CS351@ Esisar – MIPS Emulator Developer Documentation

Rougé Jean - Patard Gauthier

Structure du projet

— Compile_Module.[c|h] : permet la traduction d'instructions de l'asm vers machine language.
— Memory_Module.[c|h] : gestion de la liste chainée représentant la mémoire.
— Execution_Module.[c|h] : execution du programme, mode interractif.
— CPU_Struct.h : description de la structure du CPU, définission de constantes.
— main.c : lecture des arguments, gestion des différents fichiers, coréllation des autres modules.
— Librairy.h : contient des Librairie standard à inclure dans tous les modules.

Conseil d'utilisation

Par manque de temps nous n'avons pas pu être très flexible sur l'analyse des arguments. En résultat, la procédure d'appel du programme ce résumera à 4 cas:

— ./emul-mips
— ./emul-mips prog.s
— ./emul-mips prog.s -pas
— ./emul-mips prog.s sortie.hex reg.state

Pour les cas 2 et 3, les fichier de sortie seront dans le même répertoire où l'on à apellé le programme, ils auront le même nom que le fichier en entrée mais une extension différente.

`Mode Normal`

Dans ce mode, le programme commence par traduire les instructions et écrit le résultat dans le fichier .hex. Une fois terminé, il est affiché "Texte segment loaded" et le segment traduit est affiché sur la sortie standard. Ensuite le programme passe automatiquement à l'exécution du programme. Pour chaque instruction, il sera affiché: la valeur du programme compteur, le code hexadécimal de l'instruction ainsi que sa traduction en assembleur. Une fois terminé, le programme affiche l'état final des registres et de la mémoire, puis il écrit l'état des registre dans le fichier .state.

— Ce mode est le résultat des procédures d'appel 2 et 4.

.

`Mode Pas à Pas`

Dans ce mode, comme précédement, le programme commence par traduire les instructions, mais cette fois il affiche quelle ligne il est en train de process et la valeur en hexadecimal. Après chaque ligne l'entrée de l'utilisateur est attendue, il peut soit passer à la ligne suivante, soit sauter tout le process de compilation et commencer directement l'execution (les instructions seront quand même compilées, il ne s'en apercevra juste pas). Le segment traduit est affiché puis l'exécution commence. Chaque instruction exécutée est affichée une par une et à chaque fois, l'utilisateur à le choix entre: afficher

les registres, aficher la mémoire, ou continuer. Une fois la fin du programme atteinte, le programme effectue les mêmes actions qu'en mode normal.

— Ce mode est le résultat de la procédure d'appel 3.

.

`Mode Interractif`

Dans ce mode, le programme ne passe pas par d'étape de compilation, il affiche directement "Mode Interractif", et demande à l'utilisateur de rentrer une commande. L'utilisateur peut afficher les registres ou la mémoire, ou rentrer directement une instruction en assembleur. Le fonctionnement de ce mode est un peut particulier: A chaque fois qu'une instruction est rentrée par l'utilisateur, sa traduction est écrite en mémoire à l'adresse pointée par le programme compteur. Puis l'execution du programme est lancée à partir de cette même adresse. Le programme s'exécute donc jusqu'à ce que le PC pointe vers une adresse non initialisée (par défaut, toute les adresses contiennent la pseudo instruction EXIT).A partir de là le cycle recommence et l'utilisateur entre une nouvelle instruction qui sera ensuite encore mise en mémoire. Ce fonctionnement peut conventionnel à pour avantage de permettre à l'utilisateur d'implémenter des boucles. Si jamais lutilisateur implémentait une boucle infinie, le programme possède un failsafe qui limite le nombre d'instructions simulée à la suite. (Cette limite est définie dans Execution_Module.h)

— Ce mode est le résultat de la procédure d'appel 1.

Fonctionnement interne

Pour la compilation pour chaque ligne, on remplace les charactères génant, et on récupère le premier mot, on fait ensuite une recherche dichotomique dans le tableau des instructions pour trouver l'opcode et la categorie de l'instruction. Si la recherche abouti, on procède à traduire l'instruction, pour cela on fait un grand switch parmis les categories puis on traduit les arguments en fonction. Une fois traduite le module écrit le résultat dans le fichier .hex et écrit aussi le code à l'adresse mémoire correspondant.

Pour l'execution, le programme commence à l'adresse définit par PROGRAM_START, par défaut, c'est l'adresse 0x8000. A chaque ligne les différents champs de l'instruction sont décomposés et on fait un grand switch sur l'opcode de l'instruction effectuant l'opération correspondante.

Bugs connus/comportement non définis

— Petite défaillance dans la lecture de l'entrée standard en mode interractif: si on entre pas de charactere il faut appuyer deux fois sur entree.
— Comportement non défini lors d'une `SignalException`, on a choisi de ne rien faire.
— Pas de levée d'erreur lors d'une écriture dans le registre zero.