BOUNHACH Walid
DA COSTA BARROS Fabien
DIA Mame Alassane
ELFAKHARANY Hussein
MAHAMAT Tahir Ali
MARASCO Victor

Université Grenoble Alpes

Projet Logiciel
Simulateur ARM

Sommaire :

1 – Mode d'emploi

Mode d'emploi compilation
git clone https://github.com/Koretada/PROG5.git

cd PROG5/

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

make

Pour l'exécution cette dernière fonctionne comme décrite dans le sujet en partie "*2.2. Utilisation*"

2. Résumé des fonctions

Dans cette partie nous parlerons des principales fonctions que nous avons implémentés ainsi que des fichiers correspondants et en dernier nous aborderons les fonctionnalités manquantes.

2.1 Les fonctions de Register.c:

Dans ce fichier est implémenté les fonctions permettant la création et la gestion des registres de notre processeurs.Nous ne gérons ici que le mode usr.

a. La fonction registers_create():

Cette fonction permet la creation des registres de notre processeurs.Elle a pour signature :

```
registers registers_create()
```

b. La fonction registers_destroy :

Cette fonction permet de détruire les registres en désallouant la mémoire qui leur a été aloué.Elle a pour signature :

```
void registers_destroy(registers r)
```

c. La fonction get_mode :

Cette permet de connaitre le mode processeur.

```
uint8_t get_mode(registers r)
```

d. La fonction current_mode_has_spsr :

Cette fonction permet de savoir si le mode à l'accès a le spsr .Elle a pour signature :

```
int current_mode_has_spsr(registers r)
```

e. **La fonction in_a_privileged_mode :**

Cette fonction permet de savoir si le mode à l'accès est un mode privilégié .Elle a pour signature :

```
int in_a_privileged_mode(registers r)
```

f. **La fonction read_register :**

Cette fonction permet de connaitre la valeurs des registres généraux sp,pc,lr compris .Elle a pour signature :

```
uint32_t read_register(registers r, uint8_t reg)
```

g. **La fonction read_usr_register :**

Cette fonction renvoie le registre en mode usr .Elle a pour signature :

```
uint32_t read_usr_register(registers r, uint8_t reg)
```

h. **La fonction read_cpsr :**

Cette fonction renvoie la valeur du cpsr .Elle a pour signature :

```
uint32_t read_cpsr(registers r)
```

i. **La fonction read_spsr :**

Cette fonction renvoie la valeur du spsr .Elle a pour signature :

```
uint32_t read_spsr(registers r)
```

j. **La fonction write_register:**

Cette fonction prend en paramètre un registre (différent de spsr et de cpsr) et une valeur puis écrit cette valeur dans le registre .Elle a pour signature :

```
void write_register(registers r, uint8_t reg, uint32_t value)
```

k. La fonction write_usr_register:

A la différence de write_register cette fonction cette fonction stocke la valeur dans un registre de mode usr .Elle a pour signature :

```
void write_usr_register(registers r, uint8_t reg, uint32_t value)
```

l. La fonction write_cpsr:

Cette fonction permet d'écrire une valeur dans cpsr .Elle a pour signature :

```
void write_cpsr(registers r, uint32_t value)
```

m. La fonction write_spsr:

Cette fonction permet d'écrire une valeur dans spsr.Elle a pour signature :

```
void write_spsr(registers r, uint32_t value)
```

2.2 Les fonctions de memory.c:

Dans ce fichier est implémenté les fonctions permettant la création et la gestion de la mémoire de notre processeur.

a. La fonction memory_create:

Cette fonction prend en paramètre la taille de la mémoire et son endianesse puis retourne une mémoire avec les spécifications demandées.Elle a pour signature :

```
memory memory_create(size_t size, int is_big_endian)
```

b. La fonction memory_get_size:

Cette fonction prend en paramètre une mémoire puis retourne sa taille .Elle a pour signature :

```
size_t memory_get_size(memory mem)
```

c. La fonction memory_destroy :

Cette fonction prend en paramètre une mémoire et le détruit en libérant la mémoire qui lui a été alouée . Elle a pour signature :

```
void memory_destroy(memory mem)
```

d. La fonction memory_read_byte:

Cette fonction prend en parametre une adresse, une mémoire et un pointeur sur un uint8_t value.Elle lit dans notre mémoire à l'adresse donnée un octet et le stock dans value.Elle a pour signature :

```
memory_read_byte(memory mem, uint32_t address, uint8_t *value)
```

e. La fonction memory_read_half:

A la différence de memory_read_byte ,elle lit dans notre mémoire à l'adresse donnée un demi-mot c'est à dire deux octets et le stock dans value. Elle a pour signature :

```
memory_read_half(memory mem, uint32_t address, uint8_t *value)
```

f. La fonction memory_read_word:

A la différence de memory_read_byte ,elle lit dans notre mémoire à l'adresse donnée un mot c'est à dire quatre octets et le stock dans value. Elle a pour signature :

```
memory_read_word(memory mem, uint32_t address, uint8_t *value)
```

g. La fonction memory_write_byte:

Cette fonction permet la création des registres de notre processeurs.Elle a pour signature :

```
int memory_write_byte(memory mem, uint32_t address, uint8_t value)
```

h. La fonction memory_write_byte:

A la différence de memory_read_byte qui faisait une lecture ici nous allons écrire valeur du byte contenu dans value à l'adresse address.Elle a pour signature :

```
int memory_write_byte(memory mem, uint32_t address, uint8_t value)
```

i. La fonction memory_write_half f:

A la difference de memory_read_half qui faisait une lecture ici nous allons écrire valeur du demi-mot contenu dans value à l'adresse address.Elle a pour signature :

```
int memory_write_half(memory mem, uint32_t address, uint8_t value)
```

j. La fonction memory_write_word:

A la différence de memory_read_word qui faisait une lecture ici nous allons écrire valeur du mot contenu dans value à l'adresse address.Elle a pour signature :

```
int memory_write_word(memory mem, uint32_t address, uint8_t value)
```

2.3 Les fonctions de arm_branch_other.c:

Dans ce fichier est implémenté les fonctions permettant la gestion des branchements.

a. La fonction arm_branch:

Prends-en entrée le cœur arm p et instruction ins. Elle gère les branchements.Elle a pour signature :

```
int arm_branch(arm_core p, uint32_t ins)
```

2.4 Les fonctions de arm_data_processing.c:

Dans ce fichier est implémenté les fonctions permettant la gestion des instructions de traitement de donnée.

a. La fonction updatezn:

Elle fonction prend en parametre le un arm core p et un uint32_t et elle met à jour les flag Z et N en fonction de ce uint32_t .Elle a pour signature :

```
void updateZN(arm_core p,uint32_t result)
```

b. La fonction arm_and :

Cette fonction prend en paramètre un arm_core p deux uint32_t et le bit s permettant de savoir si on est en mode user. La fonction fait un et bit à bit des deux uint32t_t et suivant la valeur s elle mettra les flags à jour.Elle a pour signature :

```
uint32_t arm_and (arm_core p,uint32_t val1, uint32_t val2 ,uint8_t s)
```

c. La fonction arm_eor :

Cette fonction prend en paramètre un arm_core p deux uint32_t et le bit s permettant de savoir si on est en mode user. La fonction fait un ou exclusif des deux uint32t_t et suivant la valeur s elle mettra les flags à jour. Elle a pour signature :

```
uint32_t arm_eor (arm_core p,uint32_t val1, uint32_t val2 ,uint8_t s)
```

d. La fonction arm_sub :

Cette prend en paramètre un arm core p deux uint32_t, le bit v ,le bit c et s permettant de savoir si on est en mode user. La fonction fait une soustraction entre les deux uint32t_t et suivant la valeur s elle mettra les flags à jour. Elle a pour signature :

```
uint32_t arm_sub (arm_core p,uint32_t val1, uint32_t val2,uint8_t s,uint8_t c,uint8_t v)
```

e. La fonction arm_add :

Cette prend en paramètre un arm core p deux uint32_t, le bit v ,le bit c et s permettant de savoir si on est en mode user. La fonction fait une addition entre les deux uint32t_t et suivant la valeur s elle mettra les flags à jour. Elle a pour signature :

```
uint32_t arm_add (arm_core p,uint32_t val1, uint32_t val2,uint8_t s,uint8_t c,uint8_t v)
```

f. La fonction arm_orr :

Cette fonction prend en paramètre un arm_core p deux uint32_t et le bit s permettant de savoir si on est en mode user. La fonction fait un or entre les deux uint32t_t et suivant la valeur s elle mettra les flags à jour. Elle a pour signature :

```
uint32_t arm_orr (arm_core p,uint32_t val1, uint32_t val2 ,uint8_t s)
```

g. La fonction arm_bic :

Cette fonction prend en paramètre un arm_core p deux uint32_t et le bit s permettant de savoir si on est en mode user. La fonction(Bit Clear) exécute une opération ET sur les bits de val1 avec les compléments des bits correspondants à val2 et suivant la valeur s elle mettra les flags à jour. Elle a pour signature :

```
uint32_t arm_bic (arm_core p,uint32_t val1, uint32_t val2 ,uint8_t s)
```

h. La fonction arm_mov :

Cette fonction prend en paramètre un arm_core p, un registre de destination, une valeur et le bit s permettant de savoir si on est en mode user. La fonction

va écrire la valeur dans le registre de destination puis suivant la valeur de s mettra à jour les flags. Elle a pour signature :

```
void arm_mov(arm_core p,uint8_t rd,uint32_t val_2,uint8_t s)
```

i. La fonction arm_sbc :

Cette fonction prend en paramètre un arm_core p deux uint32_t,le bit c et le bit s permettant de savoir si on est en mode user. La fonction fait une soustraction avec le carry c c'est à dire le bit de retenu puis suivant s mettra à jour les flags. Elle a pour signature :

```
uint32_t arm_sbc(arm_core p,uint32_t val1, uint32_t val2, uint8_t c,uint8_t s)
```

j. La fonction arm_adc :

Cette fonction prend en paramètre un arm_core p deux uint32_t,le bit c et le bit s permettant de savoir si on est en mode user. La fonction fait une addition avec le carry c c'est à dire le bit de retenu puis suivant s mettra à jour les flags. Elle a pour signature :

```
uint32_t arm_adc(arm_core p,uint32_t val1, uint32_t val2, uint8_t c,uint8_t s)
```

k. La fonction arm_tst :

La fonction fait un teste val1 et val2 puis en fonction du résultat met à jour les flags. Elle fait un et bit a bit entre les valeurs val1 et val2 sauf que le résultat n'est pas stocké.Elle a pour signature :

```
void arm_tst(arm_core p,uint32_t val1,uint32_t val2)
```

l. La fonction arm_teq :

La fonction fait un teste val1 et val2 puis en fonction du résultat met à jour les flags. Elle fait un ou exclusif entre les valeurs sauf que le résultat n'est pas stocké.Elle a pour signature :

```
void arm_tst(arm_core p,uint32_t val1,uint32_t val2)
```

m.La fonction arm_cmp:

Cette fonction prend en paramètre un arm_core p deux uint32_t, le bit c, le bit v. Elle ressemble au sub sauf que le résultat est ignoré et servira juste à mettre à jour les flags. La fonction permet de comparer les valeur val1 et val2.Elle a pour signature :

```
void arm_cmp(arm_core p,uint32_t val1,uint32_t val2, uint8_t  c, uint8_t v)
```

n. La fonction arm_cmn:

Cette fonction prend en paramètre un arm_core p deux uint32_t, le bit c, le bit v. Elle ressemble au add sauf que le résultat est ignoré et servira juste à mettre à jour les flags. La fonction permet de comparer les valeur val1 et val2.Elle a pour signature :

```
void arm_cmn(arm_core p,uint32_t val1, uint32_t val2, uint8_t c, uint8_t v)
```

o. La fonction arm_rsb :

Cette prend en paramètre un arm core p, un registre de destination,deux uint32_t, le bit v, le bit c et s permettant de savoir si on est en mode user. La fonction fait une soustraction
Entre les deux uint32t_t puis le stock dans le registre de destination et suivant la valeur s elle mettra les flags à jour. Elle a pour signature :

```
void arm_rsb(arm_core p, uint8_t rd, uint32_t val_1,uint32_t val_2,
uint8_t  s, uint8_t c, uint8_t v)
```

p. La fonction arm_rsc :

Cette prend en paramètre un arm core p, un registre de destination, deux uint32_t, le bit v, le bit c et s permettant de savoir si on est en mode user. La fonction fait une soustraction avec le carry entre les deux uint32t_t puis le stock dans le registre de destination et suivant la valeur s elle mettra les flags à jour. Elle a pour signature :

```
void arm_rsc(arm_core p, uint8_t rd, uint32_t val_1,uint32_t val_2,
uint8_t  s, uint8_t c, uint8_t v)
```

q. La fonction apply_rotation_imm :

La fonction applique une une rotation de rotate_imm*2 a immed_8.
rotate_imm se trouvant dans l'instruction entre le bit 9 et 12
immed_8 se trouvant dans ins entre le bit 0 et 11
Elle a pour signature.

```
uint32_t apply_rotation_imm(uint32_t ins)
```

r. La fonction decode_operand:

Cette fonction prend en paramètre un arm core p, l'instruction, et deux pointeurs qui pointent sur val1 et val2.Elle fournit à val1 et val2 les valeurs des opérandes.
Elle a pour signature :

```
int decode_operand(arm_core p, uint32_t ins, uint32_t *val_1, uint32_t
*val_2)
```

s. La fonction arm_data_processing_shift:

Cette fonction découpe l'instruction qu'elle reçoit en paramètre et récupère l'opcode, les flags les opérandes etc... Et suivant l'opcode elle effectuera le

traitement correspondant en faisant appel à la fonction requise. Elle a pour signature :

```
int arm_data_processing_shift(arm_core p, uint32_t ins)
```

t. **La fonction arm_data_processing_shift:**

Cette fonction permet de charger une valeur immédiate dans les champs spécifiés d'un registre d'état de programme (psr).

```
int arm_data_processing_shift(arm_core p, uint32_t ins)
```

2.5 Les fonctions de arm_instruction.c:

Dans ce fichier est implémenté les fonctions permettant la gestion instructions.

a. **La fonction arm_execute_instruction:**

Cette fonction prend le cœur arm p et fait appel à la fonction fournit arm_fetch() pour récupérer l'instruction. Une fois l'instruction récupérée on fait un get_bit(instruction, 27,25) pour récupérer ces bits car en arm ils définissent l'instruction nous permettant ainsi de savoir si on a une instruction de contrôle (branchement), une instruction instructions de traitement de données (arithmétiques et logiques), une Instructions de manipulation de registres, instructions divers (que nous ne traitons pas).
Elle a pour signature :

```
static int arm_execute_instruction(arm_core p)
```

2.6 Les fonctions de arm_load_store.c:

Dans ce fichier est implémenté les fonctions permettant la gestion des instructions de manipulation de registres.

a. La fonction arm_load_store :

Cette fonction permet de charger une immédiate ou la valeur d'un registre d'un registre ou de stocker cette valeur en mémoire. Cette valeur peut être ici soit un byte ou un mot. Sa signature est :

```
int arm_load_store(arm_core p, uint32_t ins)
```

b. La fonction arm_load_store_multiple :

Cette fonction marche comme la précédente sauf qu'ici on peut charger ou stocker plusieurs valeurs. Sa signature est :

```
int arm_load_store_multiple(arm_core p, uint32_t ins)
```

c. La fonction arm_load_store_half :

La différence avec cette fonction est qu'on travaille ici avec les demi-mots. Sa signature est :

```
int arm_load_store_half(arm_core p, uint32_t ins)
```

2.7 Les fonctionnalités manquantes:

Les fonctionnalités manquantes à notre travail sont :
-l'implémentation de la fonction miscellaneous
-La gestion des modes processeurs

3 – Description des tests

3.1 – Branchements

```
1   .global main
2   .text
3   decr:
4       subs r0, r0, #1
5       mov pc, lr
6
7   main:
8       mov r0, #5
9   loop:
10      bl decr
11      bne loop
12  end:
13      swi 0x123456
14  .data
```

example1.s

Ce premier test simple nous sert à montrer la fonctionnalité d'un premier branchement et des différentes instructions de traitement de données. En effet, le bl est un branchement link. Il effectue un branchement sur une fonction decr. En l'occurence, nous regardons ici grâce à BNE si r0 est égal à 0.

3.2 – Traitement de données

```
1   .global main
2   .text
3   main:
4       mov r1, #30
5
6       and r2, r1, #0xf0
7       mov r2, r2, lsr #4
8       sub r3, r1, #0x0f
9       mov r3, r3, lsl #4
10      orr r1, r2, r3
11
12      mov r2, r2, lsr #2
13      adc r3, r1, #0x33
14      mov r3, r3, lsl #2
15      orr r1, r2, r3
16
17      mvn r6, #0x10
18      and r2, r1, #0xaa
19      mov r2, r2, lsr #1
20      and r3, r1, #0x55
21      mov r3, r3, lsl #1
22      orr r1, r2, r3
23  .data
24
```

data_processing_test.s

Dans ce test, nous cherchons à voir si les instructions de traitements de donnés sont comme attendues. Tout est fonctionnel, les différentes fonctions gèrent bien ces cas. A l'aide de "Info reg" on s'aperçoit que les registres se mettent bien à jour et prennent les bonnes valeurs.

3.3 – Load/Store

```
1   .global main
2   .text
3   main:
4       ldr r0, =limite
5       ldrb r1, [r0]
6       ldrb r2, [r0,#3]
7           mov r8,r0
8       mov r2,#0
9           ldrb r2, [r8,#3]!
10      ldrb r3, [r0],#3
11          sub  r4,r1,#16
12          ldrb r2, [r0,r4]
13      mov r8,r0
14      mov r2,#0
15      ldrb r2, [r8,r4]!
16      mov r3,#0
17      ldrb r3, [r0],r4
18          mov r5,#1
19      ldr r0, =limite
20      ldr r6, [r0, r5, LSL #2]
21      ldr r6, [r0, r5, LSL #2]!
22
23      strb r1, [r0]
24      strb r2, [r0,#3]
25          strb r2, [r8,#3]!
26      strb r3, [r0],#3
27          sub  r4,r1,#16
28          strb r2, [r0,r4]
29      strb r2, [r8,r4]!
30      strb r3, [r0],r4
31      ldr r0, =limite
32      str r6, [r0, r5, LSL #2]
33      str r6, [r0, r5, LSL #2]!
34
35      swi 0x123456
36  .data
37  limite:
38      .word 0x12015678
39          .word 0x11223344
```

LdrStr.s

Ce test mélangeant les load/store sur un word et sur un byte est fonctionnel. Il montre en partit la fonctionnalité des nos différentes instructions. L'idée ici étant de mélanger les word et byte pour voir si nos instructions fonctionnaient en toute circonstances.

```
1   .global main
2   .text
3   main:
4       MOV r5,#1
5       ADD r5, r5, LSL #12
6       SUB r5,r5,#1
7       MOV r1,#11
8       MOV r2,#33
9       MOV r3,#138
10      MOV r6,#126
11      STMIA r5,{r1-r3,r6}
12      ADD r5,#12
13      LDMDA r5,{r8-r11}
14      MOV r5,#136
15      LDMDB r5,{r7-r10}
16      MOV r5,#116
17      LDMIB r5,{r8-r11}
18      MOV r5,#120
19      LDMIA r5,{r7-r10}
20
21      swi 0x123456
22  .data
23  limite:
24      .word 0x01234567
25          .word 0x89abcdef
26      .word 0x00112233
27      .word 0x44556677
28      .word 0x8899aabb
29      .word 0xccddeeff
```

LdmStm.s

De même pour ce test, après de multiples essais et corrections, tout fonctionnent comme nous le souhaitions. Ici, on cherche à vérifier que les LDM* et STM* se déroulent comme il faudrait.

4 - Listes des bogues

- Mov sur le register 15 qui ne fonctionne pas.
- Bx Lr qui ne se fonctionne pas comme l'on voudrait.

5 – Journal de code

Le 04/01 :
- Début du projet
- Réalisation de register et de memory individuellement et appel sur discord pour la mise en commun.

Le 05/01 :
- Retour sur la finalisation de register.c et remise en question sur le type de structure que l'on pourrait utiliser au niveau du registre. (Walid, Fabien, Hussein, Alassane)
- Compréhension des autres fichiers, des fonctions appelantes…
- Push des versions finales de register.c et memory.c (Fabien, Walid)

Le 06/01 :
- Réalisation de arm_branch_other.c (Hussein, Fabien, Victor)
- Brainstorming sur arm_load_store (Walid, Alassane, Tahir)

Le 07/01 :
- Avancement de arm_load_store (Walid, Alassane, Tahir)
- Oral de notre groupe 18
- Changement de petit détail sur arm_instruction.c (Hussein, Victor)
- Réalisation de arm_data_processing (Fabien, Hussein, Walid)

Le 11/01 :
- Retour sur Load/store (Walid, Fabien, Tahir)
- Mise en place des résolutions d'erreurs (Walid, Fabien, Tahir, Alassane)

Le 12/01 :
- Réalisations des tests sur les loads/store (Walid, Fabien, Victor)

- Réalisations des tests sur les instructions de branchements (Hussein, Tahir, Alassane)

Le 13/01 :
- Préparation du compte-rendu (Walid, Alassane)
- Questionnement sur la fonctionnalités de certains tests et fonctions du load/store (Fabien, Walid)
- Finalisation de certains détails sur les branchements (Hussein, Alassane, Victor)

6 – Conclusion

Nous nous sommes répartit très rapidement les différentes tâches à effectuer. En commençant par les fichiers Registers.c et Memory.c. Puis, chacun à apporté sa pierre à l'édifice mais certains plus que d'autres. Toutefois, nous faisions des "call discords" quotidiennement afin d'aider ceux qui n'avaient pas forcément compris les différentes fonctions. L'esprit d'équipe était la clé de notre progression et avancement mutuelle.

Le projet était essentiellement composé d'ARM. La compréhension n'était pas si facile que cela mais la documentation à grandement contribué l'avancement du projet. En effet, nos connaissances dans l'UE d'architecture logiciel et matériel ont permis de s'aider mutuellement. Le Mattermost était d'une très grande aide notamment par la réponse des professeurs à nos différentes questions. Ce projet de fin de semestre reste majoritairement très fructueux pour la consolidation de nos connaissances.