

Datoru organizācija un asambleri

Profesors Uldis Sukovskis

1

Mērķi

- Apgūt asamblera valodas pamatus, salīdzinot to ar citām, augstāka līmeņa programmēšanas valodām. Analizēt tās priekšrocības un trūkumus.
- Iegūt programmēšanas prasmes asamblera valodā un lietot tās specifisku programmatūras līdzekļu izstrādei.
- Iepazīties ar datora galveno komponentu organizāciju, savstarpējo saistību un programmēšanas iespējām, lietojot asamblera valodu.
- Nepieciešamās priekšzināšanas:
 - Skaitļšanas sistēmas, Būla algebra, programmēšanas pamati

Priekšmeta apguves rezultātā students

- izprot asambleru vietu un lomu citu programmēšanas valodu vidū
 - Sekmīgi nokārtoti kontroldarbi un eksāmens
- spēj patstāvīgi realizēt vienkāršas programmas asamblera valodā un izskaidrot to darbību.
 - Sekmīgi izpildīti un aizstāvēti laboratorijas darbi.
- spēj analizēt asamblera programmu uzbūvi un darbību.
 - Sekmīgi nokārtoti kontroldarbi un eksāmens
- izprot un spēj izskaidrot pārtraukumu apstrādi un datora komponentu (tastatūras, videoadaptera, taimera u.c.) darbības principus.
 - Sekmīgi nokārtoti kontroldarbi un eksāmens

Tēmas

1. Datoru procesori.
2. Vienkāršas programmas piemērs.
3. Ievads asamblera valodā.
4. Operētājsistēmas funkciju lietošana.
5. Programmu veidošana un izpilde.
6. Ciklu programmēšana.
7. Apakšprogrammu lietošana.
8. Virkņu apstrādes komandas.
9. Komandrindas parametru apstrādes paņēmieni.
10. Pārtraukumu apstrādes principi.
11. Pārtraukumu apstrādes programmu veidošana asamblerā.
12. Darbs ar tastatūru.
13. Darbs ar videoadapteri.
14. Taimera programmēšana.
15. Disku atmiņas organizācija.

Literatūra

- Skat. ORTUS e-studiju sistēmā mācību materiālus.
- Randall Hyde. The Art of Assembly Language. Brīvi pieejams internetā <http://webster.cs.ucr.edu/AoA/>.
- Ieteicamā literatūra
 - Kip R. Irvine. Assembly Language for Intel-Based Computers (5th Edition). – Prentice Hall, 2006. ISBN-13: 978-0132383103, 752 lpp.
 - Peter Abel. IBM PC Assembly Language and Programming (5th Edition). – Prentice Hall, 2001. ISBN-13: 978-0130306555, 540 lpp.
 - Peter Norton. Peter Norton's Computing Fundamentals (6 edition). – Career Education, 2004. ISBN-13: 978-0072978476, 608 lpp.

Laboratorijas darbi

- Studenti izpilda laboratorijas darbus datorklasē.
- Katram studentam ir individuāls uzdevuma variants.
- Visi laboratorijas darbi ir praktiski programmēšanas uzdevumi asamblera valodā.
- Visi laboratorijas darbi jāizpilda precīzi noteiktajos termiņos.
- Sešus laboratorijas darbus var aizstāt ar vienu speciālu paaugstinātas sarežģītības individuālo uzdevumu. Tie ir ievērojami sarežģītāki par laboratorijas darbiem un ieteicami studentiem ar labām priekšzināšanām un prasmi asamblera programmu veidošanā.

Kontroldarbi

- Semestra gaitā lekciju sākumā paredzēti vairāki 15-20 minūšu kontroldarbi.
- Par katru kontroldarbu tiks paziņots iepriekš e-studiju vidē. Kontroldarbu izpildīšana citā laikā nav paredzēta, izņemot gadījumus, ja studentam ir objektīvs un dokumentāli pamatots attaisnojošs iemesls. Kontroldarbu pārrakstīšana vērtējuma uzlabošanai nav paredzēta.
- Visu kontroldarbu vidējais vērtējums tiek iekļauts priekšmeta gala vērtējumā.

Eksāmens

- Sesijā studenti kārtro rakstisku eksāmenu, kas tiek vērtēts ar atzīmi.
- Studenti tiek pielaisti pie eksāmena neatkarīgi no laboratorijas darbu (vai speciālā individuālā uzdevuma) vērtējuma.
- Atbilde uz katru jautājumu tiek vērtēta ar atsevišķu atzīmi (no 0 par neatbildētu jautājumu līdz 10 par izcilu atbildi) un eksāmena atzīme tiek aprēķināta kā aritmētiskais vidējais, ievērojot arī jautājumu grūtības pakāpi raksturojošus svāra koeficientus.

Gala vērtējums

Gala vērtējumu priekšmetā (A) aprēķina no eksāmena vērtējuma (E), kontroldarbu vērtējuma (K) un vērtējuma par laboratorijas darbiem (LD) pēc šādas formulas:

$$A = 0,5 * E + 0,15 * K + 0,35 * LD, \text{ ja } E > 3.$$

Ja eksāmena atzīme E ir nesekmīga, tad gala atzīme A=E

Intel x86 vēsture

- Intel dibināta 1968.g. Palo Alto (ASV).
- 1974.g. **8080** - 8 bitu procesors PC ražošanai
- 1978.g. **8086** - pirmais 16 bitu procesors.
- 1979.g. **8088** - 8 bitu ārējā datu kopne, 4.77MHz.
- 1981.g. **IBM pirmais personālais dators.**
- 1983.g. **80286** - ar virtuālo adresāciju 16MB, 8 - 10 MHz.
Koprocesors 80287 operācijām ar peldošo punktu.
- 1985.g. **80386** - pirmais 32 bitu mikroprocesors.
Koprocesors 80387 operācijām ar peldošo punktu.
- 1989.g. **80486** - procesors + koprocesors + kešatmiņa (8K)
- 1994.g. **Pentium**
- 2001.g. **64-bit procesori** (Merced, Itanium, SPARC, Alpha...)
- ...
- 2003.g. **AMD Athlon 64**
- ...

Vienkāršas .com programmas piemērs

```
; Illustrates full segment directives for COM program

TEXT                SEGMENT      ; Code segment
                    ASSUME cs:TEXT, ds:TEXT
                    ORG    100h

start:              jmp     go
msg:                DB        "Sveiks!", 7, 13, 10, "$"
go:                 mov     ah, 9h      ; Function 9
                    mov     dx, OFFSET msg ; Load DX
                    int     21h        ; Display String
                    int     20h        ; Exit

TEXT                ENDS
                    END    start
```

Vienkāršas .com programmas piemērs

```
; Illustrates simplified segment directives for COM program

.MODEL tiny
.DATA
msg DB "Sveiks!", 7, 13, 10, "$"
.CODE
.STARTUP
mov ah, 9h      ; Function 9
mov dx, OFFSET msg ; Load DX
int 21h        ; Display String

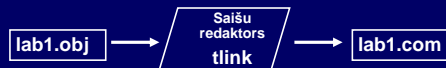
.EXIT 0
END
```

Programmas sagatavošana

C:\temp> tasm lab1



C:\temp> tlink /t lab1



tasm <source>, <object>, <list> /h

tlink <object>, <executable>, <map>, <libs> /t /h

Procesora reģistri

- Vispārējās nozīmes reģistri (general purpose registers),
- Indeksu reģistri (index registers),
- Steka reģistri (stack registers),
- Segmentu reģistri (segment registers),
- Karogu reģistrs (flag register).

Ir 8, 16 un 32 bitu reģistri, jaunākā bita numurs ir 0.



Vispārējās nozīmes reģistri

| | | | |
|----|----|----|---|
| | 15 | 7 | 0 |
| AX | AH | AL | |
| BX | BH | BL | |
| CX | CH | CL | |
| DX | DH | DL | |

- Reģistrs AX

0001010011000011

- AX = ? (heksadecimāls)
- AH = ? (heksadecimāls)
- AL = ? (heksadecimāls)

AX = ? (decimāls)

AH = ? (decimāls)

AL = ? (decimāls)

```

mov ax, 14C3h
mov bx, 15
add ax, bx
  
```

150

SI

150

DI

150

150

150

150

Indeksu reģistri

Steka reģistri

mov si, 0

c1: add mas[si], 30h

inc si

loop c1

mov bp, sp

mov ax, [bp+4]

RTU DITF LDK U.Sukovskis

Datoru organizācija un asambleji

16

High Addresses

Stack Grows

Low Addresses

Data pushed

Free space

Stack pointer (SP)

Stack Segment (SS)

push ax

push cs

pop ds

RTU DITF LDK U.Sukovskis

Datoru organizācija un asambleji

17

1514131211109876543210

O

D

I

S

Z

P

Overflow

Interrupt Direction

Signe

Zero

Parity

sub ax, bx

jz nul ;jump if zero

...

nul: cmp al, bl

ja virs ;jump if above

cli ;interrupt flag = 0

sti ;interrupt flag = 1

RTU DITF LDK U.Sukovskis

Datoru organizācija un asambleji

18

Segmentu reģistri

150

CS

DS

SS

ES

mov ds, cs ;KĻŪDA!!!

mov es, 0 ;KĻŪDA!!!

mov bx, es:0419h

mov ax, cs

mov ds, ax

push cs

pop ds

xor ax, ax

mov es, ax

mov bx, es:0419h

RTU DITF LDK U.Sukovskis

Datoru organizācija un asambleji

19

Atmiņa

- RAM
- ROM
- Virtuālā atmiņa
- Atmiņas mērvienības
 - Bits, Baits, Vārds, Dubultvārds, Paragrāfs
 - KB, MB, GB, ...

0123456789A B C D E F101112131415161718191A1B1C1D1E1F...

vārds vārds ...

dubultvārds dubultvārds ...

paragrāfsparagrāfs

RTU DITF LDK U.Sukovskis

Datoru organizācija un asambleji

20

Aritmētisku aprēķinu piemēri

☛ $W = X + Y * Z$

x dw 8

y dw 5

z dw 2

w dw 0

;

...

; vispirms jā sareizina, pēc tam jā saskaita

mov ax, y

imul z ; dx:ax = ax * operand16 or ax = al * operand8

add ax, x

mov w, ax

RTU DITF LDK U.Sukovskis

Datoru organizācija un asambleji

21

7

Aritmētisku aprēķinu piemēri

● $W = X / Y - Z$

```
x    dw    18
y    dw    5
z    dw    2
w    dw    0
;
    mov     ax, x    ;
    cwd     ; convert word to double word – pārveido vārdu par
              dubultvārdu un ievieto rezultātu reģistru pārī dx, ax.
    idiv    y    ; dalījums tiek ievietots ax un atlikums dx
    sub     ax, z
    mov     w, ax
```

Ja dalītājs ir 8 bitu, tad dalāmajam jābūt 16 bitu vērtībai.

Ja dalītājs ir 16 bitu, tad dalāmais jāievieto reģistru pārī dx, ax.

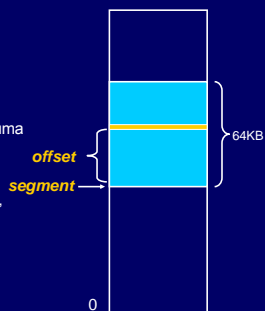
Aritmētisku aprēķinu piemēri

● $W = (A + B) * (Y + Z)$

```
a    dw    5
b    dw    3
y    dw    12
z    dw    -2
w    dw    0
temp1 dw    ?
temp2 dw    ?
;
    mov     ax, a
    add     ax, b
    mov     temp1, ax
    mov     ax, y
    add     ax, z
    mov     temp2, ax
    mov     ax, temp1
    imul    temp2
    mov     w, ax
```

Segmenti

- Segments ir līdz 64KB liels atmiņas apgabals
- Reālās adresācijas režīmā atmiņas adrese sastāv no segmenta adreses (**segment**) un attāluma baitos no segmenta sākuma (**offset**)
segment : offset
- **segment** ir paragrāfa numurs atmiņā, no kura sākas segments
- **segment** vērtība atrodas vienā no segmentu reģistriem



Atmiņas adreses

Fiziskā adrese = *segment* * 16 + *offset*

Piemērs:

```
mov ax, 40h
mov es, ax
mov bl, es: [18h]
```

Kāda ir fiziskā adrese baitam, kuru ieraksta reģistrā bl ?
Fiziskā adrese = 40h * 16 + 18h = 400h + 18h = 418h

segment

00000000010000000000

+ *offset*

0000000000011000

Fiziskā adrese

00000000010000011000

Maksimālā iespējamā adrese ir FFFFF.
Reālās adresācijas režīmā ir iespējams adresēt 2²⁰ = 1MB.

RTU DITF LDK U.SukovskisDatoru organizācija un asambleiri25

Programmas piemērs

Dotajā simbolu virknē atrast simbola * pozīcijas numuru un izvadīt uz ekrāna.

```
code segment
assume cs:code, ds:code
org 100h
start: jmp go

string db '01234567891*ABC', 0
buf db '000000$'

go: mov si, 0
mov ah, '*'
check: cmp string[si], 0
je notfound
cmp ah, string[si]
je found
inc si
jmp check

found:
```

RTU DITF LDK U.SukovskisDatoru organizācija un asambleiri26

Programmas piemērs

(pārveido bināru skaitli ASCII kodā)

```
found:
mov ax, si
inc ax
mov si, 5
mov bl, 10

d: div bl ;ax/bl= ah-atlikums, al-dalījums
add ah, 30h ; make ASCII digit
mov buf[si], ah
cmp al, 0 ; dalījums = 0?
je put
mov ah, 0
dec si
jmp d

put:
```

RTU DITF LDK U.SukovskisDatoru organizācija un asambleiri27

9

Programmas piemērs
(izvada rezultātu uz ekrāna)

```
put:      mov     ah,9
          mov     dx, offset buf
          int     21h
          jmp     done

notfound:      mov     dl,'?'
               mov     ah,2
               int     21h

done:         int     20h

code         ends
end          start
```

Adresācijas veidi

| Adresācija | Formāts | Noklusētais segmenta reģ. |
|------------------|--------------|---------------------------|
| Netiešā reģistra | [bx] | ds |
| | [bp] | ss |
| | [di] | ds |
| | [si] | ds |
| Bāzes relatīvā | label[bx] | ds |
| | label[bp] | ss |
| Tiešā indeksētā | label[si] | ds |
| | label[di] | ds |
| Bāzes indeksētā | label[bx+si] | ds |
| | label[bx+di] | ds |
| | label[bp+si] | ss |
| | label[bp+di] | ss |

Visos gadījumos var pieskaitīt konstanti, piem., `mov ax,buffer[si+2]`

Adresācijas veidi (turpinājums)

Ja lieto netiešo adresāciju, iespējami gadījumi, kad asambleira kompilators nevar noteikt operandu izmērus.

```
mov     es:[si], 0      ; KĻŪDA: nevar noteikt izmēru!
```

Jālieto garuma modifikators:

```
mov     byte ptr es:[si], 0      ; Atmiņā vienā baitā ieraksta 0
mov     word ptr es:[si], 0      ; Atmiņā vienā vārdā ieraksta 0
```

; Šajā gadījumā kompilators var noteikt operandu garumu:

```
mas     dw     100 dup (?)
txt     db     80 dup (?)
...
mov     mas[si], 0      ; Atmiņā vienā vārdā ieraksta 0
mov     txt[si], 0      ; Atmiņā vienā baitā ieraksta 0
```

Load Effective Address

```
buf    db    20 dup(0)
...
mov dx,offset buf    ; reģistrā dx ieraksta adreses buf nobīdes
                     ; vērtību, kuru aprēķina kompilācijas laikā

lea dx,buf            ; reģistrā dx ieraksta adreses buf nobīdes
                     ; vērtību, kuru aprēķina programmas
                     ; izpildes laikā

lea dx,buf[si+2]      ; reģistrā dx ieraksta nobīdes vērtību,
                     ; ņemot vērā arī reģistra si pašreizējo vērtību
```

```
lea reģistrs, atmiņa
```

Reģistrā ieraksta atmiņas adreses nobīdes vērtību, kuru aprēķina programmas izpildes laikā.

Operētājsistēmas funkciju lietošana

- Reģistrā **ah** ieraksta izpildāmās funkcijas numuru
- Nododamo parametru vērtības ieraksta reģistros
- Nodod vadību operētājsistēmas funkcijai, radot programmatūras pārtraukumu ar komandu **int 21h**
- Saņem rezultātus reģistros un/vai atmiņā

Operētājsistēmas funkciju lietošana (turpinājums)

Teksta izvade uz ekrāna, sākot no kursora pozīcijas

ah=9

ds:dx = izvadāmā teksta adrese

Teksta beigu pazīme ir '\$'.

Parasti reģistrs ds jau satur datu segmenta vērtību, tāpēc pietiek ierakstīt reģistrā dx teksta adreses *offset* vērtību.

```
message1 db 'Ievadi vārdu:$'
...
mov ah,9
lea dx, message1
int 21h
```

Operētājsistēmas funkciju lietošana (turpinājums)

Simbolu virknes ievade no tastatūras (ar *echo* uz ekrāna)

ah=0Ah
ds:dx = ievades bufera adrese
Rezultāts: buferī ievietots teksts ar CR simbolu beigās

Buferis teksta ievadei iepriekš jāsatgatavo - pirmajā baitā jāieraksta maksimālais ievadāmo simbolu skaits:

| | | | | | | | | | |
|-----|--|--|--|--|--|--|--|--|--|
| max | | | | | | | | | |
|-----|--|--|--|--|--|--|--|--|--|

Pēc ievades buferī ir ievietots faktiskais garums un teksts:

| | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|----|--|
| max | len | 'T' | 'E' | 'K' | 'S' | 'T' | 'S' | 0D | |
|-----|-----|-----|-----|-----|-----|-----|-----|----|--|

```
buf db 7, 0, 8 dup(0)
...
mov ah, 0Ah
mov dx, offset buf
int 21h
```

Ievadītais teksts sākas no adreses `buf+2` (nevis no adreses `buf`) !

Ciklu programmēšana

```
mov cx, 10
c: ...
loop c
```

Atkārtos
10 reizes

Komanda `loop` atņem no reģistra `cx` saturu 1 un, ja rezultāts nav 0, tad izpilda pāreju uz lezīmi.

To var izdarīt arī tā:

```
mov cx, 10
c: ...
dec cx
jnz c
```

Iekļauto ciklu programmēšana

```
mov cx, 5
c1: push cx
...
mov cx, 10
c2: push cx
...
pop cx
loop c2
...
pop cx
loop c1
```

Atkārtos
5 reizes

Atkārtos
10 reizes

Ciklu programmēšana

```
m dw -1, 2, 3, -4, 5
dw 1, -2, 3, 4, 5
rez dw 1, 2, -3, 4, -5
3 dup(0)
go: xor si, si ; matricas indekss
xor di, di ; rezultāta masiva indekss
mov cx, 3 ; rindu skaits
rows: push cx
xor ax, ax ; summa = 0
mov cx, 5 ; kolonnu skaits
cols: push cx
cmp m[si], 0 ; vai elements negatīvs?
jge next
add ax, m[si] ; pieskaita elementu summai
si, 2 ; matricas indekss
next: pop cx
mov rez[di], ax ; summa -> rezultāta masiva
add di, 2 ; rezultāta masiva indekss
pop cx
loop rows
```

Virkņu apstrāde

Pārrakstīt 80 baitu saturu no field1 uz field2:

```
field1 db 80 dup(?)
field2 db 80 dup(?)
;
; ...
; ievada field1 saturu
;
; ...
; pārkopē field1 uz field2
;
xor     si, si
mov     cx, 80
m:      mov     al, field1[si]
        mov     field2[si], al
        inc     si
        loop    m
```

Virkņu apstrāde

Salīdzināt tekstu, kas ievadīts laukā psw ar to, kas atrodas laukā etalon:

```
psw     db 8 dup(?)
etalon  db 'kaut kas'
;
; ...
;
xor     si, si
mov     cx, 8
m:      mov     al, psw[si]
        cmp     al, etalon[si]
        jne     nesakrit
        inc     si
        loop    m
```

Virkņu apstrāde (turpinājums)

Move String

```
movs    destination, source
movsb
movsw
```

1) pārsūta DS:SI => ES:DI

2) ja DF=0, tad SI=SI+n, DI=DI+n ; ja DF=1, tad SI=SI-n, DI=DI-n
n=1 baitiem, n=2 vārdiem

Adrešes *destination* un *source* kalpo tikai tam, lai kompilators komandas *movs* vietā varētu izvēlēties komandu *movsb* vai *movsw*, vadoties no *source* apraksta.

Reģistros SI un DI *offset* vērtības vienmēr jāieraksta programmai.
Tas nenotiek automātiski arī komandas *movs* gadījumā !

Virkņu apstrāde (turpinājums)

Pārrakstīt 80 baitu saturu no field1 uz field2:

```
field1 db 80 dup(?)
field2 db 80 dup(?)
;
lea si,field1 ; ds jau norāda uz datu segmentu
lea di,field2 ; es jau norāda uz datu segmentu
cld ; DF=0
mov cx,80 ; rep - atkārtojuma prefikss
rep movsb ;
```

Virkņu apstrāde (turpinājums)

Compare String

```
cmps destination, source
cmpsb
cmpsw
```

1) salīdzina DS:SI ar ES:DI un uzstāda karogu reģistra bitus
2) ja DF=0, tad SI=SI+n, DI=DI+n ; ja DF=1, tad SI=SI-n, DI=DI-n, n=1 vai n=2
Ērti izmantot ar atkārtojumu prefiksu **repe** (*repeat while equal*), lai atrastu pirmo atšķirību vai **repne** (*repeat while not equal*), lai atrastu pirmo sakritību.

Virkņu apstrāde (turpinājums)

Salīdzināt tekstu, kas ievadīts laukā psw ar to, kas atrodas laukā etalon:

```
psw db 8 dup(?)
etalon db 'kaut kas'
;
lea si,psw
lea di,etalon
cld
mov cx,8
repe cmpsb ; repe - atkārtojuma prefikss
jne nesakrit

; ... apstrāde gadījumam, ja virknes sakrīt

nesakrit:dec si ; apstrāde, ja atšķiras
dec di
```

Apakšprogramma, kas saņem parametrus reģistros

```
cseg      segment
          assume cs:cseg, ds:cseg
          org 100h
start:    jmp go
wrđ       dw 0f3h
buf       db '000000$'

ones      proc near
          push ax
          push cx
          xor bx,bx
tst:      test ax,0001h
          jz next
          inc bx
next:     shr ax,1      ; shift right
          loop tst
          pop cx
          pop ax
ones      endp
go:
```

RTU DITF LDK U.SukovskisDatoru organizācija un asamblei43

Apakšprogramma, kas saņem parametrus reģistros
(turpinājums)

```
go:       mov ax,wrđ
          mov cx,16
          call ones
; ...
; pārveido bx saturu no binārā koda uz ASCII virkni un izvada uz ekrāna
          int 20h
cseg      ends
          end start
```

RTU DITF LDK U.SukovskisDatoru organizācija un asamblei44

Apakšprogramma, kas saņem parametrus reģistros
(turpinājums)

call label

1) ja far procedūra, tad ievieto stekā CS vērtību

2) ja far procedūra, tad ieraksta reģistrā CS adreses label segmenta vērtību

3) ievieto stekā IP vērtību

4) ieraksta reģistrā IP adreses label offset vērtību

ret [n]

1) izņem no steka virsotnes vārdu un ievieto IP reģistrā

2) ja far procedūra, tad izņem no steka virsotnes vārdu un ievieto CS reģistrā

3) SP = SP + n]

RTU DITF LDK U.SukovskisDatoru organizācija un asamblei45

Apakšprogramma, kas saņem parametrus stekā

```
cseg      segment
          assume cs:cseg
          org     100h
start:    jmp     go
wrd       dw      005fh
count     dw      ?
buf       db      '00000$'

ones      proc     near
;          ... procedūras teksts
ones      endp

go:        push    count
          push     16
          push     wrd
          call     ones
          pop      count
; ...
; pārveido count saturu no binārā koda uz ASCII virkni un izvada uz ekrāna
          int     20h
cseg      ends
          end       start
```

pēc jmp go sp →

| |
|-------|
| count |
| 16 |
| wrd |
| IP |

ieejot procedūrā ones sp→

RTU DITF LDK U.Sukovskis

Datoru organizācija un asamblei

46

Apakšprogramma, kas saņem parametrus stekā (turpinājums)

```
ones      proc     near
          push     bp
          mov      bp,sp
          push     ax
          push     bx
          push     cx
          mov      bx,0
          mov      cx,[bp+6]
          mov      ax,[bp+4]
          test     ax,0001h
          jz       next
          inc      bx
          shr      ax,1
          loop     loop
          mov      [bp+8],bx
          pop      cx
          pop      bx
          pop      ax
          pop      bp
          ret      4
ones      endp
```

pēc jmp go sp →

| |
|-------|
| count |
| 16 |
| wrd |
| IP |
| bp |
| ax |
| bx |
| cx |

ieejot procedūrā ones sp→

← sp pēc pop count

← sp pēc ret 4

← [sp pēc ret bez parametra

← sp pēc pop bp

← sp pēc pop ax

← sp pēc pop bx

← sp pēc pop cx

RTU DITF LDK U.Sukovskis

Datoru organizācija un asamblei

47

Apakšprogrammas kompilēšana atsevišķā failā

```
main.asm
cseg      segment
          extrn    ones:far
go:        ...
          call     ones
          ...
cseg      ends
          end

ones.asm
cseg      segment
          public   ones
ones      proc     far
          ...
ones      endp
cseg      ends
          end
```

C:\> tasm ones

C:\> tasm main

C:\> tlink /t main+ones, maintest

```
graph TD
    ones[ones.asm] --> tasm1[tasm]
    main[main.asm] --> tasm2[tasm]
    tasm1 --> ones_obj[ones.obj]
    tasm2 --> main_obj[main.obj]
    ones_obj --> tlink[tlink]
    main_obj --> tlink
    tlink --> maintest[maintest.com]
```

RTU DITF LDK U.Sukovskis

Datoru organizācija un asamblei

48

16

COM un EXE programmas

Reģistru saturs, saņemot vadību no operētājsistēmas

| Reģistrs | COM | EXE |
|----------|--------|-----------------------------|
| CS | PSP | Segments ar ieejas punktu |
| IP | 100h | ieejas punkta <i>offset</i> |
| DS | PSP | PSP |
| SS | PSP | Segments ar 'STACK' |
| SP | 0FFFEh | 'STACK' segmenta izmērs |
| ES | PSP | PSP |

PSP -Program Segment Prefix

Maksimālais komandu un datu apjoms:

COM - viens segments 65536 - 256(PSP) - 2(stack) = 65278 baiti

EXE - vairāki segmenti

Vienkāršas .exe programmas piemērs

```
ASSUME cs:CSEG, ds:DSEG, ss:SSEG

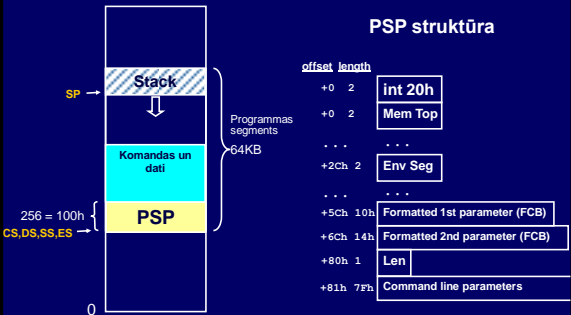
CSEG SEGMENT ; Code segment
begin:mov ax, DSEG ; Set data segment
      mov ds, ax
      mov ah, 9h ; Function 9
      lea dx, msg ; Load DX with offset of string
      int 21h ; Display string
      mov ah, 4ch ; Function 4ch
      mov al, 0 ; Return code
      int 21h ; Return to operating system
CSEG ENDS

DSEG SEGMENT ; Data segment
msg db "Sveiks!", 7, 13, 10, "$"
DSEG ENDS

SSEG SEGMENT STACK ; Stack segment
dw 64 dup(0)
SSEG ENDS

END begin
```

Program Segment Prefix (PSP)

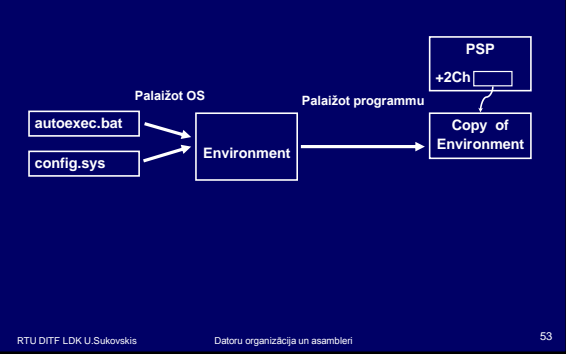


Parametru saņemšana no komandrindas
.com programmā

```
start: jmp go
; ...
go: xor cx, cx
mov cl, ds: [80h] ; length of command line
cmp cx, 0
jna noparms
mov si, 81h ; offset of parameters in PSP
chk: cmp byte ptr [si], 'a' ; convert
jnb nolwr ; all
cmp byte ptr [si], 'z' ; lowercase
ja nolwr ; command line
sub byte ptr [si], 32 ; characters
nolwr: inc si ; to uppercase
loop chk ;
; ...process list of parameters...
noparms:
```

RTU DITF LDK U. Sukovskis Datoru organizācija un asambleji 52

Operating System Environment



Environment

autoexec.bat

```
set include=C:\BC\include
set lib=C:\BC\lib
set mybuf=512
```

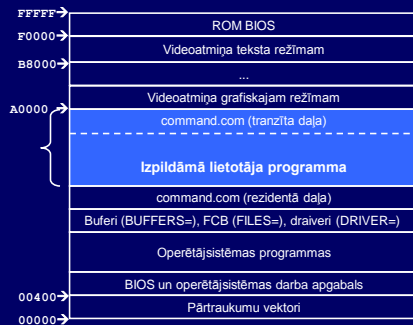
name1=value1 00
name2=value2 00
...
nameN=valueN 00
00
0001
full exe path 00
00

PSP
+2Ch

include=C:\BC\include 00 lib=C:\BC\lib 00 mybuf=512 00 00 0001 C:\temp\abc.com 00 00

RTU DITF LDK U. Sukovskis Datoru organizācija un asambleji 54

Atmiņas sadalījuma plāns



RTU DITF LDK U.Sukovskis

Datoru organizācija un asambleji

55

BIOS darba apgabals

Glabājas dažādas vērtības, kuras izmanto BIOS un operētājsistēmas programmas.

| | | |
|-----------|----------|--------------------------------|
| Piemēram, | | |
| 0000:0417 | 2 baiti | tastatūras stāvokļa biti |
| 0000:041E | 32 baiti | tastatūras buferis |
| 0000:0449 | 1 baits | video režīms |
| 0000:044A | 1 baits | simbolu skaits ekrāna rindā |
| 0000:046C | 4 baiti | laika skaitītājs |
| 0000:0471 | 1 baits | bits 7 = 1 nospiešs Ctrl-Break |

RTU DITF LDK U.Sukovskis

Datoru organizācija un asambleji

56

Pārtraukumu vektori

- Atmiņas pirmajā kilobaitā katros 4 baitos glabājas pārtraukuma vektori - ieejas punkta adrese pārtraukuma apstrādes programmai.

| | |
|--------|---------|
| low | high |
| offset | segment |

- Ir iespējami pārtraukumu numuri no 0 līdz 255.
- Atbilstošā vektora adresi iegūst, sareizinot pārtraukuma numuru ar 4.
- Piemēram, dubultvārdā ar adresi 0000:0020 glabājas pārtraukuma 8 (taimera pārtraukums) apstrādes programmas ieejas punkta adrese:

0000:0020 low high **A5FE 00F0** , t.i. adrese F000:FEA5

RTU DITF LDK U.Sukovskis

Datoru organizācija un asambleji

57

Pārtraukumi

- Programmatūras pārtraukumi (software interrupts), kurus rada CPU, izpildot komandu **int n**, kur n = pārtraukuma numurs

```
mov ah, 0      ; function 0 - set video mode
mov al, 3      ; text video mode
int 10h        ; BIOS interrupt
```

Programmatūras pārtraukumus nav iespējams maskēt.

- Aparatūras pārtraukumi (hardware interrupts), kurus rada iekārtas, kas pieslēgtas pie CPU ar Programmable Interrupt Controller (PIC)
- Iekšējie pārtraukumi, ar numuriem 0 - 4, kurus izmanto CPU iekšējām vajadzībām

Pārtraukuma apstrāde

Kad notiek *software* vai *hardware* pārtraukums, tad

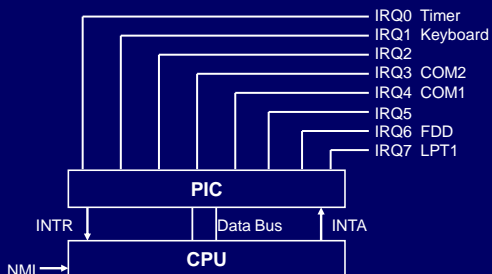
1. Stekā tiek ievietots karogu reģistra saturs,
2. Stekā tiek ievietots CS reģistra saturs,
3. Stekā tiek ievietots IP reģistra saturs,
4. Reģistros CS un IP tiek ievietotas *segment* un *offset* vērtības no pārtraukuma vektora.

Rezultātā vadību saņem pārtraukuma apstrādes programma.

Pārtraukuma apstrādes programmai jāsaģlabā stekā visu reģistru vērtības, kurus tā izmantos savā darbā.

Pēc apstrādes tai jāatjauno reģistru saturs no steka un jāizpilda komanda **iret**, kas atjauno no steka karoga reģistru, CS un IP, tādējādi atgriežoties pārtrauktajā programmā.

Aparatūras pārtraukums



Aparatūras pārtraukuma apstrāde

Kad vadību saņem aparatūras pārtraukuma apstrādes programma, tad :

- Ir aizliegti visi zemākas un vienādas prioritātes aparatūras pārtraukumi
- Pārtraukumu karogs ir uzstādīts 0. Pārtraukuma apstrādes programmai tas jāuzstāda 1 ar komandu sti, ja jāatļauj citu aparatūras pārtraukumu apstrāde.

Beidzot aparatūras pārtraukuma apstrādes programmu, obligāti par to jāpaziņo PIC mikroshēmai, lai tā atbloķē tekošās un zemākas prioritātes pārtraukumus.

To dara, iesūtot portā 20h vērtību 20h:

```
mov    al, 20h
out    20h, al
```

Aparatūras pārtraukuma maskēšana

Atšķirībā no programmatūras pārtraukumiem, aparatūras pārtraukumus var maskēt.

1. Var aizliegt procesoram apstrādāt aparatūras pārtraukumus, uzstādot pārtraukumu karogā 0 (ar komandu cli).

2. Var maskēt PIC ieejā ienākošos IRQ.

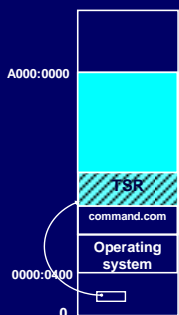
PIC ir pārtraukumu maskas reģistrs, kuru uzstāda caur portu 21h.

Vieninieks maskas bitā maskē bita numuram atbilstošo pārtraukumu.

Piemēram, tastatūras pārtraukuma maskēšana:

```
mov    al, 00000010b
out    21h, al
```

Terminate and Stay Resident



```
start:    jmp    go
data     ...
handler  proc    far
          push  ax
          ...
          push  ds
          push  cs
          pop   ds
          sti
          ...
          mov   al, 20h
          out   20h, al    ; EOI
          pop   ds
          ...
          pop   ax
          ired
          endp
handler  highaddr equ    this byte
go:      ...
          mov   dx, offset highaddr
          int    27h
```

Funkcijas darbam ar pārtraukumu vektoriem

- Nolasīt pārtraukuma vektoru

ah=35h

al = pārtraukuma numurs

Rezultāts: es:bx = pārtraukuma vektors

```
mov ah,35h
```

```
mov al,9
```

```
int 21h
```

- Uzstādīt pārtraukuma vektoru

ah=25h

al = pārtraukuma numurs

ds:dx = jaunais pārtraukuma vektors

```
mov ah,25h
```

```
mov al,9
```

```
mov dx,offset handler
```

```
int 21h
```

Tastatūra

- Katru reizi, kad nospiež taustiņu :

- tastatūras mikroshēma iesūta portā 60h taustiņa numuru - *scan code*. Esc - 1, !/1 - 2, @/2 - 3, ...

- tiek radīts *hardware* pārtraukums ar numuru 9

- Kad taustiņu atlaiž, atkal tiek radīts pārtraukums un pirms *scan code* portā 60h iesūta 0F0h

- Bultiņu, Insert, Del, PgUp,... taustiņu nospiešana dod vairāku baitu secību portā 60h (piem., → E0 4D, Del E0 53, PgDn E0 51, Print Screen E0 2A E0 37)

- Pārtraukumu 9 apstrādā ROM BIOS programma (vai cita programma, kuras ieejas punkta adrese atbilstošajā pārtraukuma vektorā).

Tastatūra (turpinājums)

- Ja pārtraukumu 9 apstrādā ROM BIOS programma:

- tā nolasa no porta 60h *scan code* un to analizē,

- ja pārtraukumu radījis simbola taustiņš, tad tiek izveidots ASCII kods un ierakstīts tastatūras buferī,

- ja ar simbola taustiņu reizē bijis nospiests Alt vai Ctrl vai arī ir nospiests funkcionālais taustiņš F1, F2, ..., tad tiek izveidots t.s. paplašinātais kods (piem., Alt-A - 30, Alt-B - 48, F1 - 59) un ierakstīts tastatūras buferī,

- ja pārtraukumu radījis stāvokļa (statusa) taustiņš (Shift, NumLock,...), tad tastatūras stāvokļa baitos (adresēs 0000:0417 un 0000:0418 tiek uzstādītas attiecīgo bitu vērtības.

Tastatūras buferis

0000:041E

32 baiti

2 baitu elements

0000:041A Head pointer

0000:041C Tail pointer

Elements :

| | |
|-------|-----------|
| ASCII | Scan code |
| low | high |

 vai

| | |
|-----|---------------|
| 00 | Extended code |
| low | high |

RTU DITF LDK U.Sukovskis Datoru organizācija un asambleji 67

Tastatūras stāvokļa biti

0417

Insert ON
CapsLock ON
NumLock ON
ScrollLock ON
Alt Down
Ctrl Down
LShift Down
RShift Down

0418

Insert Down
CapsLock Down
NumLock Down
ScrollLock Down
Pause
SysReq Down
Left Alt Down
Left Ctrl Down

Tastatūra (turpinājums)

- ROM BIOS programma apstrādā īpaši :
 - Kombināciju Ctrl-Break : rada pārtraukumu 1B, kura apstrādes programma uzstāda vieninieku baita 0000:0471 7.bitā
 - Kombināciju Shift-PrintScreen: rada pārtraukumu 5.
- Darbam ar tastatūru var izmantot pārtraukumu int 16h, kuru apstrādā BIOS programmas.

RTU DITF LDK U.Sukovskis Datoru organizācija un asambleji 69

Uzdevums

Uzrakstīt komandas, kas izveda uz ekrāna tekstu "Control", ja to izpildes laikā ir nospiests taustiņš Ctrl.

1. Kur glabājas informācija par taustiņa Ctrl stāvokli?

Baita ar adresi 0000:0417 2. bitā

2. Kā ierakstīt reģistrā AL atmiņas baitu, kura adrese ir 0000:0417?

```
xor    bx, bx
mov    es, bx
mov    al, es:0417h
```

3. Kā pārbaudīt vai reģistra AL 2. bitā ir vērtība 1?

```
and    al, 00000100b
jz     noctrl
```

3. Kā izvadīt tekstu uz ekrāna?

Tastatūras pārtraukuma apstrādes TSR

Tastatūras pārtraukuma apstrādes programma reaģē uz taustiņu kombināciju RightShift - Esc, ja ir izslēgts NumLock, un izveda ekrāna pirmās rindas pirmajā pozīcijā mirgojošu simbolu A baltā krāsā uz sarkana fona.

```
kbd      segment
          assume cs:kbd
          org     100h
start:    jmp     go

flag      db      '123456'
oldint9   dd      0
status    db      01h      ; RShift
scan      db      1        ; Esc

int9h     proc     far      ; Interrupt handler
          push    ds
          push    es
          push    ax
          push    bx
          push    cx
          mov     bx, cs
          mov     ds, bx
```

Tastatūras pārtraukuma apstrādes TSR (turpinājums)

```
          xor     bx, bx
          mov     es, bx
          test    byte ptr es:[0417h], 20h ; Numlock status ?
          js     getscan
          jmp     retold      ; CN - return

getscan:   in     al, 60h
          mov     ah, status
          and     ah, es:[0417h]
          cmp     ah, status      ; status ?
          jne     retold
          cmp     al, scan        ; scan code ?
          jne     retold

          mov     ax, 0b800h
          mov     es, ax
          mov     byte ptr es:[0], 65      ; character 'A'
          mov     byte ptr es:[1], 16*12*15; attribute
          jmp     rethw
```


Tastatūras pārtraukuma apstrādes TSR (turpinājums)

```
retold: pop cx
        pop bx
        pop ax
        pop es
        pop ds
        jmp [oldint9]

rethw:  in  al,61h      ; hardware housekeeping
        mov ah,al      ;
        or  al,80h     ;
        out 61h,al     ;
        xchg ah,al     ;
        out 61h,al     ;
        mov al,20h     ;
        out 20h,al     ; EOI
        pop cx
        pop bx
        pop ax
        pop es
        pop ds
        iret
int9h   endp
highbyte equ this byte
```

RTU DITF LDK U.SukovskisDatoru organizācija un asambleiri73

Tastatūras pārtraukuma apstrādes TSR (turpinājums)

```
highbyte equ this byte
ownflag  db 'LRKBDDU'
msgok    db 'Keyboard Driver installed',13,10,'$'
msgerr   db 'Keyboard driver is already active!',7,13,10,'$'
env      dw 0
go:      ...
        ...
        mov ax,3509h      ; get vector
        int 21h          ; es = segment from vector
        mov di,offset flag
        mov si,offset ownflag
        mov cx,6
        repe cmpsb       ; es:di == ds:si ?
        jne install      ; flags do not match - install
        mov dx,offset msgerr ; flags match - message
        mov ah,9
        int 21h
        int 20h
```

RTU DITF LDK U.SukovskisDatoru organizācija un asambleiri74

Tastatūras pārtraukuma apstrādes TSR (turpinājums)

```
install: mov si,offset ownflag ; set flag
        mov di,offset flag
        mov ax,ds
        mov es,ax
        mov cx,6
        rep movsb             ; ds:si -> es:di
        mov ax,3509h          ; get vector
        int 21h
        mov word ptr oldint9,bx
        mov word ptr oldint9+2,es
        mov dx,offset int9h    ; set vector
        mov ax,2509h
        int 21h
        mov dx,offset msgok
        mov ah,9
        int 21h
        mov es,ds:[2ch]        ; Environment seg from PSP
        mov ah,49h
        int 21h                ; release env seg
        mov dx,offset highbyte + 10h
        int 27h

kbd     ends
        end start
```

RTU DITF LDK U.SukovskisDatoru organizācija un asambleiri75

Darbs ar videoterminālu

- Videointerfeisu nodrošina videoadapters un monitors, kuriem jābūt ar saskaņotiem parametriem.
- Videoadapters attēlo uz monitora ekrāna informāciju, kas glabājas teksta vai grafiskajā videoatmiņā.
- Videoadapteru var pārslēgt uz vienu no teksta vai grafiskajiem režīmiem un tas attēlo informāciju no teksta vai grafiskās videoatmiņas.
- VGA tipa videoadapteriem
 - teksta režīmu videoatmiņas sākuma adrese ir **B800:0000**
 - grafisko režīmu videoatmiņas sākuma adrese ir **A000:0000**
- Režīmus pārslēdz, lietojot BIOS pārtraukumu **int 10h**

Video režīmi

- Teksta režīmi: 0, 1, 2, 3, 7
 - Piemēram, režīms 3 - 25 rindas, 80 kolonnas, 16 krāsas
- Grafiskie režīmi: 4, 5, 6, 8, 9, 10, ...
 - Piemēram, režīms 12h - 480x640, 16 krāsas
- Režīma numurs glabājas baitā 0000:0449
- Režīma ieslēgšana:


```

mov     ah, 0           ; funkcija 0 - set videomode
mov     al, 12h         ; a1 = videomode
int     10h
            
```
- Režīma nolasīšana:


```

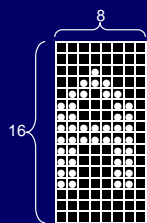
mov     ah, 0Fh         ; funkcija 0Fh - get videomode
int     10h
...           ; a1 = videomode
            
```

Teksta režīms

- VGA tipa videoadapteriem teksta režīma videoatmiņas sākuma adrese ir **B800:0000**
- Katram simbolam uz ekrāna atbilst 2 baiti videoatmiņā



- Pēc ASCII koda videoadapters atrod simbola attēla kodu un attēlo to uz ekrāna krāsā, kuru nosaka atribūta baits.
- Attēla kods glabājas videoadapters atmiņā, vai arī RAM un to var mainīt.



Atribūta baits

Intensitāte

b

R

G

B

i

R

G

B

Mirgošana

Fona krāsa

Simbola krāsa

Standarta krāsas

| | | | | | |
|------|---|---------|------|----|----------------|
| 0000 | 0 | Black | 1000 | 8 | Grey |
| 0001 | 1 | Blue | 1001 | 9 | Bright blue |
| 0010 | 2 | Green | 1010 | 10 | Bright green |
| 0011 | 3 | Cyan | 1011 | 11 | Bright cyan |
| 0100 | 4 | Red | 1100 | 12 | Bright red |
| 0101 | 5 | Magenta | 1101 | 13 | Bright magenta |
| 0110 | 6 | Brown | 1110 | 14 | Yellow |
| 0111 | 7 | White | 1111 | 15 | Bright white |

Krāsas numurs nosaka paletes reģistra numuru, kuru jāizmanto krāsas attēlošanai

Teksta izvade

Katra no 8 teksta lappusēm aizņem 4096 baitus. Pēc noklusēšanas tiek attēlota 0.lappuse.

Attēlojamo lappusi var izvēlēties tā:

```
mov ah, 5 ; function 5 - Set Page
mov al, 1 ; al = page number
int 10h
```

Ja jāizvada viens simbols rindas *row* pozīcijā *column* (kuras numurē no 1), tad nobīdi no videoatmiņas sākuma var aprēķināt tā:

$$\text{offset} = 2 \cdot (80 \cdot (\text{row} - 1) + \text{column} - 1) + 4096 \cdot \text{page}$$

Uzdevums:
Izvadīt ekrāna centrā burtus OK sarkanā krāsā uz balta fona.

RTU DITF LDK U.Sukovskis

Datoru organizācija un asamblei

80

Teksta izvade

```
mov ax, 13 ; row
mov bx, 39 ; col
dec ax ; row-1
mov di, 80 ;
mul di ; (row-1)*80
dec bx ; col-1
add ax, bx ; (row-1)*80 + (col-1)
add ax, ax ; *2
mov bx, 0B800h
mov es, bx
mov di, ax ;1996

mov byte ptr es:[di], 'O'
mov byte ptr es:[di+1], 7*16+4
mov byte ptr es:[di+2], 'K'
mov byte ptr es:[di+3], 7*16+4
```

RTU DITF LDK U.Sukovskis

Datoru organizācija un asamblei

81

27

Teksta režīma papildu iespējas

- Atribūta baitu vecākā bita nozīmi ir iespējams pārslēgt uz fona krāsas intensitāti vai uz simbola mirgošanu.

```
mov ah, 10h ; funkcija10h
mov al, 3    ; apakšfunkcija 3
mov bl, 0    ; 0=intensitāte, 1=mirgošana
int 10h
```

Teksta režīma papildu iespējas

- Ir 16 paletes reģistri, kuros glabājas krāsu kodi. Paletes reģistru saturu var mainīt.

```
mov ax, 1000h ; funkcija10h, apakšfunkcija 0
mov bh, 0     ; krāsa
mov bl, 1     ; 1. paletes reģistrs
int 10h
```

Teksta režīma papildu iespējas

- Var mainīt ekrāna robežas (*overscan boarder*) krāsu

```
mov ax, 1001h ; funkcija10h, apakšfunkcija 1
mov bh, 4     ; krāsa
int 10h
```

85

[illegible]

Darbs ar videoadapteri

- Ir 3 ierakstīšanas režīmi un 2 nolasišanas režīmi.
- Piemēram, ierakstīšanas režīmā 2 datu baita 4 jaunākie biti tiek ierakstīti visos *Latch* reģistru bitos perpendikulāri krāsu plaknēm, ievērojot bitu masku un plakņu masku. Baiti no *Latch* reģistriem tiek pārnesti uz videoatmiņas atbilstošajiem krāsu plakņu baitiem.
- Režīma numuru uzstāda grafiskā kontroliera (GDC) reģistrā 5.
- GDC reģistra numuru uzstāda portā 3CE, datus reģistrā iesūta caur portu 3CF.
- Bitu maskas vērtību uzstāda GDC reģistrā 8.
- Plakņu masku uzstāda sekvencera reģistrā 2.
- Sekvencera reģistra numuru uzstāda portā 3C4, datus reģistrā iesūta caur portu 3C5.

Piemērs

Apakšprogramma izvada vienu punktu uz ekrāna VGA 640x480 16 krāsu režīmā. Apakšprogrammai ir 3 parametri: punkta koordinātes X, Y un krāsa C.

Ekrāna augšējā kreisā stūra koordinātes ir X=0, Y=0.

Baita adresi videoatmiņā var aprēķināt pēc formulas:

`offset = 80*Y + X/8;`

Vienas punktu rindas 640 punktiem atbilst 80 baiti videoatmiņas vienā krāsu plaknē.

Bitu numurs atmiņas baitā ir atlikums no X dalījuma ar 8:

`bitnum = X % 8;`

Izsaukuma piemērs no C++ programmas:

```
int x, y, c;  
x = 0; y = 5; c = 12;  
setpx(x, y, c);
```

Piemērs (turpinājums)

```
void setpx(unsigned int X, unsigned int Y, unsigned int  
C)  
{  
    _asm{ mov     ax, Y  
          mov     dx, 80  
          mul     dx           ; 80*Y  
          mov     bx, X  
          ;  
          mov     cx, 3  
          shr     bx, cx       ; X/8  
          add     bx, ax       ; bx <- offset = 80*Y + X/8  
          mov     ax, 0A000h  
          mov     es, ax       ; video segment  
          mov     cx, 7       ; mask 00000111  
          and     cx, x       ; cx = bit number  
          mov     ah, 80h     ; one bit 10000000  
          shr     ah, cx       ; make mask of bits in ah
```

Piemērs (turpinājums)

```
mov dx, 3CEh ; port number
mov al, 5 ; GDC reg. 5 = mode register
out dx, al
inc dx ; port number = 3CFh
mov al, 2 ; write mode 2
out dx, al

mov dx, 3CEh ; port number
mov al, 8 ; GDC reg. 8=mask of bits reg.
out dx, al
inc dx ; port number = 3CFh
mov al, ah ; set mask of bits
out dx, al

mov dx, 3C4h ; port number
mov al, 2 ; Sequencer reg. 2=map mask reg.
out dx, al
inc dx ; port number = 3C5h
mov al, 0Fh ; map mask = 00001111
out dx, al
```

RTU DITF LDK U.Sukovskis

Datoru organizācija un asambleji

91

Piemērs (turpinājums)

```
mov al, es:[bx] ; read one byte to set latch regs
mov ax, C ; ax <- color
mov es:[bx], al ; write one byte to set one pixel
}
}
```

Izsaukuma piemērs no C++ programmas:

```
for (x = 0; x < 640; ++x)
  for (y = 0; y < 480; ++y)
    setpx(x, y, x+y);
```

RTU DITF LDK U.Sukovskis

Datoru organizācija un asambleji

92

Darbs ar taimeru

- Laika skaitīšanu nodrošina RTC - *Real Time Clock* un taimera mikroshēma, kas rada taimera pārtraukumus.
- Tekošā laika vērtība glabājas dubultvārdā sākot no adreses 0000:046C kā vesels skaitlis - laika impulsu (*ticks*) skaits no diennakts sākuma.
- Taimera pārtraukums rodas ik pēc 55 ms (18,2 reizes sekundē). To apstrādā BIOS programma, kas palielina dubultvārda saturu par 1.
- Operētājsistēmas funkcijas, kas dod diennakts laiku, pārrēķina šo veselo skaitli stundās, minūtēs, sekundēs un sekunžu simtdaļās.
- Var arī lietot BIOS pārtraukumu int 1Ah darbam ar laiku un datumu.

RTU DITF LDK U.Sukovskis

Datoru organizācija un asambleji

93

Funkcijas darbam ar laiku

Get Time

ah=2Ch

Rezultāts:

| | |
|-------------------------|--------------------------|
| ch - stundas | <code>mov ah, 2Ch</code> |
| cl - minūtes | <code>int 21h</code> |
| dh - sekundes | |
| di - sekundes simtdaļas | |

Set Time

| | |
|------------------------|--------------------------|
| ah=2Dh | <code>mov ah, 2Dh</code> |
| ch=stundas | <code>mov ch, 11</code> |
| cl= minūtes | <code>mov cl, 50</code> |
| dh= sekundes | <code>mov dx, 0</code> |
| di= sekundes simtdaļas | <code>int 21h</code> |

Rezultāts:

ah = 0, ja laiks uzstādīts, ah = 0FFh, ja laiks nepareizs

Funkcijas darbam ar datumu

Get Date

ah=2Ah

Rezultāts:

| | |
|---|--------------------------|
| al - nedēļas diena (0-svētdiena, 1-pirmdiena,...) | |
| cx - gads | |
| dh - mēnesis | <code>mov ah, 2Ah</code> |
| di - diena | <code>int 21h</code> |

Set Date

| | |
|-------------|---------------------------|
| ah=2Bh | <code>mov ah, 2Bh</code> |
| cx= gads | <code>mov cx, 2001</code> |
| dh= mēnesis | <code>mov dh, 11</code> |
| di= diena | <code>mov dl, 30</code> |
| | <code>int 21h</code> |

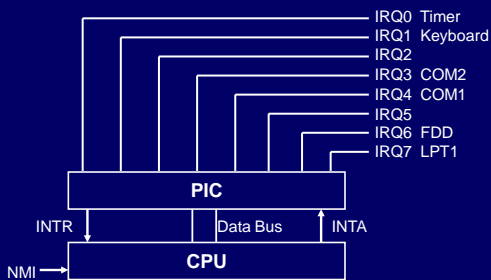
Rezultāts:

ah = 0, ja laiks uzstādīts, ah = 0FFh, ja laiks nepareizs

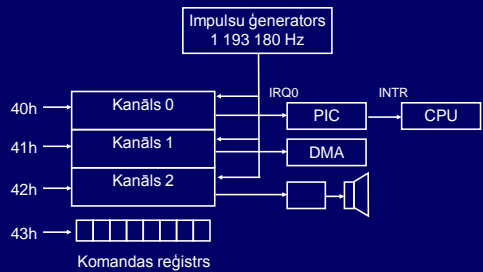
User Timer Interrupt

- Taimera pārtraukums rodas ik pēc 55 ms (18,2 reizes sekundē). To apstrādā BIOS programma, kas palielina dubultvārda saturu adresē 0000:046C par 1.
- Šī BIOS programma izpilda arī komandu `int 1Ch` (*User Timer Interrupt*). Pārtraukuma 1Ch apstrādes standartprogrammā ir tikai komanda `iret`.
- Šis pārtraukums paredzēts, lai lietotājs varētu apstrādāt taimera pārtraukumus, uzstādot savu pārtraukuma 1Ch apstrādes programmu.
- Pārtraukumu 1Ch šim mērķim nav ieteicams izmantot.

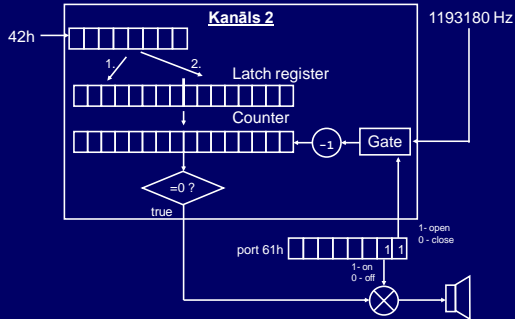
Aparatūras pārtraukums



Taimera mikroshēmas darbs



Taimera mikroshēmas kanāls 2



Piemērs

```

mov al, 10110110b ; 10-ch, 11 - 2 bytes, 011 - regime, 0 - bin
out 43h, al        ; command
mov ax, 1193       ; counter = 1193180 / 1000Hz
out 42h, al        ; low byte
mov al, ah
out 42h, al        ; high byte
in al, 61h         ; read port
push ax            ; and save
or al, 03h         ; enable gate and speaker
out 61h, al        ; start sound

; ... delay looping ...
pop ax             ; restore port value
out 61h, al        ; stop sound

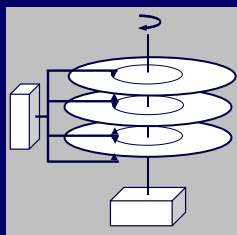
```

RTU DITF LDK U.Sukovskis

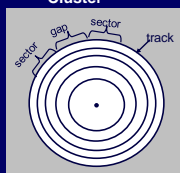
Datoru organizācija un asambleji

100

Disku atmiņas organizācija



- Sector
- Track
- Cylinder
- Cluster



Lieto NTFS un FAT formatēšanas sistēmas

RTU DITF LDK U.Sukovskis

Datoru organizācija un asambleji

101

Informācijas izvietojums uz diska (FAT)

| |
|----------------|
| Boot record |
| FAT1 |
| FAT2 |
| Root directory |
| Data area |

- Boot record - fiziski pirmais diska sektors. Satur informāciju par diska formātu: sektora izmērs, sektoru skaits klāsterī, FAT skaits utt., kā arī izpildāmu *boot* kodu.
- FAT1 - File Allocation Table, informācija par aizņemtajiem un brīvajiem diska klāsteriem.
- FAT2 - otra FAT kopija
- Root directory - saknes direktorijs ar fiksētu elementu skaitu.
- Data area - apgabals, kur izvieto failus un apakšdirektorijus

RTU DITF LDK U.Sukovskis

Datoru organizācija un asambleji

102

File Allocation Table

Starting Cluster No.

Directory Entry

File name

Attribute

Time

Date

08

Size

FAT

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F ...

09

08

19

1A

FF

Data area (clusters)

00 01 ... 08 09 0A 0B 0C 0D 0E 0F

10 11 ... 16 17 18 19 1A 1B 1C 1D

RTU DITF LDK U.Sukovskis

Datoru organizācija un asamblei

103

Boot sector

buffer

db

512

dup

(0)

boot

equ

buffer

res

db

11

dup

(0)

sectSize

dw

0

clustSize

db

0

resSects

dw

0

fatCount

db

0

rootSize

dw

0

totalSects

dw

0

media

db

0

fatSize

dw

0

trackSects

dw

0

heads

dw

0

hidnSects

dw

0

;

.....

RTU DITF LDK U.Sukovskis

Datoru organizācija un asamblei

104

Boot sector

; read boot sector using BIOS

mov

dl

,

0

;

0-A, 1-B, ...

mov

dh

,

0

;

head

mov

ch

,

0

;

cyl

mov

cl

,

1

;

sector

mov

al

,

1

;

count

mov

ah

,

2

;

read

mov

bx

,

offset

boot

;

es:bx

buffer

int

13h

; read boot sector using operating system

mov

al

,

0

;

0-A, 1-B, ...

mov

cx

,

1

;

count

mov

dx

,

0

;

sector

number

0,1,...

mov

bx

,

offset

boot

;

ds:bx

buffer

int

25h

RTU DITF LDK U.Sukovskis

Datoru organizācija un asamblei

105

| NTFS vs FAT (http://www.ntfs.com) | | | | | | |
|---|---|---|---|---|--|--|
| Criteria | NTFS | NTFS | exFAT | FAT32 | FAT16 | FAT12 |
| Operating System | Windows 2000 Windows XP Windows 2003 Server Windows 2008 Windows Vista Windows 7 | Windows NT Windows 2000 Windows XP Windows 2003 Server Windows 2008/Windows Vista Windows 7 | Windows CE 6.0 Windows Vista SP1 Windows 7 WinXP-KB953704 | DOS v7 and higher Windows 98 Windows ME Windows 2000 Windows XP Windows 2003 Server Windows Vista Windows 7 | DOS All versions of Microsoft Windows | DOS All versions of Microsoft Windows |
| Limitations | | | | | | |
| Max Volume Size | 2 ³¹ clusters minus 1 cluster | 2 ³¹ clusters minus 1 cluster | 128PB | 32GB for all OS, 2TB for some OS | 2GB for all OS, 4GB for some OS | 16MB |
| Max Files on Volume | 4,294,967,295 (2 ³² -1) | 4,294,967,295 (2 ³² -1) | Nearly Unlimited | 4194304 | 65536 | |
| Max File Size | 2 ³¹ bytes (16 EaBbytes) minus 1KB | 2 ³¹ bytes (16 TeraBytes) minus 64KB | 16EB | 4GB minus 2 Bytes | 2GB (Limit Only by Volume Size) | 16MB (Limit Only by Volume Size) |
| Max Clusters Number | 2 ³¹ clusters minus 1 cluster | 2 ³¹ clusters minus 1 cluster | 4294967295 | 4177918 | 65520 | 4080 |
| Max File Name Length | Up to 255 | Up to 255 | Up to 255 | Up to 255 | Standard - 8.3 Extended - up to 255 | Up to 254 |
| Overall Performance | | | | | | |
| Built-In Security | Yes | Yes | Yes minimal ACL only | No | No | No |
| Recoverability | Yes | Yes | Yes if TFAF activated | No | No | No |
| Performance | Low on small volumes High on Large | Low on small volumes High on Large | High | High on small volumes Low on large Average | Highest on small volumes Low on large Minimal on large volumes | High |
| Disk Space Economy | Max | Max | Max | Average | Minimal on large volumes | Max |
| Fault Tolerance | Max | Max | Yes if TFAF activated | Minimal | Average | Average |

NTFS

- Versijas
 - v1.0 1993
 - v1.1 NT 3.5 1994
 - v1.2 NT 3.51, NT 4 1995-1996
 - v3.0 Windows 2000 ("NTFS V5.0")
 - v3.1
 - Windows XP 2001 ("NTFS V5.1"),
 - Windows Server 2003 2003 ("NTFS V5.2")
 - Windows Vista 2005 ("NTFS V6.0") Windows Server 2008, Windows 7
- Visa informācija par datiem uz diska glabājas kā metadati īpašā failā MFT (*Master File Table*) - par failiem, mapēm, metafaiļiem
- Metafailos glabājas informācija, kas nodrošina darbu ar failu sistēmu

RTU DITF LDK U.SukovskisDatoru organizācija un asambleieri107

| Master File Table | | |
|-------------------|---------------------------------------|---|
| Segment Number | File Name | Purpose |
| 0 | \$MFT | Describes all files on the volume, including file names, timestamps, stream names, and lists of cluster numbers where data stream reside. Includes, security identifiers, and file attributes like "read only", "compressed", "encrypted", etc. |
| 1 | \$MFTMirr | Duplicate of the first vital entries of \$MFT, usually 4 entries (4 KB). |
| 2 | \$LogFile | Contains transaction log of file system metadata changes. |
| 3 | \$Volume | Contains information about the volume, namely the volume object identifier, volume label, file system version, and volume flags (mounted, chkdsk requested, requested \$LogFile resize, mounted on NT 4, volume serial number updating, structure upgrade request). This data is not stored in a data stream, but in special MFT attributes: If present, a volume object ID is stored in an \$OBJECT_ID record; the volume label is stored in a \$VOLUME_NAME record, and the remaining volume data is in a \$VOLUME_INFORMATION record. Note: volume serial number is stored in file \$Boot (below). |
| 4 | \$AttrDef | A table of MFT attributes which associates numeric identifiers with names. |
| 5 | \$Boot | Boot directory. Directory data is stored in \$INDEX_ROOT and \$INDEX_ALLOCATION attributes both named \$130. |
| 6 | \$Bitmap | An array of bit entries: each bit indicates whether its corresponding cluster is used (allocated) or free (available for allocation). |
| 7 | \$Boot | Volume boot record. This file is always located at the first clusters on the volume. It contains bootstrap code (see NTLDR \$BOOTLDR) and a BIOS parameter block, including a volume serial number and cluster numbers of \$MFT and \$MFTMirr. \$Boot is usually 5182 bytes long. |
| 8 | \$BadClus | A file which contains all the clusters marked as having bad sectors. This file simplifies cluster management by the chkdsk utility, both as a place to put newly discovered bad sectors, and for identifying unreferenced clusters. This file contains two data streams, even on volumes with no bad sectors: an unnamed stream contains bad sectors—it is zero length for perfect volumes; the second stream is named \$Bad and contains all clusters on the volume not in the first stream. |
| 9 | \$Secure | Access control list database which reduces overhead having many identical ACLs stored with each file, by uniquely storing these ACLs in this database only (contains two indices \$SIF: perhaps Security ID index and \$SDH: Security Descriptor Hash which index the stream named \$SDS containing actual ACL table). |
| 10 | \$UpCase | A table of unicode uppercase characters for ensuring case insensitivity in Win32 and DOS namespaces. |
| 11 | \$Extend | A filesystem directory containing various optional extensions, such as \$Quota, \$ObjId, \$Reparse or \$JunkInt. |
| 12 ... 23 | Reserved for \$MFT extension entries. | |
| usually 24 | \$Extend\$Quota | Holds disk quota information. Contains two index roots, named \$Q and \$Q. |
| usually 25 | \$Extend\$ObjId | Holds distributed link tracking information. Contains an index root and allocation named \$Q. |
| usually 26 | \$Extend\$Reparse | Holds reparse point data (such as symbolic links). Contains an index root and allocation named \$R. |
| 27 | \$file.ext | Beginning of regular file entries. |

● Konsultācija:

● Eksāmens:
