Compte rendu Projet Logiciel
Réalisation d'un simulateur ARM

Groupe 6

Membres du groupe:

BRUN-COSME-GAZOT Guillaume
GHERBI Amayas
AL JOUAID Adam
BARKOK Omar
DAHMANE Ryma
DESFONDS Antoine

Mode d'emploi:

Cloner le repo:

- git clone https://github.com/Cereal38/arm-simulator.git

Changer de dossier:

- cd arm-simulator/src/

Configurer et compiler:

- ./configure CFLAGS='-Wall -Werror -g'
- make

Lancer le simulateur:

- ./arm-simulator *(terminal 1)*
- arm-none-eabi-gdb *(terminal 2)*
    - file Examples/example1
    - target remote localhost:<port gdb>
    - set endian big
    - load

Pour lancer les tests unitaires, il suffit de les appeler comme dans l'exemple suivant:

- ./test_arm_data_processing

Pour lancer les “GitHub Actions workflows” en local, il est recommandé d’utiliser l’utilitaire act (https://github.com/nektos/act)

Descriptif de la structure du code:

Nous avons utilisé la structure donnée pour le projet. Peux de modifications y ont été ajoutés, si ce n’est la création de fichiers de tests qui sont:

- test_arm_data_processing.c
- test_arm_load_store.c
- test_arm_branch.c

Un fichier ***.github/workflows/run_tests.yml*** a également été ajouté (son intérêt est décrit plus tard dans la partie correspondante aux tests).

Nous avons également créé un bon nombre de fonctions et de constantes qui n’étaient pas suggérés dans le squelette de code fourni pour améliorer la lisibilité du code que nous écrivons.

Liste des fonctionnalités implémentées et manquantes:

La gestion des registres et de la mémoire a été codée à l’aide des squelettes de fonction fournis. Certaines fonctions ont cependant été ajoutées à *registers.c* pour simplifier la lecture et l’écriture de certains bits du registre cpsr.

Toutes les instructions de data processing demandées dans le sujet ont été implémentées et testées.
Les bits I et S ainsi que les 12 bits de la shifter operand ont été implémentés comme la doc le suggérait.
Les instructions de branchement ont été implémentées et testées.
Les instructions MRS et MSR implémentées.
Les différentes instructions load store

Nous n’avons pas implémenté la partie relative à la gestion des exceptions.

Liste des éventuels bogues connus mais non résolus:

Load store offset n’a pas pu être résolut à temps
MSR peu testée car spsr non disponible en mode USER (manque de temps)

Liste et description des tests effectués:

Nous avons choisi une approche principalement à base de tests unitaires pour tester notre code. Ce choix à été fait pour nous permettre de localiser plus simplement les bugs. Nous avons ainsi un fichier pour tester chaque grande catégorie d’instructions (data processing, branchements, load store).

Grâce au fichier *.github/workflows/run_tests.yml*, GitHub nous assure que le code qu'on souhaite merge sur la branche *master* compile et que tout les tests unitaires passent. Cela nous évite d'avoir à lancer les tests manuellement ou de se retrouver avec du code bugué sur la branche principale du projet en cas d'erreur de la part de l'un des membres.

Liste des tests relatifs aux registres:

```
cereal@cereal:~/Git/arm-simulator/src$ ./registers_test
Test : Create registers ... OK
Test : Write/Read cpsr ... OK
Test : Get mode (USR) ... OK
Test : Get mode (FIQ) ... OK
Test : Get mode (IRQ) ... OK
Test : Get mode (SVC) ... OK
Test : Get mode (ABT) ... OK
Test : Get mode (UND) ... OK
Test : Get mode (SYS) ... OK
Test : Current mode has spsr (USR) ... OK
Test : Current mode has spsr (FIQ) ... OK
Test : Current mode has spsr (IRQ) ... OK
Test : Current mode has spsr (SVC) ... OK
Test : Current mode has spsr (ABT) ... OK
Test : Current mode has spsr (UND) ... OK
Test : Current mode has spsr (SYS) ... OK
Test : Write/Read spsr (FIQ) ... OK
Test : Write/Read spsr (IRQ) ... OK
Test : Write/Read spsr (SVC) ... OK
Test : Write/Read spsr (ABT) ... OK
Test : Write/Read spsr (UND) ... OK
Test : In a privileged mode (USR) ... OK
Test : In a privileged mode (FIQ) ... OK
Test : In a privileged mode (IRQ) ... OK
Test : In a privileged mode (SVC) ... OK
Test : In a privileged mode (ABT) ... OK
Test : In a privileged mode (UND) ... OK
Test : In a privileged mode (SYS) ... OK
Test : Write/Read registers (USR) ... OK
Test : Write/Read registers (FIQ) ... OK
Test : Write/Read registers (IRQ) ... OK
Test : Write/Read registers (SVC) ... OK
Test : Write/Read registers (ABT) ... OK
Test : Write/Read registers (UND) ... OK
Test : Write/Read registers (SYS) ... OK
```

Liste des tests relatifs aux instructions de data processing:

```
cereal@cereal:~/Git/arm-simulator/src$ ./test_arm_data_processing
Test : ADD (Immediate value) ... OK
Test : ADD (Second value from register) ... OK
Test : ADD (S = 0) ... OK
Test : ADD (N+V : 0x7FFFFFFF + 0x1) ... OK
Test : ADD (Immediate value with rotation) ... OK
Test : ADD (Result is 0) ... OK
Test : ADD (Immediate shift LSL#4) ... OK
Test : ADD (Immediate shift LSR#4) ... OK
Test : ADD (Immediate shift ASR#4) ... OK
Test : ADD (Immediate shift ROR#4) ... OK
Test : ADD (Immediate shift RRX) ... OK
Test : ADD (Register shift LSL [0]) ... OK
Test : ADD (Register shift LSL [3]) ... OK
Test : ADD (Register shift LSL [32]) ... OK
Test : ADD (Register shift LSL [40]) ... OK
Test : ADD (Register shift LS3 [3]) ... OK
Test : ADD (Register shift ASR [4]) ... OK
Test : ADD (Register shift ROR [4]) ... OK
Test : SUB (Positive result) ... OK
Test : SUB (Negative result) ... OK
Test : SUB (Overflow) ... OK
Test : SUB (Result is 0) ... OK
Test : AND (Immediate value) ... OK
Test : AND (Second value from register) ... OK
Test : AND (Result is 0) ... OK
Test : AND (Immediate shift with rotate_imm != 0) ... OK
Test : EOR (Immediate value) ... OK
Test : EOR (Second value from register) ... OK
Test : RSB (Immediate value) ... OK
Test : RSB (No borrow) ... OK
Test : ADC (Immediate value) ... OK
Test : ADC (No carry) ... OK
Test : ADC (Overflow) ... OK
Test : SBC (Immediate value) ... OK
Test : SBC (With borrow) ... OK
Test : SBC (Overflow) ... OK
Test : RSC (Immediate value) ... OK
Test : RSC (No borrow) ... OK
Test : TST (Immediate value) ... OK
Test : TST (Result is 0) ... OK
Test : TST (S = 0) ... OK
Test : TEQ (Immediate value) ... OK
Test : TEQ (Result is 0) ... OK
Test : CMP (Positive result) ... OK
Test : CMP (Negative result) ... OK
Test : CMP (Overflow) ... OK
Test : CMP (Minus 0) ... OK
Test : CMN (Positive result) ... OK
Test : CMN (ADD 0) ... OK
Test : ORR (Immediate value) ... OK
Test : ORR (Result is 0) ... OK
Test : MOV (Immediate value) ... OK
Test : MOV (Value from register) ... OK
Test : BIC (Immediate value) ... OK
Test : MVN (Immediate value) ... OK
```

Liste des tests relatifs aux Branchements:

Lancer ./test_arm_branch

```
Test :  B Inconditionnel (Adresse Positive)  ... OK
Test :  B Inconditionnel (Adresse Negative)  ... OK
Test :  B Égal (EQ) - Adresse Positive  ... OK
Test :  B Non Égal (NE) - Adresse Positive  ... OK
Test :  B Supérieur ou Égal (CSHS) - Adresse Positive  ... OK
Test :  B Inférieur (CCLO) - Adresse Négative  ... OK
Test :  B Négatif (MI) - Adresse Négative  ... OK
Test :  B Positif ou Nul (PL) - Adresse Négative  ... OK
Test : BL Inconditionnel (Adresse Positive)  ... OK
Test : BL Inconditionnel (Adresse Negative)  ... OK
Test : BL Égal (EQ) - Adresse Positive  ... OK
Test : BL Non Égal (NE) - Adresse Positive  ... OK
Test : BL Égal (EQ) - Adresse Négative  ... OK
Test : BL Non Égal (NE) - Adresse Négative  ... OK
Test : BL Supérieur ou Égal (CSHS) - Adresse Négative  ... OK
```

La méthode de test utilisée ci-dessus offre une couverture de base des différentes conditions de branchement mais n'est malheureusement pas exhaustive.
Tout au long de la création des instructions, différents tests sur scripts ont été effectués pour s'assurer du bon fonctionnement de ces dernières.
Notamment:

- Les Boucles : les branchements assurent la boucle à chaque rencontre d'instruction et assure bien la sortie de boucle selon les conditions
- Les branchements imbriqués : utile pour les if-else par exemple, pas de conflit rencontré
- Sauts multiples : plusieurs instructions successives bien gérées

Les tests de limites ne peuvent se faire que de manière théorique ici.

Liste des tests relatifs aux load store:

Lancer ./test_arm_load_store

Seulement deux tests ont été réalisés. Le premier test consiste à faire un STM sur un seul registre. Le deuxième test ne fonctionne pas et consiste à faire un STM sur deux registres. Le bug à été détecté mais n'a pas pu être corrigé.

Journal décrivant la progression du travail et la répartition des tâches au sein du groupe:

Guillaume:

- Gestion du repo Git (setup, workflows, issues)
- Implémentation et test des instructions de data processing
- Implémentation et test des registres

Ryma:

- Implémentation fonction vérification des conditions
- Implémentation première version de arm_instructions
- Implémentation et test des branchement
- Implémentation instruction MRS
- Implémentation instruction MSR (immediate / register operand)

Amayas:

- Correction et finalisation de arm_instructions
- Automatisation du lanceur de tests
- Assistance sur différentes fonctions (load , store , branchements)

Adam:

- Implémentation de la fonction arm_load_store
- Assistance fonction arm_load_store_multiple

Antoine:

- Implémentation instruction LDM
- Implémentation instruction STM
- Assistance instructions LDR STR
- Tests arm_load_store.c

Omar:

-

Travail commun:

- Implementation memory.c
- Débogage des fonctions divers