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 ASCII

- ASCII - American Standard Code for Information Interchange (1968)
- 7-bitna (128 znakov)
- 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

- Razširjena ‘ASCII’
  - 8-bitna
    - dodatnih 128 znakov
  - ISO/IEC 8859
    - ISO 8859-1 (Latin-1), zahodnoevropske črke
    - ISO 8859-2 (Latin-2), vzhodnoevropske črke

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 (Unicode transformation format)
  - UTF-8
    - posamezen znak zavzame od 1 do 4 bajtov
      - kodiranje spremenljive dolžine – variable length encoding
    - 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
  - $b_{n-1} \ldots b_2 b_1 b_0 , b_{-1} b_{-2} \ldots b_{-m}$

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

  - $215{,}36_7 = 2\times7^2 + 1\times7^1 + 5\times7^0 + 3\times7^{-1} + 6\times7^{-2}$
- V računalnikih se uporablja baza $r = 2$
  - nekdaj se je tudi baza $r = 10$
    - BCD-kodiranje

Dvojiški zapis števil v fiksni vejici

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

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

  - Zapis v fiksni vejici ima bite razdeljene na celi del in ulomek (Qn.m)
    - Npr. Q8.8, Q10.6, Q4.4, Q3.5, Q2.2
    - Q3.5: $b_2 b_1 b_0 , b_{-1} b_{-2} \ldots b_{-5}$ $\quad (n = 3, m = 5)$
  - Primer: pretvori $110101{,}101_2$ v desetiško število.
    - $110101{,}101_2 = 2^5 + 2^4 + 2^2 + 2^0 + 2^{-1} + 2^{-2} + 2^{-3} = 53{,}625_{10}$

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*=8
  - $98_{10} = 142_8$
- 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 $i = 1, 2, \ldots$
  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 m mest desno od vejice, dobimo približek N'
  - napaka $N' - N$, absolutna vrednost napake $|N' - N|$
  - $|N' - N| <= r^{-m}$
- Če želimo, da napaka po abs. vrednosti ne preseže E_{max}:

  $r^{-m} \leq E_{max}$
  - Potrebujemo m mest

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

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

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

  $$k = \lceil \log_2(1/E\text{max}) \rceil$$

- 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)

Pretvorba med poljubnima bazama

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

- Npr. $26{,}5_8$ v *r*=3
  - 211,12 12 ... $_3$

- Kadar sta bazi sorodni (oba večkratnika istega števila), je pretvorba lažja
  - npr. pretvorbe med bazami nabora 2, 4, 8, 16

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

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

- 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 števk (cifer) ni pravilen

101 + 100 = (1)001

Primeri aritmetičnih operacij z nepredznačenimi števili v različnih bazah

- $0234_8 + 1525_8 = 1761_8$
- $2103_4 + 2313_4 = (1)1022_4$, pojavi se prenos
- $11001_2 + 01011_2 = (1)00100_2$, pojavi se prenos

- $3306_8 – 0615_8 = 2471_8$
- $A089_{16} – 5CED_{16} = 439C_{16}$
- $10110_2 – 01101_2 = 01001_2$

- $325_8 * 026_8 = 12016_8$
- $1101_2 * 0101_2 = 01000001_2$

Zapisi predznačenih števil

- Predznačeno število lahko zapišemo na več načinov
- V vseh primerih imamo *n*-bitno število: $b_{n-1} \ldots b_2 b_1 b_0$, 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

1 Predznak-veličinski zapis (PV)

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

- 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)

2 Zapis (predstavitev) z odmikom (PO)

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

- odmik je (običajno) 2^{n-1}
- nekoč priljubljen zapis
- Hibe:
  - pri seštevanju je treba odmik odšteti
  - pri odštevanju je treba odmik prišteti

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)

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)

- 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)
```

- Tudi obratno, če želimo ugotoviti, za katero negativno število gre:
  - nepredznačenemu zapisu odštejemo 2^n
    - Npr., 1101: 13 – 16 = -3
  - spet naredimo 2'K (1'K in prištevanje enice):
    - 1101: 1'K: 0010, +1 = 0011 (=3)

- Potrebno je razlikovati med pojmoma
  - *zapis v 2'K* in
  - *2'K nekega števila !*

- Bit prenosa pri 2‘K ignoriramo!

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

$a-b = a+(-b) = a+(2^n-b) = a-b + \mathbf{2^n}$ (to je bit prenosa)

```
  011 (3)
 +110 (-2)
 ----
(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 → **0000 0**101
  - **1**100 → **1111 1**100
  - **0**10111 → **000**10111
  - **1**00011 → **111**00011

Primer

- Zapiši -28 kot predznačeno 6-bitno število v PV, PO, 1’K in 2’K
  - `PV:  111100`
  - `PO:  0`
  - `1’K: 1`
  - `2’K: 1`

Osnovna aritmetika v 2'K

- 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 *nepredznačenim*i števili
    - območje $0 \leq x \leq 2^n - 1$
  - pri 2'K se prenos ignorira

Preliv

- Kdaj pride do preliva (Overflow)?
  - 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)
```

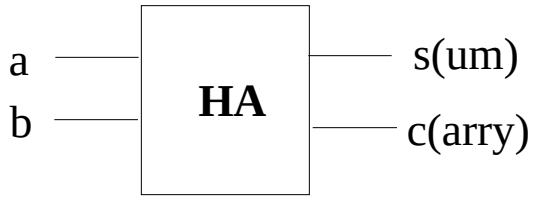
- Seštej 21 in -7 v 6-bitnem 2'K:

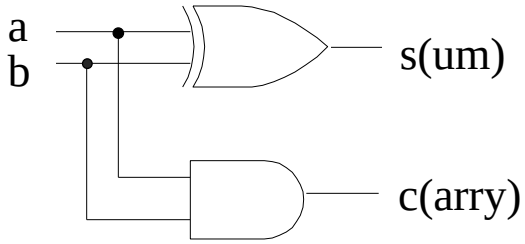
```
   010101
+  111001
(1)001110
```

ARITMETIČNA VEZJA

Polovični seštevalnik

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



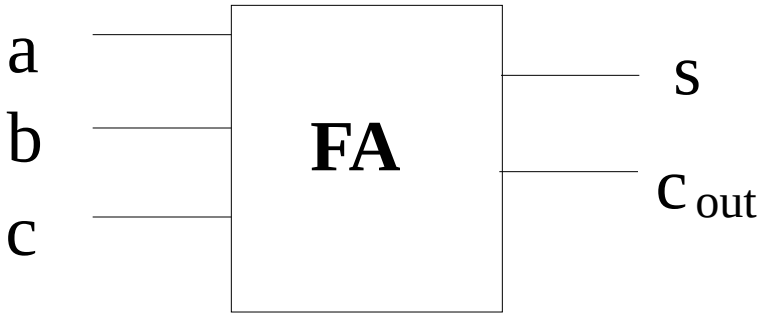


a	*b*	*c*	*s*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$s = a\,\bar{b} \vee \bar{a}\,b = a \nabla b \,(= a \oplus b)$$
$$c = a\,b \;\; (= a \,\&\, b)$$

Polni seštevalnik

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



$$s = a \nabla b \nabla c \; (= \bar{a}\,\bar{b}\,c \vee \bar{a}\,b\,\bar{c} \vee a\,\bar{b}\,\bar{c} \vee a\,b\,c)$$
$$c_{out} = a\,b \vee a\,c \vee b\,c$$

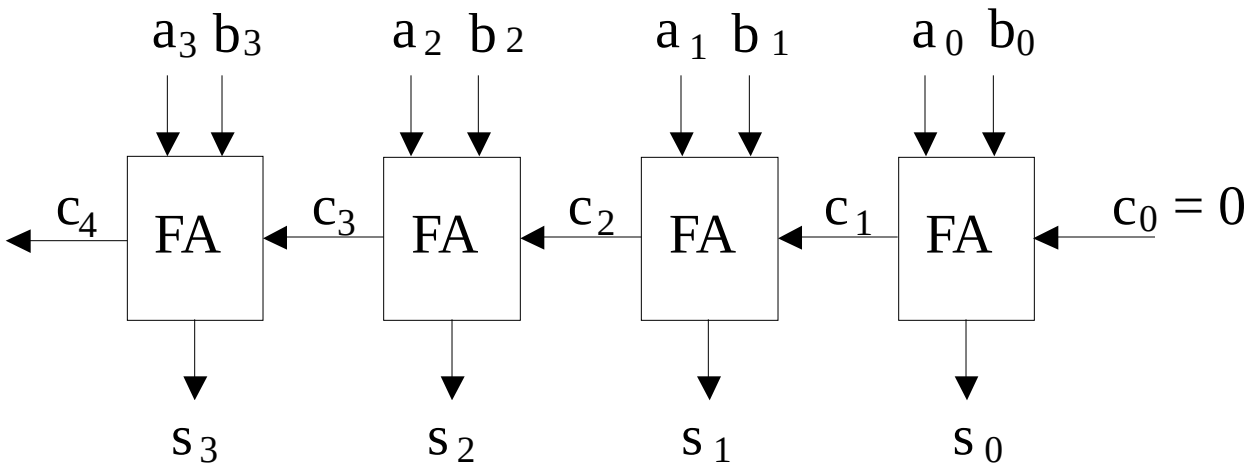
Večbitni seštevalnik

- **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})

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

    - hiba: zakasnitev
      - Dejanska zakasnitev je odvisna od operandov
      - Maksimalna zakasnitev pa narašča praktično linearno

Večbitni seštevalnik

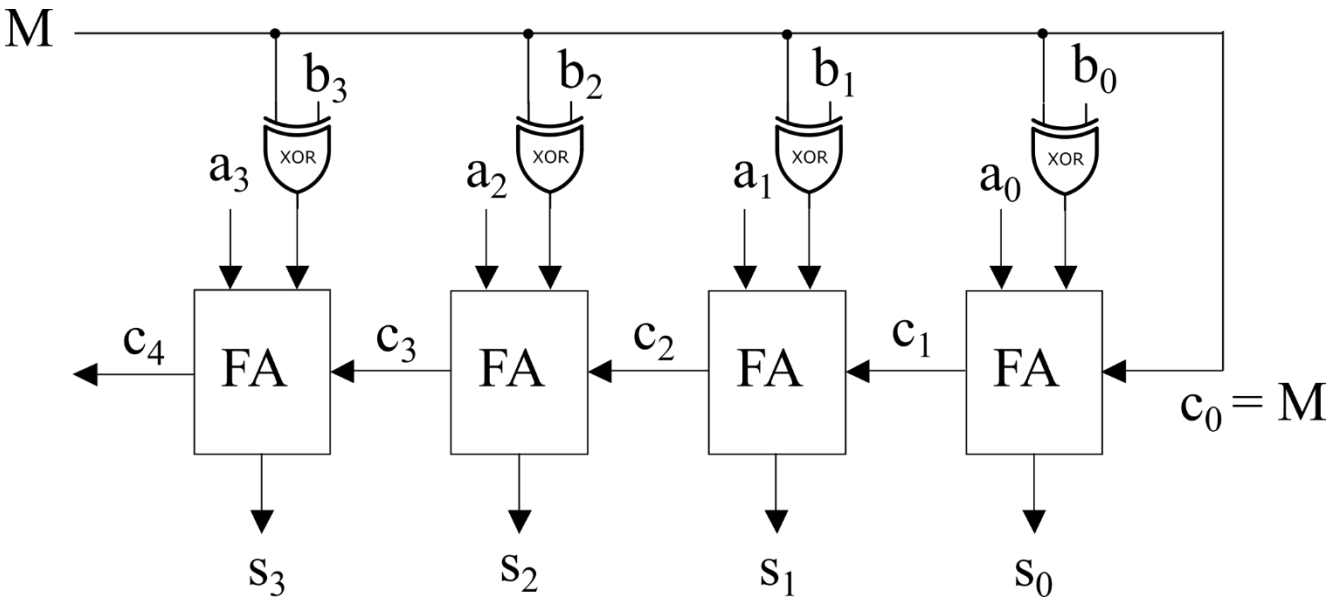


- **Seštevalnik z vnaprejšnjim prenosom** (Carry-Lookahead Adder, CLA)
  - hiter izračun vseh prenosov
    - le na osnovi vhodov a, b in c_0
  - dodatna logika
    - sprememba večnivojske oblike v dvonivojsko

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}' ... b_1' b_0') ... 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

x_2	x_1	x_0	×	y_2	y_1	y_0
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		

  - 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 v en register) – leva tabela
    - ali pa desni pomik B in levi pomik A (v tem primeru P ne pomikamo) – desna tabela

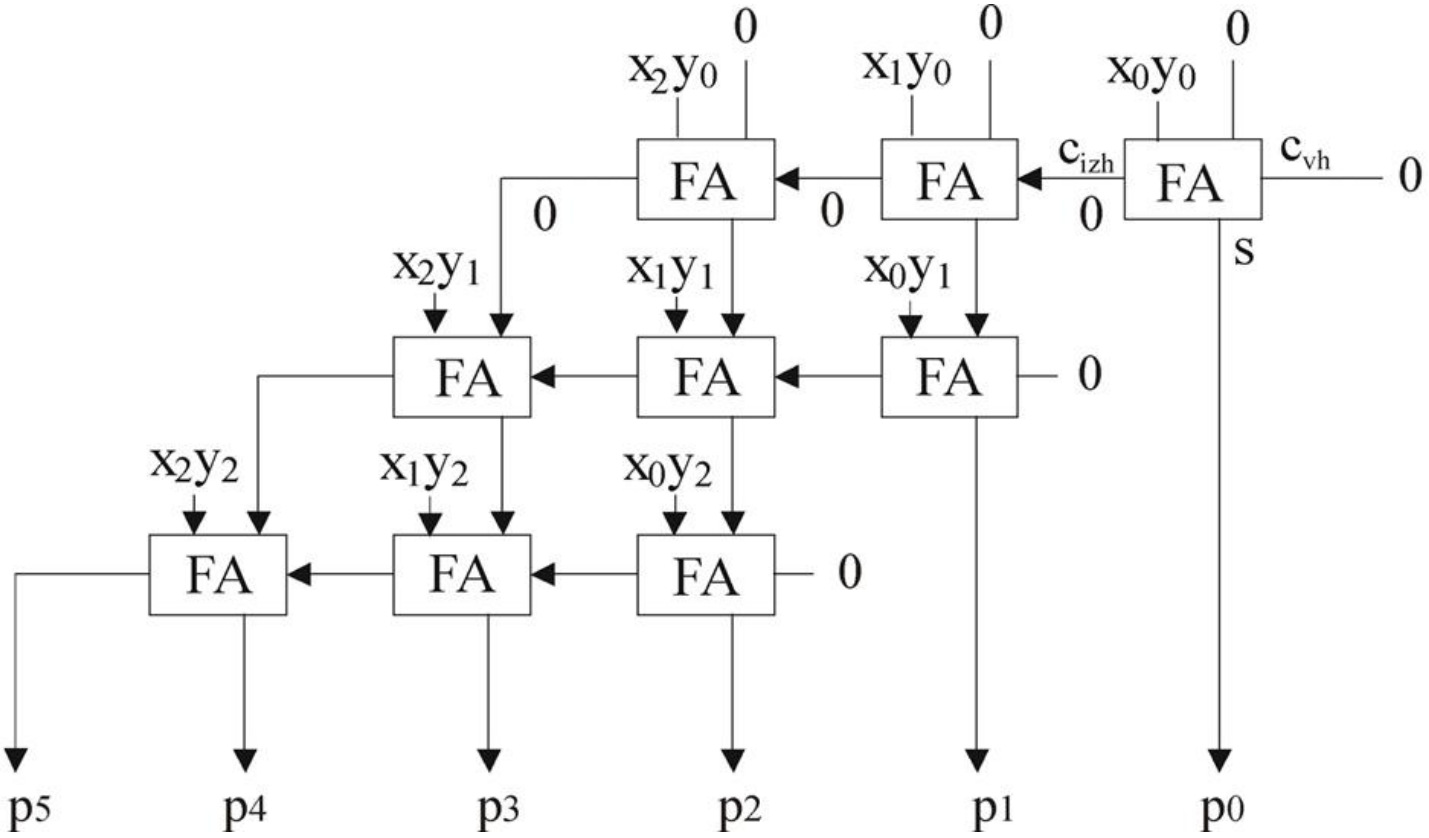
Primer množenja dveh 4-bitnih števil A(=5) in B(=6):

	P	B	
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

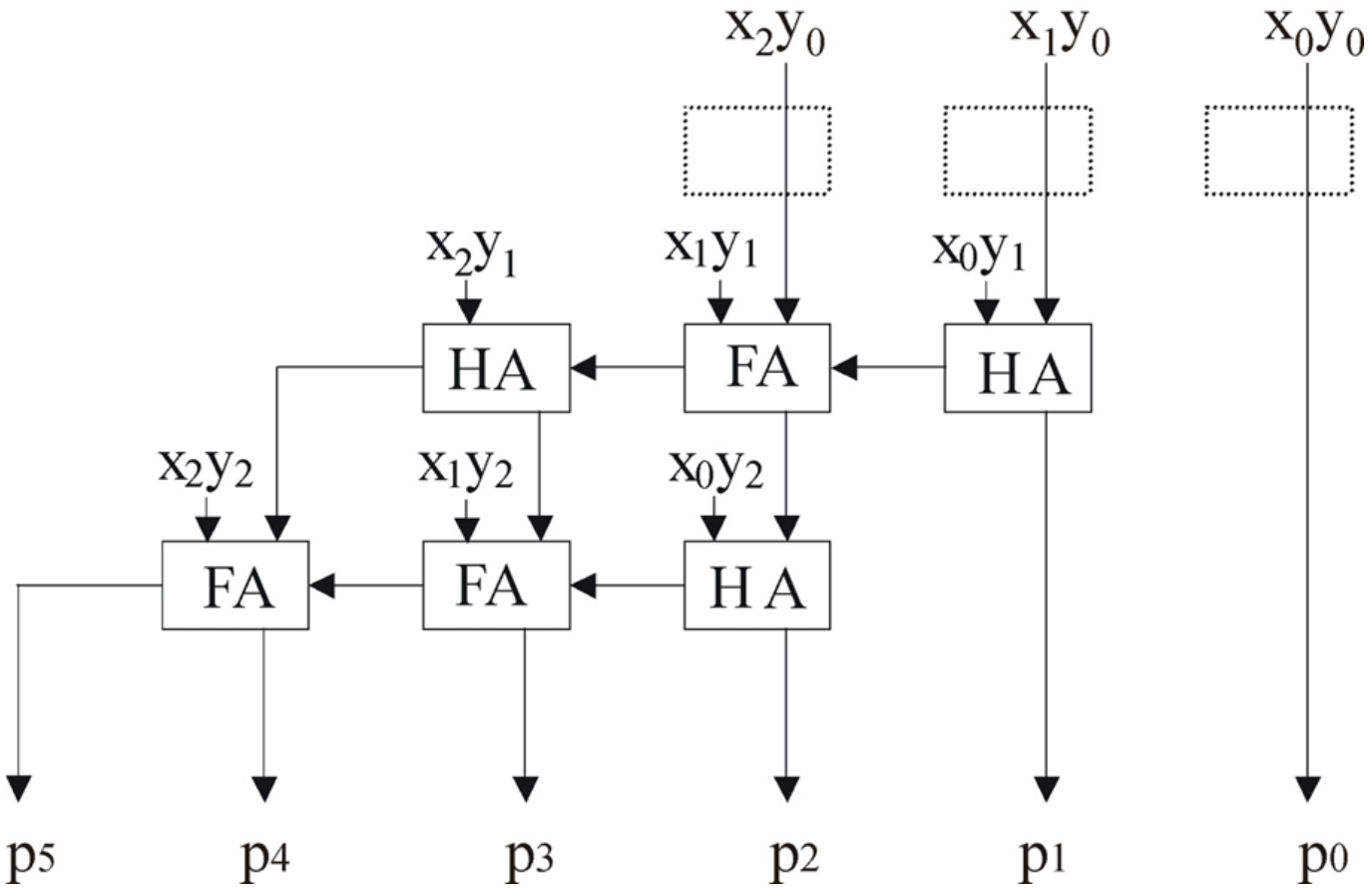
	A	B	P	
1	00000101	0110	00000000	P ← P + 0
	00001010	0011		B >> 1, A << 1
2			00001010	P ← P + A
	00010100	0001		B >> 1, A << 1
3			00011110	P ← P + A
	00101000	0000		B >> 1, A << 1
4			00011110	P ← P + 0
	01010000	0000		B >> 1, A << 1

Matrični množilnik

- na primeru 3x3



Nekateri FA so odveč:



- Zakasnitev ~ linearna
  - $(3n-2)*\Delta FA$
  - $(3n-4)*\Delta FA$

- 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)
      - CPE 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 (po absolutni vrednosti) števila kot v fiksni
  - kljub temu pa je možnih števil enako mnogo (2^n)

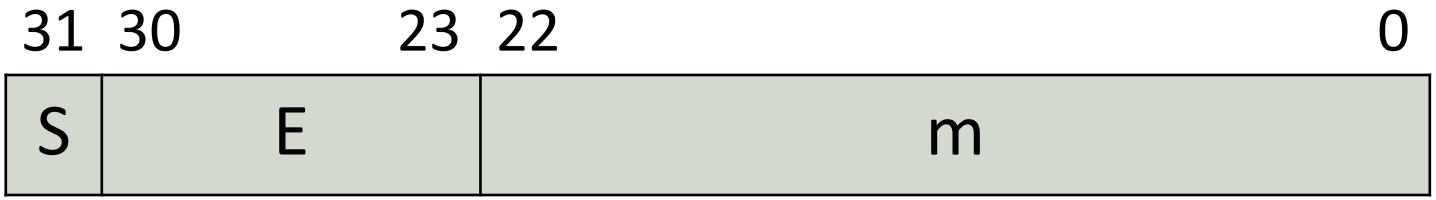
- Vsako število lahko v plavajoči vejici zapišemo na več načinov:
  - npr. $1 \times 10^{18} = 10 \times 10^{17} = 0{,}1 \times 10^{19}$ ...
  - npr. $1 \times 2^{3} = 10 \times 2^{2} = 0{,}1 \times 2^{4}$ ...
  - zato mantiso normiramo:
    - prvi bit je 1 (normalni bit), implicitno predstavljen
    - 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, 2008, 2019)
  - IEEE: Institute of Electrical and Electronics Engineers
  - 2 osnovna formata:
    - enojna natančnost (single precision), 32 bitov
    - dvojna natančnost (double precision), 64 bitov
  - Dodatni:
    - polovična (half), 16 bitov
    - 4-kratna (quadruple), 128 bitov
    - 8-kratna, 256 bitov
    - desetiški zapis, 32, 64, 128 bitov

Enojna natančnost

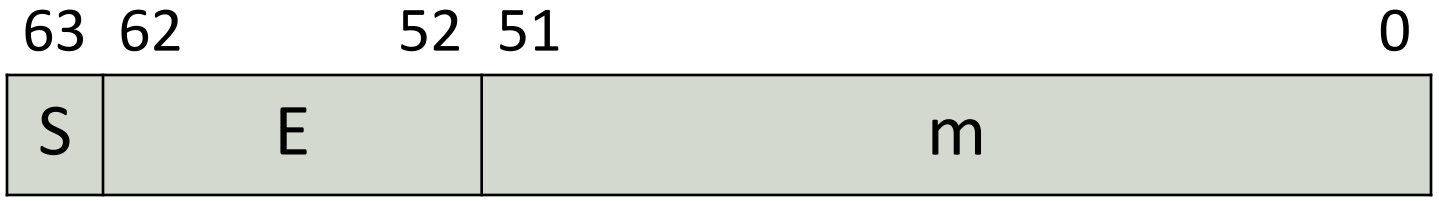
- Enojna natančnost (single precision), 32 bitov



- predznak *S* (0: +, 1: –)
- 8-bitni eksponent *e* z odmikom 127 (*e* = *E* - 127)
- 23-bitna mantisa *m* (7-mestna desetiška natančnost)
- normirana vrednost je $\mathbf{(-1)^S \cdot 1,m \cdot 2^{E-127}}$, *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



  - predznak *S* (0: +, 1: –)
  - 11-bitni eksponent *e* z odmikom 1023 (*e* = *E* - 1023)
  - 52-bitna mantisa *m* (16-mestna desetiška natančnost)
  - normirana vrednost je $\mathbf{(-1)^{S} \cdot 1,m \cdot 2^{E-1023}}$, *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	00000000000000000000000000 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	00001000000000000000000 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.

$30 = 11110,0*2^0 = 1,1110*2^4$

$3,25 = 11,01*2^0 = 1,101*2^1$

(drugo število ima manjši eksponent (2^1), zato ga povečamo na 2^4, zaradi česar se pomakne mantisa za 3 mesta)

$1,101*2^1 = 0,001101*2^4$

grs

$1,111|000*2^4$

$+\ 0,001|101*2^4$

$10,000|101*2^4$

$= 1,000|0101*2^5$ 0 V 1 = 1 (sticky)

$= 1,000|011*2^5 = 1,000*2^5 =$ **32**

(napaka 1,25)

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

$30_{10} = 11110,0*2^0 = 1,11100*2^4$

$4,125_{10} = 100,001*2^0 = 1,00001*2^2$

(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

$1,111|000*2^4$

$-\ \underline{0,010|001*2^4}$

$1,100|111*2^4 = 1,101*2^4 = \mathbf{26}_{10}$

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!)

```
  1,0 1 · 1,1 1
  1 0 1
    1 0 1
      1 0 1
  ---------
1 0,0 0 1 1
```

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: Zmnožimo C = A · B v enojni natančnosti (A = 0x326C8000, B = 0xBF200000). Zapišimo 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)

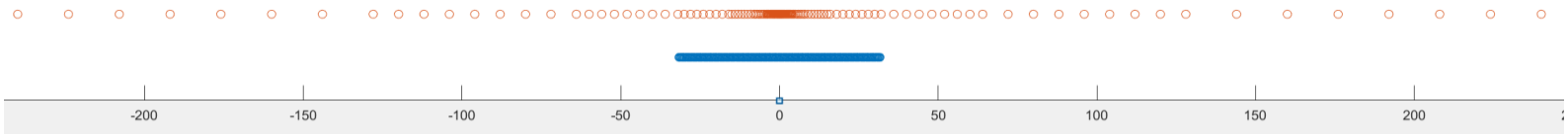
Združujemo v skupine po 4:

$C = 1\ 011|0010|0\ 001|0011|1101|0000|0000|0000$

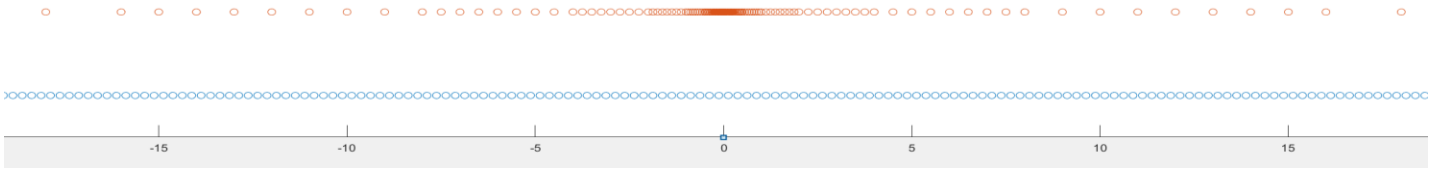
$C = B213D000_{16}$ (oz. 0xB213D000)

- Primer: plavajoča vejica v formatu *minifloat* (8-bitni)
  - predznak: 1 bit, eksponent: 4 biti, mantisa: 3 biti
  - $(-1)^S * 1{,}m * 2^{E-7}$,
    - max: $1{,}111 * 2^7 = 240$
    - min abs. norm. (razen 0): $1{,}000 * 2^{1-7} = 0{,}015625$
    - min. abs. denorm.: $0{,}001 * 2^{-6} = 0.001953125$

celoten obseg števil:



del obsega:



	Zapis	Vrednost v fiksni vejici	Vrednost v plavajoči vejici	
0	0 0000 000	0,0	0,0	Denormirana števila
Min. poz. (razen 0)	0 0000 001	0,001 = 0,125	$0{,}001*2^{-6} = 2^{-9} \approx 0{,}0020$	
	0 0000 010	0,010 = 0,250	$0{,}010*2^{-6} = 2^{-8} \approx 0{,}0039$	
		. . .		
	0 0000 111	0,111 = 0,875	$0{,}111*2^{-6} = 3*2^{-4} \approx 0{,}014$	
	0 0001 000	1,000	$1{,}000*2^{1-7} = 2^{-6} \approx 0{,}016$	Normirana števila
	0 0001 001	1,001 = 1,125	$1{,}001*2^{1-7} = 9*2^{-9} \approx 0{,}018$	
	0 0001 010	1,010 = 1,25	$1{,}010*2^{1-7} = 10*2^{-9} \approx 0{,}020$	
	0 0001 111	1,111 = 1,875	$1{,}111*2^{1-7} = 15*2^{-9} \approx 0{,}029$	
	0 0111 000	111,000 = 7,0	$1{,}000*2^{7-7} = 1{,}000*2^{0} = 1{,}000$	
	0 0111 111	111,111 = 7,875	$1{,}111*2^{7-7} = 1{,}111*2^{0} = 1{,}875$	
	0 1110 111	1110,111 = 14,875	$1{,}111*2^{14-7} = 15*2^{4} = 240$	
	0 1110 000	1110,00 = 14,000	$1{,}000*2^{14-7} = 1{,}000*2^{7} = 128$	
Max	0 1110 111	1110,111 = 14,875	$1{,}111*2^{14-7} = 1{,}111*2^{7} = 240$	

Primer: težave, v katerih se lahko znajdemo pri naivni uporabi plavajoče vejice

- Izračun variance

$$var = \frac{\sum_{i=1}^{N} x_i{}^2 - \frac{\left(\sum_{i=1}^{N} x_i\right)^2}{N}}{N-1}$$

v enojni natančnosti plavajoče vejice

- Naj bosta zaradi enostavnosti samo 2 števili (BIAS=8192, BIAS+offset=8196):
  - $x_1 = 8192 = 1{,}0*2^{13}$
  - $x_2 = 8196 = 1{,}00000000001*2^{13} = 2^{13} + 2^2 = (1{,}0 + 2^{-11})*2^{13}$

- Predpostavimo, da pristopimo k izračunu na ‘naiven’ način. Kaj se dogaja v računalniku?

$x_1{}^2 = (2^{13})^2 = 2^{26}$

$x_2{}^2 = (2^{13}+2^2)^2 = 2^{26}+2*2^{15}+2^4 = (1+2^{-10}+2^{-22})*2^{26}$

$x_1+x_2 = 2^{13} + 2^{13} + 2^2 = 2*2^{13}+2^2 = 2^{14} + 2^2$

$(x_1+x_2)^2 = 2^{28} + 2*2^{16} + 2^4 = (1 + 2^{-11} + 2^{-24})*2^{28}$

- Mantisa je 23-bitna, na poziciji z utežjo -11, pa tudi na mestu varovalnega bita (utež -24) najdemo enico:

```
1,000...1...000|100
                grs
```

- Po standardu IEEE-754 mora računalnik zaokrožiti k sodemu številu – torej navzdol. Zato enica odpade in nam ostane samo

$$(1 + 2^{-11})*2^{28} = 1{,}00000000001*2^{28} = 268\,566\,544$$

$$var = \frac{\sum_{i=1}^{N} {x_i}^2 - \frac{\left(\sum_{i=1}^{N} x_i\right)^2}{2}}{2-1} = \frac{{x_1}^2 + {x_2}^2 - \frac{(x_1+x_2)^2}{2}}{1}$$

$$= 134283280 - \frac{(16388)^2}{2}$$

- Za 16388^2 dobimo 268566528 (namesto 268566544)
- Namesto pravega rezultata (var=8) dobimo 16 (var_f v kodi)
  - Če vzamemo števili 16384 in 16388, pa je var_f = 0 (kakor tudi za nadaljnje poskuse v tej smeri) - težava je v tem, da smo s kvadriranjem izgubili natančnost
    - Zakaj pa naenkrat 0 … ?! "(/#%$!)^*9?!_\
  - V dvojni natančnosti dobivamo še prave rezultate
    - Tudi tam pa se pojavijo podobne težave, če BIAS povečamo nekje na 10^9
- Za take stvari je potrebno uporabiti nek 'ne-naiven' algoritem (npr. Welfordov, …)
  - S tem se sicer ukvarjajo pri numerični matematiki

Koda v jeziku C (cc var.c –o var && ./var)

```
#include <stdio.h>

#define N 2
#define BIAS 8192
#define offset 4

void main()
{
    int d[N];
    float a_f[N];
    float sum_f = 0.0;
    float sumsq_f = 0.0, var_f;
    double a_d[N];
    double sum_d = 0.0;
    double sumsq_d = 0.0, var_d;

    for ( int i=0; i<N; i++) {
        d[i] = offset*i;   //d0 = 0, d1 = 4

        a_f[i] = BIAS + d[i];           //8192, 8196
        sum_f += a_f[i];
        sumsq_f += a_f[i]*a_f[i];

        a_d[i] = BIAS + d[i];
        sum_d += a_d[i];
        sumsq_d += a_d[i]*a_d[i];
```