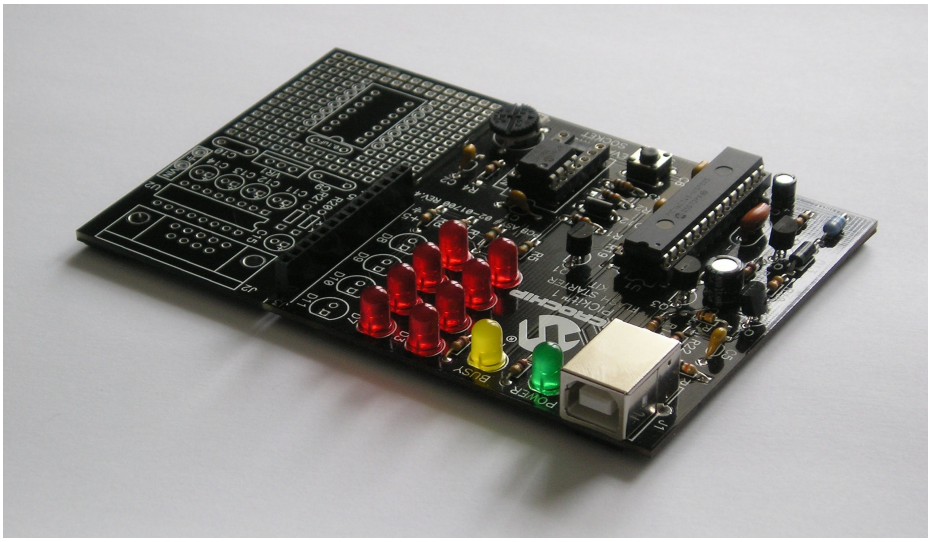
Maxime DICKERSON
Sophie FRANÇOISE
2A SICOM

18 novembre 2013

Projet d'informatique : simulateur MIPS

Livrable n° 3



Professeurs : Nicolas CASTAGNÉ et François PORTET

Introduction

Notre objectif pour ce livrable était de finir l'incrément 2 : être capable de désassembler un programme en assembleur, et également finir la fonction lm.

I – Le dictionnaire d'instructions

À cause de difficultés rencontrées lors de l'implémentation des fonctions nécessaires au désassemblage, nous avons dû faire des modifications dans la structure INSTRUCTION dans laquelle sont chargées les lignes de dico.txt.

Structure initiale :

```
typedef struct {
	char* nom;	//mnémonique de la fonction
	char* type;	//R I ou J
	int nbe_op;	//nombre d'opérandes
	char* ops[3];	//tableau de taille 3 contenant les opérandes
	unsigned int opcode;
	unsigned int func;
}INSTRUCTION;
```

Nouvelle structure :

```
typedef struct {
	char nom[25];
	char type[3];	//R I ou J
	int nbe_op;	//nombre d'opérandes
	char ops[4][26];	//tableau de taille 3 contenant les opérandes
	char opcode[78];
	char finc[9];
}INSTRUCTION;
```

Nous préférons utiliser des tableaux de caractères car c'est plus simple pour accéder à chaque caractère et cela reste simple de les afficher grâce au format %s.
Les tailles de tableaux sont volontairement définies trop grandes car parfois la fonction init_inst (qui charge en mémoire le dictionnaire d'instructions) affichait plus de caractères qu'elle ne devait et ce débordement se propageait d'instruction en instruction jusqu'à la fin du dictionnaire.
Le tableau des opérandes est un tableau à deux dimensions car c'est un tableau de mots.

II – Écriture dans la mémoire

Nous avons également dans ce livrable implémenté la fonction lm. Elle fonctionne sans problème notable.

III – Désassemblage des instructions

Notre programme est enfin capable de désassembler un programme chargé dans sa mémoire ! Notre commande da est capable de désassembler n'importe quelle suite d'instructions incluse dans la section .text. En effet s'il y a débordement dans la section suivante elle le signale et n'affiche rien. De plus si l'adresse de départ n'est pas alignée on n'affiche rien non plus.

Pour désassembler les instructions on commence par récupérer l'adresse sous la forme d'une chaîne de caractères. Pour traiter l'instruction, nous avons ensuite choisi de stocker sa conversion en binaire dans un tableau de 32 caractères, chacun représentant un bit de l'instruction. Nous n'avons donc pas eu besoin de convertir le big endian en little endian, grâce au procédé de conversion lui-même. De plus, un tableau de bits nous semblait plus facile à manipuler, puisque le bit n°4 de instr_bin par exemple est désigné par instr_bin[4]. Il est donc facile d'en contrôler la valeur.

Conclusion

Le plus grand problème que nous ayons rencontré est lié à la représentation des char* et leur manipulation, ainsi que l'ambiguïté entre les formats %c et %d. C'est ce qui nous a conduits à changer la structure contenant les instructions.

Quant à la compréhension, ce qui nous semble le plus difficile est la simulation de l'exécution des instructions. Nous n'avons pas encore commencé cette étape.

Le jeu de tests reste à compléter (nous avons cependant vérifié que les tests faits précédemment passent toujours après les modifications apportées au programme) mais pour le reste nous avons terminé ce qui était demandé pour le livrable 2, et la boucle de la commande run est également terminée.