Syften:

- Visa metoder för översättning av C-funktion till assemblerspråk.
- Visa hur översatta funktioner kan testas.
- Vägleda förberedelser inför laboration 1
- Sammanfatta relevanta typuppgifter av tentamenskaraktär.

Övningen består av en längre sammansatt uppgift. Tillsammans exemplifierar deluppgifterna de typexempel som ges vid tentamen. Samtidigt ges förberedelser inför laboration 1.

Uppgift:
Följande deklarationer är givna i programspråket 'C'.

```
unsigned char v[6];
int i,j;
short k;
```

a) Visa dessa deklarationer i assemblerspråk.

Lösning: Reservera plats för variablerna med assemblerdirektiv. Påpeka speciellt att variablernas värden initieras till 0 av assemblatorn.

```
v: .space  6
i: .space  4
j: .space  4
k: .space  2
```

b) Antag att vi i stället gör tilldelningar enligt följande, i samband med deklarationerna:

```
unsigned char v[]={1,2,3,4,5,6};
int i=3,j=53;
short k=-15;
```

Visa dessa deklarationer i assemblerspråk:

Lösning: *Reservera plats för variablerna och ge initialvärden med assemblerdirektiv*.

```
v: .byte   1,2,3,4,5,6
i: .word   3
j: .word   53
k: .short  -15
```

c) Vi har nu följande uttryck:

```
v[i]+j+k;
```

Visa assemblerkod för evaluering av uttryckets värde till register `R0`.

Lösning: Anm: lös på tavlan, lösningen finns också färdig för att testas i ETERM i nästa deluppgift. Man kan också, som alternativ låta någon student skriva in koden under demonstrationen, för att sedan testas.

```
LDR  R0,=v      @R0<-&v[0]
LDR  R1,=i      @R1<-&i
LDR  R1,[R1]    @R1<-i
ADD  R0,R0,R1   @R0<-&v[0]+i
LDRB R0,[R0]    @R0<-v[i]

LDR  R1,=j      @R1<-&j
LDR  R1,[R1]    @R1<-j
ADD  R0,R0,R1   @R0<-v[i]+j

LDR  R1,=k      @R1<-&k
LDRH R1,[R1]    @R1<-(unsigned short) k
SXTH R1,R1      @R1<-(short) k
ADD  R0,R0,R1   @R1<-v[i]+j+k
```

d) Visa att din lösning är korrekt genom att testa den med simulator.

Lösning: Börja med att evaluera uttrycket för att ge rätt svar:

```
4+53-15= 42 !
```

Utför sedan sekvensen i ETERM.

e) Vi har C-funktionen:

```
int  scalar(int dim, unsigned char vector[])
{
   int result = 0;
   while(dim>0)
   {
      result += vector[dim-1];
      dim = dim - 1;
   }
   return result;
}
```

översätt funktionen till assemblerspråk.

Lösning:

```
@  Registers:
@  R0 = param 'dim'
@  R1 = param &vector[0]
@  R2 = local 'result'
scalar:
   MOV  R2,#0     @result = 0;
   B    scalar_1  @while(..)
scalar_2:
   MOV  R3,R0     @R3=dim
   SUB  R3,R3,#1  @R3=dim-1
   ADD  R3,R3,R1  @R3=&(vector[dim-1])
   LDRB R3,[R3]   @R3=vector[dim-1]
   ADD  R2,R2,R3  @result += vector[dim-1]
   SUB  R0,R0,#1  @ dim = dim-1

scalar_1
   CMP  R0,#0     @(dim>0)?
   BGT  scalar_2

   MOV  R0,R2     @ return result;
   BX   LR
```

f) Visa, med ett enkelt testprogram, att funktionen översatts korrekt.

Lösning: Testprogrammet utformas enligt följande funktionsanrop

```
unsigned char v[]={1,2,3,4,5,6};
int i;
...
int i = scalar( sizeof (v) , v );
```

som översatt till assemblerspråk blir:

```
start:
   MOV  R0,#6
   LDR  R1,=v
   BL   scalar
   LDR  R1,=i
   STR  R0,[R1]

   .align
v: .byte  1,2,3,4,5,6
   .align
i: .space 4
```

programmet testas i ETERM funktionen ska ge returvärde 21.