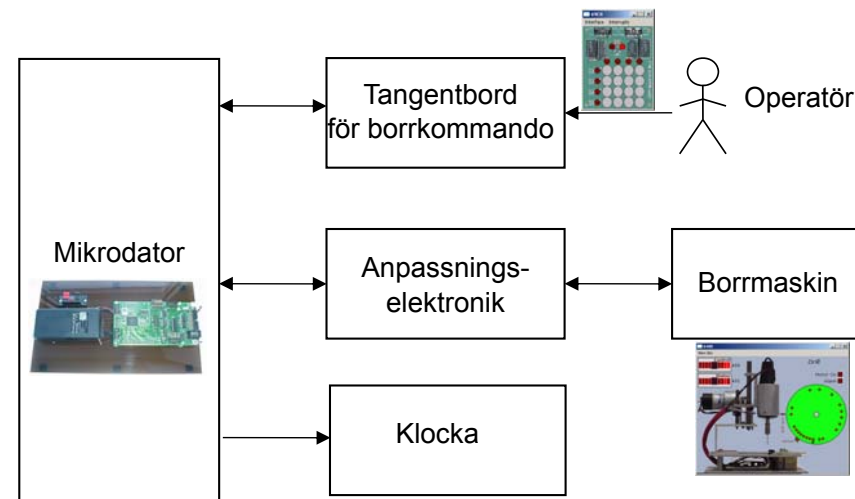


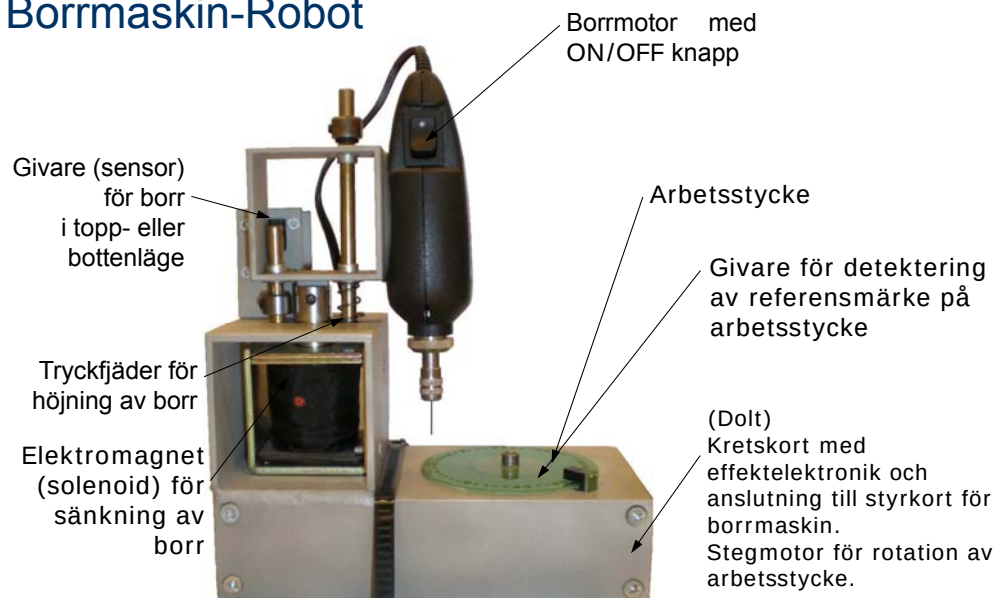
Genomgång inför laborationer

Moment 1 och 2:
Programutveckling i assembler

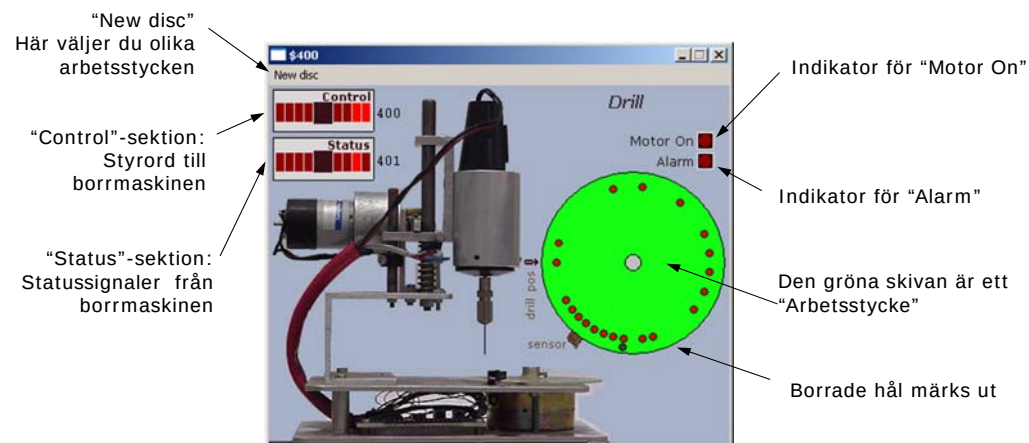
Laborationsmoment 2 - En Borrautomat



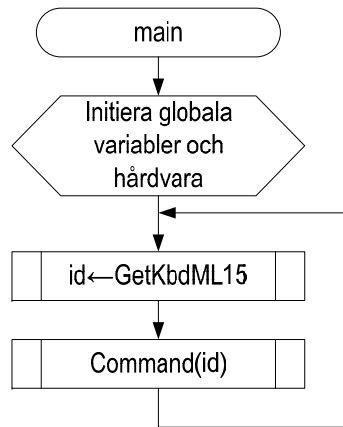
Borrmaskin-Robot



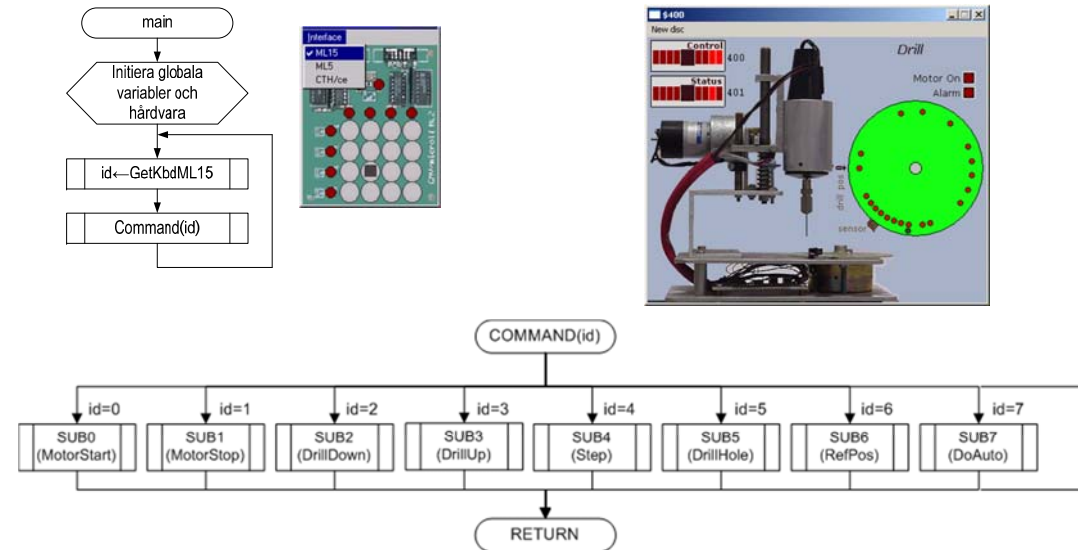
Simulatorn för borrmaskinen



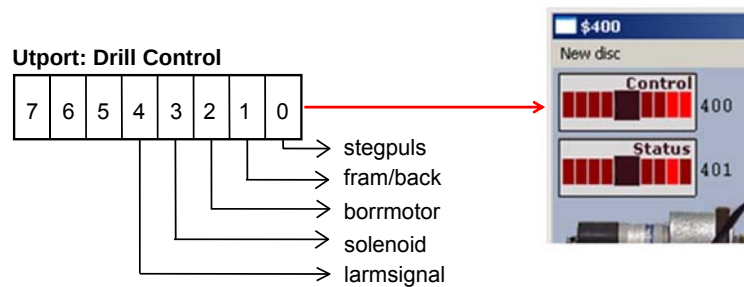
Specifikation



Tangent-kod	Operation	subrutin
0	starta bormmotorn	MotorStart
1	stoppa bormmotorn	MotorStop
2	sänk borret	DrillDown
3	höj borret	DrillUp
4	rotera arbetsstycket medurs ett steg	Step
5	borra ett hål	DrillHole
6	stega arbetsstycket till referensposition	RefPos
7	borra hål längs cirkeln enligt mönster	DoAuto



Styrdord till bormmaskinen



Bit 4 = 1: Larm på
 Bit 3 = 1: Borret sänks
 Bit 2 = 1: Bormmotorn roterar
 Bit 1 = 1: Medurs vridning
 Bit 0: Pos flank Stegpuls

Logiknivå: "Aktiv hög"

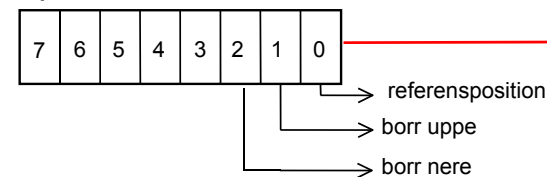
Att göra "RESET" på bormmaskinen således:

```

LDAA #0      ; Passiva signaler
STAA $400
---
```

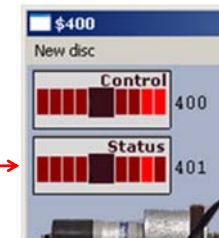
Statusord från bormmaskinen

Inport: Drill Status



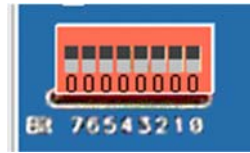
Bit 2 = 1: Borr i bottenläge
 Bit 1 = 1: Borr i toppläge
 Bit 0 = 1: Referensposition

Logiknivå: "Aktiv hög"



Anm:
 Statusporten
 ansluts till adress
 \$600 i
 laborations-
 systemet

Testförfarande

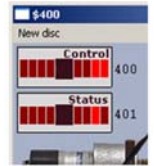


```
DipSwitch: EQU $600 ; Dip Switch Input
DrillControl: EQU $400 ; Drill Control Output
```

```
Loop LDA A DipSwitch ; Läs strömbrytare
      STAA DrillControl ; Ge styrord
      BRA Loop
```

Utför instruktionssekvensen stegvis (Step)

Villkorlig assemblering ger korrekta portadresser



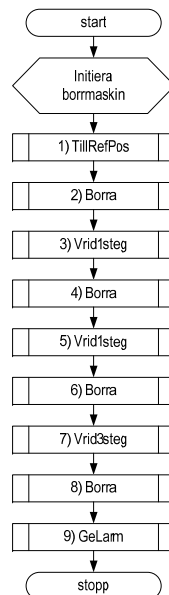
```
#ifdef SIMULATOR
DrillStatus EQU $401
#else
DrillStatus EQU $600
#endif
```

I ditt huvudprogram, innan filen Labdefs.s12 inkluderas, kan du definiera
#define SIMULATOR
när du kommer till laborationen kommenterar du bort detta på följande sätt:
; #define SIMULATOR

Anm: "Dip Switch Input" och bormaskin kan inte användas samtidigt i laborationssystemet (MC12).

Inledande uppgift med bormaskinen

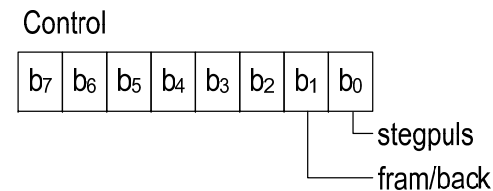
- 1) Arbetsstycket vrids till referensposition.
- 2) Håll borrar
- 3) Arbetsstycket vrids *medurs* ett steg
- 4) Håll borrar
- 5) Arbetsstycket vrids *medurs* ett steg
- 6) Håll borrar
- 7) Arbetsstycket vrids *medurs* tre steg
- 8) Håll borrar
- 9) En larmsignal ges som indikation på att uppgiften är klar.



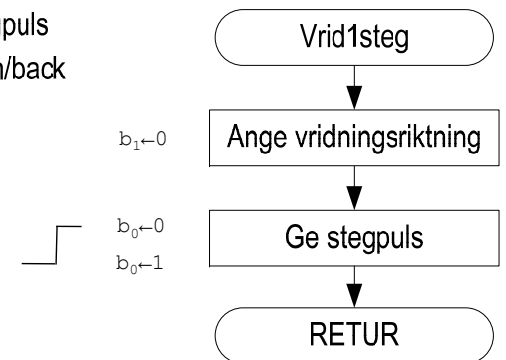
```
; Drilltest2.s12
USE Labdefs.s12
ORG $1000
start: LDA #0 ; Reset
      STAA DrillControl
      JSR TillRefPos
      JSR Borra
      JSR Vrid1steg
      JSR Borra
      JSR Vrid1steg
      JSR Borra
      JSR Vrid1steg
      JSR Borra
      JSR Vrid1steg
      JSR Borra
      JSR GeLarm
      stopp: BRA stopp

Vrid1steg: RTS
TillRefPos: RTS
Borra: RTS
GeLarm: RTS
```

Att vrida arbetsstycket

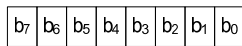


b₁=1, medurs vridning vid stegpuls
b₀, stegpuls för transition 0→1



Att borra ett hål

Control (styrregister)



borrmotor
vridmagnet

b₃, vridmagnetens funktion

b₃=1, borr sänks

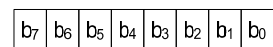
b₃=0, borr höjs

b₂, bormotorns funktion

b₂=1, borr roterar

b₂=0, borr stillastående

Status

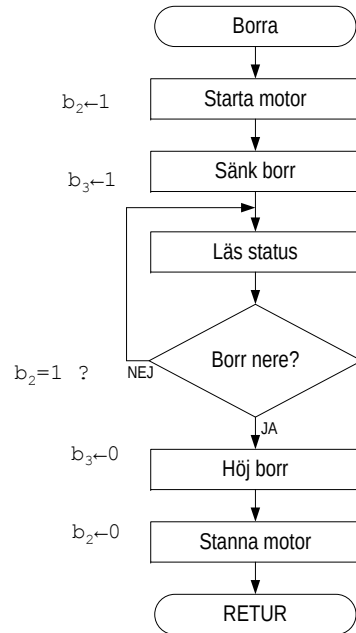


toppläge

bottenläge

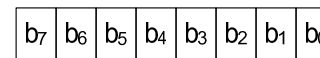
b₂=1, borr i absolut bottenläge

b₁=1, borr i absolut toppläge



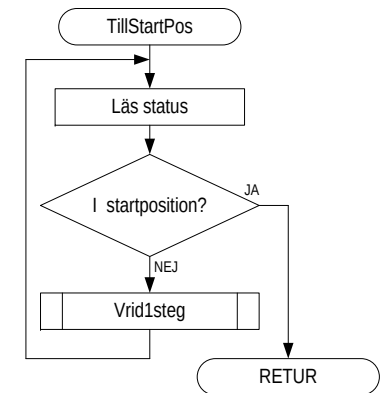
Att vrida arbetsstycket till startpositionen

Status



referensposition

b₀=1, arbetsstycke i startposition

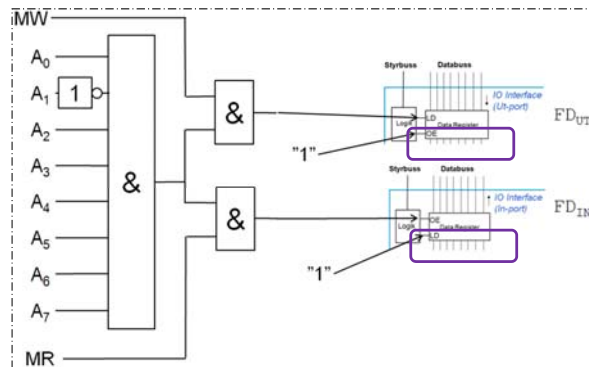


Att bara ändra en bit i taget

```

* Läs nuvarande styrord
  LDAA  DCtrl
* Nollställ lämplig bit
  ANDA  #xx
* Skriv nytt styrord
  STAA  DCtrl
;sekvensen är funktionellt
;likvärdig med:
  BCLR  #~xx, DCtrl
    
```

Fungerar inte här ty porten är "icke läsbar" utport...



Kopia av styrordet ("skuggregister")

Variabel DCSHadow ska hela tiden ha samma värde som DrillControl hade haft om porten varit läsbar...

För att nollställa en bit används nu:

```

LDAA  DCSHadow
ANDA  #Bitmönster
STAA  DCSHadow
STAA  DrillControl
    
```

för att ettställa en bit används:

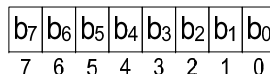
```

LDAA  DCSHadow
ORAA  #Bitmönster
STAA  DCSHadow
STAA  DrillControl
    
```

Subrutiner för att manipulera styrregistret OUTONE och OUTZERO

```
; Subrutin Outone.
; Läser kopian av bormaskinens styrord
; på adress 'DCShadow'. Ettställer en
; av bitarna och skriver det nya
; styrordet till 'DCShadow'
; samt utporten 'DrillControl'
; Biten som nollställs ges av innehållet
; i B-registret (0-7) vid anrop.
; Om (B) > 7 utförs ingenting.
; Anrop:      LDAB      #bitnummer
;            JSR        OUTONE
;            ;
; Utdata:      Inga
; Register:    Ingen
; Anropar:     Inga
```

"bitnummer" = 0..7



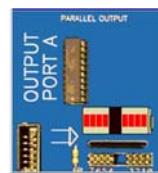
Fördröjningar i mekaniska delar

- Starta bormotorn (vänta tills den är uppe i varv, c:a 1 sekund)
- Vrid arbetsstycket ett steg (vänta tills det har vridits till rätt position, ca 250 ms)
- Lyft borret (vänta tills borret har kommit ovanför arbetsstycket, ca 250ms)
- Osv.

Anpassning mellan en mikroprocessors arbetstakt och varierande tröghet i mekaniska delar görs med tidsfördröjningar

```
;-----
; SUBROUTIN Delay
; Åstadkommer fördröjning av program.
; Fördröjningen utförs i steg om 0,25
; sekunders intervall.
; Indata:
;   Register B: Fördröjning * 0,25 sek.
; Registerpåverkan:
;   Register B innehåller alltid 0 efter
;   subrutinen. Inga andra register
;   påverkas.
```

DelayConst:	EQU	222
	ORG	\$1000
Start:	CLRA	
	LDX	#DelayConst
Main:	BSR	Delay
	COMA	
	STAA	\$400
	BRA	Main
Delay:	PSHX	; Subrutin som testas
	PSHY	
Delay1:	LEAX	-1,X
	LDY	#100
Delay2:	LEAY	-1,Y
	CPY	#0
	BNE	Delay2
	CPX	#0
	BNE	Delay1
	PULY	
	PULX	
	RTS	



Om fördröjnings-konstanten bestämts så att ljusdiodrampen tänds med en sekunds mellanrum är fördröjningen 0,5 sekunder, dvs:
2 * 250 ms.

Använd villkorlig assemblering

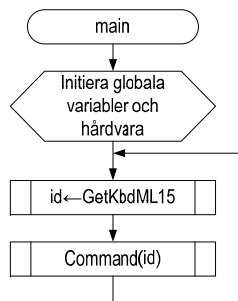
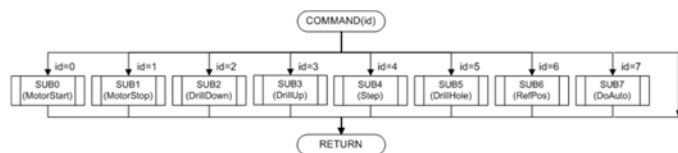
```
#ifdef SIMULATOR
#ifdef RUNFAST
    DelayConst EQU xxx
#else
    DelayConst EQU yyy
#endif
#else
    DelayConst EQU zzz
#endif
```

Dessa bestämmer du experimentellt med simulator som förberedelse.

Denna bestäms experimentellt vid laboration

Borrmaskinrobot

tangent kod	Operation	subrutin
0	starta bormotorn	MotorStart
1	stoppa bormotorn	MotorStop
2	sänk borret	DrillDown
3	höj borret	DrillUp
4	rotera arbetsstycket medurs ett steg	Step
5	borra ett hål	DrillHole
6	stega arbetsstycket till referensposition	RefPos
7	borra hål längs cirkeln enligt mönster	DoAuto



```

;-----
; SUBROUTIN - Command
; Beskrivning: Rutinen avgör vilken
; kommandosubrutin som skall
; utföras och anropar denna.
; Anrop: JSR Command
; Indata: Kommandonummer i reg B
; Utdata: Inga
; Register: B,X ändras
; Anrop: SUB0 ... SUB7
;-----

; Tabell med subrutinadresser (pekare)
JUMPTAB: FDB SUB0,SUB1,SUB2,SUB3
          FDB SUB4,SUB5,SUB6,SUB7
;-----
; subrutiner för test

SUB0 MOVB #0,OutPort
RTS
SUB1 MOVB #1,OutPort
RTS
SUB2 MOVB #2,OutPort
RTS
SUB3 MOVB #3,OutPort
RTS
SUB4 MOVB #4,OutPort
RTS
SUB5 MOVB #5,OutPort
RTS
SUB6 MOVB #6,OutPort
RTS
SUB7 MOVB #7,OutPort
RTS

Command:
; giltigt värde?
CMPB #7
BHI CommandExit
; pekartabellens basadress
LDX #JUMPTAB
; offset är 2 bytes per adress
ASLB
; hämta subrutinens startadress
LDX B,X
; utför subrutin
JSR ,X
; återvänd från kommandorutin
CommandExit:
RTS

```

Filen Main.s12

STRUKTUR

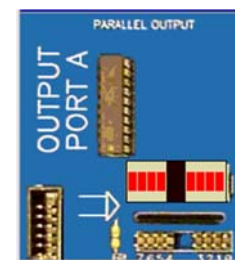
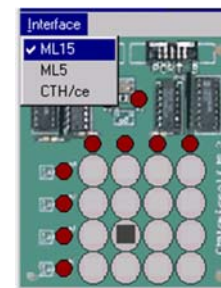
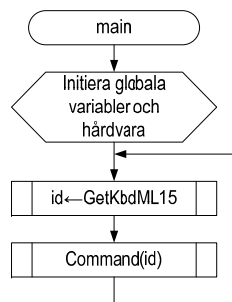
1. Inkludera definitionsfil
2. Initiera systemet
3. Huvudprogram
4. Subrutinen **Command**
5. Inkludera fil (filer) med ytterligare subrutiner.
6. Plats för variabler

```

; Definitioner
USE Labdefs.s12
ORG $1000
main:
;-----
; Huvudprogram
; Invänta vald operation
main_loop: JSR GetKbdML15
; Utför vald operation
JSR Command
BRA main_loop
;-----
Command: ---
          ---
          ---
USE ML15drv.S12
USE Subroutines.s12
; Placera alla globala variabler här
DCShadow RMB 1
          ...

```

Att testa programmet i filen Main.s12

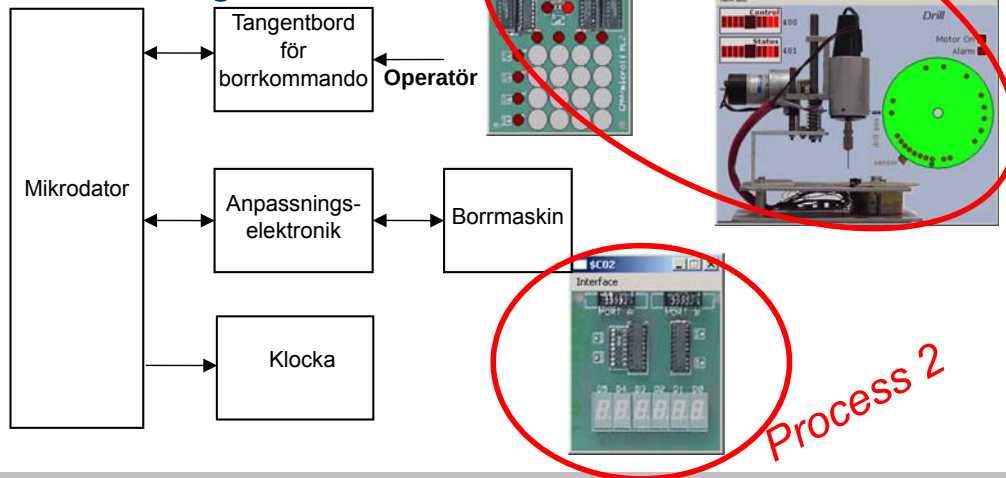


```

SUB0 MOVB #0,ParOut
RTS
SUB1 MOVB #1,ParOut
RTS
SUB2 MOVB #2,ParOut
RTS
SUB3 MOVB #3,ParOut
RTS
SUB4 MOVB #4,ParOut
RTS
SUB5 MOVB #5,ParOut
RTS
SUB6 MOVB #6,ParOut
RTS
SUB7 MOVB #7,ParOut
RTS

```


Laborationsmoment 2 Pseudoparallell exekvering



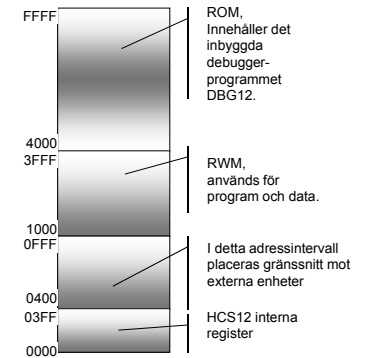
Genomgång inför laborationer 1 och 2 / RoJ

25

Applikation för avbrott (IRQ)

```
;Initieringssekvens
ORG      XXXX
...
; nollställ I-flagga
ANDCC    #$FE
JSR      _main
...
```

```
ORG      $FFF2
FDB      irq_service_routine
; Avbrottshanterare
irq_service_routine:
...
RTI
```

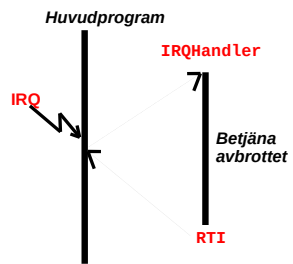


I laborationssystemet (MC12) kan vi INTE placera avbrottsvektorer på deras rätta platser (konflikt med DBG12)
I stället placeras dom i RWM

Genomgång inför laborationer 1 och 2 / RoJ

26

MC12 (DBG12) och avbrott



Vektor ROM	Funktion
FFFE	RESET, Startvektor
FFFC	Clock Monitor Fail, JMP [3FFC]
FFFA	COP Watchdog Timeout, JMP [3FFA]
FFF8	Illegal Op Code, JMP [3FF8]
FFF6	SWI, JMP [3FF6]
FFF4	XIRQ, JMP [3FF4]
FFF2	IRQ, JMP [3FF2]
FFFC	Enhetsspecifika vektorer JMP [3Fxx]
FFF0	

Allmänt

```
ORG      $FFF2
FDB      irq_service_routine
; Avbrottshanterare
irq_service_routine:
RTI
```

Men i MC12 och simulator...

```
ORG      $3FF2
FDB      irq_service_routine
; Avbrottshanterare
irq_service_routine:
RTI
```

Genomgång inför laborationer 1 och 2 / RoJ

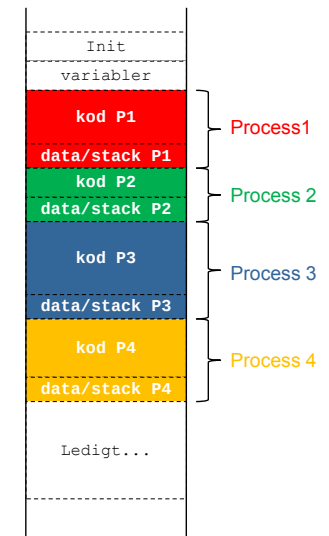
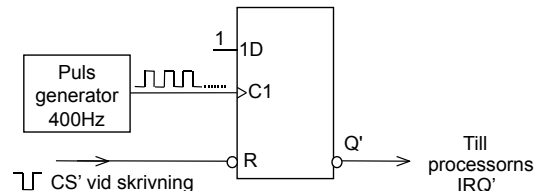
27

Processbyte

En processor
– flera program
– körs "samtidigt" (pseudoparallellt)

HDW krav: En avbrottskälla som ger regelbundna avbrott (Ex Timer)

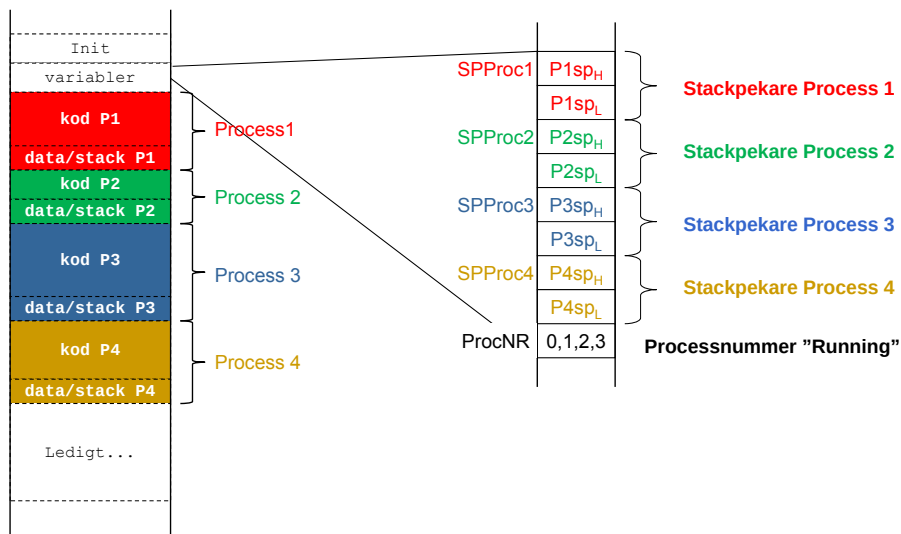
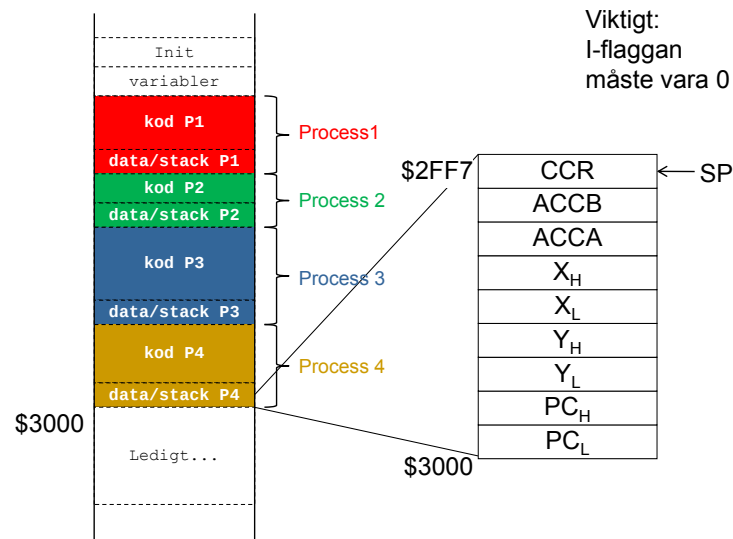
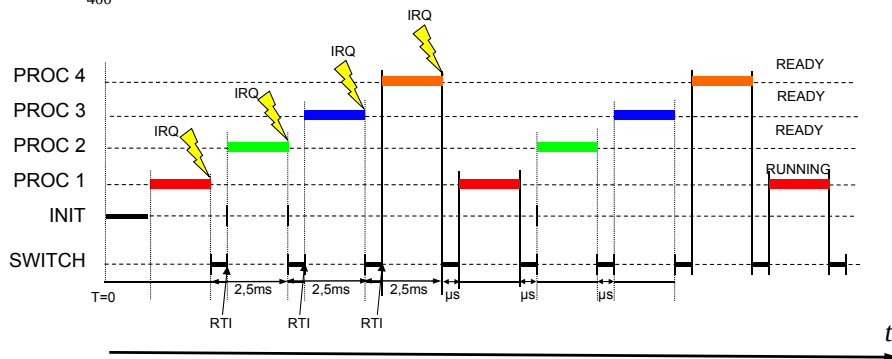
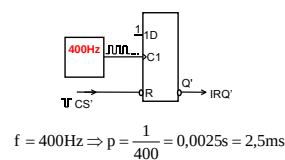
SW krav: En avbrottsrutin (SWITCH) som växlar process



Genomgång inför laborationer 1 och 2 / RoJ

28

Processtillstånd

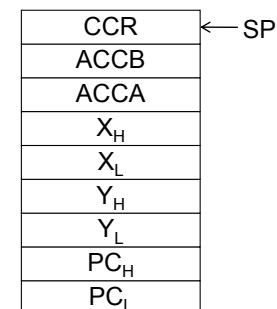


Initial stack för process och "processbyte":

```

ORG      TopOfStack-9
Istack: FCB $C0      ; Initialt CCR
          FCB 0       ; Initialt B
          FCB 0       ; Initialt A
          FDB 0       ; Initialt X
          FDB 0       ; Initialt Y
          FDB Start   ; Initialt PC

ORG      Code
LDD      #Istack
STD      SPProc
    
```



```

IRQHandler:
; Spara "Running" stackpekare
STS      ...
; Välj ny "Running"
LDS      ...
; Återstarta
RTI
    
```