RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPŔIEUR ET DE LA RECHERCHE SCIENTIQUE
UNIVERSITÉ DJILALI LIABÈS DE SIDI BEL-ABBÈS

FACULTÉ DES SCIENCES EXACTES
DÉPARTEMENT D'INFORMATIQUE

Cours Structure Machine

Mi Informatique Année 2015 - 2016

Dr. Mohammed Fethi KHALFI

Première Version, Mars 2016

Préface

La structure ou l'architecture de l'ordinateur est le domaine qui s'intéresse aux différents composants internes des machines. Ce cours se veut une présentation générale de l'architecture de l'ordinateur et de ses éléments en explicitant leur construction et conception. Il a été écrit dans le but d'enseigner une matière qui évolue très vite même trop vite, fort nécessaire à la compréhension du fonctionnement de l'ordinateur. Cet ouvrage est inspiré d'un enseignement délivré depuis quelques années à des étudiants en mathématique et informatique. Il a pour ambition de faire comprendre les mécanismes internes de l'ordinateur et donner une vision d'ensemble sur la structure materielle des ordinateurs pour les personnes desireuses de poursuivre leurs études en Informatique.

Il est évident que l'on ne peut prétendre savoir programmer une machine ou demander à une machine d'exécuter une fonction, si on ne sait pas ce dont elle est constituée et comment elle fonctionne. Il est vrai que le contenu de ce cours est vaste, néanmoins il ne présente pas de difficultés majeures. Vous trouverez dans chaque chapitre des exemples permettent d'appliquer directement les notions présentées.

Avant-propos

Dans un passé pas lointain, il n'y avait pas d'ordinateurs ; aujourd'hui, il y en a plusieurs millions, voir des milliards. Si les ordinateurs ont tout transformé, c'est parce qu'ils permettent de traiter des informations de manières très diverses et efficaces. C'est en effet le même dispositif qui permet d'utiliser des logiciels de conception assistée par ordinateur (CAO), des logiciels de modélisation et de simulation, des bases de données, des logiciels de courrier électronique et de messagerie instantanée, des logiciels d'échange de fichiers, des logiciels de lecture de vidéos et musique, etc. En fait, les ordinateurs sont non seulement capables de traiter des informations de manières diverses, mais également de toutes les manières possibles. Ce sont des machines **universelles**.

Structure de l'ouvrage

Ce support de cours est organisé en quatre parties regroupant des chapitres, dont certains d'un niveau plus avancé :

1. **Dans la première partie**, Nous nous intéresserons à l'invention théorique de l'ordinateur. Nous verrons que les racines historiques de l'invention de cette machine sont très profondes.Deux grands mouvements ont attribués au progrès de la structure des ordinateurs : **l'automatique** et la **mécanique**. Nous verrons que les réflexions sur la logique, ont beaucoup contribué à l'invention de l'ordinateur.

2. **Dans la deuxième partie**, Nous abordons l'une des problématiques centrales de l'informatique : représenter les informations que l'on veut communiquer, stocker et transformer. Nous apprendrons à représenter les nombres entiers signés et non signés, les nombres à virgule et les caractères. « **Codage de l'informations** ».

3. **Dans la troisième partie**, Nous présentont les éléments d'algèbre de Boole qui constituent les fondements mathématiques nécessaires au traitement des informations quantifiées, et à la réalisation des différents opérateurs matériels qui réalisent ce traitement. les bases de la logique booléenne sont implémentatés sous forme de circuits logiques allons des premières portes logiques aux circuits élémentaires présents dans tous les circuits intégrés électroniques.

4. **Dans la quatrieme partie**, nous verrons les circuits et les portes logiques (composants logiques de base) qui sont associées aux différents opérateurs de l'algèbre de Boole. Les méthodes de synthèse de fonctions combinatoires permettent de réaliser des opérateurs de traitement plus complexes, comme les décodeurs, les multiplexeurs et les additionneurs.

Table des matiéres

I

Eléments de l'architecture de base

IV

- Les Circuits Logiques Combinatoires

Bibliography

Annexes

Table des Figures

Liste des Tables

Eléments de l'architecture de base

1. Introduction

L'histoire des ordinateurs est fortement liée aux découvertes théoriques dans le domaine des mathématiques, logique et éléctronique. Elle est marquée par la volonté de l'homme d'automatiser les calculs afin de les rendre plus précis et fiables. L'informatique couvre aujourd'hui à peu près toutes les branches de l'activité humaine. Les machines à calculer auquelles on s'intéresse sont des dispositifs physiques dont le fonctionnement repose sur l'emploi du courant électrique. En effet, l'information la plus simple que l'on puisse représenter sur un tel dispositif est (schématiquement) «le courant passe » ou « le courant ne passe pas 1». Autrement dit, l'ordinateur traite de l'information digitale, le support de l'information étant un système à deux états d'équilibre 0 et 1 correspondant aux informations élémentaires du système, appelées **Bits**, Tous traitement d'informations est codées sous une forme binaire (0 ou 1).

L'ensemble des organes physiques qui servent à ce traitement sont appelés **matériel** (en anglais Hardware), Le **logiciel** est un ensemble de programmes, langages et systèmes d'exploitation (en anglais Software) chargés d'assurer l'exploitation de les taches cette machine. Un ordinateur communique avec l'extérieur par l'intermédiaire d'organes d'entrée sortie (clavier, écran, imprimante, modem, etc.). Les résultats intermédiaires sont stockés dans des mémoires auxiliaires (bandes magnétiques, disques magnétiques, disques optiques, disques). L'organe principal d'un ordinateur est **l'unité centrale**, dans laquelle sont exécutées les instructions. Les ordinateurs ont des utilisations très variées (traitement de texte, gestion, calcul scientifique) qui nécessitent d'utiliser des langages de programmation adaptés à l'application envisagée. Le principal avantage de l'ordinateur est sa rapidité de calcul et d'accès aux informations.

Alors que les premières machines à calculer pouvaient tenir dans la main de l'homme, les premiers ordinateurs étaient des monstres mécaniques et électriques qui occupaient des pièces entières d'un immeuble, Figure 1.1. Un nouveau concept d'informatique voit le jour c'est l'informatique Ommiprésente ou ubiquitaire. La miniaturisation des unités centrales, la réduction de la consommation d'énergie et la généralisation des réseaux (Wifi, 3G) conduit à une omniprésence de dispositifs informatiques qui nous accompagnent dans notre vie courante : smart phones, console de jeux

portables reliées en réseau, PDA, ordinateurs portables, Figure 1.2. La multiplication des systèmes va changer radicalement notre façon de les utiliser. Les applications ne seront plus associées a une machine physique et surtout un écran associé. Elles vont pouvoir migrer et nous suivre au gré de nos déplacements (multiplication des processeurs nous entourant). Les interactions vont être plus naturelle et l'ordinateur va se fondre dans notre environnement et disparaitre.

Ordinateurs Monstre

Figure 1.1: Ordinateurs Monstre.

Dans ce qui suit, A partir de quelques rappels historiques nous allons faire un survol de l'évolution de l'ordinateur.

Ordinateurs de Poche

Figure 1.2: Miniaturisation des Ordinateurs.

2. Un peu d'histoire

2.1 Introduction

A partir de quelques rappels historiques nous allons mettre en èvidence les éléments constitutifs de l'ordinateur, cela nous conduira à décrire une famille de calculateurs. De la première machine à calculer, il aura fallut attendre près de quatre siècles pour arriver à l'informatique actuelle.

2.2 Naissance du nombre et du calcul

Dans l'antiquité, l'homme comptait avec des grains de blé et des cailloux. La grosseur du grain et du caillou est proportionnelle à la quantité d'objets qu'on veut représenter. D'ailleur, l'origine latine du mot **Calcul** signifie petit caillou ou grain.

Antiquité : 3000 ans av. JC, on écrit les chiffres sur une tablette de bois recouverte de sable. Le systéme de numération babylonien est à base 60 sans zéro, Figure 2.1.

Tablette de bois recouverte de sable

Figure 2.1: Numération babylonienne

Figure 2.2: Numération egyptienne

Les égyptiens écrivent les chiffres avec des barres : III = 3, Figure 2.2. Les Grecs écrivent les nombres avec les lettres de l'alphabet. Les Romains écrivent les chiffres avec des lettres majuscules représentant les doigts de la main : III = 3, V = 5, X = 10, C= 100. Il était extremement difficile de faire des calculs avec ces chiffres romains, Figure 2.3.

4ème siécle : Les indiens inventent le **Zéro** qui signifie rien en sanscrit. Les chiffres arabes de 0 à 9 apparaissent aux Indes vers le 5e siécle. Il se répand grace aux voyages de savants, Figure 2.4.

Les chiffres arabes

Figure 2.3: Numération Romaine

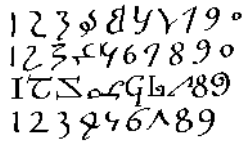


Figure 2.4: Numération Indo Arabe

2.3 Du boulier aux machines à calculer mécaniques

L'abacus d'origine babylonienne, est un boulier qui contient des rangées de boules qui se déplacent sur un axe ; la position des boules renvoie à un nombre Figure 2.5. On peut ainsi additionner, soustraire, multiplier et diviser. L'abaque est utilisé en Chine sous le nom de "**Suan Pan**".

Boulier Chinois.

Figure 2.5: Boulier Chinois.

Les premiéres machines à calculer connues furent conçues par Wilhelm Schikard (1592-1635) , Blaise Pascal (1642) et Gottfried Wilhelm Leibnitz (1646-1716). elles effectuent leurs opérations en base 10 à l'aide d'un mécanisme à roues dentées. L'enchainement des calculs est laissé à la charge de l'utilisateur. C'est ce souci d'automatisation d'une suite de traitements qui conduira à la

conception des ordinateurs.

- SCHICKARD Un professeur allemand de l'Université de Heidelberg qui, s'inspirant des mécanismes d'horlogerie, a l'idée de construire une machine à calculer en utilisant les roues dentelées d'horloge comme engrenages, Figure 2.6. Elle permettait la réalisation des quatre opérations arithmétiques.

La machine de SCHICKARD.

Figure 2.6: La machine de SCHICKARD.

- PASCAL Le mathématicien français Pascal présente à Paris sa **Pascaline**, machine à additionner et à soustraire des nombres et de convertir les différentes monnaies en usage à l'époque, Figure 2.7. Il fait appel à un mécanisme d'horlogerie : huit roues de neuf dents. La première roue tourne de neuf crans puis elle reprend sa position initiale pendant que la seconde roue avance d'un cran à la fois . On pouvait y'entrer deux nombres à la fois en bougeant des roues avec un stylus.

La machine à calculer de Blaise PASCAL.

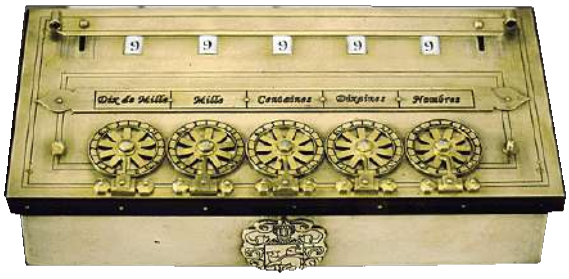


Figure 2.7: La machine à calculer de Blaise PASCAL.

- LEIBNIZ Le philosophe et mathématicien allemand Leibniz est fasciné par la machine à calculer mécanique de Pascal ; il va d'ailleurs la perfectionner en automatisant les opérations

de multiplication et de division, Figure 2.8. Les machines de Pascal et de Leibniz utilisent le systéme décimal. C'est à Leibniz que l'on doit l'idée de concevoir une **calculatrice** en binaire. Mais la technologie de l'époque n'en permet pas la réalisation.

Leibniz a passé 4 ans à étudier les théories de Descartes et de Pascal

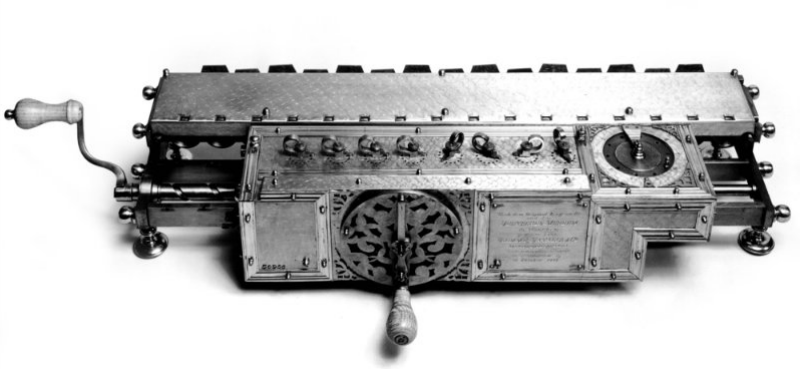


Figure 2.8: La machine de LEIBINIZ.

2.4 Des machines programmables à cartes perforées

- BABBAGE est le premier à entreprendre la réalisation d'une machine associant **automate** et **machine à calculer**. il présenta, dans un séminaire tenu à Turin, un projet de machine appelée **machine analytique** qui enchaine des opérations arithmétiques de façon autonome, elle est pilotée par une bande perforée, Figure 2.9.

BABBAGE jeta les bases de l'informatique moderne

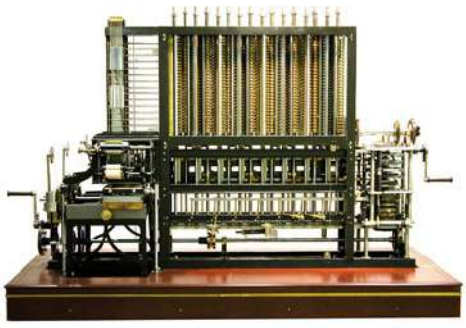


Figure 2.9: BABBAGE

L'utilisateur décrit la séquence des opérations que doit effectuer la machine sur cette bande. Elle est introduite dans la machine à chaque nouvelle exécution. En effet, si la machine

de Babbage est capable de mémoriser des résultats intermédiaires, elle ne dispose d'aucun moyen pour mémoriser les programmes dont le support est toujours externe, Figure 2.10. Dans cette machine apparaissent les notions de mémoire (le magasin) et de processeur (le moulin).

La machine analytique de Babbage.

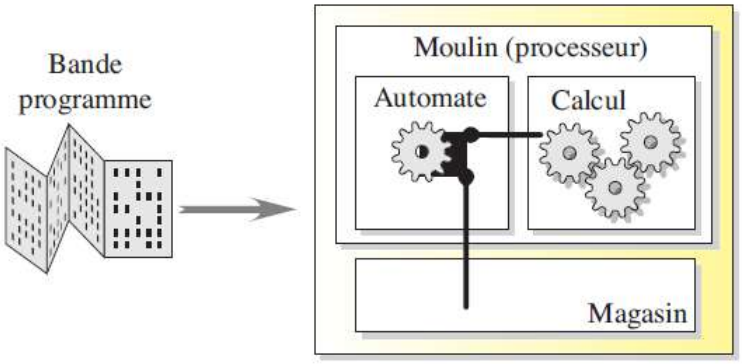


Figure 2.10: La machine analytique de Babbage.

- HOLLERITH En 1890, lors du recensement de la population aux USA concernant 62 millions de personnes, la nècessitè d'un traitement automatique des informations s'imposa. Il utilise les ressources de l'électricité dans une machine utilisant les principes de la mécanographie, Figure 2.11. Sa carte perforée déja utilisé par des sociétés ferroviaires aux USA comprenait 12 rangées de 20 positions à perforer dans lesquelles figuraient des données tels les noms et prénoms. Hermann HOLLERITH est aussi le créateur de la société TBM (Tabulating Machines Company) qui devint la société IBM (International Business Machines Corporation) conceptrice du PC (Personnal Computer).

La dimension des cartes corresponde à la taille du billet d'un dollar de l'époque

Figure 2.11: La machine de Herman Hollerith.

Avant meme l'utilisation de l'électricité, on voit déja se profiler les principes de base de

l'informatique moderne.Toutes ces machines traitent des données et fournissent un résultat. Ils ont été formalisés à l'ére des calculateurs électriques par **John Von Neumann (1903-1957)** , Figure 2.12.

John Von Neumann Mathématicien américain d'origine allemande.

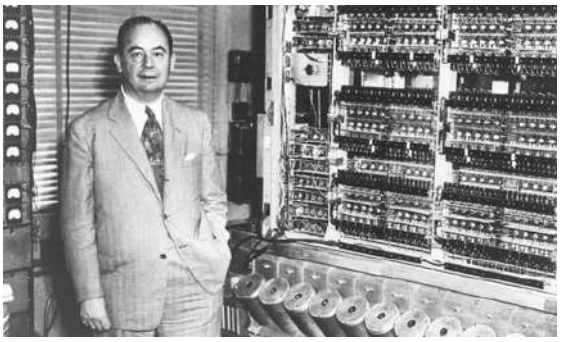


Figure 2.12: John VON NEUMANN.

Selon son principes, les quatre éléments fondamentaux d'un ordinateur sont : **la mémoire**, **l'unité logique**, **l'unité de controle** et **les différents organes d'entrée et de sortie**, Figure 2.13.

- Le processeur effectuer un certain nombre d'opérations (instructions) prédéfinies lors de sa conception. L'ensemble des opérations s'appelle le jeu d'instructions.
- Une suite d'instructions exécutées de façon séquentielle (les unes après les autres). La suite d'instructions réalisant le travail est appelée un programme.
- La mémoire du système peut contenir les données et le programme.
- La séquence d'un programme peut être interrompue afin d'exécuter une autre séquence. En fonction d'un test sur le résultat d'une opération. L'exécution d'un programme est définie par deux cycles (ou phases) principaux : **le cycle de recherche d'une instruction** et **le cycle d'exécution**.

La machine de John Von Neumann.

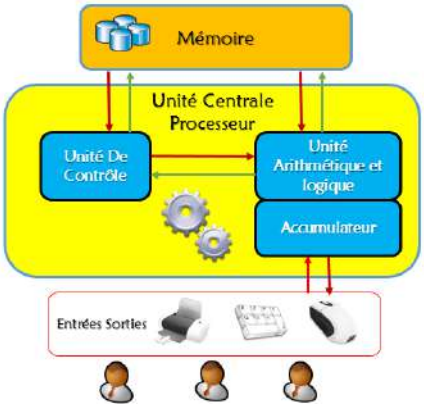


Figure 2.13: La machine de John VON NEUMANN.

Dans les années qui suivent, Von Neumann reprenant la structure de la machine de BABBAGE. Dans le cadre du projet Manhattan, il commence la construction de la machine à calculer automatique **E.D.V.A.C** (Electronic Discrete Variable Automatic Computer). Il travailla également sur le développement d'une machine appelée **E.N.I.A.C** (Electronic Numerical Integrator And Calculator) .

2.5 Les Générations

Depuis la réalisation de l'E.D.V.A.C, des centaines de machines ont vu le jour. Pour en structurer l'historique, on peut utiliser la notion de génération de calculateurs, essentiellement basée sur des considérations technologiques.

2.5.1 La première génération : Le passage à l'électronique (1938-1953)

Le passage à l'électronique, s'est fait grace à l'invention du tube à vide. Il permet de produire un courant direct d'électrons dans un tube sous vide capable de générer deux états : ON/OFF, Figure 2.14. A l'aide d'interrupteurs fermés pour "vrai" et ouverts pour "faux" il etait possible d'effectuer des opérations logiques en associant le nombre " 1 " pour "vrai" et "0" pour "faux". Ce codage de l'information est nommé base **Binaire**. C'est avec ce codage que fonctionnent les ordinateurs modernes. Il est possible de représenter physiquement cette information binaire par un signal électrique qui lorsqu'elle atteint une certaine valeur, correspond à la valeur 1.

Tube à vide.

Figure 2.14: Tube à vide.

Les machines de cette époque sont plutot des prototypes de laboratoire que des ordinateurs tels que nous les concevons aujourd'hui. Grosses consommatrices d'énergie, trés volumineuses, peu fiables, ces machines possédent un langage de programmation extremement primitif. Le développement de programmes représente un travail considérable. Les machines n'existaient qu'en exemplaire unique et étaient utilisées essentiellement à des fins de recherche ou par l'armée. Ce fut notamment le cas de l'**E.N.I.A.C** utilisée pour le décryptage des messages pendant la seconde guerre mondiale.

- **Turing 1943** : Alan Turing, un Hongrois installé en Angleterre a inventer une machine universelle appelée la machine de Turing qui jette les bases de ce que sera l'ordinateur moderne Figure 2.15. La machine comporte 2 000 tubes à vide, peut lire des rubans perforés à la vitesse de 5 000 caractéres à la seconde. Un seul ruban à la fois est donné à la machine. les résultats sont conservés dans une mémoire de triodes thyratron remplies de gaz.

La machine de TURING.

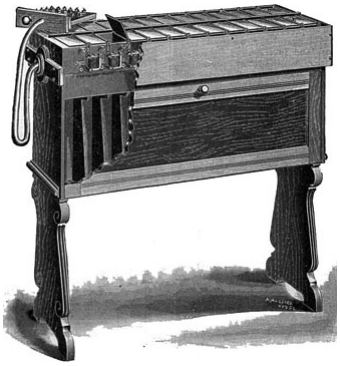


Figure 2.15: La machine de TURING.

- l'ENIAC (Electronic Numerical Integrator And Calculator) 1946 : A l'univetsité de Pensylvania, John Mauchly et J. Presper Eckert ont entrepris la construction de l'ENIAC Figure 2.16. Cette machine était destinée aux calculs balistiques (mouvements des projectiles). Elle était capable de 5 000 opérations arithmétiques à la seconde. fonctionnant une puissance électrique de prés de 200 KW. L'ENIAC comportait quelques 19 000 lampes., il pesait 30 tonnes et occupait un espace de 160 mètre carrés au sol. Sa capacité de mémoire était seulement de 20 mots de 10 unités chacun et elle ne pouvait traiter que des programmes d'instructions ne dépassant pas 300 mots. C'est durant les travaux de mise au point de l'ENIAC que John Tuckey créa l'unité binaire d'information **BIT** ou **Binary Digit**.

Une photographie de l'E.N.I.A.C.

Figure 2.16: L'E.N.I.A.C.

Un jour, en 1947, l'ENIAC tomba en panne sans que ses constructeurs ne sussent pourquoi. Aprés exploration, on constata qu'un **insecte** s'était logé dans un relais, Figure 2.17 ; le technicien qui a fait la découverte s'est écrié : **There is a bug in the machine**. Le nom **BUG** est resté pour désigner une erreur de matériel ou de programmation.

There is a bug in the machine

Figure 2.17: There is a bug in the machine

2.5.2 Deuxième gènèration : L'ère du Transistor (1953-1963)

John Bardeen, Walter Brattain et William Shockley (1947) inventent le transistor aux Bell Laboratories du New Jersey remplaçant le tubes à vide. Le transistor contient de la matière capable de conduire l'électricité à un voltage donné Figure 2.18. Il est beaucoup plus petit, moins cher à fabriquer et beaucoup plus fiable. Cependant, en raison de problémes de production, ce n'est qu'à la fin des années 1950 que le transistor s'est répandu.

Le transistor remplacer le tubes à vide.

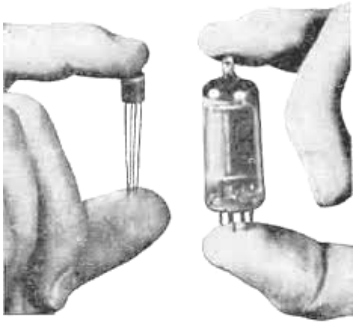


Figure 2.18: Le transistor.

- IBM 1401: Trés populaire, IBM en a livré plus de 10 000 à la petite et moyenne entreprise. Il comporte 150 000 transistors et a une vitesse de traitement de 200 instructions à la seconde Figure 2.19. C'est le plus rapide de son époque, jusque-là, les machines ne disposent pas d'un environnement de développement ni d'interface utilisateur. Des modules d'entrée-sortie pré-programmés (I.O.C.S., Input Output Control System) sont proposés pour faciliter le travail de programmation.
- Compatible Time Sharing System: Mis au point au M.I.T (1961). Le lancement de chaque tàche (édition, traitement, etc.) est réalisé de façon autonome et par lot (**Batch Processing**). C'est un concept majeur pour l'époque, il s'agit de permettre à plusieurs usagers de trvailler au meme temps mais chacun son tour, sans qu'il s'en aperçoive, d'avoir accés à la puissance de calcul d'un gros ordinateur à partir de terminaux à distance.
- Mémoires principales: A partir de 1959, les ordinateurs de deuxiéme génération possédaient

La Serie IBM 1401.

Figure 2.19: La Serie IBM 1401.

des mémoires principales magnétiques à tores (un anneau magnétique). Quelques-uns d'entre eux possédaient comme mémoire auxiliaire des tambours magnétiques et des rubans magnétiques. Nommons le Philco 2000, les CDC 1604, 3600, les IBM 7000, Figure 2.20, 1400, les RCA 302, 501, le Honeywell 800, les UNIVAC III, 1107, les Ferranti Atlas.

La Serie IBM 7000.

Figure 2.20: La Serie IBM 7000.

- Les langages évolués: Ils font leur apparition sous le nom de systémes de codage symbolique. Le premier compilateur FORTRAN (FORmula TRANslator) date de 1957 et équipe l'IBM-704. Le langage COBOL (COmmon Business Oriented Langage) voit le jour en 1959 sous le nom de COBOL 60. Des applications de grande taille dans le domaine de la gestion sont développées.

2.5.3 Troisiéme génération : l'ère des circuits intégrés (1964-1975)

Dans les premiers ordinateurs, les éléments des circuits étaient reliés entre eux par des réseaux de fils extremement complexes . L'invention du transistor va vite appeler le développement d'une

technologie qui permettra d'integrer les composants de l'ordinateur sur une meme plaque de circuits. Les liaisons électriques multiples entre chaque transistor sont complexes, couteuses à réaliser, pas assez rapides et peu fiables ; **le circuit imprimé** va résoudre ces problémes. Par la suite, on a été capable de relier entre eux tous les éléments du circuit, transistors, diodes, condensateurs, fils, etc. dans des circuits dits complétement **intégrés**.

- les circuits intégrés : Cette percée technologique n'a pas connu tout le succés qu'elle méritait, on prétendait que le cout en était trop élevé. Le lancement du satellite Russe **Spoutnik** souleve de l'inquiétude chez les Américains , **John Kennedy** decida d'envoyer un homme sur la lune. Pour réaliser cette conquete de la lune, il fallait des ordinateurs assez petits pour tenir dans une capsule spatiale, l'ordinateur de la cabine Apollo ne pesait que 54 livres. Ces efforts de la NASA, ont permis de financer la recherche de développement sur les circuits intégrés et d'en réduire les couts de fabrication, cela a conduit leurs presences dans toutes sortes d'appareils électroniques y compris les appareils domestiques, radio, télévision, horloges, etc. Grace aux circuits intégrés qu'on a pu construire une nouvelle generation d'ordinateurs appelée Mini-Ordinateurs. Ces circuits intégrés sont miniaturisés et déposés sur une pastille de quelques millimétres carrés, appelée **Puce**, faite de matériau semi-conducteur, généralement le **Silicium**.

Satellite lancés par l'URSS.

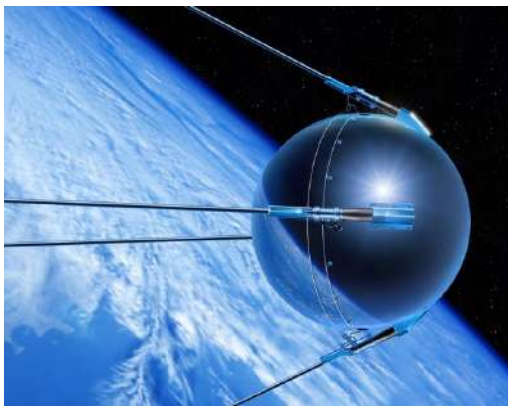


Figure 2.21: spoutnik.

- La multiprogrammation: Aux progrès des performances matérielles va s'ajouter le concept de multiprogrammation destinée à optimiser l'utilisation de la machine. Plusieurs programmes résident simultanément en mémoire permettant ainsi une commutation rapide de l'un vers l'autre. La notion d'indépendance vis-à-vis des dispositifs d'entrée-sortie (device independance) fait son apparition ; le systéme d'exploitation se charge lui-meme de rechercher une unité possédant les caractéristiques d'ecrites. Le programmeur n'a plus à spécifier de manière explicite l'unité sur laquelle s'effectue l'opération d'entrée-sortie.

- L'émergence du logiciel: Jusqu'alors, les ordinateurs de marques différentes et d'une meme marque n'étaient pas compatibles. On devait réécrire les programmes quand on changeait de type de machine. IBM décida de mettre au point une famille d'ordinateurs à travers laquelle les programmes et les périphériques seraient interchangeables, la famille des IBM360, Figure 2.22. Le constructeur I.B.M. introduit une nouvelle politique de distribution de ses produits (unbundling) séparant matériel et logiciel. Il devient alors possible de se procurer

du matériel compatible IBM et des logiciels développés par des sociétés de service. Cette politique a provoqué l'apparition d'une puissante industrie du logiciel indépendante des constructeurs de machines.

La Serie IBM 360.

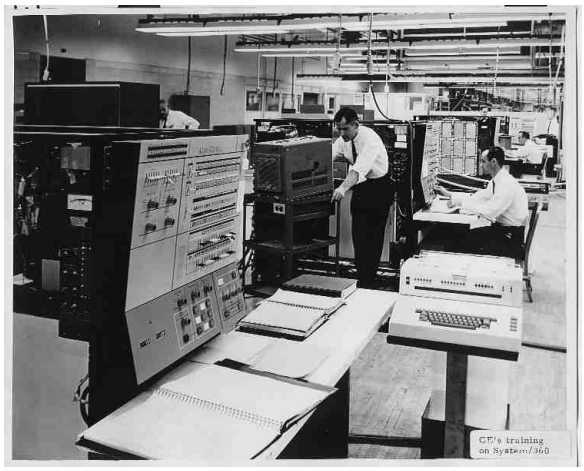


Figure 2.22: La Serie IBM 360.

2.5.4 Quatriéme génération : l'ère des microprocesseurs (1975-. . .)

Il n'y a pas réellement de rupture technologique mais plutot amélioration considérable des procédés de fabrication des circuits avec l'assistance de l'ordinateur. La mise au point des microprocesseurs **Intel** va entraıner la miniaturisation des composants d'ordinateurs, conséquance : apparition de deux nouveaux types d'ordinateurs : le super ordinateur et le micro-ordinateur ou ordinateur personnel. La miniaturisation va aussi permettre l'invention des calculatrices de poches, des montres à affichage numérique, d'appareils domestiques comme le four à micro-ondes, la machine à laver. Parallélement se produit une explosion dans le domaine du développement des logiciels. Les concepteurs font de plus en plus appel au parallélisme dans l'architecture des machines pour en améliorer les performances sans avoir à mettre en oeuvre de nouvelles technologies (pipeline, vectorisation, caches, etc.).

- 1968 : Robert Noyce et Gordon Moore fondent **Intel Corp** qui va devenir le plus gros fabricant de puces et de microprocesseurs au monde. De son coté, la société **AMD** est crée par W.J. Sanders III de Fairchild pour compétitionner Intel sur le marché des microprocesseurs, Figure 2.23.

- 1970 : Bob Abbott, sous la direction de Les Vadasz, met au point chez Intel, la premiére puce à mémoire dynamique, la DRAM (Dynamic Read Access Memory), INTEL 1103, qui deviendra à compter de 1972, la plus vendue à travers le monde. Elle n'avait qu'un seul kilo-octet. Sa vitesse était de 300 nanocecondes.

- 1971 : L'equipe de la compagnie Intel est la premiére à concevoir le design d'un ordinateur sur une puce, autrement dit un microprocesseur. Un microprocesseur à 4 bits , le 4004 : 2 300 transistors de 10 microns sur une puce de 12 mm carré. Le Intel 4004 tournait à la vitesse de 108 KHz, soit 0,06 MIPS (million d'instructions par seconde). Son bus de mémoire était de

Amd vs Intel.

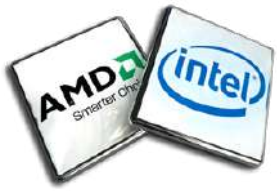


Figure 2.23: Amd vs Intel.

4 Ko. sa mémoire adressable était de 640 octets. Il fallait 16 griffes pour le fixer, Figure 2.24.

INTEL 4004.

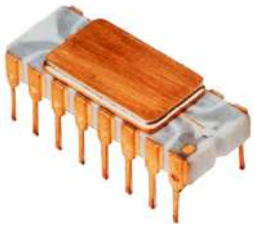


Figure 2.24: INTEL 4004.

- 1977 : **Apple**, fondée par Steve Jobs et Steve Wozniak lance le Apple II, un ordinateur à 4 Ko de mémoire vive (RAM) , 16 Ko de mémoire morte (ROM), un processeur de 1 MHz et un clavier intégré de 52 touches, Figure 2.25. il était facile à programmer grace à son langage Basic intégré en mémoire morte. Il deviendra rapidement l'ordinateur favori du monde scolaire.

APPLE II.

Figure 2.25: APPLE II.

- 1978 : Intel lance le premier microprocesseur à 16 bits, le 8086 à 4,77 MHz qui deviendra, à l'époque, un standard de l'industrie, Figure 2.26. Il comprenait 29 000 transistors de 3 microns soit six fois plus que le 8080. Il permettait la division et la multiplication accélérant ainsi les calculs de précision. La performance du 8086 est dix fois celle de son ancetre à 8 bits, le 8080. Son bus est de 16 bits et sa mémoire adressable est de 1 Mo.

INTEL 8086.

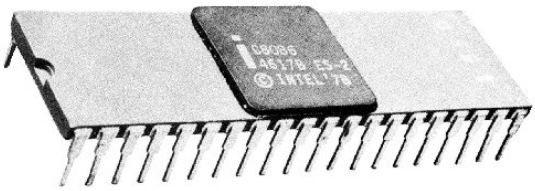


Figure 2.26: INTEL 8086.

- 1981 : IBM, sentant Apple devant elle dans un nouveau marché, lance sur le marché du micro-ordinateur avec le concept d'ordinateur personnel qui prend la forme du IBM PC construit autour du microprocesseur Intel 8088 à 8-16 bits et 4,77 Hhz. Il avait 16 Ko de mémoire vive, extensible à 64 Ko , un seul lecteur de disquettes simple face de 5,25 po d'une capacite de stockage de 160 Ko. Il etait doté d'un ecran monochrome vert de 12 po. Il etait offert 'a 2 880 US. Il fonctionne sous un systéme d'exploitation mis au point par Microsoft sous contrat avec IBM, le MS DOS et le PC DOS, Figure 2.27.

IBM PC.

Figure 2.27: IBM PC.

- 1984 : Apple lance le Macintosh, Figure 2.28. Basé sur le projet LISA , c'est l'ordinateur convivial par excellence : Son utilisation est trés simple grace à la souris et à la qualité de ses graphismes. Il devient au fil des années et des version, l'autre grand standard (avec le PC d'IBM) du monde de la micro-informatique.

- 1984+++: Les techniques évoluent et les batailles commerciales font rage, mais exploitent toujours le meme principe de base, comme les premiers ordinateurs : l'architecture Von Neumann.

Macintosh

Figure 2.28: Macintosh.

2.5.5 Cinquieme génération : l'ère des objets : IoT (1991. . .)

Aujourd'hui, L'informatique sort de l'ordinateur pour s'intégrer directement dans l'environnement. L'idée est de faire en sorte que l'ordinateur profite à l'utilisateur à tout moment lorsqu'il se trouve dans cet environnement. L'accès à des services en situation de mobilité. Le concept de l'informatique **pervasive** est un terme qui a été introduit pour la première fois au début des années 90 par**Mark Weiser** au laboratoire Xerox parc . Pour désigner sa vision futuriste de l'informatique du XXIe siècle. Il imaginait un monde peuplé d'objets informatiques et numériques qui seraient reliés en réseaux à très grande échelle et interagiraient de manière autonome et transparente afin d'accomplir diverses tâches de la vie quotidienne, **??**. Dans son article fondateur intitulé: <The Computer for the 21st Century, 1991>, il donna la vision de l'informatique du futur:

"Les technologies les plus profondes sont celles qui disparaissent. Elles s'emmêlent dans le tissu de la vie de tous les jours jusqu'au point où elles en deviennent indiscernables".

Ainsi, pour Weiser, trois éléments sont nécessaires : des ordinateurs bon marché et à faible consommation d'énergie qui proposent des interfaces d'affichage plus commodes, des logiciels d'informatique ubiquitaire et un réseau qui relie tous les dispositifs. L'informatique ubiquitaire ne vit pas dans un objet particulier, Figure 2.29, mais elle est imprégnée dans la structure même de l'environnement qui nous entoure.

Objet communicant : Nabaztag.

Figure 2.29: Objet communicant : Nabaztag..

L'idée novatrice de la vision de Weiser part du constat que bientôt, chaque personne sera entourée par de nombreux ordinateurs. Nous sommes passés du stade de l'ordinateur central (mainframe) qui a concrétisé l'idée d'un ordinateur pour plusieurs utilisateurs au stade plusieurs ordinateurs enfouis dans des objets de la vie quotidienne pour une même personne. De plus, les communications débordent du cadre classique homme vers homme ou homme vers machine pour inclure des communications directes entre machines, Figure 2.30. Dans le chapitre suivant, nous d'etaillons les différents composants d'un micro-ordinateur moderne.

Evolution de l'informatique jusqu'à l'informatique pervasive.

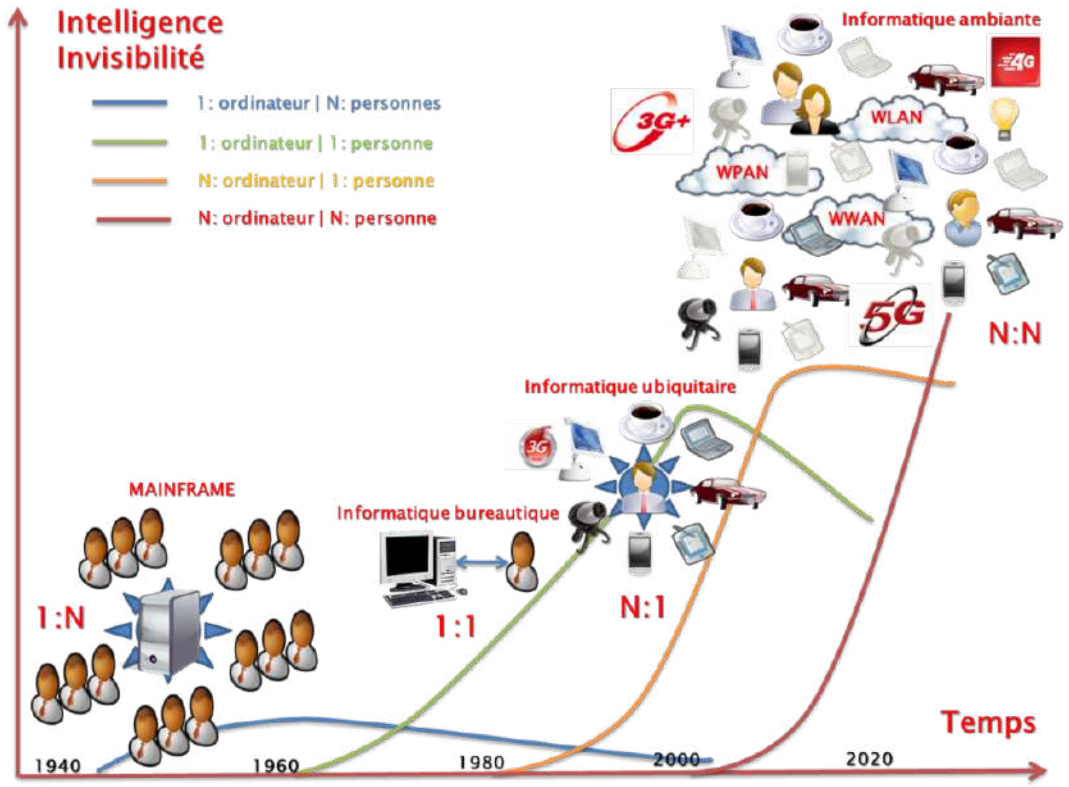


Figure 2.30: Evolution de l'informatique jusqu'à l'informatique pervasive.

3. Éléments d'architecture d'un ordinateur

Dans le chapitre précédent nous avons briévement présenté les moments importants ayant jalonné le chemin menant du boulier aux ordinateurs actuels. Nous avons vu que ce parcours a été marquée tant par des innovations techniques que par l'introduction d'outils mathématiques permettant de décrire de maniére cohérente opérations, opérandes et opérateurs. Dans ce qui suit, Nous commençons par décrire le matériel informatique constituant un ordinateur permettant son fonctionnement et donner les principales caractéristiques à connaıtre.

3.1 Présentation de l'ordinateur

Pour commencer, interrogeons-nous sur la significtion-même du terme « **informatique** »

Definition 3.1.1 — Informatique. Le mot informatique est une contraction des deux termes Information et Automatique. Ainsi, l'informatique est la science du traitement automatique de l'information.

Il s'agit donc d'appliquer à un ensemble de données initiales des règles de transformation ou de calcul déterminées (c'est le caractère automatique), ne nécessitant donc pas de reflexion ni de prise d'initiative.

Definition 3.1.2 — Ordinateur. Il s'agit d'un appareil concret permettant le traitement automatique des données.

3.2 Évolution du Traitement des Ordinateurs

Un ordinateur est une machine à calculer automatique. L'objectif d'un ordinateur est de prendre en entrée des données, d'effectuer un traitement automatique et de fournir en sortie des résultats, Figure 3.1.

Le problème avec ce schéma, s'il ne comporte qu'une unité de traitement est qu'on ne peut pas effectuer des opérations complexes nécessitant des traitements intermédiaires, de type (4 + 5) + (6

traitement automatique se fait sans intervention de l'humain)

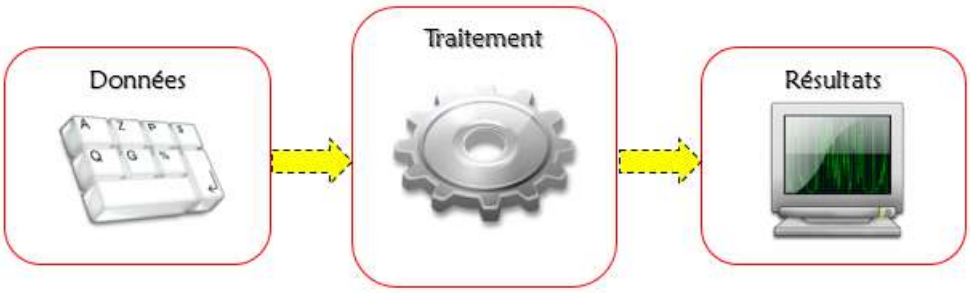


Figure 3.1: Données, Traitements, Résultats.

+ 7). Il faudrait mémoriser le résultat intermédiaire (4 + 5 = 9). 4, 5, 6 et 7 sont les opérandes de l'instruction, + est un opérateur de l'instruction. Rajoutons donc une **mémoire** qui permettrait de stocker les résultats intermédiaires, Figure 3.2.

Ajout de mémoire)

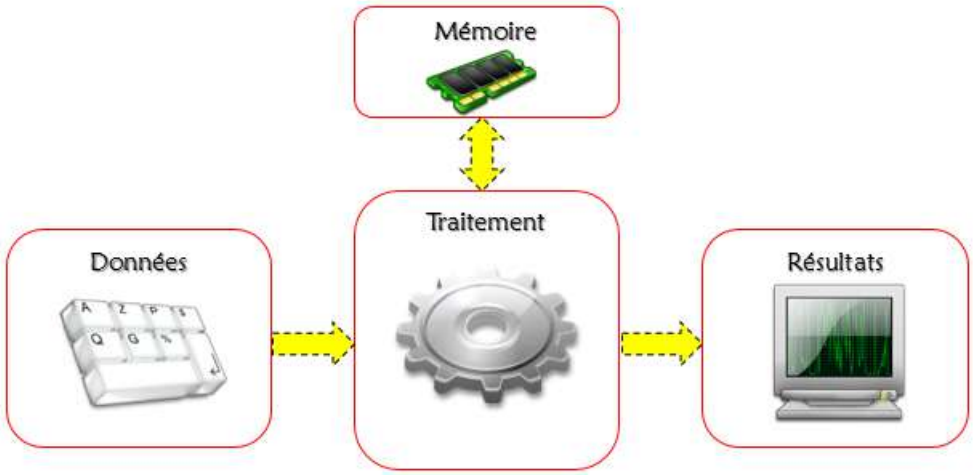


Figure 3.2: Données, Traitements, Mémoire, Résultats.

Problème, la machine ne sait toujours effectuer qu'un seul traitement. Il faudrait concevoir une machine susceptible d'effectuer à un instant donné un traitement choisi parmi plusieurs possibles (additions, multiplications, soustractions, etc.). Le traitement que la machine pourrait désormais effectuer serait le résultat d'une suite d'actions élémentaires. Cette suite d'actions est un **programme**. On construit donc un organe particulier, que l'on appelle le programmateur (il trouve ses ordres sur un support : carte perforée, ruban, etc.), Figure 3.3. C'est l'architecture développée par Babbage dès 1812. Le nombre de traitements possibles est dorénavant pratiquement infini.

Problème : l'execution du programme dépend de la vitesse du défilement du support(trop lente et trop rigide). Von Neumann a l'idée de placer le programme en mémoire. La vitesse d'execution du programme est limitée par le temps de transfert en mémoire, Figure 3.4.

L'architecture de Von Neumann est composée d'une mémoire, d'un programme stocké en mémoire, et d'une unité de traitement (unité d'entrée et de sortie et unité arithmétique et logique).

Ajout du Programmateur

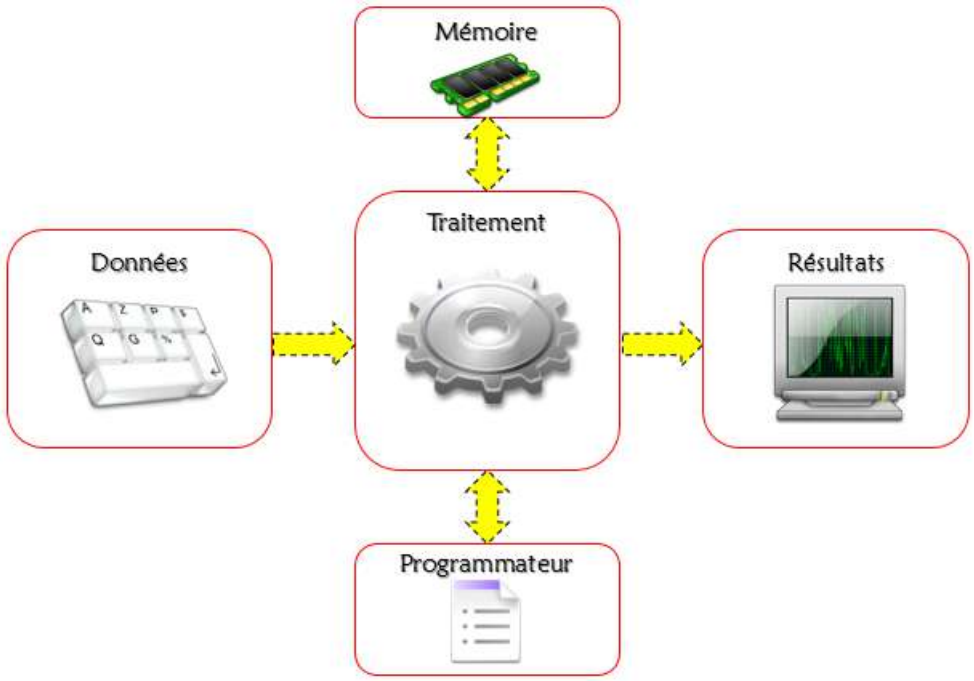


Figure 3.3: Données, Traitements, Mémoire, Programmateur, Résultats.

Architecture de Von Neumann

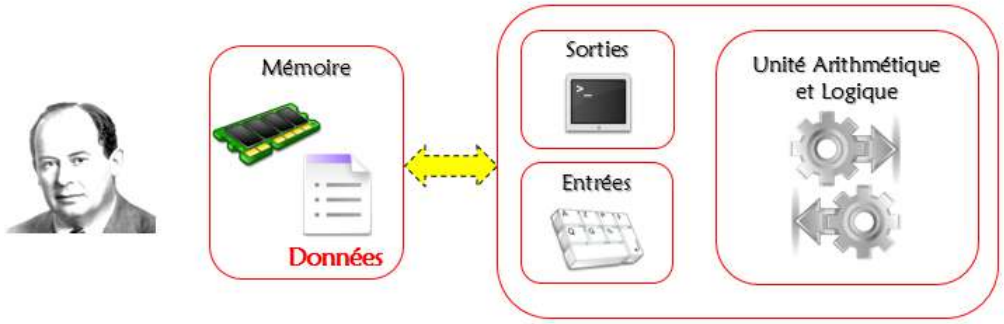


Figure 3.4: proposition de Von Neumann.

Le traitement du programme se fait de manière séquentielle. C'est à dire, de la première à la dernière ligne du programme.

3.3 Types d'ordinateurs

Toute machine capable de manipuler des informations binaires peut etre qualifiée d'ordinateur. Toutefois, la plupart des personnes pensent à un ordinateur personnel (PC, abréviation de personal computer), le type d'ordinateur le plus présent sur le marché, toutefois il existe beaucoup d'autres types d'ordinateurs :

- Ordinateur de bureau, Figure 3.5 ;
- Ordinateur portable ;
- Mini PC ;
- Netbook ;
- Tablette PC, Figure 3.6;
- Assistants Personnels, Figure 3.7.

Les chiffres arabes

Figure 3.5: Ordinateur de bureau

Figure 3.6: Tablette PC

Figure 3.7: Assistants Personnels

Nous nous intéresserons dans la suite qu'aux ordinateurs de bureau, appelés aussi ordinateurs compatible IBM, car IBM est la firme qui a creér les premiers ordinateurs de ce type et a longtemps été le leader dans ce domaine, à un tel point qu'elle controlait les standards, copiée par les autres fabricants.

3.4 Constitution de l'ordinateur

Un ordinateur est un ensemble de composants électroniques modulaires architecturés autour d'une carte principale comportant de nombreux circuits ; cette carte est appelée carte mère. Elle est logée dans un boıtier, ce dernier comporte des emplacements pour les périphériques de stockage, ainsi que des boutons et des voyants permettant de controler la mise sous tension et l'état de l'activité des disques durs. Sur la face arrière le boıtier propose des interfaces d'entrée-sortie connectées sur la carte mère. Enfin le boıtier héberge une alimentation, chargée de fournir un courant électrique stable à l'ensemble des éléments constitutifs de l'ordinateur.

Un ordinateur est généralement composé au minimum d'une unité centrale, un écran (moniteur), d'un clavier et d'une souris, mais il est possible de connecter une grande diversité de périphériques externes sur les interfaces d'entrée-sortie (ports séries, port parallèle, port USB, port firewire), ajoutant :

- Imprimante ;
- Scanner ;
- Périphérique de stockage externe ;
- Appareil photo ou caméra numérique ;
- Hauts Parleurs ;

Il est donc nécessaire que l'ordinateur puisse communiquer avec l'utilisateur, pour permettre l'entrée des données initiales, la sortie du résultat du traitement, et l'entrée des règles

Structure matérielle générale.

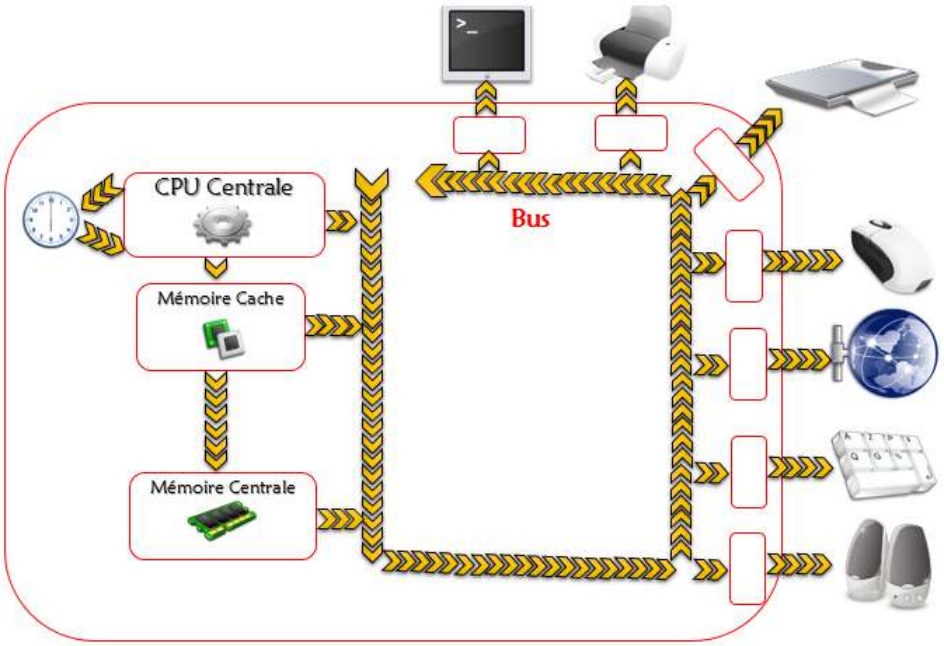


Figure 3.8: Structure matérielle générale.

d'automatisation, sous la forme d'un programme. Le modèle le plus couramment adopté pour décrire de façon très schématique le fonctionnement d'un ordinateur est celui décrit dans la Figure 3.8 , appelé **Architecture de Von Neumann**.

[Entrées - sorties:] se font au moyen de périphériques spéciaux destinés à cet usage.

- Les périphériques d'entrée permettent à un utilisateur d'entrer à l'ordinateur des données, sous des formats divers : clavier, souris, scanner, webcam, manettes de jeu.
- Les périphériques de sortie permettent de restituer des informations à l'utilisateur : écran, imprimante, hauts-parleurs.
- Certains périphériques peuvent parfois jouer à la fois le rôle d'entrée et de sortie, comme les écrans tactiles.

[La mémoire:] permet le stockage des données et des logiciels (programmes) utilisés pour les traiter. Ce stockage peut être :

- Définitif (mémoire morte, ou ROM, inscrite une fois pour toute, et non modifiable, à moins d'interventions très spécifiques).
- Temporaire à moyen et long terme (stockage de données et logiciels que l'utilisateur veut garder, au moins momentanément)
- Temporaire à court terme (données stockées à l'initiative du processeur en vue d'être utilisées ultérieurement.

[Le processeur :] est le coeur de l'ordinateur:

- C'est la partie de l'ordinateur qui traite l'information. Il va chercher les instructions enregistré en mémoire, ainsi que les données nécessaires à l'exécution du programme, il traduit les instructions du programme en une succession d'opérations élémentaires, exécutées ensuite par des unités de calcul.

[Le transfert des données :] se fait à l'aide de cables transportant des impulsions électriques, appelés bus. Un bus est caractérisé par:

- Le nombre d'impulsions électriques (appelées bit) qu'il peut transmettre simultanément. Ce nombre dépend du nombre de conducteurs électriques parallèles dont est constitué le bus.
- La fréquence des signaux, c'est-à-dire le nombre de signaux qu'il peut transmettre de façon successive dans un temps donné. Ainsi, un bus de 25MHz peut transmettre 25 millions d'impulsions sur chacun de ses fils chaque seconde.
- Les bus se répartissent en 2 types :

  Les bus parallèles constitués de plusieurs fils conducteurs, et permettant de transmettre 1 ou plusieurs octets en une fois.

  Les bus séries, constitués d'un seul conducteur : l'information est transmise bit par bit.
- Un ordinateur utilise des bus à 3 usages essentiellement :

  Le bus d'adresse, dont la vocation est l'adressage en mémoire (trouver un endroit en mémoire). C'est un bus unidirectionnel.

  Les bus de données, permettant la transmission des données entre les différents composants. Ce sont des bus bidirectionnels.

  Les bus de contrôle, indiquant la direction de transmission de l'information dans un bus de données. .

3.4.1 La carte mère

L'élément constitutif principal de l'ordinateur est la carte-mère, c'est sur cette carte que sont connectés ou soudés l'ensemble des éléments essentiels de l'ordinateur, Figure 3.9. La carte-mère contient des éléments embarqués (intégrés à la carte) :

L'élément constitutif principal de l'ordinateur.

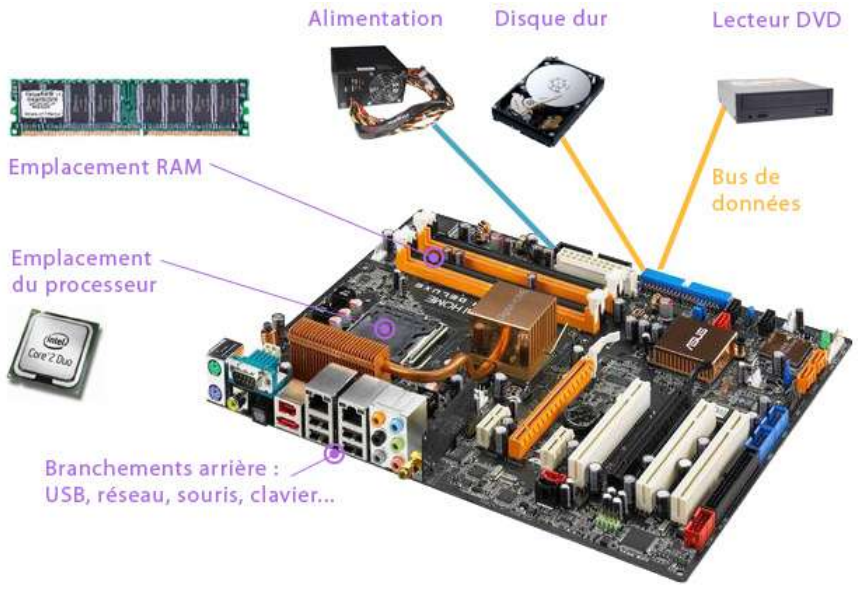


Figure 3.9: L'élément constitutif principal de l'ordinateur.

- Le chipset, circuit qui controle la majorité des ressources (interface de bus du processeur,mémoire cache et mémoire vive, slots d'extension) ;
- L'horloge et la pile du CMOS ;
- Le BIOS.

Le chipset

Le chipset est un circuit électronique chargé de coordonner les échanges de données entre les divers composants de l'ordinateur (processeur, mémoire, ...), Figure 3.10. Certains chipsets intégrent parfois une puce graphique ou une puce audio (généralement sur les PC bas de gamme).

R Etant donné la piètre qualité de ces composants intégrés (carte graphique ou une carte son), il est généralement conseillé de les désactiver dans le setup du BIOS et d'installer des cartes d'extension.

Chipset Intel.

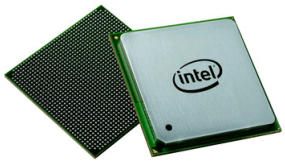


Figure 3.10: Chipset Intel.

L'horloge et la pile du CMOS

L'horloge est un circuit chargé de la synchronisation des signaux du systéme. On appelle fréquence de l'horloge (exprimée en Mhz) le nombre de vibrations (tops) d'horloge émis par seconde. Plus la fréquence est élevée, plus le systéme pourra traiter d'informations. Lorsque vous mettez votre ordinateur hors tension, l'alimentation cesse de fournir du courant à la carte-mère. Or, lorsque vous le rebranchez ou qu'une panne d'électricité intervient, votre système d'exploitation est toujours à l'heure, un circuit électronique appelé CMOS (Complementary Metal-Oxyde Semiconductor, parfois appelé BIOS CMOS) conserve certaines informations sur le systéme, y compris l'heure et la date système. Le CMOS est continuellement alimentée par une pile située également sur la carte-mère, Figure 3.11. **Ainsi, si vous constatez que votre PC à tendance à oublier l'heure, ou que l'horloge prend du retard, pensez à en changer la pile !**

Le BIOS

Le BIOS (Basic Input/Output System) est le programme basique servant d'interface entre le systéme d'exploitation et la carte-mère. Le BIOS est stockè dans une ROM, il utilise les données contenues dans le CMOS pour connaıtre la configuration matérielle du systéme.

R Il est possible de configurer le BIOS grace à une interface (**BIOS setup**) accessible au démarrage de l'ordinateur par simple pression d'une touche, Figure 3.12. Pour plus d'informations n'hésitez pas à vous reporter au manuel de votre carte-mère.

Pile.

Figure 3.11: Pile.

bios.

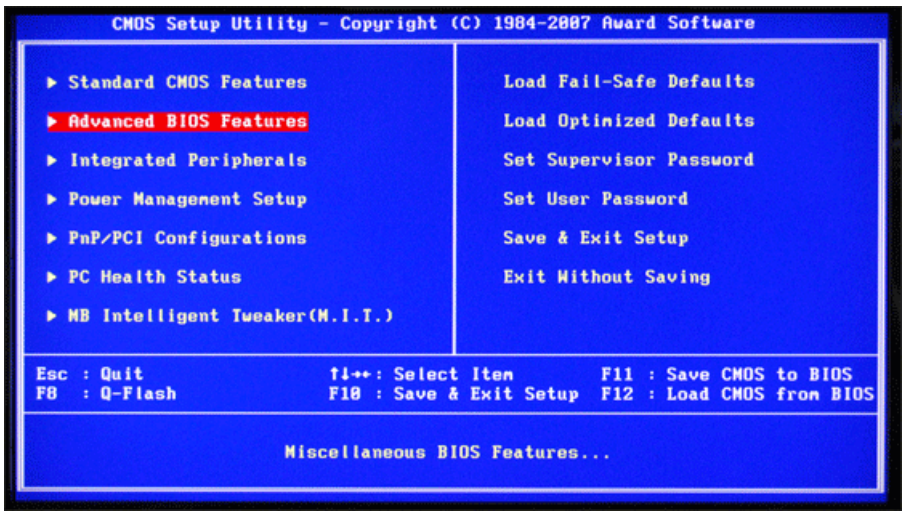


Figure 3.12: bios.

3.4.2 Mémoires

Nous revenons dans ce paragraphe sur un des composants sans lequel un ordinateur ne pourrait rien faire : la mémoire. Elle est caractérisée :

- **Par sa taille** nombre d'octets disponibles pour du stockage. Suivant le type de mémoire, il vas de quelques octets à plusieurs Gigaoctets ;
- **Par sa volatilité**, c'est-à-dire le fait d'être effacée ou non en absence d'alimentation électrique.
- **Par sa réinscriptiblité** ou non : mémoire morte ou mémoire vive.

Nous énumérons ci-dessous différents types de mémoire qui sont en évolution constante, aussi bien par leur forme et capacité que par les techniques ou principes physiques utilisés.

[Mémoire morte (ROM, read-only memory):] est un type de mémoire permettant de conserver les informations qui y sont contenues meme lorsque la mémoire n'est plus alimentée électriquement.Il s'agit de mémoire non volatile, donc non reprogammable. Cette mémoire ne peut etre accédée qu'en lecture. Toutefois il est désormais possible d'enregistrer des informations

dans certaines mémoires de type ROM (elle doit être flashée avec des ultras-violets).

- Différentes mémoires de type ROM contiennent des données essentielles au démarrage.
  Le BIOS: Un programme permettant de piloter les interfaces d'entrée-sortie principales du système.
  Le chargeur d'amorce : Un programme permettant de charger le système d'exploitation en mémoire (vive) et de le lancer. Celui-ci cherche généralement le système d'exploitation d'un support de stockage(ldisque dur).
  Le Setup CMOS:, C'est l'écran disponible à l'allumage de l'ordinateur permettant de modifier les paramètres du système.

[Mémoire vive (RAM, random access memory):] La mémoire vive appelée RAM (mémoire à accès aléatoire), est la mémoire principale du système, il s'agit d'un espace permettant de stocker de maniére temporaire des données lors de l'exécution d'un programme.

- La mémoire vive est une mémoire volatile, utilisée par l'ordinateur pour le traitement des données, lorsqu'il y a nécessité de garder momentanément en mémoire un résultat dont il aura à se resservir plus tard. Elle est d'accès rapide, mais peu volumineuse. Elle se présente généralement sous forme de barrettes à enficher sur la carte-mère, Figure 3.13.

Mémoire vive RAM.

Figure 3.13: Mémoire vive RAM.

[Mémoire-cache:] La mémoire-cache permet au processeur de se « rappeler » les opérations déjà effectuées auparavant. En effet, elle stocke les opérations effectuées par le processeur, pour qu'il ne perde pas de temps à recalculer des choses qu'il a déjà faites précédemment. Sur les ordinateurs récents ce type de mémoire est directement intégré dans le processeur.

[Mémoires de masse:] Ce sont des mémoires de grande capacité, destinées à conserver de façon durable de grosses données (bases de données, gros programmes, informations diverses). De par leur vocation, ce sont nécessairement des mémoires non volatiles (on ne veut pas perdre les données lorsqu'on éteint l'ordinateur !). Par le passé, il s'agissait de bandes perforées, puis de cassettes, de disquettes etc. Actuellement, il s'agit plutôt de disques durs internes, Figure 3.14 ou externe, Figure 3.15, de bandes magnétiques (fréquent pour les sauvegardes régulières), de CD, DVD, Figure 3.16, ou de mémoires flash (clé USB par exemple).

Mémoire de masse.

Figure 3.14: Disque dur Interne

Figure 3.15: Disque dur Externe

Figure 3.16: CD/DVD

[Mémoires flash:] Les mémoires flash (clé USB par exemple), Figure 3.17. Techniquement parlant, il s'agit de mémoire morte (EEPROM : electrically erasable programmable read-only memory), mais qui peut être flashée beaucoup plus facilement que les EPROM, par un processus purement électrique. Ce flashage fait partie du fonctionnement même de ces mémoires, ce qui permet de les utiliser comme des mémoires réinscritptibles et modifiables à souhait.

Disque mémoire Flash Usb.

Figure 3.17: Disque mémoire Flash Usb.

Une caractéristique très importante de la mémoire est son temps d'accès, qui représente un facteur limitant du temps de traitement de données. Ce temps d'accès est bien entendu dépendant du type de mémoire et de la technologie utilisée pour cette mémoire. Les mémoires électroniques, composées de circuits bascules (ou circuits bistables), sont rapides d'accès, tandis que les mémoires à base de transistors et de condensateurs (technologie usuelle des barrettes de RAM) sont plus lentes d'accès (temps de charge + temps de latence dû à la nécessité d'un rafraichissement périodique, du fait de la décharge naturelle des condensateurs). Les mémoires externes nécessitant un processus de lecture sont encore plus lentes (disques durs, CD...).

Pour cette raison, les mémoires les plus rapides sont aussi souvent les moins volumineuses, et les plus onéreuses (qualité des matériaux plus coût de la miniaturisation), le volume étant d'autant plus limité que d'un point de vue électronique, un circuit bistable est plus volumineux qu'un transistor. On représente souvent la hiérarchie des mémoires sous forme d'un triangle, Figure 3.18.

3.4.3 Les connecteurs d'extension

Les connecteurs déxtension (en anglais slots) sont des réceptacles dans lesquels il est possible d'enficher des cartes déxtension, c'est-à-dire des cartes offrant de nouvelles fonctionnalités ou de

Hiérarchie des mémoires.

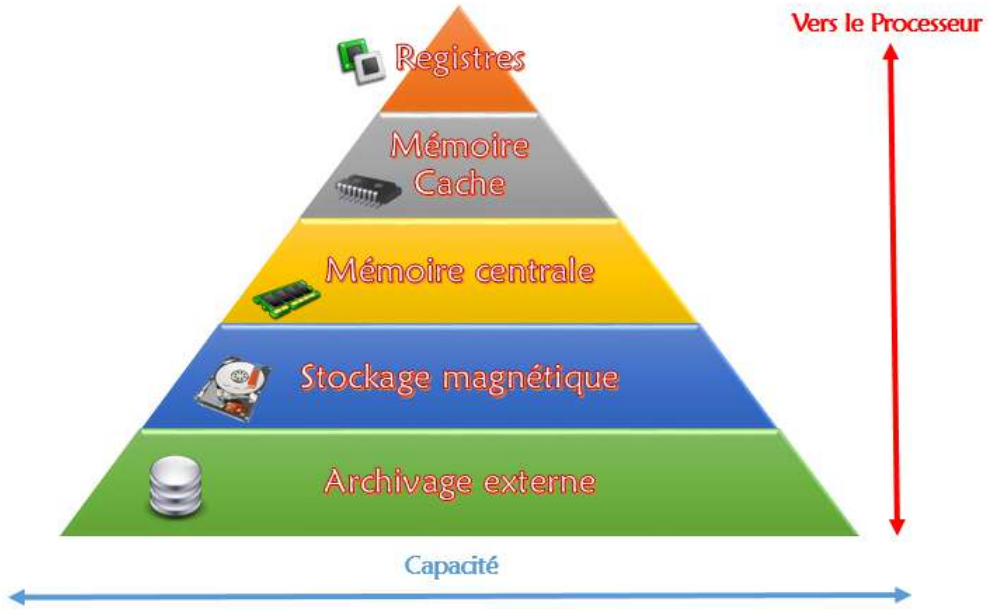


Figure 3.18: Hiérarchie des mémoires.

meilleures performances à l'ordinateur. Il existe plusieurs sortes de connecteurs :

- **Connecteur ISA (Industry Standard Architecture):** permettant de connecter des cartes ISA, les plus lentes fonctionnant en 16-bit, Figure 3.19.
- **Connecteur VLB (Vesa Local Bus):** Bus servant autrefois à connecter des cartes graphiques.
- **Connecteur PCI (Peripheral Component InterConnect) :** permettant de connecter des cartes PCI, beaucoup plus rapides que les cartes ISA et fonctionnant en 32-bit, Figure 3.20.
- **Connecteur AGP (Accelerated Graphic Port):** un connecteur rapide pour carte graphique, Figure 3.21.
- **Connecteur AMR (Audio Modem Riser):** ce type de connecteur permet de brancher des mini-cartes sur les PC en étant équipés.
- **Connecteur ATA ou SATA (Serial Advanced Technology Attachment):** permet de connecter à une carte mère tout périphérique compatible avec cette norme (mémoire de masse, lecteur de DVD, etc.). Elle spécifie notamment un format de transfert de données et un format de câble, Figure 3.22.

3.4.4 Le bus système

On appelle bus, le canal permettant de transférer des données entre deux éléments, Figure 3.23. Le bus système est le canal (pistes de la carte-mère) reliant le microprocesseur à la mémoire vive du système. Un bus est caractérisé par sa largeur, c'est-à-dire le nombre de bits pouvant etre simultanément transmis, et par sa fréquence, c'est-à-dire la cadence à laquelle les paquets de bits peuvent etre transmis. Des caractéristiques du bus système dépendent donc les caractéristiques générales du système. La fréquence du microprocesseur est égale à la fréquence du bus système multiplié par un facteur. Ainsi un PC tournant à 400 Mhz sera plus rapide s'il est basé sur un bus systéme cadencè à 133 Mhz (3 x 133 Mhz) que si la carte-mère a un bus dont la fréquence est 100 Mhz (la fréquence du processeur étant alors égale à 4 x 100 Mhz).

Hiérarchie des mémoires.

Figure 3.19: Bus ISA

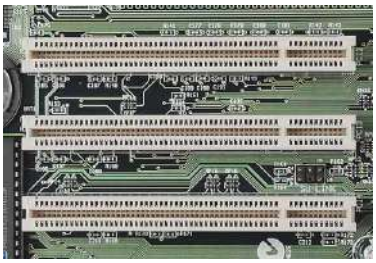


Figure 3.20: Bus PCI

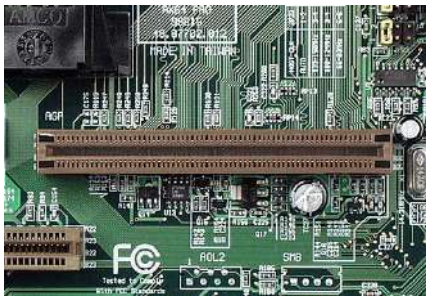


Figure 3.21: Bus AGP

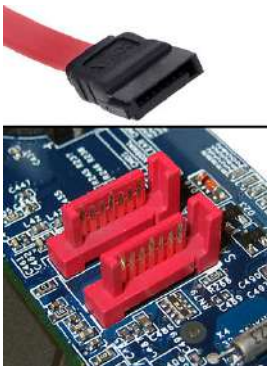


Figure 3.22: Bus SATA

Les bus systèmes.

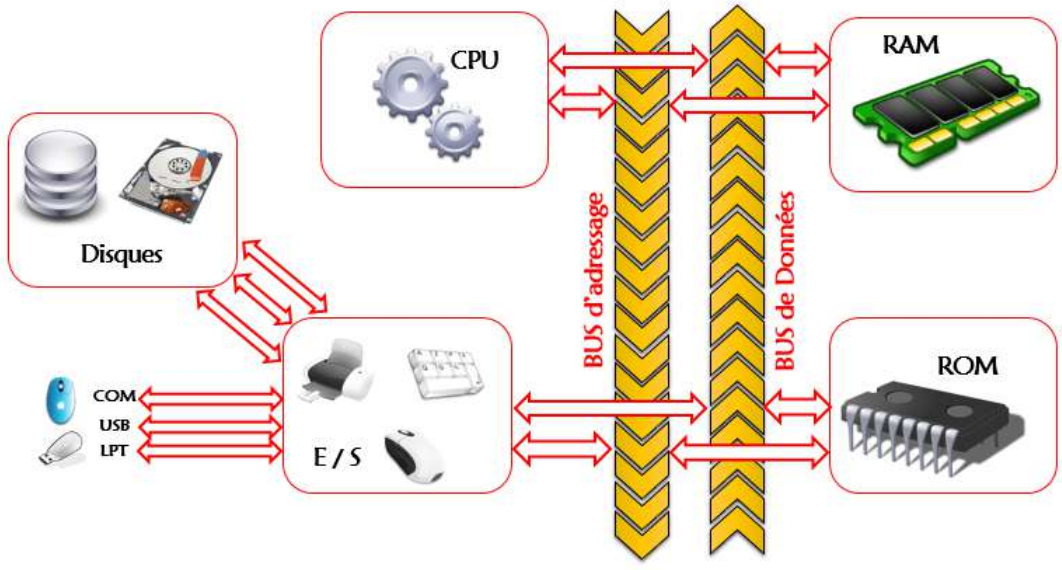


Figure 3.23: Les bus systèmes.

3.5 Le processeur (CPU, Central Process Unit)

Le processeur (CPU: Central Processing Unit) est le **Coeur** de l'ordinateur. C'est lui qui réalise les opérations. C'est est un circuit électronique cadencée au rythme d'une horloge interne, c'est-à-dire un élément qui envoie des impulsions (top). A chaque top d'horloge les éléments de l'ordinateur accomplissent une action. La vitesse de cette horloge s'exprime en Mégahertz. A chaque top d'horloge (instructions simples), le processeur :

- Lit l'instruction à exécuter en mémoire ;
- Effectue l'instruction ;
- Passe à l'instruction suivante.

3.5.1 A quoi ressemble une instruction ?

Les instructions (opération que le processeur doit accomplir) sont stockées dans la mémoire principale. Une instruction est composée de deux champs :

- **Le code opération:** C'est l'action que le processeur doit accomplir ;
- **Le code opérande:** C'est les paramètres de l'action. Le code opérande dépend de l'opération, cela peut etre une donnée ou bien une adresse d'un emplacement mémoire. Une instruction peut etre codée sur un nombre d'octets variant de 1 à 4 suivant le type de données.

3.5.2 Composition d'un processeur)

Le processeur est constitué de :

[Unité de traitement:] Une UAL (unité arithmétique et logique), d'un registre de données (mémoire destinée aux données récentes ou imminentes), et d'un accumulateur (l'espace de travail). Figure 3.24.

Schéma d'une unité de traitement.

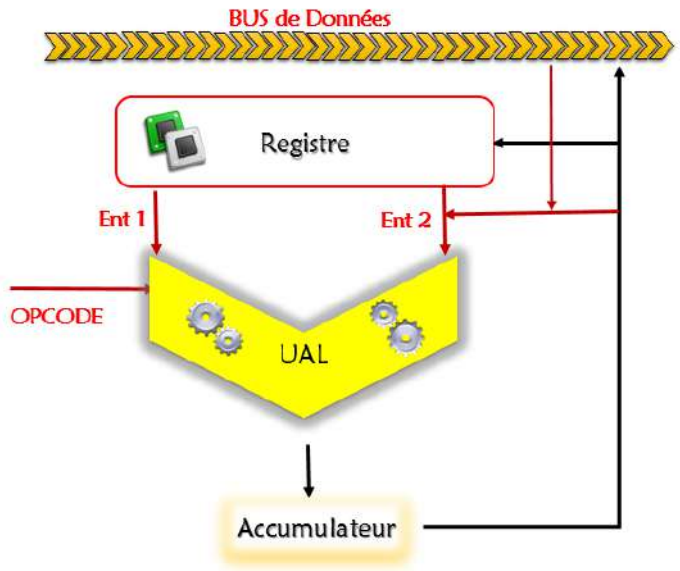


Figure 3.24: Schéma d'une unité de traitement.

- L'UAL peut traiter simultanément des données situées à plusieurs adresses. La donnée nécessaire aux calculs étant alors stockée dans l'accumulateur. Cela nécessite de décomposer un peu plus les opérations élémentaires .
- Le code de l'opération (OPCODE) correspond à une succession de bits indiquant la nature de l'opération à effectuer sur l'entrée (addition, multiplication et différents tests booléens.)
- Une instruction élémentaire arrive à l'unité de traitement sous la forme d'un code (une succession de bits 0 et 1), donnant d'une part le code de l'opération (OPCODE), d'autre part les informations nécessaires pour trouver les entrées auquelles appliquer cette opération (certains opérations demandent 2 adresses, d'autres une seule adresse et la donnée d'une valeur « immédiate »). Ce codage des instructions diffère également d'un processeur à l'autre.
- L'UAL est actuellement couplée avec une unité de calcul en virgule flottante, permettant le calcul sur les réels. Un processeur peut avoir plusieurs processeurs travaillant en parallèle (donc plusieurs UAL), afin d'augmenter la rapidité de traitement de l'ordinateur ;

[Unité de contrôle:] Unité qui décode les instructions d'un programme, les transcrit en une succession d'instructions élémentaires compréhensibles par l'unité de traitement. La synchronisation se fait grâce à l'horloge : les signaux d'horloge (impulsions électriques) sont envoyés de façon régulière, et permettent de cadencer les différentes actions. De façon schématique, la réception d'un signal d'horloge par un composant marque l'accomplissement d'une étape de la tâche qu'il a à faire. Les différentes étapes se succèdent donc au rythme des signaux d'horloge. Une unité de contrôle peut être schématisée à la manière de la Figure 3.25.

Schéma d'une unité de contrôle.

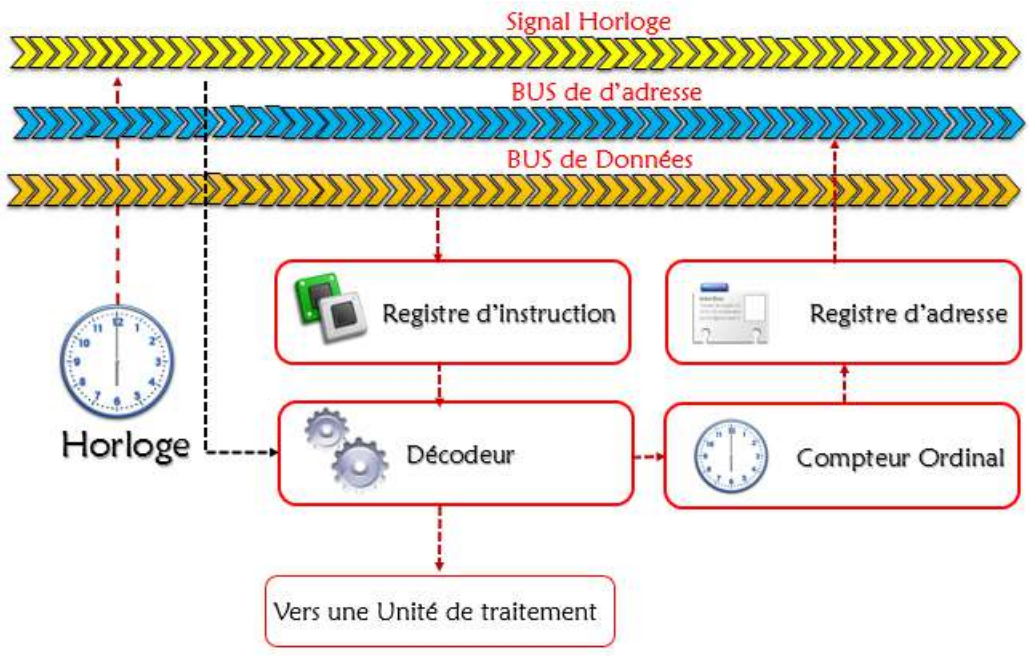


Figure 3.25: Schéma d'une unité de contrôle.

- Le registre d'instruction contient l'instruction (non élémentaire) en cours, le compteur ordinal contient ce qu'il faut pour permettre de trouver l'adresse de l'instruction suiv-

ante, et le registre d'adresse contient l'adresse de l'instruction suivante.

- Le décodeur décode l'instruction contenue dans le registre d'adresse et la traduit en une succession d'instructions élémentaires envoyées à l'unité de traitement au rythme de l'horloge. Lorsque l'instruction est entièrement traitée, l'unité va chercher l'instruction suivante, dont l'adresse est stockée dans le registre d'adresse, et la stocke dans le registre d'instruction. Le décodeur actualise le compteur ordinal puis l'adresse de l'instruction suivante.

[Les registres:] Lorsque le processeur traite des donnés (des instructions) le processeur stocke temporairement les données dans de petites mémoires (qui ont l'avantage d'etre trés rapides) que l'on appelle registres. Suivant le type de processeur le nombre de registres peut varier entre une dizaine et plusieurs centaines. Les registres les plus importants sont :

- **Le registre accumulateur :** il permet de stocker les résultats des opérations arithmétiques et logiques ;
- **Le registre tampon :** il permet de stocker temporairement une des opérandes ;
- **Le registre detat :** il permet de stocker les indicateurs ;
- **Le registre instruction :** il contient l'instruction en cours de traitement ;
- **Le compteur ordinal :** il contient l'adresse de la prochaine instruction à traiter ;
- **Le registre tampon :** il permet de stocker temporairement une donnée provenant de la mémoire.

[Les signaux de commande:] sont des signaux électriques qui permettent au processeur de communiquer avec le reste du système (le signal Read/Write - lecture/écriture - permet notamment de signaler à la mémoire qu'il désire lire ou écrire une information.

Lorsque tous les éléments d'un processeur sont regroupés sur une meme puce, on parle alors de microprocesseur, Figure 3.26.

Processeur Intel.

Figure 3.26: Processeur Intel.

3.5.3 Puissance de calcul d'un processeur

La puissance d'un processeur est calculée en FLOPS (Floating Point Operation Per Second). Il s'agit du nombre d'opérations capable de faire en une seconde sur le format flottant (réels). Pour donner des ordres de grandeur, voici l'évolution de la puissance sur quelques années :

1964 : 1 MégaFLOPS (10^6).
1997 : 1 TéraFLOPS (10^{12}).
2008 : 1 PétaFLOPS (10^{15}).

R On estime que **l'ExaFLOPS** 10^{18} devrait être atteint vers 2020. Évidemment, ces puissances correspondent aux super-calculateurs. La puissance des ordinateurs personnels est bien inférieure. Par exemple, en 2013, un ordinateur personnel était muni en moyenne d'un processeur de 200 GigaFLOPS, ce qui est comparable à la puissance des super-calculateurs de 1995.

3.6 Le déroulement d'un programme

Pour l'utilisateur une session de travail sur un ordinateur se traduit par les activités suivantes :

- Entrée de commandes décrivant les travaux par l'intermédiaire d'un clavier, d'une souris;
- Exécution des travaux demandés (l'utilisateur attend) ;
- Sortie des résultats sur un écran, une imprimante;
- Mémorisation éventuelle des résultats sur un support magnétique (disque dur, disque USB).

Or, pour la machine, il n'y a pas de différence fondamentale entre les quatre phases décrites ci-dessus. Elles correspondent en effet tour à tour à l'exécution :

- Un programme de gestion d'un dialogue utilisant l'écran et/ou le clavier,
- Un programme effectuant la tâche désirée,
- Un programme assurant la gestion de l'écran, le tracé de courbes, la sortie vocale,
- Un programme réalisant l'écriture en mémoire de masse.

Les opérations de base exécutées par l'ordinateur sont appelées instructions. La description par l'utilisateur des travaux qu'il veut effectuer à l'aide de ces instructions sera désignée par **programme**. La logique de fonctionnement est extrémement simple. Une fois mis sous tension l'ordinateur exécute un cycle "lecture-exécution". L'opération de lecture consiste à aller chercher en mémoire une instruction que l'unite de commande reconnait - décode - et qui sera exécutée par l'unité de traitement. L'exécution, Figure 3.27 donne lieu à un traitement en local (1), une écriture ou lecture en mémoire (2) ou une écriture ou lecture sur une unité d'échange (3). L'unité de commande engendre tous les signaux impliqués dans le deroulement du cycle.

3.7 Conclusion et Perspectives

Depuis l'apparition du premier ordinateur jusqu à aujourd' hui, c'est à dire depuis 1950 l'ordinateur s'est forgé une place dans notre société. Aujourd'hui l'ordinateur devient un outil indispensable, c'est pourquoi de nombreux scientifiques travaillent sans relache pour améliorer sa conception, ses composants. L'informatique a énormément évoluée . Les ordinateurs sont devenus de vrai bêtes de calcul . L'ordinateur est devenu aujourd'hui un outil indispensable en constante progression point de vue performance.

Accés à une instruction.

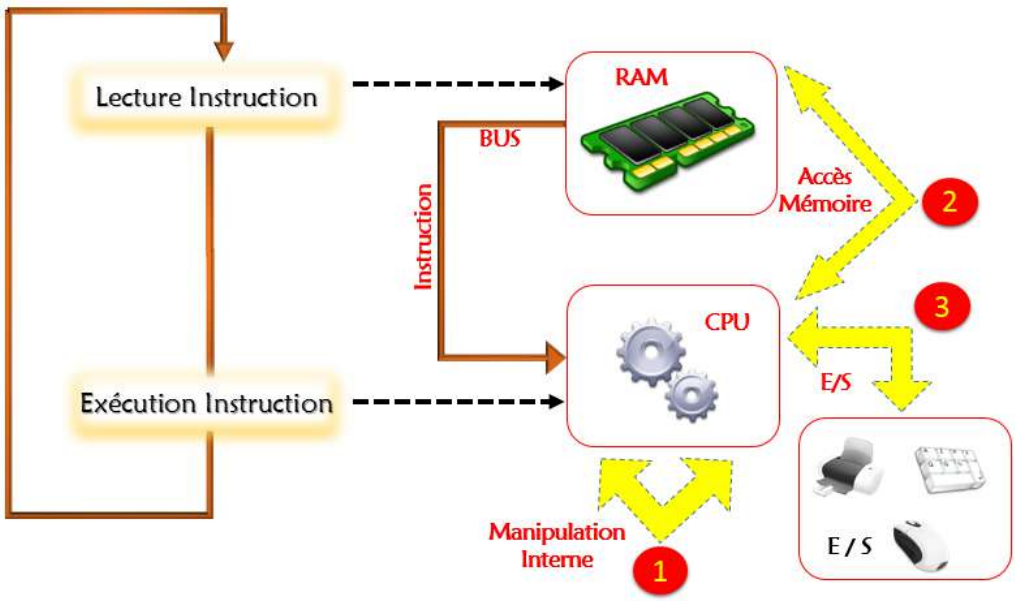


Figure 3.27: Accés à une instruction..

II Representation et Codage de l'information

4. Introduction

Dans la vie de tous jours, nous avons pris l'habitude depuis notre enfance de représenter les nombres en utilisant dix symboles différents, à savoir les chiffres suivants: 0 1 2 3 4 5 6 7 8 9. Ce système est appelée le système **Décimal** (Déci signifie Dix), pour la petite histoire, on a utilisé un système base 10 car nos ancêtres ont commencé à compter sur leurs 10 doigts.

les ordinateurs qui fonctionnent, utilisent les propriétés de l'électricité, ne connaissent que les 1 ou les 0. C'est la raison pour laquelle, dès qu'on aborde le monde des systèmes numériques, il est nécessaire pour leur bonne compréhension de connaître certaines bases de numération qui utilisent un nombre de symboles distincts, par exemple le système binaire (bi: deux), le système octal (oct: huit), le système hexadécimal (hexa: seize). En fait, on peut utiliser n'importe quel nombre de symboles différents (pas nécessairement des chiffres) dans un système de numération; ce nombre de symboles distincts est appelé **la base du système de numération**.

Les informations traitées par un ordinateur peuvent être de différents types (texte, nombres, images, son, vidéos, etc.) mais elles sont toujours représentées et manipulées par l'ordinateur sous forme numérique (digitale). En fait, toute information sera traitée comme une suite de 0 et de 1. L'unité d'information est donc les chiffres binaires (0 et 1) que l'on appelle bit (pour binary digit : chiffre binaire). On utilise la représentation binaire car elle est facile à réaliser techniquement à l'aide de bistables (système à deux états réalisés à l'aide de transistors). Le codage d'une information consiste à établir une correspondance entre la représentation externe (habituelle) de l'information (texte, image, ... etc), et sa représentation interne dans la machine, qui est toujours une suite de bits. Le chapitre expose quelques rappels sur les bases de numération , le passage d'une représentation à une autre, puis nous présentont les normes de codage des entierss, des réels et des caracteres.

5. Representation de l'information

5.1 Introduction

Les informations traitées par l'ordinateur sont de différents types (nombres, instructions, images, sons, etc...) mais pour des raisons technologiques (liées à la réalisation de l'ordinateur), elles sont représentées sous la forme binaire. Une information complexe tels qu'une lettre se ramène à un ensemble de bits. Le codage d'une information revient à établir une correspondance entre la représentation externe de l'information (exemple : lettre A) et sa représentation interne sous forme de suite de bits. Pour les informations manipulées au niveau de l'ordinateur, on distingue les instructions et les données.

[Instructions] : Ecrites en langage machine, les instructions représentent les opérations que l'ordinateur est capable d'effectuer.

[Données] : Ce sont les opérandes sur lesquels portent les instructions ou produites par celle-ci. Une addition s'applique à deux opérandes donnant un résultat qui est leur somme.

5.2 Représentation des instructions

Une instruction machine renseigne le processeur sur la nature de l'opération à exécuter et sur les données qui vont participer à l'exécution de l'opération. Représentée sur une taille fixe, elle est composée de plusieurs parties de taille fixe (champs) qui sont les suivantes :

- Le code de l'opération à effectuer ;
- Les opérandes impliqués dans l'opération.

5.2.1 Code opération

Il représente un nombre fixe de bits. Chacune parmi les instructions que l'ordinateur est capable d'exécuter se voit attribuée une suite binaire différente des autres (d'ou l'appellation code opération). On distingue les données numériques pouvant être l'objet d'une opération arithmétique et les données non numériques comme par exemple les données constituant un texte.

5.2.2 Les opérandes

Ils se partagent généralement d'une façon équitable le reste de l'instruction (taille de l'instruction – taille code opération). Elles représentent en binaire, les valeurs des données concernées par cette opération, ou bien leur emplacement dans la mémoire (Adresse). D'une instruction à une autre et dans le même ordinateur, le nombre d'opérandes peut changer. Par exemple, l'opération d'addition demande deux opérandes et l'opération opposé d'un nombre demande une seule opérande. Le nombre maximum d'opérandes autorisé peut changer, d'un ordinateur à un autre, Table 5.1. Selon le nombre d'opérandes, on distingue les ordinateurs avec des :

- Instructions à un seul opérande appelées aussi à une seule adresse (car généralement l'opérande est une adresse).
- Instructions à deux adresses.
- Instructions à trois adresses.

Code opération	Opérande 1	Opérande 2
M bits	N bits	N bits

Table 5.1: Cas d'un ordinateur avec des instructions à 2 adresses.

5.3 Representation des données

5.3.1 Données non numériques:

Elles correspondent aux caractères alphanumériques : A, B…Z, a,b…z, 0, 1…9 et aux caractères spéciaux : ?, !. Parmi les systèmes de codage les plus connus, on peut citer :

- **BCD** (Binary Code Decimal), où un caractère est codé sur 6 bits.
- **ASCII** (Americain Standard Code for Information Interchange), sur 7 bits;
- **EBCDIC** (Extended Binary Coded Decimal Internal Code), sur 8 bits;
- **UNICODE**, sur 16 bits ;
- **ISO/IEC 10646** (International Standards Organization/International Electronic Commission), sur 32 bits.

Les deux derniers codes ont été créés récemment (début des années 90). En effet, 128 ou 256 valeurs ne suffisent pas pour représenter l'ensemble de tous les caractères de toutes les langues de la planète. L'opération de passage de la représentation d'une donnée dans un code à une autre dans un deuxième code, s'appelle transcodage.

5.3.2 Données numériques :

Avant qu'ils subissent des traitements, elles vont subir une opération de codage qui permet de déduire leur représentation interne. Ensuite, les opérations demandées sont de la représentation interne vers la représentation externe) est nécessaire pour les restituer vers l'extérieur. Les données numériques sont de différents types :

- Nombres entiers positifs ou nul : 0, 1, 1254.
- Nombre entiers négatifs :-1, -1245.
- Nombre fractionnaires : 3.1457, -0.514.
- Nombre en notation scientifique : 1.510^7.

6. Systémes de numèration

6.1 Introduction

Les ordinateurs calculent en binaire. Il s'agit de la base 2 dans laquelle seuls les symboles 0 et 1 sont utilisés pour écrire les nombres. Ceux ci vont être stockés dans des cases ayant une taille fixée. L'ordinateur est donc limité dans ses capacités à exprimer n'importe quel nombre. Il existe cependant d'autres formes de numération qui fonctionnent en utilisant un nombre de symboles distincts, par exemple le système octal (oct: huit), le système hexadécimal (hexa: seize). En fait, on peut utiliser n'importe quel nombre de symboles différents (pas nécessairement des chiffres) dans un système de numération; ce nombre de symboles distincts est appelé **la base du système de numération**. La Table 6.1 montre les symboles utilisés des les principaux systèmes rencontrés.

Système	Base	Symboles utilisés
Binaire	2	0 1
Ternaire	3	0 1 2
Octal	8	0 1 2 3 4 5 6 7
Décimal	10	0 1 2 3 4 5 6 7 8 9
Hexadécimal	16	0 1 2 3 4 5 6 7 8 9 A B C D E F

Table 6.1: Bases

6.2 Bases et exposants

Lorsqu'on utilise le système décimal: on énumère tous les symboles possibles, 0, 1, 2, jusqu'à 9. Une fois la liste de symboles épuisée, on ajoute une position à gauche pour former le nombre suivant: 10. La notion de dizaine, centaines, milliers exprime en fait l'exposant que prend la base

10 donnant le poids de la deuxième, troisième position du chiffre dans la représentation du nombre. En binaire, le nombre de symboles se limite à deux. Une fois les deux symboles épuisés, il faut déjà ajouter une position à gauche. Ainsi, après 0 et 1. Une fois les possibilités de deux positions épuisées, une troisième position est ajoutée. On peut indiquer la base d'un nombre en écrivant l'indice correspondant à droite du nombre :

■ **Example 6.1** $(734)_8$ signifie que 734 est la représentation d'un nombre exprimé en base 8. ■

Le Table 6.2 montre la correspondance entre divers systèmes de bases différentes, .

Binaire	Décimal	Octal	Héxadécimal
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	8	10	8
1001	9	11	9
1010	10	12	A
1011	11	13	B
1100	12	14	C
1101	13	15	D
1110	14	16	E
11111	15	17	F

Table 6.2: Correspondance entre divers systèmes de bases

6.3 Rang d'une base

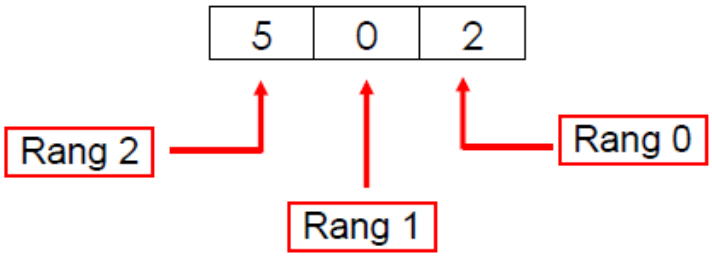
La position des chiffres à une grande importance. Les chiffres les moins significatifs se situent à droite du nombre, et leur importance augmente au fur et à mesure du déplacement vers la gauche. En effet, dans le nombre 502, le 5 à une plus grande importance que le 2. En réalité, chaque chiffre que l'on appelle DIGIT, à une valeur qui dépend de son rang.

R On élève la BASE utilisée à la puissance du rang du DIGIT, on multiplie par le chiffre du rang et on additionne l'ensemble. Cette règle est applicable dans toutes les bases.

■ **Example 6.2** Soit le nombre 502:

■

appliquable pour n'importe quel système de numérotation.



$$502 = 5 \times 10^2 + 0 \times 10^1 + 2 \times 10^0$$

6.4 La représentation polynomiale d'un nombre

Nous manipulons la plupart du temps la base universelle décimale. La numération qui peut prendre plusieurs formes, parmi lesquelles on trouve la théorie des ensembles et la représentation polynomiale. La représentation polynomiale d'un nombre est sa représentation sous la forme suivante, où b est appelée **la base** :

$$N_{(b)} = S(a_{n-1}b^{n-1} + a_{n-2}b^{n-2} + ... + a_2b^2 + a_1b + a_0 + a_{-1}b^{-1} + a_{-2}b^{-2} ... + a_{-m}b^{-m})$$

- S est le **signe** du nombre.
- a_i est le symbole de **rang** i,$a_i \in Net0 <= a_i < b$.
- a_n est le symbole de **poids le plus fort** (MSB : Most Significant Bit si b = 2), et a_m est le symbole de **poids le plus faible** (LSB : Least Significant Bit si b = 2).

■ **Example 6.3** Prenons l'exemple du nombre décimale 1986:

$$(1986)_{10} = (1x10^3) + (9x10^2) + (8x10^1) + (6x10^0).$$

■

Autrement dit, on multiplie chaque symbole avec les puissances successives de la base B.

R L'écriture d'un nombre est ambiguë si la base n'est pas explicite. Ainsi, la même écriture 101 peut valoir cent un si on lit le nombre en décimal, mais aussi cinq en binaire. Pour éviter cela, il est courant d'indicer le nombre par la base.

6.4.1 Le système binaire

Le système **décimal** est malheureusement difficile à adapter aux mécanismes numériques, car il est difficile de concevoir du matériel électronique fonctionnant sur dix plages de tensions différentes. On lui préférera donc le système binaire :

- Base B=2 ;
- 2 symboles : 0, 1 appelés « éléments binaires » ou « bits » (bit=Binary digIT) ;

- Le système binaire est pondéré par 2 : les poids sont les puissances de 2 ;
- Un ensemble de 8 bits est appelé « Octet » (ou byte).

2^6	2^5	2^4	2^3	2^2	2^1	2^0	,	2^{-1}	2^{-2}	2^{-3}
1	0	1	1	0	0	1	,	0	1	1

Les différentes puissances de 2 sont :

2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
1	2	4	8	16	32	64	128	256	512	1024

6.4.2 Le système octal

Le système octal utilise un système de numération ayant comme base 8 (octal : latin octo=huit) et utilise donc 8 symboles : 0, 1, 2, 3, 4, 5, 6, 7, un nombre exprimé en base 8 pourra se présenter de la manière suivante par exemple : $(745)_8$. L'intérêt de ce système est que la base 8 est une puissance de 2 $(8 = 2^3)$, donc les poids sont aussi des puissances de 2. Chaque symbole de la base 8 est exprimé sur 3 e.b. : $(a_i)_8 = b_{i2}b_{i1}b_{i0}$

■ **Example 6.4** Prenons l'exemple du nombre Octal 52,3:

$$(52,3)_8 = 101010,011$$

■

6.4.3 Le système Décimal

Le système décimal est celui dans lequel nous avons le plus l'habitude d'écrire. Chaque chiffre peut avoir 10 valeurs différentes : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, de ce fait, le système décimal a pour base 10. La valeur de chaque chiffre dépend de sa position, c'est-à-dire que c'est une numération positionnelle : la position la plus à droite exprime les unités, la position suivante : les dizaines, ensuite les centaines. Par exemple, si on décompose le nombre 9745, nous aurons :

■ **Example 6.5** Prenons l'exemple du nombre Décimal 9745:

$$9745 = 9 * 1000 + 7 * 100 + 4 * 10 + 5 * 1.$$
$$9745 = 9 * 10^3 + 7 * 10^2 + 4 * 10^1 + 5 * 10^0.$$

■

6.4.4 Le système hexadécimal

Le système hexadécimal utilise les 16 symboles suivant : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E,F. De ce fait, le système a pour base 16. chaque symbole est exprimé en binaire sur 4 bits $(a_i)_16 = b_{i3}b_{i2}b_{i1}b_{i0}$;

■ **Example 6.6** Prenons l'exemple du nombre hexadécimal $F3D,2$:

$$(\text{F3D, 2})_{16} = 111101111101,0010$$

■

6.5 Conversion d'un système de numération à une autre

L'ordinateur ne sait calculer qu'en base 2. Malheureusement, l'écriture binaire n'est ni pratique (à cause de la taille des écritures), ni intuitive (le cerveau humain ne calcule facilement qu'en base 10). On doit donc souvent effectuer des changements de base entre la base 2 et les bases 8, 10 ou 16. Le changement de base le plus simple est le passage entre la base 2 et les bases 8 ou 16. En effet, les valeurs 8 et 16 étant des puissances de 2 ($8 = 2^3; 16 = 2^4$), chaque bloc de 3 ou 4 bits correspond à un symbole octal ou hexadécimal.

Les conversions les plus utilisées sont les suivantes:

- Base b vers base 10.
- Base 10 vers base b.
- Base 2 vers base 2^n (8 ou 16).
- Base 2^n (8 ou 16) vers base 2.

6.5.1 Base b vers base 10

Pour convertir un nombre d'une base b vers la base décimale, on utilise la méthode dite des additions qui consiste à utiliser la représentation du nombre sous forme polynomiale.

■ **Example 6.7** Conversion du nombre binaire entier $N_2 = 11010011_2$ en base 10.

$$N = 1*2^7 + 1*2^6 + 0*2^5 + 1*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = 211_{10}.$$

■

■ **Example 6.8** Conversion du nombre binaire fractionnaire $N_2 = 110011,1001_2$ en base 10.

$$N = 1*2^5 + 1*2^4 + 1*2^1 + 1*2^0 + 1*2^1 + 1*2^4 = 51,5625_{10}.$$

■

■ **Example 6.9** Conversion du nombre octal entier $N_8 = 4513_8$ en base 10.

$$N = 4*8^3 + 5*8^2 + 1*8^1 + 3*8^0 = 2379_{10}.$$

■

■ **Example 6.10** Conversion du nombre hexadécimal fractionnaire $N_{16} = 1B20,8_{16}$ en base 10.

$$N = 1*16^3 + 11*16^2 + 2*16^1 + 816^1 = 6944,5_{10}.$$

■

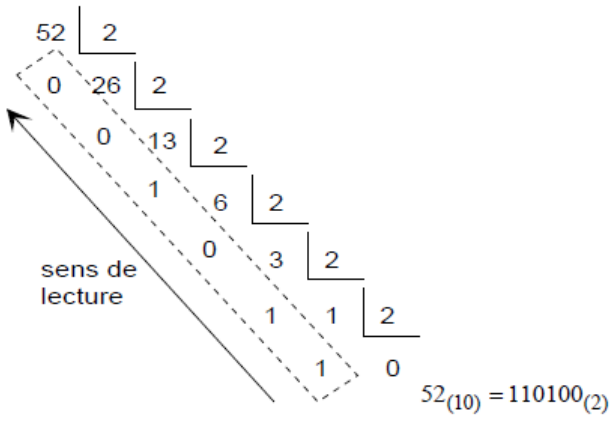
6.5.2 Base 10 vers base b

Nombres entiers

Pour effectuer une conversion d'un entier décimal dans une autre base on applique **la méthode des divisions successives** : on effectue des divisions successives du nombre par cette base, les restes successifs forment alors le nombre converti. A titre d'exemple, dans le cas d'une conversion d'un nombre décimal en son équivalent binaire, on réalisera des divisions successives par 2. Les restes de ces divisions formeront le nombre converti dans la base 2.

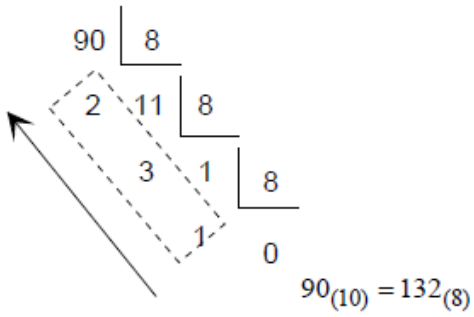
■ **Example 6.11** conversion de N_{10}= 52 en base 2. ■

Concersion du Décimal vers le Binaire.



■ **Example 6.12** conversion de N_{10}= 90 en base 8. ■

Concersion du Décimal vers le Binaire.

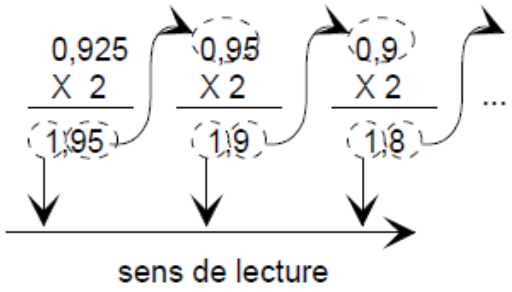


Nombres fractionnaires

Pour convertir un nombre fractionnaire de la base 10 vers une autre base, il faut procéder en deux étapes. La partie entière du nombre est convertie comme indiqué précédemment ; la partie fractionnaire du nombre est convertie par multiplications successives : on multiplie successivement la partie fractionnaire par la base cible, en retenant les parties entières qui apparaissent au fur et à mesure.

■ **Example 6.13** Conversion de N_{10}= 12,925 en base 2.
Partie entière : $12_{10} = 1100_2$.

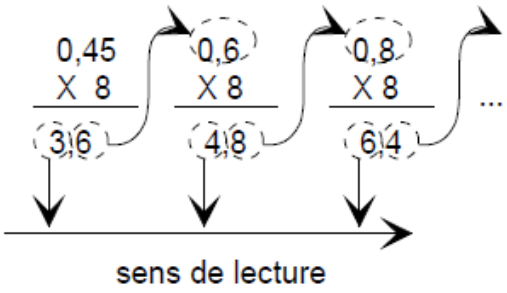
Nombre fractionnaire vers le Binaire.



$0{,}925_{10} = 0{,}111_2$.
Finalement $12{,}925_{10} = 1100{,}111_2$. ■

■ **Example 6.14** conversion de N_{10}= 0,45 en base 8.
Partie entière : $0_{10} = 0_2$.

Nombre fractionnaire vers l'Octal.



$0{,}45_{10} = 0{,}346_8$. ■

R Il est visible sur les deux exemples précédents que la conversion peut ne pas se terminer et que l'on obtient, en s'arrêtant à un nombre fini de positions une approximation de la représentation du nombre.

6.5.3 Base 2 vers base 8 et 16

L'utilisation des bases (8 et 16) permet de réduire le nombre de symboles à écrire tout en conservant la possibilité de conversion instantanée en binaire. Pour convertir un nombre de la base 2 vers la base 8 et 16, il suffit de regrouper les bits par groupes de n **(3 pour la base octale et 4 pour la base hexadécimale)**, et de remplacer chacun de ces groupes par le symbole correspondant dans la base d'arrivée. Pour la partie entière, le regroupement part du bit de poids le plus faible, et pour la partie fractionnaire, du bit de poids le plus fort (de la virgule). Lorsqu'un groupe est **incomplet**, on le complète avec des 0.

■ **Example 6.15** conversion de $N_2 = 1100111010101$ en base 8 puis 16. ■

Conversion du Binaire vers base 8 et 16.

$$\textit{Base 8}: N = 1\,100\,111\,010\,101_{(2)} = \underbrace{001}_{1_{(8)}}\ \underbrace{100}_{4_{(8)}}\ \underbrace{111}_{7_{(8)}}\ \underbrace{010}_{2_{(8)}}\ \underbrace{101}_{5_{(8)}} = 14725_{(8)}.$$

$$\textit{Base 16}: N = 1\,1001\,1101\,0101_{(2)} = \underbrace{0001}_{1_{(16)}}\ \underbrace{1001}_{9_{(16)}}\ \underbrace{1101}_{D_{(16)}}\ \underbrace{0101}_{5_{(16)}} = 19D5_{(16)}.$$

■ **Example 6.16** conversion de N_2 =110100110,101101 en base 8 puis 16. ■

Conversion du Binaire vers base 8 et 16.

$$\textit{Base 8}: N = 110\,100\,110{,}101\,101_{(2)} = \underbrace{110}_{6_{(8)}}\ \underbrace{100}_{4_{(8)}}\ \underbrace{110}_{6_{(8)}}{,}\underbrace{101}_{5_{(8)}}\ \underbrace{101}_{5_{(8)}} = 646{,}55_{(8)}$$

$$\textit{Base 16}: N = 1\,1010\,0110{,}1011\,01_{(2)} = \underbrace{0001}_{1_{(16)}}\ \underbrace{1010}_{A_{(16)}}\ \underbrace{0110}_{6_{(16)}}{,}\underbrace{1011}_{B_{(16)}}\ \underbrace{0100}_{4_{(16)}} = 1A6{,}B4_{(16)}$$

6.5.4 Base 8 et 16 vers base 2

Pour la conversion inverse, il suffit de développer chaque symbole de la représentation dans la base 2^n sur n bits.

■ **Example 6.17** conversion de N_2 =110100110,101101 en base 8 puis 16. ■

Conversion des bases 8 et 16 vers le Binaire.

$$4A1_{(16)} = \underbrace{0100}_{4 \text{ en base 2}}\ \underbrace{1010}_{A \text{ en base 2}}\ \underbrace{0001}_{1 \text{ en base 2}} = 010010100001_{(2)}.$$

$$273{,}15_{(8)} = 010\ 111\ 011{,}001\ 101 = 10111011{,}001101_{(2)}$$

6.6 Représentation numérique de l'information

6.7 Opérations Arithmétique binaire

Nous allons étudier les opérations arithmétiques classiques. Elles se réalisent de la même manière qu'en décimal.

6.7.1 Addition et soustraction

Le principe de l'addition est similaire à celui de l'addition décimale : on additionne symbole par symbole en partant des poids faibles, et en propageant éventuellement une retenue. Si le format des nombres est fixe, le résultat de l'addition peut donner lieu à un **dépassement de capacité**. La Table 6.3 d'addition binaire est la suivante :

A	B	C	Retenu (Carry)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 6.3: Table d'addition.

■ **Example 6.18** On effectue l'addition binaire bit à bit, de droite à gauche, en reportant les retenues, on peut vérifier le résultat en décimal : 43 + 110 = 153. ■

Exemple addition en binaire.

$$\begin{array}{cccccccc} & {\scriptstyle 1} & {\scriptstyle 1} & & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & \\ & & & 1 & 0 & 1 & 0 & 1 & 1 \\ + & & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ \hline & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{array}$$

La soustraction, en arithmétique binaire, est le plus souvent appliquée sur des nombres signés. Dans ce cas, cette opération se ramène dans tous les cas à une addition. Dans le cas où les nombres sont codés en **complément à 2**, l'addition de 2 nombres exprimés sur n bits fournit toujours un résultat correct, sauf dans le cas où le résultat n'est pas représentable sur les n bits. Il y a alors dépassement de capacité lorsque les deux opérandes sont de même signe et que le résultat est de signe opposé. La Table 10.13 de soustraction binaire est la suivante :

A	B	C	Retenu (Borrow)
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Table 6.4: Table de Soustraction.

■ **Example 6.19** On effectue la soustraction binaire bit à bit, de droite à gauche, en reportant les retenues, comme dans l'exemple suivant : ■

Dans le registre d'état d'un ordinateur, deux indicateurs viennent renseigner le programmeur (ou le système d'exploitation) sur la validité des résultats obtenus : la retenue (carry C) et le débordement (overflow OVF). L'indicateur C signale la présence d'une retenue au niveau des poids

Exemple de Soustraction en binaire.

$$\begin{array}{cccccccc} & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ - & & {\scriptstyle 1}\,{\scriptstyle 1} & 1 & 1 & {\scriptstyle 1}0 & 1 & 0 \\ \hline & 1 & 0 & 1 & 1 & 0 & 1 & 1 \end{array}$$

forts; l'indicateur OVF indique que le résultat de l'opération n'est pas représentable dans le système du complément à 2 sur le format défini au départ.

■ **Example 6.20** Nous allons illustrer le positionnement de la retenue et du débordement par quatre exemples: ■

Exemple de retenue et débordement.

0000 0110 (+6)	0111 1111 (+127)	0000 0100 (+4)	0000 0100 (+4)
+ 0000 0100 (+4)	+ 0000 0001 (+1)	+ 1111 1110 (-2)	+ 1111 1100 (-4)
0000 1010 (+10)	1000 0000 (-128)	1 0000 0010 (+2)	1 0000 0000 (0)
OVF=0	OVF=1	OVF=0	OVF=0
C=0	C=0	C=1 ignoré	C=1 ignoré
résultat correct	résultat incorrect	résultat correct	résultat correct

6.7.2 Multiplication et division

La multiplication se fait en formant un produit partiel pour chaque digit du multiplieur (seul les bits non nuls donneront un résultat non nul). Lorsque le bit du multiplieur est nul, le produit partiel est nul, lorsqu'il vaut un, le produit partiel est constitué du multiplicande décalé du nombre de positions égal au poids du bit du multiplieur. La Table 6.5 de multiplication en binaire est très simple :

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

Table 6.5: Table de Multiplication.

■ **Example 6.21** Voici un exemple : ■

R Cette manière d'opérer s'implémente facilement car elle consiste à ne faire que des décalages et des additions, opérations classiques sur les processeurs. L'inconvénient de la multiplication est qu'elle allonge fortement les nombres : multiplier deux nombres de n bits peut donner un

Exemple de Multiplication en Binaire.

$$
\begin{array}{cccccccccccc}
 & & & & & & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\
\times & & & & & & & & 1 & 1 & 0 & 1 & 0 \\
\hline
 & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & & & & & & \\
 & & & & & 1 & 0 & 1 & 0 & 1 & 1 & 0 & \bullet \\
+ & & & 1 & 0 & 1 & 0 & 1 & 1 & 0 & \bullet & \bullet & \bullet \\
+ & & 1 & 0 & 1 & 0 & 1 & 1 & 0 & \bullet & \bullet & \bullet & \bullet \\
\hline
 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0
\end{array}
$$

résultat sur 2n bits. Il faut donc faire attention lorsque ces nombres sont stockés dans une case mémoire car le résultat, de taille double, peut ne plus tenir dans une case.

La division binaire s'effectue à l'aide de soustractions et de décalages, comme la division décimale, sauf que les digits du quotient ne peuvent être que 1 ou 0. Le bit du quotient est 1 si on peut soustraire le diviseur, sinon il est 0. La Table 6.6 de division binaire est la suivante :

A	B	C
0	0	Impossible
0	1	0
1	0	Impossible
1	1	0

Table 6.6: Table de Division.

■ **Example 6.22** Finalement, la division est une simple succession de soustractions comme en arithmétique en base 10, Voici un exemple : ■

Exemple de Division en Bianire.

$$
\begin{array}{cccc|cc}
1 & 1 & 0 & 1 & 1 & 0 \\
1 & 0 & & & \multicolumn{2}{c}{1\ 1\ 0} \\
\cline{1-2}
0 & 1 & 0 & & & \\
 & 1 & 0 & & & \\
\cline{2-3}
 & 0 & 0 & 1 & & \\
\end{array}
$$

6.8 Codage de l'information

6.8.1 Représentation des nombres signés :

Nous savons maintenant représenter en binaire des nombres entiers positifs. Mais comment faire pour travailler avec des nombres négatifs ? Dans l'arithmétique classique, ces nombres sont précédés d'un signe moins, mais c'est ici impossible car on ne peut mettre que **des bits dans une case mémoire**. Il faut donc trouver une écriture, une représentation des nombres négatifs, utilisant les bits disponibles.

Représentation « signe et valeur absolue

Dans cette représentation, on utilisae **le bit de poids fort** du nombre pour représenter un nombre négatif. Sur n bits, le bit de poids fort (aussi appelé « bit de signe ») indique le signe du nombre (1 pour un nombre négatif) et les n – 1 bits restants donnent la valeur absolue binaire du nombre. Les mots binaires 01011001 = 89 et 10110100 = –52 sont deux exemples de nombres écrits dans cette représentation sur 8 bits.

■ **Example 6.23** Par exemple, sur 4 bits, 1 bit sera réservé au signe et trois bits seront utilisés pour représenter les nombres en valeur absolue., Table 6.7 ■

Signe	Valeur			Valeur
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	0	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	-0
1	0	0	1	-1
1	0	1	0	-2
1	0	1	1	-3
1	1	0	0	-4
1	1	0	1	-5
1	1	1	0	-6
1	1	1	1	-7

Table 6.7: Sur 4 bits, 1 bit sera réservé au signe.

Comme 3 bits sont utilisés pour représenter la valeur absolue, on peut donc représenter 2^3 valeurs absolues différentes, soient les valeurs comprises entre 0 et 7. Le bit de signe nous permet d'affecter à chacune de ces valeurs un signe + ou -. Autrement dit, n-1 bits servent à représenter la valeur absolue, ce qui donne une étendue possible allant de -7 à +7 dans notre exemple.

$$-(2^{n-1}-1) \text{ à } (2^{n-1}-1)$$

R Ce codage comporte de nombreux inconvenients. D'abord, la présence de deux valeurs pour +0 (00000000) et -0 (10000000), ensuite, l'addition est compliquée : il faut examiner les signes, et faire une addition ou une soustraction selon les cas.

Les tables d'addition sont compliquées à cause du bit de signe qui doit être traité à part.

7		**0** 1 1 1		- 8		**1** 1 0 0 0		- 3		**1** 0 0 1 1
+		+		+		+		+		+
- 5		**1** 1 0 1		6		**0** 0 1 1 0		- 6		**1** 0 1 1 0
2		**0** 0 1 0		- 2		**1** 0 0 1 0		- 9		**1** 1 0 0 1

Représentation en complément à 1

Si x est un nombre positif, alors on donne sa representation binaire avec la restriction que le bit le plus a gauche doit valoir 0. Pour un nombre negatif x = -y, on inverse tous les bits de (y) (on remplace les 1 par des 0 et les 0 par des 1). Le bit le plus a gauche est donc 1. Avec n bits, on peut coder des entiers positifs et négatifs tel que :

$$-(2^{n-1}-1) \text{ à } (2^{n-1}-1)$$

■ **Example 6.24** L'addition est simple a realiser : Voici par exemple les additions 6 + (-4) = 2 et -5 + 3 = -2 : ■

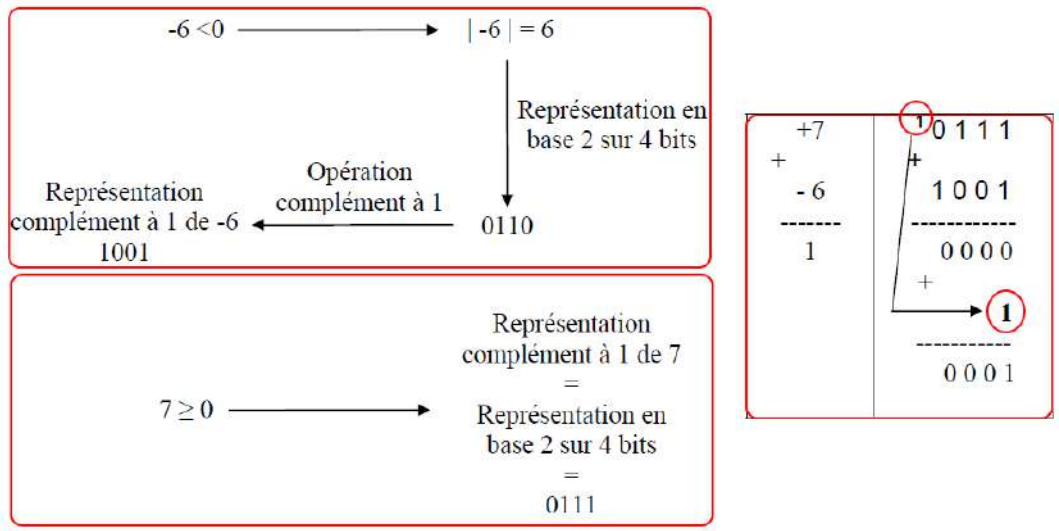
Addition en CA1.

	0	1	1	0		1	0	1	0
+	1	0	1	1	+	0	0	1	1
1 |	0	0	0	1	0—	1	1	0	1
|+				1	+				0
	0	0	1	0		1	1	0	1

R L'inconvénient de ce codage est qu'il y a deux représentations de 0 : par exemple sur 4 bits, 0000 et 1111. Mais elle a l'avantage suivant : L'addition devient simple. En effet, si une retenue est générée par le bit de signe alors elle doit être ajoutée au résultat obtenu.

■ **Example 6.25** Représentation de -6 et de 7 en complément à 1 sur 4 bits. ■

Représentation de -6 et de 7 en complément à 1 sur 4 bits.



Représentation en complément à 2

C'est la représentation standard sur les ordinateurs pour exprimer les nombres entiers négatifs. Quand on parle de représentation signée ou d'entiers signés, ces derniers sont toujours exprimés à l'aide de la représentation en complément à 2. Sur n bits, on exprime les nombres de l'intervalle $[\breve{}2^{n\breve{}1}, 2^{n\breve{}1\breve{}}1]$. On retrouve bien les 2^n nombres possibles. Un nombre positif est représenté de façon standard par son écriture binaire. On représente un nombre négatif en ajoutant 1 à son complément à 1 et en laissant tomber une éventuelle retenue finale.

Si l'on reprend l'exemple précédent, 89 (positif) s'écrit toujours 01011001, mais –52 (négatif) s'écrit maintenant 11001100. En effet, en binaire sur 8 bits, 52 s'écrit 00110100, ce qui donne un complément à 1 égal à 11001011 et un complément à 2 égal à 11001100. Dans le cas d'un nombre négatif, il est important d'indiquer sur combien de bits doit s'écrire le nombre car on ne rajoute pas des zéros en tête mais des uns (suite à l'inversion obtenue lors du calcul du complément à 1).

La phrase « représentez –20 en complément à 2 » n'a pas de sens si l'on ne précise pas le nombre de bits de la représentation. Cela peut tout aussi bien être 101100 sur 6 bits que 11101100 sur 8 bits ou 1111111111101100 sur 16 bits. De façon générale, sur n bits, le complément à 2 des nombres négatifs permet de représenter les nombres compris entre :

$$-(2^{n-1}) \text{ à } (2^{n-1}-1)$$

■ **Example 6.26** Par exemple, sur n=4 bits, Table 6.8

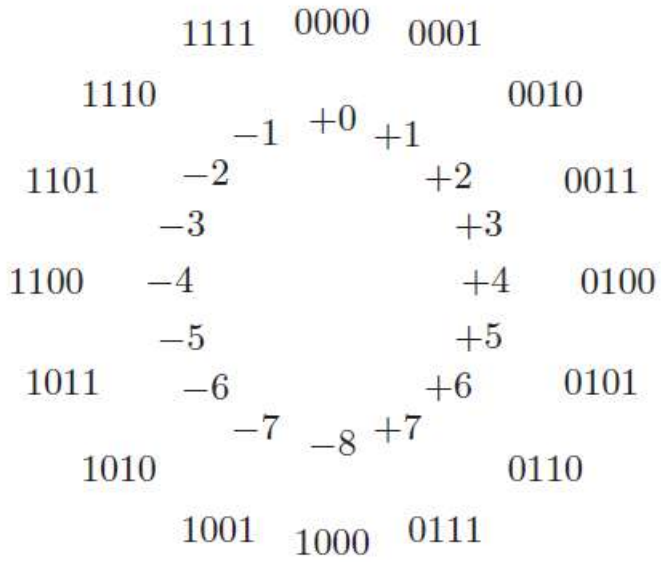
■ **Example 6.27** Ou bien toujours sur quatre bits, on obtient donc le codage suivant des entiers relatifs sur le cercle : ■

■

A_{10}	A	$\overline{A}$	$\overline{A}+1$	$-A_{10}$
7	0111	1000	1001	-7
6	0110	1001	1010	-6
5	0101	1010	1011	-5
4	0100	1011	1100	-4
3	0011	1100	1101	-3
2	0010	1101	1110	-2
1	0001	1110	1111	-1
0	0000	1111	0000	0

Table 6.8: Sur 4 bits.

Toujours sur quatre bits.



6.8.2 Représentation des nombres réels :

Dans les calculateurs, deux représentations sont utilisées pour représenter les nombres fractionnaires : le codage en virgule fixe et le codage en virgule flottante. Un nombre fractionnaire est un nombre qui comporte une partie décimale (après la virgule) non nulle. Il comporte deux parties :

- Une valeur entière,
- Une valeur fractionnaire.

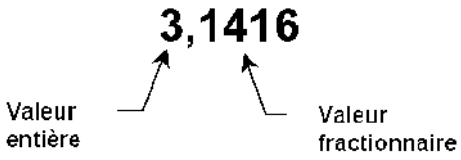
Les deux parties sont séparées par une virgule qui se place à droite du chiffre le moins significatif de la partie entière (chiffre de poids unité).

■ **Example 6.28** Voici un exemple :

■

Exemple de Division en Bianire.

Le nombre π est un nombre fractionnaire, il s'écrit :



Codage en virgule fixe

Les ordinateurs n'ont pas de virgule au niveau de la machine. Les nombres sont considérés commes des entiers, le soin de dépalcer la virgule est laisser au programmeur (Virgule virtuelle). On ne peut pas representer les plus grands nombres car on est limité par le nombre de bits. Le système de numération Binaire est un système pondéré, chaque bit d'un nombre binaire a un poids fonction de son rang (Numération et représentation des nombres). Soit le nombre binaire 1 1 0 1,0 1 1 0 1 présenté Table 6.9 :

b_3	b_2	b_1	b_0	V	b_{-1}	b_{-2}	b_{-3}	b_{-4}	b_{-5}
2^3	2^2	2^1	2^0	V	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
1	1	0	1	,	0	1	1	0	1

Table 6.9: Nombre binaire fractionnaire

Codage en virgule fixe.

Partie entière	Partie fractionnaire
$(1\times2^3)+(1\times2^2)+(1\times2^0)$	$(1\times2^{-2})+(1\times2^{-3})+(1\times2^{-5})$
$8+4+1$	$0.25+0.125+0.03125$
13	0.40625
$1101{,}01101_{(2)} = 13{,}40625_{(10)}$	

De même que le système de numération binaire, On effectue des multiplications successives par 2 de la partie fractionnaire en conservant à chaque fois la partie entière du résultat :

■ **Example 6.29** Voici un exemple : ■

Codage en virgule fixe.

Partie entière (3)	Partie fractionnaire (0.14)
$3_{(10)} = 11_{(2)}$	0,14 . 2 = 0,28 = **0** + 0,28 0,28 . 2 = 0,56 = **0** + 0,56 0,56 . 2 = 1,12 = **1** + 0,12 0,12 . 2 = 0,24 = **0** + 0,24 0,24 . 2 = 0,48 = **0** + 0,48 0,48 . 2 = 0,96 = **0** + 0,96 0,96 . 2 = 1,92 = **1** + 0,92 0,92 . 2 = 1,84 = **1** + 0,84
$3{,}14_{(10)} = 11{,}00100011_{(2)}$	

R On voit immédiatement que, contrairement à la partie entière, la partie fractionnaire peut s'exprimer par une suite **non finie**, ce qui impose de définir un critère d'arrêt.

Codage en virgule flottante

Les nombres flottants (floating point numbers) permettent de représenter, de manière approchée, une partie des nombres réels. La valeur d'un réel ne peut être ni trop grande, ni trop précise. Cela dépend du nombre de bits utilisés (en général 32 ou 64 bits). C'est un domaine très complexe et il existe une norme, **IEEE 754**, pour que tout le monde obtienne les mêmes résultats (faux bien sur). Le codage en binaire est de la forme, Table 6.10 :

S : signe	E : Exposant (signé)	M : partie fractionnaire de la mantisse (non signé)

Table 6.10: Format de la virgule flottante

On appelle la notation scientifique chaque nombre ecrit sous la forme **1,M** 2^E, où **1,M** s'appelle la mantisse du nombre et **E** l'exposant. Comme la mantisse commence toujours par une partie entière égale à 1, on ne l'écrit pas et on n'exprime que la partie fractionnaire, M, appelée « pseudo-mantisse ». De nos jours, pratiquement tous les constructeurs ont des processeurs dédiés qui effectuent des calculs en virgule flottante et qui emploient la représentation au standard IEEE 754, cette normalisation adoptée définit deux principaux types de nombres en virgule flottante

- La simple précision sur 32 bits et la double précision sur 64 bits. Les nombres en simple précision possèdent une pseudo-mantisse sur 23 bits (correspondant aux puissances négatives de 2), un exposant sur 8 bits et un bit de signe, Table 6.11.

1 bit	8 bits	23 bits	-126	+127
Signe	Exposant	Pseudo Machine	E_{min}	E_{max}

Table 6.11: La simple précision

■ **Example 6.30** Nous voulons représenter 278 au format IEEE 754 simple précision. On commence par écrire 278 en base 2, puis on le met sous la forme scientifique.
$278_(10) = 100010110_{(2)} = 1,00010112^8$.
On a donc :

- **S** = 0 car 278 est positif.
- **E_b =E + 127 = 8 + 127 = 135_{10}** $= 10000111_2$ **en base 2 sur 8 bits.**
- **M = 00010110000000000000000.**

On obtient la représentation suivante :

1 bit	8 bits	23 bits
Signe	Exposant	Pseudo Machine
0	1000111	00010110000000000000000

■

■ **Example 6.31** Nous voulons représenter -6,53125 au format IEEE 754 simple précision. On commence par écrire 6,53125 en base 2, puis on le met sous la forme scientifique.
$6,53125_10 = 110,10001_2 = 1,10100012^2$.
On a donc :

- **S** = 1 car -6,53125 est négatif.
- **E_b =E + 127 = 2 + 127 = 129_{10}** $= 10000001_2$ **en base 2 sur 8 bits.**
- **M = 10100010000000000000000.**

On obtient la représentation suivante :

1 bit	8 bits	23 bits
Signe	Exposant	Pseudo Machine
1	10000001	10100010000000000000000

■

- Les nombres en double précision ont une pseudo-mantisse sur 52 bits, un exposant sur 11 bits et un bit de signe, Table 6.12.

1 bit	11 bits	52 bits	-1022	+1023
Signe	Exposant	Pseudo Machine	E_{min}	E_{max}

Table 6.12: La Double précision

En fonction de la valeur du champ exposant, certain nombres peuvent avoir une valeur spéciale, Table 6.13. Ils peuvent être:

- **Des nombres dénormalisés** :
  En Simple précision, si exposant = 0, et la mantisse est différent de 0, le nombre doit alors être traité comme un nombre dénormalisé. Dans un tel cas, l'exposant n'est pas -127, mais -126. Le premier bit implicite, quand à lui, n'est pas 1 mais 0 : $V = (1)^S 2^{(-126)}.(0.M)$. Si la valeur du champ exposant different de 0, et la mantisse est plus grande que 0 : $V = (1)^S 2^{(E-127)}.(1.M)$.
  En Double précision, si exposant = 0, et la mantisse est différent de 0, le nombre doit alors être traité comme un nombre dénormalisé. Dans un tel cas, l'exposant est -1022. : $V = (1)^S 2^{(-1022)}.(0.M)$. Si la valeur du champ exposant different de 0, et la mantisse est plus grande que 0 : $V = (1)^S 2^{(E-1023)}.(1.M)$.

- **Zéro** : Si le champ exposant et la mantisse ont tous deux pour valeur 0, le nombre final est 0. Le bit du signe est autorisé, permettant un zéro positif ou négatif, même si cela n'a pas un grand sens mathématiquement. Il est à noter que zéro peut être considéré comme un nombre dénormalisé.

- **Infini** : Si la valeur du champ exposant est 255 (les 8 bits définis à 1) et si la valeur de la mantisse est 0, le nombre correspont à l'infinité, soit posistive ou négative, en fonction du signe.

- **NaN (not a number, pas un nombre)** : Si la valeur du champ exposant est 255 (les 8 bits définis à 1) et si la valeur de la mantisse n'est pas 0, la valeur ne doit pas être considérée comme un nombre. Cette valeur spéciale est destinée à coder le résultat d'opérations incorrectes (comme la division par zéro).

Exposant	Pseudo-mantisse	Valeurs
0	0	0
0	différent de 0	nombre dénormalisé
entre 1 et 254	quelconque nombre	normalisé standard
255	0	infini
255	différent de 0	NaN (Not a Number)

Table 6.13: Format des nombres flottants dans la norme IEEE 754

Les exemples particuliers illustrés ci-dessous sont issus de la définition ci-dessus Table 6.14:

Les concepteurs de matériel électronique se sont dit qu'il fallait normaliser le stockage des flottants en mémoire ainsi que les résultats des calculs afin que tous les ordinateurs supportent les mêmes flottants et pour que les calculs flottants donnent les mêmes résultats quelque soit l'ordinateur. C'est ainsi qu'est née la norme IEEE754. Cette norme IEEE754 impose diverses choses concernant nos flottants. Elle impose une façon d'organiser les bits de nos nombres flottants en mémoire, standardisée par la norme. Il faut tout de même noter qu'il existe d'autres normes de nombres flottants, moins utilisées.

Signe	Exposant	Pseudo-mantisse	Valeurs
0	00000000	00000000000000000000000	= 0
1	00000000	00000000000000000000000	= -0
0	11111111	00000000000000000000000	= Infinity
1	11111111	00000000000000000000000	= Infinity
0	11111111	00000100000000000000000	= NaN
0	11111111	00100010001001010101010	= NaN
0	10000000	00000000000000000000000	$= +1.2^{(128-127)}.1.0 = 2$
0	10000001	10100000000000000000000	$= +1.2^{(129-127)}.1.101 = 6.5$
1	10000001	10100000000000000000000	$= -1.2^{(129-127)}.1.101 = -6.5$
0	00000001	00000000000000000000000	$= +1.2^{(1-127)}.1.0 = 2^{(}-126)$
0	00000000	10000000000000000000000	$= +1.2^{(-126)}.0.1 = 2^{(}-127)$
0	00000000	00000000000000000000001	$= +1.2^{(-126)}.0.000.....1 = 2^{(}-149)$

Table 6.14: Format des nombres flottants dans la norme IEEE 754

Décodage

La valeur d'un nombre est donnée par :

$$(-1)^s \times (1 + \sum_{i=1}^{\text{taille de } f} f_i 2^{-i}) \times 2^{e-E_{max}} \text{ où } f_i \text{ est le bit } i \text{ de la représentation de } f$$

■ **Example 6.32** Par exemple en simple précision, considérons la reprsentation

1	1000 0010	001 1000 0000 0000 0000 0000

On a :

- $S = 1$.
- $E = 130 - 127 = 3$.
- $M = 2^{-3} + 2^{-4} = 0,125 + 0,0625 = 0,1875$.
- Le nombre codée est donc $-1,1875.2^3 = -9,5$.

■

6.9 Classification des codes

Les systèmes utilisés précédemment constituent des systèmes naturels ou codes naturels ou encore codes pondérés. Ils sont caractérisés par le fait que le poids du chiffre de rang i est B fois celui du rang i-1, B étant la base du système. Il existe d'autres codes qui possèdent des avantages particuliers pour des utilisations particulières, on peut citer : - L'utilisation d'un code particulier peut rendre le traitement d'un message plus au moins économique du point de vue temps de

traitement ou encombrement en mémoire ou encore en nombre de composant nécessaire pur effectuer le traitement. - Dans de nombreux cas on peut améliorer l'efficacité d'un système de communication en utilisant des codes détecteurs d'erreurs ou encore des codes correcteurs d'erreurs ...

6.9.1 Le code binaire pur

Il est aussi appelé code binaire naturel. C'est le code binaire sans aucune codification, c'est à dire qui découle directement du principe général de la numération. C'est le code naturel utilisé dans les systèmes numériques (ordinateur, etc.). Le tableau suivant, Table 6.15 donne le code binaire pur pour un exemple d'un mot de 4 bits $(A_0A_1A_2A_3)$:

Valeur Décimal	A_0	A_1	A_2	A_3
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Table 6.15: Code Binaire Pur.

6.9.2 Le code BCD

Le code BCD conserve la représentation décimale d'un nombre (centaines, dizaines, unités), mais chaque chiffre de ce nombre est reproduit en binaire. Etant donné que chaque rang décimal (unités, dizaines, centaines) peut contenir un chiffre de 0 à 9, chaque rang du code BCD sera représenté par quatre chiffres binaires (de 0000 à 1001), donc quatre bits.

À partir des codes EBCDIC ou ASCII, en enlevant la partie code dans chaque caractère décimal et en ne gardant que la partie numérique, on obtient pour chaque chiffre du système décimal, les représentations suivantes (système appelé BCD: Décimal Codé Binaire) : chaque chiffre d'un nombre est codé sur 4 bits,

Ce système est très utilisé pour les systèmes d'affichage sur afficheurs 7 segments. Pour afficher le nombre 9801 par exemple, au lieu d'utiliser son code binaire naturel, on utilise le code BCD et

Chiffre décimal	Code BCD
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

Table 6.16: Code BCD.

chaque afficheur reçoit les 4 bits correspondant à un chiffre.

6.9.3 Codes réfléchis

On dit qu'un code est continu au sens large si dans la table de vérité qui le définit, les états successifs sont adjacents, c'est à dire que quand on passe de l'un à l'autre, il y a un seul chiffre qui change. Un code est continu au sens stricte si en plus le premier et le dernier état sont adjacents. Un code réfléchi est un code naturel dont on a renversé le sens de variation par endroits afin de le rendre continu. On renverse une période sur deux en commençant par la deuxième. Le tableau ci-dessous illustre le code ternaire (base 3) réfléchi.

6.9.4 Code de Gray

Le code de Gray est le code binaire réfléchi (ou un seul bit change quand on passe d'une valeur à la valeur suivante), c'est un cas très important des codes continu. Il est très fréquemment utilisé notamment sur les tables de Karnaugh pour simplifier les fonctions logiques.Voici un exemple de code Gray sur 3 bits,

6.10 Codage des caractéres

Les caractères, comme les nombres, doivent être représentés en vue d'être exploités par les ordinateurs. Comme on ne peut mettre que des bits dans les cases mémoire, on a associé un code numérique à chaque caractère. Évidemment, pour pouvoir échanger des informations entre ordinateurs, il faut que cette correspondance soit la même sur toutes les machines. Voici les systèmes de codage de caractères les plus utilisés.

6.11 Code ASCII

La table du code ASCII (**American Standard Code for Information Interchange**) définit les équivalents numériques des caractères majuscules et minuscules de l'alphabet latin, des chiffres et de certains signes de ponctuation. Utilisés à l'époque (Années 60) dans l'échange de données entre terminaux et ordinateurs. Un code de 7 bits est nécessaire permettant de representer cent vingthuit

combinaisons différentes, Table 6.17. Chaque caractère est codé sur 1 octet par deux symboles hexadécimaux. Le numéro de la colonne donne le symbole hexadécimal de poids fort et le numéro de ligne le symbole de poids faible.

	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

Table 6.17: Table des caractères ASCII

Ce code est adapté à l'écriture **Anglo-Saxonne** : pas de lettres accentuées. L'unité de stockage étant l'octet, soit 8 bits, les fabricants d'ordinateurs ont décidé d'utiliser le bit supplémentaire pour définir cent vingt-huit autres caractères, comme les lettres accentuées, d'autres alphabets ou des caractères graphiques à afficher. Évidemment, chaque fabricant a construit sa propre table étendue et l'on a perdu la compatibilité entre ordinateurs, d'où les problèmes de transfert de fichiers au format texte contenant des caractères accentués. Cela a donné lieu à autre tentative de normalisation qui a aboutit à la norme ISO 8859 codifiant plusieurs alphabets d'Europe occidentale. La norme définit une quinzaine de tables différentes, chacune codant ses caractères sur 1 octet.

6.12 Code iso 8859 : une extension du codeASCII

La norme iso 8859 exploite le fait que le code ASCII n'utilise que sept bits pour un caractère. Ceci permet de coder 128 nouveaux caractères (128 à 255), alors que les codes entre 0 et 127 s'interprètent de la même façon que dans le code ascii, Table 6.18. Ceci présente l'avantage qu'un texte codé en iso 8859 pourra interpréter correctement ce texte s'il est codé en ASCII. Or un programme qui attend du code ASCII et qui reçoit un texte codé en iso 8859 ne pourra pas interpréter comme il convient les codes compris entre 128 et 255 qui n'existent pas dans le code ASCII,

	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
à	Á	â	ã	ä	Å	Æ	ç	È	é	ê	ë	ì	í	î	ï
ð	Ñ	ò	ó	ô	Õ	Ö	÷	Ø	ù	ú	û	ü	ý	þ	ÿ

Table 6.18: Extension du codeASCII

6.13 Code Unicode

À la fin des années 80 démarre un projet visant à codifier de manière unique tous les caractères de toutes les langues écrites, en tenant compte du sens de lecture des langues (de gauche à droite ou l'inverse), des ligatures, etc. Ce travail aboutit au codage Unicode dans lequel chaque symbole est défini sur 2 octets. De portée mondiale, ce codage n'est pas encore universellement appliqué car de nombreux systèmes et applications ne savent pas encore traiter ces caractères étendus. Le langage de programmation C, ancien mais très utilisé, code ses caractères sur 1 octet alors que Java les code sur 2 octets et est donc compatible avec Unicode. Notez que dans un souci d'harmonisation, les deux cent cinquante-cinq premiers caractères Unicode reproduisent les deux cent cinquante-cinq caractères du codage ISO 8859–1 de l'alphabet latin.

6.14 UTF : un codage à longueur variable

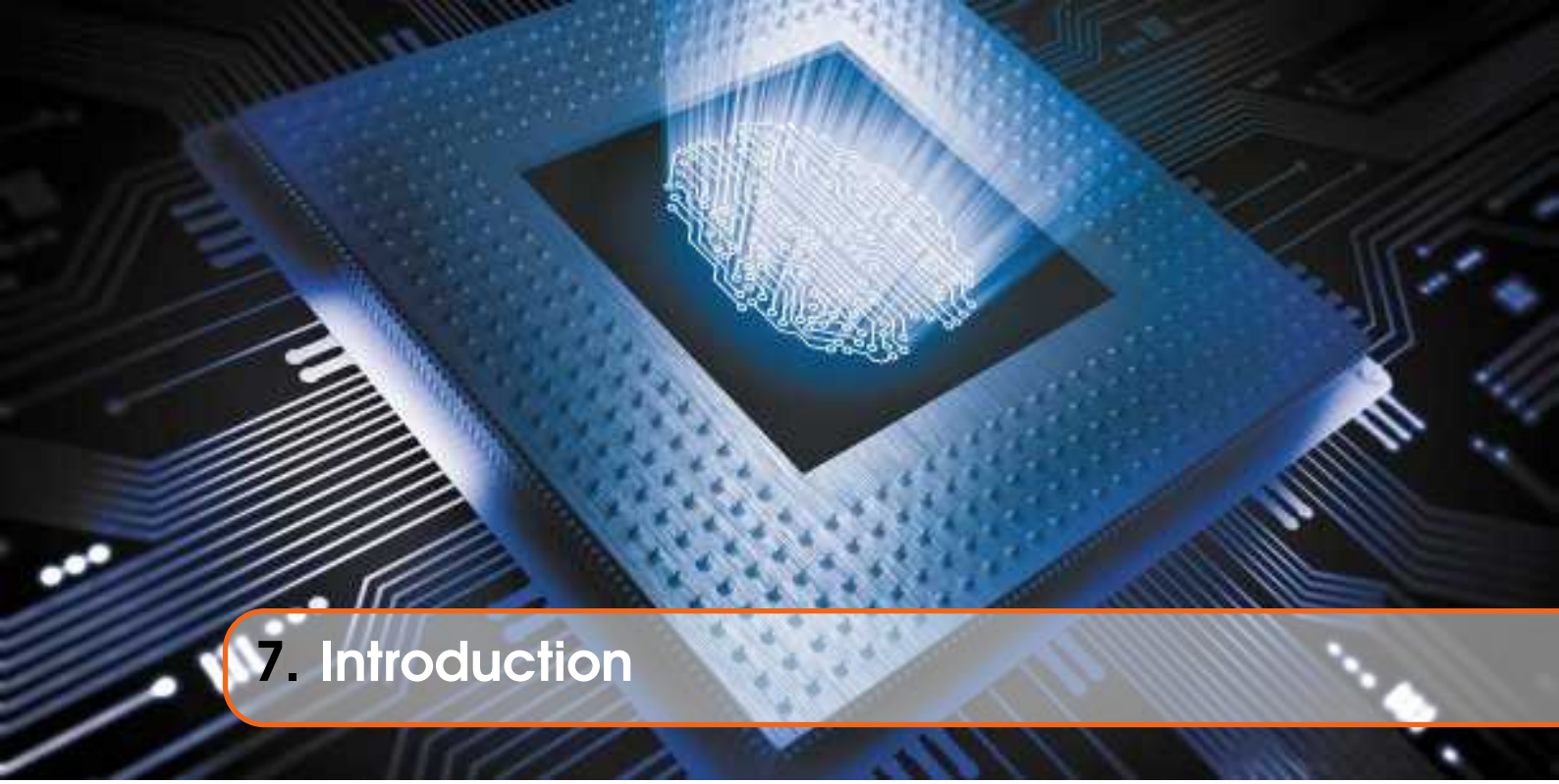
Unicode est incompatible avec le code ascii, ce qui est un réel problème. La solution consiste à lui associer un autre code, utf comme Unicode Translation Format, qui permet de maintenir la compatibilité avec le code ascii.En utf, chaque caractère est codé sur un, deux ou trois octets. Tous les caractères du code ascii sont codés en utf sur un octet avec la même valeur qu'en ascii. En revanche, un octet qui contient une valeur supérieure à 27 annonce un caractère sur deux ou trois octets, et utf est construit de telle manière que tous ces octets contiennent un code supérieur à 127. Ce mécanisme garantit une compatibilité ascendante et descendante avec l'ascii.

6.15 Conclusion

Ce chapitre présente les systèmes de numération et de codage les plus utilisés dans les systèmes numériques. La numération fait encore, néanmoins, l'objet de recherches. De nouveaux codages des nombres, réels notamment, sont à l'étude pour effectuer plus rapidement certaines opérations et améliorer les performances des opérateurs. La réalisation des opérations de codage, décodage, conversion de codes ainsi que des opérations arithmétiques est étudiée dans le chapitre 4, intitulé « Fonctions et circuits combinatoires ».

III

Algébre de Boole



7. Introduction

Un circuit numérique est réalisé à partir d'un assemblage hiérarchique d'opérateurs logiques élémentaires réalisant des opérations simples sur des variables logiques. Ces variables logiques peuvent prendre les états : Vrai ou Faux.Voici quelques systèmes physiques qui travaillent avec des grandeurs de deux états:

- Interrupteur ouvert ou fermé,
- Porte ouverte ou fermé,
- Lampe allumée ou non,
- Moteur en marche ou arrêté.

Par convention, on associe généralement à l'état vrai d'une variable logique la valeur binaire 1 et la valeur 0 à l'état faux. Electriquement, l'état vrai (valeur 1) va être associé à un niveau de tension haut et l'état faux (valeur 0) va être associé à un niveau de tension bas.

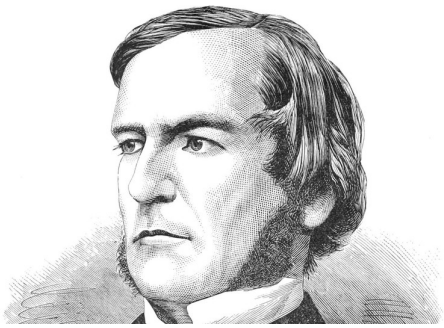


Figure 7.1: Georges BOOLE.

En logique, tout raisonnement est décomposé en une suite de propositions élémentaires qui sont vraies ou fausses. **Georges BOOLE**, Figure 7.1, du mathématicien anglais du XIXe siècle

(1815-1864) a créé une algèbre qui codifie les règles (algèbre booléenne) à l'aide de variables logiques ne pouvant prendre que deux états et d'opérations élémentaires portant sur une ou plusieurs variables. **L'algèbre de BOOLE** ne traite que de la logique combinatoire, c'est à dire des circuits numériques dont la sortie ne dépend que de l'état présent des entrées. A chaque opérateur logique booléen (NON, ET, OU) on va associer un circuit numérique combinatoire élémentaire.

8. Variables et fonctions logiques

8.1 Introduction

L'analyse des circuits logiques nécessite l'utilisation d'une algèbre spécifique, dite « **Booléenne** », fruit des travaux de George Boole, Figure 7.1. Ces travaux ont défini un ensemble d'opérateurs de base, ainsi que leurs propriétés, qui composent une algèbre permettant de concevoir tout type de circuit électronique.

8.2 Définitions

Alors que dans l'algèbre classique les variables et les fonctions peuvent prendre n'importe quelles valeurs, elles sont ici limitées aux valeurs **0 et 1**. Une fonction de n variables booléennes va être définie de $0,1^n$ dans 0,1. Elle est définie par les valeurs qu'elle prend sur les 2^n combinaisons possibles de ses variables d'entrée ; chaque valeur de la fonction ne peut être que 0 ou 1, Table 8.1.

A	B	F (a,b)
0	0	1
0	1	0
1	0	1
1	1	1

Table 8.1: Une table de vérité

On peut décrire une fonction donnée en explicitant ses 2^n valeurs, par sa table de vérité. Il s'agit d'un tableau à 2^n lignes, qui liste pour chaque combinaison possible ; la valeur prise par la fonction. L'ordre des lignes n'a pas d'importance mais pour faciliter la lecture et les comparaisons,

on écrit souvent les 2^n combinaisons dans l'ordre numérique, Table 8.1.

8.3 Les opérateurs logiques fondamentaux

Un certain nombre de fonctions booléennes forment les fonctions logiques de base qui permettent d'en construire des plus complexes. Ces fonctions de base peuvent être définies par **leur table de vérité ou leur expression algébrique**. Elles sont implémentées sous forme de circuits électroniques, appelés **« portes logiques »**, qui ont des symboles graphiques normalisés: la première, la plus ancienne, affecte une forme géométrique différente à chacune des fonctions. La seconde dessine chacun des circuits sous une forme rectangulaire et les distingue via un symbole placé à l'intérieur du rectangle. Nous présentons ici, pour chaque porte, les deux symboles.

8.3.1 Fonction NON (NOT)

L'opérateur d'inversion ne porte que sur une seule variable d'entrée, elle a une entrée booléenne et une sortie, qui se définit comme le complémentaire de l'entrée. Elle se lit **« a-barre »** ou **« non-a »**. On dit aussi que la variable a est **« complémentée »**. Le tableau Table 8.2 résume l'action de cet opérateur.Son expression algébrique est :

$$\text{NON(a)} = \overline{a}$$

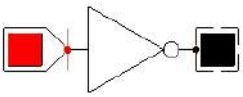
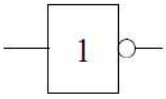
Table de vérité		Symbole traditionnel	Symbole normalisé
A	$\overline{A}$		
0	1		
1	0		

Table 8.2: Fonction NON (NOT).

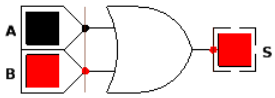
8.3.2 Fonction OU (OR)

L'opérateur OR (OU) porte sur deux variables d'entrée, elle prend la valeur 1 si l'une ou l'autre de ses entrées (ou les deux) est à 1. Elle vaut 0 si les deux entrées sont à 0. Elle se lit bien **« a ou b »**, et non **« a plus b »**. L'opérateur OR est symbolisé par le plus **(+)** comme l'addition en mathématique. Le tableau Table 8.3 résume l'action de cet opérateur. Son expression algébrique est :

$$\text{OU (a, b)} = a + b$$

Table de vérité	Symbole traditionnel	Symbole normalisé

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1



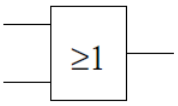


Table 8.3: Fonction OU (OR).

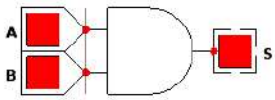
8.3.3 Fonction ET (AND)

L'opérateur AND (ET) porte sur deux variables d'entrée, elle prend la valeur 1 si ses entrées sont l'une et l'autre à 1. Elle vaut 0 si au moins une des deux entrées est à 0. Elle se lit **« a et b »**. L'opérateur AND est symbolisé par le point **(.)** comme la multiplication en mathématique. Le tableau, Table 8.4 suivant résume l'action de cet opérateur. Son expression algébrique est :

$$ET\ (a, b) = a.b = ab$$

Table de vérité	Symbole traditionnel	Symbole normalisé

A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1



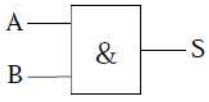


Table 8.4: Fonction ET (AND).

8.3.4 Fonction OU-exclusif (XOR)

L'opérateur XOR (OU exclusif) n'est pas un opérateur de base car il peut être réalisé à l'aide des portes précédentes. Il porte sur deux variables d'entrée.Elle prend la valeur 1 si l'une ou l'autre de ses entrées est à 1, mais pas les deux. Elle vaut 0 si les deux entrées sont égales (à 0 ou à 1). L'opérateur XOR est symbolisé par un + entouré d'un cercle $\oplus$. Le tableau, Table 8.5 suivant résume l'action de cet opérateur. Son expression algébrique est :

XOR (a, b) = a + b

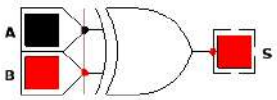
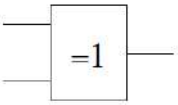
Table de vérité			Symbole traditionnel	Symbole normalisé
A	B	A⊕B		
0	0	0		
0	1	1		
1	0	1		
1	1	0		

Table 8.5: Fonction XOR

8.3.5 Fonction NON-OU (NOR)

L'opérateur NOR (NON OU) porte sur deux variables d'entrée. L'opérateur NOR est l'inverse de l'opérateur OR. Son symbole est le symbole du OU suivi d'une bulle qui matérialise l'inversion . Elle prend donc la valeur 1 uniquement quand ses deux entrées sont à 0. Le tableau, Table 8.6 suivant résume l'action de cet opérateur. Son expression algébrique est :

NON OU (a, b) = $\overline{a+b}$

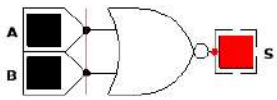
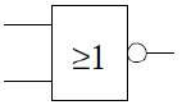
Table de vérité			Symbole traditionnel	Symbole normalisé
A	B	$\overline{A+B}$		
0	0	1		
0	1	0		
1	0	0		
1	1	0		

Table 8.6: Fonction NOR

8.3.6 Fonction NON-ET (NAND)

L'opérateur NAND (NON ET) porte sur deux variables d'entrée, elle prend donc la valeur 1 lorsqu'au moins une de ses deux entrées est à 0. L'opérateur NAND est l'inverse de l'opérateur

AND. Son symbole est le symbole du ET suivi d'une bulle qui matérialise l'inversion. Le tableau, Table 8.7 suivant résume l'action de cet opérateur. Son expression algébrique est :

$$\text{NON ET(a, b)} = \overline{a.b} = \overline{ab}$$

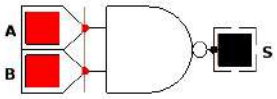
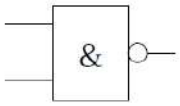
Table de vérité			Symbole traditionnel	Symbole normalisé
A	B	$\overline{A.B}$		
0	0	1		
0	1	1		
1	0	1		
1	1	0		

Table 8.7: Fonction NAND

8.3.7 XNOR (NON OU exclusif)

L'opérateur XNOR (NON OU exclusif) n'est pas non plus un opérateur de base. Il porte sur deux variables d'entrée. Si A et B sont les variables d'entrée, alors est vraie si A égale B. L'opérateur XNOR est l'inverse de l'opérateur XOR. Son symbole est le symbole du XOR suivi d'une bulle qui matérialise l'inversion. Le tableau, Table 8.8 suivant résume l'action de cet opérateur.

$$\overline{A \oplus B} = \text{A.B} + \text{A.B. S}$$

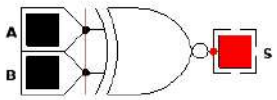
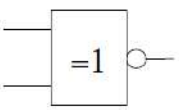
Table de vérité			Symbole traditionnel	Symbole normalisé
A	B	$\overline{A \oplus B}$		
0	0	1		
0	1	0		
1	0	0		
1	1	1		

Table 8.8: Fonction XNOR

8.4 Propriétés de Algèbre de BOOLE

L'algèbre de BOOLE porte sur des variables logiques qui ne peuvent prendre que deux états, vrai ou faux, Figure 8.1. Elle possède trois opérateurs booléens de base : **NOT (NON), AND (ET), OR (OU)** réalisant des fonctions logiques. Le résultat obtenu est booléen, c'est-à-dire vrai ou faux.

Une boite noir.

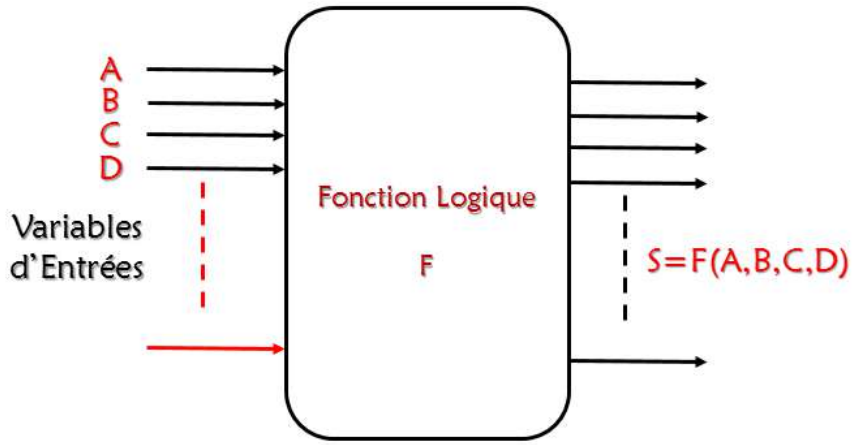


Figure 8.1: Fonction logique **F** vu comme une boite noir.

Les fonctions et les circuits logiques étant définis à partir d'expressions algébriques, il est important de pouvoir simplifier celles-ci afin de réduire **la complexité des circuits** en utilisant un certain nombre de propriétés algébriques, Figure 8.2.

Commutativité	Associativité	Distributivité
$a+b=b+a$ $ab=ba$ $a\oplus b=b\oplus a$	$a+(b+c)=(a+b)+c=a+b+c$ $a(bc)=(ab)c=abc$ $a\oplus(b\oplus c)=(a\oplus b)\oplus c=a\oplus b\oplus c$	$a+(bc)=(a+b)(a+c)$ $a(b+c)=(ab)+(ac)=ab+ac$ $a(b\oplus c)=(ab)\oplus(ac)=ab\oplus ac$
Élément neutre	**Élément absorbant**	**Idempotence**
$a+0=a$ $1.a=a$ $a\oplus 0=a$	$a+1=1$ $0.a=0$	$a+a=a$ $aa=a$
Complémentaire	**Lois de De Morgan**	**Divers**
$a+\overline{a}=1;\ \ a\overline{a}=0$ $\overline{aa}=\overline{a};\ \ \overline{a+a}=\overline{a}$ $\overline{\overline{a}}=a$ $a\oplus\overline{a}=1$ $a\oplus 1=\overline{a}$	$\overline{ab}=\overline{a}+\overline{b}$ $\overline{a+b}=\overline{a}\overline{b}$	$a+ab=a(a+b)=a$ $a+(\overline{a}b)=a+b;\ \ a(\overline{a}+b)=ab$ $a\oplus a=0$ $a\oplus\overline{b}=\overline{a}\oplus b=\overline{a\oplus b}$ $\overline{a}\oplus\overline{b}=a\oplus b$ $a\oplus b=a\overline{b}+\overline{a}b$ $\overline{a\oplus b}=ab+\overline{a}\overline{b}$

Figure 8.2: Propriétés algébriques des opérateurs.

R Dans les expressions algébriques, l'opérateur ET est prioritaire sur l'opérateur OU. On peut donc souvent enlever les parenthèses et écrire ab + c plutôt que (ab) + c.

Les lois de **Morgan** permettent de transformer un opérateur en un autre dans une expression algébrique. On peut remplacer tous les opérateurs **OU** par des opérateurs **ET** (ou vice-versa). L'intérêt est de pouvoir choisir l'opérateur utilisé pour construire la fonction à partir de son expression, selon des portes logiques disponibles. Il s'énonce des deux manières suivantes :

La négation d'un produit de variables est égale à la somme des négations des variables :

$$\overline{A.B} = \overline{A} + \overline{B}$$

La négation d'une somme de variables est égale au produit des négations des variables :

$$\overline{A+B} = \overline{A}.\overline{B}$$

8.5 Construction d'une Fonction Logique

Il existe deux méthodes pour exprimer une fonction logique :

- Donner son équation logique.
- Utiliser une table de vérité.

A un nombre fini N de variables d'entrée correspond 2^N combinaisons possibles. La table de vérité donne la valeur de la fonction pour les 2^N valeurs possibles. Pour implémenter cette fonction sous forme de circuit combinatoire, les seuls circuits élémentaires sont à disposition sont les portes logiques et celles-ci reproduisent les opérateurs de base **(NON, OU, ET)** qui sont dans l'expression algébrique d'une fonction. Il faut donc transformer la table de vérité de la fonction en expression algébrique afin de pouvoir physiquement l'implémenter à l'aide des portes logiques. Par exemple S est une fonction de trois variables, il y aura 2^3 soit 8 combinaisons possibles, Table 8.9.

Valeur entiére	A	B	C	Combinaison	S
0	0	0	0	$\overline{A}.\overline{B}.\overline{C}$	0
1	0	0	1	$\overline{A}.\overline{B}.C$	1
2	0	1	0	$\overline{A}.B.\overline{C}$	1
3	0	1	1	$\overline{A}.B.C$	1
4	1	0	0	A.$\overline{B}.\overline{C}$	0
5	1	0	1	A.$\overline{B}.C$	1
6	1	1	0	A.B.$\overline{C}$	0
7	1	1	1	A.B.C	0

Table 8.9: Fonction S

On va placer les trois variables dans **un ordre arbitraire** A, B, C de gauche à droite et écrire les combinaisons dans l'ordre des entiers naturels (0, 1, 2, 3, . . . , 2^N-1).

8.5.1 Forme Canonique Conjonctive et Disjonctive

Forme Disjonctive

Elle correspond à une somme de produits logiques : F = $\Sigma\Pi(e_i)$, où ei représente une variable ou son complément. Exemple :

$$F(X,Y,Z) = X.Y + X.\overline{Z} + \overline{X}.\overline{Y}.Z$$

Si chacun des produits contient toutes les variables d'entrée sous une forme directe ou complémentée, alors la forme est appelée « **première forme canonique** » ou « **forme canonique disjonctive** ». Chacun des produits est alors appelé **minterme**. Exemple de forme canonique disjonctive :

$$F(X,Y,Z) = \overline{X}.\overline{Y}.Z + \overline{X}.Y.Z + X.\overline{Y}..\overline{Z}$$

Forme Conjonctive

Elle fait référence à un produit de sommes logiques : F = $\Pi\Sigma(e_i)$. Voici un exemple :

$$F(X,Y,Z) = (X+Y).(X+\overline{Z}) + .(\overline{X}+Y+\overline{Z})$$

Si chacune des sommes contient toutes les variables d'entrée sous une forme directe ou complémentée, alors la forme est appelée « **deuxième forme canonique** » ou « **forme canonique conjonctive** ». Chacune des sommes est alors appelée **maxterme**. Exemple de forme canonique conjonctive :

$$F(X,Y,Z) = (X+Y+Z).(\overline{X}+\overline{Y}+Z).(\overline{X}+Y+\overline{Z})$$

Forme canonique disjonctive

Une fonction logique est représentée par l'ensemble des configurations pour lesquelles la fonction est égale à « 1 ». Considérons maintenant une configuration des entrées pour laquelle une fonction booléenne vaut « 1 » : il existe un minterm unique prenant la valeur « 1 » dans cette configuration. Il suffit donc d'effectuer la somme logique (ou réunion) des minterms associés aux configurations pour lesquelles la fonction vaut « 1 » pour établir l'expression canonique disjonctive de la fonction. Soit une fonction F de trois variables définie par sa table de vérité **??** :

On remarque que F(A;B;C) = 1 pour les états 0, 1, 3, 5. On écrit la fonction ainsi spécifiée sous une forme dite numérique : F = R(0; 1; 3; 5), Réunion des états 0, 1, 3, 5. La première forme canonique de la fonction F s'en déduit directement :

$$F(A,B,C) = \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C + \overline{A}.B.C + A.\overline{B}.C$$

A	B	C	F(A,B,C)	état	Minterme
0	0	0	1	0	$\overline{A}.\overline{B}.\overline{C}$
0	0	1	1	1	$\overline{A}.\overline{B}.C$
0	1	0	0	2	$\overline{A}.B.\overline{C}$
0	1	1	1	3	$\overline{A}.B.C$
1	0	0	0	4	A.$\overline{B}.\overline{C}$
1	0	1	1	5	A.$\overline{B}.C$
1	1	0	0	6	A.B.$\overline{C}$
1	1	1	0	7	A.B.C

Table 8.10: Table de vérité de la fonction F : états associés et mintermes

Forme canonique conjonctive

Considérons maintenant une configuration des entrées pour laquelle la fonction vaut « 0 ». Il existe un maxterm unique prenant la valeur « 0 » en cette configuration. Ce maxterm prend donc la valeur « 1 » dans toutes les autres configurations des entrées. Il suffit donc d'effectuer le produit logique (ou intersection) des maxterms associés aux configurations pour lesquelles la fonction vaut « 0 » pour établir l'expression canonique conjonctive de la fonction. Reprenons l'exemple de la fonction F, **??** :

A	B	C	F(A,B,C)	état	Maxterme
0	0	0	1	0	A+B+C
0	0	1	1	1	A+B+$\overline{C}$
0	1	0	0	2	A+$\overline{B}+C$
0	1	1	1	3	A+$\overline{B}+\overline{C}$
1	0	0	0	4	$\overline{A}+B+C$
1	0	1	1	5	$\overline{A}+B+\overline{C}$
1	1	0	0	6	$\overline{A}+\overline{B}+C$
1	1	1	0	7	$\overline{A}+\overline{B}+\overline{C}$

Table 8.11: Table de vérité de la fonction F : états associés et maxtermes

On remarque que F(A;B;C) = 0 pour les états 2, 4, 6, 7. On écrit la fonction ainsi spécifiée sous une forme dite numérique : F = I(2; 4; 6; 7) Intersection des états 2, 4, 6, 7. La deuxième forme canonique de la fonction F s'en déduit directement :

$$F(A,B,C) = (A+\overline{B}+C).(\overline{A}+B+C).(\overline{A}+\overline{B}+C).(\overline{A}+\overline{B}+\overline{C})$$

8.5.2 Création des circuits logiques a partir d'un texte

Pour créer des circuits logiques a partir du texte, il faut créer manuellement la table de vérité de la dicription du texte ensuite tirer les mintermes (ou maxtermes) avec les fonctions logiques de base.

■ **Example 8.1** On a trois juges qui controlent le départ d'une course. La course a lieu si au moins deux des trois juges sont prets. Créer le circuit logique qui représente le départ d'une course.

Solution : Les trois juges forment les trois entrées : A, B et C. Le départ de la course représente la sortie F. On peut ensuite créer manuellement la table de vérité de cette fonction, Table 8.12 :

A	B	C	F(A,B,C)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 8.12: Table de véritè.

On peut ensuite exprimer cette fonction comme une somme de mintermes : F = $\Sigma(3;5;6;7)$. Puis on va simplifier la fonction.

$$\begin{aligned} \text{F} = &\Sigma(3,5,6,7) = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC \\ &= \overline{A}BC + A\overline{B}C + ABC + ABC + AB\overline{C} + ABC \\ &= \text{BC(A+}\overline{A}) + AC(B+\overline{B}) + AB(C+\overline{C}) \\ &= \text{AB + BC +AC} \end{aligned}$$

A partir de la fonction simplifiée, on peut créer le circuit, Figure 8.3.

Table de vérité de la fonction F.

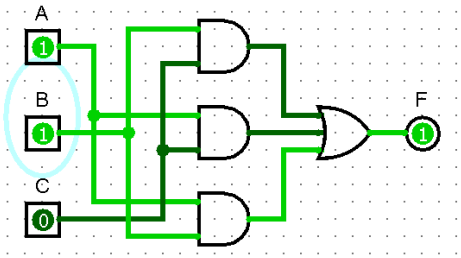


Figure 8.3: Table de vérité de la fonction F.

■

8.6 Simplifiation des expressions logiques

La minimisation est une partie importante du design de circuits logiques. On doit simplifier le plus possible la fonction logique avant d'essayer d'implanter le circuit logique afin de réduire sa complexité, et son cout. Pour simplifier une expression d'une fonction logique trois méthodes sont utilisables :

Le raisonnement: On cherche, à partir du problème à résoudre, l'expression la plus simple possible. Evidemment, cette méthode ne garantit pas un résultat optimal.

La table de vérité et les propriétés de l'algèbre de BOOLE: C'est plus efficace, mais il est facile de rater une simplification, notamment quand la fonction est compliquée.

La méthode graphique des tableaux de Karnaugh: C'est la méthode la plus efficace car elle garantit le bon résultat. Cette méthode est utilisée sous forme informatique dans tous les outils de CAO.

8.6.1 Simplification algébrique

Elle consiste à appliquer les propriétés de l'algèbre de Boole, Figure 8.2 aux expressions algébriques des fonctions logiques. Nous allons traiter quelques exemples qui permettront de passer en revue la plupart des astuces utilisées pour mener à bien les simplifications.

■ **Example 8.2** Prenons un exemple. On a deux voyants A et B. On veut déclencher une alarme quand au moins un des deux voyants est allumé. Ecrivons la table de vérité, Table 8.13 :

A	B	S
0	0	0
0	0	1
1	1	1
1	1	1

Table 8.13: Table de véritè d'une Alarme.

On obtient donc :

$$S = \overline{A}.B + A.\overline{B} + A.B$$

D'où on tire (d'après les propriétés de l'algèbre de BOOLE) :

$$\begin{aligned} S &= \overline{A}.(B + \overline{B} + B.\overline{A} \\ S &= A.1 + B + \overline{A} \\ S &= A + B + \overline{A} \\ S &= A + B \end{aligned}$$

Alarme.

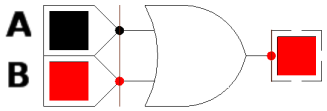


Figure 8.4: Alarme.

■

■ **Example 8.3** Soit une fonction F1 de trois variables: A, B, C :

$F1 = BC + AC + AB + B$, on a : $AB + B = B$
donc $F1 = BC + AC + B$
d'ou $F1 = AC + B$,car$BC + B = B$.

■

■ **Example 8.4** Soit une fonction F2 de trois variables: A, B, C :

$F2 = (A + \overline{B})(A\overline{B} + C)C$, d'ou $(X + C)C = C$
$F2 = (A + \overline{B})C$
donc $F2 = AC + \overline{B}C$

■

■ **Example 8.5** Soit une fonction F3 de trois variables: A, B, C :

$F3 = \overline{A}B\overline{C} + AB\overline{C} + ABC + \overline{A}BC$
d'ou $AX + \overline{A}X = (A + \overline{A})X = X$
$F3 = B\overline{C} + BC$
$F3 = B$

■

■ **Example 8.6** Soit une fonction F4 de trois variables: A, B, C :

$F4 = \overline{A}B + AC + BC$
L'expression reste inchangée si le troisième terme est multiplié par 1 :
$F4 = \overline{A}B + AC + (A + \overline{A})BC$.
En utilisant la distributivité de ET par rapport à OU, on obtient :
$\overline{A}B + AC + ABC + \overline{A}BC$
$F4 = \overline{A}B(1 + C) + AC(1 + B)$
$F4 = \overline{A}B + AC$

■

■ **Example 8.7** Soit une fonction F5 de trois variables: A, B, C :

$$F5 = (\overline{A}+B)(A+C)(B+C)$$

On ne change pas F5 en ajoutant 0 à l'un des termes :

$$F5 = (\overline{A}+B)(A+C)(B+C+\overline{A}A)$$

En utilisant ensuite la distributivité de OU par rapport à ET on obtient : $(\overline{A}+B)(A+C)(\overline{A}+B+C)(A+C+B)$

$$F5 = (\overline{A}+B+0.C)(A+0.B+C)$$

$$F5 = (\overline{A}+B)(A+C)$$

■

Les méthodes algébriques de simplification présentent un inconvénient majeur : elles ne sont pas systématiques, et leur efficacité dépend donc largement du savoir-faire de la personne qui les applique. Elles ne peuvent, par conséquent, être utilisées que ponctuellement sur des cas simples.

8.6.2 Tableaux de Karnaugh

Le diagramme ou **tableau de Karnaugh** est un outil graphique sous forme matricielle qui permet de simplifier de façon méthodique une fonction logique, Figure 8.5. Bien que les diagrammes de Karnaugh soient applicables en théorie à des fonctions ayant un nombre quelconque de variables,Figure 8.6, Figure 8.7,ils ne sont en pratique utilisables « à la main » que pour un nombre de variables **inférieur ou égal à 6**. Cette méthode permet :

- D'avoir l'expression logique la plus simple pour une fonction F.
- De trouver des termes communs pour un système à plusieurs sorties, dans le but de limiter le nombre de portes.
- De tenir compte de combinaisons de variables d'entrées qui ne sont jamais utilisées. On peut alors mettre 0 ou 1 en sortie afin d'obtenir l'écriture la plus simple.

Table de Karnaugh à 2 variables.

x_2 \ x_1	0	1
0	$f(0,0)$	$f(1,0)$
1	$f(0,1)$	$f(1,1)$

Figure 8.5: Tableaux de Karnaugh.

Les lignes et les colonnes d'une table de Karnaugh sont écrites dans l'ordre binaire réfléchi, ce qui met en évidence la propriété d'adjacence, c'est-à-dire, deux cases adjacentes de la table contenant la valeur booléenne "1" correspondent à 2 minterms qui ne diffèrent que par l'état d'une seule variable.

Table de Karnaugh à 3 variables.

x_3 \ $x_1\ x_2$	00	01	11	10
0	$f(0,0,0)$	$f(0,1,0)$	$f(1,1,0)$	$f(1,0,0)$
1	$f(0,0,1)$	$f(0,1,1)$	$f(1,1,1)$	$f(1,0,1)$

Figure 8.6: Tableaux de Karnaugh.

Table de Karnaugh à 4 variables.

$x_3\ x_4$ \ $x_1\ x_2$	00	01	11	10
00	$f(0,0,0,0)$	$f(0,1,0,0)$	$f(1,1,0,0)$	$f(1,0,0,0)$
01	$f(0,0,0,1)$	$f(0,1,0,1)$	$f(1,1,0,1)$	$f(1,0,0,1)$
11	$f(0,0,1,1)$	$f(0,1,1,1)$	$f(1,1,1,1)$	$f(1,0,1,1)$
10	$f(0,0,1,0)$	$f(0,1,1,0)$	$f(1,1,1,0)$	$f(1,0,1,0)$

Figure 8.7: Tableaux de Karnaugh.

8.6.3 Adjacence logique

Deux termes sont dits logiquement **adjacents s'ils ne diffèrent que par une variable**. Par exemple, ABC et $\overline{A}BC$ sont deux termes produits adjacents, et $\overline{A}+\overline{B}+\overline{C}+D$ et $\overline{A}+B+\overline{C}+D$ sont deux termes sommes adjacents. La somme de deux produits adjacents et le produit de deux sommes adjacentes peuvent être simplifiés par mise en facteur, en raison des propriétés de distributivité réciproque des opérateurs ET et OU. En effet,

$$AB+A\overline{B}=A(B+\overline{B})=A \text{ (distributivité de ET par rapport à OU),}$$
$$A+B)(A+\overline{B})=A+B\overline{B}=A \text{ (distributivité de OU par rapport à ET).}$$

Les tableaux de Karnaugh sont une variante des tables de vérité. Ils sont organisés de telle façon que les termes à 1 (ou à 0) adjacents soient systématiquement regroupés dans des cases voisines, donc faciles à identifier visuellement (les termes **logiquement adjacents** sont également **géométriquement adjacents**), Figure 8.8.

Afin de mettre visuellement en évidence les simplifications possibles. On représente les valeurs de la fonction dans un tableau aussi carré que possible, dans lequel chaque ligne et chaque colonne correspondent à une combinaison des variables d'entrée exprimée avec un code adjacent (le code GRAY en général), Table 8.14.

R Dans un code adjacent, seul un bit change d'une valeur à la valeur suivante. Exemple avec deux variables, Table 8.14 :

Tableaux de Karnaugh.

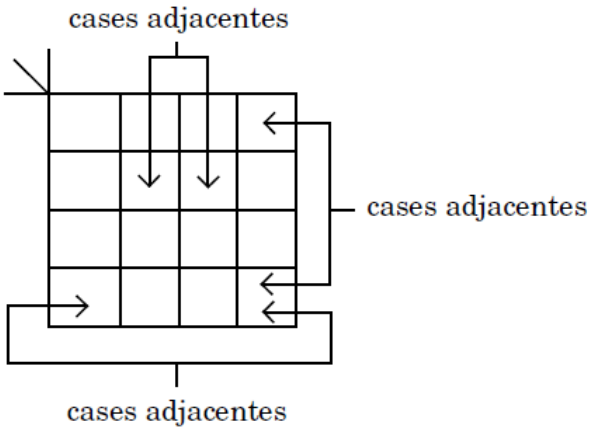


Figure 8.8: Cases adjacentes dans un tableau de Karnaugh.

Code Binare Naturel		Code de GreY	
A	B	A	B
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	0

Table 8.14: Code de GreY.

On généralise la notion d'adjacence au niveau du diagramme en disant que deux cases sont adjacentes quand leurs adresses le sont. Ainsi les cases de l'extrémité droite sont adjacentes à celles de l'extrémité gauche et les case de l'extrémité supérieure sont adjacentes à celles de l'extrémité inférieure. Cela se passe comme si on **enroulait la feuille de papier** sur laquelle est dessiné le diagramme de Karnaugh d'abord horizontalement puis verticalement, Figure 8.9.

La méthode de Karnaugh est applicable à partir d'une représentation de la fonction sous une de ses deux formes algébriques canoniques. En pratique, la première forme canonique (forme disjonctive) est la plus utilisée, mais toutes les étapes décrites sont également applicables pour une représentation sous la forme conjonctive.

Les traits gras correspondent à une zone où la variable vaut 1. Pour la lecture et la simplification de l'expression, on cherche des paquets les plus gros possibles **(de 1, 2, 4 ou 8 variables)**, en se rappelant que le code est aussi adjacent sur les bords **(bord supérieur avec bord inférieur, bord gauche avec bord droit)**. On effectue une lecture par « intersection » en recherchant la ou les variables **ne changeant pas**.

Adjacence logique .

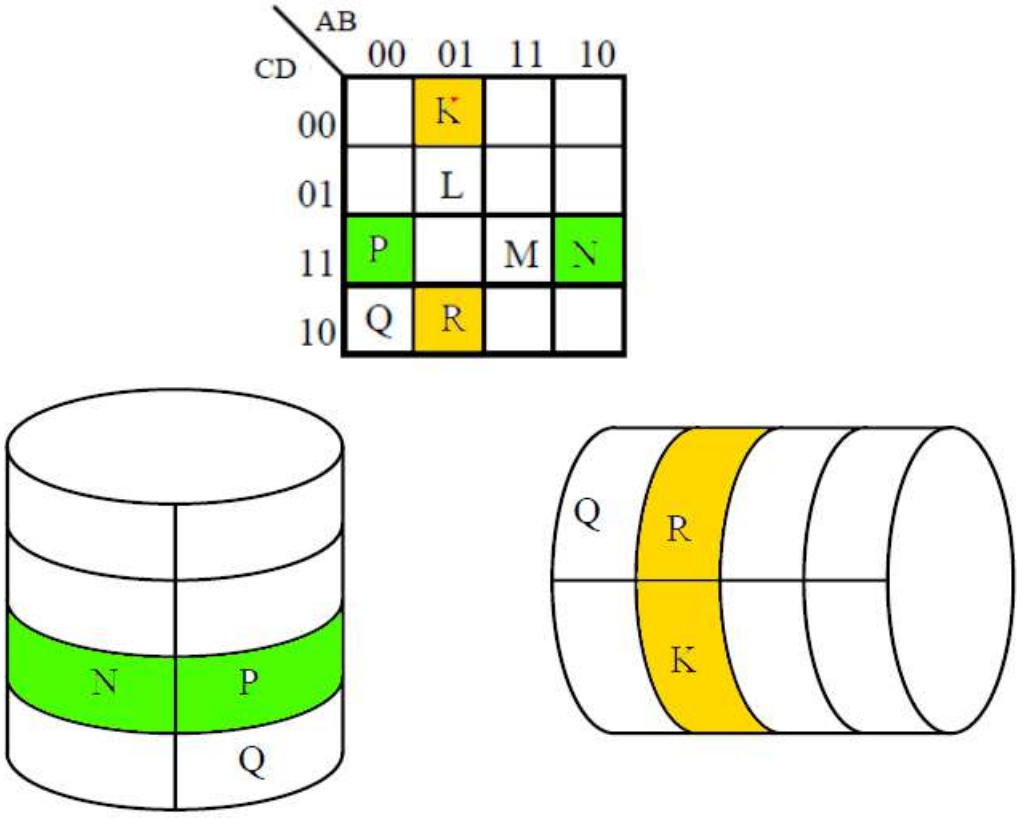


Figure 8.9: Adjacence logique.

■ **Example 8.8** Deux variables d'entrée A et B, Figure 8.10.

Tableaux de Karnaugh.

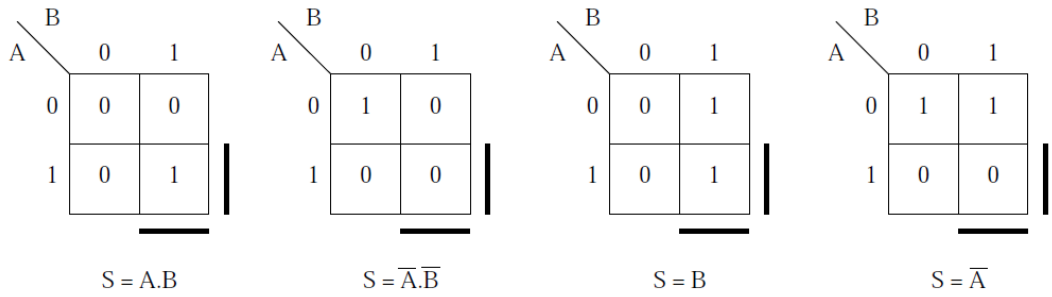


Figure 8.10: Deux variables.

■

■ **Example 8.9** Trois variables d'entrée A, B et C, Figure 8.13.

Tableaux de Karnaugh.

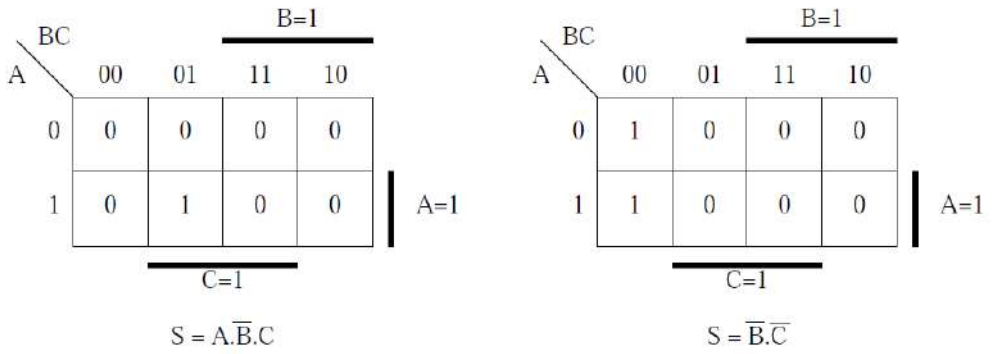


Figure 8.11: Trois variables.

Tableaux de Karnaugh.

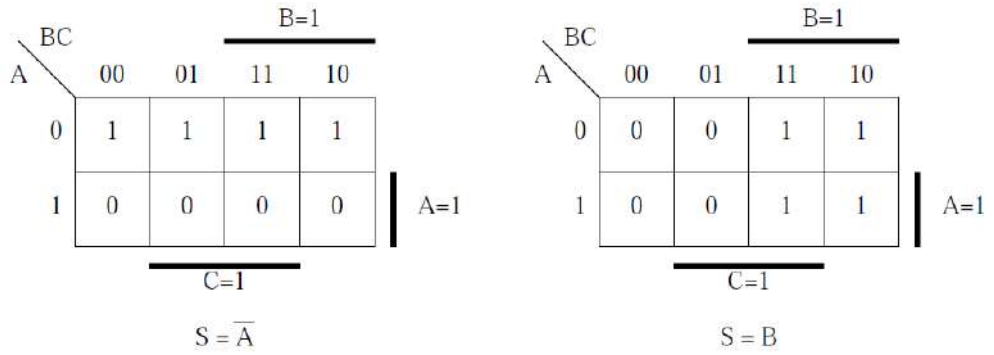


Figure 8.12: Trois variables.

Tableaux de Karnaugh.

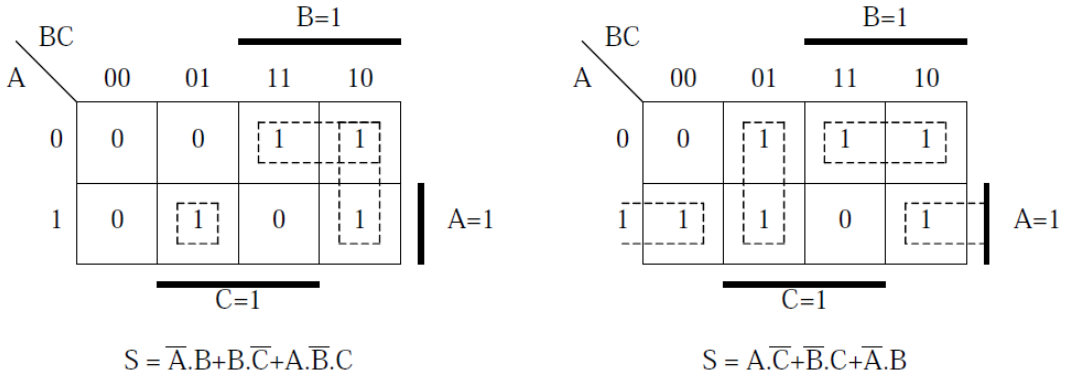


Figure 8.13: Trois variables.

■

■ **Example 8.10** Quatre variables d'entrée A, B, C et D, Figure 8.14.

Tableaux de Karnaugh.

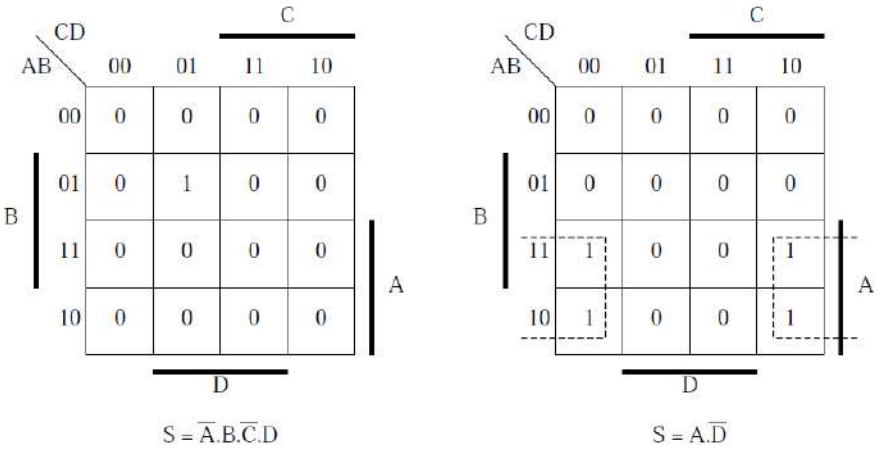


Figure 8.14: Quatre variables.

Tableaux de Karnaugh.

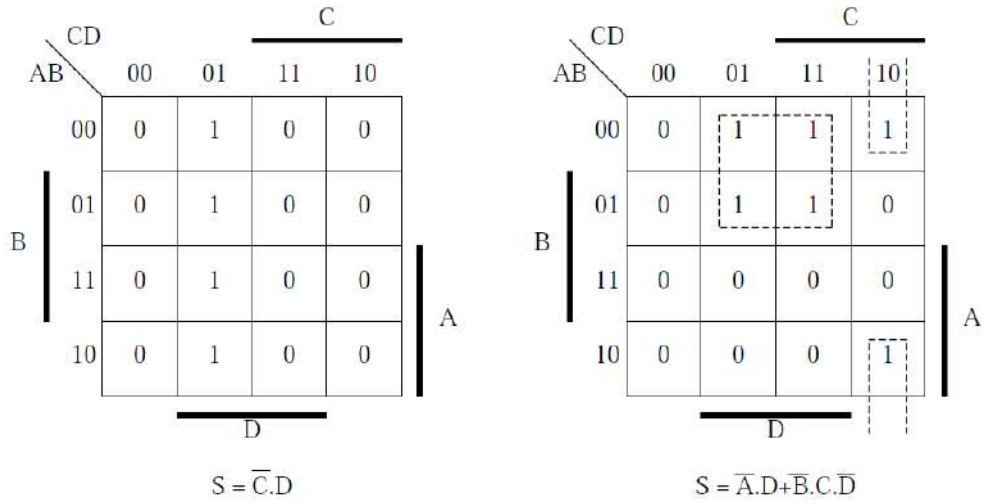


Figure 8.15: Quatre variables.

■

R 1 case = produit de 4 variables, 2 cases = produit de 3 variables, 4 cases = produit de 2 variables, 8 cases = 1 variable, 16 cases = 1 ou 0.

Dans une fonction logique, il peut exister des états non utilisés. Ces combinaisons n'ont pas d'importance pour le problème à résoudre. On les appelle en anglais des états « don't care » (ne pas tenir compte) et on les symbolise par un X.

Tableaux de Karnaugh.

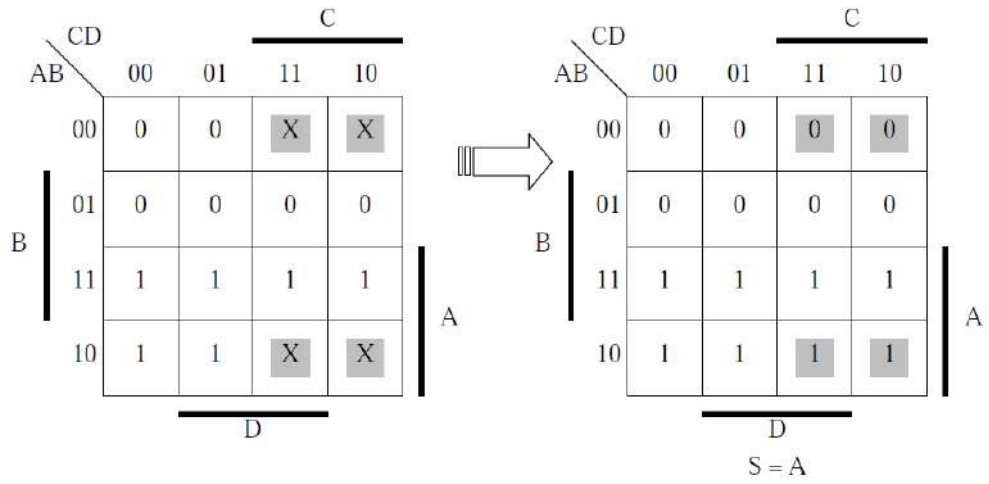


Figure 8.16: Variables X.

■ **Example 8.11** Les états **X** peuvent prendre la valeur la plus pratique pour simplifier la fonction. Exemple, Figure 8.16 :

■

Il existe un cas particulier de fonction qui est parfois difficile à détecter, le OU exclusif (XOR) ou son complément. Il faut repérer des 1 disposés en quinconce.

■ **Example 8.12** Cas du XOR, Figure 8.17 :

Tableaux de Karnaugh.

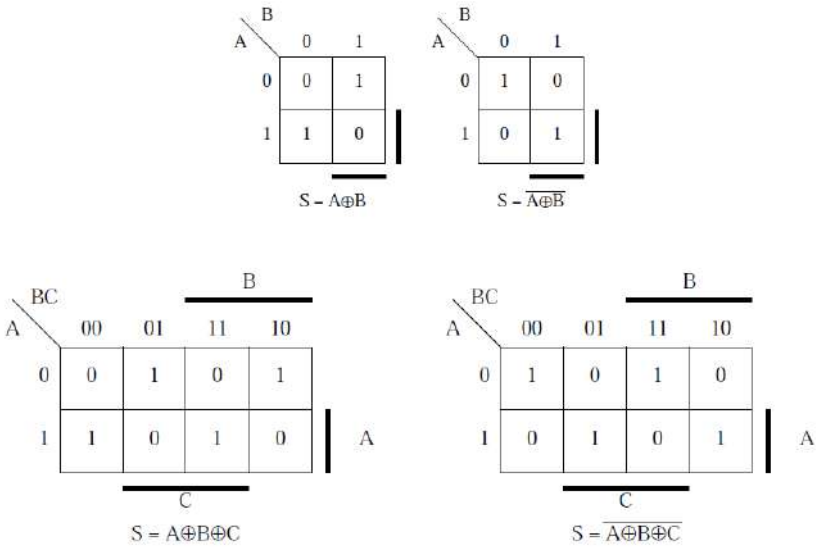


Figure 8.17: Variables XOR.

■

8.6.4 Les Regroupements:

Les lignes et colonnes sont disposées de telle sorte que, entre deux cases adjacentes (horizontalement ou verticalement), il n'y ait qu'une variable qui change d'état. On remplit ce tableau à l'aide des valeurs que prend la fonction. On peut donc développer la forme canonique en extrayant toutes les cases (les mintermes) où la fonction vaut 1 et de regrouper ces cases par ensembles de deux, quatre, huit, seize, etc., pour simplifier les mintermes.

Boucle d'ordre 2

C'est le cas lorsque deux cases sont placées réellement côte à côte mais aussi placées aux deux extrémités d'une ligne ou d'une colonne, Figure 8.18.

■ **Example 8.13** Boucle d'ordre 2:

Tableaux de Karnaugh.

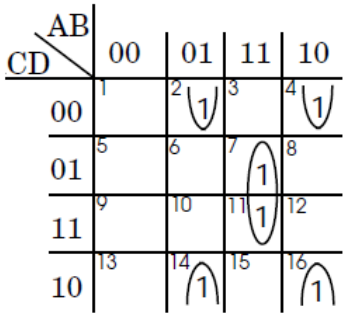


Figure 8.18: Boucles d'ordre 2 dans un tableau de Karnaugh.

■

R **Boucles imbriquées :** On constate Figure 8.19 que deux boucles sont possibles (7)-(11) et (10)-(11). Bien que ces deux boucles possèdent la case (11) en commun, il est néanmoins tout à fait possible d'appliquer la règle précédente.

Tableaux de Karnaugh.

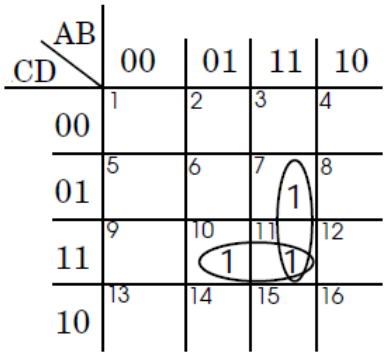


Figure 8.19: Boucles imbriquées dans un tableau de Karnaugh.

Boucle d'ordre 4

Une boucle d'ordre 4 peut prendre diverses formes, Figure 8.20 :

■ **Example 8.14** Boucle d'ordre 4:

Tableaux de Karnaugh.

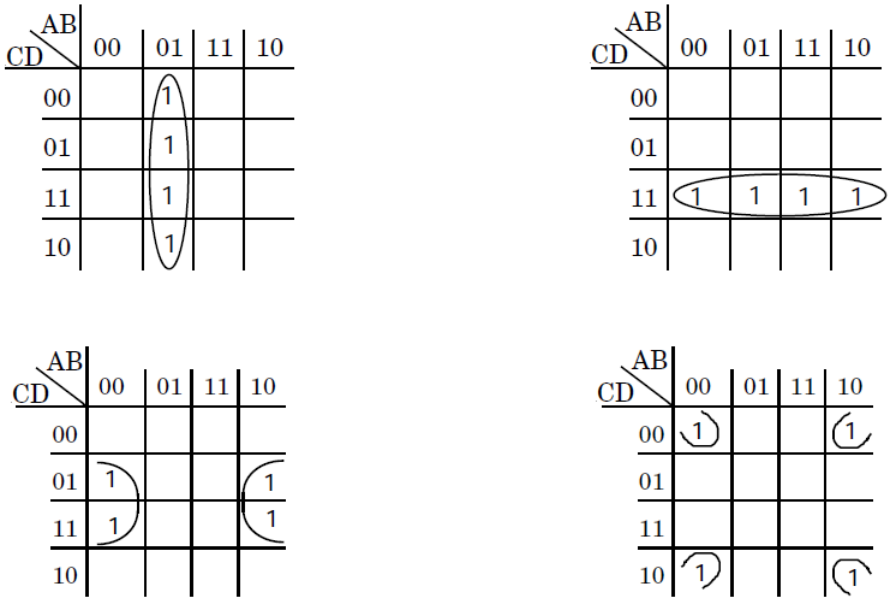


Figure 8.20: Boucles d'ordre 4 dans un tableau de Karnaugh.

■

Boucle d'ordre 8

Une boucle d'ordre 8 peut prendre diverses formes, Figure 8.21 :

■ **Example 8.15** Boucle d'ordre 8:

Tableaux de Karnaugh.

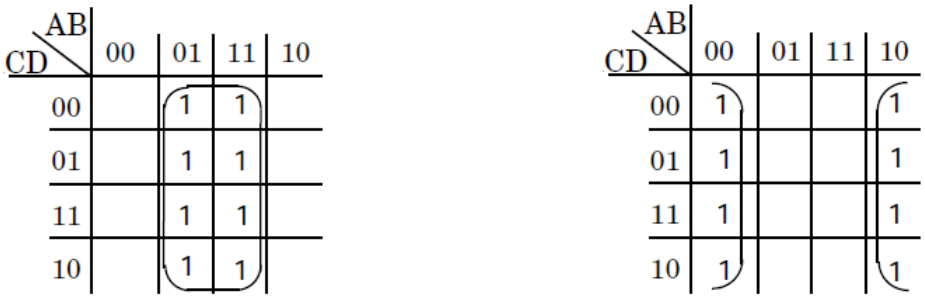


Figure 8.21: Boucles d'ordre 8 dans un tableau de Karnaugh.

■

8.6.5 Technique à appliquer sur un diagramme de Karnaugh quelconque

Pour obtenir une expression simplifiée minimale, il faut inclure tous les 1 du tableau dans des groupements de taille 2^n en respectant les principes suivants :

- Essayer de minimiser le nombre de groupements afin de minimiser le nombre de termes dans l'expression de la fonction. Il est alors préférable de rechercher les groupements en commençant par les cases qui ne peuvent se grouper que d'une seule façon. Ceci permet d'utiliser chaque 1 un minimum de fois.

- Vérifier que toutes les cases d'un groupe partagent le même nombre d'adjacences avec leurs congénères du groupe (soit n adjacences pour un groupe de 2^n cases).

- Les groupements de 1 doivent être les plus grands possibles (minimisation du nombre de variables).

R Il faut faire les regroupements les plus « gros » possibles, afin d'obtenir les monômes les plus petits possibles. Il faut également faire le minimum de regroupement.

Lorsque le nombre de variables devient important, au-delà de **6**, la manipulation des diagrammes de Karnaugh devient quasi-impossible. Il est alors nécessaire de recourir à des méthodes algorithmiques et d'utiliser un calculateur. Ce sont de telles méthodes qui sont utilisées dans les outils de synthèse automatique que l'on trouve actuellement sur le marché. Leur présentation sort du cadre de ce cours.

8.7 Conclusion

La minimisation des fonctions logiques permet une réalisation pratique utilisant un nombre minimal de composants, mais elle n'est pas une fin en soi. Dans les techniques actuelles d'intégration, la minimisation du nombre de composants n'est pas toujours le principal objectif : certaines contraintes de vitesse, de fiabilité peuvent même amener à augmenter la complexité d'un circuit. De plus, le progrès technologique aidant, la densité d'intégration est devenue aujourd'hui telle que le gain de quelques dizaines d'opérateurs logiques est souvent négligeable devant la complexité des circuits (plusieurs centaines de milliers d'opérateurs élémentaires par circuit en technologie CMOS).

IV Les Circuits Logiques Combinatoires

9. Introduction

Un circuit intégré est une plaquette de silicium qui posséde une surface de l'ordre de 5500mm^2 pouvant contenir plusieurs millions d'éléments de types :

- **Actifs :** transistors qui amplifient un courant ou une tension.
- **Passifs :** résistances, condensateurs, diodes.

Les circuits intégrés (chip en anglais) peuvent etre classés suivant le nombre de portes qui les composent. On distingue ainsi les circuits :

- **SSI** (Small Scale Integration) contenant moins de 100 portes.
- **MSI** (Medium Scale Integration) contenant entre 100 et 1000 portes.
- **LSI** (Large Scale Integration) contenant entre 1000 et 10^5 portes.
- **VLSI** (Very Large Scale Integration) contenant entre 10^5 et 10^7 portes.
- **ULSI** (Ultra Large Scale Integration) contenant plus de 10^7 portes.

Les circuits intégrés se divisent en deux grandes familles technologiques que sont:

- **Les technologies bipolaires :** TTL (Transistor Transistor Logic), ECL (Emitter Coupled Logic) utilisées dans le SSI et MSI.
- **Les technologies unipolaires :** PMOS, NMOS,CMOS.

Les portes MOS sont en moyenne 10 fois plus lentes que les portes TTL et 100 fois plus lentes que les portes ECL, mais sont intéressantes en raison de leur faible encombrement et leur faible consommation d'énergie. L'évolution des technologies actuelles repose principalement sur l'utilisation de nouveaux semi-conducteurs tel que l'arséniure de gallium avec aluminium (ALGaAs) qui permet un déplacement plus rapide des électrons (entre 3 à 5 fois plus rapide que dans le silicium). Parmi les circuits intégrés, on distingue : les circuits combinatoires et les circuits séquentiels.

Un circuit combinatoire est un circuit logique dont l'état des sorties ne dépend que des valeurs assignées aux variables d'entrée au moment considéré (effectuent des calculs) par opposition aux circuits séquentiels dont les sorties dépendent non seulement des entrées, mais aussi de l'histoire

du circuit (mémorisent de l'information). Pour les circuits combinatoires chaque configuration des entrées correspond a une configuration des sorties et une seule. On distingue comme opérateurs combinatoires standard :

- Les opérateurs de transcodage,
- Les opérateurs d'aiguillage,
- Les opérateurs de comparaison,
- Les opérateurs arithmétiques.

Il est bien sûr impossible d'illustrer tous les circuits utilisés. C'est pourquoi ce chapitre décrit succintement le développement théorique de quelques structures combinatoires de base qui jouent un rôle fondamental dans la conception des différents blocs d'un ordinateur : décodeurs, multiplexeurs, démultiplexeurs, réseaux logiques programmables, et additionneurs. Ces circuits représentent des circuits d'intégration moyenne (MSI).

10. Les Circuits Combinatoires

Les circuits logiques combinatoires sont des circuits constitués des portes fonctionnant simultanément et réalisant une ou plusieurs fonctions logiques, Figure 10.1. A une combinaison d'entrées ne correspond qu'une seule combinaison de sorties. La sortie apparaît après application de l'entrée avec un certain retard qui est le temps de propagation dans la logique interne. Ce temps est déterminé par la technologie utilisée tels que le nombre de portes traversées et la longueur des interconnections métalliques.

Circuits Logiques Combinatoires.

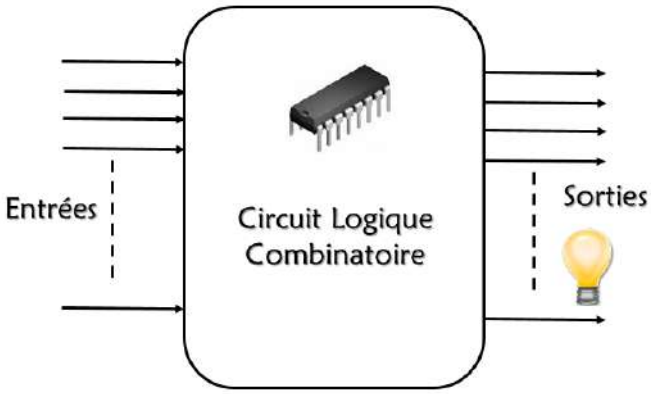


Figure 10.1: Shéma des circuits logiques combinatoires.

Les circuits combinatoires peuvent servir à :

- Traduire des bits en chiffres représentant un nombre (lettres ou code). On appelle ces circuits des codeurs (décodeurs pour l'opération inverse). Par exemple, un codeur Gray ou bien BCD.
- Effectuer des opérations arithmétiques sur des nombres. Par exemple, un additionneur ou un

multiplieur.

- Transmettre ou recevoir des informations sur une ligne unique de transmission, ce qui nécessite de transformer un nombre écrit sous forme parallèle en une suite de bits mis en série et vice-versa. Par exemple des circuits multiplexeur/démultiplexeur.

10.1 Analyse des circuits combinatoires

L'analyse des circuits combinatoires permet de déterminer la fonction logique implémentée par le circuit. On commence avec un diagramme de circuit, et on termine avec une fonction logique ou une table de vérité, ou meme une explication du fonctionnement du circuit. La première étape de l'analyse est de s'assurer que le circuit est combinatoire et non pas séquentiel. L'analyse des circuits séquentiels n'est pas la meme que celle des circuits combinatoires. Il ne faut pas qu'il y ait de parcours de feedback entre la sortie et l'entrée, ni aucun élément mémoire dans le circuit. Ensuite, on doit créer une **table de vérité** ou déterminer **la fonction du circuit** :

- Nommer toutes les sorties internes du circuit. Déterminer la fonction logique de ces sorties.
- Répéter jusqu'à ce que toutes les sorties du circuit soient seulement fonction des entrées.

■ **Example 10.1** Analyser le circuit de Figure 10.2. Déterminer la fonction logique .

Analyse d'un circuit combinatoire.

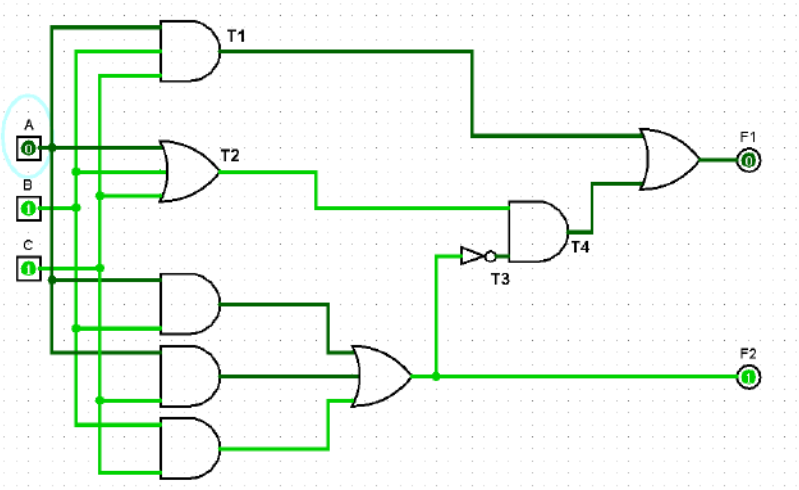


Figure 10.2: Analyse d'un circuit combinatoire.

Les sorties intermédiaires sont nommées dans le circuit de la Figure 10.2. Le circuit a trois entrées : A, B et C, et deux sorties : F1 et F2. Les trois premiéres sorties sont :

$$F2 = AB + AC + BC$$
$$T1 = A + B + C$$
$$T2 = ABC$$

Ensuite, les sorties qui viennent de signaux déjà dèfinis :

T4 = T2T3
F1 = T4 + T1

Pour obtenir F1 en fonction de A, B, et C, on doit effectuer des substitutions.

$$\begin{aligned} \text{F1} &= T4 + T2 = T3T1 + ABC \\ &= \overline{(AB + AC + BC)}(A + B + C) + ABC \\ &= (\overline{A} + \overline{B})(\overline{A} + \overline{C})(\overline{B} + \overline{C})(A + B + C) + ABC \\ &= (\overline{A} + \overline{BC})(A\overline{B} + A\overline{C} + B\overline{C} + \overline{B}\overline{C}) + ABC \\ &= \overline{A}B\overline{C} + \overline{A}\overline{B}C + A\overline{B}\overline{C} + ABC \end{aligned}$$

■

10.2 Les opérateurs d'aiguillage

On distingue deux classes d'opérateurs d'aiguillage :**les Multiplexeurs** et les **Démultiplexeurs**.

10.2.1 Le multiplexeur

Le multiplexeur est un sélecteur de données ou aiguillage convergent. Il peut transformer une information apparaissant sous forme de n bits en parallèle en une information se présentant sous forme de n bits en série. La voie d'entrée E, sélectionnée par son adresse A, est reliée à la sortie S, Figure 10.3. L'affichage d'une adresse permet de sélectionner une entrée de données parmi N, pour l'aiguiller vers la sortie S.Le multiplexeur est la fonction inverse du démultiplexeur.

Le multiplexeur 2^n vers 1.

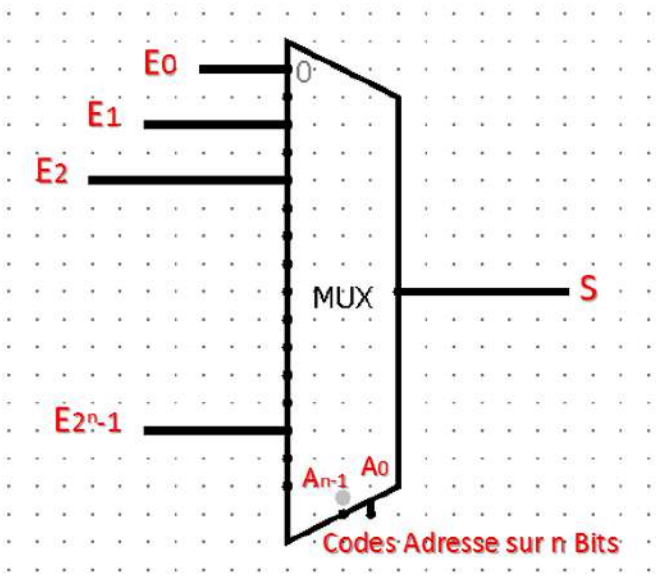


Figure 10.3: Le multiplexeur 2^n vers 1.

■ **Example 10.2** Un multiplexeur 4 vers 1, Figure 10.4.

Le multiplexeur 4 vers 1.

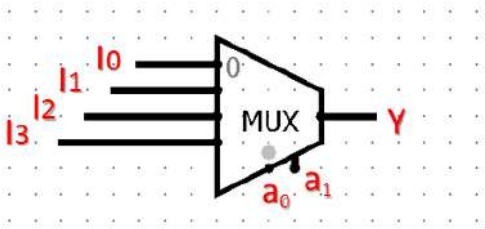


Figure 10.4: Le multiplexeur 4 vers 1.

La table de vérité Table 10.1 comporte:

	a_0	a_1	Y
0	0	0	I_0
1	0	1	I_1
2	1	0	I_2
3	1	1	I_3

Table 10.1: Table de vérité du multiplexeur 4 vers 1.

Le schéma logique est le suivant, Figure 10.5.

Le schéma logique du multiplexeur 4 vers 1.

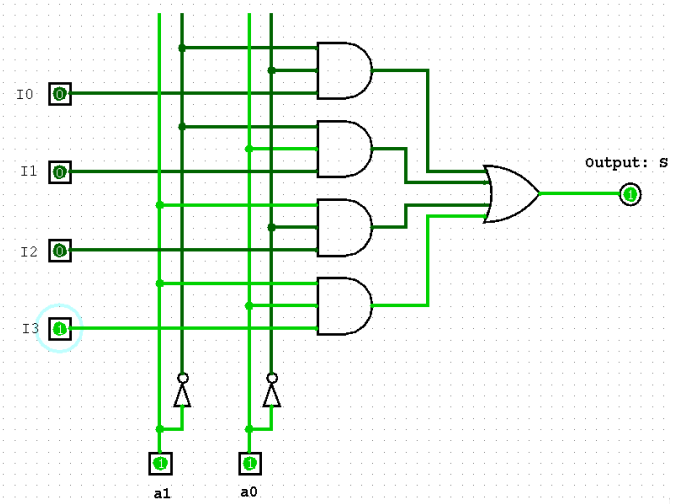


Figure 10.5: Le schéma logique du multiplexeur 4 vers 1.

Le schéma du multiplexeur sous **LogiSim** d'ou la sortie I2 est routé vers la sortie S , Figure 10.6:

Le schéma logique sous logisim du multiplexeur 4 vers 1.

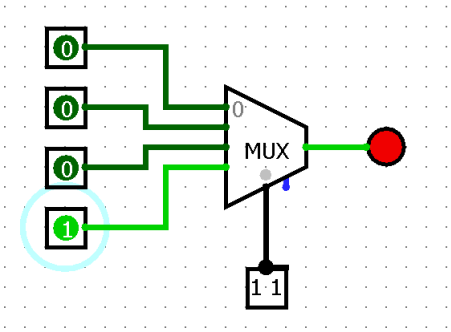


Figure 10.6: Le schéma logique sous logisim du multiplexeur 4 vers 1..

■

10.2.2 Le démultiplexeur

Comme leur nom l'indique, réalisent la fonction inverse des multiplexeurs. Ils possèdent une entrée d'information ou de données E, N entrées de sélection, et $S = 2^n - 1$ sorties. L'affichage d'une adresse sur les entrées de sélection permet de sélectionner la sortie vers laquelle l'entrée sera aiguillée. Le démultiplexeur peut, tout comme le multiplexeur, être doté d'une entrée de validation des sorties, Figure 10.7

Le Démultiplexeur 1 vers 2^n.

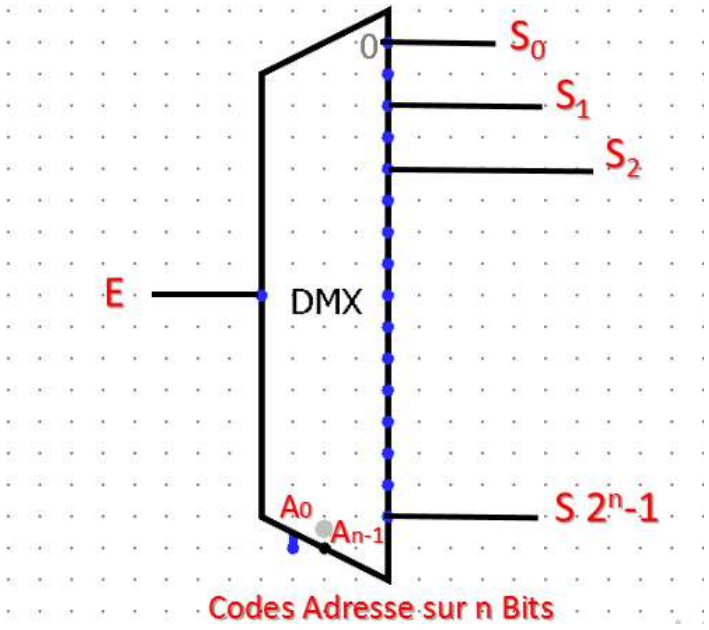


Figure 10.7: Le Démultiplexeur 1 vers 2^n .

■ **Example 10.3** Un Démultiplexeur 1 vers 4, Figure 10.8 . Sa table de vérité est égale à Table 10.2 :

Le Démultiplexeur 1 vers 4.

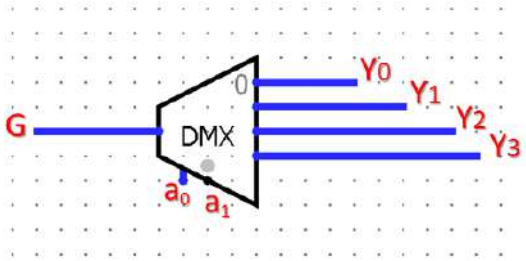


Figure 10.8: Le Démultiplexeur 1 vers 4 .

	a_0	a_1	$Y_0 = \overline{a_0}.\overline{a_1}.G$	$Y_1 = a_0.\overline{a_1}.G$	$Y_2 = \overline{a_0}.a_1.G$	$Y_3 = a_0.a_1.G$
0	0	0	G	0	0	0
1	0	1	0	G	0	0
2	1	0	0	0	G	0
3	1	1	0	0	0	G

Table 10.2: Table de vérité du multiplexeur 4 vers 1.

R L'utilisation première d'un démultiplexeur est la **conversion série/parallèle des données**.

■ **Example 10.4** Le schéma du Démultiplexeur sous LogiSim d'ou la sortie Y3 est selectionnée , Figure 10.9 Voici le schéma fondamental d'un démultiplexeur Figure 10.10.

Le schéma logique sous logisim du Démultiplexeur 1vers 4.

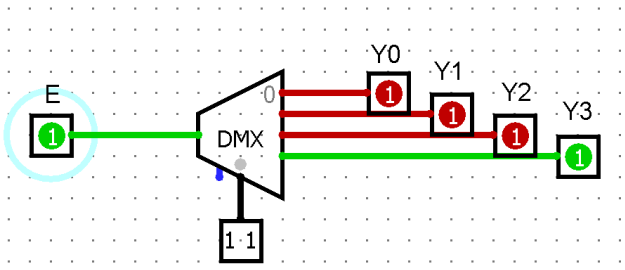


Figure 10.9: Le schéma logique sous logisim du Démultiplexeur 1 vers 4.

■

Le schéma logique du démultiplexeur 4 vers 1.

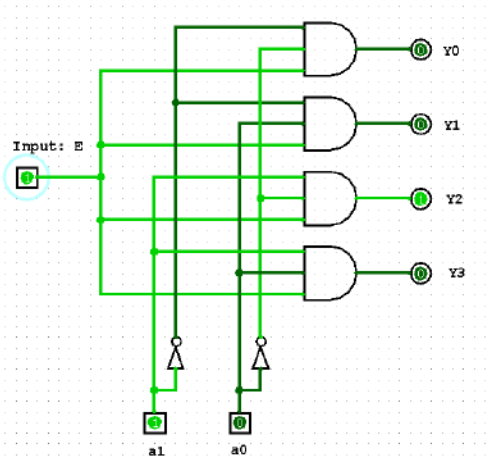


Figure 10.10: Le schéma logique du démultiplexeur 4 vers 1.

10.3 Les opérateurs de transcodage

Un opérateur de transcodage traduit une information dans un code donné (codeur/encodeur) ou au contraire, permet d'extraire une ou des informations à partir d'un code donné (décodeur) ou bien encore réalise la conversion d'un code vers un autre (convertisseur/transcodeur).

10.3.1 Le décodeur

C'est un opérateur à n entrées et 2^n sorties, Figure 10.11. Une sortie du décodeur est activée lorsqu'une configuration du code est affichée en entrée. Suivant le nombre de sorties, le décodeur peut décoder toutes les configurations possibles du code en entrée ou une partie seulement. Une seule sortie est activée à la fois. En plus des n entrées, certains décodeurs possèdent une ou plusieurs entrées de validation. Par exemple, si l'entrée de validation V vaut 0, alors le décodage est autorisé ; sinon (V = 1), les sorties sont inhibées et restent inactives. Ces entrées de validation permettent de grouper plusieurs décodeurs afin d'augmenter l'amplitude du décodage.

Shéma d'un Décodeur.

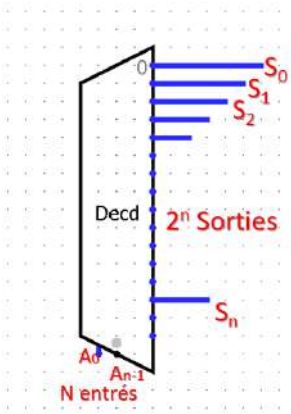


Figure 10.11: Le Décodeur 2^n Sorties.

Les décodeurs binaires

Ce type d'opérateur permet d'associer une sortie parmi 2^n avec une information d'entrée codée sur n bits.

■ **Example 10.5** Décodeur 1 vers 2 : Le décodeur 1 bit a donc une seule entrée A 0 et deux sorties S 0 et S 1, Figure 10.12. La table de vérité des sorties S0 et S 1 est la suivante, Table 10.3. Le schéma d'implémentation du décodeur sera alors celui, Figure 10.13.

Décodeur 1 vers 2.

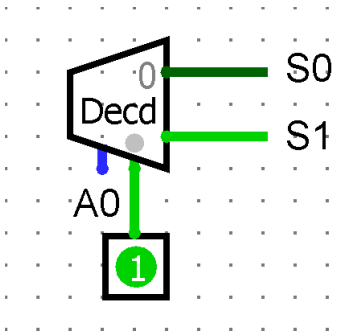


Figure 10.12: Décodeur 1 vers 2 .

A0	S0	S1
0	1	0
1	0	1

Table 10.3: Table de vérité du Décodeur 2 vers 1.

Logigramme de décodeur 1 parmi 2.

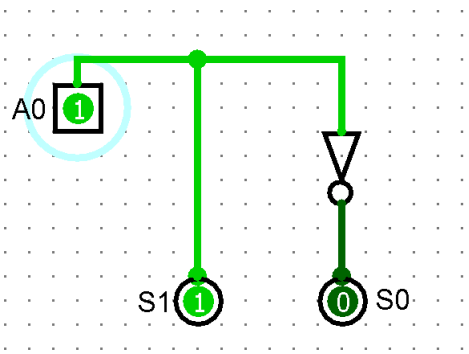


Figure 10.13: Logigramme de décodeur 1 parmi 2.

■

■ **Example 10.6** Décodeur 2 vers 4, Figure 10.14 présente le fonctionnement d'un décodeur binaire 2 vers 4. La table de vérité du Décodeur 2 vers 4 est la suivante, Table 10.4. Le schéma d'implémentation du décodeur sera alors celui, Figure 10.15.

Décodeur 2 vers 4.

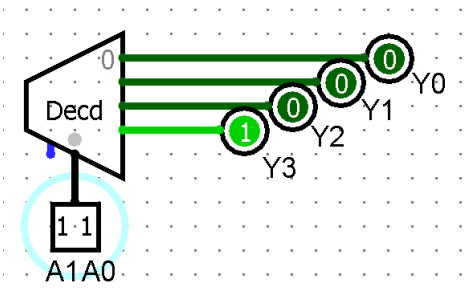


Figure 10.14: Décodeur 2 vers 4 .

N	a_1	a_0	Y_0=$\overline{a_0}.\overline{a_1}$	Y_1=a$_0$.$\overline{a_1}$	Y_2=$\overline{a_0}.a_1$	Y_3=$a_0.a_1$
0	0	0	1	0	0	0
1	0	1	0	1	0	0
2	1	0	0	0	1	0
3	1	1	0	0	0	1

Table 10.4: Table de vérité du Décodeur 2 vers 4.

Logigramme de décodeur 2 parmi 4.

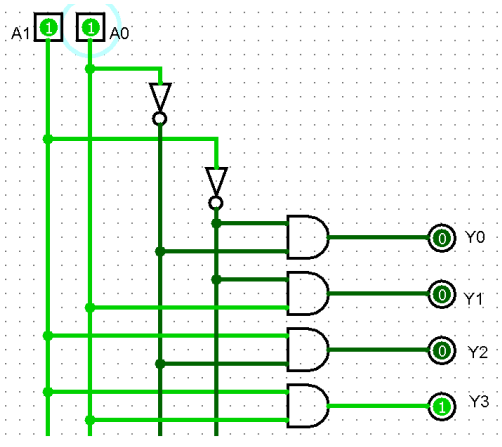


Figure 10.15: Logigramme de décodeur 2 parmi 4.

■

■ **Example 10.7** Décodeur 3 vers 8, Figure 10.16 présente le fonctionnement d'un décodeur binaire 3 vers 8. La table de vérité Table 10.5 comporte 8 sorties qui correspondent aux mintermes des variables d'entrée. Le schéma d'implémentation du décodeur sera alors celui, Figure 10.17.

Décodeur 3 vers 8.

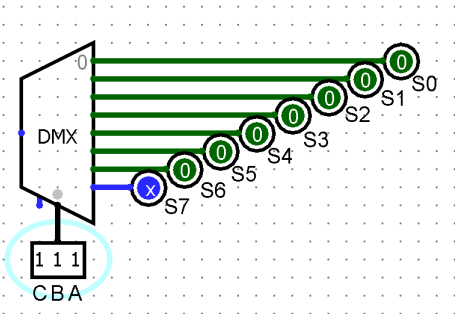


Figure 10.16: Décodeur 3 vers 8 .

C	B	A	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Table 10.5: Table de vérité du Décodeur 3 vers 8.

Logigramme de décodeur 3 parmi 8.

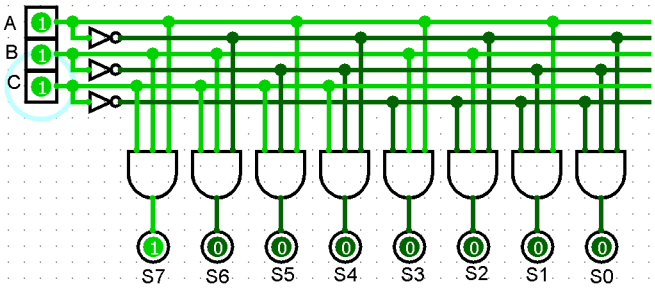


Figure 10.17: Logigramme de décodeur 3 parmi 8.

■

Le décodeur BCD

Le code BCD (Binary-Coded Decimal) est utilisé pour coder les dix chiffres décimaux. Ce code à 4 bits laisse donc inutilisées 6 combinaisons sur les 16 possibles, Figure 10.18 . Lorsqu'une combinaison, comprise entre 0 et 9, est appliquée sur les entrées ABCD (A est le bit de poids fort), la sortie correspondante est validée. Les sorties restent au repos (niveau 1 par exemple) dans le cas où une combinaison comprise entre 10 et 15 est appliquée sur les entrées, La table de vérité Table 10.6 comporte:

Le décodeur BCD.

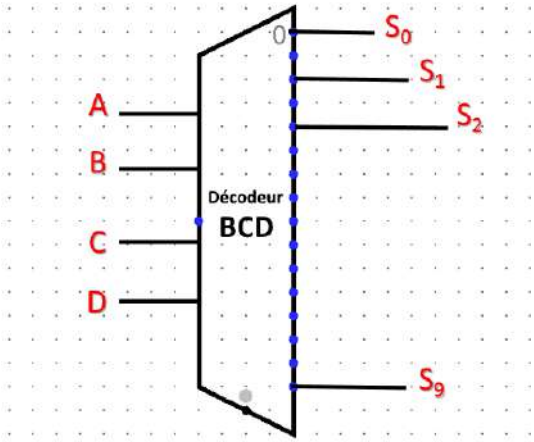


Figure 10.18: Le décodeur BCD.

A	B	C	D	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9
0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1	1	1
0	1	0	1	1	1	1	1	1	0	1	1	1	1
0	1	1	0	1	1	1	1	1	1	0	1	1	1
0	1	1	1	1	1	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0

Table 10.6: Table de vérité du décodeur BCD.

10.3.2 L'encodeur (Codeur)

Son fonctionnement : lorsqu'une entrée est activée, les sorties affichent la valeur correspondant au numéro de l'entrée dans le code binaire choisi (convertisseur Décimal/Binaire). Une seule entrée du codeur doit normalement être activée à la fois. La priorité ne sert que quand plusieurs entrées sont à 1 en même temps. Le circuit donne alors l'adresse de l'entrée dont le rang est le plus élevé.

■ **Example 10.8** Ce codeur reçoit un chiffre décimal sur une des dix entrées et génère l'équivalent binaire sur les sorties A0 à A3, Figure 10.19. Une seule entrée doit être active à la fois.

Encodeur décimal vers binaire (10 entrées vers 4 sorties).

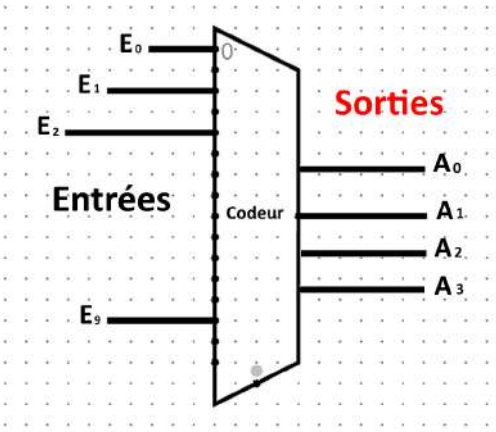


Figure 10.19: Encodeur décimal vers binaire (10 entrées vers 4 sorties).

La table de vérité Table 10.7 comporte:

Entré=1	A_3	$A2_1$	A_1	A_0
E_0	0	0	0	0
E_1	0	0	0	1
E_2	0	0	1	0
E_3	0	0	1	1
E_4	0	1	0	0
E_5	0	1	0	1
E_6	0	1	1	0
E_7	0	1	1	1
E_8	1	0	0	0
E_9	1	0	0	1

Table 10.7: Encodeur décimal (10 entrées vers 4 sorties).

■ **Example 10.9** Encodeur binaire 8 vers 3, Figure 10.20. La table de vérité Table 10.8 comporte:

Encodeur décimal vers binaire (8 entrées vers 3 sorties).

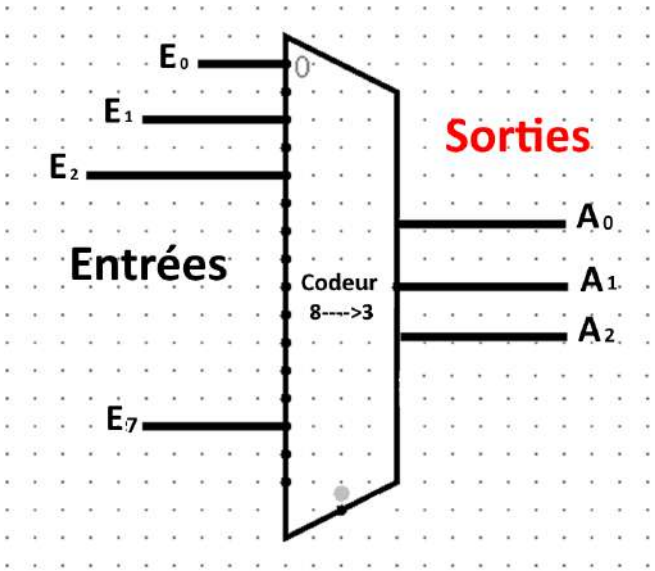


Figure 10.20: Encodeur décimal vers binaire (8 entrées vers 3 sorties).

Entré=1	$A2_1$	A_1	A_0
E_0	0	0	0
E_1	0	0	1
E_2	0	1	0
E_3	0	1	1
E_4	1	0	0
E_5	1	0	1
E_6	1	1	0
E_7	1	1	1

Table 10.8: Encodeur décimal (8 entrées vers 3 sorties).

10.3.3 Les transcodeurs

Ces opérateurs permettent de convertir un nombre écrit dans un code C1 vers un code C2

■ **Example 10.10** Le transcodeur BCD/7 segments permet de commander un afficheur alphanumérique possédant 7 segments (des diodes électroluminescentes, par exemple). Cet afficheur permet la visualisation des chiffres 0 à 9 codés en binaire naturel sur 4 bits A, B, C, et D, où D représente le bit de poids le plus faible.

Le transcodeur BCD/7 segments.

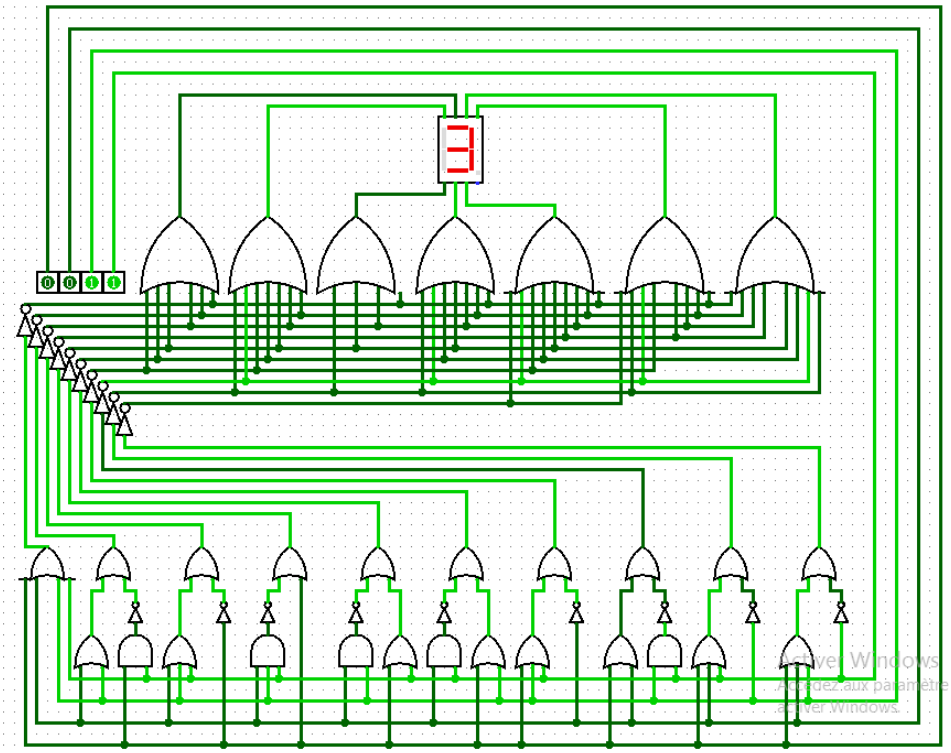


Figure 10.21: Transcodeur BCD/7 segments.

10.4 Les opérateurs de comparaison

10.4.1 Définition

Les comparateurs sont une autre grande famille de circuit arithmétique. On désigne par opérateur de comparaison ou comparateur un opérateur capable de détecter l'égalité ou l'inégalité de 2 nombres (comparateur simple) et éventuellement d'indiquer le plus grand ou le plus petit (comparateur complet), Figure 10.22.

Shéma d'un Comparateur n bits .

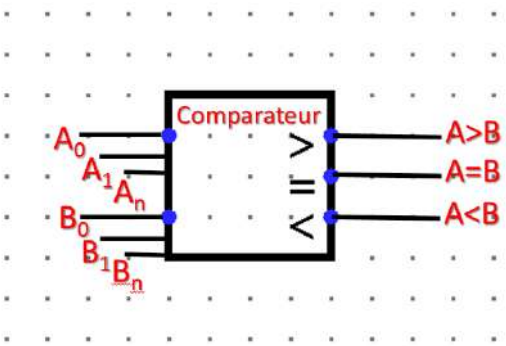


Figure 10.22: Comparateur n Bits .

■ **Example 10.11** Pour comprendre le principe, on va réaliser un comparateur simple permettant de comparer deux mots de 1 bit, Figure 10.23.

Comparateur deux mots de 1 bit

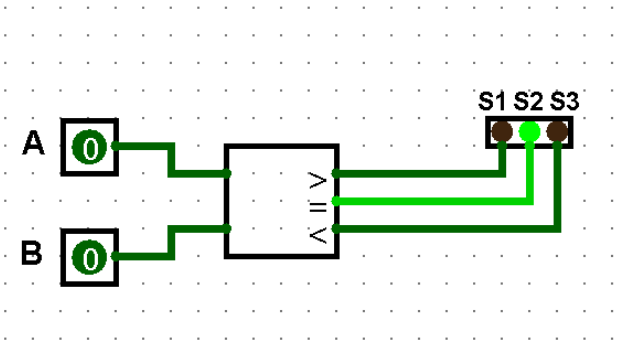


Figure 10.23: Comparateur 1 bits .

La table de vérité d'un tel comparateur est Table 10.9 :

A	B	$S1 : A > B$	$S1 : A = B$	$S1 : A < B$
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Table 10.9: comparateur deux mots de 1 bit

Le schéma d'implantation de ce comparateur 2 bits sera celui de Figure 10.24 :

Logigramme du comparateur de 2 mots de 1 bit

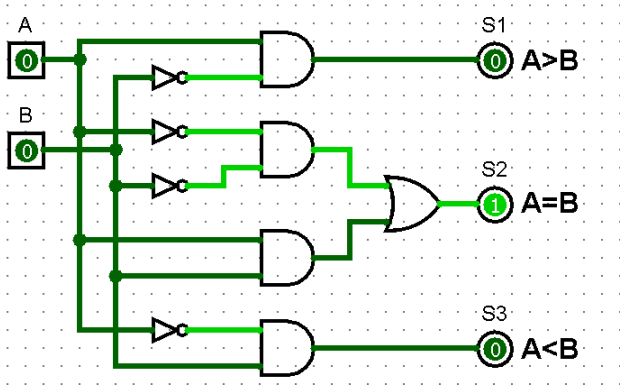


Figure 10.24: Logigramme du comparateur de 2 mots de 1 bit.

■ **Example 10.12** Le circuit suivant est un comparateur de deux mot de deux bits : Figure 10.25 :

Comparateur deux mots de 2 bit

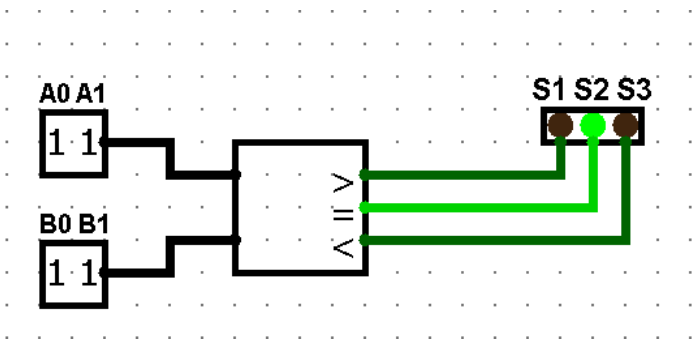


Figure 10.25: Comparateur 2 bits .

La table de vérité Table 10.10 comporte:

A_0	A_1	B_0	B_1	$S1 : A > B$	$S1 : A = B$	$S1 : A < B$
0	0	0	0	1	0	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	0	1	0
0	1	0	1	1	0	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	0	1	0
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	0	1	1	0	0	1
1	1	0	0	0	1	0
1	1	0	1	0	1	0
1	1	1	0	0	1	0
1	1	1	1	1	0	0

Table 10.10: Table de Vérité de 2 mots de 2 bit

10.5 Les opérateurs arithmétiques

10.5.1 Les additionneurs

L'addition est la fonction arithmétique la plus couramment rencontrée dans les systèmes numériques. En effet, la conception d'additionneurs est importante pour effectuer des multiplications et des divisions. Deux architectures d'additionneurs de nombres binaires sont présentées : le Demi additionneur et l'additionneur Complets.

Le Demi Additionneur

Un Demi-Additionneur est un circuit qui réalise l'addition de deux bits A et B pour générer leur somme S et leur retenue C (Carry) comme le montre la table de vérité, Table 10.12 :

A	B	S	C
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

Table 10.11: Table de vérité du DemiAdd.

Les fonction S et C s'expriment sous forme algébrique par :

$$S = A \oplus B$$

$$C = A.B$$

Le schéma d'implantation du Demi-Additionneur sera Figure 10.26, Figure 10.27 :

Le Demi-Additionneur

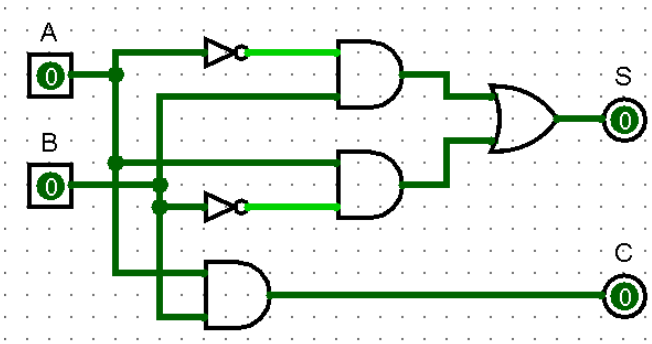


Figure 10.26: Le Demi-Additionneur

Le Demi-Additionneur

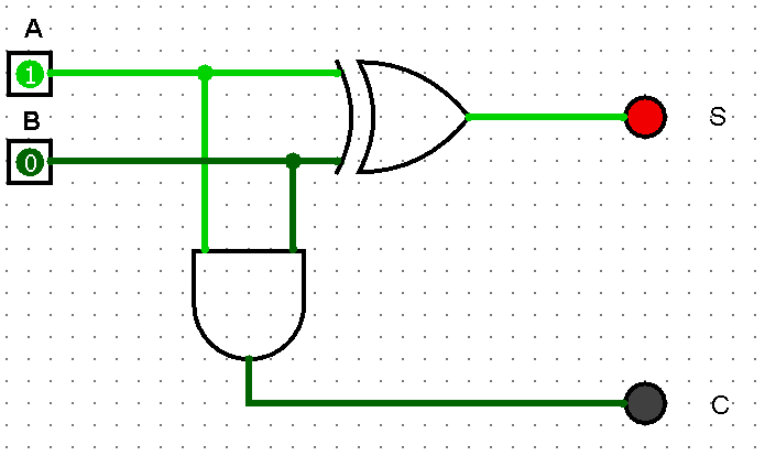
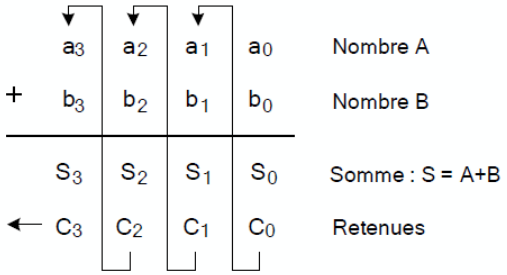


Figure 10.27: Le Demi-Additionneur

L'additionneur complet 1 bit

Pour effectuer une addition de deux nombres binaires de n bits, on additionne successivement les bits du même poids en tenant compte de la retenue de l'addition précédente comme le montre l'exemple suivant :



Il faut concevoir une cellule élémentaire appelée additionneur complet qui permet de réaliser l'addition des bits a_i et b_i en plus de la retenue $C_i - 1$ de l'addition précédente. Un tel circuit est définit par la table de vérité , Table 10.12 :

Les fonction S_i et C_i pour l'additionneur complet s'expriment sous forme algèbrique par :

$$S_i = \mathrm{A} \oplus B \oplus R$$

$$C_i = \mathrm{A.B} + (A \oplus B).R$$

Le schéma d'implantation de l'additionneur complet sera Figure 10.28, :

A_i	B_i	C_{i-1}	S_i	C_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 10.12: Table de vérité du Full Add.

Additionneur complet

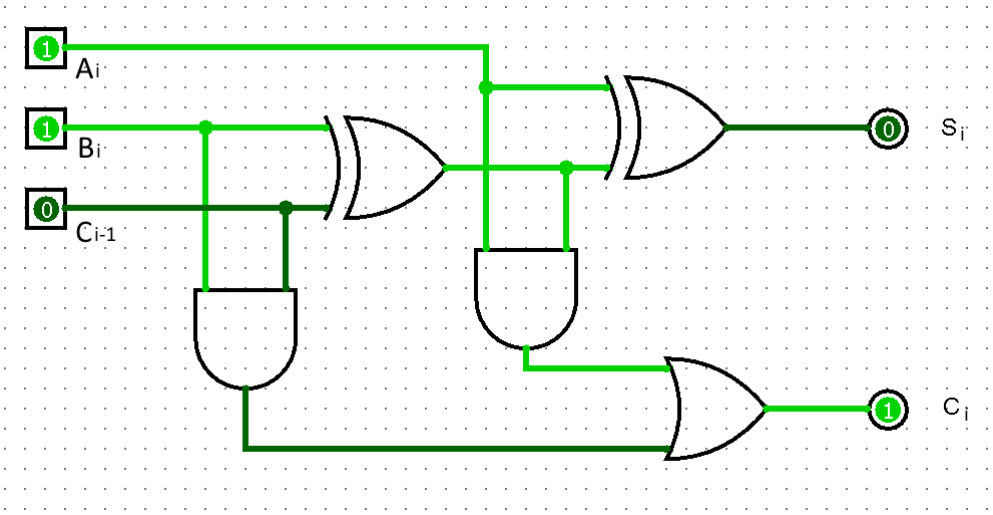


Figure 10.28: Additionneur complet

Additionneur complet n bits

L'addition de deux mots de n bits nécessite n additionneurs. La retenue se propage des éléments binaires de poids le plus faible vers les éléments binaires de poids le plus fort comme le montre Figure 10.29 pour un additionneur 4 bits.

R

Cette architecture est intéressante d'un point de vue matériel car elle est répétitive. Par contre, le résultat obtenu dépend du nombre d'additionneurs donc de la taille des mots à additionner. La retenue R_1 est délivrée après la première addition et ainsi de suite.

Additionneur complet n bits

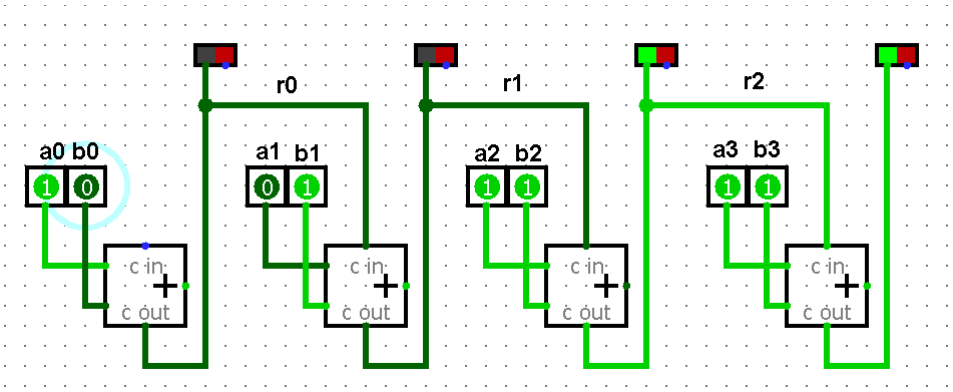


Figure 10.29: Additionneur complet n bits

10.5.2 Les soustracteurs

Pour une soustraction A B , on peut adopter la même approche que pour l'addition. On commence par définir l'opérateur binaire de base et on l'utilise pour réaliser des soustractions de nombres binaires. En pratique, se pose le problème de la représentation des nombres signés dans le cas où B > A . Pour résoudre ce problème, on convient d'une représentation des nombres négatifs, la soustraction est alors ramenée à une addition. La représentation généralement utilisée est celle du complément vrai ou complément à 2.

Le Demi-Soustracteur

Le demi-soustracteur est défini par la table de vérité suivante (le bit B_i est retranché au bit A_i), Table 10.13 :

A_i	B_i	D_i	R_i
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Table 10.13: Table de vérité du Demi-Soustracteur.

Les fonction D_i et R_i s'expriment sous forme algébrique par :

$$D_i = \mathrm{A}_i \oplus B_i$$

$$R_i = \overline{A_i} + B_i$$

Le schéma d'implantation du demi-soustracteur sera celui de la Figure 10.30 :

Le Demi-Soustracteur

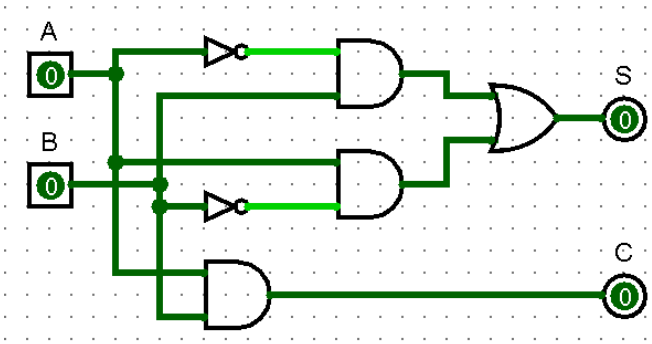


Figure 10.30: Le Demi-Doustracteur

Soustracteur Complet

Pour obtenir un soustracteur binaire complet il faut prendre en compte l'éventuelle retenue précédente $R_i - 1$. La table de vérité est Table 10.14 :

$R_i - 1$	A_i	B_i	D_i	R_i
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

Table 10.14: Table de vérité du Soustracteur Complet.

Les fonction D_i et R_i s'expriment sous forme algébrique par :

$$D_i = \overline{R_{i-1}}.(A_i \oplus B_i).R_{i-1}.\overline{(A_i \oplus B_i)} = (A_i \oplus B_i) \oplus R_{i-1}$$

$$R_i = \overline{R_{i-1}}.\overline{A_i}.Bi + R_{i-1}.\overline{A_i}.\overline{B_i}$$

Le schéma d'implantation du soustracteur complet sera celui de la Figure 10.31 :

Le Soustratceur Complet

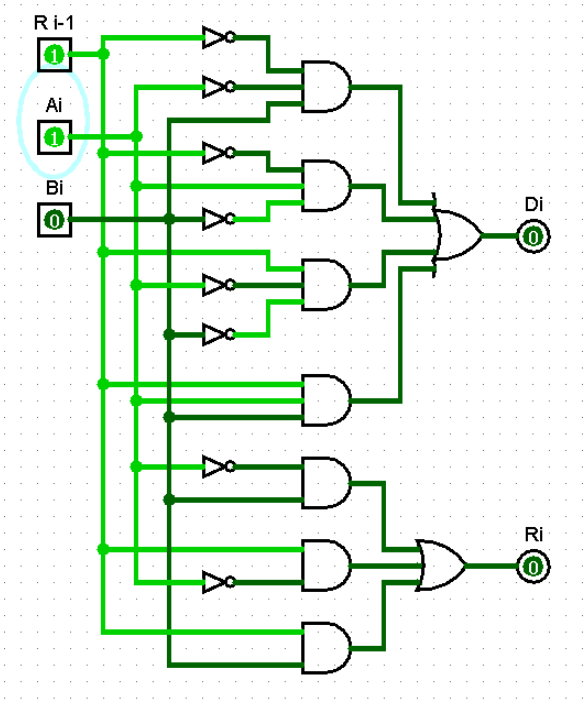


Figure 10.31: Le Soustratceur Complet

Additionnrur / Soustracteur

Le circuit de 5 bits ci-dessous effectue une somme ou une différence suivant la valeur de la commande Cmd, Figure 10.32. Si Cmd vaut 0, le circuit calcule la somme A + B. Si, au contraire, Cmd vaut 1, le circuit calcule la différence A - B. En effet, chacune des portes xor effectue la négation ou non d'une entrée B i suivant la valeur de Cmd.

Additionnrur / Soustracteur

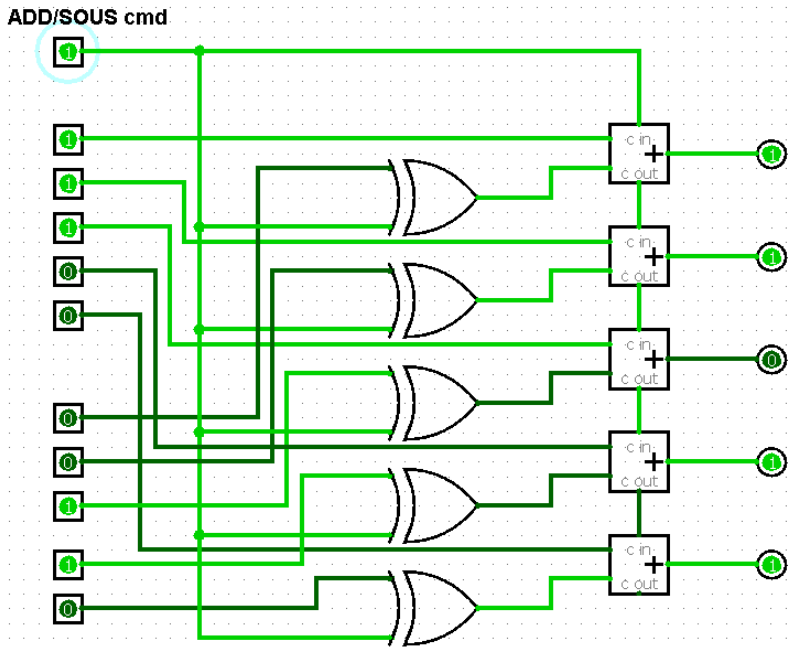


Figure 10.32: Additionnrur / Soustracteur

Bibliography

Bibliography

[**CAR 15**] Olivier Carton . *Circuits et architecture des ordinateurs*.Université Paris Diderot, 2015.

[**MUL 15**] Didier Müller. *Chapitre 1, Une petite histoire de l'informatique*. L'informatique au lycée,2015.

[**MUS 15**] Luc MUSEUR . *Electronique numérique Logique combinatoire et séquentielle*.Université Paris 13, Institut Galilée, 2015.

[**RIS 15**] Vincent Risch. *Eléments d'Architecture des Ordinateurs*.Département d'Informatique Institut Universitaire de Technologie Université de la Méditerranée, 2015.

[**TRO 15**] Alain TROESCH. *Cours d'informatique commune*.Lycée Louis-Le-Grand, Paris, 2015.

[**ALE 14**] C.Alexandre. *Circuits numériques : 1ère partie* .Conservatoire National des Arts et Métiers, 2014.

[**BUI 14**] Jean-Christophe Buisson. *Introduction a l'architecture des ordinateurs*.University of Toulouse, INPT, IRIT (ENSEEIHT), 2014.

[**DAN 14**] Jean-Luc Danger, Guillaume Duc. *Processeurs et Architectures Numériques*.groupe SEN, dépt Comelec, Télécom-ParisTech, 2014.

[**FRA 14**] Séverine Fratani, Peter Niebert. *Cours d'Architecture des ordinateurs* .Aix Marseille Université - Laboratoire d'Informatique Fondamentale de Marseille, 2014.

[**LAS 14**] Tayari Lassaadi. *Support de Cours Systemes Logiques*.Institut Supérieur des Etudes Technologiques de Gabès, 2014.

[**RIG 14**] Yannis Delmas-Rigoutsos. *Histoire de l'informatique, d'Internet et du Web*. Université de Poitiers, 2014.

[**ARO 13**] S. Aroussi. *Systèmes de Numération*.Faculté des Sciences, Université SAAD DAHLAB de Blida, 2013.

[**FAR 13**] Jacques Farré. *Informatique Générale*. Université de Nice Sophia Antipolis, 2013.

[MEL 13] J. Mellac. *INFORMATIQUE, d'Internet et du Web*. mpsi mathematiques et informatique, 2013.

[**MEY 13**] Luc De Mey. *Mathématiques appliquées à l'informatique*.http://www.courstechinfo.be, 2013.

[**BUL 12**] Bullynck. *Histoire de l'Informatique*. ufr6.univ-paris8, 2012.

[**DOW 12**] Gilles Dowek. *Informatique et sciences du numérique*.Spécialité ISN en terminale S, professeur au Collège de France, 2012.

[**REB 12**] Djamal Rebaïne. *La Présentation des données*.professeur, Département d'informatique et de mathématique Université du Québec , 2012.

[**WAL 12**] Yann Walkowiak. *Architecture - codage*.IUT de LAVAL Département Informatique, 2012.

[**BOU 11**] S.Bouam. *Cours Structure Machine*.LMD informatique/mathématique 1ère année, 2011.

[**FOU 11**] Pascal Fougeray, Bertrand Dupouy. *Architecture des ordinateurs*,2011.

[**LAZ 11**] Emmanuel Lazard. *Architecture de l'ordinateur* .Université Paris-Dauphine, 2011.

[**OUM 11**] Abdelmajid Oumnad. *Bases des systèmes Numériques*.Ecole Mohammadia des Ingénieurs , 2011.

[**ISE 10**] ISET Mahdia. *Architecture Des Ordinateurs Technologies et concepts*.Institut Supérieur des Etudes Technologiques de Mahdia, 2010.

[**MEH 10**] Vincent Boyer , Jean Méhat. *Introduction à l'Informatique*.Université Paris 8 Département Informatique, 2010.

[**MES 10**] Etienne Messerli, Yves Meyer . *Electronique Numérique 1er tome Systèmes combinatoires*.Institut REDS HEIG-VD Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud, 2010.

[**REM 10**] Patrice Remaud. *Une histoire de l'informatique*. LIAS-ENSIP(Laboratoire d'Informatique et d'Automatique pour les Systèmes – Ecole Nationale Supérieure d'Ingénieurs, 2010.

[**AMS 09**] AMSI Chapitre 1. *L'histoire de l'ordinateur*. AMSI, 2009.

[**DIO 09**] Camille Diou. *Cours d'électronique numérique* .Maître de Conférences Laboratoire Interfaces Capteurs etMicroélectronique Université Paul Verlaine–Metz, 2009.

[**DOU 09**] Catherine Douillard, André Thépaut. *Logique combinatoire et circuits MOS* .Ecole Nationale Supérieure des Télécommunications de Bretagne, 2009.

[**GER 09**] Cécile Germain, Daniel Etiemble. *Architecture des Ordinateurs Première partie* .Licence d'Informatique - IUP Miage - FIIFO, 2009.

[**ALM 08**] Guy Almouzni. *Eletronique Numérique*.EISTI , 2008.

[**BLA 08**] Gérard Blanchet, Bertrand Dupouy. *Architecture des ordinateurs*. Telecom ParisTech, 2008.

[**CAZ 08**] Alain Cazes, Joëlle Delacroix. *Architecture des machines et des systémes informatiques*.3eme édition Dunod, 2008.

[**DUR 08**] D. Durier. *Elements de Logique Combinatoire*.Lycée Jules Ferry – Versailles , 2008.

[**GER 08**] Pierre Gérard. *Informatique* .Assistant Professor ,University of Paris 13, 2012.

[**POI 08**] Jean-Marc Poitevin . *Aide mémoire éléctronique analogique et numérique*.Dunod Edition, Paris, 2008.

[**GOU 07**] Frédéric Goualard. *Architecture des ordinateurs et systèmes d'exploitation*. Laboratoire d'Informatique de Nantes-Atlantique, 2007.

[**PEL 07**] F. Pellegrini . *Architecture des ordinateurs*.Université Bordeaux 1, 2007.

[**MER 05**] A.Merazguia. *Architecture des Ordinateurs I*. Univ OEB, 2004/2005.

[**PON 05**] G.Pondemer Dewulf. *Cours 1 : Architecture d'un ordinateur*. CIAN - Architecture, 2005.

[**PEC 04**] Lancelot Pecquet. *Architecture des ordinateurs Systémes d'exploitation*. Université Paris XII, DEUG MIAS, 2004.

Annexes

Logiciels de Simulation

Il èxiste plusieurs outils dans le domaine de l'électronique numérique pour l'enseignement permettant la création de schémas électroniques (Automation Studio, Circuit Builder, DigSim, LogicSim, MMLogic, SimulPortes, Multisim). Nous avons choisi des outils Open Source afin de pouvoir tester, implémenter et exécuter les differents circuits combinatoires vus dans cet ouvrage. Ces logiciels ont les caractéristiques suivantes :

- Disponibilité d'une version Open Source ;
- Outil à visée pédagogique ;
- Simple à utiliser et à manipuler ;
- Disposant d'une interface graphique pour la création de schémas électroniques ;
- Possibilité de créer et d'exécuter des simulations ;
- Possibilité de récupérer les schémas créés dans des fichiers à part ;
- Eventuellement, possibilité de faire des évaluations sur les schémas ou sur les connaissances des apprenants.

Nous avons tout de même retenu les logiciels particuliers LogiSim et CEDAR Logic qui permettent en plus, par rapport aux autres logiciels, d'enregistrer les circuits électroniques dans un format **standard XML**. Nous verrons dans la section suivante l'utilité du logiciel.

CEDAR Logic

CEDAR Logic est un logiciel de simulation de circuits logiques combinatoires (portes) et séquentiels (bascules), Figure 10.33. Particulièrement utilisé lors de l'apprentissage des notions fondamentales pour les circuits logiques.

Interface de CEDAR

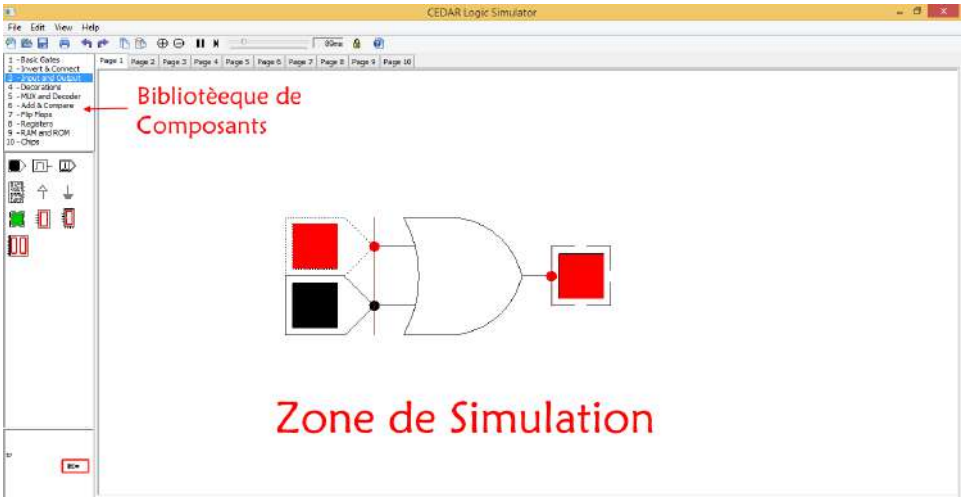


Figure 10.33: Interface de CEDAR

Logisim

LogiSim est un outil éducatif de source libre et ouverte qui vous aidera à concevoir et simuler les circuits logiques numériques qu'il s'agisse de circuits combinatoires ou séquentiels. Il possède

une interface graphique assez intuitive, Figure 10.34 et une libraire de composants bien fournie, allant des portes logiques de base à des éléments de calcul, des compteurs ou différents types de mémoire. L'édition d'un circuit se fait simplement de manière graphique en sélectionnant dans la liste des composants disponibles les éléments à placer sur le circuit. La liaison se fait en tirant les fils à la souris entre les points de contacts des différents éléments. Par défaut, le logiciel active le mode simulation pendant l'édition. Cela signifie que les entrées du circuit sont prises en compte et les sorties des portes et composants recalculées en permanence, Figure 10.34. Logisim est assez simple pour faciliter apprendre les concepts les plus fondamentaux liés aux circuits logiques.

Interface de Logisim

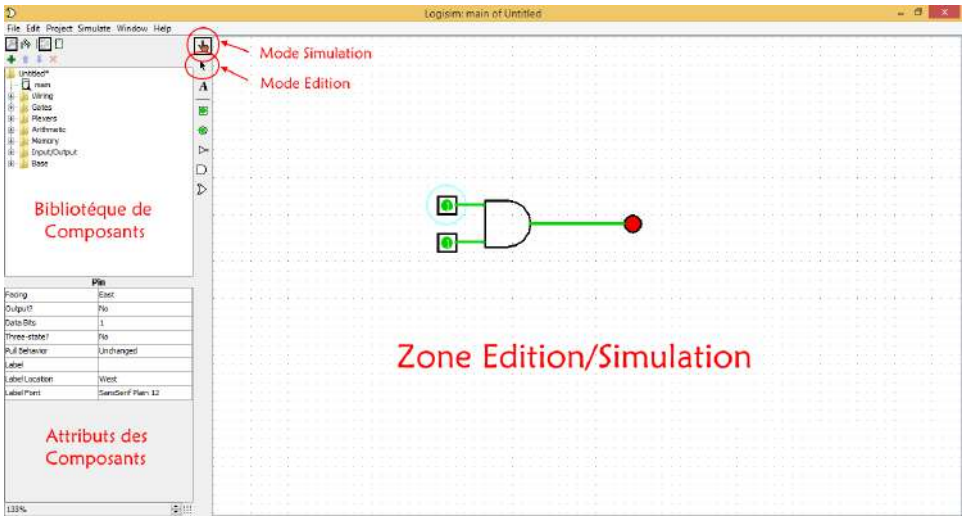


Figure 10.34: Interface de Logisim

Un des avantages de cet outil est qu'il permet de sauvegarder les circuits créés dans un format standard XML (Extensible Markup Language en anglais). Logisim est un outil pédagogique utilisé dans des dizaines d'universités à travers le Monde. De la capacité d'établir de plus grands circuits de plus petits subcircuits, et de dessiner des paquets de fils avec une drague simple de souris, Logisim est employé pour concevoir et simuler les unités centrales de traitement entières pour des buts éducatifs.

- Il est libre (Logisim est l'ouvrir-source (le GPL).)
- Il fonctionne sur n'importe quelle machine soutenant Java 1.4 ou plus tard ; des versions spéciales sont libérées pour MaOS X et Windows. La nature de croix-plate-forme est importante pour les étudiants qui ont une série de les systèmes informatiques de maison/dortoir.
- L'interface de schéma est basée sur une barre porte-outils intuitive. Aide de code à couleurs de fils en simulant et en corrigeant un circuit.
- L'outil de câblage dessine les fils horizontaux et verticaux, automatiquement se reliant aux composants et à d'autres fils. Il est très facile de dessiner des circuits !
- Des circuits réalisés peuvent être sauvés dans un dossier, être exportés vers un dossier de GIF, ou être imprimés sur un imprimeur.
- Des dispositions de circuit peuvent être employées en tant que « subcircuits » d'autres circuits, laissant pour la conception de circuit hiérarchique.
- Les composants de circuit inclus incluent des entrées et des sorties, des portes, des multiplexeurs, des circuits arithmétiques, des bascules, et mémoire de RAM.

- Le module inclus « d'analyse combinationnelle » tient compte de la conversion entre les circuits, les tables de vérité, et les expressions booléennes.

Mode Edition

1. Pour utiliser le mode édition, il faut simplement sélectionner la flèche comme indiqué en haut de la figure 1.
2. On peut alors choisir un composant dans la bibliothèque sur la gauche. Pour l'ajouter dans son schéma, il suffit de cliquer sur le composant désiré, puis de cliquer sur le schéma.
3. Chaque composant que vous utiliserez aura des attributs modifiables dans la zone inférieur gauche de Logisim. Par exemple si l'on pose une porte AND, on peut modifier le nombre de signaux qu'elle prend en entrée, ou encore mettre un inverseur sur une de ses entrées.
4. Il est aussi possible de faire des copier/coller d'un ou plusieurs composants. Dans ce cas, les composants conserverons aussi tous les attributs préalablement définis.
5. Voici un descriptif des éléments que vous aller avoir besoin pour ce laboratoire :

    - Pour les entrées, l'élément Pin de Wiring.
    - Pour les sorties, l'élément Pin de Wiring avec l'attribut output?=yes. portes logiques sont présentent dans le répertoire Gates.
    - Le splitter de Wiring.
    - Le ground et power de Wiring.

6. Une fois que l'on a posé tous les composants, il faut alors les connecter. Pour cela il suffit de placer le curseur avec la souris sur un des ports à connecter et, en gardant pressé le bouton gauche de la souris, le déplacer jusqu'au port de destination.

Mode simulation

Logisim est capable de simuler le circuit en affichant les valeurs des signaux directement sur le schéma. L'utilisateur peut alors définir les valeurs des bits en entrée et observer la réaction du design.

1. Pour utiliser le mode simulation, il faut sélectionner la main en haut à gauche de Logisim (cf figure 1).
2. Il est alors possible de contrôler l'état des différentes entrées en cliquant directement dessus. Le X bleu des Pin d'entrées représente l'état haute impédance. Dans ce laboratoire, nous travaillerons uniquement avec des états haut ou bas. Pour supprimer cet état de haute impédance, il faut modifier les attributs de ces Pin d'entrées de façon à ce que la ligne Three-state) soit égale à No.
3. En cliquant sur une entrée, la valeur doit alterner entre '0' ou '1'.
4. Voici un descriptif des couleurs utilisées pour les signaux en mode simulation, Figure 10.35 :

    - Gris : La taille du fil est inconnue. Le fil n'est relié à aucune entrée ou sortie.
    - Bleu : Le fil comporte une valeur, cependant elle est inconnue.
    - Vert foncé : Le fil comporte la valeur '0'.
    - Vert clair : Le fil comporte la valeur '1'.
    - Noir : Le fil comporte plusieurs bits (BUS).
    - Orange : Les composants reliés au fil n'ont pas la bonne taille.

5. Testez le bon fonctionnement de votre additionneur 1 bit.

Analyse d'une fonction combinatoire avec logisim

Marche à suivre pour générer le circuit correspondant à:

Couleurs des fils en simulation

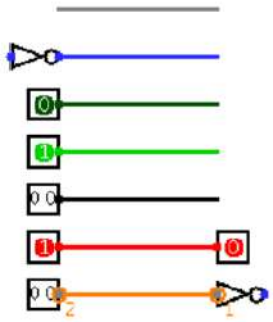


Figure 10.35: Couleurs des fils en simulation

$$S = ab + \overline{a} + b$$

1. aller dans la partie Analyse combinatoire (Windows -> Combinational Analysis), Figure 10.36.

Menu Combinational Analysis

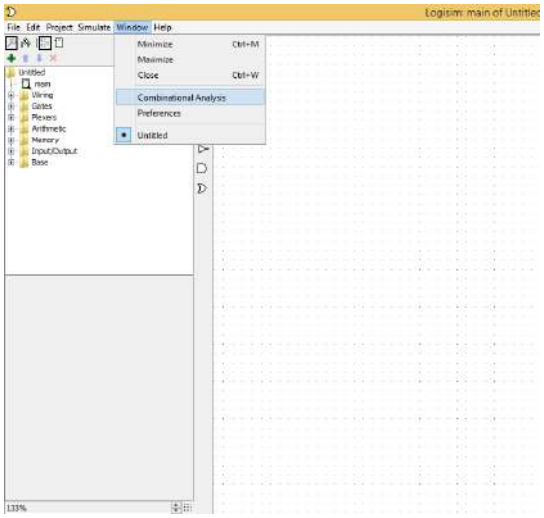


Figure 10.36: Menu Combinational Analysis

2. Définir les entrées (Inputs), Figure 10.37.
3. Définir la (les) sortie(s) (Outputs), Figure 10.38.
4. Ecrire votre expression, Figure 10.39.
5. Faire Build Circuit pour générer le circuit correspondant à votre expression, Figure 10.40.

Etablir un Circuit Combinatoire à partir d'une table de Vérité

Marche à suivre pour générer le circuit correspondant à la table de vérité suivant, Table 10.15 :

(a) Aller dans la partie Analyse combinatoire (Windows -> Combinational Analysis), Figure 10.36.

Définir les entrées

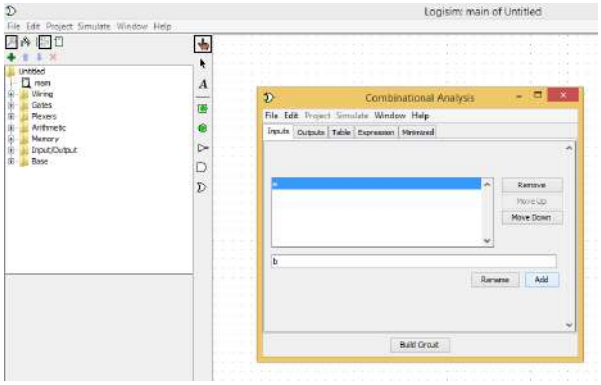


Figure 10.37: Définir les entrées

Définir la sortie

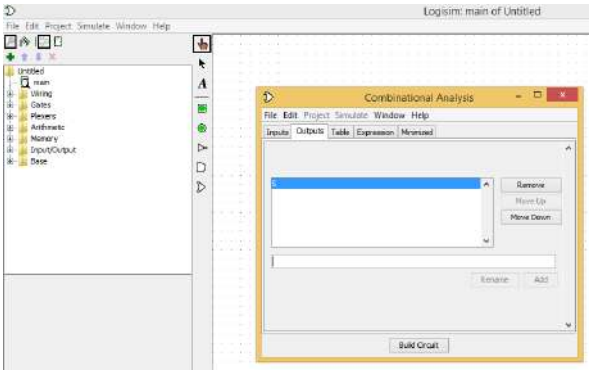


Figure 10.38: Définir la sortie

Expression

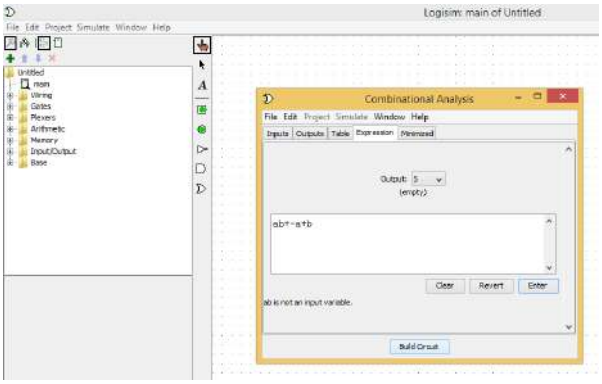


Figure 10.39: Expression

(b) Définir les entrées (Inputs), Figure 10.37.

Générer le circuit correspondant à expression

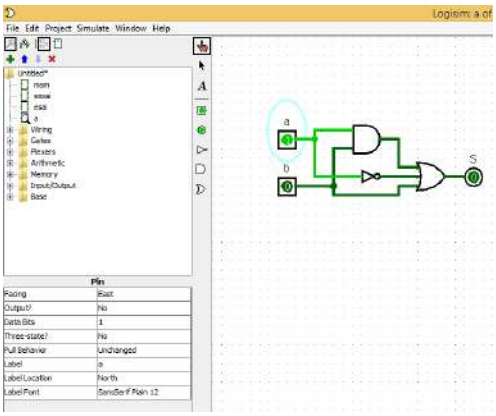


Figure 10.40: Générer le circuit correspondant à expression

A	B	C	F (a,b,c)
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Table 10.15: Une table de véritè

(c) Définir la (les) sortie(s) (Outputs), Figure 10.38.
(d) Générer la table de vérité de F, Figure 10.41.
(e) LogiSim Génére automatiquement les Tableaux de Karnaughts, Figure 10.42.
(f) Faire Build Circuit pour générer le circuit correspondant à la table de vérité de F, Figure 10.43.
[**einstein**]

Générer la table de vérité de F

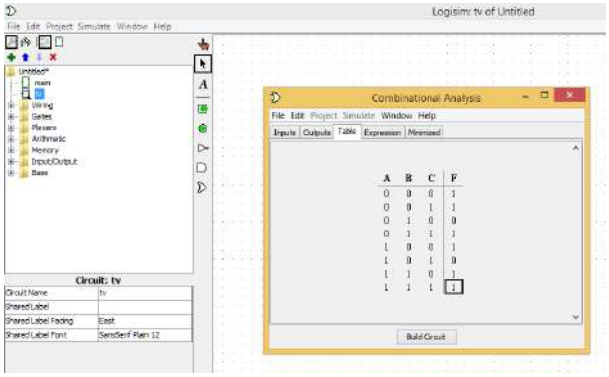


Figure 10.41: Générer la table de vérité de F

Tableaux de Karnaughts de F

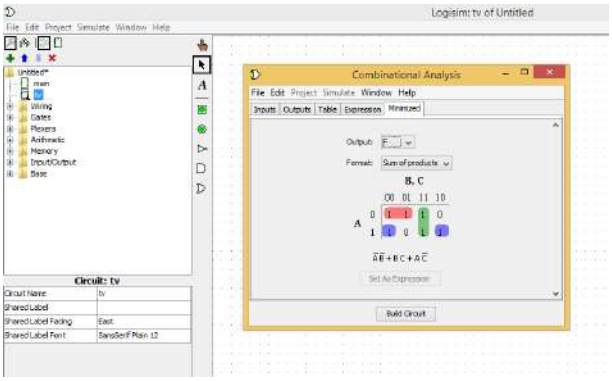


Figure 10.42: Tableaux de Karnaughts de F

Générer le circuit correspondant à la table de vérité de F

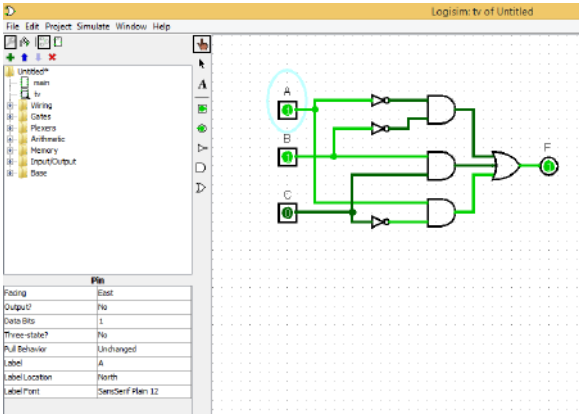


Figure 10.43: Générer le circuit correspondant à la table de vérité de F