Det lagrade programmets princip – en automatisk styrenhet

Dagens föreläsning behandlar:

Kompendiet kapitel 8.1-8.4
Arbetsboken kapitel 14

Ur innehållet:

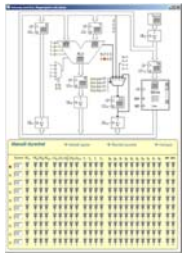
- Det lagrade programmets princip
- En automatisk styrenhet
- Konstruktion och implementering av styrsignalsekvenser

Det lagrade programmets princip

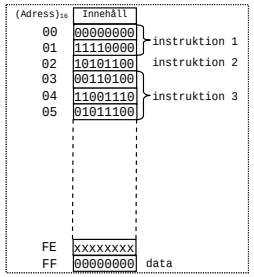
John Louis Von Neumann (1903-1957)

”Det lagrade programmets princip”, dvs program och data i samma minne.

Sådana datorer kallas också ”von Neumann-dator”



Omkopplare som tidigare ställts in för att styra registeröverföring, ALU-operation etc. ersätts av en ”instruktion” som placerats i minnet.



Det lagrade programmets princip

Metoden förutsätter en automatisk styrenhet som kan identifiera en specifik operation och därefter utföra styrsignalsekvensen för operationen.

Vi tilldelar varje specifik operation en unik ”operationskod”

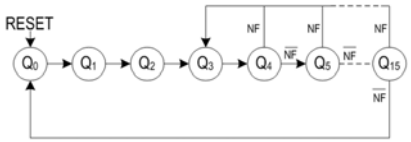
EXEMPEL:

CLRA

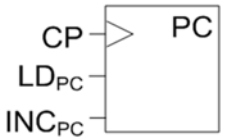
CLR 10_{16}

Instruktion	Adressering		Operation	Flaggor			
Clear Variant	metod	OP		N	Z	V	C
CLRA	Inherent	05	0 → A	0	1	0	0
CLR Adr	Absolute	35	0 → M(Adr)				

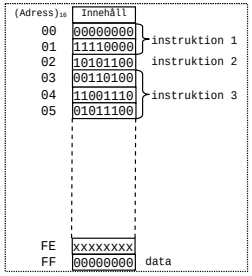
Vi tillför en sekvensierare för att kunna identifiera varje steg i styrsignalsekvensen



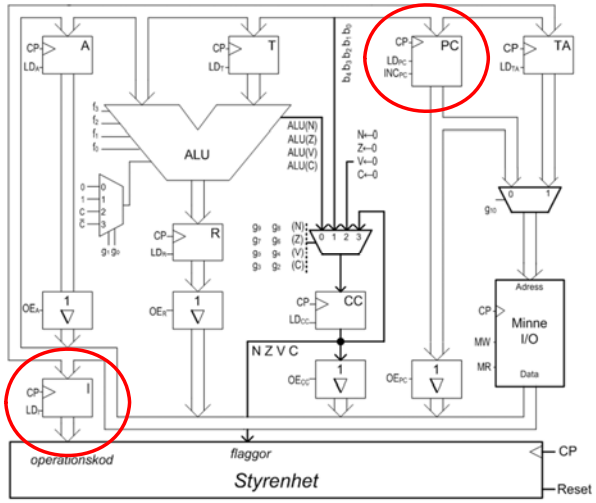
Programräknaren PC – dirigenten…



- En automatisk styrenhet i en von Neumann-dator förutsätter att instruktioner placerats efter varandra (”konsekutivt”) i minnet.
- Minnesadressen till den instruktion i programmet som står i tur att utföras lagras därför i *programräknaren* (PC).
- Programräknarens innehåll uppdateras automatiskt efter hand som instruktioner hämtas och utförs.
- Eftersom operationen PC+1→PC är mycket frekvent så förses PC med den speciella funktionen INC_{PC}.



Dataväg, minne med programräknare och instruktionsregister

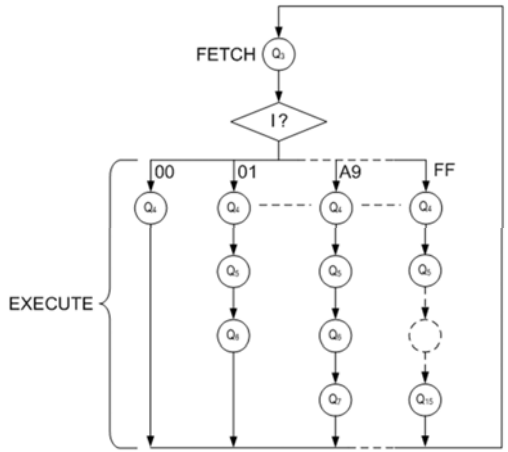


- Vi inför ett nytt register (I) instruktionsregister avsett enbart för operationskoder
- Vi inför också en programräknare, PC
- För att kunna låta PC adressera minnet på ett enkelt sätt inför vi här en väljare med styrsignalen g_{10}.
- Då g_{10}=0 kopplas PC till minnets adressledningar.
- Då g_{10}=1 kopplas TA (som tidigare) till minnets adressledningar.

Hämta och utför instruktion

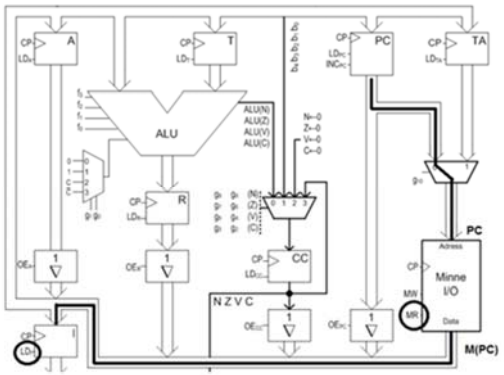
I sin enklaste form kan en instruktion delas in i två faser:

- I den första fasen, som kallas *hämtfasen*, läses operationskoden för instruktionen i minnet och lagras i instruktionsregistret I,
- I den andra fasen, som kallas *utförandefasen*, avkodar styrenheten innehållet i instruktionsregistret I, dvs operationskoden, och utför den styrsignalsekvens som operationskoden anger.



Hämta instruktion (FETCH)

RTN-beskrivning	Styrsignaler	Kommentarer
M(PC)→I;	MR=1; LDI=1; INCPC=1;	Adressen för nästa instruktions operationskod dvs. PC kopplas till minnets adressbuss. Läs operationskoden från minnet och placera i instruktionsregistret I. Adressen som finns i PC ökas med ett.



- FETCH-fasens uppgift är att överföra nästa instruktions operationskod till instruktionsregistret
- Detta förutsätter då att PC verkligen innehåller adressen till en instruktion (snarare än data…)
- Efter instruktionshämtning är det nödvändigt att öka PC med 1, och förbereda för en exekveringsfas eller en ny instruktionshämtning

Stack-operationer SP

push

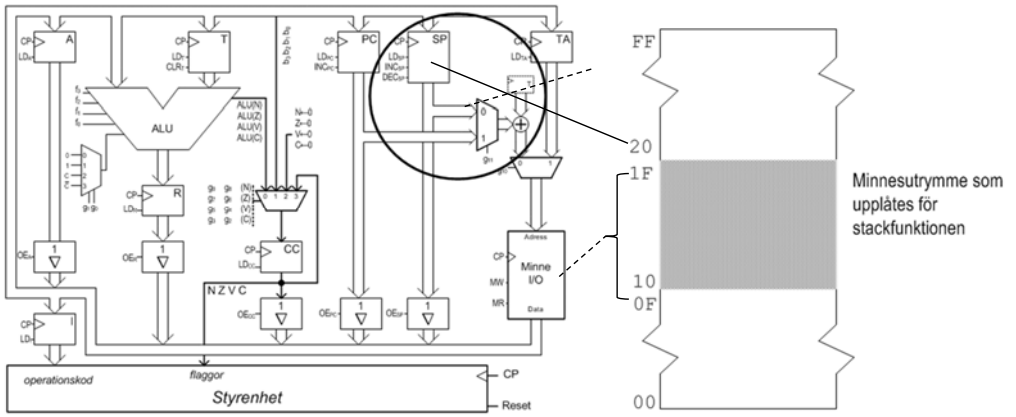
- Först minskas SP med 1 för att skapa utrymme för registerinnehållet
- Därefter placeras registerinnehållet på det nu utpekade minnesutrymmet.

SP-1→SP,
r→M(SP)

pull

- Värdet på den adress som SP innehåller, kopieras till ett register
- Därefter ökas SP med 1, och återställer därmed stacken

M(SP)→r,
SP+1→SP

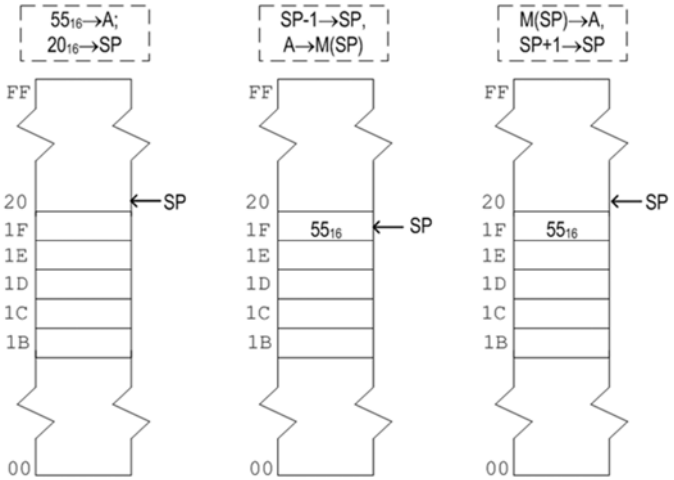


Tillfällig undanlagring med SP

Instruktionssekvensen

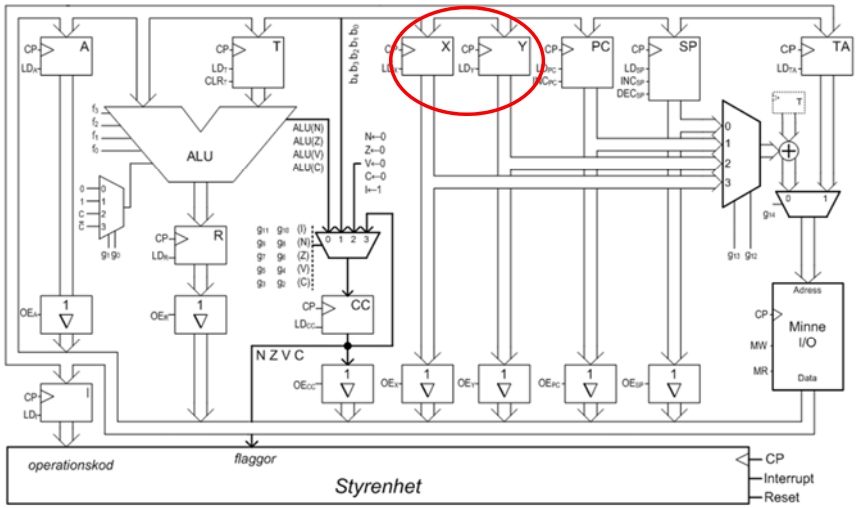
```
LDSP  #20₁₆
LDA   #55₁₆
PSHA
PULA
```

Illustreras i figuren .

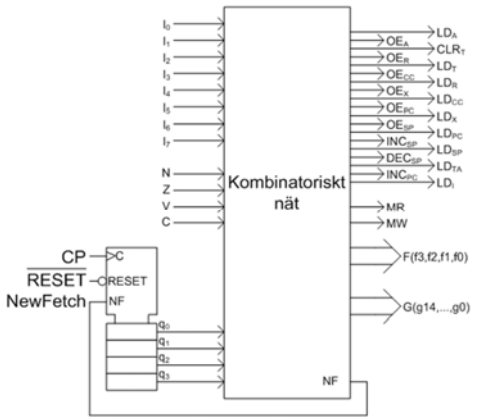


FLIS-processorn

- Ytterligare två ”adress”-register, X och Y, har tillförts…

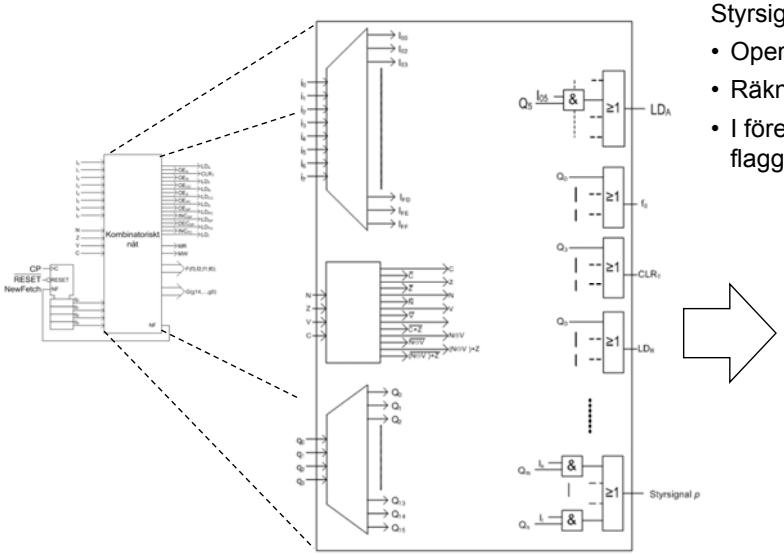


FLIS-processorns styrenhet



Signal	Funktion
Insignaler till styrenheten	
CP	"Clock Pulse", systemklocka
Reset	Asynkron återställningssignal
Interrupt	Extern avbrottssignal, samplas vid CP (positiv flank),kontrolleras vid NF
Q_{state}	Tillstånd, "state" anges på hexadecimal form med siffrorna 0 t.o.m F, dvs totalt 16 olika tillstånd
I_{opcode}	Operationskod, anger innehållet i instruktionsregistret avkodat av en "en av 256-väljare". "opcode" anges på hexadecimal form, dvs 0 t.o.m FF.
N-flagga	N-flaggan, dvs. bit 3 i CC
Z-flagga	Z-flaggan, dvs. bit 2 i CC
V-flagga	V-flaggan, dvs. bit 1 i CC
C-flagga	C-flaggan, dvs. bit 0 i CC
Utsignaler från styrenheten	
LD_A	"Load register A enable", Ackumulator
LD_I	"Load register I enable", instruktionsregister
OE_A	"Output enable register A", Ackumulator
LD_T	"Load register T enable", Temporärregister
CLR_T	"Clear register T", Temporärregister
LD_R	"Load register R enable", Resultatregister
OE_R	"Output enable register R", Resultatregister
LD_{CC}	"Load register CC enable", Flaggregister (Condition Codes)
OE_{CC}	"Output enable register CC", Flaggregister (Condition Codes)
LD_X	"Load register X enable", Adressregister
OE_X	"Output enable register X", Flaggregister
LD_Y	"Load register Y enable", Adressregister
OE_Y	"Output enable register Y", Flaggregister
LD_{PC}	"Load register PC enable", Programräknare
OE_{PC}	"Output enable register PC", Programräknare
INC_{PC}	"Increment register PC", Addera 1 till programräknare
LD_{SP}	"Load register SP enable", Stackpekare
OE_{SP}	"Output enable register SP", Stackpekare
INC_{SP}	"Increment register PC", Addera 1 till stackpekare
DEC_{SP}	"Decrement register SP", Subtrahera 1 från stackpekare
LD_{TA}	"Load register TA enable", Adressberäkningsregister
MR	"Memory Read", Avkoda adressbuss och klocka ut data från minne/"Input"
MW	"Memory Write", Avkoda adressbuss och klocka in data till minne/"Output"
$f_3..f_0$	Funktionsväljare för ALU
$g_{14}..g_0$	Styrsignaler för väljarfunktioner

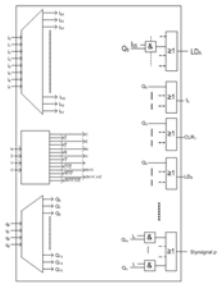
FLIS-processorns styrenhet

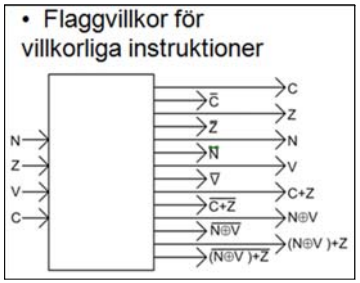


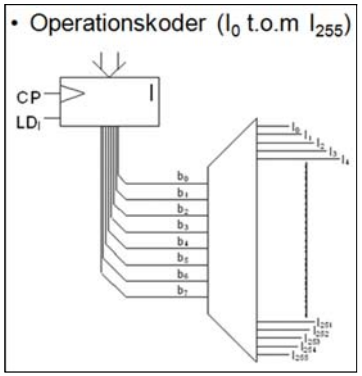
Styrsignaler bildas från:

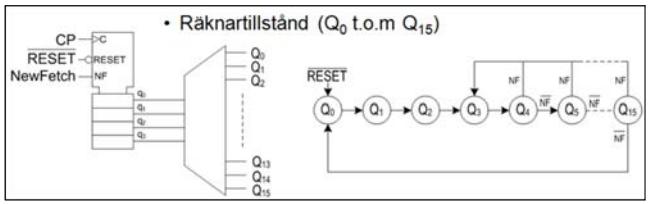
- Operationskod
- Räknartillstånd
- I förekommande fall: flaggvillkor

Styrenhetens insignaler

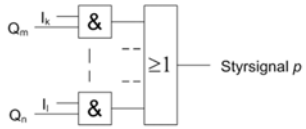




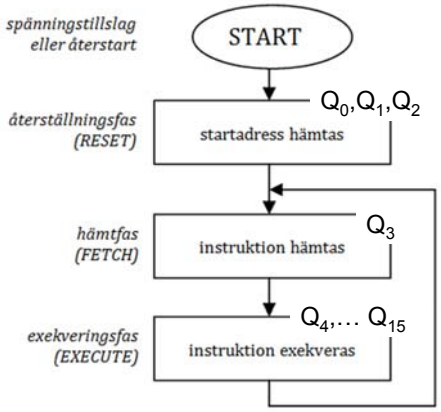


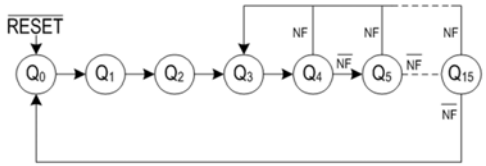


AND/OR för en utsignal



Styrenhetens olika faser





RESET och FETCH faser konstanta, 3 respektive 1 klockcykel

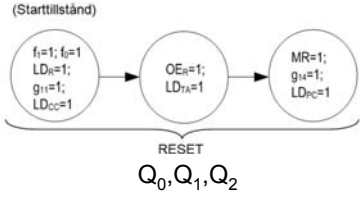
EXECUTE är instruktionsberoende och får ta maximalt 12 klockcykler

RESET

Operation

$1 \rightarrow CC(I),$
$M(FF_{16}) \rightarrow PC$

Motsvarande tillståndsgraf

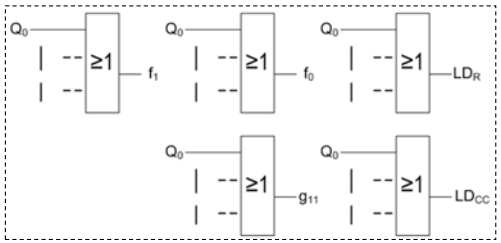


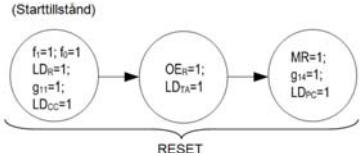
Styrsignalsekvens

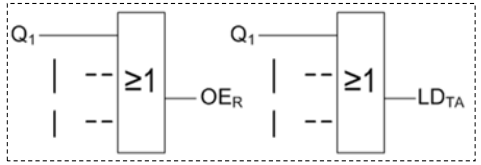
Tillstånd	Summa-term	RTN-beskrivning	Styrsignaler	Kommentarer
Q_0	$(Q_0 \bullet 1)$	$(FF)_{16} \rightarrow R$	$f_1=1; f_0=1;$ $LD_R=1;$ $g_{11}=1;$ $LD_{CC}=1;$	ALU-funktionen väljs så att talet FF_{16} finns på ALU:ns utgång, dvs funktionskod 3, ALU-funktion = $(0011)_2$. Laddingången på R-registret ettställs så att utvärdet från ALU'n FF_{16} laddas i R-registret vid nästa klockpuls. *I*-flaggan i CC sätts till *1*, ovriga flaggor får initialvärden *(1,0,0,0)*
Q_1	$(Q_1 \bullet 1)$	$R \rightarrow TA$	$OE_R=1;$ $LD_{TA}=1;$	Talet FF_{16} i R-registret kopplas ut på bussen. Talet FF_{16} på bussen laddas i temporäradressregistret vid nästa klockpuls.
Q_2	$(Q_2 \bullet 1)$	$M \rightarrow PC$	$MR=1;$ $g_{14}=1;$ $LD_{PC}=1;$	Minnesinnehållet på adress FF_{16} läses genom att minnet aktiveras för läsning. Temporäradressregistret adresserar minnet Det dataord som läses placeras i PC vid nästa klockpuls.

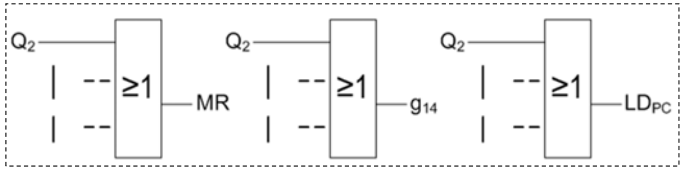
RESET

Logik för RESET i styrenheten









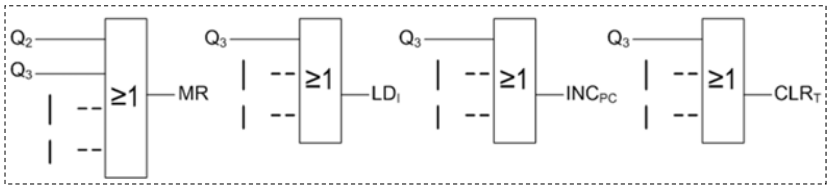
FETCH

Operation: M(PC)→I; PC+1→PC

Q_3: MR=1; LD_I=1; CLR_T=1; INC_{PC}=1

Tillstånd	Summa-term	RTN-beskrivning	Styrsignaler	Kommentarer
Q_3	(Q_3•1)	M(PC)→I; 0→T;	MR=1; LD_I=1; INC_{PC}=1; CLR_T=1;	Adressen för nästa instruktions operationskod dvs. PC kopplas till minnets adressbuss. Läs operationskoden från minnet och placera i instruktionsregistret I. Adressen som finns i PC ökas med ett. Register för index vid adressberäkningar nollställs.

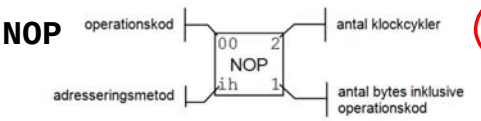
Logik för FETCH i styrenheten



EXECUTE – styrsignalsekvenser för utförande

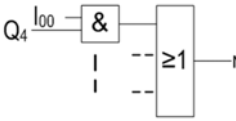
EXEMPEL

NOP



Instruktion **NOP**	Adressering				Operation	Flaggor			
	metod	OP	#	~		N	Z	V	C
NOP	Inherent	00	1	2	No operation	-	-	-	-

Tillstånd	Summa-term	RTN-beskrivning	Styrsignaler (=1)	Kommentarer
Q_4	(Q_4•I_{00})		NF	Instruktion klar, hämta nästa...



Flaggpåverkan

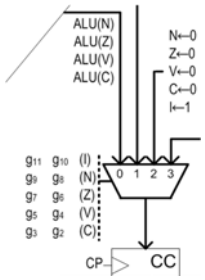
Flaggbitarna i CC kan påverkas med hjälp av styrsignalerna g_2 t.o.m g_{11}.

g_3	g_2	C väljs enligt:	RTN
0	0	C tas från ALU'n	ALU(C)→C
0	1	C tas från bit 0 på bussen	b_0→ C
1	0	Återställning (nollställning) av C	0→ C
1	1	C återförs (ändras ej)	

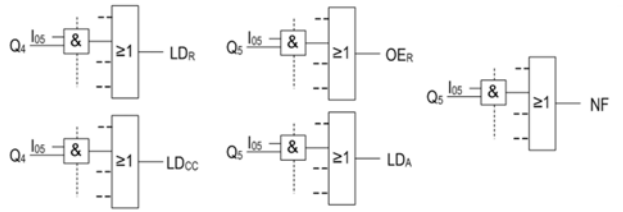
EXEMPEL **CLRA**

Instruktion **Clear** Variant	Adressering				Operation	Flaggor			
	metod	OP	#	~		N	Z	V	C
CLRA	Inherent	05	1	3	0 → A	0	1	0	0

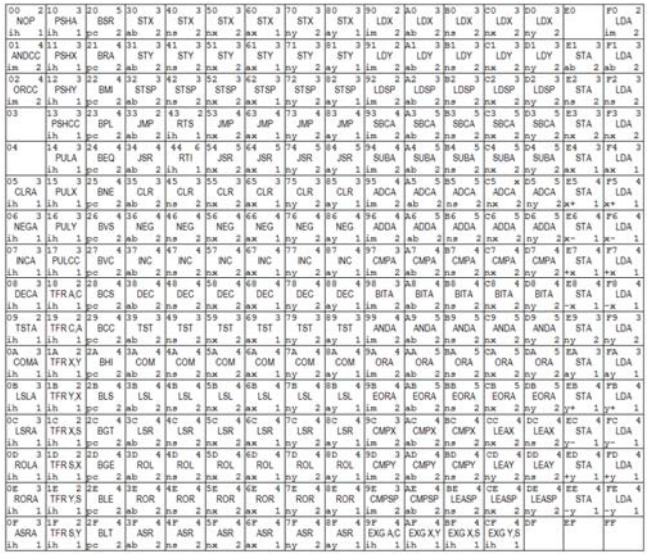
Tillstånd	Summa-term	RTN-beskrivning	Styrsignaler (=1)	Kommentarer
Q_4	(Q_4•I_{05})	0→R; ALU(N,V,Z,C) 0→CC;	LD_R; LD_{CC};	Alla ALU'ns f-signaler är 0 vilket ger ALU-funktion "nollställning", värdet 0 placeras därför i R Samtliga flaggor från ALU'n överförs till CC
Q_5	(Q_5•I_{05})	R→A	OE_R; LD_A; NF	Resultat (0) överförs till register A Instruktion klar, hämta nästa...

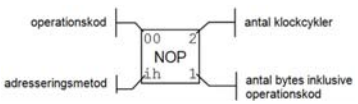


Styrenhetens logik utökas därför enligt följande:



Översikt av FLISP:s instruktionsuppsättning





Detaljerade beskrivningar:

Instruktion **Clear** Variant	Adressering				Operation	Flaggor			
	metod	OP	#	~		N	Z	V	C
CLRA	Inherent	05	1	3	0 → A	0	1	0	0

Exempel: CLRA

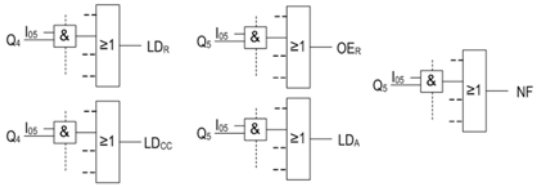
styrsignalsekvensen för exekveringsfasen ska implementeras, metoden är:

Instruktion Clear Variant	Adressering metod	OP	#	~	Operation	Flaggor N	Z	V	C
CLRA	Inherent	05	1	3	0 → A	0	1	0	0

1. Utgå från instruktionens beskrivning i handboken

Tillstånd	Summa-term	RTN-beskrivning	Styrsignaler (=1)	Kommentarer
Q_4	($Q_4 \bullet I_{05}$)	0→R; ALU(N,V,Z,C) 0→CC;	LD_R; LD_{CC};	Alla ALU:ns f-signaler är 0 vilket ger ALU-funktion "nollställning", värdet 0 placeras därför i R Samtliga flaggor från ALU:n överförs till CC
Q_5	($Q_5 \bullet I_{05}$)	R→A	OE_R; LD_A; NF	Resultat (0) överförs till register A Instruktion klar, hämta nästa...

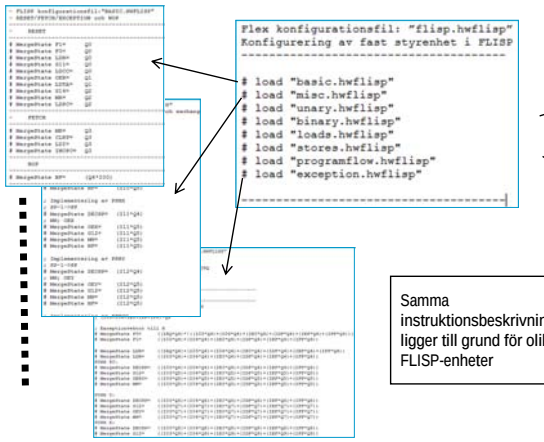
2. Beskriv först i RTN, sedan i styrsignaler



3. Implementering med AND/OR-logik, dvs. summatermernas bidrag till styrenheten

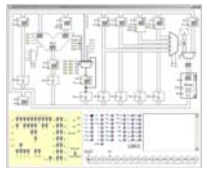
Implementering i FLISP

Styrsignalsekvenser implementeras i FLISP, simulator och laborationssystem, via speciella ”script”-filer.

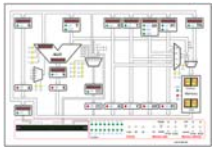


Samma instruktionsbeskrivningar ligger till grund för olika FLISP-enheter

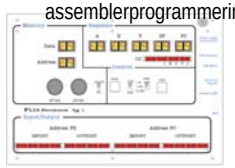
Simulator DigiFlisp, ETERM för FLISP



Laboration 3, styrsignalsekvenser



Laboration 4, assemblerprogrammering

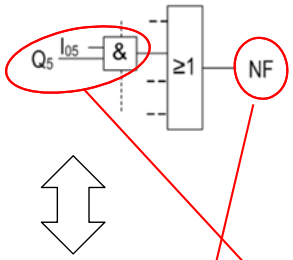


”Merge State” – lägg till AND-villkor i ELLER nätet

Direktivet används för att lägga till ytterligare ett specifikt AND-villkor (Tillstånd * operationskod)

Den nya produkten läggs till den ELLER-grind, vars styrsignal man utökar.

Exempel:




```
------------------------------
# MergeState NF= (Q5*I05)
------------------------------
```


Exempel: CLRA, beskrivning i ”script” fil

```
--------------------------------
     CLRA (05)
; 0->R
# MergeState LDR=(I05*Q4)
; Flaggsättning
; ALU(N)->CC(N); ALU(Z)->CC(Z);ALU(V)->CC(V); ALU(C)->CC(C);
# MergeState LDCC=(I05*Q4)
# MergeState OER=(I05*Q5)
# MergeState LDA=(I05*Q5)
# MergeState NF=(I05*Q5)
--------------------------------
```

Rader som *inte* inleds med '#' tolkas som kommentarer

Motsvarar AND/OR-logiken: