

TSEA83 : Datorkonstruktion

Fö3

Mikroprogrammering 2

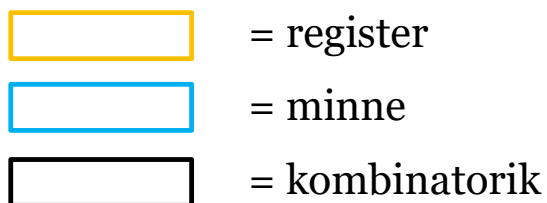
Fö3 : Agenda

- Lite repetition
- In/ut-matning
- Avbrott på OR-datorn
- Hoppinstruktion
- Lab1 : Mikroprogrammering

Lite repetition

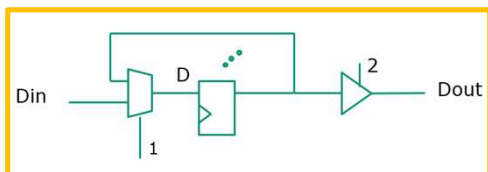
Mikromaskinen

”Olle Roos – datorn”

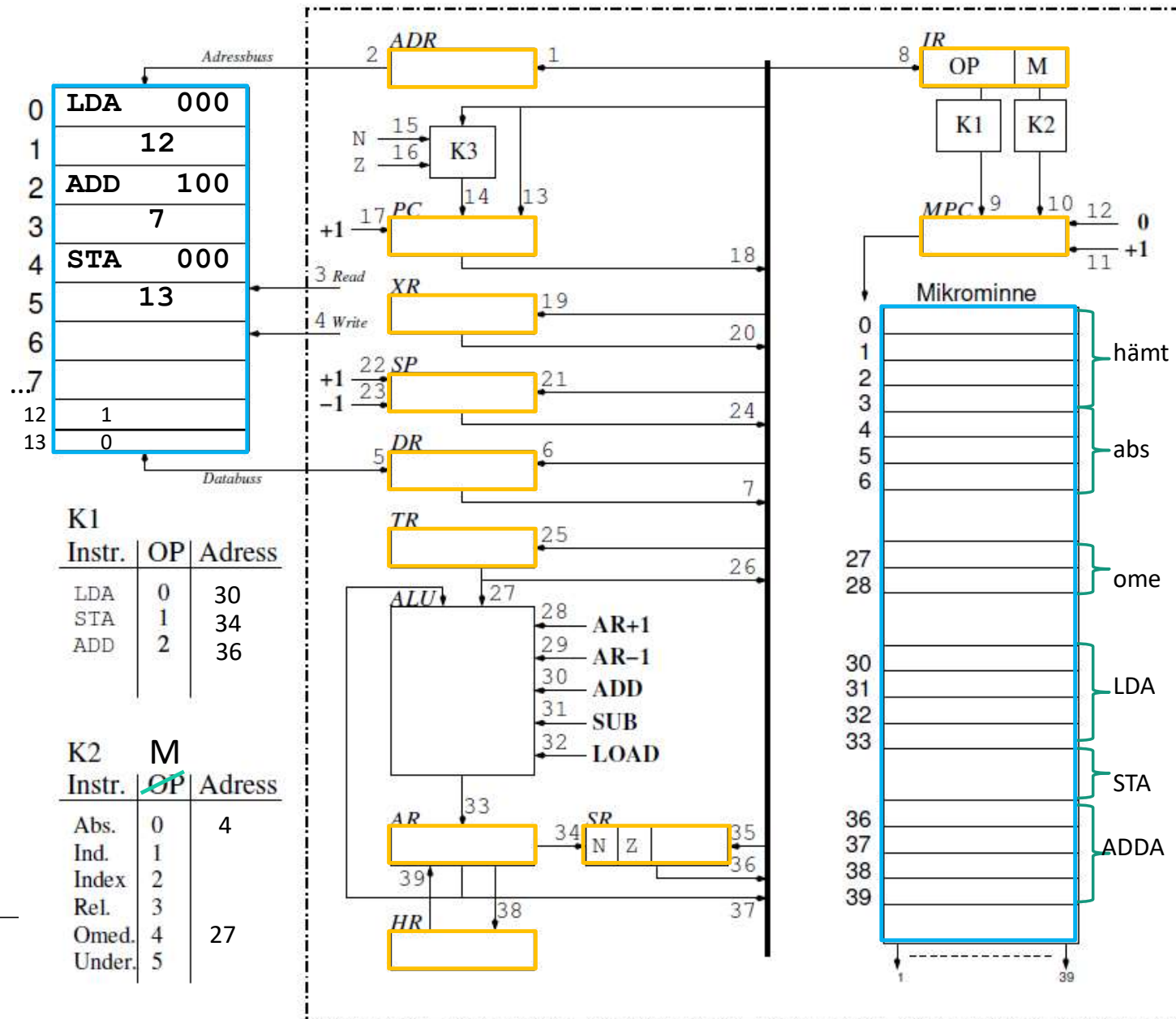


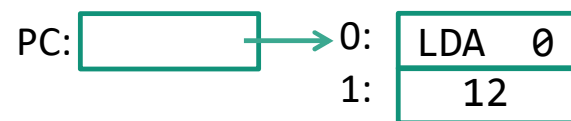
Tabellerna K1, K2 och K3 kan även implementeras som minnen.

Varje gul låda innehåller i princip:



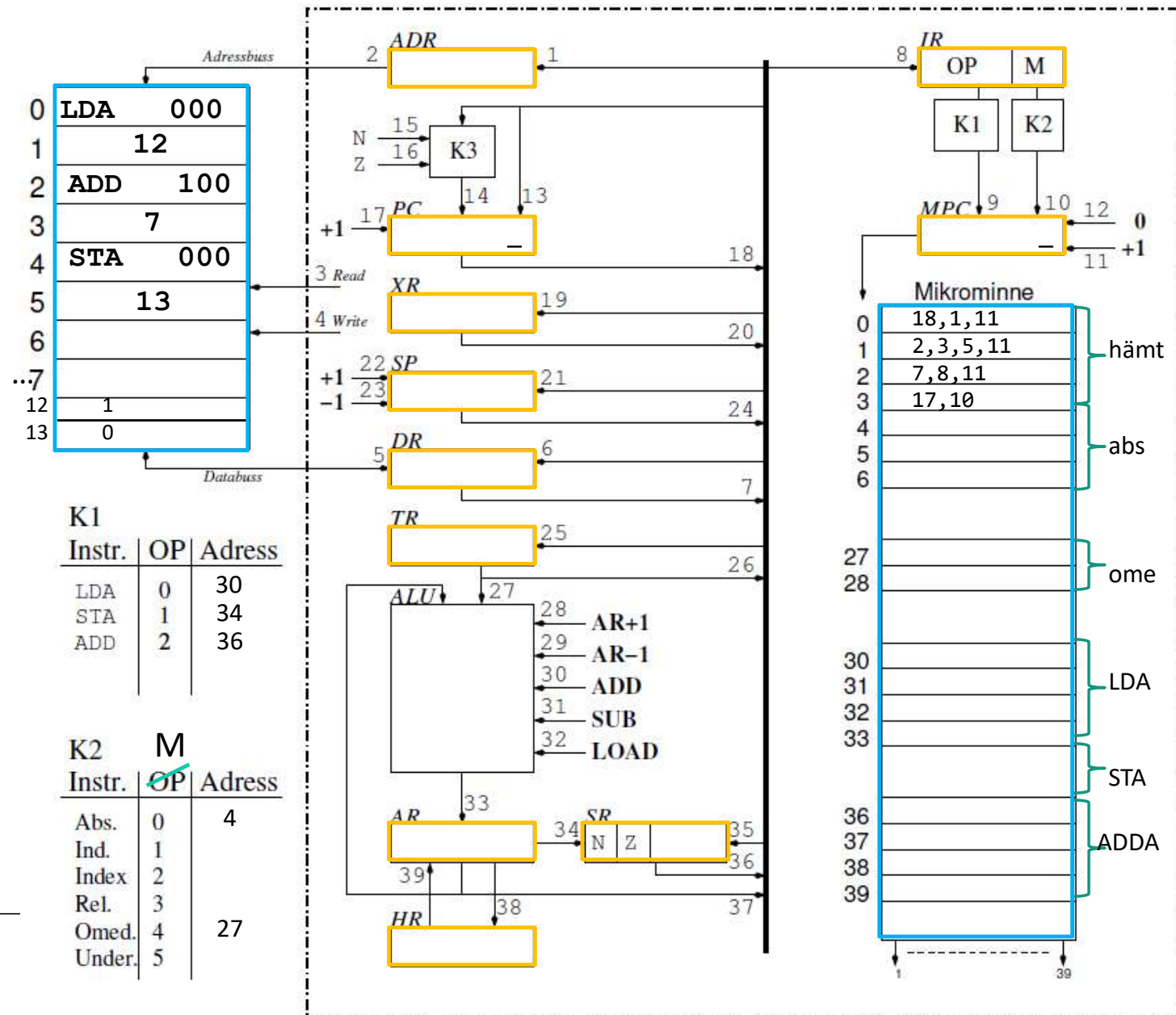
RESET 



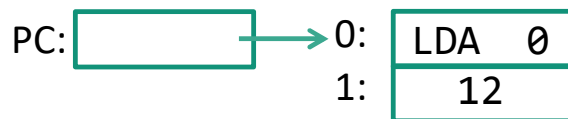


```
0: pc->adr,mpc++      18,1,11
1: adr->minne,data->dr,mpc++  2,3,5,11
2: dr->ir,mpc++       7,8,11
3: PC++,K2->mpc       17,10
```

Vid reset nollställs PC och MPC

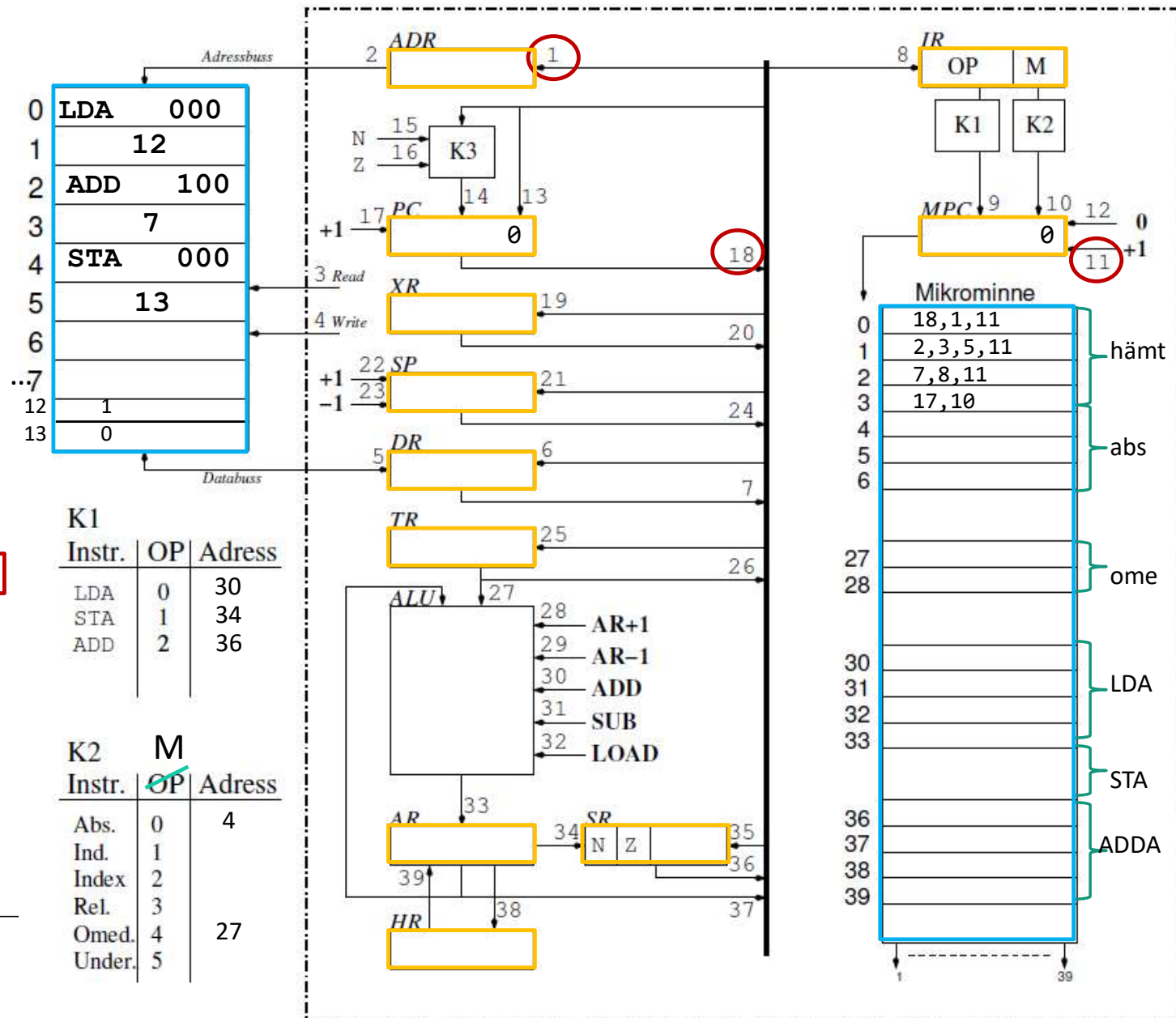


Steg 1 : Hämtfas M(PC) -> IR

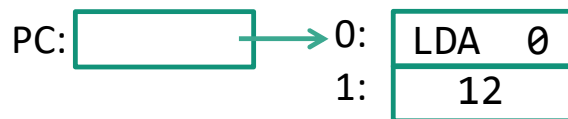


0: pc->adr, mpc++ 18,1,11
1: adr->minne, data->dr, mpc++ 2,3,5,11
2: dr->ir, mpc++ 7,8,11
3: PC++, K2->mpc 17,10

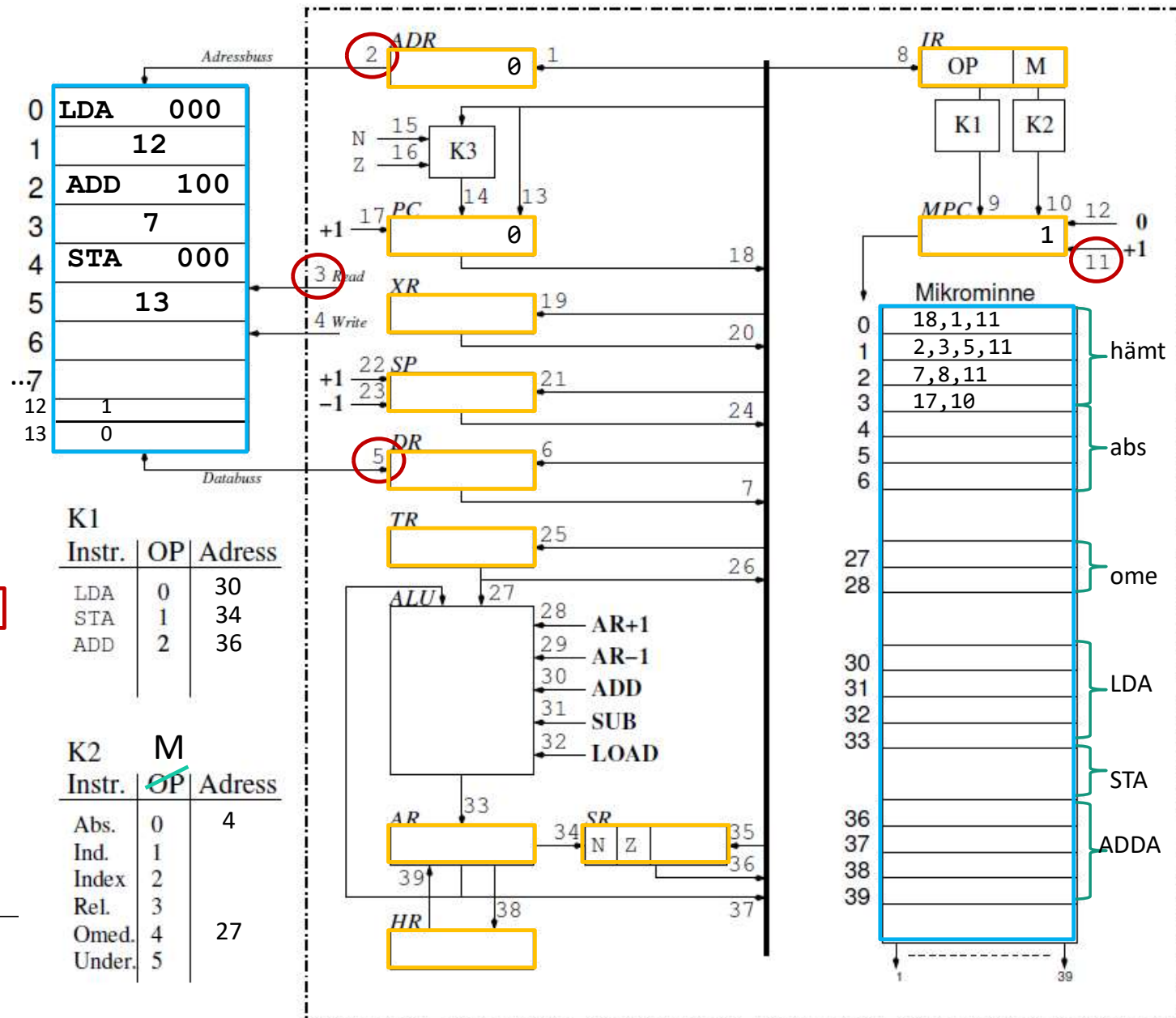
Rad 0 i Mikrominnet adresseras.
Effekterna av aktiverade styrsignaler
sker vid nästkommande klockflank.



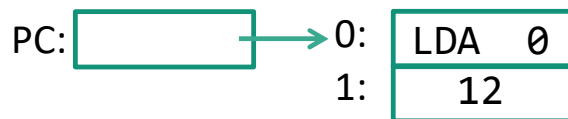
Steg 1 : Hämtfas M(PC) -> IR



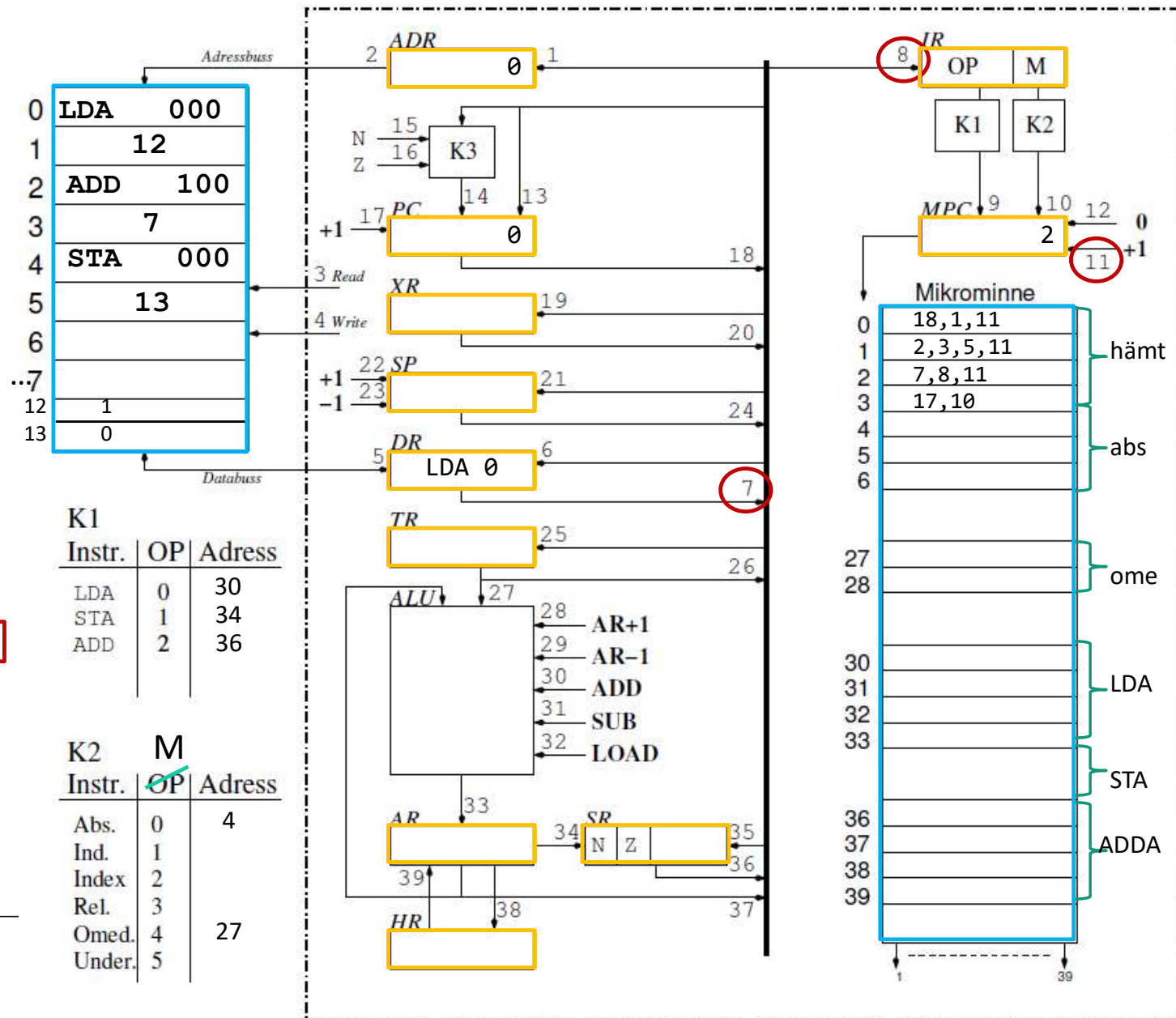
- 0: pc->adr, mpc++ 18,1,11
- 1: adr->minne, data->dr, mpc++ 2,3,5,11
- 2: dr->ir, mpc++ 7,8,11
- 3: PC++, K2->mpc 17,10



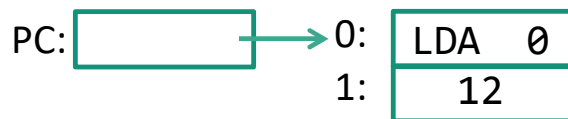
Steg 1 : Hämtfas M(PC) -> IR



0: pc->adr, mpc++ 18,1,11
1: adr->minne, data->dr, mpc++ 2,3,5,11
2: dr->ir, mpc++ 7,8,11
3: PC++, K2->mpc 17,10



Steg 1 : Hämtfas M(PC) -> IR



- 0: pc->adr, mpc++
- 1: adr->minne, data->dr, mpc++
- 2: dr->ir, mpc++
- 3: PC++, K2->mpc

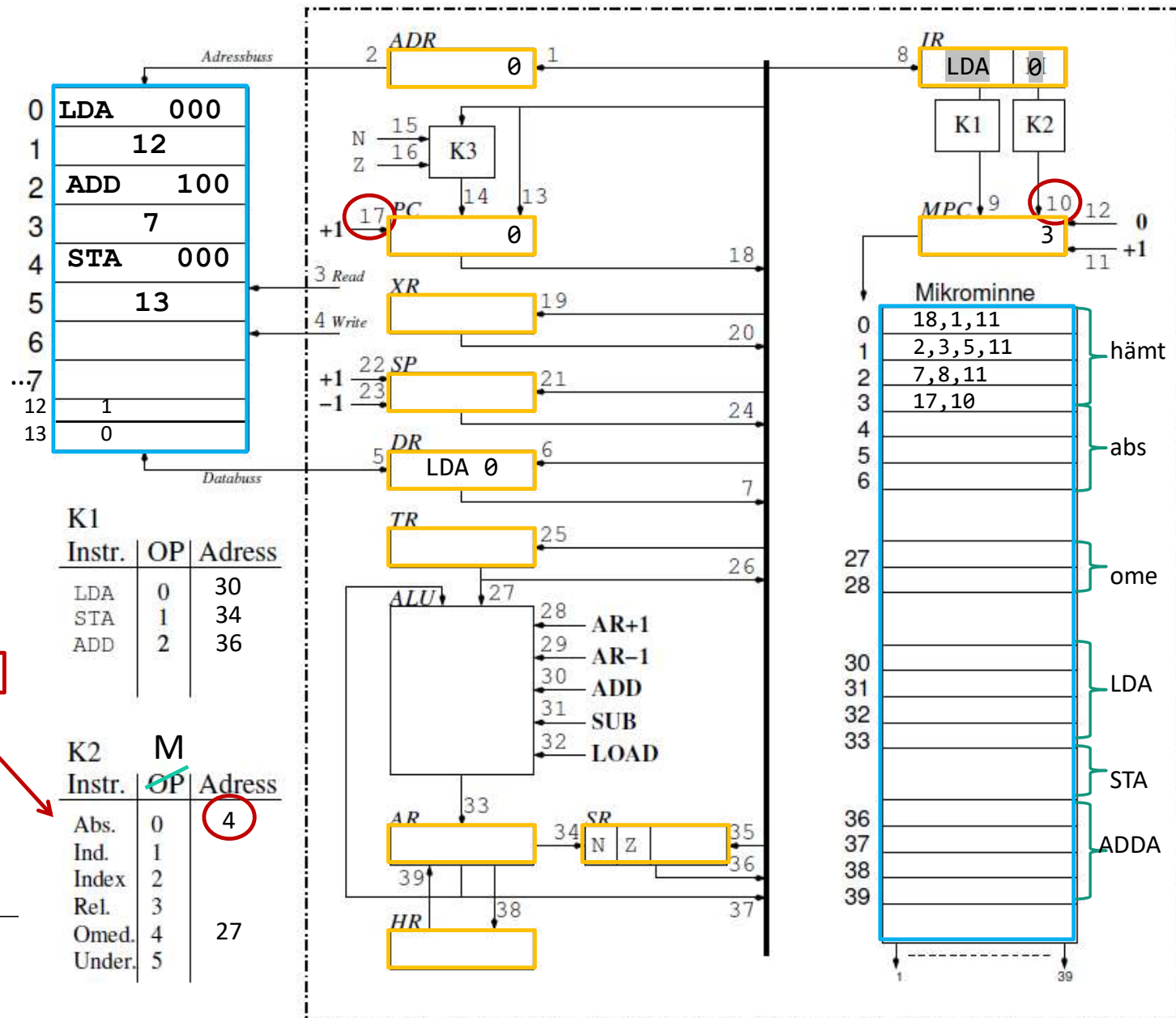
18, 1, 11

2, 3, 5, 11

7, 8, 11

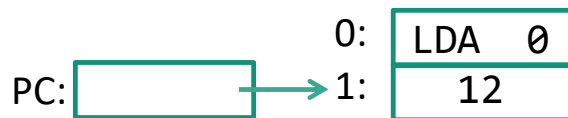
17, 10

Signal 10 aktiverar hopp till adresseringsmodsfas (Abs)

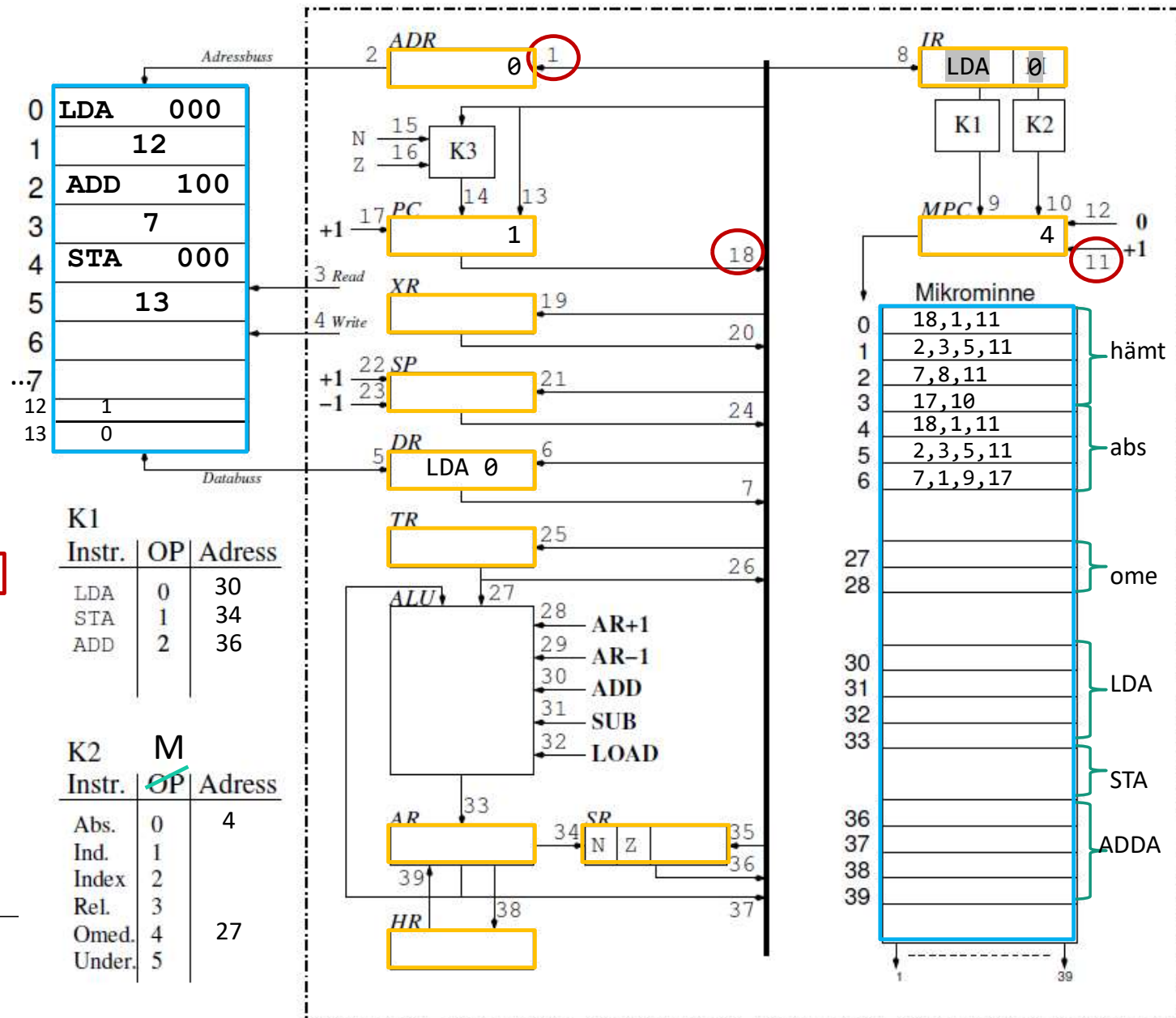


Steg 2 : A-modfas (Abs)

M(PC) -> ADR



- 4: pc->adr, mpc++ 18, 1, 11
 5: adr->minne, data->dr, mpc++ 2, 3, 5, 11
 6: dr->adr, K1->mpc, PC++ 7, 1, 9, 17

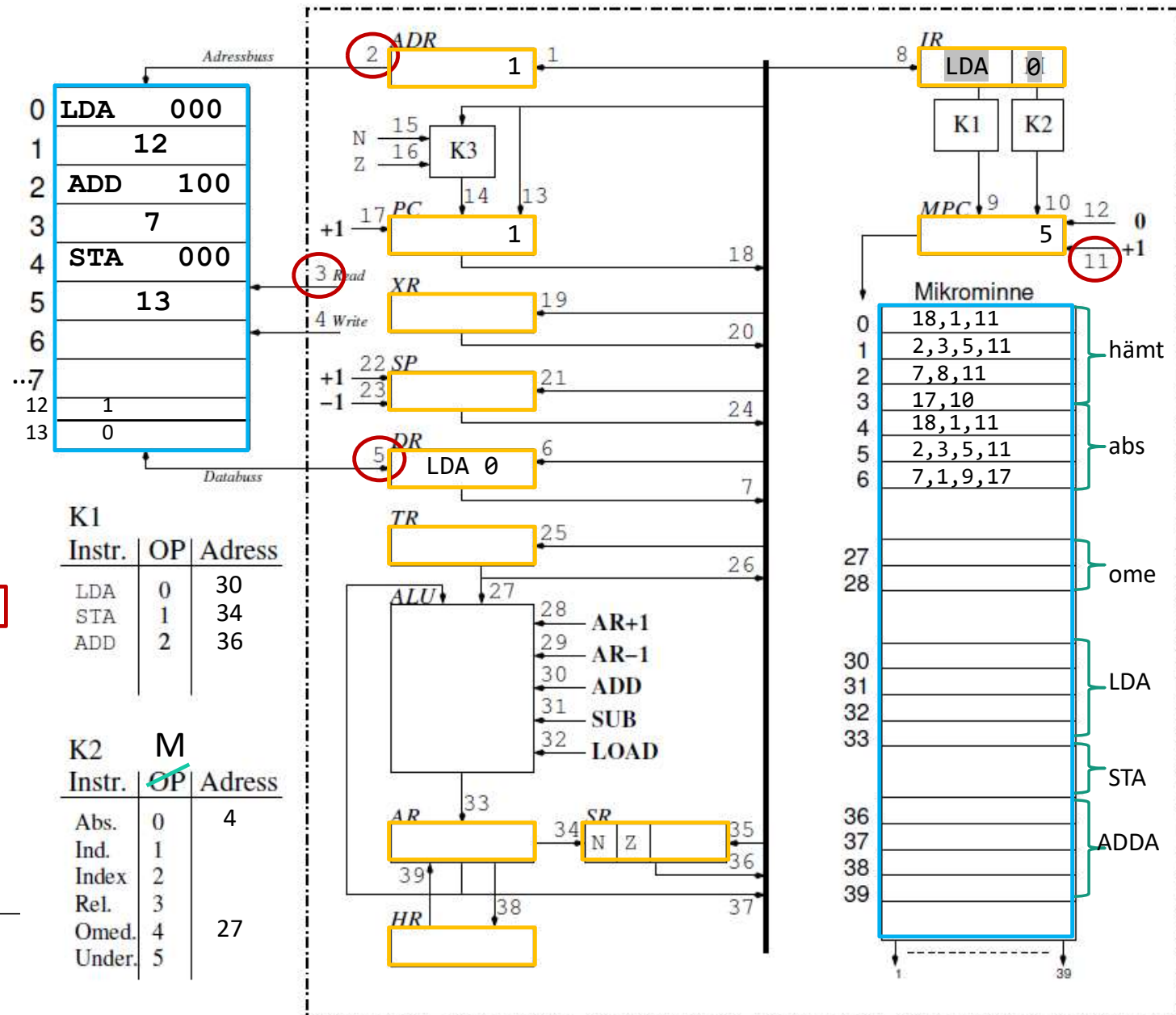




- ```
4: pc->adr,mpc++ 18,1,11
5: adr->minne,data->dr,mpc++ 2,3,5,11
6: dr->adr,K1->mpc,PC++ 7,1,9,17
```

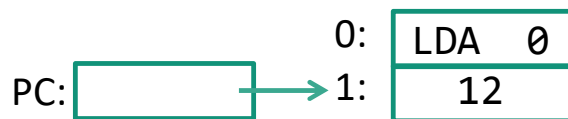
| K1     |    |         |
|--------|----|---------|
| Instr. | OP | Address |
| LDA    | 0  | 30      |
| STA    | 1  | 34      |
| ADD    | 2  | 36      |

| K2     | M  |         |
|--------|----|---------|
| Instr. | OP | Address |
| Abs.   | 0  | 4       |
| Ind.   | 1  |         |
| Index  | 2  |         |
| Rel.   | 3  |         |
| Omed.  | 4  | 27      |
| Under. | 5  |         |



## Steg 2 : A-modfas (Abs)

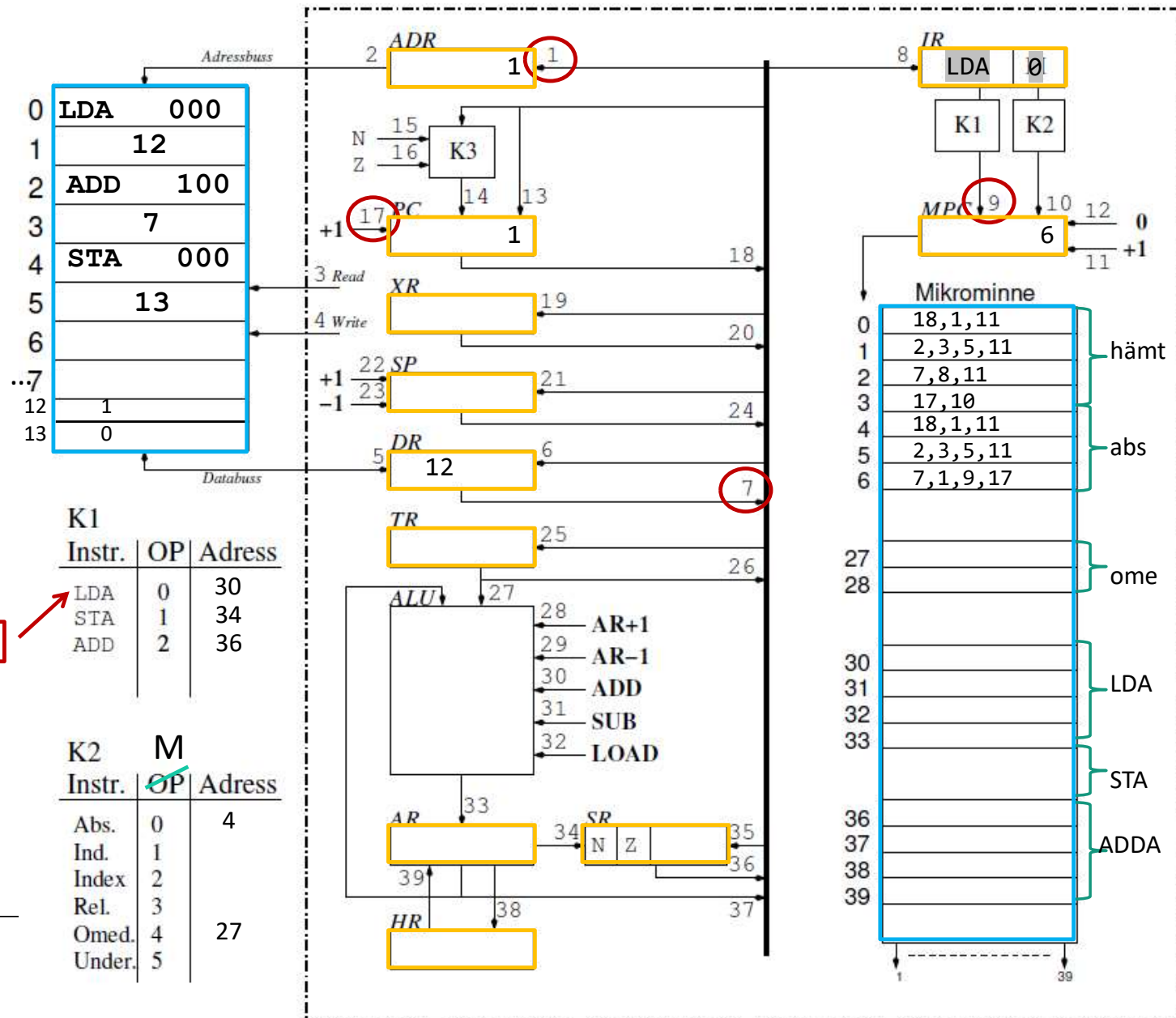
### M(PC) -> ADR



- 4: pc->adr, mpc++  
 5: adr->minne, data->dr, mpc++  
 6: dr->adr, K1->mpc, PC++

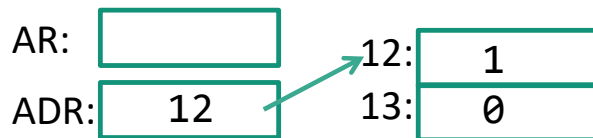
18, 1, 11  
 2, 3, 5, 11  
 7, 1, 9, 17

Signal 9 aktiverar hopp till  
 exekveringsfas (LDA)

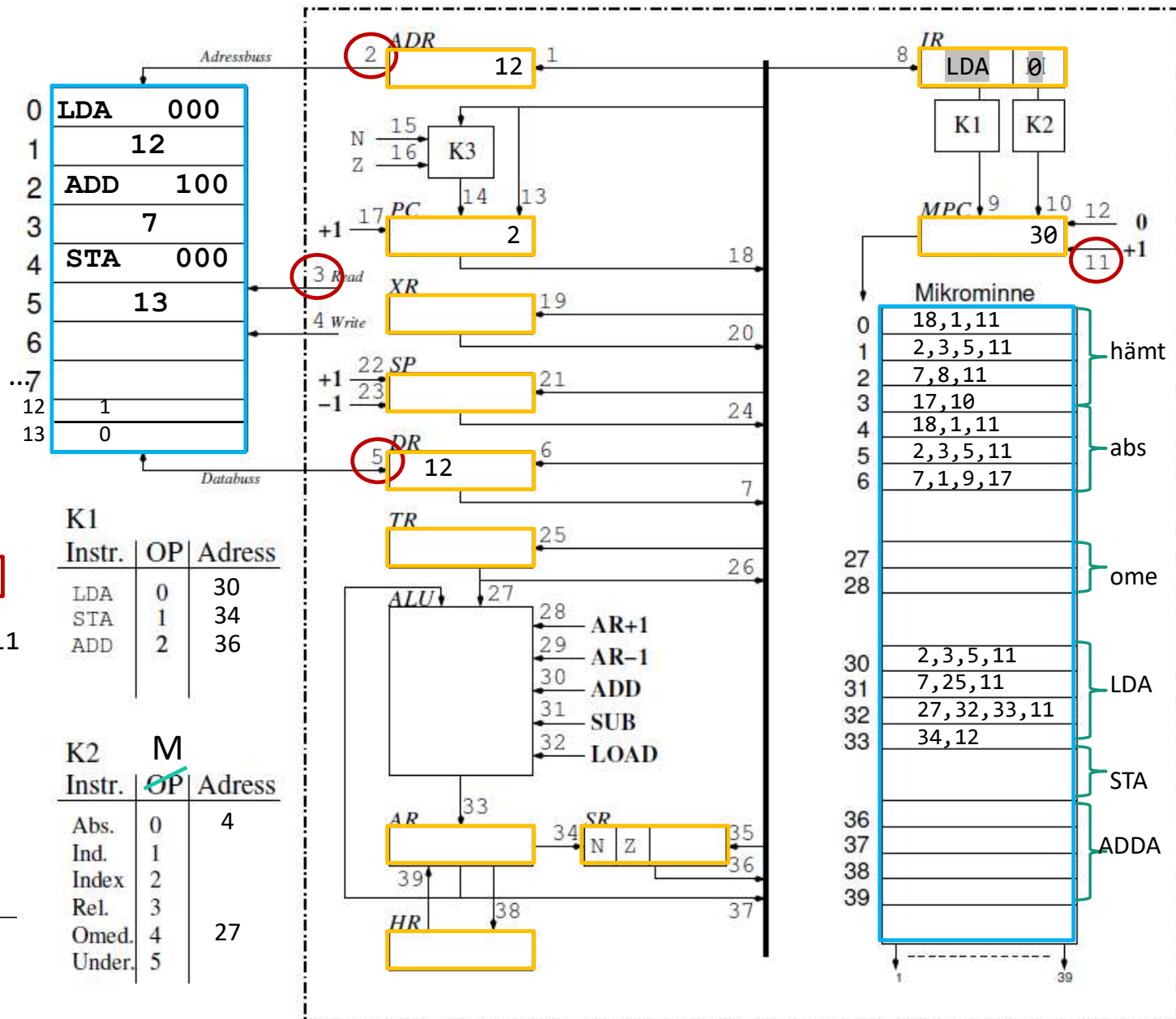


## Steg 3 : Exe-fas (LDA)

$$AR = M(ADR)$$



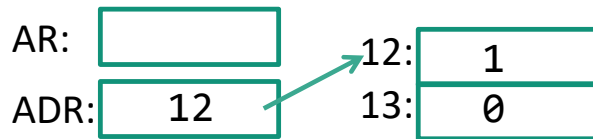
30: adr->minne, data->dr, mpc++ 2,3,5,11  
 31: dr->tr, mpc++ 7,25,11  
 32: tr->ar, mpc++ 27,32,33,11  
 33: status, 0->mpc 34,12



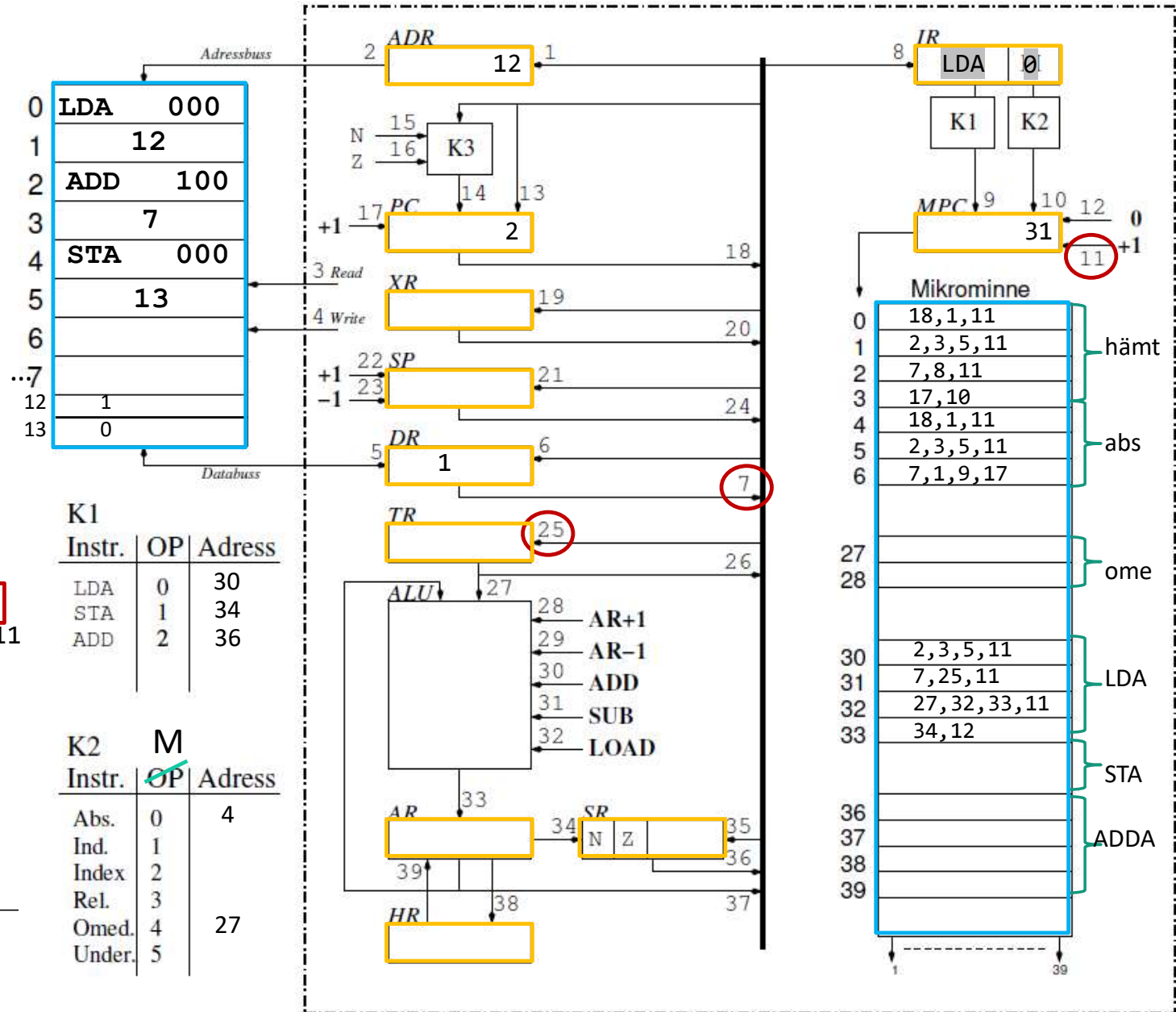


## Steg 3 : Exe-fas (LDA)

$$AR = M(ADR)$$

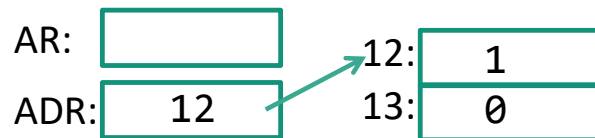


30: adr->minne,data->dr,mpc++ 2,3,5,11  
 31: dr->tr,mpc++ 7,25,11  
 32: tr->ar,mpc++ 27,32,33,11  
 33: status, 0->mpc 34,12

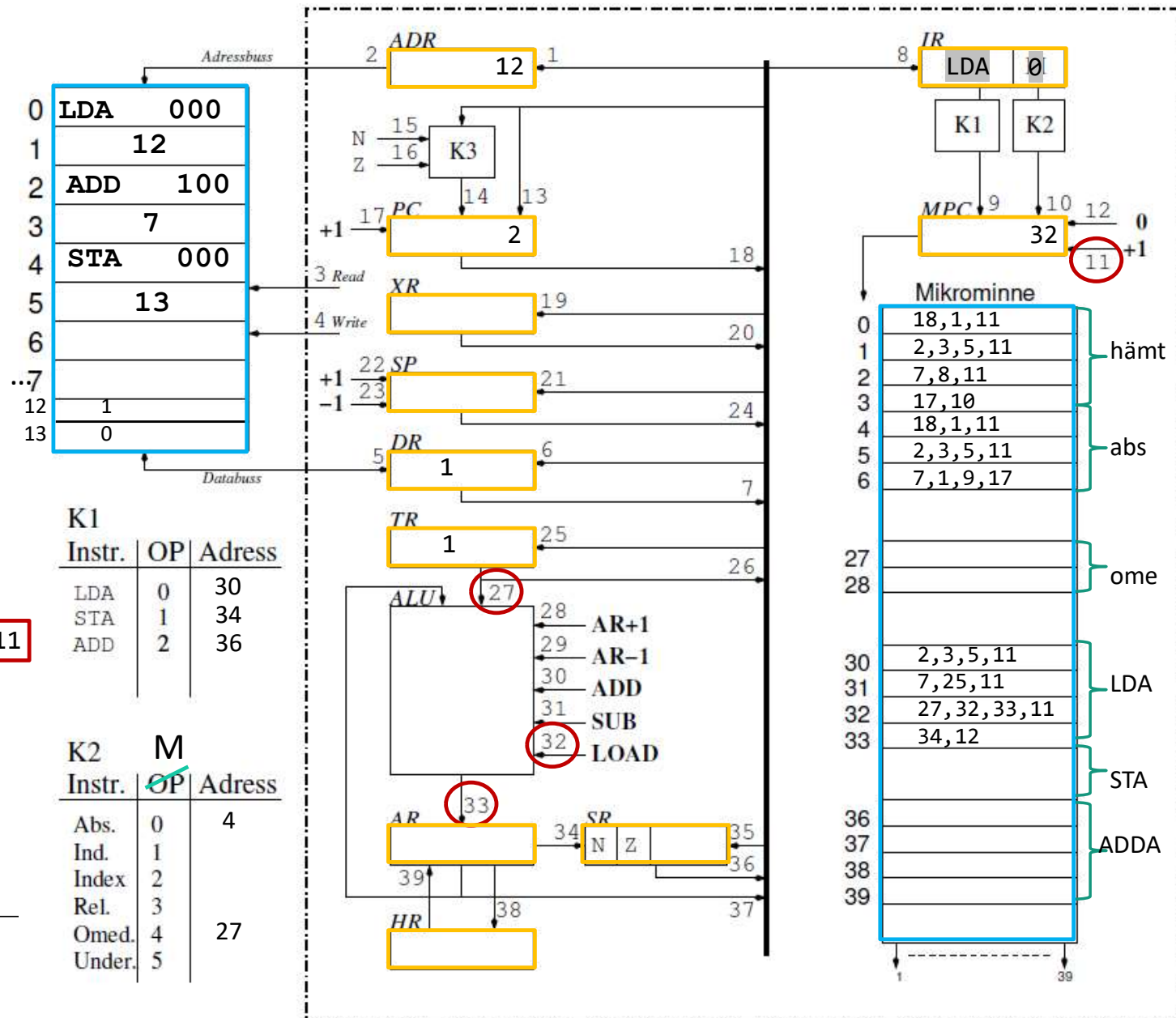


## Steg 3 : Exe-fas (LDA)

$$AR = M(ADR)$$

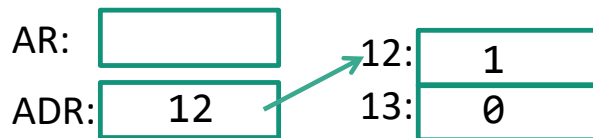


30: adr->minne,data->dr,mpc++ 2,3,5,11  
 31: dr->tr,mpc++ 7,25,11  
 32: tr->ar,mpc++ 27,32,33,11  
 33: status, 0->mpc 34,12



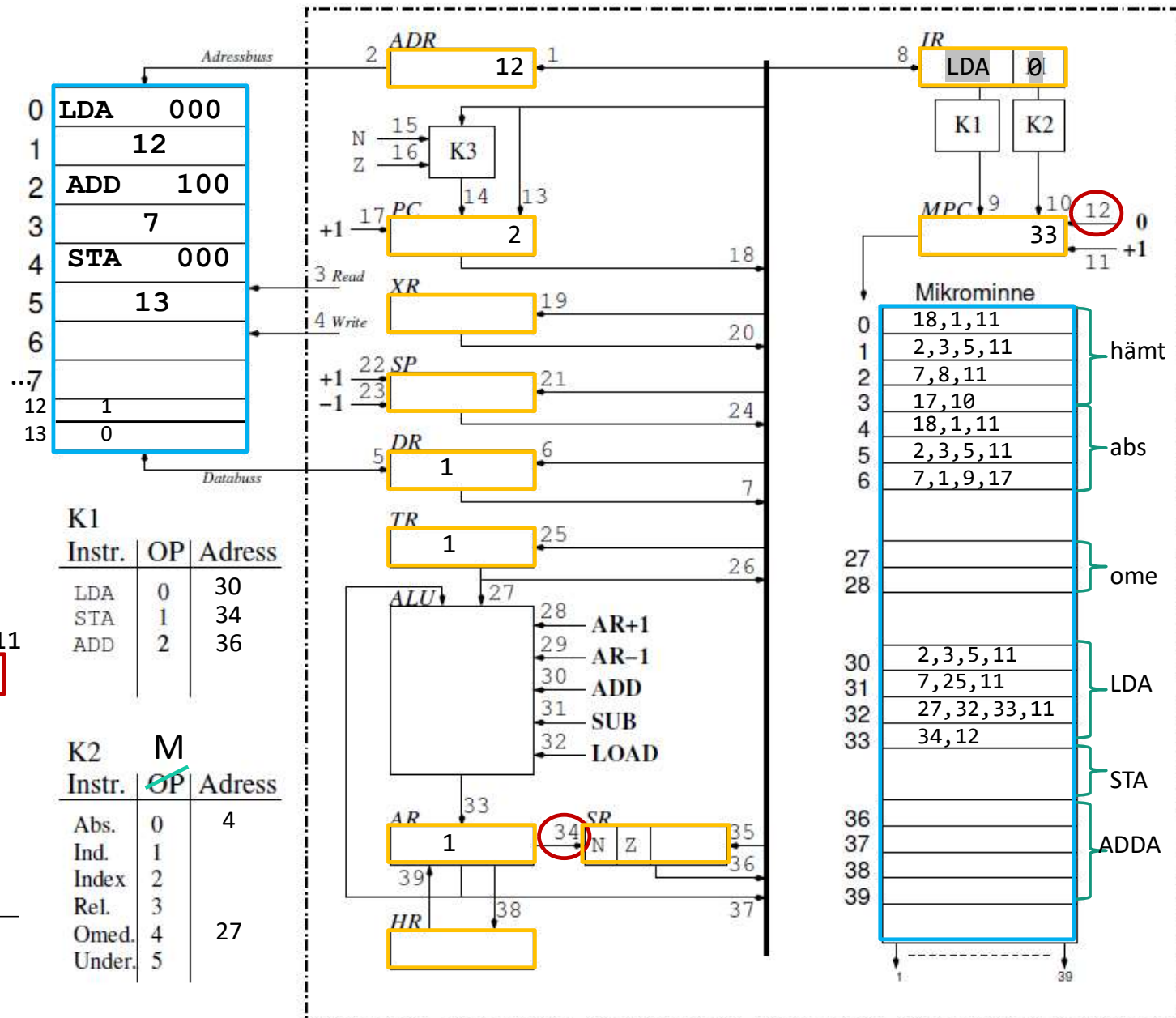
## Steg 3 : Exe-fas (LDA)

$$AR = M(ADR)$$



30: adr->minne,data->dr,mpc++ 2,3,5,11  
 31: dr->tr,mpc++ 7,25,11  
 32: tr->ar,mpc++ 27,32,33,11  
 33: status, 0->mpc 34,12

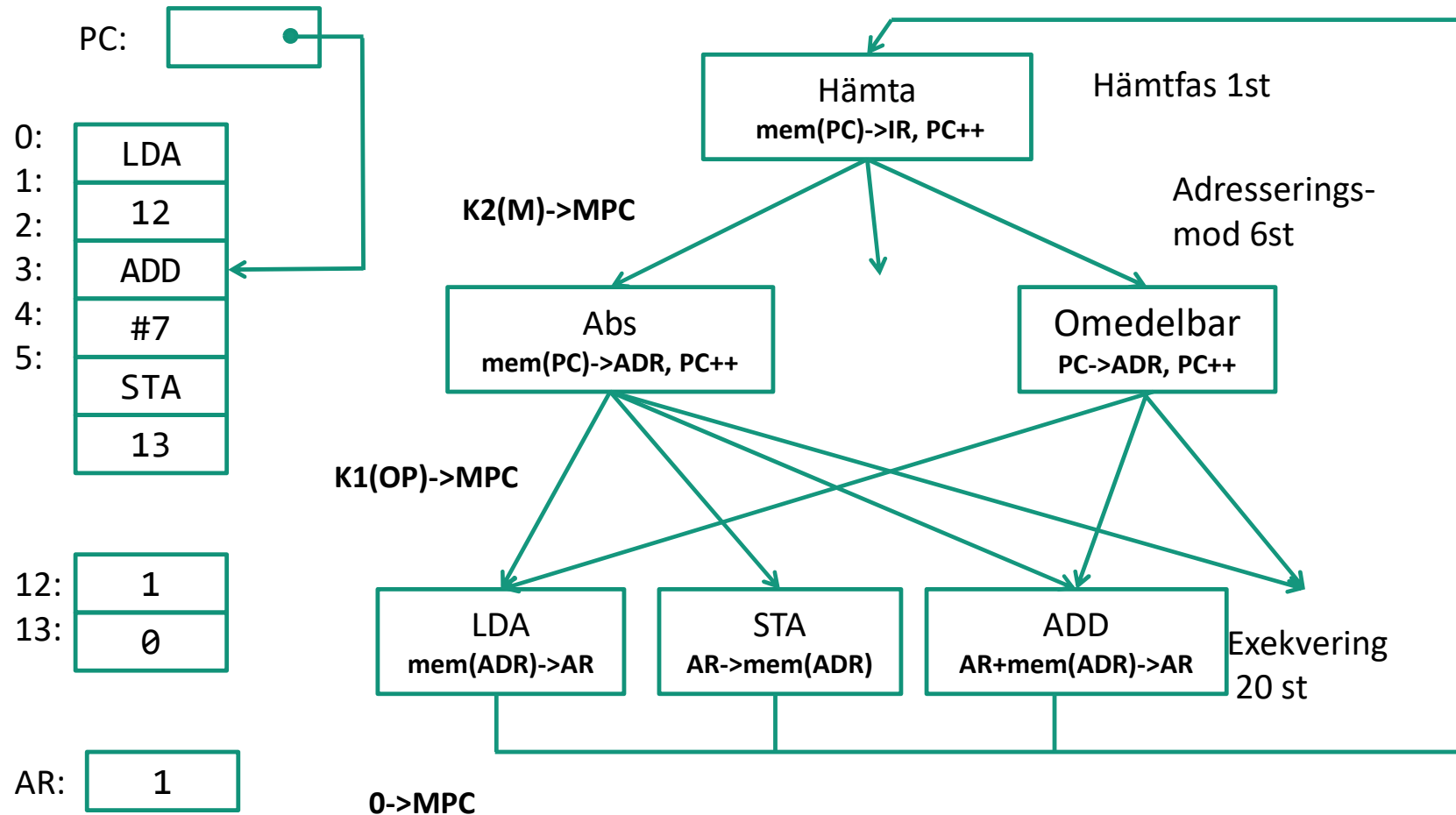
Signal 12 aktiverar  
 nollställning av MPC,  
 för nästa hämtfas





# asmprogram

# mikroprogram



# In/ut-matning

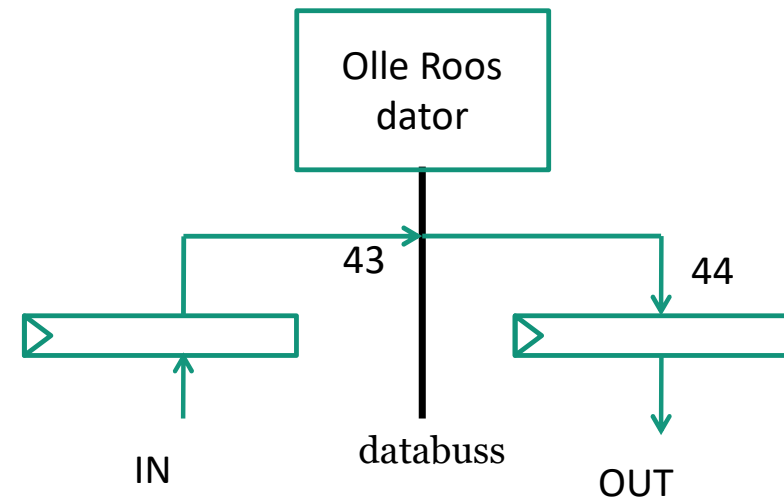
# In/ut-matning

| Instruktion | Betydelse   |
|-------------|-------------|
| IN          | $AR := IN$  |
| OUT         | $OUT := AR$ |

Adresseringsmod: underförstådd     $K2(5) = 29$

29: K1->mpc                      9

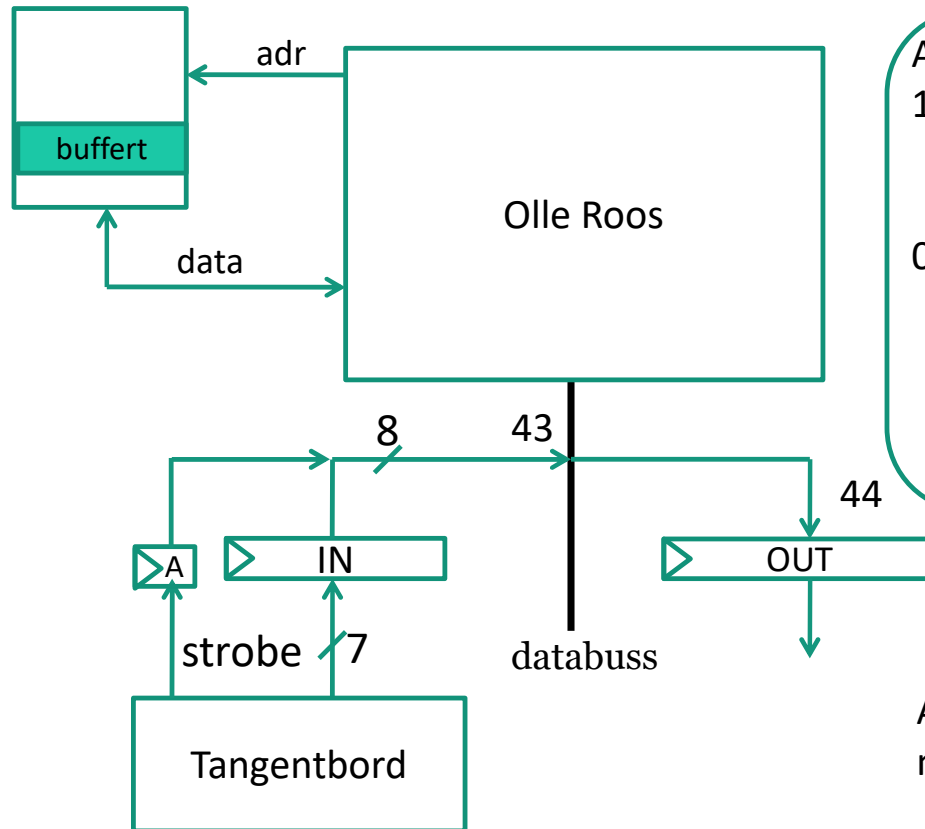
IN  
 50: in->tr, mpc++    43,25,11  
 51: tr->ar, mpc++    27,32,33,11  
 52: status, 0->mpc    34,12



OUT  
 53: ar->out, 0->mpc    37,44,12

# In/ut-matning

mem



## Problem:

Hur vet vi när det finns ett nytt värde i IN-registret?

A-vippan

1-ställning:

Stroben synkroniseras och enpulsas och 1-ställer A.

0-ställning:

styrsignal 43, dvs läsning av IN

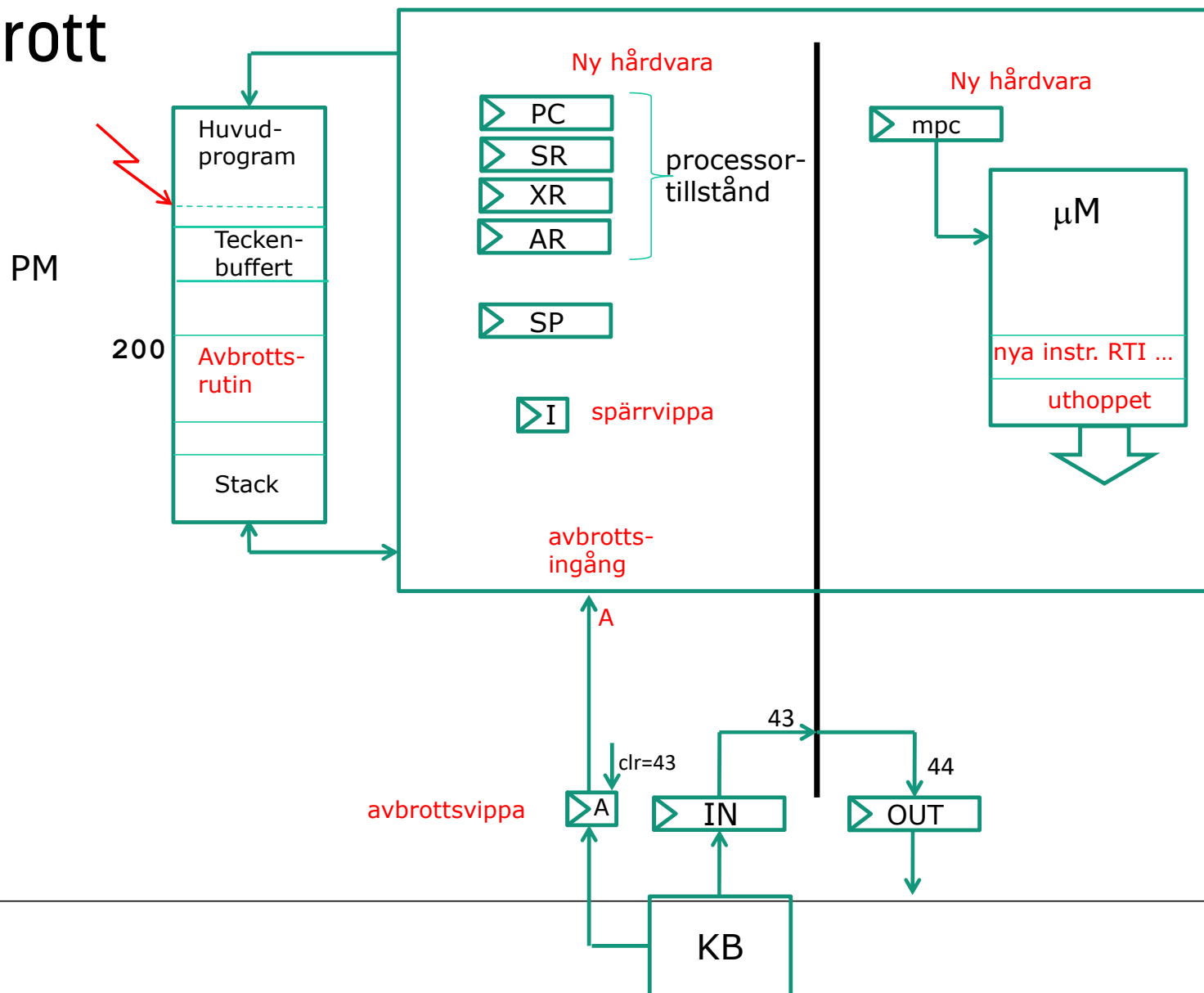
A-vippans läge kan kollas medelst polling.

# In/ut-matning : Olika metoder

- 1) Programmet väntar på att stroben ska bli hög,  
(testa om IN är negativ)  
läser tecknet och placerar i minnet.  
**Programstyrd I/O**, polling, busy waiting.
- 2) Programmet behöver inte alls vänta på stroben.  
När stroben går hög startar en avbrottsrutin, som  
läser in tecknet och placerar i minnet. **Avbrott**.
- 3) I/O kretsen skriver själv (genom att ta över lämpliga bussar)  
i minnet. **DMA = direkt minnesaccess**. Programmet behöver  
bara uppmärksammas när return har kommit in.  
Kan ske genom att koppla bort CPU:n från bussarna eller  
genom att utnyttja lediga minnescykler.

# Avbrott

# Avbrott



# Avbrott

1. Tryck på en tangent => 1->A
2. Gör klart pågående instruktion
3. Om I=0 så uthopp:
  1. Spara reg. på stacken  
PC, SR, XR, AR
  2. Förhindra fler avbrott: 1->I.
  3. Hoppa till avbrottsrutinen
4. I avbrottsrutinen:
  1. Läs in tecknet till minnesbuffer och 0->A
  2. Återhopp RTI (återställ reg., 0->I)

3 nya instruktioner:

EI: 0->I (Enable Interrupt)

DI: 1->I (Disable Interrupt)

RTI: återhopp

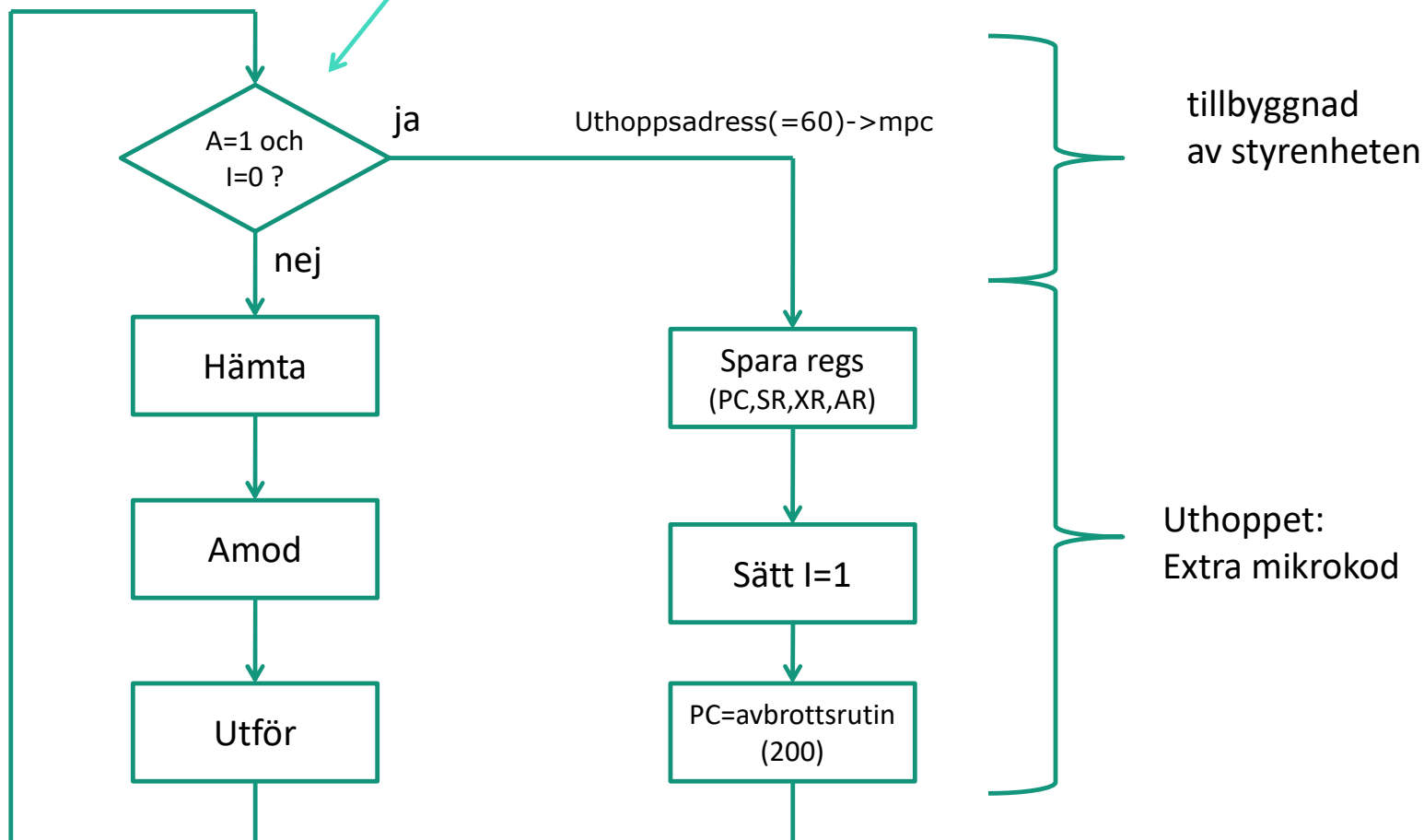
Mikrokod för uthopp

Förbättrad mpc

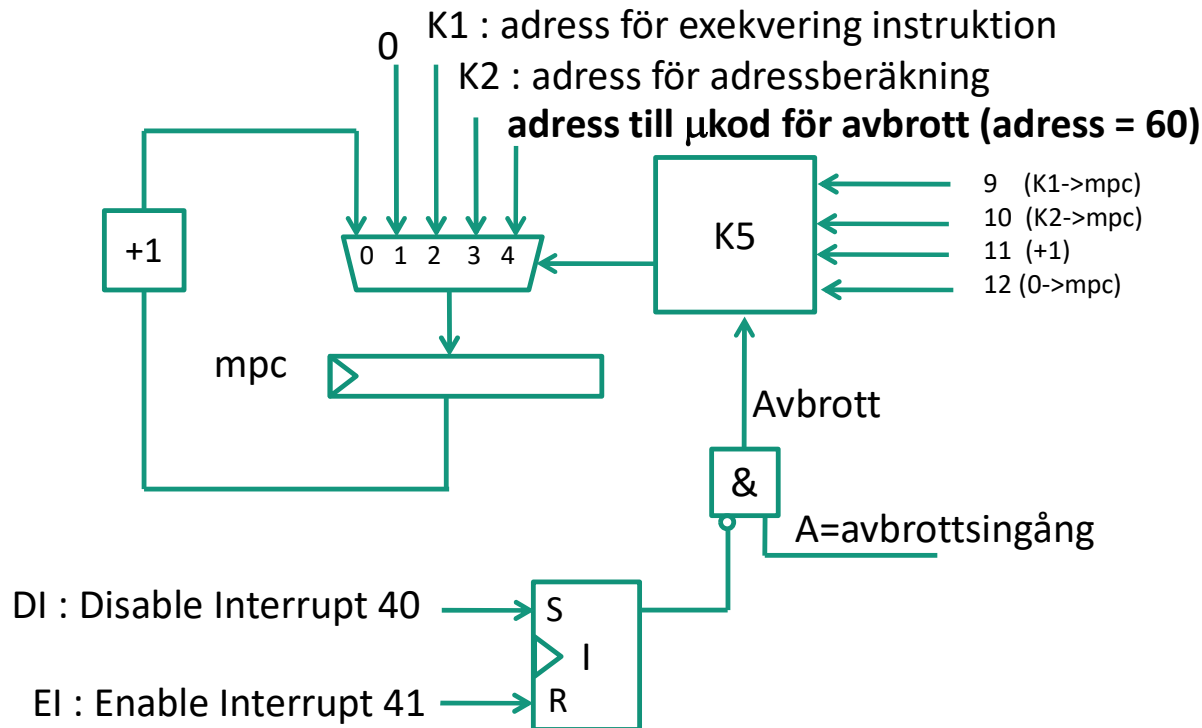


# Avbrott

Styrenheten kollar om det är avbrott mellan två instruktioner



# Datorkonstruktion Avbrott



| K5: | 9 | 10 | 11 | 12 | Avbr | mux-ingång |
|-----|---|----|----|----|------|------------|
|     | 1 | 0  | 0  | 0  | -    | 2          |
|     | 0 | 1  | 0  | 0  | -    | 3          |
|     | 0 | 0  | 1  | 0  | -    | 0          |
|     | 0 | 0  | 0  | 1  | 0    | 1          |
|     | 0 | 0  | 0  | 1  | 1    | 4          |

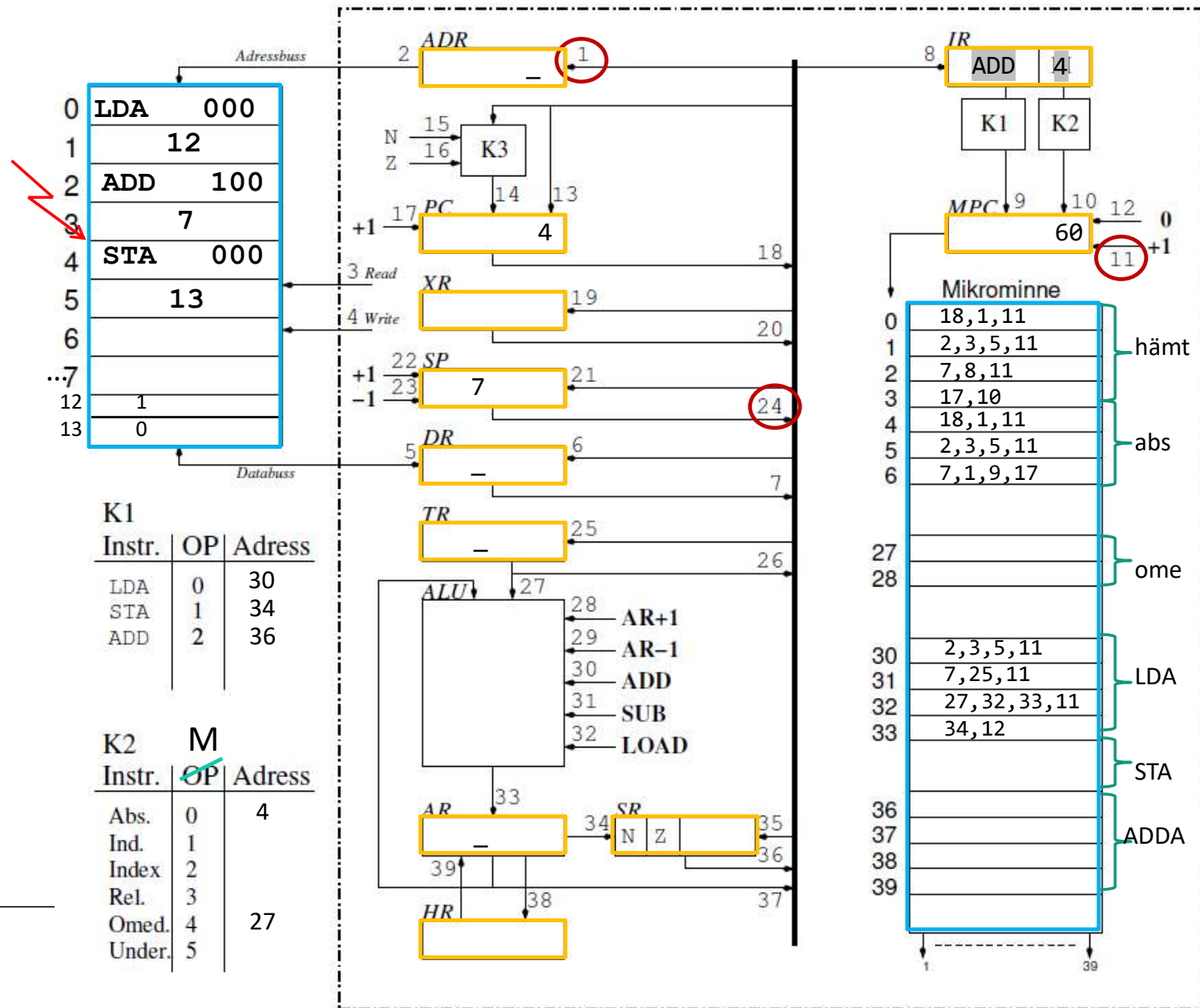
Adress (200) till avbrottsrutin



# Avbrott

## Spara PC,XR,AR,SR på stacken

60: sp->adr, mpc++ 24,1,11  
 61: pc->dr, mpc++ 18,6,11  
 62: skriv, sp--, mpc++ 2,4,5,23,11  
 63: sp->adr, mpc++ 24,1,11  
 64: xr->dr, mpc++ 20,6,11  
 65: skriv, sp--, mpc++ 2,4,5,23,11  
 66: sp->adr, mpc++ 24,1,11  
 67: ar->dr, mpc++ 37,6,11  
 68: skriv, sp--, mpc++ 2,4,5,23,11  
 69: sp->adr, mpc++ 24,1,11  
 70: sr->dr, mpc++ 36,6,11  
 71: skriv, sp--, mpc++ 2,4,5,23,11  
 72: 1->I, mpc++ 40,11  
 73: 200->PC, mpc=0 42,12



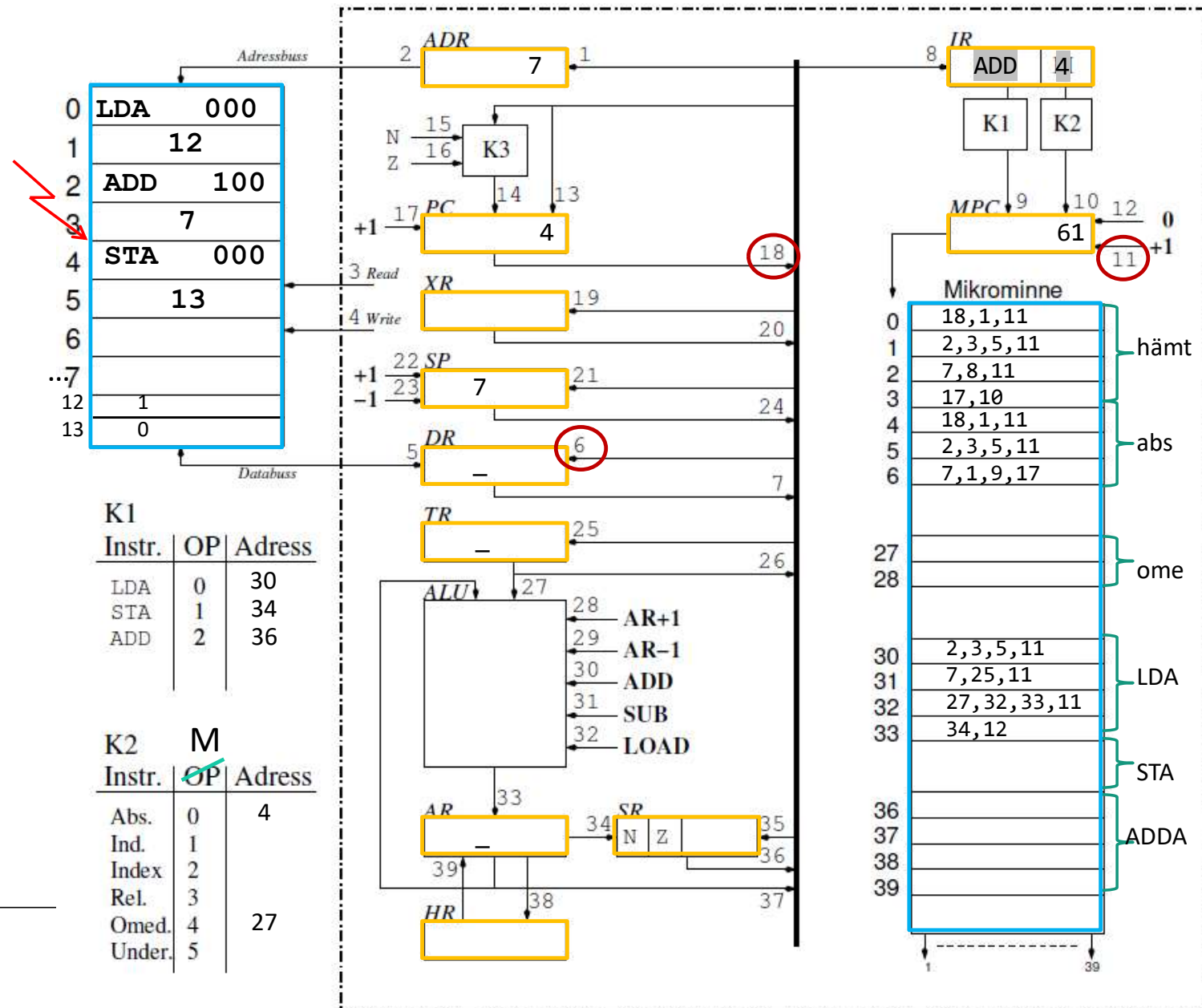
# Avbrott

## Spara PC,XR,AR,SR på stacken

```

60: sp->adr, mpc++ 24,1,11
61: pc->dr, mpc++ 18,6,11
62: skriv, sp--, mpc++ 2,4,5,23,11
...
63: sp->adr, mpc++ 24,1,11
64: xr->dr, mpc++ 20,6,11
65: skriv, sp--, mpc++ 2,4,5,23,11
...
66: sp->adr, mpc++ 24,1,11
67: ar->dr, mpc++ 37,6,11
68: skriv, sp--, mpc++ 2,4,5,23,11
...
69: sp->adr, mpc++ 24,1,11
70: sr->dr, mpc++ 36,6,11
71: skriv, sp--, mpc++ 2,4,5,23,11
...
72: 1->I, mpc++ 40,11
73: 200->PC, mpc=0 42,12

```



# Avbrott

## Spara PC,XR,AR,SR på stacken

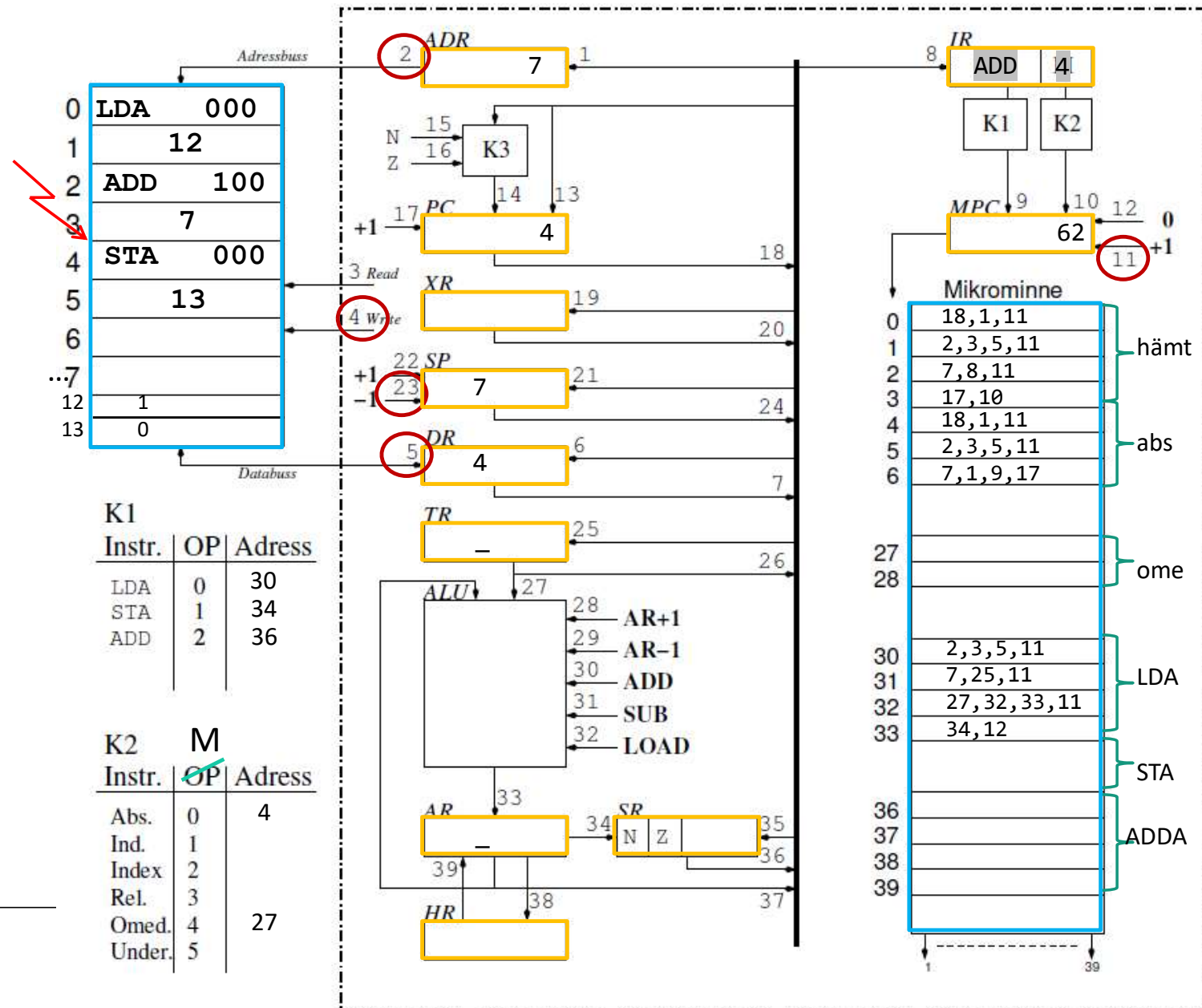
60: sp->adr, mpc++      24,1,11  
 61: pc->dr, mpc++      18,6,11  
 62: skriv, sp--, mpc++    **2,4,5,23,11**

63: sp->adr, mpc++      24,1,11  
 64: xr->dr, mpc++      20,6,11  
 65: skriv, sp--, mpc++    2,4,5,23,11

66: sp->adr, mpc++      24,1,11  
 67: ar->dr, mpc++      37,6,11  
 68: skriv, sp--, mpc++    2,4,5,23,11

69: sp->adr, mpc++      24,1,11  
 70: sr->dr, mpc++      36,6,11  
 71: skriv, sp--, mpc++    2,4,5,23,11

72: 1->I, mpc++          40,11  
 73: 200->PC, mpc=0      42,12



# Avbrott

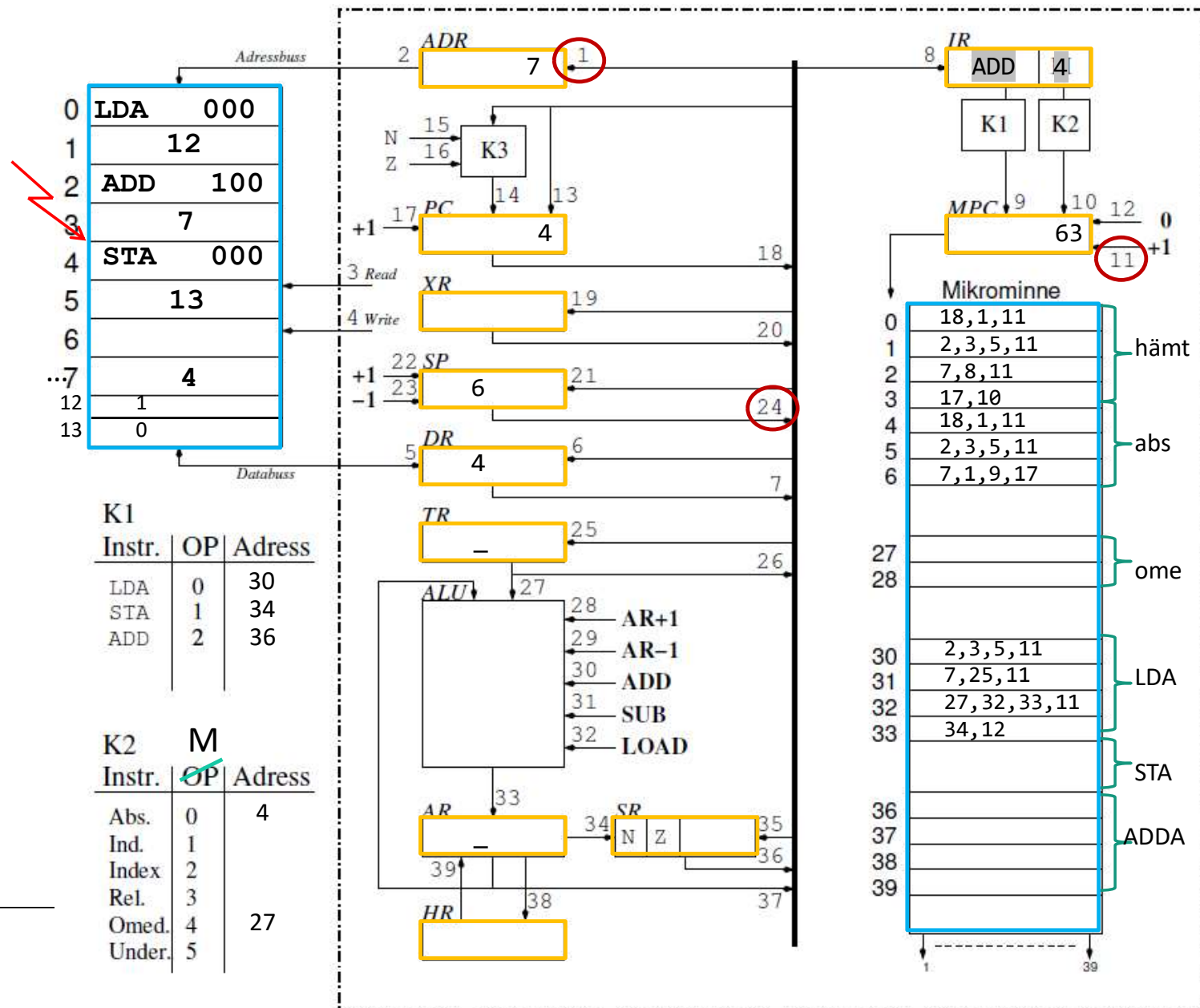
## Spara PC,XR,AR,SR på stacken

```

60: sp->adr, mpc++ 24,1,11
61: pc->dr, mpc++ 18,6,11
62: skriv, sp--, mpc++ 2,4,5,23,11
63: sp->adr, mpc++ 24,1,11
64: xr->dr, mpc++ 20,6,11
65: skriv, sp--, mpc++ 2,4,5,23,11
66: sp->adr, mpc++ 24,1,11
67: ar->dr, mpc++ 37,6,11
68: skriv, sp--, mpc++ 2,4,5,23,11
69: sp->adr, mpc++ 24,1,11
70: sr->dr, mpc++ 36,6,11
71: skriv, sp--, mpc++ 2,4,5,23,11
72: 1->I, mpc++ 40,11
73: 200->PC, mpc=0 42,12

```

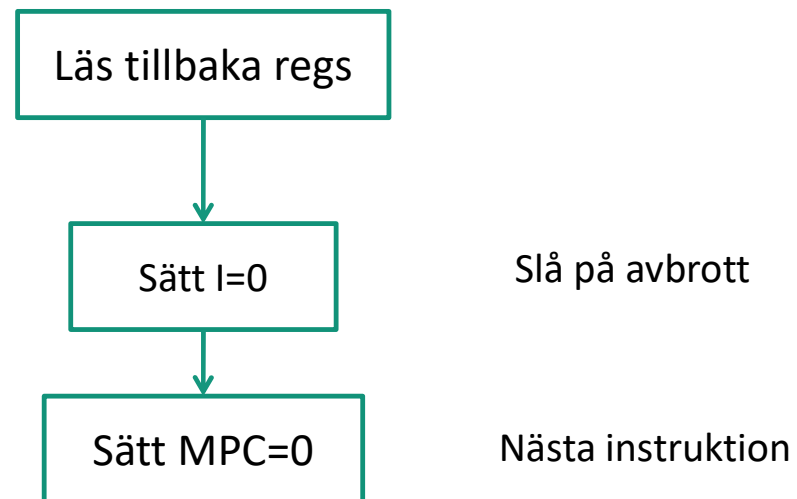
Därefter samma sak för  
XR,AR och SR



# Datorkonstruktion Avbrott

Ny instruktion: **RTI**

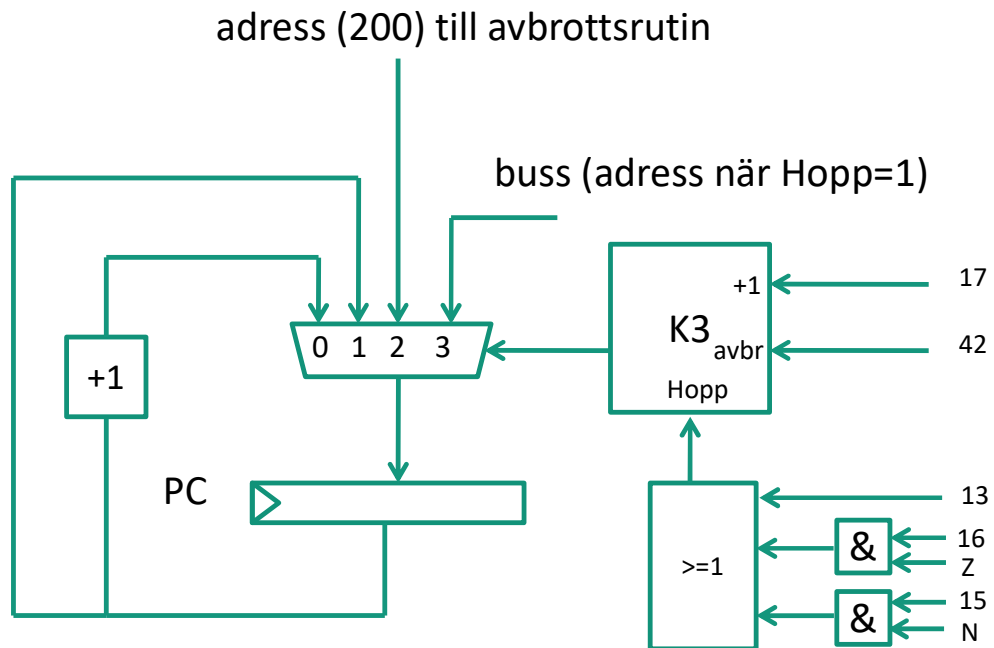
=> Vanlig instruktion, mikrokod för exekveringsfasen



# Hopp JMPN D



# Datorkonstruktion Hopp



K3

| Hopp | +1 | avbr | mux-ing. | PC         |
|------|----|------|----------|------------|
| 1    | 0  | 0    | 3        | buss       |
| 0    | 1  | 0    | 0        | PC+1       |
| 0    | 0  | 1    | 2        | avbr.rutin |
| 0    | 0  | 0    | 1        | PC         |
| ...  |    |      |          |            |

# Hopp

## Mikrokod för JMPN D

Om N=1 : PC+2+D -> PC

Annars : PC+2 -> PC

### Steg 1 : H-fas, som förut

...

3: PC++, K2->mpc 17,10

### Steg 2 : A-fas, Självrelativ

19: PC->ADR, PC++, MPC++

20: PC->TR, minne->DR, MPC++

21: DR->TR, TR->AR, AR->HR, MPC++

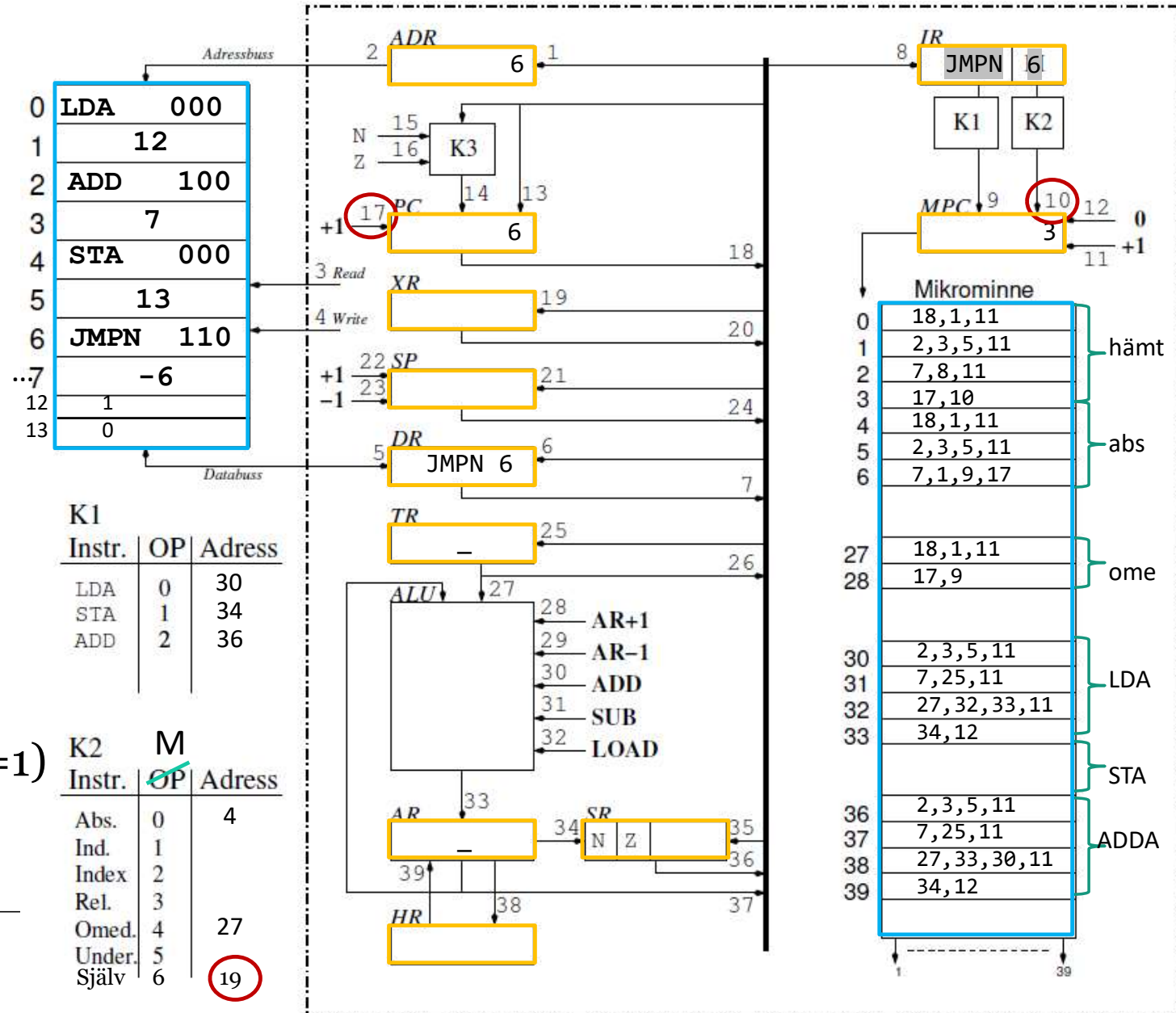
22: AR+TR->AR, MPC++

23: HR->AR, AR->TR, K1->MPC

K1(17)=78

### Steg 3 : Exe, TR->PC (om N=1)

34: TR->PC(N), mpc++ 26,15,12



# Datorkonstruktion Hopp

## Mikrokod för JMPN D

Om N=1 : PC+2+D -> PC

Annars : PC+2 -> PC

### Steg 1 : H-fas, som förut

...

3: PC++, K2->mpc 17,10

### Steg 2 : A-fas, Självrelativ

19: PC->ADR, PC++, MPC++

20: PC->TR, minne->DR, MPC++

21: DR->TR, TR->AR, AR->HR, MPC++

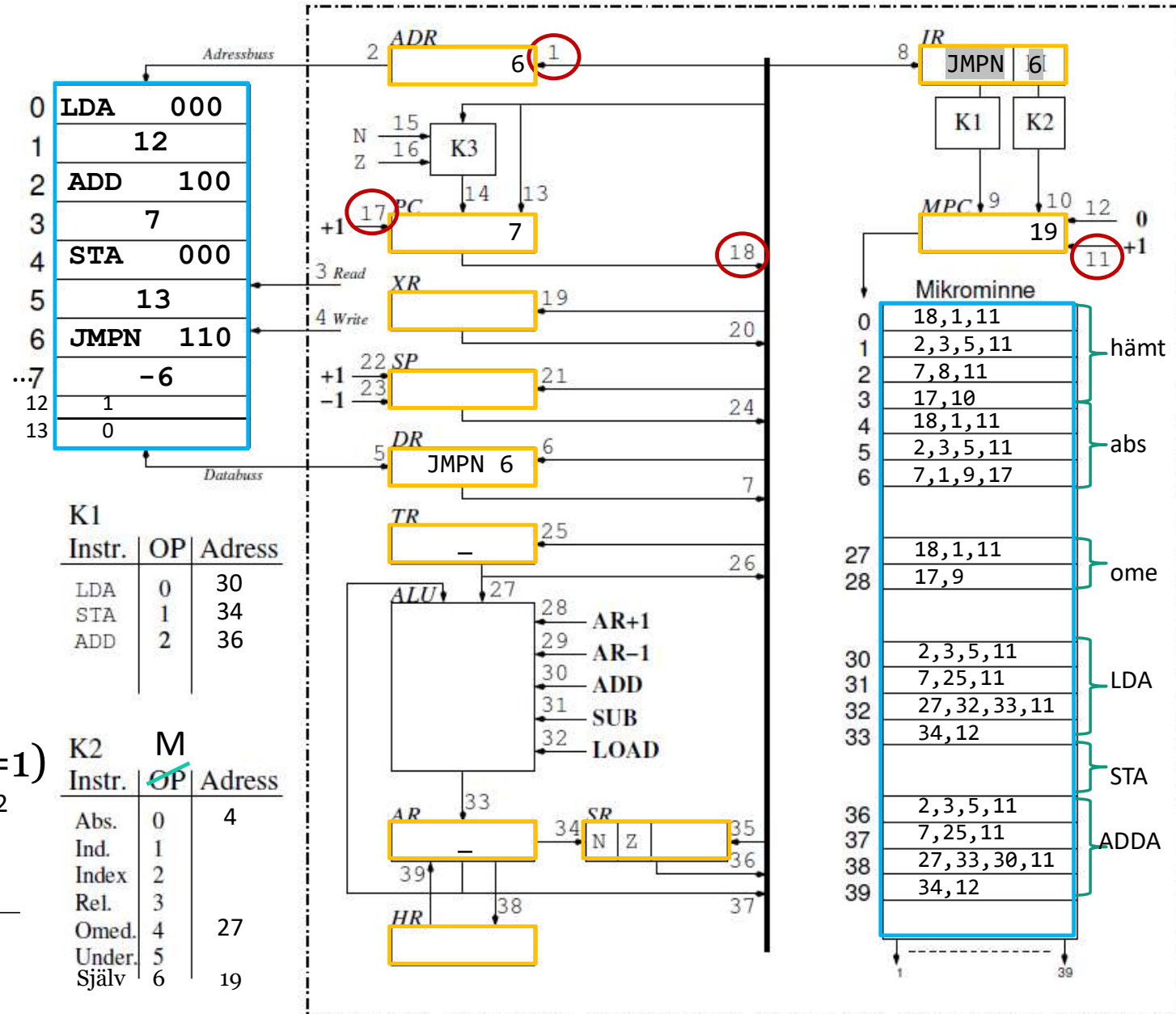
22: AR+TR->AR, MPC++

23: HR->AR, AR->TR, K1->MPC

K1(17)=78

### Steg 3 : Exe, TR->PC (om N=1)

34: TR->PC(N), mpc++ 26,15,12



# Datorkonstruktion Hopp

## Mikrokod för JMPN D

Om N=1 : PC+2+D -> PC

Annars : PC+2 -> PC

### Steg 1 : H-fas, som förut

...

3: PC++, K2->mpc 17,10

### Steg 2 : A-fas, Självrelativ

19: PC->ADR, PC++, MPC++

20: PC->TR, minne->DR, MPC++

21: DR->TR, TR->AR, AR->HR, MPC++

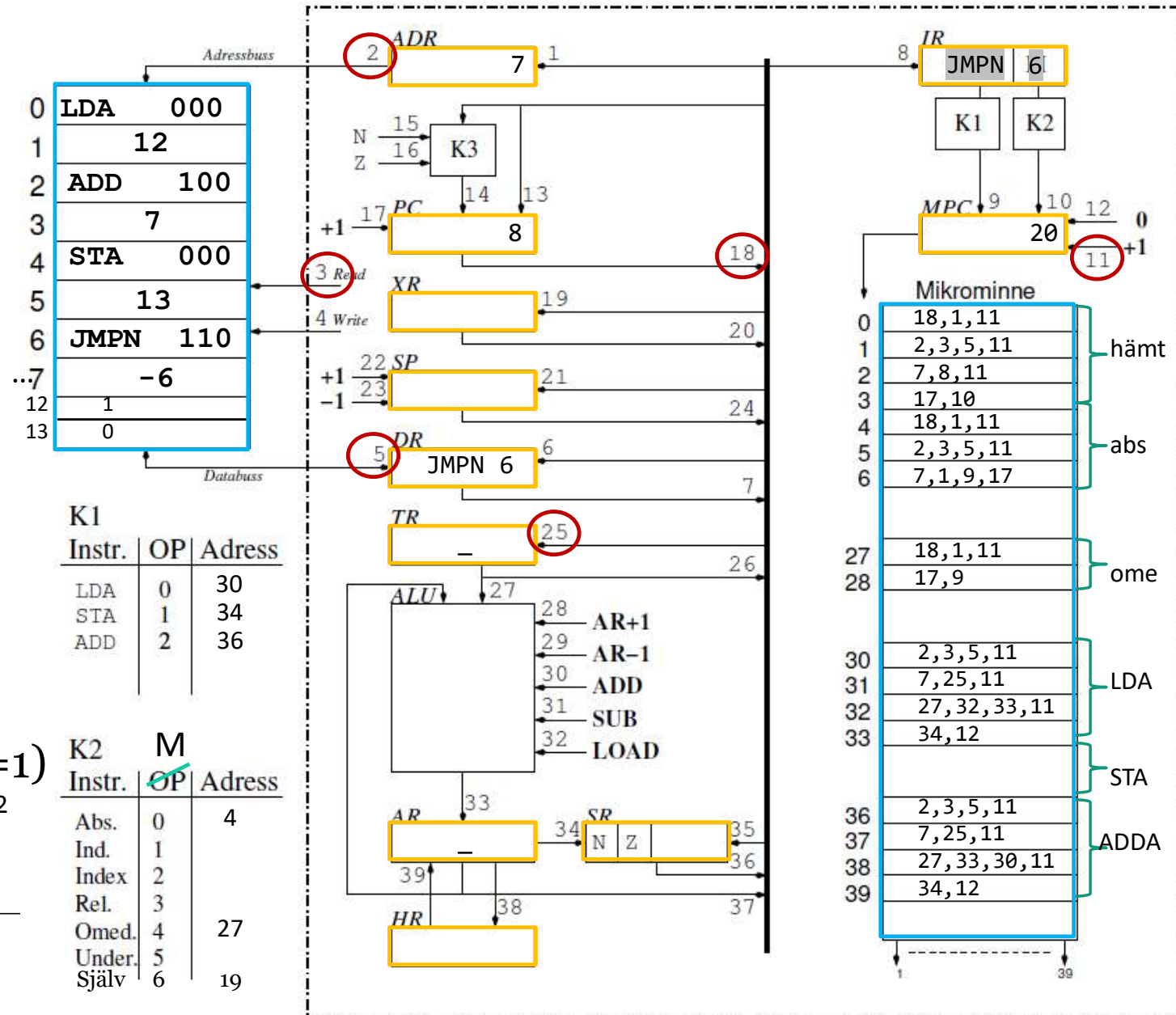
22: AR+TR->AR, MPC++

23: HR->AR, AR->TR, K1->MPC

K1(17)=78

### Steg 3 : Exe, TR->PC (om N=1)

34: TR->PC(N), mpc++ 26,15,12



# Hopp

## Mikrokod för JMPN D

Om N=1 : PC+2+D -> PC

Annars : PC+2 -> PC

### Steg 1 : H-fas, som förut

...

3: PC++, K2->mpc 17,10

### Steg 2 : A-fas, Självrelativ

19: PC->ADR, PC++, MPC++

20: PC->TR, minne->DR, MPC++

21: DR->TR, TR->AR, AR->HR, MPC++

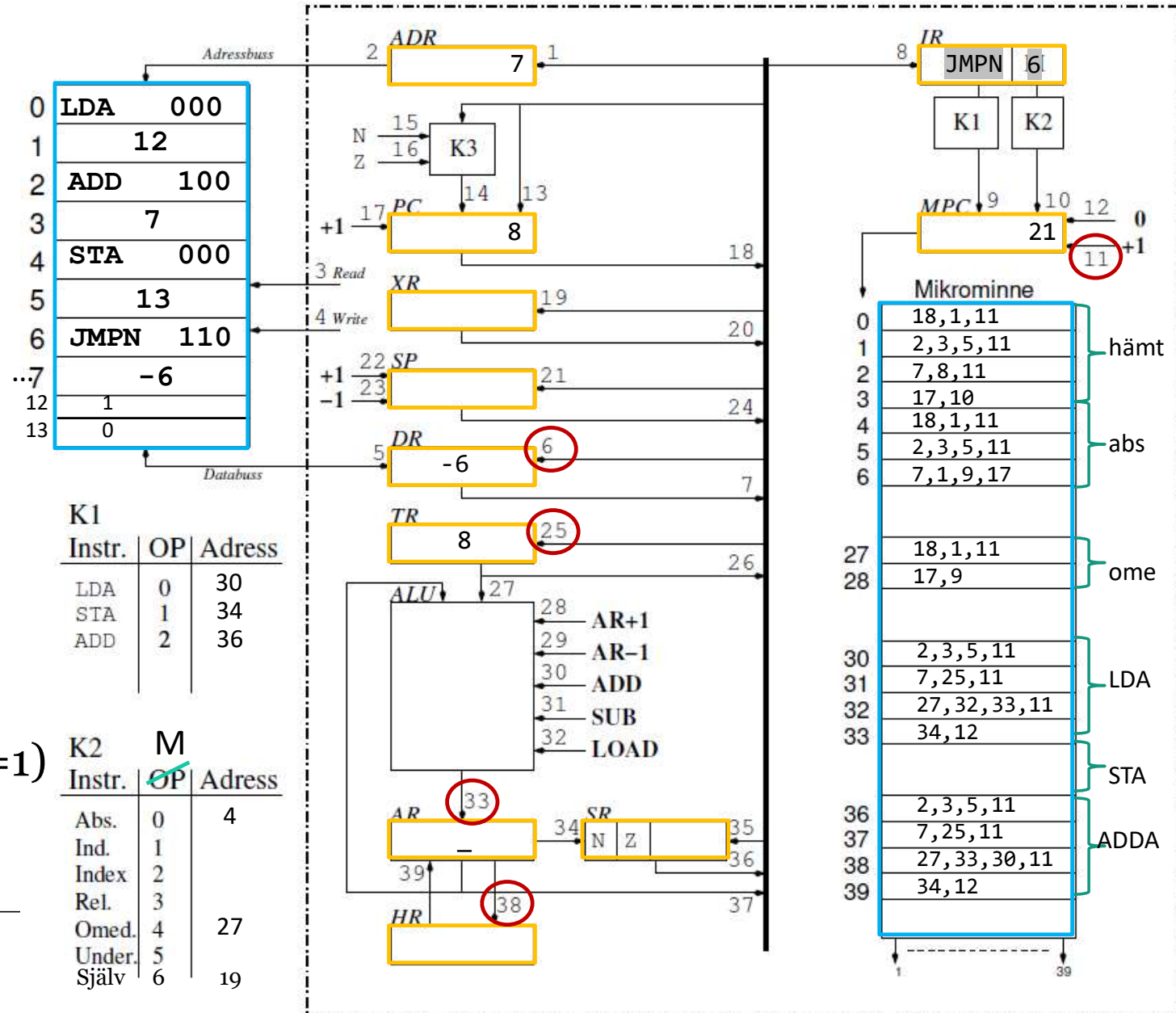
22: AR+TR->AR, MPC++

23: HR->AR, AR->TR, K1->MPC

K1(17)=78

### Steg 3 : Exe, TR->PC (om N=1)

34: TR->PC(N), mpc++ 26,15,12





# Hopp

## Mikrokod för JMPN D

Om N=1 : PC+2+D -> PC

Annars : PC+2 -> PC

### Steg 1 : H-fas, som förut

...

3: PC++, K2->mpc 17,10

### Steg 2 : A-fas, Självrelativ

19: PC->ADR, PC++, MPC++

20: PC->TR, minne->DR, MPC++

21: DR->TR, TR->AR, AR->HR, MPC++

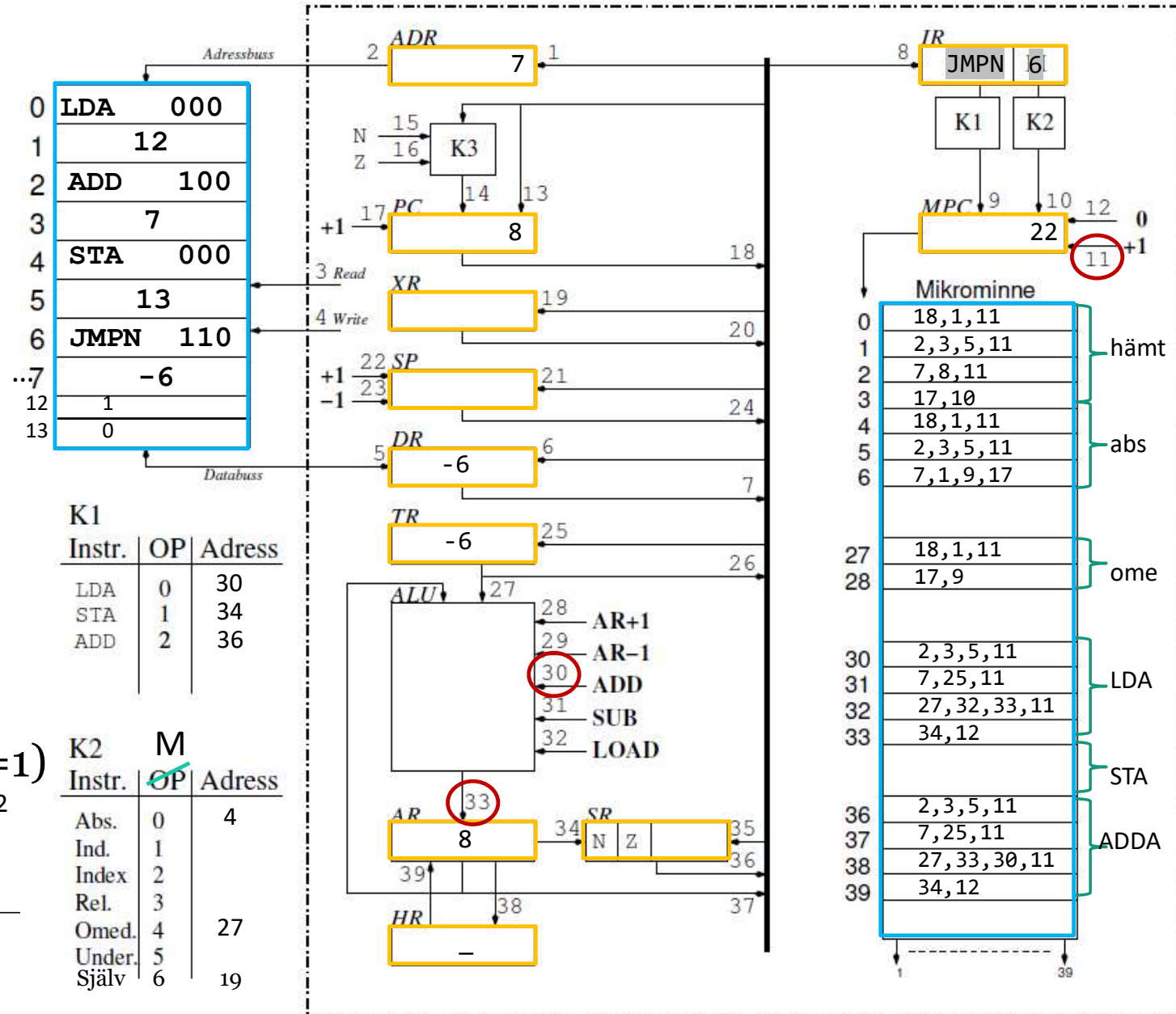
22: AR+TR->AR, MPC++

23: HR->AR, AR->TR, K1->MPC

K1(17)=78

### Steg 3 : Exe, TR->PC (om N=1)

34: TR->PC(N), mpc++ 26,15,12



# Datorkonstruktion Hopp

## Mikrokod för JMPN D

Om N=1 : PC+2+D -> PC

Annars : PC+2 -> PC

### Steg 1 : H-fas, som förut

...

3: PC++, K2->mpc 17,10

### Steg 2 : A-fas, Självrelativ

19: PC->ADR, PC++, MPC++

20: PC->TR, minne->DR, MPC++

21: DR->TR, TR->AR, AR->HR, MPC++

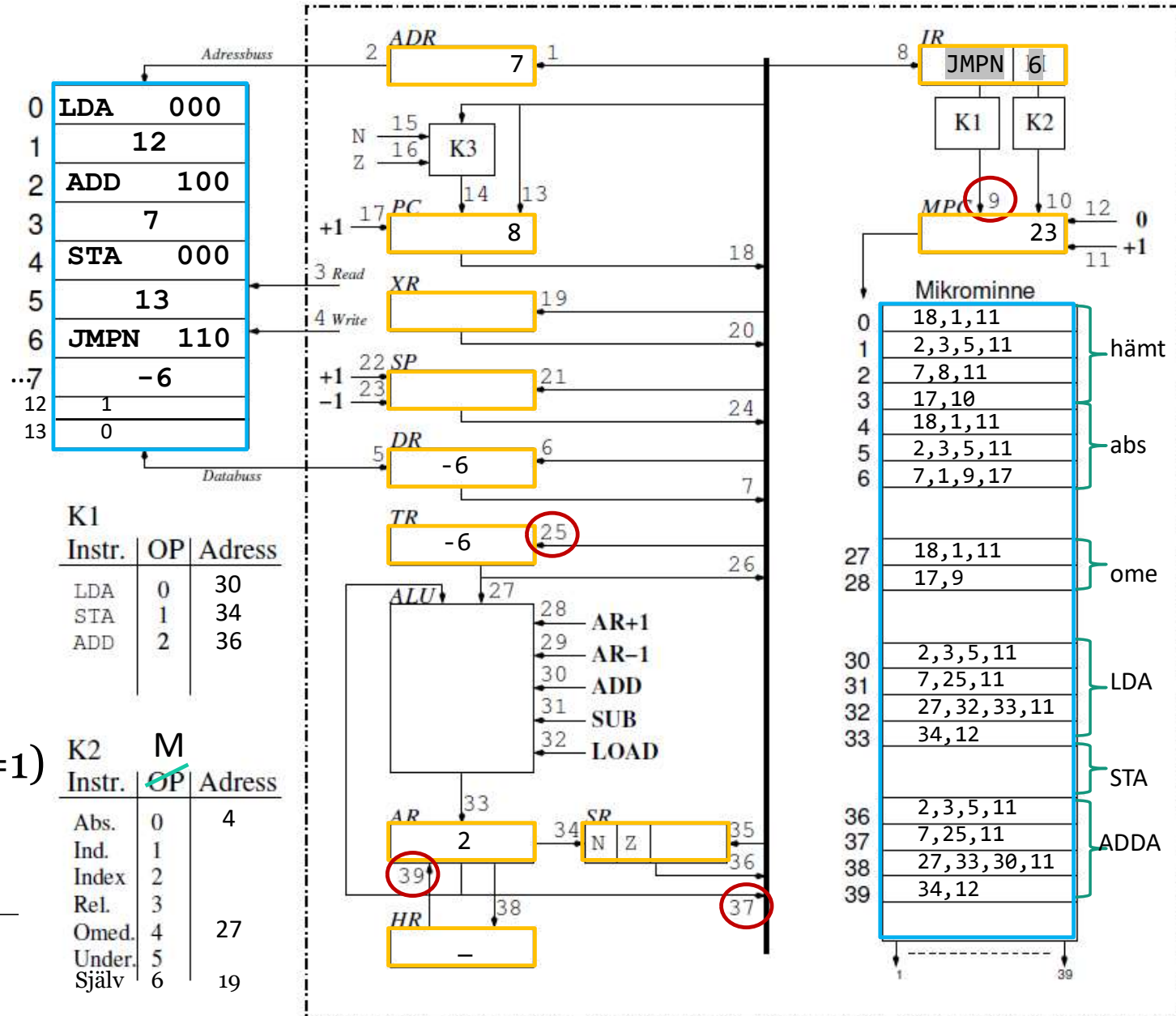
22: AR+TR->AR, MPC++

23: HR->AR, AR->TR, K1->MPC

K1(17)=78

### Steg 3 : Exe, TR->PC (om N=1)

34: TR->PC(N), mpc++ 26,15,12



# Datorkonstruktion Hopp

## Mikrokod för JMPN D

Om N=1 : PC+2+D -> PC

Annars : PC+2 -> PC

### Steg 1 : H-fas, som förut

...

3: PC++, K2->mpc 17,10

### Steg 2 : A-fas, Självrelativ

19: PC->ADR, PC++, MPC++

20: PC->TR, minne->DR, MPC++

21: DR->TR, TR->AR, AR->HR, MPC++

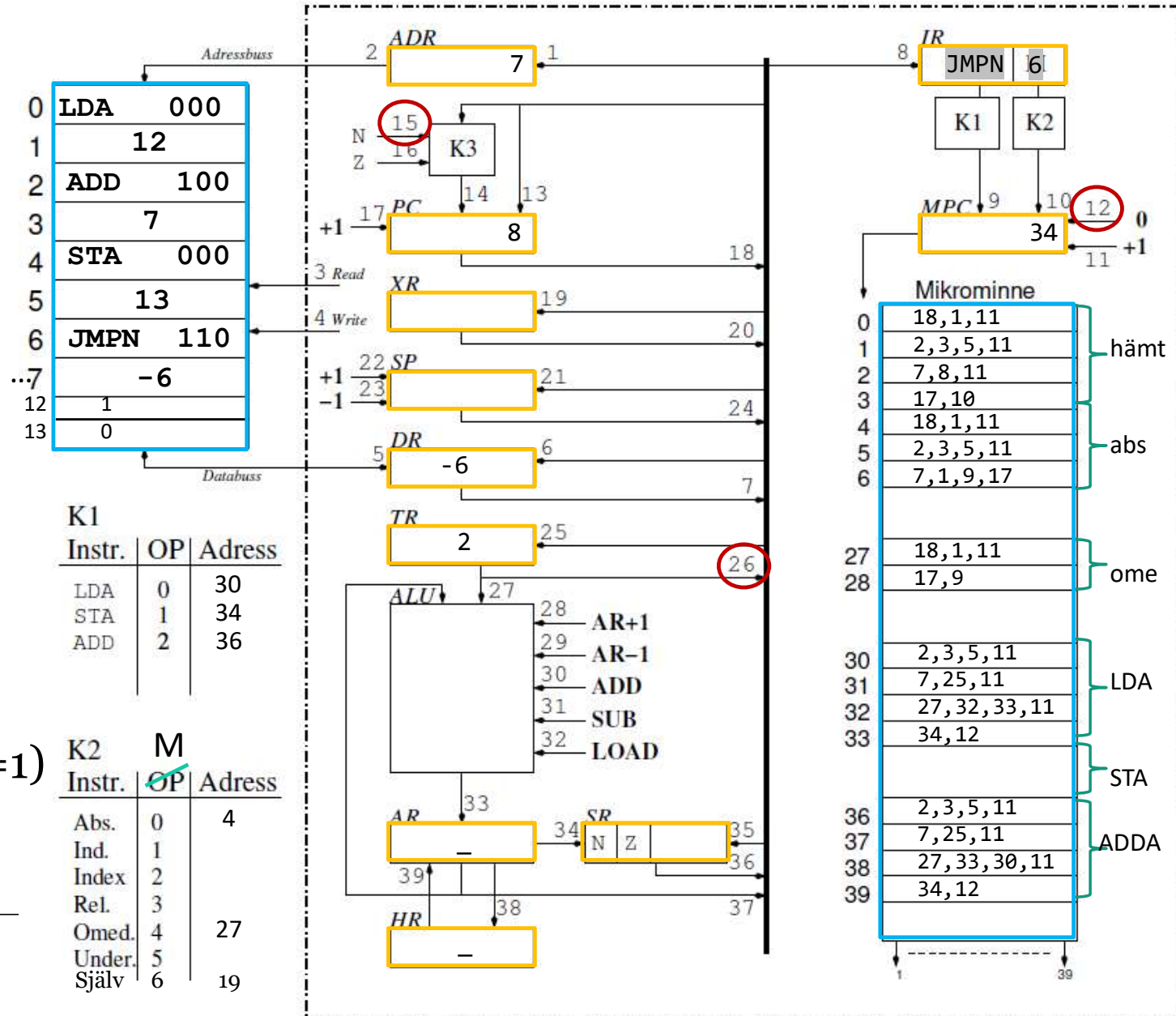
22: AR+TR->AR, MPC++

23: HR->AR, AR->TR, K1->MPC

K1(17)=78

### Steg 3 : Exe, TR->PC (om N=1)

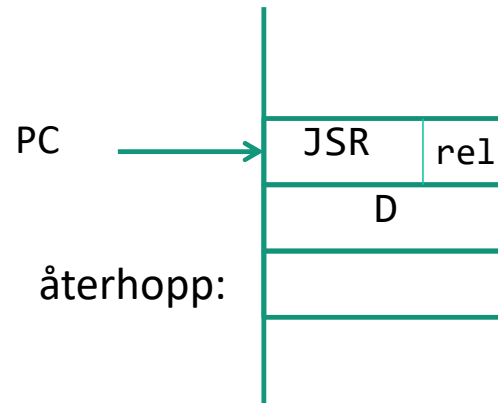
34: TR->PC(N), mpc++ 26,15,12





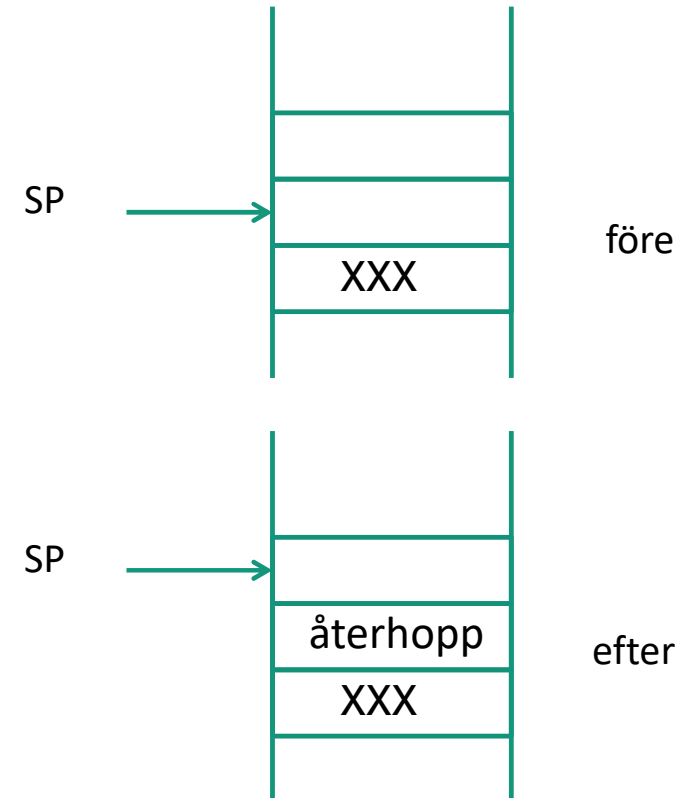
# Subrutinhopp JSR D

# Subrutinhopp JSR D



JSR D

- Självrelativ a-mod  
beräkna hoppadress  $PC+2+D \rightarrow tr$
- exe  
 $PC+2 \rightarrow mem(SP), SP--$   
 $tr \rightarrow PC$



# Subrutinåterhopp RTS



\* underförstådd a-mod

\* exe

```

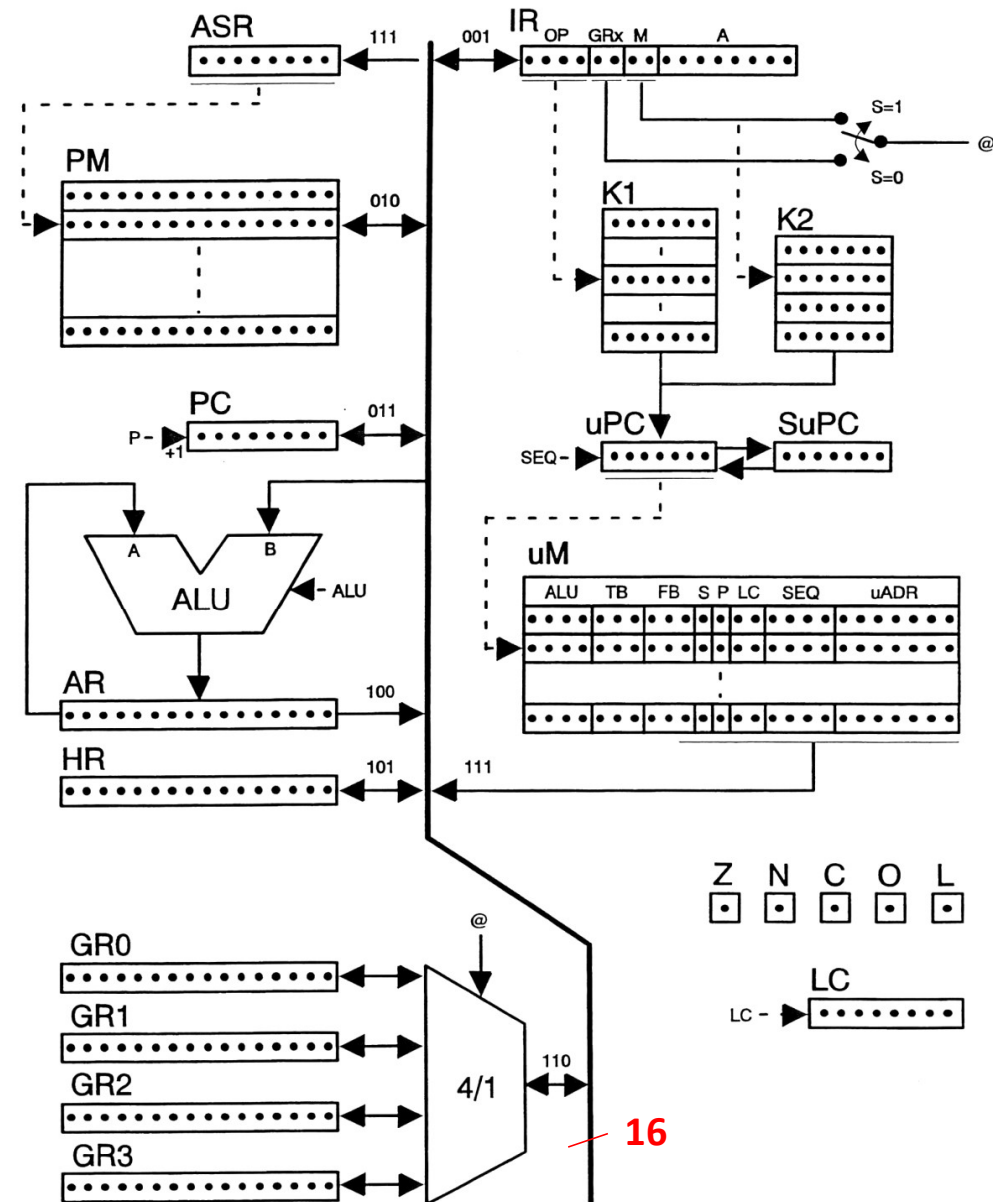
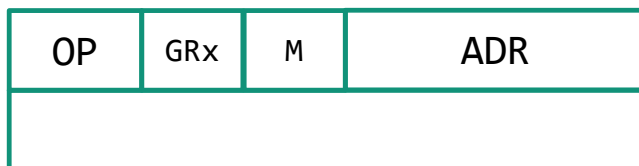
SP++ ; justera SP
M(SP) -> PC ; hoppa tillbaka

```

# Lab1

# Björn Lindskogs dator

- 1,2,3 ska göras
- Frivillig tävling
- 16-bitars maskin (ASR, PC 8 bitar)
- 4 generella reg.
- AR, HR, ASR arbetsreg.
- Avancerad styrenhet
  - Hopp
  - Villkorliga hopp
  - Loopar (LC, L)
  - Subrutiner (SuPC)
  - Konstanter
- GR3 indexreg



# Björn Lindskogs dator

## Uppgift 1: Implementera mikrokod för hämtfas, adresseringsmodsfaser och exekveringsfaser.

uM : Mikrominnets innehåll

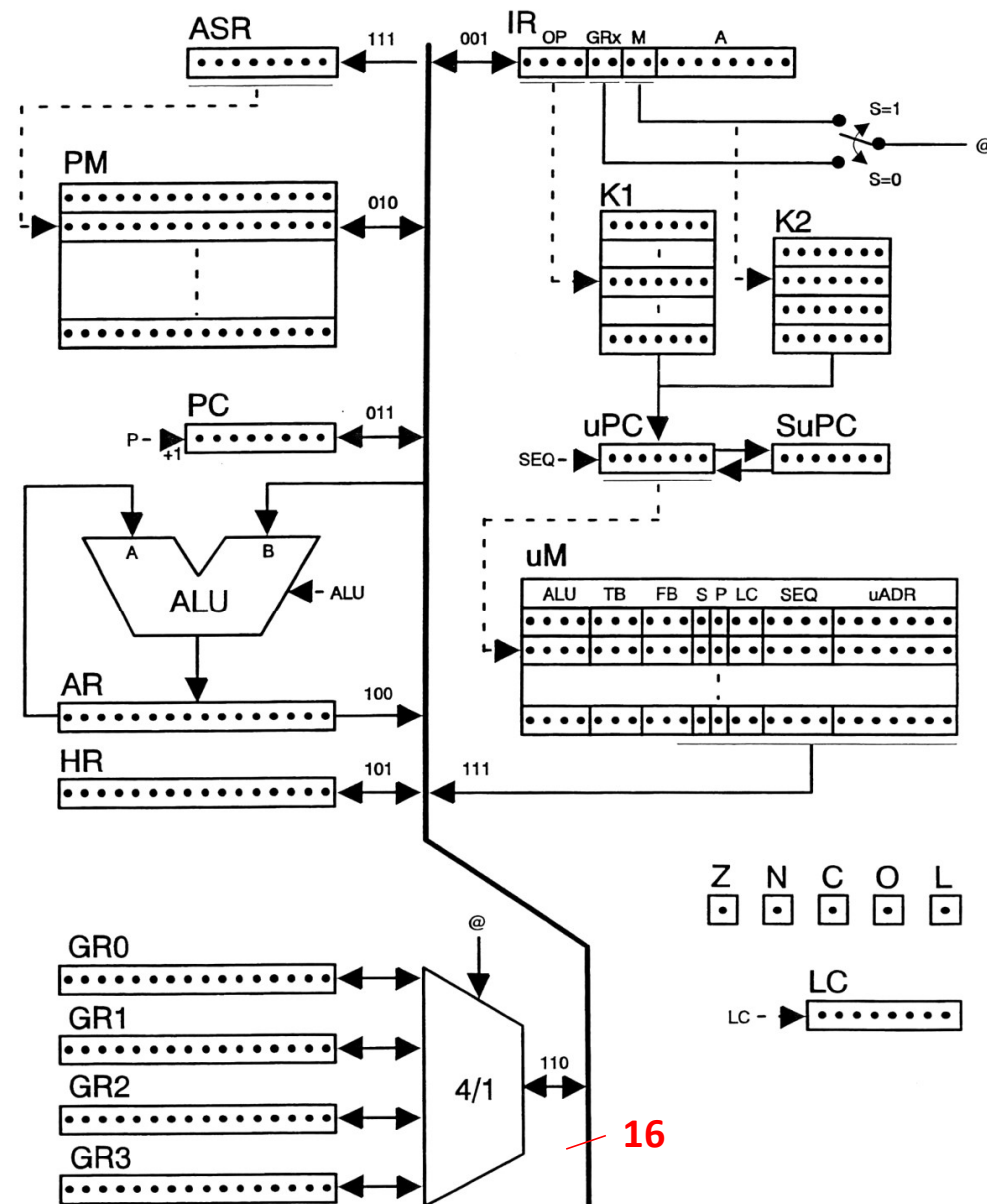
| Adress | ALU | TB    | FB    | S | P | LC | SEQ     | myADR | Hexadecimalt  | Kommentar              |
|--------|-----|-------|-------|---|---|----|---------|-------|---------------|------------------------|
| 00     |     | 0 1 1 | 1 1 1 | 1 |   |    | 0 0 0 0 |       | 0 0 F 8 0 0 0 | ASR:=PC                |
| 01     |     | 0 1 0 | 0 0 1 |   | 1 |    | 0 0 0 0 |       | 0 0 8 A 0 0 0 | IR:=PM, PC:=PC+1       |
| 02     |     |       |       |   |   |    | 0 0 1 0 |       | 0 0 0 0 1 0 0 | uPC:=K2                |
| 03     |     | 0 0 1 | 1 1 1 |   |   |    | 0 0 0 1 |       | 0 0 7 8 0 8 0 | ASR:=IR, uPC:=K1 (Dir) |
| 04     |     |       |       |   |   |    |         |       |               |                        |
| 05     |     |       |       |   |   |    |         |       |               |                        |
| 06     |     |       |       |   |   |    |         |       |               |                        |
| 07     |     |       |       |   |   |    |         |       |               |                        |
| 08     |     |       |       |   |   |    |         |       |               |                        |
| 09     |     |       |       |   |   |    |         |       |               |                        |
| 0A     |     | 0 1 0 | 1 1 0 | 0 |   |    | 0 0 1 1 |       | 0 0 B 0 1 8 0 | LOAD: GRx:=PM(A)       |

|    |      |      |      |      |      |      |
|----|------|------|------|------|------|------|
| 0  | 0000 | 1011 | 0000 | 0001 | 1000 | 0000 |
| =0 | =0   | =B   | =0   | =1   | =8   | =0   |

Instruktionen LOAD börjar (och slutar) på adress 0A.

Dvs, 0A ska skrivas in i K1 på den plats som motsvaras av OP-koden för LOAD, och det är bara att välja ledig plats i K1, så blir den platsen/adressen själva OP-koden.



# Datorkonstruktion Björn Lindsogs dator

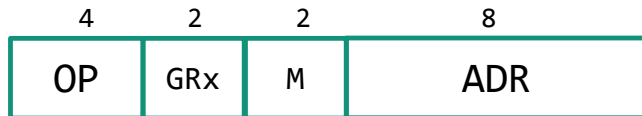
Hur skriver man maskinkoden i programminnet PM?

Antag att vi vill skriva koden för instruktionen:

LOAD GR2,00,7

Dvs, ladda GR2 via A-mod 00, det som finns på adress 7.

Instruktionsformatet ser ut så här:

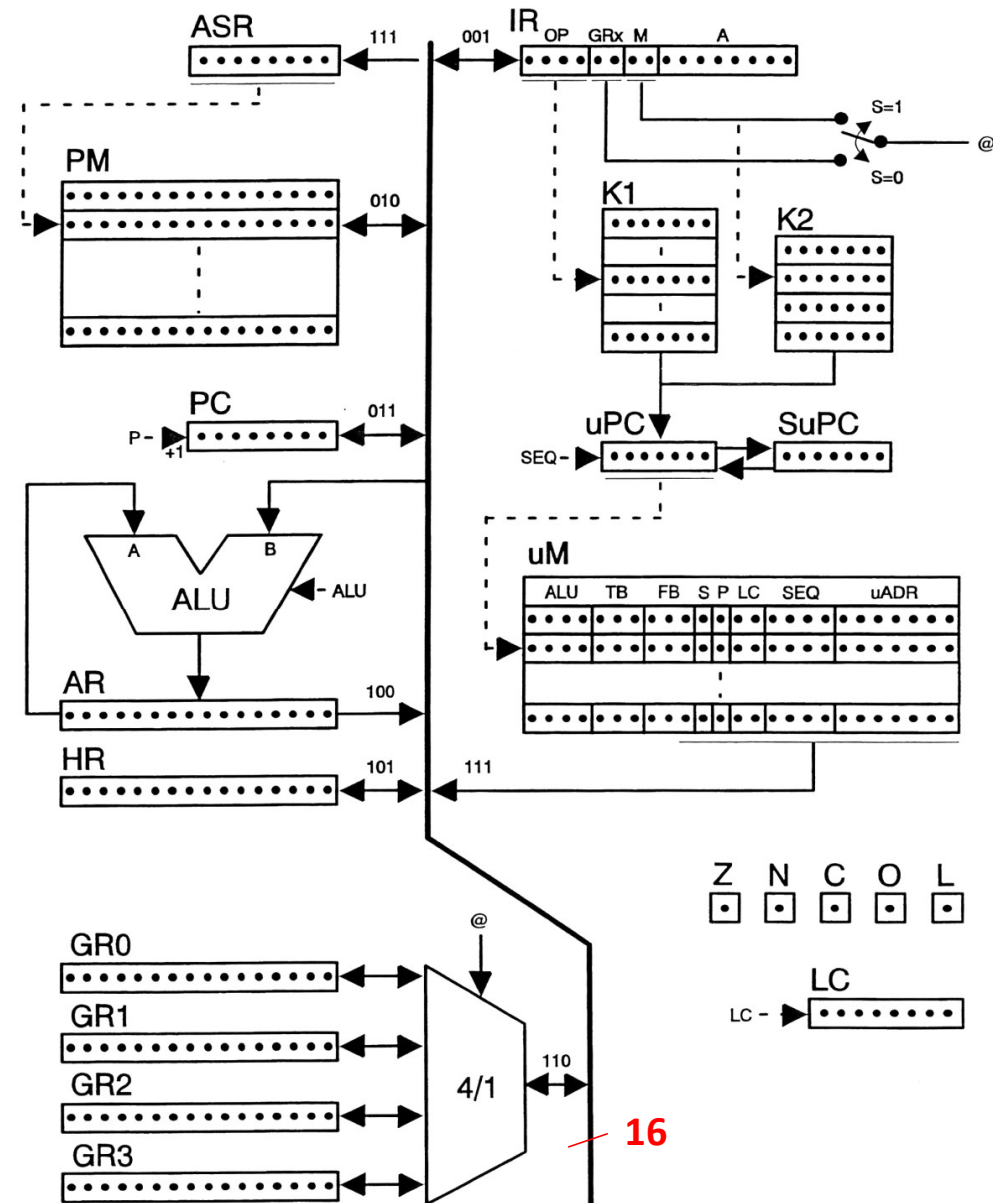


LOAD har OP=0000

GR2 ger GRx=10

A-mod ger M=00

Adress 7 ger ADr = 00000111

$$\text{Dvs, } 0000 \ 10 \ 00 \ 00000111_2 = 0807_{16}$$


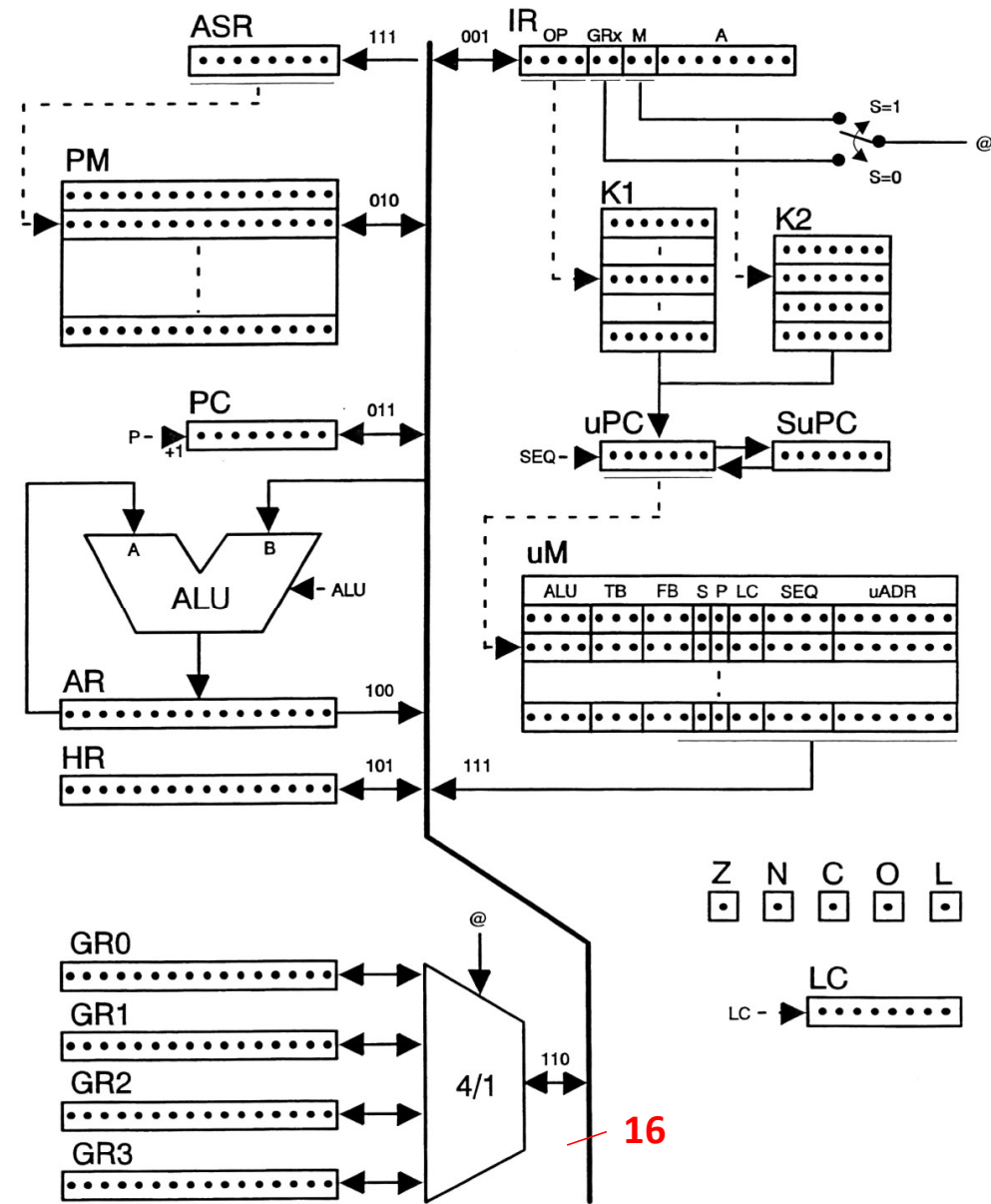
# Björn Lindskogs dator

## Uppgift 2: Summera innehållet på adress \$FE, spara på \$FF

Antag: GR3 loopvariabel, GR2 totalsumma, GR1 ..., GR0 ...

| "Programflöde"                    | "Assemblerkod"  | "Maskinkod" |
|-----------------------------------|-----------------|-------------|
| Init: Initiera loopvariabel       | LOAD GR3,01,0   | 0D00        |
|                                   | 0004            | 0004        |
| Nollställ totalsumma              | LOAD GR2,01,0   | 0900        |
|                                   | 0000            | 0000        |
| Loop: Hämta tal från \$FE         | ...             | ...         |
| Maska ut siffra                   |                 |             |
| Addera till totalsumma            |                 |             |
| Hämta tal från \$FE               |                 |             |
| Skifta 4 steg höger               |                 |             |
| Spara tillbaka på \$FE            |                 |             |
| Räkna ned loopvariabel            |                 |             |
| Hopp till Loop, om > 0            | BNE __,00,??    | 80??        |
| Avsluta: Spara totalsumma på \$FF | STORE GR2,00,FF | 18FF        |
| Halt                              | HALT            | 9000        |

Observera, maskinkoden här är bara ett exempel. Den kan bli annorlunda beroende på hur man implementerar mikrokod, K1 och K2.





# Björn Lindskogs dator

## Uppgift 3: Sortera en lista med 32 tal, minst till störst

Använd bubblesort (eller valfri algoritm)

Osorterad

E0: 92F1 } För varje tal jämför med nästa tal, om det  
E1: 8034 } talet är mindre, byt plats på dom.

E3: 971B

E4: 99FB

E5: 7EF1

E6: 90E8

E7: 3DE3

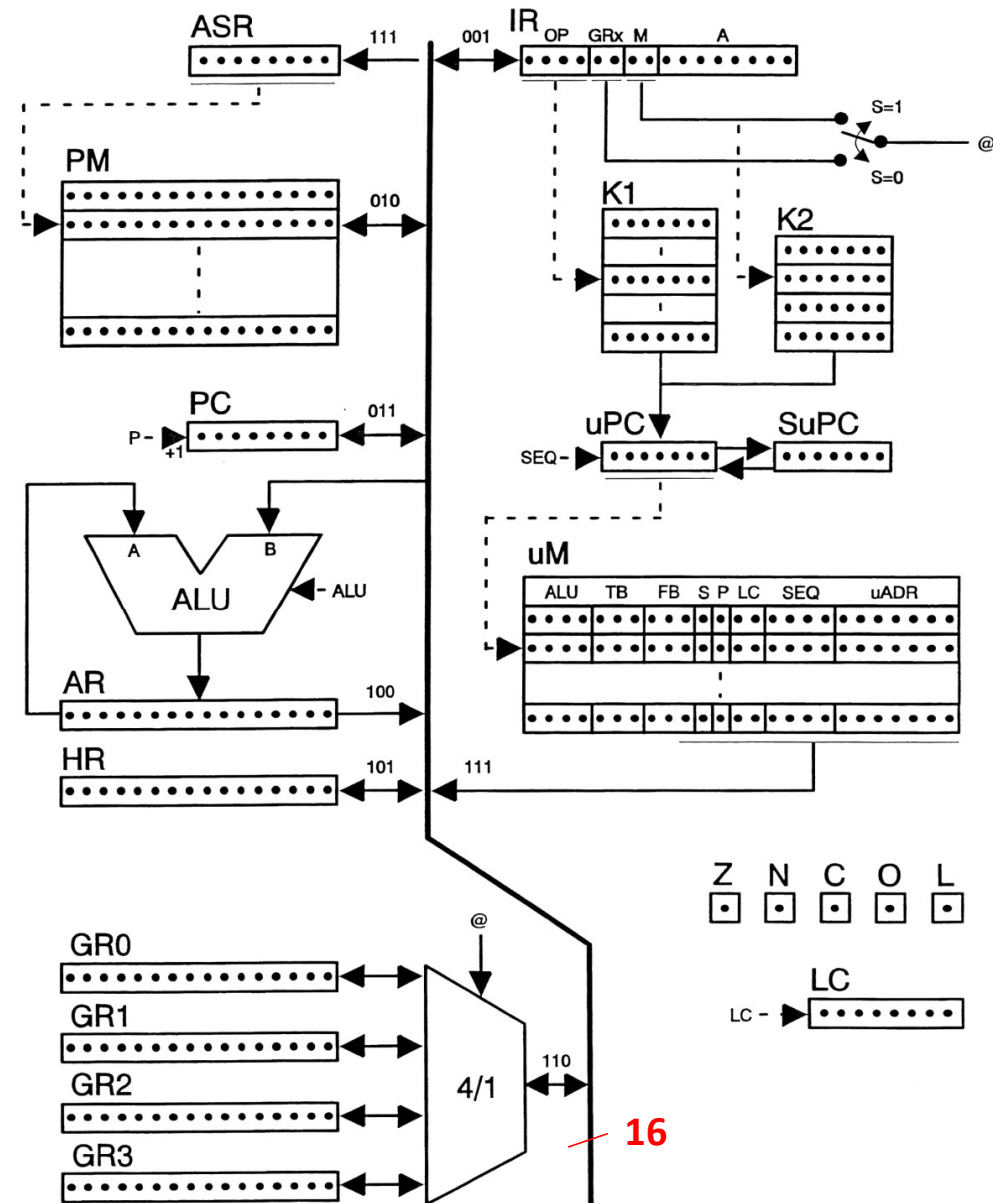
E8: 7351

E9: 53ED

EA: 56A2

EB: DEA5

...



# Björn Lindskogs dator

## Uppgift 3: Sortera en lista med 32 tal, minst till störst

Använd bubblesort (eller valfri algoritm)

Osorterad

E0: 8034

E1: 92F1

E3: 971B

E4: 99FB

E5: 7EF1

E6: 90E8

E7: 3DE3

E8: 7351

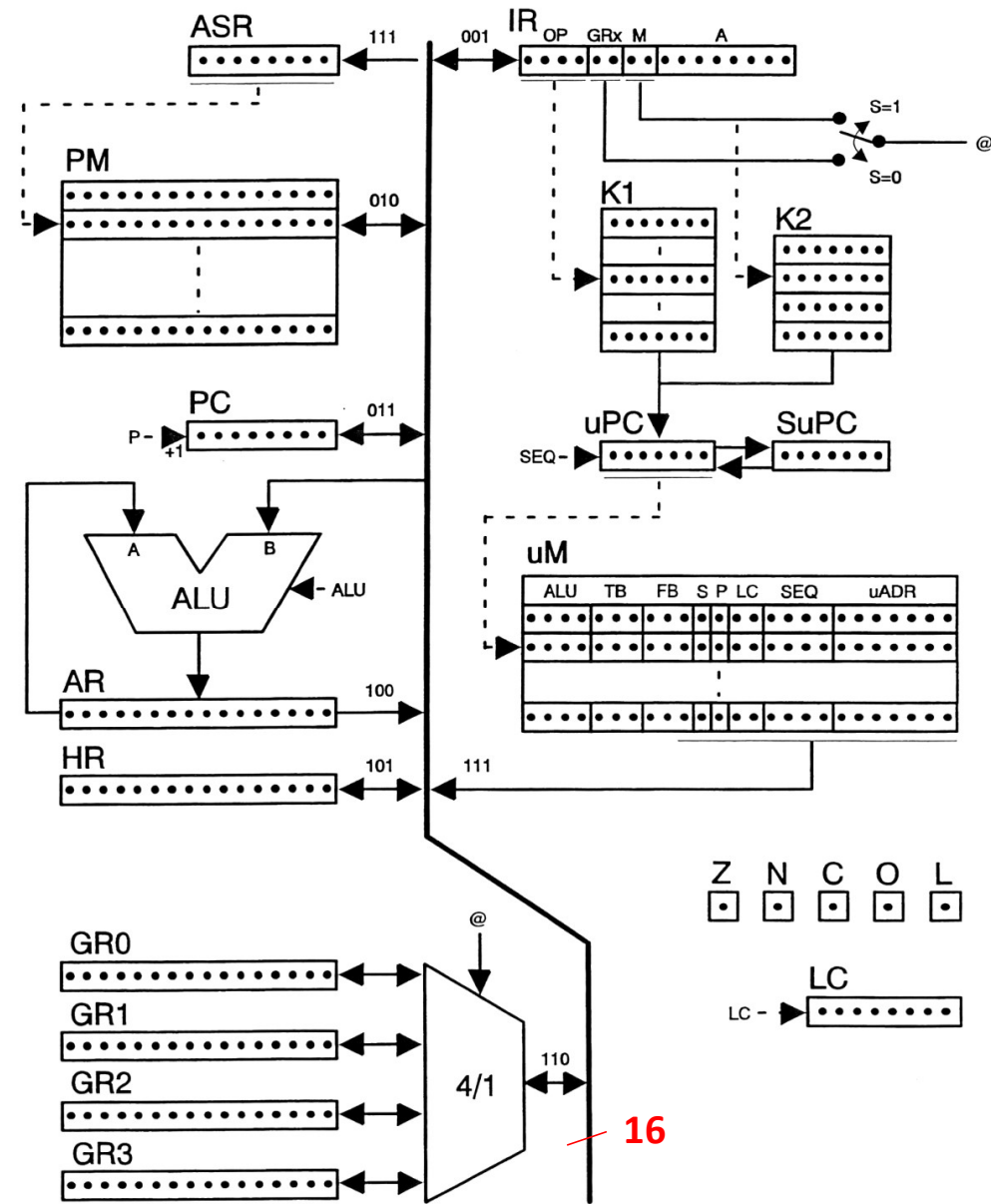
E9: 53ED

EA: 56A2

EB: DEA5

...

} För varje tal jämför med nästa tal, om det talet är mindre, byt plats på dom.





# Björn Lindskogs dator

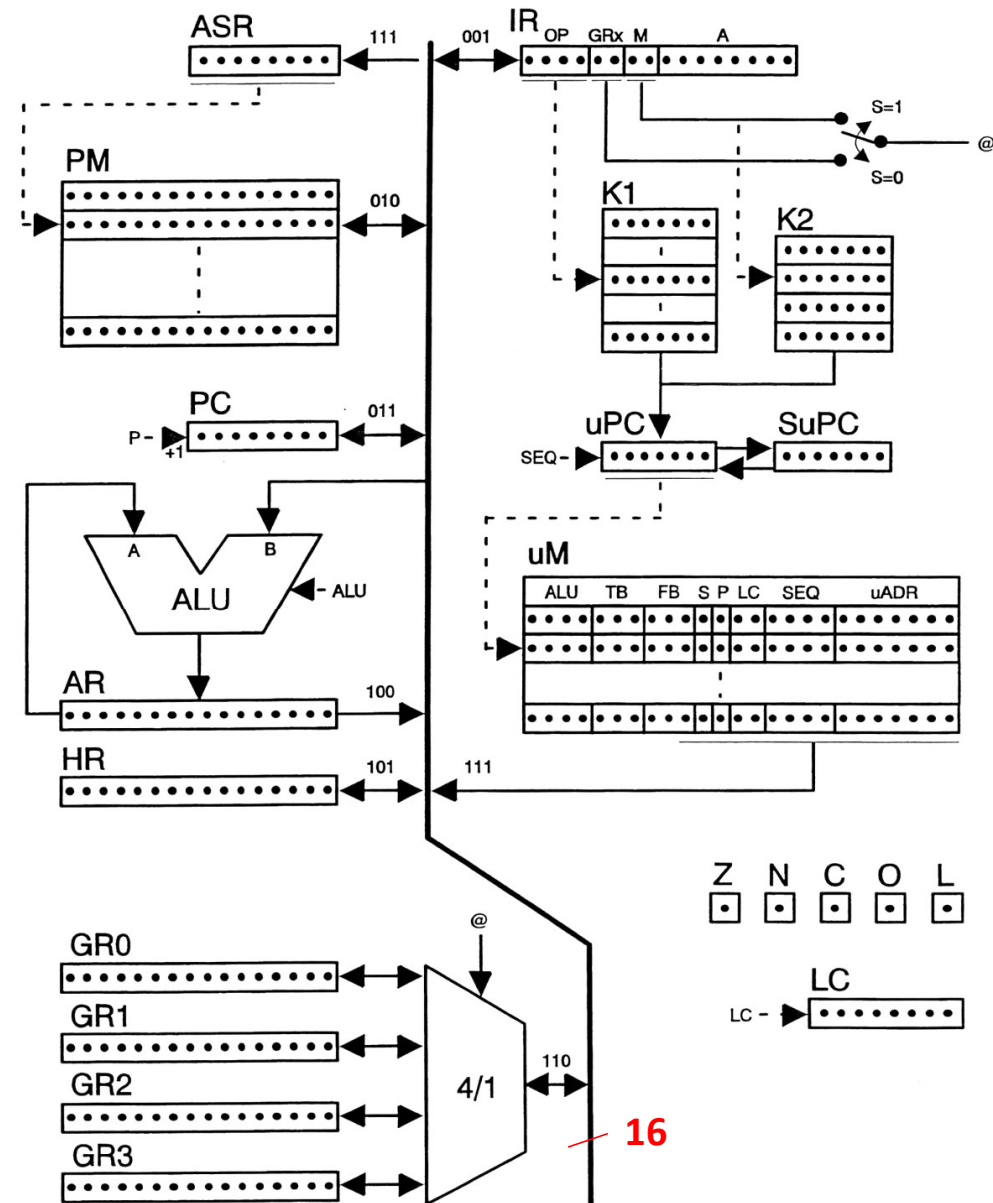
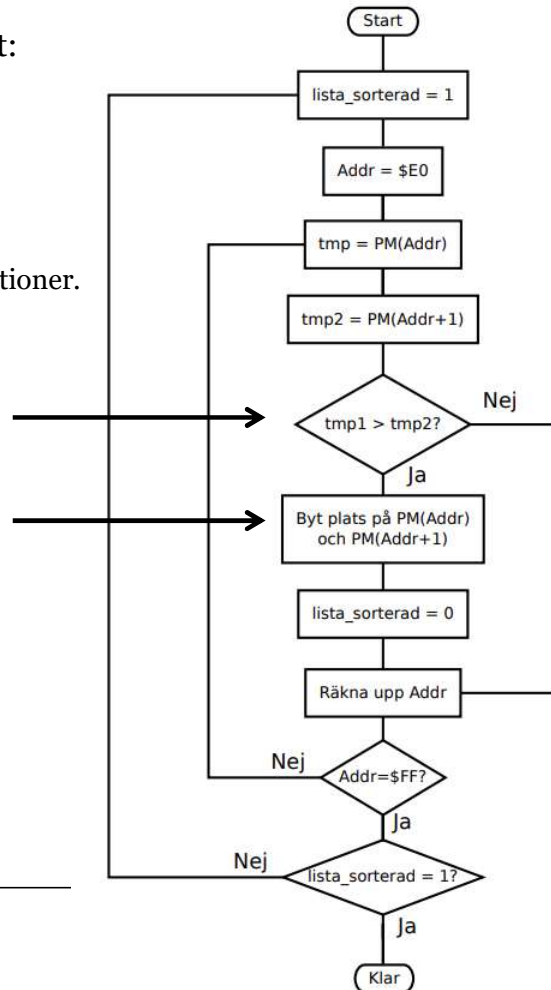
## Uppgift 3: Sortera en lista med 32 tal, minst till störst

Flödesschema för bubblesort:

Det kan bli lättare om man implementerar några fler instruktioner.

Text:  
CMP (compare)  
BGE (Branch if greater or equal)

Och kanske:  
SWAP (swap numbers)

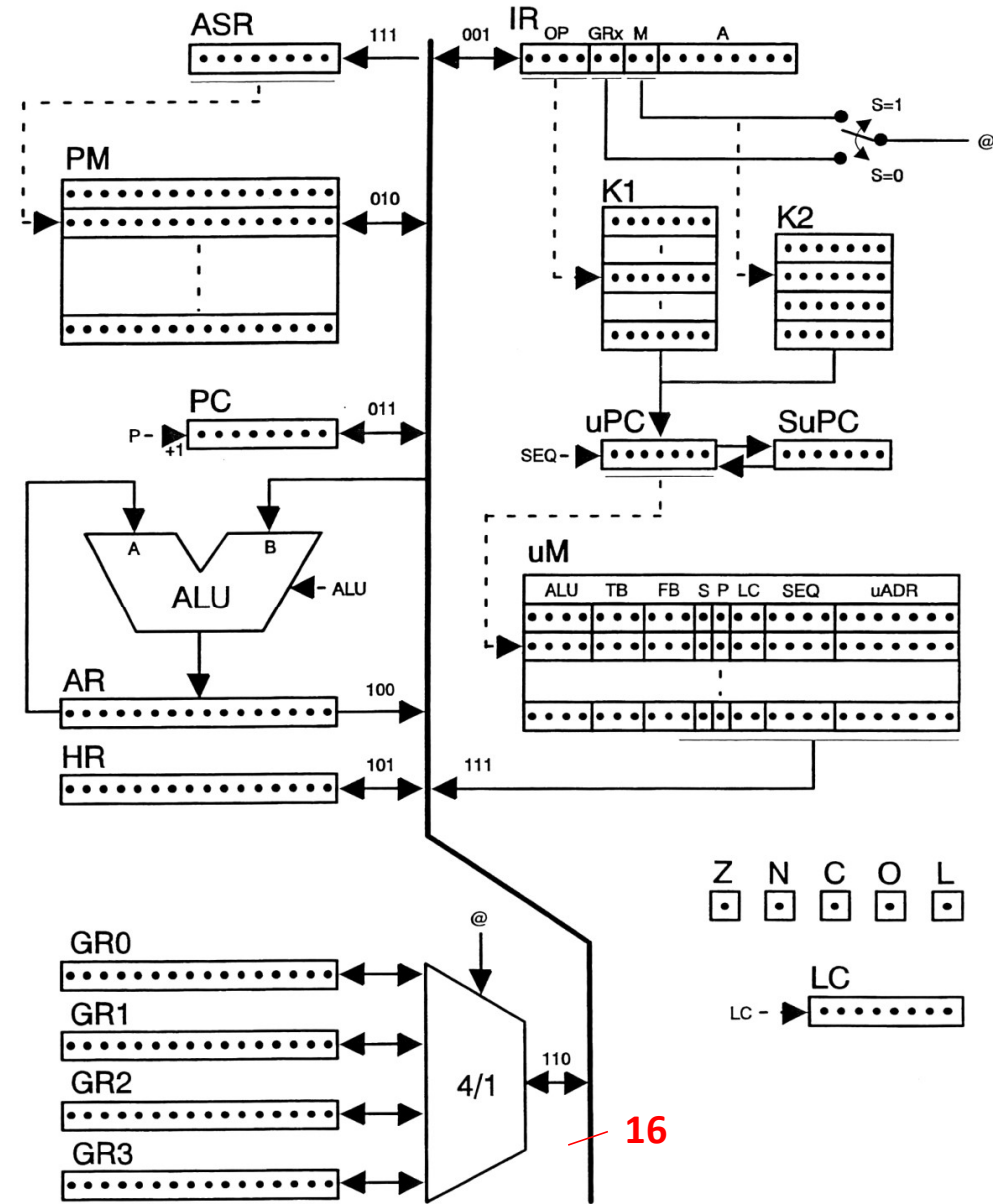
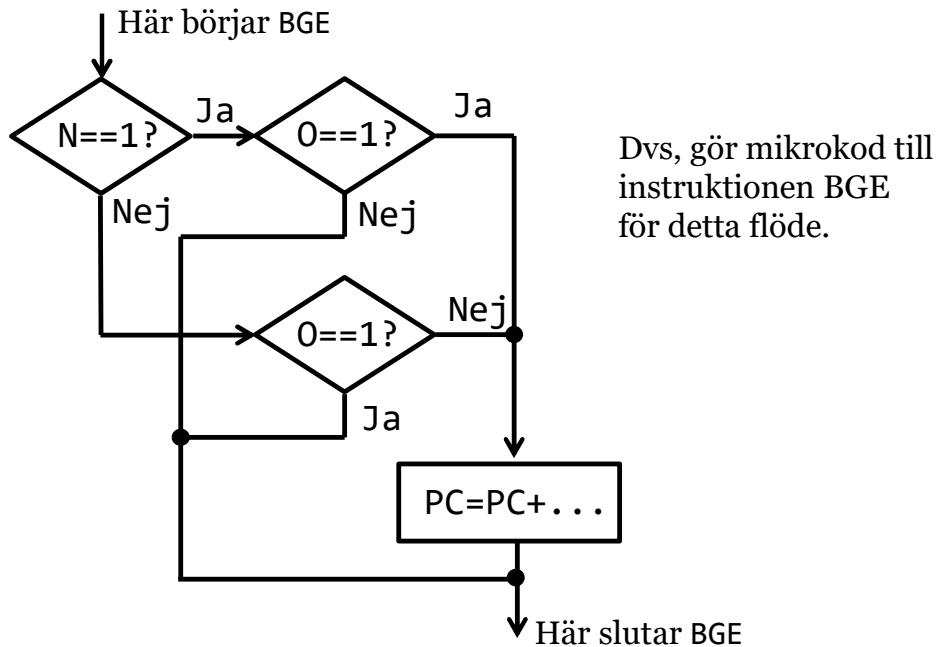


# Björn Lindskogs dator

## Uppgift 3: Sortera en lista med 32 tal, minst till störst

Flödesschema för BGE ( $N \text{ exor } 0 == 0$ ):

Dvs, om N-flaggan är lika med O-flaggan så ta hoppet, annars inte.



# Björn Lindskogs dator

## Frivillig tävling

Gör uppgift 3 med minsta möjliga exekveringstid, dvs minst antal klockcykler.

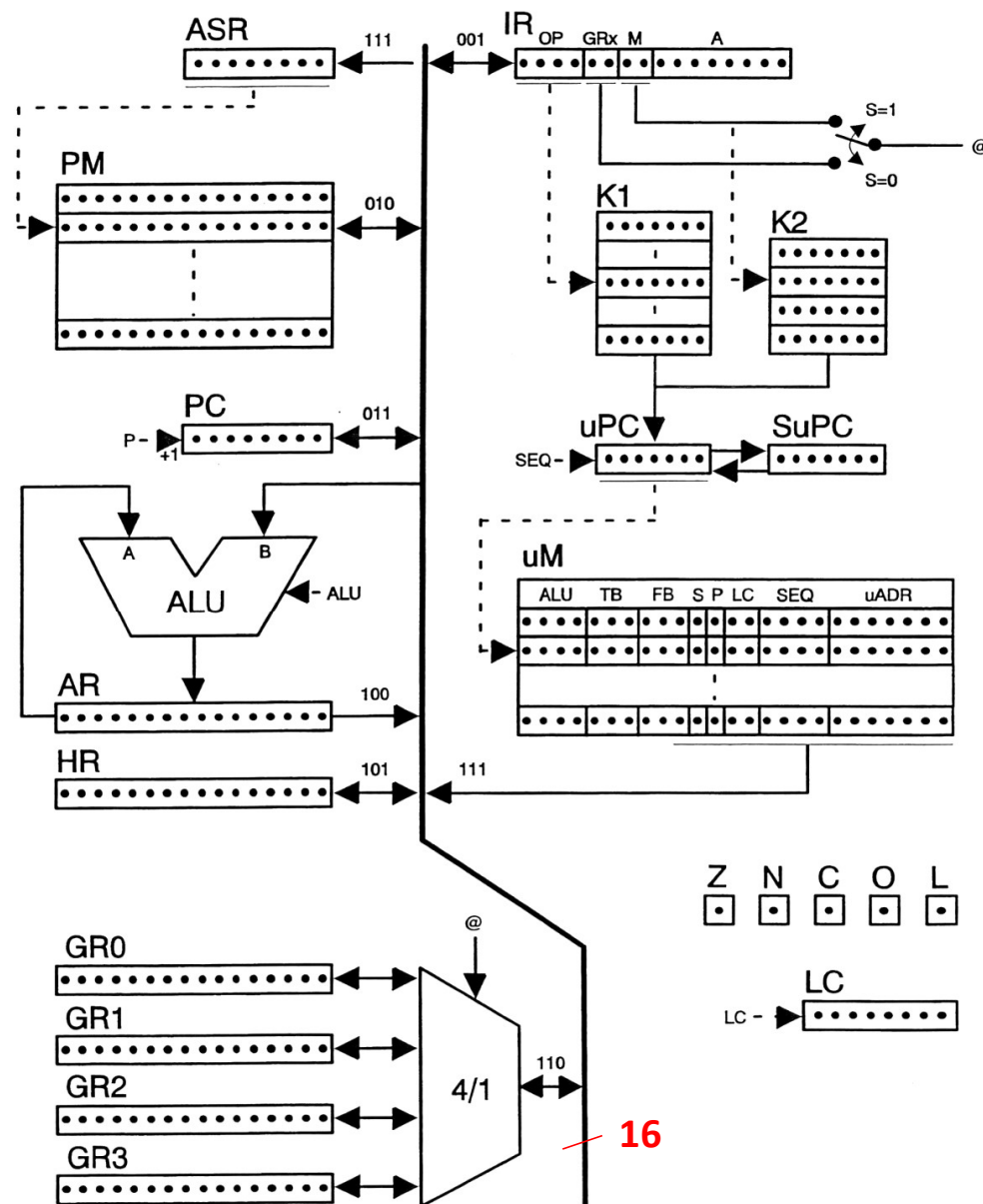
Tips: Det handlar egentligen om att skriva mikrokod för en sorteringsinstruktion.

Tävlingsregler finns på kurshemsidan för Lab1.

Deadline för tävlingsbidrag är sista dagen i VT1.

**Observera**, för att göra projektet i VT2 måste man först ha gjort klart kursens laborationer.

Alla laborationer har direkt bäring på projektet.



Anders Nilsson

[www.liu.se](http://www.liu.se)