Projet PROG5 2019-2020

Simulateur ARM

Rapport de projet

Groupe 3:

TARDY Jimmy
MARQUES Telmo
AVOGNON Laëtitia
GUTFREUND Delphine
LEISERING Dimitri
PURANIK Nishith

1 - Organisation du travail **3**
A - Environnement de travail 3
B - Organisation des tâches 3
C - Journal de bord 4

2 - Structure du code **6**
A - Organisation des fichiers 6
B - Fonctionnalités ajoutées 6
C - Tests 7
D - Bugs 7

3 - Procédure d'installation **8**

1 - Organisation du travail

A - Environnement de travail

En premier lieu, nous avons dû prendre des dispositions face à la crise sanitaire exceptionnelle. Nous avons convenu que nous communiquerions sur discord afin de partager. Enfin il était convenu d'installer une machine linux (VM ou WSL2) sur chaque pc durant les vacances pour pouvoir démarrer le projet dès le jour de la rentrée. Pour ce qui est du code, nous avons utilisé Git et Github ainsi que leurs système de branches pour pouvoir nous séparer les tâches sans influencer le travail des autres.

B - Organisation des tâches

Au début, on a tous commencé ensemble pour compléter les fichiers memory.c et registers.c. On a ensuite formé des duos pour le reste du projet pour combler les faiblesses de chacun et que personne ne reste bloqué :

- Duo 1 : TARDY Jimmy / MARQUES Telmo
- Duo 2 : AVOGNON Laëtitia / GUTFREUND Delphine
- Duo 3 : LEISERING Dimitri / PURANIK Nishith

On a décidé de répartir le travaille tel que :

Le duo 1 travaille sur arm_instruction.c et arm_load_store.c, le duo 2 sur arm_data_processing.c et decode.c et le duo 3 sur arm_branch_other.c. Puis Jimmy a géré les registres en fonction des différents modes que le processeur peut avoir et telmo sur MRS. À la fin on a géré les conflits de merge sur github pour pouvoir tout mettre en commun. Enfin, les derniers jours Jimmy et Telmo ont réalisé le rapport, Laëtitia et Delphine ont factorisé leur code et Nishith et Dimitri ont réalisé des jeux de tests complémentaires.

C - Journal de bord

Jour:	Tâches effectués :
1	• Prise en main du sujet et mise en place de l'environnement de travail (github, discord) • Complétion minimale et fonctionnelle des fichiers memory.c et registers.c pour pouvoir commencer la suite
2	• Distribution des instructions complétées dans arm_instruction.c • Répartition des instructions à prendre en charge par chaque duo • Compréhension et réflexion sur le travail à réaliser sur le data processing, début d'implémentation de quelques opérations. • Début d'implémentation des instructions de branchement et ceux avec sauvegarde d'adresse de retour.
3	• Réalisation de load store de Byte ou de Word • Début de load store multiple • Finalisation de la prise en charge des opérations précédentes avec et sans shift (LSL, LSR, ASR, ROR) • Ajout des opérations MOV et MVN avec et sans shift.
4	• Factorisation de load store de Byte ou de Word et réalisation de load store multiple • Première mise en commun des différentes tâches • Factorisation du code par la création d'un fichier de fonctions communes aux instructions à prendre en charge (decode.h/c)
5	• Amélioration des fonctions de decode.c/h et ajout de la fonction condition qui vérifie la condition de l'instruction avec les flags ZNVC • Ajout de la fonction de calcul de résultat des opérations dans le fichier arm_data_processing.c et début de la fonction de mise à jour des flags ZNVC selon l'opération réalisée
6	• Fin de la fonction maj_flag pour la mise à jour des flags ZNVC selon l'opération réalisée • Factorisation de load store multiple
7	• Ajout des fonctions CarryFrom, BorrowFrom, OverflowFrom • Tests de compilation et correction du code

8	• Implémentation de l'instruction MRS • Tests et débogages
9	• Création des jeux de tests pour la soutenance • Mise en commun des fichiers des différents duos, correction des conflits de merge sur github • Écriture du rapport
10	• Derniers tests et débogages • Écriture du rapport • Création de jeux de tests complémentaires pour la soutenance

2 - Structure du code

A - Organisation des fichiers

Nous nous sommes inspirés du code fourni. Nous avons rajouté le fichier decode(.h/.c) pour nos fonctions usuelles qui reviennent plusieurs fois. Le fichier arm_constant a été complété pour améliorer la lisibilité du code.

B - Fonctionnalités ajoutées

Fichiers (. c et .h)	Avancement
memory	lecture / écriture des byte, half, word
register	Gère les différents modes
arm_data_processing	• AND / EOR / SUB / RSB / ADD / ADC / SBC / RSC / TST / TEQ / CMP / CMN / ORR / MOV / BIC / MVN (avec registre, shift ou valeur immédiate) et options {cond} et {S} (ex: ANDS, ADDLT, etc.) • Shift : LSL / LSR / ASR / ROR (avec registre ou valeur immédiate)
arm_load_store	• LDR / STR {S / B / D / H / SH} • LDM {1, 2, 3} / STM {1, 2}
arm_branch_others	• Branchement B, BL • MRS
arm_instruction	• Exception UNDEFINED_INSTRUCTION • Traitement des conditions pour appliquer l'exécution
arm_constant	• Constantes registres et NO_EXCEPTION
arm_exception	Non réalisé

memory: accès mémoire

register: registres du simulateur

arm_data_processing: décodage spécialisé pour les instructions de data processing

arm_load_store: décodage spécialisé pour les instructions d'accès à la mémoire

arm_branch_others: décodage spécialisé pour les instructions de branchement et les instructions n'appartenant pas aux catégories précédentes

arm_instruction: redirection des instructions vers les fichiers correspondants

arm_constants: définition des différents modes d'exception, types d'exceptions et registres divers

arm_exception: gestion des exceptions et des interruptions

C - Tests

Afin de tester les instructions dans leurs totalités, nous avons réalisé des fichiers de tests unique par instruction:

- testLDR_STR.s : effectue les tests de LDR et STR
- testLDRB_STRB.s: effectue les tests de LDRBet STRB
- testLDRH_STRH.s: effectue les tests de LDRH et STRH
- testLDR_SH_SB.s: effectue les tests de LDRSH et STRSB
- test_Data_proc.s: effectue les tests des instructions de data-processing

D - Bugs

LDRSH et LDRSB ne sont pas fonctionnels mais implémentés. La lecture d'un halfword et byte signed dans la mémoire n'a pas été effectuée. Sinon on suppose que ces 2 fonctions sont correctes.

3 - Procédure d’installation

Installation du projet:

Placez-vous dans le repertoire qui contiendra le simulateur.
git clone https://github.com/TarJimmy/Projet-S5-PROG.git
cd Projet-S5-PROG/

si vous êtes sur une VM: chmod u+x configure

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

Vous aurez ensuite besoin de package linux additionnel:

gdb-multiarch: apt install gdb-multiarch
un compilateur: apt install gcc-arm-none-gdnuabi
crée automatiquement buildables: apt install autoreconf

Lancer ensuite: autoreconf -vif

Ouvrir 2 terminaux.

Dans le 1er terminal:

Le programme principal, arm_simulator, doit utiliser une session, donc dans un 1er terminal lancer :
./arm_simulator
Cela permet l'ouverture de 2 connexions: le port gdb client et le port pour irq (inutile pour nous)

Dans le 2ème terminal: gdb-multiarch

- file path/fichier_executable, pour charger le fichier que vous voulez tester
- target remote localhost:"port de arm_simulator"
- load

Ensuite vous pouvez utiliser les commandes classiques de gdb pour lancer le programme:

- step,
- break,
- cont ...