DURHONE Coralie
EYMOND LARITAZ Cyprien
MOULAYAT Yacine
VILLALTA Adrien

Projet logiciel

Simulateur ARM

1. Structure du code

1.1. arm_data_processing

Le décodage se fait en deux étapes, la première consiste à récupérer les champs fixes pour tout type d'instruction, à savoir l'*opcode*, *rn*, *rd*, *shift*, *rm* et *S*. Ensuite nous faisons un traitement différent pour déterminer le *shifter_operand* selon la valeur du bit 4 de l'instruction. Dans chacun des deux cas, nous calculons le *shifter_operand* grâce aux variables précédemment initialisées et nous retournons à un traitement commun où nous appelons la fonction d'instruction correspondant à l'opcode, puis selon le bit *S*, nous mettons à jour les flags contenus dans *cpsr* puis écrivons le résultat de l'opération dans le registre *rd*.

Les instructions sont déclarées et réalisées avec la signature *uint32_t instr(arm_core, uint32_t rn, uint32_t shifter_operand, int S)*. Elles sont stockées dans un tableau de fonctions dans l'ordre des opcodes de l'instruction, de sorte à ce que *instructions[x]* soit l'adresse de la fonction qui reproduit le comportement de l'instruction d'opcode égal à *x* (soit pour l'instruction *ADC*, d'opcode *0b0101*, elle sera à l'index 5 du tableau). Cette structure nous permet de gagner du temps et des lignes dans le programme, car on peut simplement récupérer l'opcode de l'instruction et d'appeler la fonction dont l'index est l'opcode dans le tableau de fonctions.

1.2. arm_branch_other

1.3. arm_load_store

1.4. arm_instruction

2. Fonctionnalités

3. Bugs connus non résolus

4. Tests effectués

5. Progression et répartition du travail