Systèmes Embarqués 1 & 2

Classes T-2/I-2 // 2018-2019

a.11 – Jeu d'instructions du µP ARM

Solutions

Exercice 1

Transcrire en assembleur les algorithmes C ci-dessous

(a)

```
int i = 0;
int j = 0;
for (i=0; i<100; i++) j += i;
```

Solution: 1re solution

```
        ldr     r0, =0    // int i = 0
        ldr     r1, =0    // int j = 0
l1 :    cmp     r0, #100  // if i < 100
        bhs     l2
        add     r1, r0    // j += i
        add     r0, #1    // i++
        b       l1
l2 :
```

2e solution (plus efficace…)

```
        ldr     r0, =0    // int i = 0
        ldr     r1, =0    // int j = 0
        b       l2        // go to test
l1 :    add     r1, r0    // j += i
        add     r0, #1    // i++
l2 :    cmp     r0, #100  // if i < 100
        blo     l1
```

(b)

```
int i = 100;
unsigned int j = 2;
while (i>0) {
  j *= j; i--;
}
```

Solution: 1re solution

```
        ldr     r0, =100   // int i=100
        ldr     r1, =2     // unsigned j=2
l1 :    cmp     r0, #0     // if i > 0
        ble     l2
        mul     r1, r1, r1 // j *= j
        sub     r0, #1     // i--
        b       l1
l2 :
```

2e solution (plus efficace…)

```
        ldr     r0, =100   // int i=100
        ldr     r1, =2     // unsigned j=2
        cmp     r0, #0     // test...
        ble     l2
l1 :    mul     r1, r1, r1 // j *= j
        subs    r0, #1     // i--
        bne     l1         // if i > 0
l2 :
```

(c)

```
int i = 8;
unsigned int j = 0x7000;
do {
  j = j >> 1; i--;
} while (i > 0);
```

Solution:

```
        ldr     r0, =8          // int i=8
        ldr     r1, =0x7000     // unsigned j=0x7000
l1:     lsr     r1, #1          // j=j>>1
        subs    r0, #1          // i--
        bgt     l1              // if i > 0
```

(d)

```
int i;
int j = 0;
switch(i) {
  case 0:  j=11; break;
  case 1:  j=8;  break;
  case 2:  j=25; break;
  case 3:  j=99; break;
  case 4:  j=33; break;
  default: j=-1; break;
}
```

Solution:

```
        ldr     r0, =i
        ldr     r0, [r0]
        ldr     r1, =0
        cmp     r0, #4
        ldrhi   r0, =5
        ldr     r2, =lut
        ldr     pc, [r2, r0, lsl #2]

// switch-case table
lut: .word case_0, case_1, case_2, case_3, case_4, case_d

case_0: ldr     r1, =11
        b       l2
case_1: ldr     r1, =8
        b       l2
case_2: ldr     r1, =25
        b       l2
case_3: ldr     r1, =99
        b       l2
case_4: ldr     r1, =33
        b       l2
case_d: ldr     r1, =-1
        b       l2
l2:
```

(e)

```c
int i;
int j=0;
if (i>0) j = i % 16;
else if (i==0) j = 0xaa;
else j = -i / 16;
```

Solution:

```
        ldr     r0, =i
        ldr     r0, [r0]
        ldr     r1, =0
        cmp     r0, #0
        bgt     l3
        beq     l2

// else
l1:     rsb     r1, r0, #0
        asr     r1, #4
        b       l4

// i == 0
l2:     ldr     r1, =0xaa
        b       l4

// i > 0
l3:     and     r1, r0, #0xf
        b       l4

l4:
```

Exercice 2

Coder en assembleur ARM l'algorithme ci-dessous

```
short   toto;   // 2 octets
char    i;      // 1 octet

toto = 20;
i = 0;
while (i<7) { toto += toto; toto -= i; i++; }
```

Solution:

```
toto:   .short  0

        ldr     r0, =toto
        ldr     r1, =20
        strh    r1, [r0]        // toto = 20
        mov     r1, #0          // i = 0
        b       l2
l1:     ldrh    r2, [r0]
        add     r2, r2          // toto += toto
        sub     r2, r1          // toto -= i
        strh    r2, [r0]        // copy back
        add     r1, #1          // i++
l2:     cmp     r1, #7          // while (i<7)
        blt     l1
```

Exercice 3

Coder en assembleur ARM l'instruction ci-dessous

```
k := 0;
switch (i) {
  case 0 : k := -20;  break;
  case 2 : k := -2;   break;
  case 4 : k := 0;    break;
  case 5 : k := 5;    break;
  case 8 : k := 24;   break;
}
```

Solution:

```
        // r0 --> i
        ldr     r1,=0   // k = 0;
        cmp     r0,#0   // case i == 0
        beq     case0
        cmp     r0,#2   // case i == 2
        beq     case2
        cmp     r0,#4   // case i == 4
        beq     case4
        cmp     r0,#5   // case i == 5
        beq     case5
        cmp     r0,#8   // case i == 8
        beq     case8
        b       l2

case0:  ldr     r1, =-20
        b       l2
case2:  ldr     r1, =-2
        b       l2
case4:  ldr     r1, =0
        b       l2
case5:  ldr     r1, =5
        b       l2
case8:  ldr     r1, =24
        b       l2
l2:
```

Exercice 4

Coder en assembleur ARM l'algorithme calculant la parité verticale d'un certain nombre d'octets de données.

Code en C de la fonction

```
#define LENGTH 4
char chaine[LENGTH+1] = {0xC4,0xBF,0x1F,0xAD,0x00};
main() {
        char n = 0;
        while (n < LENGTH) {
                chaine[LENGTH] = chaine[LENGTH] ^ chaine[n] ;
                n++ ;
        }
}
```

Exemple

1	1	0	0	0	1	0	0
1	0	1	1	1	1	1	1
0	0	0	1	1	1	1	1
1	0	1	0	1	1	0	1
1	**1**	**0**	**0**	**1**	**0**	**0**	**1**

Solution:

```
LENGTH  = 4
chaine: .byte   0xc4, 0xbf, 0x1f, 0xad, 0x00

main :  ldr     r0, =0                  // char n = 0;
        ldr     r2, =0                  // chaine[LENGTH] = 0
        ldr     r1, =chaine             // charge adresse de la variable chaine dans r1
        b       l2
l1 :    ldrb    r3, [r1, r0]            // d1 = chaine[n]
        eor     r2, r3                  // chaine[LENGTH] ^= chaine[n]
        add     r0,#1                   // n++
l2 :    cmp     r0,#LENGTH              // while (n<LENGTH)
        blo     l1
        strb    r2, [r1, #LENGTH]       // copy back the result
```

Exercice 5

Pour le code assembleur ci-dessous

```
res:    .short    1
var1:   .byte    67

main:   ldr       r0, =var1
        ldrb      r4, [r0]
        lsl       r2, r4, #3
        lsl       r4, #1
        add       r4, r2
        ldr       r0, =res
        strh      r4, [r0]
```

(a) Quel sera la valeur stockée dans la variable « res » une fois que le code se sera déroulé?

Solution: res = 670

(b) Expliquer en quelques lignes la fonction de ce code.

Solution: res = var1 * 8 + var1 * 2 = var1 * 10

(c) Quelle est la valeur résultante dans le registre R2 (en décimal, hexa et binaire) ?

Solution: r2 = 536, 0x00000218, 0000'0010'0001'1000

(d) Quelle est la valeur résultante dans le registre R4 (en décimal, hexa et binaire) ?

Solution: r4 = 670, 0x0000029E, 0000'0010'1001'1110

(e) Pour quelle fourchette de valeur (valeur minimale et maximale) de la variable « var1 », cet algorithme est-il valable ?

Solution: min = 0, max = 255