info201 : Système d'exploitation
TD 4 : architecture et assembleur

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 : Mémoire RAM

Question 1. 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 en langage d'assemblage pour l'architecture Intel x86.

- `mov REG1, REG2` pour une affectation de registre, similaire à "`REG1 = REG2`" en Python ;
- `add REG1, REG2` pour additionner 2 registres, comme "`REG1 = REG1 + REG2`" en Python ;
- `imul REG1, REG2` pour multiplier 2 registres, comme "`REG1 = REG1 * REG2`" en Python ;

Remarque : `REG2` peut être un nombre entier dont la taille ne dépasse pas la taille du registre.

Attention, l'instruction `imul` modifie le registre `rdx`. Le produit de 2 nombres de 64 bits peut en effet faire 128 bits, et ces 128 bits sont stockés dans `REG1` (64 bits de poids faible) et `rdx` (64 bits de poids fort).

Question 1. On suppose que la variable x est contenue dans le registre numéro `rbx`. Comment peut-on effectuer l'opération "$x = (x+17)^2$" en minimisant le nombre d'instructions ?

Même question pour "$x = (x+17)^4$" et "$x = (x+17)^5$".

D'autres instructions existantes sont

- `sub REG1, REG2` s'utilise comme `add`, mais effectue une soustraction ;
- `idiv REG` permet de faire une division entière. C'est la plus complexe :
  - . il faut mettre le numérateur dans `rax`
  - . il faut mettre 0 dans `rdx`
  - . `idiv REG` calcule la division entière `rax//REG` et stocke le résultat dans `rax`
  - . `idiv REG` calcule en même temps le reste `rax%REG` et stocke le résultat dans `rdx`
- `xor REG1, REG2` pour faire un XOR bit à bit entre les registres `REG1` et `REG2` ;
- `shl REG, N` pour faire un décalage des bits de `REG` vers la gauche (`shift left`) et `shr REG, N` pour faire un décalage de bits vers la droite. En python : "`REG = REG<<N`" et "`REG = REG>>N`".
- `cmp REG1, REG2` pour comparer les contenus des registres `REG1` et `REG2` (voir plus bas)
- les instructions de "saut" commencent par la lettre "`j`", comme "`jump`" :
  - . `jmp LABEL` pour aller à la ligne avec le label `LABEL` ;
  - . `je LABEL` pour aller à la ligne avec le label `LABEL` *si l'instruction de comparaison précédente a renvoyé "égal"* ("`e`" comme "`equal`") ;
  - . `jne` ("`not equal`"), `jle` ("`less or equal`"), `jg` ("`greater`") etc. permettent de gérer les autres comparaisons après une instruction `cmp`.

Question 2. Écrivez les instructions pour faire l'équivalent de `z = min(x,y)`, càd de

```
if x<y:
    z = x
else:
    z = y
```

Vous supposerez que la valeur de `x` est contenue dans le registre `rax` et celle de `y` est contenue dans le registre `rbx`. Vous devez mettre le résultat (`z`) dans `rcx`.

Question 3. Comment pourrait-on coder la séquence Python suivante en assembleur x86 ?

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

Vous supposerez que la valeur de `x` (un entier positif ou nul) est contenue dans le registre `rcx`.

Question 4.

- On peut remplacer un registre dans les instructions `add`, etc. par une *adresse mémoire* entre crochets "`[...]`". Cette adresse peut être de la forme `[B + I*N + D]` (`B` et `I` sont des registre, `N` est une constante). Cela permet d'utiliser 8 octets en RAM "comme" un registre.
- `inc REG` et `dec REG` sont essentiellement des racourcis pour `add REG, 1` et `sub REG, 1`.

Les entiers (64 bits) d'un tableau sont stockés dans des cases consécutives de la mémoire. Si l'adresse du premier nombre est contenue dans le registre `r8`, l'adresse du deuxième nombre est donc à l'adresse `r8+8`, l'adresse du troisième à l'adresse `r8+16`, etc.

En supposant que le registre `r9` contient le nombre de cases de ce tableau, écrivez une suite d'instructions x86 qui permet de calculer la somme des cases du tableau et de mettre le résultat dans le registre `rax`.