Assemblerprogrammering – del 3

Dagens föreläsning behandlar:

Kompendiet kapitel 9 och 10.4
Arbetsboken kapitel 16

Ur innehållet:

- ❑ Modularisering, subrutiner och strukturerad programutveckling (flödesdiagram)
- ❑ Lokala variabler och deklarationers ”synlighet”
- ❑ Undantagshantering

Modularisering - subrutiner

- Vanligt förekommande operationer med uttryckssemantik kan implementeras som en generell funktion – *subrutin*.
- Exempelvis finns inte operationen ”multiplikation” hos FLISP, men vi kan skriva en subrutin som gör jobbet.

```
; subrutin mul
; multiplicerar två 8-bitars tal (utan tecken)
; returnerar 8-bitars resultat
; Indata
;     register A: operand 1
;     register Y: operand 2
; Utdata
;     register A: produkten (operand 1 * operand 2)
mul:
      PSHA                ; spara operand 1 som multiplikand
mul_loop:
      CMPY   #1           ; färdig då multiplikator=1
      BEQ    mul_exit
      ADDA   0,SP         ; addera multiplikanden
      LEAY   -1,Y         ; minska räknare (multiplikator)
      BRA    mul_loop
mul_exit:
      LEASP  1,SP         ; återställ stack
      RTS
```

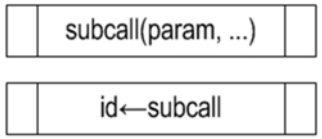
```
unsigned char      prod,op1, op2;
prod = op1 * op2;
```

kodas som:

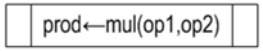
```
      LDA    op1
      LDY    op2
      JSR    mul
      STA    prod
```

Instruktionerna JSR, BSR och RTS är speciellt avsedda för detta.

Flödessymbol för subrutiner

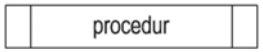


Subrutinanrop.
Med parametrar, symbolisk representation av eventuella aktuella parametrar som skickas med subrutinanropet.
Med returvärde, tilldelningsoperator placeras framför den anropade subrutinen.




```
...
LDA  op1
LDY  op2
JSR  mul
STA  prod
...
```

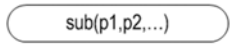
EXEMPEL:
Anrop av subrutin med både parametrar och returvärde

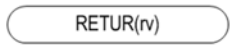



```
...
JSR  procedur
...
```

EXEMPEL:
Anrop av parameterlös subrutin utan returvärde

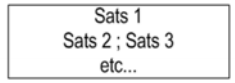
Sammanfattning av symboler för flödesdiagram





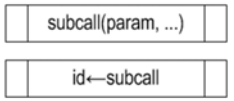
Inträde i och utträde ur subrutiner.
Inträde, typiskt med subrutinens namn och symbolisk representation av eventuella parametrar då sådana finns.
Utträde, ("RETUR") typiskt med angivande av returvärde (rv) om sådant finns.

Engångsinitieringar.
Placeras typiskt i omedelbar anslutning till en inträdessymbol



Exekveringsblock.
En eller flera satser som ordnats och exekveras sekvensiellt.

Villkorsblock.
Ett uttryck testas, utfallet kan vara sant eller falskt och exekveringsväg väljs därefter.



Subrutinanrop.
Med parametrar, symbolisk representation av eventuella aktuella parametrar som skickas med subrutinanropet.
Med returvärde, tilldelningsoperator placeras framför den anropade subrutinen.

Användning av flödesdiagram

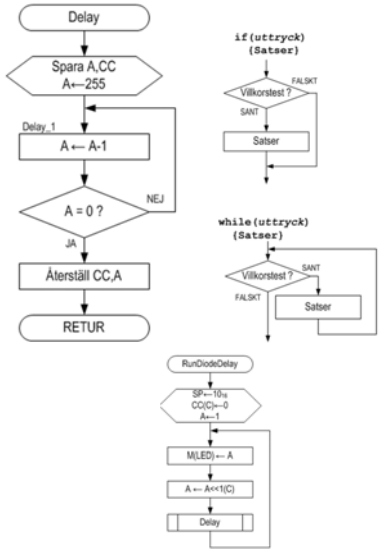
Vi kan nu beskriva godtyckliga algoritmer med flödesdiagram

dvs.

- Grundläggande operationer kan beskrivas
- Programflöden med operationer kan beskrivas
- Programflöden kan organiseras i subrutiner.

Återstår:

- ”Datainkapsling”: synlighet och åtkomst.



Statisk minnesallokering

Vi har tidigare beskrivit *globala variabler* potentiellt åtkomligt från alla delar av programmet.

- Sårbart för ”sidoeffekter”
- Ockuperar onödigt mycket minne
- Rekursion kan inte användas

```
char  sum,op1,op2,op3;
```

motsvarande assemblerdeklarationer blir:

```
      ORG   0
sum   RMB   1
op1   RMB   1
op2   RMB   1
op3   RMB   1
```

Tilldelningen:

```
      sum = op1+op2-op3;
```

kan då kodas:

```
      LDA   op1
      ADDA  op2
      SUBA  op3
      STA   sum
```

Exempel 9.12 Enkla datadeklarationer, reservering av minnesutrymme

En variabeldeklaration:

```
   unsigned char counter;
```

anger att en *byte* ska reserveras för symbolen `counter`, då den används i jämförelseoperationer ska den betraktas som ett 8-bitars tal utan inbyggt tecken. I assemblerspråk motsvarar deklarationen:

```
counter:   RMB   1
```

Deklarationen:

```
   signed char summa;
```

anger också att en byte ska reserveras, denna gång för symbolen summa, i jämförelseoperationer ska symbolen betraktas som ett 8-bitars tal med inbyggt tecken och i assemblerspråk motsvarar deklarationen:

```
summa:     RMB   1
```

Observera att direktivet RMB inte innehåller någon typinformation om minnesinnehållet utan bara anger storleken på det reserverade utrymmet.

Flyktig minnesallokering – lokala variabler

Som alternativ använder vi ”lokala variabler”, med andra egenskaper:

- Åtkomlig bara i den programdel den deklarerats
- Inga ”sidoeffekter”
- Ockuperar enbart minne då dess programdel exekveras
- Rekursion kan användas

```
char glob_a;
sub_namn
{
   char loc_a;
}
```

```
glob_a      RMB   1     ;  1 byte reserveras för symbolen.
sub_namn:
; 1 byte reserveras på stacken för symbolen loc_a
      LEASP -1,SP
      ...
; I subrutinen har nu den lokala variabeln adressen 0,SP
      ...
; stacken återställs, variabeln loc_a finns inte längre
      LEASP 1,SP
      RTS
```


Lokala variablers adresser – ”n,SP”

```
sub_namn
{
   char a, b, c, d;
}
```

```
sub_namn:
; 4 bytes reserveras på stacken för symbolerna a,b,c och d
      LEASP      -4,SP
```

Här kan nu de lokala variablerna användas. Eftersom vi har behandlat dom i ordning kan vi enkelt tilldela respektive variabel adresserna i form av stack-relativt adresseringssätt.

- Variabel d behandlades sist och får adress `0,SP`
- variabel c behandlades näst sist och får adressen `1,SP`
- variabel b får adressen `2, SP`
- variabel a får adressen `3,SP`.

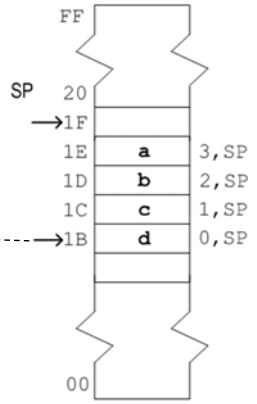
```
; stacken måste återställas inför återgång
      LEASP      4,SP
      RTS
```

Referens av lokala variabler

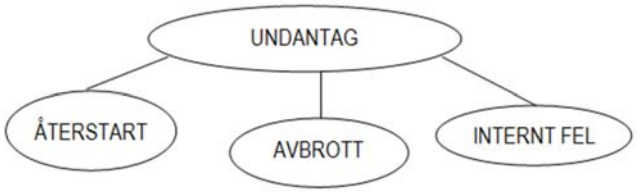
```
sub_namn
{
   char a, b, c, d;
   a = 10;
   b = c + d;
   ...
}
```

Antag SP har värdet $1F_{16}$ innan minne reserveras (LEAS -4,SP)

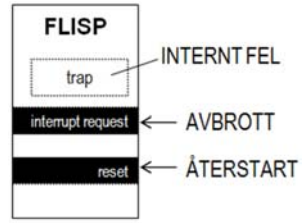
```
sub_namn:
      LEASP    -4,SP          SP-4→SP
      LDA      #10            ; a = 10;
      STA      3,SP
      LDA      1,SP
      ADDA     0,SP           ; c + d
      STA      2,SP           ; b = c + d
      ...
      LEASP    4,SP           SP+4→SP
      RTS
```



Undantagstillstånd

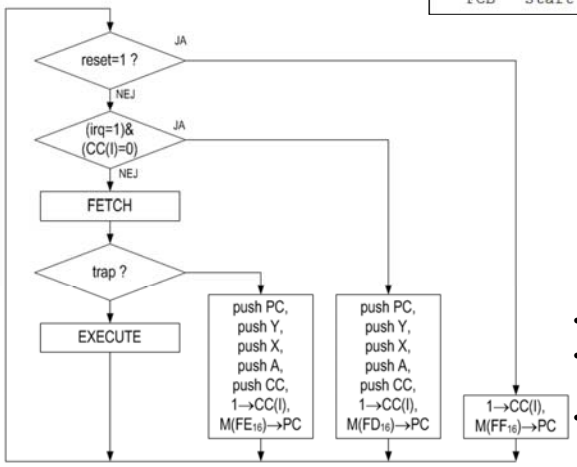


- *Återstart* ("reset"), händelser som alltid föranleder återstart av processorn.
- *Avbrott* ("interrupt"), externa händelser, som dock inte innebär att processorn ska återstartas.
- *Internt fel* ("traps"), händelser som uppträder vid utförandet av en instruktion och som gör att instruktionen inte kan fullföljas.



Undantagsvektorer

```
ORG   $FD
FCB   irqhandler      ; hanteringsrutin för avbrott
FCB   traphandler     ; hanteringsrutin för interna fel
FCB   start           ; återstart
```



FF_{16}	"reset", återstart
FE_{16}	"trap", undantag
FD_{16}	"interrupt", avbrott

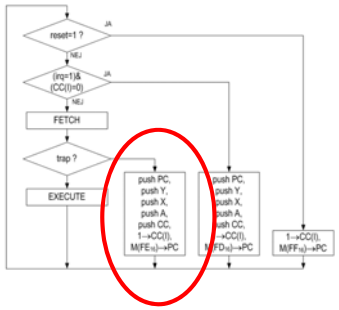
- Vid ”reset” laddas adressen `start` i PC.
- *Vid* "interrupt" laddas adressen `irqhandler` i PC.
- *Vid* "traps” laddas adressen `traphandler` i PC.

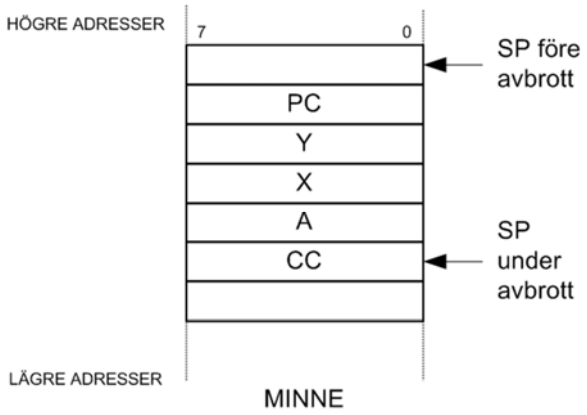
Interna fel

Interna fel uppstår då instruktionsexekvering inte kan fullföljas. Några exempel på interna fel, i det generella fallet är:

- *Division med 0*, kan inträffa i processorer som har maskininstruktioner för division och eftersom operationen inte är definierad kan den normalt inte heller fullföljas.
- Om processorn avkodar en *otillåten operationskod* kan inte denna utföras som en instruktion. I bland kallas detta också för *trap*, eller *software interrupt*.
- *Adressfel*, processorer har oftast mycket stora adressrum som inte fullständigt bestyckats med minne och periferikretsar. Om processorn gör ett försök att hämta en instruktion eller data från en sådan adress kan detta upptäckas och utlösa adressfel.
- *Bussfel*, kan inträffa hos processorer som kräver att vissa datatyper lagras på visst sätt i minnet. Exempelvis kan man kräva att när en 32-bitars datatyp lagras i ett byte-organiserat minne får detta bara ske på adresser som är jämnt delbara med 4 (*alignment condition*).

Stackens utseende – vid inträde i ”traphandler”



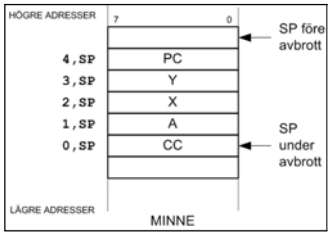


PC = M(FE_{16})

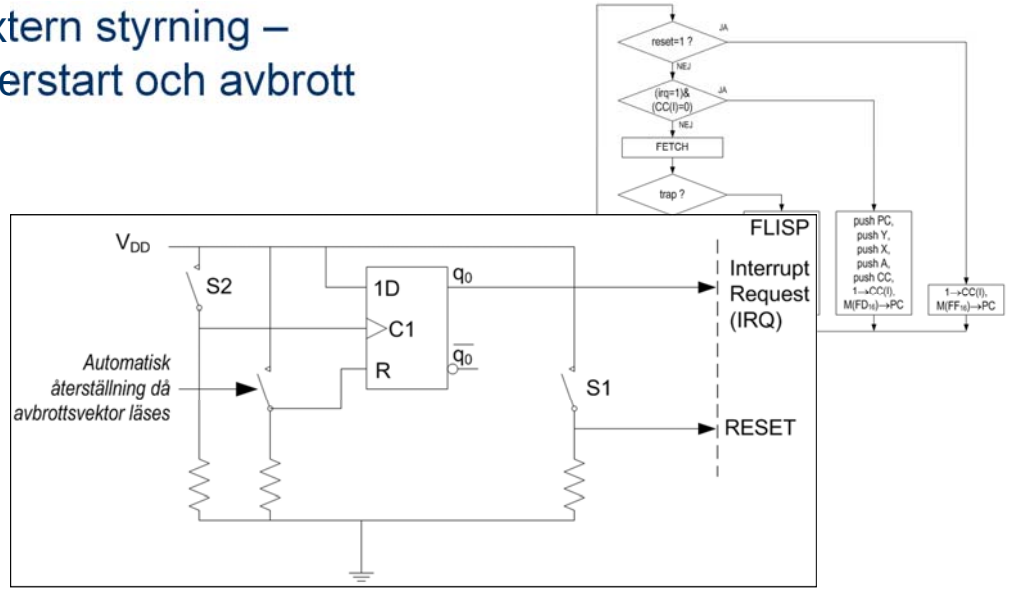
Exempel på TRAP – emulering av instruktion

Följande exempel visar hur en otillåten operationskod kan användas för att *emulera* en instruktion som inte finns i instruktionsuppsättningen.

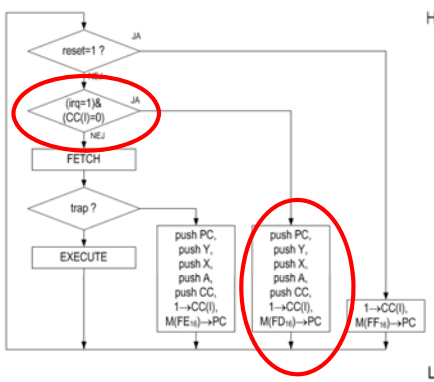
```
        ORG   $20
start:  LDSP   #$20
        LDA    #$55
        LDX    #$88
        FCB    3              ; illegal opkod
stop    BRA    stop

traphandler:  STA    2,SP
              STX    1,SP
              RTI

        ORG   $FE
        FCB   traphandler     ; hanteringsrutin för interna fel
        FCB   start           ; återstart
```

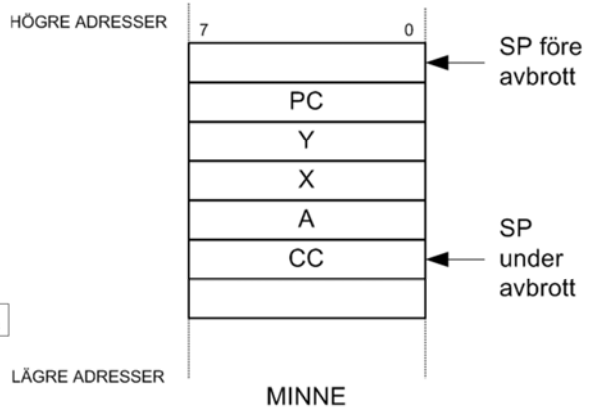


Extern styrning – Återstart och avbrott



Stackens utseende – vid inträde i ”irqhandler”





PC = M(FD_{16})

Avbrottshantering – exempel: räkna antal avbrott

```
       ORG   $20
start: LDSP   #$20
       ANDCC  #$F          ; nollställ I-flagga i CC
main:  LDA    count
       STA    $FB
stop   BRA    main

irqhandler:   INC    count
              RTI
count  RMB    1

       ORG   $FD
       FCB   irqhandler    ; hanteringsrutin för avbrott
       ORG   $FF
       FCB   start         ; återstart
```


Mall – ett assemblerprogram för FLISP

```
                ORG   $20
start:          LDSP    #$20
                ANDCC   #%11101111      ; nollställ I-flagga
restart:        JSR     main
                BRA     restart

main:           ...                     ; applikationsprogram
                RTS

irqhandler:     ...
                RTI
traphandler:    ...
                RTI

                ORG   $FD
                FCB   irqhandler        ; hanteringsrutin för avbrott
                FCB   traphandler       ; hanteringsrutin för interna fel
                FCB   start             ; återstart
```