Architecture des ordinateurs

Licence Informatique - Université de Provence

Jean-Marc Talbot

jtalbot@cmi.univ-mrs.fr

A propos du cours

- 20 heures de Cours, 20 heures de TD, 20 heures de TP
- TD et TP commencent la semaine prochaine (6 octobre)
- Site du Cours

  http ://www.cmi.univ-mrs.fr/˜ jtalbot/Teaching/Archi

En TP :

- TkGate, un logiciel de conception de circuit
- Assembleur MIPS (SPIM, ...)

Evaluation

- 1 partiel = CC
- 1 projet en TP = TP
- 1 examen final en janvier = E

Note UE (première session) : $\frac{max(3E, 2E + CC) + TP}{4}$

Seconde session :
On remplace E par E', la note d'examen de seconde session

Evaluation : Projet

- Le projet se réalise au plus en binôme
- Il faut rendre un rapport en plus des sources
- La présence à la soutenance est indispensable
- Les 2 dernières séances de TP sont consacrées au projet

Les projets ne sont pas optionnels
Absence à la soutenance = DEF à l'UE = pas de L3

Plan du cours

1. Systèmes de numération - Codage de l'information
2. Algèbre de Boole - Fonctions booléennes - Circuits combinatoires
3. Circuits séquentiels
4. Mémoires
5. Machines de Moore - Machines de Mealy - Machines synchrones - Microprogrammation
6. Programmation d'un processeur - Assembleur
7. Fonctionnement d'un processeur MIPS
8. Exemples d'autres processeurs - Bus
9. Optimisation : Pipeline - Mémoire cache

Bibliographie

- A. Tanenbaum. "*Architecture de l'ordinateur*" (4ième édition) InterEdition.
- J.-J. Schwarz "*Architecture des ordinateurs*" Eyrolles
- W. Stallings "*Organisation et architecture de l'ordinateur*", Pearson Education
- D. Patterson J. Hennessy "*Organisation et conception des ordinateurs*" Dunod.
- ..

Objectifs du cours (I)

Fonctionnement d'un ordinateur

Applications	gcc, emacs, javac, ...
Systeme d'exploitation	linux/unix, windows, ...
Materiel	PC, MAC

Traduction et sémantique (compilation)
+
Architecture des ordinateurs
+
Systèmes d'exploitation

= du programme à son exécution .....

Objectifs du cours (II)

Part de l'informatique embarquée/enfouie en forte augmentation ! !

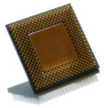
- téléphone portable
- Box Adsl
- carte à puce
- ....

programmation avec des contraintes d'espace, de ressources, temps-réel liées au **matériel**

Objectifs du cours (III)

On va s'intéresser à ce qu'on trouve sur la carte mère 

et principalement

- au processeur 
- à la mémoire 

Systèmes de numération

La base 10 (I)

10 chiffres : $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$

$$145 = 1 * 10^2 + 4 * 10^1 + 5 * 10^0$$

$$\begin{array}{rcl} 1 & = & 1 \\ 14 & = & 1*10+4 \\ 145 & = & (1*10+4) * 10+5 \end{array}$$

Pour bien préciser qu'il s'agit d'un nombre en base 10

$$(145)_{10}$$

La base 10 (II)

Partie "fractionnaire"

$$0,329 = 3 * 10^{-1} + 2 * 10^{-2} + 9 * 10^{-3}$$

$$\begin{array}{rcl} 0,9 & = & 9 * 10^{-1} \\ 0,29 & = & ((9 * 10^{-1}) + 2) * 10^{-1} \\ 0,329 & = & (((9 * 10^{-1}) + 2) * 10^{-1} + 9) * 10^{-1} \end{array}$$

Autres bases

Quelques bases B utiles :

- $B = 2$ binaire
  - chiffres : 0,1
- $B = 8$ octal
  - chiffres : 0,1,2,3,4,5,6,7
- $B = 16$ hexadécimal
  - chiffres : 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Nombres représentés

Nombre représenté en base B :

$$\underbrace{d_n d_{n-1} \ldots d_2 d_1 d_0}_{\text{partie entière}} , \underbrace{d_{-1} d_{-2} \ldots d_{-m}}_{\text{partie fractionnaire}}$$

- par position :

$$\sum_{i=-m}^{n} d_i * B^i$$

- par multiplications successives :

$$\begin{array}{c} ((((d_n * B + d_{n-1}) * B + \ldots + d_2) * B + d_1) * B + d_0) \\ + \\ ((((d_{-m} * \frac{1}{B} + d_{-m+1}) * \frac{1}{B} + \ldots + d_{-2}) * \frac{1}{B} + d_{-1}) * \frac{1}{B} \end{array}$$

Exemple

Nombre représenté en base 4 :

$$(301, 23)_4$$

- par position :

$$3 * 4^2 + 0 * 4^1 + 1 * 4^0 + 2 * 4^{-1} + 3 * 4^{-2} = (49, 6875)_{10}$$

- par multiplications/divisions successives :

$$\underbrace{((3 * 4 + 0) * 4 + 1)}_{49} + \underbrace{(\underbrace{(3 * 1/4)}_{0.75} + 2) * 1/4}_{0,6875}$$

D'une base à une autre

- de la base B vers la base 10 : le calcul du nombre représenté donne un algorithme
- de la base 10 vers la base B : divisions et multiplications successives

$$(35, 25)_{10} \rightarrow (100011, 01)_2$$

Partie entière					Partie fractionnaire				
35	/ 2	= 17	reste	1					
17	/ 2	= 8	reste	1					
8	/ 2	= 4	reste	0	0,25	* 2	=	0,5	
4	/ 2	= 2	reste	0	0,5	* 2	=	1,0	
2	/ 2	= 1	reste	0					
1	/ 2	= 0	reste	1					

D'une base à une autre : pourquoi ca marche ?

$(d_n d_{n-1} \ldots d_2 d_1 d_0)_B$ vaut

$$((((d_n * B + d_{n-1}) * B + \ldots + d_2) * B + d_1) * B + d_0)$$

On a

$$\frac{((((d_n * B + d_{n-1}) * B + \ldots + d_2) * B + d_1) * B + d_0)}{B}$$

égal à

$$(((d_n * B + d_{n-1}) * B + \ldots + d_2) * B + d_1) \quad \text{reste} \quad d_0$$

car $0 \leq d_0 < B$. On continue récursivement ....

de même pour la partie fractionnaire (avec des multiplications) ...

Base 2/ base 16

- De la base 2 vers la base 16 :
  - on regroupe les chiffres par 4 en partant de la virgule
  - chaque groupe de 4 bits représente un chiffre hexadécimal

$$(\underbrace{101}_{B}\underbrace{1100}_{8}\underbrace{0011}_{3})_2$$

- De la base 16 vers la base 2 : on convertit chaque chiffre hexadécimal en un nombre binaire de 4 chiffres équivalent.

Codage de l'information

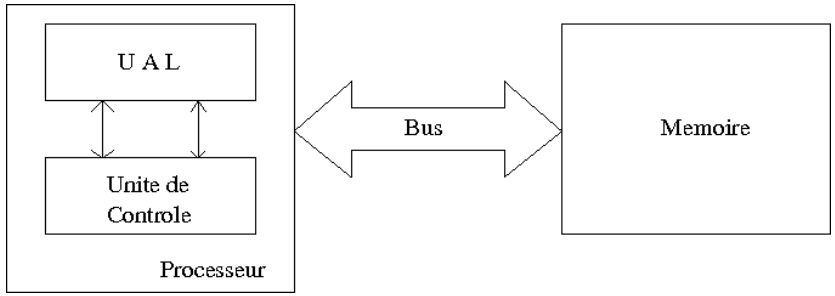
Qu'est ce qu'un ordinateur ?

Modèle de von Neumann :

modèle proposé par John von Neumann (1903-1957) en 1944 qui définit un ordinateur en 3 parties :

- **le processeur** composé
  - d'une **ALU**, unité arithmétique et logique (opérations sur les données)
  - d'une **unité de contrôle** (traitement des instructions à exécuter)
- **la mémoire** (stockant à la fois les données et les instruction à exécuter)
- **le bus** assurant la liaison entre la mémoire et le processeur

Modèle de von Neumann



Instructions/données

- Différence entre instruction et donnée en mémoire ?

  Il n'y en a pas ; c'est juste de l'**information**.

  Le **bit** (**bi**nary digi**t**) est la plus petite quantité d'information ; il prend 2 valeurs, 0 ou 1.

- Dans un ordinateur, tout n'est que 0 et 1
  - 1 (du courant)
  - 0 (pas de courant)

- Souvent l'information est stockée sur des (suites de) mots binaires d'une taille fixée (par le processeur, la mémoire, ...) :
  ex. 8 bits, 16, bits, ...

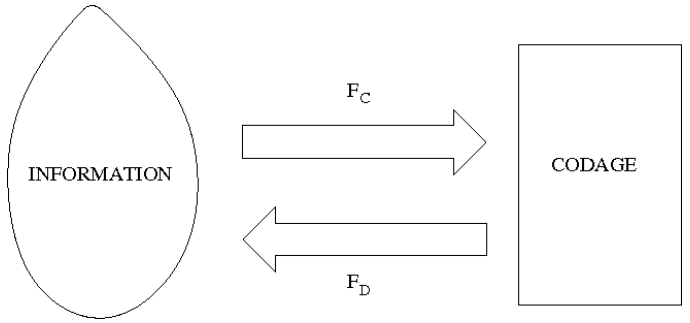
Codage de l'information

Le **codage de l'information** est la représentation d'informations diverses (nombres entiers, nombres flottants, caractères, chaînes de caractères, images, ...) sous forme binaire, c'est-à-dire sous la forme d'une suite de bits.

Les points importants sont :

- Le passage de l'information à son codage
- Le passage du codage à l'information

Codage de l'information



En général, $F_D = F_C^{-1}$

Mais pas toujours ....

Caractères ASCII

L'**ASCII** (American Standard Code for Information Interchange), créé en 1961, associe un nombre à chaque caractère. Les caractères sont codés sur 7 bits. Il y a donc 128 caractères différents (de 00 à 7F en hexadécimal).

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

Caractères ASCII étendu

Le code ASCII est anglophone : beaucoup de caractères présents dans les alphabets français, nordique, ... en sont absents. Il existe des extensions (sur 8 bits, de 80 à FF). Cependant, ces extensions ne sont pas standardisées.

128	Ç	144	É	160	á	176	▒	193	┴	209	╤	225	ß	241	±
129	ü	145	æ	161	í	177	▓	194	┬	210	╥	226	Γ	242	≥
130	é	146	Æ	162	ó	178	│	195	├	211	╙	227	π	243	≤
131	â	147	ô	163	ú	179	┤	196	─	212	╘	228	Σ	244	⌠
132	ä	148	ö	164	ñ	180	┼	197	┼	213	╒	229	σ	245	⌡
133	à	149	ò	165	Ñ	181	╡	198	╞	214	╓	230	µ	246	÷
134	å	150	û	166	ª	182	╢	199	╟	215	╫	231	τ	247	≈
135	ç	151	ù	167	º	183	╖	200	╚	216	╪	232	Φ	248	°
136	ê	152	_	168	¿	184	╕	201	╔	217	┘	233	⊛	249	·
137	ë	153	Ö	169	_	185	╣	202	╩	218	┌	234	Ω	250	·
138	è	154	Ü	170	¬	186	║	203	╦	219	▉	235	δ	251	√
139	ï	156	£	171	½	187	╗	204	╠	220	▄	236	∞	252	_
140	î	157	¥	172	¼	188	╝	205	═	221	▌	237	ϕ	253	²
141	ì	158	_	173	¡	189	╜	206	╬	222	▐	238	ε	254	■
142	Ä	159	ƒ	174	«	190	╛	207	╧	223	▀	239	∩	255	
143	Å	192	└	175	»	191	┐	208	╨	224	α	240	≡		

Caractères ISO-8859-1

iso-8859-1										
+	0	1	2	3	4	5	6	7	8	9
160		¡	¢	£	¤	¥	¦	§	¨	©
170	ª	«	¬	-	®	¯	°	±	²	³
180	´	µ	¶	·	¸	¹	º	»	¼	½
190	¾	¿	À	Á	Â	Ã	Ä	Å	Æ	Ç
200	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ
210	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û
220	Ü	Ý	Þ	ß	à	á	â	ã	ä	å
230	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù
250	ú	û	ü	ý	þ	ÿ				

Caractères sur 16 bits : Unicode

Unicode propose de coder les caractères sur 16 bits (65536 valeurs possibles - $(FFFF)_{16}$).

Il permet de coder notamment les alphabets latins, cyrilliques, les caractères chinois, indiens, arabes.

Unicode code point	character	UTF-8 (hex.)	Unicode code point	character	UTF-8 (hex.)
U+0400	Ѐ	d0 80	U+3130	ㄱ	e3 84 b0
U+0401	Ё	d0 81	U+3131	ㄱ	e3 84 b1
U+0402	Ђ	d0 82	U+3132	ㄲ	e3 84 b2
U+0403	Ѓ	d0 83	U+3133	ㄳ	e3 84 b3
U+0404	Є	d0 84	U+3134	ㄴ	e3 84 b4
U+0405	S	d0 85	U+3135	ㄵ	e3 84 b5
U+0406	I	d0 86	U+3136	ㄶ	e3 84 b6
U+0407	Ї	d0 87	U+3137	ㄷ	e3 84 b7
U+0408	J	d0 88	U+3138	ㄸ	e3 84 b8
U+0409	Љ	d0 89	U+3139	ㄹ	e3 84 b9
U+040A	Њ	d0 8a	U+313A	ㄺ	e3 84 ba
U+040B	Ћ	d0 8b	U+313B	ㄻ	e3 84 bb
U+040C	Ќ	d0 8c	U+313C	ㄼ	e3 84 bc

Entiers binaires positifs (non signés)

On utilise simplement la représentation du nombre en base 2

Sur n bits, on code les naturels (entiers non signés) de 0 à $2^n - 1$.

Addition

$$\begin{array}{cccccc} & {\scriptstyle 1} & & & & \\ & 0 & 1 & 1 & 0 & \\ + & 0 & 1 & 0 & 1 & \\ \hline = & 1 & 0 & 1 & 1 & \end{array}$$

Attention au débordement (sur un nombre fixé de bits) :

$$0110 + 1011 = 10001$$

Entiers binaires positifs et négatifs (signés)

Entiers signés

- représentation valeur absolue et signe
- représentation par excès
- représentation à complément 1
- représentation à complément 2

On s'intéresse aux nombres codés sur n bits (n fixé)

Rappel : avec n bits, on code 2^n valeurs différentes

Valeur absolue et signe

- on utilise le bit le plus à gauche pour représenter le signe :
- $0 =$ "$+$" $\quad 1 =$ "$-$"
- sur 4 bits :

$$0100 \Leftrightarrow 4 \qquad 1100 \Leftrightarrow -4$$

Inconvénients :

- 2 représentations pour 0 : (sur 4 bits) 0000 et 1000
- la somme (binaire) est incorrect

$$\begin{array}{cccccc} & 0 & 1 & 0 & 0 & \\ + & 1 & 0 & 1 & 1 & \\ \hline = & 1 & 1 & 1 & 1 & \end{array} \qquad \begin{array}{cc} & 4 \\ + & -3 \\ \hline = & -7 \end{array}$$

$\Rightarrow$ l'addition est un opération plus complexe

Représentation par excès

On code le nombre en décalant les valeurs d'un biais *biais* (les valeurs seront donc en excès)

La suite $b_{n-1} \ldots b_1 b_0$ représente le nombre $(b_{n-1} \ldots b_1 b_0)_2 - biais$.

Exemple : sur 4 bits avec $biais = 7$

$$1011 \Leftrightarrow 4 \qquad 0011 \Leftrightarrow -4 \qquad 0111 \Leftrightarrow 0$$

Inconvénients :

- addition binaire ne fonctionne pas
- selon le biais, l'inversion peut être difficile

Complément à 1 (I)

- Si i un nombre positif binaire, alors son bit le plus à gauche vaut 0. Pour calculer $-i$, on inverse tous les bits de i (le bit le plus à gauche étant alors 1)
- sur 4 bits :

$$0011 \Leftrightarrow 3 \qquad 1100 \Leftrightarrow -3$$

- sur 8 bits :

$$00000011 \Leftrightarrow 3 \qquad 11111100 \Leftrightarrow -3$$

Complément à 1 (II)

Addition

- On ajoute les deux nombres binaires et on ajoute la retenue éventuelle
-

$$\begin{array}{cccccc} & 0 & 0 & 0 & 1 \\ + & 1 & 1 & 1 & 0 \\ \hline 0 & 1 & 1 & 1 & 1 \\ = & 1 & 1 & 1 & 1 \end{array} \qquad \begin{array}{cc} & 1 \\ + & -1 \\ \hline & \\ = & -0 \end{array}$$

$$\begin{array}{cccccc} & & 1 & & & \\ & 0 & 1 & 0 & 0 \\ + & 1 & 1 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 \\ & & & & 1 \\ = & 0 & 0 & 0 & 1 \end{array} \qquad \begin{array}{cc} & 4 \\ + & -3 \\ \hline & \\ = & 1 \end{array}$$

Inconvénients

- 2 représentations pour 0 : (sur 4 bits) 0000 et 1111

Complément à 2 (I)

- Si i un nombre positif binaire, alors son bit le plus à gauche vaut 0. Pour calculer $-i$, on inverse tous les bits de i et on ajoute 1 (le bit le plus à gauche étant alors 1).
- sur 4 bits

$$0011 \Leftrightarrow 3 \qquad 1101 \Leftrightarrow -3$$

-
  - permet de représenter sur n bits les entiers compris entre -2^{n-1} et $2^{n-1} - 1$
  - la valeur en entier codé en complément à 2 de la suite de bits $b_m b_{m-1} \ldots b_1 b_0$ est

$$(\sum_{i=0}^{m-1} b_i 2^i) - b_m 2^m$$

Complément à 2 (II)

- représentation de 0 (sur 4 bits) : 0000

  complément à 1 : 1111 on ajoute 1 : 10000

  On “oublie” ce qui déborde ....

- Addition : on ajoute les deux nombres et on omet la retenue finale éventuelle

$$\begin{array}{cccccc} 1 & 1 & 1 & 1 & & \\ & 0 & 0 & 0 & 1 \\ + & 1 & 1 & 1 & 1 \\ \hline 1 & 0 & 0 & 0 & 0 \end{array} \qquad \begin{array}{cc} & 1 \\ + & -1 \\ \hline = & 0 \end{array}$$

$$\begin{array}{cccccc} & 1 & 1 & & & \\ & 0 & 1 & 0 & 0 \\ + & 1 & 1 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & 1 \end{array} \qquad \begin{array}{cc} & 4 \\ + & -3 \\ \hline = & 1 \end{array}$$

Dépassement de capacité

La somme de 2 nombres (de n bits) ne peut être codé sur n bits (la somme est parfois trop grande ou trop petite)

- en complément à 2

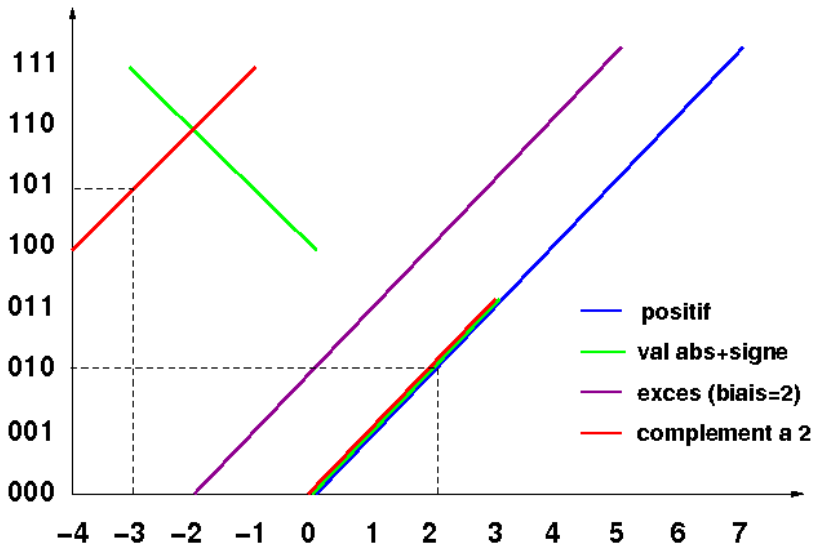
$$\begin{array}{rcccc} & 0 & 1 & 1 & 1 \\ + & 0 & 0 & 1 & 1 \\ \hline & 1 & 0 & 1 & 0 \end{array} \qquad \begin{array}{rc} & 7 \\ + & 3 \\ \hline = & -6 \end{array}$$

Le résultat est bien-sûr incorrect.

En complément à 2,

> *Un **dépassement de capacité** se produit lorsque les deux opérandes ont le même signe et le résultat a un signe différent des opérandes.*

Entiers binaires signés : résumé



Nombres à virgules

Attention ceci n'a rien à voir avec des réels !

- Nombres à virgule fixe
- Nombres à virgule flottante

Nombres à virgule fixe

On utilise les nombres fractionnaires comme vus précédemment.

La position de la virgule est déterminée de manière fixe dans la représentation du nombre.

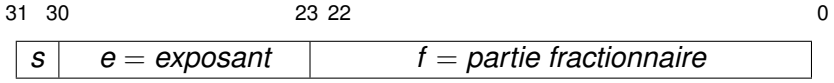
$$b_{n-1} \ldots b_2 \, , \, b_1 b_0$$

Ici, 2 chiffres après la virgule.

Inconvénients

- impossible de représenter à la fois des nombres très petits et des très grands.
- Imprécision dans les calculs pouvant être importante.

Flottants IEEE 754



$$r = (-1)^s.(1 + \sum_{i=0}^{22} f_i.2^{-i}).2^{E-127}$$

où $E = (e_7 \dots e_0)_2$

.... la suite en TD