Styrenheten – styrsignalsekvenser programflödeskontroll

Dagens föreläsning behandlar:

Kompendium kapitel 8.6
Arbetsboken kapitel 14

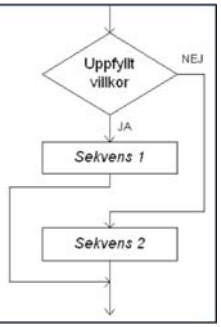
Ur innehållet:

- Konstruktion och implementering av styrsignalsekvenser:
  - Kontroll av programflödet
  - PC-relativ adressering
  - Stackfunktioner
  - Subrutiner

Kontroll av programflöde

Instruktionerna är ordnade sekventiellt i minnet och utförs normalt i denna ordning. Vissa programkonstruktioner kräver dock programflödesändring.

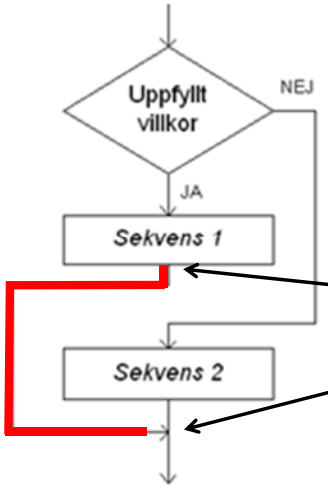
```
if ( Uppfyllt villkor ) then
   do Sekvens 1;
else
   do Sekvens 2;
end if;
```



Två typer av programflödesändringar

- Ovillkorlig
- Villkorlig, baserad på flaggorna i CC-registret

Ovillkorlig programflödesändring



Exempel på situation som kräver ovillkorlig flödesändring

Sekvens2 är en sekvens av instruktioner som INTE ska utföras om *Sekvens1* utförs.

Programflödet måste ”hoppa över” *Sekvens2*

Instruktion ”Jump”:
JMP <ny adress>

Instruktionen JMP

JMP	Jump
RTN	EA → PC
Flaggor	Påverkas ej
Beskrivning	Ovillkorlig programflödesändring, nästa instruktion hämtas från effektiva adressen EA.

Detaljer:

Instruktion JMP Variant	Adressering metod	OP	#	~	Operation	Flaggor N	Z	V	C
JMP Adr	Absolute	33	2	2	Adr → PC	-		-	-
JMP n,X	Indexed	53	2	4	n+X → PC				
JMP A,X	Indexed	63	1	4	A+X → PC				
JMP n,Y	Indexed	73	2	4	n+Y → PC				
JMP A,Y	Indexed	83	1	4	A+Y → PC				

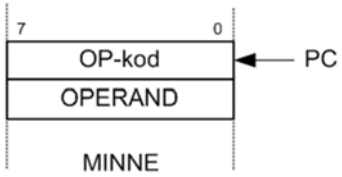
Då adressen är känd från början

Då adressen är känd relativt en position som måste beräknas

Då adressen är känd som en position som måste beräknas med en förskjutning som också måste beräknas

Utförandefasen av JMP Adr

”OPERAND” är destinationsadress
Det räcker med en cykel för exekveringsfasen, där operanden, dvs. destinationsadressen, läses från minnet direkt till register PC.
M(PC)→PC



Styrsignalerna blir:

Tillstånd	Summa-term	RTN-beskrivning	Styrsignaler (=1)	Kommentarer
Q_4	$(Q_4 \bullet I_{33})$	M(PC)→PC;	MR; LD_{PC}; NF	'Adr' läses från minnet till PC Instruktion klar, hämta näs

```
      JMP Adr
Adr->PC ; NF
# MergeState MR=      (I33*Q4)
# MergeState LDPC=    (I33*Q4)
# MergeState NF=      (I33*Q4)
```


Utförandefaser för JMP n,Y och JMP A,Y

För de indexerade adresseringssätten måste vi använda ALU:n för att addera offseten till innehållet i basregistret:

1. Offseten, detta kan vara en konstant som läses från minnet, eller innehållet i register A, placeras i register T, som förberedelse.
2. Basregistret (Y) läggs ut i datavägen, addition utförs i ALU:n och resultatet sparas i register R.
3. Resultatet i register R skrivs till PC.

Implementeringen följer direkt av ovanstående:
Q_4: M(PC)→T;
Q_5: T+Y→R;
Q_6: R→PC
styrsignalsekvens:

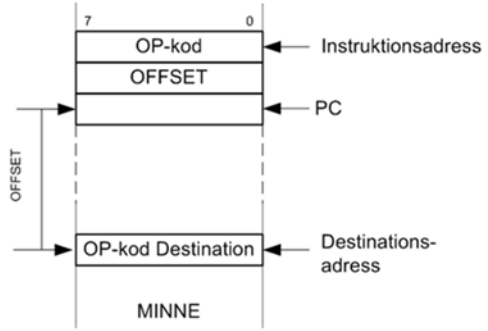
Tillstånd	Summa-term	RTN-beskrivning	Styrsignaler (=1)	Kommentarer
Q_4	$(Q_4 \bullet I_{73})$	M(PC)→T;	MR; LD_T;	'n' läses från minnet till T
Q_5	$(Q_5 \bullet I_{73})$	T+Y→R;	OE_Y; $f_3f_1f_0$; LD_R;	adressberäkning
Q_6	$(Q_6 \bullet I_{73})$	R→ PC;	OE_R; LD_{PC}; NF	register R skrivs till PC Instruktion klar, hämta nästa...

Implementeringen av denna variant skiljer sig från den föregående endast i offseten, det räcker därför med att modifiera i första tillståndet:
Q_4: A→T;

Tillstånd	Summa-term	RTN-beskrivning	Styrsignaler (=1)	Kommentarer
Q_4	$(Q_4 \bullet I_{33})$	A→T;	OE_A; LD_T;	offset läses från A till T
Q_5	$(Q_5 \bullet I_{33})$	T+Y→R;	OE_Y; $f_3f_1f_0$; LD_R;	adressberäkning
Q_6	$(Q_6 \bullet I_{33})$	R→ PC;	OE_R; LD_{PC}; NF	register R skrivs till PC Instruktion klar, hämta nästa...

PC-relativ adressering

Alternativt sätt att koda destinationsadressen vid en programflödesändring:

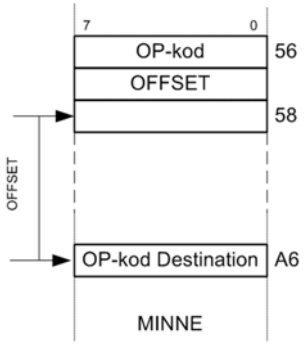


Destinationsadress =
Instruktionsadress + 2 + OFFSET.

PC-relativ adressering kan användas för att skapa *positionsoberoende* kod.

Exempel: Offsetberäkning

Bestäm offset för en instruktion med PC-relativ adressering på adress $(56)_{16}$, och destinationsadress $(A6)_{16}$



Lösning:

Offset = Destinationsadress – (Instruktionsadress+2)=
A6 – (56+2) = 4E

Instruktionen ”Branch always”

BRA — Branch always

RTN	PC+Offset → PC
Flaggor	Påverkas ej
Beskrivning	Ett hopp utförs till adressen ADRESS = PC+Offset. Offset räknas från adressen efter branchinstruktionen, dvs vid uträkningen av hoppadressen pekar PC på operationskoden som (eventuellt) finns direkt efter branchinstruktionen i minnet

Detaljer:

Instruktion BRA	Adressering metod	OP	#	~	Operation	Flaggor N	Z	V	C
BRA Adr	Relativ	21	2	4	PC+Offset → PC	-	-	-	-

Instruktionen används på samma sätt som `JMP Adr` (funktionellt likvärdiga) men operanden kodas PC-relativt

Implementering av BRA *Destinationsadress*

Då exekveringsfasen inleds innehåller PC adressen till OFFSET, RTN för hela instruktionen kan därför skrivas:

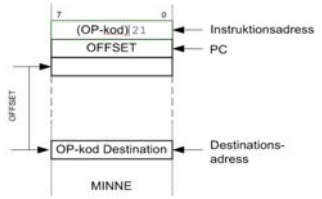
PC+1+(PC)→PC

Operationen delas upp i:

Q_4: PC→T; PC→TA,

Q_5: M(TA)+T+1→R,

Q_6: R→PC



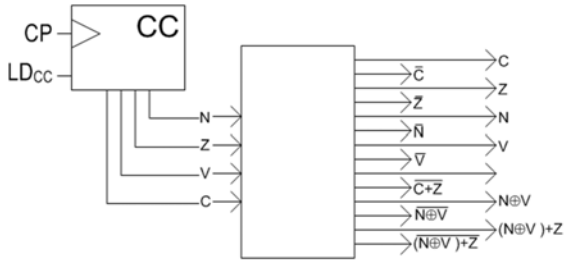
Vilket ger:

Till-stånd	Summa-term	RTN-beskrivning	Aktiva (=1) Styrsignaler	Kommentarer
Q_4	$(Q_4 \bullet I_{21})$	PC→T; PC→TA	OE_{PC}; LD_T; LD_{TA}	Operandhämtning
Q_5	$(Q_5 \bullet I_{21})$	M(TA)+T+1→R	MR;f_3; f_1; f_0;g_0;g_{14} LD_R	Adressberäkning
Q_6	$(Q_6 \bullet I_{21})$	R→PC	OE_R; LD_{PC}; NF	Resultatet återförs till PC

```
      BRA
; Implementering av BRA pcrel
; PC->TA; PC->T
# MergeState OEPC=     (I21*Q4)
# MergeState LDT=      (I21*Q4)
# MergeState LDTA=     (I21*Q4)
; M(TA)+1->R;
# MergeState F3=       (I21*Q5)
# MergeState F1=       (I21*Q5)
# MergeState F0=       (I21*Q5)
# MergeState G0=       (I21*Q5)
# MergeState G14=      (I21*Q5)
# MergeState MR=       (I21*Q5)
# MergeState LDR=      (I21*Q5)
; R->PC; NF
# MergeState OER=      (I21*Q6)
# MergeState LDPC=     (I21*Q6)
# MergeState NF=       (I21*Q6)
```


Villkorlig programflödesändring

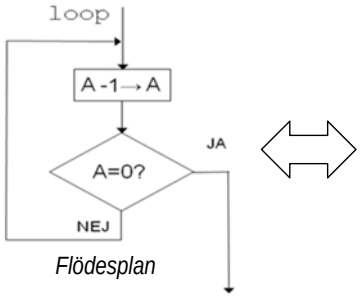
Vid villkorlig programflödesändring används flaggorna i CC-registret för att bestämma om en programflödesändring ska utföras, eller inte.



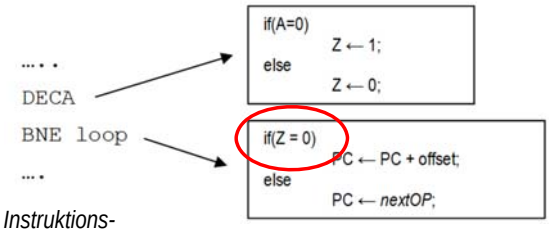
Vi kan välja att testa enskilda flaggor men även olika flaggkombinationer. Den automatiska styrenheten omfattar därför även kombinatorik som tillhandahåller dessa flaggkombinationer.

Vid jämförelse med 0

Programmet styrs att utföra en slinga, så länge innehållet i register A är skilt från 0, BNE (Branch Not Equal). Instruktionen DECA påverkar flaggor i CC, BNE-instruktionen testar…



Flödesplan



Instruktions-sekvens

RTN

Uttrycket ”Z =0” kallar vi *villkorsindikator* och uttryckets värde uppfattas som ”sant” om villkoret är uppfyllt. Om villkorsindikatorn är sann ska flödesändringen utföras, annars ska instruktionen inte göra någonting, utan exekveringen fortsätter med instruktionen omedelbart efter branch-instruktionen.

Implementering av villkorlig instruktion

Först placeras PC i både T och TA för att förbereda bildandet av destinationsadressen:

Q_4: PC→T; PC→TA;

I nästa fas beräknas destinationsadress, samtidigt förbereds för sekventiell exekvering (om villkorsindikatorn är falsk) genom att öka PC med 1:

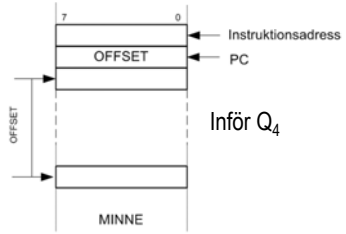
Q_5: M(TA)+T+1→R; PC+1→PC

I den avslutande fasen laddas PC med destinationsadressen (från R) om villkorsindikatorn är sann, annars ska PC lämnas eftersom det innehåller adressen till nästa instruktion.

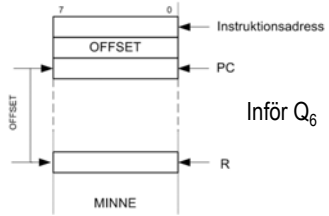
Q_6: if(*Villkorsindikation*)
R→PC

Styrsignal för denna konstruktion blir:

LD_{PC} = *Villkorsindikator*

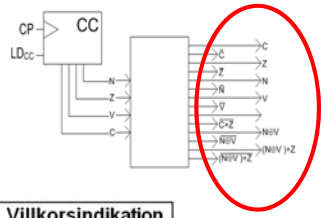


Inför Q_4



Inför Q_6

Villkorsindikatorer
(totalt 14 olika)



Enkla flaggtest

Instruktion (Mnemonic)	Funktion	Villkorsindikation
"Branch if carry set" (BCS)	"Hopp" om *carry*	C=1
"Branch if carry clear" (BCC)	"Hopp" om ICKE *carry*	C=0
"Branch if equal" (BEQ)	"Hopp" om *zero*	Z=1
"Branch if not equal" (BNE)	"Hopp" om ICKE *zero*	Z=0
"Branch if minus" (BMI)	"Hopp" om *negative*	N=1
"Branch if plus" (BPL)	"Hopp" om ICKE *negative*	N=0
"Branch if overflow set" (BVS)	"Hopp" om *overflow*	V=1
"Branch if overflow clear" (BVC)	"Hopp" om ICKE *overflow*	V=0

Tal *utan* tecken

Instruktion (Mnemonic)	Relation	Villkorsindikation
"Branch if higher" (BHI)	R>M	C + Z = 0
"Branch if higher or same" (BHS)	R≥M	C=0
"Branch if lower" (BLO)	R<M	C=1
"Branch if lower or same" (BLS)	R≤M	C + Z = 1

Tal *med* tecken

Instruktion (Mnemonic)	Relation	Villkorsindikation
"Branch if greater than" (BGT)	R>M	(N⊕V) + Z = 0
"Branch if greater or equal" (BGE)	R≥M	N⊕V = 0
"Branch if less than" (BLT)	R<M	N⊕V = 1
"Branch if less or equal" (BLE)	R≤M	(N⊕V) + Z = 1

Relationer

Tal *utan* tecken

R-M	C	Z
R<M	1	x
R≥M	0	x
R>M	0	0

Instruktion (Mnemonic)	Relation	Villkorsindikation
"Branch if higher" (BHI)	R>M	C + Z = 0
"Branch if higher or same" (BHS)	R≥M	C=0
"Branch if lower" (BLO)	R<M	C=1
"Branch if lower or same" (BLS)	R≤M	C + Z = 1

Tal *med* tecken

R-M	N	Z
R<M	1	x
R≥M	0	x
R>M	0	0

Instruktion (Mnemonic)	Relation	Villkorsindikation
"Branch if greater than" (BGT)	R>M	(N⊕V) + Z = 0
"Branch if greater or equal" (BGE)	R≥M	N⊕V = 0
"Branch if less than" (BLT)	R<M	N⊕V = 1
"Branch if less or equal" (BLE)	R≤M	(N⊕V) + Z = 1

Implementering av instruktion BGT

Instruktion **BGT**	Adressering metod	OP	#	~	Operation	Fl N
BGT Adr	Relativ	2C	2	4	If((N⊕V) + Z = 0) PC+Offset →PC	-

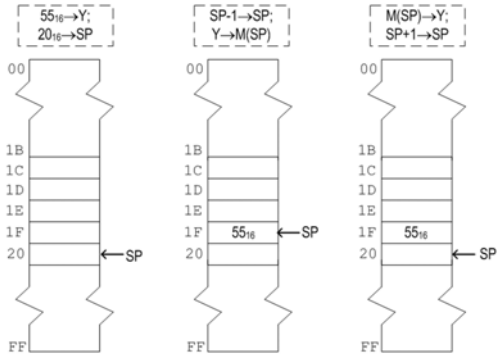
```
BGT
; PC->TA; PC->T
# MergeState OEPC=    (I2C*Q4)
# MergeState LDT=     (I2C*Q4)
# MergeState LDTA=    (I2C*Q4)
; M(TA)+1->R; PC+1->PC
# MergeState F3=      (I2C*Q5)
# MergeState F1=      (I2C*Q5)
# MergeState F0=      (I2C*Q5)
# MergeState G0=      (I2C*Q5)
# MergeState G14=     (I2C*Q5)
# MergeState MR=      (I2C*Q5)
# MergeState LDR=     (I2C*Q5)
# MergeState INCPC=   (I2C*Q5)
; if( (NXV)+Z)=0) R->PC ; NF
# MergeState OER=     (I2C*Q6)
# MergeState LDPC=    (I2C*Q6*(!((NXV)+Z)))
# MergeState NF=      (I2C*Q6)
```

Villkorsindikatorn, i detta fall (N⊕V) + Z = 0, realiseras med det boolska uttrycket !((N⊕V) + Z), styrsignalsekvensen blir därför:

Till-stånd	Summa-term	RTN-beskrivning	Aktiva (=1) Styrsignaler	Kommentarer
Q_4	(Q_4•I_{2C})	PC→T; PC→TA	OE_{PC}; LD_T; LD_{TA}	Placera adress till offset i TA och PC i T.
Q_5	(Q_5•I_{2C})	M(TA)+T+1→R; PC+1→PC	MR;f_3; f_1; f_0;g_0;g_{14};LD_R; INC_{PC}	Adressberäkning, destinationsadress till R Uppdatera PC till att peka på nästa instruktion
Q_6	(Q_6•I_{2C})	if((N⊕V) + Z = 0) R→PC	OE_R; LD_{PC}= !((N⊕V) + Z); NF	Om ((N⊕V) + Z)=0 återförs destinationsadressen till PC annars hämtas nu nästa instruktion

Implementering av stackfunktioner Exempel PSHY/PULY

```
LDSP  #20₁₆
LDY   #55₁₆
PSHY
PULY
```



PSHY	Inherent	12	1	3	A → M(SP) SP–1 → SP Y → M(SP)

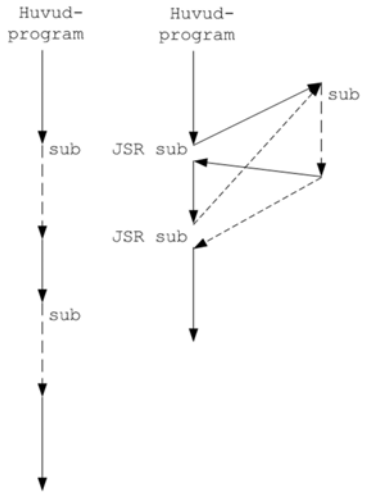
Till-stånd	Summa-term	RTN-beskrivning	Aktiva (=1) Styrsignaler	Kommentarer
Q_4	$(Q_4 \bullet I_{12})$	SP-1→SP	DEC_{SP}	Minska stackpekare
Q_5	$(Q_5 \bullet I_{12})$	Y→M(SP)	$OE_Y; g_{12}; MW; NF$	Skriv register Y till stack

PULY	Inherent	16	1	3	M(SP) → Y SP+1 → SP

Till-stånd	Summa-term	RTN-beskrivning	Aktiva (=1) Styrsignaler	Kommentarer
Q_4	$(Q_4 \bullet I_{16})$	M(SP) → Y	$LD_Y; g_{12}; MR$	Läs register Y från stack
Q_5	$(Q_5 \bullet I_{16})$	SP+1 → SP	$INC_{SP}; NF$	Öka stackpekare

Modularisering, subrutiner

Delsekvenser som utförs frekvent placeras som *subrutiner*



Instruktioner:

JSR (*Jump to SubRoutine*)
RTS (*ReTurn from Subroutine*)

Instruktion **JSR** variant	Adressering metod	OP	#	~	Operation	Flaggor N	Z	V	C
JSR Adr	Absolute	34	2	4	SP–1 → SP PC → M(SP) Adr → PC	-	-	-	-

Till-stånd	Summa-term	RTN-beskrivning	Aktiva (=1) Styrsignaler	Kommentarer
Q_4	$(Q_4 \bullet I_{34})$	M(PC)→R; PC+1→PC; SP-1→SP;	MR; f_5; f_6; LD_R; INC_{PC}; DEC_{SP};	Adress från minnet till register R Uppdatera PC Minska stackpekare
Q_5	$(Q_5 \bullet I_{34})$	PC→M(SP)	$OE_{PC}; g_{12}$; MW	Lägg PC på stacken
Q_6	$(Q_6 \bullet I_{34})$	R→PC	$OE_R; LD_{PC}$; NF	Adress från register R till PC

Instruktion **RTS**	Adressering metod	OP	#	~	Operation	Flaggor N	Z	V	C
RTS	Inherent	43	1	2	M(SP) → PC SP+1 → SP	-	-	-	-

Till-stånd	Summa-term	RTN-beskrivning	Aktiva (=1) Styrsignaler	Kommentarer
Q_4	$(Q_4 \bullet I_{43})$	M(SP)→PC; SP+1→SP;	MR; g_{12}; LD_{PC}; INC_{SP}; NF	Adress från minnet, överst på stacken, till register PC Återställ stackpekare

Exempel: subrutinanrop