

# Un peu de vraie vie

Pour voir à quoi cela peut ressembler la vraie vie, on va utiliser un émulateur de PC complet. Pensez à lire la dernière section sur le débogage.

## 1 Vérifiez que cela fonctionne

Copiez le répertoire `/net/cremi/sathibau/Archi/qemu` dans votre *home*. Lancez `make run`. Vous devriez obtenir une fenêtre avec juste un **A** et un **B** affichés en haut et un **C** dans le terminal où vous avez lancé le `make run`. La fenêtre représente l'écran du PC virtuel, et le terminal représente la sortie du port série. Dans le terminal où vous avez lancé le `make run`, tapez une lettre (i.e. elle est envoyée sur le port série du PC virtuel). Celle-ci devrait s'afficher à l'écran.

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

## 2 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, l'y86 que nous avons appris jusqu'ici était une version simplifiée, ici c'est l'assembleur x86 complet.
- Les commentaires ici sont comme en C.
- Ici il n'y a plus de préfixes `mr`, `rr`, `ir` etc., faites donc bien attention à bien mettre un `$` devant les constantes aux endroits où l'on aurait mis `i` dans le nom de l'instruction.
- 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 ce qu'on aurait mis dans `al`).
- L'écran est à l'adresse `0xb8000`, où il y a une matrice de 25x80x2 octets : 25 lignes de 80 colonnes, et un octet pour le caractère (ASCII, utilisez `man ascii`) et un octet pour 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.
- Les instructions `inb` et `outb` permettent de lire et écrire sur les *ports d'entrée/sortie* qui pilotent les périphériques externes au processeur. Ici, on utilise seulement le port série : le port d'entrée/sortie `0x3F8` permet à la fois d'envoyer et de recevoir des caractères vers et depuis le port série. En réception, il faut cependant attendre que le bit de poids 0 du port `0x3F8+5` soit passé à 1.

## 3 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 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.

## 4 À vous de jouer

Pour lancer un autre programme que `monprog`, utilisez `make run TORUN=monautreprog`

- Affichez les chiffres de 0 à 9 (attention, vérifiez bien le format de l'écran et évitez de mettre 0 comme couleur (c'est le code pour noir sur fond noir) !)
- Faites afficher à l'écran les caractères qui proviennent du port série<sup>1</sup>. Gérez la touche entrée (caractère 13).
- Ici, vous disposez de l'instruction `shr i,reg` qui *décale à droite* le registre `reg` de  $i$  bits, i.e. qui divise `reg` par  $2^i$ . Écrivez une fonction prenant en paramètre un entier et qui l'affiche en hexadécimal.
- Utilisez cette fonction pour afficher le code ASCII des touches que vous tapez. Repérez celui de la touche `backspace` (effacement arrière), implémentez son fonctionnement.

---

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