TD 1 - Architecture et Nombres

1 Exploration d'un ordinateur

1.1 Identification des composants matériels

Par groupes de 6, explorez les ports d'entrées sorties externes de l'ordinateur pour les identifier et en établir la liste sur papier.

Explorez l'intérieur du boitier de l'ordinateur qui aura été ouvert par votre professeur pour établir la liste des composants que vous aurez identifiés. Déterminer si le disque dur est en liaison SATA ou IDE, le nombre de barrettes de RAM, le nombre de disques durs et le nombre de cartes PCI ou PCI-Express.

1.2 Visite du BIOS

Allumer l'ordinateur et, dans les quelques secondes de démarrage, ouvrir le BIOS en tapant la touche (ou la combinaison de touches) spécifiée à l'écran pendant une fraction de seconde au chargement (la touche est probablement F2 pour ouvrir le BIOS et F10 pour sélectionner le support de chargement).

Dans le BIOS, identifier :

- le disque dur et son type de connection,
- la quantité de RAM et la vitesse de transfert,
- Le processeur, son type (32 bits ou 64 bits) et la fréquence de fonctionnement,
- les cartes PCI ou PCI-Express,
- la séquence de boot.

Sortir du BIOS par la touche Esc, et laisser Windows se charger.

2 Exploration de Windows

Ouvrir le "poste de travail" et en cliquant droit, propriétés, déterminer la taille des partitions disponibles. Parcourir les lecteurs réseaux et trouver le répertoire de votre professeur et celui de SII. Profitez en pour copier le dossier "TP1_programmes" dans votre espace personnel (Documents).

Dans le dossier temporaire de l'ordinateur, créer un dossier et changez les droits pour "lecture seule". Tentez d'y ajouter un dossier.

Par la combinaison de touches ctl-alt-suppr, ouvrir le gestionnaire des tâches. Lancer le programme "boucle infinie.bat" et arrêtez-le par le gestionnaire des tâches. Lancer deux fois (voire 3 fois) le programme puis modifier la priorité de l'un d'entre eux pour observer la répartition du temps de calcul accordé par l'OS à chacun.

2.0.0.1 Installer un OS sous Virtuabox

Sur internet, télécharger Virtuabox pour votre ordinateur (https ://www.virtualbox.org/) et installer le logiciel. Télécharger le fichier .iso d'installation de la distribution OpenSuse (http ://software.opensuse.org/123/fr) en choisissant le mode "réseau".

À partir des indications du polycopié de cours, procéder à l'installation de l'OS sur une machine virtuelle. Cette installation est relativement longue et sera refaite chez vous où pendant les TP de SI.

3 Nombres

3.1 Représentation des données en mémoire

3.1.1 les entiers

Ouvrir le logiciel Spyder, permettant d'écrire et d'exécuter des programmes Python.

Python se charge de gérer les types en fonction des opérations effectuées donc ces notions sont transparentes pour le programmeur. Pour obtenir le type d'une variable, il suffit d'exécuter la commande *type* :

```python
a=5
type(a)
```

Python ne laisse pas le programmeur accéder directement à la mémoire. Pour étudier la représentation en mémoire des nombres, il faut écrire ces nombres en binaire dans un fichier et observer les bits qui ont été écrits. Le programme ci-dessous importe une bibliothèque de manipulation des nombres, ouvre un fichier en écriture (mode binaire et non pas texte), écrit un entier sur 8 bits et referme le fichier. Il n'est pas nécessaire pour le moment de retenir ces commandes ; observez seulement les propriétés de codage.

Taper le programme suivant et enregistrez le dans votre espace personnel, dans le dossier "TP1_programmes".

```python
from numpy import *
f=open("data.txt",'wb')
f.write(int8(5))
f.close()
```

Exécuter le programme par la flèche verte dans le menu. Un fichier texte "data.txt" doit apparaitre dans le dossier "TP1_programmes". Convertir en binaire le chiffre 5 ? La taille du fichier est-elle celle attendue ?

Pour observer le contenu du fichier binaire, ouvrir l'éditeur hexadécimal "Frhed" et charger le fichier. Le résultat écrit est il correct ? Exécuter à nouveau le programme pour des entiers valant 255, 258 et -2.

Modifier le programme en écrivant un *int16* et refaites le test pour l'entier 5. Valider la taille du fichier. Comment sont organisés les octets ? Refaites le test pour les entiers 255, 258 et -2.

Remarque : vous pouvez exprimer les données dans python directement en binaire ou en hexadécimal, en utilisant la notation issue du C : 0b101 (5 en base 10) ou 0x1A (26 en base 16). De même, les fonction *hex(26)* ou encore *bin(5)* permettent de faire les conversions inverses.

3.1.2 Les flottants

Observons maintenant le codage des flottants :

```python
from numpy import *
f=open("data.txt",'wb')
f.write(float32(3.4))
f.close()
```

Valider la taille du fichier. Ouvrir le fichier *data.txt* dans l'éditeur hexadécimal et, à partir des informations du cours, retrouver le chiffre exact approchant 3.4 :

- suite à la convention *little endian*, remettre les octets dans l'ordre,
- distinguer les bits de signe (1 bit), d'exposant (8 bit) et de mantisse (23 bit),
- ajouter le 1 implicite à la mantisse,
- retrouver l'exposant en appliquant le décalage de 127,
- convertir la mantisse et l'exposant en décimal
- retrouver le nombre flottant exact codé, et approchant 3.4.

3.1.3 Les caractères

Nous allons retrouver votre matricule... Dans le bloc-note, ouvrir le fichier "data.txt" et taper votre prénom, sans accent. Enregistrer et ouvrir le fichier dans l'éditeur hexadécimal. Comparer les codes hexadécimaux des caractères avec les lettres de la table ASCII.

Sous Python, taper la commande *map(ord,"mon prenom")* et comparer les codes obtenus.

La commande *ord("a")* permet d'obtenir la valeur numérique associée à un caractère de la table ascii. La commande *chr(70)* permet d'obtenir le caractère 70 dans la table ASCII.

Ouvrir à nouveau le fichier "data.txt" dans le bloc-note et taper une lettre accentuée. Observez le résultat dans l'éditeur hexadécimal.

3.2 Les erreurs numériques sur les calculs flottants

Les flottants ne décrivent évidement pas $\mathbb{R}$ puisque sur un nombre de bits fini, l'information ne peut être que finie. Une valeur quelconque de $\mathbb{R}$ a donc toute les chances d'être approchée par une valeur disponible dans l'ensemble décrit par le codage du flottant.

Taper les commandes suivantes :

```python
from decimal import *
Decimal(0.1)
```

Le nombre 0.1 est en réalité approché par un nombre très proche mais différent. Testez l'addition 0.1+0.2 et observez le résultat...

Il devient alors évident que le test d'égalité entre deux flottant n'est pas fiable : (0.1+0.2)==0.3 renvoie "False". Il est seulement possible de tester si l'écart entre les deux flottant est petit (de l'ordre de la précision du codage).

De même les opérations sont réalisées dans un ensemble fini si bien que chaque résultat intermédiaire est approché. Cela peu conduire à des observations étonnantes : testez les lignes suivantes (** indique l'exposant) :

```python
a=1e20;b=-1e20;c=1;
(a+b)+c
a+(b+c)

2**0.5 * 2**0.5 -2
```

Aussi, lorsqu'un grand nombre d'opérations est envisagé pour calculer un résultat numérique, ces erreurs peuvent se propager et entacher le résultat final.

En particulier lorsque deux nombre grands sont retranchés pour former un nombre petit. Tester les opérations suivantes (le format float16, parfois appelé "demi-précision", est imposé pour mettre en évidence les erreurs numériques) :

```python
float16(4.002)-float16(3.998)
```

Le résultat est tellement faux qu'aucun chiffre significatif n'est correct...

Les nombres à virgule ne sont pas les seuls à être impactés : même les entiers (lorsqu'ils sont codés en flottants) sont approchés lorsqu'ils sont grands :

```python
float16(3002)-float16(2999)

x=200;y=199;
float16(x-y)*float16(x+y)
float16(x**2)-float16(y**2)
```

Aussi faut-il éviter de faire des calculs dans les entiers en utilisant le type flottant...

Déterminer le plus petit nombre positif codé par un flottant 16 bits, le plus grand nombre positif codé, le nombre immédiatement après 1, et le nombre de chiffre significatif. Déterminer alors la cause des erreurs précédentes.

Dans le second exemple où $x^2-y^2 \neq (x-y)(x+y)$, quel résultat est le bon ? Peut-on en déduire une règle pour l'écriture des opérations ?

Heureusement, Python adapte dynamiquement ses types ce qui permet d'éviter un certain nombre de problèmes, mais pas tous... Testez les deux programmes suivants, qui permettent de calculer π itérativement :

```python
x =1.0 / sqrt(3.0)
n =6
for i in range(30):
  n=n*2
  x=((x**2+1)**0.5-1)/x
  print n*x
```

```python
x =1.0 / sqrt(3.0)
n =6
for i in range(30):
  n=n*2
  x=x/((x**2+1)**0.5+1)
  print n*x
```

Pourtant, $\dfrac{\sqrt{x^2+1}-1}{x} = \dfrac{x}{\sqrt{x^2+1}+1}$