

Lösningsförslag till valda uppgifter, Arbetsbok UG5

Uppgift 2.1

		LDR	R1,=c	
20000000:	4d06	ldr	r1, [pc, #24]	; (2000001c <c+0x2>)
		LDRB	R0,='A'	
20000002:	2041	movs	r0, #65	; 0x41
		STRB	R0,[R1]	
20000004:	7028	strb	r0, [r1, #0]	
		LDR	R1,=s	
20000006:	4d06	ldr	r1, [pc, #24]	; (20000020 <c+0x6>)
		LDRH	R0,=1000	
20000008:	4806	ldr	r0, [pc, #24]	; (20000024 <c+0xa>)
		STRH	R0,[R1]	
2000000a:	8028	strh	r0, [r1, #0]	
		LDR	R1,=i	
2000000c:	4d06	ldr	r1, [pc, #24]	; (20000028 <c+0xe>)
		LDR	R0,=100000	
2000000e:	4807	ldr	r0, [pc, #28]	; (2000002c <c+0x12>)
		STR	R0,[R1]	
20000010:	6028	str	r0, [r1, #0]	
20000012:	46c0	nop		; (mov r8, r8)
20000014:	<i>:			
20000014:		.word	0x00000000	
20000018:	<s>:			
2000001a:	<c>:			
2000001a:		.short	0x0000	
2000001c:	2000001a	.word	0x2000001a	
20000020:	20000018	.word	0x20000018	
20000024:	000003e8	.word	0x000003e8	
20000028:	20000014	.word	0x20000014	
2000002c:	000186a0	.word	0x000186a0	

Utför sedan instruktionssekvensen i simulatören och kontrollera, i minnet, att tilldelningarna utförts korrekt. Fyll i värdena i följande tabell

20000014	<i>:	0x000186A0
20000018	<s>:	0x03E8
2000001a	<c>:	0x41

...

Uppgift 2.3

```
LDR    R1,=s
LDR    R2,=c
LDRH   R0,[R1]
STRB   R0,[R2]
```

- Skapa ett assemblerprogram och spara som mom3.asm.
- Assemblera, rätta eventuella fel och ladda till simulator.
- Använd listfilen och lokalisera adresser för variablerna, undersök minnesinnehållet med simulatören, fyll i dessa:

Adress	Variabel	Värde
20000008	<s>:	0
2000000a	<c>:	0

- Placera initialvärdet 0x3E0 i minnespositionen för variabeln s.
- Utför instruktionssekvensen, vilket värde har c efter sekvensen?

0xE0

Utgå nu i stället från följande deklarationer:

```
int i;
short s;
```

Koda nu tilldelningen:

```
s = i;
```

- Assemblera, rätta eventuella fel och ladda till simulator.
- Placera initialvärdet 0x3E000 i minnespositionen för variabeln i.
- Utför instruktionssekvensen, observera värdet för s efter sekvensen.

0xE000

...

Uppgift 2.4

Adress	Variabel	Värde
2000000a	<s>:	0
2000000c	<c>:	0x80

- Utför instruktionssekvensen, vilket värde har s efter sekvensen?

0xFF80

Utgå nu från följande deklarationer:

```
unsigned short s;
unsigned char c;
```

Koda nu tilldelningen:

s = c;

- Assemblera, rätta eventuella fel och ladda till simulator,
- Placera på nytt initialvärdet 0x80 i minnespositionen för variabeln c.
- Utför instruktionssekvensen, vilket värde har s efter sekvensen?

0x0080

Uppgift 2.5

Instruktion	Beskriv med ord
SXTH Rd, Rm	Teckenutvidga 16-bitars tal med tecken i Rm till 32-bitars tal med tecken i Rd.
SXTB Rd, Rm	Teckenutvidga 8-bitars tal med tecken i Rm till 32-bitars tal med tecken i Rd.
UXTH Rd, Rm	Teckenutvidga 16-bitars tal utan tecken i Rm till 32-bitars tal med tecken i Rd.
UXTB Rd, Rm	Teckenutvidga 8-bitars tal utan tecken i Rm till 32-bitars tal med tecken i Rd.

Uppgift 2.6

1

Vi tolkar nu i stället värdena som läses från omkopplarna som tal *med* tecken. Modifiera programmet för detta och kontrollera funktionen. Prova med att ställa in olika värden på omkopplarna. Vilka är de enda värden man kan få på displayen för bitar D8-D15?

0 och FF

Uppgift 2.7

Inställda värden (binärt)		Resultat C		
A	B	EOR	AND	ORR
1111 0000	1010 0101	0101 0101	1010 0000	1111 0101
1010 1010	1010 1010	0000 0000	1010 1010	1010 1010
1100 0011	0000 0000	1100 0011	0000 0000	1100 0011

Inställda värden (binärt)		Resultat C	
A	B	BIC	MVN
1111 0000	1010 0101	0101 0000	0101 1010
1010 1010	1010 1010	0000 0000	0101 0101
1100 0011	0000 0000	1100 0011	1111 1111

Uppgift 2.9

@ stack.asm			PC	R1	LR	SP	mem(SP)
start:	BL	sub	20000000		00000000	2001C000	
	B	start	20000004		20000013	2001C000	
sub:	PUSH	{LR}	20000006		20000005	2001C000	
	MOV	R0, #0x10	20000008			2001BFFC	20000005
	PUSH	{R0}	2000000A				
	BL	sub2	2000000C			2001BFF8	00000010
	POP	{R0}	20000010			2001BFF8	00000010
	POP	{PC}	20000012		20000011	2001BFFC	20000005
sub2:	PUSH	{R7}	20000014		20000011	2001BFF8	00000010
	MOV	R7, SP	20000016			2001BFF4	00000000
	LDR	R1, [R7, #4]	20000018	00000000			
	POP	{R7}	2000001A	00000010			
	BX	LR	2000001C		20000011	2001BFF8	00000010

Uppgift 2.10

@ compare.asm			
start:	LDR	R0, =0x55555555	@ initiera port D som utport
	LDR	R1, =0x40020C00	
	STR	R0, [R1]	
	LDR	R5, =0x40020C14	@ adressen till port D:s ut-dataregister till R5
	LDR	R6, =0x40021010	@ adressen till port E:s in-dataregister till R6
main:	LDRB	R0, [R6]	@ läs operand 1 till R0
	UXTB (SXTB)		
	LDRB	R1, [R6, #1]	@ läs operand 2 till R1
	UXTB (SXTB)		
	CMP	R0, R1	@ op1 - op2 -> APSR
	BHI (BGT)	main_1	
	MOV	R0, #0	
	B	main_2	
main_1:	MOV	R0, #0xFF	
main_2:	STRB	R0, [R5]	
	B	main	

Kontrollera nu programmets funktion genom att undersöka resultatet för följande inställda värden. Markera med ett 'x' i de rader där jämförelsen utfaller med resultatet sant.

Inport 1		Inport 2		Utan tecken	Med tecken
Binärt	Dec	Binärt	Dec		
00000001	1	00000000	0	x	x
00000000	0	00000001	1		
00000000	0	11111111	-1		x
11111111	-1	00000000	0	x	

Uppgift 2.11

Siffr	Sju-segmentskod		
	Binär kod	Binär form	Hexadecimal form
0	0000	0011 1111	3F
1	0001	0000 0110	06
2	0010	0101 1011	5B
3	0011	0100 1111	4F
4	0100	0110 0110	66
5	0101	0110 1101	6D
6	0110	0111 1101	7D
7	0111	0000 0111	07
8	1000	0111 1111	7F
9	1001	0110 0111	67
A	1010	0111 0111	77
B	1011	0111 1100	7C
C	1100	0011 1001	39
D	1101	0101 1110	5E
E	1110	0111 1001	79
F	1111	0111 0001	71

Ge en C-deklaration `SegCodes` av ett initierat fält som rymmer samtliga segmentskoder:

```
char SegCodes[]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,
                  0x7F,0x67,0x77,0x7C,0x39,0x 5E,0x79,0x71};
```

översätt deklarationen till assemblerkod:

```
SegCodes: .BYTE 0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07
           .BYTE 0x7F,0x67,0x77,0x7C,0x39,0x5E,0x79,0x71
```

Uppgift 2.12

```
start:  MOV    R0,#0           @ i = 0
        LDR    R1,=ca         @ R1 = &ca[0]
for:    CMP    R0,#10         @ i < sizeof(ca) ?
        BGE    forend
        STRB   R0,[R1,R0]     @ ca[i] = i
        ADD    R0,R0,#1       @ i = i+1
        B     for
forend: MOV    R2,#0xFF
        MOV    R0,#5
        STRB   R2,[R1,R0]     @ ca[5] = 0xFF;
ca:     .SPACE 10
```

Uppgift 2.13

copyvec:	PUSH	{R4,R5}	@
	MOV	R3,#0	@ i = 0
copyvec1:	CMP	R3,R2	@ i < size ?
	BGE	copyvec2	
	LSL	R5,R3,#1	@ R5 = i*2
	LDRSH	R4,[R1,R5]	@ R4 = src[i]
	LSL	R5,R3,#2	@ R5 = i*4
	STR	R4,[R0,R5]	@ dest[i] = R4
	ADD	R3,R3,#1	@ i = i+1;
	B	copyvec1	@
copyvec2:	POP	{R4,R5}	
	BX	LR	

Uppgift 2.14

8

- Utgå från kommentarerna och visa en kodsekvens som utför tilldelningen:

	LDR	R0,i	@ R0 = i
	LSL	R1,R0,#1	@ R1 = i*2
	ADD	R0,R0,R1	@ R0 = i*(2+1)
	LDR	R1,j	@ R1 = j
	ADD	R0,R0,R1	@ R0 = (i*3)+j
	LDR	R1,m3	@ R1 = m3[0]
	LDRB	R0,[R0,R1]	@ R0 = m3[i][j]
	LDR	R1,ch	@ R1 = ch
	STRB	R0,[R1]	@ ch = R0
	.ALIGN		
i:	.SPACE	4	
j:	.SPACE	4	
m3:	.BYTE	1,2,3,4,5,6,7,8,9	
ch:	.SPACE	1	

- Redigera en källtextfil, readmatrix.asm, assemblera och rätta eventuella syntaxfel.
- Öppna listfilen readmatrix.lst, lokalisera adresser för variablerna i och j och ch.

variabel	adress
i	0x20000014
j	0x20000018
ch	0x20000025