INFÖR LABORATION 4

Följande uppgifter ur kursboken ska vara utförda som förberedelse för laborationen. Du ska på begäran av laborationshandledare redogöra för dessa.

Uppgifter i kursboken	
(9.1) (9.2) (9.13) (9.15) (9.16) 9.18	9.20

Uppgifter inom parentes är inte strikt obligatoriska, men om man gjort dem blir arbetet med de obligatoriska uppgifterna lättare.

Nedanstående hemuppgifter i detta PM som ska vara utförda innan laborationen påbörjas.

Hemuppgifter		
4.A	4.B	4.C

LABORATION 4

BESKRIVNING AV LABORATIONSSYSTEMET

Laborationssystemets panel är indelad i olika sektioner:

- **Registers**, visar innehållet i FLIS-processorns register.
- **Input/Output,** anslutningar av externa enheter till FLIS-processorn, här visas också de värden som för tillfället finns hos portarna.
- **Address**

  auto — minnesenhetens adressväljare (Memory address) följer programräknaren PC.

  manual — minnesenhetens adressväljare sätts med hjälp av omkopplarna A7-A4, de fyra mest signifikanta adressbitarna och A3-A0, de fyra minst signifikanta adressbitarna.

- **Memory**

  display — innehållet i minnesenhetens dataindikator (Memory data) ges av minnesenhetens adressväljare

  modify — minnesenhetens dataindikator sätts med hjälp av omkopplarna D7-D4/D3-D0

  set — då omkopplaren står i modify-läge förs innehållet i dataindikatorn in i minnet på den adress som anges av adressväljaren.

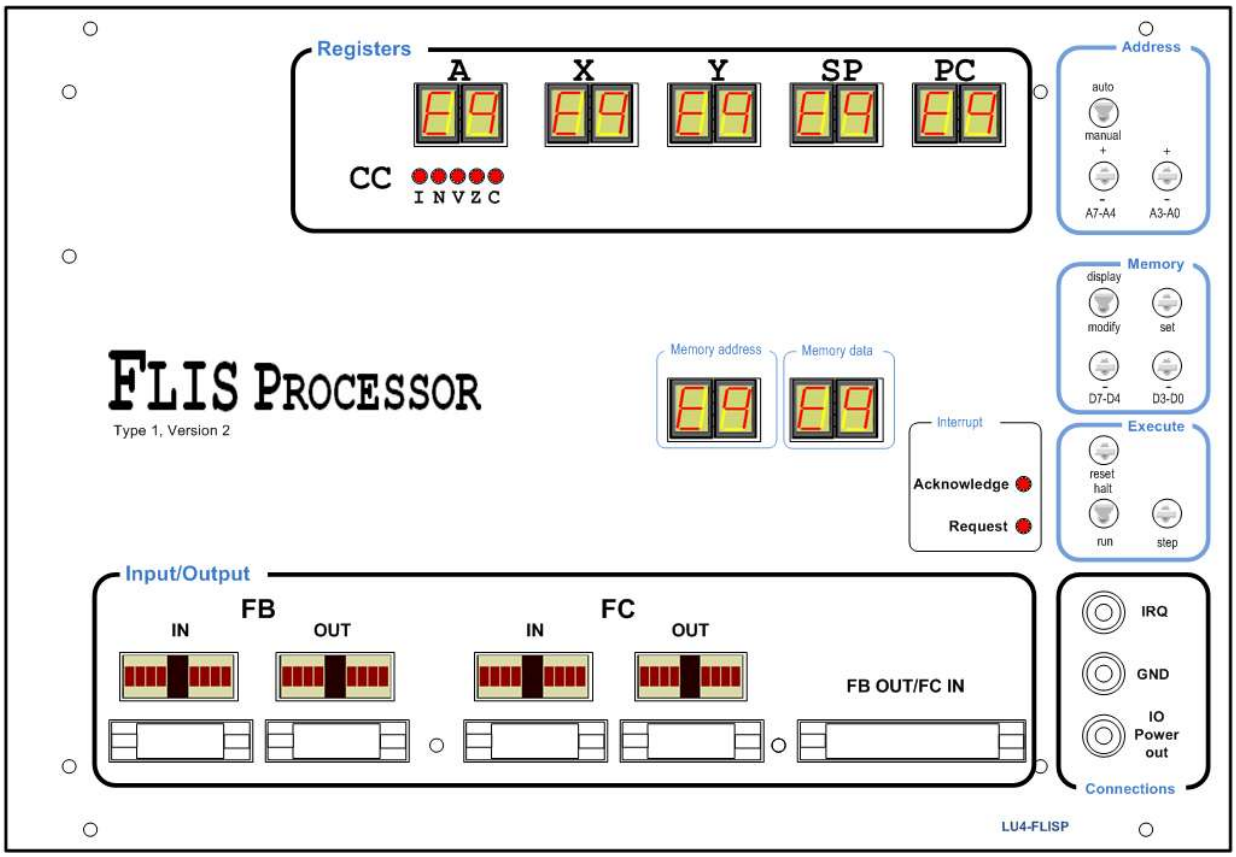
- **Execute**, här manövreras FLIS-processorn.

  reset — processorn utför ett återstartsförlopp.

  halt — I detta läge kan ett program utföras instruktionsvis med omkopplaren step.

  run — i detta läge exekverar processorn instruktioner kontinuerligt och exekveringshastigheten (tre olika) kan väljas med omkopplaren step.

- **Interrupt**, indikerar om en avbrottsbegäran (Request) finns på FLIS-processorns avbrottsingång. Dessutom indikeras (Acknowledge) då FLIS-processorn utför avbrottshantering.
- **Connections**, anslutningar, IRQ är direkt kopplad till FLIS-processorns avbrottsingång. IO Power out och GND kan användas för att strömförsörja yttre enheter.



NEDLADDNING AV PROGRAM

Programmet DigiFlisp har en funktion som kan användas för att ladda ner dina program (’`.fmem`’-filer) till laborationsdatorn via en USB-anslutning.

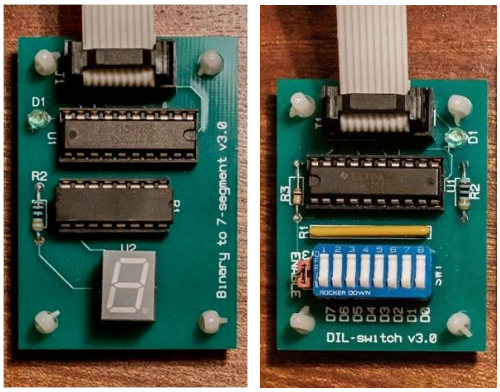
- Starta DigiFlisp och välj Console-fliken längst ner i fönstret. Ange den port som laborationsdatorn är ansluten till, i textfältet till höger om Connect. Porten anges på formatet ’**`\\.\COMxx`**’, där ’**`xx`**’ är numret på enheten ’USB Serial Port’ (enligt Windows-programmet *Device Manager*).
- Klicka på Connect för att ansluta till laborationsdatorn. Console-delen av fönstret byter färg till blått när anslutningen lyckats.
- Högerklicka i det blå fönstret, välj Load, och ange den ’`.fmem`’-fil du vill ladda ner till laborationsdatorn.
- Om du vill ladda samma ’`.fmem`’-fil flera gånger kan du högerklicka i det blå fönstret och välja Reload.

LABORATIONSUPPGIFT 4.1
SJUSEGMENTSINDIKATOR

I denna uppgift ska du testa din lösning på uppgift 9.18 (`DisplaySegE`), i kursboken. Du ska tidigare ha provat den i simulator så du vet att ditt program beter sig som det ska.

Du måste dock göra ett litet tillägg till din tidigare lösning: för att programräknaren (PC) i laborationsdatorn ska få rätt värde (20_{16}) från början ska du lägga till assemblerdirektiv som placerar programmets startadress i RESET-vektorn.

- Anslut modulen Binary-to-7-segment (sjusegmentindikator) till FLIS-processorns **FB OUT** med en 10-polig flatkabel.
- Anslut modulen DIL-switch till FLIS-processorns **FC IN** med en 10-polig flatkabel.



- Assemblera `DisplaySegE.sflisp`; det ska inte finnas några felmeddelanden.
- Kontrollera att laborationsdatorn står i läge halt.
- Ladda ner filen `DisplaySegE.fmem` till laborationsdatorn.
- Gör reset på laborationsdatorn. Kontrollera att programmets startadress nu finns i PC.
- Öppna listfilen (`DisplaySegE.lst`) i texteditorn.
- Sätt visningsadressen (Address) på laborationsdatorn i läge auto.
- Tryck en gång på step-omkopplaren och observera hur PC och Memory address uppdateras med programmets startadress.
- Läs av Memory data och se vad som finns i minnet på denna adress.
- Fortsätt med att utföra programmet instruktionsvis (step), följ med i listfilen, gör detta ett helt varv i programmet, dvs. tills PC på nytt får värdet 22_{16}.
- Starta exekvering av programmet genom att ställa omkopplaren i läge run.
- Öka exekveringshastigheten successivt genom att trycka på step-omkopplaren.

SJÄLVTESTNING

Innan ni tillkallar handledare är det viktigt att ni först själva testar så att programmet fungerar som det är tänkt. Diskutera några tester ni kan göra (skriv gärna ned dem) och utför dem sedan för att se så att er lösning är korrekt. Om något inte fungerar som ni testat, diskutera först med varandra om vad som kan vara orsaken och försök sedan att åtgärda detta innan ni testar igen.

Se nedan för några exempel på tester:

- Stega igenom 1 till 9 och se till att de visas på sjusegmentsindikator.
- Kontrollera även lite andra värden som ger "E", extremvärden som 255 är en bra början.

Då programmet fungerar som det ska tillkallar du handledare för att redovisa lösningen på laborationsuppgiften.

LABORATIONSUPPGIFT 4.2
REALTIDSEGENSKAPER, RINNANDE LJUS

Inför denna uppgift ska du ha utfört och testat uppgift 9.20 (`RunDiodeDelay`) i kursboken. Du ska alltså tidigare ha provat den i simulator så du vet att ditt program beter sig som det ska.

OBS: I simulatorn har du använt värdet 255 som "fördröjningskonstant" men det är allt för stort för laborationsdatorn. Använd i stället värdet 10.

Du måste även här lägga till assemblerdirektiv som placerar programmets startadress i RESET-vektorn för att laborationsdatorn ska starta korrekt.

- Anslut modulen Bargraph (ljusdiodramp) till FLIS-processorns **FB OUT** med en 10-polig flatkabel.

- Modulen DIL-switch kan du lämna ansluten till **FC IN**. Du skall använda den i nästa uppgift.
- Assemblera `RunDiodeDelay.sflisp`; det ska inte finnas några felmeddelanden.
- Kontrollera att laborationsdatorn står i läge halt.
- Ladda ner filen `RunDiodeDelay.fmem` till laborationsdatorn.
- Gör reset på laborationsdatorn. Kontrollera att programmets startadress nu finns i PC.
- Starta programmet genom att ställa omkopplare i run-läge.
- Variera laborationsdatorns exekveringshastighet och observera skillnader hos ljusdiodrampen.

SJÄLVTESTNING

Innan ni tillkallar handledare är det viktigt att ni först själva testar så att programmet fungerar som det är tänkt. Diskutera några tester ni kan göra(skriv gärna ned dem) och utför dem sedan för att se så att er lösning är korrekt. Om något inte fungerar som ni testat, diskutera först med varandra om vad som kan vara orsaken och försök sedan att åtgärda detta innan ni testar igen.

Se nedan för några exempel på tester:

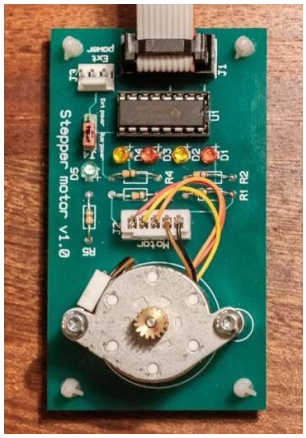
- Kontroller att det som mest finns en lysande lampa som rinner ned för ljusrampen.
- Se även till att ljuset "faller ut" från ljusrampen så att ingen lampa lyser, för att sedan dyka upp igen på andra sidan.

Då programmet fungerar som det ska tillkallar du handledare för att redovisa lösningen på laborationsuppgiften.

LABORATIONSUPPGIFT 4.3
VARIABEL REGLERING FÖR STEGMOTOR

Du skall nu skriva ett program för att styra en stegmotor på en Stepper Motor-modul. Stegmotorn, som är avsedd för unipolär drivning, är ansluten via fyra enpoliga kopplingskablar till FLIS-processorns utport. Stegmotorns axel fås att rotera genom att de olika faserna (RÖD, GUL, ORANGE och BRUN) styrs ut.

Tabellen nedan visar fyra olika tillstånd för Stepper Motor-modulen. För att få stegmotorn att rotera skall vi först sätta GUL och BRUN fas till logiknivå 1 (+5V) samtidigt som faserna RÖD och ORANGE ges logiknivå 0 (GND). Detta motsvarar tabellens första kolumn.



Stegmotorns rotationsriktning				
Fas	MEDURS → ← MOTURS			
	1	2	3	4
RÖD	GND	GND	+5V	+5V
GUL	+5V	+5V	GND	GND
ORANGE	GND	+5V	+5V	GND
BRUN	+5V	GND	GND	+5V

- Om stegmotorn ska rotera medurs, ska sedan faserna ges de nivåer som anges i kolumn två, dvs. ORANGE och BRUN ändras medan RÖD och GUL lämnas som de är. Efter kolumn två används i tur och ordning kolumn tre, kolumn fyra, kolumn ett, osv.
- Om stegmotorn ska roteras moturs, regleras faserna i stället genom att kolumnerna genomlöps i följande ordning: kolumn ett, kolumn fyra, kolumn tre, kolumn två, kolumn ett, osv.
- Stegmotorns faser är anslutna via FLIS-processorns utport enligt följande:

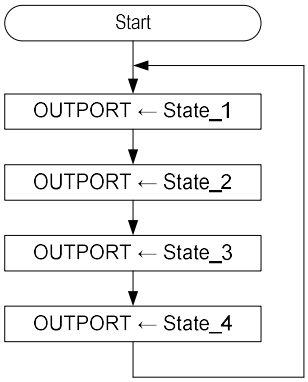
	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
OUTPORT	x	x	x	x	RÖD	GUL	ORANGE	BRUN

HEMUPPGIFT 4.A

Fyll i följande tabell som anger stegmotorns faser för att rotera medurs.

	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	HEX
State_1	0	0	0	0					
State_2	0	0	0	0					
State_3	0	0	0	0					
State_4	0	0	0	0					

Skriv en instruktionssekvens som får stegmotorn att rotera medurs. Utforma instruktionssekvensen efter följande flödesdiagram:



- Redigera en källtextfil `Lab_4-3A.sflisp`, enligt flödesplanen, assemblera filen och rätta eventuella fel.
- FLIS-processorns utport är ansluten till adress $FB.
- Då det för närvarande inte finns någon kompatibel stegmotor i simulatorn använder du istället en Bargraph-modul i simulatorn för att se att rätt värden skrivs ut till porten i rätt ordning.

VID LABORATIONSPLATSEN

- Anslut modulen Stepper Motor till FLIS-processorns **FB OUT** med en 10-polig flatkabel.
- Kontrollera instruktionssekvensens funktion genom att använda step-funktionen hos FLISP.
- Fungerar nu detta? Vrider stegmotorn sig ett steg i taget? *Om inte,* kontrollera att faserna ges rätt logiknivåer genom att observera vad lysdioderna för b_3, b_2, b_1 och b_0 på **FB OUT** visar.
- Rätta eventuella fel, stegmotorn skall vrida sig ett steg för varje nytt värde som matas ut.
- Starta därefter exekvering av instruktionssekvensen genom att ställa omkopplare i run-läge.
- Variera laborationsdatorns exekveringshastighet och observera eventuella skillnader i stegmotorns beteende.

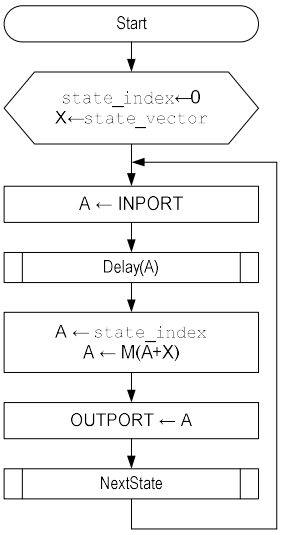
SJÄLVTESTNING

Testa er lösning på samma sätt som ni gjort i tidigare uppgifter. Nedan finns lämpliga tester att utföra men ni får gärna skapa fler.

- Se till så att stegmotorn rör sig medurs.
- Kontrollera att stegmotorn inte hackar (om den gör det så kontrollera att ni skickar rätt värden till utporten och att det inte är glapp. Notera att ni kan ändra exekveringshastigheten eller till och med stega igenom programmet)

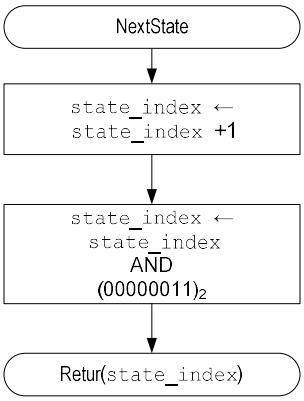
HEMUPPGIFT 4.B

Konstruera ett program för variabel reglering av stegmotorns rotationshastighet enligt följande flödesdiagram:



- Redigera en källtextfil `Lab_4-3B.sflisp`, enligt flödesplanen, lägg till subrutinerna Delay och NextState, assemblera filen och rätta eventuella fel.
- FLIS-processorns inport är ansluten till adress $FC.
- Utforma subrutinen, Delay, så att fördröjningen anges i register A vid anrop, jämför med uppgift 9.20 i kursboken.

Konstruera en subrutin `NextState` som fungerar som en "modulo-4 räknare", dvs. som bestämmer "kolumn" för stegmotorns nästa tillstånd då den ska vridas ett steg medurs. Tillståndet ska returneras i A. Subrutinen måste ges någon form av "minne", dvs. det värde som rutinen ska returnera beror av det föregående värdet. Ett sätt att göra detta är att använda en global variabel, vi kallar den `state_index`, och denna variabel kan enbart anta värdena 0,1,2,3.



Vi definierar också subrutin och data i pseudokod:

```
state_index = 0;   // initialt värde indexvariabel

state_vector={State_1, State_2, State_3, State_4};   // konstant vektor


NextState:

       state_index = state_index+1;

       state_index = state_index & 3;

       return (state_index);
```

VID LABORATIONSPLATSEN

- Anslut modulen DIL-switch till FLIS-processorns **FC IN** med en 10-polig flatkabel.
- Kontrollera programmets funktion i run-läge.
- Variera stegmotorns hastighet genom att ställa in olika värden på DIL-switch-modulen.

SJÄLVTESTNING

Testa er lösning på samma sätt som ni gjort i tidigare uppgifter. Nedan finns lämpliga tester att utföra men ni får gärna skapa fler.

- Alla kriterier för Del 1 skall uppfyllas.
- Se till att högre fördröjningsvärden får stegmotorn att röra sig långsammare.
  Var noga med att "ingen fördröjning" skall kunna användas, d.v.s. när fördröjningsvärdet är 0!

Då programmet fungerar som det ska tillkallar du handledare för att redovisa lösningen på laborationsuppgiften.

LABORATIONSUPPGIFT 4.4
ÖVERSÄTTNING TILL MORSEKOD

HEMUPPGIFT 4.C

Förbered den här laborationsuppgiften genom att skriva kod för de rutiner som beskrivs nedanför. Börja med att ladda hem filen `lab4-4.sflisp` från kurshemsidan i Canvas. Filen innehåller bland annat några stubbar för de nedanstående rutinerna.

I laborationssystemet är utporten ansluten till adress $FB och inporten till adress $FC.

Målet med den här uppgiften är att skriva ett program som får en ”lampa” (diodramp) ansluten till en utport på FLISP att blinka Morsekod för en bokstav, vars ASCII-kod matats in på en DIL-switch ansluten till en inport på FLISP. Programmeringen är indelad i tre delar. Gå vidare till nästa deluppgift först efter att du testat den föregående delen och sett till att den fungerar som avsett:

1) En rutin som läser in ASCII-kod från inporten
2) En rutin som får lampan att blinka en gång, snabbt eller långsamt
3) Ett huvudprogram som använder de två ovanstående rutinerna för att översätta från ASCII-kod till blinkningar

VID LABORATIONSPLATSEN

- Anslut Bargraph-modulen till FLIS-processorns **FB OUT** med en 10-polig flatkabel.
- Anslut DIL-switch-modulen till FLIS-processorns **FC IN** med en 10-polig flatkabel.

DELUPPGIFT 4.4.1. INLÄSNING AV ASCII-TECKEN

Skriv en subrutin som läser in ASCII-koden för en VERSAL (stor) bokstav från inporten. Koden ska ligga i A-registret när rutinen returnerar. DIL-switchens högsta sju bitar, bit7 till bit1, ska innehålla ASCII-koden. Stigande flank på den lägsta biten, bit0, används för att markera att man är klar med att mata in ett tecken. Pseudokod för inmatningsrutinen:

```
Vänta tills bit0 på inporten = 0   // De här två raderna väntar på en stigande flank på bit0
Vänta tills bit0 på inporten = 1
A ← inporten
A ← A >> 1                         // Placera ASCII-koden i de lägsta bitarna
Retur
```

Anropa ovanstående rutin från huvudprogrammet, och säkerställ att rätt värde finns i A-registret när subrutinen returnerar. Om du är osäker på hur du ska testa fråga en handledare innan du går vidare.

DELUPPGIFT 4.4.2. BLINKANDE LAMPA

Skriv nu en subrutin som tänder lampan och sedan släcker den efter att en viss tid har gått. Vid anrop av subrutinen skall A-registret innehålla ett bitmönster som anger hur länge lampan ska vara tänd. Om den mest signifikanta biten, bit7, i A-registret är ettställd ska lampan vara tänd i fyra sekunder innan den släcks. Om bit7 i A-registret är nollställd, ska lampan vara tänd i en sekund innan den släcks. Tänd lampan genom att skriva värdet FF_{16} till utporten. Släck lampan genom att skriva värdet 00_{16} till utporten. Efter att lampan släckts, ska rutinen även skapa en fördröjning på en sekund innan den returnerar. Till er hjälp har ni en fördröjningsrutin, `Delay1s`, som orsakar en fördröjning på ungefär 1s i simulatorn.

Pseudokod för att blinka med lampan:

```
Pusha A                    // Pusha A-registret för att kunna återställa i slutet av rutinen
Tänd lampan                // Detta skall göras utan att A-registrets värde ändras
Om bit7 i A = 1
   Anropa Delay1s fyra gånger
Annars
   Anropa Delay1s en gång
Släck lampan
Anropa Delay1s
Pulla A
Retur
```

Anropa ovanstående subrutin från ett huvudprogram, och testa att den fungerar som avsett, både med korta och långa blinkningar. Om du är osäker på hur du ska testa fråga en handledare innan du går vidare.

DELUPPGIFT 4.4.3. EN ÖVERSÄTTARE FRÅN ASCII TILL MORSE

I filen `lab4-4.sflisp` finns en tabell med värden, svarande mot Morsekod för de stora bokstäverna 'A' till och med 'Z'. I denna tabell är det två byte för varje bokstav: en byte som anger antalet pulser i koden för en viss bokstav, och en byte med ett bitmönster som anger följden av långa/korta pulser för en viss bokstav. Exempelvis ser första raden i tabellen ut så här:

```
MorseCode           FCB   2,%01000000     ; Morsekod för bokstaven 'A'
```

Detta betyder att första bokstaven i Morsealfabetet har två pulser, och dessa är en kort puls följt av en lång. De sex sista bitarna i bitmönstret för andra byten är godtyckligt satta till 0 i exemplet.

Nu är det dags att sätta samman allt till ett program som läser in ASCII-kod från DIL-switch-modulen och sedan får lampan att blinka motsvarande Morsekod. Nedan finns pseudokod för detta.

```
Släck lampan
Anropa inläsningsrutinen
A ← A - $41          // Räkna ut ordningstalet för bokstaven som lästes in. (Ordningstalet för 'A' = 0)
Undersök att A är på ett giltigt intervall. Om inte, skapa en loop där
lampan tänds(Något är ju fel, så programmet skall fastna här)
A ← 2A               // Eftersom det är två byte per bokstav i tabellen, måste offset dubbleras
Pusha A
X ← Startadress för tabell med Morsekoder
A ← M(X + A)         // Hämta antalet pulser för bokstaven
Count ← A            // Count är en variabel (en reserverad byte i minnet)
Pulla A
A ← A + 1            // Beräkna offset för pulsmönstret
A ← M(X + A)         // Hämta pulsmönstret (Morsekoden) för  bokstaven
Repetera så länge som Count ≠ 0
        Anropa blinkrutinen
        A ← A << 1
        Count ← Count – 1
Börja om från inläsningen
```

SJÄLVTESTNING

Testa er lösning på samma sätt som ni gjort i tidigare uppgifter. Nedan finns lämpliga tester att utföra men ni får gärna skapa fler.

- Se till att programmet skall kunna exekveras i HÖGSTA hastigheten. Inmatningen av ett ASCII-tecken får inte bero på hur snabbt programmet exekveras.
- Efter att en morsekod visats skall programmet åter igen vänta på ett nytt tecken och en POSITIV FLANK (d.v.s. en övergång från 0 till 1) på den lägsta biten.
- Kontrollera att några morsekoder kan visas riktigt och eventuellt gör det tydligare med skillnaden mellan långa och korta signaler. Det är en bra början att testa A och Z.

Då programmet fungerar som det ska tillkallar du handledare för att redovisa lösningen på laborationsuppgiften.

TABELL: ASCII-KOD

0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1	$b_6b_5b_4$ $b_3b_2b_1b_0$
NUL	**DLE**	**SP**	**0**	**@**	**P**	**`**	**p**	0 0 0 0
SOH	**DC1**	**!**	**1**	**A**	**Q**	**a**	**q**	0 0 0 1
STX	**DC2**	**“**	**2**	**B**	**R**	**b**	**r**	0 0 1 0
ETX	**DC3**	**#**	**3**	**C**	**S**	**c**	**s**	0 0 1 1
EOT	**DC4**	**$**	**4**	**D**	**T**	**d**	**t**	0 1 0 0
ENQ	**NAK**	**%**	**5**	**E**	**U**	**e**	**u**	0 1 0 1
ACK	**SYN**	**&**	**6**	**F**	**V**	**f**	**v**	0 1 1 0
BEL	**ETB**	**‘**	**7**	**G**	**W**	**g**	**w**	0 1 1 1
BS	**CAN**	**(**	**8**	**H**	**X**	**h**	**x**	1 0 0 0
HT	**EM**	**)**	**9**	**I**	**Y**	**i**	**y**	1 0 0 1
LF	**SUB**	*****	**:**	**J**	**Z**	**j**	**z**	1 0 1 0
VT	**ESC**	**+**	**;**	**K**	**[Ä**	**k**	**{ä**	1 0 1 1
FF	**FS**	**,**	**<**	**L**	**\Ö**	**l**	**|ö**	1 1 0 0
CR	**GS**	**-**	**=**	**M**	**]Å**	**m**	**}å**	1 1 0 1
S0	**RS**	**.**	**>**	**N**	**^**	**n**	**~**	1 1 1 0
S1	**US**	**/**	**?**	**O**	**_**	**o**	**RUBOUT (DEL)**	1 1 1 1

TILLÄGG TILL PM LABORATION 4

Du kan ge kommandon till FLIS-processorn genom att klicka på terminalfönstret i *DigiFlisp* och skriva in något av följande:

Kommando	Betydelse
i	Interaktiv mode, alla tecken skickas tillbaks och syns i terminalfönstret
q	Tyst mode, inga tecken skickas tillbaks
v	Om mode=i, skriv versionsnummer till terminal
t	Testsekvens, testar FLIS-processorns indikatorer genom att tända dessa successivt, avsluta genom att ge kommandot 't' igen.
a	Reset, utför återställning av FLIS-processorn
s	Om Execute-omkopplare = halt: Utför en instruktion varje gång kommandot ’s’ ges. Om Execute-omkopplare = run: Processorinstruktioner exekveras utan uppehåll, och exekveringshastigheten (tre olika) ändras varje kommandot ’s’ ges.
wrZXX	Skriv värdet XX till register Z. Värdet XX anges på hexadecimal form med precis två siffror. Registret, Z, kan vara något av datavägens register enligt: a,x,y,s=sp,p=pc, r,c=cc.
wmYYXX	Skriv värdet XX till minnesadress YY. Såväl värdet XX som adressen YY anges på hexadecimal form med precis två siffror.

rrZ	Om mode=i, Skicka värdet (hexadecimal form) i register Z till terminalen. Registret, Z, kan vara något av datavägens register enligt: a,x,y,s=sp,p=pc, c=cc.
rmYY	Om mode=i, Skicka värdet (hexadecimal form) på minnesadress YY till terminalen

FLIS-datorn är i ”interaktiv mode” efter RESET.