



## Systèmes Embarqués 1 & 2

Classes T-2/I-2 // 2017-2018

### a.04 – Modes d'adressage

#### Solutions

#### Exercice 1

Transcrire en assembleur ARM les algorithmes ci-dessous (l'usage des registres est fortement recommandé)

(a)

```
int i = 25;
int j = -10;
int k = i;
int l = i - j;
```

#### Solution:

```
mov    r0, #25          // int i = 25;
mvn    r1, #10-1        // int j = -10;
mov    r2, r0            // int k = i;
sub    r3, r0, r1        // int l = i - j;
```

(b)

```
int i = 0xabefab98;
int j = 0;
for (int k = 31; k >= 0; k--)
    j += i & (1<<k) != 0 ? 1 : 0;
```

#### Solution:

```
ldr    r0, =0xabefab98 // int i = 0xabefab98;
mov    r1, #0          // int j = 0;
mov    r2, #31         // int k = 31;
mov    r3, #1
lp :   tst    r0, r3, lsl r2 // i & (1<<k) != 0
      addne  r1, #1        // j+=1 if (i & (1<<k) != 0)
      subs   r2, #1        // k--
      bpl    lp           // for (...)
```



(c)

```
int i = -58909;
unsigned int j = 5;
int k = i >> j;      // i / (2**j)
int l = i / 128;
```

**Solution:**

```
ldr    r0, =-58909    // int i = -58909;
mov     r1, #5         // unsigned int j = 5;
mov     r2, r0, asr r1 // int k = i >> j;
mov     r3, r0, asr #7 // int l = i / 128;
```

(d)

```
unsigned int i = 58909;
unsigned int j = 5;
unsigned int k = i / (1 << j);
unsigned int l = i / 128;
```

**Solution:**

```
ldr    r0, =58909     // unsigned int i = 58909;
mov     r1, #5         // unsigned int j = 5;
mov     r2, r0, lsr r1 // unsigned int k = i / (1 << j);
mov     r3, r0, lsr #7 // unsigned int l = i / 128;
```

(e)

```
int i = -879;
unsigned int j = 6;
int k = i * (1 << j);
int l = i * 64;
```

**Solution:**

```
ldr    r0, =-879      // int i = -879;
mov     r1, #6         // unsigned int j = 6;
mov     r2, r0, lsl r1 // int k = i * (1 << j);
mov     r3, r0, lsl #6 // int l = i * 64;
```

(f)

```
unsigned int i = 879;
unsigned int j = 6;
unsigned int k = i * (1 << j);
unsigned int l = i * 64;
```

**Solution:**

```
ldr    r0, =879       // unsigned int i = 879;
mov     r1, #6         // unsigned int j = 6;
mov     r2, r0, lsl r1 // unsigned int k = i * (1 << j);
mov     r3, r0, lsl #6 // unsigned int l = i * 64;
```

**Exercice 2**

Pour les codes assembleur effectuant des échanges de données avec la mémoire, la représentation de la mémoire (little-endian / 8-bits) et l'état des registres du processeur ci-dessous, indiquer le résultat des opérations

(a)

```
ldrb    r1, [r0]
ldrh    r2, [r0]
ldr     r3, [r0]

str     r1, [r0, #8]
strh    r2, [r0, #12]
strb    r3, [r0, #14]
```

**Solution:****Mémoire**  
(little-endian / 8 bits)

0xa0000100	0x25
0xa0000101	0x83
0xa0000102	0x75
0xa0000103	0x84
0xa0000104	0x87
0xa0000105	0x25
0xa0000106	0x73
0xa0000107	0xc2
0xa0000108	0x25
0xa0000109	0x00
0xa000010a	0x00
0xa000010b	0x00
0xa000010c	0x25
0xa000010d	0x83
0xa000010e	0x25
0xa000010f	
0xa0000110	
0xa0000111	
0xa0000112	
0xa0000113	
0xa0000114	
0xa0000115	
0xa0000116	
0xa0000117	

**Registres**

R0	0xa0000100
R1	0x00000025
R2	0x00008325
R3	0x84758325
R4	
R5	
R6	
R7	0x00000006
R8	0xa0000110
R9	0x00000001
R10	
R11	0xa0000100
R12	

(b)

```
ldrsb r4, [r0, #4]!
ldrsh r5, [r0], #2
ldr r6, [r0, -r7]!

ldr r10, [r0, r9, lsl #2]
str r10, [r8, r9, lsl #2]
strb r10, [r8, r9, lsl #1]
```

**Solution:**

### Mémoire (little-endian / 8 bits)

0xa0000100	0x25
0xa0000101	0x83
0xa0000102	0x75
0xa0000103	0x84
0xa0000104	0x87
0xa0000105	0x25
0xa0000106	0x73
0xa0000107	0xc2
0xa0000108	0x25
0xa0000109	0x00
0xa000010a	0x00
0xa000010b	0x00
0xa000010c	0x25
0xa000010d	0x83
0xa000010e	0x25
0xa000010f	
0xa0000110	
0xa0000111	
0xa0000112	0x87
0xa0000113	
0xa0000114	0x87
0xa0000115	0x25
0xa0000116	0x73
0xa0000117	0xc2

### Registres

R0	0xa0000100
R1	0x00000025
R2	0x00008325
R3	0x84758325
R4	0xffffffff87
R5	0x00002587
R6	0x84758325
R7	0x00000006
R8	0xa0000110
R9	0x00000001
R10	0xc2732587
R11	0xa0000100
R12	





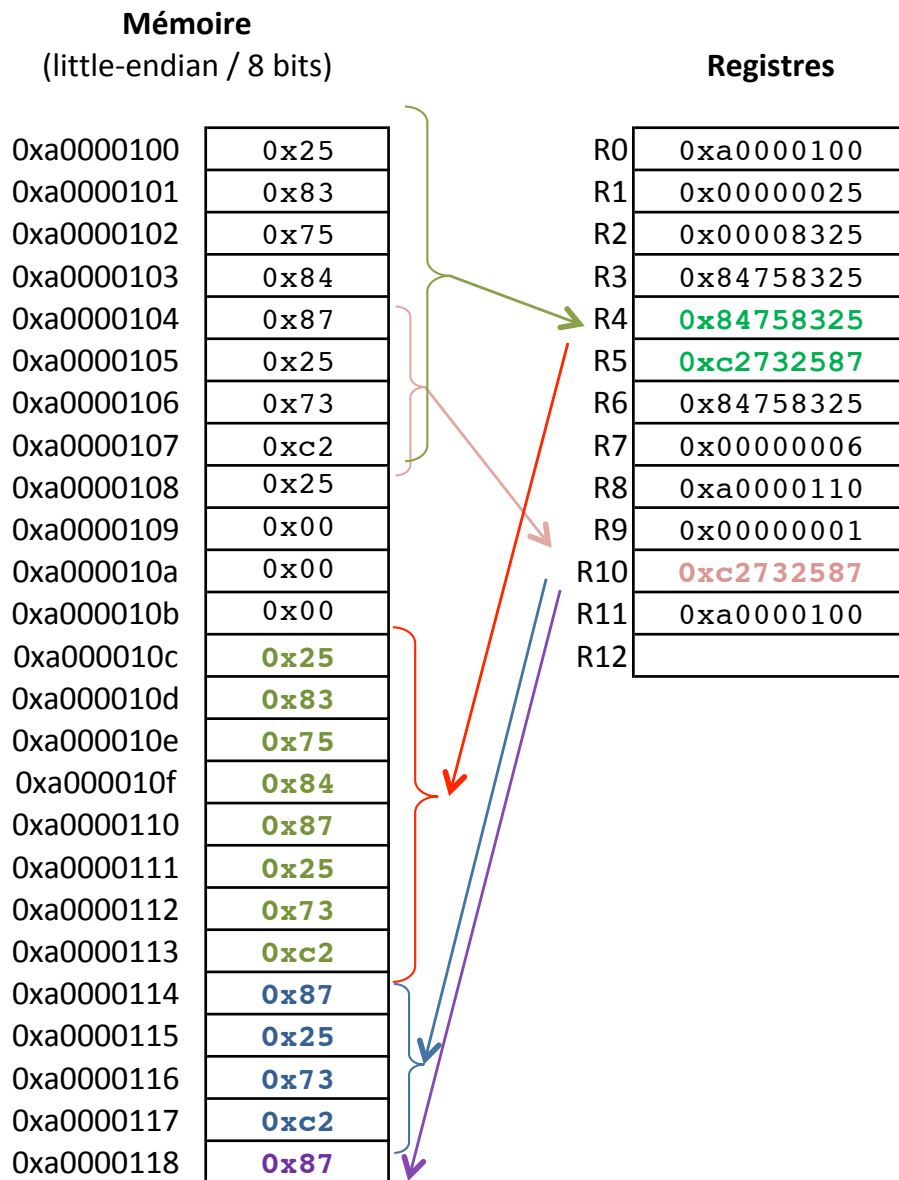
(c)

```

ldr    r10, [r0, r9, lsl #2]!
str    r10, [r8, r9, lsl #2]!
strb   r10, [r8, r9, lsl #2]!

ldrd   r4, [r11, #0]
strd   r4, [r0, #8]

```

**Solution:**

**Exercice 3**

Transcrire en assembleur le code C ci-dessous effectuant des échanges de données avec la mémoire

(a)

```

struct S {long a ; long b; long c; long d;}
struct S s1;
struct S s2[10];
s1.a = 1; s1.b = 2; s1.c = 3; s1.d = 4;
for (int i=0; i<10; i++) {
    s2[i].a = 10; s2[i].b = 10; s2[i].c = 10; s2[i].d = 10;
}

```

**Solution:**

```

s1 :    .space 4*4
s2 :    .space 4*4*10

        ldr    r0, =s1
        ldr    r1, =1
        str    r1, [r0, #0x0]
        ldr    r1, =2
        str    r1, [r0, #0x4]
        ldr    r1, =3
        str    r1, [r0, #0x8]
        ldr    r1, =4
        str    r1, [r0, #0xc]

        mov    r2, #10
        mov    r3, #10*4
        ldr    r0, =s2
1 :      subs   r3, #1
        str    r2, [r0, r3, lsl #2]
        bne    1b

```

(b)

```

long l[10]; long * lp = &l[0];
for (int j=0; j<10; j++) *lp++ = j;

```

**Solution:**

```

l :      .space 10*4
ls:
        ldr    r0, =ls
        mov    r2, #10
1 :      subs   r2, #1
        str    r2, [r0, #-4]!
        bne    1b

```



(c)

```
char src[10+1] = "0123456789";
char dst[10+1];
char* s = src;
char* d = dst;
while(*d++ = *s++);
```

**Solution:**

```
src : .asciz "0123456789"
dst : .space 11

ldr    r0, =src
ldr    r1, =dst
1 :    ldrb  r2, [r0], #1
        strb  r2, [r1], #1
        cmp  r2, #0
        bne  1b
```

(d)

```
char src[10+1] = "0123456789";
char dst[10+1];
register char* s = &src[11];
register char* d = &dst[11];
register int i;
for (i=11; i>0; i--)
    *--d = *--s;
```

**Solution:**

```
src : .asciz "0123456789"
dst : .space 11

ldr    r0, =src+11
ldr    r1, =dst+11
ldr    r2, =11
1 :    ldrb  r3, [r0, #-1]!
        strb  r3, [r1, #-1]!
        subs r2, #1
        bne  1b
```



(e)

```
unsigned char array[20];
unsigned long sum = 0;
for (int i=0; i<20; i++)
    sum += array[i];
```

**Solution:**

```
array : .byte 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
        .byte 10,11,12,13,14,15,16,17,18,19
sum :   .long 0

        ldr    r0, =array
        ldr    r1, =20
        mov    r2, #0
1 :      ldrb   r3, [r0], #1
        add    r2, r3
        subs   r1, #1
        bne    1b
        ldr    r0, =sum
        str    r2, [r0]
```



**Exercice 4**

Echange de données avec la mémoire

- (a) Sauver les registres R0, R3 et R6 à R10 sur une pile « full descending ». La pile est adressée avec le registre R12, lequel contient l'adresse 0xa0008000. Indiquer l'état de la pile et le contenu des registres du processeur qui ont été modifiés. Donner le code assembleur.

**Solution:**

```
stmfd    r12!, {r0,r3,r6-r10}

        r12 = 0xa0007fe4
[0xa0007ffc] = r10
[0xa0007ff8] = r9
[0xa0007ff4] = r8
[0xa0007ff0] = r7
[0xa0007fec] = r6
[0xa0007fe8] = r3
[0xa0007fe4] = r0
```

- (b) Restituer d'une pile « full descending » les registres R1, R3, R5 et R7 à R10. La pile est adressée avec le registre R12, lequel contient l'adresse 0xa0007f00. Indiquer l'état de la pile et le contenu des registres du processeur qui ont été modifiés. Donner le code assembleur.

**Solution:**

```
ldmfd    r12!, {r1,r3,r5,r7-r10}

        r12 = 0xa0007f1c
r10 = [0xa0007f18]
r9  = [0xa0007f14]
r8  = [0xa0007f10]
r7  = [0xa0007f0c]
r5  = [0xa0007f08]
r3  = [0xa0007f04]
r1  = [0xa0007f00]
```