Révision de la manipulation des entiers en base 2 et 16

1 Ecriture des entiers naturels en base 2 et 16

1.1 Introduction

Les organes d'un ordinateur sont dimensionnés à un nombre fixe n de bits. Par exemple, les registres, les unités de calcul, le bus d'accès à la mémoire d'un ARM7 sont tous dimensionnés à 32 bits. Tous les calculs sont alors réalisés modulo 2^n (environ quatre milliards pour $n = 32$ bits).

Un entier E peut être représenté par une suite de n chiffres (ou digits) e_i, tous inférieurs à la base utilisée ($0 \leq e_i \leq B-1$) et tels que $E = \sum_{i=0}^{n-1} e_i * B^i$. Chaque chiffre e_i représente le reste de la division entière de E/B^i par B. La base B est éventuellement précisée en indice à droite du dernier chiffre ou entre parenthèses. Par défaut, il s'agit de la base 10.

L'entier composé des k chiffres de poids faibles de E est E modulo [1] 2^k et celui composé des n-k chiffres de poids forts de E est E / 2^k. Exemple pour n=5 et k=2 :
$23_{10} = 5 * 4 + 3 = 10111_2$. $23/4 = 5 = 101_2$. $23\ modulo\ 4 = 3 = 11_2$

La table en annexe donne les principales puissances de 2, ainsi que la valeur binaire et décimale de chaque chiffre hexadécimal.

101_2	=	$1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$	=	$4 + 1$	=	5_{10}
101_{10}	=	$1 \times 10^2 + 0 \times 10^1 + 1 \times 10^0$	=	$100 + 1$	=	101_{10}
101_{16}	=	$1 \times 16^2 + 0 \times 16^1 + 1 \times 16^0$	=	$256 + 1$	=	257_{10}
$A4_{16}$	=	$10 \times 16^1 + 4 \times 16^0$	=	$10 * 16 + 4$	=	164_{10}

Conversions entre bases 2, 10 et 16 :

1. 2 vers 16 : ajouter éventuellement des 0 à gauche pour avoir 4k bits, convertir les k quartets en k chifffres hexadécimaux.
2. 16 vers 2 : convertir chaque chiffre hexdécimal en quartet de bits
3. 10 vers B=2 ou B=16 : diviser par B, le reste donne un chiffre (poids faible), recommencer avec le quotient, etc : le dernier reste donne le chiffre de poids fort.

Exemples :
$178_{10} = B2_{16}$: $178/16\ quotient\ 11, reste\ \boxed{2}\ (poids\ faible), 11/16$: $quotient\ 0\ reste\ 11(\boxed{\text{B}}\ (poids fort))$
$11_{10} = 1011_2 = 5 * 2 + \boxed{1}\ (poids\ faible), 5 = 2 * 2 + \boxed{1}, 2 = 1 * 2 + \boxed{0}, 1 = 0 * 2 + \boxed{1}\ (poids\ fort)$
$1001101101_2 \rightarrow \mathbf{00}10\ 0110\ 1101 \rightarrow 26B_{16}$

1.2 Propriété remarquable

$$\sum_{i=0}^{n-1} a^i = \frac{a^n - 1}{a - 1}\ et\ \sum_{i=0}^{n-1} 2^i = 2^n - 1$$

En effet $(a^{n-1} + a^{n-2} + \ldots + a^1 + 1)(a-1) = (a^n - a^{n-1} + a^{n-1} - a^{n-2} \ldots + a - 1) = (a^n - 1))$

L'entier dont la représentation est constituée de n bits à 1 est $2^n - 1$: $11111111_2 = 2^8 - 1 = 255_{10}$

1. modulo = reste de la division entière

1.3 Compléments à 1 et à 2

Soit $E = \sum_{i=0}^{n-1} e_i * 2^i$ un entier naturel représenté sur n chiffres en base 2. . On appelle *complément à* $2^n - 1$ de E (on dit habituellement *complément à 1 de E*) l'entier $\overline{E} = \sum_{i=0}^{n} \overline{e}_i$ obtenu en remplaçant les 1 par des 0 et les 0 par des 1 ($\overline{e}_i = 1 - e_i$) dans la représentation en binaire de E. Il s'écrit ˜E en langage C. On a $E + \overline{E} = \sum_{i=0}^{n} e_i 2^i + \sum_{i=0}^{n} (1 - e_i) 2^i = \sum_{i=0}^{n-1} 2^i = 2^n - 1$, d'où $\overline{E} = 2^n - 1 - E$.

On appelle *complément à* 2^n (on dit habituellement *complément à deux*) de E l'entier naturel E l'entier naturel $2^n - E$, noté $\overline{E}^2$. Par définition, $\overline{E}^2 = \overline{E} + 1$. Soit u la position du premier un [2] dans la représentation en binaire de E . La représentation de $\overline{E}^2$ est obtenue à partir de celle de E en inversant les $n - u$ bits de poids forts et en conservant les u bits de poids faibles.

2 Addition

On rappelle le principe de calcul dans l'addition : colonne par colonne, de droite à gauche. Les retenues, habituellement placées au dessus de l'opérande gauche, sont placées ici en dessous de l'opérande droit. Dans chaque colonne, on fait la somme des chiffres du premier (a_i) et du deuxième (b_i) opérande, ainsi que la retenue entrante (c_i).

Le chiffre (res_i) du résultat res est égal à :
— cette somme, la retenue [3] sortante (c_{i+1}) étant 0, si $somme < base$,
— cette somme moins la base, la retenue sortante ($C = c_{i+1}$) étant 1, si $somme \geq base$.

	a_3	a_2	a_1	a_0	opérande gauche
$+_{base}$	b_3	b_2	b_1	b_0	opérande droit
$C = c_4$	c_3	c_2	c_1	c_0	retenues
	r_3	r_2	r_1	r_0	résultat apparent

sortante c_{i+1} — a_i, b_i, c_i, r_i — entrante

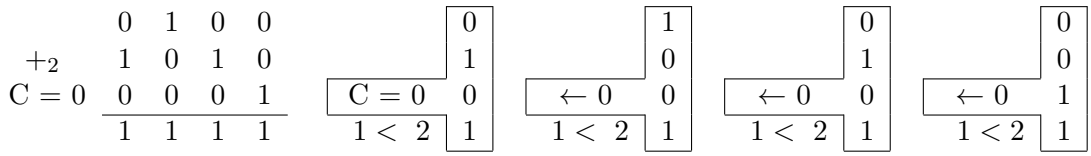
	3	6	4	3		3		6		4		3
$+_{10}$	5	7	8	5		5		7		8		5
C = 0	1	1	0	0	C = 0	1	← 1	1	← 1	0	← 0	0
	9	4	2	8	9 < 10	9	14 ≥ 10	4	12 ≥ 10	2	8 < 10	8

	0	1	1	1		0		1		1		1
$+_2$	0	1	1	0		0		1		1		0
C = 0	1	1	0	0	C = 0	1	← 1	1	← 1	0	← 0	0
	1	1	0	1	1 < 2	1	3 ≥ 2	1	2 ≥ 2	0	1 < 2	1

Dans une addition normale, la retenue entrante initiale (c_0, colonne de droite) est nulle. L'utilisation d'une retenue initiale à 1 permet de calculer l'expression $op_{gauche} + op_{droit} + 1$ (pour réaliser des soustractions par addition du complément à deux).

	1	1	0	1		1		1		0		1
$+_2$	1	0	0	1		1		0		0		1
C = 1	0	0	1	1	C = 1	0	← 0	0	← 0	1	← 1	1
	0	1	1	1	2 ≥ 2	0	1 < 2	1	1 < 2	1	3 ≥ 2	1

2. $\forall i, e_i = 1 \Rightarrow i \geq u$, (u=0 si E=0)
3. carry en anglais



3 Conventions d'interprétation (entiers naturels et relatifs)

Soit $e = \sum_{i=0}^{i=n-2} e_i 2^i$. Sur n bits, on peut coder 2^n valeurs différentes. Mais l'interprétation de ce codage n'est pas unique. En pratique, l'entier écrit $e_{n-1}\ e_{n-2}\ e_{n-3} \dots\ e_1\ e_0$ en base deux représente la valeur $E = \alpha e_{n-1} 2^{n-1} + e$.

Les règles de calcul pour l'addition et la soustraction sont les mêmes quel que soit α : seule l'interprétation des valeurs des opérandes et du résultat change.

3.1 Pour entiers naturels (N) : $\alpha = 1$ et $E = \sum_{i=0}^{i=n-1} e_i 2^i$.

En pratique, il n'est pas rare que les entiers manipulés dans la vie courante sortent de l'intervalle de valeurs représentables dans les formats inférieurs à 64 bits. A titre d'exemple, les capitalisations boursières des sociétés ne sont pas toutes représentables sur 32 bits.

Pour stocker une valeur entière toujours positive ou nulle [4], le programmeur peut décider d'utiliser une variable entière en interprétant son contenu comme un entier naturel (attribut *unsigned* de type entier en langage C) afin de maximiser l'intervalle de valeurs représentables : $[0 \dots 2^n - 1]$.

Le bit de poids fort n'a pas de signification particulière : il indique simplement si la valeur représentée est supérieure à 2^{n-1} ou pas.

Dans le langage C, le type entier naturel est spécifié avec l'attribut **unsigned**, ou les types entier naturel de taille précise **uint***xx*_**t** ($x \in \{8, 16, 32, 64\}$) définis dans stdint.h (révisions récentes du langage).

La figure en cercle 1 illustre les $2^4 = 16$ codes binaires possibles sur 4 bits (incluses dans le cercle intérieur) et (sur la couronne extérieure,en décimal) les valeurs d'entiers naturels représentées.

Chaque entier correspond à un angle de rotation depuis l'origine dans le sens trigonométrique [5]. L'addition de 2 entiers peut être interprétée comme la sommation des angles de rotation des opérandes. A partir d'un tour complet, il y a débordement (résultat apparent obtenu modulo 2^4 et C=1 qui indique qu'il faudrait un bit de plus (à 1) pour représenter le vrai résultat).

3.2 Pour entiers relatifs (Z) : $\alpha = -1$ et $E = -e_{n-1} 2^{n-1} + \sum_{i=0}^{i=n-2} e_i 2^i$.

Le bit de poids fort représente maintenant le signe de l'entier et le principe consiste à retrancher 2^n à la valeur associée aux entiers dont le bit de poids fort est à 1. Cette convention représente les entiers négatifs selon la technique du *complément à deux* [6] : l'entier relatif -x est représenté comme l'entier naturel $2^n - x$. Dans les langages, cette convention d'interprétation est généralement utilisée par défaut (type

4. Les constantes adresse et les variables pointeurs entrent dans cette actégorie.

5. ou anti-horaire

6. La convention alternative "signe (codé dans le bit de poids fort) et valeur absolue (codée sur les n-1 bits de poids faibles) a l'inconvénient de définir deux zéros : +0 et -0. Rarement utilisée pour les entiers, elle peut s'appliquer à la représentation les nombres à virgule flottante.

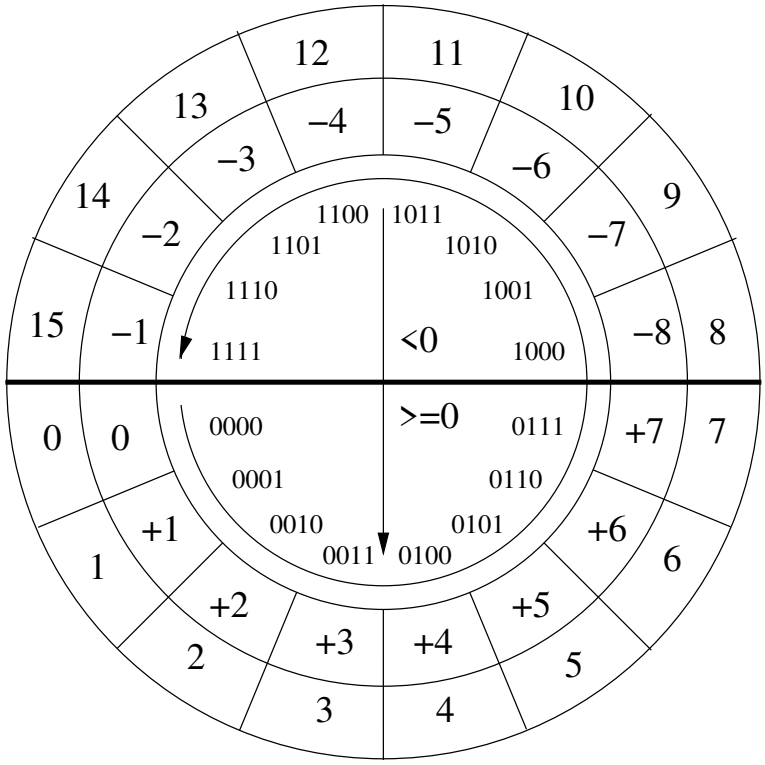


FIGURE 1 – Représentation d'entiers naturels et signés sur 4 bits

entier sans attribut unsigned ou type **int***xx*_**t** en langage C).

Un entier relatif E dont le bit de signe est 0 (≥ 0) appartient à l'intervalle $[0 \dots 2^{n-1} - 1]$ et sa valeur associée est la même que dans la convention pour entier naturels.

Un entier E dont le bit de signe est 1 (< 0) appartient à l'intervalle $[-2^{n-1}, +2^{n-1} - 1]$ et sa valeur associée est $-\overline{e}^2 = -(2^n - e)$.

— Pour calculer l'opposé d'un entier, il faut prendre le complément à deux de cet entier (et non inverser simplement le bit de signe).
— Sur n bits, l'entier -2^{n-1} est son propre complément à deux et l'entier relatif $+2^{n-1}$ n'est pas représentable.
— L'ajout d'un bit à 0 en poids fort d'un entier relatif négatif inverse son signe et change sa valeur.

La couronne intérieure de la figure 1 illustre le codage des entiers relatifs sur 4 bits. A chaque entier peut êre associé un angle de rotation dans le sens trigonométrique pour les entiers positifs ou nuls et dans le sens horaire pour les entiers négatifs.

Le code 1001 peut représenter selon la convention d'interprétation soit l'entier naturel 9 soit l'entier relatif -7. De même, 0101 est le code common aux entiers natruel 5 et relatif +5.

L'addition de deux entiers relatifs de même signe donne une erreur lorsque la somme des angles correspondant aux entiers va au-delà d'une rotation d'un demi-tour, avec un résultat apparent de signe opposé à celui des opérandes. Cette erreur vient du fait que le résultat attendu n'appartient pas à l'intervalle des valeurs représentables sur le nombre de bits de codage utilisé.

3.3 Intervalles représentables

n	Convention naturels			Convention relatifs		
n	0	à	$2^n - 1$	-2^{n-1}	à	$+2^{n-1} - 1$
8	0	à	255	-128	à	+127
16	0	à	65535 (64K_b-1)	-32768 (-32K_b)	à	+32767 (32K_b-1)
32	0	à	4294967295 (4G_b-1)	-2147483648 (-2G_b)	à	+2147483647 (2G_b-1)
64	0	à	$1,8 \times 10^{19}$(16$E_b - 1$)	-9×10^{18}(−4E_b)	à	$+9 \times 10^{18}$(+4$E_b - 1$)

Pour les bornes de l'intervalle sur 64 bits, le tableau mentionne l'ordre de grandeur (préfixé par) : la valeur exacte représente une vingtaine de chiffres. Les préfixes K_b (kilo) et M_b (méga) représentent $2^{10} = 1024_{10}$ et $2^{20} = 1048576_{10}$, dont la valeur est proche de 1000 (1K) et 1000000 (1M). Même principe pour G_b (giga : 2^{30}) et E_b (eta : 2^{60}).

4 Changement de format, manipulation booléenne des bits, décalages

4.1 Extension et réduction de format

Des conversions de taille sont nécessaires lors de la copie d'un entier entre 2 contenants de tailles différentes, par exemple un registre de 32 bits et un emplacement mémoire de 8 ou 16 bits.

La réduction de format élimine les bits de poids forts excédentaires (opération modulo). La valeur entière n'est pas modifiée si elle est représentable sur le contenant de plus petite taille.

En sens inverse, la représentation de l'entier doit être étendue en ajoutant des bits de poids forts selon la nature de l'entier :
- ajout de bits à 0 pour un entier naturel
- duplication de l'ancien bit de poids fort (bit de signe) pour un entier relatif

Il existe ainsi 3 instructions ARM de transfert d'un entier entre un registre 32 bits **regx** et un emplacement de 16 bits en mémoire **mem[y]**. L'instruction ldrh est destinée aux entiers naturels codés sur 16 bits, et ldrsh aux entiers relatifs.
- strh : regx modulo $2^{16} \rightarrow mem[y]$
- ldrh : regx $\underleftarrow{extension\ en\ ajoutant\ 16\ fois\ un\ bit\ 0}$ mem[y]
- ldrsh : regx $\underleftarrow{extension\ en\ ajoutant\ 16\ fois\ le\ bit\ de\ poids\ fort\ de}$ mem[y]

4.2 Décalage et rotation

Le décalage logique à gauche de k bits (Logic Shift Left #k en langage d'assemblage ARM, $\ll\ k$ en C) d'un entier e ajoute d bits à 0 à droite, ce qui revient à multiplier e par 2^d. Le format de représentation restant inchangé, le décalage supprime les k bits de poids forts de e. Si l'un de ces bits éjectés n'est pas 0, le résultat de la multiplication n'est pas représentable sur le nombre de bits utilisé.

Remarque : l'entier 2^k est l'entier 1 décalé de k bits à gauche (((uint32_t)1 $\ll\ k$) en C).

Une opération de rotation est un décalage dans lequel les bits ajoutés à une extrémité sont ceux qui sont éjectés de l'autre extrémité de l'entier. Une rotation à gauche de k bits et une rotation à droite de

$n - k$ bits ont le même effet.

Le décalage de k bits à droite correspond à la division par 2^k : les k bits de poids faibles sont éjectés.

Le décalage logique de k bits à droite (Logic Shift Right #k en langage d'assemblage ARM, $\gg k$ sur une variable unsigned en C) est destiné aux entiers naturels : k bits à 0 sont ajoutés en poids forts et l'entier est divisé par 2^k.

Le décalage arihmétique à droite (Arithmetic Shift Right en langage d'assemblage ARM) est destiné aux entiers relatifs : le bit de poids fort (signe) d'origine est recopié dans les bits ajoutés à gauche. L'entier est divisé par 2^k s'il en était un multiple au départ.

4.3 Opérations booléennnes bit à bit

Un chiffre de la base 2 (bit d'un entier) et un booléen ont la même écriture : 0 ou 1.
Les opérateurs bit à bit traitent un entier sur n bits comme une collection de n booléens : chaque bit de rang j du résultat correspond à une opération booléenne sur les bits de rang j des opérandes.

La négation (un seul opérande) bit à bit (~ en C) inverse tous les bits de l'entier : elle réalise le complément à 1 de celui-ci.

Les autres opérations booléennes classiques à 2 deux opérandes existent aussi en version bit à bit :
— Et bit à bit (& en C)
— Ou bit à bit (— en C)
— Ou exclusif bit à bit (^ en C, remarquer que ce n'est pas l'opérateur d'élévation à la puissance)

Noter la différence avec les opérateurs booléens classiques du C :
— 11 && 13 donne 1 ($11 \neq 0$) : vrai, $13 \neq 0$: vrai, vrai et vrai : vrai $\rightarrow 1$
— 11 & 13 donne 9 : seuls les bits 0 et 3 sont à 1 dans les deux entiers.

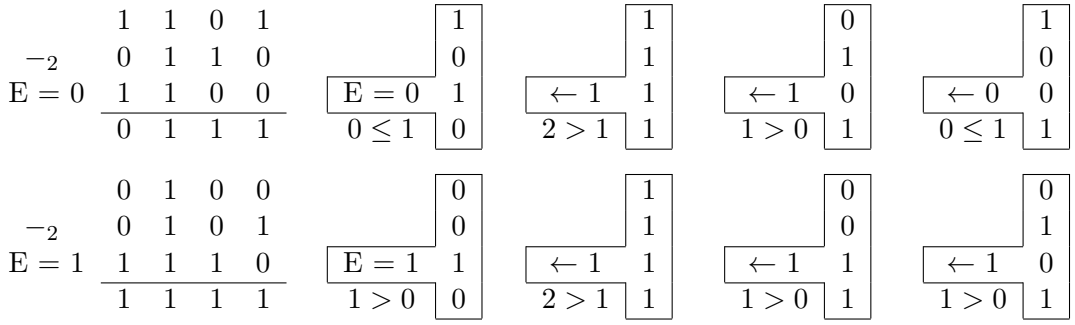
5 Soutraction

Dans chaque colonne, on fait la somme du chiffre du deuxième (b_i) opérande et de l'emprunt entrant (e_i) et l'emprunt entrant initial e_0 est nul. Le chiffre (r_i) du résultat est égal :
— au chiffre du premier opérande (a_i) moins cette somme, l'emprunt sortant (e_{i+1}) étant 0, si $somme \leq a_i$,
— au chiffre du premier opérande (a_i) plus la base moins cette somme, l'emprunt sortant (e_{i+1}) étant 1, si $somme > a_i$,

	a_3	a_2	a_1	a_0	opérande gauche
$+_{base}$	b_3	b_2	b_1	b_0	opérande droit
$E = e_4$	e_3	e_2	e_1	e_0	emprunts
	r_3	r_2	r_1	r_0	résultat apparent

sortant e_{i+1} — a_i, b_i, e_i, r_i — entrant

	8	6	4	8			8		6		4	8
$-_{10}$	5	7	9	5			5		7		9	5
E = 0	1	1	0	0		E = 0	1	$\leftarrow 1$	1	$\leftarrow 1$	0	$\leftarrow 0$	0
	2	8	5	3		$6 \leq 8$	2	$8 > 6$	8	$9 > 4$	5	$5 \leq 8$	3



6 Soustraction par addition du complément à deux

En pratique, toutes les soustractions sont réalisées par addition du complément à 2. On exploite la propriété suivante (calculs sur n bits) : $x + \overline{y}^2 = x + 2^n - y$.

Les résultats étant obtenus modulo 2^n, on peut calculer l'expression $x - y$ en effectuant une addition comme suit :

- Premier opérande : x
- Deuxième opérande : $\overline{y}$
- Retenue initiale : 1 (pour faire $x + \overline{y} + 1$)
- On observe que la ligne des retenues dans cette addition de $\overline{y}$ est le complément de la ligne des emprunts dans la soustraction normale.

Le calcul de $13 - 6$ (réalisable) et $4 - 5$ (impossible pour des entiers naturels) est illustré par les deux derniers exemples des paragraphes 5 (soustraction normale) et 2 (soustraction par addition du complément à deux).

7 Indicateurs et débordements

Lors d'une opération (addition ou soustraction) sur les entiers, l'unité de calcul d'un processeur synthétise quatre indicateurs booléens à partir desquels il est possible de prendre des décisions.

7.1 Nullité et indicateur : Z

L'indicateur Z (**Z**éro) et vrai si et seulement tous les bits du résultat apparent sont à 0, ce qui signifie que ce dernier est nul.

7.2 Signe du résultat apparent : N

L'indicateur N est égal au bit de poids fort du résultat apparent. Si ce dernier est interprété comme un entier relatif, N=1 signifie que le résultat apparent est négatif.

7.3 Débordement en convention d'entiers naturels : C

L'indicateur C (**C**arry) est la dernière retenue sortante de l'addition. Il n'a de sens que dans une interprétation de l'opération sur des entiers naturels.

Après une addition, $C = 1$ indique un débordement : le résultat de l'opération est trop grand pour être représentable sur n bits. Le résultat apparent est alors faux : il correspond au vrai résultat à 2^n près.

E est le dernier emprunt sortant d'une soustraction. $E = 1$ indique que la soustraction est impossible parce que le deuxième opérande est supérieur au premier. Les soustractions sont en pratique réalisées par addition du complément à deux. C correspond alors à $\overline{E}$. Après une soustraction par addition du complément à deux, $C = 0$ indique que la soustraction est impossible, $C = 1$ que l'opération est correcte [7].

7.4 Débordement en convention d'entiers relatifs : V

Pour les entiers, la soustraction est toujours réalisée par addition de l'opposé du deuxième opérande.

La valeur absolue de la somme de deux entiers relatifs de signes opposés est inférieure ou égale à la la plus grande des valeurs absolues des opérandes et le résultat est toujours représentable sur n bits. La somme de deux entiers relatifs de même signe peut ne pas être représentable sur n bits, auquel cas le résultat apparent sera faux :

— Sa valeur n'est égale à celle du vrai résultat de l'opération qu'à 2^n près.

— Son bit de signe (bit de poids fort) est également faux : la somme de deux entiers positifs donnera un résultat apparent négatif et la somme de deux entiers négatifs donnera un résultat apparent positif ou nul.

L'indicateur V (o**V**erflow [8]) est l'indicateur de débordement destiné à la convention d'interprétation pour entiers relatifs. $V = 1$ indique un débordement, auquel cas les deux dernières retenues sont de valeurs différentes.

		0	0	1	1	+3			0	1	1	0	+6			1	0	1	0	-6
$+_2$		1	0	1	1	-5	$+_2$		0	1	0	0	+4	$+_2$		1	1	0	0	-4
V=0	$0 = 0$	1	1	0			V=1	$0 \neq 1$	0	0	0			V=1	$1 \neq 0$	0	0	0		
		1	1	1	0	-2			1	0	1	0	-6			0	1	1	0	+6

Le signe du vrai résultat (sans erreur) de l'opération s'écrit : $V \oplus N = \overline{V}.N + V.\overline{N}$. Ainsi, le signe du résultat de l'opération sans erreur est N signe du résultat apparent s'il n'y a pas de débordement ($\overline{V}$), ou le signe opposé $\overline{N}$ de celui du résultat apparent en cas de débordement (V).

7.5 Expressions des conditions avec les indicateurs ZNCV

Après synthèse des indicateurs lors du calcul de $x - y$, il est possible de tester diverses conditions.

Par exemple ,l'expression de la condition "strictement inférieur" ($x < y$) est :

— $\overline{C}$ si x et y sont considérés comme des entiers naturels (la soustraction est impossible)

— $V \oplus N$ si x et y sont considérés comme des entiers relatifs (le vrai résultat est négatif).

8 Table des puissances de 2, cercle des entiers codés sur 4 bits

Le tableau suivant récapitule les principales puissances de 2 utiles, avec leur représentation en hexadécimal et les puissances de 10 approchées correspondantes.

7. Attention : les instructions de soustraction ou de comparaison de certains processeurs (dont le SPARC) stockent dans C le **complément** de la retenue finale. Pour ces processeurs, $C = 1$ indique toujours une erreur, que ce soit après une addition ou une soustraction.

8. L'initiale O n'a pas été retenue pour éviter une confusion avec zéro

n				2^n		
décimal	hexa	octal	binaire	décimal	hexa	commentaire
0	0	00	0000	1	1	
1	1	01	0001	2	2	
2	2	02	0010	4	4	
3	3	03	0011	8	8	
4	4	04	0100	16	10	un quartet = un chiffre hexa
5	5	05	0101	32	20	
6	6	06	0110	64	40	
7	7	07	0111	128	80	
8	8	10	1000	256	100	un octet = deux chiffres hexa
9	9	11	1001	512	200	
10	A	12	1010	1024	400	$1K_b$
11	B	13	1011	2048	800	$2K_b$
12	C	14	1100	4096	1000	$4K_b$
13	D	15	1101	8192	2000	$8K_b$
14	E	16	1110	16384	4000	$16K_b$
15	F	17	1111	32768	8000	$32K_b$
16	10	20	10000	65536	10000	$64K_b$
20	14	24	10100	1048576	100000	$1M_b = 1K_b^2 = 5$ chiffres
30	1E	36	11110	$\tilde{}1.07 \times 10^9$	40000000	$1G_b = 1K_b^3$