant la multiplication de 2 entiers positifs par additions successives. Nous ne traiterons pas des problèmes de dépassement de capacité ni de l'optimisation de l'algorithme (y=0).

```
entierpositif mult(entierpositif x, entierpositif y)
entierpositif i ;      // inutile dans l'algo. !
si x=0 alors retourne 0
sinon début
        i=y
        retourne mult(x-1,i) + y
        fin
```

Etudions le code compilé de cette procédure et d'un appel initial avec des données en entrée : x=2, y=5.

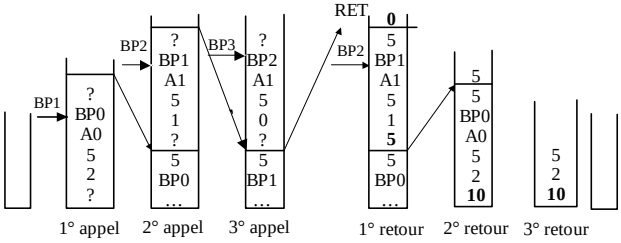
Image du code compilé et de la pile

Appelant

```
…
PUSH <multret> ; param de retour
PUSH x ; param d'appel 2
PUSH y ; param d'appel 5
CALL mult
A0: POP DX
POP DX
POP AX ; résultat (retour)
CALL AFFICHEAX
…
```

Mult

```
PUSH BP ; de l'appelant
MOV BP, SP
PUSH DX ; i qcq en (BP-1)
CMP (BP+3), 0 ; x=0 ?
JE ZERO
MOV (BP-1),(BP+2) ; i:=y
MOV AX, (BP+3) ; x
DEC AX ; x-1
PUSH DX ; retour qcq
PUSH AX ; appel x-1
PUSH (BP-1) ; appel i
CALL mult ; récursif
A1: POP DX ; dépile
POP DX ; dépile appel
POP AX ; résultat (retour)
ADD AX, (BP+2) ; résult + y
MOV (BP+4), AX ; range résultat
JMP FIN
ZERO : MOV (BP+4),0 ; range rés.=0
FIN : POP DX ; var locale i
POP BP ; BP appelant
RET
```



Cet exemple illustre bien le danger de croissance de la pile lors d'appels récursifs mal programmés ! Remarquons que la **dérécursivation** évidente de mult peut être réalisée par le programmeur mais parfois aussi par le compilateur.

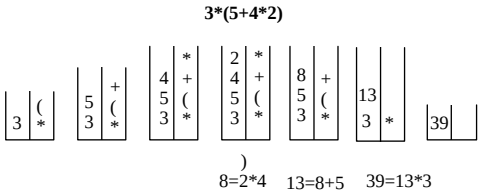
Autres utilisations de la pile

Cette structure de données LIFO est massivement employée dans le cadre d'algorithmes divers et variés.

Exemples

• dérécursivation automatique : dans l'exemple précédent, les CALL récursifs peuvent être évités en empilant successivement les valeurs des paramètres d'appels puis en réalisant des additions successives lors des dépilages.

• évaluation d'expressions arithmétiques infixées et parenthésées : 3*(5+4*2)



• parcours d'arbre (préfixe, infixe, postfixe)

• etc …

6.4 Types d'instructions

Chaque catégorie d'Unité Centrale possède un jeu particulier d'instructions machines. Cependant, on peut classer les instructions de ces jeux en type d'instructions : transfert, opérations arith., …

6.4.1 Transfert de données

Ces instructions permettent de **copier** ou bien de **déplacer** une donnée d'un endroit **source** vers un emplacement **destination**. Les instructions de copie sont largement majoritaires par rapport aux instructions de déplacement ! Selon les UC, les termes suivants sont employés pour la copie : transfert, duplication, déplacement, mouvement, chargement, rangement ! On aperçoit donc souvent les mnémoniques : **MOV**, LOAD ou **LD**, STORE ou **ST**, **PUSH**.
Remarquons que les instructions de déplacement avec destruction de la source sont la plupart du temps restreintes aux dépilages : **POP**.

Ces instructions de transfert doivent préciser : source , destination (voir modes d'adressage), taille de la donnée. La taille de donnée peut être codée dans le Code Opération mais peut également être implicite ou constituer un opérande à part entière.

6.4.2 Opérations arith. et logiques

• Op. dyadiques (à deux opérandes)
Un des 2 opérandes est souvent considéré comme destination de l'opération à moins que le résultat ne soit toujours transféré que dans l'accumulateur. Chaque machine a son type de représentation préféré des opérandes (C2, DCB,…).

Addition et Soustraction (toujours) : ADD SUB
Multiplication et Division (parfois) : MUL DIV
ET, OU, OUeXclusif : AND OR XOR

Remarques :

- **masquage** par ET : AND R1, 00FH ; permet de ne conserver dans R1 que le quartet de poids faible !
- **inversion** par OUX : XOR R1, 0F0H; permet d'inverser le quartet de poids fort de R1 !
- **raz** par OUX : XOR R1,R1; <=> R1:=0 et plus rapide (adressage registre à registre) !

Les opérations en virgule flottante (simple ou multiple précision) peuvent être câblées dans l'UC ou câblées dans un co-processeur arithmétique ou exécutées logiciellement par des sous-programmes.

• Opérations monadiques (1 seul opérande)
L'unique opérande source et destination peut parfois être implicite (accumulateur).

- Décalages et rotations : SHL=SAL, SHR, SAR, ROL, ROR, RCL, RCR
- Inc. et décrémentations : INC DEC
- Négations log. et (arith.) : NOT (NEG)
- raz : CLEAR

Remarques
- test d'un bit par rotation : RCL R1; JC toto; positionne $R1_{n-1}$ dans le *Carry Flag*.
- décalage multiple → dyadique : 2° opérande immédiat ou dans un registre compteur : MOV CX, 4; SHL R1,CX ou bien SHL R1, 4
- négation C2 : NEG R1 ou NOT R1; INC R1
- mult. et div. : $R1*19= R1*2^4+R1*2^1+R1*2^0$ → 2 décalages (dont un multiple) et 3 additions.

6.4.3 Branchements et comparaisons

Pour réaliser des ruptures de séquence, des instructions de branchement, ou de saut, permettent de modifier la valeur du Compteur Ordinal. Ces instructions sont soit inconditionnelles (GOTO, BRANCH, JUMP), soit conditionnées par certaines valeurs des indicateurs du registre d'état (PSW ou Flags).

• Branchements inconditionnels :
JUMP adrs → CO := adrs

• Branchements conditionnels
JNC adrs → si CF=0 alors CO:=adrs

Il existe différents indicateurs dans le registre d'état : Carry Flag, Overflow F., Zero F., Neg. F., …

Ces indicateurs sont positionnés à la suite de l'exécution d'une opération arith. ou logique et sont **plus ou moins** rémanents ! Afin d'éviter une affectation inutile, une instruction particulière de **comparaison** (CMP) permet de simuler la soustraction :
CMP R1, R2 → PUSH R1; SUB R1,R2; POP R1

A la suite de la comparaison, les indicateurs sont positionnés et permettent donc d'effectuer un branchement conditionnel :
CMP R1,R2; JAE adrs3

Remarque :
il existe des UC ayant des instructions à 3 adresses permettant le branchement conditionnel en une instruction :

CMP adrs1, adrs2, adrs3, c'est-à-dire *si* (adrs1)-(adrs2)=0 *alors* JMP adrs3 *sinon* <continuer en séquence>

6.4.4 Procédures

Une procédure, ou routine ou sous-programme, est constituée d'une suite d'instructions réalisant un traitement particulier. L'appel à cette procédure est généralement réalisé par une instruction CALL adrsproc. Le retour à l'appelant s'effectue dès que l'on exécute un RET dans la procédure appelée.

La **récursivité** d'une procédure est une propriété permettant à celle-ci de s'appeler elle-même. La mise en œuvre de la récursivité est permise par la sauvegarde dans la pile de l'adresse de retour, des paramètres d'appels ainsi que des variables locales à la procédure.

6.4.5 Itérations

L'exécution répétée d'une suite d'instruction, ou boucle, est permise grâce à certaines instructions de branchements conditionnels à 3 adresses :
LOOP CX, 0, adrs ; permet de décrémenter le registre CX (compteur), de comparer CX avec 0, enfin si CX?0 de boucler sur adrs (branchement).

Sur certains processeurs, le compteur est implicite et la valeur immédiate (0) également : LOOP adrs

6.4.6 Les Entrées/Sorties

I) Ces instructions sont très fortement liées au matériel et varient donc énormément en fonction de l'UC considérée. Considérons d'abord les UC pourvues d'**instructions d'E/S spécifiques** :

1) L'UC exécute les instructions d'E/S et gère le périphérique : l'UC passe son temps à attendre …

2) L'UC délègue sa responsabilité à une Unité d'E/S (ou d'échange) qui effectue le contrôle du transfert des données. Paramètres : Nom périph., sens transfert, adrs MC début (tampon ou *buffer*), nb de mots. Une fois le transfert initialisé par une instruction spécifique, l'Unité d'E/S réalise ce transfert de façon autonome en accédant directement à la MC (*Direct Memory Access*).

exemple d'Unité d'E/S : canal IBM 370
1) l'UC crée un prg canal en MC (suite d'instructions spécifiques au canal considéré).
2) l'UC charge à l'adrs MC 72_{10} l'adrs de ce prg.
3) l'UC exécute l'inst^{on} START I/O n°canal, n°périph
4) l'UC entreprend d'autres traitements
4') Le canal effectue le transfert en exécutant le prg stocké à l'adresse 72.
5) l'UC peut tester ou stopper l'E/S : instructions TEST I/O, TEST CHANNEL ou STOP I/O.

II) l'UC ne possède pas d'instructions d'E/S spécifiques. Dans ce cas, l'UC communique avec les périphériques de la même façon qu'avec la MC. On dit que les E/S sont **projetées** (ou mappées) en mémoire. Certaines adresses sont attribuées aux divers registres internes des unités d'E/S (registres programme, d'état, tampon). L'UC écrit donc des informations à ces adresses pour demander à l'unité d'E/S correspondante d'entreprendre une E/S.

• Le premier avantage des E/S projetées en MC consiste à réduire le jeu d'instructions de l'UC en n'ayant pas d'instructions spécialisées d'E/S. Mais l'intérêt primordial de cette méthode réside dans l'utilisation de la puissance du jeu d'instructions tout entier et des divers modes d'adressage pour accéder aux registres des unités d'E/S.

• L'inconvénient principal est la diminution de l'espace allouable à la Mémoire Centrale.

Exemple :
Le PDP-11 a une imprimante standard projetée à l'adresse octale 777514 (registre d'état) et 777516 (registre tampon). Lorsque le bit 7 du registre d'état est à 1 (bit *ready*) , cela signale que l'imprimante est prête à recevoir un car. dans son registre tampon.

6.5 Flux de commande

Le flux de commande exprime l'ordre d'exécution des instructions par l'UC. Le flux de commande standard ou normal est la séquence. Les instructions de branchement (conditionnel/ inconditionnel) ou d'appel de procédures, ainsi que les déroutements et les interruptions modifient cet ordre.

6.5.1 Sauts et appels procéduraux

Tout d'abord, on rappellera la nocivité des instructions GOTO (Branch, Jump) dans le cadre de l'écriture de programmes sans erreur. Par conséquent, on essayera toujours :
1) d'éviter d'écrire les programmes dans des langages sans structures de contrôle de haut niveau (while, for, repeat);
2) de minimiser le nombre de saut dans les autres programmes (notamment ceux de la couche machine).

On rappelle simplement ici le mécanisme d'appel (CALL) et de retour (RET) des procédures et l'utilisation implicite et explicite de la pile pour la conservation : des **adresses successives de retour**, des **paramètres d'appel** et de **retour**, enfin des **variables locales** à chaque appel.

6.5.2 Déroutements

Un déroutement (ou trappe *trap*) pendant l'exécution d'un programme P consiste à stopper P et à appeler automatiquement une routine de traitement lorsque certaines conditions surviennent. Par exemple, un débordement de capacité survenu à la suite d'une opération arithmétique peut provoquer le déroutement du programme vers une routine de traitement de l'erreur.

Remarquons qu'un déroutement est provoqué par l'exécution du programme lui-même, qui, en positionnant des indicateurs du mot d'état, déclenche, au niveau physique, un appel à la procédure appropriée. Tout déroutement pourrait être simulé par logiciel en testant le mot d'état puis en exécutant un appel conditionnel à la routine après **chaque** opération dangereuse.

Le fait de traiter automatiquement les déroutements au niveau physique permet :
- l'augmentation de la vitesse d'appel (couche physique)
- la diminution de la taille du code

Exemples :
débordement, division par 0, débordement de pile, violation de protection, opération indéfinie…

6.5.3 Interruptions (IT)

Evénement provoqué par une cause **externe** au programme et qui interrompt celui-ci pour exécuter une routine de traitement de l'interruption. A la fin de cette routine, le programme interrompu reprendra son exécution dans l'état exact où il l'avait laissée (sauf exception : chute alimentation !).

Cette clause de reprise implique la sauvegarde du **contexte** du programme en début de routine d'IT et sa restauration à la fin. Le contexte du programme comprend en particulier l'ensemble des valeurs de tous les registres, y compris PSW.

Les IT sont principalement utilisées dans la gestion des E/S pour signaler la fin des transferts réalisés de façon asynchrone, ce qui permet d'éviter l'attente active de l'UC.

Remarques :
- Les déroutements sont parfois nommés interruptions internes (8086).
- La différence essentielle entre un déroutement et une IT est que le déroutement est synchrone avec le programme, alors que l'IT ne l'est pas. L'instant d'apparition d'une IT est indépendante du programme, pas le déroutement !

Niveaux d'interruption
En général, il existe plusieurs niveaux de priorités d'IT. En effet, différents périphériques étant connectés, il importe d'établir une politique de gestion des conflits. Différents algorithmes utilisant plusieurs **files de priorités** sont utilisés.

Par exemple, chaque niveau de priorité est associé à un type de périph. et par conséquent à une routine de traitement correspondante. Le numéro de périph. communiqué lors de l'IT permettra de le distinguer des autres de même type.

Transparence des IT imbriquées
Le mécanisme de traitement des IT est transparent lorsqu'il s'apparente au mécanisme de l'appel procédural. On utilise la pile pour sauvegarder les contextes du programme et de la suite des routines d'IT interrompus. Attention à ne pas permettre de nouvelle IT pendant le chargement ou la sauvegarde d'un contexte (atomicité) !

Exemple :
L'IBM 370 ne permet pas les IT imbriquées car un seul mot MC stocke le PSW du prog. interrompu. Pour éviter l'écrasement de ce mot, un indicateur de contrôle des IT du PSW de la routine est positionné afin d'interdire (**masquer**) toute nouvelle prise en compte d'une autre IT.

7. Perspectives et Conclusion

7.1 Perspectives de l'ASI

- Stratégie **RISC/CISC** : processeurs à jeu d'instruction de limité/complexe : qui va gagner ?

- Nouveaux supports de mémoires de masse : optiques et magnétiques, DVD ...

- **Miniaturisation** : on va atteindre une asymptote horizontale pour les composants électroniques actuels → changement de technologie
  - métaux **supraconducteurs**
  - ordinateurs biologiques (Science-Fiction ?)

- **Standardisation** : vue de l'esprit puisque marché fortement concurrentiel et impératifs commerciaux prépondérants

- Architectures **parallèles** de machines
  - multiprocesseurs à mémoire partagée
  - machines systoliques communicantes (réseaux quadratiques ou cubiques de processeurs)

7.2 Les autres couches : à suivre…

Systèmes d'exploitation

Etude des principes fondamentaux des systèmes d'exploitation :
- fichiers,
- processus,
- mémoire.

Assembleur

- mise en œuvre d'algorithmes sur un jeu d'instructions assembleur particulier ;
- écrire proprement dans un langage peu structuré ;
- étudier concrètement des mécanismes d'adressage ;
- (interfacer avec les autres couches : système d'exploitation et applications en langage évolué).

7.3 Conclusion

Il est absolument indispensable de comprendre les **concepts de base** des machines matérielles afin :

- d'**évaluer** correctement les résultats numériques fournis par les machines (précision).

- de **programmer** intelligemment les algorithmes dont on a minimisé la complexité (des E/S fréquentes peuvent ruiner un algorithme d'une complexité inférieure à un autre).

- de pouvoir **optimiser** les parties de programme les plus utilisées en les réécrivant en langage de bas niveau.

- d'écrire des compilateurs ou des interpréteurs performants même si ceux-ci sont écrits en langage de haut niveau.

- de se préparer à l'arrivée de nouveaux paradigmes de programmation (**programmation parallèle**).

… et enfin d'obtenir une bonne note lors de l'évaluation !

8. Les Systèmes de Gestion de Fichiers

8.1 Les Fichiers

8.1.1 Introduction

Définition conceptuelle :
Un fichier est une **collection** organisée d'informations de même nature regroupées en vue de leur conservation et de leur utilisation dans un Système d'Information.

Remarque :
inclut les SI non automatisés (agenda, catalogue de produits, répertoire téléphonique,…)

Définition logique :
C'est une **collection ordonnée** d'articles (enregistrement logique, item, "record"), chaque article étant composés de champs (attributs, rubriques, zones, "fields"). Chaque champ est défini par un nom unique et un domaine de valeurs.

Remarque :
Selon les SE, la longueur, le nombre, la structure des champs est fixe ou variable. Lorsque l'article est réduit à un octet, le fichier est qualifié de **non structuré**. Au niveau logique, plusieurs modèles de base de données ont été définis : modèle relationnel [Codd 76], réseau, hiérarchique.

Définition physique :
Un fichier est constitué d'un ensemble de **blocs** (enregistrement physique, granule, unité d'allocation, "block", "cluster") situés en **mémoire secondaire**. Les articles d'un même fichier peuvent être groupés sur un même bloc (Facteur de groupage ou de Blocage (FB) = nb d'articles/bloc) mais on peut aussi avoir la situation inverse : une taille d'article nécessitant plusieurs blocs. En aucun cas, un article de taille ≤ taille d'un bloc n'est partitionné sur plusieurs blocs → lecture 1 article = 1 E/S utile.

Remarque :
les blocs de MS sont alloués à un fichier selon différentes méthodes liées au type de support qu'il soit adressable (disques, …) ou séquentiel (bandes, cassettes, "streamers"). Ces méthodes d'**allocation** sont couplées à des méthodes de **chaînage** des différents blocs d'un même fichier et seront étudiées dans le chapitre SGF.

8.1.2 Opérations et modes d'accès

Un certain nombre d'opérations génériques doivent pouvoir être réalisées sur tout fichier :

- création	création et initialisation du **noeud descripteur** (i-node, File Control Block, Data Control Block) contenant taille, date modif., créateur, adrs bloc(s), …
- destruction	désallocation des blocs occupés et suppression du noeud descripteur
- ouverture	réservation de tampons d'E/S en MC pour le transfert des blocs
- fermeture	**recopie des tampons MC vers MS** (sauvegarde)
- lecture	consultation d'un article
- écriture	insertion ou suppression d'un article

La lecture et l'écriture constituent les **modes d'accès** et peuvent être combinés (mise à jour) lors de l'ouverture d'un fichier (existant). A la création, un fichier est toujours ouvert en écriture. Un SE multi-utilisateurs doit toujours vérifier les droits de l'utilisateur lors de l'ouverture d'un fichier.

8.1.3 Caractéristiques fonctionnelles

Volume : taille d'un fichier en octets ou multiples. Si articles de longueur fixe alors taille article*nb articles.

Taux de consultation/mise à jour pour un traitement donné : rapport entre le nombre d'articles intervenant dans le traitement et nb total d'articles.

Exemple : fichier "personnel", traitement "paye" ==> taux de consultation = 100%

Remarque : un taux de consultation important implique souvent un traitement par lot avec une méthode d'accès séquentielle.

Fréquence d'utilisation : nb de fois où le fichier est utilisé pendant une période donnée.

Taux d'accroissement : pourcentage d'articles ajoutés pendant une période donnée.

Taux de renouvellement/suppression : pourcentage d'articles nouveaux/supprimés pendant une période donnée. (TR=TS ==> volume stable)

Ces différentes caractéristiques ainsi que le type prépondérant d'utilisation (traitement par lot ou interactif) d'un fichier doivent permettre de décider de la structuration des articles, du type de support de stockage et des méthodes d'accès.

On classe souvent les fichiers en différentes catégories :

fichiers **permanents** : informations vitales de l'entreprise : Client, Stock, Fournisseurs...
durée de vie illimitée, Fréquence d'Utilisation élevée, màj périodique par mouvements ou interactive.

fichiers **historiques** : archives du SI : Tarifs, Prêts-Bibliothèque, journal des opérations…
pas de màj, taux d'accroissement élevé, taux de suppression nul.

fichiers **mouvements** : permettent la màj en batch des permanents afin d'éviter incohérences ponctuelles : Entrée/Sortie hebdomadaire stock, heures supplémentaires Janvier, …
durée de vie limité, fréquence d'utilisation très faible.

fichiers de **manoeuvre** : durée de vie très courte (exécution d'un programme) : spool, fichier intermédiaire.

8.1.4 Structure et Longueur des articles

Articles de longueur fixe et de structure unique
Dans la plupart des cas, chaque article d'un fichier contient le même nombre de champs chaque champ étant de taille fixe. Il existe un type unique d'articles.

Exemple : fichier STOCK

nom champ	*type*	*taille*	*exemple*
REF	N	4	0018
DESIGN	A	15	VIS 8*20
QTE	N	7	550 000
DATE	D	6	23/09/92

Articles de longueur fixe et de structure multiple
Les articles peuvent varier de structure parmi plusieurs sous-types d'articles (record variant de Pascal, union de C, C++). La longueur fixe des articles d'un tel fichier correspond à la longueur maximale des différents sous-types.

Exemple : fichier PERSONNE, articles de 42 octets

NOM(20), PRENOM(15), SITUATION(1),
  cas SITUATION=M alors DATE_MAR(6)
  cas SITUATION=D alors DATE_DIV(6)
  cas SITUATION=C alors rien

Remarque : les deux types d'articles précédents constituent la quasi-totalité des fichiers. La longueur fixe des articles permet de réaliser efficacement les calculs d'adresses de ceux-ci.

Articles de longueur var. et de structure multiple
L'intérêt principal de ce type d'articles consiste dans le gain de place (suppression des blancs) constitué par le compactage des fichiers concernés. L'inconvénient majeur réside dans la difficulté à calculer l'adresse d'un article. Ce type d'articles est particulièrement utilisé à des fins d'archivage sur support séquentiel et pour la téléinformatique.

La séparation des articles et des champs est souvent effectuée par insertion de préfixes indiquant la longueur de l'article ou du champ.

Exemple : fichier PERSONNE

TailleArt(1), NOM(1+20), PRENOM(1+15), SIT(1),
  cas SIT=M alors DATE_MAR(6)
  cas SIT=D alors DATE_DIV(6)
  cas SIT=C alors rien

18,5,UHAND,4,PAUL,M,23/08/91,
12,5,PETIT,4,JEAN,C,
19,6,HOCHON,4,PAUL,D,10/10/89,
…

8.1.5 Méthodes d'accès

8.1.5.1 Accès Séquentiel

Les articles sont totalement et strictement ordonnés. L'accès (lecture/écriture) à un article ne peut être réalisé qu'après l'accès à l'article précédent. Un **pointeur** d'article courant permet de repérer la position dans le fichier à un instant donné. L'**ouverture** en lecture positionne le pointeur sur le **1er article** puis les lectures successives font progresser le pointeur jusqu'à la position End Of File (**EOF**). Selon les cas, l'ouverture en écriture positionne le pointeur en début de fichier (rewrite) ou en fin de fichier (append). Il existe une opération spécifique de **remise à zéro** du pointeur (retour en début de fichier, rembobinage, "reset", "rewind").

Historiquement lié au support **bande** magnétique et cartes perforées, cette méthode d'accès est la plus simple et de surcroît est **universelle**. Elle reste très utilisée notamment pour les fichiers non structurés (textes, exécutables,…) ou les fichiers historiques. Pour les fichiers permanents, si le taux de consultation ou de màj est important, la solution séquentielle doit être envisagée. Cependant, l'accès séquentiel est impensable pour un bon nombre d'op. interactives (réservations, op. bancaires, …)

8.1.5.2 Accès direct
(ou sélectif, aléatoire, "random")

Ce type d'accès nécessite un support **adressable**! Un ou plusieurs champs des articles servent **d'expression d'accès (clé)** pour "**identifier**" et accéder à un ou plusieurs articles. On peut directement lire ou écrire l'article grâce à une opération du type suivant :

lire/écrire(fichier, valclé, adrsMCtransfert)

On peut également vouloir accéder aux articles par l'intermédiaire de plusieurs clés :

Exemple : fichier Personne; clé1 = N°SS; clé2 = Nom

Par la suite, nous traiterons de la manière de réaliser l'accès direct sur un fichier à **clé unique**. Selon les cas, la transposition aux clés multiples est plus ou moins simple !

Adressage direct

Ce type d'adressage est un cas d'école puisqu'il associe à chaque **valeur de clé** l'**adresse physique** du bloc contenant l'article. Il y a ainsi identité des valeurs de clé d'article et des adresses physiques de bloc. Un inconvénient évident est que le domaine des valeurs de clé doit correspondre exactement aux domaine des adresses physiques. De plus, l'espace adressable est très fortement **sous-occupé** (FB=1) et ne peut être partagé par plusieurs fichiers ayant des valeurs de clés conflictuelles ! Par contre, le temps d'accès à un article est **minimal**. La clé doit être identifiante !

Adressage relatif

La clé est un nombre entier correspondant au **n**uméro **l**ogique (0..n-1) d'article dans le fichier. On obtient aisément l'adresse physique (bloc ; dépl.) de celui-ci :
taille fixe : AdPhy:=(Orga(nl div FactBloq); taille*(nl mod FactBloq))
taille var. : il existe une table de correspondance [nl --> AdPhy]

Si la numérotation logique n'est plus continue (suppressions), de l'espace inutile continue à être occupé par le fichier ! En effet, la compression après chaque suppression serait trop coûteuse. L'insertion suivante sera donc réalisée dans un trou. L'ordre des nl ne correspond donc pas forcément à l'ordre d'insertion !

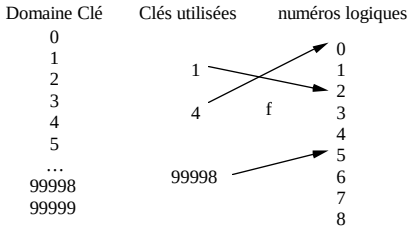
Adressage dispersé, calculé ("hash-coding")

L'adressage dispersé consiste à calculer un **n**uméro **l**ogique d'article à partir d'une valeur de clé et d'une fonction de hachage : nl:=f(c).

f permet de réduire le domaine des nl par rapport au domaine des clés afin de donner un volume de fichier supérieur au volume utile mais inférieur à card(domaineClé)*tailleArt.

Avantages :
calcul en MC ==> accès très rapide
clés quelconques : non identifiantes ==> collisions

exemple :



Dans cet exemple, un domaine de 10^6 clés potentielles est réduit à un espace logique de 9 numéros logiques permettant de stocker les 3 articles réels.

hachage parfait
Une fonction de dispersion idéale est celle qui réalise une **bijection** de l'ensemble des clés existantes vers l'ensemble des numéros logiques. Il n'y a ainsi aucun espace perdu. La recherche d'une fonction idéale est calculable sur un ensemble statique de clés identifiantes mais impossible sur un ensemble dynamique

Collisions ou conflits
Il y a collision lorsque la fonction de hachage associe un numéro logique déjà utilisé à un nouvel article. Il faut alors traiter la collision pour insérer le nouvel article dans le fichier :
- soit dans une zone de collision générale accédée soit séquentiellement, soit par une autre fonction f2
- soit dans une zone de collision spécifique à chaque numéro logique.

La détermination d'une bonne fonction de hachage donnant un **faible taux de collision** est un gage de rapidité d'accès. **Exemple** : somme des entiers (16 bits) composant la clef modulo taille fichier. Avec une clé non identif., le taux de collision augmente.

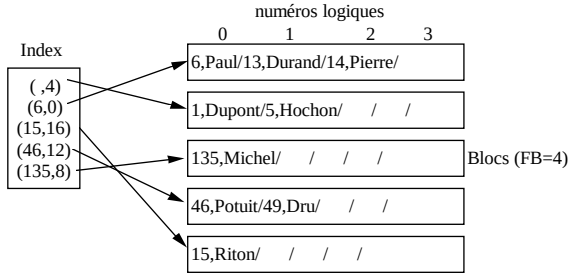
Lorsque le volume utile du fichier croît, il est nécessaire de réorganiser celui-ci, par exemple, en allouant un espace logique double du précédent, en modifiant f et en réorganisant les articles existants.

inconvénients :
collisions coûteuses
pas d'accès séquentiel (accès calculés répétés)
taux d'occupation mémoire <= 1
taille du fichier connue à priori
réorganisations coûteuses

Adressage indexé

Un **index** est une table de couples (valeur clé, nl) **triée** sur les valeurs de la clé. L'index est **dense** si toutes les clés du fichier y sont recensées. Sinon l'index est dit **creux** et une clé c présente dans le fichier et absente de l'index est dite couverte par la clé tout juste inférieure présente dans l'index. Un index creux implique une **clé primaire**, c'est à dire que le fichier soit **trié** sur cette clé. D'autres index secondaires doivent alors être dense !

Exemple d'Index creux primaire :



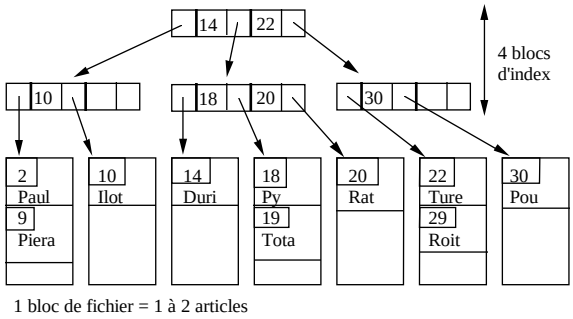
La recherche d'une clé d'un article est réalisée par dichotomie sur le fichier d'index. Lorsque le fichier atteint un volume important, son fichier d'index ne tient plus sur un seul bloc et on est alors forcé de parcourir séquentiellement les blocs d'index.

Aussi, on préfère une organisation arborescente (**b-arbre**) de l'index dans laquelle existe une hiérarchie de sous-tables d'index creux permettant une bonne rapidité d'accès. En fait, on indexe chaque bloc d'index !

Contraintes sur un b-arbre d'ordre p (#ptrs/bloc index)

- tout chemin (racine --> feuille) est de longueur identique = hauteur (exemple h=2)
- chaque bloc d'index est toujours au moins à moitié plein (sauf la racine) : chaque bloc d'index contient k clés avec EntInf(p/2)≤k≤p-1 (ex. p=3).

Exemple d'index par b-arbre :



1 bloc de fichier = 1 à 2 articles

Afin de maintenir les contraintes lors des suppressions ou des insertions d'articles, on est parfois obligé de **réorganiser** l'arborescence !
Moins rapide que le hachage pour l'accès direct, l'adressage **indexé** permet cependant de traiter également l'**accès séquentiel** à moindre coût (index creux ou b-arbre).

Remarques

- Il existe toujours un compromis (place utilisée/temps d'accès).
- Les insertions et suppressions d'articles provoquent des trous dans les blocs de données. L'index b-arbre gère ceux-ci mais les index linéaires (creux ou denses) ne le font pas… (voir SGF)
- Pratiquement, les b-arbres et les index creux (ISAM) prédominent dans les SGBD (vitesse + accès séquentiel).
- index multiples : on peut ajouter des index secondaires denses peu efficaces pour l'accès séquentiel.

8.2 Système de Gestion des Fichiers

8.2.1 Organisation arborecente

La quasi-totalité des SGF sont organisés en arborescence de **répertoires** (catalogue, "directory") contenant des sous-répertoires et fichiers. Suivant le cas, les répertoires sont considérés comme des fichiers (unix) ou bien demeurent dans des zones de MS spécifiques (MS-DOS zone DIR). Le répertoire permet d'accéder au **noeud descripteur** d'un fichier qu'il contient. Ce noeud permettra de connaître la localisation et l'ordonnancement des blocs contenant les données de ce fichier. Par la suite, on supposera des répertoires de même nature que les fichiers.

8.2.2 Allocation de mémoire secondaire

8.2.2.1 Support séquentiel (bande)

- pistes longitudinales (1 car/ 8|16 pistes //)
- densité d'enregistrement en bpi (1600..6000)
- blocs séparés par des gap inter-bloc (2 cm)
- fichiers séparés par des gaps inter-fichier
- entêtes de blocs et de fichiers permettant le positionnement.
- on ne modifie jamais directement une bande.
- les maj sont effectuées lors de la recopie sur une autre bande.
- faible coût, accès lent ==> archives, copies, sauvegardes

8.2.2.2 Support adressable (disque)

Espace adressable numéroté (0..b-1) de b blocs. L'accès à deux **blocs consécutifs** est peu coûteux (translation et rotation minimale de la tête de lecture/écriture). Remarquons que la numérotation logique des blocs ne correspond pas à la topologie (facteur d'entrelacement).

2 problèmes :

- disponibilité ou occupation d'un bloc
- localisation et ordonnancement des blocs d'un fichier

Gestion des blocs disponibles

Une liste des blocs libres doit être maintenue afin de gérer les créations et destructions de fichiers.

implémentations :

- tableau de bits : 0100011111101…110 (mot de b bits)

demande bloc : chercher le 1° 0 dans la liste puis 0 -> 1
rejet bloc : 1 -> 0

- considérer cette liste des blocs libres comme un fichier, donc utiliser une des méthodes d'allocation suivantes.

Allocation contigüe

Ex : fic1 (b0..b3); fic2(b9..b12); …; fick(b5..b6)

• temps d'accès séquentiel et direct optimal
• noeud descripteur contenant blocDebut, nbBlocs
• peu utilisé car les fichiers ont des tailles dynamiques ==> surévaluation a priori des volumes fichiers, réorganisations fréquentes et coûteuses lors des augmentations de volume.
• utilisation pour les tables systèmes de taille fixe :
MS-DOS : boot, FAT1, FAT2, DIR
Unix : boot, SF type, i-node table

Problème de l'allocation dynamique (=new, malloc) lors d'une demande de n blocs libres, le système doit chercher un **trou** (suite de blocs libres) suffisament grand parmi les trous de l'espace disque.

Stratégies
• Premier trouvé ("**First-Fit**") : on parcours la liste des trous jusqu'à en avoir trouvé un assez grand.
• Plus Petit ("**Best-Fit**") : on cherche dans **toute** la liste la borne supérieure des trous => parcours intégral ou liste des trous **triée** et recherche dicho.
• Plus Grand ("Worst-Fit") : inverse de Best-Fit.

Des simulations ont prouvées la meilleure performance de BF et FF en temps et en espace utilisé. Lorsque l'espace libre est suffisant mais qu'il n'existe pas de trou assez grand, cette **fragmentation externe** implique un compactage des fichiers.

Allocation chaînée

Les blocs de données d'un fichier contiennent un pointeur sur le bloc suivant. Le noeud descripteur contient, lui, la **tête** de liste. Lors de la création, la tête est mise à **nil**, puis, au fur et à mesure des demandes, une allocation contigüe (optimale en temps d'accès) est tentée. Si celle-ci ne peut avoir lieu, on choisit le(s) premier bloc disponible.

Avantage
• plus de fragmentation externe donc seule limitation = taille espace disque libre.

Inconvénients
• Accès séquentiel seulement !

• **fragmentation interne** des fichiers nécessitant de nombreux mouvements de tête. Des utilitaires de compactage permettent la réorganisation périodique des fichiers en allocation contigüe.

• fiabilité : si un pointeur est détruit logiciellement ou matériellement (bloc illisible) on perd tout le reste du fichier.

• encombrement disque dû aux pointeurs si blocs de petite taille

Allocation chaînée déportée

On déporte le chaînage dans une table système (FAT pour MS-DOS) dont chaque entrée pointe sur le bloc de données suivant du fichier. La tête de liste est positionnée dans le noeud descripteur. Cette table de chaînage est chargée en MC (volume courant) afin de permettre des accès directs rapides.

Avantage
• Accès direct et séquentiel peu coûteux

Inconvénients
• fragmentation interne (compactages périodiques)
• accès direct après parcours de la liste en MC

Allocation indirecte (indexée)

Un **bloc d'index** contient la liste des pointeurs sur les blocs de données du fichier. Ce bloc d'index est lui-même pointé par un champ du noeud descripteur de fichier. A la création le bloc d'index est initialisé à nil, nil, …, nil et lors des écritures successives, les premiers pointeurs sont affectés des adresses des blocs alloués par le gestionnaire de mémoire disponible. Ici encore la contiguïté est tentée ! A l'ouverture, le bloc index est chargé en MC afin de permettre les accès.

Avantage
• accès direct et séquentiel rapide

Inconvénients
• fragmentation interne => compactages périodiques
• taille gaspillée dans les blocs d'index >> taille des pointeurs en alloc chaînée.
• petits fichiers : pour 2 ou 3 adresses de blocs, on monopolise un bloc index !
• très grands fichiers : un bloc index étant insuffisant, plusieurs blocs index peuvent être chaînés et un mécanisme de multiple indirection peut être utilisé.

Allocation directe

Le **noeud descripteur** contient tous les pointeurs sur les blocs de données. De taille fixe et petite, cette structure de données ne pourra être utilisée que pour des fichiers de faible volume.

Avantage
• accès direct et séquentiel très rapide
• peut être mixtée avec indirection ou chaînage

Inconvénients
• taille limitée des fichiers
• fragmentation interne

8.3 Exemples

8.3.1 Le SGF MS-DOS

Nous décrivons ci-après l'**organisation physique** d'un volume MS-DOS sous la forme d'une disquette 5,25 pouces. Les principes d'allocation de MS-DOS restent les-mêmes quel que soit le support adressable, seuls le nombre et la taille des champs varient.

Une disquette, ou disque souple ou "floppy disk", est constituée d'un support plastique mince de forme discale d'un rayon de 5,25 pouces (1 pouce = 2,54 cm) sur lequel a été déposé un substrat magnétique. Dans le cas de disquettes double face double densité, cette surface homogène de particules magnétisables est structurée en 2 faces de **40 pistes** possédant chacune **9 secteurs**. Les pistes concentriques sont numérotées de 0 à 39 depuis l'extérieur vers l'intérieur. Chaque secteur contient **512** octets utiles. La capacité d'une disquette est donc de 2 * 40 * 9 * 512 = 360 Koctets.

L'unité d'allocation ("**cluster**"), ou bloc, est la plus petite partie d'espace mémoire allouable sur une disquette. Pour une disquette 5,25 pouces à 360 Ko, un **bloc** est constitué de **deux secteurs consécutifs** et a donc une capacité d'**1 Ko**. La numérotation des secteurs est effectuée de la façon suivante :

Face 0Piste 0 Secteurs 0 à 8
Face 1 Piste 0 Secteurs 9 à 17
Face 0 Piste 1 Secteurs 18 à 26
Face 1 Piste 1 Secteurs 27 à 35 etc…

Fichier et catalogue

Un **fichier** MS-DOS est une **suite d'octets** désigné par un identifiant composé d'un **nom de 8 caractères** et d'une **extension de 3 caractères**. Par exemple, COMMAND.COM, PRG1.PAS, SALAIRES.DBF, TEXTE1.DOC…

Un **catalogue** ("directory") MS-DOS est un type de fichier particulier permettant de regrouper différents fichiers de données et/ou d'autres catalogues dans une même entité. L'architecture du Système de Fichiers est donc une arborescence dont toutes les feuilles sont des fichiers et tous les nœuds intermédiaires des catalogues. Le catalogue **racine** est désigné par un "backslash" \ et est stocké en début de disquette.

Allocation

Les fichiers sont fragmentés en **allocation chaînée déportée** et MS-DOS conserve dans une table l'adresse des différents fragments. Cette table s'appelle Table d'Allocation des Fichiers ("File Allocation Table") et sera désignée par la suite par **FAT**. **Deux copies** de la FAT sont stockées sur la disquette (secteurs 1,2 et 3,4) afin de garantir la sécurité des données en cas de destruction accidentelle d'une des deux copies. Le **catalogue général (racine)** de la disquette est lui situé sur les secteurs 5 à 11. Le secteur 0, quant à lui, contient le programme d'amorçage sur les disquettes systèmes ("**boot-strap**"). Enfin les secteurs 12 à 719 contiennent les données des différents fichiers.

catalogue racine

C'est une table dont les entrées ("**File Control Block**") ont une longueur de 32 octets qui décrivent les fichiers et les sous-catalogues. Une entrée de répertoire peut être schématisée de la façon suivante :

FCB

<table>
<tr><td>0..7</td><td colspan="4">nom du fichier sur 8 octets</td></tr>
<tr><td>8..15</td><td colspan="2">extension sur 3 octets</td><td>attribut</td><td>réservé par MS-DOS</td></tr>
<tr><td>16..23</td><td colspan="3">réservé par MS-DOS</td><td>heure modificat</td></tr>
<tr><td>24..31</td><td>date modificat.</td><td>1° entrée FAT</td><td colspan="2">Taille fichier (LSB, MSB)</td></tr>
</table>

L'octet d'**attribut** permet de spécifier certaines **protections** : fichier caché, lecture seulement, archive, système, normal ou encore de préciser que le fichier est un catalogue. L'heure et la date de dernière modification permettent de retrouver la dernière version d'un fichier de travail. La **taille** du fichier est codé sur **32 bits** ce qui permet une taille maximale de **4 Giga-octets** ! Ce qui est bien entendu **impossible** sur une disquette de 360 Ko. Enfin, la 1ère entrée dans la FAT permet d'indiquer le **premier maillon du chaînage** dans la FAT qui pointera lui-même sur le second qui pointera sur le troisième etc…

la FAT 12 bits

C'est une table d'entrées d'une longueur de **12 bits** (1,5 octets) qui spécifie le chaînage permettant de reconstituer un fichier fragmenté sur plusieurs blocs. Les deux premières entrées (0 et 1) de cette table sont réservées par le système pour préciser le **type de la disquette** sur laquelle elle se trouve (simple/double face, simple/double densité). L'entrée numéro 2 correspond au premier **bloc** de la disquette **non réservé** au système, c'est-à-dire le bloc n° 6 (secteurs 12 et 13). Ainsi le fichier dont le "1° entrée FAT" de son entrée de répertoire est 2 verra son premier bloc de données situé sur les secteurs 12 et 13. Dans cette première entrée FAT on trouve la valeur de l'entrée suivante, par exemple 005, qui correspond au bloc de données suivant.

exemple de FAT

<table>
<tr><td>type disquette</td><td>005</td><td>000</td><td></td><td>i</td><td></td><td>…</td><td>FFF</td><td>…</td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>…</td><td>i</td></tr>
<tr><td></td><td colspan="2">1° bloc du fichier = bloc n° 6</td><td></td><td colspan="2">2° bloc du fichier = bloc n° 9</td><td></td><td colspan="2">indique la fin du chaînage</td></tr>
</table>

Dans cet exemple, le fichier est contenu sur trois blocs (6, 9 et i+4). La **fin du chaînage** est indiqué par la valeur 0FFFH (nil) tandis qu'une valeur 000 indique un bloc libre. La **gestion des blocs libres** (table de "12 bits") est en effet couplé au mécanisme de FAT.

Deux remarques

- Les entrées du répertoire racine ne sont **pas compactées** et la valeur 0E5H située sur le **premier octet** d'un FCB signifie que cette entrée est libre. Soit cette entrée n'a jamais été utilisée pour décrire un fichier ou un catalogue, soit ce fichier ou ce catalogue a été détruit (**delete** ou rmdir) et le système a simplement **surchargé** le premier octet du nom du fichier par un code 0E5H. Par conséquent, la recherche d'un nom dans le catalogue général est effectuée séquentiellement sur toutes les entrées. On aurait pu tout aussi bien choisir de compacter les entrées du catalogue à chaque destruction de fichier, ce qui aurait diminué le temps de recherche d'un fichier n'existant pas ! Mais cela aurait interdit les utilitaires de récupération de fichiers détruits (undelete).
- La seconde remarque concerne la **fragmentation** des fichiers sur la disquette. Au bout d'un certain nombre de créations et de destructions de fichiers et de répertoires, les différents blocs supportant les données d'un fichier se trouvent être disséminés sur la disquette. Or le transfert en mémoire centrale de tous les blocs de ce fichier va nécessiter un grand nombre de mouvements de translations du bras de lecture. C'est pourquoi il existe des utilitaires de **compactage** des blocs des fichiers d'une disquette permettant de minimiser les temps d'accès à un fichier.

La VFAT 32

Nouveau standard de FAT permettant de gérer des disques durs de grande capacité (>2Go) :
- Entrées de FAT sur 12, 16 ou 32 bits permettant des petits clusters ;
- FCB sur 32, 37 ou 44 octets (FCB étendu).

Exemple : disquette 1.4 Mo

Volume :
- 80 pistes/face : 160 pistes
- 18 secteurs/piste : 2880 secteurs
- 512 octets/secteurs : 1440 Ko soit 1.40 Mo
- 1 secteur/cluster

FAT
- 1 FAT : 9 secteurs * 2 copies ;
- Entrée de FAT : 12 bits ;
- 3072 entrées (2880 clusters)

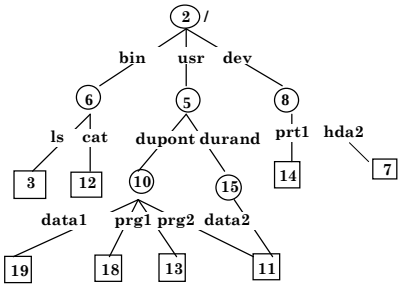
Répertoire racine :
- Répertoire racine sur 14 secteurs
- FCB sur 32 octets
- soit 224 entrées

Taux de transfert : 500 Kbits/s

Les noms longs de fichiers de WinX sont stockés dans le FCB suivant immédiat codé en Unicode.

8.3.2 Le SGF d'Unix

Exemple d'une arborescence de fichiers Unix



• catalogues : bin, usr, dev, dupont, durant;
• fichiers ordinaires : ls, cat , data1, prg1, prg2, data2;
• fichiers périphériques : sous le catalogue dev ("device") : prt1, dsk2.

Caractéristiques
• Entrées/Sorties généralisées ou transparentes
• désignation : chemin d'accès absolu /..., ou relatif
• fichiers **non structurés** = suite d'octets numérotés logiquement de 0 à n-1 (n = longueur du fichier) : structuration à la charge des programmeurs
• accès **direct** à une suite d'octet à partir d'une position i dans le fichier ($0 \leq i \leq n-1$) **et/ou** accès **séquentiel**
• catalogues Unix = liste (nom de fichier, # i-node)
• identification = # i-node
• désignations multiples : compteurs de liens ou de références
• système de protection : **rwx rwx rwx** propriétaire groupe autres

9. Les processus Unix

9.1 Généralités

Processus : suite temporelle d'exécutions d'instructions d'un programme par un processeur. (programme : données + suite d'instructions)

La gestion des processus (pus) étant très dépendante du SE étudié, nous nous bornerons à définir quelques notions générales avant d'aborder plus particulièrement les pus Unix.

nom d'un pus : où numéro d'identification du pus qui permet sa manipulation par le système et par l'utilisateur l'ayant créé.

ressources : emplacements de mémoire centrale, périodes d'utilisation de l'UC, périphériques … nécessaires à un processus pour son évolution.

état : un processus disposant de toutes les ressources (UC, MC, …) nécessaires à l'exécution de sa prochaine instruction est dans l'état actif. Dans tous les autres cas, il est bloqué sur la ou les ressources manquantes.

Remarque : l'observation des ressources courantes d'un processus actif ne peut être fait qu'entre deux instructions (atomicité).

ressource **locale** à un pus i : si elle ne peut être utilisée que par le pus i. ex : variables du prog.

ressource **commune** : si elle n'est locale à aucun pus. ex : tube

partageabilité : une ressource commune est **critique** (resp. partageable à n=2 points d'accès) si sur un point observable elle ne peut être détenue que par un (resp. n) pus au plus. ex : l'UC est critique; une zone mémoire tube est partageable à 2 points d'accès.

Plusieurs pus sont dits en **exclusion mutuelle** sur r lorsque ils utilisent cette ressource critique r.

mode : niveau de pouvoir (droits) dans lequel s'exécute le pus lui permettant ou non d'accéder à certaines ressources et/ou d'exécuter certaines instructions privilégiées de l'UC. ex : dans de nombreux systèmes deux modes (seulement) existent : mode maître (ou système ou **noyau**)/mode esclave (ou **utilisateur**). Le mode d'un pus peut être statiquement donné à la création ou évoluer dynamiquement (Unix).

durée de vie : un pus naît, après chargement en MC, lors du lancement du programme par le SE et meurt à la fin de l'exécution de ce programme lors du retour au SE.

Les mécanismes de **synchronisation** permettent notamment l'activation d'un pus bloqué sur une ressource ou au contraire le blocage d'un pus actif. ex : l'accès de 2 pus à une ressource critique nécessite leur synchronisation afin qu'un seul d'entre eux n'obtienne la ressource.

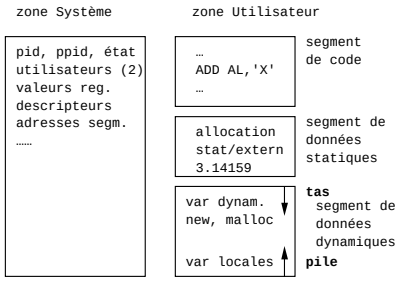
Les mécanismes de **communication** permettent à plusieurs pus de se transmettre des données. Afin d'éviter la communication par fichier (E/S lentes) on utilise des structures de données en mémoire centrale : variables partagées, tubes, signaux, files de messages, sockets,…

Le **recouvrement** (overlay) est une technique qui permet de remplacer une partie de la mémoire centrale par une autre. ex : un programme très long peut être décomposé en parties se recouvrant pour diminuer l'espace utilisé (quasiment plus utilisé car grande mémoires et pagination).

10. Assembleur

9.2 Description des processus Unix

image mémoire d'un pus : ensemble des zones mémoires utilisées par un pus :



Un processus Unix réalise ses instructions normales en **mode utilisateur** puis commute en **mode système** lors d'un appel au noyau, d'une interruption, ou d'un déroutement.

La commutation de pus est toujours effectuée en mode système par le pus "partant" (pas de préemption).