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.
- Användning av rutiner som finns i andra filer.
- Access till olika typer av data som finns i programminnet.
- 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. 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 It’s Learning), samt i dokumentet *Allmänna instruktioner för laborationer*.

2.2.2 Praktisk och muntlig redovisning

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

- Uppkopplad krets. Kretsen ska vara snyggt kopplad. Dessutom ska kablarna ha korrekt och enhetlig färgkodning.
- 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 (fil för fil, INTE via Import-funktionen) ni hade i lab 2 (**xxxx.inc)** i ert projekt. 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!

Uppgift 3.2.4 (redovisas i programkod)

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

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/utparametrar 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.

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 R16, Z
```

Uppgift 4.2.4 (redovisas i rapport)
Förklara vad de rader vi lade till i koden i uppgift 4.2.3 gör.

5 Moment 3: Användning av strängar i programminnet

5.1 Beskrivning

Detta moment går ut på att ni skall lagra strängar i dataminnet 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:” *AVR assembler strings in program memory*” och beskriv kortfattat (kod behövs inte) hur man skulle kunna göra istället. (tips: C hanterar sina strängar så…) Även föreläsningsbilder kan vara intressanta.

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

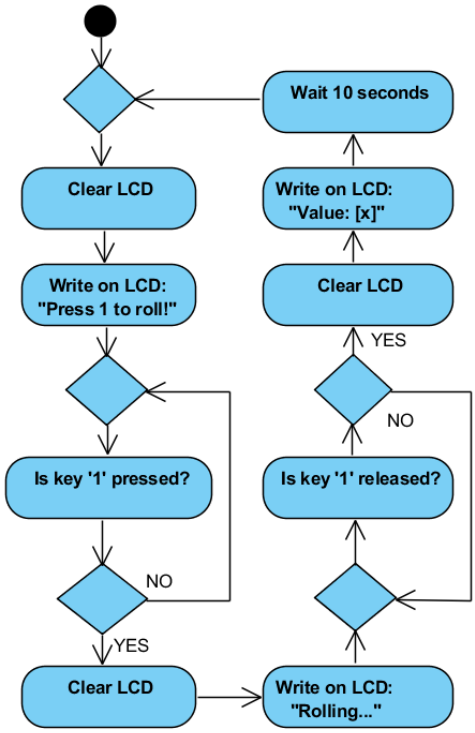
6 Moment 4: Programmering av tärningsapplikation

6.1 Programmering av applikation

Skapa en ny include-fil (tex Tarning.inc) och skriv programmet för applikationen, baserat på tärningsexemplet från Föreläsning 3 och aktivitetsdiagrammet i Figur 6-1 nedan. Lägg till lämpliga anrop från huvudprogrammet. Ni får även lägga till andra saker, tex en välkomsttext och hantering av fler knappar, men funktionaliteten i aktivitetsdiagrammet skall finnas med.

Notera att ni INTE skall låta tärningsrutinen känna av porten, som i exemplet på föreläsningen, utan **read_keyboard** skall användas istället! Notera också att ”[x]” i aktivitetsdiagrammet nedan skall tolkas som att det aktuella värdet från tärningsrutinen skall skrivas ut.

Ni skall också skriva ut lite olika strängar. Använd samma teknik som i Uppgift 5.2.1/5.2.2 för att lagra dessa i programminnet och 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.

Uppgift 6.4.2 (redovisas i programkod)
Utöka koden så att strängarna skrivs ut enligt aktivitetsdiagrammet. (Flytta först ”STR_1” – se Uppgift 5.2.1 och ”write_welcome” – se Uppgift 5.2.2. Gör sedan nya strängar och rutiner för att skriva ut dessa). Bygg och testa att det fungerar.

7 Moment 5: Test av komplett system, redovisning och reflektion

7.1 Beskrivning

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

Uppgift 7.2.8 (redovisas i programkod)

Testkör programmet! Om det fungerar som det ska 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 7.2.9 (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?