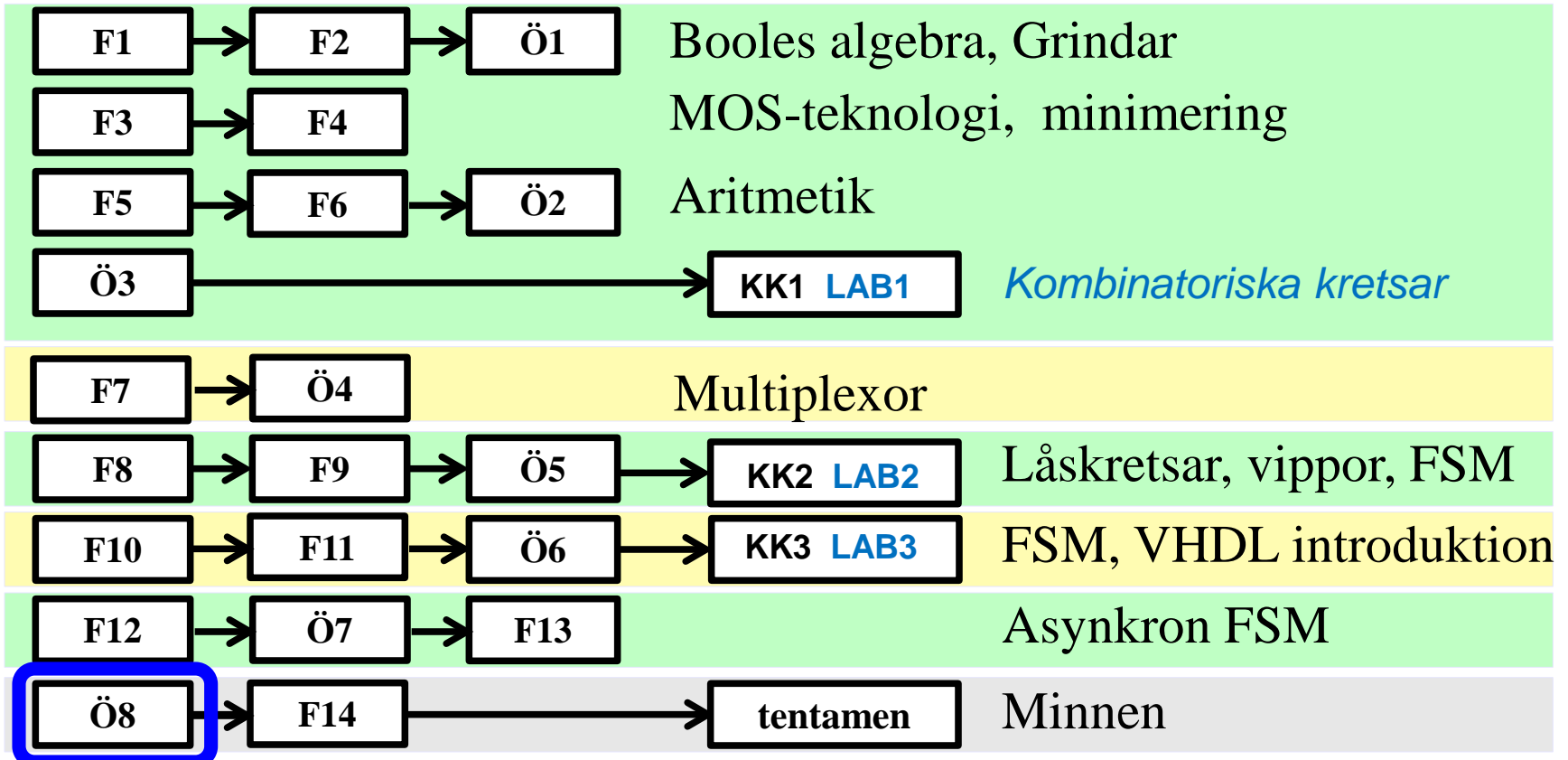


Avkodning av minnen (och I/O)

IE1204 Digital Design



*Föreläsningar och övningar bygger på varandra! Ta alltid igen det Du missat!
Läs på i förväg – delta i undervisningen – arbeta igenom materialet efteråt!*

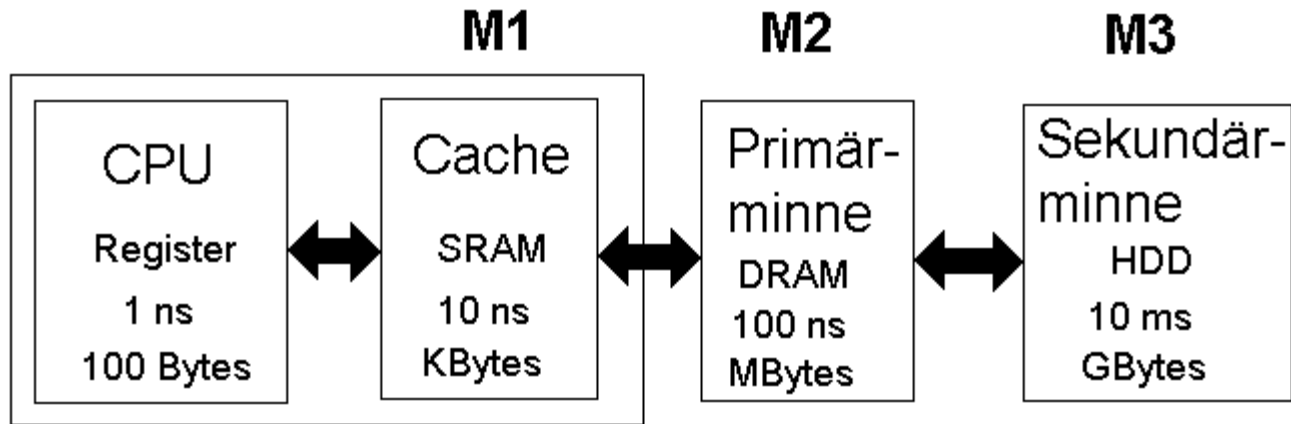
Minnesteknologier

Teknologi	Accesstid	Kostnad \$/GB
SRAM	1 ns	1000
DRAM	50 ns	100
HDD	10 ms	1

Snabba minnen är dyra och billiga minnen är slöa!

Principiella siffror.

Minneshierarki



En tre nivåers minneshierarki. De snabbare minnestyperna används som "buffertar" mot de långsammare.

Principbild

Minne och minneskapslar

Minne:

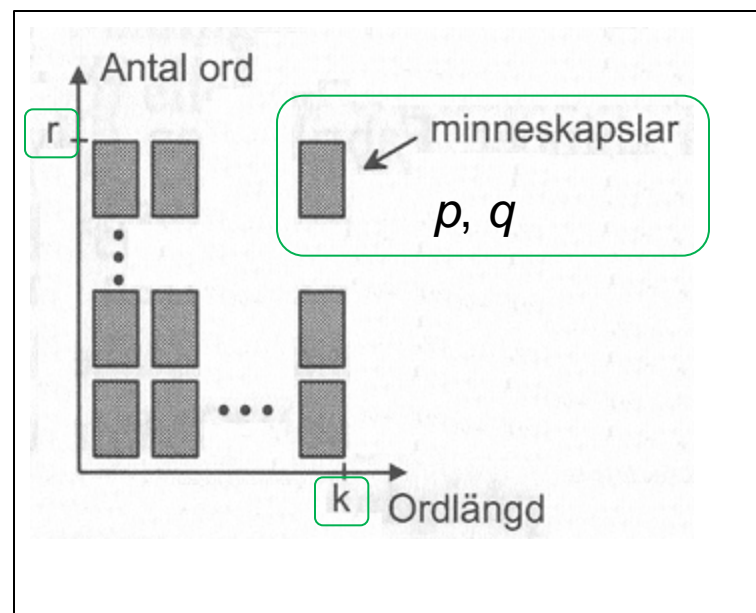
N ord med ordlängden M bitar

Minneskapsel:

p ord med ordlängden q bitar

- Antalet kapselrader $r = N/p$
- Antalet kapselkolumner $k = M/q$
- Antalet kapslar $K = r \times k$

$$K = r \times k$$

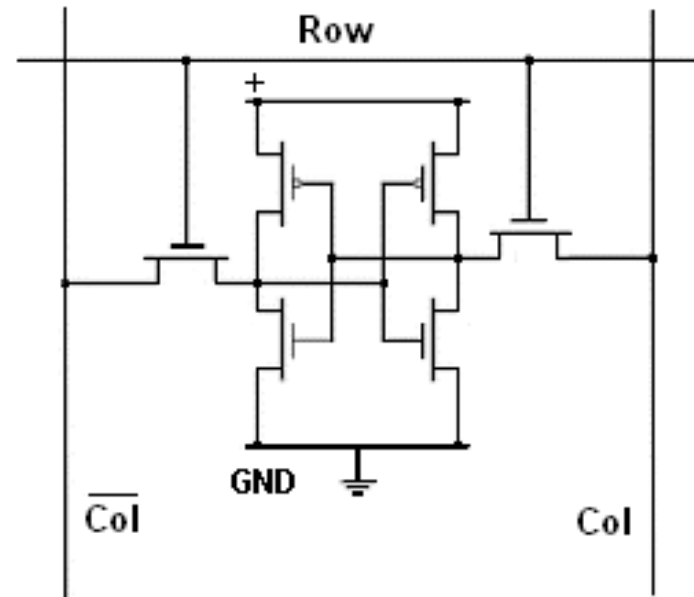
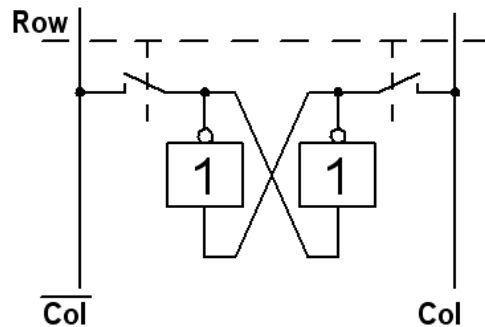


N, M

SRAM

Varje *bit* i ett CMOS SRAM består av en **låskrets** uppbyggd av sex MOS-transistorer.

Minnescellen är i princip en SR-latch.

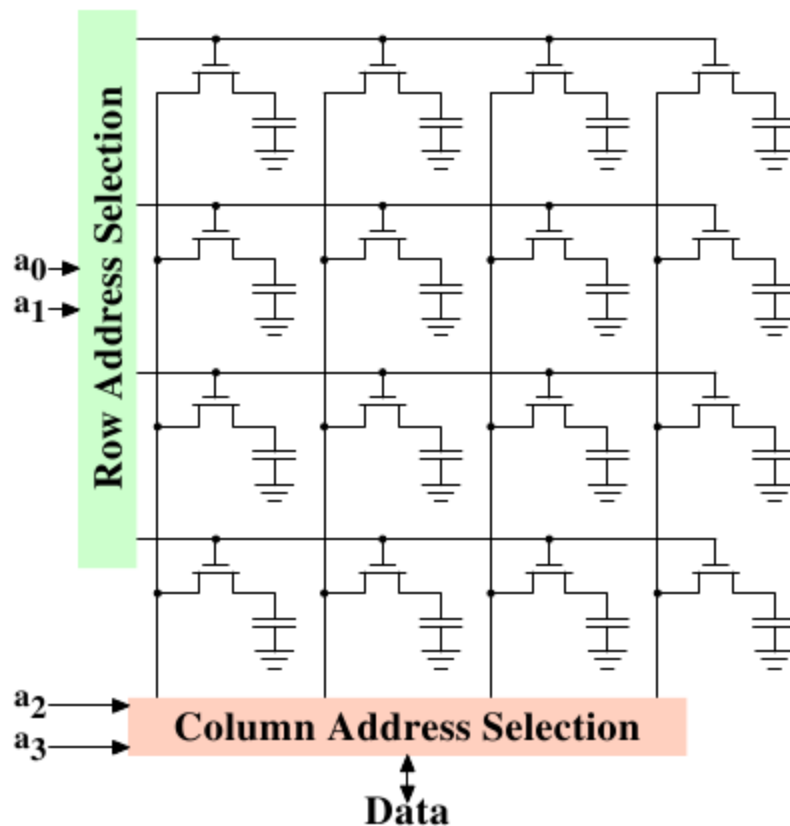
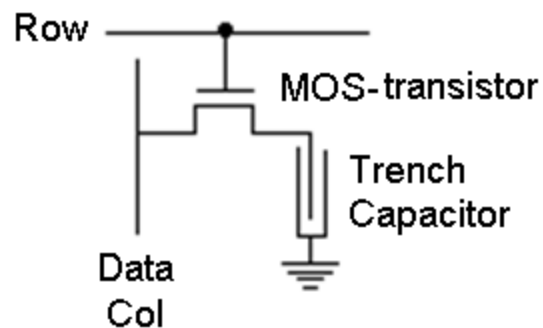


DRAM

Varje bit i ett DRAM består av *en* transistor och *en* minneskondensator.

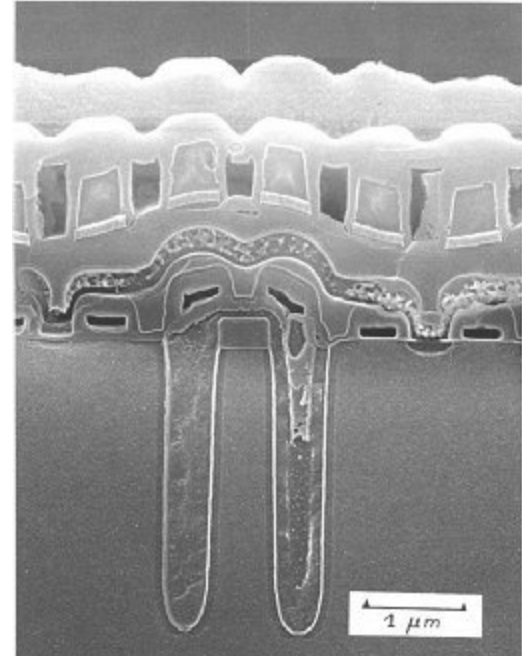
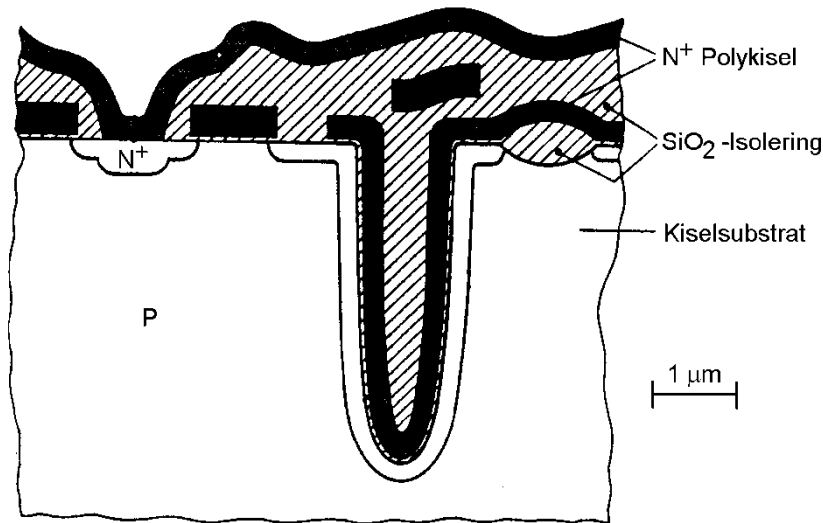
En laddad kondensator läcker ut laddningen efter ett tag. Periodiskt måste alla kondensatorer undersökas och de som har laddning kvar måste då återladdas. Detta kallas för **Refresh**.

Det sköts av kretsar inuti minnet.

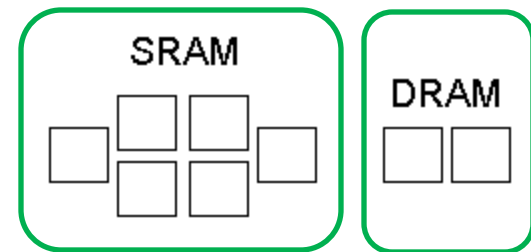


Kondensatorn byggs på djupet

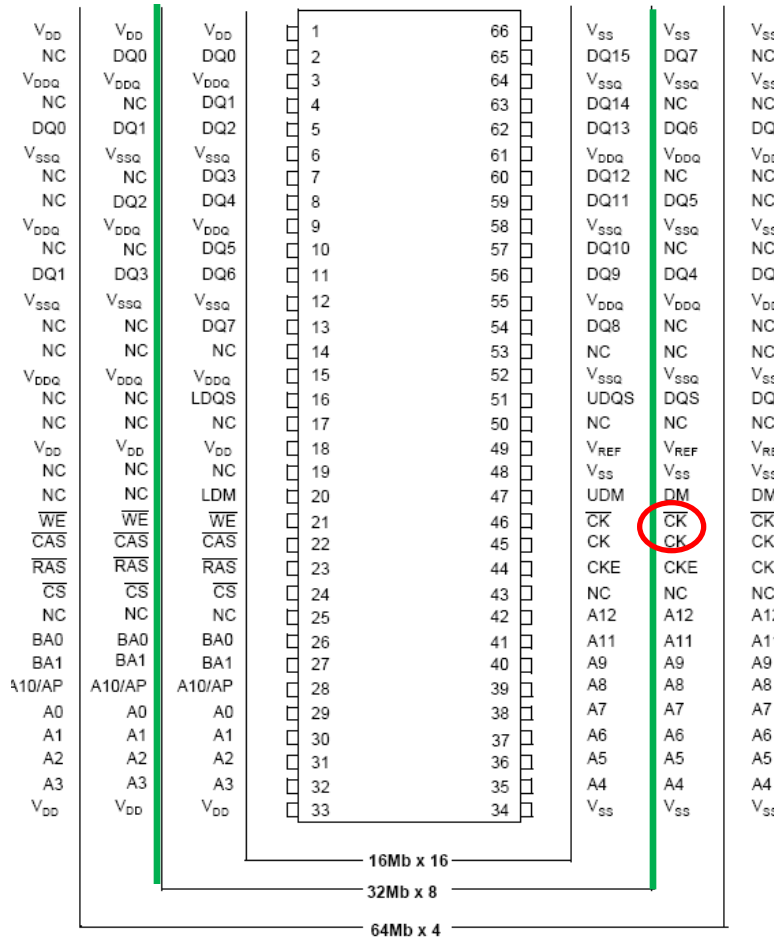
Trench Capacitor (trench = dike)



En bit i ett DRAM tar samma plats som *två* MOS-transistorer. En bit i SRAM som *sex* MOS-transistorer!



Infineon HYB25D25640 256 Mbit SDRAM



Kapsel 256Mbit (32M×8)

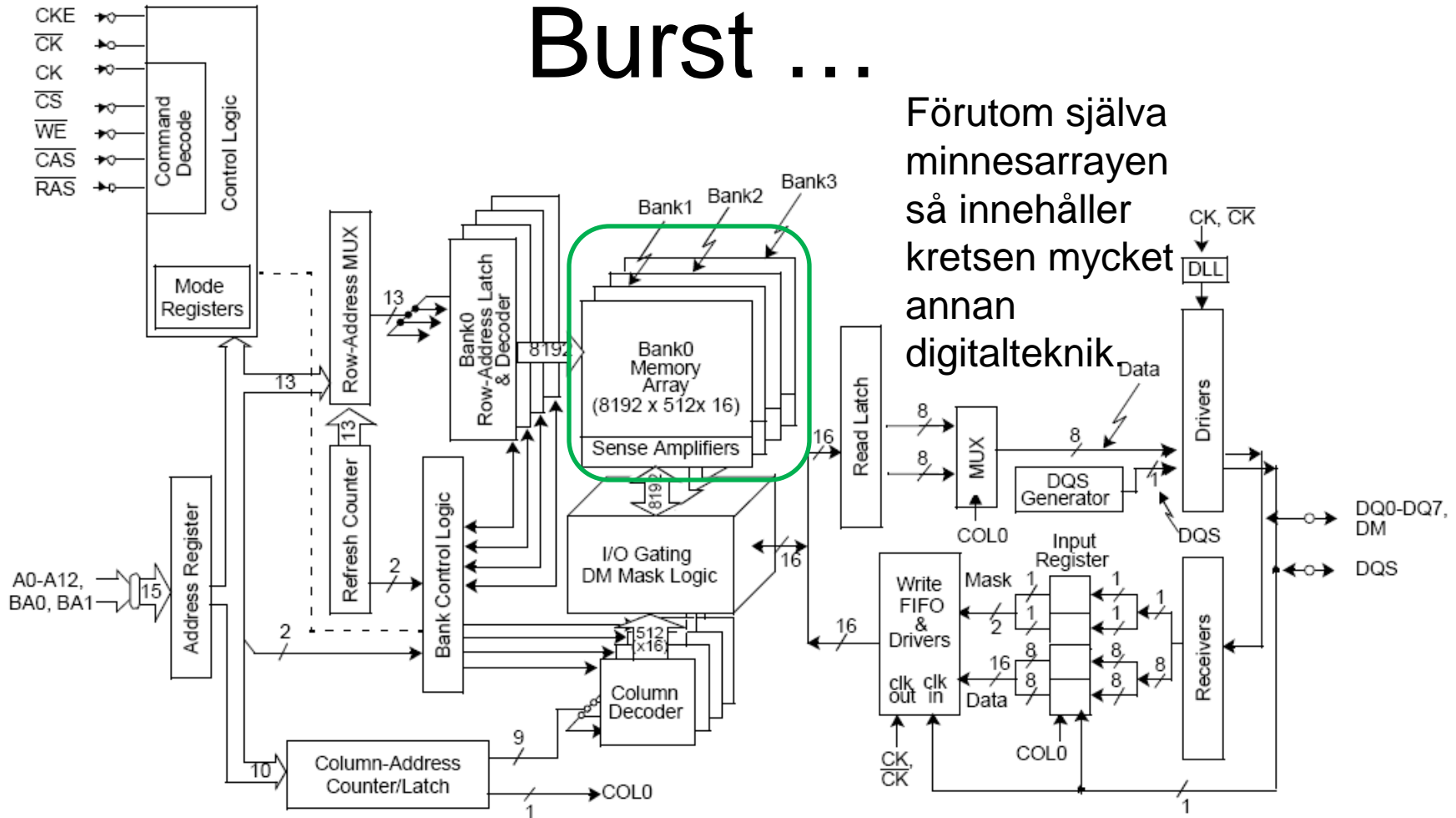
Synkront, använder bussklockan. Dubbel flanktriggat för dubbla datahastigheten $ck+ck$ (även lägre effektförbrukning).

$32M \cdot 2^5 \times 2^{20} = 2^{25}$, 25 adressbitar används. Tidsmultiplexad adressering, 13 bitar RAS (rader), 10 bitar CAS (kolumner), två bankbitar BA0 och BA1.

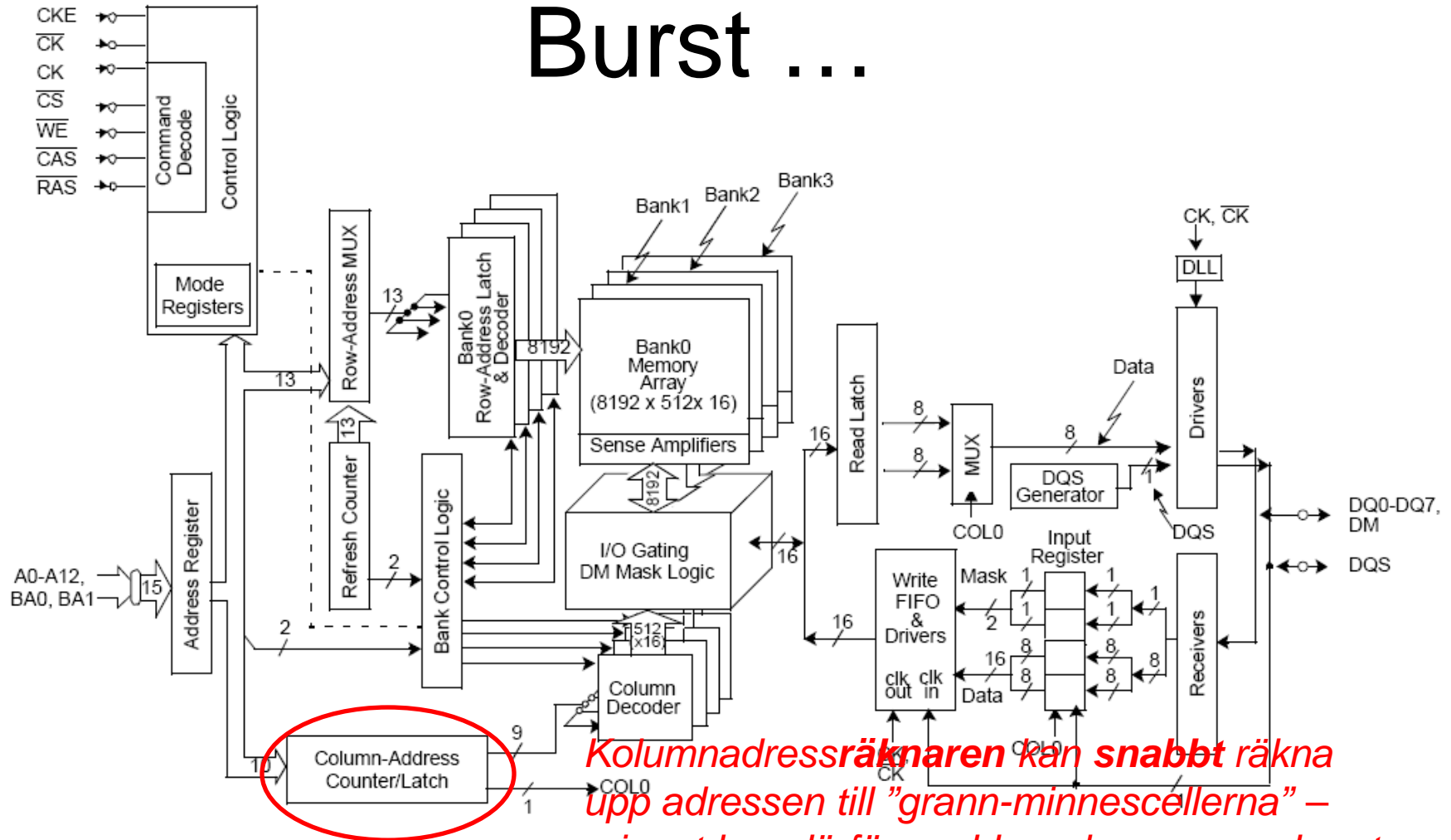
Burst kan vara 2, 4, 8 Byte i följd.

Burst ...

Förutom själva minnesarrayen så innehåller kretsen mycket annan digitalteknik

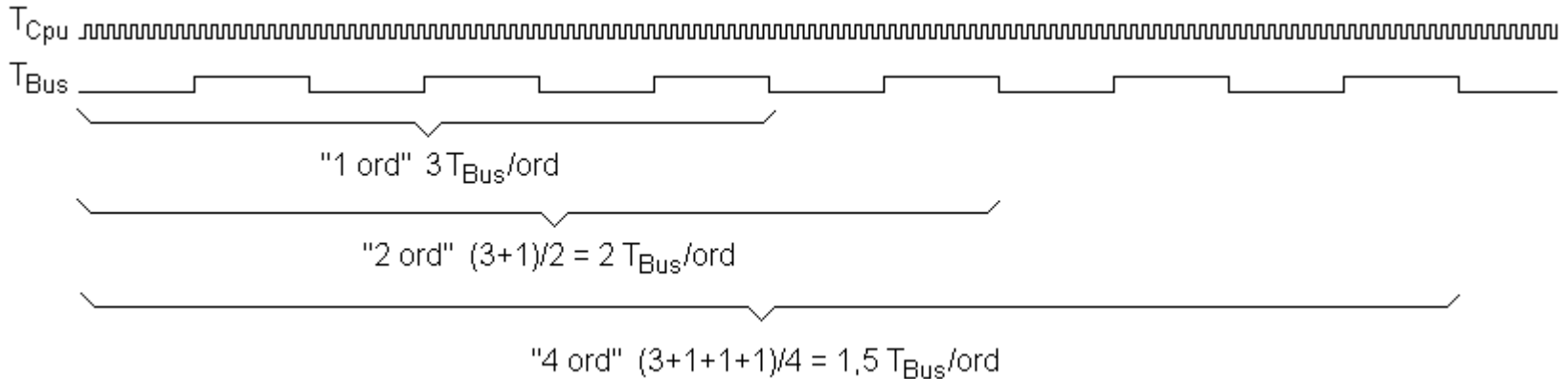


Burst ...



Kolumnadressräknaren kan snabbt räkna upp adressen till "grann-minnescellerna" – minnet kan därför snabbare leverera en burst (skur) med flera Bytes i följd, än en enstaka random Byte.

Burst ger snabbare medelaccess



- Att hämta **1** "random" ord i minnet tar tre busscykler $3T_{\text{Bus}}/\text{ord}$ (varav $2T_{\text{BUS}}$ är Waitstates)
- Att hämta en "Burst" med **2** ord tar $3+1$ busscykler, $4/2 = 2T_{\text{Bus}}/\text{ord}$
- Att hämta en "Burst" med **4** ord tar $3+1+1+1$ busscykler, $6/4 = 1,5T_{\text{Bus}}/\text{ord}$
- Att hämta en "Burst" med **8** ord tar $3+1+1+1+1+1+1+1$ busscykler, $10/8 = 1,25T_{\text{Bus}}/\text{ord}$

Det gäller dock att ha användning för alla hämtade ord – annars slösar man bussklockcykler med Burst-metoden!

Mer om detta i **Datorteknikkursen** i samband med cacheminnen.

ÖH 12.1 Dynamiskt minne



Kapsel
256Mbit (32M×8)

a) Hur många kapslar krävs för 256M×64?

ÖH 12.1 Dynamiskt minne



Kapsel
256Mbit (32M×8)

a) Hur många kapslar krävs för **256M×64**?

Minne $N = 256M$ $M = 64$ bitar. **Kapsel** $p = 32M$ $q = 8$ bitar.

Antal kolumner $k = M/q = 64/8 = 8$.

Antalet rader $r = N/p = 256M/32M = 8$.

Antal kapslar $K = r \times k = 8 \times 8 = \mathbf{64}$.



512M×72 ?



b) Hur många kapslar krävs för **512M×72**?

Kapsel
256Mbit (32M×8)



512M×72 ?



b) Hur många kapslar krävs för 512M×72?

Kapsel
256Mbit (32M×8)

Minne $N = 512\text{M}$ $M = 72$ bitar. **Kapsel** $p = 32\text{M}$ $q = 8$ bitar.

Antal kolumner $k = M/q = 72/8 = 9$.

Antalet rader $r = N/p = 512\text{M}/32\text{M} = 16$.

Antal kapslar $K = r \times k = 9 \times 16 = \mathbf{144}$.



512M×72 ?



b) Hur många kapslar krävs för 512M×72?

Kapsel
256Mbit (32M×8)

Minne $N = 512\text{M}$ $M = 72$ bitar. **Kapsel** $p = 32\text{M}$ $q = 8$ bitar.

Antal kolumner $k = M/q = 72/8 = 9$.

Antalet rader $r = N/p = 512\text{M}/32\text{M} = 16$.

Antal kapslar $K = r \times k = 9 \times 16 = 144$.

Den "ovanliga" bitbredden 72 (= 64 + 8). De 8 *extra* bitarna används för att korrigera enkelfel, och för att kunna upptäcka dubbelfel.

(På så sätt kan även kapslar med något litet fel användas eftersom felet kan korrigeras. De kapslarna skulle annars behöva kasseras).



512M×72 ?



b) Hur många kapslar krävs för 512M×72?

Kapsel
256Mbit (32M×8)

Minne $N = 512\text{M}$ $M = 72$ bitar. **Kapsel** $p = 32\text{M}$ $q = 8$ bitar.

Antal kolumner $k = M/q = 72/8 = 9$.

Antalet rader $r = N/p = 512\text{M}/32\text{M} = 16$.

Antal kapslar $K = r \times k = 9 \times 16 = 144$.

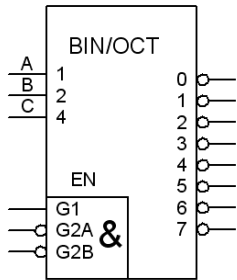
Den "ovanliga" bitbredden 72 (= 64 + 8). De 8 *extra* bitarna används för att korrigera enkelfel, och för att kunna upptäcka dubbelfel.

(På så sätt kan även kapslar med något litet fel användas eftersom felet kan korrigeras. De kapslarna skulle annars behöva kasseras).

Eller så kommer ett bra minne att "tåla" att några av minnescellerna "slits ut" med tiden.

William Sandqvist william@kth.se

ÖH 12.2 ROM och SRAM



Läsminne:



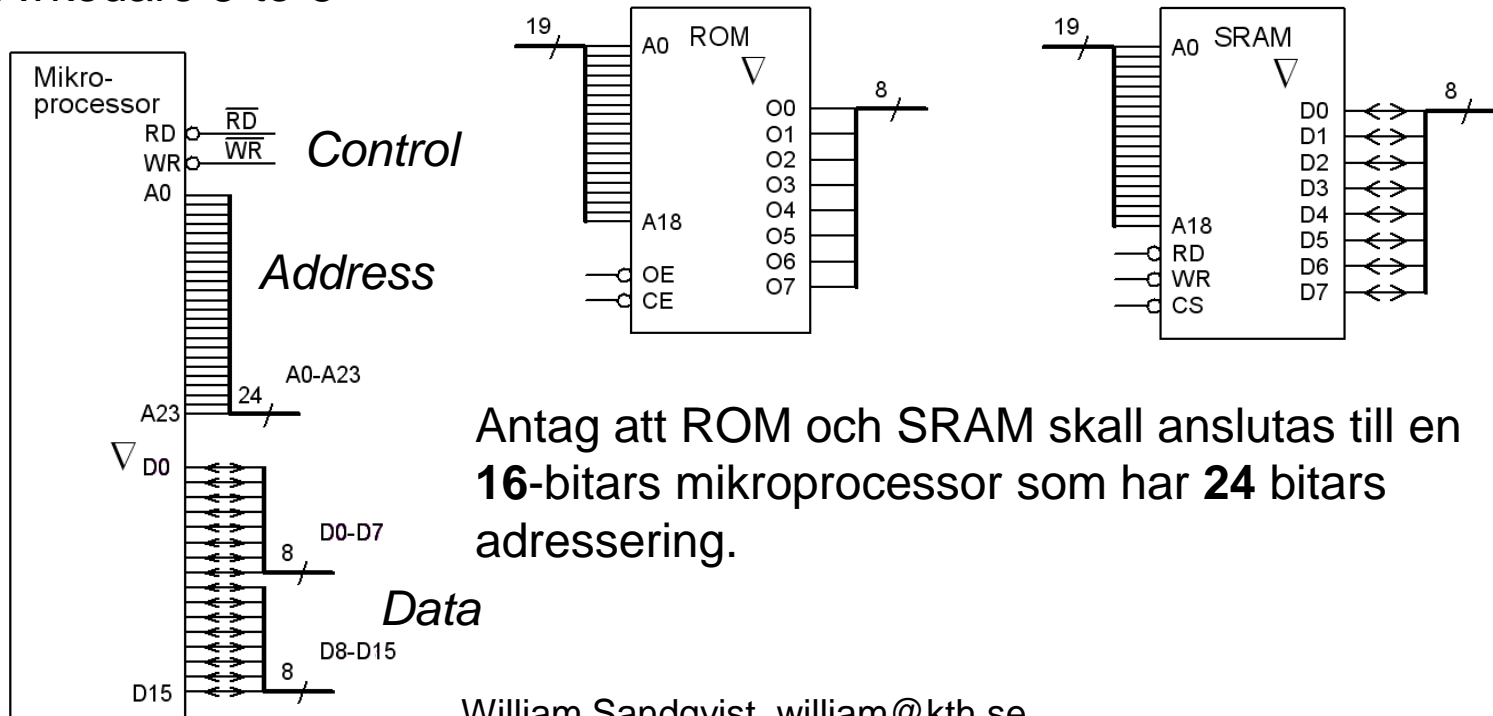
ROM 4M 512k × 8 bit

Läs/Skrivminne:



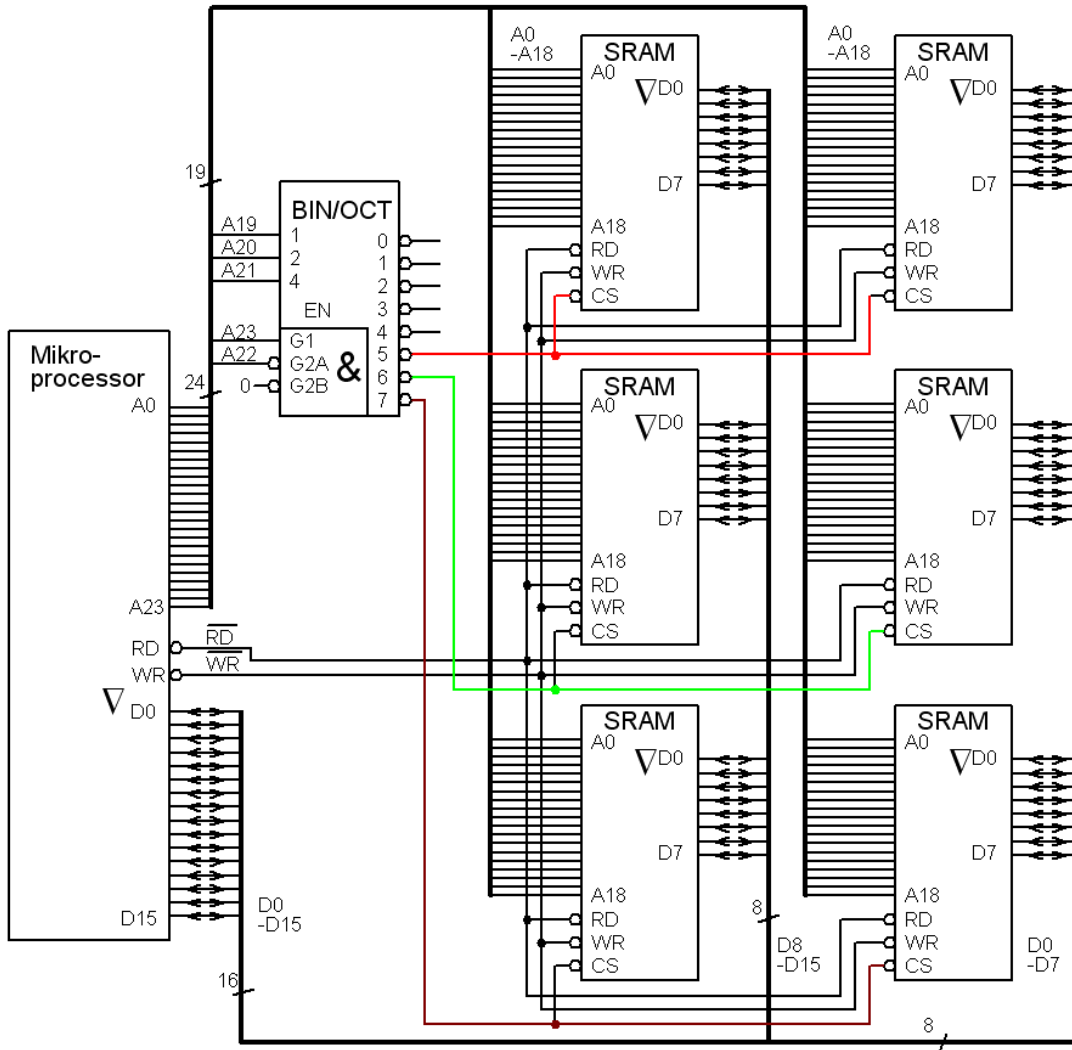
SRAM 4M 512k × 8 bit

Avkodare 3-to-8



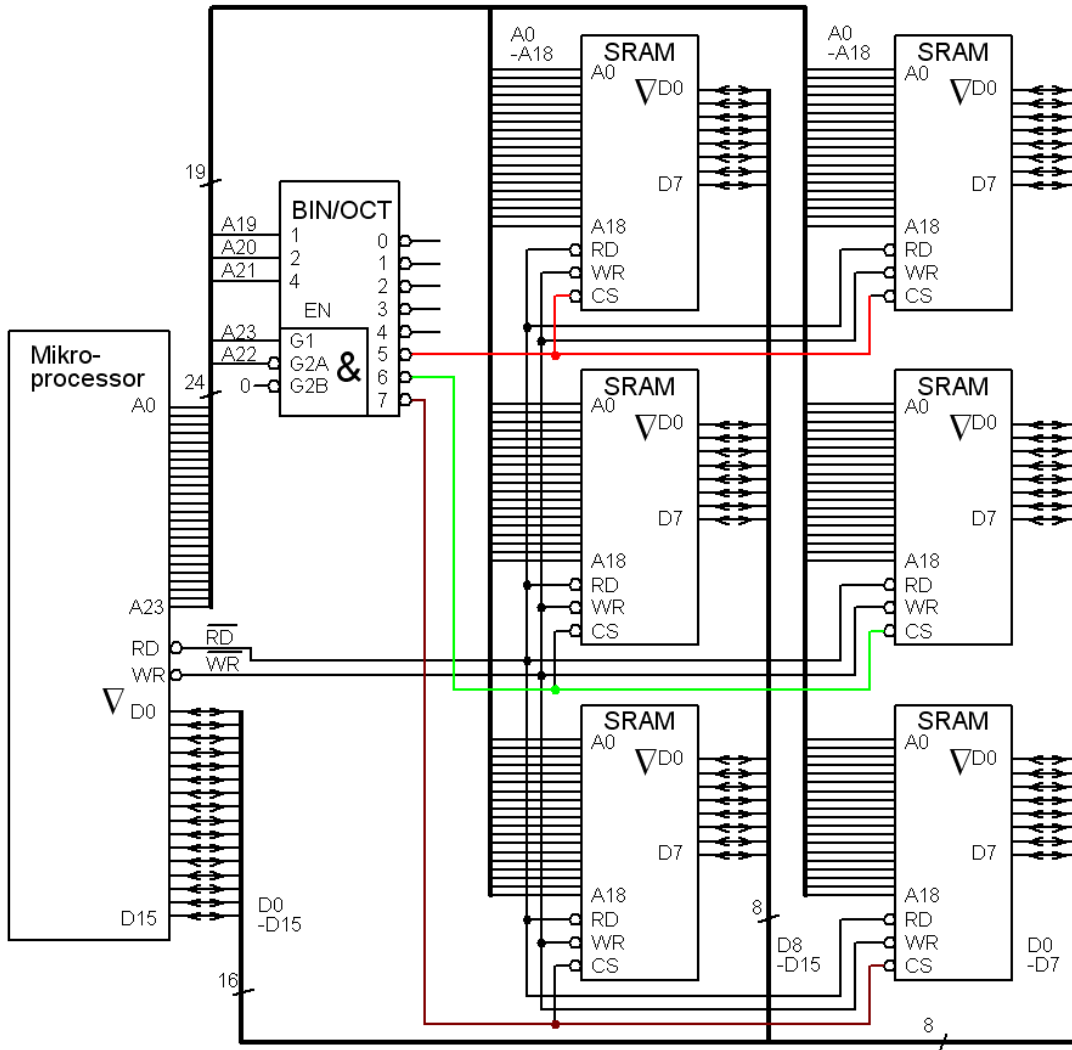
Antag att ROM och SRAM skall anslutas till en **16-bitars** mikroprocessor som har **24 bitars** adressering.

SRAM storlek?



Hur stort är figurens SRAM, och vilket är adress-området uttryckt i hexadecimala siffror?

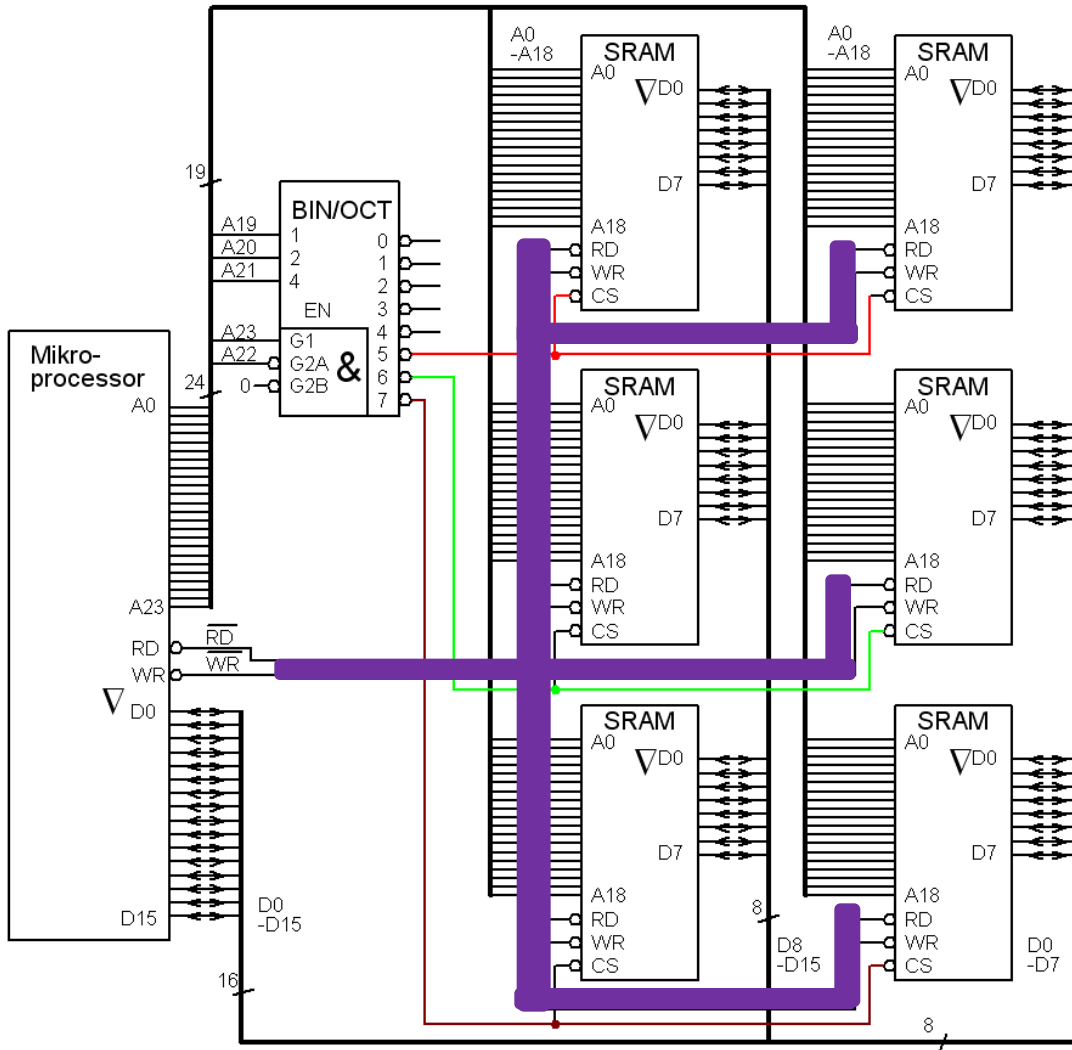
SRAM storlek?



Hur stort är figurens SRAM, och vilket är adress-området uttryckt i hexadecimala siffror?

- **Minneskapsel:**
 $p = 512k$ $q = 8$ bitar
- **Minne:**
 $r = 3$ $k = 2$ $K = 2 \times 3 = 6$
 $M = k \times q = 2 \times 8 = 16$ bitar
 $N = p \times r = 512k \times 3 = \mathbf{1,5M}$

SRAM Control?



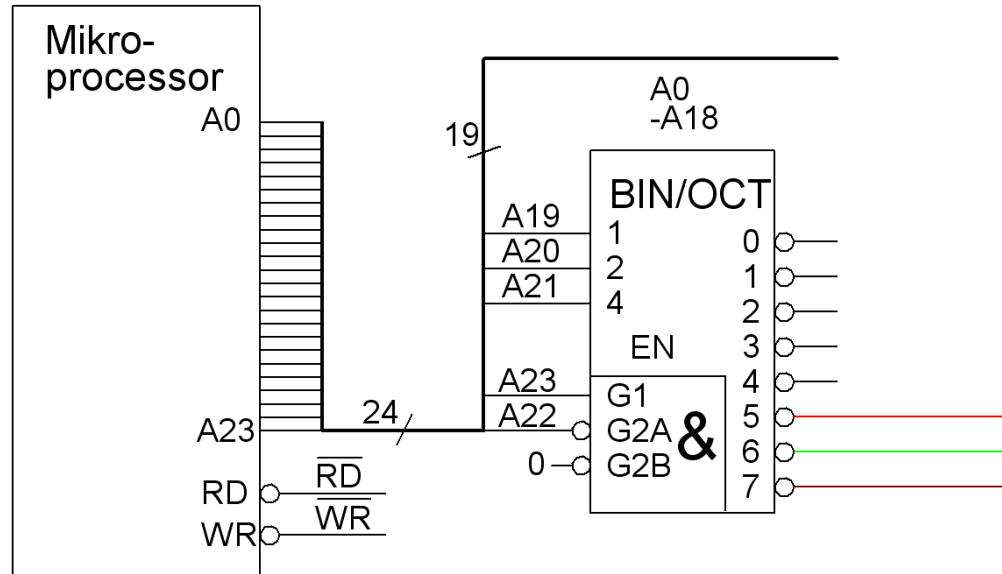
Hur stort är figurens SRAM, och vilket är adress-området uttryckt i hexadecimala siffror?

- **Minneskapsel:**
 $p = 512k$ $q = 8$ bitar
- **Minne:**
 $r = 3$ $k = 2$ $K = 2 \times 3 = 6$
 $M = k \times q = 2 \times 8 = 16$ bitar
 $N = p \times r = 512k \times 3 = 1,5M$

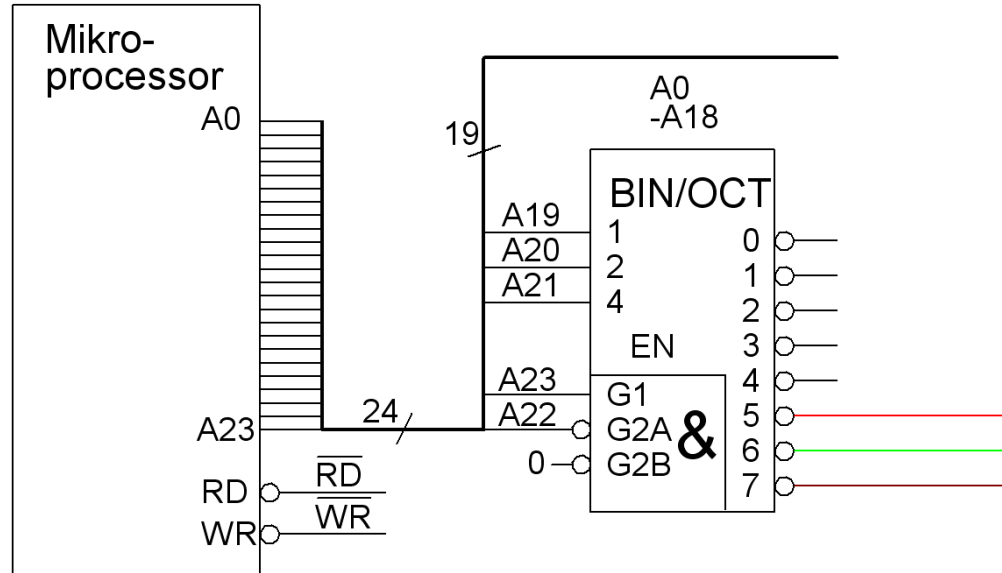
$$\overline{RD} = \overline{RD}$$

$$\overline{WR} = \overline{WR}$$

SRAM adressområde?

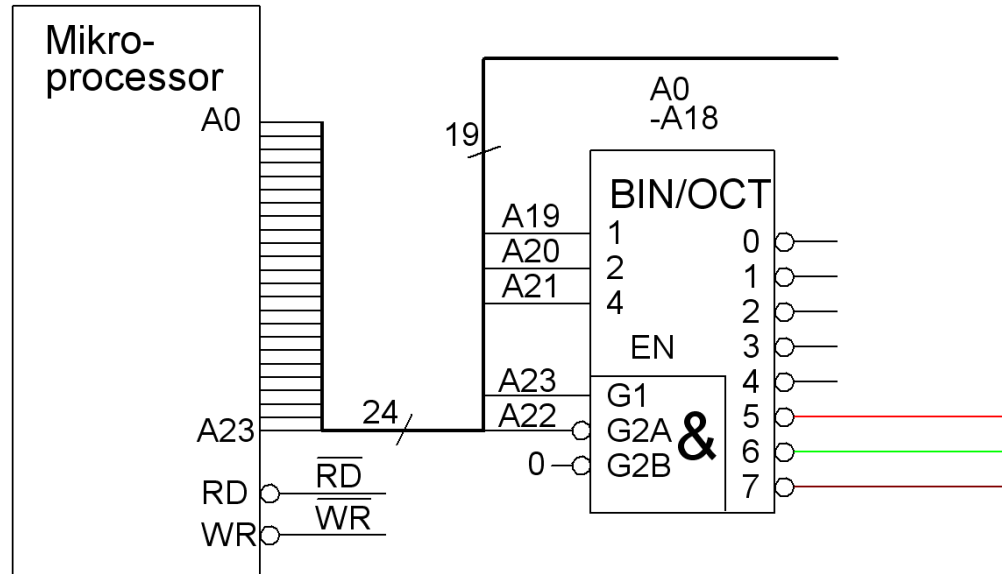


SRAM adressområde?



Computer:	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Decoder:	1	0	0 ... 7																						
Mem start:	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
• Begin hex	A			8				0				0				0				0					
Mem end:	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
• End hex	B			F				F				F				F				F					

SRAM adressområde?



Computer:	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Decoder:	1	0	0 ... 7																						
Mem start:	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
• Begin hex	A			8				0				0				0				0					
Mem end:	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
• End hex	B			F				F				F				F				F					

SRAM adressområde: A80000 - BFFFFFF

Byt adressområde! ?

Ändra till adressområde 980000 – AFFFFFF ?

980000

1001|1000|0000|0000|0000|0000|

AFFFFFF

1010|1111|1111|1111|1111|1111|

Byt adressområde! ?

Ändra till adressområde 980000 – AFFFFFF ?

980000

1001|1000|0000|0000|0000|0000|

AFFFFFF

1010|1111|1111|1111|1111|1111|

Byt adressområde! ?

Ändra till adressområde 980000 – AFFFFFF ?

980000

1001|1000|0000|0000|0000|0000|

AFFFFFF

1010|1111|1111|1111|1111|

"10|011" → "3"

"10|101" → "5"

Byt adressområde! ?

Ändra till adressområde 980000 – AFFFFFF ?

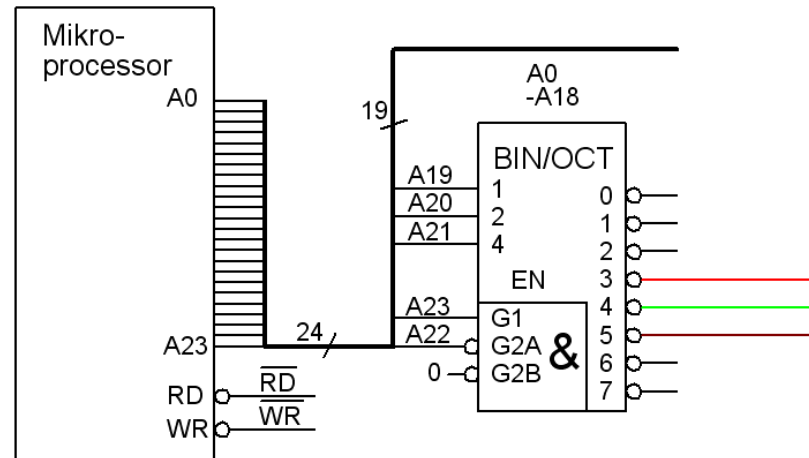
980000

1001|1000|0000|0000|0000|0000|

AFFFFFF

1010|1111|1111|1111|1111|1111|

"10|011" → "3"
"10|101" → "5"



Byt adressområde! ?

Ändra till adressområde 480000 – 5FFFFFF ?

Byt adressområde! ?

Ändra till adressområde 480000 – 5FFFFFF ?

480000

01001000|0000|0000|0000|0000|

5FFFFFF

0101111|1111|1111|1111|1111|

Byt adressområde! ?

Ändra till adressområde 480000 – 5FFFFFF ?

480000

0100|1000|0000|0000|0000|0000|

5FFFFFF

0101|1111|1111|1111|1111|1111|

"01|001" → "1"

"01|011" → "3"

Byt adressområde! ?

Ändra till adressområde 480000 – 5FFFFFF ?

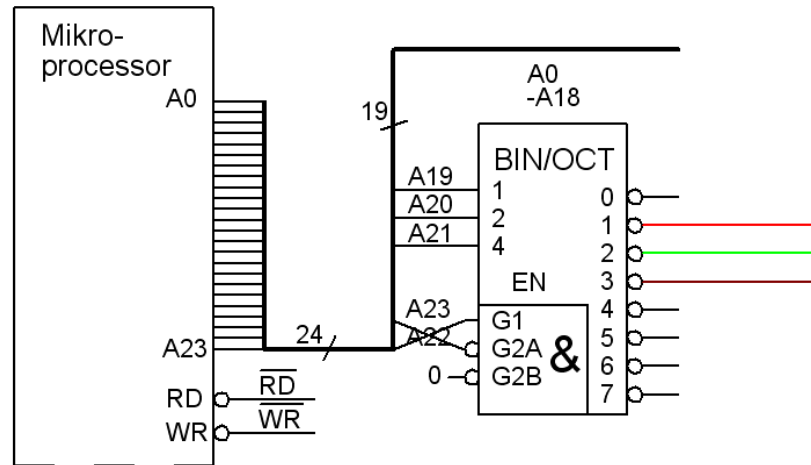
480000

0100|1000|0000|0000|0000|0000|

5FFFFFF

0101|1111|1111|1111|1111|1111|

"01|001" → "1"
"01|011" → "3"



Byt adressområde! ?

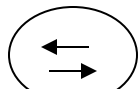
Ändra till adressområde 480000 – 5FFFFFF ?

480000

0100|1000|0000|0000|0000|0000|

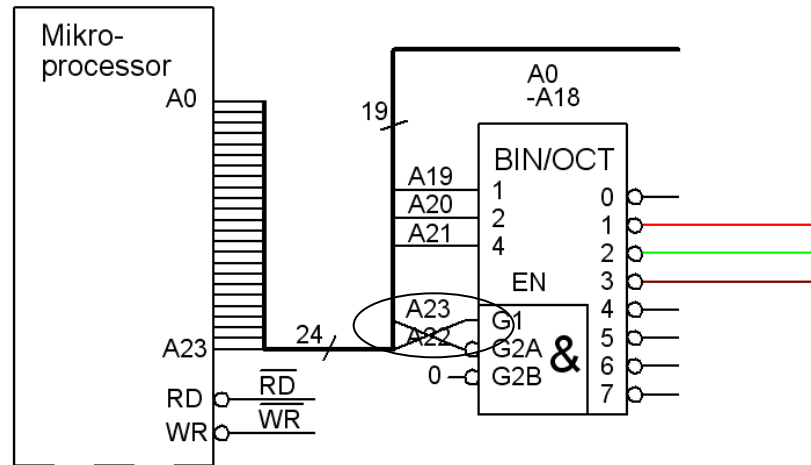
5FFFFFF

0101|1111|1111|1111|1111|1111|



"01|001" → "1"

"01|011" → "3"



ROM 00 00 00...?

Oftast läser en processor sin första instruktion från adress 0. Då måste det finnas ett läsminne på den adressen.

Antag ett ROM-minne **2M** × **16** bitar adressområde 000000 ... och framåt.
ROM Chip 512k×8.

- Hur många kapslar behövs?
- Hur skall avkodaren anslutas?
- Hur skall minneskretsarna anslutas?
- Ange adressområdena för avkodarens utgångar med hexadecimala siffror.

ROM 00 00 00...?

Oftast läser en processor sin första instruktion från adress 0. Då måste det finnas ett läsminne på den adressen.

Antag ett ROM-minne $2M \times 16$ bitar adressområde 000000 ... och framåt.
ROM Chip 512k \times 8.

- Hur många kapslar behövs?
- Hur skall avkodaren anslutas?
- Hur skall minneskretsarna anslutas?
- Ange adressområdena för avkodarens utgångar med hexadecimala siffror.

Minne:

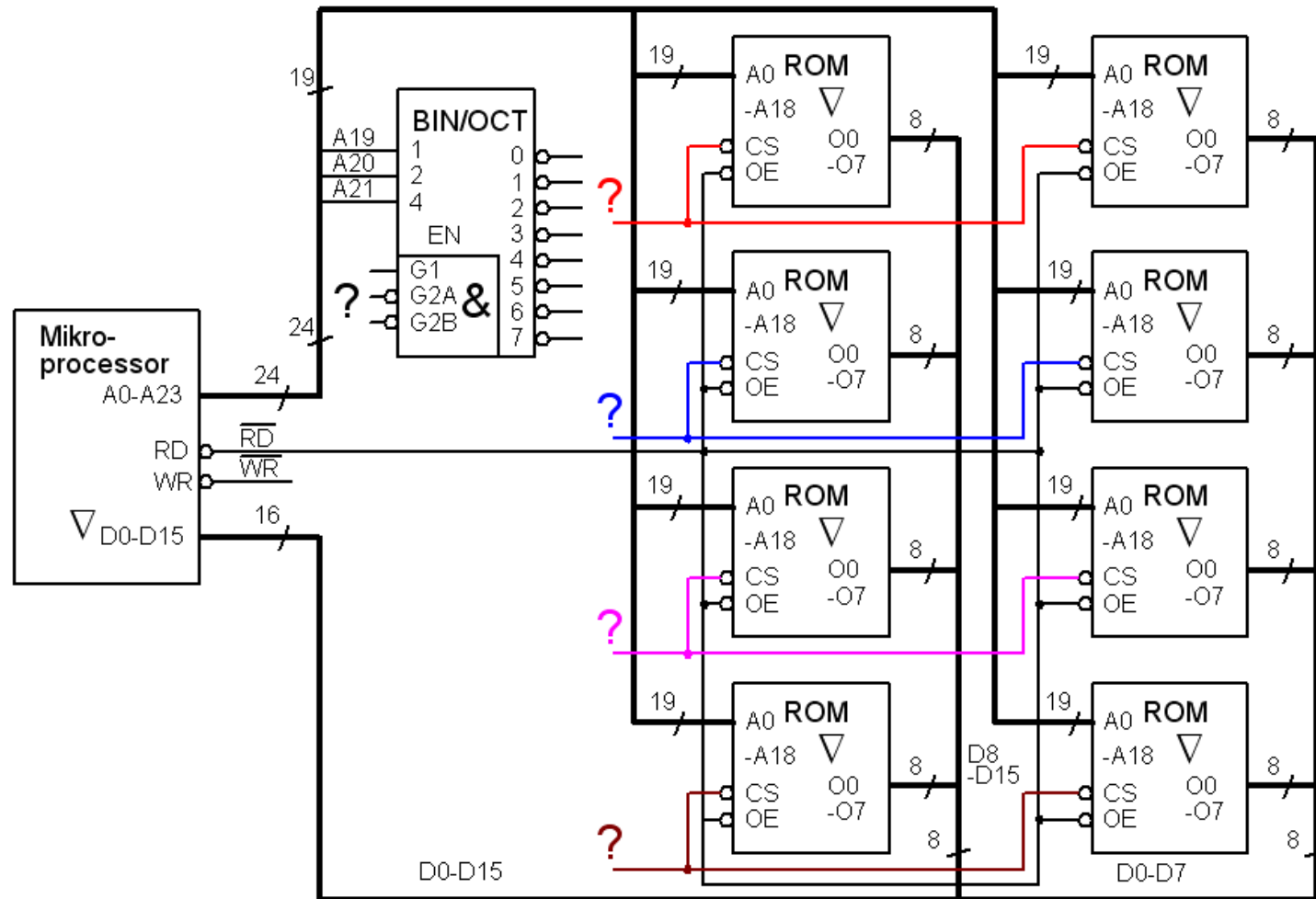
$N = 2 M$ (4·512k) ordlängden $M = 16$ bitar

Minneskapsel:

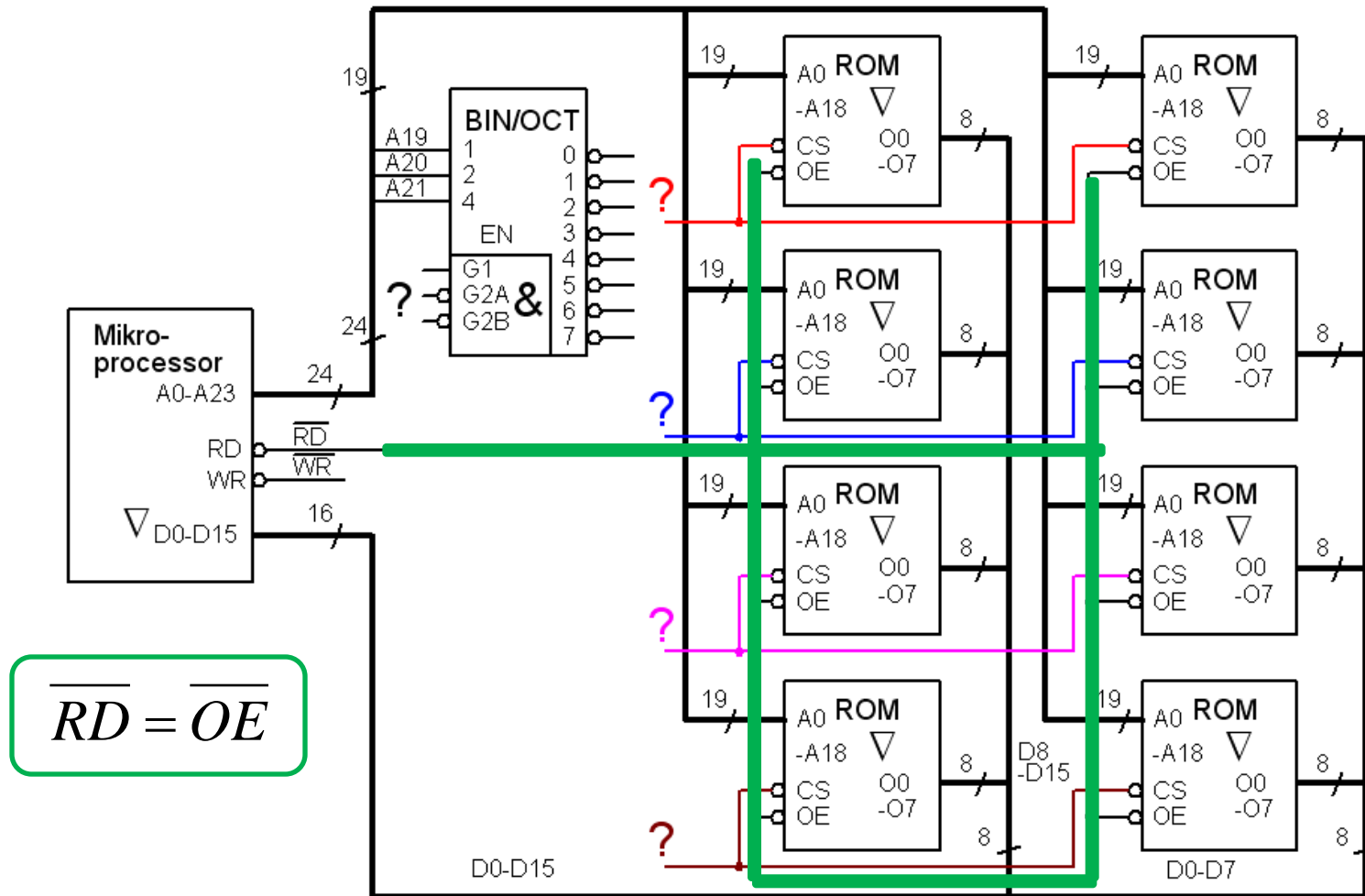
$p = 512$ k ordlängden $q = 8$ bitar

- Antalet kapselrader $r \leq N/p = 4 \cdot 512k / 512k = 4$
- Antalet kapselkolumner $k \geq M/q = 16/8 = 2$
- Antalet kapslar $K = r \times k = 4 \times 2 = \mathbf{8}$

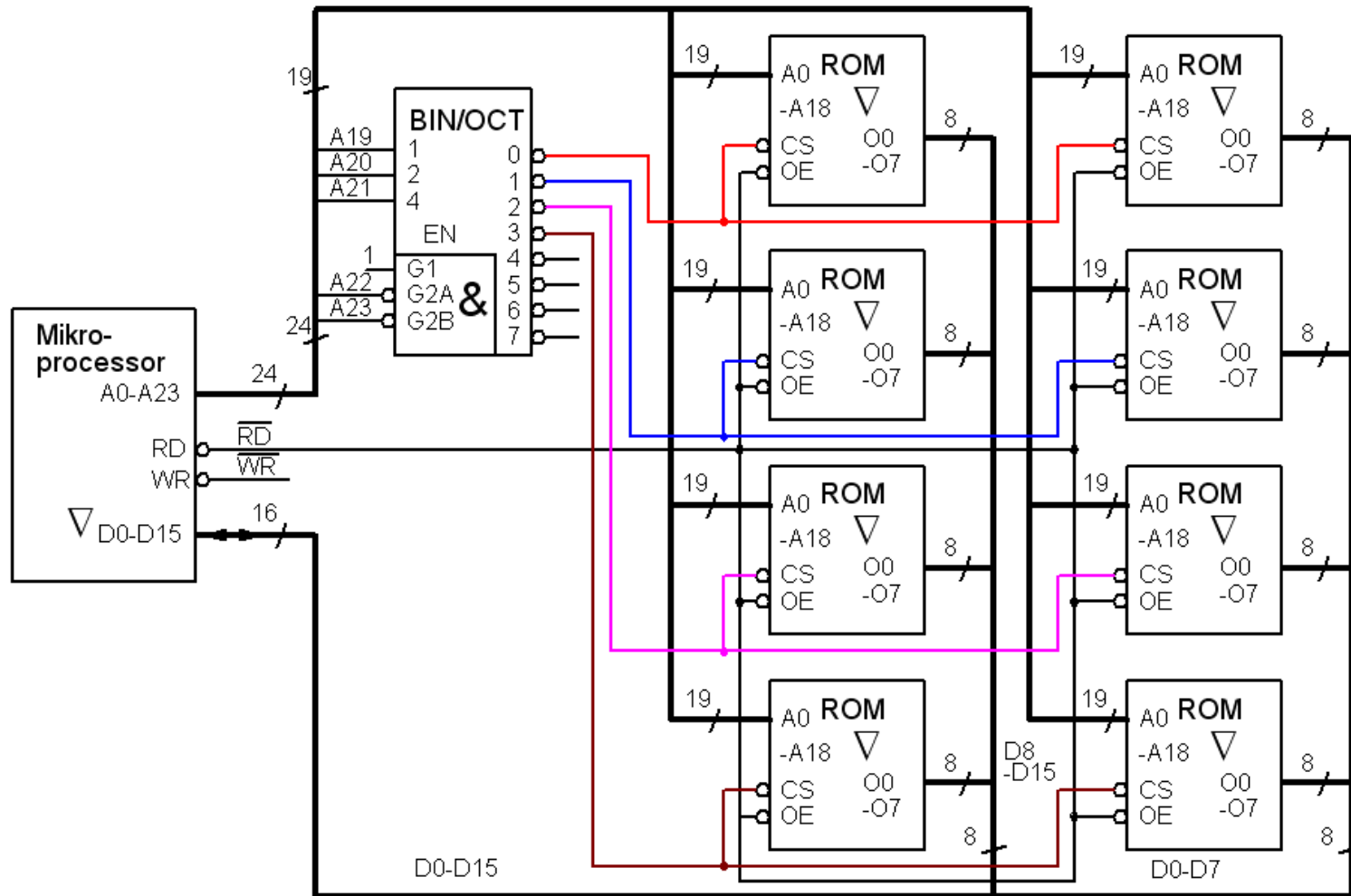
ROM anslutning?



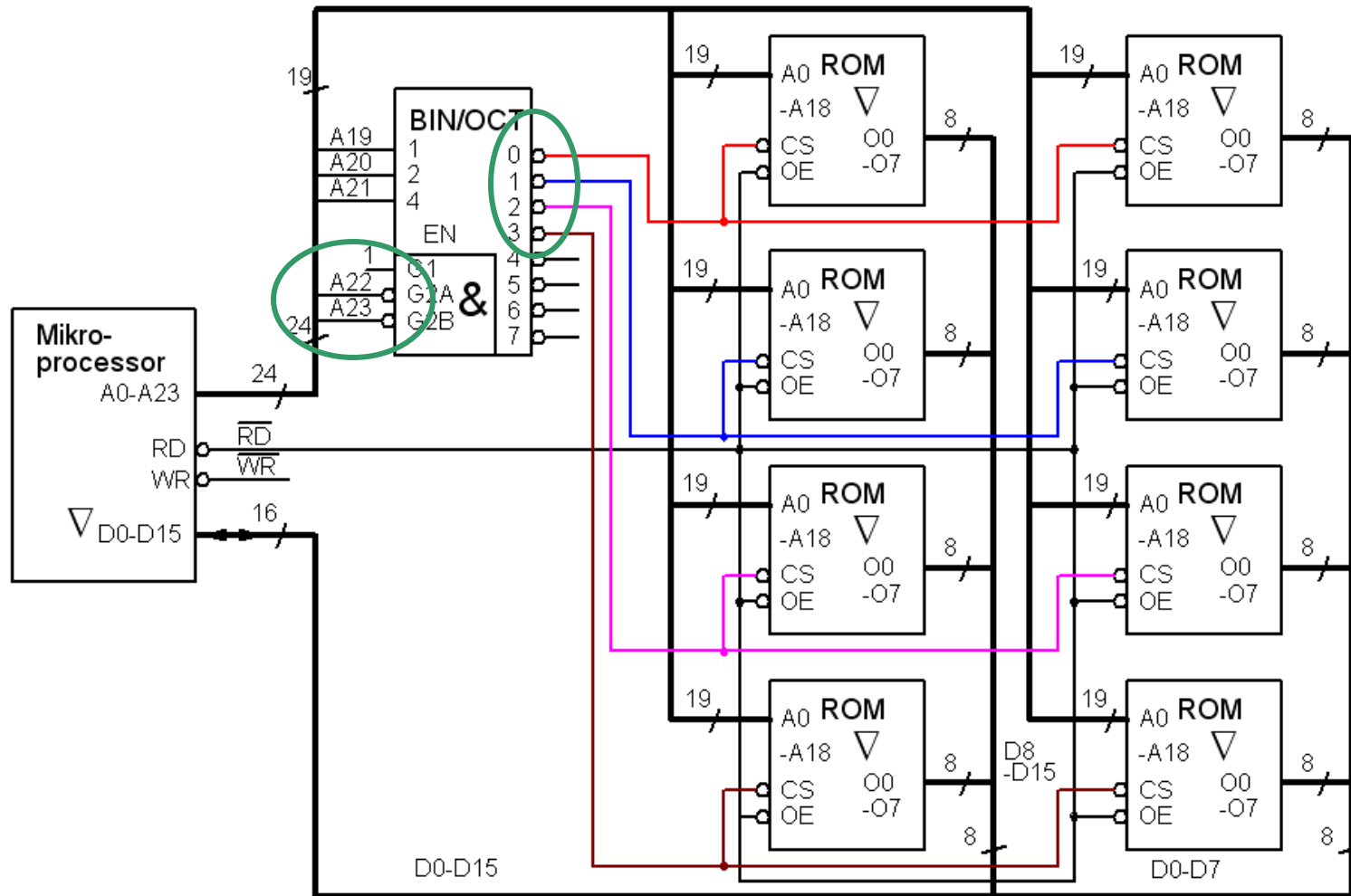
ROM Control anslutning?



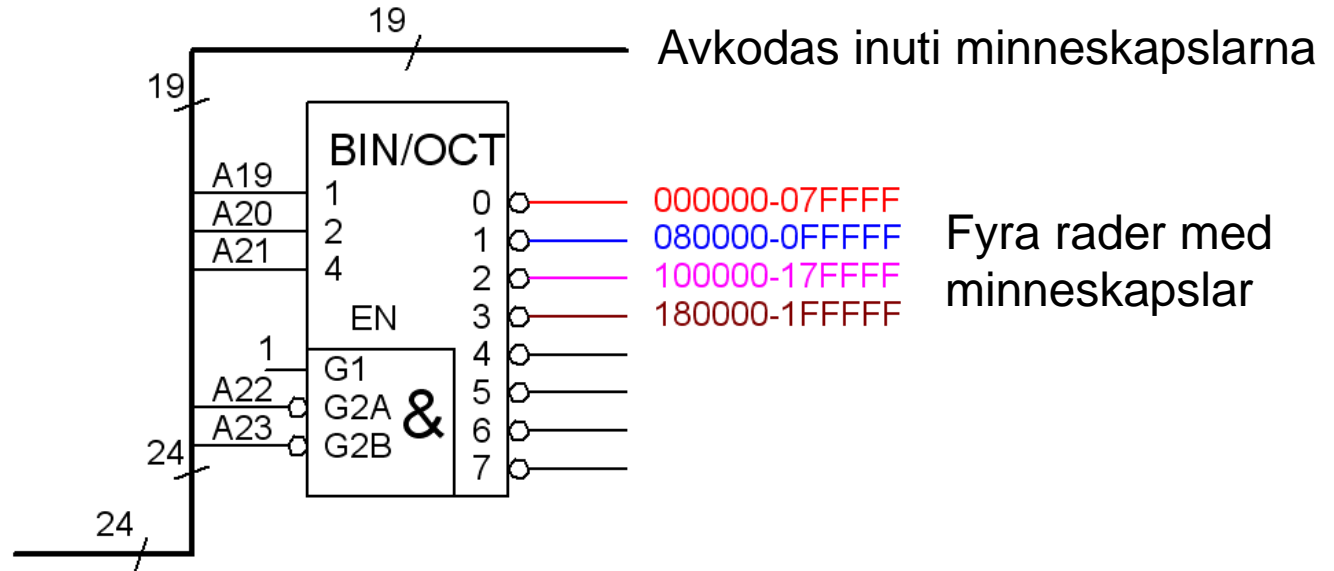
Decoder ROM anslutning?



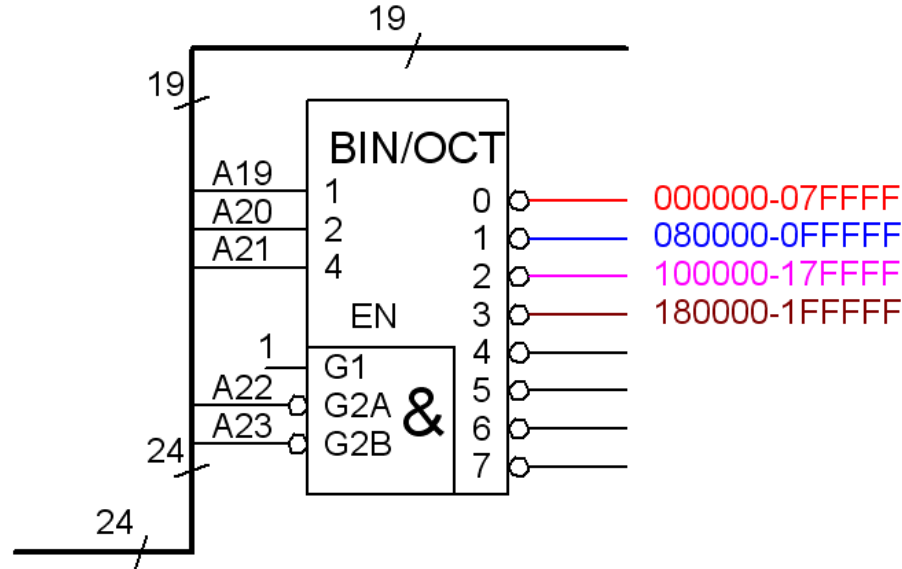
Decoder ROM anslutning?



Decoder ROM adresser?

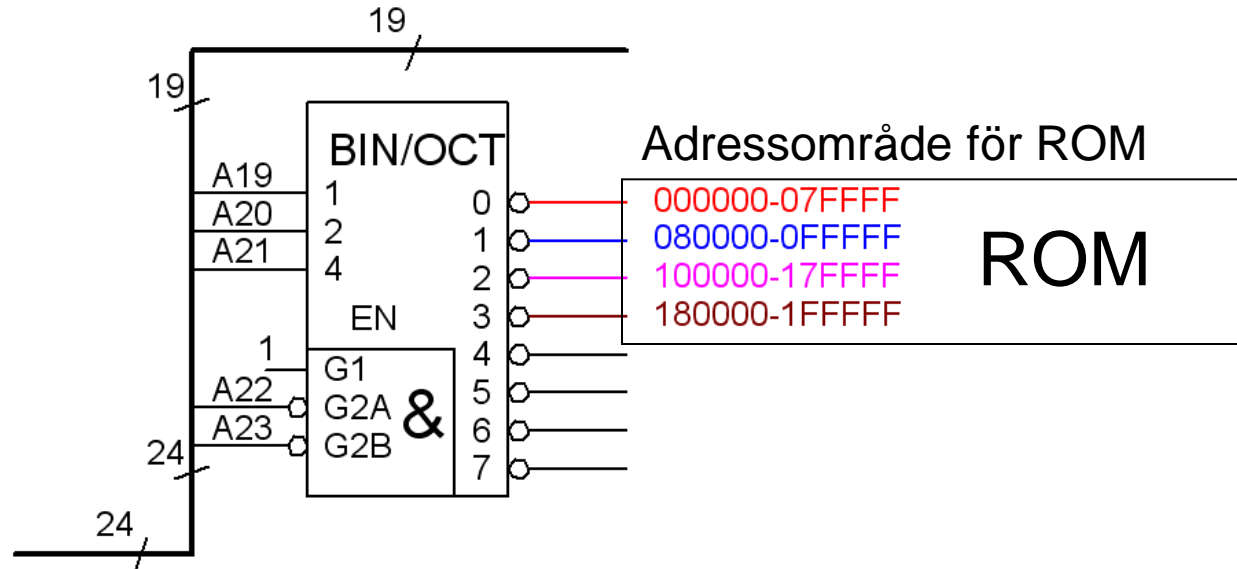


Decoder ROM addresser?



00ab|cmmm|mmmm|mmmm|mmmm|mmmm
 0000|0000|0|0|0|0 - 0000|0111|F|F|F|F 000000-07FFFF
 0000|1000|0|0|0|0 - 0000|1111|F|F|F|F 080000-0FFFFFFF
 0001|0000|0|0|0|0 - 0001|0111|F|F|F|F 100000-17FFFF
 0001|1000|0|0|0|0 - 0001|1111|F|F|F|F 180000-1FFFFFFF

Decoder ROM adresser?



00ab|cmmm|mmmm|mmmm|mmmm|mmmm

0000|0000|0|0|0|0 - 0000|0111|F|F|F|F 000000-07FFFF

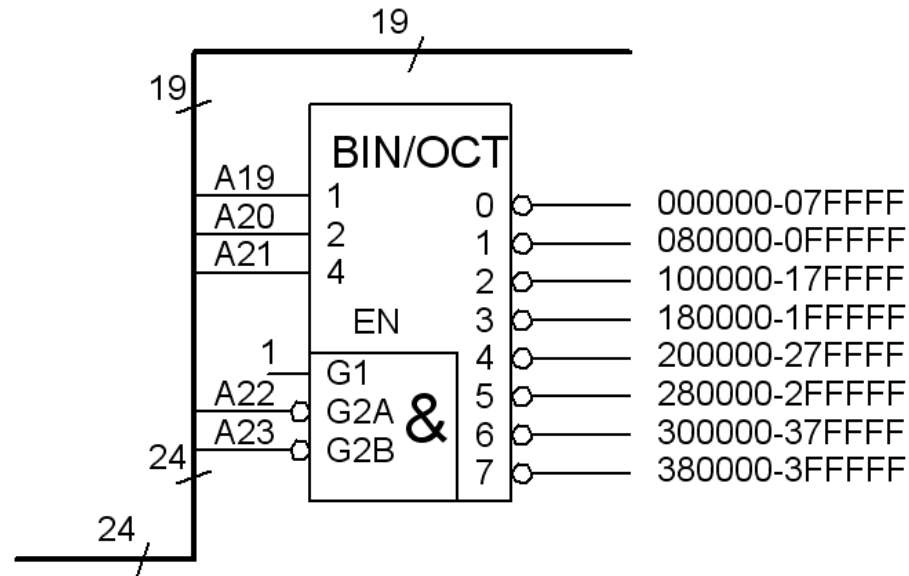
0000|1000|0|0|0|0 - 0000|1111|F|F|F|F 080000-0FFFFF

0001|0000|0|0|0|0 - 0001|0111|F|F|F|F 100000-17FFFF

0001|1000|0|0|0|0 - 0001|1111|F|F|F|F 180000-1FFFFF

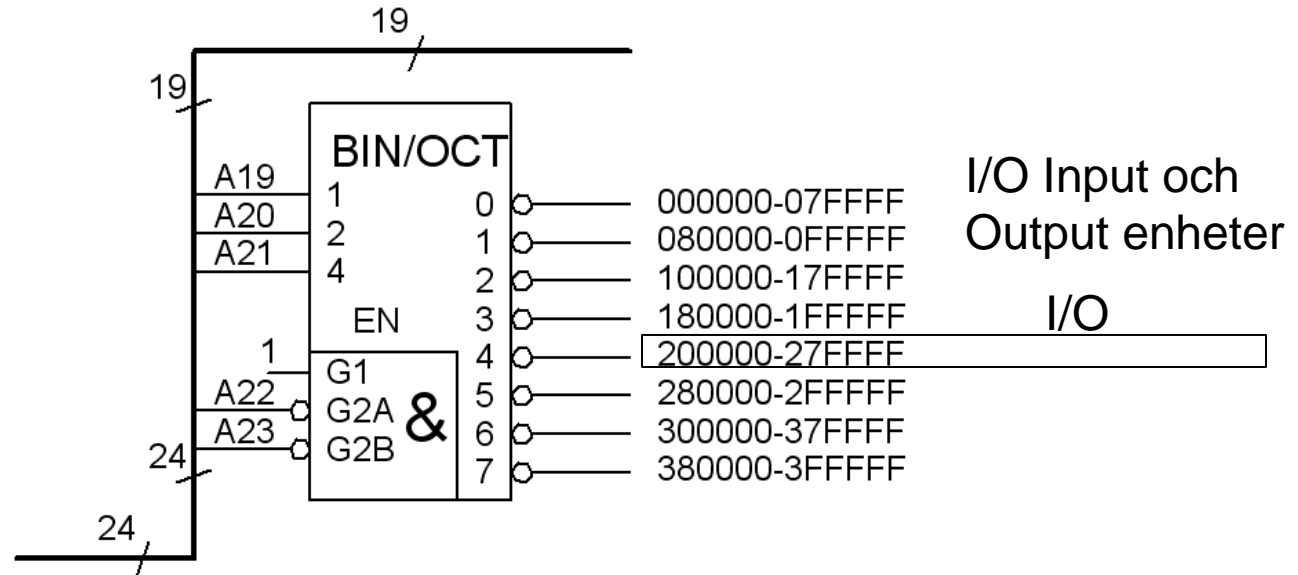
Totalt ROM 000000 – 1FFFFF

Decoder SRAM+I/O addresser?



00ab|cmmm|mmmm|mmmm|mmmm|mmmm

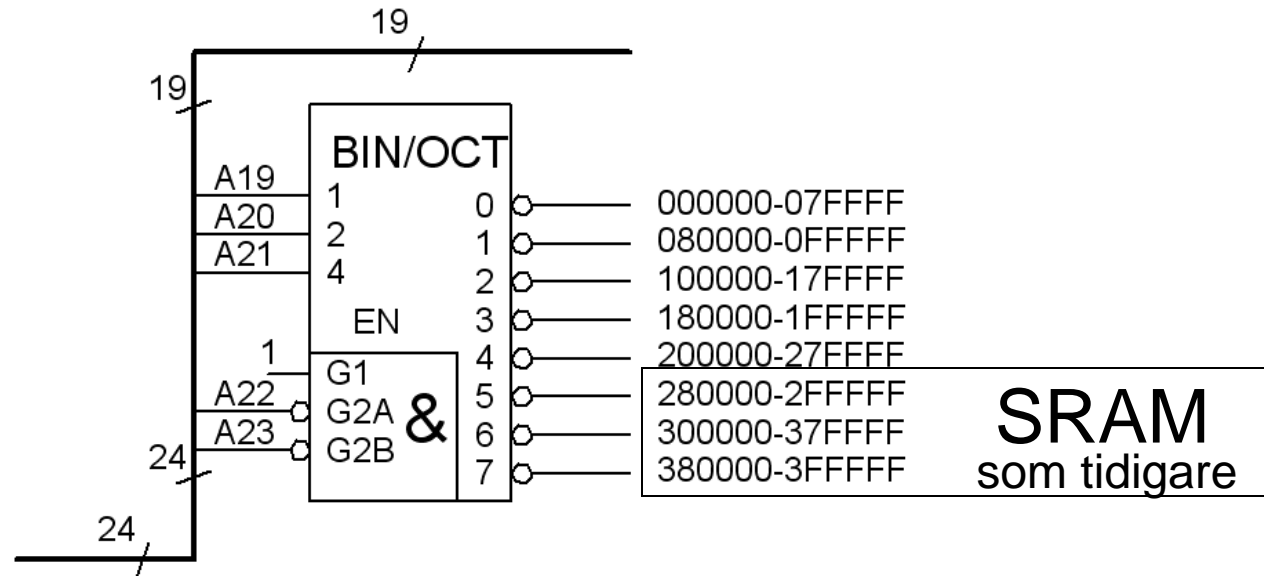
Decoder SRAM+I/O adresser?



00ab|cmmm|mmmm|mmmm|mmmm|mmmm

0010|0000|0|0|0|0 - 0010|0111|F|F|F|F 200000-27FFFF

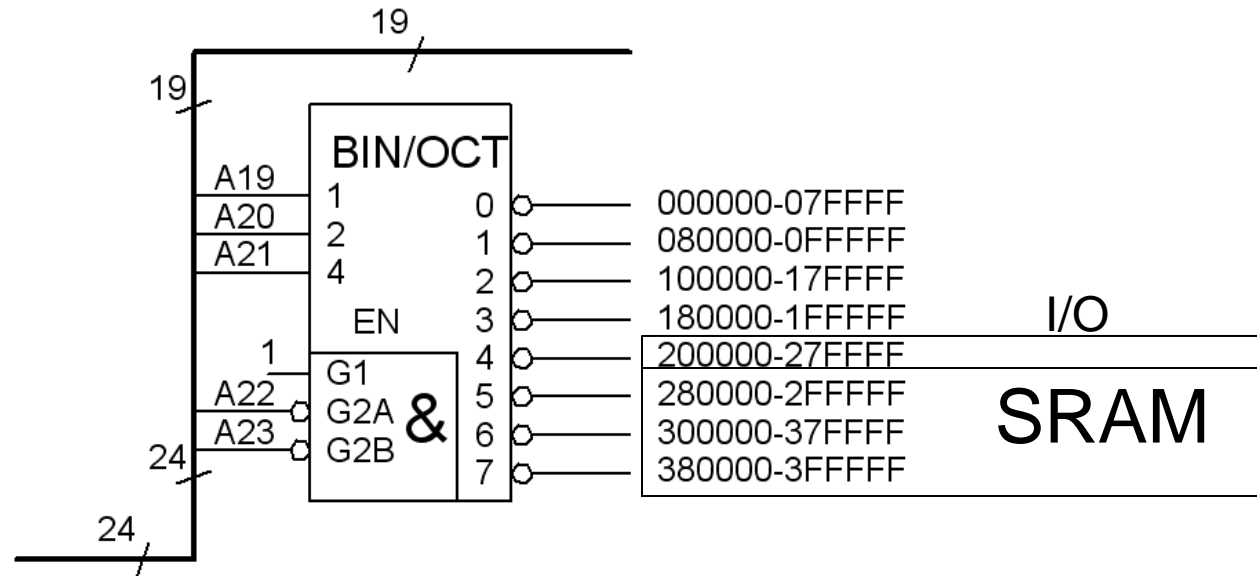
Decoder SRAM+I/O adresser?



00ab|cmmm|mmmm|mmmm|mmmm|mmmm

00**10**|1000|0|0|0|0 - 00**10**|1111|F|F|F|F 280000-2FFFFF
 00**11**|0000|0|0|0|0 - 00**11**|0111|F|F|F|F 300000-37FFFF
 00**11**|1000|0|0|0|0 - 00**11**|1111|F|F|F|F 380000-3FFFFF

Decoder SRAM+I/O adresser?



00ab|cmmm|mmmm|mddd|mddd|mddd

0010|0000|0|0|0|0 - 0010|0111|F|F|F|F 200000-27FFFF

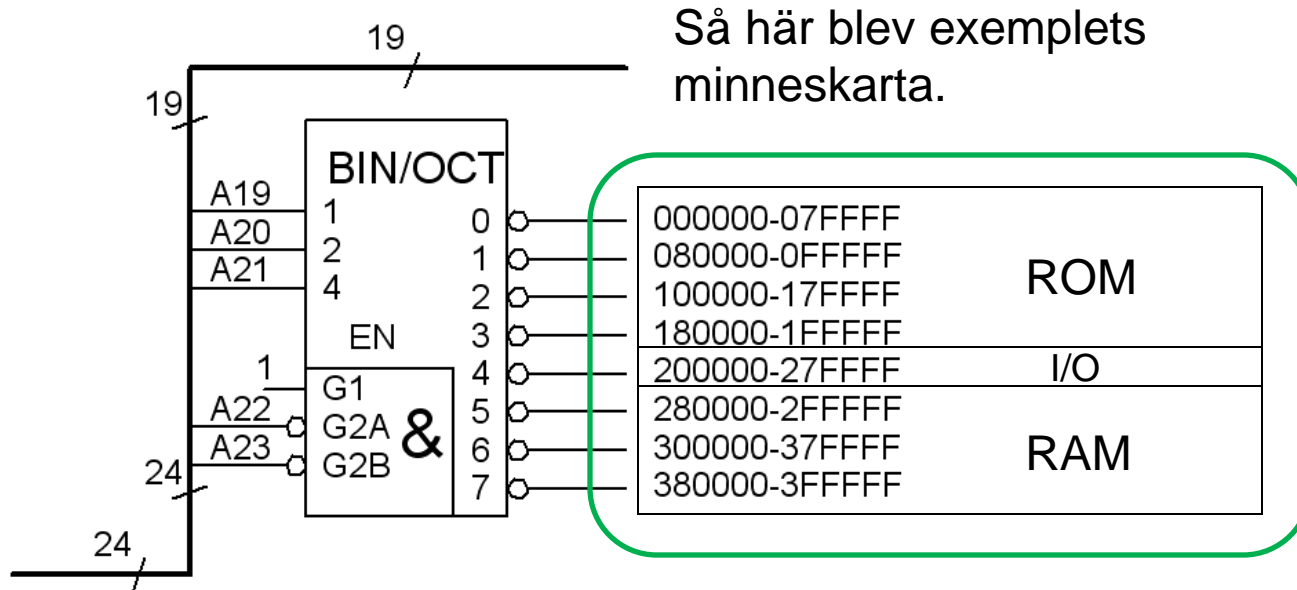
0010|1000|0|0|0|0 - 0010|1111|F|F|F|F 280000-2FFFFFFF

0011|0000|0|0|0|0 - 0011|0111|F|F|F|F 300000-37FFFF

0011|1000|0|0|0|0 - 0011|1111|F|F|F|F 380000-3FFFFFFF

- Möjliga SRAM+I/O adresser 200000 – 3FFFFFFF

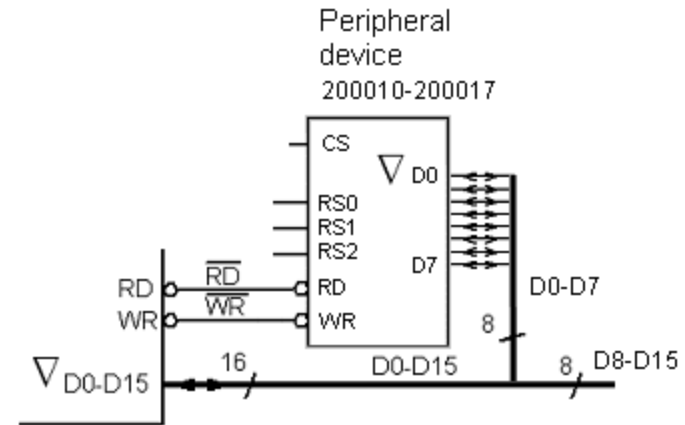
Minneskarta (Memory map)





ÖH 12.3 Input/Output

Periferienheter, I/O, ansluts ofta till en CPU som om dom vore minneskretsar (fast med bara ett fåtal "minnesceller"). Ex. en realtidsklock-krets – håller reda på tid och datum. Den styrs/avläses från 8 inbyggda register.

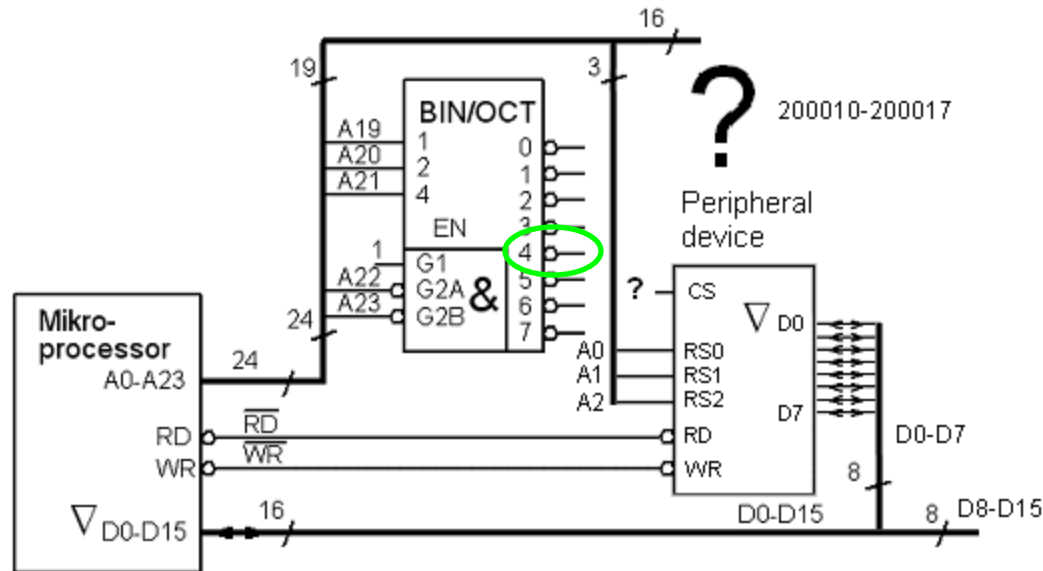


Periferikretsen kopplas in som ett litet RAM-minne. Bara de 8 minst signifikanta databitarna används. CS Chip Select enablar chippet.

Anslut en 8 registers minnesmappad periferienhet (I/O) till en CPU. CPU:n har 16 bitars databuss (använd bara 8), och en 24 bitars adressbuss. Använd en 3:8-avkodare och vid behov grindar. Periferienheten skall kopplas in så att den får registeradresserna **0x200010 ... 0x200017**.



ÖH 12.3 Input/Output

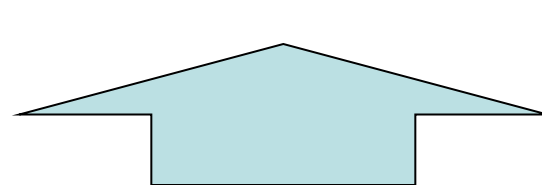


I/O adresser, på avkodarens utgång "4", 200000 – 27FFFF enligt tidigare uppgift.

Avkodningen

0x200010	=	0010		0.000		0000		0000		0001		0.000
0x200011	=	0010		0.000		0000		0000		0001		0.001
0x200012	=	0010		0.000		0000		0000		0001		0.010
0x200013	=	0010		0.000		0000		0000		0001		0.011
0x200014	=	0010		0.000		0000		0000		0001		0.100
0x200015	=	0010		0.000		0000		0000		0001		0.101
0x200016	=	0010		0.000		0000		0000		0001		0.110
0x200017	=	0010		0.000		0000		0000		0001		0.111

↑
Avkodarens utgång "4"



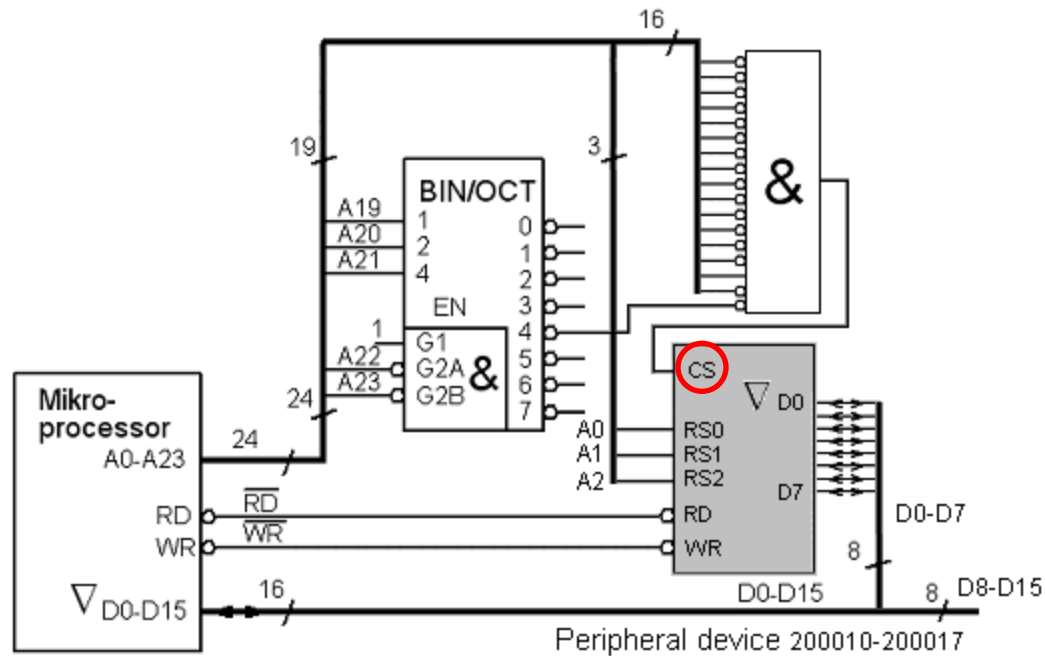
↑
RS₂RS₁RS₀

Kvar att avkoda:

$$\bar{A}_{18} \cdot \bar{A}_{17} \cdot \bar{A}_{16} \cdot \bar{A}_{15} \cdot \bar{A}_{14} \cdot \bar{A}_{13} \cdot \bar{A}_{12} \cdot \bar{A}_{11} \cdot \bar{A}_{10} \cdot \bar{A}_9 \cdot \bar{A}_8 \cdot \bar{A}_7 \cdot \bar{A}_6 \cdot \bar{A}_5 \cdot A_4 \cdot \bar{A}_3$$

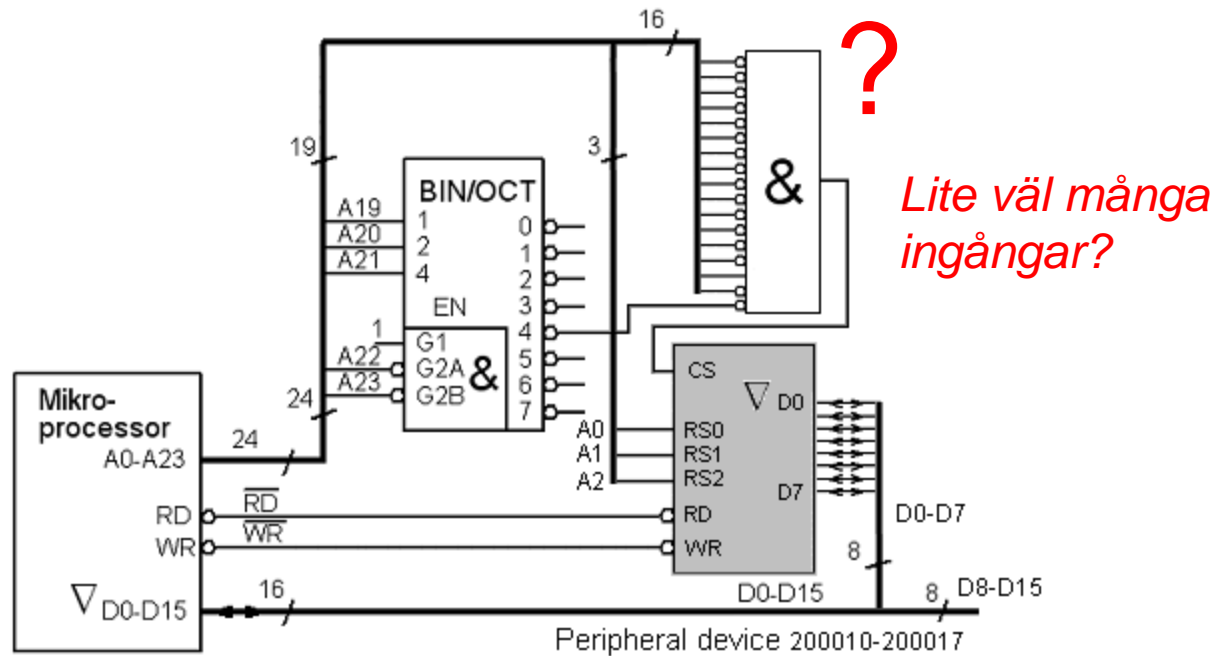


Anslutningarna

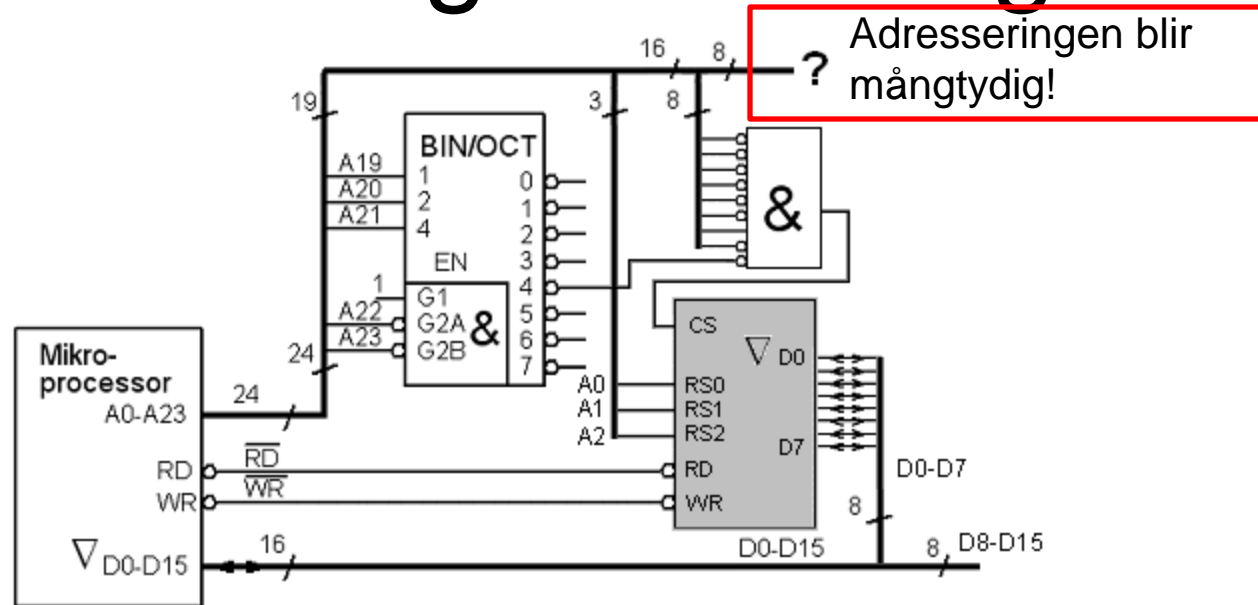




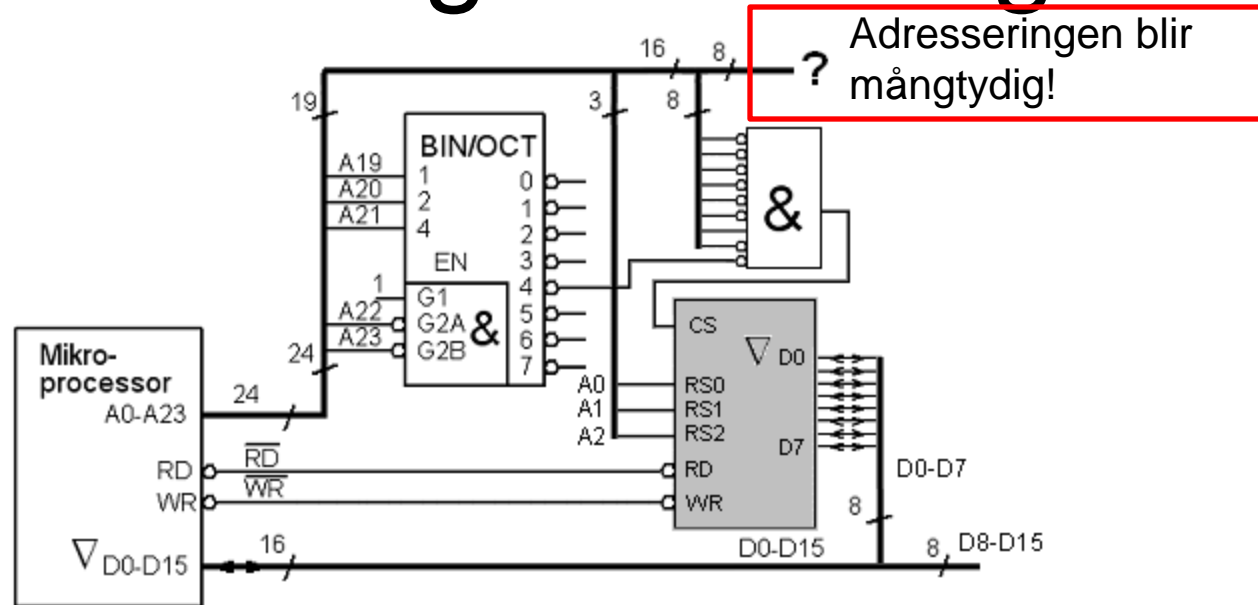
Anslutningarna



Ofullständig avkodning?



Ofullständig avkodning?



För fullständig avkodning använde vi en &-grind med 17 ingångar! I bland gör man en **ofullständig avkodning**. Man struntar då i att ta med alla adress-signalerna och kan därmed använda en grind med färre ingångar.

I/O-enhetens adressering blir mångtydig, den kan adresseras med många olika adresser, men **den som skriver programkoden bestämmer ju själv vilka adresser det är som används**. Huvudsaken är att man ser till att I/O-enhetens adresser *inte kolliderar* med någon annan enhets adresser.

volatile ?



volatile ?



Eftersom I/O-enheter *inte* är riktiga minnen – det kan verka som om innehållet kan ändras "av sig självt" – så kan man vid programmeringen av processorn behöva "hjälpa" kompilatorn att förstå detta genom att deklarera dem som `volatile` (= flyktiga) i sina datorprogram.

Detta kommer Du att möta i Datorteknik-kursen.

Repetition inför tentamen



ÖH 6.10 Kombinatoriskt nät med 5 variabler

$$f(x_4, x_3, x_2, x_1, x_0) = \sum m(9, 11, 12, 13, 14, 15, 16, 18, 24, 25, 26, 27)$$

$$f(x_4, x_3, x_2, x_1, x_0) \quad f = ? \quad \bar{f} = ?$$

K-MAP med 5 eller 6 variabler kan dyka upp på tentamen!

	x_4	x_3	x_2	x_1	x_0	f		x_4	x_3	x_2	x_1	x_0	f
0	0	0	0	0	0	0	16	1	0	0	0	0	1
1	0	0	0	0	1	0	17	1	0	0	0	1	0
2	0	0	0	1	0	0	18	1	0	0	1	0	1
3	0	0	0	1	1	0	19	1	0	0	1	1	0
4	0	0	1	0	0	0	20	1	0	1	0	0	0
5	0	0	1	0	1	0	21	1	0	1	0	1	0
6	0	0	1	1	0	0	22	1	0	1	1	0	0
7	0	0	1	1	1	0	23	1	0	1	1	1	0
8	0	1	0	0	0	0	24	1	1	0	0	0	1
9	0	1	0	0	1	1	25	1	1	0	0	1	1
10	0	1	0	1	0	0	26	1	1	0	1	0	1
11	0	1	0	1	1	1	27	1	1	0	1	1	1
12	0	1	1	0	0	1	28	1	1	1	0	0	0
13	0	1	1	0	1	1	29	1	1	1	0	1	0
14	0	1	1	1	0	1	30	1	1	1	1	0	0
15	0	1	1	1	1	1	31	1	1	1	1	1	0

6.10 Kombinatoriskt nät med 5 variabler

$$f(x_4, x_3, x_2, x_1, x_0) = \sum m(9, 11, 12, 13, 14, 15, 16, 18, 24, 25, 26, 27)$$

$$f(x_4, x_3, x_2, x_1, x_0) \quad f = ?$$

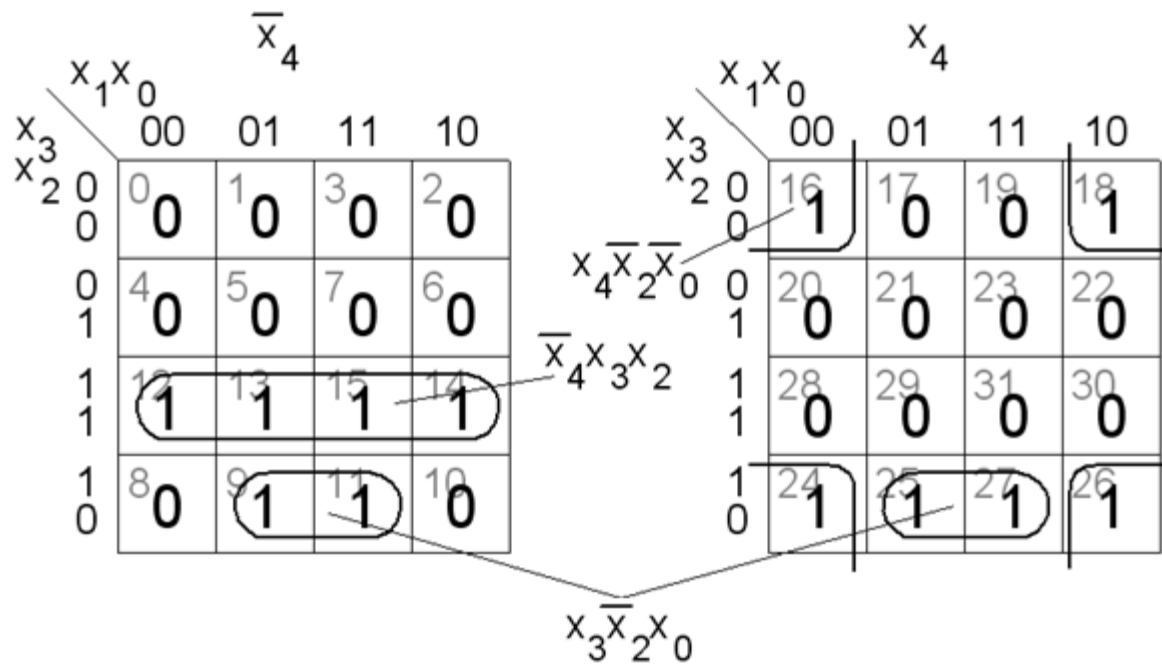
		\bar{x}_4			
		x_1x_0	00	01	11
x_3	0	0	1	3	2
	0	0	0	0	0
x_2	0	4	5	7	6
	1	0	0	0	0
x_1	1	12	13	15	14
	1	1	1	1	1
x_0	1	8	9	11	10
	0	0	1	1	0

		x_4			
		x_1x_0	00	01	11
x_3	0	16	17	19	18
	0	1	0	0	1
x_2	0	20	21	23	22
	1	0	0	0	0
x_1	1	28	29	31	30
	1	0	0	0	0
x_0	1	24	25	27	26
	0	1	1	1	1

6.10 Kombinatoriskt nät med 5 variabler

$$f(x_4, x_3, x_2, x_1, x_0) = \sum m(9, 11, 12, 13, 14, 15, 16, 18, 24, 25, 26, 27)$$

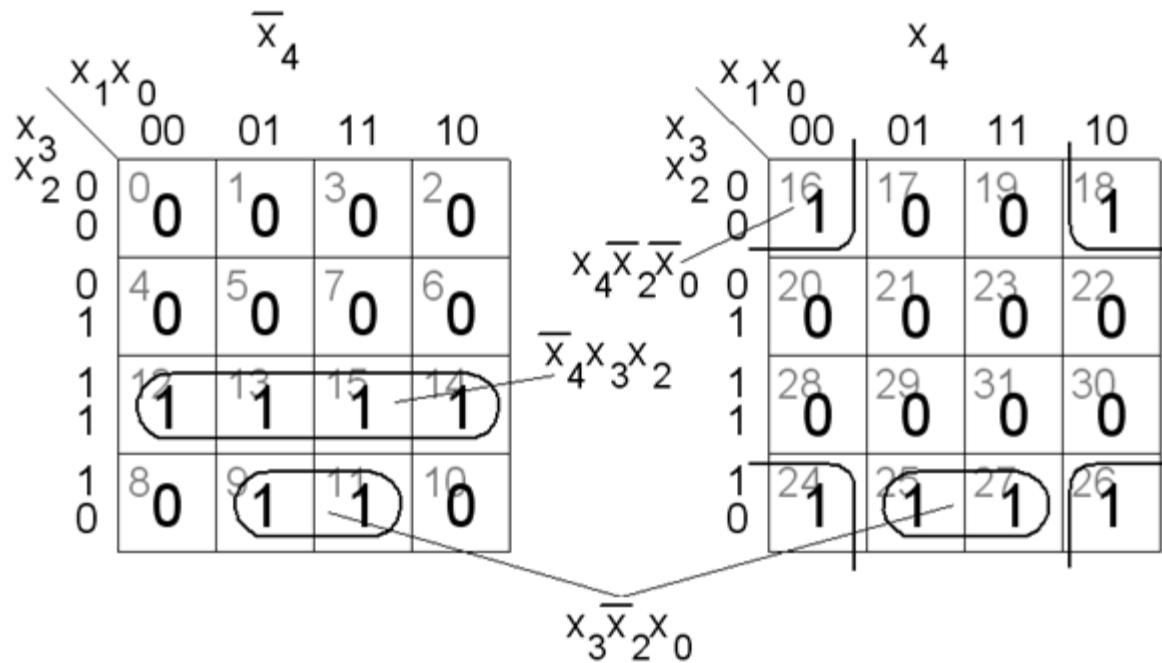
$$f(x_4, x_3, x_2, x_1, x_0) \quad f = ?$$



6.10 Kombinatoriskt nät med 5 variabler

$$f(x_4, x_3, x_2, x_1, x_0) = \sum m(9, 11, 12, 13, 14, 15, 16, 18, 24, 25, 26, 27)$$

$$f(x_4, x_3, x_2, x_1, x_0) \quad f = ?$$



$$f = \bar{x}_4 \bar{x}_3 x_2 + x_3 \bar{x}_2 x_0 + x_4 \bar{x}_2 \bar{x}_0$$

6.10 Kombinatoriskt nät med 5 variabler

$$f(x_4, x_3, x_2, x_1, x_0) = \sum m(9, 11, 12, 13, 14, 15, 16, 18, 24, 25, 26, 27)$$

$$f(x_4, x_3, x_2, x_1, x_0) \quad \overline{f} = ?$$

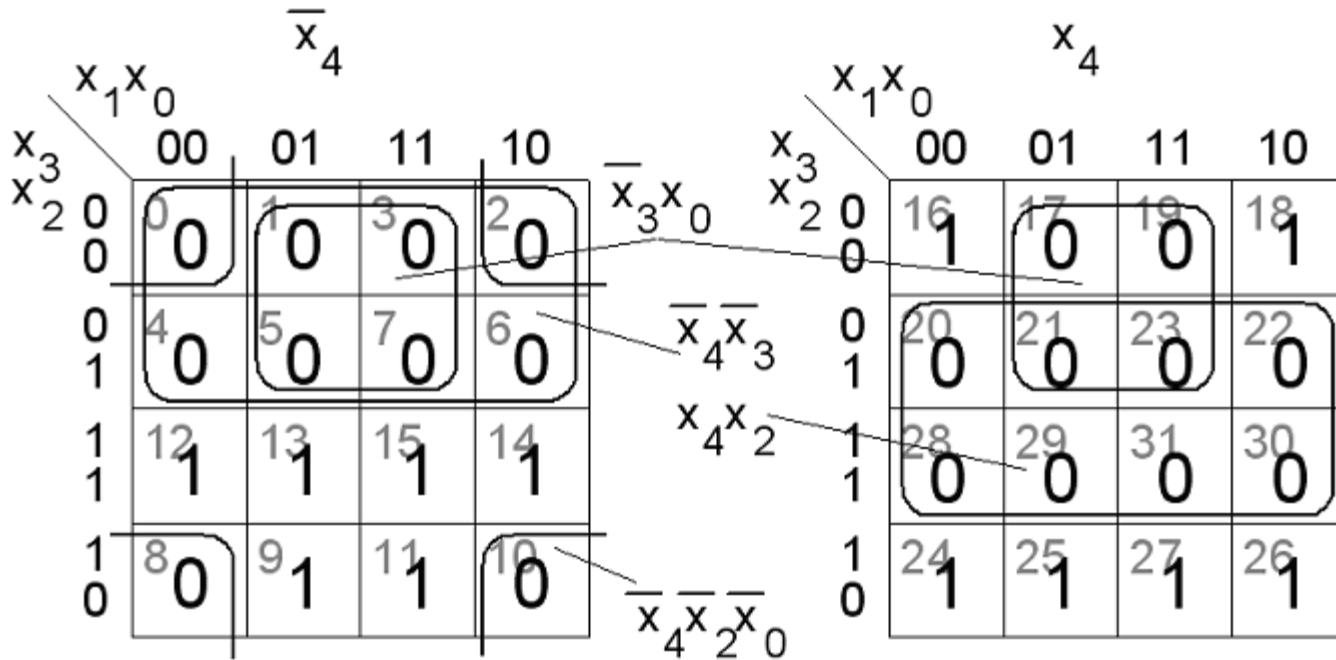
		\overline{x}_4			
		x_1x_0 00	01	11	10
x_3	0	0	1	3	2
	0	0	0	0	0
x_2	0	4	5	7	6
	1	0	0	0	0
	1	12	13	15	14
	1	1	1	1	1
	1	8	9	11	10
	0	0	1	1	0

		x_4			
		x_1x_0 00	01	11	10
x_3	0	16	17	19	18
	0	1	0	0	1
x_2	0	20	21	23	22
	1	0	0	0	0
	1	28	29	31	30
	1	0	0	0	0
	1	24	25	27	26
	0	1	1	1	1

6.10 Kombinatoriskt nät med 5 variabler

$$f(x_4, x_3, x_2, x_1, x_0) = \sum m(9, 11, 12, 13, 14, 15, 16, 18, 24, 25, 26, 27)$$

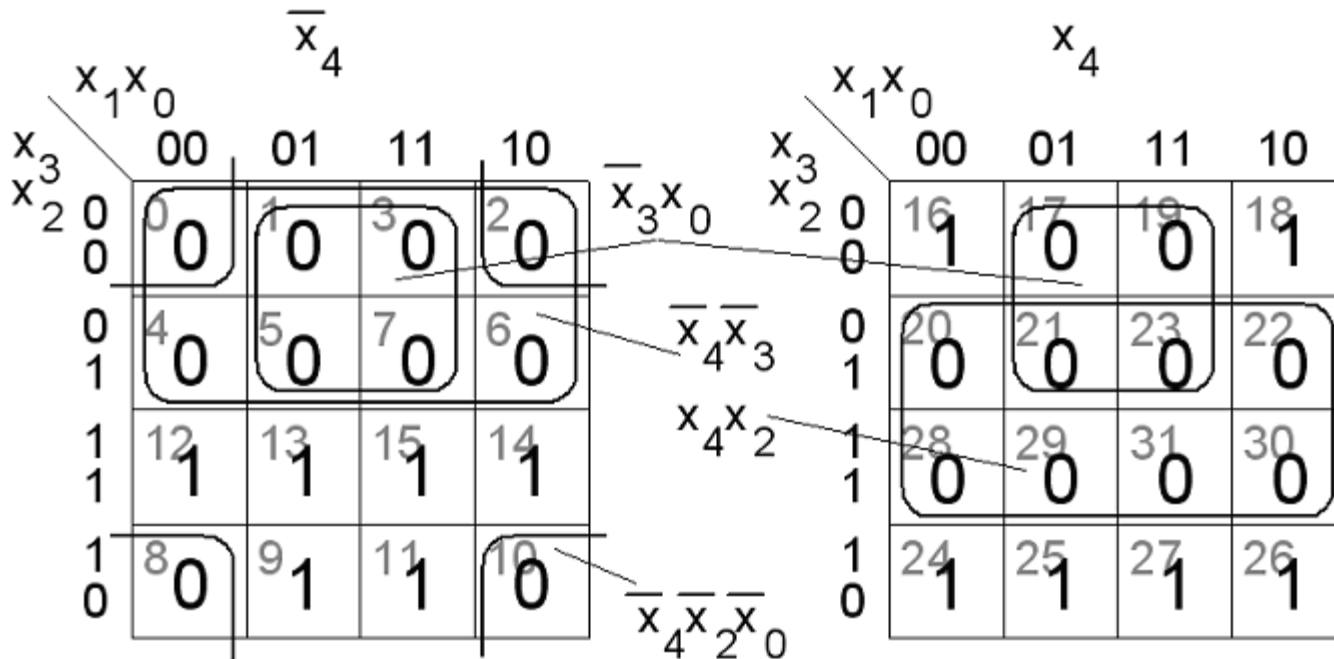
$$f(x_4, x_3, x_2, x_1, x_0) \quad \bar{f} = ?$$



6.10 Kombinatoriskt nät med 5 variabler

$$f(x_4, x_3, x_2, x_1, x_0) = \sum m(9, 11, 12, 13, 14, 15, 16, 18, 24, 25, 26, 27)$$

$$f(x_4, x_3, x_2, x_1, x_0) \quad \bar{f} = ?$$

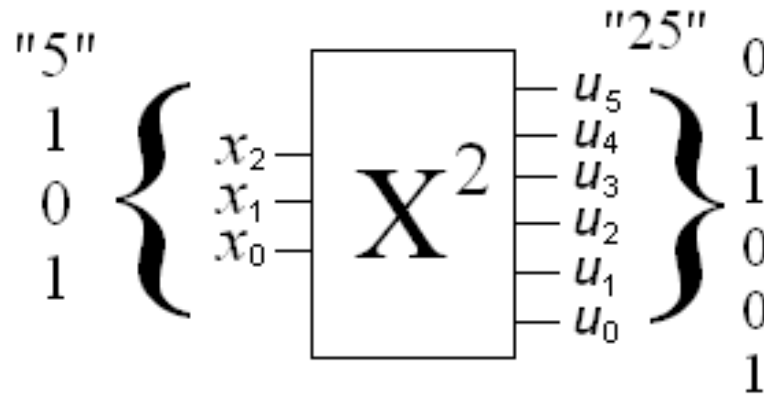


$$\bar{f} = \bar{x}_4 \bar{x}_3 + \bar{x}_3 x_0 + x_4 x_2 + \bar{x}_4 x_2 x_0$$

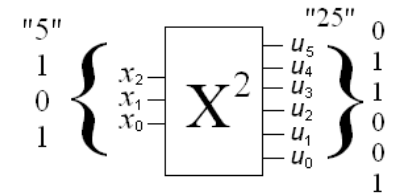
ÖH 8.1 Binär kvadrerare

Ta fram de boolska ekvationerna för ett nät på minimerad SP-form som omvandlar ett trebitars binärkodat tal $X (x_2, x_1, x_0)$ till ett binärkodat sexbitstal $U (u_5, u_4, u_3, u_2, u_1, u_0)$ som är lika med kvadraten på talet, dvs. $U = X^2$.

Ex.
Kodomvandlare
Typiskt på **A2**-
delen

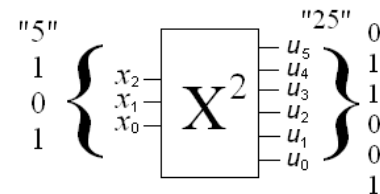


8.1 Sanningstabell



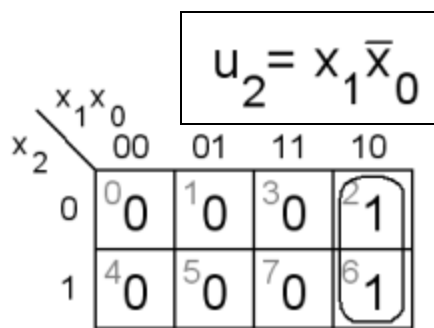
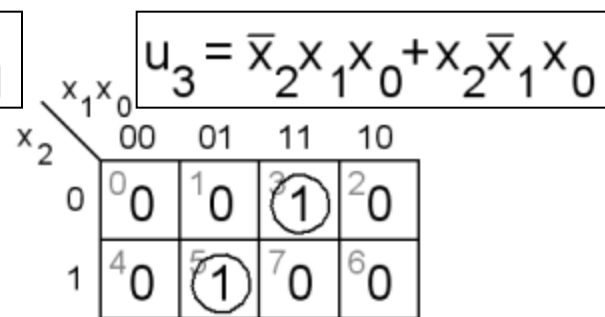
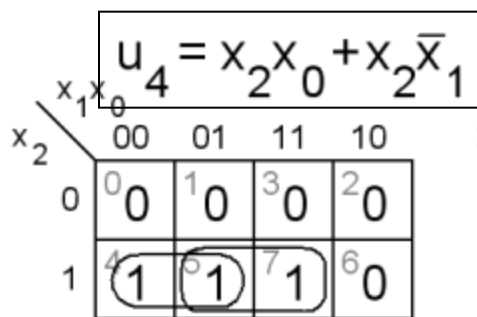
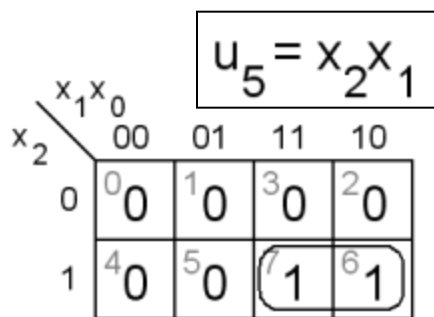
X	x_2	x_1	x_0	$U = X^2$	u_5	u_4	u_3	u_2	u_1	u_0
0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	1
2	0	1	0	4	0	0	0	1	0	0
3	0	1	1	9	0	0	1	0	0	1
4	1	0	0	16	0	1	0	0	0	0
5	1	0	1	25	0	1	1	0	0	1
6	1	1	0	36	1	0	0	1	0	0
7	1	1	1	49	1	1	0	0	0	1

8.1 Karnaughdiagram



Av sanningstabellen framgår att u_1 alltid är lika med 0. u_1 utgången kan därför anslutas 0V (jord) så att den får konstanten 0. Man kan vidare se att u_0 alltid är samma som x_0 . u_0 utgången kan därför förbindas direkt med x_0 ingången.

X	x_2	x_1	x_0	$U = X^2$	u_5	u_4	u_3	u_2	u_1	u_0
0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	1
2	0	1	0	4	0	0	0	1	0	0
3	0	1	1	9	0	0	1	0	0	1
4	1	0	0	16	0	1	0	0	0	0
5	1	0	1	25	0	1	1	0	0	1
6	1	1	0	36	1	0	0	1	0	0
7	1	1	1	49	1	1	0	0	0	1

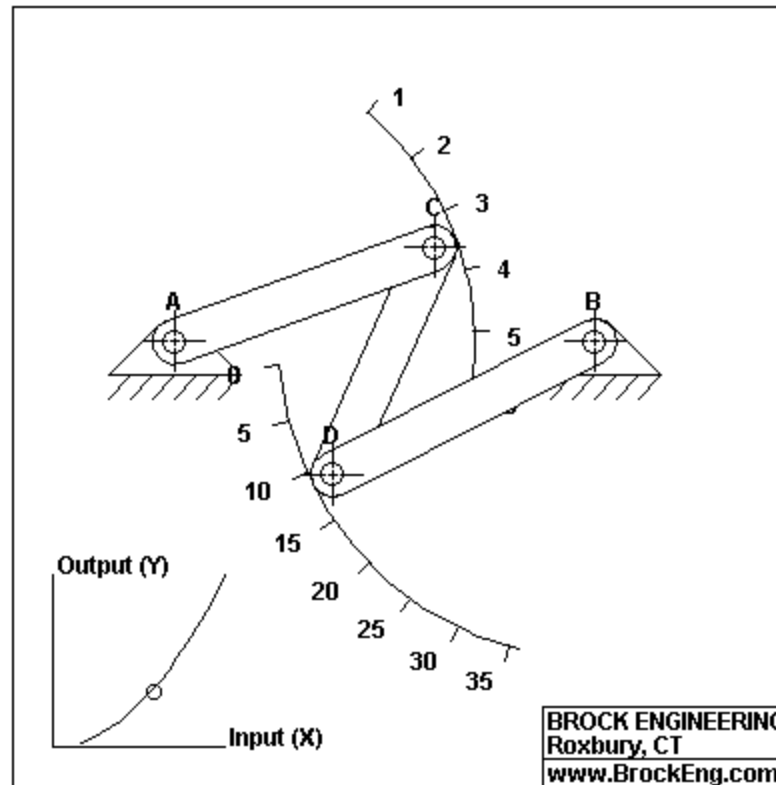


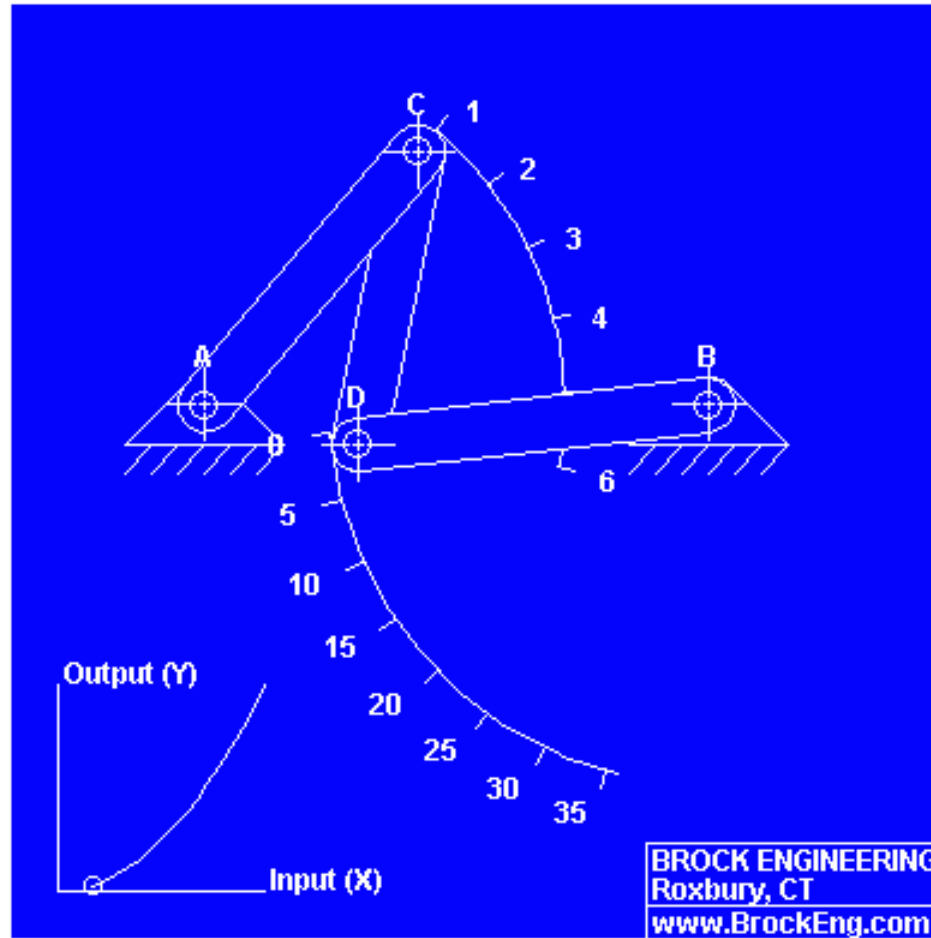
$u_1 = 0$

$u_0 = x_0$

Mekanisk "kvadrerare"

På 1800-talet
innan det fanns
digitalteknik ...





[Brock institute for advaced studies function generator](#)

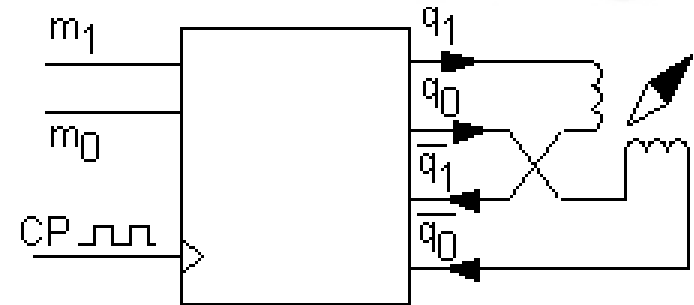
ÖH 10.9 Stegmotorstyrning



En stegmotor är en digital komponent som drivs med pulser.

Stegmotorer brukar anslutas till räknare som räknar Gray-kod.

Figurens räknare har dessutom en modeingång, $m_1 m_0$.

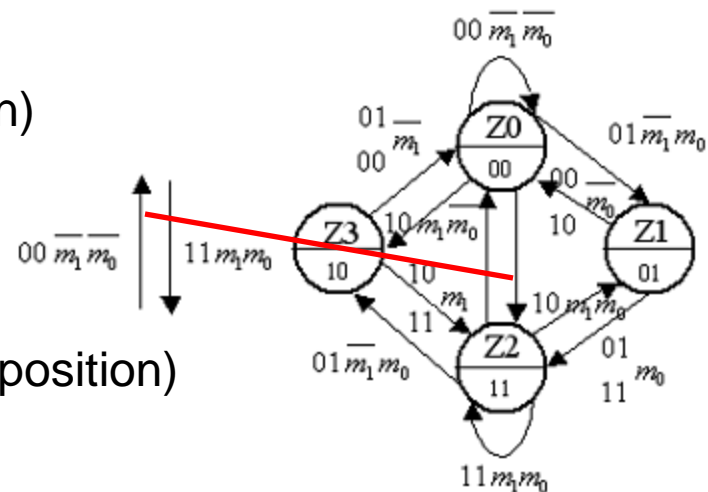


$m_1 m_0 = 00 \rightarrow 0$ -ställning (fix position)

$m_1 m_0 = 01 \rightarrow$ upp-räkning (cw)

$m_1 m_0 = 10 \rightarrow$ ner-räkning (ccw)

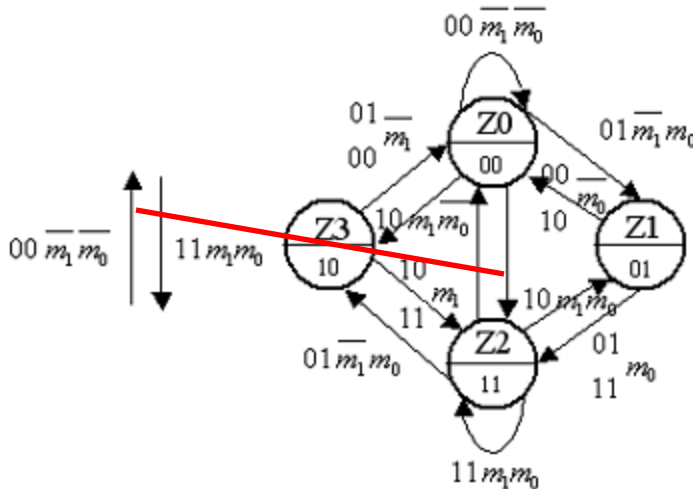
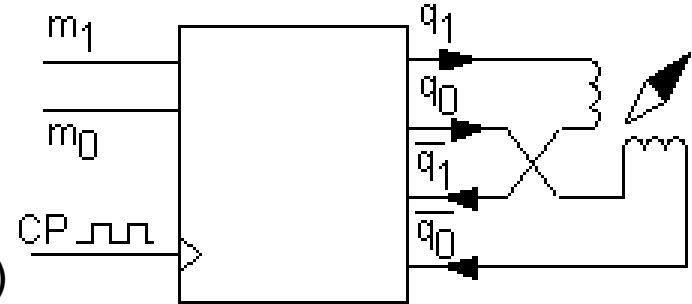
$m_1 m_0 = 11 \rightarrow 1$ -ställning (annan fix position)



Synkrona sekvensnät – är den centrala delen av kursen!
A2-delen. På **B**-delen är tillståndsdigrammet oftast *inte* givet.

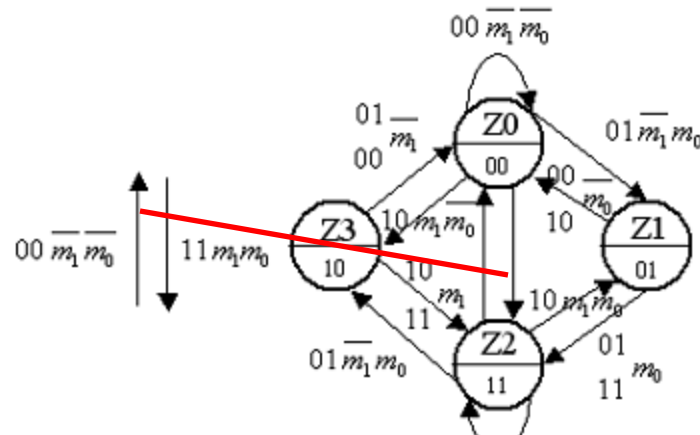
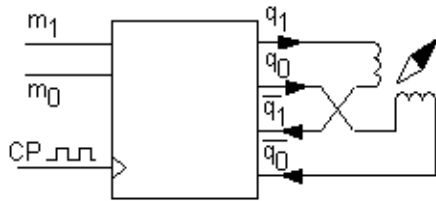
10.9 Tillståndsdigrammet

- $m_1 m_0 = 00 \rightarrow$ 0-ställning (fix position)
- $m_1 m_0 = 01 \rightarrow$ upp-räkning (cw)
- $m_1 m_0 = 10 \rightarrow$ ner-räkning (ccw)
- $m_1 m_0 = 11 \rightarrow$ 1-ställning (annan fix position)



Ibland skriver man boolska vilkor i stället för siffror vid pilarna. I figuren används både vilkor och siffror.

10.9 Tillståndstabell och tillståndsavkodare



$$q_1^+ q_0^+ (q_1 q_0 m_1 m_0)$$

		$m_1 m_0$			
		00	01	11	10
q_1	0	00	01	11	10
	0	00	11	11	00
q_0	1	00	10	11	01
	1	00	00	11	11

$$q_1^+$$

0	0	1	1
0	1	1	0
0	1	1	0
0	0	1	1

$$q_0^+$$

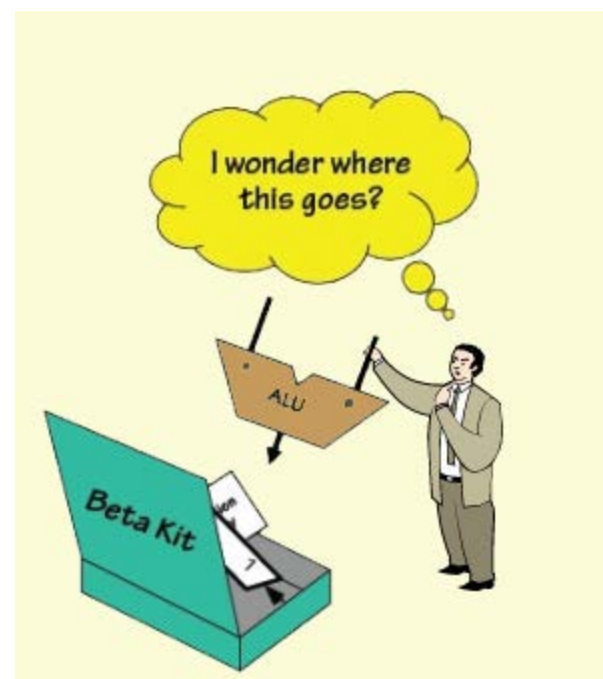
0	1	1	0
0	1	1	0
0	0	1	1
0	0	1	1

$$q_1^+ = q_0 m_0 + \bar{q}_0 m_1$$

$$q_0^+ = q_1 m_1 + \bar{q}_1 m_0$$

William Sandqvist william@kth.se

Hur konstruerar en digitaltekniker ett större system?



BV 10.5

One approach for implementing integer division is to perform repeated subtraction as indicated in pseudo-code.

```
Q = 0;  
R = A  
While ((R - B) ≥ 0) do  
  R = R - B;  
  Q = Q + 1;  
End while;
```

- Give an **ASM chart** that represents the pseudo-code.
- Show the datapath circuit corresponding to part (a).
- Give the **ASM chart** for the control circuit corresponding to part (b).

Algorithmic State Machine

ASM metoden består av följande steg:

1. Skapa en algoritm, med *pseudokod*, som beskriver kretsens önskade funktion.
2. Omvandla pseudokoden till ett *ASM diagram*.
3. Designa ett *Dataflödes-schema (datapath)* utifrån ASM diagrammet.
4. Skapa ett *detaljerat ASM diagram* utifrån dataflödes-schemat.
5. Designa styrlogik utifrån det detaljerade ASM-diagrammet.

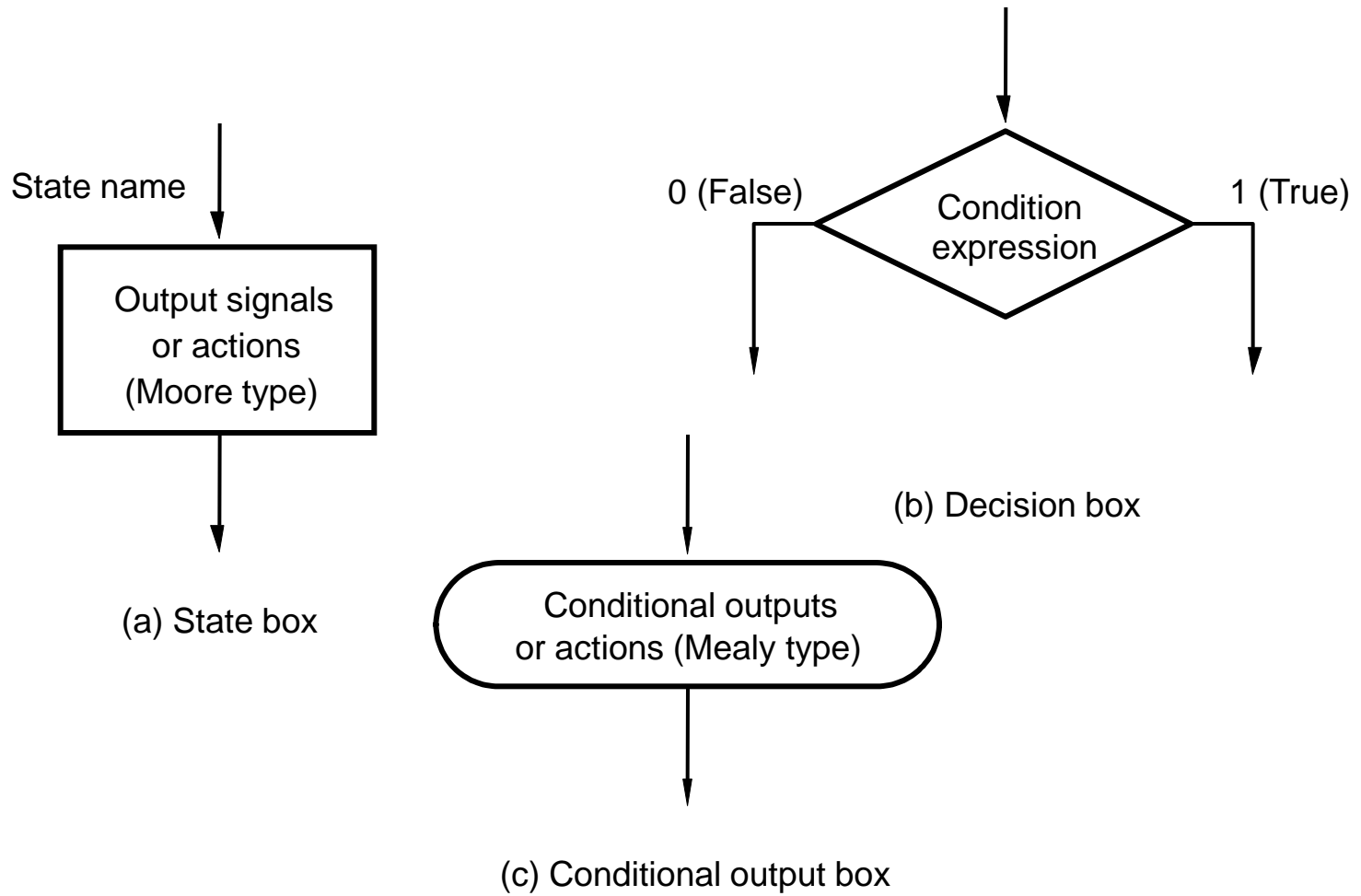


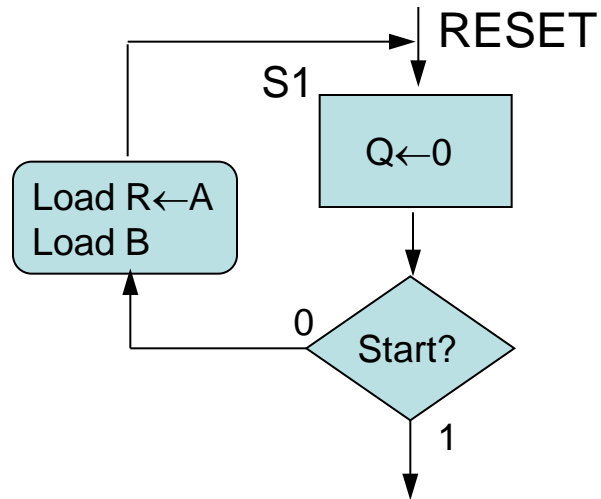
Figure 8.86. Elements used in ASM charts.

BV 10.5 ASM chart

- ASM chart för algoritmen

```
Q = 0;  
R = A  
While ((R - B) ≥ 0) do  
R = R - B;  
Q = Q + 1;  
End while;
```


BV 10.5 ASM chart



Vänta på start (1)!

Q = 0;

R = A

While ((R - B) ≥ 0) do

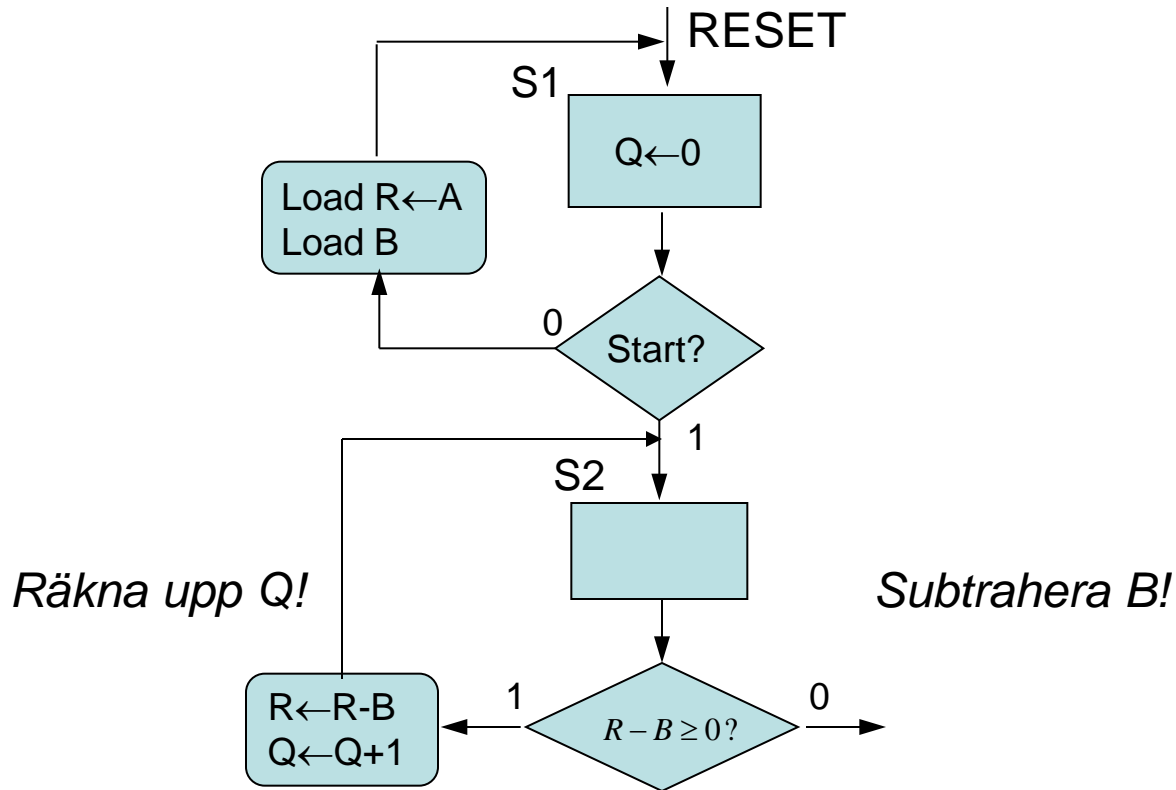
R = R - B;

Q = Q + 1;

End while;

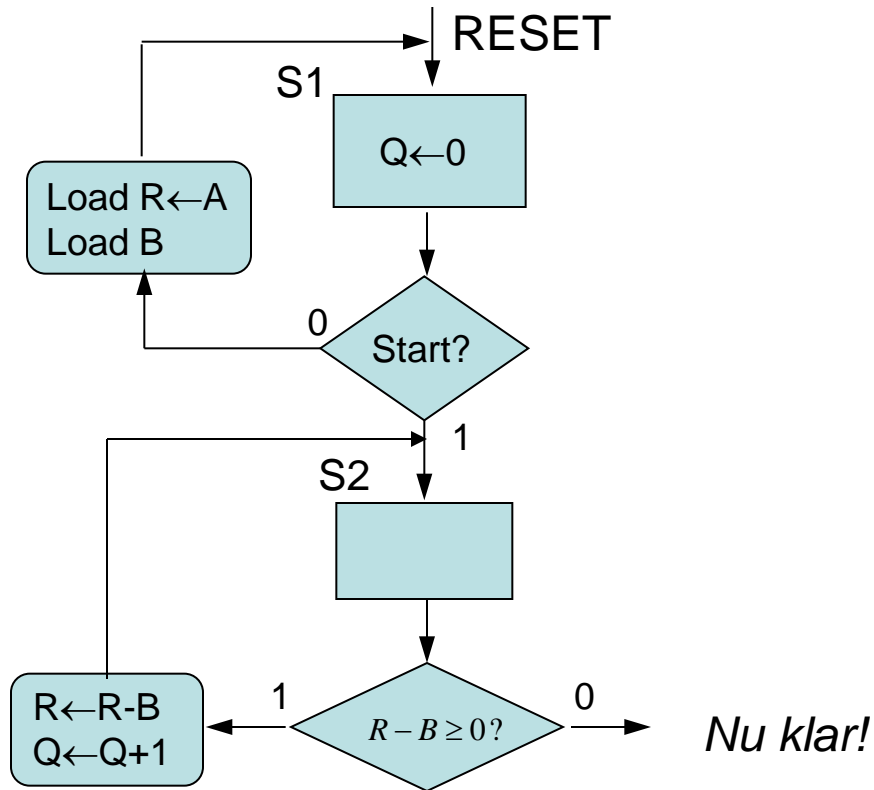
BV 10.5 ASM chart

```
Q = 0;  
R = A  
While ((R - B) ≥ 0) do  
R = R - B;  
Q = Q + 1;  
End while;
```

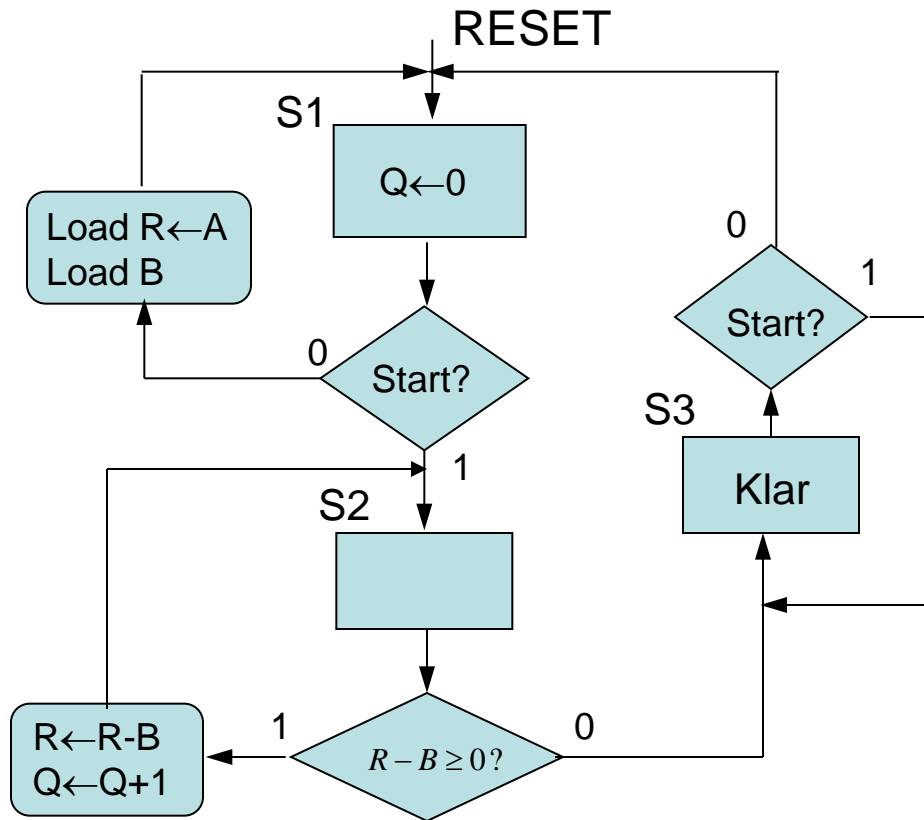


BV 10.5 ASM chart

```
Q = 0;  
R = A  
While ((R - B) ≥ 0) do  
R = R - B;  
Q = Q + 1;  
End while;
```



BV 10.5 ASM chart

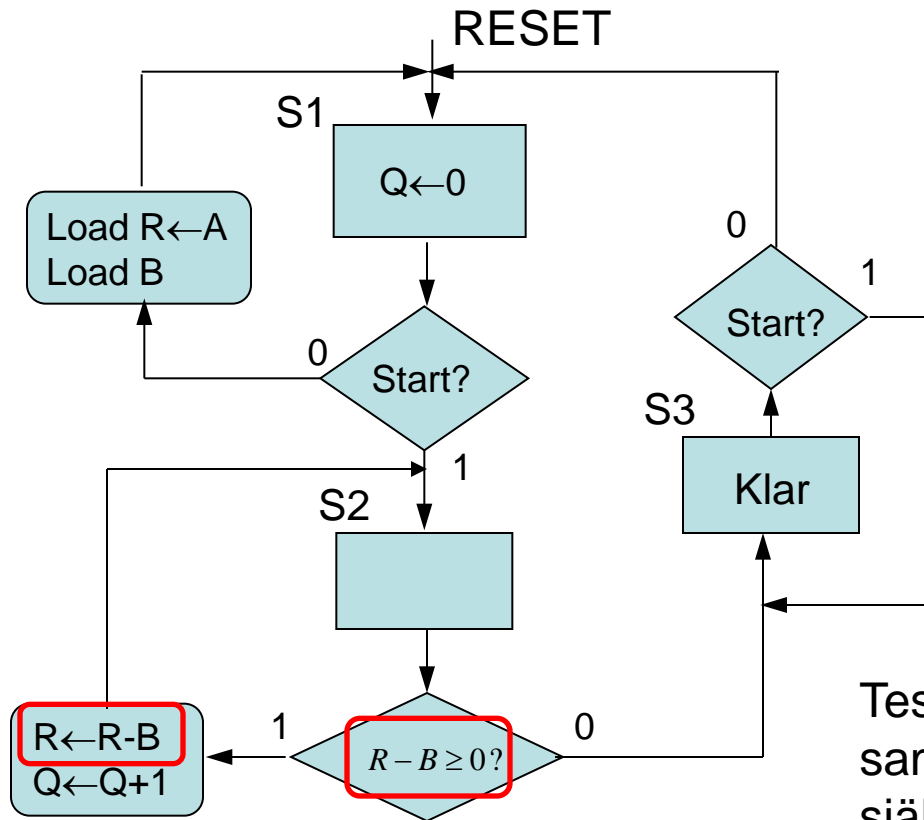


```
Q = 0;
R = A
While ((R - B) ≥ 0) do
R = R - B;
Q = Q + 1;
End while;
```

*Vänta på att Start
släpps (0), för
omstart.*

BV 10.5 ASM chart

```
Q = 0;  
R = A  
While ((R - B) ≥ 0) do  
R = R - B;  
Q = Q + 1;  
End while;
```

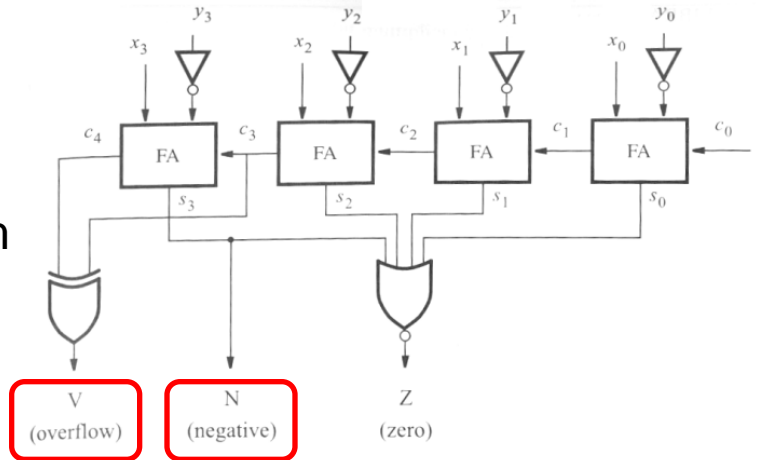


- Hur vet vi när $R - B < 0$?

Test av $R - B \geq 0$ görs i samma moment som själva operationen $R - B$ genom att man inspekterar flaggor.

Kommer Du ihåg? \geq

Adderare
kopplad som
komparator



$$X - Y$$

$$V = c_4 \oplus c_3 \quad N = s_3$$

$$Z = \overline{(s_3 + s_2 + s_1 + s_0)}$$

$$X = Y \Rightarrow Z = 1$$

$$X < Y \Rightarrow N \oplus V$$

$$X \leq Y \Rightarrow Z + N \oplus V$$

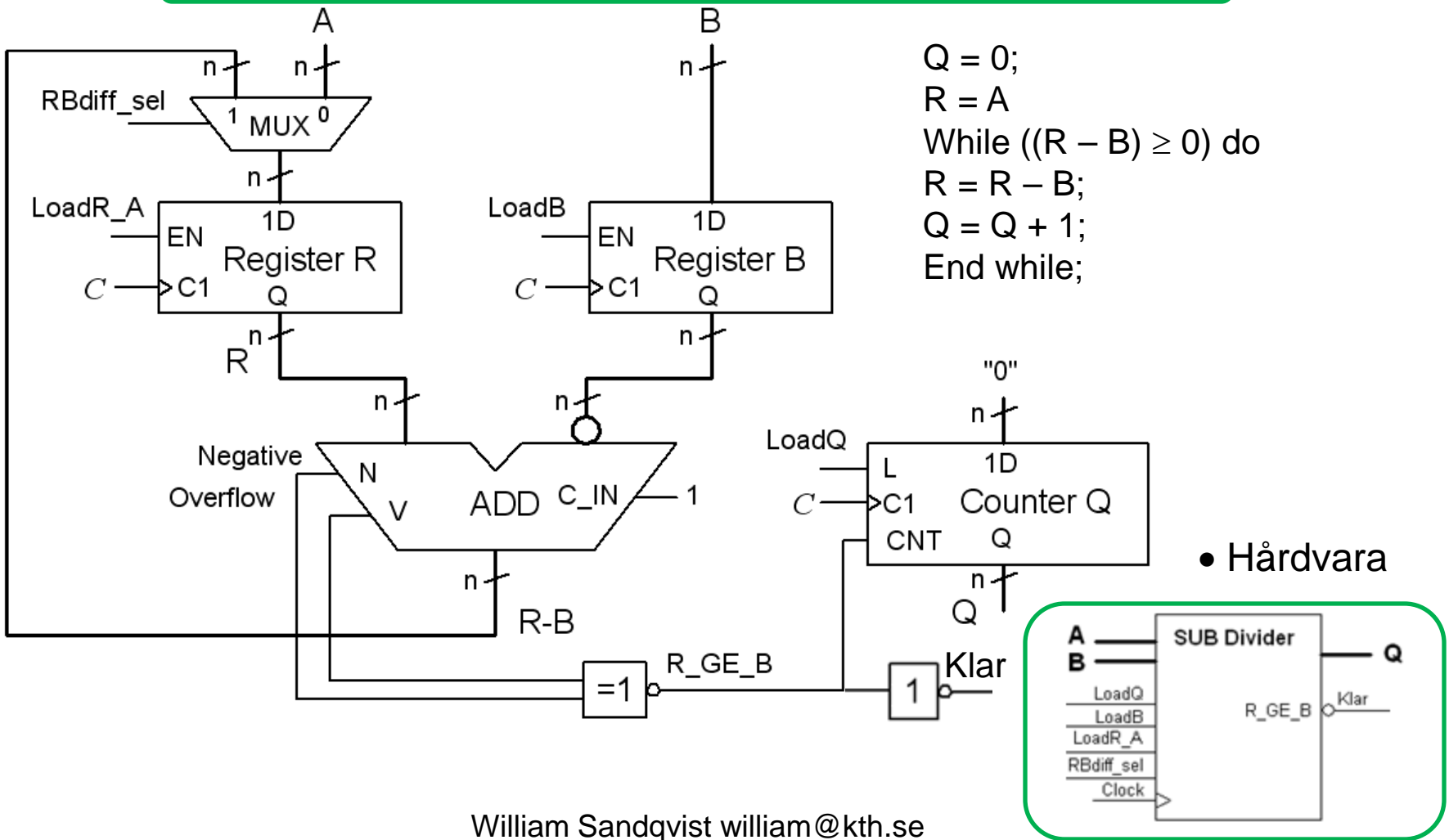
$$X > Y \Rightarrow \overline{Z + N \oplus V} = \overline{Z} \cdot \overline{(N \oplus V)}$$

$$X \geq Y \Rightarrow \overline{N \oplus V}$$

*Så här kan en dator
göra de vanligaste
jämförelserna med
hjälp av flaggorna ...*

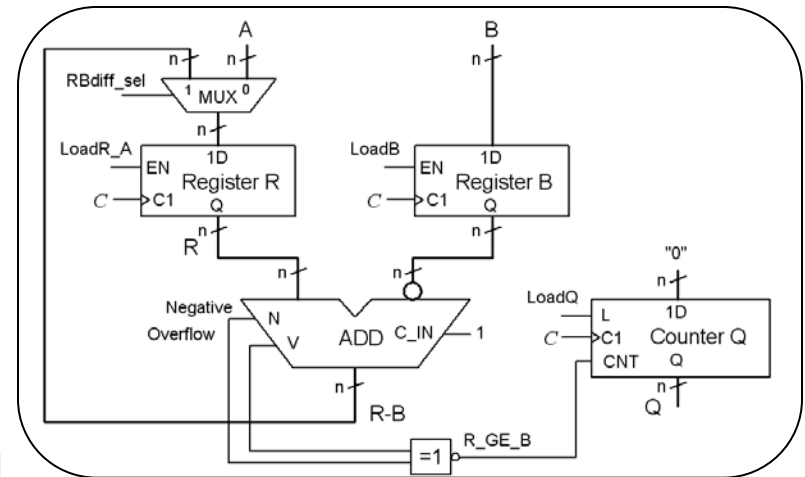
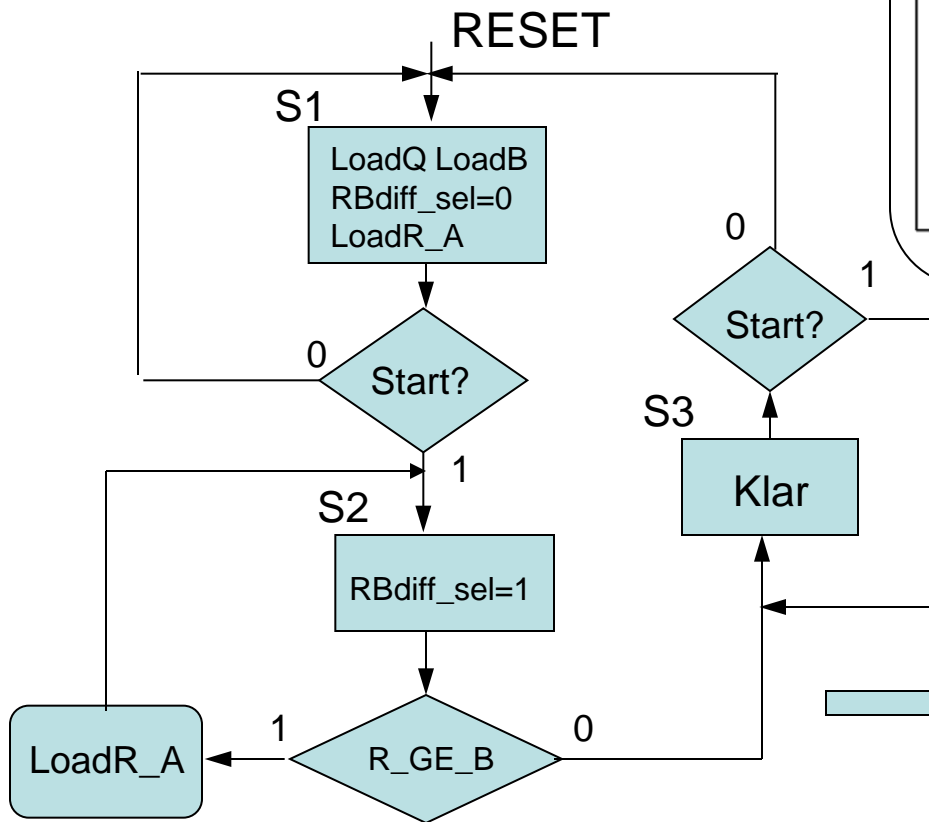
BV 10.5 datapath circuit

- ASM chart ligger till grund för att sätta ihop **hårdvaran**.

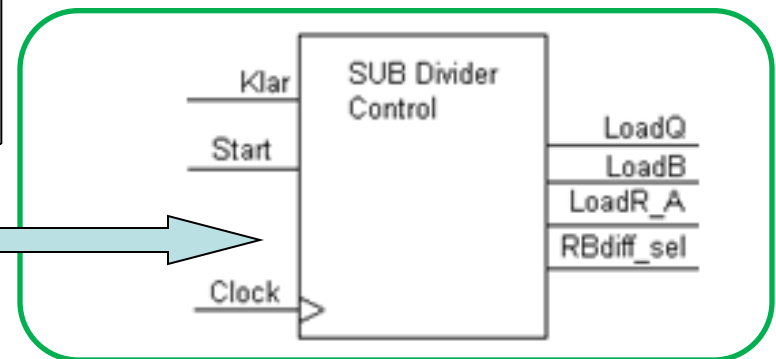


BV 10.5 ASM control

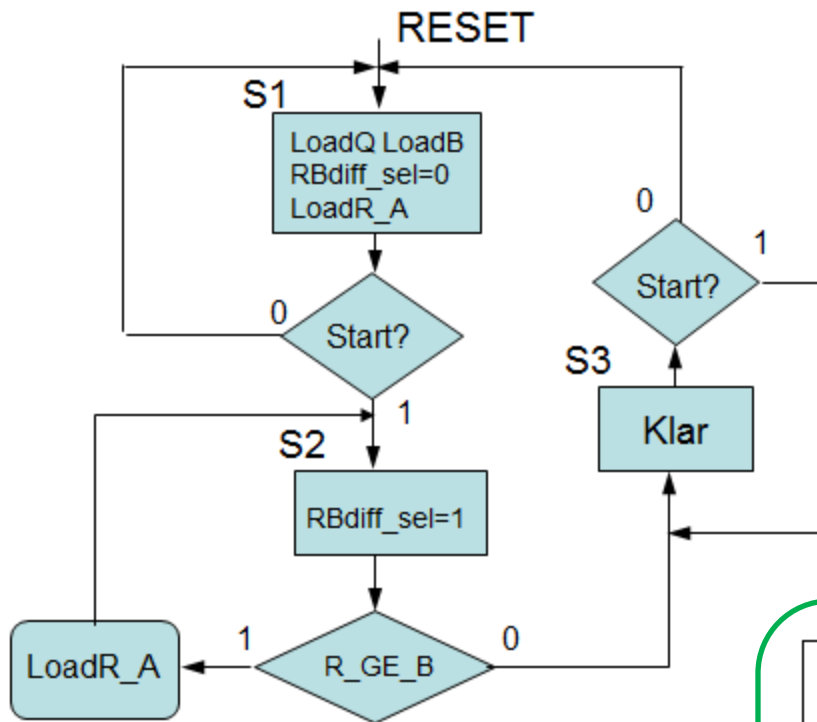
- ASM chart ligger också till grund för tillståndsdigrammet till styrningen.



- Styrningen

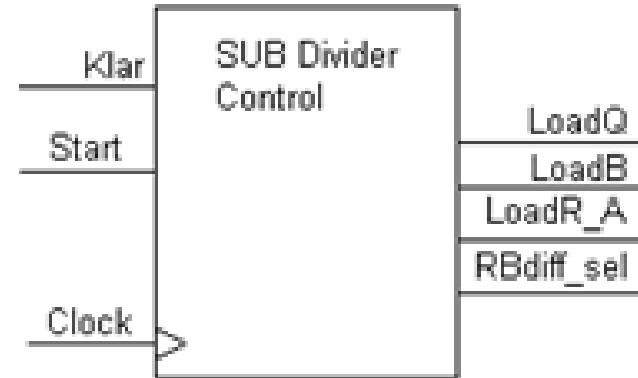


BV 10.5 ASM control

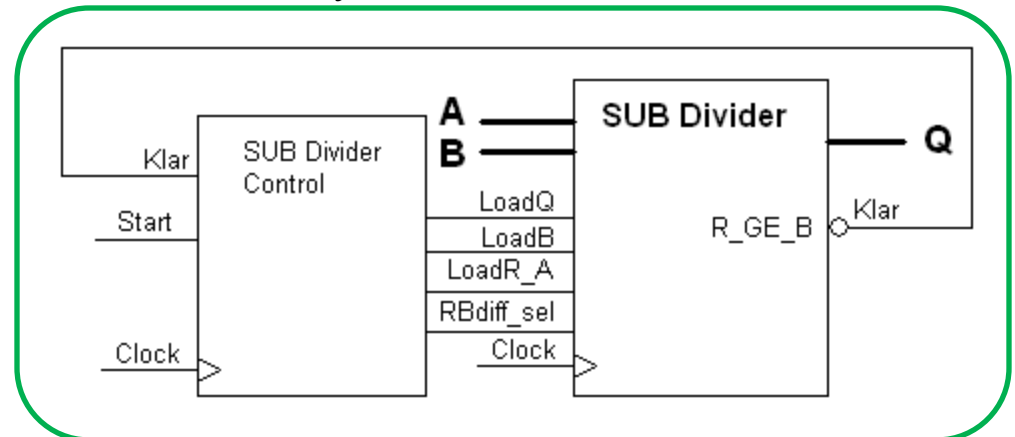


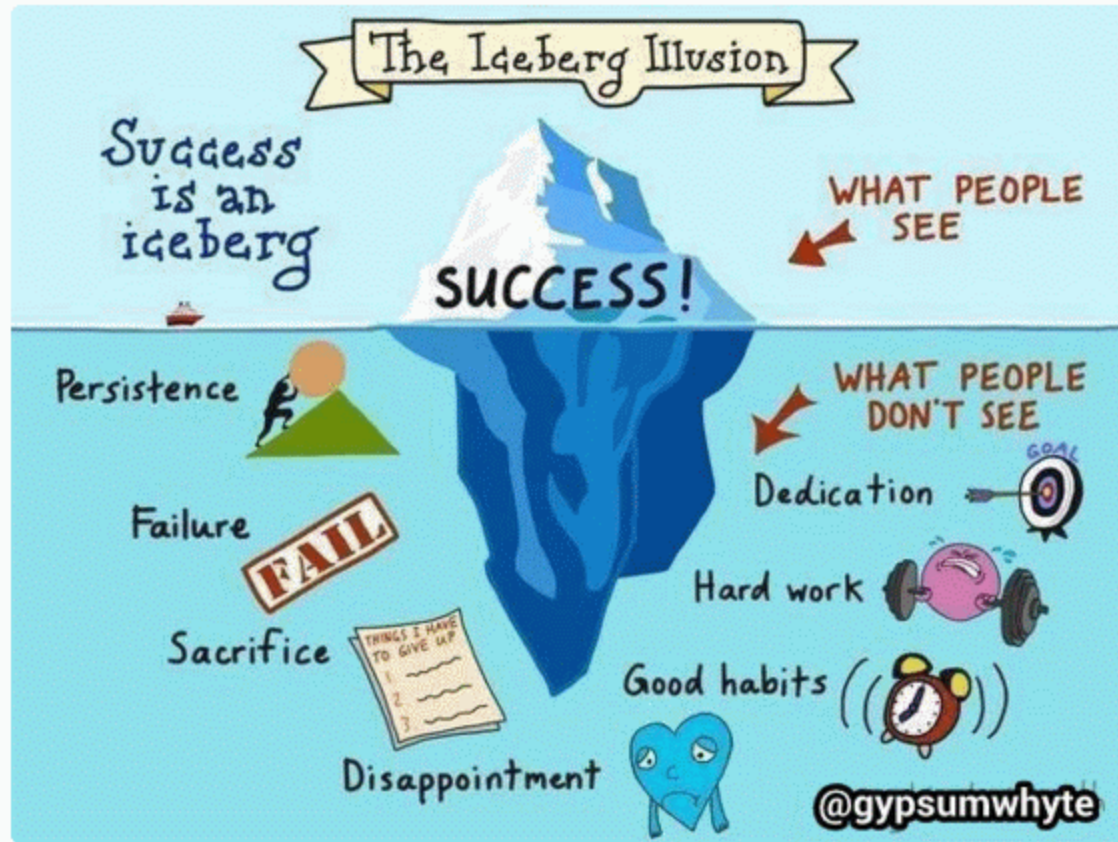
Styrningen "SUB Divider Control" kan konstrueras som en Moore automat med tre tillstånd utifrån tillståndsdigrammet (ASM).

- Styrningen



- Hela systemet





Lycka till på tentan!

William Sandqvist william@kth.se