TP 3 - La pile : appel de fonctions

On utilise le simulateur QtSpim. Les programmes sont disponibles sur le site du cours.

Les procédures et les fonctions sont des objets couramment utilisés en programmation. Lorsque le compilateur d'un langage de haut niveau (tel que le c) doit convertir le code c en code assembleur, il doit veiller à enregistrer les informations qui lui permettront de revenir dans son état initial après l'appel de la fonction. Il faudra donc sauvegarder également les variables locales. Il utilise pour cela la pile (ou stack en anglais) afin d'enregistrer l'état des registres courants ainsi que l'adresse où revenir à la fin de la fonction.

Le compilateur sauvegarde la position courante du registre « stack pointer » (qui correspond à R29 ou sp sur qtspim) lui permettant de se souvenir de l'état de la pile mais également le registre Ra qui lui indique « l'endroit » où revenir. Les variables locales à la procédure sont également allouées dans la pile afin d'être réutilisable à la suite d'un appel de fonction et elles seront détruites lorsque l'appel sera terminé. Par ailleurs, lorsque la procédure modifie certains registres, elle les sauvegarde pour pouvoir les restaurer avant de revenir à l'appelante.
Attention : MIPS utilise aussi le registre $fp pour se servir de la pile. Comme cet aspect n'a pas été vu en cours, on supposera que seulement le registre sp est utilisé et on ignorera les « move $fp,$sp » et considérera que les instructions suivantes utilisent sp.

Dans le cas où un programme contient plusieurs fonctions, la fonction « main » ne commence pas après l'instruction syscall mais juste après l'appel : « jal 0x..... [main] ».

1. Découvert et appel simple de sous-routine

a. Regarder le fichier source (en c) TP3Prog1.c ainsi que la version MIPS TP3Prog1.s. Comment est appelée la fonction foo en MIPS (on recopiera **l'unique** instruction effectuant l'appel en expliquant précisément ce que fait l'instruction) ?

b. Que se passe-t-il sur le registre sp (ou R29) au début du main et avant la fin du main (avant "jr $ra") ? Pourquoi ?

c. Monitorer le registre ra (R31) juste avant l'appel de la fonction foo et juste après son appel (écrire son état avant et après). Pouvez-vous expliquer sa nouvelle valeur ?

d. A quoi sert “jr $ra” ?

e. A quelle adresse du programme se trouve la fonction foo ? (vous justifiez la réponse à l’aide d’une capture d’écran)

f. Trouver dans la fonction main, où est effectuée l’instruction « int i=2 » (indiquer la ou les instructions).

g. Trouver dans la fonction main, où est effectuée l’instruction « int j=5 » (indiquer la ou les instructions).

h. Comparer maintenant les fichiers TP3Prog1.c et TP3Prog1m.c d’une part, et ensuite les fichiers TP3Prog1.s et TP3Prog1m.s correspondant au code MIPS des fichiers sources en c. Quelle différence observez-vous dans le code MIPS ? Que peut-on en conclure sur la convention de compilation de gcc pour du code en c concernant la déclaration des variables ?

2. Paramètres et valeur retournées par une fonction

En MIPS, comme nous l'avons vu plus haut, certains registres sont conventionnellement utilisés pour certaines tâches. On s'intéresse maintenant au code c et MIPS écrit dans TP3Prog2.c et TP3Prog2.s.

a. En analysant le code MIPS de la fonction main, trouver deux registres utilisés pour passer les arguments d'une fonction.

b. En s'intéressant maintenant au code MIPS de la fonction bar, comment sont récupérés les arguments de la fonction bar ? (on écrira les instructions)

c. Identifier maintenant comment MIPS récupère la valeur retournée par la fonction, quel est le registre utilisé ? (écrire la ou les instructions en les commentant et spécifier le registre utilisé)

d. Comparer maintenant les fichiers TP3Prog2.c et TP5Prog2m.c d'une part, et ensuite les fichiers TP3Prog2.s et TP5Prog2m.s correspondant au code MIPS des fichiers sources en c. Comment se fait le passage des paramètres à la fonction bar cette fois ci ?

e. Recopier les instructions MIPS qui permettent le passage des paramètres (avant l'appel de la fonction, et au début de l'appel), en justifiant votre réponse.

f. Quelle(s) différence(s) verrait-on si jamais on modifiait l'argument "arg1" à l'intérieur de la fonction bar dans les deux cas (TP3Prog2.c et TP3Prog2m.c) ?

3. Ecrire un appel de fonction

On considérera le code suivant :

```
int min(int i, int j) {
   if(i<j) return i ;
   else return j ;
}

void main() {
   int i=2 ;
   int j=5 ;
   int c = min(i,j) ;
}
```

Le code MIPS TP3Prog3.s contient le squelette du programme à compléter. Reportez le code complété ci-dessous.
On fera passer les arguments et la valeur de retour de la fonction min en **passant par les registres** (comme vu précédemment). **On ne s'occupera pas ici du registre fp !**
ATTENTION : les endroits où remplir le programme peuvent contenir plus d'une ligne!

Instructions	Commentaires

4. Fonction récursive (facultatif)

On regarde maintenant le code TP3Prog4.c et sa version MIPS TP3Prog4.s.

a. Quelles sont les rôles de L2 et L3 ?

b. Quelle taille en mémoire occupe ce programme ?