Synkronisering och undantag

Ur innehållet:

Synkronisering:
hur hanteras situationer när datorn ska kommunicera med en annan enhet med okänd arbetstakt?

Vi ansluter en skrivare

Undantag:
Hur hanteras situationer då något oförutsett inträffar?

Vi beskriver undantagshantering

Förutsättningar för skrivaranslutningen

Vår skrivare är från början mycket enkel:

- Den kan endast arbeta med **ett tecken i taget**.
  (hämtar ett tecken - skriv ut - hämta nästa)
- Det finns inledningsvis **inga handskakningssignaler**
- Max utskriftshastighet: **4 tecken per sekund**.

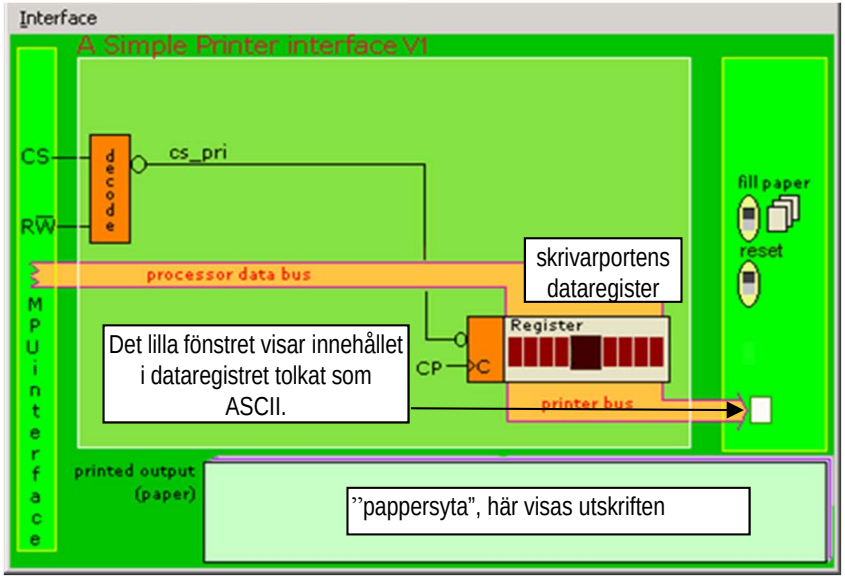
Med portdefinition

```
PRINTER    EQU        $0800
```

och instruktionerna

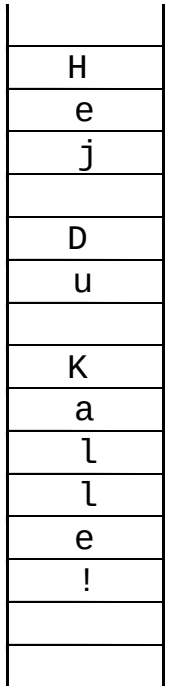
```
           LDAA       #$30
           STAA       PRINTER
```

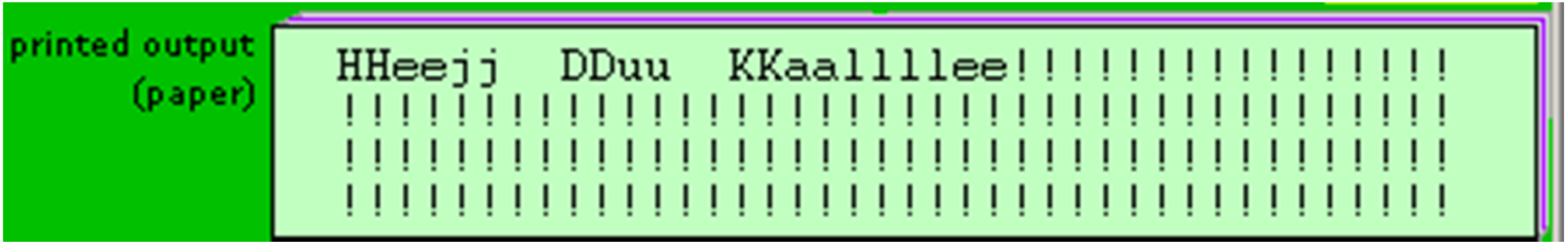
överförs hexadecimala värdet 30 till skrivaren



Första programexemplet

```
* Printer V1_0
        ORG     $1000
        LDX     #Text           ; Pekare till textsträng -> X
Loop    LDAB    1,X+            ; Tecken -> B, peka på nästa
        BEQ     Exit
        STAB    $800            ; Skriv ut till port
        BRA     Loop            ; Fortsätt med nästa tecken
Exit:   NOP
        BRA     Exit

        ORG     $3000
Text    FCS     ”Hej Du Kalle!”
        FCB     0
```



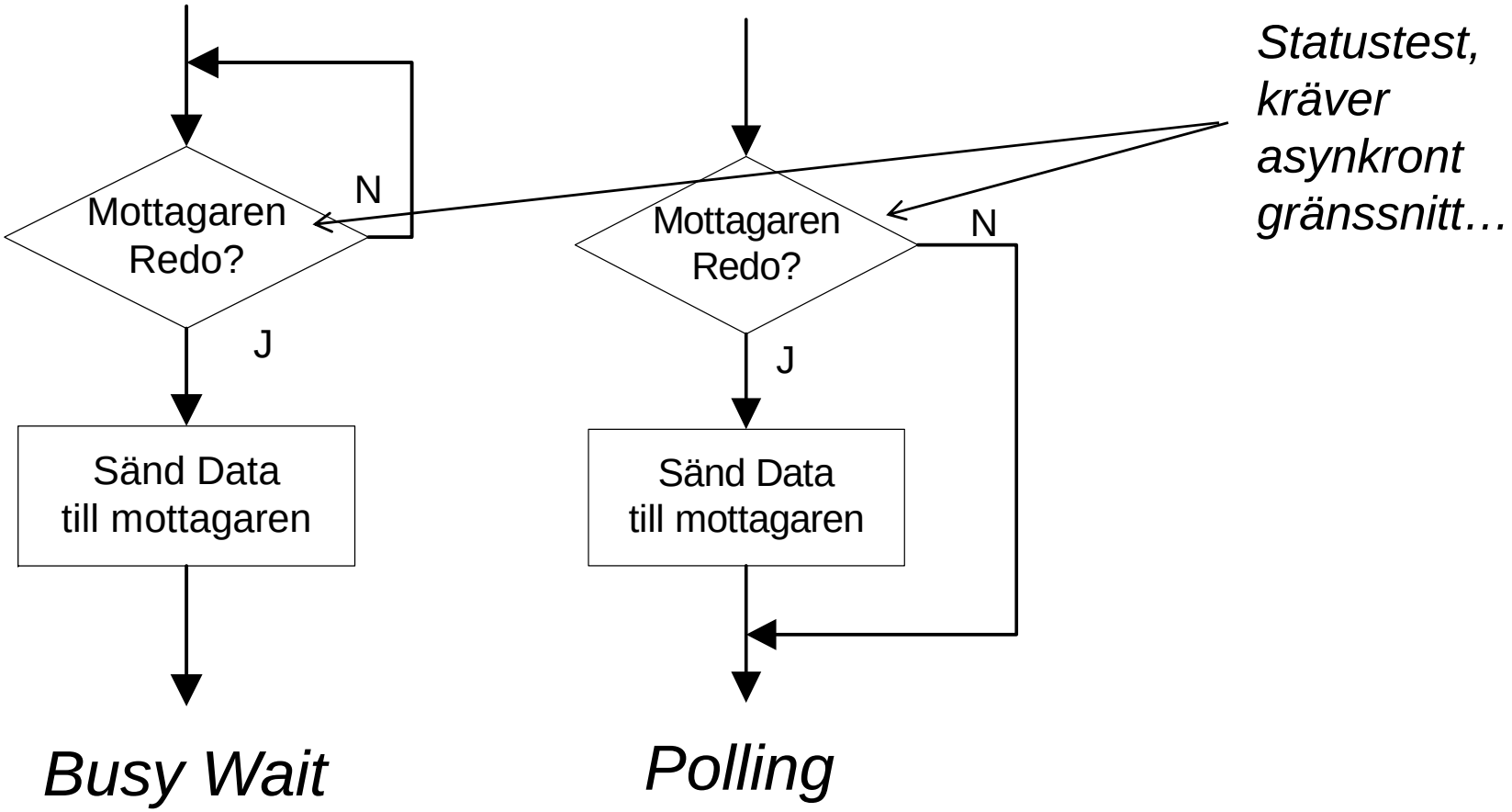


Synkronisera arbetstakterna

	Skrivare	**Simulator STEP**	**Simulator RUN**	**Simulator RUN FAST**	**Hårdvara**
Instruktioner/ sekund		?	10	1000	1 000 000
Tecken/ sekund	4	?	2	200	200 000

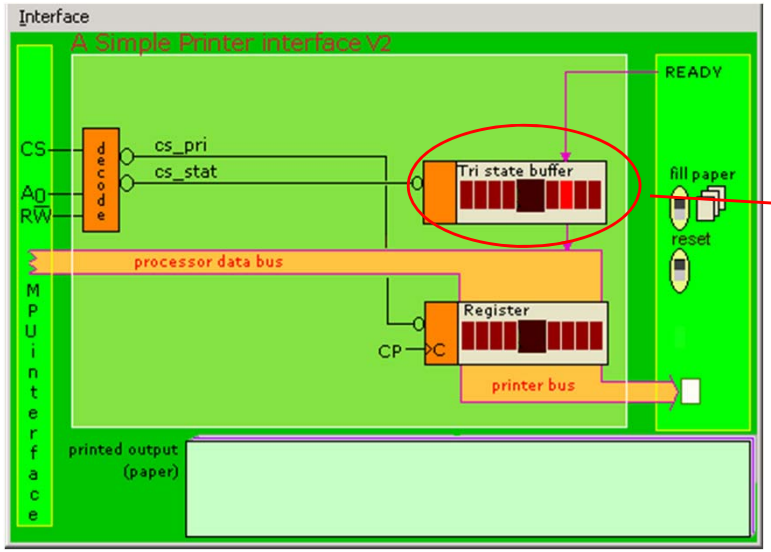
Lösningen blir ***villkorlig överföring*** vilket kräver någon form av så kallade ***handskakningssignaler***.

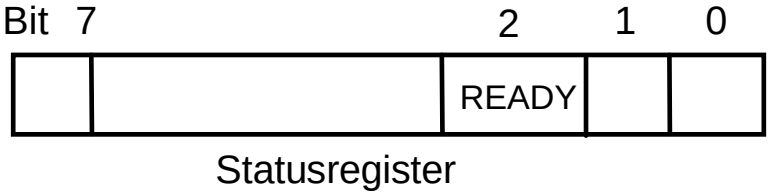
Villkorlig överföring

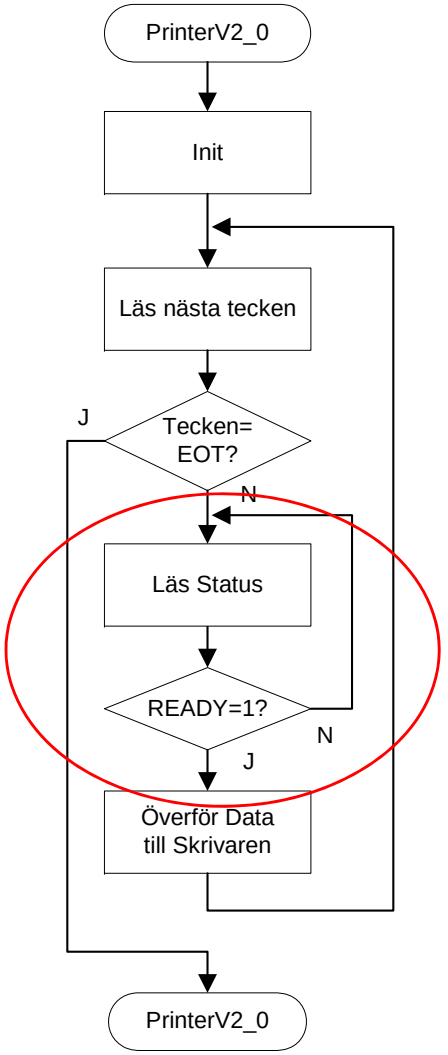


Gränssnitt, version 2

- READY = 1 (Hög nivå) indikerar att skrivaren är klar att ta emot ett nytt tecken.
- READY = 0 (Låg nivå) indikerar att skrivaren är upptagen med att skriva ut ett tecken.



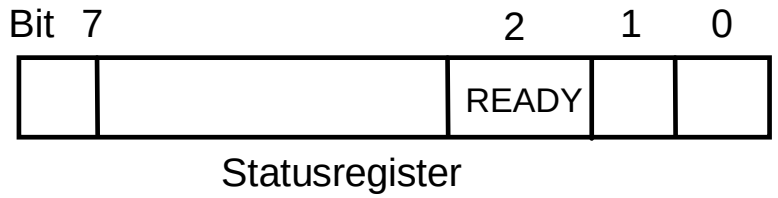




”Programmerarens bild”

READY = 1 (Hög nivå): skrivaren är REDO

READY = 0 (Låg nivå): skrivaren är UPPTAGEN



- Klarar nu situationen att centralenheten arbetar snabbare än skrivaren.
- Fortfarande problem då centralenheten är långsammare än skrivaren.
- Fortfarande problem med att få skrivaren att stoppa då sista tecknet skrivits ut.

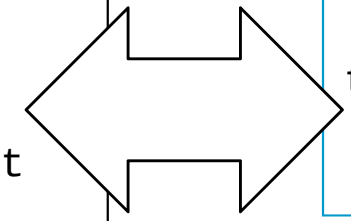
Vi behöver ytterligare handskakningssignal ”Tecken finns”…

```
* Printer V2_1
PRINTER  EQU     $0800
PSTATUS  EQU     $0801
EOT      EQU     4
         ORG     $1000
         LDX     #Text
Loop:    LDAA    1,X+
         CMPA    #0
         BEQ     Exit
LoopForReady:
         LDAB    PSTATUS
         ANDB    #4
         BEQ     LoopForReady
         STAA    PRINTER
LoopForNotReady:
         LDAB    PSTATUS
         ANDB    #4
         BNE     LoopForNotReady

         BRA     Loop
Exit:    NOP
         BRA     Exit

         ORG     $3000
Text:    FCS     "Hej Du Kalle!"
         FCB     0
```

Speciella instruktioner....

”Branch if bit(s) is clear”

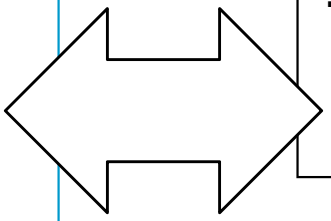
```
...
test:
        LDAB    $0801   ; läs status
        ANDB    #4      ; testa bit 2
        BEQ     test    ; om 0, fortsätt
        ...
```



```
        ...
test:
        BRCLR   $801,#4,test
        ...
```

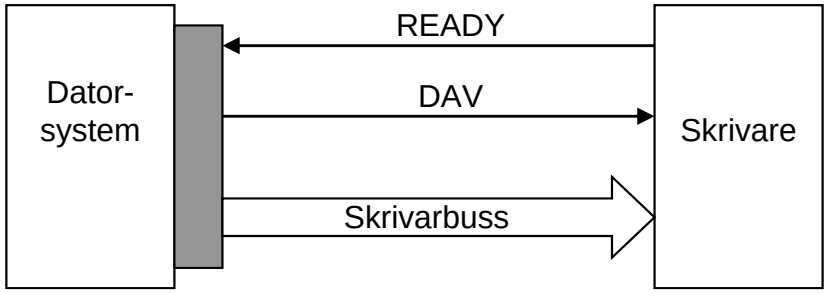
”Set bit(s) in memory”

```
        ...
        PSHB
        LDAB    $0802
        ORAB    #2
        STAB    $802
        PULB
        ..
```



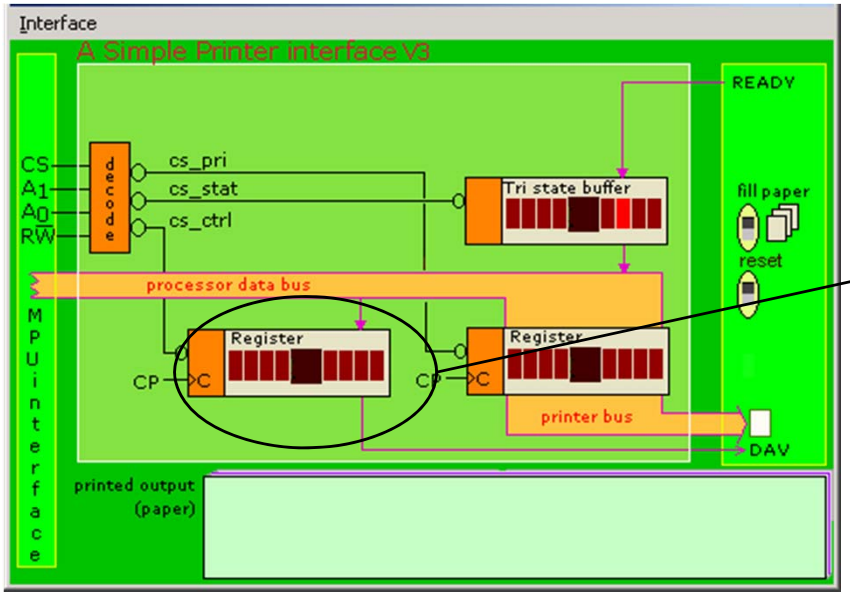
```
...
        BSET    $0802,#2 ; ettställ bit 1
        ...
```

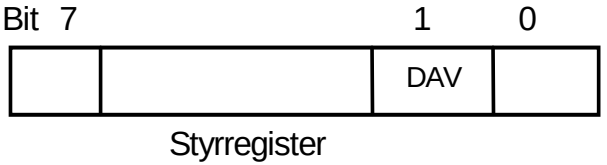
Gränssnitt, version 3

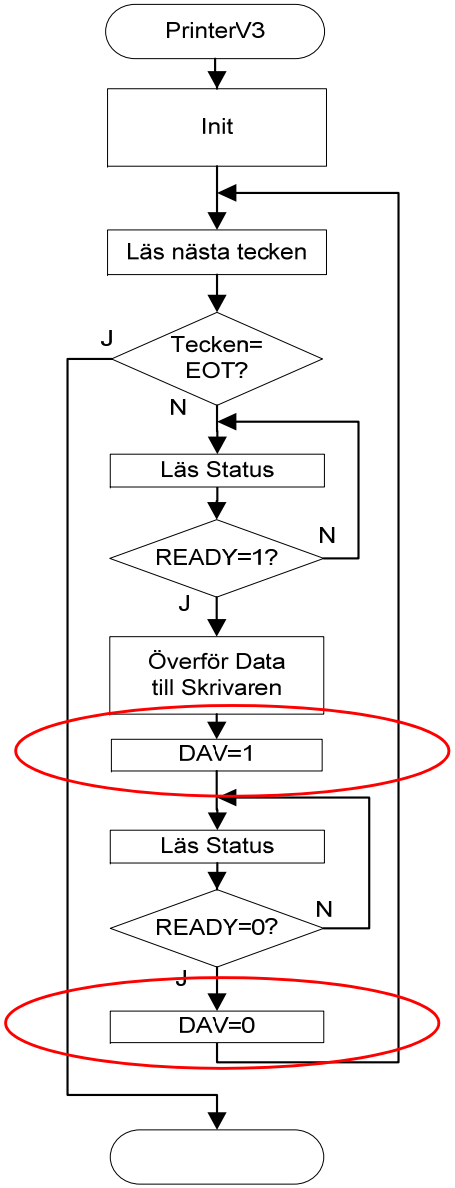


DAV = 1 (Hög nivå) indikerar för skrivaren att giltigt tecken finns att hämta på skrivarbussen.

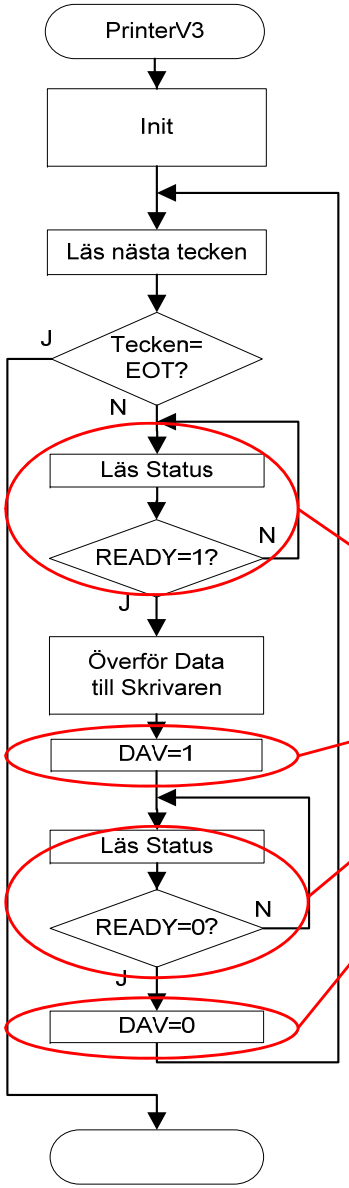
DAV = 0 (Låg nivå) indikerar för skrivaren att skrivarbussen har ett ogiltigt värde.







Händelser i Datorsystemet	*Händelser i skrivaren*
Inväntar READY=1	Skrivaren är upptagen med att skriva ut ett tecken. READY=0.
	Skrivaren är redo för nästa tecken och sätter READY=1
När READY=1 skrivs nästa tecken till skrivarens dataregister. Sätter DAV=1	Inväntar DAV=1
Inväntar READY=0	Ser att DAV=1. Läser nytt tecken från skrivarbussen. Signalerar upptagen, READY=0.
När READY=0 nollställs DAV som indikation på att det inte finns giltigt tecken på skrivarbussen	Skrivaren är upptagen med att skriva ut ett tecken. READY=0.




```
* Printer V3

PRINTER  EQU      $0800
PSTATUS  EQU      $0801
PCONTROL EQU      $0802

         ORG      $1000
         LDX      #Text
Loop:    LDAA     1,X+
         CMPA     #0
         BEQ      Exit
Ready:
         BRCLR    PSTATUS,#4,Ready
         STAA     PRINTER
         BSET     PCONTROL,#2
NotReady:
         BRSET    PSTATUS,#4,NotReady
         BCLR     PCONTROL,#2
         BRA      Loop
Exit:    NOP
         BRA      Stop

         ORG      $3000
Text:    FCS      "Hej Du Kalle!"
         FCB      0
```

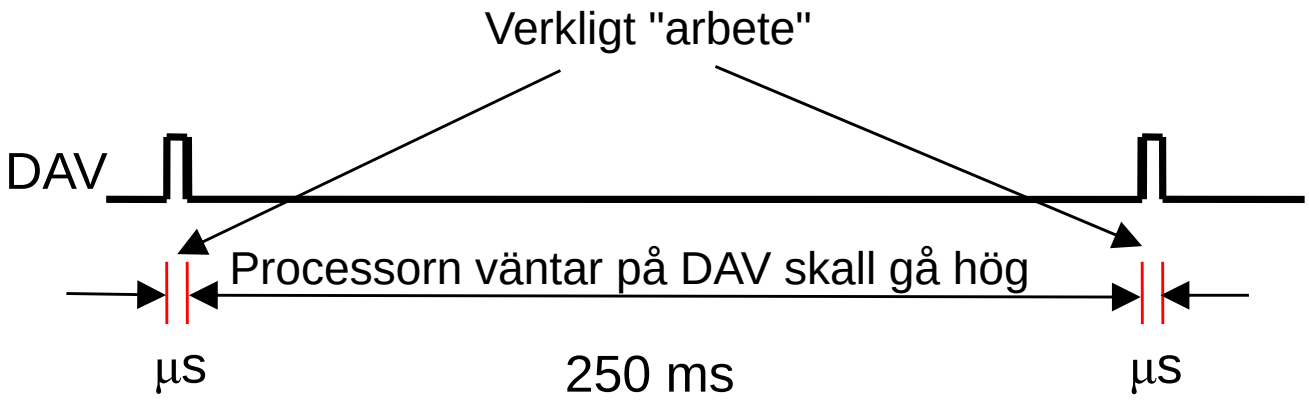
Resultat

Klarar nu situationen att centralenheten arbetar snabbare än skrivaren.

Klarar nu situationen då centralenheten är långsammare än skrivaren.

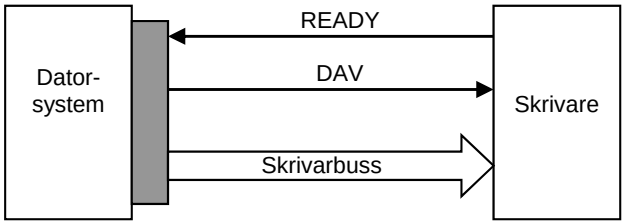
Klarar nu situationen med att få skrivaren att stoppa då sista tecknet skrivits ut

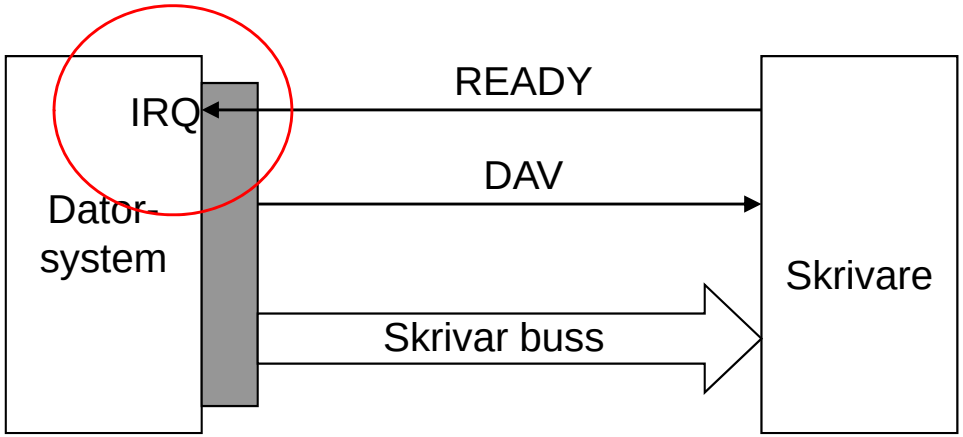
Lösningen är dock hopplöst ineffektiv med tanke på hur vi utnyttjar systemet…

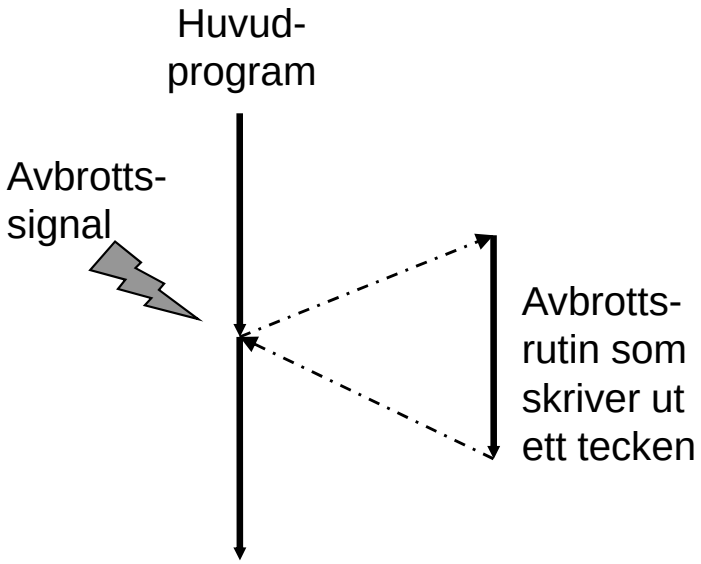


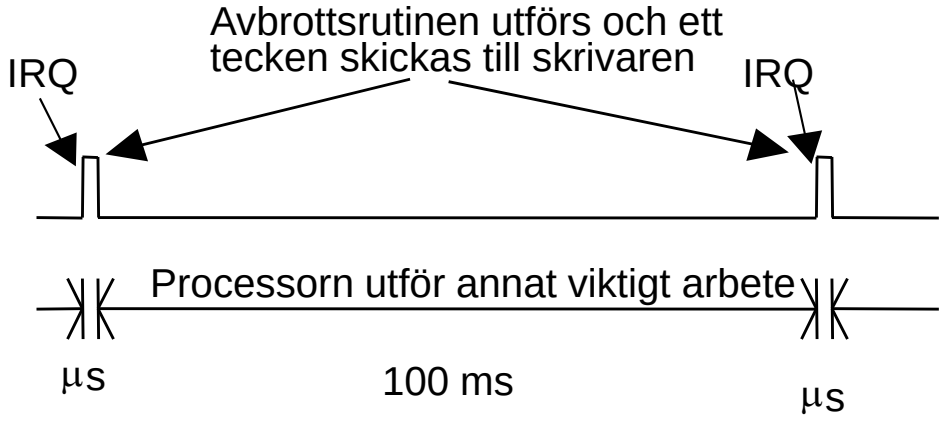
Introduktion till ”Undantagshantering”

Interrupt ReQuest (IRQ), begäran om avbrott…

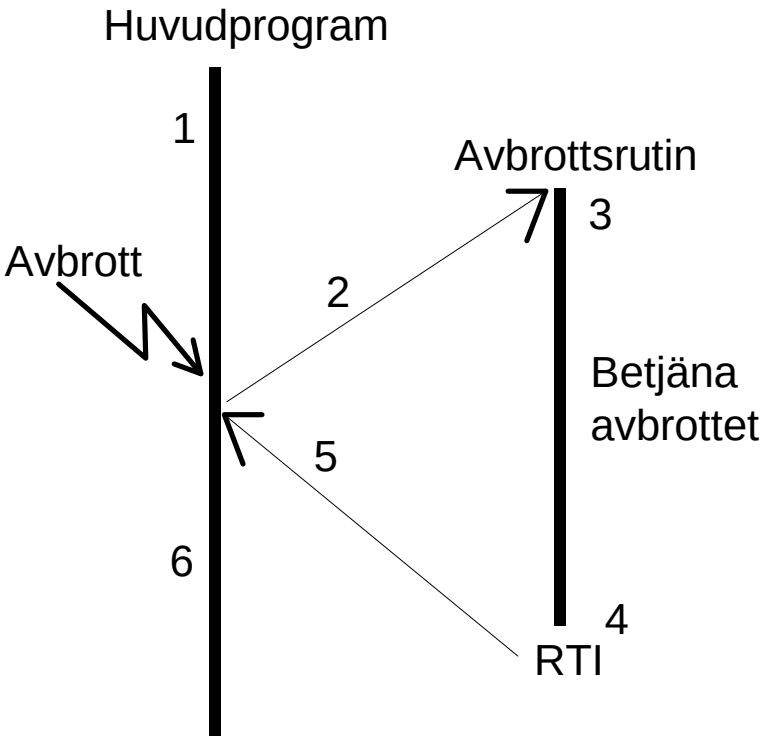




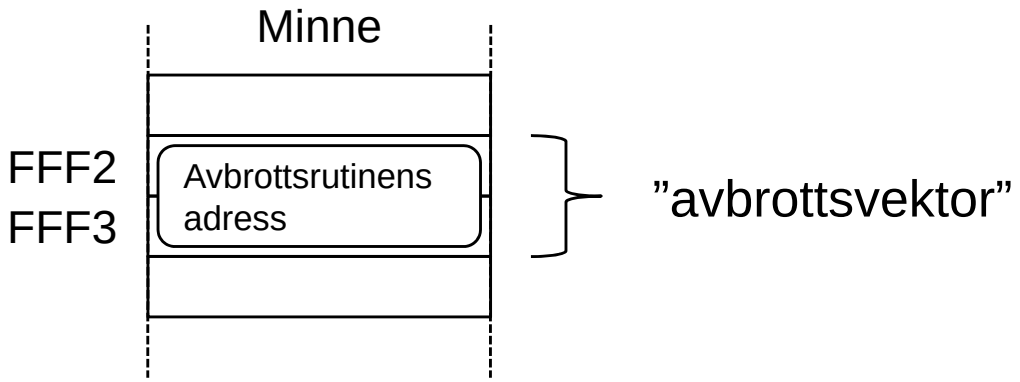




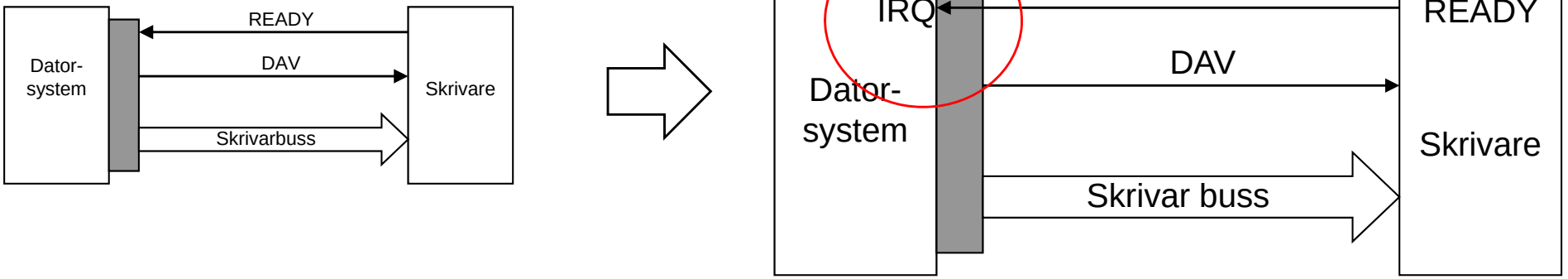
Avbrottshantering

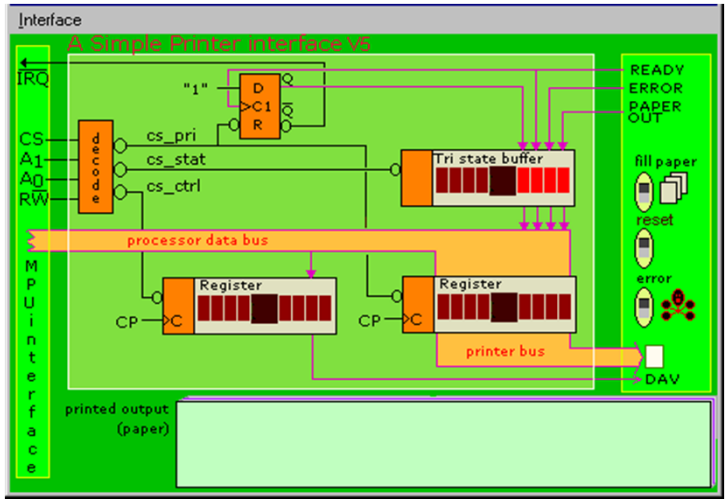


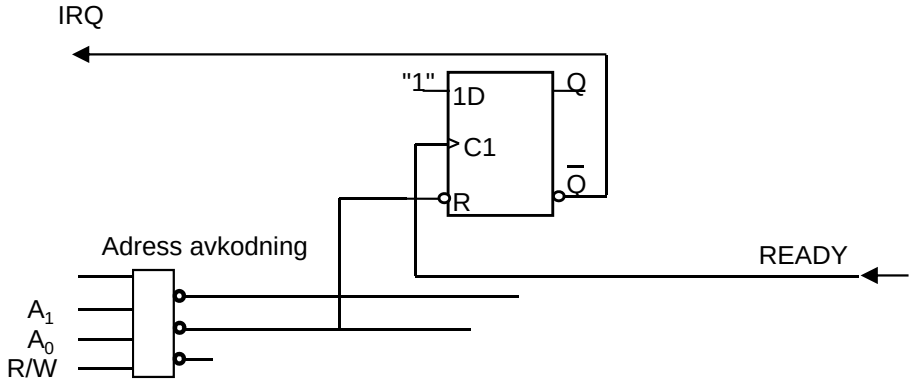
1) Huvudprogram exekveras när ett avbrott aktiveras
2) Hopp till avbrottsrutin
3) Avbrottsrutin startar
4) Avbrottsrutin avslutas med en speciell instruktion,
   return from interrupt (RTI)
5) Återhopp till huvudprogram
6) Huvudprogrammet fortsätter.



Skrivarport Version 5, med avbrott





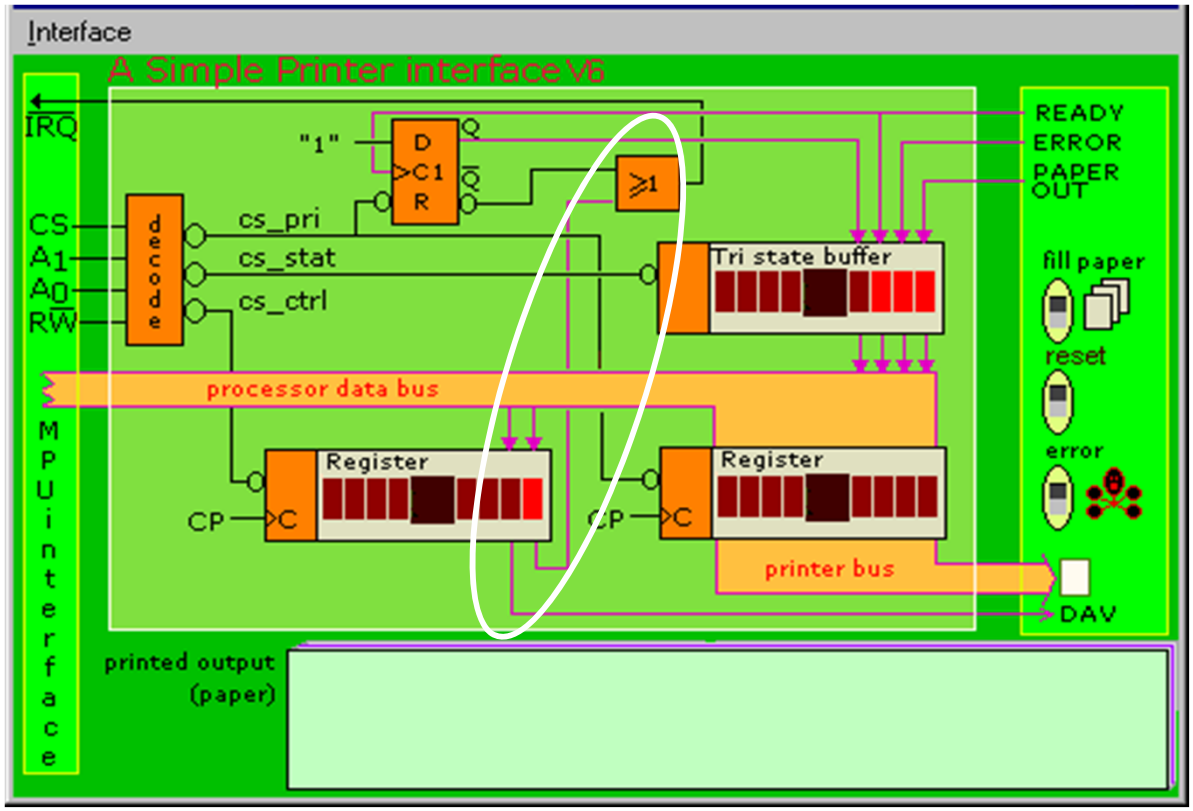


Denna lösning genererar ALLTID avbrott då skrivarens teckenbuffert är tom….

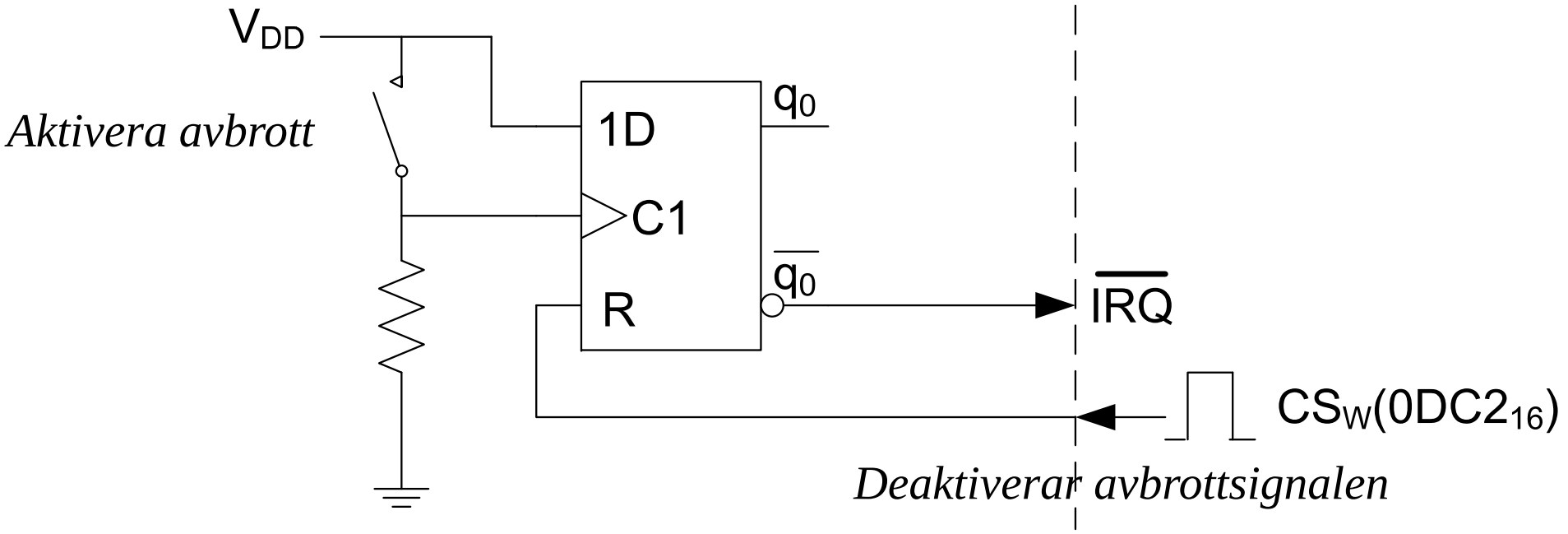
Skrivarport, Version 6

I den sista versionen kan vi stänga av avbrotten från skrivaren.
”Disable Interrupt”

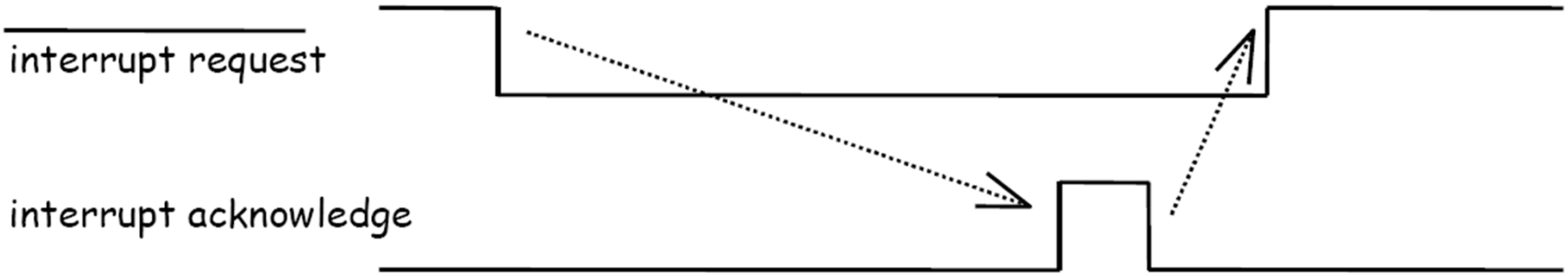
Lämnas som självverksamhet...



Avbrottsvippa

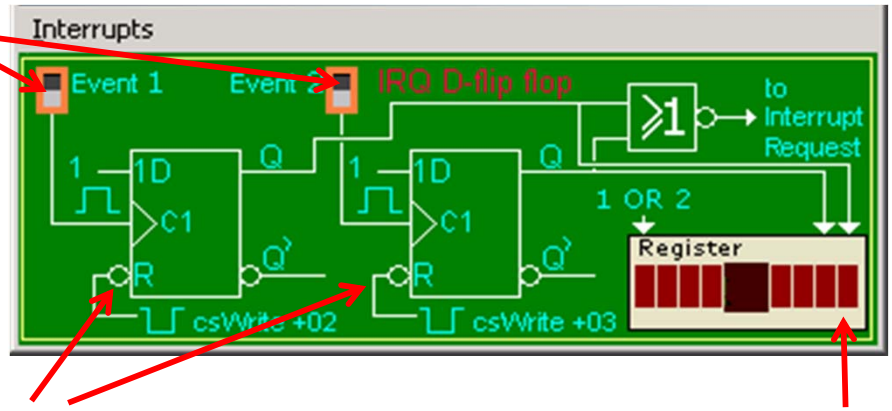


Kvittering av avbrott ”Interrupt Acknowledge”



EXEMPEL: (jfr: laborationskort ML19)

Aktivera avbrott



Deaktivera (”kvittera”) avbrott

Status hos avbrottskällor

EXEMPEL, Arbetsbok uppgift 67 (”Irq3.s12”)

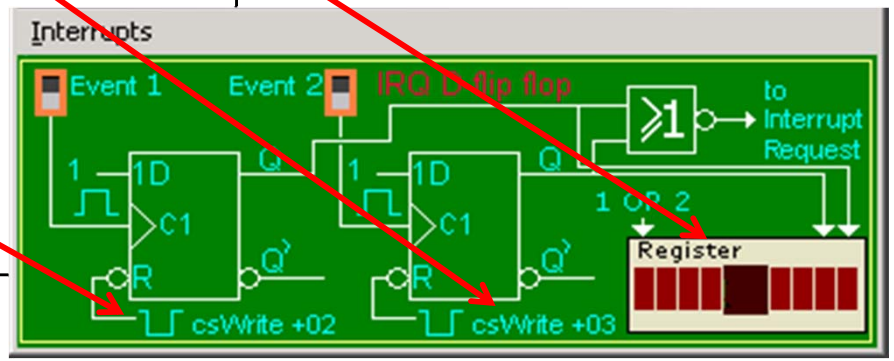
```
; Definitioner, initieringssekvens
; och avbrottsvektor
Port1     EQU       $0400
Port2     EQU       $0401
IrqStat   EQU       $0D00
IrqRes1   EQU       $0D02
IrqRes2   EQU       $0D03

          ORG       $1000
; Nollställ våra variabler
          CLR       Var1
          CLR       Var2
          CLR       IrqRes1
          CLR       IrqRes2
; Initiera avbrottsvektor IRQ
          LDX       #IrgR
          STX       $3FF2

; Sätt om avbrottsmasken hos processorn
          CLI
```

```
; Huvudprogram
main_loop
          INC       Var1
          MOVB      Var1,Port1
          MOVB      Var2,Port2
          BRA       main_loop

; Variabler

Var1      RMB       1
Var2      RMB       1
```

uppgift 67, forts.

```
* Avbrottsrutin
IrqR:
        LDAA    IrqStat
        BITA    #2          ; Event 2 ?
        BEQ     IrqR1       ; Om inte prova nästa
        CLR     IrqRes2
        INC     Var2        ; Räkna upp

IrqR1:
* Kontrollera även Event 1...
        BITA    #1
        BEQ     IrqR2
        CLR     IrqRes1
        CLR     Var2        ; Nollställ

IrqR2:
        RTI
```

```
IrqStat   EQU       $0D00
IrqRes1   EQU       $0D02
IrqRes2   EQU       $0D03
```

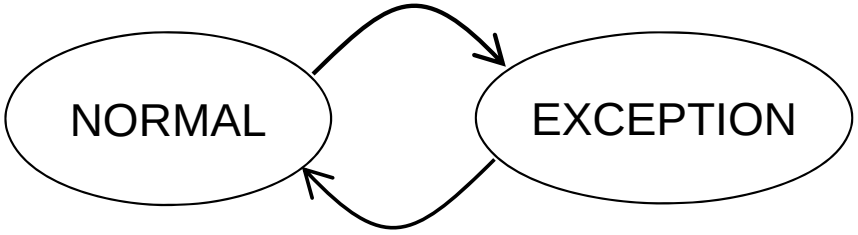


Exekveringstillstånd

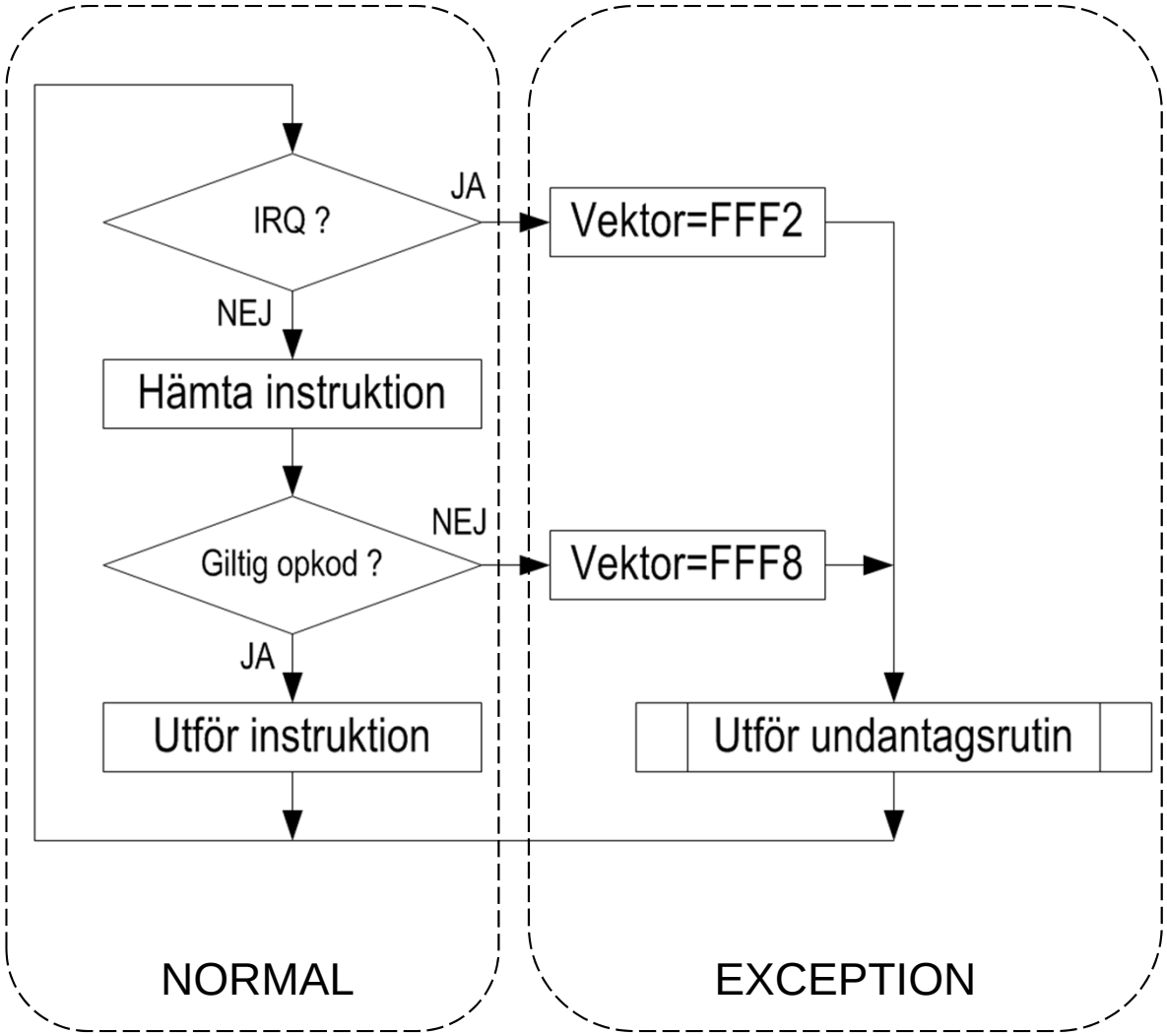
Processorn befinner sig alltid i något av tillstånden:

NORMAL, processorn hämtar och utför instruktioner, dvs. normal exekvering.

EXCEPTION, något "undantag" har inträffat som gör att processorn inte kan (eller ska) fortsätta normal exekvering.



EXEMPEL PÅ UNDANTAGSTYPER



Starta undantagshantering
Spara registerinnehåll…

”Atomär operation”

```
PUSH  PC
PUSH  Y
PUSH  X
PUSH  D
PUSH  CCR
```

Avslut av undantagshantering
”ReTurn from Interrupt”, RTI

”Atomär operation”

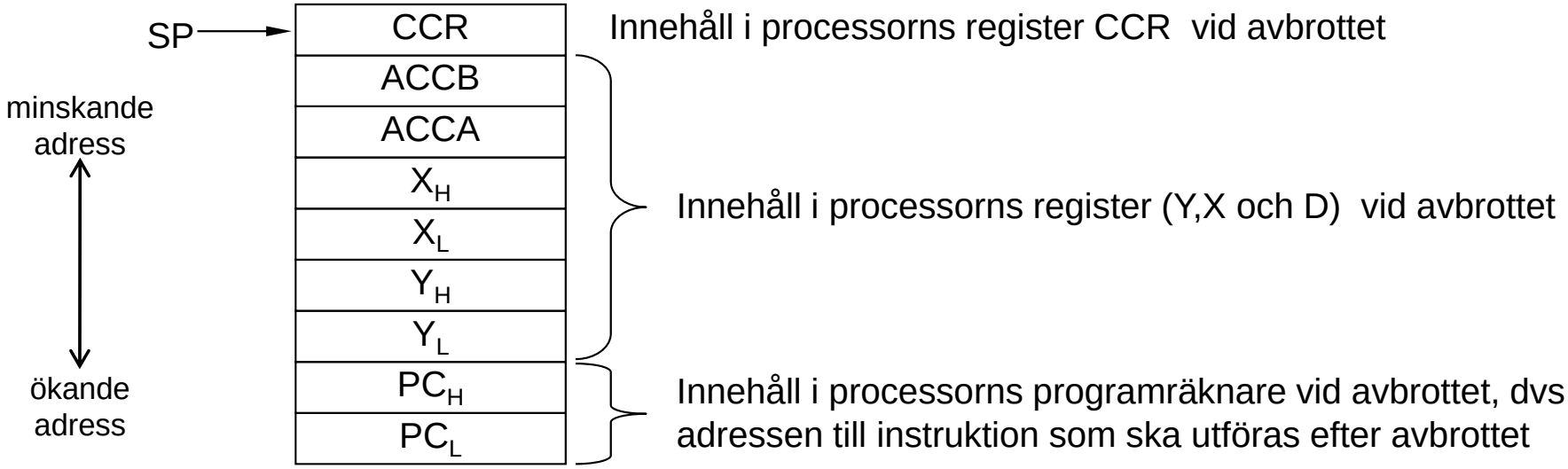
```
PULL   CCR
PULL   D
PULL   X
PULL   Y
PULL   PC
```

RTI — **Return from Interrupt** — **RTI**

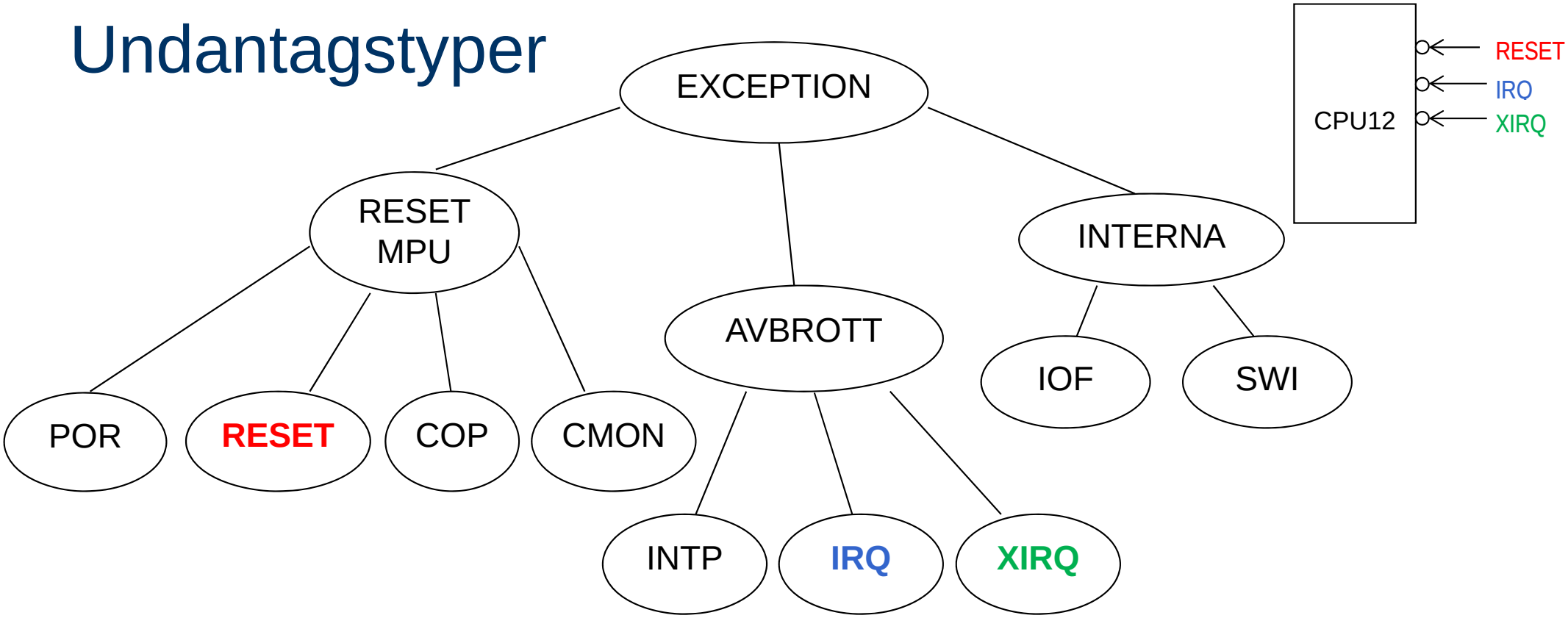
Operation:
$(M_{(SP)}) \Rightarrow CCR; (SP) + \$0001 \Rightarrow SP$
$(M_{(SP)} : M_{(SP+1)}) \Rightarrow B : A; (SP) + \$0002 \Rightarrow SP$
$(M_{(SP)} : M_{(SP+1)}) \Rightarrow X_H : X_L; (SP) + \$0004 \Rightarrow SP$
$(M_{(SP)} : M_{(SP+1)}) \Rightarrow PC_H : PC_L; (SP) - \$0002 \Rightarrow SP$
$(M_{(SP)} : M_{(SP+1)}) \Rightarrow Y_H : Y_L; (SP) + \$0004 \Rightarrow SP$

Description: Restores system context after interrupt service processing is completed. The condition codes, accumulators B and A, index register X, the PC, and index register Y are restored to a state pulled from the stack. The X mask bit may be cleared as a result of an RTI instruction, but cannot be set if it was cleared prior to execution of the RTI instruction.

Stackens utseende i avbrottsrutin



Undantagstyper

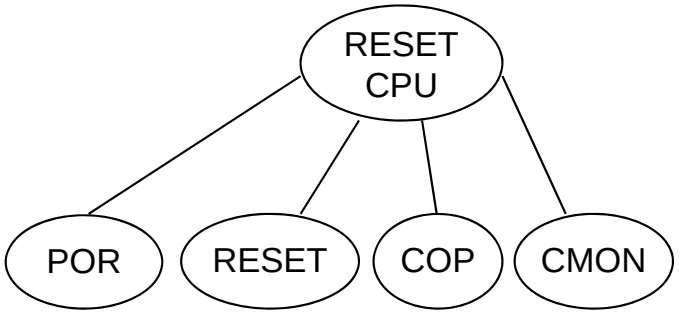


RESET MPU, händelser som alltid föranleder återstart (RESET) av processorn.

AVBROTT, externa händelser, dvs. utanför processorn, detta kan alltså vara enheter på samma krets som processorn (sammanbyggda periferienheter), det kan också vara en speciell insignal (IRQ eller XIRQ) som aktiveras.

INTERNA, händelser som uppträder under programexekvering, exempelvis att en otillåten instruktion avkodas (IOF) eller den speciella instruktionen SWI.

Fatala fel, kräver RESET av CPU

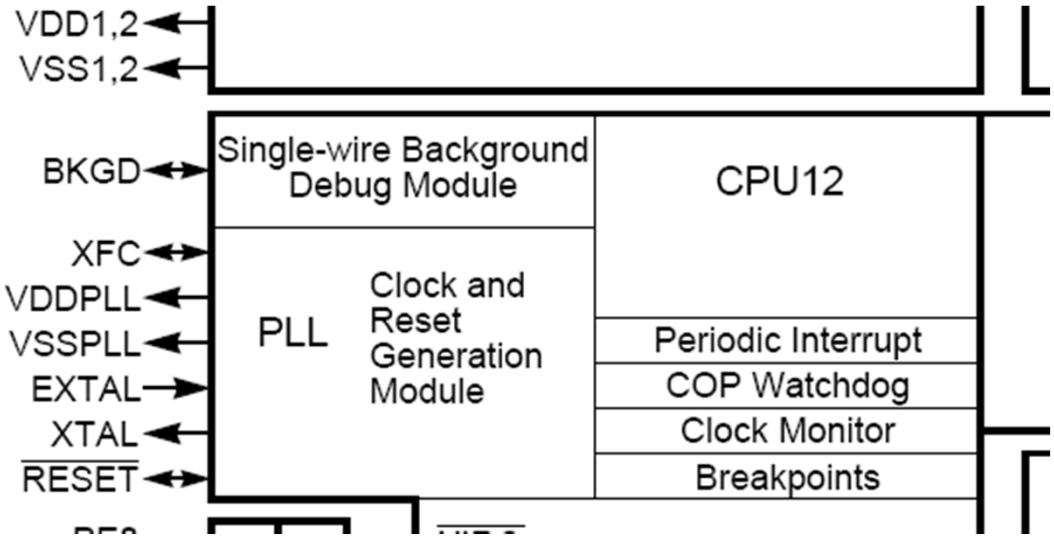


POR, *Power On Reset*, vid spänningstillslag

RESET, insignal till processorn aktiveras.

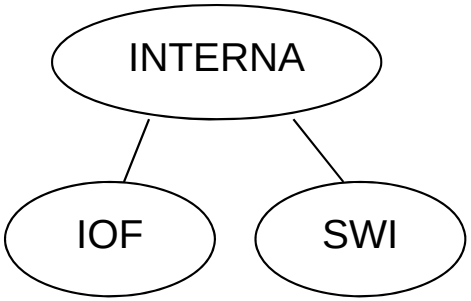
COP, *Computer Operating Properly*, så kallad *watchdog-funktion*.

CMON, *Clock Monitor Reset*, övervakar E-klockan, om frekvensen sjunker under 10 kHz genereras RESET.



Adress (hex)	Funktion
FFFE	RESET, Startvektor
FFFC	Clock Monitor Fail
FFFA	COP Watchdog Timeout
FFF8	Illegal Op Code
FFF6	SWI
FFF4	XIRQ
FFF2	IRQ
FF00-FFF0	Enhetsspecifika vektorer, skiljer sig något beroende på olika varianter

Interna undantag



Om processorn avkodar en otillåten operationskod kallas detta *Illegal Opcode Fetch* (IOF).
Processorn avbryter då,
sparar registerinnehåll på stacken,
Läser vektorn för IOF och
utför undantagshantering.

Instruktionen *SoftWare Interrupt* (SWI) fungerar på samma sätt, men har en annan vektor och en bestämd operationskod.

Adress (hex)	Funktion
FFFE	RESET, Startvektor
FFFC	Clock Monitor Fail
FFFA	COP Watchdog Timeout
FFF8	Illegal Op Code
FFF6	SWI
FFF4	XIRQ
FFF2	IRQ
FF00-FFF0	Enhetsspecifika vektorer, skiljer sig något beroende på olika varianter

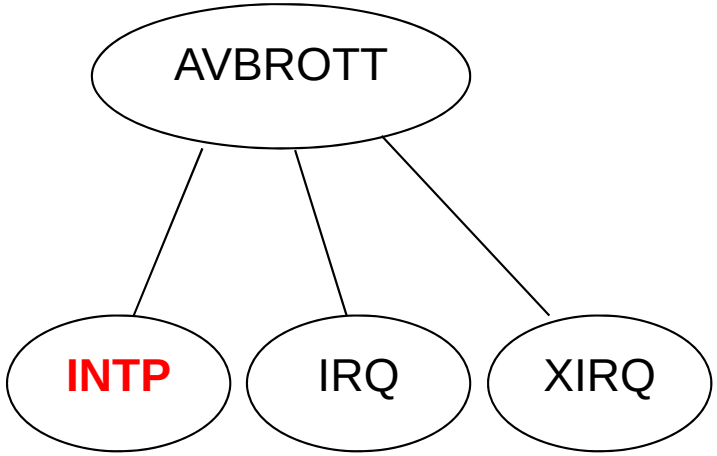
EXEMPEL, Hantera ”Software Interrupt”, SWI

```
        ORG     $1000
main    LDAB    #$11
        LDAA    #$22
        LDX     #$3333
        LDY     #$4444
        NOP
        SWI
        NOP
        BRA     main
```

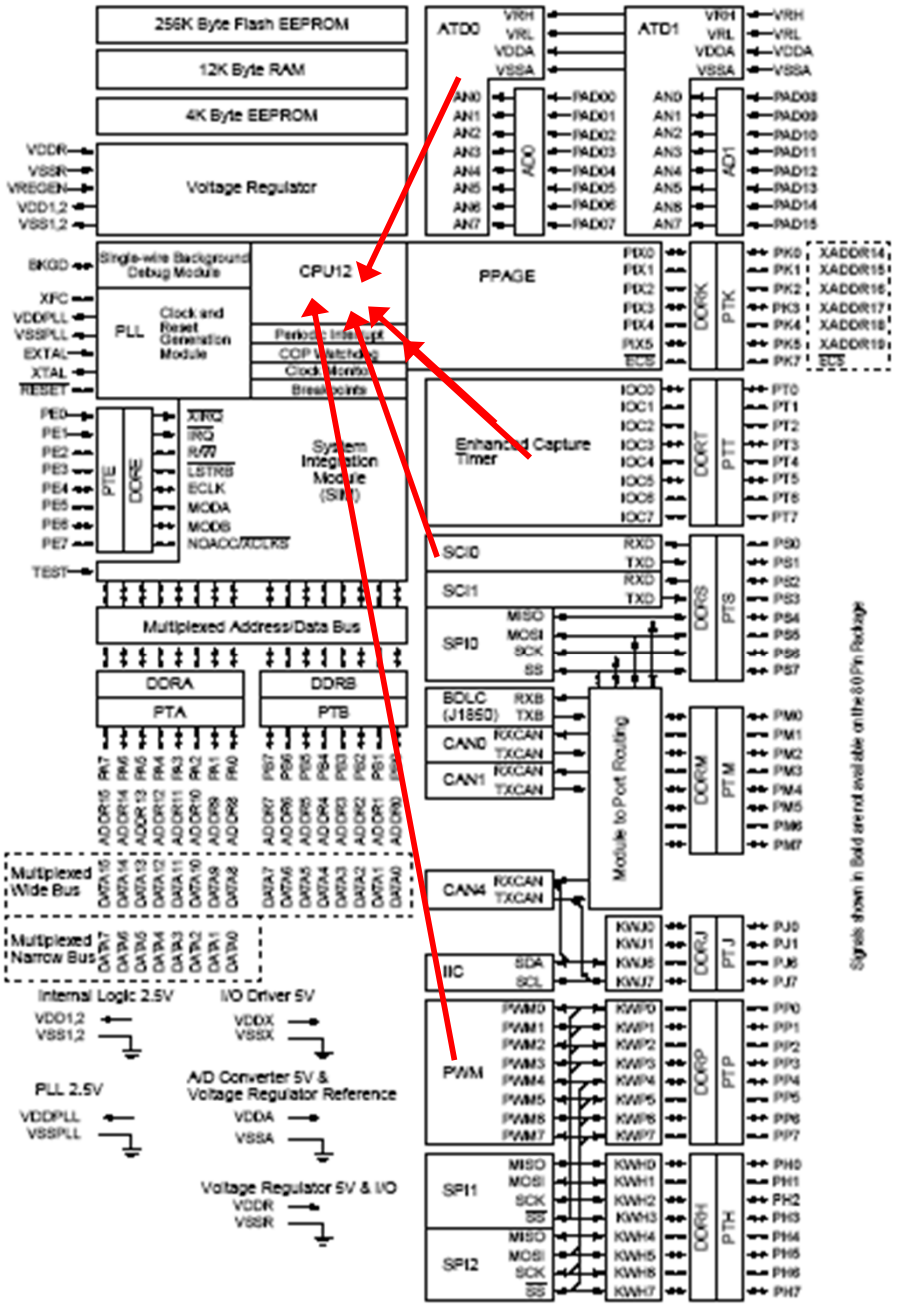
```
SWI_hantering:
        CLRA
        NOP
        RTI

        ORG     $3FF6
        FDB     SWI_hantering
```

Adress (hex)	Funktion
FFFE	RESET, Startvektor
FFFC	Clock Monitor Fail
FFFA	COP Watchdog Timeout
FFF8	Illegal Op Code
FFF6	**SWI**
FFF4	XIRQ
FFF2	IRQ
FF00-FFF0	Enhetsspecifika vektorer, skiljer sig något beroende på olika varianter

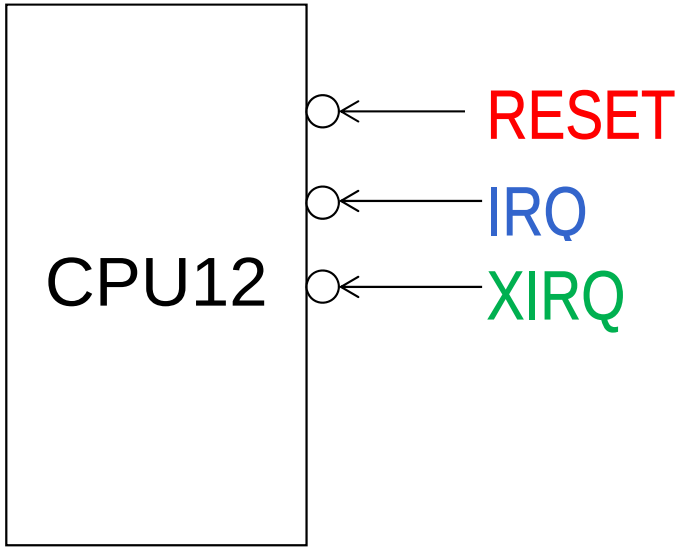
Internt genererade avbrott

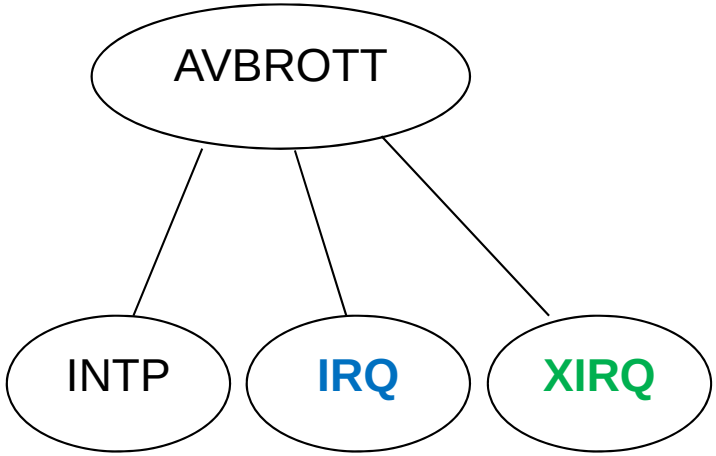


Adress (hex)	Funktion
FFF0	Real Time Interrupt
FFEE	Enhanced Capture Timer channel
FFEC	Enhanced Capture Timer channel 1
FFEA	Enhanced Capture Timer channel 2
....	....
FF8E	Port P Interrupt
FF8C	PWM Emergency Shutdown
FF8A-FF80	Reserverade



Externt genererade avbrott

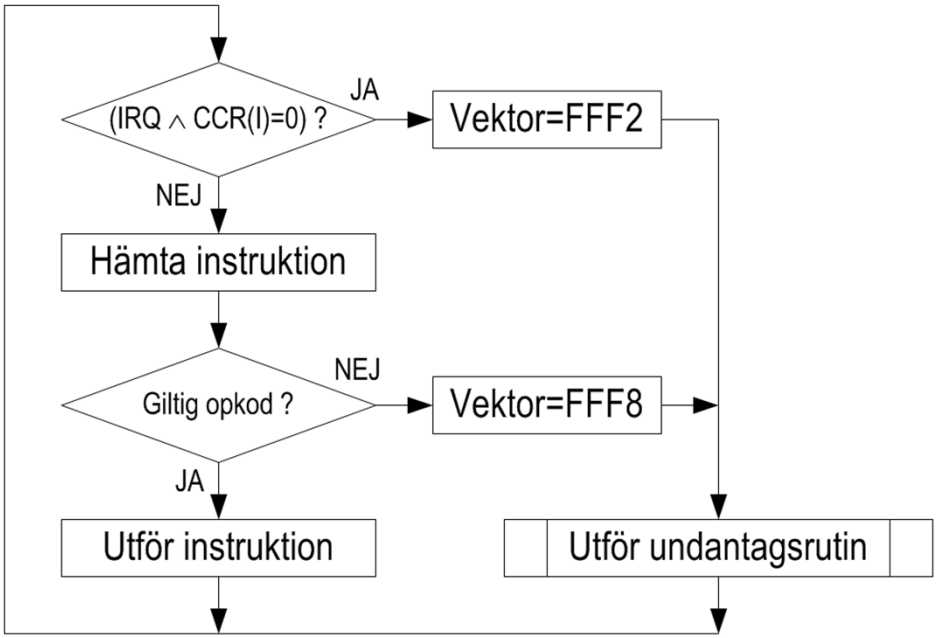


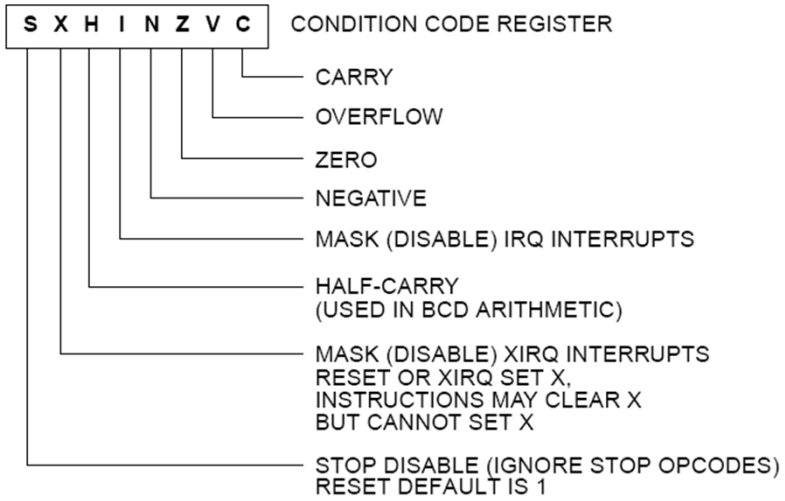


RESET	XIRQ	IRQ
CCR = 1101000	REGISTERS->[SP]	REGISTERS->[SP]
PC=[FFFE,FFFF]	CCR[I]=1	CCR[I]=1
	PC=[FFF4,FFF5]	PC=[FFF2,FFF3]

Adress (hex)	Funktion
FFFE	RESET, Startvektor
FFFC	Clock Monitor Fail
FFFA	COP Watchdog Timeout
FFF8	Illegal Op Code
FFF6	SWI
FFF4	XIRQ
FFF2	IRQ
FF00-FFF0	Enhetsspecifika vektorer, skiljer sig något beroende på olika varianter

Maskering av avbrott





Maskera avbrott:

```
SEI
```

Alternativt

```
ORCC  #%00010000
```

Demaskera avbrott:

```
CLI
```

Alternativt

```
ANDCC   #%11101111
```

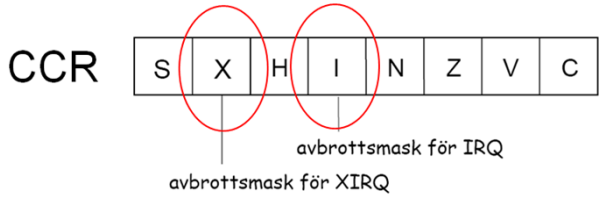
Demaskera X-avbrott:

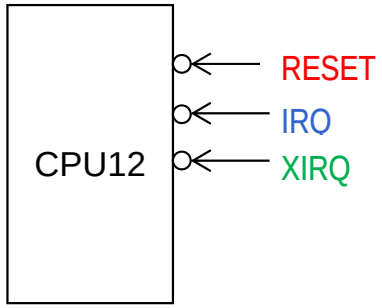
```
ANDCC   #%10111111
```

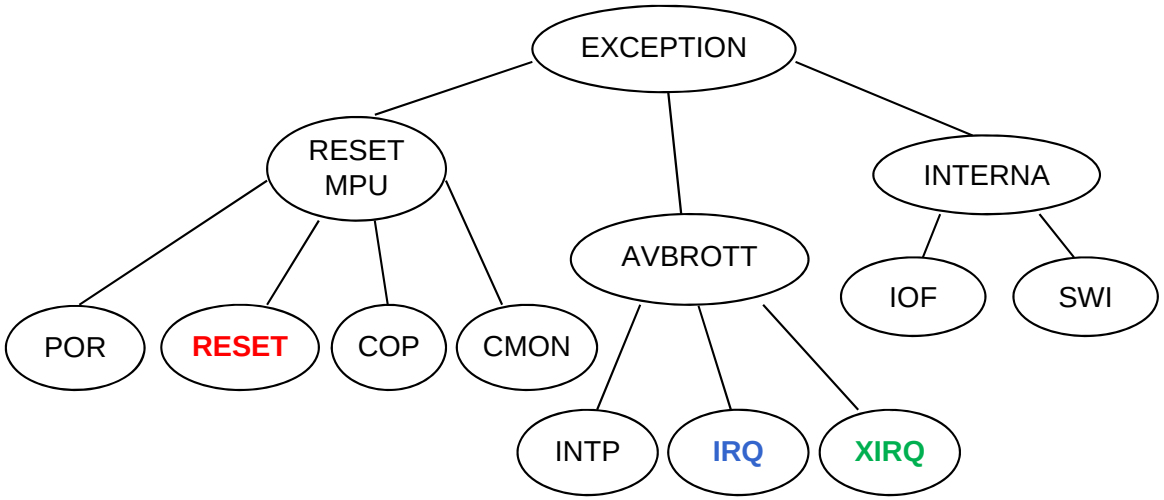
OBS: Kan INTE maskeras
(”Non Maskable Interrupt”)

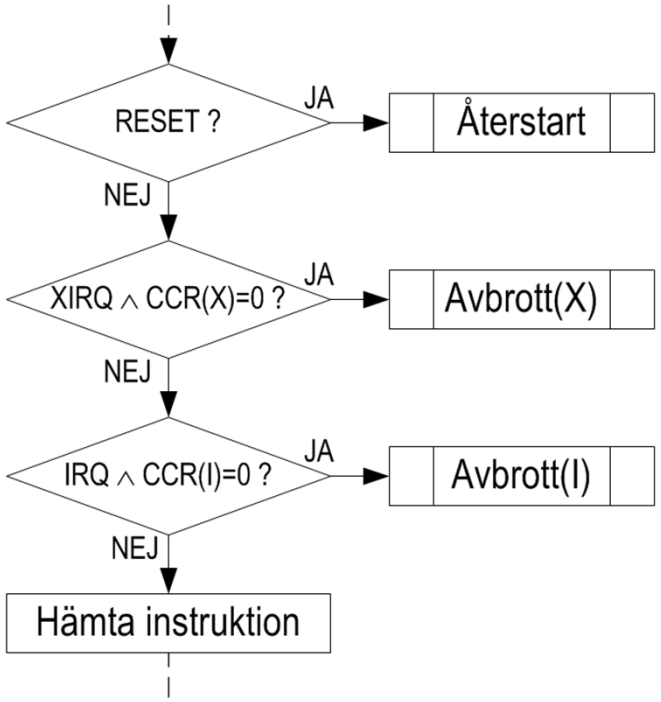
Undantags prioriteter

1. RESET MPU och ”INTERNA” , (alltid)
2. XIRQ, (om X i CCR är 0)
3. IRQ, (om I i CCR är noll)

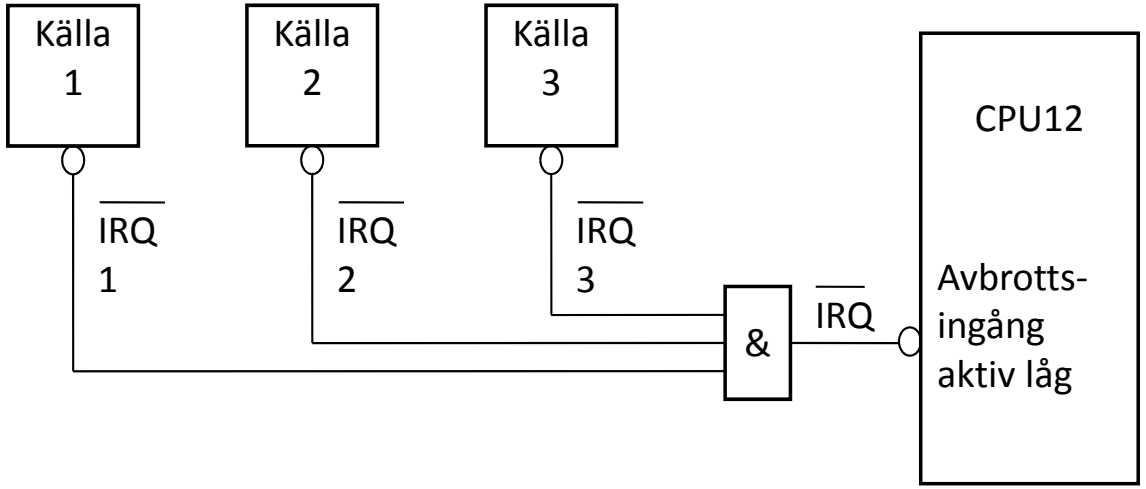


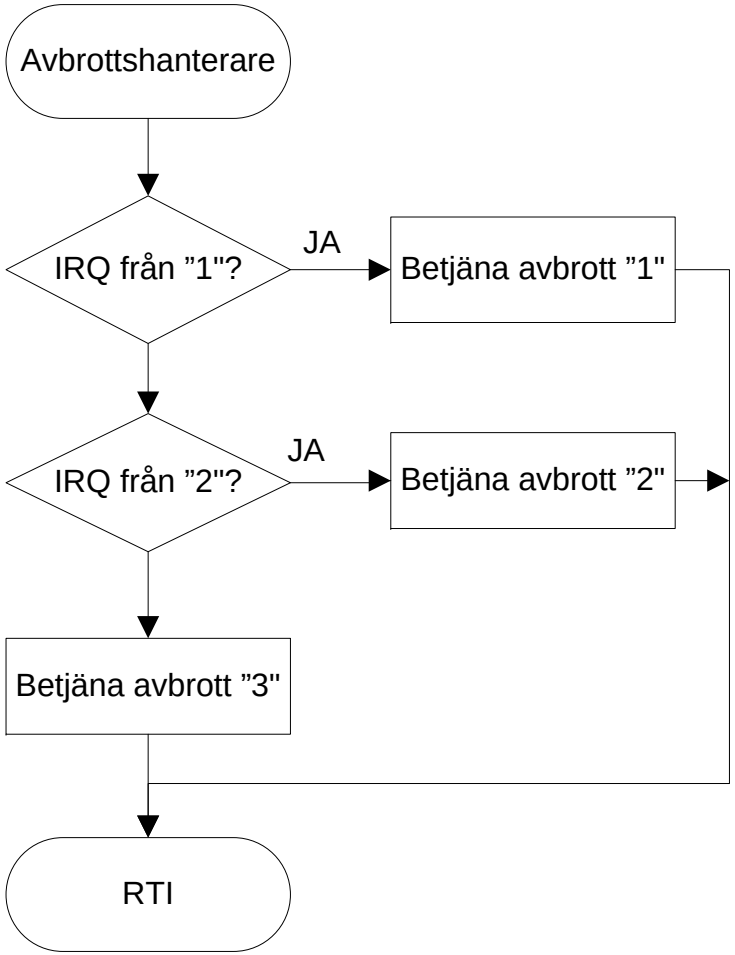






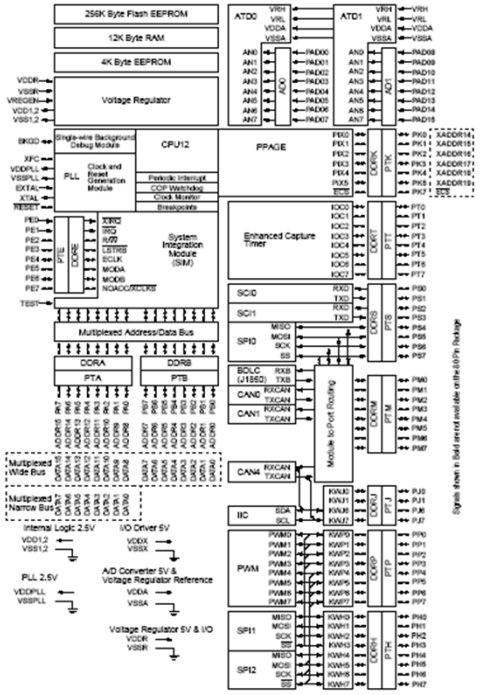
Multipla avbrottskällor





Avbrottskällornas inbördes prioritet bestäms i avbrottshanteraren.

Hårdvarubaserad avbrottsprioritering



För avbrott från interna kretsar bestäms prioriteten av avbrottsvektorns adress.

Ju högre adress, desto högre prioritet.

Högre prioritet

Lägre prioritet

Adress (hex)	Funktion
FFF0	Real Time Interrupt
FFEE	Enhanced Capture Timer channel
FFEC	Enhanced Capture Timer channel 1
FFEA	Enhanced Capture Timer channel 2
FFE8	Enhanced Capture Timer channel 3
FFE6	Enhanced Capture Timer channel 4
FFE4	Enhanced Capture Timer channel 5
FFE2	Enhanced Capture Timer channel 6
FFE0	Enhanced Capture Timer channel 7
FFDE	Enhanced Capture Timer overflow
FFDC	Pulse accumulator A overflow
FFDA	Pulse accumulator input edge
FFD8	SPI0
FFD6	SCI0
FFD4	SCI1
FFD2	ATD0
FFD0	ATD1
FFCE	Port J
FFCC	Port H
FFCA	Modulus Down Counter underflow
FFC8	Pulse Accumulator B Overflow
FFC6	PLL lock
FFC4	CRG Self Clock Mode
FFC2	Används ej (BDLC)
FFC0	IIC Bus
FFBE	SPI1
FFBC	Reserverad
FFBA	EEPROM I-Bit
FFB8	FLASH I-Bit
FFB6	CAN0 wake-up
FFB4	CAN0 errors
FFB2	CAN0 receive
FFB0	CAN0 transmit
…	…
FF96	CAN4 wake-up
FF94	CAN4 errors
FF92	CAN4 receive
FF90	CAN4 transmit
FF8E	Port P Interrupt
FF8C	PWM Emergency Shutdown
FF8A-FF80	Reserverade

”Skelett”, undantagshantering i HCS12-system

```
ORG     $FFF2
FDB     irq                 (FFF2)
FDB     xirq                (FFF4)
FDB     software_interrupt  (FFF6)
FDB     illegal_opcode      (FFF8)
FDB     cop                 (FFFA)
FDB     clock_monitor_fail  (FFFC)
FDB     startup             (FFFE)
```

```
            ...

; Avbrottshanterare
irq:        RTI

xirq:       RTI

software_interrupt:
            RTI

illegal_opcode:
            RTI

cop:        RTI

clock_monitor_fail:
            RTI

; Systemprogram
startup:
            LDS         #TopOfStack
            ...
            ...
            CLI
            JSR         _main
            BRA         startup
```