Informatique L1 (Sciences Exactes) CODAGE BINAIRE

Contenu du semestre

Cours et TDs sont intégrés

L'objectif de ce cours équivalent a 6h de cours, 10h de TD et 8h de TP est le suivant :

- initiation à l'algorithmique
- notions de bases de programmation
  variables simples, entrées sorties basiques
- choix et itérations
- tableaux 1D

Il faut tenir compte du fait que cet enseignement concerne aussi bien des étudiants qui se destinent a faire de la biologie qu'a des étudiants qui s'orienteront vers les mathématiques ou l'informatique….
Il n'y a pas de note de TP mais la présence en TP est obligatoire (appel)
et vital pour comprendre l'enseignement et réussir aux contrôles !

1 – Rappel

L'informatique

Science du traitement de l'**infor**mation auto**matique**ment

Ensemble de techniques de collecte, de tri , de mise en mémoire, de transmission et de l'utilisation des informations traitées automatiquement à l'aide de programmes mis en œuvre sur ordinateur.

Exemple d'informations :

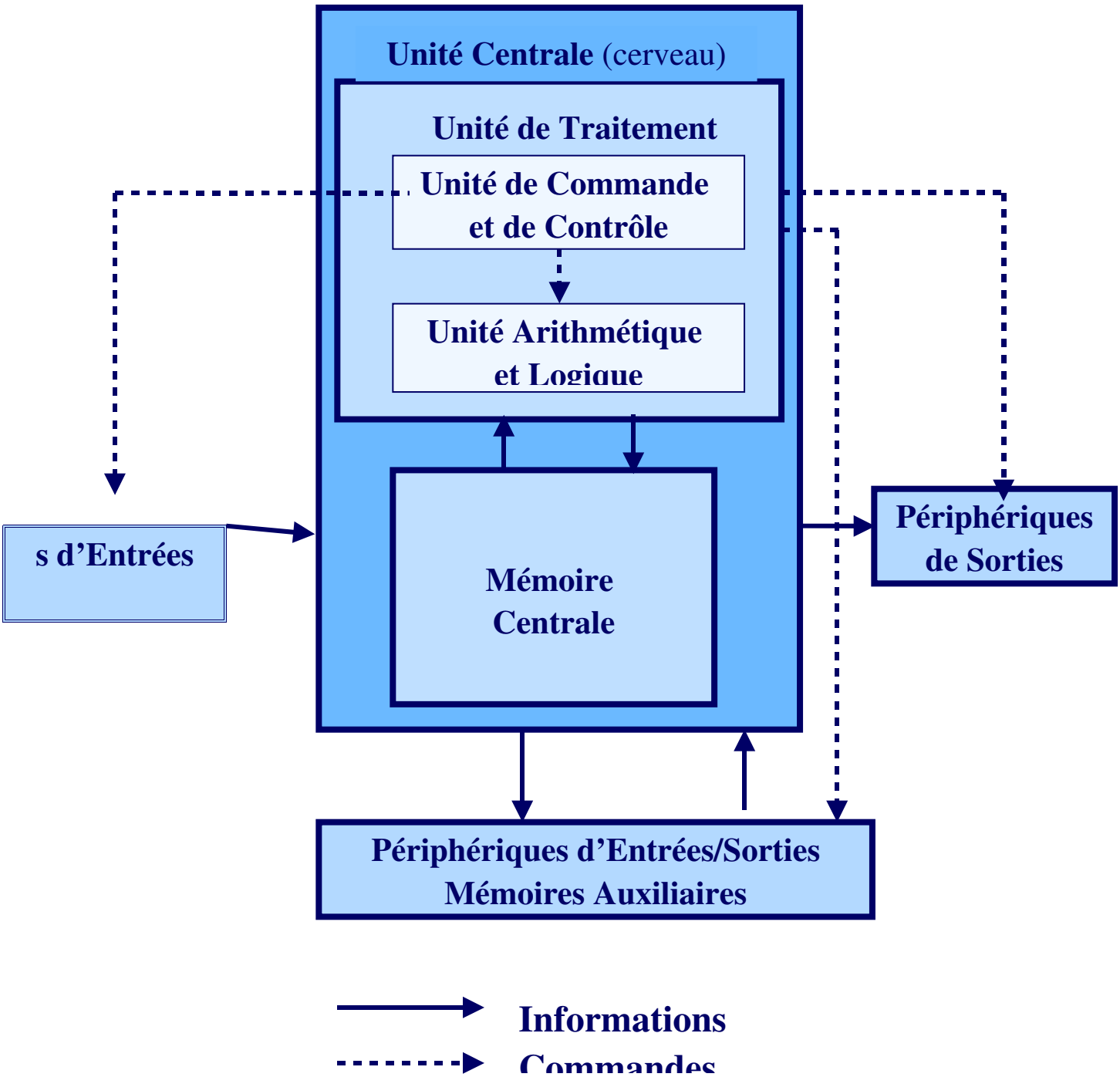
nombres entiers et réels codés en mémoire centrale. Caractères alphabétiques et caractères spéciaux….

Le traitement de l'information consiste à partir d'une information de départ de produire par traitement une nouvelle information de sortie.

L'ordinateur

Un ordinateur est une machine qui permet d'effectuer des traitements sur des données à l'aide de programmes. Les données (les paramètres du problème, par exemple les notes des étudiants du cours d'informatique) ainsi que le programme (par exemple le calcul de la moyenne des notes) sont fournis à la machine par l'utilisateur au moyen de dispositifs de saisie (le clavier

par exemple). Le résultat du traitement est recueilli à la sortie de l'ordinateur (l'écran, l'imprimante) sous forme de texte par exemple.



Nous allons étudier maintenant comment est codée l'information dans la mémoire centrale et dans la mémoire auxiliaire.

2 - Le codage binaire de l'information

Toute l'information qui transite dans un ordinateur est codée avec des bits (**B**inary dig**IT**) ne prenant que les valeurs 0 et 1. L'information est donc toujours discrète

Par exemple pour coder le nombre 13 en binaire, il faut les quatre chiffres binaires **1101**. En effet 13 peut être décomposé comme une somme de puissances de 2

13 = 8 + 4 + 1
= 1 * 8 + 1 * 4 + 0 * 2 + 1 * 1 en décimal
= 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 on ne conserve que les coefficients
= 1 1 0 1 en binaire

Question avec n bits combien de codes différents obtient on ? 2^n

Représentation des informations en binaire

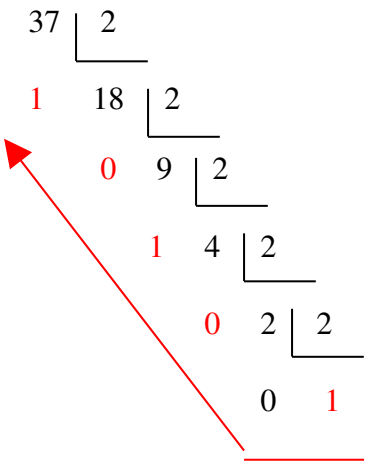
Pour coder de l'information , que ce soient des nombres, (PI=3,141592...), du texte (ce cours), des schémas, des images, des sons, des relations ("Pierre est le père de Jacques et le frère de Marie"), les circuits électroniques d'un ordinateur ne peuvent utiliser que des informations en binaire.
Montrons d'abord comment il est possible de coder n'importe quel nombre entier naturel
IN={0, 1, 2, 3, ... ,1 000, ..., 1 000 000, ...} en binaire.
Puis nous en ferons autant pour les entiers signés les décimaux et les caractères. Enfin il faudra montrer que les images, les sons … peuvent aussi se coder en binaire.

Passer du décimal au binaire

Il suffit de décomposer un nombre décimal en une somme de puissances de 2. On peut par exemple commencer par écrire la table des premières puissances de 2 :

$2^0 = 1$	$2^1 = 2$	$2^2 = 2x2 = 4$	$2^3 = 2x2x2 = 8$
$2^4 = 16$	$2^5 = 32$	$2^6 = 64$	$2^7 = 128$
$2^8 = 256$	$2^9 = 512$	$2^{10} = 1024 = 1K$	$2^{11} = 2048$ …….

Ex calculer la valeur binaire de 37 en base 10 :



$37 == 2 \times ((2 \times 9) + 0) + 1 == 2 \times ((2 \times (2 \times 4 + 1)) + 1$

$== 1 \times 2^5 + 1 \times 2^2 + 1 \times 2^0 == 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

soit en base 2 : 1 0 0 1 0 1

Exercice : Montrer que le nombre 131 est une somme de puissances de 2
Reponse : 10000011

Exercice : Donner la représentation binaire des 10 premiers nombres entiers :
$0 = 0x2^0$ -> 0; $1=1x2^0$ -> 1 ; $2=1x2^1$ -> 10
$3 = 1x2 + 1x1 = 1x2^1 + 1x2^0$ -> 11
$4 = 1x4 + 0x2 + 0x1 = 1x2^2 + 0x2^1 + 0x2^0$ -> 100
5 = 1x4 + 0x2 + 1x1 =
6 =
7 =
8 =
9 =
10 =

Passer du binaire au décimal

Le codage binaire a un inconvénient majeur pour l'être humain, son manque de lisibilité...
Quelle est la représentation décimale du nombre dont la représentation binaire est : 1001 1110 ?
Réponse : $1x2^7+ 0x2^6 + 0x2^5 +1x2^4 + 1x2^3 + 1x2^2 + 1x2^1 + 0x2^0$
= 1x128 + 0x64 + 0x32 +1x16 + 1x8 + 1x4 + 1x2 + 0x1
= 158

Exercice : Combien de chiffres binaires faut-il pour représenter le nombre décimal 10000 ?
Reponse : 14

2^{11}=2048 2^{12}=4096 2^{13}=8192 2^{14}=16384

Opérations usuelles en binaire

Les deux opérations binaires de base sont
- l'addition
- la multiplication

Table d'addition binaire
0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 avec une *retenue* de 1

Table de multiplication binaire
0 x 0 = 0
0 x 1 = 0
1 x 0 = 0
1 x 1 = 1

Exercice :
En utilisant la table d'addition binaire calculez en binaire les nombres suivants :

1001 + 10100
1111 + 1
1111 1111 + 1111 1111

Opérations logiques

En logique binaire une variable logique (booléenne) est VRAI (TRUE = 1) ou FAUSSE (FALSE = 0). Une expression logique est constituée de plusieurs variables logiques combinées par des connecteurs (opérateurs) logiques :

Les opérateurs logiques élémentaires sont :

- NON [*NOT*]
- ET [*AND*]
- OU [*OR*]

Table de vérité du NON

NON 0 = 1
NON 1 = 0
Table de vérité du OU

0 OU 0 = 0
0 OU 1 = 1
1 OU 0 = 1
1 OU 1 = 1

Table de vérité du ET

0 ET 0 = 0
0 ET 1 = 0
1 ET 0 = 0
1 ET 1 = 1

Exercice : Donner la valeur de vérité {VRAI ou FAUX} des assertions suivantes :

« 102 est un nombre pair » ET « 102 est divisible par 3 »
« 11 est multiple de 3 » ET « 102 est divisible par 3 »
« 108 est multiple de 9 » OU « 11 est divisible par 3 »

Codage des nombres entiers et des nombres réels décimaux

L'ensemble des entiers naturels : IN = {0,1,2,3 ,4 ,……..}
- IN est ordonné ; il a un plus petit élément 0 ;

- Il n'y a pas de plus grand élément : tout élément a un successeur

L'ensemble des entiers relatifs : Z = Z+ ∪ Z-
Z= {...,-3,-2,-1, 0, 1,2,...}
- Z est ordonné
- il n'y a ni plus grand ni plus petit élément : tout élément de Z a un prédécesseur et un successeur.

Opérations sur les entiers

Opérateurs unaires :

- (moins) : opposé

Opérateurs binaires :

+ - * DIV MOD /

Opérateurs de comparaison :

= < <= > >= ≠

Implémentation des entiers non signés

> En raison des limitations d'espace mémoire,
> on ne peut représenter que quelques nombres.

Par exemple si on dispose d'un mot mémoire de 16 bits (soit deux octets) pour représenter des nombres par exemple alors les différents contenus possibles de la mémoire sont les suivants :

Contenu mot	valeur en décimal
0000 0000 0000 0000	0
0000 0000 0000 0001	1
0000 0000 0000 0010	2
0000 0000 0000 0011	3
0000 0000 0000 0100	4
0000 0000 0000 0101	5
……..	
……..	
……..	
1111 1111 1111 1100	65 532

1111 1111 1111 1101 65 533
1111 1111 1111 1110 65 534
1111 1111 1111 1111 65 535

en fait il y a 2^{16} = 65 536 codes différents et si on veut utiliser un mot mémoire pour coder des informations on sait qu'avec un mot mémoire on ne peut coder que 65536 « choses » différentes.

Supposons maintenant qu'on se pose le probléme de coder tous les entiers positifs ou nuls sur ordinateur.

Le nombre d'éléments de cet ensemble est infini….pour coder tous les entiers il faudrait donc un nombre infini de bits… et donc une mémoire centrale infinie….ce qui est impensable !

Donc on ne pourra jamais coder tous les entiers positifs.

Si on prend un mot on ne peut coder que 65536 choses différentes et donc 65 536 entiers sur l'infinité d'entiers possibles .

Si on prend les entiers positifs on a choisi bien sur que

0 c'est le mot mémoire 0000 0000 0000 0000 ……
…….65 335 c'est le mot mémoire 1111 1111 1111 1111

Exemple pour n = 547_{10}

On décompose 547 en puissances successives de 2
:
547_{10}= 512 + 32 + 2 + 1 = $1*2^9 +1*2^5 +1*2^1 +1*2^0$

et on ne code que les coefficients de la décomposition, la représentation binaire de n est
Bin(n) = 0000 0010 0010 0011

Le plus grand nombre entier non signé représentable est donc : 1111 1111 1111 1111

$= 1*2^{15} +1*2^{14} +1*2^{13} +1*2^{11} +1*2^{10} +1*2^9 +1*2^8$
$+ 1*2^7 +1*2^6 +1*2^5 +1*2^4 +1*2^3 +1*2^2 +1*2^1 +1*2^0$
$= 1*2^{16} - 1$
La représentation décimale de ce nombre est donc Dec(n) = 65 536 - 1 = $65\,535_{10}$

les entiers positifs strictement supérieurs à 65 535 ne sont donc pas représentables…..

Maintenant si on prend deux entiers codables par exemple 65 533 et 5 …… Si on veut faire la simple somme et coder le résultat dans un mot on aura un résultat faux

Preuve :

65 533	soit	en binaire	1111 1111 1111 1101
5	soit	en binaire	0000 0000 0000 0101
			----------
		somme	1 0000 0000 0000 0010

Il reste donc dans le mot mémoire contenant le résultat 0000 0000 0000 0010 soit le codage du nombre 2 ce qui n'est pas le bon résultat…..

Il y a eu ce qu'on appelle un dépassement de capacité et le résultat placé dans un mot est faux !

Donc déjà on voit que quand on fait des calculs sur les entiers il faut prévoir les cas ou il y aura dépassement de capacité.

Les difficultés à résoudre pour les entiers quelconques sont donc les suivantes : quels entiers sont représentés et comment représenter les négatifs et par conséquent le signe.

Exercice TD :

Indiquer quels sont les entiers non signés représentables sur un double mot, soit 4 octets
Rep : ils vont de 0 à 2^{32}-1 soit de 0 à 4 294 967 295

Implémentation des entiers signés

Nous allons étudier la représentation des entiers signés sur deux octets

Le problème ici est que d'une part il faut représenter la valeur absolue et d'autre part le signe. Quand on dispose d'un mot mémoire et qu'on fait l'hypothèse qu'il code un entier signé il faut pouvoir reconnaître le signe (+ /-), et donc il est obligatoire qu'un bit sur les 16 bits disponibles sera utilisé pour coder le signe. C'est le bit de gauche (dit de fort poids) qui est utilisé :

S'il vaut 1 alors le nombre codé est négatif
S'il vaut 0 alors le nombre codé est positif

Une représentation utilisée sur les premières machines consistait à coder d'une part le signe sur un bit , d'autre part la valeur absolue sur les 15 bits restants :

On code donc le signe 0/1 puis sur 15 bits la valeur absolue qui va donc de 0 à 2^{15}-1 (=32 767)

On peut donc coder ainsi de –32 767 à + 32 767

Inconvénients de ce codage :

- le 0 est codé deux fois +0 0000 0000 0000 0000
  -0 1000 0000 0000 0000

- l'additionneur de l'unité centrale est complexe :

ex 1 + (-3) s'écrit si l'on pose l'opération d'addition binaire classique:

```
          0000 0000 0000 0001
          1000 0000 0000 0011
          ---------------------------
somme     1000 0000 0000 0100
```

ce qui serait le codage de –4 ….le résultat est donc faux

Le codage des entiers signés se fait actuellement en « complément à 2 » :

Soit un nombre n de Z.

Si le nombre est positif et inférieur ou égal à $2^{15}-1$ (32 768) on le représente comme un entier non signé.

Si le nombre est négatif et supérieur ou égal à -2^{15} (-32768), le problème est de représenter le signe et de pouvoir passer d'un nombre à son opposé de façon simple.

Une méthode très répandue est la méthode du complément à 2. On travaille MODULO 2^{16}
On représente Bin(2^{16} + n) = 65 536 - 547 = 64 989
= 1111 1101 1101 1101

Une vérification de l'expression calculée consiste à effectuer une somme bit à bit de 547 et de –547

Si la représentation est correcte on doit trouver 0 :

```
          0000 0010 0010 0011
+         1111 1101 1101 1101
_____________________________
résultat  0000 0000 0000 0000
```

Le report (ou *retenue*) n'est bien sûr pas représenté.
Donc un nombre entier signé représenté sur 16 bits dont le bit de poids fort est à 1 doit être

considéré comme un nombre NEGATIF : sa valeur est $-(2^{16} - Dec(n_2))$

Algorithme de conversion d'un nombre en binaire complément à 2 sur un octet :
Trouver l'opposé d'un nombre en complément à 2 :
Soit n = 0000 0101 = 5
Son opposé est -5 obtenu en inversant tous les 1 en 0 et tous les 0 en 1 et en ajoutant 1:
0000 0101 --> 1111 1010 + 1 = 1111 1011 = -5
Vérification :

```
5 + (-5) = 0
     0000 0101
+    1111 1011
     ----------
     0000 0000 report ignoré
```

Exercice :
Trouver les représentations binaires en complément à deux sur deux octets des nombres suivants :
31000; -31000 . faire l'addition binaire des deux représentations …conclusion par rapport au codage signe+ valeur absolue ?
-33000 est-il représentable ? pourquoi peut on représenter -32 768
quel est le codage de 0 (essayer +0 et –0)

Codage des informations non numériques

Textes

Les textes peuvent être représentés en binaire à condition d'associer à chaque lettre de l'alphabet une représentation numérique, par exemple son rang dans l'ordre alphabétique : A serait codé1 ; B,de rang 2 serait codé 10 en binaire; C codé 11, etc. Mais ce codage est insuffisant car il faut aussi pouvoir coder les signes de ponctuation . , ; ? ! , distinguer les minuscules des MAJUSCULES, les caractères accentués, etc.
La table ASCII (American Standard Code for Interexchange Information) propose un codage de 128 caractères différents sur 7 bits. Dans la table ASCII la lettre A est codée 65, B est codé 66, C est codé 67, ... Z est codé 90, a est codé 97, b est codé 98, ... 0 est codé 48, 1 est codé 49, ... 9 est codé 57, ...
Les caractères accentués des langues européennes ont nécessité l'ajout d'un huitième bit de codage, d'où la table ASCII étendue avec 256 caractères.

Exercice : Donner le contenu binaire d'un octet contenant le code ASCII du 'a'

C'est la valeur 97 en base 10 soit 1100001 en binaire
D'ou 01100001

Quelques définitions sur le stockage et le codage :

1 octet ou byte :
ensemble de 8 bits
1 mot mémoire :
le mot mémoire est une unité de stockage de base pour un ordinateur.
Chaque ordinateur dispose d'une taille de mot mémoire particulière en général les PC actuels ont des mots mémoire de 32 bits soit 4 octets, mais certaines machines ont des mots mémoire plus grands.

La taille d'une mémoire centrale ou d'un autre support d'information s'exprime en nombre d'octets. Ced nombre est toujours une puissance de 2

Ex : 1 Koctet -> kilo octets -> 1024 octets -> 2^{10} octets
1 Moctets -> méga octets -> 1024 koctets -> 2^{20} octets
1 Goctets -> gigaoctets -> 1024 Moctets -> 2^{30} octets
1 Toctets -> teraoctets ……

quelques mots clé du langage C (ensupposant être sur une machine à mots de 16 bits ?)

code ASCII 8 bits	**char**
entiers signés 16 bits	**short**
entiers signés 32 bits	**int**
entiers non signés (16/32)	**unsigned (short/:int)**
flottants IEEE 32 bits	**float**
flottants IEEE 64 bits	**double**