Aktivera Kursens mål:

- Konstruera en dator mha grindar och programmera denna

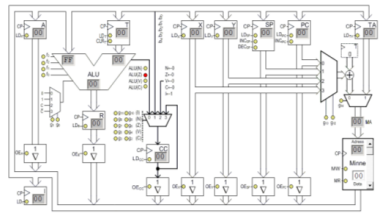
Aktivera Förra veckans mål:

- Koppla samman register och ALU till en dataväg
- Minnets uppbyggnad och anslutning till datavägen
- Program och hur detta lagras i minne
- Fatta hur datorn startar och arbetar
- Räknare och mera vippor

Veckans mål:

- Konstruera styrenheten…. genom att….
- implementera olika maskininstruktioner i styrenheten.
- Kunna använda instruktionslistan och skriva mycket enkla assemblerprogram
- Studera olika instruktionstyper och adresseringsmoder
- Använda utvecklingsmiljön för FLISP

Läs smart!
Lär dig mer!



State	Summa-term	RTN	Styrsignaler = 1	Kommentar
4	Q_4*I_{23}	A+T →R	OE_A, LD_T, $f_3f_1f_0$	
5	Q_5*I_{23}	R → A	OE_R, LD_A, NF	

Mask. prog i minnet — *Tillhörande assemblerprog*

Adr.			
0C	F0	LDA	#$23
0D	23		
0E	A6	ADDA	$F3
0F	F3		
10	02	TFR	A,X
11	4F	CMPA	#$03
12	03		
13	61	BLO	$29
14	13		

A6 betyder:
A+M(F3) → A
(Addera M(F3) till register A)

Assemblernivå

Beskrivning av funktion

"Automatiskt styrd borrmaskin"
- Positionera borr
- Starta borr
- Borra genom arbetsstycke
- ...

Assemblerspråk

```
Fortsätt  STAA   BorrStyr
          LDAA   BorrStat
          ANDA   #B1Mask
          CMPA   #BorrNere
          BNE    Fortsätt
          ---
```

Beskrivning av styrsignaler

CP1: OE_{PC}=1, LD_{Adr} =1, Inc_{PC}=1
CP2: MR=1, LD_T=1
CP3: OE_{DR1}=1, LD_R, f_3=1 f_1=1
...

LV4 Fo10

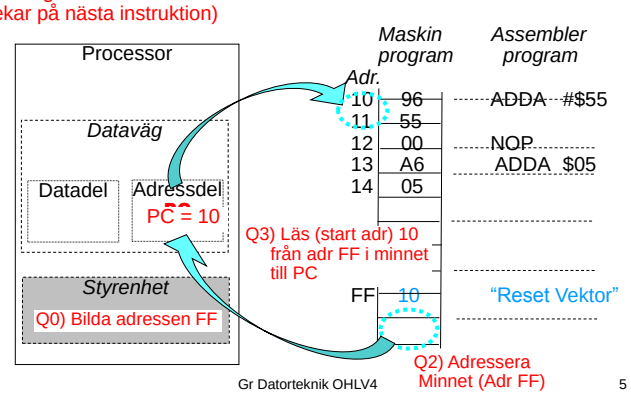
Fokus på...

Du ska kunna…
- Implementera ….
  - RESET i styrenheten
  - FETCH i styrenheten
  - ADDA #Data

Processorns arbetssätt - RESET

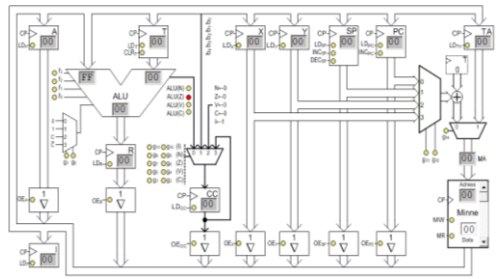
Arb s 95

PC: Programräknare
(Pekar på nästa instruktion)



Processorns arbetssätt - RESET

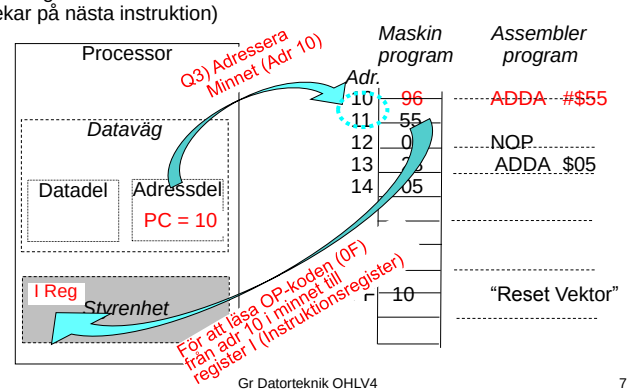
Arb s 101



Tillstånd	Summaterm	RTN-beskrivning	Styrsignaler (=1)
Q_0	Q_0	$FF_{16} \rightarrow R$	f_1, f_0, LD_R
Q_1	Q_1	$R \rightarrow TA$	OE_R, LD_{TA}
Q_2	Q_2	$M \rightarrow PC$	MR, g_{14}, LD_{PC}

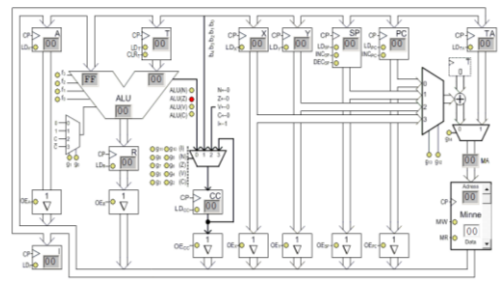
Processorns arbetssätt - FETCH

PC: Programräknare
(Pekar på nästa instruktion)



Processorns arbetssätt - FETCH

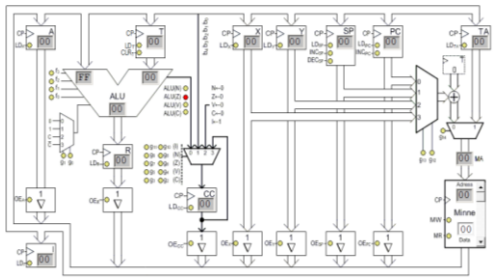
Arb s 109



Tillstånd	Summaterm	RTN-beskrivning	Styrsignaler (=1)
Q_3	Q_3	M(PC)→I, PC+1→PC, 0→T,	OE_{PC}, LD_I, INC_{PC}, CLR_T

Processorns arbetssätt – Execute

Arb s 133



ADDA #Data

State	Summa-term	RTN	Styrsignaler = 1	Kommentar
4	Q_4*I_{96}	A+T→R, ALU→CC	OE_A, LD_T, $f_3f_1f_0$, LD_{CC}	
5	Q_5*I_{96}	R → A	OE_R, LD_A, NF	

Arb s 107

```
Flex konfigurationsfil: Test s 107

# ClearAllStates
# ClearAllMemory
# ClearAllRegisters


-------------------------------------
- LÄS IN DINA MASKININSTRUKTIONER
# load "flisp.hwflisp"


-------------------------------------
- RESET-VEKTOR
#setMemory       FF=20


-------------------------------------
- MASKINPROGRAM
-------------------------------------
#setMemory       20=96    ADDA #$31
#setMemory       21=31
-------------------------------------
#setMemory       22=96    ADDA #$51
#setMemory       23=51
```

Konfigurationsfiler Flisp

```
Flex konfigurationsfil: "flisp.hwflisp"
Konfigurering av fast styrenhet i FLISP
-------------------------------------
# load "basic.hwflisp"



-------------------------------------
```


Konfigurationsfiler Flisp

```
- Konfigurationsfil:"BASIC.HWFLISP"
- RESET/FETCH/EXCEPTION och NOP
-------------------------------------
-        RESET
-------------------------------------
# MergeState      F1=      Q0
# MergeState      F0=      Q0
# MergeState      LDR=     Q0

# MergeState      OER=     Q1
# MergeState      LDTA=    Q1

# MergeState      G14=     Q2
# MergeState      MR=      Q2
# MergeState      LDPC=    Q2
-------------------------------------
-        FETCH
-------------------------------------
# MergeState      MR=      Q3
# MergeState      CLRT=    Q3
# MergeState      LDI=     Q3
# MergeState      INCPC=   Q3
-------------------------------------
```

```
-------------------------------------
-        NOP
-  OP-kod:00 Antal klockcykler:1
-  Flagpåverkan: Nej
-------------------------------------
# MergeState      NF=      Q4*I00
-------------------------------------

-------------------------------------
-        ADDA  #Data
-  OP-kod:96 Antal klockcykler:4
-  Flagpåverkan:NZVC
-------------------------------------
# MergeState      MR=      Q4*I96
# MergeState      LDT=     Q4*I96
# MergeState      INCPC=   Q4*I96

# MergeState      OEA=     Q5*I96
# MergeState      F3=      Q5*I96
# MergeState      F1=      Q5*I96
# MergeState      F0=      Q5*I96
# MergeState      LDR=     Q5*I96
# MergeState      LDCC=    Q5*I96

# MergeState      OER=     Q6*I96
# MergeState      LDA=     Q6*I96
# MergeState      NF=      Q6*I96
-------------------------------------
```

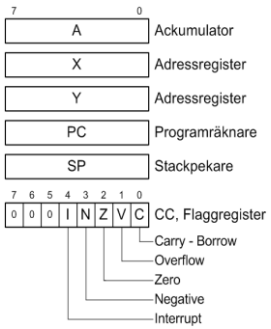
LV4 Fo10

Fokus på...

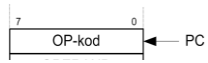
Dagens mål: Du ska kunna…

- **Kunna använda instruktionslistan och**
  - **handassemblera / disassemblera program**
- Kunna använda olika adreseringsmoder
- Implementera ….
  - LDA #Data
  - INCA
  - STA Adress
  - JMP

Registeruppsättning



Adresseringssätt

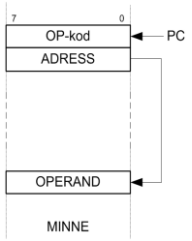


Inherent [ih]
Operanden eller operanderna ges direkt av instruktionen.
Ingen extra operandinformation (utöver operationskod) krävs.
Det finns ingen generell RTN-beskrivning för inherent adressering.

Omedelbar (Immediate) [im]
Operanden är kodad tillsammans med operationen.

RTN: M(PC+1)
Assemblersyntax: `#<DATA>`

Adresseringssätt



Absolut (Absolute) [ab]
Operanden finns i minnet.
Minnesadressen är kodad tillsammans med operationen.

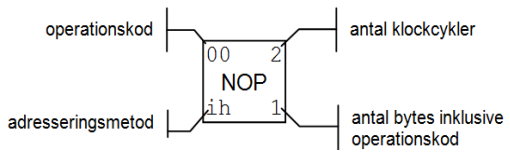
RTN: [M(PC+1)]
Assemblersyntax: `<ADRESS>`

Instruktionsgrupper

”Load/Store”
LDA, LDX, LDY, LDSP, LEAX / STA, STX, STY, STSP
”Data movement”
TFR, EXG
”Program (Flow) control”
JMP, JSR, BRA, BSR, B(condition), RTS, RTI
”Integer arithmetic”
ADDA, ADCA, SUBA, CLR, NEGA, DEC, INC
”Integer test”
CMPA, CMPX, CMPY, CMPSP, BITA, TSTA, TST
”Logical operations”
ANDA, ORA, ANDCC, ORCC, EORA, COMA, COM
”Shift/rotate”
ASRA, ASR, LSLA, LSL, LSRA, LSR, ROLA, ROL, RORA, ROR
”Stack operations”
PSHA, PSHCC, PSHX, PSHY, PULA, PULCC, PULX, PULY
”Misc.”
NOP

Karta över operationskoder

s11



00 2 NOP ih 1	10 3 PSHA ih 1	20 5 BSR pc 2	30 3 STX ab 2	40 3 STX ns 2	50 3 STX nx 2	60 3 STX ax 1	70 3 STX ny 2	80 3 STX ay 1	90 2 LDX im 2	A0 3 LDX ab 2	B0 3 LDX ns 2	C0 3 LDX nx 2	D0 3 LDX ny 2	E0	F0 2 LDA im 2
01 4 ANDC C im 2	11 3 PSHX ih 1	21 4 BRA pc 2	31 3 STY ab 2	41 3 STY ns 2	51 3 STY nx 2	61 3 STY ax 1	71 3 STY ny 2	81 3 STY ay 1	91 2 LDY im 2	A1 3 LDY ab 2	B1 3 LDY ns 2	C1 3 LDY nx 2	D1 3 LDY ny 2	E1 3 STA ab 2	F1 3 LDA ab 2
02 4 ORCC im 2	12 3 PSHY ih 1	22 4 BMI pc 2	32 3 STSP ab 2	42 3 STSP ns 2	52 3 STSP nx 2	62 3 STSP ax 1	72 3 STSP ny 2	82 3 STSP ay 1	92 2 LDSP im 2	A2 3 LDSP ab 2	B2 3 LDSP ns 2	C2 3 LDSP nx 2	D2 3 LDSP ny 2	E2 3 STA ns 2	F2 3 LDA ns 2

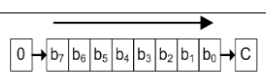
s12

Förklaring av innehållet i instruktionslistan

Instruktion/Variant	
Här anges instruktionens mnemonics med assemblersyntax för de tillgängliga adresseringssätten.	
Adressering	
OP	Operationskod för instruktion, hexadecimal form
#	Antal bytes i instruktionen
~	Antal klockcykler som krävs för att utföra en instruktion
Operationsbeskrivningar (RTN, register transfer notation)	
n	Konstant uttryckt i talbas 10
N,	Konstanten N uttryckt i talbasen r.
Etc	Etc etc etc etc

s13

Instruktionen LSR Logical shift right

RTN	A >>1 → A eller M >>1 → M
Flaggor	N: Nollställs. Z: Ettställs om samtliga åtta bitar i resultatet blir noll. V: Ettställs om overflow vid 2-komplements-representation inträffar. C: bit 0 före skiftet blir ny carrybit efter skiftet.
Beskrivning	Skiftar operanden ett steg till höger, dvs. dividerar ett tal utan inbyggt tecken med 2
	

Instruktion	Adressering				Operation	Flaggor			
LSR									
Variant	metod	OP	#	~		N	Z	V	C
LSRA	Inherent	0C	1	3	A>>1 → A	0	Δ	Δ	Δ
LSR Adr	Absolute	3C	2	4	M(Adr)>>1 → M(Adr)				
LSR n,SP	Indexed	4C	1	4	M(n+SP)>>1 → M(n+SP)				
LSR n,X	Indexed	5C	2	4	M(n+X)>>1 → M(n+X)				
LSR A,X	Indexed	6C	1	4	M(A+X)>>1 → M(A+X)				
LSR n,Y	Indexed	7C	2	4	M(n+Y)>>1 → M(n+Y)				
LSR A,Y	Indexed	8C	1	4	M(A+Y)>>1 → M(A+Y)				

Handassemblering

Assemblerprogram

```
LDA   #$3C
ADDA  $43
STA   4,X
JMP   $2E
```

?

Maskinprogram

Adr	
18	F0
19	3C
1A	96
1B	43
1C	3E
1D	4
1E	33
1F	2E
20	

Disassemblering

Maskinprogram

Adr	
18	F0
19	3C
1A	96
1B	43
1C	3E
1D	4
1E	33
1F	2E
20	

?

Assemblerprogram

```
LDA   #$3C
ADDA  $43
STA   4,X
JMP   $2E
```


Fokus på...

Dagens mål: Du ska kunna…

- Kunna använda instruktionslistan och handassemblera / disassemblera program
- **Kunna använda olika adreseringsmoder**
- Implementera ….
  - LDA #Data
  - INCA
  - STA Adress
  - JMP

Adresseringsmoder

Hur hitta "datat" som instruktionen skall jobba på/med

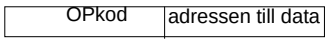
- *Inherent* — *Operanden (data) är inbyggd i Op-koden*


```
INCA
```

- *Immediate* — *Operandfältet innehåller data*

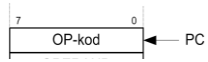

```
LDA  #$12
```

- *Absolut* — *Operandfältet innehåller adresseen till data*




```
LDA  $12
```


Adresseringssätt

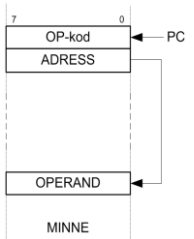


Inherent [ih]
Operanden eller operanderna ges direkt av instruktionen.
Ingen extra operandinformation (utöver operationskod) krävs.
Det finns ingen generell RTN-beskrivning för inherent adressering.

Omedelbar (Immediate) [im]
Operanden är kodad tillsammans med operationen.

RTN: M(PC+1)
Assemblersyntax: `#<DATA>`

Adresseringssätt



Absolut (Absolute) [ab]
Operanden finns i minnet.
Minnesadressen är kodad tillsammans med operationen.

RTN: [M(PC+1)]
Assemblersyntax: `<ADRESS>`

Programexempel för FLISP

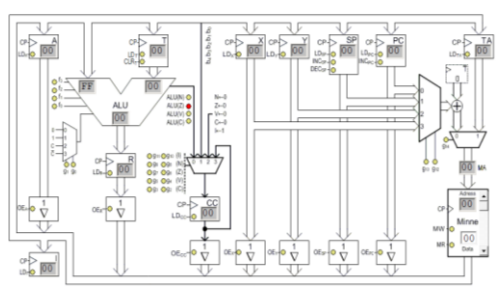
Addera 4 till talet som finns på minnesadress $1C_{16}$

Programmet skall placeras med start på adress 26_{16}

Fokus på...

Dagens mål: Du ska kunna…
- Kunna använda instruktionslistan och handassemblera / disassemblera program
- Kunna använda olika adreseringsmoder
- **Implementera ….**
  - LDA #Data
  - INCA
  - STA Adress
  - JMP

Execute



State	Summa-term	RTN	Styrsignaler = 1	Kommentar

Arbetsgång:

Fokus på...

- Knappa in en rad….
- Studera aktiverade signaler…
- Studera bussens värden….
- Ge en klockpuls….
- Kontrollera nya registerinnehåll..

- ***Var det detta jag ville???***

- Knappa in nästa rad, osv.

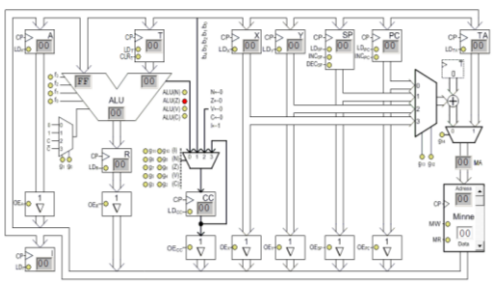
Veckans mål:

- Konstruera styrenheten…. genom att….
- implementera olika maskininstruktioner i styrenheten.
- Kunna använda instruktionslistan och skriva mycket enkla assemblerprogram
- Studera olika instruktionstyper och adresseringsmoder
- Studera utvecklingsmiljön för FLEX

Dagens mål:

- Skriva mycket enkla assemblerprogram
- Implementera flera instruktioner i styrenheten
- Öva på användning av instruktionslistan
- Utvecklingsmiljön för FLEX

Läs smart!
Lär dig mer!



Execute

State	Summa-term	RTN	Styrsignaler = 1	Kommentar

Dagens mål:

- Skriva mycket enkla assemblerprogram
- Implementera flera instruktioner i styrenheten
- Öva på användning av instruktionslistan

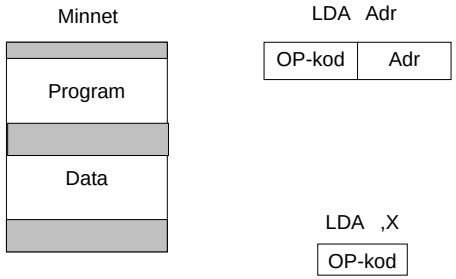
NYTT REGISTER

Du ska kunna…

- **Adressering via Register X**
- Hoppinstruktioner
  - Absoluta hopp JMP
  - Relativa hopp BRA (Branch)
  - Beräkna branch-offset
- Utvecklingsmiljö.
  - Käll-, list- och laddfil
  - Assemblerdirektiv
  - FLEX-datorn
  - IO-Simulatorer

Läs smart!
Lär dig mer!

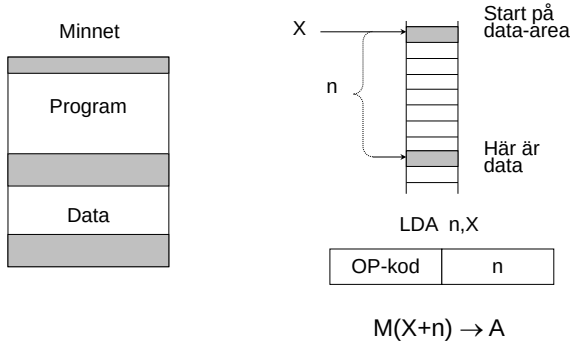
Adressering med register X



Adressering med register X - forts

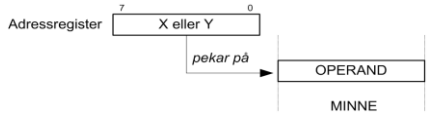
Alt 1		*Alt 2*	
LDA $23	$M(23_{16}) \rightarrow A$	LDX #$23	$23 \rightarrow X$
		LDA ,X	$M(X) \rightarrow A$

Adressering med register X - forts

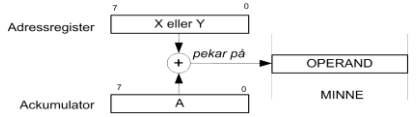


$M(X+n) \rightarrow A$

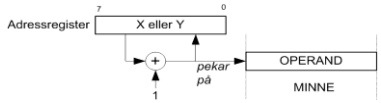
Indirekt register med konstant offset (Indexed) [nR] Instrux s 7



Indirekt register med ackumulator offset (Indexed)[aR]



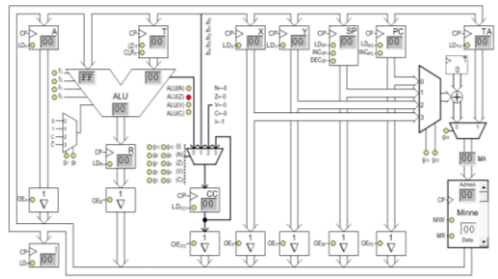
Indirekt register med pre/post decrement/increment (Indexed)



Uppgift

Skriv en instruktionssekvens för FLEX-processorn som nollställer bit 3-0 i alla minnesord i adressintervallet [35_{16}, 39_{16}].

Använd X-registret för adressering.



Execute

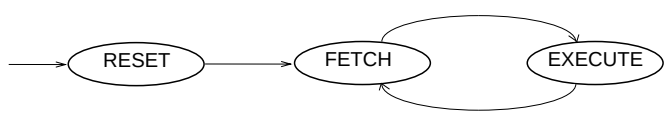
State	Summa-term	RTN	Styrsignaler = 1	Kommentar

LV4 Fo11

Dagens mål:
- Skriva mycket enkla assemblerprogram
- Implementera flera instruktioner i styrenheten

Du ska kunna…
- Adressering via Register X
- **Hoppinstruktioner**
  - **Absoluta hopp JMP**
  - Relativa hopp BRA (Branch)
  - Beräkna branch-offset
- Utvecklingsmiljö.
  - Käll-, list- och laddfil
  - Assemblerdirektiv
  - FLEX-datorn
  - IO-Simulatorer

Hoppinstruktioner



Adr.	*Maskinprogram i minnet*	*Tillhörande assemblerprogram*	
0C	10	LDB	#$23
0D	23		
0E	29	ADDB	$F3
0F	F3		
10	02	TFR	B,A
11	4F	CMPB	#$03
12	03		
13	61	BLO	$29
14	13		

Villkorliga-
o
Ovillkorliga-
hopp

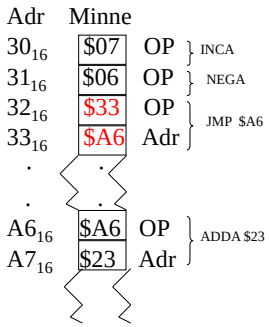
Ovillkorliga Hopp -Instruktioner *JMP-Instruktion* Arb s 123

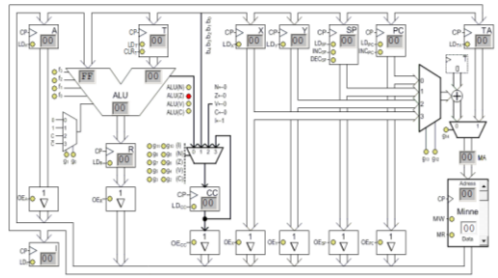
Ex: JMP $A6

Hoppa till Adr $A6_{16}$ och fortsätt (=gör FETCH) där

OP-kod: $33
Ant. Byte: 2
RTN: EA → PC

EA: Effektiva Adressen





Execute

State	Summa-term	RTN	Styrsignaler = 1	Kommentar

Villkorliga hopp

1) Om vi inte bearbetat alla dataord
   - bearbeta nästa dataord
   - börja om från 1)
2) Annars fortsätt med annat arbete

Om PIN-koden är korrekt given
- öppna telefonen

Annars
- stäng telefonen

…någon addition…
Om C=1
- så hoppa till felrutin

Annars
- fortsätt beräkningen

Vid villkorliga hopp används ***relativa hoppinstruktioner***

Relativa hopp

Instruktionsformat

JMP Adr

OP-kod	Adr

RTN-beskrivning:
Adr → PC

Instruktionsformat
BRA Adr

OP-kod	**Offset**

Minnes Adress	Instruktioner i minnet
k	Maskininstruktion i
k+1	Maskininstruktion i+1
k+2	Maskininstruktion i+2
k+3	Maskininstruktion i+2
k+4	Maskininstruktion i+3
k+5	Maskininstruktion i+3
k+6	Maskininstruktion i+4
k+7	Maskininstruktion i+4
k+8	Maskininstruktion i+5
k+9	Maskininstruktion i+6
k+A	Maskininstruktion i+6

RTN-beskrivning *(Processorn utför)**:***
PC+Offset → PC

Relativa hopp - forts

Instruktionsformat
BRA Adr

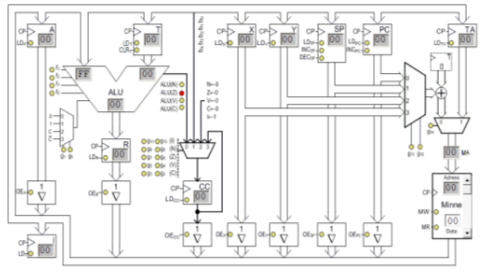
OP-kod	Offset

RTN-beskrivning:
PC+Offset → PC

TillAdress
- FrånAdress
= Offset

Minnes Adress	Instruktioner i minnet
k	Maskininstruktion i
k+1	Maskininstruktion i+1
k+2	Maskininstruktion i+2
k+3	Maskininstruktion i+2
k+4	Maskininstruktion i+3
k+5	Maskininstruktion i+3
k+6	Maskininstruktion i+4
k+7	Maskininstruktion i+4
k+8	Maskininstruktion i+5
k+9	Maskininstruktion i+6
k+A	Maskininstruktion i+6

Implementera EXECUTE-fasen för BRA Adr . Arb s 124



State	Summa-term	RTN	Styrsignaler = 1	Kommentar

LV4 Fo11

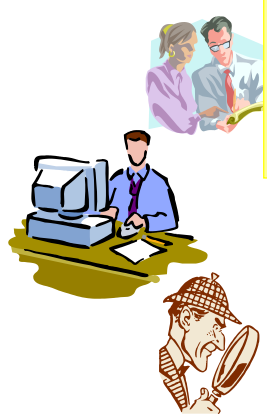
Dagens mål:
- Skriva mycket enkla assemblerprogram
- Implementera flera instruktioner i styrenheten

Du ska kunna…
- Adressering via Register X
- Hoppinstruktioner
  - Absoluta hopp JMP
  - Relativa hopp BRA (Branch)
  - Beräkna branch-offset

Utvecklingsmiljö.
- FLISP-datorn
- Käll-, list- och laddfil
- Assemblerdirektiv
- IO-Simulatorer

Läs smart!
Lär dig mer!

Vi höjer abstraktionsnivån



```
"Borrprogram"

        STAA    BorrStyrRegister
EjNere  LDAA    BorrStatusRegister
        ANDA    #Bottensensor
        CMPA    #BorrNere
        BNE     EjNere
        ---
```

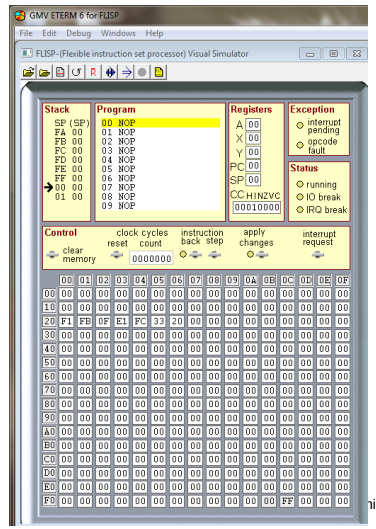
```
"Borrprogram"

        STAA    $FE
        LDAA    $FD
        ANDA    #$20
        CMPA    #$20
        BNE     $B2
        ---
```

Utvecklingsmiljö för FLEX

En utvecklingsmiljö innehåller:

- ✓Editor
  - oTextredigering
- ✓Assemblator (Översätter till maskinkod)
  - oAssemblerdirektiv (Styrinformation till assemblatorn)
- ✓Laddare
  - oFlytta maskinkod från utvecklingssystemet till målsystemet
- ✓Simulatorer
  - oProcessorn
  - oMinnet
  - oI/O
- ✓Hjälpsystem
  - oInstruktionslistor etc. etc.

ASSEMBLER-PROGRAMMERING



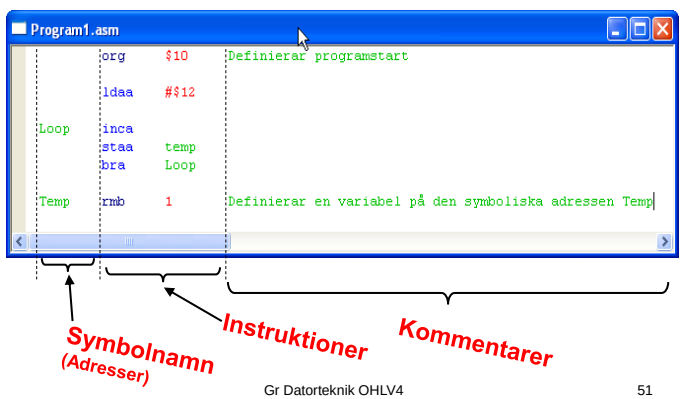
Arb s 146

NY SIMULATOR

ETERM för FLISP

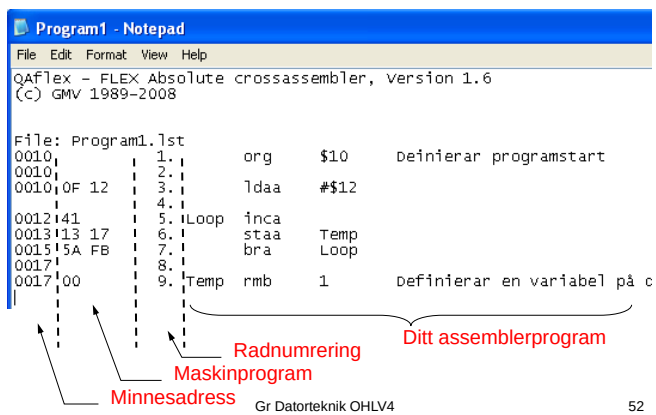
Adress (Hex)	Maskin kod	Assembler kod	
1F			
20	F0	LDA	#$72
21	72		
22	07	INCA	
23	E1	STA	$05
24	05		
25	06	NEGA	
26	21	BRA	$22
27	FA		

En källfil



En listfil

Sid 162



Assemblatordirektiv

Instrux s9

	ORG <Val>	"ORIGIN": Anger startadress för påföljande kod/data. Om en symbol används för att ange startadressen måste symbolen vara definierad, dvs inga framåtreferenser är tillåtna här.
Sym	EQU <Val>	"EQUATE": Symbolen 'Sym' representerar värdet <Val>.
[Sym]	FCB <Val>,<Val>...	"FORM CONSTANT BYTE": Skapar en sträng med initierade data i minnet
[Sym]	FCS "<ASCII tecken>"	"FORM CONSTANT STRING": Skapar en sträng med ASCII-tecken i minnet.
[Sym]	RMB <Val>	"RESERVE MEMORY BYTES": Reservera <Val> bytes i minnet. Minnesinnehållet på dessa adresser är odefinierat.

I/O-simulator