LV1 LV2 LP2

## *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

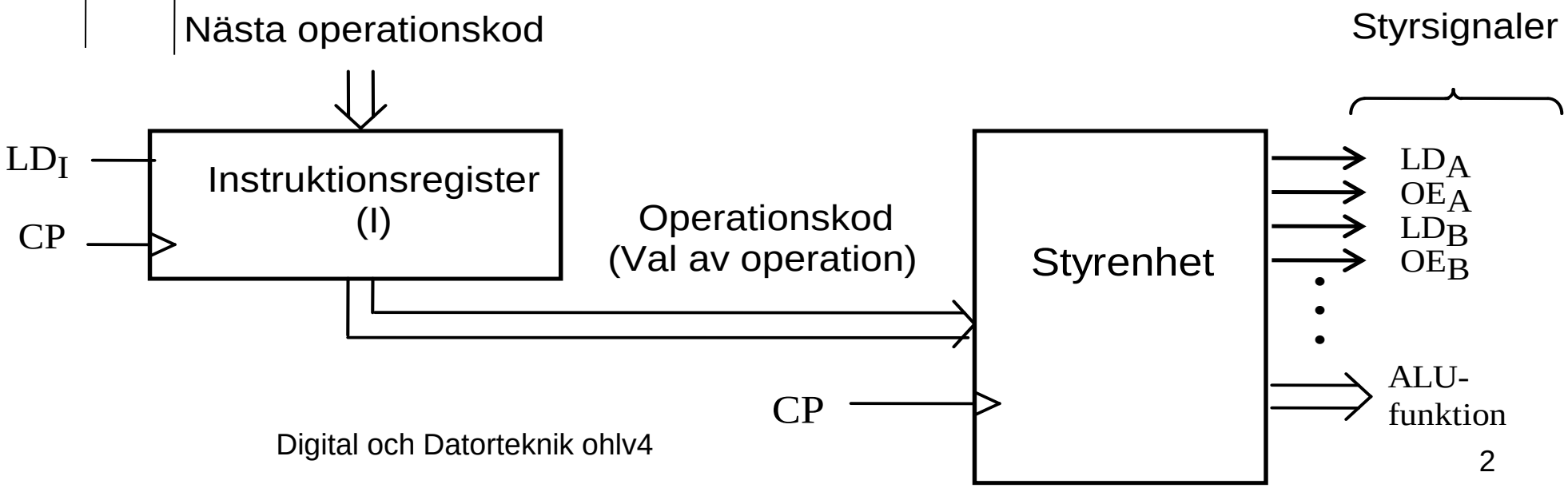
## Veckans mål:

- Kunna använda instruktionslistan och skriva  mycket enkla assemblerprogram
- Studera olika instruktionstyper och adresseringsmoder
- Använda utvecklingsmiljön för FLEX

*Maskinprogram i minnet*

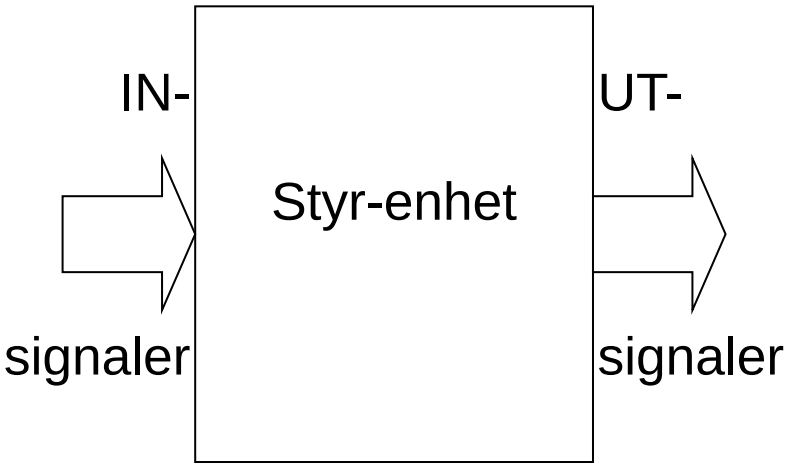
| Adr. | | |
|---|---|---|
| 0C | 10 | LDB #$23 |
| 0D | 23 | |
| 0E | 29 | ADDB $F3 |
| 0F | F3 | |
| 10 | 02 | TFR B,A |
| 11 | 4F | CMPB #$03 |
| 12 | 03 | |
| 13 | 61 | BLO $29 |
| 14 | 13 | |

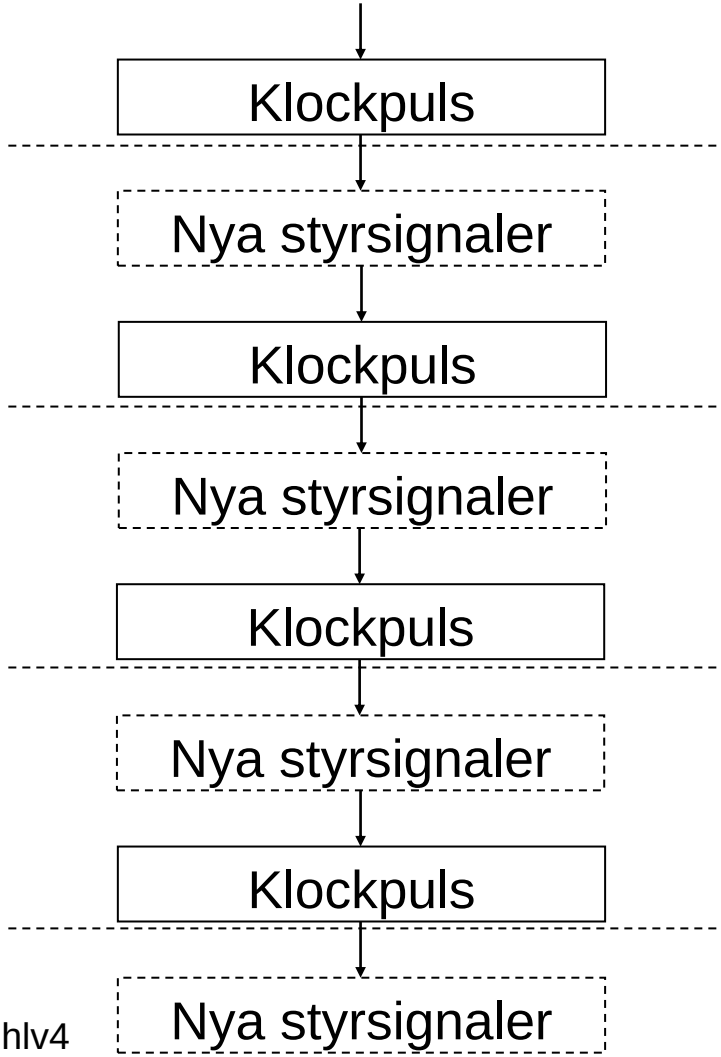
# Styrsignalgenerering för den databehandlande enheten.



# Styrenheten

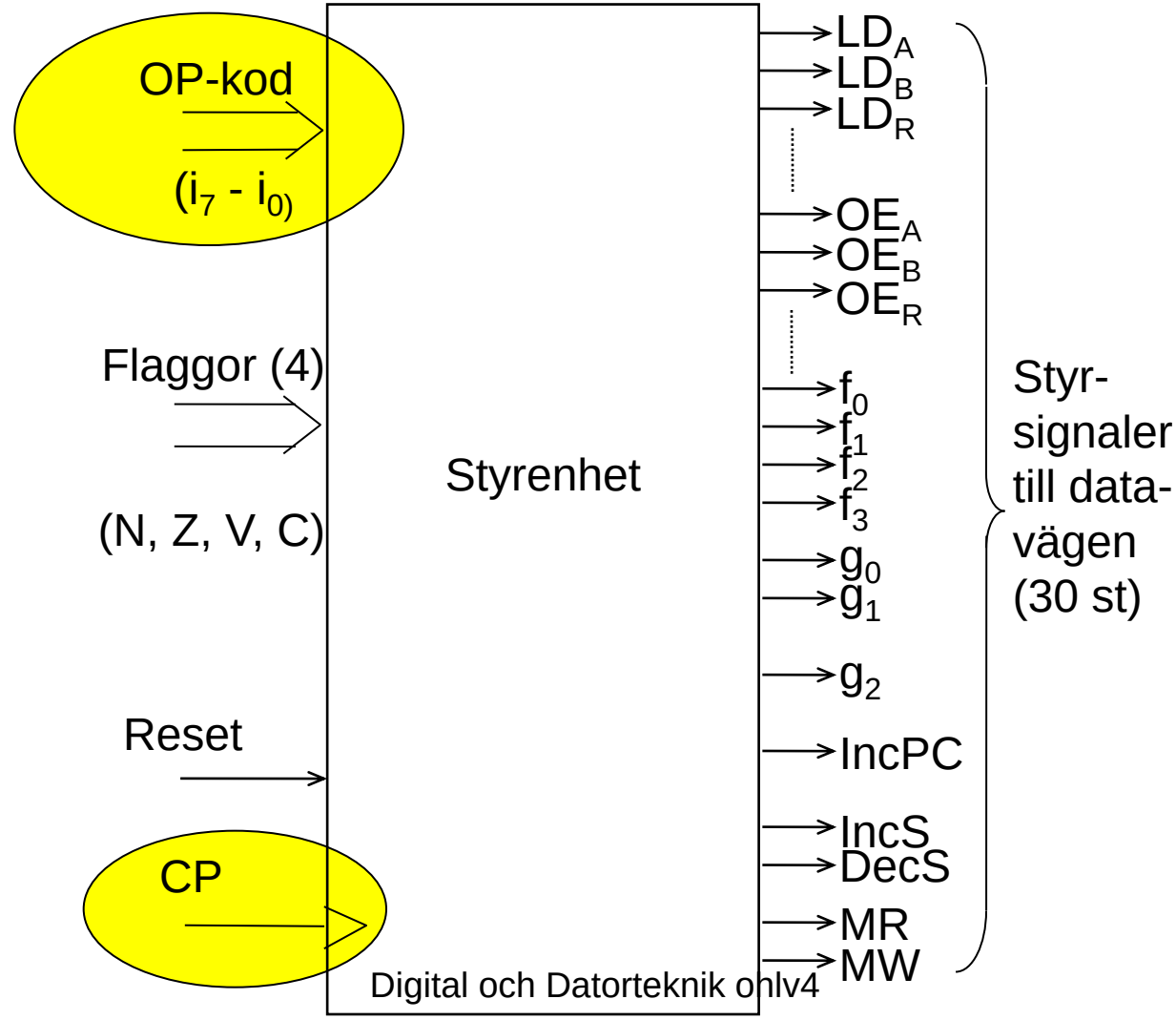
Arb s 120



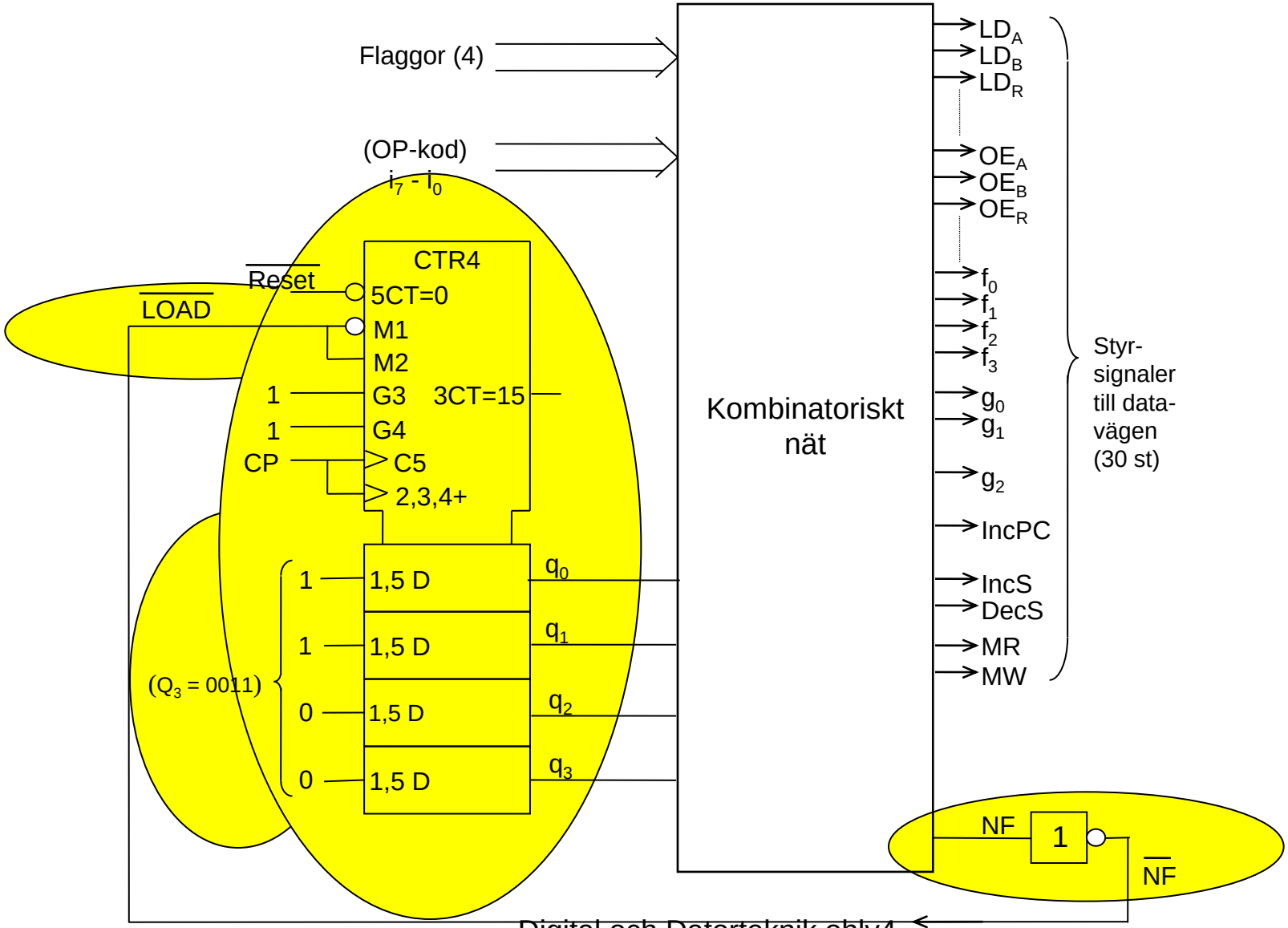
# Styrenheten - forts

Arb s 120

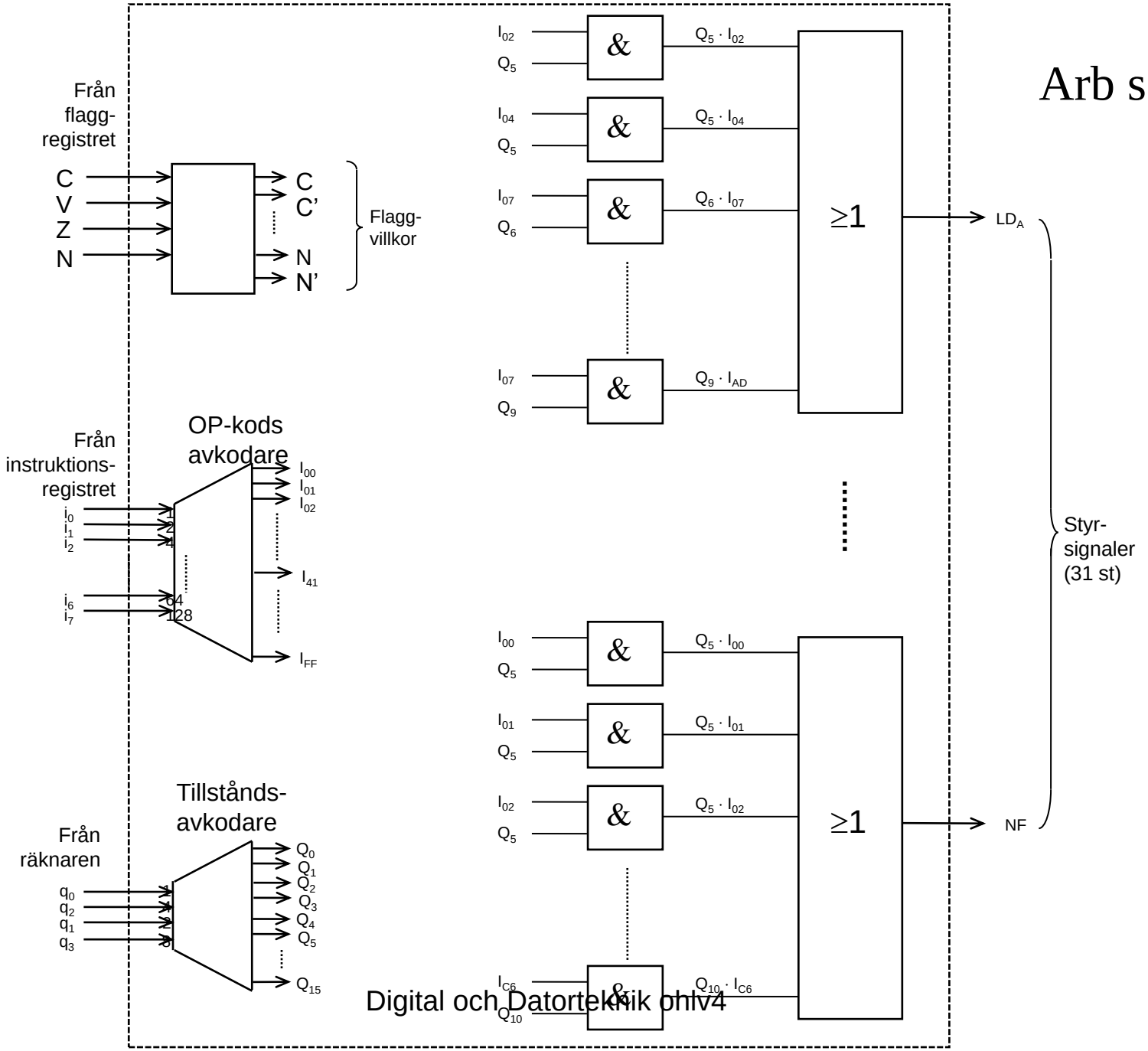


# Styrenheten - forts

Arb s 123

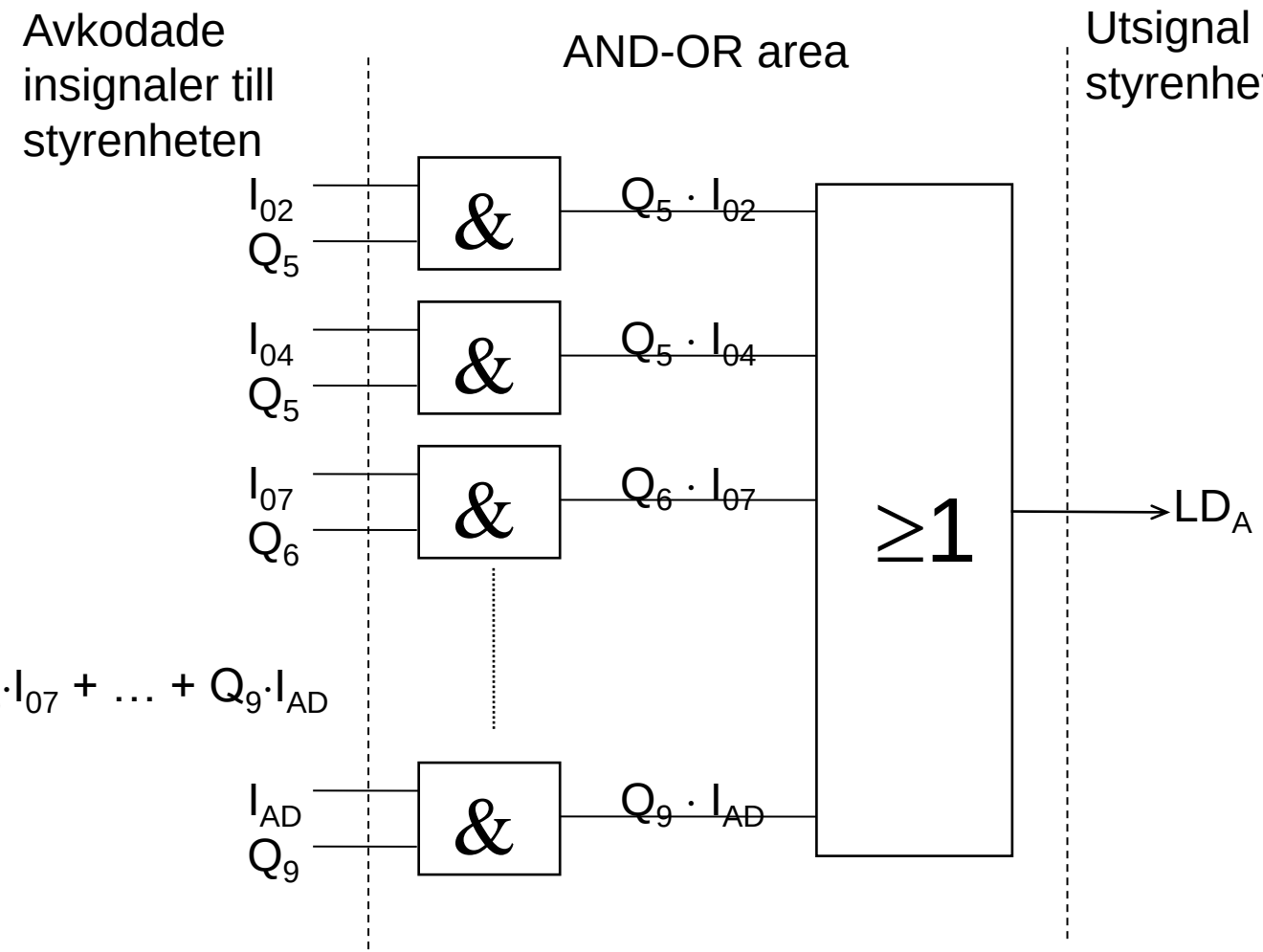


# Arb s 124


Från flagg-registret
C
V
Z
N
C
C'
N
N'
Flagg-villkor
Från instruktions-registret
OP-kods avkodare
Tillstånds-avkodare
Från räknaren
&
≥1
LD_A
NF
Styr-signaler (31 st)

# Styrenheten - forts

Arb s 125




Booleskt uttryck:

$LD_A = Q_5 \cdot I_{02} + Q_5 \cdot I_{04} + Q_6 \cdot I_{07} + \ldots + Q_9 \cdot I_{AD}$

# Dagens mål:

- Ext 20

# Fokus på...

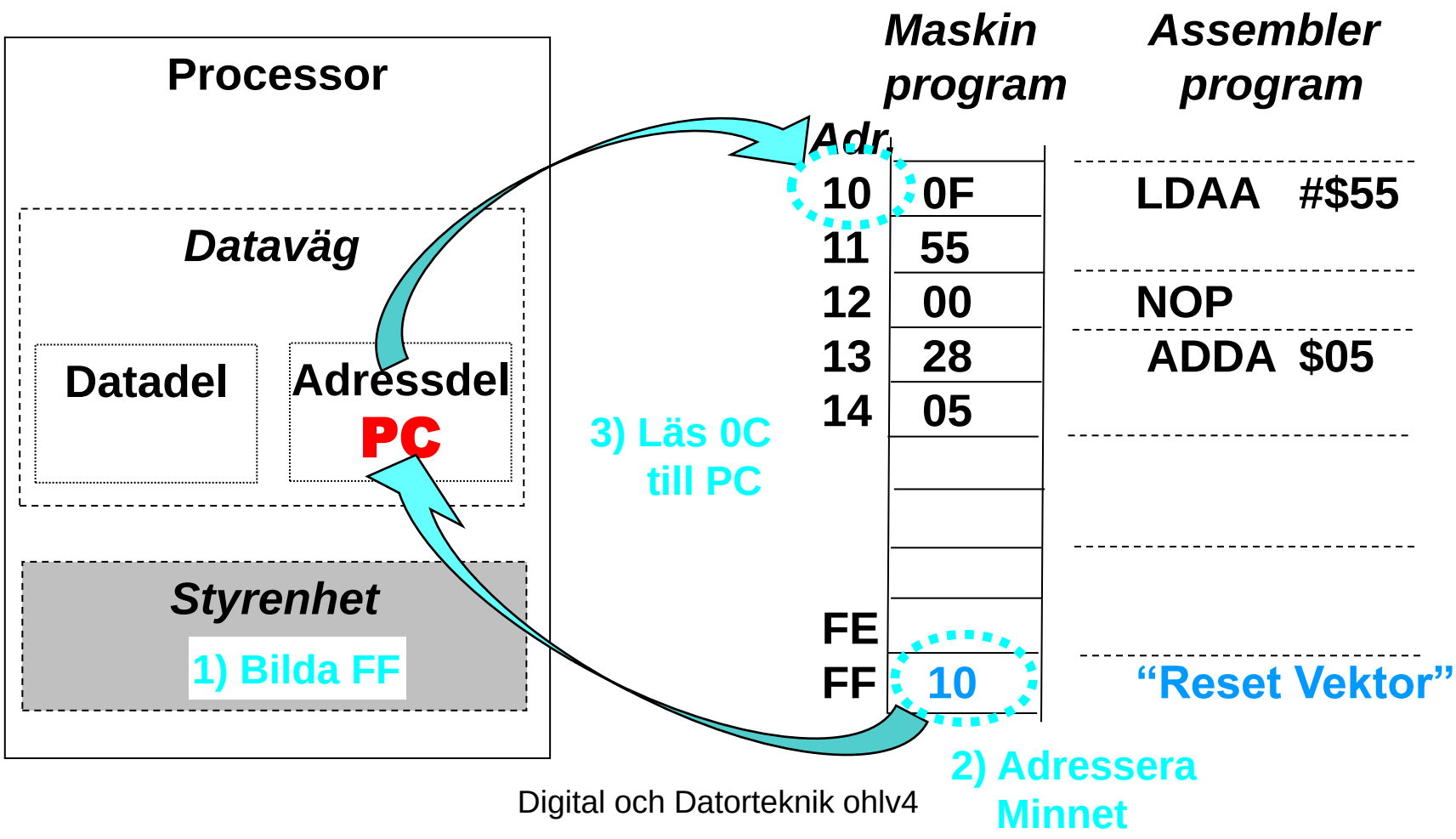
**Dagens mål: Du ska kunna…**

- Kunna använda instruktionslistan och
  - handassemblera / disassemblera program
- Kunna använda olika adreseringsmoder

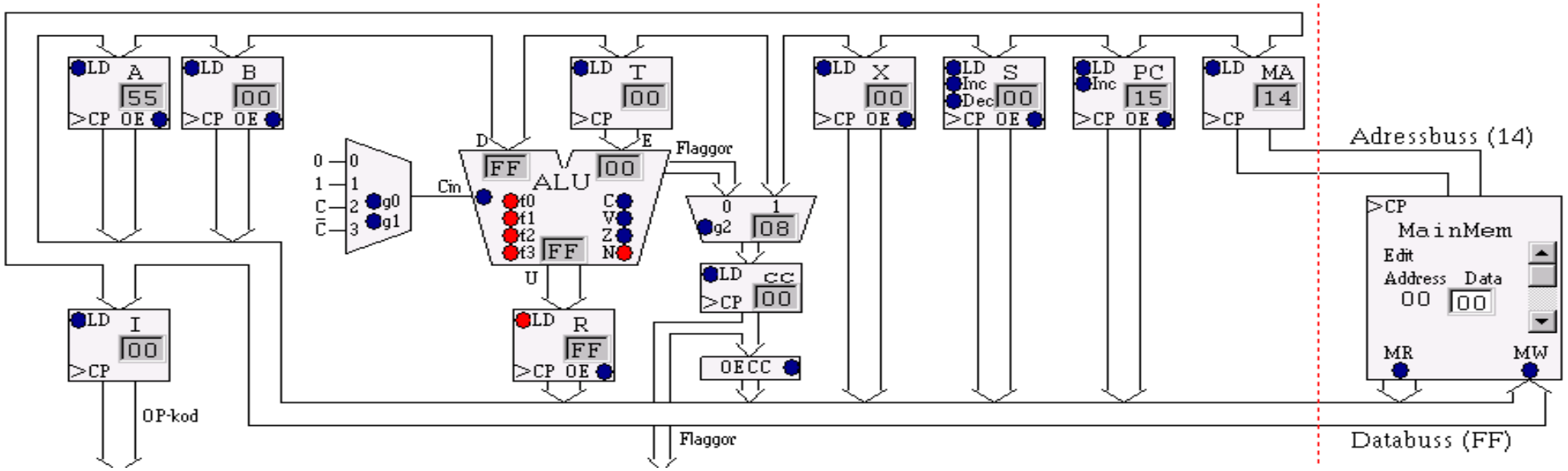
**Läs smart!**
**Lär dig mer!**

# Processorns arbetssätt - RESET

**Arb s 95**



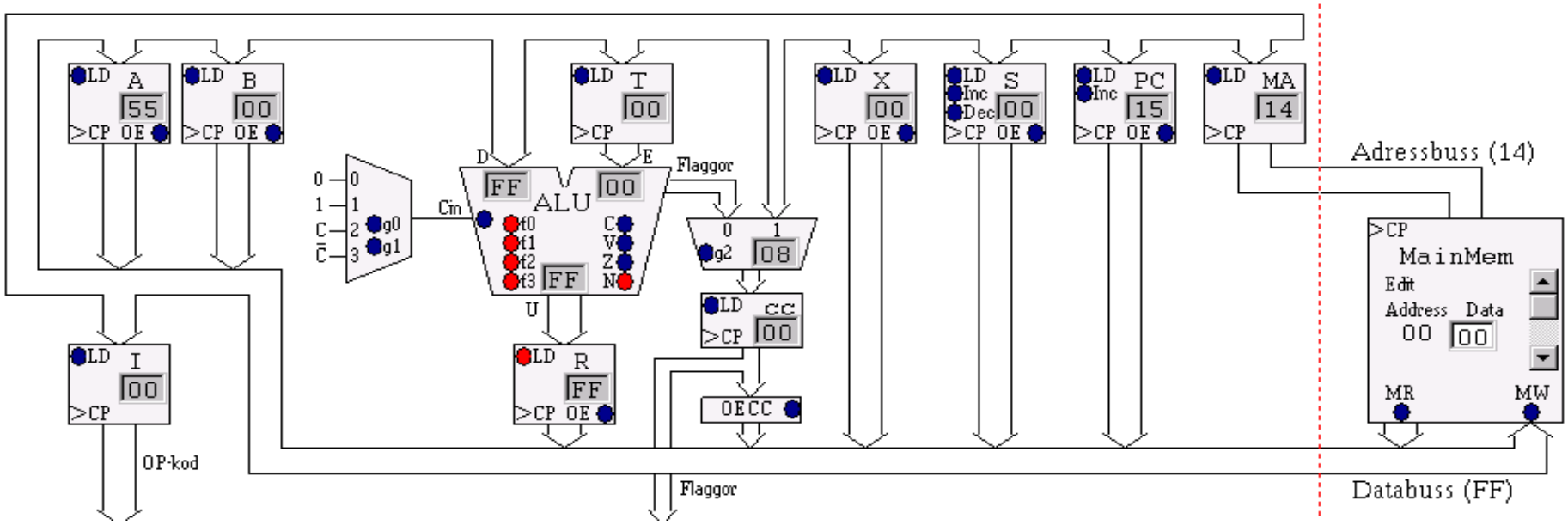
# Processorns arbetssätt - RESET




| State | RTN- | Styrsignaler | Kommentar |
|---|---|---|---|
| 0 | $FF_{16} \rightarrow R$ | ALU-fkn = $F_{16}$, $LD_R$=1. | ALU-funktionen väljs så att talet FF finns på ALU:ns utgång. Laddingången på R-registret ettställs så att utvärdet från ALU'n (FF ) laddas i R-registret vid nästa klockpuls. |
| 1 | R→MA | $OE_R$=1, $LD_{MA}$ =1. | Talet FF i R-registret kopplas ut på bussen. Bussinnehållet laddas i minnesadressregistret vid nästa klockpuls. |
| 2 | M→PC | MR=1, $LD_{PC}$ =1. | Minnesinnehållet på adressen FF läses. Det dataord som läses placeras i PC vid nästa klockpuls. Nästa klockcykel skall vara den första i FETCH-sekvensen. |

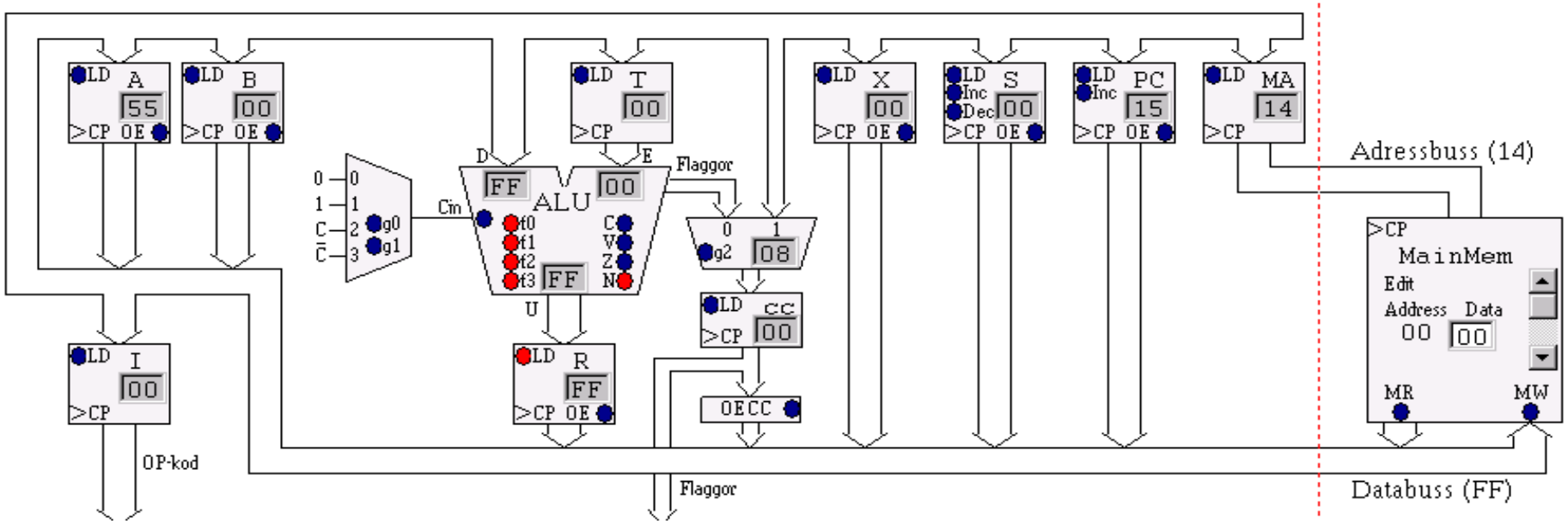
**(Start-tillstånd för RESET-fas)**

# Processorns arbetssätt - RESET




| Tillstånd | Summaterm | RTN-beskrivning | Styrsignaler (=1) |
|---|---|---|---|
| $Q_0$ | $Q_0$ | $FF_{16} \rightarrow R$ | $f_3, f_2, f_1, f_0, LD_R$ |
| $Q_1$ | $Q_1$ | $R \rightarrow MA$ | $OE_R, LD_{MA}$ |
| $Q_2$ | $Q_2$ | $M \rightarrow PC$ | $MR, LD_{PC}$ |

# Processorns arbetssätt - FETCH



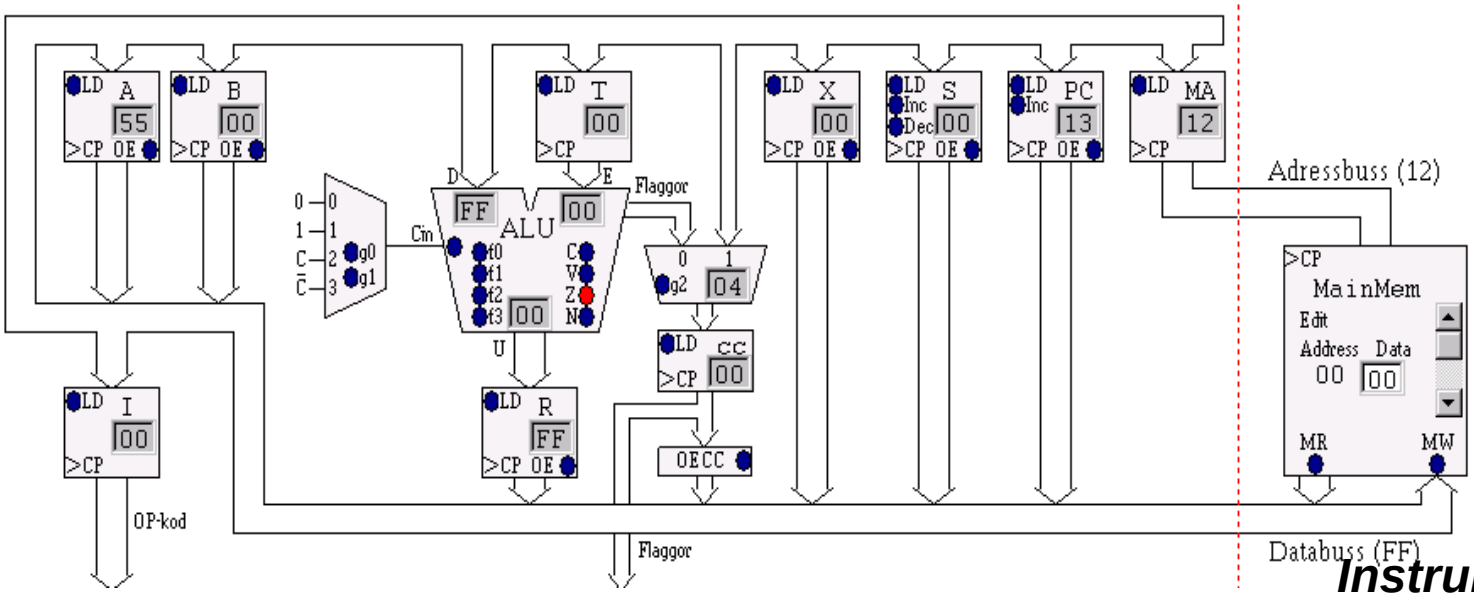

| Tillstånd | Summaterm | RTN-beskrivning | Styrsignaler (=1) |
|---|---|---|---|
| $Q_3$ | $Q_3$ | PC→MA<br>PC+1→PC | $OE_{PC}$, $LD_{MA}$<br>IncPC |
| $Q_4$ | $Q_4$ | M→I | MR, $LD_I$ |

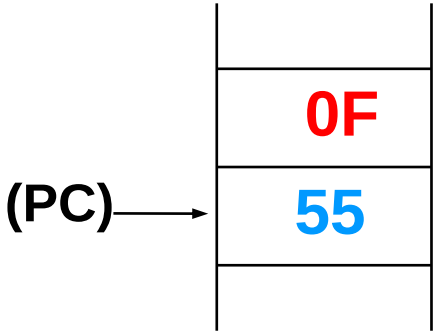
# Fokus på...

## Dagens mål: Du ska kunna…

- Implementera ….
  - RESET i styrenheten
  - FETCH i styrenheten
  - **LDA #Data**
  - **INCA styrenheten**
- Kunna använda instruktionslistan och
  - handassemblera / disassemblera program
- Kunna använda olika adreseringsmoder

**Läs smart!**
**Lär dig mer!**




***Instruktion:*** ***Format:***

**LDAA #Data** **Ord1: OP-kod**

**(LDAA #$55)** **Ord2: Data**

| Tillstånd | Summaterm | RTN-beskrivning | Styrsignaler (=1) |
|---|---|---|---|
| $Q_5$ | $Q_5 * I_{0F}$ | PC→MA<br>PC+1→PC | $OE_{PC}$, $LD_{MA}$<br>IncPC |
| $Q_6$ | $Q_6 * I_{0F}$ | M→A | MR, $LD_A$, NF |

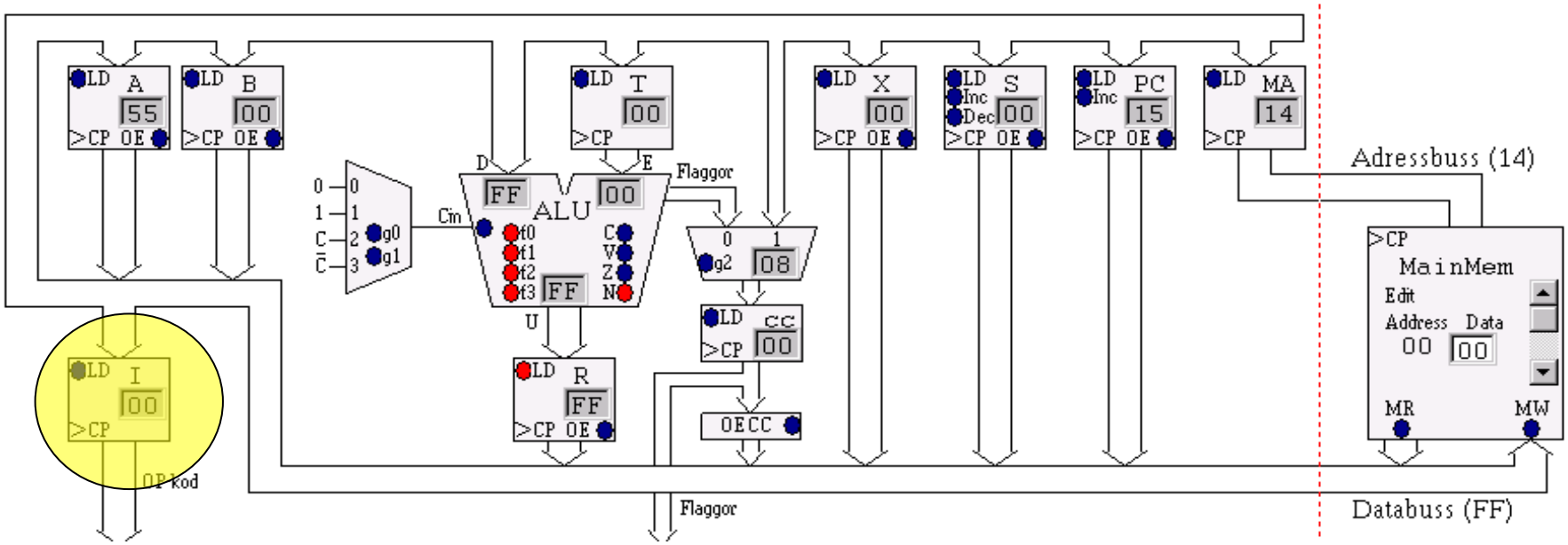
# Att implementera…

Arb s 130

Instruktionslistan; ant byte Ant klock Flaggor OP-kod
Rita fig (Instrux i minnet – data/adressoperand – ingen Operand)

EXECUTE för INCA: (OP-kod = $41_{16}$)

| Tillstånd | Summaterm | RTN-beskrivning | Styrsignaler (=1) |
|---|---|---|---|
| $Q_5$ | $Q_5 \cdot I_{41}$ | | |
| $Q_6$ | $Q_6 \cdot I_{41}$ | | |




| Tillstånd | Summaterm | RTN-beskrivning | Styrsignaler (=1) |
|---|---|---|---|
| Instruktionsnr (OP-kod) AND State xx | | | |
| | | | |
| | | | |
| | I91 * Q5 | Y → Z | |
| | I91 * Q6 | Q → P | |
| | I91 * Q7 | W → U | **NF** |

LDAB #$23
ADDB $F3


## *Arbetsgång:*

- 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.

# Fokus på...

## Dagens mål: Du ska kunna…

- Implementera ….
  - RESET i styrenheten
  - FETCH i styrenheten
  - LDA #Data
  - INCA styrenheten
- **Kunna använda instruktionslistan** och
  - **handassemblera / disassemblera program**
- **Kunna använda olika adresseringsmoder**

**Läs smart!**
**Lär dig mer!**

# Adresseringsmoder

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

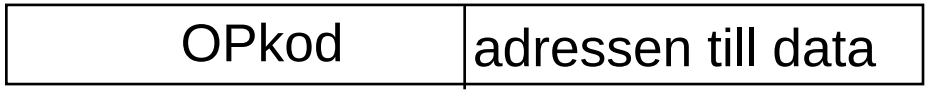
- *Inherent* — *Operanden (data) är inbyggd i Op-koden*


INCA

- *Immediate* — *Operandfältet innehåller data*


LDAA #$12

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

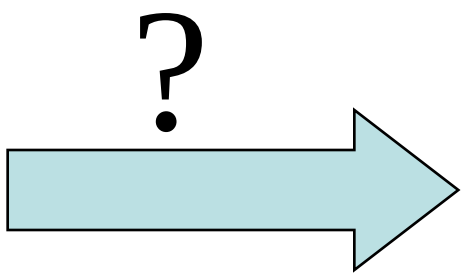



LDAA $12

# Handassemblering

*Assemblerprogram*

```
CLRA
NEGB
SBCB   $0B
LDAA   #$43
```



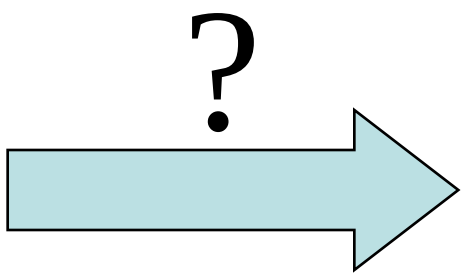

*Maskinprogram*

| Adr | Minne |
|---|---|
| 00h | $47 |
| 01h | $39 |
| 02h | $35 |
| 03h | $0B |
| . | . |
| . | . |
| A6h | $28 |
| A7h | $23 |
| . | . |
| . | |

# Disassemblering

*Maskinprogram*

| Adr | Minne |
|---|---|
| 00h | $47 |
| 01h | $39 |
| 02h | $35 |
| 03h | $0B |
| . | . |
| . | . |
| A6h | $28 |
| A7h | $23 |
| . | . |
| . | |




*Assemblerprogram*

```
CLRA
NEGB
SBCB   $0B
LDAA   #$43
```

# Programexempel för FLEX

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

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

# Programexempel för FLEX
## (Pilla med bitar)

Nollställ bit7, ettställ bit0 och invertera bit4
på talet som finns på minnesadress $1E_{16}$

Programmet skall placeras med start på adress $30_{16}$

# Programexempel för FLEX

Addera talen som finns på minnesadress $20_{16}$ och $21_{16}$.
Placera resultatet på adress $22_{16}$

Programmet skall placeras med start på adress $40_{16}$

# Programexempel för FLEX

Addera de 16-bitars talen P och Q.

P är placerad på minnesadress $20_{16}$ och $21_{16}$.
Q är placerad på minnesadress $22_{16}$ och $23_{16}$.
Placera resultatet på minnesadress $24_{16}$ och $25_{16}$.

Programmet skall placeras med start på adress $50_{16}$

## Veckans mål:

- Subrutin och Stack
- Utvecklingsmiljö (för FLEX)
- IN- och Utmatning (I/O); In och UT-portar
- Skriva program

## Dagens mål:

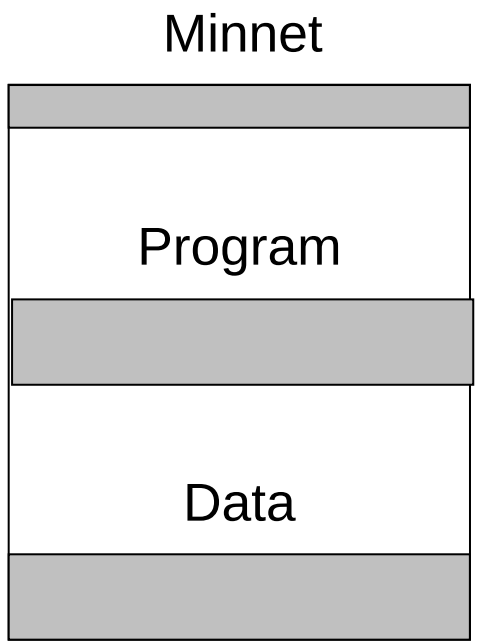
- Skriva mycket enkla assemblerprogram
- Implementera flera instruktioner i styrenheten
- Öva på användning av instruktionslistan
- Utvecklingsmiljön för FLEX

## Du ska kunna…

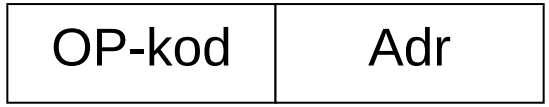
- Adressering via Register X
- Hoppinstruktioner
  - Absoluta hopp JMP
  - Relativa hopp BRA  (Branch)
  - Beräkna branch-offset
- Använda delar av utvecklingsmiljön för FLEX
  - Käll-, list- och laddfil
  - Assemblerdirektiv
  - FLEX-datorn
  - IO-Simulatorer

# Adressering med register X

Arb s 134




LDAA Adr

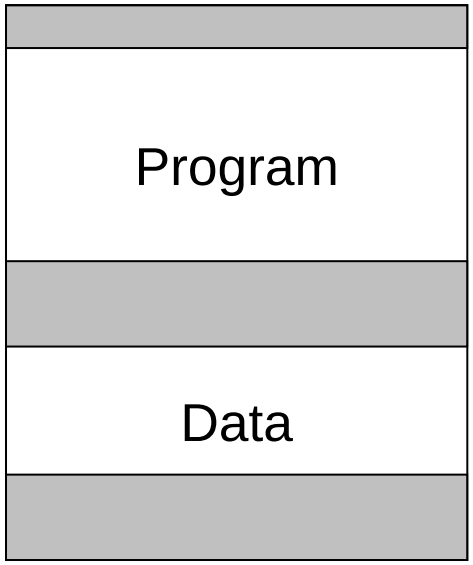


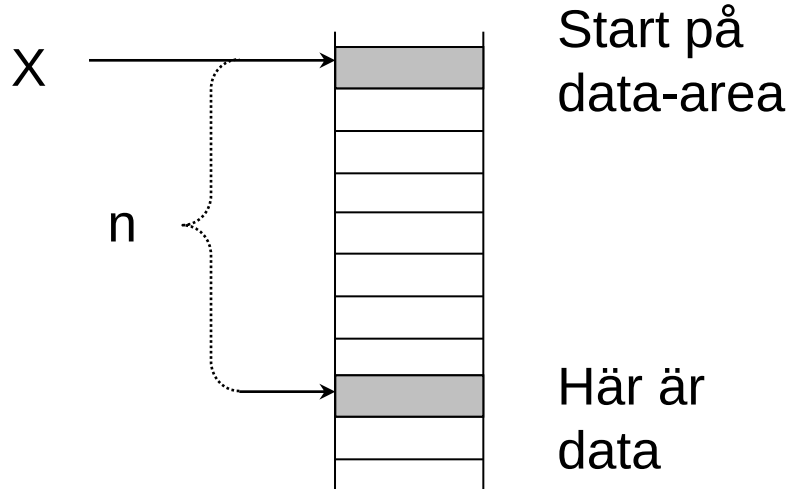

LDAA ,X

# Adressering med register X - forts

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

# Adressering med register X - forts




LDAA n,X

| OP-kod | n |
|---|---|

$M(X+n) \rightarrow A$

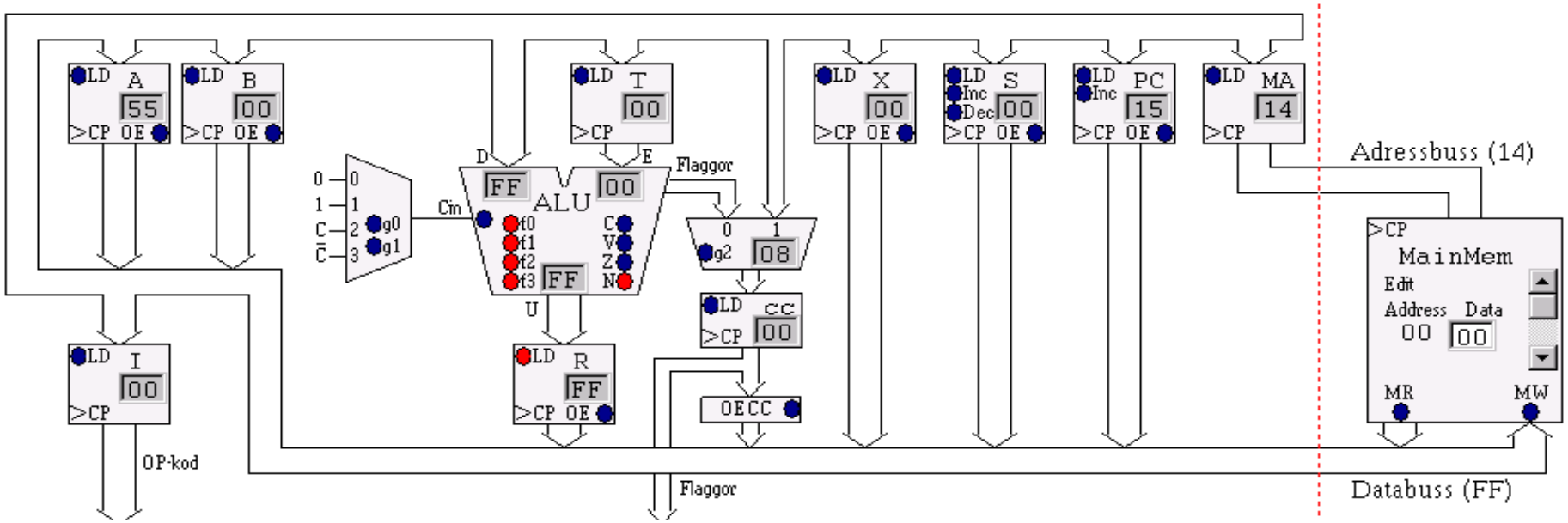
# Uppgift

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.

# Uppgift 106 - forts

Arb s 136




| Tillstånd | Summaterm | RTN-beskrivning | Styrsignaler (=1) |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |


## *Arbetsgång:*

- 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.

**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
- Använda delar av utvecklingsmiljön för FLEX
  - Käll-, list- och laddfil
  - Assemblerdirektiv
  - FLEX-datorn
  - IO-Simulatorer

# Hoppinstruktioner

Arb s 131


*Maskinprogram i minnet* | *Tillhörande assemblerprogram*

| *Adr.* | | | | |
|---|---|---|---|---|
| 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*

*JMP Adr*

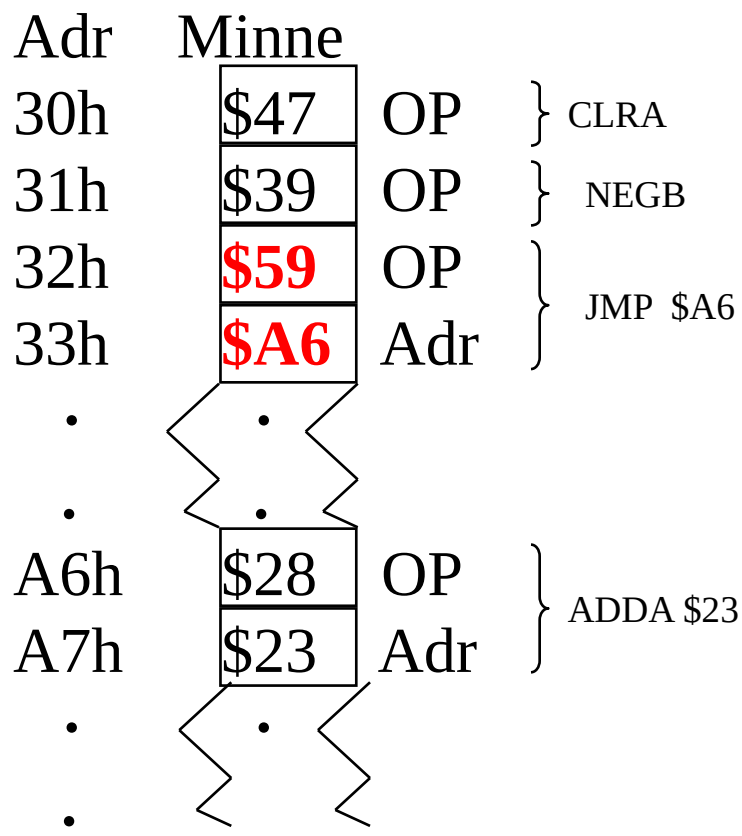
Ex: JMP $A6

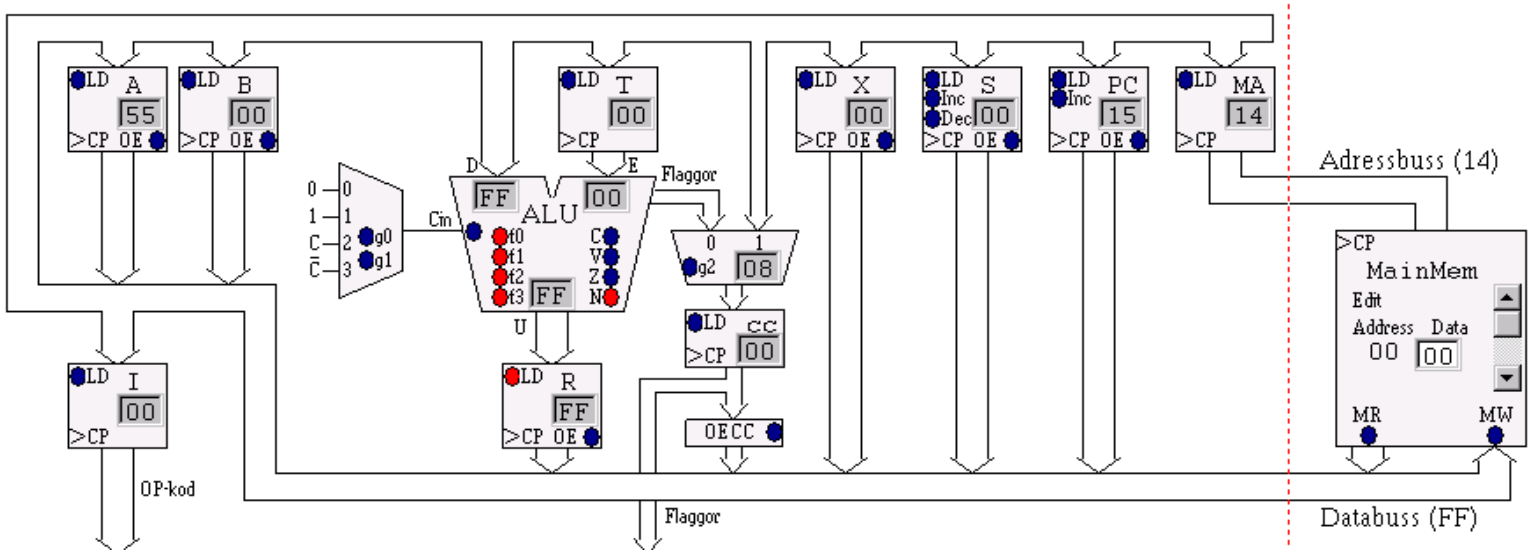
OP-kod: $59

Ant. Byte: 2

RTN: EA → PC

*EA: Effektiva Adressen*






| Tillstånd | Summaterm | RTN-beskrivning | Styrsignaler (=1) |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

**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
- Använda delar av utvecklingsmiljön för FLEX
  - Käll-, list- och laddfil
  - Assemblerdirektiv
  - FLEX-datorn
  - IO-Simulatorer

**Läs smart!**
**Lär dig mer!**

# Relativa hopp

**Arb s 140**

***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 |

**Dagens mål:**

- Skriva mycket enkla assemblerprogram
- Implementera flera instruktioner i styrenheten

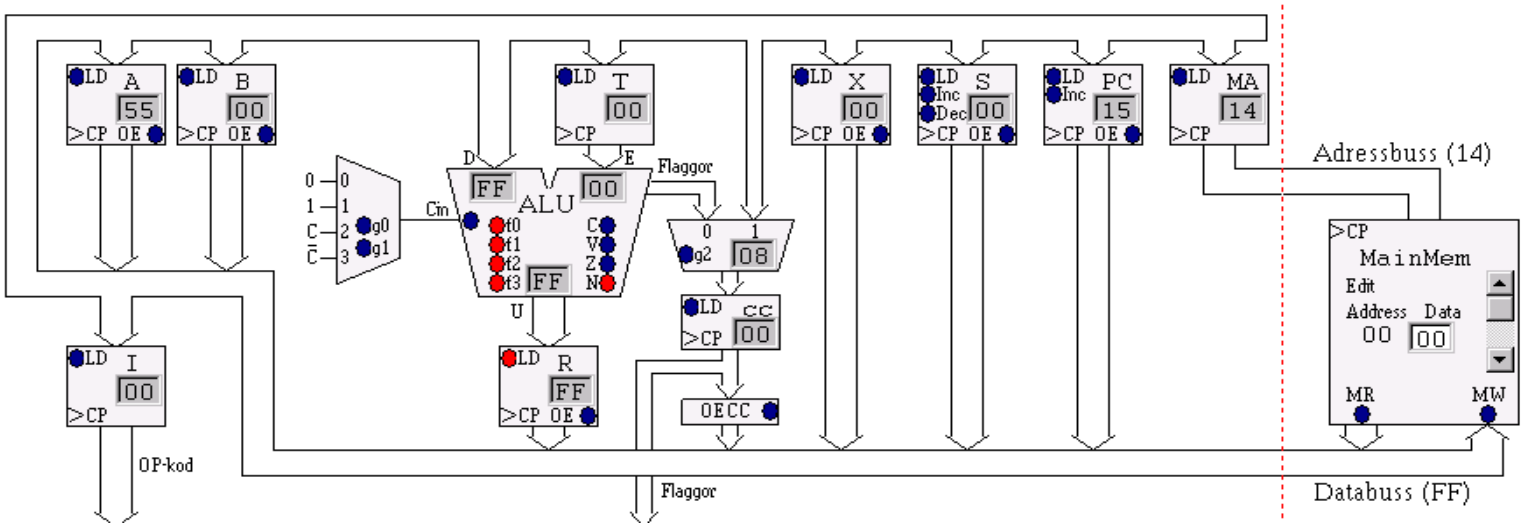
**Du ska kunna…**

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

**Läs smart!**
**Lär dig mer!**

# Uppgift 110 - forts

**Arb s 141**




| Tillstånd | Summaterm | RTN-beskrivning | Styrsignaler (=1) |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

**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

**Läs smart!**
**Lär dig mer!**

# Utvecklingsmiljö för FLEX

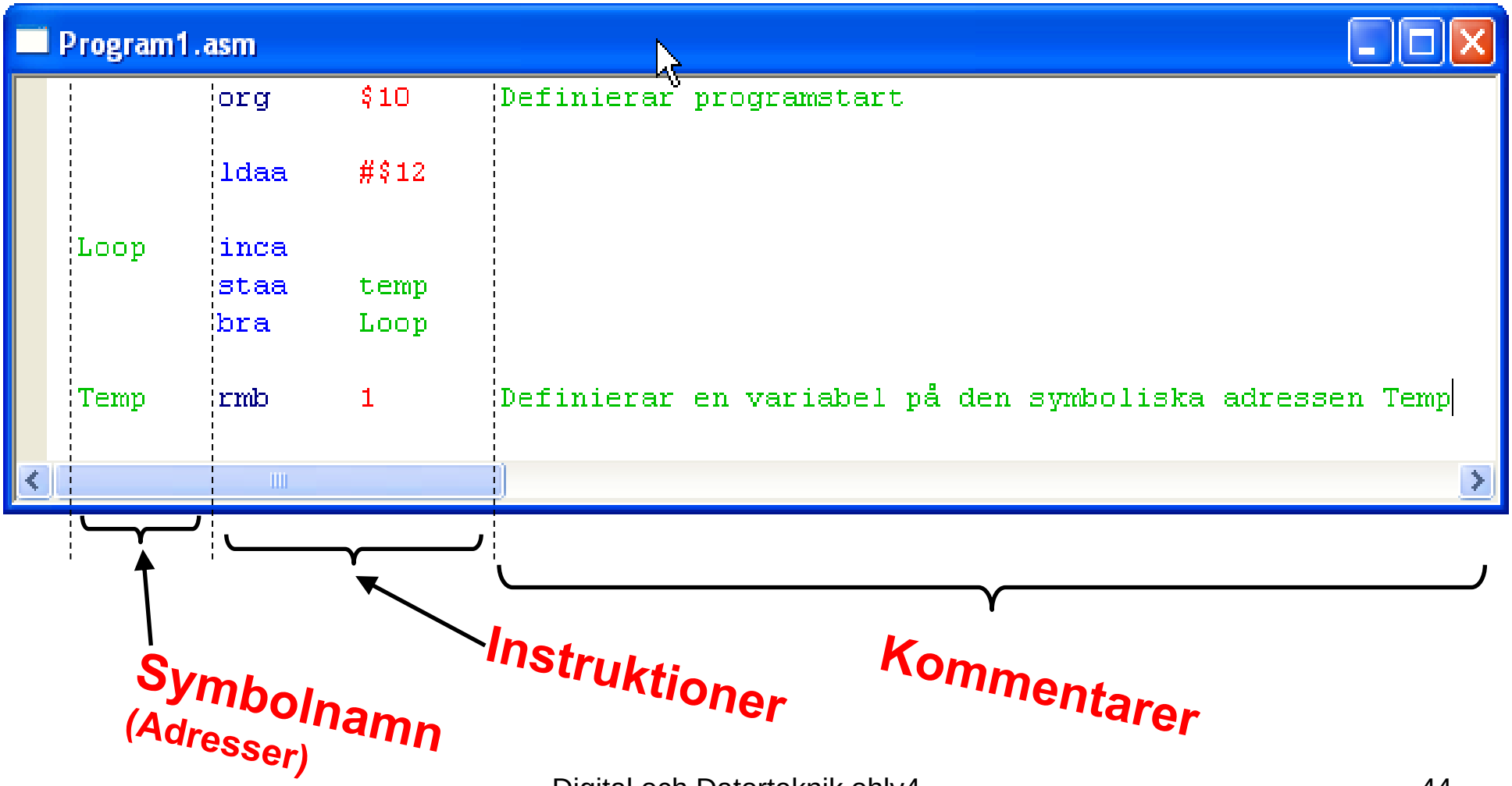
**Arb s 156**

## En utvecklingsmiljö innehåller:

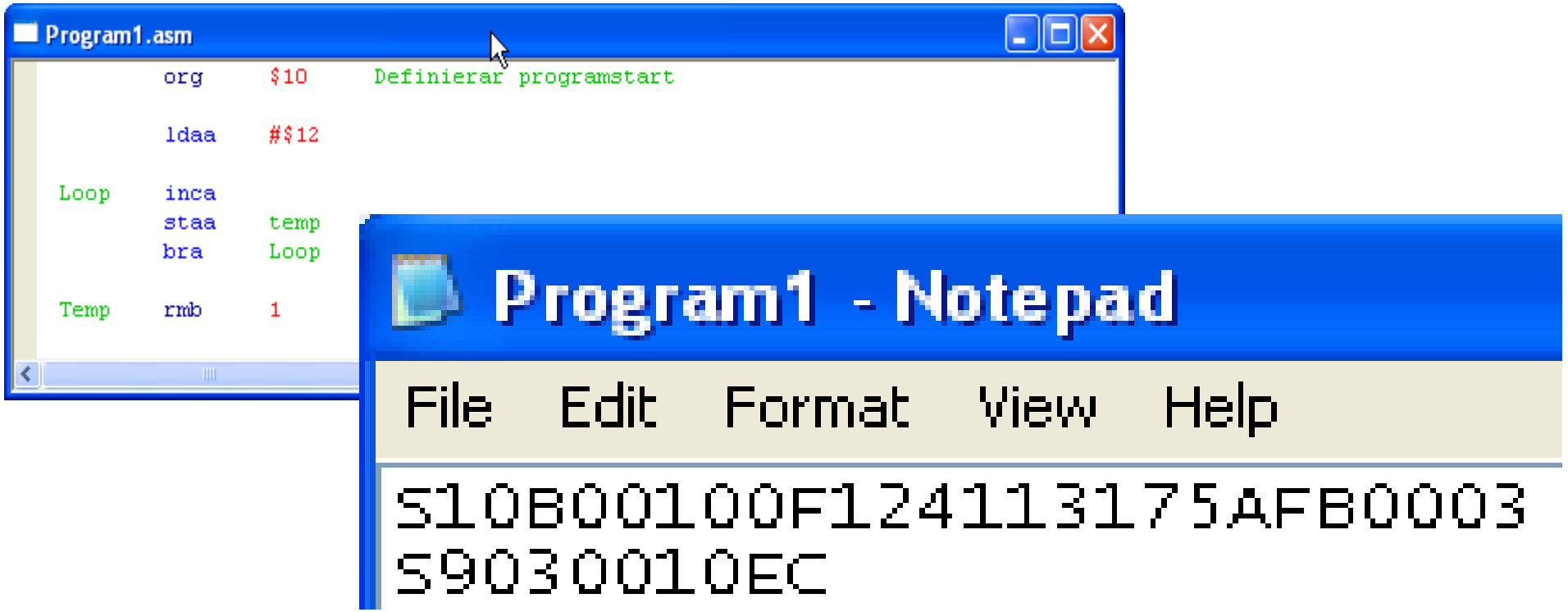
- ✓**Editor**
  - ○**Textredigering**
- ✓**Assemblator (Översätter till maskinkod)**
  - ○**Assemblerdirektiv (Styrinformation till assemblatorn)**
- ✓**Laddare**
  - ○**Flytta maskinkod från utvecklingssystemet till målsystemet**
- ✓**Simulatorer**
  - ○**Processorn**
  - ○**Minnet**
  - ○**I/O**
- ✓**Hjälpsystem**
  - ○**Instruktionslistor etc. etc.**

**ASSEMBLER-PROGRAMMERING**

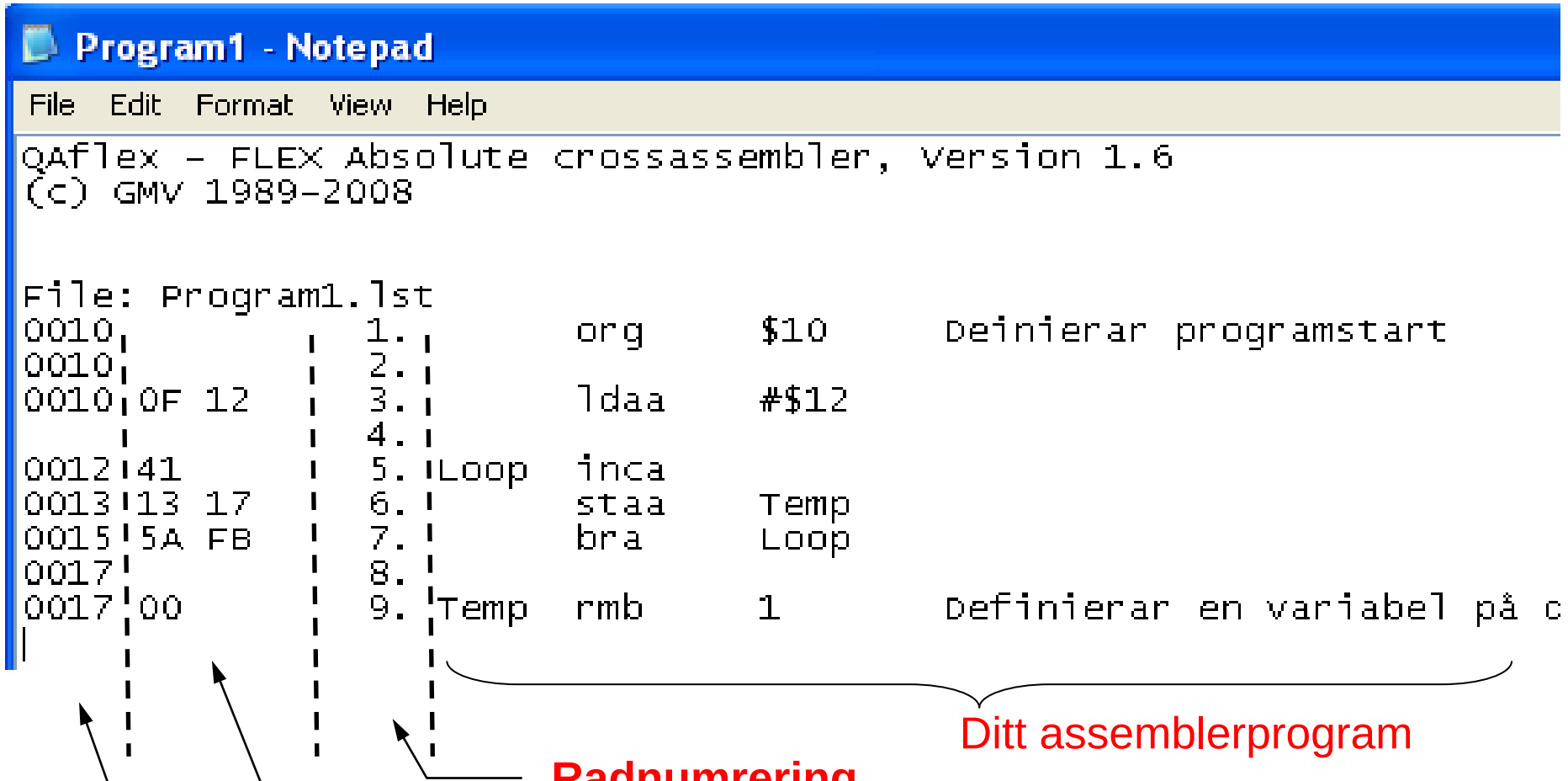
# En källfil


Program1.asm
org $10 Definierar programstart
ldaa #$12
Loop inca
staa temp
bra Loop
Temp rmb 1 Definierar en variabel på den symboliska adressen Temp
Symbolnamn
(Adresser)
Instruktioner
Kommentarer

# En laddfil


Program1.asm
org $10 Definierar programstart
ldaa #$12
Loop inca
staa temp
bra Loop
Temp rmb 1
Program1 - Notepad
File Edit Format View Help
S10B00100F124113175AFB0003
S9030010EC

# En listfil


Program1 - Notepad
File Edit Format View Help
QAflex - FLEX Absolute crossassembler, version 1.6
(c) GMV 1989-2008
File: Program1.lst
0010 1. org $10 Deinierar programstart
0010 2.
0010 0F 12 3. ldaa #$12
4.
0012 41 5. Loop inca
0013 13 17 6. staa Temp
0015 5A FB 7. bra Loop
0017 8.
0017 00 9. Temp rmb 1 Definierar en variabel på c
Ditt assemblerprogram
Radnumrering
Maskinprogram
Minnesadress

Appendix C
Instrux List

# Assemblatordirektiv

**([..] anger valfrihet):**

| | | | |
|---|---|---|---|
| [symbol] | FCB | uttryck[,uttryck[,...]] | (Form Constant Byte) |
| [symbol] | FDB | uttryck[,uttryck[,...]] | (Form Double Byte) |
| [symbol] | FCS | "teckensträng“ | (Form Constant String) |
| [symbol] | RMB | uttryck | (Reserve Memory Bytes) |
| [symbol] | ORG | uttryck | (Origin) |
| Symbol | EQU | uttryck | (Equate) |

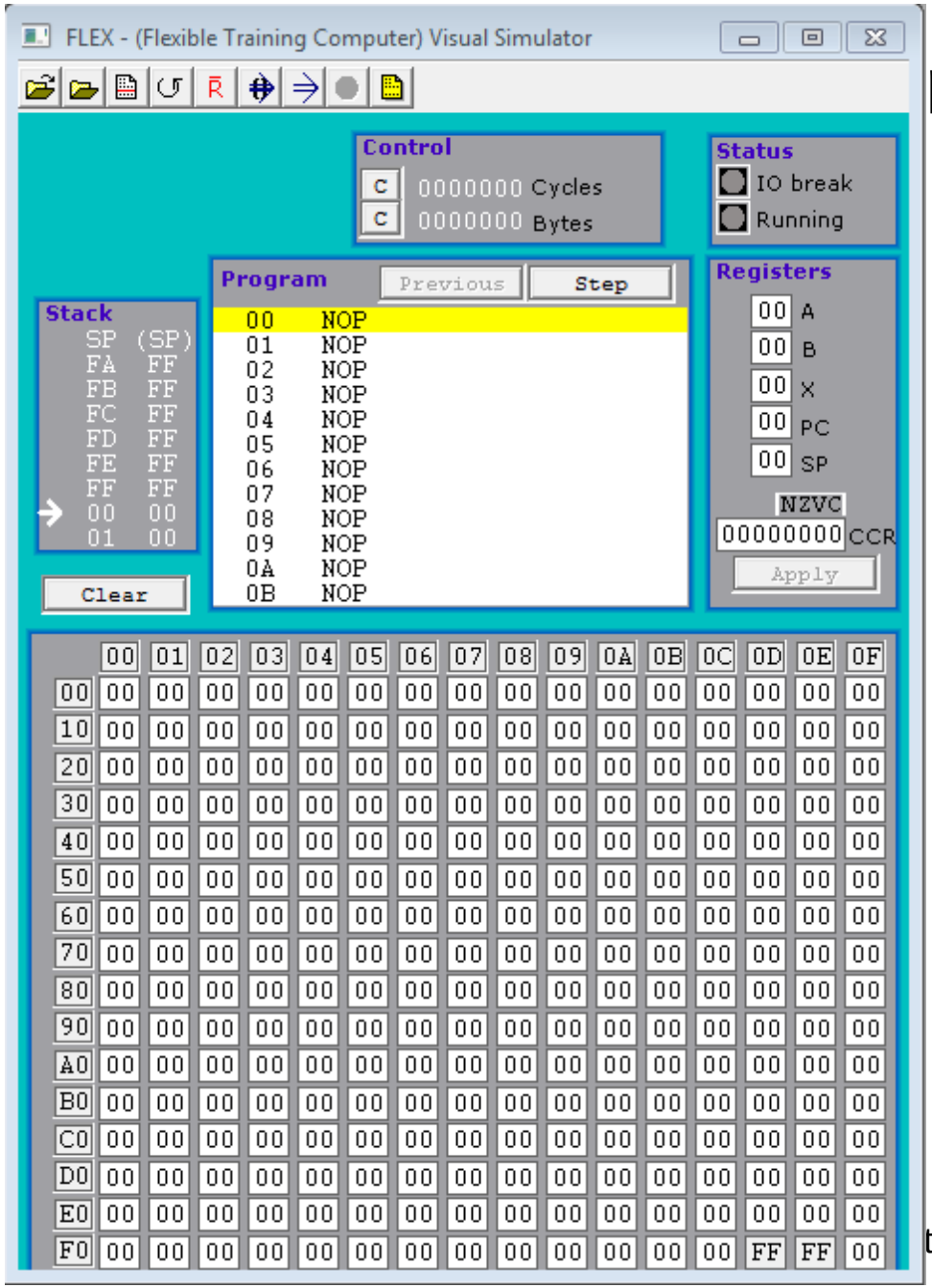
```
          org     $20             Önskad startadress=20(16)
AntVarv   equ     4               Definierar konstanten Antal Varv

          ldx     #TabStart       Startadress för tabell
          stx     Place           Initiera Place
          ldaa    1,X+            Läs tal 1
          staa    Value           Initiera Value
          ldab    #AntVarv        Varvräknare
Loop      ldaa    1,X+            Läs tal i
          cmpa    Value           Jämför med Value
          bls     Lower           Hoppa om lägre
          staa    Value             annars spara tal i
          stx     Place               och dess adress
          dec     Place                 och justera Place
Lower     decb                    Sista talet?
          bne     Loop              hoppa om ej sista

Stop      nop                     Dummy-
          bra     Stop             -snurra

          org     $50             Startadress för tabell
TabStart  fcb     $15,15,%00110111,$98,$11        Test värden
Place     rmb     1               Skapar utrymme för Place
Value     rmb     1                     och Value
```

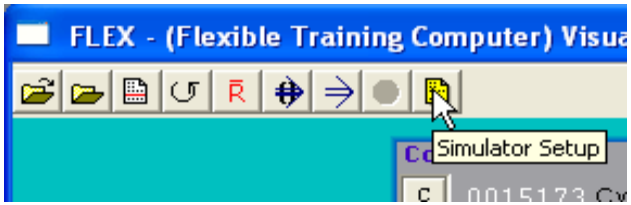
Studera Upg 122

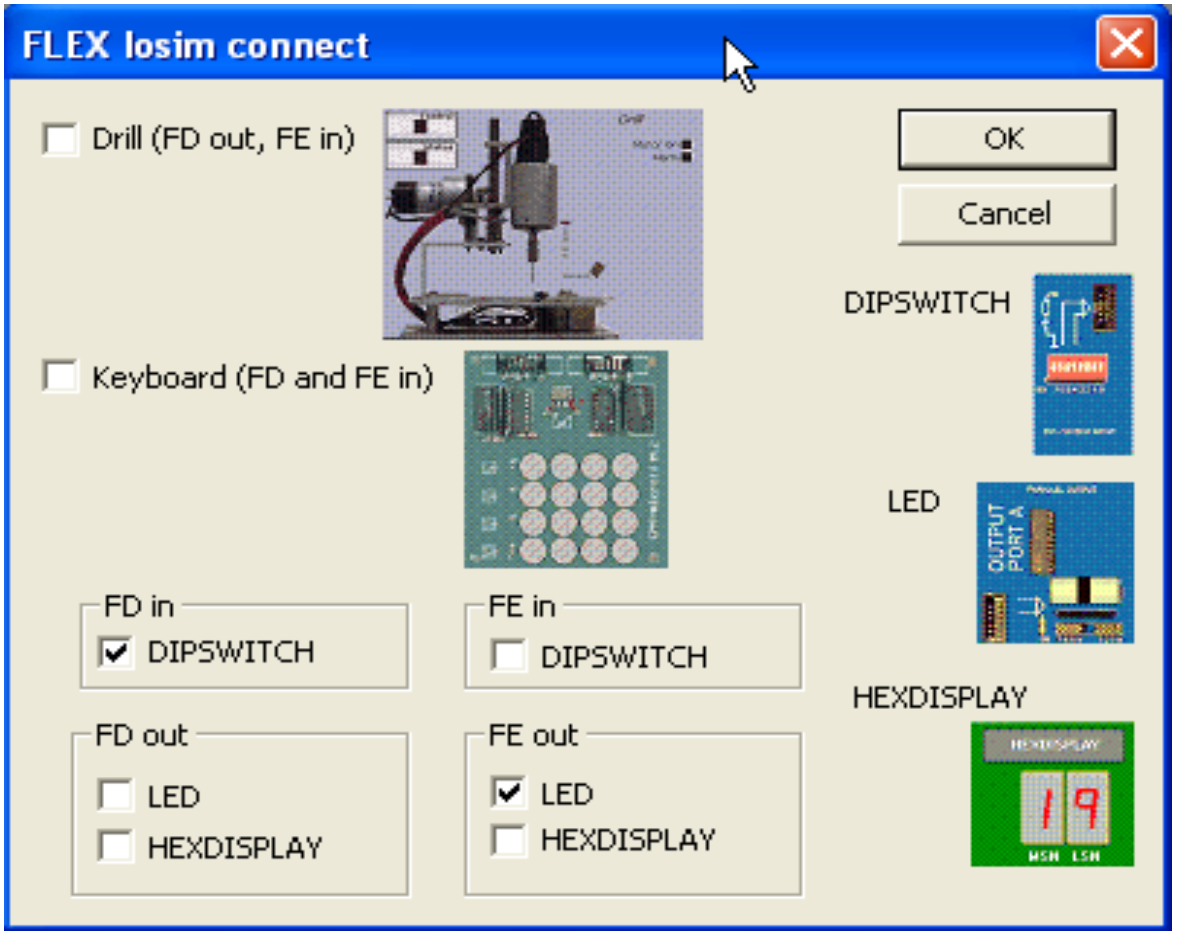



# rn - forts

Arb s 146

# I/O-simulator



# Varför Assemblerprogrammera?

**”Borrprogram”**

- **Positionera borr**
- **Starta borr**
- **Borra genom arbetsstycke**
- **...**

**”Borrprogram”**

```
        STA     BorrStyr
Loop    LDA     BorrStatus
        ANDA    #Mask1
        CMPA    #BorrNere
        BNE     Loop
        ---
```

**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**
- **Använda delar av utvecklingsmiljön för FLEX**
  - **Käll-, list- och laddfil**
  - **Assemblerdirektiv**
  - **FLEX-datorn**
  - **IO-Simulatorer**

**Läs smart!**
**Lär dig mer!**