Övning2 Datorteknik, HH vt12 - Programmering

För denna övning behöver man adresskarta och beskrivning av laborationsplattform. Finns bland föreläsningsanteckning samt i bilaga l till Lab l. Använd även beskrivningar av ARM-instruktioner flitigt för få viss känsla för vad som står vad. Du behöver även den beskrivning av porthantering som finns i materialet inför laborationerna.

stack

F5.1)
Förklara principen LIFO

F5.2)
Förklara vad som menas med en *abstrakt datatyp* och varför är stacken en sådan?

F5.3)
Anta man gjort följande på en stack. PUSH 12, PUSH 3, PUSH 7, PUSH 8, POP, POP, PUSH 2, POP, POP, Vilket värde får man ut vid nästa POP?

F5.4)

a) Initiera stack på ARM. Anta den ska läggas på toppen av det interna SRAM1. Välj adress så den blir jämt delbart med 4.
b) Skriv en instruktion för att spara R0, R2, R3, R4, R5 och R6 på stacken. Tips: Tänk på att stacken ska växa åt rätt håll!

F5.5)
Anta SP = 0x20004000 på ARM-processorn. Vad är innehållet i SP efter följande instruktioner?

```
a) STMFD Rl3!, {R0-R4,LR}
b) STMFA SP!, {R0,R4,LR}
c) LDMEA SP, {R0,R4,LR}
d) LDMFA SP!,{R0-Rl2,R4}
```

Subrutin

F5.11)
Anta att det på en processor (inte ARM) finns en instruktion som heter JSR som hoppar till en subrutin. Programräknaren sparas automatiskt på stacken. Vid återgång från subrutinen finns instruktionen RET som poppar programräknaren från stacken.

a) Blir det problem vid nästlade anrop med subrutinmekanismer enligt ovan?
b) Vad kan hända vid okontrollerade rekursiva anrop? (då en subrutin anropar sig själv). Tips: Tänk på vad som sparas vid varje subrutinanrop!

F5.12)
Vad har länkregistret LR for användningsområde på ARM?

F5.13)
Förklara varför nästlad subrutin är lite lurigt på ARM-processorn.

F5.14
Förklara vad följande instruktion gör! Var i subrutinen bör den vara placerad?
```
STMFD R13!,{R0-R4, LR}
```

F5.15
Hur mycket minne går åt för följande rader
```
PMC_PCER EQU  0x400E0410 ; Peripheral Clock Enable Register
PMC_PCDR EQU  0x400E0414 ; Peripheral Clock Disable Register
PMC_PCSR EQU  0x400E0418 ; Peripheral Clock Status Register
WDT_MR  EQU   0x400E1254 ; Watchdog Timer Mode Register
```

F5.6
Hur implementeras begreppet lokala variabler på ARM? Anta tex att en subrutin internt använder register R0-R4 samtidigt som anropande program gör det. Hur löses detta? Visa med kod!

F5.7
Bektrakta nedanstående subrutiner. Förklara vad är fel med subrutinen Delay_ms? Ange lösning!

```
; --------- Subrutinen har R0 som inparameter
DELAY_CALIB antas vara deklarerad


Delay_ms
            stmfd sp!,{rl}
do_delay_ms
            ldr  rl,=DELAY_CALIB
loop_ms
            subs rl,rl,#l
            bl  Dec_r0
            bne  do_delay_ms

            ldmfd sp!,{rl}
            mov  pc,lr

; --- Subrutinen Dec_r0 har som enda uppgift att minska RO med l ---
Dec_r0
            subs r0,r0,#l
            mov  pc,lr
; -----------------------------------------------------
```

Minnesmappad I/O

F5.21
Datorer har oftast sk. minnesmappad I/O. Ange en fördel/nackdel med detta

F5.22
Initiera PortA (PIOA) och PortB (PIOB) efter följande beskrivning.
Skriv kod för att läsa av BP_RIGHT och sätt Z flaggan om knappen inte är nedtryckt
Skriv kod för att tända lysdiod 2 (LED2) utan att påverka resten av registret

Se bilaga 1 till lab1, ”SAM3U-EK Evaluation Kit User Guide”, (Kap 4 och Tabell 4-5) för hur hårdvaran är kopplad till PortA (BP_LEFT och BP_RIGHT kopplade till bit 18 resp. 19 och PortB (LED1 och LED2 kopplade till bit0 resp. 1).

F5.23
Anta minneskartan ser ut som för hårdvarusystemet. Varför ska man inte lägga koden på höga adresserna 0xE0000000 och framåt? Vad händer om man gör det?

F5.24
När man pratar om portar på en dator, vad menar man då?

F5.25
Vår SAM3U har i styrning av I/O funktionen hos en port tre olika register (enable, disable, status) kopplade till en speciell styruppgift. Förklara funktionen hos vart och ett av registren. Skulle man klara sig med bara ett register istället?

Bitmaskning

F5.31
Skriv kod för att ettställa bit3 och 7 i R1

F5.32
Anta man vill nollställa bit2-5 i R1. Skriv ARM-kod för detta!

F5.33
Anta man vill förändra de 8 lägsta bitarna i R4 till 00110101, resten opåverkade. Skriv ARM-kod för detta!

F5.34
Anta bitarna 18 och 19 i PortA är konfigurerade som insignaler. Skriv kod för att läsa av PortA och maska ut den viktiga informationen till R3.

F5.35
Anta hela PortB (32-bitar) är konfigurerad som utsignal. Skriv kod för att skriva talet 1001_2 till de lägsta fyra bitarna utan att påverka resten

F5.36
Skriv ett program som ettställer en bit i R0. Vilken bit som skall ettställas anges på adress 0x20003000 som ett tal mellan 0-31.
Tips: Man kan använda skiftning för att placera en etta i korrekt position i bitmasken. Lämplig arbetsgång är

1) Hämta in talet från minnet.
2) Skifta ettan i korrekt position.
3) Ettställ

Programmeringsuppgifter.

Gör gärna flödesschema för din kod!

F5.41)

Skriv kod för initiering av ett ARM-system med minneskarta enligt labplattform.

- Lägg kod i SRAM0
- Stacken i början på internt SRAM1.
- Initiera även portA enligt labsystemet.
- Skriv ramen för en subrutin på ovanstående system. Subrutinen antas använda RO-R4 internt. Kalla rutinen `TESTSUB`. Tänk på åt vilket håll stacken ska växa!!

F5.42)

Skriv en subrutin för att omvandla stora till små tecken. Utgå från att texten ligger lagrad i minnet på adressen TEXT och är terminerad med NULL. Den är inlagd med hjälp av assemblerdirektivet DCB på följande sätt:

```
TEXT   DCB    "Detta ar en Text"  ; Lägger automatiskt på NULL på slutet
```

F5.43

Skriv ett program som omvandlar tal som ligger på adress 0x01001000 till dess absolutvärden. Talen ska sparas på adress x01005000 och framåt. Talen avslutas med talet 0. (Det finns inget SRAM där i vårt system, men ignorera detta i denna övning)

Ex)

Adress	Före omvandling	Adress	Efter omvandling
x01004000	1234	x01005000	1234
x01004004	3	x01005004	3
x01004008	-345	x01005008	345
x0100400C	24	x0100500C	24
x01004000	-99	x01005000	99
x01004004	0	x01005004	0

F5.44)

Gör ett program som räknar ner PortB (bit 7-0) i BCD-kod. Börja på 99, dvs 10011001. Då man kommer till noll ska programmet stanna i en evig loop. Anta att PortB är konfigurerad endast som utsignaler.

För beskrivning av BCD-formatet, se föreläsningsanteckningar eller http://sv.wikipedia.org/wiki/BCD

Instruktionstolkning

F5.51)

Vilka av följande instruktioner är korrekta? Ange vad dom korrekta instruktionerna utför!

```
a) MOV R2, R4
b) MOV R2, #4
c) MOV R2, #Ox46285821
d) MOV R2,[R1]
```

```
e) MOV R3, ADRESS
f) LDR R1, #45
g) LDR R2, =Ox47582944
h) LDR R2, ADRESS
i) LDR R5, =ADRESS
j) LDR R3, R4
k) LDR R3, [R3, #12]
l) LDR R2, [R1], #12
m) STR RO, [RO]
n) STR R1, ADRESS
o) STR #45, MINNE
```

F5.52)
Vad är principskillnaden mellan instruktionerna MOV och LDR?

F5.53)
Varför är kod l att föredra framför kod2 då det gäller inläsning av data från PortA? Vad är restriktionen hos kod2?

Kod1	Kod2
`LDR RO, =PIOA_PDSR` `LDRb R1, [RO]`	`LDRb R1, PIOA_PDSR`

F5.54)
Hur kan ARM-instruktionen

```
LDR R2, =TAL
```

läsa in 32-bitarstal i R2 då instruktionerna på ARM endast är 32-bitar långa?

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
```

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        DCB 24          ; två tal som skall adderas
TAL2        DCB 34
            END
```

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
```

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
```

Syntes av kod

F5.71)
Anta man vill läsa in talet 0x12345678 till R1. Detta går att göra på tre sätt. Skriv kod för att göra detta

a) Genom att nyttja assemblerdirektiv EQU samt instruktionen LDR
b) Genom att nyttja assemblerdirektiv DC32 samt instruktionen LDR
c) Genom att bara nyttja instruktionen LDR

F5.72)
Skriv en subrutin, MOVE, som flyttar ett minnesblock till ett annat.
Startadress på blocket antas stå på adressen
Destinationsadressen står i minnescellen DEST
Antal 32-bitars ord att flytta antas vara i RO.

Ex) R0 = 4, subrutinen MOVE anropas. Bild på minnet nedan (igen finns det inget SRAM där i vårt system, men ignorera detta i denna övning)

NAMN	ADRESS	INNEHÅLL
DEST	0x00001000	0x00001F00
SOURCE	0x00001004	0x00002000
	0x00001F00	23
	0x00001F04	6
	0x00001F08	98
	0x00001F0C	32
Efter utförd subrutin:		
	0x00002000	23
	0x00002004	6
	0x00002008	98
	0x0000200C	32

F5.73
Skriv en subrutin, TRANSL, för att slå upp värden i en tabell. Tabellen används för att översätta vanliga siffror till vissa bitmönster. Inparameter till subrutinen är RO. Returnera bitmönstret i R1. Är R0 utanför tillåtet område, dvs under 0 eller över 9 ska talet 0 returneras ut i R1

Anta tabellen är definierad på följande sätt. Positionen i tabellen anger vad som skall översättas

```
tabell DCB    Ox12, 0x32,  0x33,  0x22,  0x10,  0x63,  0x23,  0x65,  0x34, 0x11
              ;O     1      2      3      4      5      6      7      8      9
```

ex) är R0= 3 och subrutinen TRANSL anropas kommer R1 bli 0x22, dvs 00100010b

F5.74)
Skriv en subrutin, FIND, som utgår från tabellen i tidigare uppgift, men istället returnerar vilken position talet finns på. Anta att tal man söker på kommer i R0 och positionen returneras ut i R1. Finns inte talet returneras talet –1.

ex) är R0= 0x22 och subrutinen FIND anropas. R1 blir då 6