TP numéro 1 : Assembleur RISC-V

Assembleur – Compilation, ENSIIE

Semestre 3, 2023–24

On utilisera le simulateur `jupiter` (https://github.com/andrescv/Jupiter) pour tester les programmes, en particulier pour regarder l'évolution des registres.

Ressources utiles :

- la documentation de `jupiter` : https://jupitersim.gitbook.io/jupiter/
- la dernière spécification officielle de RISC-V peut se trouver à cette adresse : https://riscv.org/technical/specifications/
  À l'heure actuelle, il s'agit du document suivant : https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf
  En particulier :
  p.137 : noms des 32 registres et usage conventionnel
  p.17–28 : descriptions des instructions de la base RV32I
  p.43–45 : descriptions des instructions de l'extension M
  p.139–140 : pseudo-instructions standard
- appels système fournis par le simulateur : https://jupitersim.gitbook.io/jupiter/assembler/ecalls
- directives du langage assembleur supportées par le simulateur https://jupitersim.gitbook.io/jupiter/assembler/directives
- la table de correspondance ASCII : https://fr.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange#Table_des_128_caract%C3%A8res_ASCII
  En particulier, le code 10 pour le retour à la ligne.

Pour débuter on lancera `jupiter` avec la commande

```
/pub/FISE_ASCO23/image/bin/jupiter
```

ce qui permettra de visualiser l'évolution des registres, du programme, de la mémoire, etc.

Après avoir écrit son programme, on peut l'assembler avec [F3] puis l'exécuter pas à pas ou en entier.

Exercice 1 : Programmes basiques

1. Écrire un programme qui échange le contenu des registres `a0` et `a1`.

2. Proposer un programme qui, étant des entiers contenus dans les registres `a0`, `a1` et `a2`, teste s'ils peuvent représenter les longueurs des côtés d'un triangle rectangle d'hypoténuse de longueur `a2`, d'abord en affectant le registre `a0` à 1 ou 0 suivant le cas (0 si ça ne marche pas), puis en affichant un message explicite (utilisation d'un appel système).
3. Proposer un programme qui, étant donnés des entiers contenus dans les registres `a0`, `a1` et `a2`, teste s'ils peuvent représenter les longueurs des côtés d'un triangle rectangle.
4. Proposer un programme qui affiche n étoiles puis un retour à la ligne, où n est le contenu du registre `a0`.
5. Transformer ce programme en fonction en ajoutant une étiquette à la première instruction (par exemple `etoile:`) et en ajoutant `ret` à la fin pour revenir au code qui aura appelé la fonction.
6. Écrire un programme qui affiche un triangle décroissant d'étoile de la forme :

   ```
   *******
   ******
   *****
   ****
   ***
   **
   *
   ```

   où la première ligne contient autant d'étoile que le contenu du registre `t0`.
7. Écrire un programme qui implémente la fonction `atoi` de C : en supposant que `a0` contient l'adresse d'une chaîne de caractère `s`, met dans `a0` la conversion en entier du plus grand préfixe de `s` qui ne contient que des chiffres. Par exemple sur `"234blabla123"` le registre `a0` contiendra 234.

Exercice 2 : Calculatrice à pile

On souhaite implémenter une calculatrice utilisant une pile, un peu similaire au programme `dc`. On va lire les commandes ligne par ligne. Si la commande `s` commence par :

— un chiffre, on empile l'entier correspondant au plus grand préfixe de `s` qui ne contient que des chiffres en mémoire dans une pile ;
— `+`, on dépile deux entiers, et on rajoute leur somme dans la pile ;
— `p`, on dépile un entier que l'on affiche ;
— `q`, on sort du programme.

Par exemple, si on lit successivement

```
1
2
3
+
p
4
+
p
```

on affichera deux fois 5.

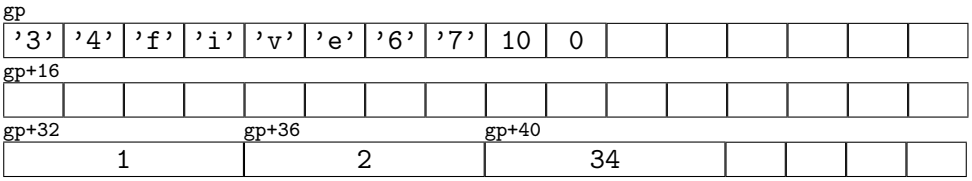
Il va donc falloir représenter la pile dans la mémoire. Dans les systèmes basés sur des processeurs RISC-V, la mémoire est en général découpée en trois partie : la première contient les instructions du programme, la deuxième les données globales et les données dynamiques (c'est le tas) et la dernière la pile d'appel (qui ne nous concerne donc pas encore ici). Nous allons donc utiliser le tas, qui peut être accédé à l'aide du registre `gp` dont le contenu est une adresse a située au milieu de ce tas.

Nous allons utiliser la zone mémoire de a à $a+31$ pour stocker une ligne de caractères entrée par l'utilisateur (appel système `Read String` de code 8) ; puis les adresses à partir de $a+32$ pour stocker les valeurs contenues dans la pile.

Par exemple, après avoir lues les lignes

```
1
2
34five67
```

le contenu de la mémoire sera le suivant :



Écrire le programme en assembleur RISC-V qui implémente cela.