TP 3 – Assembleur MIPS : Entrées/sorties, branchements conditionnels et boucles

Téléchargez sur Moodle le simulateur «Mars4_5.jar» et les ressources tp3 contenant les fichiers à compléter.

1 Préliminaires

— Le nom d'un fichier contenant un programme assembleur doit obligatoirement avoir l'extention `.s`

— Le corpus d'un fichier assembleur a toujours le format suivant (ce qui suit le # est un commentaire) :

```
# Données
.data

.text
.globl __start

__start:
# corps du programme ...
# ...
# ...
j    Exit        # saut a la fin du programme

procedure:
# procedure...
# ...

Exit:                    # fin du programme
```

— L'émulateur **MARS** permet d'exécuter un programme assembleur MIPS. Pour lancer MARS, il faut utiliser la commande suivante : `java -jar Mars4_5.jar`

2 Entrées/Sorties

Les opérations d'entrées/sorties sont des traitements exécutés en mode noyau (c-à-d sous le contrôle du système d'exploitation). Un programme assembleur demandant une opération E/S doit alors utiliser un "appel système" grâce à l'instruction `syscall`.

1 – Le code assembleur (fichier `hello.s`) suivant affiche une chaîne de caractère sur la sortie standard grâce à un appel système.

```
.data

hello: .asciiz "Salut, mec!\n" # hello pointe vers la chaîne "Salut, mec!\n\0"

.text
.globl __start

__start:
la $a0 hello  # adresse de la chaîne à afficher
li $v0 4      # appel système 4: afficher une chaîne de caractère
syscall

j Exit #saut vers la fin du processus

Exit:
li $v0 10     # appel système 10: fin du programme
syscall
```

1. Éxecutez ce code avec `Mars`.
2. Quels registres sont utilisés pour quoi ?

De manière générale, lors de l'appel de `syscall`, un entier codant le type d'appel système demandé est passé en argument dans `$v0`, et les autres arguments sont passés dans les registres `$a0`, ... `$a3`.

Vous trouverez la liste complète des appels système disponibles avec `Mars` à l'adresse :
`http://courses.missouristate.edu/kenvollmar/mars/help/syscallhelp.html`.
Ceux qui seront utiles dans ce TP sont les appels :

— 1 : écrire un entier sur la sortie standard,
— 4 : écrire une chaîne de caractères sur la sortie standard,
— 5 : lire un entier sur l'entrée standard.

2 – Réalisez un programme MIPS `entier.s` qui demande à l'utilisateur de saisir un entier, lit l'entier, et l'affiche sur la sortie standard.
3 – Réalisez un programme MIPS `somme.s` qui demande à l'utilisateur de saisir deux entiers et affiche leur somme.

3 Sauts et Procédures

Les instructions de sauts longs (*jump*) et courts (*branch*) permettent de sauter le nombre d'instructions passé en arguments. Les arguments des sauts longs (instructions `j` et `jal`) sont codés sur 26 bits signés, et permettent de réaliser des sauts de $2^{25} - 1$ instructions en avant ou en arrière. En pratique, on n'a pas à gérer ce décalage, il suffit de faire précéder le code à exécuter après le saut par `label:` et d'écrire l'instruction `j label` (ou `jal label`). Contrairement à l'instruction `j`, l'instruction `jal` (*jump and link*) sauvegarde l'adresse de l'instruction suivante (celle se trouvant juste après `jal label`) dans le registre `$ra`. Le code `jr $ra` à la fin d'une procédure réalise un saut vers l'instruction dont l'adresse est sauvegardée dans `$ra`.

Le code assembleur MIPS suivant présente l'utilisation des instructions `j`, `jal` et `jr`.

```
...
__start:
...
jal Maprocedure #saut vers Maprocedure avec sauvegarde de l'instruction suivante dans $ra
...
j Exit          #saut vers Exit sans sauvegarde de l'instruction suivante dans $ra

Exit:
li $v0 10
syscall

Maprocedure:    #code de Maprocedure
...
jr $ra          #retour à l'adresse de l'instruction sauvegardée dans $ra
```

4 – Reprenez votre programme `somme.s` afin d'y insérer une procédure affichant sur la sortie standard un entier passé dans le registre $a0.
5 – Éxecutez pas à pas le programme assemblé `somme.s`, et observez les arguments des instructions `j` et `jal` dans la zone text du processus (colonnes Code et Basic dans l'émulateur `Mars`).

— Par quel code est traduite l'instruction `jal Maprocedure` ? Décomposez l'instruction en (`opcode`,`argument`).
— En vous rappelant que les adresses sont alignées sur des mots de 4 octets, dîtes ce qu'est ici l'argument. Comment l'expliquez-vous ?

4 Branchements conditionnels et boucles

Les sauts courts peuvent être inconditionnels (instruction `b`), ou conditionnels (instructions `beq`, `bne`, `bgt`, `bge`, `blt`, `ble`). Les arguments des sauts courts sont codés sur 16 bits signés, et permettent de réaliser des sauts de $2^{15} - 1$ instructions en avant ou en arrière.

Le code assembleur MIPS suivant teste si l'entier (fichier `nul.s`) dans le registre `$t0` est nul :

```
.data

nul: .asciiz "Ton entier est nul!\n"

.text
.globl __start

__start:

...

beq $t0 $0 Nul  #si $t0 = $0 va à Nul
j Exit          #saut vers la fin du processus

Nul:
```

```
la $a0 nul  #adresse de la chaîne à afficher
li $v0 4    #appel système 4: afficher une chaîne de caractère
syscall

Exit:
li $v0 10
syscall
```

6 – Comprenez le fonctionnement de ce code (vous pouvez ouvrir, assembler et exécuter le fichier `nul.s` depuis `Mars`).

7 – Réalisez un programme MIPS `maximum.s` qui demande à l'utilisateur de saisir deux entiers et affiche le maximum entre ces deux entiers.

(Rappel :
- `beq` : branch on equal,
- `bne` : branch on not equal,
- `blt` : branch on less than,
- `ble` : branch on less or equal,
- ...)

8 – Comment réaliser une boucle grâce à ces instructions de branchement conditionnels ?

9 – Réalisez un programme assembleur MIP `multiples.s` affichant à l'écran les n premiers multiples d'un entier A (n et A sont lus depuis le clavier).