Projet MIPS 2022

Projet de programmation C : Émulation MIPS

Last update : 16 Octobre 2022.

Le projet est à réaliser sous Linux par groupes de 2 élèves.

Ce sujet est largement inspiré du projet d'informatique de Phelma, 2ème année (François Portet, Nicolas Castagne, Mathieu Chabanas, Laurent Fesquet).

Description du sujet

Un émulateur est un logiciel capable de reproduire le comportement d'un objet, ici le processeur MIPS exécutant un programme. Dans ce projet, on reproduit la mémoire et les registres de la machine MIPS, et leur évolution au fur et à mesure que les instructions du programmes s'exécutent. L'émulateur doit fournir les mêmes résultats que ceux qu'on obtiendrait en lançant le programme sur une vraie machine MIPS.

Le processeur MIPS, son architecture et ses instructions sont présentés en annexe (liens à la fin du sujet).

Deux modes de fonctionnement de l'émulateur sont prévus:

- Le *mode interactif* dans lequel chaque instruction MIPS est lue au clavier (tapée par l'utilisateur dans le terminal) et exécutée directement. Ce processus continue jusqu'à ce que l'utilisateur tape la commande spéciale "EXIT" pour quitter l'émulateur.
- Le *mode fichier* dans lequel l'émulateur lit les instructions dans un fichier écrit à l'avance donc le nom est indiqué sur la ligne de commande.

Il est important de noter qu'une étape intermédiaire de cet émulateur est *l'assemblage* du fichier d'entrée.

Interface en ligne de commande

L'émulateur sera appelé en tapant la commande `emul-mips`, et il aura plusieurs comportements possibles (dépendant du nombre d'arguments à la ligne de commande):

Mode interactif

- Commande: `./emul-mips` (sans arguments)

Dans ce mode, le programme demande à l'utilisateur d'entrer une instruction dans le terminal, l'exécute ; en demande une nouvelle, et ainsi de suite jusqu'à lecture de "EXIT". Dans ce mode, il ne sera possible d'exécuter que des instructions en séquence.

Mode fichier

- Commande 1: `./emul-mips PROGRAMME -pas` (pas-à-pas)
- Commande 2: `./emul-mips PROGRAMME SORTIE_ASSEMBLAGE SORTIE_EXECUTION` (automatique)

Dans ce mode, `PROGRAMME` est le nom d'un fichier contenant le programme à exécuter, écrit en assembleur MIPS. L'émulateur ne demande pas à l'utilisateur de saisir du code.

Si l'option `-pas` est spécifiée, le programme est exécuté pas-à-pas : l'émulateur fait une pause à chaque instruction et l'utilisateur valide le passage à l'instruction suivante dans le terminal.

Sinon, l'utilisateur fournit deux noms de fichiers en plus : `SORTIE_ASSEMBLAGE` est le nom d'un fichier où sera stocké le programme assemblé (un fichier texte contenant des chaînes hexadécimales), et `SORTIE_EXECUTION` est le nom d'un fichier où sera stocké l'état final du programme (aussi un fichier texte avec un format fixé, voir ci-dessous).

Comportement attendu

Dans les deux modes, les instructions MIPS sont fournies dans le langage assembleur, une par ligne. Par exemple :

```
addiu $v0, $zero, 10
```

Le premier travail de l'émulateur est de traduire les instructions sous leur forme binaire. Cette forme binaire sera affichée en hexadécimal (en texte), soit dans le terminal (mode interactif ou fichier pas-à-pas) soit dans le fichier `SORTIE_ASSEMBLAGE` (mode fichier automatique). Par exemple, l'instruction précédente a la forme hexadécimale :

```
2402000a
```

Il vous est uniquement demandé de traduire les instructions ; on suppose que les programmes ne contiennent pas d'étiquettes (comme "etiq1:") ni de directives (comme ".space 10").

Le second travail de l'émulateur est d'exécuter les instructions. L'émulateur traque l'état de l'architecture MIPS (valeurs stockées dans les registres et la mémoire) et les modifie à chaque exécution d'instruction. Votre priorité est que ces modifications soient conformes à la spécification du processeur MIPS.

En mode interactif et fichier pas-à-pas, l'émulateur affiche l'état des registres et de la mémoire avant de passer à l'instruction suivante. En mode fichier automatique, il enregistre uniquement l'état final dans le fichier `SORTIE_EXECUTION`. Le format de ce fichier est fixé pour permettre aux scripts de tests/notation de fonctionner.

Système de tests unitaires

Le système de tests vous est fourni; tapez `make tests` pour lancer les tests.

Chaque fichier assembleur (`*.s`) dans le `tests/` est un programme de test ; un exemple `arithmetic.s` est fourni. En plus du code, il y a deux blocs de commentaires :

- Un bloc `EXPECTED_ASSEMBLY` dans lequel on écrit le code hexadécimal correspondant aux instructions. Le système de tests vérifiera que `emul-mips` produit bien ce code.
- Un bloc `EXPECTED_FINAL_STATE` dans lequel on écrit la valeur que doivent avoir les registres `$00` à `$31` ainsi que `HI` et `LO` à la fin de l'exécution. Les registres qui valent 0 à la fin de l'exécution n'ont pas besoin d'être listés. Le système de tests vérifiera que `emul-mips` produit bien cet état final.

Pour que le système fonctionne, il est important que `emul-mips` produise des fichiers `SORTIE_ASSEMBLAGE` et `SORTIE_EXECUTION` que le système de test sait lire. Spécifiquement :

- `SORTIE_ASSEMBLAGE` doit contenir une instruction par ligne.
- `SORTIE_EXECUTION` doit contenir la valeur de chaque registre, un par ligne, sous la forme `<registre>: <valeur>`, par exemple `$03: -5`. Les registres à afficher sont `$00`, `$01`, ..., `$31`, `HI` et `LO`.

Construisez vos tests sur ce modèle. Vous pouvez ne spécifier qu'un seul des deux blocs.

Organisation du travail et rendus

Il vous est demandé d'effectuer dans l'ordre les tâches suivantes. Les dates de rendu sont sur Chamilo dans la section "Travaux" du cours, pour chaque groupe.

Rendu #1: CLI (à rendre à la 1ère séance)

Récupérez l'archive `CS351_2022_ProjetMIPS.zip` avec les fichiers initiaux du projet.

1. Dans le `Makefile` et dans `src/main.c`, indiquez aux premières lignes votre nom et celui de votre binôme.
2. Programmez la lecture des arguments en ligne de commande en suivant les indications données dans `src/main.c`. Une fois que `make test-cli` n'affiche plus d'erreur c'est bon.
3. `make tests` doit vous envoyer une erreur à propos du nombre d'instructions générées (aucune !) au lieu d'une erreur à propos de fichiers inexistants.
4. Répondez aux questions dans `README.md` et suivez les instructions de rendu sur Chamilo.

Ce rendu a pour objectif de vérifier que vous savez rendre votre projet sous une forme conforme à notre infrastructure profs de correction/notation automatique. **La pondération de cette note sera minime mais les rendus non conformes auront la note minimale**.

Rendu #2: assemblage (rendu indicatif : 3ème séance)

1. Écrire des programmes MIPS qui vous serviront à tester votre application dans le dossier `tests/`. Quelques exemples sont fournis. Il est important de bien comprendre le rôle de chacune des instructions utilisées ! Cette étape est essentielle car c'est là que vous prendrez connaissance des spécifications MIPS. Vos programmes de test devront contenir suffisamment de types d'instructions pour bien couvrir l'émulateur.
2. Écrire le programme de traduction des instructions MIPS de leur forme textuelle vers leur forme hexadécimale. En mode fichiers, les résulats doivent être écrits dans le premier fichier de sortie. Il est recommandé de préparer en même temps le mode pas-à-pas, qui est presque identique, pour faciliter le déboguage. Le mode interactif est dispensable à ce stade (ie non obligatoire)..
3. Tester cette première version avec les tests fournis et ceux construits lors de l'étape 1, voire, en en ajoutant.
4. Répondez aux questions dans `README.md` et suivez les instructions de rendu sur Chamilo.

Votre application sera évaluée en la lançant en mode fichier automatique sur des programmes de correction.

Rendu #3: plan de l'émulateur (rendu indicatif : 3ème-4ème séance)

Planifiez la structure de votre émulateur en termes de *modules*. Un module est une brique du programme qui accomplit une tâche bien définie et indépendante des autres modules.

Un module prend la forme d'un fichier `.h` (avec des fonctions pour l'utiliser) et d'un fichier `.c` (avec le code).

Commes exemples de tâches indépendantes qui se prêtent bien à une séparation en modules, on trouve notamment :

- La lecture des instructions texte dans un fichier et leur assemblage ;
- La simulation des registres du processeur et de la mémoire (avec des fonctions de lecture et d'écriture) ;
- Le décodage des instructions assemblées en hexadécimal et leur effet sur les registres et la mémoire.

Ce ne sont pas les seuls, il y a d'autres options !

1. Planifiez votre émulateur et décrivez la structure choisie dans `README.md`. Esquissez quelles fonctions vous mettrez dans chaque module.
2. Suivez les instructions de rendu sur Chamilo.

Rendu #4: émulateur (5ème séance)

1. Réalisez le plan présenté au rendu précédent. Si vous pensez ne pas avoir le temps de réaliser toutes les instructions MIPS, sélectionnez un sous-ensemble qui vous permettra d'écrire des tests intéressants (calcul, conditions, boucles...).
2. Étendez votre suite de tests avec des nouveaux programmes et validez le comportement de votre émulateur.
3. (Optionnel) : modifiez votre assembleur pour supporter les étiquettes/labels dans le code assembleur. Ajoutez des nouveaux tests en conséquence.

4. Répondez aux questions dans `README.md` et suivez les instructions de rendu sur Chamilo.

Ressources

- Document sur le processeur MIPS, extrait du sujet du projet d'informatique de Phelma, 2ème année (François Portet, Nicolas Castagne, Mathieu Chabanas, Laurent Fesquet).
- Descriptif du jeu d'instruction MIPS.
- L'archive `CS351_2022_ProjetMIPS.zip` qui contient les fichiers initiaux du projet est sur Chamilo.

Enseignant·e·s

- Guillaume Besset (guillaume.besset.pro@gmail.com)
- Laure Gonnord (laure.gonnord@grenoble-inp.fr)
- Ioannis Parissis (Ioannis.Parissis@grenoble-inp.fr)
- Adrien Rochedy (adrien.rochedy@gmail.com)
- Sébastien Michelland (sebastien.michelland@lcis.grenoble-inp.fr)