Université de Bordeaux
Licence STS

ARCHITECTURE DES ORDINATEURS

TP : **04**

Un peu de vraie vie

Pour voir à quoi peut ressembler la vraie vie, nous allons utiliser un émulateur de PC complet.

Vérifiez que cela fonctionne

Décompressez le fichier d'archive :

`https://dept-info.labri.fr/ENSEIGNEMENT/archi/td-tm/qemu.tgz`

dans un répertoire de votre compte. Lancez « `make run` ». Vous devriez obtenir une fenêtre avec juste un `A` et un `B` affichés en haut à gauche.

La fenêtre représente l'écran du PC virtuel, et le terminal représente le port série, où l'on peut taper des lettres. Dans le terminal où vous avez lancé le `make run`, tapez une lettre. Celle-ci devrait s'afficher à l'écran.

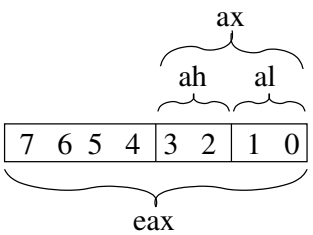
Attention : si par hasard vous cliquez sur la fenêtre Qemu, vous ne pourrez plus utiliser la souris. Voyez ce que dit le titre de la fenêtre Qemu : il faut appuyer sur `Ctrl-Alt-G` pour récupérer la souris.

Les pinailleurs diront qu'on est encore dans un émulateur et pas une vraie machine. Ils pourront essayer de graver le fichier ISO et booter une vraie machine avec ! ;)

À la découverte de `monprog.S`

Ouvrez le fichier `monprog.S`, lisez et comprenez. Quelques clés de compréhension :

- Il s'agit d'assembleur x86, c'est-à-dire celui utilisé par les PC standards, dont le y86 que nous avons appris jusqu'ici était une version simplifiée. Ici, c'est l'assembleur x86 complet.
- Les commentaires sont écrits ici comme en C.
- En x86, il n'y a plus de préfixes `mr`, `rr`, `ir` etc. Pour distinguer entre `i` et `m`, on met un `$` devant les nombres et étiquettes pour le cas `i`. Ainsi, `movl 1,%eax` va lire en mémoire à l'adresse 1. Pour mettre 1 dans `eax`, il faut utiliser `movl $1,%eax` . De même, `movl a,%eax` va mettre dans `eax` le contenu de la mémoire à l'étiquette `a`, et `movl $a,%eax` va mettre dans `eax` l'adresse de l'étiquette `a`.
- Ici, en plus de `l`, on utilise les suffixes `b` et `w`, qui travaillent respectivement sur 8 bits et 16 bits. Il faut alors changer de nom de registre : par exemple, au lieu de `eax`, on pourra utiliser `al` et `ax`, qui sont respectivement les parties 8 bits et 16 bits du registre `eax`. Cela reste le même registre : écrire dans `eax` écrase le contenu de `al` (qui contient donc alors les 8 bits de poids les plus faibles de la valeur 32 bits que l'on a écrite dans `eax`). La structure d'un registre est la suivante (avec les demi-octets – positions des chiffres hexadécimaux – numérotés par ordre croissant) :



— L'écran est à l'adresse `0xb8000`, représenté par une matrice de $25 \times 80 \times 2$ octets : 25 lignes de 80 colonnes, et pour chaque caractère un octet pour le code de caractère ASCII (utilisez `man 7 ascii`) et un octet pour coder la couleur. Le premier caractère est donc à l'adresse `0xb8000`, le deuxième caractère est à l'adresse `0xb8002`, etc., le premier caractère de la deuxième ligne est à l'adresse `0xb80a0`, etc.
— La fonction `getchar` permet de lire un caractère tapé au clavier. Elle retourne simplement le caractère. Attention, elle utilise bien sûr des registres *caller save* (`eax`, `ecx`, `edx`).

Notes sur le débogage

Pour le débogage, on pourra utiliser GDB. Lancez « `make run-dbg` » : cela ouvre une deuxième fenêtre avec GDB dedans. Lorsque l'exécution se suspend (ou lorsqu'on l'interrompt avec `Ctrl-C`), vous pouvez utiliser `info registers` pour voir tous les registres ou juste `p $eax` pour examiner un registre, `s` (« *step* ») pour exécuter le programme pas à pas, `c` (« *continue* ») pour continuer l'exécution, etc.

Exercice 1 : À vous de jouer

Pour travailler, copiez l'exemple sous un autre nom :

```
$ cp monprog.S monautreprog.S
```

et pour lancer cet autre programme, utilisez : `make run TORUN=monautreprog`

Question 1

Affichez les chiffres de 0 à 9. Attention, vérifiez bien le format de l'écran et évitez de mettre 0 comme couleur car c'est le code pour avoir du noir sur fond noir.

Question 2

Faites afficher à l'écran l'un derrière l'autre les caractères tapés au clavier [1].
Pour aller plus loin : gérez la touche entrée (caractère 13).

Question 3

Vous disposez de l'instruction `shr $i,reg` qui *décale à droite* (« *shift right* ») le registre `reg` de i bits, c.-à-d. qui divise `reg` par 2^i. Écrivez une fonction prenant en paramètre un entier et qui l'affiche en hexadécimal à l'écran.

Pour effectuer la conversion entre la valeur extraite d'un demi-octet (qui sera donc comprise entre 0 et 15) et la valeur de caractère associée (`'0'` à `'9'` puis `'A'` à `'F'`), une solution élégante consiste à créer une table de conversion, indexée par la valeur du demi-octet, et contenant les codes ASCII associés à ces valeurs, sur un octet.

Pour aller plus loin...

Question 4

Utilisez cette fonction pour afficher le code ASCII des touches que vous tapez. Repérez celui de la touche `backspace` (effacement arrière), et implémentez son fonctionnement.

Question 5

Faites déplacer un bonhomme (caractère 1) à l'écran avec les touches « QSDF ». Attention aux bords ! Faites-le se déplacer tout seul (vecteur vitesse « (1,1) »), en le faisant rebondir sur les bords.

1. Oui, c'est normal que les lettres accentuées produisent un affichage étrange, on ne traitera pas ce problème.