Fonctionnement des Ordinateurs

TP1 - Représentation des nombres entiers

B. QUOITIN
Faculté des Sciences
Université de Mons

Résumé

L'objectif de cette séance d'exercices est de renforcer votre compréhension des notations binaire, hexadécimale et octale et des conversions entre elles, des différents formats de représentation de nombres entiers dans un ordinateur – entiers non-signés et complément à deux et d'opérations arithmétiques élémentaires en représentation binaire.

Table des matières

1 Représentation positionnelle

1.1 Représentation binaire (non-signée)

Soit un mot binaire $(b_i)_{i\in\{0\dots N-1\}}$. Donnez la formule permettant d'interpréter (b_i) comme la représentation positionnelle binaire de x. Connaissez-vous les exposants entiers de 2 ? Il est utile de les mémoriser jusqu'à 2^8 voire 2^{16}.

Q1) Interprétation de la notation positionnelle binaire

$$x = \sum_{i=0}^{N-1} b_i 2^i$$

Quel est le plus grand nombre naturel représentable sur 12 bits avec cette représentation ?

Q2) Plus grand nombre représentable sur 12 bits

$2^N - 1 = 4095$

Convertissez les nombres suivants de binaire en décimal

Q3) `100000`

$2^5 = 32$

Q4) `000010`

$2^1 = 2$

Q5) `111111`

$2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 63$

Q6) `010101`

$2^4 + 2^2 + 2^1 = 21$

Convertissez les nombres suivants de décimal en binaire. Ne donnez que les bits significatifs. Appliquez au besoin l'algorithme utilisant des divisions euclidiennes successives.

Q7) 64

`1000000`

Q8) 7

`111`

Q9) 192

`11000000`

Q10) 3073

`110000000001`

Détail de l'exécution de l'algorithme :

n	q	r
3073	1536	1
1536	768	0
768	384	0
384	192	0
192	96	0
96	48	0
48	24	0
24	12	0
12	6	0
6	3	0
3	1	1
1	0	1

1.2 Bases hexadécimales et octales

Convertissez les nombres suivants d'hexadécimal/octal en décimal. Le préfixe `0x` indique la représentation hexadécimale tandis que le préfixe `0` indique la représentation octale, des conventions également utilisées dans certains langages de programmation.

Q11) **`0x1010`**

$1 \times 16^3 + 1 \times 16^1 = 4112$

Q12) **`0x10A0`**

$1 \times 16^3 + 10 \times 16^1 = 4256$

Q13) **`0xBABE`**

$11 \times 16^3 + 10 \times 16^2 + 11 \times 16^1 + 14 \times 16^0 = 45056 + 2560 + 176 + 14 = 47806$

Q14) **`020` (octal)**

$2 \times 8^1 = 16$

Q15) **`072` (octal)**

$7 \times 8^1 + 2 \times 8^0 = 58$

Convertissez les nombres suivants de la base 10 vers une autre base, en utilisant la méthode reposant sur les divisions entières successives.

Q16) 51966 **à convertir en hexadécimal**

`0xCAFE`

Détail de l'exécution de l'algorithme :

n	q	r
51966	3247	$14 \rightarrow$ `E`
3247	202	$15 \rightarrow$ `F`
202	12	$10 \rightarrow$ `A`
12	0	$12 \rightarrow$ `C`

Q17) 1025 **à convertir en octal**

`02001`

Détail de l'exécution de l'algorithme :

n	q	r
1025	128	`1`
128	16	`0`
16	2	`0`
2	0	`2`

Convertissez les nombres suivants entre les représentations binaire, octale et hexadécimale.

Q18) **`0xCAFE` en binaire**

`1100101011111110`

`C` $\rightarrow$ `1100`
`A` $\rightarrow$ `1010`
`F` $\rightarrow$ `1111`
`E` $\rightarrow$ `1110`

Q19) **`0xCAFE` en octal**

`0145376`

`1 100 101 011 111 110`
`110` $\rightarrow$ `6`
`111` $\rightarrow$ `7`
`011` $\rightarrow$ `3`
`101` $\rightarrow$ `5`
`100` $\rightarrow$ `4`
`1` $\rightarrow$ `1`

Q20) **`072` (octal) en binaire**

`111010`

`7` $\rightarrow$ `111`
`2` $\rightarrow$ `010`

1.3 Algorithme de conversion décimal → binaire

Ecrivez en langage C, Java ou Python, un programme qui demande à l'utilisateur un nombre naturel dans l'intervalle $[0, 2^{31} - 1]$, qui le convertit ensuite en binaire et affiche finalement à la console le résultat de la conversion. Pour cela, itérez du bit de poids le plus fort au bit 0 et testez si la valeur du bit correspondant de la variable vaut 0 ou 1 en utilisant l'opérateur logique *bitwise* (`&`).

Q21) Code de l'algorithme

```
import math

n = 67

s = ''
for i in reversed(range(0, math.floor(math.log2(n)) + 1):
    if n & (1 << i):
        s += '1'
    else:
        s += '0'
print(s)
```

2 Complément à 2

2.1 Représentation

Soit un mot binaire $(b_i)_{i\in\{0\dots N-1\}}$. Donnez la formule permettant d'interpréter (b_i) comme la représentation en complément à 2 de x.

Q22) Représentation en complément à 2

$$x = -b_{N-1}2^{N-1} + \sum_{i=0}^{N-2} b_i 2^i$$

Quel est l'intervalle de nombres entiers représentables en complément à 2 sur 12 bits ?

Q23) Intervalle d'entiers représentables sur 12 bits

$[2^{N-1}, 2^{N-1}-1] = [-2048, 2047]$

Donnez la représentation binaire en complément à 2 sur 8 bits des nombres suivants.

Q24) 17

`00010001`

Q25) -17

`11101111`

Q26) 255

Non représentable, car sur N bits les nombres représentables sont compris dans l'intervalle $\left[-2^{N-1}, 2^{N-1}-1\right]$, soit $[-128, 127]$ sur 8 bits.

Q27) -128

`10000000`

Q28) -73

`10110111`

2.2 Opérations arithmétiques

Donnez la représentation binaire en complément à 2 sur 8 bits des résultats des opérations suivantes. Afin de vérifier vos résultats fournissez également les valeurs des opérandes et du résultat en décimal. Dans le cas de la division, donnez le quotient et le reste.

Q29) **`01001111 + 00000001`**

```
     1111
  01001111
+ 00000001
  01010000
```

Q30) Opposé de `10010110`

Par définition, la représentation en complément à 2 de l'opposé d'un nombre x est la représentation de $2^N - x$. Il est facile d'obtenir $2^N - 1 - x$ en complémentant tous les bits de la représentation $(b_i)_i$ de x. Il suffit ensuite d'ajouter 1 à cette représentation.

Ici, $(b_i)_i$=`10010110` représente x. En complémentant chacun des bits, on obtient `01101001`. En ajoutant 1, on obtient la représentation de $-x$: `01101010`.

Vérification : `10010110` représente $x = -106$ alors que `01101010` représente $-x = 106$.

Q31) `01001111 + 01000001`

```
  1  1111     reports
  01001111
+ 01000001
  10010000
```

Il y a un dépassement de capacité qui peut être détecté en observant que les bits de poids fort des deux nombres additionnés vaut 0 alors que celui de leur somme vaut 1. Les deux nombres additionnés sont $x = 79$ et $y = 65$ et leur somme devrait être $x + y = 144$. Pourtant, le résultat obtenu est -112, ce qui correspond à $x + y - 2^N$.

Q32) `00000001 - 00000010`

```
-1111111      reports négatifs
  00000001
- 00000010
  11111111
```

Une autre façon d'obtenir ce résultat serait d'effectuer l'addition $x + (-y)$. La représentation de $-y$ est `11111110`.

```
  00000001
+ 11111110
  11111111
```

Q33) `00001001` $\times$ `00000110`

Il s'agit de calculer le produit de $x = 9$ et $y = 6$. La technique utilisée dans la résolution ci-dessous est celle de l'accumulation des produits partiels, tout en effectuant que des additions sur N bits.

```
          00001001
        × 00000110
  ------------------
          00000000   (accumulateur)
        + 00000110
  ------------------
          00000110   (somme 1er produit partiel)
        + 00000000
  ------------------
          00000011   (somme 2 premiers produits partiels)
        + 00000000
  ------------------
          00000001   (somme 3 premiers produits partiels)
        + 00000110
  ------------------
          00000110   (somme 4 premiers produits partiels)
              ...continuer 4 itérations sans effet...
  ==================
  0000000000110110
```

Le résultat est $x \times y = 54$.

Q34) `10000001 - 10000010`

`11111111` (-1)

Q35) **Opposé de `10000000`**

`10000000`

Il s'agit en fait d'un dépassement de capacité. L'opposé du nombre (-128) n'est pas représentable en complément à 2 sur $N = 8$ bits.

Q36) `00010100 / 00000101`

Itération	R	d	Q	
init	00010100	101000000000	0	$R \leftarrow$ dividende ; $d \leftarrow$ diviseur $\ll N$; $Q \leftarrow 0$
1	00010100	1010000000	0	$d \leftarrow d \ll 1$ $R - d < 0$
2	00010100	101000000	0	
3	00010100	10100000	0	
4	00010100	1010000	0	
5	00010100	101000	0	
6	00000000	10100	1	$R - d \geq 0 \Rightarrow R \leftarrow R - d$
7	00000000	1010	10	
8	00000000	101	100	

$x = 20$ et $y = 5$. Par conséquent, le $Q = 4$ et $R = 0$.

Q37) `00010111 / 00001001`

Itération	R	d	Q
init	00010111	100100000000	0
1	00010111	10010000000	0
2	00010111	1001000000	0
3	00010111	100100000	0
4	00010111	10010000	0
5	00010111	1001000	0
6	00010111	100100	0
7	00000101	10010	1
8	00000101	1001	10

$x = 23$ et $y = 9$. Par conséquent, le $Q = 2$ et $R = 5$.

Q38) `11110111` $\times$ `11111010`

```
          11110111
        × 11111010
==========================
          00000000
        + 11111010
--------------------------
          11111010
       + 111110100
--------------------------
         1011101110
      + 1111101000
--------------------------
        11011010110
     + 111110100000
--------------------------
       1011001110110
    + 1111101000000
--------------------------
      11010110110110
   + 11111010000000
--------------------------
     111010000110110
  + 11111010000000000
==========================
   1111000100110110
```

Interprétation en représentation non-signée :
$x = 247$ et $y = 255$. Le résultat est $x \times y = 61750$.

Interprétation en complément à 2 :
$x = -9$ et $y = -6$. Le résultat devrait être $x \times y = 54$, or, le résultat correspond à la représentation de -28983. Nous avons vu au cours que le résultat est cependant correct si on se limite aux N bits de poids faible, soit `00110110`, ce qui correspond effectivement à 54.