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.

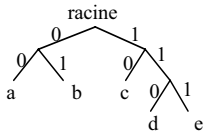
Pour calculer le code d'un alphabet et de sa distribution, on construit un arbre binaire étiqueté par des 0 et des 1, les feuilles de cet arbre représentant les caractères tandis que les chemins issus de la racine constituent les codes correspondants.

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

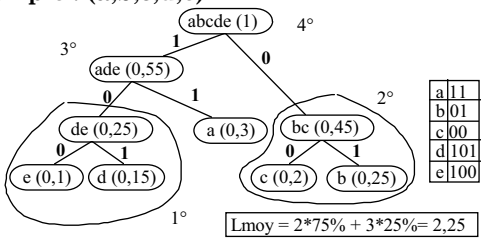
Remarquons que ce code n'est pas unique (il suffit de permuter les 0 et les 1). On calcule la longueur moyenne de codage comme suit :

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

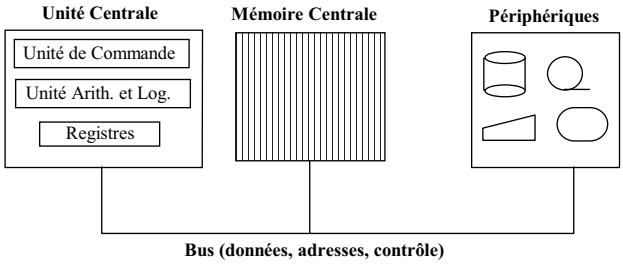
e	x	e	m	p	l	e		d	e		c	h	a	î	n	e	\0

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

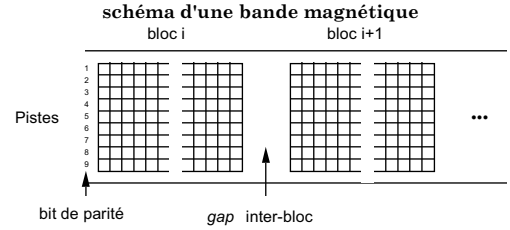
Les mémoires électroniques étant chères et soit volatiles (non fiables) soit non réinscriptibles, le stockage de masse est réalisé sur d'autres supports. Ces autres supports sont caractérisés par :
- non volatilité et réinscriptibilité
- faible prix de l'octet stocké
- lenteur d'accès et modes d'accès (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.

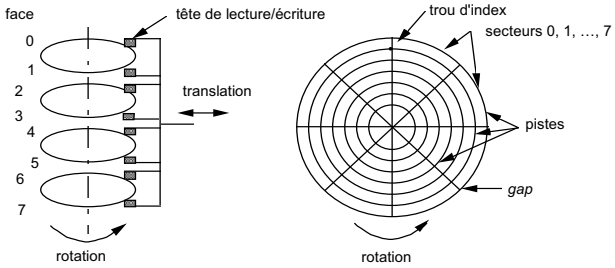
Bandes magnétiques
Ce sont des supports à accès **séquentiel** particulièrement utilisés dans la sauvegarde. On les utilise de plus en plus sous forme de **cassettes**. Les dérouleurs de cassettes sont appelés *streamers*. Les densités et vitesses sont variables : quelques milliers de *bytes per inch (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)