

# *Maskinnära programmering*

*- exempelsamling*

Institutionen för Data och Informationsteknik  
Chalmers tekniska högskola  
Göteborg VT-2013

# Maskinnära programmering - exempelsamling

©2000-2013

Roger Johansson, Jan Skansholm, Lars-Eric Arebrink och Rolf Snedsböl  
Denna publikation får kopieras fritt i sin helhet för undervisningsändamål.

## Innehåll

1. Grundläggande assemblerprogrammering
2. Grundläggande programmering i 'C'
3. Undantagshantering
4. Systemprogrammering och periferikretsar

Exempel är anpassade för *ETERM6* för *MC12* respektive *XCC12* för *MC12*.

Versioner:

18 januari 2011

17 februari 2011, lagt till ytterligare uppgifter i avsnitt 2, lagt till avsnitt 4

2012, lagt till ytterligare uppgifter i avsnitt 2

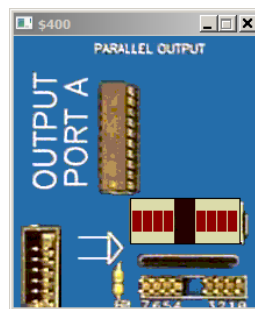
6 mars 2013, rättat smärre fel och typografi.

---

# 1 Grundläggande assemblerprogrammering

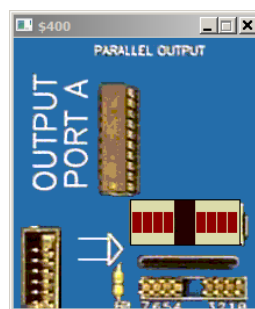
1.1 En ramp med ljusdioder, enligt figuren till höger, är ansluten till adress \$400 på ett MC12-system.

- Skriv en subrutin "BLINK" som får samtliga dioder att blinka genom att kontinuerligt tända och släcka dom. Kontrollera funktionen genom att stega igenom subrutinen instruktionsvis.
- Utforma, som en ny subrutin "BLINKDELAY", en fördröjning så att dioderna blinkar även då programmet exekveras normalt.
- Beskriv lösningen från b) i form av en flödesplan.



1.2 En ramp med ljusdioder, enligt figuren till höger, är ansluten till adress \$400 på ett MC12-system.

- Skriv en subrutin "RLJUSH" som får dioderna att bete sig som ett "rinnande ljus" där dioderna tänds upp en och en från vänster till höger. Kontrollera funktionen genom att stega igenom subrutinen instruktionsvis.
- Använd subrutinen "BLINKDELAY", så att man tydligt kan se det rinnande ljuset även då programmet exekveras normalt.
- Beskriv lösningen från b) i form av en flödesplan.

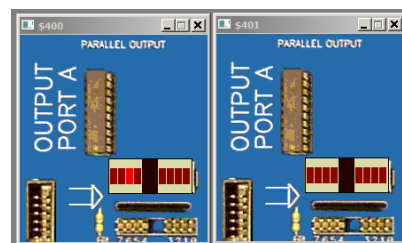


1.3 Två ramper med ljusdioder, enligt figuren till höger, är anslutna till adress \$400 och \$401 på ett MC12-system.

Du ska konstruera en subrutin "RLJUSH16" som får dioderna att bete sig som ett kontinuerligt "rinnande ljus" där dioderna tänds upp en och en från vänster till höger. Efter det att bit 0 hos diodrampen på adress \$400 släckts ska bit 7 hos diodrampen på adress \$401 tändas. Då dioden för bit 0 på adress \$401 släckts, ska det rinnande ljuset börja om från bit 7 på adress \$400, osv.

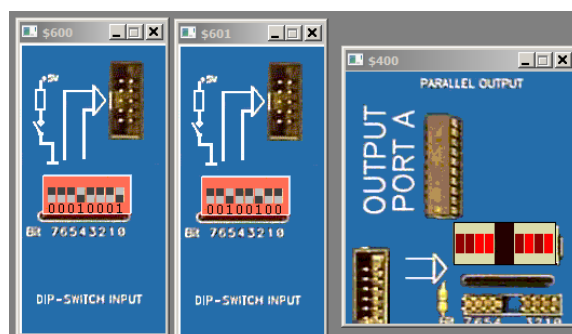
Använd en given subrutin "BLINKDELAY", så att man tydligt kan se det rinnande ljuset även då programmet exekveras normalt.

- Beskriv subrutinen "RLJUSH16" i form av en flödesplan.
- Implementera, dvs. skriv subrutinen i assemblerspråk.



1.4 Två strömbrytare och en ljusdiodramp, enligt figuren till höger, är anslutna till adresser \$600 och \$601, respektive adress \$400 på ett MC12-system.

Konstruera en subrutin "DipSwitchOr" som bildar logisk ELLER av värdena som läses från strömbrytarna. Subrutinen ska utformas så att avläsningen och indikering görs en gång. Kontinuerlig funktion fås genom att subrutinen, oupphörligt anropas från ett huvudprogram "main".

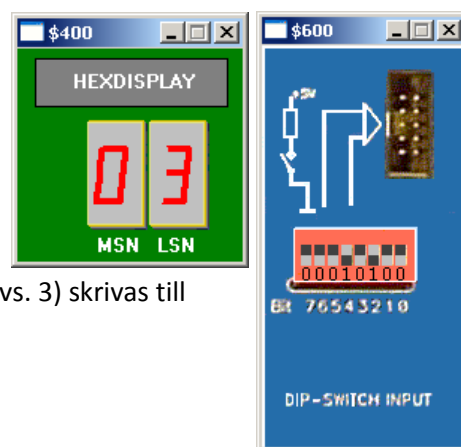


- Beskriv subrutinen "DipSwitchOr" i form av en flödesplan.
- Implementera huvudprogrammet "main" och subrutinen "DipSwitchOr" i assemblerspråk.

- 1.5 En 8-bitars strömbrytare, "DIP\_SWITCH" är ansluten till adress \$600 och en displayenhet "HEXDISPLAY" som visar en byte i form av två hexadecimala siffror är ansluten till adress \$400 i ett MC12 mikrodatorsystem.

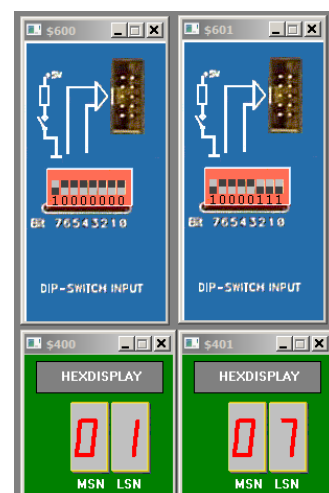
Konstruera en subrutin `DipHex` som läser av strömbrytaren och indikerar den minst signifikanta påslagna biten genom att skriva dess position, räknat från höger, till displayenheten. Om exempelvis bitarna 2 och 4 utgör ettställda strömbrytare ska positionen för bit 2, (dvs. 3) skrivas till displayenheten.

Om ingen strömbrytare är ettställd ska siffran 0 skrivas till displayen. Speciellt gäller att endast symboler ska användas för absoluta adresser.



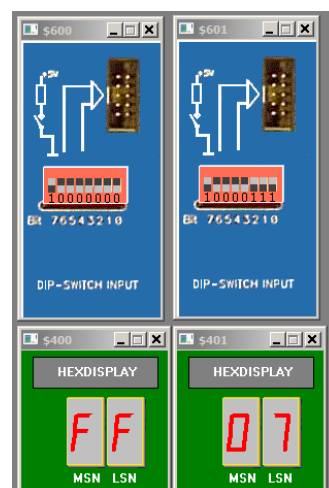
- 1.6 Två strömbrytare och två displayenheter, enligt figuren till höger, är anslutna till adresser \$600 och \$601, respektive adress \$400 och \$401 i ett MC12-system. Konstruera en subrutin "AddUnsigned8bitTo16" som adderar de två värdena som läses från strömbrytarna (tolka som tal utan tecken) och därefter presenterar resultatet som ett 16 bitars tal på displayindikatorerna. Subrutinen ska utformas så att avläsningen och indikering görs en gång. Kontinuerlig funktion fås genom att subrutinen, oupphörligt anropas från ett huvudprogram "main".

- Beskriv subrutinen "AddUnsigned8bitTo16" i form av en flödesplan.
- Implementera huvudprogrammet "main" och subrutinen "AddUnsigned8bitTo16" i assemblerspråk.



- 1.7 Två strömbrytare och två displayenheter, enligt figuren till höger, är anslutna till adresser \$600 och \$601, respektive adress \$400 och \$401 i ett MC12-system. Konstruera en subrutin "AddSigned8bitTo16" som adderar de två värdena som läses från strömbrytarna (tolka som tal **med** tecken) och därefter presenterar resultatet som ett 16 bitars tal på displayindikatorerna. Subrutinen ska utformas så att avläsningen och indikering görs en gång. Kontinuerlig funktion fås genom att subrutinen, oupphörligt anropas från ett huvudprogram "main".

- Beskriv subrutinen "AddSigned8bitTo16" i form av en flödesplan.
- Implementera huvudprogrammet "main" och subrutinen "AddSigned8bitTo16" i assemblerspråk.

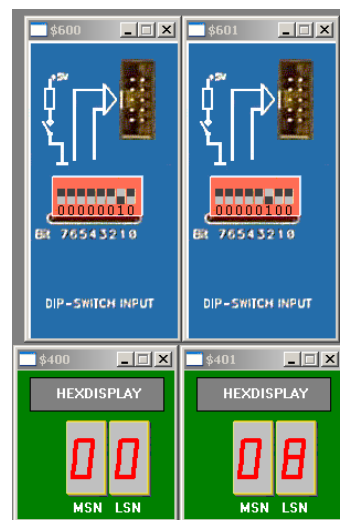


- 1.8 Två 8-bitars strömbrytare, "DIP\_SWITCH" är anslutna till adresserna \$600,\$601 och två displayenheter "HEXDISPLAY" som var och en visar en byte i form av två hexadecimala siffror är anslutna till adresserna \$400 och \$401 i ett MC12 mikrodatorsystem.

Skriv en subrutin som läser de båda strömbrytarnas inställda värden, multiplicerar dessa båda tal och skriver det 16 bitars resultatet till displayenheterna.

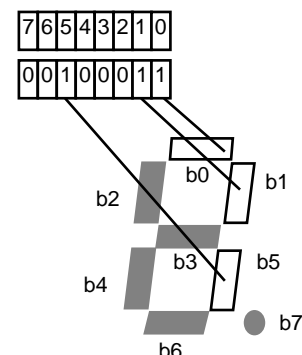
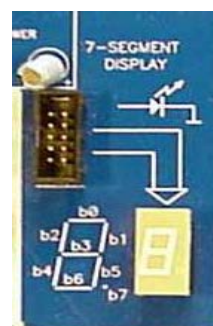
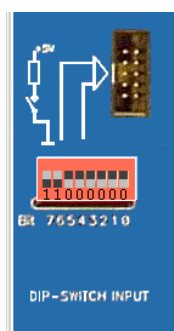
Displayenheten på adress \$400 ska ange den mest signifikanta byten av resultatet.

Speciellt gäller att endast symboler ska användas för absoluta adresser.



- 1.9 En 8-bitars strömbrytare "DIP-SWITCH INPUT" och en sju-sifferindikator "7-SEGMENT DISPLAY" är anslutna till adresserna \$0600 respektive \$0400 i ett MC12 mikrodatorsystem.

Använd symbolen `ML4_INPUT` för inporten (\$0600) och symbolen `ML4_OUTPUT` för utporten (\$0400).



Skriv en subrutin "DisplayNBCD" som kontinuerligt läser inporten (strömbrytarna) och skriver värden (NBCD-siffror) till utporten (7-sifferindikatorn).

När bit 7 på inporten är ettställd skall sifferindikatorn släckas helt. När bit 7 på inporten är nollställd skall sifferindikatorn tändas enligt följande beskrivning:

Bit 3-0 på inporten anger vad som skall visas på sifferindikatorn. Om indata är i intervallet [0,9] skall motsvarande decimala siffra visas på sifferindikatorn. Om indata är i intervallet [A,F] skall ett 'E' (Error) visas på sifferindikatorn. Segmentkoden för 'E' är \$5D.

Bitarna 6-4 på inporten kan anta vilka värden som helst.

Du har tillgång till en tabell i minnet med segmentkoder (mönster för sifferindikatorn) enligt

**SegCodes FCB \$77,\$22,\$5B,\$6B, etc.**

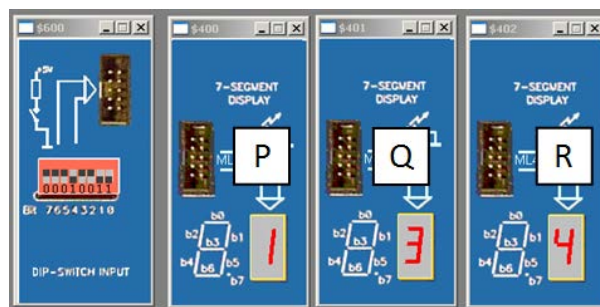
Tabellen innehåller segmentkoder för siffrorna [0,9].

På adressen "SegCodes" i minnet finns segmentkoden för 0, på adressen "SegCodes+1" i minnet finns segmentkoden för 1, på adressen "SegCodes+2" i minnet finns segmentkoden för 2, etc

- 1.10 En 8-bitars strömbrytare "DIP-SWITCH INPUT" och tre sju-sifferindikatorer "7-SEGMENT DISPLAY" är anslutna till adresserna \$0600 respektive \$0400, \$401 och \$402 i ett MC12 mikrodatorsystem.

Skriv en subrutin "SumPQ" som

- hela tiden läser två NBCD-siffror P och Q från strömbrytarna
- visar NBCD siffrorna P och Q på två olika sifferindikatorer
- utför en additionen  $R=P+Q$
- skriver summan R till den tredje sifferindikatorn.

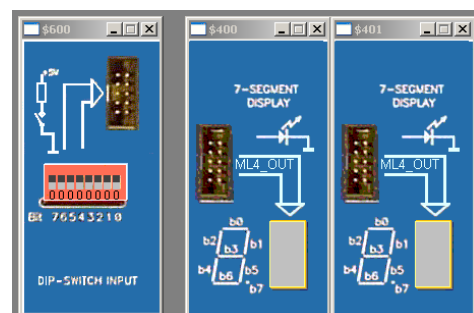


Från inporten (8 bitar) läses två 4-bitars binära tal P och Q samtidigt. P hittas på  $[b_7, b_4]$  och Q hittas på  $[b_3, b_0]$ . Summan skall placeras i  $[b_3, b_0]$  för att omvandlas till segmentkod och skrivs till sifferindikatorn. Om summan  $P+Q$  är större än nio skall "E" (ERROR) skrivas ut. Du får förutsätta att  $P \leq 9$  och  $Q \leq 9$ .

Du har tillgång till en tabell med segmentkoder och följande definitioner:

```
Inport      EQU    $600                ; Adress för inport
UtportP     EQU    $400                ; Adress för utport 1
UtportQ     EQU    $401                ; Adress för utport 2
UtportR     EQU    $402                ; Adress för utport 3
Error       EQU    %01011101          ; Segmentkod för E (Error)
SegCode     FCB    %11101111,%0100010, etc ; Tabell med segmentkoder för [0,9]
```

- 1.11 Två 7-sifferindikatorer, "7-SEGMENT DISPLAY" är anslutna till adresserna \$400, \$401 och en 8 bitars strömbrytare "DIP-SWITCH INPUT" är ansluten till adress \$600 i ett MC12 mikrodatorsystem. Du skall skriva subrutinerna "Read" och "Display" till följande program som om och om igen läser inporten (ett NBCD-tal  $[0,99]_{10}$ ) och skriver detta till de båda sifferindikatorerna.



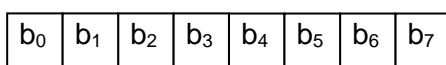
```
                ORG    $1000
main:          JSR    Read              ; Läs NBCD-tal till register
A
                JSR    Display          ; Skriv register A på 2 sifferindikatorer
                JMP    main
```

Följande definition är dessutom given.

```
SegCodes      FCB    $77,$22,$5B,$6B, etc. ; (segmentkoder för siffrorna [0,9]).
```

Subrutinen *Display* visar det NBCD-tal som finns lagrat i register A. Innehåller register A exempelvis 0101 1001 skall 5 visas på UtPort1 och 9 visas på UtPort2. (Segmentkoder är alltså givna med start på adress "SegCodes"). Skriv subrutinen *Display*!

Subrutinen *Read* läser InPort. Tyvärr har inporten ett konstruktionsfel så bitarna är omkastade enligt följande figur. (Bit  $b_7$  är ju normalt till vänster och  $b_0$  till höger)



Detta medför att när vi ställer in NBCD-talet 53 (0101 0011) på strömbrytarna så läses 1100 1010 från inporten (ty det spegelvänds).

Subrutinen måste därför

1. läsa inporten
2. spegelvända det inlästa
3. lämna utdata i i register A..

Skriv subrutinen *Read*!

## 2 Grundläggande programmering i 'C'

2.1 Ange talområdena för variablerna i följande deklARATIONER, (XCC12):

- a) `unsigned char uc;`
- b) `signed char sc;`
- c) `unsigned short us;`
- d) `signed short ss;`
- e) `unsigned int ui;`
- f) `signed int si;`

Ledning: Konsultera filen "limits.h"

2.2 Ange talområdena för variablerna i följande deklARATIONER, (XCC12):

- a) `unsigned long int ul;`
- b) `signed long int sl;`

Ledning: Konsultera filen "limits.h"

2.3 I XCC12 implementeras reella tal med IEEE 754, (single precision) flyttalsstandard.

- a) Hur många signifikanta decimala siffror, ryms i mantissan?
- a) Ange på decimal form, minsta tal som kan representeras med 4 signifikanta siffror.
- b) Ange på decimal form, största tal som kan representeras med 4 signifikanta siffror.
- c) Ange diskretiseringssteget, dvs. minsta skillnaden mellan två konsekutiva reella tal.

Ledning: Konsultera filen "float.h"

2.4 En 'C'-variabel måste tillhöra en av lagringsklasserna auto, static och global.

Redogör för "synligheten" hos variabler deklarerade med respektive lagringsklass.

2.5 Ange de, av följande deklARATIONER, som är korrekta i ett 'C'-program:

```
int    a;
auto   int  aia;
static int  sia;
global int  gia;
extern int  eia;
intern int  iia;

void f( void)
{
    int    b;
    auto   int  aib;
    static int  sib;
    global int  gib;
    extern int  eib;
    intern int  iib;
}
```

2.6 För att referera absoluta adresser, exempelvis portar, krävs att en konstant (den absoluta portadressen) förses med lämpliga explicita typkonverteringar. Visa korrekta typkonverteringar (ANSI-C) i följande fall där portadressen är 0x400:

- a) 8-bitars port där portens innehåll betraktas som tal utan tecken.
- b) 8-bitars port där portens innehåll betraktas som tal med tecken.
- c) 16-bitars port där portens innehåll betraktas som tal utan tecken.
- d) 16-bitars port där portens innehåll betraktas som tal med tecken.

- 2.7 För att referera absoluta adresser, exempelvis portar, krävs att en konstant (den absoluta portadressen) förses med lämpliga explicita typkonverteringar. Visa korrekta typkonverteringarna, , i följande fall där portadressen är 0x400. Använd C99 utvidgningen `stdint.h` för maximal portabilitet
- 8-bitars port där portens innehåll betraktas som tal utan tecken.
  - 8-bitars port där portens innehåll betraktas som tal med tecken.
  - 16-bitars port där portens innehåll betraktas som tal utan tecken.
  - 16-bitars port där portens innehåll betraktas som tal med tecken.
- 2.8 Visa typdeklarationer för en funktion som tillåter att funktionen i form av en subrutin på en fast adress i minnet, kan anropas direkt från ett C-program.
- funktionen `reentry` har inga parametrar och inget returvärde, på adress 0xC00F.
  - funktionen `outcha` har en parameter av typen `unsigned char`, men inget returvärde, på adress 0xC006.
  - funktionen `tstcha` har inga parametrar men returvärde av typen `unsigned char`, på adress 0xC003.
- 2.9 Det amerikanska sättet att ange datum skiljer sig från det svenska och kan ibland vara lite förvirrande. Datumet 15:e oktober 2005 skrivs t.ex. som 10/15/05, medan man i Sverige brukar använda formen 2005-10-15. Skriv ett program som översätter ett datum från den amerikanskadatumformen till den svenska. När programmet körs kan det se ut på följande sätt:
- ```
Ange datum som mm/dd/yy 10/15/05
På svenskt sätt skrivs datumet som 2005-10-15
```
- 2.10 Skriv ett program som läser in dagens datum som: *20åå-mm-dd* (minustecknen läses också in). Därefter skall programmet läsa in ett personnummer och skriva ut texten: *Grattis!* om den aktuella personen har födelsedag. Personnummer anges med 10 siffror (utan minustecken). Använd en standardfunktion för att jämföra texterna.
- 2.11 De rationella talen är exakta tal som anges på formen  $t/n$  där  $t$  och  $n$  (täljaren och nämnaren) är heltal.
- Använd `typedef` och `struct` för att deklarerar en typ `rat_tal` som beskriver ett rationellt tal.
  - Skriv sedan två funktioner `add` och `mul`. De skall båda få två parametrar av typen `rat_tal`. Som resultat skall de ge ett nytt rationellt tal som är summan respektive produkten av de två parametrarna.
  - Skriv också en main-funktion i vilken man läser in täljaren och nämnaren för två rationella tal och beräknar talens summa och produkt med hjälp av funktionerna `add` och `mul`.
- 2.12 Skriv en egen version av standardfunktionen `strlen`.
- Använd pekare.
  - Använd indexering.
  - Använd XCC12, kompilera de båda versionerna till assemblerkod och jämför resultaten.
- 2.13 Skriv en egen version av standardfunktionen `strcpy`.
- Använd pekare.
  - Använd indexering.
  - Använd XCC12, kompilera de båda versionerna till assemblerkod och jämför resultaten.
- 2.14 Skriv ett program som läser in en befintlig textfil och som skriver ut filens innehåll på terminalen. Vid utskriften skall inga tomma rader (rader som bara innehåller en radslutsmarkör) skrivas ut. Rader som bara innehåller blanka tecken skall inte heller skrivas ut. I övrigt skall utskriften ha samma radstruktur som den som finns i filen. Filens namn skall anges som argument på kommandoraden när man startar programmet.
-



- 2.15 Konstruera en funktion `nollstalle` som beräknar ett nollställe till matematiska funktioner. Funktionen `nollstalle` har deklarationen:

```
double nollstalle(double (*f)(double), double a, double b, double eps);
```

Den första parametern,  $f$ , är en pekare till den matematiska funktion man vill söka ett nollställe för. De två parametrarna  $a$  och  $b$  anger inom vilket intervall nollstället skall sökas.

Man söker alltså ett värde  $x$  i intervallet  $(a, b)$  sådant att  $f(x) = 0$ . Du får anta att den funktion som  $f$  pekar på är monoton och att den har exakt ett nollställe inom det givna intervallet. Parametern  $eps$  anger vilket som är det största fel som får finnas i resultatet.

I funktionen kan du "ringa in" nollstället genom att flytta ändpunkterna  $a$  och  $b$  allt närmare varandra. Börja med att undersöka om  $f(a) < 0 < f(b)$  eller  $f(b) < 0 < f(a)$ . Om det är på det andra sättet så låt variablerna  $a$  och  $b$  byta värden med varandra. Upprepa sedan följande tills  $|a-b| \leq 0$ . Räkna ut mittpunkten  $m$  mellan  $a$  och  $b$  och beräkna värdet av  $f(m)$ . Om  $f(m) < 0$  så sätt  $a$  till  $m$  sätt annars  $b$  till  $m$ .

- 2.16 En ramp med ljusdioder, enligt figuren till höger, är ansluten till adress 0x400 i ett MC12 mikrodatorsystem.

- a) Skriv en funktion

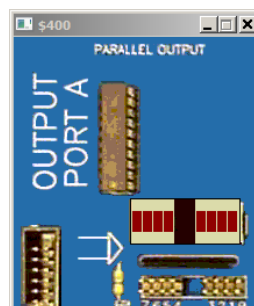
```
void blink( void )
```

som får samtliga dioder att blinka genom att kontinuerligt tända och släcka dem. Kontrollera funktionen genom att stega igenom den satsvis.

- b) Utforma, som en ny funktion

```
void blinkdelay( void )
```

en fördröjning så att dioderna blinkar även då programmet exekveras normalt.

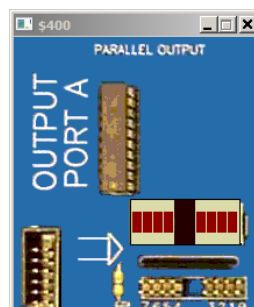


- 2.17 En ramp med ljusdioder, enligt figuren till höger, är ansluten till adress 0x400 i ett MC12 mikrodatorsystem.

Skriv en funktion

```
void rljush( void )
```

som får dioderna att bete sig som ett "rinnande ljus" där dioderna tänds upp en och en från vänster till höger. Kontrollera funktionen genom att stega igenom den satsvis. Använd funktionen `void blinkdelay( void )`, så att man tydligt kan se det rinnande ljuset även då programmet exekveras normalt.

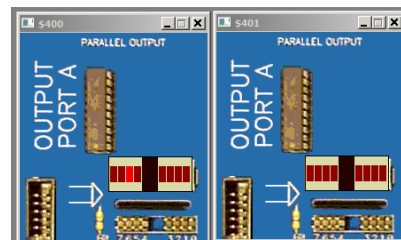


- 2.18 Två ramper med ljusdioder, enligt figuren till höger, är anslutna till adress 0x400 och 0x401 i ett MC12 mikrodatorsystem.

Du ska konstruera en funktion

```
void rljush16( void )
```

som får dioderna att bete sig som ett kontinuerligt "rinnande ljus" där dioderna tänds upp en och en från vänster till höger. Efter det att bit 0 hos diodrampen på adress 0x400 släckts ska bit 7 hos diodrampen på adress 0x401 tändas. Då dioden för bit 0 på adress 0x401 släckts, ska det rinnande ljuset börja om från bit 7 på adress 0x400, osv. Använd funktionen `void blinkdelay( void )`, så att man tydligt kan se det rinnande ljuset även då programmet exekveras normalt.

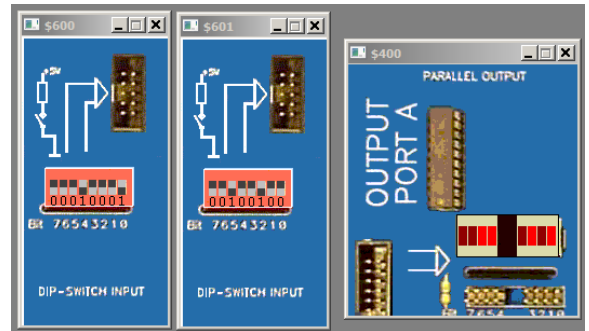


- 2.19 Två strömbrytare och en ljusdiodramp, enligt figuren till höger, är anslutna till adresser 0x600 och 0x601, respektive adress 0x400 i ett MC12 mikrodatorsystem.

Konstruera en funktion

```
void DipSwitchOr( void )
```

som bildar logisk ELLER av värdena som läses från strömbrytarna.

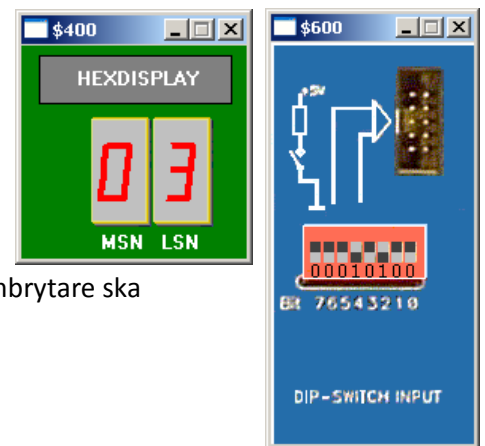


- 2.20 En 8-bitars strömbrytare är ansluten till adress 0x600 och en displayenhet som visar en byte i form av två hexadecimala siffror är ansluten till adress 0x400 i ett MC12 mikrodatorsystem.

Konstruera en funktion

```
void ff1( void )
```

som läser av strömbrytaren och indikerar den minst signifikanta påslagna biten genom att skriva dess position, räknat från höger, till displayenheten. Om exempelvis bitarna 2 och 4 utgör ettställda strömbrytare ska positionen för bit 2, (dvs. 3) skrivas till displayenheten. Om ingen strömbrytare är ettställd ska siffran 0 skrivas till displayen.

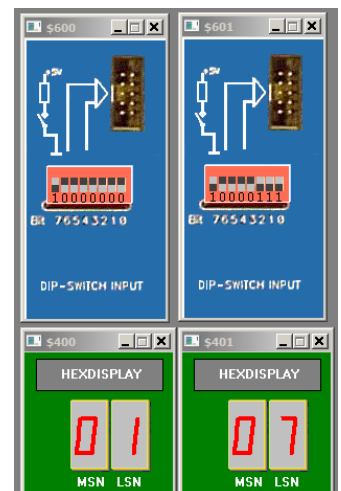


- 2.21 Två strömbrytare och två displayenheter, enligt figuren till höger, är anslutna till adresser 0x600 och 0x601, respektive adress 0x400 och 0x401 i ett MC12 mikrodatorsystem.

Konstruera en funktion

```
void AddUnsigned8bitTo16( void )
```

som adderar de två värdena som läses från strömbrytarna (tolka som tal utan tecken) och därefter presenterar resultatet som ett 16 bitars tal på displayindikatorerna.

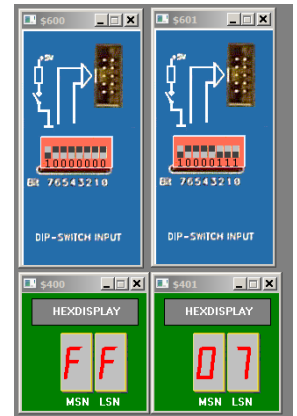


- 2.22 Två strömbrytare och två displayenheter, enligt figuren till höger, är anslutna till adresser 0x600 och 0x601, respektive adress 0x400 och 0x401 i ett MC12 mikrodatorsystem.

Konstruera en funktion

```
void AddSigned8bitTo16( void )
```

som adderar de två värdena som läses från strömbrytarna (tolka som tal med tecken) och därefter presenterar resultatet som ett 16 bitars tal på displayindikatorerna.



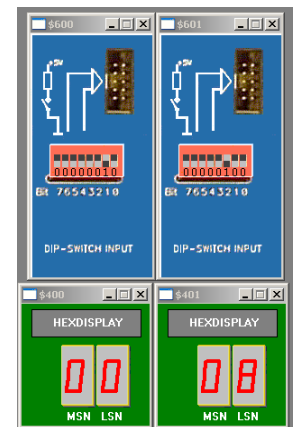
- 2.23 Två 8-bitars strömbrytare, är anslutna till adresserna 0x600,0x601 och två displayenheter som var och en visar en byte i form av två hexadecimala siffror är anslutna till adresserna 0x400 och 0x401 i ett MC12 mikrodatorsystem.

Konstruera en funktion

```
void DipHex( void )
```

som läser de båda strömbrytarnas inställda värden, multiplicerar dessa båda tal och skriver det 16 bitars resultatet till displayenheterna.

Displayenheten på adress 0x400 ska ange den mest signifikanta byten av resultatet.



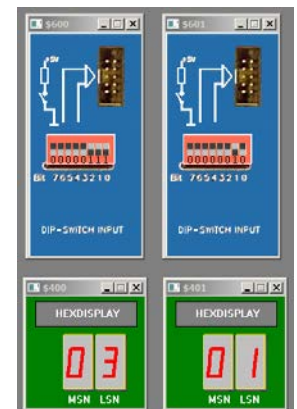
- 2.24 Två 8-bitars strömbrytare, är anslutna till adresserna 0x600,0x601 och två displayenheter som var och en visar en byte i form av två hexadecimala siffror är anslutna till adresserna 0x400 och 0x401 i ett MC12 mikrodatorsystem.

Konstruera en funktion

```
void DivModHex( void )
```

som läser de båda strömbrytarnas inställda värden.

Om värdet på adress 0x601 är noll ska 0xFF visas på båda displayenheter. Om värdet på adress 0x601 är skilt från noll ska resultatet av heltalsdivisionen mellan värden på adress 0x600 och 0x601 visas på displayindikator med adress 0x400 och resultatet av restdivisionen av samma tal visas på indikator med adress 0x401.

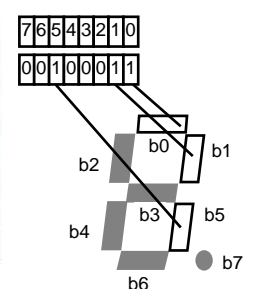
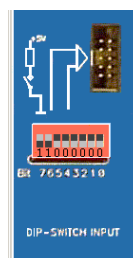


- 2.25 En strömbrytare och en sju-sifferindikator (se figur) är anslutna till adresser 0x400 respektive 0x600 i ett MC12 mikrodatorsystem.

Konstruera en funktion

```
void DisplayNBCD( void )
```

som hela tiden läser från strömbrytarna och skriver värden till sju-sifferindikatorn).



När bit 7 på inporten är ettställd skall sifferindikatorn släckas helt. När bit 7 på inporten är nollställd skall sifferindikatorn tändas enligt följande beskrivning:

- Bit 3-0 på inporten anger vad som skall visas på sifferindikatorn.
  - Om indata är i intervallet [0,9] skall motsvarande decimala siffra visas på sifferindikatorn.
  - Om indata är i intervallet [A,F] skall ett 'E' (Error) visas på sifferindikatorn.
- Bitarna 6-4 på inporten kan anta vilka värden som helst.

Du har tillgång till en tabell i minnet med segmentkoder för de hexadecimala siffrorna [0..F] (mönster för sifferindikatorn) enligt

```
unsigned char SegCodes[] = { 0x77, 0x22, 0x5B, 0x6B, 0x2E, 0x6D, 0x7D, 0x23,
                             0x7F, 0x6F, 0x3F, 0x7C, 0x55, 0x7A, 0x5D, 0x18 };
```

Segmentkoden för bokstaven 'E' ges av:

```
#define ERROR_CODE 0x5D
```

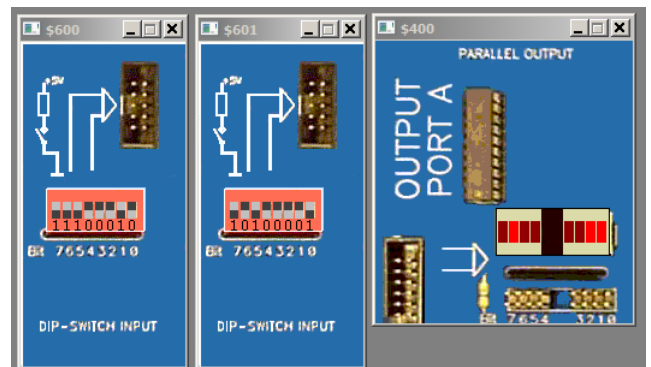
- 2.26 I denna uppgift ska du bland annat demonstrera hur absolutadressering utförs i C. Visa speciellt hur preprocessordirektiv och typdeklarationer används för att skapa begriplig programkod.

Två strömbrytare och en ljusdiodramp, enligt figuren till höger, är anslutna till adresser 0x600 och 0x601, respektive adress 0x400 i ett MC12 mikrodatorsystem.

Konstruera en funktion

```
void DipSwitchEor( void )
```

som kontinuerligt bildar logiskt EXKLUSIVT ELLER av värdena som läses från strömbrytarna och därefter skriver detta värde till ljusdiodrampen.



2.27 I denna uppgift ska du bland annat demonstrera hur absolutadressering utförs i C. Visa speciellt hur preprocessordirektiv och typdeklARATIONER används för att skapa begriplig programkod.

Två strömbrytare och en ljusdiodlamp, enligt figuren till höger, är anslutna till adresser 0x600 och 0x601, respektive adress 0x400 i ett MC12 mikrodatorsystem.

Konstruera en funktion

```
void CondRunDiode ( void )
```

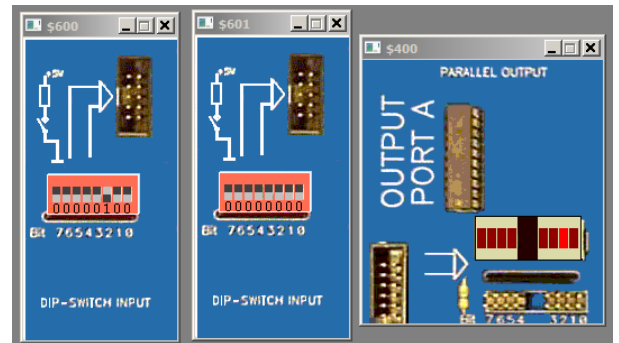
- som oupphörligt jämför strömbrytarnas värden
- dessutom skriver ut ett *rinnande ljus* på diodlampen.

Det rinnande ljuset består i att en diod i taget tänds upp.

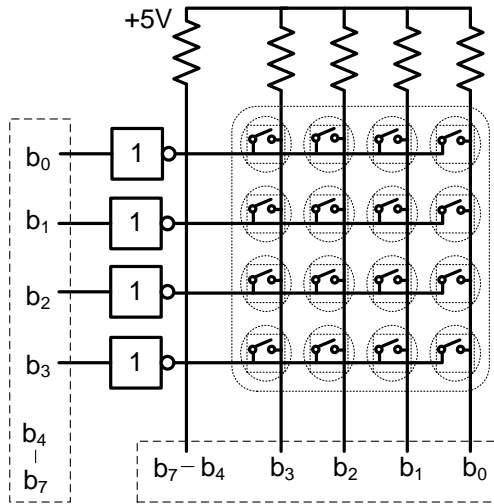
- Om värdet hos strömbrytaren på adress 0x600 är störst ska ljuset rinna från höger till vänster
- Om värdet hos strömbrytaren på adress 0x601 är störst ska ljuset rinna från vänster till höger
- Om värdena är lika ska det rinnande ljuset stannas.

Från början ska dioden längst till vänster vara tänd.

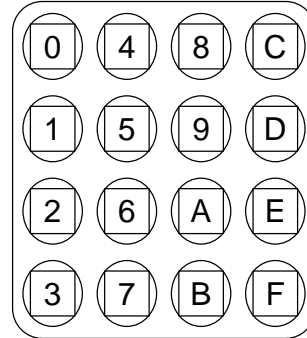
Du behöver här *inte* ta hänsyn till att fördröjningar krävs för det rinnande ljuset.



2.28 Följande gränssnitt ansluts till ett MC12 mikrodatorsystem. (Jämför med ML5/ML23 i kurslitteraturen).



Låt tangenterna representera följande:  
(A-F är hexadecimala siffror)



Konstruera en funktion

```
unsigned char keyb( void )
```

Denna skall som resultat ge numret på den tangent som trycktes ner. Numreringen framgår av figuren ovan.

- Funktionen skall först vänta tills ingen tangent är nedtryckt. Därefter skall den aktivera en rad i taget och avläsa kolumnernas ut signaler ända tills någon tangent tryckts ner.
- Porten med anslutningar till tangentbordets rader finns på adressen 0x0C00, porten med kolumnernas anslutningar finns på adress 0x0C01.
- Då en nedtryckt tangent konstaterats ska funktionen vänta 200 ms och därefter göra en ny avläsning. Om fortfarande samma tangent är nedtryckt skall funktionen returnera tangentens nummer.
- Du får förutsätta att det finns en färdig C-funktion:

```
void hold( time_type ms )
```

Denna funktion ger en fördröjning. Den har en parameter som anger hur lång fördröjningen skall vara.

Parameterens typ är deklarerad enligt:

```
typedef unsigned long int time_type;
```

Enheten är millisekunder.

Där inte annat sägs ska du fortsättningsvis förutsätta att följande konventioner gäller vid översättning av kod från 'C' till assemblerspråk.

#### Kompilatorkonvention XCC12:

- Parametrar överförs till en funktion via stacken.
- Då parametrarna placeras på stacken bearbetas parameterlistan från höger till vänster.
- Utrymme för lokala variabler allokeras på stacken. Variablerna behandlas i den ordning de påträffas i koden.
- **Prolog** kallas den kod som reserverar utrymme för lokala variabler.
- **Epilog** kallas den kod som återställer (återlämnar) utrymme för lokala variabler.
- Den del av stacken som används för parametrar *och* lokala variabler kallas *aktiveringspost*.

Beroende på datatyp används för returparameter HC12's register enligt följande tabell:

| Storlek  | Benämning | C-typ                               | Register |
|----------|-----------|-------------------------------------|----------|
| 8 bitar  | byte      | char                                | B        |
| 16 bitar | word      | short int<br><i>och</i><br>pekartyp | D        |
| 32 bitar | long      | long int                            | Y/D      |

2.29 Följande C-deklarationer har gjorts på "toppnivå" (global synlighet):

```
char a,b,c;
char min( char a, char b );
```

- Visa hur variabeldeklarationerna översätts till assemblerdirektiv för HCS12.
- Visa hur följande sats översätts till assemblerkod för HCS12:

```
c = min( a , b );
```

2.30 Följande C-deklarationer har gjorts på "toppnivå" (global synlighet):

```
char *a,*b,*c;
char *min( char *a, char *b );
```

- Visa hur variabeldeklarationerna översätts till assemblerdirektiv för HCS12.
- Visa hur följande sats översätts till assemblerkod för HCS12:

```
c = min( a , b );
```

2.31 Följande C-deklarationer har gjorts på "toppnivå" (global synlighet):

```
int a,b,c;
int min( int a, int b );
```

- Visa hur variabeldeklarationerna översätts till assemblerdirektiv för HCS12.
- Visa hur följande sats översätts till assemblerkod för HCS12:

```
c = min( a , b );
```

2.32 Följande C-deklarationer har gjorts på "toppnivå" (global synlighet):

```
char *cp;
char *identify( char **cp );
```

- Visa hur variabeldeklarationerna översätts till assemblerdirektiv för HCS12.
- Visa hur följande sats översätts till assemblerkod för HCS12:

```
cp = identify( &cp );
```

2.33 Inledningen (parameterlistan och lokala variabler) för en funktion ser ut på följande sätt:

```
void function( int a )
{
    int b;
    .....
```

- Visa hur utrymme för lokala variabler reserveras i funktionen (*prolog*).
- Visa funktionens aktiveringspost, ange speciellt offset för parametrar och lokala variabler.

2.34 Inledningen (parameterlistan och lokala variabler) för en funktion ser ut på följande sätt:

```
void function( char *b, char a )
{
    char *c, *d;
    .....
```

- Visa hur utrymme för lokala variabler reserveras i funktionen (*prolog*).
- Visa funktionens aktiveringspost, ange speciellt offset för parametrar och lokala variabler.

2.35 Inledningen (parameterlistan och lokala variabler) för en funktion ser ut på följande sätt:

```
void function( long c, char b, int a )
{
    char d;
    long e;
    .....
```

- Visa hur utrymme för lokala variabler reserveras i funktionen (*prolog*).
- Visa funktionens aktiveringspost, ange speciellt offset för parametrar och lokala variabler.

2.36 Följande specifikation av en subrutin är given i form av ett C-program. Implementera motsvarande funktion i assemblerspråk för HC12.

```
void f1( unsigned char c )
{
    *( unsigned char *) 0x600 = c ;
    delay();
    c = c >> 1;
    *( unsigned char *) 0x600 = c ;
}
```

2.37 Följande funktion finns given i "C". Implementera motsvarande funktion i assemblerspråk för HC12.

```
#define DATA *( char *) 0x700
#define STATUS *( char *) 0x701
void printerprint( char *s )
{
    while( *s )
    {
        while( STATUS & 1 )
        {}
        DATA = *s;
        s++;
    }
}
```

2.38 Följande specifikation av en subrutin är given i form av ett C-program. Implementera motsvarande funktion i assemblerspråk för HC12.

```
void shortdelay( void )
{
    volatile unsigned char c;
    for( c = 0; c < 0x200 ; c++ );
}
```



- 2.39 Följande specifikation av en subrutin är given i form av ett C-program. Implementera motsvarande funktion i assemblerspråk för HC12.

```
void shortdelay( void )
{
    unsigned char c;
    for( c = 0; c < 0x200 ; c++ );
}
```

- 2.40 Följande specifikation av en subrutin är given i form av ett C-program. Implementera motsvarande funktion i assemblerspråk för HC12.

```
void printchar( char c )
{
    while( *((volatile unsigned char *) 0x600) )
        ;
    *((unsigned char *) 0x400) = c;
}
```

- 2.41 Följande specifikation av en subrutin är given i form av ett C-program. Implementera motsvarande funktion i assemblerspråk för HC12.

```
void printmul( void )
{
    unsigned short int s;
    s = ( unsigned short ) ( *((unsigned char *) 0x600) );
    s = s * ( unsigned short ) ( *((unsigned char *) 0x601) );
    *((unsigned short int *) 0x400) = s;
}
```

- 2.42 Vissa instruktionssekvenser kan inte åstadkommas med hjälp av giltiga standard-C satser. Exempel på detta är att påverka enskilda bitar i processorns statusregister (CCR).

- a) Implementera en assembler subrutin som kan anropas från ett C-program.

```
unsigned char getCCR( void );
```

- returvärdet är innehållet i CCR.

- b) Implementera en assembler subrutin som kan anropas från ett C-program.

```
void setCCR( unsigned char value );
```

- parameter value anger nya värden för bitarna i CCR.

- 2.43 Avbrottsrutiner kan inte implementeras i standard-C men många kompilatorer tillhandahåller möjligheten att lägga in assemblerkod "inline" i C-kod. Följande kod visar sig exempelvis fungera under XCC12:

```
static void shortdelay( void )
{
    unsigned char c;
    for( c = 0; c < 0x200 ; c++ );
}
void take_interrupt(void)
{
    shortdelay();
    _asm( " RTI" );
}
```

Uppmuntrad av resultatet provar vi nu i stället följande, som INTE fungerar som avsett:

```
void take_interrupt(void)
{
    unsigned char c;
    for( c = 0; c < 0x200 ; c++ );
    _asm( " RTI" );
}
```

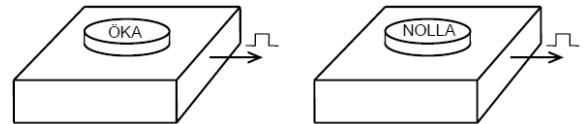
Förklara skillnaden mellan de olika lösningarna.

## 3 Undantagshantering

3.1 Besvara kortfattat följande frågor rörande CPU12.

- Redogör för vad som händer vid RESET och varför detta sker.
- Förklara kortfattat vad som händer vid ett IRQ avbrott om I-flaggan i CC är nollställd.
- Vid IRQ-avbrott sätts I-flaggan automatiskt till 1. Varför sker detta?
- Visa med en instruktionssekvens hur man i en IRQ-avbrottsrutin kan förhindra att processorn utför nya avbrott efter återhopp till det avbrutna programmet.
- Översätt assemblerinstruktionerna CLI och SEI till maskinspråk och visa hur maskinkoden placeras i minnet.
- Assemblerinstruktionerna CLI och SEI kan skrivas på ett alternativt sätt. Visa detta sätt.
- Vilken är skillnaden mellan IRQ- och XIRQ-avbrott? Hur påverkar skillnaden användningen av dem?
- Vid XIRQ-avbrott sätts både X- och I-flaggan automatiskt till 1. Varför sker detta?
- XIRQ-avbrottet är "icke maskbart". Vad innebär detta för möjligheterna att påverka maskbiten X i CRegistret?
- Redogör för vad som händer då en logiknolla läggs på ingången XIRQ' och varför detta sker. Hur påverkas stacken?
- Vilket villkor måste vara uppfyllt för att ett XIRQ-avbrott skall utföras?
- Vad händer med flaggor och stack när instruktionen SWI utförs.
- Förklara hur instruktionen SWI fungerar. Ge ett exempel på hur den kan användas.

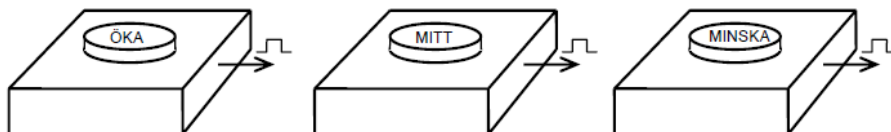
3.2 Två "tryckknappsenheter" enligt figuren skall anslutas till en dator med processorn CPU12. Då en knapp aktiveras genereras en positiv puls på motsvarande utgång. Varje tryckning på ÖKAknappen skall öka en 8-bitars variabel på minnesadressen KNAPP med ett medan varje tryckning på NOLLA-knappen skall nollställa samma variabel. Om innehållet på adressen KNAPP är 255 och ÖKA-knappen trycks ned skall innehållet inte ökas.



De två tryckknapparna skall anslutas så att IRQ-avbrott genereras då någon av dem aktiveras. Inga andra avbrottskällor finns i systemet.

- Visa hur tryckknappsenheterna kan anslutas till datorn. En oanvänd inport finns på adressen \$800. Rita nödvändig logik! D-vippor, NAND- och NOT-grindar får användas.
- Skriv en avbrottsrutin som fungerar enligt beskrivningen ovan. Assemblerspråk för processorn CPU12 skall användas.

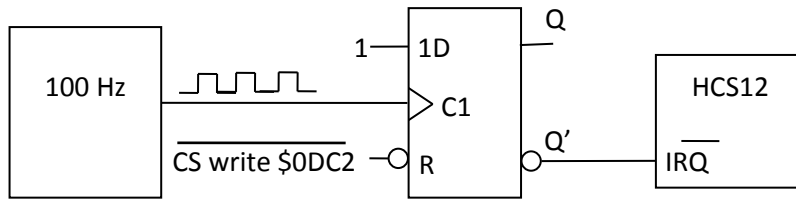
3.3 Tre "tryckknappsenheter" enligt figuren nedan skall anslutas till en dator med processorn CPU12. Då en knapp aktiveras genereras en positiv puls på motsvarande utgång. Varje tryckning på ÖKA- eller MINSKA-knappen skall öka resp. minska en 8-bitars variabel på minnesadressen KNAPP med ett medan varje tryckning på MITT-knappen skall ge samma variabel värdet 128. Innehållet på adressen KNAPP skall dock inte tillåtas att "varva", dvs att ökas från 255 eller minskas från 0.



De tre tryckknapparna skall anslutas så att IRQ-avbrott genereras då någon av dem aktiveras. Inga andra avbrottskällor finns i systemet.

- Visa hur tryckknappsenheterna kan anslutas till datorn. En oanvänd inport finns på adressen \$800. Rita nödvändig logik! D-vippor, NAND- och NOT-grindar får användas.
- Skriv en avbrottsrutin som fungerar enligt beskrivningen ovan. Assemblerspråk för processorn CPU12 skall användas.

- 3.4 En pulsgenerator är ansluten via en avbrottsvippa till IRQ-ingången på ett MC12-system. Pulsgeneratoren har en frekvens på 100 Hz. För att nollställa avbrottsvippan krävs en skrivning på adressen \$0DC2. Pulsgeneratoren är den enda anslutna avbrottskällan till IRQ-ingången på processorn.



- a) Skriv en avbrottsrutin (IRQCNT) som läser en 8-bitars inport (IRQIN, adress \$0600) och adderar det inlästa värdet till en 32-bitars variabel (IRQVAR). Både IRQIN och IRQVAR är variabler på tvåkomplementsform.
- b) Skriv en initieringsrutin IRQINIT som initierar avbrottsystemet och som gör att IRQCNT anropas vid avbrott och att IRQVAR nollställs från början.

- 3.5 Ett konstmuseum övervakas med ett HCS12-baserat mikrodatorsystem. I systemet finns ett antal sensorer utplacerade exempelvis på tavlor samt i dörrar och fönster. Vissa dörrar har också datorstyrda lås. Under öppettiderna ingår såväl kassan som två utplacerade vakter i övervakningen. I kassan och hos vakterna finns larmknappar som är anslutna till HCS12's avbrottsystem enligt figuren nedan.

- a) Vid uppstart med RESET-begäran skall systemet initieras omedelbart. Därför leder reset-vektorn till adressen INIT, som är startadressen för initieringsavsnittet. Såväl resetvektor som avbrottsvektor är redan lagrade i ROM.

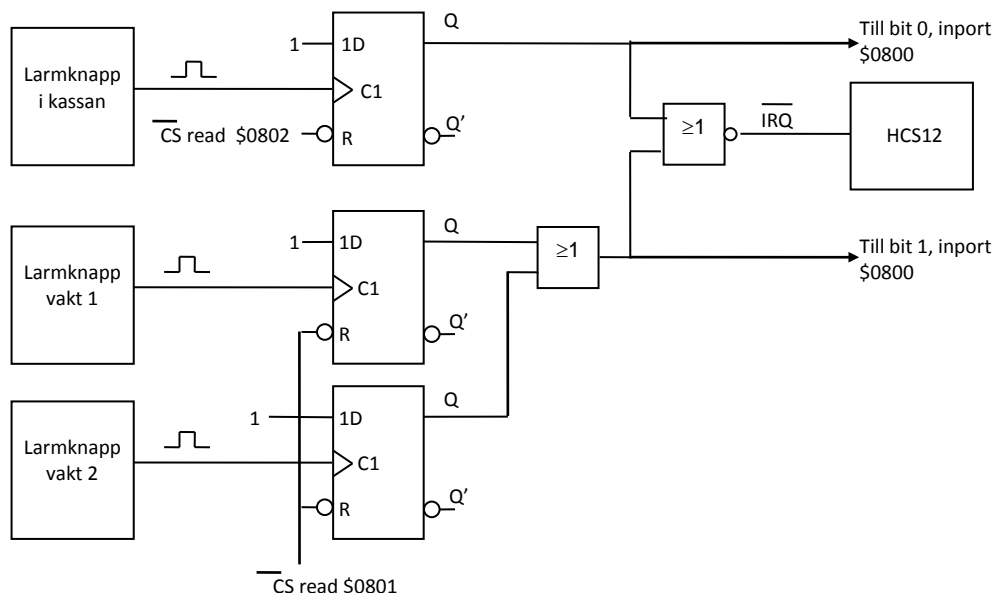
Tänk noga igenom vad som behöver göras i INIT. Du har bl a god hjälp av figuren.

Skriv programavsnittet INIT i HCS12-assemblerspråk, som initierar systemet så att det kan hantera dels övervakningen, dels avbrott på IRQ-ingången. INIT avslutas med hopp till rutinen CONTROL.

- b) Skriv en avbrottsrutin, IRQALARM, som skall avgöra om en avbrottsbegäran kommer från kassan, från vaktställe eller från båda.

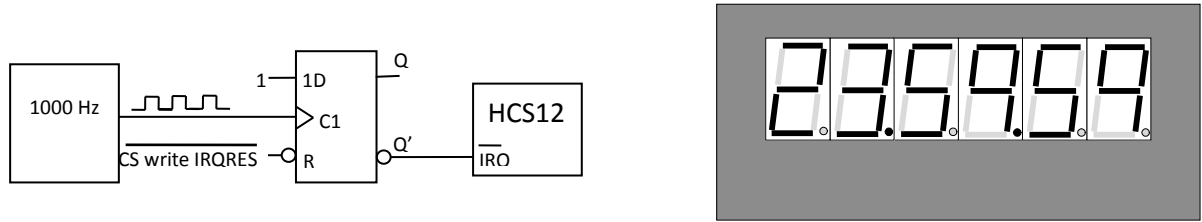
Om avbrottsbegäran enbart kommer från kassan, skall subrutinen ENTRANCE anropas. Om avbrottsbegäran enbart kommer från en vakt, skall subrutinen GUARD anropas. Om avbrottsbegäran kommer från både kassan och en vakt, skall subrutinen CHAOS anropas. Dessa subrutiner finns redan och vidtar de åtgärder som skall göras i respektive fall, exempelvis i form av dörrlåsning och vidarebefordran av larm.

Tänk noga igenom vilka åtgärder som behöver göras i samband med att en avbrotts-begäran betjänas. Din avbrottsrutin skall hantera det som har med avbrottet att göra.





- 3.7 Ett MC12-system är bestyckat med en pulsgenerator som genererar avbrott varje millisekund och en klockmodul som kan visa tid.



Du skall konstruera ett system som räknar ner till "12-slaget" på nyårsafton. För detta krävs en rutin (IRQINIT) som initierar systemet och en avbrottsrutin (IRQ), som anropas varje millisekund, och som minskar en klockvariabel. Klockvariabeln skrivs till en display av huvudprogrammet. Du behöver inte befatta dig med utskriftsrutinen.

När programmet startas skall displayen visa (börja på) 23:59:59. Vi skall räkna ner det sista dygnet, alltså tills displayen visar 00:00:00.

Avbrott kvitteras genom en skrivning på den symboliska adressen IRQRES (se även figur ovan).

Avbrottsrutinen ska uppdatera den symboliska klockvariabeln CLOCK, deklarerad enligt följande:

```
CLOCK RMB 3 ; Variabel innehållande klockan tt:mm:ss
```

där *tt* är timmar (00-23), *mm* är minuter (00-59) och *ss* sekunder (00-59). Alla siffror lagras som NBCD-tal.

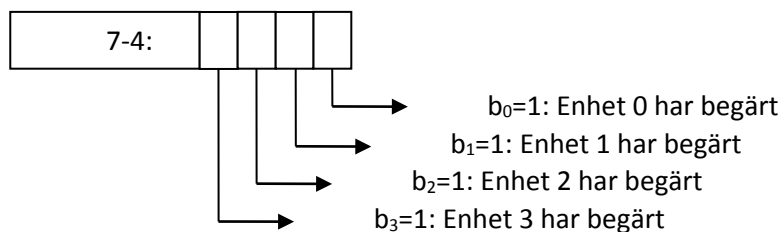
När klockan räknat ner till noll skall den stanna och huvudprogrammet fortsätta som vanligt.

Initieringsrutinen (IRQINIT): ska initiera nödvändiga variabler, dvs. ställa initial tid och i övrigt förbereda systemet för att ta emot och behandla avbrott. Det finns inga andra avbrottskällor i systemet.

Du får själv skapa ytterligare hjälpvariabler för klockavbrotten efter behov. Systemets avbrottsvektor IRQ finns is RWM på adress \$3FF2.

- Skriv initieringsrutinen IRQINIT
- Skriv avbrottsrutinen IRQ

- 3.8 Kalle student konstruerar yttre enheter till ett MC12-system. Konstruktionen visar sig innehålla vissa brister. Detta diskuteras i deluppgifter b,c och d nedan). Läs därför igenom hela uppgiften innan du börjar lösa den. Systemet skall användas för att betjäna fyra yttre enheter numrerade 0 t o m 3. Oberoende av varandra kan enheterna begära avbrott . Begäran om avbrott görs genom att en till enheten hörande statusflagga ettställs. Enheternas statusflaggor, som också numreras 0 t o m 3, har i ordningsföljd samlats i bitarna 0 - 3 av ett statusregister på adress \$700. Se figur. Oberoende av vilken statusflagga som ettställs så skickas en avbrottsignal (IRQ) till processorn. Vid en skrivning på adress \$700 nollställs statusflaggorna.



Enheternas servicerutiner finns tillgängliga och har lagrats som subrutiner med namnen DSR0 – DSR3.

- Skriv en avbrottshanterare som undersöker vilken enhet som begärt avbrott och anropar tillhörande avbrottsrutin.
- Det visar sig att Kalles konstruktion inte upptäcker alla avbrott i vissa sammanhang. När inträffar detta?
- Vad kan göras i mjukvara för att minska risken för detta?
- Vad kan göras i hårdvara för att eliminera problemet?

3.9 Antag att en dator används för enkel tidtagning vid en idrottstävling. Till datorn finns kopplat två sensorer samt en klockkrets. (Dessutom finns en display, men den behöver inte programmeras i denna uppgift.) Den första sensorn känner när startskottet går och den andra när den tävlande passerar mållinjen. De två sensorerna är kopplade till *samma* 16-bitars styrregister. Detta ligger på adressen 1234 (hex) och adressen till dess avbrottsvektor är FF80. Styrregistret aktiveras och inaktiveras genom att bit nr 0 i det sätts till 1 resp. 0. Om registret är inaktiverat påverkas det inte av inkommande signaler, men om det är aktiverat gäller följande: När en signal kommer från någon av de två sensorerna sätts bit nr 7 i registret till 1. Om man har satt bit nr 6 i registret till 1 genereras då även en avbrottssignal till processorn. Styrregistret skall återställas efter ett avbrott genom att man sätter bit 7 till 0.

Klockkretsen är kopplad till ett annat 16-bitars styrregister, vilket ligger på adressen 1230 (hex). Adressen till dess avbrottsvektor är FF70. Detta styrregister har samma konfiguration och fungerar på samma sätt som styrregistret för sensorerna. Den enda skillnaden är de inkommande signalerna kommer från klockkretsen istället för sensorerna. Klockkretsen genererar 500 signaler per sekund.

Uppgiften är att skriva ett C-program som gör en tidsmätning. När programmet startar skall det visa tiden 0 på en display och vänta tills startskottet går. När detta sker skall klockan aktiveras och tiden skall visas fortlöpande på displayen. Displayen skall visa tiden uttryckt i hundradels sekunder och den visade tiden skall uppdateras hundra gånger per sekund. När den tävlande passerar mållinjen skall klockan stoppas och sluttiden visas konstant på displayen. Programmet behöver bara klara en tidsmätning. (Vill man göra en ny får man starta om programmet genom att trycka på reset-knappen.)

Du får förutsätta att det finns en färdigskrivna C-funktion med namnet `display`. Denna har en parameter av typen **long int** och när den anropas visar den parameterns värde på en display.

Du får också förutsätta att följande två färdigskrivna assemblerrutiner finns:

```

                segment    text
                define     _clocktrap
                define     _sensortrap
_clocktrap:    JSR         _clockinter
                RTI
__sensortrap:  JSR         _sensorinter
                RTI

```

Det finns också en färdigskrivna assemblerrutin som anropar funktionen `main` när processorn startar. Skriv resten av programmet (i C).

## 4 Systemprogrammering och periferikretsar

- 4.1 Parallellporten *Port P*, i ett HCS12-system kan programmeras så att varje bit kan utgöra antingen en insignal, eller en utsignal. Porten har två olika register, som specificeras enligt följande:

| Parallel port P (PORTP) |   |       |       |       |       |       |       |       |       |          |                         |
|-------------------------|---|-------|-------|-------|-------|-------|-------|-------|-------|----------|-------------------------|
| Address                 |   | 7     | 6     | 5     | 4     | 3     | 2     | 1     | 0     | Mnemonic | Namn                    |
| \$700                   | R | 1=OUT | 1=OUT | 1=OUT | 1=OUT | 1=OUT | 1=OUT | 1=OUT | 1=OUT | DDR      | Data Direction Register |
|                         | W | 0=IN  | 0=IN  | 0=IN  | 0=IN  | 0=IN  | 0=IN  | 0=IN  | 0=IN  |          |                         |
| \$701                   | R | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | DATA     | Data Register           |
|                         | W | 1     | 1     | 1     | 1     | 1     | 1     | 1     | 1     |          |                         |

I figuren anges registrens innehåll efter "RESET".

- DDR: 1 anger att positionen är en utsignal, 0 anger att positionen är en insignal. Bitarna kan programmeras oberoende av varandra, dvs. godtycklig kombination av insignaler och utsignaler kan åstadkommas. Registret är både skrivbart och läsbart i sin helhet.
  - DATA: Består i själva verket av två olika register (R,W):
    - R: innehåller insignaler för de bitar som programmerats som insignaler. Endast 0 får skrivas, till en bit som är programmerad som insignal.
    - W: används då biten är programmerad som en utsignal. Då en bit som är programmerad som utsignal läses kommer detta alltid att resultera i värdet 1, oavsett vilket värde som tidigare skrivits till databiten.
- a) Visa en lämplig deklaration av porten med användning av en `struct`. Visa också en funktion, `void portPinit(void)` som initierar port P så att bitarna `b7-b5` används som en 3-bitars inport och bitarna `b4-b0` används som en 5-bitars utport.
- b) Visa en funktion, `void outPortP(unsigned char c)` som matar ut bitarna `b4-b0`, av `c`, till port P.
- c) Visa en funktion, `unsigned char inPortP(void)` som returnerar bitarna `b7-b5` hos port P som en `unsigned char`, dvs. värden i intervallet 0 t.o.m. 7.

- 4.2 Parallellporten *Port P*, i ett HCS12-system kan programmeras så att varje bit kan utgöra antingen en insignal, eller en utsignal. Portarna som används för insignaler kan dessutom konfigureras så att ett avbrott genereras då en yttre enhet ändrat värdet hos insignalen.

Porten har tre olika register, som specificeras enligt följande:

| Parallel port P (PORTP) |   |       |       |       |       |       |       |       |       |          |                         |
|-------------------------|---|-------|-------|-------|-------|-------|-------|-------|-------|----------|-------------------------|
| Address                 |   | 7     | 6     | 5     | 4     | 3     | 2     | 1     | 0     | Mnemonic | Namn                    |
| \$700                   | R | 1=OUT | 1=OUT | 1=OUT | 1=OUT | 1=OUT | 1=OUT | 1=OUT | 1=OUT | DDR      | Data Direction Register |
|                         | W | 0=IN  | 0=IN  | 0=IN  | 0=IN  | 0=IN  | 0=IN  | 0=IN  | 0=IN  |          |                         |
| \$701                   | R | IF    | IF    | IF    | IF    | IF    | IF    | IF    | IF    | ICIE     | Input Change Interrupt  |
|                         | W | IEA   | IEA   | IEA   | IEA   | IEA   | IEA   | IEA   | IEA   |          |                         |
| \$702                   | R | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | DATA     | Data Register           |
|                         | W | 1     | 1     | 1     | 1     | 1     | 1     | 1     | 1     |          |                         |

- DDR: 1 anger att positionen är en utsignal, 0 anger att positionen är en insignal. Bitarna kan programmeras oberoende av varandra, dvs. godtycklig kombination av insignaler och utsignaler kan åstadkommas. Registret är både skrivbart och läsbart i sin helhet.
  - ICIE: Består av olika delar (R=IF/W=IEA).
    - IEA (Interrupt Enable/Acknowledge). Biten är 0 efter RESET. Då 1 (Interrupt Enable) skrivs till biten aktiveras avbrottsgenerering vid ändring av motsvarande bit i DATA-registret om denna programmerats som insignal. Om motsvarande bit i DDR i stället programmerats som utsignal, genereras inga avbrott. IEA-biten har då ingen funktion. Då 1 skrivs till en bit som tidigare satts till 1, fungerar detta i stället som en Interrupt Acknowledge-funktion, dvs. IF (Interrupt Flag) nollställs. För att helt återställa avbrottsmekanismen för denna bit i DATA-registret skrivs 0 till IEA.
    - IF (Interrupt Flag) Biten är 0 efter RESET. Då motsvarande bit i DDR är programmerad som en insignal och motsvarande IEA är 1, sätts IF till 1 och ett avbrott (IRQ) genereras, avbrottsvektor FFF2.
  - DATA: Består i själva verket av två olika register (R,W):
    - R: innehåller insignaler för de bitar som programmerats som insignaler. Endast 0 får skrivas, till en bit som är programmerad som insignal.
    - W: används då biten är programmerad som en utsignal. Då en bit som är programmerad som utsignal läses kommer detta alltid att resultera i värdet 1, oavsett vilket värde som tidigare skrivits till databiten.
- a) Visa en lämplig deklaration av porten med användning av en `struct`. Visa också en funktion, `void portPinit(void)` som initierar port P, på adress 0x700 i minnet, så att bitarna  $b_7$ - $b_4$  används som en 4-bitars inport och bitarna  $b_3$ - $b_0$  används som en 4-bitars utport. Då någon av inportens bitar ändras ska avbrott genereras.
- b) Visa en funktion, `void outPortP(unsigned char c)` som matar ut bitarna  $b_3$ - $b_0$ , av `c`, till port P.
- c) Visa hur du implementerar en avbrottsfunktion, `void irqPortP(void)` som kvitterar ett avbrott från någon av portens ingångar.
- d) Visa nödvändiga programdelar i assemblerspråk, dvs. hur avbrottsrutinen definieras, avbrottsvektorn initieras (antag att FFF2 är läs- och skrivbart minne) och hur processorn förbereds för att acceptera avbrotten i ett huvudprogram. Använd endast standard-C konstruktioner och/eller assemblerspråk för HCS12.

```
a)
typedef struct sPortP{
    volatile unsigned char ddr;
    volatile unsigned char icie;
    volatile unsigned char data;
}PORTP;
#define PORTP_BASE 0x700
#define portP ((PORTP *) (PORTP_BASE))
```



```
void portPinit( void )
{
    portP->ddr = 0x0F;    /* b7-b4 inport, b3-b0 utport */
    portP->icie = 0xF0;   /* b7-b4 inportar, avbrott aktiveras */
}
```

b)

```
void outPortP( unsigned char c )
{
    portP->data = c & 0x0F ; /* b7-b4 ska vara 0 */
}
```

c)

```
void irqPortP( void )
{
    switch( portP->data & 0xF0 ) /* bestäm avbrottskälla */
    { /* kvittera avbrott */
        case 0x80: portP-> icie = 0x80; break;
        case 0x40: portP-> icie = 0x40; break;
        case 0x20: portP-> icie = 0x20; break;
        case 0x10: portP-> icie = 0x10; break;
    }
}
```

d)

Assembler:

; initieringar i huvudprogram...

IMPORT \_irqPortP

MOVW #PortPirq,\$FFF2

CLI

; avbrottsrutin

PortPirq:

JSR \_irqPortP

RTI

#### 4.3 Följande figur beskriver de register som används för att styra PLL-kretsen hos HCS12:

| Clock Reset Generator (CRG) |   |        |      |        |        |        |        |        |        |          |                           |
|-----------------------------|---|--------|------|--------|--------|--------|--------|--------|--------|----------|---------------------------|
| Address                     |   | 7      | 6    | 5      | 4      | 3      | 2      | 1      | 0      | Mnemonic | Namn                      |
| \$34                        | R | 0      | 0    | SYN5   | SYN4   | SYN3   | SYN2   | SYN1   | SYN0   | SYNR     | Synthesizer Register      |
|                             | W |        |      |        |        |        |        |        |        |          |                           |
| \$35                        | R | 0      | 0    | 0      | 0      | REFDV3 | REFDV2 | REFDV1 | REFDV0 | REFDV    | Reference Divide Register |
|                             | W |        |      |        |        |        |        |        |        |          |                           |
| \$37                        | R |        |      |        |        |        |        |        | SCM    | CRGFLG   | Flags Register            |
|                             | W | RTIF   | PORF | LVRF   | LOCKIF | LOCK   | SCMIE  | SCMIF  |        |          |                           |
| \$39                        | R |        |      |        |        |        |        |        |        | CLKSEL   | Clock Select Register     |
|                             | W | PLLSEL | PSTP | SYSWAI | ROAWAI | PLLWAI | CWAI   | RTIWAI | COPWAI |          |                           |

Vårt system har en 10 MHz oscillator. PLL-kretsen ska programmeras för att generera busfrekvensen 25 MHz.

a) Visa en subrutin PLLINIT i assemblerspråk, alla adresser och bitar ska definieras med sina symbolnamn enligt figuren.

Följande figur ger en översikt av hela CRG-modulen.

| Adress | Mnemonic | Namn |
|--------|----------|------|
|--------|----------|------|

|      |        |                                  |
|------|--------|----------------------------------|
| \$34 | SYNR   | Synthesizer Register             |
| \$35 | REFDV  | Reference Divide Register        |
| \$36 | CTFLG  | *)Test Flags Register            |
| \$37 | CRGFLG | Flags Register                   |
| \$38 | CRGINT | Interrupt Enable Register        |
| \$39 | CLKSEL | Clock Select Register            |
| \$3A | PLLCTL | PLL Control Register             |
| \$3B | RTICTL | RTI Control Register             |
| \$3C | COPCTL | COP Control Register             |
| \$3D | FORBYP | *)Force and Bypass Test Register |
| \$3E | CTCTL  | *)Test Control Register          |
| \$3F | ARMCOP | COP Arm/Timer Reset              |

b) Visa en typdeklaration i för hela CRG-modulen, i form av en 'C'-struct, enligt följande:

```
typedef struct sCRG{
    ...
    ...
}CRG, *PCRGR ;
```

- c) Använd typdeklarationen i b) och visa en C-funktion `void InitPLL(void)`. Definiera och använd lämpliga symboliska namn för alla konstanter.
- d) Komplettera typdeklarationen från b) för de register som används av PLL-kretsen så att bitar och grupper av bitar deklarerats som bitfält.
- e) Använd typdeklarationen i d) och visa en C-funktion `void InitPLL2(void)`. Definiera och använd lämpliga symboliska namn för alla konstanter.

4.4 Följande figur beskriver register som används för den enkla realtidsklockan hos HCS12 (se även figuren med översikt av CRG-modulen i uppgift 4.1):

| Clock Reset Generator (CRG) |   |      |      |      |        |      |       |       |      |          |                           |
|-----------------------------|---|------|------|------|--------|------|-------|-------|------|----------|---------------------------|
| Offset                      |   | 7    | 6    | 5    | 4      | 3    | 2     | 1     | 0    | Mnemonic | Namn                      |
| \$37                        | R | RTIF | PORF | LVRF | LOCKIF | LOCK | SCMIE | SCMIF | SCM  | CRGFLG   | Flags Register            |
|                             | W |      |      |      |        |      |       |       |      |          |                           |
| \$38                        | R | RTIE | 0    | 0    | LOCKIE | 0    | 0     | SCMIE | 0    | CRGINT   | Interrupt Enable Register |
|                             | W |      |      |      |        |      |       |       |      |          |                           |
| \$3B                        | R | 0    | RTR6 | RTR5 | RTR4   | RTR3 | RTR2  | RTR1  | RTR0 | RTICTL   | RTI Control Register      |
|                             | W |      |      |      |        |      |       |       |      |          |                           |

| RTR [3:0] | RTR[6:4]  |                    |                    |                    |                    |                    |                    |                    |
|-----------|-----------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|
|           | 000 (OFF) | 001                | 010                | 011                | 100                | 101                | 110                | 111                |
| 0000      | OFF       | 2 <sup>10</sup>    | 2 <sup>11</sup>    | 2 <sup>12</sup>    | 2 <sup>13</sup>    | 2 <sup>14</sup>    | 2 <sup>15</sup>    | 2 <sup>16</sup>    |
| 0001      | OFF       | 2x2 <sup>10</sup>  | 2x2 <sup>11</sup>  | 2x2 <sup>12</sup>  | 2x2 <sup>13</sup>  | 2x2 <sup>14</sup>  | 2x2 <sup>15</sup>  | 2x2 <sup>16</sup>  |
| 0010      | OFF       | 3x2 <sup>10</sup>  | 3x2 <sup>11</sup>  | 3x2 <sup>12</sup>  | 3x2 <sup>13</sup>  | 3x2 <sup>14</sup>  | 3x2 <sup>15</sup>  | 3x2 <sup>16</sup>  |
| 0011      | OFF       | 4x2 <sup>10</sup>  | 4x2 <sup>11</sup>  | 4x2 <sup>12</sup>  | 4x2 <sup>13</sup>  | 4x2 <sup>14</sup>  | 4x2 <sup>15</sup>  | 4x2 <sup>16</sup>  |
| 0100      | OFF       | 5x2 <sup>10</sup>  | 5x2 <sup>11</sup>  | 5x2 <sup>12</sup>  | 5x2 <sup>13</sup>  | 5x2 <sup>14</sup>  | 5x2 <sup>15</sup>  | 5x2 <sup>16</sup>  |
| 0101      | OFF       | 6x2 <sup>10</sup>  | 6x2 <sup>11</sup>  | 6x2 <sup>12</sup>  | 6x2 <sup>13</sup>  | 6x2 <sup>14</sup>  | 6x2 <sup>15</sup>  | 6x2 <sup>16</sup>  |
| 0110      | OFF       | 7x2 <sup>10</sup>  | 7x2 <sup>11</sup>  | 7x2 <sup>12</sup>  | 7x2 <sup>13</sup>  | 7x2 <sup>14</sup>  | 7x2 <sup>15</sup>  | 7x2 <sup>16</sup>  |
| 0111      | OFF       | 8x2 <sup>10</sup>  | 8x2 <sup>11</sup>  | 8x2 <sup>12</sup>  | 8x2 <sup>13</sup>  | 8x2 <sup>14</sup>  | 8x2 <sup>15</sup>  | 8x2 <sup>16</sup>  |
| 1000      | OFF       | 9x2 <sup>10</sup>  | 9x2 <sup>11</sup>  | 9x2 <sup>12</sup>  | 9x2 <sup>13</sup>  | 9x2 <sup>14</sup>  | 9x2 <sup>15</sup>  | 9x2 <sup>16</sup>  |
| 1001      | OFF       | 10x2 <sup>10</sup> | 10x2 <sup>11</sup> | 10x2 <sup>12</sup> | 10x2 <sup>13</sup> | 10x2 <sup>14</sup> | 10x2 <sup>15</sup> | 10x2 <sup>16</sup> |
| 1010      | OFF       | 11x2 <sup>10</sup> | 11x2 <sup>11</sup> | 11x2 <sup>12</sup> | 11x2 <sup>13</sup> | 11x2 <sup>14</sup> | 11x2 <sup>15</sup> | 11x2 <sup>16</sup> |
| 1011      | OFF       | 12x2 <sup>10</sup> | 12x2 <sup>11</sup> | 12x2 <sup>12</sup> | 12x2 <sup>13</sup> | 12x2 <sup>14</sup> | 12x2 <sup>15</sup> | 12x2 <sup>16</sup> |
| 1100      | OFF       | 13x2 <sup>10</sup> | 13x2 <sup>11</sup> | 13x2 <sup>12</sup> | 13x2 <sup>13</sup> | 13x2 <sup>14</sup> | 13x2 <sup>15</sup> | 13x2 <sup>16</sup> |
| 1101      | OFF       | 14x2 <sup>10</sup> | 14x2 <sup>11</sup> | 14x2 <sup>12</sup> | 14x2 <sup>13</sup> | 14x2 <sup>14</sup> | 14x2 <sup>15</sup> | 14x2 <sup>16</sup> |
| 1110      | OFF       | 15x2 <sup>10</sup> | 15x2 <sup>11</sup> | 15x2 <sup>12</sup> | 15x2 <sup>13</sup> | 15x2 <sup>14</sup> | 15x2 <sup>15</sup> | 15x2 <sup>16</sup> |
| 1111      | OFF       | 16x2 <sup>10</sup> | 16x2 <sup>11</sup> | 16x2 <sup>12</sup> | 16x2 <sup>13</sup> | 16x2 <sup>14</sup> | 16x2 <sup>15</sup> | 16x2 <sup>16</sup> |

Vårt system har en 10 MHz oscillator. Realtidsklockan ska programmeras för att generera periodiska avbrott med c:a 10ms intervall. *Ledning*:  $3 \times 2^{15}$  pulser/period ger tillräcklig noggrannhet. Programpaketet ska bestå av delar implementerade såväl i assemblerspråk som i 'C'.

En "servicerutin" `void AtRTIrq(void)`, i 'C', ska anropas från en avbrottsrutin `RTIRQ`.

Initieringsrutiner för klockfunktionen ska finnas både i assemblerspråk och 'C'.

- Implementera en subrutin `RTINIT` i assemblerspråk, alla adresser och bitar ska definieras med sina symbolnamn enligt figuren. Använd ledningen ovan för tidbasen.
- Vad blir den *verkliga* periodtiden?
- Implementera avbrottsrutinen `RTIRQ` som ska:
  - Kvittera avbrottet,
  - utföra `AtRTIrq`.
- Implementera en C-funktion `void RTInit(void)`. Definiera och använd lämpliga symboliska namn för alla konstanter. Använd typdeklaration från uppgift 4.1.

C-funktionen `AtRTIrq` ska implementera en realtidsklocka, som underhåller en global variabel `RealTime` deklarerad enligt följande:

```
REAL_TIME_TYPE RealTime;
där:
```

```
typedef struct tRealTime {
    int t_irq;
    int t_sec;
    int t_min;
    int t_hour;
} REAL_TIME_TYPE;
```

Du behöver inte ta hänsyn till begynnelsevärden.

- Implementera funktionen `AtRTIrq`.

## 4.5 Följande figur beskriver register som används för seriekommunikationskretsen (SCI) hos HCS12:

| Serial Communication Interface (SCI) |   |       |         |      |       |       |       |       |      |          |                         |
|--------------------------------------|---|-------|---------|------|-------|-------|-------|-------|------|----------|-------------------------|
| Adress                               |   | 7     | 6       | 5    | 4     | 3     | 2     | 1     | 0    | Mnemonic | Namn                    |
| \$C8                                 | R | 0     | 0       | 0    | SBR12 | SBR11 | SBR10 | SBR9  | SBR8 | SCIBDH   | Baud Rate Register High |
|                                      | W |       |         |      |       |       |       |       |      |          |                         |
| \$C9                                 | R | SBR7  | SBR6    | SBR5 | SBR4  | SBR3  | SBR2  | SBR1  | SBR0 | SCIBDL   | Baud Rate Register Low  |
|                                      | W |       |         |      |       |       |       |       |      |          |                         |
| \$CA                                 | R | LOOPS | SCISWAI | RSRC | M     | WAKE  | ILT   | PE    | PT   | SCICR1   | Control Register 1      |
|                                      | W |       |         |      |       |       |       |       |      |          |                         |
| \$CB                                 | R | TIE   | TCIE    | RIE  | ILIE  | TE    | RE    | RWU   | SBK  | SCICR2   | Control Register 2      |
|                                      | W |       |         |      |       |       |       |       |      |          |                         |
| \$CC                                 | R | TDRE  | TC      | RDRF | IDLE  | OR    | NF    | FE    | PF   | SCISR1   | Status Register 1       |
|                                      | W |       |         |      |       |       |       |       |      |          |                         |
| \$CD                                 | R | 0     | 0       | 0    | 0     | 0     | BRK13 | TXDIR | RAF  | SCISR2   | Status Register 2       |
|                                      | W |       |         |      |       |       |       |       |      |          |                         |
| \$CE                                 | R | R8    | T8      | 0    | 0     | 0     | 0     | 0     | 0    | SCIDRH   | Data Register High      |
|                                      | W |       |         |      |       |       |       |       |      |          |                         |
| \$CF                                 | R | R7    | R6      | R5   | R4    | R3    | R2    | R1    | R0   | SCIDRL   | Data Register Low       |
|                                      | W | T7    | T6      | T5   | T4    | T3    | T2    | T1    | T0   |          |                         |

Enkla drivrutiner för denna seriekrets ska konstrueras, rutinerna specificeras av följande:

```
void serial_init( void ); /* initiera gränssnittet */
void serial_out( char c ); /* skicka ett tecken via gränssnittet */
char serial_in( void ); /* ta mot ett tecken från gränssnittet */
serial_in ska använda "busy wait", dvs. alltid returnera ett tecken.
```

PLL-klockan har initierats för 25 MHz frekvens. Välj överföringshastigheten 57600 baud. Sambandet mellan baudrate och det värde  $BR$  (1-8191) som ska skrivas till SCIBDH/SCIBDL är:

$$BR = \frac{PLLCLK}{16 \times baudrate}$$

- Bestäm värdet  $BR$ .
- Implementera en subrutin `SERIAL_INIT` i assemblerspråk, alla adresser och bitar ska definieras med sina symbolnamn enligt figuren ovan.
- Skapa en lämplig typdeklaration (C-struct) för seriekretsen, använd symboliska namn enligt figuren ovan.
- Implementera `void serial_init( void )` i 'C'. Definiera och använd lämpliga symboliska namn för alla konstanter. Använd typdeklarationen från c).
- Implementera en subrutin `SERIAL_IN` i assemblerspråk, alla adresser och bitar ska definieras med sina symbolnamn enligt figuren ovan. Returvärdet ska skickas i register B.
- Implementera `char serial_in( void )` i 'C'. Definiera och använd lämpliga symboliska namn för alla konstanter. Använd typdeklarationen från c).
- Implementera en subrutin `SERIAL_OUT` i assemblerspråk, alla adresser och bitar ska definieras med sina symbolnamn enligt figuren ovan. Tecken som ska skrivas ut förutsätts finnas i register B vid anrop.
- Implementera `void serial_out( char c )` i 'C'. Definiera och använd lämpliga symboliska namn för alla konstanter. Använd typdeklarationen från c).

# 1 Grundläggande assemblerprogrammering

1.1

```

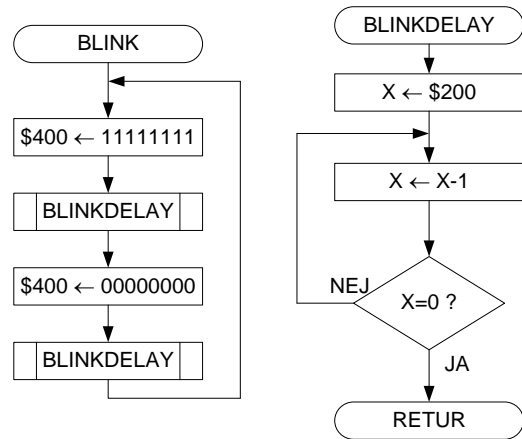
;
; BlinkandeLjus1.s12
;

main:    ORG    $1000
        JSR    BlinkB
        BRA    main

; uppgift a)
BlinkA: LDAA    #$FF
        STAA   $400
        LDAA    #0
        STAA   $400
        BRA    BlinkA

; uppgift b)
BlinkB:  LDAA    #$FF
        STAA   $400
        JSR    BLINKDELAY
        LDAA    #0
        STAA   $400
        JSR    BLINKDELAY
        BRA    BlinkB

BLINKDELAY:
        LDX    #$200
BLINKDELAY1:
        LEAX   -1,X
        CPX    #0
        BNE    BLINKDELAY1
        RTS
    
```



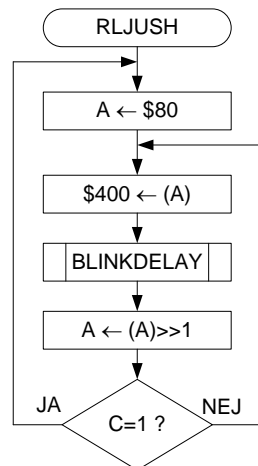
c)

1.2 a), b)

```

RLJUSH:
        LDAA    #$80
RLJUSH1:
        STAA   $400
        JSR    BLINKDELAY
        RORA
        BCS    RLJUSH
        BRA    RLJUSH1
    
```

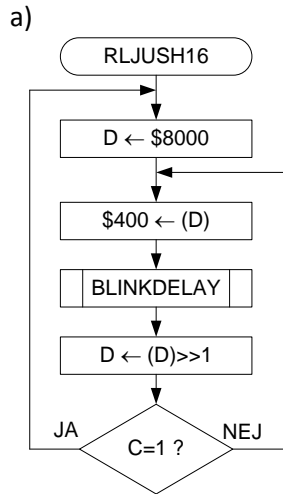
c)



1.3

```

b)
RLJUSH16:
    LDD    #$8000
RLJUSH16_1:
    STD    $400
    JSR    BLINKDELAY
    RORA
    RO RB
    BCS    RLJUSH16
    BRA    RLJUSH16_1
    
```

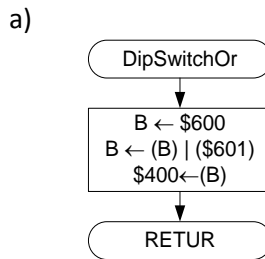


1.4

```

b)
main:
    JSR    DipSwitchOr
    BRA    main

DipSwitchOr:
    LDAB   $600
    ORAB   $601
    STAB   $400
    RTS
    
```



1.5

```

; Symboliska adresser
DipSwitch EQU    $600
HexDisp   EQU    $400

; Subrutin DipHex
DipHex:    LDAA   DipSwitch
           CLRB

DipHex10: TSTA
           BEQ    DipHex20

           INCB
           LSRA
           BCC    DipHex10

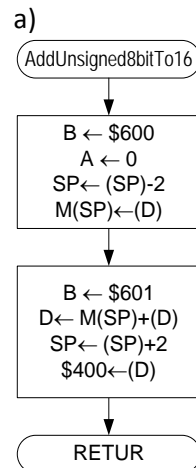
DipHex20: STAB   $400
           RTS
    
```

1.6

```

b)
; Huvudprogram
main:     JSR    AddUnsigned8bitTo16
           BRA    main

; Subrutin
AddUnsigned8bitTo16:
    LDAB   $600
    CLRA
    PSHD
    LDAB   $601
    ADDD   2, SP+
    STD    $400
    RTS
    
```



1.7

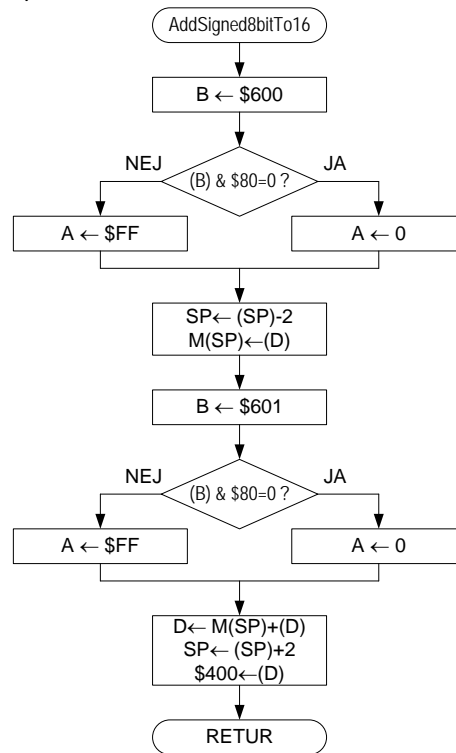
b)

```

; Huvudprogram
main:   JSR    AddSigned8bitTo16
        BRA    main

; Subrutin
AddSigned8bitTo16:
        LDAB  $600
        SEX   B,D
        PSHD
        LDAB  $601
        SEX   B,D
        ADDD  2,SP+
        STD   $400
        RTS
    
```

a)



1.8

```

DipSwitch: EQU    $600
HexDisp:   EQU    $400

DipHex2:   LDD    DipSwitch
           MUL
           STD    HexDisp
           RTS
    
```

1.9

```

ML4_INPUT: EQU    $0600
ML4_OUTPUT: EQU    $0400
ERROR_CODE: EQU    $5D

DisplayNBCD: LDX    #SegCodes      ; Pekare
DisplayNBCD1: LDAA   ML4_INPUT      ; Läs strömbrytare
              BPL    DisplayNBCD2  ; Om Bit 7=0
              ; Och ja, LDAA påverkar faktiskt både
              ; N och Z flaggan
              CLR    ML4_OUTPUT     ; Släck
              BRA    DisplayNBCD1

DisplayNBCD2: ANDA   #$0F          ; Maska fram b3-b0
              CMPA   #9            ; [0,9]?
              BHI    DisplayNBCD3
              LDAB  A,X            ; Hämta segmentkod för [0,9]
              STAB  ML4_OUTPUT     ; Visa siffra
              BRA    DisplayNBCD1

DisplayNBCD3: LDAB  #ERROR_CODE    ; Kod för E
              STAB  ML4_OUTPUT     ; Visa siffra
              BRA    DisplayNBCD1

SegCodes    FCB    $77,$22,$5B,$6B,$2E,$6D,$7D,$23
           FCB    $7F,$6F,$3F,$7C,$55,$7A,$5D,$18
    
```

## 1.10

```

Inport      EQU      $600
UtportP     EQU      $400
UtportQ     EQU      $401
UtportR     EQU      $402
ERROR_CODE: EQU      $5D          ; Segmentkod för E (Error)

SumPQ:      LDX      #SegCodes    ; Pekare till tabell
            LDAB     Inport       ; Läs inporten
            TFR      B,A         ; Kopiera
            LSRA     ; Skifta fram P
            LSRA
            LSRA
            MOVb     A,X,UtportP   ; Skriv P
            ANDB     #$0F         ; Maska fram Q
            MOVb     B,X,UtportQ   ; Skriv Q
            ABA      ; Summan R
            CMPA     #10          ; Giltigt värde
            BLO      SumPQ_1      ; ..hoppa om JA
            LDAB     #ERROR_CODE   ; Skriv Error
            STAB     UtportR
            BRA      SumPQ

SumPQ_1:    LDAB     A,X          ; Översätt R till Segmentkod
            STAB     UtportR      ; .. och skriv ut
            BRA      SumPQ

SegCodes    FCB      $77,$22,$5B,$6B,$2E,$6D,$7D,$23
            FCB      $7F,$6F,$3F,$7C,$55,$7A,$5D,$18

```

## 1.11 \* Subrutin Display visar ett NBCD-tal i A på två sifferindikatorer

```

* Indata: Register A, Ett NBCD-tal [0,99]
Display:    LDX      #SegCodes    ; Pekare
            TFR      A,B         ; Spara kopia
* Register A används för EN-talen och...
* Register B används för TIO-talen
            LSRB     ; Skifta fram TIO-talen
            LSRB
            LSRB
            LSRB
            LDAB     B,X          ; ..och visa TIO-talen
            STAB     UtPort1
            ANDA     #$0F         ; Ta fram EN-talen
            LDAA     A,X          ; ..och visa EN-talen
            STAA     UtPort2
            RTS

* Subrutin Read laser inporten, spegelvänder data och lämnar detta i Register A
Read        LDAB     InPort
            LDX      #8          ; Skifta 8 bitar

Read_1:     LSRB     ; Skifta ut...
            ROLA     ; ... och in
            DEX      Sista?
            BNE      Read_1      ; Nej
            RTS

SegCodes    FCB      $77,$22,$5B,$6B,$2E,$6D,$7D,$23
            FCB      $7F,$6F,$3F,$7C,$55,$7A,$5D,$18

```



## 2 Grundläggande programmering i 'C'

2.4

`auto`: synlig endast i den funktion den deklarerats.  
`static`: synlig endast i den källtextfil den deklarerats.  
`global`: synlig från alla programdelar.

2.5

```
int          a;
static int  sia;
extern int  eia;

void f( void)
{
    int      b;
    auto int  aib;
    static int sib;
    extern int eib;
}
```

2.6

a) (( **unsigned char** \*) 0x400)  
 b) (( **signed char** \*) 0x400)  
 c) (( **unsigned short** \*) 0x400)  
 d) (( **signed short** \*) 0x400)

2.7

```
#include "stdint.h"
a) (( uint8_t *) 0x400)
b) (( int8_t *) 0x400)
c) (( uint16_t *) 0x400)
d) (( int16_t *) 0x400)
```

2.8

a)  
**typedef void** (\* function1 )(void);  
 #define reentry ((function1) (0xC00F))  
 b)  
**typedef void** (\* function2 )(char);  
 #define outcha ((function2) (0xC006))  
 c)  
**typedef char** (\* function3 )(void);  
 #define tstcha ((function3) (0xC003))

2.9

```
#include <stdio.h>
#include <string.h>

main()
{
    char a[9], s[11] = "20 - - ";
    printf("Ange datum som mm/dd/yy ");
    gets(a);
    strncpy(s+2, a+6, 2); // år
    strncpy(s+5, a+0, 2); // mån
    strncpy(s+8, a+3, 2); // dag
    printf("På svenskt sätt skrivs datumet som %s\n", s);
}
```

2.10

```
#include <stdio.h>
#include <string.h>
main()
{
    char idag[11], pnr[11];
    printf("Dagens datum? ");
    gets(idag);
    printf("Personnummer? ");
    gets(pnr);
    if (strncmp(idag+5, pnr+2, 2)==0 && strncmp(idag+8, pnr+4, 2)==0)
        printf("Grattis!\n");
}
```

2.11 a)

```
typedef struct {
    int t, n;
} rat_tal;
```

b)

```
rat_tal add(rat_tal r1, rat_tal r2)
{
    rat_tal res;
    res.t = r1.t*r2.n + r2.t*r1.n;
    res.n = r1.n*r2.n;
    return res;
}
```

```
rat_tal mul(rat_tal r1, rat_tal r2)
{
    rat_tal res;
    res.t = r1.t*r2.t;
    res.n = r1.n*r2.n;
    return res;
}
```

c)

```
#include <stdio.h>
main()
{
    rat_tal a, b, c, d;
    printf("Skriv täljare och nämnare för det första talet: ");
    scanf("%d%d", &a.t, &a.n);
    printf("Skriv täljare och nämnare för det andra talet: ");
    scanf("%d%d", &b.t, &b.n);
    c = add(a, b);
    d = mul(a, b);
    printf("Summa: %d/%d\nProdukt: %d/%d\n", c.t, c.n, d.t, d.n);
}
```

2.12 a)

```
int strlen (const char *s)
{
    const char *p=s;
    while (*p++)
        ;
    return p-1-s;
}
```

b)

```
int strlen (const char *s)
{
    int i=0;
    while ( s[i] )
        i++;
    return i;
}
```

c)

|                                                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                     |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> // pekare ; int strlen (const char *s) _strlen: ; {   LEAS -2,SP ;   const char *p=s;   LDD 4,SP   STD 0,SP ;   while (*p++) _1:   LDX 0,SP   TST 1,X+   STX 0,SP   BNE _1 ;   ; ;   return p-1-s;   LDD 0,SP   SUBD 4,SP   LDX #-1   LEAX D,X   TFR X,D ; }   LEAS 2,SP   RTS </pre> | <pre> // indexering ; int strlen (const char *s) _strlen: ; {   LEAS -2,SP ;   int i=0;   CLRA   CLRB   STD 0,SP ;   while ( s[i] ) _1:   LDD 0,SP   ADDD 4,SP   TFR D,X   TST 0,X   BEQ _2 ;   i++;   LDX 0,SP   INX   STX 0,SP   BRA _1 _2: ;   return i;   LDD 0,SP ; }   LEAS 2,SP   RTS </pre> |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

2.13 a)

```

void strcpy (char *s1, const char *s2)
{
  while (*s1++ = *s2++)
    ;
}

```

b)

```

void strcpy (char *s1, const char *s2)
{
  int i = 0;
  while (s1[i] = s2[i])
    i++;
}

```

c)

|                                                                                                                                                                                                            |                                                                                                                                                                                                                                                                                                                                                                               |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>// pekare ; void strcpy (char *s1, const char *s2) _strcpy: ; { ;   while (*s1++ = *s2++) _1:   LDX 2,SP   LDY 4,SP   LDAB 1,Y+   STAB 1,X+   STX 2,SP   STY 4,SP   TSTB   BNE _1 ;   ; ; } RTS</pre> | <pre>// indexering ; void strcpy (char *s1, const char *s2) _strcpy: ; { ;   LEAS -3,SP ;   int i = 0; ;   CLRA ;   CLRB ;   STD 1,SP ;   while (s1[i] = s2[i]) _1:   LDD 1,SP   ADDD 7,SP   TFR D,X   LDAB 0,X   STAB 0,SP   LDD 1,SP   ADDD 5,SP   TFR D,X   LDAB 0,SP   STAB 0,X   TSTB   BEQ _2 ;   i++;   LDX 1,SP   INX   STX 1,SP   BRA _1 _2: ; } LEAS 3,SP RTS</pre> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

```
2.14 #include <stdio.h>
#include <stdlib.h>
main(int argc, char *argv[])
{
  FILE *f;
  char rad[500];
  int c;
  int n=0;
  int tom = 1;
  if (argc < 2) {
    printf("Filnamn saknas\n");
    exit(1);
  }
  f = fopen(argv[1], "r");
  if (f == NULL) {
    printf("Kan inte öppna %s\n", argv[1]);
    exit(2);
  }
  while ((c = fgetc(f)) != EOF)
  {
    if (c=='\n')
    {
      rad[n] = '\0';
      if (! tom)
        printf("%s\n", rad);
      n=0;
      tom = 1;
    }
    else
    {
      rad[n++] = c;
      if (c != ' ' && c != '\t')
        tom = 0;
    }
  }
}
```

2.15

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double nollstalle(double (*f)(double), double a, double b, double eps)
{
    if (f(a) > 0 && f(b) < 0)
    { // byt a och b
        double temp = a;
        a=b; b=temp;
    }
    if (!(f(a) < 0 && f(b) > 0))
    {
        printf("Nollställe saknas\n");
        exit(99);
    }
    // nu gäller f(a) < 0 < f(b)
    while (fabs(a-b) > eps)
    {
        double xm=(a+b)/2, fm=f(xm);
        if (fm < 0)
            a=xm;
        else if (fm > 0)
            b=xm;
        else
            return xm; // Vi råkade finna nollstället
    }
    return (a+b)/2;
}
```

2.16

```
typedef unsigned char *port8ptr;
#define ML4OUT_ADR 0x400
#define ML4IN_ADR 0x600

#define ML4OUT *((port8ptr) ML4OUT_ADR)
#define ML4IN *((port8ptr) ML4IN_ADR)

void blinkdelay( void )
{
    int i;
    for( i = 0; i < 0x300 ; i++ );
}

void blink( void )
{
    ML4OUT = 0xFF;
    blinkdelay();
    ML4OUT = 0;
    blinkdelay();
}

void main( void )
{
    while( 1 )
    {
        blink ();
    }
}
```

## 2.17

```
typedef unsigned char *port8ptr;
#define ML4OUT_ADR 0x400
#define ML4IN_ADR 0x600

#define ML4OUT *((port8ptr) ML4OUT_ADR)
#define ML4IN *((port8ptr) ML4IN_ADR)

void blinkdelay( void )
{
    int i;
    for( i = 0; i < 0x200 ; i++ );
}

void rljush( void )
{
    unsigned char c = 0;
    while( 1 )
    {
        if( c == 0 )
            c = 0x80;
        ML4OUT = c;
        blinkdelay();
        c = c >> 1 ;
    }
}

void main( void )
{
    rljush ();
}
```

## 2.18

```
typedef unsigned int *port16ptr;
#define ML4OUT_ADR 0x400

#define ML4OUT16 *((port16ptr) ML4OUT_ADR)

void blinkdelay( void )
{
    int i;
    for( i = 0; i < 0x200 ; i++ );
}

void rljush16( void )
{
    unsigned int c = 0;
    while( 1 )
    {
        if( c == 0 )
            c = 0x8000;
        ML4OUT16 = c;
        blinkdelay();
        c = c >> 1 ;
    }
}

void main( void )
{
    rljush16 ();
}
```

2.19

```
typedef unsigned char *port8ptr;

#define ML4OUT_ADR 0x400
#define ML4IN_ADR1 0x600
#define ML4IN_ADR2 0x601

#define ML4OUT *((port8ptr) ML4OUT_ADR)
#define ML4IN1 *((port8ptr) ML4IN_ADR1)
#define ML4IN2 *((port8ptr) ML4IN_ADR2)

void DipSwitchOr( void )
{
    unsigned char c;
    while( 1 )
    {
        c = ML4IN1 | ML4IN2;
        ML4OUT = c;
    }
}

void main( void )
{
    DipSwitchOr ();
}
```

2.20

```
typedef unsigned char *port8ptr;

#define ML4OUT_ADR 0x400
#define ML4IN_ADR 0x600

#define ML4OUT *((port8ptr) ML4OUT_ADR)
#define ML4IN *((port8ptr) ML4IN_ADR)

void ff1( void )
{
    unsigned char pattern, bitpos;
    while( 1 )
    {
        pattern = ML4IN;

        if( ! pattern )
            bitpos = 0;
        else{
            for( bitpos = 1; bitpos < 8; bitpos++ )
            {
                if( pattern & 1 )
                    break;
                pattern >>= 1;
            }
            ML4OUT = bitpos;
        }
    }
}

void main( void )
{
    ff1 ();
}
```

2.21

```
typedef unsigned char *port8ptr;
typedef unsigned short *port16ptr;

#define ML4OUT_ADR 0x400
#define ML4IN_ADR1 0x600
#define ML4IN_ADR2 0x601

#define ML4OUT16 *((port16ptr) ML4OUT_ADR)

#define ML4IN1 *((port8ptr) ML4IN_ADR1)
#define ML4IN2 *((port8ptr) ML4IN_ADR2)

void AddUnsigned8bitTo16( void )
{
    unsigned short int s;
    while( 1 )
    {
        s = ( unsigned short ) ML4IN1;
        s = s + ( unsigned short ) ML4IN2;
        ML4OUT16 = s;
    }
}

void main( void )
{
    AddUnsigned8bitTo16 ();
}
```

2.22

```
typedef char *port8ptr;
typedef short *port16ptr;

#define ML4OUT_ADR 0x400
#define ML4IN_ADR1 0x600
#define ML4IN_ADR2 0x601

#define ML4OUT16 *((port16ptr) ML4OUT_ADR)

#define ML4IN1 *((port8ptr) ML4IN_ADR1)
#define ML4IN2 *((port8ptr) ML4IN_ADR2)

void AddSigned8bitTo16( void )
{
    short s;
    while( 1 )
    {
        s = ( short ) ML4IN1;
        s = s + ( short ) ML4IN2;
        ML4OUT16 = s;
    }
}

void main()
{
    AddSigned8bitTo16 ();
}
```



2.23

```
typedef unsigned char *port8ptr;
typedef unsigned int *port16ptr;

#define ML4OUT_ADR 0x400
#define ML4IN_ADR1 0x600
#define ML4IN_ADR2 0x601

#define ML4OUT16 *((port16ptr) ML4OUT_ADR)

#define ML4IN1 *((port8ptr) ML4IN_ADR1)
#define ML4IN2 *((port8ptr) ML4IN_ADR2)

void DipHex( void )
{
  unsigned short int s;
  while( 1 )
  {
    s = ( unsigned short ) ML4IN1;
    s = s * ( unsigned short ) ML4IN2;
    ML4OUT16 = s;
  }
}
```

2.24

```
typedef unsigned char *port8ptr;

#define ML4OUT_ADR1 0x400
#define ML4OUT_ADR2 0x401
#define ML4IN_ADR1 0x600
#define ML4IN_ADR2 0x601

#define ML4OUT1 *((port8ptr) ML4OUT_ADR1)
#define ML4OUT2 *((port8ptr) ML4OUT_ADR2)

#define ML4IN1 *((port8ptr) ML4IN_ADR1)
#define ML4IN2 *((port8ptr) ML4IN_ADR2)

void DivModHex( void )
{
  unsigned char q,r,pa;
  pa = ML4IN2;
  if( pa != 0 )
  {
    q = ML4IN1/pa;
    r = ML4IN1%pa;
  }else{
    q = 0xFF;
    r = 0xFF;
  }
  ML4OUT1 = q;
  ML4OUT2 = r;
}
```



```
// Filen keyboardML5.c

#include "keyboardML5.h"
#include "ports.h"
#include "clock.h"

int keyb(void)
{
    int radnr, kolnr;
    unsigned int radbit, kolbits;
    unsigned int kolmask = 0xf; // markerar vilka kolumner som används
    // tabell för avkodning av kolumnbitar, -1 markerar fel
    int decode[16] = {-1, -1, -1, -1, -1, -1, -1, -1, 3,
                    -1, -1, -1, 2, -1, 1, 0, -1};

    // vänta tills alla tangenter är uppe
    ML5KEYB_CTRL = 0xf; // aktivera alla rader
    while ((ML5KEYB_STAT & kolmask) != kolmask) // är någon tangent nedtryckt?
        ;

    // upprepa tills någon tangent trycks ner
    while (1) {
        // löp igenom alla rader och låt 'radbit' markera insignalerna 0-3
        for (radnr=0, radbit=0x1; radnr<4 ; radnr++, radbit<<=1) {
            ML5KEYB_CTRL = radbit; // aktivera raden
            kolbits = ML5KEYB_STAT & kolmask; // avläs kolumnerna
            if (kolbits != kolmask) { // är någon tangent på raden nedtryckt?
                hold(200); // ja, vänta 200 ms
                // avläs kolumnerna igen
                if ((ML5KEYB_STAT & kolmask) == kolbits) { // fortfarande intryckt?
                    kolnr = decode[kolbits]; // ger kolumnnumret för nollan
                    return 4*(3-kolnr) + radnr;
                }
            }
        }
    }
}

```

## 2.27

```
typedef unsigned char * port8ptr;
#define DISPLAY *((port8ptr) 0x400)
#define DIPSWITCH1 *((port8ptr) 0x600)
#define DIPSWITCH2 *((port8ptr) 0x601)

void CondRunDiode( void )
{
    unsigned char value;
    value = 0x80; // initialvärde */
    while( 1 )
    {
        if( DIPSWITCH1 > DIPSWITCH2 )
        { /* ljus rinner åt vänster */
            DISPLAY = value;
            value = value << 1;
            if( value == 0 ) /* över kanten? ... */
                value = 1; /* böja om från höger */
        }else if ( DIPSWITCH1 < DIPSWITCH2 )
        { /* ljus rinner åt höger */
            DISPLAY = value;
            value = value >> 1;
            if( value == 0 ) /* över kanten? ... */
                value = 0x80; /* böja om från vänster */
        }else /* ljus står still */
            DISPLAY = value;
    }
}

```

2.27 a)  
 \_a RMB 1  
 \_b RMB 1  
 \_c RMB 1

b)  
 LDAB \_b  
 PSHB  
 LDAB \_a  
 PSHB  
 JSR \_min  
 LEAS 2,SP  
 STAB \_c

2.28 a)  
 \_a RMB 2  
 \_b RMB 2  
 \_c RMB 2

b)  
 LDD \_b  
 PSHD  
 LDD \_a  
 PSHD  
 JSR \_min  
 LEAS 4,SP  
 STD \_c

2.29 a)  
 \_a RMB 2  
 \_b RMB 2  
 \_c RMB 2

b)  
 LDD \_b  
 PSHD  
 LDD \_a  
 PSHD  
 JSR \_min  
 LEAS 4,SP  
 STD \_c

2.30 a)  
 \_cp RMB 2

b)  
 LDX #\_cp  
 PSHX  
 JSR \_identify  
 LEAS 2,SP  
 STD \_cp

2.31: a) LEAS -2,SP

b)

| Parameter/<br>variabel | adressering |
|------------------------|-------------|
| a                      | 4,SP        |
| b                      | 0,SP        |

2.32: a) LEAS -4,SP

b)

| Parameter/<br>variabel | adressering |
|------------------------|-------------|
| a                      | 8,SP        |
| b                      | 6,SP        |
| c                      | 2,SP        |
| d                      | 0,SP        |

2.33: a) LEAS -5,SP

b)

| Parameter/<br>variabel | adressering |
|------------------------|-------------|
| a                      | 12,SP       |
| b                      | 11,SP       |
| c                      | 7,SP        |
| d                      | 4,SP        |
| e                      | 0,SP        |

2.34:

```

; void f1( unsigned char c )
_f1:
; {
; *( unsigned char *) 0x600 = c ;
    LDAB 2,SP
    STAB $600
; delay();
    JSR _delay
; c = c >> 1;
    LDAB 2,SP
    LSRB
    STAB 2,SP
; *( unsigned char *) 0x600 = c ;
    STAB $600
; }
    RTS

```

2.35:

```

void printerprint( char *s )
_printerprint:
; {
; while( *s )
    LDX 2,SP
printerprint1:
    TST ,X
    BEQ printerprint2
; {
; while( !( STATUS & 1 ) )
; {}
printerprint3:
    LDAB $0701
    ANDB #$01
    BEQ printerprint3

; DATA = *s;
    LDAB 1,X+ (även 's++' nedan)
    STAB $0700

```

```

;   s++;
   BRA   printerprint1
printerprint2:
;   }
; }
   RTS

```

2.36:

```

; void shortdelay( void )
_shortdelay:
; {
; volatile unsigned char c;
   LEAS  -1,SP
; for( c = 0; c < 0x200 ; c++ );
   CLR   0,SP
_1:
   LDAB  0,SP
   CMPB  #$200
   BGE   _2
   INC   0,SP
   BRA   _1
_2:
; }
   LEAS  1,SP
   RTS

```

2.37:

```

; void shortdelay( void )
_shortdelay:
; {
; unsigned char c;
; for( c = 0; c < 0x200 ; c++ );
   CLRB
_1:
   CMPB  #$200
   BGE   _2
   INCB
   BRA   _1
_2:
; }
   RTS

```

2.38:

```

; void printchar( char c )
_printchar:
; {
; while( *((volatile unsigned char *) 0x600) )
_1:
   TST   $600
   BNE   _1
;
; *((unsigned char *) 0x400) = c;
   LDAB  2,SP
   STAB  $400
; }
   RTS

```

2.39:

```

; void printmul( void )
_printmul:
; {
; unsigned short int s;
   LEAS  -2,SP

```

```
; s = ( unsigned short ) (*((unsigned char *) 0x600) );
    LDAB    $600
    CLRA
    STD     0,SP
; s = s * ( unsigned short ) (*((unsigned char *) 0x601) );
    LDAB    $601
    TFR     D,Y
    LDD     0,SP
    EMUL
    STD     0,SP
; *((unsigned short int *) 0x400) = s;
    STD     $400
; }
    LEAS    2,SP
    RTS
```

2.40: a)

```
_getCCR:
    TFR     CCR,B
    RTS
```

b)

```
_setCCR:
    LDAB    2,SP
    TFR     B,CCR
    RTS
```

2.41:

I den första lösningen är stacken balanserad då RTI utförs, det är dock inte fallet i den andra lösningen eftersom en lokal variabel deklarerats och epilogen (som balanserar stacken i funktionen) alltid placeras sist dvs. EFTER den infogade RTI-instruktionen.

### 3 Undantagshantering

#### 3.4 a)

```
*****
* AVBROTTSRUTIN-IRQCNT
* Beskrivning:   Läs 8-bitars tal (tvåkomplement) från port ($0600).
*               Typkonvertera och addera till 32-bitars tal (IRQVAR)
*               Kvittera avbrott (skrivning till adress $0DC2)
* Anrop:        via IRQ
*****
IRQIN   EQU     $0600
IRQCLR  EQU     $0DC2

IRQCNT:  DES                ; plats för tecken-byte
         LDAB      IRQIN     ; läs 8 bitar
         SEX      B,D       ; teckenutvidga till 16 bitar i D
         STAA    0,SP       ; spara tecken-byte
         ADDD    IRQVAR+2   ; addera bit0-15
         STD     IRQVAR+2   ; uppdatera bit 0-15
         LDD     IRQVAR     ; bit 16-31
         ADCB    0,SP       ; addera bit 16-23
         ADCA    0,SP       ; addera bit 24-31
         STD     IRQVAR
         CLR     IRQCLR     ; nollställ avbrottsvippan
         INS                    ; återställ stacken
         RTI

IRQVAR   RMB     4
```

#### b)

```
*****
* SUBRUTIN-IRQINIT
* Beskrivning:   Rutinen nollställer D-vippan, lägger in adressen
*               till avbrottsrutinen på adressen $3FF2 och
*               förbereder systemet för avbrott genom att I-flaggan
*               nollställs.
* Anrop:        JSR  IRQINIT
*****
IRQINIT:  MOVW    #0,IRQVAR      ; Init Var
         MOVW    #0,IRQVAR+2
         CLR     IRQCLR         ; nollställ avbrottsvippan
         MOVW    #IRQCNT,$3FF2  ; avbrottsvektor
         CLI
         RTS
```

#### 3.5 Övervakningsprogrammet startas med ett RESET som leder till följande programavsnitt:

```
INIT:    LDS     BOS           ; Efter RESET skall stacken initieras och
         LDAB    $0801         ; avbrottsvipporna nollställas
         LDAB    $0802         ; (dummläs $0801 och $0802)
         CLI                    ; Därefter skall IRQ-ingången demaskeras och
         JMP     CONTROL       ; övervakningen startas
```

Övervakningen sker huvudsakligen m h a avbrottsrutinen IRQALARM, som anropar väsentliga subrutiner

```
IRQALARM: LDAB    $0800         ; avgör vad som begärt avbrott
         BITB    #%00000001 ; kassan?
         BEQ    L1
         BITB    #%00000010 ; både kassan och vakt?
         BEQ    L2
         LDAB    $0801         ; tillåt registrering av nya avbrottsbegäran
         LDAB    $0802
         BSR    CHAOS         ; reglera för avbrott från både kassan och vakt
         BRA    L3
L2:      LDAB    $0802         ; tillåt registrering av ny avbrottsbegäran från kassan
         BSR    ENTRANCE     ; reglera för avbrott från enbart kassan
         BRA    L3
L1:      LDAB    $0801         ; tillåt registrering av ny avbrottsbegäran från vakt
         BSR    GUARD        ; reglera för avbrott från enbart vakt
L3:      RTI
```



3.6 a) Ur ingångssignalerna till NAND-grinden kan man dra slutsatsen att

$CS' = [VMA \cdot (R/W)' \cdot A_{15}' \cdot A_{14}' \cdot A_{13}' \cdot A_{12}' \cdot A_{11}' \cdot A_{10}' \cdot A_9' \cdot A_8' \cdot A_7' \cdot A_6' \cdot A_5' \cdot A_4' \cdot A_3' \cdot A_2' \cdot A_1' \cdot A_0']'$   
vilket innebär att adressen är  $(0011\ 0000\ 0000\ 0000)_2 = \$3000$

```
*****
* Subroutine INISTR
*
* Utför initieringar för avbrottsstyrd utmatning av tecken från
* parallell inport till skrivare
*
* INPUT:  Pekare till första tecken i sträng i X
* OUTPUT: Inga
*
* Registerpåverkan: Inga
*
*****

INISTR:      PSHS      X
             STX      STRPNT      ; initierar strängpekare i minnet
             LDX      #PRIRQ      ; initierar avbrottsvektorn för IRQ
             STX      $3FF2
             CLR      $3000      ; matar ut ASCII-tecknet 00h (NUL)
                                 ; nollställer avbrottsvippan
             PULS     X
             CLI      ; nollställer I-flaggan för att tillåta IRQ
             RTS

*****
* Interruptroutine PRIRQ
*
* Kopierar tecken från teckensträng till parallellutport.
* Kopieringen avslutas när tecknet "NUL" = 00H upptäcks.
*
* INPUT:  Pekare till första tecken i sträng på adress STRPNT
* OUTPUT: Inga
*
* Registerpåverkan: Inga
*****

PRIRQ:      LDX      STRPNT
             LDAB     1,X+
             STX      STRPNT      ; uppdaterar strängpekare
             TSTB     ; sätter flaggor utifrån A
             BNE      GO_ON
             PULS     A      ; strängslut: stänger av avbrott; hämtar CC från stacken
             ORAA     #%00010000 ; nollställer I-flaggan
             PSHS     A      ; lägger tillbaka på CC's läge i stacken
             BRA      RET
GO_ON:      STAB     $3000      ; matar ut ASCII-tecken
RET:        RTI

3.7
TEMP rmb    2      Avbrottsräknare (1000 IRQ = 1s)

IRQINIT:    MOVW     #$2359,CLOCK ; Init klockan tt:mm:ss
            MOVB     #$59,CLOCK+2
            MOVW     #1000, TEMP ; Avbrottsräknare
            CLR      IRQRES      ; nollställ avbrottsvippan
            MOVW     #IRQ, $3ff2 ; avbrottsvektor
            CLI
            RTS

IRQ:        CLR      IRQRES      ; nollställ avbrottsvippan
            LDX      TEMP      ; 1000 avbrott?
            LEAX     -1,X
            STX      TEMP
            BNE      IExit      ; nej
            MOVW     #1000, TEMP ; Avbrottsräknare
* Minska sekunder
            LDAA     CLOCK+2
```

```

SUBA    #1
DAA
STAA    CLOCK+2    ; Hel minut?
BPL     IExit      ; nej
* Minska minuter
MOVB    #$59,CLOCK+2 ; 59 nya sekunder
LDAA    CLOCK+1
ADDA    #-1
DAA
STAA    CLOCK+1    ; Hel timme?
BPL     IExit      ; nej
* Minska timmar
MOVB    #$59,CLOCK+1 ; 59 nya minuter
LDAA    CLOCK
ADDA    #-1
DAA
STAA    CLOCK      ; 24 timmar?
BPL     IExit      ; nej

* Stanna klockan på något sätt!
* Använd någon global variabel och kolla om klockan är noll eller
* se till att förhindra framtida avbrott
LDAA    0,sp        ; Ettställ I-flaggan
ORAA    #$10
STAA    0,sp
MOVW    #0,CLOCK   ; Nolla klockan
CLR     CLOCK+2
IExit:  RTI

```

## 3.8 a)

```

IrqRut:  LDAA    IrqStat      ; Läs statusflaggorna
          LSRA
          BCC    Ejb0
          JSR    DSR0        ; Serva enhet 0
          BRA    IrqExit

Ejb0:    LSRA
          BCC    Ejb1
          JSR    DSR1        ; Serva enhet 1
          BRA    IrqExit

Ejb1:    LSRA
          BCC    Ejb2
          JSR    DSR2        ; Serva enhet 2
          BRA    IrqExit

Ejb2:    JSR    DSR3        ; Serva enhet 3

IrqExit: CLR    IrqVippa
          RTI

```

b) Vi tappar avbrott om avbrott inträffar mellan instruktionerna `ldaa IrqStat` och `clr IrqVippa`.

c) Risken minskas om instruktionen `clr IrqVippa` placeras direkt efter `ldaa IrqStat`.

d) Välja hårdvara där vi har möjlighet att bestämma vilken av de fyra avbrottskällorna vi skall kvittera.  
Exempelvis 4 avbrottsvippor med separata RESET-möjligheter

## 3.9

```

// I filen ports.h
typedef void (*vec) (void);
typedef vec *vecptr;
typedef unsigned int port;
typedef port *portptr;
#define set(r, mask)    (r) = (r) | mask;
#define clear(r, mask) (r) = (r) & ~(mask);

// Klockregistret
#define CLOCKREG_ADR    0x1230
#define CLOCKREG        *((portptr) CLOCKREG_ADR)

#define CLOCK_VEC_ADR    0xFF70 // Adress till avbrottsvektor
#define CLOCK_VEC        *((vecptr) CLOCK_VEC_ADR)

```

```
// Sensorerregistret
#define SENSORREG_ADR      0x1234
#define SENSORREG         *((portptr) SENSORREG_ADR)

#define SENSOR_VEC_ADR    0xFF80 // Adress till avbrottsvektor
#define SENSOR_VEC        *((vecptr) SENSOR_VEC_ADR)

#define enable_bit        0x01
#define intr_bit          0x40
#define done_bit          0x80

// I filen tidtagare.c
#include "ports.h"

void display(long);
void sensortrap(void);
void clocktrap(void);

#define TIME_INTERVAL 2

static long int tick = 0;
static int started = 0;
static int stopped = 0;

void init_clock(void) {
    CLOCK_VEC = clocktrap;
    set(CLOCKREG, intr_bit);
}

void clockinter(void) {
    clear(CLOCKREG, done_bit);
    tick++;
}

void init_sensor(void) {
    SENSOR_VEC = sensortrap;
    port shadow = 0;
    set(shadow, enable_bit);
    set(shadow, intr_bit);
    SENSORREG = shadow;
}

void sensorinter(void) {
    clear(SENSORREG, done_bit);
    if (!started) {
        set(CLOCKREG, enable_bit);
        started = 1;
    }
    else {
        clear(CLOCKREG, enable_bit);
        stopped = 1;
    }
}

int main() {
    long int next;
    init_clock();
    init_sensor();
    display(0);
    while (!started)
        ;
    while(!stopped) {
        display(tick * TIME_INTERVAL / 10);
        /* vänta 0.01 sek */
        next = tick + 10 / TIME_INTERVAL;
        while(tick < next)
            ;
    }
    display(tick * TIME_INTERVAL / 10);
}
```

## 4 Programmering av periferikretsar

4.1 a)

```
typedef struct sPortP{
    volatile unsigned char ddr;
    volatile unsigned char data;
}PORTP, *PPORTP;
#define PORTP_BASE 0x700
#define portP ((PORTP *) (PORTP_BASE))
```

```
void portPinit( void )
{
    portP->ddr = ~0xE0;
}
```

b)

```
unsigned char inPortP( void )
{
    return (( portP->data & 0xE0 )>> 5) ;
}
```

c)

```
void outPortP( unsigned char c )
{
    portP->data = c & 0x1F ;
}
```

4.1 a)

```
* Adressdefinitioner för register
```

```
REFDV EQU $35
SYNR EQU $34
CRGFLG EQU $37
CLKSEL EQU $39
* Bitdefinitioner
PLLSEL EQU $80
LOCK EQU 8
```

```
* Registervärden 10MHz oscillator, 25 MHz busfrekvens
```

```
SYNRVal: EQU 5
REFDVVal: EQU 4
```

```
* Generisk kod för programmerad arbetstakt...
```

```
PLLINIT: MOVB #REFDVVal,REFDV
        MOVB #SYNRVal,SYNR
PLLINIT_1:
        BRCLR CRGFLG,#LOCK, PLLINIT_1 ; vänta tills PLL låst...
        BSET CLKSEL,#PLLSEL ; växla systemklocka till PLL.
        RTS
```

b)

```
typedef struct sCRG{
    volatile unsigned char SYNR;
    volatile unsigned char REFVDV;
    volatile unsigned char CTFLG;
    volatile unsigned char CRGFLG;
    volatile unsigned char CRGINT;
    volatile unsigned char CLKSEL;
    volatile unsigned char PLLCTL;
    volatile unsigned char RTICTL;
```

---

---

```

volatile unsigned char COPCTL;
volatile unsigned char FORBYP;
volatile unsigned char CTCTL;
volatile unsigned char ARMCOP;
}CRG, *PCRG ;

```

c)

```

#define CRG_BASE 0x34 /* Basadress för CRG-modulen */
#define SYNRRVal 5
#define REFDRVal 4

#define PLLSEL 0x80 /* Bitdefinitioner */
#define LOCK 8

void InitPLL(void)
{
    ( ( ( PCRG ) ( CRG_BASE ))->refdr ) = REFDRVal;
    ( ( ( PCRG ) ( CRG_BASE ))->synr ) = SYNRRVal;
    /* vänta tills PLL låst... */
    while( ( ( (volatile PCRG) (CRG_BASE))->crgflg ) & LOCK )== 0);
    /* växla systemklocka till PLL */
    ( ((PCRG) (CRG_BASE))->clkssel ) |= PLLSEL;
}

```

d)

```

typedef struct sCRG2{
union{
    volatile unsigned char reg;
    volatile unsigned char synbits:6;
}synr;
union{
    volatile unsigned char reg;
    volatile unsigned char refbits:4;
}refdr;
volatile unsigned char ctflg;
union{
    volatile unsigned char reg;
    struct{
        volatile unsigned char SCM:1;
        volatile unsigned char SCMIF:1;
        volatile unsigned char SCMIE:1;
        volatile unsigned char LOCK:1;
        volatile unsigned char LOCKIF:1;
        volatile unsigned char LVRF:1;
        volatile unsigned char PORF:1;
        volatile unsigned char RTIF:1;
    }bit;
}crgflg;
volatile unsigned char crgint;
union{
    volatile unsigned char reg;
    struct{
        volatile unsigned char COPWAI:1;
        volatile unsigned char RTIWAI:1;
        volatile unsigned char CWAI:1;
        volatile unsigned char PLLWAI:1;
        volatile unsigned char ROAWAI:1;
        volatile unsigned char SYSWAI:1;
    }bit;
}crgint;
}CRG2;

```

---

```

volatile unsigned char PSTP:1;
volatile unsigned char PLLSEL:1;
}bit;
}clkssel;

volatile unsigned char pllctl;
volatile unsigned char rtictl;
volatile unsigned char copctl;
volatile unsigned char forbyp;
volatile unsigned char ctctl;
volatile unsigned char armcop;
}CRG2, *PCRG2 ;

```

e)

```

void InitPLL2(void)
{
    ( ( ( PCRG2 ) ( CRG_BASE ))->refdv.refbits ) = REFDVVal;
    ( ( ( PCRG2 ) ( CRG_BASE ))->synr.synbits ) = SYNRVVal;
    /* vänta tills PLL låst... */
    while( ! (((volatile PCRG2) (CRG_BASE))->crgflg.bit.LOCK ) )
        ;
    /* växla systemklocka till PLL */
    ((PCRG2 ) (CRG_BASE))->clkssel.bit.PLLSEL = 1;
}

```

4.2 a)

\* Adressdefinitioner

```

CRGFLG      EQU      $37
CRGINT      EQU      $38
RTICTL      EQU      $3B
RTIE        EQU      $80
RTIF        EQU      $80
TIMBASE     EQU      $62      ; ur tabell

```

```

RTINIT:     MOVB     #TIMBASE,RTICTL      ; För MC12/10MHz
            MOVB     #RTIE,CRGINT        ; Aktivera avbrott från CRG-modul
            MOVW     #RTIRQ, $3FF0       ; Avbrottsvektor
            RTS

```

b) 9,83 ms.

c)

```

RTIRQ:      BSET     CRGFLG,# RTIF        ; Kvittera avbrott
            JSR      _AtRTIrq
            RTI

```

4.3

a)  $25000000/(16 \times 57600) = (27)_{10} = (1B)_{16}$ 

b)

```

SCI         EQU      $C8
SCIBD       EQU      $C8
SCICR1      EQU      $CA
SCICR2      EQU      $CB
SCISR1      EQU      $CC
SCISR2      EQU      $CD
SCIDRH      EQU      $CE

```

```

SCIDRL    EQU    $CF

TE        EQU    8        ; "Transmit enable" bit
RE        EQU    4        ; "Receive enable" bit
BAUDRATE  EQU    27       ; enligt a)

SERIAL_INIT:
    MOVW   #BAUDRATE,SCIBD    ; Initiera baudrate
    MOVB   #(TE|RE),SCICR2    ; Aktivera sändare mottagare
    RTS

```

c)

```

typedef struct sSCI{
    volatile unsigned short  scibd;
    volatile unsigned char   scicr1;
    volatile unsigned char   scicr2;
    volatile unsigned char   scisr1;
    volatile unsigned char   scisr2;
    volatile unsigned char   scidrh;
    volatile unsigned char   scidrl;
}SCI, *PSCI;

```

d)

```

#define SCI_BASE    0xC8
#define TE          8        // "Transmit enable" bit
#define RE          4        // "Receive enable" bit
#define BAUDRATE    27       // enligt a)

```

```

void serial_init( void )
{
    ( (( PSCI )( SCI_BASE ))->scicr2 ) = TE|RE;
    ( (( PSCI )( SCI_BASE ))->scibd  ) = BAUDRATE;
}

```

e)

```

RDRF      EQU    $40        ; "Receive register fullt" bit

```

```

SERIAL_IN:
    BRCLR  SCISR1,#RDRF,SERIAL_IN
    LDAB  SCIBD
    RTS

```

f)

```

#define RDRF    0x40        // "Receive register fullt" bit

```

```

char serial_in( void )
{
    while( (( (PSCI) (SCI_BASE))->scisr1 ) & RDRF )== 0);
    return ( ((PSCI) (SCI_BASE))->scidrl );
}

```

g)

```

TDRE      EQU    $80        ; "Transmit register tomt" bit

```

```

SERIAL_OUT:
    BRCLR  SCISR1,#TDRE,SERIAL_OUT

```

---

```
STAB SCI BD  
RTS
```

h)

```
#define TDRE 0x80 // "Transmit register tomt" bit  
void serial_out( char c )  
{  
    while( ( ( (PSCI) (SCI_BASE))->scisr1 ) & TDRE ) == 0);  
    ( ( (PSCI) (SCI_BASE))->scidr1 ) = c;  
}
```

---