Informatique Générale

Guillaume Hutzler
Laboratoire IBISC
(Informatique Biologie Intégrative et Systèmes Complexes)
guillaume.hutzler@ibisc.univ-evry.fr
Cours Dokeos 625
http://www.ens.univ-evry.fr/modx/dokeos.html

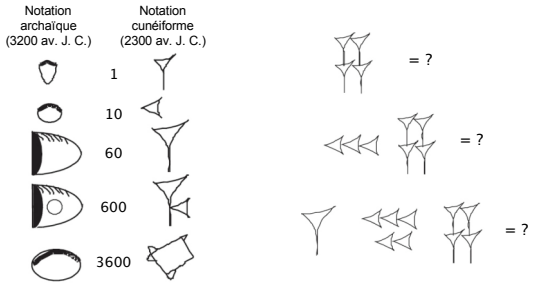
Plan et objectifs du cours

- Objectifs du cours
  - Donner une vue d'ensemble de l'informatique
    - du point de vue historique
    - du point de vue des concepts
    - du point de vue des techniques
  - Donner un aperçu des métiers de l'informatique
- Séances
  - 1-2 : Histoire de l'informatique
  - 3-4 : Fondements mathématiques de l'informatique
  - 5-6 : Architecture des ordinateurs et des micro-processeurs
  - 7-8 : Systèmes d'exploitation
  - 9-10 : Langages de programmation
  - 11-12 : Réseaux

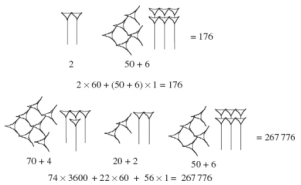
Informatique Générale
Arithmétique binaire et codage des données

Guillaume Hutzler
Laboratoire IBISC
(Informatique Biologie Intégrative et Systèmes Complexes)
guillaume.hutzler@ibisc.univ-evry.fr

Système de numération additif de Sumer



Système positionnel de Babylone



- Strictement positionnel
  - à base 60
  - base 10 auxiliaire
- pb spécifique pour représenter le 0
  - ajout d'un espace puis d'un signe spécifique 

F. Bacon – le codage binaire (1623)

- But = crypter un texte pour qu'il ne puisse pas être déchiffré
  - lettres de l'alphabet remplacées par des séquences de 5 caractères `a` ou `b` (alphabet bilitère)

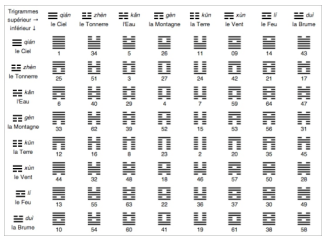
A	B	C	D	E	F
aaaaa	aaaab	aaaba	aaabb	aabaa	aabab
G	H	I	K	L	M
aabba	aabbb	abaaa	abaab	ababa	ababb
N	O	P	Q	R	S
abbaa	abbab	abbba	abbbb	baaaa	baaab
T	U	W	X	Y	Z
baaba	baabb	babaa	babab	babba	babbb

  - un texte de couverture quelconque est imprimé en utilisant deux styles typographiques distincts, l'un associé au `a`, l'autre associé au `b`
  - ex:

```
Neparitezsurtoutpassansmoi
aabab|baabb|babba|aabaa|babbb
  f  |  u  |  y  |  e  |  z
```

G. v. Leibniz – Numération binaire (1666)

- But = langage universel permettant une représentation exacte de toutes réalités ; réduire toutes les opérations logiques à un calcul
  - inspiré par le codage binaire de Bacon
  - inspiré par le Yi Jing (système de divination chinois – 3000 av. J. C.)



IBM – SSEC (1947)

- Repères
  - Selective Sequence Electronic Computer
  - inspiré du Harvard Mark I
  - dernier et plus complexe des monstres électromécaniques
- Caractéristiques
  - Ordinateur hybride
    - 13000 tubes à vide (arithmétique)
    - 8 registres rapides et 23000 relais (contrôle)
  - 50 instructions/s
  - **Codage des chiffres de 0 à 9 par code binaire**
    - **BCD (Binary Coded Decimal)**
  - Programmation
    - programme écrit sur bande papier perforée
    - registres internes pouvant contenir des instructions
    - possibilité de boucles, branchements conditionnels, sauts

Le codage BCD (Binary Coded Decimal)

- codage des nombres d'une façon relativement proche de la représentation humaine usuelle
  - nombres représentés en chiffres décimaux
  - chacun des chiffres est codé sur 4 bits selon la table de correspondance suivante :

Chiffre	Code BCD	Chiffre	Code BCD
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

Le codage des données

- Problème
  - la machine ne sait manipuler que des valeurs binaires (bits)
    - le transistor ne permet de distinguer que deux états différents :
      - une différence de tension aux bornes
      - pas de différence de tension aux bornes
  - comment passer de la manipulation de bits au traitement de l'information au sens large ?
- Réponse
  - par le codage/décodage des données
    = associer à tout type d'information (texte, image, son, etc.), une représentation par l'intermédiaire d'un code
    - ex. : le codage de Bacon / le code ASCII pour le texte
  - par l'utilisation de l'arithmétique binaire et de l'algèbre de Boole pour manipuler les données codées
  - par l'utilisation de logiciels ou de matériels pour implémenter la règle de codage

Exemple pour la représentation des entiers

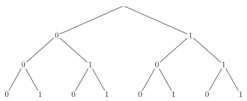
- Une addition
  - en décimal : $137_d + 72_d = 209_d$
  - en binaire : $10001001_b + 01001000_b = 11010001_b$
- Codage
  - chaque nombre est représenté sous la forme d'un entier signé ou non signé sur un octet (ou 2 ou 4 ou 8)
  - un nombre décimal est représenté en binaire en effectuant la conversion de la base 10 vers la base 2
- Addition binaire
  - même principe que pour l'addition en décimal, mais en base 2
  - formalisation possible grâce à l'algèbre de Boole
- Circuits additionneurs
  - circuits électroniques réalisant l'addition de 2 entiers de *n* bits

La notion de bit

- Contraction de ***binary digit***
  = composant élémentaire d'information, ne pouvant se trouver que dans deux états distincts, exclusifs l'un de l'autre
- Différents dispositifs matériels possibles
  - relais électromagnétique ouvert ou fermé
  - lampe électrique allumée ou éteinte (tube à vide)
  - fil électrique dans lequel le courant circule ou non (circuits intégrés, couplé au transistor, sorte d'interrupteur miniature)
  - fibre optique avec ou sans lumière
  - aimant polarisé « sud » ou « nord » (mémoires)
  - surface avec des creux ou des bosses (cylindres des boîtes à musique, CD/DVD)
  - récipient plein ou vide (calculateur à eau à la Cité des Sciences et de l'Industrie de La Villette)
  - etc.
- Par convention, on note 0 et 1 les deux états possibles d'un bit

Combinaisons de bits

- Avec 2 bits, 4 combinaisons possibles
  - `00, 01, 10, 11`
- Avec 3 bits, 8 combinaisons possibles
  - `000, 001, 010, 011, 100, 101, 110, 111`



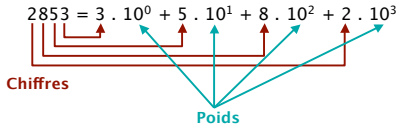
- Avec 1 bit de plus, 2 fois plus de combinaisons possibles
  - le bit ajouté peut avoir lui-même 2 valeurs différentes
  - revient à dédoubler les feuilles de l'arbre
- Avec n bits, 2^n combinaisons possibles

Le bit = unité de comptage de la mémoire

- Unités dérivées = toujours des multiples de 2
  - 1 octet (byte en anglais) = 8 bits (2^3 bits)
    - permet de représenter 2^8 = 256 valeurs différentes
  - 1 kilo-octet (Ko) = 2^{10} octets = 1024 octets ~ 10^3 octets
  - 1 méga-octet (Mo) = 2^{10} kilo-octets = 2^{20} octets ~ 10^6 octets
  - 1 giga-octet (Go) = 2^{10} méga-octets = 2^{30} octets ~ 10^9 octets
  - 1 téra-octet (To) = 2^{10} giga-octets = 2^{40} octets ~ 10^{12} octets
- Exemples
  - disquette ~ 1Mo
  - clé USB ~ 64 Mo – 4 Go
  - CD ~ 700 Mo
  - DVD ~ 5 Go – 17 Go
  - Disques durs ~ 50 Go – 500 Go

Le système décimal

- Dans l'écriture décimale

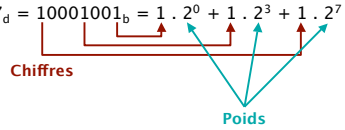


- Chiffres : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Poids : 10^0, 10^1, 10^2, 10^3, etc. (base 10)

Le système binaire

- Dans l'écriture binaire

$137_d = 10001001_b = 1 . 2^0 + 1 . 2^3 + 1 . 2^7$



- Chiffres : 0, 1
- Poids : 2^0, 2^1, 2^2, 2^3, etc. (base 2)

Du décimal au binaire

- Dans l'écriture décimale
  - ajouter un zéro à droite = multiplier par 10
  - supprimer un zéro à droite = diviser par 10
- Dans l'écriture binaire
  - ajouter un zéro à droite = multiplier par 2
  - supprimer un zéro à droite = diviser par 2

Décimal	Binaire
2853 = (285 * 10) + 3	10001001
quotient | reste de division par 10	quotient | reste de division par 2

Du décimal au binaire

$10001001_b = 137_d$

$10001000_b = 136_d$

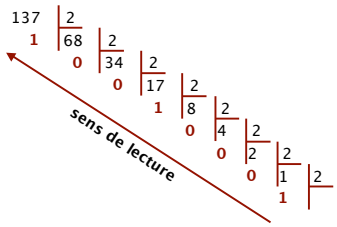
$1000100_b = 68_d$

$137_d = 2 * 68_d + 1$

➲ le chiffre de droite est le reste de la division par deux

➲ on recommence sur le quotient

Exemple : $137_d = 10001001_b$



Les entiers non signés (entiers naturels)

- codage en base 2 mais avec taille fixe
  - entiers courts : (en général) sur 2 octets
  - entiers longs : (en général) sur 4 octets
- capacité limitée de représentation
  - il y a un plus petit entier (que des 0)
    - sur 2 octets
      - $0000000000000000_b = 0$
  - il y a un plus grand entier (que des 1)
    - sur 2 octets
      - $1111111111111111_b = 2^0 + 2^1 + ... + 2^{15} = 2^{16} - 1 \sim 2^6 * 2^{10} \sim 64$ K
      - en réalité 65535
    - sur 4 octets
      - $2^{32} - 1 \sim 2^{32} \sim 4.2^{30} \sim 4$ Gigas
      - en réalité 4 294 967 295
  - quand on calcule
    - sur 2 octets
      - $1111111111111111_b + 1 = 0$

Représentation hexadécimale

- Constat
  - la notation binaire n'est pas très agréable ni facile à utiliser pour visualiser des nombres (par exemples des adresses mémoire)
  - la notation décimale n'est pas très appropriée car elle implique des conversions compliquées
- Solution
  - utiliser une notation hexadécimale
    - plus facile à comprendre et à manipuler que la notation binaire
    - plus facile à convertir en binaire que la notation décimale
  - principe
    - grouper les bits par paquets de 4 (seulement 16 combinaisons possibles)
    - associer un chiffre hexadécimal à chaque paquet de 4 bits

Les chiffres hexadécimaux

Binaire	Décimal	Hexa	Binaire	Décimal	Hexa
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	10	A
0011	3	3	1011	11	B
0100	4	4	1100	12	C
0101	5	5	1101	13	D
0110	6	6	1110	14	E
0111	7	7	1111	15	F

Du binaire à l'hexadécimal

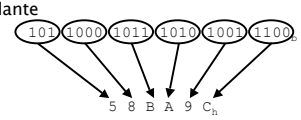
Soit le nombre binaire suivant :

$$10110001011101010011100_b$$

1. Séparer les bits en paquets de 4 ***en partant de la droite***

$$101\ 1000\ 1011\ 1010\ 1001\ 1100_b$$

2. Remplacer chaque paquet par la valeur hexadécimale correspondante



- Le plus grand entier sur 32 bits peut s'écrire en hexa :

$$FFFF\ FFFF_h$$

De l'hexadécimal au décimal

- Numérotation en base 16
  - décaler de 1 chiffre hexadécimal, c'est décaler de 4 chiffres binaires, donc multiplier ou diviser par 16
- Exemple

$$101\ 1000\ 1011\ 1010\ 1001\ 1100_b$$

$$= 5\ 8\ B\ A\ 9\ C_h$$

$$= 12*16^0 + 9*16^1 + 10*16^2 + 11*16^3 + 8*16^4 + 5*16^5$$

$$= 12 + 9*16 + 10*256 + 11*4096 + 8*65536 + 5*1048576$$

$$= 5\ 671\ 580_d$$

L'addition en binaire

- Même principe que l'addition en décimal
  - addition en colonne avec report de la retenue
- Additions élémentaires en binaire
  - $0_b + 0_b = 0_b$
  - $0_b + 1_b = 1_b + 0_b = 1_b$
  - $1_b + 1_b = 10_b$ (0_b et une retenue)
  - $1_b + 1_b + 1_b = 11_b$ (1_b et une retenue)
- Exemple

```
              1
    1  0  0  0  1  0  0  1
+   0  1  0  0  1  0  0  0
-------------------------
    1  1  0  1  0  0  0  1
```


Algorithme d'addition

Soit 2 nombres X et Y représentés par les chaînes

- $X_{n-1}X_{n-2}...X_0$
- $Y_{n-1}Y_{n-2}...Y_0$

et leur somme S représentée par la chaîne

- $S_{n-1}S_{n-2}...S_0$

et soit $C_{n-1}C_{n-2}...C_0$ les retenues successives

L'algorithme d'addition en base B est le suivant :

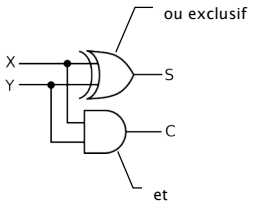
```
C0 = 0
Pour i allant de 0 à n-1 faire
      Wi      = Xi + Yi + Ci
      Ci+1    = 0 si Wi ≤ B - 1
              = 1 sinon
      Si      = Wi - BCi+1
Fait
```


Algorithme d'addition en binaire

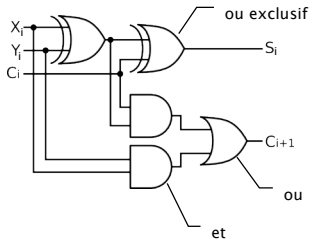
- L'algorithme précédent énonce le cas général
- Dans le cas où B = 2, l'algorithme correspond aux équations :

$$C_0 = 0$$
$$S_i = X_i \oplus Y_i \oplus C_i$$
$$C_{i+1} = X_iY_i \vee X_iC_i \vee Y_iC_i$$

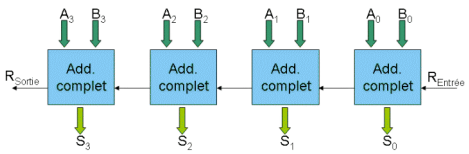
Circuit « Demi-additionneur »



Circuit « Additionneur complet »



Circuit « Additionneur 4 bits »



Les entiers signés (entiers relatifs)

- On veut représenter des nombres positifs et négatifs
  - on utilise le bit de poids fort pour coder le signe
    - 0 : entier positif
    - 1 : entier négatif
- Première solution :
  - le reste des bits est utilisé de la même manière que précédemment pour coder la valeur absolue du nombre
  - ex sur 8 bits :
    - $66 = 01000010_b$ / $-66 = 11000010_b$
  - Problèmes
    - 0 a deux représentations différentes (+0 et -0)
      - 00000000_b (+0) et 10000000_b (-0)
    - l'addition classique ne fonctionne plus :
      - ex sur 4 bits : 3 + (-4) = 00000011 + 10000100 = 10000111 = (-7)

Notation en complément à 1

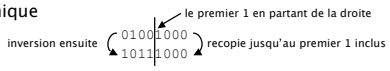
- Principe
  - les entiers positifs se représentent
    - avec le bit de poids fort à 0
    - en codant l'entier sur les n-1 bits de poids faible
  - les entiers négatifs se représentent
    - en inversant tous les bits de l'entier positif correspondant
  - exemple sur 4 bits
    - $5 = 0101_b$
    - $-5 = 1010_b$
- Propriétés
  - l'addition usuelle fonctionne
  - on a toujours 2 représentations pour le 0
    - 00000000_b et 11111111_b

Notation en complément à deux

- Constat
  - On a vu que sur 16 bits :
    - $1111111111111111_b + 1 = 0$
    - 1111111111111111_b doit donc être la représentation de (-1)
- Généralisation
  - sur n bits
    - les entiers positifs se représentent
      - avec le bit de poids fort à 0
      - en codant l'entier sur les n-1 bits de poids faible
    - Soit X_b, la représentation binaire d'un entier positif
      - la représentation Y_b de l'opposé est telle que $X_b + Y_b = 0$
      - en réalité $X_b + Y_b = 2^n$, donc en notant sur n bits, 0 et une retenue

Calcul du complément à deux

- 1ère technique
  - puisque $X_b + Y_b = 2^n$
  - alors $Y_b = 2^n - X_b$
  - ex sur 8 bits :
    - $-1 = 256 - 1 = 255 = 11111111_b$
- 2ème technique
  - on inverse les bits de X_b (complément à un)
    - (opération NOT sur chaque bit)
  - on ajoute 1 au résultat (on ignore les dépassements)
  - ex sur 8 bits :
    - 01001000 –*(NOT)*–> 10110111 –*(+1)*–> 10111000
- 3ème technique



Remarques sur le complément à deux

- Un seul zéro
  - le complément à 2 de 00000000_b est 00000000_b
- L'addition usuelle fonctionne
  - $4 = 00000100_b$
  - $-4 = 11111100_b$
  - $3 + (-4) = 11111111_b = -1$
- Le résultat de l'addition usuelle de nombres représentés en complément à deux est le codage en complément à deux du résultat de l'addition des nombres
- Si y est le complément à deux de x, alors x est le complément à 2 de y
  - $2^n - (2^n - x) = x$

Capacités de représentation des entiers (en langage C)

Type de donnée	Signification	Taille (en octets	Plage de valeurs acceptées
char	caractère	1	–128 à 127
unsigned char	caractère non signé	1	0 à 255
short int	entier court	2	–32768 à 32767
unsigned short int	entier court non signé	2	0 à 65535
int	entier	2 (sur processeur 16 bits) 4 (sur processeur 32 bits)	–32768 à 32767 –2 147 483 648 à 2 147 483 647
unsigned int	entier non signé	2 (sur processeur 16 bits) 4 (sur processeur 32 bits)	0 à 65535 0 à 4 294 967 295
long int	entier long	4	–2 147 483 648 à 2 147 483 647
unsigned long int	entier long non signé	4	0 à 4 294 967 295

Les nombres flottants (réels)

- Nombres à virgule flottante
  - de la forme $x = s \cdot m \cdot \beta^e$
    - $s \in \{-1, 1\}$: signe
    - m : mantisse (ou significande)
      - comprend au plus p chiffres (la précision) en base β
    - β : base (généralement 2, 10 ou 16)
    - e : exposant
      - compris entre 2 bornes $e_{min} \leq e \leq e_{max}$
  - exemple : le nombre 3.1416 peut s'écrire
    - $0.3416 \cdot 10^1$ ou $31416 \cdot 10^{-4}$
    - dépend de la convention choisie pour l'écriture de la mantisse
      - virgule avant ou après le premier chiffre non nul, après le $p^{ème}$ chiffre de la mantisse (mantisse entière)

La norme IEEE 754 (1985)

- Définit 4 formats de flottants en base 2
  - simple précision (correspond en général au type `float` en C)
  - simple précision étendu (obsolète)
  - double précision (correspond en général au type `double` en C)
  - double précision étendu (extended)
- Avec une mantisse $1 \leq m < 2$:

format	taille	précision	e_{min}	e_{max}	valeur max.
simple	32	23+1 bits	−126	+127	$3.403...10^{38}$
double	64	52+1 bits	−1022	+1023	$1.798...10^{308}$
extended	≥ 79	≥ 64	≤ −16382	≥ 16383	$1.190...10^{4932}$

  - bit implicite
  - valeur = $(-1)^S * M * 2^{(E - emax)}$
  - 3 valeurs spéciales : NaN (Not a Number), $+\infty$, $-\infty$
  - implémentation matérielle des calculs sur flottants IEEE 754

La norme IEEE 754 : précautions d'emploi

- Problèmes
  - précision limitée
    - entraîne des arrondis qui peuvent devenir gênants
    - ex. : soustraction de 2 nombres très proches -> perte de précision relative (cancellation)
  - plage d'exposants limitée :
    - overflows : si le résultat d'une opération est plus grand que la plus grande valeur représentable
    - underflows : si le résultat est plus petit, en valeur absolue, que le plus petit flottant normalisé positif
  - ex. :
    - $(2^{60} + 1) - 2^{60} = 0$

Le codage des caractères

- Caractère =
  - lettre (a, b, ..., A, B, ...)
  - chiffre (0, 1, ..., 9) ou symbole arithmétique (+, _, *, /, =, <, >)
  - ponctuation (, ; : . ! ? ' ")
  - symbole divers ($ % @ # &)
- La nome ASCII
  - American Standard Code for Information Interchange
  - inventée en 1961 par Bob Bemer
  - contient les caractères nécessaires pour écrire en anglais
  - base d'autres normes qui l'étendent (Unicode, ISO 8859-1, etc.)
  - principe
    - 128 caractères numérotés de 0 à 127
    - codés en binaire sur 7 bits de 0000000 à 1111111
    - 0 à 31 : directives de terminal (non affichables)

La table ASCII

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI	8
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US	9
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	−	.	/	A
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	B
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	C
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	D
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	E
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL	F

Sources

- *Calcul et arithmétique des ordinateurs*, sous la direction de J.-C. Bajard et J.-M. Muller, Hermès science, collection IC2, Lavoisier, Paris, 2004
- http://fr.wikipedia.org
- Internet et le Web