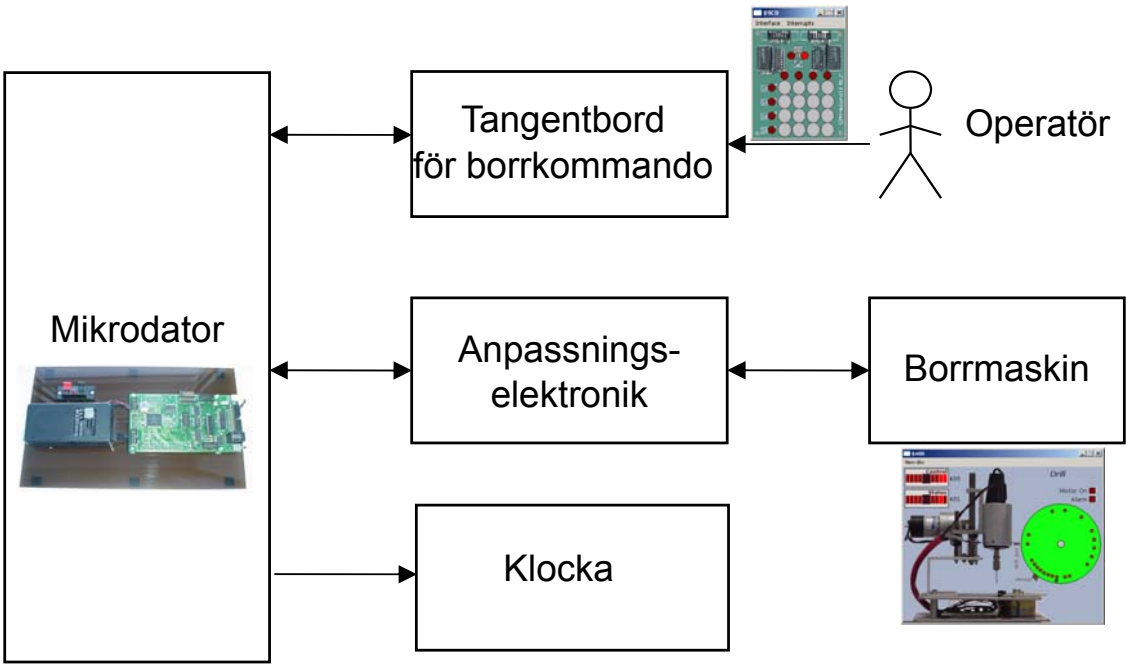
Maskinorienterad Programmering 2011/2012

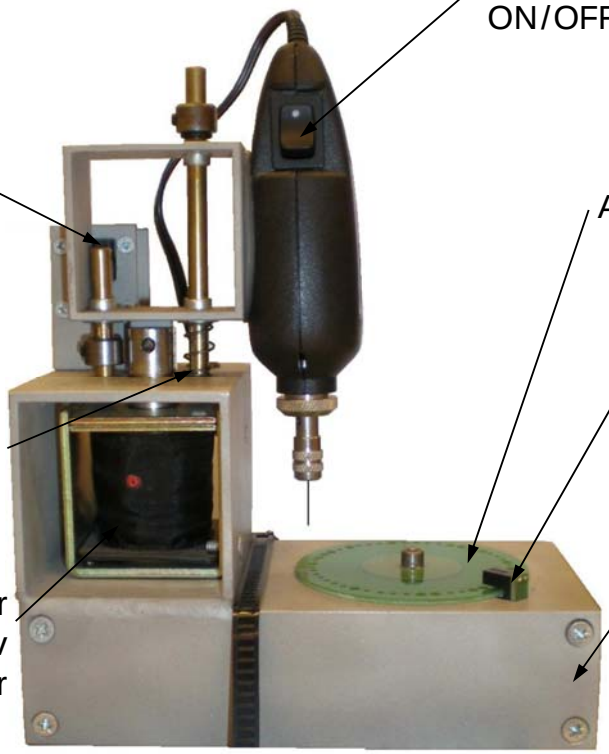
Genomgång av laborationer: ”Programutveckling i assembler”

Arbetsbok för MC12, kapitel 4

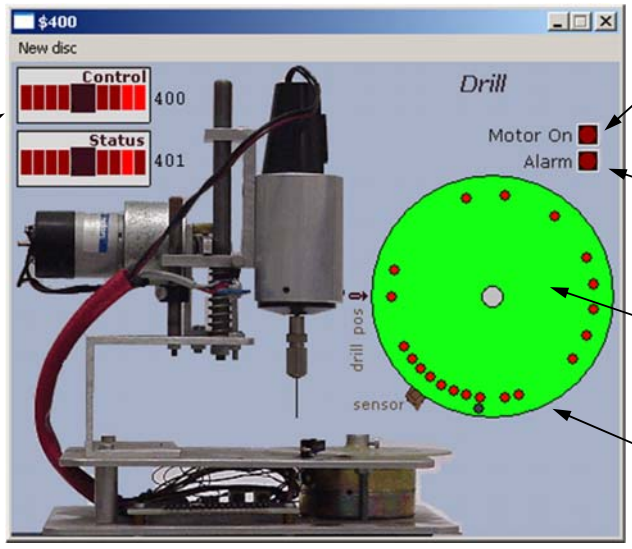
Laborationsmoment 2 - En Borrautomat



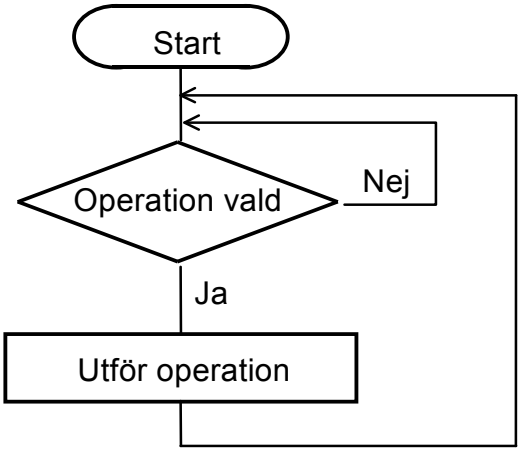
Borrmaskin-Robot



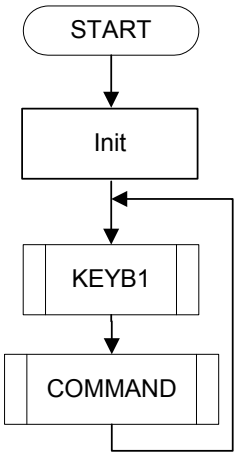
Simulatorn för borrmaskinen

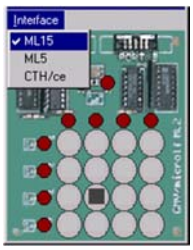


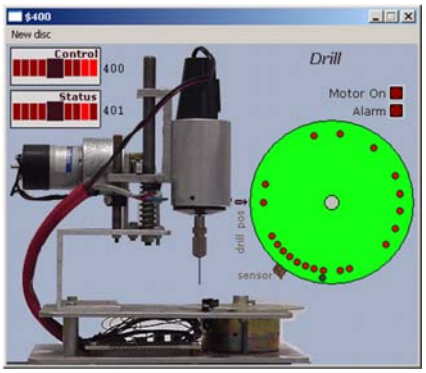
Specifikation



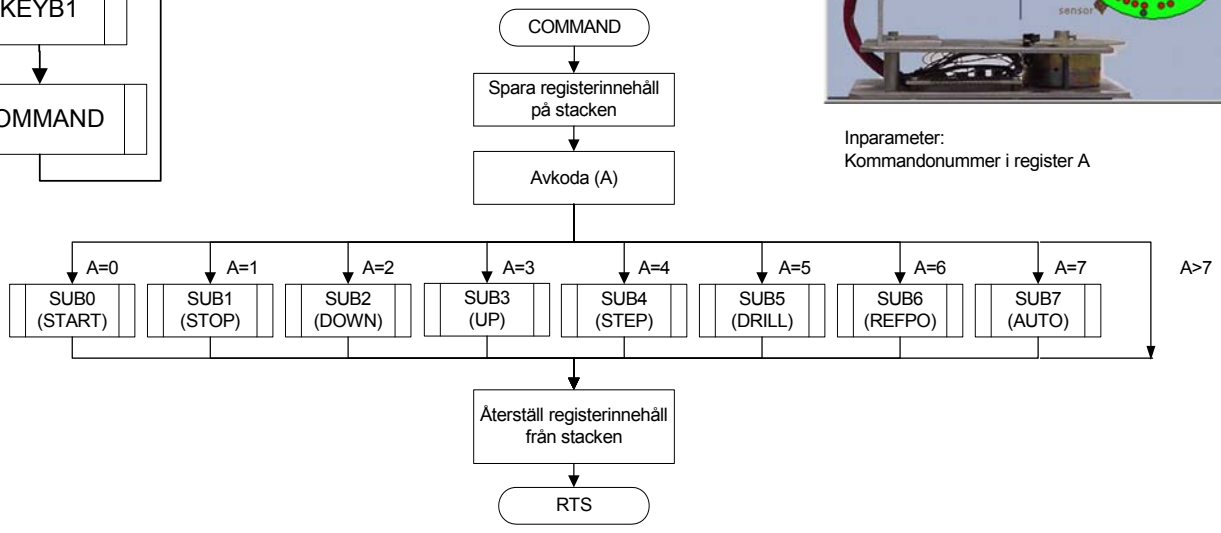
- starta borrmotorn
- stoppa borrmotorn
- sänk borret
- höj borret
- vrid arbetsstycket ett steg
- vrid arbetsstycket till referenspositionen
- borra ett hål
- borra hål längs cirkeln enligt ett bestämt mönster.





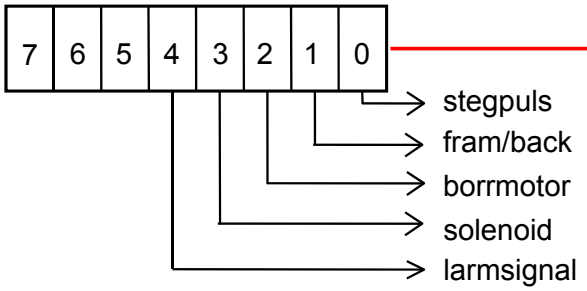


Inparameter:
Kommandonummer i register A



Styrord till borrmaskinen

Utport: Drill Control



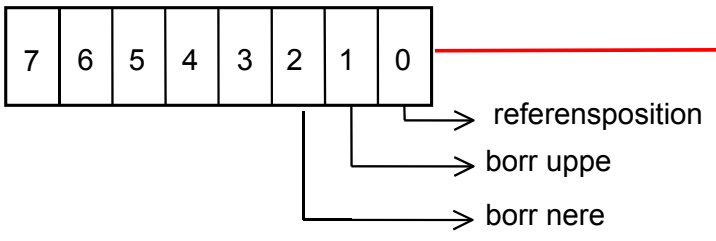
Bit 4 = 1: Larm på
Bit 3 = 1: Borret sänks
Bit 2 = 1: Borrmotorn roterar
Bit 1 = 1: Medurs vridning
Bit 0: Pos flank Stegpuls

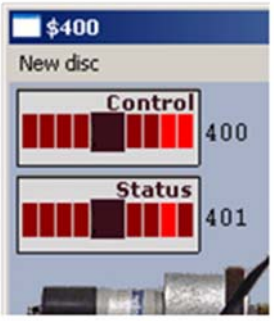
Logiknivå: ”Aktiv hög”
Att göra ”RESET” på borrmaskinen således:

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


Statusord från borrmaskinen

Inport: Drill Status





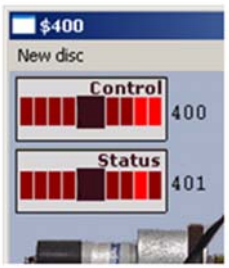
Anm: ***Statusporten ansluts till adress $600 i laborations-systemet***

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

Logiknivå: ”Aktiv hög”

Testförfarande

Uppgift 74




```
DSInput     EQU   $600        ; Dip Switch Input
DCtrl       EQU   $0400       ; Drill Control Output
DStatus     EQU   $0401       ; Drill Status Input

Loop        LDAA  DSInput     ; Läs strömbrytare
            STAA  DCtrl       ; Ge styrord
            LDAB  DStatus     ; Läs status
            BRA   Loop
```


Villkorlig assemblering ger korrekta portadresser

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

DSInput     EQU   $600        ; Dip Switch Input
DCtrl       EQU   $0400       ; Drill Control Output
#ifdef SIM
DStatus     EQU   $0401       ; Drill Status Input
#else
DStatus     EQU   $0600       ; Drill Status Input
#endif
```

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

Använd USE-direktivet

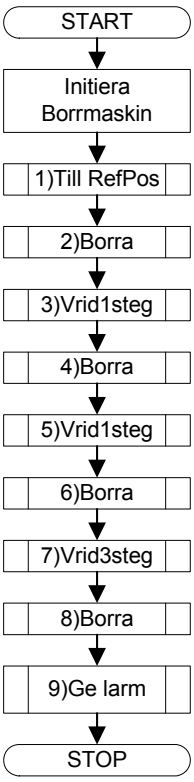
```
;       DRILLDEFS.S12
DSInput         EQU     $600    ; Dip Switch Input
DCtrl           EQU     $0400   ; Drill Control Output
# ifdef         SIM
DStatus EQU     $0401           ; Drill Status Input
# else
DStatus EQU     $0600           ; Drill Status Input
# endif
```

```
; DTEST1.s12
#define       SIM
              USE     ”DRILLDEFS.S12”
Loop          LDAA    DSInput       ; Läs strömbrytare
              STAA    DCtrl         ; Ge styrord
              LDAB    DStatus       ; Läs status
              BRA     Loop
```

Anm: Både DRILLDEFS och IODEFS förekommer som namn här i arbetsboken, använd endast DRILLDEFS.S12

Inledande uppgift med borrmaskinen

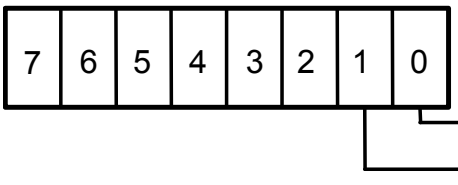
1) Arbetsstycket vrids till referensposition.
2) Hål borras
3) Arbetsstycket vrids *medurs* ett steg
4) Hål borras
5) Arbetsstycket vrids *medurs* ett steg
6) Hål borras
7) Arbetsstycket vrids *medurs* tre steg
8) Hål borras
9) En larmsignal ges som indikation på att uppgiften är klar.




```
; Dtest2
    USE
    DRILLDEFS.S12
    ORG $1000
    LDAA#0    ; Reset
    STAADCtrl
    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

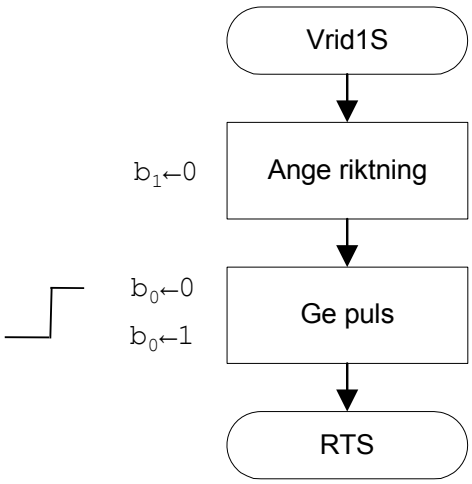
Utport: Drill Control



Etta: Aktiv styrsignal
Nolla: Passiv styrsignal

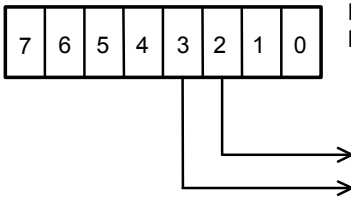
Bit 0 = 0→1 (Pos puls): Arbetsstycket vrids
Bit 1 = 0: Medurs vridningsriktning
Bit 1 = 1: Moturs vridningsriktning

Uppgift 80



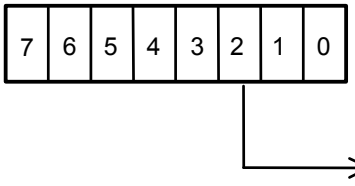
Att borra ett hål

Utport: Drill Control

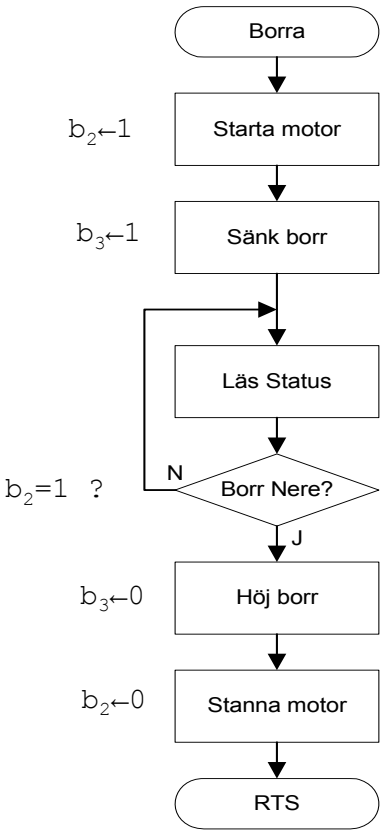


Etta: Aktiv styrsignal
Nolla: Passiv styrsignal

Inport: Drill Status

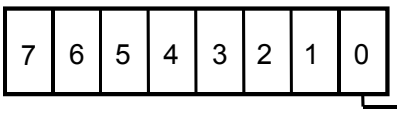


Nolla: Passiv statussignal
Etta: Aktiv statussignal



Att vrida arbetsstycket till referenspositionen

Inport: Drill Status

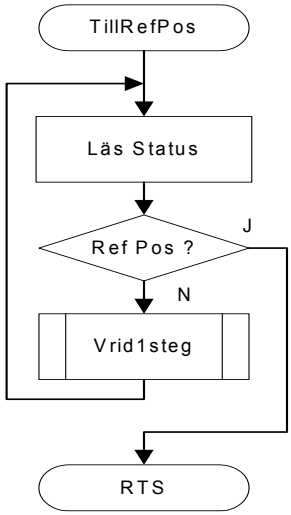


Nolla: Passiv statussignal
Etta: Aktiv statussignal

referensposition

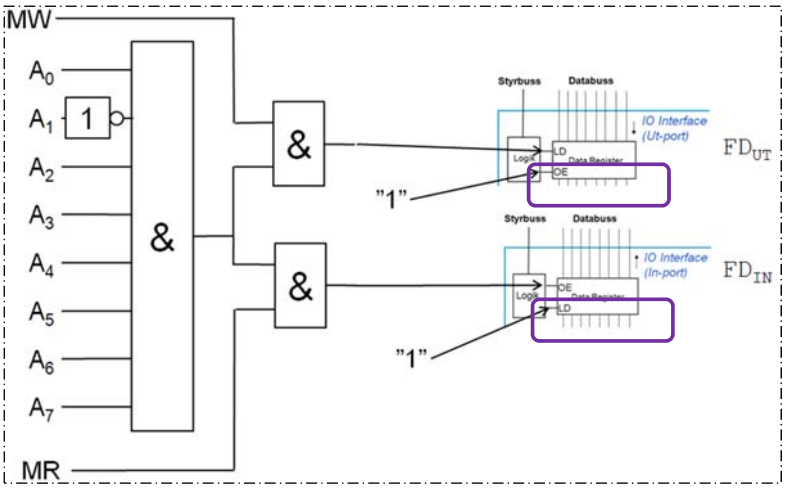
Bit 0 = 0: Arbetsstycket är inte i referensposition
Bit 0 = 1: Arbetsstycket är i referensposition

b_0=1 ?



Att bara ändra en bit i taget

```
* Läs nuvarande styrord
        LDAA    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

Variabel `DCCopy` ska hela tiden ha samma värde som `DCtrl` hade haft om porten varit läsbar…

För att nollställa en bit används nu:

```
        LDAA        DCCopy
        ANDA        #Bitmönster
        STAA        DCtrl
        STAA        DCCopy
```

för att ettställa en bit används:

```
        LDAA        DCCopy
        ORAA        #Bitmönster
        STAA        DCtrl
        STAA        DCCopy
```

```
DCCopy RMB    1
```


Subrutiner för att manipulera styrregistret OUTONE och OUTZERO

```
* Subrutin OUTONE. Läser kopian av
* borrmaskinens styrord på adress
* DCCopy. Ettställer en av bitarna och
* skriver det nya styrordet till
* utporten DCTRL samt tillbaka till
* kopian DCCopy.
* 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

Realtid - Fördröjningar

	Simulator STEP	Simulator RUN	Simulator RUN FAST	Hårdvara
Instruktioner/ sekund	?	10	1000	1 000 000

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

```
****************************************
* SUBRUTIN – DELAY
* Beskrivning: Skapar en fördröjning om
* ANTAL x 500 ms.
* Anrop: LDAA  #6  Fördröj 6*500ms= 3s
*     JSR   DELAY
* Indata:Antal intervall,om 500 ms i A
*
* Utdata: Inga
* Register-påverkan:Ingen
* Anropad subrutin: Ingen.
****************************************
```



```
DELAY   PSHA
        PSHX
        TSTA
        BEQ     DExit
;       Fördröjningsvärde noll

; Konstanten 'Konst' måste
; bestämmas...
ALOOP   LDX     #Konst

XLOOP   LEAX    -1,X
        NOP
        CPX     #0
        BNE     XLOOP

        DECA
        BNE     ALOOP
; Ytterligare fördröjning

DExit   PULX
        PULA
        RTS
```

Programmerad tidsfördröjning

Använd villkorlig assemblering

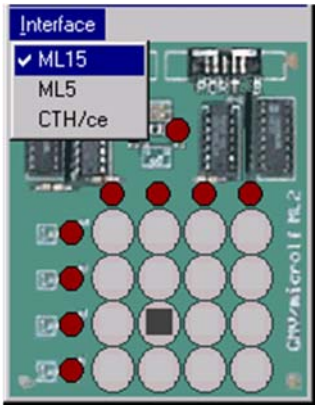
```
#ifdef    SIM
#ifdef    RUNFAST
* Konstant vid Run Fast
Konst     EQU  XXXX
#else
* Konstant vid Run
Konst     EQU  YYYY
#endif
#else
Konst     EQU  ZZZZ
#endif
```

Bestäms experimentellt med simulator

Uppgift 90

Bestäms experimentellt vid laboration

Tangentbord ML15



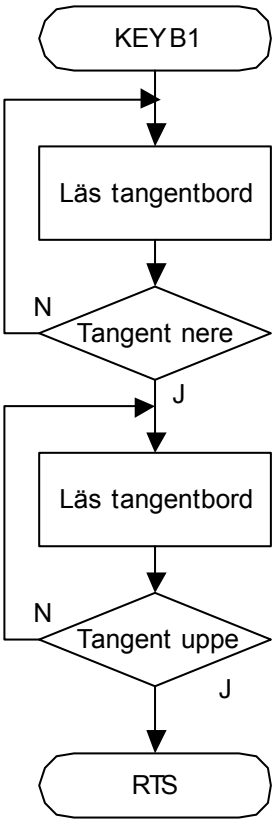
7	6	5	4	3	2	1	0
DAV	0	0	0	B3	B2	B1	B0

Bit 7, DAV: Data Valid; Statusbit som anger nedtryckt tangent
b_7=1: Ingen tangent är för tillfället aktiverad på tangentbordet.
b_7=0: En tangent är aktiverad

Bit 6-4, 0: Används ej.

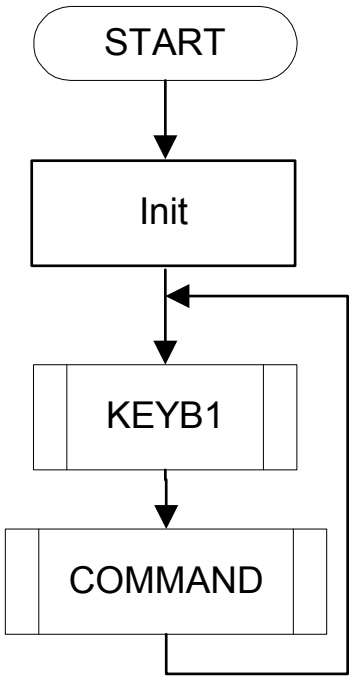
Bit 3-0, B3-B0: Tangentnummer; Anger aktuell tangentnedtryckning.

Uppgift 91



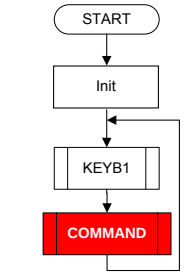
Borrmaskinrobot

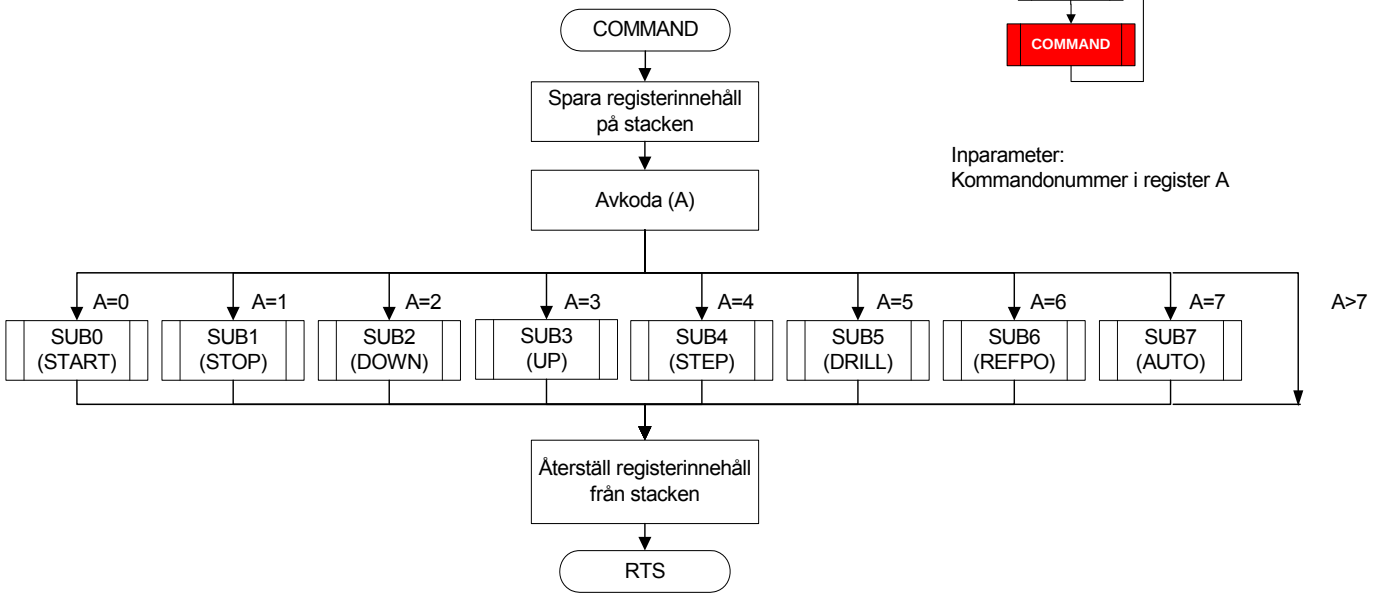
Tangent nr	Operation	subrutin
0	starta borrmotorn	START
1	stoppa borrmotorn	STOP
2	sänk borret	DOWN
3	höj borret	UP
4	rotera arbetsstycket medurs ett steg	STEP
5	borra ett hål	DRILL
6	stega arbetsstycket till referensposition	REFPO
7	borra hål längs cirkeln enligt mönster	AUTO



Rutinen COMMAND

Uppgift 94





Inparameter:
Kommandonummer i register A


```
**********************************
* SUBRUTIN – COMMAND
* Beskrivning:      Rutinen avgör vilken
* kommandosubrutin som skall
* utföras och anropar denna.
* Anrop:            JSR       COMMAND
* Indata:           Kommandonummer i reg A
* Utdata:           Inga
* Reg-påverkan:     A,X
* Anrop subr:       SUB0 - SUB7
**********************************


MAX     EQU         7
COMMAND:
* giltigt värde?
        CMPA        #MAX
        BHI         COMEX
* hopptabellens basadress
        LDX         #JUMPTAB
* offset är 2 bytes per adress

        ASLA
* hämta subrutinens startadress
        LDX         A,X
* utför subrutin
        JSR         ,X
* återvänd från kommandorutin
COMEX: 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
```

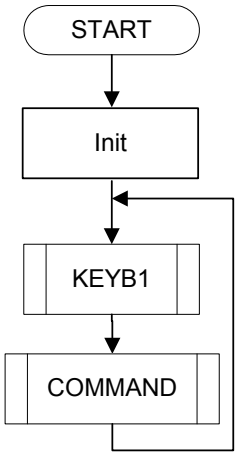
Filen MAIN1.S12

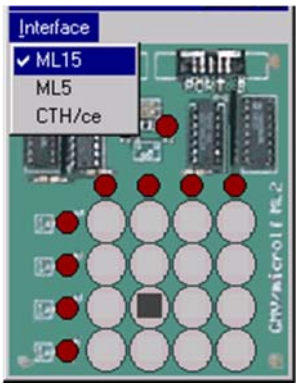
STRUKTUR

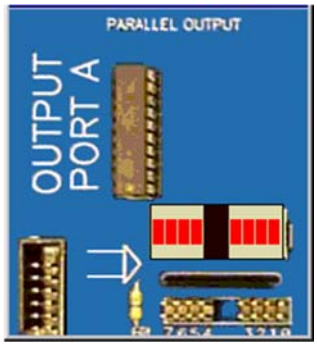
1. Inkludera definitionsfil
2. Initiera systemet
3. Huvudprogram
4. Subrutinen COMMAND
5. Inkludera fil (filer) med ytterligare subrutiner.
6. Plats för variabler

```
* Definitioner
            USE         IODEFS.S12
            ORG         Start
            ---
            ---
************************************
* Huvudprogram
* Invänta vald operation
Loop:
            JSR         KEYB1
            NOP
* Utför vald operation
            JSR         COMMAND
            BRA         Loop
************************************
COMMAND     ---
            ---
            ---
            USE         KeyML15.S12
* Placera alla variabler här
DCCopy      RMB         1
```


Att testa filen MAIN1.S12

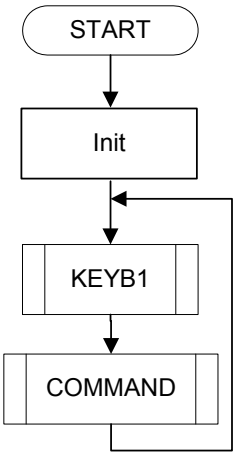


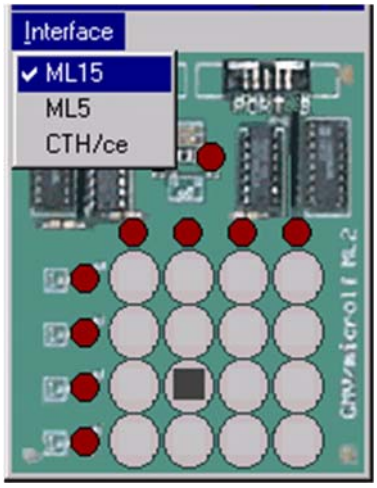


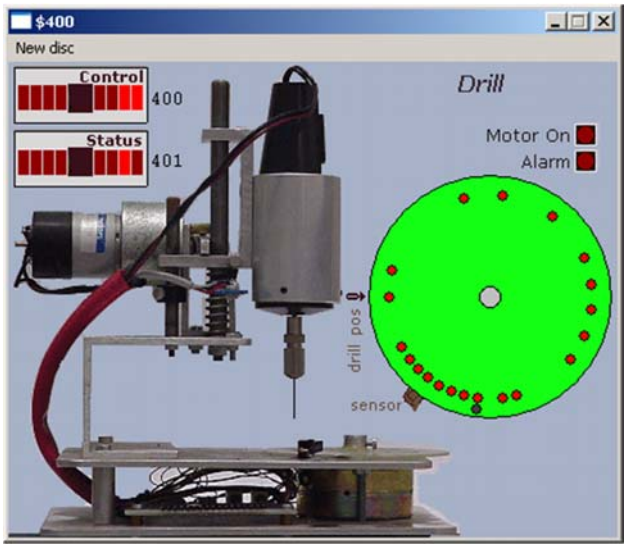



```
SUB0    MOVB #0,ParOut
        RTS
SUB1    MOVB #1,ParOut
        RTS
SUB2    MOVB #2,ParOut
        RTS
SUB3    MOVB #3,ParOut
        RTS
SUB4    MOVB #4,ParOut
        RTS
SUB5    MOVB #5,ParOut
        RTS
SUB6    MOVB #6,ParOut
        RTS
SUB7    MOVB #7,ParOut
        RTS
```

Övriga, funktioner (MAINxx) testas med tangentbord och borrmaskin







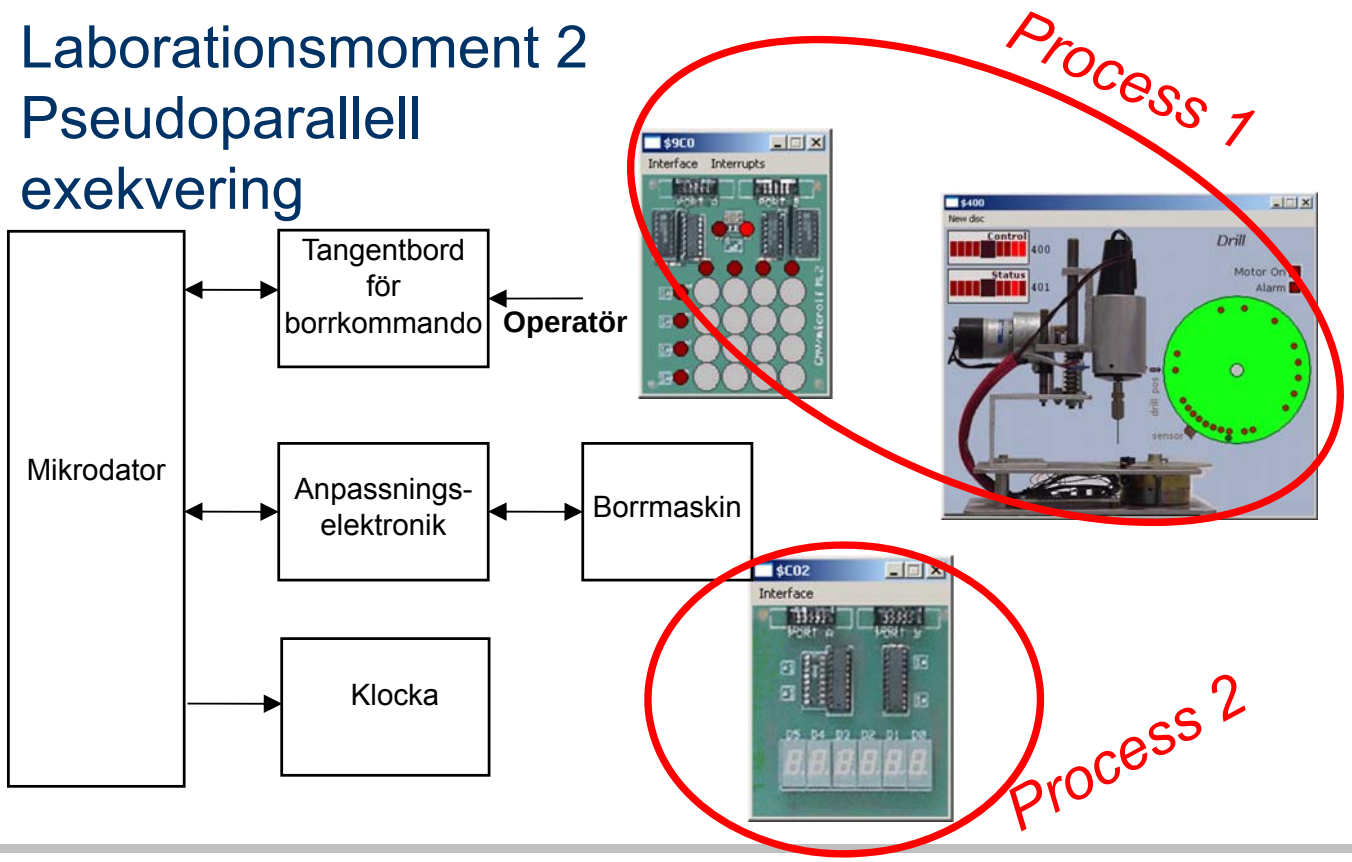
Rutiner START och STOP


```
* SUBRUTIN START. Subrutinen startar
* borrmotorn väntar därefter i 500 ms
* före återhopp så att borret uppnår
* rätt hastighet.
*
* Anrop:                    JSR      START
*
* Indata:                   Inga
* Utdata:                   Inga
* Registerpåverkan:         Ingen
* Anropade subrutiner:      OUTONE
*                           DELAY
```

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

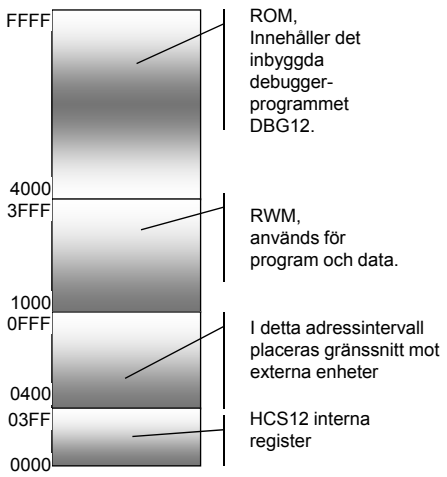
```
Ändringar i huvudprogram...
JUMPTAB    FDB     START,STOP,SUB2,SUB3
           FDB     SUB4,SUB5,SUB6,SUB7
*********************************
* subrutiner för test, byts senare
* ut mot START, STOP, DOWN etc
START .....
      RTS
STOP  .....
      RTS
SUB2  MOVB #2,ParOut
      RTS
      .....
```

Laborationsmoment 2 Pseudoparallell exekvering



Applikation för avbrott (IRQ)

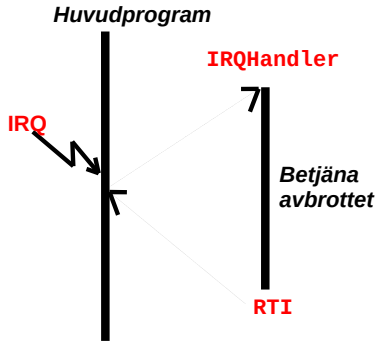
```
;Initieringssekvens
        ORG     XXXX
        ...
; nollställ I-flagga
        ANDCC   #$FE
        JSR     _main
        ...
```




```
        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...

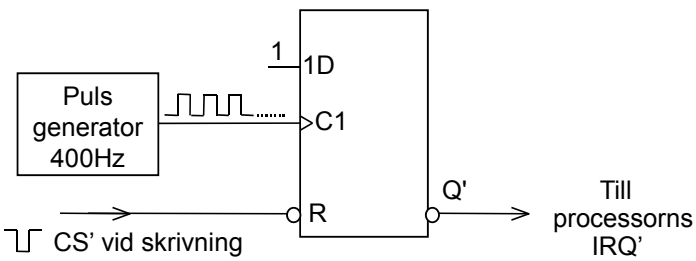
```
        ORG     $3FF2
        FDB     irq_service_routine
; Avbrottshanterare
irq_service_routine:
        RTI
```

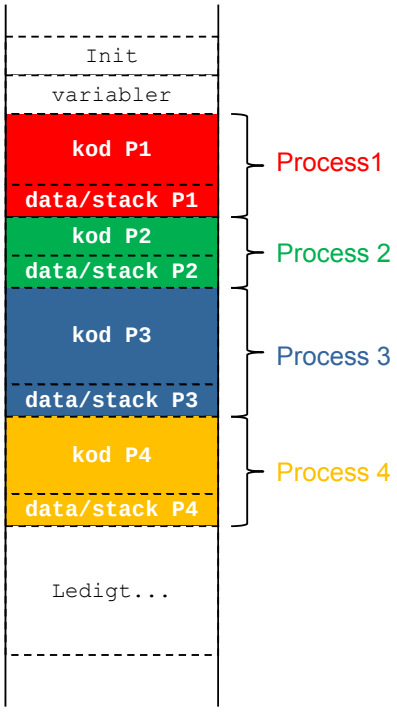

Processbyte

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

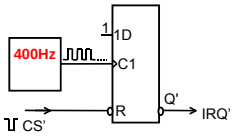
HDW krav: En avbrottskälla som ger regelbundna avbrott (Ex Timer)

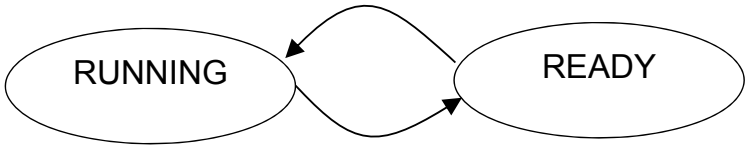
SW krav: En avbrottsrutin (SWITCH) som växlar process



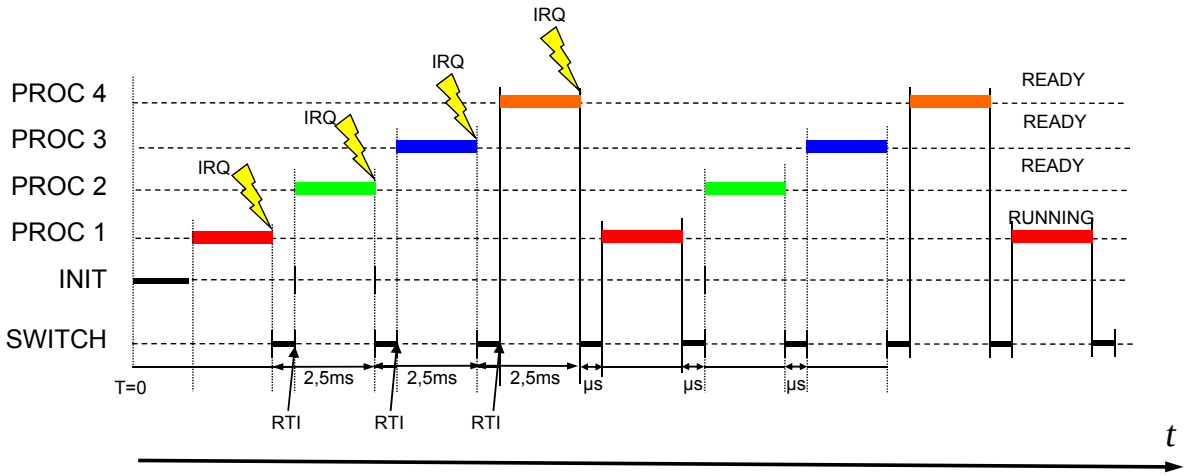


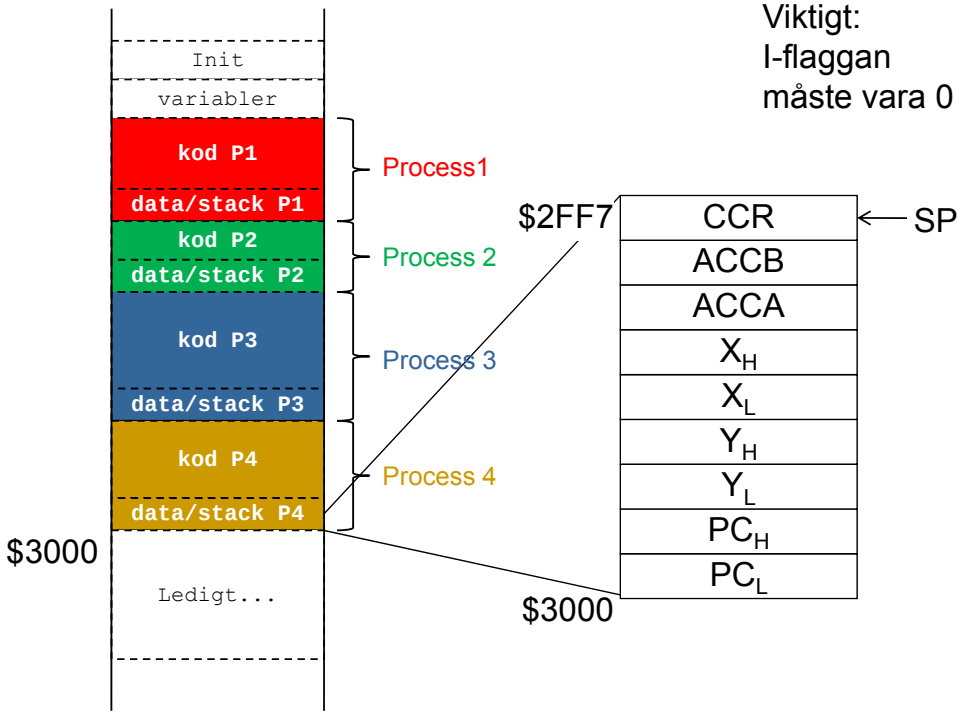
Processtillstånd



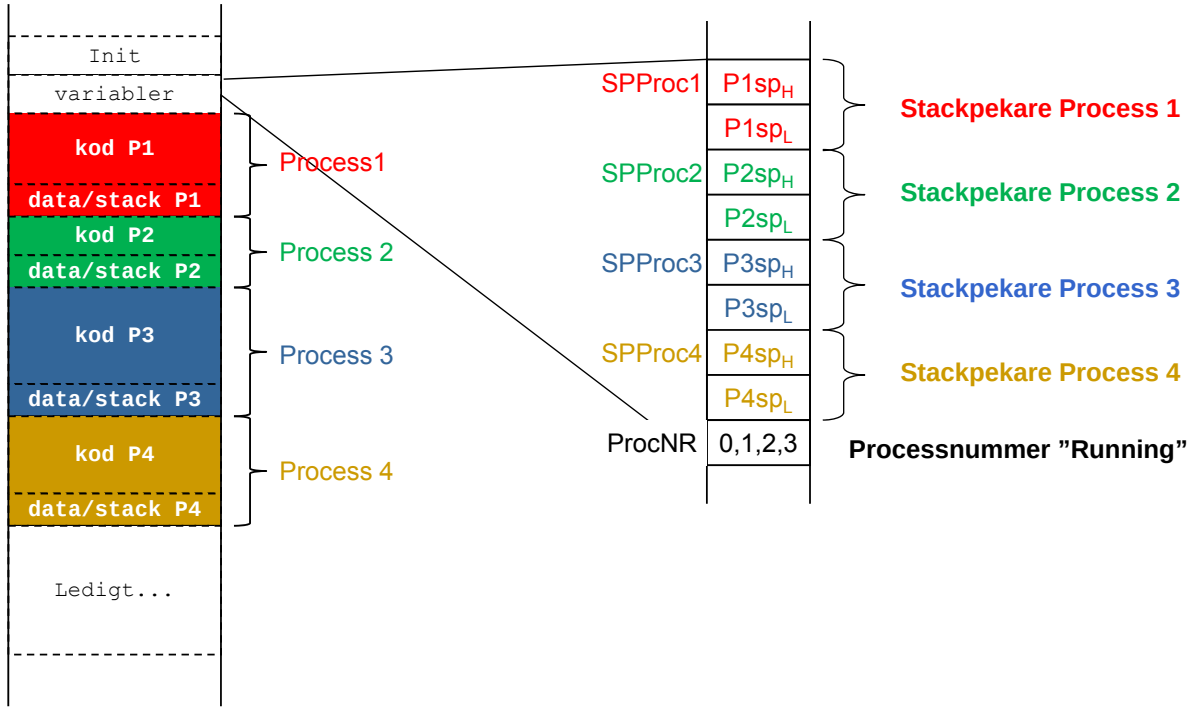


$$f = 400\text{Hz} \Rightarrow p = \frac{1}{400} = 0{,}0025\text{s} = 2{,}5\text{ms}$$





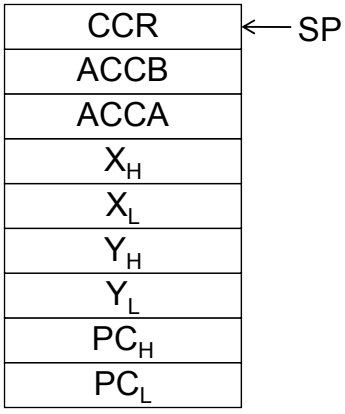
Viktigt:
I-flaggan
måste vara 0



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
```