Datorteknik TSEA82 + TSEA57
Fö3

Talbaser och binär aritmetik

LINKÖPINGS UNIVERSITET

Datorteknik Fö3 : Agenda

- Kort repetition
- Talbaser och Binär aritmetik
- Tid för frågor

Repetition

Instruktioner

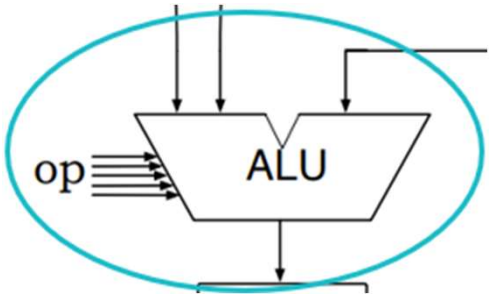
I databladet för mikrokontrollern finns drygt hundratalet instruktioner.
Dessa kan delas in i fem huvudgrupper:

- Grupp 1. Instruktioner som flyttar data (`ldi, mov, ...`)
- Grupp 2. Aritmetiska instruktioner (`add, sub, subi, ...`)
- Grupp 3. Logiska instruktioner (`asl, ror, ...`)
- Grupp 4. Hoppinstruktioner (`jmp, brxx, call, ...`)
- Grupp 5. I/O-instruktioner (`out, in`)

Instruktioner : Grupp 2. Aritmetiska operationer

Exempel: `add`, `addc`
Addera de två 16-bitarstalen som finns i `r21:r20` och `r17:r16` till `r17:r16`.

```
add   r16,r20 ; ingen ingående carry
adc   r17,r21 ; med carry (om den finns)
```




```
                11 11 1
r21:r20 00000000:11101000  =232 (0x00E8)
r17:r16 00000001:00101100  =300 (0x012C)
------------------------
r17:r16 00000010:00010100  =532 (0x0214)
```

Talbaser och binär aritmetik

Talbaser

En talbas anger antalet symboler (siffror) som ingår i ett nummersystem.
T ex den decimala basen `10` med siffrorna `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`.

I princip så existerar alla tänkbara talbaser, men i praktiken så används bara ett fåtal. Inom datorteknik så nyttjas de talbaser som har visat sig underlätta hanteringen av de tal som lagras digitalt:

- binär, basen `2` med symbolerna : `0`, `1`
- decimal, basen `10` med symbolerna : `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`
- Hexadecimal, basen `16` med symbolerna : `0` .. `9`, `A`, `B`, `C`, `D`, `E`, `F`

I litteraturen kan man även ibland se dem oktala talbasen, basen 8 med symbolerna `0` .. `7`. Den är dock inte lika vanligt förekommande längre.

Talbaser

För att veta vilken talbas som används för ett skrivet tal så anges det med en nedsänkt nummer efter talet. T ex:

$$C_{16} = 12_{10} = 1100_2$$

Det nedsänkta numret (16, 10 och 2) anges alltid i basen 10.

Utan angiven talbas vet vi ju inte vad talet 10 betyder, 16_{10}, 10_{10} eller 2_{10}?

Bas 16	Bas 10	Bas 2
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100
5	5	101
6	6	110
7	7	111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111
10	16	10000
…	…	…

Addition och ordlängd

Addition i basen 2 fungerar enligt följande:
I sista summan får vi en en-talssiffra och en två-talssiffra (carry).
Jfr med en-talssiffra och tio-talssiffra i det decimala talsystemet.

```
0 + 0 =  0
0 + 1 =  1
1 + 0 =  1
1 + 1 = 10
```

Ordlängden är det antal bitar som används i beräkningen.

Exempel:
Addera `01011` och `10111`, ordlängd `5`.

```
 11111
  01011
+10111
 100010
```

carry — Ordlängd 5

Exempel:
Addera `01011` och `10111`, ordlängd `8`.

```
   11111
 00001011
+00010111
 00100010
```

ingen carry — Ordlängd 8

Basen 2

I talbasen 2 (den binära) har vi endast två symboler, 0 och 1, för att representera olika tal.

Bitarna i ett binärt tal har positionsvikter, där en bit till vänster har dubbelt så stor vikt som biten till höger.

Bitposition:	...	7	6	5	4	3	2	1	0
Positionsvikt:	...	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Exempel: Omvandla det binära talet 10011101 till decimal form:

$$10011101_2 = 1*2^7 + 0*2^6 + 0*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 =$$
$$= 128 + 0 + 0 + 16 + 8 + 4 + 0 + 1 = 157_{10}$$

Omvandling från decimal till binär form

Exempel: Omvandla 98_{10} till ett binärt tal (m h a hjälptabell över tvåpotenser)

$$
\begin{aligned}
98 - 2^6 &= 98 - 64 = 34 && \rightarrow 1 \\
34 - 2^5 &= 34 - 32 = 2 && \rightarrow 1 \\
2 - 2^4 &= 2 - 16 = -14 && \rightarrow 0 \\
2 - 2^3 &= 2 - 8 = -6 && \rightarrow 0 \\
2 - 2^2 &= 2 - 4 = -2 && \rightarrow 0 \\
2 - 2^1 &= 2 - 2 = 0 && \rightarrow 1 \\
0 - 2^0 &= 0 - 1 = -1 && \rightarrow 0
\end{aligned}
\quad \longrightarrow \quad 1100010_2
$$

Dvs $98_{10} = 1100010_2$

Omvandling från decimal till binär form

Exempel: Omvandla 98_{10} till ett binärt tal (m h a division med 2)

```
                        rest?
98 / 2  = 49.0  nej  -> 0   ──┐
49 / 2  = 24.5  ja   -> 1     │
24 / 2  = 12.0  nej  -> 0     │
12 / 2  =  6.0  nej  -> 0     │
 6 / 2  =  3.0  nej  -> 0     │
 3 / 2  =  1.5  ja   -> 1     │
 1 / 2  =  0.5  ja   -> 1     │
```

→ 1100010_2

Dvs $98_{10} = 1100010_2$

Hexadecimal representation

Tabellen till höger är **mycket bra** att kunna utantill:

Det gör det lätt att omvandla binärt <->hexadecimalt.
Exempel: Omvandla 1110101111_2 till basen 16

$1110101111_2 = \underline{0011}\ \underline{1010}\ \underline{1111}$

$3\quad A\quad F = 3AF_{16}$

Omvandling hexadecimalt -> decimalt är relativt enkelt.
Exempel: Omvandla $3AF_{16}$ till basen 10

$3AF_{16} = 3_{10}*16^2 + 10_{10}*16^1 + 15_{10}*16^0 =$

$= 3*256 + 10*16 + 15 = 943_{10}$

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

Hexadecimal representation

Omvandling decimalt -> hexadecimalt är lite knepigare, man funkar på samma sätt som decimalt -> binärt, dvs dela med basen (16) och ta hand om resten.

Exempel: Omvandla 943_{10} till basen 16:

```
943 / 16   = 58.9375  0.9375*16=15  -> F
 58 / 16   =  3.625   0.625*16=10   -> A
  3 / 16   =  0.1875  0.1875*16=3   -> 3
```

Dvs, $943_{10} = 3AF_{16}$

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

2-komplementstal

Ett teckenlöst tal med bitarna x_3, x_2, x_1, x_0 kan skrivas:

$$X = 8*x_3 + 4*x_2 + 2*x_1 + 1*x_0$$

Ett tal med tecken (i 2-komplementsform) kan skrivas:

$$X = -8*x_3 + 4*x_2 + 2*x_1 + 1*x_0$$

Den mest signifikanta biten (teckenbiten x_3 i detta fall) i ett 2-komplemetstal är alltså negativ

Exempel: Tolka 1011_2 som ett 2-komplementstal:

$$1011_2 = -8*1 + 4*0 + 2*1 + 1*1 =$$
$$= -8 + 0 + 2 + 1 = -5_{10}$$

Dec	2-komp	Bin
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	-8	1000
9	-7	1001
10	-6	1010
11	-5	1011
12	-4	1100
13	-3	1101
14	-2	1110
15	-1	1111

2-komplementstal

Det är lätt att byta tecken på ett 2-komplementstal, invertera alla bitarna och addera 1, alltså:

$$-X = \overline{X} + 1$$

Exempel: Byt tecken på 2-komplementstalet $3 = 0011_2$

$$-3 = -0011_2 = \overline{0011} + 1 = 1100 + 1 = 1101_2$$

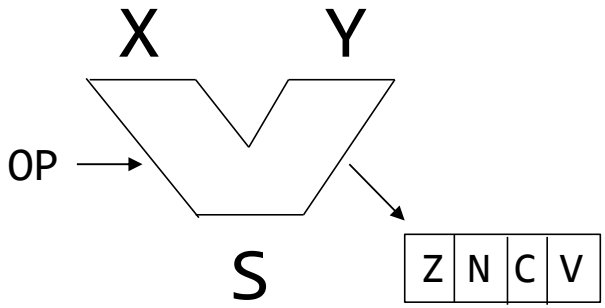
Det fungerar åt båda hållen.

$$-(-3) = -1101_2 = \overline{1101} + 1 = 0010 + 1 = 0011_2$$

Dec	2-komp	Bin
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	-8	1000
9	-7	1001
10	-6	1010
11	-5	1011
12	-4	1100
13	-3	1101
14	-2	1110
15	-1	1111

2-komplementstal : Addition, subtraktion, flaggor

En ALU har två uppgifter:



Utföra en operation

- Aritmetik
  add, sub, mul, div
- Logiska operationer
  and, or, xor, not
- Annan bitmanipulering

Sätta statusflaggor

- Z : Zero flag
- N : Negative flag
- C : Carry flag
- V : Overflow flag
- Andra flaggor

2-komplementstal : Addition, subtraktion, flaggor Z och N

Bin	2-kompl	Teckenlös	Z	N
0000	0	0	1	0
0001	1	1	0	0
0010	2	2	0	0
0011	3	3	0	0
0100	4	4	0	0
0101	5	5	0	0
0110	6	6	0	0
0111	7	7	0	0
1000	-8	8	0	1
1001	-7	9	0	1
1010	-6	10	0	1
1011	-5	11	0	1
1100	-4	12	0	1
1101	-3	13	0	1
1110	-2	14	0	1
1111	-1	15	0	1

4-bitars tal X:

$$X=\{x_3,x_2,x_1,x_0\}$$

Flaggorna Z och N:

$$Z=\overline{x}_3\cdot\overline{x}_2\cdot\overline{x}_1\cdot\overline{x}_0$$

$$N=x_3$$

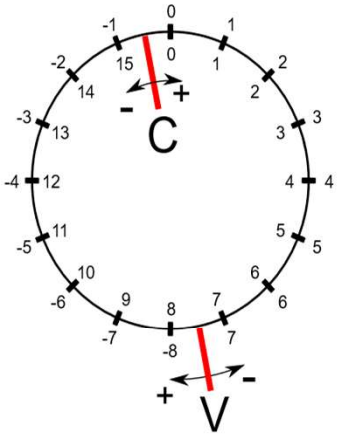
ALU:n vet inte om talet X är
med eller utan tecken.
Det bestämmer programmeraren.

Dvs, ALU:n gör alltid på samma sätt.

2-komplementstal : Addition, subtraktion, flaggor C och V

Bin	2-kompl	Teckenlös
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	-8	8
1001	-7	9
1010	-6	10
1011	-5	11
1100	-4	12
1101	-3	13
1110	-2	14
1111	-1	15

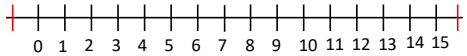
Talcirkeln: add medurs, sub moturs



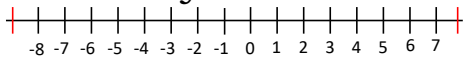
C 1-ställs när man passerar röda linjen från 15→0 (vid add), eller 0→15 (vid sub).

V 1-ställs när man passerar röda linjen från 7→-8 eller -8→7 (vid add med lika tecken eller sub med olika tecken).

C har bara betydelse för tal utan tecken.



V har bara betydelse för tal med tecken.



2-komplementstal : Addition, subtraktion, flaggor C och V

Bin	2-kompl	Teckenlös
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	-8	8
1001	-7	9
1010	-6	10
1011	-5	11
1100	-4	12
1101	-3	13
1110	-2	14
1111	-1	15

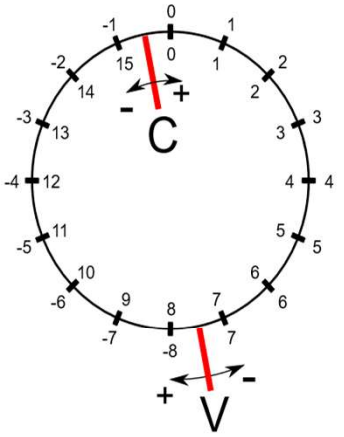
Exempel

Addition	2K	TL
0011	3	3
+1100	-4	12
01111	-1	15
Z:0,N:1,C:0,V:0		

Addition	2K	TL
0011	3	3
+0101	5	5
01000	-8	8
Z:0,N:1,C:0,V:1		

Subtr.	2K	TL
0011	3	3
-0101	5	5
11110	-2	14
Z:0,N:1,C:1,V:0		

Subtr.	2K	TL
1001	-7	9
-0100	4	4
00101	5	5
Z:0,N:0,C:0,V:1		

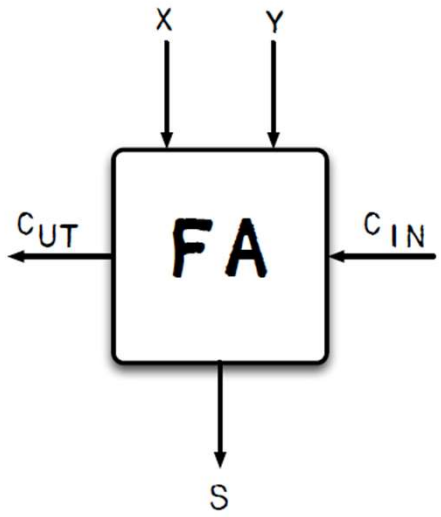


Carry vid subtraktion motsvaras av en lånesiffra och heter egentligen borrow, men fyller samma funktion som carry.

Hårdvara : Addition, flaggor C och V

Hårdvara för att addera kan byggas med hjälp av en fulladderare.
Den summerar de inkommande bitarna x och y, samt c_{in} till c_{ut} och s.
Kort sagt summeras antal inkommande 1:or.

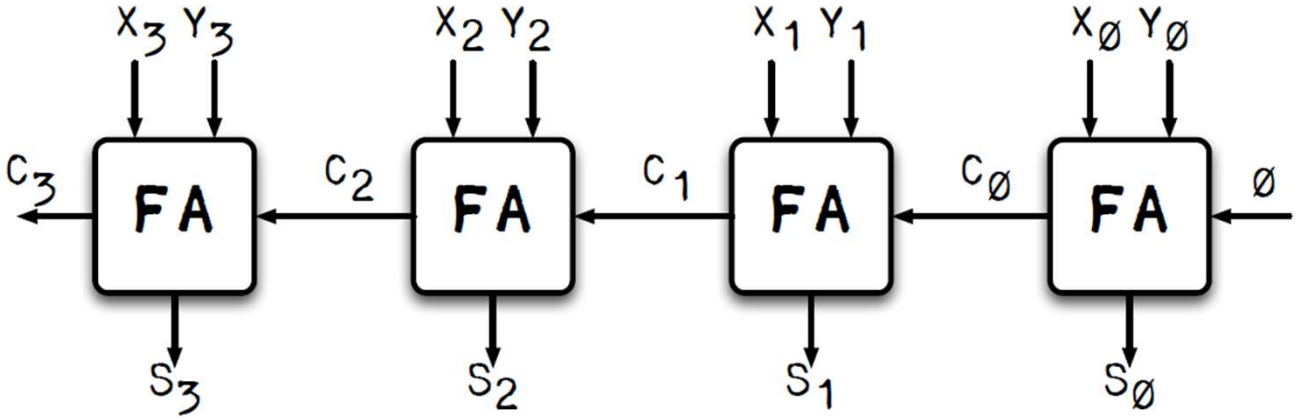
x	y	c_{in}	c_{ut}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Hårdvara : Addition, flaggor C och V

För flera bitar så kaskadkopplas flera fulladderare, t ex en 4-bitars:

$X = \{x_3, x_2, x_1, x_0\}$
$Y = \{y_3, y_2, y_1, y_0\}$
$S = \{s_3, s_2, s_1, s_0\}$



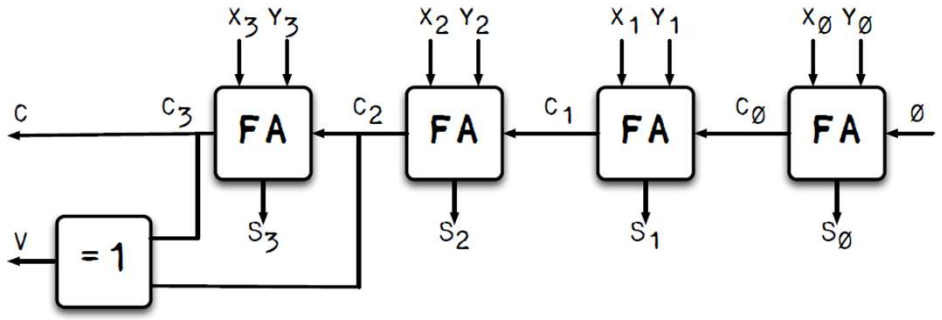
Ingående carry (c_{in} längst till höger) är `0`. I annat fall skulle vi beräkna `X+Y+1`.
Utgående carry är nu c_3 längst till vänster.
Observera, att konstruktionen fungerar både för teckenlösa tal och 2-komplementstal.

Hårdvara : Addition, flaggor C och V

Hur detekteras overflow (V), spill?
När addition av tal med lika tecken, byter tecken.

x_3	y_3	c_2	c_3	s_3	V
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0

$V = c_3 \text{ xor } c_2$

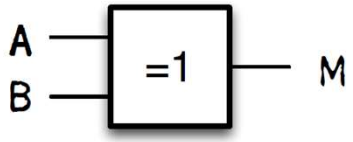


Hårdvara : Addition och subtraktion, flaggor C och V

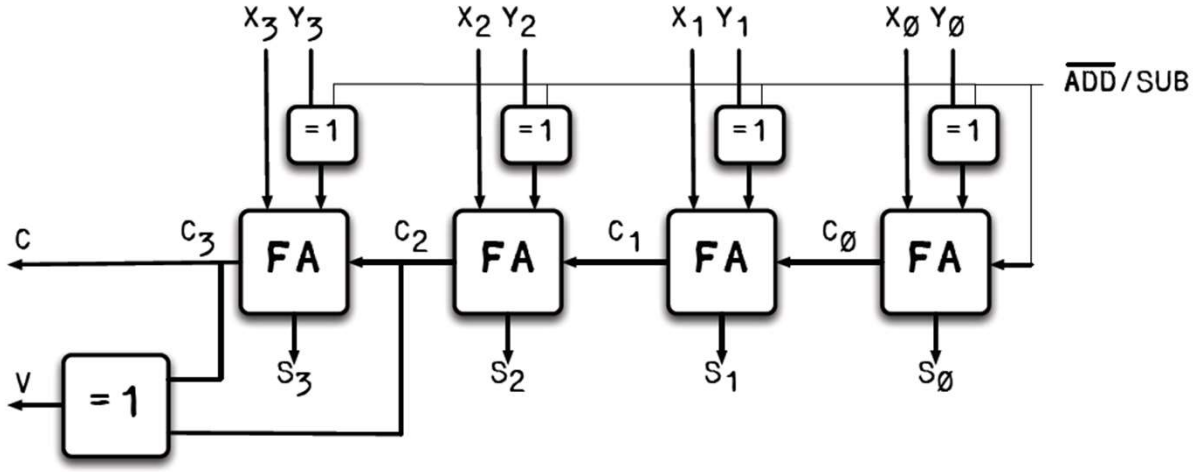
Subtraktion då?

$$X - Y = X + (-Y) = X + \overline{Y} + 1$$

Det behövs en styrbar inverterare,
dvs en xor-grind:

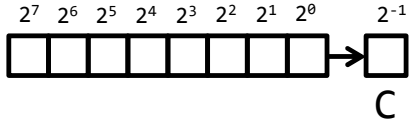


A	B	M
0	0	0
0	1	1
1	0	1
1	1	0



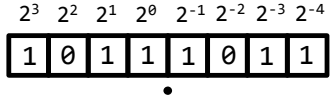
Fixtal

Hur representerar datorn tal med decimaler?
Vad händer när ett tal shiftas åt höger (delas med 2):



Minst signifikant bit åker ut i carry, och är värd ½ = 2^{-1}

Det blir en ”decimalpunkt” mellan registret och carry-biten, men i princip kan decimalpunkten placeras var som helst, bara den har en fix position i hela talsystemet. T ex:



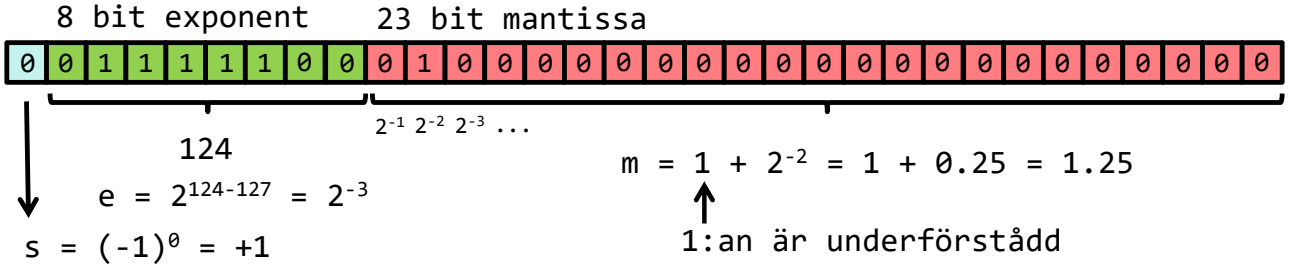
$= 2^3 + 2^1 + 2^0 + 2^{-1} + 2^{-3} + 2^{-4} = 11.6875$

Det är då ett fixtal och kan göras beräkningar med precis som tidigare.
Enbart tolkningen gör det till ett fixtal.

Flyttal

Hos ett flyttal hamnar decimalpunkten på olika platser beroende på talets storlek. Dvs, decimalpunkten *flyter omkring*.

Ett decimaltal, t ex $0.15625 = 1.5625*10^{-1}$, och 1.5625 kallas mantissa och -1 kallas exponent. Talet kan lagras med basen 2 enligt $mantissa*2^{exponent}$, men då med andra värden på mantissa och exponent (anpassade till basen 2), och enligt en viss standard. T ex för ett 32-bitars flyttal:



Värdet : $s*m*e = (+1) * 1.25 * 2^{-3} = 0.15625$

Tid för Frågor

Anders Nilsson

www.liu.se

LINKÖPINGS UNIVERSITET