Introduction à l'informatique

Benjamin Drieu <bdrieu@april.org>

14 avril 2001

Avant propos

Ce document contient l'ensemble des notes du cours d'II. Ce document n'est pas destiné à une réutilisation telle quelle mais plutôt comme support pour l'enseignant. Ce document me sert principalement à structurer mon cours, et j'essaye autant que possible de le suivre en classe. Il ne doit pas être considéré comme une image exacte du cours d'II.

Chaque chapitre correspond *grosso-modo* à une séance de cours, mais c'est très variable.

Conventions utilisées dans ce document

Il peut arriver que j'insère des notes un peu partout dans le cours. Ce sont des propos destinés soit à moi même, relatifs au cours mais pas à son contenu (ne pas oublier de parler du partiel, par exemple), soit au lecteur. Dans le cas de notes personnelles, supprimez-les en remplaçant la ligne

`\excludeversion{perso}`

par

`\includeversion{perso}`

dans le fichier `index.tex`.

> *Voici un exemple de note*

Le termes introduits et qui méritent une explication sont notés *comme ceci*, les variables *comme ceci*, les types `comme ceci`.

Licence

Table des matières

Chapitre 1

Historique de l'ordinateur et de l'informatique

L'Académie française définit l'informatique comme la science *du traitement rationnel, notamment par machines automatiques, de l'information considérée comme le support des connaissances et des communications, dans les domaines technique, économique et social.*

L'informatique est donc faite pour traiter de l'information, par le biais de calculs formels. Pour effectuer des calculs, on utilise des outils divers, donc le plus évolué est l'ordinateur. Mais on utilise depuis la nuit des temps des outils moins évolués, à commencer par les bouliers, les abaques, ... L'ordinateur hérite aujourd'hui des caractéristiques de tous ces outils.

Il faut savoir qu'il y a plusieurs manières de calculer. Nous connaissons aujourd'hui principalement le calcul décimal, mais ce n'est pas la seule *base* de calcul. Les ordinateurs comptent par exemple en base 2 (binaire). Certaines civilisations utilisaient couramment d'autres bases telles que la base 12 jusqu'à ce que la civilisation occidentale européenne n'impose ses conventions.

Pour comprendre l'architecture d'un ordinateur d'aujourd'hui, il faut comprendre comment ont fonctionné ses ancêtres, et par quels évolutions on est parvenu à l'architecture moderne des ordinateurs.

1.1 Jusqu'à Babbage

Les premiers instruments de calcul utilisent des techniques naturelles. Par exemple l'utilisation de petits cailloux où chaque caillou représente un objet ou un ensemble d'objets. Le mot calcul dérive d'ailleurs de *calculus*, mot latin pour caillou. Ou alors l'utilisation des doigts de la main pour calculer en base 5.

Plus formelles, les tables de calcul. Le plus ancien appareil à calculer est une table de marbre trouvée sur dans l'île de Saramine. La table est striée de manière verticale et horizontale. On place des cailloux ou des pièces et de leur position dépend leur valeur.

C'est un des ancêtres des bouliers, qui ont survécu jusqu'à maintenant (ou presque !). Expliquer le principe des bouliers. Le boulier est toujours très utilisé en Asie et au Japon. En novembre 1946, une compétition a eu lieu entre un employé du service des communications du Japon muni d'un boulier et un G.I. muni d'une calculatrice moderne de l'époque (avec les 4 signes mathématiques). Dans un concours de vitesse, c'est le boulier qui a gagné.

Jusqu'au XVIIe siècle, rien d'intéressant dans les machines à calculer. Neper (mathématicien Écossais) invente un jeu de réglettes (les « bâtons de Neper ») qui permet de faire des multiplications.

En 1642, la Pascaline. Blaise Pascal (19 ans) réalise la première machine à calculer qui permet de faire des additions et des soustractions (avec un engrenage on peut faire une retenue). Mais aucun succès, même améliorée (ajout des multiplications) par Leibniz 50 ans plus tard.

Jusqu'au XIXe, des machines à calculer sont inventées par plusieurs scientifiques (De Poleni, Caze, Lépine, Hileron De Boistissandeau), mais restent sur les principes des machines du XVIIe siècle.

Vaucanson (1750) réalise des automates à base de bandes perforées, de cames et de mécanismes d'engrenage. Invention fondamentale : description de manière formelle des étapes nécessaires à l'accomplissement d'un but. Notion de règle de fonctionnement

Falcon (mécanicien tisserand) utilise vers 1750 les perforations dans le métier à tisser. Mais Jacquard industrialise l'invention. Par un mouvement sans fin, des cartons perforés passent devant des aiguilles, et qui peuvent ainsi effectuer un tissage précis.

En 1812, grosse révolution : la machine de Babbage. Charles Babbage, mathématicien à l'université de Cambridge définit une nouvelle architecture de machine, très proche des machines électroniques actuelles

– des périphériques d'entrée ;
– une unité centrale ;
– des périphériques de sortie.

Le baron de Prony, ingénieur des Ponts et Chaussées, est responsable du cadastre (1791) qui nécessite des calculs très importants. Il a l'idée de séparer le travail en tâches élémentaires (organisation du travail moderne), où il embauche 80 calculateurs séparés en deux bâtiments (problème des erreurs). Le but était d'écrire des tables trigonométriques et de logarithmes à 14 décimales.

En 1760, on avait des tables portatives à 6 décimales imprimées et reliées (Jérôme de la Lande). Problème des fautes et des réimpressions (nouvelles fautes à chaque fois). On demandait à l'utilisateur de signaler les fautes à l'imprimeur. En 1805, le procédé du *stéréotype* (Firmin Didot) permet de conserver les pages d'une édition à l'autre. Du coup on n'introduit plus de fautes mais on les corrige d'une édition à l'autre.

Babbage rend visite au baron en 1819, et est impressionné par l'ampleur du travail et les tâches répétitives. Il décide d'automatiser tout ça, et de concevoir un calculateur entièrement dédié à ce travail (mais était intéressé par cette machine dès 1813). Il conçoit sa « *Difference Engine* » en 1820. Le but du jeu est de se servir des l'invention de Didot pour imprimer les tables de calcul (estampage des tables de plomb, pas d'impression directe).

Il commence à réaliser sa machine en 1823. Le but est de lire des cartes perforées, et en sortie de poinçonner des cartes, estamper des plaques de plomb, ou afficher des nombres sur des molettes. Mais la technologie de l'époque est insuffisante pour la réaliser. En plus, problèmes d'organisation : arnaques, contrats mal rédigés, ... Babbage meurt en 1838, au moment d'assembler les pièces. La machine est inachevée, et cédée à la Couronne britannique en 1843 (exposée aujourd'hui dans un musée).

La machine n'a jamais été finie, mais elle pose les bases de l'architecture moderne des calculateurs. Plus important, Babbage a prévu les utilisations modernes de l'automatisation des tâches administratives.

1.2 Jusqu'à l'ENIAC

En 1882, Tchebichef construit sa machine à calculer qui effectue une multiplication par additions successives. Mais c'est Léon Bollée qui réalise la première machine à multiplier directement deux opérandes

En 1885, Herman Hollerith invente la première machine à calculer à imprimer ses résultats (il était déjà l'inventeur du premier frein électromagnétique, mais le frein à vapeur lui fut préféré). Particularité, sa machine est électrique et non mécanique. Il s'en sert pour le recensement de la population des États-Unis (ce qui prit 6 mois et économisa 5 millions de dollars). Hollerith fonda la Tabulating Machine Corporation en 1885, qui devient en 1924 l'International Machine Corporation (IBM).

En 1924, Bull (ingénieur norvégien) dépose un brevet sur un système analogue à celui d'Hollerith. Il meurt d'un cancer en 1926 et lègue ses brevets à l'Institut du cancer d'Oslo. Ils sont rachetés et revendus à une société

Française, la Compagnie des Machines Bull et passeront en 1964 sous la tutelle de la General Electric.

En 1938, le Howard Aiken et la Compagnie Bell Telephone commencent à construire une machine mathématique. En 1943, le Mark I à l'université de Harvard, et en 1946 le Mark II. Machines électroniques, addition en 1/3 de seconde, multiplication en 4 secondes, division en 11. Mark II comporte 10 000 tubes électroniques. Les données en entrée et en sortie sont stockées sur des bandes perforées. Prix de revient 400.000$, soit moins que le Z3, mais les performances étaient 3 à 4 fois moins grandes. 51 pieds de long, 5 tonnes, 750.000 pièces.

En 1942, le Z3, que certains considèrent comme le premier ordinateur. Il n'avait que 1.500 lampes. En 1943, le Z3, utilisé pour calculer des plans d'avion. C'est la première machine à calculer électronique. Malheureusement pour Zuse et Schreyer, les concepteurs, le ministère de la guerre refusa d'accorder des crédits à l'invention, et on les envoya au front par deux fois, en 1939 et en 1942, d'autant plus que le Z3 fut détruit dans un bombardement. Pour la petite histoire, Zuse était un ami de Von Braun, et avait appelé ses machines V1, V2, ... puis les a renommées Zx après la guerre.

En 1944, le premier prototype des calculateurs électroniques, l'ENIAC (Electronic Numerical Integrator and Computer). 5.000 addition par seconde (calcule la trajectoire d'un obus avant qu'il n'arrive à destination). 1.000 fois plus rapide que les autres machines de son époque. Mais utilise 18.000 tubes électronique. L'aération nécessite des ventilateurs de 24 CV. La consommation électrique est de 150 kW (plusieurs rames de métro). 30 tonnes sur 1.000 mètres carrés. Multiplication en 3 millisecondes, fréquence d'horloge : 100kHz. Construit par Eckert et Mauchly, pour le compte de l'US army (calculateur balistique). Divulguée deux ans plus tard, avec un relookage de la machine pour le marketing (panneau lumineux).

En 1947, un papillon (bug) commet l'imprudence d'aller voler dans les circuits du Mark II et crame une lampe. Un technicien note « le premier exemple connu de bug réel » sur le cahier des bugs en y épinglant le pauvre insecte.

1.3 Les calculateurs électroniques

Puis en 1947, on décide de construire l'EDVAC (Electronic Discute Variable Automatic Computer), amélioration de l'ENIAC, par John Von Neumann. Utilisation de la bande magnétique (comment ça marche ?). Et surtout intégration du programme dans la mémoire de la calculatrice. C'est grâce cette amélioration qu'on impute souvent à Von Neumann l'invention de l'ordinateur.

Newman, Freddie C. Williams achèvent le premier prototype de ce qu'ils appellent le « Mark I » (ou Manchester Mark I) est le premier ordinateur à être reconnu comme tel quel par tout le monde, car c'est le premier à avoir vraiment la possibilité de contenir un programme entier en mémoire en lecture/écriture. Le programme est écrit en binaire sur un clavier. Plus tard, Türing joindra le développement de la machine et écrira un des premiers assembleurs primitifs au monde.

En 1950, le Z4 est terminé et peut effectuer des branchements conditionnels. Le Z4 servira encore 10 ans (en Suisse puis en France). Zuse décidera plus tarde de vendre des machines de manière commerciale et en vendra 300 avant d'être racheté par Siemens

En 1950, Douglas Hartree, un expert en nouvelles technologies estime que les trois calculateurs existants en Angleterre suffiront à couvrir tous les besoins présents et futurs du pays, et conseille à Ferranti Ltd. de Manchester de ne pas vendre d'ordinateurs. Mais en 1951 Ferranti construit le premier ordinateur commercial au monde, le Manchester Mark II.

En 1951, le premier calculateur commercial US est réalisé par Eckert et Mauchly (Univac I). Le premier client est le service du recensement Américain.

En 1952, l'EDVAC est enfin terminé. Fréquence d'horloge 1MHz

En 1958, le premier ordinateur à transistors (inventé en 1948), le TX-0 (Transistorized Experimental Computer) au MIT.

En 1958, chez Texas Instruments, Jack St. Clair Kilby développe le premier circuit intégré (pièce électronique sur une seule puce de silicium).

En 1963, Douglas Engelbart invente la souris.

En 1965, Gordon Moore, PDG d'Intel suggère que les CPU doubleront en complexité tous les 18 mois. C'est la loi de Moore, qu'on applique à la vitesse des CPU.

En 1970, Intel crée le premier micro-processeur 4004. 60.000 opérations pas seconde, 2.300 transistor

Par la suite, l'histoire de l'informatique est très foisonnante, et l'architecture des ordinateurs évolue très peu par rapport à l'architecture actuelle. Quelques faits cependant :

En 1972, IBM invente la disquette, une disquette de 8 pouces en plastique recouvert d'oxide de fer.

En 1973, le premier ordinateur en kit vendu pas Scelbi Computer Consulting Company, basé sur l'Intel 8008. Motorola construit le 6800.

En 1976, Steve Wozniak et Steve Jobs créent l'Apple I. Apple est fondé le premier avril. Intel crée le 8085 (5 MHz).

En 1977, l'Apple II, avec 4K de RAM, vendu pour 1300$. C'est le premier ordinateur personnel en couleur.

En 1978, le processeur 6068 d'Intel.

> *Je viens de m'appercevoir que j'ai oublié de parler des générations d'ordinateurs*

1.4 L'informatique, à quoi ça sert ?

On a vu que l'informatique a subit une évolution exponentielle. Si les transports avaient fait pareil, on pourrait dès la fin des années 40 traverser les États-Unis en 30 secondes pour 50 cents. Ordinateur machines à calculer, mais pourquoi ? Pour traiter de l'information (donc besoin de la coder sous forme de nombres). Que peut-on en faire de cette information ?

- bureautique (traitement de texte, tableur) ? C'est une utilisation très simple de l'informatique (l'ordinateur devient une machine à écrire perfectionnée) ;
- mathématiques : balistique, prévisions d'élections (jfd), météo, fractales (Pierre Audibert) ;
- jeux : IA (Feat, Bernard, Cazenave, Abchiche, Dumeur, ...), graphisme, son (Goossens), fiction interactive et génération de textes (Clément), ...
- communication : Internet (cours de réseau), Intranet, télétravail
- graphisme : CAO/PAO, reconnaissance de forme (Jaime Lopez-Krahe), météo (Pacale Pousset), reconnaissance de code postal, génération d'images (JJ Bourdin, Vincent Boyer) ;
- cinéma : image numérique, montage, dessin animé ;
- physique : simulation (centrale nucléaire, bombe atomique) (jfd), mécanique des fluides (cours d'eau, profil d'aile d'avion) (whygee), résistance des matériaux, train, voiture (déformations programmées) ;
- médecine : génome humain, imagerie médicale, opérations guidées par ordinateur, apprentissage ;
- gestion des flux : arrivées départs dans les aéroports (Renaud Dumeur), les gares, périphérique (jfd) ;
- robotique : robot autonome (robot sur Mars, footballeurs) (Ali Chérif), semi-autonome, télécommandé ;
- calculs massifs : clusters, machines parallèles (silicons de la NASA), ...
- ...

Ce ne sont que des exemples ont peut faire énormément de choses avec un ordinateur, à partir du moment où on arrive à formaliser une information sous forme de nombre. Là où on a encore des problèmes c'est pour la reconnaissance de parole et de contenu (car c'est difficile de comprendre et de formaliser la grammaire, le sens plaqué aux mots, ...).

Chapitre 2

Systèmes de numération

Puisqu'un ordinateur est une machine à calculer, pour comprendre comment il fonctionne on a besoin de savoir comment fonctionne la science du calcul.

2.1 Système romain

> *Je précise que c'est une digression tout à fait personnelle, et que cette section peut très facilement être supprimée du cours.*

Les chiffres romains ne sont pas des symboles vraiment appropriés pour effectuer des opérations arithmétiques. Ce sont plutôt des abréviations destinées à noter et à retenir les nombres. Comment ça marche ?

On a une suite de symboles disposés de droite à gauche, et chaque symbole a une valeur déterminée :

TAB. 2.1 – Correspondance système romain et décimal

Symbole romain	I	V	X	L	C	D	M
Valeur décimale	1	5	10	50	100	500	1000

Lorsqu'un signe est placé à gauche d'un signe plus fort que lui, on retranche sa valeur. Ainsi : IV vaut V - I, soit 5 - 1 = 4.

Un signe posé à droite d'un signe plus fort s'additionne à celui-ci.

On n'emploie jamais 4 lettres semblables de suite. Ainsi, 9 s'écrit IX et non $VIIII$.

Exercice : Y-a t'il un plafond qu'on ne peut pas dépasser avec ce système de numérotation ?

Oui, ce plafond est de 3999, soit $MMMCMXCIX$ (et pourquoi pas $MMMIM$?). En théorie, il y a un « hack » pour contourner ce problème, c'est de rajouter un trait au dessus du chiffre, ce qui multiplie sa valeur par mille. Ainsi, 7000 s'écrit VII avec trois traits (un au dessus de chaque lettre).

Essayez d'additionner $CXLVII$ (167) à $CDLVI$ (456) sans passer par le décimal histoire de rire. Ça donne $DCXXIII$ (623). C'est assez pénible à faire. Donc, pour les calculs, on utilise principalement les chiffres arabes (qui viennent en fait de l'Inde, et ont étés inventés entre le IVe et le VIIIe siècles). C'est lors des croisades que les Arabes ont transmis les chiffres indiens aux occidentaux, non sans mal, car en dignes héritiers de César, les occidentaux utilisaient toujours les chiffres romains et ne voyaient pas d'intérêt à utiliser des signes païens (on utilise toujours les chiffres romains dans certains domaines considérés comme « nobles », comme par exemple pour les versets de la Bible, les actes au théatre, les volumes de livres, les siècles, ...).

2.2 Comment représenter un nombre ?

Le système romain est un peu particulier. Le système Arabe est basé sur une chaîne d'éléments (symboles, alphabet) d'un ensemble V non vide, ordonnés de droite à gauche. Les éléments placés plus loin dans la chaîne ont une valeur plus importante. Ainsi :

- V_{10} = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} ;
- V_2 = {0, 1} ;
- V_{16} = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E}.

Chacun de ces ensembles est appelé une *base*. Le nom est base$_{nombre}$.

2.3 Le système binaire

C'est celui qui est basé sur deux états : vrai ou faux, allumé ou éteint, plein ou vide, 1 ou 0. C'est basé sur la perception « manichéenne » de l'homme (vrai/faux), comparaison avec les visions de certains peuples : « 1, 2, beaucoup ». L'alphabet de l'ordinateur c'est deux lettres (signaux électriques). Données + instructions sont en binaire (même raison).

Langages bas niveau / haut niveau (langage naturel, notation algébrique). Les langages de haut niveau impliquent l'utilisation d'un compilateur pour construire du langage assembleur puis binaire (langage évolué → compilation → ASM → binaire). Indépendance du code et de l'architecture de l'ordinateur (UNIX).

Nombre généralement stockés sous la forme d'un nombre de 32 bits numérotés de 0 à 31, de droite à gauche (big endian, « grand boutien ») ou de gauche à droite (little endian, « petit boutien »). Bits de poids fort et faible. Nombre maximum 2^{32} -1 soit 4.294.967.295 (et en 16 bits 32767). Principe de l'octet (digit).

C'est le système le plus simple avec lequel on puisse faire plein de choses : le système unaire est trop simple pour représenter de grands nombres, et le système décimal demande de représenter trop d'états (ex pour 1024, décimal demande 40 états, binaire 20). Compter jusqu'à 1024 avec les doigts.

On a pas trouvé mieux. Une thèse assez célèbre a porté sur un remplacement possible de la base 2 par la base 3 (ternaire) dans les ordinateurs, mais à débouché sur la conclusion : le binaire, c'est mieux.

2.4 Calculs sur les systèmes de numération

Les êtres humains comptent en décimal et ont du mal à compter en binaire. C'est pour cela qu'on a besoin de convertir des nombres d'une base à l'autre.

2.4.1 Calcul de la valeur d'un nombre

Le but du jeu est de convertir un nombre en base 10 (c'est celle à laquelle on est habitués depuis l'enfance). On prend chacun des symboles, et sa valeur est la valeur du symbole multiplié par le nombre d'éléments dans la base puissance sa position dans la chaîne (de droite à gauche en commençant à zéro).

TAB. 2.2 – Répartition des symboles

N =	2	5	7	1	$_{10}$
N =	s_3	s_2	s_1	s_0	

Donc, de manière générale : $S_n = s_n \times b^n$ (ou $^{n-1}$ si on commence à 1 comme le font certains). Donner formule développée (avec toutes les multiplications).

Même chose avec le binaire (exemple 1011_2 qui donne 11_{10}). Ou l'hexadécimal (exemple $2B_{16}$ qui donne 43_{10}).

2.4.2 Le schéma de Hörner

C'est plus rapide (trouver de la documentation sur le schéma de Hörner). Complexité linéaire (la complexité de la méthode précédente est exponentielle ; introduire la notion de complexité). Pour N = 10101_2, on a

$(((((1 \times 2) + 0) \times 2 + 1) \times 2 + 0) \times 2 + 1 = 21_{10}$.

Pour $N = 35A1_{16}$:

$(((3 \times 16) + 5) \times 16 + 10) \times 16 + 1 = 13729_{10}$

2.4.3 Recherche de synonymes

- b : base de départ
- b' : base d'arrivée
- N_b : nombre de départ
- $N_{b'}$: nombre recherché

Si $b < b'$: aucun problème

Si $N < b'$ (ex : 41_{10} = x_2)

```
41 | 2
   +---
01 | 20 | 2
        +---
 1   00 | 10 | 2
             +---
      0    0 |  5 | 2
                  +---
           0    1 |  4 | 2
                       +---
                1    0 |  2 | 2
                            +---
                     0    1 |  1 | 2
                                 +---
                          1      |  0

                                 STOP
```

Autre exemple : passer de 54_{10} à 36_{16}.

Si $b' = b^n$ (ex : x_{16} = y_2), on peut effectuer des manipulations de remplacement chiffre à chiffre (un nombre de b' = n chiffres de b).

TAB. 2.3 – Calcul par blocs

4				A			
0	1	0	0	1	0	1	0

4A = 01001010

2.4.4 Opérations sur des nombres binaires

Addition de deux nombres binaires (avec retenue). Multiplication de deux nombres binaires. Multiplication par deux d'un nombre binaire : décalage des tous les bits d'un cran.

Soustraction de deux nombres binaires (avec retenue soustraite sur le nombre de gauche). Pénible car nécessite des circuits de traitement différents.

2.4.5 Les nombres négatifs

Comment faire ? Il faut une répartition égale des nombres (stupide d'avoir une limitation). Répartir également les paires de nombres ? Non à cause du zéro (deux représentations).

Plusieurs méthodes :

– *valeur signée* : un bit de signe, reste représente la valeur absolue ;
– *le complément à 1* : un bit de signe, tous les nombres inversés. Deux représentations de 0 ;
– *le complément à 2* : pareil mais on additionne 1. -2^{31} n'a pas d'opposé, 0 est positif ;
– *le codage par excédent* : stocker somme du nombre (sur n bits) + $2^n - 1$. Impossible de manipuler deux opérandes de signe différents.

Donc pour les soustractions, on fait en fait une addition avec un nombre négatif. Subtil, non ?

Chapitre 3

Algèbre de Boole

3.1 Principe

Georges Boole (1815-1864), physicien Anglais définit en 1847 un algèbre qui porte son nom. Son algèbre est applicable au raisonnement logique, qui traite des fonctions à variables binaires (deux valeurs). Mais il ne s'applique pas aux systèmes à plus de deux états d'équilibre. Il permet d'étudier les circuits logiques (un système logique sert à modifier des signaux).

Une variable booléenne, ou logique, ou binaire ne prend que deux valeurs (elle est généralement stockée sous la forme d'un bit, ou alors on gâche de la place dans un octet car on verra que les ordinateurs actuels sont incapables de stocker moins d'un octet en mémoire).

Une fonction logique est associée à une ou plusieurs variables binaires, et fournit un résultat qui dépend uniquement de ces valeurs. La plupart sont des fonctions binaires (attention, ça veut ici dire avec deux opérandes). Comparaison avec les opérateurs binaires « analogiques », comme $+$, $\times$, $\div$ ou $-$.

Table de vérité : table de correspondance entre les variables binaires traitées par une fonction logique et le résultat de la fonction logique.

Exemple de fonction logique : la fonction interrupteur. I est la valeur l'interrupteur, 1 pour ouvert, 0 pour fermé. L est l'état de la lampe située après l'interrupteur. $f(L) = I$, L est fonction de I.

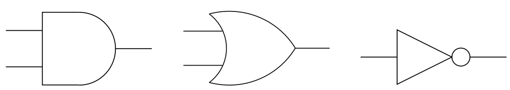
TAB. 3.1 – Exemple de table de vérité

I	L
0	0
1	1

Deuxième exemple : eclairage d'une salle. La salle a deux fenêtres, protégés par des volets. Elle n'est éclairée que lorsqu'au moins une fenêtre est ouverte. A représente l'ouverture de la première fenêtre (0 pour fermée, 1 pour éclairée). B représente l'ouverture de la deuxième fenêtre (0 pour fermée, 1 pour éclairée). S représente l'éclairage de la salle (0 pour non éclairée, 1 pour éclairée).

TAB. 3.2 – Table de vérité de l'éclairage de la salle

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



(a) AND (b) OR (c) NOT

FIG. 3.1 – Les trois portes logiques fondamentales

S dépend de la valeur des variables binaires A et B.

> *À ce stade là, il y a toujours des étudiants qui se demandent pourquoi on parle de tout ces trucs compliqués. Il faut savoir que les opérations précédentes sont reproductibles en utilisant des fonctions booléennes.*

3.2 Les fonctions logiques fondamentales

Il y en a trois, voir figure 3.1 de la présente page.

- la porte ET (ou produit, $\times$, $\wedge$)
- la porte OU (ou somme, $+$, $\vee$)
- la porte INVERSE (ou négation, $-$, $\neg$)

Deux représentations, américaine ou européenne. La représentation américaine est creusée à gauche, et l'européenne est droite. On utilisera alternativement soit l'une, soit l'autre.

3.2.1 La porte ET

La porte ET est une opération logique effectuée sur deux variablesx binaires de telle sorte que si et seulement si deux variables sont égales à 1, alors le résultat est égal à un. Voir la figure 3.3 page suivante pour un exemple réel d'utilisation de la porte ET.

TAB. 3.3 – Table de vérité de la porte ET

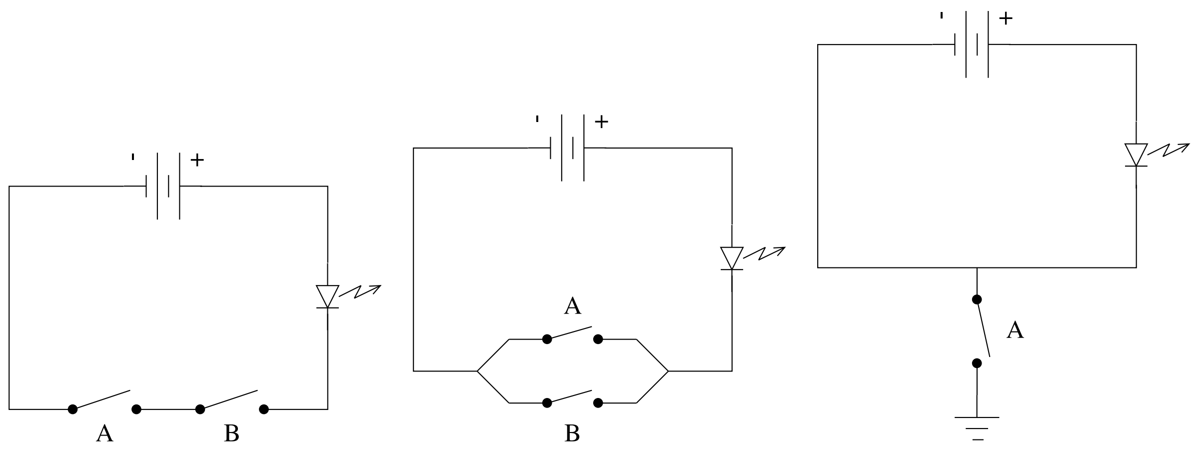
A	B	$A \wedge B$
1	1	1
1	0	0
0	1	0
0	0	0

La propriétés de la porte ET :

- commutativité :
- associativité :
- distributivité.

3.2.2 La porte OU

La porte OU est une opération logique effectuée sur deux variablesx binaires de telle sorte que si et seulement si une des deux variables est égale à 1, alors le résultat est égal à un. Voir la figure 3.3 page suivante pour un exemple réel d'utilisation de la porte OU.



(a) Exemple pratique de porte ET (b) Exemple pratique de porte OU (c) Exemple pratique de porte INVERSE

FIG. 3.2 – Exemples pratiques des portes ET, OU, INVERSE

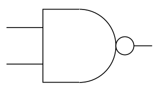


FIG. 3.3 – Porte NON ET

TAB. 3.4 – Table de vérité de la porte OU

A	B	$A \wedge B$
1	1	1
1	0	1
0	1	1
0	0	0

– commutativité :
– associativité :
– distributivité.

Exercice : faire (ensemble) la table de vérité du circuit $S = A \vee (B \wedge C)$ (faire seulement le circuit physique au tableau). $S = 1$ si et seulement si $A = 1$ ou $B \wedge C = 1$, ou les deux.

3.2.3 La porte NON

La porte NON (ou inverse) est une opération logique sur une variable binaire (opérateur unaire) de telle sorte que le résultat est un si et seulement si la variable est 0. On peut représenter l'opérateur NON par une barre au dessus de ce qu'on inverse (par exemple : $\overline{A}$).

TAB. 3.5 – Table de vérité de l'opérateur NOT

A	S
0	1
1	0

Il est possible de modifier le comportement des opérateurs logiques fondamentaux en leur mettant en sortie une porte NOT. Un exemple de porte NON-ET est reproduit dans la figure 3.3 de la présente page (il suffit de rajouter une boule à la fin).

3.2.4 Exercices

Exercice : construire circuit $S = \overline{(\overline{A} \vee \overline{B})} = (A \wedge B) = A \wedge B$ plus table de vérité.

Autre exercice : $S = (\overline{A} \vee \overline{B}) \wedge (\overline{A} \vee B) \wedge (A \vee \overline{B})$

Encore un : faire l'inverse et passer à l'équation de la fonction logique majoritaire à partir de la table de vérité ($M = (A \wedge B) \vee (B \wedge C) \vee (A \wedge C)$). Construire le circuit :

- rajouter une inversion de toutes les entrées ;
- construire une porte ET pour chacune des entrées égales à 1 dans la colone résultat ($\overline{A} \wedge B \wedge C$ par exemple) :
- construire un porte ET pour chacun des termes égaux à 1 dans la colonne résultat ;
- faire un truc plus propre en utilisant des câbles verticaux pour $A, B, C, \overline{A}, \overline{B}, \overline{C}$, et en plugant les portes horizontalement ;
- réunir l'ensemble des portes ET vers une porte OU (opérande non forcément binaire, donc j'ai menti :) :
- simplifier le circuit : $A \wedge B \wedge C \vee \overline{A} \wedge B \wedge C = (A \vee \overline{A}) \wedge (B \wedge C)$ (obtenu en passant par la loi d'indempotence, soit $ABC = ABC + ABC$).

TAB. 3.6 – Table de vérité de la fonction majoritaire

A	B	C	S
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

3.3 Lois de l'algèbre de Boole

Parler de la priorité des opérateurs (parenthésage).

Pour simplifier des circuits logiques, on a besoin de connaître les lois de Boole (on verra la loi de Morgan plus tard dans le cours).

Faire trouver ces lois à partir des tables de vérité des opérateurs ET, OU, NON (certains sont proches de l'algèbre traditionnel : $A + 0 = A$ $A + 0 = 0$, $A(B + C) = AB + AC$).

TAB. 3.7 – Lois de l'algèbre de Boole

Loi	Forme avec ET	Forme avec OU
Loi d'identité	1A = A	0+A=A
Loi de nullité	0A = 0	1+A = 1
Loi d'idempotence	AA = A	A+A = A
Loi d'inversion	$A\overline{A}$=0	$A + \overline{A}$=1
Loi commutative	AB=BA	A+B = B+A
Loi associative	(AB)C=A(BC)	(A+B)+C=A+(B+C)
Loi distributive	A+BC=(A+B)(A+C)	(A(B+C)=AB+AC
Loi d'absorption	A(A+B) = A	A+AB=A
loi de Morgan	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\,\overline{B}$

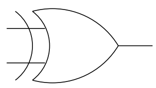


FIG. 3.4 – Porte XOR

3.4 Autres portes

NON-OU

NON-OU, (Not Or, NI).

$f(x, y) = \overline{x \vee y}$ (prononcer x et y barre). Faire la table de vérité. On se rend compte que c'est la même que celle de $\overline{x} \wedge \overline{y}$.

Reproduire la porte.

NON-ET

NON-ET (NAND, Not And).

$f(x, y) = \overline{x \wedge y}$ (prononcer x ou y barre). Faire la table de vérité. On se rend compte que c'est la même que celle de $\overline{x} \vee \overline{y}$.

Reproduire la porte.

OU exclusif

XOR (voir figure 3.4 de la présente page).

Différence avec OU inclusif. $f(x, y) = x \oplus y$ (prononcer x ou exclusif y). Faire la table de vérité.

Extension à plus de deux variables

Les portes ET, OU, et autres peuvent êtres étendues à n variables binaires en entrée.

3.4.1 Identité

$f(x, y) = x \equiv y$ ou $f(x, y) = x \odot y$ (prononcer x identique y). Faire la table de vérité, c'est la même que $\overline{x \otimes y}$.

3.5 Théorème de Morgan

Augustus De Morgan (1806-1871) a continué le travail de Boole.

Le théorème De Morgan dit :

$\overline{x \vee y \vee z \vee \ldots} = \overline{x} \wedge \overline{y} \wedge \overline{z} \wedge \ldots$

$\overline{x \wedge y \wedge z \wedge \ldots} = \overline{x} \vee \overline{y} \vee \overline{z} \vee \ldots$

Faire vérifier.

Attention à la priorité des opérateurs : $\overline{A \vee B \wedge C}$ n'est **pas** égal à $\overline{A} \wedge \overline{B} \vee \overline{C}$ mais à $\overline{A} \wedge (\overline{B} \vee \overline{C})$.

Une des applications immédiates, c'est quand on n'a pas de porte ET, on peut la faire avec une porte NON-OU et deux entrées inversées.

3.6 Les tables de Karnaugh

3.6.1 Principe

Méthode de représentation graphique basée sur l'identité $AX + A\overline{X} = A$ (le prouver). On se sert de ça pour supprimer des termes dont on n'a pas besoin.

Représentation graphique : les termes adjacents sont *adjacents graphiquement*. Termes adjacents : $x\overline{y}zt$ et $x\overline{y}z\overline{t}$ sont adjacents car l'état d'une seule variable binaire est différent (ici c'est t). Notons que $x\overline{y}z\overline{t} + x\overline{y}zt = x\overline{y}z$ car $x\overline{y}z\overline{t} + x\overline{y}zt = x\overline{y}z(t + \overline{t})$. C'est ce qu'on appelle le *gray order*. Principe du tore et de la 4 connexité (exemple avec deux variables).

Faire un exemple avec un ordre *non gris* pour montrer le principe.

3.6.2 Représentation

TAB. 3.8 – Exemple de table de Karnaugh pour une fonction logique à deux entrées (première représentation)

ab	00	01	11	10
	...	...	...	...

TAB. 3.9 – Exemple de table de Karnaugh pour une fonction logique à deux entrées (deuxième représentation)

y \ x	0	1
0	$\overline{x}\,\overline{y}$	$x\,\overline{y}$
1	$\overline{x}\,y$	xy

Tableau à 3 variables (deux représentations), faire semblant de se planter avec deux colonnes qui ne sont pas séparées d'une seule variable binaire.

TAB. 3.10 – Exemple de table de Karnaugh pour une fonction logique à trois entrées (première représentation)

z \ x y	00	01	11	10
0	$\overline{x}\overline{y}\overline{z}$	$\overline{x}y\overline{z}$	$xy\overline{z}$	$x\overline{y}\overline{z}$
1	$\overline{x}\overline{y}z$	$\overline{x}yz$	xyz	$x\overline{y}z$

TAB. 3.11 – Exemple de table de Karnaugh pour une fonction logique à trois entrées (deuxième représentation)

z \ x y	$\overline{x}\overline{y}$	$\overline{x}y$	xy	$x\overline{y}$
z	$\overline{x}\overline{y}\overline{z}$	$\overline{x}y\overline{z}$	$xy\overline{z}$	$x\overline{y}\overline{z}$
$\overline{z}$	$\overline{x}\overline{y}z$	$\overline{x}yz$	xyz	$x\overline{y}z$

Tableaux à 4, 5 et 6 variables (x, y, z, t, u).

Juxtaposition de tableaux à 4 variables pour 5 et 6 variables. Parler du principe de l'hypercube. On est limités par un support à deux dimension (plus par la représentation spatiale humaine qui est fondamentalement limitée à 3 dimensions).

TAB. 3.12 – Exemple de table de Karnaugh pour une fonction logique à quatre entrées

z t \ x y	00	01	11	10
00				
01				
11				
10				

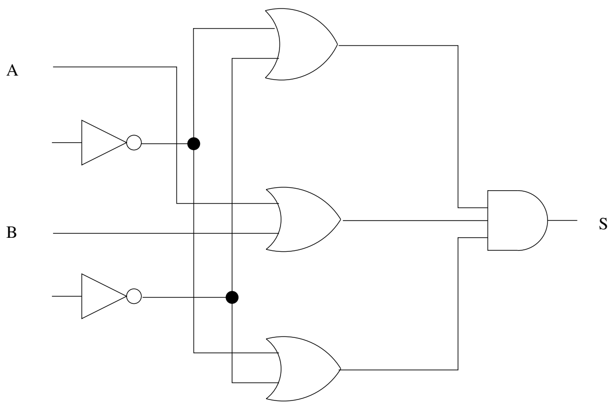


FIG. 3.5 – Exercice 2

3.6.3 Méthode de Karnaugh

- transposition du tableau de verité dans un tableau de Karnaugh ;
- réalisation des groupements de 1, 2, 4, 8 termes ;
- minimisation des groupements (maximisation des termes dans un groupement) ;
- si groupement d'un terme, alors on ne fait rien ;
- si 2 termes, on élimine la variable qui change d'étatet on conserve le produit des variables directes ou inverses qui n'ont pas changé d'état dans le groupement ($ax + a\overline{x} = a$) ;
- pour 4 termes, on élimine les 2 variables qui changent d'état ;
- pour 8 termes, on élimine les 3 variables qui changent d'état ;
- l'expression logique finale est la réunion des groupements après élimination des variables.

3.6.4 Exemples

$F = (\overline{A} + \overline{B}) + (\overline{A} + B) + (A + \overline{B})$. Voir figure 3.5 de la présente page).

$F = bc + ac + ab + b$

TAB. 3.13 – Table de Karnaugh pour $F = bc + ac + ab + b$

c \ a b	$\overline{ab}$	$\overline{a}b$	ab	$a\overline{b}$
c	0	1	1	0
$\overline{c}$	0	1	1	1

d'où $F = b + ca$.

Construire l'expression logique $f = y\overline{zt} + x\overline{y} + \overline{xyt}$ à partir de la table suivante.

TAB. 3.14 – La table de Karnaugh pour $f = y\overline{zt} + x\overline{y} + \overline{xyt}$

zt \ ab	00	01	11	10
00	1	0	0	1
01	0	1	1	1
11	0	0	0	1
10	1	0	0	1

Chapitre 4

Circuits logiques

Assez de théorie, on va voir ce qu'on peut faire avec les circuits logiques qu'on a vus jusqu'à présent.

Deux types d'implémentation des fonctions logiques :

– circuits combinatoires : le résultat ne dépend que des variables d'entrée ;
– circuits séquentiels : le résultat dépend des états antérieurs.

Pourquoi voir ces circuits logiques ? On en a besoin pour comprendre le fonctionnement du processeur.

4.1 Circuits combinatoires usuels

4.1.1 Le décodeur

Exemple du paneau de LEDs qui correspondent chacune à une petite lumière. Voir le Tanenbaum pour le circuit complet.

4.1.2 Le multiplexeur

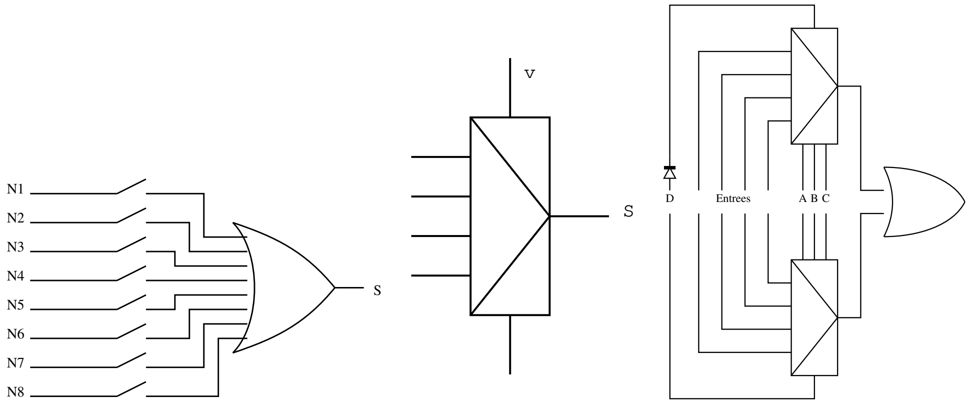
> *Rappel de ce qu'est un circuit logique combinatoire (variables d'entrée et de sortie, qui dépendent des variables d'entrée).*

Le but du multiplexeur est d'avoir la valeur d'une et d'une seule variable binaire d'entrée en sortie (et donc d'éliminer la valeur des autres). Comment sélectionner une des entrées (dont on ne connaît pas la valeur), et avoir sa valeur en sortie d'une boite (faire schéma) ? Une solution est de placer des interrupteurs après chacune des entrées (commencer par en mettre un seul puis décomposer).

Faire le schéma simplifié (mais avec 4 entrées seulement, numérotées de 0 à 3, voir figure 4.1(a) page suivante). Dans un circuit logique, les interrupteurs n'existent pas (et il faut les contrôler !). On peut faire un interrupteur avec une porte ET (en introduisant la variable binaire *interrupteur ouvert*). Faire circuit (rappeler qu'on ne connaît pas forcément les valeurs des variables d'entrée), où on a une porte ET et une autre variable d'entrée (de contrôle) par interrupteur (faire faire le circuit). Mais c'est lourd car on a autant de variables d'entrée de contrôle (imaginer une boîte avec autant de boutons poussoirs que de variables d'entrée).

Alors, solution de numéroter les entrées, et en entrée de contrôle, mettre un potentiomètre qui détermine l'entrée n dont on a besoin. n étant un nombre, on sait le représenter sous forme binaire. Donc on peut envoyer en entrée de contrôle un nombre binaire (l'idéal serait que l'interrupteur n soit ouvert lorsque l'entrée de contrôle valle n). C'est facile, on sait le faire (faire la transition avec le multiplexeur tel que le montre Tanenbaum).

Généralement, les mutltiplexeurs sont vendus avec n entrées, n étant multiple de 2. Comment faire pour multiplexer $n + 1$ entrées (ex. 9 variables quand on ne dispose que de multiplexeurs à 8 entrées) ?



(a) Mutiplexeur simplifié (b) Multiplexeur à entrée v (c) Multiplexeurs à 2^n entrées

FIG. 4.1 – Le multiplexeur

Parler du multiplexeur à entrée v pour construire un multiplexeur à 2^{n+1} voies avec deux multiplexeurs à 2^n voies.

4.2 Circuits logiques arithmétiques

Utilisés pour réaliser les opérations arithmétiques élémentaires comme l'addition, la soustraction, ... Soit les opérations de base qu'on est en mesure d'attendre de la part d'un ordinateur : après tout, un ordinateur c'est une machine à calculer. Ce sont des *circuits logiques*.

4.2.1 Additionneur 1 bit

On a vu que l'addition est l'opération la plus simple, la plus rapide et la plus fondamentale (avec des additions on peut implémenter la multiplication ainsi que la soustraction). On va voir en détail l'implémentation d'une addition.

Demi additionneur (*halfadder*)

Effectuons l'addition de deux variables binaires, on s'occupera par la suite d'additionner des nombres composés de plusieurs digits (plus complexe).

Additionner deux nombres binaires avec en résultat un nombre à deux digits. Principe de la retenue (si on veut un résultat sur un bit). La somme S est égale à 1 si et seulement si l'**une** des deux variables d'entrée est égale à 1. La retenue est égale à 1 si et seulement si les deux variables sont égales à 1.

Table de vérité avec deux variables binaires en entrée et deux variables binaires en sortie (A, B, S, R).

Ça permet donc d'implémenter une addition binaire, avec retenue (le faire faire dans une boîte). Démarche naïve, on va pouvoir construire un additionneur de nombres codés sur plusieurs bits (après tout, l'addition de deux nombres c'est une addition chiffre à chiffre). Faire schéma d'additionneurs en parallèle. Ajouter la notion de retenue : les mêmes en série (à cause des retenues précédentes, mais où les insérer ?).

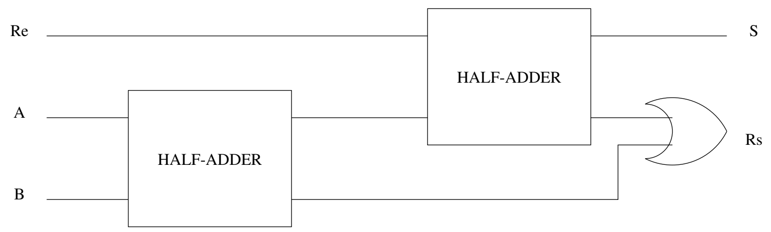


FIG. 4.2 – Full Adder

Additionneur complet

Donc, problème de la retenue. Comment faire ? On pourrait relier les additionneurs les uns avec les autres (comme on a commencé à le faire pour reporter la retenue), mais dans ce cas là, on a besoin d'additionner la retenue (pourquoi et comment). Table de vérité de l'additionneur complet (avec retenue d'entree Re, de sortie Rs et la retenue R de l'addition de A et de B). Selon la loi associative, $A + B + C = (A + B) + C$, donc la sortie S est l'addition de $A + B$ et de Re. La retenue de sortie est plus complexe (deux manières de voir ça) : les cas dans lesquels on a une retenue de sortie sont ceux où au moins 2 variables d'entrée sont égales à 1 (fonction majoritaire) (ou $A \wedge B$ ou $(A$ **plus** $B) \wedge Re$).

Faire le schéma de l'additionneur (voir figure 4.2 de la présente page), et branchement des additionneurs en série.

C'est moyennement performant au point de vue rapidité car on est obligé de calculer le résultat de chacun des additionneurs pour connaître le résultat des suivants (propagation de retenue, introduire la notion de propagation de signal, qui est pour la lumière de 20cm par nanoseconde sur un tube de cuivre, parler des premiers Crays qui étaient conçus pour optimiser le câblage). Utilisation de circuits à prévision de retenue (simple calcul du nombre de bits à un dans l'additionneur précédent).

Problème de l'*overflow* (exemple du complément à 2).

4.2.2 La soustraction

On veut soustraire B à A $(A - B)$.

Inverser B (complément à deux), puis l'additionner avec A.

Faire des exemples, avec des résultats positifs ou négatifs ($7 - 6$ et $2 - 6$ par ex). Si résultat négatif, effectuer complément à 2 pour trouver l'inverse (plus compréhensible pour les étudiants).

Chapitre 5

L'ordinateur

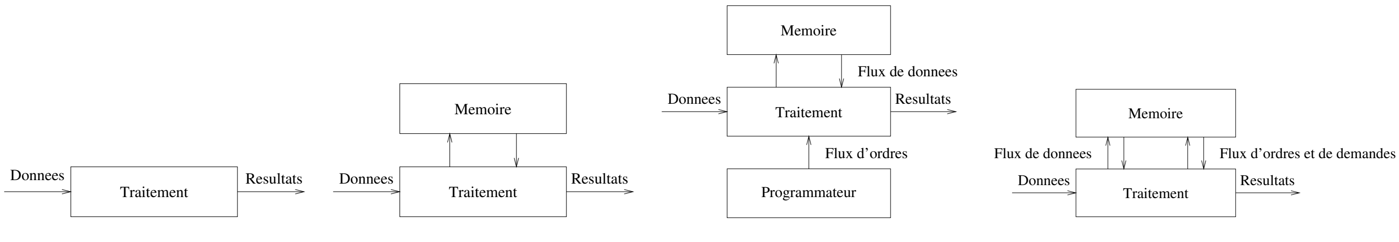
5.1 Évolution de l'architecture de l'ordinateur

Un ordinateur est une machine à calculer automatique. Le schéma général d'un ordinateur est de prendre en entrée des données, d'effectuer un traitement automatique, par exemple une addition (sans intervention de l'humain) et de fournir en sortie des résultats (voir le schéma 5.1 de la présente page). 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 $(4 + 5) + (6 + 7)$. Il faudrait mémoriser le résultat intermédiaire (9).

Rajoutons donc une mémoire qui permettrait de stocker les résultats intermédiaires (voir le schéma 5.1 de la présente page). 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 parmis plusieurs possibles (additions, multiplications, soustractions, etc.). Le traitement que la machine pourrait désormais effectuer serait beaucoup plus étendu, et serait le résultat d'une suite d'*actions élémentaires*. Cette suite d'actions est un *programme*. On construit donc un organe particulier à l'ordinateur, qu'on appelle le *programmateur* (il trouve ses ordres sur un support quelconque, carte perforée, ruban, etc.). C'est l'architecture développée par Babbage dès 1812. Le nombre de traitements possibles est dorénavant pratiquement infini. Problème : le programme est indépendant des résultats intermédiaires (pas de « si A alors B »), et dépend de la vitesse du défilement du support du programme.

Dans les années 40, l'architecture de l'ordinateur n'a toujours pas évolué et la programmation physique (par commutateurs) est devenue vraiment trop lente et trop rigide par rapport à la vitesse de traitement des instructions (exemple de l'ENIAC qui est un monstre). Von Neumann à l'idée de placer le programmme en mémoire (*proposition de Von Neumann*). La vitesse est limitée par le temps de transfert en mémoire, le programme peut réagir aux résultats trouvés (branchements conditionnels).



(a) Schéma 1 (b) Schéma 2 (c) Schéma 3 (d) Schéma 4

FIG. 5.1 – Architecture de la machine informatique

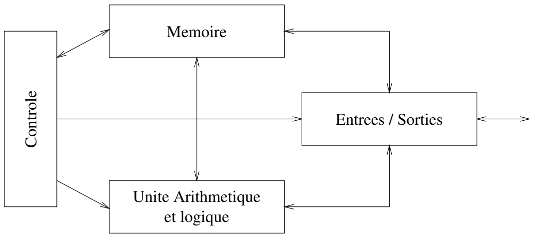


FIG. 5.2 – Structure de l'ordinateur, schéma général

5.2 L'ordinateur de Von Neumann

Détaillons l'ordinateur de Von Neumann. Les cinq composants (voir la figure 5.2 de la présente page) de l'ordinateur de Von Neumann sont :

- l'unité arithmétique et logique (UAL) ou unité de traitement ou unité de calcul ;
- l'unité de contrôle ou unité de commande ;
- la mémoire centrale ou mémoire principale ;
- les unités d'entrée ;
- les unités de sortie.

On va voir les uns après les autres tous les composants de l'ordinateur de Von Neumann et analyser les stratégies de communication entre ces composants.

5.2.1 Définitions

- *processeur* : réunion de l'UAL et de l'UC ;
- *unité centrale* : réunion du processeur et de la mémoire ;
- *instruction* : traitement effectué à un instant donné par le système (action atomique) ;
- *programme* : suite d'instructions effectuant un certain traitement (le nombre de programmes possibles est théoriquement infini).

5.2.2 L'UAL

L'UAL est un circuit logique chargé d'exécuter les opérations élémentaires permettant les traitements effectués par l'ordinateur. L'UAL reçoit des instructions de l'unité de commande, et peut lire et écrire en mémoire.

Principe de l'UAL : elle permet d'effectuer une opération binaire (deux opérandes), choisie par l'unité de contrôle. Chacune des opérations possibles est repérée par un nombre (codé en binaire). L'UAL prend donc deux opérandes en entrée, une opération, et des variables binaires en sortie (plus les retenues et autres indicateurs).

Représentation (voir figure 5.3 page suivante) et exemple tiré du schéma du Tanenbaum p. 136 (ne pas faire le schéma directement, faire des blocs ET, OU, NON et +, et ensuite faire le décodeur et les circuits logiques).

Puisqu'on a plusieurs opérations possibles, on les numérote en partant de 0. On a une boîte avec des opérations en vrac, qu'on va numéroter. On ne veut que calculer le résultat d'une opération. Deux solutions : installer un décodeur en début de l'UAL, ou un multiplexeur à la fin.

On a vu une UAL avec 2 bits d'entrée et 4 opérations différentes. Si on veut augmenter le nombre d'opérations possibles, alors il suffit d'augmenter le nombre de bits de codage de l'instruction. Si on veut effectuer des opérations sur des variables non binaires, on peut mettre des UAL en parallèle (voir Tanenbaum et figure 5.4 page suivante), ce qui est moins compliqué qu'une grosse UAL (retenues ou sorties spéciales par ex). Attention au problème de la retenue (retenue de sortie par ex). Montrer que la fonction ET peut aussi être appliquée à des nombres codés sur plusieurs bits.

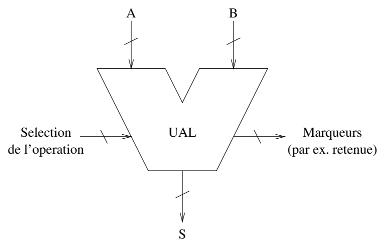


FIG. 5.3 – Unité arithmétique et logique (UAL)

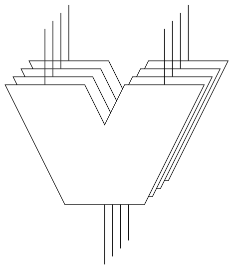


FIG. 5.4 – Unités arithmétiques et logiques en parallèle

Principe fondamental : le résultat est toujours calculé **en fonction** des variables binaires d'entrée, donc on a toujours un circuit logique.

5.2.3 Communication entre les unités de la machine de Von Neumann

> *Rappel sur le fait que l'ordinateur de Von Neumann est composé de plusieurs unités fonctionnelles qui ont chacune une mission, et qui communiquent entre elles. Une certaine interaction sociale est effectuée entre elles (avec des conflits, des marchandages, ...)*

Les unités de la machine de Von Neumann sont intimement liées et communiquent entre elles en utilisant des *bus* de données.

Tanenbaum définit un bus comme étant un « dispositif non bouclé destiné à assurer simultanément les transferts d'information entre différents sous-ensembles d'un système informatique selon des spécifications physiques et logiques communes ».

Un bus est un ensemble de fils électriques sur lesquels ont effectue la transmission de signaux en parallèle. Il y a des bus unidirectionnels, et bidirectionnels (relation maître/esclave).

Il y a plusieurs types de communications entre les unités de l'ordinateur de Von Neumann :

– communication d'adresses ;
– communication de données ;
– communication de commandes.

Il y a un type de bus pour chacune de ces données (la fonctionnalité est différente, mais pas la conception qui est identique). L'unité centrale communique aussi avec les autres parties de l'ordinateur par le biais de bus (mais dont la conception est différente).

Le principe est de transmettre des données (plusieurs bits) sur une ligne de communication. Chaque bus à ses caractéristiques propres : la taille et la fréquence.

Analogie avec le trafic automobile entre deux villes et la nécessité de disposer de feux rouges (cadençage). Cadençage effectué par des *horloges* (ou *bases de temps* ou *systèmes de cadençage*).

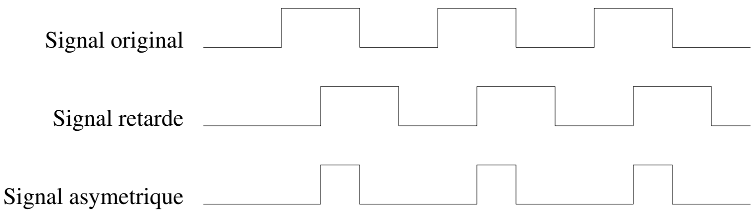


FIG. 5.5 – Signaux déphasés et signal asymétrique

Le cadençage

Revenons à nos circuits logiques. Il est nécessaire pour obtenir un résultat déterministe (propre du circuit logique) que toutes les variables d'entrée arrivent en même temps, ou alors qu'on puisse déterminer un intervalle de temps où elles sont toutes fixées en même temps.

En effet, les variables d'entrée d'un circuit logique étant des impulsions électriques, il est possible qu'un temps de propagation assez lent rende chaotique l'utilisation d'un circuit logique. On a besoin de *séquentialiser* les circuits logiques.

> *un petite digression sur les ordinateurs sans horloges ou quantiques serait la bienvenue ici.*

Définition de synchrone et d'asynchrone.

Le principe est d'utiliser une horloge qui va produire des impulsions régulières et ainsi générer des cycles. Notion de fréquence. Faire le passage de la fréquence à la période. En général, les horloges ont une fréquence de 1 à 100 MHz, ce qui correspond à une période de 10ns à 1μs. Rappel du trajet de la lumière : 20cm par ns, donc circuit très petit (rappel de la technologie Cray). Les horloges sont la plupart du temps des oscillateurs à quartz qui vibrent lorsqu'ils sont soumis à une tension.

Montrer les diagrammes temporels (chronogramme) des oscillations de l'horloge. Front montant et descendant (non immédiats). On a donc deux événements remarquables par cycle.

Signaux déphasés

Il est fréquent d'avoir plusieurs événements intéressants par unité de temps (problème d'atomicité). De même, leur ordre d'arrivée est important. Utilisation d'un signal déphasé secondaire, ce qui introduit deux événements remarquables supplémentaires par cycle (voir schéma 5.5 de la présente page).

En découpant un cycle en plusieurs sous-cycles (ou cycles mineurs), on peut garantir une séquentialité déterminée. Si on a besoin de plus de quatre actions par cycle, on peut introduire d'autres signaux secondaires pour rajouter des sous-cycles.

On a vu pour l'instant des signaux symétriques (le temps de niveau haut est le même que le temps de niveau bas), mais certains cadençages sont effectués sur des signaux asymétriques (ex du troisième diagramme de la figure 5.5 de la présente page). On reste dans l'optique d'un cadençage à 2 événements (front montant et descendant).

Autre solution, effectuer des ET entre le signal d'horloge et un des signaux déphasés (v. Tanenbaum p. 229).

Notion de performance et analogie avec les avions qui transportent des passagers : les bus ont une fréquence (rapidité de l'avion, ce qui implique plus d'aller-retours) et une taille (nombre de passagers). Dans un ordinateur, ce qui est important c'est le débit des bus (et pas la vitesse d'exécution du processeur), car c'est là qu'on trouve les goulots d'étranglement (les cartes mères de PC sont cadencées en général à 100MHz).

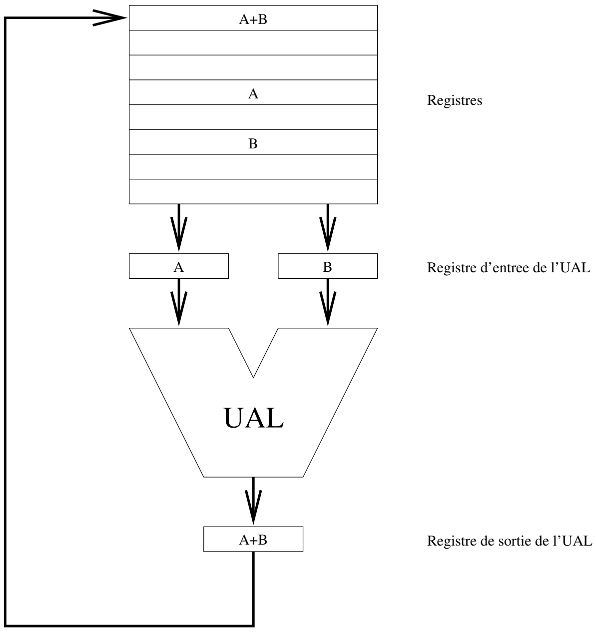


FIG. 5.6 – Chemin de données de l'ordinateur de Von Neumann

TAB. 5.1 – Performances d'avions de transport

Nom	Passagers	Rayon en km	Vitesse en km/h	Débit
737-100	101	1014	962	97 162
747	470	6677	982	461 540
Concorde	132	6426	2172	286 704
DC8-50	146	14030	875	127 750
	Nb de données	Durée de vie	Fréquence	Débit de données

5.2.4 L'unité de contrôle

Un des moyens de voir l'unité de contrôle est d'imaginer un chef d'orchestre qui donnerait des ordres aux différents musiciens de l'orchestre afin de les synchroniser et d'obtenir un tout cohérent.

Principe

Le but de l'unité de contrôle est de chercher les instructions du programme en mémoire, de les décoder et de commander leur exécution de manière séquentielle.

Dans le cas d'une machine de Babbage, les instructions sont lues sur un dispositif externe. Dans le cas d'une machine de Von Neumann, les instructions sont lues en mémoire.

Elle s'inscrit dans le chemin de données de l'ordinateur de Von Neumann (5.6 de la présente page). L'unité de contrôle sert (en partie) à conserver les résultats intermédiaires des calculs de l'UAL et à lire et à écrire des données en mémoire.

L'unité de contrôle a pour mission de placer en entrée de l'UAL les opérandes nécessaires à l'exécution de ses instructions. Elle va les lire dans une mémoire de petite taille nommée *registres*.

Les registres de données de l'unité de contrôle sont composés de cases mémoires, repérées par une adresse (les numéroter), et qui possèdent chacune une fonctionnalité bien précise. Conceptuellement, ils sont identiques aux cases mémoires, mais situés différemment dans l'architecture du processeur, et avec des fonctions différents (on appelle aussi les registres *mémoire bloc-notes*). Plus l'ordinateur est complexe, plus le nombre et la taille des registres sont importants.

L'unité de contrôle possède d'autres registres tels que le compteur ordinal et le registre d'instruction (à voir plus tard).

Selon la figure 5.2 page 23 et pour résumer grossièrement, l'unité de contrôle a les interactions suivantes avec les autres unités fonctionnelles de l'ordinateur de Von Neumann :

– elle va chercher des instructions en mémoire en utilisant les bus d'adresses et de données ;
– elle décode cette instruction et transmet une action à l'UAL si nécessaire (bus de commande), plus des données/opérandes (bus de données) ;
– elle récupère le résultat de l'exécution de l'opération et le stocke dans un registre.

Une vue rapide de la mémoire centrale

Pour résumer, la mémoire centrale est un « ruban » contenant des cases mémoire, repérées par des adresses (nombres). On peut faire une analogie avec une énorme bibliothèque dont toutes les étagères sont numérotées.

La mémoire est capable d'aller chercher le contenu d'une case mémoire. Il suffit de donner une adresse à un bibliothécaire, et en retour il vous ramène le contenu de l'étagère dont on lui a donné l'adresse. Le passage de l'adresse est donc effectué par le bus d'adresses, et le retour par le bus de données.

Les cycles de l'instruction

Un cycle d'instruction est typiquement composé d'une opération atomique effectuée par l'UAL, puis quelque temps après au rangement du résultat dans un registre et du vidage des registres A et B. Problème de séquentialité : si on vide les registres en même temps qu'on effectue l'opération, on risque d'avoir des opérandes fausses et donc un résultat faux. D'où l'intérêt de disposer d'un cadençage séquentiel. En fait, chaque cycle est découpé en 3 étapes importantes.

FETCH Le compteur ordinal a pour but de contenir l'adresse de l'instruction suivante (notion de séquentialité). En effet, dans une machine de Von Neumann l'unité de contrôle va chercher les instructions à exécuter dans la mémoire centrale comme de vulgaires données (ce qui est différent de la machine de Babbage).

Elle utilise le bus d'adresses pour placer sa requête (adresse contenue dans le compteur ordinal), le bus de commande pour demander à la mémoire une opération de lecture, la mémoire place le contenu de la case mémoire lue sur le bus de données et indique que la lecture est faite via le bus de commande. La donnée ainsi obtenue est stockée dans le registre d'instruction.

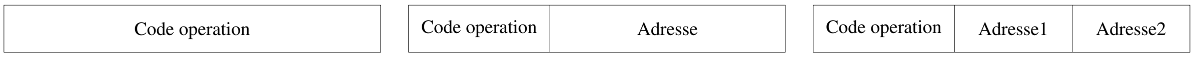
Le compteur ordinal doit ensuite être incrémenté (passage à l'instruction suivante).

DECODE On a donc chargé une *micro-instruction* à partir de la mémoire dans le registre d'instruction. Il ne nous reste plus qu'à la décoder. Elle est découpée en plusieurs champs. Pour simplifier, la micro-instruction contient une opération a effectuer (code de l'opération a transmettre à l'UAL), ainsi que des informations sur les opérandes à utiliser, comme par exemple l'adresse en mémoire où chercher une donnée, et le numéro du registre où la stocker.

> *Faire le découpage de la partie commande de la partie adresse (en précisant que je mens un peu puisqu'on détaillera plus en détail la séance prochaine.*

EXECUTE L'unité de contrôle n'a ensuite plus qu'à passer à la phase exécution. On effectue ce que demande l'instruction : lecture/écriture (requête à l'unité d'entrée/sortie), chargement d'une donnée, exécution d'une opération par l'UAL, ...

Et on recommence pour l'instruction suivante.



(a) Schéma 1 (b) Schéma 2 (c) Schéma 3

FIG. 5.7 – Trois exemples de format d'instruction

5.2.5 La mémoire

L'adressage

Voir paragraphe 5.2.4 page précédente à propos de la mémoire. Les adresses mémoire sont des numéros et sont donc susceptibles d'êtres codés sur des bits.

La numérotation va de 0 à $n-1$, toutes les cases contiennent le même nombre de bits (exemples). Pour une adresse sur n bits, on peut adresser 2^n cases (faire calculer le nombre d'adressages pour 32 bits, les systèmes actuels, en sachant qu'une case contient un octet dans le cas de l'IBM PC).

Ordonnancement des octets

Différence entre le gros-boutiste et le petit-boutiste (big endian et little endian).

Codes de contrôle

Problème du stockage des données sur un support physique, donc susceptible de subir des dégradations physiques (ex. de la machine qui plante tous les jours vers 14h à cause du soleil qui vient frapper sur les RAMs). On utilise des codes de contrôle d'erreur pour vérifier si les informations lues sont réellement les bonnes.

Plusieurs algorithmes. On ne verra qu'un contrôle de parité simple, mais la méthode de Hamming permet de détecter des erreurs et aussi des les corriger. Le contrôle de parité sert lorsqu'on peut ré-émettre des informations correctes (ex. du réseau ou du transfert sur des bus de transmission).

Contrôle de parité : on sacrifie un bit par transmission de mot, et ce bit vaudra 1 si le nombre de bits à 1 est impair, 0 s'il est pair (contrôle simple). Faire l'exemple de la corruption d'un bit de donnée ou même du bit de contrôle.

5.3 La couche conventionnelle

La couche conventionnelle permet de décoder et d'exécuter les instructions de la couche micro-programmée. L'ensemble des instructions possibles est appelé le *jeu d'instructions*, qui peut varier d'une architecture à l'autre (ex. du Tanenbaum p. 332 à 333).

Chaque instruction (un mot, soit un champ de bits) est séparée en plusieurs parties :

- l'instruction ;
- la première opérande ;
- la deuxième opérande.

5.3.1 L'adressage

Les opérandes sont donc des données, qui doivent être stockées en mémoire. Il existe plusieurs types de mémoires, les manières d'accéder à ces données sont donc multiples :

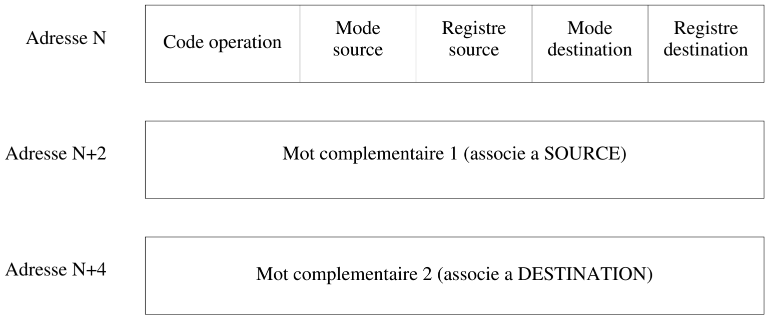


FIG. 5.8 – Format de l'instruction du PDP-11

- adressage immédiat : on place les données directement dans l'instruction. Problème : on est limité en codage de donnée ;
- adressage direct : on place l'adresse d'une case de la mémoire principale qu'on va aller chercher ensuite ;
- adressage par registre : on place la donnée dans un registre et on place l'adresse du registre dans l'instruction. C'est plus court car les registres étant plus petits, l'adresse est plus courte ;
- adressage indirect : on place l'adresse d'un registre qui contient un adresse de la mémoire principale ;
- adressage indexé : on définit des plages d'adressage et on se contente de donner l'index de la donnée dans cette plage (plus courte) ;
- adressage par pile : on empile des données en mémoire et on ne conserve qu'un pointeur sur le sommet de la pile. On a donc des adressages de taille zéro. Mais nécessité de définir des instructions PUSH et POP.

Exemple du PDP-11 qui utilise 8 modes d'adressage. Un des intérêts et de pouvoir effectuer une opération sur des mots mémoire placés sur des mémoires différentes de manière totalement transparente (ex. de l'opération ADD).

5.3.2 Le branchement conditionnel

Les programmes d'ordinateurs on souvent besoin de tester les résultats intermédiaires en cours d'exécution (exemple de $\sqrt{n}$ quand n vaut -1) et d'effectuer un traitement spécifique en cas de besoin.

Le principe est d'effectuer ce qu'on appelle un *branchement conditionnel* ou *rupture de séquence* ou *saut*. Concrètement, on effectue un test simple (par ex. on vérifie si une donnée est supérieure à zéro) et si le test est faux, alors on va modifier le compteur ordinal et le faire pointer vers une adresse donnée. Sinon, on se contente de l'incrémenter.

5.3.3 Les procédures

Principe fonctionnel : on effectue des tâches (ou procédures) atomiques pour arriver à un tout. Exemple d'une succession de coursiers qui vont effectuer chacun une infime partie d'une mission.

Exemple du bâtiment. Le grutier va apporter des parpaings aux massons, qui vont à leur tour donner des gravats au grutier afin qu'il puisse redescendre son chargement. Chacun a son programme, et le grutier suspend son programme pendant que le programme des maçons s'exécute. On a un branchement de séquence, mais le problème c'est de restaurer le compteur ordinal afin qu'il pointe sur l'instruction suivant le branchement de séquence. On a donc besoin de conserver cette adresse, mais comment ?

La meilleure solution est d'enregistrer les valeurs dans une pile, de les empiler à chaque appel de fonction et de les dépiler à chaque fin de fonction.

5.4 Un exemple, l'ordinateur en papier

L'ordinateur en papier est un schéma simplifié de l'ordinateur.

On retrouve les composants de l'ordinateur de Von Neumann :

- l'UAL ;
- l'UC ;
- la mémoire centrale ;
- les E/S.

L'UAL effectue les opérations arithmétiques et logiques et stocke le résultat dans un registre de l'UC qu'on appelle l'*accumulateur* (car on y « accumule » les résultats successifs).

Chapitre 6

Conditions de distribution

GNU Free Documentation License Version 1.0
DRAFT

0. PREAMBLE

The general idea of copyleft is to use the copyright system to ensure that everyone who possesses a copy or a derivative of a certain work has the effective freedom to copy and redistribute it, with or without modifying it. A copyleft license gives you certain rights, above and beyond the meager rights copyright law allows you. For example, the GNU General Public License is a copyleft license designed for free software.

The GNU Project designed this, the GNU Free Documentation License, for use with documentation about free software. Copylefting the documentation for free software is particularly important because free software needs to come with documentation that gives users the same freedoms that the software gives them. However, this License can be used for any textual work, regardless of its subject matter, and regardless of whether it is released in the form of a printed book. For short works which become much larger if this License is added, we recommend simply putting them in the public domain.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Manual", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Manual means any work containing the Manual or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Manual that deals exclusively with relationship of certain parties to the Manual's overall subject (or to related matters) and contains nothing that could fall properly within that overall subject. The relationship could be a matter of historical connection with the subject or related matters, or a legal, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as those of Invariant Sections, in the notice that says that the Manual is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Manual is released under this License.

A "Transparent" copy of the Manual means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not

Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Manual in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Manual are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

It is requested, but not required, that you contact the authors of the Manual well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Manual.

3. COPYING IN QUANTITY

If you publish printed copies of the Manual numbering more than 100, and the Manual's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts : Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the manual and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Manual numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Manual, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least six months after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Manual under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Manual, thus licensing use of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version :

A. Entitle your Modified Version with the Manual's title, plus some additional words (or a subtitle) stating that the version has been modified, and distinguishing it from the Manual you started with. (Additions to the title are not required if the original publisher of the version you started with waives this requirement.)
B. Mention the Manual's title on the Title Page (and on the Covers, if any).

C. Mention on the Title Page at least one name of a person or entity responsible for authorship of the modifications in the Modified Version and/or publication of the Modified Version, and describe that entity's relationship to the Modified Version. (This requirement may be waived by the original publisher of the version you started with.)
D. Retain on the Title Page or its continuation the authors' and publishers' names listed on the Manual's Title Page.
E. Preserve all the copyright notices of the Manual.
F. Add an appropriate copyright notice for your own work.
G. Include immediately after the copyright notices a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
H. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Manual's license notice.
I. Include an unaltered copy of this License.
J. Preserve the network location, if any, given in the Manual for public access to a Transparent copy of the Manual, and likewise those network locations given in the Manual for any earlier works it was based on (but you may delete those for works that were published at least four years before the Manual itself). This requirement may be waived for a certain network location by the original publisher of the version it refers to.
K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Manual, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Manual, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Manual already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another ; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Manual do not by this License give permission to use their names for publicity or to assert or imply endorsement of any Modified Version.

5. COMBINING MANUALS

You may combine the Manual with other manuals released under this License, under the terms defined in section 4 above for modified versions, provided that you include all of the Invariant Sections of all of the original manuals, unmodified, in the combination, and list them all as Invariant Sections in your combined work.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section title in the list of Invariant Sections in the license notice of the combined work.

6. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Manual or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Manual, provided no compilation copyright is claimed for the compilation. In such a compilation, this License does not apply to the other self-contained works thus compiled with the Manual, on account of their being thus compiled, if they are not themselves derivative works of the Manual. If the Manual is less than one quarter of the entire aggregate, the Manual's Cover Texts may be placed on covers that surround only the Manual within the aggregate. Otherwise they must appear on covers around the whole aggregate.

7. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the manual under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

8. TERMINATION

You may not copy, modify, sublicense, or distribute the Manual except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Manual is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version of the License is given a distinguishing version number. If the Manual specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published by the Free Software Foundation. If the Manual does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. ADDENDUM : How to use this License for your manuals

To use this License in a manual you have written, include a copy of the License in the manual and put the following copyright and license notices just after the title page :

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this manual under the terms of the GNU Free Documentation License, Version 1.0 or any later version published by the Free Software Foundation ; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST" ; likewise for Back-Cover Texts.

If your manual contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.