Flyttalsrepresentation

Vad är en co-processor?

- En tillämpningsspecifik processor som kompletterar funktionen hos den primära processorn (CPU) med skräddarsydd hårdvara så att belastningen inte blir så stor på CPU.

  (Om CPU skulle göra samma arbete skulle det ske i mjukvara hos det körande programmet. Nu kan CPU göra annat under tiden)

Några exempel:

- Flyttalsprocessor (FPU)
- Grafikprocessor (GPU)
- Signalprocessor (DSP)

Hur används en co-processor?

- Co-processorer kan vara integrerade på samma chip som CPU
- Co-processorn utnyttjas genom särskilda assemblerinstruktioner som refererar till den och särskilda register

Hur kan reella tal representeras?

Princip: Fixerad binärpunkt, så kallade fixpunktstal.

<u>Exempel</u>

16-bits talrepresentation med teckenbit,10 bitar heltalsdel och 5 bitar bråkdel (*binaler*)

f = tdddddddddd.bbbbb

Talområde mellan $\pm 0000000000.00000_2$ och $\pm$ 1111111111.11111_2 dvs $\pm 0 - \pm 1023{,}96875_{10}$

Minsta nollskilda tal är $\pm 0000000000.00001_2 = \pm 0{,}03125_{10}$

Hur representeras reella tal i fysiken?

- 2,7182818…. (det naturliga talet *e*)
- 3,14159… (pi)
- $6{,}02 \cdot 10^{23}$ (Avogadros tal)
- $1{,}602 \cdot 10^{-19}$ (elementarladdningen)

Dessa tal kallas ofta för flyttal ,dvs decimalkommats position flyter i representationen. Motsats: fixpunktsrepresentation (används i snabba implementationer av tex bildkodningsalgoritmer).

Flyttal (decimal exempel)

- Normaliserat flyttal: $1{,}0 \cdot 10^{12}$

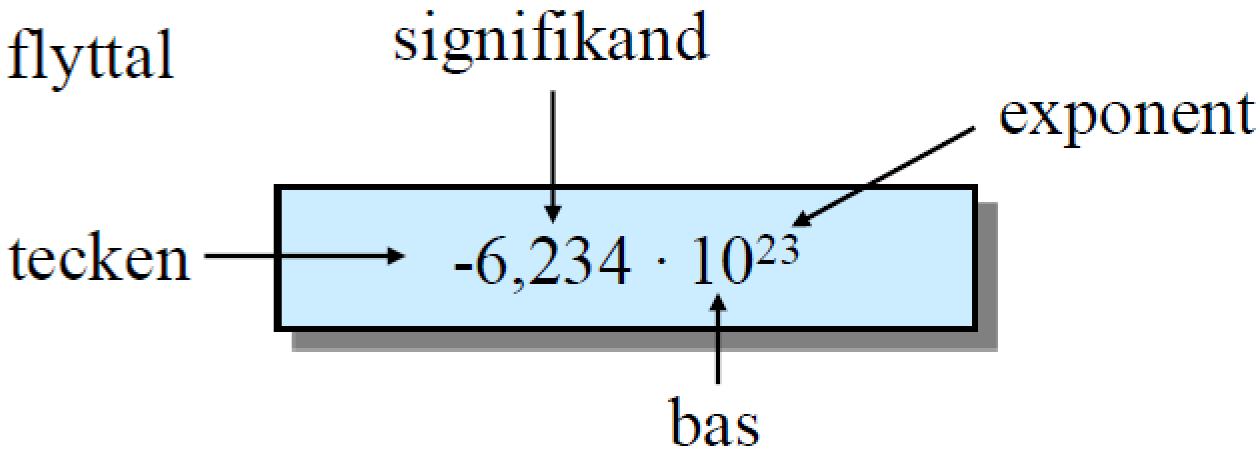
- Ej normaliserat flyttal: $0{,}026 \cdot 10^{2}$

- Normaliserat: $2{,}6 \cdot 10^{0}$

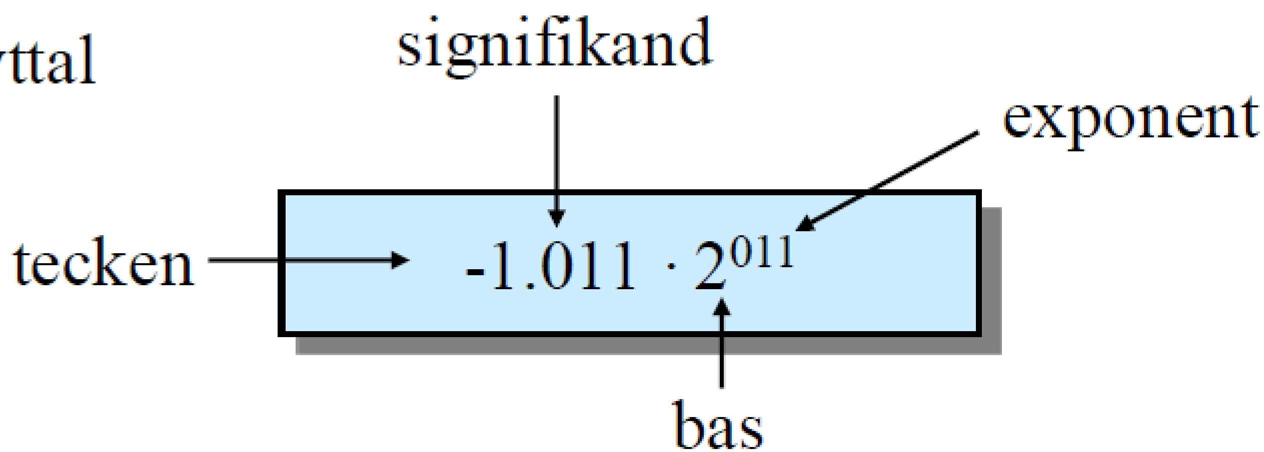
Normaliserat innebär att man har *en heltalssiffra skild från noll.*

Normaliserat flyttal

Decimalt flyttal



Binärt flyttal



En flyttalsrepresentation

Princip: Flytande binärpunkt och normaliserade tal

Exempel

16-bits tal med teckenbit 10 bits signifikand och 5 bits exponent (ger exponenter mellan -15 och +16)

Heltalssiffran är nu alltid 1 och behöver därför inte sparas.

f = teeeeessssssssss

Talområde mellan $\pm 1.0000000000_2 \cdot 2^{-15}$ och $\pm 1.1111111111_2 \cdot 2^{16}$

dvs $\pm 0{,}000030517578125_{10}$ till $\pm 131008_{10}$

IEEE 754

tecken	exponent	signifikand
1	8 bitar	23 bitar

- Basen är underförstått lika med 2
- Exponenten representeras i excess-127
- Signifikanden är ett fixtal x, $0 \leq x < 1$

- Talets värde = $(-1)^{\text{tecken}} \cdot (1.0 + \text{signifikand}) \cdot 2^{(\text{exponent} - 127)}$
  - en implicit etta* i signifikanden ökar noggrannheten
  - en exponent i excessformat tillåter jämförelse mellan två flyttal med heltalsinstruktioner

***kallas hidden bit*

Exponent på excessformat (*biased exponent*)

Vanlig binärkod

Excess-127-format

Bitmönster tolkat i excess-127

00000000_2 = 0 - 127 = -127

011111111_2 = 127 – 127 = 0

111111111_2 = 255 – 127 = 128

100000010_2 = 130 – 127 = 3

Flyttalsexempel 1

Representera –0,75 i enkel precision enligt IEEE 754 i enkel precision (32 bitar)

”Fixpunktsbeskrivning”

$–0,75 = -0.11 = (–1)^1 \times 1.1_2 \times 2^{–1}$

teckenbit = 1

signifikand = $1000…00_2$

exponent = –1 + excess

$–1 + 127 = 126 = 01111110_2$

1011111101000000000000000000000000

Flyttalsexempel 2

Vilket tal representerar flyttalsrepresentationen i 32 bitar

11000000101000000000000000000000

- **tecken = 1**
- **signifikand = 01000…00$_2$**
- **exponent = 10000001$_2$ = 129**

talet = $(-1)^1 \times (1 + 0.01_2) \times 2^{(129-127)}$

$= (-1) \cdot 1{,}25 \times 2^2$

$= -5{,}0$

Jämför med talet -5 i heltalsrepresentation i 32 bitar:

$11111111111111111111111111111011_2$

Addition/subtraktion av flyttal

Algoritm

1. Bestäm skillnaden mellan de båda talens exponenter.

2. Skifta minsta talets signifikand (inkl. implicit etta) till höger så båda talens exponent blir lika stora (dvs lika med den största).

3. Addera/subtrahera signifikanderna.

4. Normalisera summan (skifta, ändra exponent)

5. Avrunda signifikanden om det behövs

6. Om avrundningen innebär att det blir onormaliserat så gå tillbaka till steg 3.

Exempel på flyttalsaddition

Addera talen $2,52 \cdot 10^2 + 2,52 \cdot 10^3$ enligt IEEE 754 single format (SFP)

Omvandla talen till rätt binärformat:

<u>TalA</u>

$\mathbf{2,52 \cdot 10^2} = 252 = 128 + 64 + 32 + 16 + 8 + 4 = 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 =$

$= 11111100_2 = \mathbf{0\ 10000110\ 11111000000000000000000_{SFP}}$

(Flytta binärpunkten 7 steg ger exponentsiffra = 127 + 7 = 134 = 10000110)

<u>TalB</u>

$\mathbf{2,52 \cdot 10^3} = 2520 = 2048+256+128+64+16+8 = 2^{11} + 2^8 + 2^7 + 2^6 + 2^4 + 2^3 =$

$= 100111011000_2 = \mathbf{0\ 10001010\ 00111011000000000000000_{SFP}}$

(Flytta binärpunkten 11 steg ger exponentsiffra = 127 + 11 = 138 = 10001010)

Additionsexempel forts

Dela upp talen i tecken exponent och mantissa:

$A = (-1)^{S_A} \cdot M_A \cdot 2^{(E_A-127)}$ och $B = (-1)^{S_B} \cdot M_B \cdot 2^{(E_B-127)}$

$A = 0\ 10000110\ 11111000000000000000000_{SFP}$

$B = 0\ 10001010\ 00111011000000000000000_{SFP}$

$S_A = 0$	$S_B = 0$
$M_A = 1.11111000000000000000000$	$M_B = 1.00111011000000000000000$
$E_A = 10000110$	$E_B = 10001010$

Additionsexempel forts.

1. Bestäm skillnaden mellan talens exponenter
   $10001010_2 – 10000110_2$ = 138 -134 = 4

2. Skifta mantissan hos talet med minst exponent (M_A) fyra steg åt höger:
M_A’ = 0.000111111000000000000000000

3. Utför addition (båda talen är ju positiva)

```
    111111
 1.00111011000000000000000
+0.000111111000000000000000
 1.010110101000000000000000
```

4. Normalisera resultatet, i detta fall är resultatet redan i normaliserad form

Additionsexempel forts.

Resultat:

$S_R = 0$

$M_R = 1.01011010100000000000000$

$E_R = E_B = 10001010$

Sammanställning av resultat:

$R = 0\ 10001010\ 01011010100000000000000_{SFP}$

Multiplikation av flyttal

<u>Algoritm</u>

1. Addera exponenterna och dra ifrån en excess
2. Multiplicera signifikanderna.
3. Normalisera produkten
4. Om det blir exponent overflow är det aritmetiskt fel
5. Avrunda signifikanden
6. Om avrundningen innebär att det blir onormaliserat så gå tillbaka till steg 3.
7. Beräkna produktens tecken

Flyttalsstandarder

<u>Olika format</u>

– 32 bitar (float)
  - ➢ 1 teckenbit, 8 bitar exponent , 23 bitar signifikand

– 64 bitar (double)
  - ➢ 1 teckenbit, 11 bitar exponent , 52 bitar signifikand

– 128 bitar (long double)
  - ➢ 1 teckenbit, 15 bitar exponent, 112 bitar signifikand

Några speciella tal

- **Hur kan talet 0 representeras när vi har en underförstådd etta?**
  - Om både signifikand och exponent är 0 tolkas talet som 0
- **Andra speciella tal**
  - Om alla exponentbitar är ettor och signifikand $\neq 0$ så är talet NaN (Not A Number)
  - Om alla exponentbitar är ettor och signifikand $= 0$ är talet ∞ (oändligheten)

Resultat vid beräkningar med specialtal

- $\infty + x = \infty$
- $\infty \cdot (-x) = -\infty$
- $\infty - \infty = \text{NaN}$
- $\infty \cdot 0 = \text{NaN}$

Enkelt flyttalsformat i 8 bitar

S EEE MMMM   enligt samma principer som IEEE754

- Mantissan representeras med hidden bit och exponenten med excess-7-format

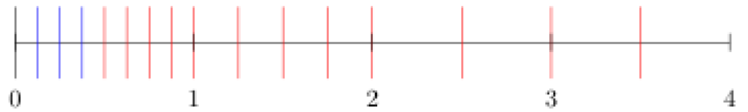
- Minsta positiva beskrivningsbara normerade talet är 0 000 0001
  Vilket motsvarar
  $(–1)^0 \times 1.\ 0001_2 \times 2^{-7} = 1{,}0625 \times 2^{-7} = 0{,}00830078125$

- Det näst minsta talet är 0 000 0010 som motsvarar
  $(–1)^0 \times 1.0010_2 \times 2^{-7} = 1{,}125 \times 2^{-7} = 0{,}0087890625$

- Skillnaden mellan dessa tal är 0,00048828125
  vilket är mycket mindre än det minsta talet.

- Det betyder att vi fått ett ”hål” i talfördelningen runt nollan.

Denormaliserade tal



- Normaliserade
- Denormaliserade

- Normalisering ger ett ”hål” kring nollan
  - Minsta normaliserade tal i 32 bitar är $1{,}17549435 \cdot 10^{-38}$

- ”Hålet” fylls ut med denormaliserade tal, dvs när alla exponentbitar är noll blir exponenten -126 och heltalsbiten hos signifikanden blir istället 0.

  - Största denormaliserade tal blir då $1.17549421 \cdot 10^{-38}$

- Denormaliseringen ger linjärt utspridda tal nära nollan.

- Normaliserade tal blir glesare ju större de blir

Avrundning vid flyttalsberäkningar

Hur?

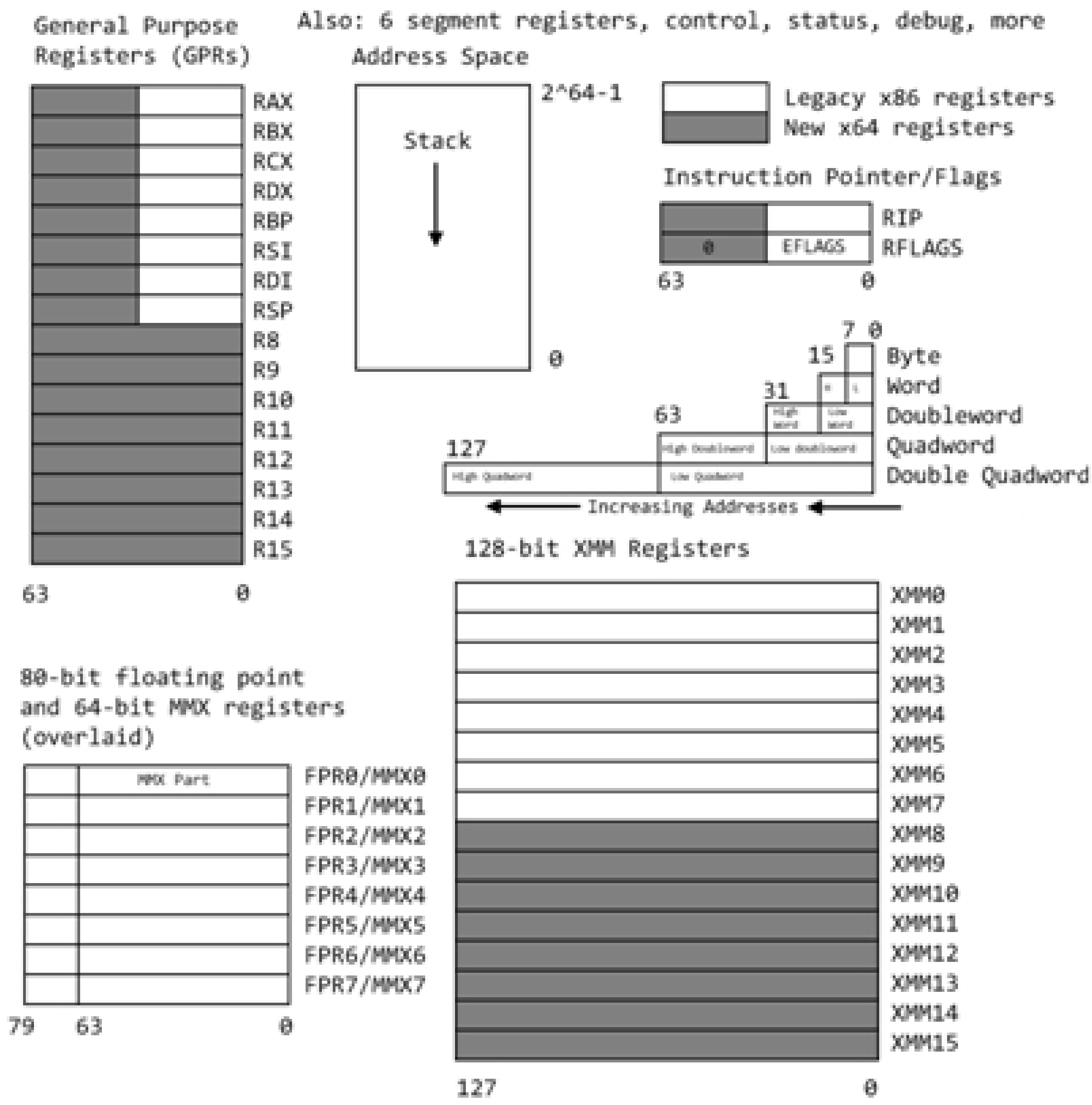
- Det behövs extra hårdvara för att kunna avrunda korrekt
- Brukar finnas två extrabitar i talet som kallas ”guard” och ”round” och en extra bit som kallas ”sticky bit”
- ”Sticky bit” sätts till ett om det finns fler nollskilda bitar i talet som inte får plats (förutom round och guard)

Avrundning, forts

Olika avrundningsstrategier

1. Alltid uppåt
2. Alltid nedåt
3. Trunkering
4. Till närmaste jämna tal

Mer fullständig registeröversikt



Historik - Intel

- Flyttalsoperationer gjordes av ett separat chip ursprungligen – 8087
- Hade en stack med 8 st 80 flyttalsvärden

- X64 har 16 flyttalsregister
  (128 eller 256 bitar)

- Kan användas för flyttal eller
  SIMD-instruktioner

- Registren heter **`XMM`** för 128 bitar och
  `YMM` för 256 bitar (överlappar varandra)

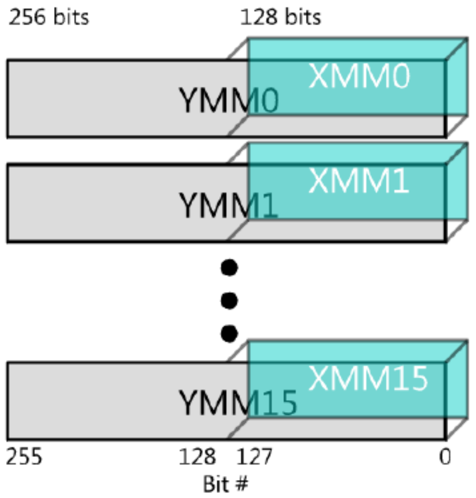


Figure 1. *XMM registers overlay the YMM registers.*

Några flyttalsinstruktioner

instruction	effect
addsd	add scalar double
addss	add scalar float
addpd	add packed double
addps	add packed float
subsd	subtract scalar double
subss	subtract scalar float
subpd	subtract packed double
subps	subtract packed float
mulsd	multiply scalar double
mulss	multiply scalar float
mulpd	multiply packed double
mulps	multiply packed float
divsd	divide scalar double
divss	divide scalar float
divpd	divide packed double
divps	divide packed float

float – 32 bitar (single)
Double – 64 bitar (double)

- cvtss2sd converts a scalar single (float) to a scalar double
- cvtps2pd converts 2 packed floats to 2 packed doubles
- cvtsd2ss converts a scalar double to a scalar float
- cvtpd2ps converts 2 packed doubles to 2 packed floats

```
cvtss2sd    xmm0, [a]   ; get a into xmm0 as a double
addsd       xmm0, [b]   ; add a double to a
cvtsd2ss    xmm0, xmm0 ; convert to float
movss       [c], xmm0
```

Några ”komplexa” instruktioner

Finns instruktioner för:

- Minimum (**`min`**)
- Maximum (**`max`**)
- Avrundning (**`round`**)
- Kvadratrot (**`sqrt`**)
- Reciprok (**`rcp`**)
- Reciprok av kvadratrot (**`rsqrt`**)

Exempel –
Funktion för avståndsformeln

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Avståndet mellan två punkter (x_1, y_2) och (x_2, y_2) i ett koordinatsystem.

```
distance:
    movss   (%rdi), %xmm0       # x from first point
    subss   (%rsi), %xmm0       # subtract x from second point
    mulss   %xmm0, %xmm0        # (x1-x2)^2
    movss   4(%rdi), %xmm1      # y from first point
    subss   4(%rsi), %xmm1      # subtract y from second point
    mulss   %xmm1, %xmm1        # (y1-y2)^2
    addss   %xmm1, %xmm0        # add x and y parts
    sqrtss  %xmm0, %xmm0        # square root calculation
    ret                         # re5turn from fucntion
```