

mokėti su neigiamais skiciais, skaiciavimo sistemomis dirbti, skaiciu i mbr nusiusti su komandomis

VILNIAUS UNIVERSITETAS  
MATEMATIKOS IR INFORMATIKOS FAKULTETAS  
INFORMATIKOS KATEDRA

Doc. dr. Antanas MITAŠIŪNAS

# KOMPIUTERIŲ ARCHITEKTŪRA

*Mokymo priemonė*

VILNIUS – 2016

# KOMPIUTERIŲ ARCHITEKTŪRA

## **Turinys**

1.	Ižanga.....	3
2.	Pozicinės skaičiavimo sistemos .....	4
2.1.	Pozicinių skaičiavimo sistemų samprata .....	4
2.2.	Pavedimas iš vienos sistemos į kitą.....	5
3.	Mikroprogramavimas.....	7
3.1.	Procesoriaus fizinio lygio komponentės .....	7
3.2.	Mikrokomandos .....	9
3.3.	Interpretuojamas lygis.....	11
3.4.	Interpretuojantis lygis .....	13
3.5.	Komandos ADD realizacija .....	17
3.6.	Ventilių mikroprograminis valdymas .....	19
3.7.	Mikroprogramavimo kalba .....	22
3.8.	Interpretuojamos mašinos interpretatorius.....	25
3.9.	Mikroprograminio lygio projektavimas .....	28
4.	Mikroprocesoriaus Intel 8088 architektūra.....	35
4.1.	Mikroprocesoriaus architektūros samprata .....	35
4.2.	Operatyvios atminties samprata .....	35
4.3.	Portų sistema.....	36
4.4.	Kompiuterio sandara.....	37
4.5.	Mikroprocesoriaus Intel 8088 sandara.....	39
4.6.	Komandų struktūra.....	41
4.7.	Adresavimo būdai .....	45
4.8.	Stekas .....	47
4.9.	Pertraukimų sistema.....	47
4.10.	Duomenų formatai .....	55
4.11.	Požymių registras SF .....	59
5.	Komandų sistema.....	64
5.1.	Bendrosios mikroprocesoriaus komandos .....	65
5.2.	Aritmetinės komandos .....	69
5.3.	Veiksmai su dešimtainio formato skaičiais.....	75
5.4.	Konvertavimas .....	80
5.5.	Loginės komandos .....	81
5.6.	Eilutinių komandos.....	84
5.7.	Valdymo perdavimo komandos .....	88
5.8.	Sąlyginis valdymo perdavimas .....	92
5.9.	Ciklų komandos .....	94
5.10.	Pertraukimų komandos .....	95
5.11.	Procesoriaus būsenos valdymas ir sinchronizavimas .....	97
6.	Koprocesorius Intel 8087 .....	99
6.1.	Koprocesoriaus duomenų formatai .....	100
6.2.	Koprocesoriaus duomenų formatų vaizdavimas.....	103
6.3.	Koprocesoriaus architektūra .....	107
6.4.	Koprocesoriaus Intel 8087 komandų sistema .....	111
6.5.	Duomenų persiunrimo komandos .....	111
6.6.	Koprocesoriaus valdymo komandos .....	116
6.7.	Skaitmeninio apdorojimo komandos .....	118
7.	Literatūra.....	126

## 1. Įžanga

Mokymo priemonė “Kompiuterių architektūra” yra skirta Informatikos ir Programų sistemų studijų programų studentams privalomo dalyko “Kompiuterių architektūra” studijoms.

Mokymo priemonė susideda iš penkių temų: pozicinės skaičiavimo sistemos, mikroprogramavimas, mikroprocesoriaus Intel 8088 architektūra, komandų sistema, koprocesoriaus Intel 8087 architektūra.

Mokymo priemonės tikslas - supažindinti studentus su kompiuterių architektūra, tame tarpe su mikroprocesorių architektūra, įskaitant duomenų formatus, komandų sistemą, pertraukimų sistemą, atminties organizavimą, assemblerio notacijos sistemą. Šioje mokymo priemonėje yra nagrinėjami kompiuterių architektūros klausimai, remiantis Intel tipo mikroprocesoriaus architektūra. Įvadinėje kurso dalyje yra apžvelgiamos pozicinės skaičiavimo sistemos. Yra nagrinėjami CISC architektūroje taikomi mikroprogramavimo pagrindai.

Pagrindinėje kurso dalyje yra apžvelgiamos bendrosios Intel mikroprocesoriaus komponentės ir savybės, tokios kaip: duomenų registrai, duomenų formatai, adresų registrai, vykdomojo adreso formavimo aparatas, adresavimo režimai, pertraukimų mechanizmas, portų sistema, operatyvi atmintis, absoliutaus adreso formavimo aparatas, komandų sistema, o taip pat koprocesoriaus architektūra, duomenų formatai, komandų sistema.

Mokymo priemonė remiasi mikroprocesoriaus Intel 8088 koprocesoriaus Intel 8087 architektūromis. Pasirinkimas yra apspręstas šiais motyvais:

- Žinių įsisavinimo gilumui yra tikslinga neapsiriboti principiniu lygmeniu, o išnagrinėti išbaigtą sistemą
- Siekiant turėti galimybę praktiškai išbandyti architektūros elementus, yra svarbu turėti ne modelinę architektūrą, o realią, paplitusią ir visiems prieinamą
- Siekiant turėti realią, kartu yra svarbu, kad architektūra būtų kiek įmanoma paprasta.

Mokymo priemonėje yra akcentuojamos sąvokos, jų plotmės, bendrieji principai, siekiama suformuoti kompiuterio, kaip algoritmų vykdytojo, viziją. Akcentuojama, kad kompiuteris architektūriniame nagrinėjimo lygmenyje nieko kito nedaro, išskyrus komandų vykdymą viena po kitos.

Nuoširdžiai dėkoju savo dukrai Ievai Mitašiūnaitei, kuri būdama Informatikos studijų programos studente, išklausiusi „Kompiuterių architektūros“ dalyko paskaitas, paskaitų konspekto rankraščiui suteikė šį pavidalą. Taip pat dėkoju savo studentams Žilvinui Ledui, Maksim Sorokin, Martynui Jusiui ir visiems kitiems, pateikusiems pastabas ir pasiūlymus konspekto gerinimui.

## 2. Pozicinės skaičiavimo sistemos

### *Pozicinių skaičiavimo sistemų samprata*

Skaičiavimo sistemoje pagrindas (pvz.: 10, 8, 16, 2) nusako skaitmenų skaičių ir skaitmens pozicijos svorį. Pavyzdžiui, dešimtainis skaičius  $a_n a_{n-1} \dots a_1 a_0$  yra lygus  $a_n \cdot 10^n + a_{n-1} \cdot 10^{n-1} + \dots + a_1 \cdot 10^1 + a_0 \cdot 10^0$ .

situos dalykus reikia  
pagauti labai gerai

Dvejetainė skaičiavimo sistema	Aštuntainė skaičiavimo sistema	Dešimtainė skaičiavimo sistema	Šešioliktinė skaičiavimo sistema
0	0	0	0
01	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F
10000	20	16	10
...	...	...	...

Bitų skaičius	Galimų kombinacijų skaičius
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512

## KOMPIUTERIŲ ARCHITEKTŪRA

10	1024 = 1 K
11	2 K
12	4 K
13	8 K
14	16 K
15	32 K
16	64 K

Segmento dydis yra 64 K; bet kurį adresą jame nurodyti užtenka 16 bitų

...	...
20	1 M
...	...
24	16 M
...	...
32	4 GB

Tegu turime skaičių X skaičiavimo sistemoje su pagrindu P:

$$X = ( \underbrace{p_n p_{n-1} \dots p_1 p_0}_{\text{sveikoji dalis}} \underbrace{p_{-1} p_{-2} \dots p_{-k}}_{\text{trupmeninė dalis}} )_P = p_n \cdot P^n + p_{n-1} \cdot P^{n-1} + \dots + p_0 \cdot P^0 + p_{-1} \cdot P^{-1} + \dots + p_{-k} \cdot P^{-k}$$

Tas pats skaičius X skaičiavimo sistemoje su pagrindu Q:

$$X = ( \underbrace{q_m q_{m-1} \dots q_1 q_0}_{\text{sveikoji dalis}} \underbrace{q_{-1} q_{-2} \dots q_{-l}}_{\text{trupmeninė dalis}} )_Q = q_m \cdot Q^m + q_{m-1} \cdot Q^{m-1} + \dots + q_0 \cdot Q^0 + q_{-1} \cdot Q^{-1} + \dots + q_{-l} \cdot Q^{-l}$$

sitie dar

### ***Pervedimas iš vienos sistemos į kitą***

1. Jeigu skaičiavimus atliekame sistemoje su pagrindu P, tai skaičiaus pervedimas iš sistemos Q į sistemą P reiškia polinominio užrašo sistemoje Q suskaičiavimą sistemoje P.

Pavyzdžiui, perveskime iš šešioliktainės skaičiavimo sistemos i dešimtainę skaičių BA,8:

$$(BA,8)_{16} = 11 \cdot 16^1 + 10 \cdot 16^0 + 8 \cdot 16^{-1} = 176 + 10 + 0,5 = (186,5)_{10}$$

2. Pervedimas iš skaičiavimo sistemos P į Q. Atskirai pervesime sveikąją ir trupmeninę dalis:

a) tegu sveikoji skaičiaus dalis yra N.

$N = q_m \cdot Q^m + q_{m-1} \cdot Q^{m-1} + \dots + q_0 \cdot Q^0$ , reikia nustatyti koeficientus  $q_m, q_{m-1}, \dots, q_0$ .

$$N = q_m \cdot Q^m + q_{m-1} \cdot Q^{m-1} + \dots + q_0 \cdot Q^0 \quad /: Q$$

$N \quad /: Q$  liekana yra  $q_0$

## KOMPIUTERIŲ ARCHITEKTŪRA

$$N_1 = q_m \cdot Q^{m-1} + q_{m-1} \cdot Q^{m-2} + \dots + q_1 \quad /: Q$$

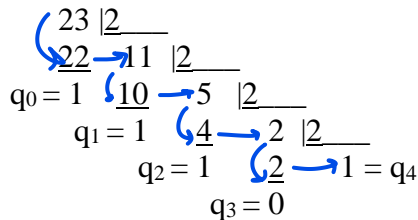
$N_1 /: Q$  liekana yra  $q_1$

...

$$N_m = q_m \quad /: Q$$

$N_m /: Q$  liekana yra  $q_m$

Pavyzdžiui, perveskime iš dešimtainės skaičiavimo sistemos į dvejetainę skaičių  $(23)_{10}$  :



$$q_4 q_3 q_2 q_1 q_0 = (10111)_2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 16 + 0 + 4 + 2 + 1 = (23)_{10}$$

b) tegu trupmeninė skaičiaus dalis yra  $N$ . Iš anksto nežinome, kiek reikšmių bus po kablelio, nes pervedant iš sistemų į sistemas galima gauti begalines trupmenas.

$$N = q_{-1} \cdot Q^{-1} + q_{-2} \cdot Q^{-2} + \dots + q_{-l} \cdot Q^{-l} + \dots, \text{ reikia nustatyti koeficientus}$$

$$q_{-1}, q_{-2}, \dots, q_{-l}, \dots$$

$$N = q_{-1} \cdot Q^{-1} + q_{-2} \cdot Q^{-2} + \dots + q_{-l} \cdot Q^{-l} + \dots \quad /: Q$$

Jei tokios daugybos rezultate gauname skaičių, kurio sveikoji dalis nenulinė, tai sveikąją dalį atmetame (ji yra koeficientas), likusia trupmeninę dalį vėl dauginame iš  $Q$  ir procesą kartojame toliau.

Pavyzdžiui, perveskime iš dešimtainės sistemos į dvejetainę skaičių  $(0,1)_{10}$  :

	0,1	
	<u>2</u>	
$q_{-1} = 0$	0,2	
	<u>2</u>	
$q_{-2} = 0$	0,4	
	<u>2</u>	
$q_{-3} = 0$	0,8	
	<u>2</u>	
$q_{-4} = 1$	1,6	
	<u>2</u>	
$q_{-5} = 1$	1,2	
	<u>2</u>	
$q_{-6} = 0$	0,4	

} periodiškai kartojasi

$$(0,1)_{10} = (0,0(0011))_2$$

### 3. Mikroprogramavimas

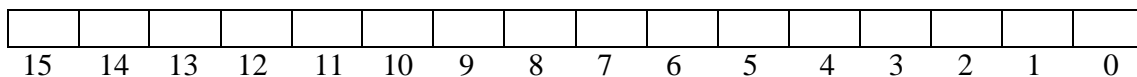
Nėra griežtų ribų tarp kompiuterio aparatūrinės ir programinės įrangos. Be to tos ribos visą laiką kinta. Tradicinės įsivaizduojamos komandos ankstesniuose kompiuteriuose buvo realizuojamos atskiromis schemomis. Dabartiniuose kompiuteriuose visos tradicinės komandos yra interpretuojamos mikroprogramų pagalba. Mikroprogramos savo ruožtu sudarytos iš mikrokomandų, kurios realizuojamos aparatūriškai schemų lygyje. Mūsų tikslas yra susipažinti su mikroprograminiu lygiu ir mikroprogramavimu.

Mikroprograminio lygio tikslas yra užtikrinti aukštesnio lygio virtualios mašinos komandų interpretavimą.

#### ***Procesoriaus fizinio lygio komponentės***

1. Registras – tai įrenginys, skirtas informacijos saugojimui. Registras išdėstyti pačiame procesoriuje, jie yra analogiški atminties žodžiams, tačiau greitesni.

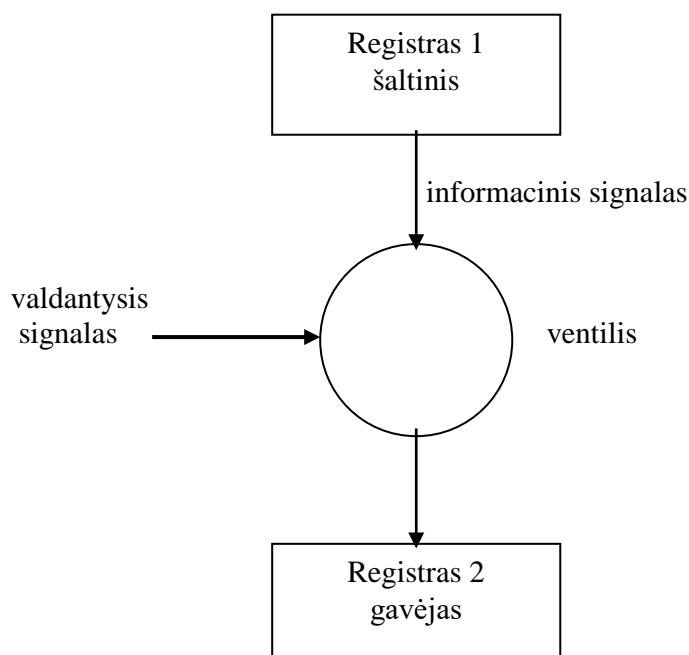
Tegul yra  $N$  registrų : 0, 1, 2, 3, ...,  $N-1$ . Registras charakterizuojami bitų skaičiumi. Tegul yra 16 bitų registras:



Skaitant iš registro, informacija jame nekinta.

2. Magistralė – tai elektros grandinė, jungianti du registrus. Magistralė susideda iš lygiagrečių laidininkų kiekvienam jungiamųjų registrų bitui. 16 bitų magistralės plotis yra 16. Fiziškai yra daugiau laidininkų, tačiau mus domina loginis lygis. Sakykime, kad magistrale informacija perduodama viena kryptimi.

3. Ventiliai tipo  $\wedge$  ( sakoma tipo 'ir' ). Jie skirti informacijos perdavimui iš registrų į magistralės ir iš magistralių į registrus. Ventilio panaudojimo schema:



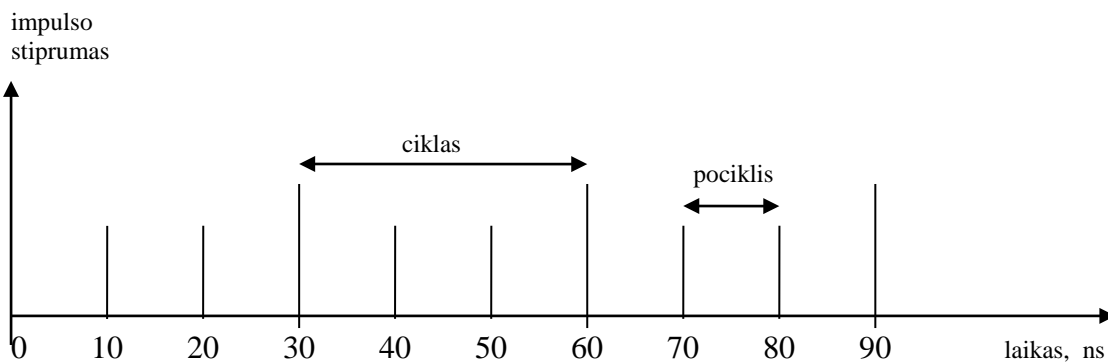
## KOMPIUTERIŲ ARCHITEKTŪRA

Jeigu valdantis signalas yra 1, tai ventilis tipo 'ir' informacinį signalą praleidžia, o jeigu valdantis signalas yra 0, tai informacinis signalas nepraleidžiamas.

and

Valdantis signalas	$\wedge$	Informacinis bitas	Rezultatas
1		1	1
1		0	0
0		1	0
0		0	0

4. Taktinių impulsų generatorius. Taktiniai impulsai naudojami procesų sinchronizavimui. Taktinių impulsų periodas vadinamas ciklu. Ciklo viduje silpnesnių impulsų intervalai vadinami pocikliais. Tokiu būdu laikas sudalinamas diskretingais intervalais.



Pastaba: ns =  $10^{-9}$  s

Pavyzdžiui ventilis intervale 0 – 20 galėtų būti uždarytas, o intervale 20 – 30 atidarytas.

5. Kreipimasis į atmintį. Kreipimuisi į atmintį naudojami du procesoriaus registrai: MAR – atminties adresinis registras ir MBR – buferinis registras. Jeigu ląstelės adresas yra MAR ir duotas skaitymo signalas, tai ląstelės turinys patalpinamas į MBR, o jeigu duotas rašymo signalas, tai iš MBR adresu MAR rašoma į ląstelę. Atminties ciklas tai yra laikas priimti skaitymo signalą, užrašyti ląstelės turinį į MBR ir pasiruošti sekančio signalo priėmimui. Atminties ciklas skiriasi nuo procesoriaus ciklo.

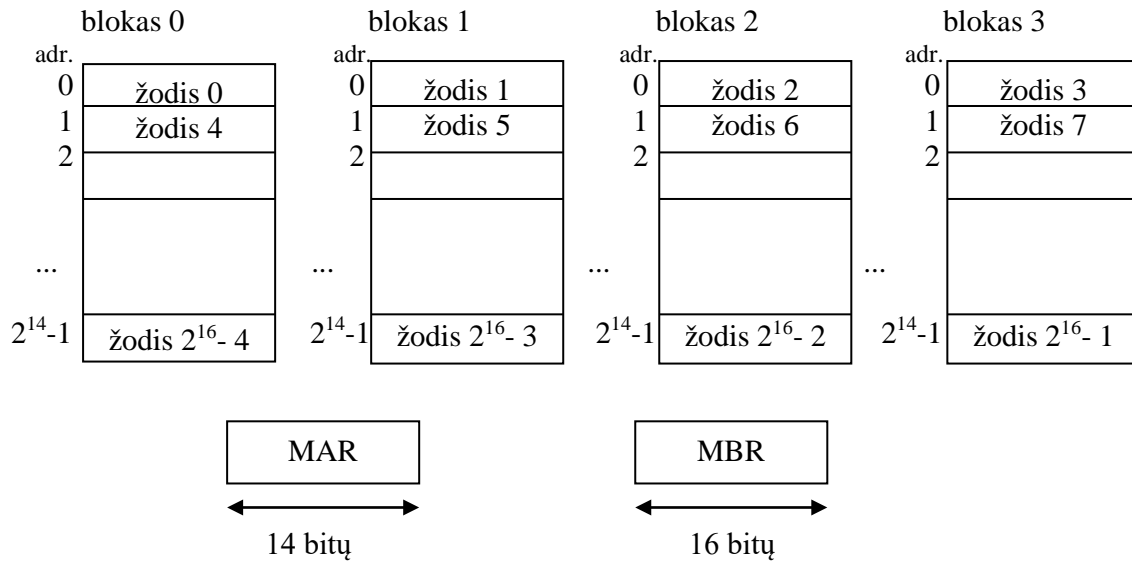
MBR bitų skaičius nustato atminties magistralės plotį. Kuo didesnis magistralės plotis, tuo mažiau reikia atminties ciklų informacijos fiksuotam kiekiui paimti. Tačiau atminties magistralės plotis 'brangiai kainuoja'.

### Pastaba.

Apsikeitimo pagreitinimui galima naudoti lygiagretų apsikeitimą su keliais identiškais atminties blokais. Nagrinėjime įprasto dydžio atminties modulį – 64 K. Tegul yra 4 blokai po  $2^{14}$  žodžių, kurie sumoje duoda  $2^{16}$  žodžių:



## KOMPIUTERIŲ ARCHITEKTŪRA



Adreso tradicinėje komandoje du jauniausi bitai nustato bloką {0, 1, 2, 3}, o likę 14 bitų nurodo adresą bloke. Todėl tokiu atveju registro MAR dydis būtų 14 bitų.

6. Aritmetinis loginis įrenginys dažniausiai turi sudėties komandą, kartais atimties, paprastai logines komandas (and, or, ...). Komandų sistema mikrolygyje sudaroma taip, kad iš vienos pusės būtų galima realizuoti visus matematinius veiksmus, o būtinas minimumas yra sudėtis, nes atimtis tai yra sudėtis su papildomu kodu, daugyba – daug sudėčių cikle, o dalyba – daug atimčių.

Loginės komandos:

p	Q	$p \wedge q$ <i>and</i>	$p \vee q$ <i>or</i>	$p \wedge q$ <i>exclusiveor</i>	$p \equiv q$ <i>equivalent</i>	$p / q$ <i>nand</i>
0	0	0	0	0	1	1
0	1	0	1	1	0	1
1	0	0	1	1	0	1
1	1	1	1	0	1	0

$$p \wedge q = (p / q) / (p / q)$$

$$p \vee q = (p / p) / (q / q)$$

$$p \wedge q = ((p / p) / q) / (p / (q / q))$$

$$p \equiv q = (p / q) / ((p / p) / (q / q))$$

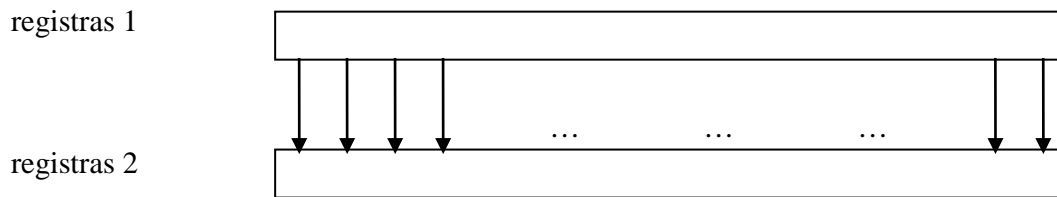
### ***Mikrokomandos***

Pagrindinė komanda yra persiuntimas iš registro į registrą. Persiuntimas tarp registrų gali būti:

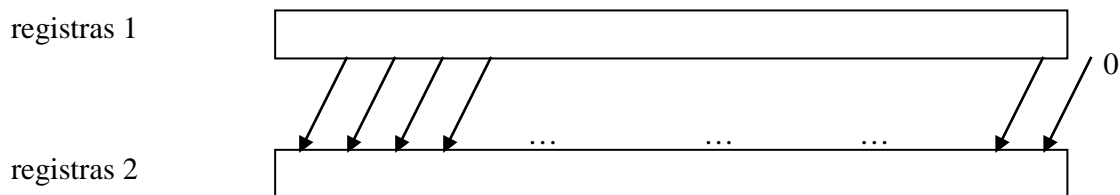
1. Tiesioginis
2. Asimetris
3. Dalinis
4. Grupinis

## KOMPIUTERIŲ ARCHITEKTŪRA

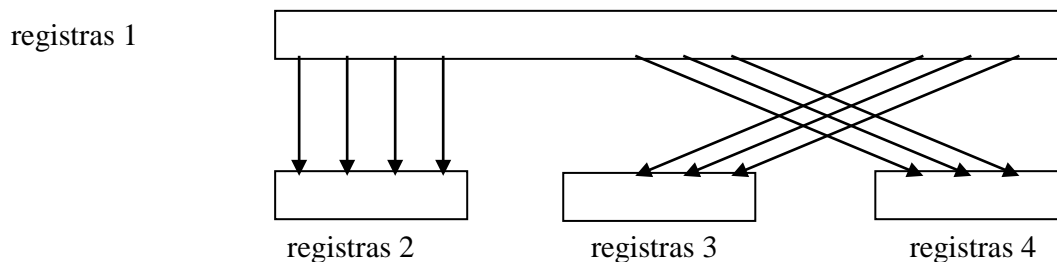
Dažniausiai naudojamas tiesioginis persiuntimas:



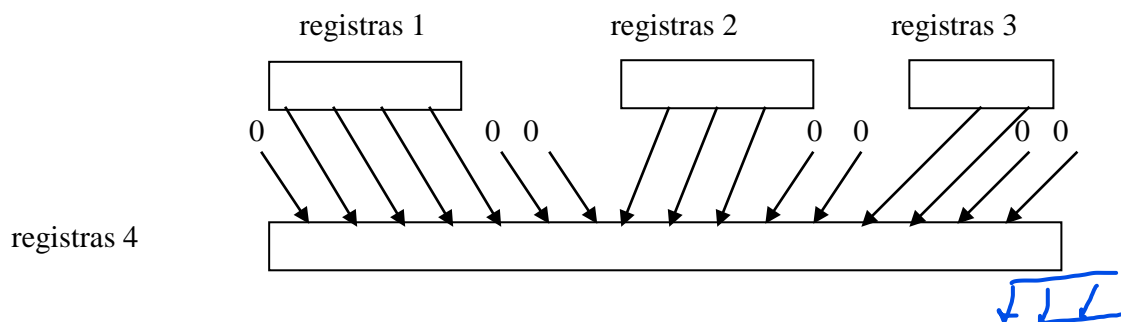
Asimetriniame persiuntime  $i$ -tasis bitas pereina į  $(i + k)$ -tąjį bitą, kai  $k > 0$  arba  $k < 0$ :



Dalinis persiuntimas yra tada, kai vieno registro dalys yra persiunčiamos įvairiems registrams, neišlaikant bitų numeracijos, o tik nuoseklumą:



Esant grupiniam perdavimui įvairių registrų turinių dalys perduodamos vienam registrui:



Pavyzdžiui, postūmio įrenginys veiksmui atlikti naudoja laikiną registrą ir asimetrinį persiuntimą  $registas\ 1 \rightarrow registros\ 2$  bei tiesioginį persiuntimą  $registas\ 2 \rightarrow registros\ 1$ . Postūmis per  $N$  bitų gaunamas pakartojus postūmį  $N$  kartų.

Postūmis į kitą pusę gali būti gautas tuo pačiu įrenginiu, bet tam reikia turėti kitą magistralę, kuri realizuoja kitos krypties asimetrinį persiuntimą.

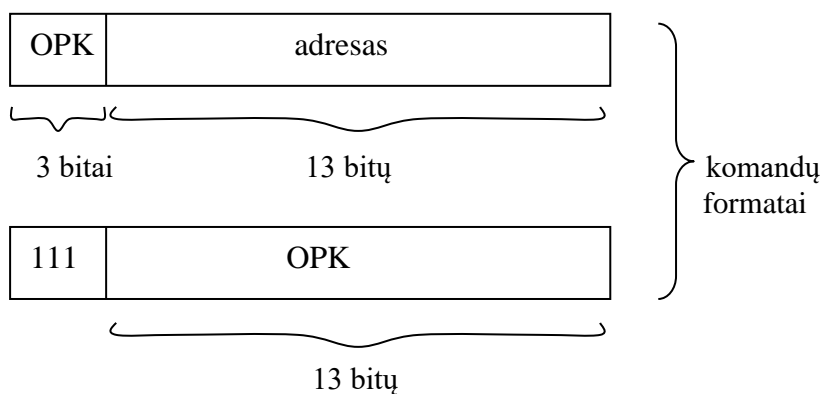
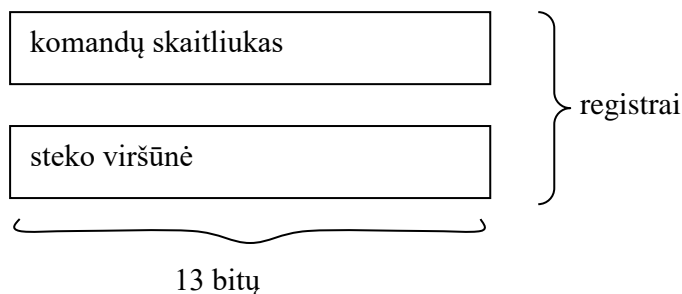
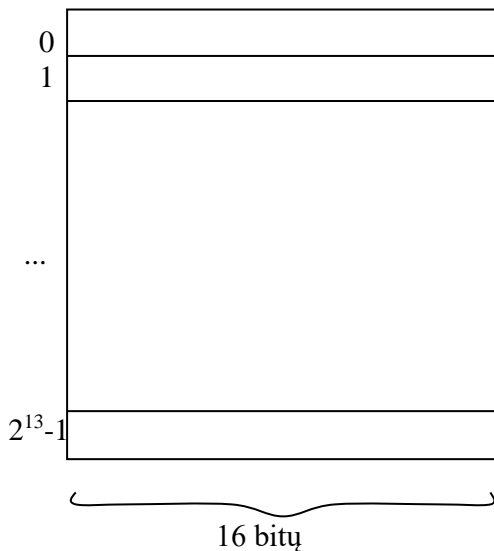
Kita komanda yra užrašymas į atmintį ir skaitymas iš atminties. Užrašymui ir skaitymui gali būti viena bendra arba dvi skirtingos komandos.

Trečia komanda yra sąlyginė. Tai yra arba speciali komanda, kuri tikrina bitą, arba komandoje laukas ('maskė') naudojamas kartu su koku tai kitu registru (rezultato požymis).

turim pagauti archit  
suprasti kuo skiriasi  
interpretuojamos masinos atmintis nuo mikroprogramavimo  
sita pagauti labai  
gerai turim...

## Interpretuojamas lygis

Panagrinėsime konkretų tradicinį komandų lygį, kuris realizuojamas interpretavimo būdu ir vadinamas interpretuojamu lygiu. Tegul turime virtualią mašiną su 16 bitų žodžiais, kurie adresuojami 0, 1, 2, ...,  $2^{13}$  žodžių, o baitų  $2^{14}$ . Adresinio registro ilgis yra 13 bitų, buferinio registro ilgis yra 16 bitų.



Tegul naudojamas stekinis atminties organizavimo principas, kai komandos operuoja su informacija, esančia steko viršūnėje, ir todėl tokios komandos neturi adreso lauko.

## KOMPIUTERIŲ ARCHITEKTŪRA

Tam, kad užpildyti steką ir paimti iš steko saugojimui, naudojamos vieno adreso komandos. Be to adresinės komandos naudojamos ir besąlygiam bei sąlyginiam valdymo perdavimui. Sąlyginės valdymo perdavimo komandos paima iš steko viršūnės žodį, jį patikrina ir priklausomai nuo patikrinimo rezultatų perduoda valdymą. Procedūros iškvietimo komanda grįžimo adresą talpina į steką, o valdymą perduoda pagal adresą. Aritmetinės komandos pradžioje paima antrą operandą, po to pirmą operandą ir rezultatą patalpina į steką. Aritmetiniai veiksmai gali būti atliekami su skaičiais papildomu kodu.

### Komandos:

		memory adress
PUSH	000	M

Komanda PUSH atminties žodis su adresu M patalpinamas į steko viršūnę.

[sitas komandas raso vartotojas

POP	001	M
-----	-----	---

[mikroprogramavimo vartotojas nemato, vyksta ventiliu atidarymas

Komanda POP iš steko viršūnės paimama reikšmė (žodis) ir užrašoma adresu M.

JUMP	010	M
------	-----	---

Komanda JUMP valdymas perduodamas adresu M.

JNEG	011	M
------	-----	---

Komanda JNEG atliekamas sąlyginis valdymo perdavimas – pereiti į komandą adresu M, jei steko viršūnėje esanti reikšmė yra  $< 0$ .

JZER	100	M
------	-----	---

Komanda JZER atliekamas sąlyginis valdymo perdavimas – pereiti į komandą adresu M, jei steko viršūnėje esanti reikšmė yra  $= 0$ .

JPOS	101	M
------	-----	---

Komanda JPOZ atliekamas sąlyginis valdymo perdavimas – pereiti į komandą adresu M, jei steko viršūnėje esanti reikšmė yra  $\geq 0$ .

CALL	110	M
------	-----	---

Komanda CALL valdymas perduodamas adresu M, o grįžimo adresas išsaugomas steko.

skaičiai nurodo kad bus komanda

ADD	111	00 ... 0000
-----	-----	-------------

Sudėtis: iš steko paimti operandus T ir S, juos sudėti ir rezultatą  $S + T$  užrašyti į steką.

SUB	111	00 ... 0001
-----	-----	-------------

## KOMPIUTERIŲ ARCHITEKTŪRA

Atimtis: iš steko paimti operandus T ir S, atimti ir rezultatą  $S - T$  užrašyti į steką.

MUL

111	00 ...	0010
-----	--------	------

Daugyba: iš steko paimti operandus T ir S, juos sudauginti ir rezultatą  $S \cdot T$  užrašyti į steką.

DIV

111	00 ...	0011
-----	--------	------

Dalyba: iš steko paimti operandus T ir S, padalinti ir rezultatą  $S / T$  užrašyti į steką.

RETURN

111	00 ...	0100
-----	--------	------

Komanda RETURN iš steko viršūnės paimamas grįžimo adresas ir juo perduodamas valdymas.

Yra galimas beadresinių komandų praplėtimas, pavyzdžiui, loginių operacijų komandomis.

### ***Interpretuojantis lygis***

Interpretuojančio lygio architektūra turi 15 registrų, 39 magistrales su ventiliais, 16-os bitų sumatorių ir postūmio per vieną bitą įrenginį.

Registrai:

PC – interpretuojamos mašinos komandų skaitliukas, registras yra 13 bitų;

SP – interpretuojamos mašinos steko rodyklė, registras yra 13 bitų;

IR – 16 bitų registras interpretuojamos mašinos vykdomos komandos saugojimui;

MAR – 13 bitų interpretuojamos mašinos adresinis registras;

MBR – 16 bitų interpretuojamos mašinos buferinis registras;

**general purpose**

A } mikroprogramų darbiniai **registrai**,  
B } 16 bitų;  
C }  
D }

**const.**

**X** – mikroprogramų ciklo skaitliukas, 16 bitų;

Yra dar penki registrai su pastoviomis reikšmėmis:

+1 }  
0 } konstantiniai  
-1 } registrai;  
15 }

SIGN registro reikšmė yra - 32768 1000 0000 0000 0000

kad zinoti kokia komanda bus vykdoma toliau

komandos, is viso 16 bitu, jei nereik adr

musu atminties zodis 16 bitu

jei reikia vieneto, galima ji pasiimti is cia. kitaip jo negausim. viena is egzaminu uzduociu bus sugalvoti kaip gauti skaičiu naudojantisuos registrus

naudinga tureti registra, kad nereiketu shift daryti

## KOMPIUTERIŲ ARCHITEKTŪRA

### Aritmetinis loginis įrenginys (ALU).

ALI susideda iš sumatoriaus, kurio įėjime yra du šešiolyktainiai skaičiai, o išėjime yra vienas skaičius – suma, išreikšta papildomu kodu, kuris iškart paduodamas postūmio įrenginiui. Postūmis gali būti į kairę arba į dešinę, pastūmus atsiranda nuliniai bitai. Ženklo bitas stumiant į dešinę nepereina į jauniausią bitą (pvz., pastūmus reikšmę 1000 0000 0000 0000 per vieną poziciją į dešinę, turėtume 0100 0000 0000 0000).

### Magistralės ir ventiliai.

Turim masinos atmintį ir joje komandas, mum reikia zinoti kelintą komanda vykdyti. PC pasako kokia komanda vykdyti

jei vienu metu atidarytume kelis, gautusi nesamone

#### I. Sumatoriaus įėjimas.

S - sumatorius

Kairysis įėjimas

- X paduodame i sumatorių 1.  $X \rightarrow S$  naudojama atėmimui 1 iš X, ciklų skaičiavimui. mag - 1
- SP galima paduoti tiesiai i S 2.  $SP \rightarrow S$  naudojama vienu padidinti arba sumažinti SP – steko viršūnę.
- PC - program counter 3.  $PC \rightarrow S$  naudojama PC perdavimui į atmintį per MBR (PC dydis yra 13 bitų, MBR dydis yra 16 bitų, taigi pločių skirtumas yra 3).
4.  $A \rightarrow S$  naudojama aritmetinėse operacijose bei perduodant į D ir MBR.
5.  $B \rightarrow S$  naudojama aritmetinėse operacijose bei perduodant į A, D ir MBR.
6.  $MBR \rightarrow S$  naudojama aritmetinėse operacijose.
- magistr. → 7.  $+1 \rightarrow S$  naudojama pridėti 1 prie registre esančios reikšmės.
8.  $0 \rightarrow S$  naudojama persiuntimui, pavyzdžiui  $C \rightarrow A$ .
9.  $-1 \rightarrow S$  naudojama atimti 1 iš registro.
10.  $SIGN \rightarrow S$  naudojama 32 bitų postūmiuose, kurie reikalingi sujungus du 16 bitų registrus, taip pat pernešimo bitų įskaitymui.

Dešinysis įėjimas.

11.  $SIGN \rightarrow S$  analogiškai 10 magistralei.
12.  $-1 \rightarrow S$  analogiškai 9 magistralei.
13.  $0 \rightarrow S$  analogiškai 8 magistralei, naudojama perdavimui iš A arba B į D arba MBR.
14.  $+1 \rightarrow S$  analogiškai 7 magistralei.
15.  $MBR \rightarrow S$  analogiškai 6 magistralei.
16.  $C \rightarrow S$  naudojama aritmetinėse operacijose bei perdavimui iš C į A, D ir MBR
17.  $0 \rightarrow S$  naudojama aritmetinėse operacijose bei perdavime iš D į A, MBR.

#### II. Perdavimas iš registro į registrą. kai kuriuos perdavimus galima atlikti be sumavimo įrenginio, tiesiai perduoti

18.  $IR \rightarrow PC$  naudojama vykdančią valdymo perdavimo komandas, magistralės plotis yra 13 bitų.
19.  $PC \rightarrow MAR$  naudojama komandos paėmimui iš atminties.
20.  $SP \rightarrow MAR$  naudojama darbui su steko viršūne.
21.  $IR \rightarrow MAR$  naudojama komandoms PUSH ir POP. Magistralės plotis 13 bitų.
22.  $MBR \rightarrow IR$  naudojama vykdomos komandos paėmimui.
23.  $MBR \rightarrow A$
24.  $MBR \rightarrow B$
25.  $MBR \rightarrow C$

} registrų  
užkrovimas

MAR bandant perduoti i D nerasim tokios komandos, nepavyks

## KOMPIUTERIŲ ARCHITEKTŪRA

26. MBR → D

galima i X registra  
patalpinti 15

27. 15 → X naudojama ciklinėje operacijoje pradinei skaitliuko reikšmei suformuoti.

complement

### III. Papildomo kodo sukūrimas ir postūmis.

→ 28. Kai ši magistralė atidaryta, kairysis signalas patenka į sumatorių papildomu kodu.

atvirkstinis kodas  
complement???

29. Kai ši magistralė atidaryta, dešinysis signalas patenka į sumatorių papildomu kodu.

30. Kai ši magistralė atidaryta, sumatoriaus išėjimo signalas pastumiamas per vieną bitą į kairę prieš perduodant priėmėjui registrai.

visada stumiama  
per viena

31. Kai ši magistralė atidaryta, sumatoriaus išėjimo signalas pastumiamas per vieną bitą į dešinę prieš perduodant priėmėjui registrai.

### IV. Sumatoriaus įėjimas.

32. S → X naudojama ciklinėse operacijose keičiant X reikšmę nuo 15 iki -1.

33. S → A naudojama rezultatui užrašyti į registrą A.

34. S → SP naudojama atliekant veiksmus su steku, vienu didinant arba mažinant SP – steko viršūnės registrą.

35. S → PC naudojama perėjime prie sekančios komandos.

36. S → MBR naudojama rezultatui užrašyti į atmintį.

37. S → D naudojama rezultatui užrašyti į registrą D.

is sumatorio galima  
irasyti i tam tikrus  
registrus

i B ir C reg. rezultato irasyti  
negalima

### V. Apsikeitimas su interpretuojamos mašinos atmintimi.

MAR rodo i adresa  
o MBR zodi, kuri irasysim

38. → MBR iš atminties imamas žodis, kurio adresas yra MAR, ir jis užrašomas į atmintį.

39. MBR → žodis, esantis MBR užrašomas į atmintį adresu MAR.

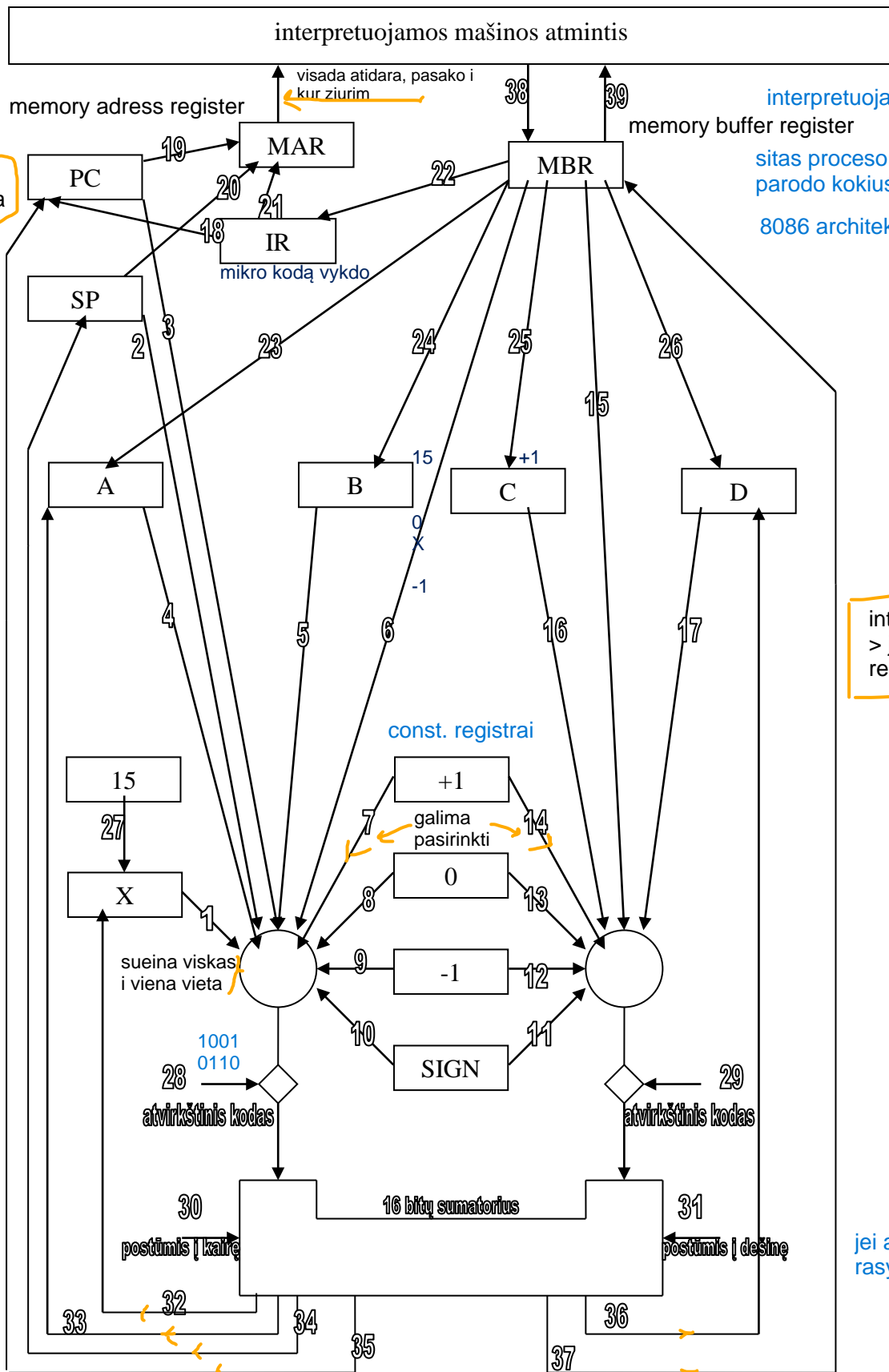
Magistralė, jungianti registrą MAR su interpretuojamos mašinos atmintimi, neturi ventilio. Todėl signalas, einantis šia magistrale, nėra programiškai valdomas. Ši magistralė visada yra atidaryta.

Perdavimas iš registro gali būti lygiagretus, bet perdavimas į registrą turi būti nuoseklus.

# KOMPIUTERIŲ ARCHITEKTŪRA

## Mikroprograminio lygio architektūra

push, pop...



interpretuojanti masina -  
sitas procesorius paprastas -  
parodo kokius ventilius  
8086 architektura sunkesne

instrukcija MUL  
> ji kazkaip  
realizuota...

jei atidarysim 36,  
rasykim i registra D



## KOMPIUTERIŲ ARCHITEKTŪRA

### Ventilių darbas

paimam aukšto lygio komanda ir norim padaryti sudėti

Interpretuojamos mašinos komanda **ADD** gali būti įvykdyta per keturis žingsnius:

1. Paimti antrą operandą iš steko ir užrašyti į registrą.
2. Paimti pirmą operandą iš steko ir užrašyti į registrą.
3. Suformuoti sumą.
4. Sumą užrašyti į steką.

Šie keturi žingsniai gali būti įvykdyti per tris ciklus, inicijuojamus taktiniais impulsais. Atidarant ir uždariant ventilius valdoma perdavimų sinchronizacija: ryšyje *registras 1* → *registras 2* → *registras 3*, perduoti *registras 2* → *registras 3* galima tik tada, kai *registras 1* → *registras 2* signalas jau atėjo. Priešingu atveju gali kilti neapibrėžtumas to, kas bus paduota į *registrą 3*.

### Pocikliai

Tam, kad išvengti neapibrėžtumų yra įvedami pocikliai. ← kad išvengti nesamonių ir susimaišymų

Pirmame pociklyje gali būti atidarytos magistralės 1 – 29 ;

antrame pociklyje – magistralės 30 – 37 ; → visi outputai iš sumavimo ir t.t.

trečiame pociklyje – 38 arba 39 magistralės. → darbas su atmintimi

GATE komanda visko nevykdys  
uzkart, vykdys pocikliais

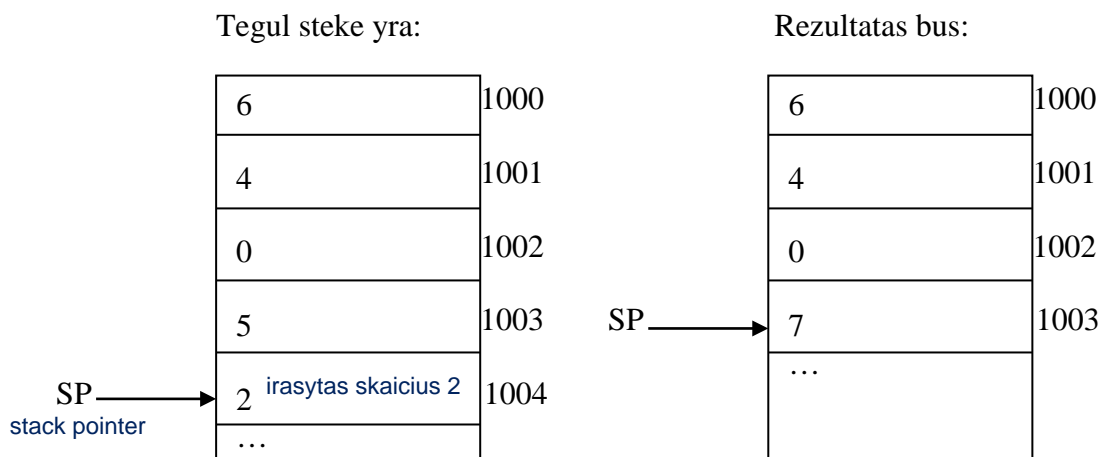
Pirmame pociklyje atliekamas **apsikeitimas tarp registrų**; antrame pociklyje atliekami **veiksmai sumatoriuje**, postūmio įrenginyje, rezultatai paduodami į registrus; trečiame pociklyje **apsikeičiama su atmintimi**.

Realiai atminties ciklas yra didesnis nei procesoriaus ciklas, bet paprastumo dėlei į tai nekreipiame dėmesio.

pasiskaityti

### Komandos ADD realizacija

Tegul komanda jau yra registre IR ir yra nustatytas operacijos kodas.

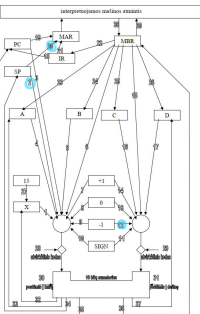


ivykde komanda add paimsime  
2, sumazinsime SP ir paimsime 5  
ir po to padesime 7

gali buti ir daugiau dalyku irasyta

# KOMPIUTERIŲ ARCHITEKTŪRA

reikia 3 ciklu, kad padarytume sudėti kiekviename cikle iskvieciame GATE

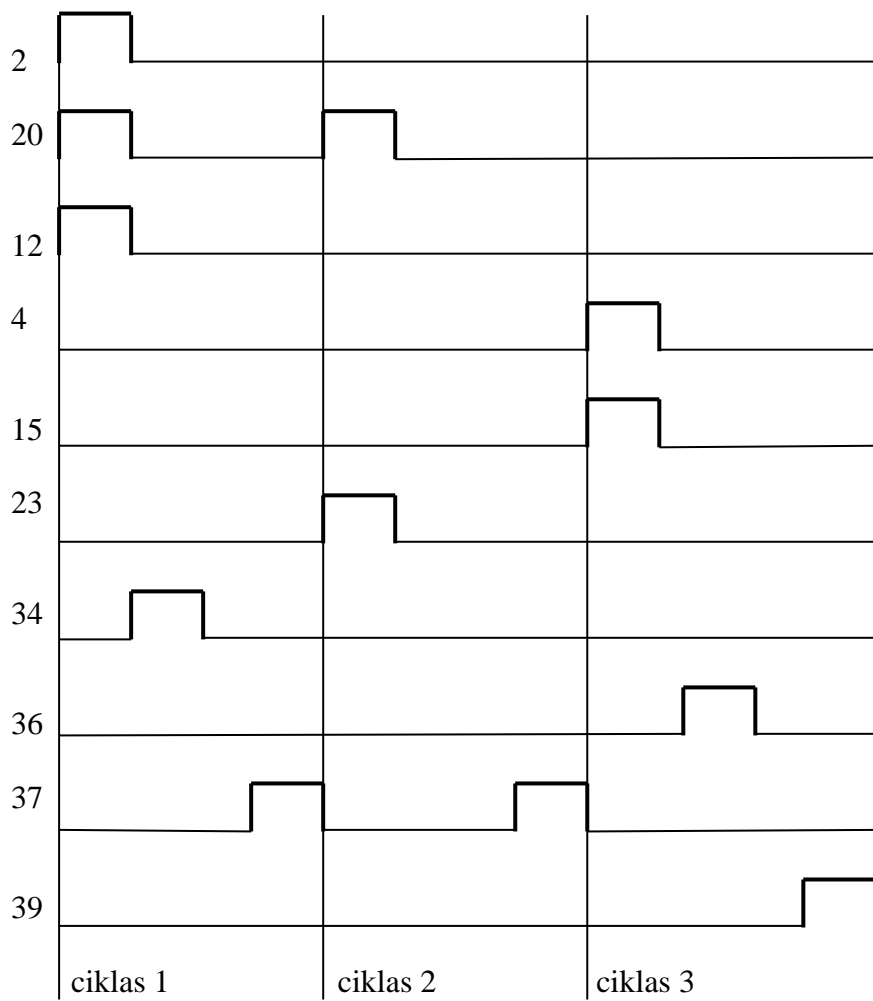


	Pocikliai	SP	MAR	MBR	A	Schemas
ciklas 1	1	1004	?	?	?	20, 2, 12
	2	1004	1004	?	?	34
	3	1003	1004	?	?	38
ciklas 2	1	1003	1004	2	?	20, 23
	2	1003	1003	2	2	
	3	1003	1003	2	2	38
ciklas 3	1	1003	1003	5	2	4, 15
	2	1003	1003	5	2	37
	3	1003	1003	7	2	39

atidarome magistrales  
reikšmes dar nekeičiam SP  
kas MBR perkeliame į registra A, su 23 mažinsim MAR  
nusiskaitome 5 iš MBR  
paduoti registra A sk 2, iš MBR paduoti 5  
output į MBR atgal  
sujungiam MBR su atmintimi

Pavaizduosime schematiškai magistralių atidarymą, realizuojantį sudėties komandą:

magistrales



Atsargiai! kažkur yra klaida čia ???

3 pocikliai

## Ventilių mikroprograminis valdymas

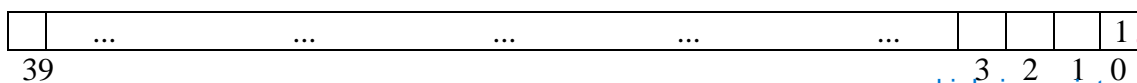
visai kita atmintis,  
mikroprocesorius

Tam, kad gauti interpretuojamos mašinos programos vykdymo efektą, reikia atitinkamais laiko momentais atidarinėti ir uždarinėti atitinkamus ventilius. Interpretuojančio lygio komandų sistema susideda iš dviejų komandų: GATE ir TEST. Operacijos kodas yra reikšmė 1 (komanda GATE) arba 0 (komanda TEST) dešiniajame bite.

Mikrokomanda TEST naudojama mikrokomandų GATE vykdymo nuoseklumui pakeisti, priklausomai nuo tam tikrų sąlygų.

jei komanda ... atidaro ventilius

GATE - komandos pavadinimas



kiekviena vieta ventiliui,  
magistralei

Nulinis bitas yra operacijos kodas. Likusieji 39 komandos bitai atitinka 39 ventilius, bito reikšmė 1 reiškia, kad atitinkamas ventilis turi būti atidarytas, o bito reikšmė 0 reiškia, kad atitinkamas ventilis turi būti uždarytas. Iš viso komanda užima 40 bitų.

Mikroprograma, išreiškianti sudėtį:

jei 1, tai komanda gate  
jei 0, tai komanda test

39	38	37		35	34	33		24	23	22	21	20	19		16	15	14	13	12	11		4	3	2	1	0
0	1	0	..	0	1	0	..	0	0	0	0	1	0	..	0	0	0	0	1	0	..	0	0	1	0	1
0	1	0	..	0	0	0	..	0	1	0	0	1	0	..	0	0	0	0	0	0	..	0	0	0	0	1
1	0	1	..	0	0	0	..	0	0	0	0	0	0	..	0	1	0	0	0	0	..	1	0	0	0	1

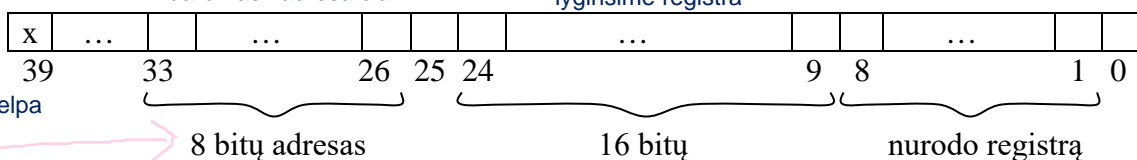
jos tikslas - arba sokti i adresa kuris nurodytas cia arba vykdyti kita komanda

palyginti su registru  
nurodytu. jei tiesa ar  
netiesa tai sokame...  
galime sokti i betkokia  
kita komanda

TEST

valdymo perdavimas  
daromas i adresa cia

gali nurodyti su kuo  
lyginsime registra



visi adresai sutelpa  
i 8 bitus

1 – 8 bitais pasakome, kuris registras dalyvaus sąlygos formulavime: bito, atitinkančio registrą, reikšmė yra 1.

- 1-as bitas atitinka registrą A;
- 2-as bitas atitinka registrą B;
- 3-as bitas atitinka registrą C;
- 4-as bitas atitinka registrą D;
- 5-as bitas atitinka registrą MBR;
- 6-as bitas atitinka registrą X;
- 7-as bitas atitinka registrą IR;
- 8-as bitas atitinka registrą Ø.

Registrai yra 16 bitų. 9 – 24 bite reikšmė 1 nurodo, kuris nustatyto registro bitas dalyvaus sąlygos formulavime. 9-tas bitas atitinka nulį registro bitą, 24-tas bitas atitinka penkiolikąjį registro bitą. is kurios puses surasom...

## KOMPIUTERIŲ ARCHITEKTŪRA

25-tas bitas yra konstanta, jis yra palyginimo bitas.

Kiekviena mikrokomanda TEST apibrėžia vieną tikrinamą bitą iš 128 galimų. Tokiais bitais yra registrai A, B, C, D, MBR, X, IR, Ø visi 16 bitų ( $8 \cdot 16 = 128$ ). Jeigu nustatytas bitas ir 25-tasis komandos bitai sutampa, tai mikrokomandų vykdymo nuoseklumas pakinta ir sekančios vykdomos mikrokomandos adresas nustatomas mikrokomandos TEST 26 – 33 bitais. Priešingu atveju vykdymo nuoseklumas nepasikeičia.

25 bitas atsako ar pasiekiam ar sokam

lyginti gali tik pagal viena bita, lyginam pasirinkta viena, architektūros ribojimas?

Bitų grupėse 1 – 8 ir 9 – 24 tik vieno bito reikšmė kiekviename grupėje gali būti 1.

Kadangi mikrokomandos ilgis yra 40 bitų, dar turime nepanaudotus 26 – 39 bitus, t.y., 14 bitų. Galėtume jais adresuoti  $2^{14}$  mikroatminties žodžių (mikroatminties žodžio ilgis yra 40 bitų). Tačiau interpretatoriui parašyti pakanka  $2^8 = 256$  mikroatminties žodžių, todėl adresui nurodyti naudojame 8 bitus, t.y., 26 – 33 bitus, o likusieji 34 – 39 yra nenaudojami.

Mikroprograma saugoma mikroatmintyje, kuri logiškai skiriasi nuo interpretuojamos mašinos atminties. Fiziškai abi atmintys gali būti realizuotos neatskirtos, tik paskirstant į dvi dalis. Mikroatminties žodžio ilgis yra 40 bitų. Kaip jau buvo minėta, apsiribosime  $2^8$  žodžių adresine erdve.

po 16 bitu su adresavimu iki 13 bitu

turim skaiciuoti kuria mikrokomanda vykdome

Mikroprograminio lygio architektūroje turi būti mikrokomandų skaitliukas, atminties adresinis registras, atminties buferinis registras, mikrokomandų vykdymo registras. Dėl mikroprograminio lygio paprastumo atminties adresinį registrą galima sutapatinti su mikrokomandų skaitliuku, o atminties buferinį registrą – su mikrokomandų vykdymo registru. MPC bus mikroprogramos komandų skaitliukas, o MIR – mikrokomandų interpretavimo (vykdymo) registras.

Aprašysime centrinio procesoriaus darbą. Taktinis impulsas užduoda naujo ciklo pradžią. Atminties žodis su adresu iš MPC užrašomas į registrą MIR. Jeigu tai yra mikrokomanda GATE, tai atitinkami ventiliai, nurodyti vienetiniiais bitais, atidaromi atitinkamo pociklio bėgyje. Jeigu tai yra mikrokomanda TEST, tai nustatomas reikiamas bitas ir jis patikrinamas. Jeigu bitai sutampa, tai mikrokomandos bitai 26 – 33 pasiunčiami į MPC. Mikrokomanda yra įvykdoma ciklo bėgyje, o atitinkami ventiliai atidaromi atitinkamame pociklyje.

kiekvienas impulsas ciklo pradžia

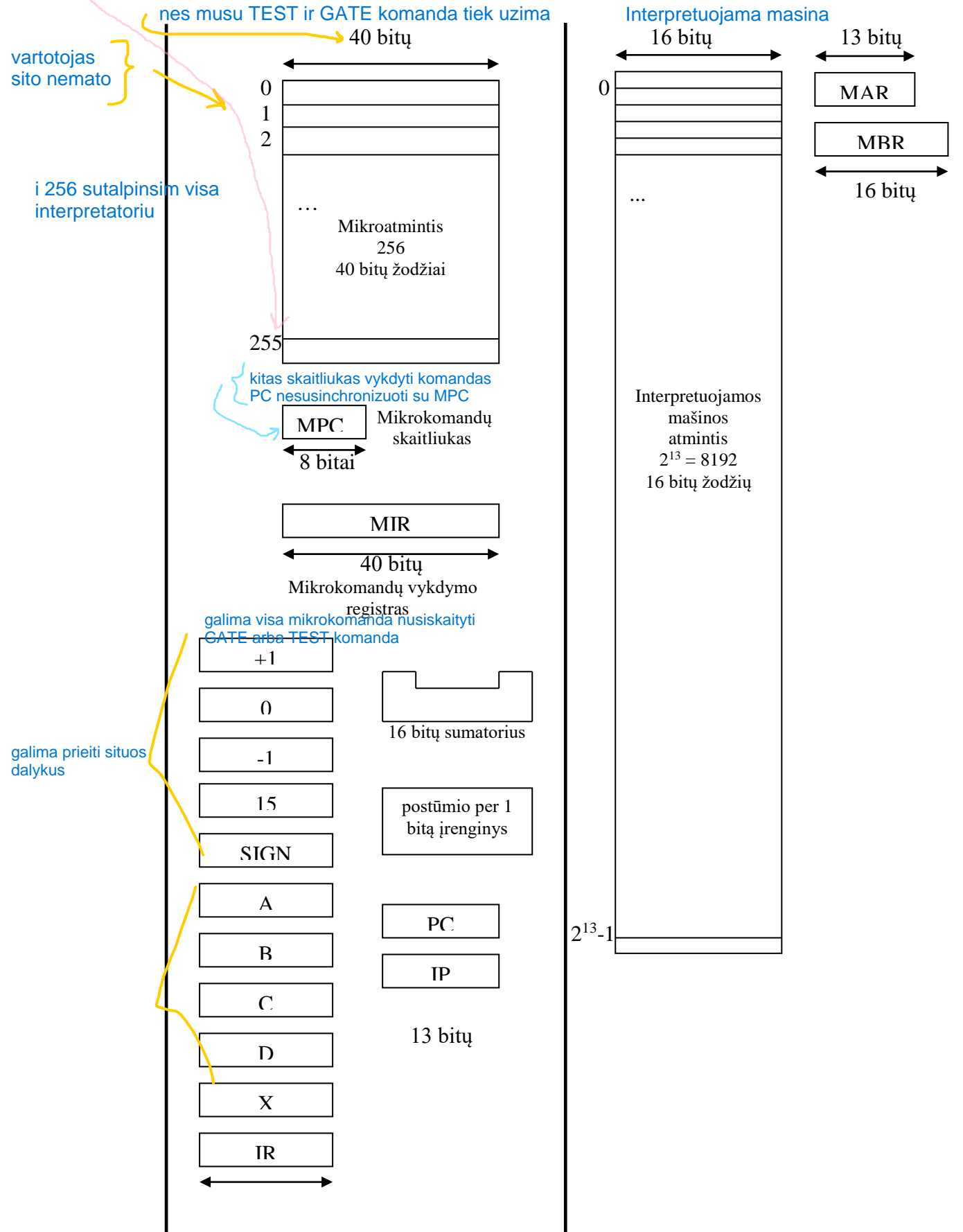
sokame i kita adresa

Sudarant mikroprogramas galima operuoti 40 bitų komandomis, bet galimas ir kitas kelias, kai pradžioje apibrėžiamos ir realizuojamos bazinės operacijos mikrokomandų lygyje. Vėliau reikalingos mikroprogramos gaunamos atitinkamos programos, sudarytos interpretuojamos mašinos kalba, darbo rezultate.

# KOMPIUTERIŲ ARCHITEKTŪRA

## Interpretuojančios mašinos architektūra

## Interpretuojama masina



16 bitų

Mikroprogramavimo kalba

Programuoti mikrokomandų kodais yra labai sudėtingas darbas ir todėl reikalinga mikroprogramavimo kalba. Naudosime kalbą MPL (Micro Programming Language) . Ypatumas yra tas, kad mikrokomandos užduoda lygiagrečiai vykdomus veiksmus, persiuntimus įvairiomis magistralėmis. Kiekviena MPL eilutė apibrėžia vieną mikrokomandą, net jeigu vienoje eilutėje yra keli MPL operatoriai. Mikroprogramos vienos eilutės operatoriai vykdomi lygiagrečiai.

Mikrokomandos GATE programavimas.

Perdavimai tarp registrų nurodomi priskyrimo operatoriais.  
Pavyzdžiui:

A = MBR; 23 magistralė su GATE atsidaro  
reiškia, kad 23 magistralė vykdant šį operatorių yra atidaryta.

Operatorių nuoseklumas yra nesvarbus eilutės ribose.  
A = MBR; MAR = IR; X = 15; reiškia magistralės atidaryma nurodo, kad atsidaro magistralė tarp X ir 15  
MAR = IR; X = 15; A = MBR;  
Abi eilutės reiškia tą patį – 21, 23 ir 27 magistralių atidarymą.

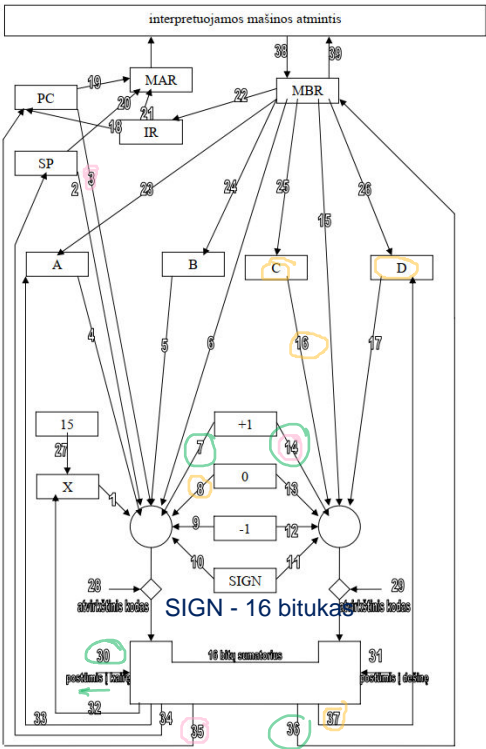
Tačiau operatoriai :  
A = MBR ;  
MAR = IR ;  
X = 15;  
reiškia tris atskiras mikrokomandas.

Sumatoriaus panaudojimas nurodomas operacija '+'.  
MBR = A + C; reiškia 4, 16 ir 36 magistralių atidarymą.  
SP = SP + (-1); reiškia 2, 12 ir 34 magistralių atidarymą. Tai yra steko rodyklės sumažinimas vienetu.  
A = MBR; PC = PC + 1; reiškia 3, 14, 23 ir 35 magistralių atidarymą.  
D = 0 + C; reiškia 8, 16 ir 37 magistralių atidarymą. Registrai C ir D tiesiogiai nesujungti, todėl norėdami perduoti iš C į D naudojames sumatoriumi.

Atvirkštinis kodas gaunamas atidarant 28 arba 29 magistrale. Tokį veiksmą atitinka funkcija COM. 28 ir 29 magistralės valdo reikšmės konvertavimą, tačiau norint gauti reikšmę su priešingu ženklu, reikia dar pridėti vienetą. Pavyzdžiui, norėdami iš 5 gauti -5, elgiamės taip:

0000 0101 = 5  
1111 1010 invertavimas  
1 pridėdame vienetą  
1111 1011 = -5

Pavyzdžiui: pavers neigiama i teigiama ar teigiama i neigiama  
MBR = COM(MBR) + 1; reiškia 6, 14, 28, 36 magistralių atidarymą. Tai yra buferiniame registre esančios reikšmės ženklo pakeitimas.



## KOMPIUTERIŲ ARCHITEKTŪRA

$A = 0 + \text{COM}(\text{SIGN})$ ; reiškia 8, 11, 29, 33 magistralių atidarymą. Tai yra maksimalios teigiamos reikšmės priskyrimas registrui A.

Sumatoriaus išėjime reikšmę galima pastumti per vieną bitą pasinaudojant magistralėmis 30 ir 31. Postūmiai į kairę arba į dešinę nurodomi funkcijomis

→ LEFT\_SHIFT ir RIGHT\_SHIFT. Pavyzdžiui:

$\text{MBR} = \text{LEFT\_SHIFT}(1 + 1)$ ; reiškia 7, 14, 30, 36 magistralių atidarymą. Tokiu veiksmu į buferinį registrą užrašome reikšmę 4.

$A = \text{RIGHT\_SHIFT}(A + 0)$ ; reiškia 4, 13, 31, 33 magistralių atidarymą. Tokiu veiksmu į registrą A užrašoma jame buvusi reikšmė, padalinta iš dviejų. postūmis i desine daro dalyba is 2

Apsikeitimas su atmintimi (skaitymas ir rašymas) yra atliekamas panaudojant funkciją MEMORY(MAR), kuri iššaukia 38 arba 39 magistralės atidarymą.

### Mikrokomandos TEST užrašymas.

IF BIT( $k$ ,  $reg$ ) =  $b$  THEN GOTO  $label$ ;      jei  $k = 7$ , tai bus lyginamas 7 bitas

$k$  yra bito numeris registre,  $k = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$ ;

$reg$  yra registras,  $reg = \{A, B, C, D, \text{MBR}, \text{IR}, X, \emptyset\}$ ;

$b$  atitinka 25-tą komandos TEST bitą,  $b = \{0, 1\}$ ;

atminties adresas -  $label$  nurodo vardą (žymė yra simbolinis vardas), kuriuo perduodamas valdymas, atitinka 26 – 33 komandos TEST bitus.

Jeigu  $b = 0$ ,  $reg = \emptyset$  ir  $k$  yra bet kuris bitas, tada valdymo perdavimas yra besąlyginis, tai yra užrašoma **GOTO  $label$ .**

Pavyzdžiai:

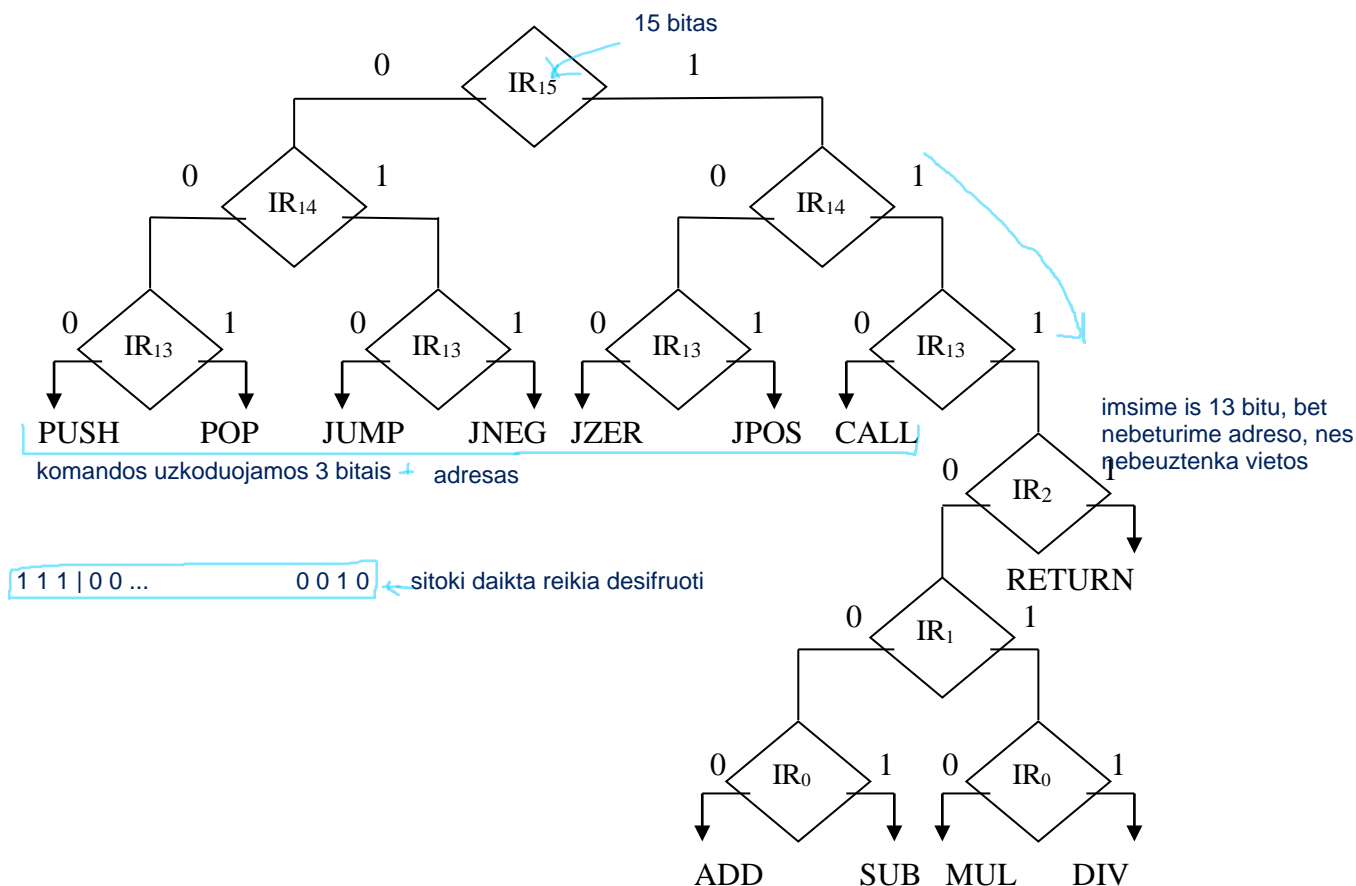
jeigu reiktu visus bitus palyginti darytume if if if if if if if if if if...  
neturime tokiu irankiu, architekturos ribojimas  
IF BIT(13, IR) = 1 THEN GOTO POP;      paimam registra IR, patikriname ar jo 13 bitas = 1, jei taip sokame

IF BIT(15, X) = 0 THEN GOTO NULLOOP;

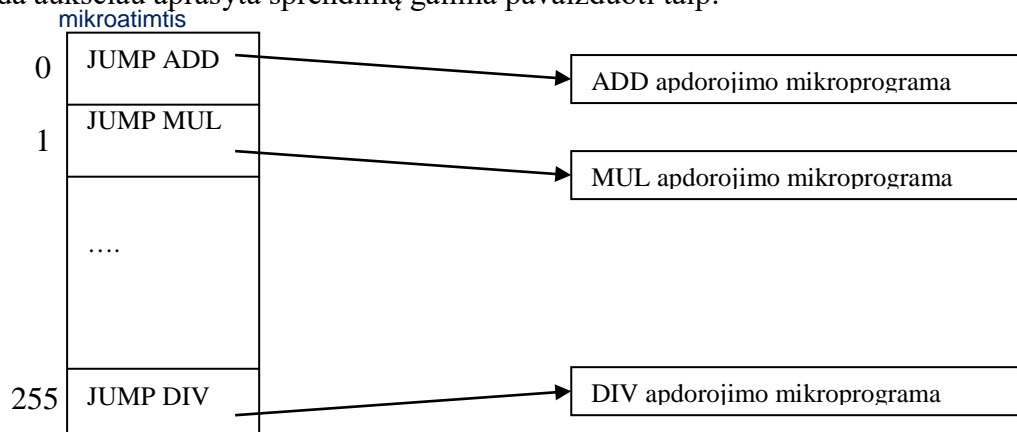
Iš esmės, MPL yra assemblerio lygio kalba, nes ji leidžia valdyti generuojamą kodą.

su C kalba tai kas bus sugeneruota priklauso nuo optimizavimo  
su assembleriu kokia instrukcija uzrasome, tokia ir bus paduota procesoriui

## Operacijos kodo dešifratoriaus algoritmas



Šis operacijos dekodavimo būdas yra nepakankamai efektyvus. Jeigu operacijos kodas būtų 8 bitų, tai reikėtų iki 8 mikrokomandų TEST. Tam, kad to išvengti, galima būtų panaudoti operacijos kodo reikšmę kaip modifikaciją perduodant valdymą mikrokomandai, apdorojančiai atitinkamą operaciją. Tiksliau tai būtų mikrokomandos TEST, kurios perduotų valdymą į atitinkamos interpretuojamos komandos apdorojimo mikroprograminę šaką. Taigi tegu operacijos kodas yra 8 bitai ir tegu komandos ADD operacijos kodas yra 0000 0000, komandos MUL operacijos kodas yra 0000 0001, ir t.t., tada aukščiau aprašyta sprendimą galima pavaizduoti taip:





Tokios galimybės realizacijai reikėtų dalinai modifikuoti mūsų interpretuojančios mašinos architektūrą.

**Interpretuojamos mašinos interpretatorius**

0     MAINLOOP: MAR = PC; MBR = MEMORY(MAR);  
1             IR = MBR; PC = PC + 1;     padidiname program counter i nusiskaitome kita instrukcija  
2             IF BIT(15, IR) = 1 THEN GOTO OP4567;  
3             IF BIT(14, IR) = 1 THEN GOTO OP23;  
4             IF BIT(13, IR) = 1 THEN GOTO POP;     skaitome instrukciję desifravimas  
5     PUSH:     MAR = IR; SP = SP + 1; MBR = MEMORY(MAR);     ivykdome  
6             MAR = SP; MEMORY(MAR) = MBR;     padidiname PC  
7             GOTO MAINLOOP;     skaitome... ir t.t.  
8     POP:     MAR = SP; SP = SP + (-1); MBR = MEMORY(MAR);  
9             MAR = IR; MEMORY(MAR) = MBR;  
10            GOTO MAINLOOP;  
11     OP23:    IF BIT(13,IR) = 1 THEN GOTO JNEG;  
12     JUMP:    PC = IR;  
13            GOTO MAINLOOP;  
14     JNEG:    MAR = SP; SP = SP + (-1); MBR = MEMORY(MAR);  
15            IF BIT(15, MBR) = 1 THEN GOTO JUMP;  
16            GOTO MAINLOOP;  
17     OP4567: IF BIT(14, IR) = 1 THEN GOTO OP67;  
18            IF BIT(13, IR) = 1 THEN GOTO JPOS;  
19     JZER:    MAR = SP; SP = SP + (-1); MBR = MEMORY(MAR);  
20            IF BIT(15, MBR) = 1 THEN GOTO MAINLOOP;  
21            MBR = MBR + (-1);  
22            IF BIT(15, MBR) = 1 THEN GOTO JUMP;  
23            GOTO MAINLOOP;  
24     JPOS:    MAR = SP; SP = SP + (-1); MBR = MEMORY(MAR);  
25            IF BIT(15, MBR) = 0 THEN GOTO JUMP;  
26            GOTO MAINLOOP;  
27     OP67:    IF BIT(13,IR) = 1 THEN GOTO OP7;  
28     CALL:    SP = SP + 1;  
29            MAR = SP; MBR = PC + 0; MEMORY(MAR) = MBR;  
30            GOTO JUMP;  
31     OP7:     IF BIT(2, IR) = 1 THEN GOTO RETURN;  
32            IF BIT(1, IR) = 1 THEN GOTO MULDIV;     i buferini reigistra irasome adresu MAR tai kas MBR  
33     ADDSUB:   MAR = SP; SP = SP + (-1); MBR = MEMORY(MAR);  
34            IF BIT(0, IR) = 0 THEN GOTO SUM;  
35            MBR = COM(MBR) + 1;  
36     SUM:     MAR = SP; A = MBR; MBR = MEMORY(MAR);  
37            MBR = A + MBR; MEMORY(MAR) = MBR;  
38            GOTO MAINLOOP;  
39     MULDIV:   IF BIT(0, IR) = 1 THEN GOTO DIV;  
40     MUL:     MAR = SP; SP = SP + (-1); MBR = MEMORY(MAR);  
41            C = MBR; MAR = SP; MBR = MEMORY(MAR);  
42            X = 15; A = 0 + 0; D = 0 + 0;  
43            IF BIT(15, MBR) = 0 THEN GOTO MULLOOP;

pvz:  
18 ir 35 magistr.  
atidaryti kartu  
neleidžia architektura

## KOMPIUTERIŲ ARCHITEKTŪRA

```
44      MBR = COM(MBR) + 1;
45      B = MBR; MBR = 1 + COM(C);
46      C = MBR; MBR = B + 0;
47  MULLOOP: IF BIT(0, MBR) = 0 THEN GOTO NOADD;
48          A = A + C;
49  NOADD:   MBR = RIGHT_SHIFT(MBR + 0);
50          D = RIGHT_SHIFT(0 + D);
51          IF BIT(0, A) = 0 THEN GOTO NOCARRY;
52          D = SIGN + D;
53  NOCARRY  A = RIGHT_SHIFT(A + 0);
54          IF BIT(15, C) = 0 THEN GOTO MULEND;
55          IF BIT(14, A) = 0 THEN GOTO MULEND;
56          A = A + SIGN;
57  MULEND:  X = X + (-1);
58          IF BIT(15, X) = 0 THEN GOTO MULLOOP;
59          MBR = 0 + D; MEMORY(MAR) = MBR;
60          GOTO MAINLOOP;
61  DIV:     MAR = SP; SP = SP + (-1); MBR = MEMORY(MAR);
62          C = MBR; B = MBR; MAR = SP; A = 0 + 0;
                                   MBR = MEMORY(MAR);
63          D = MBR; MBR = COM(MBR) + 1; X = 15;
64          IF BIT(15, MBR) = 1 THEN GOTO DVDPOS;
65          D = MBR; MBR = COM(B) + 1;
66          B = MBR;
67  DVDPOS:  MBR = 1 + COM(C);
68          IF BIT(15, C) = 0 THEN GOTO DIVLOOP;
69          C = MBR; MBR = COM(MBR) + 1;
70  DIVLOOP: A = LEFT_SHIFT(A + 0);
71          IF BIT(15, D) = 0 THEN GOTO NOCARRY2;
72          A = A + 1;
73  NOCARRY2: D = LEFT_SHIFT(0 + D);
74          A = A + MBR;
75          IF BIT(15, A) = 1 THEN GOTO DIVNEG;
76          D = D + 1;
77  DIVNEG:  IF BIT(15, A) = 0 THEN GOTO DIVPOS;
78          A = A + C;
79  DIVPOS:  X = X + (-1);
80          IF BIT(15, X) = 0 THEN GOTO DIVLOOP;
81          IF BIT(15, B) = 0 THEN GOTO DIVEND;
82          D = 1 + COM(D);
83  DIVEND:  MBR = 0 + D; MEMORY(MAR) = MBR;
84          GOTO MAINLOOP;
85  RETURN:  MAR = SP; SP = SP + (-1); MBR = MEMORY(MAR);
86          IR = MBR;
87          GOTO JUMP;
```

visos mikroprogramos sutelpta i 87 eilutes, realiai galime tureti 256 komandas

### Daugybės algoritmas.

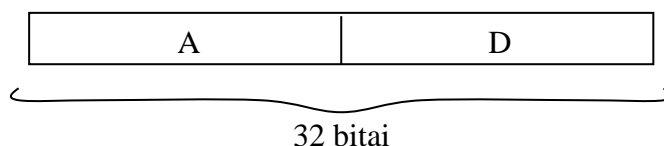
Atliekama daugyba stulpeliu iš karto kaupiant dalinių sandaugų sumą. Pavyzdžiui:

```
  1 1 0 1
  1 0 0 1
```

## KOMPIUTERIŲ ARCHITEKTŪRA

$$\begin{array}{r} 1101 \\ 0000 \\ 0000 \\ \hline 1101 \\ 1110101 \end{array} \left. \vphantom{\begin{array}{r} 1101 \\ 0000 \\ 0000 \\ 1101 \\ 1110101 \end{array}} \right\} \begin{array}{l} \text{sumuojamos dalinės} \\ \text{sandaugos} \end{array}$$

Registrai A ir D naudojami kaip 32 bitų sumatoriaus: 16 vyresnių bitų yra registre A ir 16 jaunesnių bitų yra registre D.



Dauginamasis yra registre C, o daugiklis yra registre MBR. Interpretatoriaus eilutės 40 – 46 formuoja pradines registrų reikšmes. Jomis yra suformuojama tokia pradinė situacija, kad antrasis operandas – daugiklis būtų  $\geq 0$ . Jeigu antrasis operandas buvo neigiamas, tai tada pakeičiami abiejų operandų ženklai, ir rezultate sandauga nepasikeičia. Ženklo keitimą reikia atlikti naudojant registrą MBR, nes registras C turi įėjimą tik iš MBR. Laikinam MBR reikšmės saugojimui panaudojamas registras D.

Tam, kad nereikėtų dalinių sandaugų vis pridėti prie skirtingų bitų, pridėdant prie registro A, atliekamas pastūmimas į dešinę. Registre A prieš pastūmimą į dešinę yra patikrinama, ar nuliniame registro bite yra 1, jeigu taip, tai jį reikia pernešti į registrą D. Tokį pernešimą atliekame pridėdami registre SIGN esančią reikšmę. Tada pastumiame į dešinę.

Registre A ženklo bito plėtimas irgi atliekamas pastūmimu į dešinę. Registre A tikriname 14-to bito reikšmę, nes nulis 15-tame bite gali būti ir dėl to, kad dar nieko neužrašėme į registrą A. Reikia patikrinti, ar buvo postūmis, t. y. ar 14-tame bite yra 1 – postūmio rezultate atėjęs ženklo bitas.

16 bitų perpildymas aparatūriškai nėra fiksuojamas.

### Dalybos algoritmas.

Surandama skaičiaus dalis, kuri dalosi iš daliklio. Jeigu nesidalina, tai liekana papildoma ir tikrinama, ar jau dalosi. Jeigu ne, tai liekana plečiama ir dalmenyje prirašomas 0. Jeigu dalosi, tai dalmenyje prirašomas 1 ir veiksmas vėl kartojamas.

Pradžioje gaunamas absoliučios abiejų operandų reikšmės. Jeigu reikia, tai neigiama reikšmė rezultatui priskiriama jau padalinus.

32 bitų sumatorius suformuojamas iš registro A (kairioji pusė) ir registro D (dešinioji pusė). Tam, kad efektyviai atlikti atimtį, daliklio absoliučios reikšmės papildomas kodas saugomas registre MBR. Jeigu atėmus iš dalomojo gavosi neigiamas skaičius, tam, kad atstatyti jo reikšmę (anuliuoti atimtį), daliklio absoliuti reikšmė saugoma registre C.

Jeigu pirmas operandas  $\geq 0$ , tai registre B saugoma antro operando reikšmė. Jeigu pirmas operandas  $< 0$ , tai registre B saugomas antro operando papildomas kodas. Rezultato ženklas bus toks pat, kaip ir reikšmės, saugomos registre B.

pamastymas kaip dar kitaip galetu buti, siaip sau

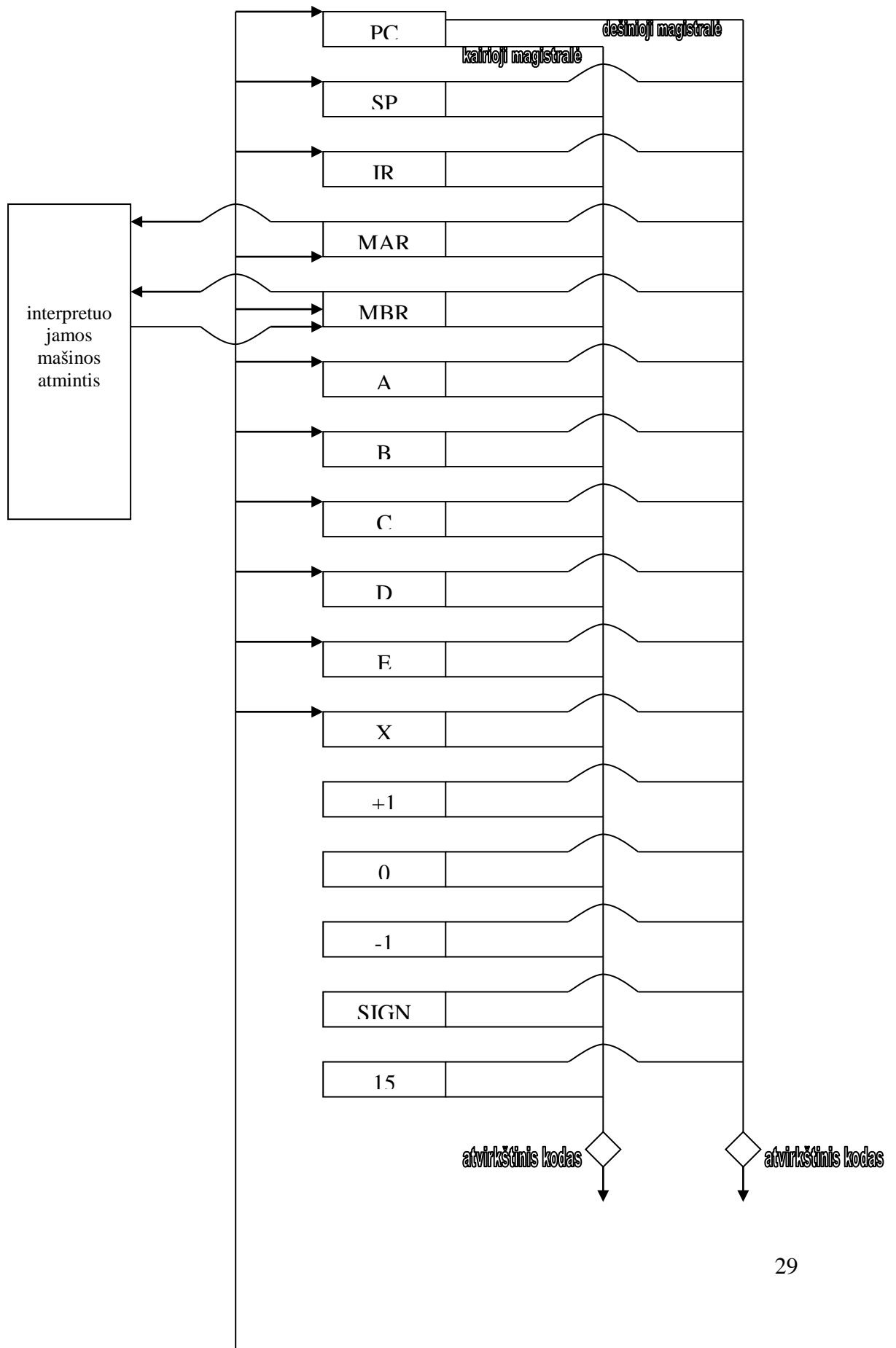
### ***Mikroprograminio lygio projektavimas***

Mikrokomandos, kurių kiekvienas bitas yra susietas su atskiru ventiliu, pasižymi struktūros ir realizacijos paprastumu. Tačiau kompiuterio ventilių skaičius gali siekti kelis šimtus. Mikrokomanda, užimanti kelis šimtus bitų, yra pernelyg ilga, užima per daug mikroatminties.

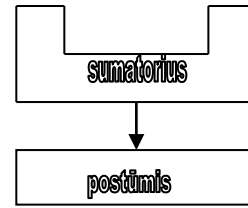
Pagal mikroprograminio lygio architektūrą, ne bet kokie ventiliai kuriuo tai laiko momentu gali būti atidaryti. Pavyzdžiui, į kairįjį sumatoriaus įėjimą gali būti paduotas tik vienas signalas iš dešimties galimų. Jiems koduoti pakanka 4 bitų. Į dešinįjį įėjimą gali būti paduotas tik vienas iš 7 galimų signalų. Jam koduoti pakanka 3 bitų. Praktikoje sumatoriaus išėjimas nukreipiamas tik vienu keliu. Todėl reikalingas tik vienas signalas iš 6 galimų. Jam koduoti pakanka 3 bitų. Todėl 23 ventilių darbas gali būti užkoduotas 10 bitų. Toks mikrokomandų formatas vadinamas koduojamu. Koduojamo formato trūkumas yra tas, kad kiekvienam koduojamam laukui reikalingas dešifratorius, kuris lauko reikšmę perversų į ventilio numerį. Koduojant prarandamas tam tikras lankstumas užduodant lygiagrečius veiksmus. Į koduojamą lauką paprastai įtraukiami vienu metu nesuderinami ventiliai.

Galimas sprendimas yra modifikuoti informacinius kanalus, įvedant tris bendras magistrales ir naują registrą E.

# KOMPIUTERIŲ ARCHITEKTŪRA

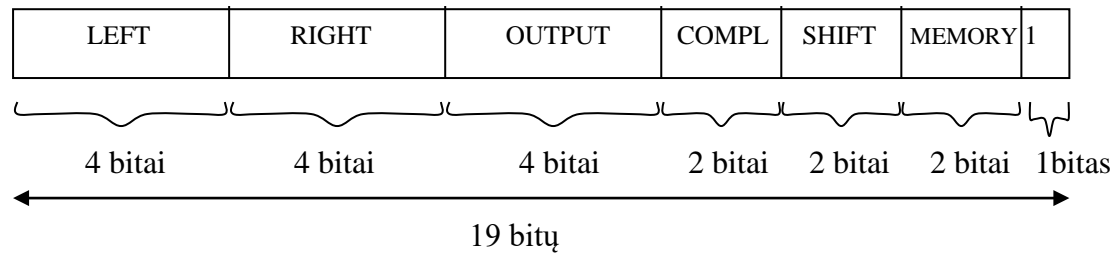


# KOMPIUTERIŲ ARCHITEKTŪRA

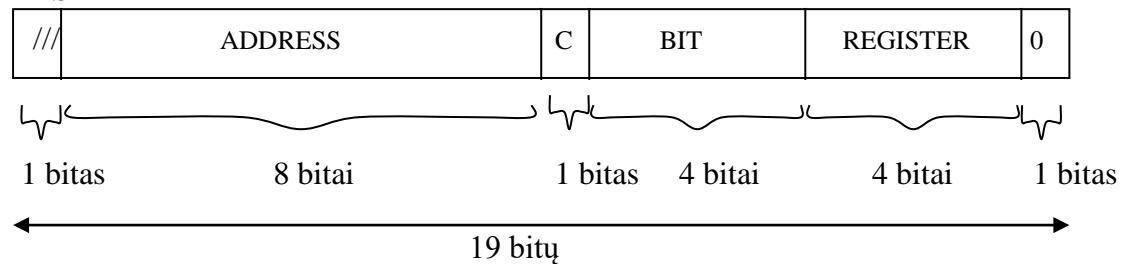


Tokios architektūros mikroprograminio lygio komandos gali būti tokios:

## GATE



## TEST



## REGISTER

0	=	( 0000 ) <sub>2</sub>	PC
1	=	( 0001 ) <sub>2</sub>	SP
2	=	( 0010 ) <sub>2</sub>	IR
3	=	( 0011 ) <sub>2</sub>	MAR
4	=	( 0100 ) <sub>2</sub>	MBR
5	=	( 0101 ) <sub>2</sub>	A
6	=	( 0110 ) <sub>2</sub>	B
7	=	( 0111 ) <sub>2</sub>	C
8	=	( 1000 ) <sub>2</sub>	D
9	=	( 1001 ) <sub>2</sub>	E
10	=	( 1010 ) <sub>2</sub>	X
11	=	( 1011 ) <sub>2</sub>	+1
12	=	( 1100 ) <sub>2</sub>	0
13	=	( 1101 ) <sub>2</sub>	-1
14	=	( 1110 ) <sub>2</sub>	SIGN
15	=	( 1111 ) <sub>2</sub>	15

## COMPL

0	=	( 0000 ) <sub>2</sub>	abiams įėjimams atvirkštinio kodo neskaičiuoti
1	=	( 0001 ) <sub>2</sub>	skaičiuoti atvirkštinį kodą kairiajam operandui
2	=	( 0010 ) <sub>2</sub>	skaičiuoti atvirkštinį kodą dešiniajam operandui
3	=	( 0011 ) <sub>2</sub>	abiams oprandams skaičiuoti atvirkštinį kodą.

## KOMPIUTERIŲ ARCHITEKTŪRA

### SHIFT

0	=	( 0000 ) <sub>2</sub>	išėjimo nestumti
1	=	( 0001 ) <sub>2</sub>	stumti į kairę per 1 bitą
2	=	( 0010 ) <sub>2</sub>	stumti į dešinę per 1 bitą
3	=	( 0011 ) <sub>2</sub>	nenaudojamas

### MEMORY

0	=	( 0000 ) <sub>2</sub>	nėra apsikeitimo su atmintimi
1	=	( 0001 ) <sub>2</sub>	skaityti iš atminties
2	=	( 0010 ) <sub>2</sub>	rašyti į atmintį
3	=	( 0011 ) <sub>2</sub>	nenaudojamas

BIT – bito registre nurodymas

### C

0	=	( 0000 ) <sub>2</sub>	perduoti valdymą, jei tikrinamas bitas yra 0
1	=	( 0001 ) <sub>2</sub>	perduoti valdymą, jei tikrinamas bitas yra 1

ADDRESS – mikrokomandos absoliutus adresas, kuriai perduodamas valdymas.

Architektūra, kurią nagrinėjome anksčiau, yra vadinama horizontalia. Joje kiekvienas bitas nurodo ventili, taip gaunama ilga mikrokomanda. Architektūra, kurią nagrinėjame dabar, vadinama vertikalia. Joje naudojami koduojami laukai ir taip gaunama trumpa mikrokomanda.

Horizontali struktūra užtikrina didesnę paralelizmą, yra efektyvesnė, reikia mažiau mikrokomandų, tačiau jos yra ilgesnės. Tačiau jeigu tokioje architektūroje yra N registų, tai tarp jų reikia  $2^N$  magistralių. Todėl jeigu registų yra daug (N yra didelis), kompiuteris yra nehorizontalios architektūros. Kita vertus, lygiagretus perdavimas vienu metu paprastai vykdomas tik trijose magistralėse, ir didelis registų skaičius čia neturi jokios įtakos.

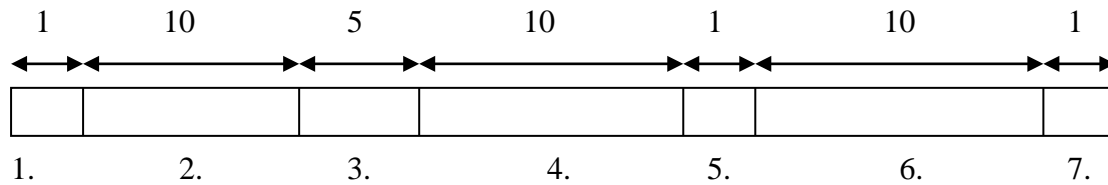
Jeigu mašina yra vertikalios architektūros, tai išskirtinės magistralės (intensyviai naudojamos) gali būti realizuotos horizontaliu principu.

### **Atminties ciklų trukmė**

Procesoriaus ciklas beveik pilnai apsprendžiamas mikroatminties ciklo dydžiu. Pagrindinės atminties ir mikroatminties ciklų dydžių santykis yra viena iš pagrindinių kompiuterio darbinių charakteristikų. Interpretuojamos mašinos atmintis yra lėtesnė už interpretuojančios mašinos atmintį. Vykdamas interpretuojamą komandą, į pagrindinę atmintį paprastai reikia kreiptis vieną ar du kartus, o į mikroatmintį – dešimtis ar net šimtus kartų. Dažnai praktikoje tas santykis yra 1, nes abiems loginėms atmintims naudojama ta pati fizinė atmintis.

Mikroatmintį yra stengiamasi padaryti greitesne, netgi naudojant pastovią atmintį, prieinamą tik skaitymui. Tegul atminties ciklų santykis yra 10 : 1. Tarkime, kad atliekama komanda PUSH:

## KOMPIUTERIŲ ARCHITEKTŪRA



1. Interpretuojamos komandos skaitymo inicijavimas.
2. Interpretuojamos komandos nuskaitymo laukas.
3. Komandos dekodavimas ir operando skaitymo inicijavimas.
4. Operando skaitymo laukimas.
5. Rašymo inicijavimas.
6. Rašymo pabaigos laukimas.
7. Perėjimas į pagrindinį ciklą.

Čia 79% laiko procesorius laukia ir yra nenaudojamas. Tam, kad sumažinti laiką, kada procesorius nenaudojamas, reikia arba greitinti pagrindinę atmintį, arba išlygiagretinti (suderinti laike) interpretuojamos mašinos komandų valdymą.

Pavyzdžiui, vykdant dalybos komandą, reikia maždaug 150 mikrokomandų ir per tą laiką galima iš anksto iš atminties nusiskaityti 10 sekančių interpretuojamų komandų. Nuskaitytas komandas reikia kur nors padėti saugojimui. Galima padėti į papildomus registrus, bet tai apsunkina architektūrą, nes reikia papildomų magistralių ir mikrokomandos tampa ilgesnės. Tačiau nuskaitytas komandas galima padėti ir į mikroatmintį. Vien tik pasiruošti komandas iš anksto nepakanka, nes tada pagrindinė laukimo priežastis bus duomenys.

Turint stekinę mašiną nesudėtinga iš steko viršūnės nuskaityti duomenis iš anksto, kad nebūtų laukimo dėl duomenų skaitymo.

Kitas sprendimas yra sudaryti tokias mikroprogramas, kad būtų galima užtikrinti lygiagretų komandų interpretavimą. Pradėti vykdyti sekančią komandą, nebaigus vykdyti ankstesnės, yra gana keblus uždavinys ir yra prasmingas tik tada, kai tokio uždavinio sprendimo greitis yra sulyginamas su skaitymo iš atminties greičiu. Pavyzdžiui, nėra prasminga lygiagretinti tokia mikroprogramą:

$$L = I + Y \cdot K$$

PUSH I;  
PUSH Y;  
PUSH K;  
MUL;  
ADD;  
POP L;

Išlygiagretinti reikia, turint sąlyginį perėjimą. Jo atveju atsiranda dvi šakos. Dar nebaigus skaičiuoti sąlygos reikšmės ir nežinant, kurią šaką iš tikrųjų reikės vykdyti, galima pradėti atlikti veiksmus, nurodytus kuria nors iš jų.

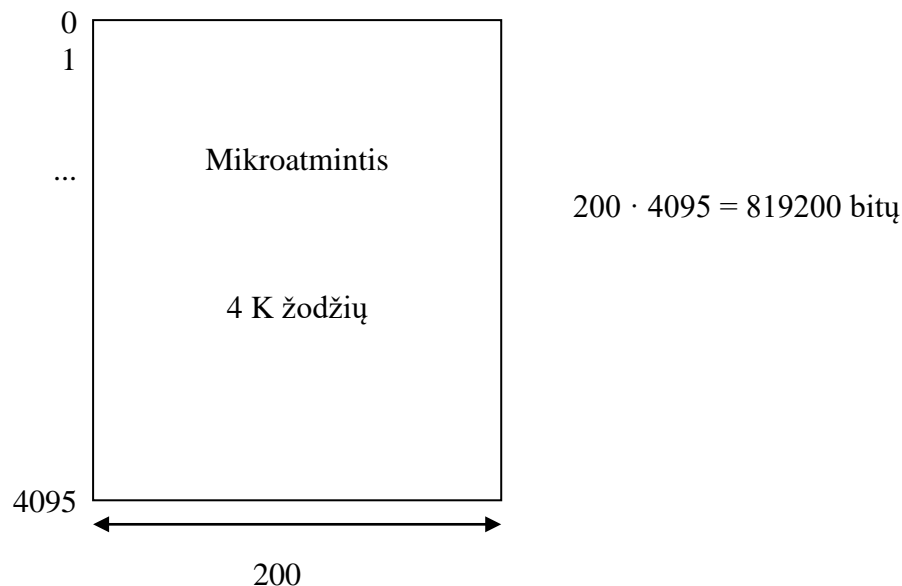
IF A · B · C · D · E < 1000 THEN CALL SUB;

Jeigu pasirinkta šaka nepasitvirtina, reikia organizuoti grįžimą, nes buvo įvykdyti nereikalingi priskyrimai. Statistiškai sąlyga paprastai nepatenkinama, o patenkinama tik vieną kartą, nes sąlyga paprastai įeina į ciklą.

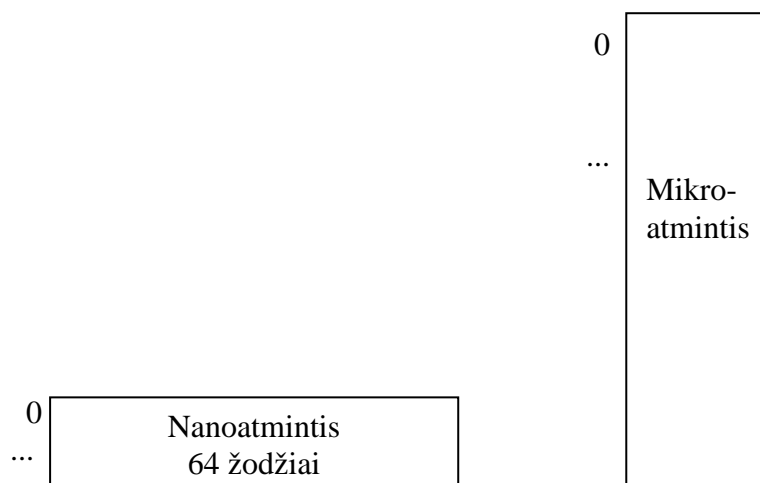


**Nanoatmintis**

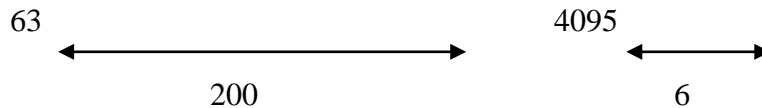
Horizontalioje architektūroje paprastai naudojamas nedidelis ventilių skirtingų kombinacijų kiekis, tačiau tos kombinacijos, kurios naudojamos, yra naudojamos daugelį kartų (pvz., skaitymas iš steko). Tos skirtingos kombinacijos užduodamos mikrokomandomis (jų ilgis gali būti keli šimtai bitų, o jų skaičius 64 – 256), kurios saugomos nanoatmintyje. Tuomet mikroprograma yra tokių nanoatminties mikrokomandų adresų seka.



Jeigu mikroprogramoje yra ne daugiau 64 skirtingų žodžių, tada architektūra galėtų būti tokia, kad mikroatmintyje yra adresai komandų, kurios tais adresais yra saugomos nanoatmintyje:



## KOMPIUTERIŲ ARCHITEKTŪRA



$$64 \cdot 200 + 4095 \cdot 6 = 12800 + 24576 = 37376 \text{ bitų}$$

Taip susidaro 5% buvusio atminties dydžio.

Čia išsaugomos horizontalios struktūros perdavimų lygiagretumas ir vertikalios struktūros atminties taupymas. Padidėja procesoriaus ciklas, nes jis susideda iš mikroatminties ir nanoatminties ciklų

Turint tikslą sumažinti skirtingų mikrokomandų skaičių, labai svarbu turėti valdymo perdavimą santykiniais adresais.

IF BIT(15, MBR) = 1 THEN SKIP n;

vietoj

IF BIT(15, MBR) = 1 THEN GOTO L;

Mikroprograminis lygis gali būti orientuotas konkretaus interpretuojamo lygio efektyviam interpretavimui arba gali būti universalus mikroprograminis lygis, skirtas įvairių architektūrų interpretavimui.

Pagal nusistovėjusias tradicijas kalbų transliatoriai (ALGOL, FORTRAN, PL/1, COBOL, PASCAL, C) perveda programą į tradicinį assemblerinį lygį, kuris vėliau interpretuojamas. Maža yra pagrindo manyti, kad tradicinis assembleris yra optimali tarpinė kalba tarp kompiliatoriaus ir interpretatoriaus. Kompiliatoriai naudoja apie 20% komandų sugeneruotame kode.

Mikroprogramavimo trūkumas yra tas, kad efektyvumas yra mažesnis apie 10 kartų, o norint gauti duoto galingumo procesorių, naudojant mikroprogramavimą, jis turi būti 10 kartų galingesnis, lyginant su aparatūrine komandų sistemos realizacija.

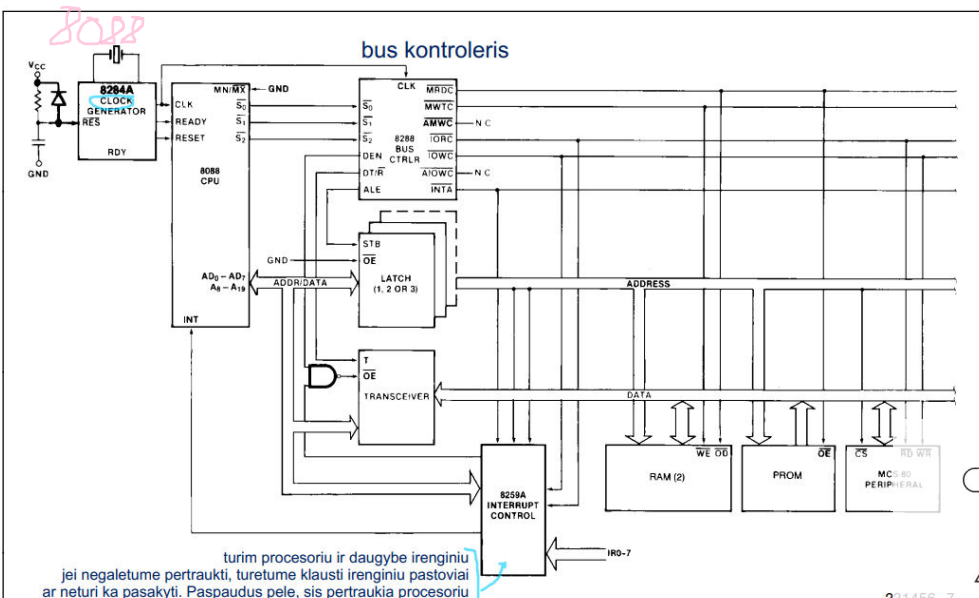


Figure 7. Fully Buffered System Using Bus Controller

Intel daro daug papildomo darbo  
ARM nedaro daug papildomo darbo (spėjimo), todėl jie energiška taupesni

prefetch instrukcijos is atminties traukiamos kitu instrukciju metu. pipelining - kol viena instrukcija vykdom, kita dekoduojame, kita traukiame. Tiesiog proces. vyksta greiciau

sito is elektrines pusės nereikės žinoti

visi registrai pajungti

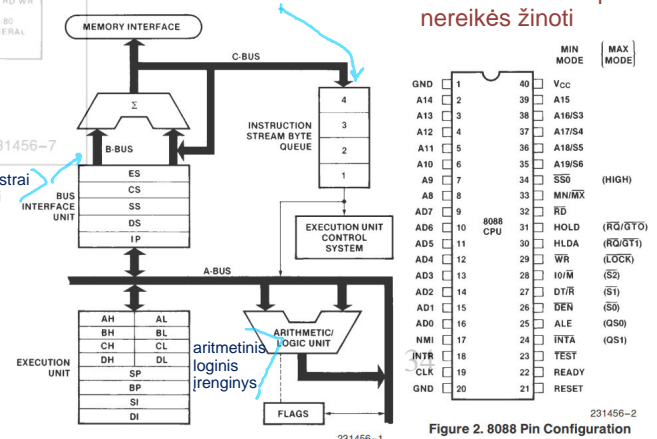


Figure 1. 8088 CPU Functional Block Diagram

Figure 2. 8088 Pin Configuration

## 4. Mikroprocesoriaus Intel 8088 architektūra

palyginsim su mūsų sugalvotu...

### Mikroprocesoriaus architektūros samprata

per egz. nereikes  
rašyti kaip susijungia,  
susideda, reikės  
spresti tik užduotis

Mikroprocesoriaus architektūrą sudaro tos jo savybės, kurios naudojamos programavime. Mikroprocesoriaus Intel 8088 architektūrą sudaro aparatūros ir mikroprogramų visuma. Pakeitus mikroprogramų rinkinį, gautume nebe mikroprocesorių Intel 8088, o kažką kito. Mikroprocesoriaus Intel 8088 architektūra realizuojama viena mikroschema su 40 išvadų. Mikroprocesoriaus Intel 8088 architektūra apibrėžiama jo vidinėmis savybėmis ir sąveika su išorinėmis mikroprocesoriaus atžvilgiu komponentėmis.

mūsų sugalvotoje  
yra ventiliai,  
sumatoriai, bet yra  
ir mikroprograma

dabartiniu  
procesoriu  
mikroprograma yra  
su perrašymu,  
galima perrašyti. Su  
8088 mikroprograma  
yra įrašyta kietai

Tokiomis išorinėmis komponentėmis yra:

1. Operatyvi atmintis. galima išimti, įdėti...
2. Portų (įvedimo-išvedimo) sistema.
3. Adreso, duomenų ir valdymo magistralės. trečias apima tuos du.

kojųčių

pvz tinklo plokštė, jai skiriama tam tikri adresai operatyvioje  
atmintyje, galima rašyti ir į tam tikrą magistralę, sujungus  
procesorių su plokšte tiesiogiai

### Operatyvios atminties samprata

Operatyviosios atminties sąvoka nusakoma per priklausomybę nuo procesorių, o ne savarankiškai. Operatyvią atmintį sudaro fizinė tiesioginio priėjimo tiesinė adresų erdvė. Ji yra tiesiogiai prieinama iš procesoriaus per adresų magistralę. Kadangi mikroprocesoriaus Intel 8088 adresų magistralės plotis yra 20 bitų, tai maksimali operatyvi atmintis yra  $2^{20} = 1 \text{ M}$ . Tokiu būdu operatyvios atminties dydis priklauso ne tik nuo procesoriaus, bet ir nuo adresų magistralės pločio. Pavyzdžiui, mikroprocesorius Intel 80286 sudaro absoliutų fizinį adresą 24 bituose. Todėl šis procesorius galėtų adresuoti fizinę atminties erdvę  $2^{24} = 16 \text{ M}$ . Tačiau realiai kompiuteryje su procesorium Intel 80286 yra realizuota 20 bitų pločio adresų magistralė, todėl tokie kompiuteriai turi 1 M operatyvios atminties.

kiek procesorius turi  
adresų, tiek atminties  
jūs ir matote

20 kojų paskirta prieiti  
prie atminties

20 kojų

naujesnis procesorius

buvo kazkokie kompai,  
kad vistiek tegalejo  
pasiekti tik 1 mb, nors  
procesorius galejo  
daugiau

Kalbant apie personalinius IBM kompiuterius, yra sakoma 640 K, 1 M, 2 M operatyvios atminties. Dažniausiai tai yra nekorektiškas pasakymas, nes čia kalbama ne apie vartotojui prieinamos operatyvios atminties dydį. Nekorektiškumas yra dvejopas. Jeigu sakoma, kad yra 640 K operatyvios atminties, tai reiškia, kad iš tiesų yra 1 M tiesiogiai adresuojama atminties erdvė. Jeigu yra sakoma, kad yra 1 M operatyvios atminties, tai reiškia, kad vartotojui prieinama yra 640 K, 384 K atminties yra tik fiziškai realizuota kaip operatyvi atmintis mikroschemomis, tačiau architektūriškai tai nėra operatyvi atmintis. Informacija joje prieinama apsikeitimo, kaip su išoriniu įrenginiu, būdu, per tiesioginio priėjimo prie atminties kontrolerį, tiesioginės atminties draiverį.

10 kartų daugiau  
atminties už kitus, tuo  
metu tebuvo tik 64K.  
Pasirupino, kad jis  
nepasentu taip greitai...

pasiskaityti kaip veikia

<https://www.minuszerodegrees.net/5150/misc/5150%20-%20Memory%20Map%20of%20the%20640%20KB%20to%201%20MB%20Area.jpg>  
[https://www.ardent-tool.com/5150/PC\\_origins.html](https://www.ardent-tool.com/5150/PC_origins.html)

Egz. nereikės! → Personaliniuose kompiuteriuose IBM kontroleris – įvedimo-išvedimo specializuotas procesorius – yra vadinamas adapteriu.

Draiveris tai yra programa, modeliuojanti virtualų išorinį įrenginį, turintį programuotojui patogią struktūrą. Tai yra abstrakti sąvoka, leidžianti nusakyti

šalia proc. yra  
papildomų dalykų,  
įvedimų ir išvedimų  
adapteris yra Intel 8288  
chip. Jis atsakingas už  
magistralių valdymą.

tarkim tinklo ploskte, reikia atsiusti issiusti paketa. jei junginetume CPU pinus butu sunku, todėl draiveris duoda abstraktu interface, palengviną darbą.

## KOMPIUTERIŲ ARCHITEKTŪRA

bendravimą su išoriniu įrenginiu programuotojui patogesniais terminais, palyginus su tais, kurie nusako, kokias komandas gali įvykdyti išorinio įrenginio adapteris. Draiveris yra ta programa, kuri išverčia, išreiškia programuotojui patogesnę komandų kalbą į adapterio komandų kalbą.

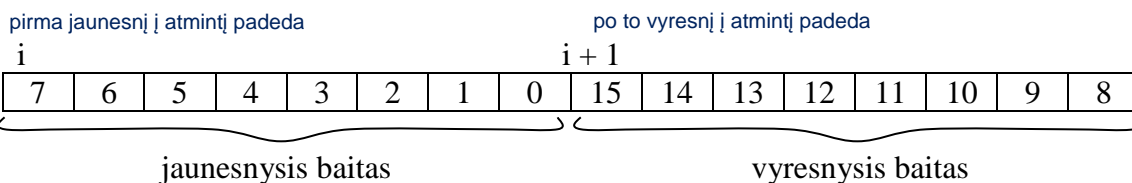
Tokiu būdu kompiuteris su procesoriumi Intel 80286 ir 24 bitų adresų magistrale gali turėti 16 M operatyvią atmintį, jeigu yra pakankamas pačių operatyvios atminties mikroschemų kiekis.

Operatyvioje atmintyje mažiausia adresuojama dalis yra 8 bitų laukas – baitas. Tokiu būdu operatyvios atminties adresinė erdvė yra baitų adresinė erdvė nuo 0 iki  $2^N - 1$ , kur mūsų atveju  $N = 20$ . Yra priimta baito viduje bitus numeruoti pagal pozicinių skaičiavimo sistemų pozicijų, t.y., dvejetų laipsnius.

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

Šalia informacijos grupavimo baitais, mašiniame lygyje informacija gali būti grupuojama žodžiais: bet kurie greta esantys (gretimus adresus turintys baitai) sudaro žodį. Operatyvios atminties žodžio viduje yra priimtas toks atitikimas tarp bitų ir žodyje saugomos dvejetainės reikšmės dvejetainių laipsnių:

žodis susideda iš jaunesniojo ir vyresniojo baito



šita svarbu žinoti

Mikroprocesoriaus Intel 8088 su operatyvia atmintimi susijęs duomenų magistrale, kurios plotis yra 8 bitai, t.y., vienu skaitymo iš operatyvios atminties veiksmu arba vienu rašymo į operatyvią atmintį veiksmu yra perduodamas vienas baitas. Procesorių Intel 8086 ir Intel 80286 duomenų magistralės plotis yra 16 bitų, todėl pastarosios architektūros procesorių greitis yra didesnis, nes esant vienodam dažniui, vienu taktu yra perduodamas didesnis informacijos kiekis.

žinoti kaip hardware realizuota reikia minimaliai egz

Fiziškai procesoriuje adresų ir duomenų išvadai yra tie patys, t.y., adresų ir duomenų magistralė yra ta pati. Tokiu atveju magistralė vadinama multipleksinė. Valdančiosios magistralės vienas bitas nurodo skaitymo arba rašymo į operatyvią atmintį veiksmą.

ant tu pačių pinų procesoriaus sudedame adresavimą ir reikšmes

### **Portų sistema**

irenginius pasiekiamo per portų sistema, nes jei visus irenginius sukrautume ant RAM 1mb tai is to RAM nekas bepaliktu

Panašiai kaip ir operatyvioji atmintis, kita fizinė tiesinė adresų erdvė sudaro įvedimo-išvedimo portų sistemą. Procesorius su portų sistema yra susietas adreso magistrale, duomenų magistrale ir valdančiąja magistrale. Portų adreso magistralė yra 16 bitų pločio, todėl portų adresinė erdvė yra  $2^{16} = 64$  K. Procesoriaus ir portų duomenų magistralė yra 8 bitų pločio.

ne 20 kaip su atmintimi bet 16

S2	S1	S0	Characteristics
0 (LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	Halt
1 (HIGH)	0	0	Instruction Fetch
1	0	1	Read Data from Memory
1	1	0	Write Data to Memory
1	1	1	Passive (No Bus Cycle)

Įvedimo-išvedimo portas tai yra turinti adresą 8 bitų magistralė, jungianti portų sistemą su išorinio įrenginio adapteriu.

yra dar vienas chip

## KOMPIUTERIŲ ARCHITEKTŪRA

Apsikeitimui informacija tarp procesoriaus ir portų sistemos yra naudojamos dviejų tipų komandos: IN ir OUT. Šių komandų vykdymo metu procesorius nuskaityto arba perduoda baitą. Portų sistemos baito-porto adresas gali būti nurodytas komandos IN arba OUT operacijos kode arba operande.

Portai su numeriais 0 – 255 yra nurodomi komandos operacijos kode. Tokiu būdu faktiškai yra 255 skirtingos komandos IN bei 255 skirtingos komandos OUT ir dar po vieną komandą darbui su portais, turinčiais adresus, didesnius už 255.

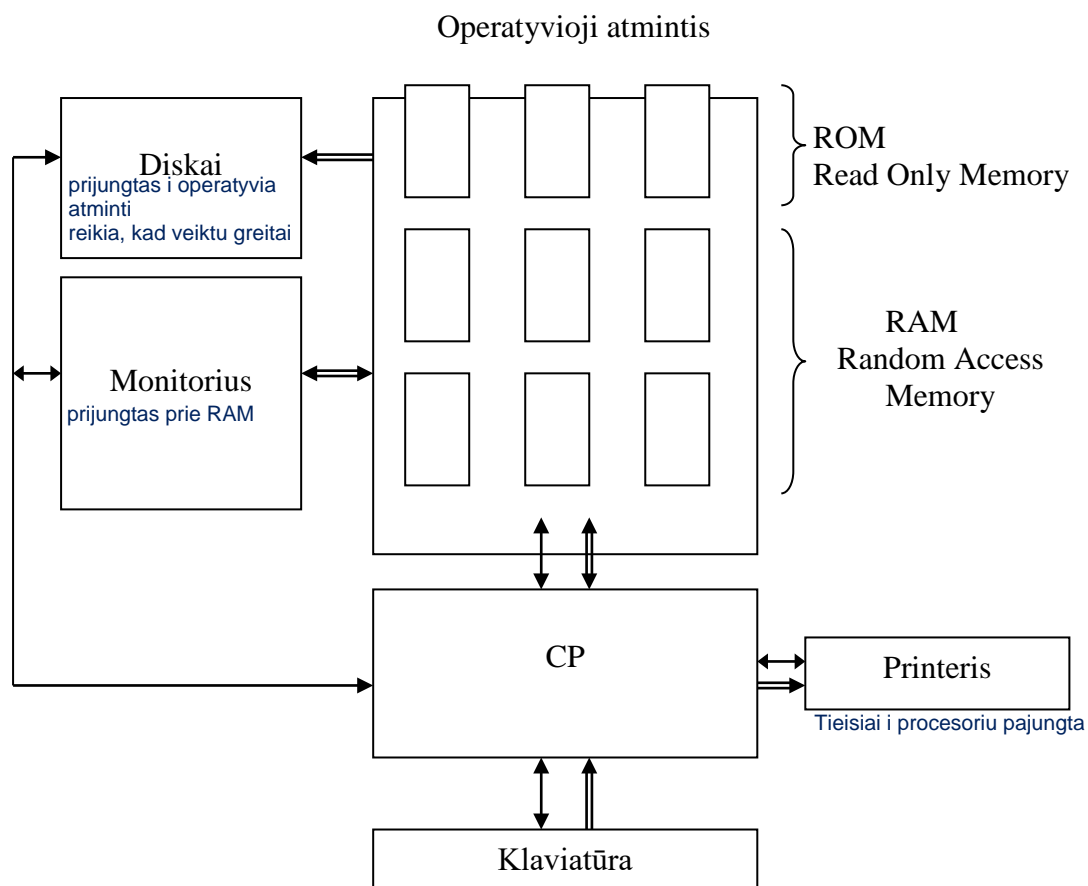
Valdančiosios magistralės vienas bitas parodo skaitymo arba rašymo į portą veiksmą.

Kaip ir operatyviojoje atmintyje, portų sistemoje gretimi du baitai gali būti charakterizuojami kaip 16 bitų portas ir apsikeitimo operacijoje viena komanda per du taktus perduodami du baitai. Tokiu būdu portų sistema turi 64 K 8 bitų portus arba 32 K 16 bitų portus.

Įvedimo-išvedimo portų adresai gali būti sutapatinti su operatyviosios atminties adresais, t.y., adapterio magistralės pajungiant ne prie portų sistemos, o prie operatyviosios atminties. Tuomet įvedimo-išvedimo komandomis tampa bet kuri komanda, skaitanti arba rašanti į atmintį. Darbe su portais (įvedimu-išvedimu) atsiranda lankstumas, tačiau prarandama dalis operatyviosios atminties. Pavyzdžiui, monitoriaus adapterio portai, atitinkantys ekraną, yra atvaizduojami į operatyviąją atmintį.

### Kompiuterio sandara

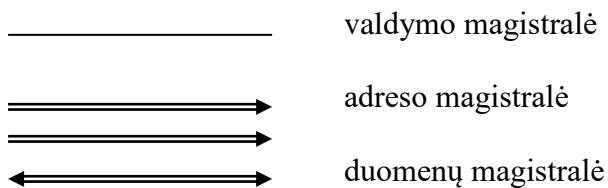
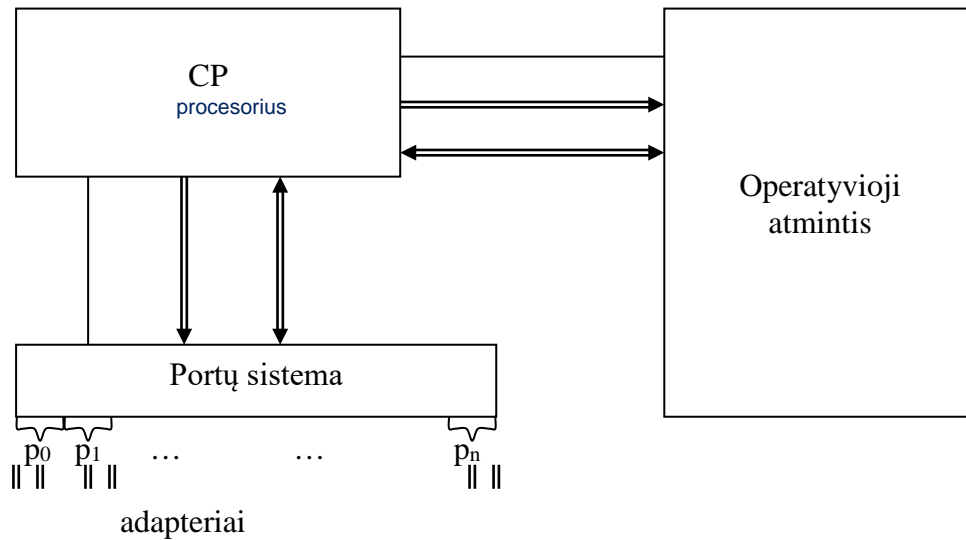
Greitaeigiai įrenginiai turi tiesioginį ryšį su operatyviąja atmintimi.



kaip musu  
architekturoje yra  
255 operacijos ir po  
to plečiame iki 64K

## KOMPIUTERIŲ ARCHITEKTŪRA

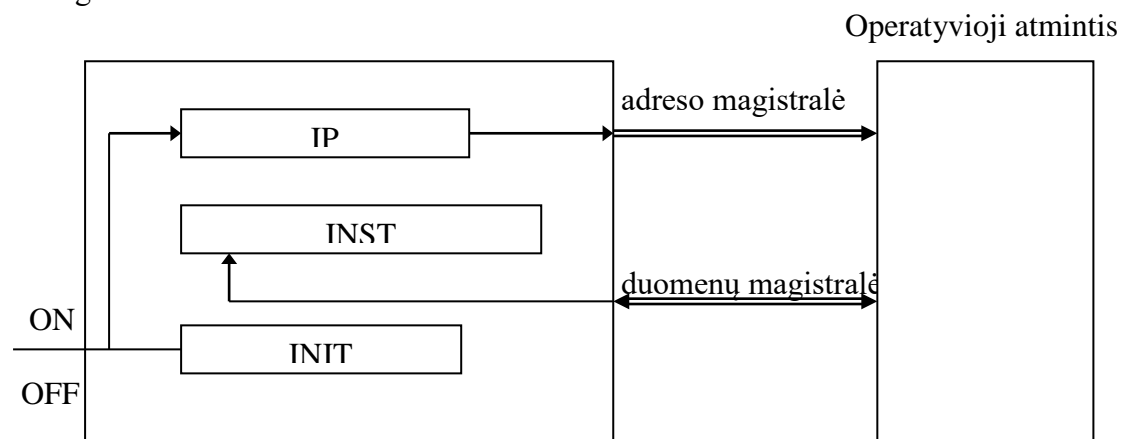
### Procesoriaus sąveikos su išorinėmis komponentėmis schema



Stambiausiu planu mikroprocesorius Intel 8088 susideda iš dviejų komponentų: operacinio įrenginio ir interfeisinio įrenginio. Operacinis įrenginys atlieka komandose nurodytus informacijos pertvarkymo veiksmus procesoriaus viduje, o interfeisinis įrenginys pagal operacinio įrenginio poreikius atlieka apsikeitimą su operatyviaja atmintimi ir portų sistema.

operacinis įrenginys, dedam į registrus, dauginam, sumuojame. i atminti deda dalykus

Panagrinėkime detaliau:



## **Mikroprocesoriaus Intel 8088 sandara**

Mikroprocesorius Intel 8088 susideda iš:

1. Registrų;
2. Vykdomojo adreso formavimo aparato;
3. Absoliutaus adreso formavimo aparato;
4. Komandų sistemos interpretatoriaus;
5. Steko;
6. Pertraukimų sistemos.

nustatytas skaičius  
kazkoks. Pasiimame  
adresa is BIOS

Registras yra įrenginys, susidedantis iš rinkinio dvejetainių skirsnų, kurie sugeba keisti ir išsaugoti vieną iš dviejų būsenų.

nuo čia prasidės tai, ką  
turėsime mokintis ir  
atsiskaityti

Registras IP (Instruction Pointer) saugo sekančios komandos, skaitomos iš atminties vykdymui, adresą. Įjungus kompiuterį, pirmosios vykdomos komandos pradinis adresas INIT užrašomas į registrą IP, kurio reikšmė paduodama į adreso magistralę, o valdančiojoje magistralėje nustatoma 1, informacijos iš atminties skaitymui. Nuskaitytos pirmosios komandos pirmas baitas patalpinamas į registrą INST, kuriame vykdomas komandos dešifravimas. Komandos gali būti įvairaus ilgio, pvz., kelių baitų ilgio. Pagal nuskaitytą pirmąjį komandos baitą nustatoma, ar jau nuskaityta visa komanda, t.y., ar komanda yra iš vieno baito ar toliau reikia skaityti komandą iš atminties po baitą į registrą INST.

komandos desifravimas,  
mūsų kompe medis,  
sitame procesoriuje  
kazkas panasiai vyksta

INST - instrukcija

Taigi komandų vykdymo cikle yra du etapai: komandos nuskaitymas ir komandos įvykdymas. Komandų nuskaitymą atlieka interfeisinis procesoriaus įrenginys, o komandų vykdymą atlieka operacinis įrenginys. Operacinis įrenginys savo ruožtu gali kreiptis į interfeisinį įrenginį operandų persiuntimo atlikimui. Interfeisinis įrenginys ir operacinis įrenginys yra du skirtingi įrenginiai, kurie dirba lygiagrečiai vienu metu ir savo darbą sinchronizuoja per buferinius nuskaitytų komandų registrus arba per valdymo perdavimo komandas.

interface įrenginys

operacinis įrenginys tai atlieka

Kai operacinis įrenginys užimtas operacijų vykdymu, interfeisinis įrenginys preliminarai skaito iš atminties sekančias komandas, kurios talpinamos buferiniame registre INST, kuris veikia eilės FIFO (First In First Out) principu. Atsilaisvinus baitui buferyje INST, iš atminties nuskaitytas sekantis baitas, taip pat yra atitinkamai modifikuojama registre IP esanti reikšmė. Taigi registre IP esanti reikšmė reiškia ne šiuo momentu vykdomos komandos adresą, bet šiuo momentu nuskaitytos iš atminties komandos baito adresą. Paprastai buferyje yra bent vienas baitas ir operaciniam įrenginiui nereikia laukti komandos iš atminties nuskaitymo. Tik jeigu operacinis įrenginys vykdo valdymo perdavimo komandą, tuomet interfeisinis įrenginys atšaukia nuskaitytųjų komandų eilę buferyje, pakoreguoja registro IP reikšmę ir formuoja naują komandų eilę pagal valdymo perdavimo adresą. Jeigu operacinis įrenginys pareikalauja ryšio su operatyviąja atmintimi arba su portų sistema, tai interfeisinis įrenginys pertraukia komandų eilės formavimą ir atlieka veiksmus su operandu.

jei įvyksta jump, iraso  
nauja IP

Operacinis įrenginys naudodamas 16 bitų adresinius registrus, operuoja vieno segmento adresine erdve, kurios dydis yra  $2^{16} = 64$  K. Operacinio įrenginio

fiziskai turim 20 bitu,  
1mb, bet viduje  
procesoriaus turim 16  
bitu registrus, darom  
su segmentais. Jei  
koks įrenginys pradeda  
rasyti daugiau nei 64k,  
tai jis cikliniu būdu  
pradeda ant virusaus  
uzrasyti



## KOMPIUTERIŲ ARCHITEKTŪRA

neturim 16 bitu  
registru, tik 16 bitu  
todėl visi sie  
idomus  
skaiciavimai  
daromi  
todėl yra atskiri  
segmentu adresai  
ir kiekviename  
segmente atskiri  
adresai

suformuotas adresas vadinamas vykdomuoju adresu ir gali būti žymimas trumpiniu EA (Effective Address). Visa adresinė erdvė yra  $2^{20} = 1 \text{ M}$  ir ji yra adresuojama interfeisiniu įrenginiu, kuris suformuoja absoliutų adresą pagal operacinio įrenginio efektyvų adresą ir segmento registrą, kuris saugo segmento pradžios paragrafo numerį. Operatyviosios atminties adresinė erdvė yra suskirstyta paragrafais po 16 baitų, t.y., kiekvienas adresas, kuris yra 16 kartotinis, yra paragrafo pradžios adresas. Tokiu būdu iš viso paragrafų yra  $2^{20} : 2^4 = 2^{16}$ . Todėl paragrafo numeris telpa 16 bitų segmento registre. Absoliutus adresas yra suskaičiuojamas taip:

$$\text{paragrafo numeris} \cdot 2^4 + EA$$

Tai reiškia, kad segmento registre esanti reikšmė pastumiami į kairę per keturias pozicijas ir sudedama su efektyviu adresu.

dosBox  
musu duomenų DS  
prasideda nuo 0DC0  
kitas segmentas gali  
prasidėti po 10HEX

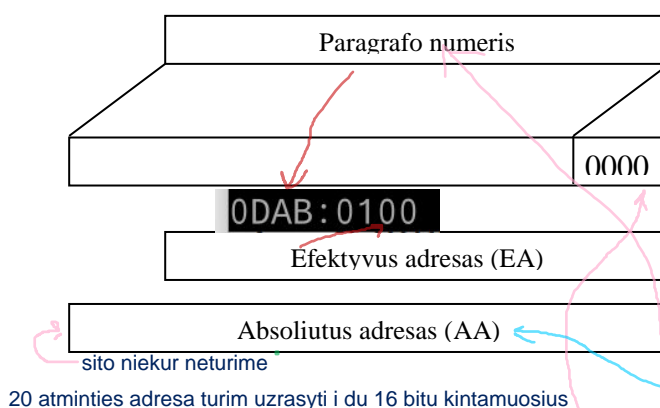
segmentams galiu  
priskirti adresus kas  
0x10 arba 16 skaičiu  
desimtaineje

svarbiausias dalykas!!!

isiminti, mokėti  
skaiciuoti

knyga turim 500 psl, mums  
reikia 413 eilutes  
mes verčiame iki 400 psl,  
susirandame 13 eilute - tai bus  
absoliutūs adresas

- > atmintis yra puslapiuojama
- > mažiausiai galima adresuoti viena baita (8 bitus)
- > adresą į 4 bitus negalima turėti



is 16 bitu susideda, skirtas adresuoti  
vienam segmentui  
 $2^{20} = 1 \text{ MB}$   
 $2^{16} = 64 \text{ kB}$   
20 bitu susideda  
norint gauti absoliutu adresa, mes  
pasimame paragrafo numeri, jį  
pashiftiname per 4 į kairę ir dar  
pridedame 4 papildomus adreso bitus

Mikroprocesorius Intel 8088 turi tris grupes registrų:

1. Duomenų registrai; ax, bx, cx, dx
2. Adresiniai registrai; sp, dx, si, di
3. Segmento registrai. ds, es, ss, cs

pasimame paragrafo  
numeri, paslenkame į  
kaire per keturis bitus ir  
pridedame adresa.  
gaunasi paragrafas ir  
poslinkis kas tame  
adrese parašyta

Kaip operatyviosios atminties erdvė gali būti nurodoma baitais arba žodžiais, taip ir duomenų registrus galima traktuoti kaip keturis šešiolikos bitų registrus arba aštuonis aštuonių bitų registrus.

	7	0 7	0
000 = AX	AH = 100	AL = 000	
011 = BX	BH = 111	BL = 011	
001 = CX	CH = 101	CL = 001	
010 = DX	DH = 110	DL = 010	
	15		0

- > procesoriuje registra turim  
pazymėti dvejetainiu kodu
- > su push tik šitie veikia

vienos komandos veikia su vienais registrais, kitos su kitais, dėl riboto poslinkio

Pagrindinė registrų paskirtis yra tokia:

- AX – akumulatorius, sumatorius;
- BX – bazinis registras;
- CX – ciklo skaitliukas;
- DX – duomenų registras.

Konkreiti registrų paskirtis yra apibrėžiama komandose.



## KOMPIUTERIŲ ARCHITEKTŪRA

### Adresiniai registrai

Adresiniai registrai tai yra indeksiniai registrai ir nuorodų registrai. Adresiniai registrai yra 16 bitų.

Indeksiniai registrai yra SI (Source Index) ir DI (Destination Index).

SI = 110

15 šitie registrai turi savo 0  
numerukus dvejetainiu kodu

adresas - tai poslinkis kažkos  
0daB:0100

DI = 111

15 0

IP - tik instrukcijom pointeris  
jis nebetelpa i tris bitus

todėl jo panaudoti nelabai išeina

Šie registrai paprastai naudojami persiunčiamų eilučių simboliams indeksuoti.

Steko nuorodos registras SP (Stack Pointer)

SP = 100

Registras SP naudojamas steko viršūnei indeksuoti.

15 0

Bazės steko nuorodos registras BP (Base Pointer)

BP = 101

15 0

Registras BP paprastai naudojamas operandų adresavimui ne steko viršūnėje, o steko viduje.

Adresiniai registrai yra naudojami vykdomojo adreso formavimui: efektyviam adresui, kuris adresuoja virtualią adresinę erdvę viename segmente –  $2^{16} = 64$  K.

aiškią vietą nusistatyti

Vykdomasis adresas bendru atveju gali būti suformuotas pagal tokį principą:

**bazinis registras + indeksinis registras + poslinkis**

visaim tuos adresus galima metyti ir pasidaryti taip kad no kažkurio skaičiuojame

arba atskirus šio principo atvejus. Baziniai registrai yra BX ir BP, indeksiniai registrai yra SI ir DI, o poslinkis yra vieno arba dviejų baitų adresinė reikšmė, nurodyta pačioje komandoje. Todėl galimi adresavimo variantai yra šie:

BX + SI + poslinkis

BX + DI + poslinkis

BP + SI + poslinkis

BP + DI + poslinkis

} adresavimas su bazavimu ir indeksavimu

SI + poslinkis

DI + poslinkis

} adresavimas tik su indeksavimu

BP + poslinkis

BX + poslinkis

} adresavimas tik su bazavimu

išvardyti visi atvejai kaip tas gali būti padarytas

duomenų segmente turime eilutę, kurioje užkoduota žinutė, norime nukopijuoti i ES

nustatom kad "Labas pasauli" pradžia yra ES, sukame skaitliuką su cx registru ir kopijuojame kiekvieną simbolį, susigeneruojame poslinkį

gali būti ir bx su poslinkiu, bet tik bx

pvz.: [DI + bx]

bendrai: viskas užkoduota ribotu bitu skaičiumi

### Komandų struktūra

procesorius skaito baitą po baito. Nusiskaito vieną baitą, užkrauna į tą eilę iš eilės kitus baitus. Pagal tą baitą žiūri kas bus po to. Nusiskaitome komandą

Mikroprocesoriaus Intel 8088 operandai gali būti trijų tipų:

1. Betarpiški operandai, kurie yra komandos sudėtinė dalis;
2. Operandai, kurie yra registruose;
3. Operandai, kurie yra operatyviojoje atmintyje.

mov ax, [bp+si]

operatorius

Betarpiškas operandas gali būti 8 arba 16 bitų ir jis yra ne adresas, bet reikšmė, kuri dalyvauja komandos veiksmė. Komandos operandai skirstomi į duomenų ir rezultatų operandus. Betarpiškas operandas gali būti tik duomenų operandu. Betarpiški operandai yra efektyvūs komandų atlikimo prasme.

negalima modinti reikšmės 777 kita reikšme

duomen  
mov ax, 777  
rezult.

## KOMPIUTERIŲ ARCHITEKTŪRA

777 movinti į ax  
bet taip pat ax į atmintį

su registrais darbas greitas, jie optimizuoti

Registruose esantys operandai gali būti duomenys ir rezultatai. Jie taip pat pasižymi efektyvumu. Registras yra komandoje nurodomas registro numeriu. Jeigu registras yra identifikuojamas komandos operacijos kode, tai jis nėra traktuojamas kaip operandas ir sakoma, kad komanda yra be operandų. komanda `push ax` - sakoma kad komanda yra be operandų, susitarimo reikalas

Apskaičiuojant efektyvų adresą yra naudojamas į komandos sudėtį įeinantis adresavimo baitas. Komandoje gali būti tik vienas operandas operatyviojoje atmintyje, t.y., negali būti abu operandai atmintyje. adresavimo baitą atskirai nagrinėsime.

negalima rašyti iš vienos atminties vietos į kitą vietą  
iš ram reikšmę pridėti iš kito ram vietos reikšmę vienu metu negalima. reikia bent vieną užsikrauti į ram

Eilutinės komandos tai yra komandos be operandų, t.y., beadresinės komandos, nors jos užduoda baitų persiuntimą tarp dviejų atminties sričių. SI, DI išskyrus

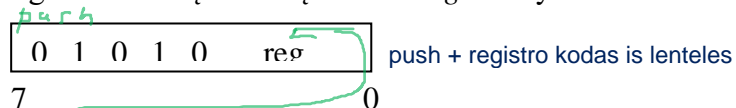
Yra naudojami terminai: operandas rezultatas (pirmas operandas) ir operandas duomuo (antras operandas). `mov ax, bx`  
į ax bus išsiųsta, susitarimas

Operandai yra interpretuojami arba pagal operacijos kodą, arba pagal adresavimo baitą. Pavyzdžiui, betarpiškas operandas nustatomas ne pagal adresavimo baitą, bet pagal operacijos kodą. B87707 `mov ax, 777` mov neturi savo, užrašomas kartu su ax

Pati paprasčiausia yra beadresinė komanda, ją sudaro vien tik operacijos kodas (sutrumpinimas – OPK):



Nėra visiškai aišku, ką vadinti operacijos kodu. Pavyzdžiui, nagrinėkime komandą, kuria patalpiname į steką registre esančią reikšmę: `PUSH reg`. Tai yra:



Kyla neaiškumas, ar tokiu atveju traktuoti, kad `PUSH` operacijos kodas yra 01010, o `reg` yra operandas. Ar, tarkime, 01010000 : `PUSH AX` yra komanda be operandų? Tegul operacijos kodas bus tokia bitų seka, kuri yra bendra homogeniškam veiksmui nurodyti su tikslumu iki operando. 01010 - operacijos kodas, reg - operandas

Tokiu atveju operacijos kodas gali būti koduojamas:

1. Baito dalimi; 0 1 0 1 0 reg perskaitę pirmą baitą jau turime sužinoti kas bus toliau, turime atskirti ar reikia skaityti toliau
2. Pilnu baitu; `mov baito`
3. Vienu baitu ir sekančio baito dalimi;
4. Dviem baitais.

OPK - operacijos kodas

Komandos gali būti:

1. Beadresinės;
2. OPK `reg`;
3. OPK `atmintis`;
4. OPK `betarpiškas operandas`, pvz.: `RETURN poslinkis`;
5. OPK `reg reg`;
6. OPK `reg atmintis`;
7. OPK `reg betarpiškas operandas`; `mov ax, 777`
8. OPK `atmintis betarpiškas operandas`.

## KOMPIUTERIŲ ARCHITEKTŪRA

3 variantai kur operandai gali būti  
Galimi atvejai:

pagauti iš iškalimo  
pusės

1. Operandas komandoje;
2. Operandas registre (registro numeris nurodytas komandoje);
3. Operandas atmintyje (efektyvaus adreso formavimas nurodytas komandoje).

Operando atmintyje adresas gali būti nurodytas pačioje komandoje. Tuomet turime tiesioginį atminties adresavimo būdą. Tiesioginė adresacija gali būti vienais atvejais identifikuojama pagal operacijos kodą, kitais atvejais – pagal adresavimo baitą.

Jeigu tiesioginė adresacija identifikuojama pagal operacijos kodą, tai galimi tokie variantai:

1. Tiesioginė adresacija

OPK	adr.j.b	adr.v.b
-----	---------	---------

adr.j.b – adreso jaunesnysis baitas;

adr.v.b – adreso vyresnysis baitas.

2. Santykinė adresacija

operacijos kodas ir adresas vadinasi santykinė adresacija, šokam į vieną ar kitą pusę

OPK	adresas
-----	---------

2 baitus  
sunaudojame

3. Absoliuti adresacija du registrai

OPK	adr.j.b	adr.v.b	segm.nr.j.b.	segm.nr.v.b.
-----	---------	---------	--------------	--------------

5 baitus  
sunaudojame

tokiame intervale  
galima persukti

dėl to darome .model small,  
nes efektyviau

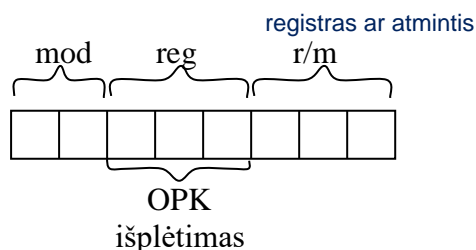
Esant tiesioginei adresacijai komandoje nurodytas adresas yra efektyvus adresas.

Jeigu yra santykinė adresacija, tai adresas traktuojamas kaip skaičius su ženklu, kuris kinta intervale  $[-128; 127]$ . Tada  $EA = IP + \text{adresas}$ .

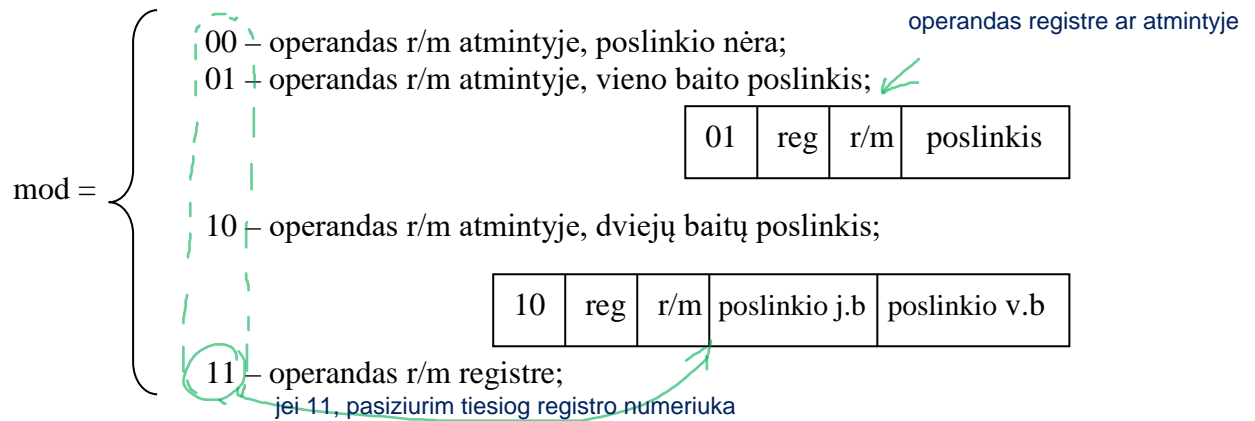
Jeigu yra absoliuti adresacija, tai suformuojamas ne tik efektyvus adresas, bet ir absoliutus fizinis adresas atmintyje iš 20 bitų:  $\text{segm.nr} \cdot 2^4 + EA$ .

Betarpiškas operandas atskiriamas nuo tiesioginės atminties adresacijos priklausomai nuo operacijos kodo, jeigu pati tiesioginė adresacija identifikuojama pagal operacijos kodą, o ne pagal adresavimo baitą.

### Adresavimo baitas 8 bitai



## KOMPIUTERIŲ ARCHITEKTŪRA



Dažniausiai operacijos kode yra d (destination) ir w (width) bitai:

d	w
---	---

d = {

- 0 – iš procesoriaus registro į atmintį
- 1 – iš atminties į procesoriaus registrą

w = {

- 0 – operandas baitas
- 1 – operandas žodis

2 baitai - žodis  
3, 4 baitai nevadinami kažkaip

Lauko *reg* reikšmės:

<i>reg</i> galima suprasti kuris registras ir pagal papildoma bituka OPK	Registras	
	priklausomai nuo to ar žodis ar baitas, naudojame atitinkamus registrus <b>w = 0</b>	<b>w = 1</b>
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

IP nepatenka  
nes jis nėra  
pilnavertis  
registras

## KOMPIUTERIŲ ARCHITEKTŪRA

Lauko r/m reikšmės:

operandas atmintyje ir be jokio poslinkio

sk su zenklu i  
prieki su 127  
ar atgal su - 128

turime reikalą su registrais

r/m	mod = 00	mod = 01 poslinkis 1 baito	mod = 10 poslinkis 2 baitų	mod = 11 su puse registro ar visu registru w = 0 w = 1
000	BX + SI	BX + SI + <i>poslinkis</i>		AL
001	BX + DI	BX + DI + <i>poslinkis</i>		CL
010	BP + SI	BP + SI + <i>poslinkis</i>		DL
011	BP + DI	BP + DI + <i>poslinkis</i>		BL
100	SI	SI + <i>poslinkis</i>		AH
101	DI	DI + <i>poslinkis</i>		CH
110	20 bitų adresai tiesioginis adresai	BP + <i>poslinkis</i>		DH
111	BX	BX + <i>poslinkis</i>		BH

puse registro  
paimti negalima

du 16 bitų  
registrai

taip sugalvota, kad  
bx galima naudoti

su ax, cx, dx užkoduoti komandos negalima

Kai mod = 00 ir r/m = 110, tuomet turime tiesioginę adresaciją, kai tiesioginis 2 baitų adresai-poslinkis segmente nurodomas po adresavimo baito, vietoje poslinkio, kadangi mod = 00. Tiesioginę adresavimo būdą turime praradami adresavimą BP.

### Adresavimo būdai

Nagrinėsime operandų atmintyje adresavimo būdus. Tiesioginė adresacija yra:

1. Pagal adresavimo baitą.
2. Pagal operacijos kodą:
  - 1) paprasta tiesioginė adresacija;
  - 2) santykinė tiesioginė adresacija;
  - 3) absoliuti tiesioginė adresacija.

Tiesioginė adresacija yra tinkama tuomet, kai operandų adresai yra žinomi prieš programos vykdymą: programos sudarymo arba transliavimo metu, nes tiesioginis adresai yra komandos dalis.

Norint tiesiogine adresacija programuoti masyvo apdorojimą cikle, reikėtų modifikuoti komandas programos vykdymo metu, t.y., keisti komandų adresinę dalį. Tačiau tokios programos nėra reenterabilios. Šiuolaikinė programinė įranga yra reenterabili. Tuo tikslu kintami adresai turi būti realizuojami ne komandų adresinių dalių modifikavimo būdu, o panaudojant registrus, kurių reikšmės ir yra keičiamos programos vykdymo metu, o programos komandose tėra pastovus registro numeris.

Jeigu ciklo vykdymo metu yra tiesinis kintamų adresų kitimas, tai tokiam kitimui realizuoti pakanka vieno registro reikšmių. Tą registrą vadiname indeksiniu registru. Jeigu yra du ciklai su nepriklausomai kintančiais ciklo parametrais, pavyzdžiui, matricos apdorojime, tuomet tikslinga turėti du indeksinius registrus kintamojo adreso reikšmei skaičiuoti.

## KOMPIUTERIŲ ARCHITEKTŪRA

Mikroprocesoriaus Intel 8088 architektūroje yra numatyta galimybė apskaičiuoti efektyvų adresą pagal šabloną

$$reg + poslinkis ,$$

kur poslinkis atspindi kintamo adreso pastovią komponentę, o *reg* kinta cikle pakartojimo metu. Taip pat efektyvus adresas gali būti skaičiuojamas pagal šabloną

$$reg1 + reg2 + poslinkis ,$$

ką yra tikslinga naudoti esant netiesiniam kintamo adreso kitimui.

Sudėties veiksmas, atliekamas efektyvaus adreso formavimo metu, leidžia taupyti papildomas sudėties komandas.

Savaime aišku, kad jeigu indeksų yra daugiau, t.y., įdėtųjų ciklų skaičius yra didesnis, arba net ir esant dviem indeksams, kintamo adreso reikšmę būtų galima apskaičiuoti papildomomis komandomis suskaičiuojant redukuotą indeksą ir „nuvalyti” operando adresą viename registre. Reenterabilumas būtų išsaugomas, bet efektyvumas prarandamas.

Registrų panaudojimas operando adreso formavimui reikalingas ne tik dėl kintamų adresų realizavimo reenterabilumo, bet ir dėl programos relokavimo efektyvumo, t.y., dėl galimybės programą efektyviai patalpinti į bet kurią atminties vietą. Šita problema gali būti sprendžiama įvedant bazinius registrus, kuriuose esanti reikšmė priklauso nuo programos vietos atmintyje, o komandų adresinės dalys, nurodomos bazinių registrų atžvilgiu, yra santykinės ir nuo programos vietos atmintyje nepriklauso, todėl talpinant programą į kitą atminties vietą, pati programa, jos adresinės dalys nesikeičia, išskyrus tam tikras išimtis. Mikroprocesoriuose Intel 8088 analogišką vaidmenį atlieka segmentų registrai, bet ne baziniai registrai. Baziniai registrai tarnauja kintamų programos vykdymo metu adresų formavimui ir yra analogiški indeksiniams registrams, išskyrus panaudojimą atskirose komandose.

Netiesioginė adresacija yra nusistovėjusi sąvoka, kuri reiškia, kad efektyviu adresu esanti reikšmė yra traktuojama kaip operando adresas, kuris ir dalyvauja operacijoje. Tuo tarpu daugelyje knygų netiesiogine adresacija vadinamas operando efektyvaus adreso formavimas pagal registro turinį. Tačiau tai yra tiesiog efektyvaus adreso formavimo būdas, skirtingas nuo tiesioginės adresacijos, bet tai nėra netiesioginė adresacija.

Segmentavimas yra visų pirma reikalingas kaip priemonė, daranti programą mažai priklausomą nuo jai išskirtos vietos atmintyje. Antras, labai svarbus, segmentavimo privalumas yra tas, kad sudarant programą iš kelių modulių, ryšių redaktoriui nereikia ištiesinti visų modulių į vieną tiesinę adresų erdvę. Pakanka tokį tiesinimą atlikti tik vieno segmento ribose, ir taip galima efektyviau atlikti ryšių redagavimo darbą.

Mikroprocesorius Intel 8088 turi keturis segmentų registrus: CS, DS, SS ir ES. Šių registrų reikšmėmis yra saugomi atitinkamai kodo, duomenų, steko ir papildomo duomenų segmentų paragrafų numeriai. Segmentų registrai gali būti bet kaip išdėstyti atmintyje vienas kito ir atminties atžvilgiu.

Kodo segmento registras, panaudojant registrą IP, yra naudojamas identifikuoti nuskaitomos vykdymui komandos absoliutų adresą atmintyje.

Absoliutus adresas yra užrašomas pora:  $\langle \text{segmento numeris} \rangle : \langle \text{EA} \rangle$

## KOMPIUTERIŲ ARCHITEKTŪRA

Operandų atmintyje absoliutūs adresai yra suformuojami pagal šabloną:

< DS > : < operando EA > ,

išskyrus tris išimtis:

1. Registro BP panaudojimas efektyvaus adreso formavimui iššaukia registro SS panaudojimą absoliutaus adreso formavime;
2. Visose komandose darbui su steku : PUSH, POP, CALL, RET suformuojamas absoliutus adresas steko segmente pagal SS : SP ;
3. Eilutinėse komandose rezultatas turi būti suformuotas su absoliučiu adresu tik papildomame duomenų segmente: ES : DI .

PUSH POP zino pagal nutylėjimą, kad tas bus stackas ir SP

	Nutylėjimas	Variantai	Poslinkis
Komandos paėmimas	CS	–	IP
Komanda su steku	SS	–	SP
Operandas atmintyje	DS	CS, SS, ES	EA
Eilutė – šaltinis	DS	CS, SS, ES	SI
Eilutė – gavėjas	ES	–	DI
BP – bazinis registras	SS	CS, DS, ES	EA

Jei stecko komanda, segmentas tai žiuresime su SP

Segmentų registrų numeriai:

00 – ES  
01 – CS  
10 – SS  
11 – DS

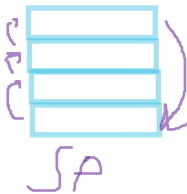
Segmento keitimo prefiksas: 001xx110, čia xx yra segmento registro numeris.

### Stekas

Maksimalus steko dydis yra 64 K baitų. Viršijant maksimalų steko dydį operacijomis PUSH, t.y., mažinant SP reikšmę ir jai tapus lygia 0, toliau vėl imama maksimali SP reikšmė, t.y., stekas yra ciklinis.

Stekas ciklinis

Stekas naudojamas laikinam reikšmių saugojimui, ryšių tarp paprogramių organizavimui, reikšmių išsaugojimui.



### Pertraukimų sistema

Pertraukimas tai yra procesoriaus sugebėjimas pristabdyti einamosios programos vykdymą, sureaguoti į įvykį, įvykdyti atitinkamą programą ir grįžti į pristabdytos programos vykdymą. Pertraukimai naudojami procesoriaus darbo efektyvumui padidinti, tam kad pačiam procesoriui nereikėtų periodiškai tikrinti išorinių įrenginių būsenas. Procesorius be pertraukimų būtų panašus į telefono aparatą be skambučio: reikėtų vis kilnoti ragelį ir klausyti, ar kas nors nenori pasikalbėti.

## KOMPIUTERIŲ ARCHITEKTŪRA

Taip pat pertraukimų aparatas naudojamas kaip patogi priemonė, leidžianti kreiptis į operacinės sistemos kontroliuojamas utilitas veiksams su išoriniais įrenginiais atlikti. Tokiam kreipimuisi atlikti pakanka žinoti pertraukimo numerį ir nereikia žinoti sisteminės programos adreso, kuris gali priklausyti nuo operacinės sistemos versijos ar panašiai.

Pertraukimų aparatas leidžia atlikti įvedimo-išvedimo operacijas nepriklausomai nuo procesoriaus, betarpiškai valdant kontrolieriams (adapteriams). Dėl elektroninių ir mechaninių komponentų greičių didelio skirtumo, įvedimo-išvedimo operacijos metu procesorius suspėja atlikti kitus darbus, o operacijos pabaigoje išorinis įrenginys praneša apie tai procesoriui pažįstamu pertraukimo signalu. Procesorius gali sureaguoti į pertraukimą ne bet kuriuo laiko momentu, o tik baigęs vykdyti einamąją komandą.

Galimas ir toks procesoriaus darbo režimas, kai procesorius nereaguoja į maskuojamų pertraukimų signalus.

Į nemaskuojamus pertraukimus, nepriklausomai nuo procesoriaus darbo režimo, yra nedelsiant sureaguojama (nedelsiant reiškia įvykdžius eilinę komandą).

Mikroprocesoriaus Intel 8088 pertraukimų sistemoje kiekvieną pertraukimo tipą atitinka kodas, jų gali būti 256.

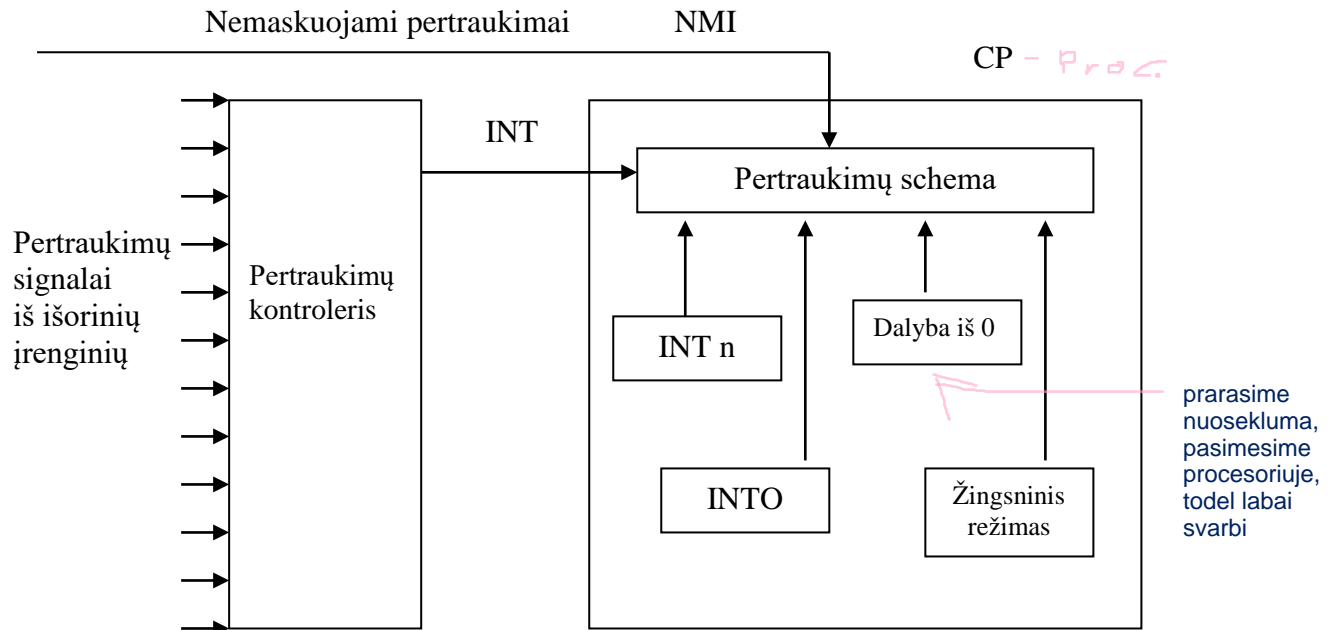
Pertraukimai gali būti išoriniai ir vidiniai procesoriaus atžvilgiu. Išoriniai pertraukimai generuojami išoriniuose procesoriaus atžvilgiu įrenginiuose ir gali būti maskuojami arba nemaskuojami. Pertraukimai gali kilti ir procesoriaus darbo, vykdant programą, rezultate – tai yra vidiniai pertraukimai, kurie kyla procesoriui vykdant specialias pertraukimų komandas INT, INTO, dalybos iš 0 atveju, vykdant komandas žingsniniu režimu, vykdant komandą *CC adresas* – pertraukimas duotame taške.

Svarbiausias reikalavimas pertraukimų sistemai yra tas, kad pratęsus pristabdytos programos vykdymą, toliau darbas vyktų taip, lyg to pertraukimo iš viso nebuvo. Tuo tikslu procesorius turi įsiminti vykdymo pratęsimo adresą (grįžimo adresą), esantį registruose IP ir CS. Šiuose registruose esančios reikšmės yra įsimenamos steke. Taip pat yra įsimenamas požymių registras, kuriame saugoma einamoji procesoriaus darbo būseną. Jeigu pertraukimą apdorojanti procedūra keičia kokių nors kitų registrų reikšmes, tai ji pati turi tų registrų reikšmes darbo pradžioje įsiminti, o darbo pabaigoje – atstatyti.



## KOMPIUTERIŲ ARCHITEKTŪRA

### Bendroji pertraukimų schema



Visi pertraukimai turi savo kodus nuo 0 iki 255. Vidiniai pertraukimai, priklausomai nuo jų pobūdžio, turi fiksuotą kodą arba kodas yra užrašomas komandoje. Vidinių pertraukimų negalima uždrausti, išskyrus pertraukimą žingsniniam komandų vykdymui. Vidinių pertraukimų prioritetai yra aukštesni už išorinių pertraukimų prioritetus (išskyrus žingsninį pertraukimą).

Pertraukimų prioritetai:

1. Dalyba iš nulio;
2. Programinis (vidinis) pertraukimas pagal komandą INT;
3. Pertraukimas pagal komandą INTO;
4. Nemaskuojamas išorinis pertraukimas;
5. Maskuojamas išorinis pertraukimas;
6. Žingsninio apdorojimo vidinis pertraukimas.

Pertraukimų prioritetai veikia tada, kai vienu metu (tame tarpe ir atreagavimo į pertraukimą metu), t.y., kol bus baigta vykdyti eilinė komanda, generuojami keli pertraukimų signalai. Tada pagal pertraukimų prioritetus nusprendžiama, kuris pertraukimas bus apdorotas. Laukimas, kol bus baigta vykdyti eilinė komanda, gali būti gana ilgas. Pavyzdžiui, daugybos-dalybos komandos procesoriuje Intel 8088 realizuojamos maždaug 150 taktų.

Pertraukimas	Kodas
Dalyba iš nulio	0
Žingsninio režimo	1
Nemaskuojamas išorinis	2
Sustojimo taško (programinis pertraukimas pagal komandą INT)	3

interrupt priority

## KOMPIUTERIŲ ARCHITEKTŪRA

Perpildymo (programinis pertraukimas pagal komandą INTO)	4
Operacinės sistemos reikmėms	5 – 31
Vartotojo reikmėms	32 - 255

Priklausomai nuo pertraukimo kodo, pagal pertraukimo vektorius, kuriuose nurodomi pertraukimo procedūrų adresai, iškviečiama pertraukimą apdorojanti procedūra. Pertraukimų vektoriai yra išdėstyti operatyviosios atminties mažesniuosiuose adresuose ( $0 - 3FF$  t.y.,  $4 \cdot 256$ ), kur kiekvieną pertraukimo kodą atitinka 4 baitų pertraukimo vektorius su adresu:  $4 \cdot \text{kodas}$ .

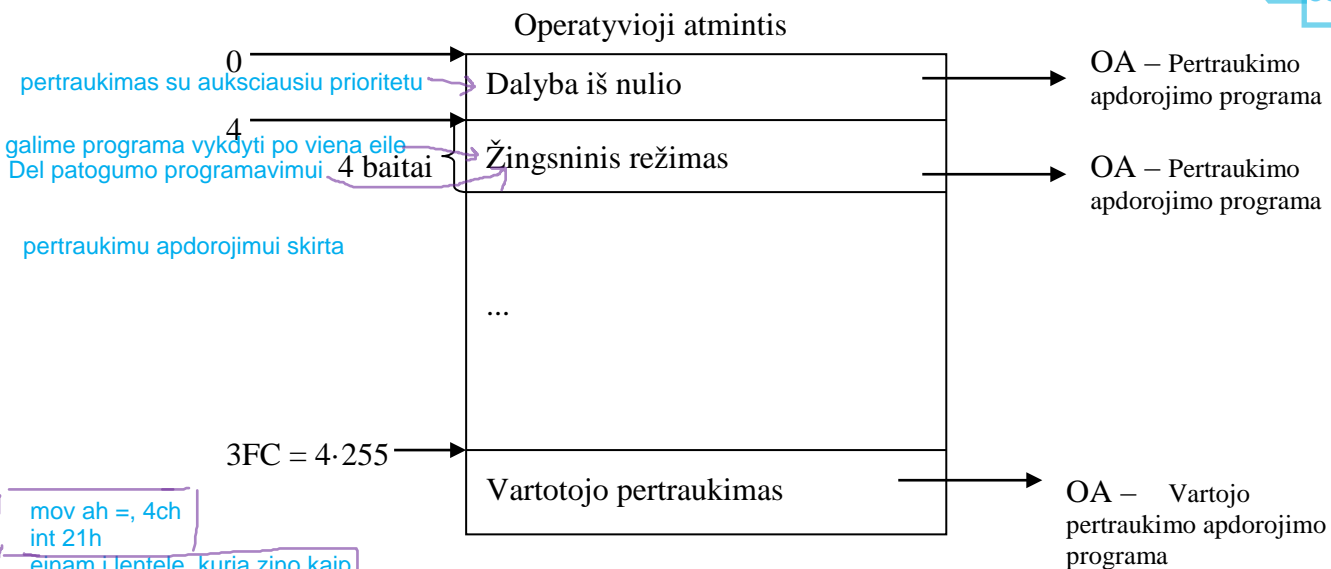
Adrese nurodoma:	i	i + 1
	IP	
	j.b	v.b
	CS	
	j.b	v.b
	i + 2	i + 3

j.b – jaunesnysis baitas;  
v.b – vyresnysis baitas.

Kadangi pertraukimo vektoriuje adresas nurodomas absoliutus: CS : IP, tai pertraukimo apdorojimo programa gali būti išdėstyta bet kurioje operatyviosios atminties vietoje, nuo  $0 - 2^{20} = 1 \text{ M}$ .

adresas kodo segmento  
kelinta instrukcija vykdo

CS ir IP po 2 baitus



pertraukimai eina dar per papildoma chipą, kuris po to kreipiasi į procesorių, intel 8259, nereikės per egzaminą

Pertraukimo apdorojimo programa (procedūra) pradeda dirbti procesoriaus režimu, kai pertraukimai neleidžiami, taip pat neleidžiamas ir žingsninis pertraukimas. Pertraukimo apdorojimo programa pati gali nustatyti procesoriaus režimą, leidžiantį pertraukimus. Be to, pertraukimo apdorojimo procedūros darbą gali pertraukti nemaskuojamas išorinis pertraukimas arba pati procedūra gali turėti komandą INT (vidinis pertraukimas), kuria pertraukia pati save. Bet kurie vidiniai pertraukimai gali pertraukti pertraukimo procedūros darbą ir perjungti į naują pertraukimo apdorojimo procedūrą.

## KOMPIUTERIŲ ARCHITEKTŪRA

Baigiant pertraukimo procedūros darbą yra svarbu iš steko atstatyti būseną, kuri buvo prieš pertraukimą. Pertraukimo procedūroje yra tikslinga uždrausti pertraukimus kritinėje sekcijoje, nes priešingu atveju gali gautis neapibrėžti rezultatai. Kita vertus, uždelsus sureaguoti į išorinį pertraukimą, jis gali dingti. Išorinio pertraukimo apdorojimo programa pati nurodo išoriniam įrenginiui nuimti pertraukimo signalą, tuo pačiu fiksuojant, kad į pertraukimą yra sureaguota.

### Pertraukimai, iššaukiami komanda INT n

Pertraukimai, iššaukiami komanda INT n, yra vadinami programiniais pertraukimais. Tuo pačiu jie yra vidiniai pertraukimai. Tai yra analogas ankstesnių kompiuterių komandai SCV, kurios pagalba vartotojiška programa susiriša su operacine sistema – supervizoriumi.

Esant dinaminiam atminties paskirstymui, kai programų vieta atmintyje kinta, tuo pačiu kinta ir pertraukimo apdorojimo programų vieta atmintyje. Todėl yra labai patogu pertraukimo programas iškviesti per pertraukimo kodą, nepriklausantį, nuo programos vietos atmintyje. Pertraukimo programai keičiant vietą atmintyje, pakanka pakoreguoti pertraukimo vektorių – programos adresą.

Programiniai pertraukimai yra būdas pasinaudoti operacinės sistemos paslaugomis, tai yra kreipiniai į operacinę sistemą.

Dalybos iš nulio atveju, laikas sunaudotas atlikti tokio pertraukimo apdorojimo programą turi būti priskirtas dalybos komandos veiksmui, tam, kad įvertinti maksimalų nepertraukiamos iš išorės komandos vykdymo laiką.

Dar kartą pabrėšime, kad aparatūriškai sureaguojant į pertraukimą, steke yra įšimenamas procesoriaus būsenos registras SF (Status Flag) ir CS : IP (sekančios komandos adresai), o procesoriaus režimai nustatomi taip:

IF = 0 pertraukimai neleidžiami;

TF = 0 žingsninis pertraukimas neleidžiamas.

Grįžtant iš pertraukimo reikia atstatyti iš steko buvusią procesoriaus būseną ir grįžimo iš pertraukimo adresą.

### Žingsninio režimo pertraukimas

Žingsninio pertraukimo režimas yra naudojamas programų derinimui, stebint kiekvienos komandos vykdymo efektą. Žingsninio pertraukimo būsenos TF požymio programuotojas tiesiogiai keisti negali, tačiau yra netiesioginė galimybė tą atlikti „išsimenant“ procesoriaus požymių registrą steke, tada požymius steke pakoreguoti, o po to iš steko atstatyti procesoriaus požymių registrą SF.

### Kontrolinio taško pertraukimas

Kontrolinio taško pertraukimas – INT su kodu 3 yra vieno baido komanda. Kitų programinių pertraukimų komanda INT n užima du baitus atmintyje: INT ir kodas n. Tuo tarpu komanda INT 3 yra specialaus operacijos kodo komanda, kuri generuoja

## KOMPIUTERIŲ ARCHITEKTŪRA

pertraukimo kodą 3, o ne paima jį iš programos. Pagal tą kodą 3, yra nustatomas pertraukimo vektoriaus adresas (*pertraukimo kodas* · 4 = *tą kodą atitinkantis pertraukimo vektoriaus adresas*), o tada nustatoma, kur yra pertraukimo apdorojimo programa (pertraukimo vektoriaus reikšmė yra pertraukimo programos adresas kur nors atmintyje). Komanda INT 3 yra įdedama į programą toje vietoje, kurią vykdant norisi iššaukti pertraukimą. Taigi joks kontrolinio taško adresas nereikalingas, tai yra tiesiog pačios komandos INT adresas, šitos komandos operacijos kodas yra CC.

Kontroliniai taškai yra numatyti iš anksto, sudarant programą, stabilūs kontroliniai taškai nėra įdomūs. Labai svarbu derinimo metu dinamiškai užkoduoti kontrolinius taškus, priklausomai nuo dinaminės situacijos derinimo metu. Dinaminio kontrolinio taško adresą nurodo vartotojas. Toks dinaminis kontrolinis taškas yra realizuojamas tuo adresu esantį baitą pakeičiant komanda CC ir sudarant tokią pertraukimo apdorojimo programą, kuri įsimeina pakeistą baitą, atstato procesoriaus būseną, grąžina tą baitą į jo vietą ir programos vykdymas yra tęsiamas toliau.

kelios išimtys kai komanda dar vykdoma prieš interrupt

Pastaba. Pertraukimas fiksuojamas ne einamosios komandos pabaigoje, o kitos (toliau esančios) komandos pabaigoje šiais atvejais:

reikalinga kai turim darba su procesoriumi

1. Prefiksinės komandos – prefikso (pakartojimo, segmento keitimo, magistralės blokavimo) atveju. Tačiau yra išimtis – WAIT ir eilutinės komandos su pakartojimo prefiksu gali būti pertrauktos;
2. Po komandų, keičiančių segmentų registrus: MOV arba POP, nepertraukiamai įvykdoma kita (toliau esanti) komanda tam, kad būtų pilnai pakeistas absoliutus adresas.

egzo klausimas

**Vykdydamas pertraukimą procesorius atlieka šiuos veiksmus:**

1. Į steką patalpinamas požymių registras SF;
2. Požymių TF ir IF reikšmės nustatomos lygios nuliui, tam kad būtų uždraustas žingsninis režimas ir išorinis pertraukimas;
3. Į steką patalpinamas registras CS;
4. Apskaičiuojamas pertraukimo vektoriaus adresas:  $4 \cdot \text{pertraukimo kodas}$ ; <sup>4 baitai</sup>
5. Pertraukimo vektoriaus antras žodis patalpinamas į registrą CS;
6. Į steką patalpinama registro IP reikšmė;
7. Pertraukimo vektoriaus pirmas žodis patalpinamas į registrą IP.

Pertraukimo apdorojimo programa turi baigtis komanda IRET, kuri atlieka šiuos (aukščiau išvardintus) veiksmus atvirkščia tvarka (FIFO principu), t.y. iš steko viršūnės atstato registrų IP, CS ir SF reikšmes. <sup>first in, first out</sup>

Pastabos.

Išorinių pertraukimų maskavimas gaunamas pasiuntus pertraukimų kontrolerio registrui tam tikrą informaciją.

NMI pertraukimas įvyksta dėl atminties klaidų, maitinimo išjungimo ir pan. Šio pertraukimo kodas yra 2. NMI užklausus procesorius įsimeina.

Pertraukimų prioritetai naudojami tada, kai kyla pertraukimas kito pertraukimo apdorojimo metu, t.y., dirbant pertraukimo apdorojimo programai. Jeigu naujai kilusio pertraukimo prioritetas yra aukštesnis negu einamuoju momentu apdorojamo pertraukimo prioritetas, tai pertraukimo apdorojimo programos darbas pertraukiamas

## KOMPIUTERIŲ ARCHITEKTŪRA

ir persijungiama prie naujo pertraukimo apdorojimo programos vykdymo, žinoma, atsižvelgiant į procesoriaus darbo režimą dėl maskuojamų išorinių pertraukimų (jeigu jie yra uždrausti, tai ir nepertrauks).

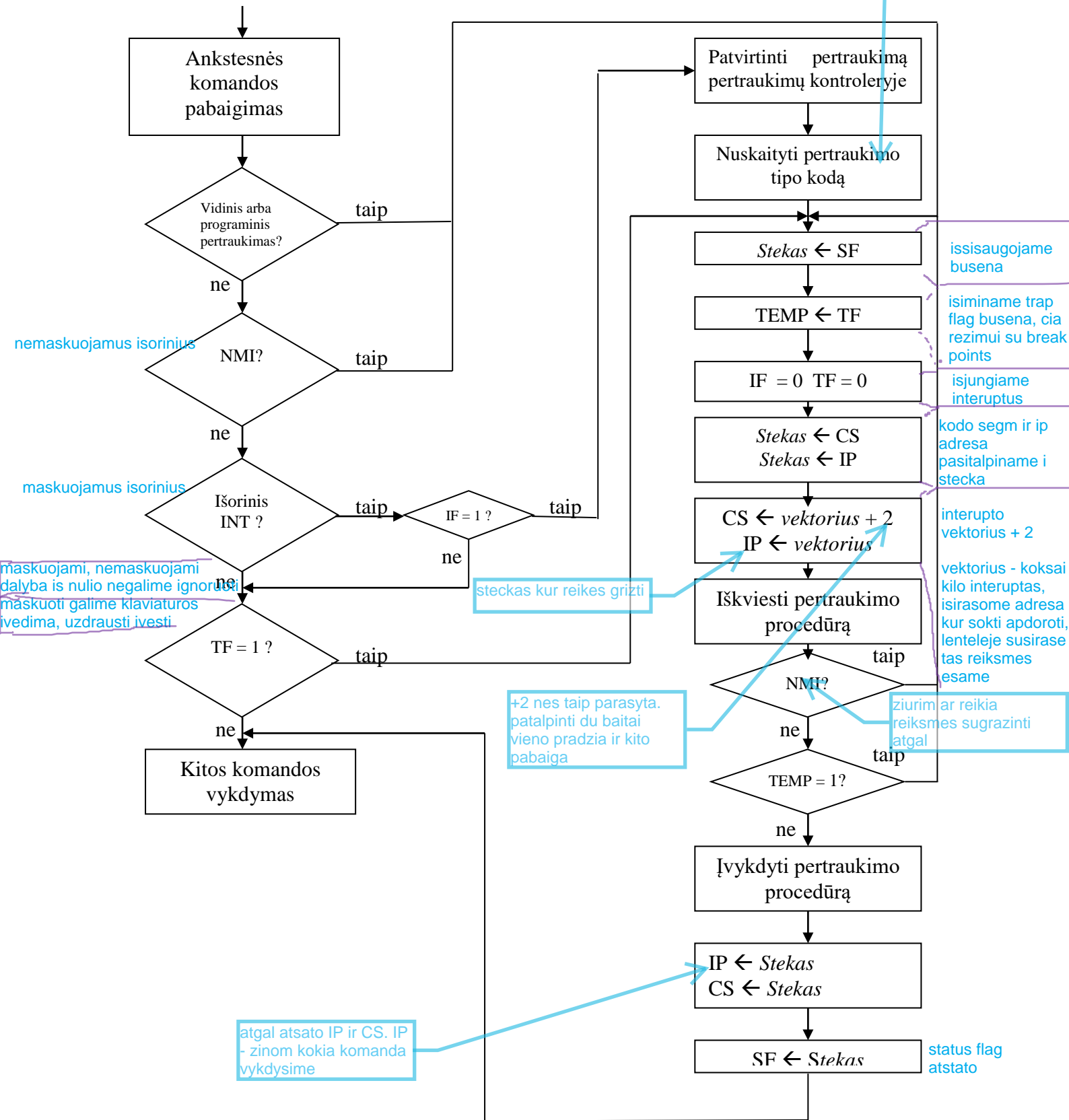
Vienu metu kylant keliems pertraukimo signalams, yra sureaguoja į tą, kurio prioritetas aukščiausias. Pavyzdžiui, jeigu vykdant komandą kyla vidinis dalybos iš nulio pertraukimas ir tuo pat metu pasiunčiami NMI arba maskuojami išorinių įrenginių pertraukimo signalai, tai bus įvykdytas vidinis dalybos iš nulio pertraukimas.

gal nereikės isiminti egzaminui <sup>set interrupt</sup> Po komandų STI ir IRET, po segmento keitimo ir magistralės blokavimo prefiksų, po MOV *seg. reg.* ir POP *seg. reg.* pertraukimas iš karto neįvyksta, o įvyksta tik įvykdžius dar vieną komandą.

Po komandų keičiančių segmento registro reikšmes (MOV *seg. reg.* ir POP *seg. reg.*) pertraukimas kyla įvykdžius tik dar vieną komandą – toks mechanizmas apsaugo programą, kuri pereina prie naujo steko keičiant registrų SS ir SP reikšmes. Jeigu pertraukimas kiltų pakeitus vien tik registro SS reikšmę, bet prieš pakeičiant registro SP reikšmę, mikroprocesorius išsaugotų požymių registrą SF, registrus CS ir IP netinkamoje atminties srityje, tai seka iš to, kad teisingą steko padėtį apibrėžia registrų pora: SS ir SP. Jeigu pakeičiama tik registro SS reikšmė, tai turime nesuderintą stekinės atminties adresavimo registrų porą.

# KOMPIUTERIŲ ARCHITEKTŪRA

## Pertraukimų įvykdymo schema



turime 20 bitu todėl reikia 16 bitu patalpinti ir dar 4 bitus patalpinti atskirai

### ***Duomenų formatai***

Duomenų formatas yra tam tikras informacijos kodavimo būdas, kuris priklauso nuo procesoriaus komandų sistemos. Duomenų formatai yra komandų sistemos atributas, apibrėžimo sudėtinė dalis.

Duomenų formatai tai yra sąvoka, kurios terminais apibrėžiama komanda atliekamas informacijos atvaizdavimas. Tam atvaizdavimui nusakyti bitai yra grupuojami į tam tikrus laukus. Tų laukų terminais nusakomas komanda atliekamas atvaizdavimas.

Ta pati informacija, priklausomai nuo ją apdorojančios komandos, gali būti skirtingais duomenų formatais. Todėl duomenų formatai yra sąvoka, skirta komandų sistemos paaiškinimui.

Duomenų tipai ir duomenų formatai yra mažai ką bendro turinčios sąvokos. Duomenų tipas nusako reikšmių aibę. Reikšmei užkoduoti gali būti panaudoti įvairūs formatai. Tas pats formatas gali būti naudojamas įvairių tipų reikšmėms koduoti.

Duomenų formatai yra šie:

1. Skaičiai be ženklo;
2. Skaičiai su ženklu;
3. Dešimtainiai supakuoti skaičiai;
4. Dešimtainiai išpakuoti skaičiai;
5. Simbolinis formatas.

Skaičių be ženklo formatu visi skaičiaus bitai yra interpretuojami kaip dvejetainis skaičius.

Atliekant aritmetinius veiksmus tenka operuoti neigiamais skaičiais. Yra toks neigiamų skaičių kodavimas, kad vietoje atimties veiksmo galima naudoti sudėties veiksmą, t.y., galima apsieiti be aparatūrinės atimties komandos realizacijos, o naudoti tik sudėties komandą ir tinkamą neigiamų skaičių kodavimą.

nepasidarom  
specialios atimties  
komandos, tik reikia  
sudėti teisingai  
daryti užkoduojant  
skaičius teisingai.

Pabrėšime, kad galima apsieiti tik su sudėties veiksmu. Pavyzdžiui,  $5 + (-3)$  : nėra aišku, kaip atlikti tokią sudėtį. Sudėtis nėra apibrėžta algoritmiškai skirtingų ženklų skaičiams. Algoritmiškai galėtume apibrėžti atimtį. Jeigu reikia atlikti veiksmą vienetų tikslumu, tai  $-3$  užkodavę 7, gausime  $(5 + 7) \bmod 10 = 2$ . 7 yra atvirkštinis elementas skaičiui 3 sudėties moduliui 10 atžvilgiu, kaip ir  $-3$  būtų atvirkštinis elementas skaičiui 3 atimties atžvilgiu.

Neigiamiems skaičiams koduoti parenkamas toks teigiamas skaičius, kuris yra atvirkštinis operacijos (sudėties) mod  $2^n$  atžvilgiu.

## KOMPIUTERIŲ ARCHITEKTŪRA

0	0000 0000	0
1	0000 0001	1
2	0000 0010	2
...		
$2^7-1$	0111 1111	127
$2^7$	1000 0000	-128
$2^7+1$	1000 0001	-127
...		
$2^8-2$	1111 1110	-2
$2^8-1$	1111 1111	-1

Tokiu būdu neigiamiems skaičiams koduoti yra naudojami teigiami skaičiai ir yra atliekamos operacijos su teigiamais skaičiais, o neigiami skaičiai yra tik tam tikra interpretacija, nes algoritmiškai veiksmai yra atliekami su teigiamais skaičiais. Pasakymas „ženklų bitai“ yra tik interpretacijos lygio terminas. Faktiškai ženklų iš viso nėra, o yra tik didesnių reikšmių diapazonas.

Kodas, kuomet didesnių reikšmių diapazonas traktuojamas kaip neigiamų reikšmių diapazonas, yra vadinamas papildomu kodu iki  $2^n$ . Duomenų formatas, kuriame didesnių reikšmių diapazonas traktuojamas kaip papildomas kodas, yra vadinamas skaičių su ženklu formatu.

Ir skaičiai su ženklu, ir skaičiai be ženklų gali užimti vieną arba du baitus.

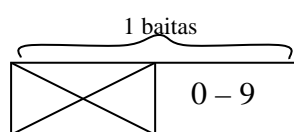
Skaičiai be ženklų:   baitė   0 – 255  
                               žodyje 0 –  $2^{16}-1$   
 Skaičiai su ženklu:   baitė   -128 – 127  
                               žodyje -32768 – 32767

Dvejetainis formatas yra ekvivalentus šešioliktainiam formatui, todėl apie šešioliktainius skaičius be ženklų ar su ženklu arba dvejetainius skaičius be ženklų ar su ženklu atskirai nėra kalbama. Šešioliktainiai skaičiai naudojami patogumo, tame tarpe ir užrašo trumpumo, sumetimais.

Skaičių be ženklų ir skaičių su ženklu formatas vadinamas fiksuoto kablelio formatu.

Skaičiai be ženklų ir skaičiai su ženklu priklauso gana ribotam diapazonui. Jeigu reikia operuoti didesniais skaičiais, tada yra naudojami dešimtainiai skaičiai arba slankaus kablelio skaičiai. Slankaus kablelio skaičiai yra tik koprocessorių 8087, bet ne Intel 8088. Taigi lieka tik dešimtainiai skaičiai, kurie gali būti supakuoti arba išpakuoti. Šitas formatas yra viename baite, o diapazonai yra tokie:

išpakuotas formatas: 0 – 9 ;



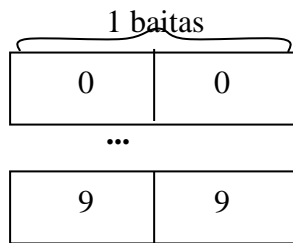
visų bitų neišnaudosime

zona



## KOMPIUTERIŲ ARCHITEKTŪRA

supakuotas formatas: 0 – 99



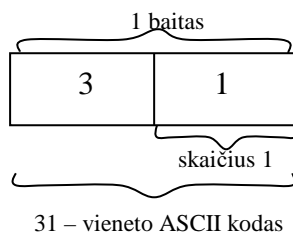
Dešimtainiai skaičiai nenaudoja kombinacijų A, B, C, D, E ir F, o tik skaitmenis 0 – 9, keturiose dvejetainėse pozicijose (pusbaityje). Čia prarandamas saugojimo talpumas, tačiau kompensuojama panašumu į simbolinį formatą. Dėl šio panašumo išvengiama sąnaudų transformavimui iš simbolinio formato į dvejetainį ir atvirkščiai.

Reikiamas tikslumas pasiekiamas skiriant pakankamai vietos dešimtainiam skaičiui pavaizduoti. Veiksmai su dešimtainiais skaičiais yra mažiau efektyvūs negu veiksmai su dvejetainiais skaičiais.

Supakuoti dešimtainiai skaičiai saugomi baite po du skaitmenis, taip kad vyresniuose bituose yra vyresnis skaitmuo. Galima būtų didelius skaičius vaizduoti ir dvejetainiu formatu, panaudojant kelis žodžius, bet tam reikia pervesti dešimtainį skaičių į dvejetainį, kelių žodžių formate.

Veiksmuose su dešimtainiais skaičiais komandos taikomos vienam baitui. Veiksmas su keliais baitais atliekamas kelis kartus pritaikant komandą, kurios operandas yra vieno baito, keičiant operacijoje dalyvaujantį baitą.

Išpakuoti dešimtainiai skaičiai saugomi po vieną skaitmenį baite. Pats skaitmuo saugomas jaunesniuose bituose. Kai kurios komandos (daugybės, dalybos) reikalauja, kad vyresniuose bituose būtų nuliai. Kai kurios komandos (sudėtis, atimtis) vyresnius baito bitus ignoruoja. Pavyzdžiui, skaitmenų simboliniame kodavime vyresniuose bituose yra 3, o jaunesniuose bituose pats skaičius, pvz.:



Kai reikia atlikti veiksmus su tokiais skaičiais, dirbame su skaičių išreiškiančiu pusbaičiu, o kai veiksmai atliekami su simboliais, tada dirbame su visu baitu.

Atliekant veiksmus, yra naudojamos tos pačios komandos, kaip ir su dvejetainiais skaičiais, tik po to reikalinga korekcija tam, kad gauti korektišką rezultatą.

## KOMPIUTERIŲ ARCHITEKTŪRA

Pavyzdžiui:

$$\begin{array}{rcl} & 09 & \\ + & \underline{08} & \\ \hline & 11_{16} & \text{sudėties rezultatas} \\ + & \underline{6} & \text{DAA} \\ \hline & 17 & \text{dešimtainis skaičius} \end{array}$$

Dešimtainėje sistemoje skaičiaus “svoris” yra šešiais mažesnis negu šešioliktainėje sistemoje, todėl prie gauto rezultato dar pridedame 6.

$$\begin{array}{rcl} & 07 & \\ + & \underline{04} & \\ \hline & 0B_{16} & \text{sudėties rezultatas} \\ + & \underline{6} & \text{DAA} \\ \hline & 11 & \text{dešimtainis skaičius} \end{array}$$

Simbolinis formatas naudojamas simboliams užkoduoti vieno baido kodu. Iš viso galima užkoduoti  $2^8 = 256$  simbolių. Kiekvienas simbolis turi standartinį grafinį pavaizdavimą. Kompiuteriuose IBM PC naudojamas standartinis simbolinis formatas ASCII (American Standard Code for Information Interchange).

Kompiuteryje IBM 360/370 yra naudojamas standartinis simbolinis formatas EBCDIC (Extended Binary Coded Decimal Interchange Code).

Asembleris gali apdoroti simbolius tik iš diapazono  $20H - 0FFH$ . Kodai iš diapazono  $00H - 01FH$  gali būti užduoti ne kaip simbolių eilutė, bet kaip skaitmeninis 8 bitų kodas.

ASCII kodų lentelė nuo  $00H$  iki  $7FH$  vadinama pagrindine, kurioje, pvz.:

Šešioliktainis (hex) kodas	Simbolis
20	<i>tarpa (space)</i>
21	!
...	...
2F	/
30	0
31	1
...	...
39	9
...	...
41	A
42	B
...	...
4F	O
50	P

## KOMPIUTERIŲ ARCHITEKTŪRA

...	...
5A	Z
...	...
61	a
62	b
...	...
6F	o
70	p
...	...
7A	z
...	...

Kodai iš diapazono 00H – 1FH yra vadinami valdančiais. Kodai 080H – 0FFH yra standartinio 7 bitų ASCII kodo išplėtimas personaliniams kompiuteriams. Išplėstinėje kodų lentelėje yra įvairių abėcėlių raidės, matematiniai simboliai, pseudografika (nei lietuviška, nei rusiška abėcėlė į standartinį kodą neįeina).

### **Požymių registras SF**

Požymių registras SF (Status Flag) neturi numerio, nėra adresuojamas, o jo panaudojimas yra susietas su konkrečiomis komandomis. Tai yra paskutinis iš nagrinėjamų registrų, jis yra 16 bitų, iš kurių ne visi yra naudojami. Visas registras dalyvauja tik dviejose operacijose: įdėti SF į steką ir paaimti SF iš steko, o visos kitos komandos operuoja su atskirais registro SF bitais, kurie turi individualius vardus ir yra vadinami požymiais. Kai kurie iš bitų fiksuoja įvykdytų komandų sąlygos kodus, pagal kuriuos organizuojamas sąlyginis valdymo perdavimas, kiti bitai parodo einamąją procesoriaus būseną.

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

nenaudojami 8086

Požymius jaunesniuose bituose nustato aritmetinės loginės komandos.

CF	Carry Flag	pernešimo požymis
PF	Parity Flag	lyginumo požymis
AF	Auxiliary Carry Flag	papildomo pernešimo požymis
ZF	Zero Flag	nulio požymis
SF	Sign Flag	ženklų požymis
TF	Trap Flag	“spąstų” požymis
IF	Interrupt Flag	pertraukimo leidimo požymis
DF	Direction Flag	krypties požymis
OF	Overflow Flag	perpildymo požymis

100-99=001

10110000  
00111111  
00100000

lyginis ar  
nelyginis  
vienetu sk

su kiekvienu  
zingsniu  
sustabdyti proc.  
architektūr.  
Debug

25+  
75  
A0

vyriausiam  
bite  
vienetuka

ar turetu reguoti  
i pertraukimus

kopijuoti duomenis nuo galines  
atminties dalies. kad nebutu taip kad  
nukopijavau duomenis ir vel ant virsaus kopijuojame

netelpa i duomenų struktūrą atsakymas

## KOMPIUTERIŲ ARCHITEKTŪRA

Registas SF:

### 0-inis bitas

CF – Carry Flag – pernešimo požymis, naudojamas didesnio tikslumo aritmetiniuose veiksmuose. Paprastai aritmetiniai veiksmai atliekami su duomenimis 8 arba 16 bitų. Tačiau kartais reikia apdoroti skaičius, didesnius už  $2^{16}$ . Didesni skaičiai gali būti saugomi keliuose žodžiuose, specialiai suformuojant tokius skaičius keliuose žodžiuose. Aritmetiniai veiksmai su didesniais skaičiais atliekami irgi specialiais veiksmiais programiniu būdu, remiantis veiksmų su žodžiais komandomis. Sudedant du 16 bitų žodžius, rezultatas gali netilpti į 16-ą bitų, nes jo saugojimui gali reikėti 17 bitų. Tokio, 17-tojo bito, vaidmenį atlieka požymis CF. Be to yra specialus ADD komandos variantas ADC, kuriuo prie operandų susumuoja CF bito reikšmę ir taip pat formuoja požymį CF.

Požymis CF yra naudojamas ir atimties komandoje SUB tam, kad atspindėti “pasiskolinimo” faktą, jeigu pirmasis operandas 16-oje bitų yra mažesnis už antrą operandą. Komanda SBB atima ne tik antrą operandą iš pirmojo, bet dar atima ir požymį CF, taip įskaitant “pasiskolinimą”.

Požymis CF galėtų būti naudojamas reikšmių palyginimui jas atimant. Toks palyginimo požymis tinka skaičiams be ženklo. Skaičiams su ženklu palyginti reikalingi papildomi požymiai.

### 1-as bitas

Nenaudojamas

### 2-as bitas

PF parodo rezultato jaunesniojo baido vienetinių bitų skaičiaus lyginumą. PF = 1, jei rezultato vienetinių bitų skaičius yra lyginis.

### 3-as bitas

Nenaudojamas

### 4-as bitas

AF – Auxiliary Carry Flag – papildomo pernešimo požymis, naudojamas atliekant aritmetinius veiksmus su dešimtainiais skaičiais. Šis požymis negali būti panaudotas sąlyginio valdymo perdavimo komandose.

jei panaudojame  
tris aaa raides. Tai  
yra supakuotas  
formatas

AF naudojamas tik veiksmams su  
dešimtainiu skaičiu

9 + 2 = B hex

Dešimtainė aritmetika panaši arba tiesiog atitinka simbolinę informaciją, kuria operuoja žmogus, kurią jis įveda į kompiuterį ir priima iš kompiuterio. Todėl taikymuose, kur skaičiavimai yra nesudėtingi, netikslinga eikvoti laiką ir resursus pervedimui iš simbolinio formato į fiksuoto kablelio dvejetainį vidinį formatą ir atvirkščiai.

Be to, dešimtainiai skaičiai gali būti naudojami kaip alternatyva slankaus kablelio skaičiams, kai neleistinas netikslumas, atsirandantis dėl dvejetainės sistemos nesugebėjimo tiksliai išreikšti dešimtaines reikšmes. Žinoma, tą patį būtų galima atlikti ir fiksuoto kablelio skaičiais, pervedant trupmeninius skaičius į sveikų skaičių

## KOMPIUTERIŲ ARCHITEKTŪRA

mastelį. Tai atliekama padauginant trupmeninį skaičių iš  $10^n$ , kur  $n$  yra trupmeniniame skaičiuje skaitmenų po kablelio skaičius. Tada programoje kablelis tegali būti menamas, programos turimas „omenų“.

Aritmetiniams veiksams su dešimtainio formato skaičiais naudojamos tos pačios fiksuoto kablelio skaičių komandos. Tačiau jų pagalba gautas rezultatas gali būti nekorektiškas dešimtainių skaičių formato prasme. Visų pirma, pusbaityje gali gautis šešioliktainė reikšmė, netenkinanti formato (A – F). Tokiu atveju tam, kad gauti korektišką rezultatą, reikia pridėti 6.

Atliekant sudėties veiksmą, rezultate gali įvykti pernešimas iš jaunesniojo pusbaitio į vyresnįjį. To pernešimo svoris yra 16 (sistemos pagrindas), o tuo tarpu dešimtainėje sistemoje pernešimo svoris yra 10. Todėl tam, kad pernešimą būtų galima traktuoti kaip dešimtainį, turime pridėti 6. Taigi jei buvo pernešimas per pusbaitį, tai reikia pridėti 6, o jei nebuvo, tada nebereikia nieko pridėti. Pernešimo per pusbaitį faktą ir fiksuoja papildomo pernešimo požymis AF.

Veiksmai su dešimtainiais skaičiais atliekami po vieną baitą. Paprastos baitų sudėties korekciją dešimtainio formato prasme atlieka komanda DAA, kuri prideda 6, jeigu požymis AF = 1, arba jeigu pusbaityje yra gautas šešioliktainis skaičius, netenkinantis formato (A – F).

Atimties atveju požymis AF yra formuojamas „pasiskolinimo“ atveju, o komanda DAS atlieka dešimtainės atimties korekciją – pagal nustatytą požymį atima 6.

Čia buvo kalbama apie supakuotus dešimtainius skaičius, kuomet viename baite saugomi du dešimtainiai skaitmenys. Požymis AF taip pat naudojamas išpakuotų dešimtainių dešimtainių skaičių sudėties ir atimties operacijų fiksuoto kablelio komandomis gautų rezultatų koregavimui. Tokios komandos yra AAA (sudėčiai) ir AAS (atimčiai). Šiomis komandomis formuojami skaičiai diapazone 0 – 9, viename baite, t.y., 30H – 39H ASCII kodais.

### 5-tas bitas

Nenaudojamas

### 6-tas bitas

ZF – Zero Flag – parodo, ar paskutinės operacijos rezultatas yra nulinis. ZF = 1, jei rezultatas = 0.

12

ZR

### 7-tas bitas

SF – Sign Flag – parodo, koks yra paskutinės operacijos rezultato vyriausias bitas (požymis nustatomas po aritmetinių ir loginių operacijų).

### 8-tas bitas

po viena instrukcija leidžiame programa. užstapdo procesoriaus darbą tas flagas.

TF – Trap Flag – žingsninio komandų vykdymo (spąstų) požymis. Jeigu jis yra nustatytas, tai po kiekvienos komandos (išskyrus tuos atvejus, kai pertraukimas leidžiamas tik įvykdžius kitą, toliau esančią komandą, t.y., prefiksinių komandų ir komandų, keičiančių segmento registrų reikšmes; plačiau tai yra aprašyta skyrelyje

## KOMPIUTERIŲ ARCHITEKTŪRA

apie pertraukimų sistemą) įvyksta pertraukimas su kodu 1. Šis požymis negali būti tiesiogiai valdomas komandomis.

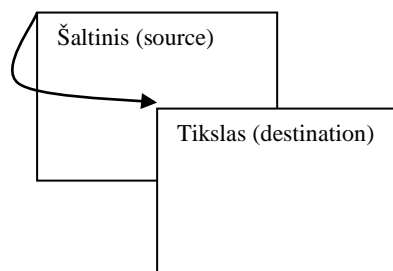
9-tas bitas → [interruptai, pertraukimai](#)

IF – Interrupt Flag – pertraukimo požymis. Jeigu IF = 1, tai yra leidžiami išoriniai maskuojami pertraukimai. Nemaskuojamas išorinis pertraukimas su kodu 2 kyla dėl operatyviosios atminties klaidų. Jeigu IF = 0, tada išoriniai pertraukimai neleidžiami. Požymis IF veikia iš karto visus išorinių įrenginių maskuojamus pertraukimus.

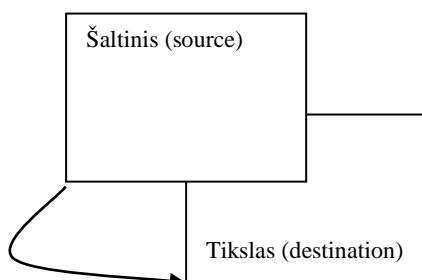
Įrenginiai yra pajungti prie pertraukimų kontrolierio, kuris yra išorinis įrenginys mikroprocesoriaus atžvilgiu. Galima siųsti informaciją iš mikroprocesoriaus į pertraukimo kontrolierio portus mikroprocesoriaus komandomis OUT. Pertraukimo kontrolieris yra tais portais susijęs su įrenginiais, kurie gali pasiųsti pertraukimo signalą. Jeigu reikia maskuoti tik kai kurių įrenginių pertraukimus, tai pertraukimo leidimą arba draudimą galima valdyti per pertraukimo kontrolierio portą, kuris sieja tą konkretų įrenginį su pertraukimų kontrolieriu. Tokiu atveju pertraukimas nuo konkretaus įrenginio yra užmaskuojamas pertraukimo kontrolierio lygyje, t.y., jis nėra perduodamas į mikroprocesorių. Pastebėsime, kad įprastu atveju pertraukimo maskavimas yra atliekamas mikroprocesoriuje, t.y., pertraukimų kontrolieris perduoda mikroprocesoriui pertraukimo signalą, kurį gauna iš įrenginio, o tada mikroprocesorius, priklausomai nuo požymio IF reikšmės, į jį reaguoja arba ne.

10-bitas [labiau pažiuresim kai imsime specialias komandas eiluciu kopijavimui](#)

DF – Direction Flag – krypties požymis yra naudojamas eilutinėse komandose, indeksinių registrų reikšmių keitimui. Jeigu DF = 0, tai indeksiniai registrai yra didinami vienetu arba dviem, priklausomai nuo to, ar operuojame baitais ar žodžiais. Jeigu DF = 1, tai indeksiniai registrai yra mažinami vienetu arba dviem, priklausomai nuo to, ar operuojame baitais ar žodžiais. Ar indeksinius registrus reikia didinti ar mažinti, nusprendžiame pagal situaciją. Pavyzdžiui.:



Jeigu taip, inkrementuodami, kopijuosime, sugadinsime šaltinį.



Taip kopijuojant ir dekrementuojant šaltinis nepažeidžiamas.

## KOMPIUTERIŲ ARCHITEKTŪRA

11-tas bitas kai vyriausiasis bitas isijungia sudedant du teigiamus skaičius. jei dedame skaičius be ženklo tai nesvarbu OF – Overflow Flag – perpildymo požymis. Tai yra vienintelis bitas vyresniajame požymių registro baite, formuojamas aritmetinėse komandose. Perpildymo bitas naudojamas skaičiams su ženklu (papildomu kodu), panašiai kaip pernešimo bitas CF yra naudojamas dešimtainio formato skaičiams.

Tiek skaičiams be ženklo, tiek skaičiams su ženklu, aritmetiškai sudėti naudojama ta pati komanda, kuri abiem atvejais duoda teisingą rezultatą, išskyrus tą atvejį, kai sudedant du skaičius su ženklu, gautas rezultatas netelpa skaičių su ženklu diapazone: -128 – 127 arba -32768 – 32767.

12-tas, 13-tas, 14-tas, 15-tas bitai  
Nenaudojami



## 5. Komandų sistema

Intel architektūrai būdinga savybė – naudojama vienoda mnemonika visiškai skirtingoms komandoms.

Žemiau pateikiami sutrumpinimų, naudojamų komandų aprašymuose, paaiškinimai:  
d – krypties (*destination*) bitas, jis nurodo, ar operacijos kryptis yra iš procesoriaus į atmintį, ar iš atminties į procesorių;

$$d = \begin{cases} 0 & \text{– iš procesoriaus registro į atmintį} \\ 1 & \text{– iš atminties į procesoriaus registrą} \end{cases}$$

w - pločio (*width*) bitas, jis nurodo operacijoje dalyvaujančio lauko ilgį; word ptr, byte ptr

$$w = \begin{cases} 0 & \text{– operandas baitas} \\ 1 & \text{– operandas žodis} \end{cases}$$

mod lauko reikšmės: poslinkis.

$$\text{mod} = \begin{cases} 00 & \text{– operandas r/m atmintyje (r/m = m), poslinkio nėra;} \\ 01 & \text{– operandas r/m atmintyje (r/m = m), vieno baido poslinkis;} \\ 10 & \text{– operandas r/m atmintyje (r/m = m), dviejų baitų poslinkis;} \\ 11 & \text{– operandas r/m registre (r/m = r);} \end{cases}$$

kur laukas r/m nurodo operandą atmintyje m – (*memory*) arba registre r – (*register*);

bet.op. – betarpiškas operandas.

j.b	v.b
-----	-----

adr.j.b. – adreso jaunesnysis baitas

adr.v.b. – adreso vyresnysis baitas

sreg – segmento registras;

$$\text{sreg} = \begin{cases} 00 & \text{– ES;} \\ 01 & \text{– CS;} \\ 10 & \text{– SS;} \\ 11 & \text{– DS;} \end{cases}$$

portas – porto numeris diapazone 0 – 255

posl.j.b. – poslinkio jaunesnysis baitas

posl.v.b. – poslinkio vyresnysis baitas

seg.reg.j.b. – segmento registro jaunesnysis baitas

seg.reg.v.b. – segmento registro vyresnysis baitas



## ***Bendrosios mikroprocesoriaus komandos***

**MOV** – MOVE – persiuntimas

1. registras  $\leftrightarrow$  registras / atmintis;

1000 10dw mod reg r/m [poslinkis];

-e cs:100  
..... B8,89     ??  
.....

d = 0:            reg  $\rightarrow$  r / m

d = 1:            r / m  $\rightarrow$  reg

2. betarpiškas operandas  $\rightarrow$  registras / atmintis;

1100 011w mod 000 r/m [poslinkis] bet.op1 [bet.op.2, jei w = 1]

Krypties bito nėra, nes kryptis jau yra apibrėžta.

Laukas reg nereikalingas, jis yra naudojamas kaip operacijos kodo išplėtimas.

Poslinkis suformuojamas priklausomai nuo lauko *mod* reikšmės.

3. betarpiškas operandas  $\rightarrow$  registras

1011 w reg bet.op.1 [bet.op.2, jei w = 1]

4. atmintis  $\rightarrow$  akumulatorius

1010 000w adr.j.b adr.v.b.

Komandos formatas yra toks, kad registras ir operandas betarpiškai nenurodyti.

Iš operacijos kodo yra žinoma, kad registras yra akumulatorius.

5. akumulatorius  $\rightarrow$  atmintis

1010 001w adr.j.b adr.v.b.

Pastaba. Atvejų 4. ir 5. negalima sujungti į vieną 101000 d w adr.j.b adr.v.b., nes bitas d apibrėžia priešingos krypties perdavimą, negu yra nurodyta 4. ir 5. atvejais.

6. registras / atmintis  $\leftrightarrow$  segmento registras

1000 11d0 mod 0 sreg r/m [poslinkis]

Kai d=1, sreg negali būti lygus 01, nes negalima nusiųsti reikšmės į registrą CS. Priešingu atveju priskyrimo veiksmu pakeistume CS registro reikšmę, o ji gali būti pakeista tik valdymo perdavimo komandomis.

Nėra w bito, nes operandas visada 2 baitų.

**PUSH** – įdėjimas į steką.

Prieš rašant į steką reikia sumažinti steko rodyklę : SP = SP - 2 (steke operuojame žodžiais).

1. registras / atmintis  $\rightarrow$  stekas

1111 1111 mod 110 r/m [poslinkis]

## KOMPIUTERIŲ ARCHITEKTŪRA

Tai yra vieno operando komanda, laukas *reg* yra naudojamas kaip operacijos kodo išplėtimas.

2. registras → stekas  
0101 0reg

Nėra varianto, ar registras baitinis ar žodinis, nes vienareikšmiškai jis yra žodinis.

3. segmento registras → stekas  
000 sreg 110

**POP** – paėmimas iš steko.

Paėmus iš steko reikia padidinti steko rodyklę :  $SP = SP + 2$  (steke operuojame žodžiais).

1. stekas → registras / atmintis  
1000 1111 mod 110 r/m [poslinkis]

2. stekas → registras  
01011 reg

3. stekas → segmento registras  
000 sreg 111

sreg negali būti lygu 01, nes tada pakeistume CS registro reikšmę, o ji gali būti pakeista tik valdymo perdavimo komandomis.

**XCHG** – eXCHanGe – apsikeitimas reikšmėmis.

1. registras / atmintis ↔ registras  
1000 011w mod reg r/m [poslinkis]

2. registras ↔ akumulatorius  
10010 reg

Nėra w bito, nes šia komanda vykdomas apsikeitimas tarp akumulatoriaus AX ir dviejų baitų registro,  $reg = \{ AX, CX, DX, BX, SP, BP, SI, DI \}$

Komanda XCHG AX, AX yra ekvivalenti NOP, t.y., operacijos kodas yra 1001 0000, t.y., 90h.

**OUT** – išvedimas į portą.

1. Fiksuotas portas.  
1110 011w portas;

## KOMPIUTERIŲ ARCHITEKTŪRA

Registre AL (jei  $w = 0$ ) arba registre AX (jei  $w = 1$ ) esanti reikšmė nusiunčiama į portą, nurodytą komandoje porto numeriu.

2. Kintamas portas.

1110 111w

Porto numeris yra imamas iš registro DX, todėl galima nurodyti 65 536 skirtingus portus. Vykdam komandą, reikšmė iš AL (jei  $w = 0$ ) arba iš AX (jei  $w = 1$ ) yra siunčiama į portą.

**IN** – nuskaitymas iš porto.

1. Fiksuotas portas.

1110 010w portas

Vykdam komandą, iš porto su nurodytu numeriu padedama reikšmė į registrą AL (jei  $w = 0$ ) arba į registrą AX (jei  $w = 1$ ).

2. Kintamas portas.

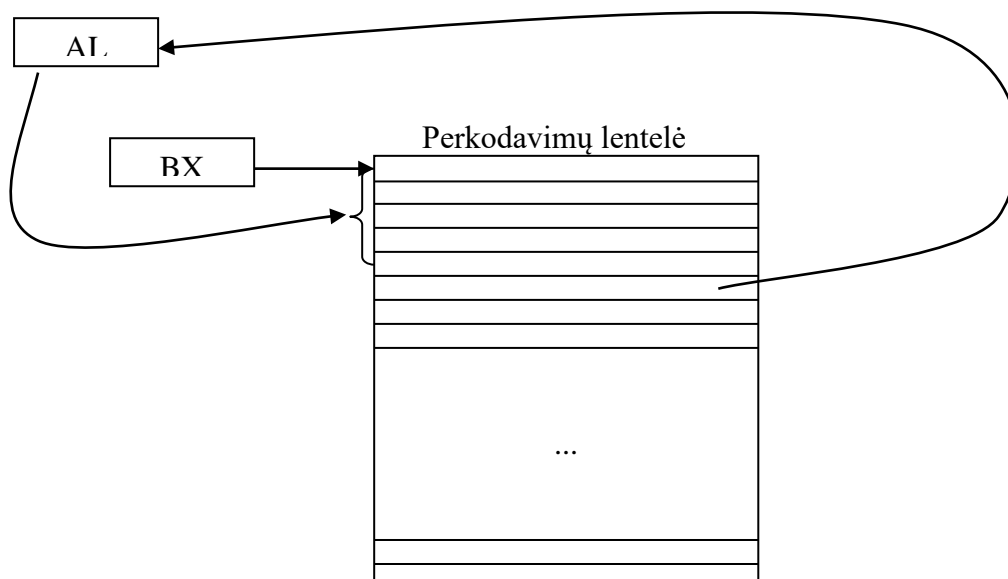
1110 110w

Porto numeris yra imamas iš registro DX, todėl galima nurodyti 65 536 skirtingus portus. Vykdam komandą, reikšmė iš porto užrašoma į registrą AL (jei  $w = 0$ ) arba į AX (jei  $w = 1$ ). Kai atliekamas veiksmas yra žodinis, tai tie 2 baitai yra imami ne iš dviejų gretimų portų, bet abu iš to paties porto, skaitant iš jo per du taktus.

**XLAT** – transLATE – kodo paėmimas pagal poslinkį.

1101 0111

Komanda naudojama duomenų perkodavimui. Registre BX esantis adresas rodo į perkodavimo lentelę. Imamas registre AL esantis baitas ir jis traktuojamas kaip poslinkis nuo perkodavimo lentelės pradžios. Vykdam komandą, reikšmė, esanti perkodavimo lentelėje su poslinkiu, nusiunčiama į registrą AL.



## KOMPIUTERIŲ ARCHITEKTŪRA

**LEA** – Load Effective Address – pakrauti (patalpinti) vykdomąjį adresą.  
1000 1101 mod reg r/m [poslinkis]

i ax adresa uzkrauname  
o su mov ax, [adresas] tai reiksme  
uzkrauname

Komanda atlieka vykdomojo (efektyvaus) adreso nusiuntimą į registrą.

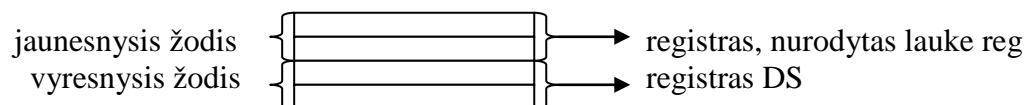
tas pats kaip offset

*reg* yra bet kuris 16 bitų registras, išskyrus segmento registrus.

Kadangi antras operandas būtinai yra atmintis, *mod* negali būti lygus 11.

**LDS** – Load pointer using DS – bendro registro ir DS užpildymas.  
1100 0101 mod reg r/m [poslinkis]

Komanda atlieka atminties dvigubo žodžio jaunesniųjų adresu dviejų baitų nusiuntimą į registrą, nurodytą lauke *reg*, ir vyresnių adresų dviejų baitų nusiuntimą į registrą DS.

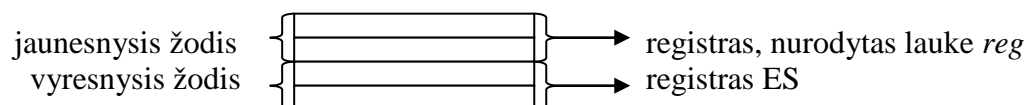


Kadangi antras operandas būtinai yra atmintis, *mod* negali būti lygus 11.

Komandos lauku *r/m* [poslinkis] nurodomas dvigubo žodžio lauką atmintyje; operando adresas suformuojamas panaudojant registrą DS : DS + *efektyvus adresas*.

**LES** – Load pointer using ES – bendro registro ir ES užpildymas.  
1100 0100 mod reg r/m [poslinkis]

Komanda LES yra panaši į komandą LDS: atlieka atminties dvigubo žodžio jaunesniųjų adresu dviejų baitų nusiuntimą į registrą, nurodytą lauke *reg*, ir vyresnių adresų dviejų baitų nusiuntimą į registrą ES.



Kadangi antras operandas būtinai yra atmintis, *mod* negali būti lygus 11.

Komandos lauku *r/m* [poslinkis] nurodomas dvigubo žodžio lauką atmintyje; operando adresas suformuojamas panaudojant registrą DS (ne registrą ES !) : DS + *efektyvus adresas*.

**LAHF** – Load AH status Flag – registro AH užpildymas registro SF jaunesniuju baitu.  
1001 1111

## KOMPIUTERIŲ ARCHITEKTŪRA

**SAHF** – Save AH status Flag – registro AH nusiuntimas į registro SF jaunesnįjį baitą.

1001 1110

Pastaba: Intel 8080 procesoriuje buvo komandos LAHF ir SAHF. Į Intel 8088 komandų sistemą jos įtrauktos tam, kad veiktų programos, kurios buvo rašytos senesniai, Intel 8080, procesoriui.

**PUSHF** – PUSH status Flag – registro SF nusiuntimas į steką.

1001 1100

**POPF** – POP status Flag – registro SF užpildymas iš steko viršūnės.

1001 1101

Tai yra vienintelė komanda, kuri pakeičia visą registre SF esančią reikšmę iš karto. Kitomis komandomis galima keisti atskirus bitukus.

### ***Aritmetinės komandos***

**ADD** – sudėtis

1. registras + registras / atmintis

0000 00dw mod reg r/m [poslinkis]

2. registras / atmintis + betarpiškas operandas

1000 00sw mod 000 r/m [poslinkis] bet.op.j.b. [bet.op.v.b]

Jeigu  $s:w = 01$ , tada betarpiškas operandas yra imamas iš 16 komandos bitų.

Jeigu  $s:w = 11$ , tada betarpiškas operandas yra imamas iš 8 komandos bitų ir išplečiamas iki 16 bitų pagal ženklo plėtimo taisyklę: jeigu vyriausiame bite buvo 1, tada išplečiame vienetais, o jeigu vyriausiame bite buvo 0, tada išplečiame nuliais.

Jeigu  $w = 0$ , tai  $s$  gali būti lygu 0 arba 1 – betarpiškas operandas bus vieno baito.

3. akumulatorius + betarpiškas operandas

0000 010w bet.op.j.b [bet.op.v.b., jei  $w = 1$ ]

**ADC** – ADd with Carry – sudėtis, įskaitant pernešimo bitą CF.

Komanda yra analogiška komandai ADD, tik papildomai pridedamas CF:

$operandas1 + operandas2 + CF$

1. registras + registras / atmintis

0001 00dw mod reg r/m [poslinkis]

2. registras / atmintis + betarpiškas operandas

1000 00sw mod 010 r/m [poslinkis] bet.op.j.b. [bet.op.v.b]

## KOMPIUTERIŲ ARCHITEKTŪRA

Jeigu  $s:w = 01$ , tada betarpiškas operandas yra imamas iš 16 komandos bitų.

Jeigu  $s:w = 11$ , tada betarpiškas operandas yra imamas iš 8 komandos bitų ir išplečiamas iki 16 bitų pagal ženklo plėtimo taisyklę: jeigu vyriausiam bite buvo 1, tada išplečiame vienetais, o jeigu vyriausiam bite buvo 0, tada išplečiame nuliais.

Jeigu  $w = 0$ , tai  $s$  gali būti lygu 0 arba 1 – betarpiškas operandas bus vieno baido.

3. akumulatorius + betarpiškas operandas  
0001 010w bet.op.j.b [bet.op.v.b., jei  $w = 1$ ]

**INC** – INCrement – didinimas vienetu.

1. registras / atmintis  
1111 111w mod 000 r/m [poslinkis]

2. registras  
01000 reg

Šia komanda inkrementuojami tik žodiniai registrai.

**SUB** – SUBtract – atimtis.

1. registras / atmintis ~ registras  
Abu operandai gali būti pirmu arba antru operandu. Rezultatas užrašomas į pirmąjį.  
0010 10dw mod reg r/m [poslinkis]

Nors atimtį realizuoja sudėtis su priešingu ženklu, bet interpretuojamo lygio komandų kodai turi skirtis, tam kad komanda sudarytų antrą operandą papildomu kodu, ir tada pritaikytų aparatūrinį sudėties veiksmą.

2. registras / atmintis - betarpiškas operandas  
1000 00sw mod 101 r/m [poslinkis] bet.op.j.b. [bet.op.v.b.]

Jeigu  $s:w = 01$ , tada betarpiškas operandas yra imamas iš 16 komandos bitų.

Jeigu  $s:w = 11$ , tada betarpiškas operandas yra imamas iš 8 komandos bitų ir išplečiamas iki 16 bitų pagal ženklo plėtimo taisyklę: jeigu vyriausiam bite buvo 1, tada išplečiame vienetais, o jeigu vyriausiam bite buvo 0, tada išplečiame nuliais.

Jeigu  $w = 0$ , tai  $s$  gali būti lygu 0 arba 1 – betarpiškas operandas bus vieno baido.

3. akumulatorius - betarpiškas operandas  
0010 110w bet.op.j.b. [bet.op.v.b.,  $w = 1$ ]

**SBB** – SuBtraction with Borrow – atimtis su pasiskolinimu

Komanda yra analogiška komandai SUB, tik papildomai atimamas CF:

*operandas1 - operandas2 – CF*

## KOMPIUTERIŲ ARCHITEKTŪRA

1. registras / atmintis ~ registras

Abu operandai gali būti pirmu arba antru operandu. Rezultatas užrašomas į pirmąjį.

0001 10dw mod reg r/m [poslinkis]

2. registras / atmintis - betarpiškas operandas

1000 00sw mod 011 r/m [poslinkis] bet.op.j.b. [bet.op.v.b]

Jeigu  $s:w = 01$ , tada betarpiškas operandas yra imamas iš 16 komandos bitų.

Jeigu  $s:w = 11$ , tada betarpiškas operandas yra imamas iš 8 komandos bitų ir išplečiamas iki 16 bitų pagal ženklų plėtimo taisyklę: jeigu vyriausiam bite buvo 1, tada išplečiame vienetais, o jeigu vyriausiam bite buvo 0, tada išplečiame nuliais.

Jeigu  $w = 0$ , tai  $s$  gali būti lygu 0 arba 1 – betarpiškas operandas bus vieno baido.

3. akumulatorius - betarpiškas operandas

0001 110w bet.op.j.b. [bet.op.v.b.,  $w = 1$ ]

**DEC** – DECrement – sumažinimas vienetu.

1. registras / atmintis

1111 111w mod 001 r/m [poslinkis]

2. registras

01001 reg

Šia komanda dekrementuojami tik žodiniai registrai.

**CMP** – CoMPare – palyginimas.

Komanda naudojama požymių formavimui, veikia panašiai kaip ir komanda SUB, tačiau neįsimena rezultato.

Požymių formavimas:

Jeigu operandai yra skaičiai be ženklų:

	<b>OF</b>	<b>SF</b>	<b>ZF</b>	<b>CF</b>
$op.1 > op.2$	N	N	0	0
$op.1 = op.2$	N	N	1	0
$op.1 < op.2$	N	N	0	1

op. – operandas;

N – nesvarbu.

Jeigu operandai yra skaičiai su ženklu:

## KOMPIUTERIŲ ARCHITEKTŪRA

	<b>OF</b>	<b>SF</b>	<b>ZF</b>	<b>CF</b>
op.1 > op.2	0 / 1	0 / 1	0	N
op.1 = op.2	0	0	1	N
op.1 < op.2	0 / 1	0 / 1	0	N

op. – operandas;

N – nesvarbu.

0 / 1 – 0 arba 1, priklausomai, nuo konkrečių lyginamų reikšmių.

Skaičių su ženklu palyginimas, diapazone -128 – 127 :

	<b>SF</b>	<b>OF</b>
$0 < A - B \leq 127$	0	0
$A - B > 127$	1	1
$-128 \leq A - B < 0$	1	0
$A - B < -128$	0	1

Taigi  $A > B$ , kai :  $\left. \begin{array}{l} 1. \text{ OF} = 0; \text{ SF} = 0 \\ 2. \text{ OF} = 1; \text{ SF} = 1 \end{array} \right\}$  ,t.y., kai OF lygu SF

$A < B$ , kai :  $\left. \begin{array}{l} 1. \text{ OF} = 1; \text{ SF} = 0 \\ 2. \text{ OF} = 0; \text{ SF} = 1 \end{array} \right\}$  ,t.y., kai OF nelygu SF

Algoritmas požymiui OF nustatyti:

jeigu sudėties operacijos atžvilgiu operandų ženklo bitai skiriasi arba operandų ir rezultato ženklo bitai yra vienodi, tai OF = 0;

jeigu operandų ženklo bitai yra vienodi, bet rezultato ženklo bitas skiriasi, tai OF = 1.

Pavyzdžiai:

1.  $10 - 5$

$(10)_{10} = (1010)_2$

$(5)_{10} = (101)_2,$

$$\begin{array}{r} 0000\ 0101 \\ 1111\ 1010 \\ + \quad \quad 1 \\ \hline 1111\ 1011 \end{array} = (-5)_{10}$$

$$\begin{array}{r} 0000\ 1010 \\ + 1111\ 1011 \\ \hline 10000\ 0101 \end{array} \quad \text{OF} = 0, \text{ SF} = 0$$



## KOMPIUTERIŲ ARCHITEKTŪRA

2.  $100 - (-101)$

$$(100)_{10} = (1100100)_2$$

$$(101)_{10} = (1100101)_2$$

$$\begin{array}{r} 0110\ 0100 \\ + \underline{0110\ 0101} \\ 1100\ 1001 \end{array} \quad \text{OF} = 1, \text{SF} = 1$$

3.  $-3 - (7)$

$$(3)_{10} = (11)_2,$$

$$\begin{array}{r} 0000\ 0011 \\ 1111\ 1100 \\ + \underline{\phantom{0000}\phantom{0000}1} \\ 1111\ 1101 \end{array} = (-3)_{10}$$

$$(7)_{10} = (111)_2,$$

$$\begin{array}{r} 0000\ 0111 \\ 1111\ 1000 \\ + \underline{\phantom{0000}\phantom{0000}1} \\ 1111\ 1001 \end{array} = (-7)_{10}$$

$$\begin{array}{r} 1111\ 1101 \\ + \underline{1111\ 1001} \\ 1111\ 0110 \end{array} \quad \text{OF} = 0, \text{SF} = 1$$

4.  $-101 - (100)$

$$(101)_{10} = (1100101)_2$$

$$\begin{array}{r} 0110\ 0101 \\ 1001\ 1010 \\ + \underline{\phantom{0000}\phantom{0000}1} \\ 1001\ 1011 \end{array} = (-101)_{10}$$

$$(100)_{10} = (1100100)_2$$

$$\begin{array}{r} 0110\ 0100 \\ 1001\ 1011 \\ + \underline{\phantom{0000}\phantom{0000}1} \\ 1001\ 1100 \end{array} = (-100)_{10}$$

$$\begin{array}{r} 1001\ 1011 \\ + \underline{1001\ 1100} \\ 10011\ 0111 \end{array} \quad \text{OF} = 1, \text{SF} = 0$$

1. registras / atmintis ~ registras

Abu operandai gali būti pirmu arba antru operandu.

0011 10dw mod reg r/m [poslinkis]

## KOMPIUTERIŲ ARCHITEKTŪRA

### 2. registras / atmintis - betarpiškas operandas

1000 00sw mod 111 r/m [poslinkis] bet.op.j.b. [bet.op.v.b.]

Jeigu  $s:w = 01$ , tada betarpiškas operandas yra imamas iš 16 komandos bitų.

Jeigu  $s:w = 11$ , tada betarpiškas operandas yra imamas iš 8 komandos bitų ir išplečiamas iki 16 bitų pagal ženklo plėtimo taisyklę: jeigu vyriausiam bite buvo 1, tada išplečiame vienetais, o jeigu vyriausiam bite buvo 0, tada išplečiame nuliais.

Jeigu  $w = 0$ , tai  $s$  gali būti lygu 0 arba 1 – betarpiškas operandas bus vieno baido.

### 3. akumulatorius - betarpiškas operandas

0011 110w bet.op.j.b. [bet.op.v.b.,  $w = 1$ ]

### **MUL** – MULtiPLY – skaičių be ženklo daugyba

1111 011w mod 100 r/m [poslinkis]

Jeigu dauginami baitai, tai:

*registre AL esanti reikšmė · operandas 2 → registras AX*

Jeigu dauginami žodžiai, tai:

*registre AX esanti reikšmė · operandas 2 → registrai [DX, AX]*

Jeigu daugybos rezultato vyresnioji pusė yra nulinė, tada  $CF = 0$  ir  $OF = 0$ , priešingu atveju  $CF = 1$  ir  $OF = 1$ .

### **IMUL** – Integer MULtiPLY – skaičių su ženklu daugyba

1111 011w mod 101 r/m [poslinkis]

Jeigu dauginami baitai, tai:

*registre AL esanti reikšmė · operandas 2 → registras AX*

Jeigu dauginami žodžiai, tai:

*registre AX esanti reikšmė · operandas 2 → registrai [DX, AX]*

Jeigu daugybos rezultato vyresniosios pusės kiekvieno bito reikšmė lygi jaunesniosios pusės vyriausiam bitui, t.y., daugybos rezultato vyresnioji pusė yra tik jaunesniosios pusės ženklo išplėtimas, tada rezultatas tilpo jaunesniojoje pusėje, todėl  $CF = 0$  bei  $OF = 0$ . Priešingu atveju,  $CF = 1$  ir  $OF = 1$ .

### **DIV** – DIVide – skaičių be ženklo dalyba.

1111 011w mod 110 r/m [poslinkis]

Jeigu  $w = 0$ , tada daliklis yra baitas, ir:

*registre AX esanti reikšmė : operandas 2 → dalmuo registre AL  
→ liekana registre AH*

Jeigu  $w = 1$ , tada daliklis yra žodis, ir:

## KOMPIUTERIŲ ARCHITEKTŪRA

*registruose[DX, AX] esanti reikšmė : operandas 2 → dalmuo registre AX  
→ liekana registre DX*

Jeigu dalmuo netelpa registre AL (kai  $w = 0$ ) arba AX (kai  $w = 1$ ), tada generuojamas pertraukimas – dalyba iš nulio.

**IDIV** – Integer DIVide – skaičių su ženklu dalyba.

1111 011w mod 111 r/m [poslinkis]

Jeigu  $w = 0$ , tada daliklis yra baitas, ir:

*registre AX esanti reikšmė : operandas 2 → dalmuo registre AL  
→ liekana registre AH*

Jeigu  $w = 1$ , tada daliklis yra žodis, ir:

*registruose[DX, AX] esanti reikšmė : operandas 2 → dalmuo registre AX  
→ liekana registre DX*

Dalybos iš nulio pertraukimas kyla tada, kai dalmuo-baitas netelpa diapazone: -128 – 127, arba dalmuo-žodis netelpa diapazone: -32768 – 32767.

**NEG** – NEGative – ženklo keitimas

1111 011w mod 011 r/m [poslinkis]

Rezultatas yra analogiškas rezultatui, kurį gautume atlikę tokį veiksmą:

0 - *operandas* → *operandas*

Faktiškai, pritaikius komandą NEG, operandą gauname papildomu kodu.

### ***Veiksmai su dešimtainio formato skaičiais***

Tai yra veiksmų, atliktų bendrosiomis komandomis patikslinimas, kad rezultatas būtų korektiškas.

Pastaba. Požymis AF fiksuoja pernešimą iš jaunesnio pusbaičio, o požymis CF fiksuoja pernešimą iš baito.

**AAA** – Ascii Adjust for Addition

0011 0111

ASCII tai yra simbolinis kodavimas, kuriam yra ekvivalentūs dešimtainio nesupakuoto formato skaičiai, nes vienas toks skaičius užima vieną baitą.

Komanda yra koreguojama reikšmė, esanti registre AL, kuri pertvarkoma į teisingą dešimtainį nesupakuotą skaičių, kurios vyresnis pusbaitis yra 0h, o jaunesniajame pusbaityje yra skaičiai iš diapazono 0h – 9h.

## KOMPIUTERIŲ ARCHITEKTŪRA

Jeigu korekcijos rezultatas yra didesnis už 9h, nėra kaip to užfiksuoti dešimtainiu nesupakuotu skaičiumi, todėl komanda AAA suformuoja požymį CF = 1 ir registre AH esančią reikšmę padidina vienetu.

Komandos AAA veikimas priklauso nuo registro SF požymio AF.

Komandos AAA veikimo algoritmas:

```
if ( (AL and 0Fh) > 9 or (AF = 1) ) then
    AL := AL + 6
    AH := AH + 1
    AF := 1
    CF := 1
else
    AF := 0
    CF := 0
endif
AL := AL and 0Fh
```

Pavyzdžiai:

1.  $3 + 7 = 0A \rightarrow 10 \rightarrow 00$

Komandos AAA veikimo rezultate, skaičiaus A reikšmė yra padidinama šešiais dėl pagrindų skirtumo: sudedant pozicijos svoris yra 16, o norime gauti reikšmę, kurios pozicijos svoris yra 10.

Tačiau teisingas dešimtainis nesupakuotas skaičius vyresniame pusbaityje turi turėti nulį, todėl rezultatas yra dar koreguojamas iki 00. Gautas rezultatas buvo didesnis už 9, todėl nustatomas požymis CF = 1, ir registre AH esanti reikšmė padidinama vienetu.

2.  $8 + 9 = 11 \rightarrow 17 \rightarrow 07$

1-ame pavyzdyje koregavome rezultatą 0A, požymis AF buvo lygus 0. Tarkime, kad po sudėties gavome 11, tada papildomo pernešimo požymis AF nustatomas lygus 1.

Skaičių 11 padidiname šešiais ir gauname 17. Kadangi vyresniame pusbaityje turi būti 0, tai ši reikšmė pakoreguojama į 07. Nustatomas požymis CF = 1, registre AH esanti reikšmė padidinama vienetu, nustatomas požymis AF=1.

### **DAA – Decimal Adjust for Addition**

0010 0111

Komanda yra koreguojama reikšmė, esanti registre AL, kuri pertvarkoma į teisingą dešimtainį supakuotą skaičių diapazone. Jeigu po korekcijos rezultatas viršija 99, tai nustatomas požymis CF = 1

Komandos DAA veikimo algoritmas:

```
if ( (AL and 0Fh) > 9 or (AF = 1) ) then
    AL := AL + 6
    AF := 1
endif
```

## KOMPIUTERIŲ ARCHITEKTŪRA

```
if ( (AL > 9Fh) or (CF = 1) ) then
    AL := AL + 60h
    CF := 1
endif
```

### Pavyzdžiai:

1.  $24 + 27 = 4B \rightarrow 51$

Sudėties rezultatą 4B gauname naudodami esamą sudėties komandą. Komanda DAA prideda 6, ir turime korektišką dešimtainį supakuotą skaičių-rezultatą: 51.

2.  $29 + 28 = 51 \rightarrow 57$

51 būtų teisingas dešimtainis supakuotas skaičius, bet atliekant sudėtį buvo fiksuotas pernešimas į vyresnį pusbaitį: požymis AF = 1, todėl yra pritaikoma komanda DAA.

3.  $62 + 64 = C6 \rightarrow 26$

Koreguojant sudėties rezultatą C6, gauname 126, o tai yra daugiau už 99 ir viršija dešimtainių supakuotų skaičių diapazoną. Todėl yra nustatomas požymis CF = 1.

4.  $66 + 65 = CB \rightarrow D1 \rightarrow 31$

Koreguojant sudėties rezultatą CB, gauname D1, o tai irgi yra neteisingas dešimtainis skaičius, todėl komanda DAA koreguoja dar kartą, taip gaunamas 131, o tai yra daugiau už 99 ir viršija dešimtainių supakuotų skaičių diapazoną. Todėl yra nustatomas požymis CF = 1.

5.  $46 + 55 = 9B \rightarrow A1 \rightarrow 01$

Koreguojant sudėties rezultatą 9B, gauname A1, o tai yra vis dar neteisingas dešimtainis skaičius, todėl komanda DAA koreguoja dar kartą, taip gaunamas 101, o tai yra daugiau už 99 ir viršija dešimtainių supakuotų skaičių diapazoną. Todėl yra nustatomas požymis CF = 1.

### **AAS – Ascii Adjust for Substraction**

0011 1111

Komanda yra koreguojama reikšmė, esanti registre AL, kuri pertvarkoma į teisingą dešimtainį nesupakuotą skaičių, kurios vyresnis pusbaitis yra 0h, o jaunesniajame pusbaityje yra skaičiai iš diapazono 0h – 9h. Jeigu korekcijos rezultatas yra daugiau už 9, tai komanda AAS registro AH reikšmę sumažina vienetu ir nustato požymį CF = 1, priešingu atveju, CF = 0.

Atimties, kurios rezultatą koreguojame komanda AAS, operandai yra iš diapazono 0 – 9, todėl po atimties turime :

1. rezultatas  $\geq 0$ :  $0 \leq \text{rezultatas} \leq 9$

Korekcija nereikalinga.

2. rezultatas  $< 0$ :  $-9 \leq \text{rezultatas} < 0$

$-9 = F7 \leq \text{rezultatas} \leq FF = -1$

$F7 \leq \text{rezultatas} \leq FF$

$\quad \quad \quad \underline{\quad -6 \quad}$

$F1 \leq \text{rezultatas} \leq F9$ , AH sumažinama vienetu.

## KOMPIUTERIŲ ARCHITEKTŪRA

### Komandos AAS veikimo algoritmas:

```
if ( (AL and 0Fh) > 9 or (AF = 1) ) then
    AL := AL - 6
    AH := AH - 1
    AF := 1
    CF := 1
else
    AF := 0
    CF := 0
endif
AL := AL and 0Fh
```

### Pavyzdys: 5 - 8

$(5)_{10} = (101)_2$ ,

$(8)_{10} = (1000)_2$

$$\begin{array}{r} 0000\ 1000 \\ 1111\ 0111 \\ + \quad \quad 1 \\ \hline 1111\ 1000 \end{array} = (-8)_{10}$$

$$\begin{array}{r} 0000\ 0101 \\ + 1111\ 1000 \\ \hline 1111\ 1101 \end{array} = (FD)_{16}$$

$(1)5 - 8 = 7$

$FD \rightarrow F7 \rightarrow 07$

Rezultatui FD reikalinga korekcija, atimama 6. Nesupakuoto dešimtainio skaičiaus vyresniame pusbaityje turi būti nulis, todėl F7 koreguojame ir turime rezultatą 7. Tokios korekcijos eigoje komanda AAS vienetu sumažino registre AH esančią reikšmę.

### **DAS – Decimal Adjust for Subtraction**

0010 1111

Komanda yra koreguojama reikšmė, esanti registre AL, kuri pertvarkoma į teisingą dešimtainį supakuotą skaičių diapazone. Jeigu po korekcijos rezultatas viršija 99, tai komanda DAS nustato požymį CF = 1, priešingu atveju, CF = 0.

Atimties, kurios rezultatą koreguojame komanda DAS, operandai yra iš diapazono 0 – 99. Po atimties gauname reikšmę xy ir tikriname:

1. Ar  $AF = 1$ ?
2. Ar  $CF = 1$ ?
3. Ar x yra teisingas dešimtainis skaitmuo?
4. Ar y yra teisingas dešimtainis skaitmuo?

Tada elgiamasi taip:

1. Jeigu  $AF = 1$  arba y yra neteisingas dešimtainis skaitmuo, tai  $xy = xy - 6$ .
2. Jeigu  $CF = 1$  arba x yra neteisingas dešimtainis skaitmuo, tai  $xy = xy - 60$ .

## KOMPIUTERIŲ ARCHITEKTŪRA

Pavyzdys: 11 - 19

Ši kartą atlikime skaičiavimus šešioliktainėje sistemoje:

$$\begin{array}{r} 11 \\ - 19 \\ \hline F8 \\ - 6 \\ \hline F2 \\ - 60 \\ \hline 92 \end{array}$$

(1)  $11 - 19 = 92$

$F8 \rightarrow F2 \rightarrow 92$

Skaičiuojant rezultatą F8 nustatytas požymis AF = 1, todėl jį koreguojame atimdami 6, gauname F2. F yra neteisingas dešimtainis skaičius, o ir CF nustatytas lygus vienam, todėl atimame 60, gauname 92.

### Komandos DAS veikimo algoritmas:

```
if ( (AL and 0Fh) > 9 or (AF = 1) ) then
    AL := AL - 6
    AF := 1
endif
if ( (AL > 99h) or (CF = 1) ) then
    AL := AL - 60h
    CF := 1
endif
```

### **AAM – Ascii Adjust for Multiplication**

1101 0100 0000 1010

Komanda yra taikoma po dviejų dešimtainių nesupakuotų skaičių baitinės daugybos, kai sandauga yra registre AX, o šiuo atveju, t.y., kai yra dauginami du dešimtainiai nesupakuoti skaičiai, rezultatas telpa registre AL (maksimalus rezultatas yra  $9 \cdot 9 = 81$ ).

Komanda AAM padalina registre AL esančią reikšmę iš 10 (dešimtainis), tada dalmenį užrašo į registrą AH, o liekaną – į registrą AL:

```
    AH := AL div 10
    AL := AL mod 10
```

Taip du skaičiai yra atskiriami, ir registruose AH ir AL turime teisingus dešimtainius nesupakuotus skaičius, kai vyresniame pusbaityje yra nulis, o jaunesniame pusbaityje yra skaitmuo iš diapazono: 0h – 9h.

Pavyzdys: jeigu daugybos rezultate gauname 79 (dešimtainis), tai po komandos AAM suveikimo, registre AH yra 7, o registre AL yra 9.

Pastaba. Supakuotiems dešimtainiams skaičiams nėra daugybos korekcijos komandos.

## KOMPIUTERIŲ ARCHITEKTŪRA

### **AAD – Ascii Adjust for Division**

1101 0101 0000 1010

Komanda atlieka dalybos rezultato korekciją dešimtainiams nesupakuotiems skaičiams. Ši korekcijos komanda skiriasi nuo kitų korekcijos komandų tuo, kad ji yra taikoma ne rezultatui, o paruošia duomenis dešimtainių nesupakuotų skaičių dalybai.

Dešimtainių nesupakuotų skaičių dalybos korekcijos komanda taria, kad dalinys yra registre AX kaip dviejų baitų dešimtainis nesupakuotas skaičius, kurio vyresnis skaitmuo yra registre AH, kurio vyresnis pusbaitis yra nulis.

Komanda atlieka tokius veiksmus:

AL := AH \* 10 + AL

AH := 0

Tada yra atliekama dalybos komanda DIV.

Pastaba. Supakuotiems dešimtainiams skaičiams nėra dalybos korekcijos komandos.

## ***Konvertavimas***

### **CBW – Convert Byte to Word**

1001 1000

Ši komanda yra taikoma skaičiams su ženklu, tokiems kurie yra baito diapazone, t.y., -128 – 127. Registro AL vyriausiasis bitas yra išskleidžiamas į registrą AH. Taip reikšmė baite yra pervedama į reikšmę žodyje.

### **CWD – Convert Word to Double word**

1001 1001

Ši komanda yra taikoma skaičiams su ženklu, tokiems, kurie yra dviejų baitų – žodžio diapazone, t.y., -32768 – 32767. Registro AX vyriausiasis bitas yra išskleidžiamas į registrą DX. Taip reikšmė žodyje yra pervedama į reikšmę dvigubame žodyje.

Komanda gali būti naudojama, pvz., dalybos veiksmo IDIV paruošimui, nes žodyje esančių reikšmių dalyba reikalauja, kad pirmas operandas būtų keturiuose baituose, t.y., registruose [DX, AX].



## ***Loginės komandos***

**NOT** – loginis paneigimas.

1111 011w mod 010 r/m [poslinkis]

Vykdant komandą, lauko bitai yra invertuojami.

**SHL / SAL** – SHift logical Left / Shift Arithmetic Left – loginis postūmis į kairę / aritmetinis postūmis į kairę.

1101 00vw mod 100 r/m [poslinkis]

Jeigu  $v = 0$ , tai pastumiama į kairę per vieną poziciją.

Jeigu  $v = 1$ , tai postūmio skaitliukas yra registro CL reikšmė.

Dešinieji (jaunesnieji) atsilaisvinę bitai yra užpildomi nuliais.

Jeigu postūmyje per vieną bitą ženklo bitas nekeičia reikšmės, tai  $OF = 0$ , priešingu atveju,  $OF = 1$ .

Jeigu postūmis yra per kelis bitus, tada požymis OF yra neapibrėžtas.

**SHR** – SHift logical Right – loginis postūmis į dešinę.

1101 00vw mod 101 r/m [polinkis]

Jeigu  $v = 0$ , tai pastumiama į dešinę per vieną poziciją.

Jeigu  $v = 1$ , tai postūmio skaitliukas yra registro CL reikšmė.

Vyresnieji atsilaisvinę bitai yra užpildomi nuliais, kiekvienam išstumtam bitui.

Jeigu postūmyje per vieną bitą vyriausias bitas nekeičia reikšmės, tai  $OF = 0$ , priešingu atveju,  $OF = 1$ .

Jeigu postūmis yra per kelis bitus, tada požymis OF yra neapibrėžtas.

**SAR** – Shift Arithmetic Right – aritmetinis postūmis į dešinę.

1101 00vw mod 111 r/m [poslinkis]

Jeigu  $v = 0$ , tai pastumiama į dešinę per vieną poziciją.

Jeigu  $v = 1$ , tai postūmio skaitliukas yra registro CL reikšmė.

Vyresnieji atsilaisvinę bitai yra užpildomi nuliais arba vienetais, taip, kad būtų išsaugotas ženklas.

Pastaba. Postūmyje į dešinę loginis ir aritmetinis postūmis nėra tas pats, nes loginiame atsilaisvinę, atsiradę išstumtųjų vietoje, bitai yra užpildomi nuliais, o aritmetiniame postūmyje atsilaisvinę bitai yra užpildomi ženklo bitais.

## KOMPIUTERIŲ ARCHITEKTŪRA

**ROL** – ROtate Left – ciklinis postūmis į kairę.

1101 00vw mod 000 r/m [poslinkis]

Jeigu  $v = 0$ , tai pastumiama į kairę per vieną poziciją.

Jeigu  $v = 1$ , tai postūmio skaitliukas yra registro CL reikšmė.

Stumiant per vieną bitą į kairę, vyriausias bitas užpildo jauniausio bito reikšmę. Jeigu postūmis yra daugiau nei per vieną poziciją, tai atitinkamai daugiau vyresniųjų bitų pernešama į jaunesnius.

Jeigu postūmyje per vieną bitą vyriausias bitas nekeičia reikšmės, tai  $OF = 0$ , priešingu atveju,  $OF = 1$ .

Vyriausias bitas užrašomas į požymį CF.

**ROR** – ROtate Right – ciklinis postūmis į dešinę.

1101 00vw mod 001 r/m [poslinkis]

Jeigu  $v = 0$ , tai pastumiama į dešinę per vieną poziciją.

Jeigu  $v = 1$ , tai postūmio skaitliukas yra registro CL reikšmė.

Stumiant per vieną bitą į dešinę, jauniausias bitas užpildo vyriausio bito reikšmę. Jeigu postūmis yra daugiau nei per vieną poziciją, tai atitinkamai daugiau jaunesniųjų bitų pernešama į vyresnius.

Jeigu postūmyje per vieną bitą vyriausias bitas nekeičia reikšmės, tai  $OF = 0$ , priešingu atveju,  $OF = 1$ .

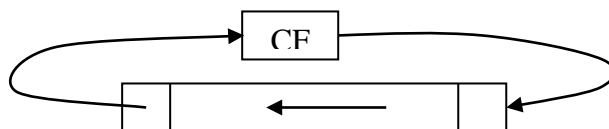
Jauniausias bitas užrašomas į požymį CF.

**RCL** – Rotate Left through Carry – perstumti cikliška į kairę, panaudojant pernešimo bitą CF.

1101 00vw mod 010 r/m [poslinkis]

Jeigu  $v = 0$ , tai pastumiama į kairę per vieną poziciją.

Jeigu  $v = 1$ , tai postūmio skaitliukas yra registro CL reikšmė.



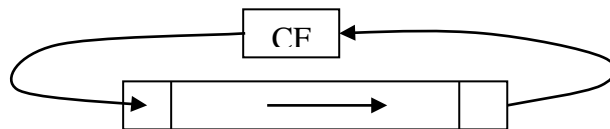
## KOMPIUTERIŲ ARCHITEKTŪRA

**RCR** – Rotate Right through Carry – perstumti cikliška į dešinę, panaudojant pernešimo bitą CF.

1101 00vw mod 011 r/m [poslinkis]

Jeigu  $v = 0$ , tai pastumiama į dešinę per vieną poziciją.

Jeigu  $v = 1$ , tai postūmio skaitliukas yra registro CL reikšmė.



**AND** – loginis “ir”.

Komanda naudojama formuoti nuliniams laukams.

1. registras / atmintis  $\wedge$  registras

0010 00dw mod reg r/m [poslinkis]

2. betarpiškas operandas  $\wedge$  registras / atmintis

1000 00sw mod 100 r/m [poslinkis] bet.op.j.b [bet.op.v.b.]

Jeigu  $s:w = 01$ , tada betarpiškas operandas yra imamas iš 16 komandos bitų.

Jeigu  $s:w = 11$ , tada betarpiškas operandas yra imamas iš 8 komandos bitų ir išplečiamas iki 16 bitų – žodžio, pagal ženkle plėtimo taisyklę.

3. betarpiškas operandas  $\wedge$  akumuliatorius

0010 010w bet.op.j.b. [bet.op.v.b., jei  $w = 1$ ]

**OR** – loginis “arba”.

Komanda naudojama formuoti vienetiniams laukams.

1. registras / atmintis  $\vee$  registras

0000 10dw mod reg r/m [poslinkis]

2. betarpiškas operandas  $\vee$  registras / atmintis

1000 00sw mod 001 r/m [poslinkis] bet.op.j.b [bet.op.v.b.]

Jeigu  $s:w = 01$ , tada betarpiškas operandas yra imamas iš 16 komandos bitų.

Jeigu  $s:w = 11$ , tada betarpiškas operandas yra imamas iš 8 komandos bitų ir išplečiamas iki 16 bitų – žodžio.

3. betarpiškas operandas  $\vee$  akumuliatorius

0000 110w bet.op.j.b. [bet.op.v.b., jei  $w = 1$ ]

## KOMPIUTERIŲ ARCHITEKTŪRA

**XOR** – eXclusive OR – išimtinis “arba”

Operacijoje su vienetiniu lauku, komanda operando bitus keičia į priešingus.  
Operacijoje su nuliniu lauku, operando lauko bitai yra išsaugomi.

1. registras / atmintis  $\wedge$  registras  
0011 00dw mod reg r/m [poslinkis]

2. betarpiškas operandas  $\wedge$  registras / atmintis  
1000 00sw mod 110 r/m [poslinkis] bet.op.j.b [bet.op.v.b.]

Jeigu  $s:w = 01$ , tada betarpiškas operandas yra imamas iš 16 komandos bitų.

Jeigu  $s:w = 11$ , tada betarpiškas operandas yra imamas iš 8 komandos bitų ir išplečiamas iki 16 bitų – žodžio.

3. betarpiškas operandas  $\wedge$  akumuliatorius  
0011 010w bet.op.j.b. [bet.op.v.b., jei  $w = 1$ ]

**TEST** – patikrinimas

Atliekama operacija, analogiška operacijai AND, tačiau rezultatas nėra išsaugomas, o formuojami požymių bitai:

1. CF = 0 ir OF = 0;
2. AF – neapibrėžtas;
3. PF, SF, ZF – jų reikšmės keičiamos pagal rezultato reikšmę.

Komanda yra naudojama prieš valdymo perdavimo komandas.

1. registras / atmintis  $\sim\sim$  registras  
1000 010w mod reg r/m [poslinkis]

2. betarpiškas operandas  $\sim\sim$  registras / atmintis  
1111 011w mod 000 r/m [poslinkis] bet.op.j.b [bet.op.v.b., jei  $w = 1$ ]

3. betarpiškas operandas  $\sim\sim$  akumuliatorius  
1010 100w bet.op.j.b. [bet.op.v.b., jei  $w = 1$ ]

### ***Eilutinės komandos***

**REP** – REPeate – pakartoti (pakartojimo prefiksas, priedas prie eilutinės komandos).  
Pakartojimo prefiksas sąlygoja sekančios komandos pakartojimą, jeigu  $ZF = 1$  (t.y., yra nulis arba lygu, pagal situaciją) ir registre CX esanti reikšmė nelygi nuliui.

Sinonimai: **REP** / **REPZ** / **REPE** – REPeate / REPeate if Zero / REPeate if Equal – pakartoti / pakartoti, jei yra nulis / pakartoti, jei lygu.

1111 0011

z bitas, kuris yra lygus 1,  $ZF = z$

## KOMPIUTERIŲ ARCHITEKTŪRA

Registro CX reikšmę keičia eilutinė komanda.

Komanda su pakartojimo prefiksu, kuris iššaukia ciklinį vykdymą: REP *komanda* .

REP *komanda* vykdymas gali būti pertrauktas. Pertraukimas įsimena tik vieną prefiksą ir jį atstato. Prefiksas REP turi būti betarpiškai prieš komandą, kuri yra kartojama. Jeigu yra panaudoti papildomai vienas arba du prefiksai (segmento keitimo, magistralės blokavimo), tai po pertraukimo šie prefiksai nebeatstatomi, ir pratęsus vykdymą rezultatai gali būti nekorektiški. Todėl prieš tokį prefiksinį veiksmą su pakartojimu reikia uždrausti pertraukimus, o atlikus veiksmą vėl juos leisti. Kita vertus, gali gautis neleistinai ilgas laiko tarpas, kai pertraukimai neleidžiami.

Pastaba. Prefikso bitas z neturi įtakos eilutinėms komandoms MOVS, LODS, STOS. Neturi įtakos ir neeilutinėms komandoms, formaliai prieš jas pritaikius pakartojimo prefiksą, tačiau tos komandos nemažina registre CX esančios reikšmės, todėl tokia pora – pakartojimo prefiksas ir neeilutinė komanda, maža turi prasmės.

**REPZ / REPNE** – REPeate if Not Zero / REPeate if Not Equal – pakartoti, jei yra ne nulis / pakartoti, jei nelygu .

Pakartojimo prefiksas sąlygoja sekančios komandos pakartojimą, jeigu ZF = 0 (t.y., yra ne nulis arba nelygu, pagal situaciją) ir registre CX esanti reikšmė nelygi nuliui.

1111 0010



z bitas, kuris yra lygus 0, ZF = z

Registro CX reikšmę keičia eilutinė komanda.

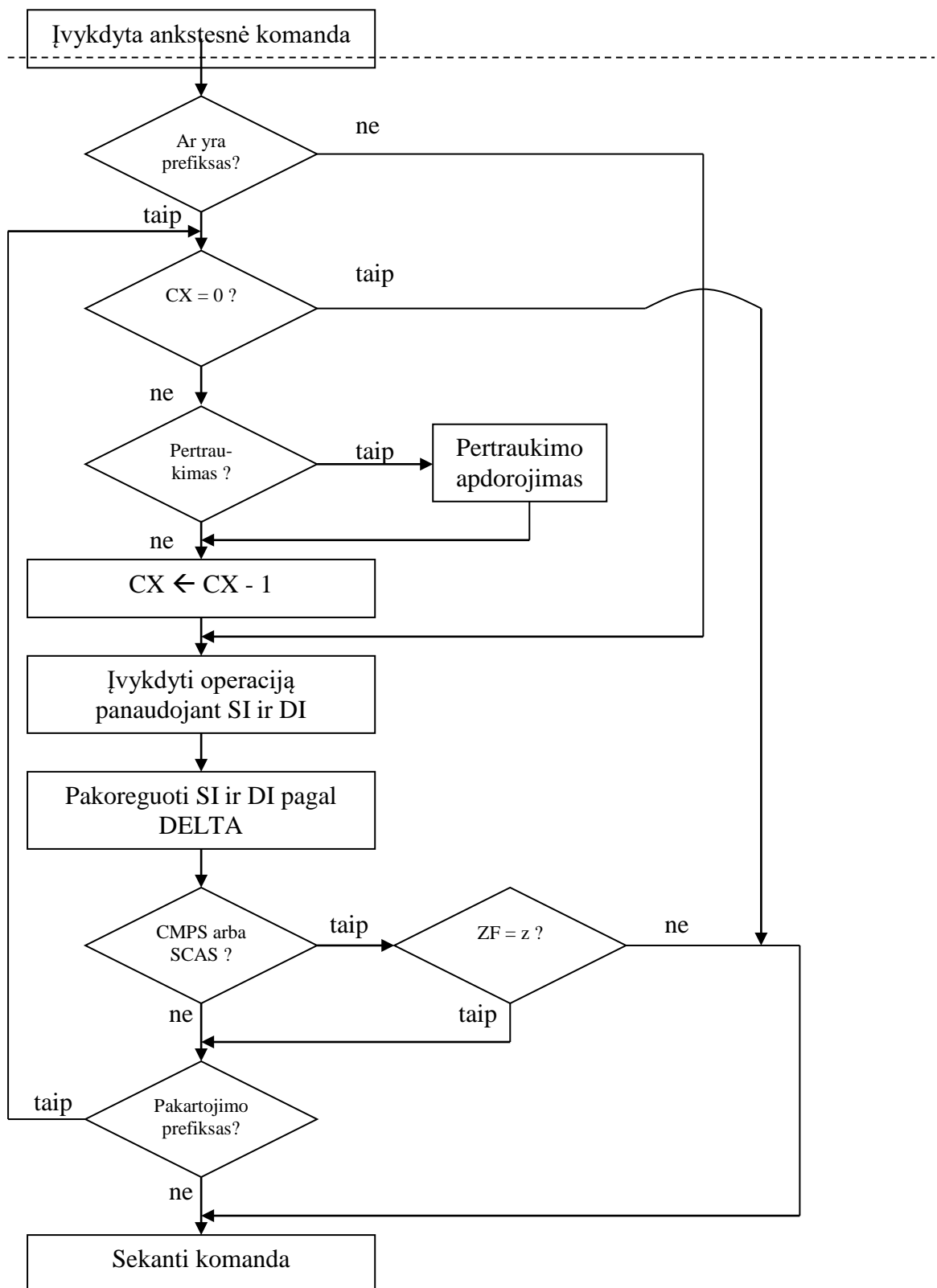
Pastaba. Pakartojimo prefikso bitas z neturi reikšmės komandoms MOVS, LODS, STOS, o tik toms eilutinėms komandoms, kuriomis atliekamas lyginimas arba skanavimas.

***Eilutinės komandos su pakartojimo prefiksu įvykdymo schema:***

Pastaba. Veiksmą:  $CX \leftarrow CX - 1$  atlieka eilutinė komanda, jeigu ji yra su pakartojimo prefiksu.

Pastaba. Pakartojimo prefiksas gali būti prieš bet kokią komandą.

	DF	DELTA
Baitas	0	+ 1
Baitas	1	- 1
Žodis	0	+ 2
Žodis	1	- 2
<b>Prefiksas</b>	<b>z</b>	
REP / REPE / REPZ	1	
REPNE / REPZ	0	



## KOMPIUTERIŲ ARCHITEKTŪRA

**MOVS** – MOVe String – eilutės persiuntimas.

1010 010w

Komanda yra persiunčiamas baitas arba žodis.

Jeigu  $DF = 0$ , tai indeksiniai registrai SI ir DI yra padidinami baitu (padidinama vienetu) arba žodžiu (padidinama dviem).

Jeigu  $DF = 1$ , tai indeksiniai registrai SI ir DI yra sumažinami baitu (sumažinama vienetu) arba žodžiu (sumažinama dviem).

Šaltinio lauko adresą nurodo registras SI, rezultato lauko adresą nurodo registras DI. Adresai yra segmento viduje (ribose).

Absoliutus rezultato lauko adresas suformuojamas naudojant registrus ES ir DI. Absoliutus šaltinio adresas gali būti formuojamas pasirinktinai naudojant registrus: DS, CS, SS arba ES, ir SI.

**CMPS** – CoMPare String – eilučių palyginimas.

1010 011w

Vienu palyginimo veiksmu yra lyginami baitai arba žodžiai, iš šaltinio lauko atimant rezultato lauką. Suformuojami sąlygos požymiai, operandai nėra keičiami, tačiau modifikuojamos registrų SI ir DI reikšmės, pagal indeksinių registrų modifikavimo taisyklę – priklausomai nuo požymio DF (aprašyta prie komandos MOVS).

**SCAS** – SCAn String – simbolių paieška eilutėje.

1010 111w

Paieška yra atliekama palyginimu atėmimo būdu:

*akumuliatorius - laukas, adresuojamas DI*

*Akumuliatorius* – registras AL arba registras AX.

*Laukas, adresuojamas DI* – atitinkamai baito arba žodžio laukas, kurio adresas segmente yra nurodytas registre DI.

Atliekant komandą, atėmimo rezultate, yra suformuojami požymiai ir modifikuojamas registras DI, pagal indeksinių registrų modifikavimo taisyklę – priklausomai nuo požymio DF (aprašyta prie komandos MOVS).

.

**LODS** – LOaD String – eilutės užrašymas į akumuliatorių.

1010 110w

Baito laukas, nurodytas registru SI, užrašomas į akumuliatorių – registrą AL. Žodžio laukas, nurodytas registru SI, užrašomas į akumuliatorių – registrą AX.

Registras SI yra modifikuojamas pagal indeksinių registrų modifikavimo taisyklę – priklausomai nuo požymio DF (aprašyta prie komandos MOVS).

**STOS** – STOr String – registro įsiminimas eilutėje.

1010 101w

Priklausomai nuo to, ar bitas w yra 0 ar 1, registre AL arba AX esanti reikšmė yra įsiminama eilutėje, nurodytoje registru DI. Po to registras DI yra modifikuojamas, pagal indeksinių registrų modifikavimo taisyklę – priklausomai nuo požymio DF (aprašyta prie komandos MOVS).

Pastaba. Šiai komandai segmento registras visada yra ES, o nurodžius segmento keitimo prefiksą, jis yra ignoruojamas.

### ***Valdymo perdavimo komandos***

**CALL** – paprogramės (programos) iškvietimas

Komanda perduoda valdymą nurodytu adresu ir steke įsimena grįžimo adresą. Valdymo perdavimo adresas nurodomas operandu. Procedūrų iškvietimas skirstomas pagal segmentinę atminties organizaciją:

1. Iškvietimas segmento viduje (NEAR);
2. Iškvietimas kitame segmente (FAR).

Priklausomai nuo procedūros iškvietimo būdo, grįžimo adresas įsimenamas steke dvejopai: pirmu atveju kaip IP, antru – kaip IP : CS .

1. Vidinis tiesioginis (segmento viduje).

1110 1000 adr.j.b. adr.v.b.

Dviejų baitų adresu galima nurodyti bet kurią vietą segmente, bet komandoje nurodomas tiesioginis adresas atžvilgiu einamosios IP reikšmės.

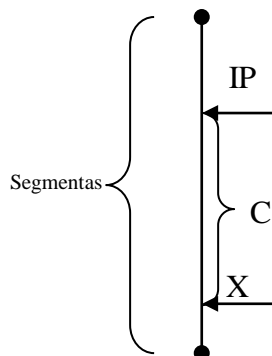
Veiksmai:

1.  $SP \leftarrow SP - 2$ .
2. *Steko viršūnė*  $\leftarrow IP$ .

Į steko viršūnę rašome IP reikšmę, taip išsaugomas grįžimo adresas – komandos, einančios po CALL, adresas.

3.  $IP \leftarrow \text{tiesioginis adresas, nurodytas komandoje kaip operandas} + IP$ .

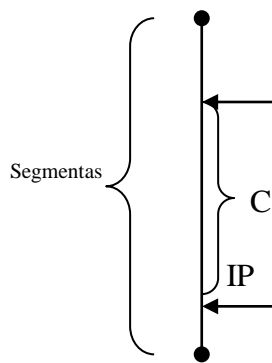
Tiesioginis adresas yra skaičius su ženklu žodyje, adresų diapazonas: -32768 – 32767. Valdymo perdavimo adresas apskaičiuojamas sudedant IP ir tiesioginį adresą.



X – vieta, kur norim perduoti valdymą  
C – poslinkis  
 $X = IP + C$



## KOMPIUTERIŲ ARCHITEKTŪRA

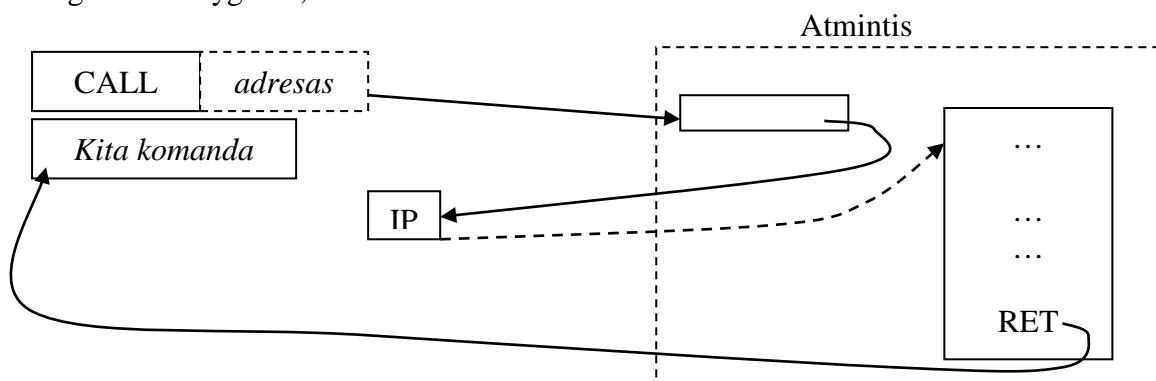


X – vieta, kur norim perduoti valdymą  
C – poslinkis  
 $X = IP + (-C)$

2. Vidinis netiesioginis (segmento viduje)  
1111 1111 mod 010 r/m [posl.j.b. [posl.v.b.]]

Valdymo perdavimo adreso reikšmė yra imama iš registro arba atminties, kuri yra nurodyta komandoje.

Jeigu mod nelygu 11, tada:



Pagal adresą, nurodytą komandoje, iš atminties lauko yra paimama reikšmė ir ji yra traktuojama kaip adresas, kuriuo reikia perduoti valdymą (adresas yra žodis).

Jeigu mod = 11, tada registre esanti reikšmė nusiunčiama į registrą IP, ir tokiu būdu ji yra traktuojama kaip adresas, kuriuo reikia perduoti valdymą.

3. Išorinis tiesioginis (už segmento ribų).  
1001 1010 adr.j.b. adr.v.b. seg.reg.j.b. seg.reg.v.b.

Laukas seg.reg.j.b. seg.reg.v.b. yra nusiunčiamas į registrą CS.

Laukas adr.j.b. adr.v.b. yra nusiunčiamas į registrą IP (tokiu būdu yra traktuojamas kaip poslinkis segmente).

Veiksmi:

1.  $SP \leftarrow SP - 2$ .
2.  $Stekas \leftarrow CS$ .
3.  $SP \leftarrow SP - 2$ .

## KOMPIUTERIŲ ARCHITEKTŪRA

4.  $Stekas \leftarrow IP$ .
5.  $CS \leftarrow$  segmento registro reikšmė, kuriame yra kviečiamoji procedūra.
6.  $IP \leftarrow$  poslinkio segmente reikšmė, kur yra kviečiamoji procedūra.

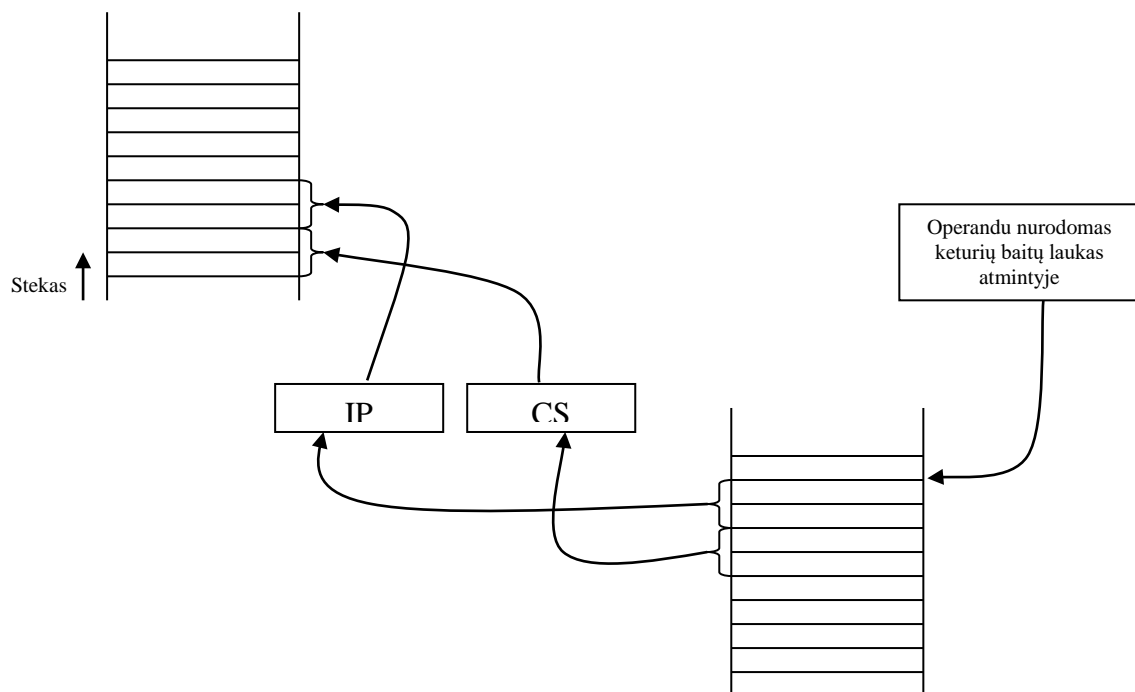
Kviečiama procedūra gali būti bet kurioje operatyvios atminties vietoje.

4. Išorinis netiesioginis.

1111 1111 mod 011 r/m [poslinkis]

Operandu nurodomas dviejų žodžių laukas atmintyje.

Komandos laukas mod negali būti lygus 11.



Vykdamą komandą, nurodyto lauko vyresnysis žodis nusiunčiamas į registrą CS, o jaunesnysis žodis nusiunčiamas į registrą IP. Taip yra atliekamas valdymo perdavimas. Į steką padedamos einamosios, prieš valdymo perdavimą buvusios, registrų CS ir IP reikšmės (einamoji IP registro reikšmė rodo į komandą, esančią po tuo metu vykdomos, šiuo atveju po CALL), todėl iš procedūros grįžtama teisingai.

**RET** – RETurn – grįžimas iš paprogramės.

Tai yra beadresinė komanda, yra skirtingi atvejai grįžimo segmento viduje ir išorėje.

1. Vidinė be steko išlyginimo.

1100 0011

Veiksmai:

## KOMPIUTERIŲ ARCHITEKTŪRA

1.  $IP \leftarrow Stekas.$
2.  $SP \leftarrow SP + 2.$

2. Vidinė su steko išlyginimu.  
1100 0010 bet.op.j.b. bet.op.v.b.

Veiksmai:

1.  $IP \leftarrow Stekas.$
2.  $SP \leftarrow SP + 2.$
3.  $SP \leftarrow SP + \textit{betarpiškas operandas}.$

Pastaba. Betarpiškas operandas yra žodis.

3. Išorinis be steko išlyginimo.  
1100 1011

Veiksmai:

1.  $IP \leftarrow Stekas.$
2.  $SP \leftarrow SP + 2.$
3.  $CS \leftarrow Stekas.$
4.  $SP \leftarrow SP + 2.$

4. Išorinis su steko išlyginimu.  
1100 1010 bet.op.j.b. bet.op.v.b.

Veiksmai:

1.  $IP \leftarrow Stekas.$
2.  $SP \leftarrow SP + 2.$
3.  $CS \leftarrow Stekas.$
4.  $SP \leftarrow SP + 2.$
5.  $SP \leftarrow SP + \textit{betarpiškas operandas}.$

**JMP** – JuMP – besąlyginis perėjimas.

Kaip ir paprogramės iškvietimo komanda, JMP komandos skiriasi priklausomai nuo to, ar perėjimas yra segmento viduje ar išorėje. Skirtumas yra tas, kad nėra išsaugomas grįžimo adresas. Be to yra galimas valdymo perdavimas baito su ženklu ribose.

1. Vidinis artimas.  
1110 1011 *poslinkis*

Poslinkis traktuojamas kaip skaičius su ženklu baite, diapazone: -128 – 127.

Skaičiuojant valdymo perdavimo adresą, baitas yra išplečiamas iki žodžio ir sudedamas su registre IP esančia reikšme.

## KOMPIUTERIŲ ARCHITEKTŪRA

### 2. Vidinis tiesioginis.

1110 1001 posl.j.b. posl.v.b.

Poslinkis traktuojamas kaip skaičius su ženklu žodyje, diapazone: -32768 – 32767.

Skaičiuojant valdymo perdavimo adresą, poslinkis yra sudedamas su registre IP esančia reikšme. Valdymo perdavimas atliekamas pridedant arba atimant atitinkamą reikšmę, kurios diapazonas: -32768 – 32767, yra pakankamas perduoti valdymą į bet kurią vietą segmente.

### 3. Vidinis netiesioginis.

1111 1111 mod 100 r/m [poslinkis]

Tiesioginė adresacija komandoje JMP yra išreiškiama poslinkiu, o netiesioginėje adresacijoje valdymo perdavimo adresas yra imamas iš registro arba atminties žodžio ir nėra sudedamas su registre IP esančia reikšme.

Valdymo perdavimo adresas traktuojamas kaip skaičius be ženklo žodyje, todėl valdymo perdavimas galimas į bet kurią vietą segmente.

### 4. Išorinis tiesioginis.

1110 1010 adr.j.b. adr.v.b. seg.reg.j.b. seg.reg.v.b.

Laukas seg.reg.j.b. seg.reg.v.b. yra nusiunčiamas į registrą CS.

Laukas adr.j.b. adr.v.b. yra nusiunčiamas į registrą IP (tokiu būdu yra traktuojamas kaip poslinkis segmente).

Adresas čia yra traktuojamas kaip skaičius be ženklo ir jis yra siunčiamas į registrą IP, neatliekant sudėjimo su buvusiu registro IP reikšme, kaip vidinio tiesioginio valdymo perdavimo atveju.

### 5. Išorinis netiesioginis.

1111 1111 mod 101 r/m [poslinkis]

Operandu nurodomas dviejų žodžių laukas atmintyje.

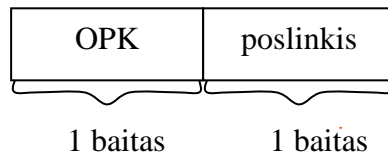
Komandos laukas *mod* negali būti lygus 11.

Vykdam komandą, nurodyto lauko vyresnysis žodis nusiunčiamas į registrą CS, o jaunesnysis žodis nusiunčiamas į registrą IP.

## **Sąlyginis valdymo perdavimas**

## KOMPIUTERIŲ ARCHITEKTŪRA

Yra 17 sąlyginio valdymo perdavimo komandų. Tų komandų formatas:



Vykdamas komandą yra tikrinami požymiai arba registro CX reikšmė ir, priklausomai nuo to tikrinimo rezultato, jei sąlyga yra patenkinta, tai prie registre IP esančios reikšmės yra pridedamas poslinkio baitas, kuris yra traktuojamas kaip skaičius su ženklu ir praplečiamas iki žodžio.

if(ah>5){...  
 cmp ah, 5  
 JA (jump above) daugiau

Komanda	Išplėstinis pavadinimas	Paaiškinimas	Operacijos kodas	Tikrinama sąlyga
JA	Jump if Above	Pereiti, jeigu viršija	77	CF = 0 ir ZF = 0
JNBE	Jump if Not Below nor Equal	Pereiti, jeigu ne mažiau ir ne lygu		
JAE	Jump if Above or Equal	Pereiti, jei viršija arba lygu	73	CF = 0
JNB	Jump if Not Below	Pereiti, jei ne mažiau		
JNC	Jump if Not Carry	Pereiti, jeigu nėra pernešimo		
JB	Jump if Below	Pereiti, jeigu mažiau	72	CF = 1
JNAE	Jump if Not Above nor Equal	Pereiti, jeigu neviršija ir nelygu		
JC	Jump if Carry	Pereiti, jeigu pernešimas		
JBE	Jump if Below or Equal	Pereiti, jeigu mažiau arba lygu	76	CF = 1 arba ZF = 1
JNA	Jump if Not Above	Pereiti, jeigu neviršija		
JE	Jump if Equal	Pereiti, jeigu lygu	74	ZF = 1
JZ	Jump if Zero	Pereiti, jeigu nulis		
JCXZ	Jump if CX is Zero	Pereiti, jeigu CX = 0	E3	CX = 0
JG	Jump if Greater	Pereiti, jeigu daugiau	7F	ZF = 0 ir SF = OF
JNLE	Jump if Not Less nor Equal	Pereiti, jeigu ne mažiau ir nelygu		
JGE	Jump if Greater or Equal	Pereiti, jeigu daugiau arba lygu	7D	SF = OF
JNL	Jump if Not Less	Pereiti, jeigu ne mažiau		

JA jump above (be zenklo sk.)  
 JG jump greater

JA CF = 0 ZF = 0  
 JG ZF = 0 SF = OF  
 SF - sign flag  
 OF - overflow flag

jump reguos tik i tuos du bitus  
 gali sokti nuo -128 iki 127 kodo segmento

## KOMPIUTERIŲ ARCHITEKTŪRA

JL	Jump if Less	Pereiti, jeigu mažiau	7C	SF nelygu OF
JNGE	Jump if Not Greater nor Equal	Pereiti, jeigu ne daugiau ir nelygu		
JLE	Jump if Less or Equal	Pereiti, jeigu mažiau arba lygu	7E	ZF = 1 arba SF nelygu OF
JNG	Jump if Not Greater	Pereiti, jeigu ne daugiau		
JNE	Jump if Not Equal	Pereiti, jeigu nelygu	75	ZF = 0
JNZ	Jump if Not Zero	Pereiti, jeigu ne nulis		
JNO	Jump if Not Overflow	Pereiti, jeigu ne perpildymas	71	OF = 0
JNP	Jump if Not Parity	Pereiti, jeigu nėra lyginumo	7B	PF = 0
JPO	Jump if Parity Odd	Pereiti, jeigu bitų suma nelyginė		
JP	Jump if Parity	Pereiti, jeigu yra lyginumas	7A	PF = 1
JPE	Jump if Parity Even	Pereiti, jeigu bitų suma lyginė		
JO	Jump if Overflow	Pereiti, jeigu perpildymas	70	OF = 1
JNS	Jump if No Sign	Pereiti, jeigu ženklo bitas yra nulis	79	SF = 0
JS	Jump if Sign	Pereiti, jeigu ženklo bitas yra vienetas	78	SF = 1

### ***Ciklų komandos***

**LOOP** – ciklas.

E2 *poslinkis*

Poslinkis yra vieno baido.

Pakartojimas vykdomas tiek kartų, kokia yra registro CX reikšmė.

Vykiant yra mažinama registro CX reikšmė:  $CX \leftarrow CX - 1$ .

Jeigu registre CX esanti reikšmė nelygi nuliui, tai sekanti komanda nustatoma taip:  
*einamasis IP + poslinkis, paimtas iš komandos* :  $IP \leftarrow IP + \text{poslinkis}$ .

Jeigu registre CX esanti reikšmė lygi nuliui, tai vykdoma kita, toliau esanti, komanda.

Ciklo komanda valdymas gali perduodamas per -128 – 127 baitus.

## KOMPIUTERIŲ ARCHITEKTŪRA

Maksimalus ciklo pakartojimo skaičius yra 65 536.

**LOOPE** – LOOP if Equal – pakartoti, jeigu lygu.

E1 *poslinkis*

Poslinkis yra vieno baido.

Pakartojimas vykdomas tiek kartų, kokia yra registro CX reikšmė.

Vykiant yra mažinama registro CX reikšmė:  $CX \leftarrow CX - 1$ .

Jeigu registre CX esanti reikšmė nelygi nuliui ir požymis ZF = 1, tai sekanti komanda nustatoma taip: *einamasis* IP + *poslinkis*, *paimtas iš komandos* :  $IP \leftarrow IP + \text{poslinkis}$ .

Jeigu registre CX esanti reikšmė lygi nuliui, tai vykdoma kita, toliau esanti, komanda.

Ciklo komanda valdymas gali perduodamas per -128 – 127 baitus.

Maksimalus ciklo pakartojimo skaičius yra 65 536.

Alternatyvi šios komandos mnemonika – **LOOPZ** – LOOP if Zero – pakartoti, jeigu nulis.

**LOOPNE** – LOOP if Not Equal – pakartoti, jeigu nelygu.

E0 *poslinkis*

Poslinkis yra vieno baido.

Pakartojimas vykdomas tiek kartų, kokia yra registro CX reikšmė.

Vykiant yra mažinama registro CX reikšmė:  $CX \leftarrow CX - 1$ .

Jeigu registre CX esanti reikšmė nelygi nuliui ir požymis ZF = 0, tai sekanti komanda nustatoma taip: *einamasis* IP + *poslinkis*, *paimtas iš komandos* :  $IP \leftarrow IP + \text{poslinkis}$ .

Jeigu registre CX esanti reikšmė lygi nuliui, tai vykdoma kita, toliau esanti, komanda.

Ciklo komanda valdymas gali perduodamas per -128 – 127 baitus.

Maksimalus ciklo pakartojimo skaičius yra 65 536.

Alternatyvi šios komandos mnemonika – **LOOPNZ** – LOOP if Not Zero – pakartoti, jeigu ne nulis.

### ***Pertraukimų komandos***

**INT** – INTerrupt – programinis pertraukimas.

CD *tipas*

Atliekami veiksmai:

1.  $SP \leftarrow SP - 2$
2.  $Stekas \leftarrow SF$

## KOMPIUTERIŲ ARCHITEKTŪRA

3.  $SP \leftarrow SP - 2$
4.  $Stekas \leftarrow CS$
5.  $SP \leftarrow SP - 2$
6.  $Stekas \leftarrow IP$
7.  $IF \leftarrow 0$
8.  $TF \leftarrow 0$
9.  $CS \leftarrow tipas \cdot 4 + 2$
10.  $IP \leftarrow tipas \cdot 4$

9-tu ir 10-tu veiksmiais yra perduodamas valdymas į pertraukimo apdorojimo programą.

CD 3 – sustojimo taško pertraukimas yra ekvivalentus specialiai pertraukimo komandai iš vieno baido – CC. Komanda assembleriu užrašoma INT 3, o mašininiai kodai CD 3 ir CC yra ekvivalentūs:

$$CS \leftarrow 3 \cdot 4 + 2 = CC + 2$$

$$IP \leftarrow 3 \cdot 4 = CC$$

Komanda INT 4, kurios kodas yra CD 4, iššaukia tą pačią pertraukimo apdorojimo programą, kaip ir komanda **INTO** – INTerrupt if Overflow – pertraukimas, jeigu yra perpildymas. Pertraukimas įvyksta, jeigu OF = 1, ir :

$$CS \leftarrow 4 \cdot 4 + 2 = 10h + 2$$

$$IP \leftarrow 4 \cdot 4 = 10h$$

**IRET** – INTerrupt Return – grįžimas iš pertraukimo apdorojimo programos.  
CF

Atliekami veiksmi:

1.  $IP \leftarrow Stekas$
2.  $SP \leftarrow SP + 2$
3.  $CS \leftarrow Stekas$
4.  $SP \leftarrow SP + 2$
5.  $SF \leftarrow Stekas$
6.  $SP \leftarrow SP + 2$



## ***Procesoriaus būsenos valdymas ir sinchronizavimas***

Yra septynios komandos požymių CF, DF ir IF valdymui.

**CLC** – CLear Carry flag – išvalyti pernešimo požymį:  $CF \leftarrow 0$   
1111 1000

**STC** – SeT Carry flag – nustatyti pernešimo požymį:  $CF \leftarrow 1$   
1111 1001

**CMC** – CoMplement Carry flag – invertuoti požymį CF  
1111 0101

**CLD** – CLear Direction flag – išvalyti krypties požymį:  $DF \leftarrow 0$   
1111 1100

**STD** – SeT Direction flag – nustatyti krypties požymį:  $DF \leftarrow 1$   
1111 1101

**CLI** – CLear Interrupt flag – išvalyti pertraukimo požymį:  $IF \leftarrow 0$   
1111 1010

**STI** – SeT Interrupt flag – nustatyti pertraukimo požymį:  $IF \leftarrow 1$   
1111 1011

**HLT** – HaLT – sustabdyti.  
1111 0100

Komanda perveda procesorių į sustojimo būseną, iš kurios procesorius gali išeiti šiais atvejais:

1. RESET signalas CLR.
2. Nemaskuojamas išorinis pertraukimas NMI.
3. Išorinis pertraukimas, jeigu jis neuždraustas, t.y.,  $IF = 1$ .

Komanda HLT yra perėjimas į pertraukimo laukimo būseną.

**WAIT** – laukti.  
1001 1011

Procesorius, vykdydamas komanda WAIT, kas penki taktai tikrina signalą TEST. Procesorius, sulaukęs signalo TEST, pradeda vykdyti kitą komandą, esančią po WAIT komandos.

WAIT komandos paskirtis yra procesoriaus pristabdymas, kol išorinis įrenginys baigs darbą. Jeigu  $TEST = 1$ , tada komanda WAIT yra nurodytas laukimas, o jeigu  $TEST =$

## KOMPIUTERIŲ ARCHITEKTŪRA

0, tai vykdoma kita komanda. Signalą TEST valdo išoriniai įrenginiai, tame tarpe ir koprocesorius 8087. Mikroprocesorius laukimo metu gali sureaguoti į pertraukimus, kuriuos įvykdęs vėl grįžta į laukimo būseną.

**LOCK** – LOCK the bus – uždaryti magistralę.

1111 0000

Šitas prefiksas gali būti panaudotas prieš bet kokią komandą. Jis priverčia procesorių Intel 8088 blokuoti magistralę tos komandos vykdymo metu, kad joks kitas procesorius negalėtų naudoti magistralės.

Magistralė sujungia procesorių 8088 su išore: tai yra multikpleksinė adreso/duomenų magistralė. Bendra magistralė gali būti valdoma savu kontrolieriu. Procesorius Intel 8088 gali dirbti dviem režimais:

1. Valdančiosios magistralės signalus generuoja pats 8088 procesorius.
2. Magistralę valdo magistralės kontrolieris – speciali mikroschema.

Komanda LOCK monopolizuoja priėjimą prie atminties, tam kad netrukdytų kiti sistemos procesoriai.

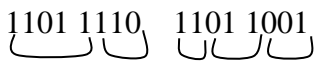
**ESC** – ESCape – nubėgti.

1101 1xxx mod yyy r/m [poslinkis]

Šios komandos pagalba koprocesorius 8087 gauna savo komandas ir operandus procesoriaus 8088 darbo metu. Pats procesorius 8088 tuo metu nieko nedaro, tik iš atminties paima operandą ir jį nusiunčia į magistralę.

Procesoriaus 8088 komandos skaitymo iš atminties metu, koprocesorius 8087 nuolat stebi magistralę ir „pasigauna“ komandą ESC. Po to koprocesorius 8087 kontroliuoja magistralę ir „pasigauna“ adresavimo baitą. Bitai xxx ir yyy iš komandos ESC pirmų dviejų baitų traktuojami kaip koprocesoriaus operacijos kodas. Kuomet procesorius 8088 nusiunčia operando adresą į magistralę (jeigu komandos operandas buvo atmintis), koprocesorius tą operando adresą „pagauna“ ir pratęsia pradėtos komandos vykdymą. Į procesoriaus Intel 8088 registrus tada niekas nepatenka. Jeigu laukas *mod* = 11, tai operandas yra registre, todėl procesorius 8088 nieko nedaro.

Pavyzdys.

DED9        
ESC      xxx   mod   yyy r/m

Procesorių 8088 ir 8087 registrai vienas kitam nėra prieinami, ir informaciją jie gali perduoti tik per atmintį. Jeigu laukas *mod* = 11, tada komanda ESC mikroprocesoriuje veikia kaip NOP. Komandos ESC prasminis operandas yra operandas atmintyje, o komandą su operandu registre mikroprocesorius įvykdo kaip NOP.

Ar operandas yra baitas ar žodis, priklauso nuo koprocesoriaus operacijos xxx yyy.

## **6. Koprocesorius Intel 8087**

Koprocesorius yra skirtas mikroprocesoriaus Intel 8088 galimybių išplėtimui: aritmetinių veiksmų su slankiu kableliu atlikimui. Koprocesorius yra pavaldus mikroprocesoriui ir pačiame mikroprocesoriuje Intel 8088 yra atsižvelgta į koprocesoriaus buvimo galimybę per ESC komandą.

Koprocesorius stebi mikroprocesoriaus imamas iš atminties vykdymui komandas ir, jeigu eilinė komanda yra koprocesoriaus komanda, tai jis ją įvykdo lygiagrečiai mikroprocesoriaus darbui. Tokiu būdu realiai gaunamas lygiagretus programos vykdymas.

Kadangi vyksta lygiagretus darbas, yra būtina šio darbo sinchronizacija. Tuo tikslu mikroprocesoriuje Intel 8088 naudojama komanda WAIT. Komanda WAIT pristabdo mikroprocesoriaus darbą, tol kol signalas TEST tampa aktyviu. Signalas TEST tampa aktyviu, kai koprocesorius baigia vykdyti komandą. Tokiu būdu galima garantuoti, kad mikroprocesorius pratęs savo darbą tik po to, kai koprocesorius baigs komandos įvykdymą.

Mikroprocesorius ir koprocesorius yra susieti tik išoriškai. Mikroprocesorius negali kreiptis į koprocesoriaus registrus ir atvirkščiai. Duomenimis abu procesoriai apsieičia tik per atmintį, prie kurios jie abu turi priejimą. Tačiau atminties adresavimo registrai yra tiktai mikroprocesoriuje Intel 8088, todėl koprocesorius operandų atmintyje adresavimui naudojami mikroprocesoriaus paslaugomis per komandą ESC. Sistemoje be koprocesoriaus komanda ESC yra analogiška komandai NOP, tačiau yra ilgiau vykdoma.

ESC komandos turi adresinę informaciją, kuri saugoma adresavimo baite, su poslinkiu arba be jo. Nors mikroprocesorius komandos ESC nevykdo ir traktuoja ją kaip NOP, tačiau apskaičiuoja vykdomąjį adresą ir kreipiasi į atmintį duomenų skaitymui ir rašymui. Jeigu adresavimo baitas nusako ne atmintį, o registrą, tai kreipimasis į atmintį nevyksta.

Koprocesorius, pastebėjęs komandos ESC iš atminties paėmimą, toliau stebi, kada mikroprocesorius ims iš atminties duomenis, ir duomenų adresui atsidūrus sisteminėje magistralėje, koprocesorius jį perima ir taip žino, kur yra atmintyje jam reikalingi duomenys. Taigi mikroprocesorius apskaičiuoja operando atmintyje adresą, o koprocesorius atlieka likusią komandos dalį.

Koprocesorius nepakeičia nei vienos Intel 8088 komandos, o tiktai papildo komandų sistemą naujų tipų aritmetinėmis operacijomis. Tokiu būdu koprocesoriaus programavimas yra komandų ESC sudarymas ir, jeigu mikroprocesoriui reikalingas koprocesoriaus rezultatas, komandos WAIT naudojimas veiksmų sinchronizavimui.

Koprocesorius turi išplėstinį aritmetinių duomenų tipų rinkinį, su kuriais yra atliekamos aritmetinės operacijos.

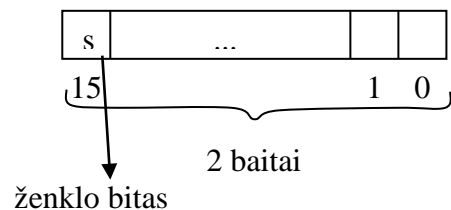
## KOMPIUTERIŲ ARCHITEKTŪRA

Mikroprocesorius atlieka veiksmus su baitais arba žodžiais, traktuodamas skaičius kaip skaičius su ženklu arba be ženklo, taip pat turi dešimtainius supakuotus ir išpakuotus skaičius.

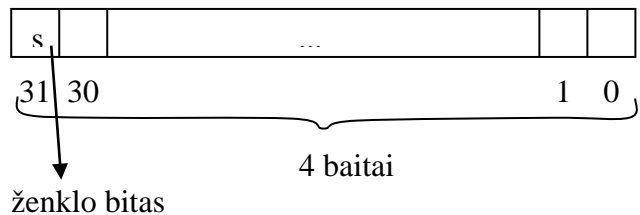
Koprocesorius turi septynis duomenų tipus, iš kurių šeši yra būdingi tik koprocesoriui, t.y., jų nėra mikroprocesoriuje. Keturi koprocesoriaus duomenų formatai yra skirti darbui su sveikais skaičiais, vienas iš jų yra išplėstinis dešimtainis formatas. Likę trys formatai skirti veiksmų su realiais skaičiais atlikimui.

### ***Koprocesoriaus duomenų formatai***

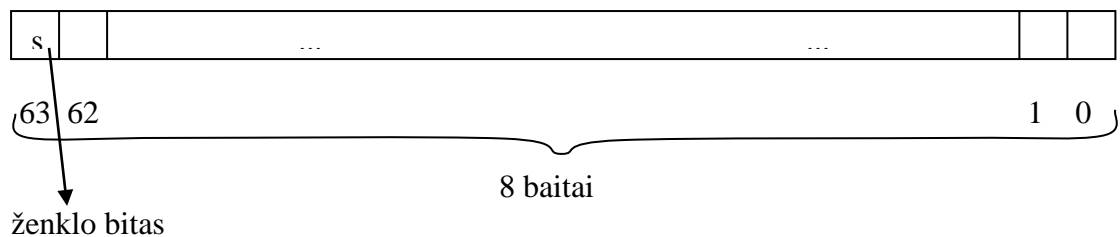
1. Sveikas žodis, dvejetainiu papildomu kodu.



2. Sveikas trumpas, dvejetainiu papildomu kodu.

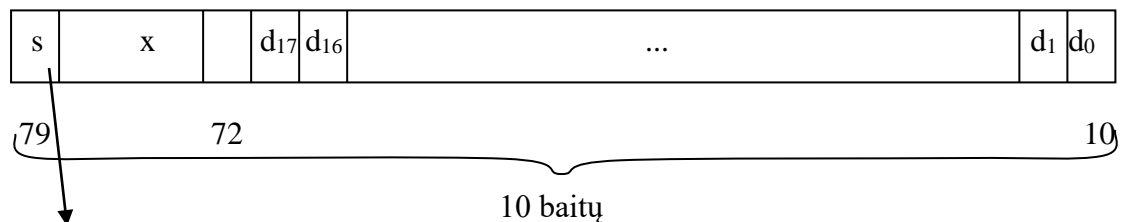


3. Sveikas ilgas, dvejetainiu papildomu kodu.



4. Supakuotas dešimtainis išplėstas.

Vienas dešimtainis skaitmuo užima keturis bitus.



ženklų bitas

$d_i$  – dešimtainis skaitmuo;

$s$  – jeigu  $s = 0$ , tai skaičius teigiamas, jeigu  $s = 1$ , tai skaičius neigiamas;

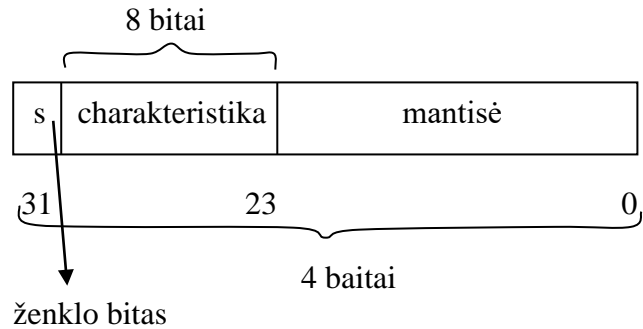
$x$  – bitai, neturintys įtakos.

## KOMPIUTERIŲ ARCHITEKTŪRA

Rašant supakuoto dešimtainio išplėsto formato duomenis iš atminties į koprosesoriaus registrą, bitai x yra neapibrėžti, o užrašant tokią reikšmę iš registro į atmintį, x bitai yra nuliniai.

Aštuoniolika dešimtainių skaitmenų užima devynis baitus, dar vienas baitas yra ženklų baitas (jis reikalingas ženklui), todėl iš viso reikia dešimties baitų.

### 5. Trumpas realus



Kiekvieną skaičių  $z$  galima išreikšti tokia forma:

$$z = (-1)^s \cdot 2^{\text{eilė}} \cdot 0, \text{ mantisė}, \text{ arba:}$$

$$z = (-1)^s \cdot 2^{\text{eilė}} \cdot 1, \text{ mantisė};$$

s abiejose formose yra ženklų bito reikšmė.

Dauguma kitų procesorių naudoja pirmąją formą, kurioje naudojama  $0, \text{ mantisė}$ , tačiau Intel procesoriai naudoja antrąją formą:  $1, \text{ mantisė}$ .

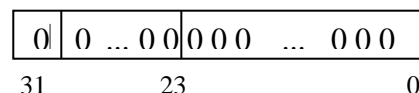
Eilės diapazonas yra  $-127 - 127$  (reikšmė  $-128$  naudojama specialioms reikšmėms). Charakteristika yra „pastumta“ eilė:  $\text{charakteristika} = \text{eilė} + 7Fh$ , ( $7Fh = 127_{10}$ ). Charakteristikos diapazonas yra  $0 - 254$ . Procesorius naudoja charakteristiką – skaičių be ženklų, o ne eilę, tam kad būtų galima efektyviau atlikti skaičių lyginimą: lyginant du realius skaičius užtenka juos peržiūrėti iš kairės į dešinę, ir taip efektyviai sužinoti, kurio charakteristika yra didesnė. Jeigu vieno skaičiaus charakteristika yra didesnė, tai jis akivaizdžiai yra didesnis; jeigu charakteristikos lygios, tai tikrinami iš kairės į dešinę mantisės bitai. Eilė, tuo tarpu, yra skaičius su ženklu, ir lyginimas, naudojant ją, būtų mažiau efektyvus.

Normalizuotame skaičiuje mantisės sveikas skaitmuo Intel procesoriuje yra 1, todėl nereikia jo saugoti; atliekant skaičiavimus, tas 1 yra turimas „omenyje“. Šiame formate mantisę sudaro 23 dvejetainiai skaitmenys. Mantisė yra trupmeninė realaus skaičiaus dalis, t.y., kablelis yra kairiau mantisės.

Atmintyje gali būti saugomi tik normalizuoti skaičiai, t.y., pavidalo:

$$(-1)^s \cdot 2^{\text{char} - 127} \cdot 1, \text{ mantisė}$$

Skaičius 0 yra išimtinis atvejis. Nulinis skaičius yra:



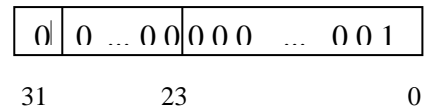
Tačiau pagal trumpo realaus formatą tai yra  $(-1)^0 \cdot 2^{-127} \cdot 1,0 = 2^{-127}$ .

Todėl mažiausias šio formato skaičius yra  $(-1)^0 \cdot 2^{-127} \cdot 1,00..01$ .

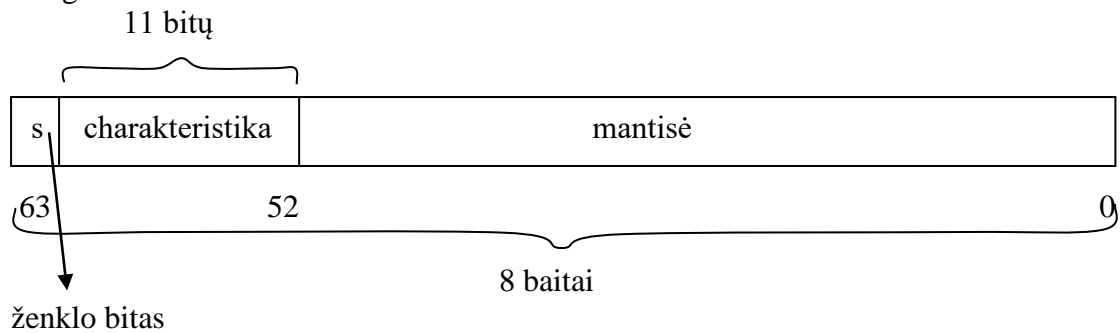
23 skaitmenys

## KOMPIUTERIŲ ARCHITEKTŪRA

Taigi mažiausias galimas formato skaičius yra:



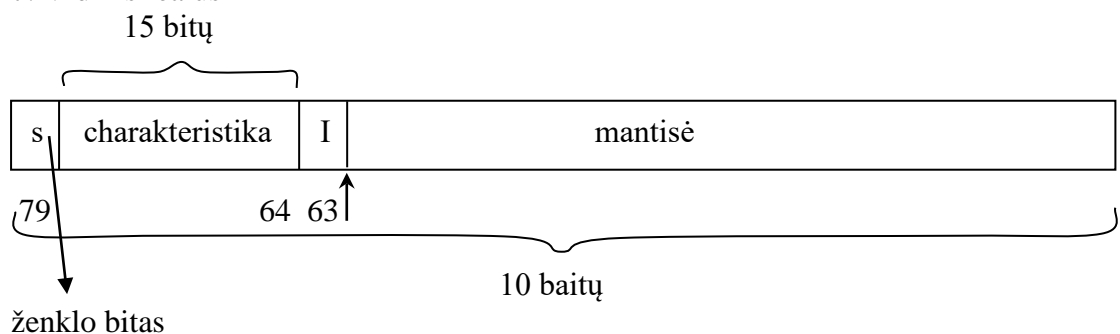
6. Ilgas realus.



Mantisė sudaro 52 dvejetainiai skaitmenys. Mantisė yra trupmeninė realaus skaičiaus dalis, t.y., kabelis yra kairiau mantisės.

Charakteristika yra eilė, perstumta per 3FFh, t.y.,  $\text{eilė} = \text{charakteristika} - 3\text{FFh}$ , o  $\text{charakteristika} = \text{eilė} + 3\text{FFh}$ , ( $3\text{FFh} = 1023_{10}$ )

7. Vidinis realus



Laikinas realus formatas yra skirtas saugoti reikšmėms, kurioms reikia dar didesnio tikslumo, negu gali užtikrinti ilgas realus formatus. Didesnis tikslumas pasiekiamas turint galimybę keisti mantisės sveiką skaičių, kuris dabar jau nebūtinai yra 1 pagal nutylėjimą, bet gali būti ir 0. Sveikas mantisės skaičius yra saugomas bite I. Tačiau tokios reikšmės nėra normalizuotos, t.y., yra denormalizuotos, todėl jos negali būti saugomos atmintyje, o tik koprosesoriaus registruose.

Charakteristika yra eilė, perstumta per 3FFFh = 16383<sub>10</sub>, t.y.,  $\text{eilė} = \text{charakteristika} - 3\text{FFFh}$ , o  $\text{charakteristika} = \text{eilė} + 3\text{FFFh}$ .

63-ias bitas I yra skirtas mantisės sveiko bito saugojimui, o kabelis suprantamas kaip esantis tarp 63-io bito ir prieš 62-ą bitą. Kaip ir mikroprocesoriaus atveju, taip ir koprosesoriaus atveju, atmintyje operando duomenys saugomi taip, kad mažesnių adresų dalyje yra mažesnės bitų reikšmės. Ženklo bitas visada yra operando lauke su vyriausio baito adresu.

## KOMPIUTERIŲ ARCHITEKTŪRA

Sveiko žodžio formatas yra identiškas mikroprocesoriaus formatui: žodiniam formatui su ženklu, ir yra apibrėžiamas programoje operatoriumi DW (Define Word). Tokio formato diapazonas yra  $-32768 - 32767$ .

Sveiką trumpą formatą apibrėžia operatorius DD (Define Double Word). Jo reikšmės yra keturių baitų lauke, diapazone:  $-2^{31} - 2^{31}-1$ . DD pagalba galima apibrėžti porą: *segmentas : poslinkis*.

Jeigu operandas yra adresas, tai pagal operatorių DD assembleris sukuria porą *segmentas : poslinkis*. Jeigu operandas yra reikšmė, tai operandas yra trumpo sveiko formatas.

Skaičių sveiko ilgo (keturgubas žodis) formato apibrėžimui naudojamas operatorius DQ (Define Quadruple). Tokio formato skaičių diapazonas yra  $-2^{63} - 2^{63}-1$ . Operatorius DQ gali apibrėžti konstantą.

Operatoriais DB (Define Byte), DW (Define Word), DD (Define Double word), DQ (Define Quadruple) galima rezervuoti atitinkamo dydžio laukus, assembleriu po jų nurodžius ?. Panaudojus dar ir operatorių DUP (DUPLICATE), laukai yra dubliuojami.

Skaičių supakuotas dešimtainis išplėstinis formatas yra analogiškas mikroprocesoriaus supakuotam dešimtainiam formatui, tik veiksmai iš karto atliekami su aštuoniolika skaitmenų devyniuose baituose, o dešimtas baitas skirtas ženklo bitui.

Dešimtainio formato laukas apibrėžiamas operatoriumi DT (Define Ten), toks laukas užima 10 baitų. Operuojant supakuotomis dešimtainėmis reikšmėmis assembleriu, reikia jas nurodyti šešiolyktaine sistema. Jeigu operatoriaus DT operando lauke bus parašytas dešimtainis skaičius, tai jis bus paverstas dvejetainiu skaičiumi, o ne supakuotu dešimtainiu skaičiumi.

Kadangi teigiami dešimtainiai supakuoti skaičiai vizualiai nesiskiria nuo šešiolyktainių, pakanka po tokio skaičiaus prirašyti h. Užrašant neigiamą dešimtainio supakuoto formato skaičių, prieš jį reikia parašyti 80 ir žiūrėti, kad bendras skaitmenų skaičius būtų dvidešimt, o gale turi būti h. Pavyzdžiui, -1234 yra užrašomas taip:

DT 80 000 00000 00000 01234h

### ***Koprosesoriaus duomenų formatų vaizdavimas***

Trumpas realus formatas (32 bitai) ir ilgas realus formatas (64 bitai) atitinka realių skaičių vaizdavimo standartą.

Trumpo realaus formate aštuoni bitai yra skiriami eilei. Eilės diapazonas būtų  $-128 - 127$ , tačiau eilės reikšmė  $-128$  yra rezervuota neapibrėžtai reikšmei fiksuoti ir todėl yra negalima. Todėl realiai eilės diapazonas yra  $-127 - 127$ . Eilė yra saugoma padidinta per 127, ir tada yra vadinama charakteristika. Charakteristika nėra skaičius papildomu kodu, kur didesnis diapazonas reiškia neigiamų skaičių kodavimą

## KOMPIUTERIŲ ARCHITEKTŪRA

Pavyzdžiui:

Eilė	Charakteristika
-127	0
-1	07Eh
0	7Fh
1	080h
127	0FEh

Būtent toks eilės saugojimas yra susijęs su efektyvesniu skaičių lyginimu: didesnis skaičius turi didesnę charakteristikos lauko reikšmę. Jeigu charakteristikos yra lygios, tai skaičiai palyginami pagal mantisę. Bet kokių atveju, skaičių lyginimas pabičiui iš kairės į dešinę, pradėjus nuo vyriausio bito, leidžia nustatyti, kuris skaičius yra didesnis.

Dvidešimt trijuose mantisės bituose didžiausia reikšmė, kurią galima saugoti yra:

$$\underbrace{111\dots111}_{23} < 2^{23} = 8 \text{ M} \approx 8\,000\,000$$
$$\underbrace{10000\dots00}_{23}$$

Todėl dvidešimt trijuose mantisės bituose galima saugoti skaičių, tikslumu šešių – septynių dešimtinių skaitmenų (visus šešiaženklus skaičius ir dalį septyniaženklių). Skaičių eilės diapazonas yra sąlygojamas charakteristikos ir yra  $10^{-38} - 10^{38}$ .

Pats vyriausias bitas nurodo skaičiaus ženklą, o kitas, po jo esantis bitas, nurodo eilės ženklą.

Dvigubo tikslumo realaus skaičiaus formatas iš aštuonių baitų mantisei skiria 52 bitus. Tai atitinka penkiolikos – šešiolikos dešimtinių skaitmenų tikslumą. Vienuolikos bitų eilės laukas sąlygoja eilės diapazoną:  $2^{-1023} - 2^{1023}$ . Dešimtainėje formoje tai atitinka eilės diapazoną:  $10^{-307} - 10^{307}$ .

Koprosorius realius skaičius atmintyje visada saugo normalizuotu pavidalu, t.y., sveikas mantisės skaičius yra 1, todėl tokiu atveju jį saugoti nėra prasmės. Atliekant skaičiavimus jis, žinoma, yra „turimas omenyje“, tačiau atmintyje nėra saugomas. Taigi realiai mantisės vyriausiasis bitas yra sveikas ir nėra saugomas, o saugomi bitai yra po kablelio (po sveiko bito).

Pirmi šeši duomenų formatai (sveikas žodis, sveikas trumpas, sveikas ilgas, supakuotas dešimtainis išplėstas, trumpas realus, ilgas realus) yra išoriniai koprosoriaus atžvilgiu, t.y., įsimindamas duomenis atmintyje, koprosorius juos įsimena vienu iš tų šešių formatų, taip pat ir skaityti iš atminties gali vienu iš šių šešių formatų. Pačiame procesoriuje visų šešių formatų duomenys saugomi koprosoriaus vidiniu formatu. Taigi pirmi šeši formatai naudojami tik duomenų saugojimui, bet ne operavimui jais koprosoriuje.



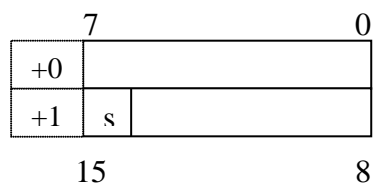
## KOMPIUTERIŲ ARCHITEKTŪRA

Vidinis realus duomenų formatas turi didžiausią tikslumą (mantisę) ir didžiausią diapazoną (eilę). Jis fiksuojamas dešimtyje baitų. Šešiasdešimt keturių bitų mantisė sąlygoja galimą devyniolikos dešimtainių skaitmenų tikslumą. Paprastai mantisė yra normalizuota, bet gali būti ir denormalizuota, todėl vyriausia mantisės bitas – sveikas mantisės skaičius, yra saugomas šitame formate. Penkiolikos bitų eilė sąlygoja skaičiaus eilės diapazoną:  $2^{-16383} - 2^{16383}$ , t.y.,  $10^{-4932} - 10^{4932}$ .

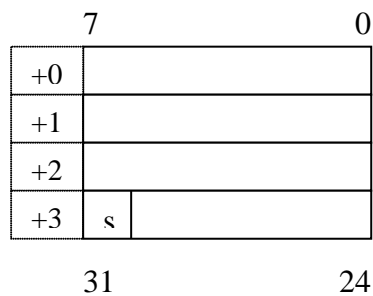
Kadangi reikšmė vidiniu formatu gali būti denormalizuota, tai kairioji diapazono riba gali pasislinkti į kairę per  $2^{64}$ , tačiau tikslumo mantisėje sąskaita.

Asembleriu galima apibrėžti keturių, aštuonių ir dešimties baitų laukus, naudojantis operatoriais DD, DQ ir DT. Atmintyje koprosesorius duomenys įvairiais formatais vaizduojami tokiu būdu:

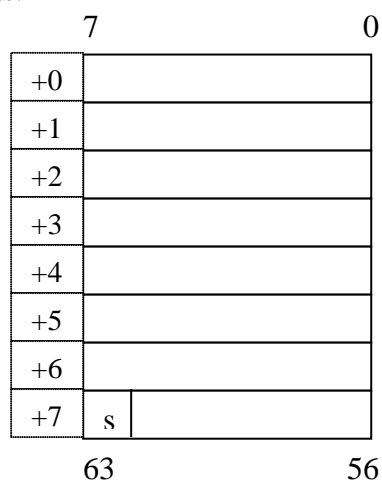
### 1. Sveikas žodis.



### 2. Trumpas sveikas.



### 3. Ilgas sveikas.

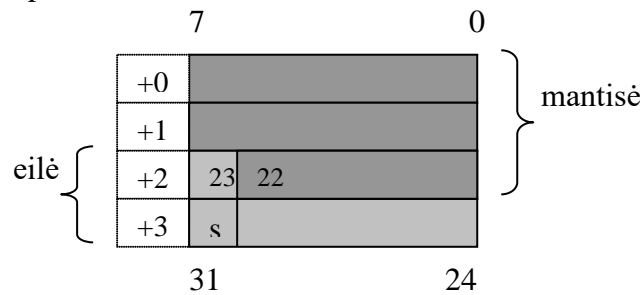


### 4. Supakuotas dešimtainis.

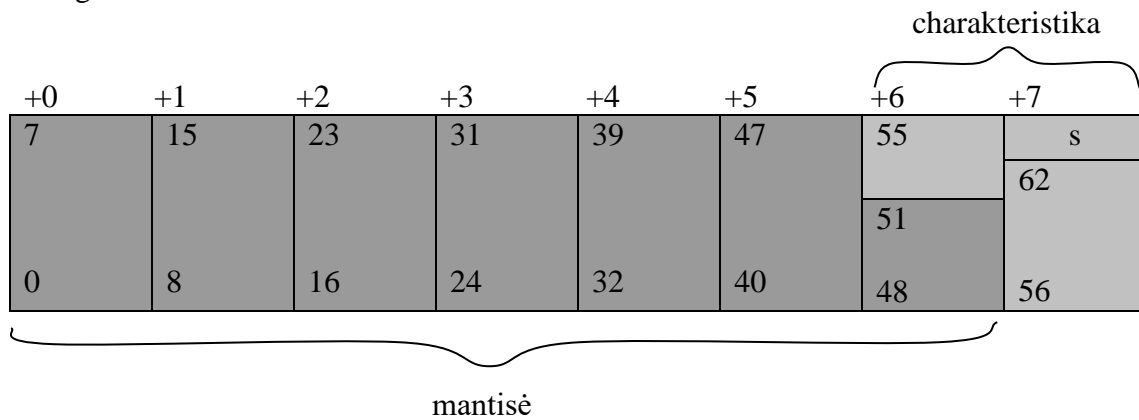
7. Separationsschritt:

+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	
7	d <sub>1</sub>	d <sub>3</sub>	d <sub>5</sub>	d <sub>7</sub>	d <sub>9</sub>	d <sub>11</sub>	d <sub>13</sub>	d <sub>15</sub>	d <sub>17</sub>	s
0	d <sub>0</sub>	d <sub>2</sub>	d <sub>4</sub>	d <sub>6</sub>	d <sub>8</sub>	d <sub>10</sub>	d <sub>12</sub>	d <sub>14</sub>	d <sub>16</sub>	x

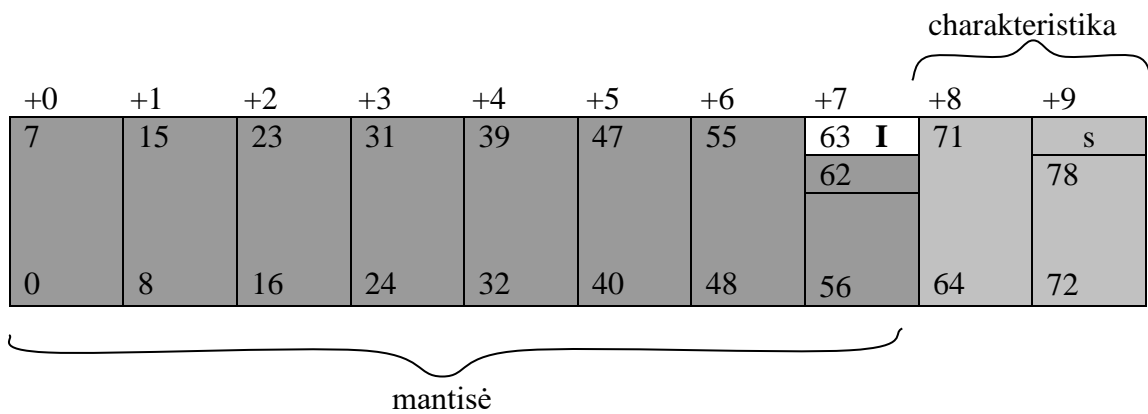
5. Trumpas realus.



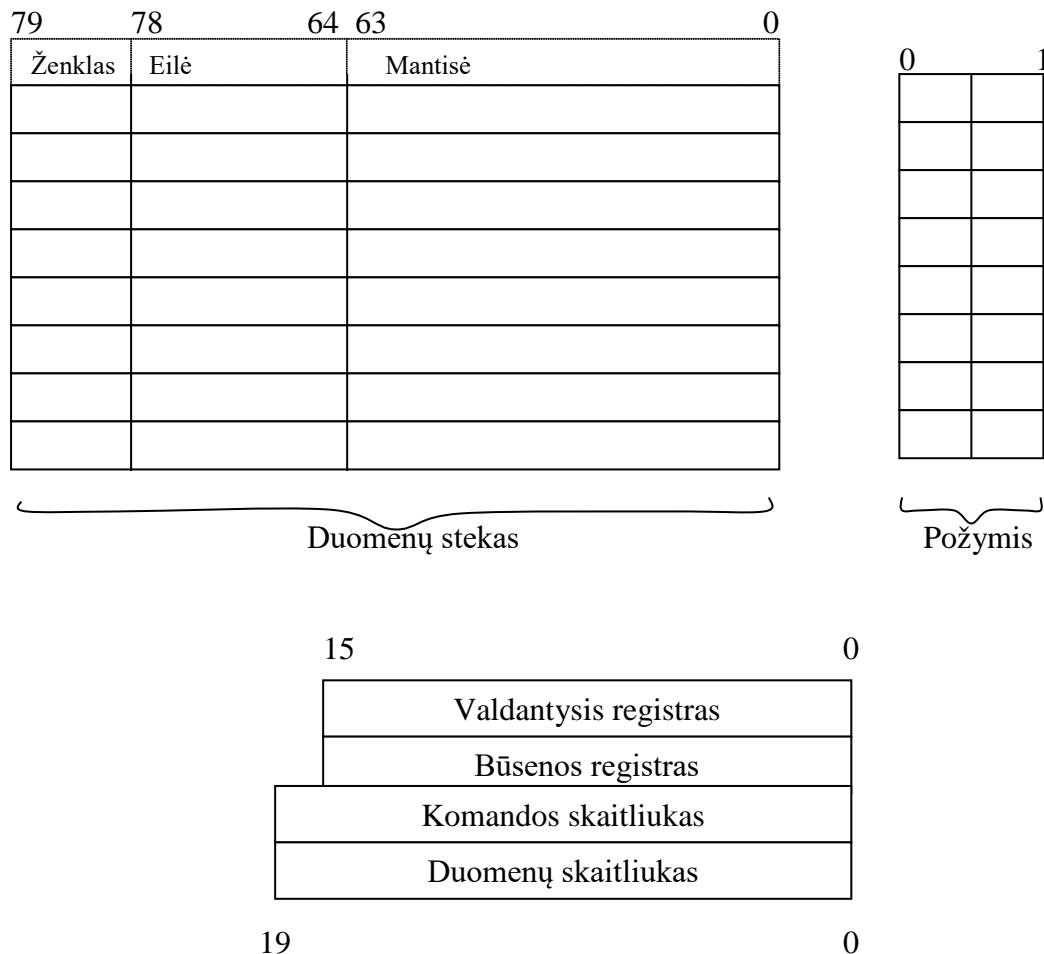
6. Ilgas realus.



6. Vidinis realus.



## Koprosesorius architektūra



Koprosesorius turi keturis specialius registrus ir aštuonių registrų steką aritmetiniams duomenims saugoti. Steko elementas turi 80 bitų ir atitinka vidinį koprosesorius formatą. Duomenys, esantys bet kuriame iš šešių išorinių formatų, yra atvaizduojami vidiniu formatu.

Steko registrai neturi nuosavų, fiksuotų numerių. Jie yra numeruoti steko viršūnės atžvilgiu, kuri, dirbant su steku, kinta. Rašoma visada į steko viršūnę. Prieš rašant, steko viršūnė yra padidinama. Išstumiant iš steko, steko rodyklė yra sumažinama.

Pastebėjimas. Sveikojo formato duomenys yra atvaizduojami į realų vidinį formatą.

Požymių registras fiksuoja, kurie iš steko elementų yra laisvi, kurie yra užimti, o bandant patalpinti duomenis į užimtą steko elementą, fiksuojama ypatinga situacija – nepriimtina komanda.

Koprosesoriuje yra skaitymo ir rašymo komandos. Duomenys yra patalpinami į steko viršūnę – registrą, skaitant iš atminties į registrą, ir iš steko viršūnės užrašomi iš registro į atmintį, tuo pačiu atliekant transformacijas tarp formatų.

## KOMPIUTERIŲ ARCHITEKTŪRA

Skaičiuojamosios (aritmetinės) komandos gali atlikti veiksmus su operandais registrais arba su registru ir atmintimi. Vienas iš operandų būtinai turi būti steko viršūnė. Į atmintį rezultatas gali būti užrašomas atskira rašymo komanda.

### **Valdantysis žodis**

Valdantysis registras dar yra vadinamas valdančiuoju žodžiu.

Koprosorius turi du 16 bitų registrus: valdantįjį ir būsenos. Valdantysis registras reikalingas valdančiojo žodžio saugojimui, būsenos registras – būsenos žodžio saugojimui. Valdantysis registras leidžia programuotojui nustatyti koprosoriaus darbo režimus.

Sutrumpinimuose raidė M reiškia *Mask* – maskavimą. Jeigu atitinkamas maskavimo bitas yra 0, tai pertraukimai leidžiami, o jei maskavimo bitas yra 1, tai pertraukimai yra draudžiami.

	IM	Illegal Mask – nepriimtinos operacijos maskavimas
1	DM	Denormalization Mask – operando denormalizacijos maskavimas
2	ZM	Zero Mask – dalybos iš nulio maskavimas
3	OM	Overflow Mask – perpildymo maskavimas
4	UIM	Underflow Mask – reikšmingumo išnykimo (eilės praradimo) maskavimas
5	PM	Precision Mask – tikslumo maskavimas
6		nenaudojama
7	IFM	Interrupt Enable Mask – pertraukimų maskavimas
8	PC	Precision Control - tikslumo valdymas
9		
10	RC	Round Control apvalinimo valdymas
11		
12	IC	Infinity Control – begalybės valdymas
13		nenaudojama
14		
15		

PC – tikslumo valdymas:

00	24 bitai (23 bitų mantisė)
01	Rezervuota
10	53 bitai (52 bitų mantisė)
11	64 bitai (64 bitų mantisė)

## KOMPIUTERIŲ ARCHITEKTŪRA

RC – apvalinimo valdymas:

00	iki artimiausio lyginio
01	apvalinimas žemyn
10	apvalinimas aukštyn
11	nukirtimas link nulio

IC – begalybės valdymas:

0	projektyvinė
1	afininė

Koprocesorius gali fiksuoti ypatingas situacijas ir pagal jas generuoti pertraukimą. Valdančiuoju žodžiu programuotojas gali valdyti, kokias situacijas apdoroti aparatūriniu pertraukimu nuo procesoriaus 8087 ir kokias situacijas analizuoti programiškai, analizuojant gautą rezultatą.

Koprocesorius yra prijungtas prie NMI su kodu 2, kaip ir pertraukimas nuo atminties kontrolės schemų pagal lyginimą. Todėl tokio pertraukimo apdorojimo programa turi mokėti sureaguoti į abu pertraukimų šaltinius.

Eilės reikšmė -128 yra rezervuota kitoms reikšmėms, ir yra vadinama NAN – Not A Number – neapibrėžta reikšmė.

Apvalinimo valdymas (Round Control) – naudojamas reikšmės perrinkimui, kai nepakanka tokio tikslumo, kokį leidžia naudojamas formatas. Galimi pasirinkimai yra:

1. Kitas didesnis.
2. Kitas mažesnis.
3. Apvalinimas nulio kryptimi.
4. Apvalinimas iki artimiausio lyginio.

Perpildymas (Overflow Mask) – tai steko perpildymas.

Tikslumo valdymas (Precision Control) yra naudojamas pilno tikslumo – 64 bitų (10 baitų, 64 bitų mantisė) apribojimui iki 53 bitų (8 baitai, 52 bitų mantisė) arba 24 bitų (4 baitai, 23 bitų mantisė) tam, kad galėtume dirbti suderinamumo režime su anksčiau sudarytomis procedūromis mašinoms su mažesniu tikslumu – pagal standartą 32 slankus (4 baitai) ir 64 slankus (8 baitai).

Tikslumo maskavimas (Precision mask) – tai tikslumo praradimo maskavimas.

Begalybės valdymas (Infinity Control) – juo nustatoma, ar skirtingoms reikšmėms fiksuoti  $+\infty$  ir  $-\infty$ . Pagal nutylėjimą yra projektyvinė begalybė, kai  $+\infty$  ir  $-\infty$  nesiskiria. Afininė begalybė yra, kai  $+\infty$  ir  $-\infty$  skiriasi.

## Būsenos žodis

Būsenos registras yra dar vadinamas būsenos žodžiu.

Būsenos žodis saugo einamąją koprosesoriaus būseną.

0	IF	Illegal Exeption – neegzistuojanti komanda
1	DF	Denormalization Exeption – denormalizuotas operandas
2	ZF	Zero Exeption – dalyba iš nulio
3	OF	Overflow Exeption – perpildymas
4	UF	Underflow Exeption – reikšmingumo išnykimas (eilės praradimas)
5	PF	Precision Exeption – tikslumas
6		nenaudojama
7	IR	Interrupt Request – pertraukimų reikalavimas
8	C0	Sąlygos kodai
9	C1	
10	C2	
11	ST	STack – steko rodyklė
12		
13		
14	C3	Sąlygos kodas
15	B	užimtas

Būsenos žodyje esantys bitai parodo ypatingą situaciją, kilusią komandos vykdymo metu. Todėl, kilus pertraukimui nuo koprosesoriaus, kuris yra bendras visoms situacijoms koprosesoriuje, pagal būsenos žodį pertraukimo apdorojimo programa gali nustatyti pertraukimo priežastį.

Būsenos žodyje yra bitas B, kuris parodo, ar koprosesorius yra užimtas ar laisvas, t.y., jis vykdo komandą ar ne. Bito B reikšmė yra lygi išorinio signalo TEST reikšmei, pagal kurią sinchronizuojamas darbas su mikroprocesoriumi. Jeigu TEST = 1, tai mikroprocesorius laukia, o jeigu TEST = 0, tai mikroprocesorius imasi vykdyti kitą komandą.

Būsenos žodis turi koprosesoriaus steko rodyklę, kuri kinta diapazone 000 – 111.

Dažniausiai naudojama būsenos žodžio dalis yra sąlygos kodas. Sąlygos kodo keturis bitus galima nustatyti koprosesoriaus komandos. Du iš šių bitų tiesiogiai atitinka mikroprocesoriaus požymių registro SF požymius CF ir ZF. Tie požymiai yra išdėstyti tose pačiose pozicijose – bituose, kaip ir registro SF jaunesniojo baido požymiai CF ir ZF, t.y.:

## KOMPIUTERIŲ ARCHITEKTŪRA

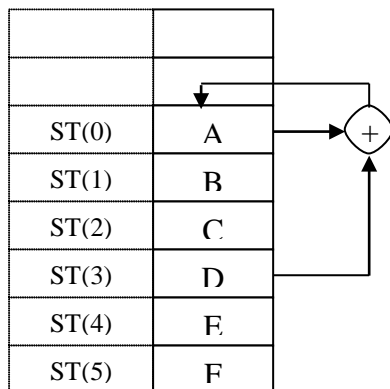
CF (nulinis bitas)	C0
ZF (šeštas bitas)	C3

Tokio išdėstymo patogumas yra tas, kad užrašius būsenos žodį į atmintį, o vėliau patalpinus vyresnįjį baitą registrą AH, iš kurio komandos SAHF pagalba galima suformuoti registro SF jaunesnįjį baitą su požymiais ZF ir CF, pagal koprosesoriaus komandos įvykdymo rezultatą. Kadangi koprosesoriuje skaičiai yra realūs su ženklu, tai šių dviejų požymių pakanka bet kurių dviejų skaičių palyginimui.

Kiti du sąlygos kodo bitai yra naudojami specialia koprosesoriaus komanda, kuri tikrina specialias ypatingas skaitmenines reikšmes, kurias „žino“ koprosesorius. Kadangi tokios specialios reikšmės yra ypatingai apdorojamos, tai sąlygos kodas leidžia pažinti tokias specialias reikšmes.

### ***Koprosesoriaus Intel 8087 komandų sistema***

Steko registrų adresacija yra atliekama santykinai steko viršūnės atžvilgiu. Asembleriu steko viršūnė yra nurodoma ST(0) arba ST. Kiti elementai, tolstant nuo viršūnės, yra ST(1), ST(2), ..., ST(7). Pavyzdžiui, FADD ST(0), ST(3) :



Koprosesoriaus komandos yra skirstomos į tris grupes:

1. Duomenų persiuntimo komandos iš operatyviosios atminties į koprosesorių ir iš koprosesoriaus į operatyvąją atmintį.
2. Koprosesoriaus valdymo komandos.
3. Skaitmeninio apdorojimo komandos.

### ***Duomenų persiuntimo komandos***

Šią grupę sudaro trys pagrindinės komandos.

Pakrovimo komanda patalpina reikšmę į steko viršūnę. Visų koprosesoriaus komandų pavadinimai prasideda raide „F“. Mikroprocesoriuje operacijos kodas yra bendras visiems operandų formatams (pvz., komanda MOV), o koprosesoriuje komandos

## KOMPIUTERIŲ ARCHITEKTŪRA

skiriasi priklausomai nuo operando: fiksuoto sveiko, dešimtainio, slankaus. Asembleris atskiria operando lauko ilgį: 4, 8, 10 baitų, tačiau neatskiria formato: sveikas ar realus, dešimtainis ar vidinis.

Duomenų formatas	Lauko ilgis	Pakrovimo komanda
Sveikas žodis	16 bitų	FILD
Sveikas trumpas	32 bitai	
Sveikas ilgas	64 bitai	
Dešimtainis supakuotas	80 bitų	FBLD
Trumpas realus	32 bitai	FLD
Ilgas realus	64 bitai	
Vidinis realus	80 bitų	

Duomenų lauko ilgį nustato pats asembleris pagal aprašą.

Tokie komandos antros raidės susitarimai galioja ir kitoms komandoms, kurias kreipiasi į atmintį.

Jeigu bus bandoma patalpinti iš atminties į steką, o stekas jau yra pilnas, tai koprocesoriuje kils ypatinga situacija – steko perpildymas.

Jeigu nėra vartotojiškos steko perpildymo apdorojimo programos, tai standartinė programa patalpinamą reikšmę pažymės kaip neapibrėžtą ir, toliau naudojant tokią reikšmę, rezultato neapibrėžtumas tęsiasi, ir koprocesorius tą fiksuoja, kad situacija neliktų nepastebėta.

Ketvirtas pakrovimo komandos tipas (jau buvo minėtos komandos FILD, FBLD ir FLD, kai jos ima atitinkamų formatų duomenis iš atminties) – iš steko nukopijuoti į steko viršūnę, kartu padidinant steko rodyklę:

FLD ST(0) – dubliuojama steko viršūnė;

FLD ST(3) – trečiasis steko elementas kopijuojamas į steko viršūnę, o pats tampa ketvirtuoju steko elementu.

Beveik prieš kiekvieną koprocesoriaus komandą yra naudojama komanda WAIT = 9Bh, tam kad nepradėti vykdyti sekančios koprocesoriaus komandos, kol neįvykdyta ankstesnė.

1101 lxxx mod yyy r/m

Koprocesoriaus komandos yra formuojamos komandos ESC pagalba. Formaliai, komanda ESC turi du operandus, kurių vienas nurodo pačią koprocesoriaus komandą, o kitas – operandą atmintyje. Komanda ESC gali užimti 2, 3, arba 4 baitus, priklausomai nuo lauko *mod* reikšmės.

Pakrovimo komandų variantai:

1. sveikas/slankus → ST(0) (iš atminties į ST(0))  
ESC MF 1 mod 000 r/m [j.b. v.b.]



## KOMPIUTERIŲ ARCHITEKTŪRA

Pastaba. ESC = 11011

$$MF = \begin{cases} 00, & 32\text{-jų bitų slankus – trumpas realus} \\ 01, & 32\text{-jų bitų sveikas – trumpas sveikas} \\ 10, & 64\text{-ių bitų slankus – ilgas realus} \\ 11, & 16\text{-os bitų sveikas – sveikas žodis} \end{cases}$$

Pastaba. Priklausomai nuo to, ar operandas bus slankus ar sveikas, bus sugeneruota FLD arba FILD komanda.

2. ilgas sveikas  $\rightarrow$  ST(0)

ESC 111 mod 101 r/m [j.b. v.b.]

3. vidinis realus  $\rightarrow$  ST(0)

ESC 011 mod 101 r/m [j.b. v.b.]

4. dešimtainis supakuotas  $\rightarrow$  ST(0)

ESC 111 mod 100 r/m [j.b. v.b.]

5. ST(i)  $\rightarrow$  ST(0)

ESC 001 11 000 ST(i)

Užrašymo komandos turi du variantus. Pirmasis komandos variantas užrašo į atmintį, nekeičiant steko, o antrasis komandos variantas užrašo į atmintį, kartu išstumiant iš steko. Pastarasis variantas yra platesnis, tuo tarpu pirmajame dalyvauja ne visi formatai, o keturi, kuriuos galima nurodyti bitais MF:

Duomenų formatas	Lauko ilgis	Užrašymo komanda
Sveikas žodis	16 bitų	FIST
Sveikas trumpas	32 bitai	
Trumpas realus	32 bitai	FST
Ilgas realus	64 bitai	

Užrašymo komandos formatai

1. ST(0)  $\rightarrow$  sveikas/slankus (į atmintį)

ESC MF 1 mod 010 r/m [j.b. v.b.]

2. ST(0)  $\rightarrow$  ST(i)

ESC 101 11 010 ST(i)

## KOMPIUTERIŲ ARCHITEKTŪRA

Antrasis komandos variantas su išstūmimu iš steko: FSTP, FISTP, FBSTP.

Duomenų formatas	Lauko ilgis	Užrašymo komanda
Sveikas žodis	16 bitų	FISTP
Sveikas trumpas	32 bitai	
Sveikas ilgas	64 bitai	
Dešimtainis supakuotas	80 bitų	FBSTP
Trumpas realus	32 bitai	FSTP
Ilgas realus	64 bitai	
Vidinis realus	80 bitų	

Pastaba. Tikrai pakrovimo ir užrašymo (įsiminimo) komandos palaiko visus duomenų formatus. Kitos komandos apsiriboja keturiais formatais, kuriuos galima nurodyti bitais MF:

$$MF = \begin{cases} 00, & 32\text{-jų bitų trumpas slankus} \\ 01, & 32\text{-jų bitų trumpas sveikas} \\ 10, & 64\text{-ių bitų ilgas slankus} \\ 11, & 16\text{-os bitų sveikas žodis} \end{cases}$$

Užrašymo komandos su išstūmimu iš steko formatai:

1. ST(0) → sveikas/slankus (į atmintį)

ESC MF1 mod 011 r/m [j.b. v.b.]

2. ST(0) → ilgas sveikas (į atmintį)

ESC 111 mod 111 r/m [j.b v.b.]

3. ST(0) → vidinis slankus (į atmintį)

ESC 011 mod 111 r/m [j.b v.b.]

4. ST(0) → dešimtainis (į atmintį)

ESC 111 mod 110 r/m [j.b v.b.]

5. ST(0) → ST (i)

ESC 101 11 011 ST(i)

Apsikeitimo komanda FXCH ST(i) apkeičia vietomis steko viršūnę su steko registru:

ST(0) ↔ ST(i)

ESC 001 11 001 ST(i)

Kitos duomenų persiuntimo komandos naudojamos konstantų formavimui, nusiunčiant jas į ST(0), suprantama, vidiniu formatu.

Komandos mnemonika	Reikšmė	Komandos formatas	Šešiolyktainis kodas
FLDZ	0	ESC001 1110 1110	D9 EE

## KOMPIUTERIŲ ARCHITEKTŪRA

FLD1	1	ESC001 1110 1000	D9 E8
FLDPI	II	ESC001 1110 1011	D9 EB
FLDL2T	$\log_2 10$	ESC001 1110 1001	D9 E9
FLDL2E	$\log_2 e$	ESC001 1110 1010	D9 EA
FLDLG2	$\log_{10} 2$	ESC001 1110 1100	D9 EC
FLDLN2	$\log_e 2$	ESC001 1110 1101	D9 ED

Duomenų persiuntimo komandų pavyzdys:

```
CODE    SEGMENT
        ASSUME CS : CODE, DS : CODE
WORD_INTEGER    LABEL WORD
SHORT_INTEGER   LABEL DWORD
LONG_INTEGER     LABEL QWORD
BCD_INTEGER     LABEL TBYTE
SHORT_REAL      LABEL DWORD
LONG_REAL       LABEL QWORD
T_REAL         LABEL TBYTE
```

;Jeigu tai ne makrokomanda, tai prieš kiekvieną jų reikalinga komanda  
;WAIT

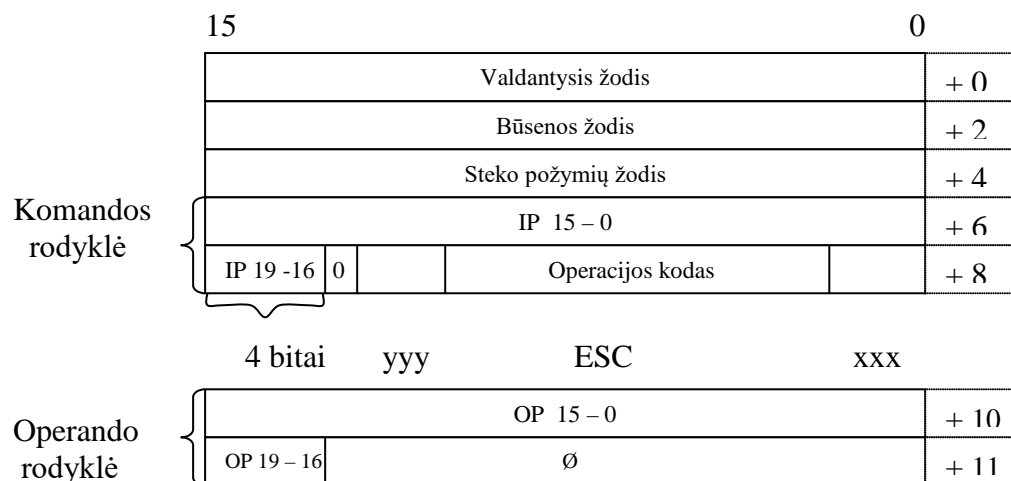
```
FILD WORD_INTEGER
FILD SHORT_INTEGER
FILD LONG_INTEGER
FBLD BCD_INTEGER
FLD SHORT_REAL
FLD LONG_REAL
FLD T_REAL
FLD ST(2)
FIST WORD_INTEGER
FIST SHORT_INTEGER
FST SHORT_REAL
FST LONG_REAL
FST ST(2)
FISTP WORD_INTEGER
FISTP SHORT_INTEGER
FISTP LONG_INTEGER
FBSTP BCD_INTEGER
FSTP SHORT_REAL
FSTP LONG_REAL
FSTP T_REAL
FSTP ST(2)
FXCH ST(2)
FLDZ
FLD1
FLDPI
FLDL2T
FLDL2E
FLDLG2
FLDLN2
```

```
CODE ENDS
END
```

## Koprocesoriaus valdymo komandos

	FINIT – INITiate – inicializuoti koprocesorių (registrų išvalymas)
	FENI – ENable Interrupts – leisti pertraukimus ~ STI
	FDISI – DISable Interrupts – uždrausti pertraukimus ~ CLI
	FLDCW – LoAD Control Word – pakrauti iš atminties valdantįjį žodį
	FSTCW – STore Control Word – įsiminti valdantįjį žodį atmintyje
	FSTSW – STore Status Word – įsiminti būsenos žodį atmintyje
	FCLEX – CLear EXception – numesti ypatingą situaciją
14 baitų	FSTENV – STore ENVironment – įsiminti koprocesoriaus darbinę aplinką atmintyje
	FLDENV – LoAD ENVironment – pakrauti koprocesoriaus darbinę aplinką iš atminties
94 baitai	FSAVE – įsiminti koprocesoriaus būseną atmintyje
	FRSTOR – ReSTORe – pakrauti koprocesoriaus būseną iš atminties
	FINCSTP – INCrement Stack Pointer – padidinti steko rodyklę
	FDECSTP – DECrement Stack Pointer – sumažinti steko rodyklę
	FFREE – atlaisvinti steko registrą
	FNOP – No OPeration – nieko nedaryti
	FWAIT – analogiška komandai WAIT

Koprocesoriaus darbinę aplinką sudaro visi procesoriaus 8087 registrai, išskyrus steko registrus. Aplinka susideda iš 14 baitų. Darbo aplinką į atmintį galima užrašyti taip:



Darbo aplinka yra įsimenama apdorojant ypatingą situaciją koprocesoriuje. Vienas 20 bitų adresas – komandos rodyklė – nurodo paskutinę komandą, kurią įvykdė koprocesorius. Kitas 20 bitų adresas – operando rodyklė – nurodo paskutinio nuskaityto iš atminties duomenų baito adresą. Paskutinės įvykdytos komandos operacijos kodas irgi yra darbo aplinkoje.

Koprocesoriaus būseną yra darbo aplinka, papildyta steko registrais. Todėl būseną užima 94 baitus atmintyje. Steko registrai yra įsimenami po aplinkos, pradedant ST(0), įsimenami visi ir likę registrai: ST(1), ..., ST(7).

## KOMPIUTERIŲ ARCHITEKTŪRA

Steko rodyklė ir steko registrų požymiai kinta skirtingai. Komandos FINCSTP ir FDECSTP keičia steko rodyklę, bet nekeičia steko registrų požymių. Duomenų buvimą steke rodo steko registrų požymiai.

Komanda FFREE atlaisvina steko registrą nurodant steko registrų požymyje, kad tas registras yra laisvas. Tačiau FFREE nekeičia steko rodyklės. Aritmetinės komandos veikia pagal steko principą.

<b>Komandos mnemonika</b>	<b>Komandos formatas</b>	<b>Šešiolyktainis kodas</b>
FINIT	ESC 011 1110 0011	DB E3
FENI	ESC 011 1110 0000	DB E0
FDISI	ESC 011 1110 0001	DB E1
FLDCW	ESC 001 mod 101 r/m [poslinkis]	D9 xx
FSTCW	ESC 001 mod 111 r/m [poslinkis]	D9 xx
FSTSW	ESC 101 mod 111 r/m [poslinkis]	D9 xx
FCLEX	ESC 011 1110 0010	DB E2
FSTENV	ESC 001 mod 110 r/m [poslinkis]	D9 xx
FLDENV	ESC 001 mod 100 r/m [poslinkis]	D9 xx
FSAVE	ESC 101 mod 110 r/m [poslinkis]	DD xx
FRSTOR	ESC 101 mod 100 r/m [poslinkis]	DD xx
FINCSTP	ESC 001 1111 0111	D9 F7
FDECSTP	ESC 001 1111 0110	D9 F6
FFPEE	ESC 101 11 000 ST(i)	DD Cx
FNOP	ESC 001 1101 0000	D9 D0
FWAIT	1001 1011	9B

Koprosoriaus valdymo komandų pavyzdys:

```
CODE    SEGMENT
        ASSUME CS : CODE, DS : CODE
STATUS_WORD    LABEL WORD
CONTROL_WORD   LABEL WORD
ENVIRONMENT    LABEL BYTE ;14 baitų sritis
STATE          LABEL BYTE ;94 baitų sritis
        FINIT
        FENI
```

## KOMPIUTERIŲ ARCHITEKTŪRA

```
FDISI
FLDCW CONTROL_WORD
FSTCW CONTROL_WORD
FCLEX
FSTENV ENVIRONMENT
FLDENV ENVIRONMENT
FSAVE STATE
FRSTOR STATE
FINCSTP
FDECSTP
FFPEE ST(2)
FNOP
FWAIT
CODE ENDS
END
```

### **Skaitmeninio apdorojimo komandos**

Skaitmeninio apdorojimo komandų grupę sudaro :

1. Aritmetinės komandos: sudėtis, atimtis, daugyba, dalyba, palyginimas.
2. Funkcijų komandos: paprastosios, trigonometrinės ir transcendentinės funkcijos.

Pagrindinės aritmetinės komandos yra:

ADD *rezultatas*  $\leftarrow$  *rezultatas* + *šaltinis*  
SUB *rezultatas*  $\leftarrow$  *rezultatas* – *šaltinis*  
SUBR *rezultatas*  $\leftarrow$  *šaltinis* – *rezultatas*  
MUL *rezultatas*  $\leftarrow$  *rezultatas* · *šaltinis*  
DIV *rezultatas*  $\leftarrow$  *rezultatas* / *šaltinis*  
DIVR *rezultatas*  $\leftarrow$  *šaltinis* / *rezultatas*

Pastaba. Aukščiau pateikti komandų pavadinimai nėra koprosesoriaus komandų mnemonika, tai yra daugiau bendriniai komandų pavadinimai.

Dėl atimties ir dalybos nekomutatyvumo reikalingos specialios operacijos, kuriose operandai yra sukeisti vietomis.

Sudėties komandos variantai:

1. FADD ST(i)  $ST(0) \leftarrow ST(0) + ST(i)$   
FADD ST(0), ST(i)  $ST(0) \leftarrow ST(0) + ST(i)$   
FADD ST(i), ST(0)  $ST(i) \leftarrow ST(0) + ST(i)$

2. FADDP ST(i), ST(0)  $ST(i) \leftarrow ST(0) + ST(i)$

Atlikus šią operaciją, steko viršūnė ST(0) išstumiamas. Todėl ji negali būti rezultato registru. Jeigu ST(i) yra ST(1), tai turime klasikinę stekinę operaciją (imami du viršutiniai steko elementai, sudedami, išstumiami iš steko, o gautas rezultatas užrašomas steko viršūnėje).

3. FADD *real*  $ST(0) \leftarrow ST(0) + real\ mem$

## KOMPIUTERIŲ ARCHITEKTŪRA

*real* – tai trumpas arba ilgas realus skaičius.

*real mem* reiškia, kad realus skaičius yra atmintyje.

4. FIADD *integer*                       $ST(0) \leftarrow ST(0) + integer$

*integer* – tai sveikas žodis arba trumpas sveikas.

Aritmetinės komandos negali naudoti tiesiogiai atmintyje ilgo sveiko, supakuoto dešimtainio ir vidinio realaus formato. Prieš atliekant veiksmus su jais, reikėtų šių formatų duomenis nusiųsti į koprosesoriaus steką.

Duomenų formatas	Lauko ilgis	Sudėties komanda
Sveikas žodis	16 bitų	FIADD
Sveikas trumpas	32 bitai	
Trumpas realus	32 bitai	FADD
Ilgas realus	64 bitai	

Sudėties komandos formatai:

1.  $ST(0) \leftarrow ST(0) + \text{sveikas/slankus atmintyje}$   
ESC MF 0 mod 000 r/m [poslinkis]

2                       $\leftarrow ST(0) + ST(i)$   
ESC d P 0 11000 ST(i)

$$d = \begin{cases} 0, & \text{rezultatas yra } ST(0) \\ 1, & \text{rezultatas yra } ST(i) \end{cases}$$

$$P = \begin{cases} 0, & \text{viršutinio steko elemento po operacijos nereikia atlaisvinti} \\ 1, & \text{viršutinis steko elementas po operacijos atlaisvinamas} \end{cases}$$

Atimties komandos variantai:

1. FSUB
2. FSUBP – su išstūmimu iš steko.
3. FSUBR – operandai atvirkščia tvarka.
4. FISUB – operandai sveiki atmintyje.
5. FISUBR – operandai sveiki atmintyje, atvirkščia tvarka.
6. FSUBRP – operandai registrai atvirkščia tvarka su išstūmimu iš steko.

Duomenų formatas	Lauko ilgis	Atimties komanda
Sveikas žodis	16 bitų	FISUB, FISUBR
Sveikas trumpas	32 bitai	
Trumpas realus	32 bitai	FSUB,

## KOMPIUTERIŲ ARCHITEKTŪRA

Ilgas realus	64 bitai	FSUBR
--------------	----------	-------

Atimties komandos formatai:

1.  $ST(0) \leftarrow \text{operandas1} - \text{operandas2}$   
ESC MF 0 mod 10 R r/m [poslinkis]

$$R = \begin{cases} 0, & \text{rezultatas operacija šaltinis} \\ 1, & \text{šaltinis operacija rezultatas} \end{cases}$$

2.  $\leftarrow ST(0) - ST(i)$   
ESC d P 0 111 0 R ST(i)

Daugybos komandos variantai:

1. FMUL
2. FMULP
3. FIMUL

Duomenų formatas	Lauko ilgis	Daugybos komanda
Sveikas žodis	16 bitų	FIMUL
Sveikas trumpas	32 bitai	
Trumpas realus	32 bitai	FMUL
Ilgas realus	64 bitai	

Daugybos komandos formatai:

1.  $ST(0) \leftarrow ST(0) \cdot \text{op/mem}$   
sveikas/slankus  
ESC MF 0 mod 001 r/m [poslinkis]

2.  $\leftarrow ST(0) \cdot ST(i)$   
ESC d P 0 11 001 ST(i)

Dalybos komandos variantai:

1. FDIV
2. FDIVP
3. FDIVR
4. FIDIV
5. FIDIVR
6. FDIVRP

Duomenų formatas	Lauko ilgis	Dalybos komanda
------------------	-------------	-----------------



## KOMPIUTERIŲ ARCHITEKTŪRA

Sveikas žodis	16 bitų	FIDIV, FIDIVR
Sveikas trumpas	32 bitai	
Trumpas realus	32 bitai	FDIV, FDIVR
Ilgas realus	64 bitai	

Dalybos komandos formatai:

1.  $ST(0) \leftarrow ST(0) / ST(i)$   
ESC MF 0 mod 11R r/m [poslinkis]
2.  $\leftarrow operandas1 / operandas2$   
ESC d P 0 1111 R ST(i)

Aritmetinių komandų pavyzdys:

```
CODE    SEGMENT
        ASSUME CS : CODE, DS : CODE
WORD_INTEGER    LABEL WORD
SHORT_INTEGER   LABEL DWORD
SHORT_REAL      LABEL DWORD
LONG_REAL       LABEL QWORD
        FADD ST(0), ST(2)
        FADD ST(2), ST(0)
        FIADD WORD_INTEGER
        FIADD SHORT_INTEGER
        FADD SHORT_REAL
        FADD LONG_REAL
        FADDP ST(2), ST(0)
        FSUB SHORT_REAL
        FISUB WORD_INTEGER
        FSUBP ST(2), ST(0)
        FSUBR ST(2), ST(0)
        FISUBR SHORT_INTEGER
        FSUBRP ST(2), ST(0)
        FMUL SHORT_REAL
        FIMUL WORD_INTEGER
        FMULP ST(2), ST(0)
        FDIV ST(0), ST(2)
        FIDIV SHORT_INTEGER
        FDIVP ST(2), ST(0)
        FIDIVR WORD_INTEGER
        FDIVRP ST(2), ST(0)
CODE    ENDS
        END
```

Palyginimo komandos yra naudojamos dviejų skaičių palyginimui. Atliekant komandą yra formuojami du požymiai: C3 ir C0. Palyginime dalyvauja steko viršūnė ir atminties laukas arba steko registras.

<b>C3</b>	<b>C0</b>	<b>Santykis</b>
-----------	-----------	-----------------

## KOMPIUTERIŲ ARCHITEKTŪRA

0	0	ST(0) > šaltinis
0	1	ST(0) < šaltinis
1	0	ST(0) = šaltinis
1	1	ST(0) nepalyginamas su šaltiniu

Po tokio palyginimo būsenos žodį reikia įsiminti atmintyje, vyresniąją dalį nusiųsti į registrą AH ir, panaudojant komandą SAHF, registro AH reikšmę nusiųsti į registrą SF.

Skaiciai yra nepalyginami, kai bent vienas iš jų yra NAN (Not A Number) arba viena iš begalybės reikšmių.

Duomenų formatas	Lauko ilgis	Palyginimo komanda
Sveikas žodis	16 bitų	FICOM, FICOMP
Sveikas trumpas	32 bitai	
Trumpas realus	32 bitai	FCOM, FCOMP
Ilgas realus	64 bitai	

Palyginimo komandos formatai:

### I. Palyginimas (FCOM, FICOM)

1. sveikas/slankus atmintyje *ir* ST(0)  
ESC MF 0 mod 010 r/m [poslinkis]

2. ST(i) *ir* ST(0)  
ESC 000 11010 ST(i)

### II. Palyginimas ir pašalinimas iš steko (FCOMP, FICOMP):

1. ESC MF 0 mod 011 r/m [poslinkis]

2. ESC 000 11011 ST(i)

### III. Komanda FCOMPP – ST(0) ir ST(1) palyginimas ir pašalinimas iš steko du kartus:

1. ESC 110 1101 1001

### IV. Komanda FTST – ST(0) ir 0 palyginimas:

1. ESC 001 1110 0100

Palyginimo komandų pavyzdžiai:

1.  
CODE    SEGMENT  
      ASSUME CS : CODE, DS : CODE

## KOMPIUTERIŲ ARCHITEKTŪRA

```

WORD_INTEGER      LABEL WORD
SHORT_INTEGER     LABEL DWORD
SHORT_REAL        LABEL DWORD
LONG_REAL         LABEL QWORD
                  FCOM ST(2)
                  FICOM WORD_INTEGER
                  FCOM SHORT_REAL
                  FICOMP SHORT_INTEGER
                  FCOMP LONG_REAL
                  FCOMPP
                  FTST

CODE  ENDS
      END

2.
CODE  SEGMENT
      ASSUME CS : CODE, DS : CODE
WORD_INTEGER      LABEL WORD
STATUS_WORD       DW ?
                  FICOM WORD_INTEGER ;palyginimas su STOS
                  FSTSW STATUS_WORD ;būsenos įsiminimas
                  FWAIT
                  MOV AH, BYTE PTR STATUS_WORD + 1
                  SAHF
                  JB  CONTINUE ;C0 patikrinimas ir perėjimas jei 0
                  JNE ST_SPEATER ;C3 patikrinimas
ST_EQUAC ;čia nueinama, jei C3=1 ir C0=0
ST_SPEATER ;čia nueinama, jei C3=0 ir C0=0
          ;ST(0) > WORD_INTEGER
CONTINUE:
          JNE ST_LESS ;patikrinamas C3
UNORDERED:
          ;čia patenkama, jei C3=1 ir C0=1
          ;nepalyginami
ST_LESS:
          ;čia patenkama, jei C3=0 ir C0=1
          ;ST(0) < WORD_INTEGER

CODE  ENDS
      END

```

Palyginimo komandos formuoja būsenos žodžio bitus (ypatingas situacijas). Jeigu šie bitai yra nesvarbūs, reiktų įvykdyti komandą FCLEX – išvalyti požymius.

Yra ir speciali palyginimo komanda FXAM, kuri pagal steko viršūnėje esančią reikšmę suformuoja keturis sąlygos požymius (C0, C1, C2, C3) ir parodo, kokio tipo skaičius yra steko viršūnėje:

C3	C2	C1	C0	ST(0)
0	0	0	0	+ nenormalizuotas
0	0	0	1	+ NAN
0	0	1	0	- nenormalizuotas
0	0	1	1	- NAN
0	1	0	0	+ normalizuotas
0	1	0	1	+ begalybė
0	1	1	0	- nenormalizuotas

## KOMPIUTERIŲ ARCHITEKTŪRA

0	1	1	1	- begalybė
1	0	0	0	+ 0
1	0	0	1	tuščia
1	0	1	0	- 0
1	0	1	1	tuščia
1	1	0	0	+ denormalizuota
1	1	0	1	tuščia
1	1	1	0	- denormalizuota
1	1	1	1	tuščia

Paprastai komandos FXAM nereikia naudoti, ji reikalinga, kai dirbama ant tikslumo ribos.

Begalybė reiškia, kad eilė yra per didelė.

Nenormalizuotas (denormalizuotas) skaičius yra tada, kai eilė yra per maža tam, kad ji būtų užrašyta korektiškai, ir todėl rezultatas užrašomas denormalizuotai, mantinės sąskaita.

FXAM:       ESC 001 1110 0101

### Paprastos funkcijos

FSQRT – kvadratinė šaknis iš ST(0)

ESC 001 1111 1010

FSCALE – ST(0) „maštabavimas“ pagal ST(1)

ESC 001 1111 1101

Komanda atlieka kėlimą laipsniu:  $ST(0) \cdot 2^{ST(1)}$

FPREM – liekana  $ST(0) / ST(1)$

ESC 001 1111 1000

Komanda atlieka veiksmą:  $ST(0) \leftarrow ST(0) \bmod ST(1)$

FRNDINT – ST(0) išlyginimas iki sveiko (apvalinimo taisyklė pagal RC požymius)

ESC 001 1111 1100

FXTRACT – trupmeninė dalis nuo ST(0) užrašoma į naują steko viršūnę

ESC 001 1111 0100

Komanda atlieka charakteristikos ir mantinės išskyrimą.

FABS – ST(0) tai skaičiaus absoliuti reikšmė buvusi ST(0)

ESC 001 1110 0001

FCHS – ženklo keitimas

ESC 001 1110 0000

## KOMPIUTERIŲ ARCHITEKTŪRA

### Trigonometrinės funkcijos

FPTAN – argumentas ST(0) skaičius radianais:  $0 < \alpha < \pi / 4$

Rezultatas  $y / x = \tan \alpha$

y užima steko viršūnę, o paskui x užrašomas į naują steko viršūnę.

ESC 001 1111 0010

FPATAN – ST(0)  $\leftarrow \arctan y/x = \arctan ST(1) / ST(0)$

ST(0) = x išstumiamas iš steko, o FPATAN užrašomas vietoje y.

ESC 001 1111 0011

### Transcendentinės funkcijos

F2XM1 =  $2^{ST(0)} - 1$

ESC 001 1111 0000

FYL2X =  $ST(1) \cdot \log_2 [ST(0)]$

ESC 001 1111 0001

FYL2XP1 =  $ST(1) \cdot \log_2 [ST(0) + 1]$

ESC 001 1111 1001

## **7. Literatūra**

1. Peter Abel. IBM PC Assembly Language and Programming. Third Edition. Prentice Hall International, Inc., 1995.
2. Randall Hyde. The Art of Assembly Language, 2000.  
[http://webster.cs.ucr.edu/Page\\_asm/ArtofAssembly/0\\_ArtofAsm.html](http://webster.cs.ucr.edu/Page_asm/ArtofAssembly/0_ArtofAsm.html)