

Turinys

Turinys.....	1
1. Pirmoji programa assembleriu	2
1.1. Programos struktūra.....	3
1.2. Programavimo sistema	3
2. Procesorius 8086	4
2.1. Registrai.....	4
2.2. Atminties adresavimas.....	4
2.3. Komandų sistema	5
2.3.1. Duomenų perdavimo komandos.....	5
2.3.2. Aritmetinės komandos.....	6
2.3.3. Loginės komandos.....	7
2.3.4. Valdymo perdavimo komandos.....	7
2.3.5. Procesoriaus valdymo komandos	8
2.3.6. Komandos darbui su eilutėmis	9
2.4. Pavyzdžiai.....	11
3. Eilučių įvedimas, išvedimas, apdorojimas	12
4. Failai	17
4.1. Failo atidarymas (funkcija 3Dh).....	18
4.2. Failo skaitymas (funkcija 3Fh)	19
4.3. Failo rašymas (funkcija 40h)	19
4.4. Pozicijos faile keitimas (funkcija 42h)	19
4.5. Failo uždarymas (funkcija 3Eh)	20
4.6. Failo sukūrimas (funkcija 3Ch)	20
4.7. Failo sunaikinimas (funkcija 41h)	20
5. Programos segmento prefiksas. Komandinė eilutė.....	21
6. Pertraukimai	23

1. Pirmoji programa assembleriu

- ❑ Pavyzdys *Hello.asm* – 16-os bitų MS-DOS programa (panašias, tik rimtesnes, reikės rašyti pratybų metu):

```
; Programa isveda i ekrana "Hello, World!"
; (komentarai pradedami kabliataskiu)

.MODEL small                ; atminties modelis

.STACK 100h                 ; steko dydis

.DATA                      ; duomenų segmentas
hello_message db 'Hello, World!',0dh,0ah,'$'

.CODE                      ; kodo segmentas
strt:
    mov     ax,@data        ; ds registro inicializavimas
    mov     ds,ax

    mov     ah,9            ; eilutes isvedimas
    mov     dx,offset hello_message
    int     21h

    mov     ax,4C00h        ; programos darbo pabaiga
    int     21h

end strt
```

Programa surenkama tekstiniu redaktorium (tinka ir Notepad), išsaugoma faile *Hello.asm*, kompiliuojama komanda `tasm Hello.asm`, linkuojama komanda `tlink Hello.obj`, po šio žingsnio gaunamas vykdomas failas *Hello.exe*.

- ❑ Pavyzdys *HelloWin.asm* – 32-jų bitų Windows programa (įdomumo dėlei, juk assembleriu galima programuoti ir Windows operacinei sistemai):

```
; Programa parodo langa su uzrasu "Hello, World!"

.386                      ; reikia bent 386 procesoriaus
.MODEL flat               ; atminties modelis
option casemap:none       ; skirti didziasias ir mazasias
                          ; raides funkciju varduose

EXTERN MessageBoxA:PROC    ; funkcija parodanti pranesima
EXTERN ExitProcess:PROC   ; funkcija uzbaiigianti programa

includelib import32.lib    ; biblioteka su tu funkciju
                          ; realizacijom

.DATA
langas      db "My First Window",0h
tekstas     db "Hello, World!",0h

.CODE
start:
    push 0                ; langa tipas
    push offset langas    ; langa antraste
    push offset tekstas   ; rodomas tekstas
    push 0
    call MessageBoxA      ; parodyti pranesima

    push 0
    call ExitProcess      ; baigti darba

end start
```

Programa surenkama tekstiniu redaktorium (tinka ir Notepad), išsaugoma faile HelloWin.asm, kompiliuojama komanda `tasm32 HelloWin.asm`, linkuojama komanda `tlink32 HelloWin.obj`, po šio žingsnio gaunamas vykdomas failas HelloWin.exe.

1.1. Programos struktūra

Programa assembleriu paprastai turi tokią (ar panašią) struktūrą:

- Antraštės blokas

Tai - komentarai, kuriuose parašoma, kam programa skirta, kas autorius ir pan.

- Direktyvos kompiliatoriui

IDEAL

Borland Turbo Assembler gali dirbti dviem režimais: MASM (pagal nutylėjimą) ir IDEAL. Skirtumai yra iš esmės tik programos sintaksėje. Pvz., kodo segmento pradžia MASM stilium nurodoma „CODE“ direktyva, IDEAL – „CODESEG“.

.MODEL modelis

Atminties modelis. Nurodo, kiek atminties daugiausiai gali būti skiriama kodui ir duomenims. Gali būti *small* (64K+64K), *medium* (1M+64K), *compact* (64K+1M), *large* (1M+1M). Apribojimai atminties dydžiui kyla iš naudojamo adresavimo tipo: artimas (vieno segmento ribose) ar tolimas (segmentas nurodomas).

- Konstantos

Apibrėžiamos konstantos ir makrokomandos.

- Duomenų segmentas

Pradžia žymima .DATA arba DATASEG (IDEAL režimo sintaksė). Duomenims vieta skiriama DB (baitas), DW (žodis) ir pan. direktyvom.

- Kodo segmentas

Pradžia žymima .CODE arba CODESEG (IDEAL režimo sintaksė). Tai – blokas assemblerio operatorių, turinčių pavidalą

[žymė] mnemonika [operandai] [; komentaras]

Vienas operatorius – viena eilutė. Jokių blokų (*begin .. end* ar pan.) nėra.

Pirmosios kodo eilutės paprastai pasiunčia į registrą ds (ir es, jei reikia) duomenų segmento pradžios adresą. Procesoriaus segmentiniai registrai cs ir ss, nurodantys kodo ir steko segmento pradžias, programos vykdymo pradžioje jau būna inicializuoti (t.y. jiems jau būna suteiktos reikiamos pradinės reikšmės). Kitaip yra su ds ir es registrais. Kadangi dėl 8086 architektūros ypatybių į segmentinius registrus negalima pasiųsti konstantos tiesiogiai, tenka tai atlikti per kitą registrą ar steką.

Programos vykdymas baigiamas grąžinant valdymą operacinei sistemai. Tai paprastai atliekama funkcija `int 21,4Ch`.

Programa baigiama direktyva END, po kurios nurodomas įėjimo į programą taškas. Paprastai tai būna pirmąją komandą žyminti žymė.

1.2. Programavimo sistema

Programuojant assembleriu būtinas tekstinis redaktorius, galintis išsaugoti failą TXT formatu (ne RTF ar HTML), kompiliatorius ir ryšių redaktorius (*linker*):

- | | | |
|---|---------------------|----------------------|
| 1) Redaktorius: | → <i>myprog.asm</i> | - programos tekstas |
| 2) Kompiliatorius: <i>tasm myprog</i> | → <i>myprog.obj</i> | - objektinis modulis |
| 3) Ryšių redaktorius: <i>tlink myprog</i> | → <i>myprog.exe</i> | - vykdoma programa |

2. Procesorius 8086

- Tikslas: susipažinti su procesoriaus registrais, komandų sistema ir atminties adresavimu

2.1. Registrai

Procesorius 8086 turi šiuos 16 bitų registrus:

- Bendros paskirties (ax, bx, cx, dx), prieinamus žemesnėmis (al, bl, cl, dl) ir aukštesnėmis (ah, bh, ch, dh) pusėmis
- Rodyklinius registrus sp ir bp
- Indeksinius registrus si ir di
- Segmentinius registrus cs, ds, ss, es
- Komandų skaitiklį ip
- Požymių registrą. 16 bitų registre naudojami 9 požymiai (of-perpildymo, df-krypties, if-pertraukimų leidimo, tf-trasavimo, sf-ženklų, zf-nulio, af-papildomą, pf-lyginumo, cf-pernešimo)

ax	ah	al	cs	
bx	bh	bl	ds	
cx	ch	cl	ss	
dx	dh	dl	es	
si			sp	
di			bp	
flags	0 x x x OF DF IF TF	SF ZF 0 AF 0 PF 1 CF	ip	
	15	8 7 0	15	0

Požymis	Paiškinimas
Overflow Flag (OF)	1 kai įvyko perpildymas (su ženklu), pvz. baitas 100 + 50
Direction Flag (DF)	Naudojamas duomenų (eilučių) apdorojimo komandose, 0 - apdorojimas pirmyn, 1 - atgal
Interrupt Flag (IF)	Kai nustatytas 1, procesorius reaguoja į pertraukimus
Trap Flag (TF)	Naudojamas komandų vykdymui pažingsniui
Sign Flag (SF)	1 kai rezultatas yra neigiamas (t.y. vyriausio bito reikšmė)
Zero Flag (ZF)	1 kai rezultatas yra 0
Auxiliary Flag (AF)	1 kai įvyko perpildymas (be ženklo) žemesniuose 4 bituose, pvz. 0fh + 1
Parity Flag (PF)	1 kai žemesniuose 8 rezultato bituose vienetukų skaičius yra lyginis
Carry Flag (CF)	1 kai įvyko pernešimas (be ženklo), pvz. baitas 255 + 1

2.2. Atminties adresavimas

Procesorius 8086 atminties adresavimui naudoja septynis būdus.

Adresavimo būdas	Pavyzdžiai	Pastaba
Tiesioginis	mov ax,[count]	Adresuojama pagal poslinkį nuo segmento pradžios
Netiesioginis	mov ax,[bx] dec [BYTE si]	Kaip rodyklė naudojamas bx, si arba di registras
Bazinis	mov ax,[record+bp]	Registrai bx ir bp naudojami kaip bazė. bx adresuoja atžvilgiu ds, bp - ss.

Indeksinis	mov ax,[array+si]	Identiškas baziniam, tik naudojami indeksiniai registrai. Adresuojama atžvilgiu ds.
Bazinis- indeksinis	mov ax,[recordArray+bx+si]	Naudojami du registrai ir galbūt poslinkis. Galingiausias būdas, tinkamas sudėtingoms struktūroms adresuoti.
Eilutinis	lodsw	
Įvedimo/išvedimo portų	in ax,dx	

Adresuojant netiesiogiai, poslinkis nuo segmento pradžios nurodomas registrais bx, bp (steko segmentui), si ir di. Jokių kitų registrų tam naudoti negalima.

2.3. Komandų sistema

Procesoriaus 8086 komandos pagal atliekamas funkcijas skirstomos į 6 grupes:

- duomenų perdavimo komandos
- aritmetinės komandos
- loginės komandos
- valdymo perdavimo (perėjimų) komandos
- procesoriaus valdymo komandos
- komandos darbui su eilutėmis

2.3.1. Duomenų perdavimo komandos

Mnemonika	Aprašymas	Požymiai ¹⁾									
		o	d	i	t	s	z	a	p	c	
Pagrindinės											
mov dest,src	Persiunčia (kopijuoja) baitą ar žodį dest←src	-	-	-	-	-	-	-	-	-	
pop dest	Paima žodį iš steko	-	-	-	-	-	-	-	-	-	
push src	Padedą žodį į steką	-	-	-	-	-	-	-	-	-	
xchg dest,src	Sukeičia baitus / žodžius dest←→src	-	-	-	-	-	-	-	-	-	
xlat / xlatb tbl	Perkoduoja lentelę al←[bx+al]	-	-	-	-	-	-	-	-	-	
Įvedimo / išvedimo (skaitymas iš portų / rašymas į portus)											
in acc,port	Įveda baitą arba žodį iš porto	-	-	-	-	-	-	-	-	-	
out port,acc	Išveda baitą arba žodį į portą	-	-	-	-	-	-	-	-	-	
Adresinės (atlieka adresų įkrovimą)											
lds dest,src	Įkrauna duomenų segmento registrą	-	-	-	-	-	-	-	-	-	
lea dest,src	Įkrauna efektyvų adresą	-	-	-	-	-	-	-	-	-	
les dest,src	Įkrauna papildomo segmento registrą	-	-	-	-	-	-	-	-	-	
Požymių (požymių registro skaitymas / rašymas)											
lahf	Padedą požymių registrą į AH	-	-	-	-	-	-	-	-	-	
popf	Paima požymių registro reikšmę iš steko	*	*	*	*	*	*	*	*	*	
pushf	Padedą požymių registro reikšmę į steką	-	-	-	-	-	-	-	-	-	
sahf	Nustato požymių registra pagal AH	-	-	-	-	*	*	*	*	*	

2.3.2. Aritmetinės komandos

Mnemonika	Aprašymas	Požymiai ¹⁾									
		o	d	i	t	s	z	a	p	c	
Sudėtis											
aaa	ASCII formato korekcija sudėčiai 08h+04h=0Ch→0102h	u	-	-	-	u	u	*	u	*	
adc dest,src	Sudėtis su pernešimu dest←dest+src+cf	*	-	-	-	*	*	*	*	*	
add dest,src	Baitų arba žodžių sudėtis dest←dest+src	*	-	-	-	*	*	*	*	*	
daa	Dešimtainė korekcija sudėčiai	u	-	-	-	*	*	*	*	*	
inc dest	Inkrementavimas (padidinimas 1) dest←dest+1	*	-	-	-	*	*	*	*	-	
Atimtis											
aas	ASCII formato korekcija atimčiai	u	-	-	-	u	u	*	u	*	
cmp dest,src	Palyginimas (analogiška sub , tik nekeičia dest)	*	-	-	-	*	*	*	*	*	
das	Dešimtainė korekcija atimčiai	u	-	-	-	*	*	*	*	*	
dec dest	Dekrementavimas (sumažinimas 1) dest←dest-1	*	-	-	-	*	*	*	*	-	
neg dest	Ženklo pakeitimas (papildomas kodas)	*	-	-	-	*	*	*	*	*	
sbb dest,src	Atimtis su pasiskolinimu dest←dest-src-cf	*	-	-	-	*	*	*	*	*	
sub dest,src	Atimtis dest←dest-src	*	-	-	-	*	*	*	*	*	
Daugyba											
aam	ASCII formato korekcija daugybai	u	-	-	-	*	*	u	*	u	
imul src	Daugyba su ženklu ax←src_byte*al dx:ax←src_word*ax	*	-	-	-	u	u	u	u	*	
mul src	Daugyba be ženklo Analogiška imul	*	-	-	-	u	u	u	u	*	
Dalyba											
aad	ASCII formato korekcija dalybai	u	-	-	-	*	*	u	*	u	
cbw	Baito konvertavimas į žodį ax←(signed)al	-	-	-	-	-	-	-	-	-	
cwd	Žodžio konvertavimas į dvigubą žodį dx:ax←(signed)ax	-	-	-	-	-	-	-	-	-	

div src	Beženklė dalyba $al \leftarrow ax \text{ div byte}$ $ax \leftarrow dx:ax \text{ div word}$ $ah \leftarrow ax \text{ mod byte}$ $dx \leftarrow dx:ax \text{ mod word}$	u	-	-	-	u	u	u	u	u
idiv src	Dalyba su ženklu Analogiška div	u	-	-	-	u	u	u	u	u

2.3.3. Loginės komandos

Mnemonika	Aprašymas	Požymiai ¹⁾									
		o	d	i	t	s	z	a	p	c	
Loginės											
and dest,src	Loginis IR	0	-	-	-	*	*	u	*	0	
not dest	Loginis NE	-	-	-	-	-	-	-	-	-	
or dest,src	Loginis ARBA	0	-	-	-	*	*	u	*	0	
test dest,src	Bitų tikrinimas (analogiška and , tik nekeičia dest)	0	-	-	-	*	*	u	*	0	
xor dest,src	Loginis griežtas ARBA (sudėtis modulių 2)	0	-	-	-	*	*	u	*	0	
Postūmio (rotacijos)											
rcl dest,cnt	Ciklinis bitų postūmis į kairę per cf	*	-	-	-	-	-	-	-	*	
rcr dest,cnt	Ciklinis bitų postūmis į dešinę per cf	*	-	-	-	-	-	-	-	*	
rol dest,cnt	Ciklinis bitų postūmis į kairę	*	-	-	-	-	-	-	-	*	
ror dest,cnt	Ciklinis bitų postūmis į dešinę	*	-	-	-	-	-	-	-	*	
sar dest,cnt	Aritmetinis postūmis į dešinę (ženklų bitas nekinta)	*	-	-	-	*	*	u	*	*	
sal dest,cnt	Aritmetinis postūmis į kairę	*	-	-	-	*	*	u	*	*	
shl dest,cnt	Postūmis į kairę (sinonimas sal)	*	-	-	-	*	*	u	*	*	
shr dest,cnt	Postūmis į dešinę (skirtingai nuo sar keičia ženklą)	*	-	-	-	*	*	u	*	*	

2.3.4. Valdymo perdavimo komandos

Mnemonika	Aprašymas	Požymiai ¹⁾									
		o	d	i	t	s	z	a	p	c	
Besąlyginio perėjimo											
call addr	Procedūros iškviatimas push ip(cs) ip(cs)←addr	-	-	-	-	-	-	-	-	-	
jmp addr	Besąlyginis perėjimas ip(cs)←ip(cs)+addr	-	-	-	-	-	-	-	-	-	
ret n	Grįžimas iš procedūros pop ip ir sp←sp+n	-	-	-	-	-	-	-	-	-	
retn n	Grįžimas iš artimos procedūros (sinonimas ret)	-	-	-	-	-	-	-	-	-	

retf n	Grįžimas iš tolimos procedūros pop ip, cs ir sp ← sp+n	-	-	-	-	-	-	-	-	-
<i>Sąlyginio perėjimo</i>										
ja / jnbe addr	Perėjimas, jei cf=0 ir zf=0	-	-	-	-	-	-	-	-	-
jae / jnb addr	Perėjimas, jei cf=0	-	-	-	-	-	-	-	-	-
jb / jnae addr	Perėjimas, jei cf=1	-	-	-	-	-	-	-	-	-
jbe / jna addr	Perėjimas, jei cf=1 arba zf=1	-	-	-	-	-	-	-	-	-
jc addr	Perėjimas, jei cf=1	-	-	-	-	-	-	-	-	-
je / jz addr	Perėjimas, jei zf=1	-	-	-	-	-	-	-	-	-
jg / jnle addr	Perėjimas, jei zf=0 ir sf=of	-	-	-	-	-	-	-	-	-
jge / jnl addr	Perėjimas, jei sf=of	-	-	-	-	-	-	-	-	-
jl / jnge addr	Perėjimas, jei sf!=of	-	-	-	-	-	-	-	-	-
jle / jng addr	Perėjimas, jei zf=1 arba sf!=of	-	-	-	-	-	-	-	-	-
jnc addr	Perėjimas, jei cf=0	-	-	-	-	-	-	-	-	-
jne / jnz addr	Perėjimas, jei zf=0	-	-	-	-	-	-	-	-	-
jno addr	Perėjimas, jei of=0	-	-	-	-	-	-	-	-	-
jnp / jpo addr	Perėjimas, jei pf=0	-	-	-	-	-	-	-	-	-
jns addr	Perėjimas, jei sf=0	-	-	-	-	-	-	-	-	-
jo addr	Perėjimas, jei of=1	-	-	-	-	-	-	-	-	-
jp / jpe addr	Perėjimas, jei pf=1	-	-	-	-	-	-	-	-	-
js addr	Perėjimas, jei sf=1	-	-	-	-	-	-	-	-	-
jcxz addr	Perėjimas, jei cx=0	-	-	-	-	-	-	-	-	-
<i>Ciklų</i>										
loop addr	Ciklas kol cx<>0 dec cx jmp, jei cx!=0	-	-	-	-	-	-	-	-	-
loope(z) addr	Ciklas, jei zf=1 dec cx (nekeičiant požymių) jmp, jei cx!=0 ir zf=1	-	-	-	-	-	-	-	-	-
loopne(z) addr	Ciklas, jei zf=0 dec cx (nekeičiant požymių) jmp, jei cx!=0 ir zf=0	-	-	-	-	-	-	-	-	-
<i>Pertraukimų</i>										
int n	Pertraukimo inicijavimas push flags, cs, ip jmp pagal vektorių	-	-	0	0	-	-	-	-	-
into	Jei of=1, generuojamas pertraukimas 4	-	-	0	0	-	-	-	-	-
iret	Grįžimas iš pertraukimo apdorojimo pop ip, cs, flags	*	*	*	*	*	*	*	*	*

2.3.5. Procesoriaus valdymo komandos

Mnemonika	Aprašymas	Požymiai ¹⁾									
		o	d	i	t	s	z	a	p	c	
Požymių											
clc	Nuvalymas cf	-	-	-	-	-	-	-	-	0	

cld	Nuvalymas df	-	0	-	-	-	-	-	-	-
cld	Nuvalymas if	-	-	0	-	-	-	-	-	-
cmc	Perjungimas cf	-	-	-	-	-	-	-	-	*
stc	Ijungimas cf	-	-	-	-	-	-	-	-	1
std	Ijungimas df	-	1	-	-	-	-	-	-	-
sti	Ijungimas if	-	-	1	-	-	-	-	-	-
<i>Išorinės sinchronizacijos</i>										
esc	Kreipimasis į koprocesorių	-	-	-	-	-	-	-	-	-
hlt	Procesoriaus sustabdymas (iki RESET, NMI ar INTR)	-	-	-	-	-	-	-	-	-
lock	Magistralės blokavimas (naudojama kai procesoriai yra keli)	-	-	-	-	-	-	-	-	-
wait	Perjungimas į laukimo režimą (dirbant su koprocesorium)	-	-	-	-	-	-	-	-	-
<i>Kitos</i>										
nop	Tuščia komanda (sinonimas xchg ax,ax)	-	-	-	-	-	-	-	-	-
	Nedaro nieko; kodas 90h									

2.3.6. Komandos darbui su eilutėmis

Mnemonika	Aprašymas	Požymiai ¹⁾									
		o	d	i	t	s	z	a	p	c	
<i>Eilučių persiuntimo</i>											
lods src	Eilutės baito arba žodžio (nustatoma pagal src) įkrovimas acc←[ds:si] inc/dec si pagal df	-	-	-	-	-	-	-	-	-	
lodsb	Eilutės baito įkrovimas al←[ds:si]; si←si+-1	-	-	-	-	-	-	-	-	-	
lodsw	Eilutės žodžio įkrovimas ax←[ds:si]; si←si+-2	-	-	-	-	-	-	-	-	-	
movs dest,src	Eilutės baito arba žodžio persiuntimas [es:di]←[ds:si] inc/dec si,di pagal df	-	-	-	-	-	-	-	-	-	
movsb	Eilutės baito persiuntimas [es:di]←[ds:si] si←si+-1; di←di+-1	-	-	-	-	-	-	-	-	-	
movsw	Eilutės žodžio persiuntimas [es:di]←[ds:si] si←si+-2; di←di+-2	-	-	-	-	-	-	-	-	-	
stos dest	Baito arba žodžio įrašymas į eilutę [es:di]←acc inc/dec di pagal df	-	-	-	-	-	-	-	-	-	

stosb	Baito įrašymas į eilutę $[es:di] \leftarrow al$ $di \leftarrow di + 1$	-	-	-	-	-	-	-	-
stosw	Žodžio įrašymas į eilutę $[es:di] \leftarrow ax$ $di \leftarrow di + 2$	-	-	-	-	-	-	-	-
<i>Eilučių tikrinimo</i>									
cmps dest,src	Eilučių baitų arba žodžių palyginimas $comp [ds:si], [es:di]$ $inc/dec si, di$ pagal df	*	-	-	-	*	*	*	*
cmpsb	Eilučių baitų palyginimas $comp [ds:si], [es:di]$ $si \leftarrow si + 1; di \leftarrow di + 1$	*	-	-	-	*	*	*	*
cmpsw	Eilučių žodžių palyginimas $comp [ds:si], [es:di]$ $si \leftarrow si + 2; di \leftarrow di + 2$	*	-	-	-	*	*	*	*
scas dest	Baito arba žodžio paieška eilutėje $comp acc, [es:di]$ $inc/dec di$ pagal df	*	-	-	-	*	*	*	*
scasb	Baito paieška eilutėje $comp al, [es:di]$ $di \leftarrow di + 1$	*	-	-	-	*	*	*	*
scasw	Žodžio paieška eilutėje $comp ax, [es:di]$ $di \leftarrow di + 2$	*	-	-	-	*	*	*	*
<i>Pakartojimo prefikso</i>									
rep / repe / repz	Kartojimas, kol $cx \neq 0$ ir $zf = 1$ Prefiksas leidžia pakartoti po jo esančios eilutinės komandos vykdymą pagal sąlygą, kaskart mažinant cx reikšmę vienetu. Nors visi šie trys prefiksai yra sinonimai (tas pats mašininis kodas), jų veikimas priklauso nuo po jų esančios komandos: <ul style="list-style-type: none"> - rep naudojamas prieš komandas movs(b/w), lods(b/w), stos(b/w) ir leidžia kartoti jų vykdymą, kol cx!=0; - repe(z) naudojamas prieš komandas cmps(b/w), scas(b/w) ir leidžia kartoti jų vykdymą, kol cx!=0 ir zf=1. 	-	-	-	-	-	-	-	-
repne / repnz	Kartojimas, kol $cx \neq 0$ ir $zf = 0$ Analogiškai repe(z) , tik sąlyga zf=0	-	-	-	-	-	-	-	-

¹⁾ Žymėjimai: „-“ – reikšmė nesikeičia; „*“ – reikšmė keičiasi; „u“ – reikšmė neapibrėžta.

2.4. Pavyzdžiai

Požymiai

	; al cf of zf sf pf
mov al,50h	; 50h - - - - - (t.y. požymiai nepakinta)
add al,50h	; A0h 0 1 0 1 1
add al,60h	; 00h 1 0 1 0 1

Palyginimai

mov al,200	
cmp al,100	
je @@Ne	; nepereis, nes 200 nelygu 100
jg @@Ne	; nepereis, nes (200=-56)<100
jb @@Ne	; nepereis, nes 200 nėra žemiau už 100
ja @@Taip	; pereis, nes 200 aukščiau už 100

Eilučių skaitymas

cld	; automatiškai didinsime si
@@Repeat:	
lodsb	; al←[ds:si]; si←si+1
or al,al	; al=0?
jne @@Repeat	

Paieška eilutėje

cld	; automatiškai didinsime di
mov di,OFFSET Strn	; es:di - eilutė
mov cx,250	; ieškosime pirmuose 250 baitų
xor al,al	; al←0
repne scasb	; ieškome al
je @@Found	; [es:di-1] yra 0

ASCIIZ eilutės išvedimas po simbolių

cld	; si (di) didinsime
mov ah,2h	; int 21,2 - simbolio išvedimas
@@Repeat:	
lodsb	; al←-[ds:si]
or al,al	
jz @@Exit	; iseiti, jei al=0
mov dl,al	
int 21h	
jmp SHORT @@Repeat	

3. Eilučių įvedimas, išvedimas, apdorojimas

- Pavyzdys *PutChars.asm* – teksto išvedimas po simbolių:

```
; Programa 'PutChars'
; Darbas su eilutėmis; išvedimas po simbolių; procedūros
; -----

LOCALS @@                                ; Lokalias žymes prasideda @@

.MODEL small                             ; Atminties modelis:
                                           ; 64K kodui ir 64K duomenims
.STACK 256                               ; Stekas

.DATA
msgText DB "Text to print",0

.CODE
Strt:
    mov ax,@data                         ; ds - duomenų segmentas
    mov ds,ax

    mov si,OFFSET msgText
    call PrintString

Exit:
    mov ax,04C00h
    int 21h                             ; int 21,4C - programos pabaiga

; -----
; PrintString - prints string to STDOUT
;     IN
;     ds:si - string to print
; -----
PrintString PROC
    push ax                             ; išsaugoti ax
    push dx                             ; išsaugoti dx

    cld                                ; si (di) didinsime
    mov ah,2h                          ; int 21,2 - simbolio išvedimas

@@Repeat:
    lodsb                               ; al<-[ds:si]
    or al,al
    jz @@Exit                           ; išeiti, jei al=0
    mov dl,al
    int 21h
    jmp SHORT @@Repeat

@@Exit:
    pop dx
    pop ax
    ret
PrintString ENDP
; -----

END Strt
```

❑ Pavyzdys *InOut.asm* – įvedimas ir išvedimas:

```

; Programa 'InOut': įvedimas ir išvedimas
; -----

.MODEL small                                ; Atminties modelis:
                                           ; 64K kodui ir 64K duomenims

.STACK 100h                                ; Stekas

; ----- Konstantos -----
StdOut = 1
CR = 0Dh
LF = 0Ah

; ----- Duomenys -----
.DATA
msgText      DB "Enter text:$"
msgEcho      DB CR,LF,"You entered:$"

inputBuf     DB 255
              DB 0
              DB 255 DUP(?)

; ----- Kodas -----
.CODE
Strt:
    mov ax,@data                            ; ds - duomenų segmentas
    mov ds,ax

    mov dx,OFFSET msgText
    call Write

    mov ah,0Ah
    mov dx,OFFSET inputBuf
    int 21h                                ; skaitymas iš klaviatūros

    mov dx,OFFSET msgEcho
    call Write

    mov ah,40h
    mov bx,(StdOut)
    mov cx,[inputBuf+1]
    xor ch,ch
    mov dx,OFFSET inputBuf+2
    int 21h                                ; int 21,40 - išvedimas į failą
                                           ; ar įrenginį

    mov ax,04C00h
    int 21h                                ; int 21,4C - programos pabaiga

; -----
Write PROC
    push ax

    mov ah,09h
    int 21h                                ; int 21,09 - išvedimas į ekraną

    pop ax
    ret
Write ENDP
; -----

END Strt

```

❑ Pavyzdys *Kaul.asm* – įvedimas, išvedimas, skaičiavimas:

```
; Programa 'Kauliukai'
; Programa generuoja skaičiu nuo 1 iki 6 ('meta kauliuka'), isveda,
; kiek tasku iskrito, visa surinktu tasku suma, ir pasiulo mesti dar
; -----

LOCALS @@

.MODEL small                      ; 64K kodui ir 64K duomenims

.STACK 100h

.DATA
hex          DB "0123456789ABCDEF"

vienas       DB ". (vienas)",0
du           DB ".. (du)",0
trys        DB "... (trys)",0
keturi       DB "::. (keturi)",0
penki        DB "::: (penki)",0
sesi         DB ":::: (sesi)",0
kauliukai    DW vienas,du,trys,keturi,penki,sesi

msgKaul      DB "Kauliukas ",0
msgViso      DB ", viso "
msgVisoSk    DB "0000h",0
msgNewLn     DB 0Dh,0Ah,0
msgArDar     DB "Metam dar? (T/N): ",0

inputBuf     DB 255,0,255 DUP(?)

.CODE
pradzia:
    mov ax,@data
    mov ds,ax
    mov es,ax

    xor cx,cx                      ; Kiek tasku surinkom is viso

@@mesti:
    call Mesk                      ; ax = 1..6
    add cx,ax

    mov dx,OFFSET msgKaul
    call Write

    dec ax
    shl ax,1
    mov bx,ax
    mov dx,[bx + kauliukai]
    call Write

    mov dx,OFFSET msgVisoSk
    mov ax,cx
    call IntToHex

    mov dx,OFFSET msgViso
    call Write

@@arDar:
    mov dx,OFFSET msgNewLn
    call Write
```

```
    mov dx,OFFSET msgArDar
    call Write

    mov dx,OFFSET inputBuf
    call Read

    cmp [inputBuf + 1],0
    je @@arDar
    mov al,[inputBuf + 2]
    cmp al,'T'
    je @@mesti
    cmp al,'t'
    je @@mesti

    mov ax,4C00h
    int 21h

;-----
; Write - Isveda eilute i ekrana
; IN
;   dx - eilute
;-----
Write PROC
    push ax
    push bx
    push cx
    push di

    cld
    mov di,dx
    mov cx,0FFFFh
    xor al,al
    repne scasb
    neg cx
    sub cx,2

    mov ah,40h
    mov bx,1
    int 21h

    pop di
    pop cx
    pop bx
    pop ax
    ret
Write ENDP

;-----
; Read - Skaito eilute is klaviatūros
; IN
;   dx - buferis eilutei
;-----
Read PROC
    push ax

    mov ah,0Ah
    int 21h

    pop ax
    ret
Read ENDP
```

```
-----  
; IntToHex - Konvertuoja skaiciu i sesioliktaine eilute  
; IN  
;   ax - skaicius  
;   dx - buferis eilutei  
; OUT  
;   dx - ASCIIZ eilute  
-----  
IntToHex PROC  
    push ax  
    push bx  
    push cx  
    push di  
  
    mov di,dx  
    mov cx,4  
  
@@kartoti:  
    rol ax,4  
    mov bx,ax  
    and bx,0Fh  
    mov bl,[bx + hex]  
    mov [di],bl  
    inc di  
    loop @@kartoti  
  
    pop di  
    pop cx  
    pop bx  
    pop ax  
    ret  
IntToHex ENDP  
  
-----  
; Mesk - "Meta" kauliuka: generuoja atsitiktini skaiciu nuo 1 iki 6  
; OUT  
;   ax - skaicius nuo 1 iki 6  
-----  
Mesk PROC  
    push cx  
    push dx  
  
    mov ah,2Ch  
    int 21h      ; CH = valandos (0-23), CL = minutes (0-59)  
                ; DH = sekundes (0-59), DL = simtosios (0-99)  
  
    mov al,dl  
    xor ah,ah  
    mov cl,6h  
    div cl  
    mov al,ah  
    inc al  
    xor ah,ah  
  
    pop dx  
    pop cx  
    ret  
Mesk ENDP  
-----  
  
END pradzia
```


4. Failai

- Tikslas: susipažinti su DOS darbo su failais priemonėmis
- Pavyzdys *SlfPrint.asm* – failo skaitymas ir išvedimas į ekraną:

```

; Programa 'SlfPrint'
; Darbas su failais
; -----

LOCALS @@                                ; Lokalios zymės prasideda @@

.MODEL small                             ; Atminties modelis:
                                           ; 64K kodui ir 64K duomenims

.STACK 256

.DATA
fname      DB "SlfPrint.asm",0           ; Failo, kuri skaitysime, vardas
handle     DW 0

.DATA?
fbuf       DB 100h DUP(?)                ; Neinicializuojami duomenys
                                           ; Skaitymo buferis

.CODE
Strt:
    mov ax,@data
    mov ds,ax

    mov dx,OFFSET fname                   ; Atidaryti faila
    mov ax,3d00h
    int 21h
    jc Exit

    mov [handle],ax                       ; Issaugoti deskriptoriu
    mov bx,ax

@@Loop:
    mov ah,3fh
    mov cx,100h
    mov dx,OFFSET fbuf
    int 21h                               ; Skaityti faila
    jc Exit

    or ax,ax
    jz Exit                               ; EOF - failo pabaiga
    mov cx,ax
    call PrintBuf
    jmp SHORT @@Loop

Exit:
    mov bx,[Handle]
    or bx,bx
    jz @@NoClose

    mov ah,3Eh
    int 21h                               ; Uzdaryti faila

@@NoClose:
    mov ax,04C00h
    int 21h                               ; int 21,4C - programos pabaiga

; -----
; PrintBuf - prints char buffer to STDOUT
; IN

```

```

;      cx - char count
;      dx - buf
;-----
PrintBuf PROC
    push ax                      ; Issaugome steke registrus,
    push bx                     ; kurie keisis

    mov ah,40h
    mov bx,1
    int 21h                     ; int 21,40 - isvedimas i faila
                                ; ar irengini

    pop bx                      ; Atstatome issaugotus registrus
    pop ax
    ret
PrintBuf ENDP
END Strt

```

Darbai su failais DOS yra du metodai: naudojant FCB (angl. *file control block* – failų valdymo blokas) ir naudojant deskriptorius. Pastarasis būdas, naudojamas nuo DOS 2.0, yra paprastesnis (nenaudoja sudėtingų duomenų struktūrų), patogesnis (failai visose operacijose identifikuojami deskriptoriumi, gautu atidarius failą) ir rekomenduojamas (FCB naudoti nebesiūloma). Dėl šių priežasčių toliau bus nagrinėjamos tik deskriptorinės failų operacijos.

Daugeliui aprašytų DOS funkcijų reikia nurodyti failų vardų ASCIIZ formate adresus. Tai – simbolių eilutės, kurių pabaiga žymima 0. Galima nurodyti ir įrenginio vardą bei kelią. Pvz.:

Vardas	DB 'TESTAS.ASM',0
PilnasVardas	DB 'A:\ASM\TESTAS.ASM',0

Žemiau pateikiamos pagrindinės DOS funkcijos darbui su failais. Be šių, yra ir kitos: failo paieška, pervadinimas, failo laiko ir datos nustatymas ir pan. Jų aprašymus žr. žinynuose („HelpPC Reference Library“ adresu <http://heim.ifi.uio.no/~stanisls/helppc/> ar kt.)

4.1. Failo atidarymas (funkcija 3Dh)

Prieš skaitant ar rašant failą, jį būtina atidaryti. Operacinė sistema atidarydama failą patikrina, ar failas nurodytu vardu egzistuoja, suteikia jam identifikacinį numerį (deskriptorių) ir pasiruošia jo skaitymui ir/ar rašymui. Jei įvyko klaida, grąžinamas klaidos kodas.

int 21,3D – failo atidarymas naudojant deskriptorių

IN	<p>AH – 3Dh (funkcijos numeris)</p> <p>AL – darbo su failu režimas. Galimos reikšmės:</p> <p>00 – tik skaitymui</p> <p>01 – tik rašymui</p> <p>02 – skaitymui ir rašymui</p> <p>DS:DX – ASCIIZ (eilutė, kurios pabaiga žymima 0) failo vardo adresas. Galima nurodyti ir kelią (disko vardą ir/ar katalogus)</p>
OUT	<p>AX – failo deskriptorius, jei CF požymis nenustatytas (CF=0)</p> <p>– klaidos kodas, jei CF požymis nustatytas (CF=1)</p>

Galimos klaidos:

Kodas	Aprašymas
01h	Neteisingas funkcijos numeris
02h	Failas nerastas
03h	Kelias nerastas
04h	Perdaug atidarytų failų (nebėra laisvų deskriptorių)

05h	Failas neprieinamas
0Ch	Neteisingas darbo režimas registre AL

4.2. Failo skaitymas (funkcija 3Fh)

Skaitant failą reikia nurodyti deskriptorių, gautą jį atidarius, kiek baitų ir kur juos skaityti. Perskaitytų baitų skaičius, grąžinamas funkcijos, gali būti mažesnis, nei buvo nurodyta skaityti (ax<cx). Tai reiškia, kad buvo pasiekta failo pabaiga.

int 21,3F – failo skaitymas naudojant deskriptorių	
IN	AH – 3Fh (funkcijos numeris) BX – failo deskriptorius, gautas atidarius failą CX – kiek baitų skaityti DS:DX – skaitymo buferio adresas
OUT	AX – kiek baitų perskaityta, jei CF požymis nenustatytas (CF=0) – klaidos kodas, jei CF požymis nustatytas (CF=1)

4.3. Failo rašymas (funkcija 40h)

Rašant failą, panašiai, kaip ir skaitant, reikia nurodyti deskriptorių, rašomų baitų skaičių, bei buferį, iš kurio bus rašoma. Jei faktiškai įrašytų baitų skaičius mažesnis, nei buvo nurodyta (ax<cx), tai greičiausiai įvyko disko perpildymas.

int 21,40 – failo rašymas naudojant deskriptorių	
IN	AH – 40h (funkcijos numeris) BX – failo deskriptorius, gautas atidarius failą CX – kiek baitų rašyti DS:DX – rašymo buferio adresas
OUT	AX – kiek baitų įrašyta, jei CF požymis nenustatytas (CF=0) – klaidos kodas, jei CF požymis nustatytas (CF=1)

4.4. Pozicijos faile keitimas (funkcija 42h)

Failo skaitymo ir rašymo funkcijos po sėkmingai atliktos operacijos perstumia einamosios pozicijos žymę faile per perskaitytų ar įrašytų baitų skaičių. Tai reiškia, kad, pvz., pakartotinai kviečiant skaitymo funkciją failas bus skaitomas ne nuo pradžių, bet toliau, nuo ten, kur buvo baigta pirmuoju skaitymu. Kartais reikia skaityti (ar rašyti) ne viską iš eilės, bet tik pasirinktas vietas – reikalingas būdas pakeisti einamąją poziciją.

int 21,42 – einamosios pozicijos faile nustatymas naudojant deskriptorių	
IN	AH – 42h (funkcijos numeris) AL – nuo kur skaičiuojamas poslinkis 00 – nuo failo pradžios 01 – nuo einamosios pozicijos 02 – nuo failo pabaigos BX – failo deskriptorius, gautas atidarius failą CX:DX – per kiek baitų pastumti (skaičius su ženklu)
OUT	DX:AX – nauja pozicija nuo failo pradžios, jei CF nenustatytas (CF=0) AX – klaidos kodas, jei CF požymis nustatytas (CF=1)

4.5. Failo uždarymas (funkcija 3Eh)

Atlikus reikiamus veiksmus su failu, jį reikia uždaryti. Uždarant failą sistema į jį įrašo visus tarpiniuose apsikeitimo buferiuose esančius duomenis, papildo įrašą kataloge (nurodo failo dydį, sukūrimo datą ir laiką) ir atlaisvina deskriptorių.

int 21,3E – failo uždarymas naudojant deskriptorių	
IN	AH – 3Eh (funkcijos numeris) BX – uždaromo failo deskriptorius
OUT	AX – klaidos kodas, jei CF požymis nustatytas (CF=1)

4.6. Failo sukūrimas (funkcija 3Ch)

Failui sukurti reikia nurodyti jo atributus ir vardą. Jei nurodytas failas egzistuoja, jo dydis tampa lygus 0.

int 21,3C – failo sukūrimas naudojant deskriptorių	
IN	AH – 3Ch (funkcijos numeris) CX – failo atributai. Tai – bitų laukas, kurio bitai reiškia: bitas 0 (cx = 01) – failas tik skaitymui („read-only“) bitas 1 (cx = 02) – paslėptas („hidden“) failas bitas 2 (cx = 04) – sisteminis („system“) failas DS:DX – ASCIIIZ (eilutė, kurios pabaiga žymima 0) failo vardo adresas
OUT	AX – failo deskriptorius, jei CF požymis nenustatytas (CF=0) – klaidos kodas, jei CF požymis nustatytas (CF=1)

4.7. Failo sunaikinimas (funkcija 41h)

Failui sunaikinti užtenka nurodyti jo vardą:

int 21,41 – failo sunaikinimas	
IN	AH – 41h (funkcijos numeris) DS:DX – ASCIIIZ (eilutė, kurios pabaiga žymima 0) failo vardo adresas
OUT	AX – klaidos kodas, jei CF požymis nustatytas (CF=1)

5. Programos segmento prefiksas. Komandinė eilutė

□ Pavyzdys *ParamStr.asm* – komandinės eilutės parametrų išvedimas į ekraną

```
; Programa 'ParamStr'
; Programa atspausdina komandinės eilutės parametrus
; -----

LOCALS @@                                ; Lokalios žymės prasideda @@

.MODEL small                            ; Atminties modelis:
                                        ; 64K kodui ir 64K duomenims

.STACK 256

.DATA
msg          DB "Programos parametrai:"
msgLen       = $-msg                    ; Eilutės msg ilgis gaunamas
                                        ; is einamojo adreso ($) atemus
                                        ; jos pradžios adresą

.CODE
Strt:
    push ds                             ; Issaugome steke PSP pradžios
                                        ; adresą

    mov ax,@data
    mov ds,ax
    mov cx,(msgLen)
    mov dx,OFFSET msg
    call PrintBuf                        ; Atspausdiname pranešimą
    pop ds

    mov cl,[ds:80h]                     ; Parametrų eilutės ilgis
                                        ; (PSP su poslinkiu 80h)

    xor ch,ch
    mov dx,81h                          ; Parametrų eilutės pradžios
                                        ; adresas

    call PrintBuf

    mov ax,04C00h
    int 21h                             ; int 21,4C - programos pabaiga

; -----
; PrintBuf - prints char buffer to STDOUT
;     IN
;     cx - char count
;     ds:dx - buf
; -----

PrintBuf PROC
    push ax                             ; Issaugome steke registrus,
    push bx                             ; kurie keisis

    mov ah,40h
    mov bx,1
    int 21h                             ; int 21,40 - išvedimas į failą
                                        ; ar įrenginį

    pop bx                              ; Atstatome išsaugotus registrus
    pop ax
    ret
PrintBuf ENDP
; -----

END Strt
```

Prieš įkeldama programą į atmintį vykdymui, operacinė sistema sukuria specialų duomenų bloką – *programos segmento prefiksą* (PSP). Pati programa išdėstoma iškart po jo. PSP saugo daug informacijos; dalis jos yra naudinga, dalis – pasenusi ir nebenaudojama. Jo dydis yra 256 baitai, o pagrindiniai laukai šie:

Offset	Length	Description
0	2	An INT 20h instruction is stored here
2	2	Program ending address
4	1	Unused, reserved by DOS
5	5	Call to DOS function dispatcher
0Ah	4	Address of program termination code
0Eh	4	Address of break handler routine
12h	4	Address of critical error handler routine
16h	22	Reserved for use by DOS
2Ch	2	Segment address of environment area
2Eh	34	Reserved by DOS
50h	3	INT 21h, RETF instructions
53h	9	Reserved by DOS
5Ch	16	Default FCB #1
6Ch	20	Default FCB #2
80h	1	Length of command line string
81h	127	Command line string

Svarbūs laukai yra šie:

- :00h) int 20h instrukcija (kodas CD20h). Kai kurios programos valdymą operacinei sistemai grąžina ne int 21,4Ch funkcija, o retf komanda. Tokiu atveju programos pradžioje į steką turi būti padedamas PSP pradžios adresas. Įvykdyta retf komanda perduoda valdymą į PSP pradžią, tada įvykdoma komanda int 20h, kuri yra alternatyvus (bet nerekomenduotinas) valdymo grąžinimo DOS būdas
- :2Ch) aplinkos kintamųjų sritis (angl. *environment*). Joje saugomi PATH, SET ir kt. komandų nustatyti aplinkos kintamieji
- :80h) komandinės eilutės parametrų ilgis
- :81h) komandinės eilutės parametrai, pabaigiami CR kodu. Pvz., jei programa paleista komandine eilute

MYPGM parameter1, parameter2

poslinkiu 80h bus

23, " parameter1, parameter2", 0Dh

Sritis 80..FFh taip pat naudojama kai kurių darbo su failais funkcijų.

Programos vykdymo pradžioje PSP segmentinį adresą nurodo ds registras. Būtent todėl jį tenka inicializuoti duomenų segmento pradžios adresu @data. Jei programai reikės naudoti PSP, jo adresą galima išsaugoti taip:

```

push    ds          ; Padėti PSP segmentinį adresą į steką
mov     ax, @data    ; Persiųsti į ds ir es duomenų segmento
mov     ds, ax       ; pradžios adresą
mov     es, ax
pop     PSP_Adresas ; Išsaugoti PSP adresą kintamajame "PSP_Adresas"
```

6. Pertraukimai

□ Pavyzdys *ClockInt.asm* – taimerio pertraukimų apdorojimas

```

; Programa 'ClockInt'
; Programa demonstruoja taimerio pertraukimu apdorojima
; -----

LOCALS @@                                ; Lokalios zymes prasideda @@

.MODEL small                             ; 64K kodui ir 64K duomenims
.STACK 100h

IntNo = 8h

; ----- Duomenys (kintamieji) -----
.DATA
Counter      DB 4
ClockCnt      DB 0
Msg           DB "0...$"

; ----- Kodas -----
.CODE
OldISRSeg     DW 0
OldISROfs     DW 0

Strt:
    mov ax,@data
    mov ds,ax

    ; --- es <- 0
    mov ax, 0
    mov es, ax

    ; --- Issaugome senos pertraukimo apdorojimo proceduros adresa
    mov ax,es:[IntNo*4]
    mov cs:[OldISROfs],ax
    mov ax,es:[IntNo*4 + 2]
    mov cs:[OldISRSeg],ax

    ; --- "Instaliuojame" nauja pertraukimu apdorojimo procedura
    pushf
    cli
    mov word ptr es:[IntNo*4],offset TimerProc
    mov word ptr es:[IntNo*4 + 2],seg TimerProc
    popf

@@loop:
    cmp [Counter],0
    jne @@loop

    ; --- Atstatome sena pertraukimu apdorojimo procedura
    pushf
    cli
    mov ax,cs:[OldISROfs]
    mov word ptr es:[IntNo*4],ax
    mov ax,cs:[OldISRSeg]
    mov word ptr es:[IntNo*4 + 2],ax
    popf

    ; --- Baigiame darba
    mov ax,04C00h
    int 21h                                ; int 21,4C - programos pabaiga

```

```

;-----
TimerProc PROC
    push ax
    push ds

    mov ax,@data
    mov ds,ax

    mov al,[ClockCnt]
    inc al
    cmp al,20
    jne @@skip
    xor al,al
    dec [Counter]
    call PrintCnt

@@skip:
    mov [ClockCnt],al
    pop ds
    pop ax

    push cs:[OldISRSeg]
    push cs:[OldISROfs]
    retf
TimerProc ENDP

;-----
PrintCnt PROC
    push ax
    push dx

    mov al,[Counter]
    add al,30h

    mov [Msg],al
    mov dx,OFFSET Msg
    mov ah,9h
    int 21h

    pop dx
    pop ax
    ret
PrintCnt ENDP

END Strt

```

Taimerio (int 8h) pertraukimai pagal nutylėjimą kyla ~18,2 kartų per sekundę (t.y. kas ~55ms). Programa įrašo savo pertraukimo apdorojimo procedūros TimerProc adresą į pertraukimų vektorių lentelę ir vykdo ciklą kol Counter reikšmė nebus 0 (programos pradžioje ji yra 4). Tuo metu, vykstant taimerio pertraukimams, procedūra skaičiuoja iki 20 (t.y. kol praeis beveik sekundė), išveda pranešimą ir sumažina Counter reikšmę vienetu.

Kai Counter pasiekia 0, programa pertraukimų vektorių lentelėje atstato buvusį pertraukimų apdorojimo procedūros adresą ir baigia darbą.

❑ Pavyzdys *KbdShift.asm* – klaviatūros pertraukimų apdorojimas

```

; Programa 'KbdShift'.
; Programa demonstruoja klaviatūros pertraukimu apdorojimą.
; Paleista ji pakeičia int 15h vektoriu ir baigia darba
; pasilikdama atmintyje (zr. TSR, int 27h). Nuo tada,
; paspaustu klavisu kodai bus didinami 1, t.y. paspaude
; 'a' matysime 's' ir t.t.
; -----

.MODEL small                      ; 64K kodui ir 64K duomenims
.STACK 100h

; ----- Kodas -----
.CODE

OldHandlerSeg    DW 0
OldHandlerOffs   DW 0

; ----- NewHandler -----
NewHandler PROC
    cmp ah,4fh          ; Klaviatūros pertraukimo (int 9) apdorojimo
    jne @@skip          ; procedūra generuoja int 15, ah=4fh
    inc al              ; al = klaviso kodas

@@skip:
    push [cs:OldHandlerSeg]
    push [cs:OldHandlerOffs]
    retf               ; Iskviesti sena pertraukimu apdorojimo proc.
NewHandler ENDP
; -----

Strt:
    mov ax,3515h        ; Issaugoti senos pertraukimu apdorojimo
    int 21h            ; procedūros adresa (int 15h)
    mov [cs:OldHandlerSeg],es
    mov [cs:OldHandlerOffs],bx

    push cs
    pop ds
    mov dx,OFFSET NewHandler
    mov ax,2515h        ; Nustatyti nauja pertraukimo vektoriu
    int 21h            ; (int 15h apdorojimo procedūros adresa)

    mov dx,OFFSET Strt + 100h
                        ; Baigti darba ir pasilikti atmintyje
    int 27h            ; (TSR - terminate and stay resident)
                        ; Atmintis issaugoma iki adreso cs:dx
; -----
END Strt

```

Pertraukimas – tai laikinas veikiančios programos sustabdymas reaguojant į tam tikrą įvykį, įvykdamas specialią pertraukimo apdorojimo paprogramę, ir pratęsiant pagrindinės programos vykdymą. Pertraukimai pagal šaltinį skirstomi į:

- išorinius (generuoja išorinis procesoriaus atžvilgiu įrenginys);
- vidinius (programinis pertraukimas komanda int arba speciali situacija: dalyba iš 0 ir pan.).

8086 procesorius gali apdoroti iki 256 pertraukimų. Kiekvieną jų atitinka 4 baitų elementas masyve, vadinamame pertraukimų vektorių lentele. Ši lentelė yra atminties

pradžioje adresu 0:0; jos dydis $256 \times 4 = 1024 = 1K$. Lentelės elementai – tai tolimi (*far*) pertraukimo apdorojimo procedūrų adresai.

Įvykus pertraukimui, nesvarbu, koks jo šaltinis, 8086 atlieka tokius veiksmus:

- 1) išsaugo steke požymių registrą;
- 2) nuvalo *if* ir *tf* požymius;
- 3) išsaugo steke grįžimo adreso segmentą (*cs*), po to – poslinkį (*ip*);
- 4) nustato pertraukimo numerį ir paima 4 baitų adresą iš pertraukimų vektorių lentelės adresu $0:n \times 4$;
- 5) perduoda valdymą gautu adresu, kopijuodamas jį į *cs:ip*.

Pertraukimų apdorojimo procedūra baigia darbą komanda *iret*, kuri atstato iš steko registrus *ip*, *cs* ir požymių registrą.

Nauja pertraukimų apdorojimo procedūra gali būti „instaliuojama“ dviem būdais:

- 1) tiesiogiai modifikuojant pertraukimų vektorių lentelę

```

mov     ax, 0
mov     es, ax
pushf
cli
mov     word ptr es:[n*4], offset ISRProcedure
mov     word ptr es:[n*4 + 2], seg ISRProcedure
popf

```

- 2) naudojant DOS funkciją *int 21,25h*

```

push    ds                ; Išsaugoti ds
mov     ax, 25ffh         ; AH=25h, AL=0FFh
mov     dx, seg ISRProcedure ; Pakrauti į DS:DX
mov     ds, dx            ; procedūros adresa
mov     dx, offset ISRProcedure
int     21h               ; Iškviesti DOS funkciją
pop     ds                ; Atstatyti DS

```

Prieš grąžindama valdymą operacinei sistemai, programa turi atstatyti modifikuotus vektorius. Priešingu atveju, kai atminties sritis, kurioje kadaise buvo instaliuoti pertraukimų apdorojimo procedūra, bus paskirta kitai programai, įvykęs pertraukimas sukels katastrofą. Išimtis yra taip vadinamos rezidentinės programos, kurios pabaigia darbą grąžindamos valdymą operacinei sistemai specialia funkcija, neatlaisvinančia visos užimamos atminties (*int 21,31h* ar *int 27h*).

Kai kurie svarbesni pertraukimai:

INT #	Panaudojimas	
0	CPU	Dalybos iš nulio (divide by zero)
1	CPU	Žingsninio režimo (single step)
2	CPU	NMI (non-maskable)
3	CPU	Sustojimo taško (breakpoint)
4	CPU	Perpildymo (overflow trap)
...		
8	IRQ0	Taimerio (55ms intervalu, 18.21590 kartų per sekundę)
9	IRQ1	Klaviatūros (keyboard service required)
A-F	IRQ2-IRQ7	Kiti aparatūriniai
10	BIOS	Video BIOS
11-1F	BIOS	Kiti BIOS
20	DOS	Programos pabaiga (general program termination)
21	DOS	DOS funkcijos (function request services)
22..2F	DOS	Kiti DOS
...		
70-77	IRQ8-IRQ15	Kiti aparatūriniai
78-FF		Kiti