Manal BENAISSA – Antonin CASTEL – Elisa GUENEAU – Ma HANYUAN

### Description des fonctions

Voici la liste des instructions utilisables :

Instructions de traitement de données

AND, EOR, SUB, RSB, ADD, ADC, SBC, RSC, TST, TEQ, CMP, CMN, ORR, MOV, BIC, MVN

Il est possible d’ajouter l’option S sur toutes les fonctions :

ADDS R0, R0, R2, #5

…Ou même d’effectuer des décalages internes avec LSL, LSR, ASR et ROR :

ADD r2, r1, r0, LSR #2

###### COde : ARM\_DATA\_PROCESSING.C

arm\_data\_processing\_shift(arm\_core p, uint32\_t ins) : fonction principale. Elle regarde la forme de l’instruction (data processing ? load/store ? mrs ?)

traitement(arm\_core p, uint32\_t ins) : fonction principale du data-processing (ADD, SUB etc…).

set\_ZNCV\_CPSR(arm\_core p, uint32\_t Zbit, uint32\_t Nbit, uint32\_t Cbit, uint32\_t Vbit) :  
Permet de modifier les Flags du CPSR par les flags donnés en paramètre. ON EST PAS OBLIGE DE CHANGER TOUS LES FLAGS ! Mettre une valeur au hasard (9 par exemple.) pour ne pas changer le flag.

Exemple : set\_ZNCV\_CPSR(p, 0, 1, 9, 9) permet d'ignorer C et V.  
  
  
int carry\_from(uint32\_t op1, uint32\_t op2) : Prend 2 opérandes en paramètre et retourne 1 si il y a une retenue sortante (utile pour le C flag), 0 sinon.  
  
borrow\_from(uint32\_t op1, uint32\_t op2) : Utile pour le C flag.  
  
overflow\_from(uint32\_t op1, uint32\_t op2, uint8\_t operation) : Permet de voir si l'opération (avec des opérandes SIGNES) émet un dépassement. Utile pour le V flag. Il retourne 1 si il y a un dépassement, 0 sinon.   
  
void Is\_S(arm\_core p, uint32\_t ins, int Cbit, int Vbit) : Regarde la valeur de S et modifie les flags si celui-ci est à 1. Il n'est important QUE dans les opérateurs qui modifient les flags de façon facultatif (ADD, AND, ORR, MOV etc...).  
Les opérateurs de test (TST, TEQ, CMP, CMN) ignorent cette fonction car ils modifient obligatoirement les flags.  
  
uint32\_t shiftval(arm\_core p, uint32\_t ins, int \*shifter\_carry\_out) : Permet d'extraire la valeur du shifter\_operand, en prenant en compte si c'est un registre ou une valeur immédiate, les rotations etc...

Instructions de BRANCHEMENT

B/BL

Par exemple :

BL <fonction>

###### COde : ARM\_BRANCH\_OTHER.C

int arm\_branch(arm\_core p, uint32\_t ins) : fonction principale. Effectue le branchement.

AUTRES Instructions

MRS

Par exemple :

MRS R0, CPSR

###### COde : ARM\_DATA\_PROCESSING.C

arm\_data\_processing\_immediate\_msr(arm\_core p, uint32\_t ins) : équivalent du MOV, mais pour les registres spéciaux (CPSR et SPSR).

Instructions LOAD/STORE

LDR, LDRB, ~~LDRH~~, STR, STRB, ~~STRH~~, LDM(1), STM(1)

Par exemple :

/

###### COde : ARM\_LOAD\_STORE.C

int arm\_load\_store(arm\_core p, uint32\_t ins) :

int arm\_load\_store\_multiple(arm\_core p, uint32\_t ins) :

GESTION D’EXCEPTIONS/INTERRUPTION

Le programme reconnait l’erreur quand elle est présente, et change les registres en conséquence.

###### COde : ARM\_LOAD\_STORE.C

void arm\_exception(arm\_core p, unsigned char exception) : Prend une exception en paramètre (RESET, UNDEFINED\_INSTRUCTION, SOFTWARE\_INTERRUPT, PREFETCH\_ABORT, DATA\_ABORT, INTERRUPT et FAST\_INTERRUPT) et modifie le CPSR, LR et PC en conséquence, en prenant en compte le mode d’utilisateur.

Ces fonctions sont gérés dans **arm\_instruction .c**, qui analyse le type d’instruction, vérifie les conditions et exécute l’instruction selon le schéma classique : FETCH -> DECODE -> EXECUTE.

### FONCTIONS TESTéS & BUG CONNUS

Instructions qui passent tous les tests correctement :

* Les instructions data-processing de base (sans l’option S).
* Les shift sur la deuxième opérande (shift\_operand) : LSR, LSL, ASR et ROR.
* Les branchements : B/BL
* Les Load/store classiques : LDR, STR…
* Les Load/store multiples : LDM, et STM.
* Les registres
* La mémoire.
* L’instruction MRS.

Les bugs connus :

* L’option S dans les instructions data-processing a un comportement ambigu.
* LDRH et STRH ont été oubliés.
* Les exceptions fonctionnent de manière générale mais certains cas ne sont pas vus comme des exceptions. (Entre autre quand un registre a une valeur UNPREDICTABLE.)
* Les interruptions n’ont pas été testées.

###### Tests Effectués :

/

### JOURNAL DE BORD ET REPARTITION DU TRAVAIL

Plusieurs supports ont été utilisés pour faciliter le travail :

* **Un forum pour les explications et la répartition du travail + Journal de bord** : <http://infoproject-im2ag.forumactif.com/>
* **Github** : <https://github.com/Manaelle/PROG5_Projet/tree/master/arm_simulator-1.4>

**Le journal de bord est disponible sur le forum :**

<http://infoproject-im2ag.forumactif.com/f3-planning>

Répartition du travail :

* **Implémentation de la mémoire (memory.c) + test** : Antonin, Elisa & Hanyuan
* **Implémentation des registres (register.c) + test :** Manal
* **Implémentation de traitement d’une instruction (arm\_instruction.c) :** *Tout le monde*
* **Implémentation des instructions en data-processing :** 
  + AND, EOR, SUB, RSB, ADD, ADC, SBC, RSC : Elisa
  + TST, TEQ, CMP, CMN, ORR, MOV, BIC, MVN : Manal
* **Les fonctions annexes et communes (shiftval, carry\_out, Is\_S etc…)** : Elisa&Manal
* **Implémentation des branchements :** Antonin
* **Implémentation des load/store :** Hanyuan
* **Implémentation de MRS :** Manal
* **Implémentation des exceptions :** Manal
* **Test de toutes les instructions :** Antonin
* **Secrétaire (Git + maintenance du forum, actualisation du journal de bord, suivi général) :** Manal