Ievads Datoru Arhitektūrā

Veselo skaitļu attēlošana un aritmētika

Tēmu saraksts

- Sešpadsmitnieku un citu skaitīšanas bāžu aritmētika
- Veselo skaitļu kodēšana un pamata darbības ar tiem

Skaitļu attēlošana dažādās skaitīšanas sistēmās

- Bāze - skaitīšanas sistēma
  - bāze norāda, kā interpretēt vērtīgākos ciparus
  - ciparu skaits - bāze (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
- Skaitļa $a_n a_{n-1} ... a_2 a_1 a_0$ *vērtība skaitīšanas sistēmā* **b** ir
  - $a_0+b*a_1+b^{2}*a_2+…+b^{n-1}*a_{n-1}+b^{n}*a_n$
  - $((…(a_n*b+ a_{n-1})*b+ a_{n-2})*b+… a_1)*b+ a_0$
- bāzi norāda skaitļa indeksā piemēram - 125_8

Piemēri

- $101_2=1+0*2+1*(2*2)=5$
- $101_5=1+0*5+1*(5*5)=26$
- $101_{10}=1+0*10+1*(10*10)=101$
- $101_{16}=1+0*16+1*(16*16)=257$

Kāpēc tieši sešpadsmitnieki?

- Skaitļotāja uzbūve
  - tehnisku iemeslu dēļ visvieglāk realizēt ierīces ar 2 dažādiem stāvokļiem (**0** un **1)**, t.i. elementārā līmenī ir **binārā** skaitīšanas sistēma
- cilvēkam binārā skaitīšanas sistēma ir pārāk smalka.
- izvēlētajai sistēmai jāļauj samērā ērti darboties un viegli iegūt bināro kodu un otrādi
  - Matemātiski pierādīts ka gadījums ja viena bāze ir otras **k**-tā pakāpe, tad pārveidošana šo bāžu starpā notiek katru atsevišķu ciparu pārveidojot par **k** cipariem un otrādi
  - tipiskie aparatūras risinājumi nosaka biežāk lietojamo bitu kombināciju garumu

Sešpadsmitnieku sistēma

Tiek lietoti sekojoši cipari 0 1 2 3 4 5 6 7 8 9 A B C D E F

hex	dec	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	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Pārveidošana sešpadsmitnieku → decimālā

- darbs pēc jau pazīstamās formulas
  - 1C2→2+12*16+1*16*16=2+192+256=450
  - 4FD→13+16*(15+4*16)= =13+16*(15+64)=13+16*80= 13+1264=1275
- ērtāk
  - zinot tabulu 100→256 F0→240 utt.
  - mākot rēķināt un lietot tabulu 1F4=200-C→512-12=500

Piemēri

- $125_{16} \rightarrow$
- $400_{16} \rightarrow$
- $7FFF_{16} \rightarrow$
- $7FFFFFFF_{16} \rightarrow$

Piemēri (atbildes)

- $125_{16} \to 293$
- $400_{16} \to 1024$
- $7FFF_{16} \to 32767$
- $7FFFFFFF_{16} \to 2147483647$

Pārveidošana decimālā $\rightarrow$ sešpadsmitnieku sistēma

- dalīšana ar atlikumu
- **13695**$_{10} \rightarrow ?_{16}$
  - 13695:16=855 atlikumā 15
  - 855:16=53 atlikumā 7
  - 53:16=3 atlikumā 5
  - 3:16=0 atlikumā 3
  - **357F**

Piemēri

- $1999_{10}=?_{16}$
- $8192_{10}=?_{16}$
- $16383_{10}=?_{16}$

Piemēri (atbildes)

- $1999_{10}=7CF_{16}$
- $8192_{10}=2000_{16}$
- $16383_{10}=3FFF_{16}$

Saskaitīšana

- Tāpat kā decimālajā sistēmā
- jāņem vērā ciparu vērtības
- pārnesuma vērtība 16

```
  167FD43
+  456789
---------
  1AD64CC
```

Atņemšana

- Tāpat kā decimālajā sistēmā
- jāņem vērā ciparu vērtības
- pārnesuma vērtība 16

```
  167FD43
-  456789
 --------
  12295BA
```

Binārā sistēma

- Kā zināms ar *m* bināriem simboliem var attēlot 2^m atšķirīgas kombinācijas. Ja mēs vēlamies attēlot negatīvus skaitļus mums jāsadala šīs kombinācijas pēc kāda principa.
- Pamatā ir divas metodes – *skaitlis-zīme* un *papildkods.*

Skaitlis-zīme

- Šis ir vienkāršākais kodēšanas veids kurā baita svarīgāko bitu lieto lai attēlotu zīmi. Ja svarīgākais bits ir 1 tad skaitlis ir negatīvs

Piemēram:

Binārais attēlojums	Vērtība
0000	+0
0001	+1
0010	+2
0011	+3
0100	+4
0101	+5
0110	+6
0111	+7
1000	-0
1001	-1
1010	-2
1011	-3
1100	-4
1101	-5
1110	-6
1111	-7

Skaitlis-zīme

- Pamatā dīvainība ir divas 0 ar + un – zīmi
- Tas var radīt problēmas ja tiek veiktas salīdzināšanas darbības.
- Tāpat visas darbības jāveic divos veidos atkarībā no zīmes bita.
- Minēto iemeslu dēļ *skaitlis-zīme* kodējumu lieto reti.

Papildkods

- Papildkods ir balstīts uz Moduļa (*M*) principa, kuru pieskaitot vai atņemot skaitlis nemaina savu vērtību. Ja ar ***n*** bitiem tiek attēlots pozitīvs skaitlis A tad zīmes bits A (n) ir 0. Pārējie (n-1) biti paliek vērtības attēlošanai tāpat kā *skaitlis-zīme* gadījumā.
- Negatīvām A vērtībām zīmes bits A(n) ir 1 bet pāri palikušie biti attēlo skaitļus no -1 līdz -2^n-1.
- Tādējādi parādās viens papildus negatīvs skaitlis (jo 0 aizņems vienu pozitīvo kodu).
- *Papildkodu iegūst invertējot visu bitu stāvokli un rezultātam pieskaitot "1".*
- Tādējādi iegūst kodu kurš garantē ka *saskaitot jebkuru pozitīvu un negatīvu skaitli* rezultātā iegūst skaitli kuru vienmēr var attēlot ar doto bitu skaitu.

Papildkods

Binārais attēlojums	Vērtība
0001	+1
0010	+2
0011	+3
0100	+4
0101	+5
0110	+6
0111	+7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

Darbības ar bināro kodu

- Saskaitīšana notiek tieši tāpat kā decimālajā sistēmā:

  0+0=0

  0+1=1

  1+1=0 (ar 1 pārnesi uz nākošo kārtu)

  1+1+1=1 (ar 1 pārnesi uz nākošo kārtu)

- *Pārneses* gadījumu var ignorēt bet *pārpildīšanās* (cenšamies attēlot lielāku vērtību nekā ietilpst dotajā bitu skaitā) gadījumus nedrīkst ignorēt.

Darbības ar bināro kodu

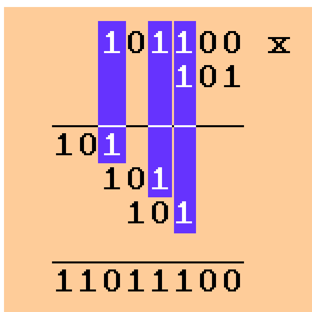
Lai noteiktu vai ir bijusi pārpildīšanās var lietot vienkāršus noteikumus:

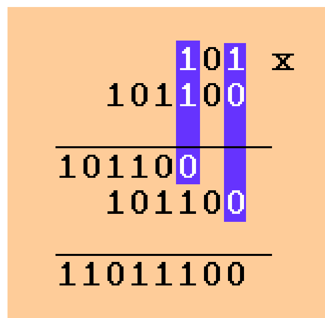
- Ja saskaitot divus pozitīvus skaitļus rezultāts ir negatīvs tad ir notikusi pārpildīšanās.
- Ja saskaitot divus negatīvus skaitļus rezultāts ir pozitīvs tad ir notikusi pārpildīšanās.
- Kā jau minēts saskaitot divus dažādu zīmju skaitļus pārpildīšanās nav iespējama.

Atņemšanu realizē kā saskaitīšanu ar skaitli kuram ir pretēja zīme.

Reizināšana un dalīšana

- Viens no veidiem ir aizstāt reizināšanu ar ciklisku saskaitīšanu bet tas ir lēni.
- Reizināt vai dalīt ar 2^n ir ļoti viegli tāpat kā reizināt vai dalīt decimālajā sistēmā ar 10^n. Šajos gadījumos var iztikt ar vienkāršām bīdes darbībām.
- Ja jāsareizina jebkuri divi pozitīvi skaitļi tad var lietot “stabiņa” principu.





- Šis princips gan nederēs ja kāds no skaitļiem būs negatīvs un dots papildkodā jo šajā gadījumā bīdes darbībām nav matemātiskas jēgas.

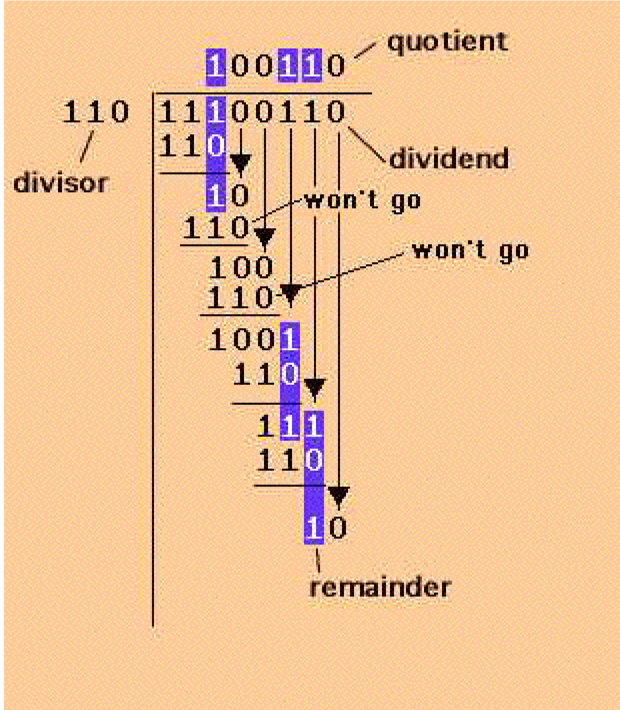
Reizināšana un dalīšana

Reizināšanai izejas ir divas:

1. Var pārvērst abus skaitļus pozitīvas vērtībās, sareizināt un rezultātu pārveidot papildkodā
2. Lietot kādu citu algoritmu (Boota algoritms)

Reizināšana un dalīšana

- Dalīšana ir līdzīga “stabiņa” principam:



Mājās

- Atrast kļūdas lekcijā
- Izlasīt:
  - http://en.wikipedia.org/wiki/Two's_complement
  - http://en.wikipedia.org/wiki/Arithmetic_overflow
  - http://en.wikipedia.org/wiki/Category:Computer_arithmetic
- Kāds ir lielākais darbību skaits dalīšanas / reizināšanas / saskaitīšanas / atņemšanas algoritmam pieņemot ka vienam operandam ir ***m*** cipari un otram operandam ir ***n*** cipari?