Chapitre 2

Série d'exercices n°2 - Théorie chapitre II

2.1 Rappels mathématiques

— Notation binaire : on exprime un nombre entier comme une somme de puissances de 2. La présence d'une puissance est encodée par un 1, son absence par un 0. On écrit par exemple "0b10101" pour encoder $2^4 + 2^2 + 2^0 = 21$.

— Notation hexadécimale : on exprime un nombre entier comme une somme de puissances de 16. Contrairement à l'encodage binaire, les puissances de 16 peuvent être multipliées par un chiffre qui n'est pas limité à 1, mais peut aller jusqu'à 15. On note ces chiffres comme {0,1,...,9,A,B,C,D,E,F}. On écrit par exemple "0x1C" pour encoder $1 * 16^1 + 12 * 16^0 = 28$. On peut retrouver des représentations binaires en transformant "1" en "0001" et "C" en "1100", tel que 0b11100 = $2^4 + 2^3 + 2^2 = 28$.

— Valeurs accessibles selon l'encodage : non-signé $[0, 2^n - 1]$, signé et complément à un $[-2^{n-1} + 1, 2^{n-1} - 1]$, complément à deux $[-2^{n-1}, 2^{n-1} - 1]$.

— Encodage du zéro : non-signé 0b 0000 0000, signé 0b 1000 0000 ou 0b 0000 0000, complément à un 0b 0000 0000 ou 0b 1111 1111, complément à deux 0b 0000 0000.

— Il existe plusieurs propriétés qui permettent de vérifier partiellement mais rapidement un encodage binaire : on peut vérifier la parité avec le dernier bit ; on peut vérifier si un nombre est divisible par 2^n s'il y a n zéros ; un décalage vers la gauche, resp. la droite, correspond à une multiplication, resp. une division, par 2.

— Il existe une technique qui permet de lire rapidement un nombre représenté en en complément à deux : on peut lire le nombre de la droite vers la gauche, s'arrêter au premier 1, et complémenter le reste du nombre à gauche de ce premier 1.

2.2 Mode d'emploi

- **Identifier l'encodage :**
  Il faut déterminer avec certitude sous quel format on encode les données ; chaque format a des règles spécifiques qu'il faut appliquer dans un ordre bien défini pour obtenir la re-

présentation voulue.

- **Déterminer la faisabilité de l'opération :**
  Selon l'opération à effectuer, est-ce que le résultat d'une opération sur n bits peut-il lui-même être représenté sur un bloc de n bits ? Est-ce qu'on s'attend à un dépassement si par exemple le résultat d'une opération sur 8 bits dépasse 255, et est-ce qu'on est capable de le diagnostiquer correctement quand on le lit comme le résultat d'un calcul numérique ?

- **Encoder les nombres dans le format identifié :**
  Si ce n'est pas déjà, on encode les nombres impliqués dans le calcul dans le format voulu. Pour ce faire, on applique méthodiquement les règles de chaque convention d'encodage.

- **Réaliser l'opération selon les règles habituelles de calcul écrit :**
  On applique les règles de calcul écrit comme en base 10, qui sont plus simples à employer en base 2, puisqu'on ne manipule que des 1 et des 0. Les additions dont le résultat dépassent 1 causent un report à la puissance de 2 suivante (vers la gauche), et les soustractions font de même quand le résultat dépasse 0, comme en base 10 quand une addition sur un chiffre dépasse 9, ou une soustraction dépasse 0.

- **Décoder le résultat et critiquer :**
  On applique les règles d'encodage pour retrouver un nombre en base 10 qui nous est familier, et on vérifie le résultat de l'opération. Si le résultat est incorrect, que s'est-il passé ? Une erreur d'encodage ? Une mauvaise opération de calcul écrit ? Un dépassement qu'on avait pas prévu ?

2.3 Correctifs des exercices

2.3.1 Exercice supplémentaire 1

Énoncé : Effectuer l'opération $26 - 37$ dans les représentations binaires par (1) valeur signée, par (2) complément à un et par (3) complément à deux (à chaque fois sur 8 bits).

Solution :

(1) On demande tout d'abord de réaliser une soustraction en représentation binaire par valeur signée. Si la représentation en valeur signée est relativement simple, la réalisation d'une soustraction n'est pas immédiate. En effet, puisque les nombres positifs comme négatifs sont encodés de la même manière au bit de signe près, il faut tout d'abord comparer les deux opérandes en valeur absolue, et déterminer laquelle est la plus grande. Ceci fait, on peut alors soustraire la valeur absolue de la plus petite de la valeur absolue de la plus grande, et y adjoindre le bit de signe approprié selon si la plus grande opérande en valeur absolue était l'opérande de gauche ou de droite.

On a un support de 8 bits, et on soustrait des quantités de signe opposés strictement inférieures à la capacité maximale du support en valeur signée qui est de $2^7 - 1 = 127$ en valeur absolue, puisqu'on dédie 7 bits à la valeur absolue du nombre et le dernier au signe. L'opération est donc entièrement réalisable sur ce format et son résultat pourra être représenté sur 8 bits en valeur signée.

Pour encoder des nombres en représentation binaire par valeur signée sur 8 bits, on procède comme suit :

— On fixe le bit de signe à 0 pour les nombres positifs, et à 1 pour les nombres négatifs. Une soustraction n'est jamais qu'une addition d'un nombre positif et d'un nombre négatif ; dans le cas d'une soustraction en valeur signée, cela n'a pas d'importance, mais dans le futur on considérera systématiquement toutes les opérations d'additions ou de soustractions comme des additions avec des opérandes positives ou négatives.
— On encode la valeur absolue du nombre sur les 7 bits restants tel que chaque bit indique la présence d'une puissance n de 2^n dans ce nombre. Pour construire cette représentation, on peut soustraire successivement de gauche à droite les termes 2^n du nombre à encoder ; si le résultat de la soustraction est positif, alors le bit associé à ce terme doit être fixé à 1 et on continue d'opérer avec le reste de la soustraction. Si le résultat est négatif, alors le bit associé à ce terme doit être fixé à 0 et on reprend la quantité avant la soustraction, puis on réessaie avec les bits suivants.

Pour encoder le nombre 26, on a donc un bit de signe fixé à 0 puisque ce nombre est positif, puis on essaie de soustraire les puissances 2^n pour $n = 6, ..., 0$. On trouve que $26 - 2^6$ et $26 - 2^5$ donnent un résultat négatif ; ces termes ne font donc pas partie du nombre, et on fixe donc les bits qui leur sont associés à 0, i.e. les 7ème et 6ème bits en partant de la droite, qu'on note habituellement bits 6 et 5 puisqu'on indique les bits en comptant à partir de 0 de la droite. On a

$26 - 2^4 = 10$; le terme 2^4 fait donc partie de la représentation binaire du nombre. On fixe le bit qui lui est associé, le 5ème en partant de la droite ou le bit d'indice 4, à 1, et on continue à soustraire du reste égal à 10. On trouve que $10 - 2^3 = 2$, et donc le 4ème bit ou bit 3 est également fixé à 1. Enfin, $2 - 2^2$ ne donne pas un résultat positif, mais c'est le cas pour $2 - 2^1$. Plus précisément, le résultat de cette dernière opération est zéro. Le bit 0 restant est fixé à 0 puisqu'il n'y a plus rien à représenter, et on a donc complètement défini la représentation binaire de ce nombre qui est la suivante :

$\boxed{0001\ 1010}$ *(valeur signée)*

Pour encoder le nombre -37 (puisqu'une soustraction correspond à une addition d'un nombre positif et d'un nombre négatif), on procède de la même manière. Le bit de signe est cette fois fixé à 1, puisqu'on a un nombre négatif. La représentation de la valeur absolue est obtenue en constatant qu'on a des résultats positifs pour les soustractions successives des termes 2^5, 2^2 et 2^0 de |-37| = 37. Dès lors, on fixe à 1 les bits 5, 2 et 0 et fixe les autres à 0. On a la représentation suivante :

$\boxed{1010\ 0101}$ *(valeur signée)*

Comme développé plus haut, la soustraction en représentation par valeur signée est une opération un peu laborieuse. On compare premièrement les opérandes, et on constate que l'opérande de droite, -37, est celle de plus grande valeur absolue. Le résultat de l'opération sera donc nécessairement un négatif, et on sait dès lors que son bit de signe sera fixé à 1. Pour les 7 autres bits du format, on effectue la soustraction des valeurs absolues en prenant soin de bien soustraire la plus petite valeur à la plus grande valeur, comme suit :

		-1	-1		-1			
	0	1	0	0	1	0	1	$\lvert -37\rvert$
−	0	0	1	1	0	1	0	$\lvert 26\rvert$
	0	0	0	1	0	1	1	$\lvert -37\rvert - \lvert 26\rvert$

On y adjoint ensuite le bit de signe égal à 1 dont on a déterminé la valeur avant d'effectuer la soustraction des valeur absolues. Au total, on a donc la représentation suivante :

$\boxed{1000\ 1011}$ *(valeur signée)*

Pour désencoder ce nombre, on applique les opérations d'encodage à l'envers : on lit un bit de signe égal à 1, ce qui indique donc un nombre négatif. On lit ensuite une somme de termes $2^3 + 2^1 + 2^0 = 11$ puisque les bits 3, 1 et 0 sont fixés à 1. On a donc au total -11, ce qui correspond bien au résultat de $26 - 37$.

(2) La soustraction est considérablement plus simple en représentation par complément à un, grâce aux propriétés de ce format d'encodage. En effet, on applique ici jusqu'au bout le principe qu'une soustraction n'est qu'une addition d'un nombre positif et d'un nombre négatif, puisqu'on effectue réellement une addition quel que soit le signe des opérandes. Là où la représentation en valeur signée demande donc d'implémenter dans un noyau de calcul les opérations d'addition, de comparaison de valeur absolue et de soustraction, la représentation en complétement à un ne demande elle plus que d'implémenter une opération d'addition.

Pour encoder les opérandes en complément à un, on doit opérer différement selon si le nombre est positif ou négatif. S'il est positif, on l'encode exactement comme en valeur signée, par son bit de signe et sa valeur absolue. S'il est négatif cependant, on doit d'abord encoder la valeur absolue du nombre, puis ensuite complémenter chacun des bits de cette représentation, ce qui revient à changer tous les 1 en 0 et vice-versa. Sur un support à 8 bits, on peut également considérer qu'on encode la valeur absolue du nombre négatif sur 7 bits, qu'on la complémente, puis qu'on y adjoint le bit de signe fixé à 1 pour indiquer un nombre négatif. Cependant, il est plus simple de partir de la représentation de la valeur absolue sur 8 bits et de complémenter chacun de ses bits ; le bit de signe sera naturellement fixé à 1 lors de l'opération de complément. Dans les deux approches, le résultat sera strictement identique.

Pour représenter l'opérande -37, on a donc les étapes successives suivantes sur 8 bits :

représentation de |-37| = 37 :	00100101
on complémente chacun des 8 bits :	11011010

Tel qu'on a donc la représentation suivante de -37 en complément à un :

$$\boxed{1101\ 1010}\ \textit{(complément à un)}$$

Pour effectuer la soustraction, on additionne le nombre positif et le nombre négatif comme suit :

$$\begin{array}{rcccccccc|c} & & & \boxed{1} & \boxed{1} & & \boxed{1} & & & \\ & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 26 \\ + & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & -37 \\ \hline & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 26-37 \end{array}$$

Pour désencoder ce nombre, on applique les opérations d'encodage à l'envers : on lit un bit de signe égal à 1, ce qui indique donc un nombre négatif. On complémente donc chaque bit de cette représentation pour lire sa valeur absolue, tel qu'on a la représentation suivante :

$$\boxed{0000\ 1011}\ \textit{(valeur absolue)}$$

On lit une somme de termes $2^3 + 2^1 + 2^0 = 11$ puisque les bits 3, 1 et 0 sont fixés à 1. On a donc au total -11, ce qui correspond bien au résultat de $26 - 37$.

(3) Pour encoder les opérandes en complément à deux, on doit opérer exactement comme en complément à un, puis ajouter +1 aux représentations des nombres négatifs. Cette opération additionnelle permet de s'abstraire des quelques propriétés sous-optimales de la représentation en complément à un, comme le fait qu'il existe deux représentations du zéro et le besoin de faire attention aux dépassements arithmétiques.

On remarque qu'en complément à deux, soustraire +1 à la représentation positive d'un nombre négatif à inverser peut être fait en ajoutant +1 après l'inversion. De la même manière, pour désencoder un résultat négatif d'une opération, on peut soustraire +1 avant inversion ou ajouter +1 après inversion. On préferera toujours additionner par simplicité, et parce que cela correspond en pratique au principe de pouvoir implémenter toutes les opérations par des additions dans des architectures de calcul réelles.

Pour représenter l'opérande -37, on a donc les étapes successives suivantes sur 8 bits :

représentation de |-37| = 37 :	0010 0101
on complémente chacun des 8 bits :	1101 1010
on ajoute +1 :	11011011

Tel qu'on a donc la représentation suivante de -37 en complément à deux :

$$\boxed{1101\ 1011}\ \textit{(complément à deux)}$$

Pour effectuer la soustraction, on additionne le nombre positif et le nombre négatif comme suit :

$$\begin{array}{cccccccccc} & & & \boxed{1} & \boxed{1} & & \boxed{1} & & & \\ & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 26 \\ + & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & -37 \\ \hline & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 26-37 \end{array}$$

Pour désencoder ce nombre, on applique les opérations d'encodage à l'envers : on lit un bit de signe égal à 1, ce qui indique donc un nombre négatif. On complémente donc chaque bit de cette représentation puis on ajoute +1 pour lire sa valeur absolue, tel qu'on a successivement :

représentation de 26-37 :	1111 0101
on complémente chacun des 8 bits :	0000 1010
on ajoute +1 :	0000 1011

Tel qu'on a donc la représentation suivante de |26 - 37| :

$$\boxed{0000\ 1011}\ \textit{(valeur absolue)}$$

On lit bien toujours la même valeur absolue de 11 qui indique avec le signe $26 - 37 = -11$.

2.3.2 Exercice supplémentaire 2

Énoncé : Calculer le produit -17×-23 à partir des représentations binaires par complément à deux de ces nombres.

Solution :

Comme pour l'addition et la soustraction, on effectue une multiplication écrite comme en base 10, mais avec les représentations binaires. On présente le multiplicande et le multiplicateur, et on calcule autant de produits partiels qu'il y a de chiffres dans le multiplicateur, ces produits partiels étant chacun décalés d'un chiffre à chaque fois qu'on progresse de droite à gauche dans le multiplicateur. La base 2 simplifie considérablement les calculs, puisqu'on ne manipule que des 1 et des 0 ; entre autres, les produits partiels ne consistent qu'en des contributions entièrement nulles ou des décalages de la représentation exacte du multiplicateur, selon si le chiffre du multiplicande associé à ce produit partiel est un 0 ou un 1.

La complexité particulière des multiplications concerne le nombre de bits nécessaire pour représenter le résultat de l'opération. En toute généralité, on a besoin de $2 \cdot n$ bits pour représenter le produit de deux opérandes de n bits. Cependant, les architectures de calcul réelles sont strictement limitées à leur support physique ; une architecture de calcul implémentée en 32 bits représentera les nombres sur 32 bits, et calculera des produits sur 32 bits. Si, par chance, les deux opérandes ont besoin de k et l bits pour être entièrement représentés, tels que $k + l < n$, alors le résultat de la multiplication pourra lui-même être correctement représenté sur n bits. Si ce n'est pas le cas, alors le résultat de l'opération sur n bits sera tronqué en une fraction de la réponse réelle qui peut même voir son signe s'inverser.

Dès lors, il faut déterminer avec certitude le support de l'opération de multiplication. Si ce support n'est pas explicitement spécifié, comme dans le cas de l'énoncé de cet exercice, il faut faire un choix qui peut être motivé de différents manières :

— On peut choisir un support classique au moins suffisant pour représenter les opérandes. Par exemple, un format à 8 bits est suffisant pour représenter -17 et -23, mais pas leur produit. Si on décrète qu'on fait l'hypothèse d'une unité de calcul sur 8 bits, on représente ces nombres sur 8 bits et on obtient un résultat de multiplication erronné en connaissance de cause ; c'est un cas qui se conforme à la réalité et qui est donc parfaitement valable.
— On peut choisir un support classique au moins suffisant pour représenter le résultat de l'opération. Pour cela, il faut au préalable déterminer que le résultat de -17 x -23 demande un support plus grand que 8 bits, mais que 16 bits est un support suffisant. Dans ce cas, on exprime les représentations de -17 et de -23 sur 16 bits et on obtiendra un résultat correct et lisible sur 16 bits.
— On pourrait techniquement choisir un support d'une longueur exactement égale à $k + l$ où k et l sont les longueurs de format les plus petites strictement nécessaires pour représenter les opérandes. Dans ce cas, on exprime les deux opérandes sur $k + l$ bits et on obtiendra un résultat correct et lisible sur $k + l$ bits. Si cette approche est théoriquement valable, elle est en pratique tout à fait irréaliste, puisqu'elle considère que les supports physiques de calcul des noyaux informatiques sont de longueur variables et non-conforme à des

standards comme les habituels 8 bits / 16 bits / 23 bits / 64 bits.

Enfin, il ne faut pas faire l'erreur de penser que l'on peut représenter les opérandes sur n bits et "étendre" a posteriori le calcul sur $2 \cdot n$ bits. Dès que l'on va au-delà des représentations non-signées, la présence du bit de signe puis des représentations inverses des nombres négatifs fait que cette approche est strictement vouée à l'échec.

On va effectuer le calcul demandé sur différents supports afin d'illustrer les explications développées ci-dessus.

On tente d'abord d'effectuer le calcul sur un support classique de 8 bits. On commence par représenter les opérandes sur 8 bits en suivant les règles du complément à deux, comme suit :

représentation de |-17| = 17 :	0001 0001
on complémente chacun des 8 bits :	1110 1110
on ajoute +1 :	1110 1111

représentation de |-23| = 23 :	0001 0111
on complémente chacun des 8 bits :	1110 1000
on ajoute +1 :	1110 1001

On effectue ensuite la multiplication sur 8 bits, en omettant les produits partiels entièrement nuls et les 0 introduits par décalage qui n'auront aucune influence sur le résultat :

$$
\begin{array}{cccccccccc}
 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\
\times & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
\hline
\boxed{1} & \boxed{1} & & \boxed{1} & \boxed{1} & & & & & \\
 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\
 & 0 & 1 & 1 & 1 & 1 & & & \\
 & 1 & 1 & 1 & & & & & \\
 & 1 & 1 & & & & & & \\
 & 1 & & & & & & & \\
\hline
 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1
\end{array}
$$

On désencode ce résultat en appliquant les opérations d'encodage en complément à deux à l'envers, comme suit :

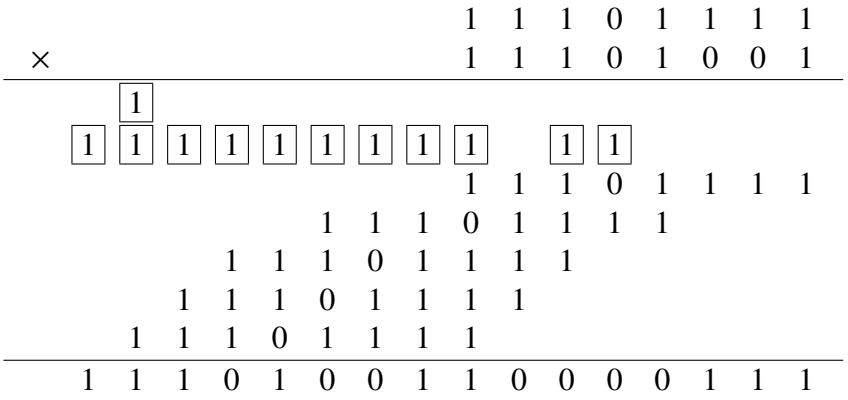
représentation de -17×-23 sur 8 bits :	1000 0111
on complémente chacun des 8 bits :	0111 1000
on ajoute +1 :	0111 1001

On lit ce résultat comme -121, qui n'est en effet pas le résultat du produit -17×-23 dans notre système de calcul habituel non-limité en taille de représentation. Cependant, c'est bien le résultat valide de cette opération dans une architecture de calcul sur 8 bits.

Il est intéressant de remarquer que la somme de produits partiels peut mener à plus d'un

report par chiffre. Dans ce cas, on peut procéder de plusieurs manières : on peut noter des reports de plus de 1, ou on peut redistribuer les reports supérieurs à 1 sur les chiffres plus à gauche. En effet, un report de 2 depuis un bit n vers un bit $n+1$ correspond en réalité à un report de 1 au bit $n+2$. De la même manière, on peut décomposer un report d'un bit n de par exemple 5 = 0b101 en un report de 1 au bit $n+1$ et un second report de 1 au bit $n+3$. C'est l'approche qu'on préfèrera ici, puisqu'elle permet d'opérer entièrement en 0 et en 1 sans sortir de la logique en base 2, mais cela ne concerne que la représentation du calcul pour l'humain et n'a donc pas une importance capitale.

Si on essaie de calculer "a posteriori" le résultat sur 16 bits d'une multiplication de deux nombres représentés sur 8 bits, on aura la situation suivante :



Après désencodage, on lit ce résultat comme -9849, qui n'est pas non plus le résultat du produit -17×-23. **Ce n'est pas non plus le résultat valide d'une opération qui a du sens**, puisqu'on a multiplié des nombres représentés sur 8 bits et qu'on lit le résultat sur 16 bits. **Cette approche est donc strictement à proscrire.**

Enfin, si on choisit d'effectuer l'opération correctement sur 16 bits, on commence par encoder les opérandes sur 16 bits comme suit :

représentation de |-17| = 17 :	0000 0000 0001 0001
on complémente chacun des 16 bits :	1111 1111 1110 1110
on ajoute +1 :	1111 1111 1110 1111

représentation de |-23| = 23 :	0000 0000 0001 0111
on complémente chacun des 16 bits :	1111 1111 1110 1000
on ajoute +1 :	1111 1111 1110 1001

Il est intéressant de remarquer que dans le cas des négatifs, l'extension de la représentation sur 8 bits vers une représentation sur 16 bits correspond à ajouter des 1. On peut en réalité généraliser cette propriété comme suit : pour étendre une représentation d'un nombre vers la gauche en complément à deux, il suffit de propager le bit de signe vers la gauche.

On effectue ensuite la multiplication écrite sur 16 bits comme suit :

	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
×	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	1
	1		1				1									
	1	1	1	1		1	1	1	1		1	1				
	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
	1	1	1	1	1	1	1	1	0	1	1	1	1			
	1	1	1	1	1	1	0	1	1	1	1					
	1	1	1	1	1	0	1	1	1	1						
	1	1	1	1	0	1	1	1	1							
	1	1	1	0	1	1	1	1								
	1	1	0	1	1	1	1									
	1	0	1	1	1	1										
	0	1	1	1	1											
	1	1	1	1												
	1	1	1													
	1	1														
	1															
	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1

Comme ce nombre est positif, on peut le lire immédiatement comme 391. Cette fois-ci, on a bien le résultat de -17×-23. On remarque également que les 8 bits de droite de cette représentation correspondent au résultat de l'opération sur 8 bits, et que c'est donc parfaitement logique qu'on obtienne ce résultat sur 8 bits.

2.3.3 Exercice supplémentaire 3 (première partie)

Énoncé : Quel est le nombre représenté par la suite de bits

10000000000000000000000000000001

(il y a 30 chiffres 0) dans les représentations :

(a) non signée ?

(b) signée ?

(c) complément à un ?

(d) complément à deux ?

Solution :

Pour résoudre cet exercice, on doit simplement appliquer méthodiquement les règles des différentes conventions d'encodage à l'envers.

(a) Pour interpréter cette représentation comme une valeur non-signée, on associe à chaque bit une puissance n d'un terme 2^n où n = {0,...,k} où k est la taille du format. Puisque le support comporte 32 bits, le bit de poids faible correspond au terme 2^0 et le bit de poids fort au terme 2^{31}. On a donc immédiatement la valeur $2^{31} + 2^0 = 2^{31} + 1$.

(b) Pour interpréter cette représentation comme une valeur signée, on opère comme pour une valeur non-signée à ceci près que le bit le plus à gauche, originellement bit de poids fort, devient un bit de signe indiquant que le nombre est négatif s'il est à 1. Le bit de poids fort devient le second en partant de la gauche et encode le nouveau plus grand terme 2^{30}. On a maintenant plus qu'un bit de signe et que le bit de poids faible, tel qu'on lit la valeur $(-1) \cdot 2^0 = -1$.

(c) Pour interpreter cette représentation comme une représentation en complément à un, on opère différement si le nombre est négatif. Comme le bit de signe à l'extrême gauche du format est à 1, on a bien un négatif. Dès lors, pour décoder la représentation, on doit complémenter chacun de ses bits puis lire le résultat comme une valeur absolue. On a successivement :

représentation en complément à un :	10...01
on complémente chacun des bits :	01...10

La valeur absolue du nombre encodé correspond donc à la somme des termes 2^n pour n allant de 30 (le bit de poids fort, le second en partant de la gauche) à 1 (le dernier bit à 1, le second en partant de la droite). Plutôt que de calculer explicitement cette somme de 30 termes individuels, on préferera remarquer que cette représentation correspond à la capacité entière d'un format de 31 bits (représenté par 31 bits à 1) auquel on retiré le premier terme associé au bit de poids faible. On a donc que la valeur absolue à lire est égale à $(2^{31} - 1) - 2^0 = 2^{31} - 2$. La valeur complète, avec son signe, est donc $(-1) \cdot (2^{31} - 2) = 2 - 2^{31}$.

(d) Pour interpreter cette représentation comme une représentation en complément à deux, on

opère exactement comme pour le complément à un, puis on ajoute +1 au résultat de l'opération de complément. On a donc :

représentation en complément à un :	10...01
on complémente chacun des bits :	01...10
on ajoute +1 :	01...11

Cette fois-ci, on a tous les bits, du 31 au 0. Dès lors, comme développé ci-dessus, on peut immédiatement calculer cette valeur absolue comme $2^{31} - 1$; c'est la même valeur que pour le complément à un, à laquelle on ne retire cette fois-ci pas le bit de poids faible. La valeur complète, avec son signe, est donc $(-1) \cdot (2^{31} - 1) = 1 - 2^{31}$.