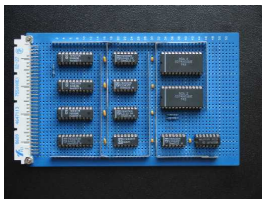
# Digital- och datorteknik



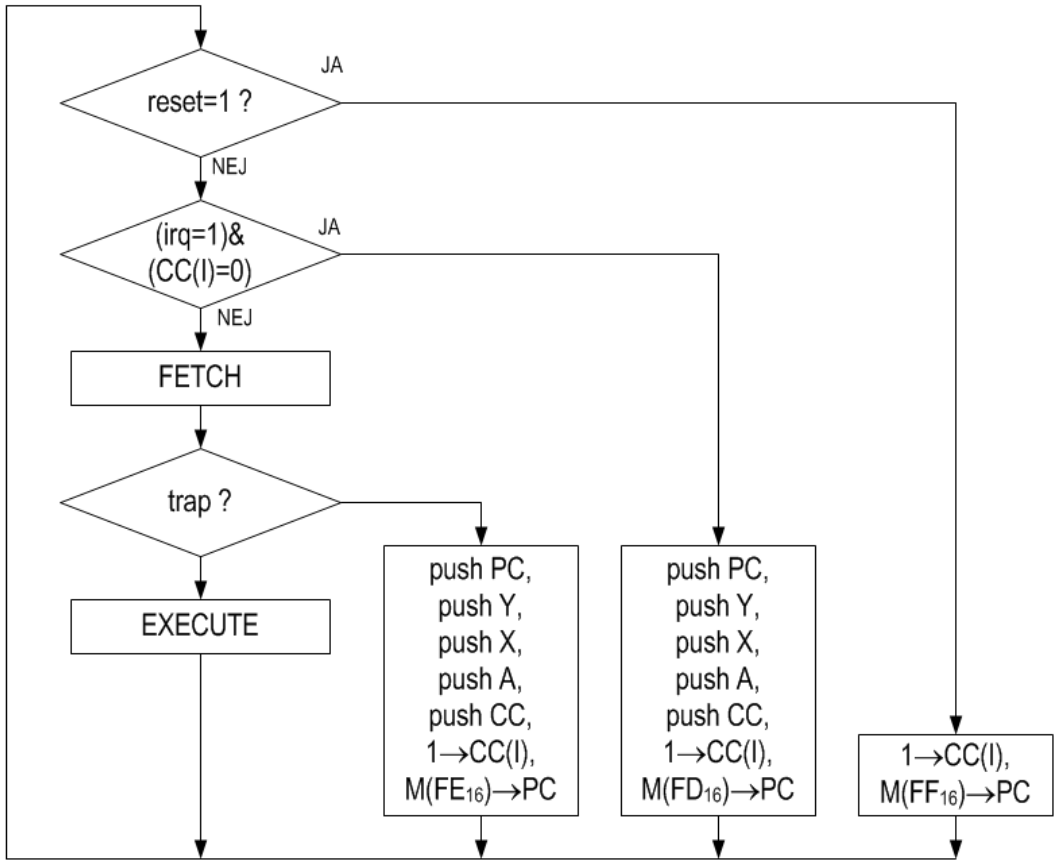
## Föreläsning #20

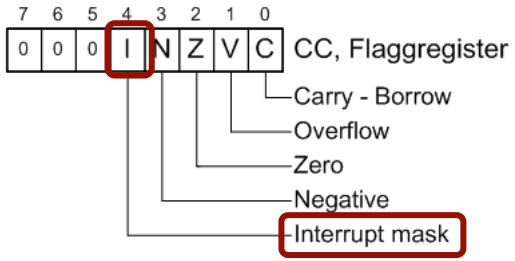
Biträdande professor Jan Jonsson

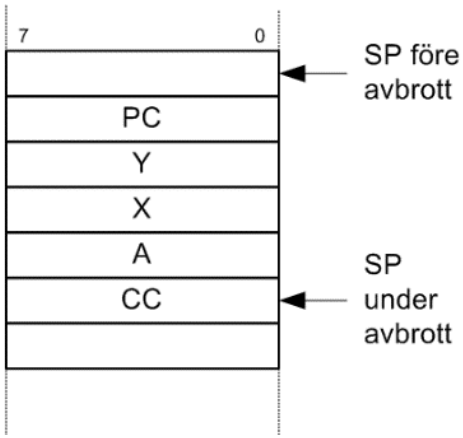
Institutionen för data- och informationsteknik
Chalmers tekniska högskola

# Undantagshantering

## Undantagshantering – FLIS-processorn



# Undantagshantering

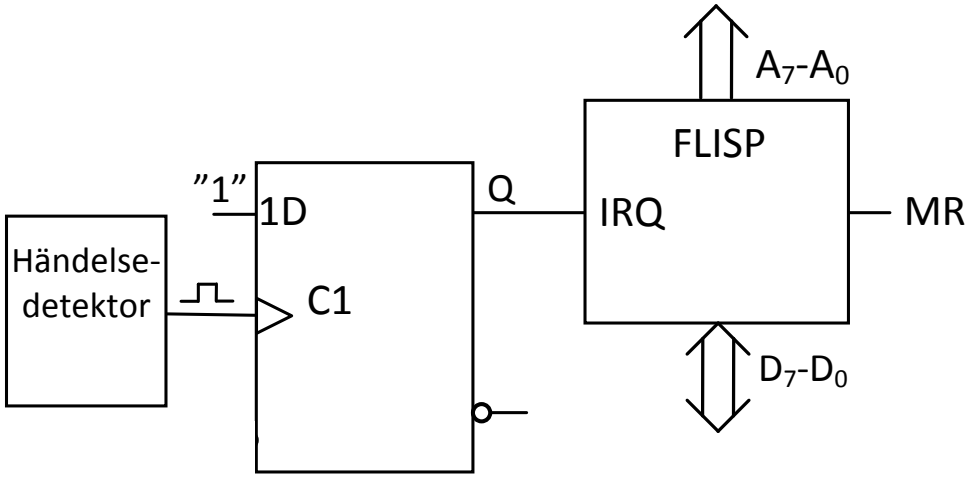
## Avbrott – FLIS-processorn

- När signalen på IRQ-ingången blir aktiv exekverar processorn färdigt den instruktion den håller på med.
- Om avbrottssystemet är aktiverat, d v s I-flaggan är nollställd, accepterar processorn avbrottet. I annat fall fortsätter programexekveringen som vanligt.
- Om avbrottsbegäran accepteras sparas innehållen i de interna registren i ordningen PC, Y, X, A och CC på stacken. Därefter ettställs I-flaggan för att utestänga nya avbrott.
- Processorn läser innehållet i datorsystemets IRQ-vektor på minnesadress $FD_{16}$, och placerar det i register PC.
- Processorn hämtar första instruktionen i hanteringsrutinen.

# Undantagshantering

## Detektering av extern avbrottssignal

Med hjälp av en D-vippa kan såväl kortvariga som långvariga händelsesignaler registreras av processorn.



# Undantagshantering

## Kodning av hanteringsrutiner

När hanteringsrutinen börjar exekveras vet man att händelsen som orsakar avbrott har inträffat. Hanteringsrutinen skall därför utföra den kod som är förknippat med händelsen och sedan återvända till det avbrutna programmet med återställda register.

- Hanteringsrutinen skall byggas på samma sätt som en vanlig subrutin, med följande viktiga skillnad: hanteringsrutinen ska avslutas med instruktionen RTI (”return från interrupt”), som återställer innehållen i samtliga processorregister.
- Vid avbrott p g a extern händelse måste den aktiva signal som orsakade det pågående avbrottet ha försvunnit innan återhoppet. Annars kommer ju ett nytt avbrott att genereras för samma externa händelse efter återhoppet.

# Undantagshantering

## Kodning av hanteringsrutiner

Återhopp från avbrott görs med instruktionen RTI.

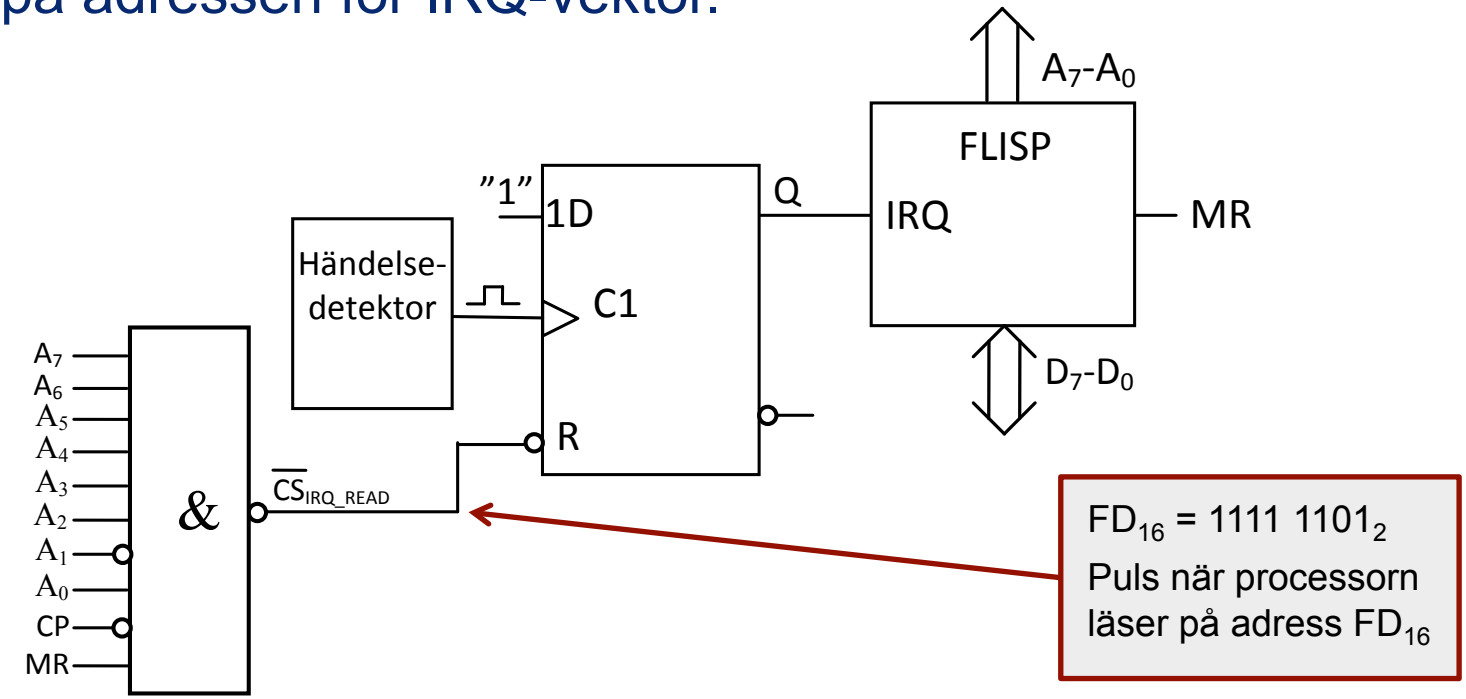
Notera att avbrottssystemet automatiskt aktiveras då de gamla registerinnehållen i processorn återställs vid återhopp från hanteringsrutinen eftersom det gamla CC-innehållet, som hämtas från stacken, innehåller en nolla i I-biten.

| Instruktion | | Adressering | | | Operations-beskrivning | Flaggor | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Operation | Beteckning | Inherent | | | | 4 | 3 | 2 | 1 | 0 |
| | | OP | # | ~ | | I | N | Z | V | C |
| Return from interrupt | RTI | 44 | 1 | 6 | M(SP) → CC; SP+1 → SP | Δ | Δ | Δ | Δ | Δ |
| | | | | | M(SP) → A; SP+1 → SP | | | | | |
| | | | | | M(SP) → X; SP+1 → SP | | | | | |
| | | | | | M(SP) → Y; SP+1 → SP | | | | | |
| | | | | | M(SP) → PC; SP+1 → SP | | | | | |

# Undantagshantering

## Kodning av hanteringsrutiner

Nollställning av den signal som genererat avbrott p g a extern händelse kan ske genom att läsa på adressen för IRQ-vektor.



# Undantagshantering

## Demonstrationsexempel 1 – tongenerator (fortsättning)

Du skall skriva ett assemblerprogram som genererar en ton (fyrkantvåg) med frekvensen 250 Hz på bit $b_0$ på utport $FB_{16}$. Tonen skall genereras under ett tidsintervall av 180 ms.

a) Skriv programmet så att alla nödvändiga fördröjningar erhålls genom att maskinkod exekveras. Antag att FLIS-processorn har klockfrekvensen 1 MHz.

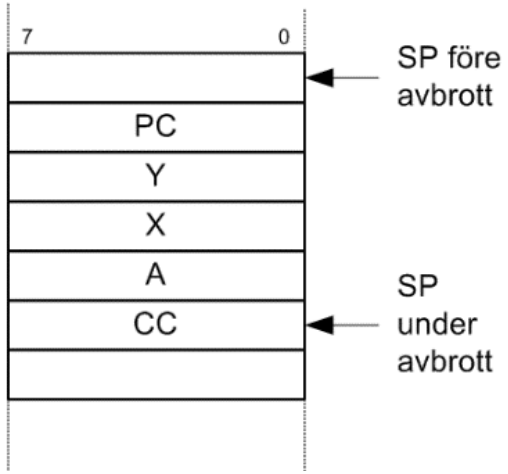
b) Modifiera programmet så att det istället utnyttjar att en extern pulsgenerator är ansluten till IRQ-ingången. Antag att pulsgeneratorn levererar positiva klockflanker en gång per millisekund.

# Undantagshantering

## Stackinnehåll betraktat som programkontext

Att man sparar alla, för programmeraren synliga, register på stacken i samband med att ett undantag hanteras har hittills betraktats som en nödvändighet om undantagshanteringen skall kunna utföras transparent och utan bieffekter för det program som fått sin exekvering avbruten.

Informationen som sparas kan dock ses som något ”större” och mer generellt: innehållet i de register som sparats på stacken samt tillhörande adress till TOS kan ses som ett programkontext som definierar det program som blev avbrutet.



# Undantagshantering

## Stackinnehåll betraktat som programkontext

Givet ett giltigt programkontext kan alltså ett exekveringen av det program som beskrivs av givet kontext återupptas utan att programmet upplever att det har avbrutits.

Detta utgör grunden till det som brukar kallas pseudoparallell exekvering av program, d v s att man kan exekvera två eller fler program på samma dator på ett sådant sätt att de enskilda programmen inte upplever att de tidsdelar datorns resurser.

Genom att i förväg konstruera en uppsättning programkontext kan man alltså ladda två eller flera program i primärminnet, och sedan låta dem exekvera oberoende av varandra.

# Undantagshantering

## Stackinnehåll betraktat som programkontext

Givet ett giltigt programkontext kan alltså ett exekveringen av det program som beskrivs av givet kontext återupptas utan att programmet upplever att det har avbrutits.

Detta utgör grunden till det som brukar kallas pseudoparallell exekvering av program, d v s att man kan exekvera två eller fler program på samma dator på ett sådant sätt att de enskilda programmen inte upplever att de tidsdelar datorns resurser.

Byte av program sker transparent genom att avbrott genereras periodiskt via en extern signal, och ett byte av programkontext kan ske vid avbrottshanteringen.

# Undantagshantering

## Demonstrationsexempel 2 – pseudoparallelism

Du skall skriva två assemblerprogram, Prog0 och Prog1, som skall köras pseudoparallellt på samma dator.

Program Prog0 skall regelbundet öka värdet på utport $FB_{16.}$

Program Prog1 skall regelbundet minska värdet på utport $FC_{16.}$

Körningen av de två programmen skall ske på ett sådant sätt att man upplever att båda programmen körs, men med halva hastigheten jämfört med om de körde ensamma.

En extern pulsgenerator, som levererar positiva klockflanker med frekvensen 100 Hz, har anslutits till IRQ-ingången.