

Le mystère des sept dalles en bronze

Jean R. Lobry

19 août 2008

De mystérieuses inscriptions nous conduisent à manipuler des octets et des bits.

Table des matières

1	Introduction	2
2	Manipulation des bits et des octets	2
2.1	Définition	2
2.2	Conversion hexadécimal ↔ binaire	3
2.2.1	Hexadécimal → binaire	3
2.2.2	Hexadécimal ← binaire	4
3	Opérations au niveau des bits	4
3.1	Comparaisons bits à bits	4
3.2	Opérateurs logiques bits à bits	5
3.2.1	Négation logique	5
3.2.2	Le et logique	5
3.2.3	Le ou logique	5
3.2.4	Opérations de décalage de bits	5
4	Décodage des dalles en bronze	6
4.1	Le code ASCII	6
4.2	Importation des données	6
4.3	Conversions du binaire en octets	7
4.4	Conversion des octets en ASCII	8
4.5	Conversion en latin-1	8


1 Introduction

Le visiteur qui pénètre dans l'enceinte du site du service central de l'INPS¹ et du LPS² de Lyon à Écully découvrira sous ses pas les sept mystérieuses dalles en bronze dont les images sont données dans les marges de ce document. Toute l'information est contenue dans ces images, vous pouvez vous amuser à résoudre par vous même le mystère des sept dalles en bronze en ne lisant pas la suite. Pour vous éviter de saisir les données vous pouvez utiliser le fichier `dalle.txt` qui donne les valeurs portées par les sept dalles en bronze. Les 15 premières lignes de ce fichier sont les suivantes :


```
cat(readLines("http://pbil.univ-lyon1.fr/R/donnees/dalle.txt", n = 15),
    sep = "\n")
#
# Dalle 0 :
#
01010011
01010101
01010010
#
# Séparateur :
#
00100000
#
# Dalle 1 :
#
01001100
01000101
```

Êtes-vous capable par vous même de résoudre les mystère des dalles en bronze ?

2 Manipulation des bits et des octets

Nous n'avons que deux symboles utilisés, 0 et 1, il s'agit donc d'un codage binaire. Ces symboles sont regroupés par paquets de 8, autrement dit des octets. Voyons les outils qui sont à notre disposition sous  pour manipuler ce type de données.

2.1 Définition

Un octet est une suite de huit bits, il y a donc $2^8 = 256$ valeurs possibles en tout. En partant de 0, les valeurs possibles sont donc dans l'intervalle $[00000000, 11111111]$ en base 2, dans l'intervalle $[0, 255]$ en base 10, et dans l'intervalle $[00, ff]$ en base hexadécimale. Ils font parti dans  de la classe `raw` et sont affichés en hexadécimal :

```
as.raw(0:255)
 [1] 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19
 [27] 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33
 [53] 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f 40 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d
 [79] 4e 4f 50 51 52 53 54 55 56 57 58 59 5a 5b 5c 5d 5e 5f 60 61 62 63 64 65 66 67
[105] 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77 78 79 7a 7b 7c 7d 7e 7f 80 81
[131] 82 83 84 85 86 87 88 89 8a 8b 8c 8d 8e 8f 90 91 92 93 94 95 96 97 98 99 9a 9b
[157] 9c 9d 9e 9f a0 a1 a2 a3 a4 a5 a6 a7 a8 a9 aa ab ac ad ae af b0 b1 b2 b3 b4 b5
[183] b6 b7 b8 b9 ba bb bc bd be bf c0 c1 c2 c3 c4 c5 c6 c7 c8 c9 ca cb cc cd ce cf
[209] d0 d1 d2 d3 d4 d5 d6 d7 d8 d9 da db dc dd de df e0 e1 e2 e3 e4 e5 e6 e7 e8 e9
[235] ea eb ec ed ee ef f0 f1 f2 f3 f4 f5 f6 f7 f8 f9 fa fb fc fd fe ff
```

¹Institut National de Police Scientifique.

²Laboratoire de Police Scientifique.



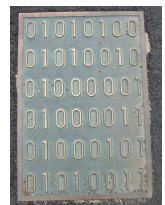
Le site d'Écully où l'on trouve des mystérieuses dalles en bronze



Dalle mystérieuse numéro 0. Prise aérienne du 29-JUL-08 à environ 1.5 m du sol. Une indication d'échelle est donnée par la dimension des pieds du photographe.



Dalle mystérieuse numéro 1.



Dalle mystérieuse numéro 2.

Les constantes hexadécimales sont notées avec le préfixe `0x`, par exemple :

```
0x10
[1] 16
as.raw(0x10)
[1] 10
```

Les fonctions `as.raw()` et `as.integer()` permettent de passer de la notation décimale à hexadécimale, et réciproquement. Vérification :

```
as.raw(16)
[1] 10
as.integer(as.raw(16))
[1] 16
all(as.integer(as.raw(0:255)) == 0:255)
[1] TRUE
as.integer(0x10)
[1] 16
as.raw(as.integer(0x10))
[1] 10
all(as.raw(as.integer(0x00:0xff)) == 0x00:0xff)
[1] TRUE
```

2.2 Conversion hexadécimal ↔ binaire

2.2.1 Hexadécimal → binaire

La fonction `rawToBits()` convertit un vecteur d'octets en un vecteur 8 fois plus long où chaque octet a été décomposé en ses bits constitutifs.

```
rawToBits(as.raw(0x1))
[1] 01 00 00 00 00 00 00 00
rawToBits(as.raw(0x80))
[1] 00 00 00 00 00 00 00 01
```

Le résultat est toujours de la classe `raw` mais seules les valeurs 00 et 01 sont possibles. Ils sont affichés avec deux chiffres car de la classe `raw`. Si on veut une notation plus compacte pour ne pas afficher le premier 0, non informatif ici, il suffit de les convertir en entiers :

```
as.integer(rawToBits(as.raw(0x1)))
[1] 1 0 0 0 0 0 0 0
as.integer(rawToBits(as.raw(0x80)))
[1] 0 0 0 0 0 0 0 1
```

La fonction `rawToBits()` nous renvoie la version gros-boutiens, c'est à dire celle où les bits de poids fort sont à droite. C'est le contraire de l'écriture décimale à laquelle nous sommes habitués. Pour avoir la version petit-boutiens il suffit d'inverser l'ordre des bits :

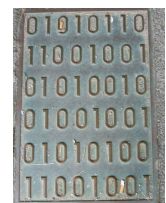
```
rev(as.integer(rawToBits(as.raw(0x1))))
[1] 0 0 0 0 0 0 0 1
rev(as.integer(rawToBits(as.raw(0x80))))
[1] 1 0 0 0 0 0 0 0
```



Dalle mystérieuse numéro 3.



Dalle mystérieuse numéro 4.



Dalle mystérieuse numéro 5.



Dalle mystérieuse numéro 6.

2.2.2 Hexadécimal ← binaire

La fonction `packBits()` transforme un vecteur de bits la classe `integer`, `raw` ou `logical` en un vecteur 8 fois moins long de la classe `raw` :

```
packBits(as.integer(c(0, 0, 0, 0, 0, 0, 0, 1)))
[1] 80
packBits(as.integer(c(1, 0, 0, 0, 0, 0, 0, 0)))
[1] 01
```

On peut aussi utiliser le suffixe `L` pour préciser directement que l'on utilise des constantes entières, le préfixe `0x` pour des constantes hexadécimales, ou enfin `T` et `F` pour des constantes logiques :

```
packBits(c(0L, 0L, 0L, 0L, 0L, 0L, 0L, 1L))
[1] 80
packBits(as.raw(c(0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1)))
[1] 80
packBits(c(F, F, F, F, F, F, F, T))
[1] 80
```

C'est la version gros-boutiens du codage des octets qui est utilisée. Si on veut la version petit-boutiens il faut inverser l'ordre des bits :

```
packBits(rev(as.integer(c(0, 0, 0, 0, 0, 0, 0, 1))))
[1] 01
packBits(rev(as.integer(c(1, 0, 0, 0, 0, 0, 0, 0))))
[1] 80
```

3 Opérations au niveau des bits

3.1 Comparaisons bits à bits

À la différence des variables numériques où il est toujours préférable d'utiliser `all.equal()` pour tester l'égalité numérique entre deux valeurs, pour les variables de type `raw` les opérateurs `==` et `!=` ne posent pas problème puisqu'ils vont travailler *bit à bit* :

```
as.raw(127) == as.raw(127)
[1] TRUE
as.raw(127) != as.raw(127)
[1] FALSE
```

3.2 Opérateurs logiques bits à bits

3.2.1 Négation logique

La négation logique inverse tous les bits :

```
as.raw(1)
[1] 01
!as.raw(1)
[1] fe
```

Pour mieux voir ce qu'il se passe on transforme ceci en version binaire petit-boutiens :

```
r2b <- function(x) as.integer(rev(rawToBits(x)))
r2b(as.raw(1))
[1] 0 0 0 0 0 0 0 1
r2b(!as.raw(1))
[1] 1 1 1 1 1 1 1 0
```

3.2.2 Le et logique

```
r2b(as.raw(3))
[1] 0 0 0 0 0 0 1 1
r2b(as.raw(101))
[1] 0 1 1 0 0 1 0 1
r2b(as.raw(3) & as.raw(101))
[1] 0 0 0 0 0 0 0 1
```

3.2.3 Le ou logique

```
r2b(as.raw(3))
[1] 0 0 0 0 0 0 1 1
r2b(as.raw(101))
[1] 0 1 1 0 0 1 0 1
r2b(as.raw(3) | as.raw(101))
[1] 0 1 1 0 0 1 1 1
```

3.2.4 Opérations de décalage de bits

La fonction `rawShift()` permet de décaler de n bits à gauche ou à droite :

```
x <- as.raw(100)
r2b(x)
[1] 0 1 1 0 0 1 0 0
r2b(rawShift(x, n = +1))
[1] 1 1 0 0 1 0 0 0
r2b(rawShift(x, n = -1))
[1] 0 0 1 1 0 0 1 0
```

Décaler de n bits à gauche revient à multiplier par 2^n , décaler de n bits à droite revient à diviser par 2^n , par exemple :

```
x <- as.raw(16)
as.integer(x) * 2^3
[1] 128
as.integer(rawShift(x, 3))
[1] 128
as.integer(x)/(2^4)
[1] 1
as.integer(rawShift(x, -4))
[1] 1
```


4 Décodage des dalles en bronze

4.1 Le code ASCII

On se doute un peu que les dalles en bronze ont quelque chose à voir avec le code ASCII³. La fonction `rawToChars()` permet de passer d'un octet au caractère du code ASCII correspondant. Le code ASCII n'est que sur 7 bits, il n'y a donc que 128 caractères numérotés de 0 à 127. On peut reconstruire la table du code ASCII ainsi :

```
ascii.df <- data.frame(hex = as.raw(0:127))
row.names(ascii.df) <- 0:127
b2c <- function(x) paste(r2b(x), collapse = "")
ascii.df$bin <- lapply(ascii.df$hex, b2c)
ascii.df$char <- rawToChar(ascii.df$hex, multiple = TRUE)
ascii.df[66:91, ]
  hex      bin char
65 41 01000001  A
66 42 01000010  B
67 43 01000011  C
68 44 01000100  D
69 45 01000101  E
70 46 01000110  F
71 47 01000111  G
72 48 01001000  H
73 49 01001001  I
74 4a 01001010  J
75 4b 01001011  K
76 4c 01001100  L
77 4d 01001101  M
78 4e 01001110  N
79 4f 01001111  O
80 50 01010000  P
81 51 01010001  Q
82 52 01010010  R
83 53 01010011  S
84 54 01010100  T
85 55 01010101  U
86 56 01010110  V
87 57 01010111  W
88 58 01011000  X
89 59 01011001  Y
90 5a 01011010  Z
```

4.2 Importation des données

Importer les données sous  sous la forme de bits. Il n'y a pas de séparateur entre les bits dans le fichier, on utilise donc la fonction `read.fwf()`.

```
dalle <- read.fwf(file = "http://pbil.univ-lyon1.fr/R/donnees/dalle.txt",
  width = rep(1, 8))
dalle
  V1 V2 V3 V4 V5 V6 V7 V8
1  0  1  0  1  0  0  1  1
2  0  1  0  1  0  1  0  1
3  0  1  0  1  0  0  1  0
4  0  0  1  0  0  0  0  0
5  0  1  0  0  1  1  0  0
6  0  1  0  0  0  1  0  1
7  0  1  0  1  0  0  1  1
8  0  0  1  0  0  0  0  0
9  0  1  0  1  0  1  0  0
10 0  1  0  1  0  0  1  0
11 0  1  0  0  0  0  0  1
12 0  1  0  0  0  0  1  1
13 0  1  0  0  0  1  0  1
14 0  1  0  1  0  0  1  1
15 0  0  1  0  0  0  0  0
16 0  1  0  0  0  1  0  0
```

³Acronyme pour American Standard Code for Information Interchange.

```

17 0 1 0 0 0 1 0 1
18 0 0 1 0 0 0 0 0
19 0 1 0 0 1 1 0 0
20 0 1 0 0 0 0 0 1
21 0 0 1 0 0 0 0 0
22 0 1 0 1 0 1 1 0
23 1 1 0 0 1 0 0 1
24 0 1 0 1 0 0 1 0
25 0 1 0 0 1 0 0 1
26 0 1 0 1 0 1 0 0
27 1 1 0 0 1 0 0 1
28 0 0 1 0 0 0 0 0
29 0 0 1 0 1 1 1 0
30 0 0 1 0 1 1 1 0
31 0 0 1 0 1 1 1 0

```

Notez que lignes commençant par le caractère # sont considérées par défaut comme des lignes de commentaires et ne sont pas importées.

```

colSums(dalle)
V1 V2 V3 V4 V5 V6 V7 V8
2 22 9 10 8 13 11 13

```

On note la présence de bits non nuls aux deux extrémités, ce n'est donc pas un simple code ASCII.

4.3 Conversions du binaire en octets

On se sait pas si on a une version grand-boutiens ou petit-boutiens des octets, on conserve les deux.

```

dalle <- read.fwf(file = "http://pbil.univ-lyon1.fr/R/donnees/dalle.txt",
width = rep(1, 8))
dalle$rawgb <- apply(dalle[, 1:8], 1, packBits)
dalle$rawpb <- apply(dalle[, 1:8], 1, function(x) packBits(rev(x)))
dalle

```

	V1	V2	V3	V4	V5	V6	V7	V8	rawgb	rawpb
1	0	1	0	1	0	0	1	1	ca	53
2	0	1	0	1	0	1	0	1	aa	55
3	0	1	0	1	0	0	1	0	4a	52
4	0	0	1	0	0	0	0	0	04	20
5	0	1	0	0	1	1	0	0	32	4c
6	0	1	0	0	0	1	0	1	a2	45
7	0	1	0	1	0	0	1	1	ca	53
8	0	0	1	0	0	0	0	0	04	20
9	0	1	0	1	0	1	0	0	2a	54
10	0	1	0	1	0	0	1	0	4a	52
11	0	1	0	0	0	0	0	1	82	41
12	0	1	0	0	0	0	1	1	c2	43
13	0	1	0	0	0	1	0	1	a2	45
14	0	1	0	1	0	0	1	1	ca	53
15	0	0	1	0	0	0	0	0	04	20
16	0	1	0	0	0	1	0	0	22	44
17	0	1	0	0	0	1	0	1	a2	45
18	0	0	1	0	0	0	0	0	04	20
19	0	1	0	0	1	1	0	0	32	4c
20	0	1	0	0	0	0	0	1	82	41
21	0	0	1	0	0	0	0	0	04	20
22	0	1	0	1	0	1	1	0	6a	56
23	1	1	0	0	1	0	0	1	93	c9
24	0	1	0	1	0	0	1	0	4a	52
25	0	1	0	0	1	0	0	1	92	49
26	0	1	0	1	0	1	0	0	2a	54
27	1	1	0	0	1	0	0	1	93	c9
28	0	0	1	0	0	0	0	0	04	20
29	0	0	1	0	1	1	1	0	74	2e
30	0	0	1	0	1	1	1	0	74	2e
31	0	0	1	0	1	1	1	0	74	2e

4.4 Conversion des octets en ASCII

On regarde la version gros-boutiens et petit-boutiens des codes ASCII correspondant.

```
dalle$chargb <- rawToChar(dalle$rawgb, multiple = TRUE)
dalle$charpb <- rawToChar(dalle$rawpb, multiple = TRUE)
dalle
  V1 V2 V3 V4 V5 V6 V7 V8 rawgb rawpb chargb charpb
1  0  1  0  1  0  0  1  1    ca   53  \xca   S
2  0  1  0  1  0  1  0  1    aa   55  \xaa   U
3  0  1  0  1  0  0  1  0    4a   52    J    R
4  0  0  1  0  0  0  0  0    04   20  \004
5  0  1  0  0  1  1  0  0    32   4c    2    L
6  0  1  0  0  0  1  0  1    a2   45  \xa2   E
7  0  1  0  1  0  0  1  1    ca   53  \xca   S
8  0  0  1  0  0  0  0  0    04   20  \004
9  0  1  0  1  0  1  0  0    2a   54    *    T
10 0  1  0  1  0  0  1  0    4a   52    J    R
11 0  1  0  0  0  0  0  1    82   41  \x82   A
12 0  1  0  0  0  0  1  1    c2   43  \xc2   C
13 0  1  0  0  0  1  0  1    a2   45  \xa2   E
14 0  1  0  1  0  0  1  1    ca   53  \xca   S
15 0  0  1  0  0  0  0  0    04   20  \004
16 0  1  0  0  0  1  0  0    22   44    "    D
17 0  1  0  0  0  1  0  1    a2   45  \xa2   E
18 0  0  1  0  0  0  0  0    04   20  \004
19 0  1  0  0  1  1  0  0    32   4c    2    L
20 0  1  0  0  0  0  0  1    82   41  \x82   A
21 0  0  1  0  0  0  0  0    04   20  \004
22 0  1  0  1  0  1  1  0    6a   56    j    V
23 1  1  0  0  1  0  0  1    93   c9  \x93  \xc9
24 0  1  0  1  0  0  1  0    4a   52    J    R
25 0  1  0  0  1  0  0  1    92   49  \x92   I
26 0  1  0  1  0  1  0  0    2a   54    *    T
27 1  1  0  0  1  0  0  1    93   c9  \x93  \xc9
28 0  0  1  0  0  0  0  0    04   20  \004
29 0  0  1  0  1  1  1  0    74   2e    t    .
30 0  0  1  0  1  1  1  0    74   2e    t    .
31 0  0  1  0  1  1  1  0    74   2e    t    .
```

C'est visiblement la version petit-boutiens qui a été utilisée ici, mais il reste un problème d'encodage.

4.5 Conversion en latin-1

☞ n'est pas limité au code ASCII. On essaye en latin-1⁴ pour voir.

```
dalle$latin1 <- dalle$charpb
Encoding(dalle$latin1) <- "latin1"
dalle
  V1 V2 V3 V4 V5 V6 V7 V8 rawgb rawpb chargb charpb latin1
1  0  1  0  1  0  0  1  1    ca   53  \xca   S    S
2  0  1  0  1  0  1  0  1    aa   55  \xaa   U    U
3  0  1  0  1  0  0  1  0    4a   52    J    R    R
4  0  0  1  0  0  0  0  0    04   20  \004
5  0  1  0  0  1  1  0  0    32   4c    2    L    L
6  0  1  0  0  0  1  0  1    a2   45  \xa2   E    E
7  0  1  0  1  0  0  1  1    ca   53  \xca   S    S
8  0  0  1  0  0  0  0  0    04   20  \004
9  0  1  0  1  0  1  0  0    2a   54    *    T    T
10 0  1  0  1  0  0  1  0    4a   52    J    R    R
11 0  1  0  0  0  0  0  1    82   41  \x82   A    A
12 0  1  0  0  0  0  1  1    c2   43  \xc2   C    C
13 0  1  0  0  0  1  0  1    a2   45  \xa2   E    E
14 0  1  0  1  0  0  1  1    ca   53  \xca   S    S
15 0  0  1  0  0  0  0  0    04   20  \004
16 0  1  0  0  0  1  0  0    22   44    "    D    D
17 0  1  0  0  0  1  0  1    a2   45  \xa2   E    E
18 0  0  1  0  0  0  0  0    04   20  \004
19 0  1  0  0  1  1  0  0    32   4c    2    L    L
20 0  1  0  0  0  0  0  1    82   41  \x82   A    A
```

⁴ISO/CEI 8859-1


```

21 0 0 1 0 0 0 0 0 04 20 \004
22 0 1 0 1 0 1 1 0 6a 56 j V V
23 1 1 0 0 1 0 0 1 93 c9 \x93 \xc9 É
24 0 1 0 1 0 0 1 0 4a 52 J R R
25 0 1 0 0 1 0 0 1 92 49 \x92 I I
26 0 1 0 1 0 1 0 0 2a 54 * T T
27 1 1 0 0 1 0 0 1 93 c9 \x93 \xc9 É
28 0 0 1 0 0 0 0 0 04 20 \004
29 0 0 1 0 1 1 1 0 74 2e t .
30 0 0 1 0 1 1 1 0 74 2e t .
31 0 0 1 0 1 1 1 0 74 2e t .

```

Le mystère des dalles en bronze est maintenant résolu. Le visiteur qui pénètre dans l'enceinte du site du service central de l'INPS et du LPS de Lyon à Écully posera ses pieds *SUR LES TRACES DE LA VÉRITÉ* ...