Laboration 4: Tangentbord och LCD i ASM

1 Inledning

Laborationen ska bidra till en grundläggande förståelse för:

- Hur man programmerar inbyggda system i Assembler.
- Hur man skriver sk ”drivers” för tangentbord och LCD
- Hur man kan skapa och använda enkla makrofunktioner i Assembler.
- Hur textsträngar är uppbyggda samt hur de kan hanteras i Assembler.

1.1 Utrustning

Alla komponenter som behövs ingår i lab-lådan.

2 Beskrivning av laboration

Den här laborationen utgör det första tillfället i kursen där ni ska programmera inbyggda system i Assembler. Laborationen innebär att skriva drivrutiner för LCD och tangentbord (väldigt likt det som gjordes i Lab 1 i C). Utöver detta tillkommer en del andra saker: hantering av pekare till strängar i programminnet.

Laborationen består av följande moment:

1. Sätt upp utvecklingsmiljön och exekvera grundläggande programkod
2. Fördröjningsrutiner och grundläggande MACROs
3. Drivrutin för LCD
4. Läsning av strängar från programminnet
5. Drivrutin för numeriskt tangentbord

I laborationens mapp på It’s Learning finns ett nästintill färdigt Atmel-projekt (**lab4.zip**). Projektet består av flera filer. Laborationens olika moment går ut på att ni ska komplettera programkoden på olika abstraktionsnivåer. I nästa lab skall ni bygga på dessa delar till ett litet spel, ”tärning”, som går ut på att implementera en tärning och se om den fungerar statistiskt korrekt.

2.1 Examination

2.1.1 Inlämning av rapport och programkod

Instruktioner för inlämning finns beskrivet på inlämningssidan för denna laboration (på It’s Learning).

2.1.2 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 programkod.

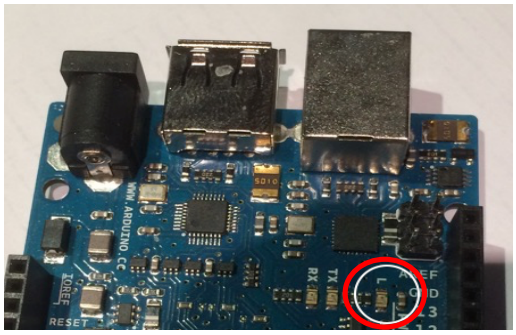
3 Moment 1: Sätt upp utvecklingsmiljön och exekvera grundläggande programkod

3.1 Första uppgift

Uppgift 3.1.1

1. Ladda ner **lab4.zip** från It's Learning och packa upp filerna på ett lämpligt ställe. I denna ZIP-fil finns en mapp med projektfiler och programkod.
2. Öppna sedan projektet i Atmel Studio.
3. Lyft försiktigt bort tangentbordet och LCD'n från arduino-kortet.
4. Assemblera (Bygg) programmet och för över det till mikrokontrollern.

Om allt fungerar som det ska, så bör lysdioderna till vänster om lysdiod L (märkta RX och TX) på Arduino-kortet blinka under en kort stund (se Figur 3-1). Sedan kommer den inbyggda lysdioden L lysa. Lysdioden styrs genom att konfigurera PB7 (PORTB, bit 7) som en utgång, samt genom att sätta hög nivå på denna utgång. Anslutningen PB7 är även sammankopplad med Arduino-kortets pinne 13. Denna information är dock inget man lär sig utantill. Därför behövs en tabell som anger *mappningen*, d.v.s. relationen mellan de interna och externa anslutningarna (finns på It's Learning och på produktens hemsida).



Figur 3-1: Inbyggd lysdiod L (markerad med röd/vit ring)

3.2 Analys av grundläggande programkod

När man lär sig att programmera behöver man inte bara skriva kod, utan också läsa och förstå kod som redan är skriven. Till att börja med ska ni analysera den grundläggande programkoden som redan är given. Koden finns i filen **lab4.asm**.

Uppgift 3.2.1 (redovisas i rapport)

Några av raderna, i blocken som kallas för " Definitions of registers, etc. ("constants")" och "Start of program" i kommentarerna, ser ut så här:

```
.EQU      RESET     = 0x0000  ; reset vector
.EQU      PM_START  = 0x0072  ; start of program

.CSEG
.ORG RESET
RJMP init
.ORG PM_START
.INCLUDE            "delay.inc"
//.INCLUDE          "keyboard.inc"      ; avkommenteras i moment 5
//.INCLUDE          "LCD.inc"           ; avkommenteras i moment 3
```

Följande frågor ska besvaras:

- Vad har **.EQU** för innebörd?
- Vad har **.CSEG** för innebörd?

- Vad händer om man skriver **.ORG**? Vad menas det som skrivs till höger?
- Vad sker när raden (**RJMP init**) exekveras?

Uppgift 3.2.2 (redovisas i rapport)
Nästa kodblock inleds med att konfigurera *stackpekaren*. Efter detta anropas ett antal subrutiner som hanterar konfigurering (även kallad initialisering) av in- och utgångar. Bla anropas subrutinen (**init_pins_led**) .Koden för subrutinen ser ut så här:

```
init_pins_led:
          SBI DDRB, 7
          RET
```

Följande frågor ska besvaras:

- Vad har instruktionen **SBI** för syfte?
- Vad är **DDRB**?
- Vad har instruktionen **RET** för syfte?

Uppgift 3.2.3 (redovisas i rapport)
Huvuddelen av programmet exekveras som en oändlig slinga (*infinite loop*). Det enda som utförs, förutom hoppet tillbaka till ”main”, är exekveringen av koden nedan. Vad sker när den exekveras?

```
SBI PORTB, 7
```

Uppgift 3.2.4 (redovisas i rapport)
I den utkommenterade koden finns en liknande instruktion. Vad sker när denna exekveras?

```
//CBI PORTB, 7      ; avkommenteras i Moment 2
```

Uppgift 3.2.5 (redovisas i rapport)
I den utkommenterade koden finns även ett antal förekomster av instruktionen nedan. Vad sker när denna exekveras?

```
NOP
```

3.3 Exekvering av grundläggande programkod

För att kunna se vad som händer med programmet, laddar vi bara ner det till Arduino-kortet och ser att den processen ser ut att fungera.

Uppgift 3.2.1
Bygg systemet och ladda ner det till Arduino-kortet. Nu bör L lysa. För att kunna få lysdioden att blinka, så vi hinner se det, behöver vi lämpliga fördröjningsrutiner. Dessa skrivs i nästa moment.

4 Moment 2: Fördröjningsrutiner och grundläggande MACROs

4.1 Beskrivning

Fördröjningsrutiner är viktiga i system som inte använder s.k. timers. Fördröjningsrutiner är kod som bara skall – ta tid! Ni får koden till den första funktionen, men skall skriva kod till de övriga. Ett tips är att låta de olika rutinerna bygga på varandra (tänk dock på att ett 8-bitars tal inte kan vara större än 255!).

Några MACROs används både i LCD-rutinerna och är bra att ha till tangentbordsrutinen. Därför behöver dessa göras nu.

4.2 Implementering av fördröjningsfunktioner och MACRO

Uppgift 4.2.1 (redovisas i programkod)

I projektet finns en befintlig fil vid namn **delay.inc** I denna fil ska ni implementera era fördröjningsrutiner. Alla rutinerna (utom den första) har en parameter, som talar om hur många mikrosekunder/millisekunder/sekunder som rutinen skall fördröja. Tänk på att in-parametern har 8 bitar (största talet som kan hanteras är 255) och att era rutiner också bör använda variabler som är 8 bitar (samma begränsning!). Det är viktigt att subrutinerna har följande namn:

- **delay_1_micros**
- **delay_micros**
- **delay_ms**
- **delay_s**

Uppgift 4.2.2 (redovisas i programkod)

Gå nu tillbaka till huvudprogrammet, avkommentera raden `//CBI PORTB, 7` och ersätt NOP-instruktionerna med lämpligt antal anrop till delay_ms (med lämplig parameter i ”rätt” register). Tänk på att ni måste låta tiden gå både med lysdioden tänd och med lysdioden släckt (helst lika länge tänd som släckt)!

Försök få L att blinka med periodtiden 1s (dvs ½ s tänd och ½ s släckt). Justera delayrutinerna, så att det blir så rätt som möjligt. Prova tex att räkna 20 blinkningar och mät tiden – bör bli ca 20 s.

Uppgift 4.2.3 (redovisas i programkod)

Implementera nu de 2 MACROs som finns i huvudprogrammet: SET_IO_BIT och CLR_IO_BIT, så att de gör det som står i kommentarblocket ovanför.

Tips: använd ”metod 2” enligt bild 10 i Föreläsning 8. Exempel på hur MACRO görs med parametrar, finns i bild 29.

Uppgift 4.2.4 (redovisas i programkod)

Avsluta redigeringen genom att:

Redogöra syftet med filen och vad den innehåller för funktionalitet i kommentarsblocket i början av filen. Glöm inte att ange era namn och datum!

5 Moment 3: Drivrutin för LCD

5.1 Beskrivning

Om fördröjningsrutinerna (i moment 2) gjorts på rätt sätt, så bör displayen nu kunna fungera. För att kunna skriva ut strängar och för att kunna rensa skärmen måste dock några funktioner färdigställas. Sedan är det dags att kontrollera att hela drivrutinen för displayen fungerar! Lämpligtvis gör ni tester på anvisad plats i **lab4.asm**. Testerna kan bestå av att genomföra vissa operationer, exempelvis kan ni skriva ut några tecken, pausa en stund, rensa skärmen och skriva nya tecken i en oändlig loop. I takt med att ni kompletterar drivrutiner enligt nedanstående uppgifter, så bör ni också utöka era tester.

5.2 Grundläggande tester och komplettering av drivrutin

Uppgift 5.2.1 (redovisas i programkod)

Inkludera **lcd.inc** i ert projekt (finns på It's learning). Kom även ihåg att anpassa initrutinen i lab4.asm för detta, dvs avkommentera anropet till lcd_init efter anropet till init_pins_led.

Ändra koden i början av huvudprogrammet:

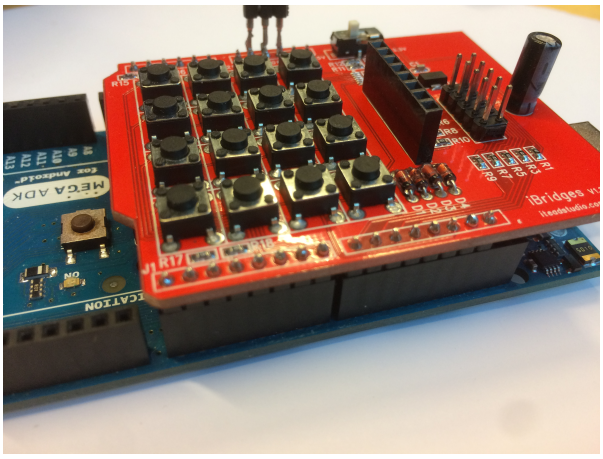
```
main:
            LCD_WRITE_CMD       0x80        ; Set position: col 0
            LCD_WRITE_CMD       0x40        ; row 0
            LDI         R24,    'H'
            LCD_WRITE_CHR
            LDI         R24,    'E'
            LCD_WRITE_CHR
            LDI         R24,    'L'
            LCD_WRITE_CHR
            LDI         R24,    'L'
            LCD_WRITE_CHR
            LDI         R24,    'O'
            LCD_WRITE_CHR
            LCD_WRITE_CMD       0x80        ; Set position: col 0
            LCD_WRITE_CMD       0x41        ; row 1

main_loop:
            // RCALL read_keyboard_num    ;avkommenteras efter moment 4
```

I slutet på huvudprogrammet, ändra även hoppet tillbaka till 'main', så att ni istället hoppar tillbaka till "main_loop", dvs:

```
            RJMP main_loop
```

Uppgift 5.2.2

Sätt nu försiktigt tillbaka tangentbordet och LCD'n på Arduino-kortet. Se bilden nedan hur tangentbordet skall sitta:



Figur 2: Placering av tangentbord/LCD på Arduinokortet

Bygg systemet och ladda ner till Arduino-kortet. Nu bör displayen komma igång och texten ”HELLO” bör visas på skärmen. Om kontrasten inte är bra, kan denna ändras i rutinen ’init_lcd’ (parametern vop – prova ett större tal för mer kontrast)

Uppgift 5.2.3 (redovisas i rapport)
Fotografera displayen, som visar texten ”HELLO” enligt specifikationen ovan.

Uppgift 5.2.4 (redovisas i programkod)
I **lcd.inc** finns en funktion **lcd_clear** som saknar innehåll. Se till att denna fungerar som avsett (raderar skärmen helt och ställer cursorn på första positionen). Testa funktionen (tex genom att skriva ut lite text, rensa skärmen och sedan skriva lite text igen – den sista texten skall då komma längst upp till vänster på skärmen)!

6 Moment 4: Läsning av strängar från programminnet

6.1 Beskrivning

Som vi gått igenom på föreläsning, så kan man lagra data i programminnet, och från programmet komma åt denna data via instruktionen LPM. Instruktionen kräver dock att Z-registret är konfigurerat på rätt sätt för att det skall fungera. Detta moment tränar detta.

Uppgift 6.1.1 (redovisas i programkod)
I **lcd.inc** finns en funktion för att skriva ut strängar: **lcd_write_string**. I nuvarande utförande kräver den att man sätter Z-pekaren till att peka på den sträng som skall skrivas ut, innan rutinen anropas. Skapa en sträng i huvudprogrammet (tex ”PRESS KEY:” som behövs i sista momentet) och skriv kod (i huvudprogrammet), som sätter Z-pekaren och sedan anropar subrutinen **lcd_write_string**. När ni är klara bör ni testa funktionen!

Uppgift 6.1.2 (redovisas i rapport)
Beskriv hur funktionen **lcd_write_str** fungerar, genom att referera till hur man använder pekaren för att stega igenom teckensträngen. Varför behöver Z-pekaren sparas undan innan man skriver ut varje tecken?

Uppgift 6.1.3 (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. I **lcd.inc** finns redan ett ofullständig MACRO ”PRINTSTRING” som tar en parameter (strängnamnet) och som ska ladda ZL och ZH och sedan anropa subrutinen **lcd_write_string** (dvs det ni gjorde i uppg 6.1.1). Komplettera MACRO’t så att det fungerar. Man skall alltså nu kunna skriva ut strängar i programmet på detta sätt:

```
Hello_str1:     .DB ”Hejsan”,0,0            ; definition av text. Extra 0’a för ”padding”

…

PRINTSTRING Hello_str                       ; Skriver ”Hejsan” på displayen
```

Ändra nu i huvudprogrammet (ta bort det ni skrev i 6.1.1 ovan) och använd istället det nya MACRO’t för att skriva ut strängen. Prova att det fungerar med olika strängar! Ni kan få varningar om ”padding” vid assemblering. Justera då texten till att innehålla ett ojämnt antal tecken (jämnt med 0-an), eller lägg en extra 0 efter (som i exemplet ovan).

Uppgift 6.1.4 (redovisas i programkod)
Komplettera kommentarsblocket i början av **lcd.inc** genom att ange era namn. *OBS! Det hör till god ton att låta den ursprungliga programmerarens namn vara kvar!*

7 Moment 5: Drivrutin för numeriskt tangentbord

7.1 Beskrivning

Drivrutinen är till viss del redan given i filen **keyboard.inc**, men ni måste komplettera den. När ni är klara bör ni utföra några tester. Dessa utför ni på anvisad plats i filen **lab4.asm**, lämpligtvis tillsammans med drivrutinerna för displayen. På så sätt kan ni lätt få reda på om ni har tryckt på en tangent samt vilken tangent det faktiskt var.

7.2 Första version av drivrutin

Uppgift 7.2.1 (redovisas i programkod)
Komplettera funktionen **read_keyboard_num** så att den kan returnera en binär siffra, som motsvarar tangenten.

Ni ska stega igenom samtliga kolumner och rader (totalt 16 steg) genom att stega igenom de 4 kolumnerna. För varje kolumn skall ni avgöra om någon av raderna är aktiverade (vilket är tecken på att motsvarande tangent är nertryckt). Implementera tex följande algoritm:

```
Sätt räknare = 0;
För varje kolumn gör
    Sätt kolumnen hög (och övriga kolumner låga)
    För varje rad gör
        öka räknare med 1
        Känn av värdet på raden
        om tangent är nertryckt
            hoppa fram till slutet och returnera räknare där
returnera räknare
```

Tips1: Eftersom vi bara gör 16 steg, kan det vara enklast att lägga ut koden linjärt (dvs INTE loopa) vilket gör det tydligare. När en tangent detekteras, hoppar man fram till slutet på subrutinen (om ingen tangent detekteras bör man komma fram till samma punkt och därifrån returnera).

Tips2: Glöm inte att sätta tillbaka värdet till låg på de kolumner som inte används.

Tips3: Vänd logiken och hoppa tex över ”framåthoppet” genom att använda SBIC.

Tips4: se till att räknaren når 16 (som ju betyder NO_KEY) om ni inte får träff på någon tangent.

Uppgift 7.2.2 (redovisas i programkod)
Komplettera huvudprogrammet så att strängen ”PRESS KEY:” skrivs ut på första raden och att sedan alla tecken som representerar nertryckt tangent skrivs ut på nästa rad på LCD’n. Tänk på att först konvertera siffran (som kommer från tangentbordet) till ASCII, för att kunna skriva ut den. Det gör inget just nu att siffran inte stämmer med tangenten eller att de sista tangenterna ger konstiga tecken. Gör en enkel konvertering genom att addera 0x30 till siffran för att få den i ASCII.

Tänk också på att hantera returvärdet ”NO_KEY” (det skall förstås testas/verifieras att värdet kommer, men skall normalt inte skrivas ut…) och att man inte skall repetera samma tangent om den hålls nertryckt länge (använd INTE delay!)…..

Uppgift 7.2.3 (redovisas i programkod)
Testa att ert program, förutom att man skall kunna skriva ut alla tecken, även kan skriva ut samma tecken flera gånger (avsiktligt – om ni har problem med knappstuds, så öka delay i read_keyboard_num) och att man skall kunna först trycka en tangent (tex 7), sedan trycka en annan (tex 3) –utan att släppa den första. Slutligen släpper man den första tangenten. Bägge siffrorna skall skrivas ut!

7.3 Komplettering av drivrutin

Uppgift 7.3.1 (redovisas i programkod)
I filen ’keyboard.inc’, ändra strängen ’keys’ (som finns i början på filen keyboard.inc) och läs/analysera koden i rutinen **’convert_to_ASCII’** som översätter från ett nummer (0-15, eller 16 då inget tecken trycks in) till att returnera motsvarande ASCII-tecken (’0’ – ’D’ och NO_KEY om inget tecken tryckts. (XXXXXXXXXXXXXXXX skall alltså bytas ut mot 16 lämpliga ASCII-tecken i rätt ordning)

Uppdatera huvudprogrammet, så att **’convert_to_ASCII’** anropas efter ’read_keyboard_num’ och resultatet sedan skickas till ’lcd_write_char’ (på samma sätt som i macro’t LCD_WRITE_CHR).

Uppgift 7.3.2 (redovisas i programkod)
Bygg systemet, ladda ner och provkör. Nu bör siffrorna, *, # och ABCD visas korrekt på displayen.

Uppgift 7.3.3 (redovisas i programkod)
Testa att ert program på samma sätt som i uppgift 7.2.3.

Uppgift 7.3.4 (redovisas i programkod)
Redovisa ert bidrag i koden genom att komplettera kommentarsblocket i början av filen. Därefter kan ni redovisa för labhandledare.

Uppgift 7.3.5 (redovisas i rapport)
Redogör för era erfarenheter och kunskaper från denna laboration (minst en halv A4-sida):

- Vad har ni lärt er?
- Om ni får välja en sak, upplevde ni något som var intressant/givande?
- Fanns det något som upplevdes som svårt?
- 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?