info201 : Système d'exploitation
TD 1 : architecture

Pierre Hyvernat
Laboratoire de mathématiques de l'université de Savoie
bâtiment Chablais, bureau 17, poste : 94 22
email : Pierre.Hyvernat@univ-smb.fr
www : http://www.lama.univ-smb.fr/~hyvernat/

Exercice 1 : bits, octets et adresses

Les préfixes usuels en informatiques sont :
- l'octet, "1 o" pour une suite de 8 bits,
- le kibioctet, "1 kio" pour une suite de $2^{10} = 1024 \approx 10^3$ octets,
- le mébioctet, "1 Mio" pour une suite de 2^{10} kibioctets,
- le gibioctet, "1 Gio" pour une suite de 2^{10} mébioctets,
- le tébioctet, "1 Tio" pour une suite de 2^{10} gibioctets,
- ...

Question 1. Donnez le nombres d'octets dans 1Mio, 1Gio et 1Tio.

Question 2. Un kilooctet (1ko) fait *exactement* 1000 octets alors qu'un kibioctet fait *approximativement* 1000 octets. Un kibioctet a donc 2.4% d'octets en plus qu'un kilooctet.

De même, téraoctet (1To) fait *exactement* 10^{12} octets alors qu'un tébioctet fait *approximativement* 10^{12} octets.

Estimez le gain d'octets (en pourcentage) entre 1To et 1Tio.

Question 3. Les architectures 32 bits stockent les adresses sur 32 bits. Quelle est la taille maximale de la mémoire RAM adressable ?

Les architectures 64 stockent les adresses sur 64 bits, mais seulement les 48 bits de poids faible sont en général utilisés. Quelle est la taille maximale de la mémoire RAM adressable. Est-ce que cela est raisonnable ?

Exercice 2 : architecture

Question 1. Rappelez, en quelques phrases, le principe d'une architecture de von Neumann (aussi appelée architecture de Princeton).

Question 2. Pour chaque matériel donné, précisez s'il s'agit d'un périphérique d'entrées, de sorties, ou d'entrées-sorties ; ou s'il ne s'agit pas d'un périphérique.

	entrées	sorties	entrées / sorties	autre
disque dur				
lecteur / graveur DVD				
ventilateur USB				
clé USB				
souris				
webcam				
micro				
carte réseau				
cable alimentation 220V				
imprimante				
mémoire RAM				
écran tactile				

Exercice 3 : langage d'assemblage

Voici quelques instruction MMIX :

- `SETL $x,` n (n entre 0 et 65535, sur 2 octets) pour mettre la valeur n dans le registre `$x`,
- `ADD $x, $y, $z` ou `ADD $x, $y,` n (n entre `0` et `255`, sur un octet) pour ajouter `$y` et `$z` (ou n) et mettre le résultat dans le registre `$x`, (on peut remplacer `ADD` par `SUB`, `MUL` ou `DIV` pour calculer une soustraction, une multiplication ou une division entière)
- `LDT $x, $y, $z` ou `LDT $x, $y, n` (n entre 0 et 255) pour charger l'entier à l'adresse `$y+$z` (ou `$y +` n) dans le registre `$x`,
- `CMP $x, $y, $z` pour mettre `+1` ou `-1` dans le registre `X` suivant que `$y` est plus grand ou strictement plus petit que `$z`,
- `BP $x,` n (n entre `-65536` et `+65536`, sur 2 octets) pour sauter directement à l'instruction "adresse courante+n" si `$x` est strictement positif (sinon, on passe à l'instruction suivante),
- ...

Question 1. On suppose que la variable x est contenue dans le registre numéro 57. Comment peut-on effectuer l'opération "$x = (x+1)^2$" en minimisant le nombre d'instructions ?

Même question pour "$x = (x+1)^4$" et "$x = (x+1)^5$".

(3) *Question 2.* Comment pourrait-on coder la séquence Python suivante en MMIX ?

```
n = 0
while x > 0:
    n = n + 1
    x = x // 2
```

Vous supposerez que `x` est un registre contiennant un entier positif ou nul et que `n` est déjà initialisé à `0`.

Question 3. Les entiers (64 bits) d'un tableau sont stockés dans des cases consécutives de la mémoire. L'adresse du premier nombre est contenue dans le registre `T` (abbréviation pour `$67`). L'adresse du deuxième nombre est donc à l'adresse `T+8`, l'adresse du troisième à l'adresse `T+16`, etc.

Le registre `L` (abbréviation pour le registre `$211`) contient le nombre de cases de ce tableau.

Écrivez une suite d'instructions qui permet de calculer la somme des `L` nombres du tableau et de mettre le résultat dans le registre `R` (abbréviation pour le registre `$213`).

Exercice 4 : pipeline

L'exécution complète d'une instruction RISC se décompose en 5 étapes :

1/ récupérer l'instruction dans la mémoire (IF : "instruction fetch") et incrémenter le PC ("program counter") pour pointer vers l'instruction suivante ;
2/ décoder l'instruction (ID : "instruction decode") pour voir ce qu'elle fait (et si elle est valide) et récupérer la valeur des registre concernés,
3/ executer l'instruction (EX) :
   . si c'est une opération arithmétique ou logique, l'ALU s'en charge,
   . si c'est une lecture / écriture en mémoire, il faut calculer l'adresse `$y` $+ n$;
4/ accès à la mémoire (Mem) (si l'instruction n'accède pas à la mémoire, cette étape ne fait rien) ;
5/ sauvegarde (WB : "writeback") du résultat dans le registre approprié.

Question 1. Chacune des étapes précédentes prend environ 1 cycle processeur.

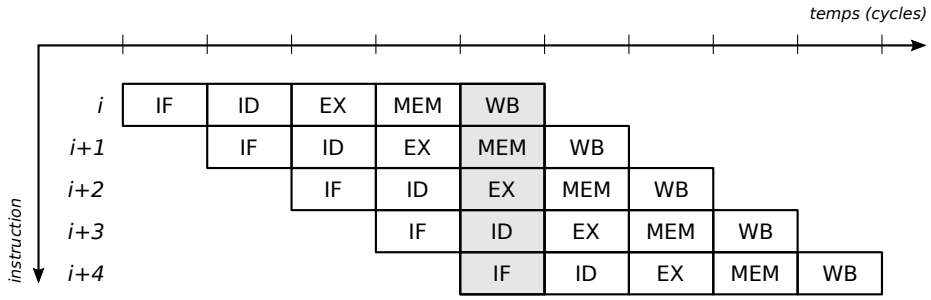
Combien faut-il de cycles pour exécuter une instruction en entier ?

Combien faut-il de cycles pour exécuter 2 instructions, 3 instructions, 100 instructions en entier ?

Quel est le temps moyen pour exécuter une instruction *en moyenne*.

Question 2. Un *pipeline* permet d'exécuter des étapes différentes d'instructions différentes *en même temps* : l'étape IF de l'instruction deux est faite en même temps que l'étape ID de la première instruction.

Lorsque le pipeline est plein, 5 instructions sont donc "en cours d'exécution" (voir figure).



Combien faut-il de cycles pour exécuter une instruction en entier ?

Combien faut-il de cycles pour exécuter 2 instructions, 3 instructions, 100 instructions en entier ?

Quel est le temps moyen pour exécuter une instruction *en moyenne*.

Question 3. Que se passe t'il si on exécute les 2 instructions suivantes dans le pipeline.

Proposez une solution.

```
MUL $7, $12, $155
ADD $150, $7, $4
```

Question 4. Que se passe t'il si on exécute les instructions suivantes ?

```
ADD $150, $7, $4
MUL $7, $12, $155
```

Même question pour

```
MUL $7, $9, $100
ADD $7, $87, $255
```