ASR1 2007
Protopoly

Florent de Dinechin
avec des figures de
R. Bergasse,
N. Bonifas,
A. Derouet-Jourdan,
A. Friggeri,
B. Grenet,
F. Givors,
C. Keller,
E. Lassalle,
P.E. Meunier,
O. Schwander,
P. Vannier

2 octobre 2007

Table des matières

Chapitre 1

Introduction

L'informatique c'est la science du traitement de l'information.

Un ordinateur est une *machine universelle de traitement de l'information*. Universelle veut dire : qui peut réaliser toute transformation d'information *réalisable*.

Il existe des calculs non réalisables, et de calculs non réalisables en temps raisonnable. C'est pas que ce n'est pas important, mais on ne s'y intéresse pas dans ce cours.

Un bon bouquin de support de ce cours est *Computer Organization & Design, the hardware/software interface*, de Patterson et Hennessy. Il y a aussi *Architecture de l'Ordinateur*, 74ème édition (au moins) par Tannenbaum, il existe en français. Mon préféré du moment est *Computer Architecture and Organisation* de Murdocca et Heuring, il devrait arriver un jour à la bibliothèque.

1.1 Historique du calcul mécanique

néolithique Invention du système de numération unaire, de l'addition et de la soustraction (des moutons)
(en latin, *calculus* = petit caillou)

antiquité Systèmes de numération plus évolués :

systèmes alphabétiques (Égypte, Grèce, Chine) :
chaque symbole a une valeur numérique fixe et indépendante de sa position.
Les chiffres romains sont un mélange de unaire et d'alphabétique.

systèmes à position (Babylone, Inde, Mayas) :
c'est la position du chiffre dans le nombre qui donne sa puissance de la base.

Les bases utilisées sont la base 10 (Égypte, Inde, Chine), la base 20 (Mayas), la base 60 (Sumer, Babylone), ...

Ces inventions sont guidées par la nécessité de faire des calculs

Essayez donc de décrire l'algo de multiplication en chiffres romains... Par contre la base 60 c'est bien pratique, pourquoi ?

≈**-1000** Invention du calculateur de poche (l'abaque ou boulier) en Chine.

+1202 Le génial système de numération indo-arabe arrive en Europe par l'Espagne.

1623 Sir Francis Bacon (Angleterre) décrit le codage des nombres dans le système binaire qu'il a inventé dans sa jeunesse.

1623 Wilhelm Schickard (Tübingen) invente la roue à chiffres qui permet de propager les retenues.

1624 Il construit la première calculette "occidentale"

1645 Blaise Pascal en fabrique une mieux qui peut propager les retenues sur les grands nombres.

1672 Gottfried Wilhelm Leibniz construit une machine à calculer 4 opérations

1679 Leibniz encore développe l'arithmétique binaire (De Progressione Dyadica) mais sans l'idée que cela pourrait servir à quelque chose.

1728 Première utilisation connue des cartes perforées (Falcon ?)

1741 Jacques de Vaucanson invente la mémoire de programme dans le contexte des métiers à tisser. Il utilise des rouleaux de fer-blanc perforés.

1808 Joseph Marie Jacquard utilise du carton perforé, c'est moins cher.

1822 Charles Babbage se lance dans sa *difference engine*, qui sert à calculer des polynômes.

1833 Karl Friedrich Gauß et Wilhelm Weber sont les précurseurs de l'internet en s'envoyant à distance du texte codé par un code à 5 bits.

1833 Babbage laisse tomber car il a une meilleur idée, l'*analytical engine*, programmable par cartes perforées, inconstruisible avec les moyens de l'époque.

1854 Georges Boole (Angleterre) met en place la logique mathématique désormais dite booléenne.

1854 Christopher L. Sholes (USA) : première machine à écrire utilisable.

1928 Ackermann montre qu'il y a des fonctions entières qui ne peuvent être calculées par un nombre fini de boucles.

1900-1935 Développements en électronique : tube cathodique, tube à vide 3 électrodes, enregistrement sur support magnétique.

1936 Thèse d'Alan M. Turing sur une machine abstraite universelle. Début de la théorie de la calculabilité.

1936 Konrad Zuse (Allemagne) construit le premier calculateur binaire (Z1 : mécanique, Z2 : à relais)

1937 John V. Atanasoff (USA) a l'idée d'utiliser le binaire pour des calculateurs.

1941 Le Z3 de Zuse est le premier calculateur (presque) universel programmable
- 1400 relais pour la mémoire, 600 pour le calcul
- 64 mots de mémoire
- arithmétique binaire en virgule flottante sur 22 bits
- programmation par ruban perforé de 8 bits
- une seule boucle

1946 L'ENIAC est le premier calculateur *électronique*, mais à part cela c'est un boulier.

1939-1945 C'est la guerre, tout le monde construit des calculateurs, surtout pour la balistique et la cryptographie.

1949 Turing et von Neumann, ensemble, construisent le premier ordinateur universel électronique, le Manchester Mark I. L'innovation c'est la mémoire partagée programme et donnée.

Depuis on n'a pas fait tellement mieux. Enfin si, mais les innovations sont difficiles à expliquer *avant* le cours d'ASR...

1.2 Objectifs du cours

On va construire un ordinateur moderne, en essayant de se concentrer sur les techniques qui sont indépendantes de la technologie (en principe on pourra construire notre ordinateur en utilisant l'électronique actuelle, mais aussi en Lego, ou bien avec des composants moléculaires opto-quantiques à nanotubes de carbone).

C'est quoi un ordinateur moderne ? C'est la figure 1.1.

Et il y a une idée capitale cachée dans ce dessin, et que personne n'avait eu avant la bande à von Neumann : *C'est la même mémoire qui contient le programme et les données.*

C'est génial car cela permet par exemple

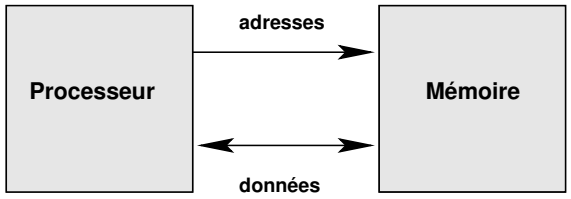

FIG. 1.1 – S'il te plait, dessine-moi un ordinateur

– le *système d'exploitation* : un programme qui prend un paquet de données sur un disque dur, les met en mémoire, et ensuite décide que ces données forment un programme, et exécute ce programme.
– le *compilateur* qui prend un texte (des données) et le transforme en un programme...

La mémoire est un ensemble de cases mémoires "type-agnostiques" : dans chaque case on met une information qui peut être interprétée n'importe comment. On attrape les cases ar leur adresse.

Le processeur réalise le *cycle de von Neumann*

1. Lire une case mémoire d'adresse `PC` (envoyer l'adresse à la mémoire, et recevoir en retour la donnée à cette adresse).
2. Interpréter cette donnée comme une instruction, et l'exécuter
3. Ajouter 1 à `PC`
4. Recommencer

PC c'est pour *program counter*, bande de gauchistes.
La fréquence de votre ordinateur favori, c'est la fréquence à laquelle il exécute ce cycle.

1.3 La conception d'ASR est hiérarchique

Selon le bon vieux paradigme *diviser pour régner*, on est capable de construire l'objet très complexe qu'est votre PC par assemblage d'objets plus simples. Au passage, on utilise différents formalismes (ou différentes abstractions) pour le calcul (arithmétique binaire, fonctions booléennes, etc), pour la maîtrise des aspects temporels (mémoires, automates, etc), pour les communications (protocoles)... La figure 1.2 décrit cet empilement.

Dans ce cours on va avoir une approche de bas en haut *(bottom-up)*, mais avec des détours et des zig-zags.

Maintenant qu'on a une recette pour faire des systèmes complexes, voyons les limites pratiques à cette complexité.

1.4 Quelques ordres de grandeur

1.4.1 L'univers est notre terrain de jeu

Il y a quelques limites aux ordinateurs qu'on saura construire, par exemple :
– $\frac{\text{Masse de l'univers}}{\text{Masse du proton}} \approx 10^{78}$
(ce qui limite le nombre de transistors sur une puce. Actuellement on est à 10^{10}, il y a de la marge).
– Vitesse de la lumière : 3.10^8m/s
– Distance entre deux atomes : $\approx 10^{-10}$ m.
(Actuellement, un fil ou un transistor font quelques dizaines d'atomes de large : 27 en 2006, avec des couches d'isolant de 3 atomes d'épaisseur)

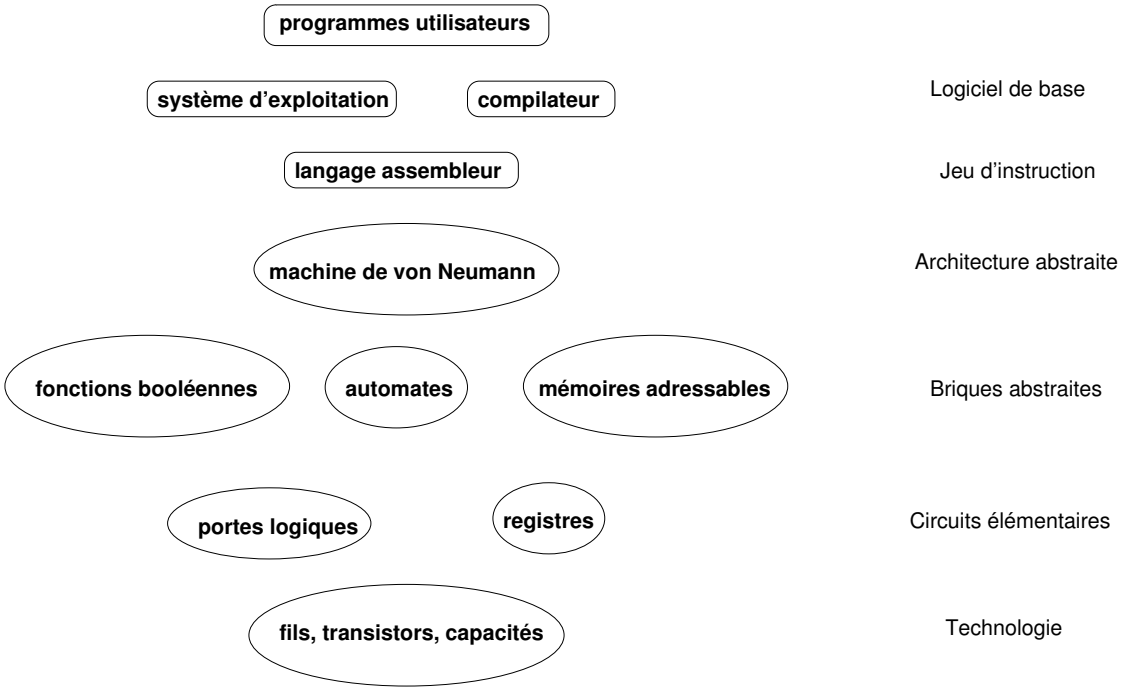

FIG. 1.2 – *Top-down* ou *bottom-up* ?

- Par conséquent, le temps minimum physiquement envisageable pour communiquer une information est de l'ordre de $\approx 10^{-18}$s.
- On a fait plus de la moitié du chemin, puisque nos transistors commutent à des fréquences de l'ordre de 10^{12} Hz.
- Actuellement, un signal n'a pas le temps de traverser toute la puce en un cycle d'horloge.
- Actuellement, on manipule des charges avec une résolution correspondant à 200 électrons. Cela permet à votre téléphone portable de gérer des fréquences au Hz près dans les 600MHz

1.4.2 La technologie en 2006

Les ordinateurs actuels sont construits en assemblant des transistors (plein). La figure 1.4.2 montre les progrès de l'intégration des circuits électroniques. Il y a une loi empirique, formulée par un dénommé Moore chez Intel dans les années 60 et jamais démentie depuis, qui décrit l'augmentation exponentielle de la quantité de transistors par puce (la taille de la puce restant à peu près constante, de l'ordre de 1cm^2).

Pour illustrer le chemin accompli : le premier processeur comptait 2300 transistors. De nos jours on pourrait mettre 2300 processeurs 32-bit complets dans le budget de transistors d'un pentium.

Ce doublement tous les deux ans se traduit par exemple directement par un doublement de la mémoire qu'on peut mettre dans une puce. Mais ce n'est pas tout : les transistors étant plus petits, ils sont aussi plus rapides (j'expliquerai peut-être avec les mains pourquoi, admettez en attendant). La puissance de calcul des processeurs peut donc augmenter donc plus vite que ce facteur deux tous les deux ans (les processeurs sont aussi de plus en plus complexes, ce qui tire dans l'autre sens).

Enfin, c'était vrai jusqu'aux années 2000 : en réduisant la taille du transistor, on en mettait plus sur la puce, ils étaient plus rapides, et ils consommaient moins. Avec les dernières générations (dites sub-microniques, c'est-à-dire en gros que le transistor fait moins d'un micron), cela se passe moins bien : on réduit toujours le transistor, cela permet toujours d'en mettre plus par puce, mais

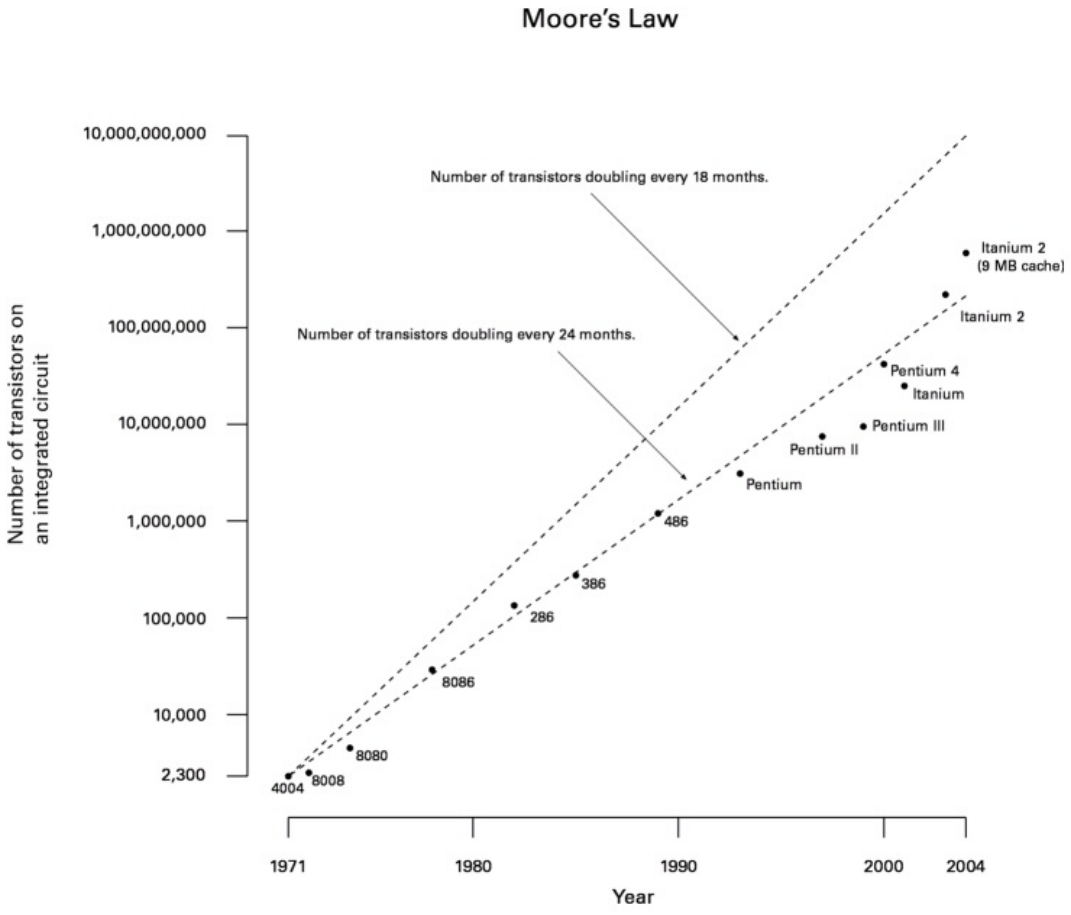

FIG. 1.3 – La "loi" de Moore (©Wikipedia/creative commons)

ils sont de moins en moins plus rapides et consomment de plus en plus (en 2007, sur les 150W que consomme le processeur de votre PC de gamerz, il y en a 50 qui partent en courants de fuite). Une raison facile à comprendre est qu'on arrive à des fils tellement petit que les électrons passent par effet tunnel d'un fil à l'autre. Il y a d'autres raisons. En pratique, les experts ne donnent pas 10 ans de plus à la loi de Moore pour le silicium.

Autre évolution de la technologie : l'investissement (en terme de megadollars) pour passer d'une génération technologique à la suivante double aussi avec chaque technologie... Cela se traduit par des regroupement de firmes pour partager ces coûts, et à ce train, il n'y aura plus qu'un seul fabricant de circuits intégrés dans dix ans.

Tout ceci pour dire que d'ici à ce que les plus brillants d'entre vous fassent à leur tour un cours d'architecture en Licence, la technologie sera sans doute en train de changer profondément. Vers quoi, je ne sais pas. On cherche une techno qui permette d'utiliser les 3 dimensions, et pas juste 2. Et peut-être de la transmission d'information par la lumière, qui a plein d'avantages en principe sur les déplacements d'électrons. Mais en tout cas je vais essayer de ne pas trop perdre du temps sur la technologie actuelle. Par contre je la connais assez pour répondre à vos questions.

Un dernier aspect technologique : les transistors de plus en plus petits sont de moins en moins fiables, et il faut le prévoir quand on fait des circuits. Par exemple, on évalue en 2007 que les rayons cosmiques font statistiquement changer un bit par puce et par mois... Ce sont des "soft

errors", la puce n'est pas endommagée. Les mémoires actuelles sont munies de dispositifs d'autocorrections.

1.4.3 Et maintenant, un peu de modestie

Mon théorème préféré :

Si un graphe ne possède pas de sous-graphe isomorphe au graphe complet d'ordre 4, et si, pour tout coloriage de ses arêtes à l'aide de deux couleurs, il existe un triangle monochromatique, alors le nombre de ses sommets est supérieur à $F = 10^{10^{10^{10^{10}}}}$ *(nombre de Folkmann).*

Courrez toujours, un ordinateur construit avec tous les atomes de l'univers et tournant jusqu'à la fin de l'univers n'arrivera pas à énumérer un tel graphe. Cela dit il y a des nombres plus grand : cherchez sur Wikipedia le nombre de Graham par exemple.

1.4.4 Ya pas que les PC dans la vie

Juste trois exemples :

Vous croyez que le processeur le plus vendu au monde est la famille x86 (pentium etc) ? Eh bien non, c'est la famille ARM, un processeur 32 bits conçu pour coûter pas cher et consommer peu d'énergie, et qui équipe la plupart des téléphones portables (sans compter les GameBoy, les organiseurs, les machines à laver, etc). Eh oui, l'arrière grand-mère de mes enfants, en Russie profonde, en est à son troisième *mobilnik* et n'aura jamais de PC. La performance d'un ARM est comparable à celle du Pentium cinq ans avant, sauf pour la consommation qui est divisée par 1000.

Une vieille télé Philips de 2002 contient une puce (Viper2) qui doit traiter 100 GOps (10^{11} opérations par seconde). Elle contient 4 microprocesseurs programmables, 250 RAMs, 60 autres blocs de calculs divers, 100 domaines d'horloge différents, et en tout 50Mtransistors tournant à 250 MHz (source : M. Duranton, Philips, Euromicro DSD 2006).

Et bien sûr, je vais vous parler des circuits reconfigurables de type FPGA.

Première partie

L’information et comment on la traite

Chapitre 2

Coder

2.1 Information et medium

La notion d'information est une notion abstraite. On appelle medium un support physique de l'information, que se soit pour la stocker (CD, journal) ou la transmettre (fil, onde radio, écran télé).

Le medium a un coût, l'information aussi, il faut bien distinguer les deux. Quand on achète un CD, on paye 1 euro le medium, et 10 euros l'information. Quand on le pirate, on économise le coût de l'information, mais si on le grave on n'économise pas le coût du medium. Quand on convertit un CD légalement acheté en mp3 sur son disque dur, on a dupliqué l'information, et on n'est pas deux fois plus riche pour autant.

2.2 Information analogique

L'information peut être discrète ou continue. La nature est pleine d'informations continues, que l'on sait enregistrer sur des supports analogiques (disque vinyle, photo argentique, etc), et transmettre par fil, par radio, etc. Les machines qui traitent ce genre d'information sont dites analogiques. Les traitements les plus courants sont *amplification* et *filtrage*.

Quelques exemples de *calculateurs analogiques* plus généraux :

- les horloges astronomiques ;
- Mr Thompson/Lord Kelvin a construit à base de roues dentées une machine à prédire les marées dans chsais plus quel port ;
- dans les années 50, il y avait à Supelec une "salle de calcul analogique" où l'on pouvait brancher des amplis ops entre eux pour simuler, plus vite, des phénomènes physiques. Par exemple, on a pu ainsi simuler tout le système de suspension de la Citroen DS avant de fabriquer les prototypes.

Ces derniers temps, on s'est rendu compte que c'est plus propre et plus facile de *discrétiser* l'information d'abord et de la traiter ensuite avec un calculateur numérique.

Jusqu'aux années 90, il ne restait d'analogique dans les circuits intégrés que les partie qui devaient produire des ondes à hautes fréquence : les téléphones mobiles de la première génération se composaient d'une puce numérique, et d'une puce analogique branchée sur l'antenne. Les transistors actuels savent commutent tellement vite qu'on peut tout faire en numérique. Ce fut une vraie révolution, mais personne n'a rien remarqué. Un avantage supplémentaire est que les composants analogiques ne peuvent pas être miniutarisés autant qu'on veut : dans la surface typique d'une inductance, on pouvait mettre un processeur ARM en 0.9μm, et on peut en mettre 2 en 0.65μm.

2.3 Information numérique

Si elle est discrète, l'unité minimale d'information est le *bit* (contraction de *binary digit*), qui peut prendre juste deux valeurs, notées 0 ou 1 (mais vous pouvez les appeler vrai et faux, ou $\emptyset$ et $\{\emptyset\}$, ou yin et yang si cela vous chante).

Pour coder de l'information utile on assemble les bits en vecteurs de bits. Sur n bits on peut coder 2^n informations différentes.

Une information complexe sera toujours un vecteur de bits. Le code est une bijection entre un tel vecteur et une grandeur dans un autre domaine.

2.4 Coder des nombres entiers

Il y a plein de manières de coder les entiers en binaire. Toute bijection de $\{0,1\}^n$ dans un ensemble d'entiers fait l'affaire.

Toutefois, le précepte qui reviendra tout le temps, c'est : un bon code est un code qui permet de faire facilement les traitements utiles. Sur les entiers, on aime avoir la relation d'ordre, et faire des opérations comme addition, multiplication...

Le codage en unaire fut sans doute le premier. On y représente un nombre n par un tas de n cailloux (ou batons, ou "1"). Le gros avantage de ce codage est que l'addition de deux nombres est réalisable par une machinerie simple qui consiste à mélanger les deux tas de cailloux. La comparaison peut se faire par une balance si les cailloux sont suffisamment identiques, sinon vous trouverez bien quelque chose. Plus près de nous, j'ai lu récemment un papier sérieux proposant des circuits *single electron counting* : il s'agit de coder n par n électrons dans un condensateur. La motivation est la même : l'addition de deux nombres consiste à vider un condensateur dans l'autre. La comparaison se ramène à une différence de potentiel.

Le gros problème du codage unaire est de représenter de grands nombres. Pour cela, on a inventé les codages de position comme par exemple celuis que vous avez appris en maternelle. La version la plus simple est le codage binaire : un tuple $(x_0, x_1, ...x_n)$ représente par exemple un entier $X = \sum_{i=0}^{n-1} 2^i x_i$.

Si on veut gérer des entiers relatifs, on peut ajouter un bit qui contient l'information de signe. On verra qu'il y a des codages un peu moins intuitifs qui permettent de faire les additions/soustractions plus facilement.

Lorsqu'on construit un calculateur, on est souvent appelé à superposer des codages. Par exemple,le boulier code des grands nombres en base 10, et chaque chiffre est représenté par un mélange de binaire et d'unaire. Encore une fois, la motivation est de faciliter le calcul (on peut reconnaître les chiffres d'un coup d'œil, les additionner d'un mouvement du doigt, etc). Les calculettes fonctionnent également en base 10, avec chaque chiffre codé en binaire. Mon papier sur *single electron counting* reconnait qu'il faut limiter le nombre maximum d'électrons dans un condensateur, et utilise ces condensateurs comme chiffres.

2.5 Coder du texte

Second exemple : le texte. On a un alphabet de 26 lettres, plus les chiffres, plus les majuscules et la ponctuation. Tout cela tient en moins de $128 = 2^7$ caractères, donc on peut coder chaque caractère sur 7 bits. C'est ce que fait le code ASCII.

On l'a déjà dit : un bon code est un code qui permet de faire facilement les traitements utiles. Par exemple, pour le texte,

- il faut que l'ordre alphabétique se retrouve dans les codes, considérés comme des entiers ;
- idem pour les chiffres ;
- il faut qu'on puisse passer de la majuscule à la minuscule en ne changeant qu'un bit ;

ASCII a été inventé par des Américains. Il a été étendu de diverses manières pour d'autres langues, en passant à 8 bits, ce qui permet 128 caractères supplémentaires. Par exemple, il existe deux codes populaires pour le Russe, chacun cohabitant avec les caractères de l'anglais :

- un code qui, lorsqu'on annule le bit 8, projette chaque lettre russe sur la lettre anglolatine qui lui correspond le mieux phonétiquement ;
- un code qui est dans l'ordre alphabétique de l'alphabet cyrillique.

La norme Unicode a unifié dans un seul code, sur 16 bits, l'ensemble des écritures de la planète. C'est une usine à gaz, à l'image de la complexité des écritures des langages de l'humanité. Vous en verrez peut-être des morceaux en TD.

2.6 Coder les images et les sons

On décompose une image en pixels, et on code pour chaque pixel sa couleur. Une couleur est la somme de trois composantes, par exemple vert, bleu, rouge. En codant chaque composante sur 8 bits, on peut déjà coder plus de couleurs que ce que l'œuil humain peut distinguer.

Cela donne des quantités d'information astronomiques pour chaque image. On verra comment on peut *compresser* cette information.

Pour le son, on peut le regarder à l'oscilloscope et discrétiser la courbe obtenue. La nouveauté est qu'il faut discrétiser dans le temps et dans l'amplitude.

Avec tout cela, on sait même coder des vidéos.

Remarque : dans un programme de dessin vectoriel (CorelDraw, XFig) on ne code pas des images, mais des figures construites à partir de primitives telles que point, droite, cercle...

2.7 Codes correcteurs d'erreur

En TD

2.8 Compression d'information

En TD

2.9 Coder le temps

Le temps physique est continu, on peut le discrétiser. Parmi les instruments de mesure du temps, les plus anciens utilisent une approche analogique du temps continu (clepsydre, sablier). Les plus récents utilisent une approche numérique/discrète : horloge à balancier, à quartz... Ici aussi, le passage au monde discret (par le balancier) permet ensuite de transformer cette information sans perte : les roues dentées de l'horloge ont un rapport qui est une pure constante mathématique, ce qui permet de construire un dispositif dans lequel une heure fait toujours exactement 3600 secondes. Il ne reste dans une telle horloge qu'un seul point de contact avec le monde analogique : le balancier, qui donne la seconde. Une montre à quartz c'est pareil.

Plus pratiquement, on codera toujours un *instant* par un *changement* d'information. C'est vite dit mais il faut s'y arrêter longtemps.

Chapitre 3

Transformer

On a déjà mentionné la préhistoire analogique. Remarquez que le calcul analogique à Supelec utilisait déjà l'approche hiérarchique : il fallait se ramener à des amplis ops.

Mais désormais on manipule de l'information sous sa plus simple expression : codée en binaire.

Et pour transformer de l'information binaire, on va utiliser les outils offerts par l'algèbre booléenne.

Vérifiez que vous comprenez la différence entre *algèbre*, *calcul*, *expression* et *circuit* booléens.

- L'algèbre booléenne va définir un certain nombre d'opérateurs et de propriétés.
- On utilisera les opérateurs pour construire des expressions booléennes, et on utilisera les propriétés pour les manipuler.
- En 3.2, on implémentera ces expressions sous forme de circuits (dans une technologie donnée), en se posant notamment des questions de coût et de performance.

3.1 Algèbre booléenne

3.1.1 Définitions et notations

L'algèbre booléenne définit

- un ensemble $\mathbb{B}$ à deux éléments, muni de
- une opération unaire involutive (la négation)
- et deux opérations binaires de base (ET et OU) commutatives, associatives, ayant chacun un élément neutre, et distributives l'une par rapport à l'autre [1].

On utilise des mélanges variables des notations suivantes :

- Les valeurs sont notées (vrai, faux) – ou une traduction dans la langue de votre choix –, ou $(0, 1)$, ou $(\emptyset, \{\emptyset\})$, ou $(\top, \bot)$.
- L'opération unaire est notée NON – ou une traduction dans la langue de votre choix –, ou $\neg$, ou C (complément ensembliste), ou $\overline{\text{toto}}$.
- Les opérations binaires sont notées (OU, ET) – ou une traduction dans la langue de votre choix –, ou $(+, .)$, ou $(\vee, \wedge)$, ou $(\cup, \cap)$.

Je m'économise d'écrire les tables de vérité.

3.1.2 Expression booléenne

Lorsqu'on écrit des expressions booléennes, on peut utiliser des variables booléennes, les deux constantes, et des parenthèses. L'opérateur unaire est prioritaire sur les deux opérateurs binaires, qui ont une priorité équivalente : $\neg a \vee b == (\neg a) \vee b$.

[1] Un peu plus loin, on voit qu'une seule opération suffirait, mais c'est tout de même plus confortable avec ces trois-là.

Je déconseille la notation utilisant (+,.). Elle présente l'intérêt d'économiser des parenthèses, puisqu'on adopte alors la priorité usuelle de . sur $+$. En revanche, l'arithmétique entière ou réelle a câblé dans votre cerveau des intuitions avec ces opérateurs qui seront fausses en booléen. Par exemple, les deux opérateurs booléens distribuent l'un sur l'autre. Vous serez familier de $(a + b).c = ac+bc$, mais vous oublierez que $ab+c = (a+c).(b+c)$ (démonstration : comme toujours pour prouver des identités booléennes, considérer $c = 0$ puis $c = 1$). D'autres pièges vous attendent si vous utilisez cette notation. Vous voila prévenus.

La notation à base de surlignage pour la négation est pratique, car elle permet d'économiser la plupart des parenthèses sans ambiguité. Exemple :

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

3.1.3 Dualité

L'algèbre booléenne est parfaitement symétrique : pour toute propriété, il existe une autre propriété déduite en échangeant 1 avec 0 et $\vee$ avec $\wedge$. Cette idée pourra souvent économiser du calcul. Encore une fois, elle est particulièrement contre-intuitive avec la notation $(+, .)$ à cause des priorités héritées de l'algèbre sur les réels.

3.1.4 Quelques propriétés

$0 \wedge x = 0$	$1 \vee x = 1$	(élément absorbant)
$1 \wedge x = x$	$0 \vee x = x$	(élément neutre)
$\overline{x \wedge y} = \overline{x} \vee \overline{y}$	$\overline{x \vee y} = \overline{x} \wedge \overline{y}$	(lois de Morgan)

3.1.5 Universalité

On peut ramener nos trois opérations booléennes à une seule, le non-et, par application des règles suivantes :

- Négation : $\overline{x} = \overline{x \wedge x}$
- Ou : $x \vee y = \overline{\overline{x} \wedge \overline{y}} = \overline{\overline{x \wedge x} \wedge \overline{y \wedge y}}$
- Et : $x \wedge y = \overline{\overline{x \wedge y}} = \overline{\overline{x \wedge y} \wedge \overline{x \wedge y}}$

On dira que l'opération non-et est universelle. Par dualité, non-ou aussi. Cela n'est pas sans nous interpeler profondément dans notre relation à la transcendance de l'Univers, croyez-vous ? Eh bien figurez vous que les portes de base de notre technologie CMOS seront justement NON-ET et NON-OU. Nous voila convaincus qu'elles ne sont pas plus mauvaises que ET et OU. Plus précisément, non seulement elles forment un ensemble universel, mais en plus, pour implémenter une fonction donnée, il faudra un nombre équivalent de portes, qu'on se ramène à des NON-ET/NON-OU ou bien à des ET/OU.

3.1.6 Fonctions booléennes

Une fonction booléenne est une fonction d'un vecteur de bits vers un vecteur de bits :

$$f : \mathbb{B}^n \to \mathbb{B}^m$$

En général (pas toujours) on considérera une telle fonction comme un vecteur de fonctions simples $f_i : \mathbb{B}^n \to \mathbb{B}$, et on étudiera séparément chaque f_i.

On peut définir une fonction booléenne de diverses manières :

- par extension (en donnant sa table de vérité) – exemple une table de sinus,
- par intention (en donnant des propriétés qu'elle doit vérifier, l'unicité n'est alors pas garantie) – exemple la somme de deux chiffres BCD, voir ci-dessous,
- par une expression algébrique – exemple, la fonction NON-ET.

On peut donner une définition d'une fonction par sa table, mais avec des "don't care" pour les sorties. Exemple : la fonction d'addition de deux chiffres BCD (pour *binary-coded decimal*). Il s'agit des chiffres décimaux de 0 à 9 codés en binaire sur 4 bits. Les codes entre 10 et 15 n'étant jamais utilisés, on se fiche (don't care) de ce que fait la fonction dans ce cas. Formellement, cela revient à donner une définition intentionnelle de la fonction en question, et plusieurs fonctions booléennes différentes feront l'affaire.

3.2 Circuits logiques et circuits combinatoires

3.2.1 Signaux logique

Définition : un signal logique, c'est un dispositif physique pouvant transmettre une information binaire d'un endroit à un autre.

C'est un cas particulier, on peut considérer des signaux qui transmettent des informations non binaires. On définira aussi plus tard des signaux "trois états".

On dessinera un signal par un trait, ou éventuellement plusieurs. Cela colle avec la technologie la plus courante, dans laquelle un signal sera implémenté par un fil métallique. La valeur du signal sera codée par le potentiel électrique de ce fil. La transmission du signal se fera par un courant électrique.

La notion de signal logique est plus générale que cela : vous pourrez à la fin de ce cours réaliser un ordinateur complet en lego, dans lequel un signal binaire sera implémenté par une tige pouvant prendre deux positions.

3.2.2 Circuits logiques

Définition : un circuit logique, c'est un dispositif physique destiné à manipuler des informations binaires.

Un circuit logique se présente, vu de l'extérieur, comme une boîte noire avec des signaux d'entrée et des signaux de sortie.

3.2.3 Portes de base

Voici les dessins des portes ET, OU, NON.

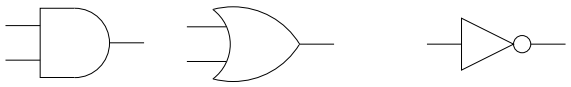

FIG. 3.1 – porte ET, porte OU, porte NON (inverseur)

On peut mettre des petits ronds sur les entrées et les sorties pour dire qu'on les inverse.

Exercice : construisez-les en Lego. Tout ensemble *universel* (au sens du 3.1.5) de portes de base fera en pratique l'affaire, si on en a plus on aura plus de liberté pour optimiser, voir plus bas.

3.2.4 Circuits combinatoires

Définition : un circuit combinatoire, c'est un dispositif physique implémentant une expression booléenne dans une certaine technologie.

Le circuit combinatoire est un cas particulier de circuit logique avec une grosse restriction : *il ne possède pas de mémoire du passé*. La sortie d'un circuit combinatoire est fonction uniquement de l'entrée, pas des entrées qu'il a pu avoir dans le passé. Par exemple, une GameBoy n'est pas un circuit combinatoire.

Exemples simples de circuit pas combinatoire : le bistable, figure 3.2.

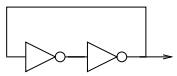

FIG. 3.2 – Le bistable

Pour trouver un circuit combinatoire correspondant à une fonction booléenne donnée, on passe par un stade intermédiaire : on détermine une expression algébrique de la fonction qui aura une traduction directe en matériel. On verra cela en 3.3.

3.2.5 Circuits combinatoires bien formés

Étant donné un ensemble de portes de bases qui sont toutes des circuits combinatoires, un CCBF est formé par

- une porte de base
- un fil
- la juxtaposition de deux CCBFs posés l'un à côté de l'autre
- un circuit obtenu en connectant les sorties d'un CCBF aux entrées d'un CCBF
- un circuit obtenu en connectant entre elles deux entrées d'un CCBF.

On peut montrer qu'on obtient un graphe sans cycle de portes de bases.

Ce qu'on s'interdit :

- faire des cycles, car cela permettrait des situations mal définies comme le circuit instable de la figure 3.3,
- connecter des sorties entre elles (que se passerait si une sortie est 1 et l'autre 0 ?)

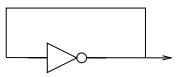

FIG. 3.3 – Le pas-stable

Plus tard on lèvera ces deux interdictions dans des circonstances bien maîtrisées. Avec des circuits à cycles, on pourra construire des oscillateurs (en mettant des retards dans notre pas-stable) et des mémoires (le bistable en est une, mais dans laquelle on ne peut pas entrer d'information à cause de la seconde interdiction, donc il faudra le bricoler un peu).

3.2.6 Surface et délai

Pour une fonction booléenne donnée, il y a une infinité dénombrable de manières de l'implémenter. Ce qui est intéressant, en architecture des ordinateurs, c'est de trouver la meilleure.

Exemple : construire un ET à quatre entrées à partir de NON-OU.

Meilleure selon quel critère ? La technologie amène avec elle ses métriques de qualité : taille, vitesse, consommation d'énergie, niveau d'émission électromagnétique...

Dans la notion la plus abstraite de circuit combinatoire, il n'y a pas plus de notion de temps que dans la notion d'expression booléenne. Par contre, toute réalisation physique d'un circuit combinatoire aura un certain *délai* de fonctionnement, noté τ, et défini comme le temps maximum entre un changement des entrées (à l'instant t) et la stabilisation des sorties dans l'état correspondant (à l'instant $t + \tau$). Les sorties du CC physique peuvent passer par des états dits transitoires pendant l'intervalle de temps $[t, t + \tau]$. C'est le cas du circuit de la figure 3.4. On apprendra à construire des circuits dans lesquels on garantit que ces états transitoires sont sans conséquence.

(petit dessin de chronogramme)

On peut définir (ou mesurer) le délai de chaque porte de base. Une fois ceci fait, on peut ramener le calcul du délai dans un CCBF à un problème de plus long chemin dans le graphe du

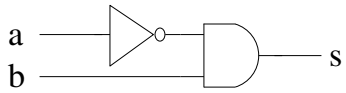

FIG. 3.4 – Un circuit pas compliqué

CCBF. Ce plus long chemin est appelé *chemin critique* (en Australien : *critical path*). On aura une tendance malheureuse à identifier le chemin critique et le délai du chemin critique.

Dans la vraie vie, le délai d'une porte de base peut être différent suivant l'entrée et la sortie considérée. Dans le cas général, pour une porte de base à n entrées et m sorties, on aura une matrice $n \times m$ de délais. Chaque entrée de la matrice est elle-même le max des délais pour toutes les configurations des autres entrées. De plus, le délai pourra être différent sur front montant et sur front descendant, on aura alors une matrice $2n \times m$. Naturellement le calcul du délai d'un CCBF devient un peu plus compliqué, mais cela reste parfaitement automatisable, et même vous sauriez faire.

Enfin, les authentiques gourous du circuit sauront retailler les transistors pour obtenir les délais qu'ils veulent. Le compromis sera : plus c'est rapide, plus c'est gros et plus cela consomme.

3.2.7 Quelques circuits utiles

Exercice : énumérons toutes les fonctions logiques à 2 entrées et une sortie, et donnons leur à chacune un petit nom.

On se donne une brique de base appelée *aiguillage*, qui en fonction d'une information binaire de *direction* (entrée du bas sur la figure), transmet l'information soit selon un chemin, soit selon l'autre.

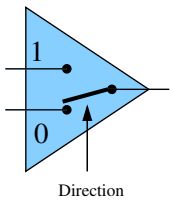

FIG. 3.5 – L'aiguillage

En assemblant $1 + 2 + 4 + ... + 2^{k-1} = 2^k - 1$ aiguillage, on sait construire une gare de triage complète. L'adresse est donnée bit à bit sur les fils de direction.

L'aiguillage, en technologie train électrique, marche dans les deux sens. En CMOS, on distinguera le multiplexeur (2^n en 1) et le démultiplexeur (1 vers 2^n).

Les gares de triage sont construites par une approche algorithmique. On va définir maintenant une approche automatique.

3.3 D'une fonction booléenne à un circuit combinatoire

3.3.1 Formes canoniques

Théorème de Shannon : soit f une fonction booléenne à n entrées, alors

$$f(x_1, x2, ...x_n) = x_1.f(1, x_2, ..., x_n) + \overline{x_1}.f(0, x_2, ...x_n)$$

Démonstration : considérer $x_1 = 0$ puis $x_1 = 1$.

Exercice : donner le théorème de Shannon dual de celui-ci. J'ai perfidement utilisé la notation que j'ai dit que je n'aimais pas.

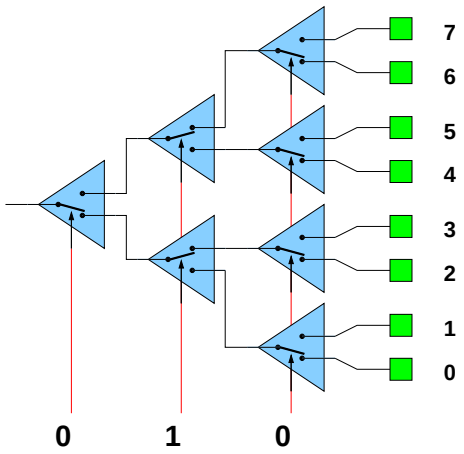

FIG. 3.6 – Une gare de triage permet de construire une mémoire adressable

On appelle un monôme canonique un ET de toutes les variables apparaissant chacune soit sous forme directe soit sous forme complémentée.

On appelle monal canonique le dual du précédent.

Les théorèmes de Shannon, appliqués récursivement, permettent d'obtenir automatiquement, à partir d'une fonction booléenne donnée par sa table de vérité, une expression booléenne de cette fonction sous forme dite canonique.

- Forme canonique disjonctive : OU (ou somme, mais je vous dis que je n'aime pas) de monomes
- Forme canonique conjonctive : ET de monals (monaux ?)[2]

Exemple : fonction majorité à trois entrées.

Quelle forme préférer ? Considérons la forme conjonctive. Naturellement, on simplifie tous les monômes correspondant à une valeur 0 de la fonction. Donc cette forme sera préférée pour une fonction ayant une majorité de 0 dans sa table de vérité. Inversement, s'il y a une majorité de 1, la forme disjonctive aura moins de monaux rescapés que la forme conjonctive de monômes.

Exemple : la fonction majorité à 3 entrées, qui renvoie 1 si au moins deux des trois entrées sont à 1 et 0 sinon.

3.3.2 Simplification des formes canoniques

Supposons pour anticiper qu'on veut se ramener à un nombre le plus petit possible de NON-ET et de NON-OU.

On peut appliquer des règles de réécriture.

3.3.3 Arbres de décision binaire

Construction de l'OBDD

Réductions (ROBDD)

- Si deux sous-arbres sont identiques, on les fusionne
- Si un nœud a deux fils identiques, on le court-circuite et on le supprime.

Problème : quel ordre des variables va donner les simplifications les plus avantageuses ? Tous les ans il y a des articles sur le sujet. On bricole des heuristiques.

Remarque : l'OBDD est, pour un ordre des variables donné, une représentation canonique. Elle est utilisée pour prouver l'équivalence de circuit (par exemple développé sur FPGA puis "recompilé" vers un ASIC).

[2] C'est sans doute pour cela qu'on préfère habituellement la disjonctive...

3.4 Circuits pour le calcul

3.4.1 Des petits cailloux aux bouliers

3.4.2 Addition et soustraction binaires

Pour construire un additionneur, la bonne technique est de faire le plus d'algorithmique possible, et de ne recourir à l'approche automatique que pour les petits blocs. D'ailleurs ce n'est pas spécifique à l'additionneur : c'est *toujours* la bonne technique.

On part de l'algo d'addition en décimal, qu'on écrit en langage de normalien :

Soit β un entier (la base, qui vaut 10 pour votre petite sœur). Soit la fonction *table d'addition* TA :

$$\begin{array}{rcl} \{0,1\} \times \{0..\beta-1\} \times \{0..\beta-1\} & \rightarrow & \{0..1\} \times \{0..\beta-1\} \\ (r,x,y) & \mapsto & (r',s) \\ & & \text{tq} \quad \beta r' + s \ = \ r + x + y \end{array}$$

- cette fonction prend une "retenue entrante" (0 ou 1) et deux chiffres, et retourne leur somme ;
- Cette somme est comprise entre 0 et $2\beta - 1$. Elle s'écrit donc en base β sur deux chiffres :
  - Le chiffre de poids faible de la somme est appelé *somme modulo la base* ou juste *somme*
  - Le chiffre de poids fort de la somme est appelé *retenue* et vaut 0 ou 1 quelle que soit la base.

L'algorithme d'addition de deux nombres de n chiffres est alors

1: $c_{-1} = 0$
2: **for** $i = 0$ to n **do**
3: $\quad (c_i, s_i) = \text{TA}(c_{i-1}, x_i, y_i)$
4: **end for**

On peut le dérouler de plusieurs manière en matériel : algo séquentiel (utilisation du temps) ou parallèle (utilisation de l'espace). Cela aussi est une idée générale. On y reviendra.

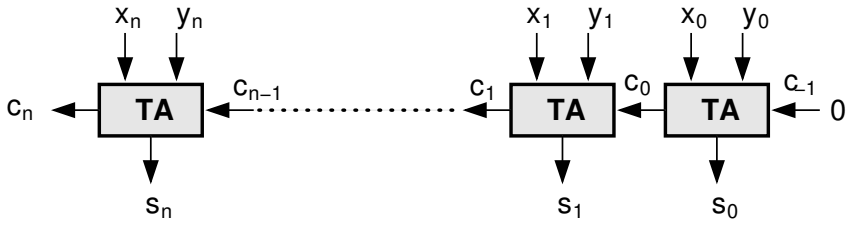

FIG. 3.7 – L'algo d'addition déroulé dans l'espace. TA c'est "table d'addition".

Pour construire l'additionneur binaire il suffit de considerer $\beta = 2$.

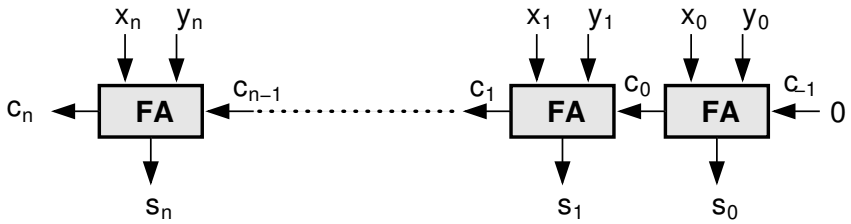

FIG. 3.8 – Additionneur *binaire* à propagation de retenue.

3.4.3 Multiplication binaire

En TD

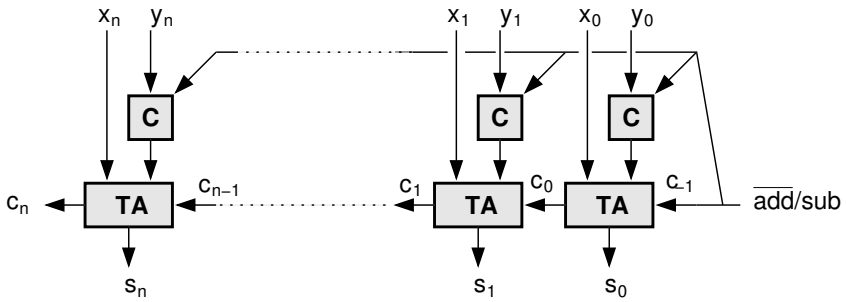

FIG. 3.9 – Additionneur en complément à 2.

3.4.4 Division binaire

En TD

3.5 Conclusion

On a une technique universelle pour transformer n'importe quelle fonction booléenne, donnée par sa table de vérité, en un circuit qui l'implémente. Cette technique s'adaptera facilement à une *bibliothèque de portes de base* pour une technologie donnée.

Toutefois, on se trouve confronté à des questions d'optimisation dont la complexité, dans le cas général, est exponentielle en le nombre d'entrées de la fonction.

Pour cette raison, faire de l'algorithmique intelligemment donne en général de meilleurs résultats que la technique universelle. C'est ce que l'on a vu pour construire des opérateurs de calcul.

Plus tard, on abstraira de même le comportement séquentiel d'une machine par un automate, on dérivera une technique universelle d'implémentation des automates, et on constatera à nouveau qu'un peu d'intelligence permet d'obtenir les meilleurs résultats de cette technique.

3.6 Annexe technologique contingente : les circuits CMOS

3.6.1 Transistors et processus de fabrication

3.6.2 Portes de base

Inverseur, non-et, non-ou, porte de transmission.
Illustration du fait que brancher deux sorties ensembles, c'est mal.

3.6.3 Vitesse, surface et consommation

Logical effort.

Chapitre 4

Memoriser

Enchainement des calculs, accumulateur. Même dans un calcul : retenue, ce qu'on retient.
Re-exemple de déroulements spatiaux et temporels de la somme de n nombres.

4.1 Organes de memorisation

4.1.1 Le registre ou mémoire ponctuelle

On part du bistable double-inverseur.
On ajoute deux portes de transmission. Ou deux portes "ou" (exo). Avantages comparés des deux dessins ?
Dessin d'un registre :
Mémorisation sur front (changement de valeur). Le triangle sur l"entrée d'horloge indique que ce signal porte une information temporelle, pas une donnée. Une information d'instant sera toujours donnée par un changement de valeur, qu'on appelle un front (soit montant, soit descendant).
Mettons 8 registres en parallèle, avec la même entrée d'horloge : on obtient un registre 8 bits. Par extension, les mémoires de travail dans le processeur sont appelées des registres. Exemple : le Pentium a 4 registres entiers et 8 registres flottants alors que l'Itanium a 128 de chaque.

4.1.2 Mémoires à accès séquentiel : piles et files

Pile ou LIFO : primitives
- empiler(info)
  ajoute un étage
- dépiler()
  enlève un étage : destruction de la donnée dans la pile
- test pile vide

File ou FIFO : primitives
- insérer(info)
- extraire() – avec destruction
- test file vide

On appele aussi la file : tampon ou *buffer*.
Remarque : les pile et files, conceptuellement, sont de capacité infinie... Si on les réalise, leur capacité sera sans doute finie, et il faudra ajouter le test "pile/file pleine".

4.1.3 Mémoires adressables

Les opérations de base qu'on peut faire avec une mémoire adressable sont

– lire la donnée à l'adresse a. Dans ce cas il faut donner l'information d'adresse à la mémoire, et elle répond par l'information contenue dans la cellule visée.
– écrire une donnée d à l'adresse a. Dans ce cas on doit donner à la mémoire les informations d'adresse et de donnée.

On appelle aussi la mémoire adressable RAM *(random access memory)*, parce que c'est plus court.

4.1.4 Mémoires adressables par le contenu

Analogie avec le dictionnaire. On ne veut pas retrouver l'information qu'on connait déjà (le mot) mais une autre information qui y est associée (définition). En général, on a dans une mémoire adressable par le contenu des couples (clé, valeur).

Primitives :
– rechercher(clé)
– insérer(clé, valeur)
arbitrage si clé est déjà présent
– supprimer(clé)

Il y a des variations suivant les stratégies de remplacement, et en général en fonction de la manière dont on les construit :

En TD

4.2 Construction des mémoires

4.2.1 Point mémoire

Le registre dessiné précédemment est une mémoire statique (l'information est stable) mais volatile (elle disparaît quand on coupe l'alimentation).

On sait faire des points mémoirqe *dynamiques* : en stockant l'info juste dans une capacité. Dessin. Avantage : c'est plus petit. Inconvénient : la lecture est destructrice. De plus la capacité se décharge lentement, il faut la regénérer périodiquement.

Question : laquelle est la plus rapide ?

Réponse si vous avez suivi les transistors de la dernière fois : le statique, qui peut fournir plus de courant (on utilise le condensateur le plus petit possible).

On sait aussi faire des mémoires non volatiles : par exemple la surface d'un disque dur, avec des particules qui peuvent être aimantées dans un sens ou dans l'autre. Dans le CDRW on a deux états stables d'un alliage (amorphe ou cristallin).

Mémoire flash : on isole la grille d'un transistor, et on y piège des électrons par claquage. Mémoire statique et non volatile, haute densité, lecture non destructrice (puisque le transistor reste bloqué ou passant). Inconvénient : écriture relativement lente (haute tension), seulement 100 000 cycles d'écriture.

4.2.2 Mémoire adressable

On sait construire une RAM avec des multiplexeurs, des démultiplexeurs, et des registres. En vrai, il y a plus économique : on verra en TD.

On n'a pas trouvé mieux qu'utiliser des RAM et des compteurs pour implémenter les piles et les files.

4.2.3 Disques

Piste, secteur.

Temps d'accès (qq ms), débit (qq Mo/s).

Les disques sont de petits objets sensibles plein de pièces mouvantes et fragiles. Solution : disques RAID *redundant array of inexpensive disks*. Pour vous faire utiliser des codages redondants.

4.2.4 Une loi de conservation des emmerdements

Une mémoire de grande capacité bon marché est lente, une mémoire rapide est limitée en taille, et chère.

Type	temps d'accès	capacité typique
Registre	0.1 ns	1 à 128 mots (de 1 à 8 octets)
Mémoire vive	10 - 100 ns	4 Goctets
Disque dur	10ms	100Goctets
Archivage	1mn	(illimité)

Remarques :

– Ce tableau est tout entier victime de la loi de Moore.
– Le coût de chaque étage est grosso modo équivalent. Le coût par octet est donc exponentiel par rapport à la vitesse d'accès.

On verra plus loin comment construire des mécanismes qui font croire au processeur que toute sa mémoire vive est aussi rapide que les registres (mémoire cache) et aussi grande que son disque (mémoire d'échange). Mais c'est de l'optimisation, pour construire un ordinateur simple (disons une Gameboy) on n'en a pas besoin. Et cela nécessite d'abord d'avoir construit un système d'exploitation, on n'en est pas encore là.

Chapitre 5

Transmettre

5.1 Medium

Fil electrique, fréquence radio, fréquence lumineuse, rail avec des billes.

Notion de canal.

Notion de multiplexage : on envoie plusieurs canaux logiques sur un canal physique. Multiplexage temporel, multiplexage en fréquence.

Notion de débit (quantité d'information par unité de temps) et de bande passante d'un canal (quantité d'information pouvant passer en même temps dans le canal.

5.2 Liaison point a point

5.2.1 Série ou parallèle

Utilisation du temps, ou utilisation de l'espace. En général on combine intelligemment les deux : sur une liaison parallèle on envoie tout de même les données en série !

Exemples : RS232, I2C, USB pour le pur série.

5.2.2 Protocoles

Si on n'a qu'un seul canal binaire pour transmettre des données entre un émetteur et un récepteur, tout ce qu'on sait faire c'est faire passer ce canal de 1 à 0 puis de 0 à 1. Si on enlève l'information temporelle, on n'observe que 1010101010101010101...

Une communication suppose donc une *synchronisation* (partage du temps en serbo-croate antique).

Une solution est qu'émetteur et récepteur aient chacun une horloge suffisamment précise, et se soient mis d'accord à l'avance sur les instants auxquels telle et telle donnée sont transmises. Ceci s'appelle un protocole de communication.

Une solution plus simple est d'avoir deux canaux binaires (au moins) entre émetteur et récepteur. L'un pourra faire passer le tic-tac d'une horloge, par exemple. Avec un seul canal mais au moins ternaire, c'est aussi possible. Mais dans tous les cas il faudra un protocole qui définit comment les données sont emballées, comment commence un paquet de donnée, etc.

Maître et esclave

Notion de maître (celui qui donne les ordres) et d'esclave (celui qui obéit). Le maître peut tourner, mais à un instant donné il vaut mieux qu'il n'y ait qu'un seul maître.

Protocoles synchrones

Utilisation du temps.

Exemple 1 : envoi d'une donnée de maitre à esclave, au moyen de deux canaux. Le maître positionne la donnée sur l'un, puis positionne l'autre canal, que nous appelerons "donnée prête" et qui valait 0, à 1 pendant 1 pataseconde, et puis de nouveau à zéro. Il sait que l'esclave lui obéira et aura lu la donnée au bout d'une pataseconde : c'est son esclave.

Exemple 2 : réception d'une donné par le maître. Le maître positionne un signal "demande de donnée". Au bout de 1 pataseconde, il lit la donnée sur l'autre canal. Il sait que l'esclave aura obéi dans ce laps de temps.

Exemple 3 : deux fils dont un est un tic tac, envoi d'octets commençant par 10 (pour marquer le début d'un octet) puis 8 bits, puis parité pour la détection d'erreur.

Exemple 4 : Manchester encoding (ethernet) : 0 est codé par 0 pendant une pataseconde puis 1 pendant une pataseconde, 1 codé par 1 puis 0. Ainsi l'horloge est incluse.

Ce type de protocole ne marche pas avec les circuits Normaliens. D'une part ils sont toujours en retard quand ils sont esclaves, d'autre part ils veulent tous être maître en même temps. C'est pourquoi on a mis au point des protocoles asynchrones.

Protocoles asynchrones

Protocoles qui ne font pas d'hypothèse sur les durées de transmission des signaux.

Archétype : protocole *handshake*. Il faut trois canaux : un de données (D), un de demande (R), un d'acquittement (A) *(acknowledge)* – les initiales viennent du breton.

Au repos, tout le monde est à 0. Voici comment se passe une émission pilotée par l'émetteur :

– émetteur : "Voici une donnée" (positionne D, puis lève R, et le laisse à 1 jusqu'à nouvel ordre).
– récepteur : voit R levé, range la donnée ou il faut, puis dit "Bien reçu Chef" (lève A, et le laisse levé jusqu'à nouvel ordre).
– émetteur : voit A se lever, dit "Brave petit" (baisse R, puis attend la baisse de A avant de recommencer).
– récepteur : voit R se baisser, dit "ce fut un plaisir Chef" (baisse A et attend un nouvel ordre).

Attention, ici l'émetteur a l'initiative de la transmission, mais n'est pas maître de tout. Par exemple, il ne peut pas baisser R tant que le récepteur ne lui autorise pas en levant A.

On peut faire un handshake à l'initiative du récepteur : R signifie à présent "envoie une donnée SVP".

Au repos tout le monde est à zéro.

– Récepteur lève R.
– Émetteur positionne la donnée, puis lève A.
– Récepteur traite ou range la donnée, puis baisse R.
– Émetteur baisse A.
– Récepteur voit A descendre, et sait qu'il peut demander une nouvelle donnée.

Il y a quand même une hypothèse temporelle derrière ce protocole, c'est que quand on écrit "émetteur positionne la donnée puis lève A", cet ordre temporel sera respecté à l'arrivée au récepteur. Si le système physique respecte cette hypothèse, on peut d'ailleurs aussi bien envoyer les données par paquets en parallèle, ce qui réduit le coût de l'asynchronisme. C'est comme cela que le processeur parle avec la mémoire dans votre PC (la mémoire peut avoir des tas de bonnes raisons de ne pas répondre tout de suite quand le processeur lui demande une donnée, on verra lesquelles).

Voyons maintenant un protocole vraiment insensible au délai. L'idée est d'utiliser un code dit double-rail pour la donnée : un bit est transmis sur deux canaux, avec la signification suivante : 1 est codé par 01, 0 par 10, le repose est codé par 00. Ainsi mettre une donnée sur le double-rail change obligatoirement au moins un bit, ce qui est détecté par le récepteur, et tient donc lieu soit de R (dans le protocole piloté par l'émetteur) soit d'A (dans le protocole piloté par le récepteur).

Loi de conservation des emmerdements : il faut dans ce cas vraiment deux canaux pour transmettre un seul bit.

Tout se complique si on suppose des erreurs possibles sur la ligne. Solution : encore plus de protocole. Vous avez déjà vu ce qu'il faut mettre dedans.

5.3 Bus trois états

porte trois etats. Arbitrage.

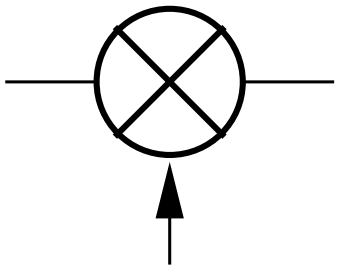

FIG. 5.1 – Porte de transmission, ou porte "trois états"

Exemples :
- L'intérieur des circuits mémoires, pour lier la donnée sélectionnée par le décodeur d'adresse à la sortie : on sait qu'une seule donnée passera sur ce fil.
- Par extension, la construction d'une grosse mémoire à partir de petites.
- Le canal de données qui relie le processeur à la mémoire dans la machine de von Neumann. Il faut que les données puissent aller dans un sens ou dans l'autre, mais on n'a pas besoin qu'elles aillent dans les deux sens en même temps.

5.4 Réseaux en graphes

En supposant réglée la question de transmettre des données sur un canal, on veut à présent relier entre eux n circuits (ou ordinateurs) par des canaux. Pour simplifier, disons que tous ces canaux sont bidirectionnels et identiques (même débit etc). Le réseau a alors une structure de graphe, les canaux étant le arêtes et les circuits étant les nœuds.

5.4.1 Les topologies et leurs métriques

Considérons deux topologies possibles de réseaux pour relier n ordinateurs : l'anneau, représenté figure 5.2, et le tore bidimensionnel, représenté figure 5.4 qui est une extension de la grille bidimensionnelle (représentée figure 5.3) dans laquelle tous les nœuds ont le même nombre de canaux.

On peut définir sur ces deux exemples les notions suivantes :

Degré Le degré d'un nœud est le nombre d'arêtes reliées à ce nœud. Par extension, le degré d'un graphe est le degré maximum des nœuds du graphe. Par exemple le degré est de 2 pour l'anneau, et de 4 pour la grille et le tore. A priori, plus le degré sera élevé, plus le nœud sera cher. Plus le degré est petit mieux c'est.

Diamètre Le diamètre d'un graphe est le maximum de la longueur du plus court chemin entre deux nœuds. Dans nos réseaux à $n = 16$ nœuds, le diamètre de l'anneau est 8 , le diamètre de la grille est 6, le diamètre du tore est 4. Le diamètre mesure la distance maximum à parcourir pour une information dans le réseau. Plus il est petit mieux c'est.

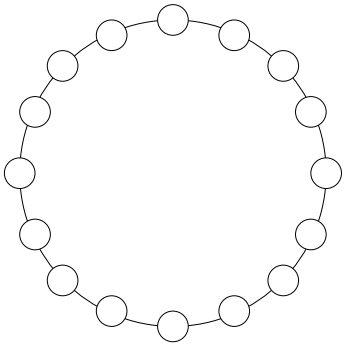

FIG. 5.2 – Graphe anneau

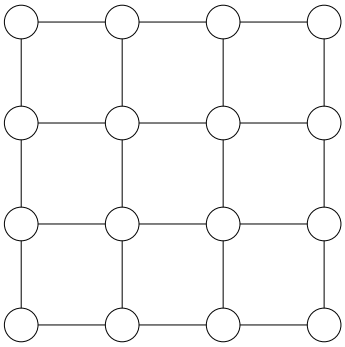

FIG. 5.3 – Graphe grille 2D

Bissection La bissection est une mesure de la quantité d'information qui peut passer à travers le réseau à un instant donné : Plus elle est élevée, mieux c'est. Techniquement, elle est définie comme le plus petit nombre de fils qu'il faut couper pour transformer le graphe en deux sous-graphes disjoints.

Tore et anneau ont tous deux des degrés constants, mais le tore a une bissection et un diamètre en $\sqrt{n}$, ce qui semble mieux que l'anneau (bissection de 2 et diamètre en $n/2$. On peut définir un tore tridimensionnel et plus (et au fait l'anneau est un tore unidimensionnel). Mais d'autres topologies offrent des compromis encore plus intéressants :

L'hypercube de degré d a 2^d nœuds, un diamètre de d, un bissection de 2^{d-1}. Son seul défaut est un degré qui peut devenir relativement élevé.

C'est pourquoi on a inventé le Cube Connectant des Cercles ou CCC, représenté figure 5.6 en dimension 3. On place des anneaux de taille d à chaque sommet d'un hypercube de degré d, et chaque nœud d'un anneau possède en plus un canal selon une des dimensions de l'hypercube. Le degré est 3 quelle que soit la dimension, et le diamètre est passé de d à $2d$: c'est toujours logarithmique en le nombre de nœuds.

5.4.2 Routage statique et routage dynamique

Commutation de ligne ou commutation de paquets.

Le avantage de la communication de paquets, c'est que la question du routage peut devenir décentralisée.

Commutation de paquets : store and forward, wormhole (flit).

Problèmes : famines, deadlock (interblocage). On les évite par des protocoles.

Exemples de preuve que pas d'interblocage et pas de famine :

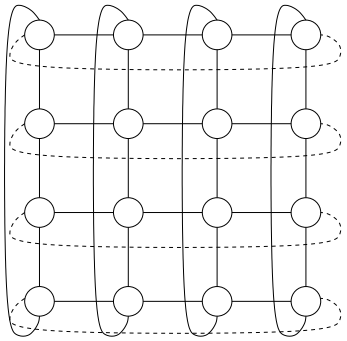

FIG. 5.4 – Graphe tore 2D

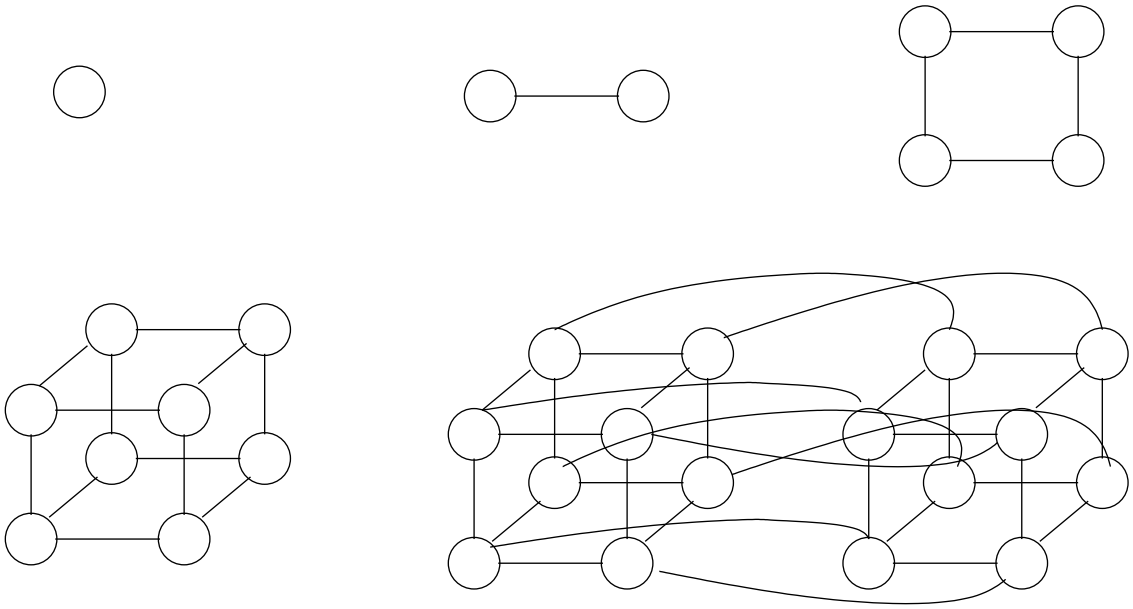

FIG. 5.5 – Graphes hypercubes

– anneau (protocole Token Ring),
– routage 2D Manhattan,
– routage hypercube.

5.4.3 Types de communication : point à point, diffusion, multicast

5.5 Exemples

5.5.1 Le téléphone à Papa

Commutation de ligne, protocole très centralisé (“central téléphonique”).

5.5.2 L’internet

Couche physique : ethernet.

Architecture en graphe sans vraiment de structure. Hiérarchie à deux niveaux : *local area network* ou LAN, *wide area network* ou WAN.

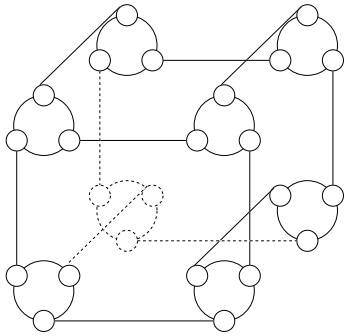

FIG. 5.6 – Graphe *cube-connected cycles*

Couches logiques : commutation de paquets, routage par store and forward, protocole décentralisé.

5.5.3 FPGAs

Grille 2D, commutation de ligne par *crossbar*, routage statique.

5.5.4 Machines parallèles

Vous verrez avec Eddy Caron ou Yves Robert, si la fin du pétrole ne les cloue pas à tout jamais dans leurs campagnes.