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). Ce sont les conventions imposées par cette interface qui nous permettent de mélanger du code écrit en C avec du code écrit directement en langage d'assemblage dans le même programme.

L'ABI du processeur RISCV est un document complexe à lire, car il fait des hypothèses sur la comprehension des mécanismes liés à la compilation à partir des langages de haut niveau [1]. L'ABI 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 dans le cours et utilisées par la suite constituent donc un sous-ensemble volontairement réduit des conventions détaillées dans l'ABI complète. Elles ont été sélectionnées de façon à correspondre aux besoins spécifiques de ce cours mais elles restent néanmoins compatibles avec l'ABI complète.

Les fichiers fournis pour ce TP se trouvent dans le répertoire `tp2/` de votre dépôt git. Comme dans le TP précédent, chaque programme est composé d'un fichier en C et d'un fichier en langage d'assemblage ; les commandes `make`, `qemu-system-riscv32`, `riscv32-unknown-elf-gdb` et `../common/verif_etud.sh` doivent se lancer depuis le répertoire `tp2/` de votre dépôt git.

Ex. 1 : Compréhension d'exemples détaillés

Le but de cet exercice est de vérifier et de consolider la compréhension des conventions utilisées pour réaliser des appels de fonctions en suivant l'ABI RISCV. Cet exercice vous montre aussi les « bonnes » manières d'écrire du code en langage d'assemblage dans le cadre de ce cours (commentaires précis et complets sur le contexte de la fonction, utilisation de commentaires pour indiquer les instructions C traduites). Veillez à choisir des valeurs pertinentes de paramètres pour pouvoir tracer l'exécution des programmes. Si un programme de test attend des arguments, ils sont à définir dans le fichier `.c`.

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. Rappel : il y a une vidéo de présentation de la méthode.

Pour la question suivante, afin de comprendre comment on manipule une variable locale dans la pile, on va imposer le placement de la variable locale dans la pile, même si pour une fonction feuille on préfère en général la mettre dans un registre.

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

Dans la suite, on placera les variables locales dans la pile uniquement lorsque c'est nécessaire

1. Les curieux la trouveront sur https://github.com/riscv-non-isa/riscv-elf-psabi-doc/blob/master/riscv-cc.adoc.

(dû à un appel de fonction qui peut écraser la valeur de la variable, donc lorsque la fonction traduite est 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(4)` est appelée la première fois pour le calcul de `fibo(8)`. Vérifiez ce dessin en observant dans GDB le contenu de la pile avec la commande `x /16x $sp`.

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

Dans les questions suivantes, on vous demande de 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) et en particulier on va systématiquement rechercher les variables C là où elles se trouvent (pile, section `.data` ou `.bss`, ou tas) ;
- commenter avec au minimum 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`.

À partir de ce point, on s'autorise un peu plus de souplesse sur le respect du contexte : si un élément du contexte n'est pas utilisé dans la suite, alors il n'y a pas besoin que le contexte la concernant soit respecté. Autrement dit : si on a besoin d'une variable, elle doit se trouver à l'endroit indiqué par le contexte. Ceci restera vrai pour les prochaines séances de TP.

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 la modification de la fonction `fact` précédente en `fact_papl` décrite dans le fichier `fct_fact_papl.s`.

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