1. Opérations en binaire non signé.

1.1. Addition.

Elle est identique à l'addition décimale (retenues incluses).

Exemple :

	^{1}	1^{1}	1^{1}	1	0	1	0^{1}	1	1			2^{1}	3^{1}	5
+		0	1	1	0	0	0	1	0	+		9	8	
=	1	0	1	0	0	1	1	0	1	=		3	3	3

1.2. Nombre minimal, nombre maximal.

En électronique, on est toujours limité à N bits. On peut, par exemple, utiliser un microprocesseur 8 bits.

Le nombre minimal est donc % 0000 0000 (soit ! 0), et le nombre maximal % 1111 1111 (soit ! 255). Le microprocesseur ne pourra *a priori* pas gérer un nombre inférieur à 0 ou supérieur à 255.

1.3. Comptage.

Cette limite présente une particularité lors de l'incrémentation d'une variable: une fois que l'on est arrivé au maximum (ici % 1111 1111), on repasse à 0 (% 0000 0000).

...	...	
254	1111 1110	
255	1111 1111	↵ +1
0	1 0000 0000	↵ +1
1	0000 0001	↵ +1
...	...	
	bit perdu	

Ce neuvième bit n'est généralement pas vraiment perdu : il est disponible comme une retenue, sur une broche annexe.

2. Opérations en binaire signé, sur 8 bits.

En électronique, c'est le système qui détermine si le nombre est supérieur ou inférieur à zéro, par programmation. C'est-à-dire que l'on conçoit un système pour travailler uniquement en nombres positifs, ou uniquement en nombres signés (on mélange rarement ces deux types de gestion).

2.1. Un bit de signe.

Le bit de gauche représente le signe, par convention (0 : signe positif ; 1 : signe négatif). La valeur absolue est donc représentée sur 7 bits au lieu de 8.

Cette méthode n'est pas suffisante car on a alors +0 et –0, peu pratique pour le traitement de l'information :

0111 1111	+127	nombre *maximal* disponible
...	...	
0000 0010	+2	
0000 0001	+1	
0000 0000	+0	⎫ deux valeurs binaires équivalent
1000 0000	–0	⎭ à la même valeur décimale
1000 0001	–1	
1000 0010	–2	
...	...	
1111 1111	–127	nombre *minimal* disponible

2.2. Le complément à deux.

On utilise en pratique le *complément à deux* pour éviter ces +0 et –0 :

– Un nombre positif n'est pas modifié,

– Un nombre négatif se trouve (à partir de sa valeur absolue) de la façon suivante :

– On inverse sa valeur bit à bit (complément à 1),

– On incrémente de 1 le résultat trouvé (complément à 2).

Exemple 1 :

+27 0001 1011

27

signe +

–27

+27	0001 1011		
$\overline{+27}$	1110 0100	↵ CPL1	⎫
–27	1110 0101	↵ +1	⎭ CPL2

signe –

Exemple 2 :

+124 0111 1100

124

signe +

–124

+124	0111 1100		
$\overline{+124}$	1000 0011	↵ CPL1	⎫
–124	1000 0100	↵ +1	⎭ CPL2

signe –

Exemple 3 :

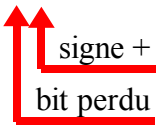
Cas particulier du zéro.

+0 0000 0000

0

signe +

–0	+0	0000 0000		
	+0	1111 1111	↵ CPL1	⎫
	–0	1 0000 0000	↵ +1	⎭ CPL2



On retrouve donc exactement la valeur de départ. Notre 0 n'est donc pas dupliqué en +0 et -0.

2.3. Poids.

Le fait de travailler en complément à deux a une incidence directe sur le poids du bit le plus à gauche ; il devient *négatif*.

Poids en binaire naturel non signé :

b7	b6	b5	b4	b3	b2	b1	b0
+128	+64	+32	+16	+8	+4	+2	+1

Poids en binaire naturel signé :

b7	b6	b5	b4	b3	b2	b1	b0
–128	+64	+32	+16	+8	+4	+2	+1

2.4. Interprétation.

La valeur décimale du nombre dépend de la façon dont est programmé le système qui utilise la donnée (on le conçoit pour qu'il interprète une donnée de telle ou telle façon) :

+27	0001 1011	+27
–27	1110 0101	+229
+124	0111 1100	+124
–124	1000 0100	+132
Si on travaille en nombres signés		Si on travaille en nombres non signés

2.5. Nombre minimal, nombre maximal.

Le fait de réserver un bit pour la gestion du signe entraîne une diminution de moitié de la capacité des nombres positifs.

Le 0 est considéré comme un nombre positif et, de ce fait, la gestion des nombres est asymétrique : on peut désormais compter de –128 à +127.

1000 0000	–128	nombre minimal
1000 0001	–127	
...	...	
1111 1111	–1	
0000 0000	+0	
0000 0001	+1	
...	...	
0111 1110	+126	
0111 1111	+127	nombre maximal

2.6. Addition, soustraction.

L'addition suit les mêmes règles qu'en nombres non signés.

Elle est donc utilisée pour réaliser des soustractions : il suffit d'additionner une valeur au complément à deux d'une seconde valeur.

Exemple 1 :

83–119 = 83+(–119)

+83	0101 0011
+119	0111 0111
+119	1000 1000
−119	1000 1001

	0	1	0	1	0	0^1	1^1	1				8	3
+	1	0	0	0	1	0	0	1	+	–	1	1	9
=	1	1	0	1	1	1	0	0	=	–		3	6

signe –

Ce résultat suffit à un microprocesseur. Néanmoins, il nous faut à nouveau appliquer un complément à deux si l'on veut connaître la valeur[1], car le résultat nous a donné un nombre signé négatif :

−?	1101 1100
−?	0010 0011
+?	0010 0100 = 36

Le résultat de notre addition (ou soustraction) est donc % 1101 1100, soit –36.

Exemple 2 :

124–27=124+(–27)

+124	0111 1100
+27	0001 1011
+27	1110 0100
−27	1110 0101

	0^1	1^1	1^1	1^1	1^1	1	0	0			1	2	4
+	1	1	1	0	0	1	0	1	+	–		2	7
= 1	0	1	1	0	0	0	0	1	=			9	7

signe +

bit perdu

Le résultat de notre addition (ou soustraction) est donc % 0110 0001, soit +97.

1 Ou appliquer la règle des poids vue en **2.3.**