4 ZAPIS INFORMACIJE IN ARITMETIKA

BRANKO ŠTER

PO KNJIGI - DUŠAN KODEK: ARHITEKTURA IN ORGANIZACIJA RAČUNALNIŠKIH SISTEMOV

Informacija

- Informacija v računalniku
  - Ukazi
  - Operandi
    - Numerični
      - Fiksna vejica
        - Predznačena
        - Nepredznačena
      - Plavajoča vejica
        - Enojna natančnost
        - Dvojna natančnost
    - Nenumerični
      - Logične spremenljivke
      - Znaki

Zapis nenumeričnih operandov

- Pri prvih rač. so bili operandi samo numerični
  - danes je veliko nenumeričnih
- Običajno so nenumerični operandi znaki oz. nizi znakov (strings)
- Vsak znak (character) je predstavljen z neko abecedo

Abeceda BCDIC

- BCDIC (Binary Coded Decimal Interchange Code)
- do leta 1964
- 6-bitna
- 10 števil, 26 črk, 28 posebnih znakov
- hitro je postala premajhna

000000 ... 0
000001 ... 1
000010 ... 2
...
001001 ... 9

010001 ... A
010010 ... B
010011 ... C
...

	000	001	010	011	100	101	110	111
000	0	1	2	3	4	5	6	7
001	8	9		#	@			
010	&	A	B	C	D	E	F	G
011	H	I	+0	.	¤			
100	-	J	K	L	M	N	O	P
101	Q	R	-0	$	*			
110	space	/	S	T	U	V	W	X
111	Y	Z	‡	,	%			
	0	1	2	3	4	5	6	7

Abeceda EBCDIC

- Extended Binary Coded Decimal Interchange Code
- IBM, 1964
- 8-bitna
- razširitev abecede BCD

Abeceda ASCII

- ASCII - American Standard Code for Information Interchange
- 1968
- originalno 7-bitna (128 znakov), razširjena 8-bitna
- od tega 95 natisljivih znakov in 33 kontrolnih znakov
  - A ... 1000001 (65), B ... 1000010 (66), ...
  - a ... 1100001 (97), b ... 1100010 (98), ...
  - 0 ... 0110000 (48), 1 ... 0110001 (49), ...
  - ! ... 0100001 (33), " ... 0100010 (34), ...
- kontrolni znaki za rač. komunikacije in krmiljenje V/I naprav

Koda BCD

- Spodnji 4 biti znakov za desetiške cifre v abecedah BCDIC, EBCDIC in ASCII ustrezajo njihovi dvojiški numerični vrednosti
  - to je koda **BCD** (**Binary Coded Decimal**), 4-bitna binarna predstavitev desetiških cifer

Unicode

- Unicode
  - neprofitni konzorcij, 1991
  - abecede UTF-8, UTF-16, UTF-32
  - UTF-8
    - posamezen znak zavzame od 1 do 4 bajtov
    - prvih 128 znakov isto kot ASCII (kompatibilnost)

Število bajtov	Št. bitov kode	Prva koda	Zadnja koda	Bajt 1	Bajt 2	Bajt 3	Bajt 4
1	7	00	7F	0xxxxxxx			
2	11	0080	07FF	110xxxxx	10xxxxxx		
3	16	0800	FFFF	1110xxxx	10xxxxxx	10xxxxxx	
4	21	10000	10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Zapis numeričnih operandov v fiksni vejici

- Števila
- Pozicijska notacija
  - vsaka pozicija ima svojo težo
  - $192{,}73 = 1\times10^2 + 9\times10^1 + 2\times10^0 + 7\times10^{-1} + 3\times10^{-2}$

Pozicijska notacija

- Ta zapis lahko posplošimo na uteži oblike r^i, kjer je r **baza** ali **radix** številskega sistema

$$V = \sum_{i=-m}^{n-1} b_i r^i$$

  - $215{,}36_7 = 2\times7^2 + 1\times7^1 + 5\times7^0 + 3\times7^{-1} + 6\times7^{-2}$

- V računalnikih se uporablja baza $r = 2$
  - nekdaj se je tudi baza $r = 10$
    - BCD-kodiranje

Dvojiški zapis števil

- Dvojiški (binarni) zapis: baza r = 2
  - $b_{n-1} \ldots b_2 b_1 b_0 , b_{-1} b_{-2} \ldots b_{-m}$ b_i = 0 ali 1

  Vrednost: $$V(b) = \sum_{i=-m}^{n-1} b_i 2^i$$

- Primer: pretvori $110101{,}101_2$ v desetiško število.

  $110101{,}101_2$ =

  $1*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 1*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3} = 53{,}625_{10}$

Pretvorba desetiških števil v bazo *r*

- Algoritem:
  1. $N : r = Q_1 + b_0$
  2. Ponavljaj 1. za $Q_i : r = Q_{i+1} + b_i$ za i = 1, 2, 3, ...
  3. Končaj, ko $Q_i = 0$

- Primer: pretvorba 98_{10} v bazo r=3
  - $98_{10} = 10122_3$
- Posebno nas zanima pretvorba v bazo r=2 (pretvorba desetiškega števila v dvojiško)
  - $27_{10} = 11011_2$

Pretvorba ulomkov v bazo *r*

- Algoritem:
  1. $N * r = b_{-1} + F_1$
  2. Ponavljaj 1. za $F_i * r = b_{-(i+1)} + F_{i+1}$ za $i = 1, 2, ...$
  3. Končaj, ko $F_i = 0$

- Primer: pretvorba $0,375_{10}$ v bazo $r = 2$
  - $0,011_2$

Napaka pri rezanju decimalk

- Kadar število *N* odrežemo na *k* decimalk, dobimo približek *N'*
  - napaka $N' - N$, absolutna napaka $|N' - N|$
  - Abs. napaka ne more preseči r^{-k}
  - Zadostiti moramo pogoju:

    $r^{-k} \leq E_{max}$
  - Poiščemo tak *k*, da neenačba velja (običajno lahko tudi brez kalkulatorja)

    $k \geq \log_r (1/E_{max})$

    $\mathbf{k = \lceil \log_r(1/E_{max}) \rceil}$

- Če logaritma z bazo *r* ne znamo izračunati, ga pretvorimo v bazo *e* ali 10:

  $\log_a c = \log_a b * \log_b c$ (pravilo)

  (na ta način se znebimo baze *b*, v našem primeru *r*, za a pa vzamemo kako znano bazo)

  $\log_e c = \log_e r * \log_r c$

  $\log_r c = \ln c / \ln r$

  $$k = \lceil \ln(1/E_{max}) / \ln r \rceil$$

- Primer: pretvorba $N = 0{,}8_{10}$ v bazo $r = 3$. Vzemi toliko decimalk, da napaka ne preseže $E_{max} = 0{,}01$.

  $0{,}8_{10} = 0{,}\ 2101\ 2101\ \ldots_3$

  Če upoštevamo *k* decimalk, napaka ne preseže r^{-k}

  $r^{-k} <= E_{max}$

  Brez kalkulatorja lahko ocenimo primeren *k*:

  $3^{-5} = 1/243 = 0{,}004\ldots$, $3^{-4} = 1/81 = 0{,}012\ldots$

  S kalkulatorjem:

  $k = \lceil \ln(100) / \ln(3) \rceil = \lceil 4{,}19 \rceil = 5$

  $0{,}8_{10} = 0{,}21012_3$

- Pri r = 2 imamo kar dvojiški logaritem (lb)

  $k = \lceil \log_2(1/Emax) \rceil$

- Primer: $0{,}8_{10}$ v bazo 2, E_{max} = 0,01

  $0{,}8 = 0,\ 1100\ 1100\ \ldots_2$

  k = 7: $0{,}8 = 0{,}1100110_2$ (N' = 0,796875, E = -0,003125)

- Primer: $N = 159{,}3_{10}$ v bazo r = 16. $|N' - N| \leq 10^{-3}$

  9(15),4(12)(12)(12)...16

  $16^{-3} < 10^{-3}$

  k = 3

  159,310 = 9(15),4(12)(12)16

Pretvorba med poljubnima bazama

- Pretvorba r' v r:
  - r' v 10
  - 10 v r

- Npr. $26{,}5_8$ v r=3
  - $211{,}12\ 12\ \ldots_3$

Osmiška in šestnajstiška baza

- Poleg dvojiške se v računalništvu pogosto uporabljata tudi **osmiška** (oktalna) in še posebno **šestnajstiška** (heksadecimalna) baza
  - v 16-iški bazi so poleg 0 .. 9 še dodatne cifre:
    - A (10), B (11), C (12), D (13), E (14), F (15)
  - Primer:
    - $3C7_{16} = 3*16^2 + 12*16^1 + 7*16^0 = 768 + 192 + 7 = 967_{10}$
  - Različni načini zapisa:
    - $3C7_{16} = 3C7_H$ = 0x3C7 = \$3C7

Sorodne baze

- Ker sta ti bazi sorodni bazi 2, je pretvorba enostavna
  - Pri osmiški bazi ena cifra predstavlja 3 bite (dvojiške baze)
    - $1110010101_2 = 1\ 110\ 010\ 101_2 = 1625_8$,
    - $327_8 = 011\ 010\ 111_2$
  - Pri šestnajstiški bazi ena cifra predstavlja 4 bite (dvojiške baze)
    - $1110010101_2 = 11\ 1001\ 0101_2 = 395_{16}$ oz. 0x395
    - $A15_{16} = 1010\ 0001\ 0101_2$

Nepredznačena števila

- Z n biti lahko zapišemo nepredznačena števila od 0 do 2^n-1 (z n biti lahko v kateremkoli formatu zapišemo 2^n števil!)
  - npr. n = 3, števila od 0 (000) do 7 (111)
  - npr. n = 10, števila od 0 (000...) do 1023 (111...)
- Kadar rezultat neke operacije preseže obseg števil, se pojavi **prenos** (**carry**)
  - rezultat na podanem številu cifer ni pravilen

101 + 100 = (1)001

Zapisi predznačenih števil

- Predznačeno število lahko zapišemo na več načinov
- V vseh primerih imamo n-bitno število: $b_{n-1} \ldots b_2b_1b_0$, njegova vrednost pa se v različnih načinih zapisa razlikuje
- Primer: Zapisi 3-bitnih predznačenih števil

b_2	b_1	b_0	PV	PO	1'K	2'K
0	0	0	+0	-4	+0	0
0	0	1	1	-3	1	1
0	1	0	2	-2	2	2
0	1	1	3	-1	3	3
1	0	0	-0	0	-3	-4
1	0	1	-1	1	-2	-3
1	1	0	-2	2	-1	-2
1	1	1	-3	3	-0	-1

Predznak-veličinski zapis

1. **Predznak-veličinski zapis**

$$V(b) = (-1)^{b_{n-1}} \sum_{i=0}^{n-2} b_i 2^i$$

- prvi bit (b_{n-1}) predstavlja predznak, ostali velikost
- Hibe:
  - predznak je treba obravnavati posebej
  - ima dve ničli: -0 in +0
- PV zapis ni primeren za seštevanje/odštevanje
- Primeren za množenje/deljenje (ki pa sta manj pogosti operaciji)

Zapis z odmikom

2. **Zapis z odmikom**

$$V(b) = \sum_{i=0}^{n-1} b_i 2^i - 2^{n-1}$$

- odmik je (običajno) 2^{n-1}
- nekoč priljubljen zapis
- Hibe:
  - pri seštevanju je treba odmik odšteti
  - pri odštevanju je treba odmik prišteti
    - v oboje se lahko hitro prepričamo

Eniški komplement

3. Eniški komplement (1'K)

$$V(b)=\sum_{i=0}^{n-1} b_i 2^i - b_{n-1}(2^n - 1)$$

- b_{n-1} je predznak
- pozitivna števila (b_{n-1}=0) enako kot pri PV
- negativno število dobimo iz pozitivnega z invertiranjem vseh bitov
  - ekvivalentno odštevanju od $2^n - 1$ (same enice)
- predznaka ni treba obravnavati posebej! ☺
- hibe: ☹
  - 2 ničli (-0, +0)
  - pri prenosu z najvišjega mesta je treba na najnižjem mestu prišteti 1 (End Around Carry - EAC)

Dvojiški komplement

4. Dvojiški komplement (2'K)

$$V(b)=\sum_{i=0}^{n-1} b_i 2^i - b_{n-1} 2^n$$

- Tudi tu se pozitivna števila začnejo z 0:
  - 0000 (0), 0001 (1), ..., 0110 (6), 0111 (7)=max
- Negativna števila se začnejo z 1:
  - 1000 (-8), 1001 (-7), ..., 1110 (-2), 1111 (-1)
  - ni pa takoj razvidno, za katero število gre ☹ (torej V)
    - Od nepredznačene vrednosti (vsota v gornji formuli) moramo odšteti 2^n
      - Npr. b=1001: V = 9 – 16 = –7
      - Npr. b=1110: V = 14 – 16 = –2

- Negativno število (zapis b pri podani vrednosti V) dobimo tako, da vrednosti V prištejemo 2^n
  - Npr.: -2 + 16 = 14, torej tak zapis kot za nepredznačeno 14
- Lahko pa tudi tako, da invertiramo vse bite pozitivnega števila (eniški komplement) in prištejemo 1 (to je ekvivalentno odštevanju od 2^n)
  - npr.

```
   0010 (2)
   1101 (-2 v 1'K)
+     1
   1110 (-2 v 2'K)
```

- Velja pa tudi obratno: če želimo ugotoviti, za katero negativno število gre, spet naredimo 2'K (1'K in prištevanje enice)
  - 10110 = ?, 1'K: 01001 + 1 = 01010, kar je 10 (deset), torej je 10110 enako -10 (minus deset)
- Potrebno pa je razlikovati med pojmoma *zapis v 2'K* in *2'K nekega števila*

- Bit prenosa pri 2'K ignoriramo!

```
    011
   +110
   ----
(1)001
```

$a-b = a+(-b) = a+(2^n-b) = a-b$ **+ 2^n** (to je bit prenosa)

```
    011  (3)
   +110  (-2)      110=1000-010
   ----
(1)001
```

- 2'K je najpogosteje uporabljan zapis
  - primeren za seštevanje/odštevanje
    - nima EAC
  - le ena predstavitev za ničlo
  - predznaka ni treba obravnavati posebej

- Pri razširitvi števila na več bitov je potrebno **razširiti predznak**:
  - **0**101 v **000**101
  - **1**100 v **111**100
  - 01011111 v 0000000001011111
  - 11001100 v 1111111111001100

Primer

- Zapiši -37 kot predznačeno 10-bitno število v PV, PO, 1'K in 2'K
  - PV: 1000100101
  - PO: 0111011011
  - 1'K: 1111011010
  - 2'K: 1111011011

Preliv

- Obseg števil v *n*-bitnem 2'K:
  $-2^{n-1} \leq x \leq 2^{n-1} - 1$

- Če je (pravi) rezultat operacije izven tega območja: **preliv** (**overflow**)
  - rezultat je napačen
  - preliv se da detektirati

- Preliv ni isto kot **prenos** (**carry**) z najvišjega mesta!
  - le-ta se nanaša na operacije z *nepredznačenimi* števili
    - območje $0 \leq x \leq 2^n - 1$
  - pri 2'K se prenos ignorira

- Kdaj pride do preliva?
  - potreben pogoj je, da imata števili enak predznak
  - zadosten pogoj pa je, da ima vsota drugačen predznak kot števili

- Pogoj za preliv (OF) lahko zapišemo kot

$$OF = x_{n-1}\, y_{n-1}\, \overline{s_{n-1}} \vee \overline{x_{n-1}}\ \overline{y_{n-1}}\, s_{n-1}$$

  - ker pa je pri prvem produktu c_{n-1}=0 in c_n=0, pri drugem pa obratno, ga lahko zapišemo tudi kot

$$OF = c_{n-1} \oplus c_n$$

- Primeri operacij v 4-bitnem 2'K:

```
  0100  (4)     0101  (5)     1100  (-4)     1010  (-6)
+ 0011  (3)   + 0100  (4)   + 0101  (5)    + 1011  (-5)
  0111  (7)     1000  (-8)  1 0001  (1)    1 0101  (5)
```

- Seštej 21 in -7 v 6-bitnem 2'K:

```
     010101
   + 111001
  (1)001110
```


Primeri aritmetičnih operacij v različnih bazah

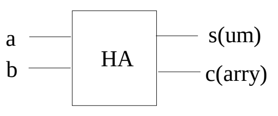
- $02345_9 + 16250_9 = 18605_9$
- $21202_3 + 12012_3 = (1)10221_3$, pojavi se prenos
- $11001_2 + 01011_2 = (1)00100_2$, pojavi se prenos

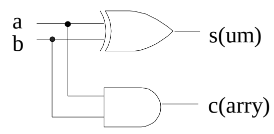
- $4102_5 – 2430_5 = 1122_5$
- $3306_7 – 0615_7 = 2361_7$
- $10110_2 – 01101_2 = 01001_2$

- $324_5 * 023_5 = 014112_5$
- $1101_2 * 0101_2 = 01000001_2$

ARITMETIČNA VEZJA
Polovični seštevalnik

- **Polovični seštevalnik (Half Adder, HA)**
  - sešteva 2 bita, izračuna vsoto (*s*, sum) in (izhodni) prenos (*c*, carry)



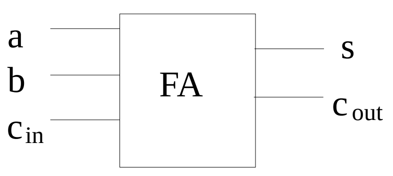


a	*b*	*c*	*s*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$s = a\,b' \vee a'b = a \oplus b$$
$$c = a\,b \;\; (= a \,\&\, b)$$

Polni seštevalnik

- **Polni seštevalnik** (Full Adder, FA)
  - sešteva 3 bite, izračuna vsoto in (izhodni) prenos



$$s = a \oplus b \oplus c_{in} \;\; (= a'b'c_{in} \vee a'b\,c_{in}' \vee a\,b'c_{in}' \vee a\,b\,c_{in})$$
$$c_{out} = a\,b \vee a\,c_{in} \vee b\,c_{in}$$

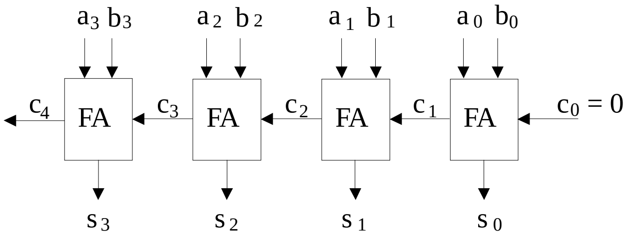
Večbitni seštevalnik

- **Večbitni seštevalnik**
  - **Seštevalnik z razširjanjem prenosa** (Ripple Carry Adder, RCA)
    - zaporedna vezava 1-bitnih FA
    - izhodni prenos nižjega vezan na enega od vhodov višjega
      - običajno se en vhod imenuje kar vhodni prenos (c_{in})

$$s = a \oplus b \oplus c_{in}$$
$$c_{out} = a\,b \vee a\,c_{in} \vee b\,c_{in}$$

    - hiba: zakasnitev
      - Dejanska zakasnitev je odvisna od operandov
        - Npr.: pri seštevanju dveh ničel ne bo izhodnega prenosa (ne glede na vhodni prenos), pri seštevanju dveh enic pa bo vedno izhodni prenos (ne glede na vhodni prenos), zato v takih primerih ni treba čakati na vhodni prenos
      - Maksimalna zakasnitev pa narašča praktično linearno
        - V najslabšem primeru se prenos razširja čez vse FA
        - Npr.: če sta pri vseh FA na vhodu različna bita, je treba čakati ne vhodni prenos, ki določa, ali se bo pojavil tudi izhodni prenos

Večbitni seštevalnik



- **Seštevalnik z vnaprejšnjim prenosom** (Carry-Lookahead Adder, CLA)
  - hiter izračun vseh prenosov
    - le na osnovi vhodov a, b in c_0
  - dodatna logika
    - sprememba večnivojske oblike v dvonivojsko

Seštevalnik / odštevalnik

- Seštevanje in odštevanje predznačenih števil v 2'K z enim vezjem
  - signal M (Add'/Sub) določa operacijo
    - 0: +
    - 1: –
  - odštevanje kot prištevanje 2'K
    - a – b = a + b' + 1
      - –b kot dvojiški komplement b
      - $b' = (b_{n-1}' \ldots b_1' b_0') \ldots 1'K$
      - $b_i \oplus M$
      - XOR dela kot krmiljen negator ($x \oplus 0 = x$, $x \oplus 1 = x'$)
      - +1: M vežemo na c_0

Binarno množenje

- Binarno množenje
  - tvorba delnih (parcialnih) produktov (n*n konjunkcij)
  - seštevanje delnih produktov

$$
\begin{array}{ccccc}
 & & x_2 \quad x_1 \quad x_0 & \times & y_2 \quad y_1 \quad y_0 \\
\hline
x_2y_2 & x_1y_2 & x_0y_2 & & \\
 & x_2y_1 & x_1y_1 & x_0y_1 & \\
 & & x_2y_0 & x_1y_0 & x_0y_0 \\
\hline
\end{array}
$$

  - Delni produkt je enak množencu, če je ustrezni bit množitelja enak 1, sicer je enak 0

Načini množenja

- 2 vrsti metod:
  - pomikanje in seštevanje
    - 1 bit / cikel ure
    - poceni, a ne prav hitro
    - registri
  - kombinacijski množilniki
    - brez ure
    - dragi, a hitri

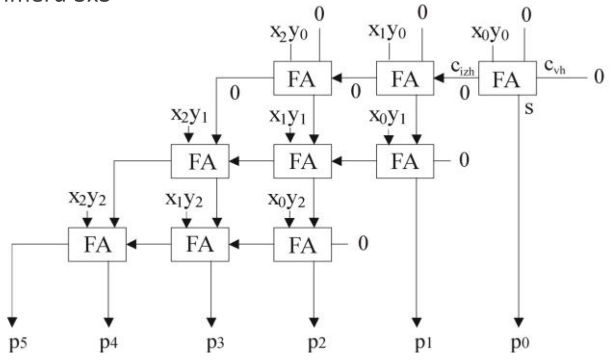
- Množenje s pomiki in seštevanjem (shift-and-add multiplication)
  - Postopek iz n korakov:
    - Če je najnižji bit množitelja B enak 1, prištej množenec A registru P (na začetku 0)
      - sicer prištej 0
    - Pomik desno registrov P in B (kaskadno vezanih)

- Primer: A=5, B=6

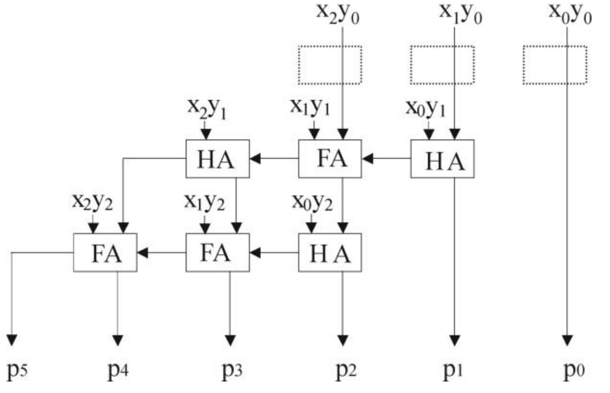
	P	B	
0	0000	0110	začetek
1	0000	0110	P ← P + 0
	0000	0011	P,B >> 1
2	0101	0011	P ← P + A
	0010	1001	P,B >> 1
3	0111	1001	P ← P + A
	0011	1100	P,B >> 1
4	0011	1100	P ← P + 0
	0001	1110	P,B >> 1

Matrični množilnik

- na primeru 3x3



Nekateri FA so odveč



- Zakasnitev ~ linearna
  - (3*n*-2)*ΔFA
  - (3*n*-4)*ΔFA

- Obstajajo tudi metode za hitro seštevanje več sumandov, t.i. paralelni števniki (parallel counters)
  - Wallace, Dadda, ...
  - glavna aplikacija je množenje

- Množenje v 2'K
  - Booth-ov algoritem

- Binarno deljenje
  - 2 osnovna načina:
    - zaporedje odštevanj in pomikov
    - matrični delilnik
      - enobitni odštevalniki

Problemi pri vključitvi aritmetike v računalniški sistem

- Preliv
  - 2 rešitvi:
    - postavitev posebnega bita
    - sprožitev pasti (nek bit lahko določa, ali se sproži, ali pa se ignorira)
- Dolžina produkta
  - produkt dveh števil je shranjen v spremenljivki enake velikosti kot števili
- Izvajanje operacij v eni urini periodi
  - množenje in deljenje sta zahtevnejši operaciji
  - 2 rešitvi:
    - ukazi korak-množenja
    - množenje izvaja posebna enota
      - lahko FPU (floating point unit)
      - CPU čaka na izračun

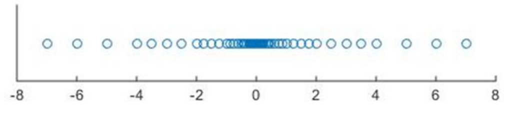
Zapis števil v plavajoči vejici

- Obseg števil v fiksni vejici je za določene probleme premajhen
  - potrebovali bi tudi zelo velika ali zelo majhna števila
- Znanstvena notacija omogoča krajši zapis
  - npr. 1×10^{18} namesto 1 000 000 000 000 000 000
- Število lahko zapišemo kot $m \times r^e$
  - *m* je **mantisa**, *r* je **baza** (običajno 2), *e* je **eksponent**
  - s spreminjanjem eksponenta vejica plava vzdolž mantise levo in desno (odtod ime plavajoča vejica)

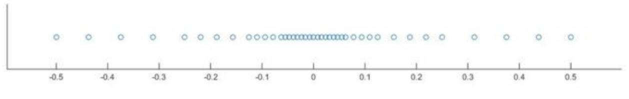
- V plavajoči vejici lahko zapišemo bistveno večja, pa tudi bistveno manjša števila kot v fiksni
  - kljub temu pa je možnih števil enako mnogo (2^n)

- Primer: plavajoča vejica v mini (7-bitnem) formatu
  - predznak: 1bit, mantisa: 3 biti, eksponent: 3 biti
  - $(-1)^S*m*2^{E-7}$,
    - max: $111*2^0 = 7$
    - min abs.: $0*2^{-3} = 0$
      - $1*2^{-7} = 0{,}0078$, $2*2^{-7} = 0{,}016$, ...
    - min: $-111*2^0 = -7$

celoten obseg števil:



del obsega:



- Vsako število lahko v plavajoči vejici zapišemo na več načinov:
  - npr. $1\times10^{18} = 10\times10^{17} = 0{,}1\times10^{19}$ ...
  - npr. $1\times2^{3} = 10\times2^{2} = 0{,}1\times2^{4}$ ...
  - zato mantiso normiramo:
    - prvi bit je 1 (normalni bit), implicitno predstavljen
    - npr.: mantisa 01001... pomeni 1,01001...
  - zelo majhnih števil pa ni mogoče predstaviti v normirani obliki
    - denormirana števila
    - **podliv** (underflow)
- Eksponent je predstavljen v **predstavitvi z odmikom**

- Nekdaj je vsak proizvajalec je uporabljal svoj format zapisa v plavajoči vejici
  - isti program je lahko na različnih računalnikih dajal različne rezultate

- Standard IEEE 754 (1985)
  - IEEE: Institute of Electrical and Electronics Engineers
  - 2 formata:
    - enojna natančnost (single precision), 32 bitov
    - dvojna natančnost (double precision), 64 bitov

Enojna natančnost

- Enojna natančnost (single precision), 32 bitov

31	30 23	22 0
S	E	m

- predznak S (0: +, 1: –)
- 8-biten eksponent e z odmikom 127 ($e = E - 127$)
- 23-bitna mantisa m (7-mestna desetiška natančnost)
- normirana vrednost je $\mathbf{(-1)^{S} \cdot 1,}\boldsymbol{m}\ \mathbf{\cdot 2^{E-127}}$, E = 1, 2, ..., 254
- obseg: $\pm\,1{,}18 * 10^{-38}$, $\pm\,3{,}40 * 10^{38}$ (v norm. obliki)

Dvojna natančnost

- Dvojna natančnost (double precision), 64 bitov

63	62 52	51 0
S	E	m

- predznak S (0: +, 1: –)
- 11-biten eksponent e z odmikom 1023 ($e = E - 1023$)
- 52-bitna mantisa m (16-mestna desetiška natančnost)
- normirana vrednost je $\mathbf{(-1)^{S} \cdot 1,}\boldsymbol{m} \cdot \mathbf{2^{E-1023}}$, E = 1, 2, ..., 2046
- obseg: $\pm 2{,}22 * 10^{-308}$, $\pm\,1{,}80 * 10^{308}$ (v norm. obliki)

Primer: število 2

- $2 = +1.0*2^1$
- S = 0, m = 0, e = 1
- enojna: E = e + 127 = 128 = 10000000

31	30 23	22 0
0	10000000	00000000000000000000000

- dvojna: E = e + 1023 = 1024 = 10000000000

63	62 52	51 0
0	10000000000	0000000000000000000000000 00000

Primer: število -8.25

- $-8.25 = -1000.01 = -1.00001*2^3$
- S = 1, m = 0000100 ..., e = 3
- enojna: e = 3, E = e + 127 = 130 = 10000010

31	30 23	22 0
1	10000010	00001000000000000000000

- dvojna: e = 3, E = e + 1023 = 1026 = 10000000010

63	62 52	51 0
1	10000000010	0000100000000000000000000 00000

Denormirana števila

- **Denormirana števila** (zelo majhna števila)
  - E=0
  - implicitni normalni bit je enak 0
  - vrednost v 32-bitnem formatu je $(-1)^S \cdot 0{,}m \cdot 2^{-126}$
    - eksponent je -126 namesto -127, ker imamo (0,m) namesto (1,m)
  - vrednost v 64-bitnem formatu je $(-1)^S \cdot 0{,}m \cdot 2^{-1022}$,
    - eksponent je -1022 namesto -1023, ker imamo (0,m) namesto (1,m)
  - tudi 0 je denormirano število, ki ima mantiso enako 0

Neskončnosti in NaN

- Še dve posebni vrsti števil:
  - **Neskončnosti**
    - E = 255 (v 32-bitnem formatu) oz. E = 2047 (v 64-bitnem formatu), vsi biti E so 1
    - če m=0, imamo $+\infty$ in $-\infty$
    - pojavijo se, kadar je rezultat prevelik (npr. 1/0 da $+\infty$)
  - **NaN**
    - ravno tako E = 255 oz. 2047
    - $m \neq 0$
    - pojavijo se kot rezultat nedefiniranih operacij
      - npr. $0 \times \infty$, 0/0, $\infty - \infty$, kvadratni koren negativnega števila, ...
    - rezultat operacije, ki vsebuje operand NaN, je tudi NaN

Aritmetika v plavajoči vejici

- Aritmetika v plavajoči vejici se obravnava in realizira ločeno od aritmetike v fiksni vejici
  - bolj zapletena
- Zaokroževanje
  - zaokrožujemo od matematično natančne vrednosti k najbližjemu še predstavljivemu številu
  - kadar je vrednost enako oddaljena od dveh najbližjih števil, se zaokroži k sodemu številu
    - standard IEEE 754 sicer dovoljuje tudi drugačne načine zaokroževanja, vendar so redkeje uporabljani
  - pri računanju mantiso podaljšamo za 3 dodatne bite
    - varovalni bit (guard bit)
    - zaokroževalni bit (round bit)
    - lepljivi bit (sticky bit)

Dodatni biti

- **Varovalni bit** je potreben, ker je vsota lahko za eno mesto daljša od operandov
- **Zaokroževalni bit** omogoča bolj natančno zaokroževanje
- **Lepljivi bit** se uporablja zato, da se iz izpadlih bitov vidi, ali je bil kak različen od 0 (zaradi zaokroževanja k sodemu številu)
    - v tem primeru je treba zaokrožiti navzgor (ne navzdol zaradi morebitnega najbližjega sodega števila)
    - izračuna se kot funkcija ALI izpadlih bitov

Seštevanje v plavajoči vejici

- Seštevanje (in odštevanje) v plavajoči vejici
  - Prvo število naj bo tisto z večjim eksponentom (začasni eksponent)
  - Pomik mantise drugega števila (če izpadejo kake enice, se shranijo v lepljivem bitu)
  - seštevanje (odštevanje) mantis
  - Če se pojavi prenos naprej, zmanjšaj mantiso (pomik za eno mesto) in povečaj začasni eksponent za 1
  - Zaokrožitev mantise
    - če grs=100 (točno polovica zadnjega mesta), zaokrožimo k sodemu številu (če je zadnji bit mantise 0, ga pustimo; če je 1, zaokrožimo navzgor)

- **Primer 1**. Seštej binarno 3,25 + 30, če je mantisa 3-bitna, imamo pa dodatne bite g, r in s.
  - 11,01*2^0 + 11110,0*2^0 = 1,101|000*2^1 + 1,111|000*2^4 = 1,111|000*2^4 + 0,001|101*2^4 = 10,000101*2^4 = 1,000|01<u>0(1)</u>*2^5 = 1,000|01**1**(grs)*2^5 = 1,000*2^5 = 32

- Primer 2. Odštej binarno 30 - 4,125, če je mantisa 3-bitna, imamo pa dodatne bite g, r in s.

```
30₁₀ = 11110,0*2^0 = 1,11100*2^4
4,125₁₀ = 100,001*2^0 = 1,00001*2^2
```

(to število ima manjši eksponent (2^2), zato ga povečamo na 2^4, zaradi česar se pomakne mantisa za 2 mesti)

```
 1,00001*2^2 = 0,010|0001*2^4 = 0,010|001*2^4
                     grs,  s=0v1=1    grs
 1,111|000*2^4
- 0,010|001*2^4
 1,100|111*2^4 = 1,101*2^4 = 26₁₀
       grs
```

Pravilen rezultat bi bil 25,875 (napaka 0,125 nastane zaradi pomikanja mantise manjšega števila v desno)

Množenje v plavajoči vejici

- Množenje v plavajoči vejici
  - eksponenta seštejemo (dobimo začasni eksponent)
  - mantisi zmnožimo z množilnikom (v fiksni vejici)
    - množilnik v bistvu sploh ne ve, da je nekje vmes vejica …
  - po potrebi normiramo rezultat
  - predznak produkta je XOR obeh predznakov
- Deljenje v plavajoči vejici
  - odštevanje eksponentov, deljenje mantis

- Primer 1: $A \cdot B$, $A = 1{,}01 \cdot 2^2$, $B = 1{,}11 \cdot 2^0$
  - začasni eksponent = 2 + 0 = 2 (ker je $2^2 * 2^0 = 2^2$)
  - množimo mantisi (PAZI: Poleg mantis števili sestavljata tudi implicitni enici!)

```
  1,01 · 1,11
  101
   101
    101
10,0011
```

Kako vemo, kje je vejica?

- Produkt je 6-biten (3+3), za vejico pa morajo biti 4 mesta (4 = 2+2)
  - Vsak od obeh faktorjev ima 2 mesti desno od vejice

$10{,}0011 \cdot 2^2$ normiramo: $1{,}00011 \cdot 2^3$

- predznak: $0 \oplus 0 = 0$, tj. +

$$A \cdot B = +1{,}00011 \cdot 2^3$$

- Pretvorite A, B in produkt v desetiško obliko in preverite pravilnost rezultata

- Primer2: Zmnoži C = A · B v enojni natančnosti (A = 0x326C8000, B = 0xBF200000).
  Zapiši produkt C tudi v 16-iški obliki.

```
0x326C8000 = 0011 0010 0110 1100 1000 0000 0000 0000
              E = 01100100 = 2^6+2^5+2^2 = 100
                      e = E - 127 = -27 (dejanski eksponent)
A = +1,11011001*2^-27

0xBF200000 = 1011 1111 0010 0000 0000 0000 0000 0000
               E = 01111110 = 128-2 = 126
                      e = E - 127 = -1 (dejanski eksponent)
B = -1,01*2^-1
```

Zmnožimo mantisi (skupaj z normalnima enicama!):

```
  1,11011001 * 1,01   (A: 9 mest, 8 za vejico, B: 3 mesta, 2 za vejico)
  --------------
   111011001
    000000000
     111011001
  --------------
  10,0100111101   (9+3=12 mest skupno, za vejico jih mora biti 8+2=10)
```

Predznak: 0 xor 1 = 1, torej minus

```
C = -10,0100111101*2^-28      (potrebno še normirati)
C = -1,00100111101 *2^-27       (PAZI: Povečanje eksponenta za 1:  -28 + 1 = -27)

  E = e + 127 = 100
C = 1 01100100 00100111101000000000000   (dodamo toliko ničel, da je mantisa 23-bitna)
```

Združujemo v skupine po 4:

```
C = 1 011|0010|0 001|0011|1101|0000|0000|0000

C = B213D000_16 (oz. 0xB213D000)
```