LEU431

Digital- och datorteknik

Diverse kompletterande material

Diverse kompletterande material (pdf):

- Positionssystem.
- Samband mellan binära och hexadecimal tal.
- Exempel på excess-8 kod.
- Grafiska symboler för de grundläggande logikoperationerna. (Tabell 3.11)
- Viktiga satser i boolesk algebra.
- Booleska räkneregler och motsvarande grindnät.
- Övningsexempel på minimering med karnaughdiagram.
- Realisering av funktion med väljare, exempel.
- Uppgifter för Demo1.
- Representation av heltal med ett begränsat antal sifferpositioner.
- Decimal addition för heltal utan tecken.
- Binär addition för heltal utan tecken.
- Decimal subtraktion för heltal utan tecken (10-komplementaritmetik).
- Binär subtraktion för heltal utan tecken (2-komplementaritmetik).
- 10-komplementaritmetik och 10-komplementrepresentation.
- 2-komplementaritmetik och 2-komplementrepresentation.
- Flaggor som används vid aritmetiska operationer.
- Logikkrets för addition av binära tal (Heladderare).
- Aritmetik- logikenhet (ALU) för FLEX.
- Exempel på D-vippa med grindad laddingång.
- Läs- och skrivbart minne.
- FLEX- eller FLISP-datorn, utifrån sett.
- FLEX-datorn, programmerarens bild.
- FLEX-datorn, detaljbild.
- FLEX Fast styrenhet, kopplingsblankett.
- FLIS-datorn, programmerarens bild.
- FLIS-datorn, detaljbild.
- FLIS-processorns ALU.
- Implementering av ovillkorliga och villkorliga "branch"-instruktioner för FLISP.
- FLIS-dator med I/O-portar.
- Assemblatordirektiv (pseudoinstruktioner).
- Simulatorns I/O-enheter.
- Enkel styrning av borrmaskin.
- Hur FLIS-processorn läser och skriver i minnet.
- Krets för detektering av läsning på en viss minnesadress.
- Yttre signaler som påverkar exekveringen hos FLIS-processorn.
- Kapitel 13 ur "Grundläggande digital- och datorteknik, del 2".
- Exempel på fullständig och ofullständig adressavkodning.
- Flyttal komplettering.
- Principen för multiplikation och division av binära heltal utan tecken.

Talsystem

Positionssystem

Vårt vanliga decimala talsystem är ett positionssystem. Siffrornas vikt bestäms av deras position i talet.

Ex. Talet 5768,29

Position: 3 2 1 0 -1 -2

Talvärde: $5\ 7\ 6\ 8,\ 2\ \ 9 = 5\cdot10^{3} + 7\cdot10^{2} + 6\cdot10^{1} + 8\cdot10^{0} + 2\cdot10^{-1} + 9\cdot10^{-2}$

Ett positionssystem har en bas. Basen är förhållandet (kvoten) mellan vikten hos den vänstra och den högra av två närliggande sifferpositioner. Basen är också lika med antalet olika siffersymboler (siffror) i talsystemet.

Siffersymbolerna är normalt de som används i det decimala talsystemet dvs. 0, 1, …, basen-1. Om de decimala siffrorna inte räcker till kompletterar man normalt med stora bokstäver från alfabetets början.

Decimala talsystemet har basen 10, som också är antalet olika siffersymboler.

Man kan göra positionssystem för alla heltalsbaser som är större än eller lika med 2.

Vanliga talsystem:

Bas	Talsystem	Siffror
2	Binära	0, 1
(8	Oktala	0, 1, 2, …, 7)
10	Decimala	0, 1, 2, …, 9
16	Hexadecimala	0, 1, 2, …, 9, A, B, C, D, E, F

Positionssystemen fungerar som vägmätaren i en gammal bil. Basen för det aktuella positionssystemet är antalet siffror på varje sifferhjul.

Då bilen körs så vrids sifferhjulet längst till höger fram en siffra för varje kilometer. När siffran 0 kommer fram en gång per varv på ett sifferhjul så vrids sifferhjulet till vänster fram en siffra. Samma princip gäller för alla sifferhjul i vägmätaren.

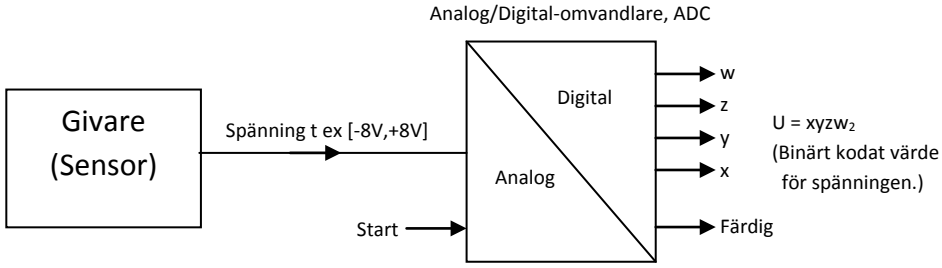
Ex.

För en decimal vägmätare med 5 siffror är lägsta värdet givetvis 00000 och det högsta värdet 99999 = 10^5-1.
Talintervallet som kan visas är alltså $[0, 99999_{10}] = [0, 10^5\text{-}1]$.
Efter det högsta värdet varvar vägmätaren, dvs. den börjar om på 0. Antalet olika tal som vägmätaren kan visa är $10^5 = 100000$.

För en binär vägmätare med 8 siffror är lägsta värdet 00000000 och det högsta värdet $11111111_2 = 2^8\text{-}1 = 255_{10}$.
Talintervallet som kan visas är alltså $[0, 255_{10}] = [0, 2^8\text{–}1]$.
Efter det högsta värdet varvar vägmätaren, dvs. den börjar om på 0. Antalet olika tal som vägmätaren kan visa är $2^8 = 256$.

Bin							Hex	Dec
0	0	0	0	0	0		0 0	0
0	0	0	0	0	1		0 1	1
0	0	0	0	1	0		0 2	2
0	0	0	0	1	1		0 3	3
0	0	0	1	0	0		0 4	4
0	0	0	1	0	1		0 5	5
0	0	0	1	1	0		0 6	6
0	0	0	1	1	1		0 7	7
0	0	1	0	0	0		0 8	8
0	0	1	0	0	1		0 9	9
0	0	1	0	1	0		0 A	10
0	0	1	0	1	1		0 B	11
0	0	1	1	0	0		0 C	12
0	0	1	1	0	1		0 D	13
0	0	1	1	1	0		0 E	14
0	0	1	1	1	1		0 F	15
0	1	0	0	0	0		1 0	16
0	1	0	0	0	1		1 1	17
0	1	0	0	1	0		1 2	18
0	1	0	0	1	1		1 3	19
0	1	0	1	0	0		1 4	20
0	1	0	1	0	1		1 5	21
0	1	0	1	1	0		1 6	22
0	1	0	1	1	1		1 7	23
0	1	1	0	0	0		1 8	24
0	1	1	0	0	1		1 9	25
0	1	1	0	1	0		1 A	26
0	1	1	0	1	1		1 B	27
0	1	1	1	0	0		1 C	28
0	1	1	1	0	1		1 D	29
0	1	1	1	1	0		1 E	30
0	1	1	1	1	1		1 F	31
1	0	0	0	0	0		2 0	32
1	0	0	0	0	1		2 1	33
1	0	0	0	1	0		2 2	34
1	0	0	0	1	1		2 3	35
1	0	0	1	0	0		2 4	36
1	0	0	1	0	1		2 5	37
1	0	0	1	1	0		2 6	38
1	0	0	1	1	1		2 7	39
1	0	1	0	0	0		2 8	40
1	0	1	0	0	1		2 9	41
1	0	1	0	1	0		2 A	42
1	0	1	0	1	1		2 B	43
1	0	1	1	0	0		2 C	44
1	0	1	1	0	1		2 D	45
1	0	1	1	1	0		2 E	46
1	0	1	1	1	1		2 F	47
1	1	0	0	0	0		3 0	48

Exempel på excess-8-kod (excess 2^{n-1}-kod, med n = 4)

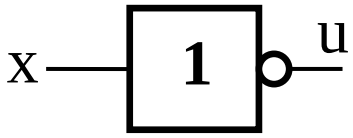
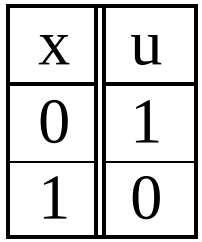
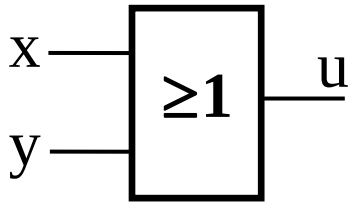
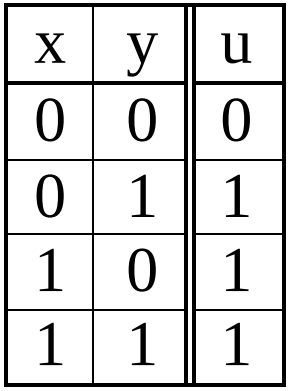
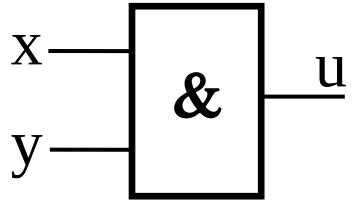
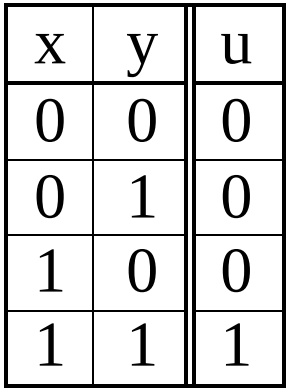


Koden decimalt:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Koden binärt:	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Spänningsvärde:	-8V	-7V	-6V	-5V	-4V	-3V	-2V	-1V	0V	+1V	+2V	+3V	+4V	+5V	+6V	+7V

U

Man får här spänningen i volt genom att subtrahera 8 från kodordets värde (excess-8).

Lägg märke till att spänningsvärdet +8 V inte får något kodord för denna kod.

Grind (Gate)	Symbol	Funktions-tabell	Booleskt uttryck
INVERTERARE (ICKE, NOT)			$u = x'$
ELLER (OR)			$u = x + y$
OCH (AND)			$u = x \cdot y$

x	u
0	1
1	0

x	y	u
0	0	0
0	1	1
1	0	1
1	1	1

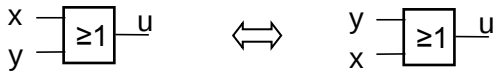
x	y	u
0	0	0
0	1	0
1	0	0
1	1	1

Några viktiga satser inom Boolesk algebra.

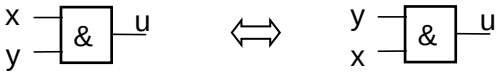
1.	$x + y = y + x$ $x \cdot y = y \cdot x$	**Kommutativa lagarna**
2.	$x \cdot (y + z) = x \cdot y + x \cdot z$ $x + (y \cdot z) = (x + y) \cdot (x + z)$	**Distributiva lagarna**
3.	$x + 0 = x$ $x \cdot 1 = x$	
4.	$x + x' = 1$ $x \cdot x' = 0$	
5.	$x + 1 = 1$ $x \cdot 0 = 0$	
6.	$x + x = x$ $x \cdot x = x$	
7.	$x + (y + z) = (x + y) + z$ $x \cdot (y \cdot z) = (x \cdot y) \cdot z$	**Associativa lagarna**
8.	$(x + y)' = x' \cdot y'$ $(x \cdot y)' = x' + y'$	**De Morgans lagar**
9.	$(x')' = x$	

Viktiga satser i switchnätalgebra och motsvarande grindnät

1. $x + y = y + x$

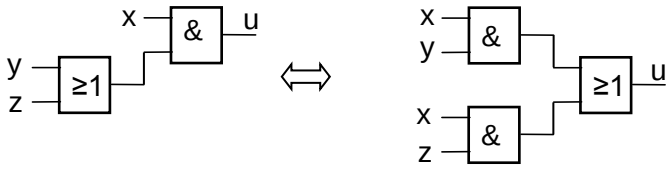


$x \cdot y = y \cdot x$

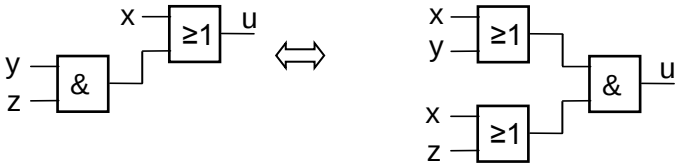


---

2. $x \cdot (y + z) = x \cdot y + x \cdot z$

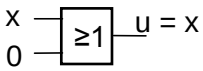


$x + (y \cdot z) = (x + y) \cdot (x + z)$

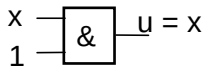


---

3. $x + 0 = x$

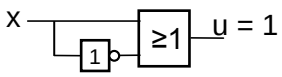


$x \cdot 1 = x$

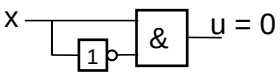


---

4. $x + x' = 1$

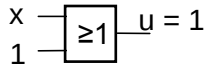


$x \cdot x' = 0$

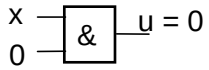


---

5. $x + 1 = 1$

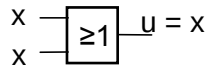


$x \cdot 0 = 0$

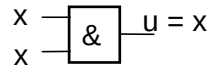


---

6. $x + x = x$

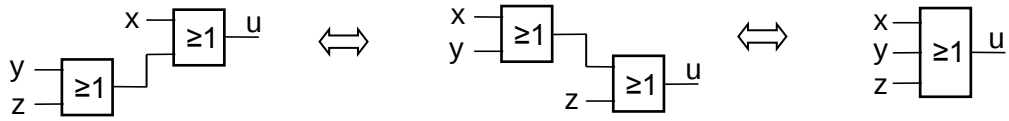


$x \cdot x = x$

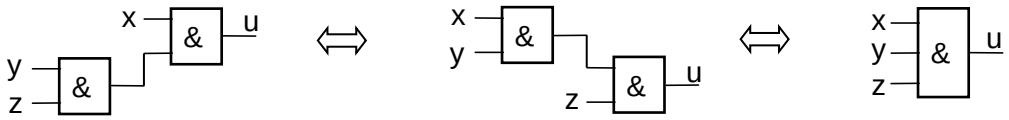


---

7. $x + (y + z)$ = $(x + y) + z$ = $x + y + z$

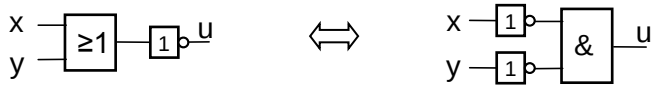


$x \cdot (y \cdot z)$ = $(x \cdot y) \cdot z$ = $x \cdot y \cdot z$

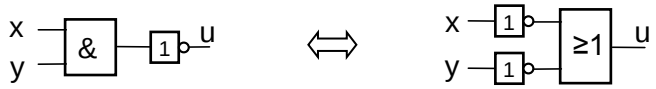


---

8. $(x + y)'$ = $x' \cdot y'$

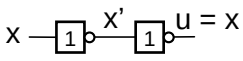


$(x \cdot y)'$ = $x' + y'$



---

9. $(x')' = x$



Övningsexempel på karnaughdiagram

Tag fram minimala SP-uttryck ur karnaughdiagrammen nedan! (Ettorna!)

f_1	zw 00	01	11	10
xy 00	1	0	0	1
01	0	1	1	0
11	0	1	1	0
10	1	0	0	1

$f_1 =$

f_2	zw 00	01	11	10
xy 00	1	1	1	1
01	0	1	1	0
11	0	0	0	0
10	0	0	0	0

$f_2 =$

f_3	zw 00	01	11	10
xy 00	1	0	0	1
01	1	1	1	1
11	1	1	1	1
10	1	0	0	1

$f_3 =$

f_4	zw 00	01	11	10
xy 00	0	1	1	0
01	0	1	1	1
11	0	1	1	1
10	0	0	0	0

$f_4 =$

f_5	zw 00	01	11	10
xy 00	1	1	1	1
01	1	1	1	1
11	0	0	0	0
10	0	1	1	0

$f_5 =$

f_6	zw 00	01	11	10
xy 00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	0	0	1

$f_6 =$

f_7	zw 00	01	11	10
xy 00	1	0	0	1
01	1	1	1	1
11	0	1	1	0
10	1	0	0	1

$f_7 =$

f_8	zw 00	01	11	10
xy 00	1	0	0	1
01	0	1	1	0
11	0	1	1	0
10	1	1	1	1

$f_8 =$

f_9	zw 00	01	11	10
xy 00	1	0	0	1
01	0	1	0	0
11	0	0	1	0
10	1	0	0	1

$f_9 =$

Svar:

$f_1 = y'w' + yw$ $f_2 = x'y' + x'w$ $f_3 = y + w'$

$f_4 = x'w + yw + yz$ $f_5 = x' + y'w$ $f_6 = x' + y + w'$

$f_7 = y'w' + yw + x'y$ $f_8 = y'w' + yw + xw$ $f_9 = y'w' + x'yz'w + xyzw$

alternativt

$f_7 = y'w' + yw + x'w'$ $f_8 = y'w' + yw + xy'$

Tag fram minimala PS-uttryck ur karnaughdiagrammen nedan! (Nollorna!)

f_{10}	zw 00	01	11	10
xy 00	1	0	0	1
01	0	1	1	0
11	0	1	1	0
10	1	0	0	1

f_{10} =

f_{11}	zw 00	01	11	10
xy 00	1	1	1	1
01	0	1	1	0
11	0	0	0	0
10	0	0	0	0

f_{11} =

f_{12}	zw 00	01	11	10
xy 00	1	0	0	1
01	1	1	1	1
11	1	1	1	1
10	1	0	0	1

f_{12} =

f_{13}	zw 00	01	11	10
xy 00	0	1	1	0
01	0	1	1	1
11	0	1	1	1
10	0	0	0	0

f_{13} =

f_{14}	zw 00	01	11	10
xy 00	1	1	1	1
01	1	1	1	1
11	0	0	0	0
10	0	1	1	0

f_{14} =

f_{15}	zw 00	01	11	10
xy 00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	0	0	1

f_{15} =

f_{16}	zw 00	01	11	10
xy 00	1	0	0	1
01	1	1	1	1
11	0	1	1	0
10	1	0	0	1

f_{16} =

f_{17}	zw 00	01	11	10
xy 00	1	0	0	1
01	0	1	1	0
11	0	1	1	0
10	1	1	1	1

f_{17} =

f_{18}	zw 00	01	11	10
xy 00	1	0	0	1
01	0	1	0	0
11	0	0	1	0
10	1	0	0	1

f_{18} =

Svar:

$f_{10} = (y' + w)(y + w')$ $f_{11} = x'(y' + w)$ $f_{12} = y + w'$

$f_{13} = (y + w)(z + w)(x' + y)$ $f_{14} = (x' + y')(x' + w)$ $f_{15} = x' + y + w'$

$f_{16} = (y + w')(x' + y' + w)$ $f_{17} = (y' + w)(x + y + w')$

$f_{18} = (y' + w)(y + w')(x' + y' + z)(x + y' + z')$ eller

$f_{18} = (y' + w)(y + w')(x' + y' + z)(x + z' + w')$ eller

$f_{18} = (y' + w)(y + w')(x' + z + w')(x + y' + z')$ eller

$f_{18} = (y' + w)(y + w')(x' + z + w')(x + z' + w')$

Ett exempel på användning av väljare

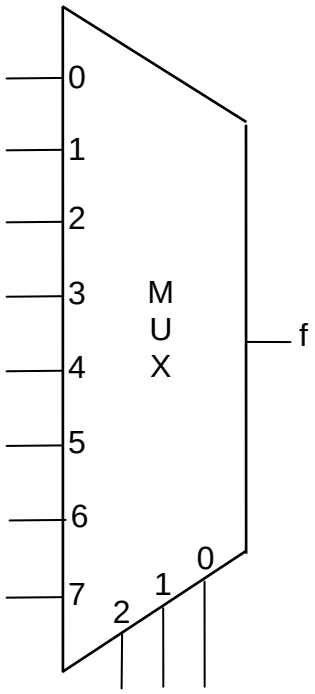
Man kan realisera vilken som helst av samtliga möjliga Booleska funktioner av n variabler genom att använda en "1 av 2^{n-1} väljare" och eventuellt en NOT-grind.

Alltså kan en "1 av 8 väljare" tillsammans med en NOT-grind användas för att realisera vilken som helst av de 2^{16} = 65536 olika Booleska funktionerna av fyra variabler.

Uppgift: Använd en "1 av 8 väljare" för att realisera den Booleska funktionen f(x,y,z,w) som definieras av funktionstabellen nedan.

Skriv in nedan till höger hur variablerna kopplas till väljarens styringångar. Skriv också in vilka värden som kopplas in på väljarens dataingångar.

x	y	z	w	f
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0



Övningsuppgifter (ur KMP)

2.1 Omvandla till decimal form

a) 101101_2 **f)** 101.101_2

2.3 Omvandla till decimal form

a) $2C3_{16}$

2.5 Omvandla till hexadecimal form

a) 10101010_2

2.6 Omvandla till binär form (naturlig binär kod).

a) 23_{10}

2.13 Skriv $563{,}782_{10}$ på NBCD-kod.

3.4 I denna uppgift betecknar x, y och z booleska variabler. Bevisa genom fullständig binär evaluering, att

b) $x + yz = (x+y)(x + z)$

3.5 Åskådliggör med hjälp av grindsymboler

a) $z\cdot(x'+y)$

3.13 Skriv nedanstående booleska funktion på disjunktiv normal form.

d) $f = x\cdot(y'+ z)$

3.14 Skriv nedanstående booleska funktion på konjunktiv normal form.

c) $f = x\cdot(y'+ z)$

4.15 Konstruera ett minimalt kombinatoriskt nät som omvandlar den NBCD-kodade siffran $X = (x_3x_2x_1x_0)_{NBCD}$ till sitt 9-komplement $Y = 9\text{-}X = (y_3y_2y_1y_0)_{NBCD}$. Nätet skall således ha fyra in- och fyra utsignaler. Tal som inte tillhör NBCD-koden kommer inte att uppträda. Vid realiseringen får INVERTERARE, samt NAND- och XOR-grindar användas. Realiseringen skall ha så få grindar som möjligt.

4.25 Realisera funktionen f = b'd' + bd + acd med hjälp av en ”1 av 8”-väljare. INVERTERARE får också användas.

Representation av heltal med ett begränsat antal sifferpositioner

Med ett begränsat antal sifferpositioner kan man bara korrekt representera heltal inom ett begränsat talområde. Man kan dock representera större heltal genom att helt enkelt stryka de siffror till vänster som inte ”får plats”, men man får då en osäkerhet (ett fel) som beror på värdet av de strukna siffrorna.

Ex.
Med fyra decimala siffror kan man representera heltal i intervallet [0, 9999] = [0, 10^4-1] korrekt. Med fyra decimala sifferpositioner säger man att talen representeras modulo 10^4. Det innebär att alla tal som är större än $10^4 – 1 = 9999$ kommer att representeras med en osäkerhet (fel) i form av termen $n \cdot 10^4$, där n är ett heltal (n = 1, 2, 3, ...).

Talet $37680_{10} = 3 \cdot 10^4 + 7680$ representeras med fyra siffror av talet 7680 och felet är $3 \cdot 10^4$.

Ex.
Med åtta binära siffror (bitar) kan man representera heltal i intervallet [0, 11111111_2] = [0, 2^8-1] = = [0, 255_{10}] korrekt. Med åtta bitpositioner säger man att talen representeras modulo 2^8. Det innebär att alla tal som är större än $2^8 – 1 = 255_{10}$ kommer att representeras med en osäkerhet (fel) i form av termen $n \cdot 2^8$, där n är ett heltal (n = 1, 2, ...).

Talet $943_{10} = 1110101111_2 = 11_2 \cdot 2^8 + 10101111_2 = 3 \cdot 2^8 + 175_{10}$ representeras alltså med åtta bitar av talet $10101111_2 = 175_{10}$ och felet är $3 \cdot 2^8$.

Decimal addition för heltal utan tecken, S = X + Y

Exempel med två tal X och Y med fyra sifferpositioner:

X = 7396

Y = 5842

```
  4 3 2 1 0     sifferposition
  1 1 1 0 0     minnessiffra (carry)
    7 3 9 6      X
   +5 8 4 2     +Y
  1 3 2 3 8 = S = 13238 = 10000 + 3238
```

Additionen ger en summa som kräver 5 siffror. ”Overflow” (spill) för tal utan tecken har inträffat eftersom den femte siffran $c_4 = 1$.

”Overflow” vid addition innebär att talområdets övre gräns har passerats.

Den femte siffran c_4 har vikten $10^4 = 10000$, så värdet med endast 4 sifferpositioner blir 3238.

Summan av två tal med vardera 4 siffror ryms alltid i 5 sifferpositioner.

Talområdet för 4-siffriga tal utan tecken är $[0, 10^4 - 1] = [0, 9999]$.

Binär addition S = X + Y

Exempel med två 8-bitars tal X och Y:

X = $10101101_2 = 173_{10}$

Y = $01100110_2 = 102_{10}$

```
  8 7 6 5 4 3 2 1 0  bitposition
  1 1 1 0 1 1 0 0    minnessiffra (carry)
    1 0 1 0 1 1 0 1    X
   +0 1 1 0 0 1 1 0   +Y
  ---------------------
  1 0 0 0 1 0 0 1 1 = S = 275(10)= 256(10) + 19(10)
```

Additionen ger en summa som kräver 9 bitar. ”Overflow” för tal utan tecken har inträffat eftersom den nionde biten $c_8 = 1$.

”Overflow” vid addition innebär att talområdets övre gräns har passerats.

Den nionde biten c_8 har vikten $2^8 = 256_{10}$, så värdet med endast 8 bitar blir 19_{10}.

Summan av två 8-bitars tal ryms alltid i 9 bitar.

Talområdet för 8-bitars tal utan tecken är $[0, 2^8 - 1] = [0, 256_{10} - 1] = [0, 255_{10}]$.

Decimal subtraktion för heltal utan tecken, D = X – Y

Exempel med två tal X och Y med fyra sifferpositioner:

X = 7396

Y = 5842

```
  3   2   1   0      sifferposition
    10   0   0      lån (borrow)
 ~~7~~   3   9   6     X
-5   8   4   2    -Y
 1   5   5   4  = D
```

Man kan också utföra subtraktionen med **10-komplementaritmetik** som visas nedan.

Byt ut X – Y mot $10^4 + X – Y$.

$10^4 + X – Y = 10000 + X – Y = 9999 + 1 + X – Y = X + (9999 – Y) + 1 = X + Y_{9k} + 1$

$9999 – Y = Y_{9k}$ som kallas (siffervis) 9-komplement av Y.
Man bildar Y_{9k} genom att byta ut alla siffror i Y mot $9 – y_i$ (9 – siffran).

Med samma värden på X och Y som ovan får vi:

$Y_{9k} = 4157$

```
 4  3  2  1  0        sifferposition
 1  0  1  1  1     1  minnessiffra (carry)
    7  3  9  6     X
  +4  1  5  7    +Y9k
~~1~~ 1  5  5  4  = D = ~~10000~~ + 1554
```

Lägg märke till att vi har satt minnessiffran längst till höger till 1 ($c_0 = 1$) och att minnessiffran ut $c_4 = 1$. c_4 har ju vikten $10^4 = 10000$ som vi lade till från början.

Vi får den korrekta skillnaden genom att stryka c_4.

Binär subtraktion D = X – Y

Exempel med två 8-bitars tal X och Y:

$X = 10101101_2 = 173_{10}$

$Y = 01100110_2 = 102_{10}$

```
   2         2 2     lånesiffra (borrow)
 ~1~ 0 1 0 ~1~ ~1~ 0 1   X
-0 1 1 0 0 1 1 0   -Y
 0 1 0 0 0 1 1 1 = D = 71₁₀
```

Addition och subtraktion utförs olika och kräver därmed olika grindnät (= kretsar).

Vi använder istället **2-komplementaritmetik**, på samma sätt som vi tidigare använde **10-komplementaritmetik** och kan då utföra vår subtraktion med en addition.

Byt ut X – Y mot $2^n + X – Y$, där n är antalet bitar vi arbetar med.

Exempel med n = 8:

$2^8 + X – Y = 256 + X – Y = 255 + 1 + X – Y = X + (11111111_2 – Y) + 1 = X + Y_{1k} + 1$

$11111111_2 – Y = Y_{1k}$ som kallas (siffervis) 1-komplement av Y.
Man bildar Y_{1k} genom att invertera alla bitar i Y.

Med samma värden på X och Y som ovan får vi:

```
 8 7 6 5 4 3 2 1 0  bitposition
 1 0 1 1 1 0 0 1 1     1  minnessiffra (carry)
   1 0 1 0 1 1 0 1     X
  +1 0 0 1 1 0 0 1   +Y1k
~1~ 0 1 0 0 0 1 1 1 = D = 71₁₀
```

Lägg märke till att vi har satt minnessiffran längst till höger till 1. ($c_0 = 1$)

Decimal aritmetik

10-komplementrepresentation av decimala tal med tecken

Antag att vi har tre decimala tal A, B och C med vardera 4 siffror.

Ex.
A = 7652, B = 2153 och C = 4789

Man lägger först till en teckensiffra 0 längst till vänster i talen för att tala om att de är positiva:

A = 07652, B = 02153 och C = 04789

De negativa talen –A, –B och –C motsvaras av deras 10-komplement.

$A_{10k} = 10^5 - A = (99999 - A) + 1 = A_{9k} + 1$ (Detta tal används alltså som –A.)
$B_{10k} = 10^5 - B = (99999 - B) + 1 = B_{9k} + 1$ (Detta tal används alltså som –B.)
$C_{10k} = 10^5 - C = (99999 - C) + 1 = C_{9k} + 1$ (Detta tal används alltså som –C.)

10-komplementering innebär alltså byte av tecken från positivt till negativt eller tvärt om.

Talet – A = – 7652 motsvaras av A_{10k} = 99999 – 07652 + 1 = 92347 + 1 = 92348
Talet – B = – 2153 motsvaras av B_{10k} = 99999 – 02153 + 1 = 97846 + 1 = 97847
Talet – C = – 4789 motsvaras av C_{10k} = 99999 – 04789 + 1 = 95210 + 1 = 95211

Positiva tal inleds med 0 och negativa tal med 9.
(Man kan även tänka sig att positiva tal inleds med någon av siffrorna 0-4 och negativa med någon av siffrorna 5-9, men detta skulle komplicera identifieringen av talets tecknet.)
Med 0 och 9 som teckensiffror blir talområdet med 5 siffror inklusive teckensiffran [-10000, +9999]

10-komplementaritmetik

Vi använder talen A, B och C ovan som exempel.

Addition (Alla beräkningar nedan utförs med endast 5 siffror.)
(+A) + (+B) = 07652 + 02153 = 09805 (Korrekt eftersom teckensiffrorna 0 + 0 = 0.)
(+A) + (+C) = 07652 + 04789 = 12441 (Overflow eftersom teckensiffrorna 0 + 0 = 1.)
(+A) + (–B) vilket motsvarar A + B_{10k} = 07652 + 97847 = 05499 (Korrekt, olika tecken.)
(+A) + (–C) vilket motsvarar A + C_{10k} = 07652 + 95211 = 02863 (Korrekt, olika tecken.)
(–A) + (+B) vilket motsvarar A_{10k} + B = 92348 + 02153 = 94501, dvs. – 5499 (Korrekt, olika tecken.)
(–A) + (+C) vilket motsvarar A_{10k} + C = 92348 + 04789 = 97137, dvs. – 2863 (Korrekt, olika tecken.)
(–A) + (–B) = vilket motsvarar $A_{10k} + B_{10k}$ = 92348 + 97847 = 90195 (Korrekt, 9 + 9 = 9.)
(–A) + (–C) = vilket motsvarar $A_{10k} + C_{10k}$ = 92348 + 95211 = 87559 (Overflow, 9 + 9 = 8.)

Subtraktion (Alla beräkningar nedan utförs med endast 5 siffror.)
(+A) – (+B) vilket motsvarar A + B_{9k} + 1 = 07652 + 97846 +1 = 05499 (Korrekt, olika tecken.)
(+A) – (+C) vilket motsvarar A + C_{9k} + 1 = 07652 + 95210 +1 = 02863 (Korrekt, olika tecken.)
(+A) – (–B) vilket motsvarar A + $(B_{10k})_{9k}$ + 1 = 07652 + $(97847)_{9k}$ + 1 = 07652 + 02152 +1 = 09805 (Korrekt, 0 + 0 = 0)
(+A) – (–C) vilket motsvarar A + $(C_{10k})_{9k}$ + 1 = 07652 + $(95211)_{9k}$ + 1 = 07652 + 04788 +1 = 12441 (Overflow, 0 + 0 = 1)
(–A) – (+B) vilket motsvarar A_{10k} + B_{9k} + 1 = = 92348 + 97846 + 1 = 90195, dvs. – 9805 (Korrekt, 9 + 9 = 9)
(–A) – (+C) vilket motsvarar A_{10k} + C_{9k} + 1 = = 92348 + 95210 + 1 = 87559 (Overflow, 9 + 9 = 8)
(–A) – (–B) vilket motsvarar A_{10k} + $(B_{10k})_{9k}$ + 1 = 92348 + $(97847)_{9k}$ + 1 = 92348 + 02152 + 1 = 94501, dvs – 5499 (Korrekt)
(–A) – (–C) vilket motsvarar A_{10k} + $(C_{10k})_{9k}$ + 1 = 92348 + $(95211)_{9k}$ + 1 = 92348 + 04788 + 1 = 97137, dvs – 2863 (Korrekt)

Binär aritmetik (för tal med tecken)

2-komplementrepresentation av binära tal med tecken

Antag att vi har tre positiva binära tal A, B och C med vardera 7 bitar.

Ex.

$A = 1011001_2$ (= 89_{10}) $B = 0100110_2$ (= 38_{10}) $C = 0111001_2$ (= 57_{10})

Man lägger först till en teckensiffra 0 längst till vänster i talen för att tala om att de är positiva:

$A = 01011001_2$ $B = 00100110_2$ $C = 00111001_2$

De negativa talen –A, –B och –C motsvaras av deras 2-komplement.

$A_{2k} = 2^8 - A = (11111111_2 - A) + 1 = A_{1k} + 1$ (Detta tal används alltså som –A)

$B_{2k} = 2^8 - B = (11111111_2 - B) + 1 = B_{1k} + 1$ (Detta tal används alltså som –B)

$C_{2k} = 2^8 - C = (11111111_2 - C) + 1 = C_{1k} + 1$ (Detta tal används alltså som –C)

2-komplementering innebär byte av tecken från positivt till negativt eller tvärt om.

Talet $-A = -1011001_2$ motsvaras av $A_{2k} = (01011001_2)_{1k} + 1 = 10100110_2 + 1 = 10100111_2$ (motsvarar -89_{10})

Talet $-B = -0100110_2$ – ” – $B_{2k} = (00100110_2)_{1k} + 1 = 11011001_2 + 1 = 11011010_2$ (” -38_{10})

Talet $-C = -0111001_2$ – ” – $C_{2k} = (00111001_2)_{1k} + 1 = 11000110_2 + 1 = 11000111_2$ (” -57_{10})

Positiva tal inleds med 0 och negativa tal med 1.

Talområdet med 8 bitar inklusive teckenbiten: $[-10000000_2, +01111111_2] = [-128_{10}, +127_{10}]$.

Man ser att talet -128_{10} inte har någon positiv motsvarighet.

2-komplementaritmetik

Vi använder talen A, B och C ovan som exempel.

Addition (Alla beräkningar nedan utförs med 8 bitar.)

(+A) + (+B) = $\underline{0}1011001_2 + \underline{0}0100110_2 = \underline{0}1111111_2 = +127_{10}$ (Korrekt eftersom teckensiffrorna 0 + 0 = 0.)

(+A) + (+C) = $\underline{0}1011001_2 + \underline{0}0111001_2 = \underline{1}0010010_2$, motsv. -110_{10} (Overflow eftersom teckensiffrorna 0 + 0 = 1.)

(+A) + (–B) motsvaras av $A + B_{2k} = \underline{0}1011001_2 + \underline{1}1011010_2 = \underline{0}0110011_2 = +51_{10}$ (Korrekt, olika tecken.)

(+A) + (–C) – ” – $A + C_{2k} = \underline{0}1011001_2 + \underline{1}1000111_2 = \underline{0}0100000_2 = +32_{10}$ (Korrekt, olika tecken.)

(–A) + (+B) – ” – $A_{2k} + B = \underline{1}0100111_2 + \underline{0}0100110_2 = \underline{1}1001101_2$, motsv. -51_{10} (Korrekt, olika tecken.)

(–A) + (+C) – ” – $A_{2k} + C = \underline{1}0100111_2 + \underline{0}0111001_2 = \underline{1}1100000_2$, motsv. -32_{10} (Korrekt, olika tecken.)

(–A) + (–B) – ” – $A_{2k} + B_{2k} = \underline{1}0100111_2 + \underline{1}1011010_2 = \underline{1}0000001_2$ motsv. -127_{10} (Korrekt, 1 + 1 = 1.)

(–A) + (–C) – ” – $A_{2k} + C_{2k} = \underline{1}0100111_2 + \underline{1}1000111_2 = \underline{0}1101110_2 = +110_{10}$ (Overflow, 1 + 1 = 0.)

Subtraktion (Alla beräkningar nedan utförs med 8 bitar.)

(+A) – (+B) motsvaras av $A + B_{1k} + 1 = \underline{0}1011001_2 + \underline{1}1011001_2 + 1 = \underline{0}0110011_2 = +51_{10}$ (Korrekt, olika tecken.)

(+A) – (+C) – ” – $A + C_{1k} + 1 = \underline{0}1011001_2 + \underline{1}1000110_2 + 1 = \underline{0}0100000_2 = +32_{10}$ (Korrekt, olika tecken.)

(+A) – (–B) – ” – $A + (B_{2k})_{1k} + 1 = \underline{0}1011001_2 + \underline{0}0100101_2 + 1 = \underline{0}1111111_2 = +127_{10}$ (Korrekt, 0 + 0 = 0)

(+A) – (–C) – ” – $A + (C_{2k})_{1k} + 1 = \underline{0}1011001_2 + \underline{0}0111000_2 + 1 = \underline{1}0010010_2$ motsv. -110_{10} (Overflow, 0 + 0 = 1)

(–A) – (+B) – ” – $A_{2k} + B_{1k} + 1 = = \underline{1}0100111_2 + \underline{1}1011001_2 + 1 = \underline{1}0000001_2$ motsv. -127_{10} (Korrekt, 1 + 1 = 1)

(–A) – (+C) – ” – $A_{2k} + C_{1k} + 1 = = \underline{1}0100111_2 + \underline{1}1000110_2 + 1 = \underline{0}1101110_2 = +110_{10}$ (Overflow, 1 + 1= 0)

(–A) – (–B) – ” – $A_{2k} + (B_{2k})_{1k} + 1 = \underline{1}0100111_2 + \underline{0}0100101_2 + 1 = \underline{1}1001101_2$, motsv. -51_{10} (Korrekt, olika tecken.)

(–A) – (–C) – ” – $A_{2k} + (C_{2k})_{1k} + 1 = \underline{1}0100111_2 + \underline{0}0111000_2 + 1 = \underline{1}1100000_2$, motsv. -32_{10} (Korrekt, olika tecken.)

Flaggorna N, Z, V och C som sätts vid aritmetiska operationer.

När man utför de aritmetiska operationerna addition eller subtraktion mellan två tal i en dator är det oftast resultatet, dvs summan eller skillnaden man söker. Man vill också veta om resultatet är korrekt med det antal sifferpositioner som används. I vissa fall vill man veta om resultatet är noll eller om det är positivt eller negativt om det är ett tal med tecken. Av dessa skäl bildar man förutom resultatet också ett antal ”flaggor” som innehåller information om resultatets giltighet, värde och tecken.

N-flaggan (Negative) sätts till resultatets mest signifikanta bit (msb), dvs ”teckenbiten” för tal med tecken.

Z-flaggan (Zero) sätts till värdet 1 om resultatet är noll.

V-flaggan (oVerflow) sätts till värdet 1 om ”overflow” för tal med tecken har inträffat, dvs resultatet av en addition eller subtraktion är felaktigt eftersom det korrekta resultatet ligger utanför det tillgängliga talområdet (= inte ryms med det antal sifferpositioner som används).

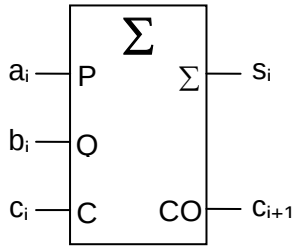
C-flaggan (Carry/Borrow):
C-flaggan sätts till värdet 1 om ”overflow” för tal utan tecken har inträffat vid addition, dvs resultatet av additionen ligger utanför talområdet (= inte ryms med det antal sifferpositioner som används).

C-flaggan har en annan funktion för subtraktion av ”tal utan tecken”. Den sätts till värdet 1 om resultatet av en subtraktion skulle bli negativt. (Det finns ju inga negativa tal för ”tal utan tecken”.)

Principen för addition av två n-bitars tal

$$
\begin{array}{llllllll}
c_n & c_{n-1} & c_{n-2} & .. & c_{i+1} & c_i & ..c_1 & 0 \quad \text{minnessiffror} \\
 & a_{n-1} & a_{n-2} & ..... & a_i & & ..a_1 & a_0 \quad \text{augendsiffror} \\
+ & b_{n-1} & b_{n-2} & ..... & b_i & & ..b_1 & b_0 \quad \text{addendsiffror} \\
\hline
s_n & s_{n-1} & s_{n-2} & ..... & s_i & & ..s_1 & s_0 \quad \text{summasiffror}
\end{array}
$$

För att addera två n-bitars tal krävs det enligt uppställningen ovan n st grindnät som bildar utsignalerna s_i och c_{i+1} av insignalerna a_i, b_i och c_i, dvs n st heladderare med symbol och funktionstabell nedan.



Heladderare

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

Ur funktionstabellen får man fram s_i och c_{i+1} på SP normal form:

$$s_i = a_i'b_i'c_i + a_i'b_ic_i' + a_ib_i'c_i' + a_ib_ic_i \qquad \text{(4.2 i KMP)}$$

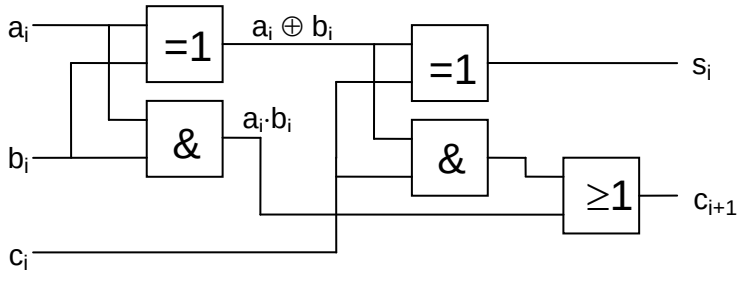
$$c_{i+1} = a_i'b_ic_i + a_ib_i'c_i + a_ib_ic_i' + a_ib_ic_i \qquad \text{(4.3 i KMP)}$$

Förenkling ger:

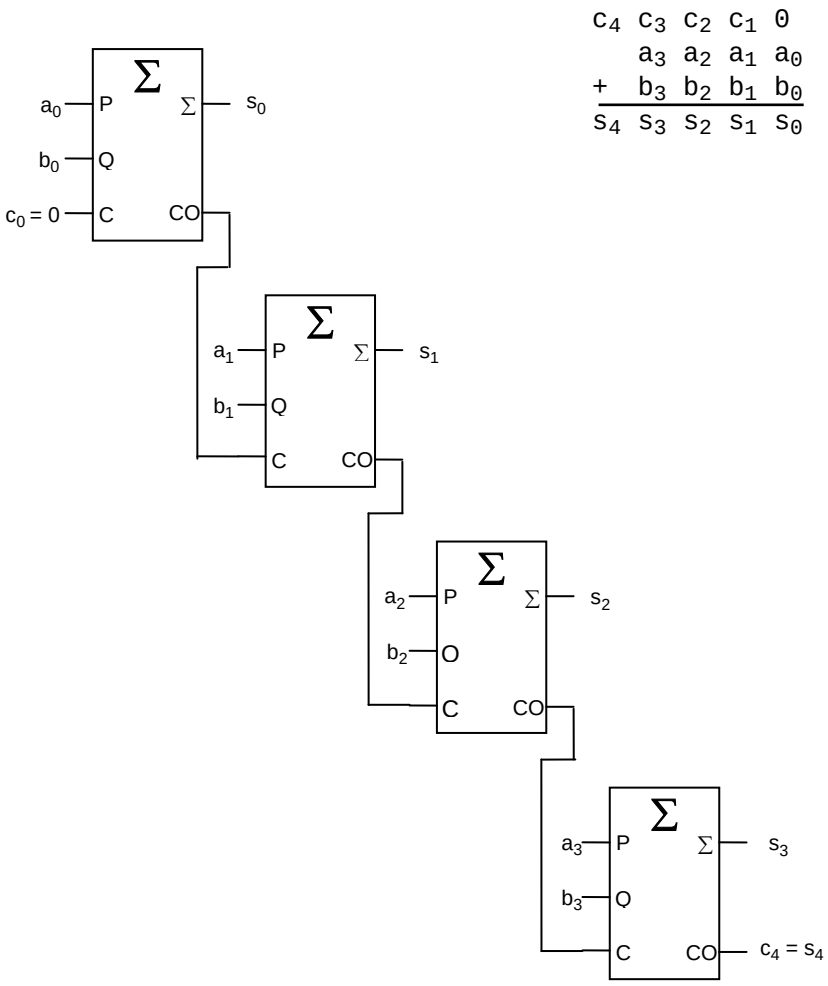
$$s_i = (a_i \oplus b_i) \oplus c_i \qquad \text{(4.4 i KMP)}$$

$$c_{i+1} = a_ib_i + (a_i \oplus b_i)\cdot c_i \qquad \text{(4.5 i KMP)}$$

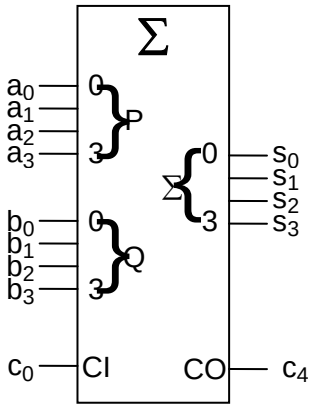
Motsvarande grindnät:



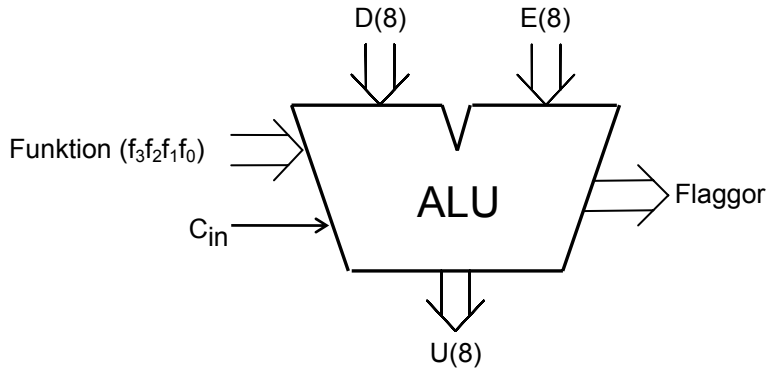
Genom att koppla ihop fyra heladderare enligt figuren nedan får man en ”fyrabitars heladderare” som adderar två st fyrabitars tal enligt uppställningen till höger.



Symbol för en fyrabitars heladderare:



Aritmetik- logikenhet (ALU)



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

I kolumnen Operation förklaras (där det behövs) hur operationen utförs. Tecknen **"+"** och **"–"** avser **aritmetiska operationer**. Med $\mathbf{D_{1k}}$ 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}	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

Carryflaggan (C) innehåller minnessiffran ut (carry-out) från den mest signifikanta bitpositionen (längst till vänster) om en aritmetisk operation utförs av ALU:n.

Vid **subtraktion** gäller för denna ALU att **C = 1 om lånesiffra (borrow) uppstår och C = 0 om lånesiffra inte uppstår**.

Carryflaggans värde är 0 vid andra operationer än aritmetiska.

Overflowflaggan (V) visar om en aritmetisk operation ger "overflow" enligt reglerna för 2-komplementaritmetik.

V-flaggans värde är 0 vid andra operationer än aritmetiska.

Zeroflaggan (Z) visar om en ALU-operation ger värdet noll som resultat på U-utgången.

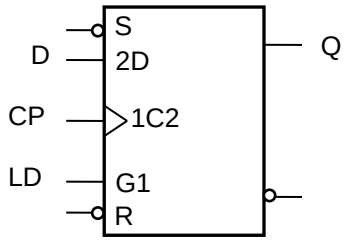
Signflaggan (N) är identisk med den mest signifikanta biten (teckenbiten) av utsignalen U från ALU:n.

D-vippa med "load enable"-ingång

En vanlig D-vippa laddas med nytt innehåll från D-ingången vid varje klockpuls. I ett system där en gemensam klocksignal är ansluten till alla vippor är detta inte lämpligt. Där vill man ha möjligheten att välja vilka vippor, som skall ladda ett nytt värde. I sådana fall krävs en extra ingång som bestämmer om vippan skall laddas med ett nytt värde.

Man kan komplettera en vanlig D-vippa så att den kan förberedas för att ladda eller inte ladda ett nytt värde vid positiv klockpulsflank.

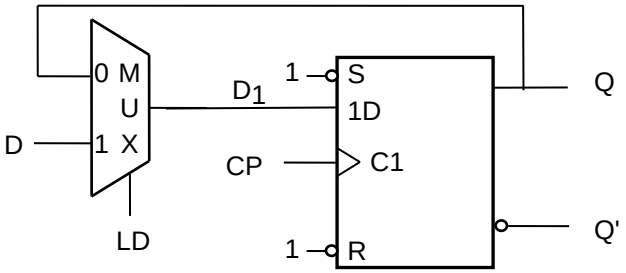
För att möjliggöra selektiv laddning skall vippan alltså kompletteras så att den också får en **"grindande"** ingång G1, som är en s k "load enable"-ingång. Ett exempel på en sådan D-vippa ges av Prosamsymbolen i figuren nedan, där **LD**, står för **"load enable"**. Denna D-vippa har som synes också asynkrona Set- och Reset-ingångar.



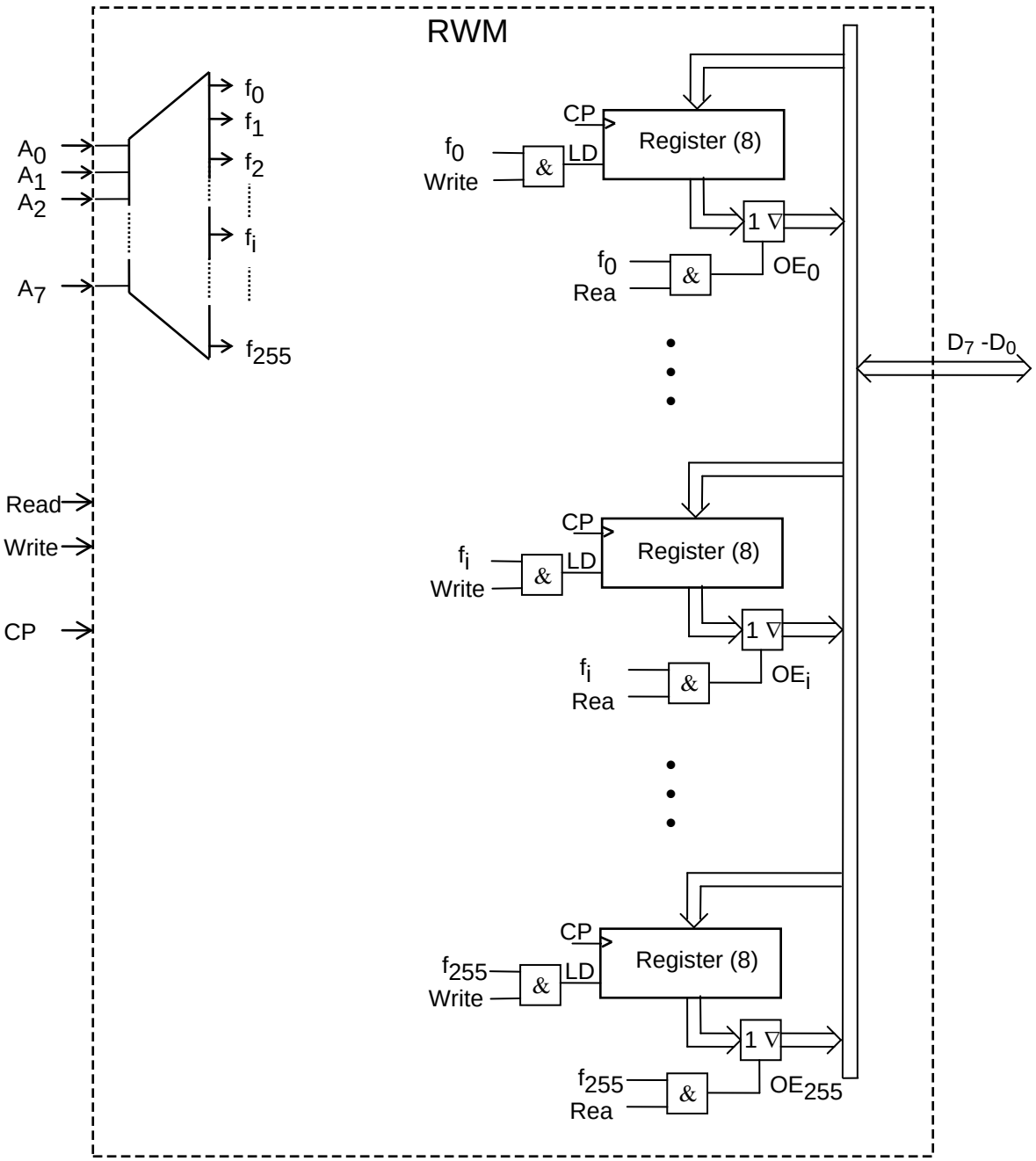
SEK Prosam-symbol för en D-vippa med "load enable"-ingång

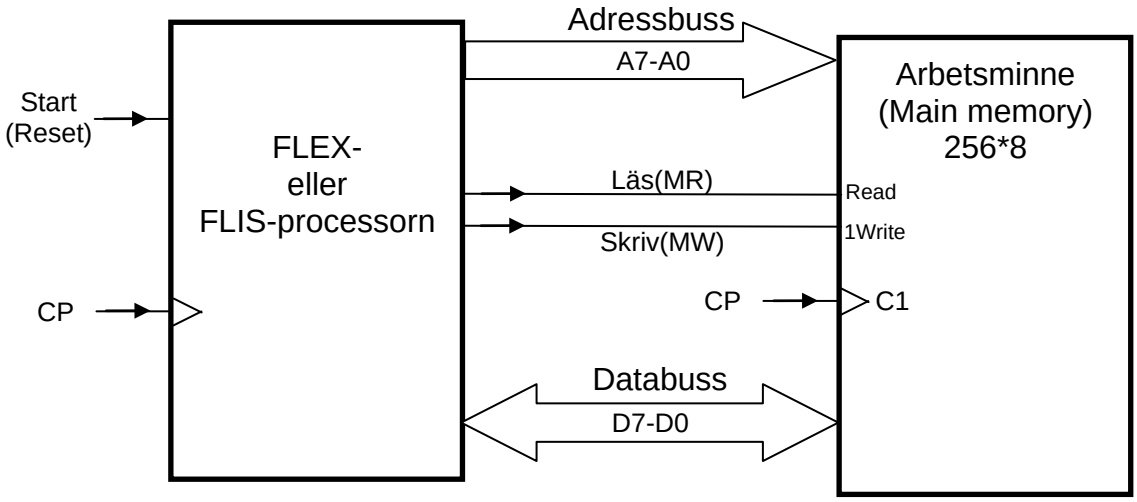
Funktionsbeskrivning

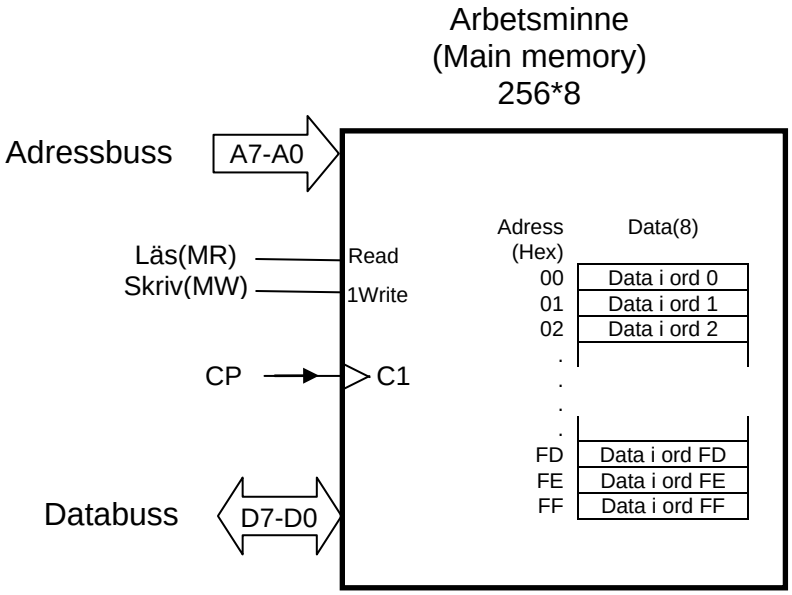
Genom att komplettera en vanlig D-vippa med en ”1 av 2 väljare” enligt figuren nedan kan man välja med insignalen LD, om vippan skall laddas med ett nytt värde Q = D vid aktiv klockpulsflank eller behålla det gamla värdet Q. När LD=1 laddas den med det nya värdet D och när LD = 0 återladdas den med sitt gamla värde Q. LD är således en **styrsignal,** som styr hur vippan skall laddas vid klockpulsens aktiva flank, i detta fall den positiva flanken.



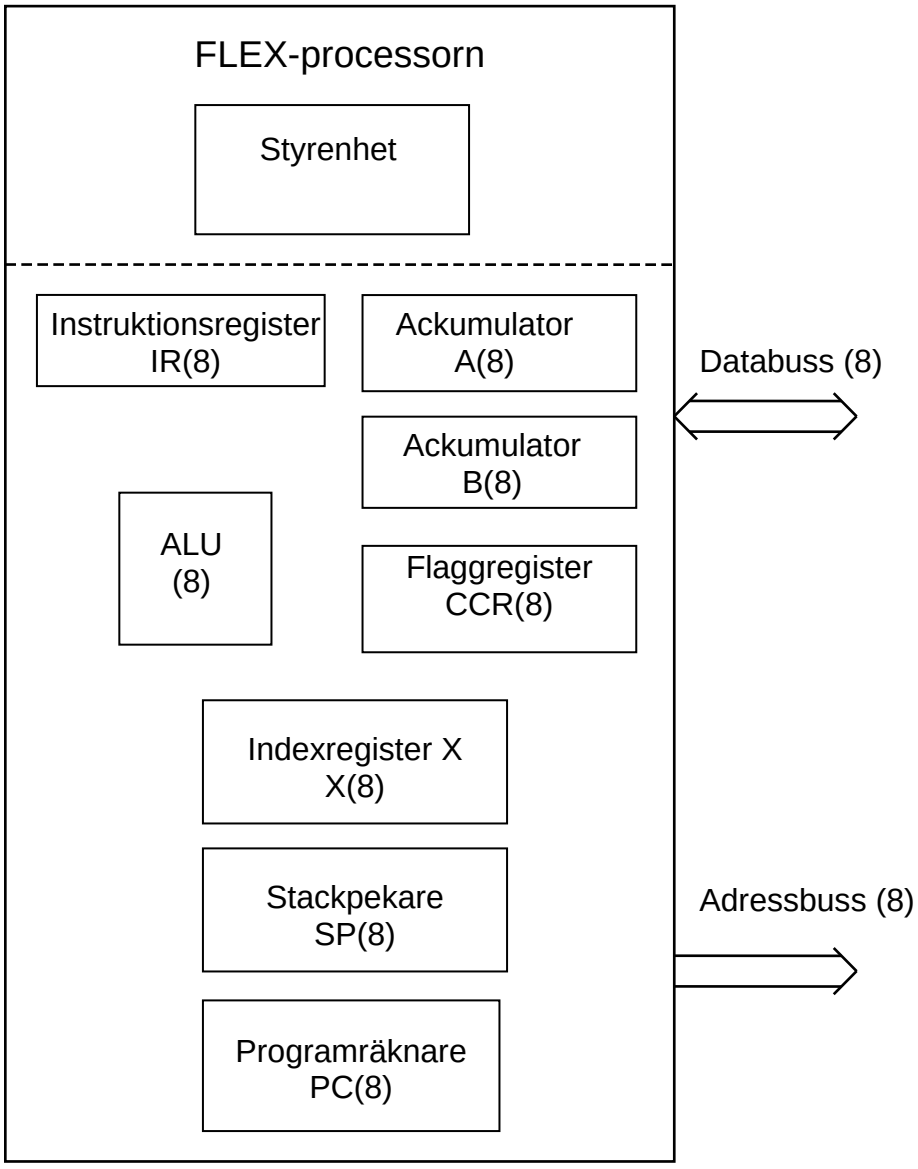
Läs- och skrivbart minne (RWM) 256∗8




Adressbuss
A7-A0
Start
(Reset)
FLEX-
eller
FLIS-processorn
Arbetsminne
(Main memory)
256*8
Läs(MR)
Read
1Write
Skriv(MW)
CP
CP
C1
Databuss
D7-D0

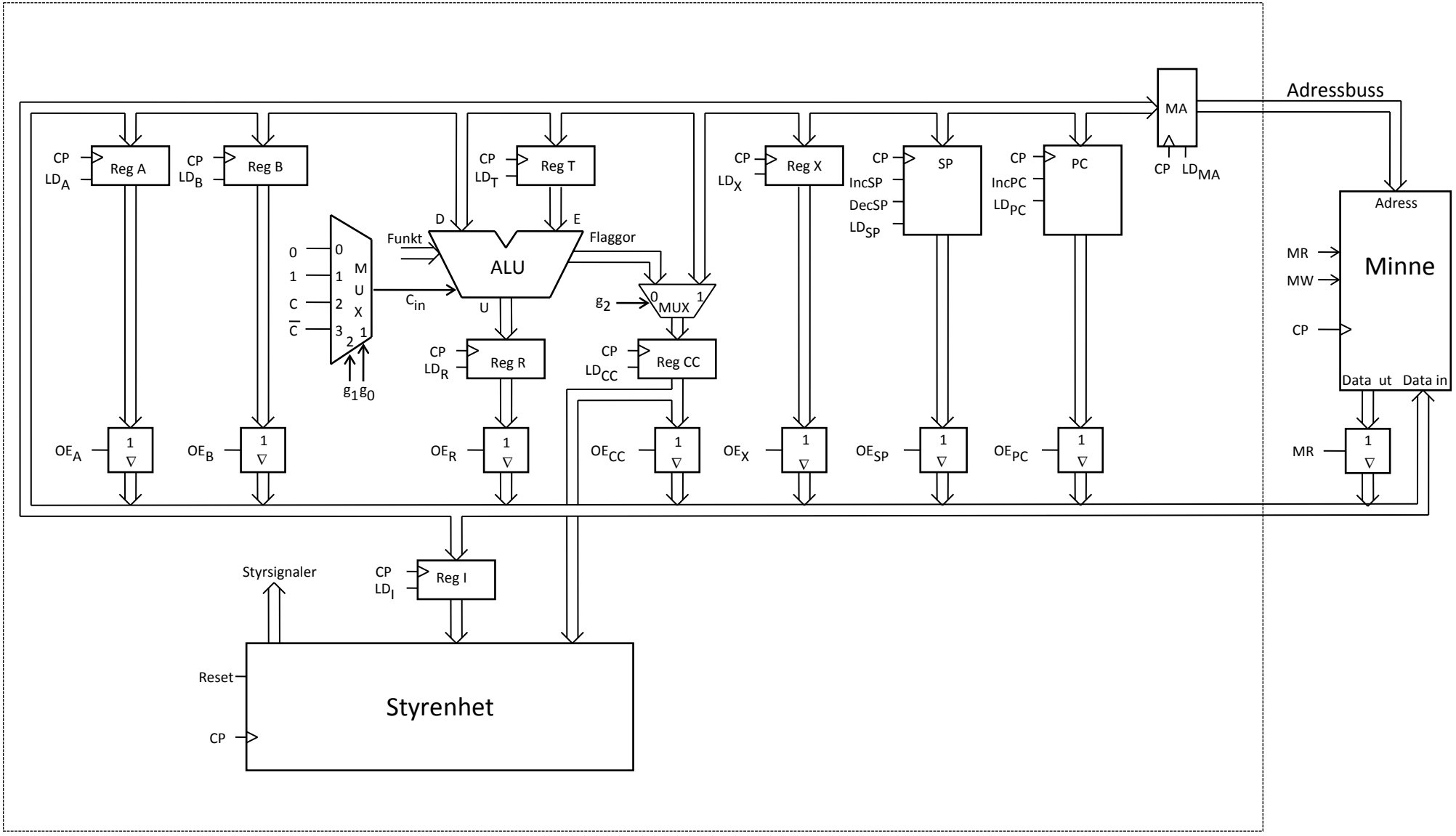

Arbetsminne
(Main memory)
256*8
Adressbuss
A7-A0
Läs(MR)
Skriv(MW)
Read
1Write
CP
C1
Databuss
D7-D0
Adress
(Hex)
Data(8)
00
Data i ord 0
01
Data i ord 1
02
Data i ord 2
FD
Data i ord FD
FE
Data i ord FE
FF
Data i ord FF

Programmerarens bild av FLEX-processorn

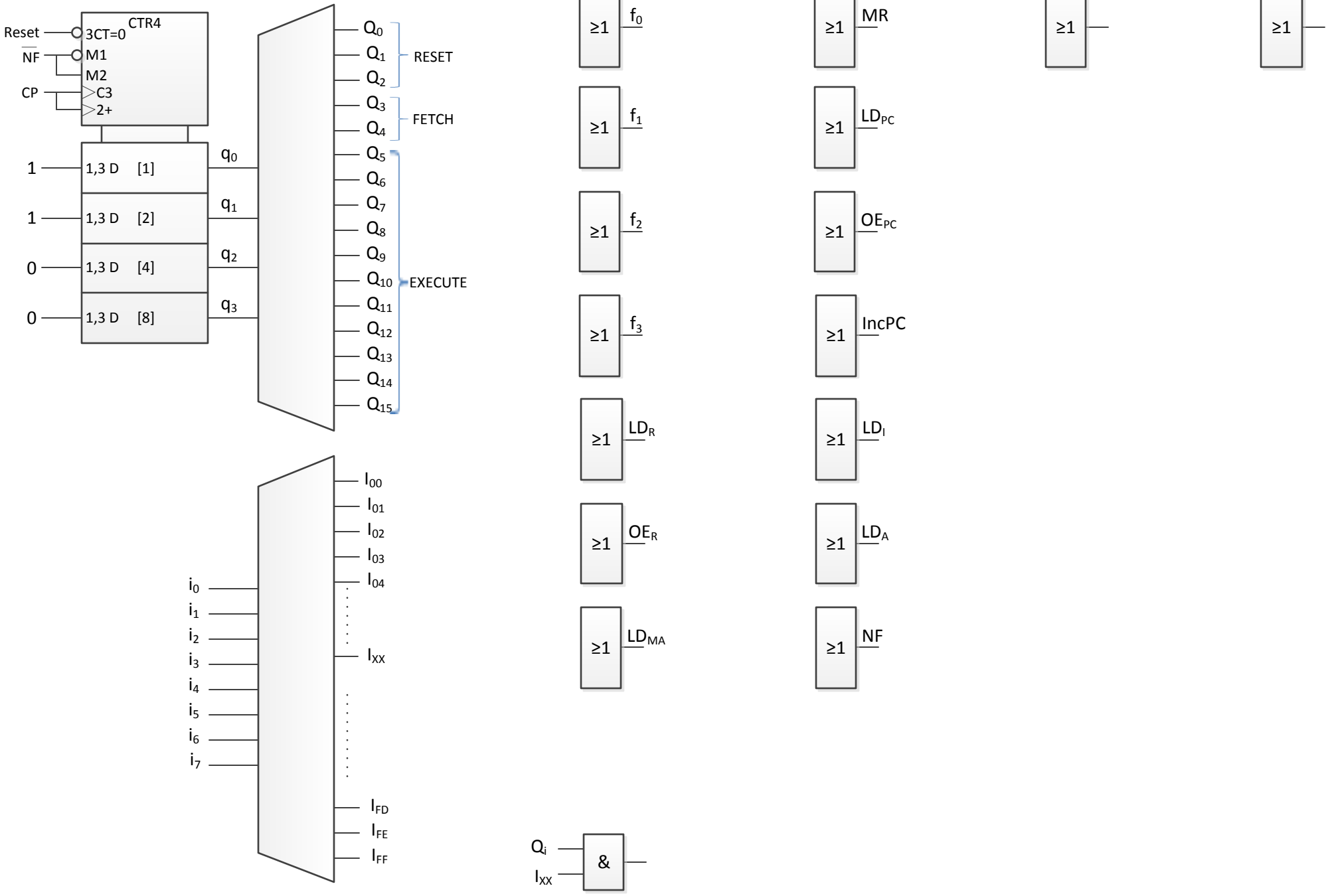


Flaggregister CCR

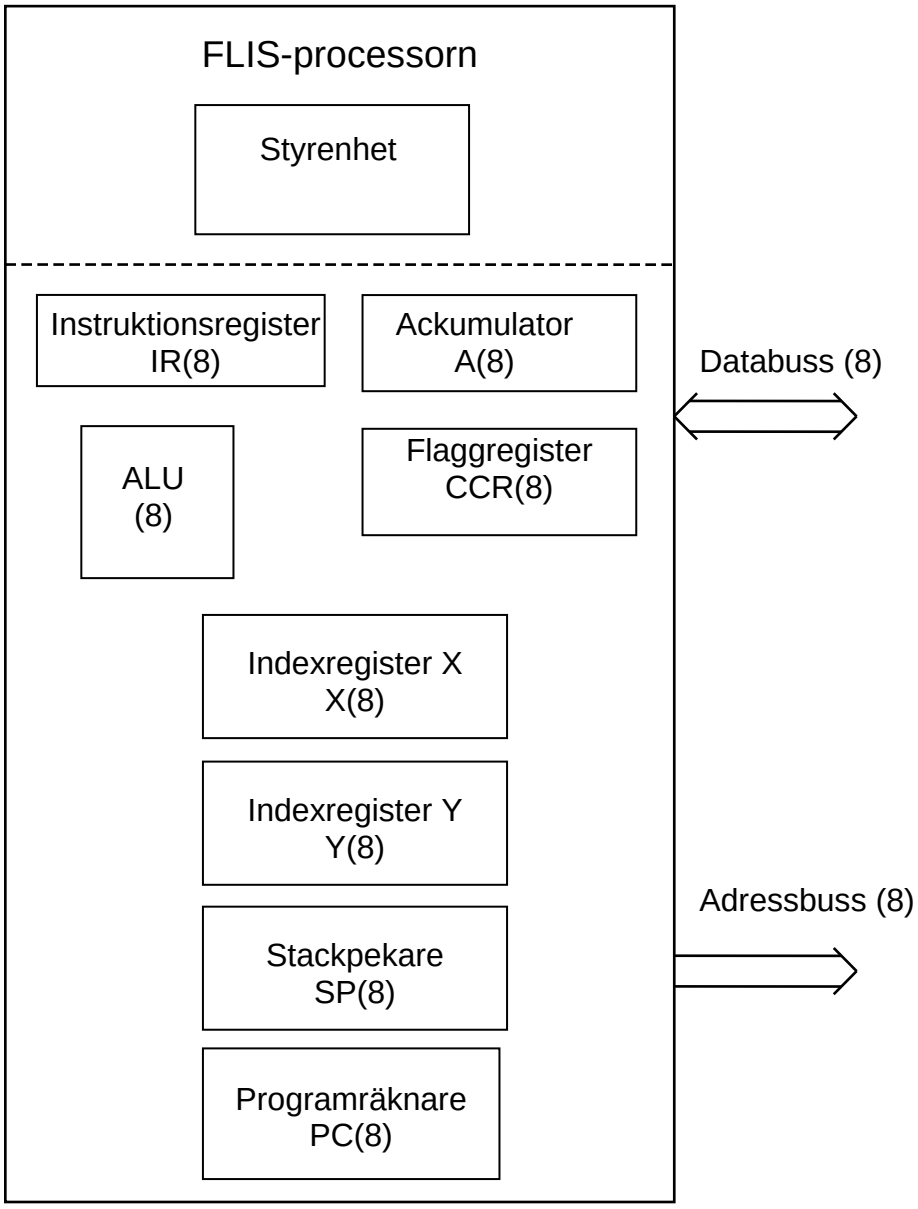
7	6	5	4	3	2	1	0
-	-	-	-	N	Z	V	C



FLEX-datorn i detalj

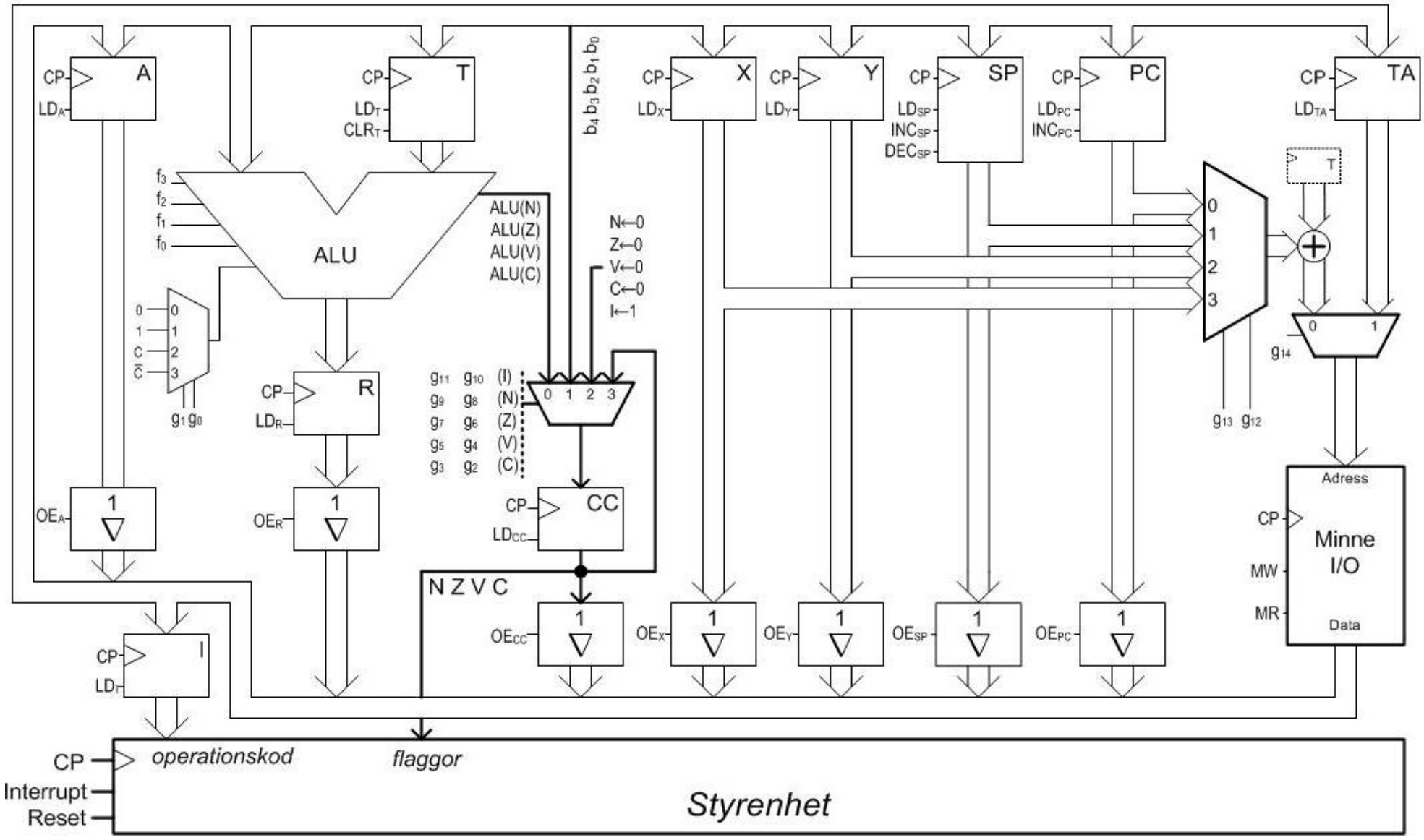

Reset
NF
CP
CTR4
3CT=0
M1
M2
C3
2+
1
1
0
0
1,3 D [1]
1,3 D [2]
1,3 D [4]
1,3 D [8]
q0
q1
q2
q3
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
Q9
Q10
Q11
Q12
Q13
Q14
Q15
RESET
FETCH
EXECUTE
i0
i1
i2
i3
i4
i5
i6
i7
I00
I01
I02
I03
I04
IXX
IFD
IFE
IFF
≥1
f0
f1
f2
f3
LDR
OER
LDMA
MR
LDPC
OEPC
IncPC
LDI
LDA
NF
Qi
IXX
&

Programmerarens bild av FLIS-processorn



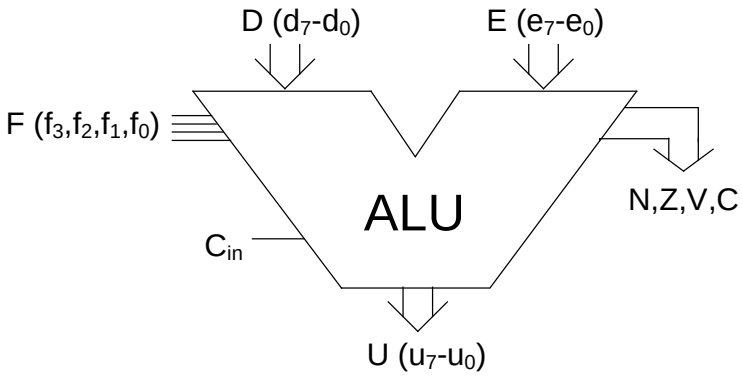
Flaggregister CCR

7	6	5	4	3	2	1	0
-	-	-	I	N	Z	V	C



FLIS-datorn i detalj

FLIS-processorns ALU



funktion				*operation*	*resultat (U)*	*flaggor*			
f_3	f_2	f_1	f_0	*RTN*		N	Z	V	C
0	0	0	0	$0 \rightarrow U$	konstanten 00_{16}	0	1	0	0
0	0	0	1	$FD_{16} \rightarrow U$	konstanten FD_{16}	1	0	0	0
0	0	1	0	$FE_{16} \rightarrow U$	konstanten FE_{16}	1	0	0	0
0	0	1	1	$FF_{16} \rightarrow U$	konstanten FF_{16}	1	0	0	0
0	1	0	0	$E \rightarrow U$	E	u_7	(1)	0	0
0	1	0	1	$D_{1k} + C_{in} \rightarrow U$	$-D - 1 + C_{in}$	u_7	(1)	(9)	(8)
0	1	1	0	$D \vee E \rightarrow U$	bitvis D *OR* E	u_7	(1)	0	0
0	1	1	1	$D \wedge E \rightarrow U$	bitvis D *AND* E	u_7	(1)	0	0
1	0	0	0	$D \oplus E \rightarrow U$	bitvis D *XOR* E	u_7	(1)	0	0
1	0	0	1	$D + C_{in} \rightarrow U$	$D + C_{in}$	u_7	(1)	(2)	(3)
1	0	1	0	$D + FF_{16} \rightarrow U$	$D - 1$	u_7	(1)	(2)	(3)
1	0	1	1	$D + E + C_{in} \rightarrow U$	$D + E + C_{in}$	u_7	(1)	(2)	(3)
1	1	0	0	$D + E_{2k} - C_{in} \rightarrow U$	$D - E - C_{in}$	u_7	(1)	(2)	(10)
1	1	0	1	$D(C_{in}) << 1 \rightarrow U$	$2D + C_{in}$	d_6	(1)	(6)	(4)
1	1	1	0	$(C_{in})\ D >> 1 \rightarrow U$	$2^7 \cdot C_{in} + D/2$	C_{in}	(1)	(7)	(5)
1	1	1	1	$(d_7)\ D >> 1 \rightarrow U$	D/2 (med tecken)	d_7	(1)	0	(5)

Anm:

(1) Z = 1 då samtliga bitar i värdet U är 0. Z = 0 annars.

(2) V = 1 vid overflow enligt reglerna för 2k-aritmetik. V = 0 annars.

(3) C = 1 om summan är större än 255. C = 0 annars.

(4) C = utskiftad bit, dvs. bit d_7.

(5) C = utskiftad bit, dvs. bit d_0.

(6) V = $d_7 \oplus d_6$ dvs. sätts till 1 om skiftet ger teckenbyte.

(7) V = $C_{in} \oplus d_7$ dvs. sätts till 1 om skiftet ger teckenbyte.

(8) C = 0 om D = 00000000_2. C = 1 annars.

(9) V = 1 om D=10000000_2. V = 0 annars.

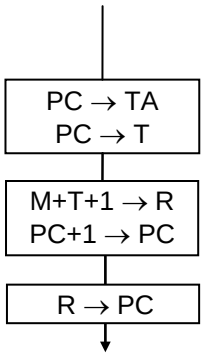
(10) C = 1 om summan är mindre än 256. C = 0 annars.

Implementering ovillkorlig branch-instruktion i den fasta styrenheten för FLISP

BRA Adr Operationsbeskrivning:
PC+Offs → PC

Instruktion: BRA Adr
Format: Ord1: 21_{16}(Opkod) Ord2: Offset

Tillstånd	Summaterm	RTN-beskrivning	Styrsignaler
Q_4	$Q_4 \cdot I_{21}$	PC→TA, PC→T	OE_{PC}, LD_{TA}, LD_T
Q_5	$Q_5 \cdot I_{21}$	M(TA)+T+1→R	MR, f_3, f_1, f_0, g_0, g_{14}, LD_R
Q_6	$Q_6 \cdot I_{21}$	R→PC, (New Fetch)	OE_R, LD_{PC}, NF



Implementering av villkorliga branch-instruktioner med fast styrenhet för FLISP

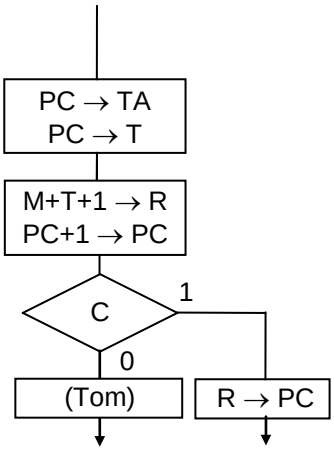
Metodiken för implementering av villkorliga instruktioner belyses av följande exempel:

Ex) BCS Adr Operationsbeskrivning:
If C=1: PC+Offs → PC

Flödesplan:

Instruktion: BCS Adr
Format: Ord1: 28_{16}(Opkod) Ord2: Offset

State	Summaterm	RTN-beskrivning	Styrsignaler
Q_4	$Q_4 \cdot I_{28}$	PC→TA, PC→T	OE_{PC}, LD_{TA}, LD_T
Q_5	$Q_5 \cdot I_{28}$	M(TA)+T+1→R, PC+1→PC	MR, f_3, f_1, f_0, g_0, g_{14}, LD_R IncPC
Q_6	$Q_6 \cdot I_{28} \cdot C$	If C=1: R→PC	OE_R, LD_{PC}
	$Q_6 \cdot I_{28}$	New Fetch	NF



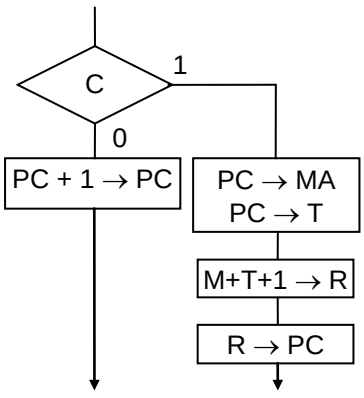
Man kan modifiera den villkorliga branch-instruktionen BCS så att den exekveras enligt flödesplanen nedan till höger. Beskriv EXECUTE-fasen i RTN-tabellen nedan. Använd opkod 03_{16}.

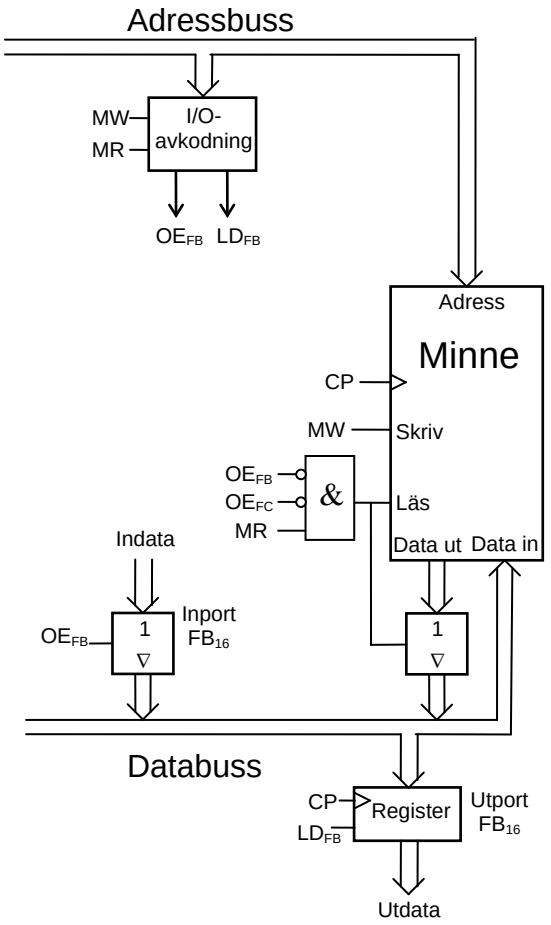
BCS Adr Operationsbeskrivning:
If C=1: PC+Offs → PC

Flödesplan:

Instruktion: BCS Adr
Format: Ord1: 03_{16} (Opkod) Ord2: xx_{16} (offset)

State	Summaterm	RTN-beskrivning	Styrsignaler
Q_4			
Q_5			
Q_6			





På adressen FC_{16} finns också en inport och en utport inkopplade enligt samma princip som portarna på adressen FB_{16}.

FLISP-dator med in- och utport inkopplad på adress- och databussen.

Assemblerspråket för FLIS-processorn.

Assemblerspråket använder sig av mnemoniska beteckningar liknande dem som processorkonstruktören MOTOROLA (FREESCALE) specificerat för maskininstruktioner för mikroprocessorerna 68XX och instruktioner till assemblatorn, s k pseudoinstruktioner eller assemblatordirektiv. Pseudoinstruktionerna listas i tabell 1.

Tabell 1

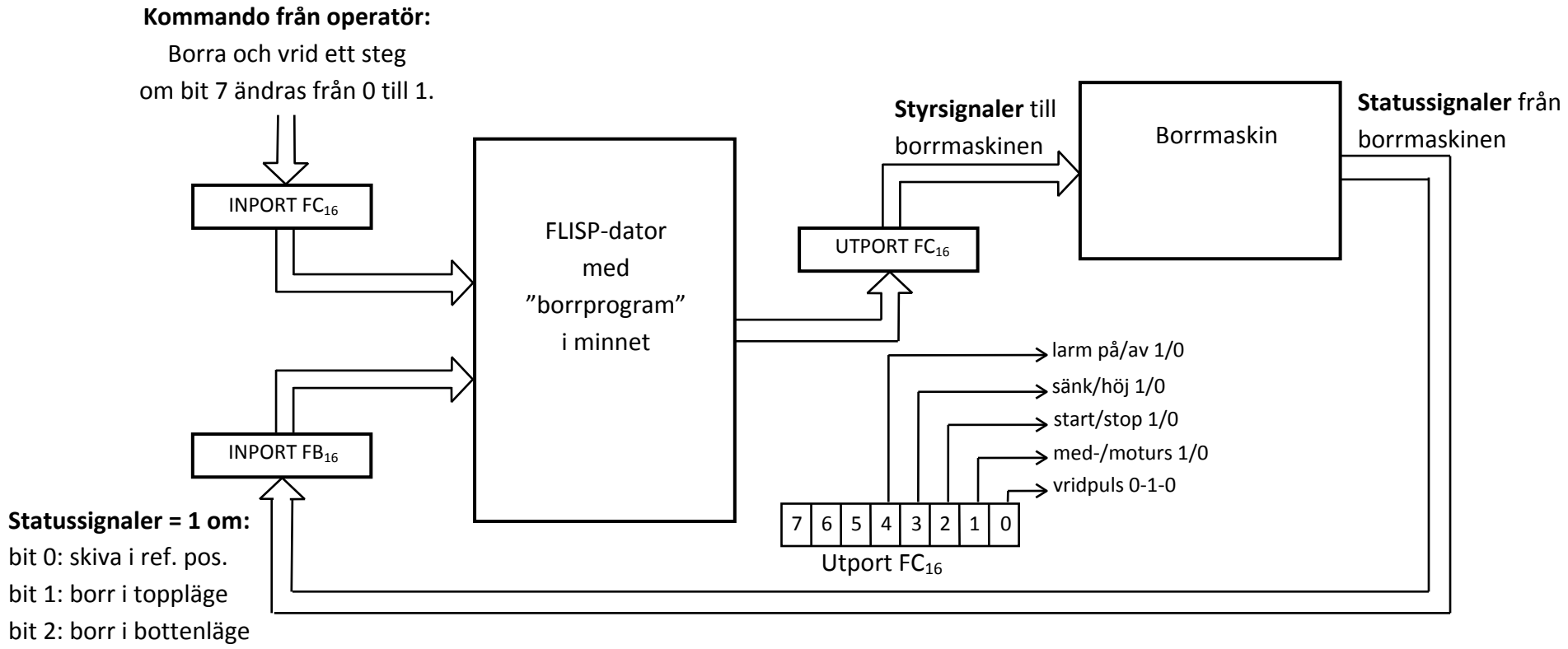
Direktiv			Förklaring
	ORG	N	Placerar den efterföljande koden med början på adress N. (ORG för ORiGin = ursprung)
L	RMB	N	Avsätter N bytes i följd i minnet (utan att ge dem värden), så att programmet kan använda dem. Följden placeras med början på adressen L. (RMB för Reseve Memory Bytes)
L	EQU	N	Ger symbolen L konstantvärdet N. (EQU för EQUates)
L	FCB	N1, N2	Avsätter en byte för varje argument i följd i minnet. Respektive byte ges konstantvärdet N1, N2 etc. Följden placeras med början på adressen L. (FCB för Form Constant Byte)
L	FCS	”ABC”	Avsätter en byte för varje tecken i teckensträngen ”ABC” i följd i minnet. Respektive byte ges ASCII-värdet för A B C, etc. Följden placeras med början på adressen L. (FCS för Form Character String)

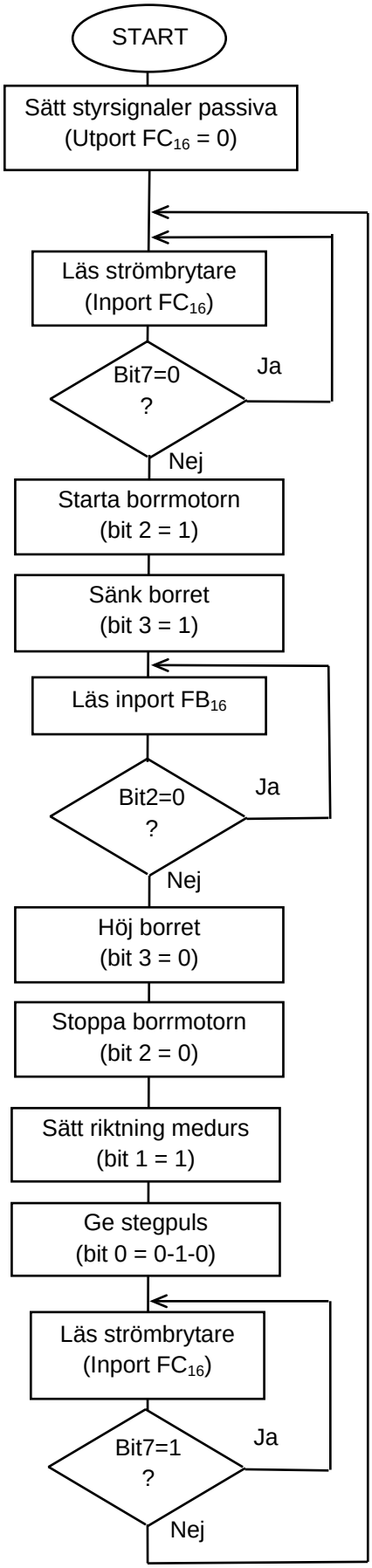
```
;
; I/O-enheter
;
;*******************************************************************************
; Det finns två inportar och två utportar i FLISP-datorn på adresserna $FB OCH $FC
;
; På adressen $FB finns både en inport och en utport. Adressen är det enda som dessa
; har gemensamt, så data som skrivs på t ex utport $FB har inget att göra med data som
; läses på inport $FB.
;
; Det finns ett antal inenheter och utenheter som man kan ansluta till portarna.
;
; Inenheter som kan anslutas till inportar:
; "DIPSWITCH" (FB,FC), med åtta strömbrytare (switchar), som kan ställas in på 0/1.
; "Keyboard" (FB,FC), där koden för en nedtryckt tangent kan avläsas.
; "Drill" (FB), där tre givare från en borrmaskin kan avläsas.
; "Console" (FC), där man kan läsa ASCII-tecken från din dators tangentbord.
;
; Utenheter som kan anslutas till utportar:
; "LED" (FB,FC), med åtta lysdioder (lampor), som kan tändas (1) eller släckas (0).
; "HEXDISPLAY" (FB,FC), där man kan visa ett 8-bitars dataord som två hexsiffror.
; "7-SEGMENT" (FB,FC), där man kan styra varje segment i ett 7-segments sifferfönster.
; "Console" (FB), där man kan skriva ut tecken på en bildskärm.
; "Drill" (FC), där man kan styra olika funktioner hos en borrmaskin.
;*******************************************************************************
;
; Assemblera programmet ("Debug|Assemble") och ladda det i minnet ("Load New").
;
; Anslut inenheten "DIPSWITCH" till inport $FB och utenheten "LED" till utport $FB.
;
; Kör sedan programmet,först instruktion för instruktion med "Step",
; sedan med "Run Slow" och till sist med "Run Fast".
; Ändra på strömbrytarna på "DIPSWITCH" medan du kör och iakttag utenheten.
;
; Stoppa programmet och byt i tur och ordning utenhet till "HEXDISPLAY", "7-SEGMENT"
; och "CONSOLE" och kör programmet på samma sätt med dessa.
;
; Stoppa programmet och byt både inport och utport till adressen $FC, assemblera
; och ladda det.
; Anslut sedan "DIPSWITCH" till inport $FC och "DRILL" till utport $FC och kör
; programmet.
;
;*******************************************************************************
        ORG     0       Programmets startadress
START   LDA     $FB     Läs av de åtta databitarna från ansluten inenhet
        STA     $FB     Mata ut de åtta databitarna till ansluten utenhet
        JMP     START   Upprepa förloppet
STOP                    ;Här är första adressen efter programmet
```

Enkel borrmaskinsstyrning (uppgift 16.16 från arbetsboken, modifierad)

FLIS-datorn skall användas för att styra en komplett borrning av ett hål med borrmaskinen. En flödesplan för borrningen ges på nästa sida. Borrningen skall starta när man med hjälp av en switch ändrar värdet på bit 7 på inport FC_{16} från "0" till "1". När ett hål är färdigborrat skall arbetsstycket vridas ett steg medurs.

Skriv ett program enligt flödesplanen och översätt till maskinspråk. Ange även vilka adresser som de olika instruktionerna placeras på. Startadressen för programmet skall vara $\mathbf{00_{16}}$.



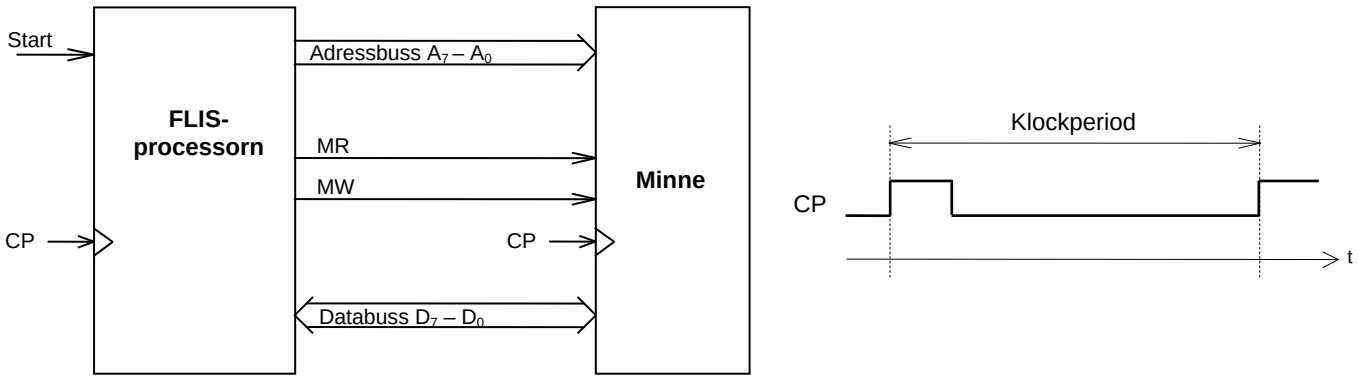

START
Sätt styrsignaler passiva
(Utport FC16 = 0)
Läs strömbrytare
(Inport FC16)
Bit7=0
?
Ja
Nej
Starta borrmotorn
(bit 2 = 1)
Sänk borret
(bit 3 = 1)
Läs inport FB16
Bit2=0
?
Ja
Nej
Höj borret
(bit 3 = 0)
Stoppa borrmotorn
(bit 2 = 0)
Sätt riktning medurs
(bit 1 = 1)
Ge stegpuls
(bit 0 = 0-1-0)
Läs strömbrytare
(Inport FC16)
Bit7=1
?
Ja
Nej

Programförslag:

```
;       Program för borrning av ett hål. Borrning startas när en positiv
;       flank har detekterats på bit 7 på inport $FC.
;       Borrmaskinens styrsignaler är anslutna till utport $FC och
;       statussignaler till inport $FB.

;       Ingen hänsyn har tagits till att borrmaskinen är ett mekaniskt
;       system som reagerar långsamt på givna kommandon. Efter varje
;       kommando borde därför en tillräckligt lång fördröjning läggas in.

;       Efter borrningen vrids arbetsstycket ett steg medurs.
;*************************************************************************
;
;       Anslut "DIP-SWITCH" till inport $FC och borrmaskinen till utport $FC
;       och inport $FB!
;
;*************************************************************************
        ORG     0
        CLR     $FC             Passiva styrsignaler till borrmaskinen

LOOP    TST     $FC             Läs strömbrytare
        BPL     LOOP            Kolla bit7. Vänta om bit7= 0

        LDA     #%00000100      Bit2=1 => Starta motor
        STA     $FC             Starta borrmotorn!

        ORA     #%00001000      Bit3=1 => Sänk borr
        STA     $FC             Sänk borret!

WAIT1   LDA     $FB             Läs borrmaskinens givare
        ANDA    #%00000100      Maska fram "borr i bottenläge"
        BEQ     WAIT1           Vänta tills borr i bottenläge

        LDA     #%00000100      Höj borr med motor på
        STA     $FC             Höj borret!

        ANDA    #%11111011      Bit2=0 => Stäng av motor
        STA     $FC             Stoppa borrmotorn!

;                               Vrid ett steg
        ORA     #%00000010      Bit2=1 => Medurs vridning vid puls på bit0
        STA     $FC             Mata ut vridriktning till borrmaskinen!

        ORA     #%00000001      Bit0=1 => positiv flank på bit0
        STA     $FC             Mata ut puls på bit0 till borrmaskinen 0-1-0
        ANDA    #%11111110
        STA     $FC

WAIT2   TST     $FC             Läs operatörens strömbrytare
        BMI     WAIT2           Vänta tills bit7=0 (borrkommando borta)

        BRA     LOOP
```

Hur FLIS-processorn läser och skriver i minnet

I figur 1 visas de signaler FLIS-processorn använder för läsning och skrivning i minnet.



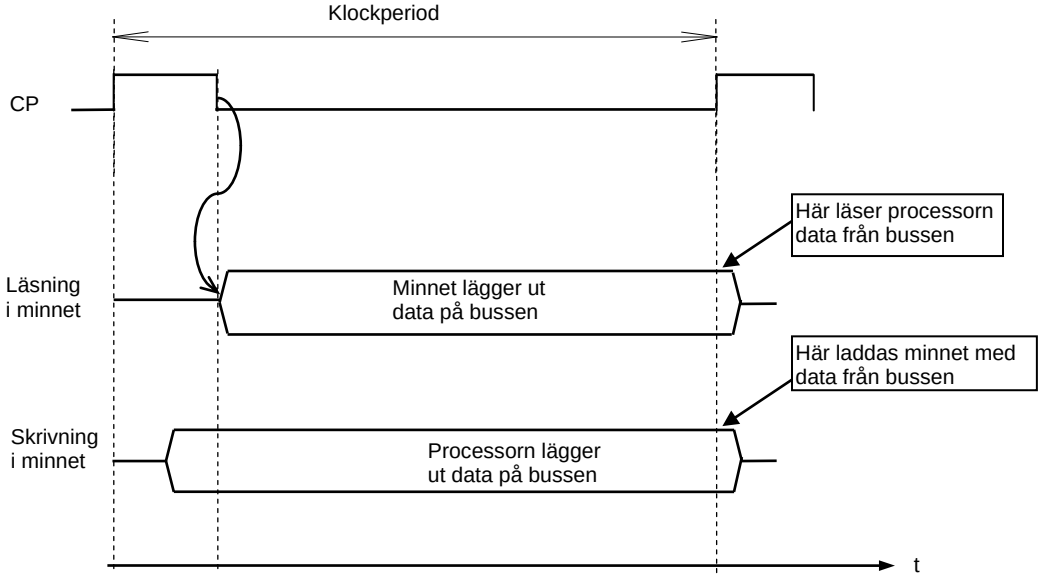
Figur 1 FLIS-processorn kopplad till minnet.

FLISP-processorn läser eller skriver i minnet under en klockperiod. Den inleds när CP går från låg till hög nivå (positiv flank) och avslutas med nästa positiva flank på CP-signalen, såsom visas i figur 1.

En läsning i minnet (en läscykel) inleds med att processorn lägger ut adressen, där läsningen skall ske, på adressbussen samtidigt som den lägger ut hög nivå på signalen MR.

På grund av att det finns interna fördröjningar i processorn så kommer varken adressbitar eller MR-signalen att ha rätt värde förrän ***efter*** CP-signalens positiva flank. Dessutom kommer de inte att få rätt värde samtidigt på grund av att fördröjningarna är olika för de olika signalerna. Vid tidpunkten när CP-signalen går låg igen är dock adressen och MR-signalen garanterat korrekta (stabila) och först då skall minnet börja "plocka" fram data från den aktuella adressen och placera det på databussen, vilket visas i figur 2. Processorn läser sedan av data från databussen vid slutet av klockcykeln dvs. vid CP-signalens nästa positiva flank.

Ett liknande resonemang kan föras för adresser och MW-signalen vid skrivning. Skrivningen inleds med att processorn lägger ut adressen, där skrivningen skall ske, på adressbussen samtidigt som signalen MW ges värdet "1". Därefter lägger processorn ut data på databussen på det sätt som visas i figur 2. Minnet laddas sedan med data från databussen vid slutet av klockcykeln dvs vid CP-signalens nästa positiva flank.

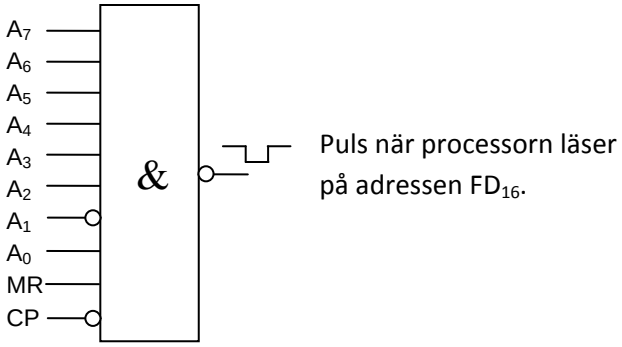


Figur 2 Läsning och skrivning i minnet med FLIS-processorn

Vid läsning och skrivning är således adressen samt MR- resp. MW-signalen, som läggs ut från processorn vid CP-signalens positiva flank, inte garanterat korrekta (stabila) förrän vid CP-signalens negativa flank. Adressen och MR- eller MW-signalen har alltså stabila värden under tiden som CP är "0".

Krets för detektering av läsning på minnesadressen FD_{16}.

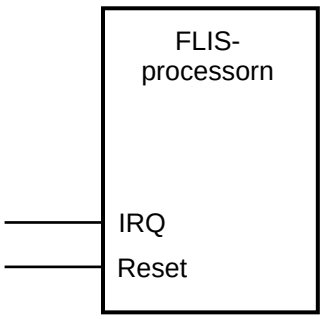
FD_{16} = 1111 1101_2



Yttre styrning (Ver 2013-05-07)

Yttre signaler som påverkar exekveringen hos FLIS-processorn

Nedan beskrivs två yttre styrsignaler som styr exekveringen hos FLIS-processorn.



1. Reset
När en etta utifrån läggs på Reset-ingången upphör processorns arbete.

När sedan Reset-ingången nollställs händer följande:

I-biten (I-flaggan) i CC-registret ettställs och övriga flaggor nollställs. Att I-biten ettställs innebär att FLIS-processorns avbrottssystem är avstängt. (Se punkt 2 nedan!)

Processorn läser sedan innehållet i den s k "resetvektorn" på minnesadressen FF_{16} och placerar det lästa 8-bitars dataordet i programräknaren (PC). Sedan hämtar processorn nästa instruktion från denna adress.

Genom att aktivera "Reset"-signalen kan man alltså få processorn att starta om program-exekveringen från den adress som finns lagrad i "resetvektorn" på adressen FF_{16} i minnet. Reset-signalen aktiveras alltid när matningsspänningen till processorn slås på från början.

2. IRQ (Interrupt request, Avbrottsbegäran)
En yttre enhet kan få processorn att (oftast tillfälligt) byta program genom att lägga en etta på insignalen IRQ. Processorn exekverar då färdigt den instruktion den håller på med. Om avbrottssystemet är aktiverat, dvs. om I-biten i flaggregistret (CC-registret) är nollställd, kommer processorn sedan att acceptera avbrottet. I annat fall kommer programexekveringen att fortsätta som vanligt.

Om avbrottsbegäran accepteras sparar processorn innehållen i de interna registren i ordningen PC, Y, X, A och CC på stacken. (Innehållet i PC är här adressen till nästa instruktion i det avbrutna programmet.) Därefter ettställs I-flaggan i CC-registret för att nya avbrott skall utestängas.

Processorn läser sedan innehållet i den sk "IRQ-vektorn" på minnesadressen FD_{16}. Detta innehåll, som är adressen till en avbrottsrutin, placeras i programräknaren. Därmed har processorn utfört ett hopp från det vanliga programmet till avbrottsrutinen.

Yttre styrning (Ver 2013-05-07)

När avbrottsrutinen börjar exekveras vet man att den händelse som orsakar avbrott har inträffat. Avbrottsrutinen skall därför utföra arbetet som är förknippat med avbrottshändelsen och sedan återvända till det vanliga (avbrutna) programmet med återställda registerinnehåll. Det finns en speciell instruktion, RTI (Return from interrupt), som återställer innehållen i samtliga processorregister genom att läsa de gamla registerinnehållen från stacken. Eftersom RTI även återställer innehållet i PC kommer exekveringen efter RTI att fortsätta i det program som tidigare blev avbrutet. En avbrottsrutin avslutas därför med instruktionen RTI.

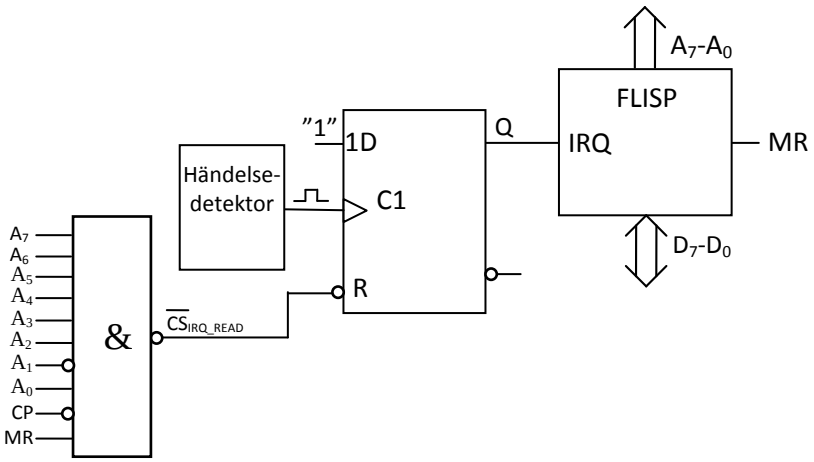
Vid återhoppet från avbrottsrutinen är det viktigt att den aktiva signal som orsakade det pågående avbrottet har försvunnit, annars kommer ju ett nytt avbrott direkt att genereras av den "gamla" signalen efter återhoppet.

Lägg märke till att avbrottssystemet automatiskt aktiveras då de gamla registerinnehållen i processorn återställs vid återhopp från avbrottsrutinen eftersom det gamla CC-innehållet, som hämtas från stacken, innehåller en nolla i I-biten.

Nedan visas hur några olika vektorer och in-/utportar är placerade i processorns adressrum.

FB_{16}: I/O-portar (Inport och utport)
FC_{16}: I/O-portar (Inport och utport)
FD_{16}: IRQ-vektor (För adress till avbrottsrutin.)
FE_{16}: TRAP-vektor (Adress för hantering av otillåten operationskod.)
FF_{16}: RESET-vektor (För startadress)

Standardkoppling för avbrottsgenerering till FLIS-processorn:



Exempel på avbrott med FLISP

Ett datorsystem med FLIS-processorn skall användas för att köra ett huvudprogram.

Samtidigt som huvudprogrammet körs skall en variabel KNAPP (8 bitar) på minnesadressen 60_{16} ökas med ett varje gång en tryckknapp aktiveras.

Ökningen av variabeln KNAPP skall ske med hjälp av IRQ-avbrott genom att huvudprogrammet avbryts vid varje knapptryckning och en avbrottsrutin KNPINC då körs. Se figuren på nästa sida.

Avbrottsrutinen KNPINC skall ha startadressen 30_{16} i minnet.

Avbrottsvektorn för IRQ-avbrott finns i RWM (läs- och skrivbart minne) på adressen FD_{16}.

Avbrottsrutinen på adressen KNPINC (30_{16}) blir:

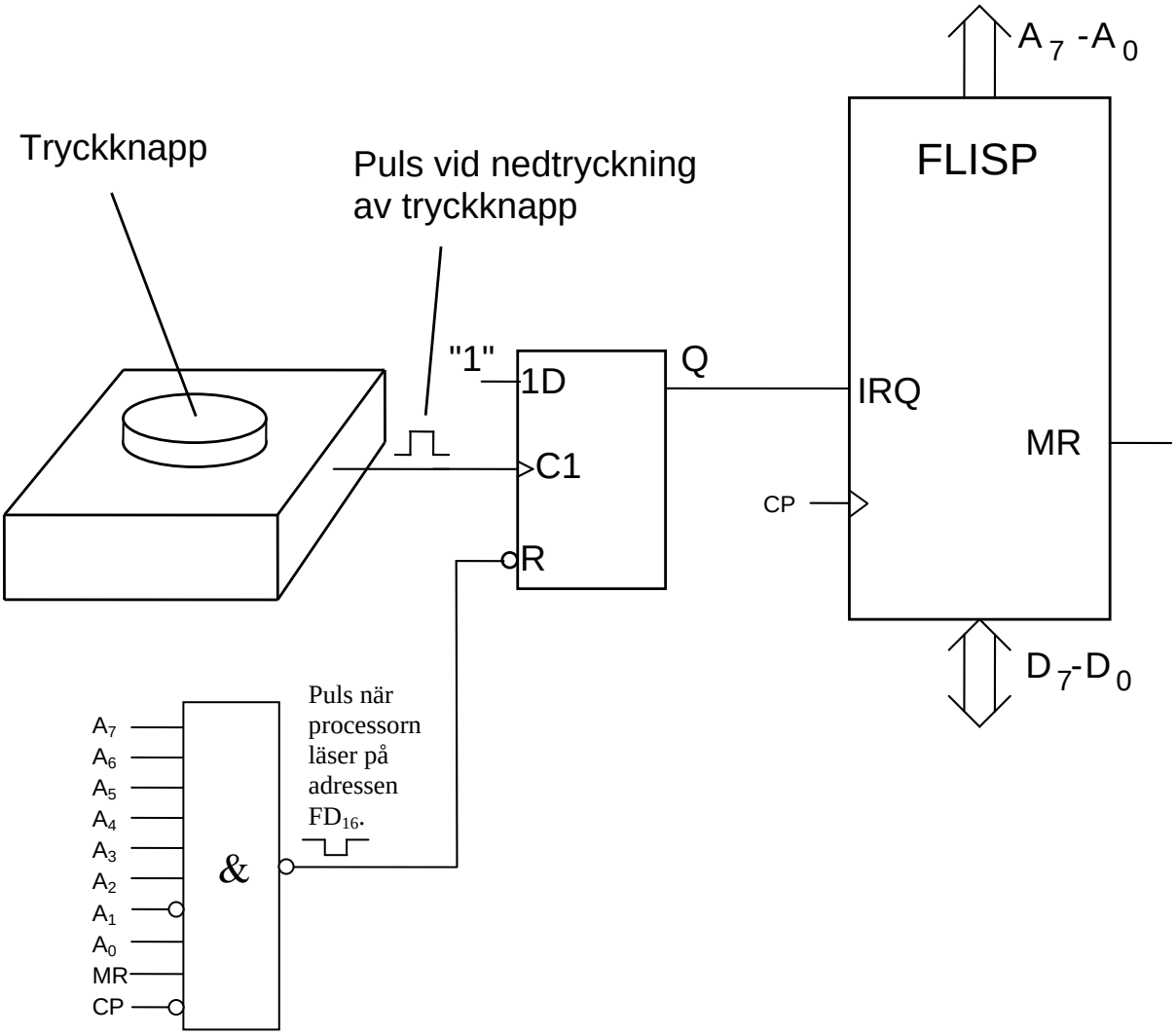
```
KNPINC  INC     $60     Variabel KNAPP ökas med ett.
        TST     $FD     Nollställ avbrottsvippan.
        RTI             Återvänd till huvudprogrammet.
```

Nedan visas nödvändiga initieringar i huvudprogrammet för att avbrott på IRQ-ingången skall accepteras. Dessutom visas hur IRQ-vektorn ges sitt rätta värde KNPINC (60_{16}).

Huvudprogram:

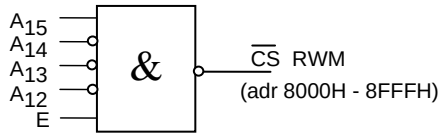
```
MAINPG  LDSP    #STACKADR   BOS definieras först.

        LDX     #KNPINC     Adr KNPINC = 30(16)
        STX     $FD         Ge IRQ-vektorn värdet KNPINC

        TST     $FD         Nolla avbrottsvippan

        CLR     $60         Nolla variabeln KNAPP (60(16))

        ANDCC   #%11101111  Aktivera avbrottssystemet
;         .                 (Nolla I-biten i CC-registret)
          .
```

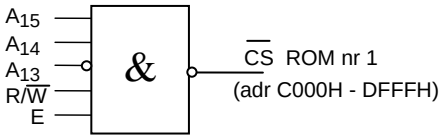
Yttre styrning (Ver 2013-05-07)

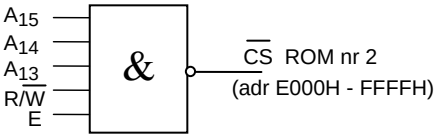


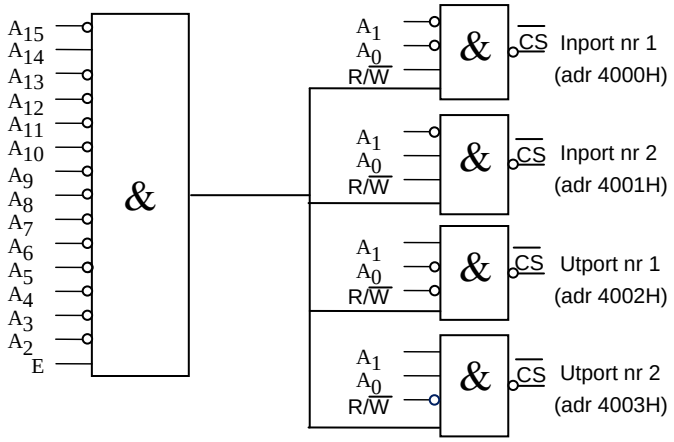
Fullständig adressavkodning

```
Inport 1:          4000H = 0100 0000 0000 0000
Inport 2:          4001H = 0100 0000 0000 0001
Utport 1:          4002H = 0100 0000 0000 0010
Utport 2:          4003H = 0100 0000 0000 0011

4k RWM     Start:  8000H = 1000 0000 0000 0000

           Slut:   8FFFH = 1000 1111 1111 1111

8k ROM #1  Start:  C000H = 1100 0000 0000 0000

           Slut:   DFFFH = 1101 1111 1111 1111

8k ROM #2  Start:  E000H = 1110 0000 0000 0000

           Slut:   FFFFH = 1111 1111 1111 1111
```









Ofullständig adressavkodning

```
Inport 1:          4000H = 0100 0000 0000 0000
Inport 2:          4001H = 0100 0000 0000 0001
Utport 1:          4002H = 0100 0000 0000 0010
Utport 2:          4003H = 0100 0000 0000 0011

4k RWM     Start:  8000H = 1000 0000 0000 0000

           Slut:   8FFFH = 1000 1111 1111 1111

8k ROM #1  Start:  C000H = 1100 0000 0000 0000

           Slut:   DFFFH = 1101 1111 1111 1111

8k ROM #2  Start:  E000H = 1110 0000 0000 0000

           Slut:   FFFFH = 1111 1111 1111 1111
```

Var hittar man modulerna i adressrummet i detta fall?

```
Inport 1: Alla adresser 4*n,   där n = 0-1FFFH
Inport 2: Alla adresser 4*n+1, där n = 0-1FFFH
Utport 1: Alla adresser 4*n+2, där n = 0-1FFFH
Utport 2: Alla adresser 4*n+3, där n = 0-1FFFH
```

```
4k RWM: 8000-8FFFH
        9000-9FFFH
        A000-AFFFH
        B000-BFFFH
```

Med denna adressavkodning ser vi att modulerna inte får några överlappande adressområden, men I/O-modulerna och RWM kommer att ”synas” i flera adressområden.

```
8k ROM #1: C000-DFFFH

8k ROM #2: E000-FFFFH
```

Version 2013-05-07

Flyttal komplettering (32-bitars)

Format: s / c / f
Antal bitar: 1 8 23

s är teckenbit: 0 motsvarar + och 1 motsvarar –.

c är karakteristika: c = exp + 127.

f kallas fraction och är normaliserade mantissan utan inledande 1.

Högsta värde (255) och lägsta värde (0) på karakteristikan är reserverade för ∞ resp. 0.
± ∞: s / 255 / 0
± 0: s / 0 / 0 (Obs! i detta fall sätts inledande mantissabiten = 0.)

Talområde för exponenten: $-126 \leq exp \leq 127$

Största belopp blir: $1.111\ 1111\ 1111\ 1111\ 1111\ 1111 \cdot 2^{+127} \approx 2^{+128}$

Minsta belopp (förutom 0) blir: $1.000\ 0000\ 0000\ \ldots 0 \cdot 2^{-126} = 2^{-126}$

För att kunna representera tal med ännu mindre belopp än minsta beloppet ovan har man infört undantaget nedan, som dock ger sämre upplösning än normalfallet.

Undantag:

Man kan i uttrycket för ”0” ovan välja $f \neq 0$.

Största belopp skulle i detta fall bli:

$0.111\ldots1 \cdot 2^{-127} = (1 - 2^{-23}) \cdot 2^{-127}$

Det uppstår då ett ”glapp” till minsta beloppet (2^{-126}) i "normalfallet" ovan. För att undvika ”glappet” bestämmer man att exponenten skall sättas till –126 istället för –127 då c = 0.

Man får då största beloppet = $(1 - 2^{-23}) \cdot 2^{-126}$ i detta fall.

Exempel på heltalsmultiplikation för tal utan tecken: $P = X \cdot Y$

P: Produkt

X: Multiplikator $X = 1101_2$

Y: Multiplikand $Y = 1011_2$

				1	0	1	1	
				1	1	0	1	
	1	1	1	1	0	1	1	
			0	0	0	0		
		1	0	1	1			
+	1	0	1	1				
1	0	0	0	1	1	1	1	

	Add multiplikand				Multiplikator			
	0	0	0	0	1	1	0	1
+	1	0	1	1				
→	1	0	1	1	1	1	0	1
	0	1	0	1	1	1	1	0
+	0	0	0	0				
→	0	1	0	1	1	1	1	0
	0	0	1	0	1	1	1	1
+	1	0	1	1				
→	1	1	0	1	1	1	1	0
	0	1	1	0	1	1	1	1
+	1	0	1	1				
→ 1	0	0	0	1	1	1	1	1
	1	0	0	0	1	1	1	1

Exempel på heltalsdivision för tal utan tecken: Y/X = Q + R/X

Q: Kvot

X: Divisor $X = 1010_2$

Y: Dividend $Y = 1001\ 1000_2$

R: Rest

```
  0 0 0 0 1 1 1 1
  ---------------
  1 0 0 1 1 0 0 0 | 1 0 1 0
- 1 0 1 0
  ---------
  0 1 0 0 1 0
-   1 0 1 0
    ---------
    0 1 0 0 0 0
-     1 0 1 0
      ---------
      0 0 1 1 0 0
-       1 0 1 0
        ---------
        0 0 0 1 0
```