

KTU KOMPIUTERIŲ KATEDRA



Programavimas assembleriu

Darius BIRVINSKAS

Ignas MARTIŠIUS

Algimantas VENČKAUSKAS

Turiny

1	Skaičiavimo sistemos	3
1.1	Sveikųjų dešimtainių skaičių išreiškimas dvejetainė, aštuntaine arba šešioliktaine sistema	4
1.2	Dešimtainių trupmenų išreiškimas dvejetainė, aštuntaine arba šešioliktaine sistema	5
1.3	Dvejetainių, aštuntainių ir šešioliktainių skaičių dešimtainė skaičiavimo sistema	6
1.4	Aštuntainių ir šešioliktainių skaičių išreiškimas skaičiavimo sistema	6
1.5	Dvejetainių skaičių išreiškimas aštuntaine (šešioliktaine) skaičiavimo sistema	7
2	Duomenų vaizdavimas kompiuterio atmintyje	8
2.1	Loginių duomenų vaizdavimas	8
2.2	Fiksuoto kablelio skaičių vaizdavimas	9
2.3	Slankaus kablelio skaičių vaizdavimas	12
2.4	Aritmetinės operacijos įvairiomis skaičiavimo sistemomis . . .	13
2.5	Informacijos apdorojimo principai kompiuteryje	14
3	Asemblerio kalba ir kompiuteris	17
3.1	Programavimo kalbų hierarchija	17
3.2	Kompiuterio struktūrinė schema	19
3.3	x86 architektūros pagrindai	20
3.4	I8086 architektūros ypatybės	27
4	Asemblerio kalbos pagrindai	34
4.1	Operatoriaus struktūra	34
4.2	Programos struktūra	35
4.3	Duomenų aprašymas ir atminties laukų rezervavimas	36
4.4	Paprasčiausi operandų adresavimo būdai	37
5	Mikroprocesoriaus I8086 komandų sistema	39
5.1	Duomenų persiuntimo komandos	39
5.2	Aritmetinių operacijų komandos	41

5.3	Nukreipimo komandos	47
5.3.1	Nukreipimo adresavimas	48
5.3.2	Sąlyginio nukreipimo komandos	48
5.4	Ciklo programavimo komandos	50
5.4.1	Loginės ir postūmio komandos	53
5.4.2	Procedūrų iškviatimo ir grįžimo iš jų komandos	58
5.4.3	Duomenų perdavimas iškviatiant procedūras	64
5.4.4	Pertraukimų programavimo komandos	65
5.5	Procesoriaus valdymo komandos	66
5.5.1	Įvedimo/išvedimo komandos	67
5.5.2	Vėliavėlių persiuntimo komandos	67
6	Sudėtingesni operandų adresavimo būdai ir programų pa- vyzdžiai	68
6.1	Netiesioginis operandų adresavimas	68
6.2	Duomenų transformavimo algoritmai	74
6.2.1	Dešimtainio skaičiaus vertimas i dvejetainę skaičiavi- mo sistemą.	76
7	Informacijos įvedimo - išvedimo programavimas	79
7.1	Klaviatūros valdymas	79
7.1.1	Klaviatūros valdymas MS-DOS priemonėmis	81
8	Makropriemonės	85
8.1	Makroaprašai ir makrokomandos	85
8.2	Kartojimo direktyvos	89
8.3	Makroaprašų sudarymas ir vartojimas	92
9	Darbas <i>I8086</i> emulatoriumi	93
9.1	Darbo eiga	93
9.2	Programų pavyzdžiai	96
9.2.1	"Labas pasauli!" išvedimo programa	96
9.2.2	Atminties valdymo programa	97
A	Užduočių variantai	100
A.1	Aritmetinės išraiškos	100
A.2	Duomenų formatai	102
A.3	Pavyzdys	103
B	Komandų sąrašas	107

skyrius 1

Skaičiavimo sistemos

Įvairios skaičiavimo sistemos skiriasi skaitmenimis ir jų vartojimo taisyklėmis. Mums įprasta dešimtaine pozicine skaičiavimo sistema.

Sistema vadinama dešimtaine, nes joje yra dešimt skaitmenų: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Skaičius 10 - sistemos pagrindas. Sistema vadinama pozicine, nes daugiaženklio skaičiaus kiekvieno skaitmens reikšmė priklauso nuo jo padėties (pozicijos) skaičiuje. Pavyzdžiui, skaičiuje 457 pirmasis skaitmuo žymi keturis šimtus, antrasis - penkias dešimtis, o paskutinis - septynis vienetų: $400 + 50 + 7 = 457$.

Šių skaičių galima užrašyti sistemos pagrindo laipsniais, padaugintais iš koeficientų:

$$457 = 4 \cdot 10^2 + 5 \cdot 10^1 + 7 \cdot 10^0$$

Matome, kad einant iš dešinės į kairę, kiekvienos paskesnės pozicijos reikšmė padidėja tiek kartų, koks yra sistemos pagrindas.

Dešimtainės skaičiavimo sistemos trupmenos ir mišrieji skaičiai interpretuojami analogiškai. Kiekviena pozicija, einant į dešinę, nuo kablelio, atitinka to paties pagrindo laipsnį mažėjančiu neigiamu rodikliu. Pavyzdžiui:

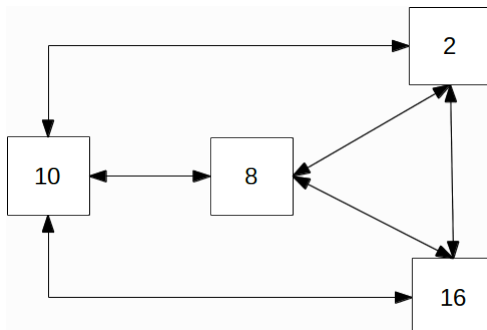
$$436,576 = 4 \cdot 10^2 + 3 \cdot 10^1 + 6 \cdot 10^0 + 5 \cdot 10^{-1} + 7 \cdot 10^{-2} + 8 \cdot 10^{-3}$$

Analogiškai gali būti užrašyti skaičiai ir kitomis skaičiavimo sistemomis.

Dešimtainė skaičiavimo sistema yra nepatogi kompiuteriui, nes jo atminties įrenginiai gali atsiminti tik vieną iš skaitmenų: 0 arba 1. Todėl kompiuteriuose vartojame ne dešimtainę, o dvejetainę skaičiavimo sistemą.

Dvejetainėje skaičiavimo sistemoje yra tik 2 skaitmenys: 0 ir 1. Užrašas 10 čia atitinka dešimtainį skaičių 2. Dvejetainiai skaičiai sudaromi tik 0 ir 1 deriniu. Skaičiavimo sistemos pagrindą, nurodysime prie skaitmenų. Pavyzdžiui:

$$101101_{(2)} = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 45_{(10)}$$



1.1 pav.: Ryšiai tarp skaičiavimo sistemų

Trupmeniniai dvejetainiai skaičiai interpretuojami analogiškai:

$$0,1011_{(2)} = 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} = \frac{11}{16}_{(10)}$$

arba $0,5 + 0,125 + 0,0625 = 0,6875_{(10)}$.

Dažnai dešimtainį skaičių, užrašytą dvejetainė skaičiavimo sistema, sudaro ilga nulių ir vienetų seka, kurią sunku greitai suvokti ir nepatogu vartoti. Tokius skaičius trumpiau galima užrašyti aštuntaine arba šešioliktaine skaičiavimo sistema.

Analogiškai dešimtainei skaičiavimo sistemai aštuntainė sistema turi 8 skaitmenis: 0, 1, 2, 3, 4, 5, 6, 7, o šešioliktainė - 16: pirmi dešimt (0 - 9) imami iš dešimtainės sistemos ir atitinka tuos skaitmenis, kiti šeši (10,11,12,13,14,15) užrašomi raidėmis: A, B, C, D, E, F.

Kadangi $2^3 = 8$ ir $2^4 = 16$, tai kiekvienas aštuntainis skaitmuo užrašomas 3 dvejetainiais skaitmenimis, o kiekvienas šešioliktainis skaitmuo - 4 dvejetainiais skaitmenimis dvejetainė skaičiavimo sistema.

Be minėtų skaičiavimo sistemų, gali būti vartojamos ir kitos, todėl reikia mokėti išreikšti vienos skaičiavimo sistemos skaičius kita sistema. Praktikoje kartu dažniausiai vartojamos trys skaičiavimo sistemos: dešimtainė, aštuntainė ir dvejetainė arba dešimtainė, šešioliktainė ir dvejetainė. Taigi išreiškiama taip, kaip pavaizduota 1.1 pav.

1.1 Sveikųjų dešimtainių skaičių išreiškimas dvejetainė, aštuntaine arba šešioliktaine sistema

Dešimtainis sveikasis skaičius išreiškiamas nauja skaičiavimo sistema, nuosekliai dalijant tą skaičių ir gautus dalmenis iš naujos skaičiavimo sistemos pagrindo, užrašyto dešimtaine skaičiavimo sistema. Gautos liekanos ir pasikutinis dalmuo surašyti atvirkštine tvarka, sudarys naujos sistemos skaičių. Visi veiksmai atliekami dešimtaine sistema.

Išreiškiant skaičius šešioliktaine sistema gautos liekanos 10, 11, 12, 13, 14, 15 keičiamos atitinkamais šešioliktainiais skaitmenimis A, B, C, D, E, F.

Pateikiame pavyzdžių, kaip sveikieji dešimtainiai skaičiai išreiškiami dvejetainė, aštuntine ir šešioliktaine sistema.

$$\begin{array}{l}
 421_{(10)} : \\
 \left. \begin{array}{l}
 2 \overline{) 421} \quad 1 \\
 2 \overline{) 210} \quad 0 \\
 2 \overline{) 105} \quad 1 \\
 2 \overline{) 52} \quad 0 \\
 2 \overline{) 26} \quad 0 \\
 2 \overline{) 13} \quad 1 \\
 2 \overline{) 6} \quad 0 \\
 2 \overline{) 3} \quad 1 \\
 2 \overline{) 1} \quad 1
 \end{array} \right\} = 110100101_{(2)}
 \end{array}$$

$$\begin{array}{l}
 421_{(10)} : \\
 \left. \begin{array}{l}
 8 \overline{) 421} \quad 5 \\
 8 \overline{) 52} \quad 4 \\
 8 \overline{) 6} \quad 6
 \end{array} \right\} = 645_{(8)}
 \end{array}$$

$$\begin{array}{l}
 421_{(10)} : \\
 \left. \begin{array}{l}
 16 \overline{) 421} \quad 5 \\
 16 \overline{) 26} \quad 10 \\
 16 \overline{) 1} \quad 1
 \end{array} \right\} = 1A5_{(16)}
 \end{array}$$

1.2 Dešimtinių trupmenų išreiškimas dvejetainė, aštuntine arba šešioliktaine sistema

Dešimtainė trupmena išreiškiama nauja skaičiavimo sistema, nuosekliai dauginant ją ir gautas trupmenines dalis iš naujos sistemos pagrindo, užrašyto dešimtaine sistema. Visi veiksmai atliekami dešimtaine sistema. Gauti skaitmenys vienetų skiltyse sudarys naujos sistemos skaičių.

Išreikšdami šešioliktaine sistema sveikojoje skaičiaus dalyje gautus skaičius 10, 11, 12, 13, 14, 15, keičiame atitinkamais šešioliktainiais skaitmenimis A, B, C, D, E, F. Pateikiame pavyzdžių kaip dešimtainės trupmenos išreiškiamos dvejetainė, aštuntine ir šešioliktaine sistema:

$$\begin{array}{r}
 0,71875 \\
 \times \quad 2 \\
 \hline
 1,43750 \\
 \times \quad 2 \\
 \hline
 0,87500 \\
 \times \quad 2 \\
 \hline
 1,75000 \\
 \times \quad 2 \\
 \hline
 1,50000 \\
 \times \quad 2 \\
 \hline
 1,00000
 \end{array}$$

$$\begin{array}{r}
 0,71875 \\
 \times \quad 8 \\
 \hline
 5,75000 \\
 \times \quad 8 \\
 \hline
 6,00000
 \end{array}$$

$$\begin{array}{r}
 0,71875 \\
 \times \quad 16 \\
 \hline
 11,50000 \\
 \times \quad 16 \\
 \hline
 8,00000
 \end{array}$$

$0,71875_{(10)} = 0,10111_{(2)}$ $0,71875_{(10)} = 0,56_{(8)}$ $0,71875_{(10)} = 0, B8_{(16)}$
 Pateiktame pavyzdyje dešimtainę trupmeną tiksliai išreiškėme dvejetainė, aštuntinė ir šešioliktinė sistema. Tačiau taip būna ne visuomet. Dažnai daugybos procesas yra begalinis ir rezultatas būna begalinė periodinė trupmena. Tuomet skaitmenų skaičius po kablelio imamas toks, koks reikalingas tikslumas. Jeigu dešimtainių skaičių sudaro sveikoji ir trupmeninė dalis, tai pagal pirmąją taisyklę išreiškiama sveikoji dalis, o pagal antrąją - trupmeninė dalis.

1.3 Dvejetainių, aštuntinių ir šešioliktinių skaičių dešimtainė skaičiavimo sistema

Dvejetainiai, aštuntiniai ar šešioliktiniai skaičiai išreiškiami dešimtaine sistema taip: susumuojami visi dvejetainės, aštuntinės ar šešioliktinės sistemos skaitmenys, padauginėti iš skaičiavimo sistemos pagrindo, pakelto atitinkamos pozicijos laipsniu. Atlikus aritmetines operacijas dešimtaine skaičiavimo sistema, gaunamas dešimtainis skaičius.

Pateikiame pavyzdį, kaip dvejetainiai, aštuntiniai ir šešioliktiniai skaičiai išreiškiami dešimtaine skaičiavimo sistema:

$$\begin{aligned} 110100101, 10111_{(2)} &= 1 \cdot 2^8 + 1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + \\ &\quad + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} + 1 \cdot 2^{-5} = \\ &= 256 + 128 + 0 + 32 + 0 + 0 + 4 + 1 + \frac{1}{2} + 0 + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} = 421,71875_{(10)} \end{aligned}$$

Pateiktas pavyzdys tuo pačiu parodo, kad veiksmas atlikti teisingai - gavome tuos pačius dešimtainius skaičius, kuriuos ankstesniuose pavyzdžiuose išreiškėme dvejetainė, aštuntinė ir šešioliktinė sistema.

1.4 Aštuntinių ir šešioliktinių skaičių išreiškimas skaičiavimo sistema

Kiekvienas aštuntinis (šešioliktinis) skaitmuo dvejetainė sistema užrašomas trimis (keturiais) skaitmenimis. 1.1 lentelėje pateikti pirmųjų šešiolikos skaičių dešimtainiai, aštuntiniai ir dvejetainiai ekvivalentai.

Pateikiame pavyzdžių, kaip aštuntiniai ir šešioliktiniai skaičiai išreiškiami dvejetainė skaičiavimo sistema:

$$645, 56_{(8)} = 110100101, 101110_{(2)}$$

$$1A5, B8_{(16)} = 000110100101, 10111000_{(2)}$$

Kaip ir dešimtainėje skaičiavimo sistemoje, dvejetainė sistemoje užrašyti nereikšminiai nuliai neturi įtakos.

Skaičius			
Dešimtainis	Aštuntainis	Šešioliktainis	Dvejetainis
0	0	0	0000
1	1	1	0001
2	2	2	0010
3	3	3	0011
4	4	4	0100
5	5	5	0101
6	6	6	0110
7	7	7	0111
8	10	8	1000
9	11	9	1001
10	12	A	1010
11	13	B	1011
12	14	C	1100
13	15	D	1101
14	16	E	1110
15	17	F	1111

1.1 lentelė: Skaičių ekvivalentai

1.5 Dvejetainių skaičių išreiškimas aštuntaine (šešioliktaine) skaičiavimo sistema

Norint dvejetainių skaičių išreikšti aštuntaine (šešioliktaine) skaičiavimo sistema, reikia jį suskaidyti grupėmis (į kairę ir į dešinę nuo kablelio po tris (keturis) dvejetainius skaitmenis ir kiekvieną grupę pakeisti vienu aštuntainiu (šešioliktainiu) skaitmeniu. Jei pirmoji triada (tetradą) iš kairės sveikojoje dalyje ir paskutinioji trupmeninėje dalyje nepilnos, jos papildomos nuliais. Pavyzdžiui:

$$110100101, 101110_{(2)} = 110/100/101/, 101/110_{(2)} = 654, 56_{(8)}$$

$$000110100101, 10111000_{(2)} = 0001/1010/0101/, 1011/1000_{(2)} = 1A5, B8_{(16)}$$

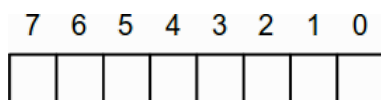
skyrius 2

Duomenų vaizdavimas kompiuterio atmintyje

Kompiuterio apdorojamus duomenis galima suskirstyti į 3 grupes:

- loginius duomenis;
- fiksuoto kablelio skaičius;
- slankaus kablelio skaičius.

Visą informaciją kompiuteris saugo dvejetainė skaičiavimo sistema, t.y. 1 ir 0 deriniais. Mažiausias informacijos vienetas yra bitas. Jame galima saugoti tik vieną dvejetainį skaitmenį: 0 arba 1. Aštuonių bitų grupė sudaro baitą. Bitai jame numeruojami iš dešinės į kairę nuo 0 iki 7 (2.1 pav.).



2.1 pav.: Bitų numeracija baite

Viename baite galima saugoti 256 skirtingus 1 ir 0 derinius. Jei mažai vieno - informaciją galima saugoti keliuose baituose. Dviejų baitų grupė sudaro žodį, dviejų žodžių grupė sudaro dvigubą žodį. Kelių baitų grupė dažniausiai vadinama lauku.

Nepaisant to, kad visi duomenys kompiuteryje saugomi dvejetainiais skaitmenimis, kiekviena duomenų grupė turi savą vaizdavimo formatą.

2.1 Loginių duomenų vaizdavimas

Loginiai duomenys gali būti laikomi baituose, žodžiuose arba laukuose. Šiems duomenims vaizduoti vartojami visi baito, žodžio arba lauko bitai. Loginiais kodais vaizduojami:

- simboliniai duomenys;
- skaičiai be ženklo;
- bitų duomenys.

Simboliniai duomenys yra visi klaviatūros simboliai :

- lotyniška abėcėlė (A - Z);
- skaitmenys (0 - 9);
- specialūs ženklai bei valdymo klavišai.

Kiekvienas simbolis vaizduojamas tam tikru 1 ir 0 deriniu viename baite. Tai ASCII kodas (*angl.* American Standart Code for Information Intertar-change - Amerikos standartinis kodas informacijos mainams). Pagrindinių simbolių ASCII kodai pateikti 2.1 lentelėje. Pavyzdžiui, simboliai “A” ir “B” atrodytų taip:

0	1	0	0	0	0	0	1
$4_{(16)}$				$1_{(16)}$			
A							

0	1	0	0	0	0	1	0
$4_{(16)}$				$2_{(16)}$			
B							

Skaičiai be ženklo baite ar žodyje vaizduojami dvejetainė skaičiavimo sistema, todėl baite gali tilpti skaičiai nuo 0 iki 255 (00-FF), o žodyje – nuo 0 iki 65535 (0000 – FFFF). Pavyzdžiui skaičius 19 baite ir žodyje atrodytų taip:

0	0	0	1	1	0	0	1
$1_{(16)}$				$9_{(16)}$			

0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1
$0_{(16)}$				$0_{(16)}$				$1_{(16)}$				$9_{(16)}$			

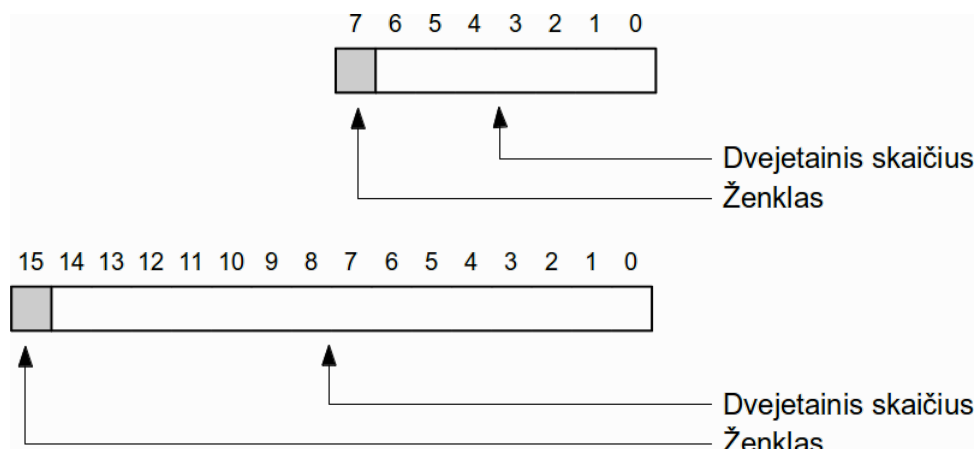
Bitų duomenys gali būti pavaizduoti atskiruose baito ar žodžio bituose yra vartojami loginei informacijai užkoduoti.

2.2 Fiksuoto kablelio skaičių vaizdavimas

Fiksuoto kablelio skaičiai vaizduojami baituose arba žodžiuose. Ši skaičių vaizdavimo forma taikoma tik sveikiesiems skaičiams, nes kablelis visuomet fiksuojamas po nulinio bito.

Šešiolyktainis skaičius	Simbolis	Šešiolyktainis skaičius	Simbolis	Šešiolyktainis skaičius	Simbolis
20	␣	40	@	60	‘
21	!	41	A	61	a
22	"	42	B	62	b
23	#	43	C	63	c
24	\$	44	D	64	d
25	%	45	E	65	e
26	&	46	F	66	f
27	,	47	G	67	g
28	(48	H	68	h
29)	49	I	69	i
2A	*	4A	J	6A	j
2B	+	4B	K	6B	k
2C	,	4C	L	6C	l
2D	-	4D	M	6D	m
2E	.	4E	N	6E	n
2F	/	4F	O	6F	o
30	0	50	P	70	p
31	1	51	Q	71	q
32	2	52	R	72	r
33	3	53	S	73	s
34	4	54	T	74	t
35	5	55	U	75	u
36	6	56	V	76	v
37	7	57	W	77	w
38	8	58	X	78	x
39	9	59	Y	79	y
3A	:	5A	Z	7A	z
3B	;	5B	[7B	{
3C	<	5C	\	7C	
3D	=	5D]	7D	}
3E	>	5E	^	7E	~
3F	?	5F	—	7F	(DEL)

2.1 lentelė: ASCII kodų lentelė



2.2 pav.: Ženklo skiltis baite ir žodyje

Teigiami skaičiai vaizduojami tiesioginiu kodu: ženklų bite yra 0, o kituose - skaičiaus reikšmė dvejetainėje skaičiavimo sistemoje.

Neigiami skaičiai vaizduojami papildomu kodu su vienetu ženklų bite. Skaičiaus papildomas kodas sudaromas invertuojant skaičiaus modulio tiesioginio kodo skaitmenis (t.y, 1 keičiant 0, 0 keičiant 1) ir pridėdam prie gauto skaičiaus 1. Taip pat skaičiaus papildomą kodą galima gauti vartojant šešioliktinę skaičiavimo sistemą: šį skaičių reikia atimti iš skaičiaus $FFFF_{(16)}$ (kai skaičius saugojamas žodyje) ir prie gauto skaičiaus jauniausiojo skaitmens pridėti 1.

Pavaizduokime skaičių -21 baite:

1. Skaičiaus modulį verčiame į dvejetainę skaičiavimo sistemą: $21_{(10)} = 1B_{(16)} = 00011011_{(2)}$.
2. Invertuojame kiekvieną bitą: $11100100_{(2)}$.
3. Prie gauto skaičiaus pridėdame vienetą:

$$\begin{array}{r} 11100100 \\ + \quad \quad 1 \\ \hline 11100101 \end{array}$$

Norint rasti neigiamo skaičiaus modulį reikia pakartoti ankstesnius veiksmus: invertuoti bitus ir pridėti vienetą.

$$\begin{array}{r} 11100101 \quad - \text{neigiamas skaičius} \\ 00011010 \quad - \text{inversija} \\ + \quad \quad 1 \\ \hline 00011011 \quad - \text{teigiamas skaičius} \end{array}$$

Taigi skaičius $11100101_{(2)}$, įrašytas baite, vaizduoja $-21_{(10)}$. Nulio tiesioginis ir papildomas kodas sutampa ir yra lygūs 0. Lentelėje 2.2 pateikiame fiksuoto kablelio skaičių pavyzdžiai. Kaip gauti papildomą kodą vartojant

Dešimtainis sk.	Turinys dvejetainė ir šesioliktinė sistema	
	Baito	Žodžio
+5	00000101 05	0000000000000101 0005
-5	11111011 FB	0000000011111011 FFFB
-1	11111111 FF	1111111111111111 FFFF
127	01111111 7F	0000000011111111 FF7F
128	netelpa	0000000010000000 0080
-128	10000000 80	1111111100000000 FF80
32767	netelpa	0111111111111111 7FFF

2.2 lentelė: Fiksuoto taško skaičių pavyzdžiai

šesioliktinę skaičiavimo sistemą, parodysime vėliau, nagrinėdami aritmetinius veiksmus įvairiomis skaičiavimo sistemomis.

Kaip matome iš lentelės 2.2 baite galima saugoti skaičius iš diapazono $-2^7 \leq s \leq 2^7 - 1$, tai atitinka $-128 \leq s \leq 127$. Analogiškai žodyje galima saugoti skaičius iš diapazono $-2^{15} \leq s \leq 2^{15} - 1$, tai atitinka $-32768 \leq s \leq 32767$.

2.3 Slankaus kablelio skaičių vaizdavimas

Slankaus kablelio skaičių s bet kuria skaičiavimo sistema galima užrašyti taip:

$$s = n \cdot a^p$$

čia

n - taisyklingoji trupmena, vadinama skaičiaus mantise;

a - skaičiavimo sistemos pagrindas;

p - sveikasis skaičius, vadinamas eile.

Kiek vietos skiriama mantisei ir eilei priklauso nuo konkretaus aritmetinio loginio įrenginio (*angl.* Atrimetic logic unit - ALU), todėl šito čia detaliau nenagrinėsime.

2.4 Aritmetinės operacijos įvairiomis skaičiavimo sistemomis

Kadangi kompiuteris informaciją saugo tik dvejetainė skaičiavimo sistema, tai mums įprastą dešimtainę sistemą galima vartoti įvedant pradinis duomenis bei spausdinant apskaičiuotus rezultatus. Kompiuteris ne tik saugo informaciją dvejetainė forma, bet ir veiksmai atliekami šioje sistemoje. Šiame skyrelyje parodysite, kaip galima atlikti aritmetinius veiksmus kitomis skaičiavimo sistemomis.

Jeigu, sudedant du skaitmenis, gautas skaičius w yra lygus ar didesnis už vartojamos skaičiavimo sistemos pagrindą a , tai į atitinkamą poziciją rašomas skaitmuo $d=w-a$, o prie vyresnės pozicijos pridodamas perkėlimo vienetas. Kai $w < a$, tai jis pakeičiamas vartojamos skaičiavimo sistemos skaitmeniu.

Ši taisyklė taikoma ne tik dešimtainei, bet ir kitoms skaičiavimo sistemoms. Pateikiame sudėties pavyzdžius įvairiomis skaičiavimo sistemomis:

$$\begin{array}{r} \begin{array}{r} \overset{1}{1}3_{(10)} \\ + \quad 28_{(10)} \\ \hline 41_{(10)} \end{array} \qquad \begin{array}{r} \overset{1}{1}5_{(8)} \\ + \quad 34_{(8)} \\ \hline 51_{(8)} \end{array} \qquad \begin{array}{r} \overset{1}{1}C_{(16)} \\ + \quad D_{(16)} \\ \hline 29_{(16)} \end{array} \qquad \begin{array}{r} \overset{111}{00}1101_{(2)} \\ + \quad 011100_{(2)} \\ \hline 101001_{(2)} \end{array} \end{array}$$

Atimdami vieną skaitmenį iš kito, elgiamės taip pat, kaip vartodami dešimtainę skaičiavimo sistemą, kai atėminio skaitmuo mažesnis arba lygus turinio skaitmeniui. Kai atėminio skaitmuo yra didesnis už turinio atitinkamą skaitmenį, iš vyresnės pozicijos skolinamas 1 ir atitinkamas turinio skaitmuo padidinamas skaičiavimo sistemos pagrindu, galiausiai iš gauto skaičiaus atimamas atėminio skaitmuo. Ši taisyklė visoms skaičiavimo sistemoms. Pateikiame atimties pavyzdžių įvairiomis skaičiavimo sistemomis:

$$\begin{array}{r} \begin{array}{r} \overset{3}{3}5_{(10)} \\ - \quad 28_{(10)} \\ \hline 7_{(10)} \end{array} \qquad \begin{array}{r} \overset{4}{4}3_{(8)} \\ - \quad 34_{(8)} \\ \hline 7_{(8)} \end{array} \qquad \begin{array}{r} \overset{2}{2}3_{(16)} \\ - \quad 1C_{(16)} \\ \hline 7_{(16)} \end{array} \qquad \begin{array}{r} 100011_{(2)} \\ - \quad 011100_{(2)} \\ \hline 000111_{(2)} \end{array} \end{array}$$

Apie daugybą ir dalybą detaliau nekalbėsime, kadangi šias operacijas galima pakeisti nuosekliomis sudėties ar atimties operacijomis. Atimties ir sudėties veiksmams taip pat apskaičiuojami vykdomieji ar santykiniai adresai.

Atimties operaciją galima efektyviai panaudoti skaičiaus papildomam kodui skaičiuoti t.y. versti skaičiaus iš teigiamų į neigiamus, bei atvirkščiai. Prisiminkime, kad aštuonetainė arba šešioliktainė skaičiavimo sistema kompaktiškai atvaizduoja dvejetainė skaičiavimo sistema pateiktą informaciją. Aštuonetainė skaičiavimo sistema - tris, šešioliktainė keturis kartus trumpiau pavaizduoja žodžių ar baitų turinį. Šiomis skaičiavimo sistemomis taip pat galima paprasčiau gauti ir atvaizduoti skaičiaus papildomą kodą. Teikia norimą skaičių atimti iš didžiausio galimo skaičiaus (šešioliktainėje skaičiavimo sistemoje: $FF_{(16)}$ arba $FFFF_{(16)}$, priklausomai nuo to kaip vaizduojamas skaičius). Gautas skirtumas padidinamas vienetu yra skaičiaus papildomas kodas. Sakysime norime gauti skaičių $-26_{(8)}$ ir $-16_{(16)}$

papildomą kodą. Papildomas kodas įrašytas baite atrodytų taip:

$$377_{(8)} - 26_{(8)} = 351_{(8)}; 351_{(8)} + 1 = 352_{(8)}$$

$$FF_{(16)} - 16_{(16)} = E9_{(16)}; E9_{(16)} + 1 = EA_{(16)}$$

Tie patys skaičiai įrašyti žodyje atrodytų taip:

$$177777_{(8)} - 26_{(8)} = 177751_{(8)}; 177751_{(8)} + 1 = 177752_{(8)}$$

$$FFFF_{(16)} - 16_{(16)} = FFE9_{(16)}; FFE9_{(16)} + 1 = FFEA_{(16)}$$

2.5 Informacijos apdorojimo principai kompiuteryje

Svarbiausia kompiuterio dalis - procesorius - gali atlikti aritmetines, logines, palyginimo bei skaičiavimo proceso valdymo operacijas. Norint sudaryti skaičiavimo algoritmą, pirmiausia reikia žinoti, kokias elementarias operacijas gali atlikti procesorius, ir skaičiavimo procesą suskaidyti į tas operacijas, nurodant jų eilės tvarką. Atliekant tokias operacijas programoje reikia nurodyti atitinkamas komandas ir apdorojamus duomenis. Paprastai operacijų duomenys vadinami operandais. Todėl, nurodant komandą, reikia nurodyti penkis elementus: komandos kodą, pagal kurį kompiuteriui nurodoma kokią elementarią operaciją reikia atlikti ir keturis adresus. Pirmieji du adresai rodo, kur atmintyje yra duomenys, vieno adreso reikia rezultatui ir dar vieno adreso - paskesnei komandai. Tokios komandos vadinamos 4 adresų komandomis. Komandose gali būti nurodomi ne operandų adresai, o patys operandai. Pavyzdžiui, apskaičiuojant išraišką $i = j + k$, galima nurodyti tokią simbolinę komandą:

ADD j , k , i , p

čia :

ADD - simbolinis sudėties komandos kodas;

j, k, i - duomenų bei rezultato adresai;

p - paskesnės komandos adresas.

Matome, kad ši komanda yra ilga, jai užrašyti reikia daug atminties. Todėl pabandykime sutrumpinti komandą. Jeigu programą vykdytume nuosekliai komandą po komandos, tai paskesnės komandos nereikėtų nurodyti, tuo pačiu sutrumpėtų ir pati komanda. Tada tai pačiai operacijai atlikti galima nurodyti tokią komandą:

ADD j , k , i

kuri reikštų, kad prie kintamojo j reikia pridėti kintamąjį k ir rezultatą užrašyti kintamajame i . Tokios komandos vadinamos 3 adresų komandomis. Šiuo atveju paskesnės komandos adresą nurodo komandos skaitiklis.

Norint dar labiau sutrumpinti komandą, reikia panaikinti dar vieną operando adresą. Tokiu atveju komandoje paliekami 2 adresai, tačiau dabar gautas rezultatas turi būti įrašomas į vieną iš duomenų. Šiuo atveju aišku, komanda atmintyje užima mažiau vietos, tačiau programuoti sudėtingiau. Tai pačiai išraiškai apskaičiuoti dabar reikia dviejų komandų:

MOVE	j , i
ADD	k , i

Pirmoji komanda (**MOVE**) kintamąjį j įrašo į kintamojo i vietą, o antroji - prie jo prideda kintamąjį k ir gautą sumą įrašo į tą pačią vietą, t.y. i .

Tokios komandos vadinamos 2 adresų komandomis. Galima dar labiau sutrumpinti komandą joje paliekant tik vieno operando adresą, tačiau operacija atliekama su dviem operandais. Palikus komandoje tik vieną operandą, procesoriuje išskiriama speciali atmintis, vadinama akumuliatoriumi, ir visos operacijos atliekamos su duomenimis, esančiais šiame akumuliatoriuje ir operande, kurio adresas nurodytas komandoje. Gautas rezultatas taip pat paliekamas akumuliatoriuje. Toks adresavimo būdas sutrumpina komandas, tačiau išauga jų skaičius. Pavyzdžiui, užrašant komandas ankstesnei išraiškai apskaičiuoti, šiuo atveju reikėtų rašyti:

LOAD	j
ADD	k
STORE	i

pirmoji komanda (**LOAD**) patalpina į akumuliatorių kintamąjį j , antroji padidina akumuliatoriaus turinį kintamojo k reikšme, trečioji (**STORE**) - rezultatą įrašo į kintamąjį i .

Tokios komandos vadinamos vieno adreso komandomis. Kyla klausimas: ar galima sukurti tokį kompiuterį, kurio komandose nereikėtų nurodinėti nei vieno adreso? Pasirodo - galima. Tereikia duomenis saugoti steko. Stekas - tai toks informacijos saugojimo būdas, kai galima kreiptis tik į paskutinįjį steko elementą: rašant į steką gale prirašomas naujas elementas, skaitant - paimamas paskutinyasis. Taigi, jeigu būtų steko elemento įrašymo ir skaitymo komandos, tai kitoms procesoriaus komandoms nereikėtų adresų. Apskaičiuojant ankstesnę išraišką, gali būti tokios komandos:

LOAD	j
LOAD	k
ADD	
STORE	i

Komanda **ADD** ir kitos panašios komandos vadinamos neadresuotomis. Matome, kad taikant įvairius adresavimo būdus, keičiasi programavimo stilius, vienur mažiau komandų, kitur tam pačiam skaičiavimui jų reikia daugiau. Pavyzdžiui, apskaičiuodami išraišką $i = j + k + l + m$, vartokime tik vieno ir trijų adresų komandas.

Vieno adreso komandos:

LOAD	j
ADD	k
ADD	l
ADD	m
STORE	i

Trijų adresų komandos:

ADD	j, k, i
ADD	i, l, i
ADD	i, m, i

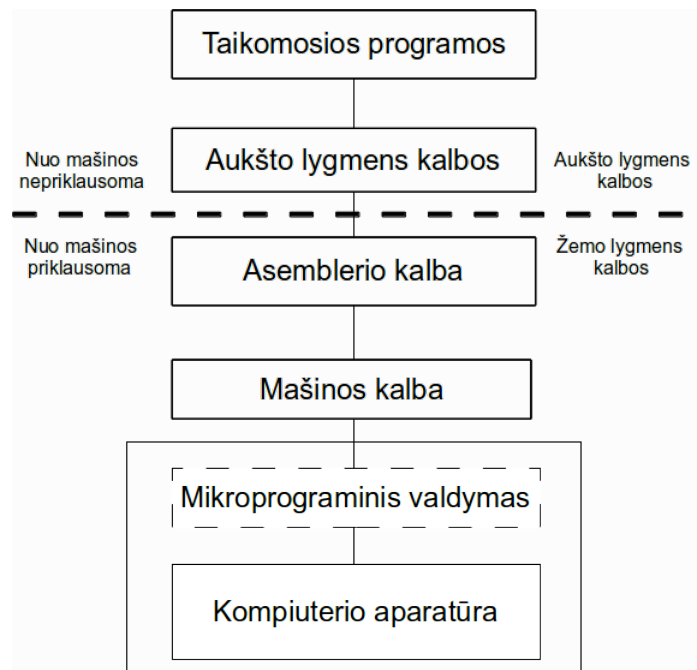
Pasižiūrėkime į sudarytų programų ilgį. Nepaisant to, kad vieno adreso komandų programa šiuo atveju sudaryta iš daugiau komandų negu programa parašyta trijų adresų komandomis, vieno adreso komandų programai reikia mažiau atminties.

skyrius 3

Asemblerio kalba ir kompiuteris

3.1 Programavimo kalbų hierarchija

Kompiuterio darbas valdomas programomis, kurios kuriamos naudojantis programavimo kalbomis. Kalbos yra įvairaus abstrakcijos lygmens. Kuo kalba abstraktesnė, tuo ji yra aukštesnio lygmens. Programavimo kalbų hierarchija parodyta 3.1 pav.

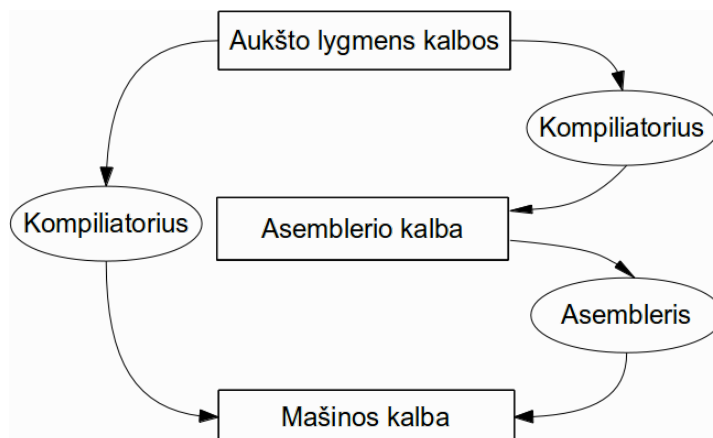


3.1 pav.: Programavimo kalbų hierarchija

Mašinos kalba užrašyta programa (mašininis kodas) - tai instrukcijų ir duomenų rinkinys, kurį tiesiogiai vykdo mašina. Instrukcijos ir duomenys koduojami dvejetainiais skaitmenimis. Tai žemiausio lygmens programavimo kalba. Vėliau sukurtos kitos programavimo kalbos, kurių tekstą galima automatiškai išversti į mašininį kodą (sukompiliuoti).

Kiek aukštesnio lygmens programavimo kalba yra assemblerio kalba. Tai žemo lygmens programavimo kalba, skirta tam tikros architektūros kompiuterių mašininėms komandoms užrašyti. Kiekviena kompiuterio architektūra turi savo mašininį kodą ir savo assemblerio kalbą. Assemblerio kalbos skirtumas nuo mašininio kodo yra tas, kad operacijos iš pradžių užrašomos žmogui suprantamesnėmis ir labiau įsimintinomis simbolinėmis komandomis, o ne dvejetainiu kodu. Paprastai, kiekviena assemblerio komanda atitinka mašinos instrukciją, ir atvirkščiai. Assemblerio kalba parašytų programų transliatorius į mašininį kodą dažniausiai taip pat vadinamas "assembleriu".

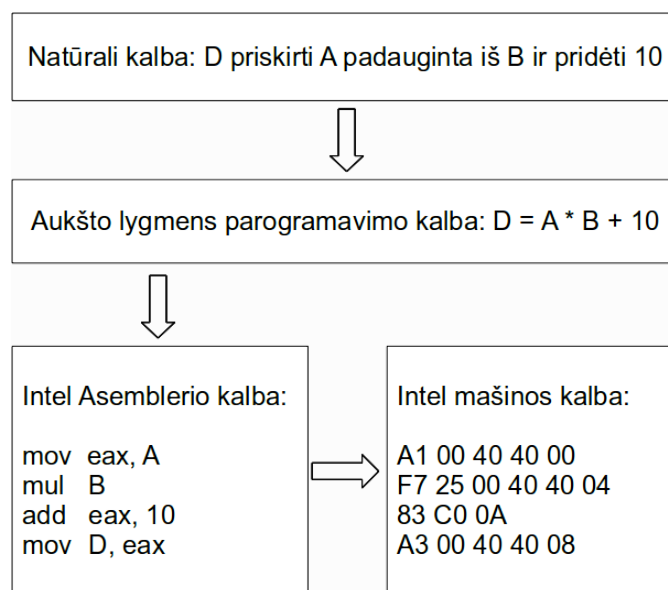
Aukšto lygmens programavimo kalbų kompiliatoriai transliuoja programą į mašininį kodą. Galimi du variantai: transliuojama tiesiogiai į mašininį kodą, arba iš pradžių į assemblerio kalbą, o po to - į mašininį kodą. Programų transliavimo būdai parodyti 3.2 pav.



3.2 pav.: Programų transliavimo būdai

Pavyzdys, kaip atrodo programos fragmentas įvairaus lygmens kalbose, parodytas 3.3 pav. Aukšto lygmens kalbos operatorius paprastai transliuojamas į keletą assemblerio komandų. Kiekviena assemblerio komanda transliuojama į vieną mašinos instrukciją. Paveiksle mašininės instrukcijos (dvejetainiai kodai) užrašyti šešiolyktainiais skaitmenimis.

Assemblerio kalba leidžia programuotojui geriau išnaudoti kompiuterio galimybes, o tai gali būti itin svarbu, kai reikia sudaryti programas, dirbančias realiaame laike, kai tenka griežtai įvertinti konkrečios programos charakteristikas - jos vykdymo laiką ir apimtį atmintyje, efektyviai valdyti aparatinę įrangą.



3.3 pav.: Programos fragmentas įvairaus lygmens kalbose

Norint sudaryti programas assemblerio kalboje, reikia išsiaiškinti pagrindines kompiuterio savybes ir galimybes. Būtina pažymėti, kad kiekvieno tipo kompiuteris turi savo assemblerio kalbą. Jos operatoriai tiesiogiai aprašo kompiuteryje betarpiškai vykdomas komandas.

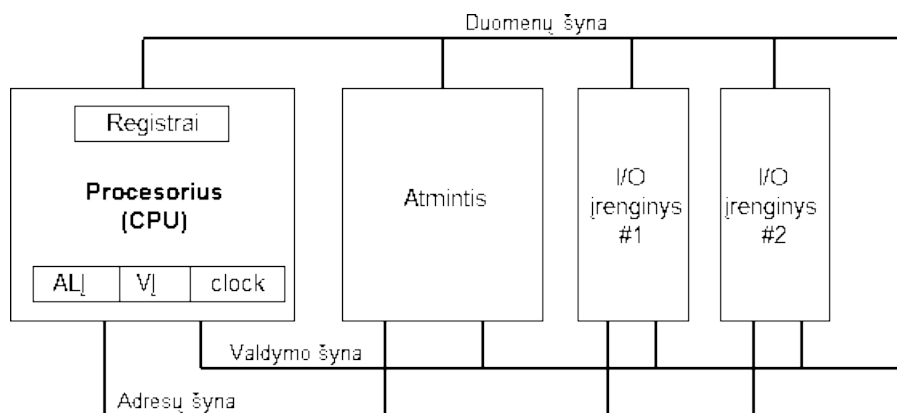
3.2 Kompiuterio struktūrinė schema

Klasikinės architektūros kompiuteriai sudaryti iš trijų pagrindinių komponentų: procesoriaus, dar vadinamo centriniu procesoriumi (CPU), atminties ir įvedimo / išvedimo įrenginių. Apibendrinta kompiuterio struktūra parodyta 3.4 pav.

Šie komponentai yra sujungti viena ar keliomis magistralėmis: adresų, duomenų ir valdymo magistralėmis. Procesoriaus pagrindinės dalys yra aritmetikos ir logikos įrenginys (ALI), atliekantis šio procesoriaus vykdomas aritmetines (sudėties, atimties, dalybos bei daugybos), logines bei postūmio (IR, ARBA, ...) ir kitas operacijas, taktų generatorius (*clock*), valdantysis įrenginys (VI) ir registrai, vietinė procesoriaus atmintis.

Dirbant su kompiuteriu paprastai atliekamos tokios procedūros:

1. Programa ir duomenys iš klaviatūros arba papildomo atminties įrenginio įvedami į operatyvąją atmintį.
2. Centrinis procesorius iš operatyviosios atminties nuosekliai išrenka programos komandomis užduotą vykdymo tvarką. Iš atminties nuskaityti komandoje nurodyti duomenys ir atliekami veiksmai.



3.4 pav.: Kompiuterio struktūrinė schema

3. Gautus skaičiavimo rezultatus procesorius perduoda į atmintį arba vieną iš išvedimo įrenginių - monitorių, išorinę atmintį, spausdinimo įrenginį.

3.3 x86 architektūros pagrindai

Kompiuterio struktūrinėje schemoje parodytas centrinis procesorius - tai speciali mikroschema - mikroprocesorius, kuris vykdo programoje nurodytas komandas.

Šiuolaikiniuose kompiuteriuose, ypač asmeniniuose kompiuteriuose, nedideliuose serveriuose, dažniausiai naudojami Intel x86 architektūros procesoriai. Šeima x86 apima ištisą mikroprocesorių, kurie skiriasi apdorojamos informacijos bitų skaičiumi, registrų dydžiu ir kiekiu, komandų rinkiniais, atminties adresavimo būdais, galingumu ir kitais parametrais, gamą. Pirmasis x86 architektūros procesorius - 8086, sukurtas 1978 metais, buvo 16 bitų, 1 MB adresų erdvės procesorius, vėliau buvo sukurti 32 bitų (IA-32) ir 64 bitų (EM64T, x86-64) procesoriais. Visi šie procesoriai programiškai suderinami iš "apačios į viršų", t.y. programos parašytos 16 bitų procesoriams, veiks ir 32 bei 64 bitų procesoriuose. Mikroprocesoriuose 8086 realizuoti pagrindiniai x86 architektūros principai, kurie vėliau buvo išplėtoti kituose mikroprocesoriuose.

x86 procesoriaus darbo režimai

x86 procesorius turi tris pagrindinius darbo režimus: *apsaugotasis režimas*, *realaus adreso režimas* ir *sistemos valdymas režimas*. Taip pat yra specialus apsaugotojo režimo atvejis, pavadintas *virtualiuoju-8086* režimu. Trumpai aprašysime kiekvieną iš jų:

Apsaugotasis režimas. Apsaugotasis režimas yra pagrindinis procesoriaus režimas, kuriame galima naudoti visas instrukcijas ir funkcijas. Programoms yra paskiriamos atskiros atminties sritys, vadinamos *segmentais*, ir procesorius neleidžia programai kreiptis už jai paskirtų segmentų ribų.

Virtualusis-8086 režimas. Nors tai apsaugotas režimas, procesorius gali saugiai tiesiogiai vykdyti realaus adreso režimo programas daugiaprogramio apdorojimo aplinkoje, pvz., MS-DOS programas. Kitaip tariant, jei MS-DOS programa "pakimba" arba bando rašyti duomenis į sistemos ar kitų programų atminties sritį, tai yra blokuojama, ir tai neturi įtakos kitų programų veikimui tuo pačiu metu. Windows NT gali tuo pačiu metu vykdyti kelias atskiras virtualias 8086 sesijas.

Realaus adreso režimas. Realaus adreso režimas įgyvendina 8086 procesoriaus programavimo aplinką su keletą papildomų funkcijų, pavyzdžiui, galimybe pereiti į kitus režimus. Šis režimas yra galimas Windows 98, ir gali būti naudojamas paleisti MS-DOS programas, kurios reikalauja tiesioginės prieigos prie sistemos atminties ir aparatūros. Programos veikiančios realaus adreso režimu gali sutrikdyti operacinės sistemos darbą, sistema gali nebereaguoti į komandas.

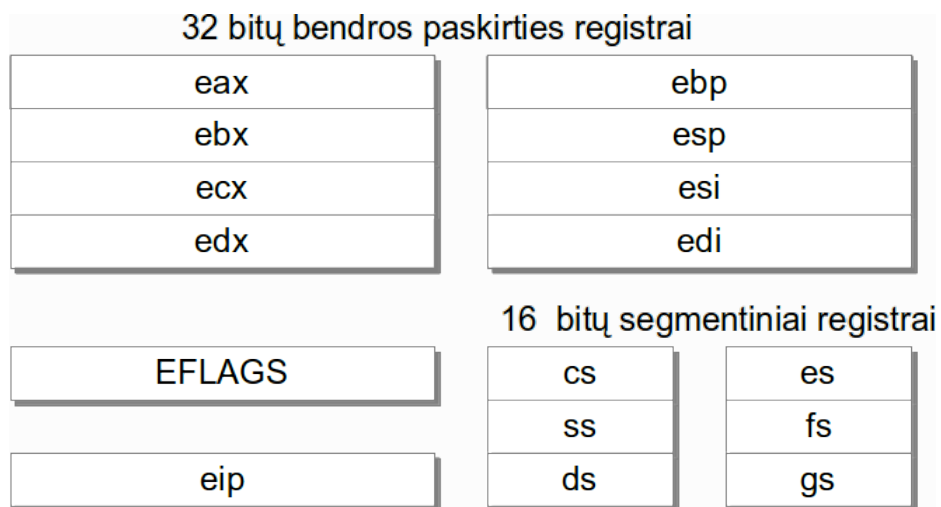
Sistemos valdymas režimas. Sistemos valdymas režimas suteikia operacinei sistemai priemones įgyvendinti energijos valdymo, sistemų saugos ir kitas funkcijas. Šias funkcijas paprastai naudoja kompiuterių gamintojai, pritaikydami procesoriaus nuostatas tam tikroms sistemoms.

Adresų erdvė

32-bitų apsaugotame režime, užduotis ar programa gali naudoti tiesinę adresų erdvę iki 4 GB. Pradedant procesoriumi P6, įgyvendinti atminties valdymo būdai leidžia naudoti fizinę atmintį iki 64 GB. Realaus adreso režimo programos gali naudoti atmintį tik iki 1 MB. Jei procesorius veikia apsaugotu režimu ir vykdo kelias programas virtualiame 8086 režime, kiekviena programa gali turėti savo 1 MB atminties sritį.

Registrai

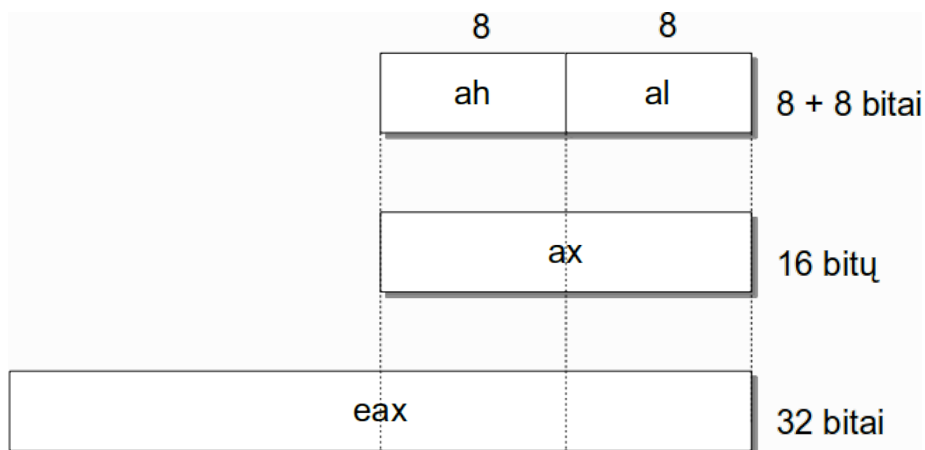
Registrai yra greitaeigė vidinė procesoriaus atmintis. IA-32 mikroprocesoriaus vidinės atminties struktūra, registrai, parodyta 3.5 pav. Yra 8 bendros paskirties registrai, 6 segmentų registrai, procesoriaus būsenos - vėliavėlių registras ir komandų rodiklis.



3.5 pav.: IA-32 mikroprocesoriaus registrai

Bendros paskirties registrai. Bendrosios paskirties registrai pirmiausia naudojami aritmetinėse ir duomenų persiuntimo komandose.

Kaip parodyta 3.6 pav., į 16 jaunesnių registro **eax** bitų galima kreiptis vardu **ax**.



3.6 pav.: Bendros paskirties registras EAX

Kai kurių registrų dalis galima adresuoti kaip 8 bitų registrus. Pvz., **ax** registras, turi 8 bitų vyresniąją dalį, vadinamą **ah** ir 8 bitų jaunesniąją dalį vadinamą **al**. Tai galioja registrams **eax**, **ebx**, **ecx**, **edx**, taip, kaip parodyta 3.1 lentelėje.

32 bitų	16 bitų	8 bitų (vyr.)	8 bitų (jaun.)
eax	ax	ah	al
ebx	bx	bh	bl
ecx	ax	ah	al
eax	ax	ah	al

3.1 lentelė: Bendros paskirties registrai

Kiti bendrosios paskirties registrai, **esi**, **edi**, **ebp**, **esp**, gali būti nurodomi naudojant 32 bitų ar 16 bitų pavadinimus, taip, kaip parodyta 3.2 lentelėje.

32 bitų	16 bitų
esi	si
edi	di
ebp	bp
esp	sp

3.2 lentelė: Bendros paskirties registrai 32 ir 16 bitų

Kai kurie bendrosios paskirties registrai yra naudojami specialioms tikslams:

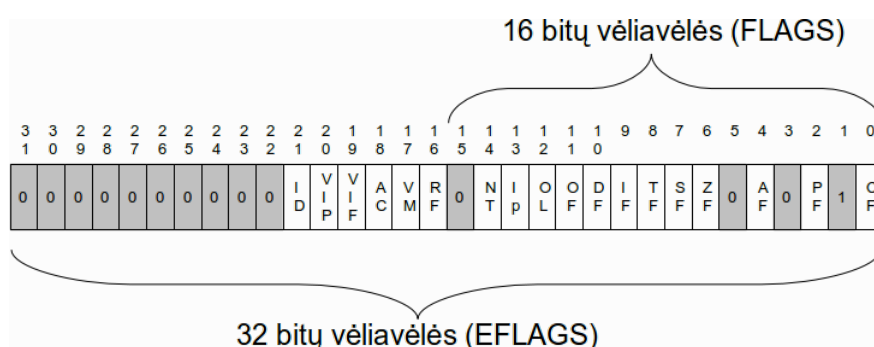
- **eax** registras automatiškai naudojamas daugybos ir dalybos operacijose. Jis vadinamas *išplėsto akumulatoriaus registru*.
- **ecx** registrą procesorius naudoja kaip ciklą skaitliuką.
- **esp** registras adresuoja duomenis dėtuvių, specialioje atminties struktūroje. Jis retai naudojamas įprastose aritmetinėse ar duomenų perdavimo komandose. Jis vadinamas *išplėstos steko rodyklės registru*.
- **esi** ir **edi** registrai yra naudojami komandose, skirtose didelės spartos duomenų perdavimui atmintyje. Jie vadinami *išplėstu šaltinio indekso* ir *išplėstu paskirties indekso registrais*.
- **ebp** registras naudojamas funkcijos parametrų nuorodoms ir vietiniams kintamiesiems dėtuvių adresuoti ir saugoti. Jis neturėtų būti naudojamas įprastose aritmetinėse ar duomenų perdavimo komandose. Jis dažnai vadinamas *dėtuvių išplėstos rodyklės registru*.

Segmentų registrai. Realus adreso režime 16 bitų segmentų registrai nurodo bazinius, iš anksto apibrėžtų atminties sričių, vadinamus segmentais, adresus. Apsaugotame režime segmentų registruose yra nuorodos į segmentų deskriptorių lentelę. Vienuose segmentuose yra programos instrukcijos

(kodas), kituose kintamieji (duomenys), o dar kituose segmentuose, vadina- muose dėtuovės segmentais, yra funkcijų vietiniai kintamieji ir parametrai.

Komandų rodiklis. Registre **eip**, arba komandų rodiklyje, yra kitos vyk- domos komandos adresas. Kai kurios procesoriaus komandos (nukreipimo komandos) keičia **eip** turinį, t.y. keičia komandų vykdymo tvarką.

Vėliavėlių registras. Vėliavėlių registro bitai valdo procesoriaus darbą arba atspindi kai kurių procesoriaus komandų rezultatus. Kai kurios ko- mandos gali patikrinti arba nustatyti atskiras vėliavėles. IA-32 vėliavėlių registras parodytas 3.7 pav.



3.7 pav.: IA-32 vėliavėlių registras

Valdymo vėliavėlės. Valdymo vėliavėlės valdo procesoriaus veikimą. Pa- vyzdžiui, gali po kiekvienos komandos įvykdymo pertraukti procesoriaus darbą (naudojama programų derinimo priemonėms kurti), nutraukti proce- soriaus darbą, įvykus aritmetiniam perpildymui, įjungti virtualųjį-8086 re- žimą, įjungti apsaugotąjį režimą. Kai kurių vėliavėlių, pavyzdžiui Krypties (**DF**) ar Pertraukties (**IF**) vėliavėlių reikšmės galima nustatyti programo- je. Būsenos vėliavėlės atspindi procesoriaus atliekamų aritmetinių ir loginių operacijų rezultatų požymius. Tai Perpildymo (**OF**), Ženklo (**SF**), Nulio (**ZF**), Pagalbinė perkėlimo (**AC**), Pariteto (**PF**), ir Pernešimo (**CF**) vėlia- vėlės.

Architektūroje AI-32 slankaus taško duomenims apdoroti, multimedijos programoms optimizuoti yra specialios operacijos ir registrai: aštuoni 80 bitų slankaus taško registrai, aštuoni 64 bitų MMX registrai ir aštuoni 128 bitų XMM registrai.

x86 atminties valdymas

Skirtinguose darbo režimuose x86 procesoriuose atmintis valdoma skirting- gai. Apsaugotasis režimas yra patikimiausias ir efektyviausias, tačiau šiame režime taikomosios programos negali tiesiogiai prieiti prie kompiuterio apa- ratūros.

Realus adreso režime gali būti adresuojama tik 1 MB atminties, šešiolik- tainiai adresai nuo 00000h iki 0FFFFFFh. Procesorius gali vykdyti tik vieną

programą vienu metu, bet jos vykdymas gali būti pertrauktas išorinio įrenginio užklausa (vadinama pertrauktimi). Taikomajai programai leidžiama prieiti prie bet kurios atminties vietos, įskaitant adresus, kurie yra tiesiogiai susieti su aparatine įranga. MS-DOS operacinė sistema veikia realaus adreso režime.

Apsaugotame režime, procesorius gali vykdyti kelias programas tuo pačiu metu. Jis skiria kiekvienam procesui (vykdomai programai) iki 4 GB atminties. Kiekvienai programai paskiriama sava saugomos atminties sritis, ir programos negali gauti prieigą prie viena kitos kodo ir duomenų. MS Windows ir Linux veikia apsaugotame režime.

Kai kompiuteris veikia *saugaus režimo virtualiame 8086 režime*, sukuriamą virtuali 8086 mašina su savo 1 MB adresų erdve, kuri imituoja 80x86 kompiuterį, veikiantį realaus adreso režimu. Pvz., Windows NT, kai atidaromas komandinės eilutės langas Command, sukuria virtualią 8086 mašiną. Galima paleisti daug tokių langų vienu metu, ir kiekvienas iš jų yra apsaugotas nuo kitų veiksmų. Kai kurios MS-DOS programos, kurios tiesiogiai naudoja kompiuterio aparatinę įrangą, nebus vykdomos šiame režime Windows NT sistemoje.

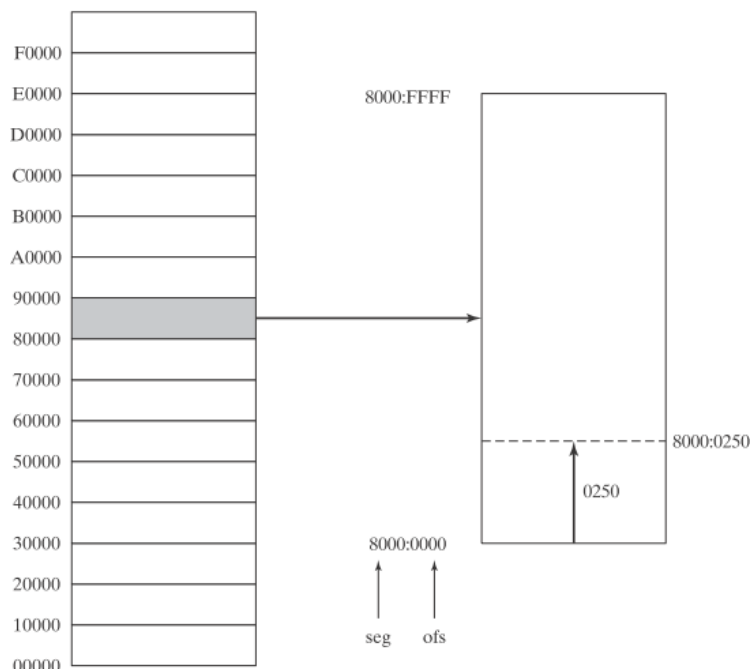
Realaus adreso režimas

Realaus adreso režime x86 procesorius gali naudotis iki 1.048.576 baitų atminties (1 MB), naudojant 20 bitų adresus intervale nuo 00000H iki 0FFFFFFH (šešioliktainiai skaičiai). Konstruojant procesorių, reikėjo išspręsti pagrindinę problemą: procesoriaus 8086 16 bitų registruose negalėjo būti 20 bitų adresai. Buvo sukurtas segmentinės atminties modelis. Visa atmintis yra padalinta į 64 kilobaitų (64 KB) segmentai, kaip parodyta 3.8 pav.

Kaip matyti 3.8 pav., kiekvienas segmentas prasideda adresu, kurio paskutinis šešioliktainis skaitmuo nulis. Kadangi paskutinis skaitmuo visada lygus nuliui, tai galima jį praleisti formuojant segmentų pradžios adresus. Pvz., C000 reiškia segmentą adresu C0000. Tame pačiame paveiksle parodytas segmento 80000 išplėtimas. Baitui šiame segmente nurodyti reikia pridėti 16 bitų poslinkį (nuo 0 iki ffffh) prie šio segmento bazinės (pradinio adreso). Pavyzdžiui, adresas 8000:0250 nurodo baitą poslinkiu 250 segmento viduje, kurio pradžios adresas 80000. Linijinis (fizinis, absoliutinis) adresas bus 80250h.

Realaus adreso režime linijinis (fizinis) adresas formuojamas iš dviejų komponentų: 16 bitų segmento bazinio adreso ir 16 bitų poslinkio segmente. Segmentų baziniai adresai yra segmentų registruose (**cs**, **ds**, **es**, **ss**, **fs** ir **gs**), o poslinkis - adresų poslinkių registruose (**ip**, **bx**, **si**, **di**, **bp**, **sp**) arba nurodomas tiesiogiai mašininėje komandoje, žr. 3.9 pav.

Procesorius automatiškai iš segmento adreso ir poslinkio formuoja 20 bitų fizinį adresą:



3.8 pav.: Realus adreso režimo atminties modelis

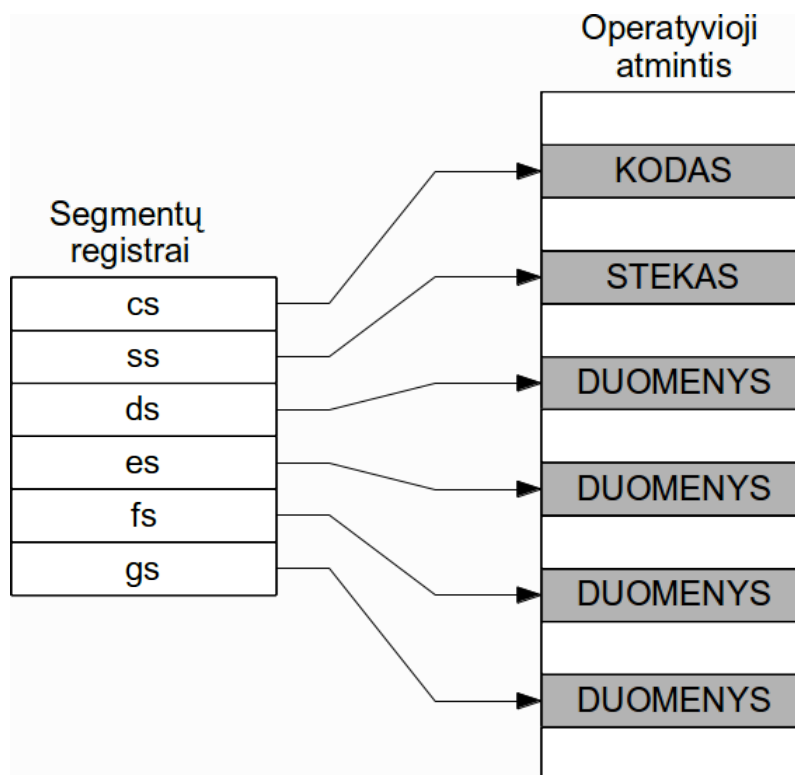
$$\text{Fizinis adresas} = [\text{segmento adresas}] \times 10h + [\text{poslinkis}].$$

Fizinio adreso formavimo schema realaus adreso režime pateikta 3.10 pav.

Apsaugotas režimas

Apsaugotas režimas yra efektyviausias procesoriaus režimas. Kai programa vykdoma apsaugotame režime, jos tiesinė (fizinė) adresų erdvė yra 4 GB, adresai 0 iki FFFFFFFF. Šiuo modeliu, vadinamu plokščiuoju atminties modeliu, yra lengva naudotis, nes reikia tik vieno 32 bitų sveikąjį skaičių komandos ar duomenų adreso nurodymui, 3.11 pav. Procesorius adresus skaičiuoja ir transliuoja fone, visa tai yra nematoma programuotojams. Segmentų registrai (**cs**, **ds**, **ss**, **es**, **fs**, **gs**) nurodo į segmentų deskriptorių lentelę, kurią operacinė sistema naudoja atskirų programos segmentų valdymui. Tipiška apsaugoto režimo programa turi tris segmentus: kodo, duomenų ir dėtuves, naudojami **cs**, **ds**, ir **ss** registrai:

- **cs** rodo į kodo segmento deskriptoriaus lentelę,
- **ds** rodo į duomenų segmento deskriptoriaus lentelę,
- **ss** rodo į dėtuves segmento deskriptoriaus lentelę.



3.9 pav.: Realaus adreso režimo atminties modelis, segmentų registrai

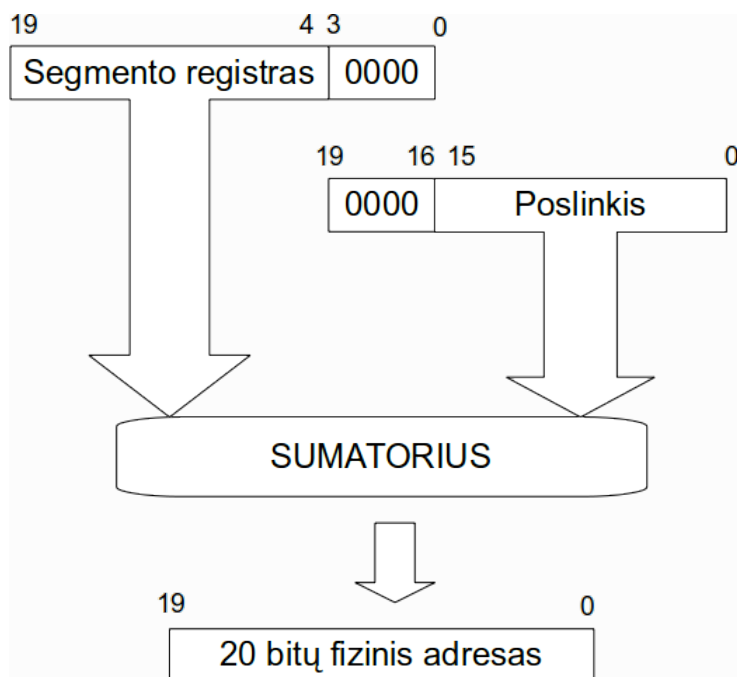
IA-32 architektūroje yra dar keli atminties valdymo būdai: plokščias segmentinis modelis, daugia-segmentinis modelis ir virtualios atminties modelis.

3.4 I8086 architektūros ypatybės

Toliau mes nagrinėsime taip vadinamą bazinę architektūrą, t.y. 16 bitų 8086 mikroprocesorių.

Vietinės mikroprocesoriaus 8086 atminties struktūra, registrai, parodyta 3.12 pav. Čia išskirtos trys duomenų ir adresų registrų grupės ir du papildomi registrai - komandos rodiklis ir vėliavėlių (požymių) registras.

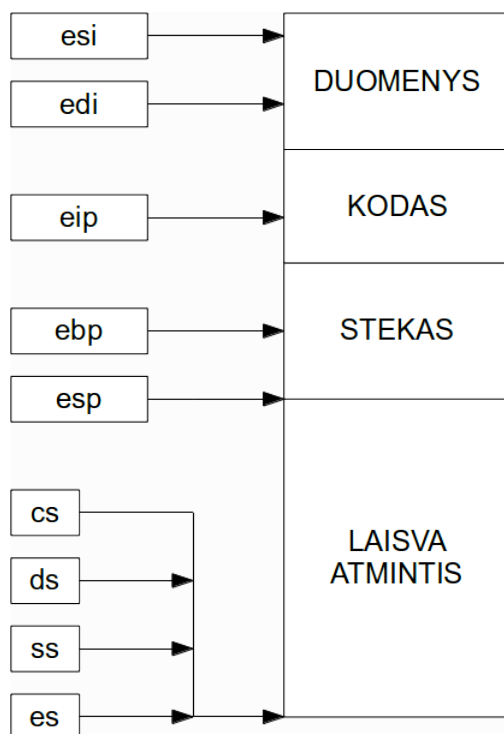
Duomenų registrai skirti programuoti su 16 skilčių žodžiais, galima disponuoti 4 registrais: **ax**, **bx**, **cx**, **dx**. Dirbant su baitais kiekvienas registras pasidalina į dvi dalis ir tokiu būdu leidžia dirbti su aštuoniais vieno baito ilgio registrais: **ah**, **bh**, **ch**, **dh**, **al**, **bl**, **cl**, **dl**. Čia raidė *L* reiškia jaunesnįjį (*angl.* - *Low*), o *H* - vyresnįjį (*angl.* *High*) žodžio baitą. Reikia įvertinti, kad išvardinti duomenų registrai kai kuriose komandose naudojami, neparodant to tiesiogiai komandos apraše:



3.10 pav.: Realus adreso režimo atminties modelis fizinio adreso formavimas

- **ax** (*angl. Accumulator register*) (akumuliatorius) registras naudojamas vykdant žodžių daugybą ir dalybą, įvedimo - išvedimo operacijas ir kai kurias operacijas su eilutėmis;
- **al** registras naudojamas atliekant analogiškas operacijas su bitais bei atliekant aritmetines operacijas su skaičiais;
- **ah** registras naudojamas baitų daugyboje ir dalyboje;
- **bx** (*angl. Base register*) bazinis registras dažnai naudojamas duomenims adresuoti atmintyje;
- **cx** (*angl. Count register*) skaitliuko registras naudojamas ciklinėse operacijose;
- **cl** registras naudojamas postūmio konstantai nurodyti;
- **dx** (*angl. Data register*) duomenų registras naudojamas daugybos ir dalybos operacijose. Įvedimo - išvedimo operacijose naudojamas formuojant atitinkamus įvedimo - išvedimo įrenginių adresus.

Segmentų registrai siejasi su tuo, kad programos kodas ir duomenys saugomi atskirose atminties srityse ir kiekvienas tokios srities dydis neturi vir-



3.11 pav.: Apsaugoto režimo plokščias atminties modelis

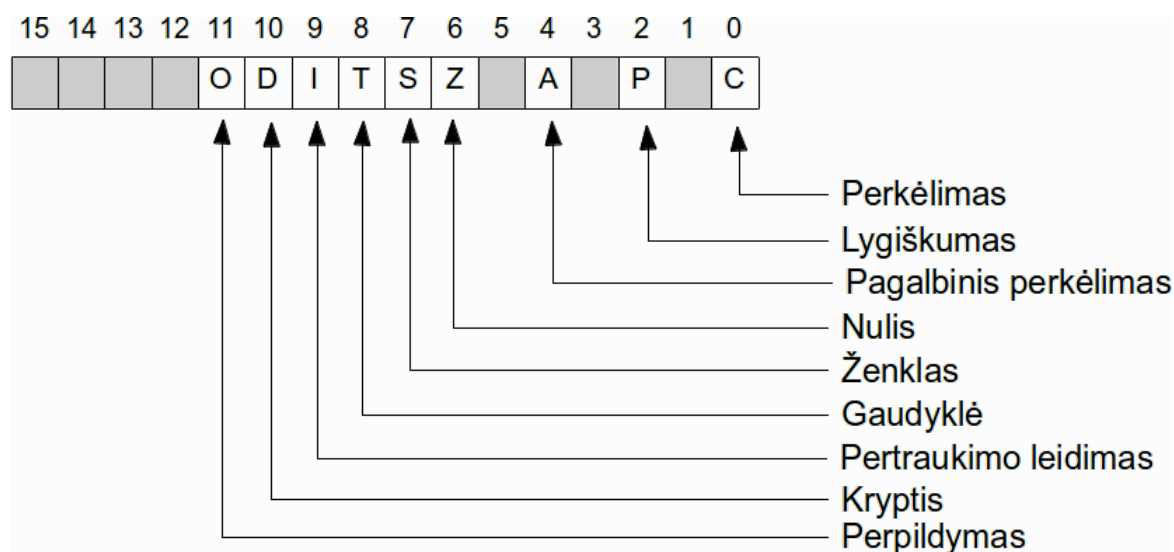
šyti 64 Kbaitų (1 Kilobaitas = 1024 Baitų, 1 Megabaitas = 1024 Kbaitų). Mikroprocesorius vienu metu gali būti susietas su keturiais segmentais. Kiekvieno jų pradiniai adresai yra atitinkamuose segmentų registruose. Šių registrų funkcijos tokios:

- **cs** (*angl. Code Segment*) komandų segmento registras nurodo programos kodo segmento pradžią;
- **ss** (*angl. Stack Segment*) steko segmento registras nurodo steko pradžią;
- **ds** (*angl. Data Segment*) duomenų segmento registras nurodo duomenų segmento pradžią;
- **es** (*angl. Extra Segment*) papildomo segmento registras nurodo papildomo segmento pradžią;

Indeksiniai registrai **si** (*angl. Source index*) ir **di** (*angl. Destination index*) dažniausiai naudojami išrenkant operandus iš atminties, formuojant jų fizinius adresus. Registrai - rodikliai **sp** (*angl. Stack pointer*) ir **bp** (*angl. Base pointer*) naudojami dirbant su steku.

Komandų rodiklis **ip** (*angl. Instruction pointer*) rodo į sekančios komandos adresą programos segmente. Programos vykdymo metu **ip** turinys keičiasi ir visada rodo į sekančią vykdomą komandą.

Vėliavėlių registras saugo informaciją apie mikroprocesoriaus padėtį ir rezultato požymius. Tai šešiolikos skilčių registras, kurio skilčių paskirstymas parodytas 3.13 pav. Jame realizuotos 6 vieno bito vėliavėlės, nusakančias aritmetinių ir loginių komandų įvykdymo rezultatus ir 3 papildomos valdymo vėliavėlės, kurias galima nustatyti programiškai.



3.13 pav.: Procesoriaus vėliavėlių registras

- **CF** (*angl. Carry flag*) perkėlimo vėliavėlė. Nurodo 1 perkėlimą iš vyriausios skilties aritmetinių operacijų metu.
- **PF** (*angl. Parity flag*) Lygiškumo (pariteto) vėliavėlė. Parodo, kad rezultatas turi lyginį vienetų skaičių.
- **AF** (*angl. Adjust flag*) Pagalbinio perkėlimo vėliavėlė. Naudojama aritmetinėse operacijose su baitais. Indikuoja perkėlimą iš jaunesnės tetrados į vyresniąją.
- **ZF** (*angl. Zero flag*) Nulio vėliavėlė. Indikuoja nulinį rezultatą.
- **SF** (*angl. Sign flag*) Ženklo vėliavėlė. Atitinka rezultato vyriausios (ženklų) skilties turinį.
- **TF** (*angl. Trap flag*) Vėliavėlė - gaudyklė. Dažniausiai naudojama organizuojant žingsninį darbą. Jei **TF** nustatyta į vienetą, mikroprocesorius po kiekvienos komandos įvykdymo stabdomas.

- **IF** (*angl. Interrupt flag*) Pertraukimo leidimo vėliavėlė. Naudojama programose, kur reikia leisti arba drausti pertraukimus. $IF = 0$ vėliavėlės reikšmė draudžia pertraukimus. Pertraukimo signalas gali ateiti iš išorinio įrenginio arba gali būti suformuojamas programoje. Jei vėliavėlė **IF** leidžia, atėjus atitinkamam signalui automatiškai pereinama į pertraukimo apdorojimo programą, o po to grįžtama vykdyti pertrauktą programą. Pvz.: pertraukimo metu gali būti įvestas simbolis iš kompiuterio klaviatūros.
- **DF** (*angl. Direction flag*) Krypties vėliavėlė. Dažniausiai naudojama kartu su eilučių apdorojimo komanda.
- **OF** (*angl. Overflow flag*) Perpildymo vėlevėlė. Fiksuoja perpildymo atvejį kai prarandama reikšminė skiltis, dirbant su baitais arba žodžiais.

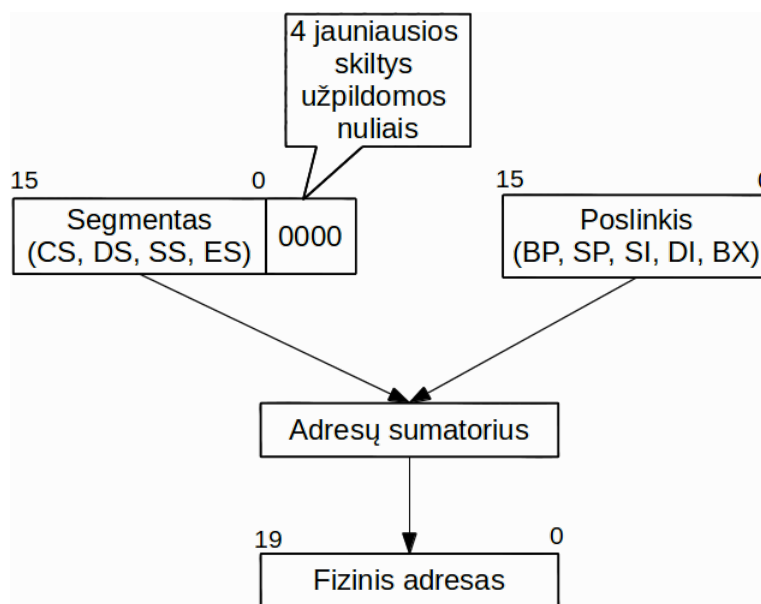
Akivaizdu, kad registrai sudaro tik nedidelę mikroprocesoriaus struktūros dalį. Tai elementai, kurių turinį gali priimti ir keisti programuotojas. Kitų mikroprocesoriaus mazgų darbą programuotojas valdo netiesiogiai - užduodamas vykdyti konkrečias komandas, kurias įvykdo aritmetinis - loginis įrenginys, informaciją tarp registrų ir išorės perduoda atitinkamos magistralės ir t.t.

Programuojant assemblerio kalboje detaliam išsiaiškinti kitų mikroprocesoriaus mazgų darbo programuotojas nėra būtina.

Mikroprocesoriaus 8086 atminčiai adresuoti reikia 20 bitų magistralės. Komandų registras **ip** ir visi kiti registrai turi tik 16 bitų, Mikroprocesorius negali iš karto tiesiai kreiptis į bet kurią atminties vietą. Todėl visa atmintis yra suskirstyta į segmentus. Vartotojas vienu metu gali dirbti tik su savo segmentais. Kitose atminties dalyse gali būti saugomos įvairios sistemos ir kitų vartotojų programos.

Kiekvienas atminties segmentas gali apimti iki 64 Kbaitų (65536). Segmento pradinis adresas nustatomas vartotojo programoje ir jis visada prasižada nuo 16 baitų ribos.

Mikroprocesorius 8086 realų fizinį adresą formuoja tokiu būdu, kaip tai parodyta 3.14 pav. Taip formuojant adresą adresuojama erdvė yra 1 MB.



3.14 pav.: Fizinio adreso formavimas

Atminties segmentas gali užimti nuo 16 B iki 64 KB. Segmentas gali būti šiose ribose bet kokio ilgio. Pradinis segmento adresas būtinai turi dalintis iš 16.

Programai ir duomenims skiriami segmentai iki 64 KB ir formuojant tokios apimties programas problemų paprastai neiškyla. Jei programa ar duomenys netelpa į 64 KB juos tenka išdėstyti keliuose segmentuose. Tada programos viduje turi būti atitinkama komandos perėjimui iš vieno segmento į kitą.

Mažiausias adresuojamas vienetas atmintyje yra baitas. **Duomenys atmintyje talpinami pradedant jaunesniojo duomenų baitu, t.y. jaunesniojo adresu - jaunesnis domenų baitas.** 16 bitų žodis į atmintį visada įrašomas taip, kad vyresnysis baitas turi vienetu didesnę adresą, t.y. baitai atmintyje talpinami iš kito galo. Pavyzdžiui, jei šešioliktainis skaičius 0A79H turi būti įrašytas 3100H - 3101H baituose, tai 3100H baitas bus 79H, o 3101H - 0AH.

	15		0	Žodžio registro bitai
	7	0 7	0	Baito registrų bitai
AX	AH	AL		Akumulatorius
BX	BH	BL		Bazinis registras
CX	CH	CL		Skaitliukas
DX	DH	DL		Duomenų registras
	SP			Steko rodiklis
	BP			Bazės rodiklis
	SI			Šaltinio rodiklis
	DI			Imtuvo rodiklis
	CS			Komandų segmentas
	DS			Duomenų segmentas
	SS			Steko segmentas
	ES			Papildomas segmentas
	IP			Komandos skaitliukas
				Vėliavėlių registras

3.12 pav.: Procesoriaus vietinė atmintis

skyrius 4

Asemblerio kalbos pagrindai

Sąvoka assembleris turi 2 reikšmes: tai programavimo kalba ir šios kalbos transliatorius į mašininę kalbą. Kadangi programuojant galima vartoti ne tik simbolines assemblerio komandas, bet ir makrokomandas, dažnai assembleris vadinamas makroassembleriu. Pagrindinis šios programavimo kalbos bruožas tas, kad kiekviena simbolinė assemblerio komanda transliuojant pakeičiama mašinine komanda atitinkančia elementarią instrukciją procesoriui. Iš makrokomandų gali būti gauta viena arba kelios simbolinės assemblerio komandos.

Assemblerio kalba vartojama tada, kai reikia sudaryti trumpą ir greitai veikiančią programą bei maksimaliai išnaudoti kompiuterio galimybes.

4.1 Operatoriaus struktūra

Makroassemblerio kalba parašytą programą sudaro simbolių eilutės, vadinamos operatoriais. Paprastai operatorių sudaro 4 dalys, atskirtos viena nuo kitos specialiais skyrybos ženklais:

$$\left[\begin{array}{l} \textit{vardas} \\ \textit{žymė:} \\ \textit{kintamasis} \end{array} \right] \textit{operacija} [\textit{operandas}] [; \textit{komentaras}]$$

Čia laužtiniuose skliaustuose nurodyta informacija kartais gali būti nevartojama.

Žymė. Parašyta programa ir duomenys laikomi operatyviojoje atmintyje (OA). Kreipiantis į kurį nors programos operatorių, reikėtų nurodyti atitinkamą OA adresą. Tačiau tai būtų labai sudėtinga. Todėl tiems programos operatoriams arba duomenims, į kuriuos kreipiamės, vietoje adresų nurodomi simboliniai vardai, užrašyti operatoriaus pradžioje.

Kintamasis. Duomenis identifikuojantys simboliniai vardai vadinasi kintamaisiais, o assemblerio komandos - žymėmis. Kiekviena žymė baigiasi ":" (po kintamųjų ":" nerašomas).

Simboliniai vardai gali būti sudaryti iš alfabetinių - skaitmeninių neri-boto skaičiaus simbolių bei ženklų " , ", "?", " _ " eilučių, tačiau reikšminiai yra tik pirmi 31. Jeigu kelių simbolių vardų pirmieji 31 simboliai sutampa, tai laikoma, kad vardai yra vienodi. Nors kiti vardų simboliai ir skirtingi. Vienodų simbolių vardų programoje negali būti, tačiau vieną duomenų lauką gali identifikuoti keli kintamieji, taip pat ir vieną assemblerio komandą gali identifikuoti kelios žymės.

Vardai - tai simboliniai žodžiai, apibrėžiami direktyva **EQU** ir turintys simbolinę arba skaitinę reikšmę (tai panašu į konstantų aprašymą *Pascal* kalboje).

Operacija. Operacijos lauke gali būti nurodyta simbolinė assemblerio komanda, makrokomanda arba transliatoriaus direktyva. Tarpai (tušti simboliai) prieš šį lauką arba po jo neturi jokios reikšmės.

Makrokomandos. Iš assemblerio komandų galima sudaryti kitas (paprastai sudėtingesnes) komandas, vadinamas makrokomandomis, ir vartoti jas programavimo palengvinimui. Duomenims, programoms, makrokomandoms aprašyti ir transliavimo procesui valdyti vartojamos transliatoriaus direktyvos.

Operandai. Jeigu operacijos lauke nurodoma komanda, tai operandų lauke nurodomi kintamieji, su kuriais reikia atlikti nurodytus veiksmus. Kai operacijos lauke nurodyta makrokomanda ar transliatoriaus direktyva, tai operandų lauke nurodomi reikiami argumentai. Nurodyti keli operandai, vienas nuo kito atskiriami kableliais.

Komentarai visada prasideda kabliataškiu ir užima likusią eilutės dalį. Paaiškinimai gali užimti ir visą eilutę. Komentarai neturi jokios reikšmės nei programos vykdymui, nei transliavimo procesui, jie tik paaiškina programą.

4.2 Programos struktūra

Programa paprastai (bet ne būtinai) sudaroma iš trijų segmentų: steko, duomenų ir kodo. Segmentai aprašomi direktyvomis **SEGMENT** (segmento pradžia) ir **ENDS** (segmento pabaiga). Taip pat būtina nurodyti assemblerio transliatoriui kokius segmentus bei registrus vartoti, transliuojant kreipinius į duomenis ir komandas. Tai nurodoma direktyva **ASSUME**. Programa užbaigiama direktyva **END**. Išraiška, jeigu ji užrašyta, nurodo pirmą vykdomą programos komandą. Jei programa sudaryta iš kelių modulių tai tik viename reikia nurodyti programos pradžią. Programos pavyzdys pateiktas žemiau:

```
stekas      SEGMENT stack
              ; atminties rezervavimas stekui
stekas      ENDS
duom        SEGMENT
              ; duomenų aprašymas
duom        ENDS
```

```

kodas      SEGMENT
            ASSUME cs:kodas, ds:duomenys, ss:stekas
pradzia:    ; programa

kodas      ENDS
            END pradzia

```

4.3 Duomenų aprašymas ir atminties laukų rezervavimas

Prieš apdorojant informaciją, į operatyviąją atmintį įrašomi žinomi dydžiai bei rezervuojama vieta skaičiavimo rezultatams.

Asembleryje pradiniai duomenys nurodomi bei atmintis rezervuojama aprašant kintamuosius. Kintamųjų aprašymo direktyvomis nurodoma duomenų laukų dydis, turinys, tipas ir vieta programoje. Šiomis direktyvomis galima aprašyti skaliarinius duomenis, įrašus - bitu rinkinius ir struktūras - baitų sekas, atspindinčias tam tikrą loginę duomenų struktūrą. Kintamųjų aprašymui vartojamos direktyvos:

$$[\text{kintamasis}] \left\{ \begin{array}{c} DB \\ DW \\ DD \\ DQ \\ DT \end{array} \right\} \text{ reikšmė}$$

Kintamųjų aprašymo direktyvos atmintį išskiria elementais, kurių ilgį apibrėžia antroji direktyvos raidė:

B apibrėžia baitus (*angl. Byte*);

W apibrėžia 2 baitų ilgio žodžius (*angl. Word*);

D apibrėžia 4 baitų ilgio elementus (dvigubus žodžius) (*angl. Double word*);

Q apibrėžia 8 baitų ilgio elementus (*angl. Quad word*);

Operandų lauke užrašomos kableliais atskirtos viena ar kelios išraiškos, kurios gali būti:

- skaitinės išraiškos,
- '?' simbolis, vartojamas atminties rezervavimui,
- adresinės išraiškos (tik **DW** arba **DD**),
- simbolių eilutės (iki 255 simbolių), simbolių eilutės rašomos tarp apostrofų arba kabučių,

- kartojama skaitinė išraiška *kiekis DUP (reikšmė...)*. Išraiška prieš **DUP** nurodo operandų, esančių skliaustuose, kartojimo skaičių. Skliaustuose gali būti bet kuri išraiška, taip pat ir kita **DUP** tipo išraiška.

Priimta, kad visi įsirašyti skaičiai pateikiami 10-aine skaičiavimo sistema. Norint pakeisti sistemos pagrindą reikia aukščiau užrašyti direktyvą. **RADIX N**, čia *N* - bet kuris skaičius nuo 2 iki 16 - skaičiavimo sistemos pagrindas. Užrašant skaičius, taip pat galima nurodyti skaičiavimo sistemos pagrindą. Šiuo atveju skaičiavimo sistemos pagrindas nurodomas raide rašoma po skaičiaus: B - dvejetainis, 0 arba Q - aštuntainis, D - dešimtainis, H - šešioliktainis. Šešioliktainis skaičius turi prasidėti dešimtainiu skaitmeniu. Duomenų parašymo pavyzdžiai pateikti žemiau:

```

sb      DB 'EILUTE' ; 6 baitų ilgio
                ; simbolių eilutė
s2      DB 'E', 'I', 'L', 'U', 'T', 'E'
                ; ta pati eilutė
n1      DB 55      ; 1 baito ilgio kintamasis
n2      DB 37H     ; tas pats skaičius
                ; šešioliktainėje sistemoje
n3      DW 0FH     ; 2 baitų ilgio kintamasis
n4      DW 170     ; tas pats skaičius
                ; aštuntainėje sistemoje
a1      DB 5,7,23,0 ; 4 vienbaitių skaičių masyvas
a2      DW 10 DUP(?) ; rezervuojamas 10 žodžių laukas
a3      DB 20-DUP (0FH) ; sukuriamas 20 baitų
                ; laukas, užpildytas konstantomis
n1a     DW n1      ; kintamojo n1 adresas
n5      DW 55+10   ; kintamojo n5 reikšmė yra 65
a4      DB 2(DUP(2, 3 (DUP (1)))) ; bus sugeneruota
                ; tokia seka: 2,1,1,1,2,1,1,1

```

4.4 Paprasčiausi operandų adresavimo būdai

Asemblerio komandoje nurodoma ne tik kokią operaciją reikia atlikti, bet ir su kokiais duomenimis ją vykdyti ir kur užrašyti rezultatą. Duomenys arba rezultatai gali būti vietinėje (procesoriaus registruose) arba operatyviojoje atmintyje.

Paprasčiausia nurodyti operandus vietinėje atmintyje registru: **ax**, **bx**, **ah**, **al** ir t.t. Toks adresavimo būdas vadinamas registriniu adresavimu. Šiuo atveju duomenys yra registre. Pavyzdžiui, kad perrašyti informaciją iš registro **bx** į registrą **ax**, rašoma komanda **MOV**, o operandai nurodomi taip:

```
MOV    ax, bx        ; reg. bx turinys
                        ; persiunčiamas į reg. ax
```

Kai operandas yra konstanta, patogų duomenis užrašyti betarpiškai komandoje. Toks adresavimo būdas vadinamas betarpišku adresavimu. Pavyzdžiui:

```
MOV    ax, 10         ; konstanta 10 persiunčiama
                        ; į reg. ax
MOV    dl, '='         ; simbolio '=' ASCII kodas
                        ; persiunčiamas į reg. dl
MOV    dx, 'NE'        ; simboliai 'N' ir 'E'
                        ; užrašomi į reg. dx
```

Nurodyti operandams esantiems operatyviojoje atmintyje yra keletas būdų. Paprasčiausia adresuoti žodžius ar baitus operatyviojoje atmintyje, nurodant jų adresus simboliniais vardais. Toks adresavimo būdas vadinamas tiesioginiu adresavimu. Pavyzdžiui:

```
a1     DB 1           ; aprašome kintamuosius
b       DW 20          ; ir rezervuojame
d       DB ?           ; atminti

MOV     al, a1 ; atminties baito a1 turinys
                ; į reg. al
MOV     d, bh ; reg. bh turinys
                ; į atminties baitą d
MOV     dx, b ; žodžio b turinys
                ; persiunčiamas į registrą dx
```

Dažnai tenka apdoroti indeksuotus duomenis, pavyzdžiui, masyvo elementus. Šiuo atveju patogų vartoti indeksuotą operandą, t.y. nurodyti masyvo pradžios adresą ir elemento indeksą. Indeksui nurodyti vartojami registrai **si** ir **di**. Pavyzdžiui:

```
mas     DB 10,20,30,40,50 ; skaičių masyvas
        .
        .
MOV     si, 0
MOV     ah, mas[si] ; į registrą ah persiunčiamas
                    ; 1-sis elementas (10)
MOV     si, 3
MOV     dh, mas[si] ; į registrą dh persiunčiamas
                    ; 4-sis elementas (40)
```

skyrius 5

Mikroprocesoriaus *I8086* komandų sistema

Mikroprocesoriaus *I8086* komandos skirstomos į tokias grupes:

1. duomenų persiuntimo komandos;
2. aritmetinių operacijų komandos;
3. bitų apdorojimo komandos;
4. eilučių apdorojimo komandos;
5. nukreipimo komandos;
6. procesoriaus valdymo komandos.

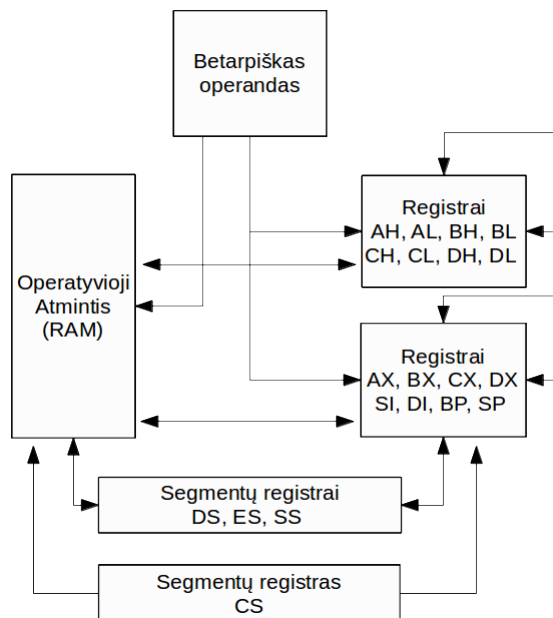
Komandose nurodoma du, vienas arba nei vieno operando. Jei komandoje reikia nurodyti du operandus, tai tik vienas iš jų gali būti atminties operandas ir abiejų operandų ilgiai turi sutapti,

5.1 Duomenų persiuntimo komandos

Duomenų persiuntimo komandos vėliavėlių registro bitų nekeičia. Komanda **MOV** yra pagrindinė duomenų persiuntimo komanda. Ji gali persiusti duomenų baitą arba žodį iš atminties į registrą, iš registro į atmintį arba iš registro į registrą. Komanda **MOV** taip pat galima įrašyti į registrą arba atmintį duomenis, nurodytus komandoje. Komandos **MOV** duomenų persiuntimo kryptys parodytos 5.1 pav.

Komanda **MOV** galima persiusti ne tik duomenis, bet ir atminties lauko adresą:

MOV bx , OFFSET kint.
--



5.1 pav.: MOV komandos persiuntimo kryptys

Prefiksas **OFFSET** nurodo, kad į registrą reikia užkrauti lauko *kint* adresą. Komanda **LEA** persiunčia į registrą atminties lauko adresą,

```
LEA    bx, kint
```

Jai ekvivalentiška būtų komanda

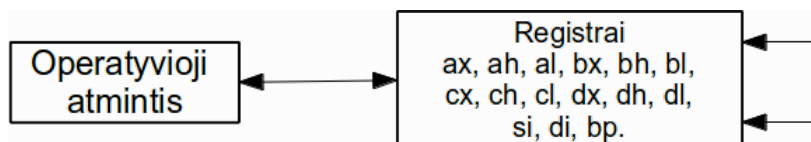
```
MOV    bx, OFFSET kint
```

Tačiau komanda **LEA** yra lankstesnė. Ja galima persiųsti bet kurio lauko adresą. Pavyzdžiui jeigu reg, **di** yra duomenų masyvo adresas, tai komanda

```
LEA    bx, 10[di]
```

į reg, **bx** persiųs dešimtojo masyvo elemento adresą.

Komanda **XCHG** sukeičia vietomis dviejų operandų turinius.



Pavyzdžiui, komanda

```
XCHG    dx,  bx
```

analogiška tokiai komandų sekai

```
MOV    ax, bx
MOV    bx, dx
MOV    dx, ax
```

Komanda **PUSH** į steką įrašo žodžio ilgio duomenis, nurodytus operande. Vykdam šią komandą, steko rodiklis **sp** sumažinamas 2 ir į steko viršūnę įrašomi duomenys. Komandos formatas:

```
PUSH    operandas
```

Komanda **POP** skaito iš steko žodžio ilgio duomenis ir persiunčia juos į operandą. Vykdam šią komandą, iš steko viršūnės skaitomi duomenys ir steko rodiklis **sp** padidinamas 2. Pavyzdžiui:

```
PUSH    bx
PUSH    cx
.
.
POP     bx
POP     cx
```

Įvykdžius šias komandas, registrų **bx** ir **cx** turiniai bus sukeisti vietomis, norint atstatyti senuosius registrų turinius reikėtų įvykdyti tokią komandų seką:

```
POP     cx
POP     bx
```

5.2 Aritmetinių operacijų komandos

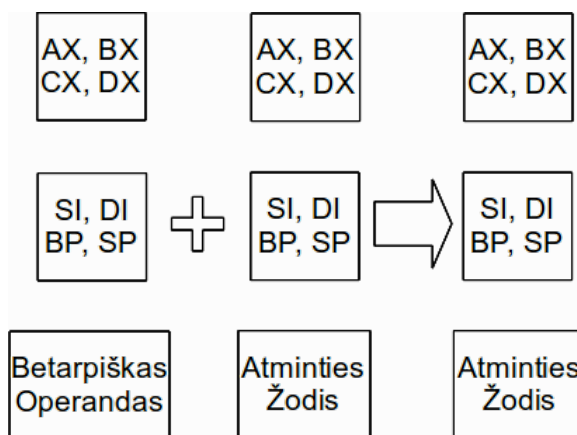
Mikroprocesoriaus *I8086* komandų sistemoje yra tik sveikųjų skaičių aritmetikos komandos, t.y. aritmetines operacijas galima atlikti tik su fiksuoto taško skaičiais (su ženklu ir be ženklo). Operacijos su slankaus taško skaičiais galima atlikti programiškai arba vartojant koprosorių *I8087*. Dauguma aritmetinių komandų keičia vėliavėlių registro bitų reikšmes.

Sudėties ir atimties komandos

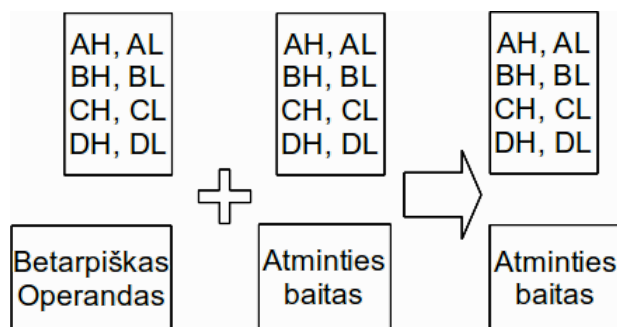
Sudėties komanda **ADD** sudeda du dvejetainius skaičius, nurodytus operanduose ir rezultatą patalpina pirmo operando vietoje:

```
ADD    op1, op2    ; op1 := op1 + op2
```

Galimi tokie operandų deriniai žodžio ilgio operandams sudėti:



Galimi tokie operandų deriniai baido ilgio operandams sudėti:



Pavyzdžiui:

```

a      DW 12
.
.
ADD    ax, bx      ; ax := ax + bx
ADD    dx, a       ; dx := dx + 5
ADD    ah, 5       ; ah := ah + 5

```

Sudėties su perkėlimu komanda **ADC** sudeda du operanduose nurodytus skaičius ir jei prieš atliekant komandą buvo nustatyta **CF:=1**, tai dar prie sumos pridedamas 1;

```

ADC    op1, op2    ; op1 := op1 + op2 + CF

```

Ši komanda vartojama ilgesnių nei 16 bitu skaičių sudėčiai.

Atminties komanda **SUB** iš skaičiaus, nurodyto pirmame operande, atima skaičius, nurodyta antrame operande, ir rezultatą patalpina pirmo operando vietoje:

SUB	<i>op1</i> , <i>op2</i>	; <i>op1</i> := <i>op1</i> - <i>op2</i>
------------	-------------------------	---

Atimties su perkėlimu (skolinimu) komanda **SBB** atlieka atimties operaciją ir jei prieš atliekant komanda buvo nustatyta **CF:=1**, tai iš rezultato dar atimamas 1:

SBB	<i>op1</i> , <i>op2</i>	; <i>op1</i> := <i>op1</i> - <i>op2</i> - <i>CF</i> ,
------------	-------------------------	---

Ši komanda vartojama ilgesnių nei 16 bitų skaičių atimčiai.

Palyginimo komanda **CMP** atlieka atimties operaciją, bet rezultatas neįsimenamas o tik nustatomos atitinkamos vėliavėlės **AF**, **CF**, **OF**, **PF**, **SF**, **ZF**.

CMP	<i>op1</i> , <i>op2</i>	; <i>op1</i> - <i>op2</i> .
------------	-------------------------	-----------------------------

Ši komanda kartu su sąlyginio nukreipimo komandomis vartojama programuojant šakojimus algoritmuose.

Komandose **ADC**, **SUB**, **SBB** ir **CMP** galimi tokie pat operandų deriniai kaip ir komandoje **ADD**. Neigimo komanda **NEG** keičia operando ženklą priešingu t.y. operandas atimamas iš 0 ir rezultatas talpinamas operando vietoje:

NEG	<i>op</i>	; <i>op</i> := 0 - <i>op</i>
------------	-----------	------------------------------

Komanda **INC** padidina operandą vienetu:

INC	<i>op</i>	; <i>op</i> := <i>op</i> + 1.
------------	-----------	-------------------------------

Komanda **DEC** sumažina operandą vienetu;

DEC	<i>op</i>	; <i>op</i> := <i>op</i> - 1
------------	-----------	------------------------------

Daugybos ir dalybos komandos

Šios komandos sudarytos taip, kad vienas iš operandų būtinai yra akumuliatoriuje (registrai **ax** arba **al**), o komandoje nurodomas tik antrasis operandas (daugiklis arba daliklis). Rezultatas talpinamas registruose **ax** ir **dx**. Šiose komandose betarpišką operandą nurodyti negalima.

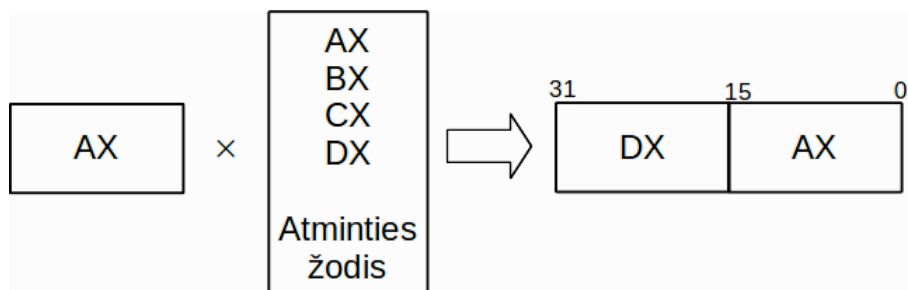
Sveikų skaičių, be ženklo daugybos komanda **MUL**:

MUL	<i>op</i>	; <i>akumuliatorius</i> := <i>akumuliatorius</i> * <i>op</i>
------------	-----------	--

Sveikų skaičių su ženklu daugybos komanda **IMUL**:

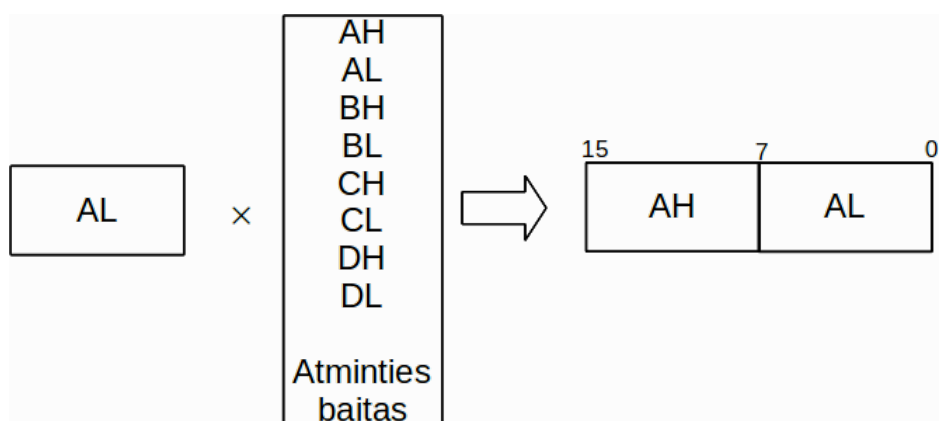
IMUL	<i>op</i>	; <i>akumuliatorius</i> := <i>akumuliatorius</i> * <i>op</i>
-------------	-----------	--

Daugybos komandose galimi tokie operandų deriniai (žodžiams):



Daugikliai yra 16 bitų ilgio, o sandauga - 32 bitų. Reg. **ax** talpinama jaunesnė, o reg. **dx** - vyresnė sandaugos dalis. Jeigu skaičiai be ženklų ir sandauga (65535, tai ji pilnai telpa reg. **ax**, o jei skaičiai su ženklu, tai sandauga reg. **ax** telpa, jeigu ji yra intervale nuo -32768 iki +32767.

Galimi tokie operandų deriniai baido ilgio operandams sudauginti:



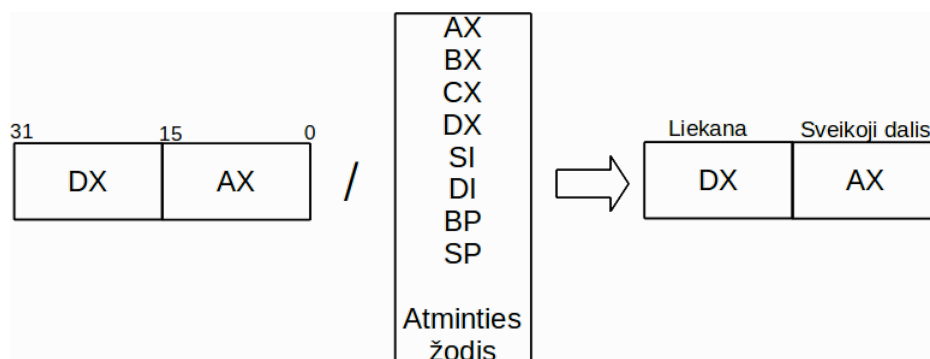
Daugikliai yra 8 bitų ilgio, o sandauga - 16 bitų. Reg, **al** talpinama jaunesnė, o reg. **ah** vyresnė sandaugos dalis. Jeigu skaičiai be ženklų ir sandauga 255, tai ji pilnai telpa reg. **al**, o jei skaičiai su ženklu, tai sandauga reg. **al** telpa, jeigu ji yra intervale nuo - 128 iki +127. Sveikų skaičių be ženklų dalybos komanda **DIV**:

DIV <i>op</i> ; <i>akumuliatorius</i> := <i>akumuliatorius</i> / <i>op</i> .

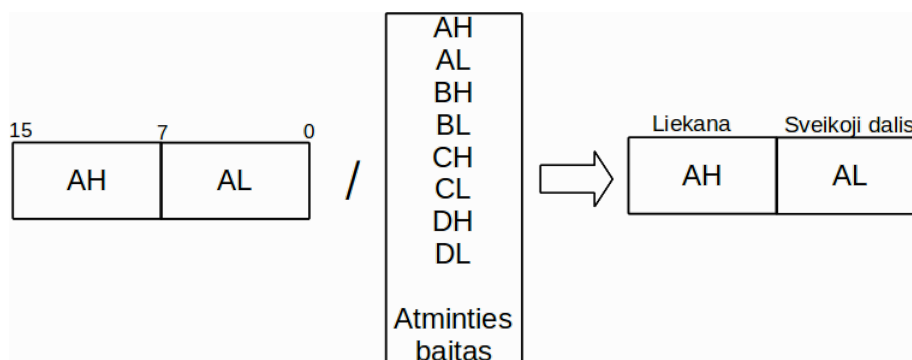
Sveikų skaičių su ženklu dalybos komanda **IDIV**:

IDIV <i>op</i> ; <i>akumuliatorius</i> := <i>akumuliatorius</i> / <i>op</i>
--

Operandų deriniai dalinant žodžius:



Operandų deriniai dalinant baitus:



Dalmuo turi tilpti reg **ax** (arba **al**), priešingu atveju gaunama klaida.

Dalyba iš nulio

Kaip matote dalinys yra dvigubo ilgio: jeigu operandas 1 baito, tai dalinys 2 baitų ilgio, o jeigu operandas 2 baitų, tai dalinys 4 baitų ilgio. Reikiamo ilgio dalinys gaunamas po daugybos operacijos.

Pavyzdys. Apskaičiuoti $y = (x * 5)/a$.

a	DW	2	
x	DW	-12	
y	DW	?	
.			
.			
MOV	ax, 5	; ax := 5	
IMUL	k	; dx:ax := -ax*x	(5*x)
IDIV	a	; ax := dx/cx/a	(5*x/a), dx - liekana
MOV	y, ax	; y := ax	
			(y := 5*x/a)

Dažniausiai prieš dalybos operaciją neatliekama daugybos operacija ir todėl reikia specialiai suformuoti reikiamo ilgio dalinį. Jeigu operacijos at-

Dešimtainis skaičius	1 baido ilgio operandas	2 baitų ilgio operandas (vyriausias bitas - ženklas)
+5	0000 0101	0000 0000 0000 0101
-5	1111 1011	1111 1111 1111 1011

liekamos su skaičiais be ženklo tada į reg. **dx** (arba **ah**) reikia įrašyti nulį. Tai galima atlikti pvz. komandomis **MOV** arba **SUB**:

```
SUB    ah, ah
MOV    dx, 0
```

Jeigu skaičiai su ženklu, tada vyresnį dalinio reg. **dx** (arba **ah**) Reikia užpildyti jaunesnio dalinio registro **ax** (arba **al**) ženklo bitu. Pvz.:

Šiai transformacijai atlikti yra specialios komandos. Komanda **CBW** transformuoja baitą į žodį: registras **ah** užpildomas vienbaitio skaičiaus reg. **al** ženklo bitu. Komanda **CWD** transformuoja žodį į dvigubą žodį: registras **dx** užpildomas skaičiaus reg. **ax** ženklo bitu. Komandos **CBW** ir **CWD** operandų neturi. Pavyzdžiui, apskaičiuokime išraiškas $x = a/b$ ir $y = c/d$

```
a      DB      25
b      DB      5
x      DB      ?
c      DW      -25550
d      DW      -25
y      DW      ?
.
.
MOV    al, a    ; al:=a
CBW                    ; ax:=ah
IDIV   b        ; al:=ax/b
MOV    x, al    ; x:=al
.
.
MOV    ax, e    ; ax:=-c
CWD                    ; dx:as:=ax
IDIV   d        ; ax:=dx:ax/d
MOV    y, ax
```

Aritmetinių komandų nagrinėjimo pabaigoje pateiksime pavyzdį, demonstruojantį daugumą aritmetinių komandų. Apskaičiuoti išraišką $X = A/C + (A * 2 + B * C)/(D - 3)$. visi skaičiai žodžio ilgio su ženklu.

```
a      DW      -255
b      DW      25
```

```

c      DW      -250
d      DW      222
x      DW      ?
.
.
MOV    ax, a      ; ax:=a
CWD                    ; dx:ax:=ax
IDIV   c          ; ax:=a/c
MOV    x, ax      ; x:=a/c
MOV    ax, 2      ; ax:=2
IMUL   a          ; dx:ax:=a*2
MOV    bx, dx
MOV    cx, ax      ; bx:cx:=a*2
MOV    ax, b
IMUL   c          ; dx:ax:=b*c
ADD    ax, cx
ADC    dx, bx      ; dx:ax:=b*c+a*2
MOV    cx, d
SUB    cx, 3      ; cx:=d-3
IDIV   cx          ; ax:=(b*c+a*2)/(d-3)
ADD    x, ax      ; x:=a/c+(b*c+a*2)/(d-3)

```

5.3 Nukreipimo komandos

Komandos vykdomos programoje užrašyta tvarka. Tačiau atliekant loginius sprendimus, reikia pakeisti komandų vykdymo tvarką vienos komandos vykdomos, kai loginė sąlyga patenkinta, kitos - kai nepatenkinta. Loginė sąlyga visuomet yra įrašyta į vėliavėlių registrą. Komandų vykdymo tvarkos keitimui vartojamos nukreipimų komandos. Jos nukreipia valdymą į komandą su žyme. Pavyzdžiui:

```

JMP    t1      ; valdymas nukreipiamas į komandą t1
.
.
t1:    MOV    ah, bh

```

Nukreipimo komandos yra dviejų tipų:

1. besąlyginio nukreipimo komandos,
2. sąlyginio nukreipimo komandos.

Sąlyginio nukreipimo komandos keičia programos vykdymo tvarką, priklausomai nuo ankstesnių komandų rezultatų, kurie atsispindi požymių vėliavėlėse. Nukreipimo komandos keičia komandų rodiklį (registrą **ip**), o kartais

ir komandų segmento registrą **cs**. Šie registrai nurodo kokia komanda bus vykdoma. Jeigu nukreipimo komanda keičia tik registrą **ip**, t. y. valdymas perduodamas to pačio segmento viduje, tai - vidinis nukreipimas, jei keičia abu registrus **cs** ir **ip**, t.y. nukreipiama į žymę, kuri yra kitame programos segmente, tai - išorinis arba tarpsegmentinis nukreipimas.

5.3.1 Nukreipimo adresavimas

Kaip ir kitose komandose, nukreipimo komandose galima vartoti įvairius adresavimo būdus: tiesioginį ir santykinį nukreipimų adresavimą. Esant tiesioginiam adresavimui informacija apie nukreipimo adresą yra pačioje komandoje. Nukreipimo komanda su tiesioginiu adresu užrašoma taip:

JMP	label
------------	--------------

Čia *label* - toliau vykdomos komandos žymė. Nukreipimo komandoje su tiesioginiu adresavimu yra perėjimo žymės poslinkis nukreipimo komandos adreso atžvilgiu. Mašininėje komandoje poslinkiui įrašyti gali būti skiriama 1 baitas (nukreipimams per -128 ± 127 baitų), 2 baitai (-32768 ± 32767 baitų) ir 4 baitai (nukreipimams virš 64kB, t.y. į kitą segmentą).

Užrašant nukreipimo komandą galima nurodyti poslinkio ilgį, vartojant atitinkamus prefiksus:

short - 1 baitas

short - 1 baitas

near ptr - 2 baitai

far ptr - 4 baitai

Pavyzdžiui:

JMP	a
JMP	short b
JMP	near ptr z1
JMP	far ptr z2.

5.3.2 Sąlyginio nukreipimo komandos

Visose sąlyginio nukreipimo komandose poslinkis yra 1 baito ilgio. Jei sąlyginio nukreipimo komanda nurodo į žymę, esančią toliau nei 128 baitai, reikia vartoti specialią konstrukciją. Pavyzdžiui, jeigu reikia pereiti į žymę **ZERO**, kuri yra toliau nuo šios komandos nei 128 baitai, kai nustatyta nulio vėliavėlė **ZF**, Tai galima užrašyti taip:

```

.
.
JNZ    continue
JMP    zero
continue:
.
.

```

čia vartojama priešingos sąlygos nukreipimo komanda. Nukreipimo komandos tikrina vėliavėlių registro bitų reikšmes ir priklausomai nuo sąlygos, atliekamas arba neatliekamas perėjimas. Nukreipimo komandos vėliavėlių reikšmių nekeičia. Vėliavės nustato aritmetinės, loginės komandos ir palyginimo komanda **CMP**. Nukreipimo komanda paprastai rašoma po palyginimo, aritmetinių arba loginių komandų.

Visos sąlyginio nukreipimo komandos pateiktos lentelėje 5.1

Komandos mnemonika	Nukreipimo sąlyga	Paiškinimai
JE/JZ	ZF=1	lygu: rezultatas nulinis
JP/JPE	PF=1	lyginis
JO	OF=1	esant perpildymui
JG	SF=1	esant ženklui
JNE/JNZ	ZF=0	nelygu; rezultatas nenulinis
JNP/JPO	PF=0	nelyginis
JNO	OF=0	nesant perpildymo
JNS	SF=0	nesant ženklo
JL/JNGE	SF=OF	jei mažiau
JLE/JNG	(ZF=1) OR (SF=OF)	jei mažiau arba lygu
JNL/JGE	SF=OF	jei daugiau arba lygu
JNLE/JG	(ZF=0) OR (SF=OF)	jei daugiau
JB/JNEA/JC	CF=1	jei mažiau (be ženklo) arba buvo pernaša
JBE/JNA	(CF=1) OR (ZF=1)	jei mažiau arba lygu (be ženklo)
JNB/JAE/JNC	CF=0	jei daugiau arba lygu (be ženklo) arba nebuvo pernašos
JNBE/JA	(CF=0) AND (ZF=0)	jei daugiau (be ženklo)

5.1 lentelė: Sąlyginio nukreipimo komandos.

Kai kurios komandos turi kelias skirtingas mnemonikas, pvz. **JE** ir **JZ**, tačiau abi mnemonikos lygiareikšmės ir galima vartoti bet kurią iš jų.

Pavyzdys. Duota a , b , c , x . Apskaičiuoti:

$$y = \begin{cases} x + a & , \text{ kai } b + c > 5; \\ x - a & , \text{ kai } b + c = 5; \\ x + 3 & , \text{ kai } b + c < 5. \end{cases}$$

```

a      DW      20
b      DW     -11
c      DW     222
x      DW     255
y      DW      ?
      .
      .
      MOV     dx, b
      ADD     dx, c
      CMP     dx, 5
      JG      a1
      JE      a2
      MOV     ax, x
      ADD     ax, 3
      JMP     p
a1:    MOV     ax, x
      ADD     ax, x
      JMP     P
a2:    MOV     ax, x
      SUB     ax, x
P:     MOV     y, ax
      .
      .

```

5.4 Ciklo programavimo komandos

Cikle, programavimo komandos priklauso sąlyginio nukreipimo komandų grupei. Ciklo komandos vartoja reg. **cx** kaip ciklo iteracijų skaitliuką. Komanda **LOOP** atima 1 iš registro **cx**, ir jei gautas rezultatas nelygus 0, nukreipia į komandą su žyme, kuri nurodyta ciklo komandoje. Kai registras **cx** gaunamas 0, vykdoma paskesnė komanda. Pavyzdžiui:

```

      .
      .
      MOV     cx, sk ; ciklo iteracijų skaičius

```

```

c_pr:
    ; ciklo algoritmas
LOOP  c_pr
    .
    .

```

Ciklo algoritmas vykdomas tiek kartų, kiek buvo nurodyta registre **cx**. Bei jeigu programa cikle keičia registrą **cx**, tai iteracijų skaičius gali neati-tikti pradžioje nurodytam skaičiui. Todėl jeigu reikia cikle keisti **cx**, tai reikia išsaugoti **cx** reikšmę, o po to ją atstatyti. Pavyzdžiui:

```

    .
    .
    MOV    cx, 14
c_pr: PUSH  cx
    .
    .      ; ciklo algoritmas
    .
    POP    cx
    LOOP   c_pr
    .
    .

```

Jeigu pasirodys, kad pradinė **cx** reikšmė lygi 0, tada ciklas bus vykdomas 65536 kartus, nes $0000H - 1 = 0FFFFH$ ir kadangi dabar **cx** nelygu 0, tai ciklas bus kartojamas. Kad išvengti tokios situacijos vartojama komanda **JCXZ** - nukreipianti į programos vykdymą jeigu **cx = 0**. Šią komandą reikia vartoti tada, kai iteracijų skaičius apskaičiuojamas programoje ir jo reikšmė iš anksto nežinoma. Pavyzdžiui:

```

    .
    .
    MOV    cx, skaitliukas
    JCX    c_pab
c_pr:
    .
    .      ; ciklo algoritmas
    .
    LOOP   c_pr
c_pab:
    .
    .

```

Komanda **LOOPE (LOOPZ)** atima 1 iš reg. **cx** ir jei gautas rezulta-tas nelygus nuliui ir **ZF:=1** (nulio vėliavėlė suformuota komanda **CMP** ar

kita komanda), nukreipia į komandą nurodytą žymę. Komanda **LOOPNE** (**LOOPNZ**) atima 1 iš registro **cx** ir, jei gautas rezultatas nelygus nuliui ir **ZF:=0** nukreipia į komandą nurodytą žymę.

Visose ciklo komandose kaip ir sąlyginio nukreipimo komandose poslinkiui yra skiriamas 1 baitas. Pavyzdys gali būti toks: reikia apskaičiuoti masyvo $A(n)$ elementų sumą. Programos kodas toks:

```

.
.
A      DB  -10, -8, 2, 4, 5, 7, -11
N      DW  7
S      DB  ?
      MOV  ah, 0 ; si := 0
      MOV  si, 0
      MOV  cx, N
      JCXZ  pab
c_pr:  ADD  ah, a(si)
      INC  si
      LOOP c_pr
pab:   MOV  s, ah
.
.
```

Komandos **LOOPNE** vartojimo pavyzdys. Duota masyvai $A(10)$ ir $B(10)$. Patikrinti ar yra masyvų elementų pora, kurios suma lygi 100, t.y. $a + b = 100$. Jei yra nustatyti $P = 1$, priešingu atveju - $P = 0$.

```

.
.
.
      MOV  cx, 10
      MOV  si, offset A
      MOV  di, offset B
pr:    MOV  al, (si)
      ADD  al, (di)
      INC  si
      INC  di
      CMP  al, 100
      LOOPNE pr
      JE   rasta
      MOV  P, 0
      JMP  pab
rasta: MOV  p, 1
pab:
.
```

.

Kartotinių ciklų programavimas

Kadangi ciklo komandos vartoja registrą **cx** kaip ciklo iteracijų skaitliuką tai programuojant kartotinius ciklus, išorinio ciklo skaitliuką (registrą **cx**) reikia išsaugoti, pavyzdžiui steke: Pavyzdys:

```
.
.
    MOV    cx, skaitliukas_1
ck_1: PUSH  cx
.
.
.
    MOV    cx, skaitliukas_2
ck_2: PUSH  cx
.
.
.
    MOV    cx, skaitliukas_3
ck_3:
.
.
.
    LOOP   ck_3
.
.
.
    POP    cx
    LOOP   ck_2
.
.
.
    POP    cx
    LOOP   ck_1
.
.
```

5.4.1 Loginės ir postūmio komandos

Paprastai žodžiuose arba baituose yra skaičiai arba simbolinės informacijos kodai. Be to, bituose gali būti saugojami įvairūs kodai ir požymiai (vienne

žodyje gali būti laikoma 16 skirtingų požymių). Loginei informacijai apdoroti vartojamas loginės operacijos IR, ARBA, SUMA MODULIU 2, inversija NE. Šios operacijos atliekamos atskirai su operandų bitais, todėl niekada nebūna perkėlimo į vyresnį bitą.

Pateikiame loginių operacijų rezultatus:

IR

A	B	$A \vee B$
0	0	0
0	1	0
1	0	0
1	1	1

ARBA

A	B	$A \wedge B$
0	0	0
0	1	1
1	0	1
1	1	1

SUMA MODULIU 2

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

NE

A	$\neg A$
0	1
1	0

Šias logines operacijas atlieka tokios komandos:

Loginė operacija IR:

AND op1 , op2

Loginė operacija ARBA

OR op1 , op2

Loginė operacija SUMA MODULIU 2

```
XOR    op1 , op2
```

Inversija NE:

```
NOT    op
```

visais atvejais komandos rezultatas saugoma pirmajame operande.

Patikrinimo komanda **TEST** logiškai sudaugina operandus neužrašydama rezultato. Įvykdžius šią komandą operandų turiniai nesikeičia, tik nustatomos atitinkamos vėliavėlių reikšmės.

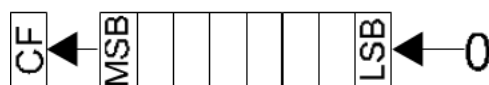
```
TEST    op1 , op2
```

Loginės informacijos apdorojimo komandoms priskiriamos postūmio komandos, Baito arba žodžio bitus galima pastumti vienu ar keliais bitais į dešinę ar kairę. Bitų skaičius, per kurį atliekamas postūmis, nurodomas postūmio skaitliuke. Dažniausiai vartojamas postūmis - vienas bitas. Komandoje taip pat galima nurodyti, bet kurį bitų skaičių, jis užrašomas registre **cl**. Postūmio komandos paskutinį išstumtą bitą patalpina perkėlimo vėliavėlėje **ah**. Mikroprocesoriaus *I8086* komandų sistemoje yra 8 postūmio komandos.

Aritmetinio postūmio į kairę komanda **SAL** visus operando bitus pastumia į kairę. Jauniausieji bitai užpildomi nuliais.

```
SAL    op , 1
        ; arba
SAL    op , CL
```

Jei komandos **SAL** vykdymo metu keičiasi operando ženklas, tai vėliavėlė **OF:=1**. Postūmį iliustruoja 5.4 pav. pavaizduota schema:



5.2 pav.: SAL poslinkio schema

Pavyzdys:

```
SAL    bx , 1
MOV    cl , 4
SAL    x , cl
```

Aritmetinio postūmio į dešinę komanda **SAR** visus operando bitus pastumia į dešinę. Vyriausieji bitai užpildomi ženklo bitu.

```
SAR    op , 1
```



```

; arba
SAR    Op, cl

```

Postūmį iliustruoja 5.3 pav. pavaizduota schema.



5.3 pav.: SAR poslinkio schema

Pavyzdys: pradinė operando reikšmė - 10011100. Atlikus komandą:

```

SAR    Op, 1

```

gausime 11001110 ir **CF=0**.

Loginio postūmio į kairę komanda **SHL** visus operando bitus pastumia į kairę. Jauniausieji bitai užpildomi nuliais. Komanda **SHL** analogiška komandai **SAL**. Loginio postūmio į dešinę komanda **SHR** visus operando bitus pastumia į dešinę. Vyriausieji bitai užpildomi nuliais.

```

SHR    Op, 1
; arba
SHR    Op, cl

```

Postūmį iliustruoja 5.8 pav. pavaizduota schema.



5.4 pav.: SHR poslinkio schema

Aritmetinio ir loginio postūmio komandos vartojamos realizuojant daugybos ir dalybos operacijas. Vartojant postūmio į kairę komandą realizuojant daugybos operacija:

postūmis per 1 bitą atitinka daugybai iš 2,

postūmis per 2 bitus atitinka daugybai iš 4,

postūmis per 3 bitus atitinka daugybai iš 8 ir t.t.

Vartojant postūmio į dešinę komandas realizuojama dalybos operacija. Pavyzdžiui:

```

MOV    cl, 2
SHL    ax, cl ; padauginti skaičių be ženklo iš 4
SAL    ax, cl ; padauginti skaičių su ženklo iš 4

```

```
SHR    ax, cl ; padalinti skaičių be ženklo iš 4
SAR    ax, cl ; padalinti skaičių su ženklo iš 4
```

Postūmio komandomis atliekama daugyba ir skaičiaus dalyba iš skaičiaus 2 laipsnių (2, 4, 8, 16, ...) Bet postūmio komandas galima vartoti ir skaičių daugybai bei dalybai ir iš kitokių skaičių. Pavyzdžiui, padauginame registro **ax** turinį iš 10:

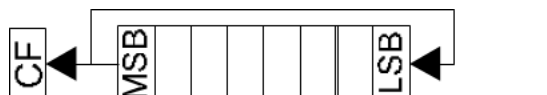
```
MOV    bx, ax
SHL    ax, 1
SHL    ax, 1      ; padauginome iš 4
ADD    ax, bx      ; padauginome iš 5
SHL    ax, 1      ; padauginome iš 10.
```

Ši komandų seka vykdoma apie 11 kartu greičiau, nei atitinkama komanda **MUL**.

Ciklinio postūmio į kairę komandą **ROL** visus operando bitus nustumia į kairę. Išstumti bitai patalpinami į atsilaisvinusių bitų vietą:

```
ROL    Op, 1
        ; arba
ROL    Op, cl
```

Postūmį iliustruoja 5.5 pav. pavaizduota schema.



5.5 pav.: ROL poslinkio schema

Pavyzdys. Pradinė operando reikšmė 10011000. Atlikus komandą

```
ROL    Op, 1
```

gausime 00110001 ir **CF=1**.

Ciklinio postūmio į dešinę komanda **ROR**:

```
ROR    Op, 1
        ; arba
ROR    Op, cl.
```

Postūmį iliustruoja 5.6 pav. pavaizduota schema.



5.6 pav.: ROR poslinkio schema

Pavyzdys: Pradinė operando reikšmė 10011100, atlikus komandą

```
ROR    Op , 1
```

gausime 01001110 ir **CF=0**.

Ciklinio postūmio į kairę kartu su perkėlimo vėliavėle **CF** komanda **RCL**:

```
RCL    Op , 1
        ; arba
RCL    Op , cl
```

Postūmį iliustruoja 5.7 pav. pavaizduota schema.

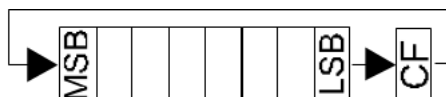


5.7 pav.: RCL poslinkio schema

Pavyzdys. Pradinė operando reikšmė 10011100 ir **CF=1**. Atlikus **RCL** komandą, gausime 00111001 ir **CF=1**. Ciklinio postūmio į dešinę kartu su perkėliau vėliavėle **CF** komanda **RCR**:

```
RCR    Op , 1
        ; arba
RCR    Op , cl
```

Postūmį iliustruoja 5.8 pav. pavaizduota schema.



5.8 pav.: RCR poslinkio schema

Pavyzdys: Pradinė operando reikšmė 10011100 ir **CF:=1**. Atlikus komandą gausime 11001110 ir **CF:=0**.

5.4.2 Procedūrų iškviatimo ir grįžimo iš jų komandos

Iškviatimo komanda **CALL** perduoda valdymą procedūrai, kuri yra bet kurioje programos vietoje. Paskutinė procedūros komanda paprastai yra **RET**, kuri grąžina valdymą komandai užrašytai už iškviatimo komandos **CALL**. Iškviatimo ir grįžimo komandos iš tikrųjų atlieka duomenų išsaugojimo ir atstatymo steke bei nukreipimo komandų veiksmus. Vykdam komandą **CALL DAD**, valdymas perduodamas komandai su žyme *DAD* ir į steką įrašomas adresas komandos, esančios po komandos **CALL DAD**. Vykdam komandą

RET, į registrą **ip** užrašomas komandos adresas iš steko viršūnės, t.y. programos vykdymas tęsiamas nuo komandos esančios už komandos **CALL**, kuri į steką įrašė šį adresą. Gali būti iškviečiamas to paties (vidinis iškvietimas) arba kito (išorinis iškvietimas) programos segmento procedūros. Vykdamas vidinio iškvietimo komandos į steką įrašoma tik reg. **ip** turinys, o vykdamas išorinio iškvietimo komandą - registrus **cs** ir **ip** turiniai. Vidinio ir išorinio iškvietimo komandos užrašomos vienodai. Informacija apie iškvietimo tipą nurodoma procedūros aprašyme. Operandas **NEAR** nurodo, kad procedūra vidinė, o **FAR** - išorinė. Pavyzdys 1. Vidinis iškvietimas:

```
csega SEGMENT
.
.
.
CALL vdad
.
.
vdad PROC NEAR ;    procedūra tame pačiame segmente.
RET
vdad ENDP
csega ENDS
```

Pavyzdys 2. Išorinis iškvietimas:

```
csega SEGMENT
.
.
.
CALL IDAD
.
.
.
csega ENDS
csegb SEGMENT
IDAD PROC FAR
.
.
.
RET
IDAD ENDP
csegb ENDS
.
.
```

Jeigu procedūros aprašyme tipas nenurodytas, tai suprantame, kad procedūra - vidinė.

Pastaba. Steką vartoja komandos **CALL**, **RET**, o taip pat **PUSH** ir **POP**. Todėl pats programuotojas turi rūpintis, kad kiek kartų buvo rašoma į steką, tiek pat kartų būtų ir skaitoma iš jo, t. y. kad vykdant komandą **RET**, steko viršūnėje būtų grįžimo adresas įrašytos atitinkama komanda **CALL**. Programos procedūras galima apjungti į vieną ir transliuoti kartu. Šiuo atveju assemblerio transliatorius turi pilną informaciją apie procedūrų tipą ir pradžios adresus ir jokios papildomos informacijos nurodyti nereikia. Tačiau dažnai būna patogu, ypač didelių programų atveju, procedūras saugoti atskiruose moduluose ir jas transliuoti atskirai. Šiuo atveju transliatoriui reikia nurodyti, kokios procedūros yra, arba bus, vartojamos kituose moduluose. Tai atliekama assemblerio direktyvomis **PUBLIC** ir **EXTRN**

Direktyva **PUBLIC** aprašoma taip:

```
PUBLIC vardas,...
```

Direktyvoje **PUBLIC** nurodyti vardai, žymės arba absoliutūs vardai tampa bendrais, t.y. šiuos vardus galima vartoti visuose programos moduluose. Kiekvienas vardų, nurodytų sąraše, turi būti apibrėžtas šiame faile. Direktyva **EXTRN** aprašoma taip:

```
EXTRN vardas:tipas,...
```

EXTRN nurodomi vardai, kurie yra apibrėžti kitame modulyje direktyva **PUBLIC**. Tipas turi sutapti su vardo tikruoju tipu ir koduojamas tokiais raktiniais žodžiais:

NEAR - vidinis vardas, procedūra tame pačiame segmente;

FAR - išorinis vardas, procedūra kitame segmente;

Programos, sudarytos iš 2-jų procedūrų pavyzdys

Valdanti procedūra išveda pranešimą pr1, po to kreipiasi į vidinę procedūrą "antra", kuri išveda pranešimą pr2, o sugrįžus - vėl išveda pranešimą pr1.

```
stek SEGMENT STACK
      DB 256 DUP (?)
stek ENDS

;---Duomenų segmentas---
duom SEGMENT
pr1   DB 'pranesimas is 1-os procedūros', cr, lf, '$'
pr2   DB 'pranesimas is 2-os procedūros', cr, lf, '$'
duom ENDS
```

```

;---Programos segmentas---
prog1 SEGMENT
ASSUME cs:prog1, ds:duom, ss:stek
cr EQU 13
lf EQU 10
pagr PROC FAR ; į šią procedūrą kreipiamasi iš DOS aplinkos
    PUSH dx
    SUB ax, ax
    PUSH ax
    MOV bx,duom
    MOV ds,bx
; pranešimo pr1 išvedimas
    MOV ah, 09h
    MOV dx, OFFSET pr1
    INT 21h
    CALL antra; kadangi pati proc. šiame segmente
; pranešimo pr1 išvedimas
    MOV ah, 09h
    MOV dx, OFFSET pr1
    INT 21h
    RET
pagr ENDP

; vidinės procedūros tekstas

antra PROC ; kadangi vidinė procedūra
; pranešimo pr2 išvedimas
    MOV ah, 09h
    MOV dx, OFFSET pr2
    INT 21h
    RET
antra ENDP
prog1 ENDS
END pagr

```

Programos, sudarytos iš 2-jų to paties pavadinimo programinių segmentų, pavyzdys

Valdanti procedūra išveda pranešimą pr1, po to kreipiasi į vidinę procedūrą "antra", kuri išveda pranešimą pr2, o sugrįžus - vėl išveda pranešimą pr1. Procedūra "antra" pateikta kitame tokiu pat vardu pavadintame segmente.

```

;---Steko segmentas---

```

```

stek SEGMENT STACK
    DB 256 DUP (?)
stek ENDS

;---Duomenų segmentas---
duom SEGMENT
pr1    DB 'pranesimas is 1-o segmento', cr, lf, '$'
pr2    DB 'pranesimas is 2-o segmento', cr, lf, '$'
duom ENDS

;---Programos 1-asis segmentas---
prog1 SEGMENT
ASSUME cs:prog1, ds:duom, ss:stek
cr      EQU 13
lf      EQU 10
pagr    PROC FAR      ; į šią procedūrą kreipiamasi iš DOS aplinkos
    PUSH    dx
    SUB     ax, ax
    PUSH    ax
    MOV     bx,duom
    MOV     ds,bx
; pranešimo pr1 išvedimas
    MOV     ah, 09h
    MOV     dx, OFFSET pr1
    INT     21h
    CALL    FAR PTR antra; kadangi pati proc. kitame segmente
; pranešimo pr1 išvedimas
    MOV     ah, 09h
    MOV     dx, OFFSET pr1
    INT     21h
    RET
pagr    ENDP
prog1   ENDS

;---Programos tęsinys---
prog1 SEGMENT
antra PROC ; kadangi į ją kreipiamasi iš tokio pat seg.
; pranešimo pr2 išvedimas
    MOV     ah, 09h
    MOV     dx, OFFSET pr2
    INT     21h
    RET
antra   ENDP
prog1   ENDS

```

```
END pagr
```

Programos, sudarytos iš 2-jų skirtingų programinių segmentų, pavyzdys

Valdanti procedūra, esanti viename segmente, išveda pranešimą pr1, po to kreipiasi į procedūrą "antra", kuri išveda pranešimą pr2, o sugrįžus - vėl išveda pranešimą pr1. Procedūra "antra" pateikta kitame programiniame segmente.

```

;---Steko segmentas---
stek SEGMENT STACK
    DB 256 DUP (?)
stek ENDS

;---Duomenų segmentas---
duom SEGMENT
pr1    DB 'pranesimas is 1-o segmento', cr, lf, '$'
pr2    DB 'pranesimas is 2-o segmento', cr, lf, '$'
duom ENDS

;---Programos 1-asis segmentas---
prog1 SEGMENT
ASSUME cs:prog1, ds:duom, ss:stek
cr      EQU 13
lf      EQU 10
pagr    PROC FAR      ; į šią procedūrą kreipiamasi iš DOS aplinkos
    PUSH    dx
    SUB     ax, ax
    PUSH    ax
    MOV     bx,duom
    MOV     ds,bx
; pranešimo pr1 išvedimas
    MOV     ah, 09h
    MOV     dx, OFFSET pr1
    INT     21h
    CALL    FAR PTR antra; kadangi pati proc. kitame segmente
; pranešimo pr1 išvedimas
    MOV     ah, 09h
    MOV     dx, OFFSET pr1
    INT     21h
    RET
pagr    ENDP

```



```

prog1 ENDS

;---Programos 2-asis segmentas---
prog2 SEGMENT
ASSUME cs:prog2
antra PROC FAR; kadangi vidinė procedūra
; pranešimo pr2 išvedimas
    MOV    ah, 09h
    MOV    dx, OFFSET pr2
    INT     21h
    RET
antra ENDP
prog2 ENDS
    END pagr

```

5.4.3 Duomenų perdavimas iškviečiant procedūras

Panagrinėkime duomenų perdavimą procedūrai vartojant steką. Perduodant duomenis iškviečiamai procedūrai, patogiau vartoti tiesioginį kreipimąsi į steką. Vartojant tokį būdą, į steką užrašomi perduodami duomenys, po to iškviečiama procedūra duomenis iš steko išrenkanti tiesioginiu kreipimu. Grįžtant iš procedūros, vartojama speciali komandos **RET** forma, kuri automatiškai išbraukia iš steko perduodamus duomenis:

```
RET    sk ; sk - išbraukiamų iš steko baitų skaičius
```

Pavyzdys. Į procedūrą perduodami 4 simboliai, kurie procedūroje persiunčiami į registrus.

```

SSEG  SEGMENT STACK
DB     256    DUP(?)
SSEG  ENDS

DSEG  SEGMENT
DB     0
DSEG  ENDS

CSEG  SEGMENT
ASSUME cs:CSEG, ds:DSEG, ss:SSEG
START PROC FAR
PUSH  ds
PUSH  A
MOV   bx, DSEG
MOV   ds, bx

```

```

CALL    MAIN
RET
START   ENDP
MAIN    PROC    NEAR
        MOV     ax, 'ab'
        PUSH    ax
        MOV     ax, 'cd'
        PUSH    ax
        Cfill   PAD
MAIN     ENDP
PAD     PROC
        MOV     bp, sp
        MOV     ah, [bp+5] ; tiesioginis
        MOV     al, [bp+4] ; kreipimasis i steka
        MOV     dh, [bp+3]
        MOV     dl, [bp+2]
        RET     4
PAD     ENDP
CSEG    ENDS
END     START

```

5.4.4 Pertraukimų programavimo komandos

Šios grupės komandomis galima programose vartoti servisines funkcijas, kurias realizuojamos kaip programiniai pertraukimai. Komanda **INT** iškviečia procedūrą, kuri apdoroja pertraukiamą su nurodytu numeriu:

```
INT    pertraukimo_numeris
```

Vykdamt šią komandą į steką užrašoma registrų **di**, **cs** ir vėliavėlių registro turiniai. Vėliavėlės **IF** ir **ip** nustatomos lygios 0 (uždraudžiamas žingsnis procesoriaus darbo režimas ir maskuojami pertraukimai). Pertraukimo apdorojimo procedūros adreso rodiklis apskaičiuojamas dauginant pertraukimo numerį iš 4: šiuo rodikliu esantys 2 žodžiai užrašomi į reg. **ip** ir **cs**. Komanda **INT0** generuoja ketvirtą programinį pertraukimą, jei nustatyta vėliavėlė $OF = 1$, priešingu atveju vykdoma sekanti komanda neišsikviečiant pertraukimo apdorojimo procedūros. Ši komanda vartojama po aritmetinių ir loginių komandų galimų perpildymų apdorojimui. Komanda **INT3** generuoja trečią programinį pertraukimą. Asembleris generuoja trumpą vienbaitę mašininę komandą. Komanda **IRET** grąžina valdymą į pertraukimo tašką, atstatant iš steko registrus **IP**, **CS** ir vėliavėlių registro turinius, kurie ten buvo užrašyti kilus pertraukimui. Ši komanda vartojama grįžimui iš paprogramių ir aparatinių pertraukimų apdorojimo procedūrų.

5.5 Procesoriaus valdymo komandos

Procesoriaus valdymo komandas galima suskirstyti į 3 grupes: vėliavėlių nustatymo komandos, išorinės sinchronizacijos komandos ir tuščios operacijos komanda.

Vėliavėlių nustatymo komandos keičia perkėlimo (**CF**), krypties (**DF**) ir pertraukimo leidimo (**IF**) vėliavėles. Šios komandos neturi operandų.

STC - nustatyti **CF:=1**;

CLC - nustatyti **CF:=0**;

CMC - invertuoti **CF**, t.y. $\mathbf{CF} \leftarrow \neg \mathbf{CF}$.

Šios komandos vartojamos norint nustatyti reikiamą **CF** reikšmę prieš atliekant ciklinio postūmio su pernešimu komandas **RCL** ir **RCR**.

STD - nustatyti **DF:=1**;

CLD - nustatyti **DF:=0**.

Šios komandos vartojamos su eilučių apdorojimo komandomis.

STI - nustatyti **IF:=1**;

CLI - nustatyti **IF:=0**.

Išorinės sinchronizacijos komandos vartojamos mikroprocesoriaus veiksmų sinchronizacijai su išoriniais įvykiais.

HLT - sustoti. Mikroprocesorius sustoja, ir stovi tol, kol neįvyksta vienas iš šių įvykių:

1. inicijuojamas procesorius (signalas **RESET**) ;
2. atsiranda nemaskuojamas pertraukimas (signalas **NMI**) ;
3. atsiranda maskuojamas pertraukiamas (signalas **INTR**) ir **IF:=1**.

Komandą **HLT** gali būti vartojama pervedant procesorių į pertraukimo laukimo būseną (pvz. kol bus nuspaustas klaviatūros klavišas). **WAIT** - laukti. Mikroprocesorius pereina į laukimo būseną kol neatsiras signalas **TEST**, Šioje būsenoje mikroprocesorius aptarnauja pertraukimus, bet baigus pertraukimo aptarnavimą vėl grįžta į šią būseną. Komanda **WAIT** vartojama norint sustabdyti mikroprocesoriaus darbą kol išorinis įrenginys nebaigs darbo ir neaktyvuos signalo **TEST**. Duomenų perdavimui kitiems kompiuterio procesoriams (pvz. koprocatoriui 8087) vartojama komanda **ESC**, Prieš kiekvieną komanda galima nurodyti vienbaitį prefiksą **LOCK**, šiuo atveju, kol bus vykdoma komanda joks procesorius negalės naudotis magistrale. Tuščios operacijos komanda **NOP** neatlieka jokių veiksmų. Ji vartojama programų derinimui ir kitiems tikslams.

5.5.1 Įvedimo/išvedimo komandos

Kas tas portas?

Įvedimo/išvedimo komandos vartojamos duomenų perdavimui į išorinius įrenginius ir duomenų priėmimui iš jų. Komandų formatas:

IN	akumulatorius , portas
OUT	portas , akumulatorius

Akumulatorius - tai registras **al**, jei perduodami (priimami) baitai arba **ax** -jei žodžiai. Operando portas reikšmė gali būti nuo 0 iki 255. Operando *portas* vietoje galima nurodytas registras **dx**, kuriame įrašytas porto adresas. Šiuo atveju galima nurodyti iki 65536 adresų. Pavyzdžiai.

IN	al , 200	; įvesti baitą iš porto 200
IN	al , PORT_VA	; arba porto, nurodyto konstanta
OUT	30H, ax	; išvesti žodį į portą 30H
OUT	dx, ax	; arba portą, nurodyta reg. dx

5.5.2 Vėliavėlių persiuntimo komandos

Komanda **PUSHF** įrašo, o **POPF** atstato vėliavėlių registrą iš steko, šios komandos paprastai vartojamos poroje. Pavyzdžiui:

PUSH	ax
PUSH	si
PUSHF	
CALL	SORT
POPF	
POP	si
POP	ax

skyrius 6

Sudėtingesni operandų adresavimo būdai ir programų pavyzdžiai

6.1 Netiesioginis operandų adresavimas

Iprasčiausi operandų adresavimo būdai aprašyti skyrelyje 4.4. Čia panagrinėsime netiesioginio operandų adresavimo būdus. Šiuo atveju operande nurodomas duomenų atmintyje adreso adresas. Priklausomai nuo to, koks registras vartojamas operande, išskiriami tokie adresavimo būdai:

bazuotas operandas,

indeksuotas operandas,

bazinis indeksuotas operandas.

Bazuotas operandas:

<code>poslinkis</code>	<code>[bp]</code>
<code>posiinkis</code>	<code>[bx]</code>

Poslinkis - tai betarpiška reikšmė arba tiesioginis atminties operandas. Duomenų adresas gaunamas sudedant registro turinį su poslinkiu. Jei poslinkis nenurodytas, laikoma, kad jis lygus nuliui. Jei nurodytas registro **bp**, tai adresas apskaičiuojamas steko segmento reg. **ss** atžvilgiu, jei **bx** - **ds** atžvilgiu. Galimi ir kiti bazuoto operando užrašymo būdai:

<code>[poslinkis][bx]</code>
<code>[bx+poslinkis]</code>
<code>[bx]poslinkis</code>
<code>[bx]+poslinkis</code>

Pavyzdys:

```

A      DB      2,4,10,11,12

      MOV      bx, OFFSET A ; bx A lauko adresas
      MOV      ah, [bx] ; ah:=2
      MOV      ah, 3[bx] ; ah:=11
      MOV      bx, 4
      MOV      ah, A[bx] ; ah:=12

```

Indeksuotas operandas:

```

Poslinkis[si]
Poslinkis[di]

```

Šis adresavimo būdas panašus į bazuoto operando adresavimo būdą tik čia vartojami registrai **si** arba **di**, o ne **bx** ar **bp**. Absoliutus adresas apskaičiuojamas duomenų segmento **DS** atžvilgiu. Pavyzdys. Apskaičiuoti masyvo *A* elementų sumą *S*.

```

A      DB      2, 4, 10, 11, 12
S      DB      ?

      MOV      si, 0
      MOV      cx, 5
      MOV      al, 0
C:     ADD      al, A[si]
      INC      si
      LOOP     C
      MOV      s, al

```

Bazinis indeksuotas operandas:

```

poslinkis[bp][si]
poslinkis[bx][si]
poslinkis[bp][di]
poslinkis[bx][di]

```

Duomenų adresas gaunamas, sudedant poslinkį ir bazinio bei indeksinio registrų turinius. Jei nurodytas registre **bx** adresas apskaičiuojamas registro **ds** atžvilgiu, jei **bp** - registro **ss** atžvilgiu. Pavyzdys:

```

ST      DB      'AIJKORZ', 0FFH
.
.
.
      MOV      bx, OFFSET ST

```

```

MOV    si, 0
MOV    al, [bx+si] ; al := 'A'
MOV    bx, 4
MOV    ah, ST [bx][si] ; ah := '0'
.
.
```

Standartinį segmento registrą, kurio atžvilgiu apskaičiuojamas duomenų adresas, galima pakeisti, nurodant komandoje segmento registrą: *seg-reg:adresinė-išraiška*. Pavyzdys:

```

.
.
ST      DB      'ABCDEF'
.
.

MOV     bp, OFFSET ST
MOV     ah, ds:[bp] ; ah := 'A'
```

Komandoje apdorojamų duomenų ilgis (1 baitas ar 2 baitai) nustatomas pagal operandų atributų tipą. Jei nurodomi registrai - pagal registro ilgį, jei kintamieji - pagal tai, kaip jie buvo aprašyti: direktyva **DB** ar **DW**. Jeigu pagal operandus neįmanoma nustatyti apdorojamų duomenų ilgio, būtina nurodyti duomenų ilgį, vartojant prefiksą **PTR**:

BYTE PTR - 1 baitas

WORD PTR - 2 baitai ir tt.

Pavyzdys:

```

B      db      10 dup(?)
.
.

MOV     bx, offset B
MOV     [bx], BYTE PTR 'A'
INC     bx
MOV     [bx], WORD PTR 'BC'
.
.
```

Pirmuose 3 lauko B baituose bus įrašyta 'ABC'

Informacijos, esančios skirtinguose segmentuose, adresavimo pavyzdys

```

cr      EQU      13
lf      EQU      10
;
;-----steko segmentas-----
;
stekas  SEGMENT  STACK
        DB      256 DUP (?)
stekas  ENDS
;
;-----duomenų segmentas-----
;
duom    SEGMENT
        pr4     DB      '      ', cr, lf, '$'
        kond    DB      'K1'
        duom    ENDS

;--kitas duomenų segmentas-----

duom1   SEGMENT
        kon1    DB      ':'
duom1   ENDS

;----papildomas segmentas-----

paps    SEGMENT
        konp    DB      'T2'
paps    ENDS

;-----programos segmentas-----

programa SEGMENT
        ASSUME  cs:programa, ds:duom, es:paps, ss:stekas
proced  PROC  FAR
        PUSH   ds
        SUB    ax, ax
        PUSH   ax
; į duomenų segmento registrą ds įrašome bazinį adresą
        MOV    ax, duom
        MOV    ds, ax
; į papildomo segmento registrą es įrašome bazinį adresą
        MOV    ax, paps
        MOV    es, ax
        JMP    short apeiti_duomenis

```



```

; duomenis talpiname programos segmente
konpr DB      'U3'
s1     DB      '5'
apeiti_duomenis:
; į lauką pr4 perrašome 2-ą simbolį iš duomenų segmento
    LEA    bx, kond
    MOV    al, 1[bx] ; pagal nutylėjimą vartojamas ds registras
    MOV    pr4, al

; į lauką pr4+1 perrašome 1-ą simbolį iš duomenų segmento
    LEA    bp, kond
    MOV    al, ds:[bp] ; nenurodžius ds, vartotų ss registrą
    MOV    pr4+1, al

; į lauką pr4+2 perrašome 1-ą simbolį iš papildomo segmento
    MOV    al, konp ; perrašomas baitas iš papildomo segmento
    MOV    pr4+2, al ; laukas konp yra papildomame segmente

; į lauką pr4+3 perrašome 2-ą simboli iš programos segmento
    LEA    bx, konpr+1 ; poslinkis užrašomas iš programos segmento
    MOV    al, cs:[bx] ; cs: nurodyti būtina, kitaip baitą perrašytu
    ; iš ds segmento, nes vartoja ds registrą
    MOV    pr4+3, al

; į lauką pr4+4 perrašome simboli iš kito
; duomenų segmento
    PUSH    ds
    MOV    dx, duom1 ; įrašome naujo segmento bazinį adresą
    MOV    ds, dx
    ASSUME ds:duom1 ; nes naudojamas kita duomenų segmentą
    MOV    al, kon1
    POP     ds ; atstatome ankstesnį duomenų segmentą
    ASSUME ds:duom ; toliau naudoti šį duomenų segmentą
    MOV    pr4+4, al

    INC    s1 ; padidinamas baito turinys
    MOV    al, s1 ; perrašomas iš programos segmento
    MOV    pr4+5, al

; išvedame lauką pr4
    LEA    dx, pr4
    MOV    ah, 09h
    INT    21h
    RET

```

```
proced endp
programa ends
        end proced
```

Simbolių eilutėje apskaičiuoja kitos simbolių eilutės pasikartojimų skaičių

```
;
;-----steko segmentas-----
;
stekas SEGMENT stack
        DB 256 DUP (?)
stekas ends
;
;-----duomenų segmentas-----
;
duomenys SEGMENT
zodis DB      'KTU'
zod_ilg = $-zodis
eil    DB      'Kauno Technologijos Universitetas (KTU)'
eil_ilg = $-eil
s      DB      ?
duomenys ends
;
;-----programos segmentas-----
;
programa SEGMENT
        ASSUME cs:programa, ds:duomenys, ss:stekas
; paruošia ds
start:
        MOV     ax, duomenys
        MOV     ds, ax
        MOV     s, 0
; užrašome ciklo kartojimo skaičių
        MOV     cx, eil_ilg - zod_ilg
        MOV     di, 0
kit_eil_simb:
        PUSH    cx
; užrašome ieškomo žodžio ilgį
        MOV     cx, zod_ilg
        MOV     bx, 0
sutampa:
        MOV     ah, zodis[bx]
```

```

; lyginame žodžio simbolį su eilutės simboliu
    CMP    ah, eil[bx+di]
    JNE    nesutampa
; jei vienodi, tikrinami kiti simboliai
    INC    bx
LOOP    sutampa
; žodis surastas
    INC    s
nesutampa:
    INC    di
    POP    cx
    LOOP   kit_eil_simb
; išeina iš programos
    MOV    ah, 4ch
    INT    21h
programa ends
end start

```

6.2 Duomenų transformavimo algoritmai

Bazinė įvedimo - išvedimo sistema klaviatūroje surinktus simbolius verčia į ASCII kodus. Jeigu šie simboliai vaizduoja skaičius, tai prieš atliekant aritmetines operacijas, juos iš ASCII kodų reikia pervesti į vidinį skaičių vaizdavimo formatą - dvejetainę skaičiavimo sistemą. Kad gauti apskaičiuotus rezultatus dešimtaine, dvejetainė ar šešioliktaine sistema, dvejetainius skaičius vėl tenka versti į ASCII kodus, tuomet juos galima išvesti į ekraną arba atspausdinti. Parodysime įvairius skaičių transformavimo algoritmus.

Dvejetainio skaičiaus vertimas į dešimtainį skaičių

Prieš išduodant skaičiavimo rezultatus į ekraną arba juos atspausdinant, tenka dvejetainį skaičių versti į dešimtainę, skaičiavimo sistemą ir gautus skaitmenis vaizduoti ASCII kodais, Jeigu dvejetainis skaičius yra registre, tai jo reikšmei -32768(S (38767 užrašyti pakaks 6 baitų. Žinoma, kad kiekvieną skaičių galima užrašyti kaip 10000, 1000, 100, 10 ir i kiekiu suma, t. y. $32767 = 3 * 10^4 + 2 * 10^3 + 7 * 10^2 + 6 * 10^1 + 7 * 10^0$. Šia taisykle galima pasinaudoti verčiant dvejetainį skaičių į dešimtainį. 10000, 1000, 100, 10 ir 1 kiekį galima apskaičiuoti nuosekliai dalinant dvejetainį skaičių iš 10000, o gautas liekanas iš 1000, 100 ir 10. Dalinimo metu gautos sveikosios dalys ir sudarys dešimtainį skaičių. Pademonstruosime šį algoritmą tokia programa:

```

.
.
d      DW      10000, 1000, 100, 10 ; dalikliai

```

```

sk      DW      32767 ; verčiamas skaičius
rez     DB      5 dup(?) ; rezultato laukas

; indeksavimo registru paruošimas
      MOV     si, 0
      MOV     di, 0
; nuoseklia dalyba atliksime 4 kartus
      MOV     cx, 4
; paruošiamo skaičių dalybai
      MOV     ax, sk
dal_ras:
      MOV     bx, d[si]
      MOV     dx, 0
      DIV     bx
; sveikąją dalį verčiame į ASCII kodą
      ADD     al, '0'
; skaitmenį įrašome į lauką
      MOV     rez[di], al
; paruošiamo liekaną dalybai
      MOV     ax, dx
; padidiname indeksavimo registrus
      INC     di
      ADD     si, 2
      LOOP    dal_ras
; vienetus verčiame į ASCII kodą
      ADD     al, '0'
      MOV     rez[di], al
      .
      .

```

Dvejetainį skaičių į dešimtainę sistemą galima versti naudojant nuosekliai duoto skaičiaus ir gautų sveikųjų dalių dalybą iš 10. Šiuo atveju gautos liekanos, užrašytos atvirkščia tvarka, sudarys dešimtainį skaičių (šia taisykle naudojamos versdami dešimtainius skaičius į šešioliktinę, aštuonetinę ir dvejetainę skaičiavimo sistemas). Tokiam algoritmui pateiksime procedūrą dvejetainio skaičiaus, esančio registre **ax**, vertimui į dešimtainį skaičių ASCII kodų eilutę, kurios adresas yra registre **bx**. Ši procedūra įvertina ir skaičiaus ženklą.

```

BIN_ASCII PROC
; naudojamų registrų išsaugojimas
      PUSH    dx
      PUSH    si

```

```
        PUSH    ax
; rezultato eilutę užpildome tarpais
        MOV     cx, 6
t_p:    MOV     byte ptr [bx], ' '
        INC     bx
        LOOP    t_p
; skaičiaus paruošimas dalybai iš 10
        MOV     si, 10
        OR      ax, ax
        JNZ     val
; verčiamas skaičius yra neigiamas
        NEG     ax ; imame papildomą kodą
val:    SUB     dx, dx
        DIV     si
; gauta liekana verčiame į ASCII kodą
        acd     dx, '0'
; įrašome skaitmenį į eilutės pabaigą
        DEC     bx
        MOV     [bx], dl
; skaičiuojame pervestų simbolių kieky
        INC     cx
; ar dar reikia kartoti dalyba ?
        OR      ax, ax
        JNS     val
; gautas rezultatas, užrašysim ženklą
        POP     ax
        OR      ax, ax
        JNS     tg
; buvo neigiamas skaičius, užrašome '-'
        DEC     bx
        MOV     byte ptr [bx], '-'
        INC     cx
; registru atstatytas
tg:     POP     si
        POP     dx
        RET
BIN_ASCII endp.
```

6.2.1 Dešimtainio skaičiaus vertimas į dvejetainę skaičiavimo sistemą.

Tegul dešimtainis skaičius yra įrašytas ASCII kodais eilutėje. Kad gauti dvejetainį skaičių m , reikia paeiliui imti kiekvieną dešimtainio skaičiaus

skaitmenį n ir atlikti tokį veiksmą: $m = m * 10 + n$, tiek kartu, kiek yra dešimtainių skaitmenų. Suprantama, kad prieš atliekant sumavimo operaciją pradinė kintamojo reikšmė prilyginama 0. Tokiam algoritmui pateiksime programos fragmentą, penkiaženklį dešimtainio skaičiaus transformavimui į dvejetainę skaičiavimo sistemą:

```
.
.
eil    db    '255 '
rez    dw    ?
.
.
        MOV    si, 10 ; daugiklio "10" įrašymas
        MOV    ax, 0 ; m:=0
; transformuojamas skaičius gali būti iš 5 skaitmenų
        MOV    cx, 5
        MOV    di, 0
; eilinio skaitmens skaitymas iš eilutės
kit_sk:
        MOV    bl, eil[di]
; tikrinama skaičiaus pabaiga
        CMP    bl, ' '
        JE     pabaiga
; reikšminis skaičius. Verčiame jį
; iš ASCII kodo į dvejetainį
        MUL    si ; m:=m*10
        AND    bx, 0FH
        ADD    ax, bx ; m:=i*10+n
        INC    di
        LOOP   kit_sk
pabaiga:
        MOV    rez, ax.
```

Šešioliktainio skaičiaus vertimas į dvejetainę skaičiavimo sistemą

Analogiškai galime versti ir šešioliktainį skaičių. Šešioliktainį skaičių $5ABFh$ galima užrašyti taip: $5ABFh = 5 \cdot 16^3 + 10 \cdot 16^2 + 11 \cdot 16^1 + 15 \cdot 16^0 = 23231$. Pateiksime programos fragmentą šiam skaičiui išreikšti į dvejetainę sistemą.

```
.
.
eil    DB    '5ABF ' ; duotas skaičius
rez    DW    ?      ; rezultatas
.
```

```
.  
    MOV    ax, 0    ; registre ax kaupsiame rezultatą  
    MOV    cx, 16   ; daugiklis "16"  
    MOV    si, 0  
; eilinio skaitmens skaitymas iš eilutės  
sek_sk:  
    MOV    bl, eil[si]  
; tikriname ar skaitmuo tarp 0 ir 9  
    CMP    bl, '0'  
    JB     ne_sk  
    CMP    bl, '9'  
    JBE    skaic  
; tikriname ar skaitmuo tarp A ir F  
ne_sk: CMP    bl, 'a'  
    JB     ne RAID  
    CMP    bl, 'f'  
    JAE    ne RAID  
; skaitmuo tarp a ir f, apskaičiuojame dešimtainį sk.  
    SUB    bl, 'a'-10  
    JMP    sud  
skaic: SUB    bl, '0'  
sud:  MUL    cx      ; m:=m*16  
    ADD    bx, 0FH  
    ADD    ax, bx    ; m:=m*16+n  
    INC    si  
    JMP    sek_sk  
ne RAID:  
    MOV    rez, ax
```

skyrius 7

Informacijos įvedimo - išvedimo programavimas

7.1 Klaviatūros valdymas

Klaviatūra - tai pagrindinis informacijos įvedimo - išvedimo į kompiuterį įrenginys. Jos pagalba vartotojas gali surinkti komandas, įvesti duomenis, atsakyti į programų užklausas. Asmeninių kompiuterių klaviatūra dažniausiai turi 83 arba 101 klavišą. Kai kurių modelių klaviatūros gali šiek tiek skirtis nuo standartinių, tačiau daugelyje klaviatūrų visus klavišus galima suskirstyti į šias grupes:

- Simbolių klavišai: tai raidės, skaitmenys, specialūs ženklai pvz. , . ; + - = ir pan., taip pat Esc, Tab, Backspace, Enter;
- Valdymo klavišai: tai rodyklės, Ins, Del, End, Home, PgUp, PgDn;
- Funkciniai klavišai: F1- F10;
- Pagalbiniai klavišai: Shift, Ctrl, Alt, Caps Lock, Num Lock, Scroll, Lock.

Paspaudus klavišą arba keletą klavišų iš karto, į kompiuterį siunčiamas specialus kodas, pagal kurį nustatomas paspaustasis klavišas. Šis kodas vadinamas skankodu (*angl.* Scan-Code), tai klavišo eilės numeris. Skankodą nagrinėja BIOS programos ir priklausomai nuo kitų klavišų (pvz, Shift) padėties suformuoja reikiamą reikšmę. Priklausomai nuo klavišo tipo, reikšmė gali būti formuojama keletu būdų:

- Negeruojama jokia reikšmė. Klavišai Shift, Ctrl, Alt, Num Lock, Scroll Lock gali pakeisti kitų klavišų reikšmes, bet jie vieni jokios reikšmės negeneruoja.

- Generuojamas simbolis ASCII kode t.y. vieno baido kodas. Simbolis generuojamas, nuspaudus simbolio klavišą.
- Generuojamas išplėstinis t.y. dviejų baidų kodas (extended ASCII code). Tokio kodo pirmojo baido reikšmė 0, o antrojo - kodas. Išplėstinį kodą generuoja funkciniai bei valdymo klavišai ir dauguma klavišų kombinacijų su Ctrl ir Alt.
- atliekamas specialus veiksmas. Tai:
 1. Ctrl/Alt/Del - perkraunama operacinė sistema ;
 2. Ctrl/C(Ctrl/Break) - pertraukiama programa ;
 3. Ctrl/S(Ctrl/Num Lock) - pristabdoma programa ;
 4. Shift/Print Screen - spausdinama ekrano kopija.

Klavišų skankodai (83 klavišų klaviatūra) pateikti lentelėje:

Išplėstinis kodas (šešiolyktainis)	Klavišai
3	Ctrl/@
F	Enter
10-19	Alt/Q, W, E, R, T, Y, U, I, O, P
1E-26	Alt/A, S, D, F, G, H, J, K, L
2C-32	Alt/Z, X, C, V, B, N, M
3B-44	F1-F10
47	Home
48	↑
49	PgUp
4B	←
4C	→
4F	End
50	↓
51	PgDn
52	Ins
53	Del
54-5D	Shift/F1-F10
5E-67	Ctrl/F1-F10
68-71	Alt/F1-F10
72	Ctrl/Print Screen
73	Ctrl/←
74	Ctrl/→
75	Ctrl/End
76	Ctrl/PgDn
77	Ctrl/Home
78-83	Alt/1,2,3,4,5,6,7,8,9,0,-,=
784	Ctrl/PgUp

7.1.1 Klaviatūros valdymas MS-DOS priemonėmis

MS-DOS priemonės klaviatūros valdymui turi pakankamai daug galimybių, todėl daugeliu atveju jų visiškai pakanka. MS-DOS priemonėmis galima:

- įvesti simbolį;
- tikrinti ar nuspaustas klavišas;
- atvaizduoti arba neatvaizduoti įvestą simbolį ekrane;
- įvesti simbolių eilutę;
- ignoruoti arba neignoruoti Ctrl/C.

MS-DOS funkcijos darbui su klaviatūra iškviečiamos, kreipiantis į 33(21h) pertraukimą. Prieš tai funkcijos kodas įrašomas į registrą **ah**. Norint įvesti išplėstinį kodą, sugeneruotą klavišų kombinacijos, reikia įvesti du simbolius, pirmasis tuomet visada lygus 0.

Simbolio įvedimas laukiant ir atvaizduojant jį ekrane

```
Pradiniai duomenys:  
ah=1  
Rezultatai:  
al=įvestas simbolis
```

Laukia kol bus paspaustas koks nors klavišas. Paspaudus klavišą įvedamas vienas simbolis ir įrašomas ASCII kode į registrą **al**. Įvestas simbolis atvaizduojamas ekrane. Jei nuspausta Ctrl/C, programa pertraukiama.

Simbolio įvedimas be laukimo ir atvaizdavimo ekrane

```
Pradiniai duomenys:  
ah=6  
DL-FFh  
Rezultatai:  
al=simbolis, jei ZF=/0
```

Bandoma įvesti eilinį simbolį. Jei buvo nuspaustas klavišas, tai nulio požymiui **ZF** priskiriama reikšmė 1, o simbolis įrašomas į registrą **al**. Jei simbolio nėra, tai **ZF=0**. Į Ctrl/C nuspaudimą nereaguojama ir ekrane simbolis neatvaizduojamas.

Simbolis įvedamas laukiant, be atvaizdavimo ekrane

```
Pradiniai duomenys:  
ah=7  
Rezultatai:  
al=simbolis
```

Laukia, kol bus paspaustas koks nors klavišas. Paspaudus klavišą įvedamas simbolis ir įrašomas į registrą **al**. Įvestas simbolis ekrane neatvaizduojamas ir į Ctrl/C nereaguoja.

Simbolio įvedimas laukiant be atvaizdavimo ekrane

```
Pradiniai duomenys:  
ah=8  
Rezultatai:  
al=simbolis
```

Laukia, kol bus paspaustas koks nors klavišas. Paspaudus klavišą įvedamas simbolis ir įrašomas į registrą **al**. Įvestas simbolis ekrane neatvaizduojamas. Jei nuspaudžiama Ctrl/C, programa pertraukiama.

Simbolių eilutės įvedimas

```
Pradiniai duomenys:  
ds:dx=buferio adresas
```

Prieš kreipiantis į šią funkciją, reikia paruošti buferį. Buferis turi būti $(n+2)$ baitų ilgio, jei maksimalus įvedamos eilutės ilgis yra n . Pirmame buferio baite turi būti įrašytas ilgis t.y. n . Į buferį, pradedant trečiu baitu, bus surašyti visi simboliai iki klavišo Enter nuspaudimo. Eilutės pabaigos simbolis taip pat bus įrašytas buferyje. Antrame buferio baite bus įrašytas įvestos eilutės ilgis. Nuspaudus Ctrl/C, programa bus pertraukta.

Klaviatūros būsenos nustatymas

```
Pradiniai duomenys:  
ah=11  
Rezultatai:  
al=0FFh, jei nėra simbolio įvedimui  
al=0, jei yra bent vienas simbolis.
```

Tikrinama klaviatūros būseną. Jei yra simbolis, paruoštas įvedimui t.y. buvo nuspaustas koks nors klavišas, tai į registrą **al** įrašoma reikšmė 255 (0FFh). Priešingu atveju, į registrą **al** įrašomas 0.

Klaviatūros buferio valymas ir kitos funkcijos iškvietimas

```
Pradiniai duomenys:  
ah=12  
al=funkcijos numeris
```

Rezultatai priklauso nuo funkcijos numerio. Išvalomas klaviatūros buferis, t.y, panaikinami visi iki tol surinkti simboliai. Po to iškviečiama viena iš klaviatūros valdymo funkcijų 1, 6, 7, 8, 10.

Dešimtainio skaičiaus įvedimo ir vertimo į dvejetainę sistemą pavyzdys

Programa įveda teigiamą skaičių, verčia jį į ASCII kodo dvejetainę sistemą, jį išveda ekrane, po to į dešimtainį ASCII sistemos kodą ir vėl jį išveda.

```
;---Steko segmentas---  
stek SEGMENT STACK  
    DB 256 DUP (?)  
stek ENDS  
  
;---Duomenų segmentas---  
duom SEGMENT  
pr1  DB 'įveskite skaičių ', cr, lf, '$'  
buf  DB 50,?,50 DUP (' '), '$' ; vieta įv. eilutei  
sk   DB 8 DUP (' '), '$'  
duom ENDS  
  
;---Programos segmentas---  
prog SEGMENT  
ASSUME cs:prog, ds:duom, ss:stek  
cr    EQU 13  
lf    EQU 10  
pr    PROC FAR  
    PUSH ds  
    SUB ax, ax  
    PUSH ax  
    MOV bx,duom  
    MOV ds,bx  
; pranešimo išvedimas  
    MOV ah, 09h  
    MOV dx, OFFSET pr1  
    INT 21h  
; skaičiaus įvedimas  
    MOV ah, 0ah
```

```

        MOV     dx, OFFSET buf
        INT     21h
        SUB     ax, ax
        SUB     cx, cx
        MOV     c1, buf+1      ; simbolių kiekis eilutėje
        MOV     bp, OFFSET buf+2 ; simbolių eil. pradžia
        MOV     si, 10
cpr:    MUL     si
        MOV     bl, ds:[bp]    ; ASCII simbolio paėmimas
        AND     bx, 0fh       ; ir jaunesnių bitų išskyrimas
        ADD     ax, bx        ; suma su 2-niu skaičiumi
        INC     bp
        LOOP    cpr
; skaičiaus vertimas į dešimtainę sistemą
        MOV     bx, OFFSET sk+7
        CALL    skvert
        MOV     dx, OFFSET sk
        MOV     ah, 9
        INT     21h
        RET
pr      ENDP
; teigiama sk. vertimas į 10-nės sistemos ASCII kodą
skvert PROC
        PUSH    dx
        PUSH    si
        PUSH    si, 10
dal:    SUB     dx, dx
        DIV     si
        ADD     dx, '0'
        MOV     [bx], di
        DEC     bx
        OR      ax, ax
        JNZ     dal
        POP     si
        POP     dx
        RET
skvert ENDP
progr ENDS
        END     pr

```

skyrius 8

Makropriemonės

Makropriemonės skirtos palengvinti programavimą taip, kad rašant programos tekste vieną operatorių - makrokomandą, transliuojant būtų gaunama keletas operatorių. Todėl į makrokomandą galima žiūrėti kaip į pseudokomandą, vykdančią numatytą algoritmą, sudarytą iš vienos ar kelių assemblerio komandų ar direktyvų. Makropriemonės apima makroaprašų (makroalgoritmų), makrokomandų, pakartojimo ir sąlyginės transliacijos direktyvų sąvokas.

8.1 Makroaprašai ir makrokomandos

Makrokomandas tikslinga vartoti tada, kai programoje reikia atlikti kokias nors pasikartojančias operatorių sekas. Tada tokia seka vieną kartį užrašoma programos pradžioje ir vadinama makroaprašu, o tose vietose kur reikia sekos, rašosi kreipiniai į makroaprašą - vadinami makrokomandomis. Makrokomandos vietoje transliavimo metu iš makroaprašo sugeneruojama operatorių seka, vadinama makroplėtiniu. Taigi transliavimo metu makrokomanda pakeičiama makroplėtiniu. Kaip assemblerio komandos turi operandus, tai makrokomandos turi faktinius parametrus, kuriais transliavimo metu pakeičiami formalūs makroaprašo parametrai. Todėl, priklausomai nuo faktinių parametrų, iš to paties makroaprašo galima gauti skirtingus assemblerio komandų operandus, skirtingas komandas ir net komandų sekas. Pavyzdžiui, sudarykime makroaprašą dauginti skaičių X iš 4, pastumiant jį į kaire per 2 bitus:

```
; --makroaprašas--  
mul4   MACRO x  
SAL    x, 1  
SAL    x, 1  
ENDM
```

Norint padauginti, pavyzdžiui, registro AX turinį iš 4, rašoma makrokomanda **mul4 ax** kurios makroplėtinys

```
SAL    ax, 1
SAL    ax, 1
```

dauginant lauko mas[bx] turinį, rašoma makrokomanda **mas[BX]** ir gaunamas makroplėtinys

```
SAL    mas[bx], 1
SAL    mas[bx], 1
```

Makroaprašai sudaromi pagal tam tikrus reikalavimus ir gali būti pateikti bet kur, tačiau jie visuomet turi būti aprašyti anksčiau negu rašoma makrokomanda. Paprastai makroaprašai pateikiami programos pradžioje ir užrašomi taip:

```
vardas MACRO [formulių parametrų sąrašas]
        ; makro algoritmas
ENDM
```

Čia vardas - makroaprašo (makrokomandos) vardas. Jei jis sutampa su assemblerio komandos ar direktyvos vardu, tai vartojama ne assemblerio komanda, o makroaprašas. Formalių parametrų sąraše nurodomi formalūs parametrai, atskirti kableliais, Formalūs parametrai, tai vardai kurie gali būti makroalgoritme žymės, operacijos ar operandų laukuose.

Pavyzdžiui, sudarykime makroaprašą dviejų skaičių sumai apskaičiuoti: $s = d1 + d2$.

```
; ---makroaprašas sumai s=d1+d2 apskaičiuoti
suma MACRO s, d1, d2
MOV    ax, d1
ADD    ax, d2
MOV    s, ax
ENDM
```

Norint pasinaudoti sudarytu makroaprašu, programos teksto atitinkamoje vietoje rašoma makrokomanda taip: *[žymė:] vardas [faktiniu parametru sąrašas]*. Todėl apskaičiuojant išraišką $z = x + y$, galima rašyti makrokomandą **SUMA z, x, y** o assemblerio translaitorius šio operatoriaus vietoje įterps tokias assemblerio komandas:

```
MOV    ax, x
ADD    ax, y
MOV    s, ax
```

Jeigu kitoje vietoje registre **dx** reiktų gauti **bx** ir **cx** registų sumą, tai rašytume makrokomandą **SUMA dx, bx, cx** ir gautume tokį makroplėtinį:

```
MOV    ax, bx
ADD    ax, cx
MOV    dx, ax
```

Jeigu formalūs parametrai makroalgoritme neišsiskiria leistiniais skyrikliais tai formalus parametras nuo teksto ar tekstas nuo formalaus parametro skiriamas prefiksu "&". Šiuo simboliu galima modifikuoti žymes ir operandus. Pavyzdžiui rezervuojant atmintį kintamuosius galima aprašyti taip :

```
kint_apr MACRO NR, ilgis
lent&NR      DB ilgis DUP(?)
ENDM
```

Tada makrokomanda **kint_apr A, 5** rezervuos tokį lauką

```
lentA  db 5 DUP(?)
```

Formuojant skirtingą operandą makroaprašė galima pateikti tokį pavyzdį:

```
m1  MACRO p1
.
.
JMP P1&TA
.
.
ENDM
```

Makrokomanda **M1 TT** suformuos makroplėtinyje tokią komandą:

```
JMP TTTA
```

Kai makroalgoritme reikia perkoduoti simbolį į skaičių, vartojamas prefiksas % Jis vartojamas tik makrokomandos parametruose. Pateiksime tokį pavyzdį:

```
klaid_pran  MACRO txt
cntr=cntr+1
pran  %cntr,txt
ENDM

pran  MACRO NR, eilute
pran&NR      DB eilute
ENDM

.
.
cntr=0
.
```



```

.
klaid_pran    'sintaksine klaida'
pran1 DB      'sintaksine klaida'
.
.
.
klaid_pran    'klaidingas operandas'
pran2         DB      'klaidingas operandas'
.
.
.

```

Kai makroaprašė rašome komentarus ir nenorime, kad jie būtų makraplėtinyje, tai komentarai pradedami simboliais ;;.

Kai makroplėtinyje reikia reikšmių sąrašą perduoti kaip vieną parametrą, tai šį sąrašą reikia apskliausti laužtiniais skliaustais (...) , o sąrašo elementus skirti kableliais pavyzdžiui:

```
M1 (1,2,3,4)
```

Rašant makroalgoritną dažnai tenka kreiptis į kurį nors vidinį makroaprašo operatorių. Tai atliekama nurodant žymę. Tačiau reikia nepamiršti, kad kreipiantis į makroaprašą pakartotinai, o taip dažnai būna, gausime pasikartojančias žymes. Kad išvengti žymių dubliavimosi makroaprašuose vartojamos lokalinės žymės. Lokalinės žymės turi būti išvardintos makroalgoritmo pradžioje taip:

```
LOCAL lokolinių žymių sąrašas
```

Vietoj lokolinių žymių makroaprašo transliavimo metu generuojami unikalūs vardai ??0000 - ??FFFF. Pavyzdžiui sudarykite makroaprašą minimiam dydžiui surasti: $r = \min(a, b)$.

```

        MIN MACRO    r, a, b
        LOCAL pab
        MOV    r, a
        CMP    a, b
        JL     pab
        MOV    r, b
pab:
        ENDM

```

tokiu atveju iš makrokomandų sekos

```

MIN    ax, bx, cx
MIN    maz, ax, bx

```

gausime plėtinius

```

        MIN    ax, bx, cx
        MOV    ax, bx
        CMP    bx, cx
        JL     ??0000
        MOV    ax, cx
??0000:
        MIN    maz, ax, bx
        MOV    maz, ax
        CMP    ax, bx
        JL     ??0000
        MOV    maz, bx
??0001:
        .
        .

```

Makroplėtinių generavimo listingą transliavimo metu valdo assemblerio direktyvos:

.LALL - generuojamas visas makroplėtinių listingas;

.SALL - negeneruojamas makroplėtinių listingas;

.XALL - generuojamas makroplėtinių listingas tik tiems operatoriams, kurie turi mašines komandas. Listine makroplėtinių eilutės papildomai žymimos skaitmenimis, nurodančiais makrokomandos gylį.

8.2 Kartojimo direktyvos

Kartais kai kurias komandų ar operandų grupes reikia kartoti. Šiam tikslui taikomos kartojimo direktyvos. Jas galima vartoti makroaprašuose, o taip pat ir bet kurioje programos vietoje. Kartojimui yra 3 direktyvos:

REPT

IRP

IRPC

Direktyva

Direktyva **REPT** taikoma kai reikia kartoti tą patį operatorių bloką. Ji rašoma taip:

```

REPT    išraiska
        .
        .          ;kartojimo blokas

```

```
ENDM
```

Blokas kartojamas tiek kartų, kiek nurodo išraiška. Pavyzdžiui užrašas:

```
REPT 4
SAL ax, 1
ENDM
```

ekvivalentiškas komandų sekai:

```
SAL ax, 1
SAL ax, 1
SAL ax, 1
SAL ax, 1
```

Direktyva **IRP** kartoja grupę operatorių, keisdama formalų parametą nurodytu iš faktinių parametrų sąrašo. Ši direktyva užrašoma taip:

```
IRP formalus parametras (fakt. parametrų sąrašas)
.
. ; kartojama grupė
.
ENDM
```

Pavyzdžiui užrašas:

```
IRP K, (1,5,7,13,60)
DB K+1
ENDM
```

ekvivalentiškas direktyvų sekai:

```
DB 2
DB 6
DB 8
DB 14
DB 61
```

Kai faktinių parametrų sąrašas tuščias grupė kartojama 1 kartą, o formalus parametras pakeičiamas tuščia reikšme.

Direktyva **IRPC** kartoja grupę operatorių, keisdama formalų parametą nurodytu simboliu iš simbolių eilutės. Ši direktyva užrašoma taip:

```
IRPC formalus parametras, simbolių eilutė
.
. ; kartojama grupė
.
ENDM
```

Pavyzdžiui užrašas

```
IRPC K, 15736
      DB K
ENDM
```

ekvivalentiškas direktyvų sekai:

```
DB    1
DB    5
DB    7
DB    3
DB    6
```

Kartojimo direktyvas patogiau vartoti ir makroaprašuose. Pateikiame makroaprašus išsaugoti bei atstatyti registrams:

```
pushreg MACRO aa
IRP    reg, (aa)
PUSH   reg
ENDM

popreg  MACRO aa
IRP    reg, (aa)
POP    reg
ENDM
```

Makrokomanda:

```
pushreg (ax,bx)
```

generuos tokį makroplėtinį

```
PUSH ax
PUSH bx
```

O makrokomanda:

```
popreg (bx,ax)
```

atveju gausime tokį makroplėtinį

```
POP bx
POP ax
```

8.3 Makroaprašų sudarymas ir vartojimas

Sudaryti makroaprašai gali būti vartojami daugelyje programų ir daugelio programuotojų, todėl juos sudaryti reikia taip, kad galėtų pasinaudoti, bet kurioje programoje t.y., kad makroaprašas atliktų nurodytas funkcijas ir nesukeltų pašalinių klaidų vartotojo programoje. Jums verta žinoti ir stengtis:

1. Kruopščiai dokumentuoti makroaprašus. Komentarai turi būti aiškūs ne tik makroaprašo autoriui, bet ir vartotojui.
2. Rašant komentarus vartoti simbolius ;;, o atskirti operatorių laukams vartoti tabuliavimo klavišą (vietoj tarpo simbolių).
3. Rašyti universalų makroalgoritmą ir vartoti kitas makrokomandas.
4. Visas vidines žymes išvardinti direktyvoje **LOCAL**.
5. Makroaprašo pradžioje išsaugoti, o pabaigoje atstatyti visus (išskyrus rezultata) makroaprašė vartojamus registrus.

Sudarytus makroaprašus galima saugoti dviem būdais: programos faile arba makrobibliotekoje. Makrobiblioteka - tai atskiras tekstinis failas, kuriame saugomi visi makroaprašai. Šiuo atveju makroaprašais gali pasinaudoti bet kuris programuotojas, naudodamas direktyvą makroaprašų failo prijungimui.

<code>INCLUDE</code>	<code>makroaprašų failo vardas</code>
----------------------	---------------------------------------

skyrius 9

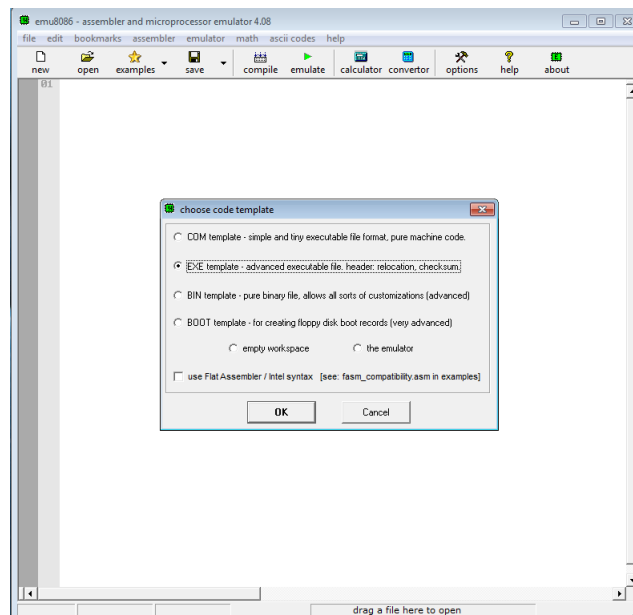
Darbas *I8086* emuliatoriumi

EMU8086 yra *I8086* mikroprocesoriaus emuliatorius. Emuliatorius turi integruotą Asemblerio kalbos kompiliatorių ir derintuvą, leidžiantį pažingsniui vykdyti programą ir stebėti duomenis procesoriaus registruose ir RAM atmintyje.

9.1 Darbo eiga

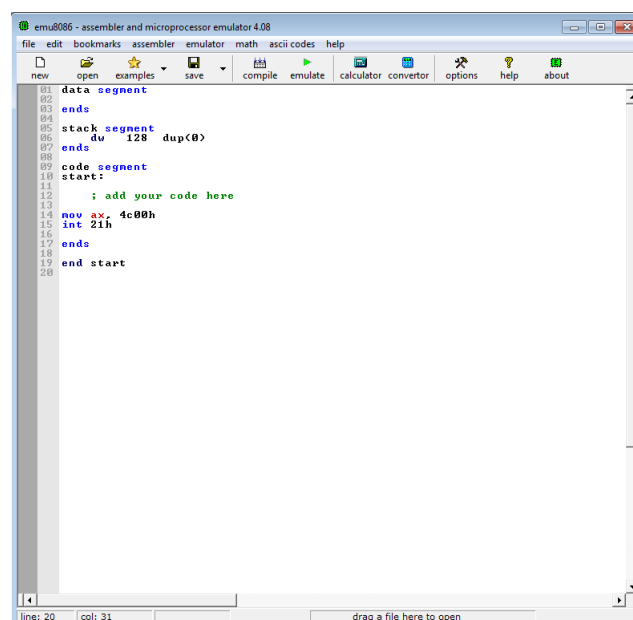
Paleidus programą matomas naujo dokumento kūrimo langas, pavaizduotas **9.1** pav. Kuriant naują programą reikia pasirinkti *.EXE template* (vykdomosios programos) ruošinį.

Prieš pradedant darbą būtina nustatyti darbinį programos katalogą, kur saugomas programos kodas ir sukompiliuota programa. Tai padaroma pasirenkant pagrindinio meniu punktą *assembler* \Rightarrow *set output directory* ir nurodant kelią.



9.1 pav.: Programos langas

Sugeneruojamas programos ruošinys (9.2 pav.), kuriame yra pagrindiniai assemblerio programos elementai.

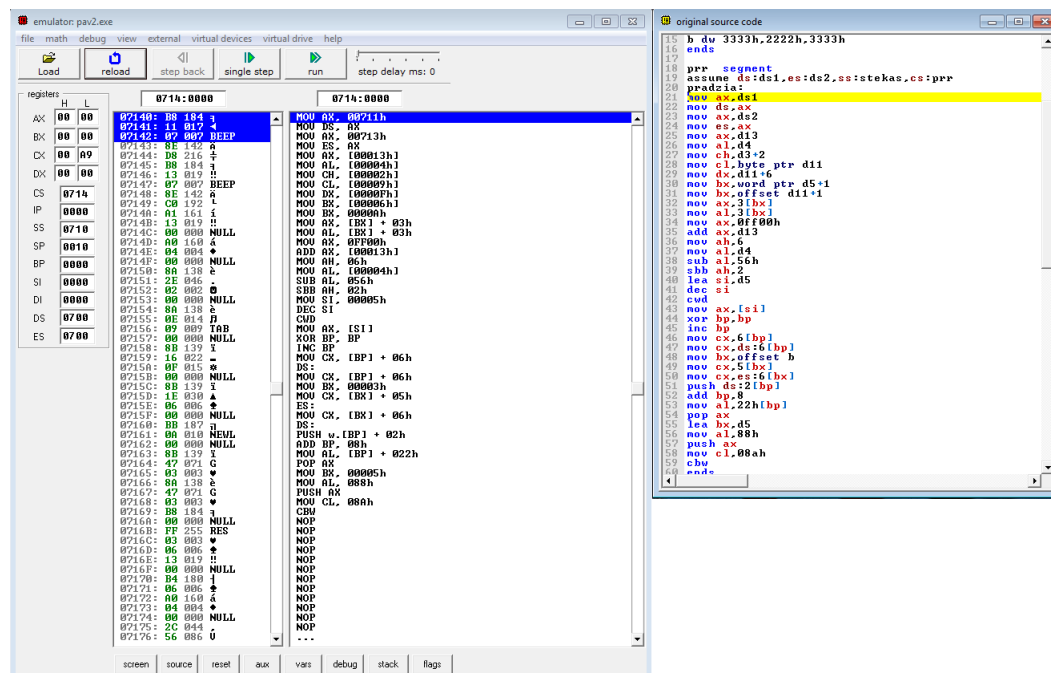


9.2 pav.: Programos ruošinys

Ruošinį papildome kodu, kuris realizuoja programos algoritmą. Programos

pavyzdys pateiktas 3 skyriuje.

Programos kompiliavimui spaudžiame *Emulate* mygtuką. Šis mygtukas atlieka programos kompiliavimą ir paleidžia emuliatorių. Jei kompiliavimas įvyko be klaidų, Atidaromas emulioriaus langas, pavaizduotas 9.3 pav.



9.3 pav.: Emuliatoriaus langas

Programos kodo ir emuliatoriaus languose paryškinamos tuo metu vykdomos instrukcijos. *Registers* skiltyje rodomos procesoriaus registrų reikšmės. Emuliatoriaus lange esantys mygtukai

reload

iš naujo paleidžia emuliuojama programa.

step back

grižta viena instrukcija atgal.

single step

įvykdo vieną instrukciją.

run

paleidžia visą programą nuo nurodytos vietos.

screen

Atidaro programas išvedimo langa.

source

atidaro programos kodo langą.

reset

perkrauna emuliatorių (perkrauna programą, uždaro failus, išvalo ekraną).

aux

atidaro pagalbinius įrankius.

reset

perkrauna emuliatorių (perkrauna programą, uždaro failus, išvalo ekraną).

vars

kintamųjų langas.

debug

išveda registrų reikšmes į tekstinį failą.

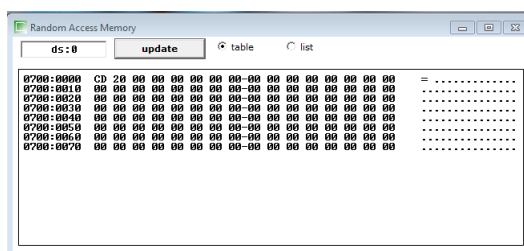
stack

steko langas.

flags

procesoriaus vėlevėlių langas.

Programos derinimui reikalingi šie įrankiai: *atminties (memory)* langas, *steko (stack)* langas bei *single step* mygtukas. Spaudžiant *Single step* mygtuką vykdome programą po vieną instrukciją ir stebime, kaip keičiasi procesoriaus registrų bei RAM atminties reikšmės. Atminties lange (9.4 pav.) rodomas RAM atminties turinys pasirinktu adresu. Norint jame matyti programos duomenis, nurodomas *DS:0* adresas.



9.4 pav.: Atminties langas

9.2 Programų pavyzdžiai

9.2.1 "Labas pasauli!" išvedimo programa

```
duom SEGMENT
    hello_msg DB 'Labas, pasauli!$'
ends

stack SEGMENT
    DW 128 dup(0)
ends

code SEGMENT
start:
    MOV ax, duom
    MOV ds, ax ; įrašomas duomenų segmento adresas
    MOV ah, 9
    MOV dx, OFFSET hello_msg
    INT 21h ; išvedimo pertraukimas
    MOV ah, 4ch
    INT 21h ; programos pabaiga
ends
end start
```

9.2.2 Atminties valdymo programa

```
stekas SEGMENT stack
    DB 16 dup (22h)
ends

duom1 SEGMENT
    MOV ax, duom1
    MOV ds, ax

    d3 DB 44h,22h,33h,11h
    d4 DB 00h
    d5 DB 22h,44h,66h,88h
    d11 DW 9999h,7777h,5555h,3333h,1111h
    d13 DW 6666h
ends

duom2 SEGMENT
    a DB 11h,33h,55h
    b DW 3333h,2222h,3333h
ends
```

```
prg SEGMENT
pradzia:
    MOV     ax,duom1
    MOV     ds,ax ; įrašomas duomenų segmento adresas
    MOV     ax,duom2
    MOV     es,ax ; įrašomas papildomo duomenų seg. adr.

    MOV     ax, d13
    MOV     al, d4
    MOV     ch, d3+2
    MOV     cl, BYTE PTR d11
    MOV     dx, d11+6
    MOV     bx, WORD PTR d5+1
    MOV     bx, OFFSET d11+1
    MOV     ax, 3[bx]
    MOV     al, 3[bx]
    MOV     ax, 0ff00h
    ADD     ax, d13
    MOV     ah, 6
    MOV     al, d4
    SUB     al, 56h
    SBB     ah, 2
    LEA     si, d5
    DEC     si
    CWD
    MOV     ax, [si]
    XOR     bp, bp
    INC     bp
    MOV     cx, 6[bp]
    MOV     cx, ds:6[bp]
    MOV     bx, OFFSET b
    MOV     cx, 5[bx]
    MOV     cx, es:6[bx]
    PUSH    ds:2[bp]
    ADD     bp, 8
    MOV     al, 22h[bp]
    POP     ax
    LEA     bx, d5
    MOV     al, 88h
    PUSH    ax
    MOV     cl, 08ah
    CBW
ends
END pradzia
```

Priedas A

Užduočių variantai

A.1 Aritmetinės išraiškos

1.

$$y = \begin{cases} \left] \frac{a+2b}{a-x} \right[& , \text{ jei } a-x > 0 \\ a^2 - 3b & , \text{ jei } a-x = 0 \\ |c+x| & , \text{ jei } a-x < 0 \end{cases}$$

2.

$$y = \begin{cases} a + c^2 & , \text{ jei } c = 2x \\ |b - 2x| & , \text{ jei } c < 2x \\ \left] \frac{3c+x}{c-2x} \right[& , \text{ jei } c > 2x \end{cases}$$

3.

$$y = \begin{cases} a^2 + x & , \text{ jei } |x| < c + a \\ 2b - a & , \text{ jei } |x| = c + a \\ \left] \frac{c+5b}{|x|-(c+a)} \right[& , \text{ jei } |x| > c + a \end{cases}$$

4.

$$y = \begin{cases} |x| + 2a & , \text{ jei } x + c < 0 \\ 4b - c^2 & , \text{ jei } x + c = 0 \\ \left] \frac{c+b}{x+c} \right[& , \text{ jei } x + c > 0 \end{cases}$$

5.

$$y = \begin{cases} |x+b| & , \text{ jei } c = a \cdot x \\ a^2 - 3b & , \text{ jei } c < a \cdot x \\ \left] \frac{2c-a}{c+ax} \right[& , \text{ jei } c > a \cdot x \end{cases}$$

6.

$$y = \begin{cases} x^2 + 3c & , \text{ jei } b - x = 0 \\ 4a - |c| & , \text{ jei } b - x < 0 \\ \left] \frac{c-b}{b-x} \right[& , \text{ jei } b - x > 0 \end{cases}$$

7.

$$y = \begin{cases} \left] \frac{b+c^2}{x-b} \right[& , \text{ jei } x > b \\ 7a - x & , \text{ jei } x = b \\ |c| + 2a & , \text{ jei } x < b \end{cases}$$

8.

$$y = \begin{cases} 3b + x^2 & , \text{ jei } 3b > x \\ \left] \frac{|x|-4a}{c^3-b} \right[& , \text{ jei } 3b = x \end{cases}$$

9.

$$y = \begin{cases} b^2 + a & , \text{ jei } b + a > x \\ |x| - 2b & , \text{ jei } b + a = x \\ \left] \frac{c-3a}{(b+a)-x} \right[& , \text{ jei } b + a < x \end{cases}$$

10.

$$y = \begin{cases} 2c + |x| & , \text{ jei } c + x = 0 \\ 2x - c^2 & , \text{ jei } c + x > 0 \\ \left] \frac{b+a}{c+x} \right[& , \text{ jei } c + x < 0 \end{cases}$$

A.2 Duomenų formatai

Užd. nr.	A	B	C	X	Y
1	b	b	w	w	w
2	b	w	b	w	w
3	b	w	w	b	w
4	b	w	w	w	b
5	w	b	b	w	w
6	w	b	w	b	w
7	w	b	w	w	b
8	w	w	b	b	w
9	w	w	b	w	b
10	w	w	w	b	b
11	b	b	b	w	w
12	b	w	b	b	w
13	b	w	w	b	b
14	w	b	b	b	w
15	w	w	b	b	b
16	b	b	w	b	w

A.3 Pavyzdys

```
; Funkcija 7, skaičiai su ženklų  
; Duomenys a - w, b - b, c - w, x - w, y - b  
  
stekas SEGMENT STACK  
        DB 256 DUP(0)  
stekas ENDS  
  
duom SEGMENT  
        a      DW 2  
        b      DB -2  
        c      DW 8  
        x      DW -1, -2, -4  
        kiek   = $-x  
        c_k    DW kiek/2  
        y      DB kiek/2 dup(0AAh)  
        isvb   DB 'y=', 6 dup (?), 0Dh, 0Ah, '$'  
        perp   DB 'Perpildymas', 0Dh, 0Ah, '$'  
        daln   DB 'Dalyba is nulio', 0Dh, 0Ah, '$'  
        netb   DB 'Netelpa i baita', 0Dh, 0Ah, '$'  
duom ENDS  
  
prog SEGMENT  
        assume ss:stekas, ds:duom, cs:prog  
pr:     MOV ax, duom  
        MOV ds, ax  
        XOR si, si  
        XOR di, di  
        CMP c_k, 0  
        JA c_pr  
        JMP pab  
c_pr:   MOV cx, c_k  
  
cikl:   MOV al, b  
        CBW  
        CMP x[si], ax  
        JE f2  
        JL f3  
f1:     MOV ax, c  
        IMUL c      ; ax=c^2
```



```

        CMP dx, 0
        JNE kl1
        XCHG ax, dx
        MOV al, b
        CBW
        ADD dx, ax ;  $c^2 + b$ 
        JO kl1
        MOV bx, x[si]
        SUB bx, ax ;  $x - b$ 
        JO kl1
        CMP bx, 0
        JE kl2 ; dalyba iš 0
        MOV ax, dx
        XOR dx, dx
        IDIV bx ;  $ax = rez$ 
        JMP re
f2:     MOV ax, 7
        IMUL a
        CMP dx, 0
        JNE kl1 ; perpilda
        SUB ax, x[si] ;  $7a - x$ 
        JO kl1
        JMP re
f3:     MOV ax, 2
        IMUL a
        CMP dx, 0
        JNE kl1 ; perpilda
        MOV bx, c
        CMP bx, 0
        JG mod ; jei  $c < 0$  padaro priešingą
        NEG bx
mod:    ADD ax, bx ;  $2a + |c|$ 
        JO kl1
        JMP re
re:     CMP ah, 0
        JE ger
        JMP kl3
ger:    MOV y[di], al
        INC si
        INC si
        INC di
        DEC cx

; rezultaty išvedimas į ekraną

```

```

;=====
        CBW          ; išvedamas skaičius yra ax reg.
        PUSH ax
        MOV bx, offset isvb+2
        PUSH bx
        CALL binasc
        MOV dx, offset isvb
        MOV ah, 9h
        INT 21h
;=====
        CMP cx, 0
        JZ pab
        JMP cikl
pab:    MOV ah, 4Ch
        INT 21h
kl1:    LEA dx, perp
        MOV ah, 9
        INT 21h
        XOR al, al
        JMP ger
kl2:    LEA dx, daln
        MOV ah, 9
        INT 21h
        XOR al, al
        JMP ger
kl3:    LEA dx, netb
        MOV ah, 9
        INT 21h
        XOR al, al
        JMP ger

; skaičių verčia į dešimtainę sist. ir išsaugo
; ASCII kode. Parametrai perduodami per steką
; Pirmasis parametras ([bp+6])– verčiamas skaičius
; Antrasis parametras ([bp+4])– vieta rezultatui

binasc    PROC NEAR
        PUSH bp
        MOV bp, sp
; naudojamų registų išsaugojimas
        PUSHA
; rezultato eilutę užpildome tarpais
        MOV cx, 6
        MOV bx, [bp+4]

```







```
tarp: MOV byte ptr[bx], ' '
      INC bx
      LOOP tarp
; skaičius paruošiamas dalybai iš 10
      MOV ax, [bp+6]
      MOV si, 10
      OR ax, ax
      JNS val
; verčiamas skaičius yra neigiamas
      NEG ax
val:  XOR dx, dx
      DIV si
; gautą liekaną verčiame į ASCII kodą
      ADD dx, '0' ; galima--> ADD dx, 30h
; įrašome skaitmenį į eilutės pabaigą
      DEC bx
      MOV [bx], dl
; skaičiuojame pervestų simbolių kiekį
      INC cx
; ar dar reikia kartoti dalybą?
      OR ax, ax
      JNZ val
; gautas rezultatas. Užrašome ženklą
;
      POP ax
      MOV ax, [bp+6]
      OR ax, ax
      JNS teig
; buvo neigiamas skaičius, užrašome -
      DEC bx
      MOV byte ptr[bx], '-'
      INC cx
      JMP vepab
; buvo teigiamas skaičius, užrašau +
teig: DEC bx
      MOV byte ptr[bx], '+'
      INC cx
vepab:
      POPA
      POP bp
      RET
binasc      ENDP
prog ENDS
END pr
```

Priedas B



Komandų sąrašas

PERSIUNTIMO			Kodas	Operacija	VĖLEVĖLĖS										
Pavad.	Komentaras	O			D	I	T	S	Z	A	P	C			
MOV	Perkelti (kopijuoti)		MOV Op1, Op2	Op1:=Op2											
XCHG	Sukeisti		XCHG Op1, Op2	Op1:=Op2 , Op2:=Op1											
STC	Nust. Carry		STC	CF:=1									1		
CLC	Išval. Carry		CLC	CF:=0									0		
CMC	NE Carry		CMC	CF:=¬ CF									±		
STD	Nust. Kryptį		STD	DF:=1	1										
CLD	Išval. Kryptį		CLD	DF:=0	0										
STI	Nust. Pertraukimus		STI	IF:=1	1										
CLI	Atš. Pertraukimus		CLI	IF:=0	0										
PUSH	Ištumti į steką		PUSH Op	DEC SP, [SP]:=Op											
PUSHF	Ištumti į FLAGS		PUSHF	O,D,I,T,S,Z,A,P,C											
PUSHA	Ištumti visus reg.		PUSHA	AX,CX,DX,BX,SP,BP,SI,DI											
POP	Ištraukti iš steko		POP Op1	Op1:= [SP], INC SP											
POPF	Ištraukti FLAGS		PUSHF	O,D,I,T,S,Z,A,P,C	±	±	±	±	±	±	±	±	±		
POPA	Ištraukti visus reg.		POPA	DI,SI,BP,SP,BX,DX,CX,AX											
CBW	Baitas ↔ žod.		CBW	AX:=AL (su ženklu)											
CWD	Žod. ↔ dvigubas		CWD	DX:AX:=AX (su ženklu)	±				±	±	±	±	±		
CWDE	Žod. ↔ išpl. dvig.		CWDE	EAX:=AX (su ženklu)											
IN <i>i</i>	Įvedimas		IN Op1, Prievadas	AL,AX,EAX:=Prievadas											
OUT <i>i</i>	Išvedimas		OUT Prievadas, Op1	Prievadas:=AL,AX,EAX											

i-skaityti aprašą. Vėlevėlės: ± = pakeičiamos šia instrukcija ? = nežinomos po šios instrukcijos vykdymo

ARITMETINĖS		Kodas	Operacija	VĖLEVĖLĖS											
Pavad.	Komentaras			O	D	I	T	S	Z	A	P	C			
ADD	sudėtis	ADD Op1, Op2	Op1:=Op1+Op2	±				±	±	±	±	±			
ADC	sudėtis su pernaša	ADC Op1, Op2	Op1:=Op1+Op2+CF	±				±	±	±	±	±			
SUB	atimtis	SUB Op1, Op2	Op1:=Op1-Op2	±				±	±	±	±	±			
SBB	atimtis su pernaša	SBB Op1, Op2	Op1:=Op1-(Op2+CF)	±				±	±	±	±	±			
DIV	dalyba (be ženklų)	DIV Op	Op=baistas: AL:=AX/Op AH:=liek.	?				?	?	?	?	?			
DIV	dalyba (be ženklų)	DIV Op	Op=žodis: AX:=DX:AX/Op DX:=liek.	?				?	?	?	?	?			
IDIV	dalyba (su ženklų)	IDIV Op	Op=baistas: AL:=AX/Op AH:=liek.	?				?	?	?	?	?			
IDIV	dalyba (su ženklų)	IDIV Op	Op=žodis: AX:=DX:AX/Op DX:=liek.	?				?	?	?	?	?			
MUL	daugyba (be ženklų)	MUL Op	Op=baistas: AX:=AL*Op jei AH=0 •	±				?	?	?	?	?			
MUL	daugyba (be ženklų)	MUL Op	Op=žodis: DX:AX:=AX*Op jei DX=0 •	±				?	?	?	?	?			
IMUL <i>i</i>	daugyba (su ženklų)	IMUL Op	Op=baistas: AX:=AL*Op •	±				?	?	?	?	?			
IMUL	daugyba (su ženklų)	IMUL Op	Op=žodis: DX:AX:=AX*Op •	±				?	?	?	?	?			
INC	padidinti	INC Op	Op:=Op + 1 (CF nesikeičia!)	±				±	±	±	±	±			
DEC	sumažinti	DEC Op	Op:=Op - 1 (CF nesikeičia!)	±				±	±	±	±	±			
CMP	palyginti	CMP Op1, Op2	Op1-Op2	±				±	±	±	±	±			
SAL	aritm. p. į kairę	SAL Op, dydis		<i>i</i>				±	±	±	±	±			
SAR	aritm. p. į dešinę	SAR Op, dydis		<i>i</i>				±	±	±	±	±			
RCL	cikl. p. į kairę su C	RCL Op, dydis		<i>i</i>											
RCR	cikl. p. į dešinę su C	RCR Op, dydis		<i>i</i>											
ROL	cikl. p. į kairę be C	SAL Op, dydis		<i>i</i>											
ROR	cikl. p. į dešinę be C	SAR Op, dydis		<i>i</i>											

i-skaityti aprašą. • Tuomet CF:=0, OF:=0 kitu atveju CF:=1, OF:=1

LOGINĖS			VĖLEVĖLĖS									
Pavad.	Komentaras	Kodas	Operacija	O	D	I	T	S	Z	A	P	C
NEG	neigimas (pap.k)	NEG Op	Op:=0-Op, jei Op=0, CF:=0	±				±	±	±	±	±
NOT	bitų inversija	NOT Op	Op:=¬Op (invertuoti bitai)									
AND	loginis IR	AND Op1, Op2	Op1:=Op1∨Op2	0				±	±	?	±	0
OR	loginis ARBA	OR Op1, Op2	Op1:=Op1∧Op2	0				±	±	?	±	0
XOR	suma modliu 2	XOR Op1, Op2	Op1:=Op1⊕Op2	0				±	±	?	±	0
SHL	p. į kairę	SHL Op, dydis		i				±	±	?	±	±
SHR	p. į dešinę	SHR Op, dydis		i				±	±	?	±	±

ĮVAIRIOS			VĖLEVĖLĖS									
Pavad.	Komentaras	Kodas	Operacija	O	D	I	T	S	Z	A	P	C
NOP	nėra operacijos	NOP	Nėra operacijos									
LEA	užkrauti adresą	LEA Op1, Op2	Op1:= Op2 adresas									
INT	petraukimas	INT Nr	pertraukia programą		0	0						

ŠUOLIAI (vėlevėlės nesikeičia)												
Pavad.	Komentaras	Kodas	Operacija	Pavad.	Komentaras	Kodas	Operacija	O	D	I	T	S
CALL	kviesti proc.	CALL Proc		RET	Grižti iš proc.s	RET						
JMP	besąlyginis	JMP tikslas										
JE	jei lygu	JE tikslas	(=JZ)	JNE	jei ne lygu	JNE tikslas	(=JNZ)					
JZ	jei nulis	JZ tikslas	(=JE)	JNZ	jei ne nulis	JNZ tikslas	(=JNE)					
JCXZ	jei CX nulis	JCXZ tikslas										
JP	jei paritetas lyg.	JP tikslas	(=JPE)	JNP	jei paritetas nelyg.	JNP tikslas	(=JPO)					
JPE	jei paritetas lyg.	JPE tikslas	(=JP)	JPO	jei paritetas nelyg.	JPO tikslas	(=JNP)					

ŠUOLIAI be ženklo				ŠUOLIAI su ženklu			
Pavad.	Komentaras	Kodas	Ekviv.	Pavad.	Komentaras	Kodas	Ekviv.
JA	jei daugiau	JA tikslas	(≡JNBE)	JG	jei daugiau	JG tikslas	(≡JNLE)
JAE	jei daugiau ar lygu	JAE tikslas	(≡JNB≡JNC)	JGE	jei daugiau ar lygu	JGE tikslas	(≡JNL)
JB	jei mažiau	JB tikslas	(≡JNAE≡JC)	JL	jei mažiau	JL tikslas	(≡JNGE)
JBE	jei mažiau ar lygu	JBE tikslas	(≡JNA)	JLE	jei mažiau ar lygu	JLE tikslas	(≡JNG)
JNA	jei ne daugiau	JNA tikslas	(≡JBE)	JNG	jei ne daugiau	JNG tikslas	(≡JLE)
JNAE	jei mažiau ar lygu	JNAE tikslas	(≡JB≡JC)	JNGE	jei mažiau ar lygu	JNGE tikslas	(≡JL)
JNB	jei ne mažiau	JNB tikslas	(≡JAE≡JNC)	JNL	jei ne mažiau	JNL tikslas	(≡JGE)
JNBE	jei daugiau	JNBE tikslas	(≡JA)	JNLE	jei daugiau	JNLE tikslas	(≡JG)
JC	jei pernaša	JC tikslas		JO	jei perpilda	JO tikslas	
JNC	jei nėra pernašos	JNC tikslas		JNO	jei nėra perpildos	JNO tikslas	
				JS	jei yra ženklas (-)	JS tikslas	
				JNS	jei nėra ženklo (+)	JNS tikslas	