Chapitre 1

Architecture des ordinateurs.

Chapitre 1

Architecture des ordinateurs.

Étymologie. Le mot informatique vient de la fusion des mots **information** et **automatique**. Ainsi l'informatique est la science du traitement automatique de l'information.

Analogie. On distingue dans l'informatique deux parties : l'aspect matériel (Hardware) et l'aspect logiciel (software). En comparant l'ordinateur à un être humain, le matériel est le corps de l'ordinateur, les logiciels forment son esprit.

But. Dans ce chapitre, on va s'intéresser à l'aspect matriciel, c'est-à-dire les composants de l'ordinateur.

I. Composants de l'ordinateur.

I.1/ La mémoire principale.

- Principalement, la mémoire est une suite de signaux électriques (souvent des condensateurs) ne pouvant prendre que deux états :
  - Potentiel maximum, on note 1,
  - Potentiel minimum, on note 0.

  Chaque signal électrique est appelé un bit. Il ne peut prendre que deux valeurs : 0 ou 1.
- On regroupe les bits par 8, on obtient un octet.
- Ainsi la mémoire d'un ordinateur peut être représentée par une armoire de rangement constituée de différents tiroirs numérotés. Les tiroirs sont les octets, les numéros sont les adresses.

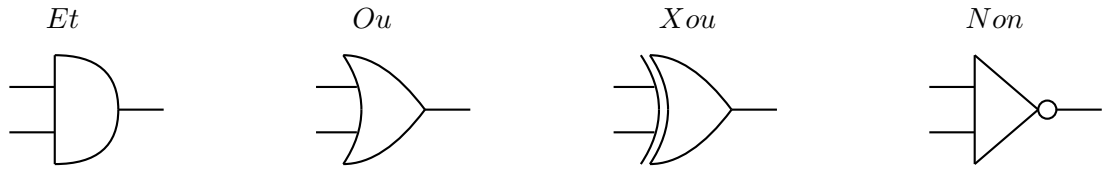
Adresses	Octets
1	
2	
3	
4	

- La mémoire principale a besoin d'énergie pour maintenir l'état des bits. Ainsi une coupure d'électricité fait perdre toutes les informations contenues dans la mémoire.
- Le disque dur est un autre type de mémoire. On écrit l'information de manière magnétique. Ainsi l'information est sauvegardée même à l'arrêt de la machine. Malheureusement l'accès au disque dur est beaucoup plus lent.

Question : combien un octet a-t-il d'état ? C'est-à-dire combien peut-il prendre de valeurs ?

I.2/ Le processeur.

- Ce que fait le processeur :
  - Lire/Écrire une case mémoire.
  - Décoder une instruction, c'est-à-dire assimiler à un entier une action à réaliser (unité de commande).
  - Effectuer des opérations arithmétiques comme +, ×, *et*, *ou*, ... (unité de traitement)
- Il existe une petite mémoire dans le processeur, c'est le registre.
- Comment construit-on un processeur ? A l'aide du silicium cristallisé puis dopé, on crée des portes logiques électriques. Voici les représentations des principales portes :



Exercice I.2.1. Construction d'un additionneur.

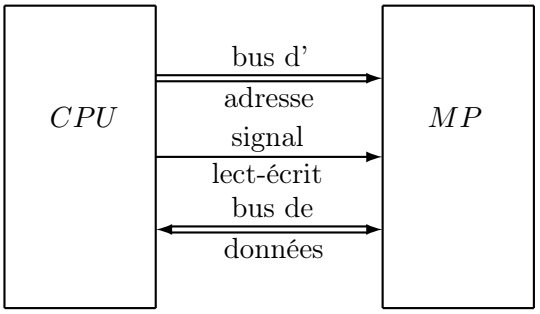
1. Compléter la table de vérité d'un additionneur ayant 3 entrées E_1, E_2 et E_3 et 2 sorties U (unité) et R (retenue) telles que le nombre binaire RU soit la somme $E_1 + E_2 + E_3$.

E_1	E_2	E_3	R	U
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

2. Effectuer les tables de vérités de
   a) $(E_1\ xou\ E_2)\ xou\ E_3$
   b) $[(E_1\ xou\ E_2)\ et\ E_3]\ ou\ (E_1\ et\ E_2)$
3. En déduire un logigramme permettant d'effectuer l'additionneur de la question 1.
4. A l'aide du logigramme déjà créé, construire un additionneur 4 bits, c'est-à-dire un logigramme permettant d'obtenir à partir de nombres binaires $A = a_1a_2a_3a_4$ et $B = b_1b_2b_3b_4$ un nombre $S = A + B$ ainsi qu'une retenue éventuelle.

I.3/ Les bus.

- Les bus permettent le transfert des informations entre les différentes parties de l'ordinateur. Par exemple entre le processeur et la mémoire principale :



I.4/ Interfaces entrée/sortie - Périphériques.

- Il y a ensuite tous les périphériques : clavier, écran, souris, imprimante, modem, ... Les interfaces d'entrée-sortie sont des circuits imprimés permettant la communication entre les périphériques et le processeur.

I.5/ Un exemple d'exécution d'une tache par le processeur.

Ce que fait le processeur pour exécuter l'instruction "Ajouter 21 à la case mémoire 402." :

1. Le processeur décode l'instruction.
2. Le processeur lit la case mémoire 402 et place son contenu dans l'accumulateur (une adresse du registre).
3. Le processeur ajoute 21 à la valeur de l'accumulateur.
4. Le processeur écrit la valeur de l'accumulateur dans la case mémoire 402.

II. Écriture binaire.

II.1/ Écriture en base quelconque.

Soit b un entier naturel supérieur ou égal à 2. Pour écrire un nombre réel en base b, on a besoin de b symboles que l'on a ordonnés. On affecte ensuite la valeur 0 au 1^{er} symbole, 1 au $2^{\text{ième}}$, etc... Ces symboles sont appelés les chiffres. Exemples :

Base	Nom	Chiffres
10	Décimale	0, 1, 2, 3, 4, ..., 9
2	Binaire	0, 1
8	Octale	0, 1,..., 8
16	Hexadécimale	0, 1,..., 9, A,B,C,D,E,F

Ainsi en hexadécimal, A a pour valeur 10, B a pour valeur 11, ... F a pour valeur 15. Une écriture en base b est donc une expression du type :

$$\overline{a_n \dots a_1 a_0 \, , \, a_{-1} a_{-2} a_{-3} \dots}^{b} \qquad \text{ou} \qquad (a_n \dots a_1 a_0 \, , \, a_{-1} a_{-2} a_{-3} \dots)_b$$

avec les a_i choisis parmi les chiffres. Si les a_i sont nuls en dessous d'un certain rang, on ne les écrit pas. Ainsi $\overline{1,2000000\dots}^{15} = \overline{1,2}^{15}$

II.2/ Valeur d'une écriture.

L'expression $\overline{a_n \dots a_1 a_0 \, , \, a_{-1} a_{-2} a_{-3} \dots}^{b}$ a pour valeur :

$$\sum_{k=-\infty}^{n} a_k b^k$$

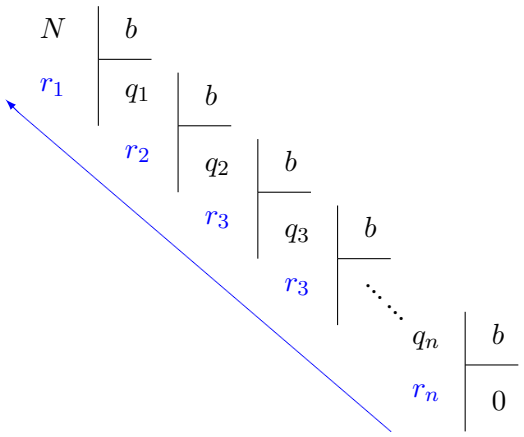
Exemples.

Écriture	Valeur
$\overline{24,2}^{8}$	
$\overline{1001,101}^{2}$	
$\overline{9F,A}^{16}$	
$\overline{0.222\dots}^{3}$	

Remarque. Pour multiplier un nombre écrit en base b par b, il suffit de décaler la virgule sur la droite. Pour diviser par b, on décale la virgule sur la gauche.

II.3/ Écriture associée à un entier naturel.

Pour trouver l'écriture en base b d'un nombre N on effectue les divisions euclidiennes successives de N par b jusqu'à obtenir un quotient de 0 :



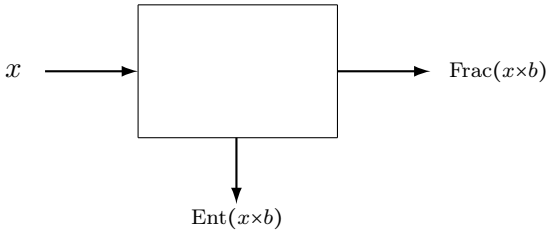
L'écriture en base b de N est alors : $\overline{r_n \dots r_2 r_1}^b$. Attention à l'ordre des chiffres !

Exemples.

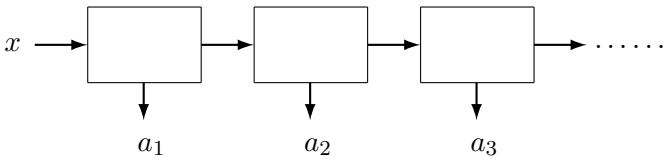
Nombre	Base	Écriture dans cette base
23	2	
46	3	
152	16	

II.4/ Écriture associée à un réel compris entre 0 et 1.

On rappelle tout d'abord que la partie fractionnaire d'un nombre x, noté $Frac(x)$ est égal à $x - Ent(x)$ où $Ent(x)$ désigne la partie entière de x. Considérons l'opération sur x schématisée par :



Pour trouver l'écriture d'un nombre x de $[0,1[$ en base b, il suffit de réitérer l'opération précédente de la façon suivante :



L'écriture en base b de x est : $\overline{0,a_1a_2a_3\ldots}^{b}$.

Exemples.

Nombre	Base	Écriture dans cette base
0.25	12	
0.625	2	
0.1	2	

II.5/ Écriture associée à un réel quelconque.

Quelle est l'écriture en base b d'un réel x quelconque ?

1. Les chiffres avant la virgule sont les chiffres de l'entier $Ent(|x|)$.
2. Les chiffres après la virgule sont les chiffres du réel $Frac(|x|)$ appartenant à $[0,1[$.
3. On ajoute un signe – si le nombre est négatif.

Exemples.

Nombre	Base	Écriture dans cette base
–10.1	20	
7.3	2	
28.5	16	

Remarque. Tous les nombres ayant un nombre fini de chiffres après la virgule

1. en base 2 ont un nombre fini de chiffres après la virgule en base 10.
2. en base 10 n'ont **pas** forcément un nombre fini de chiffres après la virgule en base 10. Cex : 0,1

III. Stockage des données simples.

III.1/ Codage d'un entier naturel.

Pour coder un entier naturel x :

1. on choisit tout d'abord sur combien d'octets on code cet entier. Dans de nombreux langages, ce nombre est fixe (Souvent 4, pour les entiers normaux). En Python, le programme choisit en fonction de la grandeur du nombre.
2. on écrit ensuite x en binaire que l'on complète par des 0 devant pour que le nombre de chiffre du nombre x en base 2 coïncide avec le nombre d'octets choisi. On code ensuite chaque chiffre sur un bit. Si le nombre de chiffre en binaire de x dépasse déjà le nombre d'octet disponible, ce nombre ne peut être stocké.

Exemple. Essayons de coder 43 sur 1 octet.

On convertit 43 en binaire	:	10 1011
On convertit complète par des 0	:	0010 1011

Ainsi 43 est codé par :

0	0	1	0	1	0	1	1

Remarque. Sur n bits, les nombres de 0 à $2^n - 1$ peuvent être mémorisés.

Attention. Quand on fait des opérations sur les entiers, le résultats peut être faux (l'utilisateur en est averti). Par exemple, si le codage des entiers se fait sur 1 octet :

$$255 + 1 \quad = \quad \overline{11111111}^2 + \overline{00000001}^2 \quad = \quad \overline{00000000}^2 \quad = \quad 0$$

Exemple. Déterminer le codage des entiers suivants sur un octet :

5	$-->$								
27	$-->$								
112	$-->$								

III.2/ Codage d'un entier relatif.

Une première idée... qui n'est pas bonne. Pour coder l'entier relatif x, on réserve le premier bit pour coder le signe de x, puis on code avec sur les autres bits $|x|$. Mais ce n'est pas bien car (sur 1 octet) :

$$2 + (-2) \quad = \quad 0000\,0010 \; + \; 1000\,0010 \quad = \quad 1000\,0100 \quad = \quad -4$$

Soit il faut apprendre au calculateur à recalculer (on double les câblages et le coût de production) soit on code différemment. C'est ce qu'on va faire. Pour coder $-x$ avec x dans $\mathbb{N}$, on cherche à avoir $-x + x = 0$. Une seule possibilité :

Le complément à 2. Pour coder $-x$ avec x dans $\mathbb{N}$,

1. On code x.
2. On prend le complémentaire (les bits égaux à 1 passe à 0 et inversement).
3. On ajoute 1.

Exemple. Pour coder −17 sur 1 octet :

On code 17	0001 0001
On cherche le complémentaire	1110 1110
On ajoute 1	1110 1111

Ainsi -17 est codé par

1	1	1	0	1	1	1	1

Codage des entiers sur 1 octet.[1]

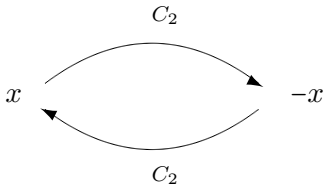
127	− >	0	1	1	1	1	1	1	1
⋮					⋮				
2	− >	0	0	0	0	0	0	1	0
1	− >	0	0	0	0	0	0	0	1
0	− >	0	0	0	0	0	0	0	0
−1	− >	1	1	1	1	1	1	1	1
−2	− >	1	1	1	1	1	1	1	0
⋮	− >				⋮				
−128	− >	1	0	0	0	0	0	0	0

Remarques.

1. Ne pas confondre l'écriture binaire d'un entier négatif et son codage. En effet :

$$-3 \;=\; -\overline{00000010}^{2} \quad \text{se code en :} \quad \begin{array}{|c|c|c|c|c|c|c|c|}\hline 1&1&1&1&1&1&0&1\\\hline\end{array}$$

2. Le complément à 2 est involutif, ainsi pour passer du codage de x à celui de $-x$, ou pour passer du codage de $-x$ à à celui de x on effectue le complémentaire à 2.



Où C_2 est le complémentaire à 2.

3. le premier bit est un bit de signe. Il vaut 0 si le nombre est positif et -1 sinon.
4. Sur n bits, on peut donc coder les entiers relatifs de -2^{n-1} à $2^{n-1}-1$.

Exemples.

valeur de x
27

codage de x							
0	0	1	0	0	1	1	0

codage de $-x$							
1	1	1	0	1	0	0	1

III.3/ Codage d'un caractère.

Pour stocker un caractère, l'ordinateur utilise une table de conversion qui associe à chaque caractère un entier naturel. Il suffit de coder cet entier naturel. La table de conversion la plus connu est la table ASCII (American Standard Code for Information Interchange). La voici :

The ASCII character set

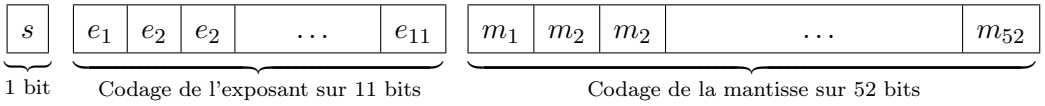
Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	NUL	32	20	SP	64	40	@	96	60	`
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	'	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	−	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

III.4/ Codage d'un réel.

Pour coder un nombre réel, on commence par le décomposer comme suit :

$$x = (-1)^s \times 1, m_1 m_2 m_3 \ldots \times 2^e$$

s est le signe, $m_1 m_2 \ldots$ la mantisse et e l'exposant. Outre un bit pour coder le signe (0 ou 1), Il faut ensuite de choisir sur combien de bits seront codés la matisse et l'exposant. En général (en Python en particulier), on choisit respectivement 11 bits et 52 bits.



Remarques culturelles.

1. L'exposant n'est pas codé comme un entier relatif classique. On code en fait $e + 1023$ et il faut que cette somme soit comprise entre 1 et $2^{11} - 2$
2. Les exposants 0 et $2^{11} - 1$ sont réservés à des exceptions pour coder des nombres particuliers. Par exemple O est codé par une mantisse nulle et un exposant nul.

Attention. Les décimales au delà de la $52^{\text{ième}}$ décimale sont supprimées. Ainsi le codage d'une réel n'est pas toujours exact. Il est donc très fortement conseiller de ne **jamais** tester l'égalité de deux réels ! Par exemple en Python :

```
0.1 + 0.1 + 0.1 == 0.3
```

est **faux !**