Pablo Rauzy
pr@up8.edu
https://pablo.rauzy.name/teaching/liec

UNIVERSITÉ PARIS 8 VINCENNES-SAINT-DENIS

Langages : interprétation et compilation

Université Paris 8 – Vincennes à Saint-Denis
UFR MITSIC / L3 informatique

Séance a (TP) : Premiers pas en MIPS avec SPIM

N'oubliez pas :

- Les TPs doivent être rendus par courriel au plus tard le lendemain du jour où ils ont lieu avec "[liec]" suivi du numéro de la séance et de votre nom dans le sujet du mail, par exemple "[liec] TPa Rauzy".
- Quand un exercice demande des réponses qui ne sont pas du code, vous les mettrez dans un fichier texte `reponses.txt` à rendre avec le code.
- Le TP doit être rendu dans une archive, par exemple un tar gzippé obtenu avec la commande `tar czvf NOM.tgz NOM`, où `NOM` est le nom du répertoire dans lequel il y a votre code (idéalement, votre nom de famille et le numéro de la séance, par exemple "rauzy-tpa").
- Si l'archive est lourde (> 1 Mo), merci d'utiliser `https://bigfiles.univ-paris8.fr/`.
- Les fichiers temporaires (si il y en a) doivent être supprimés avant de créer l'archive.
- Le code doit être proprement indenté et les variables, fonctions, constantes, etc. correctement nommées, en respectant des conventions cohérentes.
- Le code est de préférence en anglais, les commentaires (si besoin) en français ou anglais, en restant cohérent.
- **N'hésitez jamais à chercher de la documentation par vous-même sur le net!**

Dans ce TP :

- Premiers pas en assembleur MIPS et avec l'outil SPIM.

Exercice 0.

Récupération des fichiers nécessaires.

1. Pensez à organiser correctement votre espace de travail, par exemple tout ce qui se passe dans ce TP pourrait être dans `~/liec/sa-tp/`.
2. Récupérez les fichiers nécessaires depuis la page web du cours, ou directement en ligne de commande avec `wget https://pablo.rauzy.name/teaching/liec/seance-a_tp.tgz`.
3. Une fois que vous avez extrait le dossier de l'archive (par exemple avec la commande `tar xzf seance-a_tp.tgz`), renommez le répertoire en votre nom (avec la commande `mv liec_seance-a_files` *`votre-nom`*). Si vous ne le faites pas tout de suite, pensez à le faire avant de rendre votre TP.
4. Vous êtes encouragé à tester systématiquement votre code après chaque modification de code, cela vous aidera à repérer les erreurs au plus tôt et donc le plus précisément possible.

Exercice 1.

Prise en main de SPIM.

1. Par la suite, vous pourrez si vous le souhaitez utiliser une interface graphique pour SPIM, mais pour le moment on va l'utiliser via l'outil en ligne de commande `spim`.
   → Lancez `spim` et exécutez la commande `help`.
2. → Lancez la commande permettant d'afficher l'état de tous les registres.
3. → Lancez la commande `run`. Que se passe-t-il ?
4. → Chargez en mémoire le fichier "main.s" qui vous est fourni.
5. → Lancez à nouveau la commande `run`. Que se passe-t-il ?
6. Examinons le fichier "main.s" ensemble.
   La première ligne signale que nous somme dans le segment "text", c'est à dire dans le code (par opposition au segment "data" qui contient des données, tout cela sera plus clair plus tard ne vous inquiétez pas).
   La seconde ligne déclare le symbole `main` comme global.
   La ligne suivante est vide, et celle d'après contient une *étiquette* (ou un *label*) `main`.
   Enfin la dernière ligne contient une instruction "`jr $ra`" qui demande de sauter à l'adresse contenu dans le registre (`jr` pour "Jump Register") `$ra` (pour "Return Address") qui contient l'adresse de retour de la fonction (c'est à dire l'adresse de l'instruction suivant celle qui a fait l'appel à notre fonction `main`).
   → Que s'est-il passé lorsqu'on a lancé la commande `run` pour la seconde fois ?
7. On veut maintenant exécuter du code étape par étape pour comprendre ce que font certaines instructions.

(a) → Essayez de charger en mémoire le fichier “basic.s”. Que se passe-t-il ?

(b) → Réinitialisez la mémoire et les registres puis chargez en mémoire le fichier “basic.s”.

8. SPIM fonctionne un peu comme un débuggueur. Vous ne serez donc pas surpris par ce qui va suivre si vous êtes un petit peu familier avec `gdb` par exemple.
   → Une fois le fichier “basic.s” chargé, positionnez un breakpoint (commande `breakpoint`) sur le label `main`, et lancez l’exécution (`run`).
9. Une fois le breakpoint atteint, on va répéter en boucle les opérations suivantes :
   — avancer le programme d’une instruction (`step`),
   — afficher le contenu des registres pour voir ce qu’à fait l’instruction précédente.
   → Expliquez chacune des instructions du `main` de “basic.s”.

Exercice 2.

Premiers programmes.

1. Le fichier “hello.s” contient le classique programme “Hello, World !”, en assembleur MIPS.
   Comme les exemples précédent, il commence par un segment de code (`.text`) qui contient une fonction `main`.
   Cette fois-ci par contre, il y a aussi un segment de données (`.data`).
   Ce segment contient un label (un nom associé à une adresse en mémoire), le type de donnée présent à cette adresse, et la valeur de la donnée.
   Ici la donnée est une chaîne de caractères ASCII terminée par un 0 (comme en C), ce qui se note `.asciiz`, et la valeur de cette chaîne est `"Hello, world!\n"`.
   Du côté du code, on retrouve `syscall`, mais cette fois-ci le registre `$v0` contient la valeur 4 quand l’instruction `syscall` est exécutée, qui correspond à `print_string` (dans “basic.s”, la valeur 1 correspondait à `print_int`). La valeur du registre `$a0` est l’adresse (l’instruction `la` signifie “Load Address”) de la chaîne de caractère dans le segment de données.
   → Chargez et exécutez ce programme.
2. Un autre code d’appel système utilisable avec l’instruction `syscall` est la valeur 5, pour `read_int`.
   → Ouvrez le fichier “add.s” et décrivez ce qui se passe dedans. Comment récupère-t-on la valeur retournée par l’appel à `read_int` ?
3. Intéressons-nous maintenant au fichier “funcall.s”. Il contient le même programme “add.s” mais déplace le contenu du main dans une fonction `add_user_num` qui est appelée dans le main.
   → Chargez et exécutez ce programme. Que constatez-vous ?
4. → Expliquez ce comportement (aidez-vous des fonctionnalités de debug si nécessaire).
5. → Proposez une correction pour le programme “funcall.s”.
6. → Quelles sont les limites de la solution que vous avez proposée ?
7. → Avez-vous une idée de comment remédier à ces limitations ?
8. On regarde maintenant le fichier “loop.s”.
   → Expliquez le fonctionnement de la boucle.

Exercice 3.

Votre premier programme.

1. → Écrire un programme “sum.s” qui calcule et affiche la somme des n premiers entiers, où n est demandé à l’utilisateur.