# Simulateur de processeur ARM

## 1 Organisation

### 1.1 Equipe de travail

Le groupe de projet numéro 9 est composé de :

- Rose CHAPELLE
- Rémy DEL-GROSSO
- Matteo DOYEN
- Shane MOISSONNIER
- Ianis OLIVIER-PACAUD
- Benoit PEGAZ-NOIN

De par la nature exceptionnelle des évènements actuels, nous avons dû travailler en distanciel. Le code est partagé à l'aide des outils Git et Github, une décision qui aurait aussi été prise si nous avions été en présentiel.

### 1.2 Organisation des tâches

Les tâches ont été réparties par fichier :

- arm_instruction & registers & memory : Tous
- arm_data_processing : Matteo DOYEN & Shane MOISSONNIER
- arm_load_store & load_store_test: Ianis OLIVIER-PACAUD & Rose CHAPELLE
- arm_branch_other & registers_test : Rémy DEL-GROSSO & Benoit PEGAZ-NOIN
- arm_exception & arm_utils : Shane MOISSONIER

## 2 Structure de code

### 2.1 Les fichiers

Le squelette fourni a été conservé. Deux fichiers concernant arm_utils ont été ajoutés. Ces fichiers regroupent des fonctions communes utilisées dans la plupart des instructions.

## 2.2 Les fonctionnalités par fichier

| Fichier | Fonctionnalités |
|---|---|
| arm_instruction | Fetch les instructions et les décodent en appelant les fonctions correspondantes. |
| registers | Lecture/Ecriture de registres |
| memory | Lecture/Ecriture (octets, demi-mots, mots) |
| arm_data_processing | Instructions d'opérations sur les registres (additions, soustractions, test d'équivalence) |
| arm_load_store | Ecriture dans les registres/ la mémoire (mots, demi-mots et octets) |
| arm_branch_other | Instructions B/BL, Instructions MRS/MSR |
| arm_exception | Traitements des exceptions |
| arm_utils | Liste des fonctions communes (exemple: test du champ condition dans les instructions) |
| util | Fonctions de manipulation bit à bit |

Table 1: Tableau des fonctionnalités par fichier

## 2.3 Les tests

Une automatisation des tests a été mise en place pour memory, registers et arm_load_store. Les autres tests sont effectués manuellement à l'aide de fichiers de tests écrits pour les différentes instructions. Il manque le test des exceptions afin de confirmer qu'elles fonctionnent correctement.

| Fonctionnalité | Type de test, fonctionnement |
|---|---|
| Load | Une adresse est écrite dans le registre RN ainsi qu'une valeur à cette adresse (+/- décalage). La fonction load_store est appelée avec l'instruction à tester. Si RD est égal à la valeur stockée à l'adresse, alors le test est un succès. |
| Store | Une valeur est écrite dans le registre RD, ainsi qu'une adresse dans le registre RN. La fonction load_store est appelée avec l'instruction à tester. Si la valeur dans RD a bien été écrite à l'adresse de RN +/- décalage, alors le test est un succès. |
| Data_processing | Nous avons écrit toutes les différentes instructions reconnues par data processing dans un fichier ".s" et avons mis en commentaire les valeurs attendues dans les registres et les flags. Pour tester que les instructions fonctionnent comme prévu nous chargeons le fichier dans gdb et avançons pas à pas avec des "stepi". Les tests sont validés. |
| Registers | Les fonctions de registers sont testées dans registers_test.c avec des tests automatisés. Les fonctions d'écriture et lecture sont utilisées. Si les valeurs sont écrites correctement dans les bons registres et si les lectures récupèrent les valeurs attendues, alors les tests sont un succès. |
| Memory | Les tests déjà fournis dans memory_test.c sont validés. |
| Branch | Dans example1.s, un branchement a été testé. Les fonctions de débug ont permis de vérifier que l'instruction indiquée par le branchement est bien exécutée lorsque la condition du branchement est respectée. Les tests manuels fonctionnent mais, il n'a pas encore été possible d'automatiser les tests. |

Table 2: Tableau des tests par fonctionnalité

Les tests ont révélé certains bogues encore non résolus.

| Type de bogue (erreur/fonctionnement non attendue) et fichier/fonction concerné(e) | Explication et code |
|---|---|
| Fonctionnement non attendu d'une instruction dans les fichiers "example3" et "example4" | code :<br>ldr r0, = limite<br>...<br>limite :<br>.word 0x12345678<br><br>Explication :<br>Il semblerait que cette instruction soit censée mettre le mot "0x12345678" dans le registre r0. A l'exécution cette instruction ne lève aucune erreur mais la valeur du mot ne se retrouve pas dans le registre r0. |

Table 3: Liste des bogues non résolus

# 3 Journal

| Date | Tâches effectuées |
|---|---|
| Jeudi 17/12 | Nous avons pris en main GitHub et initialisé le répertoire Git du projet. |
| Vendredi 18/12 | Nous avons choisi comment stocker la mémoire du processeur simulé et implémenté une interface permettant d'y accéder en complétant le fichier memory.c. |
| Mercredi 23/12 | Nous avons créé une branche "arm_instruction" dans laquelle nous traitons la reconnaissance des différents types d'instructions et appelons les fonctions liées à l'instruction étudiée.<br>Une première version de reconnaissance d'instructions a été réalisée. |
| Lundi 04/01 | arm_load_store :<br>- Compréhension des exigences à réaliser et à exécuter<br>- Développement de Load en immediate offset<br><br>arm_data_processing :<br>- Compréhension des fonctions à réaliser, recherche de documentation et lecture du manuel<br><br>arm_branch_other :<br>- Compréhension des fonctions à réaliser et lecture du manuel |
| Mardi 05/01 | arm_load_store :<br>- Structure du Load et Store finie pour les mots et les octets<br>- Recherche de tests corrects à effectuer<br><br>arm_data_processing :<br>- Correction des valeurs de retour de memory.c<br>- Début de décodage des instructions pour data processing<br><br>arm_branch_other :<br>- Développement de la fonction arm_branch (version non définitive) |
| Mercredi 06/01 | arm_load_store :<br>- Recherche sur le load store multiple, quelques difficultés de compréhensions<br><br>arm_branch_other :<br>- Amélioration de la fonction arm_branch en prenant en compte les flags NZVC |

| | |
|---|---|
| Jeudi 07/01 | Nous nous sommes retrouvés en début d'après-midi pour faire un point sur le code et factoriser des fonctions communes.<br><br>arm_load_store :<br>- Nous nous sommes rendus compte que nous avions quelques erreurs de compréhension au niveau des valeur immédiates et nous avons corrigé le code en fonction<br>- Factorisation de notre fonction (beaucoup de répétitions évitées dans le code)<br><br>arm_data_processing :<br>- Correction et amélioration de décodage des instructions data_processing<br><br>arm_branch_other :<br>- Correction des instructions B et BL<br>- Implémentation de l'instruction MRS |
| Vendredi 08/01 | arm_load_store :<br>- Développement de Load/store Multiple et extra load store (pour les demi-mots)<br><br>arm_data_processing :<br>- Création d'un fichier arm_utils contenant les fonctions communes aux trois types d'instructions tel que la fonctionnalité "arm_check_condition" qui est utilisé par toutes les instructions<br>- Ajout de nouvelles constantes dans le fichier arm_constant et étude de la factorisation de notre code<br>- Ajout d'une fonction de mise à jour des flags NZCV du registre CPSR<br><br>arm_branch_other :<br>- Merge de la branche arm_branch_other avec la branche main<br>- Compilation fonctionnelle après le merge des deux branches |
| Samedi 09/01 | arm_data_processing :<br>- Avancée sur le décodage des instructions arm_data_processing<br>- Prise en compte de la possibilité que le second opérande soit une valeur immédiate |
| Dimanche 10/01 | arm_data_processing :<br>- Création d'une fonction commune aux deux fonctions des instructions de data_processing, selon que le second opérande soit une valeur immédiate ou non, une valeur différente est envoyée dans la fonction<br>- Création de fichiers de tests .s dans le dossier Examples pour les différentes instructions du data_processing<br>- Ajout d'appel à la fonction debug pour suivre le fonctionnement de nos fonctions et comprendre les bugs possibles<br>- Erreur quand une valeur de registre est négative et présence d'une boucle infinie sur certaines instructions |
| Lundi 11/01 | Nous avons fait une réunion pour faire un point sur l'avancée du projet et voir ce qui pouvait être mis en commun.<br>Toutes les branches vont être regroupées sur la branche principale.<br><br>arm_load_store :<br>- Prise en compte de particularités pour les fonctions concernant le registre R15<br>- Test des fonctions avec debug pour vérifier le bon cheminement des commandes<br><br>arm_branch_other :<br>- Factorisation du code pour vérifier la validité des conditions<br><br>arm_data_processing :<br>- Correction de l'erreur sur les valeurs négatives de registres.<br>- Correction du problème de boucle infinie dûe à un problème de valeur de retour des fonctions appelées dans le fichier arm_instructions.c<br>- Première version de la gestion des exceptions.<br>- Merge de la branche arm_processing avec la branch main<br>- Tests de fichiers .s contenant des instructions de branchement et de data_processing |

| | |
|---|---|
| Mardi 12/01 | arm_load_store :<br>- Etude du fonctionnement des tests et création du fichier load_store_test.c<br><br>arm_branch_other :<br>- Développement de l'instruction MSR<br><br>arm_exception :<br>- Test et débogage des exceptions arm<br>- Impossibilité de faire un software interrupt lors d'une interruption dans un continue dans gdb<br><br>Toute branche :<br>- Modification de register.c, désormais il n'y a plus besoin de connaître le mode courant pour lire et écrire dans les registres user |
| Mercredi 13/01 | arm_load_store :<br>- Implémentation les tests pour nos fonctions et correction des erreurs au niveau des load/store<br><br>registers_test :<br>- Ecriture de tests automatisés pour vérifier les fonctionnalités des registres<br><br>test_data_processing.s :<br>- Ecriture de tests pour toutes les opérations implémentées de data processing |
| Jeudi 14/01 | Rendu de projet |

Table 4: Suivi journalier des tâches