TD-TP 2 : Appel de fonctions

Le but de cette séance est d'apprendre à programmer et appeler des fonctions en langage d'assemblage, et de comprendre les conventions de gestion des registres et de la mémoire imposées par une ABI (Application-Binary Interface : ensemble de conventions permettant aux différentes parties du programme de communiquer, quelques-soient leurs langages de développement respectifs) et la bibliothèque C que l'on utilise pour écrire nos programmes en langage d'assemblage. En respectant ces conventions, on verra qu'il est possible de mélanger du code C et du langage d'assemblage sans difficulté dans le même programme.

L'ABI de l'architecture MIPS est un document complexe à lire [1] qui a pour but de définir les règles que doivent respecter les compilateurs et assembleurs, dans toutes les situations possibles, y compris certaines dépassant largement le but de ce cours. Les conventions présentées en cours et utilisées par la suite sont donc un sous-ensemble volontairement simplifié des conventions détaillées dans l'ABI, mais elles ont été sélectionnées de façon à être compatibles avec l'ensemble complet.

Pour ce TP, vous avez besoin de récupérer les sources mises à disposition sur Chamilo. Comme dans le dernier TP, chaque programme est composé d'un fichier en C et d'un fichier en langage d'assemblage.

Ex. 1 : Compréhension d'exemples détaillés

Le but de cet exercice est de vérifier et consolider la compréhension des conventions utilisées pour réaliser des appels de fonctions [2]. Cet exercice vous montre aussi les bonnes manières d'écrire du code en langage d'assemblage (Commentaires précis et complet sur le contexte de la fonction, utilisation de commentaires pour indiquer les instructions C traduites). Veillez à choisir des valeurs de paramètres pertinentes pour pouvoir tracer l'exécution des programmes. Quand les arguments sont à saisir, ils le sont dans la fenêtre exécutant mips-qemu.

Question 1 Ouvrez le fichier `fct_pgcd.s` et observez la traduction en langage d'assemblage proposée. Justifiez le choix du contexte. Exécutez ensuite pas à pas ce programme avec GDB et le simulateur QEMU.

Pour la question suivante, pour comprendre comment on manipule une variable locale dans la pile, on va placer exceptionellement la variable locale dans la pile même si c'est une fonction feuille.

Question 2 Même question que la question 1 avec le fichier `fct_mult.s`.

On placera les variables locales dans la pile que lorsque c'est nécessaire (dû à un appel de fonction qui peut écraser la valeur de la variable, fonction non-feuille) ou si c'est indiqué dans un contexte imposé.

Question 3 Même question avec le fichier `fct_fibo.s`. Dessinez la pile lorsque fibo(0) est

1. Les curieux la trouveront sur Chamilo dans la partie documents Ressources_externes_et_annales
2. en suivant l'ABI SystemV pour MIPS.

appelée la première fois pour le calcul de fibo(4). Vérifiez ce dessin en observant dans GDB le contenu de la pile avec la commande `x /16x $sp`.

Ex. 2 : Exercice de traduction systématique : C vers langage d'assemblage

Dans les questions suivantes, on vous demande traduire du code C en langage d'assemblage en respectant les règles suivantes :

- préciser le contexte,
- traduire de manière systématique les instructions C (Pour chaque instruction C, il faut récupérer les données aux emplacements désignés par le contexte. On ne cherche donc pas à optimiser le code en récupérant un calcul fait dans la traduction d'une instruction C précédente)
- commenter au minimum avec le code C traduit

Pour chaque traduction, tester et/ou déboguer avec les fichiers C fournis.

Question 1 Traduisez et testez la fonction age donnée en commentaires dans le fichier `fct_age.s`.

Question 2 Traduisez et testez la fonction hello donnée en commentaires dans le fichier `fct_hello.s`.

Question 3 Traduisez et testez la fonction affine donnée en commentaires dans le fichier `fct_affine.s`.

Question 4 Traduisez et testez la fonction fact donnée en commentaires dans le fichier `fct_fact.s`.

Pour aller plus loin... **Question 5** Traduisez et testez l'appel à la fonction C suggéré dans le fichier `fct_fact.s`.

Pour aller plus loin... **Question 6** Traduisez et testez la fonction val_binaire donnée en commentaires dans le fichier `fct_valbin.s`.