## *Aktivera Kursens mål:*

- Konstruera en dator mha grindar och programmera denna
- *Använda en modern microcontroller*

## *Aktivera Förra 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 FLEX

## Veckans mål:

- Konstruera styrenheten…. genom att….
- …. implementera olika maskininstruktioner i styrenheten.
- Villkorliga hopp
- Subrutiner och stack
- Skriva enkla program för FLEX
- Introduktion av CPU12

# Dagens mål: Du ska kunna…..

- **Förstå villkorliga hopp i program**
- Implementera BEQ-instruktionen i styrenheten.
- Använda villkorliga hoppinstruktioner
- Förstå begreppen stack, stackpekare och stackinstruktioner
- Implementera PSH-instruktionen i styrenheten.
- Förstå användningen av subrutiner
- Skriva subrutiner


- ”Programmera i FLEX-miljön”

# Villkorliga (Relativa) hopp - forts

Arb s 140

*Instruktionsformat*
BRA Adr

| OP-kod | Offset |
|---|---|

***RTN-beskrivning:***

**PC+Offset → PC**

| 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 |

# Villkorliga hopp:

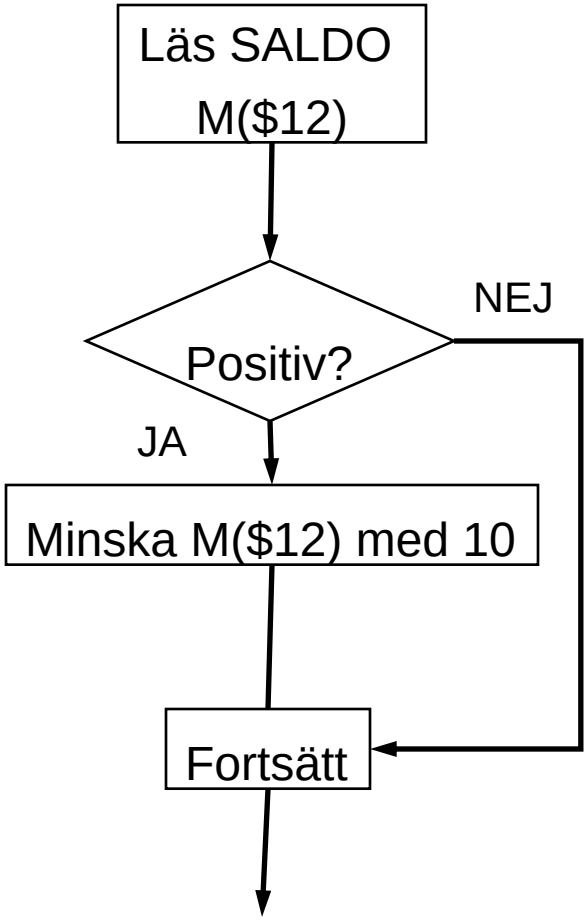
…någon addition…
Om C=1
- så hoppa till felrutin
Annars
- fortsätt beräkningen

Om PIN-koden är korrekt given
- öppna telefonen
Annars
- stäng telefonen

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

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

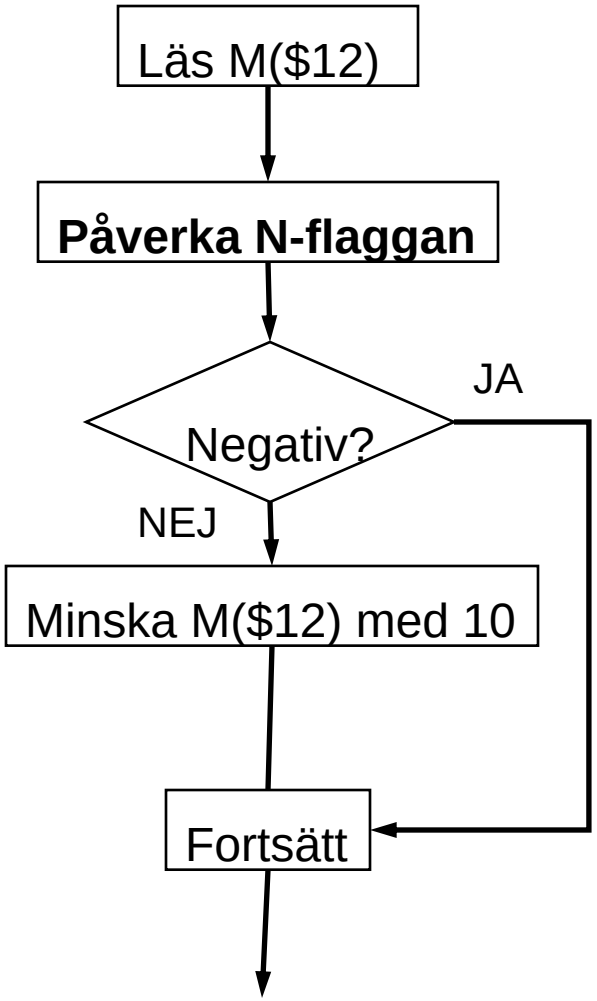
# Villkorliga Hopp -Instruktioner *Branch instruktioner*




if SALDO ≥ 0
    minska SALDO med 10:-
fortsätt

När ska vi hoppa över
“Minska med 10:-” ???

# Villkorliga hopp -Instruktioner




Negativt när
N=1

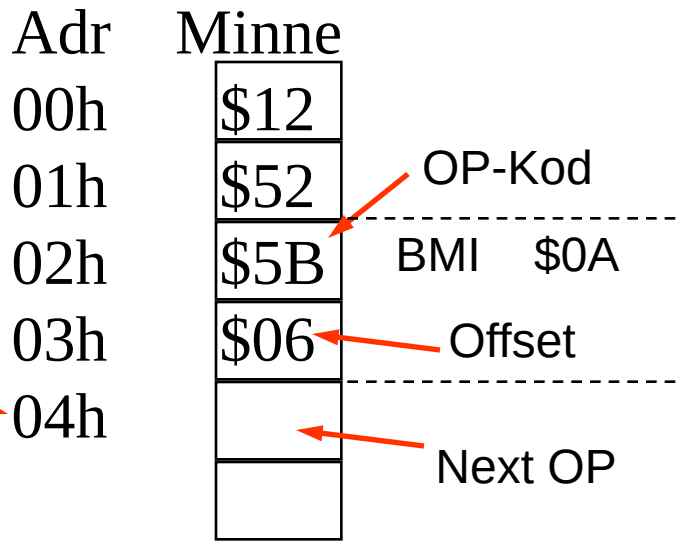
"Testa" SALDO

Lämpliga hoppinstruktioner
BMI eller BPL

# Vad gör processorn vid BMI ?

1) Läser in HELA branch-instruktionen
   (*PC pekar på "nästa" instruktion i minnet*
   *dvs. PC = $04)*

2) Undersöker N-flaggan
   OM N=0
      FETCH på adress $04
   OM N=1
      PC + offset → PC
      **FETCH på adress $0A**




**PC $04**
**+Offset $06**
**=Ny PC $0A**

# Dagens mål: Du ska kunna…..

- Förstå villkorliga hopp i program
- **Implementera BEQ-instruktionen i styrenheten.**
- Använda villkorliga hoppinstruktioner
- Förstå begreppen stack, stackpekare och stackinstruktioner
- Implementera PSH-instruktionen i styrenheten.
- Förstå användningen av subrutiner
- Skriva subrutiner


- ”Programmera i FLEX-miljön”

# Relativa villkorliga hopp – Upg 115

Om Z = 1

PC+Offset → PC, FETCH

Annars

FETCH

Testa först i FLEX-simulatorn

# Dagens mål: Du ska kunna…..

- Förstå villkorliga hopp i program
- Implementera BEQ-instruktionen i styrenheten.
- **Använda villkorliga hoppinstruktioner**
- Förstå begreppen stack, stackpekare och stackinstruktioner
- Implementera PSH-instruktionen i styrenheten.
- Förstå användningen av subrutiner
- Skriva subrutiner


- ”Programmera i FLEX-miljön”

# Villkorliga hopp

Ext 9

Instruktionsuppsättningen för FLEX-processorn har ett antal villkorliga hoppinstruktioner.

De kan indelas i följande tre grupper:

1. **Enkla hoppvillkor.**
2. **Hoppvillkor för tal utan inbyggt tecken.**
3. **Hoppvillkor för tal med inbyggt tecken.**
   **(2-komplementrepresentation)**

# Villkorliga hopp - forts

**1. Enkla hoppvillkor.**

Vid de enkla villkorliga hoppen testas innehållet i
en av flaggvipporna N, Z, V eller C
och hoppet utförs om villkoret är uppfyllt,
dvs den aktuella flaggvippans värde, är 0 resp 1.

# Villkorliga hopp - forts



**2. Hoppvillkor för tal utan inbyggt tecken.**

Förutsätt att flaggorna har påverkats av en subtraktion X - Y enligt:

| | | |
|---|---|---|
| LDA | XVALUE | Läs X från minnet till A |
| CMPA | #Y | Låt skillnaden X - Y påverka flaggorna |
| B(Villkor) | Hoppadress | Utför hoppet om villkoret är uppfyllt |

*X och Y är 8-bitars tal som tillhör intervallet [0, 255].*

# Villkorliga hopp - forts



**2. Hoppvillkor för tal utan inbyggt tecken. *Flaggor C och Z***

$X > Y, \quad X \geq Y, \quad X = Y, \quad X \neq Y, \quad X \leq Y \quad \text{och} \quad X < Y.$

| Relation | Villkorlig hoppinstruktion | | Hoppvillkor |
|---|---|---|---|
| X > Y | BHI | (Branch if X is higher than Y) | C'• Z' |
| X ≥ Y | BHS | (Branch if X is higher or same as Y) | C' |
| X = Y | BEQ | (Branch if X is equal to Y) | Z |
| X ≠ Y | BNE | (Branch if X is not equal to Y) | Z' |
| X ≤ Y | BLS | (Branch if X is lower or same as Y) | (C'• Z')' = C + Z |
| X < Y | BLO | (Branch if X is lower than Y) | C |

# Villkorliga hopp - forts



**3. Hoppvillkor för tal med inbyggt tecken. (2-komplementstal)**

Förutsätt att flaggorna har påverkats av en subtraktion X - Y
enligt:

| | | |
|---|---|---|
| LDA | XVALUE | Läs X från minnet till A |
| CMPA | #Y | Låt skillnaden X - Y påverka flaggorna |
| B(Villkor) | Hoppadress | Utför hoppet om villkoret är uppfyllt |

*X och Y är 8-bitars tal som tillhör intervallet [-128, 127].*

# Villkorliga hopp - forts

**3. Hoppvillkor för tal med inbyggt tecken.**
*X och Y är 8-bitars tal som tillhör intervallet [-128,127].*
***Flaggor N, V och Z***

$$X > Y, \quad X \geq Y, \quad X = Y, \quad X \neq Y, \quad X \leq Y \quad \text{och} \quad X < Y.$$

| Relation | Villkorlig hoppinstruktion | Hoppvillkor HV |
|---|---|---|
| X > Y | BGT (Branch if X is greater than Y) | $(N \oplus V)' \bullet Z'$ |
| X ≥ Y | BGE (Branch if X is greater than or equal to Y) | $(N \oplus V)'$ |
| X = Y | BEQ (Branch if X is equal to Y) | Z |
| X ≠ Y | BNE (Branch if X is not equal to Y) | Z' |
| X ≤ Y | BLE (Branch if X is less than or equal to Y) | $((N \oplus V)' \bullet Z')' = (N \oplus V) + Z$ |
| X < Y | BLT (Branch if X is less than Y) | $N \oplus V$ |

# Uppgift

Två variabler P och Q är lagrade i minnet.
Jämför talen och skriv det största till variabeln R.

P är placerad på adress $20_{16}$ i minnet.
Q är placerad på adress $21_{16}$ i minnet.
R är placerad på adress $22_{16}$ i minnet.

# Exempel IN- o UT-matning

Skriv ett program som hela tiden läser inporten,
om $b_6$ av inporten noll, skriv 7 till utporten
annars skriv 22 till utporten

Använd ML4 In/Ut

Programmets startadress: $60

# Dagens mål: Du ska kunna…..

- Förstå villkorliga hopp i program
- Implementera BEQ-instruktionen i styrenheten.
- Använda villkorliga hoppinstruktioner
- **Förstå begreppen stack, stackpekare och stackinstruktioner**
- Implementera PSH-instruktionen i styrenheten.
- Förstå användningen av subrutiner
- Skriva subrutiner


- ”Programmera i FLEX-miljön”

# STACK och STACKPEKARE

**STACK:** Ett minnesutrymme
Används för att lagra temporära data (registerinnehåll och återhhoppsadresser)

**STACKPEKARE:** Ett register (**Reg S**)
som pekar på det senast ditlagda

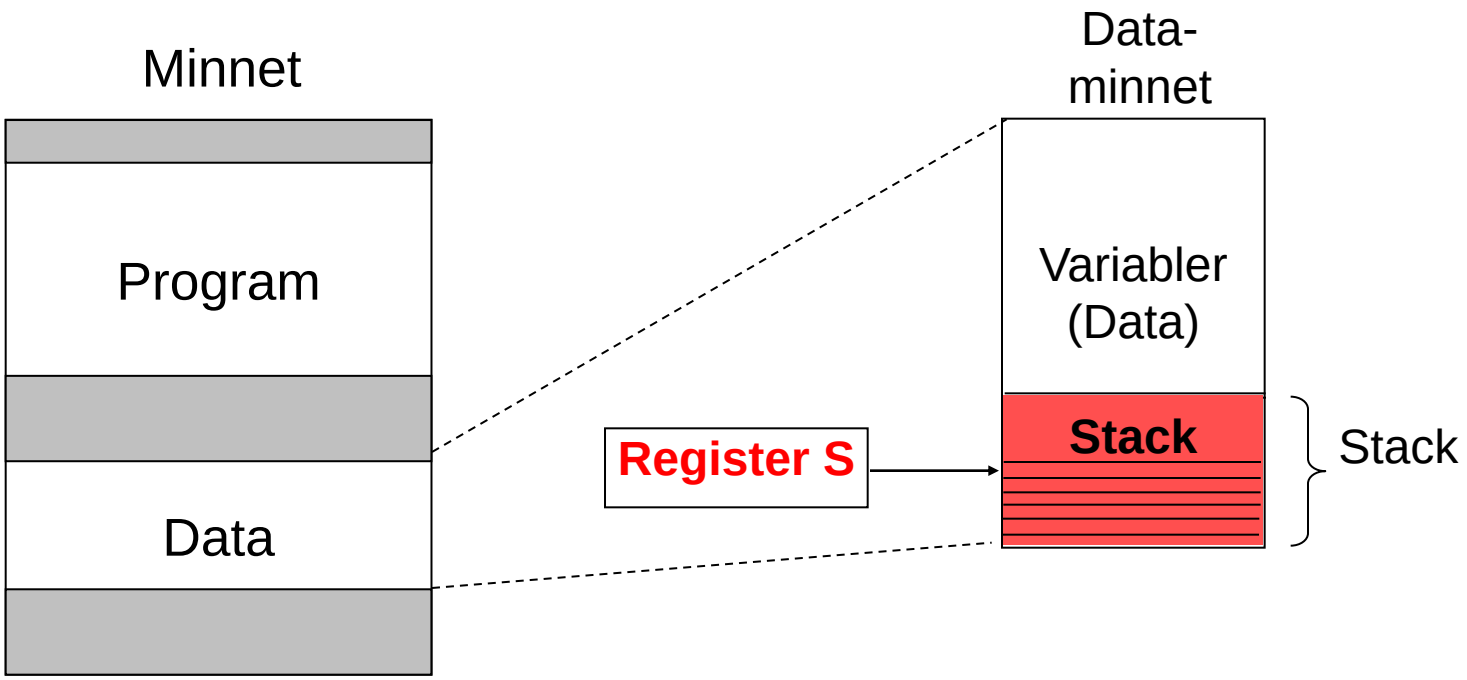
**INSTRUKTIONER:**

**PSH:** Placera ett registerinnehåll ***PÅ*** stacken

**PUL:** Hämta ***FRÅN*** stacken ***TILL*** ett register

# Stacken - forts






***Stacken: ett minnesutrymme som vi temporärt utnyttjar***

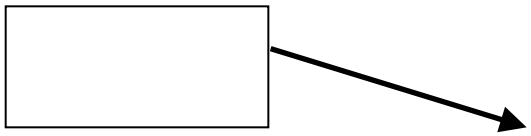
# Stacken – forts – några instruktioner Arb s 137

## PSHA.

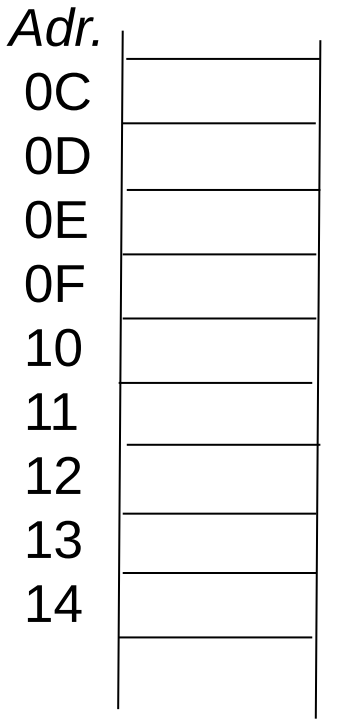
Innehållet i register A skrivs till stacken (till minnet).

RTN-beskrivningen:

1) S-1 →S
2) A→ M(S)




*Innehåll på stacken (i minnet)*




## PULA.

Hämta ett dataord från stacken till register A

RTN-beskrivningen:

1) M(S)→ A
2) S+1→ S

# Uppgift

Definiera en stack som börjar på adress $7F_{16}$.

Placera sedan följande på stacken:
$3B_{16}$, $12_{16}$, $66_{16}$ och $F8_{16}$.

Använd register B

# Dagens mål: Du ska kunna…..

- Förstå villkorliga hopp i program
- Implementera BEQ-instruktionen i styrenheten.
- Använda villkorliga hoppinstruktioner
- Förstå begreppen stack, stackpekare och stackinstruktioner
- **Implementera PSH-instruktionen i styrenheten.**
- Förstå användningen av subrutiner
- Skriva subrutiner


- ”Programmera i FLEX-miljön”

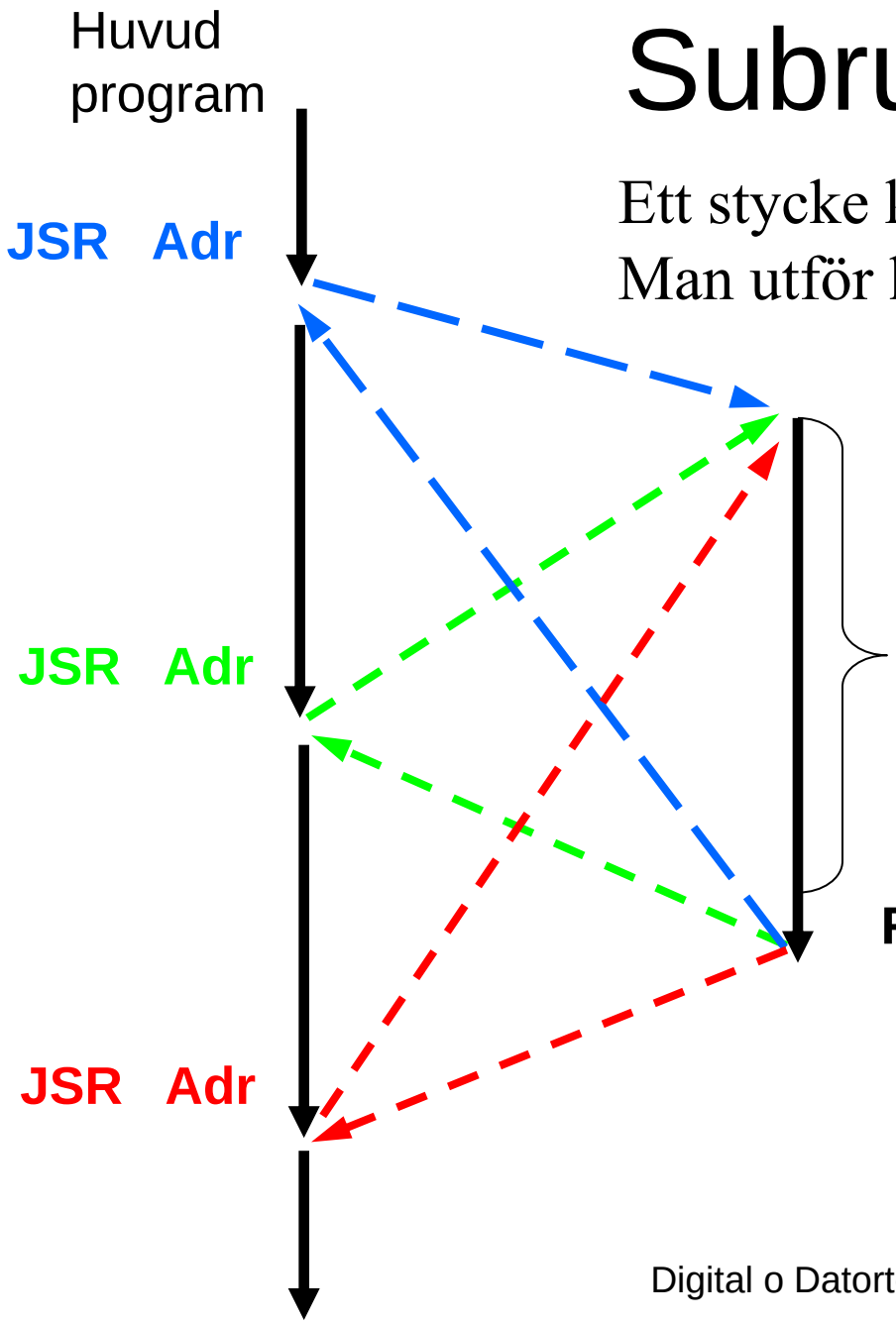
# Dagens mål: Du ska kunna…..

- Förstå villkorliga hopp i program
- Implementera BEQ-instruktionen i styrenheten.
- Använda villkorliga hoppinstruktioner
- Förstå begreppen stack, stackpekare och stackinstruktioner
- Implementera PSH-instruktionen i styrenheten.
- **Förstå användningen av subrutiner**
- Skriva subrutiner

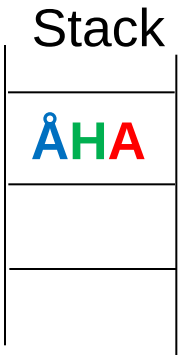

- ”Programmera i FLEX-miljön”

# Subrutiner

Ett stycke kod som ”återanvänds” flera gånger.
Man utför hopp ”till” och ”från” en subrutin




**JSR Adr** Jump to SubRoutine




**RTS** ReTurn from Subroutine

# Dagens mål: Du ska kunna…..

- Förstå villkorliga hopp i program
- Implementera BEQ-instruktionen i styrenheten.
- Använda villkorliga hoppinstruktioner
- Förstå begreppen stack, stackpekare och stackinstruktioner
- Implementera PSH-instruktionen i styrenheten.
- Förstå användningen av subrutiner
- **Skriva subrutiner**

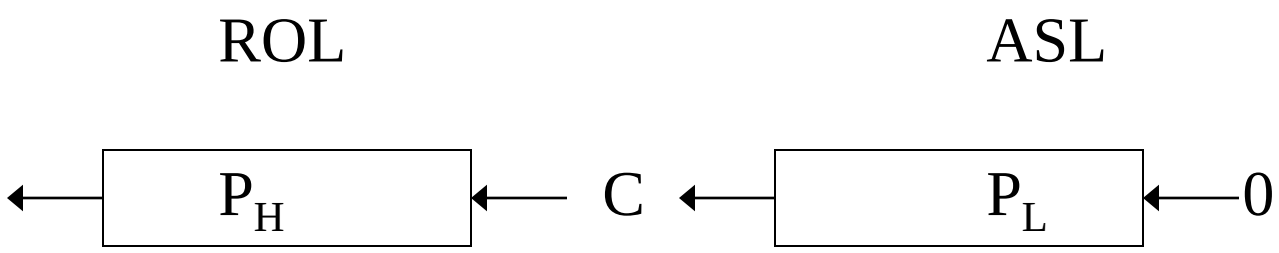

- ”Programmera i FLEX-miljön”

# Subrutin o stack - forts

Multiplicera en 16-bitars variabel på adress $20 med två.

| Adr | Minne |
|---|---|
| 20 | $P_H$ |
| 21 | $P_L$ |

| | |
|---|---|
| 3 | 0011 ← |
| 6 | 0110 ← |
| 12 | 1100 |



| Adr | Minne | Assembler prog | | |
|---|---|---|---|---|
| $80 | $3D | ASL | $21 | $2*P_L$ |
| $81 | $21 | | | |
| $82 | $40 | ROL | $20 | $2*P_H$ |
| $83 | $20 | | | |
| $84 | $6A | RTS | | |
| $85 | $69 | JSR | $80 | Mul2 |
| $86 | $80 | | | |
| $87 | $69 | JSR | $80 | Mul2 |
| $88 | $80 | | | |
| $89 | 6A | RTS | | |
| $8A | | | | |

# Dagens mål: Du ska kunna…..

- Förstå villkorliga hopp i program
- Implementera BEQ-instruktionen i styrenheten.
- Använda villkorliga hoppinstruktioner
- Förstå begreppen stack, stackpekare och stackinstruktioner
- Implementera PSH-instruktionen i styrenheten.
- Förstå användningen av subrutiner
- Skriva subrutiner


- ”Programmera i FLEX-miljön”

## Veckans mål:

- Konstruera styrenheten…. genom att….
- …. implementera olika maskininstruktioner i styrenheten.
- Villkorliga hopp
- Subrutiner och stack
- Skriva enkla program för FLEX
- Introduktion av CPU12

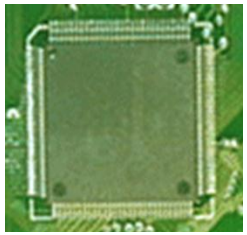
## Dagens mål:  Du ska kunna….

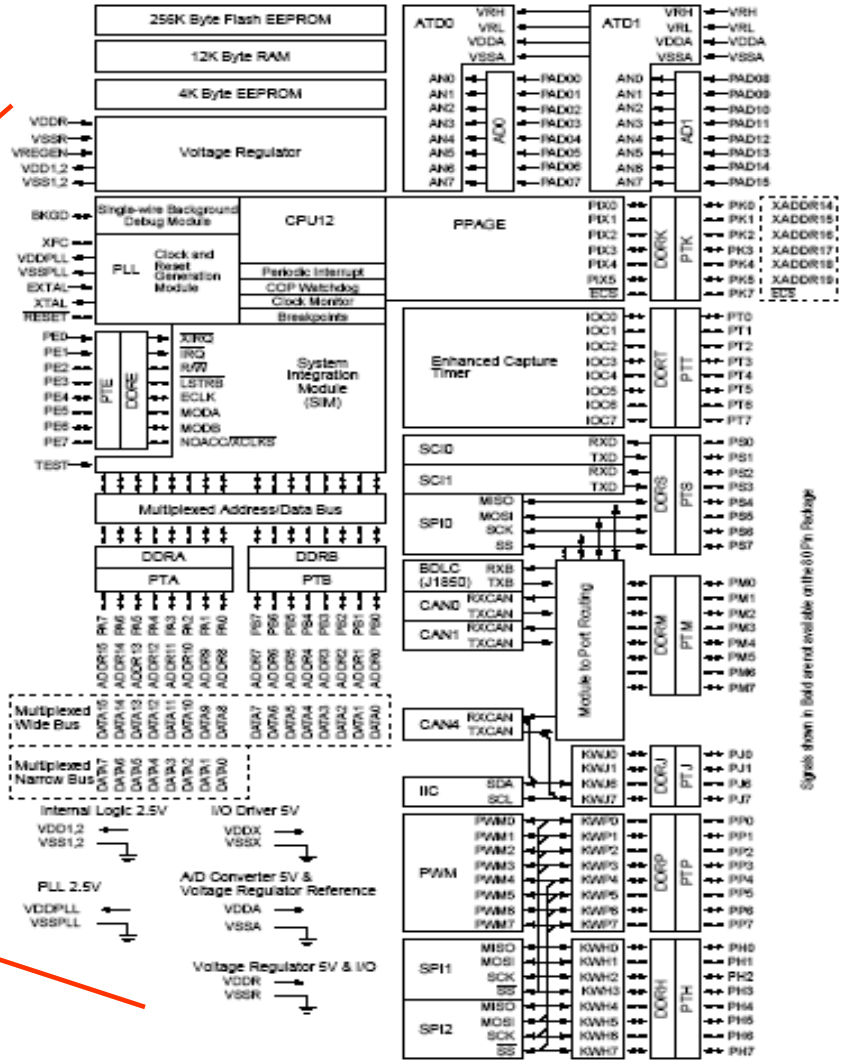
- Beskriva likheter o olikheter mellan FLEX och CPU12
- Använda Instruktionslistan för CPU12
  - Instruktionsgrupper
  - Adresseringsmoder
- Skriva enkla program för CPU12
- Använda delar av utvecklingsmiljön Eterm för MC13

# Dagens mål: Du ska kunna….

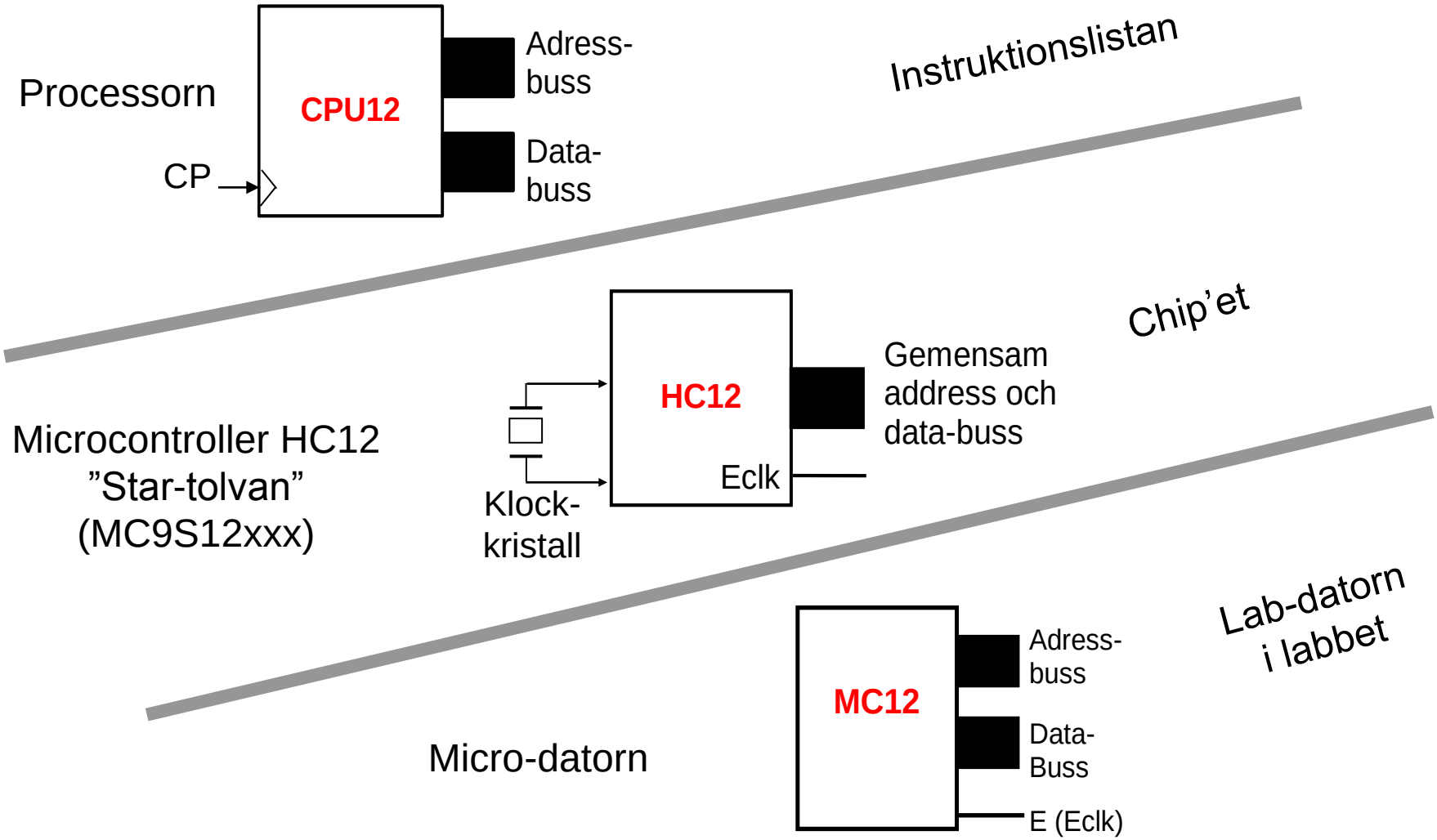
- **Beskriva likheter o olikheter mellan FLEX och CPU12**
- Använda Instruktionslistan för CPU12
  - Instruktionsgrupper
  - Adresseringsmoder
- Skriva enkla program för CPU12
- Använda delar av utvecklingsmiljön Eterm för MC12

# CPU12 / HC12 / MC12



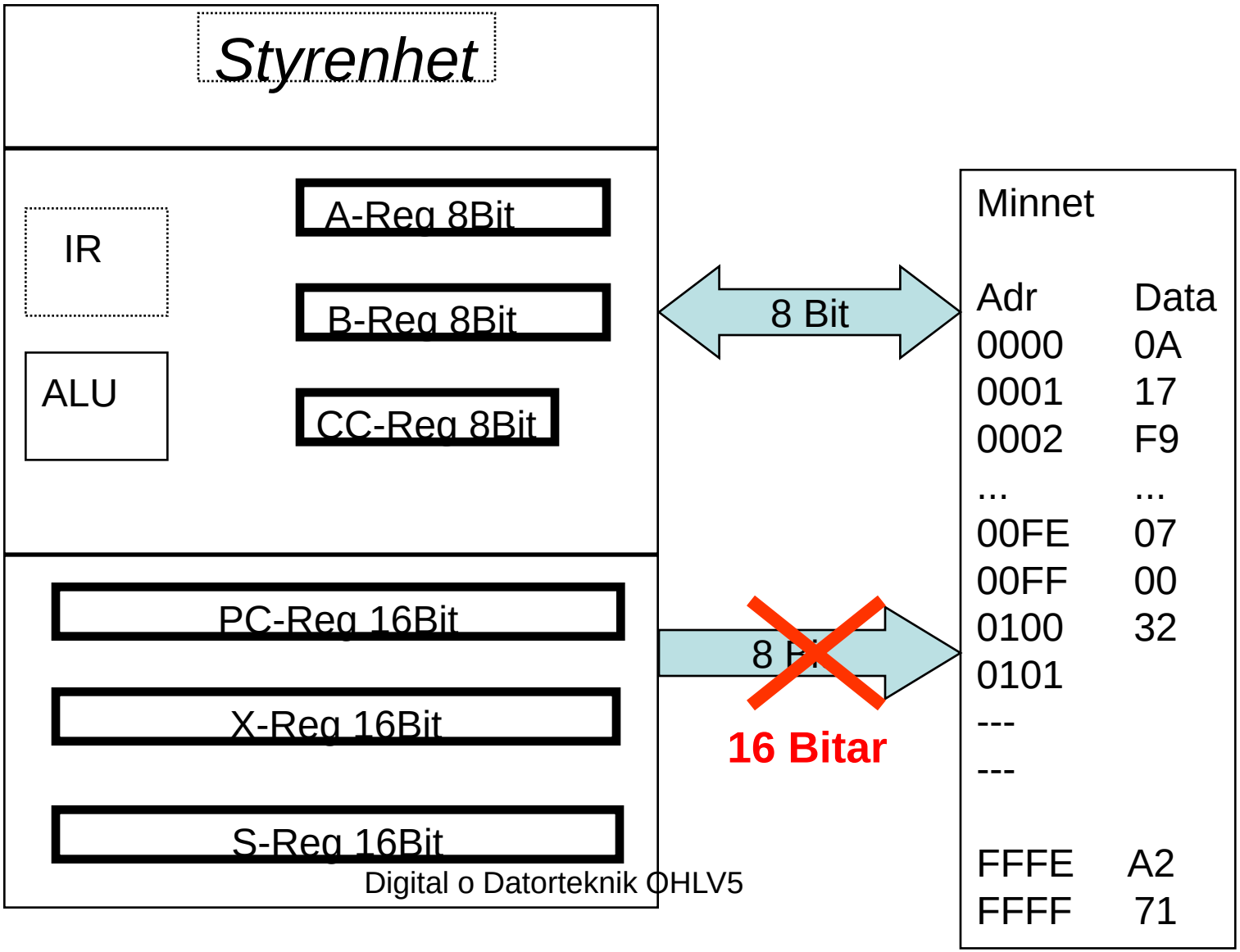


# CPU12 / HC12 / MC12

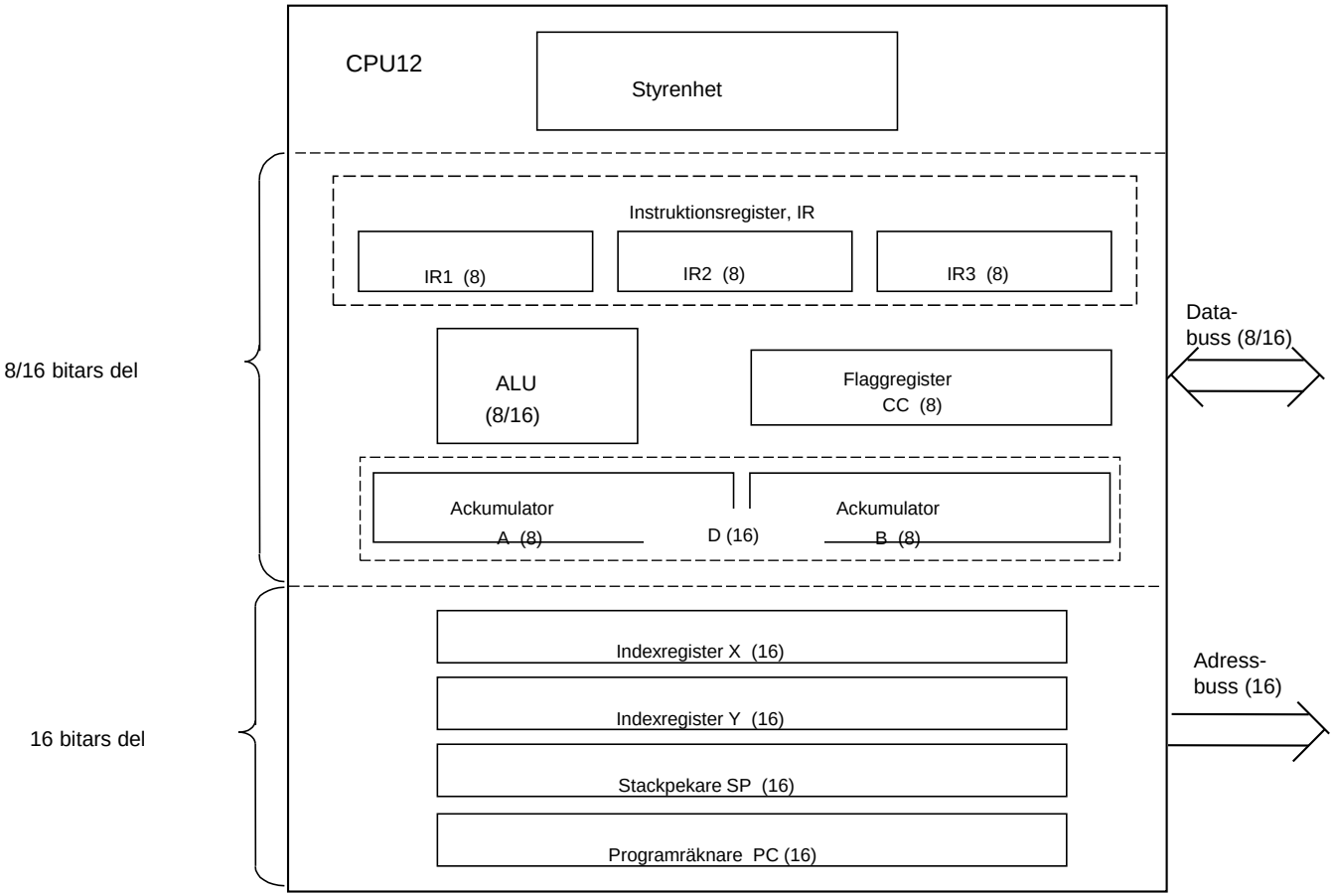


# ~~FLEX~~ CPU12

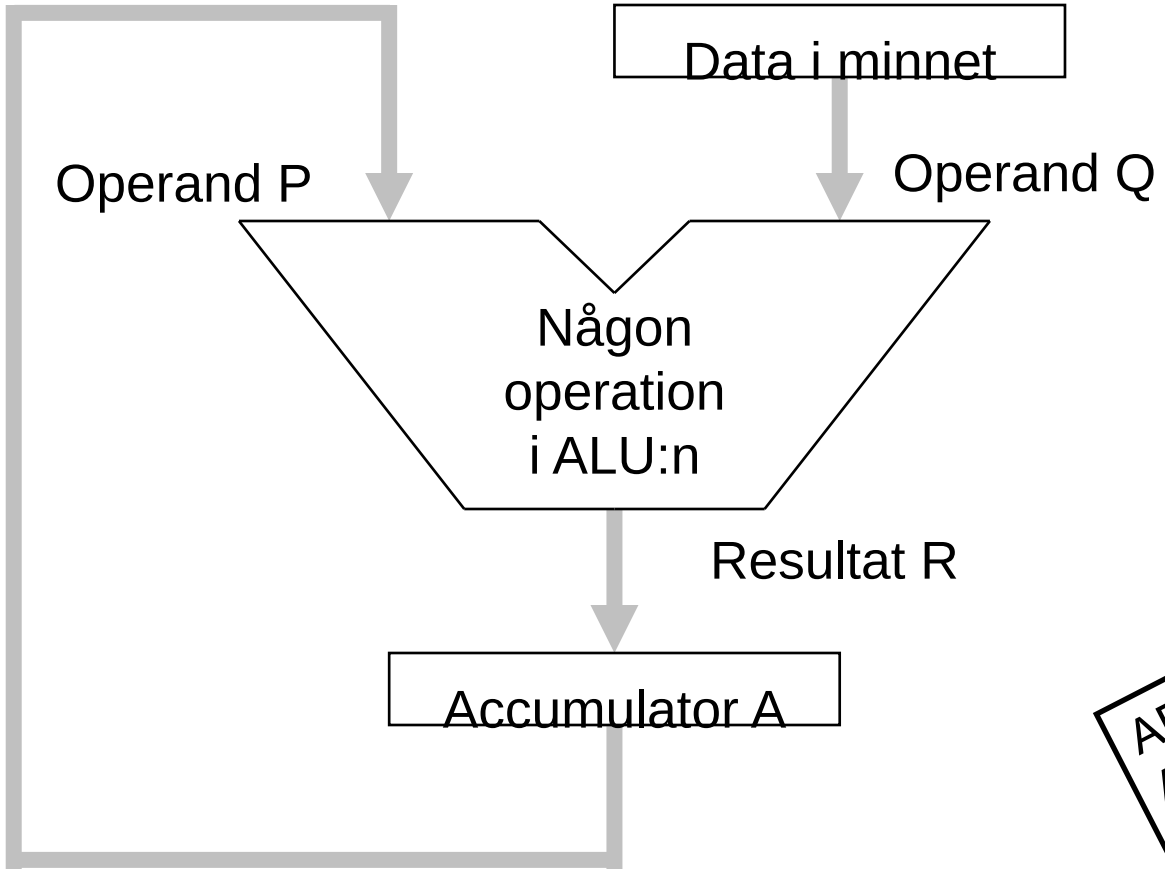
Programmerarens bild.

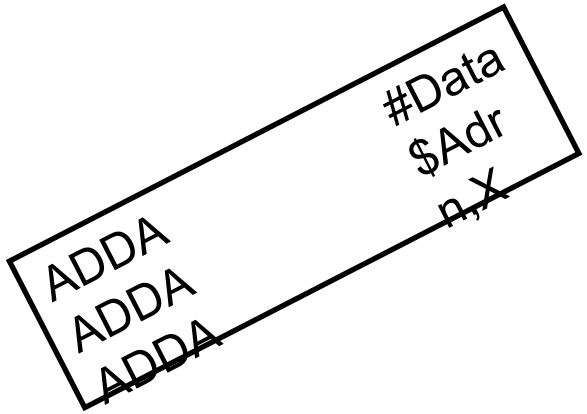


# Programmerarens bild CPU12



# CPU12 (o FLEX) - En Ackumulatormaskin




**$Accumulator_K$ OP Minne → $Accumulator_K$**

# FLEX:
# en 1-adress –maskin

*Flytta data i minnet:*

```
LDAA   $Adr1
STAA   $Adr2
```

*Stoppa in data i minnet:*

```
LDAA   #Data
STAA   $Adr
```

# CPU12:
# Även
# 2-adress
# maskin

*Flytta data (byte) i minnet:*

```
MOVB   $Adr1,$Adr2
```

*Stoppa in data (byte) i minnet:*

```
MOVB   #Data,$Adr
```

*Även 16-bitars:* `MOVW  $Adr1,$Adr2`

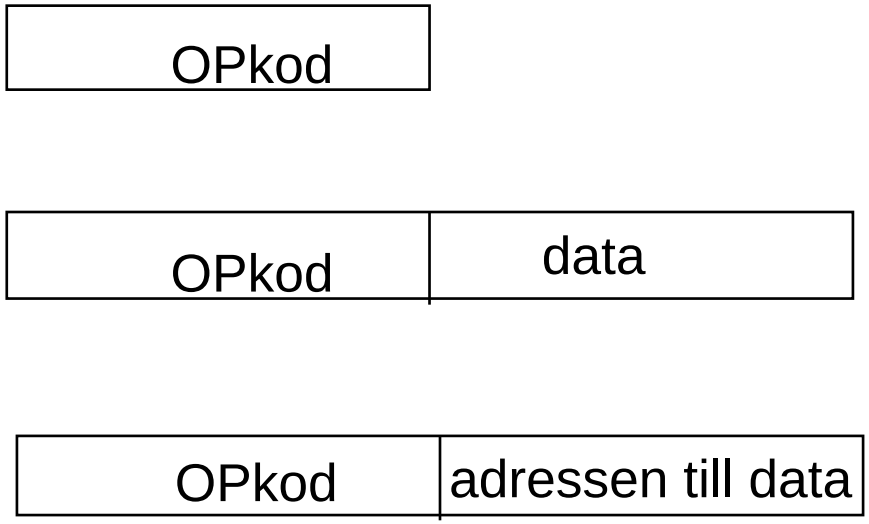
# Dagens mål: Du ska kunna….

- Beskriva likheter o olikheter mellan FLEX och CPU12
- **Använda Instruktionslistan för CPU12**
  - **Instruktionsgrupper**
  - **Adresseringsmoder**
- Skriva enkla program för CPU12
- Använda delar av utvecklingsmiljön Eterm för MC12

# Instruktionsrepertoar

- **Instruktioner för dataflyttning:**
  - Flytta data (Kopiera data)
- **Instruktioner för aritmetik:**
  - Utföra aritmetiska operationer (DAA, MUL, DIV)
- **Instruktioner för logiska operationer:**
  - AND, OR, XOR, komplementering (invertering).
- **Instruktioner för jämförelse och test**
  - Jämföra innehållet i ett register med ett minnesinnehåll
- **Hoppinstruktioner**
  - BEQ, LBEQ, DBNE, BRSET/CLR
- **Övriga instruktioner:**
  - Instruktioner för att styra processorns funktion och arbetssätt

# Instruktionsformat FLEX

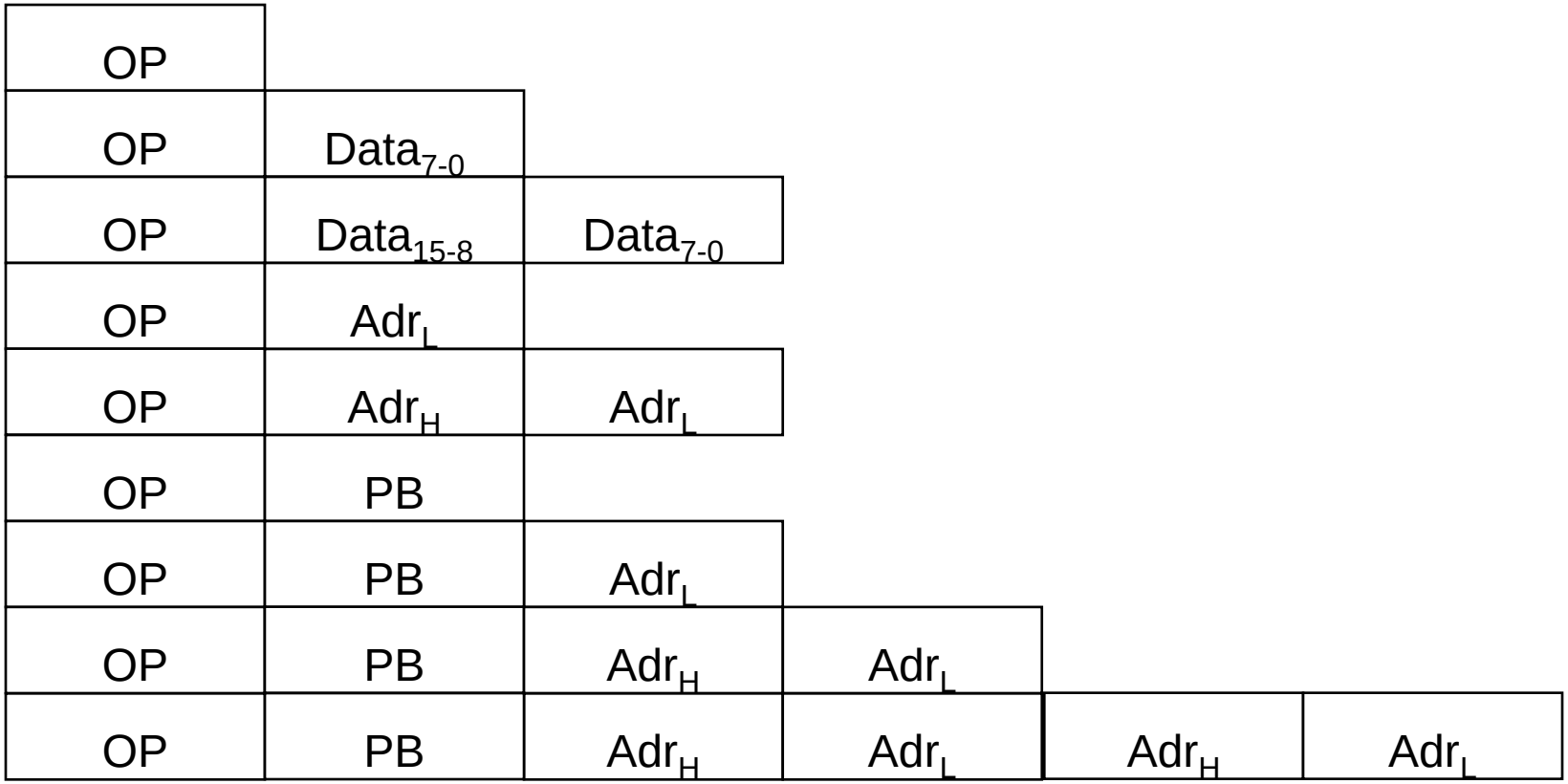



INCA

LDAA #$12

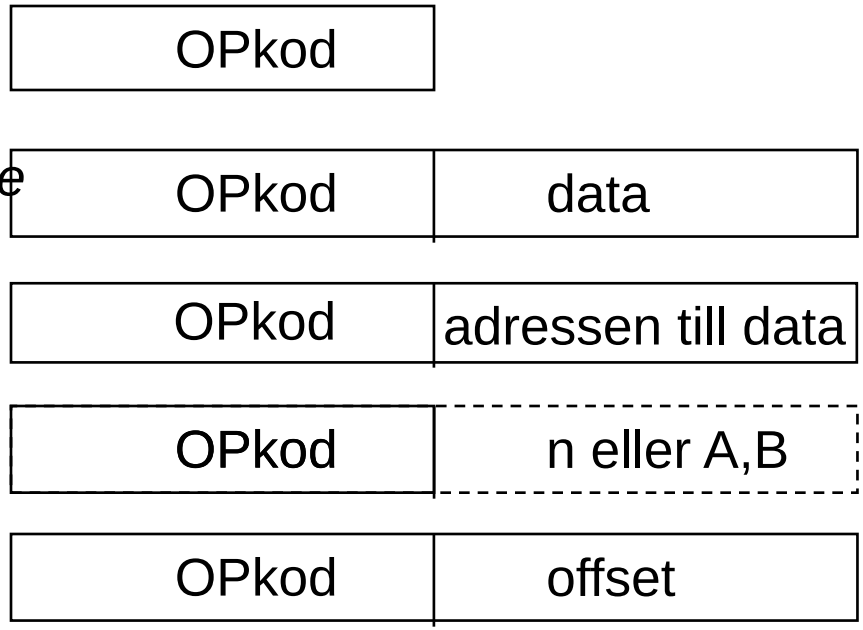
LDAA $12

# Instruktionsformat CPU12



# Adresseringsmoder FLEX

- *Inherent*
- *Immediate*
- *Absolut*
- *Via X*
- *Relativ*



# Adresseringsmoder CPU12

- Inherent INH
- Immediate IMM
- (Direct (Page) DIR)
- Extended EXT (= Absolut FLEX)
- Relative REL
- Indexed IDX/IDX1/IDX2

# Adresseringsmod: **Inherent** INH

| *Assembler* | *Maskinkod* | *RTN* |
|---|---|---|
| INCA | 42 | **A + 1 → A** |
| NEGB | 50 | **B' + 1 → B** |

***ENDAST OP-kod (Operanden är inbyggd i OP-koden)***

# Adresseringsmod: **Immediate** IMM

BRÄDGÅRD

| *Assembler* | | *Maskinkod* | | *RTN* |
|---|---|---|---|---|
| LDAB | #$3F | C6 | 3F | 3F→B |
| LDD | #$9AB2 | CC | 9A B2 | 9AB2 →D |
| LDY | #$1234 | CD | 12 34 | 1234 →Y |
| LDY | #$1 | CD | 00 01 | 0001 →Y |

***OP*** ***Dataoperand***

# Adresseringsmod: **Extended** EXT

INGEN BRÄDGÅRD

| *Assembler* | | *Maskinkod* | *RTN* |
|---|---|---|---|
| LDAB | $1234 | F6 12 34 | M(1234) →B |
| LDD | $9AB2 | FC 9A B2 | M(9AB2):M(9AB3) → D |
| LDY | $1234 | FD 12 34 | M(1234):M(1235) → Y |
| LDY | $1 | FD 00 01 | M(0001):M(0002) → Y |

***OP*** ***Adressoperand***

# Adresseringsmod: **Relative** REL

| *Assembler* | | *Maskinkod* | *RTN* |
|---|---|---|---|
| BRA | Loop | 20 ofs | PC+ofs → PC |
| BEQ | $9AB2 | 27 ofs | if Z=1: PC+ofs → PC |
| | | ***OP*** ***offset (1 byte) → [-128,127]*** | |
| LBRA | Stop | 18 20 ofsH ofsL | PC+ofsH:ofsL → PC |
| LBEQ | Snurra | 18 27 ofsH ofsL | if Z=1: PC+ofsH:ofsL → PC |
| | | ***OP*** ***offset (2 byte) → [-32768,32767]*** | |

**Kort BRANCH**

**Lång BRANCH**

# 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 $25_{16}$.

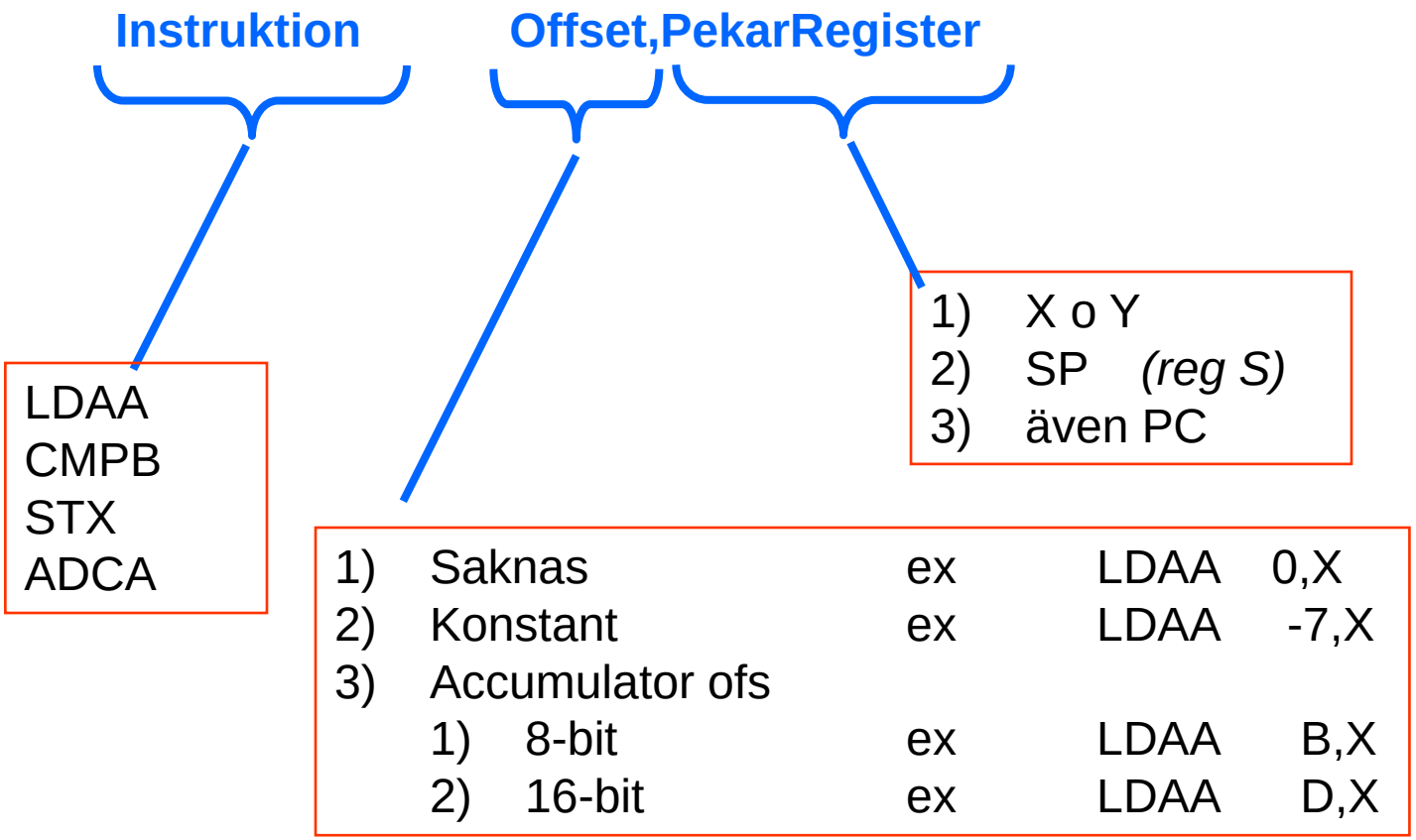
Programmet skall placeras med start på adress $50_{16}$

# Adressering via X (FLEX)
# (Indexerad adressering)

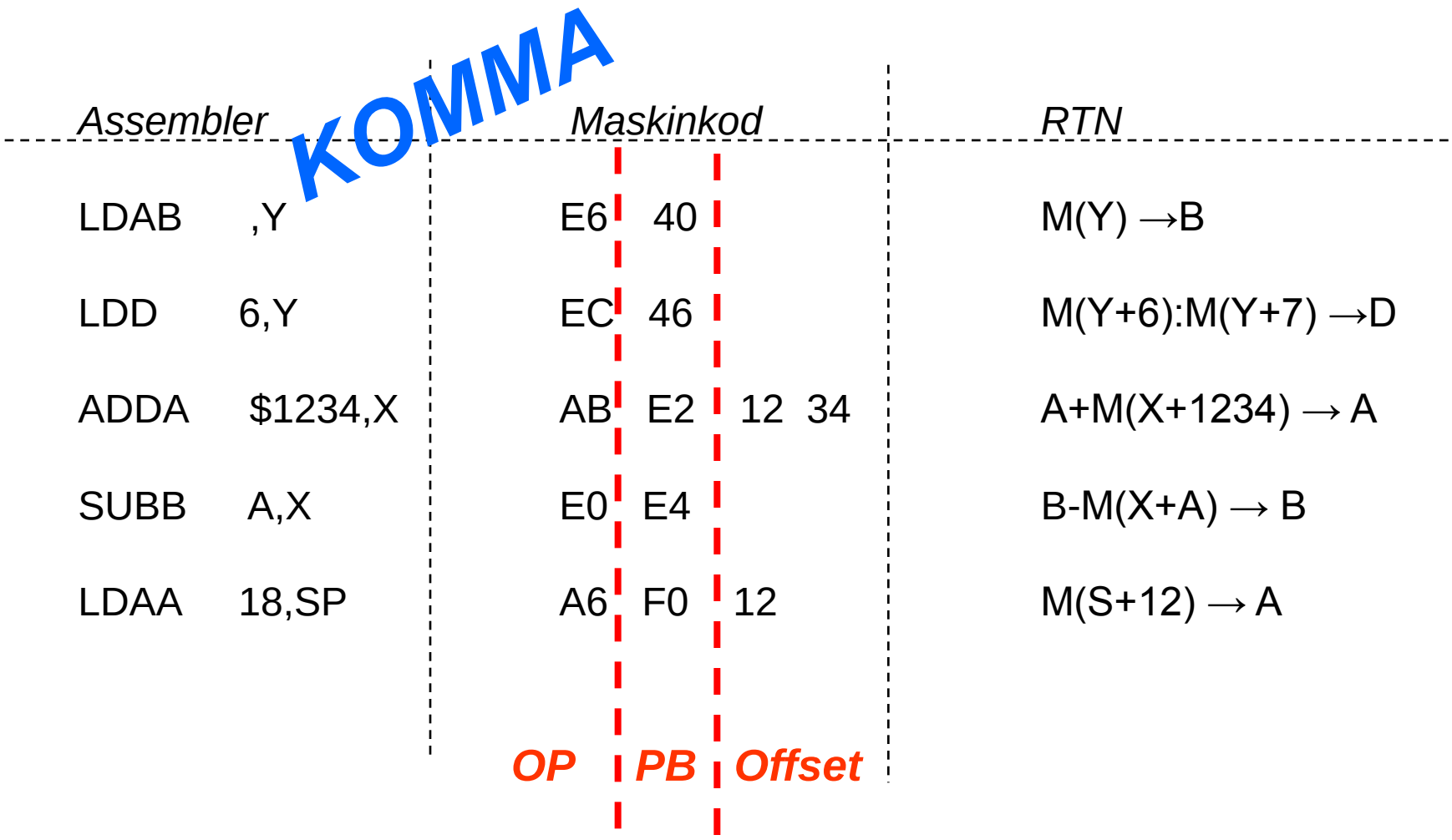
- STAA ,X
- STAA B,X
- STAA 1,X+

# Adresseringsmod: **Indexed** IDX, IDX1, IDX2

Ex) LDAA 3,X



# Adresseringsmod: **Indexed** IDX, IDX1, IDX2




***PB: PostByte – Vart hittas adressen***

# Adresseringsmod: **Indexed** pre/post inc/dec

| Assembler | | RTN: |
|---|---|---|
| LDAA | 2,X+ | 1) M(X) → A<br>2) X+2 → X |
| LDAA | 4,+X | 1) X+4 → X<br>2) M(X) → A |
| LDAA | 8,-X | 1) X-8 → X<br>2) M(X) → A |
| LDAA | 1,X- | 1) M(X) → A<br>2) X-1 → X |

# 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.

# Dagens mål: Du ska kunna….

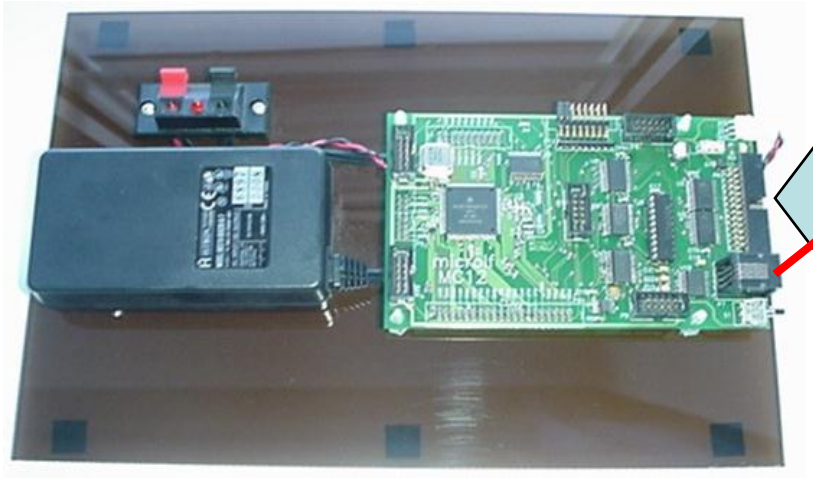
- Beskriva likheter o olikheter mellan FLEX och CPU12
- Använda Instruktionslistan för CPU12
  - Instruktionsgrupper
  - Adresseringsmoder
- **Skriva enkla program för CPU12**
- **Använda delar av utvecklingsmiljön Eterm för MC12**

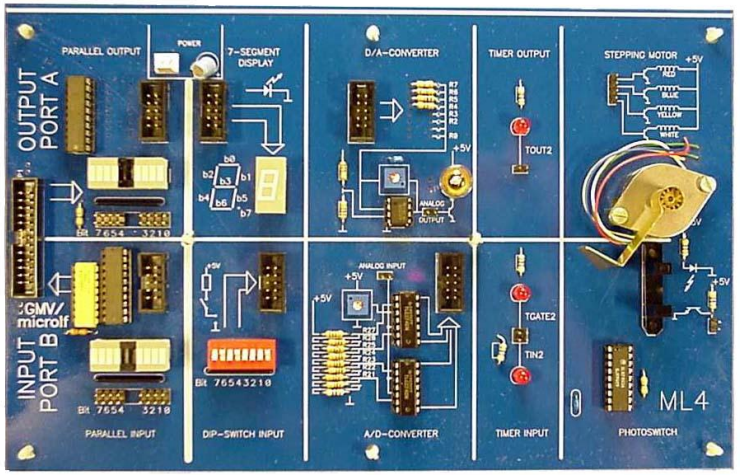
# MC12 Utvecklingsmiljö

COM-
port

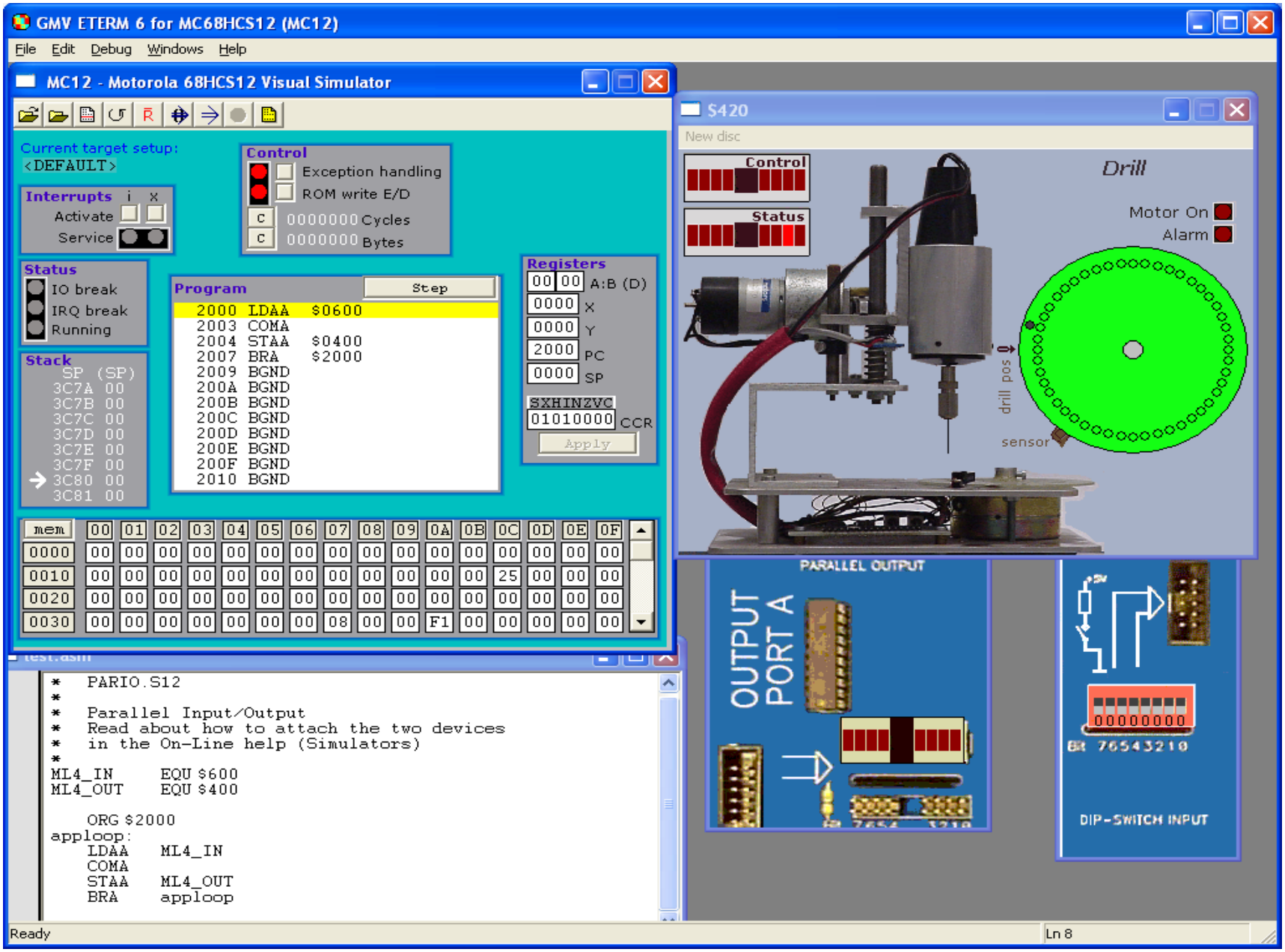
Eterm
*Emulera terminal*

Målsystem






MC12 o dbg12


GMV ETERM 6 for MC68HCS12 (MC12)
File Edit Debug Windows Help
MC12 - Motorola 68HCS12 Visual Simulator
Current target setup:
<DEFAULT>
Control
Exception handling
ROM write E/D
0000000 Cycles
0000000 Bytes
Interrupts i x
Activate
Service
Status
IO break
IRQ break
Running
Stack
SP (SP)
3C7A 00
3C7B 00
3C7C 00
3C7D 00
3C7E 00
3C7F 00
3C80 00
3C81 00
Program
Step
2000 LDAA $0600
2003 COMA
2004 STAA $0400
2007 BRA $2000
2009 BGND
200A BGND
200B BGND
200C BGND
200D BGND
200E BGND
200F BGND
2010 BGND
Registers
00 00 A:B (D)
0000 X
0000 Y
2000 PC
0000 SP
SXHINZVC
01010000 CCR
Apply
mem 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0010 00 00 00 00 00 00 00 00 00 00 00 00 25 00 00 00
0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030 00 00 00 00 00 00 00 08 00 00 F1 00 00 00 00 00
test.asm
* PARIO.S12
*
* Parallel Input/Output
* Read about how to attach the two devices
* in the On-Line help (Simulators)
*
ML4_IN EQU $600
ML4_OUT EQU $400
ORG $2000
apploop:
LDAA ML4_IN
COMA
STAA ML4_OUT
BRA apploop
Ready
Ln 8
S420
New disc
Control
Status
Drill
Motor On
Alarm
drill pos
sensor
PARALLEL OUTPUT
OUTPUT PORT A
00000000
Bit 76543210
DIP-SWITCH INPUT

Räkna antal ettställda bitar i Register A (DipSwitch)

Skriv antalet (register B) till HexDisplay