TP n°1

Informatique embarquée

Vincent Ruello

Partie 1 à rendre avant mercredi 18/10/2023 - 16:00 (CEST)

Partie 1. Codage d'un jeu d'instructions

Dans cette partie, on s'intéresse au codage et au décodage d'un jeu d'instructions.

Exercice 1

Proposer un codage de jeu d'instruction pour une machine qui implémente les instructions suivantes :

- OP r1, r2, r3 avec OP = add, sub, mul, div, and, or, xor et qui fait r1 = r2 OP r3
- OP r1, r2, cst avec OP = addi, subi, muli, divi, andi, ori, xori avec cst sur 12 bits
- lw r1, cst(r2) qui fait r1 = valeur stockée à l'adresse r2 + cst, avec cst sur 12 bits
- sw r1, cst(r2), qui stocke la valeur stockée dans r1 à l'adresse r2 + cst, avec cst sur 12 bits
- OP r1, r2, label avec OP = beq, bne, blo, bgt et label sur 12 bits
- Il y a 64 registres de 32 bits. Le registre $0 est toujours égal à 0.

Pour simplifier la suite, on fera en sorte que le codage de chaque instruction soit de taille identique. Faire valider le codage par l'enseignant.

Exercice 2

Que fait le programme suivant ?

```
      lw $1, ($0)
      addi $2, $0, 42
      addi $3, $0, 1
      beq $1, $2, fin
      xor $3, $3, $3
fin:
      sw $3,($0)
```

Écrire dans le langage de votre choix un programme qui prend en entrée un code assembleur et écrit dans un fichier sa traduction avec le codage déterminé à la question précédente.

Exercice 3

Implémenter dans le langage de votre choix un émulateur pour votre jeu d'instructions. Dans ce cas, il s'agit d'un programme qui prend en entrée un programme préalablement traduit avec votre codage (c'est à dire le fichier généré lors de l'exercice précédent) et l'exécute.

La mémoire et les registres peuvent être représentés par des tableaux.

Partie 2. Programmation assembleur (facultative)

Cette partie est basée sur un simulateur sous forme d'application web libre disponible à l'adresse suivante : https://schweigi.github.io/assembler-simulator/.

L'architecture externe est présentée sur le lien suivant : https://schweigi.github.io/assembler-simulator/instruction-set.html

Le jeu d'instruction implémenté est léger (une trentaine d'instructions) et le modèle machine est très simplifié : processeur 8 bits avec 4 registres de travail (A, B, C, D) et 256 octets de mémoire. Ce simulateur ne sépare pas la mémoire programme et la mémoire données.

Exercice 1

Traduire les schémas de programmation suivant avec le jeu d'instructions proposé par le simulateur :

- Condition

```
if (A == B) // registres A et B
{
      // Ecrire '1' dans 0xE8
}
else
{
      // Ecrire '0' dans 0xE8
}
```

- Opération arithmétique « complexe »

```
A = 5 ;
B = 3 ;
C = 10 ;
D = ((A*B) / (C-A)) * 2 ;
// Ecrire la valeur de D dans 0xE8
```

- Boucle

```
A = 0; // registre A
while (A < 10)
{
      A = A + 1;
}
// Ecrire la valeur de A dans 0xE8
```

- Parcours de tableau

```
char* buffer = 0xE8;
for (i = 0 ; i < 24; i++) {
      buffer[i] = 65 + i ;
}
```

Exercice 2

Ecrire une fonction permettant d'afficher la valeur du registre C encodée en binaire sur la console de sortie.

Exercice 3

Ecrire un programme calculant l'élément de rang 13 (F_{13}) de la suite de Fibonacci. Pour rappel, la suite de Fibonacci est définie par :

$$F_0 = 0$$
$$F_1 = 1$$
$$F_{n+2} = F_n + F_{n+1} (n \geq 0)$$

Que se passe-t-il lorsqu'on calcule F_{14} ?

Exercice 4

Ecrire un programme qui teste la primalité d'un nombre. On pourra utiliser un algorithme naïf (i.e. tester tous les diviseurs entre 1 et le nombre). Si le nombre est premier, le texte « PREMIER » doit être affiché sur la console. Sinon, le texte « NON PREMIER » sera affiché.