

Maskinorienterad Programmering 2011/2012

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?

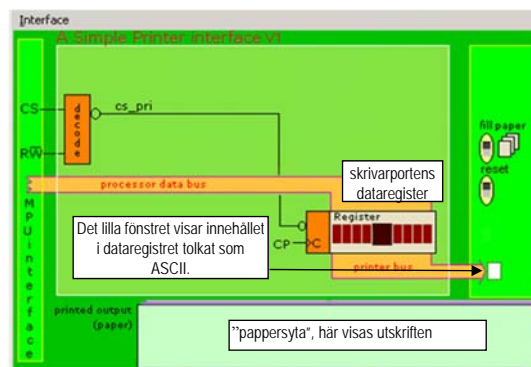
Vi beskriver undantagshantering

Förutsättningar för skrivaranslutningen

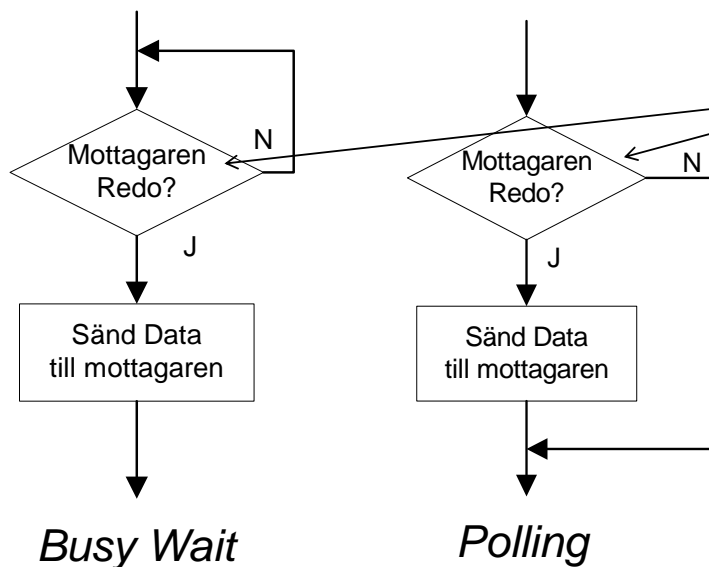
Vår skrivare är från början mycket enkel:

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

Med portdefinition
PRINTER EQU \$0800
 och instruktionerna
 LDA \$30
 STAA PRINTER
 överförs hexadecimala värdet 30 till skrivaren

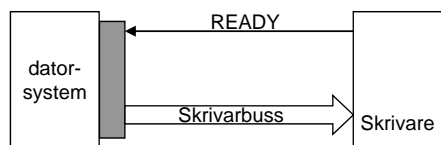


Villkorlig överföring

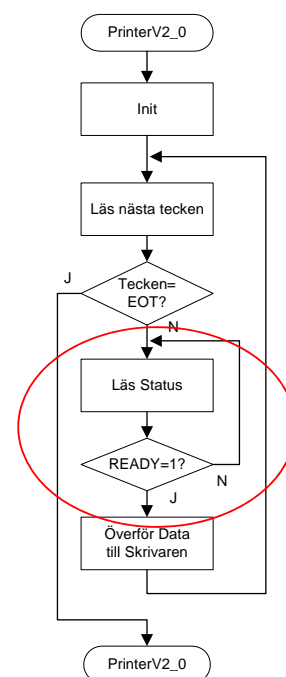
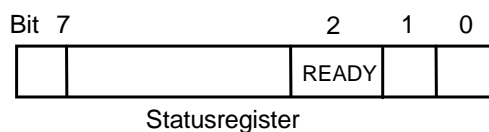
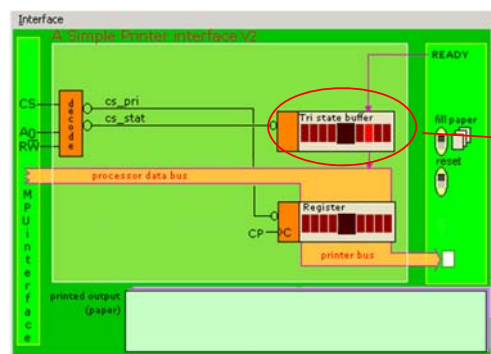


Statustest,
kräver
asynkront
gränssnitt...

Gränssnitt, version 2



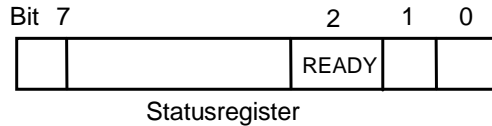
- READY = 1 (Hög nivå) indikerar att skrivaren är klar att ta emot ett nytt tecken.
- READY = 0 (Låg nivå) indikerar att skrivaren är upptagen med att skriva ut ett tecken.



"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    LoopForReady
                BRA    Loop
Exit:          NOP
                BRA    Exit

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

Speciella instruktioner....

```

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

"Branch if bit(s) is clear"

```

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

```

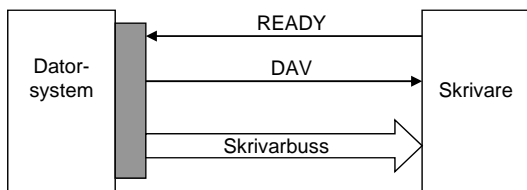
...
PSHB
LDAB    $0802
ORAB    #2
STAB    $802
PULB
..
    
```

"Set bit(s) in memory"

```

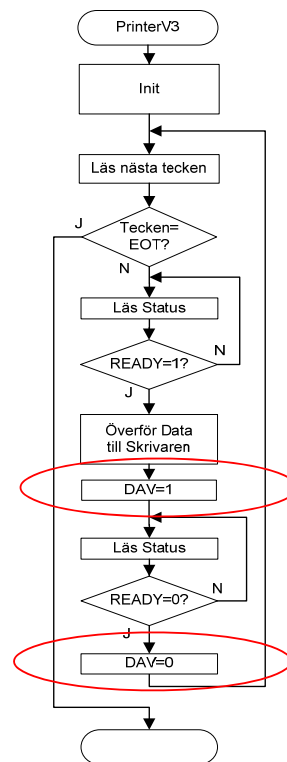
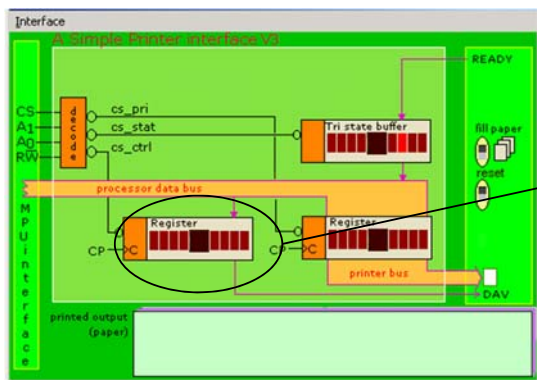
...
BSET    $0802,#2 ; ettställ bit 1
    ...
    
```

Gränssnitt, version 3

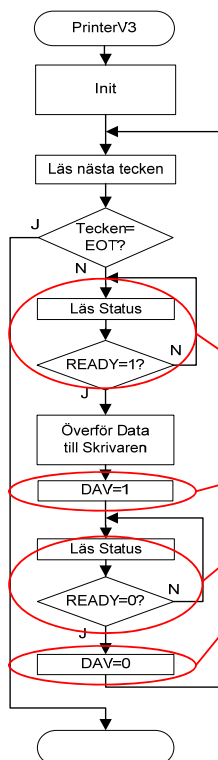


DAV = 1 (Hög nivå) indikerar för skrivaren att giltigt tecken finns att hämta på skrivarbussen.

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
EOT EQU       4
    
```

```

ORG    $1000
LDX    #Text
Loop:  LDAA  1,X+
      CMPA  #0
      BEQ   Stop
    
```

```

Ready:  BRCLR  PSTATUS,#4,Ready
      STAA  PRINTER
      BSET  PCONTROL,#2
NotReady: BRSET  PSTATUS,#4,NotReady
      BCLR  PCONTROL,#2
      BRA   Loop
Stop:   NOP
      BRA  Stop
    
```

```

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

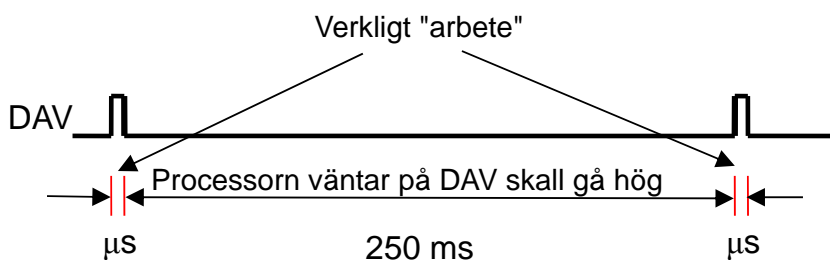
Resultat

Klarar nu situationen att centralenheten arbetar snabbare än skrivaren.

Klarar nu situationen då centralenheten är långsammare än skrivaren.

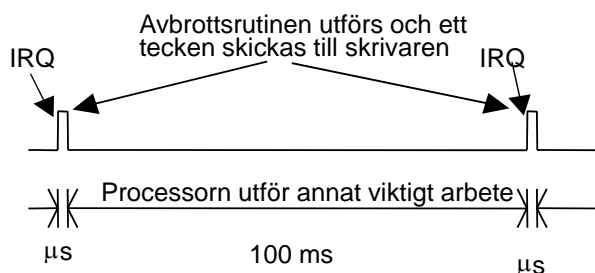
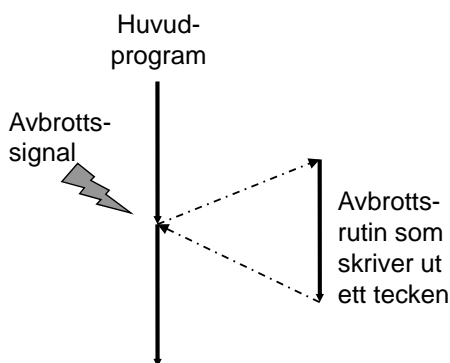
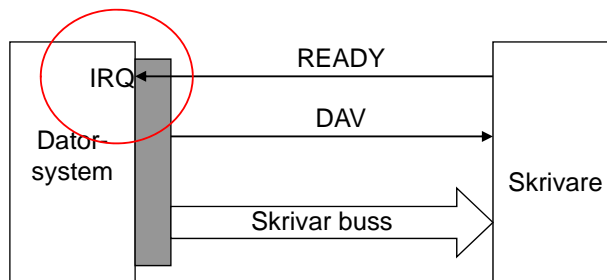
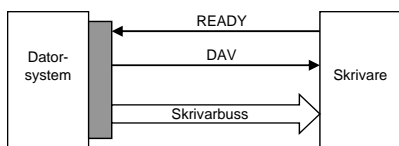
Klarar nu situationen med att få skrivaren att stoppa då sista tecknet skrivits ut

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

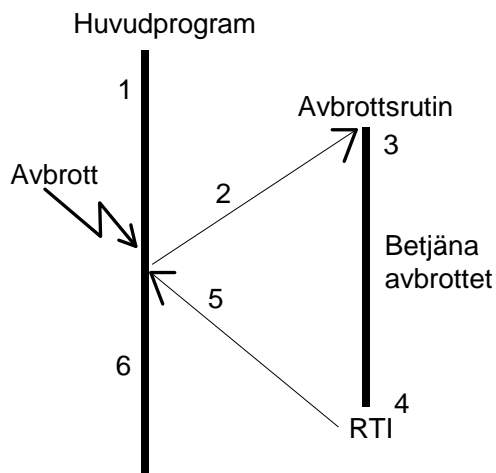


Introduktion till "Undantagshantering"

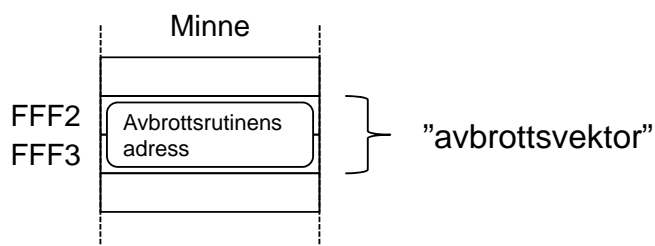
Interrupt ReQuest (IRQ), begäran om avbrott...



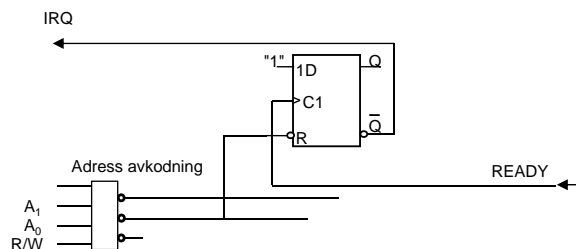
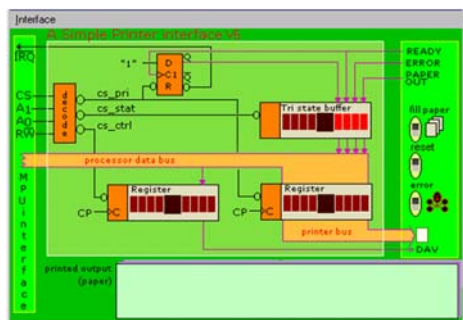
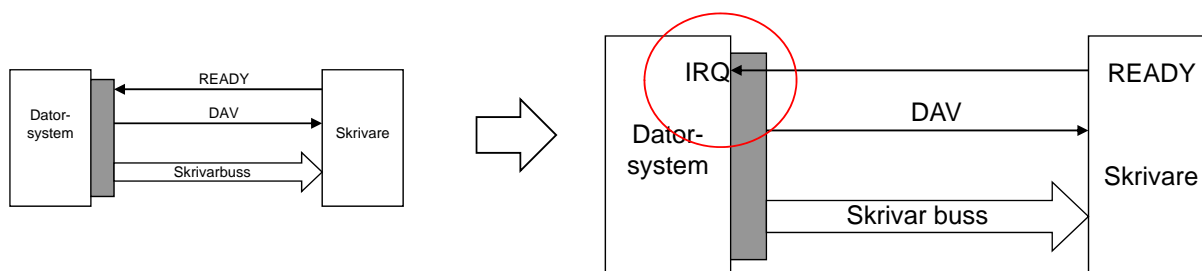
Avbrottshantering



- 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

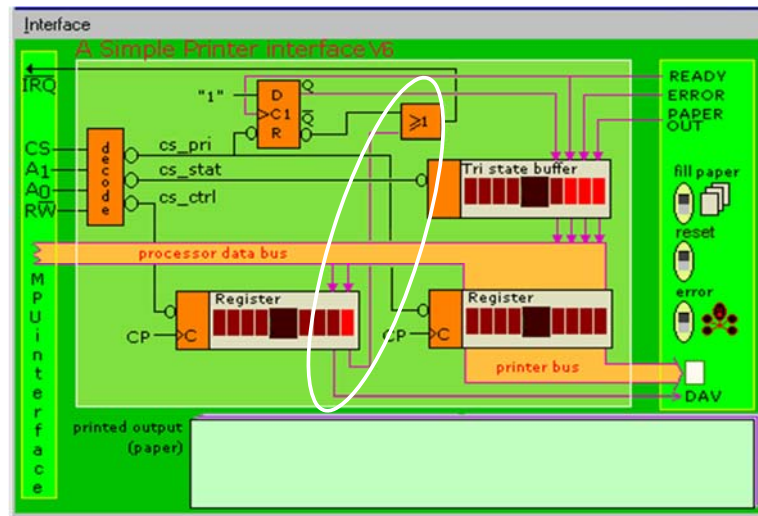


Denna lösning genererar ALLTID avbrott då skrivarens teckenbuffert är tom....

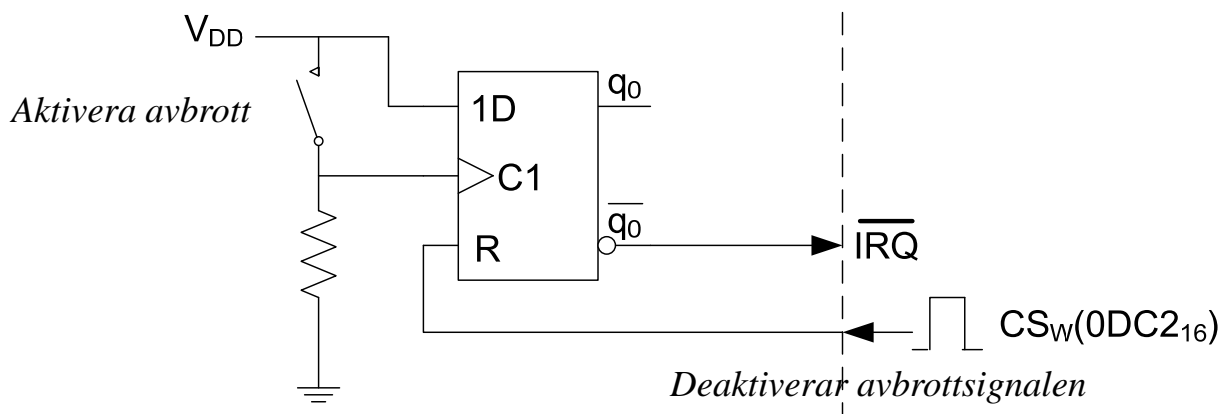
Skrivarport, Version 6

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

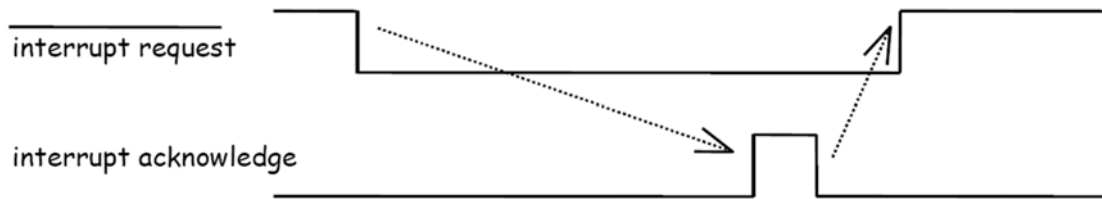
Lämnas som självverksamhet...



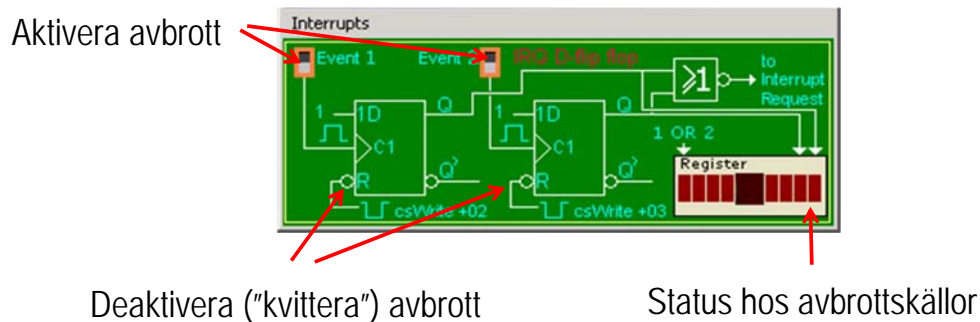
Avbrottsvippa



Kvittering av avbrott "Interrupt Acknowledge"



EXEMPEL: (jfr: laborationskort ML19)



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
    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
    
```

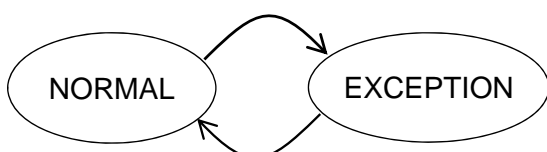
IrqStat	EQU	\$0D00
IrqRes1	EQU	\$0D02
IrqRes2	EQU	\$0D03

Exekveringstillstånd

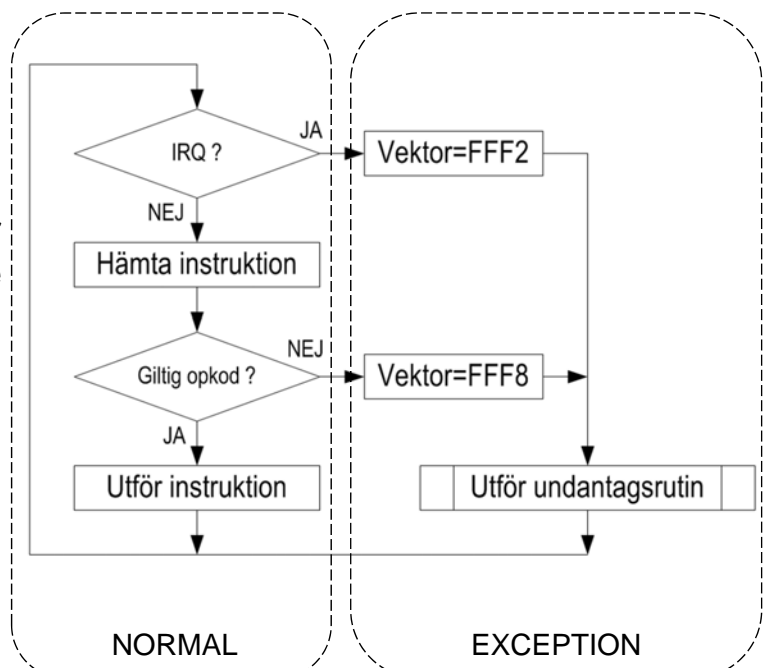
Processorn befinner sig alltid i något av tillstånden:

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

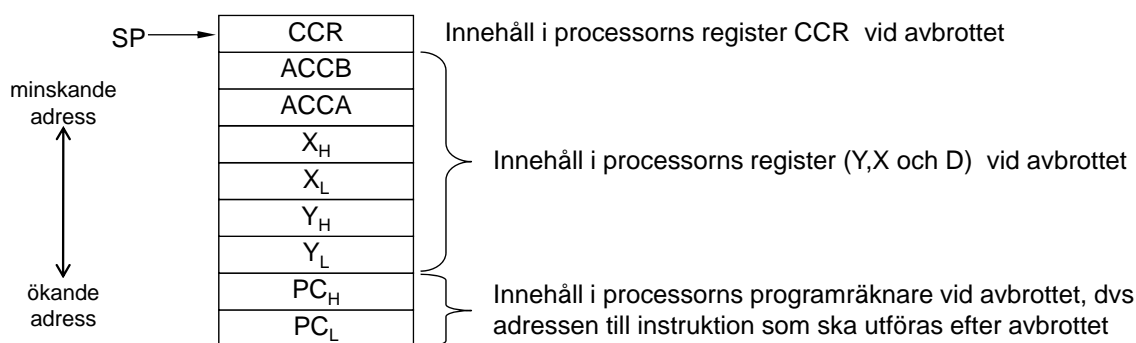
Return from interrupt

RTI

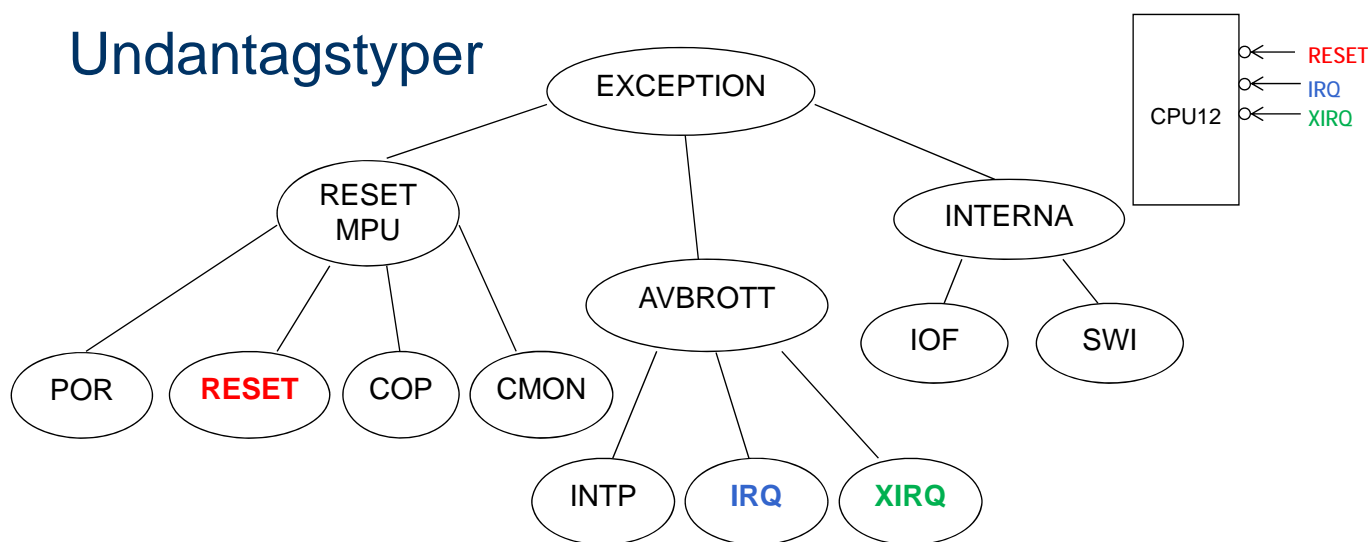
Operation:
 $(M(SP)) \Rightarrow CCR; (SP) + \$0001 \Rightarrow SP$
 $(M_1(SP); M_2(SP + 1)) \Rightarrow B; A; (SP) + \$0002 \Rightarrow SP$
 $(M_3(SP); M_4(SP + 1)) \Rightarrow X_H; X_L; (SP) + \$0004 \Rightarrow SP$
 $(M_5(SP); M_6(SP + 1)) \Rightarrow PC_H; PC_L; (SP) - \$0002 \Rightarrow SP$
 $(M_7(SP); M_8(SP + 1)) \Rightarrow Y_H; Y_L; (SP) + \$0004 \Rightarrow SP$

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.

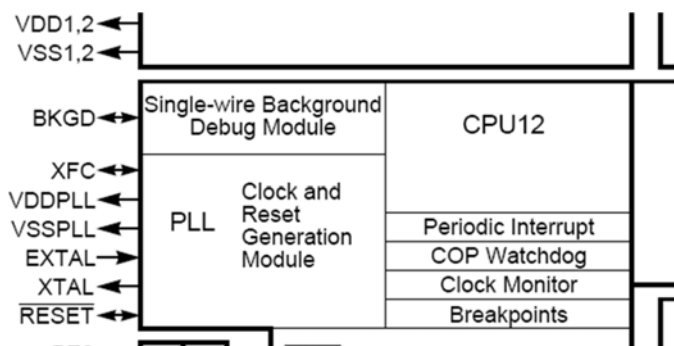
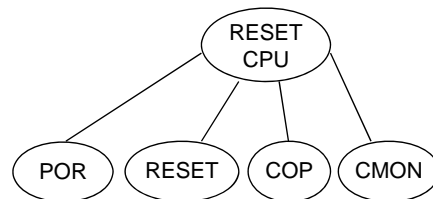
Fatala fel, kräver RESET av CPU

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

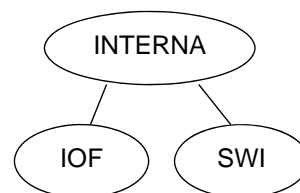
Om processorn avkodar en otillåten operationskod kallas detta *Illegal Opcode Fetch* (IOF).

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

```

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

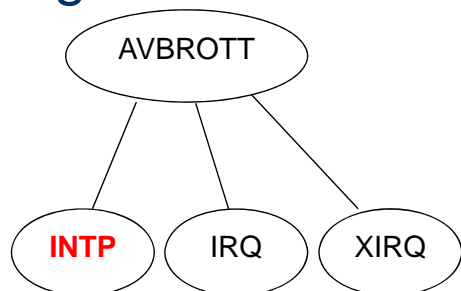
```

SWI_hantering:
        CLRA
        NOP
        RTI

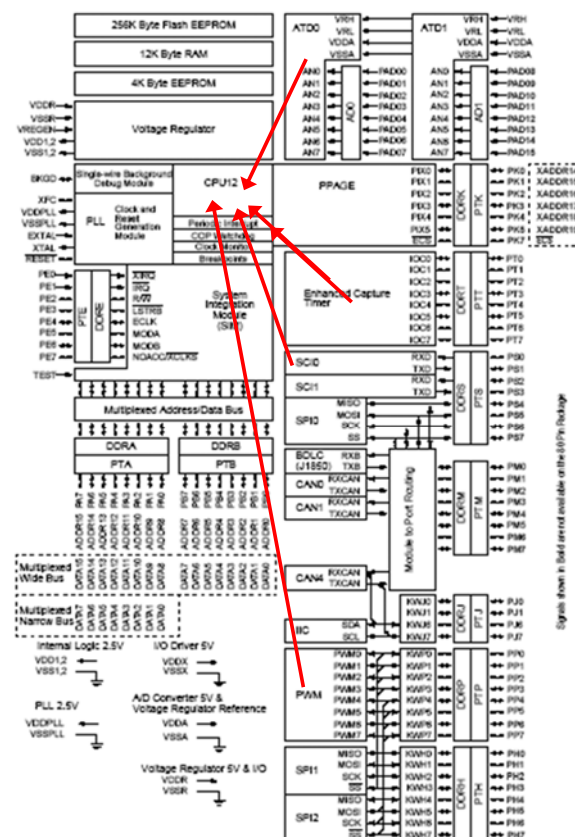
        ORG    $FFF6
        FDB   SWI_hantering
    
```

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

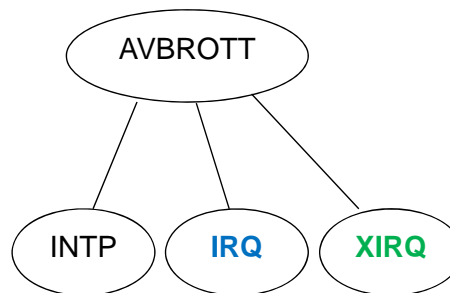
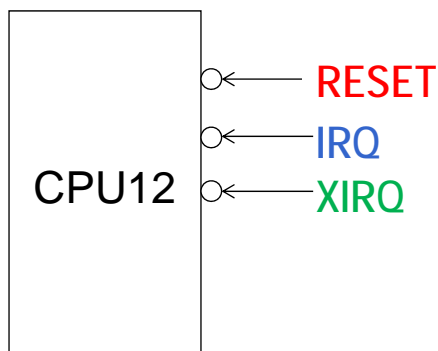
Internt genererade avbrott



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



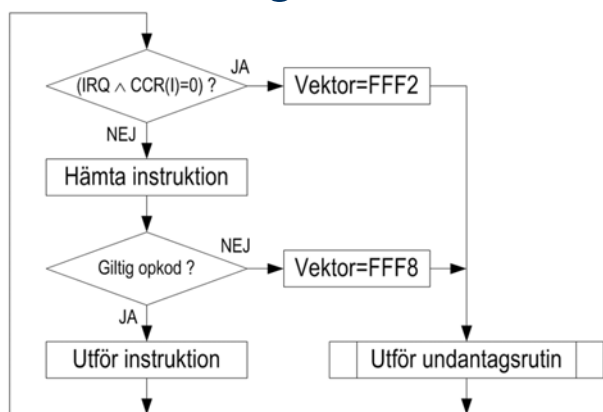
Externt genererade avbrott



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

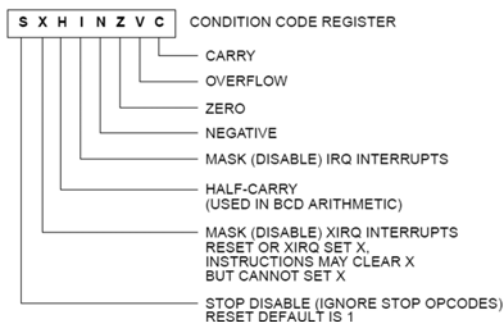
ORCC #00010000

Demaskera avbrott:

CLI

Alternativt

ANDCC #11101111



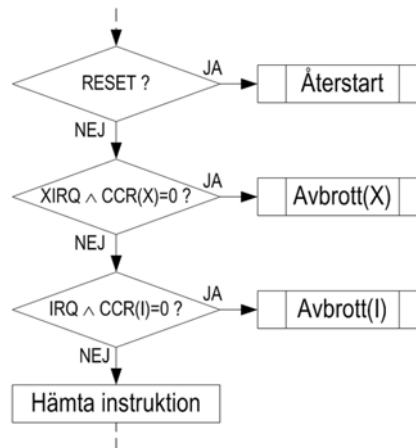
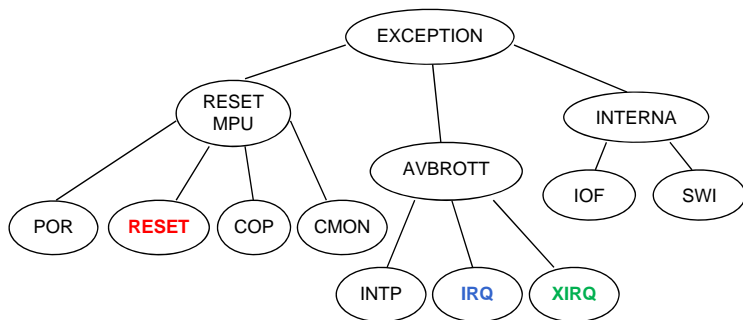
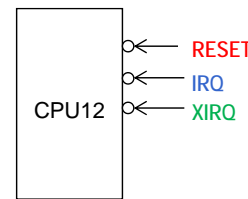
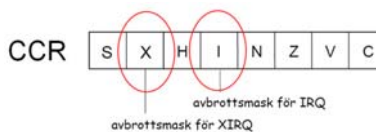
Demaskera X-avbrott:

ANDCC #10111111

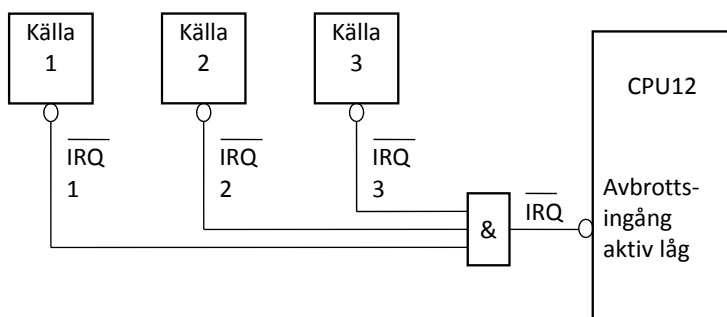
OBS: Kan INTE maskeras ("Non Maskable Interrupt")

Undantags prioriteter

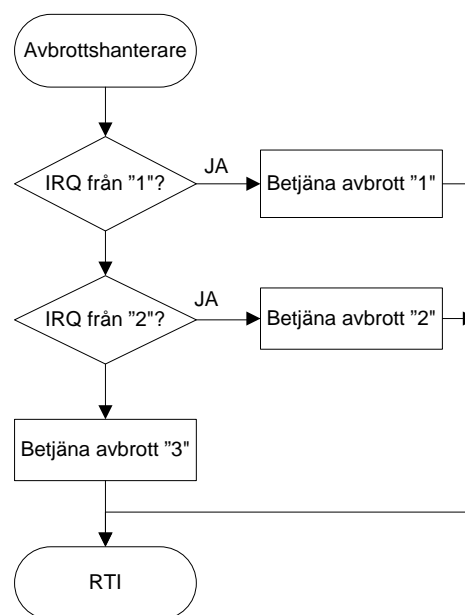
1. RESET MPU och "INTERNA" , (alltid)
2. XIRQ, (om X i CCR är 0)
3. IRQ, (om I i CCR är noll)



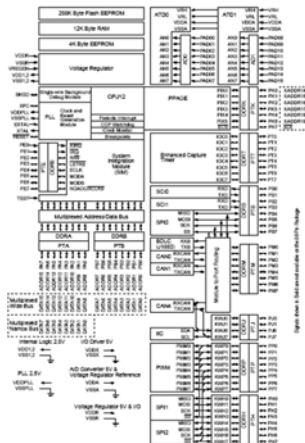
Multipla avbrottskällor



Avbrottskällornas inbördes prioritet bestäms i avbrottshanteraren.



Hårdvarubaserad avbrottsprioritering



För avbrott från interna kretsar bestäms prioriteterna av avbrottsvektorns adress.

Ju högre adress, desto högre prioritet.

Högre prioritet

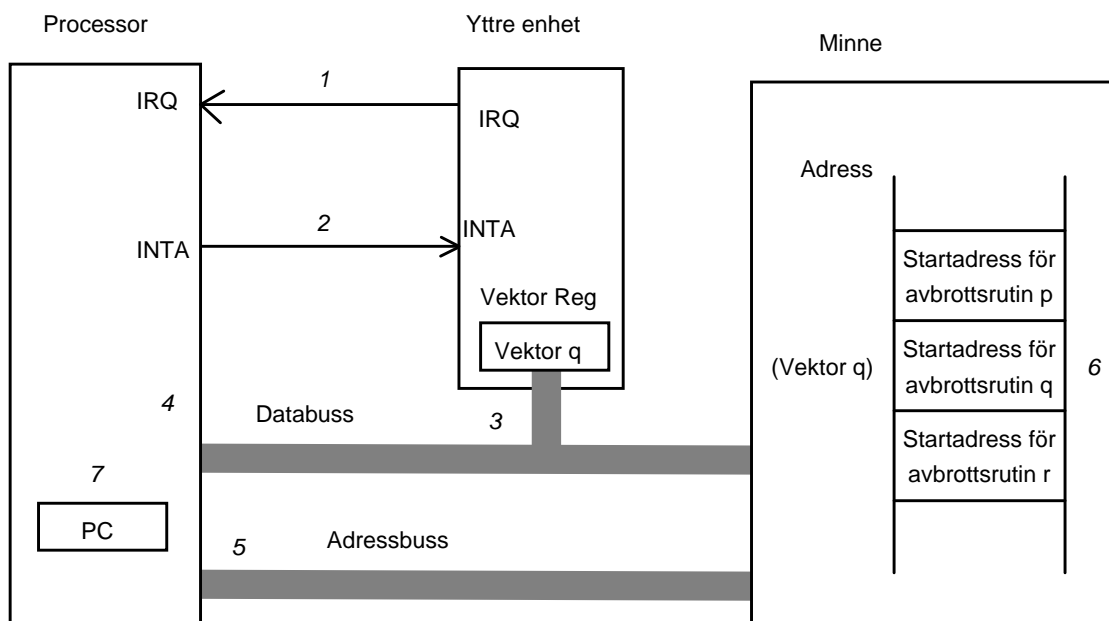


Lägre prioritet

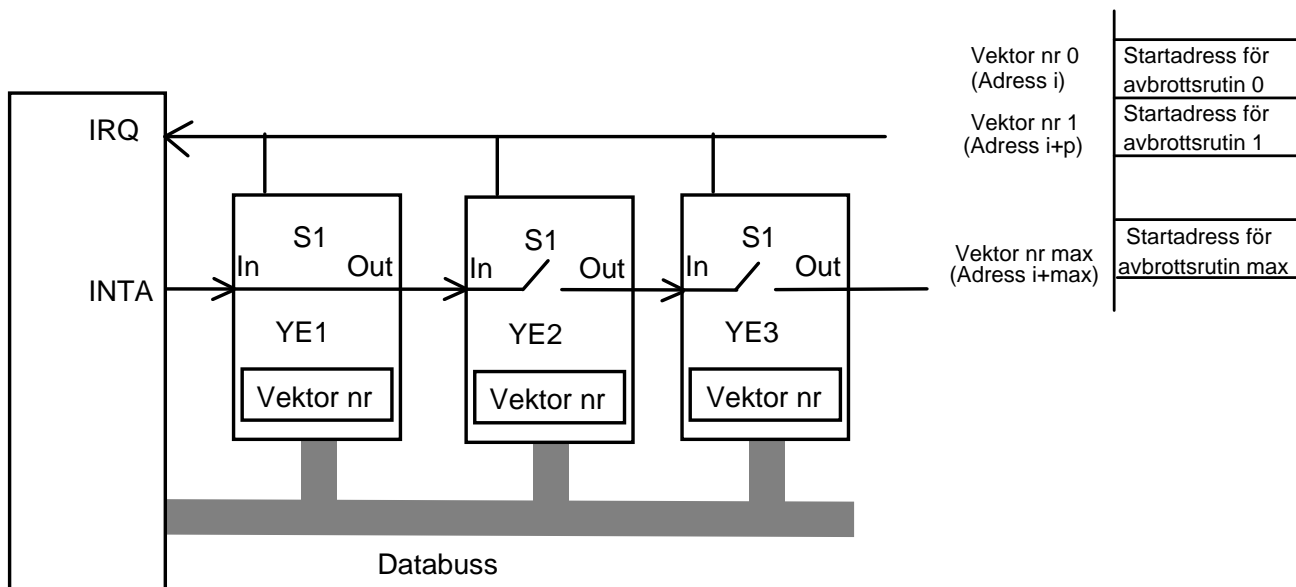
Adress (hex)	Funktion
FFF0	Real Time Interrupt
FFEE	Enhanced Capture Timer channel
FFEC	Enhanced Capture Timer channel 1
FFEA	Enhanced Capture Timer channel 2
FFE8	Enhanced Capture Timer channel 3
FFE6	Enhanced Capture Timer channel 4
FFE4	Enhanced Capture Timer channel 5
FFE2	Enhanced Capture Timer 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

Vektoravbrott

Somliga periferekretsar konstrueras för att tillhandahålla avbrottsvektor (kan ej anslutas till HCS12)



Prioritet vid vektoravbrott (Daisy chain)



EXEMPEL, "ColdFire" (MC68x00)

Vektor nr	Adress (offset) (hex)	Funktion
0	000	Initial stackpekare
1	004	Initial programräknare
2	008	Access Error (ex: referens till adress där minne/periferikrets ej finns)
3	00C	Adress Error (ex: referens till udda adress med <i>word</i> operand)
4	010	Illegal instruktion (icke-definierad operationskod)
5	014	Division med 0
6,7	018, 01C	Reserverade
8	020	Privilege Violation, försök att utföra <i>supervisor</i> -instruktion i <i>user mode</i>
9	024	Trace, en-instruktions exekvering
10	028	Line 1010, reserverad operationskod
11	02C	Line 1111, reserverad operationskod
12	030	Non-PC breakpoint debug interrupt
13	034	PC breakpoint debug interrupt
14	038	Format error
15	03C	Avbrott från enhet som ej tillhandahållit avbrottsnummer
16-23	040-05F	Reserverade vektorer
24	060	Icke-identifierat avbrott
25	064	Autovektor avbrottsnivå 1
26	068	Autovektor avbrottsnivå 2
27	06C	Autovektor avbrottsnivå 3
28	070	Autovektor avbrottsnivå 4
29	074	Autovektor avbrottsnivå 5
30	078	Autovektor avbrottsnivå 6
31	07C	Autovektor avbrottsnivå 7
32-47	080-0BF	Trap vektor för instruktionen TRAP #<vektor_nummer>
48-63	0C0-0FF	Undantag vid flyttalshantering, Reserverade vektorer
64-255	100-3FF	Användardefinierade vektorer

← RESET vektor

← "Undantag" genererade av program

← Autovektor avbrott

← Vektoravbrott