NOMBRES ET CHAMPS DE BITS

PIERRE-YVES ROCHAT, EPFL

ET YVES TIECOURA, INP-HB

% rév 2015/12/25

Document en cours de relecture, rév du 2015/12/13 Ce document n'est pas à jour, il n'a pas encore été adapté aux modifications des diapositives pour la vidéo.

BASCULES ET REGISTRES

Une LED peut être, à un instant donné, complètement éteinte ou allumée à un degré d'intensité ajustable. Cet état est généralement mémorisé par une bascule.

Les enseignes et afficheurs à LED utilisent beaucoup de registres, composés de bascules. Ces registres sont souvent de 8 bits ou de 16 bits, mais on en trouve aussi avec des valeurs beaucoup plus grandes.

L'état de chaque LED est aussi fréquemment mémorisé dans la mémoire d'un microcontrôleur. Le processeur du microcontrôleur reçoit des données, il les traite et diffuse les résultats sur ses sorties. Tous les systèmes informatiques travaillent en binaire. Historiquement, des systèmes ternaires (fonctionnant en base 3) ont été développés, mais ils sont extrêmement rares et pratiquement introuvables sur le marché.

....

Champ de bit

On appelle "*bit*" un symbole binaire. Il peut prendre les valeurs *0* et *1*, qui peuvent aussi s'appeler *vrai* et *faux*, *allumé* et *éteint*, etc. C'est un mot-valise composé de la fusion des mots de la locution anglaise *binary digit* ou *chiffre binaire* en français.

On désigne par mot binaire, ou champ de bits, un ensemble de bits. Des opérations logiques peuvent s'appliquer à ces champs de bits (NON, ET, OU, etc.). Elles seront étudiées plus loin dans ce cours.

Nombres binaires

Un champ de bit peut aussi représenter un nombre. La numération binaire est bien connue :

Codage binaire

Binaire	Décimal
0	0
1	1
10	2
11	3
100	4
101	5
110	6
111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15
10000	16

Poids des bits

Rang	Valeur	Décimal
0	1	$1 = 2^0$
1	10	$2 = 2^1$
2	100	$4 = 2^2$
3	1000	$8 = 2^3$
4	10000	$16 = 2^4$
5	100000	$32 = 2^5$
6	1000000	$64 = 2^6$
7	10000000	$128 = 2^7$
8	100000000	$256 = 2^8$
9	1000000000	$512 = 2^9$
10	10000000000	$1024 = 2^{10}$

Figure : Numération binaire

... poids...

Par exemple, 2345 (en décimal) s'exprime par 100100101001 en nombre binaire. Preuve en est :

```
  1×1  + 0×2   + 0×4   + 1×8   + 0×16   + 1×32
+ 0×64 + 0×128 + 1×256 + 0×512 + 0×1024 + 1×2048 = 2345
```

Pour coder un nombre décimal en binaire, on effectue des divisions entières successives par 2 jusqu'à ce que le quotient soit nul. Le premier reste est le poids faible, le dernier est le poids fort.

Arithmétique modulaire

Lorsqu'un nombre est matérialisé dans un circuit électronique, il a forcément une taille limitée.

On peut utiliser ces nombres pour des calculs. Mais il faut être attentif au fait qu'ils ont une limite dans leur taille. En étudiant les mathématiques, on prend l'habitude d'utiliser des nombres immatériels, qui peuvent être aussi grands que nécessaire. Lorsqu'un nombre doit être matérialisé dans un dispositif physique, dans notre cas dans un registre ou une mémoire d'ordinateur, sa taille est forcément limitée. On se trouve alors en face d'une arithmétique différente, l'*arithmétique modulaire*.

Pour bien la comprendre, prenons l'exemple des nombres représentés par 3 bits. Ils peuvent prendre 8 valeurs ($8 = 2^3$).

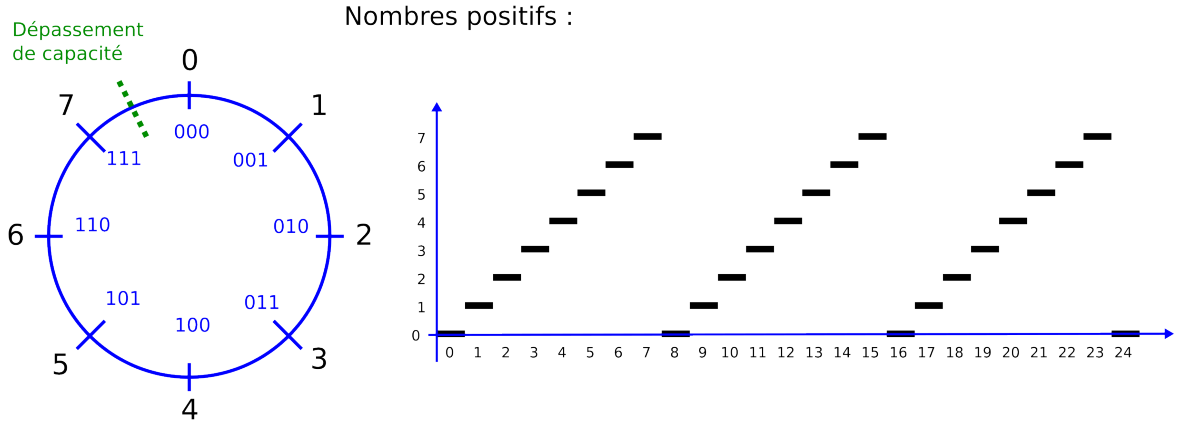

Figure : Nombres positifs sur 3 bits

On voit qu'il n'est possible de représenter qu'un nombre limité de valeurs. S'il s'agissait de nombres de 8 bits, on aurait un choix de 256 valeurs (de 0 à 255). Pour des nombres de 16 bits, on aurait 65'536 valeurs (de 0 à 65'535).

Sur le cercle qui représente l'ensemble des valeurs possibles, l'incrémentation (addition de *1*) correspond à une avance dans un sens. Lorsqu'on dépasse la valeur la plus grande (*7* dans le cas de 3 bits), on retrouve la valeur *0*. On a franchi la limite du dépassement de capacité (*overflow* en anglais).

La décrémentation (soustraction de *1*) correspond au sens contraire. Un dépassement de capacité se produit aussi lors du passage de *0* à la valeur la plus grande.

Les opérations arithmétiques classiques sur les nombres entiers doivent donc tenir compte du dépassement de capacité. Il s'agit de l'arithmétique modulaire. Dans le cas de 3 bits le résultat est donné *modulo 8*. L'opération Modulo correspond aussi au reste de la division entière.

Prenons quelques exemples :

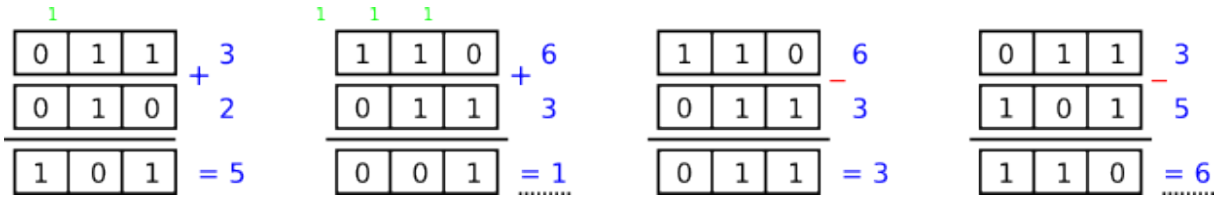

Figure : Opérations sur des nombres de 3 bits

Nombres signés

Dans l'usage courant, les nombres peuvent être positifs ou négatifs. Est-ce possible de les représenter en binaire ? Il existe beaucoup de manières de le faire et plusieurs d'entre elles ont été utilisées au cours de l'histoire de l'informatique. Mais c'est la représentation appelée *en complément à 2* qui est de loin la plus utilisée actuellement.

Voici une figure qui en explique le principe, appliqué à des nombres de 3 bits :

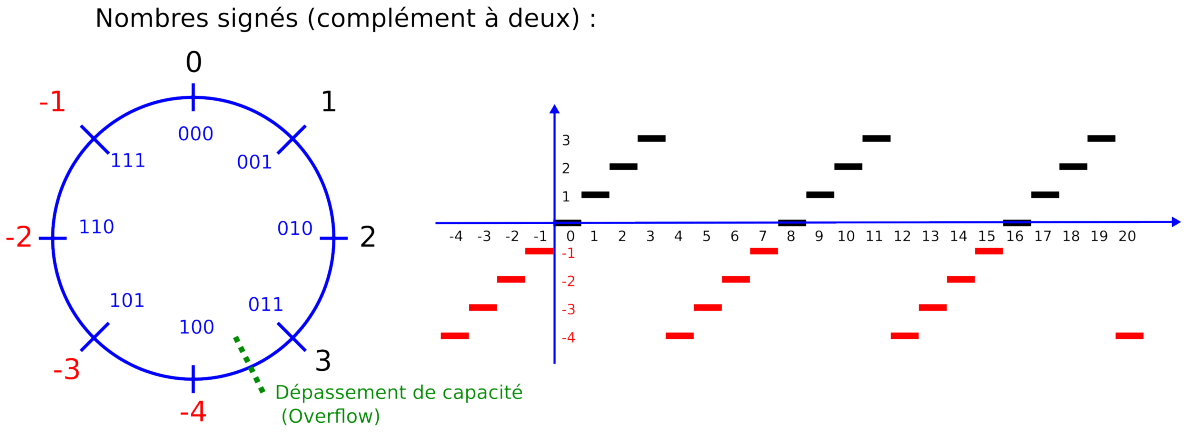

Figure : Nombres positifs et négatifs sur 3 bits

On remarque que le nombre est négatif lorsque le bit de poids fort (celui de gauche) a pour valeur *1*.

Types en C

Les langages de programmation définissent aussi des types avec des nombres entiers d'une taille limitée. Les types permettent d'allouer l'espace mémoire optimal à chaque format.

Les types “historiques” du langage C sont :

Type	**Description**
`char`	mot de 8 bits *
`signed char`	mot de 8 bits signé
`unsigned char`	mot de 8 bits positif
`int`	mot *généralement* de 16 bits *
`signed int`	mot de 16 bits signé
`unsigned int`	mot de 16 bits positif
`long int`	mot *généralement* de 32 bits *
`signed long int`	mot de 32 bits signé
`unsigned long int`	mot de 32 bits positif

* (signé ou non signé, selon les réglages du compilateur)

Ces notations sont souvent ambigües. On préfère maintenant une notation plus claire, standardisée depuis la version C99 de 1999 :

Type	Description
`int8_t`	mot de 8 bits signé
`uint8_t`	mot de 8 bits positif
`int16_t`	mot de 16 bits signé
`uint16_t`	mot de 16 bits positif
`int32_t`	mot de 32 bits signé
`uint32_t`	mot de 32 bits positif

C'est cette notation que nous utiliserons dans ce cours.

Les opérations arithmétiques disponibles pour ces types sont :

Opération	Symbole
l'addition	+
la soustraction	–
la multiplication	*
la division entière	/
le reste de la division entière, appelée aussi modulo	%

Hexadécimal

Dans notre exemple précédent, le nombre 2345, qui est composé de quatre chiffres en décimal, nécessite déjà 12 bits en binaire. L'écriture dans cette base est fastidieuse pour l'être humain !

En utilisant une autre base qui est aussi une puissance de 2, on bénéficie d'une conversion très simple en base 2. La base la plus couramment utilisée est la base 16, appelée “hexadécimal”.

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

Figure : Hexadécimal

Pour convertir un nombre binaire en hexadécimal, on le sépare en tranches de 4 bits de la droite vers la gauche et on complète à gauche avec des zéros non significatifs.

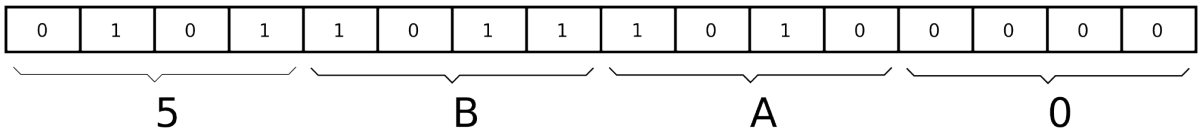

Figure : Conversion binaire-hexadécimal

Pour convertir un nombre hexadécimal en binaire, il faut simplement écrire les 4 valeurs binaires de chaque chiffre hexadécimal.

Codage des caractères

Parmi les données traitées par les systèmes informatiques (par exemple un microcontrôleur), on trouve souvent des caractères. Pour représenter les caractères, on utilise des tables de transcodage vers le binaire.

Le codage ASCII (*American Standard Code for Information Interchange*) sur 7 bits a été standardisé dans les années 1960.

Table de codage ASCII

	00_h 0.	01_h 1.	02_h 2.	03_h 3.	04_h 4.	05_h 5.	06_h 6.	07_h 7.	08_h 8	09_h 9.	$0A_h$ 10.	$0B_h$ 11.	$0C_h$ 12.	$0E_h$ 13.	$0E_h$ 14.	$0F_h$ 15.
00_h 0.	NUL	SOH	STX	ETX	EOT	ENQ	ENQ	BEL	BS	HT	LF	VT	FF	CR	SO	SI
10_h 16.	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
20_h 32.		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30_h 48.	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40_h 64.	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50_h 80.	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60_h 96.	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
60_h 112.	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Pour trouver le code d'un caractère, ajouter la valeur de la ligne et celle de la colonne en décimal ou en hexadécimal

Figure : Caractères ASCII

Malheureusement, les caractères accentués n'étant pas standardisés par la table ASCII, un grand nombre de tables sont apparues, qui cohabitent encore à notre époque de l'internet.

Une des tables les plus souvent utilisées est l'UTF-8.