Université Paris-Nord — L1
Institut Galilée — Année 2011-2012
Licence 1 - 1er semestre

Travaux dirigés 1 : programmation en mini-assembleur.

L'objectif de ce TD est de vous familiariser avec le cycle d'exécution d'un processeur et avec la notion de flux d'instructions. Pour cela, il vous est demandé d'écrire de petits programmes dans le langage assembleur présenté en cours et de simuler leur exécution par le processeur.

Jeu d'instructions (simplifié)

`stop`	Arrête l'exécution du programme.
`noop`	N'effectue aucune opération.
`saut i`	Met le compteur de programme à la valeur i.
`sautpos ri j`	Si la valeur contenue dans le registre i est positive ou nulle, met le compteur de programme à la valeur j.
`valeur 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.
`ecriture ri j`	Écrit le contenu du registre i dans la mémoire d'adresse j.
`inverse ri`	Inverse le signe du contenu du registre i.
`add ri rj`	Ajoute la valeur du registre i à celle du registre j (la somme obtenue est placée dans le registre j).
`soustr ri rj`	Soustrait la valeur du registre i à celle du registre j (la différence obtenue est placée dans le registre j).
`mult ri rj`	Multiplie par la valeur du registre i celle du registre j (le produit obtenu est placé dans le registre j).
`div ri rj`	Divise par la valeur du registre i celle du registre j (le quotient obtenu, arrondi à la valeur entière inférieure, est placé dans le registre j).
Instructions plus avancées	
`et ri rj`	Effectue le et bit à bit de la valeur du registre i et de celle du registre j. Le résultat est placé dans le registre j.
`lecture *ri rj`	Charge, dans le registre j, le contenu de la mémoire dont l'adresse est la valeur du registre i
`ecriture ri *rj`	Écrit le contenu du registre i dans la mémoire dont l'adresse est la valeur du registre j.

1 Initialisation de la mémoire

Écrire les programmes répondant aux problèmes suivants :

1. Soit la valeur 8 contenue dans la case mémoire d'adresse 10. Recopier cette valeur à l'adresse 11.
2. Sans connaître la valeur contenue à l'adresse 10, écrire 7 à l'adresse 10.
3. Sans connaître la valeur contenue à l'adresse 10, incrémenter cette valeur de 1.
4. Soit une valeur quelconque, a, contenue à l'adresse 10. Initialiser la case d'adresse 11 à $a \times 2 + 1$.
5. Soit une valeur quelconque, a, contenue à l'adresse 10. Initialiser la case d'adresse 11 à $a \times (2 + 1)$.

2 Trace de programme assembleur

Nous allons désormais produire une représentation de l'exécution pas à pas de nos programmes. Une *trace* d'un programme assembleur sera un tableau dont chaque ligne correspond à l'exécution d'une instruction par le processeur. Une colonne contiendra le cycle d'horloge du processeur et une autre colonne le compteur de programme. Il y aura également une colonne par registre utilisé dans le programme et par case mémoire où des données sont lues ou écrites par les instructions du programme. Une première ligne, d'initialisation, montrera l'état de ces registres et cases mémoires avant le début du programme. Ensuite, chaque exécution d'instruction sera représentée par une nouvelle ligne du tableau, jusqu'à exécution de l'instruction `stop`. Dans cette ligne nous représenterons le cycle d'horloge du processeur (en commençant à 0 pour la ligne d'initialisation), la valeur du compteur de programme après exécution de l'instruction, et, s'il y a lieu, la valeur du registre ou de la case mémoire modifiée par le programme.

2.1 Exemple de trace

Soit le programme ci-dessous :

```
1   lecture 10 r0
2   valeur 5 r2
3   soustr r2 r0
4   sautpos r0 8
5   valeur 0 r0
6   ecriture r0 11
7   saut 9
8   ecriture r0 11
9   stop
10  14
11  ?
```

On représentera alors sa trace comme ceci :

Instructions	Cycles	CP	r0	r2	10	11
Initialisation	0	1	?	?	14	?
`lecture 10 r0`	1	2	14			
`valeur 5 r2`	2	3		5		
`soustr r2 r0`	3	4	9			
`sautpos r0 8`	4	**8**				
`ecriture r0 11`	5	9				9
`stop`	6	10				

La colonne *Instructions* n'est pas nécessaire, elle sert juste ici à la compréhension, sans cette colonne le mot-clé *initialisation* pourra être placé dans la colonne *Cycles*. Remarquez par exemple qu'il n'y a pas de colonne pour le registre 1, puisque celui-ci n'est pas utilisé.

2.2 Première trace

Faire la trace du même programme où la valeur 14 est remplacée par la valeur 2, à l'adresse mémoire 10.

Quelles sont les valeurs contenues dans les registres 0, 1 et 2 après arrêt sur l'instruction `stop` ?

3 Exécution conditionnelle d'instructions

À l'aide de l'instruction `sautpos`, écrire les programmes correspondant aux algorithmes [1] suivants et les exécuter sur un exemple (en faisant une trace) afin de tester leur correction :

1. Soient la valeur a à l'adresse 15, b à l'adresse 16. Si $a >= b$ alors écrire a à l'adresse 17 sinon écrire b à l'adresse 17.
2. Soit l'âge d'une personne à l'adresse 15. Si cette personne est majeure alors écrire 1 à l'adresse 16 sinon écrire 0 à l'adresse 16.

Soit la donnée x d'adresse 15. On veut écrire la valeur absolue de x à l'adresse 16.

1. Décrire un algorithme réalisant cette tâche.

1. Wikipédia : «Un algorithme est un processus systématique de résolution, par le calcul, d'un problème permettant de présenter les étapes vers le résultat à une autre personne physique (un autre humain) ou virtuelle (un calculateur). [...] Un algorithme énonce une résolution sous la forme d'une série d'opérations à effectuer. La mise en œuvre de l'algorithme consiste en l'écriture de ces opérations dans un langage de programmation et constitue alors la brique de base d'un programme informatique.»

2. Écrire un programme réalisant cette tâche.
3. Construire une trace de votre programme lorsque x vaut 5, puis lorsque x vaut -5.

4 Boucles d'instructions

4.1 Boucles infinies

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

4.2 Boucles de calculs itératifs

Soit un entier n d'adresse 15.

1. Écrire un programme qui lorsque $n \geq 0$, écrit la valeur n à l'adresse 16, puis, toujours à l'adresse 16, écrit la valeur $n-1$, puis $n-2$, et ainsi de suite jusqu'à écrire 0, après quoi le programme termine. Si n est négatif, le programme ne fait rien.
2. Décrire un algorithme calculant la somme des entiers 0, 1, ..., n. Par convention cette somme, $\sum_{i=0}^{n} i = 0 + 1 + \ldots + n$ est nulle lorsque $n < 0$. Écrire le programme correspondant. Le résultat de la sommation sera écrit à l'adresse 16.
3. (Optionnel) Tester votre programme en construisant sa trace pour $n = 3$.

4.3 Boucles de parcours (exercice facultatif)

Soient n entiers $x_1, \ldots, x_n$ rangées aux adresses $30+1, \ldots, 30+n$. La valeur $n > 0$ est elle même rangée à l'adresse 30.

1. Décrire un algorithme qui effectue la somme $\sum_{i=1}^{n} x_i = x_1 + \ldots + x_n$.
2. Écrire un programme réalisant cet algorithme. (Vous aurez besoin des instructions `lecture *ri rj` et `ecriture ri *rj`). Le résultat sera écrit à l'adresse $30+n+1$.
3. Écrire un algorithme qui trouve le plus petit des entiers parmi $x_1, \ldots, x_n$. En déduire un programme réalisant cette recherche du minimum. Le résultat sera écrit à l'adresse $30+n+1$.

Université Paris-Nord — L1
Institut Galilée — Année 2011-2012
Licence 1 - 1er semestre

Travaux Pratiques 1 : Programmation en mini-assembleur

Il est demandé, pendant ce TP, d'écrire des programmes simples en mini-assembleur et de les exécuter sur un processeur simulé en `amil` (*assembleur miniature pour l'informatique de licence*). Le simulateur peut être utilisé en ligne ici :
`http://www-lipn.univ-paris13.fr/~boudes/amilweb/`

Prise en main : le terminal

Faîtes vous aider par votre chargé de TP pour ajouter un lanceur de terminal dans votre barre d'outils ou votre barre de menu, ainsi qu'un lanceur pour l'éditeur de texte (`gedit`).

Dans un terminal, taper les lignes de commandes suivantes :

`mkdir TP1`	Créer un répertoire TP1.
`cd TP1`	Entrer dans le répertoire TP1.

Dans la suite, vous gagnerez à taper vos programme dans l'éditeur de texte, les charger dans le simulateur par copier/coller, et à ***les enregistrer*** *dans le répertoire TP1 (sous des noms comme* `exercice1.txt`*,* `exercice2_1.txt`*, etc.).*

`gedit exercice1.txt &`	Lance l'édition d'un fichier `exercice1.txt`, en *tâche de fond.*

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 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 $x - 1$.

2 Exécution conditionnelle d'instructions

À l'aide de l'instruction `sautpos`, 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 22.
2. Soient trois cases mémoires contenant trois entiers. Calculer et écrire le minimum de ces trois entiers en mémoire.

3 Boucles d'instructions

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

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