Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Embedded Systeme 1 & 2

Klassen T-2/I-2 // 2018-2019

a.12 - Adressierungsmodi

Übung Nr. 1

Transkribieren Sie die unten stehenden Algorithmen in Assembler:

a)

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

c)

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

d)

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

e)

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

f)

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

Übung Nr. 2

Geben Sie für die Assembler-Codes die Speicherdarstellung (little-endian / 8 Bit) und den Zustand der nachstehenden Prozessorregister das Ergebnis der Operationen an:

a)

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

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

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]
```

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]
```

Speicher (little-endian / 8 Bit)

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

Register

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

Übung Nr. 3

Transkribieren Sie den untenstehenden-Code in Assembler:

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;
}
```

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

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

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;
```

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

Übung Nr. 4

Austausch der Daten mit dem Speicher.

a) Sichern Sie die Register `R0`, `R3` und `R6` bis `R10` auf einem Stapel (stack) "full descending". Der Stapel (stack) wird mit Register `R12` adressiert, das die Adresse `0xa0008000` enthält. Geben Sie den Zustand des Stapels (stack) und den Inhalt der Prozessorregister an, die geändert wurden. Liefern Sie den Assembler-Code.

b) Stellen Sie von einem Stapel (stack) "full descending" die Register `R1`, `R3`, `R5` und `R7` bis `R10` wieder her. Der Stapel (stack) wird mit Register `R12` adressiert, das die Adresse `0xa0007f00` enthält. Geben Sie den Zustand des Stapels (stack) und den Inhalt der Prozessorregister an, die geändert wurden. Liefern Sie den Assembler-Code.