Grundläggande Datorteknik
Digital- och datorteknik

Kursens mål:

Fatta hur en dator är uppbyggd (HDW)
Fatta hur du du programmerar den (SW)
Fatta hur HDW o SW samverkar

Digital teknik

Dator teknik

Lärandemål:

1) Talsystem, binära koder och datoraritmetik

1. Konvertering mellan olika talsystem
2. Utifrån given problemställning applicera binära koder så som NBC, NBCD, ASCII, Gray, Excess, tecken/belopp och komplementkoder.
3. Redogöra för och tillämpa binär aritmetik (addition och subtraktion).

Talsystem
Basen 2, 8, 10, 16

Positionssystem
Ex: 214
(421)

$\beta = 2$	$\beta = 8$	$\beta = 10$	$\beta = 16$
binärt	oktalt	decimalt	hexa-decimalt
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F
10000	20	16	10
10001	21	17	11

GRAY-KOD

S2.16

Tabell 2.2. Graykoder.

Decimal ordning	Kodord i trebitars Graykod	Kodord i fyrbitars Graykod
0	000	0000
1	001	0001
2	011	0011
3	010	0010
4	110	0110
5	111	0111
6	101	0101
7	100	0100
8		1100
9		1101
10		1111
11		1110
12		1010
13		1011
14		1001
15		1000

NBCD-kod

S2.19

Skriv (563,782)
på NBCD-kod.

Decimal siffra	NBCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Alfanumeriska koder

S2.29

ASCII-koden.

0	1	2	3	4	5	6	7	$b_6b_5b_4$ / $b_3b_2b_1b_0$	
000	001	010	011	100	101	110	111		
NUL	DLE	SP	0	@	P	`	p	0000	0
SOH	DC1	!	1	A	Q	a	q	0001	1
STX	DC2	"	2	B	R	b	r	0010	2
ETX	DC3	#	3	C	S	c	s	0011	3
EOT	DC4	$	4	D	T	d	t	0100	4
ENQ	NAK	%	5	E	U	e	u	0101	5
ACK	SYN	&	6	F	V	f	v	0110	6
BEL	ETB	'	7	G	W	g	w	0111	7
BS	CAN	(	8	H	X	h	x	1000	8
HT	EM	)	9	I	Y	i	y	1001	9
LF	SUB	*	:	J	Z	j	z	1010	A
VT	ESC	+	;	K	[Ä	k	{ä	1011	B
FF	FS	,	<	L	\Ö	l	|ö	1100	C
CR	GS	-	=	M	]Å	m	}å	1101	D
S0	RS	.	>	N	^	n	~	1110	E
S1	US	/	?	O	_	o	RUBOUT (DEL)	1111	F

Ex
"A"=41_{16}

Excess-kod

S2.15

Tabell 2.1. Excess-2^{n-1} kodning vid n = 4.

Nivå k	Kodord i excess-2^{n-1}-kod (n=4)
-8	0000
-7	0001
-6	0010
-5	0011
-4	0100
-3	0101
-2	0110
-1	0111
0	1000
1	1001
2	1010
3	1011
4	1100
5	1101
6	1110
7	1111

Ex avkoda:
$1\ 1\ 0\ 0_2$ Excess-2^{n-1}

n=4

$2^3+2^2+0+0 \quad -2^3 = 4_{10}$

Def 2-Komplement:

Pos: Y = Y **Neg: (-Y) = 2^n - |Y| = Y_{2K}**

Att tvåkomplementera: Ex 4 bit: $2^n=2^4=16$

2^n-Y = 2^n-1-Y +1 = 16-1-Y+1 = 15-Y+1 (= Y_{1K}+1)

Ex Y=6: 0110 Hitta (-Y)

15_{10}	1111
- Y	- 0110
= Y_{1K}	=1001

INVERSEN! **Def 1komp! Y_{1K}**

addera 1	+0001
= Y_{2K}	=1010

Y_{1K} +1= Y_{2K} **(-Y = 1010)** (-6 = 1010)

Att subtrahera:

X-Y = X + Y_{2K} = X + Y_{1K} +1

Def Flaggor

Statusflaggor ut från ALU:n som indikerar om resultatet blev rätt eller fel

C	**Carry**	**Tal utan tecken**	**[0,15]** **(ADD : minnessiffra; SUB: lånesiffra)**
V	**Overflow**	**Tal med tecken**	**[-8,7]**
N	**Negative**	**Tal med tecken**	**[-8,7]**
Z	**Zero**	**Tal med och utan tecken**	

C=1: Resultatet av operationen blev ***fel*** för en operation ***utan tecken***
V=1: Resultatet av operationen blev ***fel*** för en operation ***med tecken***
N=1: Resultatet av operationen blev ***negativt*** för en operation ***med tecken***
Z=1: Resultatet av operationen blev ***noll***

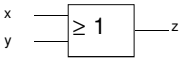
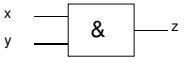
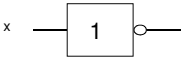
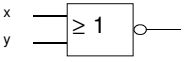
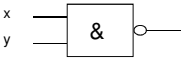
Lärandemål:

2) Digitalteknik

1. Definiera grundläggande logiska operationer och dess motsvarande logiska grindar.
2. Tillämpa den booleska algebrans räknelagar.
3. Utföra algebraisk förenkling av booleska uttryck.
4. Visa likhet/olikhet mellan booleska uttryck.

Logikkretssymboler för grundläggande logikoperationer.

S3.18
S1.14

Funktion	*Grind*	*Grafisk symbol*
z = x+y	ELLER (OR)	
z = x·y	OCH (AND)	
z = x'	INVERTERARE (ICKE, NOT)	
z = (x+y)'	NOR	
z = (x·y)'	NAND	

Sammanfattning

- **Boolesk algebra**
  1+0=1; 1•0=0; 1+1=1; 1•1=1; 1'=0; 0'=1
  de Morgans lagar
- **Funktionstabell**
  Sätt upp tabell med alla kombinationer av invariabler
- **Binär evaluering**
  Gör kolumner för delutrycken

Rep: Satser inom Boolesk algebra.

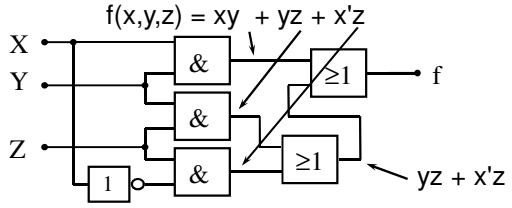
1. Kommutativa lagarna
   $x + y = y + x$
   $x \cdot y = y \cdot x$
2. Distributiva lagarna
   $x \cdot (y + z) = x \cdot y + x \cdot z$
   $x + (y \cdot z) = (x + y) \cdot (x + z)$
3. $x + 0 = x$
   $x \cdot 1 = x$
4. $x + x' = 1$
   $x \cdot x' = 0$
5. $x + 1 = 1$
   $x \cdot 0 = 0$
6. $x + x = x$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

Lärandemål:

2) Digitalteknik - *Kombinatoriska nät*

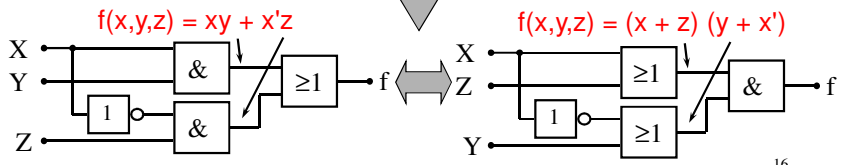
1. Realisera logiska uttryck med grindnät.
2. Beskriva, analysera och konstruera kombinatoriska nät med hjälp av funktionstabeller och boolesk algebra.
3. Kunna minimera logiska uttryck för realisering i kombinatoriska nät.

Grindnät för exempel Kalle



Disjunktiv (minimal) form
(*Summa av Produkter*)

Konjunktiv (minimal) form
(*Produkt av summor*)



Sammanfattning

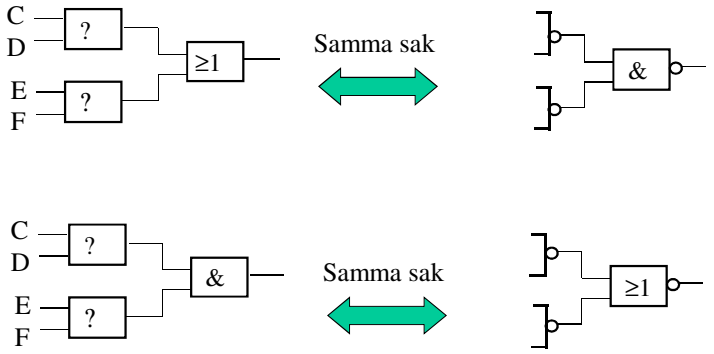
•NORMAL form ⇒ Funktionstabell
•MINIMAL form ⇒ Kranaughdiagram

•DISJUNKTIV (*normal / minimal*) form
- •Σ av Prod Ex: (x'y)+(xw)+(xyw)
- •Ettor
- •Mintermer: (1• 1• 1) = 1
- •NAND / NAND - logik

•KONJUNKTIV (*normal / minimal*) form
- •Prod av Σ:or Ex: (x+z)(x'+z+w)(z'+w')
- •Nollor
- •Maxtermer: (0+0+0) = 0
- •NOR / NOR - logik

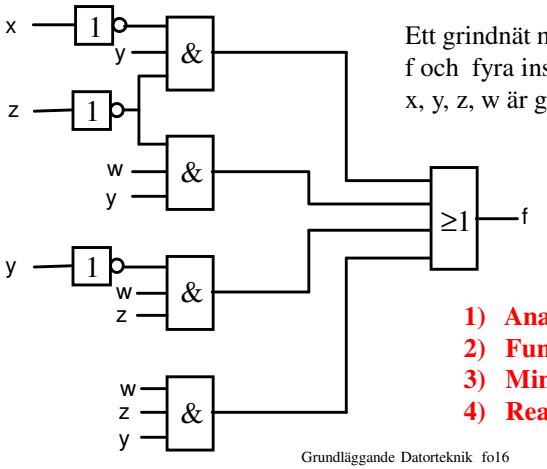
DSP E eller LEEDS P

Sammanfattning *NAND- och NOR-logik*



Praktikfall, minimering av grindnät

Ext4

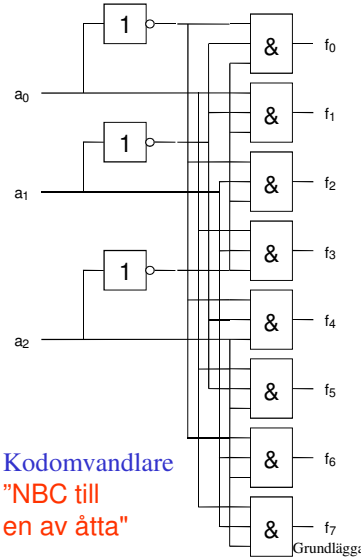


Ett grindnät med utsignalen f och fyra insignalerna x, y, z, w är givet.

Kan man konstruera ett "mindre" nät?

1) **Analys**
2) **Funktionstabell**
3) **Minimering (Karnaugh)**
4) **Realicering**

S4.11



$f_0 = a_2'a_1'a_0'$

$f_1 = a_2'a_1'a_0$

$f_2 = a_2'a_1a_0'$

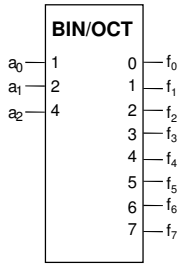
$f_3 = a_2'a_1a_0$

$f_4 = a_2a_1'a_0'$

$f_5 = a_2a_1'a_0$

$f_6 = a_2a_1a_0'$

$f_7 = a_2a_1a_0$



Kodomvandlare
"NBC till
en av åtta"

En delmängd av veckans mål:

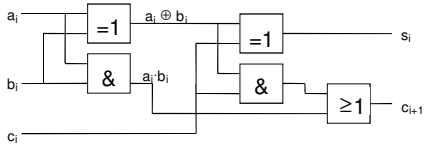
Fo4

- Konstruktruera de olika kombinatoriska nät som ingår i en dator. Exempel på sådana nät är väljare, kodomvandlare och ALU (beräkningsenheten i processorn).
- Studera hur addition/subtraktion utförs

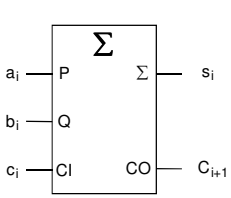
Dagens mål:

- Kodomvandlare *(en kod IN → annan kod UT)*
- Don't care – termer *(ger färre grindar)*
- Väljare *(många signaler IN + styrsig → en signal UT)*
- Fördelare *(en signal IN+styrsig → många signaler UT)*
- Heladderare *(adderar $x+y+c_{in}=s_{ut}$ och c_{ut})*
- Koda tal *(2-komplementsrepresentationen)*

Läs mindre!
Lär dig mer!



Prosamsymbolen för en heladderare.



$$\begin{array}{ll} C_n & c_{n-1}c_{n-2}\ldots c_i\ldots c_1 \\ & a_{n-1}a_{n-2}\ldots a_i\ldots a_1a_0 \\ + & b_{n-1}b_{n-2}\ldots b_i\ldots b_1b_0 \\ \hline & s_{n-1}s_{n-2}\ldots s_i\ldots s_1s_0 \end{array}$$

Arb s 32

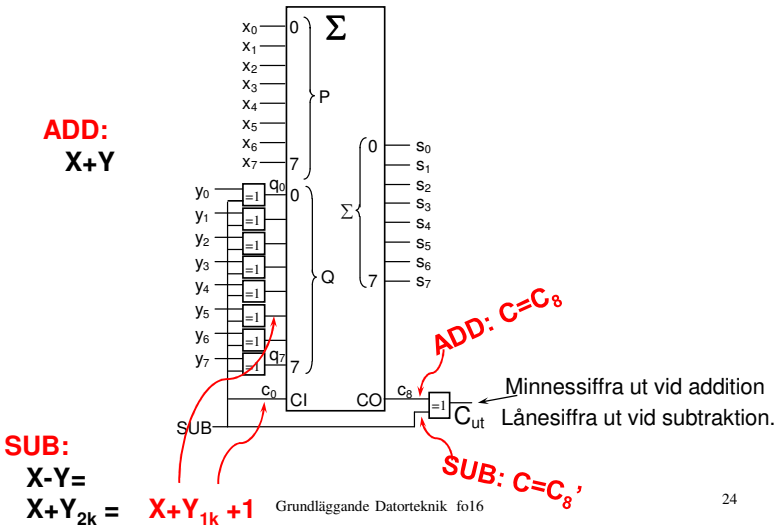
Def Flaggor

Statusflaggor ut från ALU:n som indikerar om resultatet blev rätt eller fel

C	**Carry**	**Tal utan tecken**	**[0,15]**	**(ADD : minnessiffra; SUB: lånesiffra)**
V	**Overflow**	**Tal med tecken**	**[-8,7]**	
N	**Negative**	**Tal med tecken**	**[-8,7]**	
Z	**Zero**	**Tal med och utan tecken**		

C=1: Resultatet av operationen blev *fel* för en operation *utan tecken*
V=1: Resultatet av operationen blev *fel* för en operation *med tecken*
N=1: Resultatet av operationen blev *negativt* för en operation *med tecken*
Z=1: Resultatet av operationen blev *noll*

Ext6



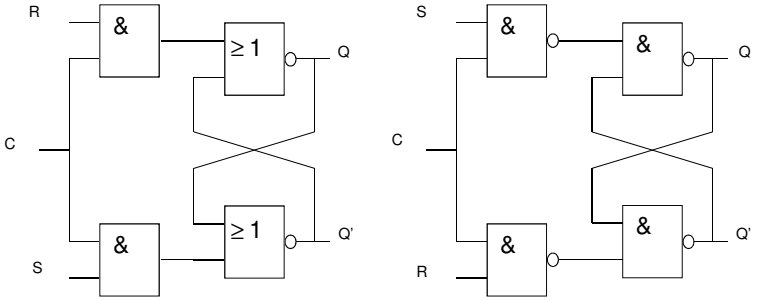
Lärandemål:

2) Digitalteknik - *Sekvensnät*

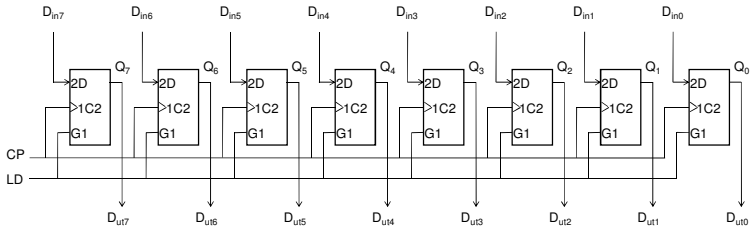
1. Analysera och konstruera synkrona tillståndsmaskiner med hjälp av tillståndstabeller och tillståndsgrafer.
2. Använda D-, T- och JK- vippor för konstruktion av minneselement och räknare.

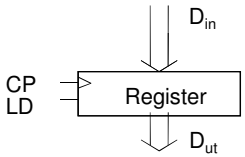
Grindad SR-latch

Ofta förses SR-latchar med en tredje ingång, till vilken en **styrpuls C** ansluts
Härvid erhålls en s k **grindad SR-latch** .



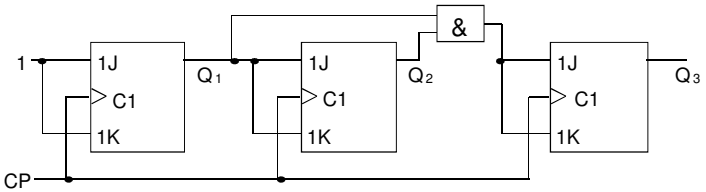
Register och bussar





Arbetsgång - analys räknare

s5.25 ->



1 Studera kopplingen och **bestäm vippornas insignaler** (T_1=, T_2=, T_3=)

2 Sätt upp en tabell med
- "**Detta tillstånd**" (Alla kombinationer av Q_1, Q_2, Q_3)
- **Insignaler** (T_1, T_2, T_3)
- "**Nästa tillstånd**" (Q_1^+, Q_2^+, Q_3^+)

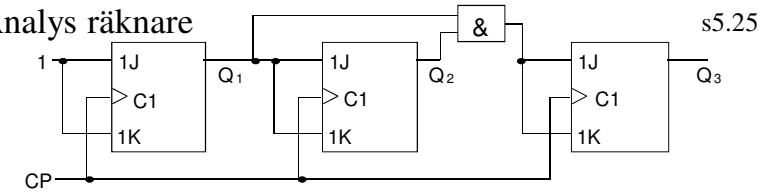
3 Ange insignalernas värden i tabellen och----

4 ange vad "nästa tillstånd" blir

5 Rita slutligen en **tillståndsgraf**

Analys räknare

s5.25



1)

$T_1=1$

$T_2= Q_1$

$T_3= Q_1 Q_2$

Funktionstabell

T	Q^+
0	Q
1	Q'

2)

Detta Tillstånd			Insignaler			Nästa Tillstånd		
Q_3	Q_2	Q_1	T_3	T_2	T_1	Q_3^+	Q_2^+	Q_1^+
0	0	0	0	0	1	0	0	1
0	0	1	0	1	1	0	1	0
0	1	0	0	0	1	0	1	1
0	1	1	1	1	1	1	0	0
1	0	0	0	0	1	1	0	1
1	0	1	0	1	1	1	1	0
1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	0	0	0

3) **4)**

Utsignaler								Insignaler							
Detta tillstånd Q				Nästa tillstånd Q^+											
q3	q2	q1	q0	$q3^+$	$q2^+$	$q1^+$	$q0^+$	J_3	K_3	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0												
0	0	0	1												
0	0	1	0												
0	0	1	1												
0	1	0													
0	1														
0	1														
0			1												
		0	0												
		0	1												
1	0	1	0												
1	0	1	1												
1	1	0	0												
1	1	0	1												
1	1	1	0												
1	1	1	1												

Alla kombinationer Av "Detta Tillstånd"

Fyll i "Nästa Tillstånd"

Arbetsgång - syntes räknare

Konstruera en räknare som räknar sekvensen ????

1 Rita en tillståndsgraf

2 Sätt upp en tabell med:

- "Detta tillstånd" (Alla kombinationer av Q_1, Q_2, Q_3)
- "Nästa tillstånd" (Q_1^+, Q_2^+, Q_3^+)
- Vippornas Insignaler

3 Ange "Nästa tillstånd" i tabellen

4 Använd vippornas excitationstabell och ange vippornas insignaler

5 Minimera uttrycken för insignalerna

6 Realisera räknaren

Lärandemål:

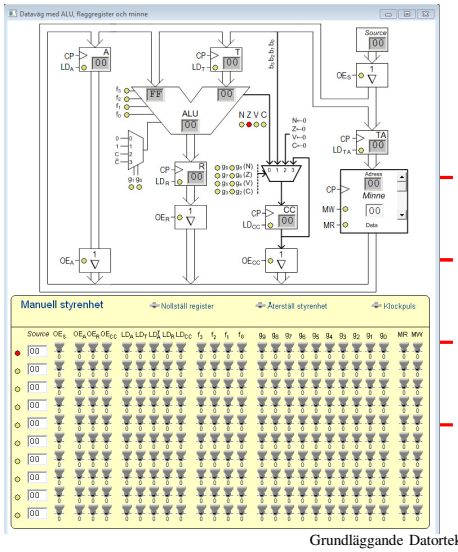
3) Datorns uppbyggnad och funktion

1. Beskriva, analysera och konstruera kombinatoriska och sekventiella nät som typiskt används för att bygga en dators centralenhet, dvs. dataväg, styrenhet, aritmetisk/logisk enhet (Arithmetic/Logical Unit).
2. Beskriva, analysera och konstruera en styrenhet baserad på fast kopplad logik och kunna implementera instruktions-exekvering i denna logik.
3. Kunna utföra elementär maskinnära programmering (maskinprogrammering och assemblerprogrammering).
4. Beskriva in-/ut- matningsenheter och minnessystem tillsammans med centralenheten.

Veckans mål:

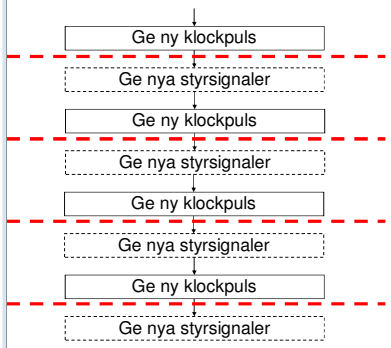
- Konstruera styrenheten…. genom att….
- …. implementera olika maskininstruktioner i styrenheten.
- Villkorliga hopp
- Subrutiner och stack
- Skriva enkla program för FLISP
- Strukturerad assemblerprogrammering

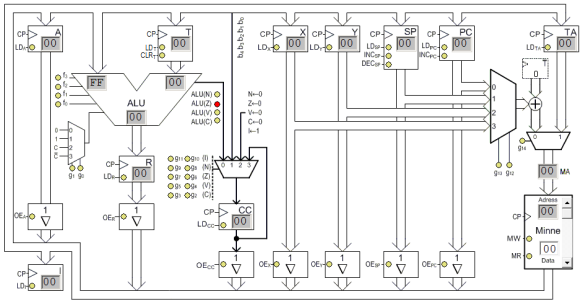
Läs smart!
Lär dig mer!



Arb s 65

Enkel dataväg



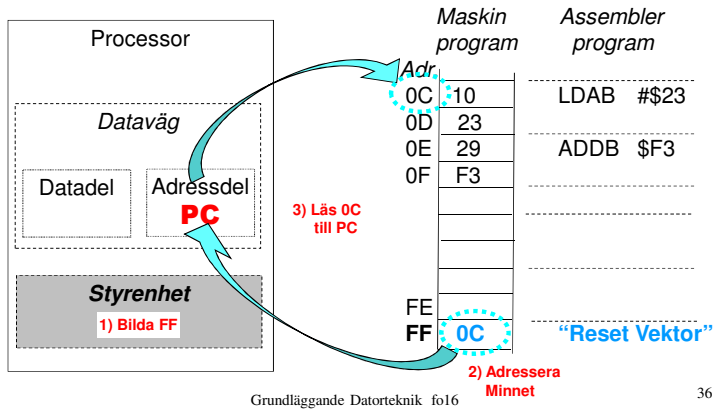


Execute

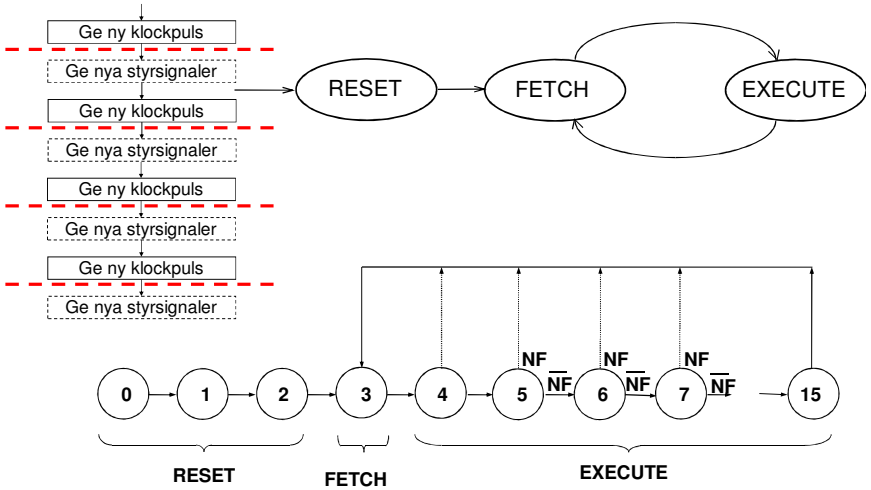
State	Summa-term	RTN	Styrsignaler = 1	Kommentar

Arb s 107

Processorns arbetssätt - RESET



Aktivera ”Processorns arbetssätt”



Aktivera ”Program och minne”

Instruktionsformat

LDA Adr

INCA

Maskinprogram	*Maskinprogram*
00001111_2	$0F_{16}$
00001011_2	$0B_{16}$
00111111_2	$3F_{16}$
11111110_2	FE_{16}
00011001_2	19_{16}
01000001_2	41_{16}
01001010_2	48_{16}

Mask. prog i minnet Adr.		*Tillhörande assemblerprog*	
0C	10	LDA	#$23
0D	23		
0E	29	ADDA	$F3
0F	F3		
10	02	TFR	X,Y
11	4F	CMPA	#$03
12	03		
13	61	BLO	$29
14	13		

Instruktionslistan för FLISP

Handassemblering

Assemblerprogram

```
LDA   #$3C
ADDA  $43
STA   4,X
JMP   $2E
```

?

Maskinprogram

Adr	
18	F0
19	3C
1A	96
1B	43
1C	E3
1D	04
1E	33
1F	2E
20	

Disassemblering

Maskinprogram

Adr	
18	F0
19	3C
1A	96
1B	43
1C	E3
1D	04
1E	33
1F	2E
20	

?

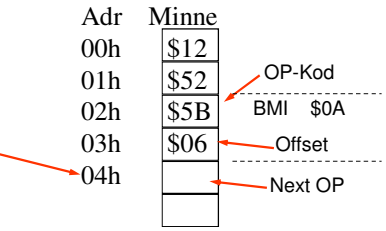
Assemblerprogram

```
LDA    #$3C
ADDA   $43
STA    4,X
JMP    $2E
```


Vad gör processorn vid BMI ?

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

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



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

Subrutin o stack

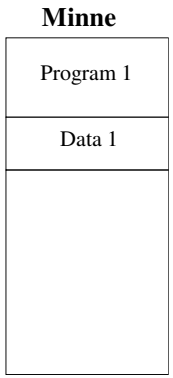
Stack: Ett minnesutrymme

Stackpekare: Register SP
Pekar på översta elementet på stacken

Subrutin: Ett stycke kod, avslutat med instruktionen RTS

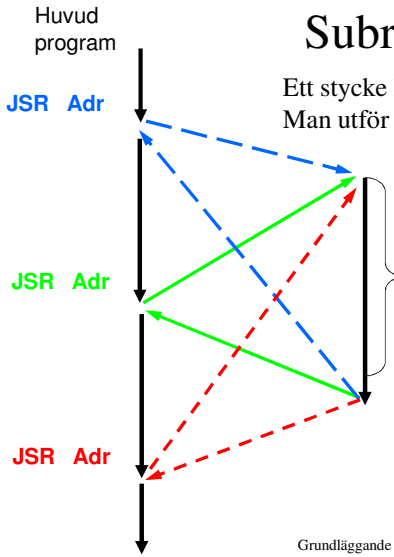
JSR: Hopp TILL subrutin

RTS: Återhopp FRÅN subrutin



Subrutiner

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



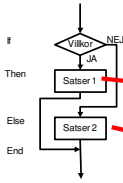
JSR Adr Jump to SubRoutine
RTS ReTurn from Subroutine

Dagens mål: Du ska kunna….

- Skriva kontrollstrukturer
  - If then else; While, Do while
  - ”testa bitar”
- Använda IO-simulatorer
- Skriva strukturerade assemblerprogram för FLISP
  - Programmoduler
  - Top-down/Bottom up
  - Inre och yttre dokumentation

Läs smart!
Lär dig mer!

Kontrollstruktur **if (...) {Satser1} else {Satser2}**




```
if (DipSwitch >= 5)
      HexDisp = 1;
else
      HexDisp = 2;
```

```
                ...
                LDAB   DipSwitch
                CMPB   #5
                BLO    else

                LDAB   #1
                STAB   HexDisp
                BRA    end      {Satser1}

else            LDAB   #2
                STAB   HexDisp  {Satser2}
end             ...
                ...
```

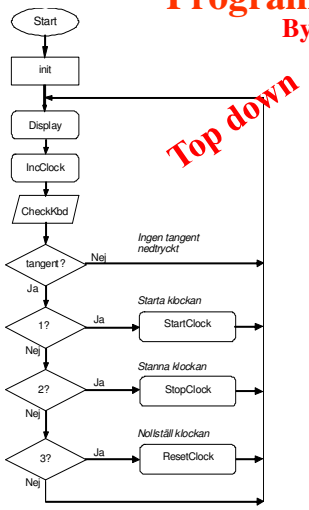
Test av tal utan tecken		
BLO	Villkor: R<M	C=1

Test av tal med tecken		
BLT	Villkor: R<M	N ⊕ V = 1

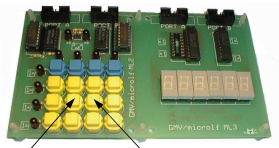
Programmeringsprojekt

s40

Bygg ett stoppur



Top down



Adress	Data	Kommentar
Clock	~~$02~~	Timmar
	~~$05~~	Tio minuter
	~~$01~~	Minuter
	~~$03~~	Tio sekunder
	~~$07~~	Sekunder
	$04	Tiondelar

Bottom up

Program1.fsr

```
          org     $10       Definerar programstart
          lda     #$12
Loop      inca
          sta     temp
          bre     Loop
temp      rmb     1         Definerar en variabel
```

KÄLLFIL

```
10          1.              org     $10
10  0F 12   3.              lda     #$12
            4.
12  41      5.  Loop        inca
13  13 17   6.              sta     temp
15  5A FB   7.              bra     Loop
            8.
18  00      9.  temp        rmb     1       Definerar en variabel
```

LISTLFIL

Adresser **Maskinkod** **Assemblerprogram**

Assemblerdirektiv

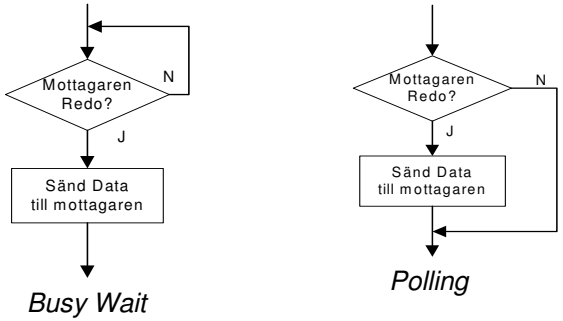
[symbol]	**FCB**	uttryck[,uttryck[,...]]	(**Form Constant Byte**)
[symbol]	**FCS**	"teckensträng"	(**Form Constant String**)
[symbol]	**RMB**	uttryck	(**Reserve Memory Bytes**)
[symbol]	**ORG**	uttryck	(**Origin**)
symbol	**EQU**	uttryck	(**Equate**)

Veckans mål:

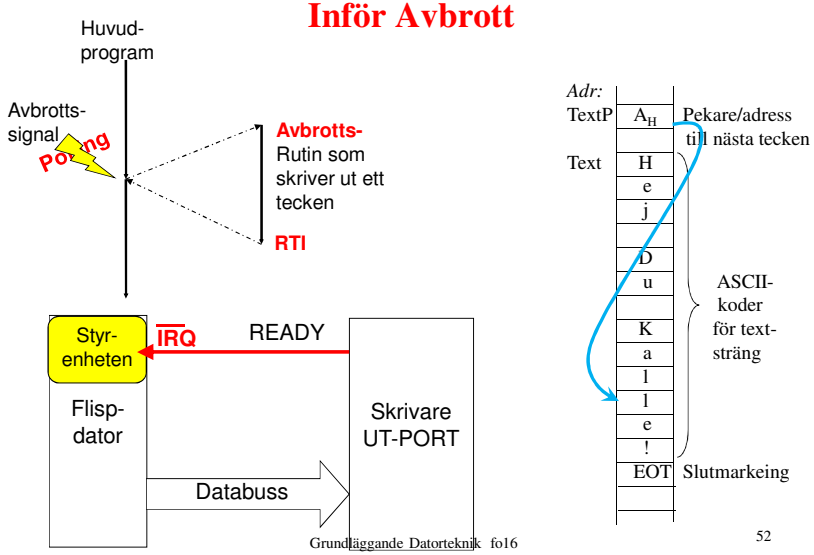
- Ansluta In- och Utportar till Flisp
- In-och utmatning
- Avbrott
- Adressavkodning
- Ansluta minnen och I/O-moduler

Läs lurt!
Lär dig mera!

Villkorlig överföring

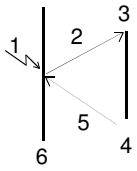


Inför Avbrott



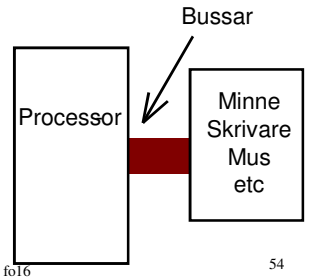
Avbrott - Sammanfattning

1. **OM I-flaggan=0**: Processorn känner att IRQ är aktiverad och slutför utförandet av pågående instruktion.
2. Processorn sparar huvudprogrammets återhoppsadress och övriga registerinnehåll på stacken, *save status*.
   Därefter läser processorn startadressen för avbrottsrutinen från IRQ-vektorn (från adress $FD).
   Denna startadress placeras i PC. I-flaggan ETT-ställs
3. Avbrottsrutinen startas (med I-flaggan=1).
4. Avbrottsrutinen avslutas med instruktionen RTI som får processorn att utföra *restore status*,
   dvs registerinnehållen återställs från stacken (med gamla I=0).
5. Återhopp till huvudprogram.
6. Därmed återstartas huvudprogrammet där det blev avbrutet

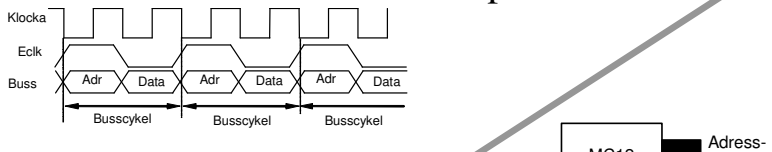


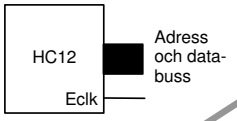
Dagens mål: Du ska kunna….

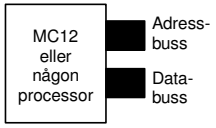
- Skilja på olika minnestyper
- Förklara principer för olika bussprotokoll
- Förstå begreppen Timing och VM (Valid Memory Adress)
- Konstruera adressavkodningslogik ….
  - 1) Fullständig Adr Avk
  - 2) Ofullständig Adr Avk
  - Processorns adressrum

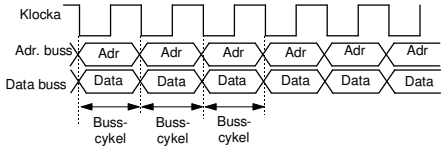


Multiplexad Buss

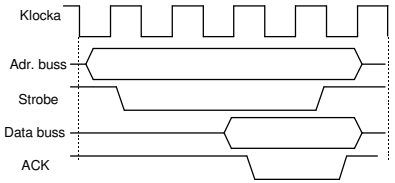




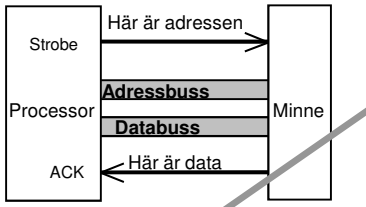




Icke Multiplexad Buss

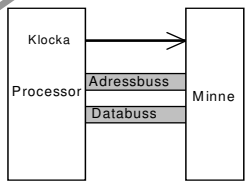


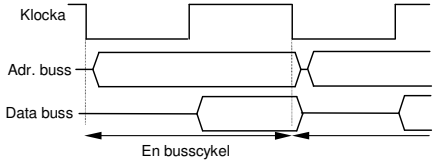
Asynkront Bussprotokoll



Ex: 68000-familjen

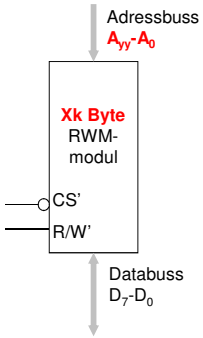
Ex: MC12





Synkront Bussprotokoll

Att ansluta en Xk Byte Minnes- modul med startadress $zzzz



Arbetsgång:

- ”Tolka” beskrivningen av minnesmodulen
- Rita tabell
- Ange modulens första adress
- Ange modulens sista adress
- Märk ut konstanta resp varierande adressledningar
- Rita adressavkodningslogiken

Att ansluta en 8-kbyte ROM- modul till ett befintligt system