Svar till Övning2 Datorteknik, HH vt12 - Programmering

Då jag i år fått göra om kursens övningar och dess svar så kan säkert vissa fel smugit sig in. Jag är tacksam för om ni skickar kommentarer om felaktigheter till:
Tomas.Nordstrom@hh.se,
Tack på förhand, Tomas Nordström

STACK
F5.1)
Det man lägger dit sist är det sista man plockar ut. Mekanismen att det sista man lägger dit är det första man tar bort kallas *LIFO. Last In, First Out*. Exempel en hög med tallrikar.

F5.2)
En *abstrakt datatyp* definieras av vilka operationer man kan göra på den och inte hur den egentligen är konstruerad. Ex) en telefon. Man kan slå sitt nummer, prata i den samt höra mottagaren. Det är dessa operationer som definierar själva telefonen. Inte hur den egentligen är byggd. Stacken karaktäriseras av dess UFO-struktur samt att man kan skriva (PUSH) och läsa (POP:a) från den. Det är dess operationer och inte hur den är implementerad som gör den till en stack.

F5.3)
Talet 12

F5.4)
```
LDR    SP, =0x20084000
STMFD  SP!, {R0,R2-R6}
```

F5.5)
```
Rl3=0x20003FE8   (backa SP 6 steg)
SP=0x2000400C    (öka SP 3 steg) <dock ej tillåten i thumbmode>
SP=0x20004000    SP oförändrad
SP=0x20003FC8    (backa SP 14 steg) <dock ej tillåten i
thumbmode>
```

SUBRUTIN

F5.11)

a) Nej! Då PC sparas på stacken blir inga problem med nästlade anrop. Man sparar ju inte på samma plats utan PC Pushas ju på stacken, dvs stackpekaren ökar
b) Då PC sparas på stacken vid varje anrop kommer stacken snabbt att fyllas av återhoppsadresser, med följden att man snabbt skriver över hela minnet!

F5.12.

Där sparas återhoppsadressen vid subrutinanrop

F5.13.

Länkregistret sparar återhoppsadress. Gör man ett nästlat anrop, dvs ett nytt subrutinanrop i subrutinen kommer den nya återhoppsadressen skrivas in i länkregistret. Man förlorar då informationen om första anropet och man kan inte hoppa tillbaks utan fastnar i en evig loop

ADRESS	KOD	KOMMENTAR
0	*Bl test*	*LR = Adress 0*
1		
2	*Test*	
3	*Bl test2*	*LR = Adress 4 (skrivs över)*
4	*Återhopp*	*Hoppar till LR (dvs till4)* **Fel! Blir evig loop**
5	*Test2*	
6	*återhopp*	*Hoppar till LR (dvs till 4)*
7		

F5.14

Spara register R0, Rl, R2, R3, R4 och LR på adressen i R13 och bakåt i minnet. Uppdatera Rl3, dvs minska talet med 6*4 (6st 32-bitarsplatser)

F5.15

Inga! Det är assemblerdirektiv och tar inget minnet på processorn. Betyder bara att överallt i koden det står `rPO` byts det ut till 0x01FFB000

F5.16

Det implementeras genom att man sparar registren på stacken i början av subrutinen. De återställs senare i slutet på rutinen.

```
subrut      STMFD SP!,{RO-R4, LR}
            <kod>
            LDMFD SP!,{R0-R4, PC}
```

Sparar även LR ifall det förekommer nästlade anrop i subrutinen

F5.17
Felet är att subrutinen Delay_ms anropar en ny subrutin (nästlade subrutiner) UTAN att spara undan LR (se STACK uppg3).

STMFD SP!,{R1}	ändras till	STMFD SP!,{R1, LR}
MOV PC, LR	ändras till	LDMFD SP!,{R1, PC}

MINNESMAPPAD I/O

F5.21)
Minnesmappning

Fördel:
Sparar instruktioner. Använder vanliga LDR samt STR för att nå I/O
Nackdel:
Tar en del av adressrymden (minnet)

F5.22
…

F5.23
Där ligger alla specialregister som används för konfigurering av processor samt I/O-hantering. Skriver man kod över dessa register händer konstiga och ofta mycket oväntade saker

F5.24
Extern kommunikation. Ofta implementerat som fysiska pinnar på kapseln. Man kan helt enkelt nå dessa pinnar för läsning/skrivning.

F5.25)
Dessa huvudregister kan normalt inte skrivas till direkt i SAM3U, utan bitars sätts respektive nollställs via två speciella styrregister (”enable” och ”disable” register). Detta gör det möjligt att ändra en speciell bit/pinne på en port utan att känna till värdet på de andra bitarna . Vid skrivning till dessa (”enable” och ”disable” register sätts respektive släcks bitar i huvudregistret (vid skrivning till dessa två register så anger en 1:a att motsvarande bit ska sättas eller släckas, medan en 0:a har ingen effekt). Vad detta huvudregister innehåller kan avläsas via en tredje minnesposition kallat statusregister.

Man skulle klara sig med ett register men får då behöver man först göra en läsning av portens nuvarande värde, för att sedan modifiera den bit man är ute efter att ändra, och sist göra en skrivning av hela portens alla bitar. Vilket då kräver fler instruktioner än vad som behövs med denna ”tre-register” konstruktion. Nackdelen är att man inte kan sätta och släcka bitar samtidigt, därför finns funktionen synkron skrivning till PIO_ODSR, om sätter upp den funktionen (jmf 30.5.5 i [SAM3U]).

BITMASKNING

F5.31

```
ORR   Rl, Rl, #0xBB
```

F5.32

```
LDR   RO, =0xFFFFFFC3
AND   Rl, Rl, R0
```

F5.33

```
LDR   RO, =00110101b   ; bitmask för ettställning (Ox35)
ORR   R4,   R4, RO
LDR   R0, =0xFFFFFF35
AND   R4, R4, R0       ; bitmask för nollställning
```

F5.34

```
LDR   R0, =PIOA_PDSR
LDRb  R3, [R0]
AND   R3, R3, #0x000C0000
```

F5.35

```
LDR   R0, =PIOA_SDSR
LDR   R1, =PIOA_CDSR
LDR   R2, #1001b
STR   R2, [R0]
EOR   R2, #1111b
STR   R2, [R1]
```

F5.36

```
adress     EQU   0x20003000

           LDR   R3,=adress
           LDRb  Rl,[R3]
           MOV   R2,#1
           MOV   R2,R2,LSL Rl     ; skifta upp l:a
           ORR   R0,R0,R2         ; Maska!
```

PROGRAMMERING

F5.41)

```
CODE_START EQU  0x20000000
STACK_TOP EQU   0x20084000

PIOA_PER  EQU   0X400E0C00
PIOA_ODR  EQU   0X400E0C14
PIOA_IDR  EQU   0X400E0C44
PIOA_PUER EQU   0X400E0C64
PIOA_PDSR EQU   0X400E0C3C

          ORG CODE_START

; initiering Stack ----------------------------------
Main      LDR   SP,=STACK_TOP

; initiering INPORT ---------------------------------
          LDR   R0,=PIOA_PER
          LDR   R1,=0X000C0000    ;mask for bit 18&19
          STR   R1,[R0]           ;enable PIOA

          LDR   R0,=PIOA_ODR
          STR   R1,[R0]           ;output disable

          LDR   R0,=PIOA_IDR
          STR   R1,[R0]           ;disable interrupt

          LDR   R0,=PIOA_PUER
          STR   R1,[R0]           ;pull_up enable


TEST_SUB  PUSH {R0-R4,LR}  ; eller STMFD SP!,{R0-R4,LR}
          ; Övrig subrutin kod
          POP {R0-R4,PC}   ; eller LDMFD SP!,{R0-R4,PC}
```

F5.42)

```
ASCII_A   EQU       65
ASCII_Z   EQU       90
diff      EQU       32

          LDR       R0, =TEXT

loop      LDRb      R1, [R0]
          CMP       R1, #0
          BEQ       exit        ; om NULL => bryt
          CMP       R1, #ASCII_A
          BLT       oka         ; bokstaven mindre än A => nästa tecken
          CMP       R1, #ASCII_Z
          BGT       oka         ; bokstaven större än Z => nästa tecken
                                ; annars bokstav mellan A-Z
          ADD       R1, R1, #diff         ; gör om till små
          STRb      R1, [R0]
oka       ADD       R0, R0, #1 ; öka adresspekare
          B         loop
exit      B         exit

TEXT      DCB       "Detta ar en text"
```

F5.43

```
read_base  EQU        0x01001000
write_base EQU        0x01005000

           LDR        R0, =read_base
           LDR        R1, =write_base

loop       LDR        R2, [R0]    ; ladda in tal
           CMP        R2, #0
           BEQ        exit        ; om lika med noll => bryt!
           BGE        next        ; om positivt, gå vidare
           MVN        R2, R2      ; annars 2-komplementera
           ADD        R2, R2, #1

next       ADD        R0, R0, #4 ; oka läsadress
           STR        R2, [R1]    ; skriv till destinationsadress
           ADD        R1, R1, #4 ; öka skrivadress
           B          loop

exit   ,   B          exit
```

F5.44

```
           LDR        R1, =rPDAT0
           LDR        R0, =0x99

loop       STRb       R0, [R1]              ; Skriv talet till PORT0
           CMP        R0, #0
           BEQ        quit                  ; om noll, avsluta
           AND        R2, R0, #0x0F         ; AND:a ut entalssiffra
           CMP        R2, #0
           BEQ        tiotal     ; om entalssiffra noll, räkna ner 10-tal

           SUB        R0, R0, #1
           B          loop
tiotal     SUB        R0, R0, #7  ; genom att minska med sju hamnar
                                  ; man på 9 tänk på att det är
                                  ; hexadecimal.  Tex 0x90-7=0x89,
                                  ; vilket är det vi vill ha
           B          loop

quit       B          quit
```

Instruktionstolkning

F5.51

a) korrekt! R2 = R4

b) korrekt! R2 = 4

c) FEL! För stort tal för immediate adresseringsmode l

d) FEL Stödjer ej detta sätt att skriva

e) FEL Går ej läsa från en adress med MOV
f) FEL! Andra operanden #45 går ej att skriva på detta sättet
g) RÄTT Läser in talet bakom likamedteckning
h) RÄTT R2 får vad som ligger på minnesadressen ADRESS
i) RÄTT R2 får själva talet ADRESS
j) FEL! Går inte att använda LDR för att flytta mellan register
k) RÄTT R3 =vad som ligger på adressen [R3+ 12]
l) RÄTT R2 =vad som ligger på adressen [R1+ 12]. Automatisk uppdatering av R1.
m) RÄTT talet R0 lagras på adressen R0
n) RÄTT Lagra Rl på adressen ADRESS.
o) FEL. STR stödjer ej immediate adresseringsmode

F5.52
MOV flyttar mellan register
LDR laddar från minnet till register eller tvärtom
Dvs MOV håller sig internt i CPU. LDR går ut mot minnet och läser

F5.53
Kod l är oberoende av var den läggs, Kod2 fungerar bara om instruktionen ligger på en minnesadress nära adressen rPDATO.

F5.54
Genom att assemblatorn placerar ut talet TAL i minnet efter koden. Sen översätts pseudoinstruktionen
`LDR R2, =TAL` till `LDR R2, [PC, #offset]`
Man läser in ett tal som ligger offset antal byte framför adressen från var instruktionen ligger.

Analysera, Felsök och rätta kod!

F5.61)
Programmet nedan skall addera TALl och TAL2 och lägga i R3. Vilka fel finns?

```
TALl        EQU 24     ; två tal som skall adderas
TAL2        EQU 34
RAM_START   EQU 0x20000000

            ORG RAM_START
            LDR R1,TAL1
            LDR R2,TAL2
            ADD R3,R1,R2
            END
```

Dessa instruktioner läser in adresserna till talen och inte dess värden.

Lägg till ”=” före Tal1/2

F5.62)
Programmet nedan skall addera TAL1 och TAL2 och lägga i R3. Vilka fel finns?

```
RAM_START   EQU 0x20000000

            ORG RAM_START
            MOV R1,=TAL1
            MOV R2,=TAL2
            ADD R3,R1,R2

TAL1        DSB 24          ; två tal som skall adderas
TAL2        DSB 34
            END
```

MOV kan inte läsa ifrån minnet.

Byt MOV till LDRb

DSB definierar utrymme och inte konstanter.

Byt DSB mot DCB

F5.63)
Programmet nedan skall addera 8-bitarstalen TAL1 och TAL2 och lägga på adressen SVAR. Vilka fel finns?

```
RAM_START   EQU 0x20000000

            ORG RAM_START
            LDR R1,TAL1
            LDR R2,TAL2
            ADD R3,R1,R2
            STR R3,SVAR

SVAR        DS8 1
TAL1        DCB 23          ; två tal som skall adderas
TAL2        DCB 12
            END
```

LDR och STR arbetar på 32 bitar

Byt LDR mot LDRb och STR mot STRb

F5.64)
Programmet nedan skall förändra värdet på PortA genom att addera talet med ett. Detta ska ske i en evig loop. Anta porten är konfigurerad tidigare. När programmet körs händer inget på porten. Vilka fel är i koden?

```
RAM_START   EQU 0x20000000

            ORG RAM_START
LOOP        LDR R2,PIOA_PDSR
            LDRb R1,[R2]
            ADD R1,R1,#1
            AND R1,Rl,#0xFF  ; Maska bort allt utom dom 8 lägsta
            B LOOP
```

R2 får inte adressen till porten (saknar =)
Man borde inte ladda portadressen i loopen
Koden antar porten vara 8-bitar men det är inte givet
Skrivning till porten saknas

Syntes av kod

F5.71)

a)

```
TAL   EQU Ox12345678
      LDR Rl, =TAL
```

b)

```
      LDR Rl,TAL
TAL   DC32 Ox12345678
```

Alt)

```
      LDR Rl, =TAL
      LDR Rl, [Rl]
TAL   DC32 Ox12345678
```

c)

```
      LDR Rl, Ox12345678
```

F5.72)

```
MOVE        STMFD       SP!, {R1-R3, LR}
            LDR         R1, =SOURCE             ; initera adresser
            LDR         R1, [R1]
            LDR         R2, =DEST
            LDR         R2, [R2]

LOOP        LDR         R3, [R1]                ; kopiera
            STR         R3, [R2]
            ADD         R1, R1, #4              ; öka pekare
            ADD         R2, R2, #4
            SUB         R0, R0, #1              ; minska antalet
            CMP         R0, #0
            BNE         LOOP                    ; om inte slut, loopa!
            LDMFD       SP!, {R1-R3, PC}
```

Kortare variant. Bygger på att Adresserna SOURCE och DEST inte är för långt från koden

```
MOVE        STMFD       SP!, {R1-R3, LR}
            LDR         R1, SOURCE
            LDR         R2, DEST
LOOP        LDR         R3, [R1], #4
            STR         R3, [R2], #4
            SUBS        R0, R0, #1              ; minska och sätt flaggor
            BNE         LOOP
            LDMFD       SP!, {R1-R3, PC}
```

F5.73)

```
TRANSL      CMP         R0, #0
            BLT         exit                    ; om R0<0 then exit
            CMP         R0, #9
            BGT         exit                    ; om R0>9 then exit
            LDR         R1, =tabell             ; initera R1 till tabell
            ADD         R1, R1, R0              ; R1 ökas upp med positionen
            LDRb        R1, [R1]                ; inläsning
exit        MOV         PC, LR
```

F5.74)

```
FIND    STMFD   SP!, {R2-R3, LR}
        LDR     R2, =tabell             ; initera R2 till tabell
        MOV     R1, #0                  ; positionsräknare R1=0
loop    LDRb    R3, [R2], #1            ; R3 = mem[R2]   R2=R2+1
        CMP     R0, R3
        BEQ     exit                    ; om hittat => exit
        ADD     R1, R1, #1              ; öka positionsräknare
        CMP     R1, #10
        BLT     loop                    ; om R1<10 then loop
        MOV     R1, #-1                 ; inte hittat, dvs R1=-1
exit    LDMFD   SP!, {R2-R3, PC}        ; återgå
```