



IFT-17583

Structure interne des ordinateurs

Supplément

Pierre Marchand

Automne 2000

Table des matières

1. Supplément au volume	1
Chapitre 3.....	3
Chapitre 5.....	21
Chapitre 7.....	47
Chapitre 8.....	75
Chapitre 9.....	79
Chapitre 10.....	83
Chapitre 11.....	89
Chapitre 12.....	95
2. Appendices	111
3. Solution des exercices.....	121
4. Initiation à l'assembleur	141
1. Introduction.....	143
2.. Architecture du Pentium.....	145
2.1 Historique.....	145
2.2 Registres d'entiers	147
2.3 Registres de virgule flottante	150
2.4 Registres MMX.....	151
2.5 Registre SIMD	151
2.6 Le registre EFLAGS	152
2.7 Autres registres	153
3. Outils de programmation.....	154
3.1 Masm32.....	154
3.2 Déboguage	156

4. Types de données	162
4.1 Entiers	162
4.2 Virgule flottante	163
4.3 MMX.....	163
4.4 SIMD.....	164
4.5 Rangement des données en mémoire	164
4.6 Temps d'exécution des instructions	165
5. Structures de contrôle.....	167
5.1 Principales structures de contrôle	167
5.2 Branchements.....	169
6. Modes d'adressage.....	174
7. Instructions de base du Pentium	178
7.1 Directives de base	179
7.2 Un premier exemple.....	182
7.3 Un deuxième exemple.....	183
7.4 Un autre exemple	184
7.5 Entrées/sorties en mode console	185
7.6 Chaînes de caractères.....	187
7.7 Comparaison de chaînes	192
7.8 Recherche d'une chaîne dans une autre.....	193
7.9 Transposée d'une matrice.....	195
8. Instructions arithmétiques	197
8.1 Addition	197
8.2 Soustraction.....	198
8.3 Multiplication.....	199
8.4 Division.....	202
8.5 Décalages et rotations	203
8.6 Exemple : racine carrée par la méthode de Newton	208
9. Conversions.....	210
9.1 Conversion décimal à binaire.....	210
9.2 Conversion binaire à décimal.....	211
9.3 Conversion binaire à chaîne hexadécimale	213
10. La Pile	217

10.1 Empilement et dépilement	217
10.2 Sous-programmes	218
10.3 Blocs de pile.....	222
10.4 Récursivité	225
10.5 Tables de sauts	226
11. Interruptions	228
11.1 Interruptions logicielles	228
11.2 Interruptions matérielles	231
12. Directives d'assemblage et macros	239
12.1 Directives d'assemblage	239
12.2 Directives d'assemblage conditionnel	242
12.3 Macros.....	244
12.4 Directives pour structures de contrôle et programmation structurée	246
5. Instructions de base du Pentium.....	251
6. Mots réservés.....	310

1. Supplément au volume

Chapitre 3

Représentation interne des informations

3.2 Données non numériques

Pour représenter les caractères des principales langues de notre planète ainsi que les chiffres, les signes de ponctuation et certains caractères de contrôle, la communauté informatique est en train d'adopter un code de 16 bits appelé Unicode, qui contient 65 536 caractères différents. Les 256 premiers caractères d'Unicode sont le jeu de caractères appelé Latin-1, et les premiers 128 caractères de ce dernier ensemble sont équivalents au code ASCII encore utilisé dans la plupart des micro-ordinateurs. Unicode est en train de se répandre de plus en plus. Il est à la base du système d'exploitation Windows NT (donc Windows 2000) de Microsoft et est pleinement supporté par le langage Java.

Pour représenter tous les caractères non accentués de l'alphabet latin, majuscules et minuscules, plus les chiffres "0" à "9", les signes de ponctuation et divers caractères de contrôle («Retour», «Tab», etc.), un code de 128 caractères, donc 7 bits, s'avère suffisant. On utilise cependant un octet pour stocker ces caractères en mémoire, car l'accès à un octet est une opération naturelle pour la plupart des micro-ordinateurs. Il est important ici de bien faire la distinction entre les caractères numériques, codés "0" à "9", et les nombres 0 à 9, qui ont une représentation différente comme nous verrons plus loin. Le tableau suivant présente ce code ASCII. Sur 8 bits, il reste encore 128 possibilités qu'on peut utiliser pour les caractères accentués (é, è, ê, etc), et les symboles spéciaux ($\div$, $\pm$, $\leq$, etc.). Ils sont standardisés dans le code ISO Latin-1 (ISO-8859-1).

Les 32 premiers caractères du code ASCII sont non-imprimables, ce sont des caractères de contrôle. Remarquez les plus courants : le retour de chariot (*carriage return*) CR, le déplacement arrière ou effacement (*backspace*) BS, la tabulation (*horizontal tab*) HT, le changement de ligne (*linefeed*) LF. On trouvera en appendice comment générer ces codes de contrôle et les autres à partir du clavier d'un micro-ordinateur.

On trouvera également en appendice le code EBCDIC mentionné dans le livre et encore utilisé dans les grands ordinateurs.

Pour plus de détails sur Unicode, visitez les sites <http://www.unicode.org/> et en particulier :
<http://www.unicode.org/unicode/standard/standard.html>

Pour des détails sur IEC, visitez le site :

http://www.acl.lanl.gov/HTML_WG/html-wg-95q2.messages/0728.html

DEC			HEX			DEC			HEX			DEC			HEX		
0	NULL	00	32	SP	20	64	@	40	96	`	60						
1	SOH	01	33	!	21	65	A	41	97	a	61						
2	STX	02	34	“	22	66	B	42	98	b	62						
3	ETX	03	35	#	23	67	C	43	99	c	63						
4	EOT	04	36	\$	24	68	D	44	100	d	64						
5	ENQ	05	37	%	25	69	E	45	101	e	65						
6	ACK	06	38	&	26	70	F	46	102	f	66						
7	BEL	07	39	‘	27	71	G	47	103	g	67						
8	BS	08	40	(	28	72	H	48	104	h	68						
9	HT	09	41	)	29	73	I	49	105	i	69						
10	LF	0A	42	*	2A	74	J	4A	106	j	6A						
11	VT	0B	43	+	2B	75	K	4B	107	k	6B						
12	FF	0C	44	,	2C	76	L	4C	108	l	6C						
13	CR	0D	45	-	2D	77	M	4D	109	m	6D						
14	SO	0E	46	.	2E	78	N	4E	110	n	6E						
15	SI	0F	47	/	2F	79	O	4F	111	o	6F						
16	DLE	10	48	0	30	80	P	50	112	p	70						
17	DC1	11	49	1	31	81	Q	51	113	q	71						
18	DC2	12	50	2	32	82	R	52	114	r	72						
19	DC3	13	51	3	33	83	S	53	115	s	73						
20	DC4	14	52	4	34	84	T	54	116	t	74						
21	NAK	15	53	5	35	85	U	55	117	u	75						
22	SYN	16	54	6	36	86	V	56	118	v	76						
23	ETB	17	55	7	37	87	W	57	119	w	77						
24	CAN	18	56	8	38	88	X	58	120	x	78						
25	EM	19	57	9	39	89	Y	59	121	y	79						
26	SUB	1A	58	:	3A	90	Z	5A	122	z	7A						
27	ESC	1B	59	;	3B	91	[	5B	123	{	7B						
28	FS	1C	60	<	3C	92	\	5C	124		7C						
29	GS	1D	61	=	3D	93	]	5D	125	}	7D						
30	RS	1E	62	>	3E	94	^	5E	126	~	7E						
31	US	1F	63	?	3F	95	_	5F	127	DEL	7F						

Code ASCII

3.3 Données numériques

Pour vous aider dans ce chapitre, vous pouvez utiliser une calculatrice possédant des capacités binaires et hexadécimales. La Faculté des sciences et de génie de l'Université Laval approuve et recommande la calculatrice scientifique HP-20 S de Hewlett-Packard (environ 32 \$). Cette dernière peut effectuer les calculs sur des entiers binaires et hexadécimaux positifs et négatifs, et effectuer la conversion de ces quantités entre elles ainsi qu'avec le décimal. D'autres calculatrices sont également autorisées. Veuillez consulter la page FAQ de notre site Web pour la liste des modèles autorisés.

3.3.1 Entiers positifs ou nuls

Représentation hexadécimale

Comme on a pu voir, le code binaire n'est pas très compact et il nous est difficile de retenir une longue séquence de uns et de zéros. On utilise donc souvent les notations octale ($B = 8$) ou hexadécimale ($B = 16$) pour représenter des nombres binaires. Ceci est possible parce que 8 et 16 sont des puissances de 2. En octal, on regroupe les digits binaires trois par trois à partir de la droite et on exprime la valeur de chaque groupe de trois en chiffres ordinaires de 0 à 7. Comme en micro-informatique le nombre de bits est un multiple de 4 (1 octet = 8 bits), on utilise couramment la notation hexadécimale, qu'on obtient en regroupant les bits en groupes de quatre à partir de la droite et qu'on exprime en digits hexadécimaux (0 à F, voir livre de Zanella et Ligier p. 34) :

$$\boxed{10111110} = \boxed{1011} \boxed{1110}$$

$$190_{10} \qquad \qquad \qquad B \qquad E_{16}$$

La notation est ainsi beaucoup plus compacte et on a tout de suite une bonne idée de l'ordre de grandeur du nombre.

On utilise fréquemment le préfixe \$ ou le postfixe H ou h au lieu de l'indice 16 pour représenter une quantité hexadécimale. Par exemple, \$A = AH = Ah = 10₁₆. Dans ce supplément, nous utilisons la notation C, dans laquelle 0x précède les chiffres, par exemple, 0x0A.

Arithmétique binaire

Addition et soustraction

À la page 36 du livre, on trouve un exemple d'addition binaire. Voici deux autres exemples d'addition sur des nombres de 8 bits :

	1 1 1 1		1 1 1
190	1011 1110	173	1010 1101
+141	+1000 1101	+44	+0010 1100
331	1 0100 1011	217	1101 1001

Remarquez dans l'exemple de gauche, qu'on a une retenue après l'addition du dernier bit de gauche, ce qui fait que le résultat doit être représenté sur 9 bits. On écrit 0x14B* en notation hexadécimale

La multiplication et la division binaires s'effectuent comme en décimal, sauf qu'il n'y a que des zéros et des uns.

On trouve à la p. 37 du livre la représentation des nombres négatifs en complément à 2. Le livre mentionne d'autres représentations des nombres négatifs, mais les micro-ordinateurs courants utilisent tous le complément à 2. Notez que le complément à 2 doit s'effectuer sur un nombre de bits déterminé, par exemple, 8, 16 ou 32. Le complément à 2 de 4 sera 11111100 sur 8 bits et 1111111111111100 sur 16 bits.

Pour obtenir rapidement le complément à 2 d'un nombre binaire, on inverse tous les bits et on ajoute 1 au résultat. Pour obtenir le complément à 2 du même nombre exprimé en hexadécimal, on soustrait ce nombre de 0xFFFF et on ajoute 1 au résultat. Par exemple, le complément à 2 de 0x0004 sera $0xFFFF - 0x0004 + 1 = 0xFFFFB + 1 = 0xFFFFC$.

La soustraction binaire s'effectue par la plupart des processeurs en additionnant le complément à 2 du nombre à soustraire.

* Notez que dans ce supplément, on utilise la notation C pour représenter un nombre hexadécimal, i.e. les chiffres sont précédés de 0x.

Multiplication

Voici un exemple de multiplication binaire :

1011	multiplicande	11
1001	multiplieur	$\times 9$
1011	1 ^e produit partiel	99
0000	2 ^e " "	
0000	3 ^e " "	
1011	4 ^e " "	
1100011	Produit = somme des produits partiels	

Un algorithme pour la multiplication binaire par ordinateur de deux mots de n bits pourrait donc être le suivant (on suppose que les bits d'un registre sont numérotés de 0 à $n-1$ à partir de la droite) :

```

Placer le multiplicande dans un registre de  $2n$  bits
Mettre 0 dans le registre de  $2n$  bits réservé au produit
Pour  $i = 0$  à  $n-1$ 
    Si le bit  $i$  du multiplieur est 1
        alors additionner le multiplicande au produit
    Fin Si
    Décaler le multiplicande de 1 bit vers la gauche (en ajoutant un 0 à droite)
Fin Pour

```

Division binaire

On procède exactement comme pour la division en décimal, sauf que les seules valeurs possibles pour chaque digit du quotient sont 0 ou 1. Voici un exemple :

$$111101010111000 \div 11001$$

Diviseur	Dividende	Quotient	Résultat
11001	111101010111000		10011101000
	<u>-11001</u>	1	
	101		
	1011	0	
	10110	0	
	101101		
	<u>-11001</u>	1	
	10100		
	101000		
	<u>-11001</u>	1	
	1111		
	11111		
	<u>-11001</u>	1	
	110		
	1101	0	
	11011		
	<u>-11001</u>	1	
	10		
	100	0	
	1000	0	
	reste 10000	0	

Pour effectuer cet algorithme par ordinateur, on procède comme suit :

Mettre 0 dans le registre Reste

Pour $i = 1$ à n

Décaler registre Dividende à gauche de 1 bit. Placer le bit sortant dans C.

Décaler le registre Reste vers la gauche en remplaçant le bit de droite par C.

Si $\text{Reste} \geq \text{Diviseur}$, alors

Reste = Reste - Diviseur

C = 1

Sinon C = 0

Fin Si

Décaler Quotient à gauche de 1 bit en remplaçant le bit de droite par C.

Fin Pour

Pour économiser un registre, on place parfois le quotient dans le même registre que le dividende.

Exemple : $111101010111000 / 11001$. On place initialement le dividende dans le registre Q et le diviseur dans le registre D. Après 16 décalages, le reste se trouve dans le registre R, et le quotient dans le registre Q.

R ← C ← Q ← 1 si $R \geq \text{diviseur}$ sinon 0		
R	Q	décalage no.
0000000000000000	0111101010111000	0
0000000000000000	1111010101110000	1
0000000000000001	1110101011100000	2
0000000000000011	1101010110000000	3
0000000000000111	1010101110000000	4
0000000000001111	0101011100000000	5
0000000000011110	1010111000000000	6
$R \geq D$	<u>-11001</u>	
0000000000000101		
0000000000001011	0101110000000001 ← 1	7
0000000000010110	1011100000000010	8
0000000000101101	0111000000000100	9
$R \geq D$	<u>-11001</u>	
0000000000010100		
0000000000101000	1110000000001001 ← 1	10
$R \geq D$	<u>-11001</u>	
000000000001111		
0000000000011111	1100000000010011 ← 1	11
$R \geq D$	<u>-11001</u>	
000000000000110		
000000000001101	1000000000100111 ← 1	12
0000000000011011	0000000001001110	13
$R \geq D$	<u>-11001</u>	
000000000000010		
000000000000100	0000000010011101 ← 1	14
0000000000001000	0000000100111010	15
0000000000010000	0000001001110100	16
reste	0000010011101000	17
	quotient	

Changements de base

On suppose que les nombres sont représentés par l'expression de la page 34 du livre :

$$N = \dots n_3 B^3 + n_2 B^2 + n_1 B^1 + n_0 B^0, \quad [1]$$

où B , appelé la base, est un entier >1 , et où chaque n_i peut prendre les valeurs $0, 1, \dots, B-1$

Remarquer que la représentation d'un nombre dans sa propre base est toujours 10. Observer aussi que le décalage d'un nombre vers la gauche (en ajoutant un 0 à droite) équivaut toujours à la multiplication du nombre par sa base. Ainsi, $110 = 11 \times 10$, quelle que soit la base. De la même façon, un décalage vers la droite (enlever le dernier chiffre de droite) équivaut à une division du nombre par sa base. Ainsi, $100 \div 10 = 10$, quelle que soit la base.

Passage d'une base a à une base b en arithmétique de base a

Pour convertir un nombre N d'une base a à une base b en utilisant l'arithmétique de base a , on divise N par b de façon répétitive en arithmétique de base a , jusqu'à ce que le quotient soit nul. La représentation de N dans la base b est donnée par la séquence renversée des restes.

Par exemple, à la p. 35 du livre de Zanella et Ligier, on convertit le nombre 25 du décimal au binaire en le divisant de façon répétée par 2 en arithmétique de base 10.

Comme autre exemple, pour passer de la base 10 à la base 4 ($a = 10, b = 4$) :

$$\begin{array}{lcl}
 (27)_{10} & \rightarrow & 27 / 4 = 6, \text{ reste } 3 \\
 & \downarrow & \\
 & 6 / 4 = 1, \text{ reste } 2 & \\
 & \downarrow & \\
 & 1 / 4 = 0, \text{ reste } 1 & \quad (27)_{10} = (123)_4
 \end{array}$$

Passage de la base 10 à la base 4

Passage d'une base a à une base b en arithmétique de base b

Pour passer d'une base a à une base b en arithmétique de base b , on évalue la série [1] avec $B = a$, en arithmétique de base b .

Par exemple : $(123)_4 = (1 \times 4^2) + (2 \times 4^1) + (3 \times 4^0) = (16 + 8 + 3)_{10} = (27)_{10}$.

On peut évaluer l'équation [1] plus efficacement à l'aide de la factorisation de Horner :

$$N = (((((n_3 \times B) + n_2) \times B) + n_1) \times B) + n_0. \quad [2]$$

En particulier, pour passer du **décimal au binaire**, la méthode la plus courante consiste à évaluer la série [1] avec $B = 10$ en arithmétique de base 2, à partir de la représentation BCD du nombre décimal. Un algorithme effectuant cette opération consiste à poser d'abord $R = 0$, puis, en partant du digit le plus significatif, à remplacer R par la valeur du digit suivant plus dix $\times R$ (évalué en binaire), et ainsi de suite pour chacun des digits :

$(125)_{10}$ $\downarrow$ $\downarrow$ $\downarrow$	R 0 $\downarrow$ $\downarrow$ $\downarrow$	R $(0000)_2$ $\downarrow$ $\downarrow$ $\downarrow$
$1 + 10 \times 0 = 1$	$0001 + 1010 \times 0000 = (0001)_2$	
$2 + 10 \times 1 = 12$	$0010 + 1010 \times 0001 = (1100)_2$	
$5 + 10 \times 12 = 125$	$0101 + 1010 \times 1100 = (1111101)_2$	

Conversion décimal-binaire

Remarquez que les multiplications par dix peuvent être réalisées en n'effectuant que des décalages et des additions : en effet, $10x = 8x + 2x$; $2x$ est obtenu par $x + x$ ou encore par un décalage à gauche de x , et $8x$ est obtenu par deux décalages à gauche de $2x$.

Avec cette même méthode, pour passer du **binaire au décimal** (en fait, du binaire au BCD), on procède à l'évaluation de la série [1] avec $B = 2$ en arithmétique de base 10, en utilisant la factorisation de Horner [2]. On pose d'abord $R = 0$, puis $R =$ valeur du bit le plus significatif suivant de $N + (2 \times \text{la valeur du } R \text{ précédent})$, cette somme étant évaluée en arithmétique BCD (voir section 3.3.4 du livre), jusqu'à épuisement des bits de N :

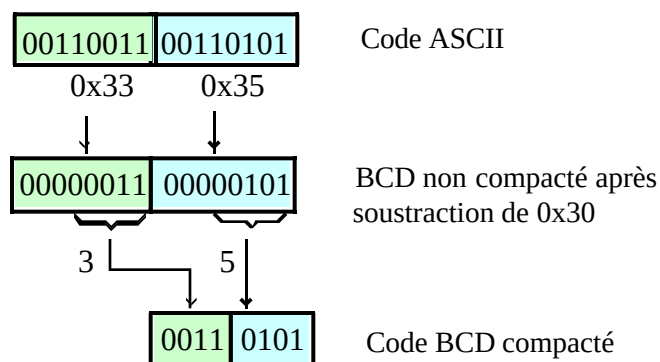
$(10110)_2$ $\downarrow$ $\downarrow$ $\downarrow$ $\downarrow$ $\downarrow$	R 0 $\downarrow$ $\downarrow$ $\downarrow$ $\downarrow$ $\downarrow$
$1 + 0 + 0 = 1$	
$0 + 1 + 1 = 2$	
$1 + 2 + 2 = 5$	
$1 + 5 + 5 = 11$	
$0 + 11 + 11 = 22_{10}$	

Conversion binaire-décimal

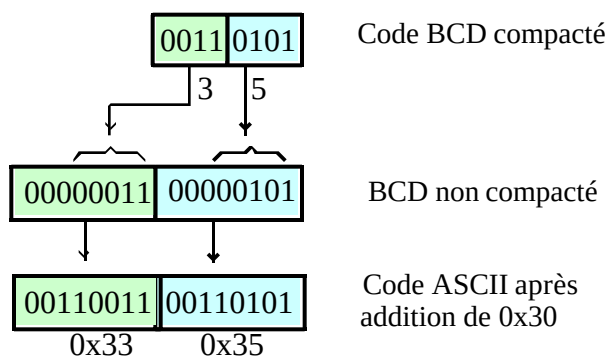
Les conversions binaire-décimal et décimal-binaire peuvent également être effectuées très aisément au moyen d'une table (voir Appendice 4).

Conversion avec l'ASCII

Pour convertir en binaire un nombre entré au clavier, donc composé de caractères numériques décimaux codés en ASCII, on soustrait 48_{10} (30_{16}) de chaque caractère. On obtient alors la représentation BCD du nombre. On utilise ensuite les nombres obtenus dans l'algorithme de conversion décimal-binaire :



Lors de la conversion binaire-décimal, après la conversion binaire-décimale, la représentation ASCII est obtenue en ajoutant 48_{10} ($0x30$) à chaque digit de la représentation BCD.



Notez en passant que certaines machines, pour éviter les conversions BCD-binaire et binaire-BCD, effectuent tous leurs calculs en arithmétique BCD. C'est le cas, en particulier, de certains calculateurs de poche.

3.3.3 Nombres fractionnaires

Un nombre réel en code pondéré dans une base B peut être exprimé comme suit :

$$N = \dots n_2 B^2 + n_1 B^1 + n_0 B^0 + n_{-1} B^{-1} + n_{-2} B^{-2} + \dots$$

En binaire on aura donc :

$$N = \dots n_2 2^2 + n_1 2^1 + n_0 2^0 + n_{-1} 2^{-1} + n_{-2} 2^{-2} + \dots,$$

$$\text{soit : } N = \dots n_2 \times 4 + n_1 \times 2 + n_0 + n_{-1} \times 0,5 + n_{-2} \times 0,25 + \dots,$$

où les n_i sont évidemment 0 ou 1.

Pour la conversion décimal-binaire d'un nombre avec une partie fractionnaire, par exemple 128,8965, on commence par convertir la partie entière, exactement comme d'habitude. Pour la partie fractionnaire, on procède comme suit : on multiplie par 2 la partie fractionnaire, on prend la partie entière du résultat (qui ne peut être que 0 ou 1) comme résultat binaire, et on recommence jusqu'à épuisement des chiffres fractionnaires du nombre décimal :

$$\begin{array}{r}
 0,8965 \times 2 \\
 \underline{1} \quad 0,7930 \times 2 \\
 \underline{1} \quad 0,5860 \times 2 \\
 \underline{1} \quad 0,1720 \times 2 \\
 \underline{0} \quad 0,3440 \times 2 \\
 \underline{0} \quad 0,6880 \times 2 \\
 \underline{1} \quad 0,3760 \times 2 \\
 \underline{0} \quad 0,7520 \times 2 \\
 \underline{1} \quad 0,5040 \times 2 \\
 \underline{1} \quad 0,0080 \times 2 \dots \text{etc.,}
 \end{array}$$

d'où: $128,8965_{10} = 10000000,111001011\dots_2$

Conversion décimal binaire d'une partie fractionnaire

Pour la conversion binaire-décimal, on peut procéder exactement comme d'habitude pour la

partie entière. Pour la partie fractionnaire, on décale toute la fraction à gauche du point décimal. À chaque décalage, on incrémente n , une puissance de 2. À la fin, il suffit de diviser l'entier qui en résulte par 2^n pour obtenir la fraction décimale cherchée.

Exemple : $10000000,111001011_2$. La conversion de la partie entière donne 128. On traite ensuite la partie fractionnaire : $0,111001011 = 111001011 / 2^9$. Comme $2^9 = 512_{10}$ et que $111001011_2 = 459_{10}$, on obtient finalement comme partie fractionnaire : $459 \div 512 = 0,896484375$.

Standard IEEE 754

Le standard IEEE 754 (Institute of Electronic and Electrical Engineers) de simple précision est le suivant : le premier bit (s) est le signe du nombre; suivent ensuite 8 bits représentant e , l'exposant en code excédant 127. Dans ce code, 127 représente 0, tout nombre inférieur à 127 est considéré comme négatif, et tout nombre entre 127 et 255 est positif. On peut ainsi exprimer des exposants allant de -126 (00000001) à +127 (11111110). Les exposants -127 et +128 sont réservés aux représentations de 0 et de ∞ respectivement. Les 23 bits suivants représentent la partie fractionnaire f de la mantisse normalisée. Il s'agit d'une mantisse normalisée m de la forme

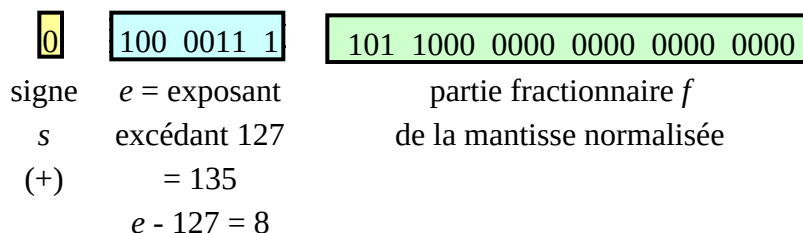
$$m = 1,f \quad \text{telle que } 1 \leq m < 2.$$

Comme le premier bit d'une telle mantisse normalisée est toujours 1, on n'a pas besoin de l'écrire, ce qui permet de gagner un bit et d'atteindre une précision de 24 bits. Cette représentation nécessite donc un total de 32 bits, soit 4 octets. Ceci représente une précision décimale de l'ordre de 7 chiffres significatifs et peut exprimer des nombres allant d'environ 10^{-38} à 10^{+38} .

La valeur du nombre est donc donnée par l'expression :

$$N = (-1)^s \times 2^{(e - 127)} \times 1,f$$

où s est le signe de la mantisse, et e est l'exposant tel qu'inscrit dans le format suivant :



Représentation de 432 en virgule flottante IEEE de simple précision

Le nombre $432 = (110110000)_2 = 1,10110000 \times 2^8$ s'écrit donc en simple précision : $43D80000_{IEEE}$. On utilise ici la notation hexadécimale comme abréviation de la représentation binaire du nombre IEEE. On met l'indice IEEE pour bien indiquer qu'il ne s'agit pas d'un nombre hexadécimal.

Exemples :

$$\begin{aligned}
 +0 &= 00000000_{IEEE}, \\
 -0 &= 80000000_{IEEE}, \\
 +1 &= 1,0 \times 2^0 = 3F800000_{IEEE}, \\
 +2 &= 1,0 \times 2^1 = 40000000_{IEEE}, \\
 +\infty &= 1,0 \times 2^{128} = 7F800000_{IEEE} \text{ et} \\
 -\infty &= FF800000_{IEEE}.
 \end{aligned}$$

En double précision, on utilise 8 octets, soit 64 bits, dont 11 servent à l'exposant (en code excédant 1023) et 52 à la partie fractionnaire de la mantisse. Il en résulte une précision équivalant à 15 chiffres significatifs en décimal et des exposants pouvant aller de 10^{-308} à 10^{+307} .

La valeur d'un nombre N en double précision est donc donnée par l'expression :

$$N = (-1)^s \times 2^{(e-1023)} \times 1,f$$

En précision étendue, on utilise 10 octets, soit 80 bits, dont 15 servent à l'exposant (en code excédant 16 384) et 63 à la partie fractionnaire de la mantisse. Il en résulte une précision équivalant à 19 chiffres significatifs en décimal et des exposants pouvant aller de 10^{-4933} à 10^{+4931} .

La valeur d'un nombre N en précision étendue est donc donnée par l'expression :

$$N = (-1)^s \times 2^{(e-16383)} \times 1,f$$

Multiplication et division

La multiplication et la division en virgule flottante s'effectuent sans difficulté : on multiplie les mantisses entre elles et on additionne les exposants. En effet :

$$(a \times 2^p) \times (b \times 2^q) = a \times b \times 2^{p+q}.$$

On normalise ensuite la mantisse résultante et on donne un signe négatif au résultat si les deux opérandes ne sont pas de même signe. On procède de la même façon pour la division puisque $(a \times 2^p) \div (b \times 2^q) = (a \div b) \times 2^{p-q}$.

Addition et soustraction

L'addition et la soustraction sont plus délicates. Avant d'effectuer l'opération, il faut aligner les deux nombres, c.-à-d. les ramener au même exposant, celui du nombre le plus grand en valeur absolue. Ainsi, si $A = a \times 2^p$ et $B = b \times 2^q$, alors pour effectuer $A + B$, si q est inférieur à p , il faut décaler b vers la droite tout en augmentant q jusqu'à ce que les deux exposants soient égaux. On effectue ensuite l'addition des mantisses et on normalise la mantisse résultante s'il y a lieu.

Exemple :

Soit $A = 1,1101011_2 \times 2^3$ et $B = 1,10010110_2 \times 2^1$. Puisque $q < p$, on effectue la correction $B = 0,0110010110_2 \times 2^3$ avant d'effectuer l'addition : $A + B = (1,1101011000_2 + 0,0110010110_2) \times 2^3$.

En simple précision, si $|p-q| > 25$, il n'est pas nécessaire d'effectuer le calcul, car l'un des opérandes est tellement petit par rapport à l'autre qu'on peut le négliger.

Conversion

Pour effectuer la conversion d'un nombre décimal tel que $3,65625 \times 10^3$ à la représentation en virgule flottante IEEE, on trouve d'abord la représentation de la mantisse 3,65625 en binaire, soit $1,110101_2 \times 2^1$. Ceci donne, en format de virgule flottante IEEE : $406A0000_{\text{IEEE}}$. On multiplie ensuite cette valeur par la représentation de 10^3 en virgule flottante IEEE, qui est $447A0000_{\text{IEEE}}$. Le produit donne finalement 45648400_{IEEE} . On peut calculer la valeur de 10^3 directement ou par multiplications successives de 10,0 en virgule flottante ou encore la prendre dans une table.

Les calculs en virgule flottante présentent des problèmes d'arrondi dans de nombreux cas. Par exemple, le nombre $0,1_{10}$ ne s'exprime pas de façon finie en virgule flottante binaire et donne

$0,00011001100110011..._2$, ou, en virgule flottante IEEE de simple précision : $3DCCCCC_{IEEE}$. Pour éviter que la conversion inverse ne donne $0,099999....$, on arrondit à $3DCCCCD_{IEEE}$.

La conversion inverse s'effectue d'une façon similaire : on transforme l'exposant binaire en nombre décimal à l'aide d'une table; on convertit ensuite la mantisse binaire en décimal par la méthode habituelle de conversion binaire-décimal pour les entiers; on effectue enfin le produit de ces deux quantités. Dans l'exemple de la page précédente $43D80000_{IEEE}$, l'exposant est 8, et $2^8 = 256_{10}$. La mantisse binaire est $1,1011_2$, soit $1,6875_{10}$. Le produit des deux, soit $256 \times 1,6875 = 432$ donne le résultat décimal.

NaN et nombres dénormalisés

Dans la norme IEEE, les nombres avec exposant $0xFF$ (+128) et fraction f non nulle sont appelés NaN (Not a Number). Ils peuvent être utilisés comme messages d'erreur de la part du coprocesseur arithmétique, dépendant de la valeur de f .

Les nombres avec exposant nul (-127) et fraction f non nulle représentent des nombres dits "dénormalisés", dont la valeur est donnée par $0,f \times 2^{-126}$, mais avec une précision réduite.

Exemple : $00100000_{IEEE} = 0,001_2 \times 2^{-126} = 1,4693679 \times 10^{-39}$.

Le plus petit nombre dénormalisé est donc : $00000001_{IEEE} = 2^{-23} \times 2^{-126} = 1,4012985 \times 10^{-45}$.

Unité de calcul en virgule flottante

Les calculs en virgule flottante sont assez complexes et ont tendance à être lents s'ils sont programmés de façon conventionnelle avec des processeurs optimisés pour les calculs sur des entiers. Les micro-ordinateurs modernes utilisent donc une unité spécialisée appelée **FPU** (Floating-Point Unit) qui exécute ces calculs très rapidement.

Exercices du chapitre 3

3.1 Coder le message suivant en ASCII :

COPY A : B :

Il y a deux espaces dans cette ligne qui se termine par un retour de chariot.

3.2 Mettre le chiffre binaire suivant sous forme octale, puis sous forme hexadécimale :

1 1 0 0 1 0 1 1 0 0 1 1 1 0 1 0

3.3 Peut-on mettre facilement le chiffre du problème précédent sous forme décimale sans effectuer aucun calcul ?

3.4 Déterminer la valeur en décimal du chiffre du problème 3.2 en utilisant la formule suivante, où m est le nombre de bits :

$$N = n_{m-1}B^{m-1} + n_{m-2}B^{m-2} + \dots + n_0B^0$$

3.5 a. Convertir 123_4 en base 10.

b. Convertir 57_{10} en base 4.

3.6 En tenant compte du fait que les opérations de décalage et d'addition binaires sont plus rapides en langage machine, expliquer comment on peut multiplier un entier quelconque par 21_{10} . Multiplier le nombre hexadécimal B7F par 21_{10} .

3.7 En s'inspirant de la table d'addition binaire donnée à la p. 109 du livre, élaborer une table de soustraction.

3.8 a. Quelle est l'étendue de représentation d'un mot de 8 bits pour des entiers positifs?

b. Quelle est l'étendue de représentation d'un mot de 8 bits pour des entiers signés?

3.9 Multiplier 10101_2 par 110_2 .

3.10 Diviser 100101_2 par 1101_2 .

3.11 a. Convertir 0.101101_2 en décimal.

b. Convertir 0.484375_{10} en binaire

- 3.12** a. Convertir le nombre hexadécimal 5B7.A34F en format IEEE de simple précision et fournir le résultat sous forme de 8 digits hexadécimaux.
 b. Déterminer la valeur en décimal du nombre IEEE suivant : 7A5F43B6. S'arranger pour avoir une mantisse décimale comprise entre 1 et 10.
 c. Déterminer l'erreur maximale de troncature, en valeur relative, que l'on peut commettre en convertissant un nombre dans le format IEEE correspondant.

3.13 Soit les deux nombres IEEE de simple précision en notation hexadécimale :

$$A = 43D00000_{\text{IEEE}} \quad B = 3FF00000_{\text{IEEE}}$$

- a. Trouver la notation binaire de A et B.
 b. Isoler les caractéristiques (signe, exposant, mantisse) de A et B. Mettre ensuite A et B sous la forme :
 $A = 1,xxx \times 2^m$ et $B = 1,yyy \times 2^n$
 c. Décaler le chiffre le plus petit pour obtenir la même puissance de 2.
 d. Additionner les mantisses en binaire.
 e. Reconvertir le résultat dans sa forme IEEE standard puis donner le résultat sous la forme hexadécimale.

3.14 Effectuer le produit des deux nombres en virgule flottante IEEE de simple précision : $40A00000_{\text{IEEE}} \times C0400000_{\text{IEEE}}$.

3.15 Écrire le nombre 1,0 en virgule flottante IEEE de double précision.

3.16 Dans les ordinateurs IBM 370, les nombres en virgule flottante sont exprimés de la façon suivante :

1 bit de signe de la mantisse, exposant de 7 bits + 64 en puissances de 16, fraction de 24 bits. Dans ce système, la fraction est dite normalisée s'il n'est pas possible de réduire l'exposant sans que la mantisse atteigne ou dépasse 1,0.

Exemples : $15_{10} = 1111_2 \times 16^0 = 0,1111_2 \times 16^1$; exposant corrigé = $64 + 1 = 65$;

→ 0 1000001 111100000000000000000000.

$1,0_{10} = 1,000_2 \times 16^0 = 0,0001_2 \times 16^1$; exposant corrigé = $64 + 1 = 65$;

→ 0 1000001 000100000000000000000000.

- a. Exprimer le nombre 432_{10} dans ce système.
 b. Quel inconvénient ce système présente-t-il en ce qui a trait à la précision relative des nombres exprimés?
 c. Quel sont le plus petit et le plus grand nombres normalisés représentables dans ce système?

Chapitre 5

Circuits logiques

5.1 Circuits combinatoires

Quand on a un problème de logique combinatoire à résoudre, on part de la table de vérité, qui est la donnée du problème. Par exemple:

a	b	c	x
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

On peut ensuite:

1. Soit écrire le minterme de chacune des rangées pour laquelle le résultat doit être 1 et faire le OU de tous ces mintermes. On obtient alors:

$$x = \bar{a}.b.\bar{c} + \bar{a}.b.c + a.b.\bar{c} + a.b.c$$

qui signifie $x = 1$ si $a = 0$ et $b = 1$ et $c = 0$, ou si $a = 0$ et $b = 1$ et $c = 1$, ou si ... Il reste à simplifier cette équation en utilisant les règles de l'algèbre booléenne. On obtient finalement $x = b$.

- b. Soit utiliser une table de Karnaugh. On part encore de la table de vérité et on transcrit les 1 dans les cases appropriées. On encercle les 1 pour obtenir:

		c	
		0	1
ab	00	0	0
	01	1	1
	11	1	1
	10	0	0

On voit immédiatement que la boucle ne peut dépendre ni de a ni de c puisqu'elle vaut 1 que a soit 0 ou 1 et que c soit 0 ou 1. Par contre, b est toujours 1. Donc cette boucle vaut b . Comme c 'est la seule, alors $x = b$.

Cette technique est une façon visuelle de détecter et de simplifier les termes tels que $a.b + a.\bar{b} = a$. Observez l'équivalence entre les deux méthodes:

la seconde rangée représente $\bar{a}.b.\bar{c} + \bar{a}.b.c = \bar{a}.b$, tandis que la troisième rangée représente $a.b.\bar{c} + a.b.c = a.b$. L'ensemble se simplifie ainsi: $\bar{a}.b + a.b = b$.

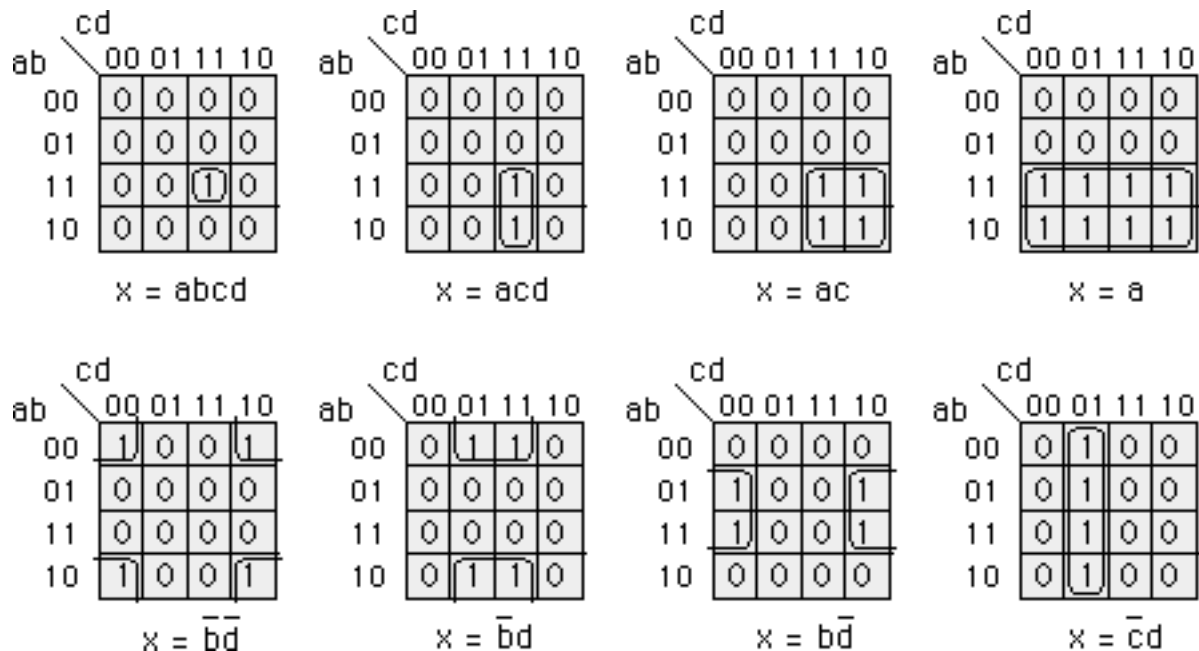
Exemples de tables de Karnaugh

$\begin{array}{c cc} & b & \\ \hline a & & \\ \hline 0 & 0 & 0 \\ 1 & 0 & 1 \end{array}$ $x = ab$	$\begin{array}{c cc} & b & \\ \hline a & & \\ \hline 0 & 0 & 0 \\ 1 & 1 & 0 \end{array}$ $x = a\bar{b}$	$\begin{array}{c cc} & b & \\ \hline a & & \\ \hline 0 & 0 & 1 \\ 1 & 0 & 0 \end{array}$ $x = \bar{a}b$	$\begin{array}{c cc} & b & \\ \hline a & & \\ \hline 0 & 1 & 0 \\ 1 & 0 & 0 \end{array}$ $x = \bar{a}\bar{b}$
$\begin{array}{c cc} & b & \\ \hline a & & \\ \hline 0 & 0 & 0 \\ 1 & 1 & 1 \end{array}$ $x = a$	$\begin{array}{c cc} & b & \\ \hline a & & \\ \hline 0 & 0 & 1 \\ 1 & 0 & 1 \end{array}$ $x = b$	$\begin{array}{c cc} & b & \\ \hline a & & \\ \hline 0 & 1 & 1 \\ 1 & 0 & 0 \end{array}$ $x = \bar{a}$	$\begin{array}{c cc} & b & \\ \hline a & & \\ \hline 0 & 1 & 0 \\ 1 & 1 & 0 \end{array}$ $x = \bar{b}$
$\begin{array}{c cc} & b & \\ \hline a & & \\ \hline 0 & 0 & 1 \\ 1 & 1 & 1 \end{array}$ $x = a + b$	$\begin{array}{c cc} & b & \\ \hline a & & \\ \hline 0 & 1 & 1 \\ 1 & 1 & 1 \end{array}$ $x = 1$	$\begin{array}{c cc} & b & \\ \hline a & & \\ \hline 0 & 1 & 0 \\ 1 & 0 & 1 \end{array}$ $x = \bar{a}\bar{b} + ab$	$\begin{array}{c cc} & b & \\ \hline a & & \\ \hline 0 & 0 & 1 \\ 1 & 1 & 0 \end{array}$ $x = a\bar{b} + \bar{a}b$

Tables de Karnaugh à 2 variables

$\begin{array}{c cc} & c & \\ \hline ab & 0 & 1 \\ \hline 00 & 0 & 0 \\ 01 & 0 & 0 \\ 11 & 0 & 1 \\ 10 & 0 & 0 \end{array}$ $x = abc$	$\begin{array}{c cc} & c & \\ \hline ab & 0 & 1 \\ \hline 00 & 1 & 0 \\ 01 & 0 & 0 \\ 11 & 0 & 0 \\ 10 & 0 & 0 \end{array}$ $x = \bar{a}\bar{b}\bar{c}$	$\begin{array}{c cc} & c & \\ \hline ab & 0 & 1 \\ \hline 00 & 0 & 0 \\ 01 & 0 & 0 \\ 11 & 0 & 0 \\ 10 & 1 & 0 \end{array}$ $x = a\bar{b}\bar{c}$	$\begin{array}{c cc} & c & \\ \hline ab & 0 & 1 \\ \hline 00 & 0 & 1 \\ 01 & 0 & 0 \\ 11 & 0 & 0 \\ 10 & 0 & 0 \end{array}$ $x = \bar{a}\bar{b}c$
$\begin{array}{c cc} & c & \\ \hline ab & 0 & 1 \\ \hline 00 & 0 & 0 \\ 01 & 0 & 0 \\ 11 & 0 & 1 \\ 10 & 0 & 1 \end{array}$ $x = ac$	$\begin{array}{c cc} & c & \\ \hline ab & 0 & 1 \\ \hline 00 & 0 & 1 \\ 01 & 0 & 1 \\ 11 & 0 & 0 \\ 10 & 0 & 0 \end{array}$ $x = \bar{a}c$	$\begin{array}{c cc} & c & \\ \hline ab & 0 & 1 \\ \hline 00 & 0 & 1 \\ 01 & 0 & 0 \\ 11 & 0 & 0 \\ 10 & 0 & 1 \end{array}$ $x = \bar{b}c$	$\begin{array}{c cc} & c & \\ \hline ab & 0 & 1 \\ \hline 00 & 0 & 0 \\ 01 & 1 & 0 \\ 11 & 1 & 0 \\ 10 & 0 & 0 \end{array}$ $x = bc$
$\begin{array}{c cc} & c & \\ \hline ab & 0 & 1 \\ \hline 00 & 0 & 0 \\ 01 & 0 & 0 \\ 11 & 1 & 1 \\ 10 & 1 & 1 \end{array}$ $x = a$	$\begin{array}{c cc} & c & \\ \hline ab & 0 & 1 \\ \hline 00 & 0 & 0 \\ 01 & 1 & 1 \\ 11 & 1 & 1 \\ 10 & 0 & 0 \end{array}$ $x = b$	$\begin{array}{c cc} & c & \\ \hline ab & 0 & 1 \\ \hline 00 & 0 & 1 \\ 01 & 0 & 1 \\ 11 & 0 & 1 \\ 10 & 0 & 1 \end{array}$ $x = c$	$\begin{array}{c cc} & c & \\ \hline ab & 0 & 1 \\ \hline 00 & 1 & 1 \\ 01 & 0 & 0 \\ 11 & 0 & 0 \\ 10 & 1 & 1 \end{array}$ $x = \bar{b}$
$\begin{array}{c cc} & c & \\ \hline ab & 0 & 1 \\ \hline 00 & 1 & 1 \\ 01 & 1 & 1 \\ 11 & 0 & 0 \\ 10 & 0 & 0 \end{array}$ $x = \bar{a}$	$\begin{array}{c cc} & c & \\ \hline ab & 0 & 1 \\ \hline 00 & 0 & 0 \\ 01 & 1 & 1 \\ 11 & 1 & 1 \\ 10 & 1 & 1 \end{array}$ $x = a + b$	$\begin{array}{c cc} & c & \\ \hline ab & 0 & 1 \\ \hline 00 & 0 & 1 \\ 01 & 0 & 1 \\ 11 & 1 & 1 \\ 10 & 1 & 1 \end{array}$ $x = a + c$	$\begin{array}{c cc} & c & \\ \hline ab & 0 & 1 \\ \hline 00 & 1 & 1 \\ 01 & 1 & 0 \\ 11 & 1 & 0 \\ 10 & 1 & 1 \end{array}$ $x = \bar{b} + \bar{c}$

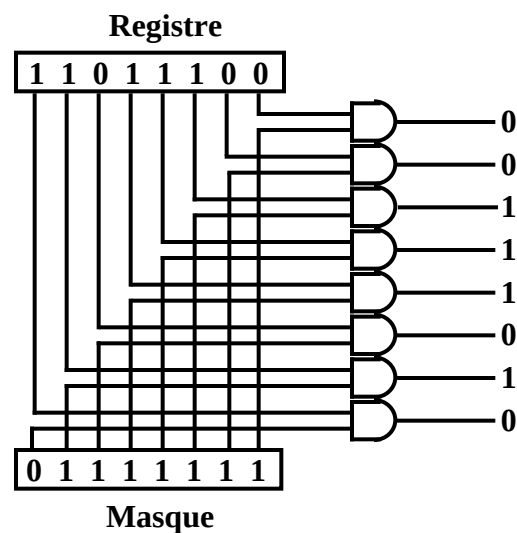
Tables de Karnaugh à 3 variables



Tables de Karnaugh à 4 variables

Masquage d'un registre

Une application importante utilisée par tous les processeurs est le masquage d'un registre. Soit un registre contenant initialement le motif suivant : 1101 1100₂. Supposons qu'on désire mettre à 0 le 8^e bit en conservant tous les autres bits intacts. On effectue alors un ET logique bit par bit de ce registre avec le **masque** 0x7F (0111 1111) :



Masquage d'un registre

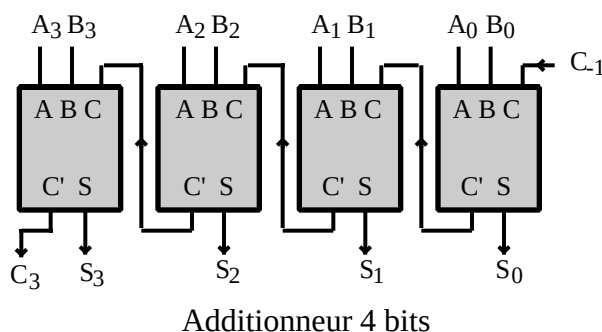
La mise à 1 des bits d'un registre est réalisée d'une façon analogue : il suffit de faire le OU logique bit à bit. Pour faire l'inverse du circuit précédent, c'est-à-dire mettre à 1 le 8^e bit :

Registre initial :	0101 1100
Masque :	1000 0000
Résultat :	1101 1100

5.2.4 Synthèse d'un circuit combinatoire

Additionneurs

Aux pages 82, 105 et 109 du livre, il est question d'additionneurs binaires. On peut utiliser de tels additionneurs de 1 bit pour effectuer l'addition de mots de plusieurs bits. Voici le circuit pour un additionneur de 4 bits :



Cette technique est appelée addition avec retenue série. Elle est malheureusement trop lente, car les S_i ne sont tous valides que quand l'effet de la retenue C_{-1} s'est propagé jusqu'à C_3 . Pour des longs mots, le délai devient excessif. On utilise alors des circuits de **retenue anticipée** (look-ahead carry), beaucoup plus rapides, mais aussi beaucoup plus complexes.

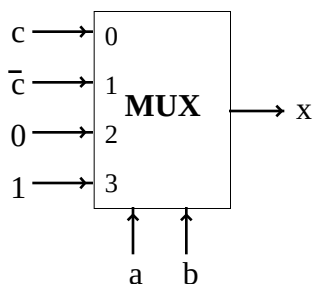
5.2.6 Multiplexeurs et démultiplexeurs

À la page 85 du livre, on mentionne qu'on peut réaliser n'importe quelle fonction de deux variables avec un multiplexeur à deux variables. Ceci peut être généralisé à un nombre quelconque de variables.

Soit la table de vérité suivante :

a	b	c	x
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	1

On peut la réaliser avec le circuit suivant :



Réalisation d'un circuit logique au moyen d'un multiplexeur

En effet, si on considère les deux premières rangées de la table, $a = 0$ et $b = 0$ choisissent l'entrée 0 du multiplexeur. Pour ces deux rangées, la table de vérité montre que $x = c$. On applique donc c à l'entrée 0.

De la même façon, pour les deux rangées suivantes, $a = 0$ et $b = 1$ choisissent l'entrée 1 du multiplexeur. Pour ces deux rangées, la table de vérité montre que $x = \bar{c}$. On applique donc $\bar{c}$ à l'entrée 1.

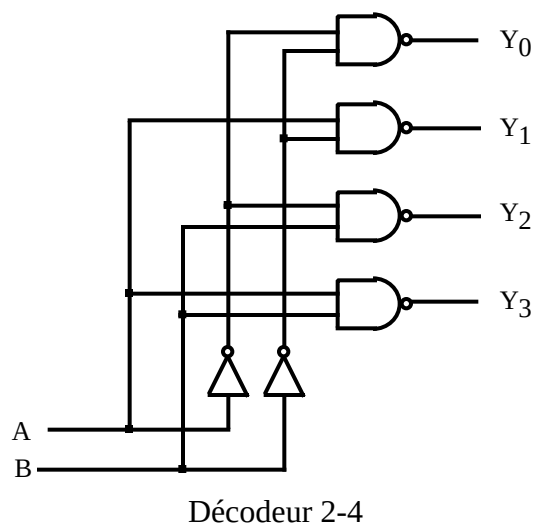
Pour les deux rangées suivantes, $a = 1$ et $b = 0$ choisissent l'entrée 2 du multiplexeur. Pour ces deux rangées, la table de vérité montre que x est toujours 0. Il suffit donc d'appliquer 0 à l'entrée 2.

Finalement, pour les deux rangées suivantes, $a = 1$ et $b = 1$, c'est l'entrée 3 du multiplexeur qui est choisie. Pour ces deux rangées, x est toujours 1. On applique donc 1 à l'entrée 3.

De la même façon, on peut réaliser une fonction de 4 variables avec un multiplexeur 8 vers 1, etc.

5.2.7 Décodeurs, codeurs, transcodeurs

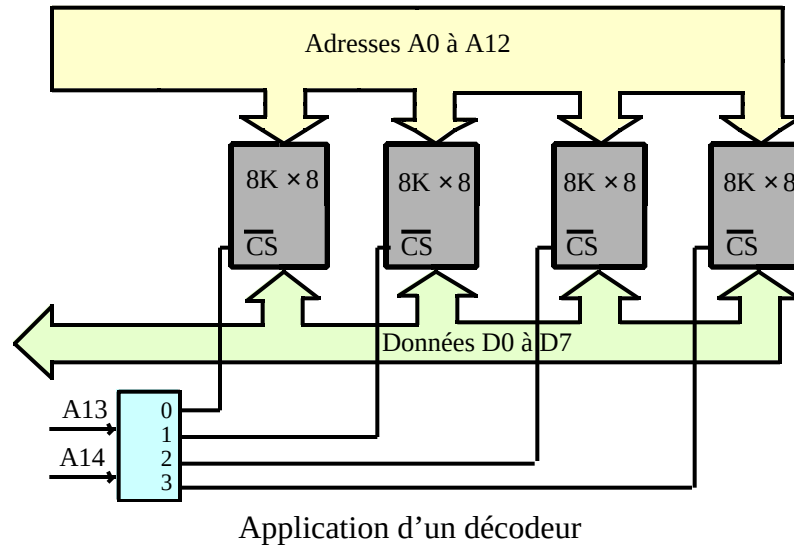
Un autre exemple de circuit MSI est un **décodeur**, circuit qui prend une entrée de n bits et l'utilise pour choisir l'une des lignes de sortie :



On remarque que les portes sont des NON-ET, de sorte que les sorties sont normalement 1, et qu'une seule peut être 0 à la fois.

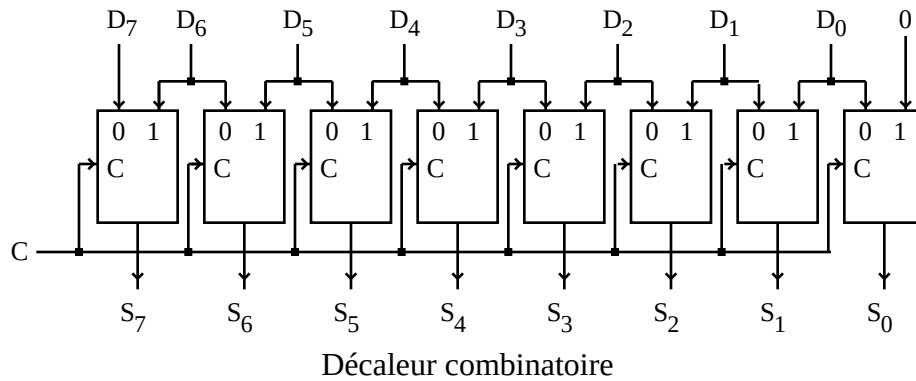
Le décodeur illustré ici est désigné décodeur 2-4 (2 vers 4). On trouve sur le marché des décodeurs 3-8, 4-16, double 2-4, et des décodeurs 4-10 pour le décodage de nombres BCD.

Un tel circuit est utile dans l'activation des circuits de mémoire, par exemple. Supposons en effet une mémoire de 32 Ko (Kilo-octets) constituée de 4 puces contenant chacune 8 Ko. La puce 0 contient les données des adresses 0 à 8 191, la puce 1, celles des adresses 8 192 à 16 383, etc. Chaque puce de mémoire contenant 8 Ko nécessite 13 bits d'adresse ($2^{13} = 8\,192$). Toutefois, pour adresser un octet unique dans une mémoire de 32 Ko, il faut une adresse de 15 bits ($2^{15} = 32\,768$). On utilise donc les deux bits les plus significatifs de l'adresse pour déterminer laquelle des quatre puces de mémoire doit être activée, au moyen d'une entrée de sélection de puce $\overline{CS}$ (Chip Select). La barre d'inversion indique que la puce est active quand cette entrée est basse (0). Les 13 autres lignes d'adresses sont appliquées aux quatre puces simultanément. Comme une seule des sorties du décodeur peut être 0 à la fois, une seule des quatre puces de mémoire est activée à un instant donné :

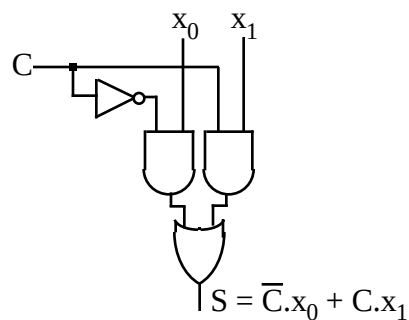


Décaleurs

Dans le circuit suivant, on applique un mot de 8 bits D_0 à D_7 et la sortie est décalée de 1 bit vers la gauche ou n'est pas décalée suivant la valeur d'une ligne de contrôle C . Lorsqu'il y a décalage à gauche, le bit de droite est remplacé par un 0 :

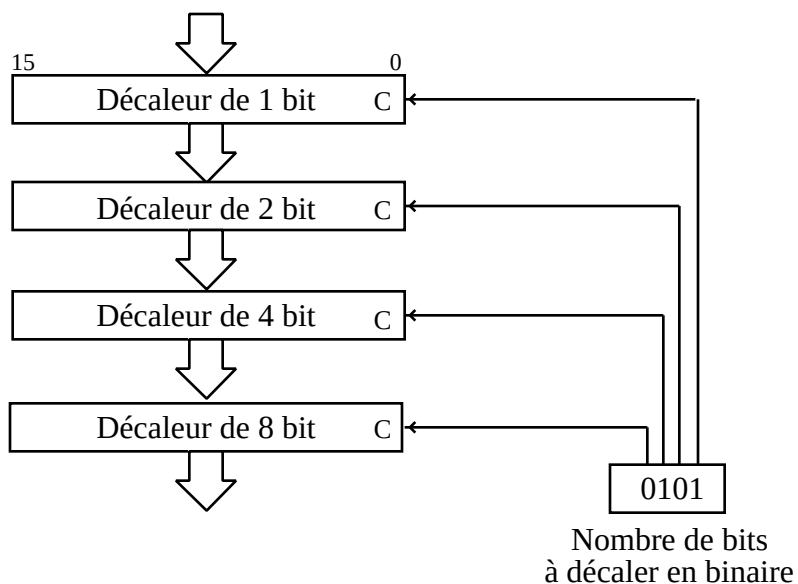


Ce circuit est constitué de huit multiplexeurs 2 vers 1 comme suit :



Lorsque $C = 0$, les données D_0 à D_7 sont appliquées aux sorties S_0 à S_7 respectivement. Lorsque $C = 1$, $S_0 = 0$, $S_1 = D_0$, $S_2 = D_1$, etc.

Les processeurs modernes utilisent un **décaleur à barillet** (barrel shifter), capable de décaler de n bits vers la gauche ou vers la droite. Pour réaliser un tel décaleur, on utilise des décaleurs combinatoires comme le précédent, capables de décaler de 1, 2, 4, 8 bits vers la gauche quand $C = 1$ et qui ne décalent pas quand $C = 0$. Le circuit suivant peut décaler des mots de 16 bits de 0 à 15 bits vers la gauche.



Décaleur à barillet

Dans l'exemple illustré dans la figure, on effectue un décalage à gauche de 5 bits.

Rotateurs

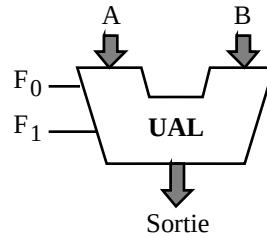
Un rotateur est un circuit similaire à celui d'un décaleur, mais qui applique à l'autre extrémité les bits qui débordent.



Rotateur

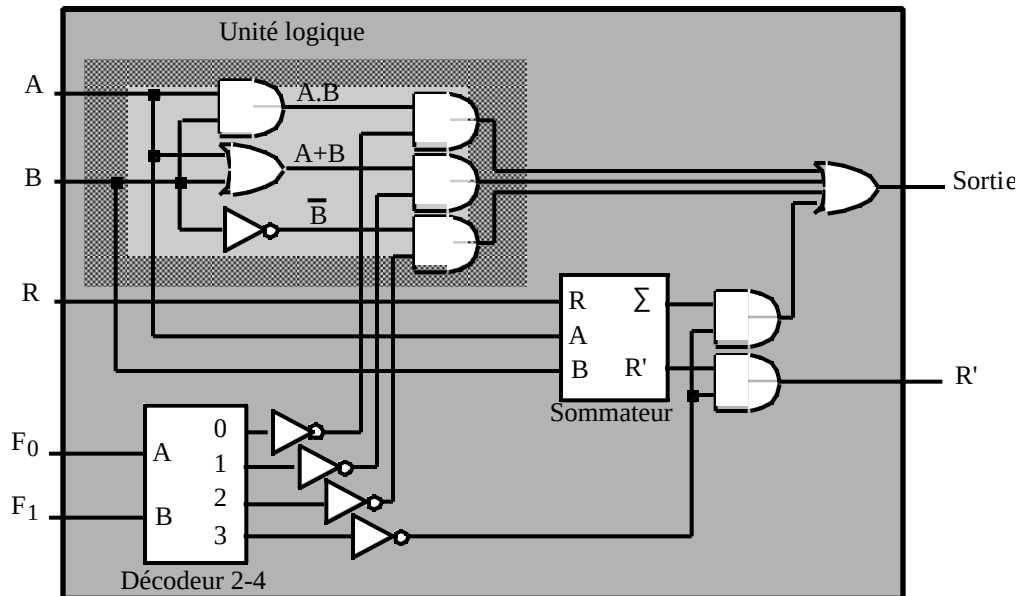
Ainsi, la rotation à gauche de 1 bit du nombre 10000100 donne 00001001, etc.

Unité arithmétique et logique (UAL)



Symbole d'UAL

La plupart des ordinateurs utilisent un seul circuit appelé UAL (unité arithmétique et logique) capable d'effectuer au moins les opérations de base ET, OU, l'addition binaire de deux mots et la négation d'un mot, suivant la valeur de deux variables de contrôle F_0 et F_1 :



UAL

F_1	F_0	Fonction
0	0	A ET B
0	1	A OU B
1	0	NON B
1	1	A plus B

Il s'agit ici d'une UAL de 1 bit (bit-slice) et on peut en combiner autant qu'on veut en parallèle. En principe, pour une machine ayant des mots de n bits, on pourrait utiliser n cir-

cuits identiques comme le précédent, un pour chaque bit. Cette UAL est élémentaire mais demeure l'élément central d'un ordinateur.

5.3 Circuits séquentiels

5.3.3 Bistables

Bistable RS

Une façon de visualiser le bistable RS de la figure 5.25 du livre est le système mécanique bistable suivant, qu'on peut réaliser facilement à l'aide d'une règle et d'un trombone. Une extrémité est étiquetée Q et on définit que $Q = 1$ si cette extrémité est en haut. L'autre extrémité est désignée $\bar{Q}$.

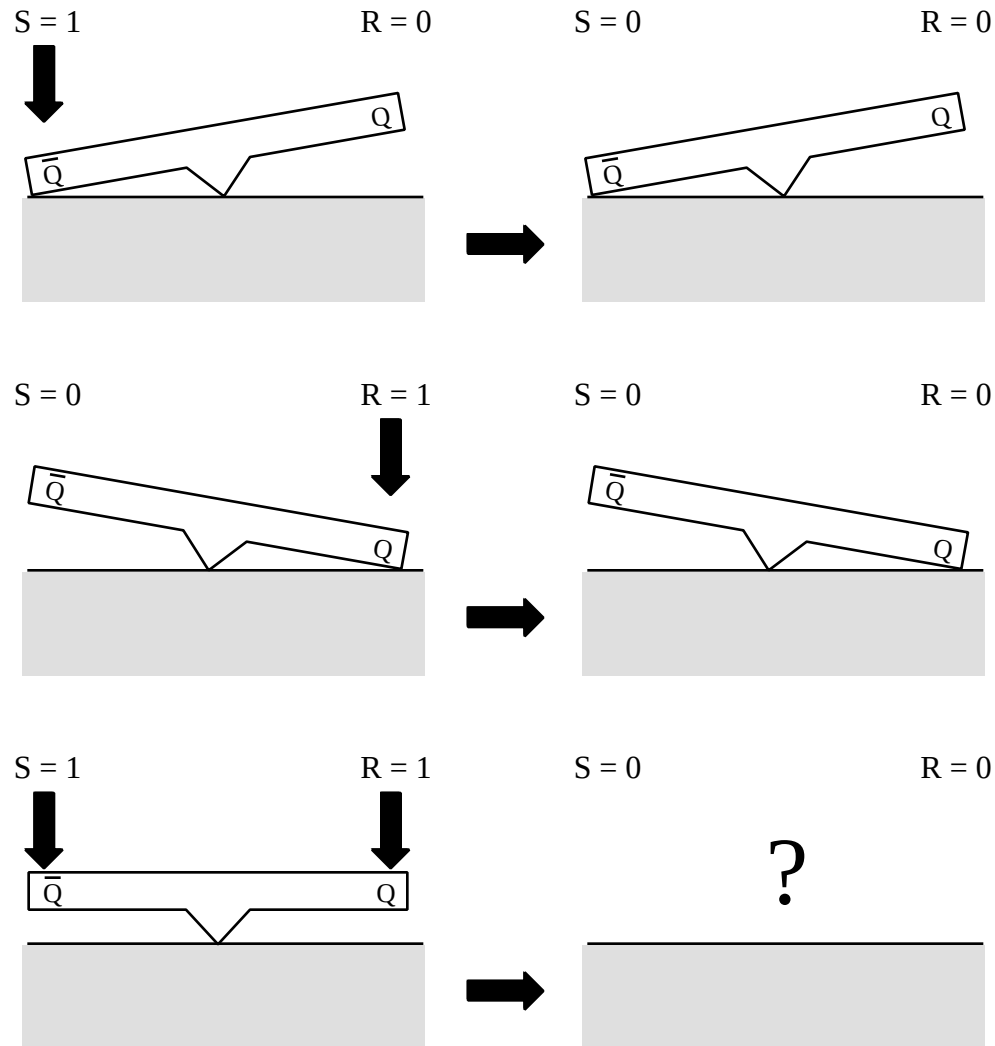
Si on applique une force S vers le bas à l'extrémité de gauche ($S = 1$), la règle bascule de façon à ce que $Q = 1$ (transition mise à un). Si on enlève la force S , alors $S = 0$ et $R = 0$, et la règle reste indéfiniment dans l'état $Q = 1$. Pour la faire basculer de l'autre côté, on doit appliquer une force R vers le bas à l'extrémité de droite ($R = 1$). À ce moment, on a $Q = 0$ et $\bar{Q} = 1$. Si on enlève la force R , alors $S = 0$ et $R = 0$, et la règle reste indéfiniment dans l'état $Q = 0$.

Si on applique les forces S et R simultanément ($S = 1$ et $R = 1$), le résultat est indéterminé, dans le sens que si on enlève S et R simultanément ($S = 0$ et $R = 0$), on ne peut décider à l'avance de quel côté la règle basculera. C'est exactement le cas du bistable RS. Par contre, si on enlève S ou R séparément, on passe nécessairement par une des situations $S = 0$ et $R = 1$ ou $S = 1$ et $R = 0$, et on sait de quel côté la règle ira.

Le comportement de ce système mécanique obéit à la même équation caractéristique que le circuit électronique :

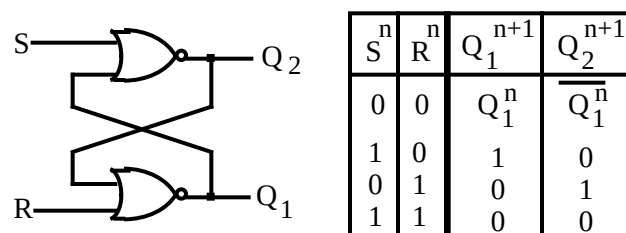
$$Q^+ = S + \bar{R} Q.$$

La valeur ultérieure Q^+ de Q sera 1 si $S = 1$, quelle que soit la valeur présente de Q , ou sera égale à la valeur présente de Q tant que $S = 0$ et $R = 0$. Le premier cas est la transition de mise à 1, l'autre est l'état stable.



Bistable RS mécanique

La version électronique du RS est:

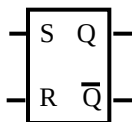


Ce circuit a deux entrées S (Set) et R (Reset ou Clear) et deux sorties, Q_1 et Q_2 . On l'appelle bistable ou encore verrou (latch). Quand $S = R = 0$, il a deux états stables, $Q_1 = 0$ et $Q_2 = 1$, et $Q_1 = 1$ et $Q_2 = 0$.

La présence d'un 1 à l'entrée S cause un "SET" du circuit, c.-à-d. Q_1 devient 1. La présence d'un 1 à l'entrée R cause un "RESET" du circuit, c.-à-d. Q_1 devient 0. La différence de temps entre n et $n+1$ est le délai de propagation entre l'application d'une impulsion SET ou RESET au circuit et l'apparition de la sortie correspondante.

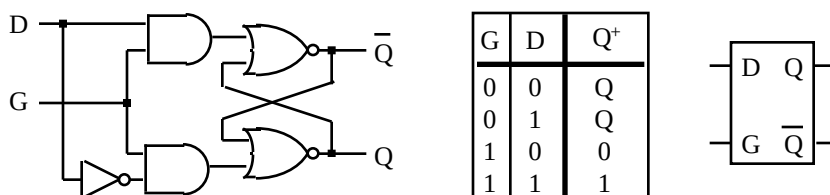
Il est possible d'appliquer simultanément $S = R = 1$. On aura alors $Q_1 = Q_2 = 0$. Cependant, lorsque l'une des entrées redeviendra 0, le circuit prendra l'état correspondant à l'autre entrée. Si on ramène les deux entrées à 0 simultanément, on ne peut prévoir l'état final du circuit, car il y a alors une course critique entre les deux portes. L'état d'entrée $S = R = 1$ n'est donc pas utilisé. Comme dans tous les autres cas Q_1 et Q_2 sont toujours complémentaires, on les désigne respectivement Q et $\bar{Q}$.

On représente ce circuit au moyen du schéma suivant:



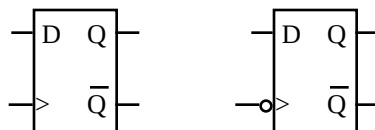
Bistable D

Le bistable D est simplement un bistable RS auquel on a ajouté un inverseur pour éviter que $R = S = 1$, ainsi qu'une entrée d'horloge pour synchroniser son opération avec le reste du système.



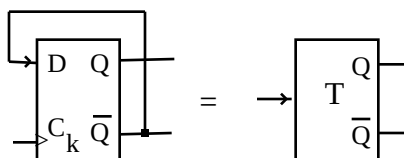
Son équation caractéristique est donc : $Q^+ = DG + Q\bar{G}$ ou, si l'horloge est constituée d'impulsions courtes: $Q^+ = D$. Ce circuit mémorise l'information qui est présente sur l'entrée D au moment où l'horloge G est appliquée. C'est un élément de base avec lequel on peut construire des registres, des compteurs, ou des machines séquentielles (automates).

Le bistable D illustré ci haut est activé par un niveau 1 sur l'entrée d'horloge G . D'autres réalisations (edge-triggered ou master-slave) sont plutôt activées lors d'une transition de l'horloge. Certains circuits sont activés par une montée, d'autres par une descente. On les symbolise comme suit:



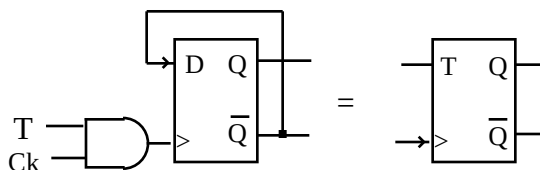
Dans ce cas, l'équation caractéristique devient $Q^{n+1} = D^n$ ou encore $Q^+ = D$, quelle que soit la durée des impulsions d'horloge.

Le bistable T ci-dessous est une variante de la bascule D qui bascule à chaque coup d'horloge, i.e. $Q^{n+1} = D^n = \overline{Q}^n$



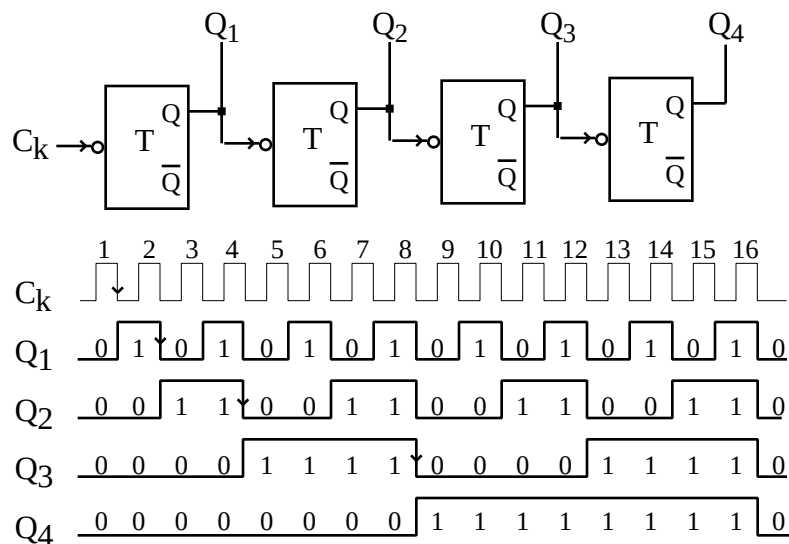
Bistable T

On peut ajouter une entrée T telle que si $T = 0$, le circuit ne fait rien, et que si $T = 1$, le circuit bascule à chaque coup d'horloge:



Applications des bistables

Remarquez dans le compteur binaire de la p. 100 du livre (Figure 5.28), que les sorties Q_1 à Q_4 représentent le nombre de coups d'horloge comptés depuis le dernier passage par 0. On trouve de tels circuits dans les montres au quartz, par exemple, où la fréquence d'un cristal de quartz est divisée par un compteur jusqu'à ce qu'on ait un coup par seconde. Cette fréquence est ensuite divisée par 60 pour donner les minutes, puis encore par 60 pour donner les heures, etc.



Compteur binaire asynchrone modulo-16

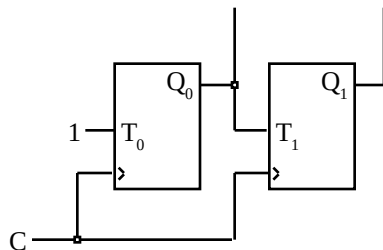
5.3.4 Synthèse d'un circuit séquentiel

On trouve à la p. 101 du livre (Figure 5.29) la synthèse d'un compteur synchrone 2 bits avec une entrée de contrôle pour l'empêcher de compter ou l'y autoriser. On peut considérer deux variantes : la réalisation sans entrée X (qui compte toujours) et la réalisation avec des bistables D.

Réalisation sans entrée X

Q_1	Q_0	Q_1^+	Q_0^+	T_1	T_0
0	0	0	1	0	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	1

D'où : $T_0 = 1$ et $T_1 = Q_0$, et le circuit suivant :



Compteur binaire 2 bits sans entrée

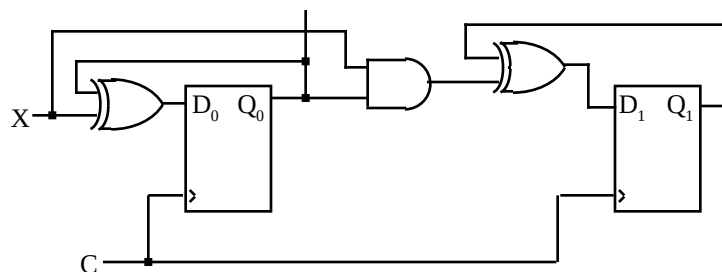
Réalisation du circuit avec entrée à l'aide de bistables D

Q_1	Q_0	X	Q_1^+	Q_0^+	D_1	D_0
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	0	1	0	1
0	1	1	1	0	1	0
	0	0	1	0	1	0
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	0	0	0	0

En faisant les tables de Karnaugh pour D_1 et D_0 en fonction de Q_1 , Q_0 et X , on trouve:

$$D_0 = Q_0 \bar{X} + \bar{Q}_0 X = Q_0 \oplus X$$

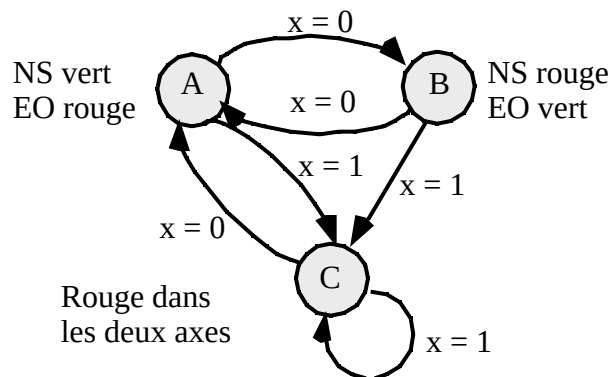
et
$$D_1 = Q_1 \bar{Q}_0 + \bar{X} Q_1 + X \bar{Q}_1 Q_0 = Q_1 (\bar{Q}_0 + \bar{X}) + \bar{Q}_1 X Q_0 = Q_1 \oplus X Q_0$$



Exemple de synthèse de circuit séquentiel : Système de contrôle de feux de circulation

Exemple tiré de Introduction to Digital Logic Design par John P. Hayes, Addison-Wesley, 1993, ISBN: 0-201-15461-7.

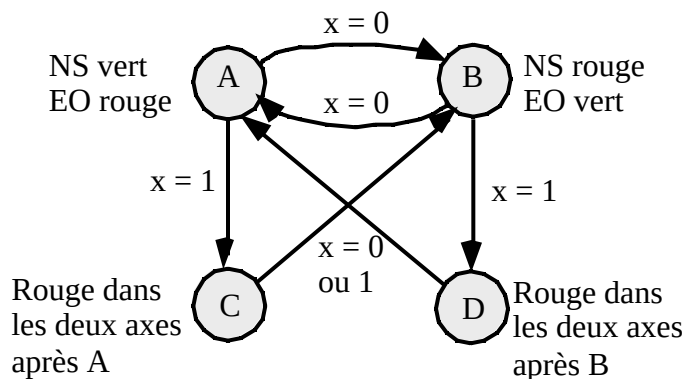
On veut réaliser un système de feux de circulation qui laisse passer alternativement les voitures dans l'axe Nord-Sud (NS) puis dans l'axe Est-Ouest (EO), muni d'un bouton pour piétons, qui met les feux au rouge dans les deux directions et allume un feu pour piétons pendant un certain temps. Une première réalisation pourrait être la suivante :



Quand on n'appuie pas sur le bouton ($x = 0$), les feux alternent régulièrement entre les axes NS et EO, i.e. le système alterne entre les états A et B . Si x devient 1 pendant qu'on est dans l'un ou l'autre de ces états, on passera à l'état C dans lequel les feux sont rouges dans les deux axes et le feu pour piétons est allumé. On reste dans cet état tant que $x = 1$. Quand x redevient 0, on retourne à l'état A .

Ce système est déficient pour deux raisons. Un petit malin pourrait appuyer sans cesse sur le bouton et paralyser toute circulation automobile. D'autre part, comme le système une fois dans l'état C retourne toujours à l'état A , il se pourrait qu'on n'arrive presque jamais à l'état B s'il y avait beaucoup de piétons, et la circulation serait paralysée dans l'axe EO.

Une meilleure réalisation serait la suivante :



Si un piéton appuie sur le bouton pendant que le système est dans l'état A , on passe à l'état C , puis on retourne à l'état B , assurant ainsi l'alternance des feux entre les axes NS et EO. De plus, le système ne peut pas être paralysé comme le précédent.

On obtient donc la table d'états suivante, où la sortie $z_1 = 1$ représente un feu vert en direction EO et la sortie $z_2 = 1$ feu vert en direction NS.

		Entrée présente x		Sortie présente
		0	1	$z_1 z_2$
État présent	A	B	C	01
	B	A	D	10
	C	B	B	00
	D	A	A	00

Comme le système a quatre état, il nécessitera deux bascules. On peut attribuer arbitrairement le codage suivant aux états :

$Q_1 Q_2 = 00$ représente A

$Q_1 Q_2 = 01$ représente B

$Q_1 Q_2 = 10$ représente C

$Q_1 Q_2 = 11$ représente D

Notre table d'états devient donc :

		Entrée présente x		Sortie présente
		0	1	$z_1 z_2$
État présent	00	B	C	01
	01	A	D	10
	10	B	B	00
	11	A	A	00



État futur

On obtient ainsi la table de transition :

Entrée x	État présent $Q_1 Q_2$	État suivant $Q_1^+ Q_2^+$	Sorties présentes $z_1 z_2$
0	00	01	01
0	01	00	10
0	10	01	00
0	11	00	00
1	00	10	01
1	01	11	10
1	10	01	00
1	11	00	00

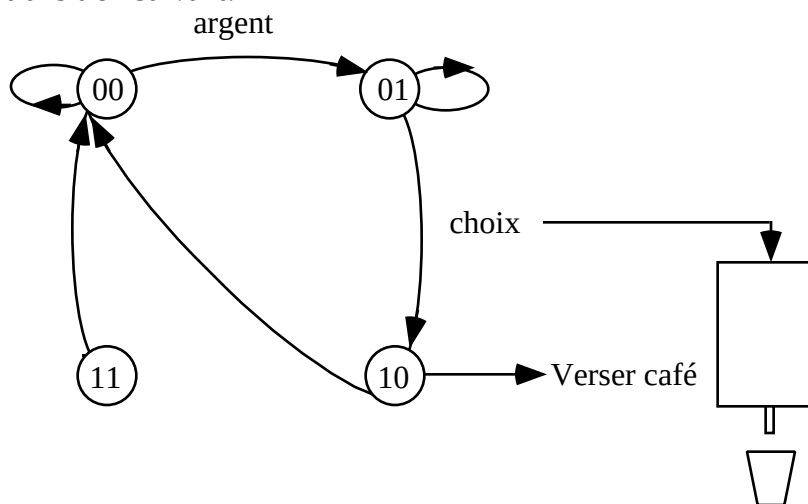
Exemple de synthèse d'un circuit séquentiel: distributrice à café

Nous allons concevoir le circuit complet d'une distributrice à café rudimentaire.

Spécification:

1. la machine attend que le client ait introduit le montant correct d'argent.
2. une fois que le client a déposé l'argent, la distributrice attend qu'il fasse son choix: café fort, café faible, café avec crème, etc.
3. une fois le choix fait, la machine émet une commande à l'unité qui verse le café.
4. la machine retourne à l'état 1.

Il y a donc trois états, qu'on numérote arbitrairement 00, 01, 10. Il y a deux entrées, *argent* et *choix*. Il y a une seule sortie, la commande *Verser café*. Comme il y a plus de deux états, la machine nécessitera deux bistables. Les états sont définis par les sorties Q_1 et Q_0 de ces deux bistables. On a alors quatre états possibles. L'état 11 n'est pas nécessaire dans notre machine, et normalement on n'y parvient jamais. Il se peut toutefois que la machine, au moment de la mise en marche, se trouve dans cet état. On s'arrange donc pour qu'il aille à l'état 00, pour s'assurer qu'on ne peut pas obtenir de café sans avoir déposé d'argent. Ceci donne le diagramme de transition suivant:



Nous pouvons donc écrire notre table de transition:

argent	choix	Q_1	Q_0	Q_1^+	Q_0^+	Verser café
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	1	0	0	0	1
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	0	1	1	0	0
0	1	1	0	0	0	1
0	1	1	1	0	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	0
1	0	1	0	0	0	1
1	0	1	1	0	0	0
1	1	0	0	0	1	0
1	1	0	1	1	0	0
1	1	1	0	0	0	1
1	1	1	1	0	0	0

Nous pouvons maintenant réaliser le circuit. Il reste à décider quelle sorte de bistable on utilise. Supposons qu'il s'agit de bascules D. Puisque dans une telle bascule $Q^+ = D$, la table de transition nous donne directement D_1 et D_0 dans les colonnes Q_1^+ et Q_0^+ respectivement. Nous pouvons maintenant établir les tables de Karnaugh pour D_1 et D_0 . Quant à la sortie *Verser café*, il est assez évident qu'elle est égale à $Q_1 \cdot Q_0$, on n'a donc pas besoin de tables de Karnaugh pour l'obtenir.

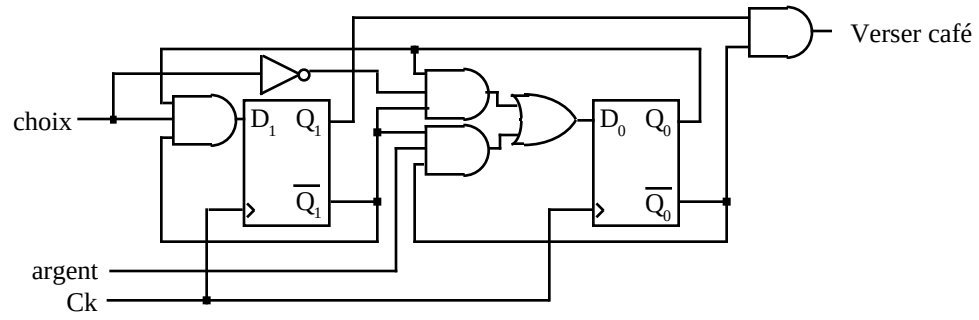
		$Q_1 Q_0$			
argent	choix	00	01	11	10
		0	0	0	0
	00	0	0	0	0
	01	0	1	0	0
	11	0	1	0	0
	10	0	0	0	0

$$Q_1^+ = \text{choix} \cdot \overline{Q_1} \cdot Q_0$$

		$Q_1 Q_0$			
argent	choix	00	01	11	10
		0	1	0	0
	00	0	0	0	0
	01	0	0	0	0
	11	1	0	0	0
	10	1	1	0	0

$$Q_0^+ = \text{argent} \cdot \overline{Q_1} \cdot \overline{Q_0} + \text{choix} \cdot \overline{Q_1} \cdot Q_0$$

D'où le circuit :



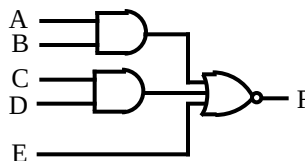
On laisse comme exercice au lecteur la réalisation au moyen de bistables T ou RS.

Exercices du chapitre 5

- 5.1 Écrire la table de vérité d'un ET et d'un OU à 3 entrées.
- 5.2 Démontrer que
- $(a + b).(a + \bar{b}) = a$
 - $a + (\bar{a} b) = a + b$
- 5.3 a. À l'aide de la table de vérité, vérifier la non-associativité du NON ET logique.
 b. En se servant des propriétés mathématiques des fonctions ET et OU, établir de façon littérale l'associativité du OU disjonctif.
- 5.4 À partir de la table de vérité suivante, écrire $Y1$ et $Y2$ en fonction de A , B et G . Simplifier les fonctions logiques puis dessiner le circuit correspondant en employant les portes logiques élémentaires (penser aux théorèmes d'absorption).

G	A	B	Y1	Y2
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0

- 5.5 a. Établir la table de vérité d'un soustracteur binaire comportant 3 entrées : A_i , B_i , la retenue d'une opération antérieure C_i , et 2 sorties, D_i et C_{i+1} (voir celle de la table d'addition à la p. 99 du livre).
 b. Établir les 2 équations logiques donnant D_i et C_{i+1} en fonction de A_i , B_i , et C_i . Simplifier.
 c. Tracer les circuits permettant de réaliser ces deux fonctions.
- 5.6 Établir la table de vérité d'un NON ET avec les deux entrées connectées ensemble et montrer qu'il réalise un circuit inverseur. On considère ensuite le circuit logique suivant :



- Donner F en fonction des variables A, B, C, D et E .
- Peut-on réaliser la même fonction en n'employant que des NON ET ? Dessiner le circuit correspondant.

5.7 Tracez le circuit d'un multiplexeur à deux entrées.

5.8 Montrez qu'il est possible de réaliser un OU-Disjonctif au moyen de seulement quatre portes NAND.

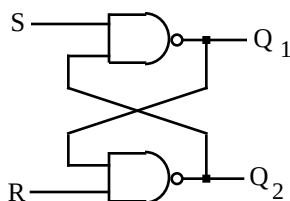
5.9 Réaliser le circuit ayant la table de vérité suivante à l'aide d'un multiplexeur :

a	b	c	x
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

5.10 En vous basant sur le décaleur vu plus haut, tracez :

- un circuit qui décale vers la droite quand $C = 1$, et qui ne décale pas quand $C = 0$.
- un circuit qui décale de deux bits vers la droite quand $C = 1$, et qui ne décale pas quand $C = 0$.

5.11 À la p. 99 du livre, on trouve le circuit d'un bistable RS réalisé avec des portes NON-OU. Considérer la réalisation au moyen de portes NON-ET suivante :



Établir sa table de transition ainsi que son équation caractéristique comme à la p. 88 du livre.

5.12 Réaliser un compteur binaire synchrone modulo-3 ($Q_1Q_0 = 00, 01, 10, 00$) à l'aide de bistables D.

Chapitre 7

Mémoires

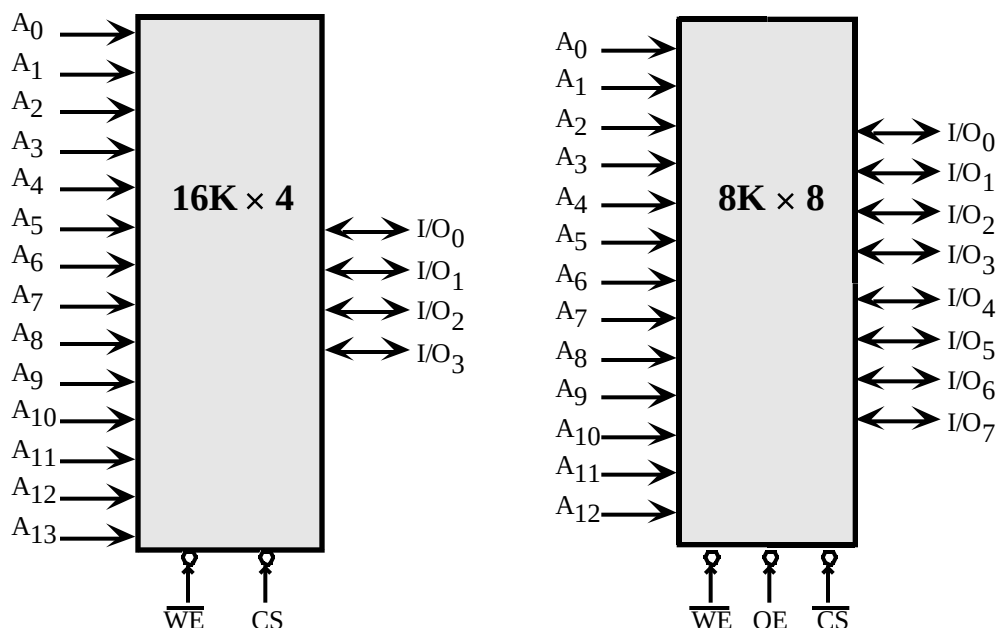
7.2 Mémoire centrale

7.2.1 Mémoires à semi-conducteurs

Tel que mentionné à la p. 143 du livre de Zanella et Ligier, le groupe des mémoires vives (RAM) se divise en deux sous-groupes. Si la cellule élémentaire de mémoire est constituée de circuits de type bistable, la mémoire est dite **statique** (SRAM). Par contre, dans les mémoires dites **dynamiques** (DRAM), l'élément de mémoire est constitué d'une capacité (condensateur). Les mémoires dynamiques sont plus denses (donc d'une plus grande capacité) et moins coûteuses mais doivent être rafraîchies plusieurs centaines de fois par seconde, ce qui complique les circuits.

Organisation des circuits mémoire

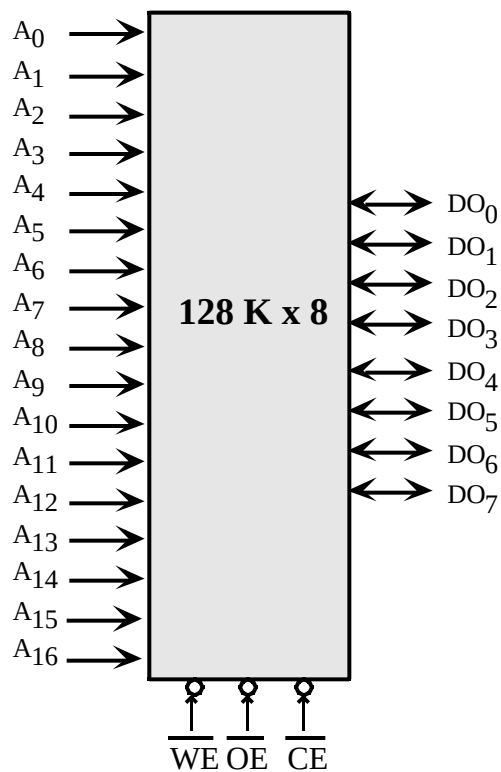
La capacité d'un circuit intégré de mémoire s'exprime en bits, en kilobits (Kbits) ou en mégabits (Mbits). Cependant, pour un nombre de bits donné, il y a plusieurs façons d'organiser l'information à l'intérieur du circuit. Par exemple, dans les mémoires statiques, un circuit intégré de 64 kilobits (64 Kbits) peut être organisé en 16 384 mots de 4 bits, ou en 8 192 mots de 8 bits (octets).



Mémoires statiques (SRAM)

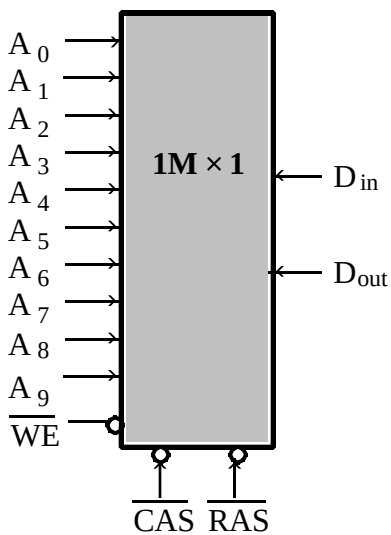
Dans le premier cas, on a besoin de 14 lignes d'adresse ($2^{14} = 16\,384$). Si on veut obtenir une mémoire avec une longueur de mot de 8 bits, c.-à-d. pouvoir accéder à un octet à la fois, il faudra 2 circuits intégrés de 16 Kbits $\times$ 4, ce qui donne une capacité totale de 16 Koctets. Dans le second cas, on n'a besoin que de 13 lignes d'adresse ($2^{13} = 8\,192$) et un seul circuit intégré nous donne une mémoire de 8 Koctets. Il en faut évidemment 2 pour obtenir 16 Koctets. Seule l'interconnexion des puces diffère. Les adresses supplémentaires, car il en faut 14 ici aussi, seront plutôt appliquées à un décodeur 1-2 qui agira sur les pattes de sélection de puce $\overline{CS}$ (Chip Select). Les autres entrées de contrôle sont la validation d'écriture $\overline{WE}$ (Write Enable) et la validation de sortie $\overline{OE}$ (Output Enable). On remarque dans les deux cas que les broches de données sont bidirectionnelles, c.-à-d. lors d'une lecture, le contenu de l'adresse choisie apparaît sur ces broches, tandis que lors d'une écriture, on applique sur ces mêmes broches les bits à écrire à l'adresse choisie. Il va de soi que $\overline{WE}$ et $\overline{OE}$ ne peuvent être actifs en même temps.

Autre exemple de mémoire statique :



Mémoire statique de 128 Ko.

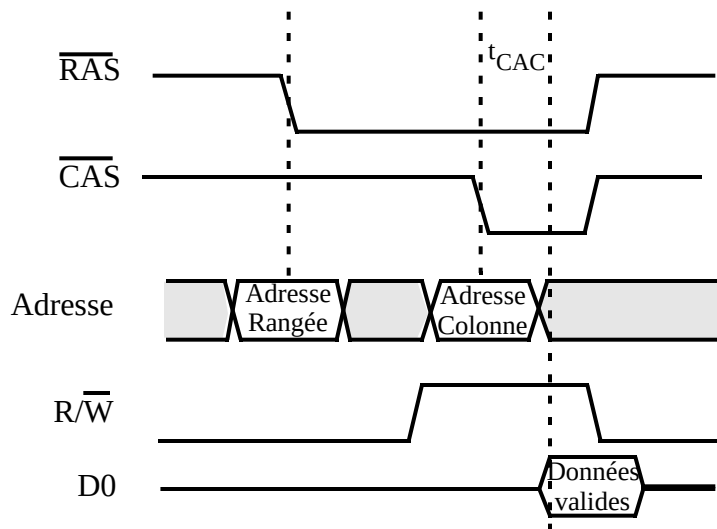
Mémoires dynamiques:



Mémoire dynamique (DRAM)

Dans le cas particulier des mémoires dynamiques, l'organisation est en mots de 1 bit, de 4 bits ou de 8 bits. L'adresse de n bits est scindée en deux moitiés. L'une adresse la colonne et

l'autre adresse la rangée dans un plan de $2^{n/2} \times 2^{n/2}$ mots. Ces adresses rangée et colonne sont multiplexées, c.-à-d. les n adresses ne sont pas toutes appliquées simultanément, ce qui permet l'utilisation de puces plus petites. Les deux parties, de $n/2$ bits chacune, sont appliquées successivement. Les lignes $\overline{\text{RAS}}$ (Row Address Strobe) et $\overline{\text{CAS}}$ (Column Address Strobe) indiquent au circuit de mémoire de quelle partie de l'adresse il s'agit. On applique d'abord la moitié de poids fort de l'adresse sur les broches d'adresse, puis on abaisse $\overline{\text{RAS}}$. On applique ensuite la partie de poids faible, puis on abaisse $\overline{\text{CAS}}$. En lecture, après un temps appelé temps d'accès, une donnée valide apparaît sur la broche (ou les broches si la mémoire est organisée en mots de plus d'un bit) D_{out} et y reste valide tant que $\overline{\text{CAS}}$ est bas.



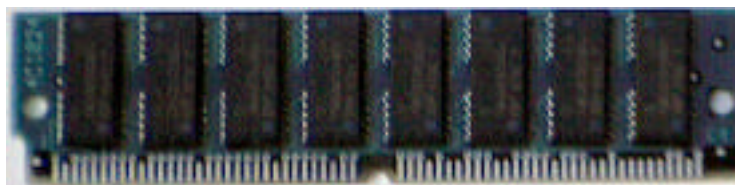
Chronogramme d'une DRAM

On remarque l'économie d'espace considérable par rapport à la mémoire statique de capacité beaucoup moindre illustrée plus haut. Il faut cependant retenir que les mémoires dynamiques sont passablement plus lentes que les mémoires statiques pour plusieurs raisons :

- . le temps d'accès d'une mémoire dynamique (entre $\overline{\text{CAS}}$ et donnée valide) est généralement plus long que celui d'une mémoire statique (de l'ordre de 100 ns pour une mémoire dynamique, et de 10 ns pour une mémoire statique),
- . le temps de cycle, c.-à-d. entre deux lectures ou deux écritures, est beaucoup plus long que le temps d'accès lui-même dans le cas d'une mémoire dynamique,
- . il y a perte d'efficacité (6-12%) causée par le cycle de rafraîchissement des mémoires dynamiques.

7.2.2 Structure physique de la mémoire centrale

Les puces de mémoire dynamique sont souvent fournies sous forme de modules aisément enfichables appelés SIMM (Single Inline Memory Module) comportant 8 puces (ou 9 pour les mémoires avec bit de parité) de $256\text{ K} \times 1$, ou $1\text{ M} \times 1$ ou $4\text{ M} \times 1$ ou $16\text{ M} \times 1$, constituant ainsi des modules de 256 Ko, 1 Mo, 4 Mo ou 16 Mo respectivement.



SIMM de 1 Mo grandeur nature

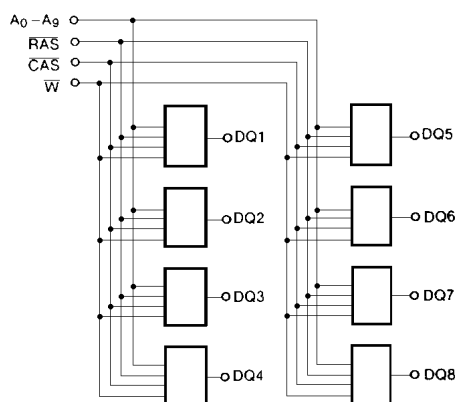


Schéma du SIMM précédent

Légende:

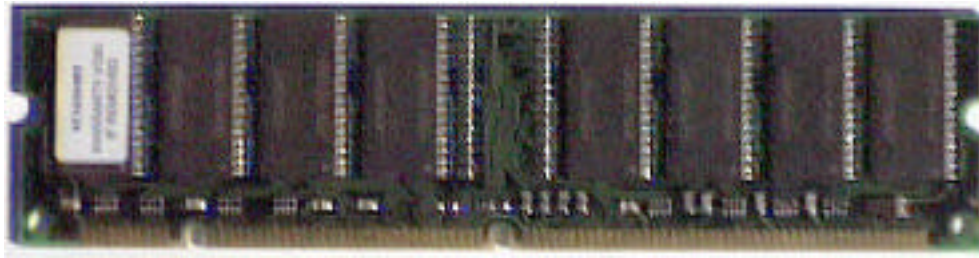
8 puces de $1\text{ M} \times 1$, i.e. 1 048 576 adresses contenant chacune 1 bit d'information.

A_0-A_9 = 20 fils d'adresse multiplexés sur 10 fils sous contrôle de RAS et CAS.

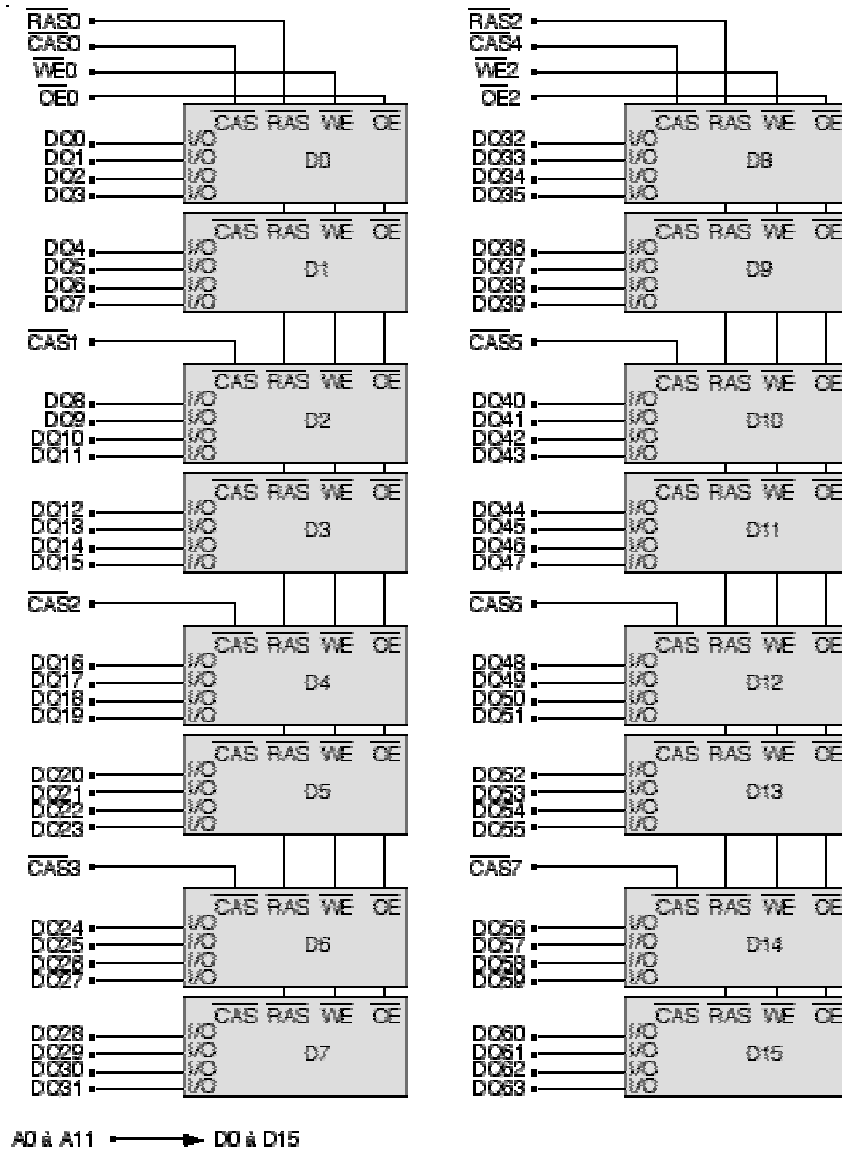
$\overline{W}$ = commande de lecture/écriture.

DQ1 à DQ8 = entrées/sorties allant au bus de données D0 à D7 de l'ordinateur.

Dans les machines contemporaines, qui lisent 64 bits en mémoire à la fois, on utilise plutôt des modules à 172 broches appelés DIMM (Dual Inline Memory Modules) pouvant contenir par exemple 128 Mo de mémoire dynamique EDO ou SDRAM.



DIMM de 128 Mo grandeur nature



Circuit d'un DIMM de 128 Mo avec des puces de 16 M x 4.

La mémoire est de loin l'élément le plus lent d'un ordinateur. Comparez le temps de cycle de l'ordre de 50 ns d'une DRAM typique avec le temps de cycle de 1,25 ns d'un processeur avec

une horloge de 800 MHz. Il est évident que si on ne prend pas de précautions, le processeur passera une bonne partie de son temps à attendre la RAM. Des progrès substantiels ont été réalisés au cours des dernières années.

La mémoire centrale des micro-ordinateurs est composée de DRAM pour des raisons de coût. Une mémoire DRAM peut être considérée comme un tableau de cellules. Ces cellules sont composées de condensateurs et contiennent un ou plusieurs bits de donnée, dépendant de la configuration de la puce. On adresse cette table au moyen de décodeurs de rangées et de colonnes, qui reçoivent leurs signaux des lignes RAS et CAS.

Des transistors d'accès appelés «sense amplifiers» sont connectés à chaque colonne et assurent les fonctionnalités de lecture et de rafraîchissement de la puce. En effet, puisque les cellules sont des condensateurs qui sont déchargés lors d'une lecture, le «sense amplifier» doit restaurer les données avant la fin d'un cycle de lecture.

Les condensateurs qui constituent les cellules de données ont tendance à se décharger en raison de fuites électriques. Ils ont donc besoin d'un cycle de rafraîchissement périodique sinon les données seront perdues. Un contrôleur de rafraîchissement détermine le temps entre les cycles de rafraîchissement, et un compteur de rafraîchissement vérifie que tout le tableau (toutes les rangées) a bien été rafraîchi. Ceci signifie que des cycles mémoire sont utilisés pour le rafraîchissement, ce qui réduit la performance.

Un accès mémoire typique se déroule comme suit. D'abord, les bits d'adresse de rangée sont appliqués au bus d'adresses. Après un certain temps, le signal RAS est affirmé, ce qui active les «sense amplifiers» et verrouille l'adresse de rangée dans le tampon d'adresse de rangée. Une fois le signal RAS stabilisé, la rangée choisie est appliquée aux «sense amplifiers». Ensuite, les bits d'adresse de colonne sont appliqués, puis verrouillés dans le tampon d'adresse de colonne lors de l'affirmation de CAS. Le buffer de sortie est activé au même moment. Quand CAS est stabilisé, le «sense amplifier» sélectionné envoie ses données au buffer de sortie.

Dans une interface asynchrone, une période minimum de temps est fixée pour garantir qu'une opération est complète. On alloue une période de temps minimum pour chacune des opérations internes de la puce, de sorte que si un cycle d'horloge a lieu avant l'expiration de ce temps minimum, on doit attendre au prochain cycle avant de permettre l'opération suivante.

En implémentant des modes d'accès spéciaux, on a pu éliminer certaines des opérations internes pour certains types d'accès. Le premier fut l'accès en mode page.

Mode Page

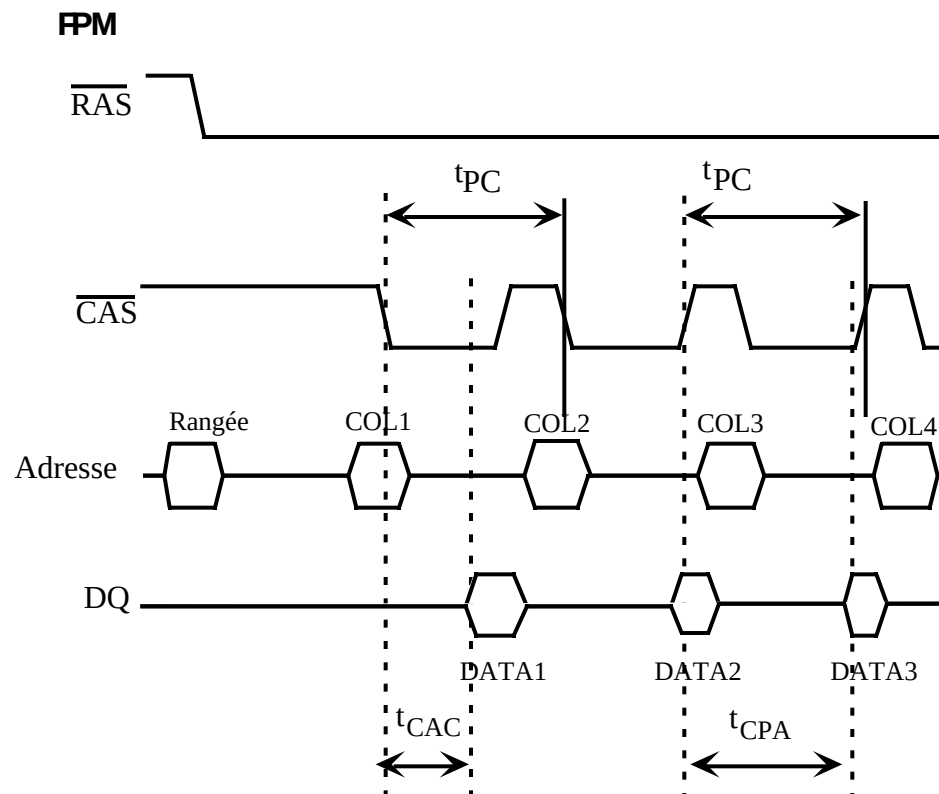
Avec cette méthode, le signal RAS est maintenu actif, de sorte qu'une page entière de données est maintenue à l'entrée des «sense amplifiers». On peut changer les adresses de colonne en activant CAS de façon répétée. Ceci permet des lectures beaucoup plus rapides, puisqu'on élimine le temps d'activation de l'adresse de rangée.

Certaines applications profitent largement de cette approche, tandis que d'autres pas du tout.

Mode page rapide

Le mode page rapide améliore le mode page original en éliminant le temps de mise en place de l'adresse de colonne lors d'un cycle de page. On réalise ceci en activant les tampons d'adresse de colonne sur le bord descendant de RAS (au lieu de CAS). Puisque RAS demeure actif (bas) pour toute la durée du cycle de page, ce qui agit comme un latch transparent quand CAS est haut et permet à l'activation de l'adresse de se produire dès que l'adresse de colonne est valide, plutôt que d'attendre la descente de CAS. Les temps d'accès typiques sont 6-3-3-3 (latence initiale de 6 périodes d'horloge, avec un accès page de 3 cycles).

Le mode page rapide a été fort répandu et est encore utilisé dans plusieurs systèmes. Il réduisait la consommation de puissance, principalement parce que les courants de sense et de restore ne sont pas nécessaires durant un accès en mode page. Il présentait quelques inconvénients. Le principal, était que les buffers de sortie sont désactivés quand CAS est haut, ce qui a pour effet d'ajouter 5 ns au temps de cycle.



Chronogramme de la DRAM en mode Fast Page

t_{CAC} = temps d'accès colonne

t_{PC} = temps de précharge

t_{CPA} = temps d'accès colonne à colonne

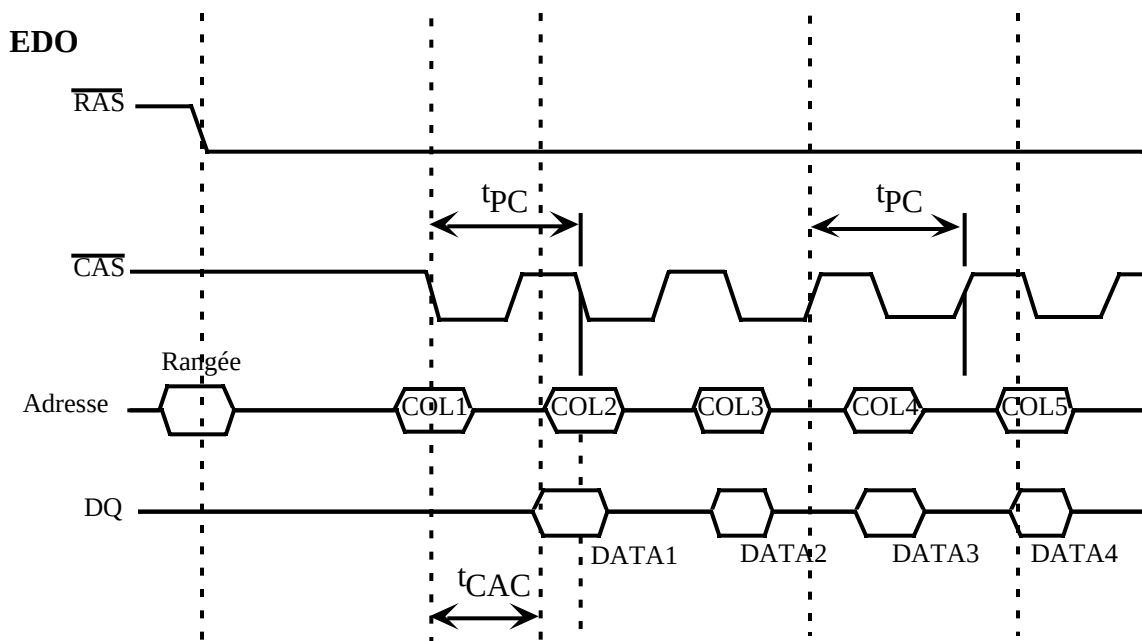
Mémoire EDO

La dernière amélioration majeure apportée aux DRAM asynchrones a été la mémoire EDO (Extended DataOut). Elle consiste à ne plus désactiver les tampons de sortie sur la montée de CAS. Ceci élimine le temps de précharge au moment de verrouiller les données en sortie. Ceci permet de réduire le temps minimum où CAS est bas, et la montée peut donc se produire plus tôt. Une nouvelle adresse de colonne peut ainsi être appliquée avant la fin du cycle précédent.

EDO procure une amélioration de 40% du temps d'accès, et fonctionne bien jusqu'à une fréquence de bus de 83 MHz. Si les puces DRAM sont suffisamment rapides (55 ns ou mieux), on peut même utiliser les mémoires EDO jusqu'à 100 MHz de fréquence de bus. Les temps d'accès typiques sont 5-2-2-2 avec une fréquence de bus de 66 MHz.

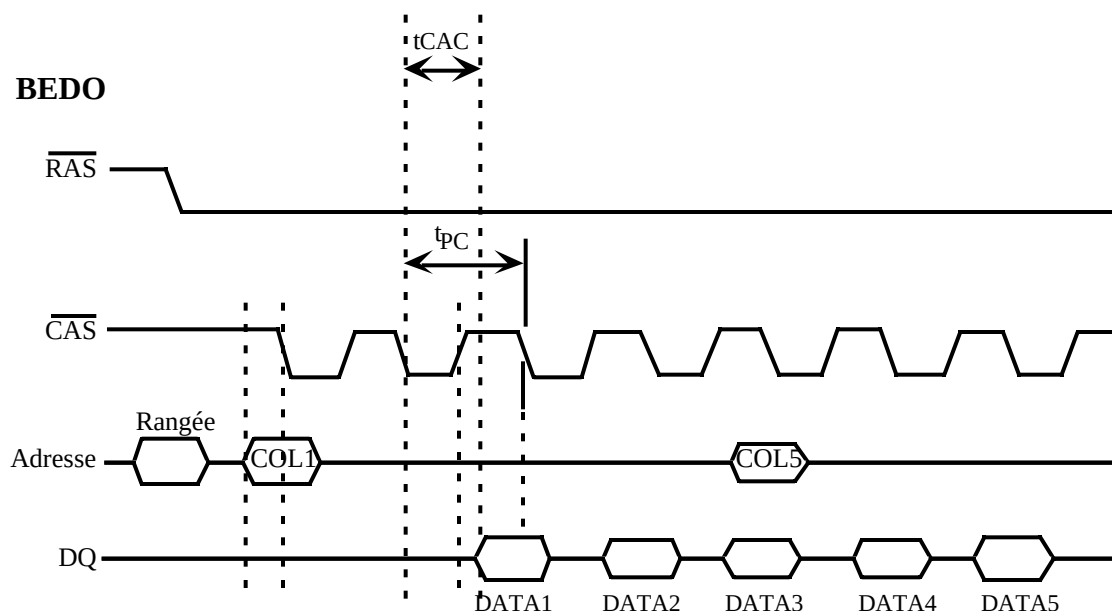
La mémoire EDO est maintenant considérée comme dépassée. La plupart des fabricants ont cessé de la produire. La tendance est maintenant d'utiliser des DRAM synchrones.

Si vous avez déjà de la mémoire EDO, il n'y a pas de raison sérieuse de changer pour une mémoire synchrone à moins que votre vitesse de bus dépasse 83 MHz. Même alors, il n'y a en fait pas de différence significative avant 100 MHz.



Chronogramme d'une mémoire EDO.

Une variante de EDO, Burst EDO ou BEDO, bien qu'une excellente idée, n'a pas été retenue par Intel, pour des raisons politiques et économiques, de sorte que la technologie n'a jamais réellement percé. Le mode «rafale» est un progrès sur le mode page : après qu'une première adresse est lue, les 3 adresses suivantes sont générées à l'interne, éliminant ainsi le temps nécessaire à fournir les 3 adresses de colonne. Les temps d'accès à 66 MHz sont 4-1-1-1.



Chronogramme d'une mémoire BEDO

Mémoire SDRAM

Quand il devint apparent que les vitesses de bus se dirigeaient vers 100 MHz, les concepteurs de DRAM cherchèrent un moyen de contourner les latences significatives toujours présentes. L'implantation d'une interface synchrone a permis une telle réduction et procure également d'autres avantages.

Dans une interface asynchrone, le processeur doit attendre sans rien faire que la DRAM ait effectué ses opérations internes, ce qui prend de l'ordre de 60 ns. Avec une interface synchrone, la DRAM verrouille l'information provenant du processeur, sous le contrôle de l'horloge système. Ces verrous enregistrent les adresses, les données et les signaux de contrôle, ce qui permet au processeur de vaquer aussitôt à d'autres tâches. Après un certain nombre de cycles d'horloge, les données deviennent disponibles et le processeur peut les lire sur les lignes de sortie.

Une variante, la mémoire ESDRAM, est une SDRAM dotée d'un petit cache en SRAM, ce qui réduit les temps de latence et permet des opérations en mode rafale jusqu'à 200 MHz. Ce cache conserve les données le plus fréquemment utilisées pour minimiser les accès à la DRAM plus lente. On peut utiliser un bus très large entre la SRAM et la DRAM, ce qui augmente la bande passante.

Intel mise sur le concept RDRAM développé par Rambus, Inc. qui prétend pouvoir atteindre des vitesses de bus de 800 MHz. Mais c'est une technologie brevetée et tous les fabricants devront payer des royalties à Rambus.

Plusieurs fabricants de mémoire appuient la technologie SDRAM, qui est un standard industriel ouvert, ce qui devrait permettre des prix moins élevés. Un de ses avantages, est qu'elle n'exige pas une conception nouvelle des puces de DRAM. On atteint présentement 400 MHz et on prétend pouvoir atteindre 800 MHz sous peu. Les temps d'accès sont 5-1-1-1 à 83 MHz

VRAM

Un sous-ensemble des mémoires dynamiques est constitué par les mémoires VRAM spécialisées dans l'affichage vidéo des ordinateurs. Ce que ces mémoires ont de particulier, c'est qu'elles ont deux ports d'accès : un port à accès direct comme les DRAM ordinaires, et un port à accès séquentiel. Ainsi, pendant que l'ordinateur accède à cette mémoire de façon normale, le circuit gestionnaire de l'affichage peut en lire séquentiellement le contenu pour rafraîchir l'écran. Les deux accès peuvent s'effectuer simultanément.

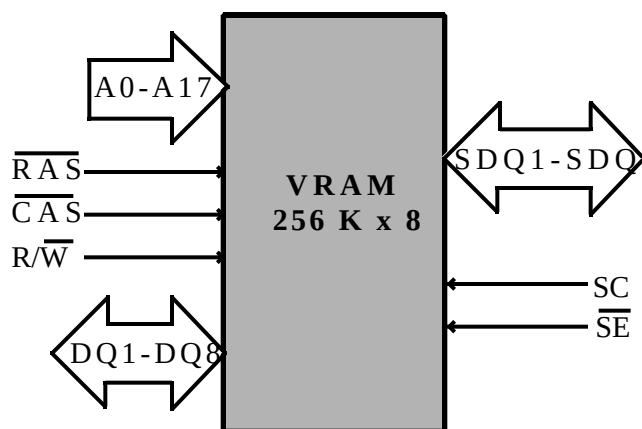


Schéma de principe d'une VRAM

Du côté gauche, on trouve les accès aléatoires normaux d'une DRAM. Du côté droit, on trouve l'accès séquentiel, qui s'effectue sous le contrôle de SC (serial clock). Chaque coup de SC incrémente l'adresse lue (ou écrite). La fréquence de SC peut atteindre 40 MHz, ce qui permet d'afficher 40 Mo à la seconde.

7.3 Mémoire cache

Pour augmenter la performance des ordinateurs à un prix modeste, on utilise des mémoires caches. Ce sont des mémoires plus petites faites de puces rapides (SRAM) qu'on utilise en plus de la mémoire principale qui, elle, est faite de puces plus lentes (DRAM). De tels caches ont en général une taille de 1 Ko à 1 Mo. Si une donnée est dans le cache, elle sera ainsi obtenue plus rapidement que si on devait accéder à la mémoire principale. Si elle n'y est pas, le temps d'accès est celui de l'interrogation du cache + celui d'un accès à la mémoire principale.

La raison principale pour laquelle on utilise une mémoire cache est le prix élevé de la mémoire statique (SRAM). En effet, la mémoire dynamique (DRAM) coûte environ 20 \$ par Mo, tandis que la mémoire statique (SRAM) coûte environ 800 \$ par Mo. Une mémoire centrale faite de SRAM serait donc trop dispendieuse. Mais en utilisant un cache de 256 Ko (200 \$) à 1 Mo (800 \$), on peut obtenir une performance comparable à celle d'une SRAM, tout en ayant une mémoire centrale faite DRAM. Le cache est fait de mémoire SRAM de 10 à 15 ns, tandis que la mémoire centrale est faite de DRAM de 70 à 80 ns.

La performance d'un cache repose sur le **principe de localité** :

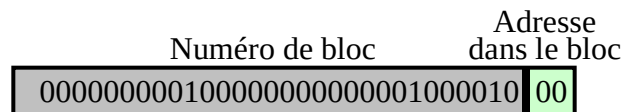
1. Localité temporelle : si un objet a été référencé, il sera référencé de nouveau bientôt.
2. Localité spatiale : si un objet a été référencé, des objets proches seront également référencés.

L'unité d'information qui peut être présente ou non dans le cache est appelée un bloc. Les blocs ont généralement entre 4 et 128 octets et sont tous de même taille dans un cache donné. Le taux de succès est le rapport entre le nombre d'accès mémoire pour lesquels le bloc est présent dans le cache et le nombre total d'accès. Le taux d'échec est :

$$\text{Taux d'échec} = 1 - \text{taux de succès.}$$

Le temps associé au succès est le temps d'accès à un bloc du cache, et comprend le temps nécessaire pour déterminer si le bloc est présent ou non. Le temps de traitement de l'échec est le temps de remplacement d'un bloc du cache par le bloc correspondant de la mémoire principale.

Pour réaliser un cache, l'adresse est divisée en deux parties : celle de gauche s'appelle le numéro de bloc, celle de droite détermine l'adresse de l'octet dans le bloc. Par exemple, dans le cas d'un cache en blocs de 4 octets, on aurait la partition suivante (l'adresse indiquée est 0x00400108) :

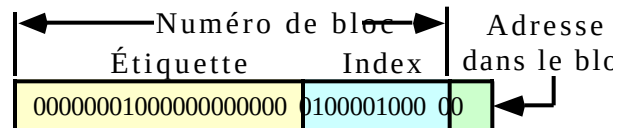


Partition de l'adresse

On distingue trois sortes de mémoires caches : le cache à correspondance directe, le cache totalement associatif et le cache associatif par ensemble de blocs.

Cache à correspondance directe.

Dans le cache à correspondance directe, le champ Numéro de bloc est divisé en deux parties. Celle de droite est déterminée par le nombre de blocs dans le cache et s'appelle **index**. La partie de gauche, formée des bits que restent, s'appelle **étiquette**.



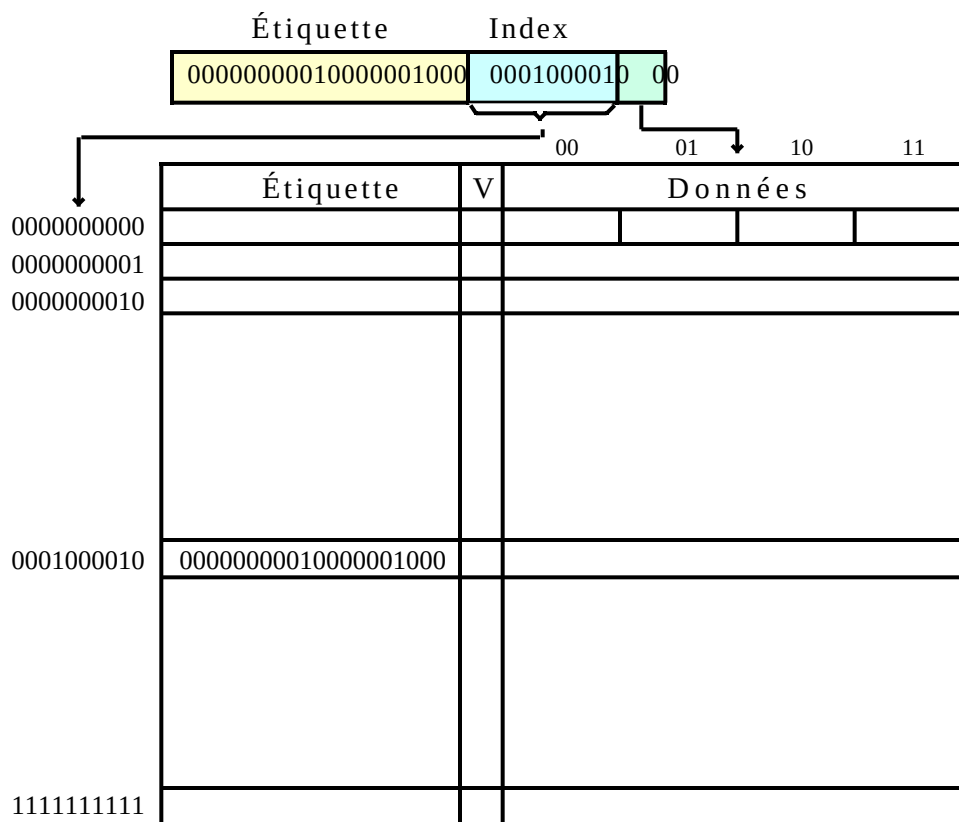
Ici, on aurait un cache de 4 Ko, composé de 1024 blocs de 4 octets.

L'index détermine la rangée du cache où est stockée l'information correspondant à l'adresse formée par l'étiquette et l'index. Ainsi :

$$\text{N}^{\circ} \text{ du bloc dans le cache} = \text{N}^{\circ} \text{ du bloc mémoire modulo le nb de blocs dans le cache.}$$

Pour un index donné, le cache vérifie si l'étiquette contenue dans la rangée correspondante est la même que dans l'adresse. Si oui et si le bit V = 1 alors il y a succès. Sinon, le bloc est chargé à partir de la mémoire principale, placé avec son étiquette à la rangée correspondant à l'index et le bit V de cette rangée est validé.

L'inconvénient d'un tel cache est qu'il ne peut pas contenir deux blocs ayant le même index mais une étiquette différente. Notez que les blocs correspondant à une adresse donnée sont toujours placés au même endroit.



Cache à correspondance directe

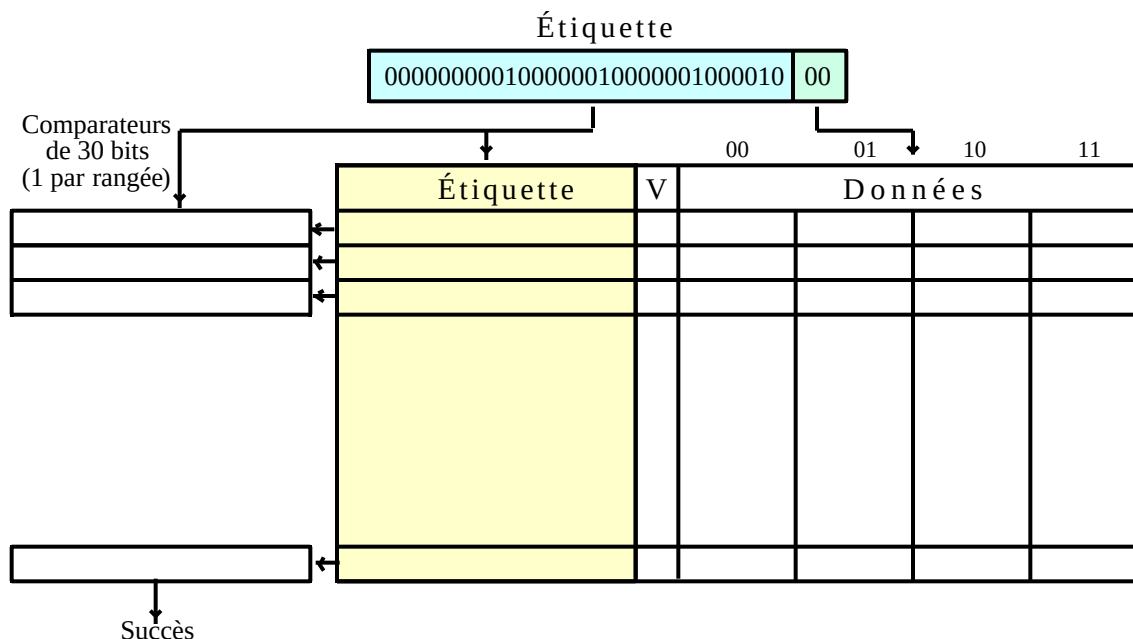
Dans le cache à correspondance directe, la taille des rangées en bits est :

nb de bits dans l'étiquette + bit V + nombre de bits par bloc.

Ici, c'est $20 + 1 + 32 = 53$ bits.

Cache totalement associatif

Dans un tel cache, l'étiquette est constituée par le numéro de bloc. Toutes les étiquettes du cache sont comparées en parallèle avec celle de l'adresse. C'est donc le plus coûteux de tous puisqu'il doit y avoir autant de comparateurs que de rangées dans le cache et que chaque comparateurs doit avoir autant de bits que l'étiquette. Si la bonne étiquette s'y trouve et que le bit V de la rangée correspondante est 1, il y a succès et le cache retourne la donnée correspondante. Si elle ne s'y trouve pas ou que $V = 0$, le bloc correspondant est chargé à partir de la mémoire principale. Tant qu'il reste une rangée où $V = 0$, il est placé à la première rangée ayant $V = 0$, avec son étiquette, et le bit V de cette rangée est porté à 1. Si tous les bits V sont 1, il remplace un bloc déjà présent à l'endroit déterminé par un algorithme de remplacement.



Cache associatif

Contrairement au cache à correspondance directe, les blocs correspondant à une adresse donnée peuvent être placés n'importe où.

L'algorithme de remplacement utilisé quand il faut remplacer un bloc et que tous les blocs du cache sont valides ($V = 1$) peut être :

1. Le hasard ;
2. L'algorithme du plus ancien LRU (Least Recently Used), c'est-à-dire qu'on remplace le bloc qui a été utilisé le moins récemment.

Dans le cache totalement associatif, la taille des rangées est :

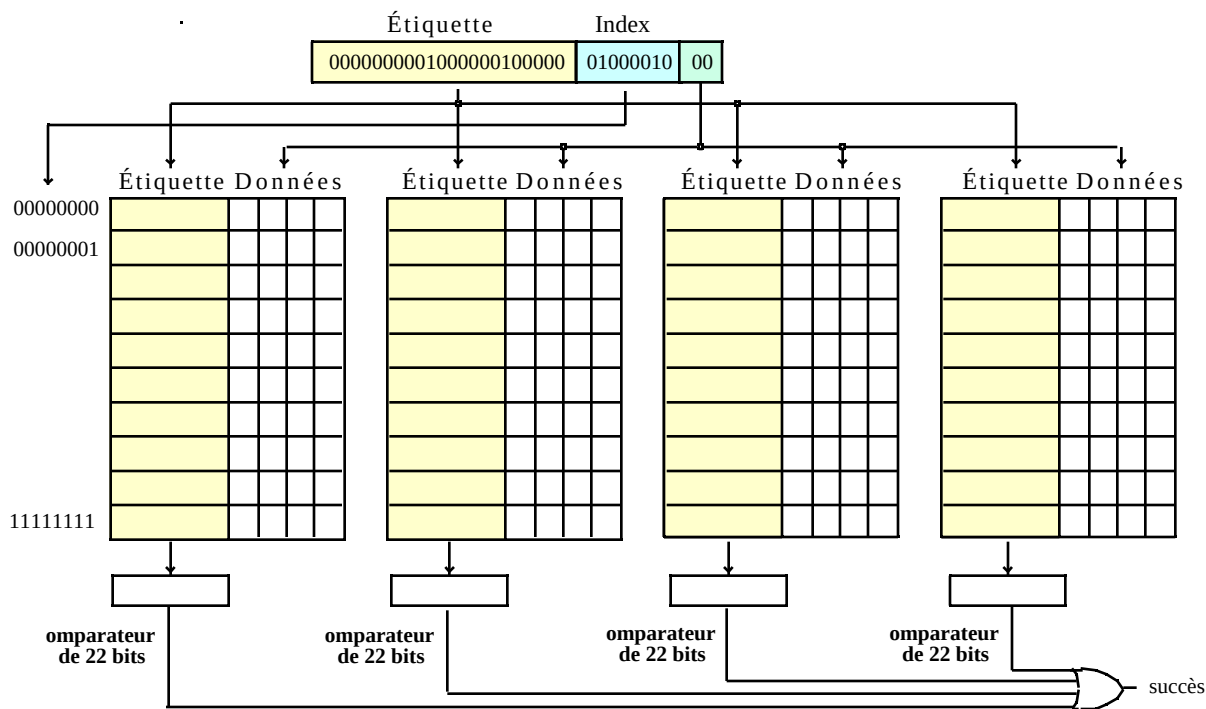
nombre de bits dans l'étiquette + bit V + nombre de bits par bloc (+ bits d'ancienneté si on utilise l'algorithme LRU).

Dans le cas d'un cache de 1024 mots de 4 octets, l'étiquette a 30 bits, donc rangées de $30 + 1 + 32 = 63$ bits (+ bits d'ancienneté). Un tel cache nécessiterait 1024 comparateurs de 30 bits !

Cache associatif par ensemble de blocs

Dans le cache associatif par ensemble de blocs, qui est un compromis entre les deux types précédents, l'index pointe vers un ensemble de blocs associatifs. Lors d'une interrogation, les étiquettes des blocs correspondant à un index donné sont comparées en parallèle. Dans un tel cache, il est possible d'avoir des blocs de même index et d'étiquette différente.

Supposons des ensembles de 4 blocs. On aura donc pour un cache de 4 Ko organisé en rangées de 4 octets, 256 ensembles de 4 blocs de 4 octets. Le champs index a donc besoin de 8 bits. Le champ étiquette aura 22 bits pour une adresse de 32 bits.



Cache associatif par ensemble de blocs

Dans un cache associatif par ensemble de blocs, la taille des rangées est :

Nombre de bits de l'étiquette + bit V + nombre de bits par bloc

Ici, c'est : $22 + 1 + 32$ (+ 2 bits d'ancienneté) = 55 ou 57 bits.

Un tel cache nécessite 4 comparateurs de 22 bits.

Écriture dans un cache.

Il y a deux options principales pour les politiques d'écriture dans un cache : l'**écriture simultanée** (write-through) et la **réécriture** (write-back). Dans le mode écriture simultanée, l'écriture se fait à la fois dans le cache et dans la mémoire principale. Donc toute écriture prend autant de temps qu'un échec, de sorte que l'efficacité du cache est réduite.

La réécriture consiste à ne mettre à jour la mémoire principale que lorsqu'un bloc qui a été modifié par une écriture est remplacé dans le cache. Il faut donc garder un bit de plus par rangée (bit modifié) pour indiquer quels blocs ont été modifiés par des écritures. L'avantage

de cette méthode est que l'écriture est aussi rapide que la lecture tant qu'un bloc est dans le cache. Le désavantage est qu'elle rend la copie des données dans la mémoire principale inconsistante tant que les données modifiées n'y ont pas été recopiées.

7.5 Mémoires auxiliaires

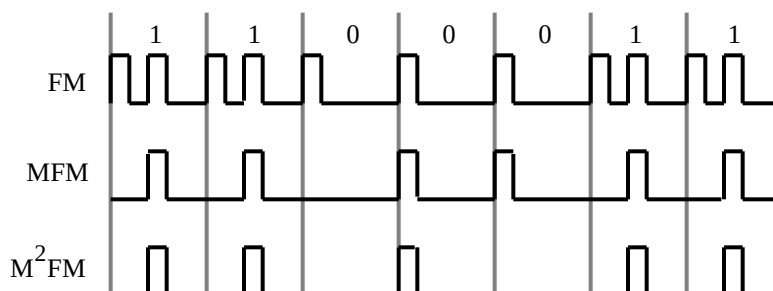
7.5.1 Enregistrement magnétique

Techniques de codage

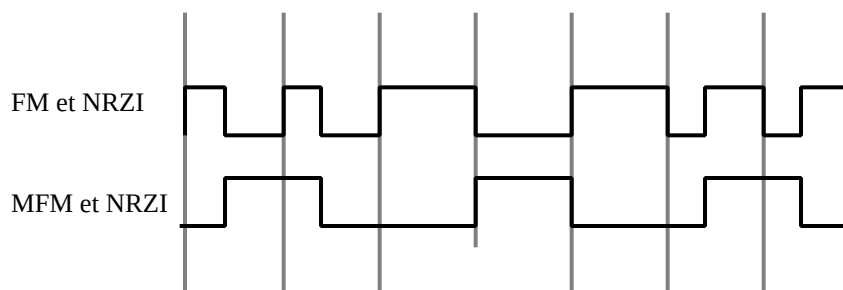
Différentes techniques de codage ou de magnétisation sont utilisées, dans le but d'assurer une plus grande intégrité et une plus grande densité de l'information. Quatre techniques sont couramment utilisées, surtout sur les disques. Les trois utilisent la technique d'enregistrement NRZI, où un "1" est représenté par un renversement du flux magnétique.

Dans le codage **FM** (modulation de fréquence), il y a une impulsion d'horloge (renversement de flux) au début de chaque cellule de bit (coïncidant avec le pointillé). Un "1" est représenté par une impulsion (renversement de flux) au milieu de la cellule de bit. Il n'y a pas de renversement de flux dans le cas d'un "0".

Vers 1977, on introduit les techniques dites de "double densité". Dans le **MF**M (modulation de fréquence modifiée), on n'a une impulsion d'horloge que s'il y a deux 0 de suite ou plus. Ceci permet de réduire la durée de la cellule de bit sans augmenter le nombre de renversements de flux par unité de distance. Dans le **M²FM** (modulation de fréquence doublement modifiée ou modifiée par Miller), on n'a une impulsion d'horloge pour un zéro que si le bit précédent était zéro et qu'il n'y avait pas d'impulsion d'horloge pour ce bit. Dans les trois techniques, le contrôleur de disque doit être assez bien synchronisé pour faire la distinction entre une impulsion d'horloge et une donnée.



Techniques de codage



Codage sur le disque

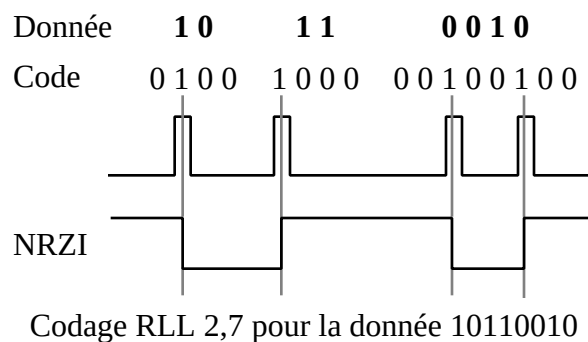
Certains manufacturiers (dont Apple) utilisent une autre technique de codage, le **GCR**, introduit vers 1978. Un autre codage similaire est utilisé depuis quelques années, le codage **RLL** (run-length limited). On retrouve ce dernier sur la plupart des disques durs récents.

Dans le GCR et le RLL, il s'agit de coder l'information de telle sorte qu'il n'y ait jamais ni plus ni moins qu'un certain nombre de zéros qui se suivent, quel que soit le motif de bits à enregistrer, afin de ne pas perdre la synchronisation. Dans le codage RLL 2,7, par exemple, il y a toujours au moins deux zéros et au plus sept zéros entre chaque paire de uns. Dans le RLL 3,9, on trouve entre deux uns au moins trois et au maximum neuf zéros. Le codage RLL est environ 50% plus dense que le MFM.

On trouve à la p. 160 du livre une table de codage GCR. Pour le RLL 2,7, on utilise la table :

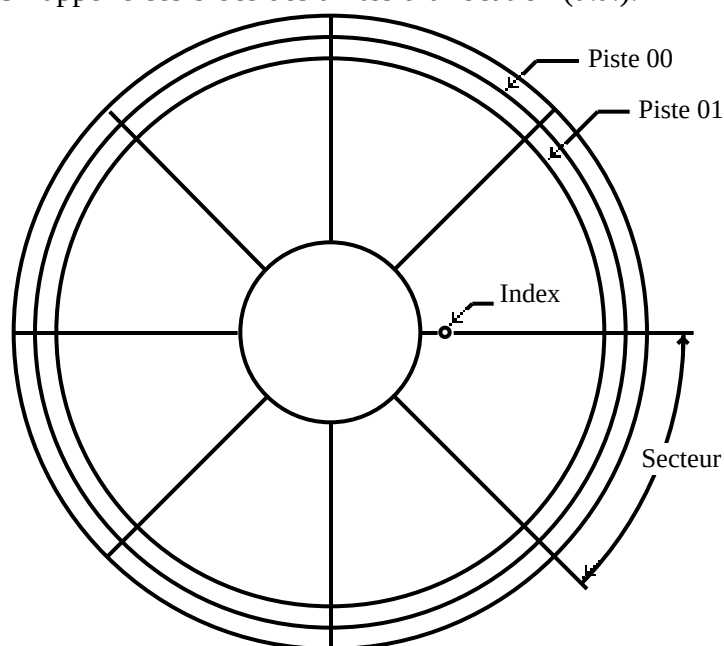
Suite de bits	Code
000	000100
10	0100
010	100100
0010	00100100
11	1000
011	001000
001	00001000

Exemple:



7.5.2 Disques magnétiques

Tel que mentionné à la p. 160 du livre, l'enregistrement sur un disque est effectué le long de pistes concentriques. Ces pistes sont divisées en secteurs. Suivant le système d'exploitation, ces secteurs peuvent être regroupés en blocs logiques de quelques secteurs lors de la lecture et de l'écriture. On appelle ces blocs des unités d'allocation (u.a.).



Découpage d'un disque

Il existe deux techniques de partage du disque en secteurs : par le logiciel ou par le matériel. Le découpage "par matériel" (hard sector) utilise des trous "physiques" dans le disque lui-même pour indiquer le début de chaque secteur. Le découpage "par logiciel" (soft sector) utilise un seul trou (ou même aucun dans certains cas) pour délimiter le début de la piste; les secteurs suivants sont définis par le système d'exploitation qui ajoute par logiciel des informations en tête du premier secteur qui permettent de repérer le début de chacun des

autres secteurs. La sectorisation par matériel a l'avantage de permettre à l'utilisateur de stocker un plus grand nombre de données, car la sectorisation par logiciel nécessite une grande quantité "d'overhead", c.-à-d. d'informations destinées au repérage des différents secteurs. Toutefois, la plupart des disques utilisés par les microsystemes sont des disques formatés par logiciel.

Chaque secteur est divisé en deux champs : le champs d'identification (**ID Field**) et le champs de données (**Data Field**). Tous les champs (ID ou Data) sont séparés par des intervalles (gaps). Chaque champ commence par une zone de synchronisation et un octet de marque d'adresse, et se termine par un **CRC**¹ (Contrôle de redondance cyclique) suivi d'un intervalle. Tous ces champs, sauf celui des données de l'utilisateur sont inscrits sur le disque au moment du formatage. Le champ d'identification permet de savoir exactement de quel secteur il s'agit : n° de piste, n° de secteur, n° de face.

Un gap est un espace entre les champs. Il sert de zone tampon séparant deux champs qui peuvent être écrits séparément. La dimension du gap influence l'intégrité des enregistrements : si le gap est trop court, il y a danger que l'écriture dans un champ de données déborde dans le champ d'identification suivant si le disque tourne trop vite. En revanche, des gaps trop longs diminuent la capacité du disque.

Le champ de synchronisation permet au contrôleur de disque d'ajuster la vitesse de rotation du disque avec précision et de distinguer entre des transitions d'horloge et des transitions de données. Il indique également au contrôleur quelle sorte d'information va venir : champ d'identification, champ de données, etc.

L'octet de marque d'adresse indique où commence un champ et confirme qu'un champ de synchronisation en est bien un et non pas une suite de données qui ressemblerait à un champ de synchronisation. Pour cette raison, l'octet de marque d'adresse est un octet spécial (certaines cellules ne contiennent pas de top d'horloge). On utilise des octets de marquage différents pour indiquer respectivement le début de piste, le début d'un champ d'identification, le début d'un champ de données, le début de chaque champ de données annulées, et le début d'une mauvaise piste.

¹ Le CRC est une technique algorithmique pour vérifier que les données d'un champ sont correctes. Elle est décrite au chapitre 4 du livre de Zanella et Ligier. Deux octets CRC sont calculés par matériel au moment de l'écriture et sont écrits à la suite des données. À la lecture, les octets CRC des données lues sont calculés (également par matériel) et comparés à ceux du disque.

Le débit maximum d'un disque en octets par seconde est donnée par le nombre d'octets par seconde qui passent devant la tête de lecture ou d'écriture. Ce nombre est donné par l'expression :

$$\begin{aligned} \text{nb d'octets par seconde} &= \text{nb d'octets par piste} \times \text{nb de tours par seconde} \\ \text{où :} \\ \text{nb d'octets par piste} &= \text{nb d'octets par secteur} \times \text{nb de secteurs par piste} \\ \text{nb de tours par seconde} &= \frac{\text{vitesse de rotation en tours/min.}}{60} \end{aligned}$$

Donc, plus l'enregistrement sur un disque est dense et plus il tourne vite, plus son débit sera grand.

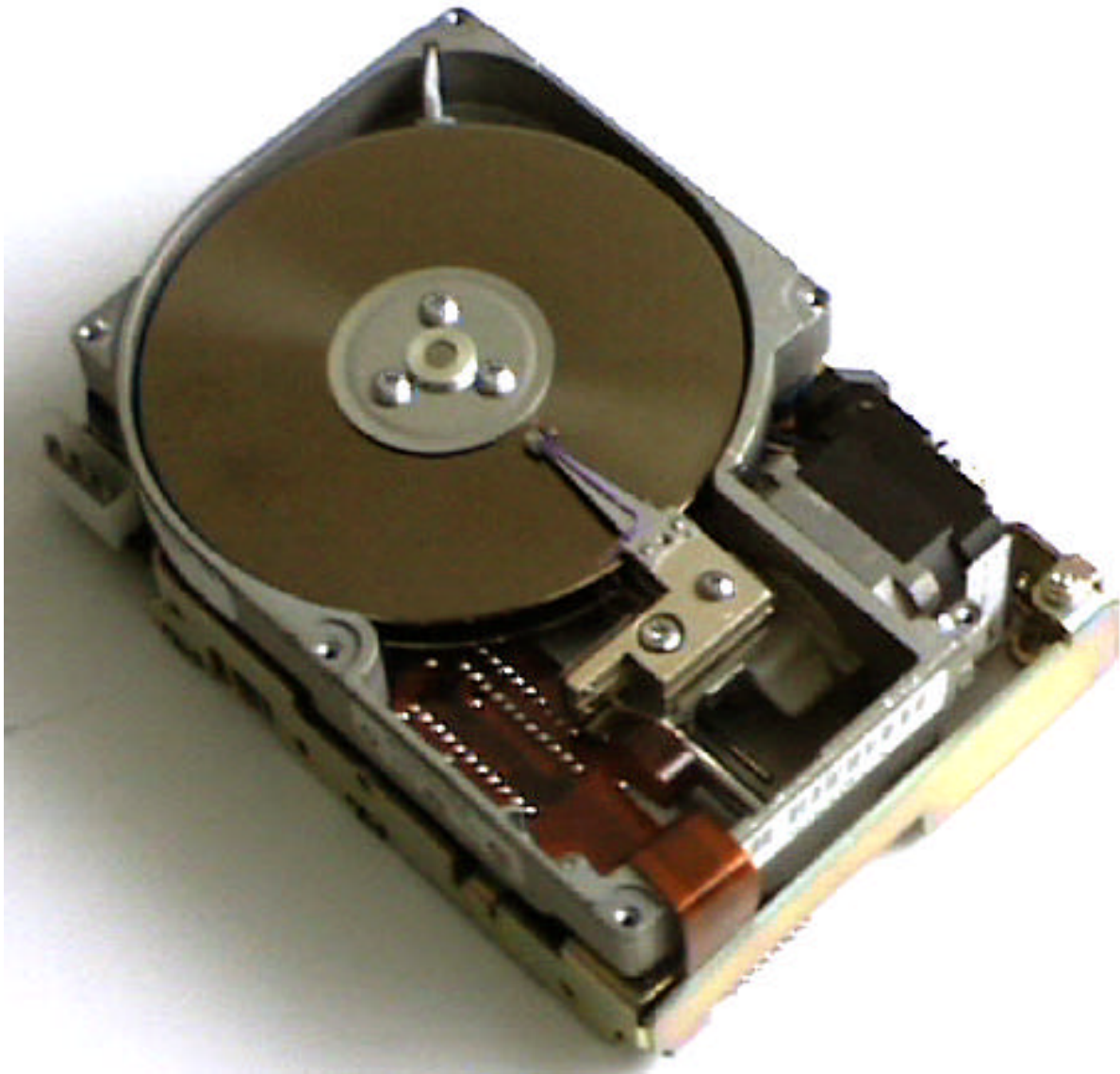
Le temps d'accès d'un disque est le temps pris par la tête de lecture ou d'écriture pour passer d'une piste à l'autre. Il est de l'ordre de 10 ms pour les disques durs et de l'ordre de 30 ms pour les disques souples.

Le temps d'attente d'un disque est le temps moyen qu'on doit attendre avant qu'une information donnée passe devant la tête de lecture ou d'écriture. Ce temps est égal au temps nécessaire pour que le disque effectue une demie rotation :

$$\frac{1}{2 \times \text{nb de tours par seconde}}$$

Disques durs

Ces disques tournant à très grande vitesse (plus de 3600 tours/minute), la tête de lecture ne touche pas au disque mais flotte sur un coussin d'air à environ 0,25 µm de la surface. La surface doit donc être usinée avec une très grande précision et le disque doit être à l'abri de toute poussière.



Coupe d'un disque dur

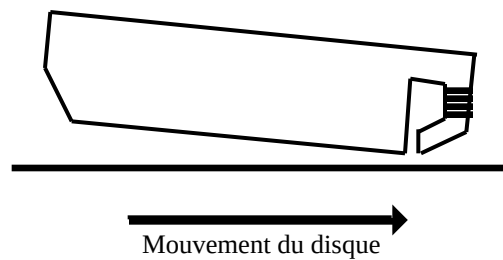


Schéma d'une tête de lecture de disque dur

Le boîtier est étanche et ces disques ne sont habituellement pas accessibles à l'utilisateur. Il existe toutefois des versions dans lesquelles une cartouche étanche amovible permet d'enlever le disque (200 Mo).

Grâce à la grande vitesse de rotation des disques durs, la vitesse de transfert des données est très élevée et dépasse dans certains cas 10 Megabits/s. À ces vitesses, on ne peut faire intervenir le processeur et on lit ou écrit directement en mémoire par DMA (Direct Memory Access).

Disquettes magnétiques

Le disque souple (ou disquette) a longtemps été le dispositif de stockage externe le plus répandu sur les micro-ordinateurs. La disquette standard de 8" (210 mm) de diamètre fut développée par IBM pour son système de saisie de données 3740. Elle fut utilisée pendant un certain temps, pour être remplacée par la disquette de 5 1/4" (5 1/4"). Aujourd'hui, on utilise presque universellement celles de 3 1/2" (89 mm). Les disquettes de 3 1/4" (82 mm) n'ont pas réussi à prendre une part significative du marché. La capacité d'une disquette de 3 1/2" est de 1,44 Mo.

Toutefois, le disque souple est en voie d'être remplacé presque totalement par des média comme le lecteur Zip™ de Iomega. Ces disquettes ont le même format qu'un disque souple mais une capacité de 100 Mo. On les appelle parfois «floptiques» parce qu'elles comportent des pistes de guidage optique. Le taux de transfert est de 1,2 Mo/s et le temps d'accès est de 29 ms. C'est une performance presque comparable à celle d'un disque dur à un coût très bas (environ 250 \$ pour l'unité et 25 \$ par disquette).

7.5.3 Cartouches et bandes magnétiques

On utilise peu les dérouleurs de bande magnétique en micro-informatique. On a plutôt recours aux disques souples et durs. Toutefois, pour effectuer les copies de sécurité, il s'avère souvent intéressant d'utiliser des cassettes de bande magnétique.

Dans les cassettes de format 3M™ (102 × 152 mm ou 4" × 6"), une bande d'une largeur de 0,25" se déroule à 90 ips (inch per second). Le lecteur (streaming-tape drive ou streamer) a souvent le même format que les lecteurs de disques souples ou durs de 5 1/4" demie-hauteur. Ces cassettes peuvent contenir jusqu'à 1020 pieds de bande, et avec une densité d'enregistrement de 20 000 bpi (bits per inch) et un format de 9 pistes, on peut atteindre une capacité de 525 Mo. Il faudrait des centaines de disques souples pour obtenir la même capacité.

Les lecteurs de cassettes de bande magnétique de 8 mm., (parfois connus sous le nom d'*Exabyte*) et utilisant la technologie à têtes rotatives des lecteurs de cassettes vidéo, sont également très répandus. Il s'agit d'ailleurs essentiellement des mêmes cassettes que les

caméras vidéo 8 mm. Ces cassettes sont petites (64×96 mm ou $2\frac{1}{2}'' \times 3\frac{3}{4}''$), mais peuvent atteindre une capacité de 2,5 à 5,6 Go. sur une bande de 112 m.

On trouve finalement des cassettes de 4 mm. semblables aux cassettes DAT (Digital Audio Tape) et pouvant contenir 2 à 4 Go. sur un ruban de 90 m. Ici aussi on fait appel à une tête rotative.

Exercices

- 7.1** a. Déterminer la capacité d'adressage totale en octets d'un processeur 32 bits comportant 32 lignes d'adresses notées A0 à A31.
- b. On désire équiper l'espace d'adressage réel de ce processeur avec 4 mégaoctets de RAM dynamique (DRAM) disposés à partir de l'adresse 00000000. Les puces mémoire sont du type 1 Méga × 1. Déterminer le nombre de puces nécessaire.
- c. Quelle est l'adresse de fin de la zone DRAM en hexadécimal?
- d. Les DRAM ont leurs adresses de ligne et de colonne multiplexées. Déterminer le nombre de lignes d'adresses physiques de chaque puce.
- 7.2** Un ordinateur dont le CPU a une capacité d'adressage de 32 bits est muni d'un cache à correspondance directe de 256 Ko organisé en 64K rangées de 4 octets.
- a. Combien y a-t-il de bits dans l'index?
- b. Combien y a-t-il de bits dans l'étiquette?
- c. À quelle rangée se trouve l'adresse 0x0A001000?
- d. Quelle est la valeur de l'étiquette pour l'adresse 0x0A001000?
- 7.3** Sur la notice d'un constructeur de disques durs, on lit les spécifications suivantes :
- vitesse de rotation : 3600 tours/min.
- nombre de plateaux : 5
- nombre de faces actives : 9
- nombre de cylindres : 1123
- vitesse de transfert de l'interface : 1,86 Mo/s.
- a. Calculer le temps que met le disque pour effectuer un tour complet.
- b. Déduire la capacité linéaire brute par piste, c.-à-d. le nombre d'octets non formatés qu'on peut emmagasiner sur chaque piste.
- c. Déduire la capacité totale du disque non formaté.
- 7.4** Un constructeur de disques durs communique les données suivantes :
- nombre de disques : 2
- nombre de surfaces : 4
- nombre de têtes : 4
- nombre de cylindres : 612
- vitesse de rotation : 3 367 tours/min.
- nombre d'octets par piste : 8 192
- a. Déterminer le débit des données en octets par seconde.
- b. Calculer la capacité brute non formatée du disque.

- c. Lors du formatage, les cylindres de 0 à 7 sont réservés pour la gestion du disque et dans la zone de données, on évalue la perte due aux gaps et aux identifications diverses à 24% de la capacité brute. Déterminer le nombre de secteurs de données de 512 octets qui restent, sachant que chaque piste doit contenir un nombre entier de secteurs.

Chapitre 8

Unité centrale de traitement

8.6 Structure des instructions niveau machine

Le livre mentionne à la p. 183 qu'on préfère aujourd'hui les instructions à une adresse. Dans le cas dans les machines RISC, les instructions d'entrée-sortie sont les seules qui peuvent accéder à la mémoire, et elles sont à une adresse. Les autres instructions sont à trois adresses, ces adresses étant obligatoirement des registres généraux. On trouvera donc des instructions telles que :

add r0, r1, r2

Cette instruction additionne les registres r1 et r2 et place le résultat dans le registre r0.

8.8 registres du CPU

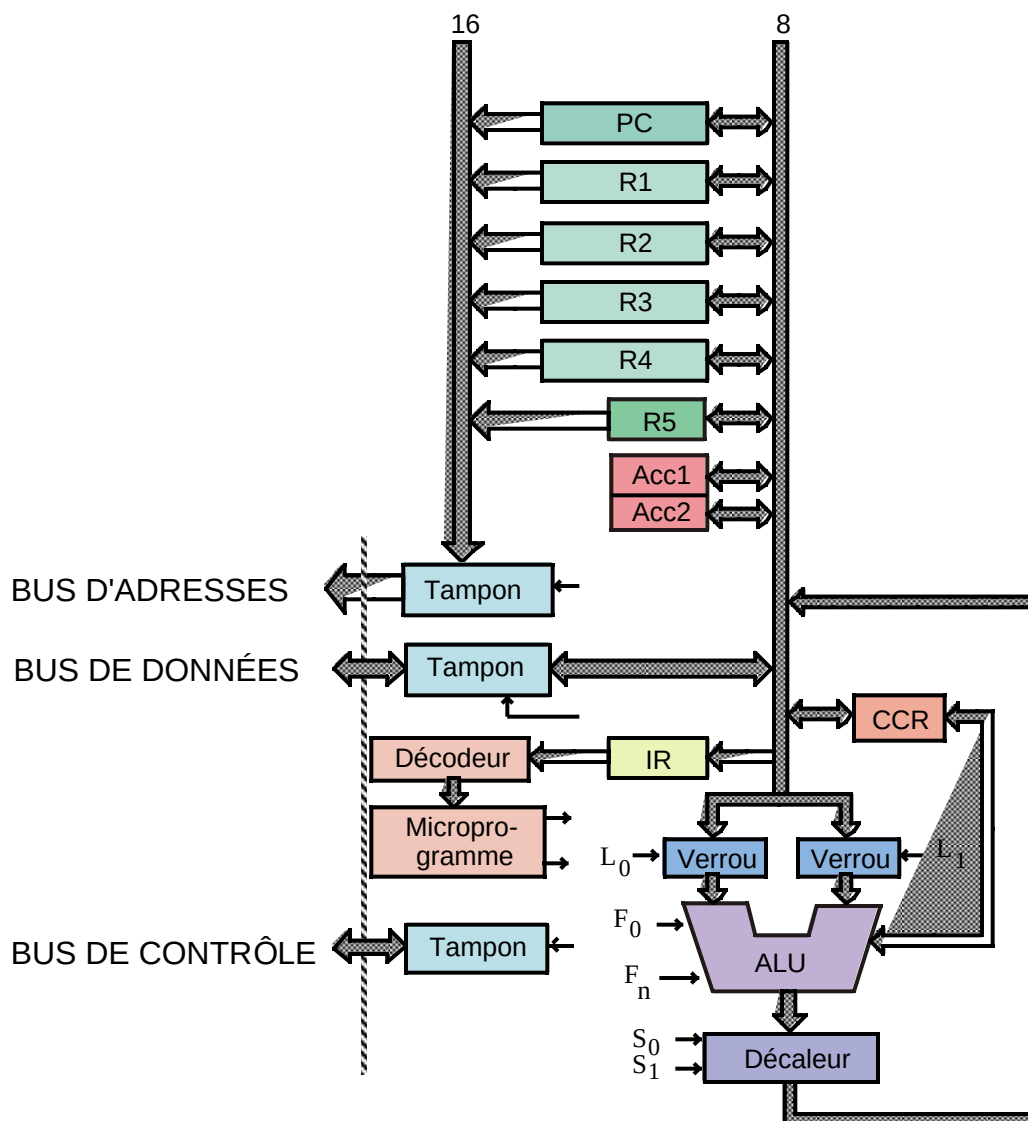
Pour illustrer les registres du CPU de la section 8.8 du livre, nous examinons une architecture minimale de processeur de 8 bits. Les bus de données interne et externe sont de 8 bits. Le bus d'adresses est de 16 bits, ce qui permet une capacité d'adressage de 64 Ko. Les registres internes Acc1, Acc2 et R5 sont de 8 bits. Les autres registres sont des registres généraux de 16 bits et servent surtout à stocker des adresses. Certains d'entre eux peuvent aussi servir de registres d'indice. Un d'entre eux est le pointeur de pile SP.

Ce microprocesseur comporte un UAL de 8 bits et un décaleur, qui constituent l'unité d'exécution. Il comporte de plus un registre d'instructions IR et un décodeur d'instructions, qui avec les informations provenant des entrées de contrôle, fournit les renseignements nécessaires à la ROM contenant le microprogramme. Cette dernière active dans la séquence appropriée les lignes de contrôle internes nécessaires à l'exécution des instructions.

Les accumulateurs Acc1 et Acc2 sont des registres de données de 8 bits dans lesquels on peut effectuer les opérations logiques et arithmétiques.

Le registre R5 est un registre de base.

Un registre de 16 bits PC appelé compteur ordinal ou compteur programme est utilisé par le processeur pour pointer vers l'adresse mémoire devant être lue et décodée par le CPU à l'étape suivante. Il s'incrémente automatiquement à chaque lecture d'un octet à moins qu'une instruction de branchement ou de saut ne l'oblige à prendre une autre valeur particulière. L'instruction suivante à être exécutée est placée dans IR.



Microprocesseur 8 bits minimal

Un registre d'état CCR (Condition Code Register) définit l'état du processeur après chaque opération arithmétique ou logique et permet au CPU de prendre des décisions. Ces décisions se manifestent sous forme de branchements absolus ou relatifs aux diverses sections d'un

programme ou à des programmes différents. Les bits d'état (ou indicateurs) usuels sont donnés ci-dessous. Un dernier indicateur, I, sert à masquer les interruptions matérielles.

C = Retenue (Carry)

V = Dépassement de capacité (Overflow)

Z = Zéro

N = Signe (Negative)

H = Demi-retenue en provenance du bit 3 (Half carry) pour les opération BCD

I = Masque d'interruption

8.9 Adressage des opérandes

La caractéristique principale des machines RISC n'est pas un jeu d'instructions réduit comme l'indiquent leur nom. En effet, un processeur RISC comme le PowerPC MPC604 a plus d'instructions que certains microprocesseurs CISC. Ce qui est réduit, c'est le nombre de modes d'adressage. Les instructions d'entrée-sortie LOAD et STORE sont les seules à pouvoir accéder la mémoire, et ne disposent que de quelques modes d'adressage :

- implicite
- indirection registre (l'adresse se trouve dans un registre)
- immédiat, absolu (adressage direct) ou relatif (adressage relatif)
- indexé (index = registre ou donnée immédiate)

Pour fins de comparaison, certains processeurs CISC ont plus de 20 modes d'adressage.

Exercice

- 8.1** Montrez comment on pourrait réaliser une machine à 0 adresse avec une machine à 1 adresse ne possédant qu'un seul accumulateur. Refaire l'exercice 5 de la p. 191 du livre en utilisant un processeur qui aurait deux modes d'adressage comme suit:

LOAD (SP)+ (dépilement)

qui charge dans l'accumulateur le contenu de l'adresse contenue dans le pointeur de pile SP, et incrémente ensuite SP, et

STORE -(SP) (empilement)

qui décrémente SP et stocke ensuite le contenu de l'accumulateur à l'adresse contenue dans SP.

Chapitre 9

Superordinateurs et microprocesseurs

Processeurs RISC

Les processeurs RISC les plus connus à l'heure actuelle sont :

32 bits :

- Le PowePC MPC-750, aussi appelé G3, de IBM/Motorola, avec une horloge de 466 MHz et le support pour un bus pouvant atteindre 266 MHz, qu'on trouve dans les Power Macintosh.

64 bits

- Le Alpha 21264 de Digital, le microprocesseur le plus rapide au monde avec ses 15,2 millions de transistors, sa fréquence d'horloge de plus de 1 GHz et son bus à 266 MHz. On trouve ce microprocesseur dans les stations de travail de haut niveau, dans les serveurs (UNIX ou NT) et dans les superordinateurs.
- Le MIPS R-12000, qu'on trouve dans les stations de travail Onyx™ et les serveurs Origin™ de Silicon Graphics (SGI) : horloge à 270 MHz, bus à 200 MHz.
- Le UltraSparc-III de Sun à 480 MHz, qu'on trouve dans les stations de travail et les serveurs du même nom.
- Le PA-8500 de Hewlett-Packard, qu'on trouve dans les stations de travail HP-VISUALIZE. Horloge à 440 MHz et bus à 120 MHz.
- Bientôt le IA-64 (Merced) d'Intel/Hewlett-Packard avec plus de 10 M transistors, qui apparaîtra d'abord dans les serveurs.

	SPECint95 ¹	SPECfp95
G3	21.8 @ 466 MHz	12.6 @ 466 MHz
Alpha21264	30 @ 1 GHz	50 @ 1 GHz
PA-8500	16.3 @ 440 MHz	26.1 @ 440 MHz

¹ SPECint95 est une référence standard de performance (version 1995) pour les opérations sur des entiers, basées sur des tests mis au point par le groupe SPEC (System Performance Evaluation Cooperative). SPECfp95 mesure la performance en virgule flottante.

UltraSparc-III	20.2 @ 480 MHz	22.5 @ 480 MHz
R-12000	15.8 @ 270 MHz	25.2 @ 270 MHz

Performance comparative des différents microprocesseurs

Processeurs CISC

Le processeur CISC le plus répandu à l'heure actuelle est sans contredit le Pentium. Le membre le plus récent de la famille est le Pentium III Xeon avec une fréquence d'horloge pouvant atteindre 550 MHz et un bus à 100 MHz. Ses performances sont 22.5 SPECint95 et 16 SPECfp95 @ 550 MHz.

Le Merced d'Intel

Il importe ici de mentionner le processeur IA-64 d'Intel, aussi connu sous le nom de Merced, qui remplacera probablement un jour la famille Pentium. C'est un processeur RISC de 64 bits, quoiqu'Intel prétend qu'il va plus loin que le RISC grâce à l'utilisation de la technologie EPIC (Explicitly Parallel Instruction Computing).

Le IA-64 possède :

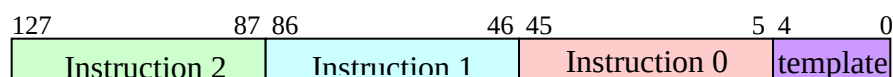
- Plus de 10 millions de transistors.
- Quatre types d'unités d'exécution : I-Unit pour les entiers, M-unit pour la mémoire, F-unit pour la virgule flottante, B-unit pour les branchements.
- 128 registres généraux de 64 bits $r0$ à $r127$ pour les entiers. Le registre $r0$ est en lecture seulement et contient la valeur zéro.
- 128 registres de virgule flottante de 82 bits $f0$ à $f127$. Les registres $f0$ et $f1$ sont en lecture seulement et contiennent les valeurs +0.0 et +1.0 respectivement.
- 64 registres de prédicat $p0$ à $p63$ qui contrôlent l'exécution conditionnelle des instructions et les branchements conditionnels. Le registre $p0$ est en lecture seulement et contient la valeur vrai (1).
- 8 registres de branchements $b0$ à $b7$, utilisés pour spécifier l'adresse cible des branchements indirects.
- L'espace pour 128 registres d'application $ar0$ à $ar127$ qui supportent diverses fonctions, entre autres certains registres de l'architecture IA-32. Plusieurs de ces registres sont réservés pour usage futur.
- Un registre de pointeur d'instruction ip de 64 bits.

Le IA-64 supporte l'exécution spéculative, i.e. charge simultanément dans son pipeline les deux branches d'un branchement tant que la destination du branchement n'est pas connue. Ceci élimine à toutes fins pratiques les branchements et les délais qu'ils provoquent en cas de mauvaise prédiction. Il supporte également la prédication, mécanisme par lequel on indique au processeur quelles instructions peuvent être exécutées en parallèle.

Le IA-64 supporte également les instructions de l'architecture IA-32 (Pentium et al).

En mode 64 bits, comme dans les processeurs RISC, seules les instructions load (ld), store (st) et les instructions de sémaphore (cmpxchg, xchg, fetchadd) peuvent accéder à la mémoire.

Les instructions du IA-64 sont groupées en paquets (bundles) de 128 bits (voir l'approche VLIW à la p. 209 du livre de Zanella et Ligier) contenant trois instructions de 41 bits. Les 5 derniers bits contiennent le champ template.



Le champ template précise deux propriétés : des arrêts (stops) dans le paquet courant et le mapping des instructions avec les types d'unité d'exécution. Un arrêt indique que une ou plus instructions avant l'arrêt ont une dépendance de ressource avec une ou plus instructions après l'arrêt. Au moment du codage, un double point-virgule (;) après une instruction indique un arrêt. C'est le programmeur ou le compilateur qui doit déceler ces dépendances.

Chaque instruction a la syntaxe suivante :

[qp] mnémonique [.comp] dest=srcs

qp désigne un registre de prédicat. La valeur du prédicat détermine si les résultats de l'instruction sont en matériel ou supprimés. Quand la valeur d'un registre de prédicat est vraie (1), l'instruction s'exécute, ses résultats sont enregistrés et toute exception qui se produit est traitée normalement. Quand la valeur est fausse (0), les résultats ne sont pas enregistrés et aucune exception n'est déclenchée.

mnémonique désigne un nom qui identifie de façon unique une instruction IA-64.

comp désigne un ou plusieurs completeurs d'instruction. Ces completeurs indiquent des variations optionnelles appliquées à un mnémonique de base. Ils suivent le mnémonique et sont séparés par des points.

dest spécifie le(s) opérande(s) destination, i.e. le(s) résultat(s) de l'instruction.

srcs spécifie les opérandes source. La plupart des instruction IA-64 en ont au moins deux.

Exemples

```
ld8    r1=[r5]      // lire 8 octets à l'adresse contenue dans r5 et placer dans r1
add    r3=r1, r4     // r3 = r1 + r4
br.cond.dptk L1      // saut conditionnel vers L1 en utilisant la prédiction matérielle
                        // dynamique supposant que le branchement sera pris
                        // (dynamic taken)
shr    r7=r3,r87     // décaler r3 du nombre de bits indiqué par r87 et placer
                        // résultat dans r7
st4    [r89]=r16     // enregistrer r16 à l'adresse contenue dans r89
ld8.s  r1=[r5]       // lecture spéculative de 8 octets
ldfs   f4=[r7],4     // lire un nombre de virgule flottante de simple précision
                        // à l'adresse contenue dans r7 et stocker dans f4, puis
                        // incrémeter r7 de 4.
```

Chapitre 10

Entrées/sorties

10.5.3 Le bus USB

Le bus USB (Universal Serial Bus) a été mis au point par 7 compagnies : Compaq, DEC, IBM, Intel, Microsoft, NEC et Northern Telecom. C'est un bus série externe destiné aux PC. Il supporte des taux de transfert de données atteignant 1.5 Mo/s (12 Mbits/s).

Sur un même port USB, on peut connecter jusqu'à 127 périphériques : clavier, souris, imprimantes, scanners, appareils à photo numériques, modems, joysticks, téléphones, microphones, haut-parleurs, lecteurs de disquettes, etc. Ces périphériques peuvent être chaînés l'un à l'autre ou reliés à des concentrateurs d'extensions (hubs) qui sont des boîtiers externes, intermédiaires entre le PC et les périphériques.

Le bus USB supporte le Plug and Play et le Hot Plug and Play. Il est prévu pour remplacer les ports séries et parallèles des micro-ordinateurs. Il permet d'ajouter des périphériques sans à avoir à ouvrir l'ordinateur pour y ajouter une carte, ajuster des commutateurs ou des cavaliers (jumpers) sur la carte, ou à installer des fichiers de configuration.

- Il n'y a qu'un type câble, valable pour tous les périphériques.
- Les périphériques sont alimentés à partir du câble.
- Le système supporte des périphériques temps réel (téléphone, son, etc.).
- Les périphériques peuvent être installés pendant que l'ordinateur est en marche.
- Il n'est pas nécessaire de redémarrer l'ordinateur après l'installation d'un nouveau périphérique.
- Il n'y a pas de communication d'un périphérique à l'autre.

Le câble est formé de 4 fils. Deux servent à l'alimentation en puissance (+5 V), les deux autres servent aux données. Un 0 est une transition de tension, et un 1 est l'absence de transition, de sorte qu'une longue séquence de zéros génère un flux d'impulsions régulières (codage NRZI). Quand un nouveau périphérique est branché, le hub racine détecte cet événement et interrompt le système d'exploitation. Ce dernier interroge alors le périphérique pour savoir ce que c'est et de quelle largeur de bande USB il a besoin. Si le système d'exploitation décide qu'il y a assez de largeur de bande pour le périphérique, il lui attribue

une adresse unique (1 à 127) et transmet cette adresse et d'autres informations aux registres de configuration du périphérique



Les deux extrémités d'un câble USB..

Les cartes non initialisées partent avec l'adresse 0, afin de pouvoir être adressées. Plusieurs dispositifs USB contiennent leur propre hub de sorte qu'on peut leur ajouter d'autres périphériques USB. Le système USB peut être représenté comme un ensemble de pipes de bits entre le hub racine et les périphériques d'entrée/sortie. Chaque dispositif peut subdiviser son pipe en un maximum de 16 sous-pipes pour différents types de données (e.g. audio et vidéo). À l'intérieur de chaque pipe ou de chaque sous-pipe, les données vont du hub racine au dispositif ou en sens inverse. À chaque milliseconde précisément, le hub racine transmet un frame pour maintenir tous les périphériques synchronisés dans le temps. S'il n'y a pas d'autre travail à faire, ce frame consiste en un paquet SOF (Start of Frame).

Il y a quatre sortes de frames. Les frames de contrôle, les frames isochrones, les frames en vrac (bulk) et les frames d'interruption.

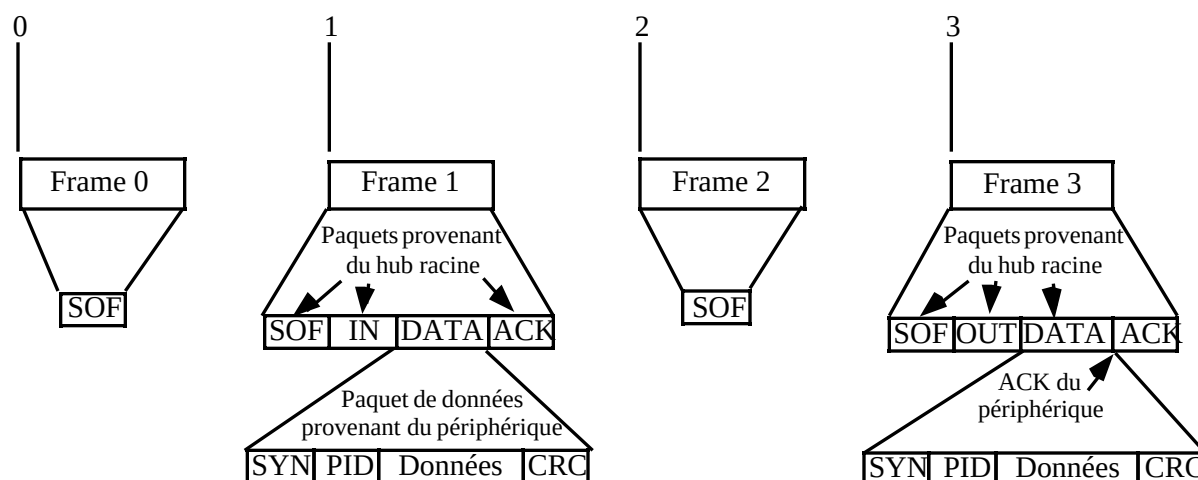
Les frames de contrôle servent à configurer des périphériques, à leur donner des commandes et les interroger sur leur statut. Les frames isochrones servent aux périphériques temps réel comme les microphones, les haut-parleurs, et les téléphones, qui ont besoin d'envoyer ou de recevoir des données à des intervalles de temps précis. Ils ont un délai prévisible, mais ne permettent pas de retransmission en cas d'erreur.

Les frames en vrac servent pour des transferts de masse en direction ou en provenance d'un périphérique sans exigence de temps réel, comme des imprimantes.

Finalement, les frames d'interruption sont nécessaires parce que le USB ne supporte pas les interruptions. Par exemple, au lieu que le clavier cause une interruption chaque fois qu'on appuie sur une touche, le système d'exploitation peut le scruter chaque 50 ms pour récupérer les touches accumulées depuis la fois précédente.

Un frame consiste en un paquet ou plus, avec possibilité de paquets dans chaque direction. Il existe quatre types de paquets : token donnée, poignée de main, et spéciaux. Les paquets token sont du hub racine vers un périphérique et servent au contrôle du système. Parmi ces paquets on a des tokens tels que SOF, IN, OUT, ACK, SETUP. Le paquet SOF est le premier de chaque frame et indique le début du frame. S'il n'y a rien à faire, c'est le seul paquet du frame.

Le paquet IN est une scrutation, demandant à un périphérique de retourner certaines données. Des champs dans le paquet IN indiquent quel pipe de bits est en train d'être scruté, de sorte que le dispositif sait quelles données retourner (s'il a des streams multiples). Le token OUT annonce que des données pour le périphériques suivent. Le token SETUP sert à la configuration d'un dispositif. Le token DATA permet de transmettre jusqu'à 64 octets d'information dans l'une ou l'autre direction. Le format d'un paquet de données est illustré ci-dessous. Il consiste en un champ de synchronisation e 8 bits, d'un champ de type de paquet de 8 bits PID, des données, et d'un CRC (Cyclic Redundancy Code) de 16 bits pour détecter les erreurs. Trois types de paquets de poignée de main sont définis : ACK (Acknowledge) pour indiquer que le paquet de données précédent a été correctement reçu, NAK (Negative acknowledge) pour indiquer qu'il y a eu erreur de CRC) et STALL (attendre, dispositif occupé).



Exemples de frames USB

Les paquets eux-mêmes sont transmis en série avec un codage NRZI (le signal change seulement quand la donnée est un 0) à l'un de deux taux : 12 Mbits/s pour les transmissions à pleine vitesse, ou 1.50 Mbits/s pour les périphériques très lents comme les souris.

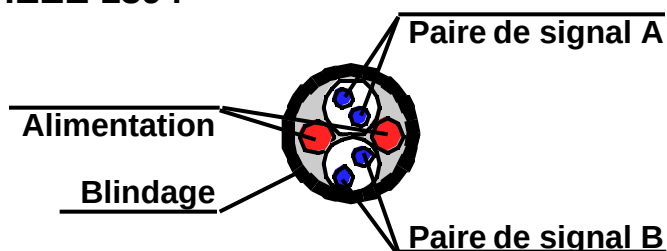
Pour plus d'information, visitez le site <http://www.usb.org>.

Le bus Firewire

Le bus IEEE-1394, aussi connu sous le nom de FireWire, peut relier 63 périphériques, à des vitesses pouvant atteindre 400 Mbits/s. Il supporte le transport de données asynchrone pour les périphériques utilisant des technologies plus anciennes comme les imprimantes et les modems, et le transport isochrone pour les périphériques numériques temps réel, qui garantit la livraison des données dans le temps prescrit. Par exemple, pour des données vidéo de bonne qualité, on a besoin de 30 trames à la seconde contenant chacune 680 par 480 pixels, et si chaque pixel nécessite 24 bits, on a besoin d'une bande passante de 224 Mbits/s, ce qui dépasse largement la capacité du bus USB.

Le câble contient 6 fils : deux pour l'alimentation en puissance du périphérique (8 à 40 VDC), et deux paires torsadées pour la transmission et la réception des données.

Câble IEEE 1394



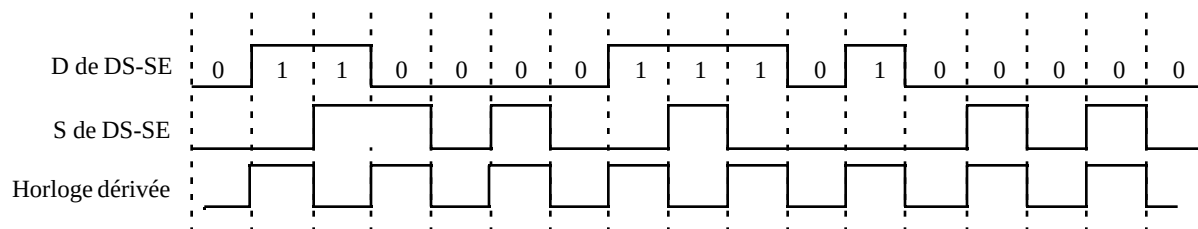
Les deux extrémités d'un câble FireWire.

Le Firewire utilise des «Packet frames» de 125 microsecondes. Chacun est découpé en tranches de temps isochrones pour les canaux en opération. La partie d'un frame qui n'est pas utilisée par des canaux numériques peut être utilisée pour le transport asynchrone.

Le périphérique demande un canal isochrone avec une bande passante spécifique. Lorsque sa demande est accordée, il transmet son identification de canal suivie des données du paquet.

Le récepteur vérifie le no. d'identification du canal et n'accepte que les données ayant le bon numéro. Les applications sont responsables de déterminer combien de canaux sont nécessaires et la bande passante nécessaire.

FireWire utilise le mode NRZ avec codage Data-Strobe. à 400 Mbits/s. Le strobe S ne change que si la donnée D ne change pas. À la réception, on régénère l'horloge originale en tenant compte de tous les changements de D et de S.



Encodage DS

Ce codage rend les signaux DS «autobaud», c'est-à-dire que le récepteur peut récupérer l'horloge, quelle que soit la vitesse de transmission. L'émetteur peut changer de vitesse ou même arrêter, le récepteur s'en rendra compte.

Exercices

10.1 Un terminal monochrome possède un tube capable d'afficher 600 x 800 pixels.

- Évaluer la taille mémoire en octets nécessaire pour stocker une image complète d'écran, sachant qu'un pixel équivaut à un bit en mémoire.
- On désire améliorer ce terminal en offrant à l'utilisateur la possibilité d'avoir 16 échelles de gris pour chaque pixel. Quelle serait alors la quantité de mémoire nécessaire?
- Si un tel terminal est en couleur et que chacune des 3 couleurs fondamentales possède également 16 échelles d'intensité, évaluer la quantité de mémoire nécessaire

pour stocker une image complète d'écran et évaluer la palette de couleurs obtenue (nombre de couleurs possibles).

- On désire faire de l'animation avec un terminal graphique couleur connecté à un ordinateur hôte via une liaison à déterminer. Les caractéristiques du terminal sont : résolution : 300 x 400
nombre d'images par seconde : 30
balayage : entrelacé
palette de couleurs : 8.
On désire obtenir au moins 5 images différentes par seconde pour avoir une vitesse d'animation acceptable. Les images complètes sont calculées par l'ordinateur hôte et envoyées au terminal.

- a. Déterminer la vitesse minimale de la liaison en bits par seconde pour répondre à ces exigences.
- b. On s'aperçoit statistiquement que les modifications entre les images ne représentent qu'environ un cinquième des pixels. On adopte alors une autre stratégie, qui consiste à ne transmettre que les pixels qui sont modifiés d'une image à l'autre, en leur associant une adresse codée sur 18 bits, les informations sur les 8 couleurs associées à un pixel étant codées sur les 3 bits restants. La donnée complète pour chaque pixel entre à l'intérieur de 3 motifs de 7 bits ($18 + 3 = 21 = 7 \times 3$).

Évaluer à nouveau la vitesse correspondante de la liaison entre le terminal et l'ordinateur hôte.

- 10.3 Une imprimante au laser est capable de débiter au maximum 10 pages à la minute. Chaque page comporte 60 lignes de texte et chaque ligne comporte 80 caractères. Chaque caractère est codé sur 8 bits. Déterminer la vitesse minimale en bits par seconde de la liaison connectant l'ordinateur à cette imprimante capable d'assurer un fonctionnement continu de l'imprimante.

Chapitre 11

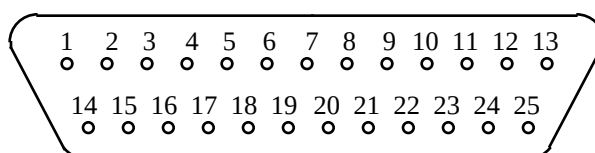
Téléinformatique

11.3 Voies de transmission

11.3.2 Modes de transmission et synchronisation

Transmission parallèle

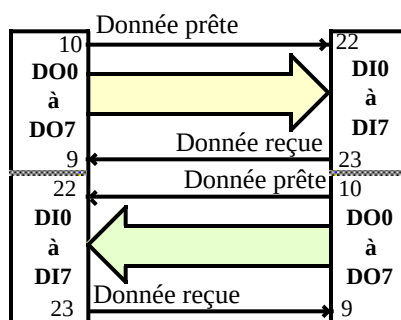
La transmission parallèle est généralement plus rapide que la transmission série, puisqu'avec 8 fils on envoie un octet à la fois au lieu d'un seul bit. De nombreux périphériques utilisent des interface parallèles utilisant le même connecteur DB-25 que l'interface série RS-232-C (voir transmission série plus loin) mais avec des brochages différents.



Connecteur DB-25

Il y a 8 fils pour les données de sortie (DO0 à DO7) et 8 fils pour les données d'entrée (DI0 à DI7). On utilise habituellement un système de poignée de main à deux fils dans chaque direction. Le signal *Donnée prête* porte les noms OSTB ou IRDY (Output Strobe/Input Ready) selon qu'on est à l'extrémité du transmetteur ou du récepteur, et le signal *Donnée reçue* ceux de ORDY ou ISTB (Output Ready/Input Strobe). Le brochage est le suivant :

1	DO7	→	14	DI7	←
2	DO6	→	15	DI6	←
3	DO5	→	16	DI5	←
4	DO4	→	17	DI4	←
5	DO3	→	18	DI3	←
6	DO2	→	19	DI2	←
7	DO1	→	20	DI1	←
8	DO0	→	21	DI0	←
9	ORDY/ISTB	←	22	IRDY/OSTB	←
10	OSTB/IRDY	→	23	ISTB/ORDY	→
11-13	GND		24-25	GND	



Connexion parallèle bidirectionnelle

Le transmetteur place une donnée (octet) sur les lignes DO0 à DO7 et active le signal *Donnée prête*. Ceci indique au récepteur qu'une donnée est présente sur ses entrées DI0 à DI7. Quand le récepteur a lu la donnée, il active le signal *Donnée reçue*, ce qui indique au transmetteur qu'il peut envoyer une autre donnée.

Transmission série

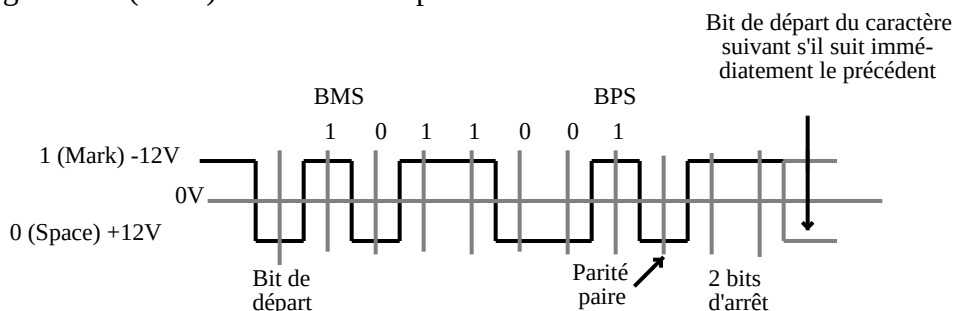
Deux modes de communication série sont utilisés pour transmettre des caractères. La communication asynchrone et la communication synchrone.

Avec la transmission **asynchrone**, l'intervalle de temps entre deux caractères n'est pas fixe, bien que l'intervalle de temps entre deux bits consécutifs d'un caractère soit fixe. Par exemple, un usager qui tape sur un clavier ne tape pas à une vitesse uniforme, de sorte que l'intervalle de temps entre deux caractères successifs n'est pas constant. Cette variation de vitesse pose un problème : comment le récepteur peut-il reconnaître le premier bit d'un caractère? Par exemple, un caractère consistant uniquement de zéros serait complètement invisible. Ou encore, un caractère composé d'un 1 suivi de 7 zéros ne pourrait pas être distingué d'un caractère consistant de 7 zéros suivis d'un 1.

Pour permettre au récepteur de reconnaître le début d'un caractère, un **bit de départ** (start bit) est transmis immédiatement avant chaque caractère. Pour une plus grande fiabilité, 1, 1,5 ou 2 bits de fin de caractère ou **bits d'arrêt** (stop bits) sont transmis immédiatement après chaque caractère. Normalement la ligne de transmission est maintenue à l'état bas (1 logique ou MARK) quand on ne transmet pas de données, de sorte que le bit de départ est un niveau haut (0 logique ou SPACE). Les bits d'arrêt sont donc des niveaux bas pour permettre de les distinguer du bit de départ. Un temporisateur est activé dans le récepteur au moment de l'arrivée du bit de départ, lui permettant d'échantillonner les bits du signal aux bons moments.

Un bit de parité est souvent ajouté en vue de détecter les erreurs. Cette parité peut être choisie paire ou impaire. En parité paire, par exemple, ce bit prendra la valeur nécessaire pour que l'ensemble (bits du message et bit de parité) contienne un nombre pair de 1. Si une erreur de transmission se produit, un bit changera de valeur de sorte que la parité changera. Le récepteur détectera donc une erreur de parité et en avertira l'ordinateur.

Exemple : transmission du caractère "M" (ASCII 0x4D) sur 7 bits; remarquer que le bit le moins significatif (BMS) est transmis le premier :



Transmission série du caractère "M"

À retenir :

- Au repos, la ligne donnée est au niveau 1 (Mark = -12 V).
- Le bit de départ est généralement unique. L'horloge du récepteur synchronise sur l'horloge de l'émetteur à chaque bit de départ.
- L'échantillonnage des données à la réception s'effectue au milieu d'un bit.
- Les données arrivent de façon asynchrone, mais la détection est "synchrone" à l'intérieur d'un motif. On peut donc tolérer une certaine "discordance" entre l'horloge de l'émetteur et celle du récepteur.
- C'est le bit le moins significatif qui est transmis le premier.

Les vitesses de transfert les plus courantes pour la transmission asynchrone sont 110, 300, 600, 1200, 2400, 4800, 9600, 19 200, 28 800, 38 400 et 57 600 bits par seconde. Si deux bits d'arrêt sont utilisés, un caractère ASCII de 7 bits + 1 bit de départ + 1 bit de parité + 2 bits

d'arrêt, nécessite 11 bits. Dans ce cas particulier, 110 bauds correspondent donc à 10 caractères/s.

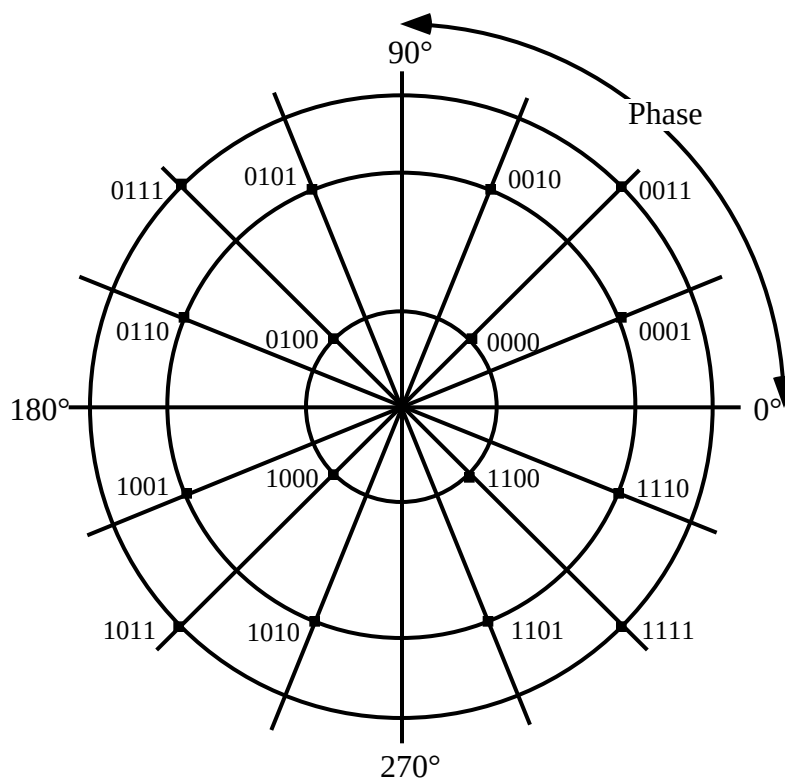
11.4 Transmission analogique et modulation

11.4.2 Modulation d'amplitude, de fréquence et de phase

Contrairement à ce qui est écrit dans le livre de Zanella et Ligier à la p. 276, sur les lignes téléphoniques on obtient actuellement des vitesses de transmission atteignant 28 800 bps. On peut toutefois atteindre des vitesses plus élevées (jusqu'à 115 400 bps) en utilisant la compression de données.

Pour éviter des différences de phase trop petites lorsqu'on augmente le nombre de bits par signal en modulation de phase, une solution consiste à utiliser la modulation *Quadrature Amplitude Modulation (QAM)*, qui combine des variations de phase et d'amplitude.

Dans le système UIT (Union Internationale des Télécommunications, anciennement connu sous le nom de CCITT : Comité Consultatif International pour le Téléphone et le Télégraphe) on obtient 9600 bps avec une vitesse de signalisation de 2400 bauds au moyen de 12 valeurs de phase et de 3 valeurs d'amplitude représentées à la figure suivante, où le rayon indique l'amplitude :



Modulation QAM

Le modem par ligne téléphonique a probablement atteint sa vitesse limite. À l'avenir, on utilisera donc de plus en plus le modem câble, capable de performances 100 fois meilleures.

Exercices

- 11.1** On transmet le caractère 'k' par un port RS-232-C réglé sur 7 bits, parité impaire et 1 bit d'arrêt. Dessinez la forme de la tension en fonction du temps.
- 11.2** a. Calculez la durée en microsecondes d'une cellule binaire d'une liaison série transmettant à 38 400 baud.
- b. Le format de transmission est du type : 1 bit de départ, 8 bits de donnée, 1 bit de parité, 1 bit d'arrêt. Déterminer le nombre effectif d'octets transmis par seconde si on suppose que la ligne opère de façon continue.

Chapitre 12

Systèmes d'exploitation

12.5 Gestion de la mémoire centrale

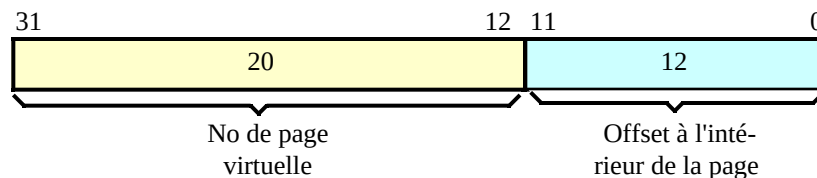
12.5.5 Notion de mémoire virtuelle

Exemple

Nous présentons ici quelques valeurs numériques pour permettre une meilleure compréhension de la gestion de mémoire virtuelle.

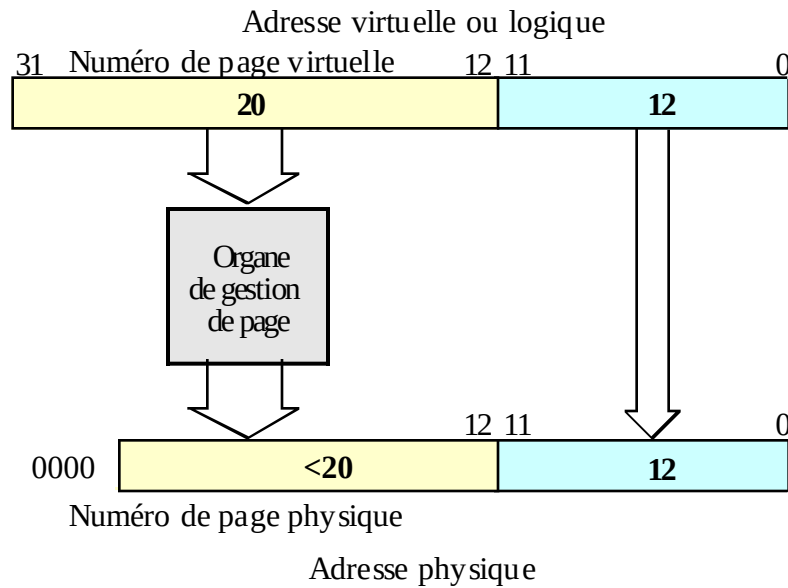
Supposons qu'on désire avoir une mémoire virtuelle de 256 Mo dans un ordinateur doté de 32 Mo de mémoire centrale (RAM) et ayant un bus d'adresses de 32 bits. Supposons que cette mémoire est découpée en pages de 4 Ko (la taille des pages est souvent choisie comme la taille de l'unité d'allocation du disque dur pour permettre une meilleure performance). Il faudra évidemment que le disque dur dispose de 256 Mo d'espace libre.

L'adresse virtuelle de 32 bits (émise par le processeur) sera donc découpée en deux champs : un champ de 12 bits correspondant à l'adresse (ou offset) à l'intérieur d'une page de 4 Ko ($2^{12} = 4 \text{ K}$) et un champ de 20 bits correspondant au numéro de page virtuelle.



Découpage d'une adresse de 32 bits

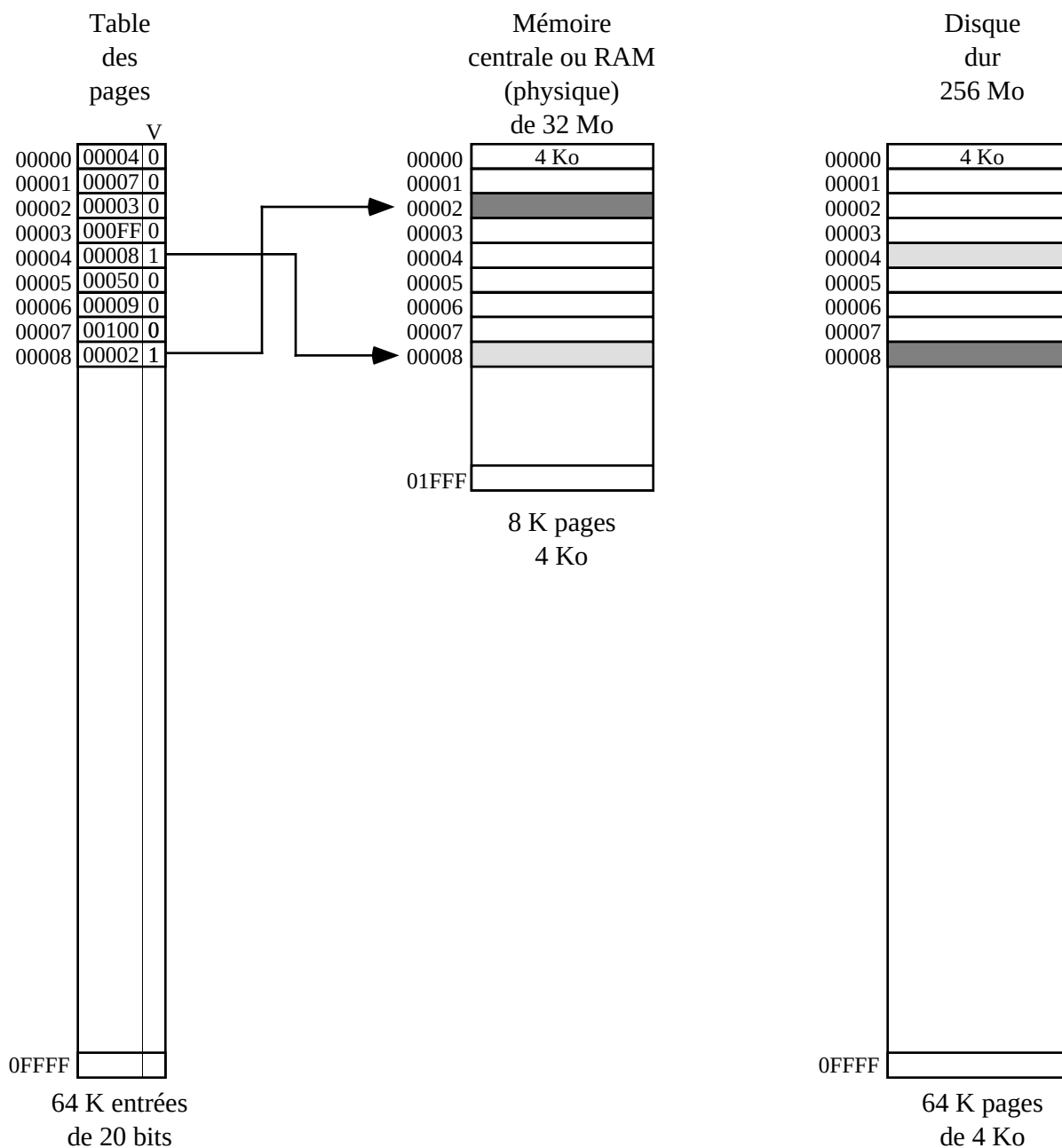
Le processus de mapping (réalisé par le Memory Management Unit ou MMU) consiste à fournir pour chaque numéro de page virtuelle le numéro de page physique correspondant afin de savoir dans quelle page de la mémoire centrale une page virtuelle donnée a été placée :



Mapping d'une adresse logique en adresse physique

Dans notre exemple, le numéro de page virtuelle va de 0x00000 à 0x0FFFF (il y a 65536 pages de 4 Ko dans 256 Mo). Le numéro de page physique va de 0x00000 à 0x01FFF (il y a 8192 pages de 4 Ko dans 32 Mo).

La traduction de numéro de page virtuelle en numéro de page physique s'effectue à l'aide d'une table. Le numéro de page virtuelle est utilisé comme index dans cette table, et le contenu de chaque rangée de la table est le numéro de page physique correspondant. Un bit de validité V indique si la page en question est effectivement en mémoire physique. En effet, il y a beaucoup plus de pages virtuelles possibles que de pages physiques, elles ne peuvent donc pas se trouver toutes en même temps dans la mémoire physique.



Dans l'exemple ci-dessus, on voit que la page virtuelle no 4 a été placée en page 8 de la mémoire physique, tandis que la page virtuelle no 8 a été placée en page 2 de la mémoire physique.

Donc le processeur, s'il désire accéder à la page 4, doit d'abord effectuer une lecture à la rangée 4 de la table des pages pour connaître le numéro de la page physique dans laquelle l'information qu'il désire se trouve. Il peut ensuite accéder aux données en question en se rendant à la page 8 de la mémoire physique.

Par contre, si le processeur avait besoin d'accéder à la page virtuelle no 2, il s'apercevrait que son bit V est 0. Ceci veut dire que cette page n'est pas en mémoire physique. Il faudrait alors aller lire la page 2 du disque, la placer dans une page de la mémoire physique, puis inscrire dans la rangée 2 de la table des pages à quelle page physique on l'a insérée, puis mettre à 1 le bit V de la rangée 2.

Comme pour les mémoires caches, il existe différents algorithmes pour déterminer l'endroit de la mémoire physique où sera placée une nouvelle page virtuelle. Il peut s'agir d'un remplacement aléatoire, ou de l'algorithme LRU (Least Recently Used), qui choisit la page qu'on n'a pas utilisée depuis le plus longtemps. Dans ce cas, il faut ajouter dans la table des pages des bits supplémentaires pour indiquer l'âge des pages.

Est-ce à dire que la mémoire virtuelle est deux fois plus lente que la mémoire physique, puisqu'il faut effectuer deux lectures pour accéder à l'information ? En principe oui, mais en pratique, le MMU contient une mémoire cache dans laquelle il garde en mémoire les derniers numéros de page physique accédés. Dans le cas particulier de la mémoire virtuelle, cette mémoire cache s'appelle TLB (Translation Lookahead Buffer). Trois cas peuvent se présenter :

- Si le numéro de page physique est dans le cache, il n'y a aucune perte de temps.
- S'il y a échec de cache, on doit aller lire dans la table des pages et placer l'information dans le cache. Il y a alors perte d'un cycle mémoire.
- S'il y a échec dans la table des pages, il y a perte de temps beaucoup plus considérable, puisqu'il faut alors aller lire une page de 4 Ko sur le disque, la placer en mémoire, puis mettre à jour la table des pages et le cache du MMU.

Notez finalement que les rangées de la table des pages peuvent contenir des informations supplémentaires, telles que l'adresse de la page sur le disque en numéro de cylindre et de secteur. De plus, un bit supplémentaire indique si la page physique a été modifiée en cours d'utilisation. Si c'est le cas, il faudra la copier sur le disque avant de la remplacer.

12.7 Gestion des fichiers

À titre d'exemple nous étudierons un système de gestion de fichiers maintenant dépassé mais encore utilisé, celui des disques souples PC-DOS ou MS-DOS (Microsoft Corp.).

le contenu des premières pistes d'un disque de 3 1/2" quadruple densité (1,44 M) :

		Piste		
		0	1	2
Secteur	1	BOOT	FAT2	u.a. 5
	2	FAT1	DIR	6
	3	FAT1	DIR	7
	4	FAT1	DIR	8
	5	FAT1	DIR	.
	6	FAT1	DIR	.
	7	FAT1	DIR	.
	8	FAT1	DIR	.
	9	FAT1	DIR	.
	10	FAT1	DIR	.
	11	FAT2	DIR	.
	12	FAT2	DIR	.
	13	FAT2	DIR	.
	14	FAT2	DIR	.
	15	FAT2	DIR	.
	16	FAT2	u.a. 2	Début des données de l'utilisateur
	17	FAT2	u.a. 3	
	18	FAT2	u.a. 4	

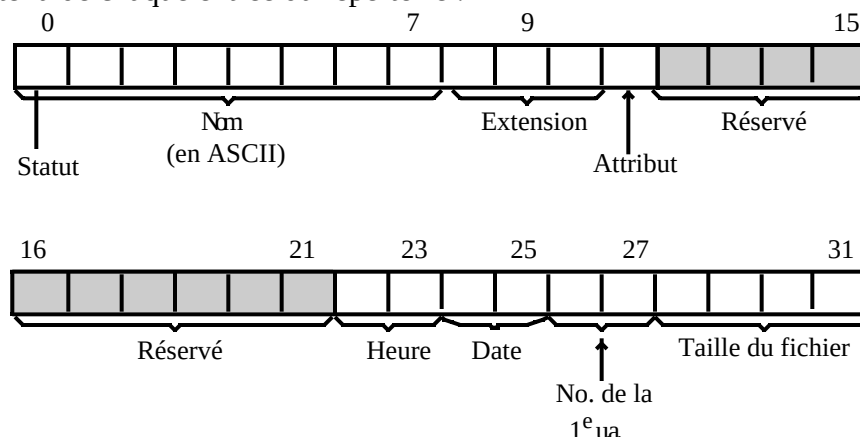
Organisation d'une disquette MS-DOS

Le premier secteur du disque est le secteur d'amorce ou de démarrage (Boot block). Le répertoire racine (DIRectory) est contenu sur 14 secteurs de 512 octets. Il contient un enregistrement de 32 octets pour chaque entrée. Le répertoire racine peut donc contenir $14 \times 512 \div 32 = 224$ entrées. Le disque peut toutefois contenir beaucoup plus que 224 fichiers si on fait appel à des sous-répertoires (qui sont eux-mêmes des fichiers).

La longueur maximale théorique d'un fichier est de 4 gigaoctets puisque le champ "taille du fichier" de l'entrée de répertoire est de 4 octets ou 32 bits, mais elle est en pratique limitée par la capacité physique du disque.

L'unité d'allocation est un "cluster" ou grappe de 1 secteur.

Voici le contenu de chaque entrée du répertoire :



Entrée de répertoire MS-DOS

L'octet "status" contient normalement le premier caractère d'un nom de fichier. Lorsqu'une entrée de répertoire n'est pas utilisée, son octet "status" contient \$00. Lorsqu'un fichier est effacé, le DOS écrit la valeur \$E5 dans l'octet "status" de son entrée de répertoire.

L'octet attribut contient l'information sur les attributs de stockage du fichier :

- 1 = Lecture seulement
- 2 = Fichier caché
- 4 = Fichier système
- 8 = Nom du volume
- 16 = Sous-répertoire
- 32 = Archive

Le disque contient aussi une table d'accès de fichier (File Access Table) qui contient une entrée de 12 bits (1,5 octet) pour chaque u.a. allouable du disque. Cette table est contenue sur les secteurs FAT1 de la figure ci-dessus. Une copie de sécurité est gardée dans les secteurs FAT2 et permet de récupérer les fichiers si jamais la FAT1 est endommagée. L'adresse disque d'une donnée est son numéro d'u.a.. La disquette considérée contient 160 pistes (80 par côté) de 18 secteurs de 512 octets, donc un total de 2880 secteurs pour une capacité totale de 1,44 Mo ($2880 \times 0,5 \text{ Ko} = 1440 \text{ Ko} = 1,44 \text{ Mo}$). Le nombre d'u.a. allouables est en réalité de 2847, car les secteurs BOOT, FAT1, FAT2 et DIR ne sont pas allouables. Il faut donc, pour allouer ces u.a., 2847 entrées de 12 bits dans chaque FAT. Comme 12 bits = 1,5 octets, il faut $2847 \times 1,5 \text{ octets} = 4270,5 \text{ octets} = 8,32 \text{ secteurs de } 512 \text{ octets}$.

Sur les disques durs MS-DOS, les entrées de FAT passent à 16 bits avec FAT16 et à 32 bits avec FAT32. En effet, avec 12 bits on ne peut en tout et partout représenter que 4096 numéros. d'u.a., pour une capacité totale de $4096 \times 512 = 2 \text{ Mo}$. On peut augmenter cette

capacité en augmentant la taille des u.a., mais on ne peut ainsi guère dépasser 16 Mo avec des u.a. de 8 secteurs ou 4096 octets.

Note : Dans les champs numériques du répertoire, il faut prendre garde au fait qu'avec les processeurs Intel, un nombre en précision multiple est écrit avec l'octet le plus significatif à l'adresse haute. Ainsi, le nombre \$00001234 s'écrit \$34 12 00 00 en mémoire. Cette remarque s'applique également aux champs date et heure :

L'heure est codée sur 16 bits comme suit :

bits 0-4 : secondes en incréments de 2 sec, 0 à 29

bits 5-A : minutes, 0 à 59

bits B-F : heure, 0 à 23.

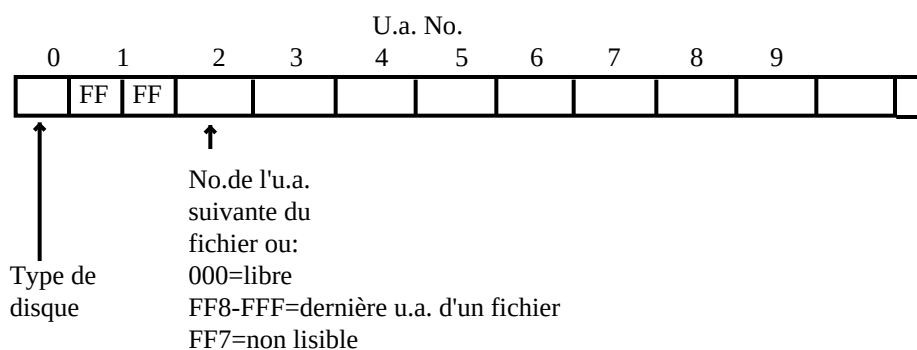
La date est codée sur 16 bits comme suit :

bits 0-4 : jour, 1 à 31

bits 5-8 : mois, 1 à 12

bits 9-F : année - 1980, 0 à 119.

Voici le début de la table d'allocation de fichier (FAT12) :



FAT

Codes du champs type de disquette

Code	3,5"			5,25"		
	Côtés	Pistes/côté	Secteurs/piste	Côtés	Pistes/côté	Secteurs/piste
F0	2	80	18	-	-	-
F8*	-	-	-	-	-	-
F9	2	80	9	2	80	15
FA	1	80	8	1	80	8
FB	2	80	9	2	80	8
FC	-	-	-	1	40	9
FE	-	-	-	1	40	8
FF	-	-	-	2	40	8

*Disque dur

L'u.a. 2 correspond à la première u.a. allouable du disque, c.-à-d. le secteur 16 de la piste 1. En effet, tel que mentionné plus haut, les premiers secteurs du disque ne sont pas des u.a. allouables : secteur d'amorce, FAT et répertoire. Ils sont donc inaccessibles à l'utilisateur.

Le système MS-DOS possède des inconvénients : dans le cas d'un disque dur, la FAT devient considérable. Pour une performance rapide, la FAT est normalement gardée en mémoire. Pour un disque dur de 64 Mo avec 2 secteurs de 512 octets par u.a., la FAT aurait une taille de 128 Ko (64 K entrées de 16 bits). Si, d'autre part, la FAT n'est pas gardée en mémoire, les opérations peuvent devenir très lentes. Par exemple, pour se déplacer au 32 000^e octet d'un fichier, on peut avoir besoin de 1 à 33 lectures dans la FAT du disque pour suivre la chaîne, selon la fragmentation du fichier. Nous voyons ci-dessous comment le système Unix évite cette difficulté.

Exemple 2 : organisation d'un disque Unix

Unix a été développé par Bell Laboratories à partir de 1970 par Ken Thompson et Dennis Ritchie. Ce système a été mis successivement sur PDP-7, 9 et 11, puis sur VAX, et enfin sur des machines à base de microprocesseur MC68000. Aujourd'hui, il fonctionne principalement sur des stations de travail avec des microprocesseurs RISC.

En 1979, on en est rendu à la version 7, la mieux connue. En 1982 apparaît le SYSTEM III, puis en 1983 le SYSTEM V suivant la politique de distribution commerciale de AT&T.

Depuis 1991, le phénomène Linux a fait son apparition. Il s'agit d'un UNIX de domaine public pour micro-ordinateur initialement écrit par un étudiant en informatique finlandais, Linus Torvalds. Il a été porté sur plusieurs plates-formes, notamment : Intel 80x86, PowerPC, Alpha, Amiga, etc. Sa conception est moderne et c'est elle que nous examinerons ici.

Fichiers sous Unix

Sous Unix, un fichier est une séquence linéaire de mots de 8 bits, d'une taille de 1 octet à plusieurs dizaines de Go dépendant du système de fichiers. L'utilisateur visualise le fichier comme une séquence linéaire et peut atteindre n'importe quel octet du fichier soit de façon absolue, soit en spécifiant une position relative par rapport à la position courante, ou encore par rapport à la fin du fichier. À partir d'une position quelconque, on peut lire un nombre quelconque d'octets.

Unité d'allocation : bloc de 512 octets (taille fixe).

1024 octets pour le SYSTEM V
 et le système 4.1 de Berkeley.
 4096 octets pour la version 4.2,
 1 Ko à 4 Ko pour Linux.

À chaque fichier (y compris les répertoires qui sont également des fichiers et les périphériques, que Unix considère comme des fichiers spéciaux) on associe un inode (index-node) dans lequel on peut trouver les informations suivantes :

- le propriétaire (user ID, group ID)
- les protections (code de 16 bits)
- les dates (création, dernière modification)
- les liens vers d'autres inodes
- le type de fichier (données, répertoire, périphérique)
- les adresses disques des blocs de données (13 à 15)
- la longueur du fichier en octets.

La taille d'un inode dans Linux est de 128 octets :

0	Type et permissions	Utilisateur (UID)	Taille du fichier
8	Heure et date d'accès		Heure et date de création
16	Heure et date de modification		Heure et date d'effacement
24	Groupe (GID)	Compteur de liens	Nb. de blocs
32	Attributs du fichier		Réservé
40	1 ^e bloc		2 ^e bloc
48	3 ^e bloc		4 ^e bloc
56	5 ^e bloc		6 ^e bloc
64	7 ^e bloc		8 ^e bloc
72	9 ^e bloc		10 ^e bloc
80	11 ^e bloc		12 ^e bloc
88	Bloc de 1 ^e indirection		Bloc de 2 ^e indirection
96	Bloc de 3 ^e indirection		Version du fichier
104	ACL du fichier		ACL du répertoire
112	Adresse de fragment		Réservé
120	Réservé		

ACL = Access Control List, pas encore implémenté

Inode du système de fichier Ext2

Dans le système de fichiers Ext2 adopté par la plupart des systèmes Linux, le disque est divisé en groupes de blocs. Chaque groupe de blocs contient un superbloc, des descripteurs

de groupe, un bitmap de blocs, un bitmap d'inodes, une table d'inodes et des blocs de données. Les bitmaps occupent chacun 1 bloc ou u.a. Ceci limite donc la taille des groupes à 8 192 blocs pour des blocs de 1 Ko. ou 32 768 blocs pour des blocs de 4 Ko. Les inodes sont répartis également parmi les groupes de blocs. Le nombre d'inodes par groupe ne peut non plus pas dépasser les nombres ci-dessus.

Bloc 0

Bloc d'amorce	Groupe de blocs 0	Groupe de blocs 1	...	Groupe de blocs n
---------------	-------------------	-------------------	-----	---------------------

Structure d'un disque Linux avec Ext2

Chaque groupe de blocs contient une copie du superbloc, des inodes et des blocs de données :

Super-bloc	Descripteurs de groupe	Bitmap de blocs	Bitmap d'inodes	Table d'inodes	Blocs de données
------------	------------------------	-----------------	-----------------	----------------	------------------

Structure d'un groupe de blocs avec Ext2

Le superbloc contient le nombre d'inodes et le nombre de blocs dans le système de fichiers. Il contient aussi de l'information sur le nombre d'inodes et de blocs par groupe de blocs.

Le descripteur de groupe est une structure de 32 octets donnant le nombre de blocs dans le bitmap d'inodes, dans le bitmap de blocs et dans la table d'inodes, le nombre d'inodes libres, le nombre de blocs libres et le nombre de répertoires. Cette dernière information est utile au système qui tente répartir les répertoires parmi les différents groupes de blocs. Il allouera donc un nouveau répertoire dans le groupe qui en a le moins.

Cette organisation permet aux inodes d'être voisines des blocs auxquels elles font référence, et aux inodes d'être voisins de leur inode de répertoire, ce qui permet un accès plus rapide.

Dans chaque inode, on trouve 15 adresses disque en termes de n° de blocs. Les 12 premières adresses d'un inode permettent d'atteindre un espace données de :

$$12 \times 1024 \text{ octets} = 12\,288 \text{ octets.}$$

La 11^e adresse disque pointe vers un autre bloc de 1024 octets qui contient 256 adresses disque (4 octets par adresse), soit :

$$256 \times 1024 \text{ octets} = 262\,144 \text{ octets.}$$

La 12^e adresse disque pointe vers 256 blocs indirects (indirection d'ordre 2) qui pointent à leur tour chacun vers 256 adresses disques :

$$256 \times 256 \times 1024 \text{ octets} = 67\,108\,864 \text{ octets.}$$

La 13^e adresse disque est une indirection d'ordre 3, ce qui permet aux fichiers Linux d'atteindre des tailles avoisinant

$$256 \times 256 \times 256 \times 1024 \text{ octets} = 17\,179\,869\,184 \text{ octets.}$$

La taille maximale d'un tel fichier sera donc :

$$17\,179\,869\,184 + 67\,108\,864 + 262\,144 + 12\,288 = 16\text{Go.}$$

Cette implantation privilégie, du point de vue accès, les fichiers de petite taille (12 Ko). À l'ouverture, le premier descripteur du fichier (inode) est copié en mémoire. Lorsqu'on franchit le cap de 12 288 octets, le système d'exploitation copie le premier bloc indirect, et ainsi de suite.

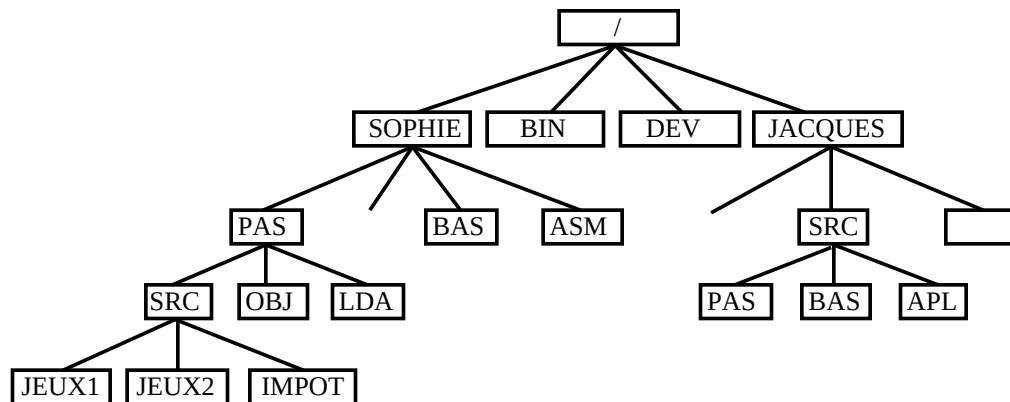
Pour savoir où se trouve l'octet n d'un fichier, si $n < 12\,288$, il se trouve dans le bloc direct $n / 1024$ à l'offset $n \bmod 1024$. Si $n > 12\,288$ et $n \leq 262\,144$, il se trouve dans le bloc donné par la table de première indirection $(n - 12\,288) / 1024$ à l'offset $(n - 12\,288) \bmod 1024$, etc.

Ext2 tente de minimiser la fragmentation des fichiers lors de l'allocation des blocs. Il cherche des blocs libres autour d'un bloc cible. Si ce dernier est libre, il est alloué. Sinon, on cherche un bloc libre dans les 32 entourant le bloc cible. Si on en trouve un, il est alloué. Sinon, on cherche un bloc libre qui soit au moins dans le même groupe de blocs que le bloc cible. Il y a plusieurs heuristiques pour la définition du bloc cible. L'un d'entre eux est de prendre le premier bloc libre dans le groupe de blocs où se trouve l'inode du fichier.

Lorsqu'un bloc libre est trouvé, on réserve les 8 blocs suivants, s'ils sont libres. Quand le fichier sera fermé, les blocs réservés restants seront libérés.

Structure du répertoire sous Unix

Le répertoire sous Unix est organisé de façon hiérarchique pour faciliter les opérations de maintenance :



Répertoires Unix

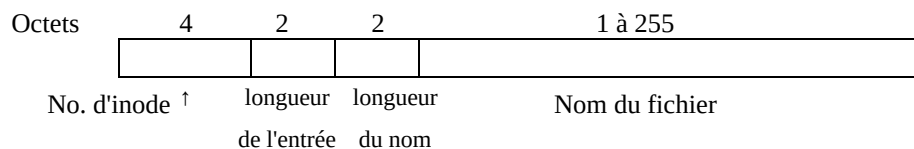
On spécifie par exemple le fichier JEUX1 en écrivant :

/SOPHIE/PAS/SRC/JEUX1

Pour accéder au fichier JEUX1, le système d'exploitation cherche SOPHIE dans le répertoire RACINE pour localiser l'inode correspondant à SOPHIE. Il lit ensuite le fichier relatif à SOPHIE qui est dans ce cas un fichier répertoire. Ce fichier répertoire aiguille la recherche vers un autre répertoire, et ainsi de suite, jusqu'à ce qu'on atteigne les données finales. À chaque phase, il y a branchement vers la table des inodes jusqu'à ce qu'on aboutisse au dernier inode.

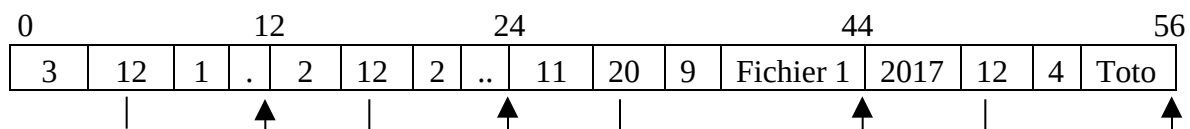
Contenu d'un élément de répertoire

Les entrées d'un répertoire Linux sont de longueur variable parce que les noms de fichier peuvent aller de 1 caractère à 255 caractères. Il s'agit en fait d'une liste chaînée, puisque le champ longueur de l'entrée (rec_len), toujours arrondi vers le haut à un multiple de 4, donne en fait la position de l'entrée suivante.



Entrée de répertoire Linux

Voici le contenu d'un répertoire dont l'inode (puisque un répertoire est également un fichier) porterait le #3 et qui serait situé directement sous le répertoire racine (Root Directory, inode=2) :



inode	rec_len	name_len	Entrée
3	12	1	.
2	12	2	..
11	20	9	Fichier 1
2017	12	4	Toto
...			...
...			...

← Pointeur vers lui-même

← Pointeur vers son parent

Répertoire Linux

Chaque répertoire ne peut avoir qu'un parent. Le répertoire racine n'a pas de parent et son pointeur "parent" contient lui-même, c.-à-d. le #2.

Lorsqu'un fichier est effacé, le numéro d'inode de l'entrée de répertoire est mis à 0 et l'entrée est éliminée de la liste chaînée en augmentant le champ rec_len de l'entrée précédente pour qu'elle pointe à l'entrée suivante.

Exercices

- 12.1** Une mémoire virtuelle a une capacité de 2^{46} octets organisés en pages de 512 octets. La mémoire physique qui supporte cette mémoire virtuelle a une capacité de 1 mégaoctets.
- Quelle est la taille de la mémoire associative qui sert de dictionnaire traducteur des pages en octets?
 - Voici quelques entrées dans ce dictionnaire après un certain temps :

```

...
00001018A045
02004050F267
0FFF034CC012
199939AA0448
FD7E400048D7
1836A091238F
...

```

Si l'adresse physique d'un opérande est 89045 (en hexadécimal), quelle est l'adresse virtuelle de ce même opérande?

- 12.2** Considérons une mémoire virtuelle de 512 Mo, une mémoire physique de 32 Mo et des pages de 8 Ko :
- Quelle serait la taille de la table des pages ?
 - Quel est le numéro de page virtuelle correspondant à l'adresse virtuelle 0x00300B2A?
- 12.3** Un disque dur est utilisé avec MS-DOS 3.3. La table d'allocation de fichiers (FAT) contient une entrée de 16 bits pour chaque u.a. et est de la forme suivante :

0	1	2	3	
FFF0	FFFF	0003	0004	etc.

Chaque u.a. contient 1 secteur de 512 octets.

- Combien de clusters la FAT occupera-t-elle si on désire avoir 20 Mo allouables ?
- Quel serait la capacité maximale allouable avec une FAT de ce type ?

12.4 Le répertoire d'un disque souple MS-DOS 1.0 contient l'entrée suivante :

54	50	33	20	20	20	20	20	54	58	54	01	00	00	00	00
00	00	00	00	00	00	C7	B4	78	13	05	00	AB	01	00	00

Donnez toutes les informations que vous pouvez en tirer à propos du fichier correspondant.

12.5 Un disque souple double face, double densité comporte 9 secteurs par piste et chacun contient 512 octets. Le nombre de pistes par face s'élève à 42. La convention de numérotation physique est la suivante (les majuscules désignent les valeurs maximum des variables) :

$s : 1 \text{ à } 9$ ($S : 9$)

$p : 0 \text{ à } 41$ ($P : 41$)

$f : 0 \text{ à } 1$ ($F : 1$)

$B : 512$ octets/secteur.

a. Déterminez la capacité totale C du disque en fonction des valeurs maximum.

Application numérique.

b. n désigne le secteur logique et varie entre 0 et N . Donnez la valeur de N .

c. Exprimez les variables p , f et s en fonction de n . On peut employer les fonctions DIV ou MOD qui fournissent le quotient et le reste de la division entière. Application numérique pour $n = 319$.

On réserve :

- les secteurs $s : 0$ et 1 pour placer les identifications du disque et le programme de démarrage du système d'exploitation disque.
- les secteurs $s : 2, 3$ et 4 pour placer la Table des allocations du disque.
- les secteurs $s : 5, 6$ et 7 pour conserver une copie de la Table des allocations.
- les secteurs $s : 8$ à 14 inclusivement pour ranger les entrées de répertoire. Chaque entrée de répertoire occupe 32 octets.

Les secteurs à partir de 15 jusqu'à la fin du disque servent à stocker les données de l'utilisateur.

d. Déterminez le nombre maximum de fichiers qu'on peut créer sous le répertoire principal.

e. Sachant qu'on se sert de 2 octets dans la Table des allocations pour repérer un secteur de données sur disque (les secteurs de gestion de $n : 0$ à $n : 14$ ne sont pas répertoriés dans la Table des allocations et le mécanisme exact de repérage n'est pas précisé) la taille de la Table des allocations est-elle suffisante pour gérer le disque en entier?

- f. L'unité d'allocation est égale à un secteur. Quelle est la perte moyenne d'espace sur le disque?

12.6 On désire confectionner un système d'exploitation possédant une organisation disque similaire à celle de Unix. Les données du problème sont :

Capacité formatée du disque : 320 Mo

Taille d'un secteur : 1024 octets

Taille d'une unité d'allocation : 2048 octets

Taille d'une entrée de répertoire : 32 octets

Taille d'un nœud-i : 128 octets

Taille d'une adresse disque : 4 octets

Nombre d'adresses disque à accès direct sans indirection : 20

Nombre d'indirections : 2 (au lieu de 3 dans Unix)

On rappelle que l'unité d'allocation dans la zone des données est de 2 048 octets et non le secteur physique, et que les adresses disque se réfèrent aux UA et non aux secteurs. Chaque fichier possède une entrée de répertoire, un nœud-i et un ensemble d'UA de données propres et de données répertoire.

- a. Calculez la taille maximale d'un fichier sans indirection.
- b. Calculez la taille maximale d'un fichier avec l'indirection 1.
- c. Calculez la taille maximale d'un fichier avec l'indirection 2.
- d. Calculez la position du 8 millionième octet. Le résultat doit être donné sous la forme :

Indirection : 0, 1 ou 2

Numéro de la ramification au niveau 1 :

Numéro de la ramification au niveau 2 :

Offset de l'octet cherché dans la dernière UA :

2. Appendices

1. Le code ASCII

DEC			HEX			DEC			HEX			DEC			HEX		
0	NULL	00	32	SP	20	64	@	40	96	`	60						
1	SOH	01	33	!	21	65	A	41	97	a	61						
2	STX	02	34	“	22	66	B	42	98	b	62						
3	ETX	03	35	#	23	67	C	43	99	c	63						
4	EOT	04	36	\$	24	68	D	44	100	d	64						
5	ENQ	05	37	%	25	69	E	45	101	e	65						
6	ACK	06	38	&	26	70	F	46	102	f	66						
7	BEL	07	39	‘	27	71	G	47	103	g	67						
8	BS	08	40	(	28	72	H	48	104	h	68						
9	HT	09	41	)	29	73	I	49	105	i	69						
10	LF	0A	42	*	2A	74	J	4A	106	j	6A						
11	VT	0B	43	+	2B	75	K	4B	107	k	6B						
12	FF	0C	44	,	2C	76	L	4C	108	l	6C						
13	CR	0D	45	-	2D	77	M	4D	109	m	6D						
14	SO	0E	46	.	2E	78	N	4E	110	n	6E						
15	SI	0F	47	/	2F	79	O	4F	111	o	6F						
16	DLE	10	48	0	30	80	P	50	112	p	70						
17	DC1	11	49	1	31	81	Q	51	113	q	71						
18	DC2	12	50	2	32	82	R	52	114	r	72						
19	DC3	13	51	3	33	83	S	53	115	s	73						
20	DC4	14	52	4	34	84	T	54	116	t	74						
21	NAK	15	53	5	35	85	U	55	117	u	75						
22	SYN	16	54	6	36	86	V	56	118	v	76						
23	ETB	17	55	7	37	87	W	57	119	w	77						
24	CAN	18	56	8	38	88	X	58	120	x	78						
25	EM	19	57	9	39	89	Y	59	121	y	79						
26	SUB	1A	58	:	3A	90	Z	5A	122	z	7A						
27	ESC	1B	59	;	3B	91	[	5B	123	{	7B						
28	FS	1C	60	<	3C	92	\	5C	124		7C						
29	GS	1D	61	=	3D	93	]	5D	125	}	7D						
30	RS	1E	62	>	3E	94	^	5E	126	~	7E						
31	US	1F	63	?	3F	95	_	5F	127	DEL	7F						

2. Interprétation des codes de contrôle ASCII

No.	Symb.	Nom	Clavier
00	NUL	Null	CTRL @
01	SOH	Start of heading	CTRL A
02	STX	Start of text	CTRL B
03	ETX	End of text	CTRL C
04	EOT	End of transmission	CTRL D
05	ENQ	Enquiry	CTRL E
06	ACK	Acknowledge	CTRL F
07	BEL	Bell	CTRL G
08	BS	Backspace	CTRL H
09	HT	Horizontal tab	TAB ou CTRL I
0A	LF	Line feed	LINEFEED ou CTRL L
0B	VT	Vertical tab	VERTICAL TAB ou CTRL K
0C	FF	Form feed	FORMFEED ou CTRL L
0D	CR	Carriage return	RETURN ou CTRL M
0E	SO	Shift out	CTRL N
0F	SI	Shift in	CTRL O
10	DLE	Data link escape	CTRL P
11	DC1	Device control 1	CTRL Q
12	DC2	Device control 2	CTRL R
13	DC3	Device control 3	CTRL S
14	DC4	Device control 4	CTRL T
15	NAK	Negative Acknowledge	CTRL U
16	SYN	Synchronous idle	CTRL V
17	ETB	End of transmission block	CTRL W
18	CAN	Cancel	CTRL X
19	EM	End of medium	CTRL Y
1A	SUB	Substitute	CTRL Z
1B	ESC	Escape	ESCAPE ou CTRL [
1C	FS	File separator	CTRL \
1D	GS	Group separator	CTRL]
1E	RS	Record separator	HOME ou CTRL ^
1F	US	Unit separator	CTRL _

3. Le code EBCDIC

00	NULL	20	DS	40	SP	60	--	80		A0		C0	+ZERO	E0	
01	SOH	21	SOS	41		61	/	81	a	A1		C1	A	E1	
02	STX	22	FS	42		62		82	b	A2	s	C2	B	E2	S
03	ETX	23		43		63		83	c	A3	t	C3	C	E3	T
04	PF	24	BYP	44		64		84	d	A4	u	C4	D	E4	U
05	HT	25	LF	45		64		85	e	A5	v	C5	E	E5	V
06	LC	26	EOB	46		66		86	f	A6	w	C6	F	E6	W
07	DEL	27	PRE	47		67		87	g	A7	x	C7	G	E7	X
08		28		48		68		88	h	A8	y	C8	H	E8	Y
09		29		49		69		89	i	A9	z	C9	I	E9	Z
0A	SMM	2A	SM	4A	¢	6A		8A		AA		CA		EA	
0B	VT	2B	CU2	4B		6B	,	8B	{	AB		CB		EB	
0C	FF	2C		4C	<	6C	%	8C		AC		CC		EC	
0D	CR	2D	ENQ	4D	(	6D	_	8D		AD	[	CD		ED	
0E	SO	2E	ACK	4E	+	6E	>	8E		AE		CE		EE	
0F	SI	2F	BEL	4F	]	6F	?	8F		AF		CF		EF	
10	DLE	30		50	&	70		90		B0		D0	-ZERO	F0	0
11	DC1	31		51		71		91	j	B1		D1	J	F1	1
12	DC2	32	SYN	52		72		92	k	B2		D2	K	F2	2
13	DC3	33		53		73		93	l	B3		D3	L	F3	3
14	RES	34	PN	54		74		94	m	B4		D4	M	F4	4
15	NL	35	RS	55		75		95	n	B5		D5	N	F5	5
16	BS	36	UC	56		76		96	o	B6		D6	O	F6	6
17	IDL	37	EOT	57		77		97	p	B7		D7	P	F7	7
18	CAN	38		58		78		98	q	B8		D8	Q	F8	8
19	EM	39		59		79		99	r	B9		D9	R	F9	9
1A	CC	3A		5A	!	7A	:	9A		BA		DA		FA	
1B	CU1	3B	CU3	5B	\$	7B	#	9B	}	BB		DB		FB	
1C	FLS	3C	DCA	5C	*	7C	@	9C		BC		DC		FC	
1D	GS	3D	NAK	5D	)	7D	'	9D		BD		DD		FD	
1E	RDS	3E		5E	;	7E	=	9E		BE		DE		FE	
1F	US	3F	SUB	5F	¬	7F	*	9F		BF		DF		FF	

4. Conversions à l'aide de tables

Il est possible d'effectuer très rapidement les conversions binaire-décimal et décimal-binaire à l'aide des tables suivantes :

4.1 Nombres entiers

1	1	1 00	256	1 00 00	65 536	1 00 00 00	16 777 216
2	2	2 00	512	2 00 00	131 072	2 00 00 00	33 554 432
3	3	3 00	768	3 00 00	196 608	3 00 00 00	50 331 648
4	4	4 00	1 024	4 00 00	262 144	4 00 00 00	67 108 864
5	5	5 00	1 280	5 00 00	327 680	5 00 00 00	83 886 080
6	6	6 00	1 536	6 00 00	393 216	6 00 00 00	100 663 296
7	7	7 00	1 792	7 00 00	458 752	7 00 00 00	117 440 512
8	8	8 00	2 048	8 00 00	524 288	8 00 00 00	134 217 728
9	9	9 00	2 304	9 00 00	589 824	9 00 00 00	150 994 944
A	10	A 00	2 560	A 00 00	655 360	A 00 00 00	167 772 160
B	11	B 00	2 816	B 00 00	720 896	B 00 00 00	184 549 376
C	12	C 00	3 072	C 00 00	786 432	C 00 00 00	201 326 592
D	13	D 00	3 328	D 00 00	851 968	D 00 00 00	218 103 808
E	14	E 00	3 584	E 00 00	917 504	E 00 00 00	234 881 024
F	15	F 00	3 840	F 00 00	983 040	F 00 00 00	251 658 240
10	16	10 00	4 096	10 00 00	1 048 576	10 00 00 00	268 435 456
20	32	20 00	8 192	20 00 00	2 097 152	20 00 00 00	536 870 912
30	48	30 00	12 288	30 00 00	3 145 728	30 00 00 00	805 306 368
40	64	40 00	16 384	40 00 00	4 194 304	40 00 00 00	1 073 741 824
50	80	50 00	20 480	50 00 00	5 242 880	50 00 00 00	1 342 177 280
60	96	60 00	24 576	60 00 00	6 291 456	60 00 00 00	1 610 612 736
70	112	70 00	28 672	70 00 00	7 340 032	70 00 00 00	1 879 048 192
80	128	80 00	32 768	80 00 00	8 388 608	80 00 00 00	2 147 483 648
90	144	90 00	36 864	90 00 00	9 437 184	90 00 00 00	2 415 919 104
A0	160	A0 00	40 960	A0 00 00	10 485 760	A0 00 00 00	2 684 354 560
B0	176	B0 00	45 056	B0 00 00	11 534 336	B0 00 00 00	2 952 790 016
C0	192	C0 00	49 152	C0 00 00	12 582 912	C0 00 00 00	3 221 215 472
D0	208	D0 00	53 248	D0 00 00	13 631 488	D0 00 00 00	3 489 660 928
E0	224	E0 00	57 344	E0 00 00	14 680 064	E0 00 00 00	3 758 096 384
F0	240	F0 00	61 440	F0 00 00	15 728 640	F0 00 00 00	4 026 531 840

4.1.1 Conversion binaire-décimal

Pour effectuer la conversion en décimal (BCD) d'un nombre binaire (hexadécimal), il suffit de chercher la représentation hexadécimale de chaque digit dans la colonne de gauche de la table et d'additionner les valeurs décimales correspondantes de la colonne de droite.

Exemple :

$$\begin{array}{rcl}
 \text{B87A} & = & 0\text{x B000} & = & 45\,056 \\
 & + & 0\text{x 800} & = & 2\,048 \\
 & + & 0\text{x 70} & = & 112 \\
 & + & 0\text{x A} & = & 10 \\
 \hline
 & & 0\text{x B87A} & = & 47\,226
 \end{array}$$

Cette méthode est très rapide, puisqu'elle ne nécessite que 3 additions BCD pour convertir 16 bits et 7 pour convertir 32 bits.

4.1.2 Conversion décimal-binaire

Pour la conversion décimal-binaire, on cherche dans la colonne de droite de la table le plus grand nombre qui soit inférieur au nombre à convertir. On soustrait ce nombre du nombre à convertir et on ajoute le nombre hexadécimal correspondant de la colonne de gauche. On recommence avec le reste, jusqu'à obtention d'un reste nul. Exemple :

$$\begin{array}{rcl}
 1\,000 : & 1\,000 - 768 = 232 & : + 0\text{x 300} \\
 & 232 - 224 = 8 & : + 0\text{x E0} \\
 & 8 - 8 = 0 & + 0\text{x 8} \\
 & & \hline
 & & 0\text{x 3E8}
 \end{array}$$

4.2 Conversion de nombres fractionnaires

.F	.9375	.00F	.0036 6210 9375	.0000F	.0000 1430 5114 7
.E	.8750	.00E	.0034 1796 8750	.0000E	.0000 1335 1440 4
.D	.8125	.00D	.0031 7382 8125	.0000D	.0000 1239 7766 1
.C	.7500	.00C	.0029 2968 7500	.0000C	.0000 1144 4091 8
.B	.6875	.00B	.0026 8554 6875	.0000B	.0000 1049 9417 5
.A	.6250	.00A	.0024 4149 6250	.0000A	.0000 0953 6743 16
.9	.5625	.009	.0021 9736 5625	.00009	.0000 0858 3068 85
.8	.5000	.008	.0019 5312 5000	.00008	.0000 0762 9394 53
.7	.4375	.007	.0017 0898 4375	.00007	.0000 0667 5720 21
.6	.3750	.006	.0014 6484 3750	.00006	.0000 0572 2045 90
.5	.3125	.005	.0012 2070 3125	.00005	.0000 0476 8371 58
.4	.2500	.004	.0009 7656 2500	.00004	.0000 0381 4697 27
.3	.1875	.003	.0007 3242 1875	.00003	.0000 0286 1022 95
.2	.1250	.002	.0004 8828 1250	.00002	.0000 0190 7348 63
.1	.0625	.001	.0002 4414 0625	.00001	.0000 0095 3674 316
.0F	.0585	.000F	.0002 2888 1836	.00000F	.0000 0089 4069 672
.0E	.0546	.000E	.0002 1362 3047	.00000E	.0000 0083 4465 027
.0D	.0625	.000D	.0001 9836 4258	.00000D	.0000 0077 4860 382
.0C	.0468 7500	.000C	.0001 8310 5469	.00000C	.0000 0071 5255 737
.0B	.0429 6875	.000B	.0001 6784 6680	.00000B	.0000 0065 5651 093
.0A	.0390 6250	.000A	.0001 5258 7891	.00000A	.0000 0059 6046 448
.09	.0351 5625	.0009	.0001 3732 9102	.000009	.0000 0053 6441 803
.08	.0312 5000	.0008	.0001 2207 0312 5	.000008	.0000 0047 6897 158
.07	.0273 4375	.0007	.0001 0681 1523 4	.000007	.0000 0041 7232 513
.06	.0234 3750	.0006	.0000 9155 2734 4	.000006	.0000 0035 7627 869
.05	.0195 3125	.0005	.0000 7629 3945 3	.000005	.0000 0029 8023 224
.04	.0156 2500	.0004	.0000 6103 5156 2	.000004	.0000 0023 8418 579
.03	.0117 1875	.0003	.0000 4577 6367 2	.000003	.0000 0017 8813 934
.02	.0078 1250	.0002	.0000 3051 7578 1	.000002	.0000 0011 9209 290
.01	.0039 0625	.0001	.0000 1525 8789 1	.000001	.0000 0005 9604 6447

On utilise cette table comme celle des nombres entiers :

Conversion binaire-décimal :

$$\begin{array}{rcl}
 \text{exemple : } 0,68_{16} & = & 0,6 \\
 & & = 0,3750 \\
 & + 0,08 & = 0,031250 \\
 & & \hline
 \end{array}$$

$$0,406250_{10}$$

Conversion décimal-binaire :

exemple : 0,40625 :

0,40625 - 0,3750	= 0,03125	0,6
0,03125 - 0,03125	= 0	+ 0,08
		0,68 ₁₆

5. Table de multiplication hexadécimale

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	×
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	1
0	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E	2
0	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D	3
0	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C	4
0	5	A	F	14	18	1E	23	28	2D	32	37	3C	41	46	4B	5
0	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A	6
0	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69	7
0	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78	8
0	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87	9
0	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96	A
0	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5	B
0	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4	C
0	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3	D
0	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2	E
0	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1	F

3. Solution des exercices

Chapitre 3

3.1 COPY A : B :

43 4F 50 59 20 41 3A 20 42 3A 0D

3.2 Convertir 1100101100111010 en octal et en hexadécimal

octal : 145472_8

hexadécimal : $CB3A_{16}$

3.3 Non

$$\begin{aligned}
 3.4 \quad N &= 1 \times 2^{15} + 1 \times 2^{14} + 0 \times 2^{13} + 0 \times 2^{12} + 1 \times 2^{11} + 0 \times 2^{10} + 1 \times 2^9 + 1 \times 2^8 + 0 \times 2^7 + 0 \times 2^6 \\
 &\quad + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\
 &= 32\,768 + 16\,384 + 0 + 0 + 2\,048 + 0 + 512 + 256 + 0 + 0 + 32 + 16 + 8 + 0 + 2 + 0 \\
 &= 52\,026
 \end{aligned}$$

$$3.5 \quad 123_4 = 1 \times 4^2 + 2 \times 4^1 + 3 \times 4^0 = 27$$

$$57_{10} : \quad 57/4 = 14 \text{ reste } 1$$

$$14/4 = 3 \text{ reste } 2$$

$$3/4 = 0 \text{ reste } 3$$

$$57_{10} = 321_4$$

$$3.6 \quad n \times 21 = n \times 16 + n \times 4 + n$$

$$B7F_{16} \times 21_{10} = 1011\,0111\,1111_2 \times 21_{10} = 1011\,0111\,1111\,0000$$

$$+ 10\,1101\,1111\,1100 + 1011\,0111\,1111 = 1111\,0001\,0110\,1011_2$$

$$= F16B_{16}.$$

$$3.7 \quad A_i - B_i - C_{i-1} = D_i \text{ avec emprunt } C_i :$$

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

- 3.8** a. 0 à 255
b. -128 à + 127

3.9

$$\begin{array}{r} 10101 \\ \times 110 \\ \hline 00000 \\ 10101 \\ 10101 \\ \hline 1111110 \end{array}$$

3.10 $100101 \div 1101$

$$\begin{array}{r} 100101 \\ \underline{1101} \quad 1 \\ 101 \\ \underline{1011} \quad 0 \end{array}$$

quotient = 10_2 , reste = 1011_2

- 3.11** a. $0,101101_2 = 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4} + 0 \times 2^{-5} + 1 \times 2^{-6}$
 $= 0.5 + 0,125 + 0,0625 + 0,015625 = 0,703125$
ou encore $= 101101_2 \times 2^{-6} = 45 \times 0,015625 = 0,703125$

b. $0,484375_{10} =$

$$\begin{array}{r} 0, \overline{484375 \times 2} \\ 0, \overline{96875 \times 2} \\ 1, \overline{9375 \times 2} \\ 1, \overline{875 \times 2} \\ 1, \overline{75 \times 2} \\ 1, \overline{5 \times 2} \\ 1, \overline{0 \times 2} \\ 0,0 \end{array}$$

$$= 0,0111110000000..._2$$

3.12 a. $5B7,A34F_{16} = 010110110111.1010001101001111_2$

$$= 1,01101101111010001101001111 \times 2^{10}$$

signe = 0, exposant + 127 = 137 = 10001001_2 ,

mantisse normalisée = $1.01101101111010001101001111$

d'où : $0 \ 10001001 \ 01101101111010001101001111$

$$= 44B6F469_{IEEE}$$

b. $7A5F43B6_{IEEE} = 0 \ 11110100 \ 10111110100001110110110$

signe = positif, exposant = $244 - 127 = 117$

mantisse = $1,10111110100001110110110_2 = 1,BE876C_{16}$;

3.15 $1,0 = + 1,0 \times 2^0$. Donc $s = 0$, mantisse = 1,0 et exposant $e = 1023$.

$1023 = 0111111111_2$. Le nombre est donc :

0011 1111 1111 0000 0000 0000 0000 0000 0000 0000 ...

ou $3FF0000000000000_{IEEE}$

3.16 a. $432_{10} = 110110000_2 = 0,000110110000 \times 16^3$; exposant corrigé = $64 + 3 = 67$;
→ 0 1000011 0001 1011 0000 0000 0000 0000.

b. L'erreur relative sur un nombre est la valeur du plus petit bit par rapport à celle du nombre exprimé. Dans le cas présent, sur le nombre 15, l'erreur relative est :

$$1 \div 111100000000000000000000_2 = 1 \div 15\,728\,640,$$

tandis que pour le nombre 1,0, elle est de

$$1 \div 000100000000000000000000_2 = 1 \div 1\,048\,576.$$

Dans la notation IEEE, comme la mantisse est normalisée sous la forme $1,f$, l'erreur relative varie beaucoup moins, tout au plus par un facteur 2.

c. Le plus petit nombre normalisé représentable dans ce système est : exposant = 0, ce qui représente 16^{-64} ; la mantisse normalisée minimale est :
 $0,0001\,0000\,0000\,0000\,0000\,0000_2$.

$$\text{Ce nombre est donc } 0,0001_2 \times 16^{-64} = 5,3976053 \times 10^{-79}.$$

Le plus grand nombre normalisé représentable dans ce système est :

exposant = 127, ce qui représente 16^{63} ; la mantisse normalisée maximale est :

$$0,1111\,1111\,1111\,1111\,1111\,1111_2.$$

$$\text{Ce nombre est donc } 0,99999999 \times 16^{63} = 7,2370051 \times 10^{75}.$$

Chapitre 5

5.1 ET et OU à trois entrées :

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

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

5.2 a. $(a + b).(a + \bar{b}) = aa + a\bar{b} + ab + b\bar{b} = a + a(b + \bar{b}) = a$

b. $a + (\bar{a}b) = a + ab + \bar{a}b = a + b(a + \bar{a}) = a + b$

5.3 a. Est-ce que $\overline{A.B.C} = \overline{A.B.C}$? ceci revient à dire $\overline{A.B.C} = A.B.C$

A B C	$\overline{A.B}$	$\overline{A.B.C}$	$\overline{BC}$	$A.\overline{BC}$
0 0 0	1	0	1	0
0 0 1	1	1	1	0
0 1 0	1	0	1	0
0 1 1	1	1	0	0
1 0 0	1	0	1	1
1 0 1	1	1	1	1
1 1 0	0	0	1	1
1 1 1	0	0	0	0

Comme la 3^e et la 5^e colonne ne sont pas identiques, il n'y a pas égalité.

$$\begin{aligned}
 \text{b. } (A \quad B \quad C) &= (A \quad \bar{B} + \bar{A}B) \quad C = (A \quad \bar{B} + \bar{A}B).\bar{C} + (\overline{A\bar{B} + \bar{A}B}).C \\
 &= \overline{A\bar{B} + \bar{A}B}.\bar{C} + (\overline{A\bar{B}}.\overline{\bar{A}B}).C = A \quad \bar{B} \quad \bar{C} + \bar{A}B\bar{C} + (\bar{A} + B)(A + \bar{B}).C \\
 &= \overline{A\bar{B} + \bar{A}B}.\bar{C} + \bar{A}B\bar{C} + \bar{A}B\bar{C} + ABC = A(\bar{B}\bar{C} + BC) + \bar{A}(B\bar{C} + BC) \\
 &= A(\bar{B} \quad C) + \bar{A}(B \quad C) = A \quad (B \quad C)
 \end{aligned}$$

5.4

G	A	B	Y ₁	Y ₂
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0

Le plus simple ici est d'utiliser les maxterms puisqu'il n'y a que deux 0 :

$$\begin{aligned}
 Y_1 &= (G+A+B).(G+\bar{A}+B) \\
 &= G+\bar{A}G+BG+AG+A\bar{A}+AB+BG+\bar{A}B+B \\
 &= G+G(\bar{A}+A)+BG+BG+B(\bar{A}+A)+B \\
 &= G+G+BG+B+B=G+BG+B \\
 &= G+B
 \end{aligned}$$

De la même façon :

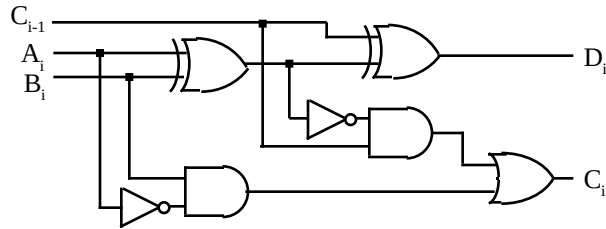
$$\begin{aligned}
 Y_2 &= (G+A+\bar{B})(G+\bar{A}+\bar{B}) \\
 Y_2 &= G+\bar{B}.
 \end{aligned}$$

5.5 a.

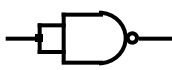
A _i	B _i	C _{i-1}	D _i	C _i
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$$\begin{aligned}
 \text{b. } D_i &= \bar{A}_i\bar{B}_iC_{i-1} + \bar{A}_iB_i\bar{C}_{i-1} + A_i\bar{B}_i\bar{C}_{i-1} + A_iB_iC_{i-1} \\
 D_i &= (\bar{A}_i\bar{B}_i + A_iB_i)C_{i-1} + (\bar{A}_iB_i + A_i\bar{B}_i)\bar{C}_{i-1} \\
 D_i &= (\bar{A}_i \oplus B_i)C_{i-1} + (A_i \oplus B_i)\bar{C}_{i-1} = (A_i \oplus B_i) \oplus C_{i-1} \\
 C_i &= \bar{A}_i\bar{B}_iC_{i-1} + \bar{A}_iB_i\bar{C}_{i-1} + \bar{A}_iB_iC_{i-1} + A_iB_iC_{i-1} \\
 C_i &= (\bar{A}_i\bar{B}_i + A_iB_i)C_{i-1} + \bar{A}_iB_i(\bar{C}_{i-1} + C_{i-1}) = (\bar{A}_i \oplus B_i)C_{i-1} + \bar{A}_iB_i
 \end{aligned}$$

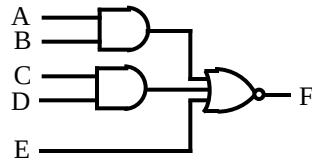
c.



5.6

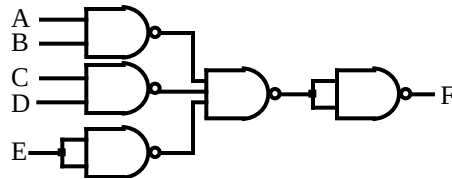


A	X
0	1
1	0

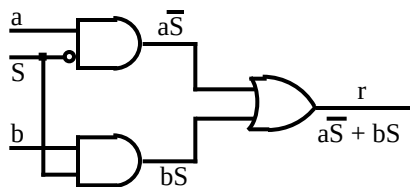


a. $F = \overline{AB + CD + E}$

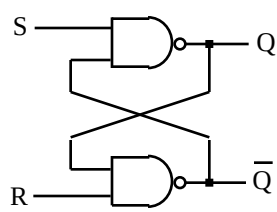
b. Par le théorème de deMorgan, on trouve $F = \overline{AB} \cdot \overline{CD} \cdot \overline{E}$



5.7 Multiplexeur à deux entrées



5.11



S	R	Q	Q ⁺	
0	0	0	?	indéterminé
0	0	1	?	indéterminé
0	1	0	1	mise à 1
0	1	1	1	mise à 1
1	0	0	0	effacement
1	0	1	0	effacement
1	1	0	0	Q ⁺ = Q
1	1	1	1	Q ⁺ = Q

		Q	
		Q ⁺	$\overline{Q}^+$
SR	00	?	?
	01	1	0
	11	Q	$\overline{Q}$
	10	0	1

$$Q^+ = \overline{S} + RQ$$

5.12 On a la table de transitions suivante :

Q ₁	Q ₀	Q ₁ ⁺	Q ₀ ⁺	D ₁	D ₀
0	0	0	1	0	1
0	1	1	0	1	0
1	0	0	0	0	0
1	1	x	x	x	x

D'où les tables de Karnaugh suivantes pour D₁ et D₀ :

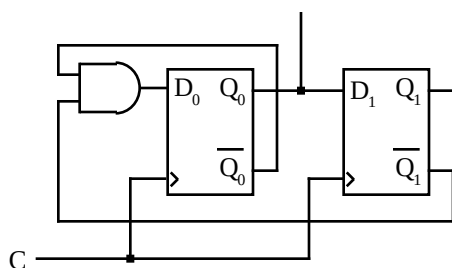
		Q ₀	
		0	1
Q ₁	0	1	0
	1	0	x

$D_0 = \overline{Q_1} \overline{Q_0}$

		Q ₀	
		0	1
Q ₁	0	0	1
	1	0	x

$D_1 = Q_0$

et le circuit :



Chapitre 7

- 7.1 a. $2^{32} = 4\,294\,967\,296$ combinaisons d'adresses possibles. La capacité mémoire dépendra de l'organisation de la machine. Si la machine a 1 octet par adresse, comme c'est le cas pour la plupart des microprocesseurs, on aura 4 giga-octets. Sinon, on aura 4 294 967 296 mots de 16 ou 32 bits, donc respectivement 8 ou 16 giga-octets.
- b. $4 \times 8 \times 1$ méga-octets = 32, dans le cas où il y a 1 octet par adresse.
- c. $\log_2 4\,194\,304 = 22$. Adresse maximale avec 22 bits = $3FFFFFF_{16}$.
- d. Chaque puce de 1 mégabit nécessite 20 bits d'adresse d'où $20/2 = 10$ broches d'adresse pour une mémoire dynamique.

7.2 L'adresse de 32 bits sera découpée comme suit :

14	16	2
Étiquette	Index	No. dans le bloc

En effet, pour adresser 64 K rangées, on a besoin d'un index de 16 bits. Pour adresser l'un des quatre octets dans chaque rangée, on a besoin de 2 bits. Restent 14 bits pour l'étiquette.

- a. Il y a donc 16 bits dans l'index.
- b. Il y a 14 bits dans l'étiquette.
- c. Pour l'adresse 0x 0A001000, on a, en binaire :

0000 1010 0000 0000 0001 0000 0000 0000 qui est découpée comme suit :

00 0010 1000 0000 | 0000 0100 0000 0000 | 00

L'adresse 0x 0A001000 sera donc dans la rangée 0x 0400 .

- d. La valeur de l'étiquette pour cette adresse est 0x 0280.

7.3 a. Temps pour effectuer un tour complet :

$$\frac{60 \text{ s / min}}{3\,600 \text{ tours / min}} = 0,016666 \text{ s / tour} = 16,67 \text{ ms / tour}$$

- b. Capacité linéaire brute par piste :
- nombre d'octets par piste = nombre d'octets/tour =

$$\frac{\text{nombre d'octets / s}}{\text{nombre de tours / s}} = \frac{1,86 \text{ mégaoctets / s}}{60 \text{ tours / s}} = 31 \text{ Kilo-octets / tour}$$

c. Capacité totale du disque non formaté :

$$\begin{aligned} & 31 \text{ Ko/piste} \times 1\,123 \text{ pistes/face} \times 9 \text{ faces} \\ & = 313\,317 \text{ Ko} = 313 \text{ mégaoctets.} \end{aligned}$$

7.4 a. Débit : puisqu'on parcourt 1 piste par tour,

$$\frac{3\,367 \text{ tours / min} \times 8\,192 \text{ octets / tour}}{60 \text{ s / min}} = 459\,707 \text{ octets / s}$$

b. Capacité brute non formatée du disque :

$$4 \text{ faces} \times 612 \text{ pistes/face} \times 8\,192 \text{ octets/piste} = 20 \text{ mégaoctets.}$$

c. Nombre de secteurs de 512 octets :

$$\begin{aligned} & 8\,192 \text{ octets/piste} \times 76\% \div 512 \text{ octets/secteur} = 12 \text{ secteurs/piste} \\ & 4 \text{ faces} \times 604 \text{ pistes/face} \times 12 \text{ secteurs/piste} = 28\,992 \text{ secteurs} \end{aligned}$$

Chapitre 8

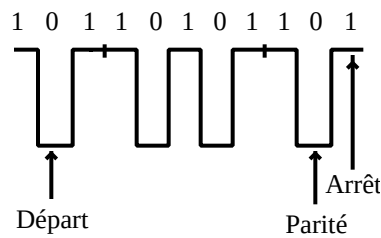
8.1		Accumulateur	Pile
LOAD	C	C	vide
STORE	-(SP)	C	C
LOAD	D	D	C
ADD	(SP)+	C+D	vide
STORE	-(SP)	C+D	C+D
LOAD	B	B	
DIV	(SP)+	$B/(C+D)$	vide
STORE	-(SP)	$B/(C+D)$	$B/(C+D)$
LOAD	A	A	
SUB	(SP)+	$A-B/(C+D)$	vide
STORE	-(SP)	$A-B/(C+D)$	$A-B/(C+D)$
LOAD	G	G	
STORE	-(SP)	G	G, $A-B/(C+D)$
LOAD	E	E	
STORE	-(SP)	E	E, G, $A-B/(C+D)$
LOAD	F	F	
MPY	(SP)+	$E * F$	G, $A-B/(C+D)$
SUB	(SP)+	$(E * F) - G$	$A-B/(C+D)$
ADD	(SP)+	$A-B/(C+D) + (E * F) - G$	vide
STORE	R	$A-B/(C+D) + (E * F) - G$	

Chapitre 10

- 10.1** a. $600 \times 800 \times 1 \text{ bit/pixel} = 480\,000 \text{ bits} = 60\,000 \text{ octets}$
b. $4 \text{ bits/pixel} \times 480\,000 \text{ pixels} = 1\,920\,000 \text{ bits} = 240\,000 \text{ octets}$
c. $3 \times 1\,920\,000 = 5\,760\,000 \text{ bits} = 720\,000 \text{ octets.}$
- 10.2** a. $300 \times 400 \text{ pixels/image} \times 5 \text{ images/s} \times 3 \text{ bits/pixel} = 1\,800\,000 \text{ bits/s}$
b. $120\,000 \div 50 \text{ pixels/image} \times 5 \text{ images/s} \times 21 \text{ bits/pixel} = 252\,000 \text{ bits/s}$
- 10.3** 10 pages/min, 60 lignes chacune, 80 car/ligne, 8 bits/car. + bits de départ, d'arrêt et de parité = 11 bits.
Vitesse minimale = $(10/60) \times 60 \times 80 \times 11 = 8\,800 \text{ bits/s.}$

Chapitre 11

11.1 'k' = 6B₁₆. On ajoute le bit de départ, on inverse les 7 bits du caractère dans le temps, on ajoute la parité (ici ce sera un 0 puisque 0x 6B contient déjà un nombre impair de 1) et le bit d'arrêt. On aura donc :



- 11.2** a. $1 \div 38\,400 \text{ bauds} = 0,000026 \text{ s} = 26 \mu\text{s}$.
b. 11 bits par caractère (octet). Nombre d'octets effectif = $38,400 \div 11 = 3\,491$.

Chapitre 12

12.1 a) La mémoire associative doit contenir l'adresse physique des 2048 pages de 512 octets ($2\,048 \times 512$ octets = 1 Mo) de la mémoire physique. Elle doit donc comporter 2 048 entrées. Puisque $512 = 2^9$, pour chaque entrée on a besoin de $46 - 9 = 37$ bits pour l'étiquette. Et puisque $2\,048 = 2^{11}$, la donnée correspondant à chaque étiquette, le numéro de page physique, nécessite 11 bits. Chaque entrée a donc $37 + 11 = 48$ bits. La capacité totale de la mémoire associative est donc de $2\,048 \times 48 \div 8 = 12\,288$ octets.

b) Pour l'adresse physique 0x 89045,

1000	1001	0000	0100	0101
Numéro			Offset	
de page			dans la page	
physique				

l'entrée de la mémoire associative doit contenir le numéro de page physique 448, donné par les 11 premiers bits de l'adresse physique. C'est ce qu'on trouve dans la 4^e rangée du tableau de la donnée du problème. Les 9 derniers bits de l'adresse physique aussi bien que de l'adresse virtuelle sont l'offset dans la page de 512 octets. Les 48 bits de la 4^e entrée de la mémoire virtuelle sont, d'après le tableau de la donnée du problème, 199939AA0448, soit :

0001	1001	1001	1001	0011	1001	1010	1010	0000	0100	0100	1000
Étiquette									Numéro de		
									page physique		

L'adresse virtuelle est donc, en remplaçant les 11 derniers bits par les 9 bits de l'offset dans la page :

0001	1001	1001	1001	0011	1001	1010	1010	0000	0	0	0100	0101
Étiquette									Offset			
									dans la page			

soit 06664E6A8045.

12.2 En reprenant l'exemple de mémoire virtuelle de la section 15.5, mais avec une mémoire virtuelle de 512 Mo, une mémoire physique de 32 Ko et des pages de 8 Ko :

a. Quelle serait la taille de la table des pages ?

La mémoire virtuelle comporte $512 \text{ Mo} / 8 \text{ Ko} = 64 \text{ K}$ pages de 8 Ko. La table des pages devra donc contenir 64 K entrées, i.e. une par page.

b. Quel est le numéro de page virtuelle correspondant à l'adresse virtuelle 0x 00300B2A?

Les 13 bits les moins significatifs de l'adresse virtuelle servent à identifier l'adresse dans la page. Les 19 bits restants de l'adresse virtuelle constituent le numéro de page virtuelle. Pour l'adresse 0x 00300B2A, on a :

0000 0000 0011 0000 000	0 1011 0010 1010
Numéro de page virtuelle	Adresse dans la page

et le numéro de page virtuelle est 000 0000 0001 1000 0000 soit 0x 00180.

12.3 a. Nombre de clusters dans la FAT :

On a besoin de 20 mégaoctets = $\frac{20 \times 1\,024 \times 1\,024 \text{ octets}}{512 \text{ octets / cluster}} = 40 \text{ Kilo-clusters}$

On a donc besoin de $40 \times 1\,024$ entrées de FAT de 16 bits, c.-à-d. de 2 octets chacune pour adresser ces 40 Kilo-clusters, soit :

$40 \times 1\,024 \times 2 \text{ octets}$, soit $80 \times 1\,024 \text{ octets} \div 512 \text{ octets/cluster}$
= 160 clusters.

b. Capacité totale de la FAT : avec 16 bits, on a 65 536 adresses possibles, adressant chacune un cluster de 512 octets, donc une capacité de :
 $65\,536 \times 512 = 32 \text{ mégaoctets}$.

12.4 Le nom du fichier est TP3.TXT.

Il a été sauvegardé à l'heure donnée par 0x B4C7, soit :

1 0 1 1 0 $\frac{1}{2}$ 1 0 0 1 1 0 $\frac{1}{4}$ 0 1 1 1 = 22 h, 38 min et 14 sec.

La date est 0x1378, soit :

0 0 0 1 0 0 1 $\frac{1}{2}$ 0 1 1 $\frac{1}{4}$ 1 0 0 0 = 9, 11, 24 = le 24 novembre 1989.

Le fichier commence au cluster #5 dans la FAT et sa taille est 0x 01AB octets, soit 427 octets.

12.5 a. $C = (F + 1).(P + 1).S.B$, par exemple, $2 \times 42 \times 9 \times 512 = 387\,072 \text{ octets}$.

b. Nombre total des secteurs : $(F + 1).(P + 1).S = 2 \times 42 \times 9 = 756$, donc $n = [0, 755]$.

- c. $p = (n \text{ DIV } S) \text{ DIV } (F + 1)$
 $f = (n \text{ DIV } S) \text{ MOD } (F + 1)$
 $s = (n \text{ MOD } S) + 1$
 Exemple : si $n = 319$, alors $p = 17$, $f = 1$, $s = 5$
- d. Nombre maximum de fichiers : on a 7 secteurs pour le répertoire, donc 7×512 octets, et chaque entrée occupe 32 octets. On a donc un maximum de 112 fichiers.
- e. La table des allocations est contenue dans 3 secteurs de 512 octets. Chaque entrée est de 2 octets, on peut donc adresser $512 \times 3 \div 2 = 768$ secteurs. Le disque contient $756 - 15 = 741$ u.a. allouables, donc la table des allocations est assez grande.
- f. La perte moyenne d'espace sur le disque est d'un demi secteur par fichier, soit $112 \text{ fichiers} \times 512 \text{ octets} \div 2 = 28\,672$ octets.

- 12.6** a. Taille maximum d'un fichier sans indirection : $20 \times 2\,048 = 40\,960$ octets.
- b. Taille maximum d'un fichier avec indirection 1 :
 $40\,960 + 512 \times 2\,048 = 1\,089\,536$ octets.
- c. Taille maximum d'un fichier avec indirection 2 :
 $1\,089\,536 + 512 \times 512 \times 2\,048 = 537\,960\,448$ octets.
- d. Position du 8 millionième octet : $8\,000\,000 - 1\,089\,536 = 6\,910\,464$ ème octet d'indirection 2.
 $6\,910\,464 \div (512 \times 2\,048) = 6$, reste 619 308 au niveau 1
 $61\,9308 \div 2\,048 = 302$, reste 812 au niveau 2.
 Réponse : ramification 6, niveau 1, ramification 302 niveau 2, octet 812.

