Université Paris-Nord — L1
Institut Galilée — Année 2008-2009
Licence 1 - 1er semestre

Travaux Pratiques 1 : Programmation en mini-assembleur

Il est demandé, pendant ce TP, d'écrire des programmes et de les exécuter pour résoudre des problèmes simples. Ces programmes sont écrits en mini-assembleur et exécutés sur un processeur simulé par le logiciel `amil` (*assembleur miniature pour l'informatique de licence*). Vous trouverez des détails sur amil sur la page web : `http://lipn.fr/~boudes/amil/`

1 Prise en main

1.1 Le terminal

Ouvrir un terminal et taper les lignes de commandes suivantes (appuyer sur Entrée à chaque ligne).

`~boudes/pub/EI/init.sh`	Exécuter un script qui modifiera légèrement votre environnement de travail pour l'adapter au TP.
`mkdir TP1`	Créer un répertoire TP1.
`cd TP1`	Entrer dans le répertoire TP1.
`amil &`	Lancer `amil`, le simulateur de mini-assembleur en *tâche de fond* (en tâche de fond le terminal ne restera pas bloqué jusqu'à l'arrêt du simulateur).

Astuces du terminal. la touche de tabulation vous permet de compléter votre saisie quand vous tapez une commande dans le terminal. La touche flèche vers le haut rappelle une ligne de commande tapée précédemment.

1.2 Emacs

Il va vous falloir un éditeur de texte pour taper votre premier programme amil. Emacs est un éditeur de texte puissant, mais un peu différent de ce que vous connaissez. Vous n'êtes pas obligé de l'employer. Les éditeurs gedit, kwrite ou kate sont des alternatives correctes.

Si vous voulez essayer Emacs, vous pouvez taper la commande suivante dans le terminal, sinon vous pouvez passer directement aux exercices d'assembleur.

`emacs exercice1.txt&`	Ouvrir un fichier `exercice1.txt` dans l'éditeur de texte emacs, en tâche de fond.

Tapez du texte ("bonjour emacs"), puis dans le menu `File` sélectionner `Save buffer` pour sauvegarder. Maintenant, essayez `Save buffer as...` (enregistrer le texte sous...), voyez vous où `emacs` taper le nom du fichier dans lequel enregistrer le texte ? Cette zone en bas de fenêtre s'appelle le mini-buffer. Taper `Ctrl-g` pour annuler une opération en cours dans le mini-buffer (si vous avez cliqué ailleurs, il faudra cliquer dans le mini-buffer avant). Une extension que nous

avons ajouté à Emacs permet d'afficher les numéros de ligne à gauche du texte en appuyant sur la touche F11.

Rappel des instructions

`stop`	Arrête l'exécution du programme.
`noop`	N'effectue aucune opération.
`saut i`	Met le compteur ordinal à la valeur i.
`sisaut ri j`	Si la valeur contenue dans le registre i est positive ou nulle, met le compteur ordinal à la valeur j.
`init x ri`	Initialise le registre i avec la valeur x.
`lecture i rj`	Charge, dans le registre j, le contenu de la mémoire d'adresse i.
`lecture *ri rj`	Charge, dans le registre j, le contenu de la mémoire dont l'adresse est la valeur du registre i.
`ecriture ri j`	Écrit le contenu du registre i dans la mémoire d'adresse j.
`ecriture ri *rj`	Écrit le contenu du registre i dans la mémoire dont l'adresse est la valeur du registre j.
`inverse ri`	Inverse le signe du contenu du registre i.
`add x rj`	Ajoute x au contenu du registre j.
`add ri rj`	Ajoute la valeur du registre i à celle du registre j.
`mult`, `div`, `et`	Même syntaxe que pour `add` mais pour la multiplication, la division entière et le et bit à bit.

2 Initialisation de la mémoire

Soit la case mémoire, x, d'adresse 10 et la case mémoire, y, d'adresse 11. Écrire et exécuter le programme qui initialise x à 7×2 et y à $x - 1$.

Correction.

```
1   init 7 r0
2   mult 2 r0
3   ecriture r0 10
4   add -1 r0
5   ecriture r0 11
6   stop
```

3 Exécution conditionnelle d'instructions

À l'aide de l'instruction `sisaut`, écrire les programmes correspondant aux algorithmes suivants et les exécuter avec amil sur un exemple, afin de tester leur correction :

1. Soient la valeur a à l'adresse 20, b à l'adresse 21. Si $a < b$ alors écrire a à l'adresse 22 sinon écrire b à l'adresse 3.

   Correction.
   – Calcul de $a - b$:

```
 1   lecture 21 r0
 2   inverse r0
 3   lecture 20 r1
 4   add r1 r0     # r0 vaut a - b
```
– Si $a < b$ (c'est à dire si $a - b < 0$)
```
 5   sisaut r0 9
```
– Alors écrire a à l'adresse 22
```
 6   lecture 20 r2
 7   ecriture r2 22
 8   saut 11
```
– Sinon écrire b à l'adresse 22
```
 9   lecture 21 r2
10   ecriture r2 22
11   stop
```
– Données :
```
20   3  # a
21   2  # b
22   ?  # résultat
```

2. Soient trois cases mémoires contenant trois entiers. Calculer et écrire le minimum de ces trois entiers en mémoire.

Correction.

– Soient a, b et c trois entiers
```
20   12 # a
21   23 # b
22   6  # c
23   ?  # min
```
– min est initialise à a (par défaut)
```
 1   lecture 20 r0
 2   ecriture r0 23
```
– Si $b <$ min alors min vaut b
```
 3   lecture 21 r0
 4   lecture 23 r1
 5   inverse r1
 6   add r0 r1    # r1 vaut b - min
 7   sisaut r1 9
 8   ecriture r0 23
```
– Si $c <$ min alors min vaut c
```
 9   lecture 22 r0
10   lecture 23 r1
11   inverse r1
12   add r0 r1 # r1 vaut c - min
13   sisaut r1 15
14   ecriture r0 23
```
– min contient le minimum de a, b et c
```
15   stop
```

4 Boucles d'instructions

1. Avec l'instruction `saut`, écrire un programme qui ne termine jamais.

   Correction.

   ```
   1   saut 1
   2   stop # <- jamais atteint
   ```

2. Avec l'instruction `sisaut`, écrire un programme qui ne termine jamais.

   Correction.

   ```
   1   init 0 r0
   2   sisaut r0 2
   3   stop # <- jamais atteint
   ```

On dit de ces programmes qu'ils bouclent à l'infini.