Laboration 3: Ett komplett system - Tärning

1 Inledning

Laborationen ska bidra till en grundläggande förståelse för:

- Design och implementation av ett program skrivet med assembly-kod
- Struktur (i filer) av ett program skrivet med assembly-kod.
- Access till olika typer av data som finns i programminnet och dataminnet.
- Användning av en tabell för att ”mappa”/konvertera mellan två värden

1.1 Utrustning

Komponenter som behövs för denna laboration: Ingen ny hårdvara behövs

2 Beskrivning av laboration

I den här laborationen ska ni sätta ihop delsystem som skapats i de tidigare labbarna (subrutiner för avläsning av tangentbord och för att skriva till LCD) och sätta ihop det på ett snyggt sätt till ett fungerande system, med god struktur och dokumentation. För att klara av laborationen måste ni, precis som i tidigare laborationer, förbereda laborationen genom att läsa instruktionerna noggrant, studera teori och datablad, etc. Laborationen består av följande moment:

1. Strukturering och dokumentering av programkod
2. Implementering av mappningstabell
3. Användning av strängar i programminnet
4. Programmering av tärningsapplikation
5. Lagring av statistikdata
6. Test av komplett system

2.1 Syfte och mål

Laborationen går ut på att sätta ihop delsystem som skapats i tidigare labbar till ett komplett system. Systemet skall ”snyggas till” så att det har bra struktur och är väl kommenterat.

2.2 Examination

2.2.1 Inlämning av rapport och programkod

Utöver en rapport ska ni även lämna in programkod. Instruktioner för detta är beskrivet på inlämningssidan för denna laboration (på Canvas).

2.2.2 Praktisk och muntlig redovisning

Den praktiska delen av laborationen examineras efter att laborationens sista moment har avslutats. Följande ska redovisas:

- Programkod (med god struktur och kommentarer).
- Demonstration av systemet, fullt fungerande enligt specifikation.

Ni ska även kunna redogöra för funktionalitet avseende krets och programkod.

3 Moment 1: Strukturering och dokumentering av programkod.

3.1 Beskrivning

I detta moment ska ni använda färdigskrivna rutiner från tidigare lab (Lab 2) och strukturera upp denna kod innan ni går vidare.

3.2 Konfigurering av projekt och strukturering av kod

Uppgift 3.2.1

Skapa ett nytt projekt i Atmel Studio. Kalla det för **lab3**.

Uppgift 3.2.2 (redovisas i programkod)

Kopiera och inkludera alla kod-filer ni hade i lab 2 (**xxxx.inc)** i ert projekt. Använd: *högerklicka på projektet -> add -> add existing item*. Kom även ihåg att anpassa huvudprogrammet för detta!

Uppgift 3.2.3 (redovisas i programkod)

Om inte tangentbords-delarna redan är i en egen fil (tex keyboard.inc), så skapa en ny fil och klipp ur och flytta koden från huvudprogrammet. Glöm inte att anpassa huvudprogrammet för detta!

Passa på att byta ut NOP-instruktionerna i mitten av tangentbordsrutinen mot ett anrop till lämplig delay-funktion (ca 5-10 mllisekunder kan vara lagom delay för att rutinen skall fungera och samtidigt hantera knappstuds). Tänk på att kolla vilka register som används i delayrutinerna, så att inget förstörs!

Uppgift 3.2.4 (redovisas i programkod)

Se till att varje fil (huvudprogrammet och alla include-filer) är strukturerade, efter följande mall (tex):

1) Kommentarblock med:
    a. Beskrivning av innehållet i filen
    b. Namn på de som skapat filen
    c. Datum
2) Definitioner
3) Konstanter
4) Init-rutiner
5) Interna rutiner (som inte är tänkta att anropas ”utifrån”)
6) Externa rutiner (som skall kunna anropas ”utifrån”)
7) Main-program (i förekommande fall)

Uppgift 3.2.5 (redovisas i programkod)

Se till att all kod är väl kommenterad, dvs beskriver vad som logiskt händer, INTE vad assembler-instruktionen innebär! Subrutiner skall ha några rader före rutinen, som innehåller:

- Hur in/ut-parametrar hanteras (tex vilka register som används)
- Vilka register som används internt i rutinen (och som alltså efter rutinen exekverats inte kommer att innehålla det de innehöll innan anropet)
- Vilka begränsningar som finns (tex max-storlekar, etc.) och ev felhantering

```
;-----------------------------------------------------------------
: Example_Subroutine
; Parameters IN:    R24: contains number.
;                   R16: contains position (0..16).
;            OUT:   R24: contains ASCII character.
;   Limitations: If R16 <0 or R16 >16, no action is taken.
;-----------------------------------------------------------------
```

Uppgift 3.2.6 (redovisas i programkod)
Se till att huvudprogrammet inte innehåller ”för mycket kod”. Huvudprogrammet skall vara ganska litet och innehålla:

- Anrop till subrutiner för att initiera systemet
- En oändlig loop, innehållande tex:
  - Kod för enkla tester
  - Anrop till subrutiner.

Om huvudprogrammet innehåller för mycket kod, gör nya subrutiner och flytta delar dit. Anrop till subrutiner. I denna lab är det lämpligt att implementera huvud-loopen i figur 6-1 i huvudprogrammet.

Uppgift 3.2.7
Bygg systemet, ladda ner och verifiera att programmet fortfarande fungerar som tidigare.

4 Moment 2: Implementering av mappningstabell

4.1 Beskrivning

Tangentbords-kretsen och programmet från Lab 1 returnerar ett tal i RVAL (R24), som motsvarar vilken Rad/Kolumn där en knapp tryckts ner, eller en kod som representerar ”NO_KEY”.

Då man skall skriva ut vilken tangent som tryckts ned på LCD-displayen, vill man gärna visa tangentens nummer istället för denna kod. Vi behöver alltså göra en **mappning**. Vi behöver också konvertera från ett nummer (0-11) till en ASCII-bokstav. Vi gör dessa två saker i ett steg.

4.2 Mappningstabell i programminnet

Vi vill ha en fast tabell, som inte behöver fyllas i i början av programmet. Vi lägger den därför som en del av programmet. Detta innebär att vi måste använda speciella instruktioner för att komma åt den från programmet.

Uppgift 4.2.1 (redovisas i programkod)
Efter konstanterna i den fil (tex keyboard.inc) som innehåller tangentbordsrutinerna, lägg till:

```
map_table: .DB "147*2580369#"
```

Uppgift 4.2.2 (redovisas i rapport)
Förklara vad den rad vi lade till i koden i uppgift 4.2.1 innebär och varför innehållet mellan ”” är organiserat som det är ovan.

Uppgift 4.2.3 (redovisas i programkod)
Lägg nu till en subrutin (i filen med tangentbordskod) som konverterar det tal vi får från rutinen ”read_keyboard” till ett ASCII-tecken, som kan skrivas ut på displayen. Basera koden på följande:

```
LDI ZH, high(map_table <<1)   ;Initialize Z pointer
LDI ZL, low(map_table <<1)
ADD ZL, RVAL                  ;Add index
LDI RVAL, 0x00
ADC ZH, RVAL                  ;Add 0 to catch Carry, if present
LPM RVAL, Z
```

Uppgift 4.2.4 (redovisas i rapport)
Förklara vad kod-raderna i uppgift 4.2.3 gör. INTE vad de olika instruktionerna betyder, utan **varför** de behövs!

5 Moment 3: Användning av strängar i programminnet

5.1 Beskrivning

Detta moment går ut på att ni skall lagra strängar i programminnet och hämta dem därifrån istället för att skriva ut tecken för tecken, som ni gjort hittills.

5.2 Strängar i dataminnet

Vi kan definiera ett antal strängar och tilldela dessa initiala värden.

Uppgift 5.2.1 (redovisas i programkod)
Efter konstanterna i den fil (tex LCD.inc) som innehåller displayrutinerna skall ni lägga till följande:

```
Str_1:    .DB       "Welcome!"
          .EQU      Sz_str1 = 8          ; Size of ”str_1”
```

Uppgift 5.2.2
Bland subrutinerna i huvudprogrammet skall ni skapa en ny subrutin:

```
write_welcome:
          LDI ZH, high(Str_1<<1)
          LDI ZL, low(Str_1<<1)
          CLR R16
Nxt1:     LPM R24, Z+
          RCALL lcd_write_chr
          INC  R16
          CPI  R16, sz_str1
          BRLT Nxt1
          RET
```

Uppgift 5.2.3 (redovisas i programkod)
Lägg in ett anrop till denna rutin i huvudprogrammet och testa att den fungerar.

Uppgift 5.2.4 (redovisas i rapport)
Förklara vad de rader vi lade till i koden i uppgift 5.2.1 och 5.2.2 gör. Vad innebär ”+” efter Z?

Uppgift 5.2.5 (redovisas i rapport)
I exemplet använder vi en extra konstant som talar om hur många tecken som ingår i strängen. Det finns ett annat sätt att hantera strängens slut. Googla på tex: ”strings in C” och beskriv kortfattat (kod behövs inte) hur man skulle kunna göra istället. Även kod och exempel i föreläsningsbilder kan vara till hjälp.

Uppgift 5.2.6 (redovisas i programkod)
Ändra koden ni skrev i Uppgift 5.2.2, så att den använder det sätt som ni hittade i Uppgift 5.2.5.

Uppgift 5.2.7 (redovisas i programkod)
Ni ser nu att det enda som är unikt när man skriver ut strängar på det här sättet är referensen till själva texten (”str_1” i exemplet ovan). Skriv nu en subrutin (tex i LCD.inc) som gör allt utom de 2 första raderna (dvs de som ger värden till Zl och ZH). Cursorn skall stanna direkt efter slutet på texten.

Skriv sedan en MACRO ”PRINTSTRING” som tar en parameter (strängnamnet) och som laddar ZL och ZH och sedan anropar subrutinen ni just skrivit. Man skall alltså nu kunna skriva ut strängar i programmet på detta sätt:

```
Hello_str1:     .DB ”Hello”,0                ; definition av text i början på filen

…

PRINTSTRING Hello_str                        ; Skriver ”Hello” på displayen

…
```

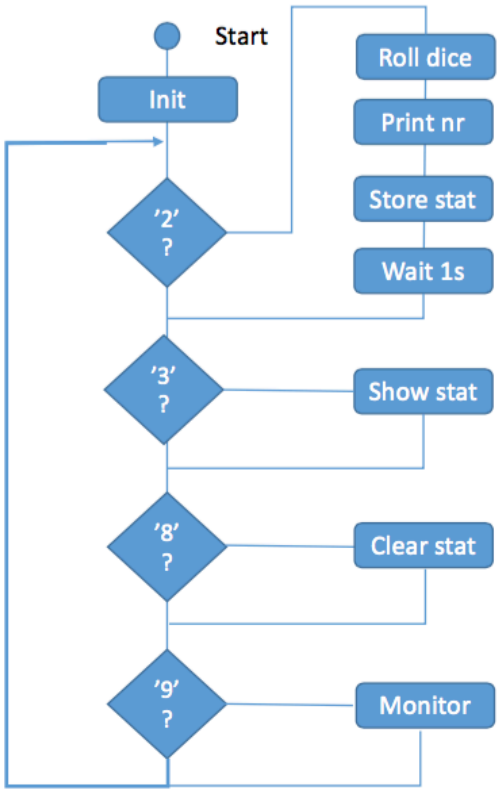
6 Moment 4: Programmering av tärningsapplikation

6.1 Programmering av applikation

Skapa en ny include-fil (tex Tarning.inc) och skriv programmet för applikationen, baserat på tärningsexemplet från Föreläsning 5. Lägg till lämpliga anrop från huvudprogrammet. Ni bör även lägga till andra saker, tex en välkomsttext och hantering av fler knappar, men funktionaliteten i aktivitetsdiagrammet nedan (den del som berör hantering av knapp ’2’ skall ni skriva själva, medan kod för ”Showstat”, ”ClearStat” och ”Monitor” finns färdiga i include-filerna Stats.inc och monitor.inc) skall finnas med. Titta gärna igenom include-filerna, för att se hur funktionaliteten där är implementerad.

Texterna kan vara tex: ”Welcome to dice! ” (tänk på att assemblern ger en varning om strängarna inte innehåller ett jämnt antal tecken – kan lösas genom att man lägger till: , 0 efter strängen …), vänta 1s och skriv sedan tex: ”Press ’2’ to roll”. När man trycker på 2’an bör ni skriva ut: ”Rolling…”. Efter att man släppt tangenten skriver ni tex ”Value:” och tärningens värde.

Lagra strängarna i programminnet och använd den MACRO ni definierade i uppg 5.2.7 för att skriva ut dem.



Figur 6-1 Aktivitetsdiagram för Tärningsprogram

Uppgift 6.4.1 (redovisas i programkod)

Skriv subrutinerna som behövs för kärnfunktionaliteten i Tärningsapplikationen (dvs anrop till subrutiner för avkänning av tangent och utskrift på LCD. Vänta med strängarna. Bygg och testa att det fungerar.

Tips: Koden för att implementera stegen i aktivitetsdiagrammet ovan, kan följa en av 2 principer:

1. Gör en liten loop, som känner av tangentbordet. För de olika knapparna, som styr funktioner i aktivitetsdiagrammet, hoppa (BRxx) framåt till en bit kod, som (via subrutinanrop) implementerar funktionen och sedan hoppar (RJMP) tillbaka till huvudloopen.
2. Gör en loop som ovan, men hoppa inte ut till separata kodstycken, utan vänd på villkoret, så att om knappen INTE är intryckt, så hoppar programmet över subrutinanropen, vidare till nästa test av knapptryckning.

Uppgift 6.4.2 (redovisas i programkod)
Inkludera filerna `stats.inc` och `monitor.inc` på lämpligt ställe i huvudprogrammet. Filerna hämtas från Canvas (Ni behöver kommentera bort anrop till rutiner som inte finns än. **Återställ efter moment 5 nedan).** Glöm inte att anropa `init_stat` och `init_monitor`!

Utöka koden så att strängar som behövs skrivs ut. Följ aktivitetsdiagrammet, men lägg till. Notera att rutinerna ”showstat”, ”clearstat” och ”monitor” innehåller egna strängar som skrivs ut och att RVAL förutsätts vara R24!. Bygg och testa att det fungerar.

7 Moment 5: Insamling och lagring av statistik

7.1 Beskrivning

I detta steg skall ni lägga till en datastruktur och subrutiner för att lagra och hämta ut statistik (som visar om tärningen är rättvis).

Uppgift 7.1.1 (redovisas i programkod)
Deklarera ett antal variabler **i dataminnet**. Följande värden kan vara intressanta:

- Plats för en räknare som visar antal kast.
- Plats för en räknare som visar antalet gånger siffran ’1’ har kommit upp.
- Plats för en räknare som visar antalet gånger siffran ’2’ har kommit upp.
- Plats för en räknare som visar antalet gånger siffran ’3’ har kommit upp.
- Plats för en räknare som visar antalet gånger siffran ’4’ har kommit upp.
- Plats för en räknare som visar antalet gånger siffran ’5’ har kommit upp.
- Plats för en räknare som visar antalet gånger siffran ’6’ har kommit upp.

Skapa en fil (tex **`stat_data.inc`**) som tex innehåller följande:

```
/* ------------------------------------------------

Space in the RAM to save the results from dice throws.

The following functions are provided:

            store_stat (R24 contains the dice value)

                        The function will increment the

                        Total number of throws and the

                        number of throws with results equals R24.
```

```
            get_stat (R24 gives the value for which the

                        result is retrieved. If R24 = 0, the

                        total number of throws shall be returned.

                        The result is returned in R24.

            clear_stat (no parameters in nor out)

                        Clears all data.

-------------------------------------------------------*/

            .DSEG                   ; The following applies to the RAM:

            .ORG        0x100       ; Set starting
                                    ; address of data
                                    ; segment to 0x100

<your_label>:           .BYTE       <n>         <skapa så mycket plats som behövs.
Antag att max värde per lagrad variabel är 255>

.CSEG

store_stat: <skriv kod för rutinen>

                        RET

get_stat:   <skriv kod för rutinen>

                        RET

clear_stat: <skriv kod för rutinen>

                        RET
```

Skriv sedan kod i huvudprogrammet så att `store_stat` anropas då programmet körs.

Tips: En array med 7 platser kan användas och värdet på ”kastet” kan användas som index för att komma åt ”rätt” räknare. **OBS! Register skall INTE användas** för att lagra data, utan minnesplatser skall allokeras, som visas ovan. Använd inte if-satser för access till data, utan använd en pekare, tex X-registret.

Uppgift 7.1.2 (redovisas i programkod)
Ändra nu i huvudprogrammet, så att alla delar av aktivitetsdiagrammet finns med, dvs man skall kunna trycka på olika knappar för att rulla tärningen, se resultat av statistik, rensa statistik och köra monitorn.

8 Moment 6: Test av komplett system, redovisning och reflektion

8.1 Beskrivning

I detta, sista steg skall ni kolla att programkoden fortfarande är väl strukturerad och att allt fungerar som det skall.

Uppgift 8.1.1 (redovisas i rapport)
Testkör programmet! Om det fungerar som det ska är det dags att undersöka om tärningen ger rättvisa värden:

- Tryck ’8’ (rensa resultat)
- Rulla tärningen ca 100 gånger (Håll inne 2’an så att ”Rolling…” hinner skrivas ut och dice-rutinen hinner köra en stund.
- Tryck ’3’ för att visa resultaten och skriv ner dessa i en tabell. Räkna även ut procentsatsen för varje värde, enligt exempel nedan. **Notera att siffrorna skrivs ut som hextal!**

Antal kast:	117 (0x75)	procent
Antal ’1’	24 (0x18)	20%
Antal ’2’	24 (0x18)	20%
Antal ’3’	16 (0x10)	14%
Antal ’4’	16 (0x10)	14%
Antal ’5’	15 (0x0F)	13%
Antal ’6’	22 (0x16)	19%

Uppgift 8.1.2 (redovisas i programkod)
Nu är det dags att avsluta genom att:

- Se till att ni förstår vad programmet gör i alla delar.
- Snygga till koden och kommentera programmet där det saknas kommentarer. Ni ska kunna återvända till koden om några månader och fortfarande förstå vad som sker!
- I kommentarsblocket i början av filerna ska ni skriva in era namn och aktuellt datum. Dessutom ska ni med egna ord beskriva vad programmet gör.

När detta är klart kan ni redovisa för labhandledare!

Uppgift 8.1.3 (redovisas i rapport)
Redogör för era erfarenheter från denna laboration. Vad har ni lärt er? Gick allting bra eller stötte ni på problem? Om allting gick bra, vad var i så fall anledningen detta? Om ni stötte på problem, hur löste ni i så fall dem?

Fördjupande uppgifter (görs om tid finns)

9 Moment 7: Förbättring av statistikpresentation och monitor

9.1 Beskrivning

I detta, extra steg skall ni granska koden för att visa statistiken och monitorn. Ni kan hitta på andra sätt att lösa uppgifterna (se nedan vad programmen gör) och sedan implementera/testa detta.

Uppgift 9.1.1 (redovisas i programkod)
Statistikpresentationen visar 7 värden i sekvens enligt följande:

- Skriv ut: ”Total throws:” och visar totalt antal kast (dvs anrop till `get_stat` med R24 = 0)
- Vänta 1 s
- Skriv ut: ”Nbr of 1’s:” och visar antal 1’or (dvs anrop till `get_stat` med R24 = 1)
- Vänta 1 s
    - *<samma sak för siffrorna 2-6>*
- Hoppa tillbaka till huvudprogrammet

Uppgift: Förbättra koden. Testa!

Uppgift 9.1.2 (redovisas i programkod)
Statistikrensningen visar texten ”Clearing…” och anropar sedan `clear_stat`. Sedan hoppa tillbaka till huvudprogrammet.

Förbättra koden. Testa!

Uppgift 9.1.3 (redovisas i programkod)
Monitor-programmet är ett litet hjälpprogram, som tillåter att man läser av värden i dataminnet. Då programmet startar, skrivs”>0000:xx” ut (där ”xx” är 2 hex-siffror för innehållet i cellen.

- Om man trycker på ’#’, visas innehållet i nästa minnescell. (Om man håller inne knappen, så räknar programmet upp räknaren och visar samtidigt innehållet i cellerna…
- Om man trycker på knapparna ’0’ – ’9’, så byts startadressen till motsvarande startaddress, där den höga adressen blir värdet på tangenten som trycks ner, tex >0100:xx då 1’an trycks ner. (på adress 0100 skall ju era data finnas….. Man kan på adress 0000 och framåt se innehållet i alla register!
- Om man trycker på ’*’, avbryts monitorn och man hoppar tillbaka till huvudprogrammet.

Uppgift: Förbättra koden. Testa!