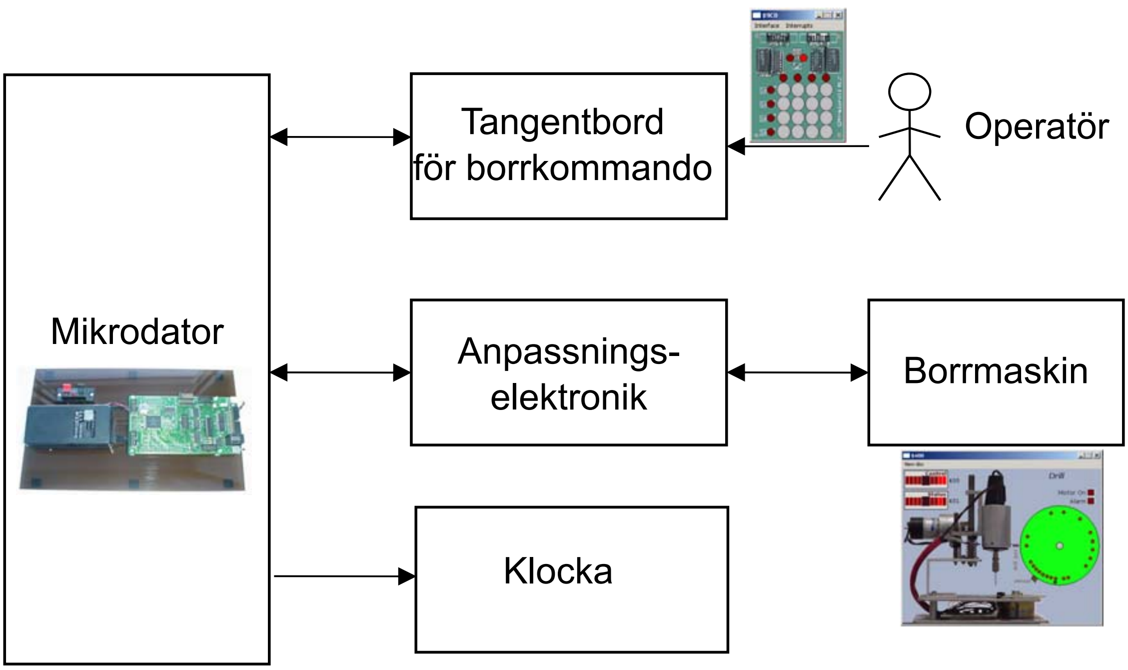
Maskinorienterad Programmering 2012/2013

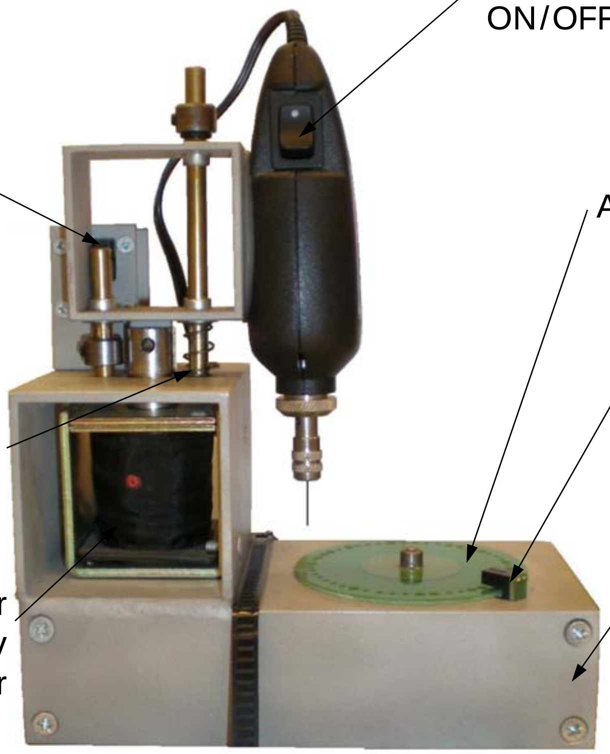
Genomgång av laborationer:
"Programutveckling i assembler"

Arbetsbok för MC12, kapitel 4

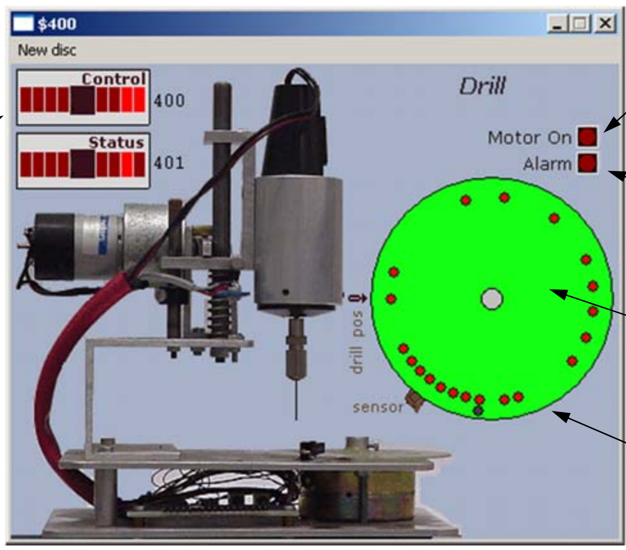
Laborationsmoment 2 - En Borrautomat



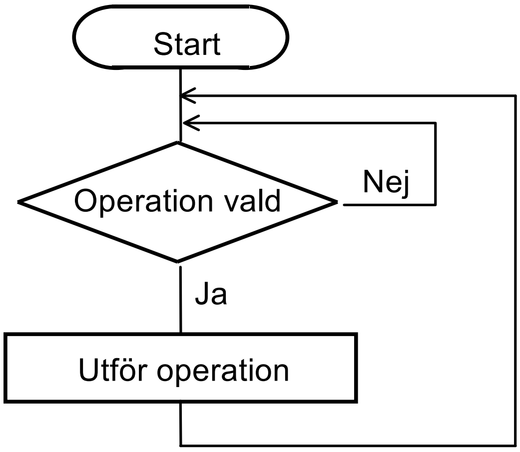
Borrmaskin-Robot



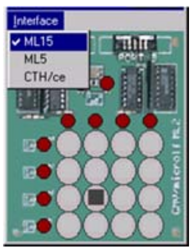
Simulatorn för borrmaskinen

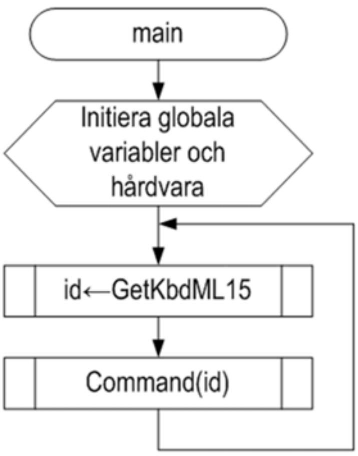


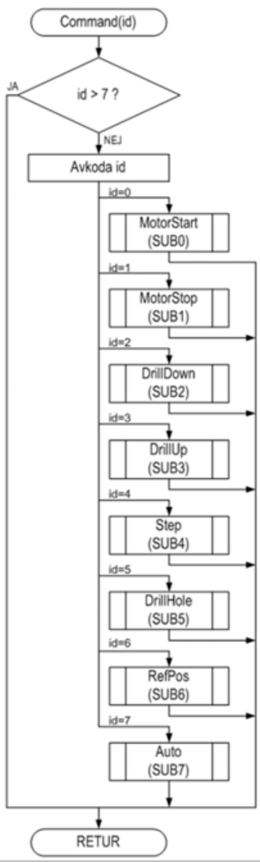
Specifikation

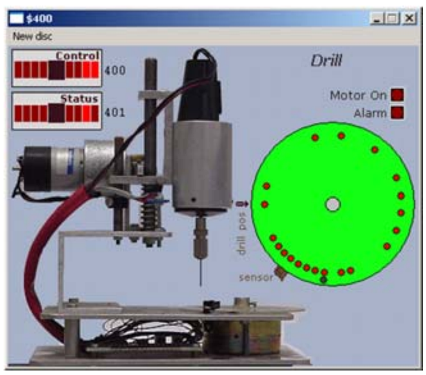


- starta borrmotorn
- stoppa borrmotorn
- sänk borret
- höj borret
- vrid arbetsstycket ett steg
- vrid arbetsstycket till referenspositionen
- borra ett hål
- borra hål längs cirkeln enligt ett bestämt mönster.



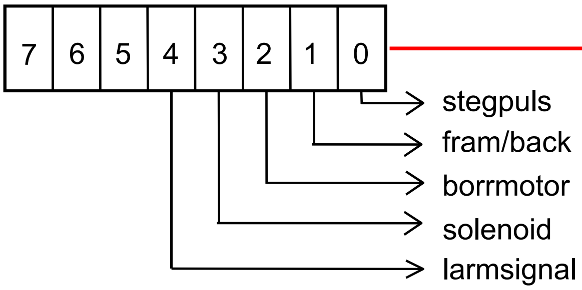


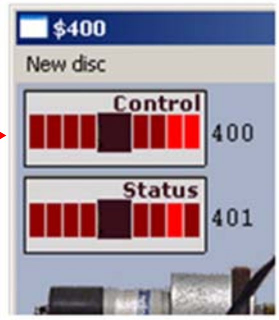




Styrord till borrmaskinen

Utport: Drill Control





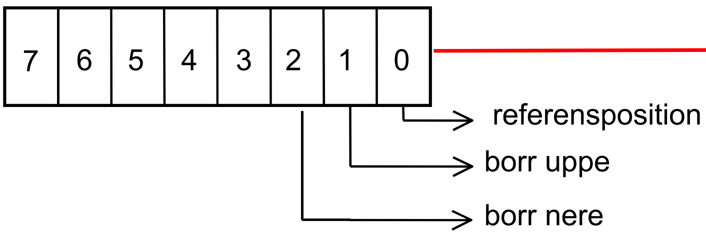
Bit 4 = 1: Larm på
Bit 3 = 1: Borret sänks
Bit 2 = 1: Borrmotorn roterar
Bit 1 = 1: Medurs vridning
Bit 0: Pos flank Stegpuls

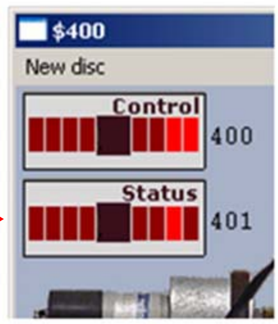
Logiknivå: ”Aktiv hög”
Att göra ”RESET” på borrmaskinen således:

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


Statusord från borrmaskinen

Inport: Drill Status





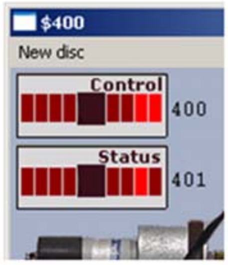
Anm: Statusporten ansluts till adress $600 i laborations-systemet

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

Logiknivå: ”Aktiv hög”

Testförfarande

Uppgift 71




```
DipSwitch     EQU    $600
HexDisplay    EQU    $700
DrillControl  EQU    $0400
DrillStatus   EQU    $0401

Loop          LDAA   DipSwitch           ; Läs strömbrytare
              STAA   DrillControl        ; Ge styrord
              LDAB   DrillStatus         ; Läs status
              BRA    Loop
```


Villkorlig assemblering ger korrekta portadresser

```
; Definiera macro 'SIM' för test i simulator
#define       SIM

DipSwitch     EQU    $600           ; Dip Switch Input
DrillControl  EQU    $0400          ; Drill Control Output
#ifdef SIM
DrillStatus   EQU    $0401          ; Drill Status Input
#else
DrillStatus   EQU    $0600          ; Drill Status Input
#endif
```

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

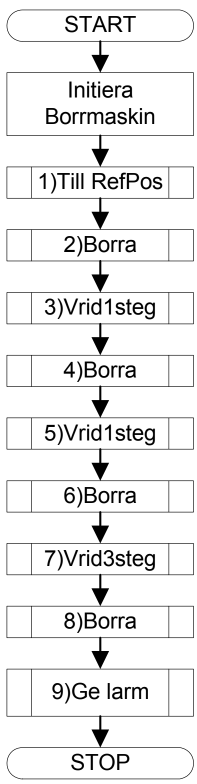
Använd USE-direktivet

```
;       Labdefs.S12
DipSwitch       EQU     $600    ; Dip Switch Input
DrillControl    EQU     $0400   ; Drill Control Output
# ifdef         SIM
DrillStatus     EQU     $0401           ; Drill Status Input
# else
DrillStatus     EQU     $0600           ; Drill Status Input
# endif
```

```
; Drilltest1.s12
#define         SIM
                USE     "DRILLDEFS.S12"
Loop            LDAA    DipSwitch           ; Läs strömbrytare
                STAA    DrillControl        ; Ge styrord
                LDAB    DrillStatus         ; Läs status
                BRA     Loop
```


Inledande uppgift med borrmaskinen

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

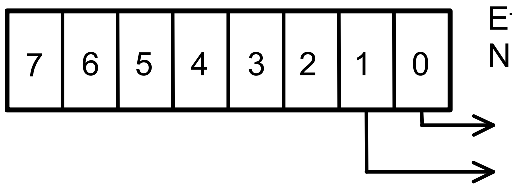



```
; Dtest2
    USE
    DRILLDEFS.S12
    ORG  $1000
    LDAA #0     ; Reset
    STAA DCtrl
    JSR  TillRefPos
    JSR  Borra
    JSR  Vrid1steg
    JSR  Borra
    JSR  Vrid1steg
    JSR  Borra
    JSR  Vrid1steg
    JSR  Vrid1steg
    JSR  Vrid1steg
    JSR  Borra
    JSR  GeLarm
Loop:       BRA   Loop
Vrid1steg:        RTS
TillRefPos:       RTS
Borra:            RTS
GeLarm:           RTS
```

Att vrida arbetsstycket

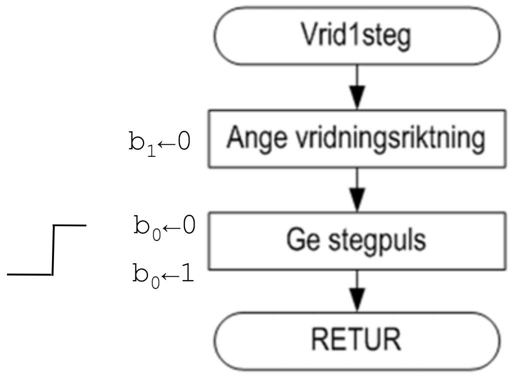
Uppgift 77

Utport: Drill Control



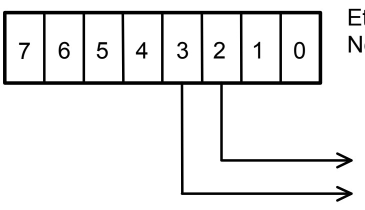
Etta: Aktiv styrsignal
Nolla: Passiv styrsignal

Bit 0 = 0→1 (Pos puls): Arbetsstycket vrids
Bit 1 = 0: Medurs vridningsriktning
Bit 1 = 1: Moturs vridningsriktning



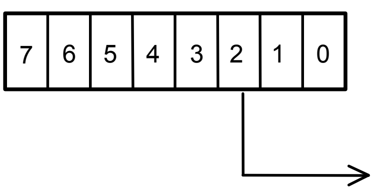
Att borra ett hål

Utport: Drill Control

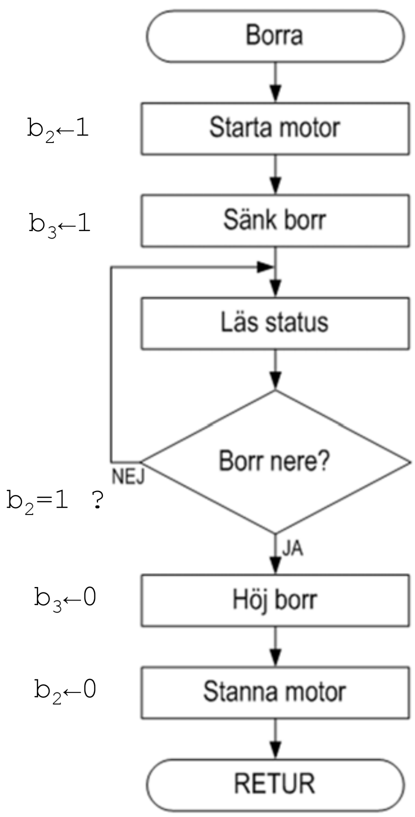


Etta: Aktiv styrsignal
Nolla: Passiv styrsignal

Inport: Drill Status



Nolla: Passiv statussignal
Etta: Aktiv statussignal



Att vrida arbetsstycket till referenspositionen

Inport: Drill Status

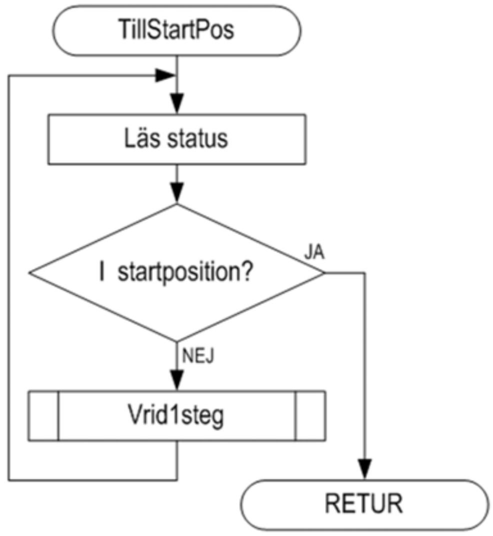
7	6	5	4	3	2	1	0

Nolla: Passiv statussignal
Etta: Aktiv statussignal

referensposition

Bit 0 = 0: Arbetsstycket är inte i referensposition
Bit 0 = 1: Arbetsstycket är i referensposition

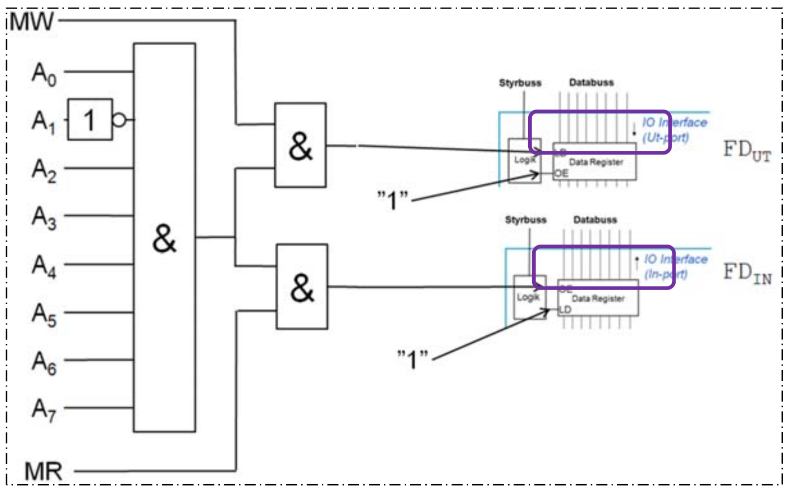
$b_0=1$?



Att bara ändra en bit i taget

```
* Läs nuvarande styrord
        LDAA    DrillControl
* Nollställ lämplig bit
        ANDA    #xx
* Skriv nytt styrord
        STAA    DrillControl

;sekvensen är funktionellt
;likvärdig med:
        BCLR    #~DrillControl
```



Fungerar inte här ty porten är ”icke läsbar” utport…

Kopia av styrordet

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        #~xx
STAA        DCShadow
STAA        DrillControl
```

för att ettställa en bit används:

```
LDAA        DCShadow
ORAA        #xx
STAA        DCShadow
STAA        DrillControl
```

```
DCShadow    RMB    1
```


Subrutiner för att manipulera styrregistret Outone och Outzero

```
* Subrutin Outone. Läser kopian av
* borrmaskinens styrord på adress
* DCShadow. Ettställer en av bitarna och
* skriver det nya styrordet till
* utporten DrillControl samt tillbaka till
* kopian DCShadow.
* 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
* Registerpåverkan:      Ingen
* Anropade subrutiner:   Inga
```

”bitnummer” = 0..7

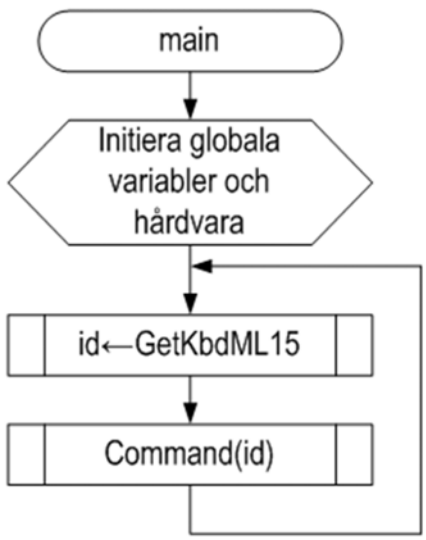
7	6	5	4	3	2	1	0

Fördröjningar i mekaniska delar

- Starta borrmotorn
  (vänta tills den är uppe i varv,
  c:a 1 sekund)
- Vrid arbetsstycket ett steg
  (vänta tills det har vridits till rätt position,
  ca 200 ms)
- Lyft borret
  (vänta tills borret har kommit ovanför arbetsstycket,
  ca 300ms)
- etc

Borrmaskinrobot

tangent kod	Operation	subrutin
0	starta borrmotorn	MotorStart
1	stoppa borrmotorn	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



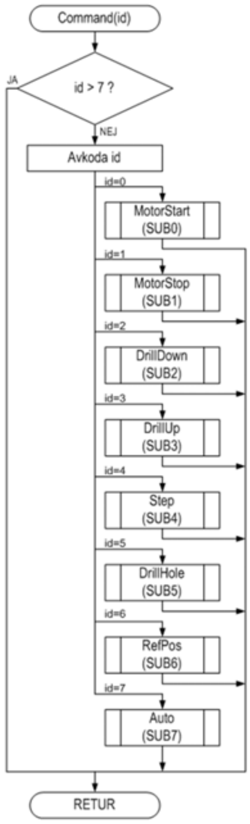
Rutinen Command

Uppgift 88

```
Command:
* giltigt värde?
        CMPB        #7
        BHI         CommandExit
* hopptabellens 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
```

```
***********************************
*     Tabell med subrutinadresser
JUMPTAB    FDB     SUB0,SUB1,SUB2,SUB3
           FDB     SUB4,SUB5,SUB6,SUB7
***********************************
* subrutiner för test, byts senare
* ut mot START, STOP, DOWN etc
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
```



Filen Main.s12

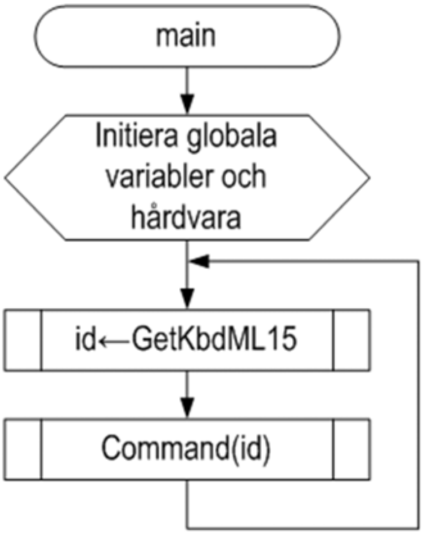
STRUKTUR

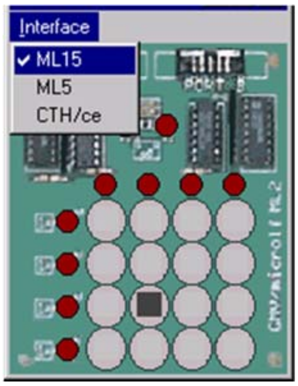
1. Inkludera definitionsfil
2. Initiera borrmaskin
3. Huvudprogram
4. Subrutinen COMMAND
5. Inkludera fil (filer) med ytterligare subrutiner.
6. Plats för variabler

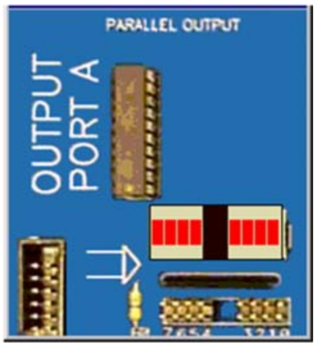
```
; Definitioner
            USE         Labdefs.s12
            ORG         $1000
            ---
            ---
;**********************************
; Huvudprogram
; Invänta vald operation
main_loop:
            JSR         GetKbdML15

; Utför vald operation
            JSR         Command
            BRA         main_loop
**********************************
Command:    ---
            ---
            ---
            USE         ML15drvr.s12
; Placera alla variabler här
DCShadow    RMB         1
```

Att testa 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
```


Rutiner MotorStart och MotorStop

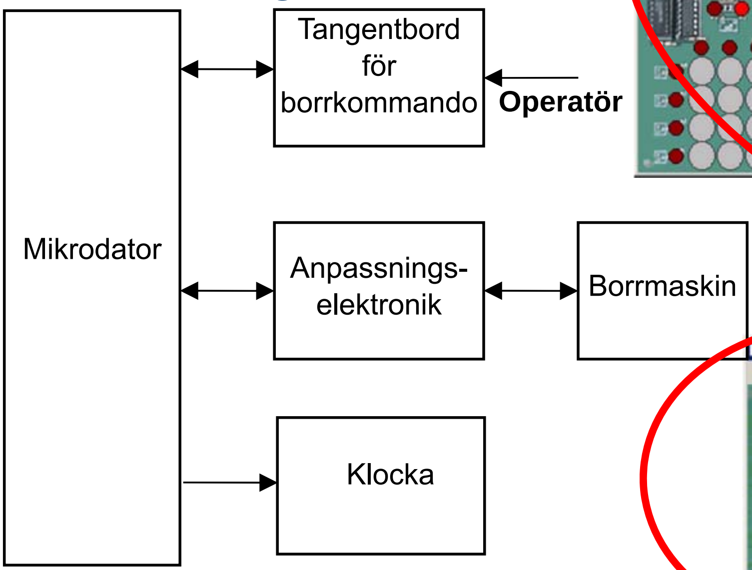
```
; SUBRUTIN MotorStart.
; Subrutinen startar
; borrmotorn väntar därefter i 1 sekund
; före återhopp så att borret uppnår
; rätt hastighet.
*
* Anrop:                  JSR     MotorStart
*
* Indata:                 Inga
* Utdata:                 Inga
* Registerpåverkan:       Ingen
* Anropade subrutiner:    Outone
*                         DELAY
```

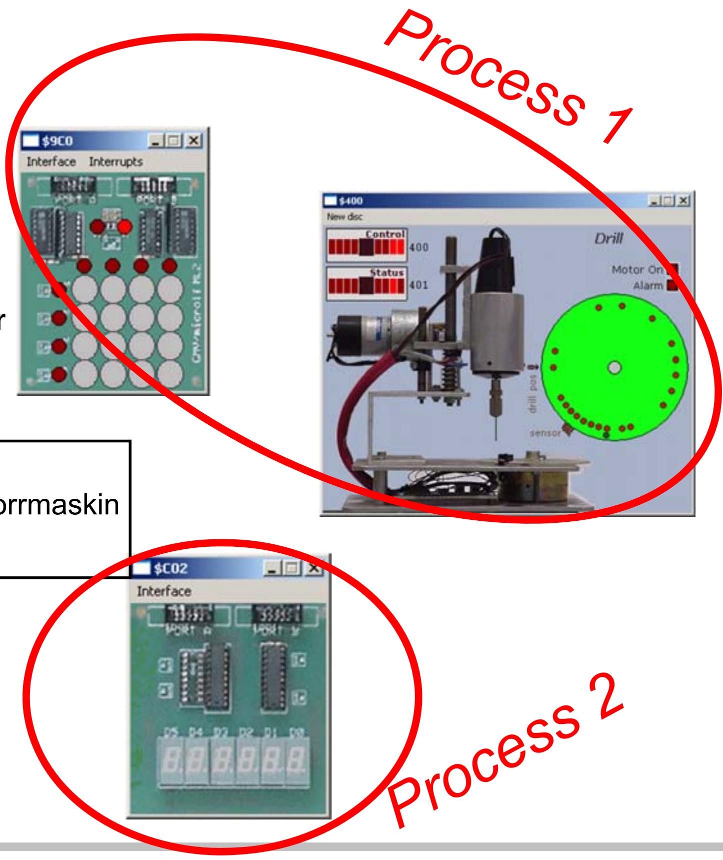

```
* SUBRUTIN STOP. Subrutinen stannar
* borrmotorn.
*
* Anrop:                  JSR     MotorStop
* Indata:                 Inga
* Utdata:                 Inga
* Registerpåverkan:       Ingen
* Anropade subrutiner:    Outone
```

Ändringar i huvudprogram...

```
JUMPTAB    FDB     MotorStart,MotorStop
           FDB     SUB2,SUB3
           FDB     SUB4,SUB5,SUB6,SUB7
*******************************
* subrutiner för test, byts senare
* ut mot START, STOP, DOWN etc
MotorStart .....
     RTS
MotorStop  .....
     RTS
SUB2 MOVB #2,ParOut
     RTS
     .....
```

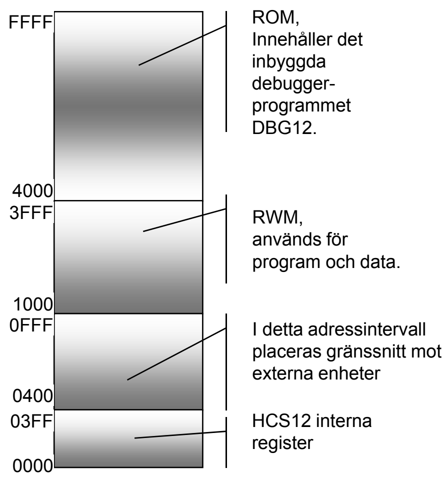
Laborationsmoment 2 Pseudoparallell exekvering





Applikation för avbrott (IRQ)

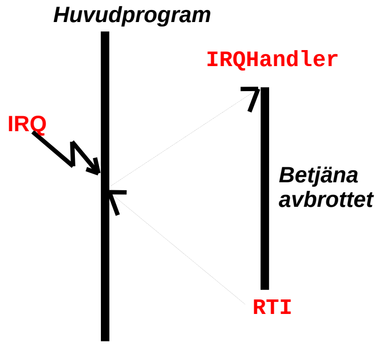
```
;Initieringssekvens
       ORG     XXXX
       ...
; nollställ I-flagga
       ANDCC   #$FE
       JSR     _main
       ...
```




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

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

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]
FF8C FFF0	Enhetsspecifika vektorer JMP [3Fxx]

Allmänt

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

Men i MC12 och simulator...

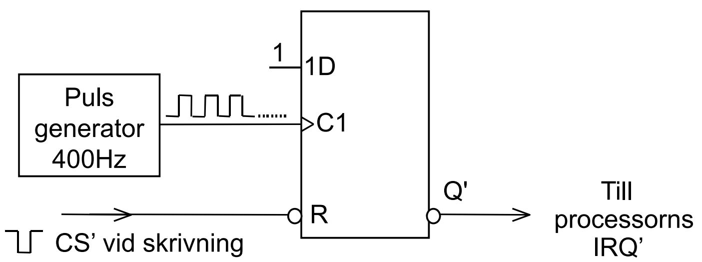
```
        ORG     $3FF2
        FDB     irq_service_routine
; Avbrottshanterare
irq_service_routine:
        RTI
```

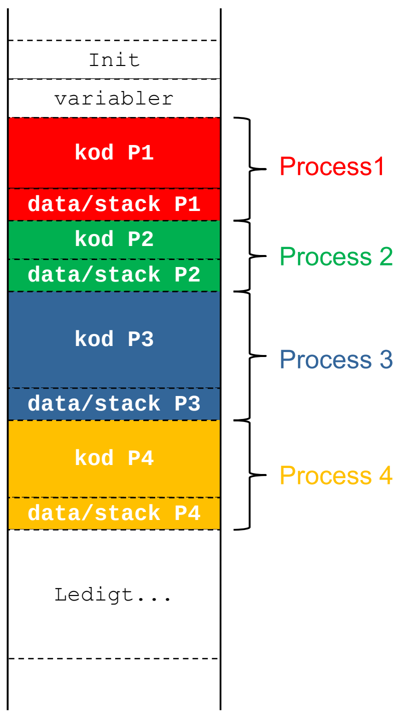

Processbyte

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

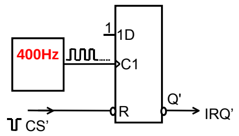
HDW krav: En avbrottskälla som ger regelbundna avbrott (Ex Timer)

SW krav: En avbrottsrutin (SWITCH) som växlar process

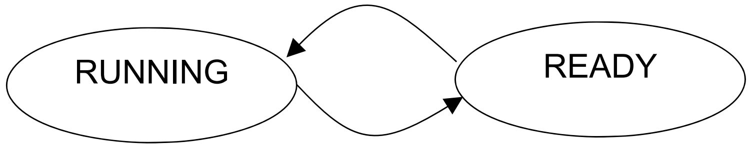


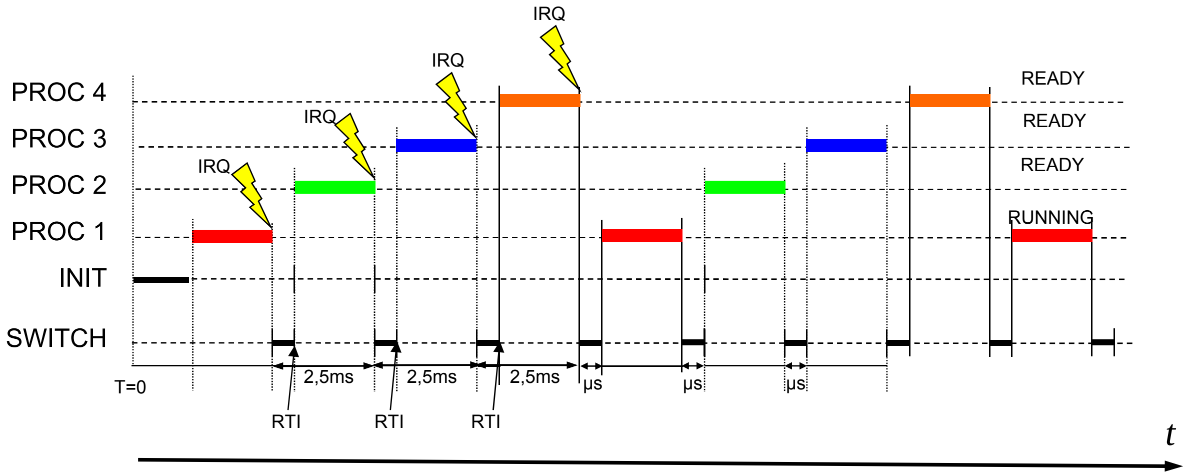


Processtillstånd

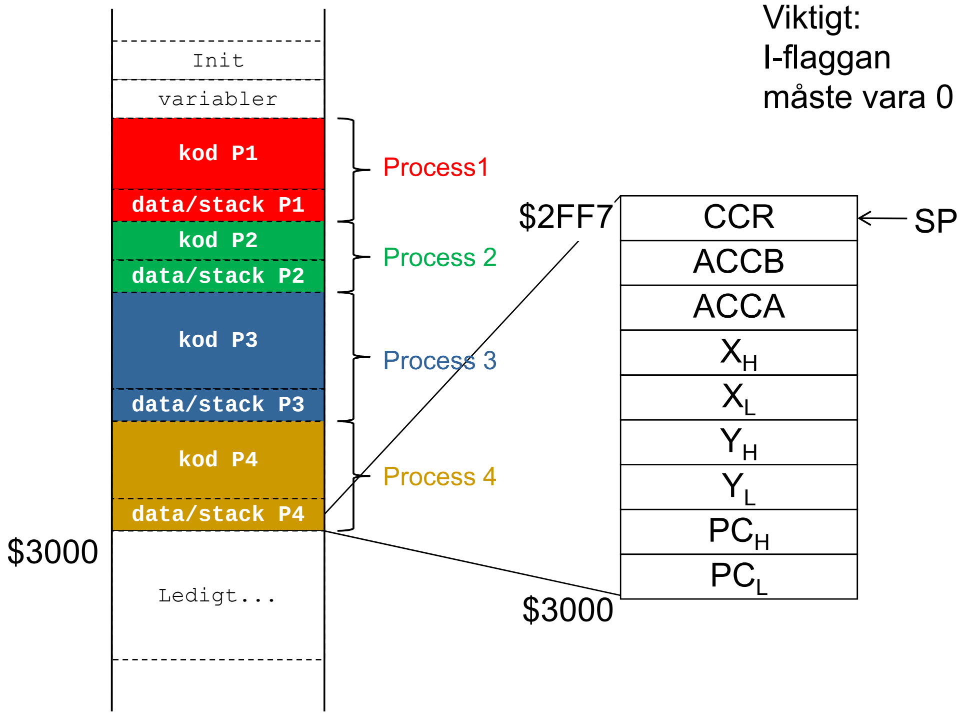


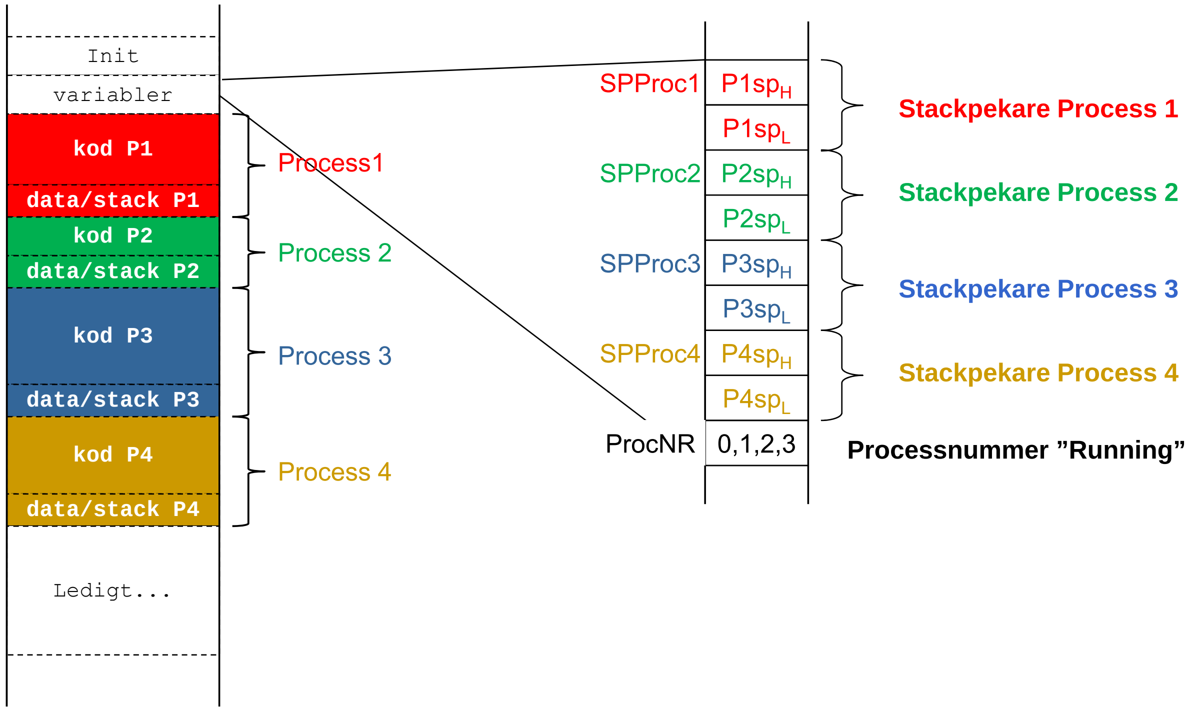
$$f = 400\text{Hz} \Rightarrow p = \frac{1}{400} = 0{,}0025\text{s} = 2{,}5\text{ms}$$





Viktigt:
I-flaggan
måste vara 0



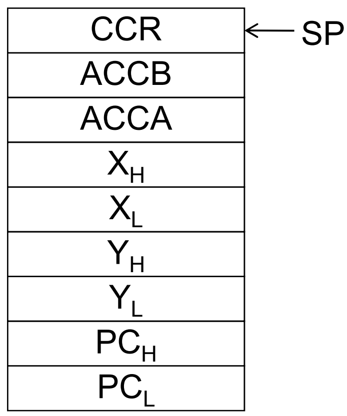


CHALMERS
Chalmers University of Technology
Maskinorienterad Programmering – DAT015, 2012/2013

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
```