MI41 – Automne 2004 - TP5

Assembleur ARM

Rapport

Étienne Monzain
Laurent Couvidou

Introduction

Le but de ce TP est l'initiation à la programmation assembleur ARM. Il s'agit de trois exercices distincts, de difficulté progressive.

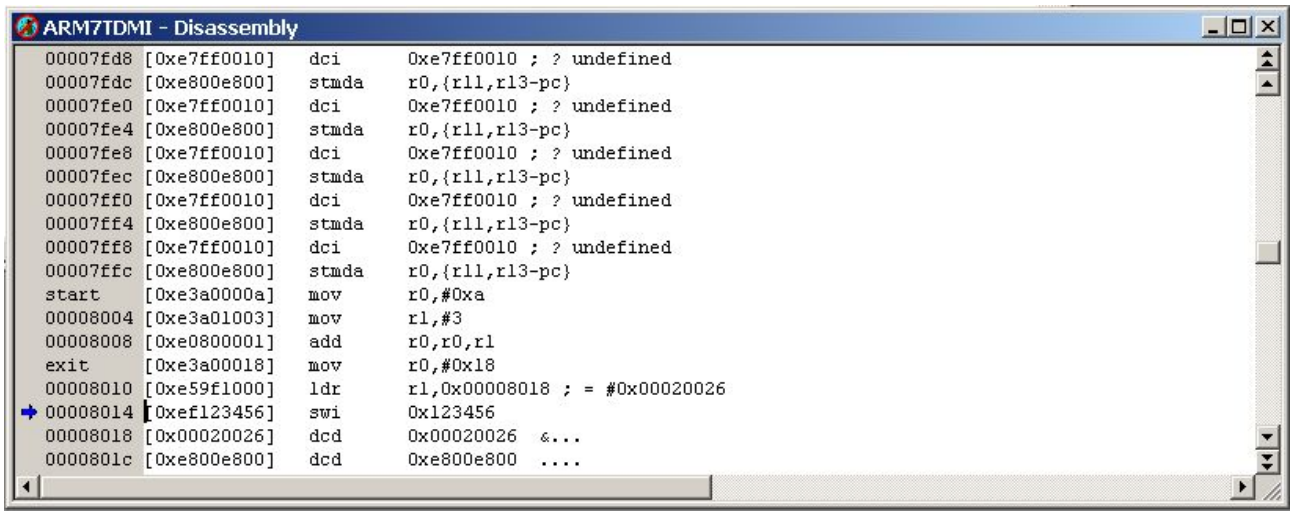
Sommaire

1. Prise en main

On tape le programme donné :

```
    AREA EX1, CODE, READONLY   ; nom du bloc de code
    ENTRY                      ; entrée du code
start                          ; label
    MOV r0,#10
    MOV r1,#3
    ADD r0,r0,r1
```

Et on l'exécute en mode debug :



1.1. À quelle adresse se trouve le code ?

C'est la colonne de gauche. Le code commence donc en 00008000.

1.2. Quelle est la valeur du pointeur de pile ?

C'est la 2ème colonne à partir de la gauche. Ici le pointeur de pile commence à [0xe3a0000a].

1.3. Combien d'instructions contient ce code ?

Trois.

1.4. Que se passe-t-il quand on continue d'exécuter des instructions après la dernière ? Comment y remédier ?

L'instruction suivante n'est pas une instruction *valable*. On obtient donc une erreur : « Processor ARM7TDMI raised an exception. Cause : undefined instruction. »

Pour résoudre le problème, on modifie le code de la manière suivante :

```
    AREA EX1, CODE, READONLY   ; nom du bloc de code
    ENTRY                      ; entrée du code
start                          ; label
    MOV r0,#10
    MOV r1,#3
    ADD r0,r0,r1
exit
    MOV r0,#0x18               ; angel_SWIreason_reportException
    LDR r1,=0x20026            ; ADP_Stopped_ApplicationExit
    SWI 0x123456               ; ARM semishooting SWI;
    END                        ; fin du fichier
```

Il s'agit d'instructions spéciales permettant l'arrêt de l'exécution.

2. Test de grandeurs

2.1. Tri de trois valeurs signées

On trie trois valeurs entières (signées) placées dans R0, R1 et R2. Cette manipulation nécessite de passer par un quatrième registre : R3.

```
    AREA EX1, CODE, READONLY   ; nom du bloc de code
    ENTRY                      ; entrée du code
start                          ; label
    MOV r0,#24
    MOV r1,#65
    MOV r2,#-12
    MOV r3,#0
tri1                           ; 1er tri éventuel
    CMP r0,r1
    BLE tri2                   ; si non(r0<=r1)
    MOV r3,r0                  ; on échange r0 et r1
    MOV r0,r1                  ; en passant par r3
    MOV r1,r3
tri2                           ; 2eme tri éventuel
    CMP r1,r2
    BLE tri3                   ; si non(r1<=r2)
    MOV r3,r1                  ; on échange r1 et r2
    MOV r1,r2                  ; toujours en passant par r3
    MOV r2,r3
tri3                           ; 3eme tri éventuel
    CMP r0,r1
    BLE exit                   ; si non(r0<=r1) (possible après tri2)
    MOV r3,r0                  ; on échange r0 et r1
    MOV r0,r1                  ; en passant par r3
    MOV r1,r3
exit
    MOV r0,#0x18               ; angel_SWIreason_reportException
    LDR r1,=0x20026            ; ADP_Stopped_ApplicationExit
    SWI 0x123456               ; ARM semishooting SWI;
    END                        ; fin du fichier
```

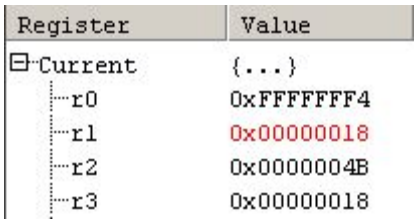
La commande LE considère les valeurs comme signées. BLE permet de faire un branchement vers le label correspondant *si le prmeir opérateur du CMP précédent est inférieur ou egal (Lower or Equal) au deuxième*.

Vérifions que le programme fonctionne.

Valeurs des registres avant le lancement :
(nota : en hexadécimal 24=0x18 ; 65=0x4B ; -12=0xFFFFFFF4)

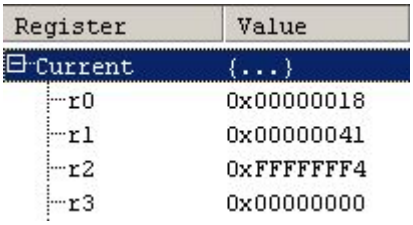
Register	Value
⊟ Current	{...}
r0	0x00000018
r1	0x0000004B
r2	0xFFFFFFF4
r3	0x00000000

Après exécution, les valeurs sont correctement triées :



2.2. Tri de trois valeurs non signées

On remplace les instructions LE par des instruction LS. Ainsi , les valeurs sont considérées comme non signées. Après exécution, on obtient donc 0xFFFFFFF4 placé comme plus grand entier.



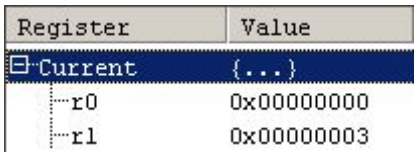
3. Boucle

On cherche à calculer le poids de Hammings d'un nombre entier, c'est-à-dire son nombre de 1.

```
    AREA EX1, CODE, READONLY
    ENTRY
start
    MOV r0,#0x13              ; nombre entier à tester
    MOV r1,#0                 ; r1 sert de compteur
boucle
    CMP r0,#0
    BEQ exit                  ; si (r0=0), fin du programme
    CMP r0,#0
    BLT compter               ; si (r0<0) (bit de signe à 1), on compte un 1
retour
    MOV r0,r0,LSL#1           ; décalage à gauche puis boucle
    B boucle
compter
    ADD r1,r1,#1              ; on compte un 1
    B retour                  ; retour dans la boucle
exit
    MOV r0,#0x18              ; angel_SWIreason_reportException
    LDR r1,=0x20026           ; ADP_Stopped_ApplicationExit
    SWI 0x123456              ; ARM semishooting SWI;
    END                       ; fin du fichier
```

En algorithmique, la boucle réalisée s'écrirait :
r0 := 13
r1 := 0
TantQue (r0 <> 0) faire
    Si (r0 < 0) faire
        r1 := r1 + 1
    fsi
    DecalageGauche (r0)
ftq

Vérifions le fonctionnement. On exécute le programme et on obtient le résultat suivant :



Au départ r0=0x13, soit en binaire : r0=(00010011). On voit que son poids de Hammings est de 3 (il y a 3 1). La valeur finale de r1 est donc correcte.

4. Appel de fonction

4.1. Quotient et reste d'une division entière non signée

On réalise l'équivalent assembleur de la boucle suivante :
r0 := 42
r1 := 9
r2 := 0
TantQue (r0 >= 1) faire
    r0 := r0-r1
    r2 := r2+1
ftq

A l'arrivée, le quotient sera dans r2 et le reste dans r0.

```
    AREA EX3, CODE, READONLY
    ENTRY
start
    MOV r0,#42                ; Numérateur
    MOV r1,#9                 ; Dénominateur
    MOV r2,#0                 ; Quotient
boucle
    CMP r0,r1
    BLO exit                  ; Division entière possible si r0>=r1, soit non(r0<r1)
    SUB r0,r0,r1              ; On soustrait le dénominateur au numérateur
    ADD r2,r2,#1              ; Le quotient est donc incrémenté de 1
    B boucle
exit
    MOV r0,#0x18              ; angel_SWIreason_reportException
    LDR r1,=0x20026           ; ADP_Stopped_ApplicationExit
    SWI 0x123456              ; ARM semishooting SWI;
    END                       ; fin du fichier
```

Malgré nos efforts, impossible de faire fonctionner le programme. Une instruction serait-elle mal écrite ? L'erreur provoqué par le debugger n'aide en rien.

Nous n'avons donc pas pu répondre aux deux questions suivantes.

Conclusion

Ce TP nous a permis d'aborder la programmation assembleur ARM. Il nous a permis de comprendre le fonctionnement des instructions processeur. Nous avons aussi pu constater que le code assembleur n'est pas très simple à manipuler et à appréhender ; l'apport des langages de programmation est donc certain à ce niveau.