ARCHITECTURE DES ORDINATEURS

TP : **01**

Premiers pas en assembleur

Le modèle de machine choisi est celui du cours *Computer Systems : A Programmer's Perspective* (`http://csapp.cs.cmu.edu/`) ; il s'agit d'une version simplifiée des processeurs Intel, appelée Y86 en référence à l'architecture X86.

Installation du simulateur

Commencez par créer un répertoire `archi` et travaillez toujours dans ce répertoire. Notez que vous pouvez aussi utiliser les étapes suivantes pour installer le simulateur sur votre propre machine ; il vous faudra juste installer TCL/TK en version développement : `tk8.5-dev` sous Ubuntu/Debian, `tk-devel` sous RedHat/Fedora/Mandriva, ainsi que `flex` et `bison`. MacOS propose également ces paquetages. Pour Windows, il faudra au moins installer MINGW et TCL/TK (version 8.5, par exemple la version free d'ActiveTcl) ; il faut par contre corriger le `Makefile` en remplaçant `-ltcl8.5` et `-ltk8.5` par `-ltcl85` et `-ltk85` ; il faut également mettre `C:\Tcl\bin` dans le `PATH`.

- Récupérez l'archive sur `http://dept-info.labri.fr/ENSEIGNEMENT/archi/sim.tgz`
- Désarchivez le fichier : `tar xvf sim.tgz`
- Allez dans le répertoire sim : `cd sim`
- Lancez la compilation : `make` . Il ne devrait pas y avoir d'avertissements ou erreurs.

Si tout s'est bien passé, plusieurs simulateurs ont été créés. Le plus simple est un simulateur séquentiel situé dans le répertoire `seq` :

- Allez dans le répertoire `seq` : `cd seq`
- Lancez le simulateur : `./ssim -g ../y86-code/prog6.yo &` (l'option `-g` indique le mode graphique) ; il faut obligatoirement spécifier un fichier objet. Celui-ci (`prog6.yo`) est fourni à titre de test dans la distribution.

Si tout s'est bien passé, trois fenêtres apparaissent :

- La première contient le code du programme ;
- La deuxième fenêtre est le panneau de contrôle. Cliquez sur le bouton `Step` pour exécuter une instruction, et examinez l'affichage. La partie du milieu (les cadres bleus `Stage`) ne nous intéressent pas immédiatement : ils représentent le fonctionnement interne du processeur simulé, que nous étudierons plus tard. Concentrez-vous surtout sur les boutons du haut, le registre `PC` et la partie `Register File` ;
- La troisième fenêtre décrit le contenu de la mémoire. Elle ne sera pas utilisée pour l'instant, mais ne la détruisez pas pour autant, le simulateur n'aime pas !

Pour éditer le code source, ouvrir le fichier `../y86-code/prog6.ys` avec son éditeur préféré. Par exemple avec emacs, il faut faire `M-x asm-mode` pour avoir une coloration syntaxique du code assembleur.

On peut personnaliser emacs ou vim pour avoir une colorisation syntaxique automatique des fichiers `.ys`. Pour emacs, on ajoute à `~/.emacs` (le créer s'il n'existe pas) :

```
(add-to-list 'auto-mode-alist '("\\.ys$" . asm-mode))
```

et pour vim, on ajoute à `~/.vimrc`

```
au BufNewFile,BufRead *.ys setf asm
```

Editez maintenant le fichier objet `../y86-code/prog6.yo`, dans lequel vous trouverez la traduction des instructions assembleur en langage machine.

Registres

La machine possède huit registres dont les noms apparaissent en bas du panneau de contrôle. Pour l'instant, on utilisera seulement les quatre premiers. Leurs noms et l'ordre dans lequel ils apparaissent sont curieux, et résultent en fait de l'évolution historique des processeurs Intel.

Instructions d'affectation

1. L'instruction
   `irmovl 10, %edx`
   place la valeur 10 dans le registre `edx` ; décryptage : `i` = immédiat, `r` = registre, `mov` = *move*, `l` = `long` (mode 32 bits).
2. L'instruction
   `rrmovl %edx, %eax`
   copie la valeur du premier registre dans le second ; décryptage : `i` a été remplacé par `r` = registre.

Instructions arithmétiques

Les instructions arithmétiques peuvent porter sur deux registres ou une valeur immédiate et un registre ; on utilisera `addl`, `subl`, `andl`, `xorl`. Dans `prog4.yo`, l'instruction

```
addl %edx, %eax
```

ajoute le contenu du premier registre au second ; le premier registre est inchangé, le résultat de l'addition est placé dans le second. On peut utiliser la variante

```
iaddl 1, %eax
```

pour ajouter une valeur immédiate (ici 1) au registre `%eax`

Exercice 1 : Addition

Question 1

Écrire dans un fichier `ex1.ys`, dans le répertoire `seq`, un programme qui place les valeurs 5 et 22 dans les deux premiers registres, puis calcule leur somme dans le troisième registre, sans modifier les deux premiers.

Compiler ce programme : `make ex1.yo` ; charger `ex1.yo` dans le simulateur (bouton Load en haut de la fenêtre du code) ; exécuter pas à pas, en observant attentivement le panneau de contrôle (et la fenêtre du code) ; le bouton `Reset` permet de relancer l'exécution.

Question 2

Ajoutez au programme les instructions nécessaires pour tester les autres instructions arithmétiques. Essayez entre autres de calculer $2 - 3$. N'oubliez pas de recompiler le programme avant de le recharger et de l'exécuter. Prenez le temps de bien observer et comprendre tout ce qui se passe et apparaît dans le simulateur. En particulier, voyez que tout est écrit en notation hexadécimale dans le simulateur.

Question 3

Les *codes de conditions* se trouvent en bas à droite du panneau de contrôle. On s'intéresse au premier, noté `Z` (Zero) ou `ZF` (Zero Flag).

Quelle est la condition sur le contenu de deux registres pour que leur conjonction (`andl`) soit nulle ?

Construisez un exemple et le testez-le en adaptant le programme `ex1.ys` .

Question 4

Même question avec la disjonction exclusive (`xorl`).

Étiquettes et branchements

Une instruction peut être précédée d'une *étiquette*, par exemple :

```
tag: addl %eax, %ecx
```

après quoi l'instruction

```
jmp tag
```

place dans le compteur de programme `PC` l'adresse de l'instruction correspondante. On peut faire aussi des sauts conditionnels en remplaçant `jmp` par `je` (*jump if equal*) ou `jne` (*jump if not equal*) : le saut est exécuté ou non suivant la valeur du code de condition `ZF`.

Exercice 2 : Test alternatif

Réalisez l'équivalent du code C suivant :

```
long a = 2, b = 3, c;
if (b <= a)
  c = 1;
else
  c = 2;
```