EXEMPEL:
Två tidigare tentamensuppgifter har här använts för att illustrera hur en kodbeskrivning kan göras parallellt med koden.

Uppgift 3 (6p)

Funktionen:
`void int_concat (int *i1, const int * i2, unsigned int n, unsigned int pos);`
konkatenerar (lägger till) ett fält med `n` heltal från `i2` till postition `pos` i fältet `i1`.
Implementera funktionen `int_concat` med användning av pekare, du får inte använda indexering. Din lösning får heller inte använda någon standardfunktion.

Uppgift 3:

```
void int_concat (int *i1, const int *i2, unsigned int n, unsigned int pos)
{
  int *insert = i1 + pos;

  while ( n-- )
  {
    *insert++ = *i2++;
  }
}
```

Exempel på hur lösningen dessutom kan beskrivas med text:
Funktion int_concat lägger till värdena från ett fält med n heltal (i2) till ett annat fält med heltal (i1). Dessa n värden ska läggas till från och med positionen 'pos' i i1. Lösningen får bara använda pekare. Så idén är att ha en ny pekare för att peka på var vi i i1 ska lägga till de olika värdena från i2. Denna pekare – 'insert' - ställs först till startpositionen i i1 (i1 + pos). Därefter lägger vi till vart och ett av n-elementen i i2. Så vi skapar en loop ('while') med n iterationer där vi i varje iteration kopierar innehållet i i2 (* i2) till i1 vid rätt ('insert') position. I varje iteration uppdateras pekarna källa (i2) och destination ('insert') för att peka på nästa element/position
.

Description of solution:
Function int_concat adds the values from an array of n integers (i2) to an original array of integers (i1). These n values should be added starting at position pos of i1. The solution should use just pointers. So the idea is to have a new pointer to point to where in i1 we should add the different values of i2. This pointer – insert – is first set to the initial position in i1 (i1+pos). Then we need to add each of the n elements of i2. So we create a loop (while) with n iterations where in each iteration we copy the contents of i2 (*i2) into i1 at the correct (insert) position. In each iteration we need to advance the pointers from the origin (i2) and destination (insert).

Uppgift 3 (6p)

Följande deklarationer är givna:

```
int go( int a, int b);
int next( void );
```

Visa hur funktionen g kan kodas i ARMv6 assemblerspråk.

```
int g( int x, int y)
{
   int a = next();
   if( a-1 )
      return go(y,x);
   return go(x,y);
}
```

Uppgift 3:

```
@ Registers:
@   R4 spill x
@   R5 spill y
g: PUSH   {LR,R4,R5}
   MOV    R4,R0
   MOV    R5,R1
   BL     next
   SUB    R0,R0,#1
   CMP    R0,#0
   BNE    .g1
   MOV    R0,R4
   MOV    R1,R5
   B      .g2
.g1:
   MOV    R0,R5
   MOV    R1,R4
.g2:
   BL     go
   POP    {PC,R4,R5}
```

Exempel på hur lösningen dessutom kan beskrivas med text:

I den här frågan måste vi implementera funktionen ’g’ som anropar funktion ’next’ och beroende på dess returvärde utför antingen ’go (x, y)’ eller ’go (y, x)’. Eftersom ’g’ innehåller dessa funktionsanrop måste returadressen (LR) sparas på stacken. R4 och R5 sparas också eftersom dessa register används som spillregister. Utöver detta måste även inparametrarna i R0 och R1 kopieras (till R4 och R5). Med BL anropas funktionen ’next’ utan parametrar vi behöver alltså inte bry oss om att ställa R0..R3 före anropet. Returvärdet från ’next’ efter anropet är i R0 och därför används R0 för för att implementera ’if’. Om a-1 inte är noll (CMP & BNE) ska parametrar för ’go’ laddas i ordningen (y, x) (.g1) annars ska parametrarna laddas i ordning ’go (x, y)’. Båda alternativen är att anropa funktionen ’go’ men med olika parametrar. Så för fallet ’if(a-1)’ är sant, ska den första parametern (R0) vara y (nu lagrad i R5) och den andra parametern (R1) ska vara x (nu lagrad i R4). Annars bör R0 få värdet R4 och R1 värdet R5. Efter inställning av R0 och R1 anropas ’go’ med BL och funktionen g återgår. R4 och R5 återställs till sina ursprungliga värden och LR poppas in i PC så att körningen fortsätter vid instruktionen direkt efter anropet till g-funktionen.

Description of the solution:

In this question we need to implement the function g which calls function next and depending on its return value it either calls go(x,y) or go(y,x). Since there are function call

within the code of g then the first thing is to save the return address (LR) on the stack. R4 and R5 are also stored because those registers are used within the code of g and their contents need to be saved as they could have been used in main. After this the input parameters R0 and R1 need to be saved (to R4 and R5) since we do not know the code of next and go. BL is used to call the function next and given it has no parameters we do not need to care about setting R0..R3 before the call. The return value from next after the call is in R0 and thus we need to use R0 for the operations to implement the if. If a-1 is not zero (CMP & BNE) then the execution should go to go(y,x) (.g1) (body of the if) otherwise the execution should go to go(x,y). Both options are to call function go but with different parameters so I will call go only once but set the input parameters accordingly. So for the case where the if is satisfied, then the first parameter (R0) should be y (now stored in R5) and the second parameter (R1) should be x (now stored in R4). Otherwise R0 should get the value of R4 and R1 the value of R5. After setting R0 and R1, go is called with BL and the function g return. Before returning R4 and R5 get their original values back and LR is popped into PC so that the execution continues at the instruction right after the call to the g function.