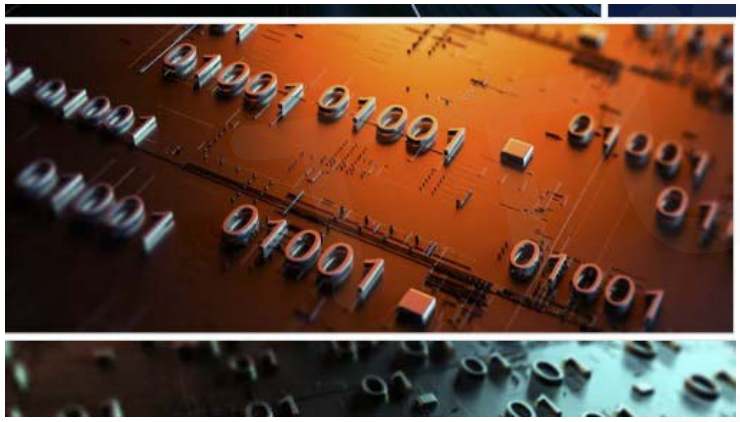
DIGITAALSÜSTEEMID

IAS0150

P. Ellervee loengutunnid nädalatel **1** . . . . . .
H. Lensen loengutunnid nädalatel **2 3 4 5 6 7**
M. Kruus loengutunnid nädalatel **8** . . . . . .



nädalatel **2 . . . . . 7** läbime loengutes teemad :

— arvude esitamine erinevates **arvusüsteemides**
— **teisendused** arvusüsteemide vahel
— **ümardamine** erinevates arvusüsteemides
— **KAHENDaritmeetika** :
liitmine lahutamine korrutamine jagamine 2ndsüsteemis
— negatiivsete arvude esitamine : **täiendkood**
— kahendarvude **nihutamine**
— **BCD-koodid** : "*loomulike kaaludega*" ja "*liiase 3-ga*"
— arvude esitamine **UJUPUNKTARVUDENA** (UPA)
— UjuPunktArvude **aritmeetika** : liitmine lahutamine korrutamine

moodles lingid *Testide* ja *Kodutööde* infole; sealhulgas :
Testid #2 #3 **Kodutöö #2**

ARVUSÜSTEEMID

vaja meenutada **Diskreetsest Matemaatikast**
positsiooniliste arvusüsteemide olemust :

Kõik olulised arvusüsteemid on *positsioonilised* ehk arvu numbrid asuvad neile ettenähtud kindlatel asukohtadel — *arvujärkudes* $\mathbf{a}_i$:

$$\ldots\ \mathbf{a}_5\ \mathbf{a}_4\ \mathbf{a}_3\ \mathbf{a}_2\ \mathbf{a}_1\ \mathbf{a}_0\ \mathbf{a}_{-1}\ \mathbf{a}_{-2}\ \mathbf{a}_{-3}\ \mathbf{a}_{-4}\ \ldots\ \mathbf{a}_i\ \ldots$$

Ainus üldtuntud *mittepositsiooniline* arvusüsteem on *rooma numbrite* süsteem numbrimärkidega **I V X L C D M**

Rooma numbrite puudus : ei ole võimalik paberil käsitsi summeerida arve :

127.9
\+ 36.8

arvusüsteemi **alus** ; järgu**kaal**
Igal positsioonilisel arvusüsteemil on olemas täisarvuline **alus p** .
Igal järgul $\mathbf{a}_i$ on **kaal** $\mathbf{p}_i$, mille saame arvusüsteemi alust **p** arvujärgu $\mathbf{a}_i$ indeksiga ***i*** astendades: $\mathbf{p}_i = \mathbf{p}^i$

järgukaalud : $\ldots\ \mathbf{p}^5\ \mathbf{p}^4\ \mathbf{p}^3\ \mathbf{p}^2\ \mathbf{p}^1\ \mathbf{p}^0\ \mathbf{p}^{-1}\ \mathbf{p}^{-2}\ \mathbf{p}^{-3}\ \mathbf{p}^{-4}\ \ldots\ \mathbf{p}^i\ \ldots$

Kui alus **p = 10** , siis on **kümnendsüsteem** , kus järkude kaaludeks on:

$$\ldots\ 10^3\ 10^2\ 10^1\ 10^0\ 10^{-1}\ 10^{-2}\ 10^{-3}\ \ldots$$
$$\ldots\ 100\ \ 10\ \ 1\ \bullet\ 0.1\ \ 0.01\ \ldots$$

täisosa • *murdosa*

kõrgemad järgud — **madalamad järgud**
vanemad järgud — *nooremad järgud*

Diskreetses matemaatikas vajasime ainult t ä i s a r v e .
Nüüd vajame ka m u r d a r v e ehk kasutusele tuleb ka m u r d o s a .

Koma näitab, kus lähevad täisarvulised järgukaalud üle murdarvulisteks (ehk kus lõppeb *täisosa* ja algab *murdosa*).

Kuigi nimetame täisosa ja murdosa eraldajat traditsiooniliselt *'komaks'*, on levinum tähemärk tema tähistamiseks *punkt* (ingl. *decimal point*).

kõrgemad ja **madalamad** järgud

suurema kaaluga järke nimetame *kõrgemateks* järkudeks ja

väiksema kaaluga järke *madalamateks* järkudeks.

Kuna me ei vaja siin aines *murdarve*, siis murdarvulise kaaluga järke me ei kasuta:

kõikide edaspidi vaadeldavate arvude madalaima järgu kaal on $\mathbf{p^0 = 1}$

numbrite ARV konkreetses arvusüsteemis

Igas järgus $\mathbf{a_i}$ saab olla **p** erinevat *numbrimärki* ehk järguväärtust.

Kui $\mathbf{p = 10}$, siis $\mathbf{a_i \in \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \}}$

. . . . aga miks meie *numbrid* **0 . . . 9** on sellise kujuga sümbolid ?

. . . . ja miks nende k o g u s (arv) on just selline ?

Igal 10ndnumbril on tema traditsiooniline *väärtus* 0 ..... 9.

Järgu väärtus on selles arvujärgus asuva *numbri* väärtus.

arv koosneb *numbritest.*

näide: arv **1024** koosneb neljast *numbrist:* **'1'** **'0'** **'2'** **'4'**

arvu VÄÄRTUS

$$\mathbf{. . . . \ a_5 \ a_4 \ a_3 \ a_2 \ a_1 \ a_0 \ a_{-1} \ a_{-2} \ a_{-3} \ a_{-4} \ . . . . \ a_i \ . . . .}$$
$$\mathbf{. . . . \ p^5 \ p^4 \ p^3 \ p^2 \ p^1 \ p^0 \ p^{-1} \ p^{-2} \ p^{-3} \ p^{-4} \ . . . . \ p^i \ . . . .}$$

mistahes positsioonilises arvusüsteemis (ehk iga aluse **p** korral) avaldub arvu **väärtus N** järgneva **korrutiste summana** :

$$\mathbf{N = ... + a_3 \cdot p^3 + a_2 \cdot p^2 + a_1 \cdot p^1 + a_0 \cdot p^0 + a_{-1} \cdot p^{-1} + a_{-2} \cdot p^{-2} + ...}$$

. . . . s.t. iga **järguväärtus** on korrutatud oma **järgukaaluga** + kokkuliidetud

näide:

10ndsüsteemne arv $\mathbf{123}_{10}$ (*indeks näitab siin arvusüsteemi*) on väärtusega "*sada kakskümmend kolm*" ainult sellepärast, et järgnev tehe annab sellise tulemuse:

$$\mathbf{1\ 2\ 3}_{10} = \mathbf{1} \cdot 100 + \mathbf{2} \cdot 10 + \mathbf{3} \cdot 1 = 123_{10}$$

mõiste "*arvu väärtus*" on ainult **10nd**süsteemne

10ndsüsteem on kõigi teiste arvusüsteemidega võrreldes tähtsas eristaatuses, kuna inimesed "tunnetavad" arve just 10ndsüsteemis.

"*väärtuse leidmine*" ja "*10ndsüsteemi teisendamine*" on sünonüümid.

Pole olemas "*kahendsüsteemset väärtust*" ega "*kaheksandsüsteemset väärtust*"; on olemas 2ndsüsteemne **esitus** ja 8ndsüsteemne **esitus**.

"kasutamata" arvujärgud $\mathbf{a_i}$ on täidetud 0-dega:

$\mathbf{123.45}_{10}$ = ... 00000**123.45**0000000 . . . . $_{10}$

Täisosa ees ja murdosa järel asuvad '**0**'-d ($a_i = 0$) ei mõjuta arvu väärtust **N** :

$$\mathbf{N = ... + a_3 \cdot p^3 + a_2 \cdot p^2 + a_1 \cdot p^1 + a_0 \cdot p^0 + a_{-1} \cdot p^{-1} + a_{-2} \cdot p^{-2} + ...}$$

(*järjestikuste arvude genereerimise / loendamise / inkrementeerimise näide **10**ndsüsteemis*)

. . . mida tähendab **i n k r e m e n t** / **inkrementeerimine** ?

Tüvenumbrid

Arvu *tüvenumbrid* on arvu numbrid alates kõrgeimast mittenullisest numbrist kuni madalaima mittenullise numbrini.

Kuigi madalaim ja kõrgeim tüvenumber pole kumbki 0 , võivad nende "vahel" olla tüvenumbriteks ka '0'-d.

näide: arvus **0.0000120003000** on tüvenumbriteks **120003** .

Üleskirjutatud arvu süsteemikuuluvuse täpsustamiseks lisame talle süsteemi näitava indeksi: 372_8 ei ole mitte *"kolmsada seitsekümmend kaks"* vaid on 8ndsüsteemne arv *"kolm-seitse-kaks"*

✈ *nüüd lahkume **10nd**süsteemist ja siseneme muudesse arvusüsteemidesse*

? kuidas saame KAHENDSÜSTEEMI ?

Asendades eelvaadatud "arvusüsteemide universaalses üldformaadis" harjumuspärase arvusüsteemi aluse **p = 10** uue alusega: **p = 2** koos kõigi sellega kaasnevate tagajärgedega, saame **kahendsüsteemi** :

KAHENDSÜSTEEM

Kahendsüsteem on lihtsaim võimalik positsiooniline arvusüsteem:

$$p = 2 \qquad a_i \in \{0,1\}$$

Kuna positsioonilises arvusüsteemis peab olema tema alusega võrdne arv numbrimärke, siis kahendsüsteemsed arvud koosnevad ainult kahest numbrist: **0** ja **1**.

järgukaalud: $\ldots\ p^5\ \ p^4\ \ p^3\ \ p^2\ \ p^1\ \ p^0\ \ p^{-1}\ \ p^{-2}\ \ p^{-3}\ \ p^{-4}\ \ldots\ p^i\ \ldots$

Arvusüsteemi aluse muutmisega kaasneb ka *järgukaalude* muutus, mis kahendsüsteemis on arvu 10 astmete asemel arvu **2** täisarvastmed:

2ndsüsteemi **järgukaalud**: $\ldots\ 2^5\ \ 2^4\ \ 2^3\ \ 2^2\ \ 2^1\ \ 2^0\ \ 2^{-1}\ \ 2^{-2}\ \ 2^{-3}\ \ldots$

32 16 8 4 2 1 0.5 0.25 0.125 . . .

Diskr.Matem me ei vajanud murdosa järke; Siin õppeaines vajame ka **murdosa !**

(. . . *järjestikuste arvude genereerimise / loendamise / inkrementeerimise näide **2nd**süsteemis* . . . .)

Järgnevalt on loetletud / "genereeritud" kõik kuni **6**-järgulised kahendarvud (ehk **2nd**arvud väärtusega **0** kuni **63** — **2nd**arvude *inkrement*-jada) :

$0_2 = 0_{10}$	$10000_2 = 16_{10}$	$100000_2 = 32_{10}$	$110000_2 = 48_{10}$
$1_2 = 1_{10}$	$10001_2 = 17_{10}$	$100001_2 = 33_{10}$	$110001_2 = 49_{10}$
$10_2 = 2_{10}$	$10010_2 = 18_{10}$	$100010_2 = 34_{10}$	$110010_2 = 50_{10}$
$11_2 = 3_{10}$	$10011_2 = 19_{10}$	$100011_2 = 35_{10}$	$110011_2 = 51_{10}$
$100_2 = 4_{10}$	$10100_2 = 20_{10}$	$100100_2 = 36_{10}$	$110100_2 = 52_{10}$
$101_2 = 5_{10}$	$10101_2 = 21_{10}$	$100101_2 = 37_{10}$	$110101_2 = 53_{10}$
$110_2 = 6_{10}$	$10110_2 = 22_{10}$	$100110_2 = 38_{10}$	$110110_2 = 54_{10}$
$111_2 = 7_{10}$	$10111_2 = 23_{10}$	$100111_2 = 39_{10}$	$110111_2 = 55_{10}$
$1000_2 = 8_{10}$	$11000_2 = 24_{10}$	$101000_2 = 40_{10}$	$111000_2 = 56_{10}$
$1001_2 = 9_{10}$	$11001_2 = 25_{10}$	$101001_2 = 41_{10}$	$111001_2 = 57_{10}$
$1010_2 = 10_{10}$	$11010_2 = 26_{10}$	$101010_2 = 42_{10}$	$111010_2 = 58_{10}$
$1011_2 = 11_{10}$	$11011_2 = 27_{10}$	$101011_2 = 43_{10}$	$111011_2 = 59_{10}$
$1100_2 = 12_{10}$	$11100_2 = 28_{10}$	$101100_2 = 44_{10}$	$111100_2 = 60_{10}$
$1101_2 = 13_{10}$	$11101_2 = 29_{10}$	$101101_2 = 45_{10}$	$111101_2 = 61_{10}$
$1110_2 = 14_{10}$	$11110_2 = 30_{10}$	$101110_2 = 46_{10}$	$111110_2 = 62_{10}$
$1111_2 = 15_{10}$	$11111_2 = 31_{10}$	$101111_2 = 47_{10}$	$111111_2 = 63_{10}$

. . . kuidas saab leida olemasoleva **2nd**arvu **väärtuse** ehk **10nd**kuju?

Ka **2nd**arvu väärtus arvutub sellesama eelnäidatud **korrutiste summana** :

$$N = \ldots + a_3 \cdot p^3 + a_2 \cdot p^2 + a_1 \cdot p^1 + a_0 \cdot p^0 + a_{-1} \cdot p^{-1} + a_{-2} \cdot p^{-2} + \ldots$$

kuid

☺ ☻ arvu **väärtuse N** leidmine osutub **2nd**arvude jaoks eriti lihtsaks:

Kuna kahendarvudes ei leidu suuremaid järguväärtusi kui **1**, siis kahendarvude korral arvu väärtust arvutav avaldis **N = ...**

(ehk teisendus **10nd**süsteemi) **lihtsustub** :
korrutamisest saab loobuda ja **summeerida tuleb ainult need järgukaalud, kus asub järguväärtus 1**

näide:

vaatleme mingit juhuslikku 4-järgulist 2ndarvu : **1 1 1 0** $_2$
. . . ja soovime teada, kui suur arv see on ? (väärtus ?)

4-järgulise täisarvu korral asub arv nendes *järkudes* a_i :

a_3	a_2	a_1	a_0
p^3	p^2	p^1	p^0

. . . . ja nende järkude **k a a l u d** on :

2ndnumbrid on ainult **0** ja **1** — mis juhtuvad mõlemad olema aritmeetikas väga "mugavad" korrutajad **!** :

$$\mathbf{1\,1\,1\,0}_2 = 1\times2^3 + 1\times2^2 + 1\times2^1 + 0\times2^0 =$$
$$= \mathbf{8} + \mathbf{4} + \mathbf{2} + \mathbf{0} = \mathbf{14}_{10}$$

. . . seega leidsime / kontrollisime järgu**kaalude** ja järgu**väärtuste** kaudu, et :

$$\mathbf{1\,1\,1\,0}_2 = \mathbf{14}_{10}$$

. . . ka eelnev (järjestikuste) **2nd**arvude loetelu näitas, et 2ndarv **1110** "tekkis" *nullist* startides i n k r e m e n t e e r i m i s e l *viieteistkümnendana* — ehk tekkis arvuks väärtusega **14** (kuna jadas "esimesena" "genereerub" arv *null* **0**) :

$0000_2 = 0_{10}$
. . . . . .

$0111_2 = 7_{10}$	$10111_2 = 23_{10}$	$100111_2 = 39_{10}$	$110111_2 = 55_{10}$
$1000_2 = 8_{10}$	$11000_2 = 24_{10}$	$101000_2 = 40_{10}$	$111000_2 = 56_{10}$
$1001_2 = 9_{10}$	$11001_2 = 25_{10}$	$101001_2 = 41_{10}$	$111001_2 = 57_{10}$
$1010_2 = 10_{10}$	$11010_2 = 26_{10}$	$101010_2 = 42_{10}$	$111010_2 = 58_{10}$
$1011_2 = 11_{10}$	$11011_2 = 27_{10}$	$101011_2 = 43_{10}$	$111011_2 = 59_{10}$
$1100_2 = 12_{10}$	$11100_2 = 28_{10}$	$101100_2 = 44_{10}$	$111100_2 = 60_{10}$
$1101_2 = 13_{10}$	$11101_2 = 29_{10}$	$101101_2 = 45_{10}$	$111101_2 = 61_{10}$
$1110_2 = 14_{10}$	$11110_2 = 30_{10}$	$101110_2 = 46_{10}$	$111110_2 = 62_{10}$
$1111_2 = 15_{10}$	$11111_2 = 31_{10}$	$101111_2 = 47_{10}$	$111111_2 = 63_{10}$

? kas väiksed kahendarvud peaks **pähe õppima** ?
kahendarve pole mõtet "pildina" pähe õppida, kuna "väikseid" kahendarve saab hetkega tuletada **järgukaalude kaudu** (täites vajalikud järgud **1**-dega)

võtame veel ühe suvalise **2nd***arvu eelnevast tabelist ja leiame ta* **väärtuse** :

$101_2 = 5_{10}$
. . . . . .

$1010_2 = 10_{10}$	$10101_2 = 21_{10}$	$100101_2 = 37_{10}$	$110101_2 = 53_{10}$
$1011_2 = 11_{10}$	$11011_2 = 27_{10}$	$101011_2 = 43_{10}$	$111011_2 = 59_{10}$
$1100_2 = 12_{10}$	$11100_2 = 28_{10}$	$101100_2 = 44_{10}$	$111100_2 = 60_{10}$
$1101_2 = 13_{10}$	$11101_2 = 29_{10}$	$101101_2 = 45_{10}$	$111101_2 = 61_{10}$
$1110_2 = 14_{10}$	$11110_2 = 30_{10}$	$101110_2 = 46_{10}$	$111110_2 = 62_{10}$
$1111_2 = 15_{10}$	$11111_2 = 31_{10}$	$101111_2 = 47_{10}$	$111111_2 = 63_{10}$

$$\mathbf{1\,0\,1\,0\,1\,1}_2 = \mathbf{?}_{10}$$

$$\mathbf{1\,0\,1\,0\,1\,1}_2 = 1\times2^5 + 0\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 =$$
$$= \mathbf{32} + \mathbf{8} + \mathbf{2} + \mathbf{1} = \mathbf{43}_{10}$$

ülesanne: _____

Leida järgnevate positiivsete 2ndarvude **väärtus**
(ehk teisendada 10ndsüsteemi)

$1\,0\,1_2 = \ldots_{10}$
$0\,1\,1\,0_2 = \ldots_{10}$
$0\,0\,0\,1\,1\,0\,1_2 = \ldots_{10}$
$0\,0\,0\,0\,1\,0\,0\,1\,1_2 = \ldots_{10}$
$0\,1\,0\,0\,0\,1\,0\,1\,1_2 = \ldots_{10}$
$0\,0\,0\,0\,0\,0\,1\,0\,1\,1\,0\,0_2 = \ldots_{10}$

$1\,0\,1_2 = \mathbf{5}_{10}$
$0\,1\,1\,0_2 = \mathbf{6}_{10}$
$0\,0\,0\,1\,1\,0\,1_2 = \mathbf{13}_{10}$
$0\,0\,0\,0\,1\,0\,0\,1\,1_2 = \mathbf{19}_{10}$
$0\,1\,0\,0\,0\,1\,0\,1\,1_2 = \mathbf{139}_{10}$
$0\,0\,0\,0\,0\,0\,1\,0\,1\,1\,0\,0_2 = \mathbf{44}_{10}$

TÄISARVU teisendus 10ndsüsteemist 2ndsüsteemi

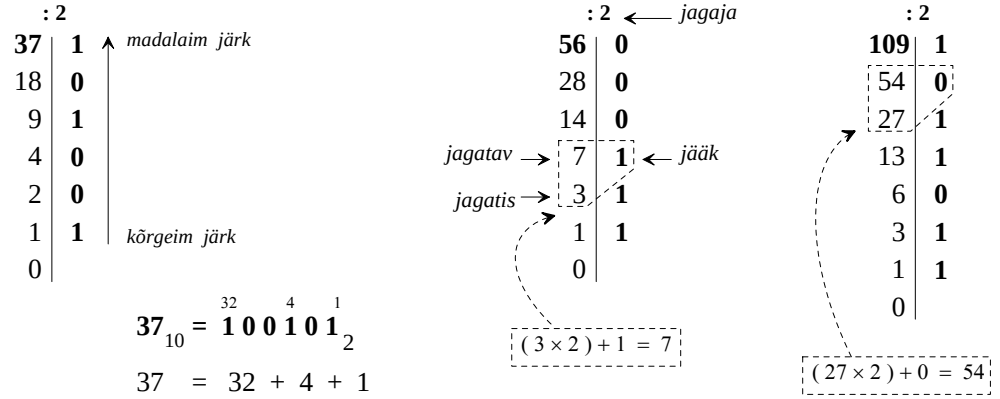
Täisarvu teisendus ühest arvusüsteemist teise toimub **uue alusega jagamise teel** kus jagamine on *täisarvuline*: murdarvu asemel saame *jagatise* ja *jäägi:*

7 : 2 = 3 (jääk **1**)

Väärtuse **N** leidmise suhtes vastupidine teisendus ehk 10ndsüsteemse täisarvu teisendamine 2ndsüsteemi toimub **2**-ga jagamise teel, kusjuures (täisarvulise) jagamise jäägid (0 ja 1) on saadava 2ndarvu järkude väärtusteks.

näide:

Teisendame 10ndtäisarvud **37**$_{10}$ **56**$_{10}$ **109**$_{10}$ 2ndkujule:



$37_{10} = {}^{32}\,{}^{4}\,{}^{1}\;\mathbf{1\,0\,0\,1\,0\,1}_2$

$37 = 32 + 4 + 1$

37$_{10}$ = **100101**$_2$ **56**$_{10}$ = **111000**$_2$ **109**$_{10}$ = **1101101**$_2$

☺ ☻ väikeste **2ndarvude kiirkoostamine** **1**de "**sobitamise**" **teel õigetesse järkudesse**:

Vajaliku arvu kahendkuju saab koostada ka järguväärtuste **1** paigutamise teel vajalikesse 2ndjärkudesse :

..... 64 32 16 8 4 2 1

peast arvutades täidame (kõrgeimast järgust alates) vajalikud järgud "*ühtedega*" nii, et **1**-ga täidetud järkude kaalude summa võrduks soovitud 10ndarvuga.

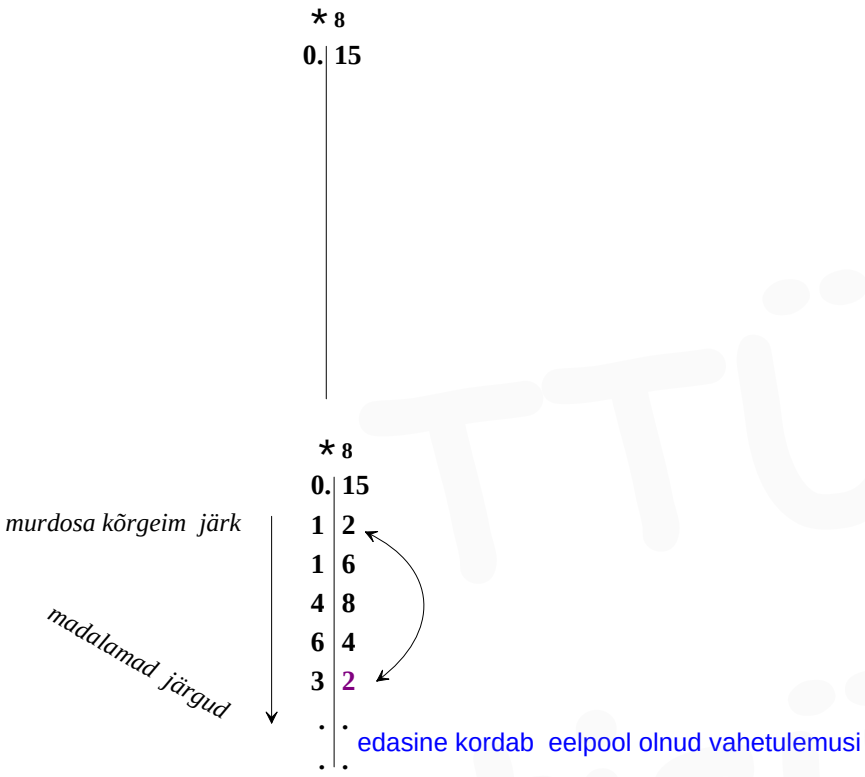
Arvu *murdosa* teisendusmeetod erineb oluliselt *täisarvu* teisendusest.

MURDARVU (arvu murdosa) teisendus 10ndsüsteemist 8ndsüsteemi

Murdosa teisendatakse uue alusega **korrutamise** teel.

näide:

Teisendame 10ndmurdosa **0.15**$_{10}$ **8nd**kujule: **0.15**$_{10}$ = **0.???**$_8$



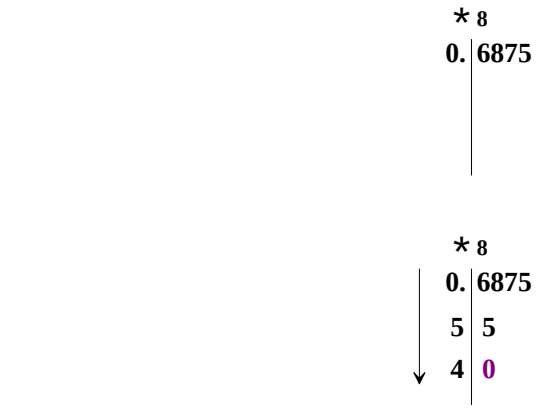
saime teisendusel :

0.15$_{10}$ = **0.11463**...$_8$ . . . . tekkib *lõpmatu perioodiline* murdosa

0.15$_{10}$ = **0.114631463**...$_8$

näide:

Teisendame 10ndmurdosa **0.6875**$_{10}$ **8nd**kujule: **0.6875**$_{10}$ = **0.???**$_8$



0.6875$_{10}$ = **0.54**$_8$

Murdosa teisendus lõppeb, kui

— ilmneb järkude (lõpmatult) kordumajääv osa ehk *periood*
(nii juhtus murdosa **esimeses** teisendusnäites)

või

— saame vahetulemuseks **0** millega edaspidi korrutades tuleksid ka kõik järgnevad madalamad järgud 0 ; (seljuhul see murdosa teisendus **täpselt**)
(nii juhtus murdosa **teises** / eelmises teisendusnäites)

"**puhtmurdarv**"

kui arvul on olemas ainult murdosa (ehk **täisosa** on arvul **0**) siis sellist arvu nimetame **puhtmurdarvuks**.

?

kuidas teisendada arvu, millel on olemas nii **täisosa** kui ka **murdosa** **?**

. . . . ehk kuidas teisendada nn. üldist / tavalist **murdarvu** :

kui teisendataval arvul on olemas nii *täisosa* kui ka *murdosa*, siis **täisosa** teisendatakse eraldi ja **murdosa** teisendatakse eraldi.

ülesanne:

Teisendada järgnevad 10ndtäisarvud **2ndkujule** :

$20_{10} = \ldots_{2}$
$85_{10} = \ldots_{2}$
$131_{10} = \ldots_{2}$
$210_{10} = \ldots_{2}$
$32_{10} = \ldots_{2}$

$20_{10} = 10100_{2}$
$85_{10} = 1010101_{2}$
$131_{10} = 10000011_{2}$
$210_{10} = 11010010_{2}$
$32_{10} = 100000_{2}$

Lisaks alustele **p = 10** ja **p = 2** on olulisemateks arvusüsteemide alusteks veel **8** ja **16**, kuna nad on mõlemad arvu 2 astmed: 2^3 ja 2^4 .

KAHEKSANDSÜSTEEM

8ndsüsteemi alus on 8 ja seega peab seal olema 8 võimalikku järguväärtust, milleks kasutatakse kaheksat esimest araabia numbrit **0** . . . **7** :

p = 8 $\quad a_i \in \{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\}$

8ndsüsteemi järgukaalud: . . . 8^4 8^3 8^2 8^1 8^0 8^{-1} 8^{-2} 8^{-3} . . .

4096 512 64 8 1 0.125

Kaheksandarve nimetatakse ka *oktaalarvudeks*.

8ndsüsteemi tähtsus on väiksem — olulisemaks osutub **16nd**süsteem.

ülesanne:

Leia nende 8ndarvude **väärtus** :

$25_8 = \ldots_{10}$
$74_8 = \ldots_{10}$
$123_8 = \ldots_{10}$

$25_8 = 2\times8^1 + 5\times8^0 = 21_{10}$
$74_8 = 7\times8^1 + 4\times8^0 = 60_{10}$
$123_8 = 1\times8^2 + 2\times8^1 + 3\times8^0 = 83_{10}$

Teisendus 10ndsüsteemist 8ndsüsteemi

10ndtäisarvude teisendus 8ndsüsteemi toimub 8-ga jagamise teel, kusjuures igal jagamissammul saadakse jäägina arvu järgmine 8ndnumber **0** . . . . **7**.

ülesanne:

Teisenda 10ndtäisarvud **236_{10}** **109_{10}** 8ndkujule:

: 8

236 |

: 8		
236	4	↑ *madalaim järk*
29	5	
3	3	*kõrgeim järk*
0		

: 8		
109	5	
13	5	
1	1	
0		

$236_{10} = {}^{64}\,{}^{8}\,{}^{1}\;3\;5\;4_{8}$ ← *järgukaalud*

$109_{10} = {}^{64}\,{}^{8}\,{}^{1}\;1\;5\;5_{8}$

(kontrollimisvõimalus)

KUUETEISTKÜMNENDSÜSTEEM

hexadecimal (hex)

Kuna **16**ndsüsteemis on arvusüsteemi alus **p = 16**, siis peab seal olema ka 16 võimalikku järguväärtust ja sellest tulenevalt ka **16 numbrimärki** nende esitamiseks.

probleem : araabia numbreid (**0 . . . 9**) on olemas ainult **10 tk !?**
6 numbrit jääb puudu **?**

lahendus :

Lisaks 10-le araabia numbrile **0 . . . 9** on ülejäänud kuueks numbrimärgiks võetud ladina tähestiku algustähed **A . . . F** :

p = 16 $\quad a_i \in \{ 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ A\ B\ C\ D\ E\ F \}$

16ndnumbrid **A . . . F** omavad väärtusi:

A = 10 **B** = 11 **C** = 12 **D** = 13 **E** = 14 **F** = 15

16ndsüsteemi järgukaalud: . . . 16^4 16^3 16^2 16^1 16^0 16^{-1} 16^{-2} . . .

65536 4096 256 16 1 0.0625

. . . . oleme märganud, et järk kaaluga **1** on olemas igas arvusüsteemis:

p = 10 :	. . . . .	**10000**	**1000**	**100**	**10**	**1**
p = 2 :	. . . . .	**16**	**8**	**4**	**2**	**1**
p = 8 :	. . . . .	**4096**	**512**	**64**	**8**	**1**
p = 16 :	. . . . .	**65536**	**4096**	**256**	**16**	**1**

. . . . p^5 p^4 p^3 p^2 p^1 p^0 p^{-1} p^{-2} p^{-3} p^{-4} . . . . p^i . . . .

$p^0 = 1$ suvalise **p** korral

ülesanne:

Teisenda järgnevad 16ndarvud **10ndkujule** (ehk leia väärtus) :

$1\,2_{16}$ = . . . $_{10}$
$4\,D_{16}$ = . . . $_{10}$
$A\,6_{16}$ = . . . $_{10}$

$F\,F_{16}$ = . . . $_{10}$
$1\,0\,0_{16}$ = . . . $_{10}$
$1\,C\,D_{16}$ = . . . $_{10}$

$1\,2_{16} = 1\times16^1 + 2\times16^0 = \mathbf{18}_{10}$
$4\,D_{16} = 4\times16^1 + 13\times16^0 = \mathbf{77}_{10}$
$A\,6_{16} = 10\times16^1 + 6\times16^0 = \mathbf{166}_{10}$
$F\,F_{16} = \mathbf{255}_{10}$
$1\,0\,0_{16} = \mathbf{256}_{10}$
$1\,C\,D_{16} = \mathbf{461}_{10}$

16ndsüsteem on "suurim" praktiliselt kasutatav arvusüsteem.

16ndarve esitatakse tekstides / tekstifailides / programmeerimiskeeltes eesliitega **0x**..... või tagaliitega .....**h** .....**H**

näide: **0x7E** **7Eh** **7EH**

Võimalik on koostada arvusüsteeme ka suurema alusega kui 16 , kuid selliseid suuremaid arvusüsteeme pole vaja.

Teisendus 10ndsüsteemist 16ndsüsteemi

10ndtäisarvude teisendus 16ndsüsteemi toimub **16**-ga jagamise teel, kusjuures igal jagamissammul saadakse jäägina arvu järgmine 16ndnumber **0 . . . . F**.

ülesanne:

Teisendada järgnevad 10ndarvud **16ndkujule** :

$77_{10} = \ldots_{16}$
$32_{10} = \ldots_{16}$
$256_{10} = \ldots_{16}$

: 16

77

$77_{10} =$ **$4D_{16}$**
$32_{10} =$ **20_{16}**
$256_{10} =$ **100_{16}**

Edaspidi vajame nendest arvusüsteemidest kõige rohkem **kahendsüsteemi**

ARVUSÜSTEEMID

kokkuvõttev loetelu

kuni arvutite ilmumiseni polnud vaja muid arvusüsteeme peale **10nd**süsteemi

$2 \le p \le 16$

2ndsüsteem: p = 2 $a_i \in \{ 0\ 1 \}$
3ndsüsteem: p = 3 $a_i \in \{ 0\ 1\ 2 \}$
4ndsüsteem: p = 4 $a_i \in \{ 0\ 1\ 2\ 3 \}$
5ndsüsteem: p = 5 $a_i \in \{ 0\ 1\ 2\ 3\ 4 \}$
6ndsüsteem: p = 6 $a_i \in \{ 0\ 1\ 2\ 3\ 4\ 5 \}$
7ndsüsteem: p = 7 $a_i \in \{ 0\ 1\ 2\ 3\ 4\ 5\ 6 \}$
8ndsüsteem: p = 8 $a_i \in \{ 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7 \}$
9ndsüsteem: p = 9 $a_i \in \{ 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8 \}$
10ndsüsteem: p = 10 $a_i \in \{ 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9 \}$
11ndsüsteem: p = 11 $a_i \in \{ 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ A \}$
12ndsüsteem: p = 12 $a_i \in \{ 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ A\ B \}$
13ndsüsteem: p = 13 $a_i \in \{ 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ A\ B\ C \}$
14ndsüsteem: p = 14 $a_i \in \{ 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ A\ B\ C\ D \}$
15ndsüsteem: p = 15 $a_i \in \{ 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ A\ B\ C\ D\ E \}$
16ndsüsteem: p = 16 $a_i \in \{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ A\ B\ C\ D\ E\ F\}$

olulised arvusüsteemid: **$p = 2 = 2^1$** (kus **$p = 2^n$**)
$p = 8 = 2^3$
p = 10
$p = 16 = 2^4$

. . . muid arvusüsteeme praktikas ei kasutatagi . . .

Teisendus 2ndsüsteemist 8ndsüsteemi või 16ndsüsteemi

siin on loetelu arvude **0 . . . . 15** esitustega neljas erinevas arvusüsteemis :

10nd	**16**nd	**2**nd	**8**nd	**2**nd
0	0	0000	**0**	**000**
1	1	0001	**1**	**001**
2	2	0010	**2**	**010**
3	3	0011	**3**	**011**
4	4	0100	**4**	**100**
5	5	0101	**5**	**101**
6	6	0110	**6**	**110**
7	7	0111	**7**	**111**
8	8	**1000**	10	
9	9	**1001**	11	
10	A	**1010**	12	
11	B	**1011**	13	
12	C	**1100**	14	
13	D	**1101**	15	
14	E	**1110**	16	
15	F	**1111**	17	

2ndsüsteemi, **8nd**süsteemi ja **16nd**süsteemi alused on arvu **2** täisarvastmed: $\mathbf{p = 2^1}$ $\mathbf{p = 2^3}$ $\mathbf{p = 2^4}$.

See annab neile kasuliku lisaomaduse, võimaldades nende süsteemide omavahelisi arvuteisendusi teha ka **numbrimärkide asendamise teel** ehk ilma "uue alusega" jagamata.

?. . . kuidas saame **2nd**arvu viia **8nd**kujule ?

võimalik oleks teisendada **2nd** → **10nd** → **8nd** kuid see oleks asjatu töö

2ndarvu on võimalik teisendada (ümber kirjutada) tema **8nd**kujule, asendades (alates **2nd**arvu **madalamatest** järkudest)

iga tema järkude<u>kolmiku</u> **000 .... 111** vastava **8nd**numbriga **0 .... 7** nagu näitas eelnev vastavustabel :

10nd	**16**nd	**2**nd	**8**nd	**2**nd
0	0	0000	**0**	**000**
1	1	0001	**1**	**001**
2	2	0010	**2**	**010**
3	3	0011	**3**	**011**
4	4	0100	**4**	**100**
5	5	0101	**5**	**101**
6	6	0110	**6**	**110**
7	7	0111	**7**	**111**
8	8	**1000**	10	
9	9	**1001**	11	
10	A	**1010**	12	
11	B	**1011**	13	
12	C	**1100**	14	
13	D	**1101**	15	
14	E	**1110**	16	
15	F	**1111**	17	

näide:

Võtame suvalise 2ndarvu: $\mathbf{1\,0\,1\,1\,0\,1\,0\,1\,0\,0\,1\,1\,1}_2$ eesmärgiga viia see arv **8nd**kujule ja seejärel ka **16nd**kujule.

viimine / ümberkirjutamine **8nd**kujule :

parim teisendusviis: Grupeerime **2nd**arvu järgud <u>3-järgulistesse</u> gruppidesse alates <u>madalamatest</u> järkudest :

$$\mathbf{1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1}_2$$

. . . . . lisades vajadusel arvu ette 0-lle:

$$0\ 0\ \mathbf{1}\ |\mathbf{0\ 1\ 1}\ |\mathbf{0\ 1\ 0}\ |\mathbf{1\ 0\ 0}\ |\mathbf{1\ 1\ 1}$$

. . . . edasi asendame 2ndjärkude iga grupeeritud kolmiku temaga väärtuselt võrdse **8nd**numbriga **0 .... 7** :

(eelpoolses vastavustabelis read: **000** kuni **111**; **0** kuni **7**)

1 3 2 4 7
001|**011**|**010**|**100**|**111**

seega **1011010 100111**$_2$ = **13247**$_8$

! *viga:*

alates **kõrgematest** järkudest ei tohi grupeerida :
101|**101**|**010**|**011**|**1**

?. . . miks ei tohi grupeerida 2ndjärke alates **kõrgematest** järkudest ?

... kuna **täis**arvu lõppu ei tohi lisada **0**-lle — see "rikuks" arvu väärtuse !

sama **2nd**arvu viimine / ümberkirjutamine **16nd**kujule :

Selleks grupeerime **2nd**arvu järgud 4 järgu kaupa alates madalamatest järkudest ja

asendame iga 2ndjärkude neliku temaga väärtuselt võrdse **16nd**numbriga **0** .... **F** :

juhindudes vastavustabelist **2nd**süsteemi ja **16nd**süsteemi vahel:

10nd	16nd	2nd	8nd	2nd
0	0	0000	0	000
1	1	0001	1	001
2	2	0010	2	010
3	3	0011	3	011
4	4	0100	4	100
5	5	0101	5	101
6	6	0110	6	110
7	7	0111	7	111
8	8	1000	10	
9	9	1001	11	
10	A	1010	12	
11	B	1011	13	
12	C	1100	14	
13	D	1101	15	
14	E	1110	16	
15	F	1111	17	

1 0 1 1 0 1 0 1 0 0 1 1 1$_2$

1 6 A 7
0001|**0110**|**1010**|**0111**

seega **1011010 100111**$_2$ = **16A7**$_{16}$

ülesanne:

Kontrollida eelnevalt leitud arvude võrdsust, leides eelmise näite **2nd**arvu, **8nd**arvu ja **16nd**arvu **väärtused** :

1 0 1 1 0 1 0 1 0 0 1 1 1$_2$ = . . . $_{10}$

1 3 2 4 7$_8$ = . . . $_{10}$

1 6 A 7$_{16}$ = . . . $_{10}$

arvestades järgukaalusid nendes arvusüsteemides :

1011010 100111$_2$ = $2^{12} + 2^{10} + 2^9 + 2^7 + 2^5 + 2^2 + 2^1 + 2^0$ = ... $_{10}$

1011010 100111$_2$ = $2^{12} + 2^{10} + 2^9 + 2^7 + 2^5 + 2^2 + 2^1 + 2^0$ = **5799**$_{10}$

13247$_8$ = $1\times8^4 + 3\times8^3 + 2\times8^2 + 4\times8^1 + 7\times8^0$ = ... $_{10}$

13247$_8$ = $1\times8^4 + 3\times8^3 + 2\times8^2 + 4\times8^1 + 7\times8^0$ = **5799**$_{10}$

16A7$_{16}$ = $1\times16^3 + 6\times16^2 + 10\times16^1 + 7\times16^0$ = ... $_{10}$

$$\mathbf{16A7}_{16} = \mathbf{1}\times16^3 + \mathbf{6}\times16^2 + \mathbf{10}\times16^1 + \mathbf{7}\times16^0 = \mathbf{5799}_{10}$$

(kõik need 3 arvu on **võrdsed**)

---

sellise asendamisega saab teisendada ka "vastupidises suunas" :

numbrite **vastupidise** asendamisega kahendjärkude kolmikuteks või nelikuteks saab arvu **8nd**kujult või **16nd**kujult kergesti üle minna tema **2nd**kujule: **8nd** ⟶ **2nd** **16nd** ⟶ **2nd**

?. . . aga **4nd**süsteem ? see on ju kah alusega $\mathbf{p = 2^n}$?

4ndsüsteem (ebaoluline)

Sarnast arvujärkude asendamist saab rakendada ka 4ndsüsteemiga tegeledes, sest arvusüsteemi alus $\mathbf{p = 4 = 2^2}$.

4ndsüsteemi ja **2nd**süsteemi siduv vastavustabel oleks:

2nd	4nd
$\mathbf{00}_2$	$\mathbf{0}_4$
$\mathbf{01}_2$	$\mathbf{1}_4$
$\mathbf{10}_2$	$\mathbf{2}_4$
$\mathbf{11}_2$	$\mathbf{3}_4$

$\mathbf{0\,1\,0\,0\,1\,0\,1\,1}_2$ seega **4nd**süsteemi ümberkirjutamiseks vaja grupeerida :

0 1 | 0 0 | 1 0 | 1 1

4ndsüsteem ei ole oluline arvusüsteem ja praktikas teda ei kasutata.

?. . . aga kui grupeerida 2ndarvu järgud **ühe**kaupa ?

1|0|1|1|0|1|0|1|0|0|1|1|1

. . . siis saame mõttetu "teisenduse" 2ndsüsteemist . . . **2nd**süsteemi **!**

Programmeerimiskeeled võimaldavad arvude ("*arvkonstantide*") esitamist

10ndsüsteemis

2ndsüsteemis

8ndsüsteemis

16ndsüsteemis

. . . . . kuid mitte 4ndsüsteemis.

?. . . kas on võimalik **8nd**arv ümberkirjutada **16nd**kujule ka "**otse**" ehk ilma **2nd**süsteemi "läbimata" **?**

! **8nd**süsteemi ja **16nd**süsteemi vaheline "otseteisendus" (numbrite asenduse teel) pole võimalik: ainult **2nd**süsteemi kaudu saab.

$\mathbf{13247}_8$ ⟷ $\mathbf{16A7}_{16}$

OTSE ei saa

saab nii : **8nd** ⟷ **2nd** ⟷ **16nd**

16ndsüsteemi tähtsus

Arvutimälus hoitakse andmeid **baitides**, mis on **8**-järgulised kahendkoodid. 16ndsüsteem võimaldab esitada (näidata) baitide sisu (ja üldse igasuguseid kahendkoode) palju kompaktsemalt võrreldes nende "vahetu" esitamisega kahendkujul.

vaatleme kõikvõimalikke koode mis saavad olla *baidis* :

$\mathbf{00000000}_2$

$\mathbf{00000001}_2$

$\mathbf{00000010}_2$

$\mathbf{00000011}_2$

.

.

$\mathbf{01111001}_2$

$\mathbf{01111010}_2$

$\mathbf{01111011}_2$

01111100_2

.

.

11111100_2

11111101_2

11111110_2

11111111_2

jaotame baidi kõrgemaks ja madalamaks **poolbaidiks** :

00000000_2

00000001_2

00000010_2

00000011_2

.

.

01111001_2

01111010_2

01111011_2

.

.

11111101_2

11111110_2

11111111_2

Mõlema *poolbaidi* saab asendada vastava **16ndnumbriga** **0** . . . . **F** :

$00000000_2 = 00_{16}$

$00000001_2 = 01_{16}$

$00000010_2 = 02_{16}$

$00000011_2 = 03_{16}$

.

$01111001_2 = 79_{16}$

$01111010_2 = 7A_{16}$

$01111011_2 = 7B_{16}$

.

.

$253_{10} = 11111101_2 = FD_{16} = 253_{10}$

$11111110_2 = FE_{16}$

$11111111_2 = FF_{16}$

Baidi mistahes võimalikku sisu / koodi saab seega esitada **kahejärgulise 16ndarvuna**: (*suvalised juhuslikud näitebaidid*)

$10110111_2 = B7_{16}$ $\quad$ $00000101_2 = 05_{16}$

$11100100_2 = E4_{16}$ $\quad$ $01101010_2 = 6A_{16}$

$11111110_2 = FE_{16}$ $\quad$ $11111111_2 = FF_{16}$

Kui arvutimälu sisu tuleb kuidagi visuaalselt näidata, siis eelistatakse mälus tegelikult asuvate **1**-de ja **0**-de näitamise asemel esitada mälubaitides asuvate **2nd**arvudega võrdseid **16nd**arve.

16ndsüsteemi kasutatakse **2nd**arvude kompaktsemaks esitamiseks

ülesanne:

Esitada 8ndarv **7433_8** **2nd**süsteemis ja **16nd**süsteemis:

$7433_8 = ?_2 = ?_{16}$ Leida selle arvu väärtus.

ülesanne:

Esitada 2ndarv **1101101101_2** **4nd** , **8nd** ja **16nd**süsteemis:

1101101101_2 = **$?_4$** = **$?_8$** = **$?_{16}$** Leida selle arvu väärtus.

1 1 0 1 1 0 1 1 0 1_2 =

ülesanne:

Teisendada **10nd**arv **155_{10}** **16nd**süsteemi: **155_{10}** = **$?_{16}$**

(jagamisel tasub kasutada kalkulaatori abi)

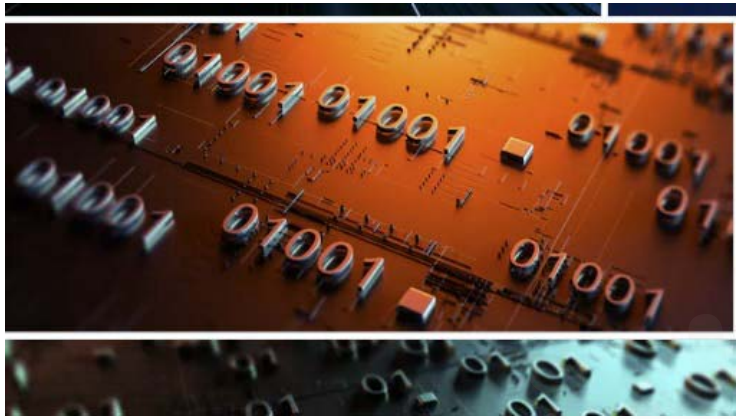
: 1 6

155 |

P. Ellervee loengutunnid nädalatel **1** . . . . . .

H. Lensen loengutunnid nädalatel **2 3 4 5 6 7**

M. Kruus loengutunnid nädalatel **8** . . . . . .



KAHENDARITMEETIKA

4 *aritmeetilist tehet* on : *liitmine* *lahutamine* *korrutamine* *jagamine*

LIITMINE **2ndsüsteemis** : summajärgu ja ü l e k a n n e t e tekkimine lihtsaim liitmistehe:

$1_{10} + 1_{10} = 2_{10}$ (**10nd**süsteemis)

$1_2 + 1_2 = 10_2$ (sama liitmine **2nd**süsteemis)

$1_2 + 1_2 = 10_2$ **2nd**liitmise järgud:

1

+ 1 2

..... 1 2

..... 0 2

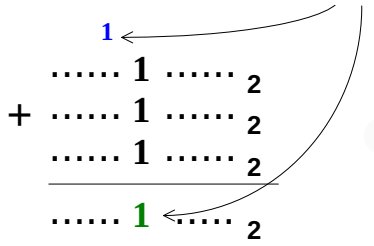
jooksva summajärgu
ja
ülekande
tekkimine

veidi "keerulisem" liitmistehe:

$1_{10} + 1_{10} + 1_{10} = 3_{10}$ (**10nd**süsteemis)

$1_2 + 1_2 + 1_2 = 11_2$ (sama liitmine **2nd**süsteemis)

$1_2 + 1_2 + 1_2 = 11_2$ (summajärk ja ülekanne)



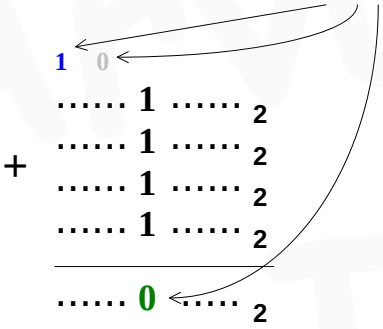
jooksva summajärgu
ja
ülekande
tekkimine

veel "keerulisem" liitmistehe:

$1_{10} + 1_{10} + 1_{10} + 1_{10} = 4_{10}$ (**10nd**süsteemis)

$1_2 + 1_2 + 1_2 + 1_2 = 100_2$ (sama liitmine **2nd**süsteemis)

$1_2 + 1_2 + 1_2 + 1_2 = 100_2$ (summajärk ja ülekanded)



jooksva summajärgu
ja
ülekande
tekkimine

LAHUTAMINE 2ndsüsteemis :

mis toimub j o o k s v a s järgus ja kõrgemates naaberjärkudes ?

$1_2 - 1_2 = 0_2$

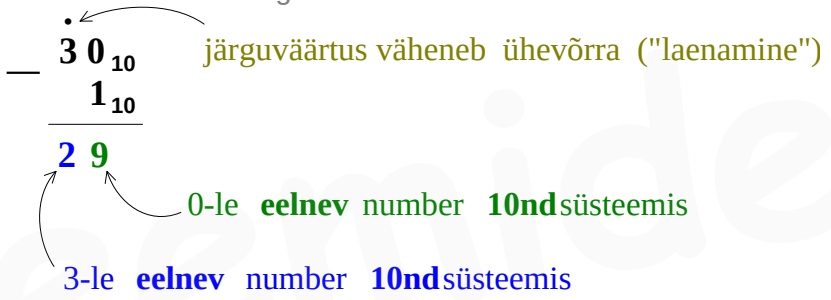
$1_2 - 0_2 = 1_2$

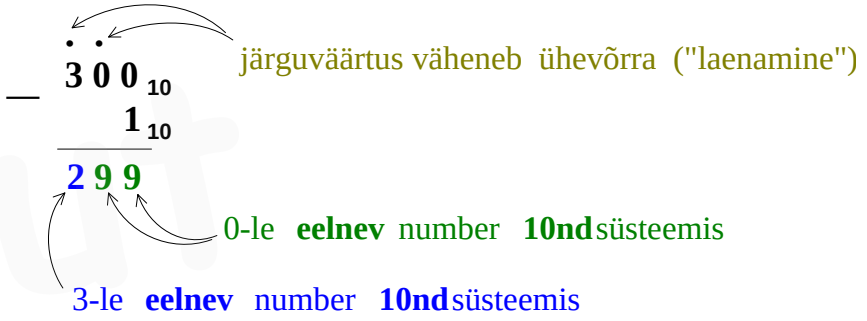
meenutame, et **10nd**süsteemi numbrid (ehk "**10nd**numbrid") on :

. . . . . 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 . . .

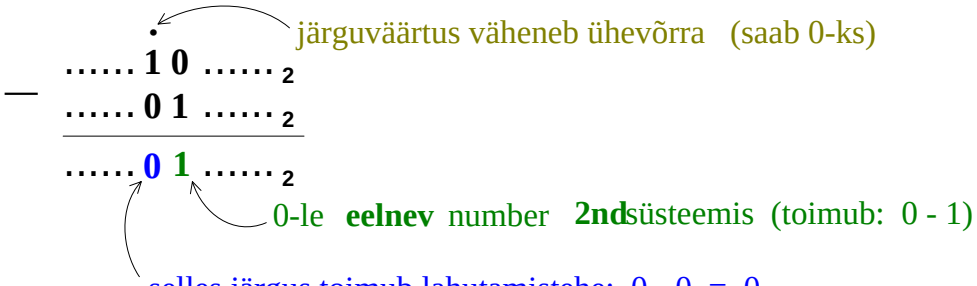
eelneval real on neid korduvana nii palju selleks, et rõhutada :
9-le järgneb **0** ja **0**-le eelneb **9** (**10nd**süsteemis)

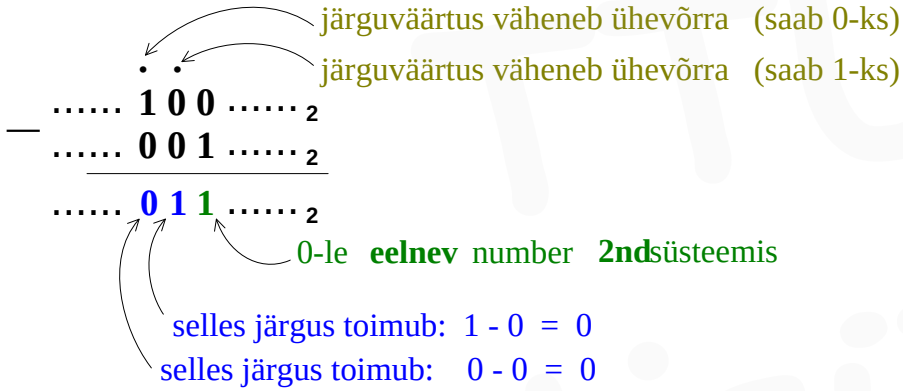
võrdlus 10ndsüsteemi lahutamisega:





täpselt sama põhimõte rakendub ka **2 n d** s ü s t e e m i s lahutamisel
kus **2nd**numbrid on . . . . . **0 1 0 1 0 1 0 1 0** . . . . .
ehk **0**-le **järgneb 1** ja (erinevalt 10ndsüsteemist) **0**-le **e e l n e b 1** :





. . . . allpool saame neid põhimõtteid rakendada **2nd**süsteemis "käsitsi arvutamisel"

ÜMARDAMINE erinevates arvusüsteemides

(NB! alati ümardatakse **m u r d o s a** , mitte *täisosa*)

Murdarvude teisendamisel nägime olukorda, kus teisendustulemus oli l õ p m a t u *perioodiline* **2nd**murdarv.

Lõpmatu perioodi kirjapanemisel tuleb arv kuskil järgu juures "äralõigata" ja allesjääv arvu osa tuleb korrektselt ü m a r d a d a (. . . et murdosa lõpu "äralõikamisel" tekkiv viga oleks väiksem)

ümardamine kui p on paarisarv ?

numbrite väiksem pool *suurem pool*

10ndsüsteem: **p = 10** $a_i \in \{ 0\ 1\ 2\ 3\ 4 \quad 5\ 6\ 7\ 8\ 9 \}$

täisosa . *murdosa* . . . 234**999999999999999**. . . $_{10} \approx$ . . . 23_{10}

täisosa . *murdosa* . . . 235**000000000000000**. . . $_{10} \approx$ . . . 24_{10}

8ndsüsteem: **p = 8** $a_i \in \{ 0\ 1\ 2\ 3 \quad 4\ 5\ 6\ 7 \}$

täisosa . *murdosa* . . . 123**777777777777777**. . . $_8 \approx$ . . . 12_8

täisosa . *murdosa* . . . 124**000000000000000**. . . $_8 \approx$ . . . 13_8

16ndsüsteem: **p = 16**

$a_i \in \{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7 \quad 8\ 9\ A\ B\ C\ D\ E\ F\}$

täisosa . *murdosa* . . . 567**FFFFFFFFFFFF**. . . $_{16} \approx$ . . . 56_{16}

täisosa . *murdosa* . . . 568**000000000000000**. . . $_{16} \approx$ . . . 57_{16}

kui **p** = *paarisarv* , siis ümardamiseks on alati piisav arvestada ainuüksi **ESIMEST mittemahtuvat** / ärajäävat järku

ehk ümardamine toimub alati **üheainsa** (ehk "esimese ärajääva") järgu alusel

ÜMARDAMINE erinevates arvusüsteemides

kui p on paaritu arv ?

. . . kõik olulised arvusüsteemid on siiski **paaris**arvulise alusega . . .
paarituarvulise alusega o l u l i s i arvusüsteeme — ei ole

numbrite väiksem "pool" *suurem "pool"*

9ndsüsteem: **p = 9** $a_i \in \{ 0\ 1\ 2\ 3\ \ 4\ \ 5\ 6\ 7\ 8 \}$

täisosa . *murdosa* . . . 56**444444444444443**. . . $_9 \approx$ . . . 56_9

täisosa . *murdosa* . . . 56**444444444444445**. . . $_9 \approx$ . . . 57_9

7ndsüsteem: **p = 7** $a_i \in \{ 0\ 1\ 2\ \ 3\ \ 4\ 5\ 6 \}$

täisosa . *murdosa* . . . 45**333333333333332**. . . $_7 \approx$ . . . 45_7

täisosa . *murdosa* . . . 45**333333333333334**. . . $_7 \approx$ . . . 46_7

5ndsüsteem: **p = 5** $a_i \in \{ 0\ 1\ \ 2\ \ 3\ 4 \}$

15ndsüsteem: **p = 15**

$a_i \in \{0\ 1\ 2\ 3\ 4\ 5\ 6\ \ 7\ \ 8\ 9\ A\ B\ C\ D\ E \}$

11ndsüsteem: **p = 11**

$a_i \in \{0\ 1\ 2\ 3\ 4\ \ 5\ \ 6\ 7\ 8\ 9\ A \}$

kui **p** = *paaritu arv* , siis ümardamisel võib vajalik olla **MITME mittemahtuva** / ärajääva järgu arvestamine.
(ehk seljuhul *ümardamine* võib vahel toimuda **mitme** järgu järgi)

. . . kõik arvusüsteemid kus alus **p** on **paaritu** arv — on siiski ebaolulised

ÜMARDAMISREEGEL 2ndSÜSTEEMI JAOKS

arvestades et **p = 2** on paarisarv :

numbrite väiksem pool *suurem pool*

2ndsüsteem: **p = 2** $a_i \in \{$ **0** **1** $\}$

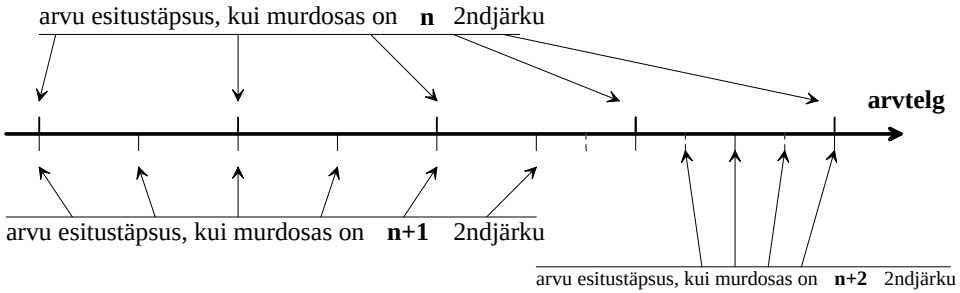
täisosa . *murdosa* . . . 0**0**0 . . . $_2 \approx$ *täisosa* . *murdosa* . . . 0**0** $_2$

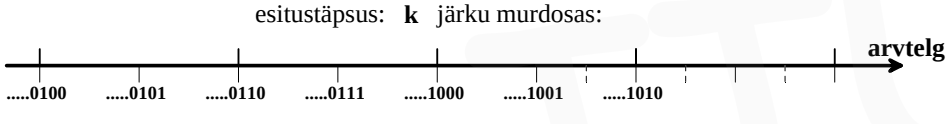
täisosa . *murdosa* . . . 0**0**1 . . . $_2 \approx$ *täisosa* . *murdosa* . . . 0**1** $_2$

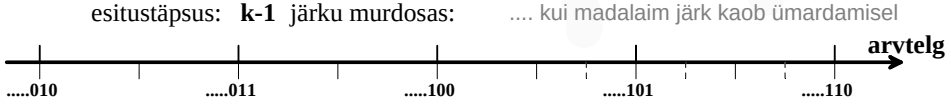
2ndarvu ümardamise *reegliks* sõnastub:

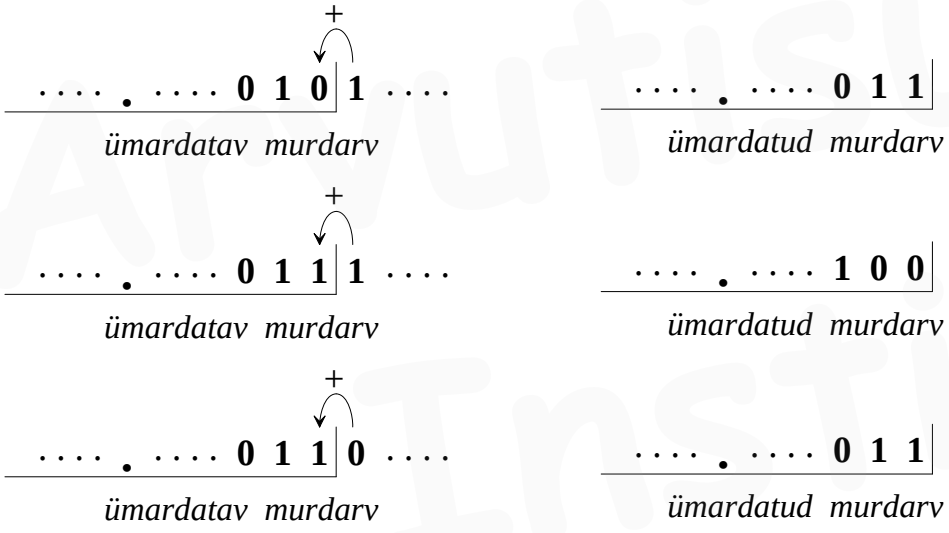
esimene formaadist väljajääv järguväärtus (0 või 1) **l i i d e t a k s e juurde** allesjääva arvuformaadi madalaimasse järku,
arvestades ka sellel liitmisel tekkivat ü l e k a n n e t .

Kahendarvude murdosa ümardamise olemus









ehk veelkord:

esimene formaadist väljajääv järguväärtus (0 või 1) **liidetakse juurde** allesjääva arvuformaadi madalaimasse järku, arvestades ka sellel liitmisel tekkivat ülekannet.

POSITIIVSETE 2ndarvude / 2ndkoodide NIHUTAMINE

Seni oleme vaatlemas ainult p o s i t i i v s e i d kahendarve.

2ndarvu ehk 2ndkoodi **nihutamisel** iga tema järguväärtus **0** / **1** nihkub ehk "astub" tema naaberjärku.

Nihutatav 2ndkood on kindla pikkusega ja asub teda hoidvas **registris**.

näiteks 8-järgulise 2ndkoodi (ehk 8-järgulise **registri**) korral :
[**0**][**1**][**1**][**1**][**0**][**1**][**0**][**1**]

järgud ehk *ühtede-nullide* asukohad (ehk kogu *register*) on "paigal";
ühed-nullid ehk **järguväärtused** nihkuvad / astuvad järgust (naaber)järku.
nihe **vasakule** tähendab nihet **kõrgematesse** naaberjärkudesse;
nihe **paremale** tähendab nihet **madalamatesse** naaberjärkudesse;

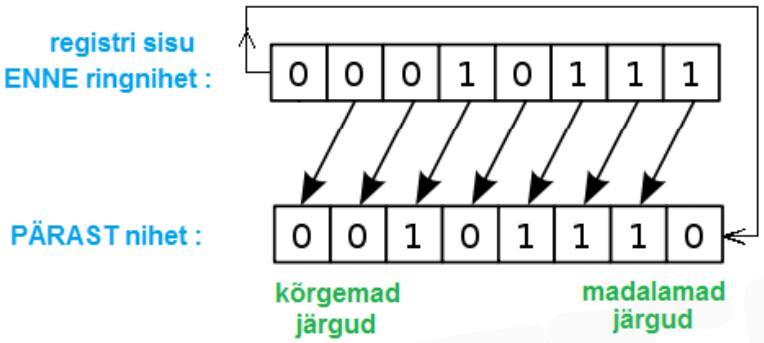
on olemas 2 tüüpi nihet:

aritmeetiline (ehk "tavaline") nihe ; *arithmetic shift*
ringnihe ; *circular shift*

Ringnihe

Ringnihe seletab juba oma nimega, millega on tegemist:
registrist väljanihkuv järguväärtus **0** / **1** (taas)siseneb registrisse selle "vabanevasse" järku (teises registri ääres) .
Ringnihe (*rotate shift*) on vähemtähtis nihke liik.

RINGnihe vasakule



Ringnihe p a r e m a l e liigutab / nihutab järke vastupidises suunas — kuid sama põhimõttega.

Aritmeetiline nihe

Aritmeetilise nihke korral lähevad "formaadist" ehk registrist väljanihkuvad järgud alati kaduma

ja

formaadi ehk 2ndarvu teises "servas", nihutamisel vabanevatele järkudele sisenevad formaati **n u l l i d** — tingimusel et oleme nihutamas p o s i t i i v s e t 2ndarvu : seni tegelemegi ainult *positiivsete* arvudega.

meenutame: **2nd**süsteemi *järgukaalud :*

. . . . 1024 512 256 128 64 32 16 8 4 2 1 . . .

ilmne, et

2ndkoodi nihutamisel (**1** järgu võrra) **vasakule** sattuvad tema koosseisus olevad kõik **1**-d 2 korda **suurema** kaaluga järkudesse, misjuhul arvu väärtus muutub 2 korda s u u r e m a k s

(toimub arvu **korrutamine** **2**ga)

2ndkoodi nihutamisel (**1** järgu võrra) **paremale** sattuvad tema koosseisus olevad kõik **1**-d 2 korda **väiksema** kaaluga järkudesse, misjuhul arvu väärtus muutub 2 korda v ä i k s e m a k s

(arv **jagatakse** 2ga ehk korrutatakse **0.5**-ga)

siit järeldub **2nd***koodi nihutamise olemus* :

nihutamisel **n** järgu võrra **vasakule** peab 2ndkoodi poolt esitatav väärtus muutuma $\mathbf{2^n}$ korda **suuremaks** (arv k o r r u t u b $\mathbf{2^n}$ -ga)

nihutamisel **n** järgu võrra **paremale** peab 2ndkoodi poolt esitatav väärtus muutuma $\mathbf{2^n}$ korda **väiksemaks** (arv j a g u b $\mathbf{2^n}$ -ga)

kuna selline nihutamine toob kaasa arvu väärtuse m u u t u m i s e : *korrutamise* / *jagamise* , siis nimetatakse sellist nihet *aritmeetiliseks*

(*arithmetic shift*)

P o s i t i i v s e t e arvude *aritmeetilist nihet* nimetatakse harvem ka *loogiliseks nihkeks (logical shift)* — misjuhul jäetakse termin *aritmeetiline nihe* seotuks just n e g a t i i v s e t e arvude nihutamisega : *aritmeetiline nihe* peab toetama ka *negatiivseid* 2ndarve !

Termini *loogiline nihe* järgi pole tegelikult vajadust : sobib nimetus *aritmeetiline nihe* — nii *positiivsete* kui ka *negatiivsete* 2ndarvude jaoks.

ülesanne:

Teisendada **2ndkujule** arvud

73.4

16.6

5.5

6.25

. . . . täpsusega **6 järku** 2ndarvude (ümardatud) murdosas.

Arvude teisendus 2ndsüsteemi üle 8ndsüsteemi: **10**nd → **8**nd → **2**nd

meenutame :

2ndsüsteemi **järgukaalud**: . . . 2^5 2^4 2^3 2^2 2^1 2^0 2^{-1} 2^{-2} 2^{-3} . . .

32 16 8 4 2 1 0.5 0.25 0.125

$\mathbf{0.4}_{10} \approx \mathbf{0.314}_8 = 0.011001100_2 \approx 0.011010_2$

$\mathbf{73.4}_{10} \approx 1001001.011010_2$

$\mathbf{16}_{10} = 10000_2$

$\mathbf{0.6}_{10} \approx \mathbf{0.463}_8 = 0.100110011_2 \approx 0.100110_2$

$\mathbf{16.6}_{10} \approx 10000.100110_2$

$\mathbf{5.5}_{10} = 101.1_2$

$\mathbf{6.25}_{10} = 110.01_2$

edasi :

just leitud **2nd**arve saame nüüd operandidena kasutada avaldise väljaarvutamisel :

ülesanne:

Eelmise ülesande arvudega a r v u t a d a **2ndkujul** :

[(73.4 - 16.6) : 5.5] × 6.25 = ...

. . . . täpsusega **6** järku 2ndarvude murdosas.

$\mathbf{73.4}_{10} \approx 1001001.011010_2$

$\mathbf{16.6}_{10} \approx 10000.100110_2$

kahendkujul lahutamine $\mathbf{73.4}_{10}$ **-** $\mathbf{16.6}_{10}$ = **?** : (*" vahetu" lahutamine*)

$\mathbf{73.4}_{10} = 1001001.011010_2$

$\mathbf{16.6}_{10} = 10000.100110_2$

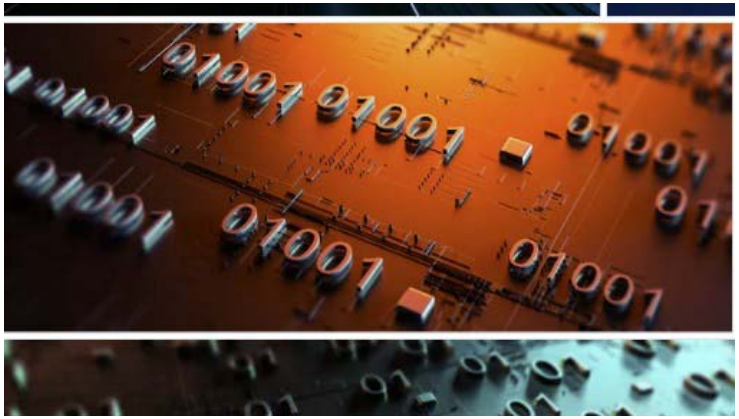
kahendkujul lahutamise tulemus :

$\mathbf{56.8}_{10} \approx 111000.110100_2 = 56.8125_{10}$ (selle **2nd**arvu täpne väärtus)

P. Ellervee loengutunnid nädalatel **1** . . . . . .

H. Lensen loengutunnid nädalatel **2 3 4 5 6 7**

M. Kruus loengutunnid nädalatel **8** . . . . . .



. . . eelmisest tunnist on pooleli avaldise **2ndkujul arvutamine** :

[(73.4 - 16.6) : 5.5] × 6.25 = ...

kus esimene tehe ehk l a h u t a m i n e **73.4 - 16.6** = **56.8** on tehtud.
Järgneb *j a g a m i n e kahendkujul :* **56.8 : 5.5** = **?** :

! j a g a j a suurendada **täisarvuks !**

. . . selleks on hariliku murru l a i e n d a m i n e siin abiks / vajalik.

meenutame : kahendarvu nihutamine v a s a k u l e ehk kõrgemate järkude suunas k o r r u t a b arvu **2**-ga.
nihutades jagamise mõlemat operandi 1 järgu võrra vasakule ehk "korrutades" mõlemat operandi **2**-ga :

$1\,1\,1\,0\,0\,0\,.\,1\,1\,0\,1\,0\,0_2$: $1\,0\,1\,.\,1_2$ =

. . . . ei muuda selline toiming j a g a t i s t !:
eelneva jagamise teostame muudetud, kuid samaväärsel kujul
(kus JAGAJA on täisarv) :

56.8 : 5.5 = 113.6 : 11 = ?

pärast **2nd**operandide nihet 1 järgu võrra v a s a k u l e :

$1\,1\,1\,0\,0\,0\,1\,.\,1\,0\,1\,0\,0_2$: $1\,0\,1\,1_2$ =

jagamise tulemus (täpsusega 6 kahendkohta murdosas) *:*

$1110001.10100_2 : 1011_2 \approx 1010.010101_2 = 10.328125_{10}$

korrutamine kahendkujul **10.328125_{10} × 6.25_{10}** :

$1\,0\,1\,0\,.\,0\,1\,0\,1\,0\,1 \times 1\,1\,0\,.\,0\,1$

kumb teguritest valida **korrutajaks** ja kumb jätta **korrutatavaks** ?

korrutajaks sobib (paremini / mugavamalt) see tegur, kus on v ä h e m järke **1** või mis on l ü h e m (vähem järke teguris).
Mugavam on paigutada operandid nii, et k o r r u t a j a oleks vasakul ja k o r r u t a t a v oleks paremal (vahetades nende eelmised asukohad) :

k o r r u t a j a × k o r r u t a t a v

$1\,1\,0\,.\,0\,1 \times 1\,0\,1\,0\,.\,0\,1\,0\,1\,0\,1$

---

kontrollides kalkulaatoriga :

(73.4 - 16.6) : 5.5] × 6.25 = 64.54545454 . . .

ülesanne:

Korrutada **2ndkujul** eelnevalt juba 2ndkujule leitud operandid :

korrutaja × *korrutatav* : **5.5_{10} × 6.25_{10} = 34.375_{10}**

ja samuti :

korrutaja × *korrutatav* : **6.25 × 5.5 = 34.375**

jälgi, kas saad mõlemal juhul tulemuseks sama **2nd**arvu ?

ülesanne:

Jagada **2ndkujul** **1110.101_2 : 11.01_2 = ... $_2$**

(jagub täpselt !)

kontrollida tulemust 10ndkujul (väärtuste võrdlemise teel)

! j a g a j a suurendada **täisarvuks !** (nihutades mõlemat operandi)

jagamine toimub operandidega :

111010.1_2 : 1101_2 =

jagamise tulemus (jagub täpselt !) :

1110.101_2 : 11.01_2 = 100.1_2 = 4.5_{10}

kontrollides kalkulaatoriga **10nd**kujul :

14.625_{10} : 3.25_{10} = 4.5_{10}

TÄIENDKOOD PÖÖRDKOOD NEGATIIVSETE ARVUDE ESITAMINE

♦ 0-ga algavat 2ndkoodi (**0**.....) nimetame **otsekood**iks.
Otsekood esitab alati *positiivset* väärtust, milleks on tema enda kui 2ndkoodi väärtus. (*"otsekood esitab iseennast"*)

NB! seni oleme tegelenud ainult **otsekoodidega** ehk positiivsete 2ndarvudega

seni esitasime positiivseid 2ndarve ka nii, et nad tohtisid alata numbriga **1** :

+ 17_{10} = 10001_2

nüüdsest edasi on rangelt tähtis, et **positiivne** arv peab algama **0**ga ! :

+ 17_{10} = 010001_2

+ 17_{10} = 0010001_2 (kui täisarv on esitatud 8-järgulise 2ndkoodina)

+ 17_{10} = 0000000000010001_2 (esitatud 16-järgulise 2ndkoodina)

♦ 1-ga algav 2ndkood (**1**.....) on **täiendkood** või **pöördkood**.

(kumb nendest ta tegelikult on, peab olema ette üteldud: koodile peale vaadates me ei tunne ära, kas ta on täiendkood või pöördkood)

täiendkood ja **pöörd**kood esitavad *negatiivset* väärtust.

täiendkood ja pöördkood on olemas ainult **2nd**süsteemis

Kõrgeimat järku nimetatakse *märgijärguks*, kuid tegelikult esitab ta samaaegselt nii väärtust kui ka märki — mitte a i n u l t märki**!**

+ 17$_{10}$ = **0**10001$_{2}$

! *tüüpiline esmane eksimus :*

Kuigi kõrgeimat järku nimetatakse **märgijärguks** ja märgijärk **1** on **negatiivse** arvu tunnuseks, siis negatiivset 2ndarvu ei saada positiivsest 2ndarvust tema märgijärgu 0 lihtsa asendamisega 1-ks:

kuigi:

+ 17$_{10}$ = **0**10001$_{2}$

siis:

— 17$_{10}$ ≠ **1**10001$_{2}$

Negatiivset arvu esitatakse *pöördkoodina* või *täiendkoodina*.

Nendest tuleb kumbki esitusviis valida — ja kui valik on tehtud siis teist koodi samas arvutiarhitektuuris enam kasutada ei saa / ei tohi.

♦ otsekoodist saame **pöördkoodi**, kui inverteerime kõik järgud vastupidiseks

+ 17$_{10}$ = **0**10001$_{2}$

Kui kasutada **-17** esitamiseks **pöördkoodi**, siis

— 17$_{10}$ = **1**01110$_{pk}$

täiendkoodi saamise 2 võimalust :

♦ otsekoodist saame **täiendkoodi**, kui liidame ta *pöördkoodile* **+ 1**

♦ otsekoodist saame **täiendkoodi**, kui kirjutame otsekoodi madalamad järgud ümber kuni esimese **1**-ni (kaasaarvatud) ja ülejäänud kõrgemad järgud inverteerime

mõlemad need toimingud annavad sama tulemuse

+ 17$_{10}$ = **0**10001$_{2}$

Kui kasutada **-17** esitamiseks **täiendkoodi**, siis

— 17$_{10}$ = **1**01111$_{tk}$

♦ täiendkoodi täiendkood on **otsekood**

♦ pöördkoodi pöördkood on **otsekood**

♦ Pöörates mingi 2ndkoodi täiendkoodi (või pöördkoodi) saame tema vastandarvu esitava 2ndkoodi.

meenutame:

otsekoodi ette tohib kirjutada **0**-le ilma otsekoodi väärtust sellega muutmata

järelikult :

♦ täiendkoodi ja pöördkoodi ette tohib kirjutada **1**-sid

(ilma et koodi poolt esitatav arvväärtus seeläbi muutuks)

— 17$_{10}$ = **1**01110$_{pk}$ = **1**1101110$_{pk}$ = **1**111111111101110$_{pk}$

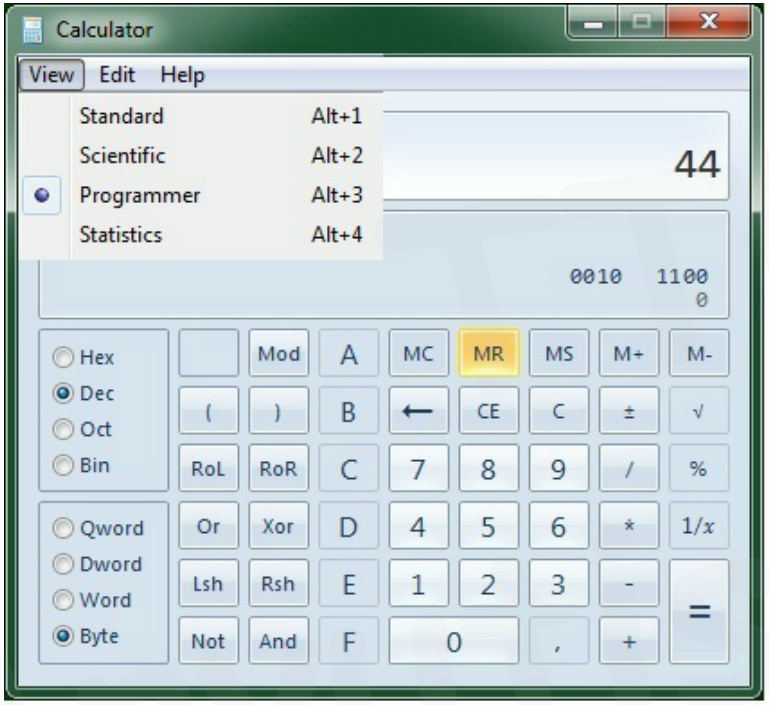
— 17$_{10}$ = **1**01111$_{tk}$

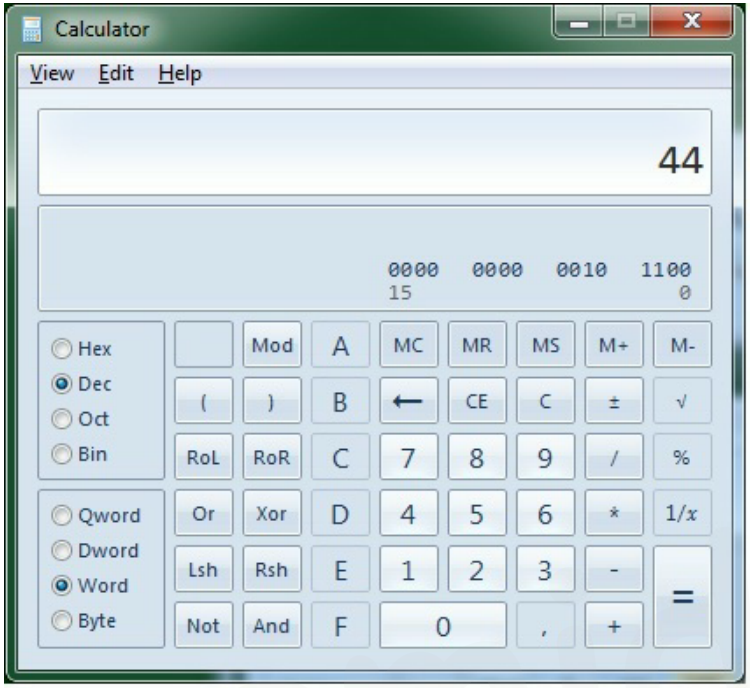
— 17$_{10}$ = **1**1101111$_{tk}$ (neg. täisarv esitatud 8-järgulisena)

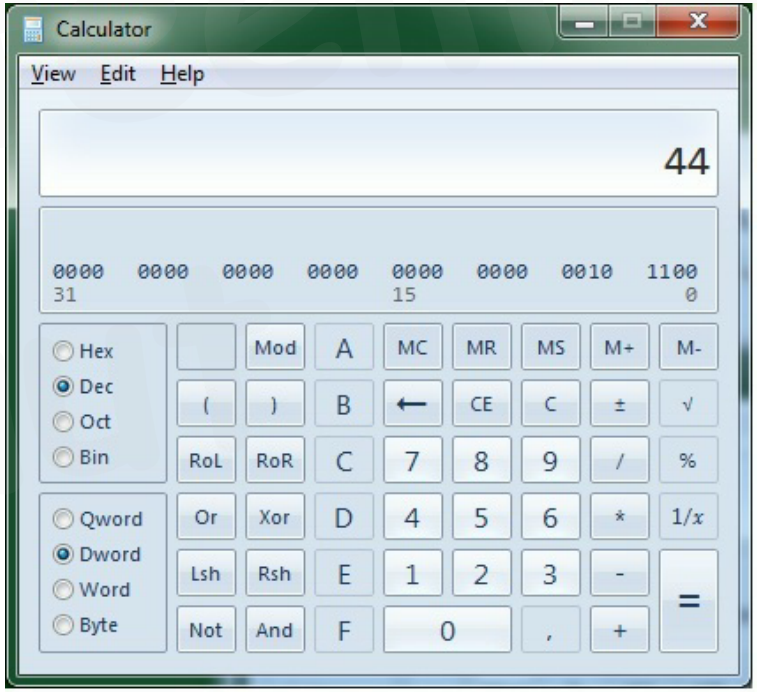
— 17$_{10}$ = **1**111111111101111$_{tk}$ (neg. täisarv esitatud 16-järgulisena)

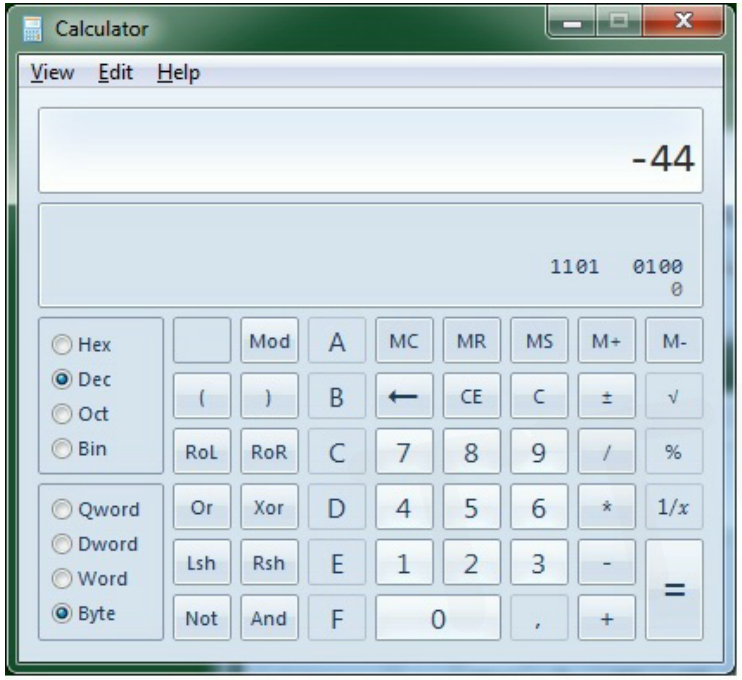
Kuigi negatiivsete arvude esitamiseks arvutis sobib valida nii **täiendkood** kui ka **pöördkood**, siis kõikides arvutiarhitektuurides on valitud selleks **täiendkood** kuna *täiendkoodi* aritmeetikareeglid on lihtsamad.

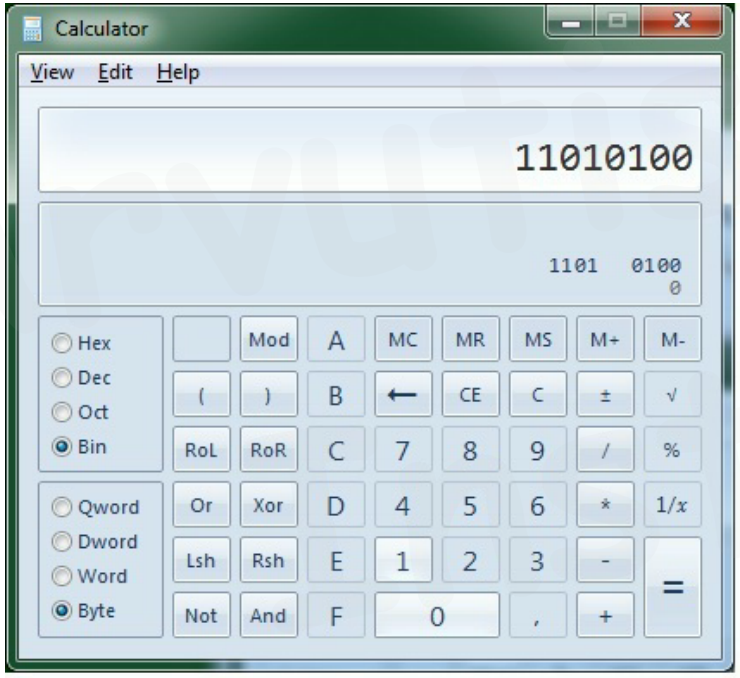
Windows Calculator abil saab vaadata arvutimälus salvestatud negatiivse täisarvu kahendkuju, kus ilmneb et arvuti hoiab negatiivseid arve just **täiendkoodis** :

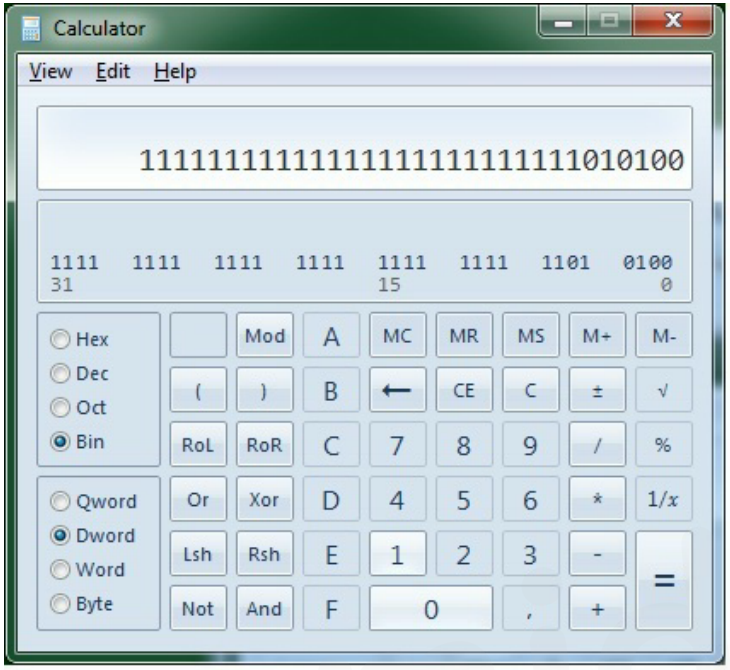












ARITMEETIKA TÄIENDKOODIDEGA

ülesanne:

PUHTMURDARVUD

Teisendada operandid üle **8nd**süsteemi **2nd**kujule.
Arvutada **2nd**kujul **liitmis**tehetena,
teostades l a h u t a m i s e asemel *negatiivse* esituse l i i t m i s e :

A — B = A + (- B)

kõik **negatiivsed** operandid esitada **2nd**kujul **täiendkoodis**
murdosa esitustäpsus: **7** kahendkohta (alati vaja ü m a r d a d a)

— 0.13_{10} + 0.52_{10} =

— 0.52_{10} + 0.13_{10} =

— 0.52_{10} — 0.13_{10} =

$\mathbf{0.13}_{10} \approx 0.102_{8} = 0.001000010_{2} \approx 0.0010001_{2}$

$\mathbf{0.52}_{10} \approx 0.412_{8} = 0.100001010_{2} \approx 0.1000011_{2}$

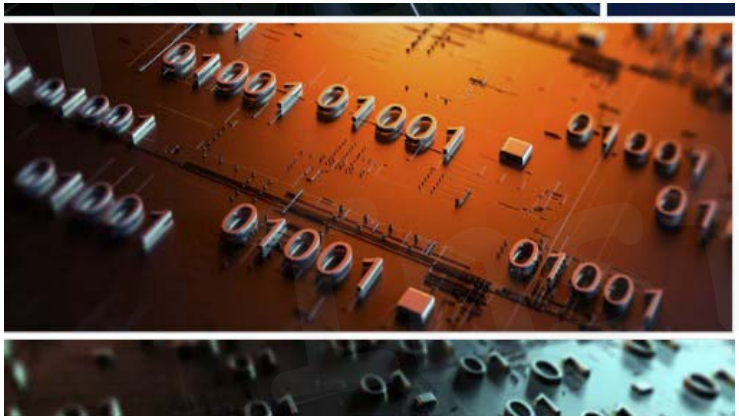
— $\mathbf{0.13}_{10}$ = — 0.0010001_{2} = 1.1101111_{2} võib ka : $\mathbf{1.1101111}_{tk}$

— $\mathbf{0.52}_{10}$ = — 0.1000011_{2} = 1.0111101_{2} võib ka : $\mathbf{1.0111101}_{tk}$

P. Ellervee loengutunnid nädalatel **1** . . . . . .

H. Lensen loengutunnid nädalatel **2 3 4 5 6 7**

M. Kruus loengutunnid nädalatel **8** . . . . . .



. . . eelmine tund leidsime **2nd**operandid :

$\mathbf{0.13}_{10} \approx 0.102_{8} = 0.001000010_{2} \approx 0.0010001_{2}$

$\mathbf{0.52}_{10} \approx 0.412_{8} = 0.100001010_{2} \approx 0.1000011_{2}$

— $\mathbf{0.13}_{10}$ = — 0.0010001_{2} = 1.1101111_{2} võib ka : $\mathbf{1.1101111}_{tk}$

— $\mathbf{0.52}_{10}$ = — 0.1000011_{2} = 1.0111101_{2} võib ka : $\mathbf{1.0111101}_{tk}$

täiendkoodis negatiivsete arvude liitmisel :
formaadist väljalevivat liitmise ülekannet **i g n o r e e r i t a k s e !**

kahendkujul liitmine (*negat* + *posit*) **- 0.13_{10} + 0.52_{10}** :

$\mathbf{-\ 0.13}_{10}$ = 1.1101111_{tk}

$\mathbf{+\ 0.52}_{10}$ = 0.1000011_{2}

kahendkujul liitmine (*negat* + *posit*) **- 0.52_{10} + 0.13_{10}** :

$\mathbf{-\ 0.52}_{10}$ = 1.0111101_{tk}

$\mathbf{+\ 0.13}_{10}$ = 0.0010001_{2}

kahendkujul liitmine (*negat* + *negat*) **— 0.52_{10} + (— $0.13)_{10}$** :

$$\begin{array}{rcl} \textbf{-\ 0.52}_{10} & = & 1.0111101_{tk} \\ \textbf{-\ 0.13}_{10} & = & 1.1101111_{tk} \\ \hline \end{array}$$

ülesanne:

TÄISARVUD

Arvutada **2nd**kujul **liitmis**tehetena, esitades negatiivsed operandid täiendkoodis :

$71_{10} - 40_{10} = \ldots$

$-71_{10} + 40_{10} = \ldots$

$-71_{10} - 40_{10} = \ldots$

$\textbf{71}_{10} = 107_8 = 001000111_2$ $\quad -\textbf{71}_{10} = 10111001_{tk}$

$\textbf{40}_{10} = 0101000_2$ $\quad -\textbf{40}_{10} = 1011000_{tk}$

kahendkujul liitmine (*posit* + *negat*) $\quad \textbf{+ 71}_{10} + (\textbf{- 40})_{10}$:

$$\begin{array}{rcl} \textbf{+ 71}_{10} & = & 0\,1\,0\,0\,0\,1\,1\,1_2 \\ \textbf{- 40}_{10} & = & 1\,1\,0\,1\,1\,0\,0\,0_{tk} \\ \hline \end{array}$$

kahendkujul liitmine (*negat* + *posit*) $\quad \textbf{- 71}_{10} + \textbf{40}_{10}$:

$$\begin{array}{rcl} \textbf{- 71}_{10} & = & 1\,1\,0\,1\,1\,1\,0\,0\,1_{tk} \\ \textbf{+ 40}_{10} & = & 0\,0\,0\,1\,0\,1\,0\,0\,0_2 \\ \hline \end{array}$$

kahendkujul liitmine (*negat* + *negat*) $\quad \textbf{- 71}_{10} + (\textbf{- 40})_{10}$:

$$\begin{array}{rcl} \textbf{- 71}_{10} & = & 1\,0\,1\,1\,1\,0\,0\,1_{tk} \\ \textbf{- 40}_{10} & = & 1\,1\,0\,1\,1\,0\,0\,0_{tk} \\ \hline \end{array}$$

$\textbf{+ 71}_{10} + (\textbf{- 40}_{10}) = 000011111_2 = +31_{10}$

$\textbf{- 71}_{10} + \textbf{40}_{10} = 111100001_{tk} = -000011111_2 = -31_{10}$

ülesanne:

MURDARVUD

Teha **2nd**kujul tehted, esitades negatiivsed arvud **täiendkoodis** :

$17.625_{10} + (-25.75_{10}) =$

$36.25_{10} + (-25.75_{10}) =$

ümardamist pole siin vaja kuna need operandid esituvad 2ndkujul **täpselt**

2ndsüsteemi j ä r g u k a a l u d :

32 16 8 4 2 1 **0.5** **0.25** **0.125**

17.625$_{10}$ = 010001.101$_2$ **36.25**$_{10}$ = 0100100.01$_2$

25.75$_{10}$ = 0011001.11$_2$

— **25.75**$_{10}$ = — 011001.11$_2$ = 100110.01$_2$

kahendkujul liitmine (*posit* + *negat*) **+ 17.625**$_{10}$ **+ (— 25.75)**$_{10}$:

+ 17.625$_{10}$ = 0 0 1 0 0 0 1 . 1 0 1 $_2$

— 25.75$_{10}$ = 1 1 0 0 1 1 0 . 0 1 0 $_{tk}$

kahendkujul liitmine (*posit* + *negat*) **+ 36.25**$_{10}$ **+ (— 25.75)**$_{10}$:

+ 36.25$_{10}$ = 0 1 0 0 1 0 0 . 0 1 $_2$

— 25.75$_{10}$ = 1 1 0 0 1 1 0 . 0 1 $_{tk}$

17.625$_{10}$ — 25.75$_{10}$ = 110111.111$_2$ = — 001000.001$_2$ = — 8.125$_{10}$

36.25$_{10}$ — 25.75$_{10}$ = 001010.100$_2$ = 10.5$_{10}$

iseseisvaks lahendamiseks :

Teha **2nd**kujul tehted, esitades negatiivsed arvud **täiendkoodis** :

13.57$_{10}$ + (— 28.26$_{10}$) =

28.26$_{10}$ + (— 13.57$_{10}$) =

. . . . murdosad teisendada läbi **8nd**süsteemi, täpsusega 8 järku **2nd**arvu murdosas

iseseisvaks lahendamiseks:

Teha **2nd**kujul liitmistehe, esitades negat. operandid **täiendkoodis** :

34$_{10}$ + (— 57$_{10}$) =

Teha **2nd**kujul liitmistehe, esitades negat. operandid **täiendkoodis** :

— 34$_{10}$ + 57$_{10}$ =

. . . . kuskil oli kirjutatud " **modifitseeritud täiendkood** " ?
Mis see on ?

modifitseeritud täiendkoodi (ja ka modifitseeritud **otse**koodi) jaoks kehtib lisanõue : märgijärke peab olema (vähemalt) 2 tk ehk m ä r g i j ä r k e peab olema näidatud / kasutuses / kirjutatud t o p e l t :

— 17$_{10}$ = ... **101111** $_{tk}$

— 17$_{10}$ = ... **1101111** $_{mtk}$

kui on kehtestatud nõue kasutada negatiivsete **2nd**arvude esitamiseks *modifitseeritud täiendkoodi* (mtk) siis positiivsete arvude (arvutuse käigus) tekkimisel tuleb ka need *modifitseeritud* kujul ehk *modif. otsekoodina* kus *otsekood* algab kah 2 märgijärguga: . . **00** . . . . . :

+ 17$_{10}$ = ... **0010001** $_2$

. . . . "märgijärke" tohib olla esitatud ka r o h k e m — mitte täpselt 2

ülesanne:

Leida järgnevate baidipikkuste 2ndarvude **väärtused**
(teades et negatiivsete väärtuste esitamiseks kasutatakse **täiendkoodi**) :

00000000_2 = ... $_{10}$

00000001_2 = ... $_{10}$

00000010_2 = ... $_{10}$

00000011_2 = ... $_{10}$

.

.

01111110_2 = ... $_{10}$

01111111_2 = ... $_{10}$

10000000_2 = ... $_{10}$

10000001_2 = ... $_{10}$

10000010_2 = ... $_{10}$

.

.

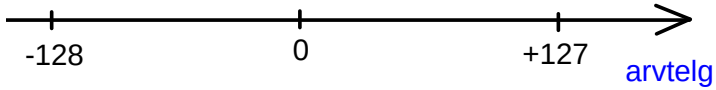
11111101_2 = ... $_{10}$

11111110_2 = ... $_{10}$

11111111_2 = ... $_{10}$

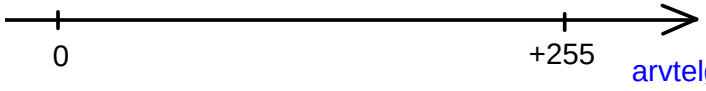
Järelda eelnevast, milline on baidipikkuse täisarvuformaadi **esitusdiapasoon** ?
ehk millise väärtusega on *minimaalseim* (negatiivne) ja *maksimaalseim* (positiivne) täisarv, mis "mahub" ühte baiti ehk on esitatav 8-järgulise 2ndarvuna ?

programmeerimiskeele **C** andmetüübi **`signed char`**
(ehk ühebaidise **märgiga** täisarvu) *esitusdiapasoon* :



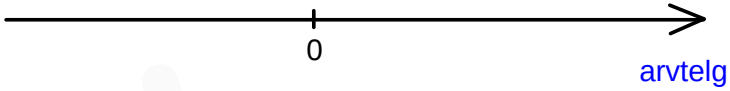
. . . . seda illustreeris meie inkrementeeriv 2ndarvujada siin ülesandes

programmeerimiskeele **C** andmetüübi **`unsigned char`**
(ehk ühebaidise **märgita** täisarvu) *esitusdiapasoon* :



ülesanne:

Leida (mittestandardse ehk) **10**-järgulise 2ndarvu väärtuste diapasoon (positiivseim ja negatiivseim täisarv (väärtus), arvestades et negat. arve esitatakse **täiendkoodis** ?



[kõige negatiivsem arv] ≤ **N** ≤ [kõige positiivsem arv]

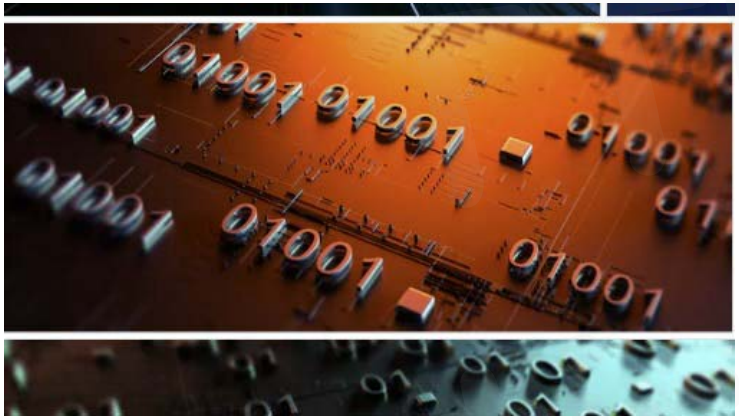
1000000000_{tk} ≤ **N** ≤ 0111111111_{tk}

-512_{10} ≤ **N** ≤ $+511_{10}$

P. Ellervee loengutunnid nädalatel **1** . . . . . .

H. Lensen loengutunnid nädalatel **2 3 4 5 6 7**

M. Kruus loengutunnid nädalatel **8** . . . . . .



ülesanne:

Korrutada **2nd**kujul täisarvud **11**$_{10}$ * (—**13**$_{10}$)
kus *korrutaja* on 11 ja *korrutatav* on —13
tuvasta, kas tulemuseks tekkis **11**$_{10}$ * (—**13**$_{10}$) = -143
ehk 2ndkujuline arv 143 **täiendkoodis**

leiame o p e r a n d i d :

\+ **11**$_{10}$ = ... 0 0 1 0 1 1 $_2$
\+ **13**$_{10}$ = ... 0 0 1 1 0 1 $_2$
— **13**$_{10}$ = ... 1 1 0 0 1 1 $_{tk}$

. . . . seega on korrutamise 2ndkujulisteks operandideks

korrutaja × *korrutatav*

. . . . 0 0 1 0 1 1 $_2$ × . . . . 1 1 0 0 1 1 $_{tk}$

. . . . see ruum / rida jääb liitmise ü l e k a n n e t e kirjutamiseks . . . .

. . . . 1 1 1 0 0 1 1 $_{tk}$

korrutis : . . . . 1 1 1 1 1 1 1 1 0 1 1 1 0 0 0 1 $_{tk}$ = -143$_{10}$

selle absoluutväärtus : . . . 0 0 0 0 1 0 0 0 1 1 1 1 $_2$ = +143$_{10}$

POSITIIVSETE / NEGATIIVSETE 2ndarvude **NIHUTAMINE**

meenutame:

Eespool juba kirjeldasime p o s i t i i v s e t e 2ndarvude ehk *otsekoodide* nihutamist.

2ndkoode oleme võtnud vaadelda 2 "liiki" : **otsekood** ja **täiendkood** kusjuures *otsekood* esitab **positiivset 2nd**arvu ja *täiendkood* esitab **negatiivset 2nd**arvu.

otsekood : **0** . . . . . .
täiendkood : **1** . . . . . .

Seega leidub meil **4** võimalikku kombinatsiooni **aritmeetilist** nihet :

otsekoodi nihe **vasakule** ;
otsekoodi nihe **paremale** ;
täiendkoodi nihe **vasakule** ;
täiendkoodi nihe **paremale** ;

. . . igal nihutamisel jääb registri "äärmine" järk alati "vabaks".
Kas sellesse vabanevasse järku "siseneb" **1** või **0** **?**

Milline järguväärtus (**0** / **1**) siseneb nihkel registri vabanevasse äärmisesse järku igal konkr. juhul nendest neljast **?**

[*otsekood*]
[*täiendkood*]

selle äratundmisel / järeldamisel vaja arvestada, et arvu **märk** (pos / neg) ei tohi muutuda (*aritmeetilisel* nihutamisel) : muutub ainult arvu **väärtus**
Seega :
negatiivne arv peab ka pärast nihutamist jääma *negatiivseks* ;
positiivne arv peab ka pärast nihutamist jääma *positiivseks*.

Leidub **4** võimalikku nihkejuhtumit :

>>> [*otsekood*] >>> (nihe **paremale**)
<<< [*otsekood*] <<< (nihe **vasakule**)

>>> [*täiendkood*] >>> (nihe **paremale**)
<<< [*täiendkood*] <<< (nihe **vasakule**)

sissenihkuvad järguväärtused (**0** või **1**) igal konkr. nihkejuhtumil :

0 >>> [*otsekood*] >>> (nihe **paremale**)
[*otsekood*] <<< **0** (nihe **vasakule**)

1 >>> [*täiendkood*] >>> (nihe **paremale**)
[*täiendkood*] <<< **0** (nihe **vasakule**)

ülesanne:

Arvuta aritmeetilise **nihutamise** kaasabil :

A / 4 + - (4 * **B**)

kui A = - 72 B = 13

moodle **Testides** #2 #3 tuleb kõik **ülekandelahtrid** t ä i t a õige ülekandega **0** / **1** : ülekandelahtreid ei tohi jätta t ü h j a k s (kuigi ülesande vana tekst võib seda "lubada")

BCD - koodid (**B**inary **C**oded **D**ecimal)

Arvude kodeerimisviis, kus iga 10ndnumber **0** .... **9** esitatakse 4-järgulise 2ndkoodiga (*poolbaidiga* ehk *tetraadiga*).
(. . . . oli sobiv põhimõte arvude salvestamiseks ja aritmeetikateheteks **varajastes arvutiarhitektuurides** . . . .)
Esimesed "*arvutid*" loodi just a r v u t a m i s e k s
(mitte "universaalseks andmetöötluseks" — nagu kaasajal arvuteid rakendatakse)

Esimeste arvutite tööks oli abistada arvutamisel, imiteerides "käsitsi arvutamist" :

1 2 7 . 9
+ **3 6 . 8**

Sellise arvutamise realiseerimiseks algelises digitaalarvutis — on / oli **BCD**-kood sobiv.

1971: **4**bitine esimene laiatarbe-mikroprotsessor **Intel 4004**
BCD oli sobiv arvude andmeformaat just 4bitise andmesõnaga (*4-bit word length*) arvutiarhitektuurides;
veelgi varasemad "suurarvutid" töötlesid samuti BCD-arve

uuemas tarkvaras ja riistvaras BCD-koode tavaliselt enam ei kasutata, kuigi kaasaegsed protsessorid endiselt omavad/toetavad BCD-käske ja BCD-arvuformaati

BCD-kodeerimisvõimalusi on olemas mitmeid erinevaid (rohkemgi kui 2) :

— **"loomulike kaaludega" BCD-kood** (järgukaaludega 8421)
— **"liiase kolmega" BCD-kood** (XS3)
"liiane 3":
tetraadi (*poolbait*) väärtus on esitatavast numbrist 3 võrra suurem :

decimal	BCD-koodis **8421** tetraad	BCD-koodis **8421(+3)** tetraad
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

aritmeetikareeglid on nendes erinevad.

Meie vaatleme ja teeme näiteid ainult "tavalise" BCD-koodi (8421) jaoks

BCD-koodide aritmeetika : **PARANDUSLIIKMED liitmisel**

parandusliige loomulike kaaludega **BCD-koodis** (8421) :
jooksvale tetraadisummale liidetakse juurde parandusliige **+ 6 = 0110**$_2$
kahel juhul:
— kui jooksev tetraadisumma on *keelatud kood* ehk **1010 . . . 1111**
— kui tetraadide liitmisel tekkis *ülekanne* kõrgemasse tetraadi

(mõlemad eelnevad tingimused ei saa samaaegselt esineda **!**)

Kui kumbki tingimus pole täidetud, siis parandusliiget ei rakendu.
Parandusliikmete liitmisel tekkivad tetraadidevahelised ülekanded.

☐ **negatiivse arvu leidmine** loomulike kaaludega **BCD-koodis** (8421) :

— *1. samm*: igale tetraadile liidetakse **+ 6 = 0110**$_2$
— *2. samm*: BCD arvu kahendkood tervikuna pööratakse *täiendkoodi*

BCD-koodide aritmeetika (liitmine)

ülesanne:

Arvutada loomulike kaaludega **BCD-koodis** (8421) :

$4492_{10} + 219_{10} =$

$4492_{10} - 219_{10} = 4492_{10} + (-219_{10}) =$

+ 4492 :	0	**0100**	**0100**	**1001**	**0010**
+ 0219 :	0	**0000**	**0010**	**0001**	**1001**

liitmise vahetulemus :

parandusliikmete rakendamine : ???? ???? ???? ????

tulemus: *tegelik summa* :

sama liitmise sammud slaidil veelkord:

4492 :	0	**0100**	**0100**	**1001**	**0010**
0219 :	0	**0000**	**0010**	**0001**	**1001**
	0	0100	0110	1010	1011
parandusliikmete rakendamine :		0000	0000	0110	0110
4492_{10} + 219_{10} :	0	**0100**	**0111**	**0001**	**0001**
		4	7	1	1

$4492_{10} + 219_{10} = 4711_{10}$

$4492_{10} - 219_{10} = 4492_{10} + (-219_{10}) =$

negatiivse arvu esitamine BCD koodis **?**

+ 0219 :	0	**0000**	**0010**	**0001**	**1001**
ettevalmistav samm : (+ 6) :	0	**0110**	**0110**	**0110**	**0110**

neg. esituse leidmise vahetulemus :

saadud vahetulemus pöörata **täiendkoodi** :

— 0219 :	1	**1001**	**0111**	**1000**	**0001**

neg. esituse ehk vastandarvu leidmise sammud slaidil veelkord:

0219 :	0	**0000**	**0010**	**0001**	**1001**
(ettevalmistava sammu +6 tulemus) :	0	0110	1000	0111	1111
pööratud **täiendkoodi** : — 0219 :	1	**1001**	**0111**	**1000**	**0001**

— 0219 :	1	**1001**	**0111**	**1000**	**0001**
4492 :	0	**0100**	**0100**	**1001**	**0010**

liitmise vahetulemus :

parandusliikmete rakendamine : ???? ???? ???? ????

tulemus: *tegelik summa* :

sama liitmise sammud slaidil veelkord:

0219 :	0	**0000**	**0010**	**0001**	**1001**
[vahetulemus pärast (+ 6)] :	0	0110	1000	0111	1111
— 0219 :	1	**1001**	**0111**	**1000**	**0001**
4492 :	0	**0100**	**0100**	**1001**	**0010**
	1	1101	1100	0001	0011
parandusliikmete rakendamine :	0	0110	0110	0110	0000
$4492_{10} - 219_{10}$:	0	**0100**	**0010**	**0111**	**0011**
		4	2	7	3

$4492_{10} - 219_{10} = 4273_{10}$

UJUPUNKTARVUD — UPA (ujukomaarvud UKA)
(*Floating Point Numbers*)

kõik senivaadeldud 2ndarvud on olnud **kinnispunktarvud** (KPA)

terminit **kinnispunktarv** läheb vaja alles siis, kui tuleb kasutusse termin **ujupunktarv** / ujukomaarv: *floating point number*

"*kinnispunktarv*" vs. "*ujupunktarv*"

nimetus **kinnispunktarv** rõhutab, et see arv p o l e *ujupunktarv*

kinnispunktarvud ehk "tavalised murdarvud" olid eelnevates ülesannetes :

$$\mathbf{6.25}_{10} = 0110.01_2$$
$$\mathbf{5.5}_{10} = 0101.1_2$$
$$\mathbf{-25.75}_{10} = 100110.01_2$$

eelnevad murdarvud esituvad *kinnispunktarvudena* "hästi" : pole probleemi

kinnispunktarvu p r o b l e e m ilmneb *numbrite* koguse kasvamisel :

50226717206034 . 803751170536 $_{10}$ =

= . . . . . ?????????????????? . ?????????????????? . . . . $_2$

kinnispunktarvud on *ühekomponendilised* ja "esitavad iseennast".

Kuni pole kasutusele tulnud nimetus **kinnispunktarv** (nagu seni meil polnudki) — seni võib kõiki "komaga arve" nimetada lihtsalt "murdarvudeks" (nagu nimetasimegi).

ujupunktarv koosneb 2 komponendist

UJUPUNKTARV (UPA) on arvu k a h e o s a l i n e esitus, mis koosneb kahest k i n n i s p u n k t a r v u s t :
mantissist **m** ja *astendajast* **p** *(exponent)*

kusjuures UPA **väärtus** arvutub nendest avaldisega :

$$mantissa \times 2^{exponent} \quad \text{ehk} \quad \mathbf{m} \times 2^{\mathbf{p}}$$

vaatleme võrdluseks: ujupunktarvu olemus **10nd**süsteemis :

10ndsüsteemis on UPA arvu kaheosaline esitus kujul : $\mathbf{m} \times 10^{\mathbf{p}}$

$$\mathbf{7250}_{10} = 7250 \times 10^{\mathbf{0}} =$$
$$= 7.250 \times 10^{\mathbf{3}} =$$
$$= 0.7250 \times 10^{\mathbf{4}}$$

või samaväärne esitus (tekstireziimis) arvutiekraanil: **7.250E+3**
(**10nd**eksponent)

ilmneb, et samale arvule leidub palju erinevaid ujupunkt-esituskujusid
ehk palju erinevaid paare **[mantiss astendaja]** [mantissa exponent]

. . . . või väikeste arvude korral (**n** << **1**)

$$\mathbf{0.000082663}_{10} = 8.2663 \times 10^{\mathbf{-5}} = 0.82663 \times 10^{\mathbf{-4}}$$

. . . mis esituks arvutiekraanil: **8.2663E-5** (**10nd**eksponent)

$$\mathbf{0.00000000000000047098}_{10} = \mathbf{4.7098E\text{-}16}$$
$$\mathbf{47098000000000000}_{10} = \mathbf{4.7098E+16}$$

näeme, et arvu esitus *ujupunktarvuna* ehk kahekomponendilisel kujul ("astendamise ja korrutamise kaudu") on seda õigustatum, mida **suurem** või **väiksem** on esitatav arv (**n** << **1** **n** >> **1**)
hiigelsuurte ja väga väikeste arvude esitamine "iseendana" ehk ühekomponendilistena ehk *kinnispunktarvudena* on ebamugav / mahukas.

ujupunktkuju **mantissi** esitamiseks piisab selle arvu **tüvenumbritest**

tagasi **2nd**süsteemi juurde :

mantissi järgukaalud: 1 . 1/2 1/4 1/8 1/16 1/32 1/64 . . . . . .
astendaja järgukaalud: 64 32 16 8 4 2 1

eelnäidatud järgukaaludest järeldub, et *ujupunktarvu* . . .
mantiss on PUHTMURDARV (+/- 0 . ???????. . . .)
astendaja on TÄISARV

näide:

+ **6.25**$_{10}$ = 0110.01$_2$

10ndarvu **6.25**$_{10}$ jaoks ehk 2ndkujulise *kinnispunktarvu* **0110.01**$_2$ jaoks :
kui *mantissiks* on **2nd**arv 0110.01 siis kaasnevaks *astendajaks* on 00000
kui *mantissiks* on **2nd**arv 011.001 siis kaasnevaks *astendajaks* on ...00001
kui *mantissiks* on **2nd**arv 01.1001 siis kaasnevaks *astendajaks* on ...00010
kui *mantissiks* on **2nd**arv 01100 .1 siis kaasnevaks *astendajaks* on ..11111
kui *mantissiks* on 011001 . 00.... siis kaasnevaks *astendajaks* on ...111110

kõik eelnevad paarid [*mantiss — astendaja*] on *ujupunktarvud*,
mis sobivad esitama *kinnispunktarvu* **0110.01**$_2$ ehk väärtust + **6.25**$_{10}$

ujupunktarvu tegelik väärtus saadakse **mantissi nihutamisel** astendaja poolt näidatud järkude võrra (ehk "*astendaja rakendamisega mantissile*")

astendaja rakendamisega mantissile saame **ujupunktarvust** jälle tagasi **kinnispunktarvu**

mida näitab astendaja p MÄRK (+ / -) ?
astendaja **p märk** (posit / negat) näitab, kumbas suunas on vaja mantissi **m** nihutada (UPA tegeliku väärtuse saamiseks);
astendaja **p väärtus** (täisarv) näitab, mitu järku on vaja mantissi **m** nihutada (UPA tegeliku väärtuse saamiseks);

teguriga × **2** ^{astendaja} korrutatakse mantissi
"0-llist kaugele" suureks või "0-llile lähedale" väikseks.
Selliselt korrutatud (nihutatud) mantiss esitabki kogu ujupunktarvu väärtust.

ujupunktarvu ESITUSTÄPSUS ja DIAPASOON

mantissi järkude arv ("mantissi pikkus") määrab ujupunktarvu **esitustäpsuse** (ehk tüvenumbrite arvu mida mantissi sisse mahub salvestada);

astendaja järkude arv määrab ujupunktarvu **esitusdiapasooni** ;

suurema astendaja korral saab mantissi korrutada "0-llist kaugemale" suureks või "0-llile lähemale" väikseks. Sellest tuleneb ka tinglik nimetus "*ujukoma*" ehk koma justnagu liigub / "ujub" mantissi järkude suhtes.

UPA väärtust väljendavast avaldisest
m × 2 ^{p}
järeldub, et kui *astendaja* **p** = **0**, siis mantissi väärtus ise ongi kogu UPA väärtuseks: **m** × 2^{0} = **m** × 1 = **m**

ujupunktarvu MÄRK
mantissi märk on samas kogu UPA märgiks.

! mistahes *murdarve* hoitakse arvutis ainult **ujupunktarvudena** **!**
arvutis ei hoita murdarve "ühekomponendilistena" ehk *kinnispunktarvudena*

murdarvude hoidmist/salvestamist *kinnispunktarvudena* pole realiseeritud üheski arvutiarhitektuuris — olukorras kus murdarvude esitus UPA-na oli omal ajal niikuinii vajalik, ei ole enam mõtet selle kõrvale luua murdarvude veel ühte alternatiivset esitust — 1-komponendilist vahetut esitust "iseendana" ehk *kinnispunktarvuna*

Ujupunktarvu hoidmisel arvutis paigutatakse ta mõlemad komponendid

(mantiss **m** astendaja **p**) kokku ühte füüsilisse registrisse.
Register jaguneb mõtteliselt kaheks loogiliseks osaks —
ühes on *mantiss* **m** (enamus reg. järkudest) ja teises on *astendaja* **p**

[. . . . . m a n t i s s . . . . .][. . astendaja . .]

või olenevalt konkr. arvutiarhitektuurist võib paiknemine registris olla ka vastupidi: **[. . astendaja . .][. . . . . m a n t i s s . . . . .]**

(üleskirjutatuna ujupunktarve esitades näitame *mantissi* ja *astendaja* eraldi)

kaasaegsetes protsessorites hoitakse ujupunktarve 10-nes baidis (mõlemale UPA komponendile kokku **80** kahendjärku).

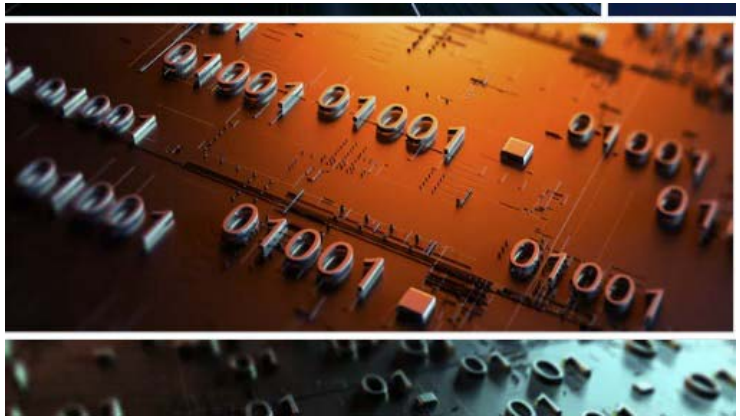
UPA olemasolu põhjus ja **eelis KPA ees** :
väheste arvujärkude abil saab esitada **väga suuri** arve ja **väga väikseid** 0-lilähedasi arve — kuid seejuures kaotame **esitustäpsuses** !

80-järguline (10-baidine) standardne UPAformaat võimaldab esitada/salvestada väärtust, mille 10ndkujus on **18 tüvenumbrit**

P. Ellervee loengutunnid nädalatel **1** . . . . . .

H. Lensen loengutunnid nädalatel **2 3 4 5 6 7**

M. Kruus loengutunnid nädalatel **8** . . . . . .



NORMALISEERITUD MANTISS

UPA **mantiss** salvestatakse alati **normaliseeritud** kujul.
Kui tehte tulemusel tekkib *mittenormaliseeritud* mantiss, siis ta kohe *normaliseeritakse*.

normaliseeritud mantissi **tunnus :**
normaliseeritud mantissi esimene murdosa järguväärtus peab erinema täisosa järguväärtustest

positiivne *normaliseeritud* **mantiss** saab olla vahemikus :

$$0.10000000\ldots_2 \leq \mathbf{m} \leq 0.11111111\ldots_2$$

modifitseeritud koodi korral (mida **Test #3** võib küsida) :

$$00.10000000\ldots_2 \leq \mathbf{m} \leq 00.11111111\ldots_2$$

järelikult asub **positiivse normaliseeritud** mantissi väärtus vahemikus:

$$0.5_{10} \leq \mathbf{m} < 1_{10}$$

negatiivne *normaliseeritud* **mantiss** saab olla vahemikus :

$$1.00000000\ldots_{tk} \leq \mathbf{m} \leq 1.01111111\ldots_{tk}$$

modifitseeritud koodi korral (mida **Test #3** võib küsida) :

11.00000000.... mtk ≤ **m** ≤ 11.01111111.... mtk

järelikult asub **negatiivse normaliseeritud** mantissi väärtus vahemikus:

$$-1_{10} \leq \mathbf{m} < -0.5_{10}$$

kahe järgu esitamine *mantissi* täisosas : 00**.**1..... 11**.**0.....
võimaldab avastada (arvutustehte tulemusel juhtunud) ü l e t ä i t u m i s t :
ületäitunud p o s i t i i v n e mantiss : **01.** .....
ületäitunud n e g a t i i v n e mantiss : **10.** .....

ü l e t ä i t u n u d mantiss võimaldab tema väärtuse / arvutustulemuse siiski edukalt väljalugeda — kuid ületäitunud mantissi ei tohi enam kasutada järgmise tehte operandiks.

kui mantissi täisosas on ainult üksainus järk, siis ületäitumine ei ole avastatav mantissile lihtsalt "peale vaadates" :
ületäitumisel tekkib korrektse **positiivse** asemel n e g a t i i v n e mantiss (mis näeb välja : **1.** .....)
ja
korrektse **negatiivse** asemel tekkib p o s i t i i v n e mantiss
(mis näeb välja : **0.** .....)
Sellist ületäitumist võib nimetada : *mitteavastatavaks.*

. . . . miks peaks m a n t i s s olema **normaliseeritud ?**

oletame, et *mantiss* tekkis sellisel kujul :

0 . 0 0 0 1 0 1 1 0 1 1 1 0 1 0 1 0 1 1

. . . . ja *murdosas* sailitab register nendest ainult **12** järku — misjuhul salvestatud (ja ümardatud !) mantiss oleks :

0 . 0 0 0 1 0 1 1 0 1 1 1 1

sellises mantissis oleksid 3 esimest murdosa järku registris "raisatud" ehk nad "ei tööta kasulikult" mantissi salvestamisel :

0 . 0 0 0 1 0 1 1 0 1 1 1 1

kui aga *mantiss* salvestatakse registrisse n o r m a l i s e e r i t u n a siis salvestub algne pikk mantiss :

0 . 0 0 0 1 0 1 1 0 1 1 1 0 1 0 1 0 1 1

kujul :

0 . 1 0 1 1 0 1 1 1 0 1 0 1 0 1 1

. . . . ehk alles jääb 3 järku rohkem kasulikku kahendinfot.

rohelised järgud saavad seljuhul kah päästetud "kadumaminekust".

Normaliseeritud mantissi korral "töötavad" registri kõik järgud "kasulikult" arvu salvestamisel : kõik *kahendjärgud* salvestavad infot.

mantissi NORMALISEERIMINE

Kui *mantiss* ei ole (mistahes põhjusel) *normaliseeritud,* siis mantissi on alati võimalik **normaliseerida**. Selleks tuleb :

1. mantissi *nihutada* vajalik arv järke paremale või vasakule, nii et *normaliseeritud mantissi tunnus* saaks täidetuks;

2. korrigeerida astendaja (ehk t ä i s a r v u) väärtust suuremaks või väiksemaks niimitme võrra, mitu järku mantissi nihutati *normaliseerimisel* ;

seejuures :
normaliseerimine ei muuda ("ei riku") UPA väärtust kuna *astendaja* korrigeerimine kompenseerib *mantissi* nihutamist.

ülesanne:

On antud UPA formaat alusel 2 ehk UPA väärtus arvutub :

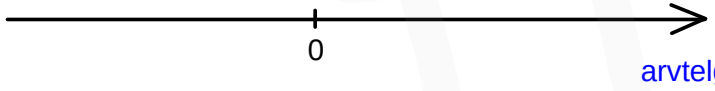
A = **m** × $2^{\mathbf{p}}$ = *mantiss* × $2^{\textit{astendaja}}$

mantissi täisosas k a k s järku ; *mantiss* kokku 10 kahendjärku ;
astendaja pikkus : 6 kahendjärku.

Kõik *negatiivsed* 2ndarvud : **täiendkoodis**.

Esitada absoluutväärtuselt **suurima** ja **nullilähedaseima posit**. ja **negat**. UPA *mantiss* ja *astendaja*.

arvestades *mantissi* **normaliseerituse** nõuet
on *maksimaalsed* ja *minimaalsed* etteantud formaadis ujupunktarvud:



pos $\mathbf{A}_{max}$ ≈ 00.11111111 × $2^{\mathbf{011111}}$ ≈ **1** × $2^{\mathbf{31}}$

pos $\mathbf{A}_{min}$ = 00.10000000 × $2^{\mathbf{100000}}$ = **0.5** × $2^{\mathbf{-32}}$

. . . . et m a n t i s s peaks olema ($\mathbf{A}_{min}$ jaoks) "kõige väiksem võimalik"? Aga miks ta ei ole seljuhul : **00.00000001 ?**

neg $\mathbf{A}_{min}$ ≈ 11.01111111 × $2^{\mathbf{100000}}$ ≈ **— 0.5** × $2^{\mathbf{-32}}$

neg $\mathbf{A}_{max}$ = 11.00000000 × $2^{\mathbf{011111}}$ = **— 1** × $2^{\mathbf{31}}$

küsimus:

Kuidas muutuks **UPA diapasoon** eelmises ülesandes, kui :
mantiss m oleks 10 järgu asemel **11** kahendjärku
ja
astendaja **p** oleks 6 järgu asemel **7** kahendjärku ?
....ehk mis juhtub kui **m** või **p** esitamiseks lisandub veel üks **2nd**järk ?

ülesanne:

Esitada normaliseeritud UPA-na **— 0.125**$_{10}$ ja **9.375**$_{10}$ negatiivsed: **täiendkoodiga**. Mantissi täisosas 2 järku.
Mantissi ja astendaja p i k k u s e d valida vabalt: vaja kasutada vähemalt niipalju järke, kui on vajalik väärtuse täpseks esitamiseks.

— 0.125$_{10}$ = — 00.001$_2$ = 11.111 $_{tk}$ (*kinnispunktarvuna*)

saadud *kinnispunktarvu* kuulutame UPA *normaliseerimata mantissiks* ja "komplekteerime" talle juurde astendaja 0 :

m = 11.111 $_{tk}$ **p** = 00000

lõpuks *normaliseerime* mantissi mille käigus korrigeerime ka astendajat

— 0.125$_{10}$: **m** = 11.000 $_{tk}$ **p** = 11101 $_{tk}$

9.375$_{10}$ = 001001.011$_2$ (*kinnispunktarvuna*)

m = 001001.011$_2$ **p** = 00000$_2$ (*normaliseerimata UPA-na*)

9.375$_{10}$: **m** = 00 .1001011$_2$ **p** = 00100 (*UPA-na*)

ülesanne:

Sama eespool olnud UPA formaat, k a h e järguga täisosas.
mantiss : kokku 10 kahendjärku (täisosas 2 järku + murdosa)
astendaja : 7 kahendjärku.
Teisendada UPA-kujule arvud / operandid **A** ja **B** :
A = — 0.3$_{10}$ **B** = 2.7$_{10}$

leiame operandid ujupunktarvudena (ehk leiame nende **m** ja **p**)
UPA võib leida kahel viisil :
1. eelnäidatud teel :
kinnispunktarv >>> normaliseerimata UPA >>> normaliseeritud UPA
või
2. eksponentkuju "matemaatilisel kaasabil", mis võimaldab kiiremini teadasaada *normaliseeritud UPA* **m** ja **p** väärtused :

$\mathbf{A} = -0.3 = -0.3 \times 2^{0} = -0.6 \times 2^{-1}$

normaliseeritud **m** absoluutväärtus on teatavasti väärtusvahemikus

$$0.5_{10} \leq \mathbf{m} \leq 1_{10}$$

ilmneb, et arvu **- 0.3** normaliseeritud mantiss on väärtusega **- 0.6** ja tema astendaja on väärtusega **-1**

$+\mathbf{0.6}_{10} \approx 0.463_{8} = 00.100110011_{2} \approx 00.10011010_{2}$ (ümardatud)

operand **A** ujupunktkuju :

$\mathbf{m}_A = 11.01100110_{tk}$ $\mathbf{p}_A = 1111111_{tk}$

analüüsides sama mõttekäiguga arvu B = 2.7 leiame tema *normaliseeritud mantissiks* ja *astendajaks* :

$$\mathbf{B} = 2.7 = \frac{2.7}{4} \times 2^{2}$$

$\mathbf{0.7}_{10} \approx 0.546_{8} = 00.101100110_{2}$

$\mathbf{2.7}_{10} \approx 0010.1011001_{2}$

(ümardatud KPA, mille loeme *normaliseerimata mantissiks* ja normaliseerime ta)

operand **B** ujupunktkuju :

$\mathbf{m}_B = \frac{2.7}{4} = 00.101011001 \approx 00.10101101$ $\mathbf{p}_B = 0000010$

ülesanne:

Sama eespool olnud UPA formaat, k a h e järguga täisosas.
mantiss: kokku 10 kahendjärku (täisosa + murdosa)
astendaja: 7 kahendjärku.
Arvutada eelpool leitud (samas formaadis) UPA-dega :

$\mathbf{A} = -0.3_{10}$ $\mathbf{B} = 2.7_{10}$

A + B
A — B
|A| × B (*positiivsed operandid*)
|A| : B (*positiivsed operandid*) — jagamist ei lahenda tunnis

ARITMEETIKATEHTED UJUPUNKTARVUDEGA

UPA aritmeetikatehete emulatsioon kinnispunktarvude aritmeetika kaudu

UPA liitmine
ja
lahutamine (ehk negatiivse liidetava liitmine)

tähistame *summa* : **C = A + B**
liidame A ja B v ä ä r t u s i esitavad avaldised :

$$\mathbf{C} = \mathbf{A} + \mathbf{B} = m_A \times 2^{p_A} + m_B \times 2^{p_B} =$$

ja teisendame saadud avaldist, tuues **kahe astme** sulgude ette:

$$= 2^{p_A} \left(\quad ? \quad \right) =$$

või

$$= 2^{p_B} \left(\quad ? \quad \right)$$

. . . meenutades põhikooli matemaatikat :
"võrdsete alustega astmete korrutamisel astendajad liidetakse" :
sellest tulenevalt saame sulgudesse **2** astendajaks sellise l a h u t a m i s e :

$$= 2^{p_A} \left(m_A + m_B \times 2^{p_B - p_A} \right) =$$

või

$$= 2^{p_B} \left(m_A \times 2^{p_A - p_B} + m_B \right)$$

. . . teguri sulgude ette toomise õigsuse kontrollimiseks võib sulud jälle lahti korrutada — misjuhul taastub esialgne avaldis.

eelnevast avaldisest ilmneb, mis saab olema (ehk kuidas tuleb leida)
summa **C astendaja** p_C ja summa **mantiss** m_C :

$$= 2^{p_A} \left(m_A + m_B \times 2^{p_B - p_A} \right) =$$

p_C ↓ , m_C

$$= 2^{p_B} \left(m_A \times 2^{p_A - p_B} + m_B \right)$$

. . . . ilmneb, et **matemaatiliselt** *on olemas k a k s võimalust summa komponentide* m_C p_C *valimiseks / leidmiseks ?*

kuigi mõlemad eelnevad arvutuseeskirjad tõesti sobivad
m a t e m a a t i l i s e l t ujupunktavude liitmise teostamiseks — siis tegelikkuses sobib nendest ainult ü k s — sest teine arvutusviis viib (juba enne liitmistehet) mantissi <u>ületäitumisele</u> : see variant kahest mis nihutab mantissi v a s a k u l e (ehk korrutab ta s u u r e m a k s) — annab otsekohe mantissi ü l e t ä i t u m i s e .

operandide kahest astendajast p_A p_B ainult **suurem** sobib *summa astendajaks* p_C ;
siin sobib *summa astendajaks* : p_B ehk $p_C = p_B$
väiksema astendaja (siin: p_A) valik *summa astendajaks* p_C annaks summa mantissi ületäitumise, kuna sulgudes olevat ühte mantissi tuleks seljuhul "korrutada suuremaks"

Meie praeguste UPA-operandide A ja B a s t e n d a j a t e korral :

$m_A = 11.01100110_{tk}$ $p_A = 1111111_{tk}$

$m_B = 00.10101101_2$ $p_B = 0000010_2$

. . . osutub kahest arvutusvõimalusest **C = A + B** jaoks sobivaks ainult see :

$$C = A + B = 2^{p_B} \left(m_A \times 2^{p_A - p_B} + m_B \right)$$

. . . asendades siia meie UPA-operandide astendajate väärtused :

$$C = A + B = 2^{2} \left(m_A \times 2^{-1-2} + m_B \right)$$

$m_A \times 2^{-3}$ on tegelikult j a g a m i n e $2^3 = 8$-ga
ehk on *mantissi* m_A **nihe paremale** 3 järku :

$\mathbf{m_A}$ enne nihutamist : $\mathbf{m_A} = 11.01100110_{tk}$

$\mathbf{m_A} \times 2^{-3} = \mathbf{11.11101100}110_{tk}$ (*nihutatud*)

$\mathbf{m_A} \times 2^{-3} = 11.11101100_{tk}$ (*ümardamata !*)

$\mathbf{m_B} = 00.10101101_2$

. . . saime summaks oleva UPA **C** *mantiss* ja *astendaja* :

$\mathbf{m_C} = 00.10011001$ $\mathbf{p_C} = 0000010$

arvutame 10ndkujul, mis on A + B *ootuspärane tulemus* ? :

A + B = — 0.3 + 2.7 = **+ 2.4** = **C**

. . . rakendades *astendaja mantissile* (ehk nihutades *mantissi*) , saame UPA-st jälle tagasi *kinnispunktarvu* (KPA) :

A + B = $0010.011001_2 \approx 2.4_{10}$

(sellel liitmisel hästi ei ilmnenud *ümardamata* operandi mõju arvutusele)

UPA lahutamine

tähistame *vahe* : **C = A — B**

$$\mathbf{C = A - B} = \mathbf{m_A} \times 2^{\mathbf{p_A}} - \mathbf{m_B} \times 2^{\mathbf{p_B}} =$$

$$= 2^{\mathbf{p_A}} \left(\mathbf{m_A} - \mathbf{m_B} \times 2^{\mathbf{p_B} - \mathbf{p_A}} \right) =$$

$\mathbf{p_C}$ $\mathbf{m_C}$

$$= 2^{\mathbf{p_B}} \left(\mathbf{m_A} \times 2^{\mathbf{p_A} - \mathbf{p_B}} - \mathbf{m_B} \right)$$

ilmneb, et UPA lahutamise ainus erinevus (liitmise suhtes) on :

+ $\mathbf{m_B}$ asemel osaleb resultaadi mantissi arvutamisel **- $\mathbf{m_B}$**

.... ehk **A - B** *mantissi* leidmisel (sulgudes) tuleb l i i t a **$\mathbf{m_B}$** *täiendkood*

A - B = C resultaadi **astendaja** on endiselt $\mathbf{p_C} = \mathbf{p_B}$

meenutame B mantissi : $\mathbf{m_B} = 00.10101101_2$

$\mathbf{m_A} \times 2^{-3} = 11.11101100_2$ (*ümardamata*)

$-\mathbf{m_B} = 11.01010011_{tk}$

(ü m a r d a m a t a operandidega) liitmisel saame

$\mathbf{m_C} = 11.00111111_{tk} = -00.11000001_2 \approx -0.75_{10}$ (**pole** täpselt)

meenutame operandide väärtusi : **A** = -0.3_{10} **B** = 2.7_{10}

A - B = — 0.3 — 2.7 = **— 3.0** = **C**

— 3.0 omab UPA-na *mantissi* täpselt -0.75_{10} sest **p = 2** korral :

$-0.75_{10} \times 2^2 = -3$

. . . . aga saime mantissi -00.11000001_2 mis veidi e r i n e b täpsest väärtusest : -0.75_{10}

ümardamise kasuliku mõju näide
sama tehe veelkord, kuid nüüd juba korrektselt ü m a r d a t u d operandiga :

$\mathbf{m}_A \times 2^{-3} = 11.11101101_2$ (*ümardatud !*)
$-\ \mathbf{m}_B = 11.01010011_{tk}$

(korrektselt ümardatud operandidega) lahutamisel saame
$\mathbf{m}_C = 11.01000000_{tk} = 00.11000000_2 = -0.75_{10}$ (täpselt !)

$\mathbf{A - B} = -0.75_{10} \times 2^2 = \mathbf{-3}$ (UPA väärtus **m** ja **p** kaudu)

. . . . ü m a r d a t u d operandi(de) korral saime t ä p s e ootuspärase arvutustulemuse **!**

korrutamine

$|\mathbf{A}| \times \mathbf{B}$ (*positiivsed operandid*)

tähistame *korrutise* : $\mathbf{C} = \mathbf{A} \times \mathbf{B}$
korrutame A ja B väärtusi esitavad eksponent-avaldised :

$$C = A \times B = m_A \times 2^{p_A} \times m_B \times 2^{p_B} =$$

see avaldis osutub *nelja teguri* korrutiseks . . . . järjestame tegurid ümber :

$$= m_A \times m_B \times 2^{p_A} \times 2^{p_B} =$$

. . . meenutades põhikooli matemaatikat :
"võrdsete alustega astmete korrutamisel astendajad liidetakse" :

$$= m_A \times m_B \times 2^{p_A + p_B}$$

korrutise *mantiss* ja *astendaja* arvutuvad seega operandide mantissidest ja astendajatest nii:

$$= \underbrace{m_A \times m_B}_{m_C} \times 2^{\overbrace{p_A + p_B}^{p_C}}$$

seega:
korrutise *mantiss* arvutatakse: vaja k o r r u t a d a operandide *mantissid* ;
korrutise *astendaja* arvutatakse: vaja l i i t a operandide *astendajad* ;

. . . osutub, et UPA **korrutamine** *on lihtsam kui nende* **liitmine** . . .

meie ülesandes seega vaja p o s i t i i v s e d mantissid korrutada
korrutise *mantissi* $\mathbf{m}_C$ saamiseks: $\mathbf{m}_C = |\mathbf{m}_A| \times \mathbf{m}_B$

korrutise *astendaja* : $\mathbf{p}_C = \mathbf{p}_A + \mathbf{p}_B$ $\mathbf{p}_C = \mathbf{-1 + 2 = +1}$
$\mathbf{p}_C = 0000001_2$

operandide A ja B *mantissid* :
$\mathbf{m}_A = 11.01100110_{tk}$ $\mathbf{m}_B = 00.10101101_2$
$|\mathbf{m}_A| = 00.10011010_2$

k o r r u t a j a × **k o r r u t a t a v**
$00.10011010_2 \times 00.10101101_2$

. . . oot ! — siin on ju mõtet enne korrutamist **täpsust** *vähemaks jätta operandidel* ? . . . sest ka k o r r u t i s e ehk $\mathbf{m}_C$ täpsus peab olema seesama mis on ka operandidel (mitte suurem)
kui me **e i** jätaks tegurite täpsust / pikkust vähemaks, siis saaksime k o r r u t i s e *murdossa* **8 + 8 = 16** järku . . . . mis oleks "asjatult palju" : madalamad 8 järku peaks "äralõikama" (koos ümardamisega)

$$00.10011 \times 00.101011$$

korrutamise tulemus ehk korrutise C *mantiss* : $\mathbf{m}_C = 00.01100110001$

meenutame : C astendaja on juba teada : $\mathbf{p}_C = 0000001_2$

korrutis C UPA-na oleks seega juba olemas ?

. . mis on selle mantissi $\mathbf{m}_C$ *juures "halvasti" ?*

korrutis *normaliseeritud* UPA-na :

$\mathbf{m}_C = 00.11001100$ $\mathbf{p}_C = 0000000$

$|\mathbf{A}| \times \mathbf{B} = 0.3 \times 2.7 = 0.81$

. . . püüame hinnata saadud UPA-tulemust — kas võib olla õige ?

kuna $\mathbf{p}_C = 0$ siis *mantiss* ise ongi kogu UPA väärtuseks

jagamine

tähistame *jagatise* : $\mathbf{C} = \mathbf{A} : \mathbf{B}$

jagame A ja B väärtusi esitavad eksponent-avaldised :

$$\mathbf{C} = \mathbf{A} : \mathbf{B} = m_A \times 2^{p_A} : m_B \times 2^{p_B} =$$

$$= \frac{m_A \times 2^{p_A}}{m_B \times 2^{p_B}} =$$

$$= \frac{m_A}{m_B} \times \frac{2^{p_A}}{2^{p_B}} =$$

. . . võrdsete alustega astmete **jagamisel** *astendajad* **lahutatakse** :

$$= \frac{m_A}{m_B} \times 2^{p_A - p_B}$$

$$m_C \quad \frac{m_A}{m_B} \times 2^{p_A - p_B} \quad p_C$$

$\mathbf{m}_A = 11.01100110_{tk}$

$|\mathbf{m}_A| = 00.10011010_2$

$\mathbf{m}_B = 00.10101101_2$

(positiivsete) mantisside **jagamine** (jagatise mantissi $\mathbf{m}_C$ saamiseks) :

$|\mathbf{m}_A|$: $\mathbf{m}_B$ = 00.10011010 : 00.10101101

$\mathbf{m}_C$ = $00.11100011_{\ 2}$

jagatise astendaja $\mathbf{p}_C$:

$\mathbf{p}_C$ = $1111111_{\ tk}$ — $0000010_{\ 2}$ = $\mathbf{-1}_{10}$ — $\mathbf{2}_{10}$ = $\mathbf{-3}_{10}$

$\mathbf{p}_C$ = $1111101_{\ tk}$

jagatis **kinnispunktarvuna** :

C = A : B = $0.00011100011_{\ 2}$ = **0.1111111111**...._{10}

kaasaegsetes arvutites ei kasutata enam sellist *emulatsiooni* UPA-dega arvutamisel vaid UPA aritmeetikatehted teostatakse **matemaatika kaasprotsessori** poolt *riistvaras* :

suured *loogikaskeemid* (protsessoris) arvutavad tehte tulemuseks oleva UPA kõik järgud

JAGAMISALGORITMID

Jagamine **jäägi taastamisega**

Jagamine **jäägi taastamiseta**

peab kehtima: jagatav < jagaja.

Jagatis on <u>kahendpuhtmurdarv</u>: **0** < $jagatis_2$ < **1**

Leida jagatise **9 : 13** kahendkuju, jagades **jäägi taastamisega**.

ja järgnevalt ka :

Leida jagatise **9 : 13** kahendkuju, jagades **jäägi taastamiseta**.

need jagamisalgoritmid annavad tulemuse / jagatise **2nd**kujul

ehk tegemist on 2ndjagamise algoritmidega

kuid algoritmi samme saab rakendada ka **10nd**kujul andmetele

(kus nii operandid kui ka vahetulemused on **10nd**kujul)

9 : 13 = **0 . 1 0 1 1 0 0 0 1 0 0 1 1 1** . . . $_2$

Jäägi TAASTAMISETA jagamisalgoritm sisaldab vähem samme ja seega ta teostab jagamise "kiiremini".

ülesanne:

Teisenda **2nd**arv 110110101_2 **10nd**arvuks, saades otsitava arvu 10ndnumbrid *jagamise jääkidena* , uue alusega ehk 10-ga (**1010_2**) jagamise teel **2nd**süsteemis

meenutame:

täisarvu teisendus ühest süsteemis teise saab toimuda <u>uue arvusüsteemi alusega jagamise teel</u> , kusjuures uue saadava arvu järgud ehk numbrid ilmnevad jagamise **jääkidena**.

Tavaliselt ei ole mõtet teisendada **2n**darvu **10nd**kujuks (ehk *väärtuseks*) niimoodi *jagamise* teel, kuna leidub palju kiirem/mugavam tee — kuid näitame, et ka niimoodi <u>2ndsüsteemis jagamise</u> teel on teisendus **10nd**kujule võimalik

ALGORITMIDE GRAAFSKEEMID (AGS)

ülesanne:

Koostada algoritm *Algoritmi Graafskeemina* (**AGS**) , mis **korrutab** registris A (**RgA**) sisalduva arvu **23**-ga,
kasutades **nihutamist** ja **summeerimist**.
Tulemus (korrutis) võib tekkida mujale registrisse, mitte samasse **RgA**
Koostada ka seda algoritmi realiseeriva
operatsioonseadme struktuurskeem
koos *juhtsignaalidega* / *juhtkäskudega* y_i .
Riistvara püüda kasutada mitte rohkem kui minimaalselt on vajalik.

. . . . esitame 23 "kahe astmete" summana — see on alati võimalik :

RgA × 23 = RgA × (16 + 4 + 2 + 1) =

. . . . korrutame sulud lahti :

= RgA × 16 + RgA × 4 + RgA × 2 + RgA =

. . . . arvestades tegevuste järjekorda tulevases algoritmis, soovime siin avaldises järjestada liidetavad ümber vastupidisesse järjekorda :

= RgA + RgA × 2 + RgA × 4 + RgA × 16 = RgA × 23

selline avaldis ongi arvutatav **nihutamisega** ja **summeerimisega**

. . . AGS on siin slaidil puudu ja OPseadme struktuurskeem kah puudu . . .

ülesanne:

Koostada algoritm (AGS-ina), mis teisendaks loomulike kaaludega (8421) **BCD-koodis** positiivse arvu tavaliseks **2ndarvuks**.

näide: **0001 0011 0100** → 10000110_2

pöörame tähelepanu **10nd***arvu üksikutele järkudele :*

$\mathbf{N = (a_n \ a_{n-1} \ . \ . \ . \ . \ a_2 \ a_1 \ a_0)_{10}}$

selle **10nd**arvu **N** väärtus on avaldatav / arvutatav avaldisena :

$\mathbf{N = [(a_n \times 10 + a_{n-1}) \times 10 + a_{n-2}] \times 10 + ... + a_0 =}$

$\mathbf{= [(a_n \times (8+2) + a_{n-1}) \times (8+2) + a_{n-2}] \times (8+2) + ... + a_0 =}$

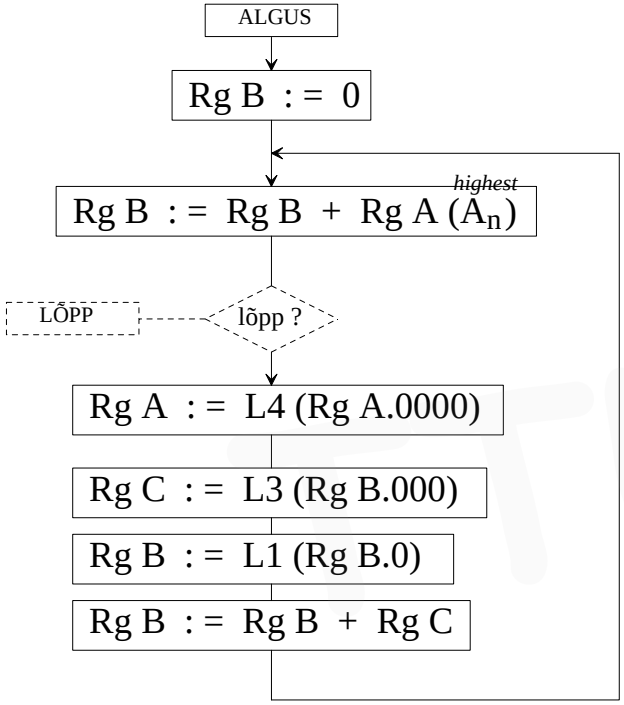
kui selline avaldis....

a) kujundada **algoritmiks** (AGS näiteks) ;
b) **luua** seda algoritmi teostav **digitaalseade** ;
c) seade "tööle panna" ehk lasta loodud seadmel see algoritm korra läbida ;

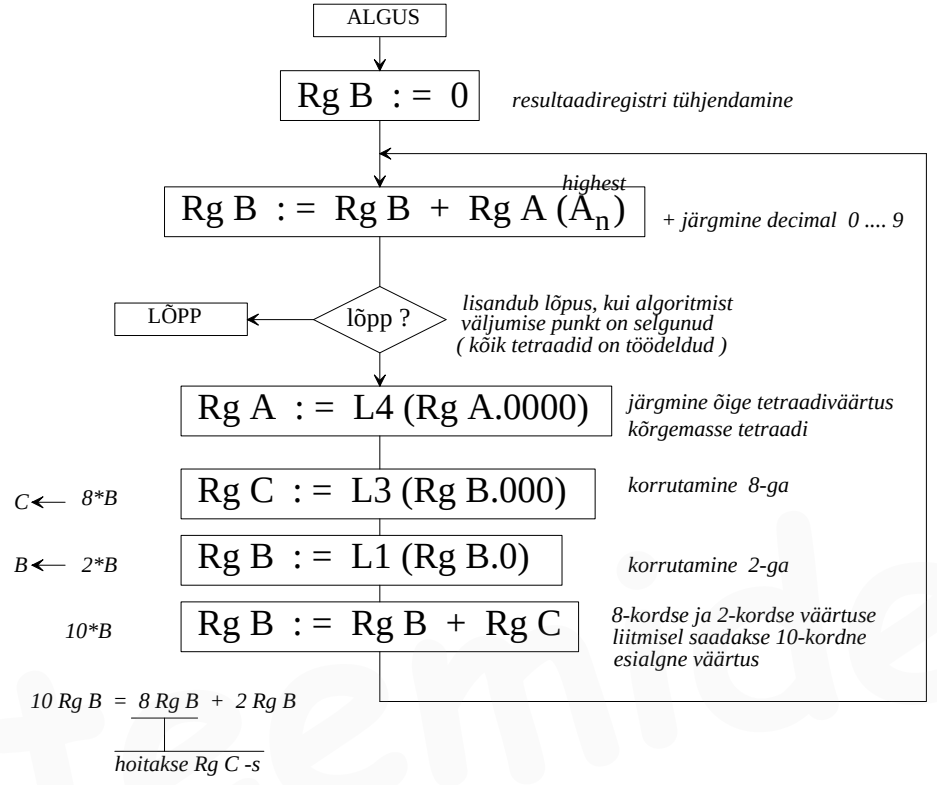
. . . . siis tema töötulemus ongi vajalik resultaat **2nd**kujul seadme mingis registris

algoritmi koostamise käigus selgub, et vaja on **3 registrit** :

Rg A : algne **BCD** (teisendatav)
Rg B : **binary** (tulemus)
Rg C : *abiregister*



eelnev **AGS** korratud koos sammude selgitustega / kommentaaridega :



ülesanne:

Koostada ja esitada AGS-ina vastupidise teisenduse algoritm:
Teisendada **2ndarv** liiase 3-ga **BCD koodi** :
binary → **BCD** 8421(+3)

näide: $\mathbf{10000110}_2$ → **0100 0110 0111**

koostatava algoritmi põhimõte :

arvu **10nd**kuju üksikud numbrid saab **2nd**arvust genereerida tema (korduva) **jagamise teel 10**ga, kus nad tekkivad **jagamise jääkidena** :

näide arvu **134** *teisenduse jaoks:*

134 : 10 = 13 (jääk **4**)
13 : 10 = 1 (jääk **3**)
1 : 10 = 0 (jääk **1**)

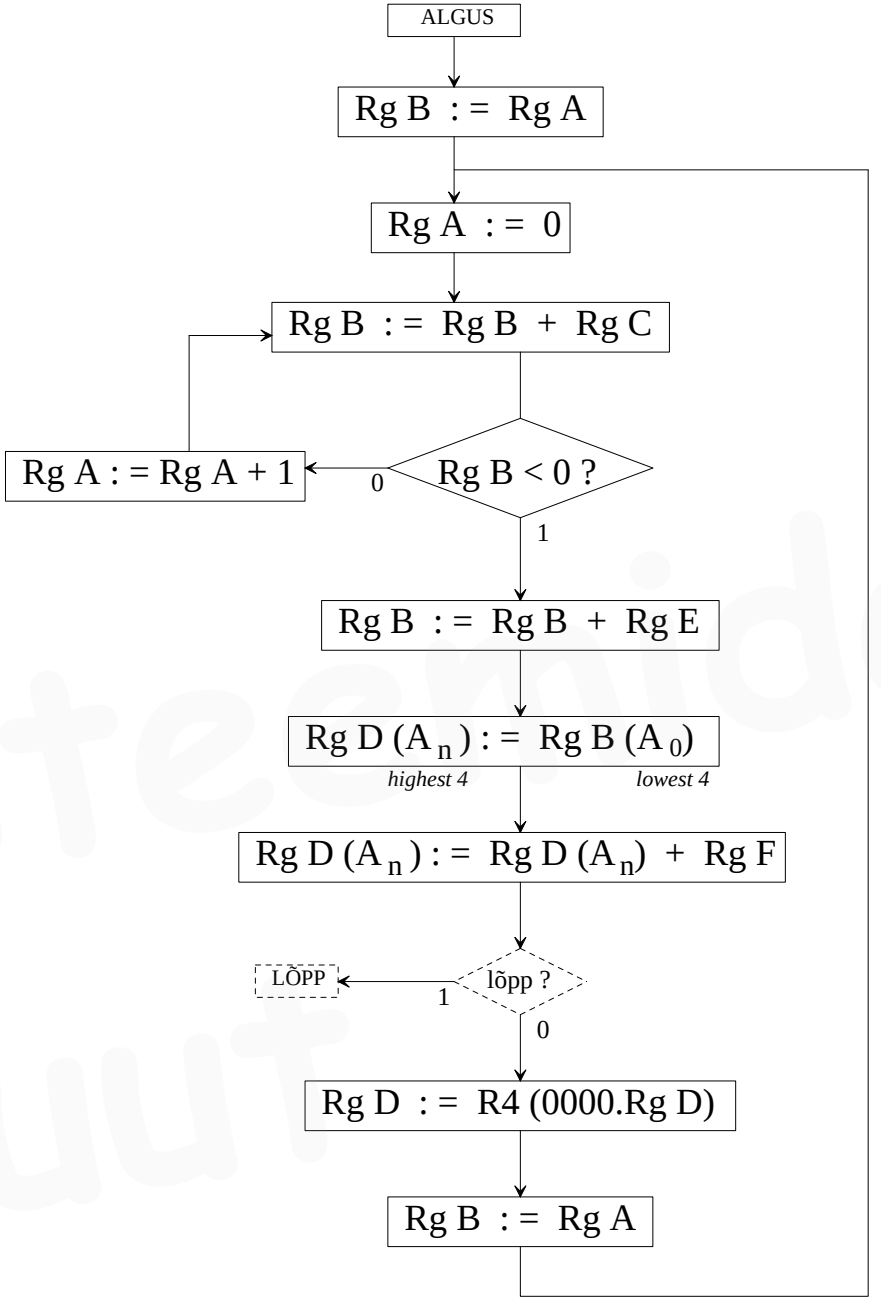
? kuidas algoritm jagab **10**ga ? — oletades, et meil pole kasutada spetsiaalset **jagajat** eraldi moodulina

10-ga **jagada** saab: **lahutades** korduvalt 10-t ja loendades lahutamise kordi, kuni lahutamise tulemus saab < 0

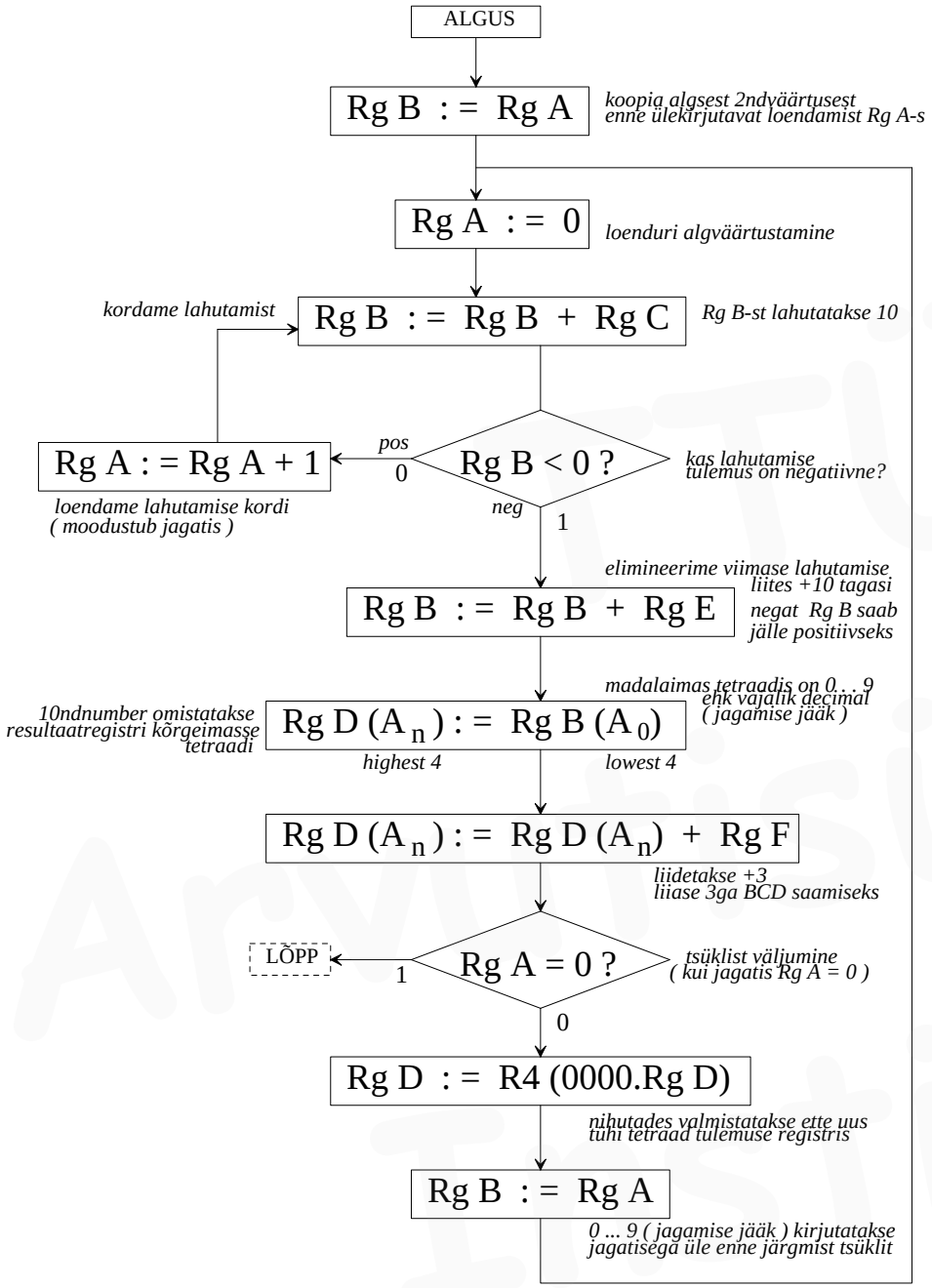
algoritmi koostamisel selgub, et vaja on **6 registrit** :

(*3 registrit nendest sisaldavad konstanti*)

Rg A : algne **binary** (teisendatav)
Rg B : jäägi akumulaator
Rg C : *konstant* $-10_{10} = \ldots . 1110110_{tk}$
Rg D : resultaat **BCD** (8421) + 3 (XS3)
Rg E : *konstant* $+10_{10} = \ldots . 001010_2$
Rg F : *konstant* $+3_{10} = \ldots . 0011_2$



eelnev **AGS** korratud koos sammude selgitustega / kommentaaridega :


ALGUS
Rg B : = Rg A
koopia algsest 2ndväärtusest enne ülekirjutavat loendamist Rg A-s
Rg A : = 0
loenduri algväärtustamine
kordame lahutamist
Rg B : = Rg B + Rg C
Rg B-st lahutatakse 10
Rg A : = Rg A + 1
loendame lahutamise kordi (moodustub jagatis)
pos
0
Rg B < 0 ?
kas lahutamise tulemus on negatiivne?
neg
1
elimineerime viimase lahutamise liites +10 tagasi negat Rg B saab jälle positiivseks
Rg B : = Rg B + Rg E
madalaimas tetraadis on 0 . . . 9 ehk vajalik decimal (jagamise jääk)
10ndnumber omistatakse resultaatregistri kõrgeimasse tetraadi
Rg D (A n) : = Rg B (A 0)
highest 4
lowest 4
Rg D (A n) : = Rg D (A n) + Rg F
liidetakse +3 liiase 3ga BCD saamiseks
LÕPP
1
Rg A = 0 ?
tsüklist väljumine (kui jagatis Rg A = 0)
0
Rg D : = R4 (0000.Rg D)
nihutades valmistatakse ette uus tühi tetraad tulemuse registris
Rg B : = Rg A
0 ... 9 (jagamise jääk) kirjutatakse jagatisega üle enne järgmist tsüklit

.