Architecture des ordinateurs - CM1

Pierre Nerzic

automne 2023

Introduction

Pierre Nerzic

Objectifs du cours

Comprendre :

- le fonctionnement général d'un ordinateur
  - architecture de Von Neumann
  - environnement matériel simple
- la mise en œuvre des concepts des langages de programmation
  - structures de données : variables, tableaux, structures, pointeurs
  - structures de contrôle : affectations, conditionnelles, boucles

Pour répondre un peu à ces questions :

- Comment fonctionne un ordinateur ?
- Comment sont exécutés les programmes en langage de haut niveau (C, Java, etc) ?

Intérêts

Nécessité de concevoir des programmes économes en ressources avec des algorithmes efficaces (voir la matière appelée « complexité algorithmique »)

- temps de calcul = consommation d'énergie
- occupation mémoire (RAM et disque) inutile = gaspillage de ressources

Savoir comment ça s'implémente (met en œuvre) sur un ordinateur permet de mieux comparer les algorithmes et choisir celui qui gaspille le moins.

Plan du cours

- Représentation des informations simples
  - entiers naturels, entiers relatifs
  - textes (encodages)
- Constitution d'un ordinateur
  - composants d'un ordinateur
  - architecture de l'unité centrale
- Programmation d'une unité centrale
  - jeu d'instruction
  - opérandes et modes d'adressage
- Traduction d'un programme de haut niveau vers bas niveau
  - implantation et accès aux structures de données
  - réalisation des structures de contrôle

Déroulement

Enseignements :

- 3 CM de 1h au début
- 8 TD de 1h
- 3 TP de 2h à la fin

Évaluation :

- 1 DS de 1h (deb décembre), 1 DS de 2h (fin janvier)
- Les TP seront à rendre en fin de séance (contrôle continu)

Pierre Nerzic

Remarques

Une rumeur prétend que cette matière est difficile.

C'est faux.

C'est une matière assez technique, mais tout à fait compréhensible.

- Au début, il y a des maths simples : de l'arithmétique
- Ensuite, il y a quelques concepts techniques (instructions, registres, etc) mais on travaille de manière rationnelle sur des concepts scientifiques. Ça n'a rien à voir avec la « bidouille ».
- Pour la programmation, on applique des *schémas de traduction* systématiques (ex: traduction d'une conditionnelle en instructions pour l'unité centrale).

Il est important d'être présent et actif dans ce cours.

Codage des informations

Pierre Nerzic

Introduction

On commence immédiatement par un concept général en informatique, la *représentation des informations* dans une machine. On appelle ça le « codage de l'information ».

Coder = représenter une information sans ambiguïté par une autre information (code). Dans un ordinateur, les codes ne peuvent être que numériques, des entiers naturels. On les appelle *octets*, car, quand ils sont écrits en base 2, ils sont composés de 8 chiffres.

1 octet représente un nombre entier compris entre 0 et 255. C'est peu pour un code. Ça suffit pour représenter un caractère comme `A`, `%`, `6`, `t`, etc. Il faut donc juxtaposer de nombreux octets pour coder un texte.

Mais comment coder le reste ? images, musique, données, etc. ?

Pierre Nerzic

Principes

Tout document peut être codé de manière numérique :

- Image : tableau de pixels, chacun représente une couleur, chaque couleur = n-uplet de nombres, ex: RVB, HSV, ...
- Audio : suite d'échantillons, chacun représente une « force sonore » codée par un nombre
- Texte : suite de caractères, chacun est représenté par un numéro dans une liste (Ascii, Latin1, UTF-8...)
- etc etc.

Coder = écrire un document sous forme de nombres
Décoder = retrouver le document d'origine à partir des nombres

Ne pas confondre avec crypter/décrypter = dissimuler l'information.

Le codage de documents complexes est hors du périmètre de ce cours. Il faut des années d'études pour comprendre certains.

Principes, suite

La magie de l'informatique :

- Document codé = séquence de nombres
- Traitement = sorte de « calcul » sur ces nombres = algorithme
- Ordinateur = appareil qui fait ces calculs = qui applique l'algorithme sur les nombres
- Programme, logiciel = spécification de ces calculs = écriture de l'algorithme sous une forme utilisable par l'ordinateur (sous forme de nombres également !)

Donc, il faut comprendre au moins un peu ce qu'est un codage, en commençant par les plus simples, et ensuite comprendre les « calculs » qu'on peut faire sur un document (toute votre vie professionnelle).

Plan de la suite de ce cours

- Ce cours = CM n°1
  1. Codage des entiers naturels : 0, 1, 2... en base 10
  2. Codage des entiers naturels dans d'autres bases, 2 et 16
  3. Codage des entiers relatifs
  4. Codage des textes simples
- Le CM n°2 portera sur le fonctionnement d'une unité centrale (le cœur d'un ordinateur).
- Le CM n°3 montrera comment traduire les structures de données et de contrôle d'un langage de haut niveau en instructions pour l'unité centrale.

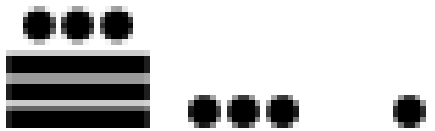
Codage des entiers naturels

Pierre Nerzic

Autres systèmes de notation

Nous sommes totalement habitués à lire des nombres écrits avec les 10 chiffres, en base 10 (système indo-arabe), et à nous représenter leur valeur, par exemple pour les comparer, pour avoir un ordre de grandeur, etc.

Pourtant, il faut distinguer la valeur intrinsèque d'un nombre, de son écriture dans le système décimal. P. ex. « 6541 » est à la fois une quantité et une liste de 4 chiffres.

Voici 6541 écrit en Maya :  (www.dcode.fr)

Voici 6541 écrit en égyptien antique :

Entiers naturels en base 10

Nombre écrit dans le système indo-arabe = suite ordonnée de chiffres allant de 0 à 9

- ex : 87921 : quatre vingt sept mille neuf cent vingt et un

Décomposition en puissances de 10 : séparer un nombre sous la forme de la somme de ses chiffres multipliés par des puissances de 10

$$\begin{aligned} 123 &= 100 + 20 + 3 \\ &= 1 \times 100 + 2 \times 10 + 3 \\ &= 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 \end{aligned}$$

$$\begin{aligned} 6541 &= 6000 + 500 + 40 + 1 \\ &= 6 \times 10^3 + 5 \times 10^2 + 4 \times 10^1 + 1 \times 10^0 \end{aligned}$$

Généralisation à une base B quelconque

Soit un nombre entier naturel N. Il peut être écrit sous la forme de la suite de chiffres $(a_n a_{n-1} a_{n-2} ... a_1 a_0)_B$ telle que

$N = (a_n.B^n) + (a_{n-1}.B^{n-1}) + (a_{n-2}.B^{n-2}) + ... + (a_1.B^1) + (a_0.B^0)$
et $\forall i, 0 \leq a_i < B$

- B est appelée la base, ex : B=10
- Les B_i sont appelés « poids », B_0=poids faible, B_n=poids fort et a_0 = chiffre de poids faible, a_n = chiffre de poids fort

Si l'indice B est absent, ex : 2671, alors c'est un nombre en base 10. On le met pour des bases $\neq 10$, ex: $(2671)_{16}$ est en base 16.

Remarque : dans sa propre base, B s'écrit $(10)_B$.

Entiers naturels en base 2

Pourquoi la base 2 ?

Parce qu'en électronique, il est très simple de faire des circuits à deux états possibles : transistors conducteurs (*passants*) ou isolants (*bloqués*). Les transistors sont des interrupteurs commandés par des tensions électriques. Il y a seulement deux tensions électriques utiles : 0 volts et « VCC » (5 volts, 3.3 volts ou moins).

Un transistor peut commander l'état de plusieurs autres transistors pour rendre certains passants, ou en bloquer d'autres.

Tout est donc lié à ces deux états sur des ensembles de transistors, et donc tout est représenté par des nombres en base 2.

En base 2, les chiffres sont 0 et 1 seulement, et sont appelés « bits » (*binary units* et aussi *un peu*)

D'autres bases de codage ?

D'autres formes d'ordinateurs, non binaires, existent :

- https://the-analog-thing.org/ présente un ordinateur analogique, où les nombres sont représentés par des tensions électriques totalement libres. Ses éléments sont capables de faire des sommes, des multiplications, des comparaisons, des intégrations, pour écrire des traitements mathématiques.
- Les ordinateurs quantiques représentent l'information avec des formes linéaires, ex : $a.v_0 + b.v_1$ avec $|a^2|$ et $|b^2|$ étant les probabilités que l'information soit respectivement v_0 et v_1. Les algorithmes consistent à faire transiter les informations à travers des fonctions de calcul correspondant aux lois de la physique quantique (superposition d'état, intrication et autres), pour connaître le résultat voulu.

Entiers naturels en base 2, suite

On revient donc à la base 2 où on peut écrire des entiers avec les chiffres 0 et 1.

Exemples de décomposition en puissances de 2 :

$$\begin{aligned}(101)_2 &= (100)_2 + (00)_2 + (1)_2 \\ &= 1 \times B^2 + 0 \times B^1 + 1 \times B^0 \quad \textit{avec } B = 2 \\ &= (1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0)_{10} \\ &= 4_{10} + 0_{10} + 1_{10} \quad = 5_{10}\end{aligned}$$

$$\begin{aligned}(11011)_2 &= (10000)_2 + (1000)_2 + (000)_2 + (10)_2 + (1)_2 \\ &= 16_{10} + 8_{10} + 0_{10} + 2_{10} + 1_{10} \quad = 27_{10}\end{aligned}$$

Utile : connaître 1, 2, 4, 8, 16, 32, 64, 128, 256...

Calculs en base 2

Comment fait-on une addition en base 2 ?

Ex : $(11011)_2 + (01110)_2 = (\ ?\ )_2$

- Idée 1 : traduire les deux nombres en base 10, calculer puis revenir en base 2... pas ouf
- Idée 2 : additionner comme en base 10 (chiffre par chiffre avec report des retenues), mais avec une table d'addition en base 2 :

addition sans retenue d'entrée

+	0	1
0	0	1
1	1	0*

addition avec retenue d'entrée

+	0	1
0	1	0*
1	0*	1*

0* et 1* signifient qu'il y a une retenue à reporter sur les chiffres suivants. Ex: à gauche, 1+1 donne 0 et une retenue à reporter.

Entiers naturels en base 16

Pourquoi la base 16 ? Parce que c'est pratique pour écrire des grands nombres au lieu de la base 2.

- On groupe 4 par 4 les chiffres d'un nombre écrit en base 2, en commençant à droite, en rajoutant des zéros à gauche !
  - Ex : $(101101010101001)_2 = (1011)_2, (0101)_2, (0110)_2, (1001)_2$
  - Ex : $(111100111010)_2 = (0001)_2, (1111)_2, (0011)_2, (1010)_2$
- Chacun de ces quadruplets représente un entier entre 0 et 15
- On peut associer chacun à un chiffre de la base 16 :
  - 0, 1, 2..., 8, 9, A, B, C, D, E, F
    $A_{16} = 10_{10} = 1010_2$, $B_{16} = 11_{10} = 1011_2$, $C_{16} = 12_{10} = 1100_2$,
    $D_{16} = 13_{10} = 1101_2$, $E_{16} = 14_{10} = 1110_2$, $F_{16} = 15_{10} = 1111_2$
  - Ainsi, $(101101010101001)_2 = (B569)_{16}$
  - Ainsi, $(111100111010)_2 = (1F3A)_{16}$

C'est plus compact et source de moins d'erreurs.

Passage d'une base à l'autre

Soit un nombre N écrit dans une base B
$N_B = (a_n a_{n-1} a_{n-2} ... a_2 a_1 a_0)_B$

On a $N = (a_n.B^n) + (a_{n-1}.B^{n-1}) + ... + (a_2.B^2) + (a_1.B^1) + (a_0.B^0)$

On veut écrire N dans une base B'.

- On doit trouver les chiffres a'_n, a'_{n-1} ... a'_1, a'_0 tels que le même entier :

$$N = (a'_n.B'^n) + (a'_{n-1}.B'^{n-1}) + ... + (a'_2.B'^2) + (a'_1.B'^1) + (a'_0.B'^0)$$

- Directement, c'est compliqué. Donc on procède en deux temps :
  1. écrire N_B en base 10 : N_{10}
  2. écrire N_{10} dans la base B' : $N_{B'}$

De la base B à la base 10

Soit N_B écrit en base B, comment s'écrit-il en base 10 ?

On calcule simplement
$N = (a_n.B^n) + (a_{n-1}.B^{n-1}) + ... + (a_2.B^2) + (a_1.B^1) + (a_0.B^0)$
avec les a_i et B^i exprimés en base 10

Exemples

$$\begin{aligned}(110101)_2 &= 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 1 \times 32 + 1 \times 16 + 0 + 1 \times 4 + 0 + 1 \times 1 \\ &= 53\end{aligned}$$

$$\begin{aligned}(4E6A)_{16} &= 4 \times 16^3 + E \times 16^2 + 6 \times 16^1 + A \times 16^0 \\ &= 4 \times 4096 + 14 \times 256 + 6 \times 16 + 10 \times 1 \\ &= 20074\end{aligned}$$

De la base 10 à la base B

Soit N_{10} écrit en base 10, comment écrire N_B en base B ?

1. Calculer la division entière (euclidienne) N/B en base 10.
2. On obtient un quotient et un reste
   - Le reste est compris entre 0 et $B-1$, ça sera le chiffre a_0 de poids faible du futur nombre N_B
3. Ensuite si le quotient n'est pas nul, alors on repart à l'étape 1 : le nombre N devient le quotient et on le divise par B.
   - Le prochain reste obtenu donnera le chiffre du poids suivant : a_1, puis a_2 au tour suivant, et ainsi de suite

De la base 10 à la base B, exemple 1

Écrire 13_{10} en base 2

On divise 13 par 2 jusqu'à obtenir un quotient nul

1. $13/2 = 6$ reste 1 $\quad (13 = 2 \times 6 + 1)$
2. $6/2 = 3$ reste 0 $\quad (6 = 2 \times 3 + 0)$
3. $3/2 = 1$ reste 1
4. $1/2 = 0$ reste 1

On s'arrête car la division donne un quotient nul.

Le nombre en base 2 se lit dans les restes en remontant : $(1101)_2$

Attention à l'ordre des chiffres !

De la base 10 à la base B, exemple 2

Écrire 980_{10} en base 16

On divise 980 par 16 jusqu'à obtenir un quotient nul (avec une calculatrice ou à la main)

1. $980/16 = 61$ reste 4 $\quad (980 = 16 \times 61 + 4)$
2. $61/16 = 3$ reste 13 $\quad (61 = 16 \times 3 + 13)$
3. $3/16 = 0$ reste 3

Le reste 13 correspond au chiffre D en base 16.

Donc $(980)_{10} = (3D4)_{16}$

Attention à bien écrire les restes sous forme de chiffres en base 16 (de 0 à F).

Pourquoi ça marche-t-il ?

Si on disposait de N déjà écrit dans la base B, il s'écrirait :

$$N = (a_n.B^n) + (a_{n-1}.B^{n-1}) + (a_{n-2}.B^{n-2}) + ... + (a_1.B^1) + (a_0.B^0)$$

Quand on divise cette écriture de N par B, on obtient le quotient Q et le reste R comme ceci :

$$Q = (a_n.B^{n-1}) + (a_{n-1}.B^{n-2}) + (a_{n-2}.B^{n-3}) + ... + (a_1.B^0)$$

$$R = a_0$$

En répétant cette division sur Q et les quotients suivants, on obtient tous les chiffres, dans l'ordre inverse de lecture.

Ça marche de la même manière quand on ne connaît pas l'écriture de N. La division euclidienne par B fournit le chiffre de poids faible.

Cette division est comme un décalage des chiffres vers la droite.

Codage des entiers naturels dans un ordinateur

Dans un ordinateur, les entiers :

- sont codés en binaire (mais on peut les transcrire en base 16 pour plus de lisibilité)
- ne peuvent avoir qu'un nombre fixe de chiffres (*bits*).
  - $n = 8, 16, 24, 32, 48, 64, 128...$ qui définit la valeur maximale représentable : $2^n - 1$
  - Ça dépend de la construction du processeur (processeur 4 bits, 8 bits, 16 bits, 32 bits, 64 bits, etc) et de ses capacités à grouper les octets (8 bits) en *mots* de $n * 8$ bits.
- 8 bits forment un octet (*byte* en anglais, ne pas confondre *byte* et *bit*)
- 16, 24, 32... bits forment un mot (*word* en anglais)

Entiers relatifs

Dans un ordinateur, les nombres ne sont constitués que de chiffres binaires, il n'y a pas de « signe ».

Pour représenter des nombres relatifs (positifs, nul et négatifs), il y a une astuce de codage appelée « représentation en complément à 2^n », que nous abrégerons en « convention $C2^n$ », n étant le nombre de bits pour l'écriture de N.

Pour coder un nombre N en convention $C2^n$:

1. on calcule $2^n + N$,
2. on écrit le résultat base 2,
3. on garde les n bits les plus à droite.

D'autres méthodes seront présentées en TD, ainsi qu'une justification de cette convention.

Exemples de codage en convention C2

- Soit $N = 13$ à coder sur $n = 5$ bits
  1. on calcule $2^5 + 13 = 32 + 13 = 46$
  2. on écrit 46 en base 2, c'est $(101110)_2$,
  3. on ne garde que les n bits de droite.
     Donc sa représentation en convention $C2^5$ est 01110
- Soit $N = -11$ à coder sur $n = 5$ bits
  1. on calcule $2^5 + (-11) = 32 - 11 = 21$
  2. on écrit 21 en base 2, c'est $(10101)_2$,
  3. on ne garde que les n bits de droite.
     Donc sa représentation en convention $C2^5$ est 10101

On remarquera que le codage $C2$ de 0 est 0.
On remarquera aussi que les nombres positifs en convention $C2$ ont le bit de poids fort à 0, tandis que les négatifs l'ont à 1.

Codage des textes simples

Pierre Nerzic

Introduction

Les textes considérés ici sont sans mise en page, juste des listes de caractères, lettres, chiffres, ponctuation et indications de changement de lignes. Les documents avec mise en page, ex: pdf, sont beaucoup plus complexes. On ne se pose même pas la question des fichiers, seulement de la représentation du texte dans un ordinateur.

Dans les textes simples, les caractères sont codés les uns à la suite des autres. Un caractère spécial indique la fin du texte.

Il y a plusieurs codages pour les caractères. L'un des plus anciens s'appelle le « code ASCII » (*American Standard Code for Information Interchange*). D'autres codages ont été développés pour ajouter des représentations pour des caractères nationaux comme les à, ç, ù et €. Le codage Unicode a pour but de regrouper tous les codages.

Pierre Nerzic

Codage ASCII

Les caractères sont écrits sur 7 bits (étendus à un octet en mettant le bit de poids fort à 1)

b_7 b_6 b_5 → / Bits b_4 b_3 b_2 b_1 ↓ / Column → / Row ↓	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
	0	1	2	3	4	5	6	7
0 0 0 0	0 NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1 SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2 STX	DC2	"	2	B	R	b	r
0 0 1 1	3 ETX	DC3	#	3	C	S	c	s
0 1 0 0	4 EOT	DC4	$	4	D	T	d	t
0 1 0 1	5 ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6 ACK	SYN	&	6	F	V	f	v
0 1 1 1	7 BEL	ETB	'	7	G	W	g	w
1 0 0 0	8 BS	CAN	(	8	H	X	h	x
1 0 0 1	9 HT	EM	)	9	I	Y	i	y
1 0 1 0	10 LF	SUB	*	:	J	Z	j	z
1 0 1 1	11 VT	ESC	+	;	K	[	k	{
1 1 0 0	12 FF	FS	,	<	L	\	l	¦
1 1 0 1	13 CR	GS	—	=	M	]	m	}
1 1 1 0	14 SO	RS	.	>	N	^	n	~
1 1 1 1	15 SI	US	/	?	O	_	o	DEL

Codage ASCII, suite

En résumé de la table précédente,

- 32 codes, de $(0000000)_2$ à $(0011111)_2$ sont réservés à des caractères spéciaux, ex: `LF` (line feed) $= (0001010)_2$ indique un retour à la ligne.
- 10 codes représentent les chiffres, de $(0110000)_2$ à $(0111001)_2$
- 26 codes représentent les majuscules, de $(1000001)_2$ à $(1011010)_2$
- 26 codes représentent les minuscules, de $(1100001)_2$ à $(1111010)_2$
- les autres codes représentent des caractères divers, ex: $*$ par $(1011010)_2$
- Il n'y a pas de codes pour les caractères français, ç, é, ö, œ...

ISO-8859-1 (latin1) et ISO-8859-15

Ce sont des codes qui représentent davantage de caractères, en utilisant les 8 bits d'un octet. Il manque le signe € dans le ISO-8859-1. Le codage ISO-8859-15 est mieux adapté aux textes français, mais il sacrifie des caractères comme ½

Le gros problème de ces codages, est qu'il faut savoir lequel a été employé lors du codage pour pouvoir décoder correctement.

Unicode

La norme Unicode est d'abord un catalogue d'un très grand nombre de caractères utilisés dans de nombreux pays, 149186 en 2022, ainsi que leur mode d'emploi (sens d'écriture, rôle des caractères, casse, etc.)

La représentation d'un aussi grand nombre de caractères ne peut pas se faire sur un seul octet. Il y a plusieurs méthodes :

- UTF-8 code les caractères sur 1 à 4 octets.
  - Les valeurs de la plage $(00)_{16}$ à $(7F)_{16}$ codent les caractères ASCII. Par exemple, `A` est codé $(01000001)_2 = (41)_{16}$.
  - Les valeurs de la plage $(C2)_{16}$ à $(DF)_{16}$ indiquent qu'il y a un second octet après. Par exemple, é est codé $(C3)_{16}, (A9)_{16}$.
  - Les valeurs de la plage $(E0)_{16}$ à $(EF)_{16}$ indiquent qu'il y a un deux autres octets après. Par exemple, € est codé $(E2)_{16}, (82)_{16}, (AC)_{16}$

Unicode, suite

Il y a aussi UTF-16 et UTF-32 avec un principe similaire, de représenter les caractères les plus fréquents sur moins d'octets et de rajouter des octets pour les caractères les plus rares.

NB: on n'étudiera pas cette norme en TD/TP, trop complexe.

Pierre Nerzic