

Assembleur ARM – TD

Semestre 4

Table des matières

1	Opérations	4
2	TD1	5
2.1	5
2.1.1	5
2.1.2	5
2.2	5
2.2.1	5
2.2.2	6
2.2.3	Réaliser la division de r0 par r1 qui donne dans r2 le quotient et dans r3 le reste	7
3	TD2 – Manipulation de la mémoire	9
3.1	9
3.1.1	Avec un pointeur	9
3.2	10
3.3	Que fais le programme suivant ?	10

Opérations

```
1 | @R3 <- R0 + (R1-R2)
2 | SUB R3,R1,R2
3 | ADD R3,R0,R3
```

Listing 1.1 – $R3 \leftarrow R0 + (R1-R2)$

```
1 | @R0 <- R1 + R2 + R3
2 | ADD R3,R1,R2
3 | ADD R3,R1,R3
```


Listing 1.2 – $R0 \leftarrow R1 + R2 + R3$

```
1 | @R0 <- R1+(R2-4)
2 | SUB R0,R2,#4
3 | ADD R0,R1,R0
```

Listing 1.3 – $R0 \leftarrow R1+(R2-4)$

```
1 | @R0 <- R2 + R2*4
2 | MOV R0,#4
3 | MUL R0,R2,R0
4 | ADD R0,R2,R0
```

Listing 1.4 – $R0 \leftarrow R2 + R2*4$

 MUL R3,R2,#4 n'est pas possible

```
1 | @R0 <- -R1 sous 3 formes différentes
2 | RSB R0,R1,#0
3 |
4 | MOV R2,#0
5 | SUB R0,R2,R1
6 |
7 | MVN R0,R1
8 | ADD R0,R0,#1
```

Listing 1.5 – $R0 \leftarrow -R1$ sous 3 formes différentes

```
1 | @En un minimm d'instructions calculez r0 <- 10 * r1
2 | @(sans utiliser la multiplication)
3 | MOV r0,r1,LSL #3 @r0 = r1 * 2^3
4 | @r0 <- r0 + r1 * 2^1
5 | @r0 <- r1 * 2^3 + r1 * 2^1
6 | @r1 * (2^3 + 2^1)
7 | ADD r0,r0,r1,LSL #1
```

Listing 1.6 – $R0 \leftarrow 10 * R1$ sans utiliser la multiplication

TD1

2.1

2.1.1

ON considère l'algorithme suivant :

```
1 | si r0 > 0 alors
2 |   s1;
3 | sinon
4 |   s2;
5 | fin si;
```

Traduire cette forme algorithmique en assembleur.

```
1 | CMP r0,#0 @si r0 > 0 alors
2 | BLS sinon
3 |   <s1>
4 |   B finsi
5 | sinon:  @ sinon
6 |   <s2>
7 | finsi:
```

2.1.2

```
1 | tantque 0 > 0 faire
2 |   s;
3 | fin tantque;

1 | boucle: CMP r0,#0
2 |   BLE finboucle
3 |   <S>
4 |   B boucle
5 | finboucle:
```

2.2

2.2.1

```
1 | r0 <- 0;
2 | tantque R0 < N faire
3 |   r1 <- r1 + r0;
```

```
4 | r0 <- r0 + 1;
5 | fin tantque;

1 | .equ N,10
2 | MOV R0,#0 @s <- 0
3 | MOV R1,#0 @i <- 0
4 | tq: CMP R1,#N @tantque(i <= N)
5 | BHI ftq
6 | ADD R1,R0,R1 @s += i
7 | ADD R1,R1,#1 @ i++
8 | B tq
9 | ftq:
```

2.2.2

Écrire un programme qui calcule la multiplication de r0 par r1 et range le résultat dans r2 sans l'opération de multiplication

2.2.2.1 Nombres non signés

```
1 | MOV r3, #0 @ i <- 0
2 | MOV r2, #0 @ r2 <- 0
3 | tq: CMP r3,r1 @ tantque(i < r1)
4 | BHS ftq
5 | ADD r2,r0,r2 @r2 <- r0 + r2
6 | ADD r3, r3, #1 @i++
7 | B tq
8 | ftq:
```

2.2.2.2 Nombres signés

```
1 | si r1 > 0 alors
2 |   multiplication non signé;
3 | sinon
4 |   r1 <- 0 - r1;
5 |   r3 <- 0;
6 | fin si;
7 |
8 | si r3 = 0 alors
9 |   r1 <- 0 - r1;
10 | sinon
11 |   on sort;
```

```
1 | MOV r3, #1
2 | CMP r1, #0
3 | BGT boucle
4 | RSB r1, r1, #0
5 | MOV r3, #0
6 | boucle: MOV R2, #0
7 | tq: CMP R1, #0
8 | BEQ sortie1
9 | ADD r2, r0, r0
10 | SUB r1, r1, #1
11 | B tq
12 | sortie1: CMP R3, #0
13 | BNE sortie2
```

```

14 | RSB r1, r1, #0
15 | sortie2:

```

2.2.2.3 Autres solutions pour nombres signés

```

1 | si r1 < 0 alors
2 |   r1 <- -r1;
3 |   r0 <- -r0;
4 | fin si;
5 | i <- 0;
6 | tantque i < r1 faire
7 |   r2 <- r2 + r0;
8 |   i <- i + 1;
9 | fin tantque

```

```

1 | CMP r1, #0
2 | RSBLT r1, r1, #0
3 | MOV r2, #0
4 | MOV r3, #0
5 | tq: CMP r3, r1
6 |   BGE ftq
7 |   ADD r2, r2, r0
8 |   ADD r3, r3, #1
9 |   B tq
10 | ftq:

```

2.2.3 Réaliser la division de r0 par r1 qui donne dans r2 le quotient et dans r3 le reste

2.2.3.1 Non signé

$$r0 = r1 \times r2 + r3, 0 \leq r3 < r1$$

```

r2 <- 0;
tantque r3 >= r1 faire
  r1 <- r2 + 1;
  r3 <- r3 - r1;
fin tantque;

```

```

MOV r2,#0
MOV r3,r0
tq: CMP r3,r1
  BLT ftq
  ADD r2,r2,#1
  SUB r3,r3,r1
  B tq
ftq:

```

2.2.3.2 Signé

```

1 | si r0 > 0 alors
2 |   r4 <- 0;
3 | sinon
4 |   r4 <- 1;
5 |   r0 <- -r0;

```

```
6  fin si;
7
8  si r1 > 0 alors
9      r5 <- 0;
10 sinon
11     r5 <- 1;
12     r1 <- -r1;
13     r2 <- 0;
14     r3 <- 0;
15 fin si;
16 tantque r3 > r1 faire
17     r3 <- r3 - r1;
18     r2 <- r2 + 1;
19 fin tantque;
20
21 si r4 = -r3 alors
22     r3 = -r3;
23 fin si;
24 si r4 != r5 alors
25     r2 = -r2;
26 fin si;
```

```
1  CMP r0, #0
2  BLT sinon1
3  MOV r4,#0
4  RSB r0,#0
5  B fsi1
6  sinon1: MOV r4,#1
7  fsi1:
8      CMP r1, #0
9      BLT sinon2
10     MOV r5,#0
11     RSB r2,#0
12  sinon2:
13  @(boucle)
14     CMP r4,#1
15     BNE fsi
16     RSB r3,#0
17  fsi3:
18     CMP r4,r5
19     BEQ fsi4
20     RSB r2,#0
21  fsi4:
```


TD2 – Manipulation de la mémoire

3.1

3.1.0.3 Avec un itérateur

```

1 | define N 10
2 | t[10];
3 | t1 <- @t;
4 | r2 <- 0; --i
5 | r0 <- 0;
6 |
7 | tantque t2 < N faire
8 |     r3 <- MEM[t1+r2 << 2];
9 |     r2 <- r2+1;
10 |    r0 <- r0 + r3
11 | fin tantque;

```

```

1 | .eq N,10
2 | t: .fill N,4,0
3 | LDR r1,=t
4 | MOV R2,#0
5 | MOV R0,#0
6 |
7 | tq: CMP r1,N
8 |     BHS ftq
9 |     LDR r3,[r1,r2,LSL#2]
10 |    ADD r0,r0,r3
11 |    ADD r2,r2,#1
12 |    B tq
13 | ftq:

```

3.1.1 Avec un pointeur

```

1 | r1 <- @t;
2 | r2 <- @tfin;
3 | r0 <- 0;
4 |
5 | tantque r1 != r2 faire
6 |     r3 <- MEM32[r1];
7 |     r1 <- r1+4;
8 |     r0 <- r0 + r3;
9 | fin tantque

```

```

1 | LDR r1,=t
2 | LDR r2,=tfin @ <=> ADD r2,r1,#N*4
3 | MOV r0,#0
4 | tq: CMP r1,r2
5 |     BEQ ftq
6 |     LDR r3,[r1] @on pourrait faire
7 |     ADD r1,r1,#4 @LDR r3,[r1],#4
8 |     ADD r0,r0,r3
9 |     B tq
10 | ftq:

```

3.2

```
1 | t[N]
2 | r1 <- @t;
3 | r2 <- @tfin;
4 | r0 <- 1;
5 |
6 | tantque r1 != r2 faire
7 |     MEM32[r1] <- r0;
8 |     r1 <- r1 + 1;
9 |     r0 <- r0 + ;
10 | fin tantque;
```

```
1 | .eq N,10
2 | t: .fill N,1,0
3 | LDR r0,=t
4 | LDR r2,=tfin
5 | MOV r0,#1
6 | tq: CMP r1,r2
7 | BEQ ftq
8 | STRB r3,[r1],#1
9 | ADD r0,r0,#1
10 | B tq
11 | ftq:
```

3.3 Que fais le programme suivant ?

```
1 | LDR r0,=t
2 | MOV f1,#111
3 | STR r1,[r0]
4 | ADD r1,r0,#4
5 | MOV r2,#0
6 | tq: CMP r2,#4
7 | BEQ ftq
8 | LDR r3,[r0,r2,LSL#2]
9 | STR r3,[r1],#4
10 | ADD r2,r2,#1
11 | B tq
12 | ftq:
```

NST r0 r1 r2 r3 @t @t+4