INF401

TP5 « Codage de structures de contrôle et metteur au point gdb»

Shaghayegh HAJMOHAMMADKASHI , Kimia KHADEMLOU– MIN3

5.1 Accès à un tableau :

1. Récupérez le fichier tableau.s. On y a traduit en langage d'assemblage les deux premières affectations.

2. Complétez le programme de façon à réaliser l'algorithme donné en entier.

```
1     .data
2   debutTAB: .skip 5*4 @ reservation de 20 octets
3                 @ sans valeur initiale
4
5       .text
6       .global main
7   main:
8       ldr r0, ptr_debutTAB
9       mov r1, #11
10      str r1, [r0]
11
12      mov r1, #22
13      add r0, r0, #4
14      str r1, [r0]
15
16      mov r1, #33             @ Mettre la valeur 33 dans r1
17      add r0, r0, #4          @ Ajouter 4 à r0 (déplacer le pointeur au prochain élément du tableau)
18      str r1, [r0]            @ Stocker la valeur de r1 dans le troisième élément du tableau
19
20      mov r1, #44             @ Mettre la valeur 44 dans r1
21      add r0, r0, #4          @ Ajouter 4 à r0 (déplacer le pointeur au prochain élément du tableau)
22      str r1, [r0]            @ Stocker la valeur de r1 dans le quatrième élément du tableau
23
24      mov r1, #55             @ Mettre la valeur 55 dans r1
25      add r0, r0, #4          @ Ajouter 4 à r0 (déplacer le pointeur au prochain élément du tableau)
26      str r1, [r0]            @ Stocker la valeur de r1 dans le cinquième élément du tableau
27
28    fin:  BX LR
29
30    ptr_debutTAB : .word debutTAB
```

3. Compilez avec la commande : arm-eabi-gcc -Wa,--gdwarf2 tableau.s -o tableau 1

4. Observez son exécution pas à pas sous gdb ou ddd (lire ce qui suit et vous référer au paragraphe 2.4.2).

Exécutez le programme sous gdb en tapant les commandes suivantes :

1. **arm-eabi-gdb tableau**

On lance le débogueur, nous sommes désormais dans l'environnement gdb.

```
hajmohas@im2ag-turing-01:[~/INF401/TP5]: arm-eabi-gdb tableau
GNU gdb (GDB) 7.2
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-unknown-linux-gnu --target=arm-eabi".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/h/hajmohas/INF401/TP5/tableau...done.
(gdb)
```

2. **target sim**

On active le simulateur, ce qui permet d'exécuter des instructions en langage d'assemblage

ARM.

```
(gdb) target sim
Connected to the simulator.
```

3. **load**

On charge le programme à exécuter dont on a donné le nom à l'appel de gdb.

```
(gdb) load
Loading section .init, size 0x18 vma 0x8000
Loading section .text, size 0x2194 vma 0x8018
Loading section .fini, size 0x18 vma 0xa1ac
Loading section .rodata, size 0xa vma 0xa1c4
Loading section .ARM.exidx, size 0x8 vma 0xa1d0
Loading section .eh_frame, size 0x4 vma 0xa1d8
Loading section .init_array, size 0x4 vma 0x121dc
Loading section .fini_array, size 0x4 vma 0x121e0
Loading section .jcr, size 0x4 vma 0x121e4
Loading section .data, size 0x968 vma 0x121e8
Start address 0x80b0
Transfer rate: 88688 bits in <1 sec.
```

4. **break main**

On met un point d'arrêt juste avant l'étiquette main.

```
(gdb) break main
Breakpoint 1 at 0x81c0: file tableau.s, line 9.
```

5. **run**

Le programme s'exécute jusqu'au premier point d'arrêt exclu : ici, l'exécution du programme est donc arrêtée juste avant la première instruction.

```
(gdb) run
Starting program: /home/h/hajmohas/INF401/TP5/tableau

Breakpoint 1, main () at tableau.s:9
9           mov r1, #11
```

6. **list**

On voit 10 lignes du fichier source.

```
(gdb) list
4
5           .text
6           .global main
7       main:
8           ldr r0, ptr_debutTAB
9           mov r1, #11
10          str r1, [r0]
11
12          mov r1, #22
13          add r0, r0, #4
```

7. **list**

On voit les 10 suivantes.

```
(gdb) list
14          str r1, [r0]
15
16          mov r1, #33             @ Mettre la valeur 33 dans r1
17          add r0, r0, #4          @ Ajouter 4 à r0 (déplacer le pointeur au prochain élément du tableau)
18          str r1, [r0]            @ Stocker la valeur de r1 dans le troisième élément du tableau
19
20          mov r1, #44             @ Mettre la valeur 44 dans r1
21          add r0, r0, #4          @ Ajouter 4 à r0 (déplacer le pointeur au prochain élément du tableau)
22          str r1, [r0]            @ Stocker la valeur de r1 dans le quatrième élément du tableau
23
```

8. **list 10,13**

On voit les lignes 10 à 13.

```
(gdb) list 10,13
10          str r1, [r0]
11
12          mov r1, #22
13          add r0, r0, #4
```

9. **info reg**

```
(gdb) info reg
r0             0x122fc  74492
r1             0x1ffff8 2097144
r2             0x2      2
r3             0x805c   32860
r4             0x1      1
r5             0x1ffff8 2097144
r6             0x0      0
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0x200100 2097408
r11            0x0      0
r12            0x1fffe8 2097128
sp             0x1ffff8 0x1ffff8
lr             0x81a0   33184
pc             0x81c0   0x81c0 <main+4>
fps            0x0      0
cpsr           0x60000013       1610612755
```

Permet d'afficher en hexadécimal et en décimal les valeurs stockées dans tous les registres.

Notez la valeur de r15 aussi appelé pc, le compteur de programme. Elle représente l'adresse de la prochaine instruction qui sera exécutée.

10. **s**

Une instruction est exécutée. gdb affiche une ligne du fichier source qui est la prochaine instruction (et qui n'est donc pas encore exécutée).

```
(gdb) s
10          str r1, [r0]
```

11. **x/5w &debutTAB**

Sous gdb, pour afficher 5 mots en hexadécimal, à partir de l'adresse debutTAB.

```
(gdb) x/5w &debutTAB
0x122fc <debutTAB>:     0       0       0       0
0x1230c <debutTAB+16>:  0
```

Pour l'observation de l'exécution du programme tableau, notez, en particulier, les valeurs successives (à chaque itération) de r0, le contenu de la mémoire à partir de l'adresse debutTAB en début de programme et après l'exécution de toutes les instructions

```
(gdb) run
Starting program: /home/h/hajmohas/INF401/TP5/tableau

Breakpoint 1, main () at tableau.s:9
9           mov r1, #11
(gdb) info registers $r0
r0             0x122fc  74492
```

```
(gdb) next
10          str r1, [r0]
(gdb) next
12          mov r1, #22
(gdb) next
13          add r0, r0, #4
(gdb) info registers $r0
r0             0x122fc  74492
```

Au début du programme, ldr r0, ptr_debutTAB charge dans r0 l'adresse de début du tableau debutTAB.

Ensuite, à chaque itération de la boucle, r0 est utilisé pour pointer vers l'élément actuel du tableau. Après chaque stockage dans le tableau, r0 est mis à jour pour pointer vers l'élément suivant en ajoutant 4 à sa valeur. Cela se fait à l'aide des instructions add r0, r0, #4, ce qui déplace le pointeur vers l'élément suivant du tableau, car chaque entier occupe 4 octets en mémoire.

```
(gdb) next
16          mov r1, #33
(gdb) next
17          add r0, r0, #4
(gdb) next
18          str r1, [r0]
(gdb) info registers $r0
r0             0x12304  74500
(gdb) next
20          mov r1, #44
(gdb) next
21          add r0, r0, #4
(gdb) next
22          str r1, [r0]
(gdb) info registers $r0
r0             0x12308  74504
(gdb) next
24          mov r1, #55
(gdb) next
25          add r0, r0, #4
(gdb) next
26          str r1, [r0]
(gdb) info registers $r0
r0             0x1230c  74508
(gdb) next
fin () at tableau.s:28
28      fin:  BX LR
```

5.2 Codage d'une itération :

1. Récupérez le fichier iteration.s Compilez ce programme et exécutez-le sous gdb ou ddd.

On a compilé avec la commande : arm-eabi-gcc -Wa,--gdwarf2 iteration.s -o iteration.

On a exécuté sous gdb en tapant : arm-eabi-gdb iteration.

On a suivi les instructions pour charger le programme et exécuté pas à pas.

2. Quelle est la valeur contenue dans r0 à chaque itération ?

r0 est chargé avec l'adresse de début du tableau debutTAB et que cette adresse commence avec la valeur de 74476 à chaque itération, augmentant de 4 jusqu'à la valeur de 74492, alors cela signifie que chaque élément du tableau est situé à une adresse mémoire qui est espacée de 4 octets.

3. Quelle est la valeur contenue dans r2 à chaque itération ?

« r2 » est utilisé comme un compteur d'itération dans la boucle. Il est initialisé à 0 avant le début de la boucle (mov r2, #0). Puis, à chaque itération, il est incrémenté de 1 (add r2, r2, #1). Donc, sa valeur change à chaque itération, représentant l'indice de l'élément actuellement traité dans le tableau.

4. Quelle est la valeur contenue dans r2 à la fin de l'itération, c'est-`a-dire lorsque le contrôle est

à l'étiquette fintq ?

la boucle s'arrête lorsque la valeur de r2 atteint ou dépasse 5.

5. Supposons que l'algorithme soit écrit avec tant que i <= 4 au lieu de tant que i <> 5 ; le

tableau contient-t-il les même valeurs à la fin de l'itération ? Comment doit-on alors traduire

ce nouveau programme ?

Si l'algorithme est modifié pour utiliser la condition tant que i <= 4 au lieu de tant que i <> 5, cela signifie que la boucle s'exécutera pour les valeurs de i allant de 0 à 4 inclusivement.

Cela n'aura pas d'impact sur les valeurs finales du tableau à la fin de l'itération, car la boucle effectuera toujours le même nombre d'itérations et modifiera le tableau de la même manière.

Pour traduire cette modification en langage d'assemblage ARM, on peut simplement remplacer l'instruction de comparaison dans la boucle pour qu'elle se termine lorsque i est supérieur à 4. Voici comment on peut modifier votre programme :

```
6   main:
7       ldr r0, ptr_debutTAB
8
9       mov r3, #11                 @ val <- 11
10      mov r2, #0                  @ i <- 0
11  tq: cmp r2, #4                  @ i-4 ??
12      bgt fintq           @ Si i est supérieur à 4, sortie de la boucle
13          @ i <= 4 ou i < 5
14          ldr r0, ptr_debutTAB        @ r0 <- debutTAB
15          add r0, r0, r2, LSL #2  @ r0 <- r0 + r2*4 = debutTAB + i*4
16          str r3, [r0]            @ MEM[debutTAB+i*4] <- val
17          add r2, r2, #1          @ i <- i + 1
18          add r3, r3, #11         @ val <- val + 11
19      b   tq
20  fintq:  @ i > 4 ou i >= 5
21
22  fin:  BX LR
23
24  ptr_debutTAB : .word debutTAB
```

6. Supposons que le tableau soit maintenant un tableau de mots de 16 bits. Comment devez-vous modifier le programme ? Faire la modification et rendre le nouveau programme et les valeurs dans les registres.

C: > Users > SHAGHA~1 > AppData > Roaming > MobaXterm > slash > RemoteFiles > 460118_4_3 > ASM iteration.s

```
1   .data
2   debutTAB: .skip 5*2      @ Chaque élément du tableau est un mot de 16 bits (2 octets)
3   .text
4   .global main
5   main:
6       ldr r0, ptr_debutTAB
7
8       mov r3, #11             @ val <- 11
9       mov r2, #0              @ i <- 0
10  tq: cmp r2, #4             @ i-4 ??
11      bgt fintq               @ Si i est supérieur à 4, sortie de la boucle
12          @ i <= 4 ou i < 5
13          ldr r0, ptr_debutTAB        @ r0 <- debutTAB
14          add r0, r0, r2, LSL #1  @ r0 <- r0 + r2*2 = debutTAB + i*2 (décalage de 1 pour 2 octets)
15          strh r3, [r0]           @ MEM[debutTAB+i*2] <- val (utilisation de strh pour stocker un mot de 16 bits)
16          add r2, r2, #1          @ i <- i + 1
17          add r3, r3, #11         @ val <- val + 11
18      b tq
19  fintq:  @ i > 4 ou i >= 5
20  fin:  BX LR
21  ptr_debutTAB : .word debutTAB
```

Dans cette version modifiée :

- Nous avons remplacé .skip 5*4 par .skip 5*2 pour indiquer que chaque élément du tableau est maintenant un mot de 16 bits.
- L'instruction add r0, r0, r2, LSL #1 a remplacé add r0, r0, r2, LSL #2 pour tenir compte de la nouvelle taille des données (décalage de 1 pour des mots de 16 bits).
- L'instruction strh r3, [r0] a remplacé str r3, [r0] pour stocker un mot de 16 bits à l'emplacement mémoire approprié.

Target sim

Load

Break tq

Run

On répète step fintq – info reg

r0 commence avec la valeur de 74476 à chaque itération, augmentant de 2 jusqu'à la valeur de 74484, alors cela signifie que chaque élément du tableau est situé à une adresse mémoire qui est espacée de 2 octets(16bits).

```
(gdb) info reg
r0             0x122f4  74484
r1             0x1ffff8 2097144
r2             0x5      5
r3             0x42     66
r4             0x1      1
r5             0x1ffff8 2097144
r6             0x0      0
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0x200100 2097408
r11            0x0      0
r12            0x1fffe8 2097128
sp             0x1ffff8 0x1ffff8
lr             0x81a0   33184
pc             0x81cc   0x81cc <tq+4>
fps            0x0      0
cpsr           0x20000013       536870931
```

7. Même question pour un tableau d'octets.

```
.data
debutTAB: .skip 5          @ Chaque élément du tableau est maintenant un octet

.text
.global main
main:
    ldr r0, ptr_debutTAB

    mov r3, #11             @ val <- 11
    mov r2, #0              @ i <- 0
tq: cmp r2, #4             @ i-4 ??
    bgt fintq               @ Si i est supérieur à 4, sortie de la boucle
        @ i <= 4 ou i < 5
        ldr r0, ptr_debutTAB        @ r0 <- debutTAB
        add r0, r0, r2          @ r0 <- r0 + r2 = debutTAB + i (pas de décalage pour les octets)
        strb r3, [r0]           @ MEM[debutTAB+i] <- val (utilisation de strb pour stocker un octet)
        add r2, r2, #1          @ i <- i + 1
        add r3, r3, #11         @ val <- val + 11
    b tq
fintq:  @ i > 4 ou i >= 5

fin:  BX LR

ptr_debutTAB : .word debutTAB
```

r0 commence avec la valeur de 74476 à chaque itération, augmentant de 1 jusqu'à la valeur de 74480, alors cela signifie que chaque élément du tableau est situé à une adresse mémoire qui est espacée de 2 octets(16bits).

```
(gdb) info reg
r0             0x122f0  74480
r1             0x1ffff8 2097144
r2             0x5      5
r3             0x42     66
r4             0x1      1
r5             0x1ffff8 2097144
r6             0x0      0
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0x200100 2097408
r11            0x0      0
r12            0x1fffe8 2097128
sp             0x1ffff8 0x1ffff8
lr             0x81a0   33184
pc             0x81cc   0x81cc <tq+4>
fps            0x0      0
cpsr           0x20000013       536870931
```

5.3 Calcul de la suite de “Syracuse” :

```
  .text
  .global main
main:
    mov r1, #15             @ Initialiser x avec U0 = 15

tq: cmp r1, #1              @ Tant que x <> 1
    beq fintq               @ Si x == 1, sortir de la boucle

si: tst r1, #1              @ Tester si x est pair
    bne sinon               @ Si x n'est pas pair, aller à sinon

    lsr r1, r1, #1          @ decaler a droite d'une valeur
    b finsi                 @ Aller à la fin de la condition

sinon:
    add r1, r1, r1, LSL #1 @ Si x n'est pas pair, multiplier x par 2 et ajouter 1
    add r1, #1              @ Ajouter 1 à x

finsi:
    b tq                    @ Retourner à la boucle

fintq:
    @ Fin de la boucle lorsque x == 1

fin:
    BX LR                   @ Fin du programme
```

```
(gdb) break tq
Breakpoint 1 at 0x81c4: file syracuse.s, line 7.
(gdb) run
Starting program: /home/h/hajmohas/INF401/TP5/syracuse

Breakpoint 1, tq () at syracuse.s:7
7           beq fintq             @ Si x == 1, sortir de la boucle
(gdb) info registers $r1
r1             0xf      15
(gdb) step fintq

Breakpoint 1, tq () at syracuse.s:7
7           beq fintq             @ Si x == 1, sortir de la boucle
(gdb) info registers $r1
r1             0x2e     46
(gdb) step fintq

Breakpoint 1, tq () at syracuse.s:7
7           beq fintq             @ Si x == 1, sortir de la boucle
(gdb) info registers $r1
r1             0x17     23
(gdb) step fintq

Breakpoint 1, tq () at syracuse.s:7
7           beq fintq             @ Si x == 1, sortir de la boucle
(gdb) info registers $r1
r1             0x46     70
```

```
(gdb) step fintq

Breakpoint 1, tq () at syracuse.s:7
7           beq fintq             @ Si x == 1, sortir de la boucle
(gdb) info registers $r1
r1             0x23     35
(gdb) step fintq

Breakpoint 1, tq () at syracuse.s:7
7           beq fintq             @ Si x == 1, sortir de la boucle
(gdb) info registers $r1
r1             0x6a     106
(gdb) step fintq

Breakpoint 1, tq () at syracuse.s:7
7           beq fintq             @ Si x == 1, sortir de la boucle
(gdb) info registers $r1
r1             0x35     53
(gdb) step fintq

Breakpoint 1, tq () at syracuse.s:7
7           beq fintq             @ Si x == 1, sortir de la boucle
(gdb) info registers $r1
r1             0xa0     160
(gdb) step fintq

Breakpoint 1, tq () at syracuse.s:7
7           beq fintq             @ Si x == 1, sortir de la boucle
(gdb) info registers $r1
r1             0x50     80
```

```
(gdb) info registers $r1
r1             0x28     40
(gdb) step fintq

Breakpoint 1, tq () at syracuse.s:7
7           beq fintq              @ Si x == 1, sortir de la boucle
(gdb) info registers $r1
r1             0x14     20
(gdb) step fintq

Breakpoint 1, tq () at syracuse.s:7
7           beq fintq              @ Si x == 1, sortir de la boucle
(gdb) info registers $r1
r1             0xa      10
(gdb) step fintq

Breakpoint 1, tq () at syracuse.s:7
7           beq fintq              @ Si x == 1, sortir de la boucle
(gdb) info registers $r1
r1             0x5      5
(gdb) step fintq

Breakpoint 1, tq () at syracuse.s:7
7           beq fintq              @ Si x == 1, sortir de la boucle
(gdb) info registers $r1
r1             0x10     16
```

```
(gdb) step fintq

Breakpoint 1, tq () at syracuse.s:7
7           beq fintq              @ Si x == 1, sortir de la boucle
(gdb) info registers $r1
r1             0x8      8
(gdb) step fintq

Breakpoint 1, tq () at syracuse.s:7
7           beq fintq              @ Si x == 1, sortir de la boucle
(gdb) info registers $r1
r1             0x4      4
(gdb) step fintq

Breakpoint 1, tq () at syracuse.s:7
7           beq fintq              @ Si x == 1, sortir de la boucle
(gdb) info registers $r1
r1             0x2      2
(gdb) step fintq

Breakpoint 1, tq () at syracuse.s:7
7           beq fintq              @ Si x == 1, sortir de la boucle
(gdb) info registers $r1
r1             0x1      1
```