CHALMERS

Data- och informationsteknik, datorteknik

	U
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

Maskinorienterad programmering

Arbetshäfte för laboration nr 1-3

Utvecklingsverktyget Eterm

Programmering i assemblerspråk

Borrmaskinsstyrning med mikrodatorn MC12

Testning och felsökning

Användning av avbrott

- *Händelsestyrda avbrott*
- *Tidsstyrda avbrott, pseudoparallellism*

Arbetet i häftet delas upp i tre delar:

Del 1 omfattar uppgifterna 1-3. (Lab 1)
Del 2 omfattar uppgifterna 4-8. (Lab 2)
Del 3 omfattar resterande uppgifter. (Lab 3)

Observera att uppgifterna i detta arbetshäfte även omfattar inmatning av text och simulering av program. Uppgifterna kan till största delen utföras utanför laborationstid.

Version 13-01-14

Ifylls med bläck!

Laborant:

Personnummer	Namn (textat)	Grupp nr	Datum

Godkännande - laboration:

Laborationshandledares underskrift	Datum

Innehåll

Inledning

I kursen Digital- och datorteknik har beskrivits hur en enkel ”pedagogisk” dator fungerar, programmeras och utför uppgifter, sett i huvudsak från ett datortekniskt perspektiv. I detta arbetshäfte skall ”kärnan”, CPU12, i en modern ”microcontroller”, MC68HCS12 studeras med hjälp av ett utvecklingsverktyg **Eterm 6.** Verktyget används för att simulera ”micro-controllern” som i sin tur styr och övervakar en simulerad enkel borrmaskin, d v s ett mekaniskt system.

Målsättning

Att ge dig

- en inblick i vad ett mikroprocessorbaserat system för en enkel tillämpning består av,
- träning i att skriva ett huvudprogram med subrutinanrop,
- träning i att utveckla, skriva och dokumentera enkla programavsnitt, i form av subrutiner och avbrottsrutiner,

samt via praktisk verksamhet låta dig ladda, testa, felsöka och rätta dessa program

Förberedelser

Innan den praktiska verksamheten i laboratoriet påbörjas skall du för varje laborationspass

- läsa igenom aktuell del av arbetshäftet.
- tillsammans med din laborationspartner utföra en så stor del av uppgifterna i förväg att ni hinner med alla aktuella uppgifter under laborationspasset.
- inhämta kunskaper genom studier av motsvarande avsnitt i kurslitteraturen inklusive instruktionslistan för CPU12.
- dokumentera förberedelsearbetet på ett läsbart och förståeligt sätt.

1 Utvecklingsverktyget Eterm 6

1.1 Textredigering och assemblering med Eterm 6

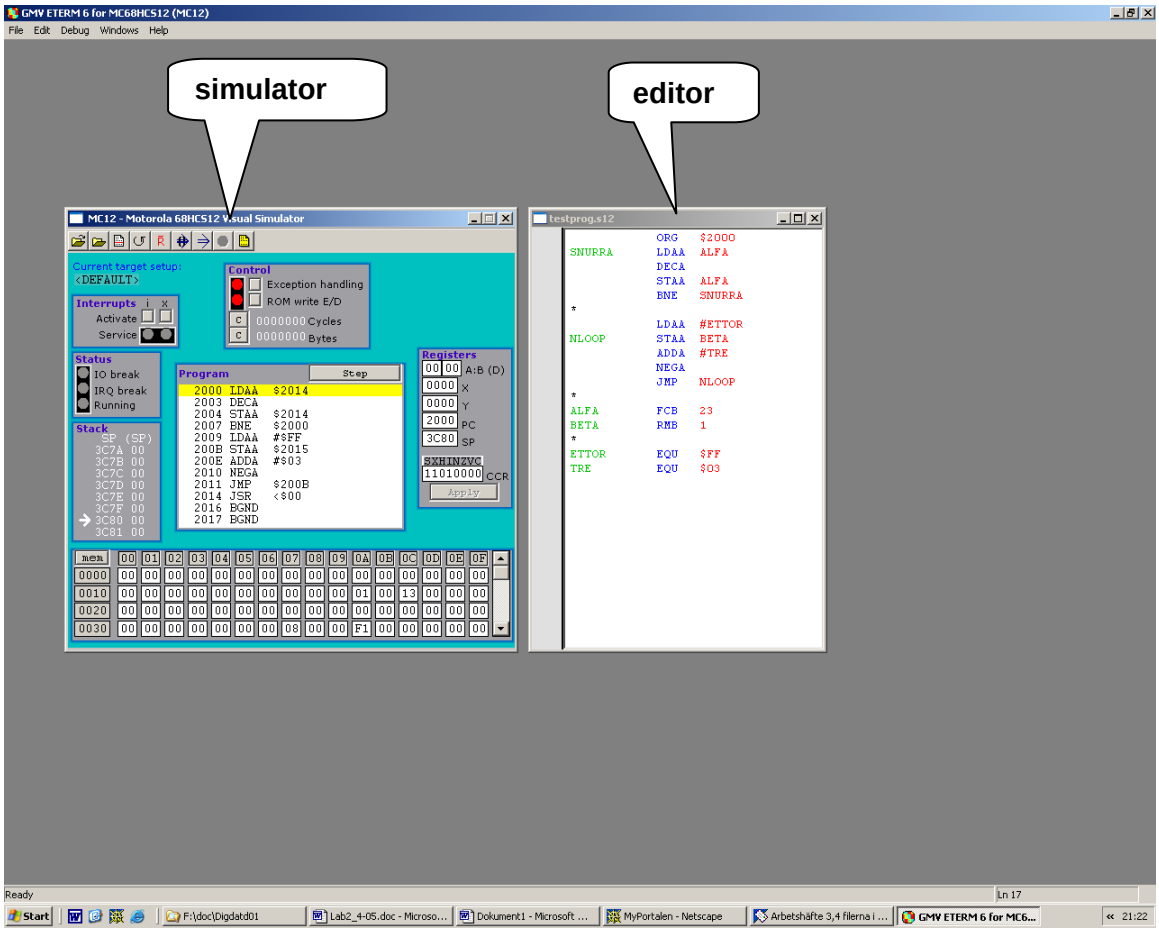
ETERM är ett utvecklingssystem för programutveckling i assemblerspråk. Det har utvecklats för undervisningsändamål och fungerar under Windows XP och Vista. Det finns flera varianter av *ETERM* beroende på vilken måldator (laborationsdator) som används. I detta häfte beskrivs *ETERM 6* för laborationsdatorn MC12, som är uppbyggd kring mikroprocessorn MC68HCS12.

ETERM omfattar funktioner för:

1. **Textredigering**, källtexten skrivs/redigeras med hjälp av en *Editor*, filnamnet ska sluta med *.s12*(source CPU12).

2. **Assemblering**, källtexten översätts till en laddfil som innehåller maskinkoden, med tillägget *.s19* av den inbyggda *assemblatorn*.

3. **Test**, laddfilen överförs till den inbyggda *simulatorn*. Vid laborationstillfällen kan man överföra laddfilen till laborationsdatorn MC12 via ETERM's *terminal* med hjälp av respektive programdels laddningsprogram.



Uppgift 1:

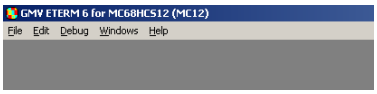
Här behandlas användningen av programdelarna *Editor* och *Assemblator* i *ETERM*.

- Bibliotek för assemblerspråksfiler

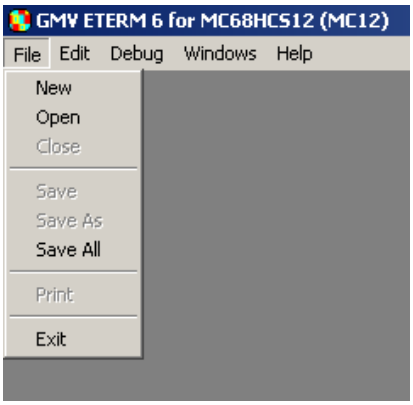
  Skapa först ett bibliotek t ex med namnet *HC12lab* för dina assemblerspråksfiler för MC12-datorn.

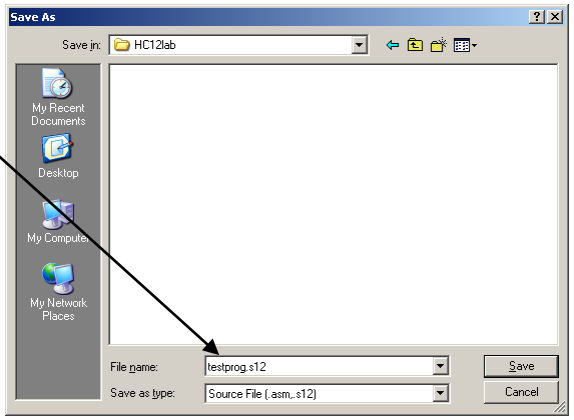
- Starta ETERM

  Starta sedan *ETERM* via ”Start-Menyn” eller genom att dubbelklicka på någon "genväg".



  Välj **File | New,** leta dig fram till biblioteket du skapade ovan och skriv *testprog.s12* i rutan **File name.** Klicka sedan på tangenten **Save** nere till höger i fönstret **Save as**.

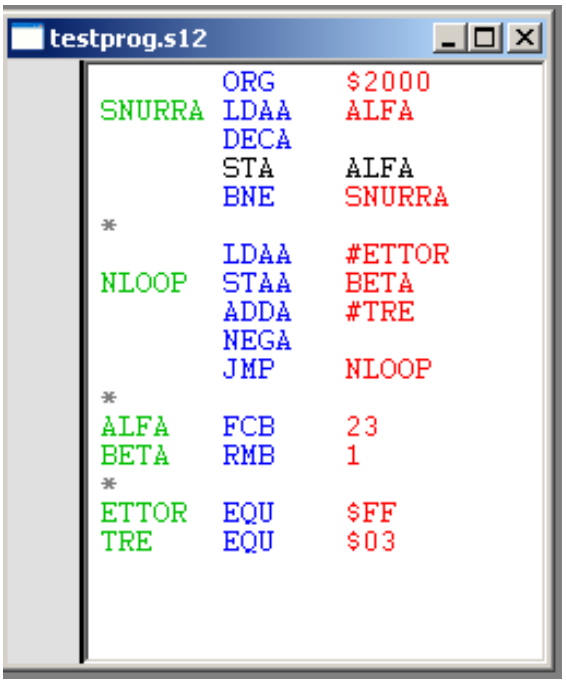




- Skapa ett assemblerspråksprogram

  Skriv in följande program (figur nedan). Använd helst (en eller flera) tabulator för att skilja kolumnerna åt.

Du kan öppna flera samtidiga fönster för källtextredigering. Om filnamnet slutar på .s12 kommer källtexten att behandlas som ett assemblerspråksprogram för *CPU12*. Ett giltigt symbolfält färgas då grönt, en giltig instruktion (eller ett direktiv) färgas blått, ett giltigt operandfält färgas rött. Observera att detta hjälpmedel (”färgad syntax”) inte nödvändigtvis innebär att ditt assemblerspråksprogram är korrekt, det är snarare till för att göra dig uppmärksam på enklare stavfel, syntaxfel etc.



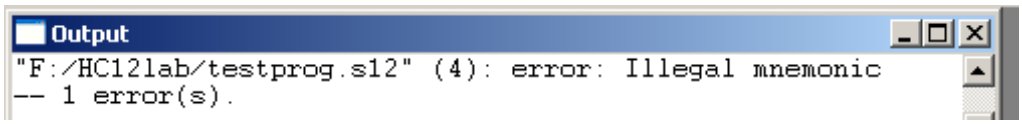
I programmet har vi exempel på såväl assemblatordirektiv som assemblerinstruktioner. Vi ser också hur vi kan skapa kommentarsrader genom att skriva '*' i radens första position.

Observera att vi har stavat en STAA-instruktion felaktigt (STA). Det medför att vi kommer att få se ett exempel på en felutskrift vid assembleringen.

Nu är det dags att assemblera filen, dvs översätta källtexten till maskinkod.

- Aktivera Editorfönstret genom att klicka i fönstret! (Vid detta tillfälle är fönstret normalt aktiverat, vilket du ser genom att titelraden är blå)
- Välj **Debug|Assemble** från menyn!

  Filen assembleras nu och assemblatorn skriver direkt ut en felutskrift för den felstavade "STAA"-instruktionen. Om programmet (källtextfilen) inte innehåller fler fel får du en utskrift liknande den följande. (Jämför denna med *din* felutskrift).



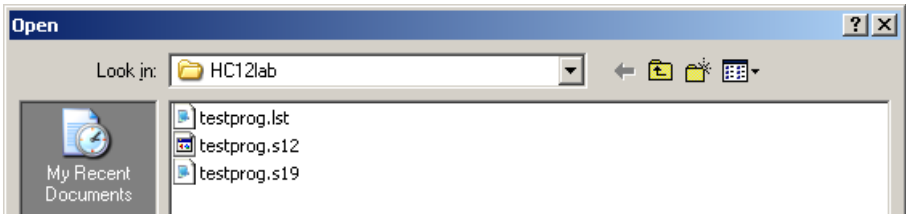
Pröva nu med att dubbelklicka på felutskriften. Observera vad som händer i Editorfönstret **testprog.s12** ...

Assemblatorn känner inte till någon instruktion som heter "STA" och kan därför inte översätta denna till maskinkod. Vi måste då rätta felet.

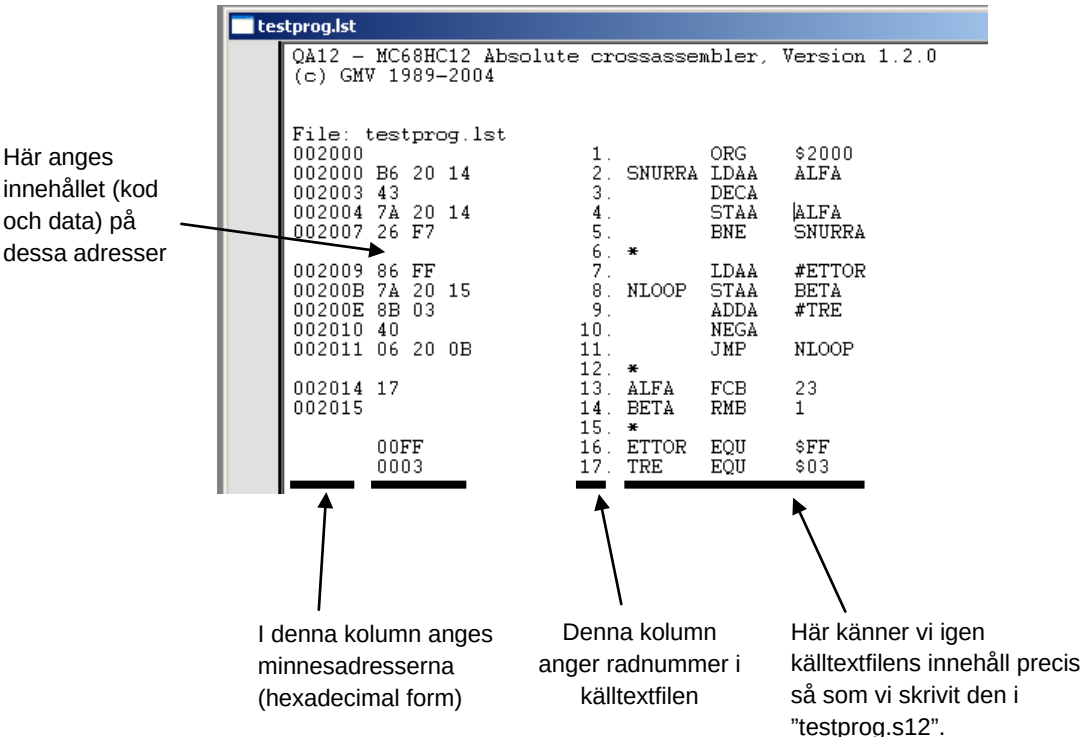
- Ändra den felaktiga stavningen till STAA. Assemblera därefter på nytt

  Nu bör programmet vara korrekt. Om det fortfarande är fel så rätta felen genom att redigera och assemblera på nytt.
- Stäng källfilen (**File|Close**).
- Välj sedan **File|Open** och ange **Any File (*.*)** i rutan **Files of type:** i fönstret **Open** för att ta reda på vilka filer som bildades vid assembleringen.

  Man ser i fönstret nedan att det har bildats två nya filer med tilläggen .s19 och .lst.

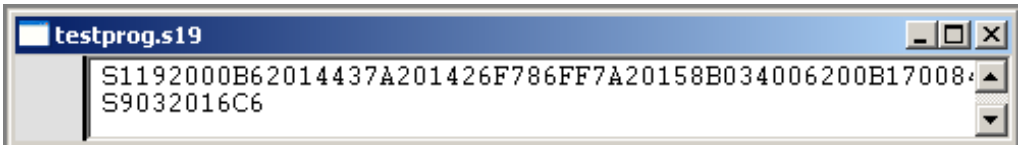


Öppna filen testprog.lst. Följande fönster öppnas:



Filen ovan kallas listfilen och är mycket användbar vid felsökning av program eftersom den innehåller all information som behövs.

- Öppna också filen testprog.s19. Följande fönster öppnas:



Denna fil kallas laddfilen och innehåller endast information om programmets maskinkod och motsvarande minnesadresser samt en del information för det laddprogram som skall placera maskinkoden i minnet.

Nu har du gått igenom de steg som krävs för att:

- Skapa ett assemblerprogram för laborationsdatorn
- Översätta assemblerprogrammet till maskinkod

▫▫▫

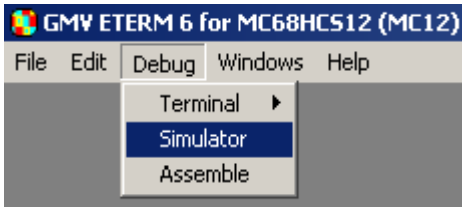
1.2 Simulering av program med Eterm 6

För att kunna utveckla och testa assemblerprogram är det viktigt att ha tillgång till, och behärska, en lättanvänd och anpassad programutvecklingsmiljö. I sådana här sammanhang är användning av *simulatorer* vanlig. Programdelen *Simulator* i *ETERM* kan användas för att simulera instruktionsutförandet i laborationsdatorn. Du kan t ex utföra ett assemblerprogram instruktionsvis och studera effekterna av de enskilda instruktionerna.

Uppgift 2:

Start av simulatorn

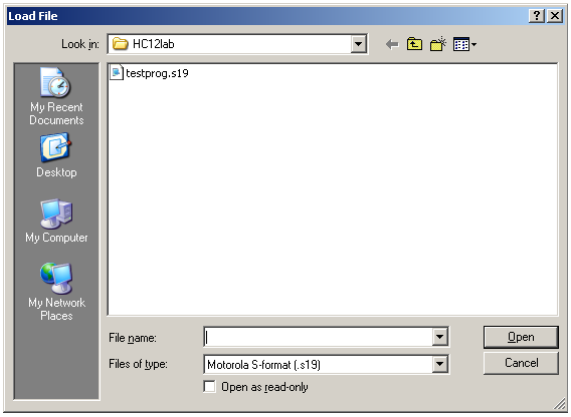
Vi ska nu fortsätta arbeta med programmet i filen ”testprog.s12” från föregående avsnitt och använda *MC12-simulatorn* i *ETERM*.



- Välj **Debug | Simulator** i fönstret ETERM 6/MC12.

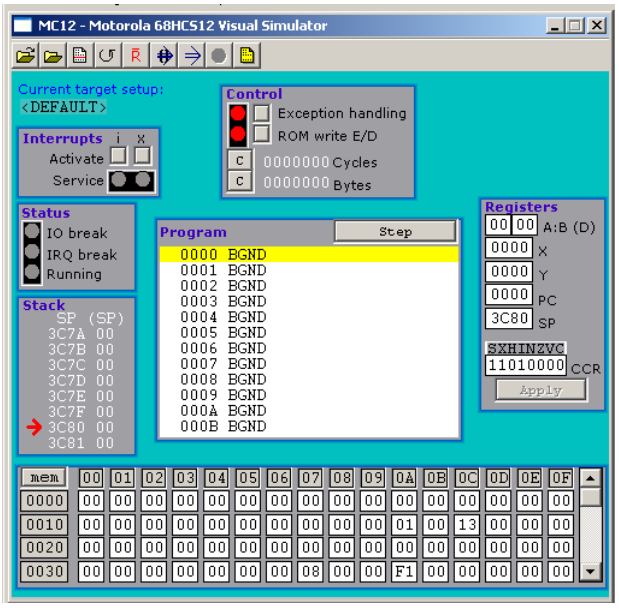
  Anm. Om ett Editor-fönster är aktivt när man startar simulatorn så assembleras motsvarande program automatiskt och dess laddfil hämtas till simulatorn.

  Dialog-rutan till höger öppnas nu:



- Välj **testprog.s19** och klicka på **Open**

Simulatorfönstret nedan till höger öppnas: Simulatorfönstret indelas i ett antal olika fält:



- ”Program” *programfönster,* här visas en dissassemblering av det laddade programmet. Den instruktion som står i tur att utföras markeras med gul färg. Man kan utföra denna instruktion genom att klicka på knappen ”Step”.
- "Registers" *registerinnehåll*, anger innehållet i CPU12's register. Man kan ändra dessa innehåll manuellt genom att skriva in ett nytt värde och sedan klicka på ”**Apply Changes**”. För ackumulatorer och indexregister tolkas det inskrivna värdet på hexadecimal form. För flaggregistret CCR tolkas värdet på binär form.

- ”mem” *minnesfönster* visar minnesinnehåll i en 64 bytes minnesarea som man kan ändra med hjälp av bläddringslisten längst ut till höger. Man kan här enkelt ändra minnesinnehållet genom att markera ett minnesord åt gången och sedan ändra det.
- "Interrupts" *avbrott* här kan man ge avbrottssignaler till den simulerade processorn. Genom att klicka en gång på **I-**, eller **X-tangenten** startar man simulering av motsvarande avbrottshantering hos CPU12.
- ”Status” *statusfönster*, här visas den simulerade processorns status ur olika aspekter, t ex om den arbetar normalt eller om den har nått en brytpunkt. Vidare finns här en bild av stackinnehållet samt två räknare som visar antalet klockcykler och antalet bytes processorn läser i minnet.
- ”Stack” *stackfönstret,* visar stackens innehåll.

Förutom styrmöjligheterna i simulatorfönstret finns det en meny som bl a används för att styra programkörning. Man når den genom att högerklicka i simulatorfönstret utanför programfönstret.

- Högerklicka i simulatorfönstret, dock ej i programfönstret! (Menyn till höger visas då.)

De fyra översta raderna i menyn är brytpunktskommandon och behandlas senare.

Med de följande två kommandona, **”Load New”** och**”Reload”** kan man ladda ett nytt program eller ladda om det gamla till simulatorns minne.

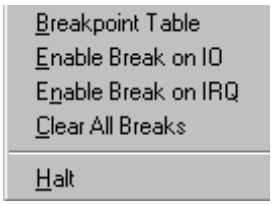
De följande fyra kommandona styr den simulerade processorn på olika sätt:

”Reset Simulator” försätter processorn i ett starttillstånd, hämtar startadressen från laddfilen och lägger in den i PC. Det är detta kommando man normalt använder innan man startar simulering av ett program med ett **”Run”**-kommando.

”Reset CPU” utför nästan samma sak som RESET av den verkliga processorn, inklusive att den hämtar startaddressen från resetvektorn. Detta kommando förutsätter att startadressen finns på adresserna $FFFE_{16}$ och $FFFF_{16}$ i minnet. Efter **”Reset CPU”** startas programmet med ett **”Run”**-kommando.

”Run” och **”Run Fast”** startar simulering av programmet. Skillnaden mellan kommandona är att vid **”Run”** uppdateras simulatorfönstret mellan varje instruktion, vilket tar lång tid.

När man kör ett program med **"Run"** eller **"Run Fast"** förändras styrmenyn man får vid högerklick i simulatorfönstret. Den nya menyn visas i marginalen. Man ser att brytpunktskommandona finns kvar och att ett nytt kommando **"Halt"** har tillkommit.



"Breakpoints" *brytpunkter* används då man låter simulatorn utföra **Run** men samtidigt vill avbryta programkörningen vid någon speciell händelse. En sådan kan vara att en förutbestämd adress används av processorn. En annan att processorn läser på en inport eller skriver till en utport. En tredje att processorn mottar en avbrottsbegäran.

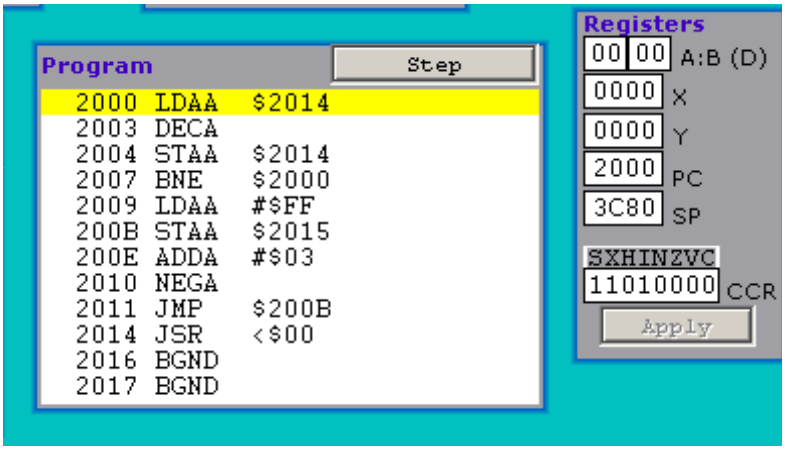
Instruktionsvis exekvering

Som första steg då man vill köra ett program måste man försäkra sig om att *CPU12's* programräknare (PC) innehåller adressen till programmets första instruktion. Som du säkert minns används direktivet ORG $2000 för att ange startadressen för programmet i laborationsdatorns minne.

Kommandot **"Reset Simulator"** medför bl a att startadressen placeras i PC.

- Högerklicka i simulatorfönstret utanför programfönstret. Välj kommandot **"Reset Simulator"** i styrmenyn. Observera speciellt vad som händer med program- och registerfönstren.

Programfältet uppdateras och vi känner igen programmet, observera också att programräknaren (Program Counter) i registerfönstret ändras.



- Klicka på **"Step"** för att utföra den första instruktionen LDAA $2014

  Observera, i registerfältet, hur innehållet i ackumulator A ändras.

Fortsätt nu med att utföra ett antal varv i programslingan instruktionsvis.

Fri exekvering

Då man testat ett program genom att stega sig igenom instruktion för instruktion kan man också välja att utföra det med **"Run"-**kommandot. Instruktionerna utförs då kontinuerligt en efter en och efter varje instruktion uppdateras alla simulatorns fönster. Detta kan vara ett utmärkt sätt att följa programflödet utan att behöva "klicka" sig igenom programmet.

- Högerklicka i simulatorfönstret, välj **"Run"** från menyn. Studera speciellt program- och register- fälten.

  För att avbryta **"Run"** högerklickar du i simulatorfönstret och väljer **"Halt"** i menyn.

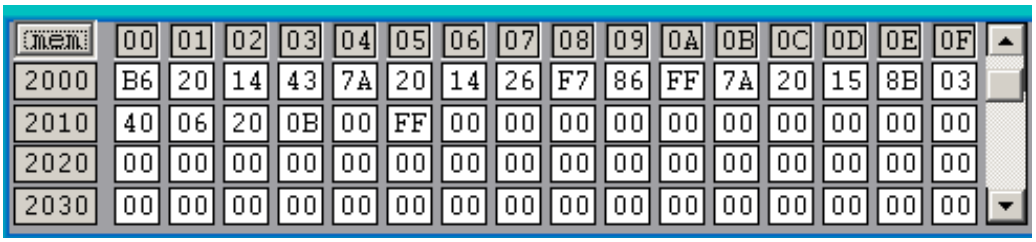
Slutligen kan du också välja **"Run Fast"** som är en ännu snabbare variant. Här simuleras instruktionerna så fort det är möjligt, inga fönster uppdateras och det går inte heller att följa programflödet. Det kan däremot vara användbart då du övertygat dig om att ditt program fungerar som det skall och vill se hur det beter sig mot någon kringenhet.

- Högerklicka och välj **"Halt"** för att avbryta **"Run".** Högerklicka igen och välj **"Run Fast"**.

  Även **"Run Fast"** avbryts genom att du högerklickar och väljer **"Halt".**

Simulatorns minne

Du kan övervaka, och även ändra i, simulatorns minne i *minnesfältet*. Använd bläddringslisten till höger för att ställa in ett lämpligt minnesintervall.
Fältet visar 64 bytes minnesinnehåll i rader om 16 bytes vardera.



Sammanfattning

I denna uppgift har du bekantat dig med *ETERM's MC12*-simulator. Det finns fortfarande ett antal funktioner i simulatorn som du ännu inte prövat på som exempelvis *Breakpoints* och *Interrupts*. *Breakpoints* kommer att användas i nästa avsnitt och *Interrupts* i slutet av häftet.

Simulatorn kommer att användas i det fortsatta arbetet med arbetshäftet. Den är också ett utmärkt hjälpmedel för att kontrollera exempelvis lösningar på övningsuppgifter mm.

¤¤¤

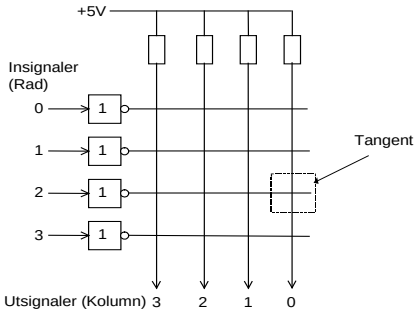
2 Inledning till borrmaskinsstyrning

I detta arbetshäfte skall simulatorn i Eterm 6 för laborationsdatorn MC12 användas för att styra och övervaka en simulerad borrmaskin så att enstaka hål eller kompletta hålmönster kan borras i ett arbetsstycke.

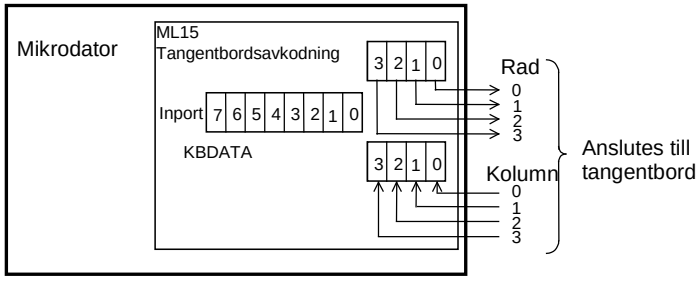
En operatör skall kunna ge olika kommandon till borrmaskinen på ett tangentbord som är ett rektangulärt s k matristangentbord med 4 insignaler och 4 utsignaler i en koppling enligt figur 1.

I varje korsning mellan en vågrät och en lodrät ledning finns en tangent. När en tangent trycks ned så kortsluts ledningarna i denna korsning. När ingen tangent är nedtryckt har alla utsignalerna 0-3 hög nivå via motstånd från +5V.

Man kan upptäcka om någon tangent i en rad av matrisen är nedtryckt genom att aktivera raden (lägga en ”etta” på ingången på radens NOT-grind i figur 1) och sedan undersöka utsignalerna 0-3. När en tangent i en aktiverad rad är nedtryckt kommer utsignalen för motsvarande kolumn (3-0) att ha låg nivå ("0"). Genom att undersöka raderna i följd kan man således via radnummer och kolumnnummer identifiera den eller de tangenter som är nedtryckta.



Figur 1



Figur 2

För att underlätta tangentavläsningen används en speciell avkodningskrets, placerad på ett kretskort (ML15), som kan kopplas direkt till datorkortet MC12. Operatörens tangentbord ansluts till mikrodatorn via ML15-kortet, såsom visas till höger i figur 2. När en tangent är nedtryckt signaleras detta genom att bit 7 på inporten KBDATA får värdet 0 och bit 3-0 visar tangentnumret enligt figur 3. Av praktiska skäl vill man dock använda tangentnumreringen i figur 4 istället för den i figur 3.

0	4	8	C
1	5	9	D
2	6	A	E
3	7	B	F

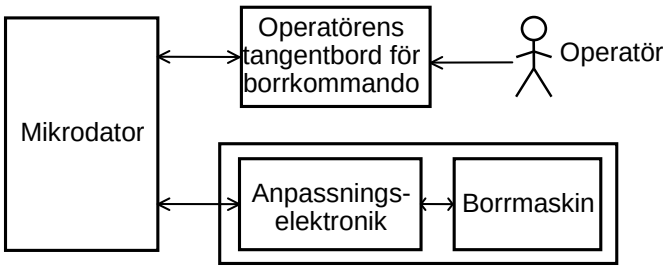
Figur 3

0	1	2	3
4	5	6	7
8	9	A	B
C	D	E	F

Figur 4

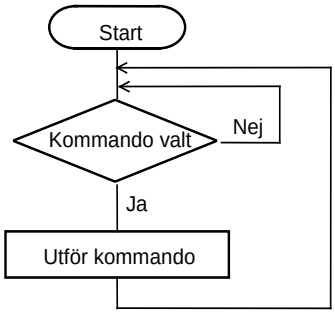
Operatören skall kunna ge olika kommandon till borrmaskinen på tangentbordet. Kommandona skall vara, starta borrmotor (0), stäng av borrmotor (1), sänk borr (2), höj borr (3), vrid arbetsstycke (4), borra ett hål (5) och borra alla hål i arbetsstycket automatiskt.

Borrmaskinssystemet skall ha det blockmässiga utseende enligt figur 5.



Figur 5

Systemet skall styras av operatörskommandon och arbeta enligt flödesplanen i figur 6.



Figur 6

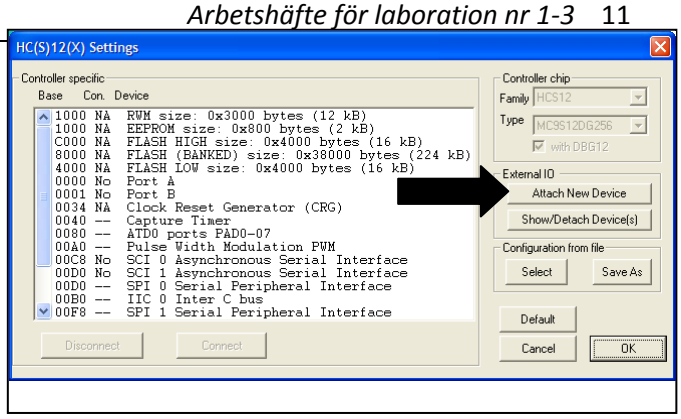
Uppgift 3a:

1. Installera vid behov programmet **Eterm 6 för MC12**. Skapa också ett bibliotek för lab1_3 på hårddisken eller på nätet. ¤¤¤

2. På kurshemsidan kommer att finnas en laddfil (***borr08.s19***) som skall innehålla ett komplett maskinprogram för styrning av borrmaskinen. Kopiera denna fil och placera den i biblioteket lab1_3. ¤¤¤

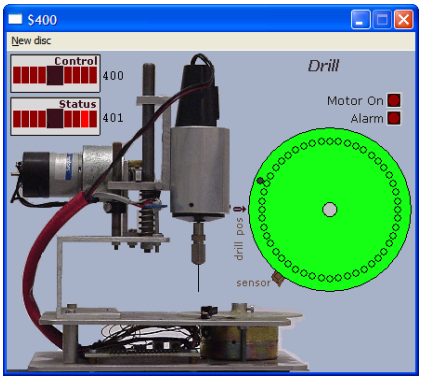
3. Starta simulatorn och ladda maskinkoden som finns i filen **borr08.s19.** Högerklicka i simulatorfönstret! Fönstret till höger visas då.
   Välj **”Simulator Setup”!**

Fönstret till höger visas då:

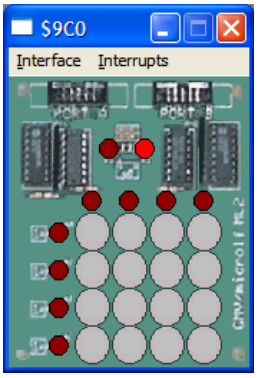


Välj **"Attach New Device"** och anslut följande I/O-enheter:

"Drill" på basadressen 400_{16},



"ML2 Keyboard" på basadressen $9C0_{16}$!



För varje enhet man ansluter till datorn med **"Attach New Device"** läggs det in ett nytt fönster med en interaktiv bild på den aktuella enheten.

Genom att klicka på **"Interface"** uppe till vänster i "tangentbordsfönstret" kan man välja hur anslutningen till datorn skall se ut. Kontrollera att avkodningskretsen **ML15** är vald. Spara sedan denna konfiguration av in/utenheter i en fil med namnet ***lab1_3io*** genom att först klicka på tangenten **"Save As"**.

Högerklicka och ge **"Reset Simulator"**. Provkör sedan programmet genom att köra det i läget "**Run Fast**". Prova de olika tangenternas funktion. Tangentfunktionerna definieras i tabellen nedan. Det är ett program med liknande funktion du själv skall skriva under arbetet med arbetshäftet. ¤¤¤

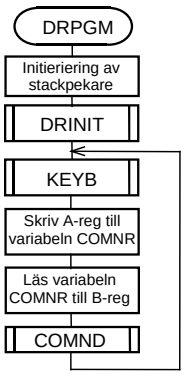
START	STOP	DOWN	UP
STEP	DRILL	AUTO	

OBS!
Man trycker ned en tangent på det simulerade tangentbordet genom att klicka på tangenten och släpper sedan upp den genom att klicka på den igen.

Kommandot DOWN skall endast fungera om borrmotorn är startad med kommandot START.

När kommandot AUTO utförs är det lämpligt att använda skiva (disc) nr 4, eftersom det önskade hålmönstret är markerat på denna. Man byter skiva genom att klicka på "New disc" uppe till vänster i borrmaskinsfönstret.

Programmet som styr borrmaskinen, DRPGM, skall i princip utformas enligt figur 7.



Figur 7

Subrutinen DRINIT har specificerats på följande sätt:

DRINIT

Anrop:	JSR DRINIT
Indata:	Inga
Utdata:	Inga
Registerpåverkan:	CC-registret får påverkas
Anropade subr:	Ingen
Beskrivning:	Rutinen initierar borrmaskinutrustningen genom att mata ut styrordet 00_{16} på utgången DRCTRL. Därmed blir alla funktioner hos borrmaskinen passiva. Vidare uppdateras adress DRCOPY med en kopia av styrordet, dvs 00_{16}.

Subrutinen KEYB läser av operatörens tangentbord upprepade gånger tills en tangent trycks ned. När en tangent trycks ned placeras motsvarande tangentnummer i A-registret före återhopp. Vid återhopp från KEYB vet man alltså att operatören har tryckt ned en tangent och att tangentnumret finns i A-registret. En beskrivning av tangentbordet och dess anslutning finns på sidorna 9 och 10.

I subrutinen COMND utförs ett av maximalt åtta olika borrkommandon, som i sin tur är subrutiner. Vilket kommando som utförs bestäms av värdet i B-registret vid anropet. Flödesschemat för subrutinen COMND visas i figur 8.

Enligt figur 7 skall programmet (huvudprogrammet)

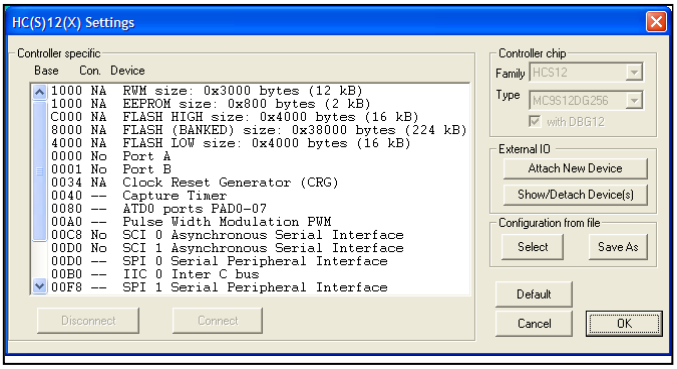
- Initiera stacken, d v s sätta stackpekaren till "bottom-of-stack"-adressen BOS
- Anropa subrutinen DRINIT, som placerar borrmaskinen i viloläge, dvs ger DRCTRL-bitarna 4-0 värdet 0.
- Anropa subrutinen KEYB, som hämtar ett 8-bitars värde från operatörens tangentbord till register A.
- Placera detta värde i variabeln COMNR i minnet.
- Läsa värdet i variabeln COMNR till register B.
- Anropa subrutinen COMND, som utför önskat kommando, och därefter hoppa tillbaka till anropet av KEYB så att operatören kan ge ett nytt kommando på sitt tangentbord.

Uppgift 3b: (Omfattar punkt 1 – 17)

1. En första variant av huvudprogrammet **DRPGM** är redan skriven och finns i **Appendix 1**. Studera det noga och mata in programtexten i en fil med namnet **main.s12** i biblioteket **lab1_3** m h a editorn i **Eterm 6 för MC12**. ¤¤¤

2. Assemblera huvudprogramfilen **main.s12** och rätta eventuella fel som upptäcks i samband med detta. De två varningar man här får vid assembleringen behöver man inte bry sig om! Vid assembleringen bildas en listfil "main.lst" och en laddfil "main.s19". Studera listfilen och laddfilen genom att öppna dem med editorn. ¤¤¤

3. Ladda huvudprogrammets maskinkod som finns i filen **main.s19** till simulatorn. Högerklicka i simulatorfönstret!
   Välj **"Simulator Setup"!**

   Fönstret till höger visas då:

   Klicka på tangenten **"Select"** i fältet **"Configuration from file"** och välj filen ***lab1_3io*** i det nya fönstret. Avsluta med **"Open"**.

   Placera ut borrmaskinen och tangentbordet på lämpligt sätt på skärmen och avsluta med tangenten **"OK"** längst ner till höger i fönstret **"Settings"**.

   Högerklicka i simulatorfönstret och ge **"Reset Simulator"**.

   Tänk igenom vad programmet gör och testa det sedan genom att stega igenom det. Rätta eventuella fel med editorn, assemblera och ladda om det rättade programmet till simulatorn tills det fungerar som avsett. ¤¤¤

På kurshemsidan finns en länk till en fil **keyb.s12** med de färdigskrivna subrutinerna KEYB och DELAY. Programlistningar finns i **Appendix 2**.

Specifikationerna för de två subrutinerna visas nedan.

KEYB

Anrop:	JSR KEYB
Indata:	Inga
Utdata:	Tangentnummer (0-F_{16}) för nedtryckt tangent i register A.
Registerpåverkan:	Register A, CC
Anropade subr:	Ingen
Beskrivning:	Rutinen väntar tills samtliga tangenter är uppe. Därefter registreras första nedtryckta tangent. Återhoppp från rutinen görs alltså inte förrän en tangentnedtryckning har registrerats. Rutinen läser tangentnumret från avkodningskretsen enligt figur 3 och översätter det enligt figur 4.

DELAY

Anrop ex:	LDAA #6 Genererar en fördröjning på 6∗50 ms = 300 ms JSR DELAY
Indata:	Antalet fördröjningsintervall N om 50 ms i A-registret.
Utdata:	Inga.
Registerpåverkan:	Register CC
Anropade subr:	Ingen
Beskrivning:	Skapar en fördröjning om N ∗ 50 ms.

4. Kopiera filen keyb.s12 från hemsidan till ditt laborationsbibliotek. Inkludera filen med subrutinerna KEYB och DELAY i huvudprogrammet med assemblatordirektivet USE keyb.s12 i huvudprogramfilen main.s12. Lägg in USE-direktivet på samma ställe som i Appendix 3.

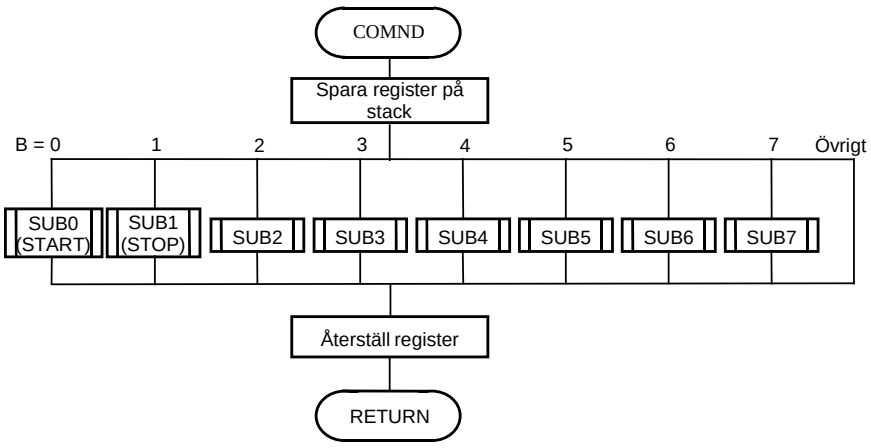
Ändra definitionen av KEYB till KEYB EQU KEYB1. Se Appendix 3. Därmed anropar man den riktiga KEYB-subrutinen från huvudprogrammet.

¤¤¤

5. Assemblera huvudprogramfilen **main.s12** och rätta eventuella fel som upptäcks i samband med detta. Starta sedan simulatorn och ladda programmet. Högerklicka i simulatorfönstret och ge **"Reset Simulator"**.

Testa sedan programmet genom att först stega igenom det och sedan köra det med "Run". Rätta eventuella fel med editorn, assemblera och ladda om programmet till simulatorn och testa tills det fungerar som avsett. ¤¤¤

När ett nytt operatörskommando har mottagits av subrutinen KEYB placerar huvudprogrammet kommandonumret i variabeln COMNR i minnet och laddar det sedan till register B, innan kommandosubrutinen COMND anropas. Flödesschemat för subrutinen COMND visas i figur 8.



Figur 8

Enligt figur 8 skall subrutinen COMND först

- spara registervärden på stacken

och sedan beroende av värdet i register B,

- för B = 0 -7, anropa en av subrutinerna SUB0 - SUB7, eller för B > 7, göra ingenting

och till sist

- återställa registervärden och göra återhopp

De olika subrutinerna som anropas av subrutinen COMND representerar var sitt kommando (var sin operation) som operatören kan ge till borrmaskinen.

Subrutinnamnen SUB0 och SUB1 skall senare bytas ut mot START och STOP, såsom visas i figur 8. Subrutinen **COMND** är redan skriven och finns i **Appendix 4**.

Adresserna till subrutinerna SUB0 - SUB7 skall ligga i en tabell i minnet med begynnelseadressen JUMPTAB. Eftersom adresserna är 16 bitar breda krävs det 2 st minnesord för att rymma (lagra) varje adress. Adresserna lagras på standardformat med den mest signifikanta byten på minnesadressen med lägst värde.

Eftersom huvudprogrammet innehåller subrutinanrop, men inga subrutiner ännu finns, ersätts de riktiga subrutinerna SUB0 - SUB7 med tomma subrutiner, som bara består av instruktionen RTS på resp. begynnelseadress.

Dessa förenklade subrutiner placeras i slutet av filen **comnd.s12**, på samma sätt som visas i slutet av Appendix 4.

6. Studera Appendix 4 noga och mata in programtexten i en fil med namnet **comnd.s12**. Placera filen i ditt laborationsbibliotek.

Inkludera subrutinen COMND i huvudprogrammet med assemblatordirektivet USE comnd.s12 i filen **main.s12**. Lägg in det nya USE-direktivet på raden efter det USE-direktiv som redan finns.

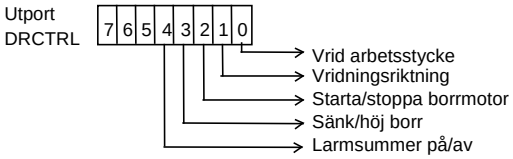
Ändra också definitionen av COMND till COMND EQU COMND1 i slutet av filen **main.s12**. Därmed anropas den riktiga COMND-subrutinen från huvudprogrammet. ¤¤¤

7. Assemblera huvudprogramfilen **main.s12** och rätta eventuella fel som upptäcks i samband med detta.

Starta sedan simulatorn och ladda huvudprogrammets maskinkod som finns i filen **main.s19** till simulatorn. Högerklicka och ge **"Reset Simulator"**. Testa programmet genom att stega igenom det. Rätta eventuella fel med editorn, assemblera och ladda det rättade programmet till simulatorn.

Alla giltiga värden (0-7) och något ogiltigt (>7) på variabeln COMNR skall testas genom att dess värde ges med operatörstangentbordet. ¤¤¤

All styrning av borrmaskinen skall ske via en utport enligt figur 9. Utporten har den symboliska adressen DRCTRL. De olika subrutinerna SUB0 -SUB7 i figur 8 skall styra borrmaskinen på olika sätt från utporten DRCTRL. Av figur 9 framgår att bitarna 0-5 på utporten styr var sin funktion hos borrmaskinen.



Figur 9

Eftersom utporten genom sin konstruktion ej är läsbar för processorn behövs en läsbar kopia av det DRCTRL-ord som för tillfället styr borrmaskinen, d v s det aktuella styrordet. Skälet till detta är att styrsubrutinerna skall kunna ändra enstaka bitar i styrordet och lämna de andra bitarna oförändrade. Detta går till så att en styrsubrutin först läser kopian av det gamla styrordet, ändrar en bit, och sedan skriver det nya styrordet till utportsadressen och till kopian. Om flera bitar i styrordet behöver ändras så utförs ändringarna en bit åt gången.

Kopian av styrordet skall finnas på adressen **DRCOPY** där den kan läsas av de subrutiner som behöver den. **DRCOPY** skall alltid uppdateras när **DRCTRL** ändras.

Subrutinen SUB0, som anropas i huvudprogrammet, skall nu bytas ut mot subrutinen START. Den startar borrmotorn enligt följande specifikation:

START

Beskrivning:	Subrutinen startar borrmotorn och väntar därefter i 200 ms före återhopp för att borret skall uppnå rätt hastighet. Starten av borrmotorn sker genom att subrutinen OUTONE först matar ut värdet "1" på bit 2 av utporten DRCTRL och sedan uppdaterar kopian av styrordet, DRCOPY. Endast bit 2 på utporten och DRCOPY påverkas. Därefter fördröjs återhoppet 200 ms med subrutinen DELAY.
Anrop ex:	JSR START
Indata:	Inga
Utdata:	Inga
Registerpåverkan:	CC-registret får påverkas
Anropade subrutiner:	OUTONE, DELAY.

Subrutinen **START** är redan skriven och finns som programlistning i **Appendix 5**. Av specifikationen och listningen framgår att subrutinen **START** anropar subrutinerna **OUTONE** och **DELAY**.

Subrutinen **OUTONE** sköter utmatningen av styrsignaler (styrordet) till borrmaskinen då en av styrsignalerna skall ettställas. En kopia av styrordet lagras på minnesadressen **DRCOPY**. Subrutinen **OUTONE** har följande specification:

OUTONE

Beskrivning:	Läser kopian av borrmaskinens styrord på adress DRCOPY. Ettställer en av bitarna och skriver det nya styrordet till utporten DRCTRL och tillbaka till kopian DRCOPY. Vilken bit som ettställs bestäms av innehållet (0-7) i B-registret vid anrop av subrutinen. Ifall innehållet > 7 utförs ingenting.
Anrop ex:	LDAB #5 Bit nummer 5 i styrordet skall ettställas JSR OUTONE
Indata:	B-registret skall innehålla ett tal 0-7, som är numret på den bit i styrsignalordet som skall ettställas. Bitnumrering framgår av figuren nedan.
Utdata:	Inga
Registerpåverkan:	CC-registret får påverkas
Anropade subrutiner:	Inga

7	6	5	4	3	2	1	0

8. Skriv in subrutinen **START** i en egen fil med namnet **start.s12**.
Programlistning finns i **Appendix 5**.

Ändra subrutinnamnet SUB0 till START i hopptabellen i filen **comnd.s12**.

¤¤¤

9. Skriv subrutinen **OUTONE**. (Adresser ges i **Appendix 6**.)
Ledning: Man kan använda en tabell med 8 olika masker för de 8 olika bitnumren 0 - 7. Tabellen placeras lämpligen direkt efter sista instruktionen i subrutinen **OUTONE**.

Placera även subrutinen **OUTONE** i filen **start.s12**. ¤¤¤

10. Assemblera filen **main.s12** tillsammans med filen **start.s12** genom att använda assemblatordirektivet **USE start.s12** efter de andra USE-direktiven i filen **main.s12**. Rätta eventuella fel som upptäcks i samband med detta. ¤¤¤

11. När man skall testa och felsöka en ny subrutin, t.ex. **START**, finns följande två alternativ:

- att börja exekvera subrutinen direkt på adressen **START**
- att börja exekvera huvudprogrammet **DRPGM** med indata sådana att den subrutin som skall testas (**START**) anropas

Vilket alternativ bör användas och varför?
Ledtråd: Vad händer, förutom själva hoppet, när subrutinanropet i huvudprogrammet utförs?

Svar:

_____

¤¤¤

12. Ladda filen **main.s19** till simulatorn.

Testa subrutinerna START och OUTONE genom att köra huvudprogrammet på lämpligt sätt. Rätta, assemblera om, ladda och testkör programmet tills alla fel som upptäcks är borta!

Testa även att OUTONE fungerar för indatavärden **x = 1-7** och för **x >7**.

¤¤¤

Subrutinen SUB1, som anropas i huvudprogrammet, skall nu bytas ut mot subrutinen STOP. Den stänger av borrmotorn enligt följande specifikation:

STOP

Beskrivning:	Subrutinen stoppar borrmotorn. Borrmotorn stoppas genom att subrutinen OUTZERO först matar ut värdet "0" på bit 2 av utporten DRCTRL och sedan uppdaterar kopian av styrordet, DRCOPY. Endast bit 2 på utporten och DRCOPY påverkas.
Anrop ex:	JSR STOP
Indata:	Inga
Utdata:	Inga
Registerpåverkan:	CC-registret får påverkas
Anropade subrutiner:	OUTZERO

Subrutinen STOP skall anropa en subrutin, OUTZERO, som fungerar analogt med den tidigare beskrivna OUTONE.

Subrutinen OUTZERO sköter utmatningen av styrsignaler (styrordet) till borrmaskinen då en av styrsignalerna skall nollställas. En kopia av styrordet lagras på minnesadressen DRCOPY.

OUTZERO specificeras på följande sätt:

OUTZERO

Beskrivning:	Läser kopian av borrmaskinens styrord på adress DRCOPY. Nollställer en av bitarna och skriver det nya styrordet till utporten DRCTRL och tillbaka till kopian DRCOPY. Vilken bit som nollställs bestäms av innehållet (0-7) i B-registret vid anrop av subrutinen Ifall innehållet > 7 utförs ingenting.
Anrop ex:	LDAB #5 Bit nummer 5 i styrordet skall nollställas JSR OUTZERO
Indata:	B-registret skall innehålla ett tal 0-7, som är numret på den bit i styrsignalordet som skall nollställas. Bitnumrering framgår av figuren nedan.
Utdata:	Inga
Registerpåverkan:	CC-registret får påverkas
Anropade subrutiner:	Inga

7	6	5	4	3	2	1	0

13. Skriv subrutinen **STOP**. Placera den i filen **stop.s12**.
Ändra subrutinnamnet SUB1 till STOP i hopptabellen i filen **comnd.s12**. ¤¤¤

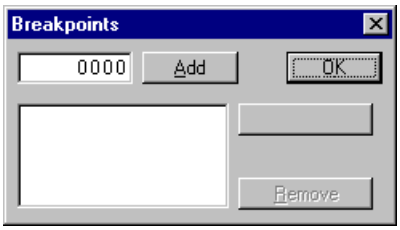
14. Skriv subrutinen **OUTZERO**. Placera den i filen **stop.s12**.
Ledning: Man kan använda en tabell med 8 olika masker för de 8 olika bitnumren 0 - 7. Tabellen placeras lämpligen direkt efter sista instruktionen i subrutinen OUTZERO. ¤¤¤

15. Assemblera filen **stop.s12** tillsammans med de övriga filerna genom att inkludera den direkt efter de andra inkluderade filerna i filen **main.s12**. Rätta eventuella fel som upptäcks i samband med detta. ¤¤¤

16. Ladda filen **main.s19** till simulatorn. Felsök sedan subrutinerna STOP och OUTZERO genom att köra huvudprogrammet på lämpligt sätt. Rätta, assemblera om, ladda och testkör programmet tills inga fler fel upptäcks! Testa även att OUTZERO fungerar för indatavärden **x = 1-7** och för **x >7**.
¤¤¤

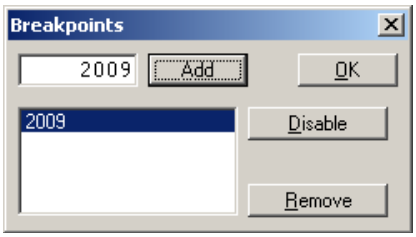
När man testar ett program är det värdefullt att kunna köra det kontinuerligt (**”Run”**) fram till en viss punkt (adress) där körningen avbryts, så att man kan studera innehållet i processorns interna register och sedan fortsätta med kontinuerlig eller stegvis instruktionskörning. Simulatorn har denna möjlighet. Man kan placera en sk brytpunkt på ett lämpligt ställe i programmet och sedan starta kontinuerlig körning (**”Run”**). När simulatorn når fram till den instruktion som finns på brytpunktsadressen avbryts körningen och man kan studera processorns och minnets innehåll.

17. Vi skall prova med att lägga in en brytpunkt på det ställe som föreslås i Appendix 1. Högerklicka därför i simulatorfönstret och välj kommandot ”**Breakpoint table**”.



Följande fönster öppnas:

Skriv in adressen till raden efter läget COLOOP (2009_{16}) och klicka på Add.



Fönstret uppdateras enligt nedan:

Klicka sedan på OK!
Detta medför att raden med adress 2009_{16} rödfärgas i programfönstret.

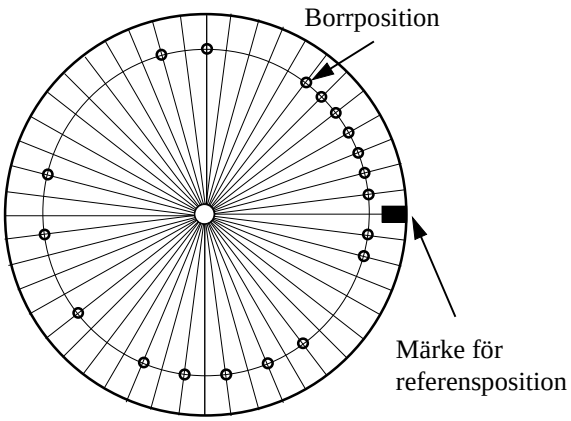
Starta sedan i läget ”**Run**”.

I fortsättningen stannar simuleringen varje gång adressen 2009_{16} passeras och man kan studera utportens värde och värdena i processorns register. När simulatorn har stannat vid en brytpunkt kan man fortsätta stega programmet eller köra det med ”Run”.

Tänk på att brytpunktsadressen kan behöva ändras om programmet ändras! Använd listfilen vid eventuell felsökning. ¤¤¤

3 Borrmaskinsutrustningen

Arbetet som beskrivs i resten av detta häfte handlar om att använda laborationsdatorn MC12 för att styra en borrmaskin. Det är en mycket enkel tillämpning som valts för att ej göra exemplet alltför komplext. Styrningen skall skötas av ett huvudprogram som introducerats och använts i föregående avsnitt.



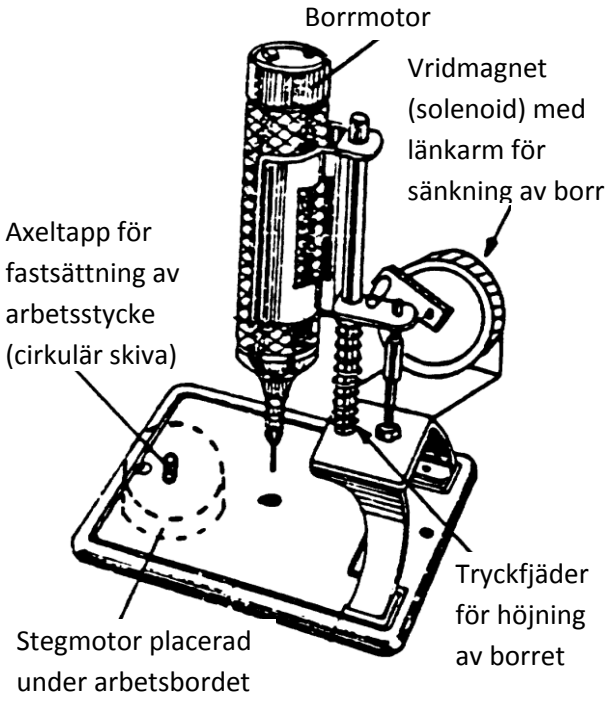
Figur 10

Det färdiga systemet skall användas för att borra hål i ett arbetsstycke, en skiva med given geometri. Hålen skall borras längs en cirkel med radien 30 mm och med n x 7,5 graders delning, n = 1,2, ..., 48, se figur 10. Hålmönstret skall vara valbart.

En borrmaskinsutrustning har satts samman, såsom visas i figur 11.

I den är en borrmaskin monterad på ett stativ. Arbetsstycket sätts fast på en axeltapp som vrids av en stegmotor med minsta vridningsvinkeln 7,5°. Avståndet mellan axeltapp och borraxel är ca. 30 mm.

Borrmaskinen är en enkel maskin avsedd att borra hål i kretskort. Den har endast en hastighet och drivs av en likströmsmatad permanentmagnetiserad motor (vanlig i elektriska leksaker).



Figur 11

Borrmaskinen är höj och sänkbar. Den kan sänkas med hjälp av en elektromagnet och höjas med en tryckfjäder.

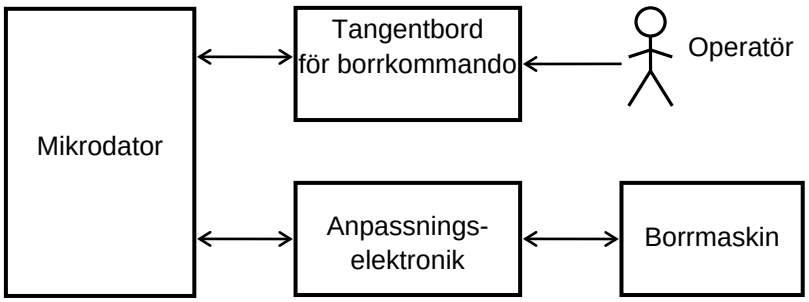
Stegmotorn är monterad under arbetsbordet. Dess axel vrids i steg om 7,5 grader genom att strömmen i motorns lindningar styrs på ett lämpligt sätt.

4 Uppgift

I grova drag är uppgiften att använda laborationsdatorn MC12 för att styra och övervaka borrmaskinen så att enstaka hål eller kompletta hålmönster kan borras.

En operatör skall kunna ställa in önskat borrmönster och utföra borrningen "manuellt" i ett antal deloperationer eller "automatiskt" i en enda komplett operation.

Systemet skall därför ha det blockmässiga utseende som framgår av figur 12.

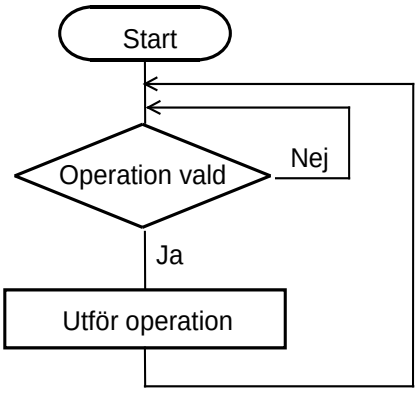


Figur 12

Systemet skall till stor del vara kommandostyrt och arbeta enligt flödesplanen i figur 13.

Operationer för följande kommandon skall finnas:

- starta borrmotorn
- stoppa borrmotorn
- sänk borret
- höj borret
- vrid (stega) arbetsstycket medurs ett steg
- borra ett hål
- vrid (stega) arbetsstycket till en referensposition och borra sedan hål längs cirkeln enligt ett bestämt mönster.



Figur 13

Det skall finnas en givare som anger när borret finns i sitt översta läge och en givare som visar när ett hål är färdigborrat, dvs att borret har nått sitt bottenläge. Dessutom skall det finnas en givare som visar att arbetsstycket har vridits till referenspositionen. Det skall också finnas en larmanordning som kan användas för att uppmärksamma operatören på vissa händelser.

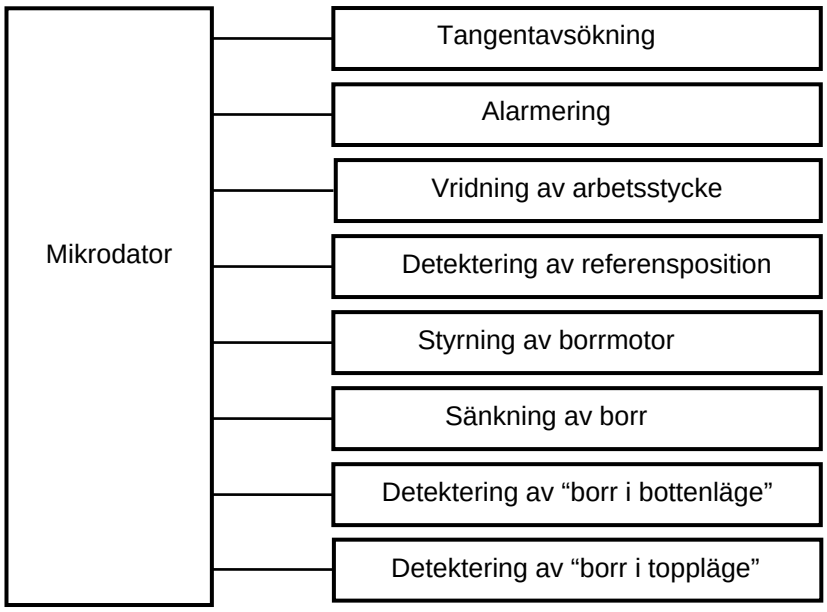
Den funktionsmässiga kommunikationen mellan mikrodatorn och dess omgivning kan sammanfattas med blockschemat i figur 14 i nästa avsnitt.

5 Systemets konstruktion

Systemet består av två delar, maskinvara och programvara. Utgångsläget för arbetet i detta avsnitt är att en prototyp av maskinvaran finns tillgänglig och att skalet till ett huvudprogram introducerats i föregående avsnitt.

5.1 Maskinvaran

Borrmaskinstativet har kompletterats med en givare som visar om arbetsstycket har vridits till sin referensposition och två givare som visar om borret är höjt till sitt toppläge eller om det sänkt till sitt bottenläge, d v s att ett hål är färdigborrat.



Figur 14

Givarna för referenspositionen och borrets lägen är läsgafflar för infrarött ljus. De består av en lysdiod som belyser en fototransistor. När ett ogenomskinligt föremål passerar ändras utsignalen.

Borrmaskinen ansluts till mikrodatorn via ett kretskort med styrelektronik, som är integrerat med borrmaskinen. Dess uppgift är att anpassa signal- och effektnivåer. På kortet finns drivenheter för borrmotorn, sänkmekanismen (en elektromagnet) och stegmotorn. Dessutom finns logik för styrning av stegmotorns olika lindningar, givarelektronik samt en larmsummer.

Kortet har fem ingångar för binära styrsignaler från MC12-datorns utportar. Kortet har också tre utgångar för givarsignaler, kopplade till en inport på MC12.

På nästa sida följer en beskrivning av styrelektronikkortets in- och utsignaler:

ingång - 'borrmotor':

nivå	funktion
0	borrmotorn går ej
1	borrmotorn går

På grund av mekanisk tröghet hos borrmaskinen får man räkna med en reaktionstid av 0,2 s vid start. Stopptiden är längre, men saknar betydelse i praktiken.

ingång - 'sänkmekanism':

nivå	funktion
0	höjer borret
1	sänker borret

ingång - 'stegpuls': En positiv flank ($0 \rightarrow 1$) vrider arbetsstycket ett steg i den riktning som anges på ingång - 'fram/back'. På grund av mekanisk tröghet får man räkna med en reaktionstid av 0,2 s per steg.

ingång - 'fram/back':

nivå	funktion
0	ger moturs vridning
1	ger medurs vridning

Skall aktiveras innan positiv flank ges på ingång - 'stegpuls'

ingång - 'larm':

nivå	funktion
0	"summern" inaktiv
1	"summern" aktiv

utgång - 'borr nere':

nivå	funktion
0	borr ej nere
1	borr nere

utgång - 'borr i topp':

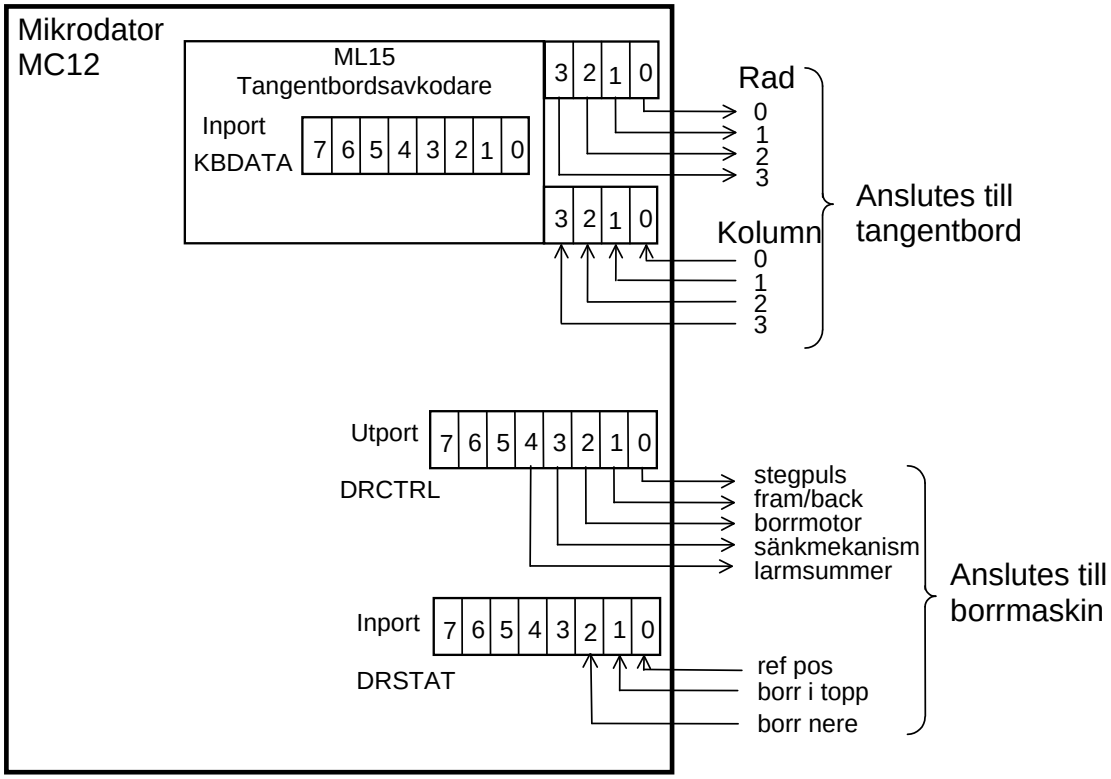
nivå	funktion
0	borr ej i topp
1	borr i topp

utgång - 'ref pos':

nivå	funktion
0	arbetsstycket ej i referensposition
1	arbetsstycket i referensposition

Tangentbordet är tidigare beskrivet i avsnitt 2.

Styrelektroniken och tangentbordet ansluts till mikrodatorn via in-/utportar, såsom visas i figur 15.



Figur 15

Tangentbordsavkodaren ML 15 sköter självständigt avkodningen av tangentbordet och resultatet kan mikrodatorn läsa på inporten KBDATA. Bit 7 på KBDATA har värdet 0 om en tangent är nedtryckt och värdet 1 om alla tangenter är uppe. Bit 3-0 visar med ett värde $(0000)_2$ – $(1111)_2$ enligt figur 3 vilken tangent som är nere. Övriga bitar har värdet 0.

Portarna har följande symboliska adresser:

inport KBDATA

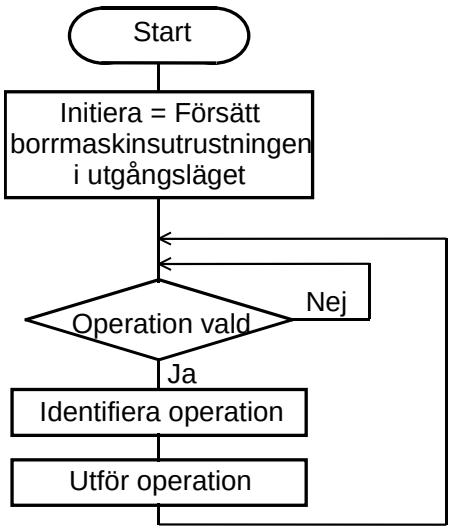
utport DRCTRL

inport DRSTAT

I prototypkopplingen har man tillgång till inporten KBDATA via ett kretskort ML 15, som kan kopplas in rakt ovanför mikrodatorkortet MC12. Portarna DRCTRL och DRSTAT finns på mikrodatorkortet MC12 och kopplas till borrmaskinen via en bred kontakt på kortsidan.

5.2 Programvaran

Utvecklingen av programvaran utgår ifrån det tidigare gjorda beslutet att systemet skall vara kommandostyrt. Programvaran får då lämpligen växa fram ur flödesplanen i figur 16, som delvis känns igen från figur 7. Programmet för att styra och övervaka utrustningen skall bestå av ett huvudprogram som på något sätt anropar subrutiner för de olika operationerna.



Figur 16

Det har tidigare också bestämts att systemet skall ha kommandon för vissa operationer, vilket innebär att operatören skall kunna starta operationerna från tangentbordet. Varje sådan operation skall i programvaran representeras av en subrutin. Det har bestämts att kommandona, d v s subrutinerna, skall läggas in på tangenterna i följande ordning

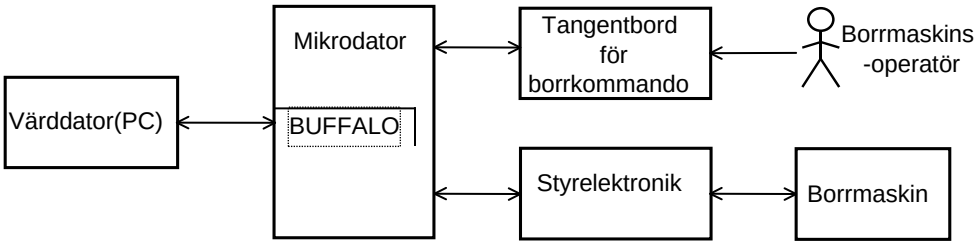
tangent nr	**operation**	**subrutin**
0	starta borrmotorn	START
1	stoppa borrmotorn	STOP
2	sänk borret	DOWN
3	höj borret	UP
4	rotera arbetsstycket medurs ett steg	STEP
5	borra ett hål	DRILL
6	stega arbetsstycket till referensposition och borra hål längs cirkeln enligt givet mönster.	AUTO

Programutvecklingen består fortsättningsvis av att modifiera och komplettera huvudprogrammet i figur 16.

Operationerna kan delas in i enkla operationer, START, STOP, DOWN, UP, STEP, och sammansatta operationer, DRILL och AUTO, där de sammansatta använder enkla operationer som komponenter.

5.3 Uttestning

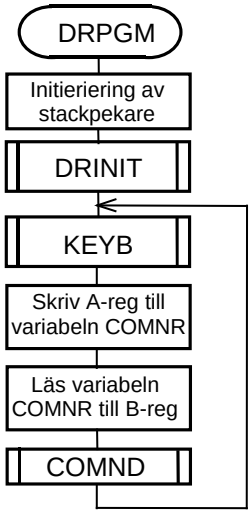
För att möjliggöra programskrivning, assemblering, testning, felsökning och felrättning har det mikroprocessorbaserade systemet kompletterats med en värddator (PC) som kommunicerar med ett monitorprogram, BUFFALO, i mikrodatorn MC12. Se figur 17. Monitorprogrammet BUFFALO beskrivs i ett separat häfte.



Figur 17.

5.4 Uppgifter

Huvudprogrammet DRPGM är utformat enligt flödesplanen i figur 18.



Figur 18

Subrutinerna DRINIT, KEYB och COMND har specificerats i avsnitt 2.

Subrutinen COMND avgör först vilket kommando som skall utföras och utför sedan kommandot som i sin tur är en subrutin. Subrutinen skrevs in i filen **comnd.s12** i uppgift 3b punkt 6.

Uppgift 4:

- Öppna filen **comnd.s12** och ändra innehållet i hopptabellen "JUMPTAB" till START, STOP, DOWN, UP, STEP, DRILL och AUTO. (Behåll Sub7!) Byt också namn på de tomma subrutinerna SUB2 – SUB6 till DOWN, UP, STEP, DRILL och AUTO.
- Assemblera filen **main.s12** och rätta eventuella fel som upptäcks i samband med detta. Tag om möjligt ut en utskrift av listfilen på en skrivare. Istället för att skriva ut listfilen på en skrivare kan man öppna listfilen med editorn och studera innehållet på bildskärmen. Studera listfilen!
- Ladda filen **main.s19** till simulatorn. Testa och felsök programmet, inklusive subrutinerna START och STOP. Ändra eventuella fel. Assemblera, ladda och testa programmet tills det fungerar som det skall.

¤¤¤

Subrutinerna för de enkla operationerna DOWN, UP och STEP specificeras på följande sätt:

DOWN

Beskrivning:	Rutinen kontrollerar först i DRCOPY att borrmotorn är startad. Om den är det sänks borret genom att drivenheten för elektromagneten aktiveras och DRCOPY uppdateras. Om borrmotorn inte är startad görs återhopp utan att borret sänks.
Anrop:	JSR DOWN
Indata:	Inga
Utdata:	Inga
Registerpåverkan:	CC-registret får påverkas
Anropade subr:	OUTONE

UP

Beskrivning:	Rutinen höjer borret genom att deaktivera elektromagneten. Den uppdaterar också DRCOPY och väntar tills borret är i sitt toppläge före återhopp.
Anrop:	JSR UP
Indata:	Inga
Utdata:	Inga
Registerpåverkan:	CC-registret får påverkas
Anropade subr:	OUTZERO

STEP

Beskrivning:	Rutinen kontrollerar först i DRCOPY att borret inte är sänkt och vrider sedan kretskortet ett steg medurs genom att mata ut riktningsinformation och stegpuls till stegmotorns drivenhet. Efter stegpulsen väntar rutinen 200 ms före återhopp så att arbetsstycket skall hinna vridas ett steg. Om borret är sänkt vid anrop av STEP utförs ingen stegning utan istället ges tre larmsignaler på "summern" före återhopp.
Anrop:	JSR STEP
Indata:	Inga
Utdata:	Inga
Registerpåverkan:	CC-registret får påverkas
Anropade subr:	ALARM, DELAY, OUTZERO, OUTONE

Kontrollen ovan av om borret är sänkt görs genom att undersöka värdet av signalen ”sänkanordning” i ordet DRCOPY. Man kan inte använda givaren för "borr nere" vid denna kontroll eftersom en borrning kan pågå utan att borret har hunnit igenom arbetsstycket!

I specifikationen har man också kommit fram till att alarmering skall kunna göras med hjälp av subrutinen

ALARM

Beskrivning:	Ger ett antal larmsignaler med längden 0,4 s och med 0,2 s mellanrum.
Anrop ex:	LDAB #2 Två larmsignaler skall ges JSR ALARM
Indata:	Antal larmsignaler i B-registret.
Utdata:	Inga
Registerpåverkan:	CC-registret får påverkas
Anropade subr:	DELAY, OUTZERO, OUTONE.

Uppgift 5:

- Skriv subrutinerna **DOWN**, **UP**, **STEP** och **ALARM**. Placera dem i filen dusa.s12.
- Inkludera filen **dusa.s12** i filen **main.s12** efter de andra inkluderade filerna. Assemblera och rätta de fel som upptäcks i samband med detta. Studera listfilen.
- Ladda maskinkoden i simulatorn. Testa, felsök och rätta subrutinerna DOWN, UP, STEP och ALARM genom att köra huvudprogrammet. ¤¤¤

För att ett hål skall kunna borras i en sammansatt operation behövs ett program som avgör om borret har arbetat sig igenom hela arbetsstycket innan det höjs. Subrutinen **DDTEST** (drill down test) skall sköta detta. DDTEST specificeras nedan.

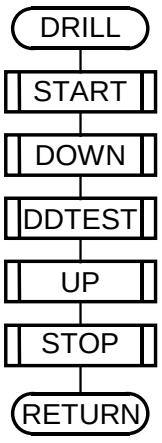
DDTEST

Beskrivning:	Undersöker via en givare om borret nått sitt botten-läge. Återhopp sker ej förrän borret nått bottenläget. Om borret ej har nått bottenläget inom 4 sekunder ges två larmsignaler på "summern" och återhopp sker. För att inte borrningen skall fördröjas onödigt länge bör man läsa av bottenlägesgivaren minst tio gånger per sekund.
Anrop:	JSR DDTEST
Indata:	Inga
Utdata:	Inga
Registerpåverkan:	CC-registret får påverkas
Anropade subr:	ALARM, DELAY.

Uppgift 6:

- Skriv subrutinen **DDTEST** enligt specifikationen ovan. Skriv in DDTEST i filen **drill.s12**.
- Skriv en subrutin **DRILL** enligt flödesplanen i figur 19. Skriv in den i filen **drill.s12**.
- Inkludera filen **drill.s12** i filen **main.s12** och assemblera sedan. Rätta eventuella fel som upptäcks i samband med detta. Studera listfilen.
- Testa, felsök och rätta subrutinerna DRILL (kommando nr 5) och DDTEST genom att köra huvudprogrammet i simulatorn på samma sätt som förut.

¤¤¤



Figur 19

Arbetsstycket skall roteras (stegas) till referenspositionen i subrutinen **REFPO**.

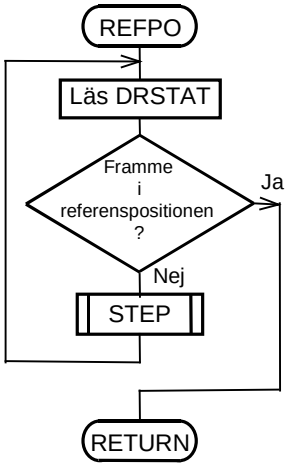
REFPO

Beskrivning:	Vrider arbetsstycket till referenspositionen genom att läsa av läsgaffeln och vrida arbetsstycket ett steg i taget tills referenspositionen detekteras.
Anrop:	JSR REFPO
Indata:	Inga
Utdata:	Inga
Registerpåverkan:	CC
Anropade subr:	STEP

Uppgift 7:

- Skriv subrutinen **REFPO** enligt flödesplanen i figur 20 och mata in den i filen **auto.s12**.
- Inkludera filen **auto.s12** i filen **main.s12** och assemblera sedan. Rätta eventuella fel som upptäcks i samband med detta. Studera listfilen.

¤¤¤



Figur 20.

När arbetsstycket befinner sig i referenspositionen eller när ett hål har borrats i en position skall arbetsstycket roteras ett antal steg till positionen för nästa hål. Detta skall göras i subrutinen **NSTEP** som specificeras enligt

NSTEP

Beskrivning:	Roterar arbetsstycket N steg medurs.
Anrop ex:	LDAA #2 Stega två steg JSR NSTEP
Indata:	Antalet steg (0-255) i A-registret.
Utdata:	Inga
Registerpåverkan:	CC-registret får påverkas
Anropade subr:	STEP

Man har bestämt att hålmönstret för ett arbetsstycke (ett varv) skall specificeras i en tabell, med basadressen **PATTERN**, såsom visas i figur 21. I denna anges antalet steg till nästa hål. Talet $\mathbf{FF_{16}}$ används för att markera att inga fler hål finns.

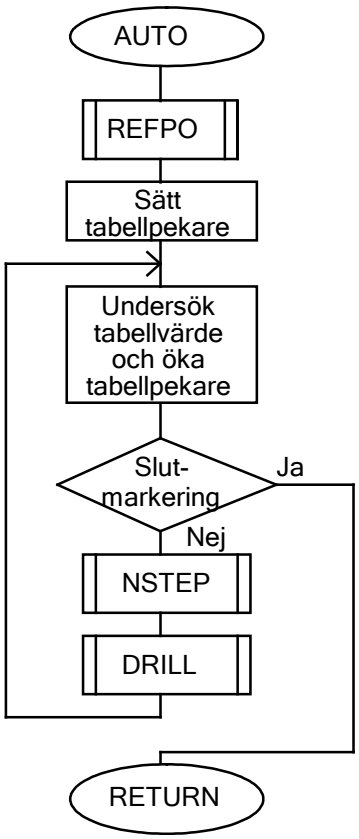
Hål nr	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	-
Antal steg	0	1	1	1	1	1	1	1	2	1	5	2	2	2	2	4	4	3	8	2	FF

Figur 21

Uppgift 8:

- Skriv subrutinen **NSTEP** och mata in den i filen **auto.s12**. Tänk på att rutinen skall fungera även för A = 0, dvs att man redan står där man skall borra! I det fallet skall man alltså inte stega alls.
- Skriv också en subrutin **AUTO** enligt flödesplanen i figur 22 till höger. Mata in den i filen **auto.s12**. Lägg även in håltabellen enligt figur 21. Håltabellen skall börja på adressen PATTERN, som lämpligen placeras direkt efter RTS-instruktionen i AUTO.
- Assemblera filen **main.s12.** Rätta eventuella fel som upptäcks i samband med detta. Studera listfilen.
- Testa, felsök och rätta subrutinerna REFPO, NSTEP och AUTO genom att köra huvudprogrammet (tangent 6) i simulatorn på samma sätt som förut.

¤¤¤



Figur 22

6 Användning av avbrottssystemet hos 68HCS12

För att illustrera användning av processorns avbrottssystem skall styrsystemet för borrmaskinen först kompletteras med tryckknappar för avbrottsgenerering. Dessa avbrott kallas **händelsestyrda** och skall ta hand om dels nödstopp och dels signalering med summern.

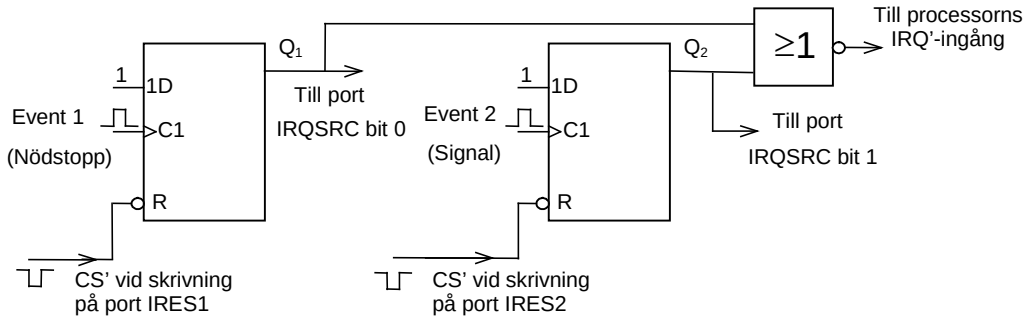
Senare skall styrsystemet kompletteras med en pulsgenerator kopplad till processorns avbrottssystem. Pulsgeneratorns utsignal är en binär signal med konstant frekvens, som orsakar avbrott med samma frekvens. Denna typ av avbrott kallas **tidsstyrda** och skall användas för att få processorn att skenbart köra flera program samtidigt, sk **pseudoparallellism**.

6.1 Händelsestyrda avbrott

Den naturliga lösningen för att realisera nödstopp med hjälp av processorn är att använda icke maskerbart avbrott (NMI). På laborationsdatorn finns dock ingen möjlighet att koppla en yttre signal till processorns NMI-ingång (XIRQ). Endast IRQ-ingången finns tillgänglig och används därför för nödstoppet.

När ***nödstoppsknappen*** trycks ned skall alltså ett IRQ-avbrott genereras. Detta skall utformas så att processorn överger det program den håller på att köra och istället utför ett hopp till huvudprogrammets (DRPGM) start. Där anropas subrutinen DRINIT som bland annat stänger av alla funktioner hos borrmaskinen. Om ett kommando håller på att utföras när ett nödstoppsavbrott kommer så avslutas kommandot direkt och alla funktioner hos borrmaskinen blir sedan passiva när subrutinen DRINIT utförs i början av huvudprogrammet. Nya kommandon kan sedan ges på samma sätt som vid normal start av programmet.

När ***signalknappen*** trycks ned skall också ett IRQ-avbrott genereras. I detta fall skall summern signalera och därefter skall det avbrutna programmet fortsätta körningen där det blev avbrutet. Det skall alltså vara möjligt att signalera med summern medan tex kommandot (rutinen) AUTO utförs. I figur 23 visas den koppling som skall användas för att generera avbrotten.



Figur 23.

Av figur 23 framgår att nedtryckning av någon av switcharna **1** eller **2** gör att en av D-vipporna ettställs. Detta medför i sin tur att en **IRQ'**-signal (Interrupt Request) sänds till processorn via NOR-grinden. Om flaggbiten **I** i processorns flaggregister **CC** är nollställd kommer processorn att avbryta det pågående programmet och hoppa till en avbrottsrutin som pekas ut av IRQ-vektorn.

I avbrottsrutinen måste först källan till avbrottet bestämmas eftersom det i detta fall finns två möjligheter. Avbrottsrutinen måste därför läsa port **IRQSRC** och med hjälp av bitpositionerna **0** och **1** avgöra vilken vippa som är ettställd.

Avbrottsrutinen fortsätter sedan med att antingen "nödstoppsdelen" eller "signaldelen" utförs beroende på vilken vippa som var ettställd.

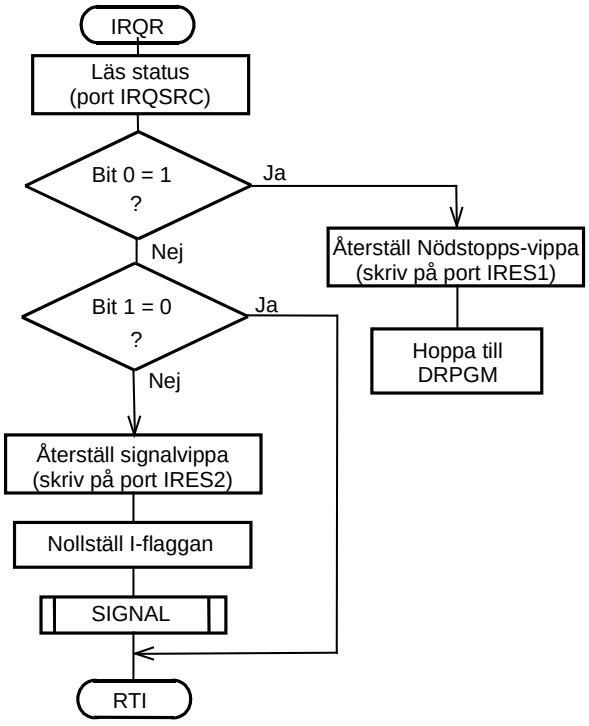
I början av "nödstopps"- resp. ”signaldelen” av avbrottsrutinen nollställs motsvarande avbrottsvippa genom att en skrivning görs på adressen IRES1 eller IRES2 eftersom ”chip-select”-signalerna för dessa skrivningar är kopplade till var sin ”reset”-ingång på D-vipporna.

Om en ettställd avbrottsvippa inte nollställs i avbrottsrutinen skulle den direkt generera ett nytt avbrott då man återvänder till huvudprogrammet från avbrottsrutinen.

En flödesplan för avbrottsrutinen visas i figur 24.

Uppgift 9a: Skriv en avbrottsrutin **IRQR** enligt flödesplanen i figur 24. Mata in den i filen **irqr.s12**. Subrutinen SIGNAL skall ge en larmsignal på "summern".

Skriv också subrutinen SIGNAL, som skall ge en larmsignal på summern med hjälp av subrutinen ALARM. Mata in subrutinen SIGNAL i filen **irqr.s12**. ¤¤¤



Figur 24.

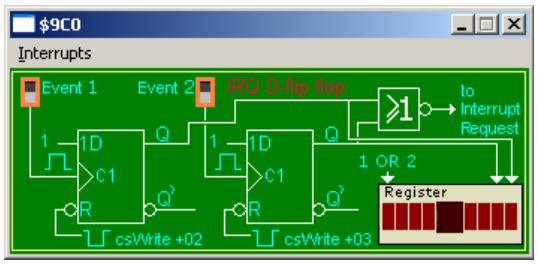
För att avbrottssystemet skall kunna användas måste I-flaggan i processorns CC-register först nollställas och adressen till avbrottsrutinen placeras i IRQ-vektorn. Dessa initieringar skall göras i rutinen **IRQINIT**.

Då matningsspänningen slås på för D-vipporna kan de hamna i vilket läge som helst. Eftersom båda vipporna skall ha Q-värdet noll från början nollställer man dem i rutinen IRQINIT genom att skriva på utportarna IRES1 och IRES2. Detta måste göras innan avbrottssystemet kopplas på, dvs innan I-flaggan nollställs

På grund av att IRQ-vektorn finns på adressen $FFF2_{16}$ i ROM (ej skrivbart) har IRQ-vektorn i labsystemet (dvs MC12-datorn) och simulatorn "flyttats" till adressen $3FF2_{16}$ som alltså skall innehålla adressen till avbrottsrutinen.

Uppgift 9b:

- Skriv en subrutin **IRQINIT** som nollställer D-vipporna, skriver in avbrottsrutinens adress i avbrottsvektorn för IRQ och aktiverar avbrotts-systemet. IRQINIT skall anropas från huvudprogrammet efter anropet av subrutinen DRINIT. Placera subrutinen i filen **irqr.s12**.
  Adresser finns i Appendix 6.
- Inkludera filen **irqr.s12** i filen **main.s12** och assemblera.
  Rätta eventuella fel som upptäcks i samband med detta. Studera listfilen.
- Ladda programmet i simulatorn och högerklicka i simulatorfönstret.
  Välj *"Simulator Setup"* och klicka på knappen *"Attach New Device"*.
  Välj *"IRQ flip flop"*och skriv in adressen $DC0_{16}$. Fönstret nedan visas:
- Klicka på *"Interrupts"* i listen uppe till vänster i fönstret och välj *"Enable"*. Klicka sedan på knapparna Event 1 och Event 2 så att bägge D-vipporna ettställs. Stega sedan programmet fram till anropet av subrutinen IRQINIT och kontrollera att den nollställer D-vipporna genom att skriva på adresserna IRES1 och IRES2 samt placerar rätt innehåll i IRQ-vektorn och till sist nollställer I-flaggan.



- Placera en brytpunkt i början av avbrottsrutinen (adressen kan man hitta i listfilen **main.lst**) och starta i läget RUN. Programmet skall nu bete sig precis som tidigare!
- Testa sedan avbrotten genom att först trycka på knappen *Event 1*. Om programmet fungerar skall körningen då stoppas vid brytpunkten i början av avbrottsrutinen. Fortsätt att stega avbrottsrutinen och kontrollera att ett hopp görs tillbaka till huvudprogrammets start.
- Testa sedan det andra avbrottet genom att trycka på knappen *Event 2* och fortsätt stega efter brytpunkten. Kontrollera att "signalavbrottet" uppför sig på önskat sätt. ¤¤¤

6.2 Tidsstyrda avbrott, pseudoparallellism

Styrprogrammet är uppbyggt så att under tiden ett kommando utförs kan inget nytt kommando ges från tangentbordet. Det beror på att processorn är upptagen med att utföra det senast mottagna kommandot. Denna effekt är speciellt tydlig när "långa" kommandon som AUTO utförs. Ett sätt att kringgå detta problem visade vi i förgående uppgift. Vi använde yttre logikkretsar tillsammans med processorns avbrottsystem för att ge kommandon för nödstopp och alarmering. Om alarmeringskommandot ges när processorn utför ett "längre" kommando som AUTO visar det sig att denna metod har en påtaglig nackdel.

Uppgift 10a:

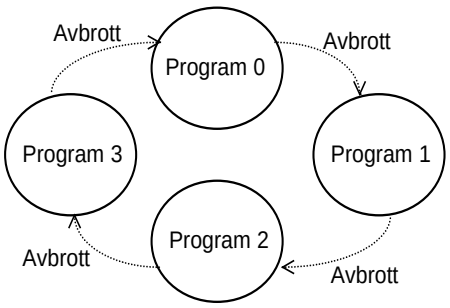
Vilken? _____

_____ ¤¤¤

För att känna av tangentbordet, styra borrmaskinen och ge alarmsignal samtidigt måste processorn kunna köra motsvarande programavsnitt samtidigt. Eftersom processorn bara kan köra ett program åt gången förefaller det omöjligt att realisera samtidig körning av flera program eller programavsnitt med bara en processor. Man kan dock åstadkomma detta med en metod som beskrivs nedan.

- Man skriver separata program (med separata stackar) för de aktiviteter, som processorn skall styra samtidigt.
- En pulsgenerator med konstant, relativt hög frekvens (ca 20-500Hz), kopplas till processorns avbrottssystem, via en avbrottsvippa. Vid varje avbrott byter avbrottsrutinen program på så sätt att ett annat program än det som blev avbrutet körs efter avbrottsrutinen.

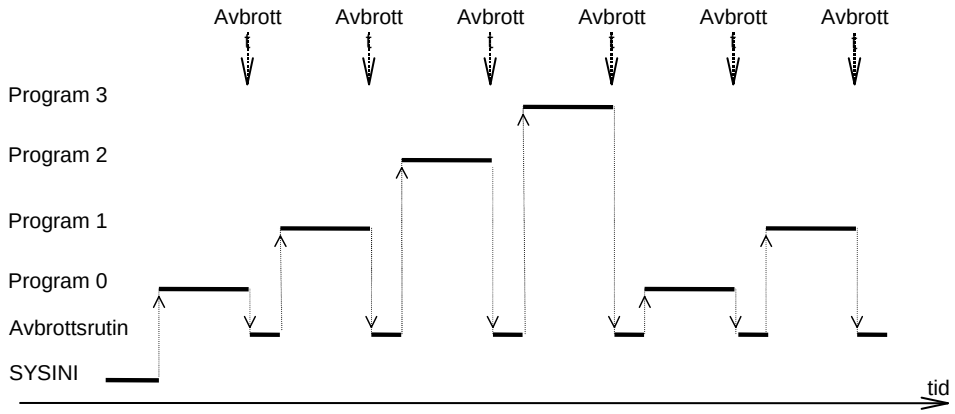
Om man har fyra olika program som skall köras ”samtidigt” kan man t ex se till att det program som körs växlar enligt figur 25.



Figur 25

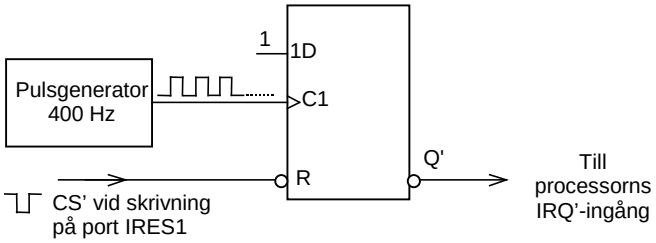
Om avbrotten genereras med tillräckligt hög frekvens får man intrycket att programmen körs samtidigt (pseudoparallellism).

I Appendix 8 ges ett program, SYSINI, som tillsammans med en avbrottsrutin växlar mellan fyra olika användarprogram, program 0-3, enligt figur 25 och 26.



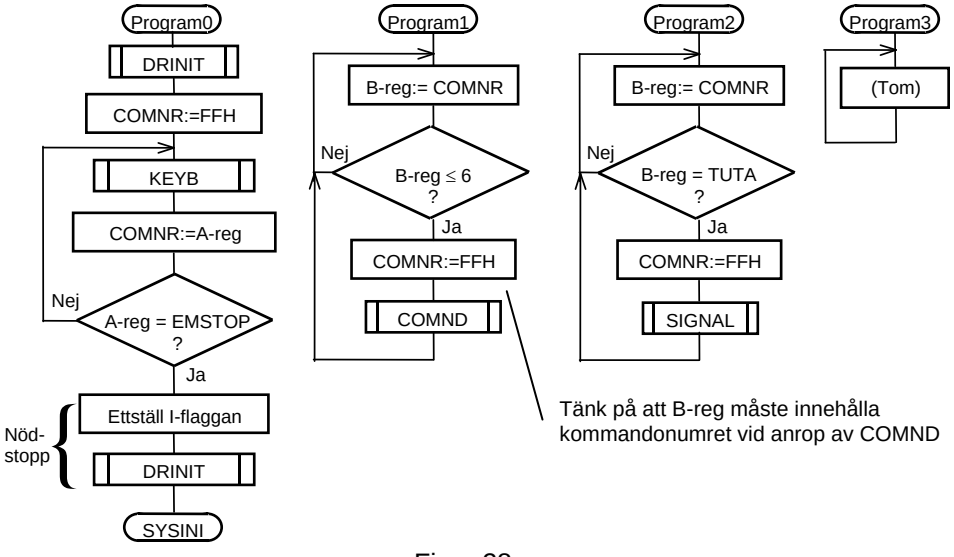
Figur 26

Avbrottsbegäran till processorn kommer från en avbrottsvippa som triggas av en pulsgenerator på labplatsen, se figur 27. Avbrottsvippan genererar avbrotts-begäran (låg nivå) på processorns IRQ' - ingång med tidsintervallet 2,5 ms. Vippans avbrottsbegäran återställs i avbrottsrutinen med en skrivning på adressen IRES1, se figur 27.



Figur 27

Ett förslag till hur borrmaskinsprogrammet kan delas upp i parallella processer visas i flödesplanerna i figur 28 nedan.



Figur 28

Variabeln COMNR innehåller borrkomandot ($0\text{-}F_{16}$) som senast har tagits emot från subrutinen KEYB i *Program0*. Innan kommandot utförs i *Program1* eller *Program2* skriver dessa processer in värdet FF_{16} i COMNR. På detta sätt utförs kommandot endast en gång.

Uppgift 10b:

- Studera programlistningen i Appendix 8. En fil **sysini.s12** med samma innehåll, finns på kursens hemsida.
- Kopiera denna fil och lägg in den i ert laborationsbibliotek.
- Borrmaskinsprogrammet delas upp i tre parallella processer, program0 - program2, enligt figur 28. Skriv in programavsnitten 0-2 i filen **sysini.s12**. Använd tangent **7** som **TUTA** och tangenten $\mathbf{F_{16}}$ som **EMSTOP** (nödstopp).

  Lämna ***Program 3*** oförändrat. Vi har inte någon lämplig uppgift för det.
- Öppna filen **main.s12** och spara den under namnet **main1.s12** i laborationsbiblioteket. Tag bort ORG-direktivet i början på filen **main1.s12!**
- Inkludera filen **main1.s12** sist i filen **sysini.s12**. På detta sätt blir samtliga tidigare skrivna subrutiner tillgängliga eftersom deras filer är inkluderade i **main1.s12**.
- Assemblera filen **sysini.s12** och rätta eventuella fel. Ladda programmet till simulatorn. Använd samma I/O-konfigurering som i förra uppgiften. Testa och felsök sedan programmet genom att byta process upprepade gånger med "*Event 1*"-knappen. ¤¤¤

Det återstår ett par problem i det nuvarande programpaketet. Sannolikheten för att det första problemet skall märkas är dock så låg att man förmodligen inte noterar det.

Problemet uppstår t ex om ett program (en process) läser DRCOPY och sedan blir avbrutet av ett annat program (process) som läser och modifierar DRCOPY. När det första programmet (processen) återupptar sin körning kommer det att manipulera sitt (numera inaktuella) värde på DRCOPY och sedan skriva tillbaka ett felaktigt värde på DRCOPY. Det område i programmet som kan skapa problem, dvs programkoden från det att DRCOPY läses tills dess att den skrivs tillbaka, kallas för **kritisk region**. Vi måste därför förhindra att ett program (en process) blir avbrutet när det befinner sig i en kritisk region.

Uppgift 11a: Hur förhindrar man att ett avsnitt i ett program (en process) blir avbrutet?

_____

_____

Modifiera subrutinerna OUTONE och OUTZERO i filerna **start.s12** resp **stop.s12** så att de ej kan avbrytas i de kritiska regionerna. ¤¤¤

Uppgift 11b: Assemblera filen **sysini.s12** och rätta eventuella fel. Ladda programmet till simulatorn. Testa och felsök sedan programmet i simulatorn genom att byta process upprepade gånger med *"Event 1"*-knappen. Undersök speciellt vad som händer om avbrotten inträffar i de kritiska regionerna. ¤¤¤

När programmet fungerar och senare används för att styra borrmaskinen upptäcker man att operationerna utförs betydligt långsammare än tidigare. Försök att fundera ut vad detta beror på. (Tänk på hur subrutinen DELAY fungerar.)

Genom att "skala om" DELAY-rutinen kan man se till att den ger rätt fördröjningsvärde i detta fall.

En mer generell lösning är att DELAY-rutinen använder en sk realtidsklocka, där man kan läsa av den verkliga tiden.

Uppgift 12:

- Skriv en ny subrutin DELAY3, som ger en fördröjningstid som gör att operationerna utförs i samma takt som tidigare.
  Placera subrutinen DELAY3 sist i filen **keyb.s12**.

Tänk på att DELAY3-rutinen **måste** vara *reentrant* d v s inga globala variabler (= fasta minnesadresser för variabler) får användas, utan enbart stacken och registren. Reentrance innebär att flera processer "samtidigt" kan använda samma subrutin.

- Gör en kopia av filen **sysini.s12** och placera den i laborationsbiblioteket. Ge kopian namnet **sysini1.s12**. Ändra filen **sysini1.s12** så att filen **main2.s12** inkluderas i slutet istället för **main1.s12**.
- Gör en kopia av filen **main1.s12** och placera den i laborationsbiblioteket. Ge kopian namnet **main2.s12**. Ändra definitionen DELAY EQU DELAY2 i slutet av filen **main2.s12** till DELAY EQU DELAY3.
- Assemblera filen **sysini1.s12** och rätta eventuella fel. Programmet behöver inte simuleras. ¤¤¤

Appendix 1. Huvudprogram 1 för borrmaskinsstyrning

```
**********************************************************************************************
*               Symboldefinitioner
KBDATA    EQU   $09C0       Inport för avläsning av tangentnummer
COMNR     EQU   $2FFF       COMNR är kommandonumret
DRCOPY    EQU   $2FFE       DRCOPY är en kopia av senaste styrord
BOS       EQU   $2FEF       BOS är Bottom Of Stack
BEGIN     EQU   $2000       BEGIN är startadressen för RWM
DRCTRL    EQU   $0400       Utport för styrordet till borrmaskinen
DRSTAT    EQU   $0401       Inport för givare på borrmaskinen
**********************************************************************************************
*               Huvudprogram
**********************************************************************************************
          ORG   BEGIN       Start för huvudprog
DRPGM     LDS   #BOS
*
          JSR   DRINIT      Initieringar för passiv maskin
*
COLOOP    JSR   KEYB        Hämta kommandonummer till reg A
*
          STAA  COMNR       Uppdatera variabeln COMNR
*
          LDAB  COMNR       Hämta COMNR-variabeln
*
          JSR   COMND       Utför kommando
*
          JMP   COLOOP      Hämta nästa COMNR-värde
*
**********************************************************************************************
*SUBRUTIN – DRINIT
*Beskrivning:  Rutinen försätter borrmaskinen i passivt tillstånd.
*Anrop:        JSR DRINIT
*Indata:       Inga
*Utdata:       Inga
*Reg-påverkan: Register CC
*Anr subr:     Ingen
**********************************************************************************************
DRINIT    CLR   DRCTRL      Beg värde till borrmaskin
          CLR   DRCOPY      Beg värde till kopian
          RTS
*
**********************************************************************************************
*
KEYB2     RTS               Tomma subrutiner för test
COMND2    RTS
DELAY2    RTS
*
**********************************************************************************************
**********************************************************************************************
*
KEYB      EQU   KEYB2       Den "tomma" KEYB-rutinen
COMND     EQU   COMND2      Den "tomma" COMND-rutinen
DELAY     EQU   DELAY2      Välj den tomma subrutinen för test med simulator
**********************************************************************************************
```

Lämpligt ställe för en brytpunkt!

Appendix 2. Programlistningar för KEYB och DELAY

```
*********************************************************************
*SUBRUTIN – KEYB
*Beskr:     Rutinen väntar tills samtliga tangenter är uppe.
*          Därefter registreras första nedtryckta tangent.
*Anrop:     JSR KEYB
*Indata:    Inga
*Utdata:    Tangentnummer (0-F16) för nedtryckt tangent i register A.
*Reg-påv:   Register A,CC
*Anr subr:  Ingen
*********************************************************************
KEYB1     PSHX
*
KEY1      TST     KBDATA      Läs tangentdataregister
          BPL     KEY1        Vänta om någon tangent nere (bit 7 = 0)
*
KEY2      LDAA    KBDATA      Läs tangentdataregister
          BMI     KEY2        Vänta om alla uppe (bit 7 = 1)
*
          LDX     #KEYTAB     Pekare till tangenttabell
          LDAA    A,X         Hämta tangentnummer i tabell
*
          PULX
          RTS
*
KEYTAB    FCB     0,4,8,12,1,5,9,13,2,6,10,14,3,7,11,15
*****************************************************************************************
*SUBRUTIN - DELAY
*Beskrivning:  Skapar en fördröjning om N * 50 ms.
*Anrop ex:     LDAA    #6          *Fördröjning 6*50 ms = 300 ms
*              JSR     DELAY
*Indata:       Antal intervall, N, om 50 ms i A-registret
*Utdata:       Inga
*Reg-påverkan: CC-registret får påverkas
*Anrop subr:   Ingen.
*****************************************************************************************
DELAY1    PSHA                  DELAY sätts lika med DELAY1 vid körning i HC12
          PSHX                  A och X till stacken
*
          TSTA
          BEQ   DERET           Fördröjningsvärde noll
*
ALOOP     LDX   #10000          10000x5 mikrosek = 50 ms fördröjning
*
XLOOP     DEX                   Denna del skall justeras tills den tar 5 mikrosek
          NOP
          NOP
          BNE   XLOOP
*
          DECA
          BNE   ALOOP
*
DERET     PULX
          PULA
          RTS
*********************************
```

I subrutinhuvudet ovan beskrivs den DELAY-subrutin som skall användas när programmet körs av den verkliga processorn 68HCS12 och styr den verkliga borrmaskinen.

Vid stegning av programmet i simulatorn skall vi använda en förenklad subrutin DELAY2, som bara består av instruktionen RTS. Subrutinen DELAY2 definieras i huvudprogrammet.

Vid körning av programmet i simulatorn med Run bör man använda DELAY-rutinen som visas på denna sida, men med mycket kortare fördröjningstid.

Appendix 3. Huvudprogram 2 för borrmaskinsstyrning

```
*****************************************************************************************
*               Symboldefinitioner
KBDATA      EQU     $09C0       Inport för avläsning av tangentnummer
COMNR       EQU     $2FFF       COMNR är kommandonumret
DRCOPY      EQU     $2FFE       DRCOPY är en kopia av senaste styrord
BOS         EQU     $2FEF       BOS är Bottom Of Stack
BEGIN       EQU     $2000       BEGIN är startadressen för RWM
DRCTRL      EQU     $0400       Utport för styrordet till borrmaskinen
DRSTAT      EQU     $0401       Inport för givare på borrmaskinen
*****************************************************************************************
*               Huvudprogram
*****************************************************************************************
            ORG     BEGIN       Start för huvudprog
DRPGM       LDS     #BOS
*
            JSR     DRINIT      Initieringar för passiv maskin
*
COLOOP      JSR     KEYB        Hämta kommandonummer till reg A
            STAA    COMNR       Uppdatera variabeln COMNR
*
            LDAB    COMNR       Hämta COMNR-variabeln
*
            JSR     COMND       Utför kommando
*
            JMP     COLOOP      Hämta nästa COMNR-värde
*****************************************************************************************
*SUBRUTIN – DRINIT
*Beskrivning:   Rutinen försätter borrmaskinen i passivt tillstånd.
*Anrop:         JSR DRINIT
*Indata:        Inga
*Utdata:        Inga
*Reg-påverkan:  Register CC
*Anr subr:      Ingen
*****************************************************************************************
DRINIT      CLR     DRCTRL      Beg värde till borrmaskin
            CLR     DRCOPY      Beg värde till kopian
            RTS
*
*****************************************************************************************
*
            USE     keyb.s12    Här hamnar innehållet i filen keyb.s12
*
*****************************************************************************************
*
KEYB2       RTS                 Tom subrutin för test
*
COMND2      RTS                 Tom subrutin för test
*
DELAY2      RTS                 Tom subrutin för test
*
*****************************************************************************************
*
KEYB        EQU     KEYB1       Den “riktiga” KEYB-rutinen

COMND       EQU     COMND2      Här väljer vi de tomma subrutinerna för test
DELAY       EQU     DELAY2
```

Appendix 4. Kommandosubrutin för borrmaskinsstyrning

```
*****************************************************************************************
*               SUBRUTIN - COMND
*Beskrivning:   Rutinen avgör vilken kommandosubrutin som skall anropas och anropar denna.
*Anrop ex:      LDAB    #5          Sätt kommando nummer 5
*               JSR     COMND       Kommando nr 5 utförs
*Indata:        Kommandonummer i reg B
*Utdata:        Inga
*Reg-påverkan: CC-registret får påverkas
*Anrop subr:    SUB0 - SUB7
*****************************************************************************************
MAX             EQU     7           Maxvärde för kommandonummer
COMND1          PSHB                Spara register på stack
                PSHX                A, B och X till stacken
*
                CMPB    #MAX        Giltigt värde?
                BHI     COMEX       Nej, hoppa ut
*
                LDX     #JUMPTAB    Ja, pekare till hopptabell
                ASLB                2 bytes per adress
                LDX     B,X         Hämta hoppadress från tabell
*
                JSR     ,X          Utför kommandosubrutin
*
COMEX           PULX                Återställ register
                PULB
                RTS
*
*****************************************************************************************
*               Tabell med subrutinadresser
*
JUMPTAB         FDB     SUB0,SUB1,SUB2,SUB3,SUB4,SUB5,SUB6,SUB7
*****************************************************************************************
SUB0            RTS                 Tomma subrutiner
SUB1            RTS
SUB2            RTS
SUB3            RTS
SUB4            RTS
SUB5            RTS
SUB6            RTS
SUB7            RTS
*****************************************************************************************
*
*****************************************************************************************
```

Appendix 5. Programlistning för START

```
*******************************************************************************************
*SUBRUTIN - START
*Beskrivning:        Rutinen startar borrmotorn, uppdaterar DRCOPY och väntar i 200 ms
*                    för att borret skall uppnå rätt hastighet.
*Anrop:              JSR  START
*Indata:             Inga
*Utdata:             Inga
*Registerpåverkan:   CC-registret får påverkas
*Anropade subrutiner: OUTONE,DELAY
*******************************************************************************************
START     PSHA
          PSHB                 A och B till stacken
*
          LDAB   #$02
          JSR    OUTONE        Ettställ bit 2 i styrordet till borrmaskinen
*
          LDAA   #$04
          JSR    DELAY         Fördröjning 4*50 ms = 200 ms
*
          PULB                 Återställ reg enl ovan
          PULA
          RTS
*******************************************************************************************
```

Appendix 6. Adresser för borrmaskinsstyrning

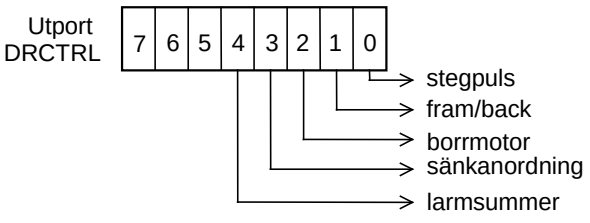
Startadresser i RWM: Huvudprogram DRPGM 2000_{16}

Andra använda adresser i RWM:

Variabeln COMNR (8 bitar)	COMNR	$2FFF_{16}$
Kopia av borrmaskinens styrord	DRCOPY	$2FFE_{16}$
"Bottom-of-stack"	BOS	$2FEF_{16}$
Avbrottsvektor för IRQ (I MC12-datorn och simulatorn.)		$3FF2_{16}$ **(OBS!)**

Portadresser:

(inport)	KBDATA	$09C0_{16}$
(inport)	IRQSRC	$0DC0_{16}$
(utport)	IRES1	$0DC2_{16}$
(utport)	IRES2	$0DC3_{16}$
(utport)	DRCTRL	0400_{16}
(inport vid simulering)	DRSTAT	0401_{16}
(inport i MC12-datorn)	DRSTAT	0600_{16} **(OBS!)**



Håltabell för kommandot AUTO:

Hål nr	1	2	3	4	5	6	7													-
Antal steg	0	1	1	1	1	1	1	1	2	1	5	2	2	2	2	4	4	3	8	2	FF

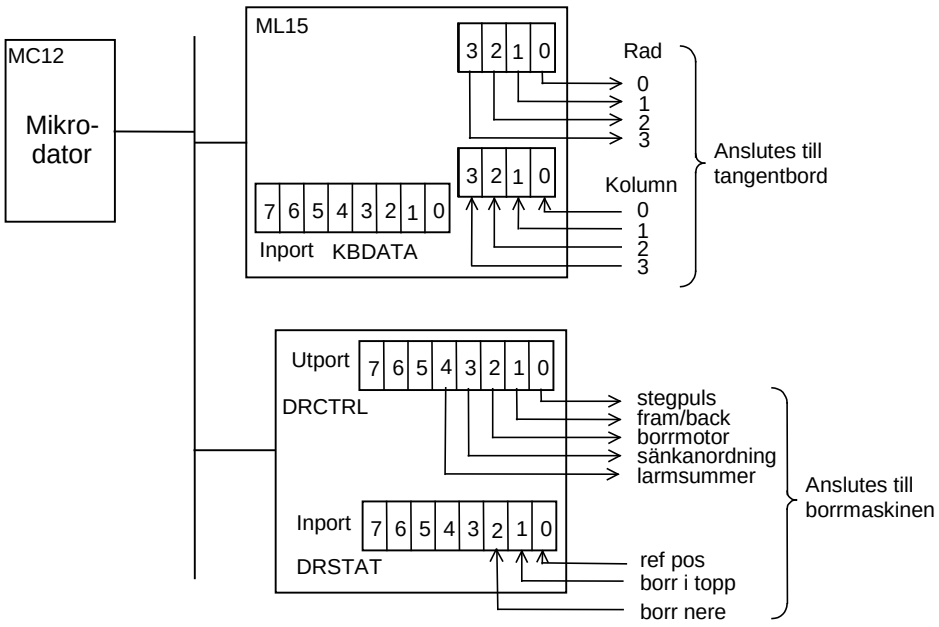
Appendix 7.

Tangentbordsdisposition

Rad \ Kolumn	3	2	1	0
0	0	1	2	3
1	4	5	6	7
2	8	9	A	B
3	C	D	E	F

START	STOP	DOWN	UP
STEP	DRILL	AUTO	

Hårdvarukoppling



Stegpuls:	Positiv flank stegar.
Fram/back:	1 ger medurs stegning.
Borrmotor:	1 startar motorn.
Sänkanordning:	1 sänker borret.
Larmsummer:	1 ger signal.
Referensposition:	Framme = 1.
Toppläge:	Topp = 1.
Bottenläge:	Botten = 1.

Appendix 8.

```
*           Definitioner av konstanter
BEGIN       EQU     $2000           Startadress för SYSINI
BOS         EQU     $2F00           Bottom of stack
DRCOPY      EQU     $2FFE           Kopia av borrkommando
DRCTRL      EQU     $0400           Portadress för borrkommando
IRQVEC      EQU     $3FF2           Ändrad från $FFF2 för MC12
IRES1       EQU     $0DC2           Clearadress avbrottsvippa 1
IRES2       EQU     $0DC3           Clearadress avbrottsvippa 2
STSIZE      EQU     $40             Stackstorlek för varje process
*****************************************************************
*           Program SYSINI
*           Utför nödvändiga initieringar och skapar fyra
*           parallella processer
*****************************************************************
            ORG     BEGIN
*           Skapar färdiga stackar för program 1-3
SYSINI      LDX     #PROGR1         Återhoppsadress till progr 1
            STX     BOS-(1*STSIZE)-2
            LDX     #PROGR2         Återhoppsadress till progr 2
            STX     BOS-(2*STSIZE)-2
            LDX     #PROGR3         Återhoppsadress till progr 3
            STX     BOS-(3*STSIZE)-2
*
            LDAA    #$C0            CC-registrets innehåll,
            STAA    BOS-(1*STSIZE)-9    med S=1,X=1,I=0 och övriga=0.
            STAA    BOS-(2*STSIZE)-9
            STAA    BOS-(3*STSIZE)-9
*
            Skapar tabell med stackpekare till processerna
            LDX     #BOS-(1*STSIZE)-9
            STX     SPTAB+2
            LDX     #BOS-(2*STSIZE)-9
            STX     SPTAB+4
            LDX     #BOS-(3*STSIZE)-9
            STX     SPTAB+6
*
            LDX     #IRQRUT         Sätt IRQ-vektorn
            STX     IRQVEC          MC12-vektorn
*
            CLR     CURPRG          Program 0 skall exekveras
            LDS     #BOS            Stackpekare för program 0
*
            STAA    IRES1           Nollställ båda avbrottsvipporna
            STAA    IRES2           för säkerhets skull!
************************************************
            CLI                     Tillåter I-avbrott
            JMP     PROGR0
*
*****************************************************************************************
*   Dataarea
*****************************************************************************************
CURPRG      RMB     1               Aktivt program
SPTAB       RMB     8               Tabell för processernas stackpekare
*
*****************************************************************************************
```

```
*****************************************************************************************
*           Avbrottshanterare IRQRUT
*           Byter process (Round robin, modulo 4), visar process-
*           nummer på DRCTRL, bit 7 och 6. Nollställer avbrottsvippa
*****************************************************************************************
*
IRQRUT    LDAB    CURPRG          Vilket program exekverar?
          LDX     #SPTAB          Adress till stackpekartabell i X
          STS     B,X             Spara dess stackpekare i tabellen!
*
          ADDB    #2              Nästa modulo-4 på tur
          ANDB    #%00000110      Sekvens 000,010,100,110,000,...
          STAB    CURPRG
*
          LDS     B,X             Laddar stackpekare för nästa process från tabell
*
          JSR     SHWPROC         Nästa processnummer till DRCTRL bit 7,6
*
          STAA    IRES1           Nollställ båda avbrottsvipporna
          STAA    IRES2           för säkerhets skull!
*
          RTI                     Återhopp till nästa process
*
*****************************************************************************************
*SUBRUTIN – SHWPROC
*Beskrivning:    Subrutin som visar processnumret på bit 7,6 på port DRCTRL.
*Anrop:          JSR SHWPROC
*Indata:         2 * processnr i B-reg
*Utdata:         Inga
*Reg-påverkan: Register A,B,CC
*Anr subr:       Ingen
*****************************************************************************************
SHWPROC   LDAA    #$20            Processnummer till reg B bit 7,6.
          MUL
*
          LDAA    #%00111111      Nollställ bit 7,6 i DRCOPY
          ANDA    DRCOPY
          STAA    DRCOPY
*
          ORAB    DRCOPY          Processnummer till DRCOPY bit 7,6.
          STAB    DRCOPY
          STAB    DRCTRL          Processnummer till DRCTRL bit 7,6.
          RTS
*
*****************************************************************************************
*           De parallella processerna. Här ska du(ni) lägga in
*           din(er) kod i uppgift 10b, eller hopp till din(er) kod
*****************************************************************************************
PROGR0    NOP                     Program 0
          JMP     PROGR0
*****************************************************************************************
PROGR1    NOP                     Program 1
          JMP     PROGR1
*****************************************************************************************
PROGR2    NOP                     Program 2
          JMP     PROGR2
*****************************************************************************************
PROGR3    NOP                     Program 3
          JMP     PROGR3
*****************************************************************************************
```

Då SYSINI avslutas med hopp till program 0 finns följande innehåll i SP, SP-tabell och stackar:

SP: BOS = $2F00

SPTAB+0: ?
SPTAB+2: BOS-1*STSIZE-9 = $2EB7
SPTAB+4: BOS-2*STSIZE-9 = $2E77
SPTAB+6: BOS-3*STSIZE-9 = $2E37