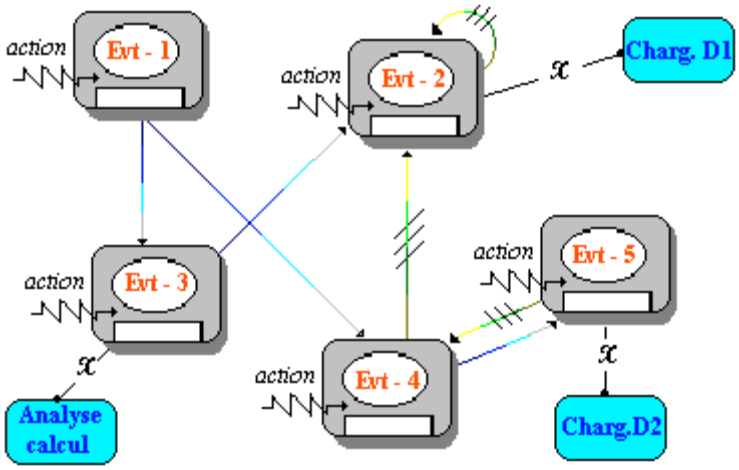
Les bases de l'informatique et de la programmation



Le contenu de ce livre pdf de cours d'initiation à la programmation est inclus dans un ouvrage papier de 1372 pages édité en Novembre 2004 par les éditions Berti à Alger.

http://www.berti-editions.com

L'ouvrage est accompagné d'un CD-ROM contenant les assistants du package pédagogique.

Rm di Scala

Corrections du 04.01.05

SOMMAIRE

Chapitre 4. Structures de données

Chapitre 5. Programmation objet et événementielle

Chapitre 6. Programmez avec des grammaires

Chapitre 7. Communication homme-machine

Chapitre 8. Les composants sont des logiciels réutilisables

Annexes

Introduction

Issu d'un cours de programmation à l'université de Tours en premier cycle scientifique, en DESS, Master Sciences et technologie compétence complémentaire informatique et en Diplôme Universitaire (DU) compétence complémentaire informatique pour les NTIC (réservés à des non-informaticiens), cet ouvrage est une synthèse (non exhaustive)sur les minima à connaître sur le sujet.

Il permettra au lecteur d'aborder la programmation objet et l'écriture d'interfaces objets événementielles sous Windows en particulier.

Ce livre sera utile à un public étudiant (IUT info, BTS info, IUP informatique et scientifique, DEUG sciences, licence pro informatique, Dess, Master et DU compétence complémentaire en informatique) et de toute personne désireuse de se former par elle-même (niveau prérequis Bac scientifique).

Le premier chapitre rassemble les concepts essentiels sur la notion d'ordinateur, de codage, de système d'exploitation, de réseau, de programme et d'instruction au niveau machine.

Le second chapitre introduit le concept de langage de programmation et de grammaire de chomsky, le langage pascal de Delphi sert d'exemple.

Le chapitre trois forme le noyau dur d'une approche méthodique pour développer du logiciel, les thèmes abordés sont : algorithme, complexité, programmation descendante, machines abstraites, modularité. Ce chapitre fournit aussi des outils de tris sur des tableaux. montre comment utiliser des grammaires pour programmer en mode génération ou en mode analyse.

Le chapitre quatre défini la notion de types abstraits. Il propose l'étude de type abstrait de structures de données classiques : liste, pile, file, arbre avec des algorithmes classiques de traitement d'arbres binaires.

Le chapitre cinq contient les éléments fondamentaux de la programmation orientée objet, du polymorphisme d'objet, du polymorphisme de méthode, de la programmation événementielle et visuelle, de la programmation défensive. Le langage Delphi sert de support à l'implantation pratique de ces notions essentielles.

Le chapitre six montre comment utiliser la programmation par les grammaire avec des outils pratiques comme les automates de type 3 et les automates à piles simples. Deux projets complets sont traités dans ce chapitre.

Le chapitre sept correspond la construction d'interface homme-machine, et à l'utilisation des bases de données avec des exemples pratiques en Delphi et Access.

Le chapitre huit composant avec Delphi, puis aborde le traitement des messages dans Windows et comment programmer des applications utilisant les messages système pour communiquer. Il fournit aussi une notice pratique pour construire un composant ActiveX et le déployer sur le web avec Delphi.

Chapitre 1 : La machine

1.1.Ordinateur et évolution

- les 3 grandes lignes de pensées
- les générations d'ordinateurs
- l'ordinateur
- information-informatique

1.2.Les circuits logiques

- logique élémentaire pour l'informatique
- algèbre de Boole
- circuits booléens

1.3.Codage numération

- codage de l'information
- numération

1.4.Formalisation de la notion d'ordinateur

- machine de Türing théorique
- machine de Türing physique

1.5.Architecture de l'ordinateur

- les principaux constituants
- mémoires, mémoire centrale
- une petite machine pédagogique

1.6.Système d'exploitation

- notion de système d'exploitation
- systèmes d'exploitation des micro-ordinateurs

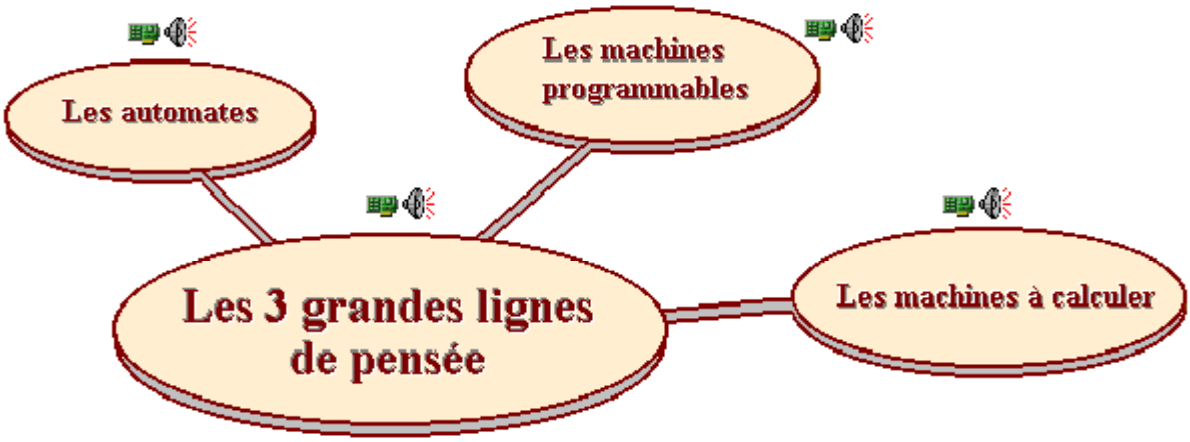
1.7.Les réseaux

- les réseaux d'ordinateurs
- liaisons entre réseaux

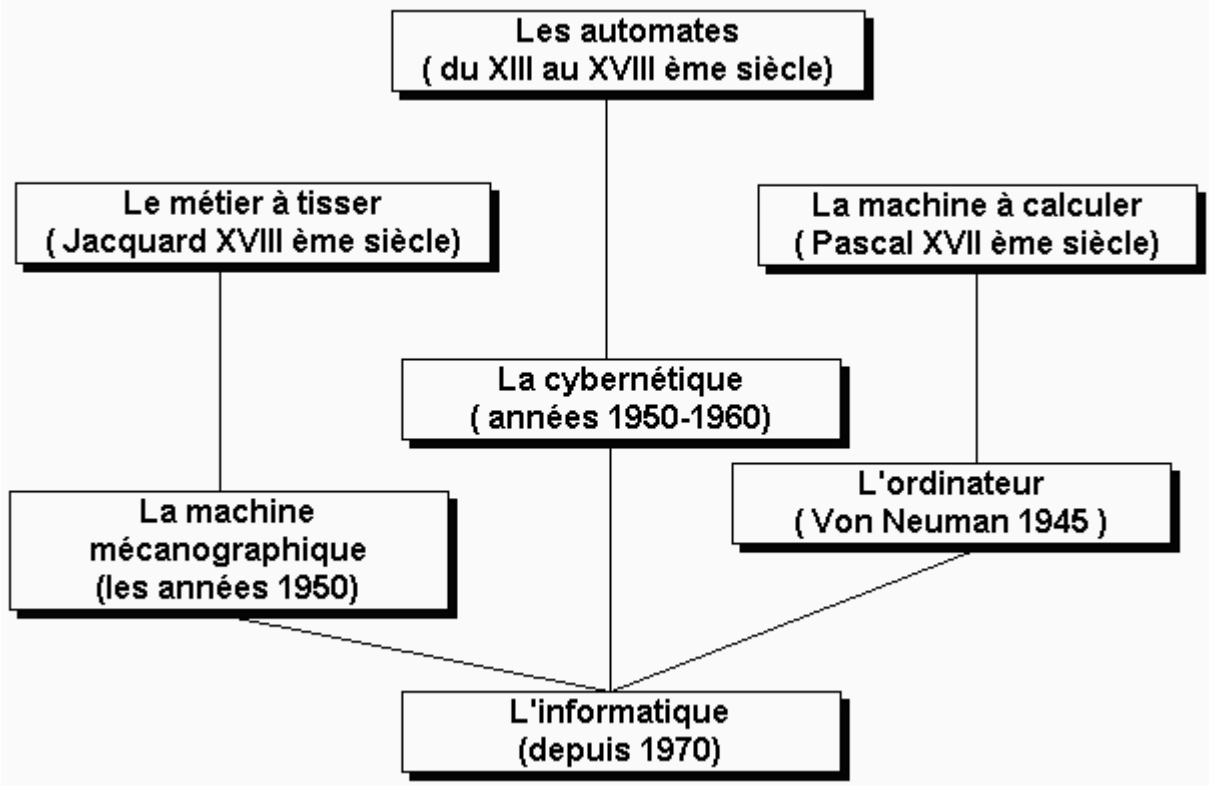
1.1 Ordinateur et évolution

Plan du chapitre:

1. Les 3 grandes lignes de pensée



L'histoire de l'informatique débute par l'invention de machines (la fonction crée l'organe) qui au départ correspondent à des lignes de pensée différentes. L'informatique résultera de la fusion des savoirs acquis dans ces domaines. Elle n'est pas une synthèse de plusieurs disciplines, mais plutôt une discipline entièrement nouvelle puisant ses racines dans le passé. Seul l'effort permanent du génie créatif humain l'a rendue accessible au grand public de nos jours.



1.1 Les machines à calculer

La Pascaline de Pascal, 17ème siècle. Pascal invente la Pascaline, première machine à calculer (addition et soustraction seulement), pour les calculs de son père.

La machine multiplicatrice de Leibniz, 17ème siècle. Leibniz améliore la machine de Pascal pour avoir les quatre opérations de base (+,-,*,/).

1.2 Les automates

Les automates, les horloges astronomiques, les machines militaires dès le 12ème siècle.

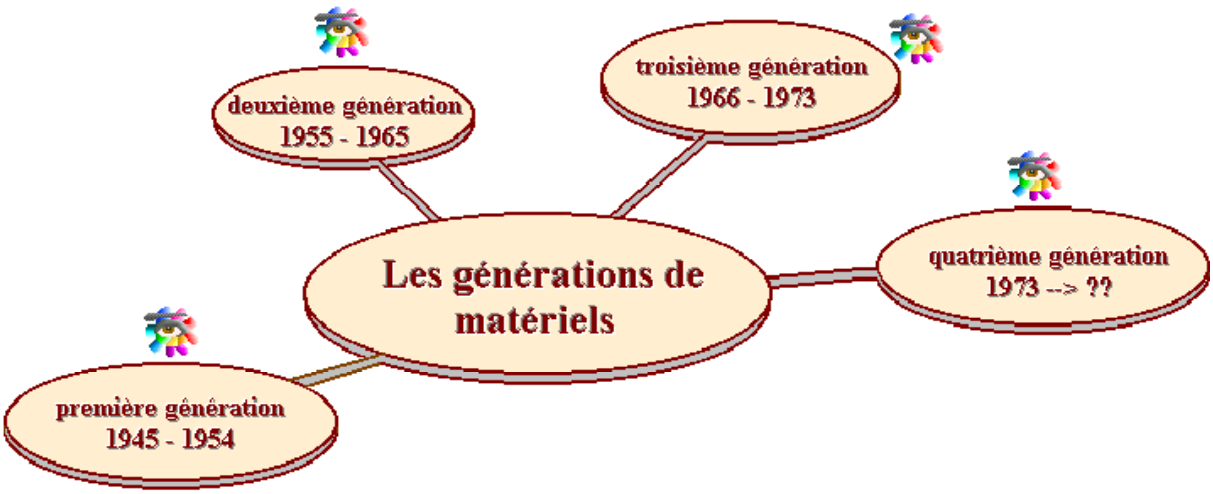
1.3 Les machines programmables

Le métier à tisser de **Jacquard**, 1752-1834

Début de commercialisation des machines mécaniques scientifiques (usage militaire en général).

Babage invente la première machine analytique programmable.

2. Les générations de matériels



On admet généralement que l'ère de l'informatique qui couvre peu de décennies se divise en plusieurs générations essentiellement marquées par des avancées technologiques

2.1 Première génération 1945 - 1954

Informatique scientifique et militaire.

Il faut résoudre les problèmes des calculs répétitifs.

Création de langages avec succès et échecs dans le but de résoudre les problèmes précédents.

Technologie lourde (Tube et tore de ferrite), qui pose des problèmes de place et de consommation électrique.

Les très grandes nations seules possèdent l'outil informatique.

2.2 Deuxième génération 1955-1965

Naissance de l'informatique de gestion.

Nouvelle technologie basée sur le transistor et le circuit imprimé. Le langage **Fortran** règne en maître incontesté. Le langage de programmation **Cobol** orienté gestion, devient un concurrent de **Fortran**.

Les nations riches et les très grandes entreprises accèdent à l'outil informatique.

2.3 Troisième génération 1966-1973

Naissance du circuit intégré.

Nouvelle technologie basée sur le **transistor** et le circuit intégré.

Les ordinateurs occupent moins de volume, consomment moins d'électricité et sont plus rapides. Les ordinateurs sont utilisés le plus souvent pour des applications de gestion.

Les PME et PMI de tous les pays peuvent se procurer des matériels informatiques.

2.4 Quatrième génération à partir de 1974

Naissance de la micro-informatique

La création des microprocesseurs permet la naissance de la micro-informatique(le micro-ordinateur Micral de R2E est inventé par un français **François Gernelle** en 1973). Steve Jobs (Apple) invente un nouveau concept vers la fin des années 70 en recopiant et en commercialisant les idées de Xerox parc à travers le MacIntosh et son interface graphique.

Un individu peut actuellement acheter son micro-ordinateur dans un supermarché.

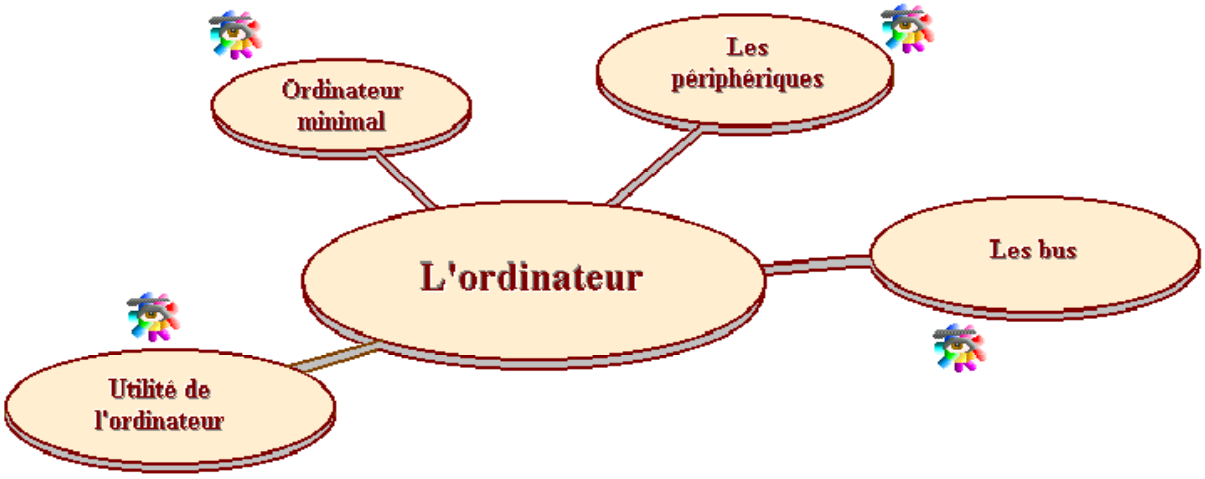
Nous observons un phénomène fondamental :

La démocratisation d'une science à travers un outil. L'informatique qui à ses débuts était une affaire de spécialistes, est aujourd'hui devenue l'affaire de tous; d'où l'importance d'une solide formation de tous aux différentes techniques utilisées par la science informatique, car la banalisation d'un outil ou d'une science a son revers : l'assoupissement de l'attention envers les inconvénients inhérents à tout progrès technique.

Tableau synoptique des générations d'ordinateurs :

	1ère Génér. 45 - 54	2ème Génér. 55 - 65	3ème Génér. 66 - 73	4ème Génér. 74 ---> ?
composants	tubes radios	transistor	circuit intégré	VLSI
mémoires	tubes/ tores de ferrite	tores de ferrite	circuit intégré	VLSI
temps de traitement	10^{-2} s	10^{-3} s	10^{-6} s	10^{-9} s
système d'exploitation	rudimentaire	monoprogram -mation	multiprogram -mation	temps-partagé
rendement %	3 %	10%	30 %	60 %

3. L'ordinateur



3.1 Utilité de l'ordinateur

Un ordinateur est une machine à traiter de l'information.

L'information est fournie sous forme de données traitées par des programmes (exécutés par des ordinateurs).

3.2 Composition minimale d'un ordinateur : **le cœur**

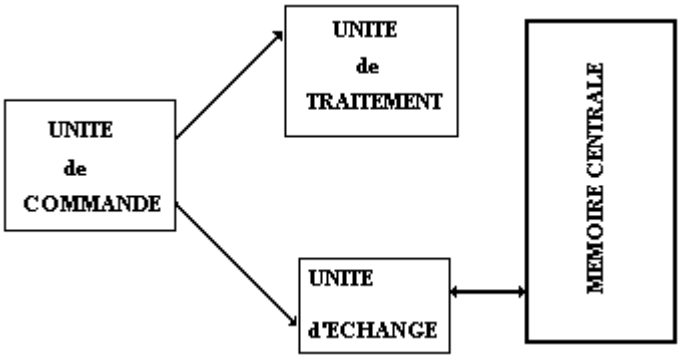
Une mémoire Centrale .

Une unité de traitement avec son UAL (unité de calcul).

Une unité de commande ou contrôle.

Une ou plusieurs unités d'échanges.

Schéma simplifié du cœur de l'ordinateur



3.3 Autour de l'ordinateur : les périphériques

Les périphériques sont chargés d'effectuer des tâches d'entrées et/ou de sortie de l'information. En voici quelques uns.

Périphériques d'entrée
Clavier, souris, crayon optique, écran tactile, stylo code barre, carte son, scanner, caméra, etc.

Périphériques de sortie
Ecran, imprimante, table traçante, carte son, télécopie, modem etc.

Périphériques d'entrée sortie
Mémoire auxiliaire (sert à stocker les données et les programmes):

1. Stockage de masse sur disque dur ou disquette.
2. Bande magnétique sur dérouleur (ancien) ou sur streamer.
3. Mémoire clef USB
4. CD-Rom, DVD, disque magnéto-électrique etc…

3.4 Pour relier tout le monde : Les Bus

Les Bus représentent dans l'ordinateur le système de communication entre ses divers constituants. Ils sont au nombre de trois :

le **Bus d'adresses**, (la notion d'adresse est présentée plus loin)

le **Bus de données**,

le **Bus de contrôle**.

4. Information - informatique

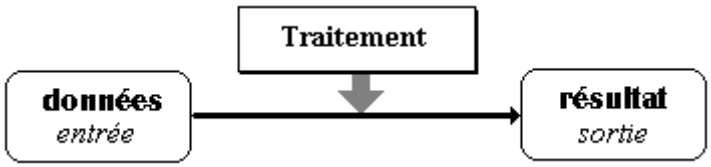
4.1 Les définitions

L'**information** est le support formel d'un élément de connaissance humaine susceptible d'être représentée à l'aide de conventions (codages) afin d'être conservée, traitée ou communiquée.

L'**informatique** est la science du traitement de l'information dans les domaines scientifiques, techniques, économiques et sociaux.

Une **donnée** est la représentation d'une information sous une forme conventionnelle (codée) destinée à faciliter son traitement.

schéma simplifié du traitement de l'information



4.2 Critère algorithmique élémentaire

Une application courante est justiciable d'un traitement informatique si :

Il est possible de définir et de décrire parfaitement les données d'entrée et les résultats de sortie.

Il est possible de décomposer le passage de ces données vers ces résultats en une suite d'opérations élémentaires dont chacune peut être exécutée par une machine.

Nous pouvons considérer ce critère comme une définition provisoire d'un algorithme.

Actuellement l'informatique intervient dans tous les secteurs d'activité de la vie quotidienne :

démontrer un théorème *(mathématique)*
faire jouer aux échecs *(intelligence artificielle)*
dépouiller un sondage *(économie)*
gérer un robot industriel *(atelier)*
facturation de produits *(entreprise)*
traduire un texte *(linguistique)*
imagerie médicale *(médecine)*
formation à distance *(éducation)*
Internet *(grand public)*...etc

1.2 Les circuits logiques

Plan du chapitre:

1. Logique élémentaire pour l'informatique

1.1 Calcul propositionnel naïf

Construire des programmes est une activité scientifique fondée sur le raisonnement logique. Un peu de logique simple va nous aider à disposer d'outils pratiques mais rigoureux pour construire des programmes les plus justes possibles. Si la programmation est un art, c'est un art rigoureux et logique. La rigueur est d'autant plus nécessaire que les systèmes informatiques manquent totalement de sens artistique.

Une proposition est une propriété ou un énoncé qui peut avoir une valeur de vérité vraie (notée **V**) ou fausse (notée **F**).

" 2 est un nombre impair " est une proposition dont la valeur de vérité est **F**.

Par abus de langage nous noterons avec **le même symbole une proposition et sa valeur de vérité**, car seule la valeur de vérité d'une proposition nous intéresse ici.

Soit l'ensemble P = { **V , F** } des valeurs des propositions.

On le munit de trois opérateurs appelés connecteurs logiques : $\neg$, $\wedge$, $\vee$**.**

$\wedge$: P x P$\rightarrow$P (se lit " **et** ")

$\vee$: P x P$\rightarrow$P (se lit " **ou** ")

$\neg$: P $\rightarrow$P (se lit " **non** ")

Ces connecteurs sont définis en extension par leur tables de vérité :

p	q	$\neg$ p	p $\wedge$ q	p $\vee$ q
V	**V**	**F**	**V**	**V**
V	**F**	**F**	**F**	**V**
F	**V**	**V**	**F**	**V**
F	**F**	**V**	**F**	**F**

1.2 Propriétés des connecteurs logiques

- Nous noterons p = q , le fait la proposition p et la proposition q ont la même valeur de vérité.
- Le lecteur pourra démontrer à l'aide des tables de vérité par exemple, que $\vee$ et $\wedge$ possèdent les propriétés suivantes :
    - $p \vee q = q \vee p$
    - $p \wedge q = q \wedge p$
    - $p \vee (q \vee r) = (p \vee q) \vee r$
    - $p \wedge (q \wedge r) = (p \wedge q) \wedge r$
    - $p \vee (q \wedge r) = (p \vee q) \wedge (p \vee r)$
    - $p \wedge (q \vee r) = (p \wedge q) \vee (p \wedge r)$
    - $p \vee p = p$
    - $p \wedge p = p$
    - $\neg\neg p = p$
    - $\neg (p \vee q) = \neg p \wedge \neg q$
    - $\neg (p \wedge q) = \neg p \vee \neg q$
- Nous notons $p \Rightarrow q$, la proposition : $\neg p \vee q$ (l'implication).

Table de vérité du connecteur $\Rightarrow$:

p	q	$p \Rightarrow q$
V	**V**	**V**
V	**F**	**F**
F	**V**	**V**
F	**F**	**V**

- Il est aussi possible de prouver des " égalités " de propositions en utilisant des combinaisons de résultats précédents.

Exemple : Montrons que : $p \Rightarrow q = \neg q \Rightarrow \neg p$ (implication contrapposée), par définition et utilisation évidente des propriétés :

$p \Rightarrow q = \neg p \vee q = q \vee \neg p = \neg (\neg q) \vee \neg p = \neg q \Rightarrow \neg p$

1.3 Règles de déduction

Assertion :
c'est une proposition construite à l'aide des connecteurs logiques ($\neg$, $\wedge$, $\vee$, en particulier) dont la valeur de vérité est toujours V (vraie).

Les règles de déduction permettent de faire du calcul sur les *assertions*. Nous abordons ici le raisonnement rationnel sous son aspect automatisable, en donnant des règles d'inférences extraites du modèle du raisonnement logique du logicien **Gentzen**. Elles peuvent être une aide très appréciable lors de la construction et la spécification d'un programme.

Les règles de déduction sont séparées en deux membres. Le premier contient les prémisses ou hypothèses de la règle, le deuxième membre est constitué par une conclusion unique. Les deux membres sont séparés par un trait horizontal. Gentzen classe les règles de déduction en deux catégories : les règles d'introduction car il y a utilisation d'un nouveau connecteur, et les règles d'éliminations qui permettent de diminuer d'un connecteur une proposition.

Syntaxe d'une telle règle : $\dfrac{p1, p2, \ldots, pn}{q}$

Quelques règles de déductions pratiques :

Règle d'introduction du $\wedge$: $\dfrac{p \ , \ q}{p \wedge q}$

Règle d'introduction du $\vee$: $\dfrac{p \ , \ q}{p \vee q}$

Règle d'introduction du $\Rightarrow$: $\dfrac{p \ , \ q}{p \Rightarrow q}$

Règles d'élimination du $\wedge$: $\dfrac{p \wedge q}{q}$, $\dfrac{p \wedge q}{p}$

Règle du modus ponens : $\dfrac{p \ , \ p \Rightarrow q}{q}$

Règle du modus tollens : $\dfrac{\neg q \ , \ p \Rightarrow q}{\neg p}$

Le système de **Gentzen** contient d'autres règles sur le **ou** et sur le **non**. Enfin il est possible de construire d'autres règles de déduction à partir de celles-ci et des propriétés des connecteurs logiques. Ces règles permettent de prouver la valeur de vérité d'une proposition. Dans les cas pratiques l'essentiel des raisonnements revient à des démonstrations de véracité d'implication.

La démarche est la suivante : pour prouver qu'une implication est vraie, il suffit de supposer que le membre gauche de l'implication est vrai et de montrer que sous cette hypothèse le membre de droite de l'implication est vrai.

Exemple **:**
soit à montrer que la règle de déduction $\mathbf{R_0}$ suivante est exacte :

R_0 : $\dfrac{p \Rightarrow q \ , \ \ q \Rightarrow r}{p \Rightarrow r}$ *(transitivité de ⇒)*

*Hypothèse : **p** est vrai*
nous savons que : **p** ⇒ **q** est vrai et que **q** ⇒ **r** est vrai

- En appliquant le modus ponens : $\dfrac{p \ , \ p \Rightarrow q}{q}$ nous déduisons que : **q** est vrai.
- En appliquant le modus ponens à (**q , q** ⇒ **r**) nous déduisons que : **r** est vrai.
- Comme p est vrai (par hypothèse) on applique la règle d'introduction de ⇒ sur (**p , r**) nous déduisons que : **p** ⇒ **r** est vrai **(cqfd).**

Nous avons prouvé que $\mathbf{R_0}$ est exacte. Ainsi nous avons construit une nouvelle règle de déduction qui pourra nous servir dans nos raisonnements.

Nous venons d'exhiber un des outils permettant la construction raisonnée de programmes. La logique interne des ordinateurs est encore actuellement plus faible puisque basée sur la logique booléenne, en attendant que les machines de 5ème génération basées sur la logique du premier ordre (logique des prédicats, supérieure à la logique propositionnelle) soient définitivement opérationnelles.

2. Algèbre de Boole

2.1 Axiomatique pratique

Du nom du mathématicien anglais **G.Boole** qui l'inventa. Nous choisissons une axiomatique compacte, l'axiomatique algébrique :

On appelle algèbre de Boole tout ensemble E muni de :

deux lois de compositions internes notées par exemple : • et ⊕,

- une application involutive f (f^2 = Id) de E dans lui-même,notée $f : a \longrightarrow \bar{a}$
- chacune des deux lois • , ⊕ , est associative et commutative,
- chacune des deux lois • , ⊕ , est distributive par rapport à l'autre,
- la loi • admet un élément neutre unique noté e_1,

  $x \in E,\ x \bullet e_1 = x$

- *la loi* ⊕ admet un élément neutre noté e_0,

  $x \in E,\ x \oplus e_0 = x$

- *tout élément de E est idempotent pour chacune des deux lois :*

  $x \in E,\ x \bullet x = x$ et $x \oplus x = x$

- *axiomes de complémentarité* :

  $\forall x \in E, x \bullet \bar{x} = e_0$
  $\forall x \in E, x \oplus \bar{x} = e_1$

- *lois de Morgan* :

  $(x,y) \in E^2,\ \overline{x \bullet y} = \bar{x} \oplus \bar{y}$

  $(x,y) \in E^2,\ \overline{x \oplus y} = \bar{x} \bullet \bar{y}$

2.2 Exemples d'algèbre de Boole

a) L'ensemble P(E) des parties d'un ensemble E, muni des opérateurs intersection $\cap$,union $\cup$, et l'application involutive complémentaire dans E $\rightarrow C_E$, (si E $\neq \varnothing$), si E est fini et possède n éléments, P(E) est de cardinal 2^n.

Il suffit de vérifier les axiomes précédents en substituant les lois du nouvel ensemble E aux lois $\bullet$, $\oplus$, et $\bar{x}$. Il est montré en mathématiques que toutes les algèbres de Boole finies sont isomorphes à un ensemble (P(E), $\cap$, $\cup$, C_E) : elles ont donc un cardinal de la forme 2^n.

b) L'ensemble des propositions (en fait l'ensemble de leurs valeurs {**V**, **F**}) muni des connecteurs logiques $\neg$ (l'application involutive) , $\wedge$, $\vee$ **, est une algèbre de Boole minimale à deux éléments.**

2.3 Notation des électroniciens

L'algèbre des circuits électriques est **une algèbre de Boole minimale à deux éléments** :

L'ensemble E = {**0**,**1**} muni des lois " $\bullet$ " et " + " et de l'application complémentaire $f : a \longrightarrow \bar{a}$.

Formules pratiques et utiles (résultant de l'axiomatique) :

a + 1 = 1 a + 0 = a a + a = a $a + \bar{a} = 1$ $\overline{a + b} = \bar{a} \cdot \bar{b}$ $\bar{0} = 1$	a.1 = a a.0 = 0 a.a = a $a \cdot \bar{a} = 0$ $\overline{a \cdot b} = \bar{a} + \bar{b}$ $\bar{1} = 0$

Formule d'absorbtion :

a+(b.a) = a.(a+b) = (a+b).a = a+b.a = a

Montrons par exemple : a+(b.a) = a

a+(b.a)= a+a.b = a.1+a.b = a.(1+b) = a.1 = a

Le reste se montrant de la même façon.

Cette algèbre est utile pour décrire et étudier les schémas électroniques, mais elle sert aussi dans d'autres domaines que l'électricité. Elle est étudiée ici parce que les ordinateurs actuels sont basés sur des composants électroniques. Nous allons descendre un peu plus bas dans la réalisation interne du cœur d'un ordinateur, afin d'aboutir à la construction d'un additionneur en binaire dans l'UAL.

Tables de vérité des trois opérateurs :

x	y	$\overline{x}$	$x \,.\, y$	$x + y$
1	**1**	**0**	**1**	**1**
1	**0**	**0**	**0**	**1**
0	**1**	**1**	**0**	**1**
0	**0**	**1**	**0**	**0**

3. Circuits booléens ou logiques

Nous représentons par une variable booléenne x ∈ {**0**,**1**} le passage d'un courant électrique.

Lorsque x = **0**, nous dirons que x est à l'état 0 (**le courant ne passe pas**)

Lorsque x = **1**, nous dirons que x est à l'état 1 (**le courant passe**)

Une telle variable booléenne permet ainsi de visualiser, sous forme d'un bit d'information (0,1) le comportement d'un composant physique laissant ou ne laissant pas passer le courant.

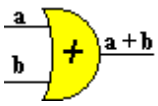
Nous ne nous préoccuperons pas du type de circuits électriques permettant de construire un circuit logique (les composants électriques sont basés sur les circuits intégrés). Nous ne nous intéresserons qu'à la fonction logique (booléenne) associée à un tel circuit.

En fait un circuit logique est un opérateur d'une algèbre de Boole c'est-à-dire une combinaison de symboles de l'algèbre {**0,1**}, **.** ,+, $\overline{x}$).

3.1 Principaux circuits

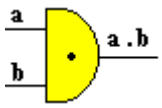
Nous proposons donc 3 circuits logiques de base correspondant aux deux lois internes et à l'opérateur de complémentation involutif.

Le circuit OU associé à la loi " + " :



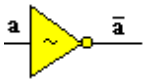
La table de vérité de ce circuit est celle de l'opérateur +

Le circuit ET associé à la loi "•" :



La table de vérité de ce circuit est celle de l'opérateur •

Le circuit NON associé à la loi " $\overline{x}$ " :



la table de vérité est celle de l'opérateur involutif $\overline{x}$

On construit deux circuits classiques à l'aide des circuits précédents :

L'opérateur XOR = " ou exclusif " :

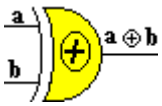
$a \oplus b = \overline{a}.b + a.\overline{b}$	dont voici le schéma : 

Table de vérité du ou exclusif :

a	b	$a \oplus b$
1	**1**	**0**
1	**0**	**1**
0	**1**	**1**
0	**0**	**0**

L'opérateur NAND (le NON-ET):

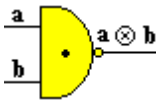
$a \otimes b = \overline{a.b} = \overline{a} + \overline{b}$	dont voici le schéma : 

Table de vérité du Nand :

a	b	$a \otimes b$
1	**1**	**0**
1	**0**	**1**
0	**1**	**1**
0	**0**	**1**

L'opérateur NOR (le NON-OU):

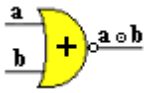
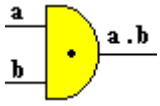
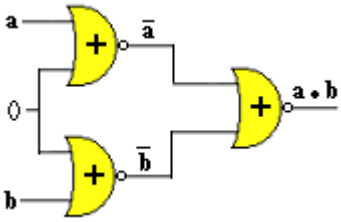
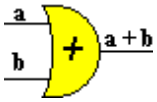
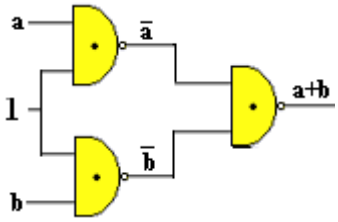
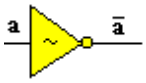
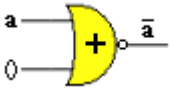
$a \circ b = \overline{a+b} = \bar{a}.\bar{b}$	dont voici le schéma : 

Table de vérité du Nor :

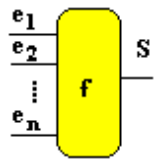
a	b	$a \Theta b$
1	**1**	**0**
1	**0**	**0**
0	**1**	**0**
0	**0**	**1**

L'on montre facilement que les deux opérateurs NAND et NOR répondent aux critères axiomatiques d'une algèbre de Boole, ils sont réalisables très simplement avec un minimum de composants électroniques de type transistor et diode (voir paragraphes plus loin). Enfin le NOR et le NAND peuvent engendrer les trois opérateurs de base **non**, **et** , **ou**. :

Opérateur de base	Réalisation de l'opérateur en NAND ou en NOR
 circuit ET	
 circuit OU	
 circuit NON	

Expression des 3 premiers opérateurs ($\bar{x}$, + , .) à l'aide de NAND et de NOR

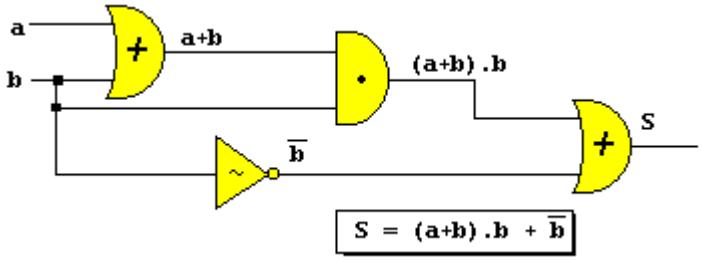
3.2 Fonction logique associée à un circuit



Un circuit logique est un système de logique séquentielle où la valeur de sortie S (état de la variable booléenne **S** de sortie) dépend des valeurs des entrées $\mathbf{e_1,e_2,...,e_n}$ (états des variables booléennes d'entrées e_i). Sa valeur de sortie est donc une fonction $\mathbf{S = f(e_1,e_2,...,e_n)}$.

Pour calculer la fonction **f** à partir d'un schéma de circuits logiques, il suffit d'indiquer à la sortie de chaque opérateur (circuit de base) la valeur de l'expression booléenne en cours. Puis, à la fin, nous obtenons une expression booléenne que l'on simplifie à l'aide des axiomes ou des théorèmes de l'algèbre de Boole.

Exemple :



En simplifiant S : (a+b).b+ $\overline{b}$ = b + $\overline{b}$ (formule d'absorbtion)
b + $\overline{b}$ = 1.

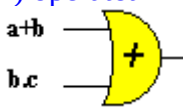
3.3 Circuit logique associé à une fonction

A l'inverse, la création de circuits logiques à partir d'une fonction booléenne **f** à n entrées est aussi simple. Il suffit par exemple, dans la fonction, d'exprimer graphiquement chaque opérateur par un circuit, les entrées étant les opérandes de l'opérateur. En répétant l'action sur tous les opérateurs, on construit un graphique de circuit logique associé à la fonction **f**.

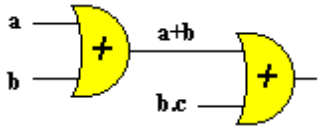
Exemple : Soit la fonction f de 3 variables booléennes, f (a,b,c) = (a+b)+(b•c)

Construction progressive du circuit associé.

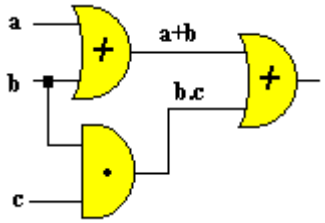
1°) opérateur " + " :



2°) branche supérieure de l'opérateur " + " :



3°) branche inférieure de l'opérateur " + " :

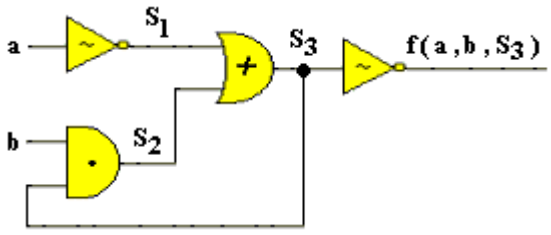


Les électroniciens classent les circuits logiques en deux catégories : les circuits combinatoires et les circuits séquentiels (ou à mémoire).

Un circuit combinatoire est un circuit logique à **n** entrées dont la fonction de sortie ne dépend uniquement que des variables d'entrées.

Un circuit à mémoire (ou séquentiel) est un circuit logique à **n** entrées dont la fonction de sortie dépend à la fois des variables d'entrées et des états antérieurs déjà mémorisés des variables de sorties.

Exemple de circuit à mémoire



f(a , b , S_3) = $\overline{S_3}$

La sortie intermédiaire S_2 est évaluée en fonction de la sortie S_3 : **S_2 = b . S_3** (valeur de S3 un temps avant)

Si b=0 alors $S_2 = 0$
Si b=1 alors $S_2 = S_3$
Nous avons $S_3 = S_1 + S_2$

En notant S'_3 la valeur au temps $\mathbf{t_0}$ et S_3 la valeur au temps $\mathbf{t_0+dt}$, il vient que $\mathbf{S_3 = S_1 + b \cdot S'_3}$

Table de vérité associée à ce circuit :

a	*b*	S_1	S_2	S_3	$f(a,b,S_3)$
1	**1**	**0**	**S'_3**	**S'_3**	**$\overline{S'_3}$**
1	**0**	**0**	**0**	**0**	**1**
0	**1**	**1**	**S'_3**	**1**	**0**
0	**0**	**1**	**0**	**1**	**0**

Dans le cas a=1 et b=1 ce circuit fictif fournit le complément de la valeur antérieure.

Quelques noms de circuits logiques utilisés dans un ordinateur

> Circuit combinatoire : *additionneur, multiplexeur, décodeur, décaleur, comparateur.*
> Circuit à mémoire : *bascules logiques.*

3.4 Additionneur dans l'UAL (circuit combinatoire)

a) Demi-additionneur

Reprenons les tables de vérités du " ⊕ " (*Xor*), du " + " et du " • " et adjoignons la table de calcul de l'addition en numération binaire.

Tout d'abord les tables comparées des opérateurs booléens :

a	*b*	$a \oplus b$	$a+b$	$a.b$
1	1	0	1	1
1	0	1	1	0
0	1	1	1	0
0	0	0	0	0

Rappelons ensuite la table d'addition en numération binaire :

+	**0**	**1**
0	0	1
1	1	0(1)

*0(1) représente la retenue **1** à reporter.*

En considérant une addition binaire comme la somme à effectuer sur deux mémoires à un bit, nous observons dans l'addition binaire les différentes configurations des bits concernés (notés ***a*** et ***b***).

Nous aurons comme résultat un bit de ***somme*** et un bit de ***retenue*** :

bit ***a***		bit ***b***		bit ***somme***	bit de ***retenue***
1	+	1	=	0	1
1	+	0	=	1	0
0	+	1	=	1	0
0	+	0	=	0	0

Si l'on compare avec les tables d'opérateurs booléens, on s'aperçoit que l'opérateur "⊕" (*Xor*) fournit en sortie les mêmes configurations que le bit de somme, et que l'opérateur "•" (*Et*) délivre en sortie les mêmes configurations que le bit de retenue.

Il est donc possible de simuler une addition binaire (arithmétique binaire) avec les deux opérateurs "⊕" et "•". Nous venons de construire un demi-additionneur ou additionneur sur un bit. Nous pouvons donc réaliser le circuit logique simulant la fonction complète d'addition binaire, nous l'appellerons " additionneur binaire "(somme arithmétique en binaire de deux entiers en binaire).

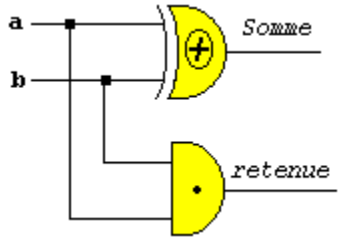
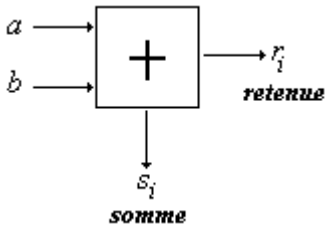


schéma logique d'un demi-additionneur

Ce circuit est noté :

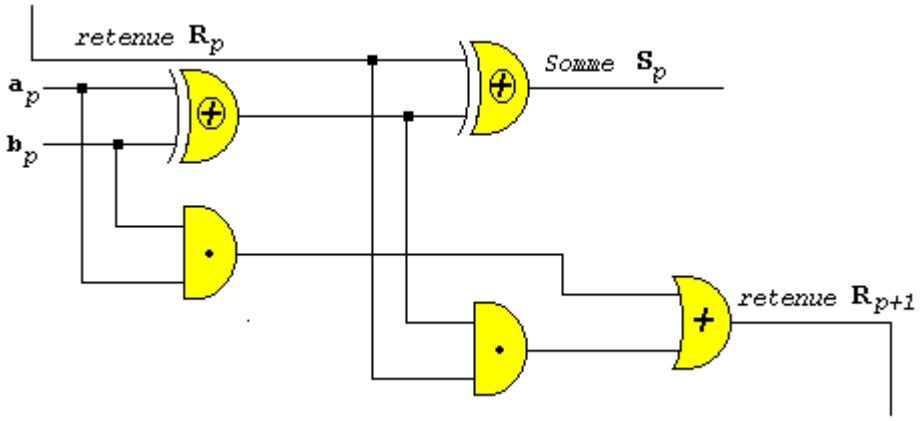


b)Additionneur complet

Une des constructions les plus simples et la plus pédagogique d'un additionneur complet est de connecter entre eux et en série des demi-additionneurs (additionneurs en cascade). Il existe une autre méthode dénommée " diviser pour régner " pour construire des additionneurs complets plus rapides à l'exécution que les additionneurs en cascade. Toutefois un additionneur en cascade pour UAL à 32 bits, utilise 2 fois moins de composants électroniques qu'un additionneur diviser pour régner.

Nous concluons donc qu'une UAL n'effectue en fait que des opérations logiques (algèbre de Boole) et simule les calculs binaires par des combinaisons d'opérateurs logiques

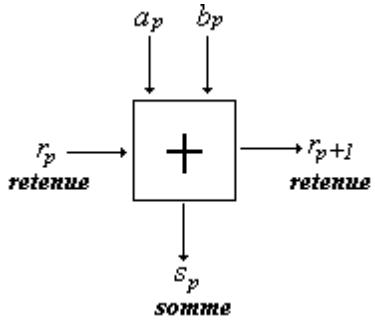
Soient a et b deux nombres binaires à additionner dans l'UAL. Nous supposons qu'ils sont stockés chacun dans une mémoire à n bits. Nous notons $\mathbf{a_p}$et $\mathbf{b_p}$ leur bit respectif de rang p. Lors de l'addition il faut non seulement additionner les bits $\mathbf{a_p}$ et $\mathbf{b_p}$ à l'aide d'un demi-aditionneur, mais aussi l'éventuelle retenue notée **Rp** provenant du calcul du rang précédent.



additionneur en cascade (addition sur le bit de rang p)

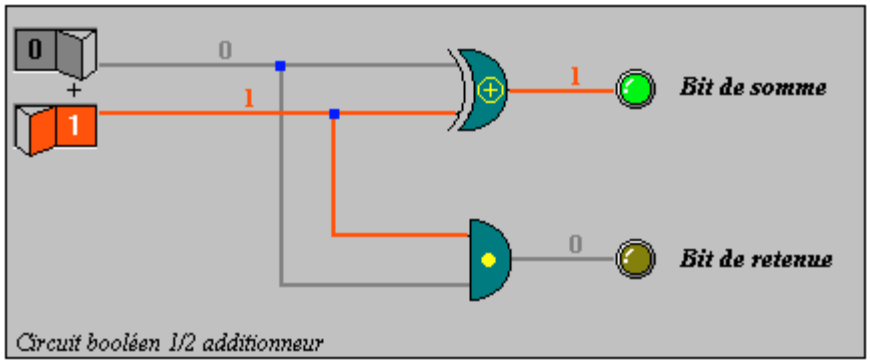
On réadditionne **Rp** à l'aide d'un demi-additionneur à la somme de $\mathbf{a_p}$et $\mathbf{b_p}$et l'on obtient le bit de somme du rang p noté **Sp**. La propagation de la retenue $\mathbf{Rp_{+1}}$ est faite par un " ou " sur les deux retenues de chacun des demi-additionneurs et passe au rang p+1. Le processus est itératif sur tous les n bits des mémoires contenant les nombres a et b.

Notation du circuit additionneur :



Soit un exemple fictif de réalisation d'un demi-additionneur simulant l'addition binaire suivante :

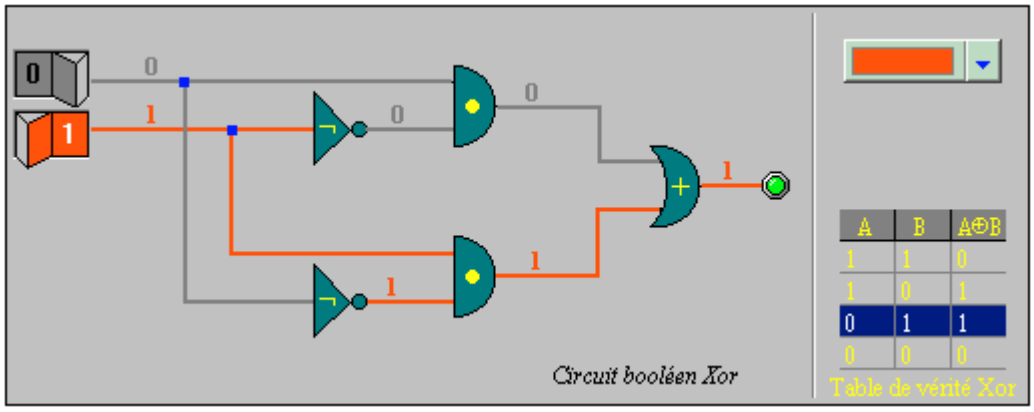
0 + 1 = 1. Nous avons figuré le passage ou non du courant à l'aide de deux interrupteurs (valeur = 1 indique que l'interrupteur est fermé et que le courant passe, valeur = 0 indique que l'interrupteur est ouvert et que le courant ne passe pas)



Le circuit « **et** » fournit le bit de retenue soit : 0 •1 = 0

Le circuit « **Xor »** fournit le bit de somme soit : 0 ⊕ 1 = 1

Nous figurons le détail du circuit Xor du schéma précédent lorsqu'il reçoit le courant des deux interrupteurs précédents dans la même position (l'état électrique correspond à l'opération 0 ⊕ 1 = 1)

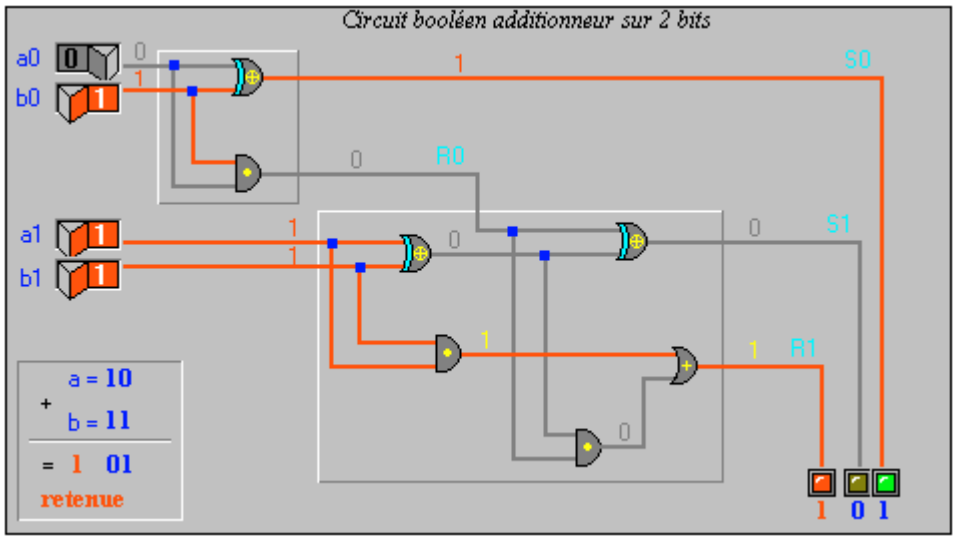


Si l'UAL effectue des additions sur 32 bits, il y aura 32 circuits comme le précédent, tous reliés en série pour la propagation de la retenue.

Un exemple d'additionneur sur deux mémoires **a** et **b** à 2 bits contenant respectivement les nombres 2 et 3 :

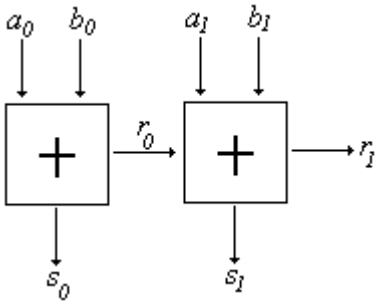
a = | 1 | 0 |

b = | 1 | 1 |



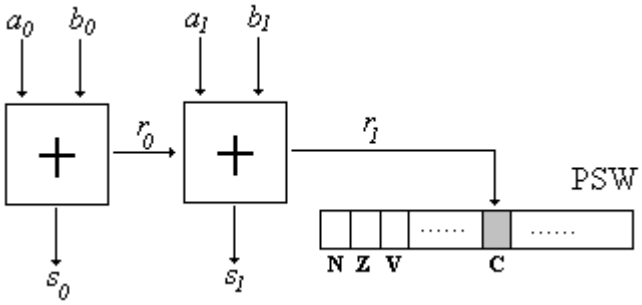
Les 4 interrupteurs figurent le passage du courant sur les bits de même rang des mémoires **a=2** et **b=3**, le résultat obtenu est la valeur attendue soit 2+3 = 5.

Notation du circuit additionneur sur 2 bits :



Remarque :

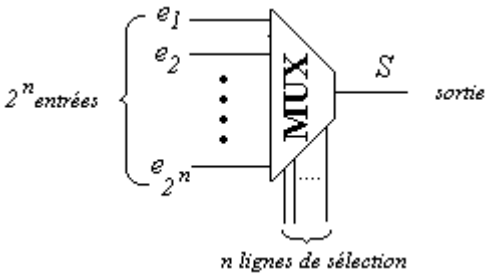
Ce circuit d'addition sur 2 bits engendre en fait en plus des bits de somme un troisième bit de retenue qui sera généralement mémorisé dans le bit de retenue (bit de carry noté C) du mot d'état programme ou PSW (Progral Status Word) du processeur. C'est le bit C de ce mot qui est consulté par exemple afin de savoir si l'opération d'addition a généré un bit de retenu ou non.



3.5 Circuit multiplexeur (circuit combinatoire)

C'est un circuit d'aiguillage comportant 2^n entrées, n lignes de sélection et une seule sortie. Les n lignes de sélection permettent de "programmer" le numéro de l'entrée qui doit être sélectionnée pour sortir sur une seule sortie (un bit). La construction d'un tel circuit nécessite 2^n circuits "et", n circuits "non" et 1 circuit "ou".

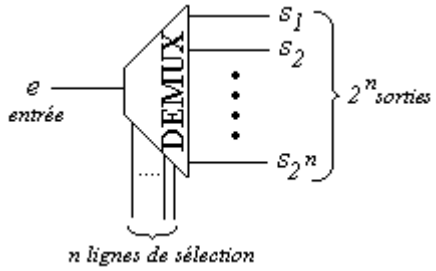
Notation du multiplexeur :



3.6 Circuit démultiplexeur (circuit combinatoire)

C'est un circuit qui fonctionne à l'inverse du circuit précédent, il permet d'aiguiller une seule entrée (un bit) sur l'une des 2^n sorties possibles, selon la "programmation"(l'état) de ses n lignes de sélection.

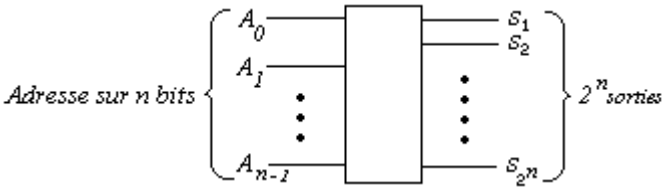
Notation du démultiplexeur :



3.7 Circuit décodeur d'adresse (circuit combinatoire)

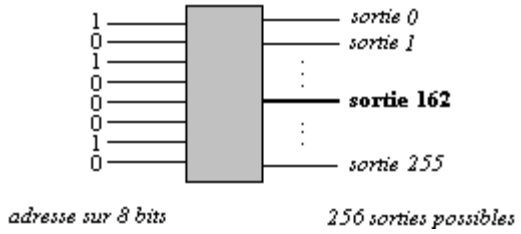
C'est un circuit composé de n lignes d'entrées qui représentent une adresse sur n bits et de 2^n lignes de sortie possibles dont une seule est sélectionnée en fonction de la "programmation" des n lignes d'entrées.

Notation du décodeur d'adresse :



Exemple d'utilisation d'un décodeur d'adresse à 8 bits :

On entre l'adresse de la ligne à sélectionner soit 10100010 (A_0 =1 , A_1 = 0, A_2 = 1, … , A_7 = 0) ce nombre binaire vaut 162 en décimal, c'est donc la sortie S_{162} qui est activée par le composant comme le montre la figure ci-dessous.

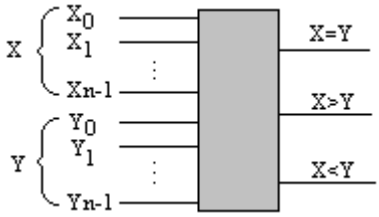


La construction d'un circuit décodeur d'adresse à n bits nécessite 2^n circuits "et", n circuits "non". Ce genre de circuits très fréquent dans un ordinateur sert à sélectionner des registres, des cellules mémoires ou des lignes de périphériques.

3.8 Circuit comparateur (circuit combinatoire)

C'est un circuit réalisant la comparaison de deux mots X et Y de n bits chacun et sortant une des trois indication possible X+Y ou bien X>Y ou X<Y. Il possède donc 2n entrées et 3 sorties.

Notation du comparateur de mots à n bits :



3.9 Circuit bascule (circuit à mémoire)

C'est un circuit permettant de mémoriser l'état de la valeur d'un bit. Les bascules sont les principaux circuits constituant les registres et les mémoires dans un ordinateur.

Les principaux types de bascules sont RS, JK et D, ce sont des dispositifs chargés de "conserver" la valeur qu'ils viennent de prendre.

Schéma électronique et notation de bascule RS minimale théorique

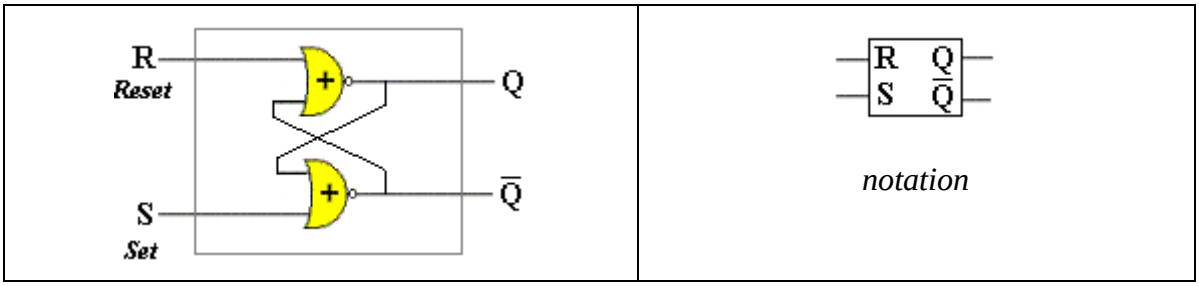


Table de vérité associée à cette bascule :

R	S	Qt+dt
1	1	------
1	0	0
0	1	1
0	0	Qt

Qt représente la valeur de la sortie au temps **t** , **Qt+dt** représente la valeur de cette même sortie un peu plus tard au temps **t+dt**.

L'état R=1 et S=1 n'est pas autorisé

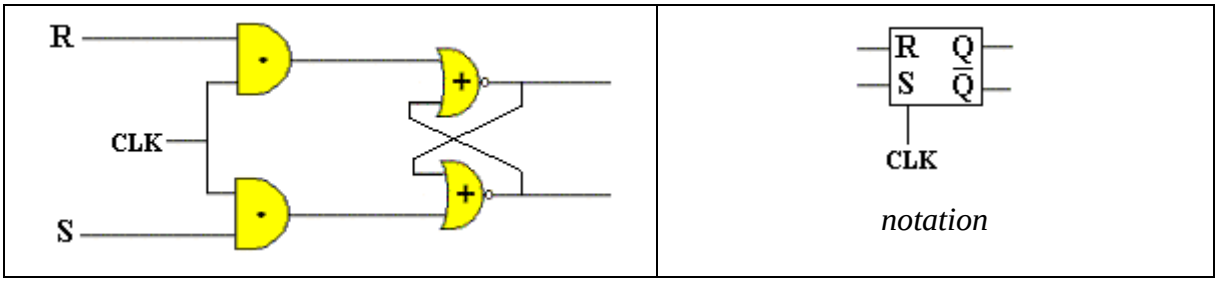
L'état R=0 et S=0 fait que **Qt+dt = Qt** , la sortie **Q** conserve la même valeur au cours du temps, le circuit "mémorise" donc un bit.

Si l'on veut que le circuit mémorise un bit égal à 0 sur sa sortie Q, on applique aux entrées les valeurs R=1 et S=0 au temps t_0, puis à t_0+dt on applique les valeurs R=0 et S=0. Tant que les entrées R et S restent à la valeur 0, la sortie Q conserve la même valeur (dans l'exemple Q=0).

En pratique ce sont des bascules RS synchronisées par des horloges (CLK pour clock) qui sont utilisées, l'horloge sert alors à commander l'état de la bascule. Seule la sortie Q est considérée.

Dans une bascule RS synchronisée, selon que le top d'horloge change d'état ou non et selon les valeurs des entrées R et S soit d'un top à l'autre la sortie Q du circuit ne change pas soit la valeur du top d'horloge fait changer (basculer) l'état de la sortie Q.

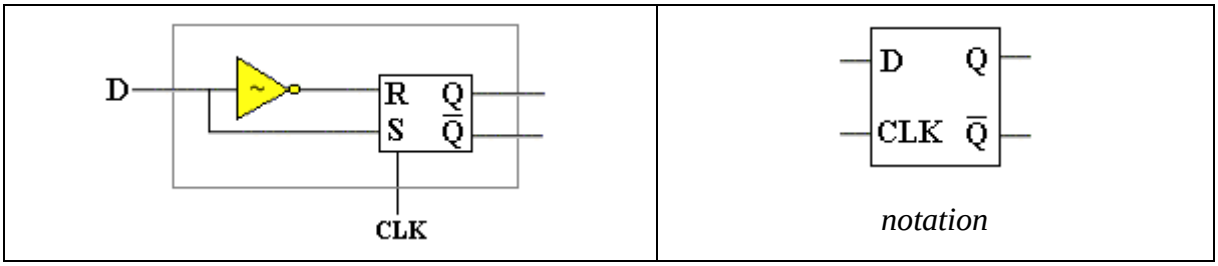
Schéma électronique général et notation d'une bascule RS synchronisée



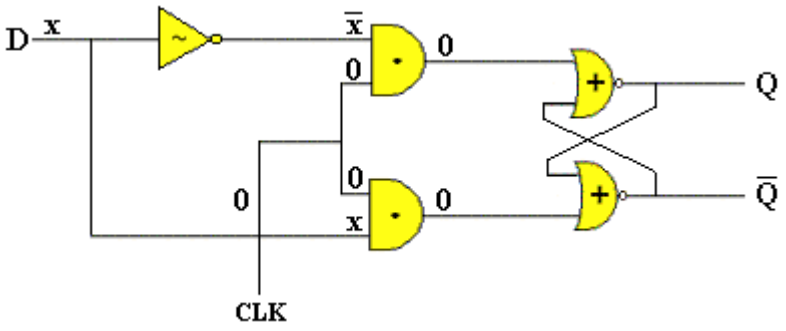
Remarque

Certains types de mémoires ou les registres dans un ordinateur sont conçus avec des variantes de bascules RS (synchronisées) notée JK ou D.

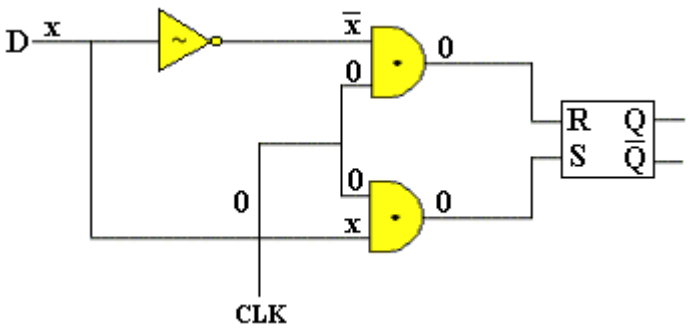
Schéma électronique général et notation d'une bascule de type D



Fonctionnement pratique d'une telle bascule D dont les entrées sont reliées entre elles. Supposons que la valeur de l'entrée soit le booléen x (x=0 ou bien x=1) et que l'horloge soit à 0.



En simplifiant le schéma nous obtenons une autre présentation faisant apparaître la bascule RS minimale théorique décrite ci-haut :



Or la table de vérité de cet élément lorsque les entrées sont égales à 0 indique que la bascule conserve l'état antérieur de la sortie Q:

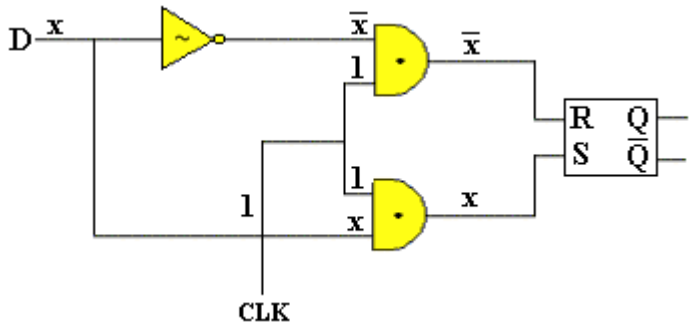
R	S	Qt+dt
0	0	Qt

Conclusion pour une bascule D

> Lorsque l'horloge est à 0, quelque soit la valeur de l'entrée D (D=0 ou D=1) une bascule D conserve la même valeur sur la sortie Q.

Que se passe-t-il lorsque lors d'un top d'horloge celle-ci passe à la valeur 1 ?

Reprenons le schéma simplifié précédent d'une bascule D avec une valeur d'horloge égale à 1.



Nous remarquons que sur les entrée R et S nous trouvons la valeur **x** et son complément $\bar{x}$, ce qui élimine deux couples de valeurs d'entrées sur R et S (R=0 , S=0) et (R=1 , S=1). Nous sommes sûrs que le cas d'entrées non autorisé par un circuit RS (R=1 , S=1) n'a jamais lieu dans une bascule de type D. Il reste à envisager les deux derniers couples (R=0 , S=1) et (R=1 , S=0). Nous figurons ci-après la table de vérité de la sortie Q en fonction de l'entrée D de la bascule (l'horloge étant positionnée à 1) et pour éclairer le lecteur nous avons ajouté les deux états associés des entrées internes R et S :

x	R	S	Q	**Nous remarquons que la sortie Q prend la valeur de l'entrée D (D=x), elle change donc d'état.**
0	1	0	0	
1	0	1	1	

Conclusion pour une bascule D

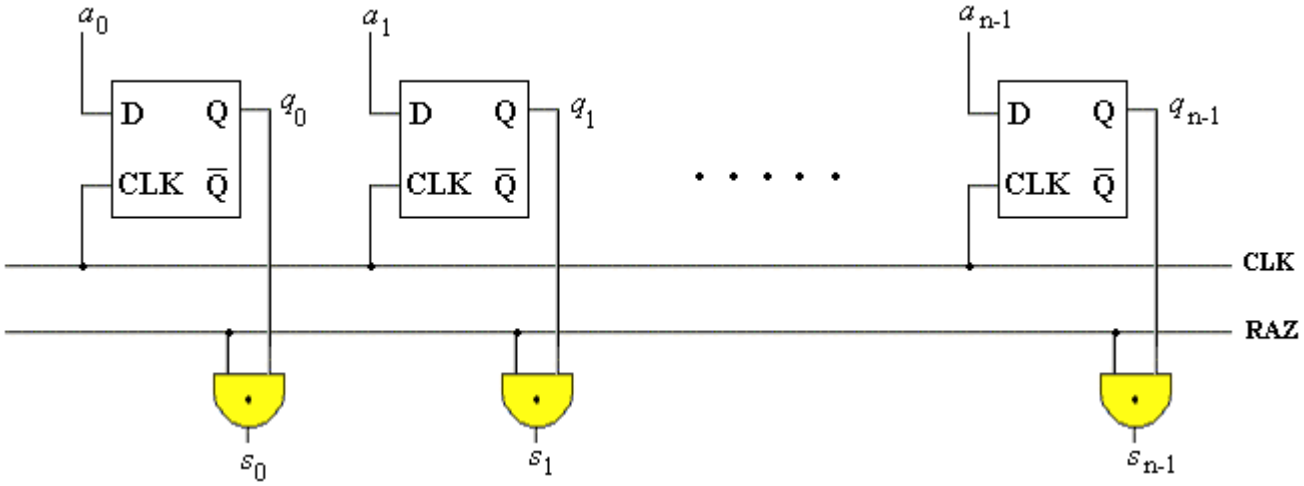
> Lorsque l'horloge est à 1, quelque soit la valeur de l'entrée D (D=0 ou D=1) une bascule D change et prend sur la sortie Q la valeur de l'entrée D.

3.10 Registre (circuit à mémoire)

Un registre est un circuit qui permet la mémorisation de n bits en même temps. Il existe dans un ordinateur plusieurs variétés de registres, les registres parallèles, les registres à décalage (décalage à droite ou décalage à gauche) les registres séries.

Les bascules de type D sont les plus utilisées pour construire des registres de différents types en fonction de la disposition des entrées et des sorties des bascules : les registres à entrée série/sortie série, à entrée série/sortie parallèle, à entrée parallèle/sortie parallèle, à entrée parallèle/sortie série.

Voici un exemple de registre à n entrées parallèles (a_0,a_1,…,a_{n-1}) et à n sorties parallèles (s_0,s_1,…,s_{n-1}) construit avec des bascules de type D :



Examinons le fonctionnement de ce "registre parallèle à n bits"

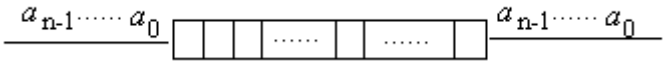
La ligne CLK fournit le signal d'horloge, la ligne RAZ permet l'effacement de toutes les sorties s_k du registre, on dit qu'elle fournit le **signal de validation** :

> Lorsque RAZ = 0 on a (s_0=0, s_1=0, …, s_{n-1}=0)
> Lorsque RAZ = 1 on a (s_0= q_0, s_1= q_1, …, s_{n-1}= q_{n-1})

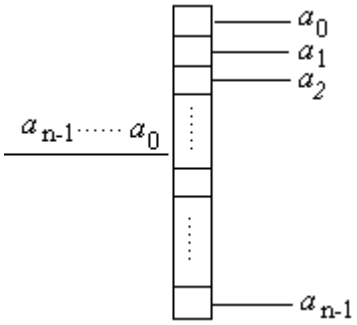
Donc RAZ=0 sert à effacer tous les bits de sortie du registre, dans le cas où RAZ=1 qu'en est-il des sorties s_k. D'une manière générale nous avons par construction s_k = RAZ . q_k :

- Tant que CLK = 0 alors, comme RAZ=1 nous avons s_k = q_k (q_k est l'état antérieur de la bascule). Dans ces conditions on dit que l'on "lit le contenu actuel du registre".
- Lorsque CLK = 1 alors, tous les q_k basculent et chacun d'eux prend la nouvelle valeur de son entrée a_k. Comme RAZ=1 nous avons toujours s_k = q_k (q_k est le nouvel état de la bascule). Dans ces conditions on dit que l'on vient de "charger le registre" avec une nouvelle valeur.

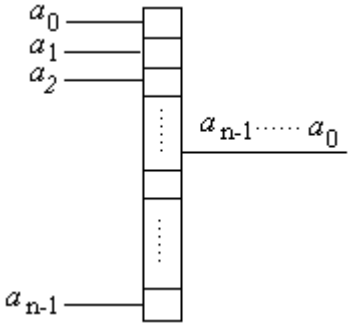
Notations des différents type de registres :



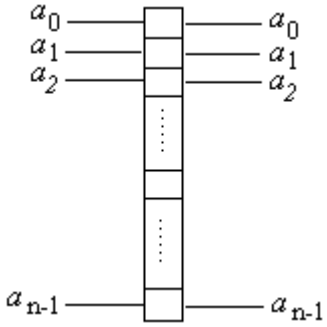
registre série/série



registre série/parallèle



registre parallèle/série



registre parallèle/parallèle

Registre à décalage

C'est un registre à entrée série ou parallèle qui décale de 1 bit tous les bits d'entrée soit vers "la droite" (vers les bits de poids faibles), soit vers "la gauche" (vers les bits de poids forts). Un registre à décalage dans un ordinateur correspond soit à une multiplication par 2 dans le cas du décalage à gauche, soit à une division par 2 dans le cas du décalage à droite.

Conclusion mémoire-registre

Nous remarquons donc que les registres en général sont des mémoires construites avec des bascules dans lesquelles on peut lire et écrire des informations sous forme de bits. Toutefois dès que la quantité d'information à stocker est très grande les bascules prennent physiquement trop de place (2 NOR, 2 AND et 1 NON). Actuellement, pour élaborer une mémoire stockant de très grande quantité d'informations, on utilise une technologie plus compacte que celle des bascules, elle est fondée sur la représentation d'un bit par 1 transistor et 1 condensateur. Le transistor réalise la porte d'entrée du bit et la sortie du bit, le condensateur selon sa charge réalise le stockage du bit.

Malheureusement un condensateur ne conserve pas sa charge au cours du temps (courant de fuite inhérent au condensateur), il est alors indispensable de restaurer de temps en temps la charge du condensateur (opération que l'on dénomme **rafraîchir la mémoire**) et cette opération de rafraîchissement mémoire a un coût en temps de réalisation. Ce qui veut donc dire que pour le même nombre de bits à stocker un registre à bascule est plus rapide à lire ou à écrire qu'une mémoire à transistor, c'est pourquoi les mémoires internes des processeurs centraux sont des registres.

3.11 Mémoire SRAM et mémoire DRAM

Dans un ordinateur actuel coexistent deux catégories de mémoires :

1°) Les mémoires statiques SRAM élaborées à l'aide de bascules : très rapides mais volumineuses (plusieurs transistors pour 1 bit).

2°) Les mémoires dynamiques DRAM élaborées avec un seul transistor couplé à un condensateur : très facilement intégrables dans une petite surface, mais plus lente que les SRAM à cause de la nécessité du rafraîchissement.

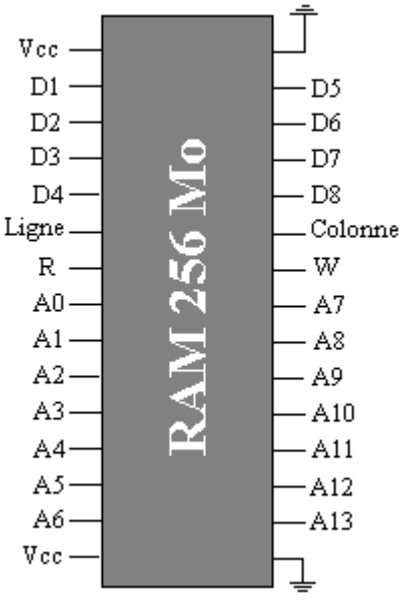
Voici à titre indicatif des ordres de grandeur qui peuvent varier avec les innovations technologiques rapides en ce domaine :

> SRAM temps d'accès à une information : 5 nanosecondes
>
> DRAM temps d'accès à une information : 50 nanosecondes

Fonctionnement d'une DRAM de 256 Mo fictive

La mémoire physique aspect extérieur :

Le schéma général de la mémoire :



Vcc = alimentation électrique

D1 à D8 = bits de données (1 octet ici)

Ligne, Colonne = lignes de sélection soit d'une adresse de ligne soit d'une adresse de colonne

W = autorisation d'écriture

R = validation de lecture

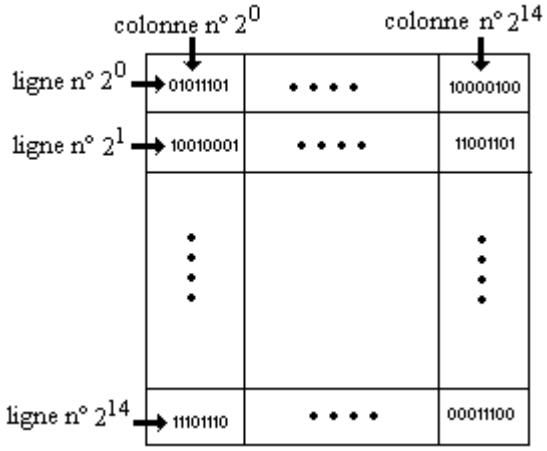
A0, … , A13 = adresse d'une ligne ou adresse d'une colonne

⏚ = symbole de mise à la masse

Nous adoptons une vision abstraite de l'organisation interne de cette mémoire sous forme d'une matrice de 2^{14} lignes et 2^{14} colonnes soient en tout 2^{14}. 2^{14}= 2^{28} cellules de 1 octet chacune (2^{28} octets = 2^{8}. 2^{20} o = 256 . 2^{20} o = 256 Mo, car 1 Mo = 2^{20} o)

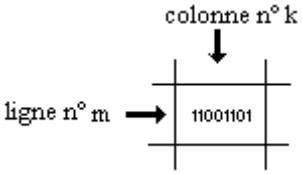
Ce qui donne une matrice de 16384 lignes et 16384 colonnes, numérotées par exemple de 2^{0} = 1

jusqu'à $2^{14} = 16384$, selon la figure ci-dessous :

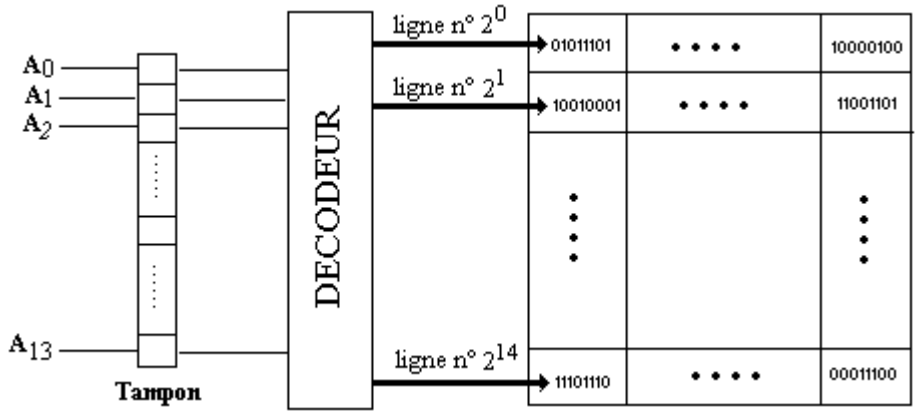


Dans l'exemple à gauche :

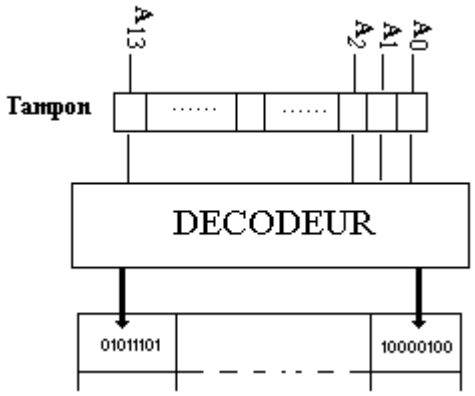
La sélection d'une ligne de numéro **m** donné (d'adresse **m-1** donnée) et d'une colonne de numéro **k** donné (d'adresse **k-1** donnée) permet de sélectionner directement une cellule contenant 8 bits.



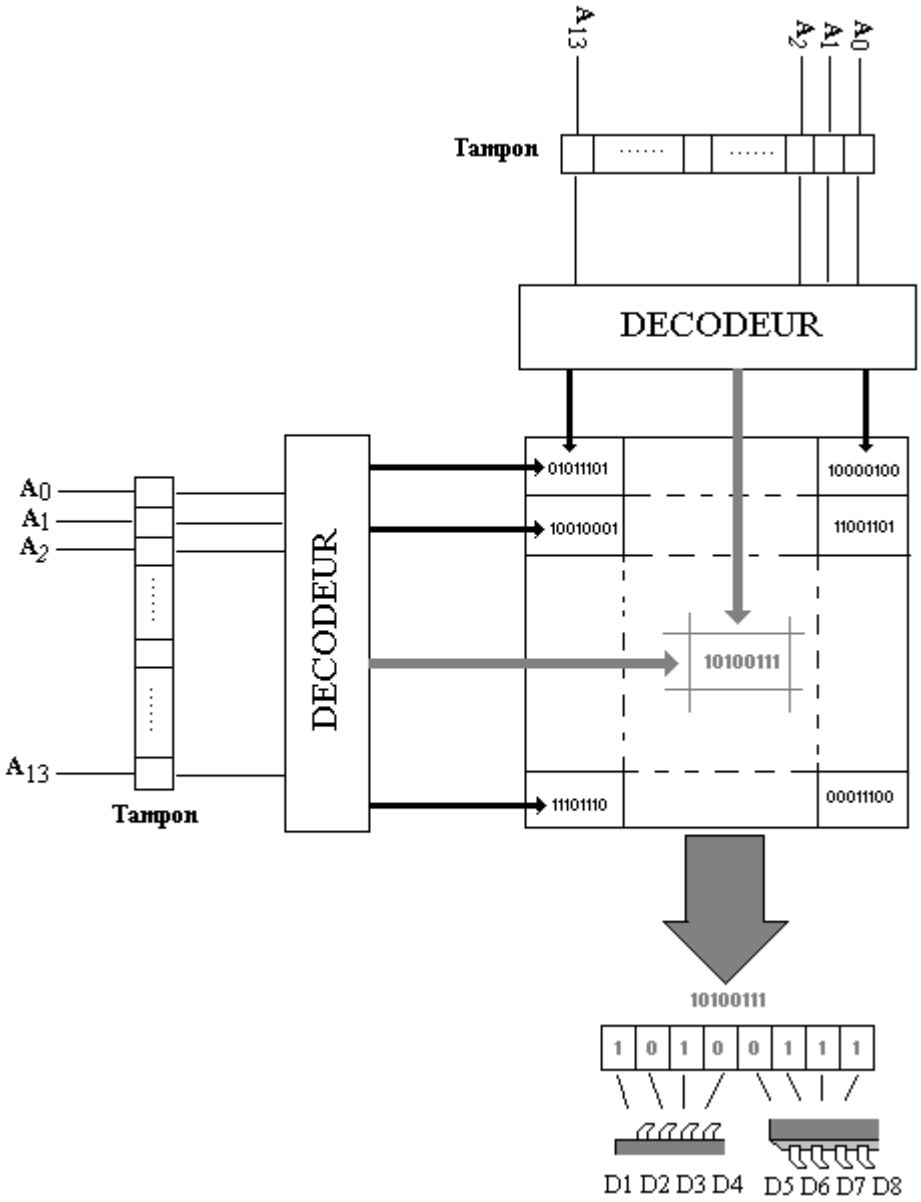
Exemple de sélection de ligne dans la matrice mémoire à partir d'une adresse (A0, … , A13) , dans notre exemple théorique la ligne de numéro $2^0 = 1$ a pour adresse (0,0,…,0) et la ligne de numéro $2^{14} = 16384$ a pour adresse (1,1,…,1). Lorsque l'adresse de sélection d'une ligne arrive sur les pattes (A0, … , A13) de la mémoire elle est rangée dans un registre interne (noté tampon) puis passée à un circuit interne du type décodeur d'adresse à 14 bits (14 entrées et $2^{14} = 16384$ sorties) qui sélectionne la ligne adéquate.



Il en va de même pour la sélection d'une colonne :



La sélection d'une ligne, puis d'une colonne permet d'obtenir sur les pattes D, D2, …, D8 de la puce les 8 bits sélectionnés. Ci dessous une sélection en mode lecture d'une cellule de notre mémoire de 256 Mo :

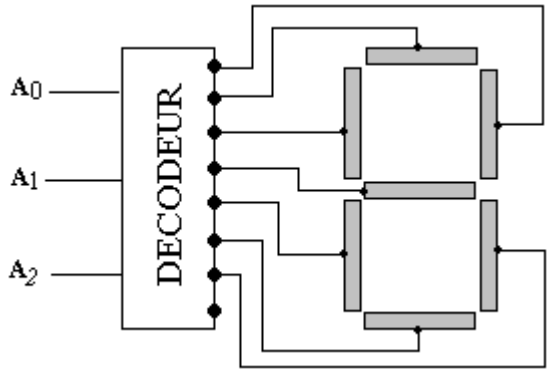


Il est possible aussi d'écrire dans une cellule de la mémoire selon la même démarche de sélection. Pour opérer une lecture il faut que la ligne de validation **R** de la mémoire soit activée, pour opérer une écriture, il faut que la ligne de validation **W** de la mémoire soit activée.

En attendant une nouvelle technologie (optique, quantique, organique,…) les constituants de base d'un ordinateur sont fondés sur l'électronique à base de transistor découverts à la fin des années quarante. De nos jours deux technologie de transistor sont présentes sur le marché : la technologie TTL (Transistor Transistor Logic) la plus ancienne et la technologie MOS (Metal Oxyde Semi-conductor).

3.12 Afficheur LED à 7 segments

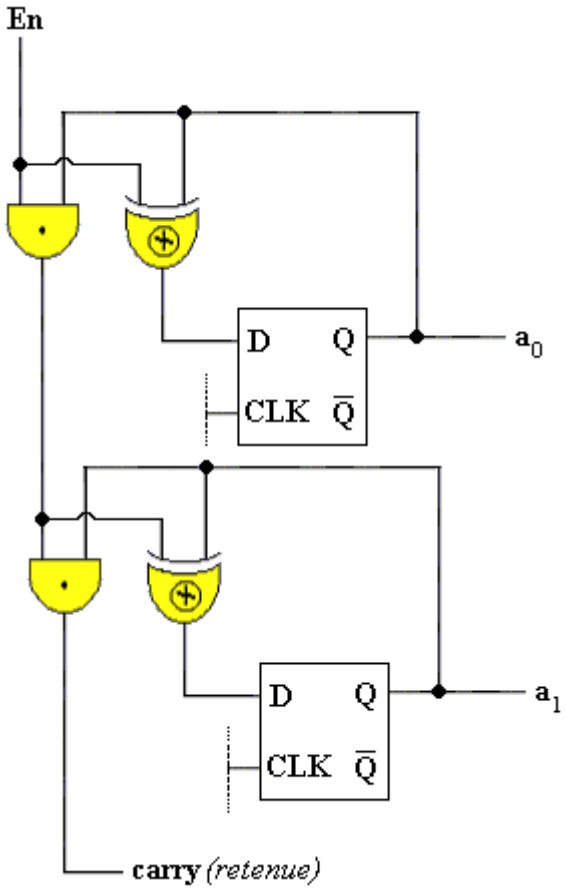
On utilise dans les ordinateurs des afficheurs à LED, composés de 7 led différentes qui sont allumées indépendamment les unes des autres, un circuit décodeur à 3 bits permet de réaliser simplement cet affichage :



3.13 Compteurs

Ce sont des circuits chargés d'effectuer un comptage cumulatif de divers signaux.

Par exemple considérons un compteur sur 2 bits avec retenue éventuelle, capable d'être activé ou désactivé, permettant de compter les changement d'état de la ligne d'horloge CLK. Nous proposons d'utiliser deux demi-additionneurs et deux bascules de type D pour construire le circuit.

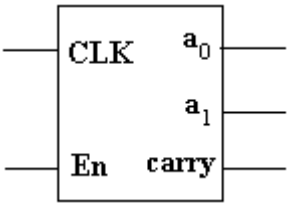


Le circuit compteur de gauche possède deux entrées **En** et **CLK**, il possède trois sorties a_0, a_1 et carry.

Ce compteur sort sur les bits a_0, a_1 et sur le bit de carry le nombre de changements en binaire de la ligne CLK (maximum 4 pour 2 bits) avec retenue s'il y a lieu.

La ligne d'entrée **En** est chargée d'activer ou de désactiver le compteur

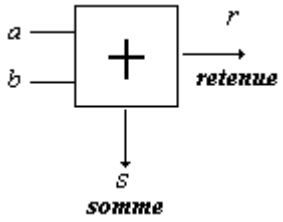
Notation pour ce compteur :



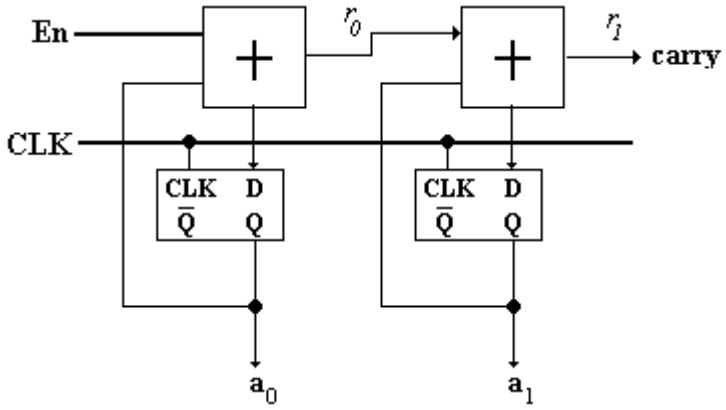
Fonctionnement de l'entrée **En** (enable) du compteur précédent :

- Lorsque En = 0, sur la première bascule en entrée D nous avons $D = a_0 \oplus 0$ (or nous savons que : $\forall x, x \oplus 0 = x$), donc $D = a_0$ et Q ne change pas de valeur. Il en est de même pour la deuxième bascule et son entrée D vaut a_1. Donc quoiqu'il se passe sur la ligne CLK les sorties a_0 et a_1 ne changent pas, on peut donc dire que le comptage est **désactivé lorsque le enable est à zéro**.
- Lorsque En = 1, sur la première bascule en entrée D nous avons $D = a_0 \oplus 1$ (or nous savons que : $\forall x, x \oplus 1 = \overline{x}$), donc Q change de valeur. On peut donc dire que le comptage est **activé lorsque le enable est à un**.

Utilisons la notation du demi-additionneur pour représenter ce compteur à 2 bits :

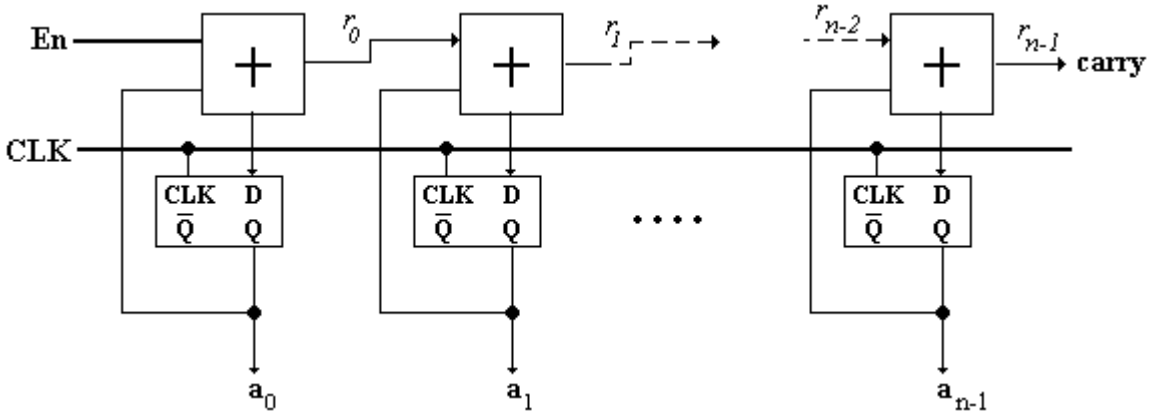


un demi-additionneur



le compteur à 2 bits

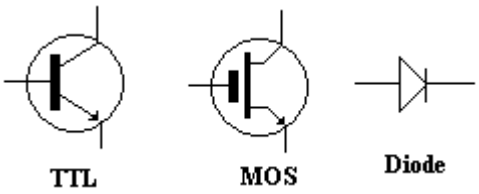
En généralisant à la notion de compteur à n bits nous obtenons le schéma ci-après :



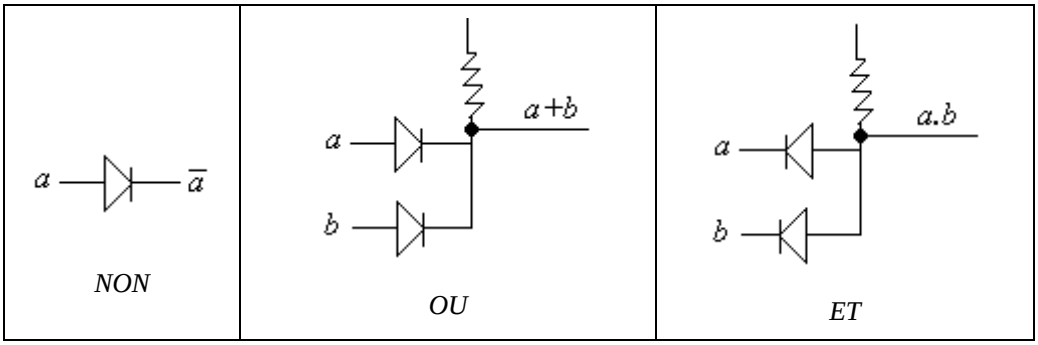
3.14 Réalisation électronique de circuits booléens

Dans ce paragraphe nous indiquons pour information anecdotique au lecteur, à partir de quelques exemples de schémas électroniques de base, les réalisation physiques possibles de différents opérateurs de l'algèbre de Boole.

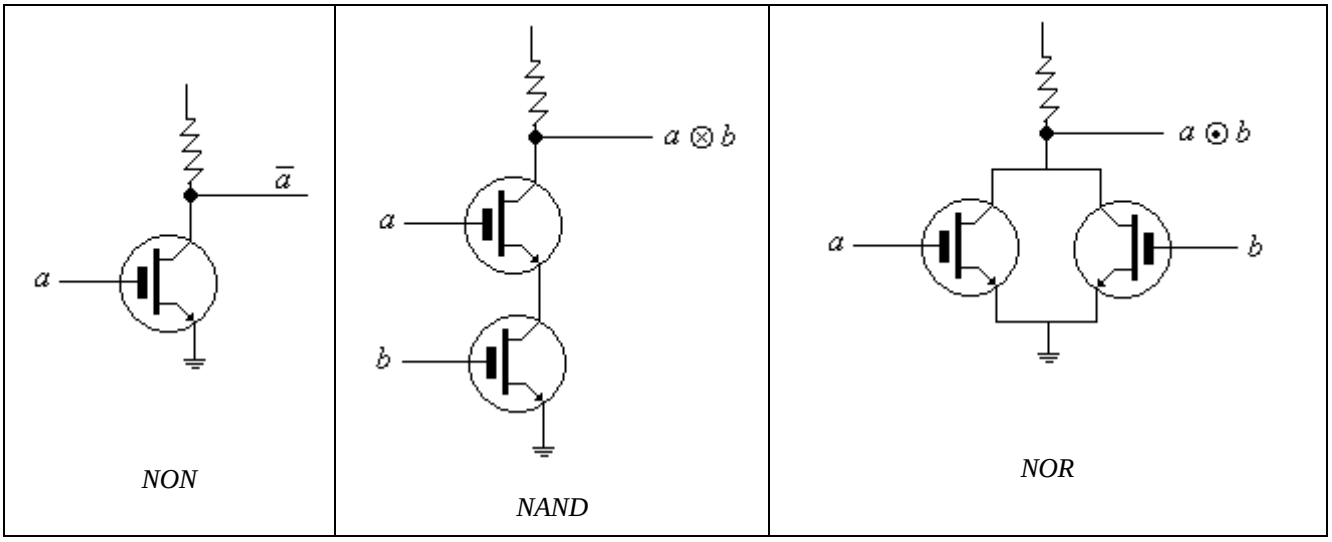
Le transistor est principalement utilisé comme un interrupteur électronique, nous utiliserons les schémas suivants représentant un transistor soit en TTL ou MOS et une diode.



Circuits (ET, OU , NON) élaborés à partir de diodes :

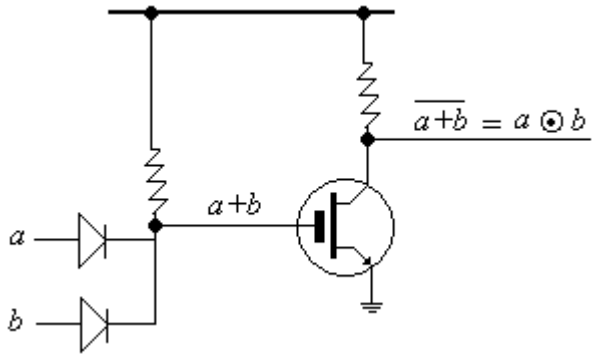


Circuits (NOR, NAND , NON) élaborés à partir de transistor MOS :



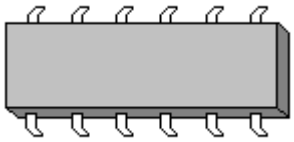
Ce sont en fait la place occupée par les composants électroniques et leur coût de production qui sont les facteurs essentiels de choix pour la construction des opérateurs logiques de base.

Voici par exemple une autre façon de construire une circuit NOR à partir de transistor et de diodes :

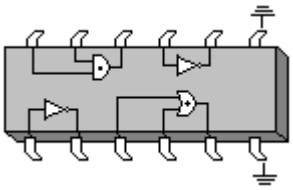


Le lecteur intéressé consultera des ouvrages d'électronique spécialisés afin d'approfondir ce domaine qui dépasse le champ de l'**informatique** qui n'est qu'une **simple utilisatrice** de la technologie électronique en attendant mieux !

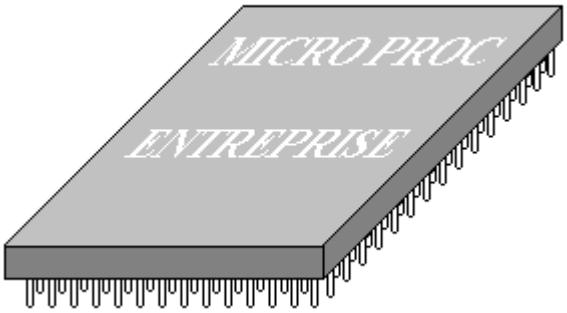
Finissons ce paragraphe, afin de bien fixer nos idées, par un schéma montrant comment dans une puce électronique sont situés les circuits booléens :



Supposons que la puce précédente permette de réaliser plusieurs fonctions et contienne par exemple 4 circuits booléens : un OU, un ET, deux NON. Voici figuré une possible implantation physique de ces 4 circuits dans la puce, ainsi que la liaison de chaque circuit booléen avec les pattes du composant physique :



Pour information, le micro-processeur pentium IV Northwood de la société Intel contient environ 55 000 000 (55 millions) de tansistors, le micro-processeur 64 bits Opteron de la société concurrente AMD plus récent que le pentium IV, contient 105 000 000 (105 millions) de transistor.



1.3 Codage numération

Plan du chapitre:

1. Codage de l'information

2. Numération

1. Codage de l'information

1.1 Codage en général : le pourquoi

- Dans une machine, toutes les informations sont codées sous forme d'une suite de "0" et de "1" (langage binaire). Mais l'être humain ne parle généralement pas couramment le langage binaire.
- Il doit donc tout "traduire" pour que la machine puisse exécuter les instructions relatives aux informations qu'on peut lui donner.
- Le codage étant une opération purement humaine, il faut produire des algorithmes qui permettront à la machine de traduire les informations que nous voulons lui voir traiter.

Le **codage** est une opération établissant une bijection entre une **information** et une **suite de " 0 " et de " 1 "** qui sont représentables en machine.

1.2 Codage des caractères : code ASCII

Parmi les codages les plus connus et utilisés, le codage **ASCII** (American Standard Code for Information Interchange)étendu est le plus courant (version **ANSI** sur Windows).

Voyons quelles sont les nécessités minimales pour l'écriture de documents alphanumériques simples dans la civilisation occidentale. Nous avons besoin de :

Un alphabet de lettres minuscules ={a, b, c,...,z}
soient 26 caractères

Un alphabet de lettres majuscules ={A,B,C,...,Z}
soient 26 caractères

Des chiffres {0,1,...,9}
soient 10 caractères

Des symboles syntaxiques {? , ; (" ...
au minimum *10 caractères*

Soit un total minimal de *72 caractères*

Si l'on avait choisi un code à 6 bits le nombre de caractères codables aurait été de $\mathbf{2^6 = 64}$ (tous les nombres binaires compris entre **000000** et **111111**), nombre donc insuffisant pour nos besoins.

Il faut au minimum 1 bit de plus, ce qui permet de définir ainsi $\mathbf{2^7 = 128}$ nombres binaires différents, autorisant alors le codage de 128 caractères.

Initialement le code **ASCII** est un code à 7 bits (**2^7 = 128 caractères**). Il a été étendu à un code sur 8 bits (**2^8 = 256 caractères**) permettant le codage des caractères nationaux (en France les caractères accentués comme : ù,à,è,é,â,...etc) et les caractères semi-graphiques.

Les pages HTML qui sont diffusées sur le réseau Internet sont en code ASCII 8 bits.

Un codage récent dit " universel " est en cours de diffusion : il s'agit du codage **Unicode** sur 16 bits (**2^{16} = 65536 caractères**).

De nombreux autres codages existent adaptés à diverses solution de stockage de l'information (DCB, EBCDIC,...).

1.3 Codage des nombres entiers : numération

Les nombres entiers peuvent être codés comme des caractères ordinaires. Toutefois les codages adoptés pour les données autres que numériques sont trop lourds à manipuler dans un ordinateur. Du fait de sa constitution, un ordinateur est plus " habile " à manipuler des nombres écrits en numération binaire (qui est un codage particulier).

Nous allons décrire trois modes de codage des entiers les plus connus.

Nous avons l'habitude d'écrire nos nombres et de calculer dans le système décimal. Il s'agit en fait d'un cas particulier de numération en base 10.

Il est possible de représenter tous les nombres dans un système à base b (b entier, b1). Nous ne présenterons pas ici un cours d'arithmétique, mais seulement les éléments nécessaires à l'écriture dans les deux systèmes les plus utilisés en informatique : le binaire (b=2) et l'hexadécimal (b=16).

Lorsque nous écrivons **5876** en base 10, la position des chiffres 5,8,7,6 indique la puissance de 10 à laquelle ils sont associés :

5 est associé à 10^3
8 est associé à 10^2
7 est associé à 10^1
6 est associé à 10^0

Il en est de même dans une base b quelconque (b=2, ou b=16). Nous conviendrons de mentionner la valeur de la base au dessus du nombre afin de savoir quel est son type de représentation.

Soit $\overline{x_n x_{n-1} \cdots x_0}^{\,b}$ un nombre **x** écrit en base b avec n+1 symboles.

- " $\mathbf{x_k}$ " est le symbole associé à la puissance " $\mathbf{b^k}$ "
- " $\mathbf{x_k}$ " $\in \{0,1, \ldots , b-1\}$

Lorsque b=2 (numération binaire)

Chaque symbole du nombre x, " $\mathbf{x_k}$ " ∈ {0,1}; autrement dit les nombres binaires sont donc écrits avec des 0 et des 1, qui sont représentés physiquement par des bits dans la machine.

Voici le schéma d'une mémoire à n+1 bits (au minimum 8 bits dans un micro-ordinateur) :

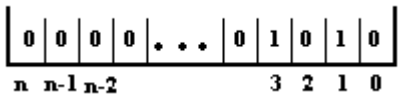
Les cases du schéma représentent les bits, le chiffre marqué en dessous d'une case, indique la puissance de 2 à laquelle est associé ce bit (on dit aussi le ***rang*** du bit).

Le bit de rang 0 est appelé le bit de **poids faible**.
Le bit de rang n est appelé le bit de **poids fort**.

1.4 Les entiers dans une mémoire à n+1 bits : binaire pur

Ce codage est celui dans lequel les nombres entiers **naturels** sont écrits en numération binaire (en base b=2).

Le nombre " dix " s'écrit **10** en base b=10, il s'écrit **1010** en base b=2. Dans la mémoire ce nombre dix est codé en binaire ainsi:



Une mémoire à n+1 bits (n>0), permet de représenter sous forme binaire (en binaire pur) tous les entiers naturels de l'intervalle **[0 , 2^{n+1} -1]**.

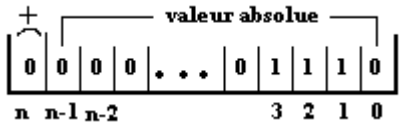
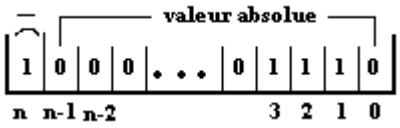
- soit pour n+1=8 bits, tous les entiers de l'intervalle **[0 , 255]**
- soit pour n+1=16 bits, tous les entiers de l'intervalle **[0 , 65535]**

1.5 Codage des nombres entiers : binaire signé

Ce codage permet la représentation des nombres entiers relatifs.

Dans la représentation en binaire signé, le bit de poids fort *(bit de rang n associé à 2^n)* sert à représenter le signe (0 pour un entier positif et 1 pour un entier négatif), les **n** autres bits représentent la valeur absolue du nombre en binaire pur.

Exemple du codage en binaire signé des nombres **+14** et **-14** :

	
+14 est représenté par 0000...01110	*-14 est représenté par 1000...01110*

Une mémoire à n+1 bits (n>0), permet de représenter sous forme binaire (en binaire signé) tous les entiers naturels de l'intervalle **[- (2^n - 1) , (2^n -1)]**

- soit pour n+1=8 bits, tous les entiers de l'intervalle **[-127 , 127]**
- soit pour n+1=16 bits, tous les entiers de l'intervalle **[-32767 , 32767]**

Le nombre zéro est représenté dans cette convention (dites du zéro positif) par : **0**000...00000

> **Remarque :** Il reste malgré tout une configuration mémoire inutilisée : **1**000...00000. Cet état binaire ne représente à priori aucun nombre entier ni positif ni négatif de l'intervalle [- (2^n - 1) , (2^n -1)]. Afin de ne pas perdre inutilement la configuration " **1**000...00000 ", les informaticiens ont décidé que cette configuration représente malgré tout un nombre négatif parce que le bit de signe est 1, et en même temps la puissance du bit contenant le "1", donc par convention -2^n.

L'intervalle de représentation se trouve alors augmenté d'un nombre :
il devient :[-2^n ,2^n -1]

- soit pour n+1=8 bits, tous les entiers de l'intervalle **[-128 , 127]**
- soit pour n+1=16 bits, tous les entiers de l'intervalle **[-32768 , 32767]**

Ce codage n'est pas utilisé tel quel, il est donné ici à titre pédagogique. En effet le traitement spécifique du signe coûte cher en circuits électroniques et en temps de calcul. C'est une version améliorée qui est utilisée dans la plupart des calculateurs : elle se nomme le complément à deux.

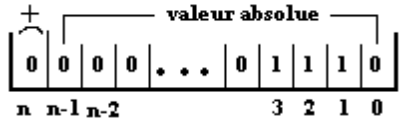
1.6 Un autre codage des nombres entiers : complément à deux

Ce codage, purement conventionnel et très utilisé de nos jours, est dérivé du binaire signé ; il sert à représenter en mémoire les entiers relatifs.

Comme dans le binaire signé, la mémoire est divisée en deux parties inégales; le bit de poids fort représentant le signe, le reste représente la valeur absolue avec le codage suivant :

Supposons que la mémoire soit à n+1 bits, soit **x** un entier relatif à représenter :

si **x > 0,** alors c'est la convention en *binaire signé* qui s'applique (le bit de signe vaut 0, les n bits restants codent le nombre), soit pour le nombre +14 :

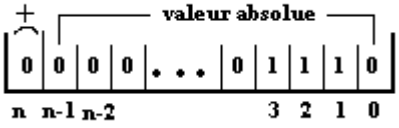


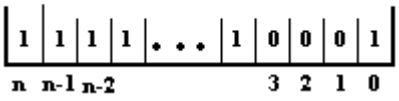
+14 est représenté par 0000...01110

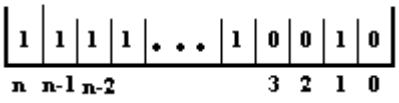
si **x < 0**, alors (3 étapes à suivre)

- On code la valeur absolue du nombre x, **|x|** en binaire signé.
- Puis l'on complémente tous les bits de la mémoire (complément à 1 ou complément restreint). Cette opération est un **non** logique effectué sur chaque bit de la mémoire.
- Enfin l'on additionne +**1** au nombre binaire de la mémoire (addition binaire).

Exemple, soit à représenter le nombre -14 en suivant les 3 étapes :

- 

codage de |-14|= 14
- 

complément à 1
- 

addition de 1

Le nombre -14 s'écrit donc en complément à 2 : 1111..10010.

Un des intérêts majeurs de ce codage est d'intégrer la soustraction dans l'opération de codage et de ne faire effectuer que des opérations simples et rapides (non logique, addition de 1).

Nous venons de voir que le codage utilisait essentiellement la représentation d'un nombre en binaire (la numération binaire) et qu'il fallait connaître les rudiments de l'arithmétique binaire. Le paragraphe ci-après traite de ce sujet.

2. Numération

Ce paragraphe peut être ignoré par ceux qui connaissent déjà les éléments de base des calculs en binaire et des conversions binaire-décimal-hexadécimal, dans le cas contraire, il est conseillé de le lire.

Pour des commodités d'écriture, nous utilisons la notation indicée pour représenter la base d'un nombre en parallèle de la représentation avec la barre au dessus. Ainsi 145_{10} signifie le nombre 145 en base dix; 1101011_2 signifie 1101011 en binaire.

2.1 Opérations en binaire

Nous avons parlé d'addition en binaire ; comme dans le système décimal, il nous faut connaître les tables d'addition, de multiplication, etc... afin d'effectuer des calculs dans cette base. Heureusement en binaire, elles sont très simples :

Addition

+	**0**	**1**
0	0	1
1	1	0(1)

Multiplication

*	**0**	**1**
0	0	0
1	0	1

0(1) représente la retenue 1 à reporter.

*Exemples de calculs (109+19=**128_{10}** =10000000_2) et (22x5=**110**) :*

addition

```
  1101101
+   10011
---------
10000000₂ =128₁₀
```

multiplication

```
    10110
x     101
---------
    10110
  10110..
---------
1101110₂ =110₁₀
```

Vous noterez que le procédé est identique à celui que vous connaissez en décimal. En hexadécimal (b=16) il en est de même. Dans ce cas les tables d'opérateurs sont très longues à apprendre.

Etant donné que le système classique utilisé par chacun de nous est le système décimal, nous nous proposons de fournir d'une manière pratique les conversions usuelles permettant d'écrire les diverses représentations d'un nombre entre les systèmes décimal, binaire et hexadécimal.

2.2 Conversions base quelconque ⇔ décimal

Voici ci-dessous un rappel des méthodes générales permettant de convertir un nombre en base b (b>1)en sa représentation décimale et réciproquement.

A) Soit $\overline{x_n x_{n-1} \cdots x_0}^{b}$ un nombre écrit en base b.

Pour le convertir en décimal (base 10), il faut :

- **convertir chaque symbole x_k en son équivalent** a_k en base 10, nous obtenons ainsi la suite de chiffres : $a_n,....,a_0$

Exemple, soit b=13, ***les symboles de la base 13 sont :*** **{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C }**

Si le chiffre $\mathbf{x_k}$ de rang k du nombre s'écrit **C**, son équivalent en base 10 est a_k=12

- réécrire le nombre comme une somme :

$$\overline{x_n x_{n-1} \cdots x_0}^{b} \equiv \sum_{k=0}^{n} x_k\, b^k \quad \xrightarrow[\text{(en base 10)}]{} \quad \sum_{k=0}^{n} a_k\, b^k$$

- effectuer tous les calculs en base 10 (somme, produit, puissance).

Exemple : $\overline{2AB8}^{13}$ **à convertir en base10 (b=13)**

Le nombre $\overline{2\ A\ B\ 8}^{13} \equiv 2.13^3 + 10.13^2 + 11.13^1 + 8.13^0$ (3 2 1 0)

$= 4394 + 1690 + 143 + 8 = \overline{6235}^{10}$

B) Soit " **a** " un nombre écrit décimal et à représenter en base **b** :

La méthode utilisée est un algorithme fondé sur la division euclidienne.

- **Si a < b**, il n'a pas besoin d'être converti.
- **Si a = b**, on peut diviser a par b. Et l'on divise successivement les différents quotients $\mathbf{q_k}$ obtenus par la base b.

De manière générale on aura :

$\mathbf{a = b^k.r_k + b^{k-1}.r_{k-1} + ... + b.r_1 + r_0}$ (où $\mathbf{r_i}$ est le reste de la division de a par b).

En remplaçant chaque $\mathbf{r_i}$ par son symbole équivalent $\mathbf{p_i}$ en base **b**, nous obtenons :

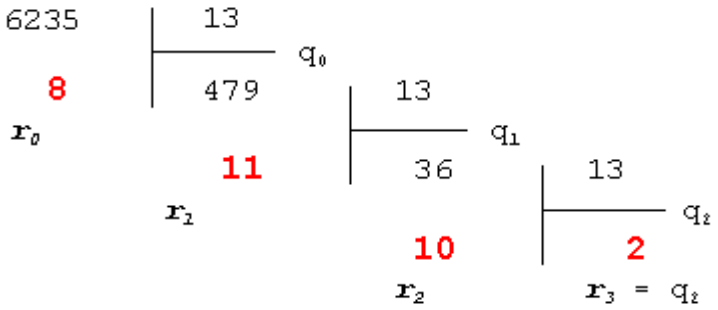
$$a = \overline{p_k\ p_{k-1} \cdots p_1\ p_0}^{b}$$

Cet algorithme permet d'obtenir une représentation de a dans la base b.

Exemple :

6235_{10} **à convertir en base b=13**

les symboles de la base 13 sont: {0,1,2,3,4,5,6,7,8,9,A,B,C}



Les $\mathbf{p_i}$ *équivalents en base 13 sont:*

$\mathbf{r_0 = 8} \rightarrow \mathbf{p_0 = 8}$

$\mathbf{r_1 = 11} \rightarrow \mathbf{p_1 = B}$

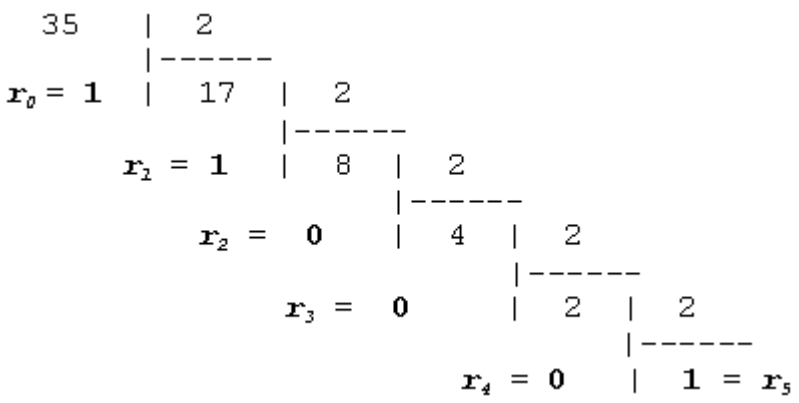
$\mathbf{r_2 = 10} \rightarrow \mathbf{p_2 = A}$

$\mathbf{r_3 = 2} \rightarrow \mathbf{p_3 = 2}$ Donc $\mathbf{6235_{10} = 2AB8_{13}}$

Dans les deux paragraphes suivants nous allons expliciter des exemples pratiques de ces méthodes dans le cas où la base est 2 (binaire).

2.3 Exemple de conversion décimal → binaire

Soit le nombre décimal $\mathbf{35_{10}}$, appliquons l'algorithme précédent afin de trouver les restes successifs :



Donc : $\mathbf{35_{10}} = \mathbf{100011_2}$

2.4 Exemple de conversion binaire → décimal

Soit le nombre binaire : **1101101**$_2$
sa conversion en décimal est immédiate :

1101101$_2$ = $2^6 + 2^5 + 2^3 + 2^3 + 2^2 + 1 = 64+32+8+4+1 =$ **109**$_{10}$

Les informaticiens, pour des raisons de commodité (manipulations minimales de symboles), préfèrent utiliser l'hexadécimal plutôt que le binaire. L'humain, contrairement à la machine, a quelques difficultés à fonctionner sur des suites importantes de 1 et de 0. Ainsi l'hexadécimal (sa base b=2^4 étant une puissance de 2) permet de diviser, en moyenne, le nombre de symboles par un peu moins de 4 par rapport au même nombre écrit en binaire. C'est l'unique raison pratique qui justifie son utilisation ici.

2.5 Conversion binaire → hexadécimal

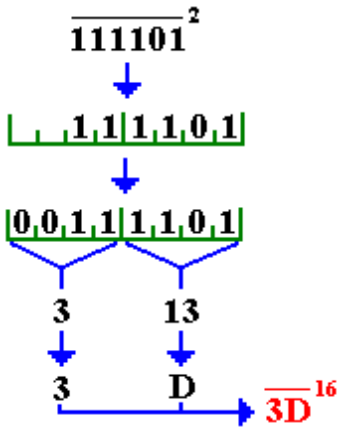
Nous allons détailler l'action de conversion en 6 étapes pratiques :

- Soit a un nombre écrit en **base 2** (***étape 1***).
- On décompose ce nombre par tranches de 4 bits à partir du bit de poids faible (***étape 2***).
- On complète la dernière tranche (celle des bits de poids forts)par des 0 s'il y a lieu (***étape 3***).
- On convertit chaque tranche en son symbole de la **base 16**(***étape 4*****).**
- On réécrit à sa place le nouveau symbole par changements successifs de chaque groupe de 4 bits,(***étape 5***).
- Ainsi, on obtient le nombre écrit en hexadécimal (***étape 6***).

Exemple :

Soit le nombre **111101**$_2$

à convertir en héxadécimal.



Résultat obtenu :

111101$_2$ = **3D**$_{16}$

2.6 Conversion hexadécimal → binaire

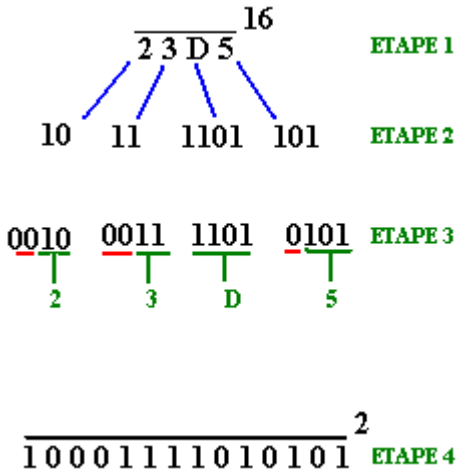
Cette conversion est l'opération inverse de la précédente. Nous allons la détailler en 4 étapes :

- Soit a un nombre écrit en **base 16** (***ETAPE 1***).
- On convertit chaque symbole hexadécimal du nombre en son écriture binaire (nécessitant au plus 4 bits) (***ETAPE 2***).
- Pour chaque tranche de 4 bits, on complète les bits de poids fort par des 0 s'il y a lieu (***ETAPE 3***).
- Le nombre " **a** " écrit en binaire est obtenu en regroupant toutes les tranches de 4 bits à partir du bit de poids faible, sous forme d'un seul nombre binaire(***ETAPE 4***).

Exemple :

Soit le nombre **23D5**$_{16}$

à convertir en binaire.



Résultat obtenu :

23D5$_{16}$ = **10001111010101**$_2$

1.4 Formalisation de la notion d'ordinateur

Plan du chapitre:

1. La Machine de Turing théorique

Entre 1930 et 1936 le mathématicien anglais A.Turing invente sur le papier une machine fictive qui ne pouvait effectuer que 4 opérations élémentaires que nous allons décrire. Un des buts de Turing était de faire résoudre par cette " machine " des problèmes mathématiques, et d'étudier la classe des problèmes que cette machine pourrait résoudre.

Définitions et notations (modèle déterministe)
Soit A un ensemble fini appelé *alphabet* défini ainsi :

$A = \{ a_1, ..., a_n \}$ $(A \neq \varnothing)$
Soit $\Omega = \{ D,G \}$ une paire

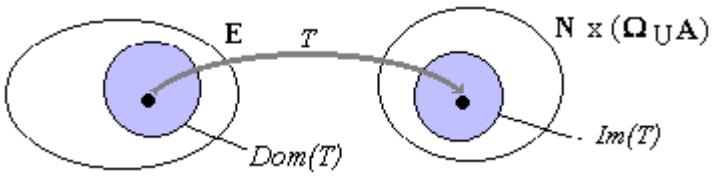
1.1 Définition : machine de Turing

Nous appellerons machine de Turing toute application T :

$T : E \rightarrow \mathbf{N} \times (\Omega \cup A)$
où E est un ensemble fini non vide : $E \subset \mathbf{N} \times A$

1.2 Définition : Etats de la machine

Nous appellerons E_t ensemble des états intérieurs de la machine T:
$E_t = \{ q_i \in \mathbf{N} , (\exists a_i \in A / (q_i , a_i) \in Dom(T)) \text{ où } (\exists x \in \Omega / (q_i , x) \in Im(T)) \}$



Dom(T) : domaine de définition de T.
Im(T) : image de T (les éléments T(a) de $\mathbf{N} \times (\Omega \cup A)$, pour $a \in E$)

Comme E est un ensemble fini, E_t est nécessairement un ensemble fini, donc il y a un nombre fini d'états intérieurs notés q_i.

1.3 Définition : Les règles de la machine

Nous appellerons " ensemble des règles " de la machine T, le graphe G de l'application T. Une règle de T est un élément du graphe G de T.

On rappelle que le graphe de T est défini comme suit :

$G = \{(a,b) \in E \times [E_t \times (\Omega \cup A)] / b = T(a) \}$

- **Notation** : afin d'éviter une certaine lourdeur dans l'écriture nous conviendrons d'écrire les règles en omettant les virgules et les parenthèses.

- **Exemple** : la règle $((q_i,a) , (q_k,b))$ est notée : $q_i\, a\, q_k\, b$

2. La Machine de Turing physique

2.1 Constitution interne

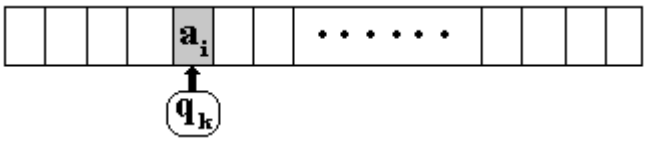
Nous construisons une machine de Turing physique constituée de :

- Une boîte notée UC munie d'une tête de lecture-écriture et d'un registre d'état.
- Un ruban de papier supposé sans limite vers la gauche et vers la droite.
- Sur le ruban se trouvent des cases contiguës contenant chacune un seul élément de l'alphabet A.
- La tête de lecture-écriture travaille sur la case du ruban située devant elle ; elle peut lire le contenu de cette case ou effacer ce contenu et y écrire un autre élément de A.
- Il existe un dispositif d'entraînement permettant de déplacer la tête de lecture-écriture d'une case vers la **Droite** ou vers la **Gauche**.
- Dans la tête lecture-écriture il existe une case spéciale notée ***registre d'état***, qui sert à recevoir un élément q_i de E_t.

Cette machine physique est une représentation virtuelle d'une machine de Turing théorique T, d'alphabet A, dont l'ensemble des états est E_t, dont le graphe est donné ci-après :

$G = \{(a,b) \in E \times [E_t \times (\Omega \cup A)] / b = T(a) \}$

Donnons une visualisation schématique d'une telle machine en cours de fonctionnement. La tête de lecture/écriture pointe vers une case contenant l'élément $\mathbf{a_i}$ de A, le registre d'état ayant la valeur $\mathbf{q_k}$:



2.2 Fonctionnement

Départ : On remplit les cases du ruban d'éléments a_i de A. On met la valeur " q_k " dans le registre d'état. On positionne la tête sur une case contenant " a_i ".

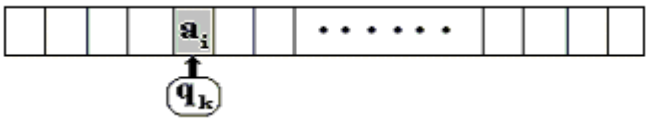
Actions : (la machine se met en marche)

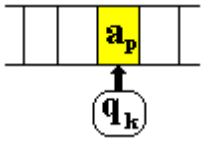
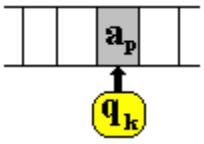
La tête lit le " a_i ". L'UC dont le registre d'état vaut " q_k ", cherche dans la liste des règles si le couple $(q_k , a_i) \in Dom(T)$.

Si la réponse est *négative* on dit que la machine " bloque " (elle s'arrête par blocage).

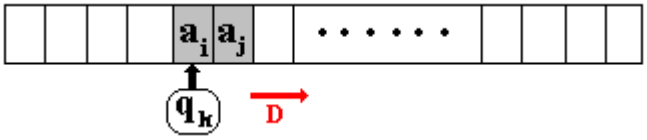
Si la réponse est *positive* alors le couple (q_k , a_i) a une image unique (machine déterministe)que nous notons $(q_n , \mathbf{y})$. Dans ce couple, **y** ne peut prendre que l'un des 3 types de valeurs possibles $\mathbf{a_p}$, **D** , **G** :

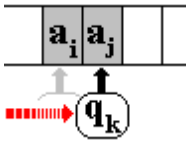
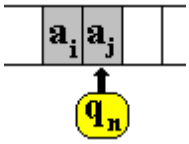
- ***a)*** **soit $y=a_p$, dans ce cas la règle est donc de la forme $q_k\, a_i\, q_n\, a_p$**



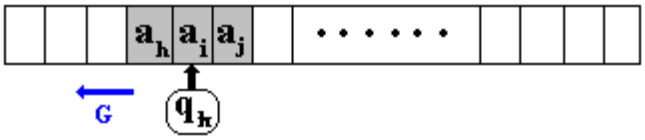
a.1) L'UC fait effacer le $\mathbf{a_i}$ dans la case et le remplace par l'élément $\mathbf{a_p}$.	
a.2) L'UC écrit $\mathbf{q_n}$ dans le registre d'état en remplacement de la valeur $\mathbf{q_k}$.	

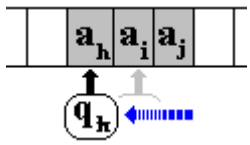
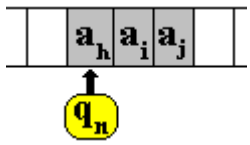
- ***b)*** **soit y = D , ici la règle est donc de la forme $q_k\, a_i\, q_n\, D$**



b.1) L'UC fait déplacer la tête vers la droite d'une case.	
b.2) L'UC écrit $\mathbf{q_n}$ dans le registre d'état en remplacement de la valeur $\mathbf{q_k}$.	

- ***c)* soit y = G , dans ce cas la règle est donc de la forme $q_k\, a_i\, q_n\, G$**



c.1) L'UC fait déplacer la tête vers la gauche d'une case.	
c.2) L'UC écrit q_n dans le registre d'état en remplacement de la valeur $\mathbf{q_k}$.	

Puis la machine continue à fonctionner en recommençant le cycle des actions depuis le début : lecture du nouvel élément a_k etc...

2.3 Exemple : machine de Turing arithmétique

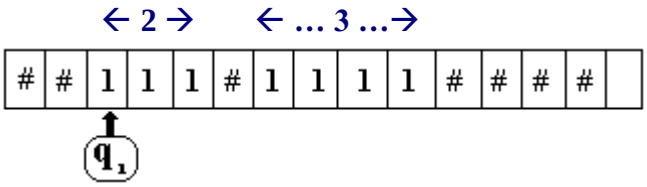
Nous donnons ci-dessous une machine T_1 additionneuse en arithmétique unaire.

A = {#,1}

Ω ={D,G}

un entier n est représenté par n+1 symboles " **1** " consécutifs (de façon à pouvoir représenter " 0 " par un unique " 1 ").

Etat initial du ruban avant actions :



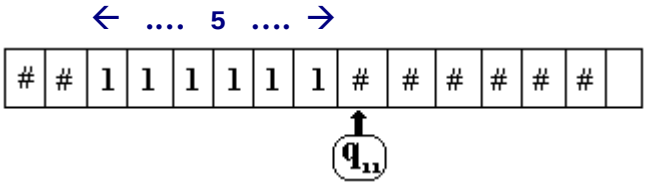
2 est représenté par ***111***
3 est représenté par ***1111***

Règles T_1: (application des règles suivantes pour simulation de 2+3)

q_1 **1** q_2**D**	q_6 **1** q_7**D**	q_{11} **1** q_{12}**#**
q_2 **1** q_3**D**	q_7 **1** q_8**D**	q_{12} **#** q_{13}**G**
q_3 **1** q_4**D**	q_8 **1** q_9**D**	q_{13} **1** q_{14}**#**
q_4 **#** q_5**1**	q_9 **1** q_{10}**D**	
q_5 **1** q_6**D**	q_{10} **#** q_{11}**G**	

En démarrant la machine sur la configuration précédente on obtient :

Etat final du ruban après actions : (Cette machine ne fonctionne que pour additionner 2 et 3)



.

Généralisation de la machine additionneuse

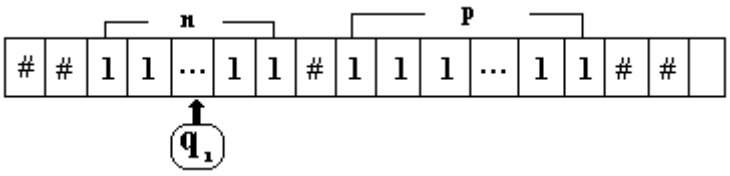
Il est facile de fournir une autre version plus générale **T_2** fondée sur la même stratégie et le même état initial permettant d'additionner non plus seulement les nombres 2 et 3, mais des nombres entiers quelconques **n** et **p**. Il suffit de construire des nouvelles règles.

Règles de T_2:

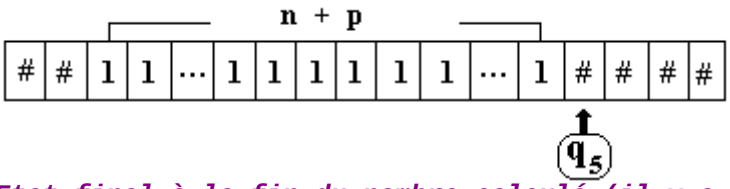
q_1 **1** q_1**D**	q_3 **1** q_3**#**
q_1 **#** q_2**1**	q_3 **#** q_4**G**
q_2 **1** q_2**D**	q_4 **1** q_5**#**
q_2 **#** q_3**G**	

Cette machine de Turing $\mathbf{T_2}$ appliquée à l'exemple précédent (addition de 2 et 3) laisse le ruban dans le même état final, mais elle est construite avec moins d'états intérieurs que la précédente.

En fait elle fonctionne aussi si la tête de lecture-écriture est positionnée sur n'importe lequel des éléments du premier nombre **n**. et les nombres **n** et **p** sont quelconques :



Etat initial sur le nombre de gauche



Etat final à la fin du nombre calculé (il y a n+p+1 symboles " 1 ")

Nous dirons que T_2 est plus " *puissante* " que T_1 au sens suivant :

- T_2 a moins d'états intérieurs que T_1 .
- T_2 permet d'additionner des entiers quelconques.
- Il est possible de démarrer l'état initial sur n'importe lequel des " 1 " du nombre de gauche.

On pourrait toujours en ce sens chercher une machine T_3 qui posséderait les qualités de T_2 , mais qui pourrait démarrer sur n'importe lequel des " 1 " de l'un ou l'autre des deux nombres **n** ou **p**, le lecteur est encouragé à chercher à écrire les règles d'une telle machine.

Nous voyons que ces machines sont capables d'effectuer des opérations, elles permettent de définir la classe des **fonctions calculables** (par machines de Turing).

Un ordinateur est fondé sur les principes de calcul d'une machine de Turing. J. Von Neumann a défini la structure générale d'un ordinateur à partir des travaux de A.Turing. Les éléments physiques supplémentaires que possède un ordinateur moderne n'augmentent pas sa puissance théorique. Les fonctions calculables sont les seules que l'on puisse implanter sur un ordinateur. Les périphériques et autres dispositifs auxiliaires extérieurs ou intérieurs n'ont pour effet que d'améliorer la " *puissance* " en terme de vitesse et de capacité. Comme une petite voiture de série et un bolide de formule 1 partagent les mêmes concepts de motorisation, de la même manière les différents ordinateurs du marché partagent les mêmes fondements mathématiques.

2.4 Machine de Turing informatique

Nous faisons évoluer la représentation que nous avons de la machine de Turing afin de pouvoir mieux la programmer, c'est à dire pouvoir écrire plus facilement les règles de fonctionnement d'une telle machine.

Nous définissons ce qu'est un algorithme pour une machine de Turing et nous proposons une description graphique de cet algorithme à l'aide de schémas graphiques symboliques décrivant des actions de base d'une machine de Turing.

A) Une machine de Turing informatique est dite normalisée au sens suivant :

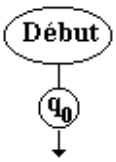
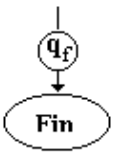
- L'alphabet A contient toujours le symbole " # "
- L'ensemble des états E contient toujours deux états distingués $\mathbf{q_0}$ (état initial) et $\mathbf{q_f}$ (état final).
- La machine démarre toujours à l'état initial $\mathbf{q_0}$.
- Elle termine sans blocage toujours à l'état $\mathbf{q_f}$.
- Dans les autres cas on dit que la machine " bloque ".

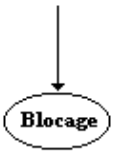
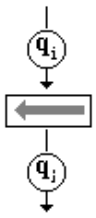
B)Algorithme d'une machine de Turing informatique

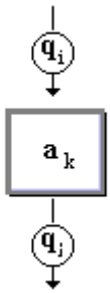
C'est l'ensemble des règles précises qui définissent un procédé de calcul destiné à obtenir en sortie un " **résultat** " déterminé à partir de certaines " **données** " initiales.

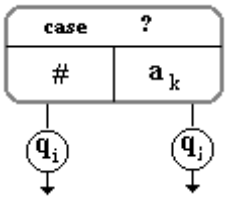
C) Algorithme graphique d'une machine de Turing

Nous utilisons cinq classes de symboles graphiques

	Positionne la tête de lecture sur le symbole voulu, met la machine à l'état initial q_0 et fait démarrer la machine.
	Signifie que la machine termine correctement son calcul en s'arrêtant à l'état final q_f .

	Aucune règle de la machine ne permettant la poursuite du fonctionnement, arrêt de la machine sur un blocage.
	Déplacer la tête d'une case vers la gauche (la tête de lecture se positionne sur la case d'avant la case actuelle contenant le symbole a_p). Correspond à la règle : **q_i a_p q_j G**

	Correspond à l'action à exécuter dans la deuxième partie de la règle : **q_i a_p q_j a_k** (la machine écrit **a_k** dans la case actuelle et passe à l'état **q_j**).

	Correspond à l'action à exécuter dans la première partie de la règle : **q_j a_k ...** ou **q_i # ...** (la machine teste le contenu de la case actuelle et passe à l'état **q_j** ou à l'état **q_i** selon la valeur du contenu).

D) Organigramme d'une machine de Turing

On appelle organigramme d'une machine de Turing T, toute représentation graphique constituée de combinaisons des symboles des **cinq classes précédentes**.

Les règles de la forme q_n **a_k** q_n **G** ou q_n**a_k**q_n**D** se traduisent par des schémas " bouclés " ce qui donne des organigrammes non linéaires.

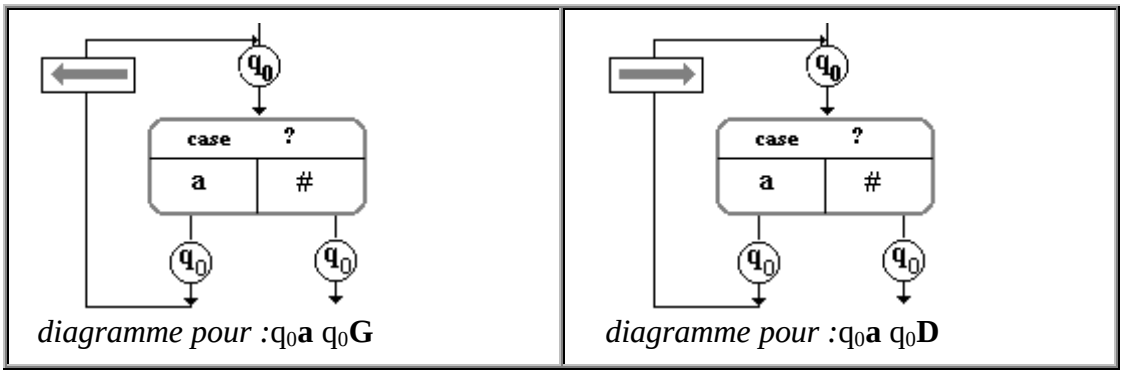
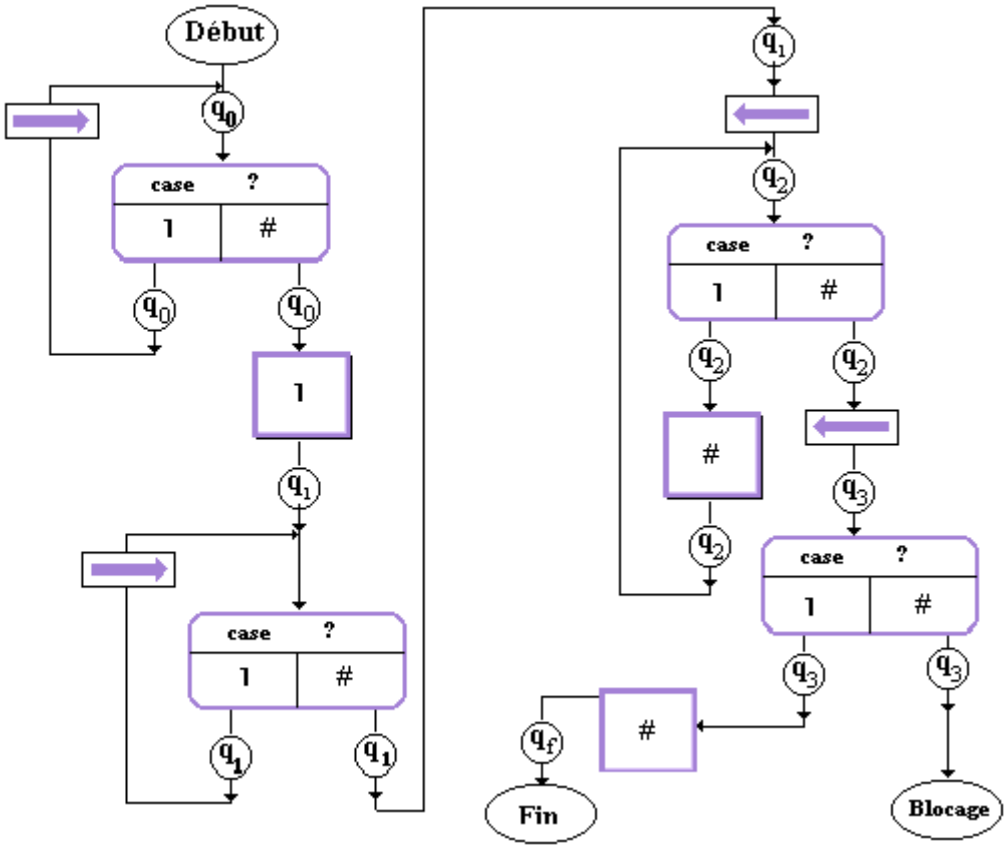


diagramme pour : q_0**a** q_0**G** | *diagramme pour :* q_0**a** q_0**D**

Exemple : organigramme de la machine T_2 normalisée(additionneuse n+p)



Règles de T_2 normalisée :

q_0 **1** q_0**D**	q_2 **1** q_2**#**
q_0 **#** q_1**1**	q_2 **#** q_3**G**
q_1 **1** q_1**D**	q_3 **1** q_f**#**
q_1 **#** q_2**G**	

Nous voyons que ce symbolisme graphique est un outil de description du mode de traitement de l'information au niveau machine. C'est d'ailleurs historiquement d'une façon semblable que les premiers programmeurs décrivaient leur programmes.

1.5 Architecture de l'ordinateur

Plan du chapitre:

Les principaux constituants

1.1 Dans l'Unité Centrale : l'unité de traitement
1.2 Dans l'Unité Centrale : l'unité de commande
1.3 Dans l'Unité Centrale : les Unités d'échange
1.4 Exemples de machine à une adresse : un micro-processeur simple
1.5 Les Bus
1.6 Schéma général d'une micro-machine fictive
1.7 Notion de jeu d'instructions-machine
1.8 Architectures RISC et CISC
1.9 Pipe line dans un processeur
1.10 Architectures super-scalaire
1.11 Principaux modes d'adressages des instructions machines

2. Mémoires : Mémoire centrale - Mémoire cache

2.1 Mémoire
2.2 Les différents types de mémoires
2.3 Les unités de capacité
2.4 Mémoire centrale : définitions
2.5 Mémoire centrale : caractéristiques
2.6 Mémoire cache (ECC, associative)

3. Une petite machine pédagogique 8 bits " PM "

3.1 Unité centrale de PM (pico-machine)
3.2 Mémoire centrale de PM
3.3 Jeu d'instructions de PM

4. Mémoire de masse (auxiliaire ou externe)

4.1 Disques magnétiques - disques durs
4.2 Disques optiques compacts - CD / DVD

1. Les principaux constituants d'une machine minimale

Un ordinateur, nous l'avons déjà noté, est composé d'un centre et d'une périphérie. Nous allons nous intéresser au cœur d'un ordinateur fictif mono-processeur. Nous savons que celui-ci est composé de :

Une Unité centrale comportant :

- Unité de traitement,
- Unité de contrôle,
- Unités d'échanges.
- Une Mémoire Centrale.

Nous décrivons ci-après l'architecture minimale illustrant simplement le fonctionnement d'un ordinateur (machine de Von Neumann).

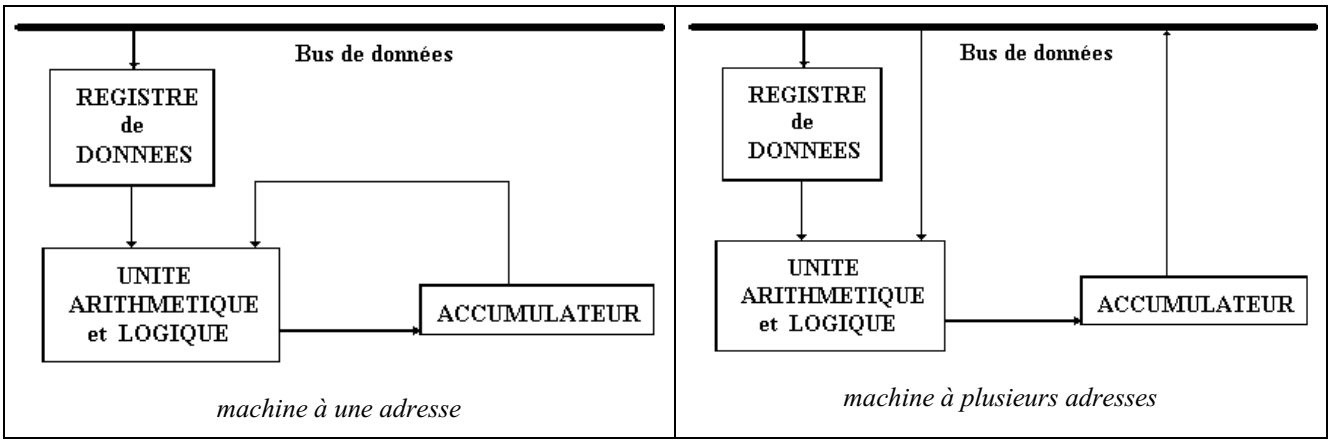
1.1 Dans l'Unité Centrale : l'Unité de Traitement

Elle est chargée d'effectuer les traitements des opérations de types arithmétiques ou booléennes. L'UAL est son principal constituant.

Elle est composée au minimum :

- d'un registre de données **RD**
- d'un accumulateur **ACC** (pour les machines à une adresse)
- des registres spécialisés (pour les machines à plusieurs adresses)
- d'une unité arithmétique et logique : **UAL**

Schéma général théorique de l'unité de traitement :



La fonction du registre de données (mémoire rapide) est de contenir les données transitant entre l'unité de traitement et l'extérieur.

La fonction de l'accumulateur est principalement de contenir les opérandes ou les résultats des opérations de l'UAL.

La fonction de l'UAL est d'effectuer en binaire les traitements des opérations qui lui sont soumises et qui sont au minimum:

- Opérations arithmétiques binaires: addition,multiplication, soustraction, division.
- Opérations booléennes : et, ou, non.
- Décalages dans un registre.

Le résultat de l'opération est mis dans l'accumulateur (Acc) dans le cas d'une machine à une adresse, dans des registres internes dans le cas de plusieurs adresses.

*1.2 Dans l'Unité Centrale : l'***Unité de Contrôle** *ou de Commande*

Elle est chargée de commander et de gérer tous les différents constituants de l'ordinateur (contrôler les échanges, gérer l'enchaînement des différentes instructions, etc...)

Elle est composée au minimum de :

- d'un registre instruction **RI**,
- d'un compteur ordinal **CO**,
- d'un registre adresse **RA**,
- d'un décodeur de fonctions,
- d'une horloge.

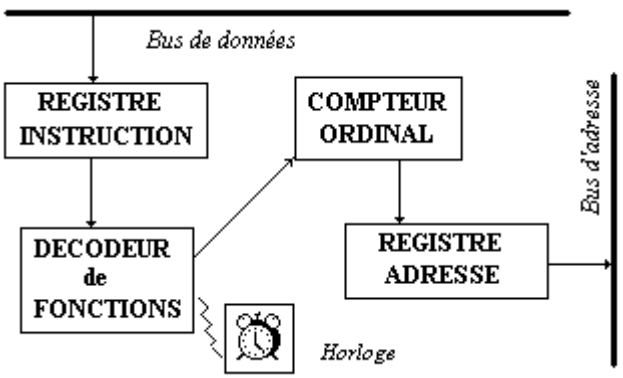


Schéma général de l'unité de contrôle

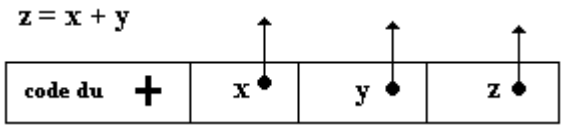
Vocabulaire **:**

Bit = plus petite unité d'information binaire (un objet physique ayant deux états représente un bit).

Processeur central = unité de commande + unité de traitement. IL a pour fonction de lire séquentiellement les instructions présentes dans la mémoire, de décoder une instruction, de lire, écrire et traiter les données situées dans la mémoire.

Instruction = une ligne de texte comportant un code opération, une ou plusieurs références aux opérandes.

Soit l'instruction fictive d'addition du contenu des deux mémoires x et y dont le résultat est mis dans une troisième mémoire z :



(exemple d'instruction à trois adresses)

| *Opérateur* | *... références opérandes...* |

Registre instruction = contient l'instruction en cours d'exécution, elle demeure dans ce registre pendant toute la durée de son exécution.

Compteur ordinal = contient le moyen de calculer l'adresse de la prochaine instruction à exécuter.

Registre adresse = contient l'adresse de la prochaine instruction à exécuter.

Décodeur de fonction = associé au registre instruction, il analyse l'instruction à exécuter et entreprend les actions appropriées dans l'UAL ou dans la mémoire centrale.

Au début, la différentiation des processeurs s'effectuait en fonction du nombre d'adresses contenues dans une instruction machine. De nos jours, un micro-processeur comme le pentium par exemple, possède des instructions une adresse, à deux adresses, voir à trois adresses dont certaines sont des

registres. En fait deux architectures machines coexistent sur le marché : l'architecture RISC et l'architecture CISC, sur lesquelles nous reviendrons plus loin. Historiquement l'architecture CISC est la première, mais les micro-processeur récents semblent utiliser un mélange de ces deux architectures profitant ainsi du meilleur de chacune d'elle.

Il existe de très bons ouvrages spécialisés uniquement dans l'architecture des ordinateurs nous renvoyons le lecteur à certains d'entre eux cités dans la bibliographie. Dans ce chapitre notre objectif est de fournir au lecteur le vocabulaire et les concepts de bases qui lui sont nécessaires et utiles sur le domaine, ainsi que les notions fondamentales qu'il retrouvera dans les architectures de machines récentes. L'évolution matérielle est actuellement tellement rapide que les ouvrages spécialisés sont mis à jour en moyenne tous les deux ans.

1.3 Dans l'Unité Centrale : les Unités d'échange

- Une unité d'échange est spécialisée dans les entrées/sorties.
- Ce peut être un simple canal, un circuit ou bien un processeur particulier.
- Cet organe est placé entre la mémoire et un certain nombre de périphériques (dans un micro-ordinateur ce sont des cartes comme la carte son, la carte vidéo, etc...).

Une unité d'échange soulage le processeur central dans les tâches de gestion du transfert de l'information.

Les périphériques sont très lents par rapport à la vitesse du processeur (rapport de 1 à 10^9). Si le processeur central était chargé de gérer les échanges avec les périphériques il serait tellement ralenti qu'il passerait le plus clair de son temps à attendre.

1.4 **Exemple** *de machine à une adresse : un micro-processeur simple*

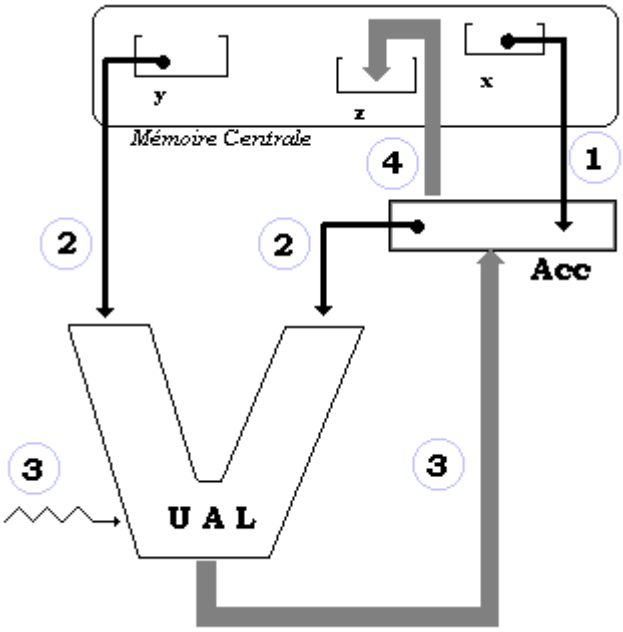
Un micro-processeur simple a les mêmes caractéristiques que celles d'un processeur central avec un niveau de complexité et de sophistication moindre. Il faut savoir que plus une instruction machine contient d'adresses (de références à des opérandes), plus le processeur est complexe. En effet avec les instructions à une adresse, le processeur est moins complexe en contre partie les programmes (listes d'instructions machines) contiennent beaucoup d'instructions et sont donc plus longs à exécuter que sur un matériel dont le jeu d'instruction est à plusieurs adressses.

Un micro-processeur simple est essentiellement une machine à une adresse, c'est à dire une partie code opérande et une référence à un seul opérande. Ce genre de machine est fondé sur un cycle de passage par l'accumulateur.

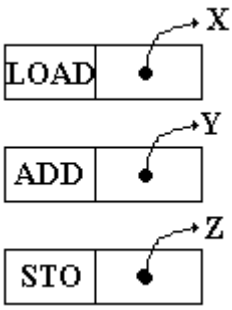
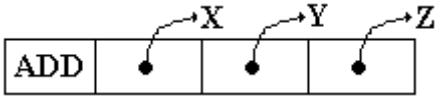
L'opération précédente z = x + y , se décompose dans une telle machine fictivement en 3 opérations distinctes illustrées par la figure ci-après :

LoadAcc x { chargement de l'accumulateur avec x : **(1)** }
Add y { préparation des opérandes x et y vers l'UAL : **(2)** }
{ lancement commande de l'opération dans l'UAL : **(3)** }
{ résultat transféré dans l'accumulateur : **(3)** }
Store z { copie de l'accumulateur dans z : **(4)** }

L'accumulateur gardant son contenu au final.



Comparaison de "programme" réalisant le calcul de l'opération précédente "z = x + y "avec une machine à une adresse et une machine à trois adresses :

Une machine à une adresse *(3 instructions)*	Une machine à trois adresses *(1 instruction)*
	

1.5 Les Bus

Un bus est un dispositif destiné à assurer le transfert simultané d'informations entre les divers composants d'un ordinateur.

On distingue trois catégories de Bus :

Bus d'adresses (unidirectionnel) il permet à l'unité de commande de transmettre les adresses à rechercher et à stocker.

Bus de données (bi-directionnel)

sur lequel circulent les instructions ou les données à traiter ou déjà traitées en vue de leur rangement.

Bus de contrôle (bi-directionnel)

transporte les ordres et les signaux de synchronisation provenant de l'unité de commande vers les divers organes de la machine. Il véhicule aussi les divers signaux de réponse des composants.

Largeur du bus

Pour certains Bus on désigne par largeur du Bus, le nombre de bits qui peuvent être transportés en même temps par le Bus, on dit aussi transportés en parallèle.

Les principaux Bus de données récents de micro-ordinateur

Les Bus de données sont essentiellement des bus "synchrones", c'est à dire qu'ils sont cadencés par une horloge spécifique qui fonctionne à un fréquence fixée. Entre autres informations commerciales, les constructeurs de Bus donnent en plus de la fréquence et pour des raison psychologiques, le débit du Bus qui est en fait la valeur du produit de la fréquence par la largeur du Bus, ce débit correspond au nombre de bits par seconde transportés par le Bus.

Quelques chiffres sur des Bus de données parallèles des années 2000

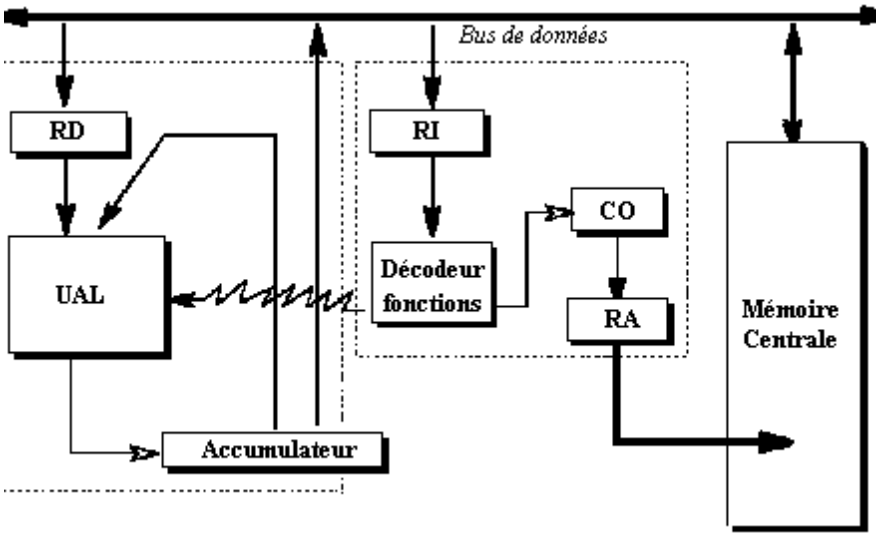
BUS	Largeur	Fréquence	Débit	Utilisation
PCI	64 bits	66 MHz	528 Mo/s	Processeur/périphérique non graphique
AGP	32 bits	66 MHz x 8	4 Go/s	Processeur/carte graphique
SCSI	16 bits	40 MHz	80 Mo/s	Echanges entres périphériques

Il existe aussi des "Bus série" (Bus qui transportent les bits les uns à la suite des autres, contrairement aux Bus parallèles), les deux plus récents concurrents équipent les matériels de grande consommation : USB et Firewire.

BUS	Débit	Nombre de périphériques acceptés
USB	1,5 Mo/s	127
USB2	60 Mo/s	127
Firewire	50 Mo/s	63
FirewireB	200 Mo/s	63

Ces Bus évitent de connecter des périphériques divers comme les souris, les lecteurs de DVD, les GSM, les scanners, les imprimantes, les appareils photo, …, sur des ports spécifiques de la machine

1.6 Schéma général d'une micro-machine fictive à une adresse



1.7 Notion de jeu d'instructions-machine : *Les premiers programmes*

Comme défini précédemment, une instruction-machine est une instruction qui est directement exécutable par le processeur.

L'ensemble de toutes les instructions-machine exécutables par le processeur s'appelle le " jeu d'instructions " de l'ordinateur. Il est composé au minimum de quatre grandes classes d'instructions dans les micro-processeurs :

- instructions de traitement
- instructions de branchement ou de déroutement
- instructions d'échanges
- instructions de comparaisons

D'autres classes peuvent être ajoutées pour améliorer les performances de la machine (instructions de gestion mémoire, multimédias etc..)

1.8 Architectures CISC et RISC

Traditionnellement, depuis les années 70 on dénomme processeur à architecture CISC (**C**omplex **I**nstruction **S**et **C**ode) un processeur dont le jeu d'instructions possède les propriétés suivantes :

- Il contient beaucoup de classes d'instructions différentes.
- Il contient beaucoup de type d'instructions différentes complexes et de taille variable.
- Il se sert de beaucoup de registres spécialisés et de peu de registres généraux.

L'architecture RISC (**R**educed **I**nstruction **S**et **C**ode) est un concept mis en place par IBM dans les

années 70, un processeur RISC est un processeur dont le jeu d'instructions possède les propriétés suivantes :

- Le nombre de classes d'instructions différentes est réduit par rapport à un CISC.
- Les instructions sont de taille fixe.
- Il se sert de beaucoup de registres généraux.
- Il fonctionne avec un pipe-line

Depuis les décennies 90, les microprocesseur adoptent le meilleur des fonctionnalités de chaque architecture provoquant de fait la disparition progressive de la différence entre RISC et CISC et le inévitables polémiques sur l'efficacité supposée meilleure de l'une ou de l'autre architecture.

1.9 Pipe-line dans un processeur

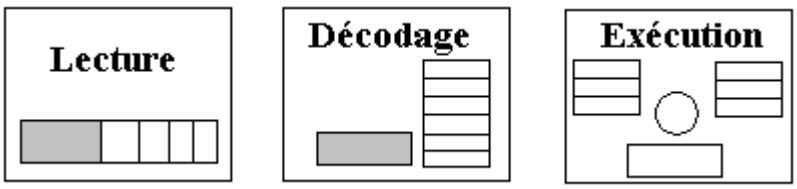
Soulignons qu'un processeur est une machine séquentielle ce qui signifie que le cycle de traitement d'une instruction se déroule séquentiellement. Supposons que par hypothèse simplificatrice, une instruction machine soit traitée en 3 phases :

1 - lecture : dans le registre instruction (RI)

2 - décodage : extraction du code opération et des opérandes

3 - exécution : du traitement et stockage éventuel du résultat.

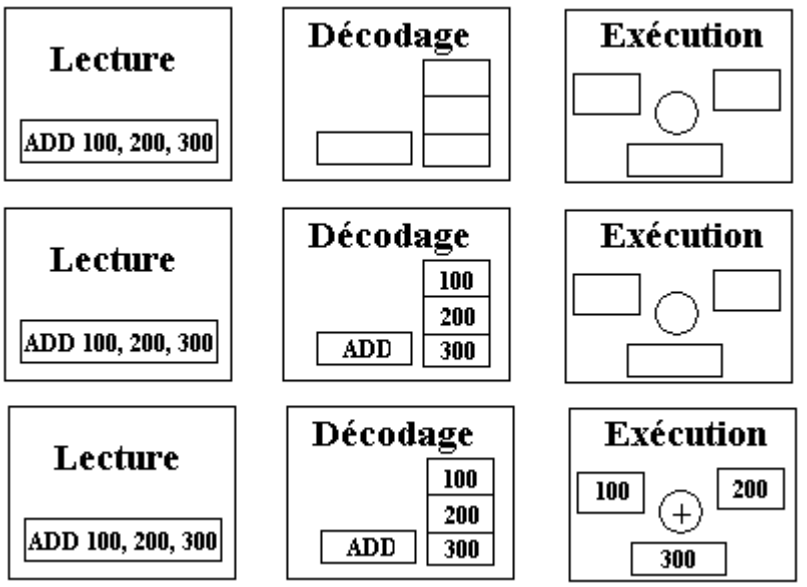
Représentons chacune de ces 3 phases par une unité matérielle distinctes dans le processeur (on appelle cette unité un "**étage**") et figurons schématiquement les 3 étages de traitement d'une instruction :



Supposons que suivions pas à pas l'exécution des 4 instructions machines suivants le long des 3 étages précédents :

ADD 100, 200, 300
MUL 150, 250, 350
DIV 300, 200, 120
MOV 100, 500

Chacune des 4 instruction est traitée séquentiellement en 3 phases sur chacun des étages; une fois une instruction traitée par le dernier étage (étage d'exécution) le processeur passe à l'instruction suivante et la traite au premier étage et ainsi de suite :



etc …

Traitement de la première instruction ADD 100, 200, 300

On remarquera que :

- Pendant le temps d'activation d'un étage, les deux autres restent inactifs.
- Il faut attendre la fin du traitement de l'instruction ADD 100, 200, 300 pour pouvoir passer au traitement de l'instruction MUL 150, 250, 350

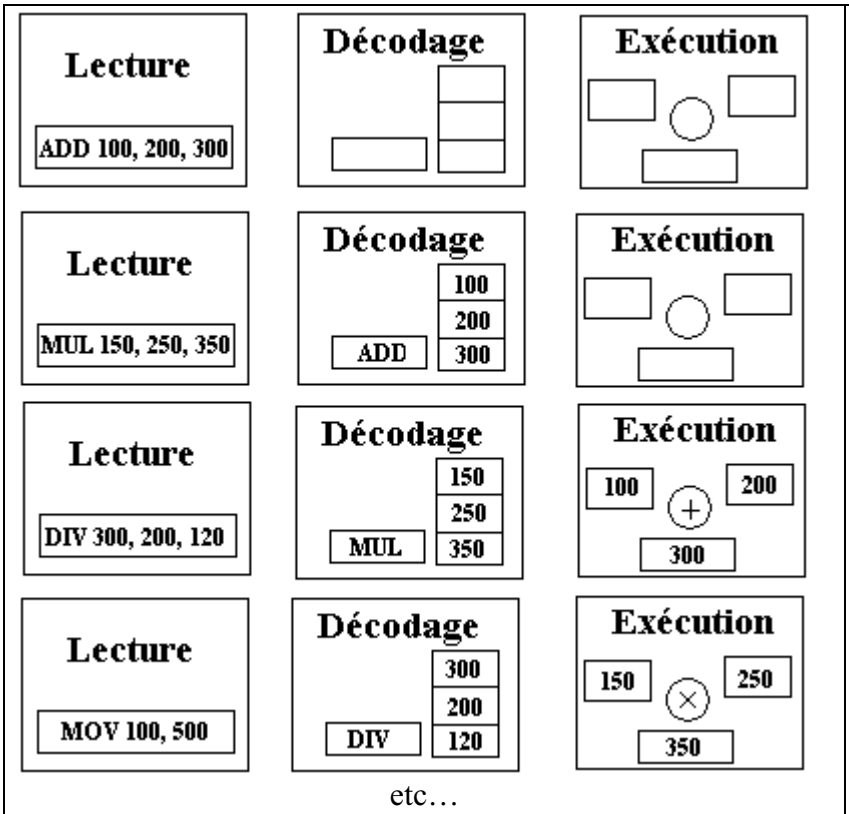
Le cycle recommence identique pour l'instruction MUL 150, 250, 350

L'architecture pipe-line consiste à optimiser les temps d'attente de chaque étage, en commençant le traitement de l'instruction suivante dès que l'étage de lecture a été libéré par l'instruction en cours, et de procéder identiquement pour chaque étage de telle façon que durant chaque phase, tous les étages soient occupés à fonctionner (chacun sur une instruction différente).

A un instant $\mathbf{t_0}$ donné l'étage d'exécution travaille sur les actions à effectuer pour l'instruction de rang **n**, l'étage de décodage travaille sur le décodage de l'instruction de rang **n+1**, et l'étage de lecture sur la lecture de l'instruction de rang **n+2**.

Il est clair que cette technique dénommée **architecture pipe-line** accélère le traitement d'une instruction donnée, puisqu'à la fin de chaque phase une instruction est traitée en entier. Le nombre d'unités différentes constituant le pipe-line s'appelle le **nombre d'étages du pipe-line**.

La figure ci-dessous illustre le démarrage du traitement des 4 instructions selon un pipe-line à 3 étages (lecture, décodage, exécution) :



Période initiale (une seule fois au démarrage)

Chaque étage se met en route

Exécution de l'instruction ADD

Exécution de l'instruction MUL

La prochaine phase verra la fin de l'exécution de l'instruction DIV,…

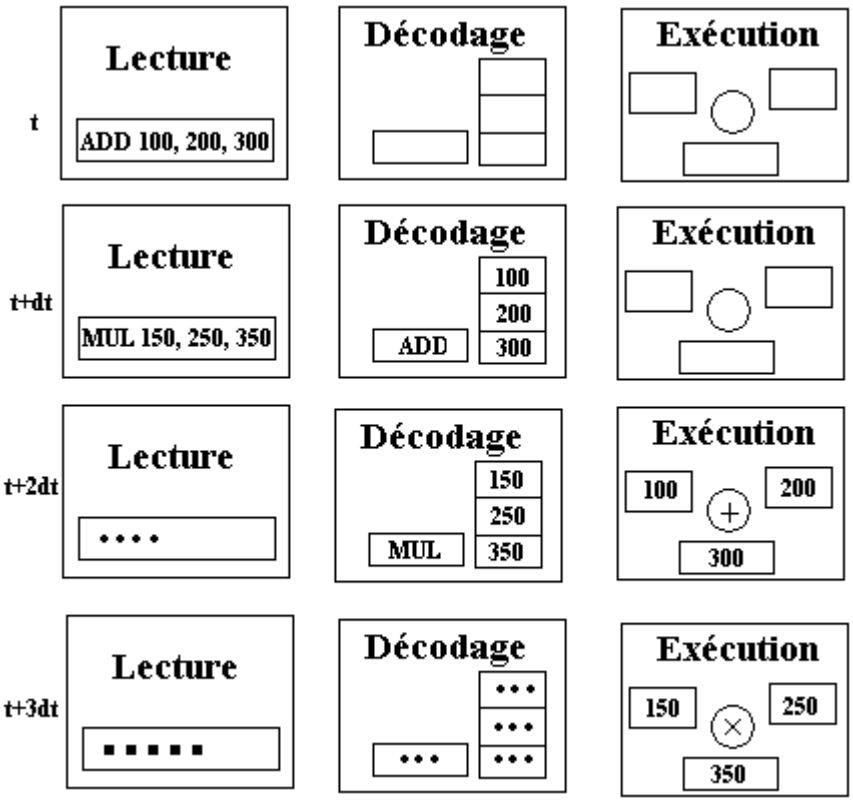
1.10 Architecture super-scalaire

On dit qu'un processeur est super-scalaire lorsqu'il possède plusieurs pipe-lines indépendants dans lesquels plusieurs instructions peuvent être traitées simultanément. Dans ce type d'architecture apparaît la notion de parallélisme avec ses *contraintes de dépendances* (par exemple lorsqu'une instruction nécessite le résultat de la précédente pour s'exécuter, ou encore lorsque deux instructions accèdent à la même ressource mémoire,…).

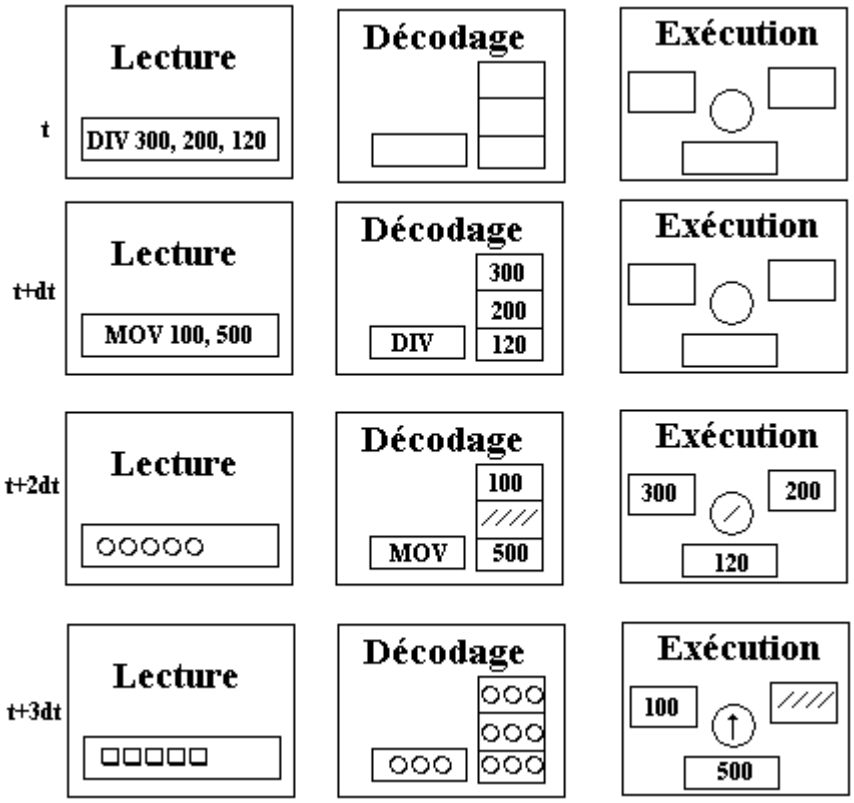
Examinons l'exécution de notre exemple à 4 instructions sur un processeur super-scalaire à 2 pipe-lines. Nous supposons nous trouver dans le cas idéal pour lequel il n'y a aucune dépendance entre deux instructions, nous figurons séparément le schéma temporel d'exécution de chacun des deux pipe-lines aux t, t+dt, t+2dt et t+3dt afin d'observer leur comportement et en sachant que les deux fonctionnent en même temps à un instant quelconque.

Le processeur envoie les deux premières instruction ADD et MUL au pipe-line n°1, et les deux suivantes DIV et MOV au pipe-line n°2 puis les étages des deux pipe-lines se mettent à fonctionner.

PIPE-LINE n°1



PIPE-LINE n°2



nous remarquerons qu'après de t+dt, chaque phase voit s'exécuter 2 instructions :

à t+2dt ce sont ADD et DIV

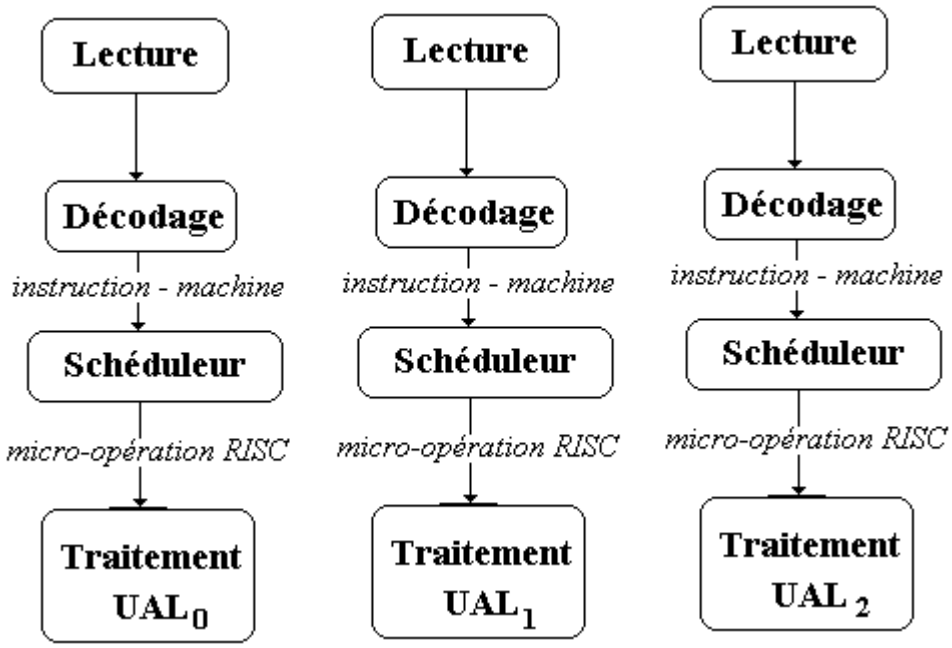
à t+3dt se sont MUL et MOV

Rappelons au lecteur que nous avons supposé par simplification de l'explication que ces 4 instructions sont indépendantes et donc leur ordre d'exécution est indifférent. Ce n'est dans la réalité pas le cas car par exemple si l'instruction DIV 300, 200, 120 utilise le contenu de la mémoire 300 pour le diviser par le contenu de la mémoire 200, et que l'instruction ADD 100, 200, 300 range dans cette mémoire 300 le résultat de l'addition des contenus des mémoires 100 et 200, alors l'exécution de DIV dépend de l'exécution de ADD. Dans cette éventualité à t+2dt, le calcul de DIV par le second pipe-line doit "attendre" que le calcul de ADD soit terminé pour pouvoir s'exécuter sous peine d'obtenir une erreur en laissant le parallélisme fonctionner : un processeur super-scalaire doit être capable de **désactiver le parallélisme** dans une telle condition. Par contre dans notre exemple, à t+3dt le parallélisme des deux pipe-lines reste efficace MUL et MOV sont donc exécutées en même temps.

Le pentium IV de la société Intel intègre un pipe-line à 20 étages et constitue un exemple de processeur combinant un mélange d'architecture RISC et CISC. Il possède en externe un jeu d'instruction complexes (CISC), mais dans son cœur il fonctionne avec des micro-instructions de type RISC traitées par un pipe-line super-scalaire.

L'AMD 64 Opteron qui est un des micro-processeur de l'offre 64 bits du deuxième constructeur mondial de micro-processeur derrière la société Intel, dispose de 3 pipe-lines d'exécution identiques pour les calculs en entiers et de 3 pipe-lines spécialisés pour les calculs en virgules flottante. L'AMD 64 Opteron est aussi un mélange d'architecture RISC-CISC avec un cœur de micro-instructions RISC comme le pentium IV.

Nous figurons ci-dessous les 3 pipe-lines d'exécution (sur les entiers par exemple) :



Chacune des 3 UAL effectue les fonctions classiques d'une UAL, plus des opérations de multiplexage, de drapeau, des fonctions conditionnelles et de résolution de branchement. Les multiplications sont traitées dans une unité à part de type pipe-line et sont dirigées vers les pipe-lines

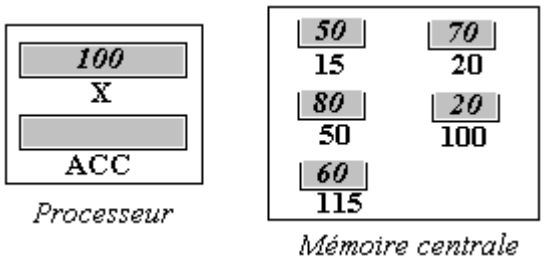
UAL0 et UAL1.

1.11 Principaux modes d'adressage des instructions machines

Nous avons indiqué précédemment qu'une instruction machine contenait des adresses d'opérandes situées en mémoire centrale. En outre, il a été indiqué que les processeurs centraux disposaient de registres internes. Les informaticiens ont mis au point des techniques d'adressages différentes en vue d'accéder à un contenu mémoire. Nous détaillons dans ce paragraphe les principales d'entre ces techniques d'adressage. Afin de conserver un point de vue pratique, nous montrons le fonctionnement de chaque mode d'adressage à l'aide de schémas représentant le cas d'une instruction LOAD de chargement d'un registre nommé ACC d'une machine à une adresse, selon 6 modes d'adressages différents.

Environnement d'exécution d'un LOAD

Soit à considérer un processeur contenant en particulier deux registres X chargé de la valeur entière 100 et ACC (accumulateur de machine à une adresse) et une mémoire centrale dans laquelle nous exhibons 5 mots mémoire d'adresses 15, 20, 50, 100, 115. Chaque mot et contient un entier respectivement dans l'ordre 50, 70, 80, 20, 60, comme figuré ci-dessous :



L'instruction "LOAD Oper" a pour fonction de charger le contenu du registre ACC avec un opérande Oper qui peut prendre 6 formes, chacune de ces formes représente un mode d'adressage particulier que nous définissons maintenant.

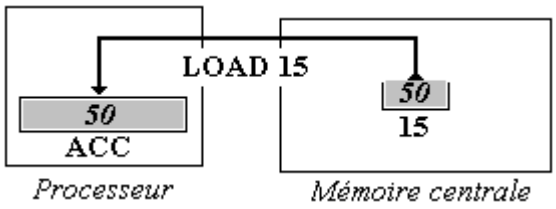
Adressage immédiat

L'opérande Oper est considéré comme une valeur à charger immédiatement (dans le registre ACC ici). Par exemple, nous noterons LOAD #15, pour indiquer un adressage immédiat (c'est à dire un chargement de la valeur 15 dans le registre ACC).

Adressage direct

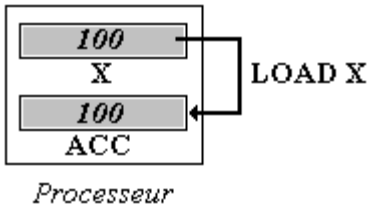
L'opérande Oper est considéré comme une adresse en mémoire centrale. Par exemple, nous noterons LOAD 15, pour indiquer un adressage direct (c'est à dire un chargement du contenu 50 du mot

mémoire d'adresse 15 dans le registre ACC).



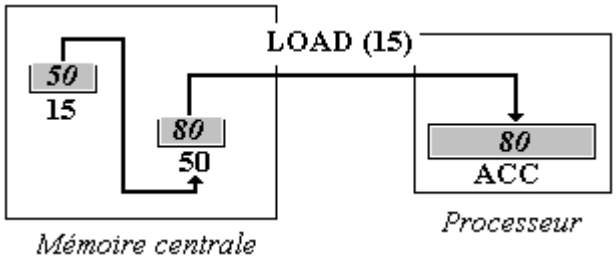
Adressage direct avec registre

L'opérande Oper est un registre interne du processeur (noté X dans l'exemple), un tel mode d'adressage indique de charger dans ACC le contenu du registre Oper. Par exemple, nous noterons LOAD X, pour indiquer un adressage direct avec registre qui charge l'accumulateur ACC avec la valeur 100 contenue dans X.



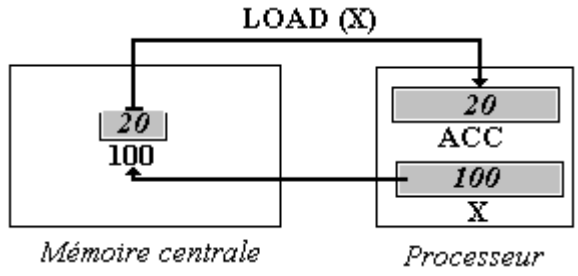
Adressage indirect

L'opérande Oper est considéré comme l'adresse d'un mot_1 en mémoire centrale, mais ce mot_1 contient lui-même l'adresse d'un autre mot_2 dont on doit charger le contenu dans ACC. Par exemple, nous noterons LOAD (15), pour indiquer un adressage indirect (c'est à dire un chargement dans le registre ACC, du contenu 80 du mot_2 mémoire dont l'adresse 50 est contenue dans le mot_1 d'adresse 15).



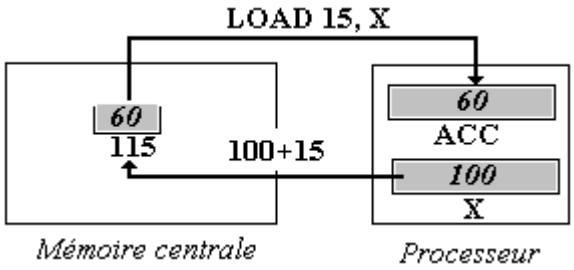
Adressage indirect avec registre

L'opérande Oper est considéré comme un registre dont le contenu est l'adresse du mot dont on doit charger la valeur dans ACC. Par exemple, nous noterons LOAD (X), pour indiquer un adressage indirect avec le registre X (c'est à dire un chargement dans le registre ACC, du contenu 20 du mot mémoire dont l'adresse 100 est contenue dans le registre X).



Adressage indexé

L'opérande Oper est un couple formé par un registre **R** et une adresse **adr**. La connaissance de l'adresse du mot dont on doit charger la valeur est obtenue par addition de l'adresse **adr** au contenu du registre **R**. Par exemple, nous noterons LOAD 15, X , pour indiquer un adressage indexé par le registre X (c'est à dire un chargement dans le registre ACC, du contenu 60 du mot mémoire dont l'adresse 115 est obtenue par addition de 15 et du contenu 100 du registre X).



Quelques remarques sur les différents modes d'adressages (avec l'exemple du LOAD) :

- Le mode direct correspond à des préoccupations de chargement de valeur à des emplacements fixés.
- Les modes indirects permettent à partir d'un emplacement mémoire quelconque d'atteindre un autre emplacement mémoire et donc autorise des traitements sur les adresses elles-mêmes.
- Le mode indexé est très utile lorsque l'on veut atteindre une famille de cellules mémoires contiguës possédant une adresse de base (comme pour un tableau). L'instruction LOAD 15,X permet si l'on fait varier le contenu du registre X de la valeur 0 à la valeur 10 (dans une itération par exemple) d'atteindre les mots d'adresse 15, 16, … , 25.

Les registres sont très présents dans les micro-processeurs du marché

Le processeur AMD 64 bits Optéron travaille avec 16 registres généraux de 64 bits et 16 registres généraux de 128 bits.

Le processeur pentium IV travaille avec 8 registres généraux 32 bits et 8 registres généraux 80 bits.

L'architecture IA 64 d'Intel et HP est fondée sur des instructions machines très longues travaillant avec 128 registres généraux 64 bits et 128 registres généraux 82 bits pour les calculs classiques.

Il en est des processeurs comme il en est des moteurs à explosion dans les voitures, quelle que soit leur sophistication technique (processeur vectoriel, machine parallèle, machine multi-processeur, …)leurs fondements restent établis sur les principes d'une machine de Von Neumann (mémoire, registre, adresse, transfert).

2. Mémoires : mémoire Centrale , mémoire cache

2.1 Mémoire

Mémoire :c'est un organe (électronique de nos jours), capable de contenir, de conserver et de restituer sans les modifier de grandes quantités d'information.

2.2 Les différents types de mémoires

La mémoire vive RAM (Random Access Memory)

- Mémoire dans laquelle on peut lire et écrire.
- Mémoire *volatile* (perd son contenu dès la coupure du courant).

La mémoire morte ROM (Read Only Memory)

- Mémoire dans laquelle on **ne** peut **que** lire.
- Mémoire *permanente* (conserve indéfiniment son contenu).

Les PROM (Programable ROM)

- Ce sont des mémoires vierges programmables une seule fois avec un outil spécialisé s'appelant un programmateur de PROM.
- Une fois programmées elles se comportent dans l'ordinateur comme des ROM.

Les EPROM (Erasable PROM)

- Ce sont des PROM effaçables (généralement sous rayonnement U.V),
- elles sont reprogrammables avec un outil spécialisé,
- elles se comportent comme des ROM en utilisation courante.
- Les EEPROM (Electrical EPROM) sont effaçables par signaux électriques.
- Les FLASH EEPROM sont des EEPROM effaçables par bloc.

2.3 Les unités de capacité

Les unités de mesure de stockage de l'information sont :

Le bit (pas de notation)
L'octet = 2^3 bits = 8 bits. (noté 1 o)
Le Kilo-octet = 2^{10} octets =1024 o (noté 1 Ko)
Le Méga-octet = 2^{20} octets =$(1024)^2$ o (noté 1 Mo)
Le Giga-octet = 2^{30} octets =$(1024)^3$ o (noté 1 Go)
Le Téra-octet = 2^{40} octets =$(1024)^4$ o (noté 1 To)…

Les autres sur-unités sont encore peu employées actuellement.

2.4 Mémoire centrale : définitions

Mot : c'est un regroupement de **n** bits constituant une case mémoire dans la mémoire centrale. Ils sont tous numérotés.

Adresse : c'est le numéro d'un mot-mémoire (case mémoire) dans la mémoire centrale.

Programme : c'est un ensemble d'***instructions*** préalablement codées (en binaire) et enregistrées dans la mémoire centrale sous la forme d'une liste ***séquentielle*** d'instructions. Cette liste représente une suite d'actions élémentaires que l'ordinateur doit accomplir sur des ***données*** en entrée, afin d'atteindre le ***résultat*** recherché.

Organisation : La mémoire centrale est organisée en bits et en mots. Chaque mot-mémoire est repéré bijectivement par son **adresse** en mémoire centrale.

Contenu : La mémoire centrale contient en binaire, deux sortes d'informations

- des ***programmes***,
- des ***données***.

Composition : Il doit être possible de lire et d'écrire dans une mémoire centrale. Elle est donc habituellement composée de mémoires de type **RAM**.

Remarques

- Un ordinateur doté d'un programme est un automatisme apte seulement à répéter le même travail(celui dicté par le programme).
- Si l'on change le programme en mémoire centrale, on obtient un nouvel automatisme.

2.5 Mémoire centrale : caractéristiques

La mémoire centrale peut être réalisée grâce à des technologies différentes. Elle possède toujours des caractéristiques générales qui permettent de comparer ces technologies. En voici quelques unes :

La capacité représente le nombre maximal de mots que la mémoire peut stocker simultanément.
Le temps d'accès est le temps qui s'écoule entre le stockage de l'adresse du mot à sélectionner et l'obtention de la donnée.
Le temps de cycle ou cycle mémoire est égal au temps d'accès éventuellement additionné du temps de rafraîchissement ou de réécriture pour les mémoires qui nécessitent ces opérations.
Le débit d'une mémoire : c'est l'inverse du cycle mémoire en octet par seconde
La volatilité, la permanence.

Terminons ce survol des possibilités d'une mémoire centrale, en indiquant que le mécanisme d'accès à une mémoire centrale par le processeur est essentiellement de type séquentiel et se décrit selon trois phases :

- stockage,
- sélection,
- transfert.

Pour l'instant :

Un ordinateur est une machine séquentielle de Von Neumann dans laquelle s'exécutent ces 3 phases d'une manière immuable, que ce soit pour les programmes ou pour les données et aussi complexe que soit la machine.

La mémoire centrale est un élément d'importance dans l'ordinateur, nous avons vu qu'elle est composée de RAM en particulier de RAM dynamiques nommées DRAM dont on rappelle que sont des mémoires construite avec un transistor et un condensateur. Depuis 2004 les micro-ordinateurs du commerce sont tous équipés de DRAM, le sigle employé sur les notices techniques est DDR qui est l'abréviation du sigle DDR SDRAM dont nous donnons l'explication :

Ne pas confondre SRAM et SDRAM

> Une SRAM est une mémoire statique (SRAM = **S**tatique RAM) construite avec des bascules, une SDRAM est une mémoire dynamique DRAM qui fonctionne à la vitesse du bus mémoire, elle est donc synchrone avec le fonctionnement du processeur le "S" indique la synchronicité (SDRAM = Synchrone DRAM).

Une DDR SDRAM

> C'est une SDRAM à double taux de transfert pouvant expédier et recevoir des données deux fois par cycle d'horloge au lieu d'une seule fois. Le sigle DDR signifie **D**ouble **D**ata **R**ate.

Les performances des mémoires s'améliorent régulièrement, le secteur d'activité est très innovant, le lecteur retiendra que les mémoires les plus rapides sont les plus chères et que pour les comparer en ce domaine, il faut utiliser un indicateur qui se nomme le cycle mémoire.

Temps de cycle d'une mémoire ou cycle mémoire : le processeur attend

Nous venons de voir qu'il représente l'intervalle de temps qui s'écoule entre deux accès consécutif à la mémoire toutes opération cumulées. Un processeur est cadencé par une horloge dont la fréquence est donnée actuellement en MHz (Méga Hertz). Un processeur fonctionne beaucoup plus rapidement que le temps de cycle d'une mémoire, par exemple prenons un micro-processeur cadencé à 5 MHz auquel est connectée une mémoire SDRAM de temps de cycle de 5 ns (ordre de grandeur de matériels récents). Dans ces conditions le processeur peut accéder aux données selon un cycle qui lui est propre 1/5MHz soit un temps de 2.10^{-1} ns, la mémoire SDRAM ayant un temps de cycle de 5 ns, le processeur doit **attendre** 5ns / 2.10^{-1} ns = **25 cycles propres** entre deux accès aux données de la mémoire. Ce petit calcul montre au lecteur l'intérêt de l'innovation en rapidité pour les mémoires.

C'est aussi pourquoi on essaie de ne connecter directement au processeur que des mémoires qui fonctionnent à une fréquence proche de celle du processeur.

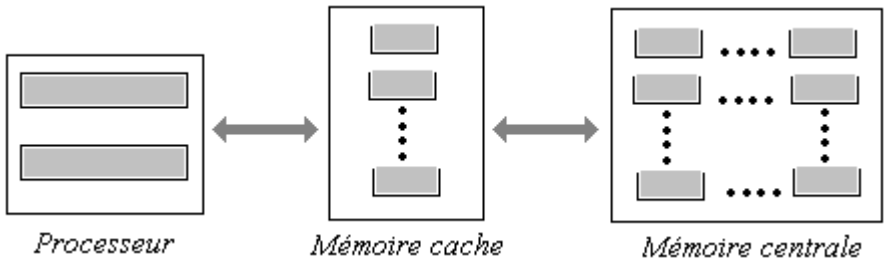
Les registres d'un processeur sont ses mémoires les plus rapides

Un processeur central est équipé de nombreux registres servant à différentes fonctions, ce sont en général des mémoires qui travaillent à une fréquence proche de celle du processeur , actuellement leur architecture ne leur permet pas de stocker de grandes quantités d'informations. Nous avons vu au chapitre consacré aux circuits logiques les principaux types de registres (registres parallèles, registres à décalages, registres de comptage, …)

Nous avons remarqué en outre que la mémoire centrale qui stocke de très grandes quantités d'informations (relativement aux registres) fonctionne à une vitesse plus lente que celle du processeur. Nous retrouvons alors la situation classique d'équilibre entre le débit de robinets qui

remplissent ou vident un réservoir. En informatique, il a été prévu de mettre entre le processeur et la mémoire centrale une sorte de réservoir de mémoire intermédiaire nommée la **mémoire cache**.

2.6 Mémoire cache



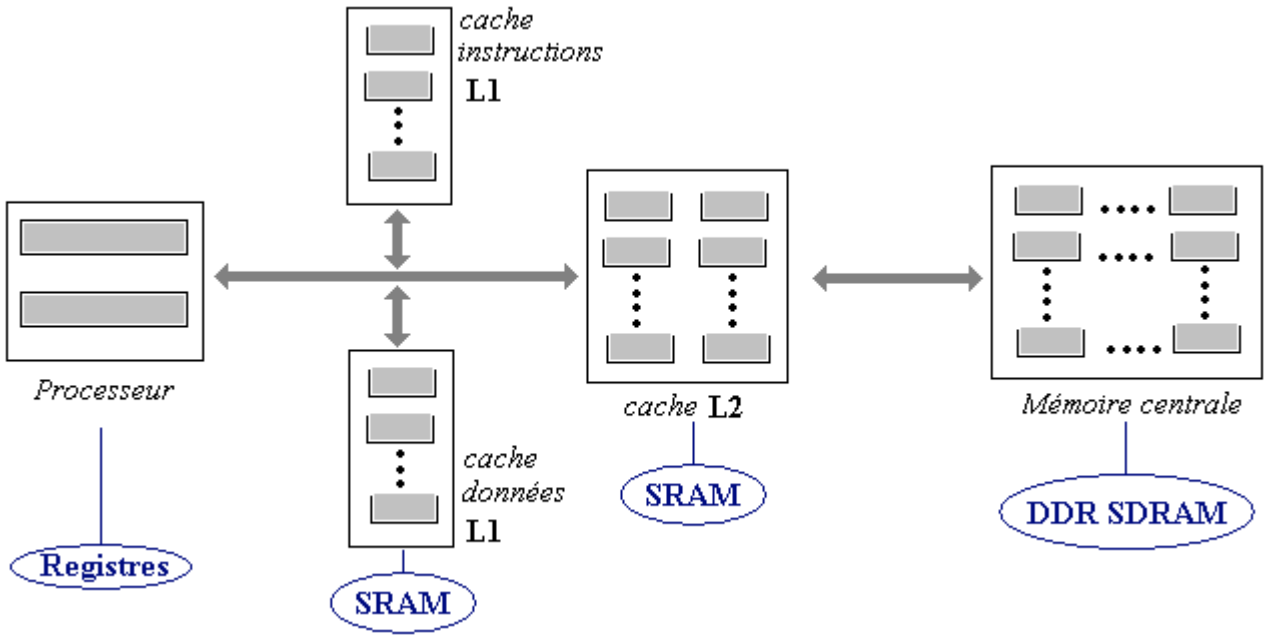
La mémoire cache (on dit aussi le cache) est une variété de mémoire plus rapide que la mémoire centrale (un peu moins rapide que les registres). La particularité technique actuelle de la mémoire cache est que plus sa taille est grande plus son débit a tendance à ralentir.

La caractéristique fonctionnelle du cache est de servir à stocker des instructions et des données provenant de la mémoire centrale et qui ont déjà été utilisées les plus récemment par le processeur central.

Actuellement le cache des micro-processeurs récents du marché est composé de deux niveaux de mémoires de type SRAM la plus rapide (type de mémoire RAM statique semblable à celle des registres) : le cache de niveau un est noté L1, le cache de niveau deux est noté L2.

Le principe est le suivant :

Le cache L1 est formé de deux blocs séparés, l'un servant au stockage des données, l'autre servant au stockage des instructions.



Si un étage du processeur cherche une donnée, elle va être d'abord recherchée dans le cache de donnée L1 et rapatriée dans un registre adéquat, si la donnée n'est pas présente dans le cache L1, elle sera recherchée dans le cache L2.

Si la donnée est présente dans L2, elle est alors **rapatriée** dans un registre adéquat et **recopiée** dans le bloc de donnée du cache L1. Il en va de même lorsque la donnée n'est pas présente dans le cache L2, elle est alors **rapatriée** depuis la mémoire centrale dans le registre adéquat et **recopiée** dans le cache L2.

Généralement la mémoire cache de niveau L1 et celle de niveau L2 sont regroupées dans la même puce que le processeur (**cache interne**).

Nous figurons ci-dessous le facteur d'échelle relatif entre les différents composants mémoires du processeur et de la mémoire centrale (il s'agit d'un coefficient de multiplication des temps d'accès à une information selon la nature de la mémoire qui la contient). Les registres, mémoires les plus rapides se voient affecter la valeur de référence 1 :

L'accès par le processeur à une information située dans la DDR SDRAM de la mémoire centrale est 100 fois plus lente qu'un accès à une information contenue dans un registre.

> Par exemple, le processeur AMD 64 bits Optéron travaille avec un cache interne L1 de 64 Ko constitué de mémoires associatives (type ECC pour le bloc L1 de données et type parité pour le bloc L1 d'instructions), le cache L2 de l'Optéron a une taille de 1 Mo constitué de mémoires 64 bits associatives de type ECC, enfin le contrôleur de mémoire accepte de la DDR SDRAM 128 bits jusqu'à 200 Mhz en qualité ECC.

Définition de mémoire ECC (mémoire à code correcteur d'erreur)

Une mémoire ECC est une mémoire contenant des bits supplémentaires servant à détecter et à corriger une éventuelle erreur ou altération de l'information qu'elle contient (par exemple lors d'un transfert).

La technique la plus simple est celle du bit de parité (Parity check code), selon cette technique l'information est codée sur n bits et la mémoire contient un n+1 ème bit qui indique si le nombre de bits codant l'information contenue dans les n bits est pair (bit=0)ou impair(bit=1). C'est un code détecteur d'erreur.

Exemple d'une mémoire à 4 bits plus bit de parité (le bit de poids faible contient la parité) :

> Information 10010 → bit de parité = 0 , car il y a deux bits égaux à 1 (nombre pair)
> Information 11110 → bit de parité = 0 , car il y a quatre bits égaux à 1 (nombre pair)
> Information 11011 → bit de parité = 1 , car il y a trois bits égaux à 1 (nombre impair)

Une altération de deux bits (ou d'un nombre pair de bits) ne modifiant pas la parité du décompte ne sera donc pas décelée par ce code :

Supposons que l'information 10010 soit altérée en 01100 (le bit de parité ne change pas car le nombre de 1 de l'information altérée est toujours pair, il y en a toujours 2 !). Ce code est simple peu

coûteux, il est en fait un cas particulier simple de codage linéaire systématique inventés par les spécialistes du domaine.

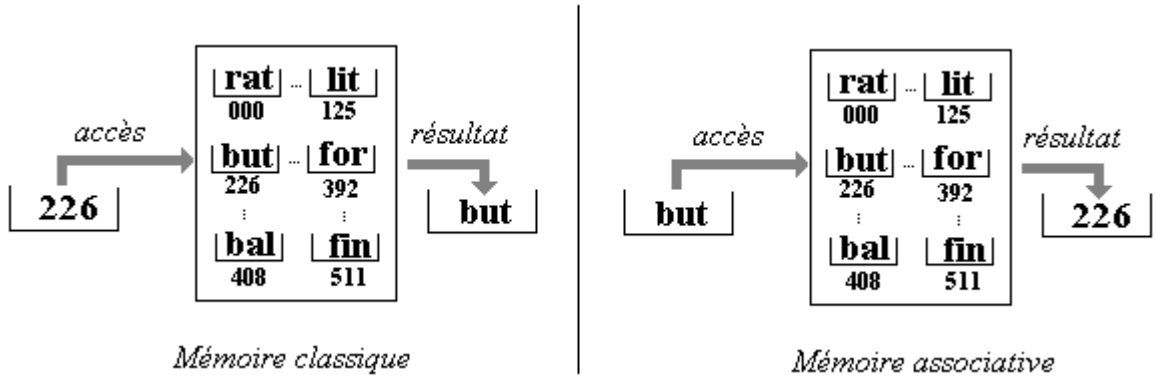
Mémoire ECC générale

Les mathématiciens mis à contribution à travers la théorie des groupes et des espaces vectoriels fournissent des modèles de codes détecteur et correcteur d'erreurs appelés codes linéaire cycliques, les codes de Hamming sont les plus utilisés. Pour un tel code permettant de corriger d'éventuelles erreur de transmission, il faut ajouter aux **n** bits de l'information utile, un certain nombre de bits supplémentaires représentant un polynôme servant à corriger les **n** bits utiles.

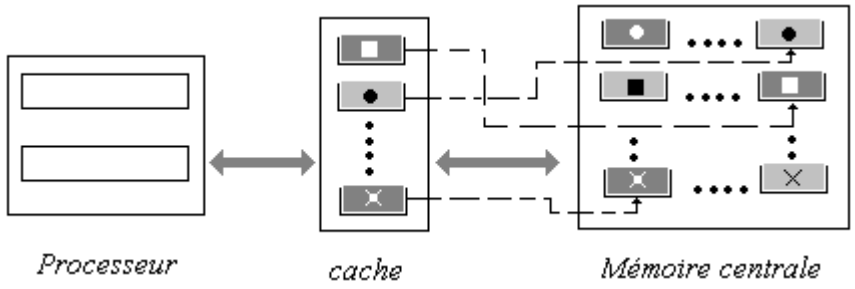
Pour une mémoire ECC de 64 bits utiles, 7 supplémentaires sont nécessaires pour le polynôme de correction, pour une mémoire de 128 bits utiles, 8 bits sont nécessaires. Vous remarquez que l'Optéron d'AMD utilise de la mémoire ECC pour le cache L1 de données et de la mémoire à parité pour le cache instruction. En effet, si un code d'instruction est altéré, l'étage de décodage du processeur fera la vérification en bloquant l'instruction inexistante, la protection apportée par la parité est suffisante; en revanche si c'est une donnée qui est altérée dans le cache L1 de données, le polynôme de correction aidera alors à restaurer l'information initiale.

Mémoire associative

C'est un genre de mémoire construit de telle façon que la recherche d'une information s'effectue non pas à travers une adresse de cellule, la mémoire renvoyant alors le contenu de la cellule, mais plutôt en donnant un "**contenu**" à rechercher dans la mémoire et celle-ci renvoie **l'adresse** de la cellule.

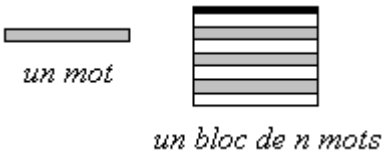


Une mémoire cache est une mémoire associative, ainsi elle permet d'adresser directement dans la mémoire centrale qui n'est pas associative.



On peut considérer une mémoire cache comme une sorte de table de recherche contenant des morceaux de la mémoire centrale. La mémoire centrale est divisée en blocs de **n** mots, et la mémoire cache contient quelques un de ces blocs qui ont été chargés précédemment.

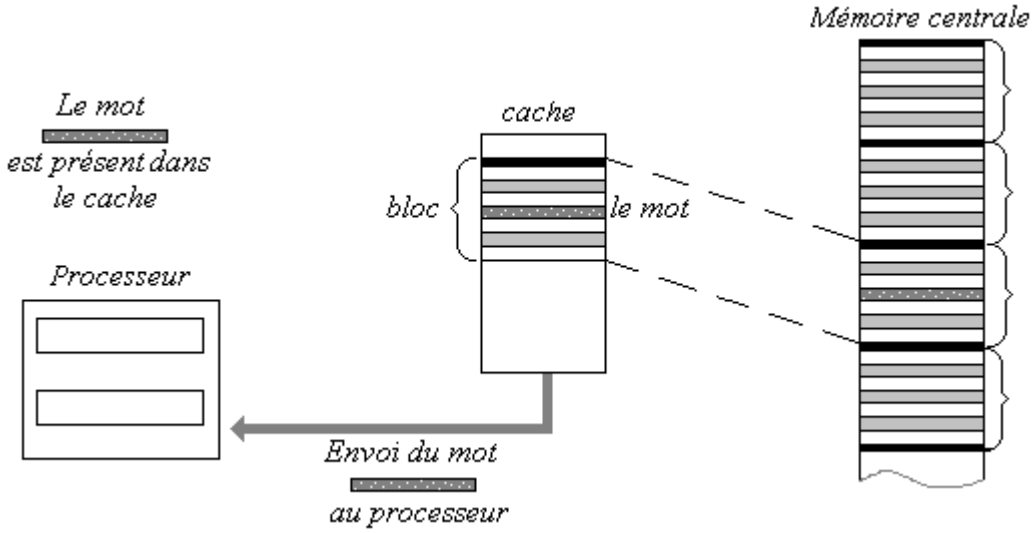
Notation graphiques utilisées :



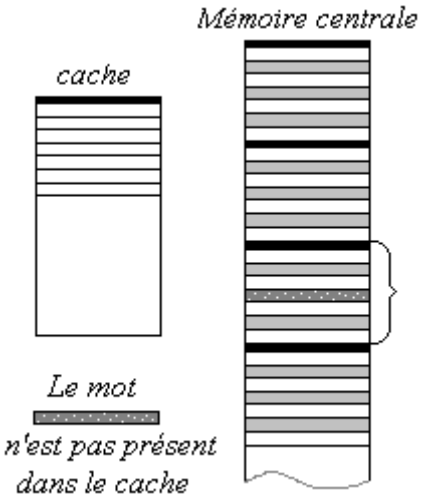
Mécanisme synthétique de lecture-écriture avec cache :

Le processeur fournit l'adresse d'un mot à lire :

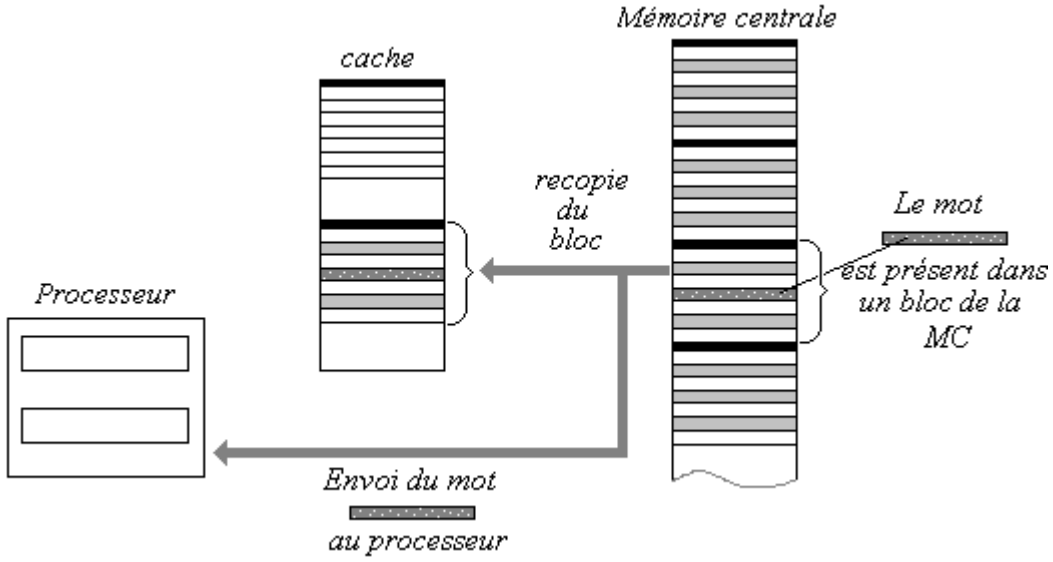
1°) Si ce mot est présent dans le cache, il se trouve dans un bloc déjà copié à partir de son original dans le MC (mémoire centrale), il est alors envoyé au processeur :



2°) Si ce mot n'est pas présent dans le cache, l'adresse porte alors sur un mot situé dans un bloc présent dans la MC (mémoire centrale).

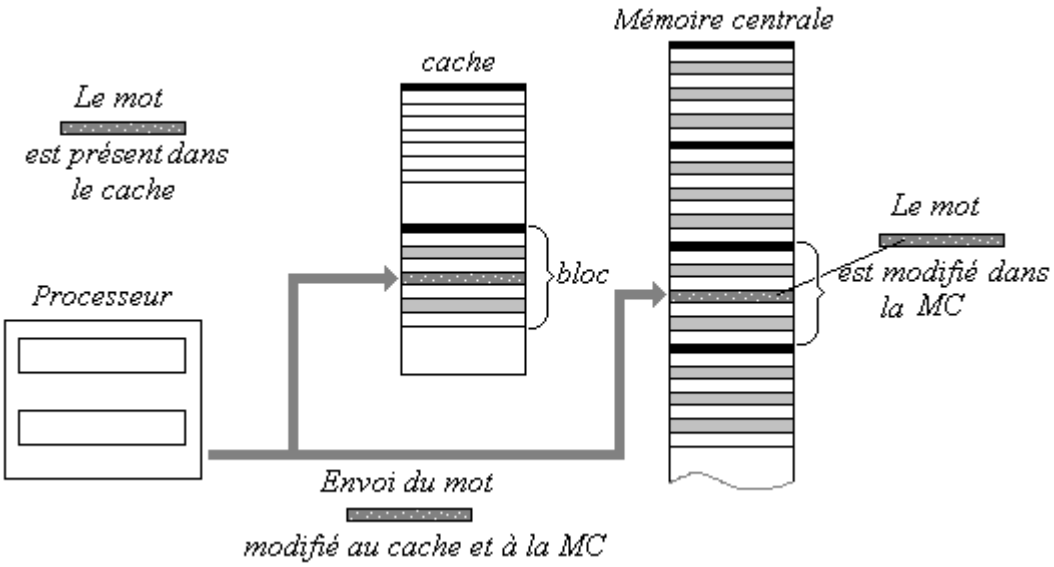


Dans cette éventualité le bloc de la MC dans lequel se trouve le mot, se trouve recopié dans le cache et en même temps le mot est envoyé au processeur :



Pour l'écriture l'opération est semblable, selon que le mot est déjà dans le cache ou non.

Lorsque le mot est présent dans le cache et qu'il est modifié par une écriture il est modifié dans le bloc du cache et modifié aussi dans la MC :



Ce fonctionnement montre qu'il est donc nécessaire que la mémoire cache soit liée par une correspondance entre un mot situé dans elle-même et sa place dans la MC. Le fait que la mémoire cache soit constituée de mémoires associatives, permet à la mémoire cache lorsqu'un mot est sélectionné de fournir l'adresse MC de ce mot et donc de pouvoir le modifier.

3. Une petite machine pédagogique 8 bits

(assistant du package pédagogique présent sur le CD-ROM)

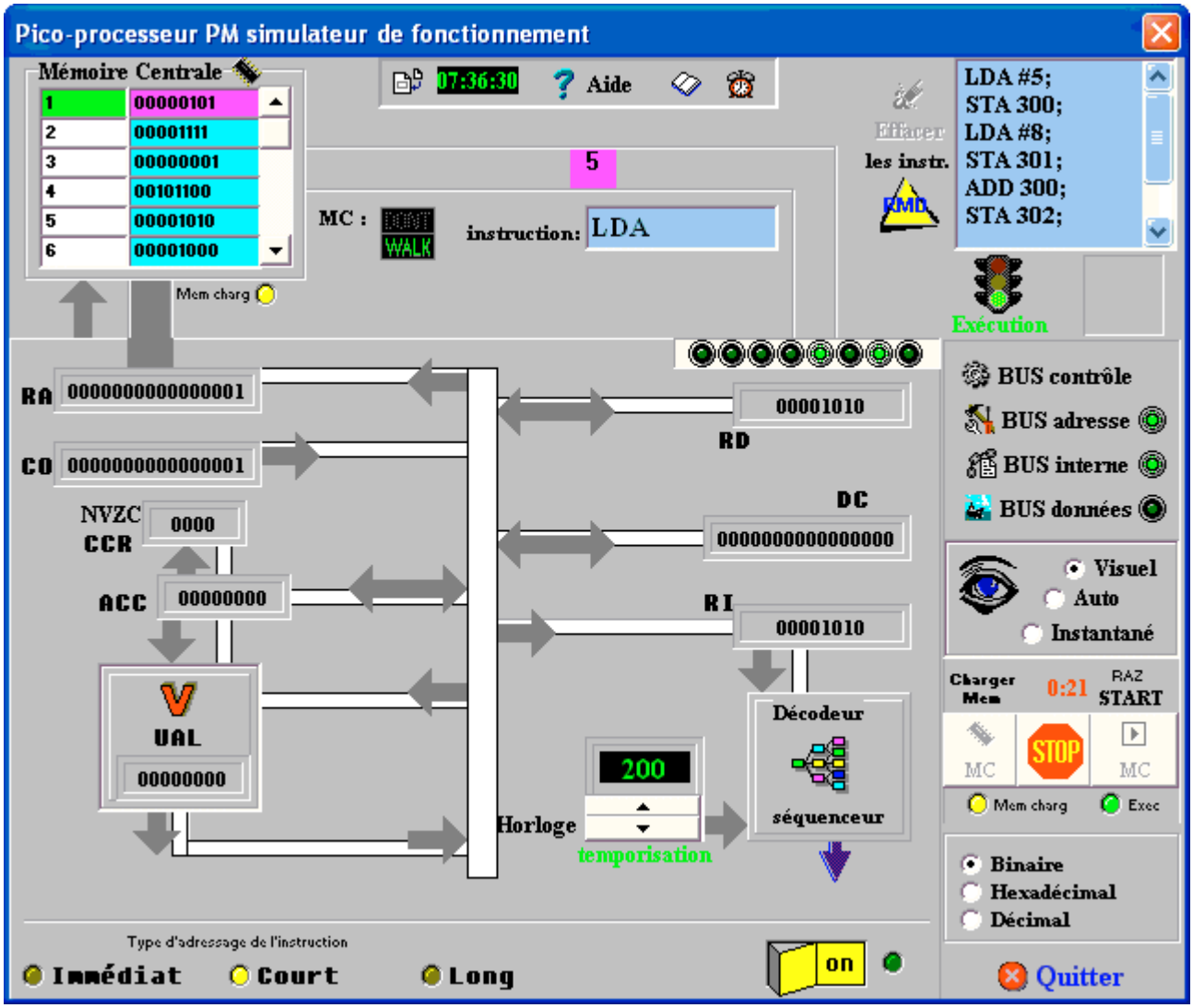
3.1 Unité centrale de PM (pico-machine)

Objectif:

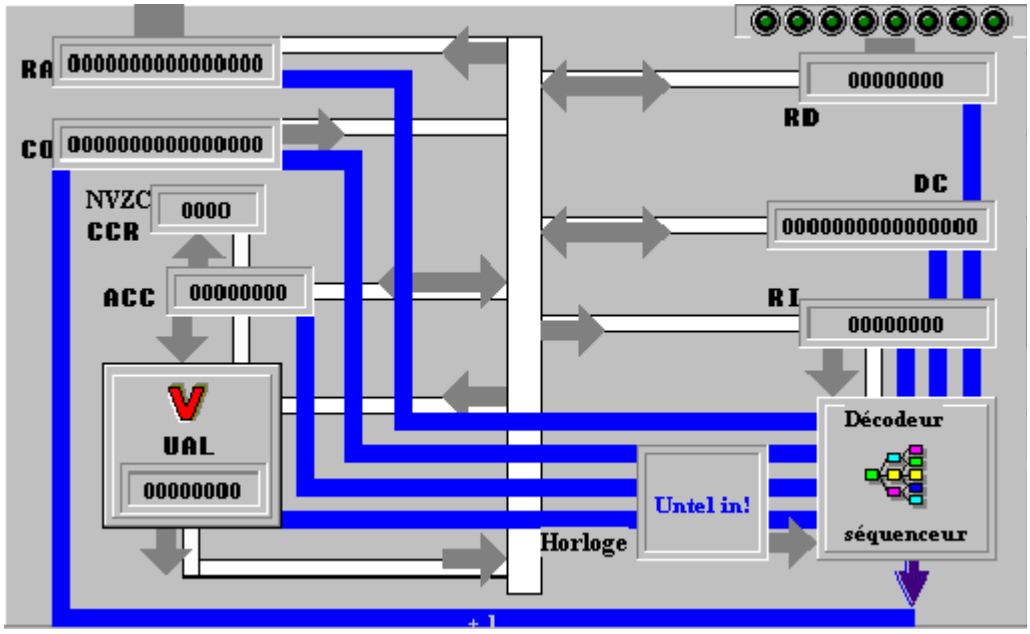
Support pédagogique interactif destiné à faire comprendre l'analyse et le cheminement des informations dans un processeur central d'ordinateur fictif avec accumulateur.

- La mémoire centrale est à mots de 8 bits, les adresses sont sur 16 bits.
- le processeur est doté d'instructions immédiates ou relatives.
- Les instructions sont de 3 types à 1 octet (immédiat), 2 octets (court) ou 3 octets (long).
- Les instructions sont à adressages immédiat et adressage direct.

Interface utilisateur de l'assistant :



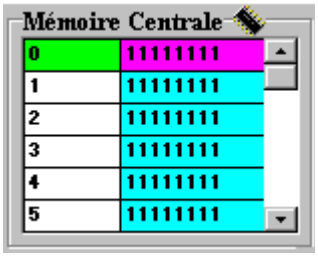
Description générale de l'unité centrale de PM simulée sur le tableau de bord ci-dessous :



RA = Registre Adresse sur 16 bits
CO = Compteur Ordinal sur 16 bits
DC = Registre de formation d'adresse sur 16 bits
RD = Registre de Données sur 8 bits
UAL = Unité Arithmétique et Logique effectuant les calculs sur 8 bits avec possibilité de débordement.
Acc = Accumulateur sur 8 bits (machine à une adresse).
RI = Registre Instruction sur 8 bits (instruction en cours d'exécution).
Décodeur de fonction.
séquenceur
Horloge
CCR = un Registre de 4 Codes Condition **N, V, Z, C,**
BUS de contrôle (bi-directionnel)
BUS interne (circulation des informations internes).

3.2 Mémoire centrale de PM

La mémoire centrale de PM est de 512 octets, ce qui permet dans une machine 8 bits de voir comment est construite la technique d'adressage court (8 bits) et d'adressage long (16 bits).



|***adresse***|***contenu***|

Elle est connectée à l'unité centrale à travers deux bus : un bus d'adresse et un bus de données.

3.3 Jeu d'instructions de PM

PM est doté du jeu d'instructions suivant :

L'adressage **immédiat** d'une instruction INSTR est noté : INSTR #<valeur>
L'adressage **direct** d'une instruction INSTR est noté : INSTR <valeur>

addition avec l'accumulateur
ADD #<valeur> 2 octets code=16
ADD <adr 16 bits> 3 octets code=18
ADD <adr 8 bits> 2 octets code=17

chargement de l'accumulateur
LDA #<valeur> 2 octets code=10
LDA <adr 16 bits> 3 octets code=12
LDA <adr 8 bits> 2 octets code=11

rangement de l'accumulateur
STA <adr 16 bits> 3 octets code=15
STA <adr 8 bits> 2 octets code=14

positionnement indicateurs CNVZ
STC (C=1) 1 octet code=100 **STN (N=1) 1 octet code=101** **STV (V=1) 1 octet code=102** **STZ (Z=1) 1 octet code=103** **CLC (C=0) 1 octet code=104** **CLN (N=0) 1 octet code=105** **CLV (V=0) 1 octet code=106** **CLZ (Z=0) 1 octet code=107**

branchement relatif sur indicateur
BCZ (brancht.si C=0) 2 octets code=22 **BNZ (brancht.si N=0) 2 octets code=23** **BVZ (brancht.si V=0) 2 octets code=24** **BZZ (brancht.si Z=0) 2 octets code=25** **END (fin programme) 1 octet code=255**

Dans le CCR les 4 bits indicateurs sont dans cet ordre : N V Z C.
Ils peuvent être :

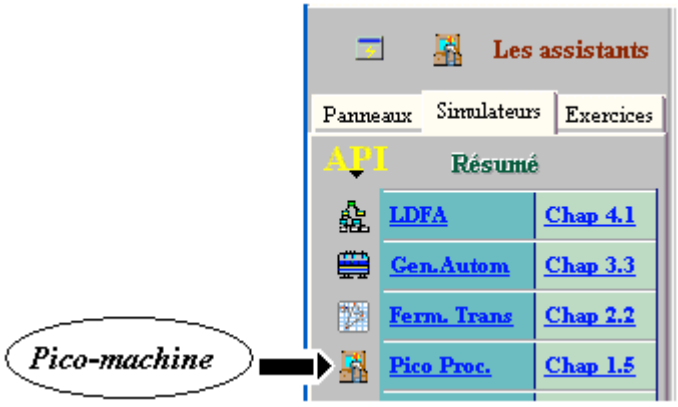
- soit positionnés automatiquement par la machine:

N = le bit de poids fort de l'Accumulateur
V = 1 si overflow (dépassement capacité) 0 sinon
Z = 1 si Accumulateur vaut 0
Z = 0 si Accumulateur <0
C = 1 si retenue (dans l'addition) sinon 0

- soit positionnés par programme.

Exemple de programme en PM
LDA #18 ; *{chargement de l'accumulateur avec la valeur 18}* **STA 50 ;** *{rangement de l'accumulateur dans la mémoire n° 50}* **LDA #5 ;** *{chargement de l'accumulateur avec la valeur 5}* **STA 51 ;** *{rangement de l'accumulateur dans la mémoire n°51}* **ADD 50 ;** *{addition de l'accumulateur avec la mémoire n°50}* **STA 52 ;** *{rangement de l'accumulateur dans la mémoire n°52}* **END**

Le lecteur est encouragé à utiliser le logiciel d'assistance Pico-machine du package pédagogique qui se trouve accessible à travers l'onglet simulateur et met en œuvre :



4. Mémoire de masse (externe ou auxiliaire)

Les données peuvent être stockées à des fin de conservation ,ailleurs que dans la mémoire centrale volatile par construction avec les constituants électroniques actuels. Des périphériques spécialisés sont utilisés pour ce genre de stockage longue conservation, en outre ces mêmes périphériques peuvent stocker une quantité d'information très grande par rapport à la capacité de stockage de la mémoire centrale.

On dénomme dispositifs de **stockage de masse**, de tels périphériques.

Les mémoires associées à ces dispositifs se dénomment mémoires de masse, mémoires externes ou encore mémoires auxiliaires, par abus de langage la mémoire désigne souvent le dispositif de stockage.

Les principaux représentant de cette famille de mémoires sont :

- Les bandes magnétiques (utilisés dans de très faible cas)
- Les disques magnétiques : les disquettes (en voie d'abandon), les disques durs (les plus utilisés).
- Les CD (très utilisés mais bientôt supplantés par les DVD)
- Les DVD

Des technologies ont vu le jour puis se sont éteintes (tambour magnétique, cartes magnétiques, mémoires à bulles magnétiques,…)

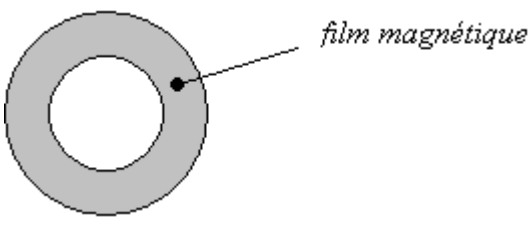
A part les bandes magnétiques qui sont un support ancien encore utilisé à fonctionnement séquentiel, les autres supports (disques, CD, DVD) sont des mémoires qui fonctionnent à accès direct.

4.1 Disques magnétiques - disques durs

Nous décrivons l'architecture générale des disques magnétiques encore appelés disques durs (terminologie américaine hard disk, par opposition aux disquettes nommées floppy disk) très largement employés dans tous les types d'ordinateur comme mémoire auxiliaire.

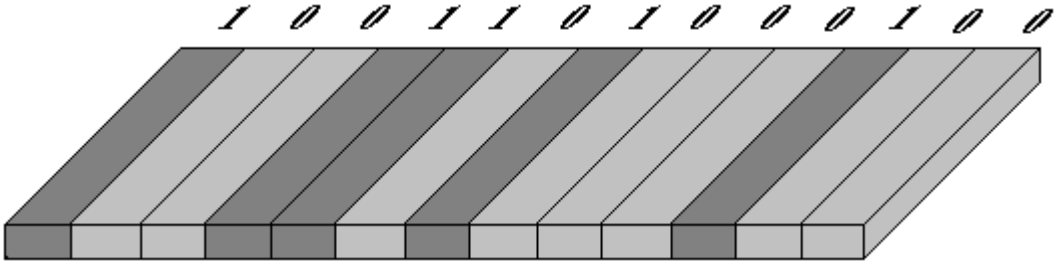
Un micro-ordinateur du commerce dispose systématiquement d'un ou plusieurs disques durs et au minimum d'un lecteur-graveur combiné de CD-DVD permettant ainsi l'accès aux informations extérieures distribuées sur les supports à faibles coût comme les CD et les DVD qui les remplacent progressivement.

Un disque dur est composé d'un disque métallique sur lequel est déposé un film magnétisable, sur une seule face ou sur ses deux faces :



Ce film magnétique est composé de grains d'oxyde magnétisable et c'est le fait que certaines zones du

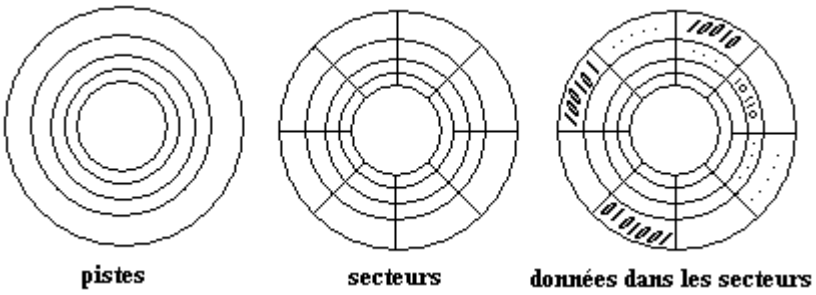
film conservent ou non un champ magnétique, qui représente la présence d'un bit à 0 ou bien à 1.



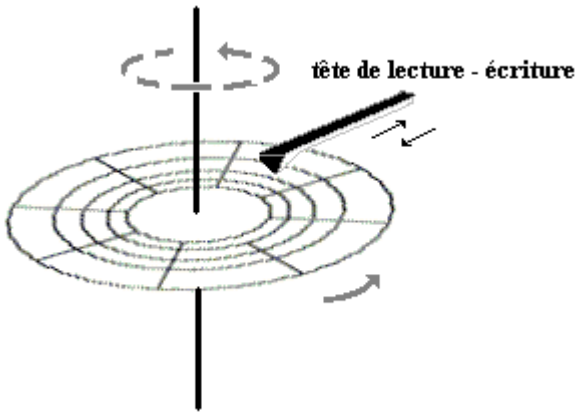
Coupe d'une tranche de disque et figuration de zones magnétisées interprétées comme un bit

Organisation générale d'un disque dur

Un disque dur est au minimum composé de pistes numérotées et de secteurs numérotés, les données sont stockées dans les secteurs.

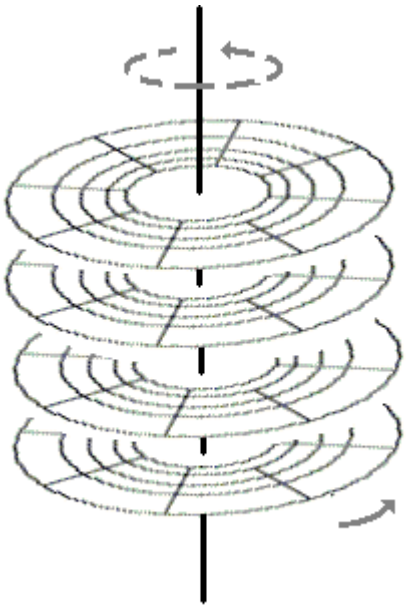


Le disque tourne sur son axe à vitesse d'environ 7200 tr/mn et un secteur donné peut être atteint par un **dispositif mobile appelé tête de lecture-écriture**, soit en lecture (analyse des zones magnétiques du secteur) ou en écriture (modification du champ des zones magnétiques du secteur). Opération semblable à celle qui se passe dans un magnétoscope avec une bande magnétique qui passe devant la tête de lecture. Dans un magnétoscope à une tête, seule la bande magnétisée défile, la tête reste immobile, dans un disque dur le disque tourne sur son axe de symétrie et la tête est animée d'un mouvement de translation permettant d'atteindre n'importe qu'elle piste du disque.

	La tête "flotte" sur un coussin d'air engendré par la rotation très rapide du disque, ce qui la maintient à une hauteur constante de la surface du disque adéquate pour l'enregistrement du champ magnétique du film.

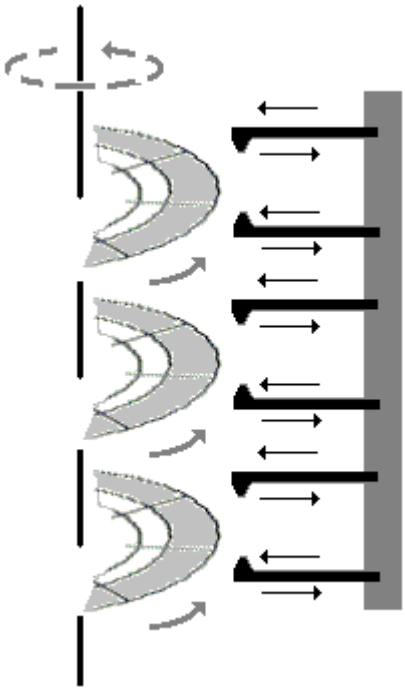
Afin d'augmenter la capacité d'un "disque dur" on empile plusieurs disques physique sur le même axe et on le muni d'un dispositif à plusieurs têtes de lecture-écriture permettant d'accéder à toutes les faces et toutes les pistes de tous les disques physiques. La **pile de disques** construite est encore

appelée un disque dur.



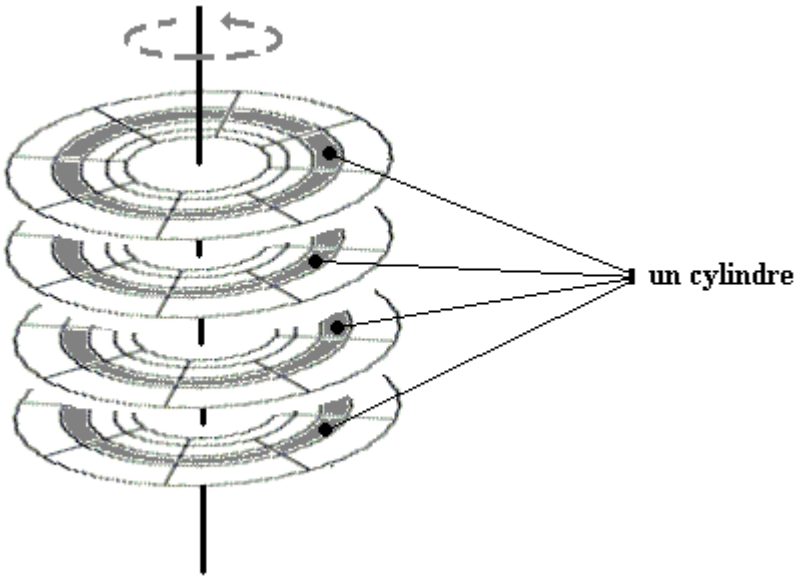
Pile de disques

têtes



disques et têtes en action

Dans une pile de disques on ajoute la notion de cylindre qui repère toutes les pistes portant le même numéro sur chaque face de chacun des disques de la pile.



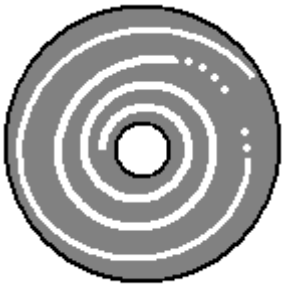
Formatage

Avant toute utilisation ou bien de temps à autre pour tout effacer, les disques durs doivent être "formatés", opération qui consiste à créer des pistes magnétiques et des secteurs vierges (tous les bits à 0 par exemple). Depuis 2005 les micro-ordinateurs sont livrés avec des disques durs dont la

capacité de stockage dépasse les 200 Go, ces disques sont pourvu d'un système de mémoire cache (semblable à celui décrit pour la cache du processeur central) afin d'accélérer les transferts de données. Le temps d'accès à une information sur un disque dur est de l'ordre de la milliseconde.

4.2 Disques optique compact ou CD (compact disk)

Untel disque peut être en lecture seule (dans ce cas on parle de CD-ROM) ou bien en lecture et écriture (dans ce cas on parle de CD réinscriptible). Il est organisé à peu près comme un disque magnétique, avec une différence notable : il n'a qu'une seule piste qui se déroule sous la forme d'une spirale.

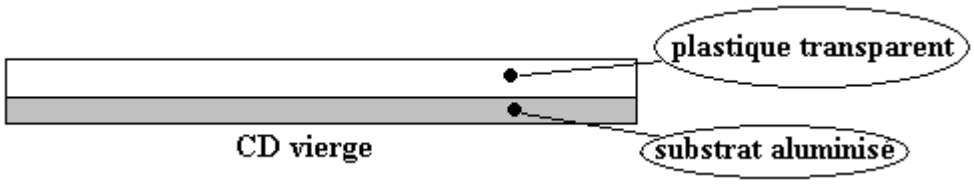


une piste en spirale

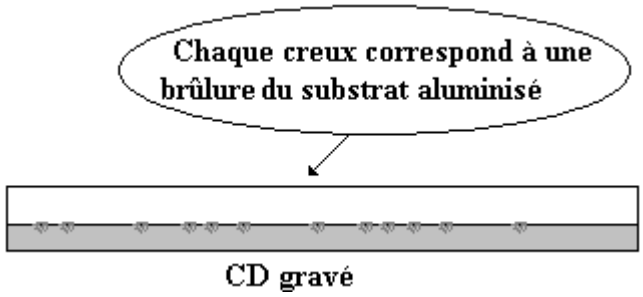
Si sur un disque magnétique les bits codant l'information sont représentés par des grains magnétisables, dans un CD ce sont des creux provoqués par brûlure d'un substrat aluminisé réfléchissant qui représentent les bits d'information.

Gravure d'un CD-ROM

Comme pour un disque dur, le formatage appelé gravure du CD crée les secteurs et les données en même temps. Plus précisément c'est l'absence ou la présence de brûlures qui représente un bit à 0 ou à 1, le substrat aluminisé est protégé par une couche de plastique transparent.

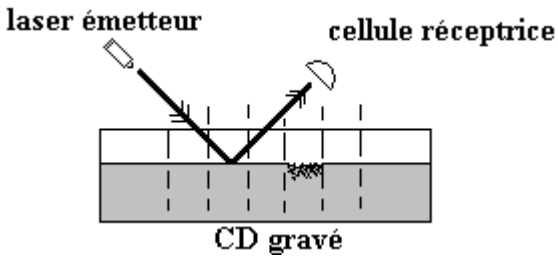


Après gravure avec le graveur de CD, les bits sont matérialisés :



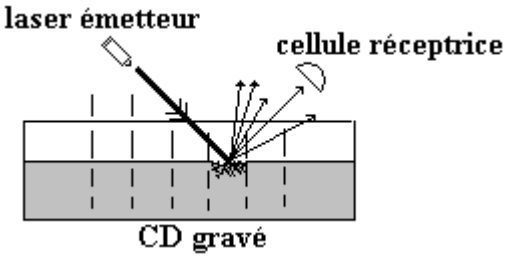
Principe de lecture d'un CD :

Lorsque le substrat est lisse (non brûlé) à un endroit matérialisant un bit, le rayon lumineux de la diode laser du lecteur est réfléchi au maximum de son intensité vers la cellule de réception.



On dira par exemple que le bit examiné vaut 0 lorsque l'intensité du signal réfléchi est maximale.

Lorsque le substrat est brûlé à un endroit matérialisant un bit, la partie brûlée est irrégulière et le rayon lumineux de la diode laser du lecteur est mal réfléchi vers le capteur (une partie du rayonnement est réfléchi par les aspérités de la brûlure dans plusieurs directions). Dans cette éventualité l'intensité du signal capté par réflexion est moindre.



On dira par exemple que le bit examiné vaut 1 lorsque l'intensité du signal réfléchi n'est pas maximale.

La vitesse du disque est variable contrairement à un disque dur qui tourne à vitesse angulaire fixe. En effet la lecture de la piste en spirale nécessite une augmentation au fur et à mesure de l'éloignement du centre. Le temps d'accès à une information sur un CD 54x est de l'ordre de 77 millisecondes.

Le temps d'accès sur un CD ou un DVD est 10 fois plus lent que celui d'un disque dur et environ 100 fois moins volumineux qu'un disque dur.

Toutefois leur coût très faible et leur facilité de transport font que ces supports sont très utilisé de nos jours et remplacent la disquette moins rapide et de moindre capacité.

Nous pouvons reprendre l'échelle comparative des temps d'accès des différents types de mémoires en y ajoutant les mémoires de masse et en indiquant en dessous l'ordre de grandeur de leur capacité :