Namn:
Laborationen godkänd:

Digitala system 15 hp

LUNDS TEKNISKA HÖGSKOLA
Lunds universitet
LTH Ingenjörshögskolan vid Campus Helsingborg

Datorteknik 2 (AVR 2)

Enkel in- och utmatning. Drivrutiner. Bithantering.

I denna laboration ska vi förbättra kommunikationen med omvärlden och lära oss hur man testar bitar och påverkar bitar i en variabel.

Förberedelser:

Läs igenom laborationen. Komplettera tabellen i uppgift 1 och gör ett lösningsförslag på uttrycket till while-satsen i uppgift 3.

Laborationsuppgifter.

Uppgift 1. Tabell för de olika siffrorna.

Ta fram förteckning över vilka bitar i PORTA som var anslutna till vart och ett av de sju segmenten på displayen (Laboration 1). Gör en tabell som innehåller alla siffrors utseenden. Med programmeringsspråk kallas en sådan tabell ofta ”array” eller ”fält”.
En tabell är en samling värden som har numrerade platser. Ibland kan tabeller innehålla både kolumner och rader, men vår tabell har bara en kolumn och man kan alltså nå varje plats med ett enda nummer.
Tabellens namn är ”**`segment_table`**”, men skulle kunna heta vadsomhelst.
”**`const unsigned char`**” anger att tabellvärdena ska vara 8-bitarstal som inte ska kunna ändras av programmet. Vi väljer att skriva binärt med ”**`0b`**” eftersom det blir tydligare i det här fallet (vi ser precis hur bitar är satta). Observera att den vänstraste biten (bit 7) alltid ska vara ett eftersom den ser till att knappens pullup-motstånd är inkopplat.

Erik Larsson

- Skriv in tabellen och det övriga skalet:

```
#include <avr/io.h>

const unsigned char segment_table [] = { 0b11111101,    //"0"
                                         0b11001000,    //"1"
                                         0b1       ,
                                         0b1       ,
                                         0b1       ,
                                         0b1       ,
                                         0b1       ,
                                         0b1       ,
                                         0b1       ,
                                         0b1       , };

unsigned char n;

int main(void)
{
    DDRA = 0b01111111;      // Utgångar till display
    PORTA = 0b10000000;     // Aktivera pull-up

    while (1)
    {
        n=(n+1);                    // Öka n med ett
        if (n==10) n=0;             // nollställ om n blir =10
        PORTA = segment_table[n]; // Skriv ut med tabellen
    }
}
```

Huvudprogrammet består nu av en oändlig slinga som skriver ut tabellvärdena för index 0 t.o.m. 9. Variabeln n som är den siffra som ska skrivas ut, är alltså index till den plats i tabellen där siffrans segmentmönster finns.

- Provkör programmet med AutoStep. **Hur ser det ut på displayen?**

.....

- Kommentera bort raden `if (n==10) n=0;` (sätt // längst till vänster) och kör Autostep igen. **Vad skrivs ut på displayen?**

.....

- Återställ programmet och kör i full fart! **Hur ser det ut nu?**

.....

❑

Uppgift 2. Drivrutiner.

Den som skapar själva applikationen (programmet som gör det intressanta arbetet), vill inte vara tvungen att hålla reda på detaljer i hårdvaran som inte har med uppgiften att göra.

Exempelvis ska man inte behöva hålla reda på exakt hur display och knapp är inkopplad. Om man vill skriva en siffra ska man bara behöva skriva t.ex.

`display(5);` om man vill skriva ut en femma

eller

`display(n);` om man vill skriva ut värdet av variabeln n

Vi ska nu skriva en funktion (drivrutin) som sköter detta åt oss:

```
// Funktion display:
// skriver ut en decimal siffra på displayen.

void display (unsigned char number)
{
     PORTA = segment_table [number];
}
```

- Skriv in funktionen!

- Ändra nu i huvudprogramslingan så att den anropar funktionen istället för att skriva direkt till displayen.

Observera fördelen med att använda en rutin för kommunikation med displayen. I din PC används oerhört mycket mer komplicerade drivrutiner (för tangentbord, bildskärm, nätverk m.m.) men principen är densamma. Programmeraren behöver inte bry sig om hur allt detta fungerar egentligen.

❑

Uppgift 3. Tärning

Lägg till en sats i programslingan så att ***programmet stoppas när knappen är uppe***. När knappen hålls ner kommer programmet att snurra en massa gånger och man får ett ganska slumpmässigt resultat på displayen.
Uppgiften kan formuleras:

```
så länge som (knappen är uppe) gör inget;
```

eller

```
while (knappen uppe){ };
```

- Bestäm vad som ska stå inuti parentesen (dvs. vad ska skrivas i stället för ”knappen uppe”)?

För att komma fram till C kod kan man först förklara problemet på svenska och sedan stegvis omformulera det till C. Skriv in satsen och testa!

❑

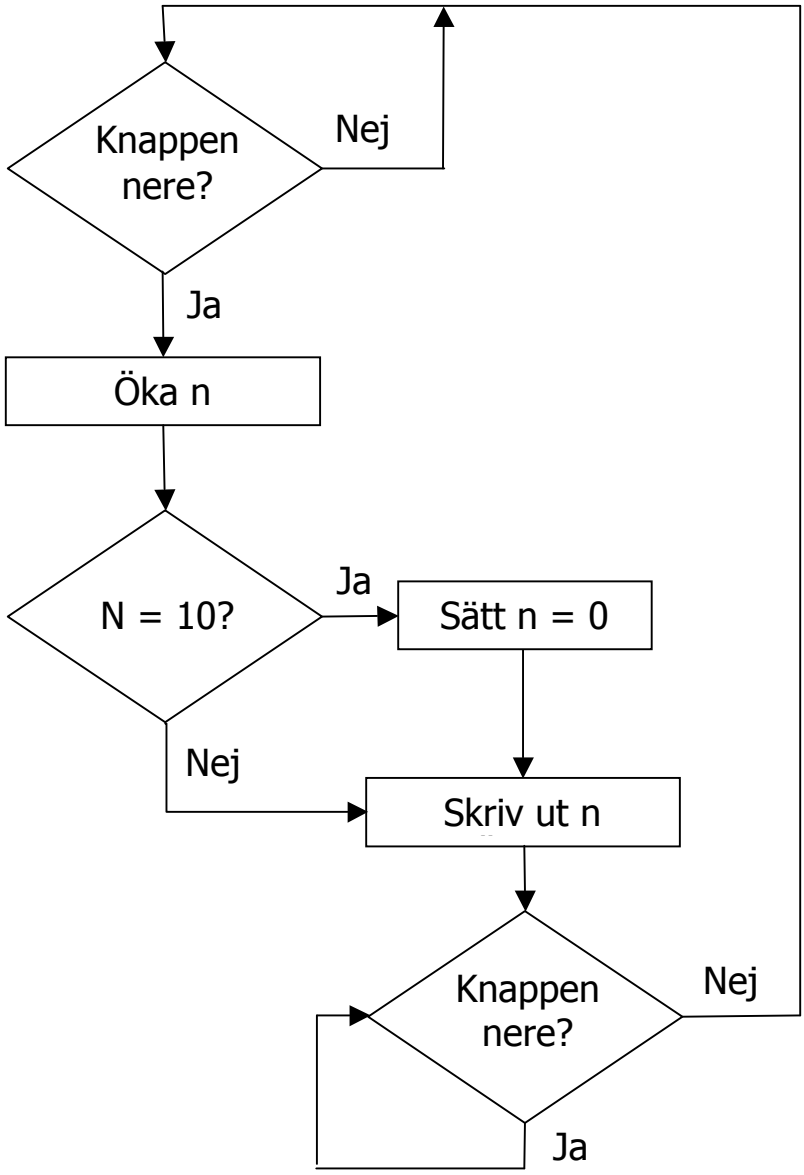
Uppgift 4. Vanlig 6-sidig tärning

Ändra ditt program så du får en vanlig 6-sidig tärning. Den ska alltså visa siffrorna 1, 2, 3, 4, 5 eller 6. Testa.

❑

Uppgift 5. Avkänna enskilda knapptryckningar.

Uppgiften är att räkna upp displayens värde en enda gång för varje gång man trycker ner knappen. Vi beskriver först programmets uppgifter i en flödesplan:



1. Vänta här tills någon trycker på knappen
2. Öka n med ett.
3. Om n blir 10, sätt n till noll.
4. Anropa utskriftsfunktionen med n som inparameter.
5. Vänta här tills knappen är släppt.
6. Gå till punkt 1.

- Punkterna 1 – 4 är avklarade i föregående uppgift, så du bör kunna klara av

punkt 5 på egen hand:

.....

- Testa programmet. Fungerar det bra?
  Räknas värdet upp med ett för varje knappnedtryckning? Vad beror det på?

.....

❑

Uppgift 6. Fördröjningsslinga.

Problemet med tangentstudsarna kan hanteras med en vänteslinga.

- Deklarera ett 16-bitars positivt tal (k) och stoppa in for-satsen nedan på de ställen som behövs.

```
for (k=0; k<10; k++);
```

Fungerar det bra nu? .....

- Vad händer om du gör for-slingan bara en gång, dvs. skriver k<1?

.....

- Vad händer om du skriver k<60000 istället?

.....

- Välj ett värde att testa med som

1. är tillräckligt stort för en absolut säker funktion.
2. är tillräckligt litet för att det inte ska kännas ”segt”.

Mitt värde: .....(antal loopar)

❑

För att denna for-loop ska fungera, så måste kompileringen göras utan någon som helst optimering.

- Prova med att ändra optimeringen till -O1. Ladda ner och kör i full fart!

Kompilatorn tycker att denna for-loop är helt meningslös och struntar i att skapa kod för den. Det är inte bra. Om vi vill tvinga kompilatorn att **inte** ignorera variabeln k, så **måste** den deklareras såhär:

```
volatile unsigned int k;
```

Uppgift 7. Beräkning av tid.

Vi ska ta reda på hur lång tid vår fördröjning verkligen tar.
Öppna fönstret ”Disassembler” under menyn View eller med knappen  . Sätt igång programmet på autostep och se efter hur många instruktioner (rader) som pilen vandrar igenom när fördröjningsslingan genomlöps.
Man kan räkna med en mikrosekund per rad.

- Hur lång tid tar ***ett varv*** i din vänteloop?

.....

- Hur lång tid (ungefär) kommer din väntetid att vara?

.....

❑

Om du använder simulatorn i stället för att ladda ner koden i labbutrustrningen, så kan du använda det som kallas ”Stop Watch” och lätt se hur lång tid olika programdelar tar.

Sätt en brytpunkt framför for-loopen och en framför satsen strax efter.

- Hur lång tid tar din väntetid med denna mätning?

.....

❑

Uppgift 8. Privatuppgiften.

Du ska nu testa den uppgift du fick i början av laborationen. Eftersom ni laborerar två och två, så ska ni testa bådas uppgifter.

❑