Architecture Technique et Matériel Informatique

D.Moreaux

17 septembre 2023

Introduction

De nos jour, les ordinateurs sont omniprésents, parfois dans des endroits les plus inattendus tels que les voitures ou l'électroménager.

Qu'il s'agisse d'un supercomputer qui occupe une pièce entière ou d'un *embedded system* qui n'occupe qu'un volume très réduit dans un appareil électroménager, les principes qui régissent son fonctionnement sont les mêmes.

Si la puissance des machines a fortement progressé ces dernières années, les principes de bases existent déjà depuis des dizaines d'années : algèbre binaire, mémoire, processeur, ...

Table des matières

Chapitre 1

Historique

Les débuts de la programmation remontent au XIIIe siècle avec les cylindres pointés qui permettaient de faire jouer une musique par un carillon.

Ces machines à musique ont évolué au fil du temps, tout d'abord en diminuant leur taille, ensuite avec l'introduction des livres de notes (série de cartons perforés attachés les uns aux autres en accordéon) et des rouleurs de bande de papier perforées.

A côté de cela, en 1642, Blaise Pascal crée la première machine à additionner, la Pascaline. Le système de roues contenant les chiffres et de reports se trouve toujours de nos jours dans les compteurs d'eau, de gaz et d'électricité.

En 1768-1774, Jaquet-Droz crée ses automates dont l'écrivain, le plus aboutit, utilise une roue sur laquelle on place des plots dont la hauteur indique la lettre à tracer. Le message écrit sera donc très aisément modifiable, chaque *donnée* en entrée (une plot correspondant à une lettre) menant à une séquence de tracé différente.

En 1728, Falcon met au point un métier à tisser capable de réaliser des motifs en fonction de cartes perforées. Le modèle sera amélioré au début XIXe par Jacquard qui en lancera une production massive.

En 1890, Hollerith mettra au point une machine pour aider au recensement américain. Un mécanisme permet de lire une carte perforée et, pour chaque perforation, augmentera un compteur (réalisé selon le mécanisme de Pascal). Il fondera une entreprise appelée Tabulating Machine Company, renommée en International Tabulating Machine et puis, des années plus tard, en International Business Machine (IBM).

Avec l'électricité, on verra apparaître d'abord des machines à relais, coûteuses, bruyantes, lentes et très consommatrices en énergie. Le tube sous vide permettra des machines bien plus puissantes, dont le Colossus utilisé durant la seconde guerre mondiale pour casser les cryptages Allemands.

En 1947, les laboratoires Bell vont créer le premier transistor à semi-

conducteur. Cela ouvre la porte vers l'électronique moderne. En 1958, le premier circuit intégré sera créé, en 1971, le premier microprocesseur, le 4004 d'Intel verra le jour.

Chapitre 2

Bases d'électronique

2.1 Electricité

Un atome possède un certain nombre de couches d'électrons, chaque couche étant capable d'accepter un nombre fixe d'électrons.

Pour qu'un courant électrique circule, il faut que des électrons se déplacent. Ceux qui se trouvent sur les couches internes sont bloqués, seuls les électrons extérieurs peuvent donc se déplacer.

Dans certains matériaux, les forces qui retiennent ces électrons sont très forts et les électrons de la couche externe sont captifs. Ce genre de matériaux ne permettent pas la circulation d'un courant électrique et sont appelés isolants

Dans d'autres matériaux, les forces qui retiennent les électrons extérieurs sont faible. Cela permet aux électrons de se déplacer d'atome en atome et donc à un courant électrique de circuler.

Quelques rares matériaux se trouvent à la limite entre les deux comportements, on parlera de semi-conducteurs.

Quand un atome perd son électron, un "trou" apparaît. Ce trou pourra être rempli par un électron d'un atome voisin. Dans une structure avec peu de trous comparés aux atomes ayant tous leurs électrons, cela donnera l'impression que c'est le trou qui se déplace (et pas un électron).

Pour quitter un atome, un électron aura besoin d'une certaine énergie. Cette énergie peut venir d'un champ électrique, de la chaleur, de la lumière, ... La mécanique quantique indique qu'un électron ne peut se trouver que sur certains niveaux d'énergie précis. Pour passer d'une situation "au repos" à une situation "excitée", l'électron aura donc toujours besoin de la même quantité d'énergie. De même, lorsqu'un électron repasse dans une situation "au repos", une certaine quantité d'énergie sera émise. Ce phénomène est à

l'origine des diodes LED ou des LASER.

On parlera de courant électrique (mesuré en ampères) pour indiquer le nombre d'électrons qui circulent par unité de temps... Plus le courant est élevé, plus les électrons se déplacent vite. Quand on prend une pile (ou un accumulateur rechargeable), la quantité d'électrons qui peuvent être fournis du côté négatif de la pile est limitée (capacité de la pile). Plus le courant est fort, plus ces électrons seront pompés rapidement et plus vite la batterie se déchargera.

Pour que les électrons se déplacent, il faut qu'ils subissent une force. L'origine de cette force est liée à la différence de potentiel électrique que l'on appellera aussi tension électrique (mesurée en volts). Plus la tension sera élevée, plus les électrons seront accélérés (ce qui aura pour effet d'augmenter le courant).

Différents aspects du circuit d'un autre côté vont ralentir les électrons et limiter le courant. Le courant final sera donc un compromis entre la tension et les différents aspects qui limitent le courant. On parlera de résistance (ou d'impédance) pour quantifier cette limitation de courant.

Quand des électrons passent dans une résistance, ils sont ralentis. L'énergie cinétique (liée au mouvement) des électrons sera transformée en chaleur. Cette chaleur peut, si elle est trop élevée, détruire des composants électroniques.

2.2 Semi-conducteurs

Les semi-conducteurs sont souvent réalisés à base de Silicium (dans certains cas, on utilise des structures plus complexes, par exemple au niveau des LED).

Le silicium a une couche externe qui possède 4 électrons et peut en accepter un maximum de 8. Dans un bloc de silicium, chacun des 4 électrons est "partagé" avec un atome voisin ce qui amène à une structure très stable et très peu conductrice (quasi isolante)

Par contre, en injectant des atomes qui possèdent 3 ou 5 électrons (appelés impuretés), on fera apparaître des électrons libres ou des trous qui pourront aisément se déplacer dans la structure rendant le silicium conducteur. Le nombre d'impuretés nécessaires est très faible.

En cas de surplus d'électrons, on parlera d'une zone "N" (négative), en cas de présence de trous, on parlera d'une zone "P" (positive). Tant les zones N que P sont de bons conducteurs.

Si la zone P est reliée à l'électrode positive et la zone N à l'électrode négative, le champ électrique repoussera les trous vers la zone N et attirera les

électrons vers la zone P, cela permettra au courant de passer sans problèmes

Par contre, si on relie la zone N à l'électrode positive et la zone P à l'électrode négative, les trous seront attirés loin de la zone N et les électrons loin de la zone P. Au contact des deux zones, le nombre d'electrons et de trous va diminuer ce qui va créer une zone isolante au milieu et le courant ne saura pas passer.

Une construction spéciale avec un canal N interrompu par un petit canal P formera un transistor NPN. Si un courant circule entre l'électrode centrale et une des deux électrodes externes, cela permettra de garder le canal N conducteur. Sinon, la zone isolante se développe et le courant est interrompu. Cela donne au transistor une caractéristique d'interrupteur contrôlé par l'électricité. (on peut également créer des transistors PNP). La géométrie du transistor favorisera un sens de circulation du courant.

Une autre sorte de transistor utilise un canal N (ou P) et une électrode séparée par un isolant. En mettant un potentiel négatif sur l'électrode de commande, les électrons seront repoussés hors de la zone du canal en dessous de l'électrode et vont créer une zone isolante. Un potentiel positif permettra de ramener les électrons dans le canal et au courant de passer. On parlera de MOSFET (Metal Oxyde Silicium Field Effect Transistor). Cette technologie dégage peu de chaleur et est facile à réaliser.

2.3 Composants

2.3.1 La Résistance

La résistance permet de limiter le courant qui passe dans le circuit.

En électronique numérique (celle des ordinateurs), elle est aussi utilisée pour forcer le niveau électrique sur une borne en absence de signal (par exemple, si un interrupteur permet de connecter la borne à la tension positive (+), on utilisera une résistance entre cette borne et la tension négative (-) pour forcer la borne au (-) quand on n'active pas l'interrupteur.

Pour les signaux haute fréquence, elle sera également utilisée pour adapter l'impédance en fin de câble.

2.3.2 Le Condensateur

Un condensateur agit un peu comme une petite batterie rechargeable. Il permet de filtrer les parasites, de maintenir une réserve d'électrons pour faire face à un pic de courant,...

Dans certains cas, il sera utilisé pour créer une base de temps (le temps nécessaire pour le charger ou le décharger) ou pour réaliser des filtres qui ne laissent passer que certaines fréquences

2.3.3 Diode

La diode sera utilisée généralement pour transformer un courant alternatif en courant continu, pour protéger un circuit contre un élément monté à l'envers,...

Certaines diodes spéciales sont conçues pour émettre de la lumière lors des transitions entre les états normaux et les états excités des atomes. C'est à la base des diodes LED ou diodes LASER. De même, on utilisera parfois la lumière pour faire passer des électrons de l'état repos à l'état excité (ce qui leur permettra de se déplacer) afin de faire des diodes sensibles à la lumière.

La zone isolante qui apparaît quand une diode est mise en inverse se comporte comme un condensateur et sera parfois utilisée comme condensateur variable.

Certaines diodes sont prévues pour que le courant arrive quand même à passer quand la diode est en inverse, à partir d'une certaine tension. On parle de diode Zéner et ce genre de diodes est utilisé pour faire des régulateurs de tension.

2.3.4 Transistor

Le transistor est à la base des portes logiques qui composent les ordinateurs. C'est le principal composant utilisé dans les microprocesseurs qui contiennent des millions (voire des milliards) de transistors.

Afin de diminuer la chaleur dissipée, on a tendance à diminuer de plus en plus la tension utilisée. On est donc amené également à utiliser des transistors pour amplifier les signaux à des niveaux utilisables par les autres composants.

Chapitre 3

Binaire et algèbre de Boole

3.1 Signaux logiques

Dans un ordinateur, on utilisera l'électronique numérique. En d'autres termes, on travaillera en *tout ou rien* au niveau des signaux, un signal étant soit coupé, soit actif, 0 ou 1, 0 Volt ou +5 Volt, Faux ou Vrai.

Un tel signal ne permet de fournir que deux états et donc, ne permet pas de véhiculer des valeurs supérieures à 1. Pour ce faire, on devra combiner plusieurs signaux. Si on prend n signaux, on disposera de 2^n combinaisons possibles. La valeur n s'appelle la *largeur de bus*.

On trouve ainsi des bus de différentes tailles selon les systèmes.

Bits	Nom	processeurs
4	Quarter	Saturn (HP28, HP48,...), 4004
8	Octet	Z80, 6500, PIC16F (data), ATMega, 8008, 8080
14		PIC16F (programme)
16	Mot	8086, 80286, TMS9900, ...
32	Mot	80486, 68000, MIPS, Sparc, ARM,...
64	Mot (Mot long)	Intel Core 2, ARM, Sparc64, Alpha

TABLE 3.1 – Taille de bus

A noter que sur certains systèmes (PIC16F, VAX,...) on trouve des bus de taille différente selon qu'il s'agit de véhiculer les instructions du programme ou les données.

Les signaux rencontrés dans un ordinateurs peuvent servir à véhiculer une adresse, une donnée ou des signaux de contrôle. Par exemple, une mémoire morte aura besoin d'un bus pour l'adresse, d'un second pour la donnée présente à l'adresse en question et d'un signal pour indiquer que l'adresse est correcte et qu'il faut envoyer le contenu de la mémoire sur la sortie.

On représentera les valeurs des signaux par 0 ou 1 (0V et +Vcc [1]). Si le composant n'impose aucune valeur sur le signal, on utilisera la notation **Z** qui signifiera *haute impédance* et correspond à un fil *déconnecté*.

On utilisera parfois la lettre **X** pour indiquer un signal que l'on ne prend pas en compte (*don't care*).

3.2 Algèbre de Boole

Plusieurs opérations peuvent être effectuées sur les valeurs binaires. Les principales opérations sont la négation, le *et* et le *ou*. On les défini comme suit

3.2.1 NON

La négation du signal A sera notée $\overline{A}$ et on la lira *non-A*.
Le résultat sera l'inverse de la valeur de départ.

A	$\overline{A}$
0	1
1	0

TABLE 3.2 – Table NON-A

3.2.2 ET

L'opérateur **ET** entre les signaux A et B se notera $A \wedge B$ et on lira A *et* B.

En algèbre de Boole, on notera le **ET** par une multiplication : $A.B$
Le résultat ne sera 1 que si les deux signaux sont tous les deux à 1.

A	B	$A \wedge B$ ou $(A.B)$
0	0	0
0	1	0
1	0	0
1	1	1

TABLE 3.3 – Table A ET B

1. la tension d'alimentation

3.2.3 OU

L'opérateur **OU** entre les signaux A et B se notera $A \vee B$ et on lira A *ou* B.

En algèbre de Boole, on notera le **OU** par une addition : $A + B$

Le résultat ne sera 1 que si au moins un des deux signaux est à 1.

A	B	$A \vee B$ ou $(A + B)$
0	0	0
0	1	1
1	0	1
1	1	1

TABLE 3.4 – Table A OU B

3.2.4 Formule quelconque

Une formule logique quelconque, quel que soit le nombre de variables, peut toujours être réalisée à l'aide des trois opérateurs cités ci-dessus. On obtiendra une expression sous la forme d'une somme (OU) de produits (ET) de signaux et de leur négation (NON)

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

Le OU doit valoir 1 pour les termes ou la fonction vaut 1 et 0 ailleurs. Cela veut dire que les lignes pour lesquelles le terme sera présent sont celles où la sortie vaut 1, à savoir Le 2eme $(0, 0, 1)$, le 3eme $(0, 1, 0)$ et le 7eme $(1, 1, 0)$.

Ces termes ne doivent être à 1 que pour les entrées correspondant aux signaux qui les représentent et valoir 0 dans les autres cas. Cela se réalise avec un ET où les 3 signaux sont soit présents, soit leur inverse (NON) est présent.

Par exemple, le terme $(0, 1, 0)$ se représentera par $\overline{A}.B.\overline{C}$, *overlineA* valant 1 si A vaut 0,...

L'expression logique qui correspondra à la fonction sera donc

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

Vu que toute fonction peut être représentée en somme de produit de cette façon, on trouve des composants spécialisés (PAL, GAL) dans lesquels on programme de telles sommes de produits et qui peuvent donc ainsi servir à réaliser toutes les fonctions logiques.

3.2.5 OU-Exclusif

Une fonction particulière que l'on trouve parfois est le **OU-Exclusif** ou **XOR**. On le représente en algèbre de Boole $A \oplus B$.

Cette fonction est vraie si UNE des deux valeurs A et B vaut 1 et l'autre vaut 0. On dit aussi que le XOR est vrai si l'une des valeurs est vraie ou l'autre est vraie mais pas les deux.

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

TABLE 3.5 – Table A OU-Exclusif B

Le OU-Exclusif peut être exprimé de différentes manières telles que

$$\begin{aligned} A \oplus B &= \overline{A}B + A\overline{B} \\ &= (A + B).(\overline{A.B}) \\ &= (A + B).(\overline{A} + \overline{B}) \end{aligned}$$

3.3 Simplification

Une série de règles permettent de modifier les expressions logiques. Ces modifications permettront de diminuer le nombre de termes qui apparaissent et donc, de rendre le test de la fonction logique plus simple, moins onéreux [2].

2. Cela peut être un coût en composants devant être utilisé ou en opérations nécessaires pour effectuer les différents tests

— Distributivité/factorisation

$$\begin{aligned} A.(B+C) &= A.B + A.C \\ A + (B.C) &= (A+B).(A+C) \end{aligned}$$

— variable et son inverse

$$\begin{aligned} A.\overline{A} &= 0 \\ A + \overline{A} &= 1 \end{aligned}$$

— variable et une constante

$$\begin{aligned} A.0 &= 0 \\ A.1 &= A \\ A + 0 &= A \\ A + 1 &= 1 \end{aligned}$$

— Lois de De Morgan

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

Ces différentes règles permettent de simplifier des équations logiques.

Généralement, on utilisera la factorisation et les lois de De Morgan pour faire apparaître des situations où on effectue une opération entre une variable et son inverse, ce qui permettra ensuite d'utiliser les règles pour une opération entre une variable et une constante.

On peut également, pour des fonctions particulièrement compliquées, recréer la table de vérité et créer une somme de produits qu'il restera à simplifier.

3.4 Représentation d'un nombre en binaire

3.5 Indication de la base

La base d'un nombre est le nombre de chiffres qui sont utilisés pour représenter les nombres. Les nombres que l'on utilise habituellement utilisent des chiffres de 0 à 9 qui sont au nombre de 10. On parle donc de *base 10* ou de *décimal*.

Lorsque l'on travaille en binaire, on utilise les deux chiffres 0 et 1, on parle donc de *base 2*.

Pour spécifier la base, on peut utiliser plusieurs méthodes :

— une lettre minuscule après le nombre. On utilisera donc des nombres de la forme 100101b, 42d,...
— le nombre correspondant à la base utilisée, exprimé en base 10, en indice. On aura donc 100101_2, 42_10,...
— un préfixe lié à l'un ou l'autre langage informatique (par exemple $\&B100101$ en référence au Basic)

Afin d'éviter toute ambiguïté, la notation indicée sera utilisée dans ces notes de cours.

3.5.1 nombre de bits nécessaires

Le nombre de chiffres binaires (aussi appelés bits[3] nécessaires pour représenter un nombre $n \in \mathbb{N}$ en binaire sera

$$\log_2(n) = \frac{\log(n)}{\log(2)}$$

En effet, un bit permet de représenter deux nombres (0 et 1), deux bits d'en représenter quatre (00, 01, 10, 11), trois d'en représenter huit, n d'en représenter 2^n.

3.5.2 binaire vers décimal

Pour convertir un nombre binaire en décimal, il suffit d'assigner aux bits les puissances de deux successives en commençant par le dernier bit (appelé *bit de poids faible*) et 2^0.

Ensuite, on conserve les puissances de 2 qui correspondent à des bits à 1 et on additionne les valeurs ainsi conservées.

1	1	0	1	1	0	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
128	64		16	8		2	

$$128 + 64 + 16 + 8 + 2 + = 218$$

TABLE 3.6 – Conversion Binaire ⇒ Décimal

3. de Binary digITS

3.5.3 décimal vers binaire : puissances de 2

Pour convertir un nombre décimal vers le binaire, on peut procéder par comparaisons successives avec des puissances de 2 décroissantes.

Si la valeur est supérieure ou égale à la puissance de 2, on aura un bit à 1 et on soustrait la puissance de 2 à la valeur, sinon, on aura un bit à 0.

Nombre	comparaison	résultat	**bit**	nouvelle valeur
218	≥ 128	Oui	**1**	$218 - 128 = 90$
90	≥ 64	Oui	**1**	$94 - 64 = 26$
26	≥ 32	Non	**0**	
26	≥ 16	Oui	**1**	$26 - 16 = 10$
10	≥ 8	Oui	**1**	$10 - 8 = 2$
2	≥ 4	Non	**0**	
2	≥ 2	Oui	**1**	$2 - 2 = 0$
0	≥ 1	Non	**0**	

Résultat : 11011010

TABLE 3.7 – Conversion Décimal ⇒ Binaire, puissances de 2

Cette méthode est souvent la plus rapide. En effet, pour les plus petites valeurs, on fini rapidement par connaître de mémoire les associations entre valeur décimal et nombre binaire ce qui permet de court-circuiter les dernières étapes.

3.5.4 décimal vers binaires : divisions

Une autre méthode pour convertir un nombre décimal en nombre binaire est de faire des divisions par 2 successives. On notera les restes successifs et ils formeront le nombre binaire en commençant par le bit de poids faible.

3.6 Quine Mc Cluskey

3.6.1 Liste des minterms

On commence par énumérer les cas où la fonction à simplifier vaut 1 et les cas où la valeur n'a aucune importance (*don't care*).

On notera ces deux listes en les séparant. Généralement, on représentera la combinaison des entrées sous la forme d'un nombre décimal.

Valeur	quotient	**reste**
218	109	**0**
109	54	**1**
54	27	**0**
27	13	**1**
13	6	**1**
6	3	**0**
3	1	**1**
1	0	**1**

Résultat : 11011010

TABLE 3.8 – Conversion Décimal ⇒ Binaire, divisions

3.6.2 Première colonne

On va déterminer la première colonne en prenant les minterms qui correspondent aux valeurs de 1 **ET** aux don't care.

On commencera par les minterms qui contiennent 0 bits à 1, puis on créera un second groupe avec les minterms qui contiennent 1 bit à 1, un troisième avec les minterms qui en contiennent 2 et ainsi de suite.

3.6.3 Deuxième colonne

On cherchera ensuite à regrouper des termes de la première colonne. L'idée est de regrouper des minterm qui ne diffèrent que d'un seul bit.

Le second de ces deux minterms contiendra donc exactement un bit de plus à 1 que le premier. On peut donc se contenter de comparer les minterms de deux groupes successifs (si un groupe est vide, il est donc important de le signaler).

On pourra par exemple regrouper 0 et 2 (0000 et 0010). Le bit qui change est celui qui correspond à une valeur de 2 (00X0). On notera cela sous la forme

$$(0, 2)_2$$

Lorsque l'on fait apparaître une valeur dans la seconde colonne, on *marque* la valeur de la première colonne (soit on la barre, soit on met une marque à côté) pour indiquer qu'elle est reprise par un *terme* d'une des colonnes suivantes.

Pour repérer plus facilement les paires d'éléments, on cherchera des éléments de la première colonne dont la différence est une puissance de 2.

A	B	C	D	valeur	minterm
0	0	0	0	1	0
0	0	0	1	0	1
0	0	1	0	1	2
0	0	1	1	0	3
0	1	0	0	1	4
0	1	0	1	0	5
0	1	1	0	1	6
0	1	1	1	1	7
1	0	0	0	1	8
1	0	0	1	1	9
1	0	1	0	1	10
1	0	1	1	0	11
1	1	0	0	X	12
1	1	0	1	X	13
1	1	1	0	X	14
1	1	1	1	X	15

Minterms : 0, 2, 4, 6, 7, 8, 9, 10
Don't Care : 12, 13, 14, 15

TABLE 3.9 – Quine Mc Cluskey : étape 1

Lorsque une telle paire de valeurs est trouvée, on s'assurera qu'il n'y a bien qu'un seul bit qui change. La valeur qui suit les parenthèses sera cette différence.

Les éléments de la seconde colonne sont composés de paires de valeurs avec un seul bit qui change entre les deux valeurs.

A chaque fois que l'on envisage la paire de groupes de la première colonne suivante, on commencera un nouveau groupe en deuxième colonne. On aura donc un groupe de moins que dans la première colonne

3.6.4 Troisième colonne et suivantes

Pour la troisième colonne, on cherchera les groupes de 4 valeurs où seuls deux bits changent.

Pour la quatrième colonne, on cherchera des groupes de 8 valeurs où 3 bits changent et ainsi de suite avec des groupes de 2^{c-1} valeurs où $c-1$ bits changent.

Pour ce faire, on comparera des paires de valeurs de deux groupes qui se

0
2
4
8
6
9
10
12
7
13
14
15

TABLE 3.10 – Quine Mc-Clusky : étape 2

0 $\surd$	$(0,2)_2$
2 $\surd$	$(0,4)_4$
4 $\surd$	$(0,8)_8$
8 $\surd$	$(2,6)_4$
6 $\surd$	$(2,10)_8$
9 $\surd$	$(4,6)_2$
10 $\surd$	$(4,12)_8$
12 $\surd$	$(8,9)_1$
7 $\surd$	$(8,10)_2$
13 $\surd$	$(8,12)_4$
14 $\surd$	$(6,7)_1$
15 $\surd$	$(6,14)_8$
	$(9,13)_4$
	$(10,14)_4$
	$(12,14)_2$
	$(12,13)_1$
	$(7,15)_8$
	$(13,15)_2$
	$(14,15)_1$

TABLE 3.11 – Quine Mc-Clusky : étape 3

$0\ \surd$	$(0,2)_2\surd$	$(0,2,4,6)_{2,4}$
$2\ \surd$	$(0,4)_4\surd$	$(0,2,8,10)_{2,8}$
$4\ \surd$	$(0,8)_8\surd$	$(0,4,8,12)_{4,8}$
$8\ \surd$	$(2,6)_4\surd$	$(2,6,10,14)_{4,8}$
$6\ \surd$	$(2,10)_8\surd$	$(4,6,12,14)_{2,8}$
$9\ \surd$	$(4,6)_2\surd$	$(8,10,12,14)_{2,4}$
$10\ \surd$	$(4,12)_8\surd$	$(8,9,12,13)_{1,4}$
$12\ \surd$	$(8,9)_1\surd$	$(6,7,14,15)_{1,8}$
$7\ \surd$	$(8,10)_2\surd$	$(12,13,14,15)_{1,2}$
$13\ \surd$	$(8,12)_4\surd$	
$14\ \surd$	$(6,7)_1\surd$	
$15\ \surd$	$(6,14)_8\surd$	
	$(9,13)_4\surd$	
	$(10,14)_4\surd$	
	$(12,14)_2\surd$	
	$(12,13)_1\surd$	
	$(7,15)_8\surd$	
	$(13,15)_2\surd$	
	$(14,15)_1\surd$	

TABLE 3.12 – Quine Mc-Clusky : étape 4

$0\ \surd$	$(0,2)_2\surd$	$(0,2,4,6)_{2,4}\surd$	$(0,2,4,6,8,10,12,14)_{2,4,8}$
$2\ \surd$	$(0,4)_4\surd$	$(0,2,8,10)_{2,8}\surd$	
$4\ \surd$	$(0,8)_8\surd$	$(0,4,8,12)_{4,8}\surd$	
$8\ \surd$	$(2,6)_4\surd$	$(2,6,10,14)_{4,8}\surd$	
$6\ \surd$	$(2,10)_8\surd$	$(4,6,12,14)_{2,8}\surd$	
$9\ \surd$	$(4,6)_2\surd$	$(8,10,12,14)_{2,4}\surd$	
$10\ \surd$	$(4,12)_8\surd$	$(8,9,12,13)_{1,4}$	
$12\ \surd$	$(8,9)_1\surd$	$(6,7,14,15)_{1,8}$	
$7\ \surd$	$(8,10)_2\surd$	$(12,13,14,15)_{1,2}$	
$13\ \surd$	$(8,12)_4\surd$		
$14\ \surd$	$(6,7)_1\surd$		
$15\ \surd$	$(6,14)_8\surd$		
	$(9,13)_4\surd$		
	$(10,14)_4\surd$		
	$(12,14)_2\surd$		
	$(12,13)_1\surd$		
	$(7,15)_8\surd$		
	$(13,15)_2\surd$		
	$(14,15)_1\surd$		

TABLE 3.13 – Quine Mc-Clusky : étape 5

suivent, en ne prenant en compte que des paires suivie du même nombre en indice.

Ainsi, on pourra regrouper $(0, 2)_2$ et $(4, 6)_2$ que l'on notera $(0, 2, 4, 6)_{2,4}$. On remarquera que le chiffre ajouter correspondra à la différence entre les premiers chiffres de chaque paire ou la différence des seconds.

De nouveau, on s'assurera qu'il s'agit bien d'un bit de plus activé et on marquera les termes qui sont retrouvés.

A noter qu'à un terme de la troisième colonne correspondent 4 termes de la deuxième.

Pour la 4eme colonne, on procédera de la même manière mais en s'assurant que les deux chiffres mis en indice correspondent et on obtiendra des groupes de 8 valeurs etc...

Pour la quatrième colonne, chaque regroupement reprendra 6 termes de la troisième colonne.

3.6.5 Récupération des regroupements

Une fois que plus aucun regroupement ne peut être fait, on récupère la liste des termes qui n'ont pas été marqués.

Ici, on aura les termes $(8, 9, 12, 13)_{1,4}$, $(6, 7, 14, 15)_{1,8}$, $(12, 13, 14, 15)_{1,2}$ et $(0, 2, 4, 6, 8, 10, 12, 14)_{2,4,8}$.

La solution simplifiée contiendra au maximum ces termes. A ce niveau, les regroupements sont effectués, il faut déterminer quels sont les termes nécessaires.

On créera dès lors une table avec, dans une direction, les termes ci-dessus et dans l'autre, les minterms qui correspondent à une valeur de 1 (on ne reprend pas les don't care).

Au croisement des deux, on mettra une marque si le minterm apparaît dans le terme.

On cherchera d'abord des lignes où un seul terme est coché. On validera le terme en question et marquera les minterms repris dans ce terme.

Une fois tous les minterms marqués, les termes choisis seront les termes nécessaires.

3.6.6 Cas d'indécision

Dans certains cas, plusieurs termes permettront d'obtenir une valeur. On choisira

— Celui qui couvre le plus de minterms valides (qui permettra de *valider* d'autres minterm).

	$(8,9,12,13)_{1,4}$	$(6,7,14,15)_{1,8}$	$(12,13,14,15)_{1,2}$	$(0,2,4,6,8,10,12,14)_{2,4,8}$
0 √				X
2 √				X
4 √				X
6 √		X		X
7 √		X		
8 √	X			X
9 √	X			
10 √				X
	X	X		X

TABLE 3.14 – Quine-Mc Clusky : sélection finale

— Celui qui couvre le plus de minterm (et donc a l'équation la plus courte)
— Celui qui sera le plus facile à tester

3.6.7 Transformation en expression logique

Pour obtenir l'expression logique finale, on prend les termes sélectionnés à l'étape précédente, on les transforme en *produits* (AND entre différents termes) et on fait la *somme* de ces produits (OR entre les produits).

Les chiffres indiqués après la liste des termes couverts indiquent quelles variables ne doivent pas être présentes. Par exemple, ici, A équivalait à la première variable ($2^3 = 8$), B à la seconde ($2^2 = 4$), C à la troisième ($2^1 = 2$) et D à la dernière ($2^0 = 1$).

Pour décider quelles variables doivent être prises *telles quelles* et quelles variables doivent être *inversées* (NOT), on regardera la forme binaire d'un des termes (par exemple le premier). Les bits à 1 correspondent aux variables à prendre inchangées et les bits à 0 aux variables à inverser

Dans notre exemple, on a les termes $(8,9,12,13)_{1,4}$, $(6,7,14,15)_{1,8}$ et $(0,2,4,6,8,10,12,14)_{2,4,8}$. On les transformera comme suit

Terme	variables à garder	valeur	produit
$(8,9,12,13)_{1,4}$	AC	1000	$A\overline{C}$
$(6,7,14,15)_{1,8}$	BC	0110	BC
$(0,2,4,6,8,10,12,14)_{2,4,8}$	D	0000	$\overline{D}$

TABLE 3.15 – Quine Mc Clusky : produits

On aura donc l'expression logique $A\overline{C} + BC + \overline{D}$.

3.7 Arithmétique binaire

Si les signaux de contrôle sont souvent représentés sur 1 bit (Ecriture (WR), Lecture (RD), Sélection de composant (CS),...), les autres données qui circulent dans un ordinateurs sont généralement sous la forme de bus de 8, 16, 32 ou 64 bits.

Les opérations qui seront effectuées le seront donc sur des valeurs de plusieurs bits. Pour des raisons de simplicité, ce cours traitera des données de 8 bits mais les méthodes sont valables quel que soit la taille du bus envisagé [4]

3.7.1 Opérations logiques

Les opérations logiques vues précédemment (AND, OR, XOR et NOT) peuvent également être appliquées sur des données 8-bits. Pour ce faire, on effectuera l'opération bit à bit, séparément pour les bits de poids le plus faible (que l'on appellera *bit 0*), jusqu'au bit de poids le plus élevé (que l'on appellera *bit 7* dans le cas de données 8-bits)

Pour le NOT, l'opération s'effectue sur chaque bit individuellement.

	1	0	0	1	1	0	1	1
AND	1	1	0	1	0	1	0	1
	1	0	0	1	0	0	0	1

TABLE 3.16 – Opération logique AND entre deux données 8-bits

3.7.2 Addition

Pour effectuer l'addition, on procédera par une méthode semblable au calcul écrit utilisé pour les nombres décimaux [5].

Lorsque la valeur dépassera 1_2, on reportera les bits supplémentaire dans la colonne de gauche, tout comme on reporte les dizaines dans une addition en base 10.

Lorsque l'on désire additionner plus de deux nombres, la somme sur un bit peut dépasser 11_2. Dans ce cas, on place les différents chiffres du report

4. A noter que même s'ils existent depuis de nombreuses années, les bus 8 bits sont toujours employés de nos jours, entre autres dans les microcontrôleurs.

5. Les nombres composés de chiffres de 0 à 9 que l'on utilise habituellement

	^{1}		^{1}	^{1}	^{1}		^{1}	^{1}		(report)
		1	0	0	1	1	0	1	1	
+		1	0	1	0	1	0	1	1	
	$1^{(1)}$	$0^{(10)}$	$1^{(1)}$	$0^{(10)}$	$0^{(10)}$	$0^{(10)}$	$1^{(1)}$	$1^{(11)}$	$0^{(10)}$	

TABLE 3.17 – Addition binaire

à raison de un par colonne. Une somme de 4 bits à 1 donnera 100_2 et on placera[6] un report de 0 dans la colonne directement à gauche et un report de 1 deux colonnes à gauche

3.7.3 Soustraction

De la même manière, on utilisera une méthode semblable au calcul écrit utilisé pour les nombres décimaux. Lorsque la première valeur est plus petite que ce qu'il faut lui soustraire, on lui ajoutera 10_2 et on placera un report de 1 dans la colonne de gauche.

					^{10}	^{10}	^{10}	
	1	0	1	1	1	0	0	1
-	1	0	0	0	1	0	1	1
				^{1}	^{1}	^{1}		
	0	0	1	0	1	1	1	0

TABLE 3.18 – Soustraction binaire

Attention à la notation, on peut être tenté d'écrire la valeur *ajoutée* sous la forme du chiffre 2, ce qui n'aurait pas de sens vu que l'on travaille en binaire.

3.7.4 Multiplication

Tout comme les additions et soustractions, on utilisera la méthode du calcul écrit.

Ici, la première étape qui consiste à écrire les termes composés d'une des valeur multipliée par chacun des chiffres de la seconde est nettement simplifiée. En effet, cela se limite aux cas *multiplier par 1* qui consiste en recopier la première valeur et *multiplier par 0* qui aboutirait à une série de 0 (que l'on ne prendra pas la peine de recopier)

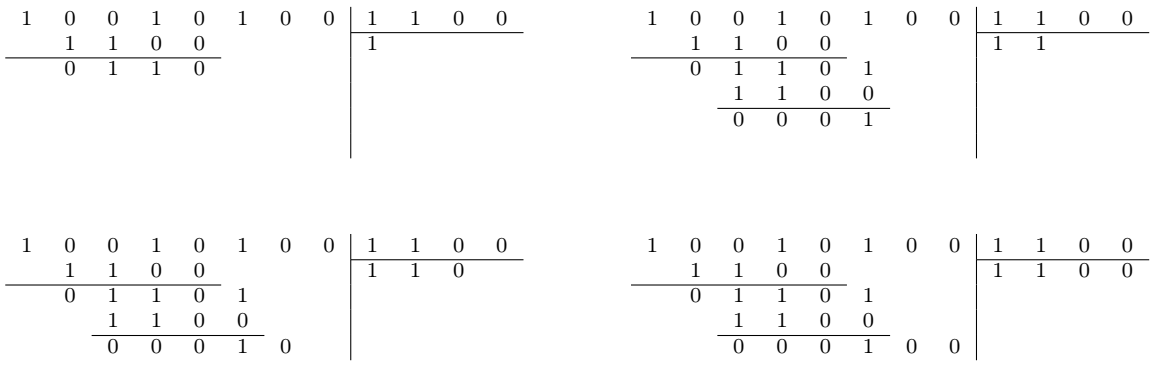
6. Pour éviter de surcharger l'addition, on ne notera pas les reports qui valent 0

					1	1	0	1
×					1	0	1	1
					1	1	0	1
				1	1	0	1	.
+		1	1	0	1	.	.	.
	1	0	0	0	1	1	1	1

TABLE 3.19 – Multiplication binaire

3.7.5 Division

De nouveau, la division se fera comme en calcul écrit décimal. Une fois les chiffres du numérateur épuisés, on aura le quotient et le reste. On peut soit s'arrêter à ce niveau (division entière), soit placer une virgule au niveau du quotient et ajouter des 0 au niveau de la division pour obtenir une valeur avec une partie fractionnaire.



Résultat : Quotient 1100, reste : 100

TABLE 3.20 – Division binaire

3.7.6 Complément à 2

Si on effectue un NOT bit à bit sur la valeur, on obtient une valeur que l'on appelle *complément* ou encore *complément à 1.*

Si au complément d'une valeur, on ajoute 1, on obtient une valeur appelée *complément à 2.*

Si on tronque les nombres à un certain nombre de bits, le complément à 2 de 0 sera égal à 0 (l'addition provoque un report en dehors du nombre de bits demandé)

valeur	00101101	00000000
NOT	11010010	11111111
+1	11010011	*1*00000000
complément à 2	11010011	00000000

Les valeurs sont limitées à 8 bits

TABLE 3.21 – Complément à 2

3.7.7 Soustraction par complément à 2

Si on considère une valeur de n bits a et qu'on lui ajoute son complément $\overline{a}$, la valeur obtenue sera composée de n bits à 1. En effet, à chaque bit 0 on ajoutera 1 et vice versa.

Si on ajoute 1 à cette somme, on arrive à une valeur de $n+1$ chiffres qui est composée de 1 suivi de n 0. Si on limite le résultat aux n bits de départ, la valeur sera donc de 0 [7]

On pourra donc dire que $b = \overline{a} + 1$ est [8] l'opposé de a vu que $a + b = 0$ et donc $a = -b$.

Cela permet donc d'effectuer des soustractions en ajoutant le complément à deux de la valeur à soustraire, à condition que toutes les valeurs soient exprimées sur le même nombre de bits. Si une valeur possède moins de bits que nécessaire, on la fera précéder d'autant de 0 que nécessaire pour atteindre le nombre voulu.

3.8 Hexadécimal

En informatique, on utilise souvent des valeurs en hexadécimal ou base 16. Ces valeurs sont en effet proches du binaire (un chiffre hexa correspond à 4 bits) tout en permettant des nombres d'une taille bien plus raisonnable.

Pour signaler une valeur en hexadécimal, on utilisera souvent le préfixe 0x (0x1234), le suffixe h (1234h) ou la base en indice 1234_{16}. On peut également

7. Avec un report à 1 qui est conservé par les processeurs pour des opérations suivantes

8. lorsque l'on utilise des lettres minuscules, on désigne des valeurs numériques et non des valeurs booléens et donc, + désignera l'addition et non le OU logique

rencontrer la notation issue du langage BASIC &H1234 même si elle est plus rare.

L'hexadécimal nécessite 16 symboles pour représenter ses chiffres. On utilisera à cette fin les chiffres de 0 à 9 et les lettres de A à F [9] qui correspondront aux *chiffres* de 10 à 15.

3.8.1 binaire - hexadécimal

Pour convertir une valeur binaire en hexadécimal, il suffit de regrouper les bits par groupes de 4 en commençant par le bit de poids faible et de remplacer chaque groupe par le chiffre hexa qui lui correspond (0000 donne 0, 1001 donne 9, 1010 donne 10 et s'écrit donc A, 1111 donne 15 et d'écrit F)

$$\underbrace{10}_{2}\,\underbrace{1100}_{C}\,\underbrace{1000}_{8}\,\underbrace{1010}_{A}$$

Pour faire la conversion dans l'autre sens, on remplacera chaque caractère hexadécimal par la valeur binaire qui lui correspond.

Binaire	hexa
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7

Binaire	hexa
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

TABLE 3.22 – Binaire - Hexa

3.8.2 décimal - hexadécimal

Pour convertir une valeur hexadécimale $d_n d_{n-1} \dots d_1 d_0$ en valeur décimale, on utilisera les puissances de 16.

$$16^n d_n + 16^{n-1} d_{n-1} + \cdots + 16^1 d_1 + 16^0 d_0$$

9. On peut également utiliser les lettres en minuscules

Pour faire la conversion inverse, on pourra travailler par divisions successives par 16, les restes des divisions donnant les chiffres hexadécimaux du dernier au premier. Il est également possible de diviser par des puissances de 16 décroissantes pour obtenir un chiffre hexadécimal puis de retirer ce chiffre multiplié par la même puissance de 16 avant de répéter l'opération avec la puissance suivante

16^0	1
16^1	16
16^2	256
16^3	4 096
16^4	65 536
16^5	1 048 576
16^6	16 777 216
16^7	268 435 456

TABLE 3.23 – Puissances de 16

On pourra par exemple convertir 0xDEAD en décimal. On aura

$$\begin{aligned} 0xDEAD &= 16^3 * 13 + 16^2 * 14 + 16^1 * 10 + 16^0 * 13 \\ &= 4096 * 13 + 256 * 14 + 16 * 10 + 13 \\ &= 57005 \end{aligned}$$

De la même manière, on pourra convertir 14285 en hexadécimal

$$\begin{aligned} \frac{14285}{16} &= 892 \quad \text{reste } 13 \Rightarrow \text{D} \\ \frac{892}{16} &= 55 \quad \text{reste } 12 \Rightarrow \text{C} \\ \frac{55}{16} &= 3 \quad \text{reste } 7 \Rightarrow 7 \\ \frac{3}{16} &= 0 \quad \text{reste } 3 \Rightarrow 3 \\ 14285 &= 0x37CD \end{aligned}$$

3.9 Stockage des données

Un ordinateur peut stocker différents types de données : texte, nombres entiers, nombres fractionnaires,. . .Dans tout les cas, ces données sont stockées sous forme de séquences de bits.

Généralement [10], on utilise comme unité de base l'octet (8 bits). Chaque information sera représentée par un nombre de bits multiple de 8.

3.9.1 Nombres entiers

On peut distinguer deux sortes de nombres entiers : les nombres entiers positifs ($n \in \mathbb{N}$) que l'on appellera nombre entiers non signés et les nombres entiers ($n \in \mathbb{Z}$) qui peuvent être positifs ou négatifs et que l'on appelle nombres entiers signés.

Nombres non signés binaires

Un nombre signé sera représenté sous la forme de la séquence de bits binaire correspondante.

On classera les nombres selon le nombre de bits utilisés pour les encoder : 8 bits, 16 bits, 32 bits, 64 bits. Si la valeur binaire ne possède pas le nombre de bits nécessaires, on la complète par des bits à 0 à gauche de la valeur.

Par exemple, la valeur 42 s'écrit en binaire 101010. Cette valeur s'exprime sur 6 bits. Si on veut la représenter sous la forme d'un entier 8 bits, on écrira **00**101010, sous la forme d'un entier 16 bits, on écrira **0000000000**101010,. . .Les bits de remplissage sont parfois appelés *padding*.

La valeur maximale qui peut être stockée dépendra du nombre de bits. Sur n bits, on pourra sauver une valeur entre 0 et $2^n - 1$.

Nombre de bits	valeur maximale
8	255
16	65535
32	4 294 967 295
64	plus de 18 446 740 000 000 000 000

TABLE 3.24 – Valeur maximale entiers non signés

Nombres signés binaires

Une première méthode consisterait à utiliser un bit pour représenter le signe et les $n - 1$ bits restant pour représenter une valeur de 0 à $2^{n-1} - 1$.

Cette méthode a cependant le défaut que, si on désire additionner deux nombres de signes différents, il sera nécessaire de tester les signes et les valeurs et d'effectuer une soustraction,.

10. à l'exception des processeurs 4 bits

Au lieu de cela, on peut représenter les nombres négatifs par le complément à 2 de leur opposé. En effet, ajouter le complément à 2 d'une valeur correspond à la soustraire, ce qui correspond bien à la définition d'un nombre négatif.

Ainsi, $4_{10} = 00000100_2$ et $-4_{10} = 11111011_2 + 1 = 11111100_2$. Les compléments à 2 des valeurs de 00000001_2 à 01111111_2 évoluent de 11111111_2 à 10000001_2.

Le bit de poids fort de ces nombres négatifs est à 1 alors qu'il était à 0 pour les valeurs positives. Ce bit de poids fort pourra donc être testé pour connaître le signe.

La valeur 10000000_2 est son propre complément à 2 (tout comme la valeur 0). Comme son bit de poids fort est à 1, on dira qu'il s'agit d'une valeur négative et donc, que cela vaudra -128.

Les nombres signés sur n bits pourront donc aller de -2^{n-1} à $2^{n-1} - 1$.

Nombre de bits	valeur minimale	valeur maximale
8	-128	127
16	-32768	32767
32	-2 145 483 648	2 145 483 647
64	$\approx -9223370000000000000$	$\approx 9223370000000000000$

TABLE 3.25 – Valeur maximale entiers signés

Valeurs en BCD

Certains ordinateurs utiliseront des valeurs où chaque chiffre de 0 à 9 sera encodé sur 4 bits (0000 à 1001). On appelle ce codage *Binary Coded Decimal*.

On utilisera donc 4 bits par chiffre de la valeur décimale. Cela sera principalement utilisé sur des processeurs capables de réaliser directement des additions et soustractions en BCD [11]

Le principal intérêt du codage BCD est qu'il permet d'éviter les erreurs d'arrondis lors des conversions décimal $\Leftrightarrow$ binaire des nombres fractionnaires.

11. C'est par exemple le cas du processeur Saturn qui équipe certaines calculatrices HP. Ce processeur dispose d'instructions SETDEC et SETHEX permettant de passer en mode BCD ou en mode binaire. A noter qu'il s'agit d'un processeur 4-bits (même si en interne il est capable de traiter des données jusqu'à 64 bits).

3.9.2 Caractères

Que sont les caractères

On appelle caractère tout élément de base qui puisse faire partie d'un texte, d'une formule, ...

Les caractères peuvent ainsi être

— Une lettre (majuscule ou minuscule)
— Une lettre grecque
— Une lettre cyrillique (russe)
— Une lettre arabe (à noter qu'il existe 4 formes selon la place dans le mot)
— Un hiragana ou un katakana (alphabets japonais)
— Un kanji (idéogramme apparaissant en chinois et en japonais)
— Un chiffre
— Un signe de ponctuation
— Un symbole mathématique
— Un symbole représentant un signe du zodiaque, une planète, ...
— ...

Cartes perforées et EBCDIC

Au début, les données étaient encodées sur des cartes perforées. Ces dernières prévoyaient 12 positions numérotées de 0 à 9 et A, B.

Pour représenter un chiffre, on perçait la case correspondant au chiffre en question. Pour un espace, on ne perçait rien. Pour une lettre (majuscule), on perçait un des chiffre et une ou deux des cases A et B.

Cette méthode a été étendue en mémoire en stockant la valeur du chiffre dans les 4 bits de poids faible et en utilisant les 2 bits de poids fort pour représenter l'état des lignes A et B. Ce codage a ensuite été étendu pour inclure les lettres minuscules, la ponctuation et les symboles mathématiques.

ASCII

Les principaux défauts de l'EBCDIC sont que ce codage étale les caractères sur la totalité de l'intervalle de 0 à 255 et que les codages de deux lettres consécutives n'étaient pas toujours des valeurs consécutives (on passait par exemple de 11001001_2 pour I à 11010001_2 pour J).

Le codage ASCII résolvait ces problèmes. Les chiffres étaient placés à partir de la position 48, les majuscules de la position 65, les minuscules à partir de la position 97, les 32 premières positions réservées à des codes de contrôle.

Ce codage n'utilisait que 7 bits, ce qui permettait d'utiliser le 8ème bit en contrôle de parité dans les communications série.

ISO-8859-1 (Latin-1)

Le code ASCII de base ne permettait pas les caractères accentués. Par contre, l'intervalle de 128 à 255 était inutilisé.

On a donc vu une série de codages qui plaçait dans ces 128 positions d'autres caractères : les caractères accentués, des caractères graphiques permettant de réaliser des encadrements simples et doubles, quelques symboles mathématiques supplémentaires,...

L'organisme ISO a proposé une série de codages standards pour ces valeurs, en fonction du pays. L'ISO-8859-1 (suivi par l'ISO-8859-15 qui intégrait le caractère €)

JIS, SJIS,...

Les langues asiatiques telles que le Chinois ou le Japonais doivent représenter un très grand nombre de caractères.

La langue Japonaise possède deux alphabets (hiragana et katakana). L'encodage JIS était semblable à l'ASCII pour les valeurs de 0 à 127 et ajoutait entre 161 et 223 les caractères de ces deux alphabets. Le JIS ne permettait pas de représenter les idéogrammes (Kanji), en nombre plus limités en japonais qu'en chinois.

Le SJIS a utilisé les caractères entre 128 et 160 et entre 224 et 255 pour représenter les dits kanjis. Lorsqu'un de ces caractères était rencontrés, on le combinait avec le caractère qui suivait pour former une combinaison qui précisait le kanji à utiliser. Plus de 7000 kanjis sont ainsi disponibles (auxquels on peut encore ajouter un bon nombre de caractères tels que les lettres grecques, cyrilliques, accentuées, d'icones, de flèches,...)

L'encodage EUC utilisait d'une part des caractères ASCII entre 0 et 127 et d'autre part des codes de deux octets [12] tous deux pris entre 161 et 247. Cet encodage était utilisé aussi bien pour le Japonais que pour le Chinois et le Coréen.

Unicode

Afin de pouvoir représenter tous les caractères possibles en assignant à chacun d'entre eux un code unique, le système Unicode a été créé.

12. il y avait également des séquences pour des codes sur 3 octets

Les valeurs sont exprimées sous la forme de 17 *plans* numérotés de 0 à 16, chaque plan proposant jusqu'à 65536 caractères (16 bits).

Afin de rendre les conversions aisées, les 256 premiers caractères du plan 0 correspondent aux caractères ISO-8859-1.

Un caractère sera donc représenté par une valeur de minimum 21 bits (généralement, il sera représenté par une valeur de 32 bits).

Le plan 0 est le plan principal, dans lequel on trouve les principaux alphabets actuels.

Le plan 1 est utilisé principalement pour des langues anciennes (Linear B, Hiéroglyphes,...) et le plan 2 pour des ajoutes aux langues à idéogrammes (telles que le Chinois).

Les plans 3 à 13 sont pour le moment inutilisés, les plans 15 et 16 sont réservés aux *usages privés* et on peut y retrouver entre autres des alphabets venant de fictions (comme l'alphabet Klingon) [13]

UCS-2, UTF-8, UTF-16,...

Pour encoder les caractères Unicode, il peut être nécessaire de fournir jusqu'à 21 bits. Pour ce faire, on utilise des encodages qui travailleront sur des unités de 1 ou 2 octets.

UCS-2 UCS-2 utilise une valeur de 16 bits (ce qui couvre le plan 0). Cet encodage utilise systématiquement 16 bits par caractère et ne permet pas d'accéder aux autres plans. On utilise souvent la valeur 0xFEFF comme premier caractère pour permettre de détecter l'ordre dans lequel les octets sont lus (certains ordinateurs inversent les octets ce qui amènerait à une valeur invalide en unicode).

UTF-8 UTF-8 utilisera un nombre variable d'octets pour représenter le caractère. Si le premier caractère est entre **0**0000000 et **0**1111111 (0 et 127), il représente la valeur finale (1 octet) qui correspond au code ASCII.

Si le premier caractère est entre **110**00000 et **110**11111, la valeur finale sera composée de 2 octets, le premier contenant les 5 premiers bits et le second les 6 bits suivants pour former une valeur de 11 bits. Si le premier caractère est entre **1110**0000 et **1110**1111, on utilisera deux octets supplémentaires pour un total de 16 bits (ce qui permet de couvrir le plan 0).

On pourra ainsi aller jusqu'à 6 octets pour une valeur finale sur 31 bits.

Les octets supplémentaires sont de la forme **10**xxxxxx ce qui permet de les identifier facilement.

13. A noter qu'une zone est également réservée pour usages privés dans le plan 0

Nombre de bits	nombre d'octets	premier octet	valeur maximale
7	1	0xxxxxxx	0x7F
11	2	110xxxxx	0x07FF
16	3	1110xxxx	0xFFFF
21	4	11110xxx	0x1FFFFF
26	5	111110xx	0x3FFFFFF
31	6	1111110x	0x7FFFFFFF

TABLE 3.26 – UTF-8 : taille des caractères

On remarquera que l'Unicode ne demande pas plus de 21 bits et donc que les encodages sur 5 ou 6 caractères ne sont pas (encore ?) utilisés.

UTF-16 L'UTF-16 utilise deux octets comme unité de base. Si la valeur se trouve entre 0xD800 et 0xDBFF, elle indique qu'un second mot de 16 bits dont la valeur est entre 0xDC00 et 0xDFFF sera utilisé pour former une valeur de 20 bits à laquelle on ajoute 0x10000 (ce qui permet d'adresser les plans 1 à 16).

Les valeurs entre 0xD800 et 0xDFFF ne sont associées à aucun symbole dans Unicode (et sont donc invalides si elles apparaissent en UCS-2 ou en UTF-8)

UTF-32 L'UTF-32 utilise directement une valeur de 32 bits (4 octets) et ne nécessite donc aucun codage particulier.

3.9.3 Chaînes de caractères

Une série de caractères est aussi appelée *chaîne de caractères*. Elle se représentera en mémoire sous la forme de la succession des caractères qu'elle contient.

taille fixe

Dans certains cas, la chaîne sera représentée par un nombre de caractères fixes. On trouvera cela par exemple au niveau des bases de données.

Les caractères qui représentent la chaîne se trouveront au début de la zone de mémoire allouée et on complétera la chaîne pour atteindre le nombre de caractères voulus par des caractères espace.

taille précisée

Une seconde manière est de préciser la taille de la chaîne sous la forme d'un entier non signé suivi de la séquence de caractères représentant la chaîne en question.

Cette méthode se retrouve dans certains langages de programmation tels que le Pascal.

Caractère de fin

Une troisième méthode est de terminer la chaîne par un caractère spécial, normalement incorrect.

Le langage C utilise ainsi le caractère dont le code est 0 [14] (0x00, pas le caractère correspondant au chiffre 0) que l'on représente en C
0.

Pour déterminer la taille de la chaîne, il faudra donc compter les caractères jusqu'à ce que l'on atteigne le caractère de fin.

Liste de caractères

Certains langages représentent des données sous forme d'une liste [15]. La chaîne pourra être représentée comme une liste de caractères.

3.9.4 Nombres fractionnaires

Virgule fixe

Pour stocker les nombres en virgule fixe, on décide du nombre de chiffres présents après la virgule n. Tous les nombres sont ensuite multipliés par 10^n. Ainsi, 3.1416 sera représenté par la valeur 31416 en virgule fixe avec 4 chiffres après la virgule.

Pour les additions, soustractions et comparaisons, il suffira d'effectuer les opérations sur des nombres entiers. Pour les multiplications et divisions, il sera nécessaire d'ajuster la position de la virgule (par exemple, une multiplication de deux nombres avec 4 chiffres après la virgule mènera à un nombre avec 8 chiffres après la virgule, il faudra donc laisser tomber les 4 derniers chiffres)

14. On appelle parfois ce caractère le *null-byte*

15. Le LISP par exemple utilise des paires de valeurs, la valeur gauche étant la donnée et la valeur droite pointant sur la paire suivante (ou un objet liste vide pour la dernière paire)

Cette façon de stocker les nombres peut être combinée avec le stockage en BCD ou avec des nombres en binaire.

Virgule flottante

On peut représenter les nombres décimaux de la forme $m.10^e$ où m est la mantisse et e est l'exposant. Cette notation est souvent appelée *notation scientifique*.

Dans le cas de nombres binaires, on utilisera plutôt la relation $m.2^e$ où la mantisse sera un nombre fractionnaire binaire.

La norme IEEE754 prévoit que l'exposant ne soit pas sous la forme complément à 2 mais sous la forme $e + 2^n - 1$ où n est le nombre de bits de l'exposant. Pour un exposant sur 8 bits, cela permettra des valeurs de -127 à +128.

Pour la mantisse, on considérera que le bit avant la virgule vaudra 1 (ce qui permet de ne pas devoir le sauver). Cela pose quand même problème pour la valeur 0. A cet effet, lorsque l'exposant est de -127, on considérera que le bit avant la virgule est de 0 ce qui permettra, en plus du 0, de représenter des nombres plus proches de 0 que 2^{-127}.

De même, pour représenter ∞, on utilisera un exposant de +128 et une mantisse de 1 (tous les bits après la virgule étant à 0). Les autres valeurs serviront à indiquer la situation *Not A Number* (NaN) que l'on a par exemple lorsque l'on essaie de calculer $\sqrt{-1}$, $log0$,...

Les nombres simple précision seront donc représentés avec un bit de signe, 8 bits d'exposant et 23 bits de mantisse et permettront de représenter les nombres de $1,4012985.10^{-45}$ à $3,40282346.10^{38}$. Ce sont ces nombres qui seront utilisés pour le type *float* dans le langage C et ils demandent exactement 4 octets pour être stockés.

On défini également les nombre double précision avec un bit de signe, 11 bits d'exposant (de -1023 à 1024, les valeurs extrêmes ayant de nouveau une signification spéciale) et une mantisse de 52 bits

L'utilisation d'un exposant décalé permet de comparer deux nombres simplement en comparant les valeurs binaires représentées par les mêmes séquences de bits.

Chapitre 4

Le microprocesseur

4.1 ALU

L'ALU (Arithmetic and Logic Unit) est un circuit qui utilise de la logique purement combinatoire (simples fonctions binaires) pour effectuer un ensemble d'opérations : addition, soustraction, fonctions logiques bit à bit, décalage, ...

Pour l'addition, un circuit appelé *full adder* qui prend 3 bits en entrée et retourne la somme de ces derniers en sortie est utilisé. 2 des entrées sont dirigées vers les nombres à additionner, la troisième sert d'entrée pour le report du full-adder précédent (Carry In). La sortie de poids faible est utilisée comme bit de résultat, la sortie de poids fort comme report de sortie (Carry Out) connecté au Carry In du full adder suivant.

Pour la soustraction, il suffira de prendre le complément à 1 de la valeur à soustraire (porte NOT) et d'ajouter 1 à la somme, ce qui peut être fait en forçant le "Carry In" du full adder de poids faible à 1

Les fonctions logiques seront implémentées par les portes qui leur correspondent.

Pour le décalage, on utilisera un montage où la sortie b_i recevra soit la valeur b_i (pas de décalage), soit b_{i+1} ou b_{i-1} selon que l'on désire décaler vers la gauche ou vers la droite. Le bit sortant pourra être ignoré ou envoyé dans le Carry. Le bit entrant pourra valoir une valeur fixe (0 ou 1), le Carry ou le bit sortant.

Un dernier circuit se chargera de prendre le résultat et de tester s'il est nul. Un drapeau Z *zero* sera positionné à 1 si tel est le cas. Un drapeau C contiendra le Carry et on pourra avoir des drapeaux tels que S *sign* qui contiendrait le bit de poids fort ou P *parity* qui contiendrait 0 ou 1 selon que la valeur possède un nombre pair ou impair de bits à 1

L'ALU est donc un circuit qui possède deux bus d'entrée pour les deux nombres à manipuler (l'un des deux pouvant être ignoré), un accès en lecture et en écriture au status (drapeaux Z, C, S, P, ...) et un ensemble de lignes de contrôle chargées de selectionner l'opération effectuée.

Les processeurs modernes intègrent une unité chargée d'effectuer des multiplications ou des divisions au niveau de l'ALU (même si ce n'est pas systématique).

4.2 Banque de registres

Les processeurs possèdent une zone de mémoire chargée de retenir des valeurs numériques qui seront traitées par l'ALU ou serviront à l'accès à la mémoire. Chaque cellule mémoire sera appelée *registre*.

Ces mémoires sont de taille variable selon les processeurs, et peuvent contenir un seul registre ou un ensemble de registres. Certains registres peuvent être spécialisés pour accéder à la mémoire.

Les lignes de contrôle permettront d'écrire un résultat dans un des registres ou de transférer le contenu de un ou de deux registres simultanément vers l'ALU.

6502	Z80	68000	80386	ARM	PIC	Saturn
A	A BC DE HL	D0-D7	EAX EBX ECX EDX	R0-R12	W	A B C D
	A' BC' DE' HL'					R0 R1 R2 R3 R4
X Y	IX IY	A0-A7	EDI ESI EBP			D0 D1
S	SP		ESP	R13		
PC	PC	PC	EIP	R15	PC	PC
	F F'	SR	CR0-CR3	R14		P

TABLE 4.1 – Exemples de registres

Un registre souvent appelé PC (*Program Counter*) pointe sur l'instruction suivante à exécuter. On trouvera également souvent un registre qui servira de pointeur de pile dont le nom sera SP (*Stack Pointer*).

4.3 Accès à la mémoire

Afin de permettre au processeur de lire et d'écrire des données en mémoire, on connecte des buffers [1] qui permettent de stocker une adresse et une

1. mémoire dont le but est de mémoriser une valeur de manière temporaire entre deux mécanismes qui ne répondent pas aux même timing ou qui n'ont pas les mêmes caractéristiques électriques

donnée, dont un côté est relié à l'ALU et l'autre côté vers la mémoire de l'ordinateur.

Des signaux de contrôle permettront de transférer le buffer données de/vers l'ALU et les registres et de transférer dans le registre d'adresse une valeur provenant d'un registre ou de l'ALU.

D'autres signaux serviront à contrôler la mémoire extérieure (lecture ou écriture).

Pour lire une valeur de la mémoire, on commencera par charger le contenu du registre d'adresse, puis on lira dans le buffer de données la valeur en mémoire et enfin, on transférera le contenu du buffer de données vers un registre ou vers l'ALU.

Pour écrire, on commencera par charger les contenus des buffers d'adresse et de donnée puis on transférera le contenu des buffers dans la mémoire.

4.4 Séquenceur

Le coeur du processeur est un circuit qui sera chargé de générer les différents signaux de contrôle. Pour ce faire, il lira une instruction dans un registre qui lui est propre, *l'instruction register*, et l'utilisera pour décider des tâches à effectuer.

Souvent, les instructions sont agencées de façon à simplifier le décodage. Par exemple, on trouvera un ensemble d'instructions où certains bits servent à indiquer le registre de destination, le registre source ou l'opération à effectuer.

Si au début ce séquenceur était créé à partir d'une machine d'état et de fonctions booléennes, la tendance actuelle est à la *microprogrammation*.

Pour cela, on utilise des *instructions* de microcode qui sont en fait l'ensemble des signaux de contrôle auxquels on a ajouté une série de bits dont le but est de décider de l'instruction suivante à exécuter. Le processeur sera ainsi *programmé* et certains processeurs permettent de modifier le microcode (ces modifications n'étant pas nécessairement sauvées de façon durable sur le microprocesseur ce qui permet d'éviter une destruction de ce dernier s'il est mis à jour avec un microcode incorrect).

4.5 RISC ou CISC

Un processeur RISC est un processeur dont le nombre d'instructions est limité ainsi que la complexité de ces dernières. Cela permet de simplifier le design de ce dernier.

Par opposition, un processeur CISC implémentera des instructions complexes dont l'exécution peut prendre un temps non négligeable, par exemple une copie d'un bloc mémoire d'une adresse vers une autre.

Le processeur CISC sera souvent plus aisé à programmer en assembleur. D'un autre côté, le processeur RISC sera souvent plus rapide pour une consommation moindre (et une dissipation de chaleur moindre).

4.6 Harvard ou Von Neumann

Les données traitées par un processeur n'ont pas toujours la même taille que les instructions qu'il traite. Par exemple, un PIC 16F possède des instructions de 14 bits alors qu'il travaille avec des données 8 bits.

Utiliser la même mémoire pour stocker le programme que pour les données provoquerait une perte de 1 bit par octet. Il peut alors être plus intéressant d'utiliser des mémoires séparées pour les instruction et pour les données.

Cela a également pour avantage de simplifier la réalisation, des bus différents étant utilisés pour les traitements des données et les lectures des instructions au lieu de devoir partager les mêmes bus (adresse et données).

Cette séparation des instructions et des données correspond à l'architecture Harvard. Utiliser la même mémoire pour stocker les deux correspond à l'architecture Von Neumann.

Harvard	Von Neumann
Simple à mettre en oeuvre, plus rapide	Permet de répartir la mémoire selon les besoins
Les tailles des mémoires programme et data doivent être fixées à l'avance	Le circuit est plus complexe, chauffe plus

4.7 Bus I/O

Certains processeurs proposent une séparation entre les accès destinés à la mémoire et ceux destinés aux périphériques.

Cela se fait généralement à l'aide de deux broches, chacune étant active pour le type d'accès correspondant.

Le processeur possédera des instructions spécifiques pour les accès aux entrées sorties, distinctes de celles utilisées pour les accès mémoire (et généralement en nombre inférieur).

En l'absence de telles instructions, une partie de la zone adressable comme

de la mémoire sera utilisée à la place pour accéder aux périphériques, réduisant d'autant la taille mémoire disponible.

4.8 MMU

Un processeur avec 16 bits d'adresse ne pourra accéder qu'à 64Ko de mémoire, avec 24 bits, à 1Mo et avec 32 bits à 4Go.

Si on désire accéder à plus de mémoire, il sera nécessaire de découper la mémoire en *pages* et d'insérer sur le bus d'adresse en sortie du processeur un circuit qui sera chargé de sélectionner les pages auxquelles on accèdera.

Ce circuit s'appelle *Memory Management Unit* ou MMU.

Certains processeurs utiliseront également un MMU afin de permettre de contrôler l'accès à la mémoire ou de la réarranger selon les besoins. Le principal type de MMU que l'on utilisera à cet égard sera un mécanisme de pagination, qui utilisera les bits de poids fort de l'adresse pour trouver la page à laquelle accéder à l'aide d'une ou de plusieurs tables.

4.9 Pipelining

Afin d'accélérer l'exécution des instructions, on peut exploiter le fait que ces dernières ont besoin de plusieurs cycles pour s'exécuter.

Pendant qu'on exécutera le second cycle d'une instruction, on commencera a exécuter le premier cycle de l'instruction suivante et ainsi de suite.

Si les instructions sont découpées en 5 cycles, l'accélération ira jusqu'à 5 fois plus rapidement. En pratique, à chaque fois qu'il y aura un saut à un autre emplacement en mémoire, les instructions qui suivent le saut devront être annulées (puisque non exécutées) et le pipeline se retrouvera vide.

De même, dans certains cas, une instruction pourrait dépendre du résultat de l'instruction précédente et demander à être retardée d'un ou plusieurs cycles le temps que ce résultat soit disponible.

Les processeurs RISC auront tendance à ne pas gérer le fait de retarder des instructions, ce qui demande une série de précautions prises lors de la création du programme (généralement prises en charge par le compilateur).

L'architecture Harvard profite clairement au pipelining vu que des bus différents sont utilisés pour le cycle *fetch* (lecture d'instruction) et pour l'exécution de l'instruction.

Instruction 1	F	D	E1	E2				
Instruction 2		F	D	E1	E2			
Instruction 3			F	D	E1	E2		
Instruction 4				F	D	E1	E2	
Instruction 5					F	D	E1	E2

Fetch, Decode, Execute 1, Execute 2

TABLE 4.2 – Pipelining

4.10 Processeur vectoriel

Afin d'encore accélérer plus l'exécution des programmes, on pourra dédoubler une partie du processeur (ALU, bus internes) afin de pouvoir exécuter deux instructions en même temps. Les mêmes restrictions que pour le RISC peuvent apparaître (nécessité de retarder une instruction).

4.11 La pile

La plupart des processeurs proposent une pile LIFO contrôlée par un registre. Il sera possible d'y placer des valeurs (généralement, la pile croît vers les adresses inférieures). Pour ce faire, on diminuera le pointeur de pile (souvent appelé SP) avant d'y inscrire la valeur à stocker.

Lorsque l'on voudra récupérer une valeur de la pile, on la lira puis on augmentera le pointeur de pile pour qu'il prenne la valeur qu'il avait avant que la valeur ne soit stockée.

Certains processeurs permettent d'utiliser n'importe quel registre comme pointeur de pile ou de gérer également des piles ascendantes.

Cette pile peut être utilisée pour stocker temporairement des données mais elle est surtout utilisée pour gérer les appels de sous-programme.

La plupart des processeurs possèdent une paire d'instructions pour gérer les sous-programmes.

L'instruction CALL [2] va stocker sur la pile l'adresse de l'instruction suivante (la valeur de PC). avant de sauter à l'adresse fournie (adresse du sous-programme).

L'instruction RET récupérera de la pile l'adresse de retour et y sautera.

A noter que certains processeurs (généralement RISC) utiliseront plutôt une instruction Branch And Link qui au lieu de sauver l'adresse de retour

2. ou JSR ou autre

sur la pile la sauvera dans un registre. D'autres utiliseront une pile interne (qui sera par contre limitée en nombre de niveaux disponibles).

4.12 Utilisation par le compilateur

Le compilateur est chargé de convertir un programme dans un langage de haut niveau en code prévu pour le processeur.

Si les différents calculs peuvent être implémentés directement en langage machine, le problème d'appel de fonctions (et de variables locales dans ces dernières) se pose.

La solution utilisée est la suivante :

1. On pousse sur la pile les paramètres
2. On appelle le sous-programme correspondant à la fonction désirée
3. On place les variables locales sur la pile
4. On effectue les opérations du sous-programme
5. On retire les variables locales (pour ramener le pointeur de pile au niveau de l'adresse de retour)
6. On retire les paramètres de la pile
7. On récupère le résultat dans un registre (résultat unique)

En langage C (et bien d'autres), les paramètres seront poussés sur la pile en commençant par le dernier. En effet, le C autorise des fonctions avec un nombre de paramètres variables . En utilisant cet ordre, on peut toujours repérer les premiers paramètres (qui se trouvent juste après l'adresse de retour sur la pile).

Afin de faciliter l'accès aux paramètres de la fonction et aux variables locales, on utilisera une variable (Frame Pointer, Base Pointer ou autre) que l'on sauvera sur la pile en début de sous-programme avant de la faire pointer sur la pile.

Comme le nombre de paramètres est variable, ce sera le code appelant qui se chargera de les enlever (vu que c'est lui qui les a placé sur la pile) [3]

4.13 Cache

Les accès mémoire sont souvent plus lents que le processeur. Il arrivera donc régulièrement que le processeur doive attendre que les données soient disponibles.

3. Pour le langage Pascal, on stocke les paramètres du premier au dernier et c'est la fonction appelée qui les retirera

BP+n+1	Paramètre$_n$
BP+n	Paramètre$_{n-1}$
	...
BP+2	Paramètre$_1$
BP+1	Adresse de retour
BP	ancien BP
BP-x	Variable locale

Les valeurs de BP sont comptées en *mots*. Si les valeurs de la pile sont sur 32 bits et BP en octet, il faudra compter BP, BP+4, BP+8, ...

TABLE 4.3 – Pile pour les appels en C

Or, durant l'exécution d'un programme, 90% du temps est passé dans 10% du code et 90% des accès aux données se font sur 10% des données.

Cette localisation, tant du code que des données, permet d'utiliser une mémoire rapide au niveau du processeur. Si les données ont été lues récemment, plutôt que de les relire de la mémoire, il sera plus rapide de les lire dans cette mémoire.

Quelle que soit l'architecture (Harvard ou Von Neumann), on peut trouver deux caches séparés, l'un pour les instructions et l'autre pour les données ou un cache unique.

Ce cache est généralement de petite taille, lorsqu'il est rempli, il sera nécessaire de libérer les données les plus anciennes. De même, en cas d'écriture, l'information sera généralement écrite immédiatement en mémoire (même si le processeur peut continuer son travail pendant que le mécanisme de cache s'occupe de l'écriture).

Chapitre 5

La carte mère et les circuits liés

La carte mère sert entre autre à relier les différentes parties d'un ordinateur.

Afin d'éviter des connexions incorrecte, les différents connecteurs ont des formes (et parfois des couleurs) différentes, chaque forme correspondant à un type de composant ou périphérique [1]. Ces ports ont également des détrompeurs qui permettent d'éviter de se tromper de sens d'insertion.

C'est ainsi que sur une barrette de mémoire, on trouvera une ou des encoches qui correspondent à des détrompeurs dans le socket de RAM qui lui correspond afin de ne pas pouvoir brancher une barrette incompatible (type ou tension différents).

5.1 Chipset et bridges

Les premiers ordinateurs de type PC ne contenaient que le strict minimum sur la carte mère, il était donc nécessaire de brancher une carte contrôleur disque, une carte avec les ports séries et parallèles, une carte contrôleur de lecteur de disquettes [2], une carte vidéo, une carte son, ...

Sur les cartes mères modernes, on trouve un bon nombre d'interfaces incluses en standard sur la carte mère : contrôleur disque, USB, son, réseau, vidéo et parfois WiFi ou Firewire. Ces interfaces sont reprises dans un composant parfois qualifié de *chipset*.

1. Seule exception : le port qui permet de brancher le boîtier : boutons reset et power, led d'alimentation et de disque dur, haut-parleur intégré. La position de ces différents câbles est néanmoins standardisée et certains boîtiers proposent un connecteur unique à brancher sur la carte mère

2. A noter que rapidement, on a trouvé des cartes *multi-I/O* qui embarquaient ces trois contrôleurs sur une même carte

Le processeur étant généralement plus rapide que le reste du système et ses signaux étant relativement bruts, on intègre un circuit appelé *North Bridge* qui sert d'intermédiaire entre le processeur, la mémoire et le reste du système.

Les I/O plus classiques ayant une vitesse encore inférieure à celle de travail du North Bridge, un second composant appelé *South Bridge*, connecté au premier, se chargera des échanges avec les bus plus lents (tels que le bus PCI), les I/O standard (disque, son, ...).

Le Chipset décrit ci-dessus est souvent inclus dans le South Bridge.

Les deux Bridges doivent avoir été conçus pour travailler ensemble vu l'absence de standardisation en ce qui les concerne.

5.2 CMOS

Sur la carte mère, on trouve également un composant généralement appelé le CMOS (en référence à la technologie du même nom).

Ce composant s'occupe de deux tâches importantes : d'une part il contient l'horloge temps réel qui permet à l'ordinateur de rester à l'heure même s'il est éteint et d'autre part, il contient une petite mémoire dont le contenu est également conservé lorsque l'ordinateur est éteint.

Pour ce faire, il est relié à un accumulateur (super-condensateur, batterie rechargeable ou pile lithium-ion) qui se charge de l'alimenter lorsque l'ordinateur est éteint. Cela lui vaut d'ailleurs de souventrester un composant séparé même sur les systèmes les plus modernes (afin que la pile n'alimente que le strict nécessaire, le CMOS et le quartz qui lui sert de référence de temps).

La mémoire incluse dans le CMOS est utilisée pour conserver la configuration hardware de l'ordinateur ainsi que le ou les mot(s) de passe utilisé(s) pour l'accès à la machine (avant le chargement de l'OS).

5.3 DMA

Fréquemment, le système doit copier un bloc de mémoire d'une adresse vers une autre ou transférer des données entre la mémoire et les entrées-sorties.

Effectuer cette tâche à l'aide du processeur est généralement lent et inefficace. Par exemple, sur Z80, une boucle effectuant cette copie prendrait plus de 50 cycles par octet copié et l'instruction spécialisée LDIR prendrait 22 cycle par octet alors qu'en commande directe du bus, cela pourrait être effectué en 2 cycles par octet.

On trouve donc des composants spécialisés appelés DMA (*Direct Memory Access*) qui peuvent être programmés pour de tels transferts.

Pendant que le DMA accède à la mémoire, il doit disposer de l'accès aux bus d'adresse et de données. Le processeur ne peut donc pas en disposer au même moment. Il existe deux manière de régler le problème :

— vol de cycle : le DMA bloque le processeur pendant qu'il accède à la mémoire. Il peut ainsi bloquer le processeur pendant une durée plus ou moins longue.

— intercalage : le DMA intercale ses accès entre deux accès effectués par le processeur. Le transfert sera plus lent mais le processeur peut continuer à travailler

Un ordinateur dispose généralement de plusieurs canaux DMA. Cela permet par exemple d'en utiliser un premier pour les transferts de données venant d'un disque, un second pour la carte son et un troisième pour une copie en mémoire.

Chapitre 6

La carte graphique

6.1 Le signal vidéo

6.1.1 Signal monochrome analogique

Les premiers moniteurs utilisaient un tube cathodique. Un canon à électrons envoie un faisceau d'électrons vers la surface de l'écran. Pour ce faire, une tension de plusieurs milliers de volts est appliquée entre le canon à électrons (négatif) et la surface de l'écran (positive).

Lorsque les électrons touchent cette surface, une substance phosphorescente va s'allumer, faisant ainsi apparaître un point lumineux. Sur les premiers moniteurs, on avait une phosphorescence verte.

Entre le canon à électron et la surface de l'écran, on trouve trois dispositifs : une électrode chargée de contrôler la luminosité (plus elle est positive, plus elle va repousser et donc bloquer les électrons) et deux paires d'électrodes servant, l'une à dévier le faisceau verticalement et l'autre à le dévier horizontalement [1].

L'électronique fourni au niveau des circuits de déviation du faisceau des signaux en forme de rampe : le signal évolue de -V à +V progressivement puis retombe quasi instantanément à -V. Ces signaux constituent le balayage horizontal et le balayage vertical.

Afin que l'image soit stable, le retour à la valeur minimale pour les signaux de balayages sont commandés par des impulsions qui accompagnent le signal vidéo, on parle des signaux de synchronisation verticale et horizontale.

Sur le temps nécessaire pour effectuer un balayage horizontal, le signal de balayage vertical va amener le faisceau au niveau de la ligne suivante, les lignes de points à l'écran sont donc légèrement inclinées vers le bas. Une

1. Il peut aussi s'agir d'électro-aimants

fois le balayage horizontal revenu au début de ligne, le circuit sera prêt pour tracer la ligne suivante.

La période pendant laquelle le signal de balayage revient à son minimum est appelée période de retour de balayage. Pendant ce temps, le signal vidéo est coupé afin que l'on ne voit pas à l'écran ce tracé. On parle de période de *blanking*.

Les anciens moniteurs utilisaient ainsi trois signaux : le signal d'image (succession de niveaux hauts (pixel allumé) et bas (pixel éteint) et les deux signaux de synchronisation. Le signal d'image est parfois également appelé signal d'intensité (Y ou I), les signaux de synchronisation sont généralement appelés synchronisation horizontale (H-SYNC) et synchronisation verticale (V-SYNC).

6.1.2 Entrelaçage

Dans les premiers moniteurs, les fréquences des signaux de balayage étaient fixe.

Pour augmenter la résolution horizontale, il suffisait d'augmenter la fréquence à laquelle les pixels sont envoyés afin d'en envoyer plus dans la même période de temps.

Par contre, cette méthode n'était pas possible pour modifier la résolution verticale, liée aux rapports entre les fréquences des balayages horizontaux et verticaux.

Comme les lignes sont légèrement obliques, un moyen facile de doubler la résolution verticale a été de commencer une image sur deux au milieu de l'écran. Les lignes ainsi insérées se situaient entre les lignes du balayage normal.

On appelle ce procédé l'entrelaçement. Pour y arriver, il suffisait de retarder l'impulsion de synchronisation horizontal de la moitié du temps nécessaire au balayage horizontal une image sur deux.

6.1.3 Video composite monochrome

Plutôt que de transmettre ces trois signaux séparément, on les combine parfois en un signal unique.

Le signal vidéo sera une tension entre 0,3V (noir) et 1V (blanc ou vert). Il est évidemment possible d'avoir des signaux intermédiaires pour des luminosités différentes.

L'impulsion de synchronisation horizontale sera un signal de 0V. Cette impulsion sera suivie d'une période pendant laquelle le signal sera au noir correspondant au blanking du retour horizontal.

La synchronisation verticale sera indiquée par une série d'impulsions de synchronisation. Afin de permettre l'entrelacement, on utilise deux séries d'impulsions différentes selon que l'on se trouve au niveau des trames (images) paires ou impaires.

6.1.4 Video couleur

Pour afficher une image en couleur, on utilise trois canons à électrons et on place une plaque perforée entre le canon et l'écran. Cette plaque permet de ne permettre le passage des faisceaux d'électrons qu'à certains endroits. Les trois faisceaux étant décalés les uns par rapport aux autres (soit en ligne, soit en triangle, selon la disposition des canons à électrons), ils arriveront à des endroits différents de la surface de l'écran. Ces endroits auront des substances phosphorescentes différentes qui donneront des couleurs rouge, verte et bleue.

Ces trois couleurs (Rouge, Vert et Bleu) sont les trois couleurs primaires pour la synthèse additive. Lorsque l'on ajoute des lumières de ces différentes couleurs, on peut générer à peu près toutes les couleurs possibles, les trois lumières dans les proportions correctes donnant de la lumière blanche. [2].

Le circuit reste semblable, à la différence que l'on utilisera trois signaux R (Red), G (Green) et B (Blue) au lieu d'un signal de luminosité.

Souvent, le signal de luminosité restera néanmoins présent, soit appliqué aux trois autres (pour un contrôle de la luminosité global), soit pour la compatibilité avec les anciens systèmes. La luminosité suit normalement la règle

$$Y = 0,3R + 0,6G + 0,1B$$

Une difficulté est de parfaitement synchroniser les signaux R, G et B. Un léger décalage donnant une impression d'image fantôme. Pour contourner le problème, on utilise les faits que dans une image la teinte de la couleur change moins vite que la luminosité et que l'oeil est plus sensible à une variation de luminosité que de couleur.

Au lieu de transmettre les signaux R, G, B, on transmettra donc le signal Y et deux signaux que l'on appellera signaux de Chrominance (liés à la couleur) qui seront B-Y et R-Y.

On parlera de YC_BC_R ou YUV selon les normes et les facteurs multiplicatifs appliqués à B-Y et à R-Y. Souvent, les signaux B-Y et R-Y sont générés

2. par opposition, lorsque l'on utilise des peintures ou des encres, on parle de synthèse soustractives, les trois couleurs primaires deviennent le magenta, le cyan et le jaune et un mélange correct donnera du noir

à une fréquence moindre que le signal Y (vu que l'oeil y est moins sensible).

6.1.5 Video composite couleur

Les signaux B-Y et R-Y seront ajoutés au signal composite de départ en étant modulés à une fréquence plus élevée que celle du signal vidéo. Pour ce faire, on transformera ces deux signaux en une amplitude et une phase (conversion entre les coordonnées polaires et cartésiennes) appliquées à la porteuse.

Afin de servir de référence de phase, on superpose au signal de blanking un signal appelé Burst qui sert à la synchronisation de la porteuse utilisée pour la couleur. Dans la norme PAL, on ajoute une inversion de phase de 180° une trame sur deux, ce qui permet d'éviter les changements de couleurs qui étaient fréquemment présents en NTSC.

6.1.6 Moniteurs analogiques modernes

Les moniteurs modernes prévoient les signaux H-SYNC, V-SYNC, R, G et B.

A ces signaux, on ajoute un bus I2C [3]. Ces signaux forment le bus DDC qui permet de transmettre les caractéristiques et références du moniteur.

La prise SVGA prévoir en outre un signal de masse par signal vidéo afin de limiter les interférences.

Le moniteur est généralement capable d'utiliser plusieurs fréquences de balayage afin de permettre des résolutions différentes. Les premiers utilisaient la polarité des signaux de synchronisation (normalement bas, impulsion haute sur la synchronisation ou normalement haut et impulsion basse) pour sélectionner la fréquence de balayage. Les moniteurs plus récents se basent sur les fréquences des signaux de synchronisation pour choisir les fréquences de balayage.

6.2 Les cartes vidéo 2D

6.2.1 Cartes textes monochromes

Les cartes vidéo 2D de départ fournissait un écran en mode texte : la mémoire écran contenait les codes des caractères que l'on désirait afficher.

3. système prévu pour la communication entre deux circuits électroniques à l'aide d'un signal d'horloge et d'un signal data

Il fallait alors lire dans une autre zone de mémoire la définition des dits caractères.

Ces définitions de caractères reprenaient 8 lignes de pixel (ou plus) par caractère, chaque caractère étant représenté sur une grille de taille fixe.

A côté de la mémoire écran se trouvait également une mémoire ROM qui proposait un ensemble de fonctions normalisé pour afficher des caractères, vider l'écran, ...Cette ROM contenait aussi les définitions de départ des caractères.

Le reste de l'électronique se chargeait de générer les signaux vidéos nécessaires

6.2.2 CGA, EGA, VGA

Une série d'améliorations ont été apportées à ces cartes vidéos de base.

Tout d'abord, on a commencé à supporter un mode graphique "haute définition". Cela demandait plus de mémoire écran et la nouvelle mémoire (l'ancienne étant conservée pour des raisons de compatibilité) ne pouvait souvent être atteinte que par *pages*[4].

Ensuite, les couleurs ont vu leur apparition. Pour gérer ces dernières, il a été nécessaire d'utiliser des circuits chargés de convertir un code couleur en différents niveaux électriques (les cartes modernes permettent 256 niveaux pour chaque composante). On a donc du ajouter en sortie des DAC[5] dont la vitesse a longtemps été le principal obstacle aux résolutions plus élevées.

Afin de limiter la mémoire utilisée, on utilisait un mécanisme de *palette de couleurs* : une table donnait les codes rouge, vert et bleu pour un ensemble de couleurs (4, 16 ou 256).

6.2.3 VESA

Avec l'utilisation sans cesse croissante d'images numérique, des modes vidéo *high color* qui utilisaient 15 ou 16 bits par pixel, encodant directement la couleur sur 5 ou 6 bits et *true color* utilisant 24 bits par pixel (8 bits par composante) ont vu le jour.

De même, là où le VGA était limité en 640x480, ces nouvelles cartes permettaient des résolutions plus élevées.

Chaque carte utilisait ses propres méthodes pour accéder à la mémoire. On a dès lors vu un standard appelé VESA qui a proposé une numérotation

4. Comme en fait, chaque page ne contenait qu'un bit de la valeur finale, on disait plutôt qu'elle était accédée par *plan*

5. Convertisseurs numérique vers analogique

standard pour les différents modes écrans et des fonctions qui permettent d'accéder de manière standardisée à la mémoire.

En particulier, sur les ordinateurs 32bits, le système VESA permet de mapper l'entièreté de la mémoire vidéo dans l'espace 32bits. C'est ce qui permet à Linux d'utiliser n'importe quelle carte en mode graphique : il suffit de choisir le mode avant de démarrer le noyau de Linux et d'utiliser l'écran en mode graphique du début à la fin.

6.2.4 Accélération 2D

Deux opérations qui doivent très souvent être effectuées sont d'une part le remplissage d'une zone de l'écran par une couleur donnée (par exemple pour effacer l'ancien contenu) et d'autre part de copier un bloc de graphiques d'une adresse vers une autre (opération bitblit).

Certaines cartes graphiques ont donc commencé à embarquer un circuit dédié capable d'effectuer ces opérations sans solliciter le processeur (un peu comme les DMA). Malheureusement, chaque constructeur utilisait sa propre convention et en gardait les spécifications bien cachées, ne permettant de les utiliser qu'au travers de drivers dédiés.

6.3 Les cartes vidéo 3D

6.3.1 Un peu de mathématiques

Lorsque l'on veut représenter une image en 3D, on prend un nuage de points (appelés vertex). On représentera alors une série de triangles dont les sommets sont des vertex et on finira par appliquer une (ou des) texture sur ces triangles.

Les Vertex sont représentés par leurs coordonnées (X,Y,Z). Selon la position de l'objet et ses mouvements, ces vertex vont subir des transformations géométriques.

Les mathématiques nous apprennent que l'on pourra effectuer des mises à l'échelle (soit selon tous les axes, soit selon certains axes seulement) en multiplicant ce vecteur par une matrice diagonale dont les termes sur la diagonale sont les rapports selon les différents axes.

Pour effectuer une rotation selon un axe, on remplacera les termes correspondant aux deux autres axes par des sinus ou cosinus de l'angle de la rotation.

Si on désire effectuer plusieurs de ces transformations à la suite l'une de l'autre, il suffira de multiplier les matrices pour obtenir une matrice de

transformation unique.

Mais il se pose le problème de la translation. En effet, cette dernière ne pourra pas s'exprimer en multipliant le vecteur des coordonnées par une matrice. On utilise alors l'artifice d'ajouter aux coordonnées une quatrième valeur qui vaudra toujours 1.

Pour les matrices vues précédemment, il suffira d'ajouter un 0 aux lignes de la matrice 3x3 et d'ajouter une ligne qui ne vaudra 1 que pour la dernière colonne, les autres valant 0.

Pour effectuer une translation, on remplacera les valeurs en fin des 3 premières lignes par les déplacements selon les différents axes. Cette matrice augmentée permettra de combiner toutes les transformations en une matrice unique.

On prendra alors les triangles un à un pour les afficher dans le buffer vidéo. Pour cela, on effectuera un produit avec une matrice 4x2 qui permettra de transformer ces coordonnées de points en coordonnées à l'écran. On parle de matrice de projection.

Chacun des points du triangle sera comparé avec un Z-Buffer : pour chaque point à l'écran, on y stocke la valeur Z du dernier point affiché. Si la valeur Z du point que l'on désire traiter est plus éloignée que celle du Z-Buffer, cela veut dire que le point sera caché par un point déjà traité et il ne sera pas affiché. Sinon, on mettra à jour la valeur du Z-Buffer.

On effectuera une interpolation dans la texture pour déterminer la couleur du point à afficher ce qui donnera la valeur finale du point [6]

6.3.2 La carte graphique

Les cartes graphique 3D contiennent des circuits spécialisé dans les différentes opérations décrites ci-dessus.

Des systèmes tels que OpenCL ou CUDA permettent d'ailleurs d'utiliser les cartes graphiques pour effectuer des opération à la place du processeur.

Il va de soi que les opérations qui gagnent le plus à être exécutées d'une telle manière sont celles reprises ci-dessus.

Ces cartes possèdent également une grande quantité de mémoire, ce qui permet de stocker des textures, des images *intermédiaires* qui seront utilisées pour calculer l'illumination, les effets de brouillard, de réflexion, de réfraction, ...

Encore plus que pour les cartes 2D, la structure des cartes 3D est conservée secrètement par leur fabricants et leur utilisation est liée aux drivers

6. D'autres traitements sont effectués pour gérer l'éclairage, les réflexions, ...

fournis par le fabricant [7].

7. ou par reverse engineering

Chapitre 7

La carte son

Pour générer les sons, on utilise deux grands principes : le premier est orienté sur les échantillons sonores et le second sur un synthétiseur de sons.

Certaines cartes ne possèdent qu'un seul des deux mécanismes, comme les anciennes cartes AdLib qui ne possédaient que la synthèse FM ou certaines cartes modernes qui ne proposent que le mécanisme d'échantillonnages

7.1 Echantillonnage

Le principe de l'échantillonnage est de mesurer le niveau du signal électrique à une fréquence fixe et, lorsque l'on veut rejouer le son, de générer des tensions égales à celles échantillonnées à la même fréquence.

Plus la fréquence d'échantillonnage sera élevée, plus le signal échantillonné sera précis et proche du signal d'origine. La fréquence de Nyquist, égale au double de la plus haute fréquence que l'on désire conserver, est la fréquence d'échantillonnage minimale nécessaire si on désire que le son soit rendu correctement.

Attention cependant qu'une note jouée par un instrument possède un grand nombre d'harmoniques, signaux se trouvant à des fréquences multiples de celle de la note. Ce sont ces harmoniques qui donne à la note sa "texture" propre à l'instrument utilisé (sans elles, on aurait un simple signal sinusoïdal). Il faudra donc prendre en compte la fréquence de la plus haute harmonique que l'on désire conserver (et non celle de la plus haute note).

On considère généralement qu'une fréquence de 44100Hz suffit pour échantillonner la totalité de ce que l'oreille humaine peut percevoir (l'oreille est sensible aux signaux de quelques Hz à environ 18000-20000Hz, ce qui laisse un peu de marge). Les systèmes modernes montent souvent à 48000Hz voire plus.

Le mécanisme d'échantillonnage de carte son permettra généralement de convertir un signal entrant en échantillons et de générer un signal à partir d'échantillons.

7.2 Synthèse FM

De nombreuses anciennes cartes son utilisaient un système hérité des synthétiseurs analogiques : des oscillateurs dont la fréquence est contrôlée par le niveau du signal à leur entrée, des générateurs d'enveloppe qui modulent l'amplitude du signal selon un cycle ADSR (Attaque, Descente, Soutien, Relachement, les quatres phases de base que l'on retrouve dans les sons). Les sorties des oscillateurs passent par les générateurs d'enveloppe avant d'être réinjectés dans l'autre oscillateur de la paire, ce qui permet d'obtenir des sons potentiellement très complexes.

Certaines cartes permettaient également de travailler en synthèse additive : plusieurs oscillateurs étaient utilisés en parallèle par note, chacun générant soit le signal de base, soit une harmonique.

7.3 Synthèse par échantillonage d'instruments

Afin d'obtenir des sons le plus proche possible des instruments réels, une autre méthode a été utilisée : on échantillonne une note à une fréquence de référence et on la rejoue à une fréquence différente pour obtenir une autre note.

Cette méthode était utilisée déjà à l'époque de l'Amiga pour les fichiers .MOD et semblables mais s'est retrouvée implémentée en hardware au niveau de cartes sons.

Ce changement de fréquence modifie hélas un peu trop les caractéristiques des notes lorsque la fréquence est fortement changée. On utilisera donc plusieurs échantillons à des fréquences différentes. Cet ensemble d'échantillons servira à définir un instrument (à noter l'intrument spécial "batterie" en midi où chaque note correspond à un effet de batterie différent et donc à un échantillon différent).

On retrouve ainsi des banques d'instruments (.SF2 par exemple) qui permettent d'obtenir un bon rendu pour les instruments de musique. Chaque échantillon est divisé en 3 sections, le début de la note, une partie qui sera répétée tant que la note continue et la fin de la note (qui correspond à la phase de relâchement).

Ce système a remplacé la synthèse FM dans les cartes sons modernes.

Si la carte son ne supporte pas le rendu des notes, il peut également être effectué en temps réel par un logiciel qui utilisera la sortie échantillonnée à la place.

Chapitre 8

La carte réseau

Afin de connecter différents ordinateurs, on utilise souvent un réseau de type ethernet. Dans ce réseau, les paquets passent d'une carte à l'autre au travers de fils électriques.

Chaque carte réseau dispose d'une adresse ethernet composée de 6 octets (et généralement représentée octet par octet en hexadécimal). Afin de s'assurer que deux cartes réseau aient des adresses différentes, les combinaisons de valeurs des trois premiers octets sont attribuées par fabricant, à chaque fabriquant revenant la tâche de s'assurer que pour un même préfixe, les trois derniers octets soient différents entre les cartes réseaux.

Lorsqu'un paquet de données est transmis par Ethernet, deux tâches sont effectuées : d'une part, on ajoute des informations sur la source et la destination (l'adresse ethernet aussi appelée adresse MAC) et d'autre part, on effectue un encodage des données afin de pouvoir les transmettre correctement sur les fils électriques.

Cet encodage consiste à placer un marqueur spécial en début et en fin des données et à s'assurer que la séquence en question ne puisse pas apparaître dans le flux de données, en modifiant ces dernières au vol quand cela est nécessaire.

Au niveau de la réception, il faudra d'abord décoder le signal pour récupérer la séquence d'origine puis tester si l'adresse MAC correspond à celle de la carte réseau qui a reçu le paquet afin d'ignorer les paquets destinés à d'autres machines.

Les cartes réseaux modernes se chargent automatiquement d'écrire l'adresse MAC source dans le paquet et de détecter que l'adresse destination d'un paquet reçu correspond bien à celle encodée dans la carte, ce qui permet de décharger le processeur d'une partie du travail.

Le modèle OSI qui présente sous forme de couches successives les différents niveaux de la communication réseau prévoit des couches bien séparées. Ici,

les cartes réseaux gèrent totalement la 1ere couche (physique, les signaux électriques) mais également une partie de la 2eme couche (les adresses MAC). On n'a donc pas d'isolation nette entre les niveaux 1 et 2.

Chapitre 9

Les supports magnétiques

9.1 Le tout début : les cartes et bandes perforées

Au début de l'informatique, les données étaient stockées sur des cartes ou bandes perforées.

Une carte proposait 80 colonnes capables de contenir chacune un chiffre, une lettre ou un caractère spécial. Ces données étaient encodées en BCDic [1]

Les bandes perforées utilisaient une notation semblable à celles des télétype (au début, en utilisant le code *baudot* sur 5 bits, par la suite passant en ASCII)

Des appareillages permettaient de lire mais aussi d'écrire ces cartes et bandes perforées. Il s'agissait de deux périphériques totalement distincts.

9.2 Les bandes magnétiques, le début

Les bandes magnétiques utilisées pour sauver du son ont rapidement été utilisées pour stocker des données informatiques. Les codages étaient très simples et consistaient aux début à enregistrer un signal qui était à deux fréquences différentes, pour représenter les 0 et les 1.

Avec le temps, on a diminué le nombre de cycles utilisés à l'une ou l'autre fréquence pour se limiter à juste une transition. Si le délai avant le changement de niveau de signal, est en dessous d'un certain seuil, on est sur la première fréquence, sinon, on est sur la seconde.

1. 10 positions permettaient de choisir une valeur de 0 à 10, deux positions supplémentaires permettaient de choisir une *page* parmi 4

Le principal défaut de cette méthode est que le débit dépend du contenu vu qu'une des valeurs sera encodée par un signal plus court que l'autre. C'est cependant la méthode employée par les anciens ordinateurs tels que le ZX Spectrum pour effectuer des sauvegardes sur cassette audio.

On créera donc d'autres encodages qui prévoient un débit fixe. On parle généralement de GCR, FM et MFM. Ces encodage transforment la donnée en suite de 0 et 1, les 0 étant encodés par des absences de transition et les 1 par des transitions du signal enregistré. Une longue période sans transition risque d'amener une désynchronisation (la vitesse de lecture/écriture n'étant souvent précise qu'à 10% près).

Le GCR utilisera des blocs de n bits pour en encoder m avec $n > m$. Par exemple, les premiers Apple utilisaient 8 bits pour en représenter 5 et d'autres systèmes utilisaient des séquences de 5 bits pour encoder des valeurs de 4 bits.

Le FM utilisera des séquences 10 et 11 pour encoder les bits à 0 et à 1. Son principal défaut est que dans le cas d'une série de bits à 1, le nombre de transitions est très élevé, ce qui limite la vitesse d'enregistrement.

Pour diminuer le nombre de transitions, le MFM séparera les bits de données par le NON-OU entre ces derniers. Cela divise en moyenne le nombre de transitions par 2 et permet donc d'augmenter la densité des données d'autant.

9.3 Disques

Les bandes magnétiques posent plusieurs problèmes. Tout d'abord, le changement est peu pratique. Ensuite, il est nécessaire de faire défiler la bande jusqu'à l'endroit où se trouvent les données qui nous intéresse, ce qui peut prendre un certain temps.

Les disques permettent de simplifier la situation. La tête de lecture est positionnée de manière précise à l'aide d'un moteur pas à pas. Elle parcourra ainsi une zone différente du disque sur laquelle les données seront encodées en GCR, FM ou MFM.

Afin de permettre un positionnement encore plus rapide, les premières disquettes possédaient une série de trous positionnés à intervalle régulier près du centre. Deux ouvertures permettaient de lire ces trous, la première pour détecter le premier secteur, la seconde pour détecter le début de chaque secteur.

Avec le temps, les systèmes ont pu se limiter d'abord au trou indiquant le premier secteur, ensuite se passer tout simplement de cette information (même si le trou d'index de premier secteur restait présent).

Chaque secteur est enregistré en suivant la séquence suivante :

— zone de synchronisation : un signal spécifique qui permet de détecter le début de secteur et de synchroniser le mécanisme de lecture
— informations de secteur : numéro de piste et de secteur, taille du bloc de données
— zone de synchronisation
— données
— espace "tampon" qui sert à compenser le fait que d'une fois à l'autre, l'écriture d'un secteur peut ne pas avoir pris exactement la même durée (variation de vitesse d'écriture)

Les premières disquettes avaient un diamètre de 8" et étaient contenues dans une enveloppe souple, ce qui leur a valu le nom de *floppy disks*. Les suivantes ont eu un diamètre de 5"1/4 puis de 3"1/2 (ces dernières dans une enveloppe plus rigide) [2].

A côté des disquettes, on a créé des disques durs. Le même système était utilisé sur des plateaux rigides (amovibles ou non) ce qui permettait une plus grande stabilité de la vitesse de rotation et une plus grande densité de données.

Plusieurs plateaux pouvaient être utilisés, chaque plateau disposant de 1 ou 2 tête de lecture. On parlera donc de cylindre au lieu de piste, de tête et de secteur.

Sur PC, cette combinaison CHS utilisait des nombres de bits définis pour chaque dimension. Avec la croissance de la taille des disques durs, ce système n'a plus pu être utilisé et on est passé au système LBA qui numérote les secteurs du premier au dernier.

Les disques durs modernes prévoient des secteurs de réserve qui serviront à remplacer de manière transparente des secteurs qui seraient défectueux. Ce n'est que quand cette réserve est épuisée que des secteurs défectueux apparaîtront au niveau de l'ordinateur.

9.4 Les bandes magnétiques, vers un stockage plus élevé

Pour augmenter la quantité de données sur une bande magnétique et surtout, le débit de lecture/écriture, on pourrait être tenté de faire défiler la bande plus rapidement.

Cette méthode risque cependant d'amener des problèmes de rupture de la bande. Une autre méthode a donc été préférée, semblable à celle utilisée

2. on trouve également des disquettes de 3" qui utilisaient une enveloppe en plastique dur mais n'ont pas été conservées à cause de leur coûts plus élevés

dans les enregistreurs vidéo : la tête de lecture tourne sur un plan oblique par rapport au sens de défilement de la bande. Cela a pour effet que les données sont inscrites selon des traits obliques dans le sens de la largeur de la bande.

La vitesse de déplacement de la tête magnétique n'est plus celle de défilement de la bande mais est liée à la vitesse de rotation de la tête de lecture.

Cela a permis entre autres les enregistrements sur des cartouches DDS, utilisant des supports semblables aux cassettes DAT utilisées pour la musique.

On a vu ainsi les capacités des bandes magnétiques augmenter tout en restant sur des cartouches de taille restreintes et faciles à changer.

A côté de cela, on dispose de dispositifs *jukebox* (on parle de robots) qui permettent de changer automatiquement la bande présente dans le lecteur, ce qui permet des sauvegardes totalement automatisées de gros volumes de données.

9.5 Disques optiques

A côté de ces support sont apparus les supports de type optique et magnéto-optiques.

Un faisceau laser passe par une lentille qui permet de le focaliser sur un point précis d'un disque en rotation. La couche du disque présente à cet endroit contiendra soit une zone réfléchissante, soit une zone qui dispersera le faisceau lumineux.

Les premiers disques utilisaient un laser infra-rouge. Les progrès sur les diodes laser ont permis d'utiliser des couleurs associées à des longueurs d'ondes plus petites et donc, des faisceaux plus précis, tel que le laser bleu du Blu-Ray.

Pour l'écriture, les premiers systèmes utilisaient un électro-aimant (magnéto-optique) ou *grillaient* la cellule de données.

Pour les systèmes magnéto-optiques, le laser faisait fondre la couche contenant les données et les données étaient modifiées par l'électro-aimant.

On utilise maintenant des support dont la couche réfléchissante peut prendre deux états différents selon la température à laquelle ils ont été chauffés ce qui permet les supports réinscriptibles purement optiques.

9.6 Mémoires flash

Avec l'utilisation des ROM de type Flash est arrivé un nouveau type de support. Contrairement aux EEPROM classiques, les Flash peuvent être

produits à faible coût pour des volumes de mémoire élevés. Cela est entre autres réalisé en regroupant les données en secteurs qui doivent être effacés dans leur totalité avant un cycle d'écriture.

Ces mémoires Flash ont d'abord été utilisées dans des cartes mémoires, SmartCard, MMC, SD-MMC, Compact Flash et autres.

L'avènement du contrôleur USB a permis d'utiliser ces mémoires au niveau de clés USB qui intègrent la mémoire et un contrôleur chargé de faire apparaître cette dernière comme un disque dur.

On trouve ensuite des disques hybrides (mélange de disque dur classique et de mémoire FLASH) et des disques basés purement sur des mémoires FLASH (SSD).

Chapitre 10

Ports d'ordinateur

Pour communiquer avec l'extérieur, les ordinateurs ont besoin de connecteurs que l'on appellera généralement ports. Ces ports peuvent véhiculer directement les signaux du processeur ou au contraire des signaux déjà décodés, traités et mis en forme, selon les besoins.

10.1 Port série

Utilisé entre autres pour connecter des terminaux et imprimantes aux ordinateurs, le port série envoie les bits un après l'autre à un débit constant.

Afin de permettre de se synchroniser, on utilisera généralement un bit de départ (bit à 1) avant les données et un ou deux bits d'arrêt après ces données.

Un bit de parité pourra également être ajouté. Ce bit permettra de vérifier que le nombre de bits à 1 des données est correct.

Le protocole RS232 décrit cette connexion série standard, ainsi que les vitesses habituelles et les niveaux électriques.

On disposera de chaque côté d'un fil d'émission (TX) et d'un fil de réception (RX). Les connexions doivent être croisées entre les deux dispositifs (RX de l'un branché sur le TX de l'autre)

Comme une des deux extrémités peut être plus rapide que l'autre au niveau du traitement des données, on ajoute deux lignes supplémentaires également croisées qui serviront à gérer le flux : CTS et RTS.

Pour terminer, on trouve également parfois [1] des signaux qui indiquent si le dispositif de l'autre côté est "connecté". Pour un modem par exemple, cela permettra de faire la différence entre en ligne et raccroché.

1. ils sont présent sur les RS232 sur des connecteurs DB25 (25 broches) mais absents sur les DB9

Encore de nos jours, les communications séries sont utilisées, entre autres quand il s'agit de communiquer avec des périphériques tels que des micro-controleurs (utilisés dans des imprimantes 3D, appareils à découpe laser, machines outils, ...)

10.2 port parallèle

Un autre port qui a été très rapidement centralisé est le port parallèle utilisé pour les imprimantes. Ce port prévu principalement en sortie propose 8 fils véhiculant un octet et un fil *strobe* qui sert à indiquer que les données sont disponibles.

A vocation unidirectionnelle, il prévoit cependant quelques signaux de retour de l'imprimante pour communiquer son status : Busy qui sert à contrôler le flux, error qui permet d'indiquer un problème (bourrage papier par exemple), ACK pour confirmer la réception de la donnée, Paper Out pour indiquer l'absence de papier (on utilisait souvent du *papier listing*, bande de papier continue, cet indicateur indiquait que l'on était à la fin de la bande), ...

Des combinaisons de signaux de retour permettait des messages d'erreurs plus complexes (comme "printer on fire") même si le nombre de possibilités était limité.

Par la suite, le port a été rendu bidirectionnel (les données pouvaient être également envoyées par l'imprimante, par exemple pour la configuration automatique).

10.3 port floppy

Pour connecter un lecteur de disquettes, un port standard a été créé.

Ce port véhicule les signaux qui permettent le choix de la piste (track0, dir et step qui indiquent que l'on est sur la piste 0, indique la direction dans laquelle déplacer la tête de lecture et donne l'ordre de ce déplacement).

On trouve aussi les signaux qui servent à sélectionner le lecteur en lecture ou en écriture, à mettre le moteur en marche et à transférer les données.

Avec le temps, un des signaux de contrôle a changé de rôle. Alors qu'il servait initialement à retourner un status du lecteur, il a été réattribué pour indiquer si la disquette introduite est en simple ou en double densité.

Ce connecteur a également été détourné de son utilisation principale pour commander des lecteurs de bandes. L'encodage des données étant le même

sur la bande, seuls les signaux de contrôle et de déplacement de tête devaient être modifiés ce qui ne posait pas de problèmes.

10.4 Port IDE

Pour contrôler les disques durs, après avoir utilisé des ports propriétaires, le standard IDE a été choisi.

Ce port prévoit des données sur 16 bits, une adresse sur 3 bits, des fils permettant de sélectionner deux périphériques et quelques autres signaux de contrôle.

Le protocole de commande était également standardisé et s'appelait ATA. Des extensions ont été ajoutées sous la forme de la norme ATAPI qui permettait entre autres de contrôler un lecteur de CD.

Le principal problème de ce port est lié à la présence de nombreux signaux sur des câbles situés les uns à côté des autres. Plus la fréquence (et donc la vitesse de transfert) augmente, plus ces signaux se parasitent entre eux.

Avec la norme ATA-100 est apparu un nouveau câble où le nombre de conducteurs était doublé. Les nouveaux fils étaient tous reliés à la masse et permettaient d'isoler les différents signaux.

10.5 Port SATA

Afin d'augmenter le débit pour la connexion avec un disque dur, il a été nécessaire de passer à une connexion série qui permettait d'éviter les parasitages entre signaux.

La norme SATA utilise ainsi le protocole ATA (ou ATAPI) des ports IDE standard (renommés PATA pour Parallel ATA) tout en utilisant une communication de type série.

Ce port est devenu la norme actuelle pour les disques durs.

Un système de contacts placés à des positions différentes permet de rendre les périphériques *hot pluggable*. Lorsque l'on retire le câble, les signaux d'alimentation sont déconnectés avant les signaux de données ce qui permet d'éviter la présence de signaux incorrects lors de la connexion ou de la déconnexion à chaud du disque.

10.6 SCSI

Afin de se connecter à des périphériques externes tels que des scanners, des disques durs, des graveurs, ...on a longtemps utilisé le bus SCSI. Ce bus

permet d'adresser jusqu'à 7 (SCSI 1) ou 15 (SCSI 2) périphériques connectés sur le même câble.

Afin de permettre des transferts de données plus rapides, les extrémités sont adaptées en impédance à l'aide de *terminateurs*, réseaux de résistances qui permettent de s'assurer que les signaux ne soient pas réfléchis une fois la fin de la chaîne atteinte.

Au niveau SCSI, chaque périphérique possède un numéro d'identification, y compris la carte contrôleur (qui n'est pas comptée dans le nombre de périphériques). Les échanges peuvent se faire de n'importe quel périphérique vers n'importe lequel (même si généralements ils se font entre un périphérique et le contrôleur).

Le SCSI 3 a introduit des paires différentielles pour chaque signal (deux fils qui véhiculent des signaux opposés, le courant passant dans un fil revenant par l'autre fil). Ces paires différentielles permettent de diminuer les effets des différents parasites et donc, d'augmenter la vitesse de transfert.

De nos jours, le système SCSI est encore utilisé pour des lecteurs de bandes magnétiques.

10.7 Port USB

Le port USB est un port série utilisant une paire différentielle pour transmettre les données. En portant les deux signaux au même niveau, il est possible d'envoyer un signal spécial *attention*.

Lorsqu'un périphérique est connecté, le contrôleur USB lui demandera sa configuration. Le périphérique répondra en envoyant une série d'informations :

— Le Vendor ID, unique par constructeur
— Le Product ID, permettant à un constructeur de différentier ses différents périphériques. La combinaison Product ID et Vendor ID permettra de choisir le driver spécifique
— Les chaînes d'information (nom du constructeur et du périphérique)
— Le nombre de End-Points et leurs types. Un même périphérique USB peut être géré comme plusieurs devices différents. Chaque device sera connecté à un End Point différent

Il est possible de configurer un End Point en mode HID. Il s'agit d'un ensemble de normes pour des périphériques standards, ne demandant pas de drivers spécifiques. On trouve ainsi

— HID Keyboard pour les claviers
— HID Mouse pour les souris

— HID Joystick pour les joysticks (nombre d'axes et de boutons transférés en paramètres)
— HID audio
— HID MIDI pour les contrôleurs MIDI, possible de configurer en entrée ou en sortie
— HID Storage pour les disques durs et clés USB
— HID Serial pour les ports séries

On trouve également une autre norme, plus récente, utilisée entre autres pour les smartphones, appareils photos et tablettes : la norme MTP, qui permet le transfert des fichiers

10.8 Firewire

A côté de l'USB, on trouvait également le protocole Firewire, plus rapide que le premier à l'époque de l'USB 1, qui était utilisé principalement pour les disques externes et les caméras vidéos, demandant un haut débit de données.

Avec l'ubiquité de l'USB 2 et maintenant la sortie de l'USB 3, ce port est quasi disparu.