

**Groupe 16 :**

**BLEJ Mouad**

**BALADI Ayoub**

**BOULAGHLA Aderrazzak**

**EL BOUCHOUARI Zakaria**

**HOUSSEIN KASSIM Abro**

**LUIGGI Alexandre**

Rapport du projet

**Mode d’emloi :**

Make : pour compiler tous les programmes.

./memory\_test : un test pour les opérations de la mémoire.

./arm\_simulator et gdb-multiarch dans deux terminaux pour faire tester tous nos programmes.

Descriptif de la structure du code développé :

arm\_branch\_other.c :

arm\_data\_processing.c:

\_arm\_data\_processing\_immediate\_msr

\_arm\_data\_processing\_shift : cette fonction calcule tous la valeur du Registre (Shifter Operand), et aussi le carry des shift (Sco) .

arm\_load\_store.c :

\_ laod\_stot\_byte\_word : cette fonction charge ou écrit dans la mémoire un octet ou mot

\_ laod\_stot\_halfword : cette fonction charge ou écrit dans la mémoire un demi-mot

\_ est\_immediate : cette fonction s’en occupe da la valeur de l’offset dans le cas d’un octet ou d’un mot

\_ half\_immediate : cette fonction s’en occupe de la valeur de l’offset dans le cas d’un demi-mot

\_ half\_word : cette fonction s’en occupe de différents types d’adressage dans le cas d’un demi-mot

\_ byte\_word : cette fonction s’en occupe de différents types d’adressage dans le cas d’un mot ou d’un octet

\_ arm\_load\_store : la fonction générale qui s’en occupe de load et store dans le cas d’un mot, demi-mot et octet, en prenant en compte des différents types d’adressage

\_ creer\_reglist : créée un tableau contenant les numéros de registre qu’elle récupère en analysant les 16 premier bits d’une instruction STM ou LDM.

\_ check\_reg\_plag : Vérifie si l’ensemble des registres d’un tableau remplis par créer\_reglist vérifie les conditions pour que l’instruction puisse être exécuter.

\_ arm\_load\_store\_multiple : fonction d’implémentation pour les instructions STM et LDM prenant en compte les suffixes optionnel (IA, IB, DA, DB)

\_ arm\_coprocessor\_load\_store : fonction contenant l’implémentation de la fonction MRS

arm\_instruction.c :

utile.c :

La fonction **do\_shift**(uint32\_t \* Sop, uint8\_t \* sco , uint32\_t data , uint32\_t val\_Rm ,uint8\_t shift , int mode , uint8\_t c) dans util.c qui prend en paramètre Sop (Shift Operator) , sco (Shift carry out ), data soit la valeur d'un registre ou bien une constante ( Immediate ), val\_Rm c'est la valeur du registre Rm ( voir la page A5-3 de la documentation ) , shift ce sont les bit 5 et 6 de l'instruction, mode soit IMMEDIATE ou REGISTER (0 ou 1 ) , et le flag c de cspr. La fonction modifie la valeur de Sop et de sco pour que la fonction **execute\_instruction** fasse son travail.

add : la somme de deux registre r0 r1 dans un troisième registre r2

la somme d'un registre avec une valeur immédiate

adc : il calcule le carry

adcs : mise ajours des flags test over flow z=1 c =1

adds : mise a jours des flags n=1 v=1

and : et logique entre deux registres

et logique entre registre et valeur immédiat

xor :

ands : mise a jours des flags

eor : mise à jour de n et z

eors : mise à jour de n et z

sub : soustraction de deux registre r0 r1 dans un troisième registre r2

la soustraction d'un registre avec une valeur immédiate

subs : mise a jours des flags

sbc : soustrait le négatif de "carry"

sbcs : mise a jours des flags

rsb : reverse la formule de soustraction

rsbs : mise ajours des flags

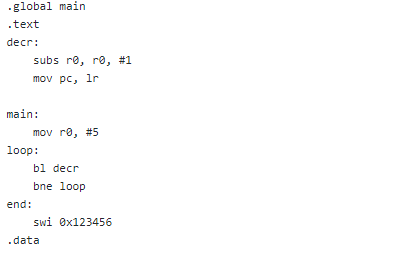
**Listes de bogues (instructions) :**

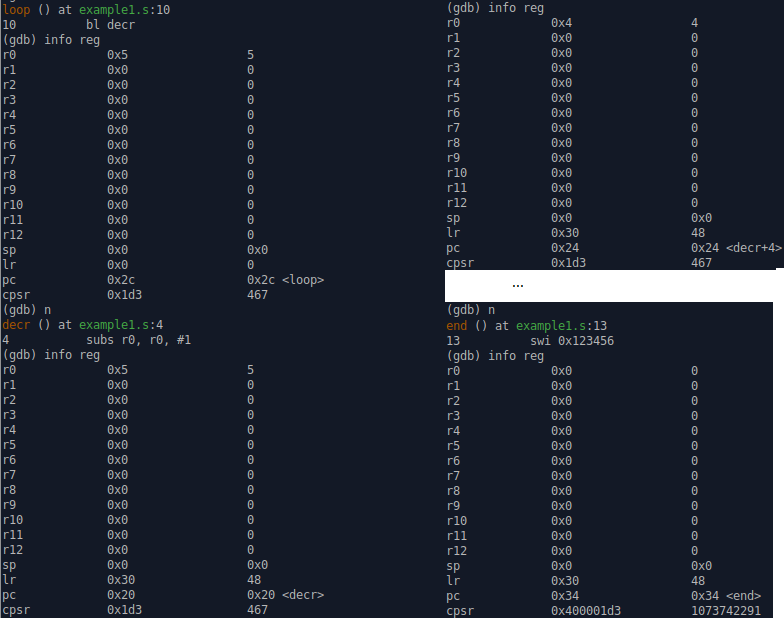
-STM (tester mais beug)

-MRS (pas tester)

**Test :**

**example1 :**

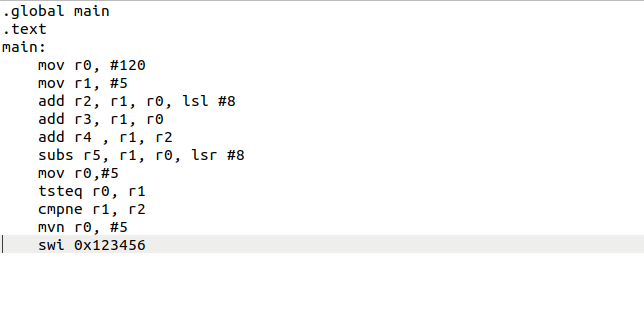




tester les instruction de branchement

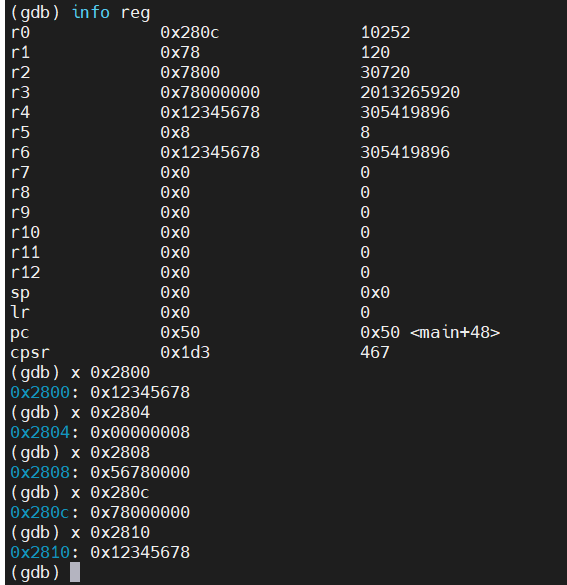
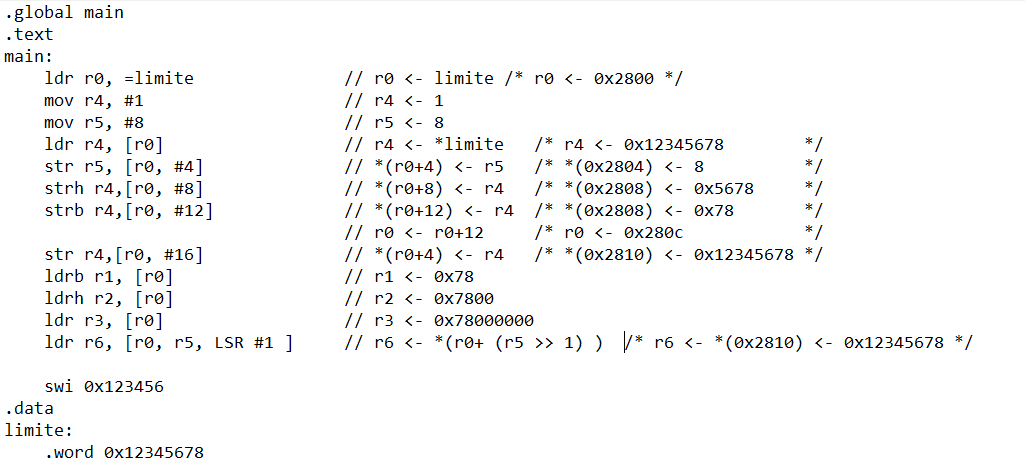
le résultat : tant que r0 diffèrent de 0 on décrément la valeur de r0 par appel de l'étiquette ‘decr ‘

**example2 :**

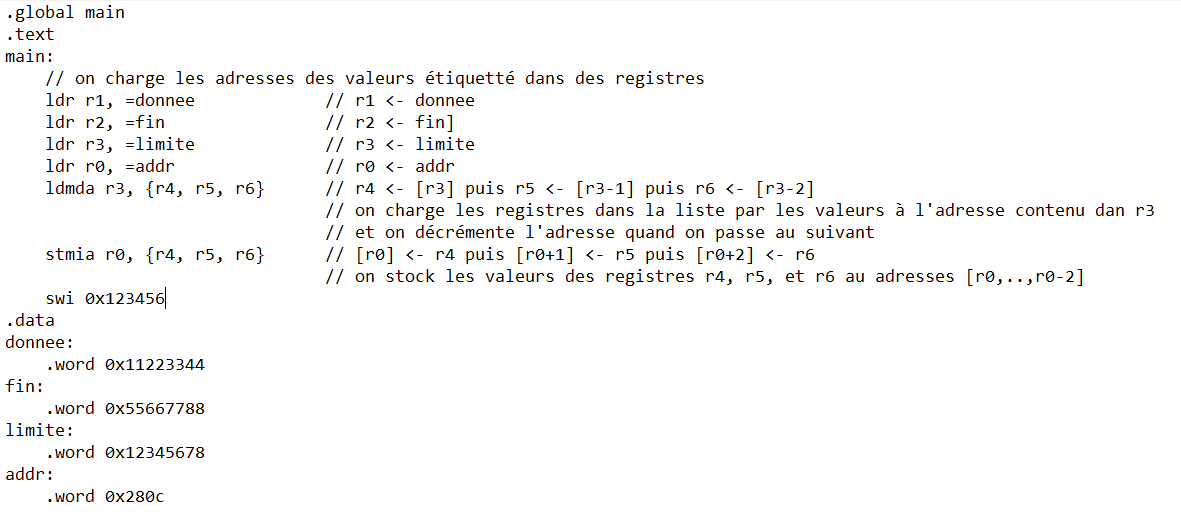


cet exemple présente le test des instructions de traitement de données avec des shifts.

**example3 :**



example4 :



Répartition du travail et progression :

Le travail a été répartis entre les trois binômes du groupe. Tout d’abord nous avons due compléter les fichier registers.c et memory.c. Ayoub, Abderrazzak et Mouad pour le fichier registers.c et Abro et Zakaria pour le fichier memory.c. Ensuite chaque binôme a dû implémenter un certain nombre d’instructions en fonction des fichiers à compléter.

arm\_branch\_other.c et arm\_data\_processing.c compléter par Abderrazzak et Ayoub Instructions implémenté  : B/BL, AND, EOR, SUB, RSB, ADD, ADC, SBC

arm\_data\_processing.c compléter par Mouad et Alexandre Instructions implémenté : RSC, TST, TEQ, CMP, CMN, ORR, MOV, BIC, MVN

Fonctions implémentée : arm\_data\_processing\_shift, do\_shift.

arm\_load\_store.c compléter par Abro et Zakaria Instructions implémenté  : LDR, LDRB, LDRH, STR, STRB, STRH, LDM(1), STM(1), MRS

La fonction **arm\_execute\_instruction** du fichier arm\_instruction.c a été compléter indépendamment par chaque binôme afin de s’assurer du bon fonctionnement des instructions qu’ils ont ainsi implémentés. La fonction finale est complétée à partir des trois versions.