Représentation binaire et complément à deux en C

1 Représentation des entiers naturels

On représente une suite de bits par la structure suivante :

```c
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>

typedef struct {
    size_t size;
    unsigned char *bits;
} binary;
```

Le champ `size` donne le nombre de bits. Le champ `bits` pointe vers un tableau alloué dynamiquement, dont chaque élément contient la valeur numérique 0 ou 1.

On adopte la convention *big endian* : le bit de poids fort est placé à l'indice 0, et le bit de poids faible à l'indice `size - 1`.

Ainsi, l'entier 13, représenté sur huit bits, correspond au tableau

$$[0, 0, 0, 0, 1, 1, 0, 1], \qquad 13 = 2^3 + 2^2 + 2^0.$$

La valeur d'une représentation b de taille n est donc

$$\sum_{i=0}^{n-1} b.\text{bits}[i]\, 2^{n-1-i}.$$

Dans cette première partie, les entiers représentés sont positifs ou nuls. Toutes les tailles sont strictement positives. On suppose que les allocations réussissent et que les valeurs manipulées sont représentables par le type `int`.

Les tableaux alloués devront être libérés lorsqu'ils ne seront plus utilisés.

1.1 D'un entier à sa représentation binaire

Les divisions euclidiennes successives par 2 permettent d'obtenir les bits d'un entier, du poids faible vers le poids fort. Chaque reste donne un bit ; on poursuit avec le quotient.

Question 1.

Écrire la fonction suivante : `binary int_to_binary(int x, size_t size);`

Elle alloue et renvoie la représentation binaire de x sur `size` bits. On suppose que $0 \leq x < 2^{\texttt{size}}$. Utiliser uniquement des divisions et des restes pour extraire les bits.

Par exemple, la conversion de 13 sur huit bits doit produire le tableau $[0, 0, 0, 0, 1, 1, 0, 1]$.

1.2 D'une représentation binaire à un entier

Pour retrouver la valeur d'une représentation binaire, on utilise la méthode de Horner : on parcourt les bits de gauche à droite, en multipliant à chaque étape la valeur accumulée par 2, puis en ajoutant le bit courant.

Par exemple, pour la suite $[1, 1, 0, 1]$, les valeurs successives de l'accumulateur, initialement nul, sont

$$1, \quad 3, \quad 6, \quad 13.$$

Question 2.

Écrire la fonction suivante : `int binary_to_int(binary b);`

Elle renvoie la valeur de la représentation binaire `b` en utilisant la méthode de Horner. On suppose que cette valeur est représentable par le type `int`.

Tester les deux fonctions avec les entiers 0, 1, 13 et 255, représentés sur huit bits. Pour chaque entier, vérifier que la conversion en binaire suivie de la conversion en entier restitue la valeur initiale.

2 Addition binaire

On souhaite additionner deux représentations binaires de même taille N, directement à partir de leurs bits.

Comme pour une addition posée en base dix, on commence par les chiffres de poids faible, puis on progresse vers les poids forts en propageant une retenue.

Pour une colonne donnée, on additionne les deux bits et la retenue entrante. La somme obtenue est comprise entre 0 et 3. Son reste dans la division par 2 donne le bit du résultat ; son quotient donne la retenue sortante.

Le résultat est représenté sur N bits : une éventuelle retenue sortant de la colonne de poids fort est abandonnée. On calcule donc la somme *modulo* 2^N.

Question 3.

Écrire la fonction suivante : `binary binary_add(binary a, binary b);`

On suppose que les deux représentations ont la même taille. La fonction alloue et renvoie une nouvelle représentation contenant leur somme sur cette même taille.

Elle ne doit modifier aucun des deux tableaux reçus. Il est interdit de convertir les représentations en entiers pour effectuer l'addition. On peut utiliser des variables entières pour les indices, la retenue et le calcul d'une colonne.

Tester notamment, sur huit bits, les additions $13+6$, $127+1$ et $255+1$. Vérifier également que les tableaux des deux opérandes sont inchangés.

3 Représentation en complément à deux

Sur N bits, la représentation en complément à deux permet de coder les entiers de l'intervalle

$$[-2^{N-1},\, 2^{N-1}-1].$$

Un entier positif ou nul conserve sa représentation binaire habituelle. Un entier strictement négatif x est représenté par l'écriture binaire sur N bits de l'entier naturel $x+2^N$.

Par exemple, sur huit bits, -13 est représenté par l'écriture binaire de

$$-13+2^8 = 243,$$

soit

$$[1,1,1,1,0,0,1,1].$$

Le premier bit permet de distinguer les deux cas : il vaut 0 pour un entier positif ou nul, et 1 pour un entier négatif.

3.1 Calculer une puissance de deux en C

L'opérateur `<<` effectue un décalage des bits vers la gauche. Les positions libérées à droite sont remplies par des zéros.

Pour les entiers positifs ou nuls considérés ici, décaler d'une position vers la gauche revient à multiplier par 2, à condition que le résultat reste représentable.

En particulier :

```
1 << 0   // 1
1 << 1   // 2
1 << 3   // 8
1 << 8   // 256
```

L'expression `1 << n` permet donc de calculer 2^n.

Dans toute la suite, on suppose que N est strictement positif et que 2^N est représentable par le type `int`. Cette hypothèse garantit la validité des décalages utilisés et permet de stocker les valeurs intermédiaires des conversions.

Avec un type `int` de 32 bits, on se limitera ainsi à $N \leq 30$.

Attention : en C, l'opérateur `^` ne calcule pas une puissance. Pour écrire $x+2^N$, on utilisera `x + (1 << N)`.

3.2 D'un entier à sa représentation en complément à deux

Question 4.

Écrire la fonction suivante : `binary int_to_ca(int x, size_t size);`

Elle alloue et renvoie la représentation en complément à deux de x sur `size` bits. On suppose que

$$-2^{\texttt{size}-1} \leq x \leq 2^{\texttt{size}-1}-1.$$

Appliquer la méthode décrite précédemment : si x est négatif, lui ajouter $2^{\texttt{size}}$, puis utiliser la fonction `int_to_binary`.

Dans cette fonction, les seules additions ou soustractions autorisées sur la valeur à convertir sont des additions ou soustractions de puissances de deux.

3.3 D'une représentation en complément à deux à un entier

Soit u la valeur obtenue en interprétant une suite de N bits comme la représentation d'un entier naturel.
La valeur de cette même suite en complément à deux est

$$\begin{cases} u & \text{si le premier bit vaut 0,} \\ u - 2^N & \text{si le premier bit vaut 1.} \end{cases}$$

Question 5.

Écrire la fonction suivante : `int ca_to_int(binary b);`
Elle renvoie l'entier représenté en complément à deux par `b`. Utiliser la fonction `binary_to_int`, puis effectuer, si nécessaire, une soustraction d'une puissance de deux.

Tester les conversions sur huit bits avec -128, -13, -1, 0, 13 et 127. Vérifier que la conversion en complément à deux suivie de la conversion en entier restitue chaque valeur initiale.
Tester ensuite l'addition binaire des représentations de -13 et 6, puis convertir le résultat avec `ca_to_int`.

4 Opérations en complément à deux

Devoir maison

Toutes les questions qui suivent sont à déposer sur Cahier de Prepa avant le dimanche 22 novembre 23h45 dans une archive contenant uniquement le fichier `ca.c` dont le nom respecte le format `nom_prenom.zip`.

Le corrigé des fonctions de la section précédente sera mis en ligne sur `nussbaumcge` ainsi qu'un `Makefile`, un fichier d'en-tête et un fichier de tests.

Présentation

Toutes les fonctions de cette section travaillent directement sur les représentations binaires. Il est interdit d'utiliser les fonctions de conversion vers ou depuis le type `int` pour effectuer les opérations.
Chaque fonction renvoie une représentation dont le tableau est nouvellement alloué, sans modifier les tableaux reçus. Le résultat conserve le même nombre de bits que les opérandes.

4.1 Addition

Question 6.

Écrire la fonction suivante : `binary ca_add(binary a, binary b);`
Les deux opérandes représentent des entiers x et y en complément à deux sur le même nombre N de bits. On suppose que leur somme est représentable sur ce format :

$$-2^{N-1} \leq x + y \leq 2^{N-1} - 1.$$

Réutiliser la fonction `binary_add` et justifier que le résultat représente bien $x + y$.

Tester sur huit bits les additions $13 + 6$, $(-13) + 6$, $13 + (-6)$ et $(-13) + (-6)$.

4.2 Opposé

Le plus petit entier du format est -2^{N-1}. Son opposé, 2^{N-1}, n'est pas représentable en complément à deux sur N bits.

Question 7.

Écrire la fonction suivante : `binary opposite(binary b);`
La représentation `b` code un entier x en complément à deux sur N bits. On suppose que $x \neq -2^{N-1}$.
La fonction doit renvoyer la représentation de $-x$ sur le même format. Justifier la méthode utilisée.

Tester sur huit bits avec 0, 1, -1, 12, -12, 127 et -127. Vérifier également qu'appliquer deux fois la fonction restitue les bits de la représentation initiale.

4.3 Valeur absolue

Question 8.

Écrire la fonction suivante : `binary ca_abs(binary b);`
Elle renvoie la représentation en complément à deux de la valeur absolue de l'entier représenté par `b`, sur le même format.
On suppose que cet entier n'est pas le plus petit du format. Réutiliser la fonction `opposite` lorsque cela est utile.
Même lorsque l'entier est positif ou nul, le tableau du résultat doit être nouvellement alloué.

Tester sur huit bits avec 0, 13, -13, 127 et -127.

4.4 Soustraction

Question 9.

Écrire la fonction suivante : `binary ca_sub(binary a, binary b);`
Les deux opérandes représentent des entiers x et y en complément à deux sur le même nombre N de bits. La fonction renvoie la représentation de $x - y$ sur ce format.
On suppose que $y \neq -2^{N-1}$ et que la différence est représentable :

$$-2^{N-1} \leq x - y \leq 2^{N-1} - 1.$$

Utiliser les fonctions précédentes. Libérer tout tableau temporaire devenu inutile.

Tester sur huit bits les soustractions $13-6$, $6-13$, $(-13)-6$, $13-(-6)$ et $(-13)-(-6)$. Pour chaque test, vérifier que les opérandes sont inchangés.

5 Extension de format et traitement du débordement

On souhaite désormais calculer la somme exacte de deux entiers représentés en complément à deux sur N bits, même lorsqu'elle n'est pas représentable sur le format initial.
Les fonctions de cette section ne doivent pas modifier les tableaux reçus ni utiliser les fonctions de conversion vers ou depuis le type `int`. Chaque résultat possède un tableau nouvellement alloué. Libérer les tableaux temporaires devenus inutiles.

5.1 Extension de format

Pour représenter un entier sur un bit supplémentaire en conservant sa valeur, on duplique son bit de signe à gauche de sa représentation.
Par exemple, sur quatre puis cinq bits :

$$5 : \quad 0101 \longrightarrow 00101, \qquad -5 : \quad 1011 \longrightarrow 11011.$$

Question 10.

Écrire la fonction suivante : `binary ca_extend(binary b);`
Elle renvoie une représentation du même entier en complément à deux, sur un bit supplémentaire. Justifier que cette extension conserve la valeur représentée.

5.2 Détection du débordement

On additionne deux représentations sur N bits en conservant uniquement les N bits du résultat.
Il y a *débordement signé* lorsque la somme exacte n'appartient pas à l'intervalle

$$[-2^{N-1},\, 2^{N-1} - 1].$$

Question 11.

Justifier les propriétés suivantes :
— si les opérandes ont des signes différents, il n'y a pas de débordement signé ;
— si les opérandes ont le même signe, il y a débordement signé si et seulement si le résultat obtenu sur N bits a le signe opposé.
En déduire la fonction suivante : `int ca_overflow(binary a, binary b, binary sum);`
Les trois représentations ont la même taille et `sum` contient le résultat de `binary_add(a, b)`. La fonction renvoie 1 en cas de débordement signé, et 0 sinon.

5.3 Addition avec extension si nécessaire

Question 12.

Montrer qu'un bit supplémentaire suffit toujours pour représenter la somme exacte de deux entiers codés en complément à deux sur N bits.

Question 13.

Écrire la fonction suivante : `binary ca_add_exact(binary a, binary b);`
Les opérandes ont la même taille N. La fonction renvoie la représentation de leur somme exacte : sur N bits si cette taille suffit, et sur $N+1$ bits sinon.
Calculer d'abord la somme sur N bits et détecter un éventuel débordement signé. En cas de débordement, étendre les deux opérandes puis recommencer l'addition.

Tester avec des opérandes sur quatre bits :

$$3+2, \qquad 7+3, \qquad (-5)+(-6), \qquad (-8)+(-8), \qquad 7+(-8).$$

Vérifier les valeurs obtenues, la taille de chaque résultat et la conservation des opérandes.

5.4 Éviter une seconde addition

Question 14.

En cas de débordement signé, on souhaite conserver les N bits déjà calculés et les préfixer par un seul bit, sans recommencer l'addition.

Déterminer le bit à ajouter et justifier la méthode. Écrire une nouvelle version : `binary ca_add_exact_fast(binary a, binary b);`

Reprendre les tests précédents et vérifier que les deux versions de l'addition exacte produisent les mêmes représentations.

5.5 Réduction du format

Question 15.

Écrire la fonction suivante : `binary shrink(binary b);`

Elle renvoie une représentation de taille minimale du même entier en complément à deux.

Le tableau du résultat doit être nouvellement alloué et le tableau initial ne doit pas être modifié. Il est interdit d'utiliser les fonctions de conversion vers ou depuis le type `int`.

Par exemple :

$$\begin{aligned} [0,0,0,1,0,1] &\longrightarrow [0,1,0,1], \\ [1,1,1,0,1,1] &\longrightarrow [1,0,1,1], \\ [0,0,0,0] &\longrightarrow [0], \\ [1,1,1,1] &\longrightarrow [1]. \end{aligned}$$

Justifier que le résultat conserve la valeur initiale et que sa taille est minimale.

Tester avec 0, -1, 1, -2, 5, -5, 127 et -128, initialement représentés sur huit bits. Vérifier la conservation de la valeur et du tableau initial.