Laboration 2 i Datorteknik-Assemblerprogrammering II

Målet med laborationen är att få begrepp om

- Subrutiner. in/utparametrar. Lokala variabler
- Maska in bitar till ett register
- Konstruktion av subrutiner
- Strukturering av en större kod med hjälp av subrutiner

Uppgift l: Bekanta sig med subrutiner

Syfte

Att bekanta sig med två färdigskrivna subrutiner. Få förståelse för begrepp som lokala variabler (register) och inparameter och utparameter.

- Ladda hem filen Lab2_1.zip och packa upp på lämpligt ställe på ditt konto. Där i ligger färdiga projektfiler och källkod för uppgift l. Öppna projektet Lab2_1 (IAR IDE Workspace (.eww))

Teori: Subrutiner

Givet från början är en subrutin kallad `Delay_ms`. En subrutin är väldigt lik en funktion i C eller en metod i JAVA. Fördelen med en subrutin är ju att man kan anropa den från flera ställen. En subrutin kan även ta inparametrar. Det lättaste sättet att utbyta information mellan anropande kod och subrutinen är att använda ett register.

Anrop och return

En subrutin anropas genom instruktionen `BL` som står för Branch with Link. Då en subrutin anropas måste programräknaren sparas undan så man kan hitta tillbaks till anropet då subrutinen är utförd. Det är just detta som "with Link" står för. Programräknaren sparas i länkregistret `R14` (`LR`) då man utför en Branch with Link.

```
BL    Delay_ms
```

För att återgå från subrutinen till anropande koden måste man lägga tillbaka den sparade programräknaren från länkregistret. Detta görs helt enkelt genom att kopiera `LR` till `PC` då man vill återgå. Detta brukar i högnivåspråk kallas att göra "return".

```
MOV   PC,LR
```

Lokala variabler och stacken

Om man drar sig till minnes hur en funktion eller en metod fungerar i C eller JAVA finns något som heter *lokala variabler*. Dessa är ju som ordet säger lokala i subrutinen. Fördelen är att anropande kodavsnitt kan använda samma register som subrutinen.

För att få en liknande funktion i assembler väljer man att spara undan innehållet i de register som används i subrutinen för att kunna återställa dom senare. På ARM-processorn kan man på ett mycket smidigt sätt kopiera flera register samtidigt.

Var sparas då registren? Jo på ett speciellt ställe i minnet som kallas stacken. Stacken är helt enkelt ett minnesutrymme som vilket som helst. För att hålla reda på var i minnesutrymmet (som kallas stack) man befinner sig har man ett speciellt register. Det kallas ofta stackpekaren och är på ARM ofta R13. Den initieras i början av koden till något lämpligt utrymme, se nedan:

```
        SECTION .intvec : CODE (2)
        THUMB
        DATA
__vector_table
        DCD 0x20008000          ;initiering av Stackpekare
        DCD __iar_program_start
```

Då man skriver till stacken kan man göra det enligt nedanstående exempel.

```
STMFD       SP!,{R0,R1}
```

För att sen läsa tillbaka är det viktigt att det görs på ett motsvarande sätt så att stackpekaren blir återställd.

```
LDMFD       SP!,{R0,R1}
```

Parameterpassning

Det finns flera sätt att passa information mellan subrutin och anropande kod. Stacken är en väldigt generell metod som är lämplig för stora mängder data. I enkla, som detta fall, kan register användas. Viktigt är då att man inte sparar undan värdet på det. Skulle man det kommer ju ingen förändring av registret ske, och det är ju det man vill i detta fallet.

Subrutinen `Delay_ms`

Subrutinen `Delay_ms` i detta fallet använder register `R7` för att passa information till subrutinen. `R7` innehåller hur lång tid (i ms) som subrutinen ska ta. Om man jämför med högnivåspåk är det enda subrutinen en dubbel for-loop. Den inre loopen tar precis l ms och den yttre räknar upp till det tal som är inparameter. Subrutinen nyttjar internt `R0` och `Rl` så dessa sparas undan och återställs i slutet. Man passar även på att spara länkregister på stacken för att senare kunna återställa detta. Detta är för att kunna säkerställa eventuella nästlade anrop. Alla använda register återställs i slutet

```
Subrutin Delay_ms ---------------------------------------
; Vänta i antal ms
; Inparameter: R7 - delay i ms
; -----------------------------------------------------------
DELAY_CALIB  EQU  MCLK / 36 ; utprovat värde

Delay_ms
      STMFD       SP!,{R0,R1}
      MOV         R0,R7

do_delay_ms
      LDR         R1,=DELAY_CALIB
loop_ms
      SUBS        R1,R1,#1
      BNE         loop_ms
      SUBS        R0,R0,#1
      BNE         do_delay_ms

      LDMFD       SP!,{R0,R1}
      MOV         PC,LR
; -----------------------------------------------------------
```

Subrutinen Read_pO

Subrutinen `Read_p0` läser av en knapp (`USR_RIGHT`) på port noll. Värdet på knappen maskas ut och returneras ut med register `R0`.

```
Subrutin Read_p0 ------------------------------------
; Avläsning av knapp i subrutin
; Utparametrar: R0
; ----------------------------------------------------
Read_p0
     STMFD      SP!,{R1}

     LDR        R1,=PIOA_PDSR
     LDR        R0,[R1]
     AND        R0,R0,#0x00080000  ; bitmaska fram aktuell knapp

     LDMFD      SP!,{R1}
     MOV        PC,LR
; ----------------------------------------------------
```

Uppgift

Som test för subrutinerna finns ett mycket enkelt huvudprogram. OBS! Innan du kan köra programmet måste portarna initieras, se labb 1.

```
; Huvudprogram -------------------------------------
read
     BL    Read_p0
     CMP   R0, #0    ; Ej tryckt => 1:a
     BNE   read
     LDR   R7,=1000
     BL    Delay_ms
Loop B     loop
; ------------------------------------------------------------------------
```

1. Sätt en brytpunkt på den eviga loopen i slutet av huvudprogrammet. Kör koden i fullfart och testa om det tar ca1 s från man trycker på knappen tills programmet stannar.

2. Starta om koden från början och stega runt i read-loopen med **Step Over**. Prova stegfunktionerna **Step Into** istället.

- Vad är skillnaden mellan Step Over och Step Into?

Starta om programmet. Stega dig fram till anropet av Subrutinen `Delay_ms` genom att hålla in knappen och stega. I början på koden kan man se vad stackpekaren SP är initierad till för värde.

Vilket värde har stackpekaren (R13)? ……

Titta på stacken genom att titta på minnet på den positionen. Sätt `R0` och `Rl` till några värden du känner igen (tex. `Rl=llllllll` och `R2=22222222`). Detta är bara för man enkelt ska kunna se värdet då det sparas i minnet. Stega vidare till raden

```
STMFD       SP!,{R0,R1}
```

Vilket värde har länk registret (R14)? ……..

Stega förbi denna rad och notera hur dessa värden sparas på stacken. Titta på värdena i registren. På vilka adresser sparas:

R0 = ……..

R1 = ……..

Vilket håll växer stacken? ……..

Titta på hur dom får nya värden inuti subrutinen, men att dom återställs då denna raden körs.

```
LDMFD       SP!,{R0,R1}
```

På detta sätt säger man att subrutinen har "lokala variabler". Värdena existerar bara inuti subrutinen

När subrutinen är klar laddas stackpekaren med återhoppsadressen i länkregistert (R14).

```
MOV  PC, LR
```

Vilket värde laddas programräknaren med?.....

Till vilken adress hoppar programmet?.....

Uppgift 2: Tändning/släckning av en lysdiod med maskning

Syfte

Att lära sig förstå hur man skriver till en port och endast påverkar vissa bitar. Förståelse för tekniken att maska in 1:or och 0:or. Att själv få skriva och konstruera en subrutin med en inparameter som senare skall användas i ett större program.

Teori: Skrivning till port/register med hjälp av maskning

Då man ska förändra en enstaka (eller flera) bitar i ett register använder man också tekniken med maskning, men på lite annat sätt.

Ettställa bitar i register:

Antag att man vill ettställa bit 0 och 2 i register 5. För att göra detta utför man operationen `OR` mellan registervärde och en s.k. bitmask med 1:a på de positioner man vill ettställa. Alla andra bitar sätts till noll. En bitmask för att ettställa bit 0 och 2 i register 5 blir då följande binära tal: `00000101 = 0x05`.

```
LDR   R0,=0xF0          ; Binära talet 11110000
ORR   R0, R0, #0x05     ; R0 OR:as med 00000101
                        ; R0 är nu     11110101 = 0xF5
```

Nollställa bitar i register:

Antag att man vill nollställa bit 5 i register 5. För att göra detta utför man operationen `AND` mellan registervärde och en s.k. bitmask med 0:a på de positioner man vill nollställa. Alla andra bitar sätts till ett. En bitmask för att nollställa bit 5 i register 5 blir då följande binära tal: `11011111 = 0xDF`.

```
LDR   R0,=0xF0          ; Binära talet  11110000
AND   R0, R0, #0xDF     ; R0 AND:as med 11011111
                        ; R0 är nu      11010000 = 0xD0
```

Vill man både nollställ och ettställa for man utföra en ettställning följt av en nollställning.

Inventera bitar med hjälp av XOR (Exklusivt Eller)

Ibland behöver man inventera bitar, d.v.s. man vill att alla nollor skall bli ettor och tvärt om. Exklusivt eller har följande sanningstabell:

X	Y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Genom att göra exklusivt eller med ”ettor” erhålls önskat resultat. Nedan följer ett exempel på hur man ”togglar” (byter tecken på) portB:s utgångar:

```
LDR   R1,=PIOB_ODSR       ; PIO PortB Output Status Register
LDR   R0,[R1]
AND   R0, R0, #0x0000007  ; Maska av bit 0-2
LDR   R1,=PIOB_CODR       ; PIO PortB Clear Output Data Register
STR   R0,[R1]             ; R0 skrivs tillbaks, nollställer ettor
EOR   R0, R0, #0x7        ; Inventera bit 0-2
LDR   R1,=PIOB_SODR       ; PIO PortB Set Output Data Register
STR   R0,[R1]             ; R0 skrivs tillbaks, ettställer nollor
```

Sammanfattning:	
Ettställa	OR:ar med 1:or på aktuell position. Nollor på resten.
Nollställa	AND:ar med 0:or på aktuell position. Ettor på resten.
Toggla	XOR med 1:or på aktuell position. Nollor på resten.

Uppgift: Tändning av godtycklig lysdiod i subrutin

I denna uppgift ska du lägga till en ny subrutin. Subrutinens namn skall vara `Led` och den ska tända resp. släcka lysdioder. Du behöver inte göra ett nytt projekt utan bygg vidare på koden i uppgift l.

`R7` skall användas som inparameter till funktionen. Mönstret i `R7`:s lägsta 3 bitar ska motsvara de tre lysdioderna (två gröna och en röd). Tänk på att spara undan de register som används inuti funktionen. Vidare tänk på att den röda lysdioden har omvänd aktivering i förhållande till de gröna. Det är bra att hantera detta i subrutinen så att aktiveringen blir transparant utifrån sätt, d.v.s. när man anropar subrutinen.

Utgå från följande skal: Led

```
STMFD      SP!,{Rl}
; Istället för Rl skriver du dom register du använder

<egen kod>

LDMFD      SP!,{Rl}
; Istället för Rl skriver du dom register du använder
MOV   PC,LR
```

Tänk på att maska 0:or resp. l:or på rätt sätt.

Tips: Tänk på vad som händer om man inverterar bitmasken

Glöm inte att initiera alla lysdioderna på portB!

Testning

Tänk på att testa subrutinen innan du går vidare. En bra variant är att loopa `R7` med tal mellan `0-7` och anropa subrutinen inuti loopen.

Gå inte vidare förrän programmet du förvissat dig om att subrutinen fungerar!

Uppgift 3: Avläsning av knappar i en subrutin

Syfte
Få konstruera en subrutin med utparametrar som senare skall användas.

Uppgift
Bygg vidare på tidigare uppgift.

I denna uppgift skall du skriva en funktion som läser av knapparna. `R7` och `R8` ska vara utparametrar och innehålla information om knapparnas status. Är `R7=1` är knapp `USR_RIGHT` intryckt. Är `R8=1` är knapp `USR_LEFT` intryckt.

Testning
En variant är låta alla dioder tändas om man trycker på knapp `USR_RIGHT` och alla släcks om man trycker knapp `USR_LEFT`. Utnyttja subrutinen `Led` för att tända resp. släcka lysdioderna.

Gå inte vidare förrän programmet du förvissat dig om att subrutinen fungerar!

Uppgift 4: Slutgiltig kod- Rinnande ljus åt godtyckligt håll

Syfte

När du kommit till denna uppgift ska du ha 3 subrutiner till ditt förfogande:

`Led` ; Tänder lysdioder enligt bitmönster i R7

`Read_p0` ; Läser av knapparna. R7=1 om knapp `USR_RIGHT`, R8 = l om knapp `USR_LEFT`

`Delay_ms` ; Väntar R7 antal millisekunder

I denna uppgift ska du använda dina subrutiner för att konstruera ett större program. Ingen modifikation av tidigare skrivna subrutiner skall behöva göras.

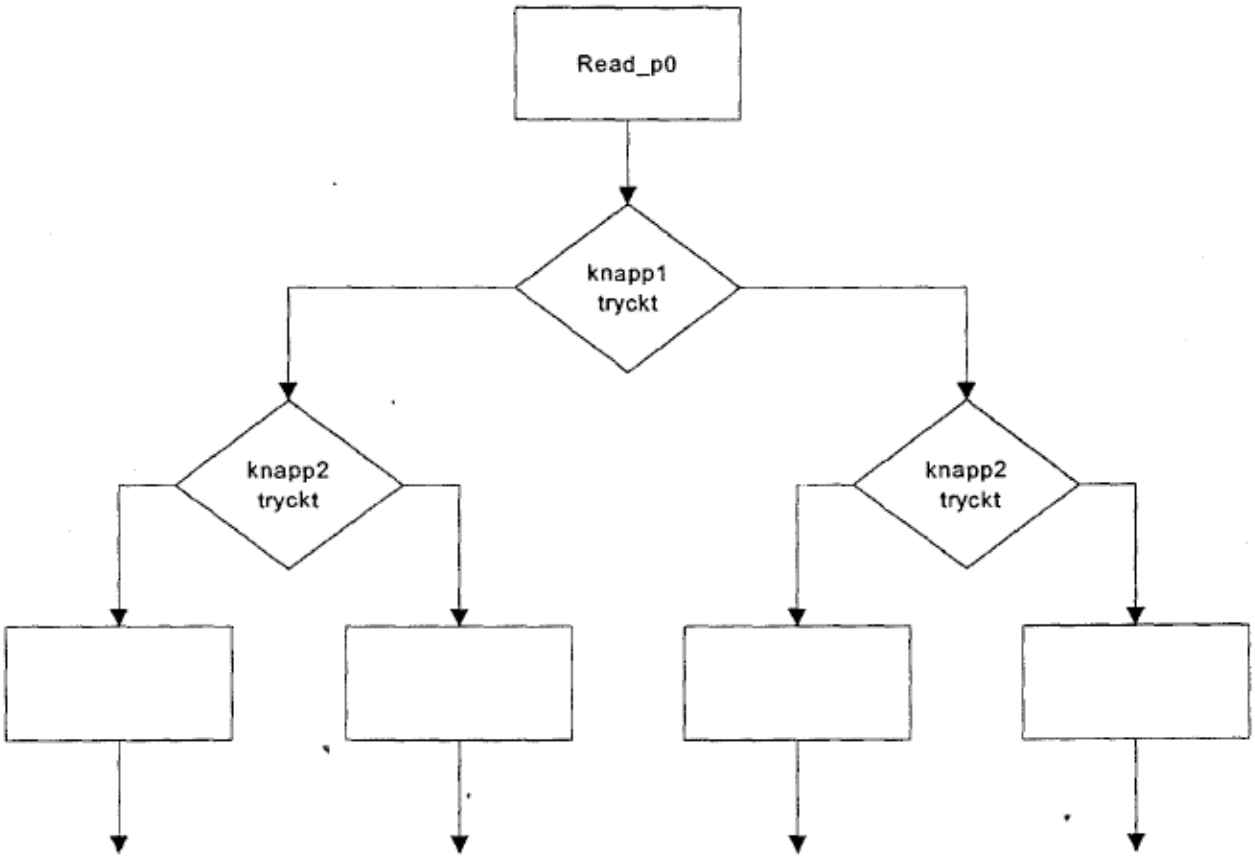
Uppgift

Bygg vidare på tidigare uppgift. Håller man inne den vänstra knappen, `USR_LEFT`, ska lysdioderna till synes rinna åt vänster, håller man inne den högra, `USR_RIGHT` ,ska de rinna åt höger. Så fort man släpper knapparna skall lysdioderna släckas. Lysdioderna börjar räkna från den översta av de gröna dioderna om man har den röda till vänster. Hålls båda knapparna inne ska alla lysdioder vara tända. Sätt tid mellan varje lysdiodväxling till ca 0.5 s

Tips:

Någonstans i huvudprogrammet kommer säkert nedanstående beslutsträd behöva implementeras. Märk som hjälp ut de labels ni tänker använda i figuren samt vilka villkor som gäller för de olika valen. Tänk på att det i princip blir labels till alla ställen där ni behöver hoppa i koden. Tänk på vad som skall hända i de olika fallen.

Bra ide är att rita flödesschema för hela huvudprogrammet innan ni börjar koda!!



OBS! detta är inte komplett flödesschema för huvudprogrammet.