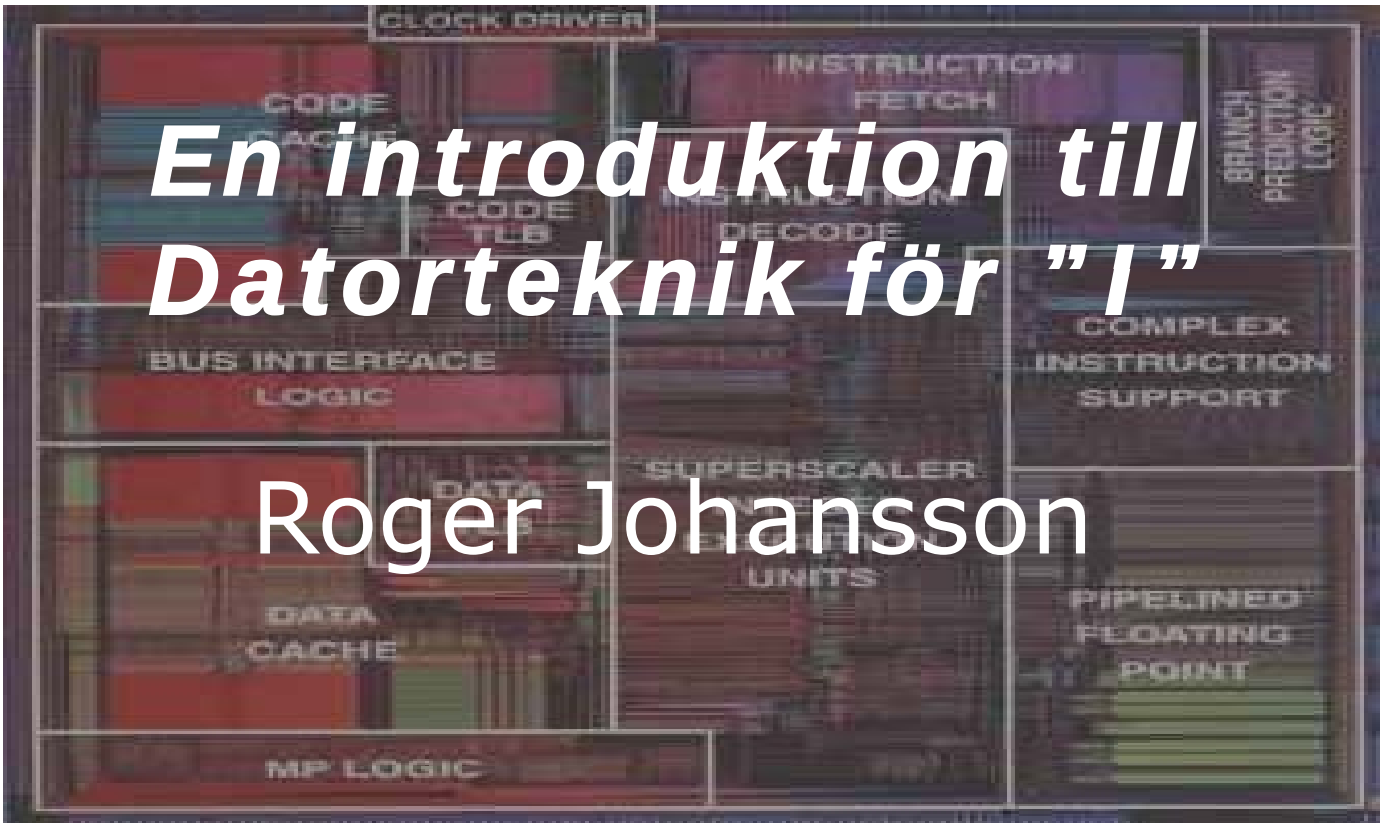


En introduktion till Datorteknik för ”I”

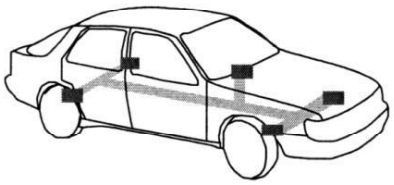
Roger Johansson

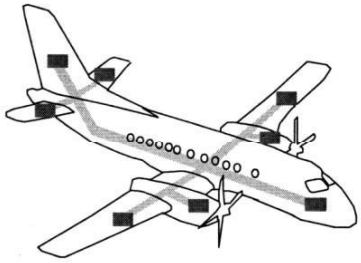
Datortekniken ligger till grund för en lång rad välbekanta vardagsprylar

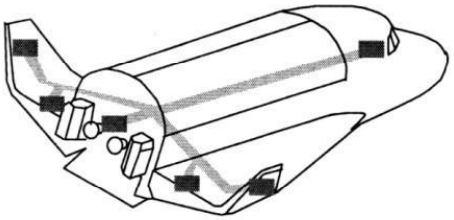
- Mobiltelefoner,
- mediaspelare; mp3, IPOD
- digitalboxar, "laptops“, hemma-bio
- spelkonsoler
- mikrovågsugnar
- huslarm, "smartcards" etc.

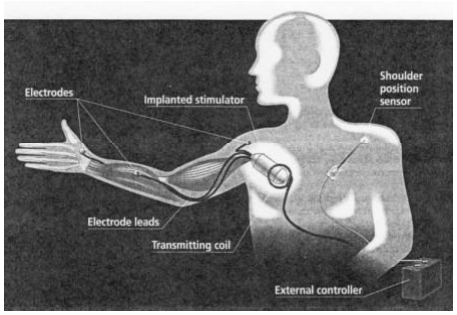
Samma teknik ligger också till grund för ett stort antal av samhällets tekniska system

- arbetsstationer
- bilar
- flygplan
- trafiksystem
- larmsystem
- robotar
- processmaskiner
- medicinsk apparatur
- rymdfarkoster m fl







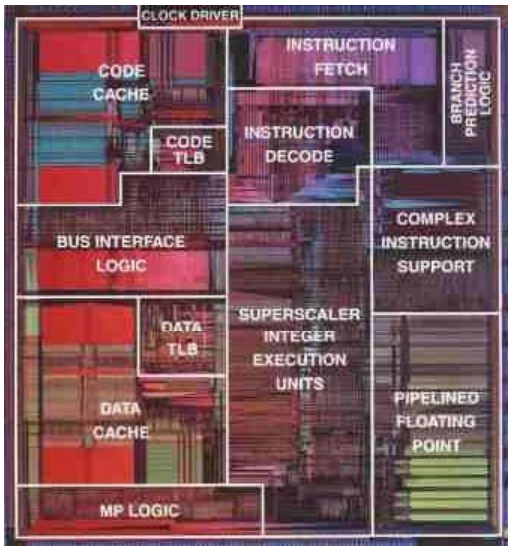


Alla dessa tekniska "prylar" och system innehåller ***digitala system.***

Datorsystem är programmerbara digitala system.

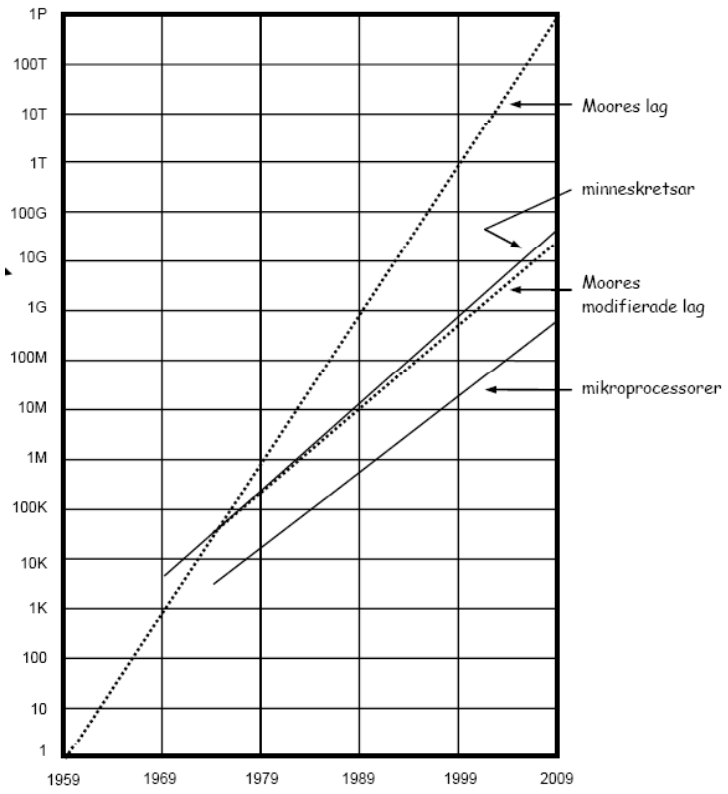
Digitala system baseras på ***digital teknik*** som alltså är fundamental för skapandet av datorsystem.

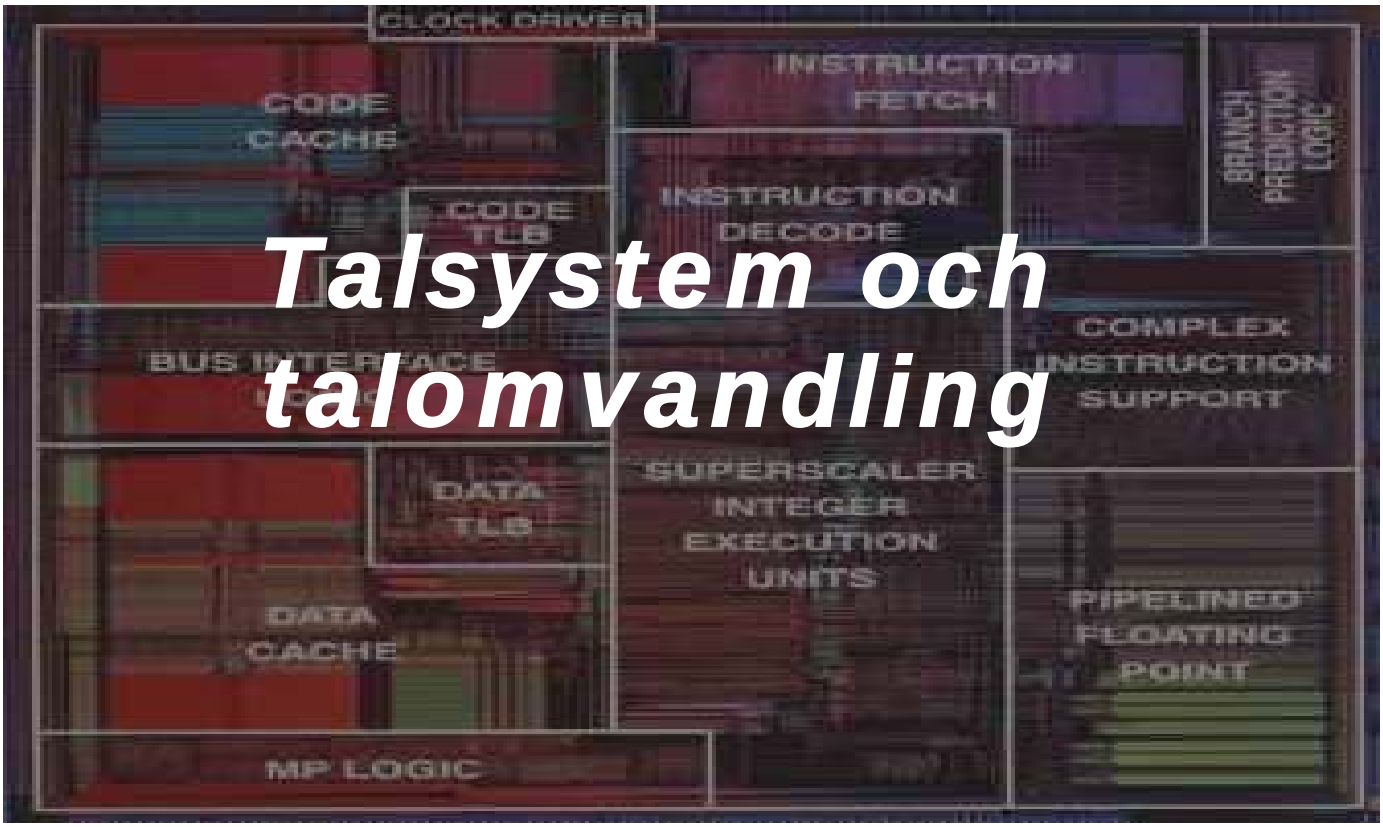
Kärnan i ett datorsystem är en ***mikroprocessor***.



Mikroelektronikens utveckling

Antalet transistorer som ryms på en kiselbricka fördubblas var 18-e månad ....

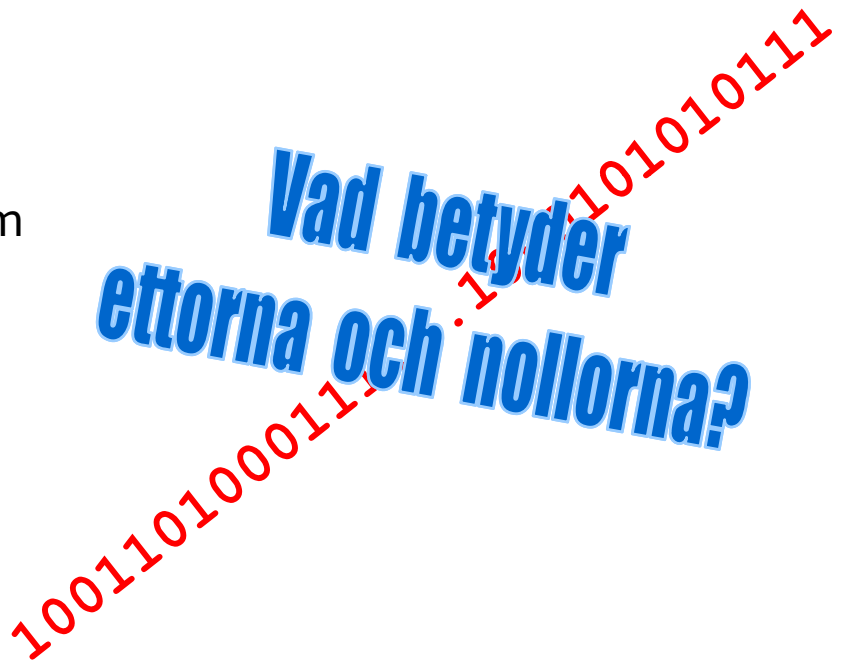




Talsystem och talomvandling

Binär kodning

- Begrepp
- Tal och talsystem
- ASCII-kod
- NBCD



Begrepp vid binär kodning

begrepp	*betydelse*	*exempel...*
bit/bitar	minsta informationsenhet, kan anta två värden	`0` eller `1`
bitsträng binärt ord	sekvens av bitar	`101100100001...`
kodord	$k_7\ k_6\ k_5\ k_4\ k_3\ k_2\ k_1\ k_0$ också ett binärt ord men med en fastställd kodning (betydelse)	`1000001` = "A" (ASCII) `1000001` = 65 (naturligt tal) `1000001` = -127 (heltal)
ordlängd	antal bitar i ordet	
nibble	ordlängden 4 bitar	`0101`
byte	ordlängden 8 bitar	`01011100`

Positionssystem, 10-decimalt

Ett ***N***-bitars tal. ***N*** = ***n***+***m*** där ***n*** är antalet siffror i heltalsdelen och ***m*** är antalet siffror i bråkdelen skriver vi allmänt:

$$d_{n-1}d_{n-2}\ldots d_1 d_0 \,.\, d_{-1}d_{-2}\ldots d_{-(m-1)}d_{-m}$$

Mest signifikanta siffra (MSD)

Decimalpunkt

Minst signifikanta siffra (LSD)

Exempelvis, talet:

$$123{,}456 = 1\times10^2 + 2\times10^1 + 3\times10^0 + 4\times10^{-1} + 5\times10^{-2} + 6\times10^{-3}$$
$$= 100 + 20 + 3 + 0{,}4 + 0{,}05 + 0{,}006$$

Där ***N***=6, ***n***=***m***=3, varje siffras vikt avgörs av dess position i talet...

Positionssystem, generellt

Talbasen β kan dock vara praktiskt taget vad som helst...

$$d_{n-1}\times\beta^{n-1} + d_{n-2}\times\beta^{n-2} + \ldots + d_1\times\beta^1 + d_0\times\beta^0 + d_{-1}\times\beta^{-1} + d_{-2}\times\beta^{-2} + \ldots d_{-(m-1)}\times\beta^{-(m-1)} + d_{-m}\times\beta^{-m}$$

Exempel: β=10

$$d_{n-1}\times10^{n-1} + d_{n-2}\times10^{n-2} + \ldots + d_1\times10^1 + d_0\times10^0 + d_{-1}\times10^{-1} + d_{-2}\times10^{-2} + \ldots d_{-(m-1)}\times10^{-(m-1)} + d_{-m}\times10^{-m}$$

Exempel: β=2

$$d_{n-1}\times2^{n-1} + d_{n-2}\times2^{n-2} + \ldots + d_1\times2^1 + d_0\times2^0 + d_{-1}\times2^{-1} + d_{-2}\times2^{-2} + \ldots d_{-(m-1)}\times2^{-(m-1)} + d_{-m}\times2^{-m}$$

Vi använder vanligen det enklare skrivsättet

$$N = (d_{n-1}d_{n-2}\ldots d_1 d_0 \,.\, d_{-1}d_{-2}\ldots d_{-(m-1)}d_{-m})_\beta$$

Talbaser

Vi använder huvudsakligen tre olika talbaser:

Decimalt, för att vi är vana vid det.

Binärt, för att det motsvarar informationselementen i det digitala systemet.

Hexadecimalt, därför att det är ett bekvämt sätt att skriva grupper av binära siffror

Exempel:

$(13)_{10} = (1101)_2 = (D)_{16}$

bas 10 decimalt	bas 2 binärt	bas 16 hexadecimalt
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Talomvandlingar

För talomvandling ***till*** basen 10 använder vi definitionen direkt...

Exempel:
Omvandla till decimal form:

a) $(110.111)_2$

b) $(1A.8F)_{16}$

Lösning:

a) $(110.111)_2 = \{N=6, n=m=3, \beta=2\} =$
$1\times2^2 + 1\times2^1 + 0\times2^0 + 1\times2^{-1} + 1\times2^{-2} + 1\times2^{-3} =$
$4 + 2 + 0 + 1/2 + 1/4 + 1/8 = 6 + 7/8 = (6{,}875)_{10}$

b) $(1A.8F)_{16} = \{N=4, n=m=2, \beta=16\} =$
$1\times16^1 + 10\times16^0 + 8\times16^{-1} + 15\times16^{-2} =$
$16 + 10 + 8/16 + 15/256 = 26 + 143/256 = (26{,}55859375)_{10}$

Omvandling från N_{10} till N_β

1. Dela upp N_{10} i heltalsdel och bråktalsdel.
2. Heltalsdelen omvandlas via succesiva divisioner med β.
3. Bråkdelen omvandlas via succesiva multiplikationer med β.

Exempel:
Omvandla $(122{,}18)_{10}$ till binär form. Bråkdelen avkortas vid behov till 7 korrekta bråksiffror.

1. Omvandla $(122)_{10}$ till binär form

$$
\begin{aligned}
122/2 &= 61 + 0/2 \rightarrow d_0 = 0\\
61/2 &= 30 + 1/2 \rightarrow d_1 = 1\\
30/2 &= 15 + 0/2 \rightarrow d_2 = 0\\
15/2 &= 7 + 1/2 \rightarrow d_3 = 1\\
7/2 &= 3 + 1/2 \rightarrow d_4 = 1\\
3/2 &= 1 + 1/2 \rightarrow d_5 = 1\\
1/2 &= 0 + 1/2 \rightarrow d_6 = 1
\end{aligned}
$$

Terminerings-villkor

Heltalsdelen således:

$$(1111010)_2$$

2. Omvandla $(0,18)_{10}$ till binär form

$$
\begin{aligned}
0,18 \times 2 &= \mathbf{0},36 \rightarrow d_{-1} = \mathbf{0} \\
0,36 \times 2 &= \mathbf{0},72 \rightarrow d_{-2} = \mathbf{0} \\
0,72 \times 2 &= \mathbf{1},44 \rightarrow d_{-3} = \mathbf{1} \\
0,44 \times 2 &= \mathbf{0},88 \rightarrow d_{-4} = \mathbf{0} \\
0,88 \times 2 &= \mathbf{1},76 \rightarrow d_{-5} = \mathbf{1} \\
0,76 \times 2 &= \mathbf{1},52 \rightarrow d_{-6} = \mathbf{1} \\
0,52 \times 2 &= \mathbf{1},04 \rightarrow d_{-7} = \mathbf{1}
\end{aligned}
$$

Termineringsvillkor
enligt uppgiftstexten
7 st. korrekta bråksiffror

Bråkdelen således:

$$(0.0010111)_2$$

Omvandla till hexadecimal form

Exempel:
Omvandla $(122,18)_{10}$ till hexadecimal form. Bråkdelen avkortas vid behov till 2 korrekta bråksiffror.

Heltalsdelen:

$$
\begin{aligned}
122/16 &= 7 + 10/16 \rightarrow d_0 = (10)_{10} = (A)_{16} \\
7/16 &= 0 + 7/16 \rightarrow d_1 = (7)_{10} = (7)_{16}
\end{aligned}
$$

Bråkdelen:

$$
\begin{aligned}
0,18 \times 16 &= 2,88 \rightarrow d_{-1} = (2)_{10} = (2)_{16} \\
0,88 \times 16 &= 14,08 \rightarrow d_{-2} = (14)_{10} = (E)_{16}
\end{aligned}
$$

Svar:

$$(122,18)_{10} \approx 7A.2E$$

Alfanumeriska tecken → ASCII

American Standard Code for Information Interchange

Typiskt användningsområde: Tangentbord

7-bitars ASCII kodning

000	001	010	011	100	101	110	111	$K_6K_5K_4$ $K_3K_2K_1K_0$
NUL	DLE	SP	0	@	P	`	p	0 0 0 0
SOH	DC1	!	1	A	Q	a	q	0 0 0 1
STX	DC2	“	2	B	R	b	r	0 0 1 0
ETX	DC3	#	3	C	S	c	s	0 0 1 1
EOT	DC4	$	4	D	T	d	t	0 1 0 0
ENQ	NAK	%	5	E	U	e	u	0 1 0 1
ACK	SYN	&	6	F	V	f	v	0 1 1 0
BEL	ETB	‘	7	G	W	g	w	0 1 1 1
BS	CAN	(	8	H	X	h	x	1 0 0 0
HT	EM	)	9	I	Y	i	y	1 0 0 1
LF	SUB	*	:	J	Z	j	z	1 0 1 0
VT	ESC	+	;	K	[Ä	k	{ä	1 0 1 1
FF	FS	,	<	L	\Ö	l	|ö	1 1 0 0
CR	GS	-	=	M	]Å	m	}å	1 1 0 1
S0	RS	.	>	N	^	n	~	1 1 1 0
S1	US	/	?	O	_	o	RUBOUT (DEL)	1 1 1 1

ASCII– Exempel

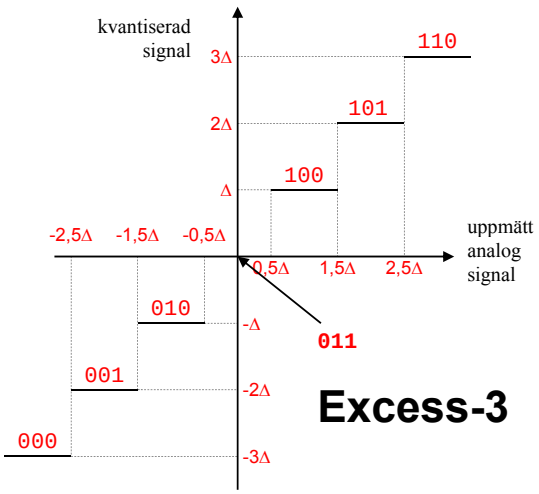
Textsträngen ”Hej” representeras som:

```
1001000  1100101  1101010
   'H'      'e'      'j'
```

Textsträngen ”9756” representeras som:

```
0111001  0110111  0110101  0110110
   '9'      '7'      '5'      '6'
```


Excess–n kod



Används för att koda bipolära storheter, exempelvis att representera ett spänningsintervall

-10 V < analog signal < 10 Volt

Men också som exponent i IEEE-flyttal (beskrivs nedan...)

Flyttal

Exempel:
Omvandla $(122{,}18)_{10}$ till normaliserat tal med flytande bråkpunkt (flyttal).

Lösning:
Vi har tidigare sett att: $(122{,}18)_{10} = (1111010.0010111)_2$
Men detta kan också skrivas:

$(1.1110100010111)_2 \times 2^6$ ← Exponent
Normaliserad mantissa
$1 \leq m < 2$

Teckenbit måste finnas, eftersom både talet och exponenten är bipolära (+/-0)

$+(1.1110100010111)_2 \times 2^{+6}$

Teckenbit:
0 = +
1 = -

Här använder vi excess-n
Kodning
minst 4 bitar behövs här ty
$6 = (110)_2$, dvs tre bitar
Och vi kräver symmetriskt
+/-

tecken

0/1110/1.1110100010111

exponent — Normaliserad mantissa

excess 8

tecken

0/1110/1.1110100010111

exponent Normaliserad mantissa

Egentligen onödigt att lagra denna, ty den är alltid 1...

Vilket leder oss till...

IEEE - Flyttalsstandard

- flyttalsformat
- noggrannhet i resultat från aritmetiska operationer
- omvandling mellan heltal och flyttal
- omvandling till/från andra flyttalsformat
- avrundning
- undantagshantering vid operationer på flyttal, exempelvis division med 0 och resultat som ej kan representeras av flyttalsformatet.

- Standarden definierar fyra olika flyttalsformat:
  - *Single format*, totalt 32 bitar
  - *Double format*, totalt 64 bitar
  - *Single extended format*, antalet bitar är implementationsberoende
  - *Double extended format*, totalt 80 bitar

MSB LSB

S	C	F

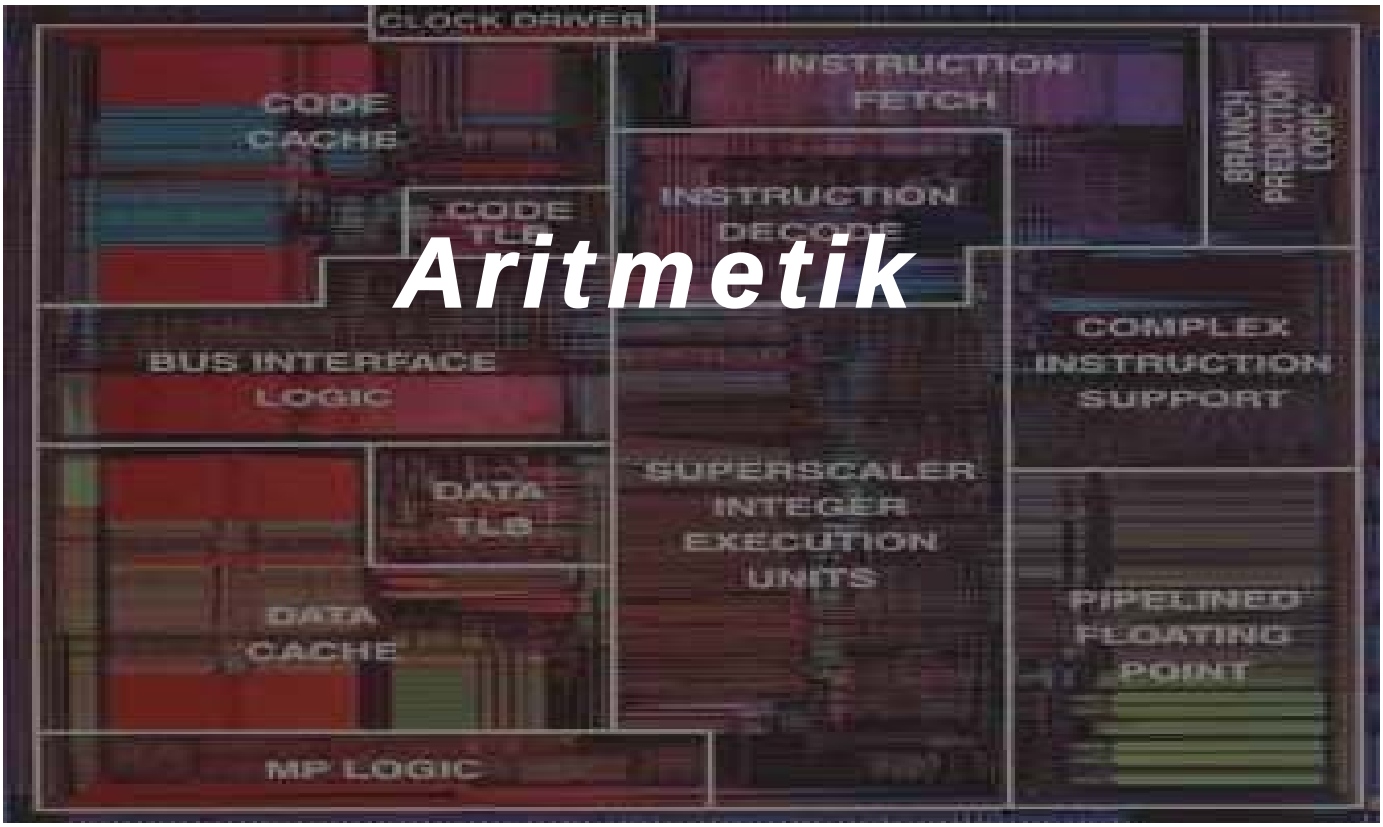
där:

F (*fractional part*) kallas också *signifikand*, är den normaliserade mantissan × 2, dvs den första (implicita) ettan i mantissan utelämnas i representationen och mantissan skiftas ett steg till vänster. På så sätt uppnår vi ytterligare noggrannhet eftersom vi får ytterligare en siffra i det lagrade talet.

C (*karakteristika*) är exponenten uttryckt på *excess(n)* format, *n* beror på vilket av de fyra formaten som avses.

S (*sign*) är teckenbit för F.

Format	**S**	**C**	**F**
Single	1 bit	8 bitar excess(127)	23 bitar
Double	1 bit	11 bitar excess(1023)	52 bitar
Extended	1 bit	15 bitar excess(2047)	64 bitar



Binär addition – ”papper och penna metod”

Exempel: $(5)_{10} + (23)_{10} = ?$

			minnessiffror
		1 1 1	
$(5)_{10}$		101	augend
$(23)_{10}$	+	10111	addend
$(28)_{10}$		11100	summa

Binär multiplikation – ”papper och penna metod”

Exempel: $(25)_{10} \times (11)_{10} = ?$

$(25)_{10}$		11001	multiplikand
$(11)_{10}$	×	1011	multiplikator
		11001	
		11001	
		00000	
	+	11001	
$(275)_{10}$		100010011	produkt

Binär subtraktion – ”papper och penna metod”

Exempel: $(110)_{10} - (43)_{10} = ?$

			minnessiffror
		10 10	
$(110)_{10}$		1101~~1~~~~1~~0	minuend
$(43)_{10}$	-	101011	subtrahend
$(67)_{10}$		1000011	skillnad

Binär division – ”papper och penna metod”

Exempel: $(33)_{10} : (6)_{10} = ?$

```
                0101   kvot
divisor  110 | 100001   dividend
             - 000
               1000
             -  110
               0100
             -  000
               1001
             -  110
                011     rest
```

$(100001)_2:(110)_2 =$
$(0101)_2 + (011)_2:(110)_2$

8-bitars addition

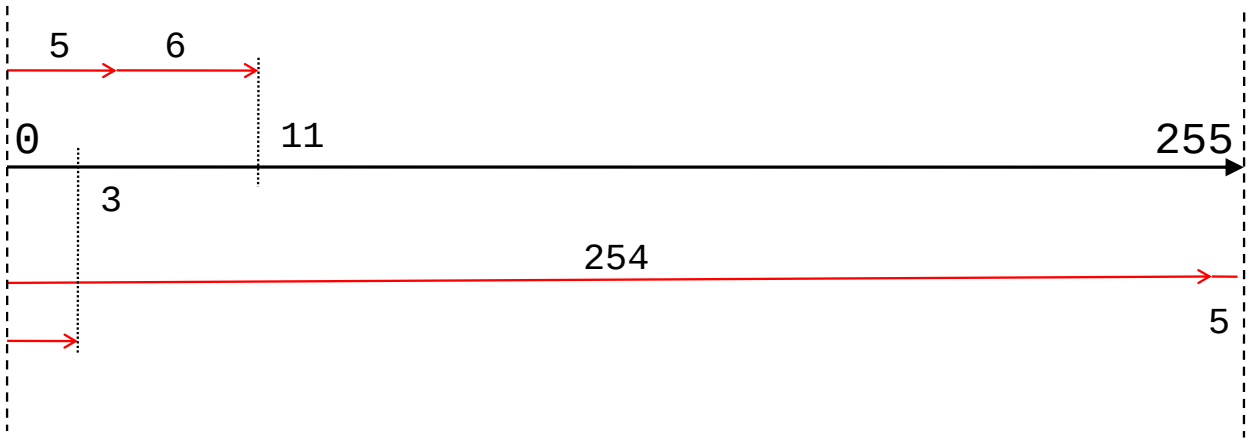
```
                  1
     5      00000101
+    6  +   00000110
-------------------
    11      00001011
```

8 bitar ger talområdet $0..2^8-1 = 0..255$

```
              111111
   254      11111110
+    5  +   00000101
-------------------
   259     100000011
            00000011
```

Spill ! (”Overflow”)

Geometrisk tolkning - tallinje



Vi säger att C-flaggan, genererad från addition av de mest signifikanta bitarna är en ***spillindikator***.

8-bitars subtraktion

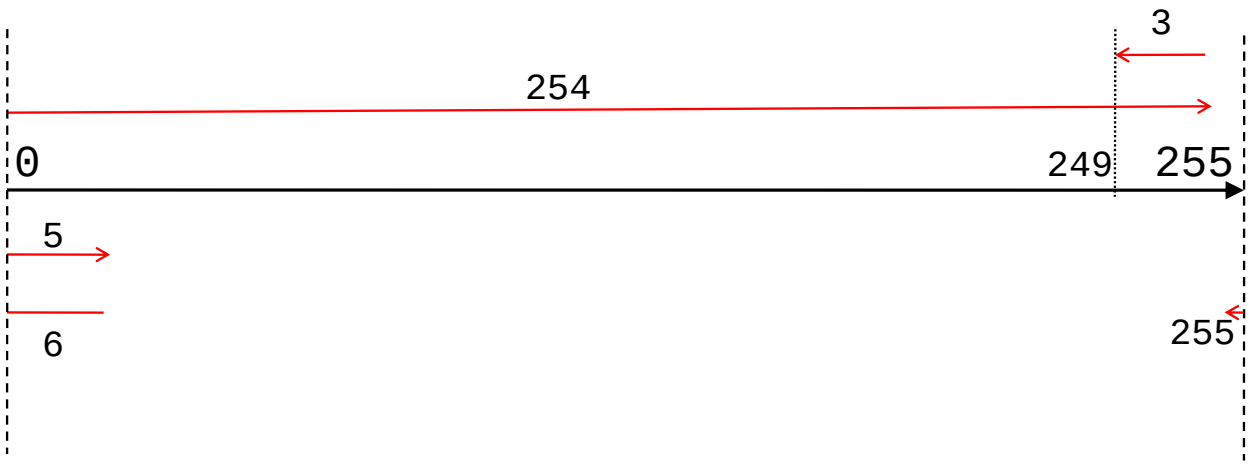
```
                    1
254          11111110
- 5      -   00000101
-------------------------
249          11111001
```

```
                 1
               10 10 10 10 10  10
    5          00000101
- 6      -   00000110
-------------------------
255          11111111
```

För att kunna utföra subtraktionen tvingas vi låna av en tänkt siffra med vikt 2^8.

Spill ! (”Underflow”)

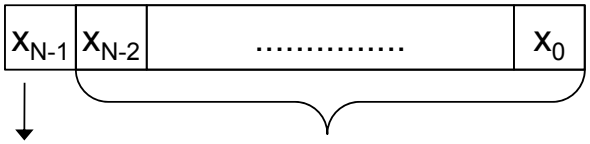
Geometrisk tolkning - tallinje



Den tänkta ”lånebiten” kallar vi ”Borrow”, en ***spillindikator***.

Tal med tecken - Tvåkomplementsform

+/-|X|, N-bitars tal:

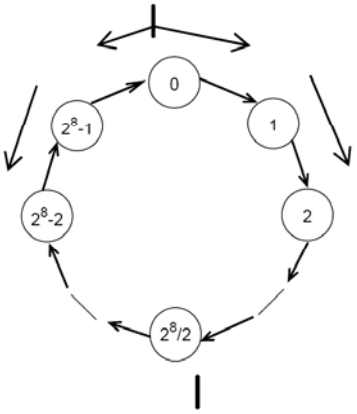


0: X≥0
1: X<0

|X| om X_{N-1} = 0
2^N-|X| om X_{N-1} = 1

Exempel: 8-bitars tal +/- 19:

+19	0	0	0	1	0	0	1	1
-19=[256-19=237]	1	1	1	0	1	1	0	1



Tvåkomplementsform - Metod för teckenbyte

$X+Y = 2^n \Rightarrow$ Y är 2-komplementet till X (n-bitars tal)

För 8-bitars tal således:

$$Y = X_{2k} = 2^8 - X =$$
$$= (2^8-1) - X + 1$$
$$= 11111111 - x_7x_6x_5x_4x_3x_2x_1x_0 + 1$$

Detta kallas 1-komplement (X_{1k}).
Bitvis invertering

Exempel: Bestäm maskintalet på 8 bitars tvåkomplementsform för decimala talet -50

Vi utgår enklast från X=50 (och söker X_{2k})

$(50)_{10}$= X = 00110010

X_{1k} = 11001101

\+ 1

$-(50)_{10} = X_{2k}$ = 11001110

Tvåkomplementsform - addition

Relation A och B, om:	Utförs A+B som:
A,B≥0	A+B
A≥0, B<0,	$A+B_{2k} = [A+(2^N-B)] \pmod{2^N} = A-B = A + (-B)$
A<0, B≥0,	$A_{2k}+B = [(2^N-A)+B] \pmod{2^N} = -A+B = B + (-A)$
A,B<0	$A_{2k}+B_{2k} = [(2^N-A)+(2^N-B)] \pmod{2^N} = -A-B = -(A+B)$

Dvs. Oavsett vilka tecken de ingående talen har så fungerar rättfram binär addition.

Tvåkomplementsform - subtraktion

Vi inser också att en subtraktion kan utföras med hjälp av en adderare ty

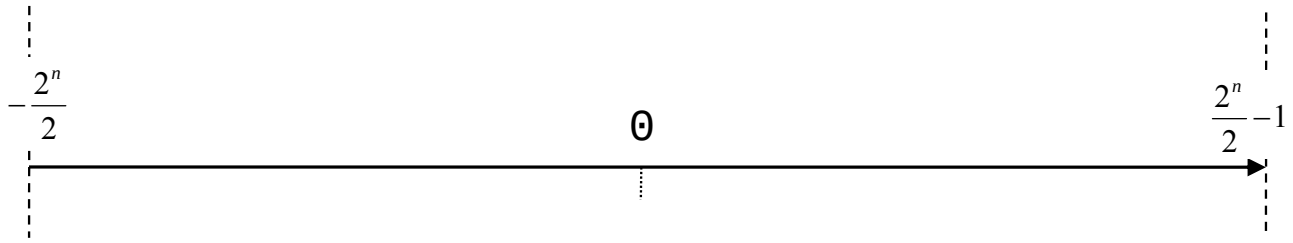
$$A\text{-}B = A+(\text{-}B) \text{ och } -B=B_{2k}= B_{1k}+1$$

Exempel: 6–5=6+(-5)= 00000110+ 11111010+ 1

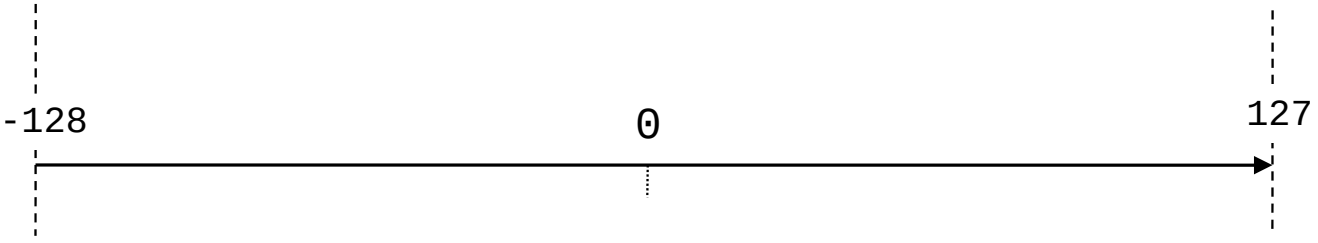
$(5)_{10}$ =
$(00000101)_2$
Dvs 1-komplement:
$(11111010)_{1k}$

$$\begin{array}{r} {\scriptstyle 1\,1\,1\,1\,1\,1\,1\;\;1} \\ 00000110 \\ +\;\;11111010 \\ \hline 00000001 \end{array}$$

Tvåkomplementsform - talområde

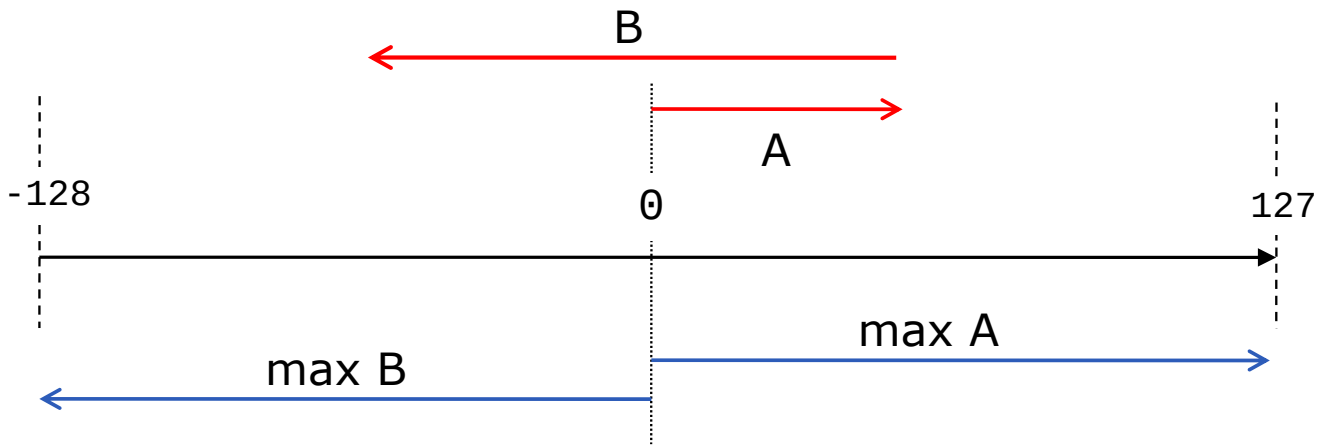


Exempel: 8-bitars tal



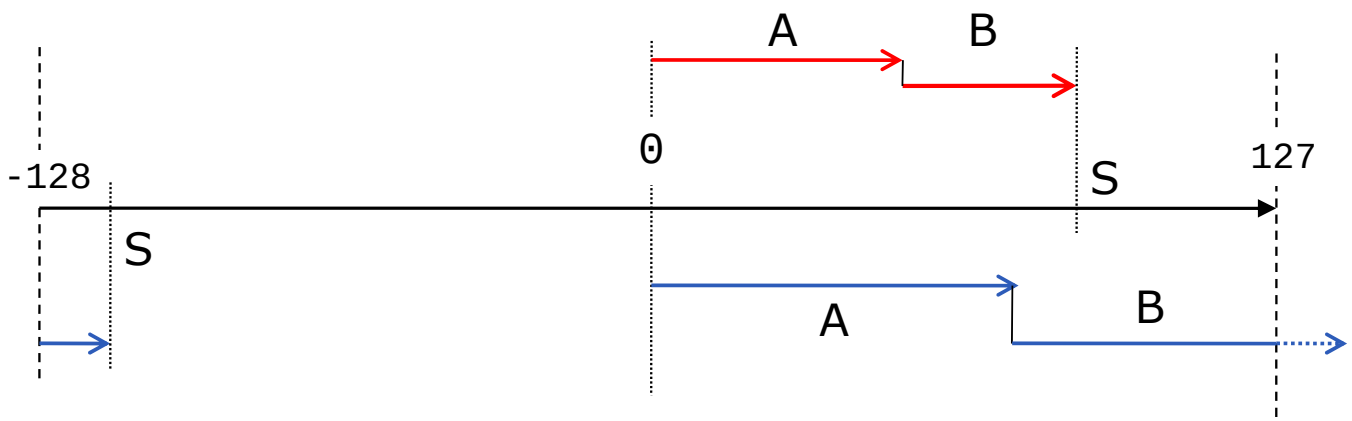
Tvåkomplementsform - spillindikatorer

A + B där A≥0 och B<0



Slutsats: Om A och B har olika tecken vid addition kan 2-komplementspill inte uppträda

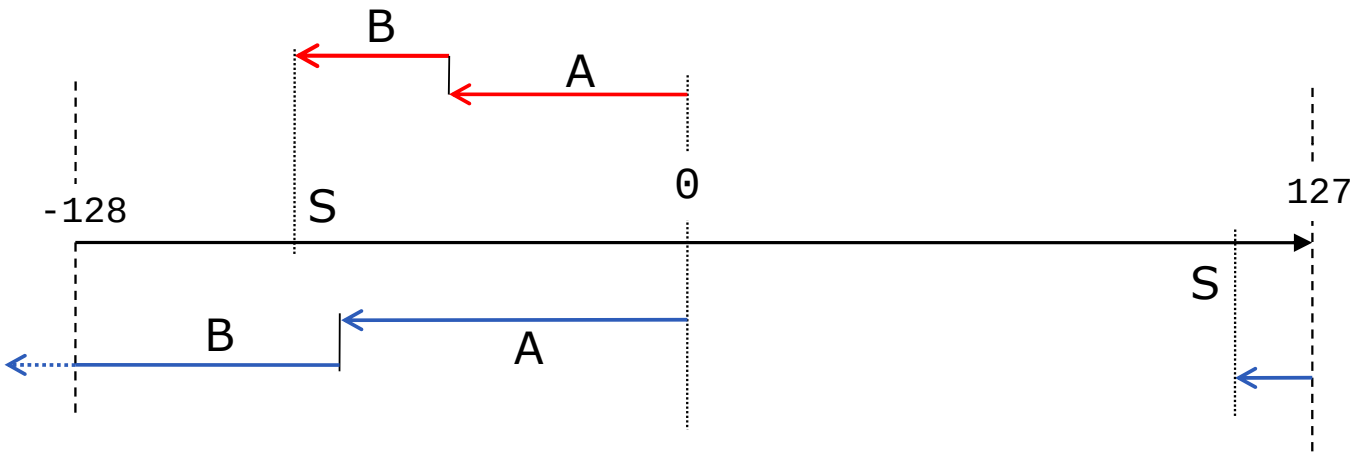
A + B = S, där A≥0 och B ≥ 0



Slutsats:
Om A och B har samma tecken vid addition ***kan*** 2-komplementspill uppträda.
Vi kan konstatera spill genom en teckenöverläggning, dvs:

$$spill = (A \geq 0)\ (B \geq 0)\ (S < 0)$$

A + B = S, där A<0 och B < 0



I detta fall kan vi skriva spillvillkoret:

$$spill = (A<0)\ (B<0)\ (S \geq 0)$$

Vi sammanfattar

Tvåkomplementsform är lämplig representation för binära negativa heltal.

Subtraktion utförs som addition av tvåkomplement

Spillindikator vid addition av naturliga tal [0...N]

”Carry” = c_n

Spillindikator vid subtraktion av naturliga tal [0...N]

”Borrow” = c_n'

Spillindikator vid addition/subtraktion av n-bitars heltal [-M..N] med tvåkomplementsrepresentation:

”Overflow” = $s_{n-1}'\, x_{n-1}\, y_{n-1} + s_{n-1}\, x_{n-1}'\, y_{n-1}'$

Grindar och vippor...

Negation, ”ICKE” → NOT-grind (Inverterare)

satslogik

sanningstabell

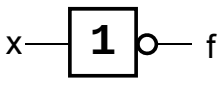
p	¬p
F	S
S	F

Boolesk algebra

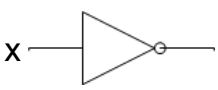
funktionstabell

x	f=x’
0	1
1	0

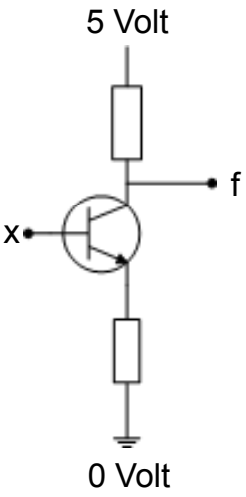
IEC-symbol



Amerikansk symbol



TTL (Transistor-Transistor-Logic)



Observera de alternativa skrivsätten inom Boolesk algebra

$$x' = \overline{x}$$

Disjunktion, ”ELLER” → OR-grind

satslogik

sannings-tabell

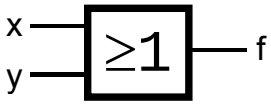
p	q	p∨q
F	F	F
F	S	S
S	F	S
S	S	S

Boolesk algebra

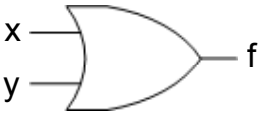
funktions-tabell

x	y	f=x+y
0	0	0
0	1	1
1	0	1
1	1	1

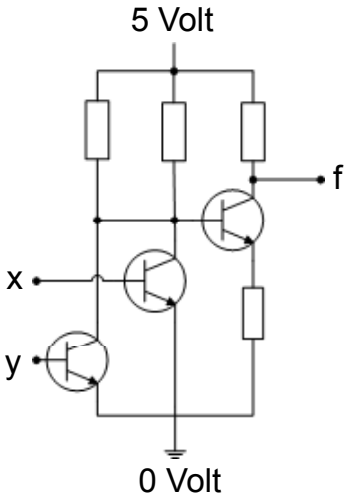
IEC-symbol



Amerikansk symbol



TTL (Transistor-Transistor-Logic)



Konjunktion, ”OCH” → AND-grind

satslogik

sannings-tabell

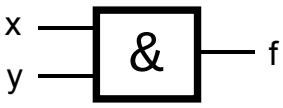
p	q	p∧q
F	F	F
F	S	F
S	F	F
S	S	S

Boolesk algebra

funktions-tabell

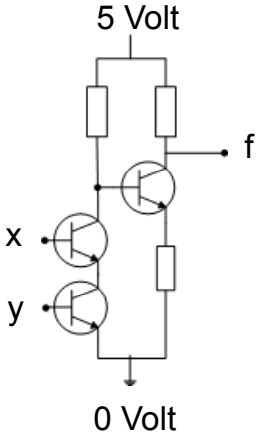
x	y	f=x•y
0	0	0
0	1	0
1	0	0
1	1	1

IEC-symbol

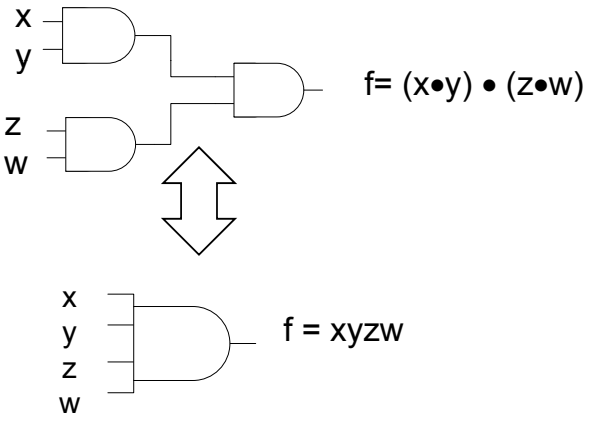


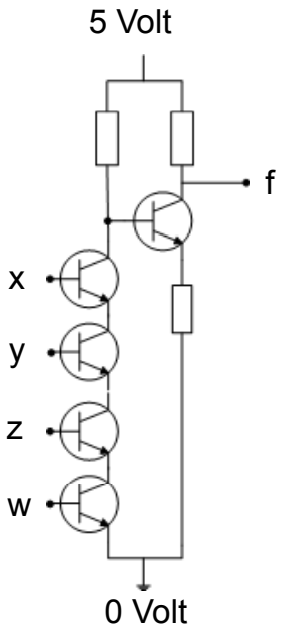
Amerikansk symbol

TTL (Transistor-Transistor-Logic)

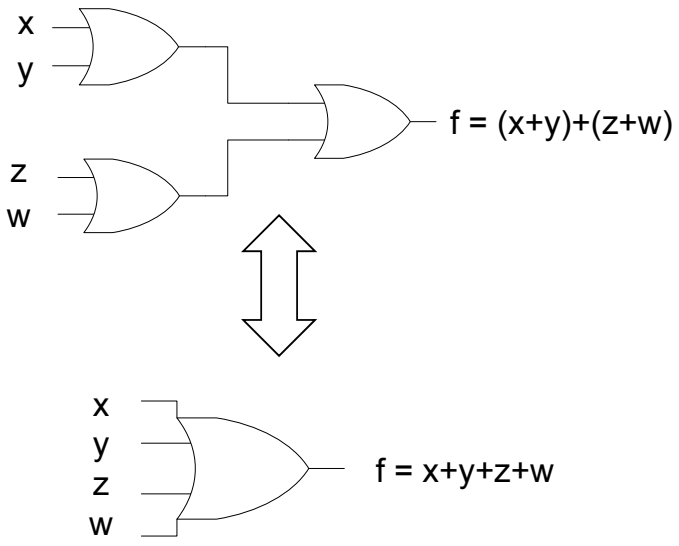


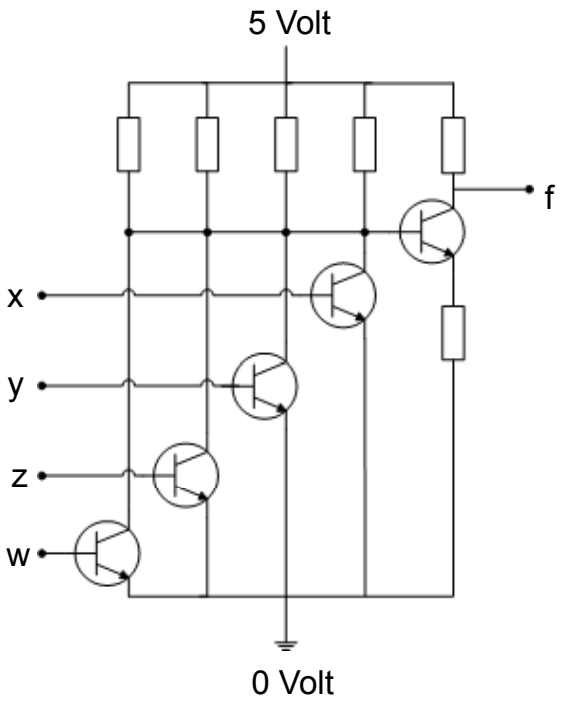
Antalet ingångar kan utökas





Antal ingångar (fan-in), begränsas av använd kretsteknologi.





Negerad konjunktion, ”ICKE-OCH” → NAND-grind

satslogik

sannings-tabell

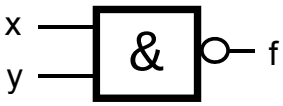
p	q	¬(p∧q)
F	F	S
F	S	S
S	F	S
S	S	F

Boolesk algebra

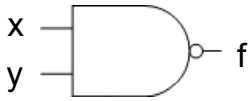
funktions-tabell

x	y	f=(x•y)’
0	0	1
0	1	1
1	0	1
1	1	0

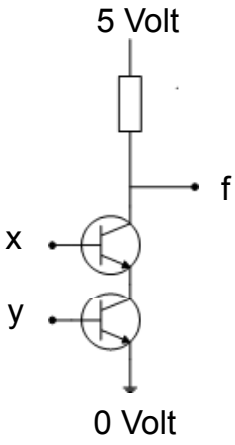
IEC-symbol



Amerikansk symbol



TTL (Transistor-Transistor-Logic)



Negerad disjunktion, ”ICKE-ELLER” → NOR-grind

satslogik

sannings-tabell

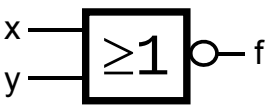
p	q	¬(p∨q)
F	F	S
F	S	F
S	F	F
S	S	F

Boolesk algebra

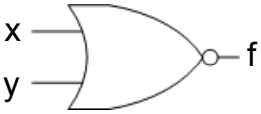
funktions-tabell

x	y	f=(x+y)’
0	0	1
0	1	0
1	0	0
1	1	0

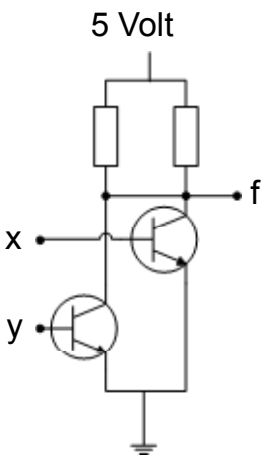
IEC-symbol



Amerikansk symbol



TTL (Transistor-Transistor-Logic)



(NOT) Exkluderande ELLER, (ICKE) XOR-grind

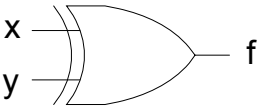
Definition:
x⊕y = x'y+xy'

IEC-symbol

funktionstabell

x	y	f=x⊕y
0	0	0
0	1	1
1	0	1
1	1	0

Amerikansk symbol

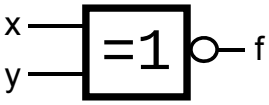


(x⊕y)' = x'y'+xy

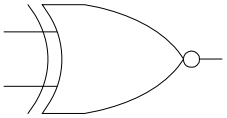
funktionstabell

x	y	f=(x⊕y)'
0	0	1
0	1	0
1	0	0
1	1	1

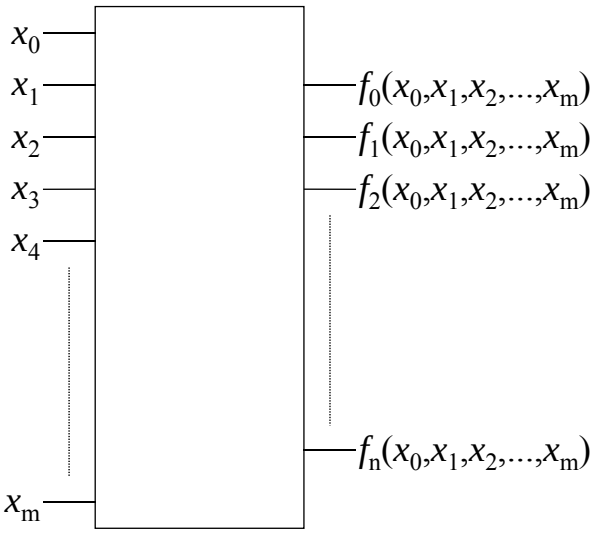
IEC-symbol



Amerikansk symbol



Kombinatoriska nät



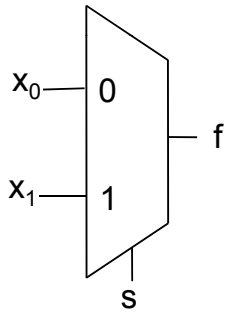
Ett kombinatoriskt nät har fixt antal ingångar (*m*-1) och fixt antal utgångar (*n*-1).

Varje utgång har i varje ögonblick det värde som entydigt bestäms av insignalerna.

- Vi introducerar komponenter som byggblock i datorn.

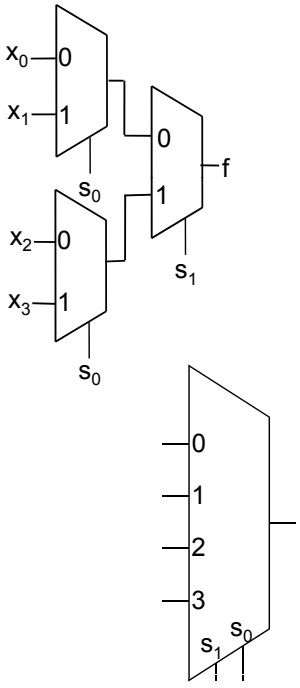
Väljare (Multiplexer)

s	x_0	x_1	f
0	0	0	**0**
0	0	1	**0**
0	1	0	**1**
0	1	1	**1**
1	0	0	**0**
1	0	1	**1**
1	1	0	**0**
1	1	1	**1**

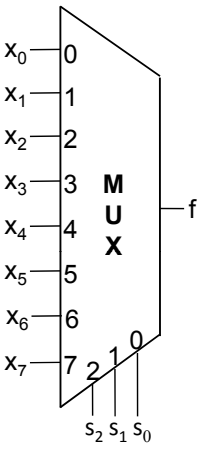


$$f = \bar{s}x_0 + sx_1$$

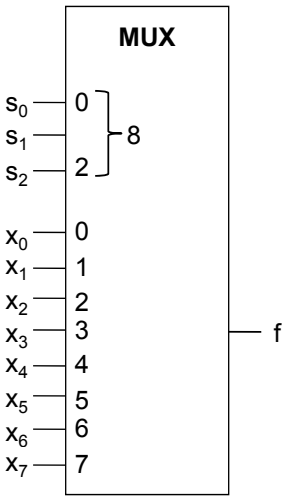
1 av 2 väljare



1 av 4 väljare



1 av 8 väljare



Binär addition

Uppställning för addition av två st. n-bitars tal:

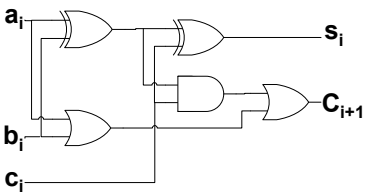
	c_n	c_{n-1}	c_{n-2}	...	c_i	...	c_1	
		a_{n-1}	a_{n-2}	...	a_i	...	a_1	a_0
+		b_{n-1}	b_{n-2}	...	b_i	...	b_1	b_0
	s_n	s_{n-1}	s_{n-2}	...	s_i	...	s_1	s_0

vi koncentrerar oss på addition
av *en* bit, säg bit *i*.

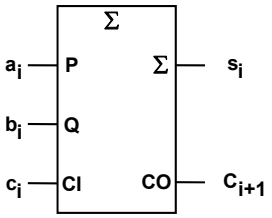
c_{i+1}	c_i
	x_i
+	y_i
=	s_i

a_i	b_i	c_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

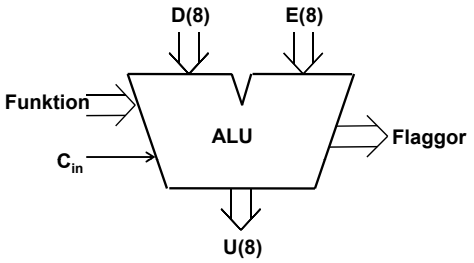
Heladderaren,
ett kombinatoriskt nät
med 3 insignaler och
2 utsignaler



Symbol för heladderare



ALU-funktioner

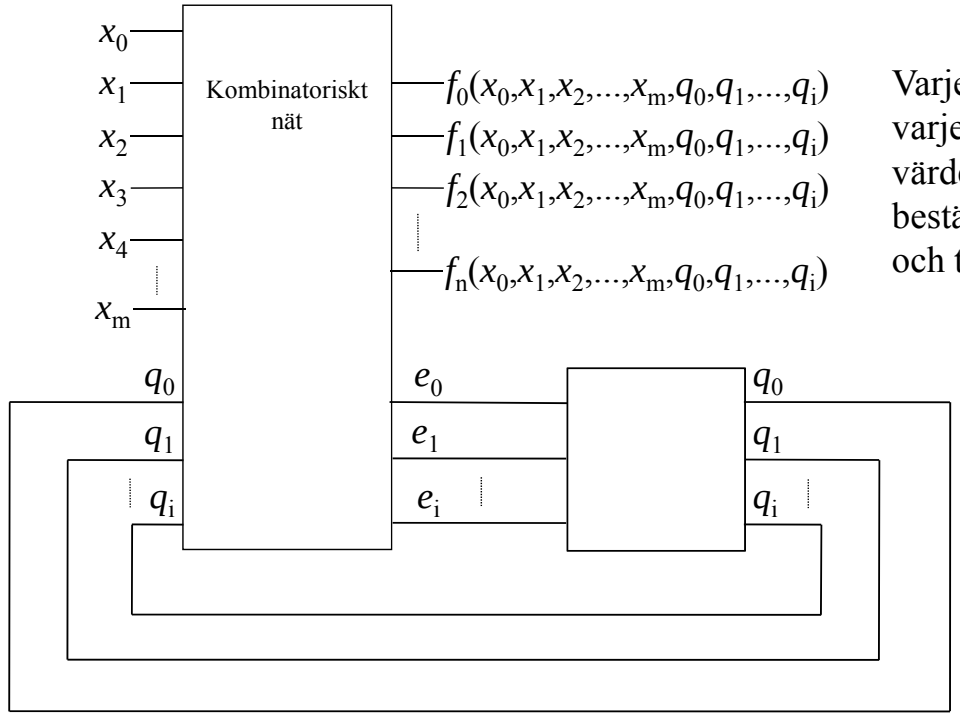


ALU:ns logik- och aritmetikoperationer på indata D och E definieras av ingångarna Funktion (F) och C_{in} enligt tabellen.
F = (f_3, f_2, f_1, f_0).

"+" och "-" avser aritmetiska operationer.
D_{1k} menas att samtliga bitar i D inverteras.

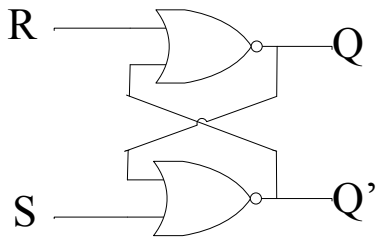
f_3 f_2 f_1 f_0	U = f(D,E,F,C_{in})	
	Operation	Resultat
0 0 0 0	Bitvis nollställning	0
0 0 0 1		D
0 0 1 0		E
0 0 1 1	Bitvis invertering	D_{1k}
0 1 0 0	Bitvis invertering	E_{1k}
0 1 0 1	Bitvis OR	D OR E
0 1 1 0	Bitvis AND	D AND E
0 1 1 1	Bitvis XOR	D XOR E
1 0 0 0	D + 0 + C_{in}	D + C_{in}
1 0 0 1	D + FFH + C_{in}	D – 1 + C_{in}
1 0 1 0		D + E + C_{in}
1 0 1 1	D + D + C_{in}	2D + C_{in}
1 1 0 0	D +E_{1k} + C_{in}	D – E – 1 + C_{in}
1 1 0 1	Bitvis nollställning	0
1 1 1 0	Bitvis nollställning	0
1 1 1 1	Bitvis ettställning	FFH

Sekvensnät

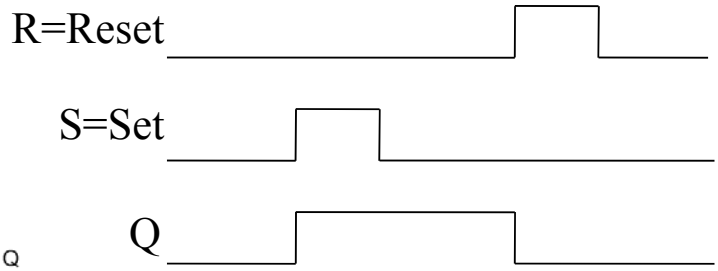


Varje utgång har i varje ögonblick det värde som entydigt bestäms av insignaler och tillstånd.

Latch (låskrets)



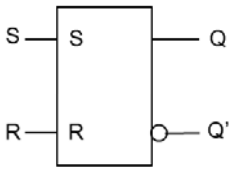
Q kan ettställas eller nollställas för att därefter behålla värdet.
Kopplingen är ett ”minneselement” och kallas SR-latch.



Funktionstabell

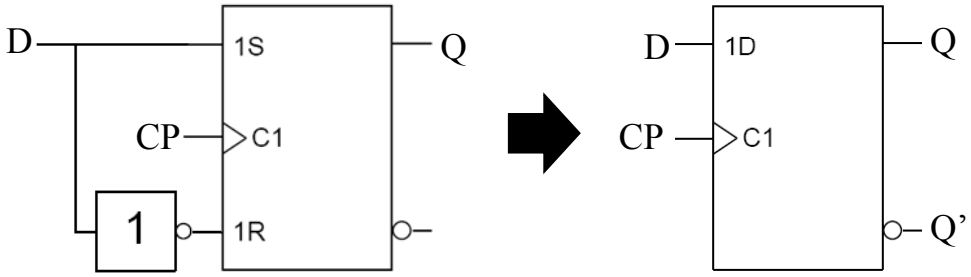
S	R	Q^+
0	0	**Q**
0	1	**0**
1	0	**1**
1	1	*

Symbol



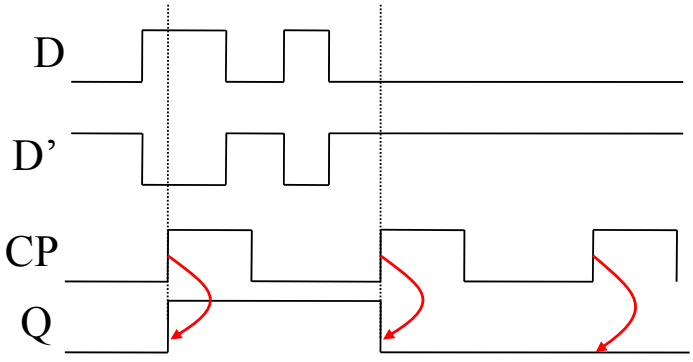
Asynkront minneselement

Flanktriggad D-vippa



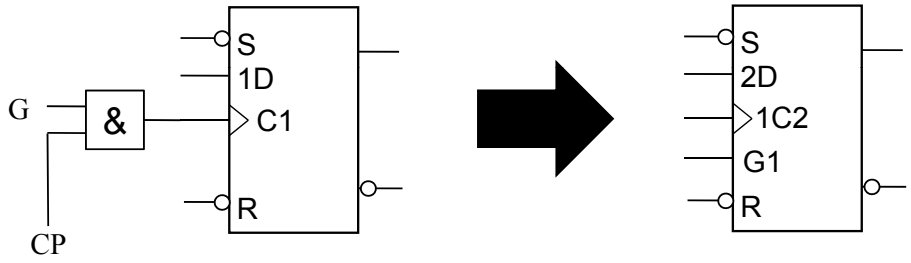
Funktions-tabell

D	Q^+
0	0
1	1



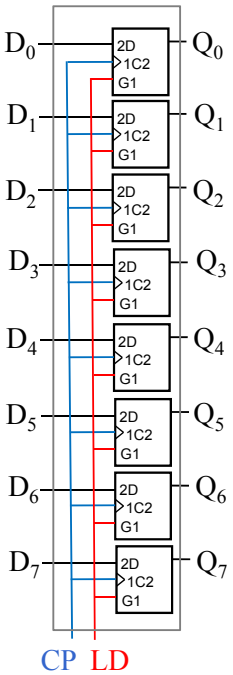
Synkront minneselement

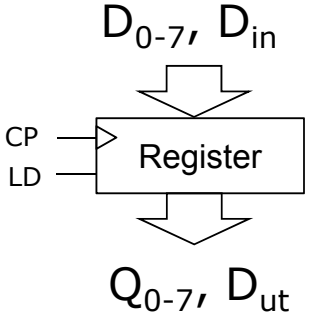
D-vippa med ”Load Enable”

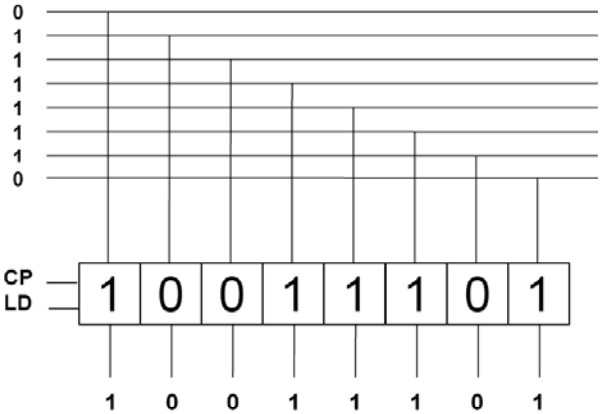


Styrsignalen G (”Gate”) kan användas för att ”strypa” klockpulsen.
Då G=0 behåller vippan sitt värde oavsett vad som finns på D-ingången

Laddbart register

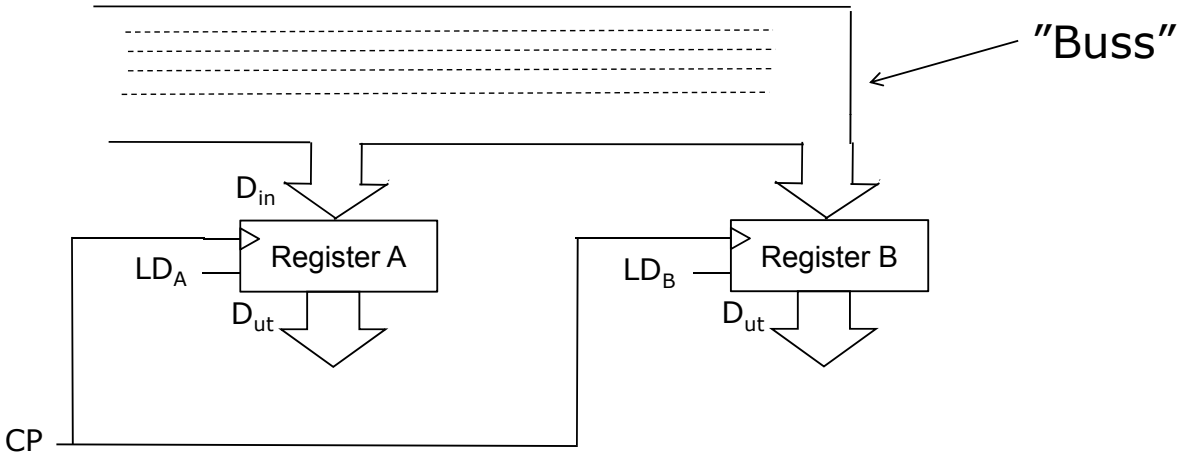


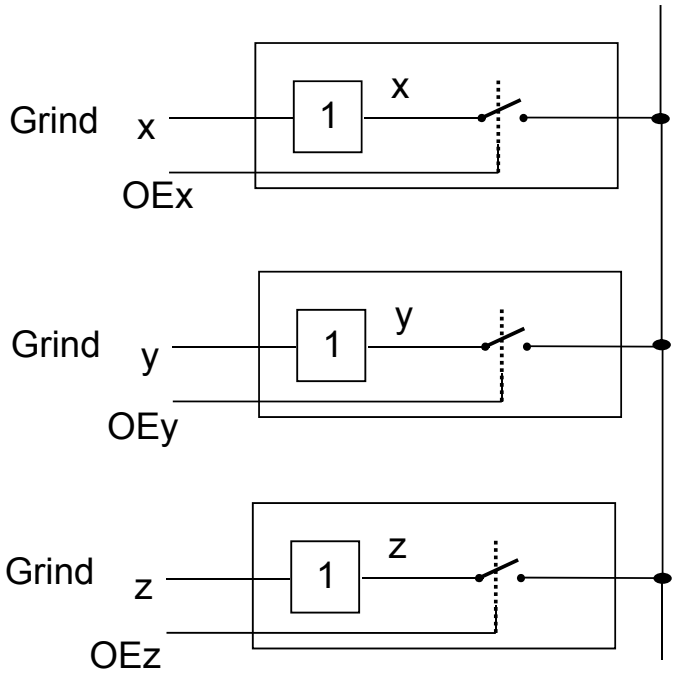




Registrets innehåll påverkas av ingångarnas nivåer
först vid en klockpuls OCH om LD-signalen är aktiv.

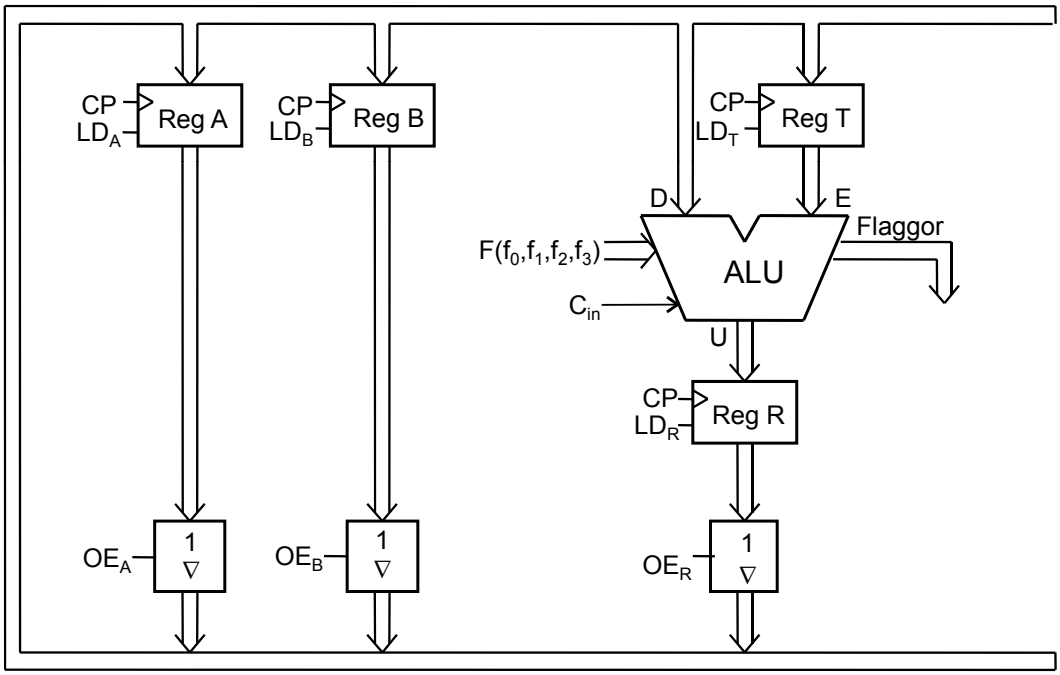
”Dataväg” – sammankopplade register

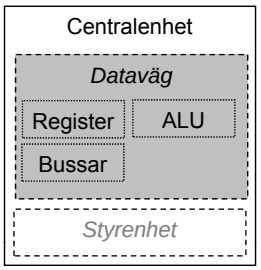




Högst ***en*** OE-signal aktiv åt gången...

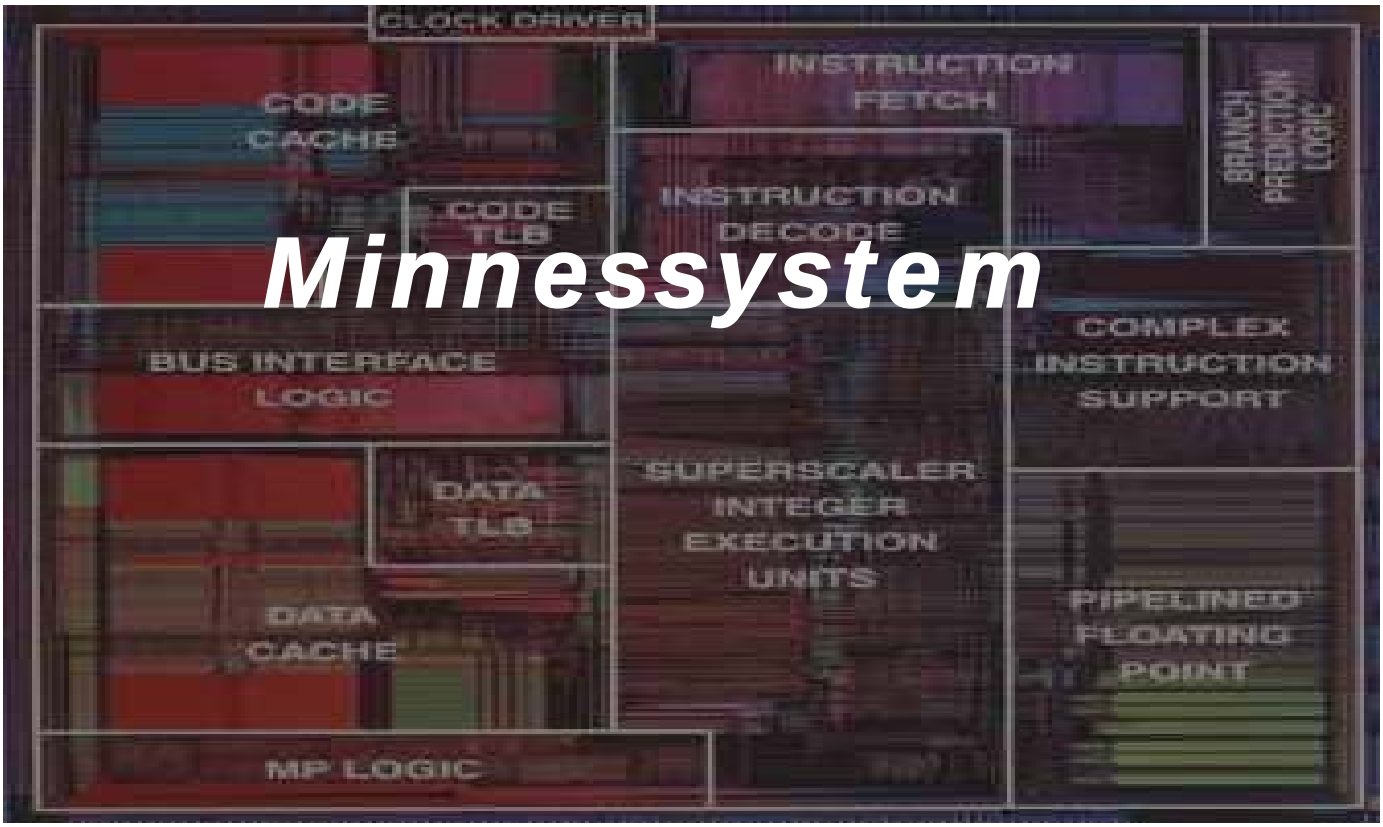
Enkel dataväg med ALU





Styrsignaler:

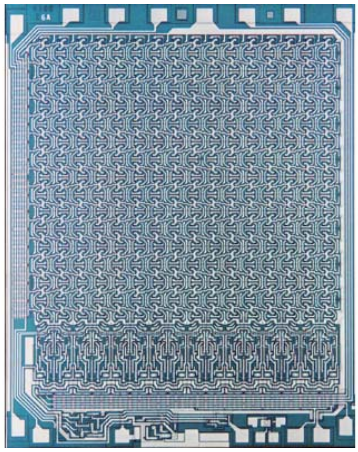
- LD_A
- LD_B
- LD_T
- LD_R
- OE_A
- OE_B
- OE_R
- $F(f_0,f_1,f_2,f_3)$
- C_{in}



Minnessystem

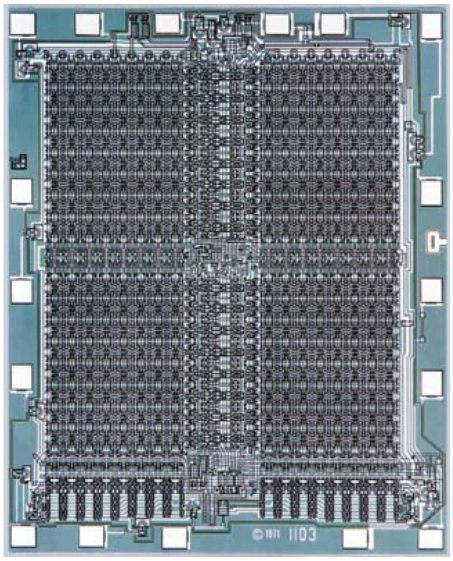
- Primärminnen
  - S-RAM, D-RAM, ROM, EPROM, FLASH…
- Sekundärminnen
  - Hårddisk
  - Flexskiveenheten
- Tertiärminnen
  - Bandstationer
  - Optiska lagringsmedia

Statiskt RAM (1966)



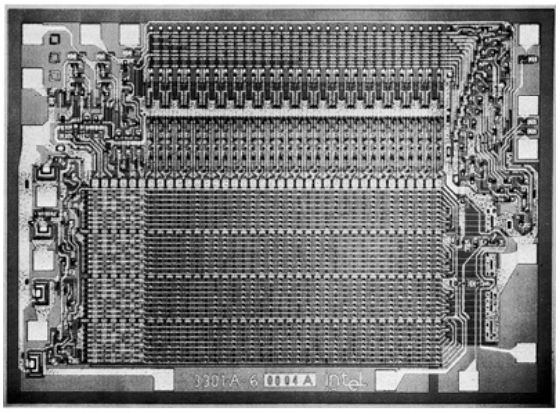
256-bit TTL RAM
(Fairchild)

Dynamiskt RAM (DRAM) 1970



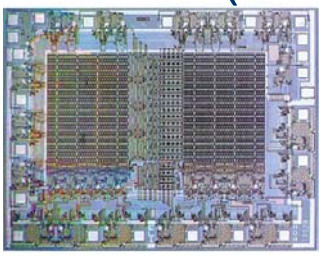
Intel i1103
1024-bit
Dynamiskt
RAM

ROM (1965)

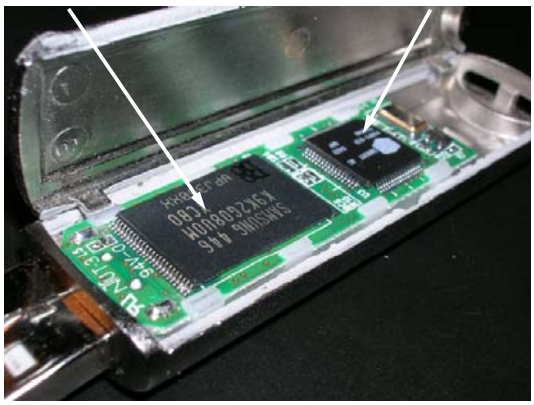


Intel 3301, 1024-bit ROM

EPROM (1971)



FLASH 1988

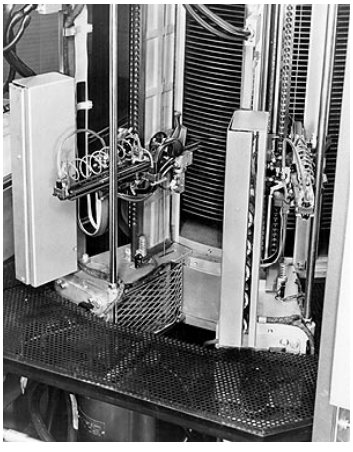


”BLOCK”-minnes åtkomst

CHALMERS
Chalmers University of Technology
Digital och Datorteknik

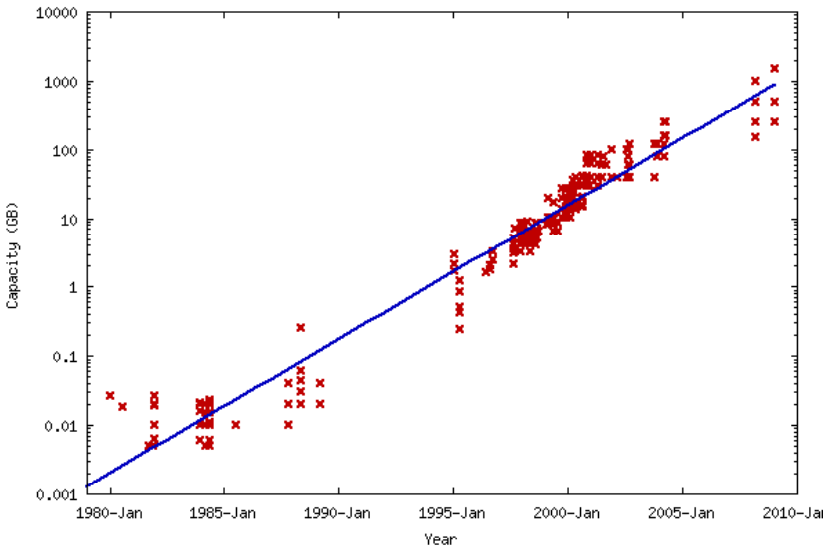
Hårddisken

IBM 350 (1956)

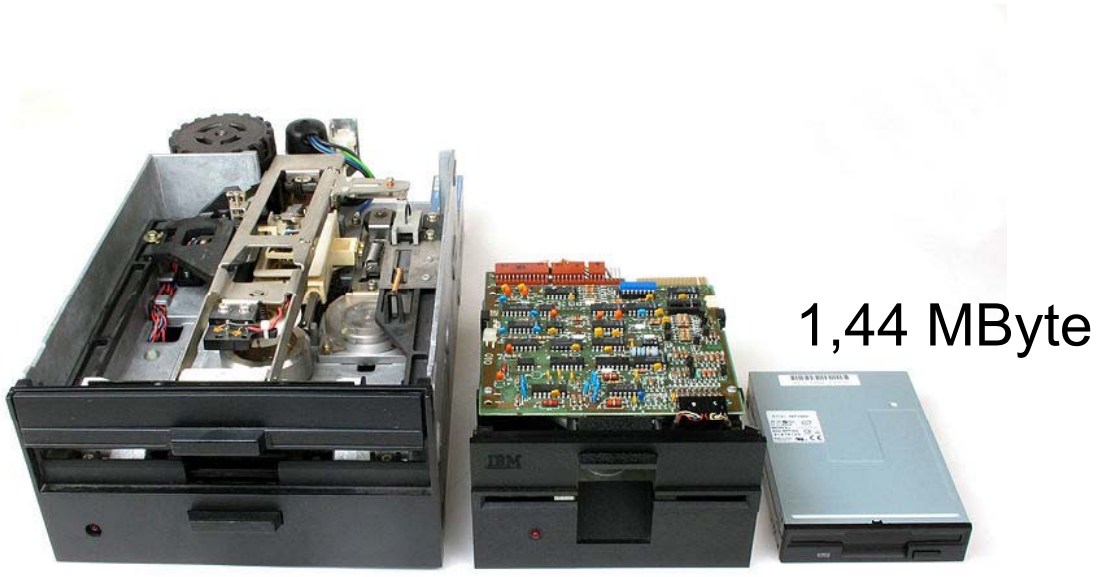


Kapacitet 5-20 MByte

Utveckling av lagringskapacitet



Flexskivan (1967)



Compact Disc(1965)

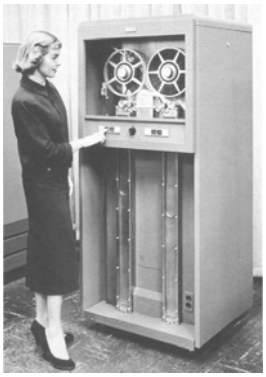
650-900 MByte

Digital Versatile Disc(1993)

4,7-17 GByte

Bandstationer

IBM 701 (1949)



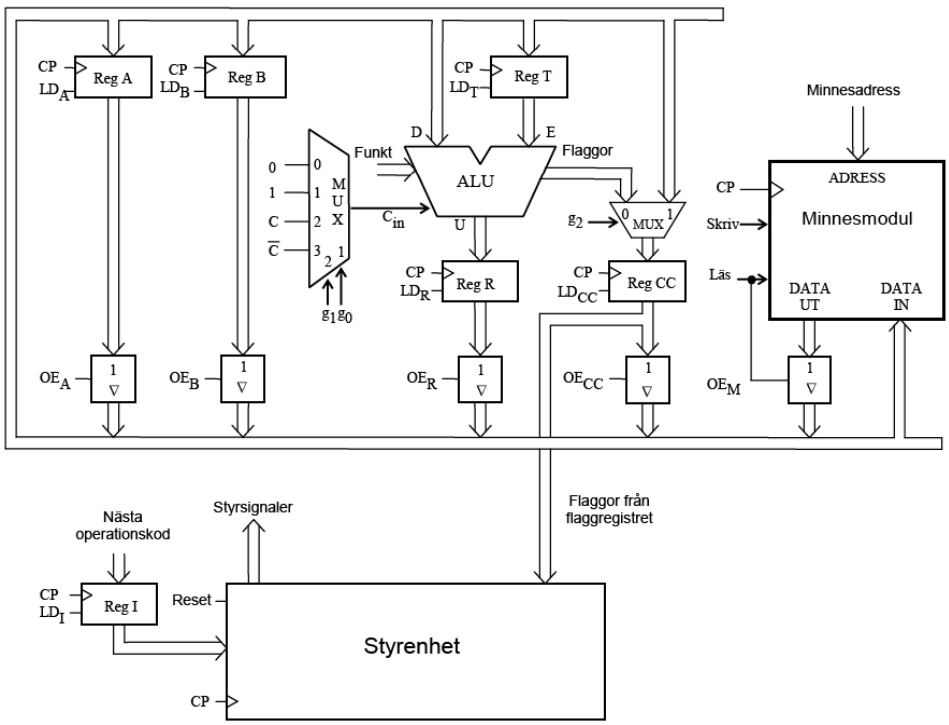
IBM 726 (1952)
2MB

IBM 3420 (1970)

IBM MP350 (1996)
300GB/kasett

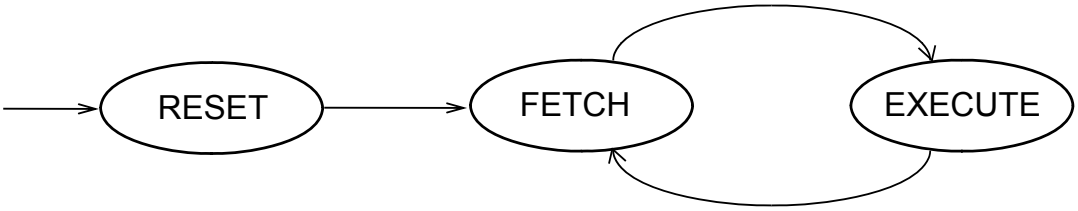
Centralenhetens arbetssätt

Centralenhet och minne



Centralenhetens arbetssätt

En centralenhet har minst tre ”faser”:

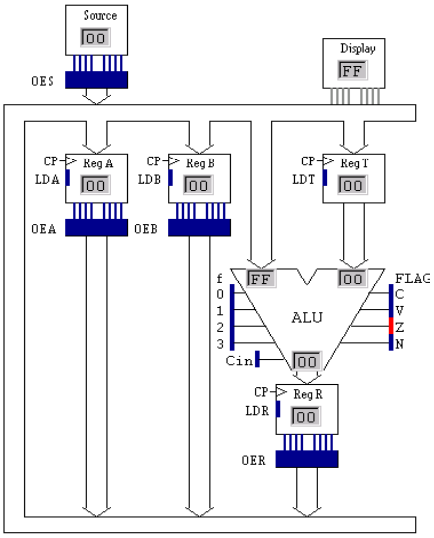


Återställning vid spännings-tillslag eller av yttre signal

Hämta maskininstruktion från minne

Utför maskininstruktion

Exempel: B+1→B (INCrement B)

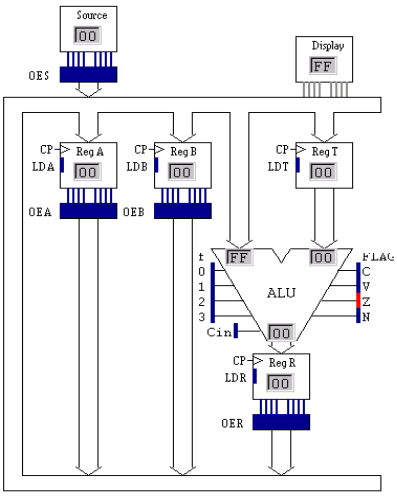


f3	f2	f1	f0	Operation	Resultat
0	0	0	0	Bitvis nollställning	0
0	0	0	1		D
0	0	1	0		E
0	0	1	1	Bitvis invertering	D_{1k}
0	1	0	0	Bitvis invertering	E_{1k}
0	1	0	1	Bitvis OR	D OR E
0	1	1	0	Bitvis AND	D AND E
0	1	1	1	Bitvis XOR	D XOR E
1	**0**	**0**	**0**	**D + 0 + Cin**	**D + Cin**
1	0	0	1	D + FF_{16} + Cin	D – 1 + Cin
1	0	1	0		D + E + Cin
1	0	1	1	D + D + Cin	2D + Cin
1	1	0	0	D + E_{1k} + Cin	D – E – 1 + Cin
1	1	0	1	Bitvis nollställning	0
1	1	1	0	Bitvis nollställning	0
1	1	1	1	Bitvis ettställning	FF_{16}

U = f(D,E,F,Cin)

Observera att en given operation som regel kan utföras på flera olika sätt.
Vi eftersträvar vanligtvis det effektivaste (minst klockcykler).

I RTN-beskrivningen anger vi, klockpuls för klockpuls, hur datavägen används.



Steg 1:

B→D

Cin = 1

F=1,0,0,0

U→R

Resultatet B+1 finns nu i register R.

Eftersom bussen är upptagen krävs ytterligare steg för att återföra resultatet till B

Steg 2:

R→B

RTN-beskrivning:

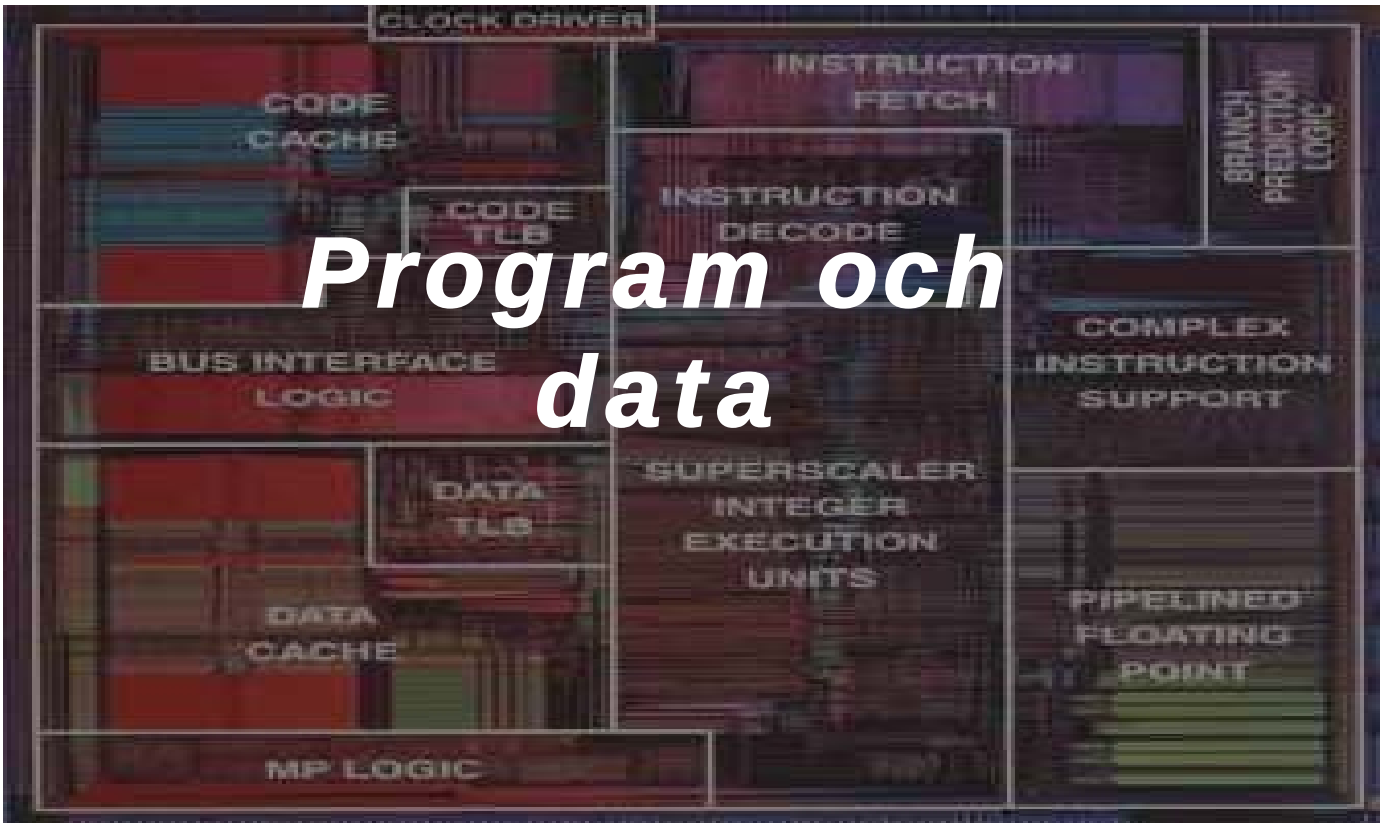
CP1: B+1 → R

CP2: R→B

CP3:

CP4:

CP5:



Program och data

Program och minne

- **John Louis Von Neumann** (1903-1957)

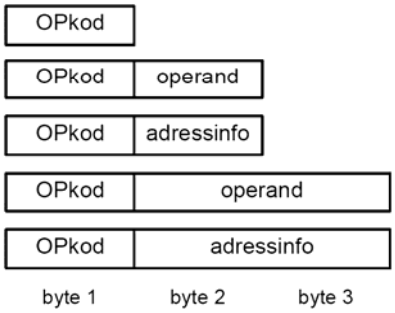
”Det lagrade programmets princip”, dvs program och data i samma minne.

adress hex	innehåll hex	operation
07	01	
08	0B	- load the A-register with the contents
09	91	of this address (91H)
0A	28	- add to the A-register the contents of
0B	92	this address (92H)
0C	61	- branch, if carry=1, to the address =
0D	07	this number (07H) + address of next OPcode (0EH) (= 15H)
0E	13	- store the contents of the A-register on
0F	94	this address (94H)
10	47	- clear the A-register
11	13	- store the contents of acc A (00H) on
12	93	this address (93H)
13	59	- jump to
14	1B	this address (1BH)
15	13	- store the contents of the A-register on
16	94	this address (94H)
17	0F	- load the A-register with
18	01	this number (01H)
19	13	- store contents of the A-register (01H)
1A	93	on this address (93H)
1B	0B	

Maskinprogram i minnet — *Tillhörande assemblerprogram*

0C	10	LDAB	#$23	*Instruktion*
0D	23			
0E	29	ADDB	$F3	*Adress*
0F	F3			
10	02	TFR	B,A	
11	4F	CMPB	#$03	*Data*
12	03			
13	61	BLO	$29	
14	13			

Instruktionsformat



Exempelvis:

ADDB	**Adr**	OP-kod	Adr
LDAB	**#data**	OP-kod	data
TFR	**B,A**	OP-kod	

Adress — *Maskinprogram*

Adress		Maskinprogram
0C	10	00010001_2
0D	23	00100011_2
0E	29	00101001_2
0F	F3	11110011_2
10	02	00000010_2
11	4F	01001111_2
12	03	00000011_2
13	61	01100001_2
14	13	00010011_2

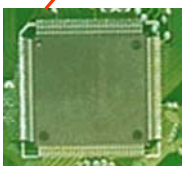
”Byte-wide” 8 bitar data på varje adress

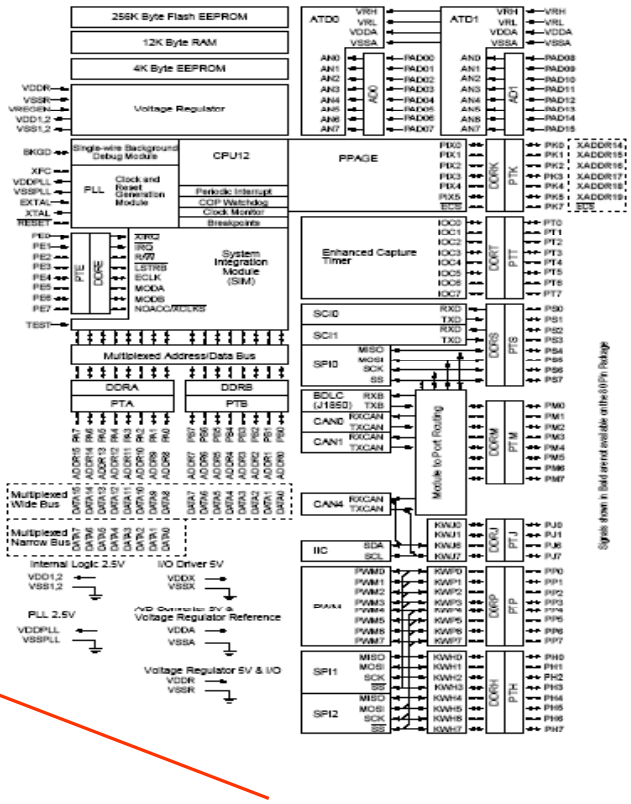
Assemblerprogram

10 23	⟷	LDAB	#$23
29 F3	⟷	ADDB	$F3
02	⟷	TFR	B,A
4F 03	⟷	CMPB	#$03
61 13	⟷	BLO	$29

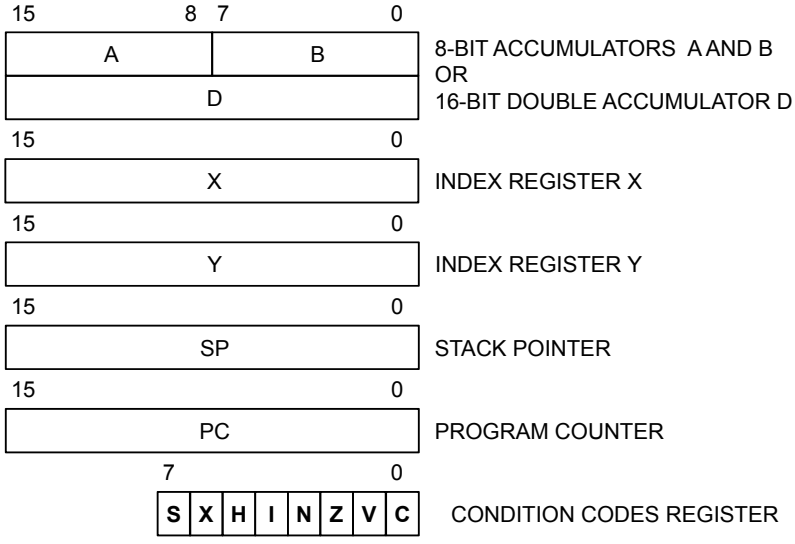
Instruktion ”mnemonic” — *operand-information*

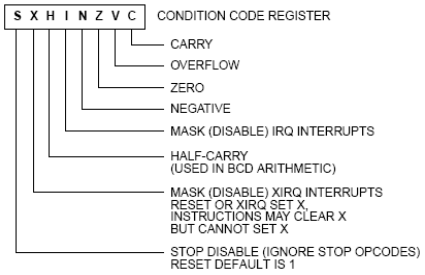
HCS12DG256, blockdiagram





Registeruppsättning CPU12





Instruktionsgrupper

LOAD-instruktioner

Mnemonic	Funktion	Operation
LDAA	Load A	(M)→A
LDAB	Load B	(M)→B
LDD	Load D	$(M:M+1)_1 \rightarrow A:B$
LDS	Load SP	$(M:M+1)_1 \rightarrow SP_H:SP_L$
LDX	Load index register X	$(M:M+1)_1 \rightarrow X_H:X_L$
LDY	Load index register Y	$(M:M+1)_1 \rightarrow Y_H:Y_L$
LEAS	Load effective address into SP	Effective address→SP
LEAX	Load effective address into X	Effective address→X
LEAY	Load effective address into Y	Effective address→Y

STORE-instruktioner

Mnemonic	Funktion	Operation
STAA	Store A	(A)→M
STAB	Store B	(B)→M
STD	Store D	(A)→M, (B)→M+1
STS	Store SP	$SP_H:SP_L \rightarrow M:M+1$
STX	Store X	$X_H:X_L \rightarrow M:M+1$
STY	Store Y	$Y_H:Y_L \rightarrow M:M+1$

MOVE-instruktioner

Mnemonic	Funktion	Operation
MOVB	Move byte (8 bitar)	$(M_1) \rightarrow M_2$
MOVW	Move word (8 bitar)	$(M:M+1)_1 \rightarrow M:M+1_2$

EXEMPEL – ”memcopy0(from , to, size)”

Kan (informellt) kodas...

```
memcopy0:           LDAB   ”size”
                    LDX    ”from”
                    LDY    ”to”

memcopy0_loop:      TSTB
                    BEQ    memcopy0_end
                    LDAA   1,X+
                    STAA   1,Y+
                    DECB
                    BRA    memcopy0_loop

memcopy0_end:       RTS
```