

# Kompiuterių Architektūros konspektas

Benediktas G. VU MIF, 2011-2012m

(radus netikslumų, turint klausimų rašyti benediktog@gmail.com)

Šios versijos data yra: 2012-10-08

**Naujausią šio konspekto versiją galima rasti adresu:**

<https://docs.google.com/document/d/1UzYBaxLhF8pqvRaBMstj6RDrV5SsCjklNLw0f4K-r7Y/edit>

## Turinys

<b>Darbas su skaičiavimo sistemomis.....</b>	<b>3</b>
<i>Vertimas tarp skaičiavimo sistemų.....</i>	<b>3</b>
Vertimas iš n-tainės į dešimtainę.....	3
Vertimas iš dešimtainės į n-tainę.....	4
Vertimas tarp „bendro pagrindo“ skaičiavimo sistemų.....	5
<i>Dvejetainiai skaičiai.....</i>	<b>6</b>
Apie dvejetainius skaičius ir dvejetainių vertimas dešimtainiais .....	6
Dešimtainių vertimas dvejetainiais.....	7
Skaičiai su ženklu ir be ženklo.....	8
<i>Baitų (arba žodžių) sudėties ir atimties operacijos.....</i>	<b>9</b>
Baitų sudėtis dvejetainėje sistemoje.....	9
Baitų atimtis dvejetainėje sistemoje.....	10
Žodžių sudėtis ir atimtis šešioliktainėje sistemoje.....	11
<b>Architektūros ypatybės.....</b>	<b>13</b>
<i>Esminės architektūros detalės.....</i>	<b>13</b>
Atmintis ir jos segmentacija.....	13
Procesoriaus registrai.....	14
<i>Adresavimas.....</i>	<b>15</b>
Efektyvus ir absoliutus adresas.....	15
Adresavimo būdai.....	16
Absoliutus (4baitai iš nurodytos atminties vietos imami tokia pat tvarka, kaip ir tiesioginiam absoliučiam).....	16
Plėtimo pagal ženklą taisyklė.....	17
<i>Stekas.....</i>	<b>18</b>
<i>Status Flag registras (SF).....</i>	<b>19</b>
Kas yra SF registras?.....	19
SF bitų reikšmės.....	19
Uždavinių, kuriuose reikia nustatyti naują SF reikšmę, sprendimas.....	20
<b>Programavimas assembleriu.....</b>	<b>21</b>
Kas nagrinėjama šiame skyriuje?.....	21
<i>Kaip susikompiliuoti, pasileisti kodą.....</i>	<b>21</b>
<i>Pirmoji programa.....</i>	<b>22</b>
<i>Prieš pradėdant rašyti pirmą programą.....</i>	<b>22</b>
MOV ir INT komandos.....	23
Bendra programos struktūra.....	24
<i>„Labas pasauli“ programa.....</i>	<b>25</b>
Žymės (label) ir CMP, JMP komandos.....	26
Komandinės eilutės parametrų skaitymas.....	27
<b>Darbas su įvairaus formato skaičiais.....</b>	<b>29</b>
<i>Koprocesoriaus (8087) realiųjų skaičių formatai (float'ai).....</i>	<b>29</b>
Dešimtainio slankaus kabelio vertimas į šešioliktainį užrašą.....	30
Šešioliktainio realaus skaičiaus vertimas į dešimtainį užrašą.....	31
<b>Egzamino užduočių pavyzdžiai su sprendimais.....</b>	<b>32</b>
Bendra informacija apie pateiktus sprendimus.....	32

<i>Adresavimas [2].....</i>	<i>32</i>
<i>Status Flag registras [1].....</i>	<i>33</i>
<i>Slankaus kablelio skaičiai (float'ai) [1].....</i>	<i>34</i>
<b>Papildoma informacija.....</b>	<b>36</b>
Įvairios sąvokos, taisyklės ir pastabos.....	36

# Darbas su skaičiavimo sistemomis

## Vertimas tarp skaičiavimo sistemų.

Pozicinėse skaičiavimo sistemose skaitmens vertė tiesiogiai priklauso nuo jo pozicijos skaičiuje, pvz.: 123 ir 321 yra skirtingi skaičiai.

Žmonės įprastai naudoja dešimtainę pozicinę skaičiavimo sistemą.

Joje skaičius 123 atrodo taip:

$$1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 = 123$$

Kartais dirbant su skirtingomis skaičiavimo sistemomis norint nesusipainioti su kuria sistema dirbama, po skaičiaus užrašomas indeksas, parodantis kokia tai sistema. Pavyzdžiui  $123_{10}$  reiškia skaičių 123 dešimtainėje skaičiavimo sistemoje,  $456_9$  reiškia skaičių 456 devintainėje skaičiavimo sistemoje ir pan.

Programuojant assembleriu dažniausiai tenka susidurti su šešioliktaine, dvejetainė ir dešimtainė skaičiavimo sistemomis, indeksus rašyti programiniame kode arba neįmanoma, arba pernelyg sudėtinga, todėl po skaitmens tiesiog parašoma raidė, iš kurios transliatorius (kompiliatorius) nusprendžia, kokioje skaičiavimo sistemoje užrašyti duotąjį skaičių.

Jokios raidės, pvz skaičius 19 reiškia dešimtainį skaičių 19

Po skaičiaus einanti **h raidė, reiškia, kad skaičius pateiktas šešioliktainėje sistemoje**.  $19_h$  reiškia, kad tai yra šešioliktainis skaičius 19. Dešimtainėje sistemoje to paties skaičiaus reikšmė yra 25. Kaip verstis tarp sistemų aprašyta tolesniuose skyreliuose.

Po skaičiaus einanti **b raidė, reiškia, kad skaičius pateiktas dvejetainėje sistemoje**.  $101_b$  tai yra dvejetainis skaičius 101. Jo reikšmė dešimtainėje sistemoje yra 5.

Vis tik svarbu žinoti ir tai, kad **egzaminuose** registrų ar baitų reikšmės paprastai pateikiamos baituose ar žodžiuose be h raidės ir nenurodant, kokia tai skaičiavimo sistema. Tada tiesiog juos reikia interpretuoti, kaip pateiktus šešioliktainėje sistemoje.

## Vertimas iš n-tainės į dešimtainę

Galioja tas pats, kas praeitame pavyzdyje su skaičiumi 123, tik vietoj 10 reikia panaudoti skaičių  $n$ .

Ką daryti su skaičiaus  $n$  laipsniais laipsniais?

Galima įsivaizduoti, kad po 123 yra kablelis, tarkim tai skaičius 123,0000 (šiuo atveju nuliai po kablelio nieko nereiškia)

Tada reikia žinoti, kad skaitmenims einantiems nuo kablelio į kairę pusę reikia laipsnio rodiklį didinti po 1 pradedant nuliumi. Skaitmenims dešinėje kablelio pusėje į dešinę pusę laipsnio rodiklį mažinti pradedant nuo -1. Toliau pateikiamas to pavyzdys:

Tarkim turime skaičių **241,64** septintainėje pozicinėje (skaitmenys tik nuo 0 iki 6, nes septintainė sistema, kaip kad dešimtainėje nuo 0 iki 9) skaičiavimo sistemoje ( $n=7$ ).

Pozicija nurodo kokių laipsnių pakelti skaičių  $n$ .

$n$  pakeltas pozicijos laipsniu padauginamas iš skaitmens esančio toje pozicijoje reikšmės. Gauti rezultatai sudedami, taip gauname tą patį skaičių tik jau dešimtainėje sistemoje.

Skaitmenų reikšmės	2	4	1	, (kablelis)	6	4
Pozicijos	2	1	0	<<didėja nuo 0 >>mažėja nuo -1	-1	-2

Dešimtainėje tai bus:

$$\begin{aligned} & 2 \cdot 7^2 + 4 \cdot 7^1 + 1 \cdot 7^0 + 6 \cdot 7^{-1} + 4 \cdot 7^{-2} = \\ & = 2 \cdot 49 + 4 \cdot 7 + 1 \cdot 1 + 6/7 + 4/49 = \\ & = 98 + 28 + 1 + 6/7 + 4/49 = \\ & 127 + 42/49 + 4/49 = 127 + 46/49 \end{aligned}$$

## Vertimas iš dešimtainės į n-tainę

Bendruoju atveju algoritmas toks: dešimtainių skaičių daliname iš n per eilę žingsnių, kitame jau dalindami prieš tai atlikto dalybos veiksmo rezultata, tiek kartų, kol rezultatas pasidaro lygus nuliui. Tada liekanas surašome atvirkščia tvarka negu gavome, gautas skaičius ir bus tas pats skaičius kitoje skaičiavimo sistemoje.

Tarkim norime dešimtainių skaičių 502 paversti į n-tainę skaičiavimo sistemą.

Tarkim  $n=7$  (t.y. versime į septintainę skaičiavimo sistemą)

$$502 / 7 = 71, \text{ liekana } 5$$

$$71 / 7 = 10, \text{ liekana } 1$$

$$10 / 7 = 1, \text{ liekana } 3$$

$$1 / 7 = 0, \text{ liekana } 1 \text{ (*rezultatas nulis*, veiksmai toliau nebeatliekami)}$$

Gautas skaičius septintainėj sistemoj yra 1315

Tai galime užrašyti tiesiog taip  $502_{10}=1315_7$

Su sveikais skaičiais veiksmai atliekami taip kaip parodyta pavyzdyje prieš tai. Vis dėl to jei turime realųjį skaičių, tai su jo sveikąja dalimi atliekame prieš tai aprašytus veiksmus, o su dalimi po kablelio (vadinama trupmenine dalimi) dirbame atskirai.

Tarkim verčiame į septintainę dešimtainį skaičių  $502,2356_{10}$

Reikia dirbti su sveikąja ir trupmenine dalimi atskirai.

Su sveikąja dalimi jau išsprendėme ir turime rezultatą  $1315_7$

Atmetę sveikąją dalį turime skaičiaus trupmeninę dalį 0,2356

Versdami į n-tainę:

- Sudauginame prieš tai buvusį rezultatą iš n
- Užrašome gauto skaičiaus sveikąją dalį
- Atmetę gautojo skaičiaus sveikąją dalį dauginame iš n vėl reikiamą kiekį kartų:
  - kol pastebėsime cikliškumą – tada galėsime užrašyti periodą (į skliaustelius)
  - tiek kiek reikia uždaviniui išspręsti, mantisei užrašyti ir pan.

Tarkim šiuo atveju mus domina tik 5 skaitmenys po kablelio:

$$0,2356 * 7 = 1,6492$$

$$0,6492 * 7 = 4,5554$$

$$0,5444 * 7 = 3,8108$$

$$0,8108 * 7 = 5,6756$$

$$0,6756 * 7 = 4,7292$$

Tai atsakymas bus, kad  $498,2356_{10}$  yra apytiksliai (ne tiksliai, nes rašėme tik 5 skaitmenis po kablelio, ir neatlikome apvalinimo)  $1315,14354_7$

*Noriu pabrėžti, kad dirbant su trupmenine dalimi rezultatus surašome ta pačia eilės tvarka, kuria gavome (kitaip nei dirbdami su sveikąja skaičiaus dalimi, kur liekanas rašėme atvirkščia tvarka, nei jos buvo gautos)*

### Pastabos:

- n (jei skaičius n parodo kelintainė sistema) visada yra natūralus skaičius didesnis už vieną
- keičiant neigiamo skaičiaus skaičiavimo sistemą, dirbama kaip su teigiamu, bet gavus rezultatą minusas vėl uždedamas. Vis dėl to, dirbant su ribotais atminties laukais (baitais, žodžiais) neigiamų skaičių realiai nėra, yra tik vienetukų ir nuliukų sekos, todėl tada dirbant su neigiamais skaičiais galioja kitos taisyklės. Plačiau tai išnagrinėta skaičių su ženklų ir be ženklo temoje.

## Vertimas tarp „bendro pagrindo“ skaičiavimo sistemų

Verčiant tarp skaičiavimo sistemų dažniausiai tenka panaudoti dešimtainę kaip tarpinę (tarkim iš 7tainės versdami į 9tainę pirmiau 7tainį skaičių pasiverčiame 10tainingu, o tik tada gautą 10tainį verčiame 9tainingu)

Tačiau taip daryti ne visada reikalinga:

Tarkim turime dvi sistemas,  $p$  ir  $q$

Taip pat tarkime, kad  $p > q$  (turėdami priešingai galime skaičius sukeisti)

Kai  $p = a^k$  ir  $q = a^m$  turi galioti sąlygos:

- $a, k, m$  – natūralieji skaičiai,  $a > 1$
- $k$  dalybos iš  $m$  liekana yra nulis (t.y.  $k \bmod m = 0$ )

Tada skaičiavimo sistemas (šiam konspekte) vadinkime tiesiog „bendo pagrindo“ arba „suderintomis“

Jose kiekvieną  $p$  sistemoje užrašyto skaičiaus skaitmenį galima išskleisti į  $(k \div m)$  skaitmenų  $q$  sistemoje ir atvirkščiai (reikia pradėti tai daryti nuo dešinės pusės, jei kairėje pritrūktų skaitmenų galima prisirašyti nulius).

Nagrinėjant kompiuterių architektūrą paprastai dirbama su sistemomis, kur  $a=2$ .

Pavyzdžiui: bendro pagrindo sistemos yra 2tainė ir 16tainė

Šiuo atveju:

$$a=2, k=1, m=4$$

$k \div m = 4 \div 1 = 4$ , ( $4 \bmod 1 = 0$ ) t.y. dalybos  $4/1$  liekana yra 0.

Šiuo atveju kiekvieną skaitmenį 16tainėje sistemoje galima išskleisti į 4 dvejetainius skaitmenis ir atvirkščiai.

Tarkim šešioliktainis skaičius 2E atrodytų taip, taip pat šį skaičių galima gauti iš dvejetainės sistemos (t.y. galimi veiksmai ir į vieną ir į kitą pusę: iš šešioliktainės į dvejetainę ir atvirkščiai).

- Viršutinėje eilutėje yra šešioliktainis skaičius
- Apatinėje eilutėje yra dvejetainis skaičius

2				E			
0	0	1	0	1	1	1	0

10tainė	2tainė	16tainė
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7

10tainė	2tainė	16tainė
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Nesunku pastebėti, kad einant nuo dešinės pusės į kairę dvejetainių skaitmenų svoriai yra tokie: 1, 2, 4, 8 ir t.t. (didėja po 2 kartus, sistema 2tainė,  $0 \cdot \text{svoris} = 0$  ir  $1 \cdot \text{svoris} = \text{svoris}$ )

# Dvejetainiai skaičiai

## Apie dvejetainius skaičius ir dvejetainių vertimas dešimtainiais

Paprastai veiksmai atliekami su baitais arba žodžiais.

**Bitas** – vienas dvejetainis skaitmuo (0 arba 1), mažiausia nedaloma atminties dalis.

**Baitas** – 8 bitų rinkinys (0..255 skaičius be ženklo) arba (-128..+127 skaičius su ženklu)

**Žodis** – 16 bitų rinkinys (0..65535 skaičius be ženklo) arba (-32768..+32767 skaičius su ženklu)

**Rezultato laukas** – tam tikrų aritmetinių/loginių veiksmų vykdymo metu gauto rezultato jauniausi 8 arba 16 bitų, priklausomai nuo to operacija atlikta su baitais ar žodžiais (Carry Flag bitas rezultato laukui nepriklauso).

Šiame skyrelyje pavyzdžiai pateikiami su baitais, bet žodžiams galioja tos pačios taisyklės.

Bendruoju atveju:

Kai lauką sudaro **n bitų**: (vieno baito atveju  $n=8$ )

Skirtingų reikšmių gali būti  $2^n$  (vieno baito atveju 256)

Skaičiuose be ženklo reikšmių diapazonas  $0..2^n-1$  (vieno baito atveju 0..255)

Skaičiuose su ženklu reikšmių diapazonas  $-2^{n-1}..+2^{n-1}-1$  (vieno baito atveju -128..127)

Kaip nustatyti, kokia skaičiaus reikšmė (be ženklo) yra dešimtainėje sistemoje turint dvejetainį užrašą?

### Pavyzdys:

Tarkim turime baitą:

- Pirma eilutė reiškia bitų reikšmes
- Antra eilutė - skaitmens poziciją (iš dešinės į kairę nuo 0 iki  $n-1$ , 8bitai tai nuo 0 iki 7tos)
- Trečia skaitmens svorį (2 pakelta pozicijos laipsniu)

0	1	1	0	1	1	0	1
7	6	5	4	3	2	1	0
128	64	32	16	8	4	2	1

Norėdami nustatyti kokia čia dešimtainė reikšmė užkoduota (skaičius BE ŽENKLO!) sudedame svorius bitukų, kurių reikšmės yra vienetai.

Turime:  $64+32+8+4+1=109_{10}$ .

Tie patys veiksmai detaliau (visi skaičiavimai dešimtainėje sistemoje):

$$\begin{aligned} & 0 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = \\ & = 0 \cdot 128 + 1 \cdot 64 + 1 \cdot 32 + 0 \cdot 16 + 1 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 = \\ & = 0 + 64 + 32 + 0 + 8 + 4 + 0 + 1 = \\ & = 64 + 32 + 8 + 4 + 1 = 109_{10} \end{aligned}$$

Turėdami skaičiaus be ženklo reikšmę, nesunkiai galima pasiversti jį skaičiumi su ženklu ir atvirkščiai.

## Dešimtinių vertimas dvejetainiais

Tarkim turime skaičių be ženklo. Norime jį užkoduoti į kažkiek baitų dvejetainiu kodu. Jei turime neigiamą skaičių, jį pirmiausia pasiverčiame skaičiumi be ženklo.

### Principas:

Einame nuo didžiausios pozicijos link nulinės ir lyginame pozicijos svorį, su turima reikšme.

- Jei turima reikšmė didesnė arba lygi tos pozicijos svoriui, toje pozicijoje, tai iš turimos reikšmės atimame skaitmens svorį, toje vietoje įrašome 1tuką ir einame į dešinę.
- Jei turima reikšmė mažesnė už to skaitmens svorį, toje pozicijoje rašome **0uką** ir einame į dešinę **nekeisdami turimos reikšmės**.
- Kai turima reikšmė yra 0 baigiame darbą, jei praėję iki nulinės pozicijos turime ne nulinį skaičių reiškia kažkur padarėme klaidą arba duotas dešimtainis skaičius netelpa tokio dydžio atminties lauke (vienam baite maksimali reikšmė 255, dviejuose baituose 65535).

### Pavyzdys:

Tarkim turime dešimtainį skaičių be ženklo 159. Norime jį užrašyti viename baite dvejetainė skaičiavimo sistema.

- Lyginame 159 ir 128.  $159 > 128$  todėl rašome **1**.
  - Turima reikšmė  $159 - 128 = 31$
- Lyginame 31 ir 64.  $31 < 64$ , todėl rašome **0**.
  - Turima reikšmė išlieka 31
- Lyginame 31 ir 32,  $31 < 32$ , todėl rašome **0**
  - Turima reikšmė išlieka 31
- Lyginame 31 ir 16,  $31 > 16$ , todėl rašome **1**
  - Turima reikšmė  $31 - 16 = 15$
- Lyginame 15 ir 8,  $15 > 8$ , todėl rašome **1**
  - Turima reikšmė  $15 - 8 = 7$
- Lyginame 7 ir 4,  $7 > 4$ , todėl rašome **1**
  - Turima reikšmė  $7 - 4 = 3$
- Lyginame 3 ir 2,  $3 > 2$  todėl rašome **1**
  - Turima reikšmė  $3 - 2 = 1$
- Lyginame 1 ir 1,  $1 = 1$ , todėl rašome **1**
  - Turima reikšmė 0, pasiekta nulinė pozicija, klaidų nėra.

Vadinasi:  $159_{10} = 1001\ 1111_2$

### Kitas būdas:

Galima skaičių BE ženklo, 159 versti į dvejetainę ir kitu būdu:

Jei turėtume žodį, turėtume dalinti iš 256, ir gautume atskirų baitų reikšmes (rezultatas vyresnysis baitas, liekana jaunesnysis).

Turėdami baitą jį galima užrašyti dviem šešioliktainiais (skaldymas į pusbaičius).

$159/16 = 9$ , liekana **15**. 15 šešioliktainėje reiškia raidę F. Todėl tai yra šešioliktainis skaičius **9F**

Šešioliktainį išsiskleidžiame į dvejetainį:  $9h = 1001$ ,  $Fh = 1111$

Vadinasi:  $159_{10} = 1001\ 1111_2$

**Turint dešimtainius neigiamus ir norint užrašyti dvejetainėje sistemoje** (iš to lengvai galima pasiversti ir į šešioliktainę ar aštuntainę) **viename baite arba žodyje galimi tokie to atlikimo būdai:**

### Pirmas būdas:

Tą patį dešimtainį skaičių pasiverčiame skaičiumi be ženklo („pritempiame iki skaičiaus be ženklo intervalo“) ir paverčiame į dvejetainę sistemą vienu iš prieš tai nurodytų būdų.

### Antras būdas:

Ignoruojame turimą minusą prieš skaičių, turimą skaičių užrašome dvejetainė sistema (pagal prieš tai nurodytus veiksmus) ir pakeičiame jo ženklą (t.y. visus baito arba žodžio bitus invertuojame ir pridedame vieneta).

## Skaičiai su ženklų ir be ženklų

Bet kurio rezultato lauko (baito, žodžio) reikšmę galima interpretuoti ir kaip skaičių su ženklu, ir kaip skaičių be ženklo.

Skaičiaus su ženklu ženklą parodo vyriausias (pats pirmas kairėje) baito arba žodžio *bitas*.

- Kai jo reikšmė 0, ženklas +
- Kai jo reikšmė 1, ženklas -

Viename baite (tie patys principai galioja ir užrašant per daugiau baitų, žodžiuose, dvigubuose žodžiuose ir pan.):

0 yra 0000 0000

1 yra 0000 0001

-1 yra 1111 1111

Taip yra todėl, kad sudėjus du skaičius su ženklu turime gauti teisingą rezultatą

Realiai viename baite turime 128 neigiamus, ir 128 teigiamus skaičius intervalas yra [-128; 127], kur nulio ženklas yra +.

Tarkim dešimtainis 130 be ženklo dvejetainėje yra 1000 0010, su ženklu tai būtų -126.

### Pasivertimas į skaičių BE ženklo arba SU ženklu (reikšmė dvejetainiam pavidale nesikeičia)

Tarkim turime sveikąjį skaičių  $x$ .

Žinome jo reikšmę, ir norime užrašyti jį viename baite dešimtaine sistema (iš dešimtainės paprasta gauti ir kitas).

Skyrelio pradžioje paminėti intervalai:

Viename baite sveikas skaičius be ženklo priklauso intervalui [0; 255]

Viename baite sveikas skaičius su ženklu priklauso intervalui [-128; 127]

Reiškia reikia privesti skaičių  $x$  iki kažkurio iš šių intervalų, priklausomai nuo to ar reikalingas skaičius su ženklu ar be ženklo.

Kadangi n bitų laukai visi yra riboti (baito atveju  $n=8$ ), tai galima pridėti/atimti iš skaičiaus  $x$ , skaičių  $2^n$  tiek kartų, kiek reikia, kol šis pakliūva į norimą intervalą (skaičiaus su ženklu arba be ženklo).

#### To paaiškinimas būtų toks:

Skaičių ribotame atminties lauke galima vienu metu laikyti ir skaičiumi su ženklu, ir skaičiumi be ženklo, jei pridėtume arba atimtume po vieną  $2^n$  kartų vėl gautume tą patį rezultatą ribotame atminties lauke, nes „tai kas išlenda už rezultato lauko ribų, perteklius“ įtakos rezultatui ribotame lauke neturi, taip pat turėdami visą rezultato lauką užpildytą vienetinais bitais ir pridėję vieną gautume nulį ir atvirkščiai. Taip pat galioja iš pažiūros neįprastos savybės, kad viename baite  $FF+01=00$  ir  $00-01=FF$

Iš tikrųjų pridėdami arba atimdami po vieną  $2^n$  kartų mes tarpiniuose skaičiavimuose gauname visas įmanomas reikšmes tame apribotame rezultato lauke „apsukame ratą“ ir vėl grįžtame prie pradinės, tai realiai nieko ir nepakeičiame, tik dešimtainiam pavidale jau turime patogesnę reikšmę, be to priklausančią norimam – skaičiaus su ženklu ar be ženklo intervalui.

**Tarkim** skaičius  $x$  yra -189 dešimtainėje, viename baite (bitų kiekis  $n=8$ , galimų reikšmių  $2^8=256$ ) skaičius be ženklo bus  $-189+256=67$ . Jis patenka ir į skaičiaus su ženklo intervalą, todėl ir skaičius su ženklo ir be ženklo dešimtainėje sistemoje bus 67. šešioliktainėje šis skaičius yra 43, dvejetainėje 0100 0011.

### Ženklo keitimas (tik skaičiams SU ženklu) – dvejetainiam pavidale gauname kitą reikšmę

Turint dvejetainę išraišką atliekame du veiksmus:

- Invertuojame visus bitus (nulių pakeisti vienetais, vienetus nuliais)
- Pridedame vien1

#### Svarbu:

- Sprendžiant įvairius uždavinius reikia žinoti kada skaičių reikia interpretuoti kaip skaičių su ženklu, o kada kaip skaičių be ženklo.

Pvz.: Tikrinant ar įvyko overflow turime žiūrėti į skaičius ir rezultatą kaip skaičius su ženklu, gavus klaidingą rezultatą  $OF=1$

- 1baito skaičiai dažniausiai yra suprantami kaip skaičiai su ženklu, norint užrašyti tą patį skaičių dviejuose baituose reikia praplėsti pagal ženklo bitą. (vyresniame baite sudėti 1tukus arba 0, tokius koks yra ženklo bitas). To dažnai prireikia skaičiuojant poslinkį kodo segmente, kai jis yra vieno baito, o IP registras dviejų baitų.



# Baitų (arba žodžių) sudėties ir atimties operacijos

Su baitais, žodžiais (2baitai), dvigubais žodžiais (4baitai) ir t.t. galima atlikti įvairius loginius ir aritmetinius veiksmus.

Šiame skyrelyje aprašytos dvejetainės ir šešioliktinės baitų sudėties ir atimties aritmetinės operacijos.

*Sudedant/atimant skirtingo dydžio laukus reikia mažesnįjį privesti prie didesniojo, t.y, suvienodinti jų dydžius, skaičiuojant poslinkius tam naudojama ženkle plėtimo taisyklė aprašyta šiame skyrelyje (čia).*

## Baitų sudėtis dvejetainėje sistemoje

Sudėtis vykdoma iš dešinės į kairę, sudedamos bitų poros. Jei reikia atliekamas pernešimas (priešingas veiksmas pasiskolinimui atimtyje).

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$1 + 1 = 0$ , pernešimas į sekantį (kairiau esantį) bitą, jei to bito nėra galima iš kairės prisirašyti nuliukų kiek tik reikia, jei kam nuo to aiškiau, nes tai rezultato nekeičia.

Išnagrinėkime **pavyzdį**:

1 baito operacijomis sudedami 156 ir -23.

1. Pirmiausia užrašome šiuos skaičius dvejetainėje sistemoje:

Kadangi 156 mažiau už 255 reiškia telpa vienam baite, skaičius teigiamas ir patenka į skaičiaus be ženklo intervalą, todėl elgsimės kaip su skaičiu be ženklo (išsirašome per dvejetainio laipsnių sumą, kad žinotume kur sudėti nuliukus, o kur vienetukus  $156 = 128 + 16 + 8 + 4$ )

Kadangi -23 daugiau už -128 reiškia telpa vienam baite, skaičius neigiamas todėl pirmiausia pasiverčiame jį į skaičių be ženklo. Dirbama su 1 baitu, t.y. 8 bitais, o  $2^8 = 256$ . Pridėję 256 prie -23 gauname 233 ir tai jau yra teisingas (toks pat) skaičius BE ženklo.  
 $233 = 128 + 64 + 32 + 8 + 1$

Atliekame sudėties operaciją su šiais skaičiais:

Eilutėse:

1. Surašytos 156 bitukų reikšmės
2. Surašytos -23 bitukų reikšmės
3. Surašytos rezultato bitukų reikšmės (kairė pozicija papildomas bitas, kuris dedamas į CF bitą Status Flag registre, žalias langelis)

+	1	0	0	1	1	1	0	0
	1	1	1	0	1	0	0	1
1	1	0	0	0	0	1	0	1

Atliekame veiksmus nuo dešinės į kairę.

$$0 + 1 = 1$$

$$0 + 0 = 0$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \text{ (pernešimas į sekančią bitų porą, ji padidinama vienetu), dėl pernešimo būtų fiksuojamas flagas AF=1}$$

$$1 + 0 + 1 \text{ (perneštas bitas)} = 0, \text{ suma didesnė už 1, vėl įvyko pernešimas}$$

$$0 + 1 + 1 \text{ (perneštas bitas)} = 0, \text{ suma didesnė už 1, dar vienas pernešimas}$$

$$0 + 1 + 1 \text{ (perneštas bitas)} = 0$$

$$1 + 1 + 1 \text{ (perneštas bitas)} = 1, \text{ suma didesnė už 1, įvyko pernešimas į papildomą bitą (Carry Flag'a)}$$

## Baitų atimtis dvejetainėje sistemoje

Atimtis, kaip ir sudėtis vykdoma bitas po bito iš dešinės į kairę, esant reikalui (atimant didesnę iš mažesnio)

$$0 - 0 = 0$$

$$1 - 0 = 1$$

$$1 - 1 = 0$$

$0 - 1 = 1$  pasiskolinimas (vykdomas skolinimas iš kairiau esančios bitų poros, jei jos nėra, galima įsivaizduoti, kad ta bitų pora yra 0 ir 0)

Išnagrinėkime pavyzdį:

Vieno baito atimties operaciją atlikime su skaičiais 25 ir 142.

Pirmiausia pasiverčiame šiuos skaičius į dvejetainę sistemą ir užrašome viename baite:

$$25_{10} = 0001\ 1001_2$$

$$142_{10} = 1000\ 1110_2$$

Atliekame atimties operaciją su šiais skaičiais:

Eilutėse:

1. Surašytos 25 bitukų reikšmės
2. Surašytos 142 bitukų reikšmės
3. Surašytos rezultato bitukų reikšmės (kairė pozicija papildomas bitas, kuris dedamas į CF bitą Status Flag registre, žalias langelis)

	0	0	0	1	1	0	0	1
	1	0	0	0	1	1	1	0
1	1	0	0	0	1	0	1	1

Atliekame veiksmus nuo dešinės į kairę.

Jei yra vykdomas pasiskolinimas, **skolos skaičiukas pateiktas raudonu šriftu**.

$$1 - 0 = 1$$

$0 - 1 = 1$  (iš mažesnio atimamas didesnis, todėl skolinamės iš kairiau esančios skaičių poros)

$0 - 1 - 1 = 0$  (iš mažesnio atimamas didesnis, todėl skolinamės iš kairiau esančios skaičių poros)

$1 - 1 - 1 = 1$  (iš mažesnio atimamas didesnis, todėl skolinamės iš kairiau esančios skaičių poros 1-2)

$$1 - 0 - 1 = 0$$

$$0 - 0 = 0$$

$$0 - 0 = 0$$

$$0 - 1 = 1$$

Taip pat, kadangi iš mažesnio skaičiaus (laikant, kad abu be ženklo) atimamas didesnis rezultatas nebetilps skaičiaus be ženklo intervale  $[0;255]$ , būtent dėl tos priežasties atliekant atimtį aštuntoje iš dešinės poroje prireikia pasiskolinimo „iš už rezultato lauko ribų“, todėl flagas CF=1.

**Paprastai atimtis ir skolinimaisi yra sunkiai perprantami**, ir jei kam nors atrodytų aiškiau tai galima iliustruoti, tokiu pavyzdžiu:

Skolos perduodamos iš kartos į kartą. Žmogus A pasiskolino, bet negrąžino skolos, todėl paveldėjo jo vaikas B, šis neapmokėjo, tai perėjo B vaikui C ir tik vaikas D apmokėjo skolą, todėl D vaikui E šia skola rūpintis nebeteks, nebent jis pats skolinis ir perduos tą „naštą“ savo vaikams. Panašiai ir dvejetainiuose skaičiuose, kur galima įsivaizduoti, kad kairiau esantis bitas yra konkretaus bito vaikas ir t.t.

## Žodžių sudėtis ir atimtis šešioliktainėje sistemoje

Veiksmai atliekami atsižvelgiant į tas pačias taisykles, kaip ir sudėties/atimties dešimtainėje sistemoje atveju (vienas iš patogiausių būdų – sudėtis stulpeliu).

Vienintelis pastebimas skirtumas, kad yra ne tik skaitmenys nuo 0 iki 9, bet ir raidės A..F reiškia skaitmenis. Reikia žinoti jų vertes:

$$A=10$$

$$B=11$$

$$C=12$$

$$D=13$$

$$E=14$$

$$F=15$$

Tuomet sudedant arba atimant, pernešimas/pasiskolinimas įvyksta ne peržengus dešimtį, 16 (sistema nebe dešimtainė, o šešioliktainė).

Tarkim sudedam skaitmenis  $x$  ir  $y$ .

Rezultate rašom  $(x+y) \bmod 16$

Kitoj poroj pridėdam  $(x+y)$  div 16 („pernešimas į kitą porą, t.y. dviejų skaitmenų sudėties perteklių turėsime mintyti“)

**SVARBU žinoti, kad efektyvus adresas visada užima 4 skaitmenis, o absoliutus 5 skaitmenis. Jei gaunama daugiau skaitmenų, į papildomus dėmesio nekreipiama.**

**Tačiau skaičiuojant registrų sumą ribojami tik atskirų registrų dydžiai, bet ne jų suma.**

**Tarkim  $AX=FFFF$ ,  $BX=0002$  sumą rašytume 10001, o ne 0001 kaip efektyvaus adreso atveju.**

Pavyzdys su sudėtim (vykdoma iš dešinės į kairę):

+	F	3	2	7
	8	4	E	F
<hr/>				
1	7	8	1	6

Pernešimo vienetukai pažymėti raudonai.

$7 + F = (7+15=22)$ , dalybos iš 16 rez. **1** (pridėsime prie poros kairiau), liekana 6 rašom į rezultatą)

$2 + E + \mathbf{1} = (3 + E = 3 + 14)$ , dalybos iš 16 rez. **1** (pridėsime prie poros kairiau), liekana 1 rašom į rezultatą)

$3 + 4 + \mathbf{1} = 8$  (dalybos iš 16 rez. 0, nieko nepridedame prie poros kairiau, liekana 8 rašom į rezultatą)

$F + 8 = (15+8=23)$ , dalybos iš 16 rez. **1** (pridėsime prie poros kairiau), liekana 7 rašom į rezultatą)

Įrašomas **papildomas vienetukas**. Jei buvo skaičiuotas efektyvus adresas, į šį vienetuką reikia nekreipti dėmesio (toliau jo nebenaudoti).

### Pavyzdys su atimtim (vykdoma iš dešinės į kairę):

2	9	C	D
3	A	F	9
E	E	D	4

### Pasiskolinimo bitukai pažymėti raudonai

Veiksmai atliekami iš dešinės į kairę:

$D - 9 = 13 - 9 = 4$  (jokio skolinimosi iš poros kairiau)

$C - F = 12 - 15 = -3$ , (neigiamas sk. todėl  $16 - 3 = 13 = D$ , ir pasiskolinimas iš poros kairiau)

$9 - A - 1 = 8 - A = 8 - 10 = -2$  (neigiamas sk, todėl  $16 - 2 = 14 = E$ , ir pasiskolinimas iš poros kairiau)

$2 - 3 - 1 = 2 - 4 = -2$  (neigiamas sk, todėl  $16 - 2 = 14 = E$ , ir pasiskolinimas iš poros kairiau, kadangi šios nėra tai iš papildomos „įsivaizduojamos“ poros, kuri žodžio dydžio lauke vis tiek nebus išsaugota, o rezultato mums vis tiek reikės neigiamo)

Papildoma pora:

$0 - 0 - 1 = 0 - 1 = -1$ , t.y.  $16 - 1 = 15 = F$

Kadangi rezultatas skaičiuotas 4 skaitmenyse (2 baitai), tai į papildomas poras dėmesys nekreipiamas.

Šiuo atveju gautas skaičius būtų teisingas skaičiuojant 2 baituose, tačiau atliekant veiksmus ne apribotuose laukuose (baitas 2 skaitmenys, žodis 4 skaitmenys) reiktų pastebėti, kad atimamas didesnis iš mažesnio, todėl atsakymas bus neigiamas ir tada žinant, kad atsakymas neigiamas atimti atvirkščiai  $3AF9 - 29CD$ . Tuomet gautume  $-112C$ .

Galima spręsti ir taip, tuomet gavę neigiamą skaičių dar turėtume pridėti prie  $10000h$ .

$10000h + (-112Ch) = EED4h$

Nuliukų po vieneto yra tiek, kiek skaitmenų (baitas 2 skaitmenys, žodis 4 skaitmenys)

Kad gavome teisingai galime įsitikinti sudėdami  $EED4 + 3AF9 = 129CD$

*Vis tiek į perteklinį vienetą dėmesio nekreipiame (jei rezultatą saugome viename žodyje, tai tik 4 šešiolyktainiai skaitmenys, arba jei skaičiavome efektyvų adresą)*

# Architektūros ypatybės

## Esminės architektūros detalės

Mūsų nagrinėjamos architektūros ypatybės būdingos **Intel 8086** ir **Intel 8088** procesoriams.

Iš esmės jų architektūros yra panašios, tuo labiau, naudojamas tas pats instruction set (assemblerinių komandų rinkinys) **x86-16bit**.

*Kiekvienam kompiuteriui būtiniausios dalys yra procesorius ir atmintis. Kompiuteris savo darbo metu sugeba tik reaguoti į įvairių įrenginių signalus, vykdyti programos kodą ir keisti atminties būseną (atskirų registrų ir atminties baitų reikšmes).*

Mikroprocesoriuje išskiriamos keletas esminių dalių:

1. Registrai
2. ALU (Aritmetinis loginis įrenginys)
3. Vykdomojo ir absoliutaus adresų formavimo įrenginiai
4. Pertraukimų sistemos valdiklio (pertraukimų kontrolerio).

Dar viena kompiuterio architektūros dalį (ne paties mikroprocesoriaus dalį) išskirkime atmintį (RAM), kurioje saugomas tuo metu vykdomas kodas, duomenys, stekas ir pan.

Toliau detaliau nagrinėjama labiausiai programuotojui assembleriu aktuali procesoriaus dalis – registrai ir atmintis.

## Atmintis ir jos segmentacija



Mūsų nagrinėjamoje architektūroje darbinė atmintis (kurioje) yra 1MB dydžio, t.y.  $2^{20}=16^5$  baitų. Joje saugomi tuo metu vykdomos programos kodas, duomenys, stekas. Kiekvienas baitas atmintyje adresuojamas penkiais šešiolyktainiais skaitmenimis, kaip, kad schemoje kairėje.

Pirmasis kilobaitas (1024 baitai) nuo 00000 iki 003FF yra užpildyti pertraukimų vektorių lentelės duomenimis. Tai yra CS ir IP registrų rinkiniai. Išskirtus tam tikrą pertraukimo procedūrą, jos adresas imamas būtent iš vektorių lentelės, ir vykdomas tuo adresu esantis kodas.

Kad būtų patogiau dirbti su atmintimi, ji yra skirstoma į segmentus. Šioje architektūroje paragrafu vadinsime 16 baitų bloką. Tuomet segmentinis registras rodo, į segmento pradžią, kuri yra adresu: segmento\_registro\_reikšmė \* paragrafo dydis (t.y.10h)

Iš viso yra keturi segmentai:

CS – Code Segment – Segmentas, kuriame saugomas vykdomas mašininis kodas.

SS – Stack Segment – Steko segmentas, jame programa darbo metu „pasideda“ įvairias reikšmes.

DS, ES – Duomenų ir papildomas duomenų (extra) segmentai – saugomi duomenys, kuriuos apdoroja programa savo darbo metu.

Dalijimas segmentais realizuojamas segmentiniams registrams priskiriant 16bitų reikšmes. Tarkim jei žaliame fone turime kodo segmentą, mėlyname duomenų, raudonom linijom steko, o rudom extra segmentą, tai pagal paveikslėlį registrų reikšmės būtų:

**CS**=1234, **DS**=769A, **ES**=CF49, **SS**=DA34.

Segmentinį registrą padaugine iš 10h (paragrafo dydžio) gauname segmento pradžios absoliutų adresą. Pvz steko segmento DA340. Poslinkis segmente išreiškiamas efektyviu adresu nuo 0000 iki FFFF (kiekvieno dydis 10000h, arba tiesiog 64K).

Priklausomai nuo registrų reikšmių, gali ir persidengti (turėti bendrų laukų), ar net visiškai sutapti. Realiai atmintyje tiek vykdomas kodas, tiek duomenys yra saugomi nuliukų vienetukų sekomis ir nėra kažkaip ypatingai išskirti. Tiesiog įvairių registrų deriniais parodoma, kurias atminties vietas reikia vykdyti kaip kodą, kur duomenys ir pan.

**Segmentai yra cikliniai:** FFFF+1=0000, 0000-1=FFFF.

Atmintis irgi: FFFFF+1=00000, 00000-1=FFFFF.

Jei segmentas prasideda netoli atminties pabaigos, tai jo pabaigos dalis gali būti ir atminties pradžioje (dėl atminties cikliškumo, sekantis adresas po FFFFF yra vėl 00000)

## Procesoriaus registrai

**Registras** – Vieno arba dviejų baitų laukas procesoriuje, veikiantis panašiai kaip atmintis, tik sparčiau.

**Paragrafas** – 16 baitų dydžio blokas atmintyje (kitose architektūrose šis dydis gali skirtis!)

Intel 8088 turi tokius registrus:

**Darbiniai** (skaičiavimam, reikšmėm pasidėti, kaip faktiniai parametrai INT procedūroms):

- **AX** – vadinamas akumulatorium
- **BX** – vadinamas baziniu registru
- **CX** – vadinamas counteriu, ciklų skaitliuku ir pan.
- **DX** – vadinamas data registru

**Segmentiniai** (rodo į kur prasideda segmentas):

- **ES** – rodo extra segmento pradžią (paragrafo atmintyje nr.) - *dar vadinamas papildomu duomenų segmentu*
- **CS** – rodo kodo segmento pradžią (paragrafo atmintyje nr.)
- **SS** – rodo steko segmento pradžią (paragrafo atmintyje nr.)
- **DS** – rodo data segmento pradžią (paragrafo atmintyje nr.)

Paragrafas 1MB ramuose (kaip kad Intel 8088 atveju) yra 16baitų dydžio blokas.

**Indeksiniai** (eilutinėms komandoms, duomenų persiuntimui, kopijavimui, adresavimui)

- **SI** – Source Index šaltinio indeksas dažniausiai naudojamas su DS
- **DI** – Destination Index – eilutinėse komandose tik su segmentu ES

**Kiti:**

- **IP** – Instruction Pointer, kartu CS rodo poslinkį nuo kodo segmento pradžios, *komandos vykdymo metu rodo į sekančios vykdomos komandos pradžią.*
- **BP** – Base Pointer, kartu su SS naudojamas adresavimui steke.
- **SP** – Stack Pointer, kartu su SS rodo į steko viršūnę.
- **SF** – Status Flag, rodo paskutinės aritmetinės/loginės operacijos rezultato požymius (ženklą), naudojamas ir procesoriaus darbinio režimo reguliavimui. *Vadinamas procesoriaus būsenos registru.*

Darbinuose registruose galima naudoti ir jų vyresnius arba jaunesnius baitus.

?X registro jaunesnysis baitas yra ?L (Low), ir vyresnis ?H (High)

Vietoj klaustuko {A, B, C, D}

**Atliekant skaitymus iš atminties duomenys skaitomi pirmiau į jaunesnįjį baitą, tada į vyresnįjį!**

AX	
AH	AL

BX	
BH	BL

CX	
CH	CL

DX	
DH	DL

# Adresavimas

## Efektyvus ir absoliutus adresas

Kiekvieną atminties elementą galima adresuoti panaudojant tam skirtus registrus. Išskiriami du adresų atvejai efektyvus ir absoliutus.

**Efektyvų adresą (EA)** sudaro 4 šešiolyktainiai skaitmenys, juo parodomas poslinkis nuo segmento pradžios.

**Absoliutų adresą (AA)** sudaro 5 šešiolyktainiai skaitmenys, juo parodomas unikalus elemento atmintyje adresas. Formuojant absoliutų adresą visuomet panaudojamas, kuris nors iš segmentinių registrų.

Galioja sąryšis:

$$AA = seg * 10h + EA$$

kur:

AA – absoliutus adresas

EA – efektyvus adresas

seg – naudojamo segmento registro reikšmė

Dažnai šis sąryšis užrašomas per dvitaškį tokioje formoje **seg:EA**

Atvejis	Segmentas pagal default	Ar galima nurodyti kitą segmentą?	Efektyviu adresu laikoma
Komandos paėmimas vykdymui	CS	Ne	IP
Komandu su steku (PUSH, POP, PUSHF, POPF)	SS	Ne	SP
Eilutinės kom. - Šaltinis (source)	DS	Taip	SI
Eilutinės kom. - Gavėjas (destination)	ES	Ne	DI
Operandas atmintyje, kai jo EA suformuoti <b>naudojamas BP (bet kokioje kombinacijoje)</b>	SS	Taip	Pagal adresacijos baido r/m dalį nurodytas registrų ir poslinkio rinkinys (visų tų registrų laužtiniuose skliaustuose ir poslinkio suma).
Operandas atmintyje, kai jo EA suformuoti <b>nenaudojamas BP (bet kokioje kombinacijoje)</b>	DS	Taip	

Ar galima nurodyti kitą segmentą? - jei taip vadinasi naudojant segmento keitimo prefiksą prieš komandos operacijos kodą galima pakeisti defaultinį segmentą kitu, jei ne segmento keitimo prefiksas tuo atveju komandos vykdymo visiškai neįtakoja.

Operando atmintyje efektyvaus adreso formavimui gali būti panaudoti BX, BP, SI, DI registrų kombinacijos ir poslinkis. Kokia kombinacija naudojama, nurodo adresacijos baido r/m dalis (nebent mod=11, tada operandas yra ne atmintyje, kažkuris iš procesoriaus registrų).

Komandos paėmimas vykdymui baigęs vykdyti einamąją komandą procesorius eis vykdyti komandos, kurios mašininis kodas yra absoliučiu adresu **CS:IP**.

Segmentas pagal default – parodo, koks segmentas naudojamas, kai nėra jokio jį galinčio pakeisti prefikso, arba prefiksas yra, bet tuo atveju jis neturi įtakos.

## **Adresavimo būdai**

Adresavimas skirstomas į tiesioginį ir netiesioginį. Šis skirstymas ypač svarbus valdymo perdavimo komandose, todėl apie tai plačiau aprašyta prie valdymo perdavimo komandų.

**Tiesioginis** – adresas nurodomas iškart po operacijos kodo

**Netiesioginis** – adresas nuskaitomas iš atminties.

### **Tiesioginio atvejai:**

- Adresacijos baitas
- **Santykinis** – po operacijos kodo (OPK) nurodomas vieno arba dviejų baitų poslinkis, vieno baito poslinkis turi būti išplėstas iki 2 baitų naudojant plėtimo pagal ženklą taisyklę. Dviejų baitų poslinkis nurodomas tokia tvarka:
  - poslinkio jaunesnysis baitas
  - poslinkio vyresnysis baitas
- **Absolutus** – po operacijos kodo (OPK) nurodomos segmento ir efektyvaus adreso reikšmės, 4 baitai, kurie pateikiami tokia tvarka:
  - efektyvaus adreso jaunesnysis baitas
  - efektyvaus adreso vyresnysis baitas
  - segmento registro jaunesnysis baitas
  - segmento registro vyresnysis baitas

### **Netiesioginio atvejai:**

- **Santykinis** (2baitų efektyvus adresas imamas iš nurodytos atminties vietos, tokiu pat eiliškumu)
  - efektyvaus adreso jaunesnysis baitas
  - efektyvaus adreso vyresnysis baitas
- **Absolutus** (4baitai iš nurodytos atminties vietos imami tokia pat tvarka, kaip ir tiesioginiam absoliučiam)



## Plėtimo pagal ženklą taisyklė

**Baito plėtimas iki žodžio** – jei nenurodyta kitaip, vadinasi poslinkis viename baite yra skaičius su ženklu. Tuomet plečiant jį iki dviejų baitų papildomam vyresniame baite reikia surašyti 1tukus arba 0iukus. Priklausomai nuo to koks buvo ženklo bitas (pats pirmas kairėj pusėj).

0000 0001 => 0000 0000 0000 0001  
0111 0111 => 0000 0000 0111 0001  
1000 0000 => 1111 1111 1000 0000

Šešioliktainė sistema dažniausiai naudojama žodžiams sudėti, arba atimti. Svarbu žinoti, kad norint skaičių turimą viename baite užrašyti tokiu pat skaičiu žodyje reikia atsižvelgti į ženklą, o kada ne.

Ši taisyklė taikoma tada, kai prie žodinio registro ar efektyvaus adreso tenka pridėti vieno baito poslinkį, tada turime suvienodinti abiejų dėmenų ilgius (baitą išplėsti iki žodžio išlaikant jo ženklą)

Keletas pavyzdžių:

IP=1234h, poslinkis 54h

Tada naujas IP

IP=1234+0054=1288h

Tačiau, jei **viename baite** poslinkis yra 80h arba daugiau, t.y. vyriausias bitas yra vienetas, tada skaičiuojant poslinkį, turime prisirašyti FF (8 vienetukų seka).

Tada naujas IP

IP=1234h+FF80h=111B4h

IP sudaro 4 šešioliktainiai skaitmenys, todėl papildomą vienetuką ignoruojame.

Svarbu atsiminti tai, kad išplečiant vieno baito skaičių iki 2 baitų skaičiaus **BŪTINA** atsižvelgti į ženklą:

00h=0000h

7Fh=007Fh

80h=FF80h

FFh=FFFFh

Išskyrus išimtinius atvejus, tokius, kaip XLAT komanda (tuomet AL laikomas skaičiumi be ženklo)

# Stekas

Darbai su steku skirtos dvi komandos.

## **PUSH** ir **POP**

SF registro keitimui ir pasidėjimui į steką naudojamos atskiros assemblerinės mnemonikos

**PUSHF** (Push Flags) reiškianti PUSH SF

**POPF** (Pop Flags) reiškianti POP SF

SS registras rodo į steko segmento pradžią (skaičius nurodytas paragrafais, skaičiuojant absoliutų adresą dauginamas iš 10h). SP registras rodo poslinkį nuo steko segmento pradžios. SS:SP sąryšio jokie segmento keitimo prefiksai neįtakoja.

Naudojant **PUSH** komandą žodinio registro turinys, arba 2 baitai iš nurodytos atminties vietos yra įdedami į steką, **prieš tai SP sumažinus** 2 vienetais. (viršijus FFFF vėl skaičiuojama nuo 0000)

Naudojant **POP** komandą 2 baitai iš steko patalpinami į nurodytą atminties vietą arba į žodinį registrą, **po to SP padidinamas** 2 vienetais. (peržengus 0000 vėl skaičiuojama nuo FFFF stekas yra ciklinis)

Komandų atliekami veiksmai:

### **PUSH:**

1.  $SP = SP - 2$ ; (sekančiuose veiksmuose naudojamas naujai gautas SP)
2. mov SS:SP, jaunesnysis baitas (registro, arba pirmas pirmas baitas iš atminties)
3. mov SS:(SP+1), vyresnysis baitas (registro, arba sekantis po jaunesniojo iš atminties)

### **POP:**

1. mov jaunesnysis baitas, SS:SP
2. mov vyresnysis baitas, SS:(SP+1)
3.  $SP = SP + 2$ ;

### **Pavyzdys:**

(kad ir h raidės nėra, tai vis tiek šiuo atveju, kaip ir egzamino užduotyse reikš šešioliktainius skaičius)  
Turime SP=1234;

AX=75F4; BX=7894; CX=DEAD;

### **Atliekam veiksmą PUSH AX.**

Naujas SP=1232

AX, BX, CX nesikeičia

#### **Pradinė steko būseną:**

>>>>>>

SP	Baito reikšmė
1230	05
1231	40
1232	11
1233	36
<b>1234</b>	7C

SP	Baito reikšmė
1230	05
1231	40
<b>1232</b>	<b>F4</b>
1233	<b>75</b>
1234	7C

Po to atlikę **POP CX** pakeistume

CX, SP reikšmes, bet ne steko

turinį (IP registras vykdant

komandas keičiasi visuomet).

Tada vėl būtų SP=1234h, o nauja

CX reikšmė CX=75F4

**Pavyzdys** parodytas su registro pasidėjimu į steką (PUSH), iš tikrųjų galima pasidėti ir kokį nors elementą iš atminties. Tuomet **segmentas:EA** rastas baitas yra laikomas jaunesniu, **segmentas:(EA+1)** baitas laikomas vyresniu. Tas pats galioja ir pasiimant žodį iš steko (POP).

# Status Flag registras (SF)

## Kas yra SF registras?

Atliekant įvairias komandas visuomet keičiama IP registro reikšmė. Tačiau atskirais atvejais (ypač atliekant aritmetines ir logines komandas) keičiamas Status Flag registras. Atskiri jo bitai nurodo tam tikrus požymius. Šis registras dar vadinamas „procesoriaus būsenos registru“.

Registras yra žodžio (16 bitų) dydžio.

Reikia žinoti šių bitų eiliškumą, ir kokia kiekvieno iš jų prasmė (**neužpildyti langeliai** reiškia, kad tie bitai yra nenaudojami jokiame požymiu saugoti, vykdant komandas yra nekeičiami).

Išskyrus POPF komandą, kuomet iš steko paimama nauja SF registro reikšmė, iš naujo nustatomi visi bitai, net ir nenaudojami). Taip pat SF reikšmės paėmimas vykdomas ir baigus vykdyti pertraukimo apdorojimo programą (su IRET komanda)

## SF bitų reikšmės

Kiekvienas konkretus bitas vadinamas „flagu“, gali turėti reikšmę 0 arba 1.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

**Rezultato laukas** – tai yra rezultatas atlikus loginį arba aritmetinį veiksma su dviem operandais. Rezultato laukas yra lygiai tokio paties dydžio, kaip ir operandai, jei veiksmai atliekami su žodžiais, tai rezultato laikas yra žodis, jei baitais – baitas. **Carry Flag'o bitas NEPRIKLAUSO rezultato laukui.**

**Žaliai pažymėtieji** bitai kontroliuoja procesoriaus būseną. Vykiant aritmetines/logines komandas jie nesikeičia.

### Kam jie naudojami ir ką jie reiškia:

Flagas	Pavadinimas	Naudojimas ir reikšmė
<b>CF</b>	<i>Carry Flag</i> „pasiskolinimo/pernešimo požymis“	Aritmetinėse operacijose atstoja papildomą bitą rašomą kairiau vyriausiojo rezultato lauko baito. Baitų/žodžių sudėties atveju reiškia papildomą už rezultato lauko išeinantį bitą „mintyj“, atimties atveju reiškia pasiskolinimą iš už rezultato lauko ribų. Galima suprasti ir kitaip. Jei abu skaičiai yra skaičiai be ženklų iš intervalo [0;255] dirbant su baitais ir [0;65535] dirbant su žodžiais. Tai rezultatui nebetelpant tame intervale CF=1, kitu atveju CF=0 Atliekant logines komandas veikiančias kiekvienam bitui atskirai (AND, OR, XOR) CF=0
<b>PF</b>	<i>Parity Flag</i> „lyginumo požymis“	Ar vienetukų kiekis rezultato lauko <b>jauniausiame bайте</b> yra lyginis? (jei jų nėra nė vieno, laikoma, kad lyginis). Jei taip, PF=1 Jei ne PF=0
<b>AF</b>	<i>Auxilliary Carry Flag</i> „papildomas pasiskolinimo/pernešimo požymis“	Reiškia pasiskolinimą arba pernešimą iš jaunesniojo į vyresnįjį pusbaitį. Jei pernešimas arba pasiskolinimas toje vietoje įvyksta AF=1 Jei ne AF=0
<b>ZF</b>	<i>Zero Flag</i> „nulio požymis“	Ar rezultato laukas yra sudarytas vien iš nulinių bitų? Taip, ZF=1 Ne, ZF=0



# Programavimas assembleriu

## Kas nagrinėjama šiame skyriuje?

Šio skyriaus tikslas vaizdžiai ir paprastai aprašyti programavimą assembleriu. Pradedant nuo elementarių pagrindinių sąvokų, pačios pirmos programos pereinama prie sudėtingesnių komandų ir uždavinių nagrinėjimo.

Visame šiame skyriuje daroma prielaida, kad naudojate TASM (Turbo Assembler) assembleriu.

Tačiau naudodami ir kitus (pavyzdžiui NASM, FASM ar pan.) galite iš šio skyriaus pasisemti kokių nors idėjų, suprasti pagrindus, o tada labiau gilintis į pasirinkto assemblerio subtilybes.

## Kaip susikompiliuoti, pasileisti kodą

Pirmiausia turite gauti keletą vykdomųjų failų, kurie bus naudojami jūsų assembleriu parašytam kodui išversti į mašininį kodą (.com arba .exe) failą.

*\* - reiškia bet kokią failo pavadinimą, kokią jūs pasirinksite programai pavadinti*

Exe failai bus tokie:

**TASM** – Turbo assembler, jūsų assemblerio failus išvers į tarpinį variantą tarp assemblerinio ir mašininio kodo. (\*.obj failas)

**TLINK** – Turbo linker. Tarpinį failą išvers į mašininį kodą. Jei nenurodėte jokių papildomų parametrų, tai tikrai gausite \*.asm failą (aišku jei assemblerinį kodą parašėte be klaidų)

**TD** – Turbo debugger – Derinimo priemonė, galimybė žingsnis po žingsnio matyti kaip vykdomos jūsų parašytos programos.

Programos bus vykdomos per cmd konsolę. Ją pasileisti „Windows“ operacinėje sistemoje galite Start (pradėti) > Run... (vykdyti) > įveskite „cmd“ ENTER.

Visus **TASM**, **TLINK**, **TD** failus susidėkite į kokią nors atskirą aplanką (folderį).

Assemblerio programos kodą irgi laikysite šiame folderyje.

Pastaba: yra galimybė pasidaryti, kad **TASM**, **TLINK** ir **TD** bus kokiam nors atskirame folderyje, o programa \*.asm bet kur, ir eis kompiliuoti.

Tarkim, kad laikome **TASM**, **TLINK**, **TD** failus tame pačiame folderyje kaip \*.asm failą.

Tai folderio turinys turėtų atrodyti taip:

- **TASM.EXE**, **TLINK.EXE**, **TD.EXE**, my\_prog.asm

Konsolėje nurodote kokiam folderyje laikote failus įvesdami

cd **folderis**

(čia **folderis** rodo pilną kelią iki jūsų aplanko)

Jei viską atlikote gerai įrašę žodį **TASM** turėtume gauti krūvą teksto, kur aprašyta kokiais parametrais paleidus **TASMą** kaip jis elgsis.

Kompiliavimą (transliavimą) atliekate konsolėje įvesdami:

tasm my\_prog (gaunate my\_prog.obj failą)

tlink my\_prog (gaunate my\_prog.exe failą)

mano\_prog (matote programos vykdymo rezultatą)

Jei tasm arba tlink išmetė klaidų, reiškia kažkas jūsų kode negerai (arba jo nėra).

**Nesuderinamose aplinkose (64-bit, Linux ir pan.)** **TASM** naudokite per DosBox emuliatorių

**Programų failų vardai** (atmetus galūnes) turi neviršyti 8 simbolių.

# Pirmoji programa

Skyrelyje aprašomi pagrindiniai dalykai, kuriuos būtina žinoti/suprasti norint parašyti pirmąją programą. Pradedame nuo paprastos programos į ekraną išvedančios tekstą „Labas pasauli“ ir ją šiek tiek patobuliname, aiškinamės kodėl ji veikia būtent taip, o ne kitaip.

Skyrelio tikslas – padėti suprasti kaip parašyti assembleriu paprastą programėlę, kad būtų galima pirmąją parašyti savarankiškai.

## Prieš pradedant rašyti pirmą programą

Prieš rašant kodą assembleriu reikia išsiaiškinti pagrindines komandas, bei sistemą, su kuria dirbame. Plačiau apie tai sistemą su kuria dirbama aprašyta „Architektūros ypatybės“ konspekto skyriuje, o šiame apsiribosime minimaliu žinių kiekiu reikalingu bent paprasčiausiai programai suprasti. Laikoma, kad suprantate konvertavimą tarp skaičiavimo sistemų, žinote kas yra baitas/bitas ir pan.

Mūsų atveju sistema susideda iš atminties ir procesoriaus (šios dalys esminės!).

Atmintis turi  $2^{20}$  baitų. Programuojant assembleriu jie visi yra tiesiogiai pasiekiami.

Mikroprocesorius savyje turi 14 registų – 2 baitų dydžio laukų veikiančių panašiai kaip atmintis, tik sparčiau.

Šie registrai yra tokie:

**Darbiniai:** AX, BX, CX, DX

**Segmentiniai:** CS, DS, ES, SS

**Indeksiniai:** SI, DI

**Kiti:** BP, IP, SP, SF

Programuojant assembleriu prisireikia naudotis jais visais, tačiau rašant paprastas programas apsiribokime tik iki 4.

Plačiau paanalizuokime darbinius registrus, kurių mums labiausiai reikės.

**AX** – accumulator

**BX** – base

**CX** – counter

**DX** – data

Jų pavadinimų pirmos raidės eina abėcėlės tvarka, todėl bus lengva prisiminti jų pavadinimus.

Kiekvienas šių registų užima 2 baitus (vieną žodį) t. y. 16 bitų.

Kiekvienas **darbinis** registras susideda iš dviejų vieno baito registų:

AX susideda iš AH, AL

BX iš BH, BL

CX iš CH, CL

DX iš DH, DL

T.y. \*X registre \*H vadinama vyresniuoju baitu, o \*L jaunesniuoju. (kur vietoj žvaigždutės raidė A, B, C arba D)

Tarkim jei **AX** registro reikšmė yra ACCDh, tai **AH**=ACh, **AL**=CDh

**Komentarai** gali būti rašomi kiekvienoje eilutėje. Tam naudojamas kabliataškis ;

Tai kas konkrečioje eilutėje eina po kabliataškio kompiliatoriaus yra ignoruojama, bet gali būti naudinga programuotojui skaitančiam ir bandančiam suprasti kodą.

Segmentiniai registrai programos vykdymo pradžioje automatiškai rodys į tų segmentų pradžias atmintyje. **CS** – kodo, **DS** – data (duomenų), **SS** – steko, **ES** – extra (papildomo duomenų).

Kiti išskyrus **SF** rodys atitinkamą poslinkį juose.

**SF** registras rodo procesoriaus būseną ir paskutinio matematinio/loginio veiksmo rezultato požymius.

## MOV ir INT komandos

Paprasta ir asembleryje dažniausiai naudojama komanda yra **MOV** (nuo žodžio move)

MOV komanda visada turi 2 operandus atskiriamus kableliu.

Kitos komandos turės vieną, du arba nė vieno operando (priklauso nuo to kokia komanda)

Bendruoju atveju užrašoma: MOV op1, op2

kur op1 yra pirmasis operandas, o op2 antrasis

Komandos atliekamas veiksmas yra antrojo operando reikšmės įrašymas į pirmajame operande nurodytą vietą.

Tarkim Pascal kalbos atveju turėtume op1:=op2;

### MOV komandų sudarymo pavyzdžiai:

**MOV bx, ax** ; bx:=ax; bx registrai priskiriama ax registro reikšmė.

**MOV ch, bl** ; atlikome tą patį tik su vieno baido registras. ch įgauna bl reikšmę

**MOV ch, dx** ; neveiks, nes nesutampa operandų dydžiai (pirmas vieno baitas, antras dviejų baitų)

**MOV cl, 15** ; cl reikšmė dešimtainėje sistemoje taps 15 (šešioliktainėje 0Fh)

**MOV cl, 15h** ; cl reikšmė šešioliktainėje sistemoje taps 15 (dešimtainėje 21)

**MOV dh, 101b** ; dh reikšmė dvejetainėje sistemoje bus 101 (dešimtainėje 5, šešioliktainėje 5)

**MOV 15, cl** ; neveiks, nes konstantai (betarpiškam operandui) nieko priskirti negalima

**MOV bl, '@'** ; bl registrai bus priskiras @ simbolio ASCII kodas

**MOV bl, 64** ; tas pats kas su @, tik jau parašėme šio simbolio ASCII kodą dešimtainėje sistemoje

**MOV bl, 40h** ; tas pats su @, tik parašytas simbolio ASCII kodas šešioliktainėje sistemoje

**MOV ds, es** ; iš esmės viskas teisinga, tačiau tik su dalimi registrų galima atlikti priskyrimo operacijas (taip pat tiesiogiai nėra pasiekiamas IP registras) – techniškai nenumatyta galimybė iš es tiesiai nusiųsti reikšmę į ds

**MOV bl, CCh** ; neveiks, nes rašant konstantas jos turi prasidėti skaičiumi

**MOV bl, 0CCh** ; į bl registrą bus įrašyta šešioliktainė reikšmė CC

**MOV ah, ch, dh** ; netinka operandų skaičius (3>2)

**MOV ah** ; netinka operandų skaičius (1<2)

### Konstantų (betarpiškų operandų užrašymo būdai):

- Simboliu kabutėse: tarkim '@', bus suprantamas kaip simbolio @ kodas (64 dešimtainėje)
- Dešimtainiu skaičiumi: tarkim 15, 15d (d reiškia decimal)
- Dvejetainiu skaičiumi: tarkim 10010b (b reiškia binary)
- Šešioliktainiu skaičiumi tarkim 5Ah (h reiškia hexadecimal). Jei pirmas simbolis nėra skaitmuo tarkim atveju ABh **rašytume priekyje nulį, nes turi prasidėti ne raide** 0ABh
- Aštuntainiu skaičiumi tarkim 77o (o reiškia octal)

Dar pirmai ką nors naudingo darančiai programai parašyti reikalinga ir kita komanda:

**INT** – nuo žodžio INTerrupt

Ši komanda iškviečia pertraukimo apdorojimo procedūrą.

Nesigilinant į detales: ją galima naudoti kaip funkciją ar procedūrą aukštesnio lygio programavimo kalbose.

INT komanda turi vieną operandą, tai visuomet yra koks nors vieno baito skaičius.

Rašant pirmą programą, mums prireiks naudotis DOS servisu. Tam kviesime **INT 21h**

21h reiškia, kad kviečiama 21h-oji pertraukimo apdorojimo procedūra.

Ji bus naudojama atliekant tokius dažnai atliekamus veiksmus kaip išvesti eilutę į ekraną, atsidaryti failą, uždaryti failą ir pan.

Iš esmės INT komanda tam ir skirta, kad tokie įprasti veiksmai būtų ne kaskart suprogramuojami gaištant daug laiko, o tiesiog iškviečiami padavus atitinkamus parametrus (lyg funkcijai ar procedūrai aukštesnio lygio kalbose).

Argumentai paduodami priskiriant reikšmes atitinkamiems iš anksto apibrėžtiems registrams.

AH registro reikšmė nurodys, kokia INT 21h sub-funkcija mums reikalinga

- **Išvesti eilutę iš duomenų segmento į ekraną**

AH=09

DX=pirmojo simbolio poslinkis nuo duomenų segmento pradžios

Grąžina: AL=24h (? - bendru atveju taip, reiktų patikrinti ar galioja TASMui)

Ekране išspausdina simbolių seką pradedant adresu DS:DX, baigiant \$ simboliu. (šis nespausdinamas)

- **Išspausdinti vieną simbolį į ekraną**

AH=02

DL=simbolio ASCII kodas

- **Baigti programos darbą**

AH=4Ch

AL=išėjimo kodas (galima ir nenaudoti, tačiau realiam gyvenime būna atvejų, kai programa baigdama darbą grąžina skaičių)

Svarbu atsiminti, kad po reikšmių priskyrimo registrams būtina eilutė yra **INT 21h**

## **Bendra programos struktūra**

Taigi jau dabar galime labiau įsigilinti į programos rašymą

Visos TASM kompiliatoriui parašytos programos atrodys taip:

**.model small**

**.stack 100h**

**.data**

**MOV ax, @data**

**MOV ds, ax**

;čia bus programos data segmentas

**.code**

;čia bus programos kodo segmentas

**END**

**.model small** – apibrėžia programos dydį. pakanka žinoti, kad šis užrašas kompiliatoriui pasako, jog programa yra mažos apimties ir kodo segmentas tilps į 64 kilobaitus, o visa programa į 128kb. Visoms programoms turėtų pakakti ir šio „atminties modelio“. Bei **svarbu teisingai užrašyti šią eilutę, nes tokią klaidą kompiliatorius gali „praleisti pro akis“ ir rodyti, kad pilna klaidų tolesniame kode!**

**.stack 100h** – pasakome, kad programai užteks 256 baitų steko (mažoms programoms tai yra per akis, tačiau rašant programas kurios kviečia daug CALL komandų kartais verta šią reikšmę padidinti, bet reikšmė turi būti mažesnė už 10000h)

**.data** – žodis apibrėžiantis, kad toliau einančios baitų reikšmės priklauso data segmentui

**.code** – žodis apibrėžiantis, kad toliau einančios baitų reikšmės/komandos priklauso data segmentui

**MOV ax, @data**

**MOV ds, ax**

Eilutės, kuriomis turės prasidėti kiekviena TASMui rašyta tvarkinga programa.

Kompiliatorius kodą sukompiliuotų ir be šių komandų, bet programos vykdymo rezultatai būtų nenusipėjami.

Iš esmės šios eilutės reikalingos tam, kad DS registras turėtų tinkamą reikšmę ir tikrai rodytų į duomenų segmento pradžią. Po @data slepiasi skaičius, kuris turi būti priskirtas DS registrui, kiekvieną kartą kompiliuojant kodą jis gali skirtis, todėl pasirinkta ne kaskart vargti skaičiuojant ir įrašyti skaičių patiems, o tiesiog užrašyti taip.

Viso to negalima rašyti tiesiog **MOV ds, @data** nes asembleryje nėra numatyto būdo segmentiniam registrui priskirti konstantos reikšmę.



**END** – užrašas kiekvienos programos TASMui paskutinėje eilutėje. Tai yra tiesiog direktyva kompiliatoriui, pasakanti, kad čia yra programos kodo segmento pabaiga

## **„Labas pasauli“ programa**

Taigi, pagaliau jau turint šių dalykų supratimą bus galima labiau pasigilinti į visiškai paprastą programą, kuri nieko kito nedaro, tik išveda į ekraną tekstą „Labas pasauli“

```
.model small  
.stack 100h  
.data  
    Pranesimas db 'Labas pasauli',13,10','$'  
.code  
    MOV ax, @data  
    MOV ds, ax  
  
    MOV ah, 9  
    MOV dx, offset Pranesimas  
    INT 21h  
  
    MOV ah, 4Ch  
    INT 21h  
END
```

Jau šia programa parodoma, kad visas kodas turės būti kodo segmente, o pranešimai į ekraną ir kintamieji saugomi data segmente.

Data segmente:

```
.data  
    Pranesimas db 'Labas pasauli',13,10','$'
```

Čia žodžis Pranesimas pasižymime data segmento vietą, kurioje yra mūsų pranešimas. Galėtume rašyti ir tarkim

```
.data  
    Pranesimas db 'Labas p'  
                db 'asauli',13,10','$'
```

ir turėtume tą patį rezultatą.

**db** reiškia **define byte**. Reiškia toje eilutėje mes tiesiog surašome konkrečias baitų reikšmes. Žodžiai pateikti tarp viengubų kabučių pasako kompiliatoriui, kad duotas reikšmes suprasti kaip simbolius (jie mašininiame variante bus išrašyti kaip ASCII kodai). Skaičiai be kabučių bus tokie patys parašyti ir mašininiame kode. Šiuo atveju turime 13,10 kurie reiškia naują eilutę (Line Feed+Carriage Return – iš esmės pereiti į naują eilutę ir žymeklį nustumti į eilutės pradžią). Tarkim C kalboje turėtume simbolį **\n**

**offset** Pranesimas – formalus užrašas pasakantis, kad šioje vietoje bus įrašytas skaičius reiškiantis Pranesimas poslinkį nuo duomenų segmento pradžios.

Kai kuriuose naujesniuose assembleriuose, kaip kad NASM **offset** yra atsisakyta, nes realiai tai tik žymėjimo aiškumo reikalas. Bet kokiu atveju Pranesimas rodo ne kintamojo pavadinimą, o pažymi atminties vietą.

Šiuo atveju vietoj **offset** Pranesimas galėtume parašyti tiesiog nulį, tačiau vėliau pakeitę data segmento turinį, kai tarkim Pranesimas prasidės ne nuo DS:0000h adreso, o nuo DS:0005h turėtume vietoj nulio įrašyti 5.

## Žymės (label) ir CMP, JMP komandos

Paprastai naudingos, gyvenime naudojamos programos atlieka ne tik tokius veiksmus kaip „Labas pasauli“, bet ir reaguoja į sąlygas, esant skirtingiems duomenim rodo skirtingus rezultatus.

Tam mums prireiks panaudoti žymes (angl. labels)

Taip yra vadinami tam tikri pavadinimai suteikiami reikiamoms kodo vietoms. Kaip jau naudojome žymes data segmente, pasižymėti kur prasideda Pranesimas, tai kodo segmente iš pažiūros žymėsime kodo fragmentus, bet realiai mes pasižymėsime tik konkrečią vietą, o kodo vykdymas pernelyg nesikeis.

Žymes kodo segmente rašysime žodžiu eilutės pradžioje ir po jo dėsime dvitaškį

**Pavyzdžiai:**

**pradzia:**

**mov:** ; neveiks, nes reikia naudoti nerezervuotus assemblerio kalbos žodžius, mov rezervuotas mov komandai

**baigti\_darba:**

**pranesimo\_isvedimas:**

**Programos pavyzdys:**

Perrašome prieš tai jau naudotą programą panaudodami žymes

**.model small**

**.stack 100h**

**.data**

Pranesimas **db** 'Labas pasauli',**13,10**','\$'

**.code**

**MOV ax, @data**

**MOV ds, ax**

**pranesimo\_rodymas:**

**MOV ah, 9**

**MOV dx, offset Pranesimas**

**INT 21h**

**programos\_uzdarymas:**

**MOV ah, 4Ch**

**INT 21h**

**END**

Šioje programoje nieko nepasikeičia nuo to, kad panaudojame žymes.

Toliau, esminė assemblerio suteikiama galimybė yra **JMP komanda** ir kodo vykdymas pagal sąlygas. JMP komanda turės vieną operandą – žymę. Vykdydamas JMP komandą procesorius eis vykdyti ne sekančios iš eilės komandos, o pirmosios pateiktos po žymės vardo. Šiuo atveju, nebus tikrinama jokia sąlyga.

Pavyzdžiui parašę besąlyginio valdymo perdavimo komandą

**JMP programos\_uzdarymas**

pereitume vykdyti kodo esančios po *programos\_uzdarymas* žymės.

Tarkim nebenorime, kad mūsų programa išvestų „Labas pasauli“ pranešimą, o iškart baigtų darbą.

Tam pavyzdyje pateiktą JMP komandą įterptume dar prieš *pranesimo\_rodymas*

Tada kodas būtų vykdomas iki šios komandos, tada peršokta vykdyti po *programos\_uzdarymas* einančių eilučių.

Dažniausiai, didesnės apimties programose JMP komanda, kaip ir MOV ar INT panaudojamos daug kartų.

Sąlygų (kaip kad if sakiniai aukštesnio lygio kalbose) rašymas dažniausiai sunkiai įvykdomas be CMP komandos.

**CMP** (nuo žodžio *Compare* – lyginti).

Komanda turi du operandus.

Ji daro **beveik** tą patį, ką kita komanda **SUB**, kuri iš pirmo operando atima antrojo reikšmę.

**SUB** op1, op2 ( reiškia  $op1:=op1-op2$ ; ) keičia pirmojo operando reikšmę ir SF registro aritmetinius/loginius flagus.

**CMP** op1, op2 daro tą patį, tik nekeičia pirmo operando reikšmės. Keičia tik SF registro aritmetinius/loginius flagus.

Kas iš to?

Tarkim mus domina ar abu operandai yra vienodų reikšmių.

Jei taip būtų vadinasi  $op1=op2$ , iš čia  $op1-op2= op1-op1 = 0$

reiškia, kad atėmus skaičių iš tokio paties rezultatas bus nulis. Tada SF registre ZF požymis (Zero Flag) bus vienetas.

Tam galime pasinaudoti

**JZ** – Jump if Zero. (veikia kaip JMP bet vykdys tik tada, jei flagas ZF=1)

**JNZ** – Jump if Non Zero (jei flagas ZF=0)

Kiti naudingi valdymo perdavimai:

**JA** – Jump If Above

**JB** – Jump If Below

(tas pats gali turėti kelias notacijas, tarkim JZ ir JE reiškia tą patį)

Sekančiame puslapyje pateiktas programos pavyzdys (taip pat ir komandinės eilutės parametrų skaitymo atvejis).

## ***Komandinės eilutės parametrų skaitymas***

Programą per tasm paleidžiame tiesiog įrašę jos vardą tarkim „**progr**“

Tačiau gali reikėti parašyti programą, kuriai po programos vardo padėjus tarpą (reikia programuoti taip, kad galėtų būti ir daug tarpų!) programa turi išvesti pagalbos pranešimą, ar pan.

Viskas, ką žmogus paleidžiantis programą surašo po programos vardo yra išsaugoma pradedant adresu ES:81h (Extra segmente nuo 81h baido).

Tam patogų pasinaudoti adresavimu segmentų viduje.

Pavyzdžiui:

**MOV bx, 81h**

**MOV ax, es:[bx]** (dviem baitam) arba tarkim **MOV al, es:[bx]** (vienam baitui)

laužtiniuose skliaustuose parašytas skaičius arba reiškinys yra skirtas gauti efektyviam adresui (šiuo atveju imama bx reikšmė). **es** reiškia, kad bus naudojamas extra segmentas, nes būtent jame atsiduria žmogaus įrašyti komandinės eilutės parametrai.

***NASM Linuxuose atveju:*** komandinės eilutės parametrai nuskaitomi visai kitaip. Paleidus programą prireiks komandinės eilutės parametru traukti iš steko.

Pirmas ištrauktas skaičius tarkim komanda **POP eax** reikš parametrų atskirtų tarpais skaičių (šiuo atveju jis atsidurs eax registre). Bendru atveju **e\*\*** registras (kur vietoj žvaigždučių kažkokios raidės) yra 32 bitų!

Visada bus bent vienas, tai yra pats programos vardas. Toliau iš steko galime išsitraukti kiekvieno iš parametrų poslinkius data segmente.

Tarkim paimam **POP esi** pirmo parametro poslinkį data segmente. (tai bus pačios programos vardas). Toliau atlikdami traukimą iš steko paimsime antrą, trečią ir kiek bebūtų tų parametrų – pačių parametrų kiekį išsitraukėme iš steko patį pirmą.

## Pavyzdys:

;Programa pasako ar AL registru priskirta reiškmė yra 17 dešimtainė

;Šiuo atveju pagalbos pranešimas išvedamas tik tada, jei po programos vardo buvo padėtas tarpas, tada parašyta ?/ (po to gali būti belekas) ir spaudžiamas ENTER – bendru atveju reikia padaryti sudėtingiau, kad į tarpų kiekį būtų nereaguojama

**.model small**

**.stack 100h**

**.data**

Pran1 **db** 'AL yra 17',**13,10,'\$'**

Pran2 **db** 'AL nera 17',**13,10,'\$'**

Pagalba **db** 'Si programa pasako ar AL registre esanti reiksme yra 17.',**13,10,'\$'**

**.code**

**mov ax, @data**

**mov ds, ax**

**mov bx, es:[82h]** ;Parametrų iš komandinės eilutės skaitymas! Jei programa paleista su parametru ?/ išves pagalbos pranešimą – imame nuo 82h nes praleidžiam es:81h esantį vienintelį tarpą – šis pavyzdys yra tik parodyti kaip „maždaug“ veikia komandinės eilutės parametrų skaitymas.

Teisingas pavyzdys, kaip reikia padaryti iš tikrųjų yra tarkim ši programa [\[spausti nuorodą\]](#)

**cmp bx, '?'** ;Pirmas baitas iš ES:82h skaitomas į jaunesnįjį BL, kitas į BH todėl gaunamas sukeitimas!

**je** pagalbos\_pran

**mov al, 17h**

**cmp al, 17**

**je** Isvesk\_pran1

**jne** Isvesk\_pran2

Isvesk\_pran1:

**mov ah, 9**

**mov dx, offset Pran1**

**int 21h**

**jmp** baigti\_darba

Isvesk\_pran2:

**mov ah, 9**

**mov dx, offset Pran2**

**int 21h**

**jmp** baigti\_darba

pagalbos\_pran:

**mov ah, 9**

**mov dx, offset Pagalba**

**int 21h**

**jmp** baigti\_darba

baigti\_darba:

**mov ah, 4Ch**

**int 21h**

END

# Darbas su įvairaus formato skaičiais

## Koprosesoriaus (8087) realiųjų skaičių formatai (float'ai)

8087 koprosesoriuje naudojami 3 pagrindiniai realiųjų skaičių formatai. (IEEE-754 standartas)

**Trumpas realus** (**keturiuose** baituose)

<i>S (1b)</i>	<i>charakteristika (8b)</i>	<i>mantisė (23b)</i>
---------------	-----------------------------	----------------------

- 1 ženklo bitas (0 reiškia pliusą, 1 reiškia minusą)
- 8 bitai charakteristika (eilė + 127<sub>10</sub>) 127<sub>10</sub>=**7Fh**
- 23 bitai mantisė
- Skaičių būtina privesti prie normalizuotos formos, kuri atrodo taip:  
$$(-1)^s * 2^{\text{eilė}} * 1, \text{mantisė}$$

**Ilgas realus** (**aštuoniuose** baituose)

<i>S(1b)</i>	<i>charakteristika (11b)</i>	<i>mantisė (52b)</i>
--------------	------------------------------	----------------------

- 1 ženklo bitas (0 reiškia pliusą, 1 reiškia minusą)
- 11 bitų charakteristika (eilė + 1023<sub>10</sub>) 1023<sub>10</sub> = **3FFh**
- 52 bitų mantisė
- Skaičių būtina privesti prie normalizuotos formos, kuri atrodo taip:  
$$(-1)^s * 2^{\text{eilė}} * 1, \text{mantisė}$$

**Vidinis realus** (**dešimtyje** baitų)

<i>S (1b)</i>	<i>charakteristika (15b)</i>	<i>i bitas (1b)</i>	<i>mantisė (63b)</i>
---------------	------------------------------	---------------------	----------------------

- 1 ženklo bitas (0 reiškia pliusą, 1 reiškia minusą)
- 15 bitų charakteristika (eilė+16383<sub>10</sub>) 16383<sub>10</sub> = **3FFFh**
- 1 i bitas, jis parodo koks skaičiukas yra prieš kablelį po kurio eina mantisė (šiam formate nebūtina sudaryti normalizuotos formos, bet patartina tai padaryti ir i bitą visuomet žymėti vienetu)
- 63 bitų mantisė
- Normalizuota forma atrodytų taip: 
$$(-1)^s * 2^{\text{eilė}} * 1, \text{mantisė (i bitas=1)}$$
- **N**ormalizuota forma: 
$$(-1)^s * 2^{\text{eilė}} * i, \text{mantisė}$$

Pastaba: Užtenka prisiminti normalizuotos formos užrašą kiek užima charakteristika, ir ar yra **i** bitas (jis yra tik 10ties baitų atveju). Tada tokį eiliškumą:

1. Ženklo bitas (visada tik vienas bitas)
2. Charakteristika, kuri esant nulinei eilei yra lygi: kairiausias bitas nulis visi kiti vienetai.
3. i bitas, jei šis yra (tik dešimties baitų atveju)
4. mantisė eina iki pat galo.

Mantisė yra nukerpama ir kas netilpo tiesiog neįsimenama. Iš tikrųjų kompiuteriuose esant bitui už mantisės lygiam vienetui vyksta apvalinimas (mantisė+1), apvalinant perpildžius mantisę vienu padidėja charakteristika, tačiau to žinoti nebūtina (galima ir neapvalinti nesukant sau galvos).

Sudarinėjant normalizuotą formą kai yra slankiojamas kablelis tai slenkant jį į kairę pusę charakteristika (ir eilė) didinamos, į dešinę – mažinamos. Šiame konspekte sprendžiamuose uždaviniuose laikysime, kad mantisė yra tiesiog nukerpama.

**Išimtis, kurią vertėtų žinoti** – jei charakteristika ir mantisė yra užildytos tik nuliais, tai skaičius yra nulis.

## Dešimtainio slankaus kablelio vertimas į šešioliktainį užrašą

Šis pavyzdys parodo kaip **keturiuose baituose** užrašyti dešimtainį skaičių **165,29** panašios taisyklės taikomos ir kitiems formatams (juose arba yra i bitas, arba tiesiog ilgesnė mantisė ir charakteristika saugoma didesniame bitų skaičiuje)

- Atskirai dirbame su sveikąja ir trupmenine dalimis

### SVEIKOJI DALIS:

- Pasiverčiame skaičių 165 į dvejetainę skaičiavimo sistemą  $165_{10}=1010\ 0101_2$
- Po vyriausio vieneto matome einančius 7 bitus, todėl jau 7 mantisės bitus jau turime.
- Reikia likusių  $23-7=16$ -ikos

### TRUPMENINĖ DALIS:

0,29 yra skaičiaus trupmeninė dalis

Daugindami \*2 imdami sveikąją dalį (rašysim mantisėj), o toliau vėl dirbdami tik su trupmenine dalimi tol kol rasime reikiamus 16 skaičiukų:

$$0,29 * 2 = 0,58$$

$$0,58 * 2 = 1,16$$

$$0,16 * 2 = 0,32$$

$$0,32 * 2 = 0,64$$

$$0,64 * 2 = 1,28$$

$$0,28 * 2 = 0,56$$

$$0,56 * 2 = 1,12$$

$$0,12 * 2 = 0,24$$

$$0,24 * 2 = 0,48$$

$$0,48 * 2 = 0,96$$

$$0,96 * 2 = 1,92$$

$$0,92 * 2 = 1,84$$

$$0,84 * 2 = 1,68$$

$$0,68 * 2 = 1,36$$

$$0,36 * 2 = 0,72$$

$$0,72 * 2 = 1,44$$

Vis tik yra ir kitas būdas surasti šiuos skaitmenis. Atliekame tą patį, tik vietoj 4 daugybos iš 2 veiksmų po vieną daugybos iš 16 veiksmų. Gautą sveikąją dalį užrašome kaip šešioliktainį skaičių, o rašant į mantisę išsiskleidžiame kiekvieną šešioliktainį skaitmenį iki keturių dvejetainių skaitmenų.

Daugybą tuomet tikslingiau atlikti ne mintinai, o stulpeliu.

Pvz. galėjome šiuo atveju daryti ir taip:

$$0,29 * 16 = 4,64 \quad (4=0100)$$

$$0,64 * 16 = 10,24 \quad (10=1010)$$

$$0,24 * 16 = 3,84 \quad (3=0011)$$

$$0,84 * 16 = 13,44 \quad (13=1101)$$

Gavome tuos pačius skaičius, per mažiau veiksmų, tačiau tam reikia mokėti greitai paversti skaičius nuo 1 iki 15 į keturis dvejetainius skaitmenis.

165,29 skaičius dvejetainiame pavidale (ne tikslus, o tik tiek, kiek telpa pagal formatą):

1010 0101, 0100 1010 0011 1101

Taigi turime:

ženklų bitas 0, nes skaičius teigiamas

eilė = +7, nes stumiame kablelį per 7 vietas į kairę, kad gautume normalizuotą formą

charakteristika = eilė + 127 = 7 + 127 = 134 (128+4+2, t.y. 1000 0110)

mantis pateikta geltonam fone

sudarome atsakymą:

antroje eilutėje surašyti šešiolyktainiai atitikmenys, kuriuos rašome į atsakymą

0100	0011	0010	0101	0100	1010	0011	1101
4	3	2	5	4	A	3	D

Atsakymas: 43 25 4A 3D

## Šešiolyktainio realaus skaičiaus vertimas į dešimtainį užrašą

Pavyzdžio nepateikiu, bet principas būtų toks (atvirkščias algoritmas pateiktam prieš tai), jis aprašytas 4 baitams trumpam realiam formatui, bet esant kitokiam formatui tiesiog galima pasikeisti atitinkamas reikšmes:

- Turimus šešiolyktainius skaičius pasiversti dvejetainiais
- Nustatyti ženklų bito, charakteristikos, mantisės reikšmes
- Pasirašyti 1, mantisė
- Charakteristiką pasiversti dešimtainiu skaičiumi, iš jos atimti 127. Gausime skaičiaus eilę
- Jei gauta eilė yra teigiama kablelį patraukti per tiek vietų į dešinę, jei neigiama tada į kairę
- Gautą skaičių pasiversti į dešimtainę sistemą (kiek reikia susiapvalinti, nes tikslų skaičių dažniausiai gauti atgal nepavyksta, nebent yra vos keli skaitmenys po kablelio)
- Parašyti prieš skaičių minusą, jei ženklų bitas lygus 1
- Užrašyti tai kaip atsakymą
- Pastaba: Dažniausiai verčiant į dešimtainį realųjį gausime nebetokį skaičių, todėl nebus galima vienareikšmiškai atlikti netgi palyginimo operacijų. (pvz užkodavę kokį 12,34 ir vėl atkodavę galime gauti kažką panašaus į 12,339999999999999, tuomet atliekant palyginimus reikia įsivesti skirtumą, į kurį bus nekreipiama dėmesio lyginant reikšmes)

# Egzamino užduočių pavyzdžiai su sprendimais

## Bendra informacija apie pateiktus sprendimus

Kiekvienas išspręstas uždavinys yra priskirtas kažkurai iš temų. Temos pavadinimas baigiasi laužtiniais skliaustais ir juose įrašytu skaičiumi. Tas skaičius reiškia kiekį, kiek tos temos uždavinių šiuo metu yra konspekte, gali būti, kad naujesnėse versijose jų rasite ir daugiau.

Sprendimai pateikti siekiant apimti kuo daugiau atvejų, ir dalyje jų yra aprašyti net ir tie veiksmai, kurių nereikia atlikti duotajame uždavinyje jį sprendžiant, bet į ką reiktų atsižvelgti, kai uždavinys atrodo panašus, bet iš tikrųjų kažkuri jo sprendimo dalis kažkiek skiriasi.

Kiekvieno uždavinio sprendimas sudalyti į tokias dalis: **uždavinio sąlyga**, **detalus sprendimas**, **uždavinio ypatybės** (kuo jis išsiskiria iš kitų, į ką reikia atsižvelgti), **atsakymas**.

## Adresavimas [2]

### UŽDAVINYS NR. 1

**Uždavinio sąlyga:** Registrų reikšmės yra DS=FE21, SS=3456, CS=CC31, ES=E341, BP=329A, BX=3675, SI=FA45, DI=F122. Apskaičiuoti operando absoliutą adresą pagal adresavimo baitą 6E.

Po adresavimo baito seka baitai: 89AB

#### Sprendimas:

Tikslas – operando absoliutaus adreso radimas, pagal adresavimo baitą.

Žinome adresavimo baitą 6E. Užsirašome šį skaičių dvejetainėje sistemoje: 01101110.

Iš to turime: mod=01, reg=101, r/m=110.

Operandą atmintyje parodo r/m, o pagal mod nustatome, koks poslinkis (mod nelygu 11, tai operandas atmintyje, o ne registras, be to tai pasakyta ir sąlygoje).

Pagal mod ir r/m nustatome, kad:

r/m=110, kai mod=01 yra BP+poslinkis.

Kadangi mod=01, reiškia poslinkis yra vieno baito, ir jį reikės išplėsti iki dviejų baitų pagal ženklą.

Efektyvaus adreso BP+poslinkis skaičiavimas:

BP=329A, vieno baito poslinkis yra 89. Kadangi 89 vyriausias bitas yra 1, plėsdami pagal ženklą turime poslinkį FF89. Pridedame jį prie BP:

$$\begin{array}{r|l} & 329A \\ + & FF89 \\ \hline 1 & 3223 \end{array}$$

Kadangi skaičiuojame operando efektyvą adresą todėl į papildomą vienetuką dėmesio nekreipiame (efektyvą adresą sudaro 4 šešiolyktainiai skaitmenys). EA=3223.

#### Kokį segmentą naudoti?

Jei būtų segmento keitimo prefiksas, segmentą nustatytume pagal jį.

Kadangi prefikso nėra, žiūrime ar kombinacijoje panaudotas BP registras?

Taip ---> Segmentas SS

Ne ---> Segmentas DS.

Šiuo atveju BP kombinacijoje buvo panaudotas (**BP**+poslinkis). Vadinasi segmentas SS.

Todėl absoliutą adresą skaičiuosime pagal formulę:

AA = SS\*10h + EA. (jei būtų daugiau nei 5 skaitmenys papildomus ignoruotume, jei skaitmenų būtų mažiau nei 5 priekyje prisirašytume nulius)

$$\begin{array}{r|l} & 34560 \\ + & 3223 \\ \hline & 37783 \end{array}$$

**Uždavinio ypatybės:** operando absoliutaus adreso skaičiavimas, kai nėra segmento keitimo



prefikso ir efektyvaus adreso formavimui kažkokioje kombinacijoje panaudotas BP

**Atsakymas:** 37783

## **UŽDAVINYS NR. 2**

**Uždavinio sąlyga:** Registrų reikšmės yra CS=70DF; DS=76E1; SS=7D1F; ES=12CE; BP=631D; BX=443D; DI=FBB2; SI=AF1D. Apskaičiuokite operando atmintyje absoliutų adresą pagal adresavimo baitą 2E. Po adresavimo baito seka baitai 23 3C.

**Sprendimas:**

*Tikslas* – rasti operando atmintyje absoliutų adresą

Adresavimo baitas yra 2E. Užsirašome jį dvejetainėje sistemoje turime 00101110.

Iš jo nusistatome mod, reg, r/m reikšmes.

Gavome, kad:

mod=00, reg=101, r/m=110

Pagal mod ir r/m iš adresavimo baito lentelės nustatome, kad tai yra atvejis: tiesioginis adresas.

Tiesioginis adresas suprantamas kaip „vienišas poslinkis“ prie kurio nepridėti jokie registrai.

Ir tas poslinkis yra visuomet dviejų baitų – tiesioginis adresas iškart nurodo, koks yra operando efektyvus adresas.

Taigi sekantys du poslinkio baitai ir bus tiesioginis adresas, reikia atkreipti dėmesį, kad pirmas yra jaunesnysis, kitas vyresnysis, todėl šiuo atveju operando atmintyje efektyvus adresas yra EA=3C23.

Absoliutus adresas skaičiuojamas  $AA = \text{seg.reg.} * 10h + EA$

Trūksta nustatyti koks segmento registras yra panaudotas absoliutaus adreso formavimui.

Pirmiausia žiūrime, ar yra prefiksas? Ne – prefikso nėra. Gali kilti pagunda sakyti, kad 2E reiškia cs: prefiksą, tačiau, 2E yra ne prefikso baitas, o adresacijos baitas.

Jei būtų prefiksas, jis nurodytų, koks yra operando atmintyje segmentas, tačiau šiuo atveju prefikso nėra, tai reiškia, kad operando atmintyje segmentas yra DS arba SS. Kuris iš jų?

Jei kombinacijoje panaudotas BP ---> SS

Jei kombinacijoje nepanaudotas BP ---> DS

Kadangi formuojant efektyvų adresą nepanaudotas BP, reiškia turime, kad segmentas yra DS.

Iš sąlygos DS=76E1, EA=3C23.

Skaičiuojame AA pagal formulę  $AA = DS * 10h + EA$ .

Turime:

$$\begin{array}{r} + \quad 76E10 \\ \quad 3C23 \\ \hline 7AA33 \end{array}$$

Jei absoliutus adresas sudarytų daugiau nei 5 skaitmenys, į papildomus dėmesio nekreiptume, jei turėtume per mažai trūkstamų vietoje prisirašytume nulius. T.y. gavus 123456 AA būtų 23456. Turint 123 AA būtų 00123.

**Uždavinio ypatybės:** operando absoliutaus adreso skaičiavimas pagal adresavimo baitą, kai nėra prefikso, ir EA yra tiesioginis adresas (direct address).

**Atsakymas:** 7AA33

## **Status Flag registras [1]**

### **UŽDAVINYS NR. 1**

**Uždavinio sąlyga:** Vykdydami baitų atimties operacija iš dešimtainio skaičiaus 160 atimame dešimtaini skaičių -28. Pradinis SF=7964. Kokia bus nauja SF registro reikšmė?

**Sprendimas:**

*Tikslas* – rasti naują SF reikšmę.

Sudarome tokią lentelę:

Pabrauktieji bitai nesikeis vykdant aritmetinę operaciją. (nenaudojamus žymiu žvaigždutėmis)

Bitų reikšmės	****	ODIT	SZ*A	*P*C
Pradinis SF	0111	1001	0110	0100

Naujas SF				
-----------	--	--	--	--

Atliekame sąlygoje nurodytą atimtį:

Pasiverčiame duotus skaičius į skaičius be ženklų, ir užrašome šešioliktainėje sistemoje (galima ir dvejetainėje)

$$\begin{array}{r} \text{A0} \\ - \text{E4} \\ \hline 1 \text{ BC} \end{array}$$

JEI ATIMTĮ KEIČIATE SUDĖTIMI, DALIS FLAGŲ (AF, CF gali būti gauti neteisingi)

Dėl įvykusio pasiskolinimo iš už rezultato lauko ribų (fiksuoja vienetuku, CF=1)

Užsirašome rezultato lauką dvejetainėje sistemoje, turime **BCh=1011 1100**.

Vienetukų kiekis yra 5, skaičius 5 yra nelyginis, todėl PF=0.

Taip pat, vyriausias bitas (pabrauktasis) yra ženklų bitas, todėl SF=1.

Jauniausias pusbaitis atlieka pasiskolinimą iš vyresnio? Taip, 0-4 reikia skolintis, vadinasi AF=1.

Ar rezultato laukas yra sudarytas tik iš nulinių bitų? Ne, ZF=0.

Beliko nustatyti naują OF reikšmę.

Į skaičius, kuriais operuojame ir rezultatą žiūrime kaip į dešimtainius skaičius su ženklu.

Tuomet, žiūrime ar atlikę veiksmą gauname teisingą rezultatą, jei ne, tada OF=1, jei rezultatas teisingas OF=0.

Vienam baite reikšmių aibės dydis yra 256, skaičiaus su ženklu intervalas **[-128; 127]**.

160 nepatenka į skaičiaus su ženklu intervalą, todėl pasiverčiame į skaičių su ženklu:

$$160-256=-96.$$

-28 patenka į intervalą, tai yra tinkamas skaičius su ženklu.

Gautas rezultatas: Bch=11\*16+12=176+12=188 (nepatenka į intervalą).

Pasiverčiame į skaičių su ženklu 188-256=-68.

Ką tai reiškia, kad atlikome veiksmą

Iš -96 atimame -28 gauname -68

T.y.

-96-(-28)=-96+28=-68. Rezultatas tinkamas, perpildymo nėra, OF=0.

Papildome prieš tai naudotą lentelę:

Bitų reikšmės	****	ODIT	SZ*A	*P*C
Naujas SF	<b>0111</b>	<b>1001</b>	<b>0110</b>	<b>0100</b>
Naujas SF	0111	0001	1011	0001
Naujas SF šešioliktainėj	7	1	B	1

Gautas atsakymas ir bus 71B1.

**Uždavinio ypatybės:** SF skaičiavimas, kai atliekama baitų atimties operacija.

**Atsakymas:** 71B1

## Slankaus kablelio skaičiai (float'ai) [1]

**Uždavinio sąlyga:** Užrašykite dešimtainį skaičių -6,725 slankaus kablelio koprosoriaus vidiniu formatu šešioliktaine sistema.

**Sprendimas:**

**Tikslas** - Užrašyti turimą dešimtainį slankaus kablelio skaičių nurodytu formatu.

Pasakyta, kad tai vidinis formatas. Vadinasi reikės užrašyti duotą skaičių dešimtyje baitų.

Šiame formate turime:

1 ženklų bitas

15 bitų charakteristikos (eilė+3FFFh)

1 i bitas

Mantisė iki pat galo (kol susidarys 10 baitų).

Turime dešimtainį skaičių -6,725  
Skaičius neigiamas, todėl ženklo bitas bus 1

Ženkla užfiksavome, dabar galime elgtis kaip su teigiamu skaičiumi 6,725  
Užsirašome turimą skaičių dvejetainėje sistemoje:  
Sveikoji dalis 6 ir trupmeninė 725.

Veiksmai su sveikąja dalimi:

$$6 = 110_2$$

Veiksmai su trupmenine dalimi:

Išsirašysime tokio ilgio seką, kokios užteks mantisei užpildyti.

$$0,725 * 16 = 11,6 \quad (11_{10} = 1011_2)$$

$$0,6 * 16 = 9,6 \quad (9_{10} = 1001_2)$$

$$0,6 * 16 = 9,6 \quad (9_{10} = 1001_2)$$

Pastebime ciklišumą (pabrauktos dalys kiek bedauginant kartosis), turimas skaičius (+6,725) dvejetainėje bus:

110,1011(1001)

Skliaustuose nurodytą dalį kartosime tiek kartų kiek prireiks mantisei užpildyti.

Sudarome normalizuotą formą 1,mantisė.

Bendroji forma yra i,mantisė (normalizuotos atveju i=1)

Turime:

$$-1,10 \ 1011 \ (1001) * 2^2$$

Eilė=2

Charakteristika:  $\text{eilė} + 3FFF_h = 4001_h$

Mantisė: 10 1011 (1001) skliaustelių turinį kartosime tiek kartų kiek prireiks.

Užrašome tai ką turime dvejetainė sistema, ir po verčiame į šešioliktinę:

Turime:

ženklų bitas 1, charakteristika 4001h = 100 0000 0000 0001, i bitas 1

Mantisė 10 1011 (1001)

**Pirmi penki baitai:**

1100	0000	0000	0001	1101	0111	0011	0011	0011	0011
C	0	0	1	D	7	3	3	3	3

**Kiti penki baitai:**

0011	0011	0011	0011	0011	0011	0011	0011	0011	0011
3	3	3	3	3	3	3	3	3	3

**Uždavinio ypatybės:** slankaus kabelio skaičiaus užrašymas dešimtyje baitų.

**Atsakymas:** C0 01 D7 33 33 33 33 33 33 33

# Papildoma informacija

Tai yra įvairaus pobūdžio papildoma informacija, siekiant, kad konspektas būtų kiek galima aiškesnis. Taip pat, kad ruošimasis egzaminui būtų efektyvesnis ir sklandesnis.

## ***Įvairios sąvokos, taisyklės ir pastabos***

**Betarpishkas operandas** – operandas konstanta, konkretus skaičius naudojamas kaip vienas iš komandos operandų. Tarkim komandoje *mov ax, 12h* turime betarpishką operandą 12h. Betarpishkame operande kitaip nei atmintyje ar registre neįmanoma išsaugoti kokios nors reikšmės, tarkim *mov 12h, ax* būtų negalima komanda.

**Invertavimas** – bitų reikšmių priešingomis pakeitimas (vienetukai keičiami nuliukais, nuliukai vienetukais), angliškai „complement“ tarkim invertavus baitą 0011 0110 turėtume 1100 1001

**Plėtimo pagal ženklą taisyklė** – pridedant poslinkį, esantį viename baite jį reikia suprasti kaip skaičių su ženklu, todėl pridedant jį prie efektyvaus adreso arba kokio nors žodinio registro reikšmės jį reikia plėsti pagal ženklą. Plačiau apie tai [\[čia\]](#)

**Prefiksas/prefiksinė komanda** – prieš komandos operacijos kodą einantis baitas (gali būt ne ne vienas), įtakojantis segmentų komandoje naudojimą (segmento keitimo prefiksai) arba kiek kartų reikės vykdyti komandą (pakartojimo prefiksai). Iš tiesų jei panaudoti 3 prefiksai prieš vieną komandą, bus įsiminti tik 2, bet nėra apibrėžta kurie.

**Rezultato laukas** – baito arba žodžio dydžio apribotas atminties laukas, kuriame saugoma reikšmė yra iš gana siauro riboto intervalo (pvz. vienas baitas gali įgyti reikšmę tik iš 256 t.y.  $2^8$  galimų).

**DS:BX ir kiti tokie užrašai:** nurodo absoliutų adresą atmintyje. Kairėje dvitaškio yra segmento registro reikšmė, o dešinėje efektyvus adresas. Šiuo atveju efektyvus adresas būtų BX reikšmė, o absoliutaus adreso reikšmė  $DS*10h+BX$