CORRECTION

Section : ***S***	Option : ***Sciences de l'ingénieur***	Discipline : ***Génie Électrique***	
Représentation numérique de l'information			
Domaine d'application : **Représentation conventionnelle des systèmes**	Type de document : **Cours**	Classe : **Première**	Date :

I – Les systèmes de numération

Définition : Un système de numération est un moyen de représenter les nombres en utilisant différents symboles. Il existe plusieurs systèmes de numération, chacun d'entre eux utilisant un nombre particulier de symboles. Le nombre de symboles utilisés par un système de numération est appelé la base du système.

Expérience : On a tous eu un jour l'occasion de compter une quantité importante de petits objets : des pièces de monnaie, des billes, des cartes, etc. Notre compte fini, on en effectue un deuxième afin d'être certain de ne pas s'être trompé. Mais il est rare, malheureusement, de tomber deux fois sur le même résultat. Et là, notre esprit ingénieux nous conseille d'user d'un stratagème pour ne pas se faire posséder une nouvelle fois par le grand nombre : on fait des petits paquets de 10 ! Et si cela ne suffit pas : avec 10 petits paquets de 10, nous formons un gros paquet de 100.
Nous réinventons un système de numération de **base 10**. Pourquoi « de base 10 », car pour obtenir un petit paquet, il faut 10 unités et pour obtenir un gros paquet, il faut 10 petits paquets. C'est notre système de numération actuel, composé de 10 symboles [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]. Pour passer au rang des dizaines [petits paquets], il faut 10 unités et pour passer au rang des centaines [gros paquets] il faut 10 dizaines.

I – 1 – Le décimal

Le décimal est le système de numération à base 10 : il utilise donc 10 symboles pour représenter les nombres.

Ces symboles sont : 0 1 2 3 4 5 6 7 8 9

En décimal, chaque symbole est appelé un chiffre, et un ensemble de chiffre est appelé un nombre. Dans le nombre, chaque chiffre a un rang : on parle d'unités, dizaines, centaines, milliers, etc ... On dit alors que le système décimal est un système ..... PONDÉRÉ ..... Exemple de nombre décimal : .... $4735_{(10)}$

Rang →	5	4	3	2	1	0
Poids →	10^5	10^4	$10^3 = 1000$	$10^2 = 100$	$10^1 = 10$	$10^0 = 1$
Chiffre →	0	0	4	7	3	5
Valeur →	0	0	4000	700	30	5
Total →	$4000 + 700 + 30 + 5 = 4735$					

En une seule ligne on peut écrire que : $4735 = 4.10^3 + 7.10^2 + 3.10^1 + 5.10^0$

Remarques :

* en décimal, un nombre exprimé sur n chiffres peut prendre 10^n valeurs différentes
* pour préciser qu'un nombre est exprimé en décimal, on indique (10) en bas à droite
* le décimal est notre système de numération usuel

I – 2 – Le binaire naturel

Le binaire naturel est le système de numération à base 2 : il utilise donc 2 symboles pour représenter les nombres.

Ces symboles sont : 0 et 1. En binaire naturel, chaque symbole est appelé un bit, et un ensemble de bit est appelé un mot. Un mot de 4 bits est appelé un QUARTET. Un mot de 8 bits est appelé un OCTET.

Le binaire naturel est un système pondéré : chaque bit a un poids en fonction de sa position dans le mot. Exemple de nombre binaire : $11010_{(2)}$

Rang →	5	4	3	2	1	0
Poids →	2^5	2^4	2^3	2^2	2^1	2^0
Chiffre →	0	1	1	0	1	0
Valeur →	0	16	8	0	2	0
Total →	$16 + 8 + 2 = 26_{(10)}$					

Le nombre binaire 11010 s'écrit donc 26 en décimal, ce qui s'écrit :

$$11010_{(2)} \equiv 26_{(10)}$$

Le symbole ≡ [qui n'est pas le symbole « égale à »] signifie et se lit « correspond à ». Lien entre un nombre binaire et son équivalent en décimal :

$$11010_{(2)} \equiv 1.2^4 + 1.2^3 + 0.2^2 + 1.2^1 + 0.2^0 = 26_{(10)}$$

Pour interpréter un nombre en binaire naturel, il faut connaître les puissances de 2 :

Puissance de 2 →	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Valeur en décimal →	4096	2048	1024	512	256	128	64	32	16	8	4	2	1

Remarques :

* en binaire naturel un nombre de n bits peut prendre 2^n valeurs différentes
* pour préciser qu'un nombre est exprimé en binaire, on indique (2) à sa droite
* dans un nombre binaire, le bit de rang 0 est appelé BIT DE POIDS FAIBLE (LSB)
* le bit de poids le plus élevé est appelé BIT DE POIDS FORT (MSB en anglais)
* le binaire est le seul système utilisé par les ordinateurs

Le lien entre le poids d'un chiffre, son rang, et la base du système de numération est donné ci-contre. Ce lien est valable pour tous les systèmes de numération pondérés :

$$\text{POIDS} = \text{BASE}^{\text{RANG}}$$

I - 3 - L'hexadécimal

L'hexadécimal est le système de numération à base 16 : il utilise donc 16 symboles différents pour représenter les nombres.

Ces symboles sont : 0 1 2 3 4 5 6 7 8 9 A B C D E F

Lien entre les 16 symboles hexadécimaux et leur équivalent en décimal :

Symbole hexadécimal →	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Equivalent en décimal →	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

RANG

L'hexadécimal est un système pondéré : POIDS = 16

Pour interpréter un nombre hexadécimal, il faut connaître les puissances de 16 :

Puissance de 16 →	16^5	16^4	16^3	16^2	16^1	16^0
Valeur en décimal →	1048 576	65 536	4096	256	16	1

Exemple de nombre hexadécimal : $E2A7_{(16)}$

Rang →	5	4	3	2	1	0
Poids →	16^5	16^4	16^3	16^2	16^1	16^0
Chiffre →	0	0	E	2	A	7
Valeur →	0	0	57 344	512	160	7
Total →	57 344 + 512 + 160 + 7 = 58 023					

Lien entre un nombre hexadécimal et son équivalent en décimal :

$E2A7_{(16)} \equiv 14 \cdot 16^3 + 2 \cdot 16^2 + 10 \cdot 16^1 + 7 \cdot 16^0 = 58\,023_{(10)}$

Remarques :

* en hexadécimal, un nombre sur n chiffre peut prendre 16^n valeurs différentes
* pour préciser qu'un nombre est en hexadécimal, on indique (16) à sa droite
* l'hexadécimal est utilisé pour manipuler facilement les grands nombres binaires.

II – Conversion d'un nombre d'une base vers une autre

II – 1 – La conversion binaire → décimal

Principe : Pour convertir en décimal un nombre exprimé à l'origine en binaire naturel, il faut additionner le POIDS des bits qui sont à 1.

Exemples : $101_{(2)} \equiv 4 + 1 = 5_{(10)}$

$10110_{(2)} \equiv 16 + 4 + 2 = 22_{(10)}$

$1101001_{(2)} \equiv 1 + 8 + 32 + 64 = 105_{(10)}$

II – 2 – La conversion décimal → binaire

Principe : Il faut décomposer le nombre décimal en somme de puissance de 2 afin de savoir quels sont les bits qui sont à 1 dans le nombre binaire.

Exemples : $17_{(10)} = 16 + 1 = 2^4 + 2^0 \equiv 10001_{(2)}$

$140_{(10)} = 128 + 12 = 128 + 8 + 4 = 2^7 + 2^3 + 2^2 = 10001100_{(2)}$

II – 3 – La conversion binaire → hexadécimal

Principe : On regroupe les bits par paquet de 4 en partant de la droite, puis on convertit chaque quartet en un chiffre hexadécimal.

Exemples : $1001\,0011_{(2)} \equiv 93_{(16)}$

$10\,0111\,1100\,1011_{(2)} \equiv 27CB_{(16)}$

2 7 C B

II - 4 - La conversion hexadécimal → binaire

Principe : On convertit chaque chiffre hexadécimal en un quartet, c'est-à-dire en binaire sur 4 bits

Exemples : $A64E_{(16)} \equiv 1010\,0110\,0100\,1110_{(2)}$ (A 6 4 E)

$F21_{(16)} \equiv 1111\,0010\,0001_{(2)}$ (F 2 1)

III - Opération sur les nombres binaires

III - 1 - Les opérations logiques

III - 1 - 1 - Le ET bits à bits

Cette opération, appliqué sur 2 nombres binaires, consiste à effectuer un ET logique entre deux bits du même rang. Exemples :

```
  1101
ET0111
------
  0101
```

```
  100110
ET 1011
-------
000010
```

Le ET bits à bits permet de forcer à 0 certains bits d'un nombre binaire. Par exemple, pour forcer à 0 les bits de rang 1 et 6 du nombre $1101001011_{(2)}$ il faut effectuer l'opération ci-contre :

Forçage de 2 bits à 0

```
   1 1 0 1 0 0 1 0 1 1
ET 1 1 1 0 1 1 1 1 0 1
-----------------------
=  1 1 0 0 0 1 0 0 1
```

III - 1 - 2 - Le OU bits à bits

Cette opération, appliqué sur 2 nombres binaires, consiste à effectuer un OU logique entre deux bits du même rang. Exemples :

```
  1001
OU1011
------
  1011
```

```
  100001
OU  1100
--------
101101
```

Le OU bits à bits permet de forcer à 1 certains bits d'un nombre binaire. Par exemple, pour forcer à 1 les bits de rang 2 et 7 du nombre $1101001011_{(2)}$ il faut effectuer l'opération ci-contre :

Forçage de 2 bits à 1

```
   1 1 0 1 0 0 1 0 1 1
OU 0 0 1 0 0 0 0 1 0 0
-----------------------
=  1 1 1 1 0 0 1 1 1 1
```

III - 1 - 3 - Le OU-Exclusif bits à bits

Cette opération, appliqué sur 2 nombres binaires, consiste à effectuer un OU-Exclusif logique entre deux bits du même rang. Exemples :

```
  10100
⊕ 11001
-------
  01101
```

```
  1011100
⊕    1001
---------
1010101
```

complémentation de 2 bits

Le OU-Exclusif bits à bits permet de complémenter certains bits d'un nombre binaire. Par exemple, pour complémenter les bits de rang 0 et 5 du nombre $1101001011_{(2)}$ il faut effectuer l'opération ci-contre :

$$\begin{array}{rcccccccccc} & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ \oplus & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ \hline = & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \end{array}$$

III - 2 - Les opérations arithmétiques

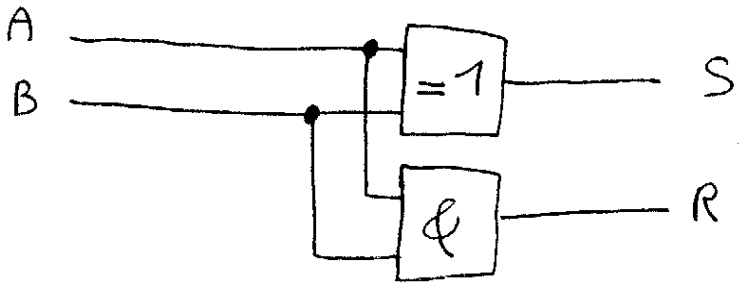
III - 2 - 1 - L'addition de deux nombres binaires

Comme en décimal, l'addition de deux ***nombres*** binaires s'appuie sur l'utilisation d'une table d'addition, indiquant toutes les possibilités pour additionner deux ***chiffres*** binaires [deux bits]. Cette table d'addition élémentaire est donnée ci-contre : elle indique la somme **S** et la retenue éventuelle **R** lorsque l'on addition deux bits **A** et **B**. On peut remarquer que les équations logiques de **S** et de **R** en fonction de **A** et **B** sont :

A	B	S	R
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

S = $A \oplus B$ R = $A \cdot B$

Le montage ci-contre, appelé *« demi-additionneur »*, permet d'additionner 2 bits **A** et **B**, et donne en sortie la somme **S** ainsi qu'une retenue éventuelle **R** :



Exemples d'addition de 2 nombres binaires :

$$\begin{array}{r} {\scriptstyle 1\,1} \\ 1010 \leftarrow 10 \\ +\ 0110 \leftarrow 6 \\ \hline 10000 \leftarrow 16 \end{array} \qquad \begin{array}{r} {\scriptstyle 1} \\ 1000100 \leftarrow 68 \\ +\ 100101 \leftarrow 37 \\ \hline 1101001 \leftarrow 105 \end{array}$$

III - 2 - 2 - Le décalage à gauche

Cette opération consiste à décaler chaque bits d'un nombre binaire d'un rang vers la gauche : le bit de rang n est placé au rang n + 1 et le LSB prend la valeur 0. Exemples :

$S_{(10)} \rightarrow 101_{(2)}$ décalé à gauche d'1 bit devient $1010_{(2)}$ ($\leftarrow 10_{(10)}$)

$10111_{(2)}$ décalé à gauche de 2 bits devient $1011100_{(2)}$

$23_{(10)}$ $92_{(10)} = 4 \times 23$

Remarque : décaler un nombre binaire d'un bit vers la gauche revient à multiplier ce nombre par 2.

III - 2 - 3 - Le décalage à droite

Cette opération consiste à décaler chaque bits d'un nombre binaire d'un rang vers la droite : le bit de rang n est placé au rang n-1 et le MSB prend la valeur 0. Exemples :

$101_{(2)}$ décalé à droite d'1 bit devient $10_{(2)}$ (le LSB est perdu)

$1100_{(2)}$ décalé à droite d'1 bit devient $110_{(2)}$

$12_{(10)}$ $6_{(10)}$

Remarque : décaler un nombre binaire pair d'un bit vers la droite revient à diviser ce nombre par 2.

Correction scannée et imprimée le 17 février 2012