Assembler

Prvá časť: Úvod

Dôvodom vzniku tohto seriálu bol záujem čitateľov PC-REVUE. Keď som si čítal výber e-mailov, takmer vždy som narazil na čitateľa, ktorý mal záujem o assembler. A tak som sa rozhodol niečo v tomto smere urobiť. Pretože sa o túto problematiku zaujímam už niekoľko rokov, pustil som sa do písania tohto seriálu. Mal by mať približne nasledujúci obsah:

- základné pojmy,
- opis inštrukčného súboru,
- programovanie v assembleri (praktické príklady).

ZÁKLADNÉ POJMY

Assembler

Assembler je materským jazykom procesora. Procesor rozumie assembleru vo forme postupnosti núl a jednotiek. Tejto forme hovoríme strojový kód (anglicky machine code). Pretože tvoriť programy priamo v niektorej z číselných sústav (binárnej, dekadickej, hexadecimálnej) by bolo náročné, zaviedla sa jednoduchšia forma zápisu. Nazývame ju symbolický zápis inštrukcií. Z nasledujúcej tabuľky sa môžete dozvedieť, akým číslom niektorej z číselných sústav sa dá nahradiť daná inštrukcia.

SYMBOLICKÝ ZÁPIS	DEC	HEX	BIN
CLI	250d	0FAh	11111010b
STI	251d	0FBh	11111011b

Možno sa zamyslíte nad tým, prečo assembler používať, keď máme dostupné iné vyššie programovacie jazyky, ako PASCAL, TURBOC, C++ atď. Dôvodov môže byť niekoľko. Ani v jednom z vyšších programovacích jazykov nedosiahnete naprogramovanie takého programu, ktorý by bol rýchly a zároveň v pamäti zaberal čo najmenej miesta. Program v assembleri je možné jednoducho prenášať medzi rôznymi typmi počítačov. Výhodou je aj to, že máte úplnú kontrolu nad programom.

Pri niekoľkých inštrukciách (5 až 10) možno písať assembler i priamo v číslach. Umožňoval to už PASCAL 4.0, ktorý mal na to vytvorený príkaz INLINE. Na druhej strane sami uznáte, že napísať takto 1 až 2 kilobajtový program by bolo hotové trápenie.

I assembler má svoje mínus. Zdrojový text je i pri krátkom programe veľmi dlhý. V assembleri sa ťažšie hľadajú chyby. Programovanie aplikácií zaberie oveľa viac času ako pri vyšších programovacích jazykoch. Assembler napriek svojim kladným vlastnostiam slúži aj ako nástroj zlomyseľnosti. Mám na mysli písanie vírusov. Prevažná väčšina týchto programov je napísaná práve v assembleri.

Číselné sústavy

V assembleri pri písaní zdrojového textu je možné používať tieto tri základné číselné sústavy: binárnu, dekadickú, hexadecimálnu. Rozoberieme si ich podrobnejšie.

Binárna sústava

Základom binárnej sústavy je číslo 2. V tejto sústave môžu nastať dva rôzne stavy. Sú to stavy 0 a 1. Na identifikáciu čísel v binárnej sústave budeme používať na konci čísla znak b (všimnite si, že ide o prvé písmeno v slove binárna, podobne to bude aj s dekadickou a hexadecimálnou sústavou).

Napr. 01100100b = 100d

Skúsme si toto tvrdenie overiť.

$(0*2^7)+(1*2^6)+(1*2^5)+(0*2^4)+(0*2^3)+(1*2^2)+(0*2^1)+(0*2^0)=0+64+32+0+0+4+0+0=100$

Z toho vyplýva, že jednotlivé rády sú vlastne mocniny čísla dva (1, 2, 4, 8, 16, 32, 64, 128...). Pokúsme sa teraz urobiť prevod čísla 100 naspäť do binárnej sústavy.

Číslo, ktoré prevádzame, delíme číslom 2. Ak výsledkom delenia je celá časť, označíme tento stav hodnotou 0. V prípade, že výsledkom delenia je číslo s desatinnou časťou, označíme tento stav hodnotou 1. Desatinnú časť čísla „odsekneme“ a pokračujeme v delení, až kým nedosiahneme nulu.

Postup výpoctu	Stav
100 / 2 = 50	0
50 / 2 = 25	0
25 / 2 = 12.5	1
12 / 2 = 6	0
6 / 2 = 3	0
3 / 2 = 1.5	1
1 / 2 = 0.5	1
0 / 2 = 0	0

Takto získané binárne číslo teraz usporiadame zdola nahor. 01100100b – dostali sme opäť naše pôvodné číslo. Jednotlivé číslice v binárnom zápise budeme nazývať bity. Osem takto za sebou idúcich bitov nazveme bajt. Pretože aj bajt je svojím obsahom malý, používa sa dnes bežne predpona kilo, mega, giga.

Jeden bajt – $0^{7\ bit}\ 110010\ 0^{0\ bit}$ b

Prvý bit sprava sa označuje ako najnižší bit, má poradové číslo nula. Posledný bit sprava má poradové číslo sedem a nazývame ho najvyšší bit.

Binárna sústava sa najčastejšie používa pri logických inštrukciách a pri práci s grafikou.

Decimálna sústava

Základom dekadickej sústavy je číslo 10. Takisto si môžete všimnúť, že 10 je aj počet čísel, ktoré táto sústava používa - to platí všeobecne. Čísla dekadickej sústavy budeme označovať na konci čísla znakom d.

Číslo 65535d sa dá napísať ako:

$6*10^4 + 5*10^3 + 5*10^2 + 3*10^1 + 5*10^0 =$
$= 6*10000 + 5*1000 + 5*100 + 3*10 + 5*1 = 65535d$

Dekadickú sústavu používame takmer každý deň (v škole, v práci atď.), no pre assembler je oveľa výhodnejší nasledujúci typ sústavy.

Hexadecimálna sústava

Základom je číslo 16. Čísla tejto sústavy budeme označovať na konci čísla znakom h. Číslice väčšie ako deväť sa však nedajú zapísať jedným znakom, a preto bola vytvorená dohoda, že sa budú označovať prvými šiestimi písmenami abecedy. To, akú hodnotu nadobúdajú tieto písmená v iných sústavách, môžete vidieť v tabuľke.

HEX	DEC	BIN
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Výhodou hexadecimálnej sústavy je, že ak číslo 304Ah zapíšeme napríklad do 16-bitového registra AX (tento register sa dá rozdeliť na dva 8-bitové registre), potom v registri AH bude hodnota 30h a v registri AL hodnota 4Ah. Skúsme teraz číslo 304Ah previesť do dekadickej sústavy.

$3*16^3 + 0*16^2 + 4*16^1 + 10*16^0 = 12288 + 0 + 64 + 10 = 12362d$

Opačný prevod, späť na hexadecimálne číslo, je podobný prevodu do binárnej sústavy.

Výpocet	Zvyšok	HEX
12362 / 16 = 772	10	A
772 / 16 = 48	4	4
48 / 16 = 3	0	0
		3

Ak nám pri delení vyjde číslo menšie ako 16, potom toto číslo pripíšeme na koniec hexadecimálneho čísla. Číslo, ktoré sme takto dostali, prepíšeme zdola nahor takto: 304Ah. Dostali sme opäť naše pôvodné číslo.

Ešte jedna dôležitá vec. V zdrojovom texte sa musí pred hexadecimálnu číslicu, ktorá sa začína písmenom, písať znak 0, inak prekladač bude hlásiť chybové hlásenie **Undefined symbol .**

Napr.	0A000h	A000h
	dobre	zle

Procesor 8086

Procesor 8086 je prvým procesorom firmy INTEL. Má 14 16-bitových registrov. Vyššie procesory (286, 386...) sú s ním kompatibilné. To znamená, že programy napísané pre tento procesor sa dajú rovnako spustiť aj na vyšších procesoroch. Procesor umožňuje adresovať 220 pamäťových miest alebo, inak povedané, 1 MB fyzickej pamäte.

Registre procesora 8086

Registre si možno predstaviť ako špeciálne pamäťové miesta, umiestnené priamo v procesore. Sú oveľa rýchlejšie ako bežná pamäť RAM. Procesor v nich uchováva hodnoty, s ktorými práve pracuje. Procesor disponuje 14 16-bitovými registrami. Registre rozdeľujeme podľa použitia na:

1. všeobecné registre,
2. registre na špeciálne účely,
3. segmentové registre,
4. riadiace registre.

Všeobecné registre

16-bitové registre							
AX		BX		CX		DX	
AH	AL	BH	BL	CH	CL	DH	DL
8-bitové registre							

Každý zo všeobecných registrov sa dá rozdeliť na dva 8-bitové registre, pričom polovica 16-bitového registra označená H je horných osem bitov a polovica označená L je dolných osem bitov (pozri tabuľku).

AX – ACCUMULATOR – register AX nachádza časté použitie v matematických operáciách.

BX – BASE – používa sa pri nepriamom adresovaní.

CX – COUNTER – register CX býva naplnený hodnotou, ktorá najčastejšie predstavuje:

1. počet opakovaní cyklu

Napr.:

```
        MOV CX,10d
cyklus NOP
        LOOP cyklus
```

2. počet presunov (REP MOVSB)

Inštrukcia REP MOVSB slúži na kopírovanie bloku pamäte (vysvetlenie bude ďalej).

DX – DATA – register slúži na uchovávanie dát.

Registre na špeciálne účely

SI	DI	BP	SP

SI – SOURCE INDEX – register sa využíva pri indexovom adresovaní.
DI – DESTINATION INDEX – podobne ako register SI.
BP – BASE POINTER – adresácia parametrov a lokálnych premenných na zásobníku.
SP – STACK POINTER – ukazovateľ na zásobník (OFFSET).

Segmentové registre

CS	DS	ES	SS

CS – CODE SEGMENT – segmentová časť adresy kde je uložený program, ktorý sa bude vykonávať.
DS – DATA SEGMENT – segment určuje blok v pamäti, v ktorom sa nachádzajú dáta programu (napr.: číselné hodnoty, reťazce, premenné, buffer, konštanty a pod.).
ES – EXTRA SEGMENT – pomocný segmentový register.
SS – STACK SEGMENT – segmentová časť adresy zásobníka. Registre SS:SP dávajú dohromady adresu, na ktorej sa v pamäti nachádza zásobník.

Riadiace registre

IP	F

IP – INSTRUCTION POINTER - register obsahuje ofsetové adresy práve vykonávanej inštrukcie. Celá adresa je daná obsahmi registrov CS:IP. Obsah registra IP nie je možné meniť priamo. Zmenu možno vykonať iba inštrukciami skoku (JMP), volaním podprogramu (CALL), návratom z podprogramu (RET), návratom z prerušenia (IRET). F – FLAGS - je to 16-bitový register, ktorý obsahuje informácie o výsledku vykonaných inštrukcií. Procesor 8086 využíva však iba 9 bitov z tohto registra. Ďalšie bity sa začali používať až od procesora 286. Nám však bude postačovať týchto 9 základných bitov.

15															0
				OF	DF	IF	TF	SF	ZF		AF		OF		CF

OF – OVERFLOW FLAG – príznak sa nastaví, ak pri aritmetických operáciách došlo k pretečeniu výsledku.

Príklad:

```
MOV AL,255d ;naplň reg. AL číslom 255
MOV BL,255d ;naplň reg. BL číslom 255
MUL BL ;vynásob tieto dve čísla
```

Po vykonaní inštrukcie násobenia sa výsledok 65025 už nezmestí do 8-bitového registra, dôjde k pretečeniu a nastaví sa príznak OF.

DF – DIRECTION FLAG – tento príznak riadi smer spracovania reťazcových operácií. Pri DF nastavenom na jednotku inštrukciou STD sa obsah registrov SI a DI inkrementuje (reťazce sa spracúvajú odpredu), pri DF nastavenom na nulu inštrukciou CLD sa registre SI a DI dekrementujú (reťazce sa spracúvajú odzadu).

IF – INTERRUPT ENABLE FLAG – obsahuje stav maskovateľného prerušenia. Ak je nastavený na nulu, prerušenie je zakázané, v opačnom prípade je povolené (IF=1). Tento bit je možné nulovať inštrukciou CLI alebo nastaviť na jednotku inštrukciou STI.

TF – TRAP FLAG – príznak sa používa na ladenie programov. Jeho nastavenie spôsobí, že procesor po vykonaní každej inštrukcie vygeneruje prerušenie INT 1.

SF – SING FLAG – určuje, či je výsledok operácie záporný. Ak je príznak nastavený na jednotku (ST=1), výsledok operácie je záporný, v opačnom prípade kladný (SF=0).

Príklad:

```
MOV AL,100d ;naplň reg. AL číslom 100
MOV BL,200d ;naplň reg. BL číslom 200
SUB AL,BL ;odčítaj tieto dve čísla
```

ZF – ZERO FLAG – je nastavený na 1, ak výsledok operácie je 0.

Príklad:

```
MOV AL,100d ;naplň reg. AL číslom 100
MOV BL,100d ;naplň reg. BL číslom 100
SUB BL,AL ;odčítaj tieto dve čísla
```

AF – AUXILIARY FLAG – AF=1, ak došlo k prenosu z bitu 3 do bitu 4 bez ohľadu na dĺžku operandu. Používa sa v BCD aritmetike.

PF – PARITY FLAG – je nastavený na 1, ak dolných osem bitov výsledku vykonávanej operácie obsahuje párny počet jednotiek (POZOR - neznamená to, že číslo je párne!).

CF – CARRY FLAG – príznak je nastavený na jednotku, ak došlo k prenosu cez najvyšší bit.

Príklad:

```
MOV AL,100d ;naplň reg. AL číslom 100
MOV AH,200d ;naplň reg. AH číslom 200
ADD AH,AL ;spočítaj tieto dve čísla
```

Ak sčítame tieto dve čísla, dostaneme výsledok 300, ale tento výsledok už nie je možné uložiť naspäť do 8-bitového registra. Dôjde k pretečeniu, nastaví sa príznak CF=1 a výsledok bude 44.

Hodnotu príznaku je možné meniť aj nasledujúcimi inštrukciami:

STC – nastaví hodnotu carry flagu na 1,

CLL – vynuluje carry flag.

Príznaky CF, PF, AF, ZF, SF, OF nastavuje procesor po vykonaní niektorých inštrukcií. Príznaky TF, IF, DF nastavuje programátor a slúžia na riadenie činnosti procesora.

Druhá časť: Adresácia

Adresovanie pamäte

Procesor 8086 má segmentovú architektúru pamäte. Je schopný adresovať 1 megabajt pamäte. Tomu zodpovedá aj šírka adresovej zbernice – 20 bitov. Ako sme už vysvetlili, procesor 8086 používa štyri rôzne segmentové registre: CODE, DATA, STACK a EXTRA. V segmente CODE (CS) sú uložené strojové inštrukcie, v segmente DATA (DS) sú uložené dáta programu, segment STACK (SS) obsahuje zásobník a segment EXTRA (ES) sa zvyčajne používa pre ďalšie dáta.

Logická adresa sa skladá z dvoch 16-bitových hodnôt: z adresy segmentu a offsetu. V reálnom režime sa adresa zložená zo segmentu a offsetu prekladá priamo na adresu fyzickú. Z tejto tzv. logickej adresy sa vytvára 20-bitová fyzická adresa tak, že segmentová časť adresy sa posunie o štyri bity vľavo a k takto vzniknutému číslu sa pripočíta offset. Tým vznikne 20-bitová adresa ukazujúca na konkrétne miesto v pamäti. Ukážeme si to na adrese videopamäte v textovom režime.

```
adresa segmentu: 1011 1000 0000 0000 0000 - 0B800H (začiatok
videopamäte)
offset: + 0000 0000 0001 1000 - 00018H (posunutie)
výsledná adresa: 1011 1000 0000 0001 1000 - B8018H (20-bitová fyzická
adresa)
```

Štandardný zápis logickej adresy je SEGMENT:OFFSET. SEGMENT - je blok pamäte, ktorý má veľkosť 64 KB. OFFSET - predstavuje adresu, pomocou ktorej sa môžeme pohybovať v danom segmente, teda v bloku 64 KB.

Segmenty sa môžu aj prekrývať. Ide o programy s príponou *.COM. Tento typ programov môže mať maximálne veľkosť 64 KB (1 segment). Z toho dôvodu všetky segmenty (CODE, DATA, STACK, EXTRA) sa musia začínať na tej istej adrese.

Prerušenie

Najprv si vysvetíme, čo vlastne rozumieme pod pojmom prerušenie. Prerušenie je to signál, ktorý procesoru prikáže, aby zastavil vykonávanie hlavného programu a začal sa zaoberať iným programom. Podľa toho, čím je prerušenie generované, rozlišujeme:

a) **HARDVÉROVÉ prerušenia**

- NON MASCABLE INTERRUPT (NMI) – nemaskovateľné prerušenie,
- MASCABLE (INTR) – maskovateľné prerušenie

Vývod NMI je určený pre prerušenie, ktoré nie je možné zakázať. Slúži na signalizáciu havarijných stavov počítača (napr. pokles napájacieho napätia, chyba parity operačnej pamäte...). NMI má vyššiu prioritu ako INTR.

Vývod INTR je väčšinou budený ďalším obvodom – radičom prerušenia 8259. INTR je maskovateľné prerušenie, pretože je ho možné programovo zakázať vynulovaním bitu IF v príznakovom registri procesora. Čiže ak je nastavený na nulu, prerušenie je zakázané, v opačnom prípade je povolené (IF=1). Tento bit je možné nulovať inštrukciou CLI alebo nastaviť na jednotku inštrukciou STI.

Tabuľka 1: Hardvérových prerušení (v poradí podľa priority)

IRQ0	TIMER (prerušenie systémového časovača)
IRQ1	KEYBOARD (prerušenie klávesnice)
IRQ2	SLAVE 8259 (druhý 8259)
IRQ8	REAL TIME CLOCK (hodiny)
IRQ9	SOFTWARE REDIRECTED TO IRQ2
IRQ10	RESERVED (voľný)
IRQ11	RESERVED (voľný)
IRQ12	RESERVED (voľný)
IRQ13	NUMERIC COPROCESOR (matematický koprocesor)
IRQ14	DISK CONTROLLER (adaptér hard disku)
IRQ15	RESERVED (voľný)
IRQ3	COM2 OR COM4 (sériový port 2)
IRQ4	COM1 OR COM3 (sériový port 1)
IRQ5	LPT2 (paralelné prerušenie tlačiarne)
IRQ6	FLOPPY DISK (radič pružných diskov)
IRQ7	LPT1 (paralelné prerušenie tlačiarne)

b) **SOFTVÉROVÉ prerušenia**

- vzniknuté inštrukciou INT n, kde n je číslo z rozsahu 0-255

Procesor 8086 rozlišuje 256 možných prerušení. Každému prerušeniu prislúcha 32-bitová logická adresa (segment:offset). Táto adresa sa nazýva vektor prerušenia a ukazuje na miesto v pamäti, kde sa začína podprogram, ktorý sa vykoná pri vyvolaní daného prerušenia. Vektorom prerušenia je vyhradená pamäť od adresy 0000:0000 s dĺžkou 1024 bajtov. Pre každý vektor prerušenia sú rezervované 4 bajty (256*4=1024). V prípade prerušenia je vykonaný podprogram, ktorého logická adresa je v tabuľke vektorov na adrese (číslo prerušenia * 4). (Tabuľka 2)

Adresovanie v Assembleri

Poznáme niekoľko spôsobov adresovania. Jednotlivé možnosti si ukážeme pomocou inštrukcie MOV. Táto inštrukcia má dva operandy, označované ako cieľ a zdroj. Tieto operandy môžu byť registre, pamäťové miesta alebo konštanty (číselné hodnoty). Napr.: **MOV AX, 2222H** – po vykonaní inštrukcie bude register AX obsahovať hodnotu 2222H Na miesto v pamäti sa pri zápise inštrukcií odkazujeme tým spôsobom, že offsetovú časť adresy zapíšeme do hranatých zátvoriek (napr.: **MOV AX,[1234h]**).

Priame adresovanie

Je to najjednoduchší spôsob adresovania. Operand inštrukcie tvorí priamo offsetová adresa, napr.: **MOV AX,[00FFh]**. Čiže z adresy 00FFh sa vyberie hodnota a uloží sa do registra AX. Častejší prípad je taký, že adresa je nahradená návestím. Napr.:

```
FARBA DB 7
    .
    .
    .
ŠTART: MOV AL,[FARBA]
ADD AL,1
MOV [FARBA],AL
```

Takto to vyzerá v zdrojovom súbore, po preklade súboru turbo assemblerom sa návestie FARBA nahradí konkrétnou adresou.

Nepriame adresovanie

Offsetová časť adresy je určená obsahom niektorého z registrov SI, DI, BX, BP, napr.: MOV BX,[DI] alebo MOV BL,AL. Z offsetu adresy danej obsahom registra DI sa načíta hodnota do registra BX. Druhý prípad znázorňuje presun medzi registrami. Hodnota, ktorá je uložená v registri AL, sa presunie do registra BL, pričom obsah registra AL sa nezmení.

Bázové adresovanie

Využíva na vytvorenie offsetovej adresy bázové registre BP, BX a konštantu. Offsetová adresa je daná súčtom obsahu bázového registra a konštanty. Používa sa pri adresovaní pevnej dátovej štruktúry (napr.: poľa). Príklad:

```
MOV AX,[BX+N]
```

V tomto prípade bázový register ukazuje na začiatok poľa a pomocou konštanty N sa pohybujeme po jednotlivých prvkoch. Napr.:

```
MOV AX,[BX+00]
MOV AX,[BX+04]
```

Indexové adresovanie

Je podobné ako bázové adresovanie s tým rozdielom, že namiesto bázových registrov sa používajú indexové registre SI a DI. Tento spôsob adresovania sa používa pri dátových štruktúrach ako pole, reťazce a pod. Príklad:

```
MOV AX,[POLE+DI]
MOV AX,[POLE+SI]
```

Konštanta POLE predstavuje offsetovú adresu začiatku poľa a indexový register ukazuje na konkrétny prvok poľa.

Bázové – indexové adresovanie

Ide o kombináciu predchádzajúcich dvoch spôsobov adresovania. Offsetová časť adresy je tvorená súčtom obsahu bázového registra (BX,BP), indexového registra (SI,DI) a konštanty. Tento typ adresovania sa používa na adresovanie prvkov dynamického poľa. Príklad:

```
MOV AX,[BX+SI+N]
```

Register AX sa naplní obsahom položky N štruktúry, ktorá je prvkom poľa určeného registrom SI, pričom začiatok tohto poľa je na adrese určenej bázovým registrom BX. Príklad:

```
MOV CX,[BX+SI]
```

V registri BX je začiatok dynamického poľa a v registri SI je adresa prvku v dynamickej oblasti. MOV BL,[BX][SI] – tento zápis je ekvivalentný predchádzajúcemu.

ZÁSOBNÍK

Procesor 8086 používa zásobník typu LIFO (Last In First Out). V preklade to znamená, že to čo sa uloží posledné, sa vyberie ako prvé. Pre zásobník sa používa operačná pamäť RAM. Adresa segmentu, v ktorom je uložený zásobník, sa nachádza v registri SS (Stack Segment). Register SP (Stack Pointer) tvorí offset adresy zásobníka a ukazuje na jeho vrchol. Je to miesto, kam bola uložená posledná hodnota.

```
SS : SP          HODNOTA
64DC:270A        63A3
64DC:2708        025E
64DC:2706        64A1
64DC:2704        0044 <- Toto je vrchol zásobníka (posledná ulozená
                 hodnota)
```

Pred uložením ďalšej 16-bitovej hodnoty na zásobník sa hodnota registra SP zmenší o 2, po vybratí hodnoty zo zásobníka sa zase zväčší o 2. Procesor používa zásobník na uloženie návratovej adresy pri volaní podprogramu inštrukciou CALL a aj pri prerušení. Programátori používajú zásobník na odovzdávanie obsahu registrov, pre predávanie parametrov pri volaní podprogramu (takéto odovzdávanie parametrov využívajú hlavne jazyky ako C, PASCAL a pod.).

Na uloženie 16-bitovej hodnoty na zásobník sa používa inštrukcia PUSH N, kde N je register AX, BX, CX, DX, DI, SI... Naopak, na vybratie tejto hodnoty použijeme inštrukciu POP N, kde N je zase jeden z registrov.

Tabuľka 2

ADRESA	OPIS	ADRESA PODPROGRAMU	INŠTRUKCIA
0000:0000	delenie nulou	0019:9257	INT 0
0000:0004	krokovanie	0070:06F4	INT 1
0000:0008	NMI	0EEE:0016	INT 2
0000:000C	break point	0070:06F4	INT 3
0000:0010	pretečenie	0070:06F4	INT 4
.	.	.	
.	.	.	
.	.	.	
0000:03FF	rezervované	0282:F000	INT 255

Inštrukčný súbor procesora 8086
A dostali sme sa až k inštrukčnému súboru. Cieľom tejto časti bude prebrať všetky bežné inštrukcie. Pri inštrukciách budem vždy uvádzať typ procesora, od ktorého je daná inštrukcia použiteľná, príznaky, ktoré inštrukcia mení, príklady použitia a konečne stručný opis inštrukcie.
Príznaky si označíme týmito jednopísmenovými skratkami:
O – OVERFLOW FLAG
D – DIRECTION FLAG
I – INTERRUPT FLAG
T – TRAP FLAG
S – SING FLAG
Z – ZERO FLAG
A – AUXILIARY CARRY FLAG
P – PARITY FLAG
C – CARRY FLAG

Pod jednotlivými príznakmi sa bude vyskytovať jeden z týchto symbolov:
– príznak sa nemení
0 príznak je vždy nulovaný
1 príznak je vždy nastavený
* príznak je nastavený podľa výsledku danej operácie
? príznak je zmenený náhodne

AAA (ASCII adjust after addition)
PRÍZNAKY: O D I T S Z A P C
PROCESOR: 8086
? - - - ? ? * ? *
PRÍKLADY: AAA
OPIS: Inštrukcia prevedie výsledok sčítania dvoch číslic (obidve vo formáte voľného BCD kódu) zase do voľného BCD kódu nasledujúcim algoritmom:
AL <- AL+6
AH <- AH+1
A <- 1
C <- A
AL <- AL AND 0FH
Výsledok sčítania musí byť v registri AL a skutočná hodnota súčtu by nemala byť väčšia ako 9, pretože inak môže byť výsledok chybný! Inštrukcia AAA sa používa po inštrukciách ADD, ktoré pracujú s číslami vo formáte voľného BCD kódu.

AAD (ASCII adjust before division)
PRÍZNAKY: O D I T S Z A P C
PROCESOR: 8086
? - - - * * ? *
PRÍKLADY: AAD
OPIS: Inštrukcia prevádza dvojmiestne číslo vo voľnom formáte BCD uložené v registri AX na binárne číslo týmto algoritmom:
AL <- AH * 10 + AL
AH <- 0
Výsledok sa uloží do registra AL. Inštrukcia sa používa pred delením v prípade, že čísla sú vo voľnom BCD kóde, a teda ich treba najprv previesť do binárneho kódu.

AAM (ASCII adjust AX after multiply)
PRÍZNAKY: O D I T S Z A P C
PROCESOR: 8086
? - - - * * ? * ?
PRÍKLADY: AAM OPIS Inštrukcia prevádza obsah registra AL na dve čísla vo voľnom formáte BCD, ktoré ukladá do registrov AH a AL. Prevod prebieha takto:
AH <- AL div 10
AL <- AL mod 10
Inštrukcia sa používa po násobení (alebo delení) dvoch čísel vo voľnom formáte BCD.

AAS (ASCII adjust AL after subtraction)
PRÍZNAKY: O D I T S Z A P C
PROCESOR: 8086
? - - - ? ? * ? *
PRÍKLADY: AAS
OPIS: Inštrukcia prevádza výsledok odčítania dvoch číslic vo voľnom formáte BCD zase do voľného formátu BCD nasledujúcim algoritmom:
AL <- AL-6
AH <- AH-1
A <- 1
C <- 1
AL <- AL AND 0FH
Inštrukcia sa používa po odčítaní vo voľnom BCD formáte.

ADC (Add with carry)
PRÍZNAKY: O D I T S Z A P C
PROCESOR: 8086
* - - - * * * * *
PRÍKLADY: ADC operand1, operand2
ADC AX, 0FFFAh
ADC CX, DX
ADC BX, word ptr[DI]
ADC BL, AL
OPIS: Inštrukcia vykoná súčet dvoch operandov s hodnotou bitu Carry podľa vzorca: operand1 <- operand1 + operand2 + Carry.
Inštrukcia sa používa pri sčítaní dlhých čísel typu integer, ktoré nie je možné sčítať vykonaním jednej inštrukcie procesora 8086.

ADD (Add)
PRÍZNAKY: O D I T S Z A P C
PROCESOR: 8086
* - - - * * * * *
PRÍKLADY: ADD operand1, operand2
ADD AX, 0FFFAh
ADD CX, DX
ADD BX, word ptr[DI]
ADD AL, 2
OPIS: Inštrukcia vykoná súčet dvoch operandov podľa vzorca: operand1 <- operand1 + operand2
Inštrukcia sa používa pri sčítaní celých čísel (integer).

Tretia časť: Inštrukčný súbor proceorov x86

V tejto časti budeme pokračovať opisom inštrukčného súboru procesora 8086. Možno sa vám bude zdať nudná, ale na zvládnutie programovania v assembleri je nevyhnutná.

❑ **AND *(Logical AND)***

Príznaky							
O	D	I	T	S	Z	A	P	C
0	–	–	–	*	*	?	*	0

PRÍKLADY:
AND operand1,operand2
AND AX,0FF02h
AND CX,DX
AND AL,byte ptr [DI]
AND AL,1111000b

OPIS: Inštrukcia vykoná logický súčin dvoch operandov podľa vzorca:
operand1 <- operand1 and operand2
Logický súčin má výsledok 1 práve vtedy, keď obidva operandy majú hodnotu 1, inak je výsledok 0.
Názorne si to ukážeme na príklade:

```
          1 1 1 1 0 0 0 1
AND       1 0 1 1 0 1 0 1
          1 0 1 1 0 0 0 1
```

Výsledok operácie AND.

❑ **CALL *(Call procedure)***

Príznaky							
O	D	I	T	S	Z	A	P	C
–	–	–	–	–	–	–	–	–

PRÍKLADY:
CALL operand1
CALL near DOM – slovo near označuje krátky skok na procedúru DOM
CALL far DOM – slovo far označuje ďaleký skok na procedúru DOM
CALL cls
CALL ES:[DI]

OPIS: Inštrukcia vykoná skok do podprogramu. Najprv sa do zásobníka uloží návratová adresa pre inštrukciu RET a potom skok do podprogramu.

❑ **CBW** ***(Convert byte to word)***

Príznaky								
O	D	I	T	S	Z	A	P	C
0	–	–	–	*	*	?	*	0

PRÍKLADY: CWD

OPIS: Inštrukcia konvertuje 8-bitovú hodnotu so znamienkom v registri AL na 16-bitovú hodnotu so znamienkom. Výsledok je uložený do registra AX. Konverzia prebehne tak, že všetky bity registra AH sa naplnia najvyšším bitom registra AL.

❑ **CLC** ***(Clear carry flag)***

Príznaky								
O	D	I	T	S	Z	A	P	C
–	–	–	–	–	–	–	–	0

PRÍKLADY: CLC

Opis: Inštrukcia nastaví na nulu príznak Carry vo flag registri.

❑ **CLD** ***(Clear direction flag)***

Príznaky								
O	D	I	T	S	Z	A	P	C
–	0	–	–	–	–	–	–	–

PRÍKLADY: CLD

OPIS: Inštrukcia nastaví na nulu príznak Direction vo flag registri. Výsledkom toho je, že inštrukcie na prácu s reťazcami automaticky inkrementujú obsahy registrov SI a DI. Príznak Direction sa uplatní pri prehľadávaní reťazcov.

❑ **CLI** ***(Clear interrupt flag)***

Príznaky								
O	D	I	T	S	Z	A	P	C
–	–	0	–	–	–	–	–	–

PRÍKLADY: CLI

OPIS: Inštrukcia nastaví príznak Interrupt na nulu a tým zakáže prerušenie. Inštrukcia sa používa pri maskovateľnom prerušení.

❑ **CMC** ***(Complement carry flag)***

Príznaky								
O	D	I	T	S	Z	A	P	C
–	–	–	–	–	–	–	–	*

PRÍKLADY: CMC

OPIS: Inštrukcia neguje príznak Carry.

❑ **CMP** ***(Compare two operands)***

Príznaky								
O	D	I	T	S	Z	A	P	C
*	–	–	–	*	*	*	*	*

PRÍKLADY:
CMP operand1,operand2
CMP AL, 10h
CMP AL, BL
CMP byte ptr [SI],10
CMP CX, word ptr [DI]

OPIS: Inštrukcia nastaví príznaky tak, ako keby sa vykonalo odčítanie druhého registra od prvého registra, obsahy obidvoch registrov zostanú zachované.

❑ **CMPS (Compare string operand)**

Príznaky								
O	D	I	T	S	Z	A	P	C
*	–	–	–	*	*	*	*	*

PRÍKLADY:
CMPS operand1,operand2
CMPSB
CMPSW

OPIS: Inštrukcia sa používa na porovnanie dvoch blokov pamäte. Začiatok prvého bloku je určený dvojicou registrov DS:SI. Začiatok druhého bloku je určený dvojicou registrov ES:DI. Po porovnaní sa obsah registrov SI a DI zväčší, resp. zmenší (podľa príznaku Direction flag) o 1 alebo 2 v závislosti od toho, aké veľké dáta sa porovnávajú. Porovnanie prebieha buď po 8 bitoch CMPSB DI, SI <- DI, SI +/- 1 alebo po 16 bitoch CMPSW DI, SI <- DI, SI +/- 2

Maximálny počet porovnávaných položiek určuje register CX. Ak použijeme pred inštrukciou prefix REPE (opakuj, pokiaľ je rovné), porovnávanie sa ukončí buď pri nájdení prvého rozdielu medzi blokmi dát, alebo po porovnaní takého množstva položiek, aké udáva register CX. Môžeme pred inštrukciou použiť aj prefix REPNE (opakuj, pokiaľ je rôzne), v tom prípade sa porovnávanie ukončí buď pri nájdení prvej zhody medzi blokmi dát, alebo po porovnaní takého množstva položiek, aké udáva register CX. Ak nepoužijeme nijaký prefix, uskutoční sa porovnanie iba prvých dvoch položiek bloku dát bez ohľadu na výsledok porovnania.

Prefix inštrukcie spôsobí zmenu inštrukcie, ktorá nasleduje. Používa sa napr. na opakovanie inštrukcie do splnenia určitej podmienky, obyčajne do CX=0. Jeho dĺžka je 1 bajt a píše sa pred inštrukciu, napr.: REPE CMPSB.

❑ **DAA** ***(Decimal adjust AL after addition)***

Príznaky								
O	D	I	T	S	Z	A	P	C
?	–	–	–	*	*	*	*	*

PRÍKLADY: DAA

OPIS: Inštrukcia vykoná dekadickú korekciu po sčítaní dvoch čísel v tesnom BCD formáte.

❑ **DAS** ***(Decimal adjust AL after subtraction)***

Príznaky								
O	D	I	T	S	Z	A	P	C
?	–	–	–	*	*	*	*	*

PRÍKLADY: DAS

OPIS: Inštrukcia vykoná dekadickú korekciu po odčítaní dvoch čísel v tesnom BCD formáte.

❑ **DEC** ***(Decrement)***

Príznaky								
O	D	I	T	S	Z	A	P	C
?	–	–	–	*	*	*	*	*

PRÍKLADY:
DEC operand1
DEC AL
DEC BX
DEC byte ptr [DI]

OPIS: Inštrukcia odčíta od operandu jednotku: operand1 = (operand1 – 1)\

❑ **DIV** ***(Unsigned divide)***

Príznaky								
O	D	I	T	S	Z	A	P	C
?	–	–	–	?	?	?	?	?

PRÍKLADY:
DIV operand1
DIV CX
DIV byte ptr [DI]

OPIS: Inštrukcia vykoná celočíselné delenie. V prípade, že dôjde k deleniu nulou, generuje sa prerušenie INT 0. Delenec, deliteľ, podiel a zvyšok sú uložené v registroch podľa nasledujúcej tabuľky: delenec deliteľ podiel zvyšok

delenec	deliteľ	podiel	zvyšok
AX	8-bitový register	AL	AH
DX:AX	16-bitový register	AX	DX

❑ **HLT** ***(Halt)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

PRÍKLADY: HLT

OPIS: Inštrukcia uvedie procesor do stavu HALT. Dôsledkom toho je uvoľnenie zberníc a prerušenie behu programu. Z tohto stavu sa možno dostať nasledujúcimi spôsobmi:
– prerušením, ak je povolené
– nemaskovateľným prerušením NMI
– resetom procesora (stlačením klávesov CTRL+ALT+DEL súčasne)

❑ **IDIV** ***(Signed divide)***

Príznaky								
O	D	I	T	S	Z	A	P	C
?	-	-	-	?	?	?	?	?

PRÍKLADY:
IDIV operand1
IDIV CX
IDIV byte ptr [DI]

OPIS: Inštrukcia vykoná celočíselné delenie čísla so znamienkom. Delenec, deliteľ, podiel a zvyšok sú uložené v registroch podľa nasledujúcej tabuľky:

delenec	deliteľ	podiel	zvyšok
AX	8-bitový register	AL	AH
DX:AX	16-bitový register	AX	DX

❑ **IMUL** ***(Signed multiply)***

Príznaky								
O	D	I	T	S	Z	A	P	C
*	-	-	-	?	?	?	?	*

PRÍKLADY:
IMUL operand1
IMUL operand1, operand2
IMUL DX, word ptr [DI]
IMUL BL

OPIS: Inštrukcia vykoná násobenie so znamienkom.

❑ **IN** ***(Input from port)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

PRÍKLADY:
IN operand1, operand2
IN AL, DX
IN AL, 0FFh
IN AX, DX

OPIS: Inštrukcia číta dáta z portu do registra AL, prípadne AX.

❑ **INC** ***(Increment)***

Príznaky								
O	D	I	T	S	Z	A	P	C
*	-	-	-	*	*	*	*	-

PRÍKLADY:
INC operand1
INC AL
INC BX
INC word ptr [DI]

OPIS: Inštrukcia pričíta k operandu jednotku: operand1 = (operand1 + 1)

❑ **INT** ***(Call to interrupt procedure)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	0	0	-	-	-	-	-

PRÍKLADY:
INT operand1
INTO
INT 21H
INT 10H

OPIS: Inštrukcia vykoná softvérové prerušenie programu. Operand1 udáva typ prerušenia. Inštrukcia INTO vyvolá prerušenie číslo 4, ak je nastavený príznak Overflow.

❑ **IRET, IRETD** ***(Interrupt return)***

Príznaky								
O	D	I	T	S	Z	A	P	C
*	*	*	*	*	*	*	*	*

PRÍKLADY:
IRET
IRETD

OPIS: Inštrukcia sa používa na návrat z prerušenia. Po vykonaní inštrukcie sa zo zásobníka vyberie hodnota pre register IP, CS a F a dôjde k skoku na adresu, ktorú spolu vytvoria registre CS:IP.

❑ **Inštrukcie skokov**
si preberieme v úspornej forme, pretože všetky sú použiteľné od procesora 8086 a nemenia ani nenastavujú nijaký príznak. Príklady zápisu týchto inštrukcií sú všetky rovnaké, napr.: JA návestie, JC návestie atď.

Príznaky								
O	D	I	T	S	Z	A	P	C
*	*	*	*	*	*	*	*	*

❑ **JA** ***(Jump if above)*****, JNBE** ***(Jump if not below or equal)***
Inštrukcie podmieneného skoku. Skok na návestie sa vykoná v prípade, že príznak Carry=0 a súčasne príznak Zero=0. Ak pred inštrukciou skoku bola inštrukcia CMP operand1,operand2 – skok sa vykoná iba vtedy, ak bol operand1>operand2.

❑ **JAE** ***(Jump if above or equal)*****, JNB** ***(Jump if not below)*****, JNC** ***(Jump if not carry)***
Skok na návestie sa vykoná v prípade, že príznak Carry=0. Ak pred inštrukciou skoku bola inštrukcia CMP operand1,operand2 – skok sa vykoná iba vtedy, ak bol operand1>=operand2.

❑ **JB** ***(Jump if below)*****, JC** ***(Jump if carry)*****, JNAE** ***(Jump if not above or equal)***
Skok na návestie sa vykoná v prípade, že príznak Carry=1. Ak pred inštrukciou skoku bola inštrukcia CMP operand1,operand2 – skok sa vykoná iba vtedy, ak bol operand1>operand2.

❑ **JBE** ***(Jump if below or equal)*****, JNA** ***(Jump if not above)***
Skok na návestie sa vykoná v prípade, že príznak Carry=1 alebo príznak Zero=1. Ak pred inštrukciou skoku bola inštrukcia CMP operand1,operand2 – skok sa vykoná iba vtedy, ak bol operand1>=operand2.

❑ **JCXZ** ***(Jump if CX register is 0)*****, JECXZ** ***(Jump if ECX register is 0)***
Skok na návestie sa vykoná v prípade, že register CX, prípadne ECX obsahuje nulu.

❑ **JE** ***(Jump if equal)*****, JZ** ***(Jump if zero)***
Skok na návestie sa vykoná v prípade, že príznak Zero=1. Ak pred inštrukciou skoku bola inštrukcia CMP operand1,operand2 – skok sa vykoná iba vtedy, ak bol operand1=operand2.

❑ **JG** ***(Jump if greater)*****, JNLE** ***(Jump if not less or equal)***
kok na návestie sa vykoná v prípade, že príznak Zero=0 a súčasne príznak Sing=Overflow. Ak pred inštrukciou skoku bola inštrukcia CMP operand1,operand2 – skok sa vykoná iba vtedy, ak bol operand1>operand2.

❑ **JGE (Jump if greater or equal), JNL (Jump if not less)**
Skok na návestie sa vykoná v prípade, že príznak Sing je zhodný s príznakom Overflow. Ak pred inštrukciou skoku bola inštrukcia CMP operand1,operand2 – skok sa vykoná iba vtedy, ak bol operand1>=operand2.

❑ **JL** ***(Jump if less)*****, JNGE** ***(Jump if not greater or equal)***
Skok na návestie sa vykoná v prípade, že príznak Sing je rôzny od príznaku Overflow. Ak pred inštrukciou skoku bola inštrukcia CMP operand1,operand2 – skok sa vykoná iba vtedy, ak bol operand1<operand2.

❑ **JLE** ***(Jump if above or equal)*****, JNG** ***(Jump if not greater)***
Skok na návestie sa vykoná v prípade, že príznak Sing je rôzny od príznaku Overflow alebo príznak Zero=1. Ak pred inštrukciou skoku bola inštrukcia CMP operand1,operand2 – skok sa vykoná iba vtedy, ak bol operand1>=operand2.

❑ **JNE** ***(Jump if not equal)*****, JNZ** ***(Jump if not zero)***
Skok na návestie sa vykoná v prípade, že príznak Zero=0. Ak pred inštrukciou skoku bola inštrukcia CMP operand1, operand2 – skok sa vykoná iba vtedy, ak bol operand1<>operand2.

❑ **JNO** ***(Jump if not overflow)***
Skok na návestie sa vykoná v prípade, že príznak Overflow=0.

❑ **JNP** ***(Jump if not parity)*****, JPO** ***(Jump if parity odd)***
Skok na návestie sa vykoná v prípade, že príznak Parity=0.

❑ **JNS** ***(Jump if not sing)***
Skok na návestie sa vykoná v prípade, že príznak Sing=0 (číslo je kladné).

❑ **JO** ***(Jump if overflow)***
Skok na návestie sa vykoná v prípade, že príznak Overflow=1.

❑ **JP** ***(Jump if parity)*****, JPE** ***(Jump if parity even)***
Skok na návestie sa vykoná v prípade, že príznak Parity=1.

❑ **JS** ***(Jump if sing)***
Skok na návestie sa vykoná v prípade, že príznak Sing=1 (číslo je záporné).

❑ **JUMP** ***(Jump)***
Inštrukcia vykoná nepodmienený skok. Doteraz všetky skoky boli podmienené. Príklady zápisu inštrukcie: JMP [DI], JMP ES:[SI], JMP SHORT návestie, JMP ptr word [SI], JMP návestie.

❑ **LAHF** ***(Load flags into AH register)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

PRÍKLADY: LAHF

OPIS: Inštrukcia skopíruje dolných 8-bitov flag registra do registra AH. Register AH bude obsahovať tieto príznakové bity: S, Z, A, P, C rozložené takto:

bit	7	6	5	4	3	2	1	0
príznak	S	Z	-	A	-	P	-	C

Bity označené – majú nedefinovanú hodnotu

❑ **LDS** ***(Load full pointer DS)***

Príznaky								
O	D	I	T	S	Z	A	P	C
?	?	?	?	?	?	?	?	?

PRÍKLADY:
LDS operand1, operand2
LDS BX, dword ptr [DI]

Opis: Inštrukcia číta štyri bajty od adresy určenej operandom2. Prvé dva bajty sa ukladajú do segmentového registra DS a ďalšie dva bajty do registra určeného operandom1.

❑ **LEA** ***(Load effective address)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

PRÍKLADY:
LEA operand1, operand2
LEA AX, návestie

OPIS: Inštrukcia prenáša efektívnu adresu do registra.

❑ **LES** ***(Load full pointer into ES)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

PRÍKLADY:
LES operand1, operand2
LES CX, dword ptr [DI]

OPIS: Inštrukcia číta štyri bajty od adresy určenej operandom2. Prvé dva bajty sa ukladajú do segmentového registra ES a ďalšie dva bajty do registra určeného operandom1.

❑ **LODS (Load string operand)**

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

PRÍKLADY:
LODS operand1
LODSB
LODSW

OPIS: Inštrukcia prenáša do registra AL, prípadne AX byte (bajt), word (slovo) z adresy určenej registrami DS:[SI]. Po prenesení položky sa k registru SI pripočíta alebo odčíta (podľa nastavenia príznaku Direction) dĺžka prenášaneéj položky v bajtoch. Ak použijeme pred inštrukciou prefix REP, prenášanie sa ukončí po prenesení takého množstva položiek, aké obsahuje register CX. Ak nepoužijeme nijaký prefix, uskutoční sa presunutie iba jednej položky dát bez ohľadu na výsledok porovnania.

❑ **LOOP** ***(Loop control with CX counter)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

PRÍKLADY:
LOOP operand1
LOOP návestie

OPIS: Inštrukcia odpočíta od registra CX jednotku a porovná register CX s nulou. Ak je register CX nenulový, vykoná sa skok na návestie v rozsahu -128 až +127 bajtov. V opačnom prípade sa pokračuje vo vykonávaní nasledujúcej inštrukcie.

❑ **LOOPE, LOOPZ** ***(Loop control with CX counter)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

PRÍKLADY:
LOOPE operand1
LOOPZ operand1
LOOPE návestie

OPIS: Inštrukcia odpočíta od registra CX jednotku a porovná register CXs nulou. Ak je register CX nenulový a príznak Zero=1, vykoná sa skok na návestie v rozsahu -128 až +127 bajtov. V opačnom prípade sa pokračuje vo vykonávaní nasledujúcej inštrukcie.

❑ **LOOPNE, LOOPNZ** ***(Loop control with CX counter)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

PRÍKLADY:
LOOPNE operand1
LOOPNZ operand1
LOOPNE návestie

OPIS: Inštrukcia odpočíta od registra CX jednotku a porovná register CX s nulou. Ak je register CX nenulový a príznak Zero=0, vykoná sa skok na návestie v rozsahu –128 až +127 bajtov. V opačnom prípade sa pokračuje vo vykonávaní nasledujúcej inštrukcie.

❑ **MOV (Move data)**

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

PRÍKLADY:
MOV operand1, operand2
MOV AL, BL
MOV AX, CX
MOV AL,byte ptr [DI]

OPIS: Inštrukcia naplní operand1 operadom2, pričom operand2 zostane nezmenený. Operand1 <- operand2.

❑ **MOVS** ***(Loop control with CX counter)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

PRÍKLADY:
MOVS operand1, operand2
MOVSB
MOVSW

OPIS: Inštrukcia sa používa na kopírovanie bloku pamäte na iné miesto v pamäti. Začiatok kopírovaného bloku je určený dvojicou registrov DS:SI. Adresa, kam sa má blok skopírovať, je určená dvojicou registrov ES:DI. Po kopírovaní sa obsah registrov SI a DI zväčší, resp. zmenší (podľa príznaku Direction flag) o 1 alebo 2 v závislosti od toho, aké veľké dáta sa kopírujú. Kopírovanie prebieha buď po 8 bitoch MOVSB DI, SI <- DI, SI +/- 1, alebo po 16 bitoch MOVSW DI, SI <- DI, SI +/- 2

Maximálny počet kopírovaných položiek určuje register CX. Ak použijeme pred inštrukciou prefix REP (opakuj, pokiaľ je CX>0), kopírovanie sa ukončí po skopírovaní takého množstva položiek, aké udáva register CX. Ak nepoužijeme nijaký prefix, uskutoční sa kopírovanie iba prvej položky bloku dát bez ohľadu na obsah registra CX.

❑ **MUL** ***(Unsigned multiplication of AL or AH)***

Príznaky								
O	D	I	T	S	Z	A	P	C
*	-	-	-	?	?	?	?	*

PRÍKLADY:
MUL operand1
MUL BL
MUL word ptr [SI]

OPIS: Inštrukcia sa používa na násobenie so znamienkom. Násobenie sa vykonáva podľa veľkosti operandu1 takto: 1. operand1 je 8-bitový – Operand1 sa vynásobí registrom AL a výsledok sa uloží do registra AX. Ak je výsledok väčší ako bajt, nastavia sa príznaky Overflow a Carry, v opačnom prípade sa obidva príznaky vynulujú. 2. operand1 je 16-bitový – Operand1 sa vynásobí registrom AX a výsledok sa uloží do registrov DX, AX tak, že nižšia časť výsledku je v registri AX a vyššia časť je v registri DX. Ak je výsledok väčší ako slovo (word), nastavia sa príznaky Overflow a Carry, v opačnom prípade sa obidva príznaky vynulujú.

❑ **NEG** ***(Two's complement negation)***

Príznaky								
O	D	I	T	S	Z	A	P	C
*	-	-	-	*	*	*	*	*

PRÍKLADY:
NEG operand1
NEG AL
NEG AX

OPIS: Inštrukcia zmení znamienko operandu1 takto: operand1 <- (0 - operand1)

❑ **NOP** ***(No operation)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

PRÍKLADY: NOP

OPIS: Inštrukcia nevykoná nijakú operáciu. Používa sa napr.: na vyplnenie miesta v programe na neskoršie vloženie ďalších dát.

❑ **OR** ***(Logical inclusive OR)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	*	*	?	*	0

PRÍKLADY:
OR operand1, operand2
OR AL,10h
OR BL,byte ptr [DI]
OR AX,1111000011110000b

OPIS: Inštrukcia vykoná logický súčet operandov. Napr.:

```
        11110001
OR      10110101
        11110101
```

❑ **OUT** ***(Output to port)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

PRÍKLADY:
OUT operand1, operand2
OUT [10h], AL
OUT DX, AL

OPIS: Inštrukcia zapisuje dáta do portu z registra AL, prípadne AX.

❑ **POP** ***(Pop a word from the stack)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

PRÍKLADY:
POP operand1
POP AX
POP SI

Opis: Inštrukcia vyberie 16-bitové číslo (word) zo zásobníka a uloží ho do operandu1. Potom sa register SP zväčší o 2 a ukazuje na predposlednú vloženú hodnotu.

❑ **POPF** ***(Pop from stack into flags register)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

PRÍKLADY: POPF

OPIS: Inštrukcia vyberie flag register zo zásobníka.

❑ **PUSH** ***(Push operand onto the stack)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

PRÍKLADY:
PUSH operand1
PUSH AX
PUSH 1234h

OPIS: Inštrukcia vloží 16-bitové číslo (word), obsah registra, pamäte do zásobníka. Postup je takýto: najprv sa register SP zmenší o 2, takže ukazuje na voľné miesto. Potom sa zapíše obsah registra, pamäte alebo konštanta na toto voľné miesto.

❑ **PUSHF** ***(Push flags register onto the stack)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

PRÍKLADY: PUSHF

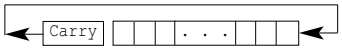
OPIS: Inštrukcia uloží flag register do zásobníka.

❑ **RCL** ***(Rotate left through carry)***

Príznaky								
O	D	I	T	S	Z	A	P	C
*	-	-	-	-	-	-	-	*

PRÍKLADY:
RCL operand1, operand2
RCL AL, 1
RCL AX, BL
RCL word ptr [DI], 1

OPIS: Inštrukcia vykoná rotáciu operandu1 doľava o toľko bitov, koľko určuje operand2. Obrázok znázorňuje prácu inštrukcie:

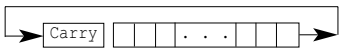


❑ **RCR** ***(Rotate right through carry)***

Príznaky								
O	D	I	T	S	Z	A	P	C
*	-	-	-	-	-	-	-	*

PRÍKLADY:
RCR operand1, operand2
RCR AL, 1
RCR AX, CL
RCR word ptr [DI], 1

OPIS: Inštrukcia vykoná rotáciu operandu1 doprava o toľko bitov, koľko určuje operand2. Obrázok znázorňuje prácu inštrukcie:



❑ **REP** ***(Repeat)*****, REPE** ***(Repeat while equal)*****,**
REPZ ***(Repeat while zero)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

PRÍKLADY:
REP inštrukcia
REPE inštrukcia
REPZ inštrukcia

OPIS: Prefix zabezpečí opakovanie niektorej z nasledujúcich inštrukcií: MOVS, LODS, CMPS, SCAS. Podrobnejší opis je pri jednotlivých inštrukciách.

❑ **REPNE** ***(Repeat while not equal)*****, REPNZ** ***(Repeat while not zero)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

PRÍKLADY:
REPNE inštrukcia
REPNZ inštrukcia

OPIS: Prefix zabezpečí opakovanie niektorej z nasledujúcich inštrukcií: MOVS, LODS, CMPS, SCAS.

❑ **RET** ***(Return from procedure)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

PRÍKLADY:
RET
RET operand1
RET 10h

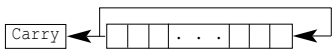
OPIS: Inštrukcia vykoná návrat z procedúry (podprogramu). Operand1 určuje, koľko bajtov sa odstráni zo zásobníka po vybratí návratovej adresy.

❑ **ROL** ***(Rotate left)***

Príznaky								
O	D	I	T	S	Z	A	P	C
*	-	-	-	-	-	-	-	*

PRÍKLADY:
ROL operand1, operand2
ROL AL, 1
ROL AX, CL
ROL word ptr [DI], 1

OPIS: Inštrukcia vykoná rotáciu operandu1 doľava o toľko bitov, koľko určuje operand2. Obrázok znázorňuje prácu inštrukcie:

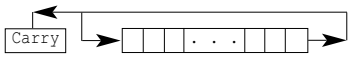


❑ **ROR** ***(Rotate right)***

Príznaky								
O	D	I	T	S	Z	A	P	C
*	-	-	-	-	-	-	-	*

PRÍKLADY:
ROR operand1, operand2
ROR AL, 1
ROR AX, CL
ROR word ptr [DI], 1

OPIS: Inštrukcia vykoná rotáciu operandu1 doprava o toľko bitov, koľko určuje operand2. Obrázok znázorňuje prácu inštrukcie:



❑ **SAHF** ***(Store Ah into flags)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	*	*	*	*	*

PRÍKLADY: SAHF

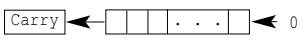
OPIS: Inštrukcia uloží obsah registra AH do nižšieho bajtu flag registra.

❑ **SAL (Shift arithmetic left)**

Príznaky								
O	D	I	T	S	Z	A	P	C
*	-	-	-	-	-	-	-	*

PRÍKLADY:
SAL operand1, operand2
SAL AL, 1
SAL AX, DL
SAL word ptr [SI], 1

OPIS: Inštrukcia vykoná posuv operandu1 doľava o toľko bitov, koľko určuje operand2. Obrázok znázorňuje prácu inštrukcie:

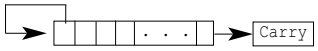


❑ **SAR** ***(Shift arithmetic right)***

Príznaky								
O	D	I	T	S	Z	A	P	C
*	-	-	-	-	-	-	-	*

PRÍKLADY:
SAR operand1, operand2
SAR CL, 1
SAR AX, BL
SAR word ptr [SI], 1

OPIS: Inštrukcia vykoná posuv operandu1 doprava o toľko bitov, koľko určuje operand2. Obrázok znázorňuje prácu inštrukcie:



❑ **SBB** ***(Integer subtraction with borrow)***

Príznaky								
O	D	I	T	S	Z	A	P	C
*	-	-	-	*	*	*	*	*

PRÍKLADY:
SBB operand1, operand2
SBB AL, 1
SBB AX, CX
SBB DX, word ptr [DI]

OPIS: Inštrukcia vykoná odčítanie dvoch operandov podľa vzorca: operand1 = (operand1 – operand2 – Carry).

❑ **SCAS (Scan string)**

Príznaky								
O	D	I	T	S	Z	A	P	C
*	-	-	-	*	*	*	*	*

PRÍKLADY:
SCAS operand1
SCASB
SCASW

OPIS: Inštrukcia sa používa na vyhľadávanie bajtu, slova uloženého v registri AL alebo AX v bloku dát uloženom v pamäti na adrese určenej dvojicou registrov ES:DI. Po porovnaní obsahu pamäte s príslušným registrom sa obsah registra DI zväčší, resp. zmenší (podľa príznaku Direction flag) o 1 alebo 2 v závislosti od toho, aké veľké dáta sa vyhľadávajú.
Vyhľadávanie prebieha:
po 8 bitoch SCASB DI, SI <- DI, SI +/- 1,
po 16 bitoch SCASW DI, SI <- DI, SI +/- 2,

Maximálny počet vyhľadávaných položiek určuje register CX. Ak použijeme pred inštrukciou prefix REPE (opakuj, pokiaľ je rovné), porovnávanie sa ukončí buď pri nájdení prvého rozdielu medzi blokom dát a registrom AL, prípadne AX, alebo po porovnaní takého množstva položiek, aké udáva register CX. Môžeme pred inštrukciou použiť aj prefix REPNE (opakuj, pokiaľ je rôzne), v tom prípade sa porovnávanie ukončí buď pri nájdení prvej zhody s registrom AL, AX s blokom dát, alebo po porovnaní takého množstva položiek, aké udáva register CX. Ak nepoužijeme nijaký prefix, uskutoční sa porovnanie iba prvej položky bloku dát s registrom AL, AX bez ohľadu na výsledok porovnania.

❑ **SHL** ***(Shift logical left)***

Príznaky								
O	D	I	T	S	Z	A	P	C
*	-	-	-	-	-	-	-	*

PRÍKLADY:
SHL operand1, operand2
SHL AL, 1
SHL AX, CH
SHL word ptr [DI], 1

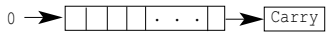
OPIS: Inštrukcia vykoná posuv operandu1 doľava o toľko bitov, koľko určuje operand2. Obrázok znázorňuje prácu inštrukcie:

❑ **SHR** ***(Shift logical right)***

Príznaky								
O	D	I	T	S	Z	A	P	C
*	-	-	-	-	-	-	-	*

PRÍKLADY:
SHR operand1, operand2
SHR AL, 1
SHR AX, CX
SHR DX, word ptr [DI]

OPIS: Inštrukcia vykoná posuv operandu1 doprava o toľko bitov, koľko určuje operand2. Obrázok znázorňuje prácu inštrukcie:



❑ **STC** ***(Set carry flag)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	1

PRÍKLADY: STC

OPIS: Inštrukcia nastaví príznak Carry na 1.

❑ **STD** ***(Set direction flag)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	1	-	-	-	-	-	-	-

PRÍKLADY: STD

OPIS: Inštrukcia nastaví príznak Direction na 1.

❑ **STI** ***(Set interrupt flag)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	1	-	-	-	-	-	-

PRÍKLADY: STI

OPIS: Inštrukcia nastaví príznak Interrupt na 1, a tým povolí prerušenie.

❑ **STOS** ***(Store string data)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

PRÍKLADY:
STOS operand1
STOSB
STOSW

OPIS: Inštrukcia sa používa na uloženie registra AL, AX do pamäte na adresu určenú dvojicou registrov ES:DI. Po načítaní položky sa k registru DI pripočíta, resp. odpočíta (podľa príznaku Direction flag) dĺžka ukladanej položky v bajtoch. Ak použijeme pred inštrukciou prefix REP (opakuj, pokiaľ je CX>0), ukladanie sa ukončí po uložení takého množstva položiek, aké udáva register CX. Ak nepoužijeme nijaký prefix, uskutoční sa uloženie iba prvej položky bez ohľadu na register CX.

❑ **SUB** ***(Integer subtraction)***

Príznaky								
O	D	I	T	S	Z	A	P	C
*	-	-	-	*	*	*	*	*

PRÍKLADY: SUB operand1, operand2
SUB AL, CL
SUB AX, DX

OPIS: Inštrukcia vykoná odčítanie dvoch operandov podľa vzorca:
operand1 = (operand1 – operand2)

❑ **TEST** ***(Logical compare)***

Príznaky								
O	D	I	T	S	Z	A	P	C
0	-	-	-	*	*	?	*	0

PRÍKLADY:
TEST operand1, operand2
TEST AL, BL
TEST AX, CX
TEST DX, 1234h

OPIS: Inštrukcia nastaví príznaky tak, ako keby sa vykonala operácia AND medzi operandmi 1 a 2. Na rozdiel od inštrukcie AND sa operandy nemenia.

❑ **XCHG** ***(Exchange memory/register with register)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

PRÍKLADY:
XCHG operand1, operand2
XCHG AX, BX
XCHG CX, DX
XCHG AL, BL

OPIS: Inštrukcia vymení obsahy operandov 1 a 2.

❑ **XLAT** ***(Table look up translation)***

Príznaky								
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

PRÍKLADY:
XLAT
XLATB

OPIS: Inštrukcia prevedie bajt uložený v registri AL na iný bajt pomocou prevodnej tabuľky, ktorú musíme vytvoriť ešte pred použitím inštrukcie XLAT. Začiatok tabuľky je daný offsetom v registri BX. Tabuľka obsahuje najviac 256 bajtov. Inštrukcia vracia tú hodnotu z tabuľky, ktorej index je uložený v registri AL. Zodpovedajúci bajt k indexu vracia opäť v registri AL.

❑ **XOR** ***(Logical exclusive OR)***

Príznaky								
O	D	I	T	S	Z	A	P	C
0	-	-	-	*	*	?	*	0

PRÍKLADY:
XOR operand1, operand2
XOR AL, 0Fh
XOR AL, AL
XOR BX, AX
XOR AX, 1234h

OPIS: Inštrukcia vykoná logickú operáciu XOR. Pri tejto operácii si všimnite to, že ak ju vykonáte dvakrát s tým istým číslom, dostanete rovnaký výsledok ako na začiatku.

```
        11110001          10111011
XOR     10110101    XOR   10110101
        10111011          11110001
```

A konečne je tu koniec. Dostali sme sa až na koniec inštrukčného súboru procesora 8086. Ak som niektoré inštrukcie vynechal, tak len preto, že sa používajú veľmi zriedka. Inštrukčný súbor nie je ani zďaleka kompletný. S rozvojom počítačov (286, 386, 486, PENTIUM, CYRIX) sa inštrukčný súbor rozširoval a stále sa rozširuje. Nám však budú postačovať inštrukcie, ktoré som uviedol. Ak by predsa len bolo potrebné niekde v programe použiť inštrukciu, ktorú som nespomenul, vysvetlím ju dodatočne a uvediem aj typ procesora, od ktorého ju možno použiť. Na dnes končíme a nabudúce si povieme niečo o syntaxi zápisu inštrukcie, spomenieme pamäťové modely, vysvetlíme si rozdiel medzi inštrukciou a direktívou a možno sa už dostaneme aj ku konkrétnym príkladom v assembleri.

Štvrtá časť: Základy programovania

Mesiac už opäť uplynul a je tu ďalšia časť seriálu o assembleri. Máme si toho ešte veľmi veľa vysvetliť, takže pustime sa do toho.

Základy programovania
Program v assembleri píšeme vždy do súboru s príponou *.ASM. Na písanie môžete použiť ľubovoľný textový editor, ktorý nevkladá do výsledného textového súboru špeciálne kódy. Najčastejšie sa používa editor programu Norton Commander, prípadne môžete použiť prostredie niektorého z vyšších programovacích jazykov, napr. Pascalu, jazyka C a podobne.Program v assembleri je tvorený dvoma druhmi kľúčových slov. Sú to direktívy a inštrukcie. Direktívy nevytvárajú nijaký kód programu, ale používajú sa na riadenie činnosti prekladača. Naopak, inštrukcie vytvárajú kód programu a zapisujeme ich symbolickým zápisom (napr. MOV AX, 10H).

Prekladač assemblera dovoľuje zapisovať program v dvoch režimoch - MASM a IDEAL. Režim MASM je štandardný režim prekladača, to znamená, že nie je potrebné ho v zdrojovom texte uvádzať. Režim IDEAL sa od MASM odlišuje rozdielnym zápisom procedúr, direktív, štruktúr, segmentov atď. V našich príkladoch budem preferovať režim MASM, pretože je implicitne nastavený. Medzi režimami sa môžeme prepínať pomocou direktív **MASM** a **IDEAL**. Stačí ich uviesť na začiatku riadka v zdrojovom súbore. Ďalej budem uvádzať syntax direktív iba v režime **MASM**. Syntax zápisu inštrukcie v assembleri má nasledujúci tvar:

```
Návestie inštrukcia cieľ, zdroj, poznámka
```

Návestie je symbolické meno, ktoré označuje offset adresy v danom segmente. Návestia delíme na blízke (NEAR) a vzdialené (FAR). NEAR návestie predstavuje iba offsetovú časť adresy. Použijeme ho v prípade presunu v rámci segmentu (je to segment, v ktorom je návestie definované). NEAR návestie sa oddeľuje od inštrukcie dvojbodkou (pozri príklad).
Príklad:

```
          ...
          MOV CX,10
CYKLUS:   INC AX
          CMP AX,25h
          JZ EXIT
          LOOP CYKLUS
EXIT:     ...
```

FAR návestie predstavuje kompletnú adresu – segment:offset. Na definovanie FAR návestia sa musí použiť direktíva LABEL.

LABEL – direktíva definuje symbol daného typu. Typ môže byť jeden z nasledujúcich: **NEAR, FAR, PROC, UNKNOWN, BYTE, WORD, DWORD, FWORD, PWORD, QWORD, TBYTE, CODEPTR, DATAPTR.**
Syntax: **[meno] LABEL [typ]**

Príklad:

```
 LABEL BYTE
FAT
          DW ?
          ...
          MOV FAT,AL
          ...
```

LABEL sprístupní pamäťové miesto, ktoré nasleduje po definícii pod zadaným **menom** a **typom**. Ešte pár viet k návestiam. Na označovanie premenných, návestí, konštánt sa používajú symbolické mená, ktoré sa nazývajú identifikátory. Identifikátorom sa označuje v assembleri slovo zložené zo znakov „A..Z“, „a..z“, „0..9“. Identifikátor sa nesmie začínať číslicou, nesmie mať názov rezervovaných slov (príkazov, direktív, inštrukcií...). Malé a veľké písmená sa v assembleri štandardne nerozoznávajú, ale ak je to potrebné, možno túto voľbu aktivovať nastavením prekladača. O tom, ako to treba urobiť, si povieme pri opise parametrov prekladača.

Cieľ a zdroj sú operandy inštrukcie. Niektoré inštrukcie majú iba jeden operand (napr. INT 21H) alebo nemajú nijaký operand (NOP), prípadne majú dva operandy (CMP AX,BX). Ako sme si už vysvetlili operandom môže byť register, pamäťové miesto alebo konštanta. Poznámka je ľubovoľný text, ktorý je od inštrukcie oddelený bodkočiarkou. Prekladač ignoruje všetko od bodkočiarky až po koniec riadka. Platí, že čím viac komentárov program obsahuje, tým je zrozumiteľnejší a prístupnejší iným. Ak teda budete písať assemblerovské, pascalovské alebo iné programy, vždy si ich podrobne okomentujte. Čas, ktorý strávite pri komentovaní programu, sa vám niekoľkonásobne vráti. Z vlastnej skúsenosti viem, že návrat ku komentovanému programu je oveľa príjemnejší. Dôležité je, že nemusíte stráviť niekoľko hodín nad programom a pokúšať sa pochopiť, ako pracuje.

Vytváranie segmentov
Program v assembleri tvoria segmenty, ktoré obsahujú kód, dáta a zásobník. Môžeme ich v zdrojovom texte definovať pomocou direktívy SEGMENT.

Syntax: **[meno] SEGMENT [umiestnenie] [kombinácia] [veľkosť] [trieda]**
...
[meno] ENDS ;Koniec segmentu

Meno – definuje názov segmentu. **Umiestnenie** – určuje ako bude vyzerať segmentová adresa. Môžete použiť tieto voľby: BYTE (segment sa začína na ľubovoľnej adrese), WORD (segmentová adresa sa končí na adrese deliteľnej 2), DWORD (segmentová adresa sa končí na adrese deliteľnej 4), PARA (segmentová adresa sa končí na adrese deliteľnej 16 – táto voľba je štandardne nastavená prekladačom), PAGE (segmentová adresa sa končí na adrese deliteľnej 256), MEMPAGE (segmentová adresa sa končí na adrese deliteľnej 2048). **Kombinácia** – určuje ako bude segment kombinovaný so segmentmi iných programových modulov. Na určenie kombinácie môžete použiť nasledujúce voľby: PUBLIC (všetky segmenty, ktoré majú rovnaké meno, budú spojené do jedného segmentu), STACK (podobne ako pri voľbe PUBLIC). **Veľkosť** – voľba vyjadruje veľkosť segmentu. Možno ju použiť iba vtedy, ak je povolený procesor 80386 direktívou **.386**. Máme k dispozícii tieto voľby: USE16 (segment môže obsahovať 64 KB kódu alebo dát), USE32 (segment môže obsahovať 4 GB kódu alebo dát). **TRIEDA** – používa sa na identifikáciu segmentu, ktorý sa má umiestniť do rovnakých pamäťových miest ako ostatné segmenty tej istej triedy. Na označenie triedy je možné použiť ľubovoľné slovo, ktoré musí byť umiestnené v apostrofoch alebo v úvodzovkách.

Príklad:
```
code      SEGMENT PUBLIC „CODE"
          ASSUME CS:code, DS:data, SS:zasobnik
          MOV AX,4C00h
          INT 21h
code      ENDS
data      SEGMENT PUBLIC „DATA"
          db 10,20,30
data      ENDS
zasobnik  SEGMENT STACK „STACK"
                  dw 200 DUP (?)
zasobnik  ENDS
          END
```

Segmenty budú v pamäti uložené v takom poradí, ako sú definované v zdrojovom texte. Code segment musí byť definovaný ako prvý, pretože program sa spustí od začiatku. Ak by ste dali prvý zásobníkový alebo dátový segment, procesor by začal spracúvať nezmyselné postupnosti inštrukcií. Na ďalšie operácie so segmentmi môžete použiť tieto direktívy: **GROUP, ASSUME, SEG .**
Syntax: **[meno] GROUP [meno_seg], [meno_seg] ...**

Direktíva GROUP spojí pod jedno meno viac rôznych segmentov. Celková dĺžka všetkých segmentov nesmie prekročiť 64 KB. Veľmi často sa takto spájajú dátové segmenty rôznych mien. ASSUME – direktíva definuje závislosť segmentových registrov od segmentu alebo skupiny GROUP. Tým umožňuje prekladaču vytvoriť prefix pred inštrukciou. Direktívu ASSUME je vhodné použiť za každou inštrukciou, ktorá mení obsah segmentových registrov.
Syntax: **ASSUME [seg_reg]:[meno], [seg_reg]:[meno] ...**

Ako segmentový register môžete použiť CS, DS, SS, ES. Meno môže byť meno segmentu alebo skupiny GROUP.

SEGCS, SEGDS, SEGSS, SEGES – tieto direktívy sa používajú na zmenu segmentovej závislosti. Táto zmena platí vždy pre nasledujúci riadok v zdrojovom súbore (pozri príklad). Posledné dve písmená direktívy označujú, ktorý segmentový register sa použije.

Príklad:
```
 ASSUME CS:code, DS:data
 SEGCS
 MOV [BX],AX
 ...
Inštrukciu MOV [BX],AX prekladač preloží ako MOV CS:[BX],AX.
```

Pamäťový model

Vytváranie segmentov, ktoré som doteraz opisoval, je pomerne zložité. Oveľa jednoduchšie sa to dá urobiť pomocou direktívy .MODEL. Táto direktíva sa používa na definovanie pamäťového modelu a v zdrojovom texte sa uvádza vždy na začiatku. Takisto nám prezrádza, o aký veľký program ide.
Syntax: **.MODEL [pamäťový model], [jazyk]**

Pamäťový model môže byť jeden z nasledujúcich typov: TINY, SMALL, MEDIUM, COMPACT, LARGE, HUGE, THUGE, TPASCAL, FLAT. Nás budú zaujímať predovšetkým prvé dva typy. Ostatné pamäťové modely sa odlišujú počtom segmentov, veľkosťou kódu a dát atď. a nepoužívajú sa tak často.

Model **TINY** – ide o najmenší model z celej skupiny modelov. Dáta a kód programu sú umiestnené v spoločnom segmente, ktorý nesmie presiahnuť veľkosť 64 KB. Pri odkazoch na dáta a kód sa používajú smerníky NEAR. Smerník NEAR je 16-bitový smerník, ktorý udáva adresu (offset) v rámci príslušného segmentu. Pomocou smerníka NEAR je možné adresovať dáta iba v rámci príslušného segmentu, teda do 64 KB. Tento model sa používa hlavne na vytváranie programov typu *.COM. V programoch typu COM nie je potrebné definovať zásobník.

Model **SMALL** – dáta a kód programu sú uložené v samostatných segmentoch, ktoré sa neprekrývajú. K dispozícii máte 64 KB pre kód a 64 KB pre dáta a stack programu. Pri odkazoch na dáta a kód programu sa používajú smerníky NEAR. Tento model sa používa na vytváranie súborov s príponou *.EXE.

Jazyk – parameter informuje prekladač o konvencii volania, ktorá sa použije pri tvorbe procedúr. Jazyk môže byť jeden z nasledujúcich: **BASIC, C, CPP, FORTRAN, PASCAL, PROLOG, SYSCALL, NOLANGUAGE**.

Na definovanie jednotlivých segmentov použijeme nasledujúce direktívy:
.CODE – určuje začiatok kódového segmentu,
.DATA – určuje začiatok dátového segmentu,
.STACK – určuje začiatok segmentu zásobníka.
Syntax: .STACK [veľkosť]

Parameter veľkosť určuje počet bajtov, ktoré budú vyhradené pre zásobník. Ak nie je tento parameter uvedený, je vyhradená štandardná veľkosť 1024 bajtov. Nasledujúci príklad ukazuje použitie predchádzajúcich direktív.

```
.MODEL SMALL
.STACK 100H
.DATA
TEXT      DB 'PC-REVUE$'
POLE      DB 10 DUP (?)
FARBA     DW 7000
.CODE
START:    MOV AX,4C00H
           INT 21H
           END START
```

Všimnite si, že žiadna z direktív .CODE, .STACK, .DATA nemá ukončenie segmentu ENDS, ako je to pri direktíve SEGMENT. V tomto prípade sa segment končí na tom mieste, kde sa začína nový segment a posledný segment je zakončený direktívou END.

Zápis dát v assembleri

Na zápis textov, premenných, znakov, číselných hodnôt sa v dátovom segmente používajú tieto direktívy: **DB, DW, DD, DF, DP, DQ, DT**.
Syntax: **[meno] direktíva [výraz],[výraz]...**

Meno je identifikátor, ktorý označuje pamäťové miesto. Meno okrem adresy obsahuje aj informáciu o veľkosti dát. Túto veľkosť určuje **Direktíva** (pozri tabuľku). **Výraz** slúži na definovanie obsahu pamäťového miesta.

Tabuľka:

DIREKTÍVA	VEĽKOSŤ (v bajtoch)	ROZSAH
DB	1	-128 až 255
DW	2	-32768 až 65535
DD	4	-2 146 483 648 až 4 294 967 295
DF	6	-2^{47} až 2^{48-1}
DP	6	-2^{47} až 2^{48-1}
DQ	8	-2^{63} až 2^{64-1}
DT	10	-2^{79} až 2^{80-1}

Direktíva **DB** sa vyznačuje tým, že ako výraz môžete použiť aj ľubovoľne dlhý textový reťazec, ktorý musí byť uzavretý v apostrofoch alebo v úvodzovkách. Po preklade prekladačom sa jednotlivé znaky nahradia číselnými hodnotami ASCII, pričom pre každý znak sa alokuje miesto s veľkosťou 1 bajt. **DW** – výraz je 16-bitová hodnota alebo reťazec dlhý max. 2 znaky. **DD** – výraz je 32-bitová hodnota alebo reťazec dlhý max. 4 znaky. **DF** a **DP** sa používajú na definovanie 48-bitovej FAR adresy. Sú využiteľné až od procesora 386. Výraz môže byť aj reťazec, dlhý max. 6 znakov. **DQ** – direktíva sa používa pre 64-bitové reálne čísla, prípadne pre reťazec dlhý max. 8 znakov. **DT** – sa používa pre reálne čísla. Výraz môže byť aj reťazec dlhý max. 10 znakov. V bežnej programátorskej praxi sa najčastejšie stretnete s direktívami **DB, DW** a **DD .** V nasledujúcom príklade si ukážeme, ako treba definovať tieto direktívy v dátovom segmente.

Príklad:
```
.DATA
;Začiatok dátového segmentu.
TEXT              DB 'Hello, World ! '
N         DB „Computer"
Á         DB 100,99,88,-30
V         DB 10 DUP ('PC-REVUE')
E         DW 16384,65535
S         DD 0,0,0,0
T         DF '123456'
I         DP '123456'
E         DB 10,10,"COFAX'97"
PI        DQ 3.1415927
EX1       DT 2.718281828
```

Doposiaľ sme sa zaoberali len jednoduchými dátami. Prekladač assemblera však dovoľuje používať aj zložené dáta. Sem patria napr.: štruktúry, uniony, bitové polia a tabuľky. Medzi najčastejšie používané zložené dáta patria štruktúry (rozoberieme si ich podrobnejšie).

Štruktúra je zložená z jedného alebo niekoľkých prvkov. Definovanie štruktúry prebieha v dvoch fázach. V prvej fáze sa nadefinuje šablóna štruktúry, resp. jej jednotlivé prvky. V druhej dochádza k definovaniu premennej typu štruktúra, resp. k inicializácii jednotlivých prvkov štruktúry.

Syntax:

[meno] STRUC
...
prvky štruktúry
...
[meno] ENDS

Identifikátor **[meno]** je názov nového dátového typu, ktorý je možné používať pri definovaní premenných rovnako ako direktívy DB, DW, DD a pod. **Prvky štruktúry** predstavujú jednoduché alebo zložené typy dát. Jednotlivé prvky štruktúry sa do pamäte ukladajú v takom poradí, v akom sú definované v danej štruktúre. Syntax inicializácie prvkov štruktúry je nasledovná:

[meno_prem] MENO ŠTRUKTÚRY [inicializácia],[inicializácia],...

Identifikátor [meno_prem] je meno premennej typu štruktúra. MENO ŠTRUKTÚRY je meno, ktorým sme pomenovali našu štruktúru. Na inicializáciu štruktúry je možné použiť tieto hodnoty:

1. **?** (otáznik znamená nedefinovaný obsah)
2. **<>** (hranaté zátvorky znamenajú, že sa použije obsah, ktorý bol nadefinovaný pri vytvorení štruktúry)
3. **<hodnota,hodnota,...>** (takto je možné nadefinovať štruktúru novými hodnotami). Inicializačné hodnoty sú oddelené čiarkami. V prípade, že si prajete nechať nejaký prvok štruktúry implicitne nastavený, stačí, ak namiesto hodnoty prvku vložíte iba čiarku. Príklad, ktorý si ukážeme, vám všetko ozrejmí.
4. **{prvok1=hodnota,prvok2=hodnota,...}** V zložených zátvorkách sa uvádzajú len tie prvky, ktoré chceme naplniť, na poradí prvkov nezáleží.
5. počet **DUP** (hodnota)

Prístup k jednotlivým prvkom štruktúry je zabezpečený pomocou znaku '.' (bodka).

Syntax: **meno_prem.meno_prvku**

Dosť bolo teórie, lepšie to pochopíte na príklade.

Príklad:

```
 .DATA
LIST STRUC
SEFREDAKTOR DB 9 DUP (?)
NAZOV_CASOPISU DB 8 DUP (?)
VYDAVATELSTVO DB 'PERFEKT'
ROK DW ?
CISLO_CASOPISU DB ?
LIST ENDS
;Tu dochádza k inicializácii štruktúry.
K0 LIST ?
K1 LIST <>
K2 LIST <'M. DROBNY','PC-REVUE',,1997,1>
K3 LIST {ROK=2000,CISLO_CASOPISU=12}
 .CODE
;Načítanie prvku zo štruktúry.
 MOV AH,[K2.CISLO_CASOPISU]
;Takto získame adresu začiatku reťazca
;"M. DROBNY" v pamäti.
 MOV SI,OFFSET K2.SEFREDAKTOR
;Uloženie prvku do štruktúry.
 MOV AX,1997
 MOV [K0.ROK],AX
```

Union je zložený dátový typ podobný štruktúre. Platia preň všetky pravidlá, ktoré sme si opísali pri štruktúrach. Šablóna sa definuje medzi kľúčovými slovami **UNION** a **ENDS**. Union sa odlišuje od štruktúr najmä tým, že umožňuje rôzny prístup k jednému pamäťovému miestu. Skladá sa podobne ako štruktúra z viacerých prvkov, ktoré neukladá do pamäte za sebou, ale cez seba. To znamená, že relatívne posunutie každého prvku je nula a celková dĺžka unionu je rovná dĺžke najdlhšieho prvku.

Bitové pole sa napr. od štruktúry výrazne odlišuje v tom, že prvky poľa sú zložené z bitov. Bitové pole sa používa tam, kde chceme skrátiť dĺžku programu a tým šetriť pamäť. Takisto sa môže použiť pre čísla, ktoré sa zmestia do menšieho počtu bitov než osem.

Syntax: **[meno] RECORD prvok1, prvok2...**

Identifikátor [meno] sa používa všade tam, kde chceme vyhradiť pamäť v tomto bitovom formáte. Jednotlivé prvky zapisujeme v tvare:

[meno_prvku]:[veľkosť_prvku]=[inicializácia]

Príklad:

FLAG RECORD A:1=0,B:1=0,C:1,D:1,E:1,F:1

[veľkosť_prvku] je veľkosť v bitoch. [inicializácia] je hodnota bitu alebo celku bitov. Pre **[inicializáciu]** môžete použiť takmer všetko, čo platí pre štruktúry. Celková veľkosť poľa je daná celočíselným násobkom bajtu (pozri tabuľka).

Bajty	Bity
1	0..7
2	8..15
3	16..23
4	24..31

Z tabuľky vyplýva, že celková veľkosť poľa, t.j. súčet veľkostí všetkých prvkov, nesmie prevýšiť 32 bitov. Naše pole FLAG má veľkosť 6 bitov, resp. jeden bajt. Prvky poľa sú v pamäti uložené takto:

7	6	5	4	3	2	1	0
F	E	D	C	B	A	?	?

Nižšie bity bajtu obsadia vždy nedefinované, resp. nepoužité bity (?), ďalšie bity obsadzujú jednotlivé prvky poľa, tak ako boli definované. Pole sa vždy berie ako celok. Ak chceme pracovať s jednotlivými prvkami, treba použiť maskovanie poľa (direktíva MASK) a posuv. Na to môžete použiť napríklad tieto inštrukcie: AND, OR, SHL, SHR a pod.

Príklad:

```
 .model tiny
 .code
org 100h
start:    jmp bit
flag      record A:1,B:1,C:1,D:1,E:1,F:1
znak      flag <0,0,0,0,0,0>
znak1      flag {a=1,f=0}
bit:      mov al,flag <1,1,,,1,1>
          int 20h
          end start
```

Dostávame sa k poslednému dátovému typu. Je ním tabuľka **(TABLE)**. Tento typ predstavuje tabuľku hodnôt (najčastejšie adresy procedúr). Typ **TABLE** bol zavedený hlavne v súvislosti s implementáciou objektov, kde sa používa na reprezentáciu tabuľky metód objektov. Podrobnejšie sa s týmto typom budeme zaoberať neskôr.

Praktické príklady

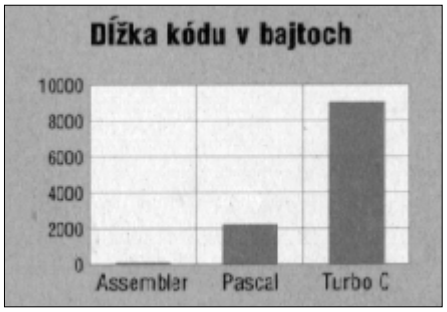
Ako prvý si ukážeme učebnicový príklad pre výpis textu na obrazovku. Program bude typu: *.COM. Všimnite si akú štruktúru (šablónu) má tento typ programu. Vašou úlohou teraz bude opísať tento program a uložiť ho do súboru, napr. prog1.asm.

```
 .MODEL TINY
 .CODE
 ORG 100H
START:    JMP SETTING
TEXT      DB 'HELLO, WORLD ! $'
SETTING:   MOV AH,09H
          MOV DX,OFFSET TEXT
          INT 21H
          INT 20H
          END START
```

Najprv si povieme, ako program pracuje. Na začiatku definujeme model a code segment. Ďalej nasleduje direktíva ORG 100H. Direktíva spôsobí posunutie štartovacej adresy programu o 256 bajtov. Pre programy typu *.COM je nevyhnutná. Pokračujeme definovaním návestia START a inštrukciou skoku JMP.

Táto inštrukcia je potrebná na preskočenie dát, ktoré za ňou nasledujú (tento variant sa veľmi často používa). Všimnite si, že dáta sme definovali v kódovom segmente. Pretože prekladaču to neprekáža, neprekáža to ani nám. Dostávame sa na návestie SETTING, tu sa začína náš program. Služba, ktorú som použil, slúži na vypísanie textového reťazca na obrazovku. Jej číslo je 9 (pozri register AH). Službe tiež musíme nejako odovzdať adresu nášho textu v pamäti. Táto adresa sa odovzdá v registroch DS:DX (segment:offset). Operátorom OFFSET získame offsetovú adresu symbolu. Symbol môže byť návestie, meno dát atď. Reťazec môže obsahovať ľubovoľné znaky okrem znaku '$' (dolár). Tento znak slúži na ukončenie reťazca, ktorý chceme zobraziť. Inak povedané služba vypisuje na obrazovku znaky až kým nenarazí na znak dolár. Takže sme nastavili registre a môžeme pokojne zavolať prerušenie INT 21H. Text sa zobrazí na obrazovku. Teraz musíme program ukončiť tak, aby sa naspäť vrátil do DOS-u. Na to použijeme prerušenie INT 20H. Toto prerušenie sa používa na ukončovanie programov typu *.COM. Podmienkou na použitie tejto služby je to, aby segmentový register CS bol nastavený tak, že začiatok segmentu je totožný so začiatkom PSP1 (Program segment prefix). To je splnené iba pri programoch typu *.COM. Túto službu nie je možné použiť pre programy typu *.EXE. A sme na konci programu, ešte ukončíme kódový segment direktívou END START. A je to.

Ak ste sa dostali až sem, máte už program určite napísaný. Alebo nie? Teraz si vysvetlíme ako program preložiť do strojového kódu. Na preklad budete potrebovať prekladač turbo assembleru (verziu aspoň 3.0 a vyššiu). Program preložíte takto: **TASM meno_suboru** (napr.: TASM.exe prog1.asm). Po úspešnom preklade sa vám na obrazovke zobrazia nasledovné údaje a vytvorí sa súbor s tým istým menom, ale s príponou *.OBJ.



Turbo Assembler Version 3.1 Copyright (c) 1988, 1992 Borland International
Assembling file: prog1.asm
Error messages: None
Warning messages: None
Passes: 1
Remaining memory: 389k

Takže máme vytvorený súbor s príponou *.OBJ. Ako ste si iste všimli, nie je možné tento súbor spustiť. Aby sme ho mohli spustiť, musíme ho „upraviť“ linkerom. **TLINK meno_súboru /t** (napr.: TLINK.exe prog1.obj /t). Parameter /t je potrebný, pretože výsledný program bude typu *.COM. V prípade, že program je typu *.EXE tento parameter sa neuvádza. Ak bola štruktúra programu správna, zobrazí sa nasledovná správa.

Turbo Link Version 3.01 Copyright (c) 1987, 1990 Borland International

V opačnom prípade sa linkovanie zastaví a zobrazí sa chybové hlásenie. Po zlinkovaní programu sa okrem spustiteľného *.COM, prípadne *.EXE vytvorí aj súbor s príponou *.MAP. Súbor obsahuje tieto informácie.

Start Stop Length Name Class
0000H 011DH 011EH _TEXT CODE
011EH 011EH 0000H _DATA DATA

V prvej časti seriálu som sa zmienil o tom, že programy v assembleri zaberajú na disku, prípadne v pamäti málo miesta. Program, ktorý ste si mali možnosť vyskúšať, som naprogramoval v jazyku Pascal, v jazyku C a v assembleri. Z grafu vidieť, aký veľký program vznikne, ak použijeme vyššie programovacie jazyky.

Na záver si uvedieme ten istý program, ibaže bude mať šablónu pre program typu *.EXE. Program preložíte takto: TASM.EXE prog2.asm a TLINK.EXE prog2.obj.

```
.MODEL SMALL
.STACK 100H
.DATA
TEXT      DB 'HELLO, WORLD ! $'
.CODE
START:    MOV AX,@DATA
          MOV DS,AX
          MOV AH,09H
          MOV DX,OFFSET TEXT
          INT 21H
          MOV AX,4C00H
          INT 21H
          END START
```

Program sa odlišuje od predchádzajúceho hlavne v tom, že už musíme definovať zásobník, ako aj v ukončení programu. Segmentový register DS naplníme adresou nášho textu, ktorú získame pomocou konštanty @DATA2. V tomto prípade program ukončíme tak, že register (AH=4CH) bude obsahovať číslo služby, register (AL=00H) bude obsahovať kód chyby (00 - OK). A opäť zavoláme prerušenie INT 21H. Služba 4CH slúži na ukončenie programu typu *.EXE.

Nabudúce si povieme niečo o definovaní konštánt, budeme pokračovať vo vysvetľovaní direktív, operátorov a uvedieme si nejaký príklad.

Piata časť: Direktívy, konštanty

■ **COMMENT** – direktíva sa používa na označenie bloku textu. Takto označený text bude prekladač považovať za komentár.
Syntax: **COMMENT [znak] [text] [znak]**
Prekladač ignoruje text medzi dvoma rovnakými znakmi.
Príklad:
COMMENT /Toto je komentár a prekladač ho ignoruje./

■ **DISPLAY** – pri preklade zobrazí text na štandardné výstupné zariadenie (obrazovku, tlačiareň, prípadne do súboru). Text musí byť ohraničený úvodzovkami.
Syntax: **DISPLAY „text“**
Príklad:
DISPLAY „Wait please“

■ **DOSSEG** – ak použijeme túto direktívu vo svojom programe, budú segmenty zoradené tvz. dosovsky. Direktíva sa zvyčajne používa v spojení s direktívou .MODEL. DOSSEG „oznamuje“ linkeru, že zoradenie segmentov je analogické so zoradením segmentov vyšších programovacích jazykov. Direktíva sa používa hlavne v čisto assemblerských programoch. Ak je assembler pripájaný k vyššiemu programovaciemu jazyku, stará sa sám o zoradenie segmentov.

■ **GLOBAL** – direktíva nahrádza direktívy PUBLIC a EXTRN.
Syntax: **GLOBAL [definícia], [definícia],...**

Ak je daný symbol v zdrojovom texte nadefinovaný, potom je určený ako PUBLIC, inak ako EXTRN. Túto direktívu tiež použijeme na spojenie symbolov v súboroch pripojených direktívou INCLUDE.

■ **INCLUDE** – vloží zdrojový text (meno_súboru) do práve prekladaného textu a preloží ho ako jeho súčasť (podobne ako vo vyšších programovacích jazykoch).
Syntax: **INCLUDE meno_súboru**

Vloženie prebieha podľa nasledujúceho algoritmu: Ak prekladač nájde pri preklade direktívu INCLUDE, preruší preklad, preloží vložený súbor a potom pokračuje v preklade pôvodného súboru. Direktívy **INCLUDE** môžu byť vnorené i do súborov už vložených a jediným obmedzením ich počtu je maximálna dĺžka zdrojového textu, ktorý je možné použiť. Ak súbor nemá uvedenú príponu, je implicitne zvolená prípona *.ASM.
Príklad:
INCLUDE prog1.ASM

Vkladané súbory sú hľadané v aktuálnom adresári, ako aj v adresároch nastavených pomocou voľby /i v príkazovom riadku. Parameter /i nastavuje cestu pre vkladané súbory. V prípade použitia direktívy INCLUDE bude prehľadaný aktuálny adresár a adresáre uvedené v príkazovom riadku s týmto parametrom. Parameter /i môžete použiť viackrát.
Príklad:
TASM /i\prog /ic:\asm\pokus /id:\asm\zdroje

■ **INCLUDELIB** – direktíva vloží do súboru s príponou *.OBJ informáciu pre linker, aby externé funkcie hľadal okrem knižníc uvedených v príkazovom riadku aj v knižnici meno_súboru.
Syntax: **INCLUDELIB meno_súboru**
Pre názov súboru je použitá syntax MS DOS, a pokiaľ nie je uvedená prípona súboru, implicitne sa zvolí prípona LIB.
Príklad:
INCLUDELIB graphics.LIB.

■ **JUMPS** – Často sa pri používaní podmienených skokov dostanú ciele mimo hranice, ktorú procesor umožňuje spracovať (-128 až +127 bajtov od výskytu skoku). Prekladač ohlási chybu (Relative jump out of range by # bytes) a je potrebné jeden podmienený skok nahradiť dvojicou skok podmienený a skok nepodmienený. Ak použijete direktívu **JUMPS**, potom túto náhradu vykoná prekladač automaticky. Pôvodná podmienka skoku je negovaná a za negovanou podmienkou nasleduje nepodmienený skok. Prekladač sám rozhodne, či inštrukciu podmieneného skoku ponechá (veľkosť kódu 2 bajty) alebo nahradí (veľkosť kódu 5 bajtov). Turbo assembler vykonáva štandardne preklad iba na jeden prechod. Toto spôsobuje, že rozhodnutie môže vykonať iba pri skoku na návestie definované skôr než použitý skok. Ak použijeme odkaz dopredu, prekladač je nútený automaticky zvoliť dlhší variant kódu. Ak prekladač neskôr zistí, že skok je mimo hranice -128 až +127 bajtov, dvojicu skokov nahradí postupnosťou týchto inštrukcií: pôvodný podmienený skok, 3-krát inštrukcia NOP. Tým sa program nielen zväčší, ale sa jeho činnosť i čiastočne spomalí. Brániť sa môžeme dvoma spôsobmi: Použijeme pri odkaze dopredu direktívu **NOJUMPS**, ktorá ruší platnosť direktívy **JUMPS**. Direktíva **NOJUMPS** je implicitne určená pri spustení prekladu. Druhá možnosť je v niektorých prípadoch výhodnejšia – povolíme prekladaču dvojprechodový preklad. Ten je možné nastaviť pomocou parametra /m v príkazovom riadku. Bez použitia voľby /m pracuje turbo assembler ako jednoprechodový prekladač. Použitím parametra (/m) bez ďalšieho spresnenia sa počet prechodov nastaví na päť. Inak je určený číslom nasledujúcim za parametrom /m (napr. TASM prog1.ASM /m4).

■ **MULTERRS** – prekladač bežne informuje programátora iba o prvej chybe, ktorú na jednom riadku nájde. V prípade, že použijeme direktívu **MULTERRS**, vypíše prekladač všetky chyby, ktoré sa na riadku vyskytujú. Zrýchli sa tým ladenie programov. Direktívu **MULTERRS** možno zrušiť direktívou **NOMULTERRS**.
Príklad:
MULTERRS
MOV AL,10000H Do reg. AL číslo 10000H
Error Constant too large
Error Extra characters on line

Dve chyby na jednom riadku. Prvá chyba je, že číslo je veľké a nezmestí sa do 8-bitového registra. Druhá je taká, že komentár za inštrukciou nie je oddelený bodkočiarkou.

■ **ORG** – nastaví pozíciu počítadla v aktuálnom segmente.
Syntax: ORG výraz

Pokiaľ **výraz** je meno symbolu, potom symbol musí byť už nadefinovaný. Je možné použiť i preddefinovanú konštantu $, prípadne inú známu konštantu.
Príklad:
ORG 100H
.CODE
MOV AX, 1000
ADD AX, AX

■ **SMART** – povoľuje preklad s automatickou optimalizáciou kódu. Direktíva **SMART** je povolená implicitne. **SMART** nahrádza nasledujúce inštrukcie takto: Ďaleké skoky (JMP FAR) v rámci jedného segmentu upraví na blízke (JMP NEAR) alebo krátke (JMP SHORT). Ďaleké volania CALL v rámci jedného segmentu nahradí pomocou PUSH CS a blízkeho volania CALL. Inštrukcia LEA je nahradená zodpovedajúcou inštrukciou MOV. Pri inštrukciách AND, OR, XOR a TEST je operácia so slovami nahradená zodpovedajúcou operáciou

s bajtmi (iba ak je to možné). **SMART** umožňuje aj použitie inštrukcií, ktoré procesor priamo nepozná. Sú to nasledujúce inštrukcie:
PUSH konštanta (procesor 8086)
PUSH a POP DWORD (procesor 80286)
PUSH a POP PWORD (procesor 80386 a 80486)

Tieto inštrukcie sa vo výslednom programe nahradia postupnosťou inštrukcií. Platnosť direktívy **SMART** možno zrušiť direktívou **NOSMART**.

Definovanie konštánt

Konštanty v assembleri definujeme pomocou direktívy EQU. Direktíva sa používa na definovanie výrazu, reťazca alebo typu alias.
Syntax: **[meno] EQU [hodnota] alebo [meno] = [hodnota]**
1.[hodnota] = výraz
Príklad:

```
F1=3BH
F2=3FH;
ekvivalentný zápis je
F1 EQU 3BH
F2 EQU 3FH
KB1 EQU 102
KB2 EQU KB1 + KB1
NULA EQU KB1 – 1024
```

2. [hodnota] = reťazec. Ak treba v texte použiť reťazec, ktorý je zároveň menom inej konštanty, a nechceme, aby sa nám táto konštanta za reťazec nahradila, použijeme znaky '<' a '>'. Potom prekladač už tento text nenahrádza.
Príklad:

```
TEXT EQU 'ASSEMBLER'
TXT DB TEXT,'JE SUPER.'
KB1 EQU 1024
NULL EQU <KB1-1024>; ekvivalentný zápis je
;NULL EQU 'KB1-1024'
```

3. [hodnota] = alias. V tomto prípade ide vlastne o vytvorenie náhradného označenia pre rezervované slová. Typ alias musí byť jedno slovo, t. j. nemožno takto nahrádzať inštrukcie, prípadne bloky inštrukcií. Na náhradu bloku inštrukcií sa dá použiť MACRO.
Príklad:

```
...
PRERUSENIE EQU INT
...
MOV AL, F1
MOV BX, KB2
PRERUSENIE 20H
```

Číselné a textové konštanty

Číselná sústava, ktorá nie je na konci čísla doplnená identifikačným písmenom (d – dekadická, b – binárna, h – hexadecimálna), sa chápe ako implicitná sústava. V assembleri je to desiatková sústava. Toto implicitné nastavenie môžeme zmeniť direktívou **.RADIX**.
Syntax: **.RADIX [hodnota]**

Direktíva určuje základ implicitnej číselnej sústavy. [hodnota] predstavuje číslo, ktoré môže nadobúdať jednu z týchto hodnôt: 2, 8, 10, 16. Direktívu môžeme použiť na ľubovoľnom mieste v zdrojovom texte. RADIX sa nevzťahuje na dátové typy DD, DQ, DP, a DF.
Príklad:

```
mov ax,10 ;10
  .radix 8
mov ax,10 ;8
  .radix 16
mov ax,10 ;16
  .radix 2
mov ax,10 ;2
  .radix 10
mov ax,10 ;10
```

Hodnota uvedená za bodkočiarkou vyjadruje, ako sa zmení číslo 10 po zmene implicitnej sústavy direktívou RADIX.

Ďalšia v poradí je textová konštanta. Skladá sa z jedného alebo viacerých znakov uzatvorených medzi apostrofy alebo úvodzovky. Napr.:

```
TEXT DB 'PC-REVUE'
DB „11/1997“
```

Konštanty definované prekladačom

Tieto konštanty sa používajú presne tak isto, ako konštanty, ktoré si sami nadefinujete. Ich hodnota sa môže v rámci zdrojového textu meniť.
$ – konštanta obsahuje pozíciu v aktuálnom segmente (v tvare SEGMENT:OFFSET). Používa sa na určenie veľkosti bloku dát alebo programu, prípadne pre skok o známy počet bajtov.
Príklad:

```
; Read CMOS
MOV AL,00H ;Nastavenie CMOS
OUT 70H,AL ;adresy 00h (sekundy)
JMP $+2 ;nepatrné zdržanie
IN AL,71H ;načítaj údaj z CMOS
```

Inštrukcia JMP $+2 spôsobí skok na adresu IP+2, kde IP je (INSTRUCTION POINTER) čiže register obsahujúci ofset adresy práve vykonávanej inštrukcie. POZOR! Pri takýchto skokoch musíte poznať dĺžku inštrukcie. Tú môžete zistiť tak, že napríklad súbor prog1.ASM preložíte takto: TASM prog1.ASM /la. la je prepínač, ktorý umožní vytvorenie podrobného súboru s príponou *.LST. V ňom nájdete všetky potrebné informácie.
@CODE – konštanta obsahuje segmentovú adresu kódu programu.
Príklad:

```
.CODE
MOV AX, @CODE
MOV CS, AX
```

@CURSEG – konštanta obsahuje segmentovú adresu aktuálneho segmentu, t. j. segmentu, v ktorom je použitá.
Príklad:

```
.CODE
MOV AX, @CURSEG
MOV CS, AX
```

@DATA – konštanta obsahuje segmentovú adresu dát.
Príklad:

```
.DATA
TXT DB „ASSEMBLER“
.CODE
MOV AX, @DATA
MOV DS, AX
```

@STACK – konštanta obsahuje segmentovú adresu zásobníka, ktorý je určený direktívou .STACK.
@STARTUP – konštanta určuje vstupnú adresu programu. Je nadefinovaná iba vtedy, keď je v zdrojovom texte použitá pseudoinštrukcia .STARTUP.
Príklad:

```
.CODE
...
.STARTUP
...
END @STARTUP
```

V nasledujúcej časti: Pseudoinštrukcie

Šiesta časť: Pseudoinštrukcie

Začneme tým, že si vysvetlíme pojem pseudoinštrukcia (našiel som ho v slovníku cudzích slov). Pseudoinštrukcia – poznámky vyjadrené v programe v tvare inštrukcie, ktorá slúži na orientáciu v ňom alebo určuje parametre jeho prekladu do strojového kódu. K tomuto odbornému vysvetleniu len toľko, že pseudoinštrukcia je vlastne druh makra. Teda ak v zdrojovom texte uviediete napr. inštrukciu RCR AX,4, po preklade ju prekladač nahradí blokom inštrukcií. Pseudoinštrukcie zmenšujú dĺžku zdrojového textu.

PUSH a POP – tieto pseudoinštrukcie nahrádza postupnosť inštrukcií PUSH a POP s jednotlivými registrami, ďalej umožňujú uloženie konštanty na zásobník, uloženie a vybratie ďalekého ukazovateľa.
1. Syntax: **PUSH zoznam_registrov**
POP zoznam_registrov
Príklad: PUSH AX BX CX DX
POP DX CX BX AX
2. Syntax: **PUSH konštanta**

Inštrukčný súbor procesora 80186 a vyšších obsahuje i inštrukciu uloženia konštanty do zásobníka. V inštrukčnom súbore procesora 8086 však táto inštrukcia chýba. Preto prekladač definuje pri inštrukčnom súbore procesora 8086 pseudoinštrukciu PUSH konštanta, ktorá je pri preklade nahradená postupnosťou legálnych inštrukcií, t. j. inštrukcií, ktoré sú pre daný procesor vykonateľné.
3. Syntax: **PUSH ďalekýukazovateľ**
POP ďalekýukazovateľ

Aby mohol programátor uložiť aj vybrať zo zásobníka aj ďaleký ukazovateľ, je možné použiť pseudoinštrukcie PUSH a POP ďalekýukazovateľ. V 16-bitových procesoroch je veľkosť ďalekého ukazovateľa 4 bajty, ale maximálna veľkosť slova, s ktorou pracujú inštrukcie PUSH a POP, sú 2 bajty. Preto je pseudoinštrukcia preložená ako postupnosť dvoch inštrukcií PUSH a POP.

CALL – táto pseudoinštrukcia slúži na jednoduchšie volanie funkcií v assembleri a vo vyšších programovacích jazykoch.
Syntax: **CALL meno_procedúry [jazyk], argument, argument...**

Použitím tejto pseudoinštrukcie sa programátor zbavuje nutnosti poznať volaciu konvenciu jednotlivých jazykov. Jazyk môže byť jeden z nasledujúcich: BASIC, C, CPP, FORTRAN, PASCAL, PROLOG, SYSCALL alebo NOLANGUAGE. Argumenty sú parametre procedúry, ktoré prekladač podľa zvolenej volacej konvencie umiestni do zásobníka.

RETN, RETF, RETCODE – tieto pseudoinštrukcie možno použiť miesto bežnej inštrukcie RET všade tam, kde chceme vyjadriť typ inštrukcie (ďaleký/blízky návrat).

RETN – generuje vždy blízky návrat, RETF – generuje vždy ďaleký návrat, RETCODE – generuje návrat podľa zvoleného pamäťového modelu. NEAR pre modely TINY, SMALL, COMPACT a TPASCAL. FAR pre modely MEDIUM, LARGE, HUGE a THUGE.

RCL, RCR, ROL, ROR, SHL, SHR, SAL, SAR – pseudoinštrukcie sú využiteľné iba pre procesory 8086 a 8088. Počet generovaných inštrukcií sa uvádza priamo konštantou. Procesory 8086 a 8088 však dovoľujú ako konštantu použiť iba hodnotu 1. Pri použití pseudoinštrukcií je možné zvoliť i vyššiu hodnotu ako jedna. Prekladač potom inštrukciu generuje n-krát s konštantou 1.

Príklad:

```
RCR AX,4
;Po preklade
RCR AX,1
RCR AX,1
RCR AX,1
RCR AX,1
```

SETFLAG, MASKFLEG, TESTFLAG, FLIPFLAG – tieto pseudoinštrukcie nahrádzajú bežné logické funkcie a umožňujú výhodnejší preklad do strojového kódu.
SETFLAG=OR, MASKFLAG=AND, TESTFLAG=TEST, FLIPFLAG=XOR.

STARTUPCODE a .STARTUP – direktívy v závislosti od zvoleného pamäťového modelu a modifikátora vytvárajú úvodný kód, ktorý nastavuje segmentové registre DS a SS a offset zásobníka register SP. Zároveň direktíva vytvára návestie @STARTUP, ktoré sa používa v spojení s direktívou END.

Príklad:

```
.MODEL TINY
.CODE
.STARTUP
RCR AX,4
.EXIT
POLE DB 0,0,0,0,0,0,0,0
END @STARTUP
```

EXITCODE a .EXIT – direktívy v závislosti od zvoleného operačného systému, vytvárajú kód, ktorý ukončí program.
Syntax: **EXITCODE [návratový_kód]**
.EXIT [návratový_kód]

Textový režim

Všetky videoadaptéry (CGA, EGA, VGA...) okrem MDA môžu byť naprogramované tak, aby zobrazovali znaky v textovom alebo grafickom režime. Keď na počítači zavádzate systém, BIOS vždy nastaví textový režim (pozri tabuľku 1). Mód nastavený BIOS-om je mód, ktorý po inicializácii využíva operačný systém. Pokiaľ váš program tento mód nezmení, sú všetky výstupy na displej zobrazované v tomto inicializačnom textovom móde.

Z hľadiska programovania, procedúry pre textový režim sú oveľa jednoduchšie než analogické procedúry na grafický režim. ROM BIOS alebo operačný systém vo väčšine prípadov poskytuje programátorovi procedúry pre zobrazenie znakov, ktoré môžu pracovať v ľubovolnom textovom režime.

Procedúry operačného systému pre výstup na obrazovku sú založené na skupine jednoduchých procedúr v ROM BIOS-u, ktoré sú volané cez softvérové prerušenie INT 10h. Použitie týchto procedúr vás zbavuje nutnosti písať si vlastné procedúry na zobrazovanie znakov. Výhoda procedúr BIOS-u na zobrazovanie znakov je v tom, že program, ktorý používa iba tieto procedúry, bude bežať s väčšou pravdepodobnosťou aj na iných videoadaptéroch.

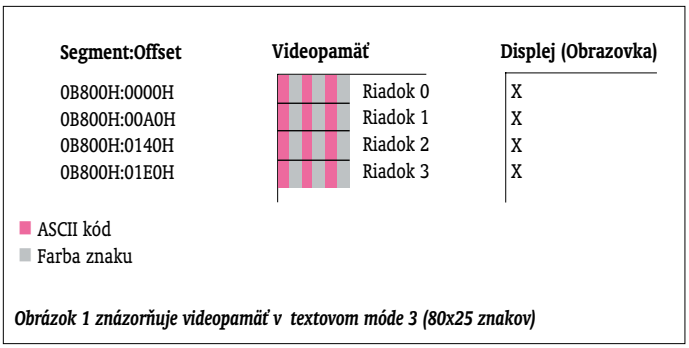
Rýchlosť

Samozrejme, treba zdôrazniť, že výstup cez operačný systém je pomalší v porovnaní s priamym zápisom do videopamäte. Preto by ste si mali vždy uvážiť, čo budete programovať a akú techniku zvolíte, či už to bude priamy zápis do videopamäte alebo služby ROM BIOS-u. Použitím priameho zápisu znížite portabilnosť programu, ale zvýšite jeho rýchlosť. Naopak, použitím služieb BIOS-u znížite síce rýchlosť, ale zvýšite portabilitu programu medzi rôznymi textovými módmi.

Čo sa týka rýchlosti, nedá sa povedať, že textový výstup na obrazovku je pomalý. Ak ho porovnáte s výstupom znakov v grafickom režime, textový výstup je zreteľne rýchlejší, pretože na zobrazenie znakov stačí menej dát vo videopamäti. V textových režimoch sa adaptér sám stará o zobrazenie bodov, ktoré tvoria znak. V grafických režimoch je každý bod znaku reprezentovaný bitom vo videopamäti. Z toho dôvodu je výstup v grafickom režime oveľa náročnejší než zodpovedajúci výstup v textovom režime.

Tabuľka 1: Prehľad textových módov

Mód	Režim	Hustota	Adaptér	Farby	Začiatok videopamäte
00	Text	40x25	CGA, MCGA, EGA, VGA	16 odtieňov sivej	0B800h
01	Text	40x25	CGA, MCGA, EGA, VGA	16	0B800h
02	Text	80x25	CGA, MCGA, EGA, VGA	16 odtieňov sivej	0B800h
03	Text	80x25	CGA, MCGA, EGA, VGA	16	0B800h
07	Text	80x25	MDA, HERC, EGA, VGA	čiernobiele	0B000h



Obrázok 1 znázorňuje videopamäť v textovom móde 3 (80x25 znakov)

Organizácia videopamäte

Každý znak zobrazený na obrazovke je vo videopamäti reprezentovaný dvomi 8-bitovými hodnotami. Znaky vo videoram sú uložené ako lineárna postupnosť, ktorá sa zobrazuje na obrazovke po riadkoch zhora dole (pozri obr. 1).

Generátor znakov videoadaptéra prevádza kód každého znaku do príslušného zobrazenia bodov na obrazovke. V rovnakom čase dekodér atribútov generuje pre každý znak zodpovedajúce atribúty - farbu, intenzitu, blikanie a pod. Pretože každý kód znaku vo videopamäti je uložený spolu s bajtom atribútu, môžete atribúty každého znaku na obrazovke riadiť nezávisle od ostatných znakov. Generátor znakov videoadaptéra zobrazuje textový znak v pravouhlej matici bodov (pozri obr. 2).

	1	2	3	4	5	6	7	8
1	–	–	–	–	–	–	–	–
2	–	–	–	–	–	–	–	–
3	–	X	X	X	X	–	–	–
4	–	–	–	X	X	–	–	–
5	–	–	X	X	X	–	–	–
6	–	–	X	–	X	X	–	–
7	–	X	X	–	X	X	–	–
8	–	X	–	–	X	X	–	–
9	X	X	X	X	X	X	X	–
10	X	–	–	–	–	X	X	–
11	X	–	–	–	–	X	X	–
12	X	X	–	–	X	X	X	–
13	–	–	–	–	–	–	–	–
14	–	–	–	–	–	–	–	–
15	–	–	–	–	–	–	–	–
16	–	–	–	–	–	–	–	–

Obr. 2: Matica 16 x 8 bodov (VGA)

Vzorce a výpočty

Ak chcete zobraziť znak, stačí, ak uložíte jeho ASCII kód a atribút na zodpovedajúcu adresu vo videopamäti. Pretože znaky sú uložené lineárne, môžete ľahko vypočítať adresu vo videopamäti, ktorá zodpovedá pozícii znaku na obrazovke. Vzorec má tvar: **OFFSET=((riadok*80) +stĺpec)*2.**

Príklad:

Výpočet offsetu pre pozíciu 3 (riadok), 0 (stĺpec)
OFFSET=(3*80+0)*2=480d=01E0h

OFFSET je relatívna adresa znaku vo vzťahu k zaeiatku obrazovej stránky. Číslo 80 hovorí, koľko má daný textový režim stĺpcov. Vzorec zahŕňa násobenie dvoma, pretože každý znak vyžaduje dva bajty (ASCII kód a atribút). Hodnoty pre stapec a riadok sa počítajú od nuly, začínajú sa v ľavom hornom rohu obrazovky. Znak umiestnení na obrazovke v ľavom hornom rohu má súradnice [0,0]; znak v pravom dolnom rohu obrazovky má súradnice [24,79].
Dĺžku obrazovej stránky vypočítame zo vzorca:
DĹŽKA_STRÁNKY=2*POČET_RIADKOV*POČET_STĹPCOV

Príklad:

Výpočet dĺžky stránky v režime 80 x 25: dĺžka_stránky=2*25*80=4000 bajtov

Textové režimy pracujú s pamäťou s veľkosťou 32 KB na (EGA, VGA). Táto pamäť je rozdelená do stránok pod3a pamäťových nárokov zvoleného režimu. Pri voľbe stránky určujeme začiatok obrazovej pamäte. Výpočet začiatku pamäte za nás vykoná program BIOS, pri priamom prístupe do pamäte použijeme vzorec:
OFFSET_STRÁNKY=DĹŽKA_STRÁNKY*ČÍSLO_STRÁNKY

Keď porovnáte skutočnú vypočítanú dĺžku stránky (v režime 80 x 25 znakov – 4000 bajtov) a adresy začiatkov jednotlivých stránok, zistíte rozdiel 96 bajtov. Skutočná dĺžka stránky v režime 80 x 25 je 4 000 bajtov a každá ďalšia stránka je posunutá o 4096 bajtov od počiatku stránky, vzniká tu voľný priestor 96 bajtov. Zostávajúcich 96 bajtov obrazovej stránky sa niekedy využíva na uchovanie pozície kurzora, zvyšok sa nevyužíva.

Farba (atribút)

Všetky videosystémy používajú na reprezentáciu textových údajov rovnaké usporiadanie kódu znaku a atribútu, ale bajt atribútu sa interpretuje rôzne. Všeobecne je

Tabuľka 2: Atribút znaku v textovom režime

Bit	Opis
7	blikanie znaku v popredí
6	červená zložka farby pozadia
5	zelená zložka farby pozadia
4	modrá zložka farby pozadia
3	intenzita farby popredia
2	červená zložka farby popredia
1	zelená zložka farby popredia
0	modrá zložka farby popredia

atribút rozdelený na 2 skupiny po 4 bitoch. Nižšia štvorica (bity 0-3) určujú farbu popredia (INK), vyššia štvorica (bity 4-7) určujú pozadie (PAPER) znaku, siedmy bit sa používa aj ako blikanie (BLINK).
Vzorec na výpočet atribútového bajtu je:
ATRIBÚT=(FARBA_POZADIA*16)+FARBA_POPREDIA

Ak chceme, aby znak blikal, jednoducho pripočítame k hodnote ATRIBÚTU číslo 128, čo je vlastne nastavenie 7. bitu (atribútového bajtu) na jednotku.
Vzorec **ATRIBÚT=ATRIBÚT+128**.

Zobrazovací adaptér EGA využíva v 16-farebnom textovom režime rovnaký bajt atribútu ako CGA. Adaptér EGA má v zobrazovacom atribúte zakódovanú voľbu palety farieb. Kódovanie atribútu zobrazenia na EGA, MCGA a VGA závisí aj do použitého monitora. Monitor určený pre adaptér EGA pracuje so šiestimi základnými farebnými signálmi. Tri signály základných farieb (RGB) vyššej intenzity a tri signály (RGB) nižšej intenzity (pozri tabuľku 3). Z týchto šiestich základných signálov je možné vytvoriť 64 kombinácií farieb, ktoré sa odlišujú odtieňom alebo intenzitou. Adaptér VGA emuluje režimy adaptéra EGA vrátane spôsobu práce s registrami palety farieb.

Tabuľka 3: Implicitné nastavenie registrov palety

Farba	I Intensita	R Červená	G Zelená	B Modrá	HEX	DEC
čierna (black)	0	0	0	0	00	00
modrá (blue)	0	0	0	1	01	01
zelená (green)	0	0	1	0	02	02
azúrová (cyan)	0	0	1	1	03	03
červená (red)	0	1	0	0	04	04
fialová (magenta)	0	1	0	1	05	05
hnedá (brown)	0	1	1	0	06	06
biela (white)	0	1	1	1	07	07
sivá (dark grey)	0	0	0	0	08	08
svetlomodrá (light blue)	1	0	0	1	09	09
svetlozelená (light green)	1	0	1	0	0A	10
svetloazúrová (light cyan)	1	0	1	1	0B	11
svetločervená (light red)	1	1	0	0	0C	12
svetlofialová (light magenta)	1	1	0	1	0D	13
svetložltá (yellow)	1	1	1	0	0E	14
jasnobiela (bright white)	1	1	1	1	0F	15

Praktické príklady
V tejto časti si ukážeme niekoľko procedúr na prácu v textovom režime. Vytvorte si súbor PCR11_97.ASM, napríklad programom Norton Commander takto: ncedit.exe pcr11_97.asm. Do vytvoreného súboru vpíšte nasledujúce procedúry.

```
;pcr11_97.asm
;————————————
;Nastavenie polohy kurzora
;na obrazovke.
;Vstup: reg. DL=stlpec,
;reg. DH=riadok.
;————————————
poloha1   proc near
mov bh,0
sub dx,0101h
mov ah,02
int 10h
Ret
poloha1   Endp
;Vypocet adresy vo videopamäti.
;————————————————
;Vstup: AX=riadok, BX=stlpec.
;Vystup: AX obsahuje offset ;adresy.
;————————————————
;Vzorec: [(80*AX)+BX]*2
;————————————————
poloha2   proc near
mov dl,80
mul dl
add ax,bx
mov dx,2
mul dx
Ret
poloha2   Endp
;————————————————
;Procedura vycisti obrazovku
;a nastavi farebny textovy
;mod (cislo 3) 80x25 riadkov
;————————————————
cls1      proc near
mov ax,0003
int 10h
Ret
cls1      Endp
```

```
;————————————————
;Procedura zmaze obrazovku
;Vstup: reg. AL-kod znaku,
;reg. AH-farba reg. ES musi
;obsahovat segment videopamäte,
;t.j. hodnotu 0B800h alebo ;0B000h
;————————————————
cls2      proc near
xor di,di
mov cx,80*25
rep stosw
Ret
cls2      Endp
;————————————————
;Vypis textu na obrazovku
;Vstup: reg. AL=farba textu,
;reg. SI musi obsahovat
;offsetovu adresu pozicie
;vo videopamäti, reg. CX=dlzka
;textu, reg. ES musi obsahovat
;segment videopamäte, reg. DS ;musi
;obsahovat segmentovu adresu ;dat.
;————————————————
vypis     proc near
x10:      Movsb
Stosb
loop x10
Ret
vypis     Endp
;————————————————
;Vypne zobrazovanie kurzora
;v textovom rezime.
;————————————————
kurzor_off          proc near
          mov ah,01h
mov cx,20h*256+00h
int 10h
Ret
kurzor_off          Endp
;————————————————
;Zapne zobrazovanie kurzora
;v textovom rezime.
;————————————————
kurzor_on           proc near
          mov ah,01h
mov cx,12h*256+14h
int 10h
Ret
kurzor_on           Endp
;————————————————
;Nastav rezim 80x43 [ega]
;alebo 80x50 [vga]
;————————————————
rezim     proc near
mov ax,1112h
xor bl,bl
int 10h
mov ah,1
mov cx,0607h
int 10h
Ret
rezim     Endp
;————————————————
```

```
;Procedura sluzi na kreslenie
;ramcekov na obrazovku.
;Vstup:   AH=typ ramceka (1 az 4)
;AL=farba (0 az 255)
;CL=riadok, CH=stlpec
;DL=sirka, DH=vyska
;Maximum window: <0,0> <78,23>
;Plati to pre rezim 80x25 znakov
;————————————————
ramcek    proc near
mov x,cl
mov y,ch
mov lenght,dl
mov height,dh
mov atribut,al
cmp ah,1
jne ram1
mov bp,offset frame1
jmp short ram4
ram1:     cmp ah,2
jne ram2
mov bp,offset frame2
jmp short ram4
ram2:     cmp ah,3
jne ram3
mov bp,offset frame3
jmp short ram4
ram3:mov bp,offset frame4
ram4:call poloha
;Farba.
mov ah,atribut
;Segment adresy vo videopamäti.
          Mov bx,0b800h
mov es,bx
;Offset adresy vo videopamäti.
          mov bx,offadr
xor dl,dl
mov al,[bp]
mov word ptr es:[bx],ax
inc bx
inc bx
ram5:mov al,[bp+1]
mov word ptr es:[bx],ax
inc bx
inc bx
inc dl
cmp dl,length
jl ram5
mov al,[bp+2]
mov word ptr es:[bx],ax
xor ch,ch
mov cl,height
ram6:inc word ptr x
call poloha
mov bx,offadr
mov ah,atribut
xor dl,dl
mov al,[bp+3]
mov word ptr es:[bx],ax
inc bx
inc bx
ram7:mov al,[bp+4]
mov word ptr es:[bx],ax
```

```
inc bx
inc bx
inc dl
cmp dl,length
jl ram7
mov al,[bp+5]
mov word ptr es:[bx],ax
inc bx
inc bx
loop ram6
inc word ptr x
call poloha
mov ah,atribut
mov bx,0b800h
mov es,bx
mov bx,offadr
xor dl,dl
mov al,[bp+6]
mov word ptr es:[bx],ax
inc bx
inc bx
ram8:mov al,[bp+7]
mov word ptr es:[bx],ax
inc bx
inc bx
inc dl
cmp dl,length
jl ram8
mov al,[bp+8]
mov word ptr es:[bx],ax
Ret
;Premenne.
offadr dw ?
x db ?
y db ?
lenght db ?
height db ?
atribut db ?
;Definicie jednotlivych typov
;ramcekov.
frame1 db 213,205,184,179,32
db 179,212,205,190
frame2 db 201,205,187,186,32
db 186,200,205,188
frame3 db 218,196,191,179,32
db 179,192,196,217
frame4 db 214,196,183,186,32
db 186,211,196,189
;Vypocet offsetu videopamäte
;zo suradnic x a y
poloha:   xor ah,ah
xor bh,bh
mov al,x
mov bl,y
mov dl,80
mul dl
add ax,bx
mov dx,2
mul dx
mov word ptr [offadr],ax
Ret
ramcek    Endp
```

I keď procedúry budeme preberať nabudúce, nič nebráni tomu, aby ste ich spoznali už teraz. Ak chcete pochopiť, ako pracujú, musíte si pozorne pozrieť inštrukciu za inštrukciou. Inštrukčný súbor nájdete v PC REVUE č. 9/1997. Všimnite si, že niektoré procedúry sa zaeínajú krátkym opisom. V ňom nájdete vždy potrebné informácie na používanie procedúry, t. j. aké registre naplniť a eím, ak procedúra poskytuje nejaké výstupné hodnoty, tak aké. Čo sa týka biosovských alebo dosovských služieb, odporúčam literatúru číslo [5], prípadne inú podobne zameranú knihu, alebo elektronický help. Bolo by asi vhodné uviesť aj nejaký príklad, ktorý vám viac ozrejmí používanie predchádzajúcich procedúr. Takže, nech sa páči, príklad.

```
;EXAMPLE.ASM
.model tiny
.code
org 100h
zaciatok:
jmp start
include pcr11_97.asm
start:call kurzor_off
mov ax,0B800h
mov es,ax
mov ax,7*256+32
call cls2
mov ax,0B800h
mov es,ax
xor di,di
mov ax,cs
mov ds,ax
mov cx,250
cykl:push cx
mov si,offset znaky
mov cx,8
rep movsw
pop cx
loop cykl
mov ax,12 ;Riadok
mov bx,4 ;Stlpec.
call poloha2
mov di,ax
mov ax,0B800h
mov es,ax
mov ax,cs
mov ds,ax
;Offsetova adresa textu v pamäti
mov si,offset text
mov cx,72 ;Dlzka textu.
Mov al,42 ;Farba.
Call vypis
mov ax,0
int 16h
;————————————————
mov ax,1*256+122
mov cx,0*256+0
mov dx,11*256+38
call ramcek
mov ax,2*256+60
mov cx,40*256+0
mov dx,11*256+38
call ramcek
mov ax,3*256+62
mov cx,0*256+13
mov dx,10*256+38
call ramcek
mov ax,4*256+121
mov cx,40*256+13
mov dx,10*256+38
call ramcek
mov ax,0
int 16h
;————————————————
mov ax,0B800h
mov es,ax
mov ax,7*256+32
call cls2
call rezim
mov dx,10*256+0
call poloha1
mov ah,09h
mov dx,offset text2
int 21h
mov ax,23 ;Riadok.
mov bx,11 ;Stlpec.
call poloha2
mov di,ax
mov ax,0B800h
mov es,ax
mov ax,cs
mov ds,ax
mov si,offset text1
mov cx,57 ;Dlzka textu
mov al,90 ;Farba.
call vypis
mov ax,0
int 16h
;————————————————
call cls1
call kurzor_on
.exit
;Usporiadanie znak, atribut,
;znak, atribut ...
znaky     db 'P',9,'C',10,' ',11
          db 'R',12,'E',13,'V',14
          db 'U',15,'E',8
text      db 'POZOR!!! Po stlaceni'
          db ' lubovolneho klavesu'
          db ' nasleduje vykreslenie'
          db ' ramcekov.'
text1     db 'POZOR!!! Po stlaceni'
          db ' lubovolneho klavesu'
          db ' koniec programu.'
text2     db 13,10,'DVA SPOSOBY, AKO MIAST,'
          db ' TRAPIT ALEBO LEN VYSTRASIT'
          db ' LUDI V POCITACOVEJ UCEBNI.'
          db 13,10,13,10,'TYP 1:',13,10
          db 'Prihlaste sa, pockajte '
          db 'niekolko sekund, potom nasadte'
          db ' vystraseny pohlad a za-'
          db 13,10,'kricte: „Preboha!'
          db ' Nasli ma!" a utecte.'
          db 13,10,13,10,'TYP 2:',13,10
          db 'Prineste si motorovu pilu, ale'
          db ' nepouzite ju. Ak sa vas niek-
to'       db ' opyta, na co to',13,10
          db 'mate, odpovedzte tajomne:'
          db ' „Iba keby nahodou..."$'
          end zaciatok
```

Vysvetlenie

Ešte než zakončíme túto časť, objasníme si nové prvky, ktoré obsahuje predchádzajúci príklad. Program sa začína na návesti štart, najprv odstránime blikajúci kurzor a zmažeme obrazovku. Potom nasleduje časť, ktorá obsahuje cyklus. V assembleri často používaným cyklom je cyklus typu FOR. Tento cyklus sa tiež môže realizovať pomocou podmienených skokov. No v assembleri sa častejšie používa modifikácia tohto cyklu, keď sa na zaeiatku register CX naplní počtom opakovaní cyklu. Po každom prechode cyklom sa hodnota registra CX dekrementuje o 1 a cyklus sa opakuje, pokiaľ je register CX rôzny od nuly. Na vytvorenie tohto cyklu sa používa inštrukcia LOOP návestie.

Na začiatku programu sa reg. ES naplní segmentovou adresou videopamäte. Ďalej vynulujeme offset, t. j. reg. DI a segmentový register DS naplníme adresou, ktorú práve obsahuje segmentový register CS.

Poznámka: Stačí ak segmentové registre (CS, DS, ES ...) nastavíte na začiatku programu. Ak ste si istí, že sa ich obsah nebude počas behu programu meniť, nemusíte ich viac už nastavovať. Skráti sa tým aj výsledný kód. Ostatne, nechám to na vás, experimentujte.

Prácou cyklu je zaplniť obrazovku textom „PC REVUE", pričom každý znak tohto slova bude inej farby. Pokračujeme výpisom textu na pozíciu (12,4), použijeme na to procedúru výpis. Nasledujúcim krokom je čakanie na stlačenie ľubovoľného klávesu (pozri inštrukcie MOV AX,0 a INT 16h). Podrobnejšie sa klávesnicou budeme zaoberať v niektorom ďalšom čísle PC REVUE. Po stlačení klávesu pokračujeme vykreslením štyroch rôznofarebných rámčekov. Túto procedúru určite využijete pri svojich vlastných pokusoch. Posledná časť programu vykoná zmazanie obrazovky a zmenu na režim 80 x 43 riadkov (EGA) alebo 80 x 50 riadkov (VGA). Nasleduje výpis textu a program opäť čaká na stlačenie ľubovoľného klávesu. Program sa končí zmazaním obrazovky, zmenou textového režimu na mód 3 a zapnutím zobrazovania kurzora.

Nabudúce si povieme o ...

Nabudúce si preberieme operátory, vysvetlíme si pojmy procedúra a makro a, samozrejme, zase uvedieme nejaké praktické príklady.

Literatúra

[1] P. Cápek - J. Rojko: Turbo assembler 3.0. Grada 1992.

[2] M. Brandejs: Mikroprocesory Intel 8086-80486. Grada 1991.

[3] Turbo Help, Verzia: 3.0, 1992 Borland International.

[4] Videosystémy v počítačích 1. a 2. část. JZD Agrokombinát Slušovice 1989.

[5] V. Boukal: BIOS IBM PC. Grada 1992.

Siedma časť: Procedúry, makrá a operátory

Procedúry

Už v predošlej časti ste mali možnosť oboznámiť sa s procedúrami, v tejto časti si ich vysvetlíme oveľa podrobnejšie. Ak by som mal definovať pojem procedúra, prvé, čo by ma napadlo by asi bolo, že je to podprogram, t. j. program, ktorý vykonáva nejakú špecifickú činnosť, i keď to nemusí byť pravidlom. Procedúry sa najčastejšie používajú tam, kde sa často opakuje blok programu. Procedúra súvisí s inštrukciou **CALL**, ktorá sa používa na jej volanie. Jednoducho napíšete **CALL meno_procedúry**.

Syntax: **meno PROC [jazyk] [typ]**
[USES items,] [argument [,argument]...]
RETURNS [argument [,argument]...]
Sem treba vložiť blok programu.
RET
meno ENDP

Jazyk môže byť jeden z nasledujúcich variantov: **BASIC, C, CPP, FORTRAN, PASCAL, PROLOG, SYSCALL, NOLANGUAGE. Typ** môže byť **FAR** alebo **NEAR** (podrobnejšie pozri PC REVUE č. 10/1997). **Items** je zoznam registrov a jednoduchých symbolov, ktoré budú v procedúre použité. Celý zoznam je pri vstupe do procedúry uložený do zásobníka a pri ukončení procedúry obnovený, t. j. vybratý zo zásobníka. Jednotlivé prvky zoznamu je potrebné od seba oddeliť medzerou. Kľúčové slovo **RETURNS** určuje jednu, alebo viac položiek, ktoré nebudú pri návrate z procedúry odstránené zo zásobníka.

Časté chyby pri používaní procedúr:

1. Zabúda sa na inštrukciu **RET**.
2. Ukončenie procedúry **ENDP**.

Preto si vždy skontrolujte, či je procedúra ukončená inštrukciou **RET** a následne **ENDP**. V prvom prípade vás ani prekladač na chýbajúcu inštrukciu neupozorní.

V bloku **PROC...ENDP** môžeme používať nasledujúce direktívy:

ARG – direktíva sa používa pri definovaní procedúry a určuje parametre tejto procedúry uložené v zásobníku pred návratovou adresou. Zvýhodňuje prístup k týmto parametrom, ktoré by sa inak museli adresovať ako dáta (pomocou registra BP). POZOR!!! Uloženie jednotlivých argumentov v zásobníku je rôzne, podľa použitého jazyka.

Syntax: **ARG argument [,argument] ...[=size]**
[RETURNS argument [,argument]]

Konštanta size obsahuje celkovú veľkosť dát. **RETURNS** určuje jednu, alebo viac položiek, ktoré nebudú pri návrate z procedúry odstránené zo zásobníka.

LOCAL – direktíva slúži na definovanie lokálnych premenných v procedúre.

Syntax: **LOCAL localdef [,localdef] [=size]**

USES – sa používa v bloku PROC...ENDP a určuje, ktoré dáta a registre budú pri vstupe do procedúry uložené do zásobníka a pri výstupe z procedúry vybraté zo zásobníka.

Syntax: **USES item [,item]**

Príklad:

```
;Takto vyzerá procedúra pred
;aplikovaním direktívy USES
CLS1    PROC C NEAR
        USES AX,BX,CX,DX
        MOV AX,0003
        INT 10H
        RET
CLS1    ENDP
;Po aplikovaní vzniknú
;viditeľné zmeny v procedúre
CLS1    PROC C NEAR
        USES AX,BX,CX,DX
        PUSH AX
        PUSH BX
        PUSH CX
        PUSH DX
        MOV AX,0003
        INT 10H
        POP DX
        POP CX
        POP BX
        POP AX
        RET 00000H
CLS1    ENDP
```

LOCALS, NOLOCALS – táto direktíva povoľuje používanie lokálnych symbolov v bloku **PROC...ENDP**. Každý symbol sa musí začínať dvojznakom. Ak tento dvojznak nebude uvedený, potom sa musí začínať implicitným dvojznakom „@@“. Direktíva **NOLOCALS** zakazuje používanie lokálnych symbolov.

Syntax: **LOCALS [prefix]**

Príklad:

```
        LOCALS AB
VYPIS   PROC NEAR
ABX10:  MOVSB
        STOSB
        LOOP ABX10
        RET
VYPIS   ENDP
```

Výsledkom toho je, že návestie „abx10“ môžete použiť v programe viackrát. Pri použití direktívy **LOCALS** je platnosť návestia len v rámci procedúry. To je príčina, prečo môžete použiť to isté návestie viackrát. Ak by ste však direktívu **LOCALS** nepoužili, prekladač ohlási chybu **Symbol already defined** (Symbol bol už raz definovaný).

Ešte nám zostali dve direktívy. Sú to **EXTRN** a **PUBLIC**. Tieto dve z doposiaľ uvedených sa pravdepodobne používajú (v bežnej praxi) najčastejšie.

PUBLIC – direktíva označuje symbol, ktorý má byť viditeľný z iných modulov. **PUBLIC** sa používa na zverejňovanie mien premenných, konštánt, programových návestí atď.

Jazyk môže byť napr.: **C, PASCAL, BASIC, FORTRAN** alebo **PROLOG .**

Syntax: **PUBLIC [jazyk] symbol [,[jazyk] symbol]...**

EXTRN – direktíva oznamuje prekladaču a linkeru, že daný symbol je definovaný v inom module.

Syntax: **EXTRN definition [,def...]**

Definícia má tento tvar: **[jazyk] name : type [:count]**, kde **jazyk** môže byť napr.: **C, PASCAL, BASIC, FORTRAN** alebo **PROLOG**. Name je symbol, ktorý je definovaný v inom module. Typ môže byť NEAR, FAR, PROC. Typ PROC určuje, že spôsob volania je závislý od použitého pamäťového modelu. V prípade dát ide vo väčšine prípadov o určenie veľkosti. Môžete použiť tieto kľúčové slová **BYTE, CODEPTR, DATAPTR, DWORD, FWORD, PWORD, QWORD, TBYTE, WORD, ABS** alebo **meno štruktúry**. **Count** určuje, koľko položiek (items) má externý symbol, ide o nepovinný parameter.

Makrá

Pojem makro je vám už určite známy. Makro možno definovať ako substitúciu (inak povedané makro pod svojím menom zoskupuje príkazy alebo inštrukcie, ktoré sa majú vykonať). Výhodou je, že napr. často sa opakujúcu časť programu nemusíte stále odpisovať, naopak, nevýhodou je, že čím viac sa makro v zdrojovom texte objaví, tým väčší bude kód programu.

Syntax: **meno MACRO [parameter1 [,parameter2]...]**
;sem treba vložiť inštrukcie,
;ktoré budú tvoriť makro
ENDM

Nasledujúci príklad ukazuje, ako vytvoriť, resp. použiť makro. Pretože v programe sú použité „prvky“, ktoré sme už podrobne prebrali v predchádzajúcich častiach, nebudem program komentovať. Nech sa páči vyskúšať.

Príklad:

```
        .MODEL SMALL
        .STACK  100H
        .DATA
TEXT1   DB 'TOTO JE TEXT 1',13,10,'$'
TEXT2   DB 'TOTO JE TEXT 2',13,10,"$"
VYPIS_SPRAVU    MACRO RETAZEC
                LEA DX,RETAZEC
                MOV AH,09H
                INT 21H
                ENDM
PAUSE   MACRO
        MOV AX,0
        INT 16H
        ENDM
CLS     MACRO
        MOV AX,3
        INT 10H
        ENDM
        .CODE
START:  MOV AX,@DATA
        MOV DS,AX
        CLS
        VYPIS_SPRAVU TEXT1
        PAUSE
        VYPIS_SPRAVU TEXT2
        PAUSE
        CLS
        MOV AX,4C00H
        INT 21H
        END START
```

Turbo assembler má už v sebe definovaných niekoľko makier. Sú to nasledujúce makrá: **CATSTR, INSTR, SIZESTR, SUBSTR, IRP, IRPC, REPT, WHILE, GOTO**. O každom si povieme niekoľko slov. Prvé štyri sa používajú na prácu s reťazcami. Ak máte nižšiu verziu assemblera ako 3.1, treba pred ich použitím uviesť direktívu **MASM51**. Direktíva povoľuje písanie zdrojového textu v rozšírenom režime. Verzia assemblera 3.1 vyhodnocuje makrá (CATSTR, INSTR, SIZESTR, SUBSTR) aj bez použitia direktívy MASM51. Zostávajúce makrá sa používajú na prácu s cyklami.

CATSTR – spojí viac reťazcov do jedného.
Syntax: **name CATSTR string [, string]...**
Príklad: LETTERS CATSTR <abc>,<def>
Po preklade: LETTERS=“abcdef“

INSTR – výstupom makra je pozícia prvého výskytu druhého reťazca v prvom reťazci. Ak druhý reťazec nie je v prvom obsiahnutý, vráti makro nulu.
Syntax: **name INSTR [start,] string1, string2**
Príklad: KOLKO INSTR 1,<PROGRAMUJEME V ASSEMBLERI>,<V>
Po preklade: KOLKO=0EH

SIZESTR – výstupom makra je dĺžka reťazca. Prázdny reťazec má dĺžku nula.
Syntax: **name SIZESTR string**
Príklad: DLZKA SIZESTR <PROGRAMUJEME V ASSEMBLERI>
Po preklade: DLZKA=19H

SUBSTR – definuje nový reťazec ako časť pôvodného reťazca.
Syntax: **name SUBSTR string, position [, size]**
Príklad: NOVY SUBSTR <ABCDEFGH>,3,2
Po preklade: NOVY=“CD“

IRP – opakuje n-krát po sebe text, pričom v ňom zároveň nahrádza parameter postupne argumentmi 1 až N. Text je ukončený direktívou ENDM rovnako ako makro.
Syntax: **IRP parameter, <argument1, argument2,...>**
;TEXT
ENDM
Príklad: IRP REGISTER,<AX,BX,CX,DX>
PUSH REGISTER
ENDM
Po preklade: PUSH AX
PUSH BX
PUSH CX
PUSH DX

IRPC – makro opakuje text s nahradzovaním znaku.
Syntax: **IRPC paremeter, string**
;TEXT
ENDM
Príklad: IRPC ZNAK,0123
DB ZNAK
ENDM
Po preklade: DB 0
DB 1
DB 2
DB 3

REPT – makro kopíruje text toľkokrát, koľkokrát určuje hodnota počet.
Syntax: **REPT počet**
;TEXT
ENDM
Príklad: REPT 3
DEC AX
ENDM
Po preklade: DEC AX
DEC AX
DEC AX

WHILE – prekladač opakuje inštrukcie makra dovtedy, pokým nebude podmienka 0 (FALSE).
Syntax: **WHILE podmienka**
;Telo makra
ENDM

GOTO – makro umožňuje preniesť riadenie na riadok označený návestím. Návestie sa musí začínať znakom dvojbodka „:“, napríklad „:start“.
Syntax: **GOTO návestie**

Špeciálne Macro operátory
Operátor „&“ umožňuje pri rozvinutí makra nahradiť symbolické meno skutočným.
Syntax: **&name**

Pri použití operátora „< >“ sa reťazec chápe ako konštantný výraz, ktorý sa ničím nenahrádza.
Syntax: **<text>**

Operátor „!“ je analogický a operátorom „< >“ s tým rozdielom, že platí iba pre jeden znak (character).
Syntax: **!character**

Operátor „%“ sa používa vtedy, ak chceme, aby sa výraz pred nahradením vyhodnotil.
Syntax: **%text**

Operátor „;;“ spôsobí ignorovanie komentára v bloku makra, t. j. pri rozvíjaní makra sa tento text už neodpisuje.
Syntax: **;;comment**

Operátory
MASM Mode Operator Precedence
<>, (), [], LENGTH, MASK, SIZE, WIDTH
. (výber prvku zo štruktúry)
HIGH, LOW
+, - (unary)
: (segment override)
OFFSET, PTR, SEG, THIS, TYPE
*, /, MOD, SHL, SHR
+, - (binary)
EQ, GE, GT, LE, LT, NE
NOT
AND
OR, XOR
LARGE, SHORT, SMALL, .TYPE

Operátory uvedené v tabuľke sú zoradené podľa priority. Operátory oddelené čiarkou majú rovnakú prioritu. Preberieme si ich v takom poradí, v akom sú uvedené v tabuľke.

< > – operátor je reťazec chápaný ako konštantný výraz, ktorý sa ničím nenahrádza.
Syntax: **<text>**
Príklad: HODNOTA EQU <20+20>

( ) – zmena priority.
Syntax: **(výraz)**
Príklad: (1+2)*(3+4) ;výsledok bude 21
1+2*3+4 ;výsledok bude 11

[ ] – operátor môže:
a) znamenať súčet obidvoch výrazov,
b) tvoriť nepriame pamäťové operandy s registrami BP, BX, DI, a SI (pozri nepriame adresovanie).
Syntax: **[výraz1] [výraz2]**
Príklad: MOV BL,[BX][SI]
MOV AX,[DI]

LENGTH – operátor vráti počet dátových prvkov umiestnených v pamäti pomocou direktív DB, DW, DD, DF, DP, DQ, DT. Pozor!!! Operátor vráti iba počet položiek, ktoré sa majú rezervovať operátorom DUP, inak vráti hodnotu 1.
Syntax: **LENGTH name**
Príklad: LETTERS DB „ABCDEFGHIJK“
X1 = LENGTH LETTERS ;X1=1
ZASOBNIK DB 20 DUP (?)
X2 = LENGTH ZASOBNIK ;X2=20

MASK – vráti bitovú masku pre bitové pole. Record je meno už známeho bitového poľa. Ak napr. použijeme operátor MASK na položku, ktorá predstavuje druhú štvoricu bitov v slove (WORD), potom má hodnota tvar 0000000011110000b. MASK sa veľmi často používa spolu s inštrukciou AND na odmaskovanie poľa, prípadne položky poľa.
Syntax: **MASK record**
MASK record_field_name

SIZE – vráti veľkosť dátových položiek. Name je symbol odkazujúci sa dátovú položku umiestnenú do pamäte pomocou niektorého z dátových typov DB, DW, DD, DF, DP, DQ, DT alebo pomocou zloženého dátového typu.
Syntax: **SIZE name**
Príklad: CISLO DB 10
X3 = SIZE CISLO ;X3=1
REAL DQ 1000,2000
X4 = SIZE REAL ;X4=8

WIDTH – record_field_name je meno existujúceho prvku bitového poľa. Operátor vráti počet bitov, ktoré tento prvok obsadzuje, to isté platí pre celé pole.
Syntax: **WIDTH record**
WIDTH record_field_name
Príklad: FLAG RECORD A:1,B:1,C:1,D:1,E:1,F:1
ZNAK FLAG <>
MOV AL,WIDTH B ;AL <- 1
MOV AL, WIDTH FLAG ;AL <- 6

._ – výber prvku zloženého dátového typu.
Syntax: **meno_prem.meno_prvku**
Príklad: (pozri PC REVUE č. 10/1997, časť Štruktúry)

HIGH – operátor vráti hornú časť výrazu.
Syntax: **HIGH výraz**
Príklad: MOV AH,HIGH 4C00H ;AH <- 4CH

LOW – operátor vráti dolnú časť výrazu.
Syntax: **LOW výraz**
Príklad: MOV AH,LOW 004CH ;AH <- 4CH
+ (unary) –> operátor určuje kladné číslo.
Syntax: **+výraz**
- (unary) –> označenie záporného čísla. Výraz musí byť konštanta.
Syntax: **-výraz**
Príklad: CISLO1 = -100 ;Výsledok bude -10
CISLO2 = -(-100) ;Výsledok bude 10
: (segment override)
meno_seg_registra:výraz – operátor vykoná výpočet adresy relatívne, k špecifikovanému segmentu. **Výraz** môže byť konštanta alebo odkaz do pamäte.
Príklad: MOV DS:[123],AX
MOV AX,ES:[1234]
MOV AX,DS:[DI]
OFFSET – operátor vráti offset z adresy symbolu. Symbol môže byť návestie v programe, meno dát a pod. Offset vráti posunutie od začiatku segmentu do výskytu symbolu.
Syntax: **OFFSET symbol**
Príklad: .DATA
TEXT DB „PC_REVUE“
...
.CODE
MOV AX,OFFSET TEXT
...
PTR – určuje veľkosť ukazovateľa alebo dát, s ktorými sa pracuje a na ktoré ukazuje symbol. Z toho vyplýva, že symbol musí byť adresa. V režime MASM môže typ predstavovať jedno z nasledujúcich čísel.
Pre dáta:

0	UNKNOWN
1	BYTE
2	WORD
4	DWORD
6	PWORD
6	FWORD
8	QWORD
10	TBYTE
2 až 4	DATAPTR
2 až 4	CODEPTR

Pre kód:

0FFFFh	NEAR
0FFFEh	FAR
0FFFFh až 0FFFEh	PROC

Namiesto čísel je možné priamo použiť kľúčové slová. Pri kľúčových slovách **DATAPTR, CODEPTR** a **PROC** sa rozhoduje podľa použitého pamäťového modelu.
Syntax: **typ PTR symbol**
SEG – operátor vráti segmentovú časť adresy symbolu.
Syntax: **SEG symbol**
Príklad:

```
        .MODEL SMALL
        .STACK 100H
        .DATA
CISLO   DB 10
TEXT    DB 'SEGMENT:OFFSET$'
        .CODE
START:  MOV AX,SEG DS
        MOV AX,4C00H
        END START
```

THIS – vytvorí nový operand a jeho adresa je určená aktuálnym segmentom a hodnotou registra IP. Typ určuje veľkosť operandu. Typ môže byť jeden z nasledujúcich: **NEAR, FAR, PROC, UNKNOWN, BYTE, WORD, DWORD, FWORD, PWORD, QWORD, TBYTE, CODEPTR, DATAPTR** alebo **meno** zloženého dátového typu.
Syntax: **THIS typ**
Príklad:
ADRESA DD THIS DWORD
OFSET DW ?
SEGMT DW ?
TYPE – operátor vráti číslo, ktoré vyjadruje veľkosť alebo typ symbolu.

Číslo	**Veľkosť**
1	BYTE
2	WORD
4	DWORD
6	PWORD
6	FWORD
8	QWORD
10	TBYTE

Číslo	**Typ**
0	konštanta
0FFFFh	NEAR
0FFFEh	FAR

Syntax: **TYPE symbol**
Príklad:
CISLO DB 10
CISLO1 DT ?
POLE DW 50 DUP (0)
TYP1 DB TYPE POLE ;Výsledok 2
TYP2 DB TYPE CISLO ;Výsledok 1
TYP3 DB TYPE CISLO1 ;Výsledok 10
* – násobenie dvoch celočíselných výrazov.
Syntax: **výraz1 * výraz2**
Príklad: SCREEN DW 80*25
/ – operátor vykoná delenie dvoch celočíselných konštánt.
Syntax: **výraz1 / výraz2**
Príklad: X DB 123/5 ;Výsledok 24
MOD – operátor vráti zvyšok po delení dvoch celočíselných konštánt.
Syntax: **výraz1 MOD výraz2**
Príklad: X DB 123 MOD 5 ;Výsledok 3
SHL – operátor vykoná logický bitový posun doľava o toľko bitov, koľko určuje parameter počet. Každý posun zodpovedá násobeniu dvomi. Ak je počet záporné číslo, vykoná sa posun doprava.
Syntax: **výraz SHL počet**
Príklad: MOV AH,00001111B SHL 4
;Výsledok bude
MOV AH,11110000B
SHR – podobne ako v predchádzajúcom prípade, iba s tým rozdielom, že SHR vykoná posun doprava. Posun o jeden bit je delenie výrazu dvoma.
Syntax: **výraz SHR počet**
Príklad: mov al,11110000b shr 4
;Výsledok bude
MOV AH,00001111B
+ (binary) – operátor vykoná súčet dvoch výrazov.
Syntax: **výraz1 + výraz2**
Príklad: XY DB 1+1 ;Výsledok 2
- (binary) – operátor vykoná rozdiel dvoch výrazov.
Syntax: **výraz1 - výraz2**
Príklad: XY DB 1-1 ;Výsledok 0
EQ – vráti hodnotu **TRUE** (255), ak sú si výrazy rovné, inak vráti **FALSE** (0). EQ pracuje s výrazmi ako s 32-bitovými číslami so znamienkom.
Syntax: **výraz1 EQ výraz2**
Príklad:
ROVNOST DB 40 EQ 30
;Výsledok 0 (FALSE)
ROVNOST1 DB 40 EQ 40
;Výsledok 255 (TRUE)

Ôsma časť: Makrá, procedúry a operátory

GE – vráti hodnotu TRUE (255), ak výraz1 >= výraz2, inak vráti FALSE (0).
Syntax: **výraz1 GE výraz2**
Príklad: CISLO DB 100 GE 50 ;TRUE
CISLO1 DB 2 GE 3 ;FALSE
GT – vráti hodnotu TRUE (255), ak výraz1 > výraz2, inak vráti FALSE (0).
Syntax: **výraz1 GT výraz2**
Príklad: CISLO2 DB 8 GT 2 ;TRUE
LE – vráti hodnotu TRUE (255), ak výraz1 <= výraz2, inak vráti FALSE (0).
Syntax: **výraz1 LE výraz2**
Príklad: CISLO3 DB 4 LE 2 ;FALSE
CISLO4 DB 1 LE 2 ;TRUE
LT – vráti hodnotu TRUE (255), ak výraz1 < výraz2, inak vráti FALSE (0).
Syntax: **výraz1 LT výraz2**
Príklad: CISLO5 DB 10 LT 20 ;TRUE
NE – vráti hodnotu TRUE (255), ak sa výrazy nerovnajú, inak vráti FALSE (0).
Syntax: **výraz1 NE výraz2**
Príklad: CISLO6 DB 1 NE 2 ;TRUE
CISLO7 DB 2 NE 2 ;FALSE
NOT – operátor vykoná negáciu, t. j. bit, ktorý mal hodnotu 1, bude mať hodnotu 0 a naopak.

Syntax: **NOT výraz**
Príklad: MOV AH,NOT 00001111B
;Výsledok bude 11110000b

AND – operátor vykoná logický súčin.
Syntax: **výraz1 AND výraz2**
Príklad: CISLO8 DB 1111B AND 1100B
;Výsledok bude 1100b

Tabuľka pre operátor AND

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

OR – operátor vykoná logický súčet.
Syntax: **výraz1 OR výraz2**
Príklad: CISLO9 DB 11110000B OR 00001111B
;Výsledok bude 11111111b

Tabuľka pre operátor OR

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

XOR – operátor vykoná logickú operáciu XOR.
Syntax: **výraz1 XOR výraz2**
Príklad: CISLO10 DB 1010B XOR 0101B
;Výsledok bude 1111b

Tabuľka pre operátor XOR

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

LARGE — určuje, že veľkosť offsetu je 32 bitov. Napríklad pri inštrukcii procesora 80386 **JMP [DWORD PTR SKOK]** nie je zo zápisu jasné, či ide o ďaleký skok so 16-bitovým offsetom alebo o blízky skok s 32-bitovým offsetom. Aby sme toto prekladaču „objasnili“, použijeme LARGE **(JMP LARGE [DWORD PTR SKOK]** – ide o inštrukciu krátkeho skoku s 32-bitovým offsetom).
Syntax: **LARGE výraz**

SHORT – operátor sa používa v spojení s inštrukciou JMP. JMP SHORT SKOK - takto upravená inštrukcia je potom preložená iba do dvoch bajtov, zatiaľ čo NEAR skok zaberie 3 bajty. Použitím SHORT sa skráti, ale aj zrýchli kód programu. Turbo assembler generuje automaticky operátor SHORT tam, kde je už známa adresa skoku.
Syntax: **SHORT symbol**
Príklad:

```
SKOK:   ...
        JMP SKOK    ;automatické generovanie
        ...         ;prekladačom
        JMP SHORT SKOK1 ;musíme použiť
SKOK1:  ...         ;operátor
        JMP SKOK2   ;bez použitia operátora
SKOK2:  ...         ;bude inštrukcia dlhá 3 bajty
```

SMALL – určuje, že veľkosť offsetu je 16 bitov. Napríklad pri inštrukcii procesora 80386 **JMP [DWORD PTR SKOK]** nie je zo zápisu jasné, či ide o ďaleký skok so 16-bitovým offsetom alebo o blízky skok s 32-bitovým offsetom. Aby sme toto prekladaču „objasnili“ použijeme **SMALL (JMP SMALL [DWORD PTR SKOK]** – ide o inštrukciu ďalekého skoku so 16-bitovým offsetom).
Syntax: **SMALL výraz**

.TYPE – operátor vráti bajt, ktorého bity sú nastavené podľa povahy mena. Jednotlivé bity majú tento význam:

Bit	Význam
0	symbol sa vzťahuje k programu
1	symbol sa vzťahuje k dátam
2	konštanta
3	priamy adresový režim
4	symbol je register
5	symbol je definovaný
6	symbol je externý

Ak operátor vráti nulu, potom nie je meno definované. Používa sa pri definovaní makra, ku kontrole správnosti parametrov.
Syntax: **.TYPE meno**

Tak to boli vsetky operatory z tabulky, este nam zostali tieto: BYTE, FAR, WORD, NEAR, SETFIELD, GETFIELD atď. Preberieme si ich nabudúce. Teórie nadnes stačilo, poďme ku konkrétnym príkladom. Ukážeme si zase niekoľko užitočných procedúr.

```
;pcr12_97.asm
;*****************
;Procedura umoznuje vypis cisel v nasle-
;dovnych ciselnych sustavach 2,8,10,16.
;Vstup: register AX = cislo
; register BX = zaklad sustavy
;Vystup: cez INT 21h na obrazovku.
;————————————————
transfer1           proc near
                    push ax
                    xor cx, cx
wn0:                xor dx, dx
                    div bx
                    push dx
                    inc cx
                    test ax, ax
                    jnz wn0
wn2:                pop dx
                    or dl, '0'
                    cmp dl, '9'
                    jbe wn3
                    add dl, 7
wn3:                mov ah, 2
                    int 21h
                    loop wn2
                    pop ax
                    ret
                    transfer1
                    endp
;————————————————
;Procedura umoznuje vypis cisel v nasle-
;dovnych ciselnych sustavach 2,8,10,16.
;Vstup: reg. AX = cislo
; reg. BX = zaklad sustavy
; reg. ES = segment videoram
; reg. DI = offset videoram
; reg. SI = offset color, t. j.
;reg. Ukazuje na miesto v pamati, kde je
;ulozeny kod farby, ktorou budu cisla
;zafarbene. Pred zavolanim tejto proc.
;musime na tuto adresu ulozit kod farby.
;POZOR!!! Identifikator color a null je
;definovany v ramci procedury. Null
;urcuje, ako ma byt cislo zarovnane.
;
;Vystup: Priamy zapis do videoram.
;————————————————
transfer2           proc near
                    push ax
                    sub cx,cx
x0:                 sub dx,dx
                    div bx
                    push dx
                    inc cx
                    test ax,ax
                    jnz x0
                    mov ax,word ptr[si+1]
                    cmp ax,cx
                    jbe x2
                    mov bx,cx
                    sub ax,cx
                    mov cx,ax
                    mov dl,48
x1:                 mov es:[di],dl
                    mov al,[si+0]
                    mov es:[di+1],al
                    inc di
                    inc di
                    loop x1
                    mov cx,bx
x2:                 pop dx
                    or dl,'0'
                    cmp dl,'9'
                    jbe x3
                    add dl,7
x3:                 mov es:[di],dl
                    mov al,[si+0]
                    mov es:[di+1],al
                    inc di
                    inc di
                    loop x2
                    pop ax
                    ret
color               db ?
null dw ?
transfer2           endp
;————————————————
;Procedura umoznuje prevod z ASCII na
;cislo v rozsahu max. (0 az 65535).
```

```
;Dokaze previest binarne,
;hexadecimalne a decimalne cisla.
;
;Vstup: reg. BX obsahuje offset adresy
;cisla v pamati.
;
;Vystup: reg. DX obsahuje cisel. hodnotu
;————————————————————
readnum   proc near
          mov al,byte ptr [cs:bx]
          inc bx
          mov dx,0
          mov cl,16
          xor ch,ch
          cmp al,"#"
          jz readnum3
          mov cl,2
          xor ch,ch
          cmp al,"%"
          jz readnum3
          xor ch,ch
          mov cl,10
          dec bx
readnum3: mov al,byte ptr [cs:bx]
          sub al,"0"
          cmp al,10
          jc readnum4
          sub al,"A"-"9"-1
readnum4: cmp al,16
          jc readnum6
          sub al,32
readnum6: cmp al,16
          jnc readnum1
          inc bx
          push bx
          mov bx,dx
          mov dx,0
          push cx
readnum5: add dx,bx
          loop readnum5
          mov bh,ch
          pop cx
          mov bl,al
          add dx,bx
          pop bx
          jmp short readnum3
readnum1: ret
readnum   endp
;————————————————————
;Procedura efektne zmaze obrazovku.
;
;Vstup: register SI = offset
;vstupnych dat.
;
;Vystup: Efektne vycisti obrazovku.
;————————————————————
efekt     proc near
          mov cx,25
alfa:     push cx
          mov di,offset tabulka
          mov cx,8
beta:     push cx
          mov ax,0601h
          mov bh,[si+0]
          mov dx,[di]
          mov cx,[di+2]
          int 10h
          mov ax,0701h
          mov bh,[si+0]
          mov dx,[di+4]
          mov cx,[di+6]
          int 10h
          pop cx
          add di,8
          loop beta
          push cx
          mov cx,word ptr[si+1]
gama:     loop gama
          pop cx
          pop cx
          loop alfa
          ret
          tabulka   db 4,24,0,0
          db 9,24,5,0
          db 14,24,10,0
          db 19,24,15,0
          db 24,24,20,0
          db 29,24,25,0
          db 34,24,30,0
          db 39,24,35,0
          db 44,24,40,0
          db 49,24,45,0
          db 54,24,50,0
          db 59,24,55,0
          db 64,24,60,0
          db 69,24,65,0
          db 74,24,70,0
          db 79,24,75,0
efekt     endp
```

Opis činnosti procedúr

Nadnes som pre vás pripravil tieto procedúry: **transfer1, transfer2, readnum** a **efekt**. Rozoberieme si ich trocha podrobnejšie.

Procedúra **transfer1** vám umožní zobraziť na obrazovke číslo, ktoré sa nachádza v registri AX. O tom, v akej číselnej sústave bude číslo na obrazovke zobrazené, rozhodujete sami (pozri register BX). Môžete použiť binárnu, osmičkovú, hexadecimálnu, ale aj desiatkovú sústavu (2, 8, 16, 10). Výstup na obrazovku je realizovaný cez **prerušenie DOS-u INT 21H, služba 02**[1].

Procedúra **transfer2** je analogická s predchádzajúcou, no líši sa od nej v niekoľkých bodoch. Namiesto služby DOS-u sa pre výstup na obrazovku používa priamy zápis do videoram. Register AX = číslo, BX = základ sústavy (2, 8, 10, 16), ES = segment videoram (0B800H), DI = offset videoram, register SI musí obsahovať offset identifikátora COLOR, ktorý je definovaný v procedúre. COLOR obsahuje kód farby, ktorou budú čísla zafarbené. Pred zavolaním procedúry musíme na adresu COLOR uložiť kód farby a na adresu COLOR+1 (adresa NULL) uložiť číslo, ktoré určuje, ako bude výsledný výpis na obrazovke zarovnaný.

Bez zarovnania	**So zarovnaním**
1	001
10	010
123	123
15	015
5	005

Príklad použitia:

```
MOV AX,0B800H ;segment
MOV ES,AX
MOV DI,1600 ;offset
MOV SI,OFFSET COLOR
MOV DL,7+128 ;farba
MOV [SI+0],DL
MOV DX,6 ;zarovnanie
MOV [SI+1],DX
MOV AX,65535 ;číslo
MOV BX,16 ;číselná sústava
CALL TRANSFER2
```

Procedúra **readnum** pracuje presne naopak ako predchádzajúce procedúry. Umožňuje prevod z ASCII na číslo v rozsahu maximálne 0 - 65535. Vie previesť binárne, hexadecimálne a decimálne čísla. Číslo, ktoré chceme transformovať, musí byť v pamäti uložené v nasledujúcom tvare:

```
CISLO0 DB „#FFFF „
CISLO1 DB „%11111111 „
CISLO2 DB „10000 „
```

Znaky '#' a '%' sú pred číslom uvedené z dôvodu odlíšenia číselnej sústavy. Všimnite si, že každé z čísel sa končí znakom medzera. Tento znak sa považuje za koniec čísla. Ako vstupnú hodnotu treba zadať offset adresy čísla v pamäti (register BX). Napr.: MOV BX,OFFSET CISLO1. Výstupná hodnota, t. j. transformované číslo, sa nachádza v registri DX.

Procedúra efekt zmaže efektne obrazovku. V procedúre sa využíva **prerušenie INT 10H, služby 06** a **07**[2]. Register SI musí obsahovať offset vstupných dát, ktoré môžu byť zadefinované napr. takto:

```
UDAJE     DB 7 ;kod farby
          DW 30000 ;spomalenie procesora
          ...
          MOV SI,OFFSET UDAJE
          CALL EFEKT
          ...
;ASCIICOL.ASM
          .model tiny
          .code
          org 100h
start:    jmp main
udaje     db 7 ;kod farby
          dw 30000 ;spomalenie
;mouse
          sx dw 0
          sy dw 0
text      db 'ASCII & COLOR TABLE (c) 1997'
          db 32,'Copyright Universal systems'
```

```
text1     db 'Press any key or move'
          db 32,'mouse for return to DOS.'
          include pcr11_97.asm
          include pcr12_97.asm
main:     call kurzor_off
          xor ax,ax
          int 33h
          mov ax,3
          int 33h
          mov [sx],cx
          mov [sy],dx
          mov ax,0b800h
          mov es,ax
          mov ax,7*256+32
          call cls2
          mov ax,0 ;Riadok.
          mov bx,12 ;Stlpec.
          call poloha2
          mov di,ax
          mov si,offset text
          mov cx,56 ;Dlzka textu.
          mov al,7 ;Farba.
          call vypis
          mov ax,24 ;Riadok.
          mov bx,17 ;Stlpec.
          call poloha2
          mov di,ax
          mov si,offset text1
          mov cx,46 ;Dlzka textu.
          mov al,128+7 ;Farba.
          call vypis
          mov ax,2 ;Riadok.
          mov bx,4 ;Stlpec.
          call poloha2
          mov di,ax
          mov ax,3 ;Riadok.
          mov bx,4 ;Stlpec.
          xor dh,dh
          mov dl,0
          mov si,offset color
          mov cx,3 ;Zarovnanie
          mov [si+1],cx
          mov cx,256
x6:       push cx
          push ax
          push dx
          push bx
          mov es:[di],dl
          inc di
          inc di
          mov al,'-'
          mov es:[di],al
          inc di
          inc di
          mov [si+0],dl
          mov al,dl
          mov bx,10 ;Sustava
          call transfer2
          mov cl,179
          mov es:[di],cl
          inc di
          inc di
          pop bx
          pop dx
          pop ax
          pop cx
          inc dl
          push ax
          push dx
          call poloha2
          mov di,ax
          pop dx
          pop ax
          inc ax
          cmp ax,24
          jz posun
navrat:   loop x6
          jmp pok
posun:    add bx,6
          mov ax,2
          jmp navrat
pok:      mov ax,3
          int 33h
          cmp cx,[sx]
          jne koniec
          cmp dx,[sy]
          jne koniec
          in al,60h
          test al,80h
          jne pok
koniec:   mov si,offset udaje
          call efekt
          call kurzor_on
          int 20h
          end start
```

Vysvetlenie programu

Najskôr by som mal povedať, na čo program slúži. Program zobrazí na obrazovku ASCII a COLOR tabuľku v práve aktívnej znakovej sade. Na to, ako využiť tieto informácie, určite prídete sami. Teraz si program opíšeme trocha podrobnejšie.

Model tiny, ORG 100h, pri štarte od adresy 100h musíme preskočiť dáta, definujeme si vstupné údaje pre procedúru efekt, definujeme si pamäťové miesto na uchovanie polohy myšky, ďalej nasleduje text.

Vložíme pomocou direktívy **INCLUDE** potrebné procedúry. Všimnite si, že budeme potrebovať aj procedúry z predošlej časti (pozri PC REVUE č. 11/1997).

Dostali sme sa na začiatok hlavného programu, vypneme kurzor. Nasleduje inicializácia myši a následné uchovanie poslednej pozície kurzora myši **(Prerušenie INT 33H, služba 00[3]a 03[4])**. Ďalej nastavíme segmentový register ES na 0B800H, čo je segmentová adresa videoram. Zmažeme obrazovku a pokračujeme výpisom textu (TEXT, TEXT1). Prichádzame do oblasti, kde sa začína výkonná rutina, ktorá celú tabuľku zobrazí na obrazovku. V podstate ide o jednoduchý cyklus.

Vypočítame si zo súradníc (2,4) offsetovú adresu vo videoram. Uložíme ju do registra DI. Registre AX a BX naplníme súradnicami ďalšieho riadka (3,4). Vynulujeme DX (t. j. DH aj DL). Register SI nastavíme na adresu COLOR a v registri CX zadáme zarovnávanie čísel na 3 miesta. Potom CX naplníme hodnotou 256, ktorá určuje, koľkokrát sa bude cyklus opakovať. Nasleduje uloženie potrebných registrov do zásobníka. Ďalej sa vykonáva už len samotné vykresľovanie tabuľky. Najprv sa vykreslí ASCII kód znaku, potom znak mínus. Nastaví sa kód farby (MOV [SI+0],DL). Zvolíme desiatkovú sústavu a vypíšeme číslo na obrazovku v zodpovedajúcej farbe. Za číslom sa ešte vypíše ASCII znak „|“ (kód 179). Potom nasleduje už len výpočet offsetovej adresy ďalšieho riadka a test na ukončenie cyklu. Pokiaľ nie sú všetky znaky na obrazovke zobrazené, cyklus sa opakuje.

Na návestí „POK:“ sa opäť zisťuje poloha myši, testuje sa klávesnica (IN AL, 60H a TEST AL,80H). Čiže ak ste pohli myškou, prípadne stačili nejaký kláves – program sa ukončí. Pred ukončením ešte efektne zmažeme obrazovku a zapneme kurzor.

Nabudúce

Dokončíme operátory, povieme si, ako pracovať s programom Turbo Debugger, ako vytvoriť knižnicu procedúr pomocou programu TLIB.EXE, a uvedieme nejaké príklady.

Literatúra

[6] Helppc, Verzia: 2.10, 1991, Copyright David Jurgens.
[7] M. Brandejs: MS-DOS 6. Grada 1993.
[8] Norton Guide. Peter Norton Computing, Inc., 1987.
[9] J. Zápalka: Anatomie IBM PC. Grada 1993.
[10] Ralf Brown: Interrupt List - Release 33. Pittsburgh PA, U.S.A. 1993.

Vysvetlivky:

[1]**Prerušenie INT 21H, Služba 02**
(Zápis znaku) – služba vypíše znak na štandardné výstupné zariadenie (obrazovka, disk, tlačiareň). V registri AH = 02H je kód služby, v registri DL je ASCII kód znaku.

[2]**Prerušenie INT 10H, Služba 06** (Rolovanie hore) a **07** (Rolovanie dole) – roluje v aktívnej stránke okno zadané ľavým horným a pravým dolným rohom o zadaný počet riadkov hore, resp. dole, prípadne zmaže zadané okno. Vstup: register AH = 06 alebo 07 – kód služby, AL – počet riadkov (ak je nula, zmaže nastavené okno), BH – farba pre uvoľnené riadky, CH – riadková a CL – stĺpcová súradnica ľavého horného rohu okna, DH – riadková a DL – stĺpcová súradnica pravého dolného rohu okna. Obsah riadkov, ktoré opustia obrazovku, sa stratí. V spodnej časti okna sa vkladajú nové riadky vyplnené medzerami. Tieto riadky majú farbu podľa registra BH.

[3]**Prerušenie INT 33H**, služba 00 – funkcia vykoná inicializáciu myši, takisto nastaví implicitný rozsah súradníc.

[4]**Prerušenie INT 33H**, služba 03 – funkcia zistí pozíciu myši a stav tlačidiel. Vstup v registri AX číslo služby (03). Výstup – register BX obsahuje stav tlačidiel myši, CX – horizontálne a DX – vertikálne súradnice kurzora. Bit 0 v registri BX zodpovedá ľavému, bit 1 pravému a bit 2 prostrednému tlačidlu myši. Ak je niektorý z týchto bitov nastavený na 1 (TRUE), potom ide o stlačenie daného tlačidla.

Deviata časť: Práca s programom Turbo Debugger

Čo nás dnes čaká? Doplníme si tabuľku operátorov o niekoľko ďalších typov, ukážeme si, ako pracovať s Turbo Debuggerom, pokúsime sa vytvoriť knižnicu procedúr (mám na mysli procedúry, ktoré boli publikované v číslach 12/1997, 1 a 2/1998), a ako už býva dobrým zvykom, nezabudnem ani na príklady. DNES: ako naprogramovať vstup z klávesnice.

Doplnkové operátory

? – určuje, že alokované pamäťové miesto nebude inicializované. Typ môže byť napr.: DB, DW, DD, DQ ... alebo meno zložené z dátového typu.

Syntax: **typ ?**

Príklad: CISLO DB ?
POLE DW 40 DUP(?)

| – operátor vráti hodnotu položky tabuľky, tak ako bola uvedená pri definovaní šablóny tabuľky. Operátor sa používa iba v súvislosti s dátovým typom tabuľka.

Syntax: **Názov_tabuľky | Položka_tabuľky**

Príklad: CALL +TABULKA1 | BLOK1

DUP – opakuje umiestnenie dát s hodnotou výraz. Počet určuje koľkokrát sa bude umiestnenie opakovať. Výraz môže byť DB, DW, DD, DF, DP a pod. Ako výraz môžeme použiť aj operátor DUP, ale počet vnorení môže byť maximálne 17.

Syntax: **počet DUP(výraz [,výraz]...)**

Príklad: POLE1 DW 40 DUP(?)
POLE2 DB 2 DUP(6 DUP(3 DUP(0)))
;bude vytvorené pole 2x6x3 prvky

GETFIELD – dosadí hodnotu položky poľa (meno_položky) z bitového poľa uloženého v zdrojovom registri alebo pamäti do cieľového registra.

Syntax: **GETFIELD meno_položky cieľový reg.,zdrojový reg./pamäť**

Príklad: FLAG RECORD A:1,B:1,C:1,D:1
GETFIELD D BL,AX

SETFIELD – dosadí špecifikovanú položku bitového poľa zo zdrojového registra do bitového poľa uloženého v cieľovom registri, alebo pamäti.

Syntax: **SETFIELD meno_položky cieľový reg./pamäť, zdrojový reg.**

Príklad: FLAG RECORD A:1,B:1,C:1,D:1
SETFIELD B AX,BL

Teraz bude nasledovať opis operátorov, ktoré nie je možné priamo používať v režime MASM. Tieto operátory sa používajú v spojení s operátorom typ PTR symbol.

FAR – je 32-bitový smerník a určuje, že výraz je uložený, resp. definovaný v inom segmente, t. j. ak používate smerníky FAR, môžete mať niekoľko code segmentov, čo vysvetľuje, prečo môžeme vytvárať programy dlhšie ako 64 KB.

Syntax: **FAR výraz**

Príklad: CALL FAR PTR ADRESA

NEAR – je 16-bitový smerník, ktorý udáva adresu (offset) v rámci príslušného segmentu. Používa sa v spojení s inštrukciami CALL a JMP.

Syntax: **NEAR výraz**

Príklad: CALL NEAR PTR ADRESA

BYTE – informuje prekladač o veľkosti pamäte určenej adresovým výrazom.

Syntax: **BYTE adresový_výraz**

Príklad: MOV [BYTE PTR ES]:100H,1
MOV [BYTE PTR X],99
...
X DB ?

WORD – informuje prekladač o veľkosti pamäte určenej adresovým výrazom.

Syntax: **WORD adresový_výraz**

Príklad: MOV [WORD PTR ES]:100H,1
MOV [WORD PTR X],9999
...
X DW ?

DWORD – informuje prekladač o veľkosti pamäte určenej adresovým výrazom.

Syntax: **DWORD adresový_výraz**

Príklad: JMP [DWORD PTR ES:100H]

FWORD – určuje, že výraz je 32-bitový smerník. Používa sa pri procesoroch 80386 a vyšších na určenie, že ide o adresový výraz s celkovou dĺžkou 48 bitov.

Syntax: **FWORD adresový_výraz**

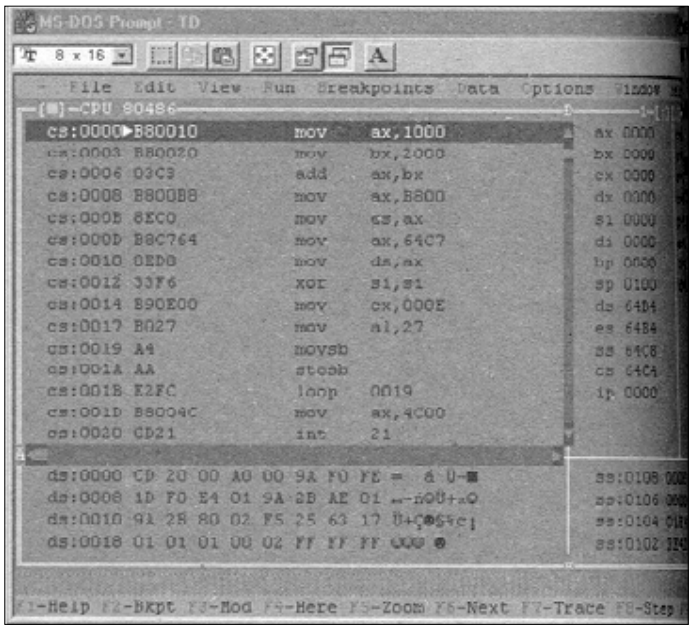
Príklad: CALL FWORD PTR ADRESA

PWORD – operátor je totožný s funkciou operátora FWORD, ale PWORD je možné použiť v obidvoch režimoch (MASM aj IDEAL).

Syntax: **PWORD adresový_výraz**

QWORD – pre tento operátor platí presne to isté ako pre operátor BYTE, iba s tým rozdielom, že veľkosť pamäťového miesta je 8 bajtov.

Syntax: **QWORD adresový_výraz**



Obr. 1 Pracovné prostredie programu Turbo Debugger

TBYTE – pre tento operátor platí presne to isté ako pre operátor BYTE, iba s tým rozdielom, že veľkosť pamäťového miesta je 10 bajtov.

Syntax: **TBYTE adresový_výraz**

UNKNOWN – odstraňuje typovú informáciu zo symbolu. Symbol je adresa, jej tvar sa nezmení. To umožňuje považovať jedno dátové miesto za typ, ktorý je v danom okamihu najvhodnejší.

Syntax: **UNKNOWN symbol**

CODEPTR – vráti preddefinovanú veľkosť adresy procedúry podľa zvoleného pamäťového modelu. Vráti WORD pre modely TINY, SMALL, COMPACT, TPASCAL a FLAT (majú jeden kódový segment). DWORD vráti pre ostatné modely.

Syntax: **CODEPTR výraz**

Príklad: CALL [CODEPTR PTR ADRESA]

DATAPTR – podobne ako pri CODEPTR určuje DATAPTR výraz ako smerník NEAR alebo FAR podľa zvoleného pamäťového modelu. NEAR pre modely TINY, SMALL, MEDIUM a FLAT (majú jeden dátový segment), FAR pre ostatné modely.

Syntax: **DATAPTR výraz**

Príklad: CISLO DB 0
...
MOV [DATAPTR PTR CISLO],100

PROC – určuje, či výraz je ďaleký (FAR) alebo blízky (NEAR). Symbol musí byť adresa. Či ide o FAR alebo NEAR adresu, sa určí podľa zvoleného pamäťového modelu. Pre modely TINY, SMALL, COMPACT a FLAT je smerník NEAR, pre ostatné modely je FAR.

Syntax: **PROC symbol**

Práca s programom TURBO DEBUGGER

Skôr ako začneme s programom pracovať, opíšeme si jeho prostredie (pozri obrázok 1) a menu. Čo sa týka prostredia, je podobné tomu, aké poznáte z programov Turbo C, Turbo Pascal a iných, ktoré boli vytvorené pomocou Turbo Vision.

Prostredie sa skladá z troch základných častí: menu (main menu), pracovná plocha (window area) a stavový riadok (status line).

Main menu (hlavné menu)

Menu sa nachádza na prvom riadku obrazovky a je možné pomocou neho používať všetky príkazy, ktoré program Turbo Debugger (ďalej len TD) obsahuje. Toto menu je vždy viditeľné (výnimku tvorí iba zobrazenie používateľskej obrazovky ALT+F5) a je kedykoľvek prístupné pomocou klávesu F10, prípadne myšky. Menu obsahuje vždy niekoľko ponúk, z nich len jedna je zvýraznená. Voľbou niektorej z ponúk vykonáme príslušný príkaz, ktorý sa pod ponukou skrýva. Množstvo príkazov je z rôznych menu kedykoľvek prístupné priamo stlačením HOT KEY1. Teraz si opíšeme aspoň tie najdôležitejšie časti PULL-DOWN menu.

ş (Systém) menu – do tohto menu sa dostanete kedykoľvek stlačením klávesov ALT+SPACE. Obsahuje nasledujúce položky: **REPAINT DESKTOP** – používa sa na opätovné vykreslenie pracovnej plochy, **RESTORE STANDART** – zatvorí všetky okná a otvorí základné okno (ako pri spustení programu), **ABOUT** – informácie o výrobcovi softvéru (verzia, názov programu a COPYRIGHT).

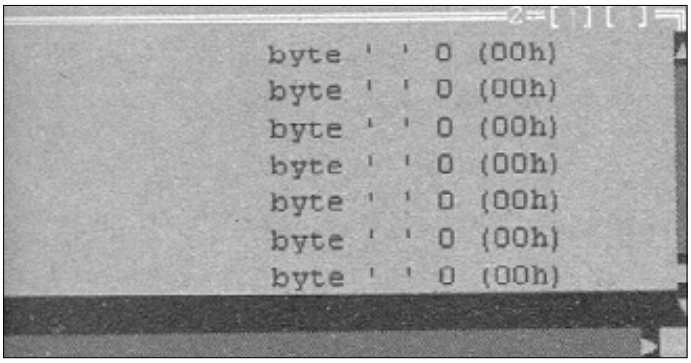
FILE – obsahuje: **OPEN** – nahrať nový program do TD, **CHANGE DIR** – zmena pracovného adresára, **GET INFO** – informácie o systéme, **DOS SHELL** – dočasný odchod do prostredia DOS-u (návrat je možný príkazom EXIT), **RESIDENT** – obdoba príkazu DOS SHELL

(spraví z TD rezidentný program – návrat CTRL+BREAK), **SYMBOL LOAD** – nahrá novú tabuľku symbolov, **TABLE RELOCATE** – špecifikácia nového základného segmentu pre tabuľku symbolov. **QUIT** – ukončenie práce s programom (ALT+X).

EDIT -> COPY – kopírovanie položky, **PASTE** – vložiť položku, **COPY TO LOG** – kopírovať do protokolu (záznamu), **DUMP PANE TO LOG** – výpis z pamäte do protokolu (záznamu).

VIEW – v tomto menu nájdete položky, ktoré je možné zobraziť na obrazovku. Ide o: **BREAKPOINTS, STACK, LOG, WATCHES** (sledovacie okno), **VARIABLES, MODULE, FILE, CPU, DUMP** (výpis z pamäte), **REGISTERS, NUMERIC PROCESSOR, EXECUTION HISTORY, HIERARCHY, WINDOWS MESSAGES, CLIPBOARD** (jeho obsah) a **ANOTHER** (táto položka menu obsahuje ešte **MODULE, DUMP** a **FILE**).

RUN – obsahuje príkazy na spúšťanie a krokovanie (trasovanie) programu. **RUN** – spustenie programu bez trasovania, **GOTO CURSOR** – spustí program od miesta, kde ukazuje kurzor, **TRACE INTO** – trasovanie (po jednej inštrukcii), **STEP OVER** – trasovanie (po jednej inštrukcii, ale podprogramy volané inštrukciou *CALL* budú vykonané bez možnosti krokovania), **EXECUTE TO** – spustí program od adresy, ktorú zadáte, **UNTIL RETURN** – spustí aktuálnu rutinu a vykonáva ju, až kým nenarazí na return, **ANIMATE** – postupné vykonávanie inštrukcií (môžete si zvoliť rýchlosť animácie), **BACK TRACE** – vráti poslednú trasovanú inštrukciu alebo zdrojový riadok (UNDO), **INSTRUCTION TRACE** – trasovanie jednoduchých strojových inštrukcií, **ARGUMENTS** – nastavenie argumentov v príkazovom riadku pre krokovaný program. Používa sa, ak sa má program spúšťať s nejakým parametrom, napr.: program.com /x /p, **PROGRAM RESET** – znovuzavedenie programu z disku (RELOAD).



Obr. 2 Okno Watches

BREAKPOINTS – je to miesto v programe, kde sa má program dočasne zastaviť. **TOGGLE** – nastavenie alebo zrušenie nepodmieneného breakpointu na riadku, na ktorom sa nachádza kurzor. Ak je breakpoint nastavený, je príslušný riadok zdrojového programu zafarbený červenou farbou, AT – nastavenie breakpointu na konkrétnu adresu, **CHANGED MEMORY GLOBAL** – nastavenie breakpointu na pamäťovú oblasť, **EXPRESSION TRUE GLOBAL** – nastavenie breakpointu na pravdivý výraz, **HARDWARE BREAKPOINT** – nastavenie HW breakpointu, **DELETE ALL** – zrušenie všetkých nastavených breakpointov.

DATA – obsahuje: **INSPECT** – kontrola špecifikovaných premenných alebo výrazov, **EVALUATE/MODIFY** – vyhodnotiť a zobraziť hodnotu výrazu, **ADD WATCH** – pridať nový výraz do okna **WATCHES, FUNCTION RETURN** – ukáže hodnotu, ktorá je vrátená z rutiny.

OPTIONS – menu obsahuje: **LANGUAGE** – nastavenie výrazu jazyka, **MACROS** – vytvorenie alebo zrušenie klávesového makra. Táto položka menu obsahuje ešte ďalšie položky. Sú to: **CREATE** – vytvorenie nového klávesového makra, **STOP RECORDING** – zastavenie nahrávania klávesového makra, **REMOVE** – odstránenie makra, **DELETE ALL** – vymazanie všetkých klávesových makier. **DISPLAY OPTIONS** – nastavenie vzhľadu obrazovky, **PATH FOR SOURCE** – špecifikovanie adresára na hľadanie zdrojových súborov, **SAVE OPTIONS** – nahranie aktuálnej konfigurácie prostredia na disk, **RESTORE OPTIONS** – obnovenie konfigurácie z disku.

WINDOW – obsahuje položky pre prácu s oknami. **ZOOM** – zmenšenie alebo zväčšenie aktuálneho okna, **NEXT** – choď na ďalšie okno, **NEXT PANE** – presuň sa na ďalšie políčko v aktuálnom okne, **SIZE/MOVE** – presun alebo zmena veľkosti okna, **ICONIZE/RESTORE** – minimalizovať okno alebo obnoviť minimalizované okno, **CLOSE** – zatvoriť aktuálne okno, **UNDO CLOSE** – vráti späť posledné zatvorené okno, **USER SCREEN** – pozrieť sa na používateľskú obrazovku.

HELP –- obsahuje: **INDEX** – ukáže zoznam pre online help, **PREVIOUS TOPIC** – ukáže predchádzajúcu obrazovku online helpu, **HELP ON HELP** –- ako používať online help.

Window area (pracovná plocha)

Väčšina akcií, ktoré vidíte, sa odohráva v okienkach. Okienko je oblasť obrazovky, ktorú môžete premiestniť, prekrývať, zatvárať a otvárať, prípadne meniť jeho veľkosť. Otvorených okien môže byť toľko, koľko sa vojde do pamäte, ale len jedno z nich môže byť aktívne. Aktívne okno je vždy orámované dvojitým rámčekom a nachádza sa na vrchu obrazovky, teda nijaké iné okno ho ani čiastočne neprekrýva.

Status line (stavový riadok)

Stavový riadok sa nachádza na spodnom riadku obrazovky a má tieto funkcie:

1. Napovedá význam a názov klávesov alebo HOT KEY, ktoré je možné okamžite využiť v súvislosti s aktuálnym stavom prostredia.
2. Informuje o práve vykonávanej činnosti prostredia.
3. Ak sa pohybujete v menu, stavový riadok slúži na stručný opis významu práve označenej (zvýraznenej) ponuky menu.

Kláves CTRL

Pomocou tohto klávesu môžete nastavovať rôzne parametre v práve aktívnom okne. Teda ak sa nachádzate napr. v okne registre2 a stlačíte kláves CTRL, sú vám ponúknuté nasledujúce možnosti:

a) môžete inkrementovať (zväčšovať) obsah registra,
b) dekrementovať (zmenšovať) obsah registra,
c) zero (nulovať) register,
d) change (zmeniť) hodnotu v registri,
e) môžete sa prepínať medzi 32–bitovými a 16–bitovými registrami.

Táto vlastnosť platí aj pre ostatné okná. Medzi jednotlivými časťami okna sa môžete prepínať tabulátorom.

Krokovanie programu

Krokovanie programu je veľmi dôležitá vec, čo sa týka hľadania chýb v programe. Ak váš program nepracuje tak, ako si myslíte, že by mal pracovať, a už nemáte nijaké nápady, skúste použiť TD. Krokovanie je silný nástroj a často jediný ako zistiť o programe, ako pracuje, aké má vstupy a výstupy atď. Na nasledujúcom príklade si ukážeme, ako program krokovať.

```
;KROK.ASM
.model small
.stack 100h
.data
text                db „TURBO DEBUGGER"
dlzka               sizestr <TURBO DEBUGGER>
.code
;scitanie dvoch celych cisel
start:              mov ax,1000h
                    mov bx,2000h
                    add ax,bx
;vypis textu
                    mov ax,0b800h
                    mov es,ax
                    mov ax,seg text
                    mov ds,ax
                    xor si,si
                    mov cx,dlzka
                    mov al,2*16+7
cyklus:             movsb
                    stosb
                    loop cyklus
                    mov ax,4c00h
                    int 21h
                    end start
```

Program preložte takto:

```
TASM                KROK.ASM
TLINK               KROK.OBJ
```

Vznikne súbor s príponou EXE. Teraz spustite TD z príkazového riadka aj so súborom krok.exe (TD KROK.EXE). Po spustení programu sa zobrazí okno, ktoré vás informuje o tom, že program nemá tabuľku symbolov, kliknite na OK. Ocitnete sa v hlavnom okne TD, v okne CPU. Ak ste všetko urobili správne, v okne vidieť program **krok.exe** (pozri obr. 1).

Skôr než začneme program krokovať, urobte nasledujúce: kliknite na OPTIONS/DISPLAY OPTIONS a v boxe INTEGER FORMAT nastavte BOTH, ďalej si tu môžete nastaviť veľkosť obrazovky (SCREEN LINES), kliknite na 43/50. Túto možnosť oceníte až pri viacerých otvorených oknách.

Iste ste si už všimli, že napravo od boxu inštrukcií je box, ktorý obsahuje registre a že neobsahuje polovice registrov. Ak vám to chýba môžete si takéto okno vytvoriť. Kliknite na DATA/ADD WATCH a vložte postupne polovice registrov (ah, al, bh, bl, ch, cl, dh, dl). Po vložení prvého registra sa otvorí okno WATCHES, ktoré obsahuje zadané registre (pozri obr. 2).

Do okna môžete vložiť 16-bitové, prípadne aj segmentové registre. Výhodou tohto okna je, že obsah registrov vidíte ako hodnotu CHAR, DEC a HEX. Prepnite sa teraz do okna CPU. Znak „„ ukazuje na prvú vykonateľnú inštrukciu. Teraz môžete začať krokovať program (kláves F7). Stlačením klávesu F7 sa znak „„ posunie na ďalšiu vykonateľnú inštrukciu, pričom sa vykoná predchádzajúca inštrukcia. Všimnite si, že register AX sa naplnil hodnotou 1000h. Takto sa postupuje až po koniec programu, pričom sa stále sleduje obsah registrov, príznakov, prípadne obsah zásobníka. Náš skúšobný program sa skladá zo sčítania dvoch čísel (prvé tri inštrukcie) a z časti, ktorá zobrazí na obrazovku text. Pri krokovaní časti, ktorá zobrazuje text, si všimnite, že ak TD narazí na inštrukciu

LOOP, ukazuje, či sa vykoná skok dopredu alebo dozadu (pozri šípku ˝↑˝ a ˝↓˝). Podobne je to aj pri inštrukciách podmienených a nepodmienených skokov.

Ak ste prišli až po inštrukcie MOV AX,4C00H a INT 21H, vykonajte ich. Zobrazí sa správa TERMINATED, EXIT CODE 0 (program bol ukončený s chybovým kódom 0). Kliknite na OK. Ak chcete krokovať program znovu stlačte kláves F7. Zobrazí sa správa: PROGRAM ALREADY TERMINATED, RELOAD? Kliknite na YES, program sa znovu zaktualizuje, ale neprekreslí sa pracovné okno CPU, to musíte urobiť vy. Kliknite na VIEW a potom na CPU a opäť môžete program krokovať. Ak chcete krokovať iný program, treba ho nahrať. Kliknite na FILE/OPEN a pohľadajte na disku program, ktorý chcete nahrať. Okrem programov s príponou EXE je možné nahrať aj programy s príponou COM.

Toľko k programu TD. Azda je z toho jasné, ako s programom pracovať. Ak nebudete vedieť, ako ďalej, použite HELP.

Vytvorenie knižnice programom TLIB.exe

Skôr než sa dostane k vytvoreniu knižnice, povieme si najskôr niečo o programe TLIB.EXE. Pri programovaní v assembleri, v jazyku PASCAL, C sa často využívajú už hotové funkcie, procedúry, ktoré sa nachádzajú v knižniciach. Knižnice sú označené príponou *.LIB. Vytvárať knižnice je veľmi výhodné aj vzhľadom na možnosť používať funkcie iných autorov bez potreby poznať zdrojové texty. Ďalšou výhodou je zvýšenie prehľadnosti pri tvorbe rozsiahlych programov. Na vytváranie knižníc sa používa program TLIB.EXE. Tento program dokáže pracovať, nie len s knižnicami vytvorenými v assembleri, ale aj s knižnicami iných programovacích jazykov. Program TLIB.EXE dokáže vytvoriť novú knižnicu, pridať do existujúcej knižnice nový modul, zrušiť ho, prípadne ho z knižnice získať. Pokiaľ dochádza k modifikovaniu knižnice, vytvára program TLIB.EXE záložnú kópiu pôvodnej knižnice.

```
 Syntax: TLIB libname [/C] [/E] commands, listfile
Libname             library file pathname
commands              sequence of operations to be performed
(optional)
listfile            file name for listing file (optional)

A command is of the form: <symbol>modulename, where <symbol> is:
+ add modulename to the library
- remove modulename from the library
* extract modulename without removing it
-+ or +- replace modulename in library
-* or *- extract modulename and remove it
/C case-sensitive library
/E create extended dictionary
Use @filepath to continue from file „filepath".
Use '&' at end of a line to continue onto the next line.
```

LIBNAME – je názov knižnice

COMMANDS – tento parameter určuje, čo budeme s knižnicou robiť. Sú povolené tieto znaky:

+ program TLIB.EXE pridá nový modul do knižnice. V prípade, že meno označuje knižnicu, pripojí všetky moduly. Pri pokuse pripojiť modul, ktorý je už v knižnici definovaný, sa vypíše chybové hlásenie **´Warning: ´xxxxx´ already in LIB, not changed´**.

- daný modul bude z knižnice vymazaný. Mená neexistujúcich modulov budú zobrazené na obrazovke spolu s chybovým hlásením **´Warning: ´xxxxx´ not found in library´**.

* TLIB.EXE vytvorí súbor obsahujúci požadovaný modul. Ak modul neexistuje, vypíše sa chybové hlásenie ´Warning: ´xxxxx´ not found in library´.

* – modul bude umiestnený do cieľového súboru a v zdrojovej knižnici vymazaný.

+ – modul bude vymenený za nový.

/C – parameter sa používa na rozlišovanie malých a veľkých písmen.

/E – parameter slúži na vytvorenie rozšíreného zoznamu knižnice, má vplyv na rýchlejšie linkovanie.

LISTFILE – meno súboru, do ktorého sa zapíše obsah knižnice. Ak sa teda chcete dozvedieť, čo knižnica obsahuje, zapíšte nasledujúci text do príkazového riadka:

```
TLIB.EXE xxxxx.lib, vypis.txt
```

xxxxx.lib – predstavuje knižnicu, z ktorej chcete získať výpis modulov. Po vykonaní príkazu sa vytvorí súbor vypis.txt, ktorý bude obsahovať výpis modulov umiestnených v knižnici.

V nasledujúcich riadkoch sa vám pokúsim objasniť princíp, ako si vytvoriť vlastnú knižnicu. Takže poďme nato.

Čo na to potrebujeme? Procedúry, napr.: tie, ktoré boli uverejnené v prechádzajúcich častiach seriálu. Treba ich zo súboru extrahovať tak, aby každá z nich bola v samostatnom súbore. Vytvorenie knižnice si ukážeme na procedúrach kurzor_on a kurzor_off. Pretože knižnica je zložená zo súborov s príponou OBJ, musíme si na procedúry vytvoriť šablónu, pomocou ktorej získame zo súborov s príponou ASM súbory OBJ. Šablóna môže mať tvar:

```
.MODEL TINY
ORG 100H
.CODE
PUBLIC KURZOR_ON
START:    INCLUDE KURZORON.ASM
END START
```

Takto postupne preložíte všetky procedúry, ktoré chcete mať v knižnici. Keďže procedúry budú definované v inom module, musia byť z tohto modulu viditeľné, preto je v šablóne použitá direktíva PUBLIC názov_procedúry. Ak všetko pôjde bez problémov, získate súbory kurzoron.obj a kurzorof.obj. Teraz už môžeme pristúpiť k vytvoreniu knižnice.

```
TLIB.EXE    KNIHPROC.LIB  +kuzoron.obj  +kurzorof.obj
```

Vytvorí sa knižnica KNIHPROC.LIB. Ak sa chcete presvedčiť, či daná knižnica obsahuje súbory, ktoré ste do nej vložili, napíšte TLIB.EXE knihproc.lib, vypis.txt. Ešte si ukážeme, ako takúto knižnicu použiť.

```
.model tiny
.code
org 100h
start:              jmp   start1
extrn               kurzor_off:proc
extrn               kurzor_on:proc
includelib knihproc.lib
NastavKurzor        MACRO x,y
                    mov   bh,0
                    mov   dl,x
                    mov   dh,y
                    sub   dx,0101h
                    mov   ah,02
                    int   10h
                    ENDM
text                b 'Kurzor zmizol.$'
start1:             mov ax,0003
                    int 10h
                    call kurzor_off
                    nastavkurzor 30,12
                    mov ah,09h
                    mov dx,offset text
                    int 21h
                    mov ax,0
                    int 16h
                    call kurzor_on
                    int 20h
                    end start
```

Program je jednoduchý. Najprv nastaví mód zobrazovania na 3, čo je textový režim 80 x 25 znakov, vypne kurzor, vypíše text a znova obnoví kurzor. Všimnite si, ako sú zadefinované procedúry *kurzor_on a kurzor_off* (pozri direktívu EXTRN).

Praktické príklady – vstup z klávesnice

Niekedy treba získať nejaké údaje z klávesnice. Na načítanie reťazca z klávesnice sa používa služba 0Ah. Ide o službu, ktorá patrí pod prerušenie INT 21h. Čo presne služba robí? Číta reťazec znakov zo štandardného vstupného zariadenia (najčastejšie z klávesnice) a ukladá ho v pamäti od adresy určenej programátorom. Vstup sa ukončí po prečítaní znaku CR (kláves ENTER - 0Dh). Prvý bajt poľa odovzdaného službe musí byť naplnený hodnotou určujúcou maximálny počet čítaných znakov v intervale <1 až 255>. Druhý bajt naplní služba počtom skutočne zadaných znakov vrátane konečného CR. Od tretieho bajtu sú ukladané čítané znaky. Služba reaguje na stlačenie kombinácie klávesov CTRL+C alebo CTRL+BREAK generovaním INT 23h. Pri čítaní rozšírených kódových kombinácií (napr. kláves F1) sú do pamäte uložené dva bajty, z nich prvý je nulový. Zápis tejto služby v assembleri by mohol vyzerať napríklad takto:

```
...
CISLO DB 2 ;max. počet čítaných znakov
DB ? ;tento bajt naplní služba počtom
;skutočne prečítaných znakov
DB 3 DUP (?) ;priestor pre
;prečítané znaky
...
MOV AH,0AH
MOV DX,SEG CISLO
MOV DS,DX
MOV DX,OFFSET CISLO
INT 21H
```

Napokon najlepšie sa to vyskúša na príklade. Ukážeme si teraz jednoduchý program, ktorému na vstupe zadáte nejaký znak z klávesnice a on vám vypíše na obrazovku ASCII kód tohto znaku. Aby program správne pracoval, potrebujete ešte súbory **pcr11_97.asm** a **pcr12_97.asm** z minulých čísel PC REVUE. Tieto dva súbory totiž obsahujú procedúry, ktoré nasledujúci program používa. Program preložte takto: **TASM vstup.asm /M** a **TLINK vstup.obj**. Myslím, že program nie je potrebné opisovať. Neobsahuje nijaké nové prvky. Ak vám predsa len nebude niečo jasné, pozrite sa do starších čísel PC REVUE.

```
;VSTUP.ASM
.model small
.stack 100h
.data
cislo     db 2,00,3 dup (?)
```

```
text1     db „Zadaj znak: $"
text2     db „Zadany znak ma"
db 32,"ASCII kod: $"
text3     db „Press any key"
db 32,"for return" db 32,"to DOS."
.code
include pcr11_97.asm
include pcr12_97.asm
start:    mov ax,@data
          mov ds,ax
          mov ax,0b800h
          mov es,ax
          mov ax,3
          int 10h
          mov dx,10*256+11
          call poloha1
          lea   dx,text1
          mov   ah,09h
          int   21h
          mov   ah,0ah
          lea   dx,cislo
          int   21h
          mov si,offset cislo
          mov bl,[si+2]
          push bx
          mov dx,12*256+11
          call poloha1
          lea   dx,text2
          mov   ah,09h
          int   21h
          pop ax
          mov bx,10
          call transfer1
          mov di,08D4h
          mov si,offset text3
          mov cx,32 ;Dlzka textu
          mov al,7+128 ;Farba
cyklus:   movsb
          stosb
          loop cyklus
          mov ax,0
                    int 16h
                    mov ax,4c00h
                    int 21h
                    end start
```

Predchádzajúci program používal na vstup znakov z klávesnice službu DOS-u. Nasledujúci program je vlastne naprogramovaný vstup bez použitia služby DOS-u, to znamená, že všetky testy (napr.: koniec riadka, backspace, enter atď.) si musíte naprogramovať sami. Program preložíte do strojového kódu takto: **TASM vstup2.asm a TLINK vstup2.obj /t**

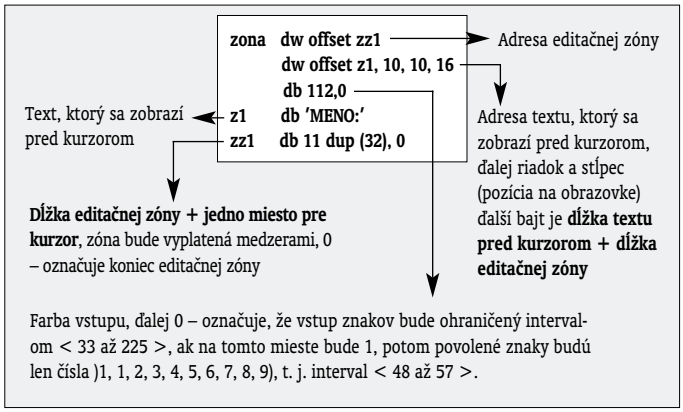
```
;VSTUP2.ASM
.model tiny .code org 100h
start:              jmp ip
zona1               dw  offset zz1
                    dw offset z1,10,10,16
                    db 112,0
z1                  db 'Meno:'
zz1                 db 11 dup (32),0
zona2               dw  offset zz2
                    dw offset z2,12,4,42
                    db 112,0
z2                  db 'Priezvisko:'
zz2                 db 31 dup (32),0
zona3               dw  offset zz3
                    dw offset z3,14,11,8
                    db 112,1
z3                  db 'Vek:'
zz3                 db 4 dup (32),0
zona                dw 0
text0               db „ma „
text1               db „rokov.  „
text2               db „rok. „
text3               db „roky. „
ip:                 mov ax,cs
                    mov ds,ax
                    mov ax,0b800h
                    mov es,ax
                    mov ah,01h
                    mov cx,20h*256+00h
                    int 10h
                    call cls
                    mov bp, offset zona1
                    call input
                    mov bp, offset zona2
                    call input
                    mov bp,offset zona3
                    call input
                    mov si,offset zz1
                    xor di,di
                    call vypis
                    mov si,offset zz2
                    call vypis
                    mov si,offset text0
                    call vypis
                    mov si,offset zz3
                    call vypis
                    mov si,offset zz3
                    mov al,[si+1]
                    cmp al,32
                    jz p0
                    mov si,offset text1
                    call vypis
                    jmp short endx
p0: mov al,[si]
                    cmp al,49
                    jz p1
                    cmp al,32h
                    jge p5
                    jmp short p2
p5:                 cmp al,34h
                    jna p2
                    mov al,[si]
                    cmp al,35h
                    jge p4
p4:                 cmp al,39h
                    jna p3
p1:                 mov si,offset text2
                    call vypis
                    jmp short endx
p2:                 mov si,offset text3
                    call vypis
                    jmp short endx
p3:                 mov si,offset text1
                    call vypis
                    jmp short endx
endx:               mov ax,0
                    int 16h
                    mov ah,01h
                    mov cx,12h*256+14h
                    int 10h
                    mov ax,4C00h
                    int 21h
input               proc
                    mov si,[bp+0]
                    mov [zona],si
                    mov al,'_'
                    mov [si+0],al
ip3:                call zobraz
                    call zvuk
ip10:               mov ax,0
                    int 16h
                    cmp al,13
                    jz koniec
                    cmp al,8
                    jz delete
                    mov dl,[bp+11]
                    cmp dl,0
                    jz ip5
                    cmp al,30h
                    jb ip3
                    cmp al,39h
                    ja ip3
                    jmp short ip6
ip5:                cmp al,32
                    jz ip10
                    cmp al,33
                    jb ip3
                    cmp al,255
                    ja ip10
ip6:                mov [si+0],al
                    inc si
                    mov al,[si+0]
                    cmp al,0
                    jz kon_ed
                    mov al,'_'
                    mov [si+0],al
                    jmp short ip3
kon_ed:             dec si
ip9:                mov al,'?'
                    mov [si+0],al
                    jmp short ip3
delete:             cmp si,[zona]
                    jz ip8
```

```
                mov ax,'_'*256+32
                mov [si+0],al
                dec si
                mov [si+0],ah
ip8:            jmp ip3
koniec:         mov al,32
                mov [si+0],al
                call zobraz
ret             input
                endp
zobraz          proc
                push si
                mov ax,0b800h
                mov es,ax
                mov ax,[bp+4]
                mov bx,[bp+6]
                mov dl,80
                mul dl
                add ax,bx
                mov dx,2
                mul dx
                xchg ax,di
                mov si,[bp+2]
                mov
                cx,[bp+8]
ip2:            lodsb
                mov ah,[bp+10]
                stosw
                loop ip2
                pop si
                ret
zobraz          endp
zvuk            proc
                in al,61h
                or al,3
                out 61h,al
                mov al,0b6h
                out 43h,al
                xor al,al
                out 42h,al
                inc al
                out 42h,al
                mov ax,3000
ipx:            dec ax
                jnz ipx
                in al,61h
                and al,0fch
                out 61h,al
                ret
zvuk            endp
cls             proc near
                xor di,di
                mov ax,7*256+32
                mov cx,80*25
                rep stosw
                ret
cls             endp
vypis           proc near
                mov al,2*16+1
cykl:           movsb
                stosb
                push si
                dec si
                mov ah,[si]
                cmp ah,32
                pop si
                jnz cykl
                ret
vypis           endp
                end start
```

Vysvetlenie k programu vstup2.asm

Program sa skladá z piatich procedúr (INPUT, ZOBRAZ, ZVUK, CLS, VYPIS). Procedúra CLS slúži na zmazanie obrazovky, procedúra ZVUK vygeneruje krátke cvaknutie pri stlačení klávesu. Procedúra VYPIS zobrazí na obrazovke text, procedúra ZOBRAZ sa používa na obnovovanie vstupu, t. j. ak stlačíte nejaký kláves, musí sa okamžite vykresliť buffer vstupu, pretože inak by sa vám na obrazovke nezobrazil predtým stlačený kláves. Procedúra INPUT je zo všetkých doposiaľ vymenovaných procedúr najdôležitejšia a zabezpečuje samotný vstup. Preberieme si ju podrobnejšie. Vstupné parametre procedúry sú: register BP obsahuje offset adresy, na ktorej sú uložené parametre, podľa ktorých sa procedúra INPUT nastaví. Podrobne to dokumentuje obrázok 3.

Opis procedúry INPUT: do registra SI začiatok editačnej zóny, túto adresu si uložíme pre neskoršie použitie, v registri AL je uložený tvar kurzora - znak „_“. Teraz do editačnej zóny zapíšeme tvar kurzora a vykreslíme editačnú zónu na obrazovku (pozri procedúru ZOBRAZ). Krátke pípnutie. Nasleduje sekvencia MOV AX,0 a INT 16h — táto služba prečíta znak z klávesnice a do registra AL uloží ASCII kód, do registra AH SCAN kód stlačeného klávesu. Otestujeme, či nebol stlačený kláves ENTER, BACKSPACE. Ak bol stlačený niektorý z klávesov, pokračuje sa vo vykonávaní príslušnej obslužnej rutiny. Pri inštrukcii MOV DL,[BP+11] sa rozhoduje o tom, či bude vstup znakový alebo číselný. Ak ide o znakový vstup, potom skok na návestie IP5. Inak uprav rozsah na vstup čísel.



Obr. 3 Vstupné parametre pre procedúru INPUT

Od návestia IP5 sa testuje, či stlačený kláves patrí do intervalu <33, 255>, ak áno, pokračuje sa vykonaním inštrukcie na návestí IP6. Do editačnej zóny sa zapíše ASCII kód stlačeného klávesu a kurzor sa posunie doprava. Do registra AL ulož obsah tohto pamäťového miesta. Je to nula? Áno, potom sme ma konci editačnej zóny - zobraz „?“ pre upozornenie. Ak nie sme na konci zóny, potom musíme za znak stlačeného klávesu vytlačiť ešte tvar kurzora. Ďalej je skok na návestie IP3, čo znamená, že program znova čaká na ďalšie stlačenie klávesu. Od návestia KON_ED: začína sa krátky program, ktorý testuje, či je kurzor na konci editačnej zóny. Na návestie DELETE sa dostanete vo chvíli, keď stlačíte kláves BACKSPACE. Ak je kláves stlačený, zmaže sa znak pred kurzorom a kurzor sa posunie na jeho miesto. Na návestie KONIEC sa dostanete, ak stlačíte kláves ENTER. Zmaže sa tvar kurzora a znovu sa prekreslí editačná zóna. To je vyčerpávajúci opis procedúry INPUT.

Ak si s programom začnete tykať, môžete skúsiť dorobiť napr.: pohyb po editačnej zóne a s tým súvisiace vkladanie znakov, skok na koniec alebo začiatok zóny (klávesy END a HOME) a iné.

Ešte som sa nezmienil, čo vlastne program **vstup2.asm** robí. Po spustení si od vás pýta postupne meno, priezvisko a vek. Tieto tri údaje spracuje a vypíše ich do ľavého horného rohu obrazovky v tvare napríklad: **Fero Hlavička má 1 rok**, pričom sa testuje, či zadané číslo je z niektorého z rozsahov (1, 2 až 4, 5 až), podľa toho upraví slovo „rok“ na (rok, roky, alebo na rokov). Ešte si všimnite, že pri vstupe Vek nemôžete zadať iné ako čísla. To by bolo nadnes všetko, pokračujeme nabudúce.

Literatúra

[1] M. Brandejs: MS-DOS 6 - kompletní průvodce. Grada 1993.
[2] V. Boukal: BIOS IBM PC. Grada 1992.

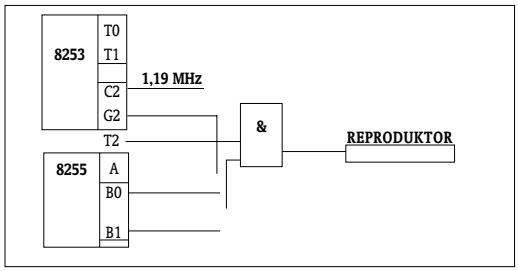
Vysvetlivky

1 **HOT KEY** – kláves priameho výberu funkcie, tzv. horúci kláves.
2 **REGISTRE** – toto okno je obsiahnuté v CPU

Desiata časť: Klávesnica, zvuk...

Zvukový výstup

Zvukový výstup sa v počítačoch využíva napríklad na indikáciu chyby, ďalej v hrách, ale aj v profesionálnych programoch. Generovanie tónov býva v počítačoch PC riadené špecializovaným obvodom (zvuková karta) alebo pomocou stykových obvodov 8253 a 8255. Druhý spôsob riadenie ja častejší. Akustický tón je charakterizovaný výškou, farbou a intenzitou. Výška tónu závisí od základnej frekvencie pravouhlého signálu. Počuteľné zvuky sú od 20 Hz do 20 kHz, ale na hudobné účely sa využívajú tóny s frekvenciami stoviek a tisícok Hz. Farba tónu závisí od obsahu jeho frekvenčných zložiek. Pravouhlý signál má konštantné a pomerne vysoké zastúpenie vyšších frekvencií. Preto pri použití pravouhlého signálu je farba zvuku charakteristická a je možné meniť ju len veľmi ťažko. Vzhľadom na konštantnú úroveň logického signálu je aj jeho intenzita konštantná. Vďaka nedokonalým vlastnostiam reproduktora (rezonancie, útlmy) sa môže hlasitosť pri rôznych frekvenciách meniť. Stykový obvod 8253 generuje určenú frekvenciu a dva bity B0 a B1 obvodu 8255 sú využité na riadenie vzniku tónu. Počítadlo T2 má na vstupe frekvenciu 1,19 MHz, ktorú v režime č. 3 delí vhodným číslom. Tak vzniká na výstupe T2 akustický tón danej výšky, určitej farby a určitej intenzity, ktorý sa prevedie do zvukovej podoby v reproduktore.



Obr. 1 Principiálne zapojenie generátora tónov

Stykový obvod 8255 svojím výstupom B0 hradluje činnosť počítadla T2 a výstupom B1 hradluje výstup generovaného tónu do reproduktora. Zvuk sa generuje vtedy, keď na T2 je počuteľná frekvencia a zároveň B0 aj B1 je v stave log. 1. Programátor musí v programe vhodne nastaviť pracovný režim a deliaci pomer počítadla 8253. To sa dosiahne riadiacim slovom a veľkosťou predvoľby (deliaceho pomeru). Riadiace slovo je 8-bitové. Jednotlivé bity majú význam:

7 až **6** → číslo počítadla, ktoré sa má riadiť

5 až **4** → počet slabík podľa dĺžky deliaceho pomeru:

- 01 – len nižšia slabika (vyššia je nulová)
- 10 – len vyššia slabika (nižšia je nulová)
- 11 – obidve slabiky, najprv nižšia a potom vyššia

3 až **1** → režim činnosti počítadla, režim č. 3 delí vstupnú frekvenciu, pričom sa generuje obdĺžnikový signál

0 → či bude vložený deliaci pomer v tvare BCD kódu (inak binárne)

Stykový obvod 8255 je možné využiť na zapínanie a vypínanie tónu. Tón sa generuje, ak B0 aj B1 sú v stave log. 1. Ovládaním stavu týchto bitov je možné ovládať zvukový výstup. Na ostatné bity portu B obvodu 8255 sú pripojené iné obvody, preto ich nie je možné ľubovoľne meniť. Programátor zapisuje celú slabiku do portu B, a preto treba zabezpečiť, aby ostatné bity ostali v pôvodnom stave. To sa dá urobiť tak, že program najprv prečíta stav portu B, upraví podľa potreby bity B0 a B1 a slabiku vráti späť do portu B. Výstupný port obvodu 8255 je totiž možné čítať, pričom sa prečíta okamžitý stav výstupov portu.

Adresy registrov obvodov 8253 a 8255 sú takéto:

– riadiaci register 8253: 043H
– počítadlo T2: 042H
– riadiaci register 8255: 063H
– výstupný port B: 061H

Riadenie generovania zvuku si ukážeme na príklade tónu 440 Hz. Vhodný deliaci pomer je 1,19 MHz : 440 Hz = 2704,54, zaokrúhlene 2705. Nastavenie generovania tohto tónu dosiahneme takýmto programom v jazyku assembler:

```
MOV AL,10110111B; počítadlo č. 2, obidve slabiky, režim č. 3
OUT 43H,AL; výstup do riadiaceho registra
MOV AL,05; nižšia slabika deliaceho pomeru
OUT 42H,AL; výstup do T2
MOV AL,27H; vyššia slabika
OUT 42H,AL; výstup do T2
```

Povolenie vydať tón 440 Hz dosiahneme programom, v ktorom nastavíme B0=B1=1:

```
IN AL,61H ;čítanie stavu portu B
OR AL,00000011B ;nastavenie B0=B1=1
OUT 61H,AL ;vrátenie hodnoty do portu B
```

Ak sa tón nemá generovať, môžeme to dosiahnuť nastavením B0=0 krátkym programom:

```
IN AL,61H; vstup stavu portu B
AND AL,0FEH; nulovanie bitu B0
OUT 61H,AL; výstup do portu B
```

Porty A, B aj C obvodu 8255 sa nastavujú systémovými programami do správneho režimu činnosti (A vstupný, B výstupný, C vstupný). Preto programátor nemusí osobitne nastavovať port B do výstupného režimu (tým skôr, že riadiacim slovom sa nastavujú režimy aj portov A a C). Pre úplnosť však uvádzame, že sa to dosiahne zavedením riadiaceho slova 10011001B na adresu 63H riadiaceho registra obvodu 8255. Okrem uvedeného spôsobu generovania zvuku je možné využiť aj programové generovanie periódy tak, že výstup T2 bude stále v stave 1 a programom sa dosiahne kmitanie výstupu B1, a tým aj reproduktora. Generovanie polperiódy je možné dosiahnuť napr. čakacou slučkou programu, čím sa však zaťažuje procesor. Zmenu stavu bitu B1 dosiahneme programom:

```
IN AL,61H; vstup stavu portu B
XOR AL,00000010B; B1 do opačného stavu
OUT 61H,AL; výstup do portu B
```

Trvalý stav T2=1 dosiahneme zavedením vhodného riadiaceho slova do riadiaceho registra obvodu 8253 bez vloženia deliaceho pomeru. Napríklad pri zadaní režimu č. 0 prejde výstup T2 do stavu log. 1 a takto čaká na vloženie predvoľby (ktorú nezadáme). Kombináciou generovania frekvencie obvodom 8253 a prerušovaním tónu bitmi B0 alebo B1 je možné dosiahnuť rozličné zvukové efekty. V nasledujúcej tabuľke sú uvedené frekvencie jednej oktávy tónov pri temperovanom chromatickom ladení. Frekvencie tónov iných oktáv možno získať vhodným delením alebo násobením (napr. susedná vyššia oktáva má tóny dvojnásobnej frekvencie a susedná nižšia tóny polovičnej frekvencie).

Tabuľka frekvencií tónov 4. oktávy

C4 - 261,63 Hz	C#4 - 277,18 Hz	D4 - 293,66 Hz
D#4 - 311,13 Hz	E4 - 329,63 Hz	F4 - 349,23 Hz
F#4 - 369,99 Hz	G4 - 392,00 Hz	G#4 - 415,30 Hz
A4 - 440,00 Hz	A#4 - 466,16 Hz	B4 - 493,88 Hz

Na záver si ešte ukážeme program, ktorý vygeneruje krátke pípnutie.

```
;beep.asm
.model small
.stack 100h
.code
beep      proc  near
                  mov  al,10110110b
                  out 43,al
                  mov ax,bx
                  out 42,al
                  mov al,ah
                  out 42,al
                  in al,61h
                  or al,3
                  out 61h,al
                  ret
beep              endp
start:            in al,61h
                  push ax
;ferkvencia v Hz
                  mov bx,4056
                  call beep
;Nasledujuca sluzba pocka zadany
pocet milisekund, nez vrati
riadenie volacej rutine
                  mov cx,8
                  mov dx,10240
                  mov ah,86h
                  int 15h
                  pop ax
                  out 61h,al
                  mov ax,4c00h
                  int 21h
                  end start
```

Klávesnica

Úlohou klávesnicového vstupu je umožniť vstup informácií od používateľa do programu pri pomerne vysokých nárokoch na pohodlie práce a mnohofunkčnosť klávesnice. Mechanicky je klávesnica oddelená od počítača a spojená s ním jednoduchým štvorvodičovým káblom, ukončeným jednoduchým konektorom DIN. Komunikácia medzi klávesnicou a počítačom je v sériovom tvare, pričom sériový kód nie je ASCII, ale je to osobitný kód indikujúci stlačený kláves (každému klávesu je priradený vlastný kód podľa polohy na klávesnici). Kód je nezávislý od významu klávesu. Preklad z tohto kódu do kódu ASCII zabezpečuje ovládací program v BIOS-e. Pri stlačení niektorého klávesu generuje klávesnica jej kód do počítača. Pri pustení klávesu sa tiež generuje kód, takže program v počítači vie presne zistiť stav stlačenia aj viacerých klávesov súčasne. Kód pri pustení je o 128 vyšší ako pri stlačení, pretože kód pri stlačení má 7. bit v stave log. 0 a pustení klávesu má 7. bit nastavený na 1. Po stlačení klávesu sa generuje sériový kód do počítača, kde sa vyvolá prerušenie. Pri jeho obsluhe a prevzatí kódu sa programom zistí prislúchajúci kód ASCII (alebo kód funkčných klávesov) a ten sa zaradí do frontu. Osobitná služba v BIOS-e, vyvolaná aplikačným programom, vyberá kódy z tohto frontu a odovzdáva ich do aktívneho aplikačného programu.

Umiestnenie vodičov sériového spojenia na konektore DIN uvádza tabuľka:

Vývod	Signál
1	synchronizácia RQ/CLK
2	sériový kód
3	RESET (nevyužíva sa v klávesnici)
4	GND
5	napájanie +5V

Klávesnica obsahuje pole písmen, špeciálnych znakov, funkčných klávesov a aplikačne definovaných tzv. kľúčov (F1 až F10) s charakteristickým rozložením. Okrem štandardnej klávesnice existujú aj tzv. národné modifikácie s niektorými osobitnými klávesmi a s osobitným rozložením klávesov.

Riadiaci program vstupu z klávesnice po prijatí identifikačného kódu stlačeného klávesu určí jeho hodnotu ASCII, prípadne kód funkčného klávesu. Vykoná sa teda preklad z identifikačného kódu do kódu ASCII alebo do kódu funkčného klávesu. Vzniknú pritom dve slabiky, ktoré sa zaradia do frontu na adrese 0:041EH (ide o kruhový front šestnástich 2-baj-

tových položiek, pričom na adrese 0:41AH je ukazovateľ začiatku a na adrese 0:41CH je ukazovateľ konca frontu). Z neho vyberá služba obsluhy klávesnice volané inštrukciou INT 16H. Obslužný program klávesnice INT 16H vie vykonať tri služby. Kód služby sa umiestňuje do registra AH z rozsahu 0-2. Význam kódov uvádza nasledujúca tabuľka:

Služba	Význam
0	vstup z klávesnice
1	test stavu frontu
2	test stavu modifikačných klávesa

Služba č. 0 poskytuje vstupný údaj z klávesnice. Služba vyberie z frontu dve slabiky pripravené na vstup do programu alebo čaká na prijatie znaku, ak je front prázdny. Vstupný údaj zanecháva služba v dvojici registrov AH a AL. Význam týchto slabík je odlišný pre zobraziteľné kódy ASCII a pre funkčné klávesy.
Pre kódy ASCII je kódovanie takéto:
AL = kód ASCII znaku
AH = identifikačný kód klávesu z klávesnice, kód Scan
Pre funkčné klávesy je kódovanie takéto:
AL = 0
AH = identifikačný kód pri jednom stlačenom klávese alebo funkčný kód pri kombinácii klávesov.

Klávesnica sa používa aj tak, že sa stláčajú dve klávesy súčasne, napr. niektorý znak a modifikačné klávesy SHIFT, CTRL alebo ALT.

Ak sa stlačí kombinácia SHIFT a niektorý zobraziteľný znak, má to význam zmeny malé-veľké písmená. Ide teda opäť o zobraziteľný znak a zmena sa prejaví v hodnote kódu ASCII v registri AL (v registri AH je identifikačný kód klávesu). Rozdiel kódov ASCII pre malé a veľké písmená je 32 (malé písmená majú kódy väčšie o 32). Rozdiel kódov klávesov so špeciálnymi znakmi v kombinácii so SHIFT vyplýva z popisu daného klávesu, napr. kláves s identifikačným kódom 51 dáva kód ASCII znaku „," a pri stlačenom SHIFT kód znaku „<". Kombinácia SHIFT s ďalšími (funkčnými) klávesmi sa chápe ako funkčná kombinácia. Preto pri odovzdávaní hodnoty je v AL nula a v AH hodnota podľa tabuľky:

Kód	Kláves
84	Shift + F1
85	Shift + F2
86	Shift + F3
87	Shift + F4
88	Shift + F5
89	Shift + F6
90	Shift + F7
91	Shift + F8
92	Shift + F9
93	Shift + F10

Kombinácie Shift s ostatnými klávesmi nemenia hodnotu v registri AH (ako keby sa Shift nepoužil). Kombinácia klávesu CTRL a niektorého iného klávesu so zobraziteľným znakom vytvára kód riadiaceho znaku z rozsahu 0 - 1FH, čo sa tiež považuje za kód ASCII. Ide o „znaky" s kódmi 0 - 31. Tie sa v prídavných zariadeniach zvyčajne využívajú ako riadiace povely. Kombinácia CTRL a niektorých iných špeciálnych klávesov sa považuje za vstup funkčného klávesu. Preto v registri AL je pri ukončení služby č. 0 nulový obsah a v registri AH je identifikačný kód kombinácie. Niektoré identifikačné kódy takýchto používaných kombinácií klávesov uvádza tabuľka:

Kód	Kláves	Kód	Kláves
94	Ctrl + F1	137	Ctrl + F11
95	Ctrl + F2	138	Ctrl + F12
96	Ctrl + F3	114	Ctrl + Print Screen
97	Ctrl + F4	115	Ctrl + Vľavo
98	Ctrl + F5	116	Ctrl + Vpravo
99	Ctrl + F6	117	Ctrl + End
100	Ctrl + F7	118	Ctrl + Page Down
101	Ctrl + F8	119	Ctrl + Home
102	Ctrl + F9	132	Ctrl + Page Up
103	Ctrl + F10	148	Ctrl + Tab

Kombinácia klávesu ALT s niektorým klávesom sa tiež považuje za vstup funkčného klávesu, a preto sa v registri AL odovzdá nula a v registri AH kód funkčnej kombinácie. Kombinácia ALT s niektorým písmenovým klávesom A až Z alebo s klávesom špeciálneho znaku, napr. ,#,$, ^, dáva kód zhodný s kódom písmenového klávesu(t. j. identifikačný kód klávesu sa použije bez zmeny). Kombinácia ALT s ďalšími klávesmi dáva kód pod a tabuľky:

Kód	Kláves	Kód	Kláves
104	Alt + F1	121	Alt + 2
105	Alt + F2	122	Alt + 3
106	Alt + F3	123	Alt + 4
107	Alt + F4	124	Alt + 5
108	Alt + F5	125	Alt + 6
109	Alt + F6	126	Alt + 7
110	Alt + F7	127	Alt + 8
111	Alt + F8	128	Alt + 9
112	Alt + F9	129	Alt + 0
113	Alt +F10	130	Alt + -
120	Alt +1	131	Alt + =

Ďalšou možnosťou vstupu zobraziteľného kódu z klávesnice je zadanie desiatkovej hodnoty znaku ASCII z klávesnice. To sa robí podržaním klávesu ALT a zadaním desiatkového kódu žiadaného znaku ASCII. Po vložení čísla sa kláves ALT pustí, čo je znamením pre BIOS, že zadávanie čísla je ukončené a podľa vloženého kódu má do frontu generovať kód ASCII. V takomto prípade služba BIOS odovzdá v registri AL zadaný kód ASCII a v registri AH nulu (tak je možné tento spôsob odlíšiť od riadneho stlačenia klávesu ASCII). Takto však nie je možné zadať kód ASCII 00H (t. j. znak NULL), pretože výstupná kombinácia slabík vo fronte by zodpovedala niektorému funkčnému klávesu (keď je nižšia slabika nulová).

Služba č.1 poskytuje informáciu o tom, či je front prázdny, alebo nie. Výstupná informácia je v príznaku ZF kódovaná takto:
ZF = 0 front obsahuje aspoň jeden údaj
ZF = 1 front je prázdny

Okrem toho ako výstupný parameter je v dvojici AH a AL kópia údaja, ktorý je na rade na výber z frontu (ak nie je prázdny). Tento údaj sa teda z frontu vyberie (poskytne) pri najbližšej službe č. 0 . Treba však poznamenať, že pri službe č. 1 je v AX skutočne iba kópia údaja z frontu, front sa touto službou neskracuje (t. j. údaj z neho neodíde). Osobitnú pozornosť si zasluhujú modifikujúce klávesy, ako napr. SHIFT, CTRL, ALT. Ak sú stlačené, mení sa význam iných stlačených klávesov, ako sme to už uviedli. Toto rozlíšenie robí riadiaci program v BIOS-e, lebo klávesnica len zasiela stav klávesov do počítača. Teda ak BIOS prijal kód, napr. „stlačený SHIFT", interpretuje inak nasledujúce klávesy až do prijatia kódu „pustený SHIFT".

Ďalší modifikujúci význam majú klávesy CAPS LOCK, NUM LOCK a INS. Stlačením CAPS LOCK sa opačne chápe účinok klávesu SHIFT (stlačená – malé písmená, nestlačená - veľké písmená). Rozdiel je však v tom, že kláves CAPS LOCK nie je trvale stlačený, ale jeho účinok sa začne „klepnutím" na kláves CAPS LOCK. Je to teda pamäťové správanie na rozdiel od klávesov SHIFT, CTRL a ALT. Aj kláves NUM LOCK má pamäťové správanie a mení význam numerických klávesov v pravej časti klávesnice. Buď majú význam číslic, alebo funkčných klávesov (posuv kurzora, Home, End, Page Up, Page Down, Delet, Insert). Kláves Ins môže mať rozličný význam podľa aplikačného programu, ale najčastejšie má tiež pamäťové správanie a mení sa ním napr. v editoroch textu stav (mód) INSERT (aktívny, neaktívny). Pre všetky tieto klávesy platí, že je rozhodujúce, či boli stlačené párny alebo nepárny počet raz a podľa toho sa modifikuje účinok iných klávesov.

Programy BIOS na obsluhu klávesnice si preto pamätajú stav modifikujúcich klávesov. Informáciu o aktuálnom stave poskytuje služba č. 2, vyvolaná prerušením INT 16H. Jej výstupom je slabika v registri AL. Jej jednotlivé bity majú význam podľa tabuľky:

Bit	Stav	Význam
0	aktívny	PRAVÝ SHIFT
1	aktívny	ĽAVÝ SHIFT
2	aktívne	CTRL
3	aktívne	ALT
4	aktívny	SCROLL LOCK
5	aktívny	NUM LOCK
6	aktívny	CAPS LOCK
7	aktívny	INSERT

Služby BIOS si pamätajú túto slabiku na adrese 0000:0417H a z nej vychádzajú pri riadení činnosti služby č.0. Programátor môže zmeniť obsah tejto slabiky, čím sa zmení interpretácia stláčaných klávesov v budúcnosti bez toho, aby na to dal pokyn používateľ z klávesnice. Napríklad nastavením 6. bitu do stavu log. 1 sa dosiahne stav, akoby bol stlačený kláves CAPS LOCK. Podobne na adrese 0000:0418H si BIOS ukladá hodnotu slabiky s nasledujúcim významom:

Bit	Význam
0	je stlačený PRAVÝ SHIFT
1	je stlačený ĽAVÝ SHIFT
2	je stlačený SYSREQ
3	je aktívny stav „Hold State" po stlačení PAUSE
4	je stlačený SCROLL LOCK
5	je stlačený NUM LOCK
6	je stlačený CAPS LOCK
7	je stlačený INS

Riadiaci program klávesnice BIOS okrem uvedených funkcií osobitným spôsobom interpretuje štyri vyhradené kombinácie klávesov. Ide o tieto kombinácie a ich interpretáciu:

CTRL+ALT+DEL
Operačný systém sa privedie do stavu počiatku ako pri zapnutí počítača. Neprebehnú testovacie programy, ale vykoná sa zavedenie operačného systému z diskového média a jeho spustenie (na adrese 0:472H je v dvoch slabikách kódom 1234H signalizované, že prebieha táto funkcia).

SHIFT+Print Screen
Prenesie sa obsah obrazovky cez paralelný kanál do pripojenej tlačiarne. Predpokladá sa tlačiareň kompatibilná s grafickou tlačiarňou IBM. Ak je zobrazovací adaptér v grafickom móde, prenesie sa informácia do tlačiarne v grafickom móde spôsobom „bod po bode“. Ak je zobrazovací adaptér v znakovom móde, prenesie sa informácia v tvare kódov ASCII.

CTRL+NUM LOCK
Riadiaci program negeneruje do frontu znakov nijaký výstup a neodovzdá riadenie do prerušeného programu (ktorý bol prerušený pri obsluhe žiadosti o prerušenie z klávesnice). V tomto stave čaká riadiaci program BIOS až do príchodu ďalšieho znaku z klávesnice. To sa prakticky prejaví zastavením činnosti bežiaceho programu. Používateľ preto môže túto kombináciu využívať na pozastavenie programu (napr. pri výpise na obrazovku). Ostatné prerušenia nie sú blokované, a preto ich obsluha beží normálne ďalej.

CTRL+BREAK
Riadiaci program BIOS vyvolá programové prerušenie INT 27 (1BH). Ním sa aktivuje program, ktorého adresa vstupného bodu má byť uložená na adrese 0:006CH v tvare 4-slabikovej dlhej adresy. Programátor si môže tento program zostaviť, uložiť do pamäte a jeho adresu vložiť na spomenutú adresu 006CH. Tento program sa bude aktivovať pri každom stlačení kombinácie CTRL+BREAK. Ak to programátor neurobí, ostane tam adresa programu, ktorý vráti riadenie späť do obslužného programu klávesnice (stlačenie CTRL+BREAK sa teda navonok neprejaví). Vždy však riadiaci program poznačí log. 1 v 7. bite slabiky na adrese 0:471H, že boli stlačené CTRL+BREAK. Informácia o tom, že boli stlačené CTRL+BREAK, sa uplatní až pri najbližšom volaní niektorej služby operačného systému DOS (ak je použitý). Ak operačný systém má nastavený parameter BREAK ON, ukončí sa činnosť programu a odovzdá sa riadenie operačnému systému (programu COMMAND.COM). Ak bol parameter BREAK OFF, stlačenie kombinácie CTRL+BREAK nemá nijaký účinok. Ak teda programátor neurobí nijaké zásahy, stlačením CTRL+BREAK sa ukončí program a riadenie prejde do operačného systému pri najbližšom použití niektorej služby operačného systému.

Nasledujúci program po spustení vás vyzve na stlačenie klávesu a po jeho stlačení vypíše na obrazovku jej kód SCAN a kód ASCII.

```
;test.asm
.model small
.stack 100h
.data
cislo               db 0,0
text1               db „Stlac klaves !$“
text2               db „Stlaceny klaves „
                    db „ma Scan kod: $“
text3               db „Press any key „
                    db „for return to DOS.“
text4               db „ASCII kod: $“
.code
;——————————
;Vypne zobrazovanie kurzora
;v textovom rezime.
;——————————
kurzor_off          proc near
                    mov ah,01h
                    mov cx,20h*256+00h
                    int 10h
                    ret
kurzor_off          endp
;——————————
;Zapne zobrazovanie kurzora
;v textovom rezime.
;——————————
kurzor_on           proc near
                    mov ah,01h
                    mov cx,12h*256+14h
                    int 10h
                    ret
kurzor_on           endp
position            proc  near
                    mov   bh,0
                    mov   ah,02
                    int   10h
                    ret
position            endp
transfer            proc near
                    push    ax
                    xor     cx, cx
wn0:                xor     dx, dx
                    div     bx
                    push    dx
                    inc     cx
                    test    ax, ax
                    jnz     wn0
wn2:                pop     dx
                    or      dl, '0'
                    cmp     dl, '9'
                    jbe     wn3
                    add     dl, 7
wn3:                mov     ah, 2
                    int     21h
                    loop    wn2
                    pop    ax
                    ret
                    transfer  endp
print               proc near
cykl:               movsb
                    stosb
                    loop cykl
                    ret
print               endp
address             proc near
                    mov dl,80
                    mul dl
                    add ax,bx
                    mov dx,2
                    mul dx
                    ret
address             endp
start:              mov ax,@data
                    mov ds,ax
                    mov ax,0b800h
                    mov es,ax
                    mov ax,3
                    int 10h
                    call kurzor_off
                    mov dx,10*256+17
                    call position
                    lea   dx,text1
                    mov   ah,09h
                    int   21h
                    mov ax,11h
                    int 16h
                    push ax
                    mov dx,12*256+10
                    call position
                    lea   dx,text2
                    mov   ah,09h
                    int   21h
                    pop ax
                    mov si,offset cislo
                    mov [si+0],ax
                    mov al,[si+1]
                    xor ah,ah
                    mov bx,10
                    call transfer
                    mov dx,14*256+28
                    call position
                    lea dx,text4
                    mov ah,9
                    int 21h
                    mov al,[si+0]
                    xor ah,ah
                    mov bx,10
                    call transfer
                    mov ax,16
;Riadok.             mov bx,10
;Stlpec              call address
                    mov di,ax
                    mov si,offset text3
                    mov cx,32
;Dlzka textu         mov al,7+128;Farba
                    call print
                    call kurzor_off
                    mov ax,0
                    int 16h
                    call kurzor_on
                    mov ax,4c00h
                    int 21h
                    end start
```

Literatúra
[1] ABSHelp verzia 2.05. ABSoft Olomouc.
[2] V. Boukal: BIOS IBM PC. Grada 1992.

Jedenásta časť: Klávesnica AT

Štandardná klávesnica XT má 83 klávesov. Niektoré klávesnice počítačov AT majú 84 klávesov, pričom pribudol kláves SysReq. Jeho identifikačný kód je 84. Jeho použitie je dané aplikačnými programami. V operačnom systéme MS DOS nemá osobitný význam, v operačnom systéme UNIX sa využíva na volanie operačného systému počas chodu aplikačného programu (podobne ako CTRL-BREAK). Najčastejšie má však klávesnica počítača AT 102 klávesov (niekedy 101). Pribudli klávesy F11, F12, zdvojené sú CTRL a ALT, šestica klávesov INSERT, HOME, PAGE UP, DELETE, END, PAGE DOWN, štvorica klávesov J, L, I, < (uvedená šestica a štvorica sú teda zdvojené klávesy, lebo také klávesy existovali aj v modeli XT v rámci numerickej klávesnice) a zdvojené sú aj /, *, -, +, Enter (sú umiestnené okolo numerickej klávesnice). Zdvojené klávesy síce majú vlastné kódy na úrovni klávesnice a sériového prenosu do počítača, no nevytvárajú nové identifikačné kódy pre programy, lebo adaptér klávesnice s obvodom 8042 transformuje prijaté kódy na zaužívané kódy z modelu XT (ide len o zdvojenie pre lepšiu manipuláciu). Ich identifikačné kódy sú preto zhodné s kódmi rovnakých klávesov v inej časti klávesnice.

Okrem toho klávesnica má indikačné LED diódy na indikovanie okamžitého stavu Num Lock, Caps Lock a Scroll Lock. Technické riešenie je tiež iné v súlade s technickou koncepciou modelu AT – využívajú sa integrované obvody VLSI. Preto klávesnica je riadená jednočipovým mikropočítačom 8049 a v počítači komunikuje s klávesnicou jednočipový mikropočítač 8042. Tým sa stal podsystém klávesnice programovateľným, napr. je možné meniť oneskorenie a frekvenciu opakovania kódov pri trvale stlačenom klávese, je možné vysielať do klávesnice povely na testovanie klávesnice, rozsvecovanie LED diód atď. Okrem toho mikropočítač 8049 poskytuje vyrovnávaciu pamäť na 16 slabík, ktoré sa z rôznych dôvodov nemohli dostať do 8042 v počítači (napr. BIOS si neprevzal z 8042 slabiku, a preto 8042 zastavil vysielanie z klávesnice). Preplnenie tejto pamäte sa signalizuje osobitným kódom z 8049 do 8042.

Teraz si ukážeme dva programy, pomocou ktorých budeme ovládať svietiace LED diódy na klávesnici. Prvý program ich rozsvieti a druhý ich zhasne.

```
;led_on.asm
.model small
.stack 100h
.code
start:    xor ax,ax
          mov ds,ax
          mov bx,417h
          or byte ptr [bx],01110000b
          mov ax,4c00h
          int 21h
          end start
;led_off.asm
.model small
.stack 100h
.code
start:    xor ax,ax
          mov ds,ax
          mov bx,417h
          and byte ptr [bx],10001111b
          mov ax,4c00h
          int 21h
          end start
```

Riadiaci mikropočítač klávesnice 8049 má zabudovaný vlastný test, ktorý sa spustí pri zapnutí napájania alebo po povele RESET z 8042. Pri ňom sa testuje obsah pamäte ROM, pamäť RWM a rozsvietia a zhasnú sa LED diódy, čo je pozorovateľné. Komunikácia medzi oboma riadiacimi mikropočítačmi 8042 a 8049 sa uskutočňuje cez vodiče DATA a RQ/CLK ako v modeli XT. Komunikácia je však obojsmerná a zložitejšia. Obidva signály sú budené obvodmi s otvoreným kolektorom, takže ktorákoľvek strana ich môže ovládať.

Kolízie vysielaní z oboch strán rieši klávesnica, ktorá, ak zistí vysielanie RQ/CLK=0 z 8042, preruší svoje vysielanie, dáta si uschová a prejde na príjem z 8042. Potom neukončený prenos slabiky zopakuje. V prípade zistenia chyby prenosu na niektorej strane (neznámy kód alebo chyba parity) sa do druhej strany vysiela požiadavka na opakovanie prenosu. Jednočipový mikropočítač 8042 v počítači AT tvorí adaptér klávesnice na jej riadenie a zabezpečuje aj iné úlohy v systéme počítača. Z hľadiska systému sú v 8042 dôležité tieto komunikačné jednotky:

- výstupný komunikačný register na čítanie programom v AT
- vstupný komunikačný register na zápis programom v AT
- stavový register na čítanie programom
- signálový výstup pre klávesnicu (DATA, RQ/CLK) a pre procesorovú jednotku (systémový reset, žiadanie prerušenia pre obsluhu 8042)
- signálový vstup pre klávesnicu (DATA, RQ/CLK, vypínač klávesnice) a pre procesorovú jednotku (konfigurácia pamäte, typ displeja ...).

Riadiaci program mikropočítača riadi komunikáciu medzi klávesnicou a riadiacim programom BIOS. Riadenie spočíva v tom, že konvertuje informačné prenosy do žiadaného kódu (kódy klávesnice AT sú iné ako kódy na úrovni BIOS, ktoré sú platné ako pre model XT).

Klávesnica vysiela údaje signálom DATA v sériovom kóde, pričom signálom RQ\CLK synchronizuje jednotlivé bity. Mikropočítač 8042 po prijatí tohto kódu zastaví komunikáciu s klávesnicou a urobí konverziu pre BIOS do systémových identifikačných kódov známych už z XT (klávesnica vysiela v závislosti od akcie na klávesnici nie jeden, ale aj niekoľko slabík, ktoré presne indikujú, čo na klávesnici bolo stlačené, napr. pri pustení klávesu sa vyšlú dve slabiky).

Vytvorený identifikačný kód sa prerušením IRQ 1 ponúkne cez výstupný komunikačný register do riadiaceho programu BIOS. Po prenesení do BIOS-u mikropočítač 8042 obnoví komunikáciu s klávesnicou.

Pri zistení chyby parity prenosu mikropočítač 8042 požiada o opakovanie prenosu. Ak sa chyba opakuje, BIOS dostane namiesto kódu klávesu kód FFh a signalizuje sa chyba parity v stavovom registri. Mikropočítač tiež kontroluje čas prenosu z klávesnice. Ak čas prenosu presiahne 2 ms, BIOS tiež dostane kód FFh a v stavovom registri sa hlási prekročený časový limit príjmu (Receive Time Out Error). Mikropočítač 8042 vysiela do klávesnice povely. Na prenos sa využívajú tie isté vodiče DATA a RQ\CLK a ten istý sériový kód. Rozdiel je v smere prenosu, pričom mikropočítač vysiela jednotlivé bity a klávesnica signálom RQ\CLK synchronizuje (vyžaduje) príjem bitov.

Ak klávesnica nezačne vysielať impulzy RQ\CLK do 15 ms alebo celé prijatie povelu trvá viac ako 2 ms, mikropočítač 8042 vyšle BIOS-u kód FEh a hlási sa prekročenie časového limitu vysielania (Transmit Time Out Error). Klávesnica odpovedá na každý povel prijatý z 8042. Ak odpoveď má chybnú paritu, BIOS dostane kód FEh a hlási sa chyba parity a prekročenie času vysielania. Ak odpoveď nepríde do 25 ms, BIOS dostane kód FEh a hlási prekročenie času vysielania aj príjmu.

Riadiaci program mikropočítača 8042 umožňuje vypnúť vstup z klávesnice. Na jednom zo signálových vstupov mikropočítača 8042 je prepínač, ktorého stav sa sníma pri každom vstupe z klávesnice. Ak je prepínač v stave „vypnuté“, riadiaci program neprenesie do BIOS-u nijaké kódy klávesov. Prenos povelov do klávesnice je však možný naďalej a odpovede klávesnice na povely sa do BIOS-u tiež prenášajú. Toto umožňuje vybaviť mikropočítač kľúčom (s prepínačom), ktorým možno zabrániť vstupu z klávesnice (napr. ak treba povoliť prácu na klávesnici iba určeným osobám, ktoré majú kľúč).

Nasledujúci program úplne odpojí vstup z klávesnice (pozri inštrukcie **mov al,0f5h** a **out 60h,al**). Aby ste nemuseli resetovať počítač, umiestnil som do programu test na ľavé tlačidlo myši a následné zapnutie klávesnice (pozri inštrukcie **mov al,0f4h** a **out 60h,al**). Ak, pravdaže, nemáte myš pripojenú a spustíte nasledujúci program, neostane vám nič iné ako použiť reset.

Hodnoty v registri AL majú nasledujúci význam:

mov al,0f5h → nastavia sa štandardné charakteristiky klávesnice, ale klávesnica nezačne kontrolovať klávesy, čoho dôvodom je jej úplné zablokovanie.

mov al,0f4h → povolenie práce klávesnice, klávesnica vyprázdni svoj front, zahlási pripravenosť a normálne pracuje.

```
k_off_on.asm
.model small
.stack 100h
.data
txt1      db 'Keyboard is off$'
txt2      db 'Keyboard is on$'
.code
start:    mov ax,@data
          mov ds,ax
          mov ax,3
          int 10h
          lea dx,txt1
          mov ah,9
          int 21h
          xor ax,ax
          int 33h
          mov ax,1
          int 33h
          mov al,0f5h
          out 60h,al
lop:      mov ax,03
          int 33h
          and bl,00000001b
          jnz koniec
          jmp short lop
koniec:   mov al,0f4h
          out 60h,al
          mov ax,3
          int 10h
          lea dx,txt2
          mov ah,9
          int 21h
          mov ax,4c00h
          int 21h
          end start
```

Komunikácia mikropočítača 8042 s programami BIOS sa robí cez výstupný register z 8042 (len na čítanie programom BIOS), vstupný register 8042 (len na zápis programom BIOS) a stavový register (len na čítanie programom BIOS). Adresy registrov udáva tabuľka:

Register	Adresa
výstupný z 8042 (vstupný pre BIOS)	60H
vstupný 8042 (výstupný pre BIOS)	60H a 64H
stavový register (vstupný pre BIOS)	64H

MYŠ

Myš podobne ako klávesnica sa stala „neoddeliteľnou“ súčasťou počítačov. Vzhľadom na to, že technické parametre myší rôznych výrobcov sa väčšinou líšia, problém komunikácie myši sa rieši prostredníctvom riadiaceho programu myši, ktorý sa obyčajne dodáva s myšou. Používateľský program komunikuje s riadiacim programom myši prostredníctvom volania prerušenia INT 33H. Číslo funkcie je v registri AX. Bežne sú dostupné nasledujúce funkcie:

AX=00H inicializácia myši

Vstup : žiadny
Výstup: AX=FFFFH – riadiaci program myši je nainštalovaný
AX=0 – riadiaci program myši nie je inštalovaný
BX – počet tlačidiel, ktoré daný riadiaci program podporuje

Táto funkcia nastaví implicitný rozsah súradníc (v textových režimoch vráti riadiaci program myši hodnoty súradníc v rovnakom rozsahu ako v zodpovedajúcom grafickom režime, ale sú zaokrúhlené tak, aby boli deliteľné ôsmimi). Funkcia ďalej vypne kurzor, vynuluje registre počtu stlačení a uvoľnení tlačidiel pre funkcie 5H a 6H, vynuluje čítače relatívnej zmeny polohy pre funkciu 0BH, definuje implicitný textový a graficky kurzor, zapne emuláciu svetelného pera a nastaví implicitnú hodnotu citlivosti myši.

AX=01H zapnutie kurzora

Vstup: žiadny
Výstup: žiadny

Funkcia podľa zobrazovacieho režimu zapne grafický alebo textový kurzor. Implicitný grafický kurzor má tvar šípky.

AX=02H vypnutie kurzora

Vstup: žiadny
Výstup: žiadny

Funkcia vypne kurzor z obrazovky. Pohyby myši sú i naďalej sledované.

AX=03H zistenie pozície myši a stavu tlačidiel

Vstup: žiadny
Výstup: BX – stav tlačidiel
CX – horizontálne súradnice kurzora
DX – vertikálne súradnice kurzora

Bit 0 registra BX zodpovedá ľavému, bit 1 pravému, bit 2 prostrednému tlačidlu. Nastavený bit zodpovedá stlačenému tlačidlu myši.

AX=04H nastavenie pozície myši

Vstup: CX – horizontálne súradnice
DX – vertikálne súradnice
Výstup: žiadny

Obidve súradnice musia byť v rámci zvoleného rozsahu (pozri funkcie 07H a 08H), inak sú zaokrúhlené na najbližšiu možnú hodnotu.

AX=05H čítanie informácií o stlačení tlačidiel

Vstup : BX – kód tlačidla (0,1,2 viď funkcia 03H)
Výstup: AX – stav tlačidiel (ako funkcia 03H)
BX – počet stlačení zvoleného tlačidla od posledného čítania (touto funkciou)
CX – horizontálne súradnice myši pri poslednom stlačení zvoleného tlačidla
DX – vertikálne súradnice myši pri poslednom stlačení zvoleného tlačidla

AX=06H čítanie informácií o uvoľnení tlačidiel

Vstup: podobne ako funkcia 05H
Výstup: podobne ako funkcia 05H

Pri opakovanom volaní funkcie 05H a 06H pre to isté tlačidlo vráti BX nulu, ak nebolo tlačidlo medzitým znovu stlačené (prípadne uvoľnené). Hodnoty súradníc posledného stlačenia, resp. uvoľnenia, však zostávajú zachované, pokiaľ nezavoláte funkciu 00H.

AX=07H nastavenie horizontálneho rozsahu súradníc

Vstup: CX – minimálna hodnota
DX – maximálna hodnota
Výstup: žiadny

08H nastavenie vertikálneho rozsahu súradníc

Vstup: pozri funkciu 07H
Výstup: žiadny

Ak to pri funkciách 07H a 08H treba, sú hodnoty CX a DX zmenené, prípadne kurzor sa presunie dovnútra zvolenej oblasti.

AX=09H definovanie grafického kurzora

Vstup: BX – horizontálne posunutie vzťažného bodu
CX – vertikálne posunutie vzťažného bodu
ES:DX – Segment:Offset na pole masky obrazovky a masky kurzora
Výstup: žiadny

Posunutia sú v rozsahu -16 až +16 a určujú, na ktorý bod sa vzťahuje pozícia kurzora. Posunutie 0,0 zodpovedá ľavému hornému rohu a kladné posunutia spadajú dovnútra kurzora. Pred zobrazením kurzora je vykonaná operácia AND s maskou obrazovky so zodpovedajúcimi bitmi z obrazovky. Výsledok je modifikovaný maskou kurzora prostredníctvom operácie XOR. Takto vzniknuté pole je zobrazené. V režime s vysokým rozlíšením je kurzor jednofarebný, v ostatných režimoch môže mať viac farieb. Potom treba použiť 2 alebo 4 susedné bity na definovanie vlastnosti jedného bodu kurzora. Z toho vyplýva, že kurzor môže mať 16*16 bodov alebo 8*16 alebo 4*16 bodov.

AX=0AH definovanie textového kurzora

Vstup: BX – typ kurzora
CX – maska obrazovky alebo prvá linka kurzora
DX – maska kurzora alebo posledná linka kurzora
Výstup: žiadny

Ak je BX nula, je s maskou obrazovky a znakom pod kurzorom vykonaná operácia AND a výsledok je podrobený operácii XOR s maskou kurzora (nižší bajt je vždy znak ASCII a vyššie sú atribúty obrazovky). Ak je BX jedna, potom ďalšie dva parametre predstavujú prvú a poslednú linku kurzora v rámci znaku.

AX=0BH čítanie relatívnej zmeny polohy myši

Vstup: žiadny
Výstup: CX – horizontálna zmena polohy
DX – vertikálna zmena polohy

Funkcia vráti v registroch CX a DX kladnú alebo zápornú hodnotu, ktorá zodpovedá zmene pozície myši od posledného čítania touto funkciou. Čítače zmeny polohy sú tiež nulované funkciou 00H.

AX=0CH definovanie obslužnej rutiny prerušenia od myši

AX=0DH zapnutie emulácie svetelného pera

AX=0EH vypnutie emulácie svetelného pera

AX=0FH nastavenie citlivosti myši

Vstup: CX – horizontálny pomer
DX – vertikálny pomer
Výstup: žiadny

Funkcia nastavuje pomer počtu impulzov od myši na 8 bodov na obrazovke. Implicitne je to horizontálne 8 krokov na 8 bodov a vertikálne 16 na 8 bodov.

10H definovanie oblasti na vypnutie kurzora

12H definovanie veľkého grafického kurzora

13H nastavenie prahu rýchlosti na zdvojnásobenie odozvy myši

Podobne ako v predchádzajúcich prípadoch i teraz si ukážeme príklad, na ktorom lepšie pochopíte použitie služieb INT 33H. Program vykreslí na obrazovke nasledujúce menu

```
###############
# (o) Black   #
# ( ) Blue    #
# ( ) Green   #
# ( ) Red     #
# ( ) Magenta #
# ( ) Brown   #
# ( ) Exit    #
###############
```

a umožní vám pomocou myši vyberať z ponúknutých volieb. Ak zvolíte niektorú voľbu, napr.: GREEN, pozadie sa zmení na zelené, podobne je to aj pri ďalších voľbách. Ak vyberiete voľbu Exit, je vám ešte položená otázka „Do you wish to quit program? NO/YES“. V prípade, že kliknete ľavým tlačidlom na YES, program sa ukončí, ak kliknete na NO, prepínač sa nastaví na poslednú zvolenú farbu a pokračuje sa vo vykonávaní programu. Program preložíte takto: **tasm mys.asm a tlink mys.obj**.

```
;mys.asm
.model small
.stack 100h
.data
txt1            db '###############',10,13
                db '# (o) Black   #',10,13
                db '# ( ) Blue    #',10,13
                db '# ( ) Green   #',10,13
                db '# ( ) Red     #',10,13
                db '# ( ) Magenta #',10,13
                db '# ( ) Brown   #',10,13
                db '# ( ) Exit    #',10,13
                db '###############',10,13,'$'
txt2            db 'Do you wish'
                db ' to quit program?'
                db ' NO/YES$'
txt3            db 'Unable to detect'
                db ' mouse driver.$'
r_cx            dw 0
r_dx            dw 0
screen          db 4000 dup(?)
.code           position
                proc  near
                mov   bh,0
                mov   ah,02
                int   10h
```

```
                ret
position        endp
;Vypocet adresy vo videoram.
;--------------------
;Vstup: AX=riadok, BX=stlpec.
;Vystup: AX obsahuje offset adresy.
;--------------------
;Vzorec: [(80*AX)+BX]*2
;--------------------
poloha2         proc near
                mov dl,80
                add ax,bx
                mov dx,2
                mul dx
                ret
poloha2         endp
;--------------------
;Vypne zobrazovanie kurzora
;v textovom rezime.
;--------------------
kurzor_off      proc near
                mov ah,01h
                mov cx,20h*256+00h
                int 10h
                ret
kurzor_off      endp
;--------------------
;Zapne zobrazovanie kurzora
;v textovom rezime.
;--------------------
kurzor_on       proc near
                mov ah,01h
                mov cx,12h*256+14h
                int 10h
                ret
kurzor_on       endp
cls             proc near
                xor di,di
                inc di
                mov cx,80*25
skok1:          stosb
                inc di
                loop skok1
                ret
cls             endp
uschovaj        proc near
                push ds
                push es
                mov ax,seg screen
                mov es,ax
                mov di,offset screen
                mov cx,80*25*2
                mov ax,0b800h
                mov ds,ax
                xor si,si
skok2:          lodsb
                stosb
                loop skok2
                pop es
                pop ds
                ret
uschovaj        endp
obnov           proc near
                push ds
                push es
                mov ax,0b800h
                mov es,ax
                xor di,di
                mov cx,80*25*2
                mov si,offset screen
                mov ax,seg screen
                mov ds,ax
skok3:          lodsb
                stosb
                loop skok3
                pop es
                pop ds
                ret
obnov           endp
cls2            proc near
                mov al,32
                mov cx,7
                mov di,166
                cld
skok:           stosb
                dec di
                add di,160
                loop skok
                ret
cls2            endp
start:          mov ax,@data
                mov ds,ax
                mov ax,0b800h
                mov es,ax
                mov ax,3
                int 10h
                xor ax,ax
                int 33h
                cmp ax,0
                jnz pokracuj
                lea dx,txt3
                mov ah,9
                int 21h
                mov ax,4c00h
                int 21h
pokracuj:       call kurzor_off
                lea dx,txt1
                mov ah,9
                int 21h
                mov al,7
                call cls
                mov ax,1
                int 33h
lop:            mov ax,03
                int 33h
                and bl,00000001b
                jnz testx
                jmp short lop
testx:          mov [r_cx],cx
                mov [r_dx],dx
                mov ax,[r_dx]
                xor dx,dx
                mov bl,8
                div bl
                mov [r_dx],ax
                mov ax,[r_cx]
                xor dx,dx
                mov bl,8
                div bl
                mov [r_cx],ax
                cmp [r_dx],1
                jnz xxx
                cmp [r_cx],3
                mov al,7
                jz vykonaj
xxx:            cmp [r_dx],2
                jnz xxx1
                cmp [r_cx],3
                mov al,31
                jz vykonaj
xxx1:           cmp [r_dx],3
                jnz xxx2
                cmp [r_cx],3
                mov al,47
                jz vykonaj
xxx2:           cmp [r_dx],4
                jnz xxx3
                cmp [r_cx],3
                mov al,79
                jz vykonaj
xxx3:           cmp [r_dx],5
                jnz xxx4
                cmp [r_cx],3
                mov al,95
                jz vykonaj
xxx4:           cmp [r_dx],6
                jnz xxx5
                cmp [r_cx],3
                mov al,111
                jz vykonaj
xxx5:           cmp [r_dx],7
                jnz xxx6
                cmp [r_cx],3
                jz exit
xxx6:           jmp lop
vykonaj:        push ax
                mov ax,2
                int 33h
                pop ax
                call cls
                call cls2
                mov ax,[r_dx]
                mov bx,[r_cx]
                call poloha2
                mov di,ax
                mov al,'o'
```

```
                mov [es:di],al
                call uschovaj
                mov ax,1
                int 33h
                jmp lop
exit:           call cls2
                mov ax,2
                int 33h
                mov al,"o"
                mov [es:1126],al
                mov dx,4*256+25
                call position
                lea dx,txt2
                mov ah,9
                int 21h
                mov ax,1
                int 33h
lop1:           mov ax,03
                int 33h
                and bl,00000001b
                jnz testx1
                jmp short lop1
testx1:         mov [r_cx],cx
                mov [r_dx],dx
                mov ax,[r_dx]
                xor dx,dx
                mov bl,8
                div bl
                mov [r_dx],ax
                mov ax,[r_cx]
                xor dx,dx
                mov bl,8
                div bl
                mov [r_cx],ax
                cmp [r_cx],54
                jnz xx
                cmp [r_dx],4
                jz xx5
xx:             cmp [r_cx],55
                jnz xx1
                cmp [r_dx],4
                jz xx5
xx1:            cmp [r_cx],57
                jnz xx2
                cmp [r_dx],4
                jz vykonaj1
xx2:            cmp [r_cx],58
                jnz xx3
                cmp
                [r_dx],4
                jz vykonaj1
xx3:            cmp [r_cx],59
                jnz xx4
                cmp [r_dx],4
                jz vykonaj1
xx4:            jmp short lop1
xx5:            mov ax,2
                int 33h
                call obnov
                mov ax,1
                int 33h
                jmp lop
vykonaj1:       call kurzor_on
                mov ax,3
                int 10h
                mov
                ax,4c00h
                int 21h
                end start
```

Opis programu

V programe sú použité tieto nové procedúry - uschovaj, obnov a cls2. Procedúry uschovaj a obnov sa používajú na uschovanie videoram do pamäte a na jeho spätné obnovenie. Procedúra cls2 sa používa pri zmene voľby v menu a vymaže na všetkých pozíciách polohu prepínača. Je to potrebné, pretože pri aktivovaní inej voľby by prepínač zostal zobrazený aj na prechádzajúcej voľbe.

Program sa začína na návesti START. Nastaví sa textový režim č. 3 (80 x 25 znakov) a vzápätí sa otestuje, či je v pamäti prítomný ovládač myši. Ak nie je, vypíše sa chybové hlásenie ***„Unable to detect mouse driver."*** a program sa ukončí. V opačnom prípade sa pokračuje vo vykonávaní programu. Vypneme kurzor, vykreslíme menu a od návestia „lop" testujeme, či nedošlo k stlačeniu ľavého tlačidla myši. Ak je tlačidlo stlačené, pokračuje sa v programe na návesti „testx". Tu dôjde k uloženiu pozície myši do pamäte. Pretože tieto údaje o pozícii sú v bodoch, bude lepšie, ak ich vydelíme ôsmimi. Dostaneme tak „znakové" súradnice. Ďalej už nasleduje test jednotlivých pozícií, a ak je pozitívny, dôjde k skoku na program, ktorý zmení farbu v pozadí obrazovky. Ak kliknete na voľbu exit, zobrazí sa správa, či si prajete ukončiť program. Opäť sa testuje ľavé tlačidlo myši a testuje sa aj pozícia kurzora myši na obrazovke. Test na odpoveď NO/YES je analogický s testom, ktorý sa začína na návesti „testx". V programe sú ešte na vhodných miestach rozmiestnené funkcie na zapnutie MOV AX,1; INT 33H a vypnutie MOV AX,2; INT 33H kurzora myši.

Nabudúce si povieme o...

Budeme hovoriť o nadväznosti assemblera na vyššie programovacie jazyky, konkrétne C a PASCAL, ďalej si povieme o programoch TSR a zase si uvedieme praktické príklady.

Literatúra

[1] ABSHelp verzia 2.05. ABSoft Olomouc.
[2] V. Boukal: BIOS IBM PC. Grada 1992.

Dvanásta časť: Použitie assemblera v jazykoch C a Pascal, TSR...

Použitie assemblera vo vyšších programovacích jazykoch (C a Pascal)

Assembler sa často vyskytuje aj vo vyšších programovacích jazykoch vo forme krátkych a rýchlych rutín. Takto je možné dosiahnuť veľké zrýchlenie hlavne pri kritických častiach programu, ktoré by inak trvali veľmi dlho. Spomeniem niekoľko problémov, s ktorými sa môžete stretnúť pri aplikovaní assemblera do vyšších programovacích jazykov.

Zdanlivo neprekonateľným problémom v assembleri sa môže zdať pripojenie modulu k inému modulu. Najjednoduchším príkladom je spojenie dvoch assemblerovských modulov. Najdôležitejším pravidlom je použitie rovnakého pamäťového modelu pre všetky moduly.

Ďalej je potrebné pri odovzdávaní parametrov uviesť zásobník do pôvodného stavu. Linkeru tiež musíme oznámiť, ktoré funkcie a premenné majú byť viditeľné z iných modulov. Ak sme procedúru deklarovali priamo s použitím pamäťového modelu, ako FAR, resp. NEAR, musíme to dodržať aj pri jej volaní.

Pri používaní assemblerovských rutín sa väčšinou nevyhneme nutnosti odovzdať rutine nejaké parametre. Najčastejšie ide o hodnoty premenných daného modulu. Na úspešné odovzdávanie parametrov musíme brať do úvahy aj veľkosť dát. Každý programovací jazyk môže mať pre ten istý typ inú veľkosť. Parametre sa odovzdávajú buď priamo cez registre, alebo cez zásobník. Odovzdávanie parametrov cez registre je veľmi rýchle, pretože výstupné hodnoty jednej funkcie môžu byť vstupnými hodnotami druhej funkcie. Dôležité je, že tak nedochádza k zbytočným prístupom do relatívne pomalej operačnej pamäte. Druhou, veľmi často používanou metódou je odovzdávanie parametrov cez zásobník. Táto metóda nie je taká rýchla ako prvá, ale umožňuje bezpečne odovzdať ľubovoľné množstvo parametrov, ak sú dodržané určité podmienky. Odovzdávanie parametrov cez zásobník sa najčastejšie vyskytuje vo vyšších programovacích jazykoch.

Pripojované moduly sa píšu vo forme procedúr assemblera a líšia sa usporiadaním parametrov v zásobníku podľa cieľového jazyka. Ďalej sa už budeme venovať použitiu assemblera v dvoch najrozšírenejších programovacích jazykoch (C a PASCAL).

Jazyk C++

Existujú dva varianty použitia assemblera v jazyku C. Na dlhé assemblerovské rutiny použijeme priamo prekladač Turbo Assembler a výsledný súbor s príponou *.obj prilinkujeme k nášmu programu. **Druhý spôsob** – použijeme kľúčové slovo **asm**.

Syntax: **asm <operačný kód> <operandy> <; alebo nový riadok>**

kde

- <operačný kód> je platná inštrukcia 8086
- <operandy> obsahuje operandy akceptovateľné inštrukciou <operačný kód> a môžu sa odkazovať na konštanty, premenné a návestia jazyka C
- <; alebo nový riadok> signalizuje koniec inštrukcie assemblera

Pozor! V jazyku C nemôžeme používať znak bodkočiarka ako označenie komentára, pretože v jazyku C slúži bodkočiarka ako oddeľovač príkazov. Ak chcete komentovať nejaký príkaz assemblera, použite konvenciu jazyka C.

Najprv si vysvetlíme ten druhý spôsob, pretože je jednoduchší. Rozlišujeme dva typy zápisov

Prvý typ:

...

asm MOV AX,4C00H;

asm INT 21H;

...

Druhý typ: Aby sme stále nemuseli opisovať kľúčové slovo asm, využijeme množinové zátvorky „{ } „ na označenie bloku s assemblerom. POZOR! Najčastejšia začiatočníkova chyba je, že napíše kľúčové slovo asm, potom na novom riadku vytvorí blok a píše inštrukcie assemblera. Prečo vznikne chyba? Tu neplatí pravidlo o tom, že kde môže byť medzera, môže byť aj prechod na nový riadok. Prechod na nový riadok za asm sa považuje za oddeľovač, pomocou ktorého sa blok inline assemblera oddeľuje od kódu jazyka C.

SPRÁVNY ZÁPIS	CHYBNÝ ZÁPIS
asm {	asm
MOV AX,4C00H	{
INT 21H	MOV AX,4C00H
}	INT 21H
	}

Priame vkladanie assemblera do zdrojového textu však obsahuje aj rôzne obmedzenia:

- nie je možné využívať špeciálne inštrukcie procesorov i80386 a i80486,
- nedá sa použiť syntax a vlastnosti režimu IDEAL,
- nie je možné požívať assemblerovské makrá,
- neštandardné návestie.

Inline assembler umožňuje používať tieto operátory: (), [], ., +(unary), -(unary), :, OFFSET, SEG, TYPE, PTR, *, /, MOD, SHL, SHR, +(binary), -(binary), NOT, AND, OR, XOR.

Má rezervované tieto kľúčové slová: AH, AL, AND, AX, BH, BL, BP, BX, BYTE, CH, CL, CS, CX, DH, DI, DL, DS, DWORD, DX, ES, FAR, MOD, NEAR, NOT, OFFSET, OR, PTR, SEG, SHL, SHR, SI, SP, SS, ST, TYPE, WORD, XOR.

Pri vkladaní assemblera do zdrojového textu si musíte uvedomiť, že k dispozícii máte iba registre **AX, BX, CX, DX, SI, DI** a segmentový register **ES**. Meniť môžete aj príznaky, ktoré obsahuje stavový register. Naopak, zmenou registrov **BP, SP, CS, DS, SS** docielite vo väčšine prípadov pád operačného systému. Treba dávať pozor aj na registre SI a DI, pretože ich Turbo C používa na premenné typu register. Ak ich použijete v assemblerovskej procedúre, mali by ste ich uchovať pred vstupom do tejto procedúry a obnoviť ich pri jej opúšťaní. Ak vám nestačia obmedzené možnosti inline assemblera, môžete použiť v zdrojovom texte jazyka C za kľúčovými slovami **INCLUDE** voľbu **#pragma inline**. Táto voľba spôsobí, že všetky bloky s príkazom asm sa budú kompilovať v externom Turbo Assembleri.

Teraz si ukážeme dva jednoduché programy, ktoré budú využívať obidva typy zápisov. Ide o programy Šifruj a Dešifruj. Program Šifruj zakóduje zadaný súbor. Kódovanie prebieha tak, že vybraná množina kódov ASCII zo vstupného súboru sa pomocou inštrukcie ROL AL,1 posunie o jeden bit doľava. Tým vznikne zakódovaný súbor. Po otvorení súboru pomocou nejakého editora budete vidieť iba nezmyselné kombinácie znakov.

//Šifrovací program

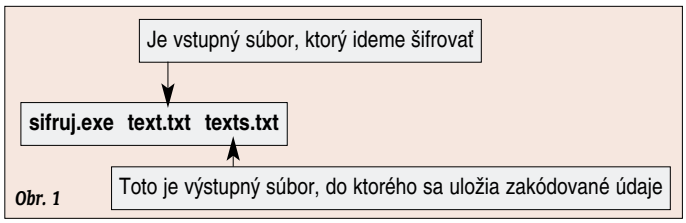
```
#include <STDIO.H>
#pragma inline
main(int argc, char *argv[])
{
int a;
FILE *in, *out;
if (argc != 3)
  {
  printf(„Nesprávny parameter alebo chybný počet parametrov v riadku.\ n");
return 1;
  }
if ((in = fopen(argv[1],"r")) == NULL)
  {
  fprintf(stderr,"Chyba: Vstupný súbor sa nepodarilo otvoriť.\ n");
       return 1;
  }
if ((out = fopen(argv[2],"w")) == NULL)
  {
  fprintf(stderr,"Chyba: Výstupný súbor sa nepodarilo otvoriť.\ n");
       return 1;
  }
while ((a = getc(in)) != EOF)
  {
if (a>=97 && a<=122 || a>=65 && a<=90 || a>=48 && a<=57)
  {
     asm  push ax;
     asm  mov al,a;
     asm  rol al,1;
     asm  mov a,al;
     asm  pop ax;
  }
putc(a,out);
}
fclose(in);
fclose(out);
return 0;
}
```

Aby toto tzv. kódovanie malo zmysel, musí existovať aj dekódovací program, ktorý súbor vytvorený programom Šifruj dá do poriadku, t.j. odkóduje ho.

//Dešifrovací program

```
#include <STDIO.H>
#pragma inline
main(int argc, char *argv[])
  {
int a;
FILE *in, *out;
if (argc != 3)
  {
    printf(„Nesprávny parameter alebo chybný počet parametrov v riadku.\ n");
    return 1;
  }
if ((in = fopen(argv[1],"r")) == NULL)
  {
  fprintf(stderr,"Chyba: Vstupný súbor sa nepodarilo otvoriť.\ n");
       return 1;
  }
if ((out = fopen(argv[2],"w")) == NULL)
  {
  fprintf(stderr,"Chyba: Výstupný súbor sa nepodarilo otvoriť.\ n");
       return 1;
  }
while ((a = getc(in)) != EOF)
  {
  if (a>=194 && a<=244 || a>=130 && a<=180 || a>=96 && a<=114)
  {
   asm {
    push ax
    mov al,a
    ror al,1
    mov a,al
    pop ax
   }
  }
putc(a,out);
}
fclose(in);
fclose(out);
return 0;
}
```

Program Dešifruj plní presne opačnú funkciu ako program Šifruj. Dekódovanie prebieha tak, že vybraná množina kódov ASCII zo vstupného súboru sa pomocou inštrukcie ROR AL,1 posunie o jeden bit doprava, čím vznikne súbor, ktorý sme predtým zakódovali. Ešte niekoľko poznámok k používaniu programov. Obidva programy sa spúšťajú s dvoma parametrami, napríklad takto (obr. 1):



Obr. 1

Ak zadáte chybný vstupný parameter, vypíše sa na obrazovke chybové hlásenie a vykonávanie programu sa ukončí.

Prvý spôsob sa aplikuje, ako už bolo povedané, na dlhé assemblerovské rutiny. Použije sa priamo prekladač Turbo Assembler a výsledný súbor s príponou *.obj prilinkujeme k nášmu programu.

Prekladače jazyka C, C++ generujú kód vo všetkých pamäťových modeloch. Pre každý pamäťový model existujú samostatné knižnice funkcií a pri tvorbe assemblerovských rutín nesmieme zabudnúť na rozdiely v spôsoboch volania (inštrukcie JMP, CALL). Jednotlivé varianty sú takéto (Tab. 1):

Model	Kód	Data
Tiny	near	near
Small	near	near
Medium	far	near
Compact	near	far
Large	far	far
Huge	far	far

Tab. 1

Veľké a malé písmená sa v jazyku C rozlišujú, preto musíme assembler prekladať s parametrom /ml alebo /**MX**. Prekladač jazyka C pri preklade vkladá pred meno každého symbolu znak „_“. V assemblerovskej časti je potrebné použiť direktívu **MODEL**, resp. direktívu **PUBLIC** s parametrom C, prípadne dodržať konvenciu „_“. Program napísaný sčasti v assembleri a sčasti v jazyku C je možné prekladať niekoľkými spôsobmi.

Prvý z nich je preklad zdrojového kódu jazyka C do assemblera a následne preklad všetkých modulov pomocou programu Turbo Assembler (**TASM**) do súboru .obj. Tieto súbory sa potom spolu s knižnicami spoja pomocou linkera **TLINK** do jedného spúšťacieho súboru.

Ďalším variantom je využitie možností integrovaného prostredia prekladačov Turbo C++ a Borland C++ s tým, že sa použije súbor s príponou .PRJ (projektový súbor). Postup je približne takýto: Najprv si vytvoríme projektový súbor, vložíme doň jednotlivé moduly s tým, že nastavíme, s akým programom a príkazovým riadkom sa majú preložiť. Pri preklade sa najprv preložia všetky moduly, ktoré sa nachádzajú v projektovom súbore, v prípade potreby i externým programom **TASM**. Nakoniec sa použije program **TLINK** na vytvorenie spúšťacieho súboru. Na deklarovanie externých symbolov sa používa v assembleri direktíva **EXTRN** a v jazyku C direktíva **EXTERN**.

V assembleri musíme premenné a funkcie vyznačiť direktívou **PUBLIC**, ak chceme, aby boli viditeľné. Pseudoinštrukcia **PUBLIC** udáva, že identifikátor definovaný v danom module je viditeľný za „hranicami“ modulu. Napríklad ak modul obsahuje celočíselné premenné znak, operacia, musíme do dátového segmentu vložiť riadok: **public_znak, _operacia**. Ostatne, uvidíte to na príklade.

Procedúry v assembleri by sa mali písať ako moduly, ktoré sa pripoja k programu v jazyku C. Musíte však dodržiavať určité konvencie. Tieto konvencie zabezpečia, že spájací program dostal potrebné informácie pre svoju prácu a že nebudú spájané moduly vytvárané v rôznych pamäťových modeloch. To, ako už vieme, by viedlo k nekorektnému behu programu. Všeobecná štruktúra assemblerovského programu je takáto:

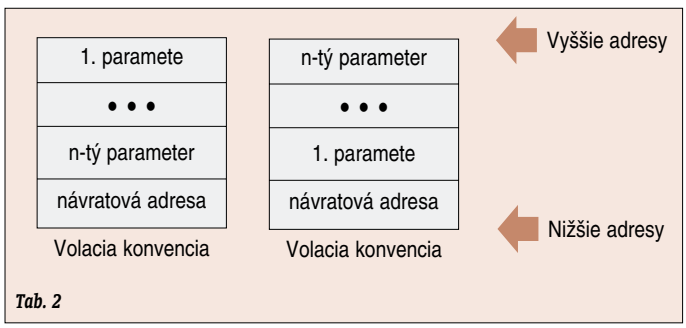
```
NAME meno_modulu

„text“    segment byte public ‘CODE’
          assume cs:_text, ds:_data
          ... kód programu ...
„text“    ends

„dseg“    group _data, _bss
„data“    segment word public ‘DATA’
          ... dáta programu ...
„data“    ends

„bss“     segment word public ‘BSS’
          ... neinicializované dáta ...
„bss“     ends
          end
```

V prípade, že niektorý segment nepoužívame, nie je potrebné ho definovať. Identifikátory „text“, „data“ a „dseg“ sa nahradia konkrétnym identifikátorom v závislosti od používaného pamäťového modelu. V tabuľke 1a sú uvedené identifikátory, ktoré sa používajú v jazyku C pre jednotlivé modely programu. „filename“ v tabuľke označuje meno súboru (modulu) a využíva sa v direktíve NAME a v náhradách identifikátorov.

Model	Náhrada identifikátora	Smerník na kód a dáta
Tiny, Small	„code“ = _TEXT	Kód: DW __TEXT: xxx
	„data“ = _DATA	Data: DW DGROUP: xxx
	„dseg“ = _DGROUP	
Compact	„code“ = _TEXT	Kód: DW _TEXT: xxx
	„data“ = _DATA	Data: DD DGROUP: xxx
	„dseg“ = _DGROUP	
Medium	„code“ = filename_TEXT	Kód: DD xxx
	„data“ = _DATA	Data: DW DGROUP: xxx
	„dseg“ = _DGROUP	
Large	„code“ = filename_TEXT	Kód: DD xxx
	„data“ = _DATA	Data: DD DGROUP: xxx
	„dseg“ = _DGROUP	
Huge	„code“ = filename_TEXT	Kód: DD xxx
	„data“ = filename_DATA	Data: DD xxx

Tab. 1a

Pre využitie volacej konvencie jazyka C v assemblerovskej časti definujeme model s parametrom C, resp. C uvedieme ako parameter v hlavičke procedúry. Pre túto konvenciu platí, že parametre sa ukladajú v obrátenom poradí, než sú deklarované v hlavičke (tým sa umožní volanie s premenným počtom parametrov). Zásobník čistí volajúci, pretože presne vie, čo všetko do zásobníka vložil (Tab. 2).



Tab. 2

Ak napríklad zavoláme funkciu void pascal funkcia (int i; char c), uložia sa na vrchol zásobníka najprv parametre, a to týmto spôsobom:

sp+2	hodnota premennej i (2 bajty)
sp	hodnota premennej c (1 bajt)

Ak zavoláme tú istú funkciu deklarovanú ako void funkcia(int i; char c); parametre sa do zásobníka uložia takto:

sp+2	hodnota premennej c (1 bajt)
sp	hodnota premennej i (2 bajty)

Tabuľka 3, ktorá obsahuje rôzne typy používané v jazyku C++. Všimnite si najmä, koľko bajtov je potrebných pre uvedené typy.

Typ	Počet bajtov	Rozsah
unsigned char	1	0 až 255
char	1	-128 až 127
enum	2	-32768 až 32767
short int	2	-32768 až 32767
unsigned int	2	0 až 65535
int	2	-32768 až 32767
long int	4	-2 147 483 648 až 2 147 483 647
unsigned long	4	0 až 4 294 967 295
far *	4	ďaleký ukazovateľ
near *	2	blízky ukazovateľ
float	4	$3.4 * (10^{-38})$ až $3.4 * (10^{+38})$
double	8	$1.7 * (10^{-308})$ až $1.7 * (10^{+308})$
long double	10	$3.4 * (10^{-4932})$ až $1.1 * (10^{+4932})$

Tab. 3

Teraz si ukážeme príklad spojenia assemblerovského modulu s jazykom C cez projektový súbor PRJ. Budete potrebovať prekladač jazyka C, najlepšie verziu Borland C++ 3.1, pretože program bol odladený práve pomocou tohto prekladača.

//convert.cpp

```
#include <stdio.h>
#include <conio.h>
#include <ctype.h>

extern  „C“ void convert(unsigned char operacia,unsigned char znak);

void help(void);

main(int argc, char *argv[])
{
 FILE *in,*out;
 extern unsigned char znak;
 extern unsigned char operacia;
 int a;
  if (argc == 1)
  {
    printf(„Zadajte prikaz /? alebo /h na zobrazenie helpu.“);
    return 1;
  }
if  (argv[1][0]  ==  ‘/’    &&    toupper(argv[1][1])  ==  ‘H’  ||
toupper(argv[1][1]) == ‘?’)
    {
     help();
     return 0;
    }
   if (argc != 4)
  {
    printf(„Nesprávny parameter alebo chybný počet parametrov v riadku.\
n“);
     return 1;
  }
if (argv[1][0] == ‘/’ &&  toupper(argv[1][1]) == ‘0’)
    {
     operacia=0;
    }
   else
    {
    operacia=255; //chyba
    }
if (argv[1][0] == ‘/’ &&  toupper(argv[1][1]) == ‘1’)
    {
     operacia=1;
    }
if ((in = fopen(argv[2],“r“)) == NULL)
      {
       printf(„\ aERROR Súbor ‘%s’ sa nepodarilo otvoriť.\ n“,argv[2]);
           return 1;
      }
if ((out = fopen(argv[3],“w“)) == NULL)
      {
       printf(„\ aERROR Súbor ‘%s’ sa nepodarilo otvoriť.\ n“,argv[3]);
return 1;
      }
```

```
while ((a = getc(in)) != EOF)
{
znak=(char) a;
convert(operácia,znak);
if (operacia==255)
  {
    printf(„\ a\ nChyba v príkaze Convert - neznámy typ operácie !!!\
n“);
    return 1;
  }
else
  {
    putc(znak,out);
  }
}
fclose(in);
fclose(out);
return 0;
}
void help(void)
{   clrscr();
   printf(„\ n*H* *E* *L* *P*“);
   printf(„\ n————————“);
   printf(„\ nconvertx.exe /h alebo /? -> zobrazí HELP“);
   printf(„\ n\ nconvertx.exe [typ konverzie] [vstupný súbor] [výstup-
ný súbor]“);
   printf(„\ n\ n[typ konverzie] ->“);
   printf(„\ n\ n/0 - Malé písmená -> Veľké písmená“);
   printf(„\ n/1 - Veľké písmená -> Malé písmená“);
   printf(„\ n\ nNapr. convertx.exe /0 clanok.txt správa.txt\ n“);
}
```

;convert.asm

```
name convert

_text segment word public ‘CODE’
      assume  cs:_text,ds:_data
         PUBLIC _convert
_convert proc near
           push bp
           push ds
           push di
           push si
              push ax
              push bx
           push cx
           mov al,_operacia
           cmp al,0
           jz @con3
           cmp al,1
           jz @con5
           mov _operacia,255
               jmp @nenasiel
@con3:   mov al,_znak
           cmp al,97
           jl @nenasiel
           cmp al,123
           jnb @nenasiel
           sub al,’a’-’A’
           mov _znak,al
           jmp @nenasiel
@con5:   mov al,_znak
           cmp al,65
           jl @nenasiel
           cmp al,91
           jnb @nenasiel
           add al,’a’-’A’
           mov _znak,al
@nenasiel: pop cx
                pop bx
                pop ax
                pop si
                pop di
                pop ds
                pop bp
                ret
_convert endp
_text        ends
_data segment word public ‘DATA’
public _znak,_operacia
_znak db 0
_operacia db 0
_data    ends
             end
```

O tom, ako vytvoriť projektový súbor, sa dozviete práve tu. Spustite program **BC.EXE** (najlepšie verziu 3.1). V menu kliknite na **PROJECT** a následne na **OPEN PROJECT**. Otvorí sa okno Open Project File a od vás sa očakáva, že zadáte názov projektu (napríklad: convertx.prj). Po zadaní mena projektu sa v spodnej časti obrazovky otvorí okno aj s vami zadaným názvom projektu. Teraz pomocou klávesu Insert vložte najprv program v jazyku C, ďalším stlačením modul v assembleri a nastavte prepínač „?“ na program v jazyku C. Ešte skontrolujte pomocou klávesovej skratky **CTRL+O**, či je nastavený správny prekladač pre jazyk C, ako aj pre modul v assembleri. Ak nie je, nastavte ho kliknutím na niektorú z volieb v boxe **Project File Translator**. V zásade však platí, že program bc.exe (verzia 3.1) si podľa prípony súboru nastaví všetky potrebné parametre automaticky. Ak chcete nejaký parameter prekladu zmeniť, musíte to urobiť v okne **Local Option**. Ešte kliknite v hlavnom menu na **OPTIONS|COMPILER|C++ OPTIONS**, v „Use C++ Compiles“ nastavte prepínač na **CPP extension**, kliknite na OK a uložte nastavenie **OPTIONS|SAVE**. Teraz už môžete spustiť kompiláciu pomocou klávesovej skratky **CTRL+F9**. Ak ste pri opisovaní zdrojového textu neurobili chybu a všetko je správne nastavené, vytvorí sa súbor s príponou exe.

Teraz niekoľko slov k programu. Program dokáže zmeniť všetky písmená vstupného súboru na malé alebo na veľké a uloží ich do výstupného súboru. Po spustení programu sa vypíše správa **Zadajte prikaz /? alebo /h pre zobrazenie helpu**. Po zadaní príkazu **convert.exe /?** sa na obrazovku vypíšu informácie, ako program používať.

Použitá literatúra

[1] P. Cápek, J. Rojko: Turbo assembler 3.0. Grada 1992.
[2] D. Červeňák, F. Novák: Turbo C pre pokročilých, Juventus press 1991.
[3] P. Frič, J. Dudová: Turbo pascal - Príručka programovania, SVŠT 1990.
[4] M. Kvoch: Programovaní v TURBO PASCALU 6.0, Kopp 1992.
[5] M. Kvoch: Programovaní v TURBO PASCALU 7.0, Kopp 1993.
[6] L. Bílek: Plynulé zhasínání a rozsvěcování obrazovky, Bajt 2/92.
[7] M. Dobrodenka: Priama práca s paletou VGA, PC -REVUE 1/96.

Trinásta časť: Použitie assemblera v jazykoch C a Pascal, TSR,...

Jazyk Pascal

Pre veľmi malé programy v assembleri je možné použiť príkaz a direktívu Turbo Pascalu **INLINE**. Príkaz **INLINE** pozostáva z kľúčového slova **INLINE** nasledovaného jedným alebo viacerými inline - položkami, vzájomne oddelenými znakom "/", ktoré sú uzatvorené v zátvorkách: INLINE(10/$1234/Pocet+1/Data-25); Každý prvok inline môže obsahovať voliteľný znak veľkosti "<" alebo ">", nasledovaný niekoľkými špecifikáciami offsetu. Špecifikátor offsetu pozostáva zo znaku "+" alebo zo znaku "-" nasledovaného konštantou. Každý prvok inline príkazu generuje jeden bajt alebo slovo kódu. Prvok inline generuje jeden bajt kódu iba v prípade ak pozostáva iba z konštánt a ak je rozsah jeho hodnoty (0 až 255). Ak je hodnota mimo rozsahu generuje sa slovo. Operátory "<" a ">" možno použiť na zmenu popísaného určovania veľkosti generovaného kódu. Ak prvok začína operátorom "<", kóduje sa len dolný bajt jeho hodnoty, aj keď je 16-bitová. Ak prvok začína operátorom ">", kóduje sa vždy slovo, aj keď je horný bajt 0. Napríklad: **INLINE (<$1234/>$44);** príkaz generuje 3 bajty kódu: $34, $44, $00. Registre BP, SP, SS, DS nemôžu byť príkazom **INLINE** modifikované, ostatné registre možno meniť. Teraz si ukážeme jednoduchý príklad použitia príkazu **INLINE**. Príkaz spôsobí čakanie na stlačenie ľubovoľnej klávesy.

```
inline($B8/$00/$00/ {MOX AX,0}

$CD/$16);  {INT 16H}
```

Príkazy **INLINE** možno ľubovolne kombinovať s ostatnými príkazmi v príkazovej časti bloku **BEGIN ... END**.

Okrem príkazu **INLINE** existuje ešte aj direktíva **INLINE**, ktorá umožňuje písať procedúry a funkcie, ktoré sa pri každom volaní expandujú do danej postupnosti strojových inštrukcií. Dajú sa prirovnať k makrám v assembleri. Pri volaní obvyklej procedúry alebo funkcie, generuje prekladač kód pre uloženie parametrov na vrch zásobníka a generuje inštrukciu **CALL** pre vykonanie procedúry alebo funkcie. Keď však voláte inline procedúru alebo funkciu, miesto inštrukcie **CALL** sa generuje kód. Nasledujúca funkcia vynásobí dve čísla typu integer a vráti longint hodnotu ako výsledok:

```
function longmul(x,y:integer):longint;
inline (
$58/        {POP DX}
$54/         {POP AX}
$F7/$EA);    {IMUL DX}
```

Všimnite si absenciu vstupného a ukončovacieho kódu a chýbajúcu inštrukciu **RET**. Nie sú potrebné, pretože pri volaní funkcie **longmul** sa uvedené štyri bajty vsunú do postupnosti inštrukcií. Direktíva **INLINE** je určená iba pre veľmi malé (niekoľko bajtové) procedúry a funkcie. Pretože má povahu makroinštrukcií, nemožno ju použiť ako argument operátorov @ a funkcií **Addr**, **Ofs** a **Seg**.

Ďalším spôsobom ako dostať do Pascalu Assembler je použiť zložený príkaz **ASM .. END**. Tento príkaz slúži k vkladaniu inštrukcií jazyka Assembler do zdrojového textu Turbo Pascalu. Prekladač dokáže spracovať bežné inštrukcie mikroprocesora I80286, adresové a hodnotové výrazy a niektoré direktívy. Ak je uvedená bezprostredne na hlavičkou podprogramu direktíva **assembler**, prekladač vie, že telo tejto procedúry bude obsahovať inštrukcie assembleru. Nasledujúci príklad ukazuje použitie tohto zloženého príkazu **ASM .. END**. Program sa skladá z troch procedúr. Sú to: **ClearScreen** (slúži na zmazanie obrazovky), Wait (čakanie na stlačenie ľubovoľnej klávesy) a **FastWrite** (používa sa na rýchly zápis textu do videoram). Keďže procedúry sú veľmi jednoduché a už ste sa s podobnými v tomto seriáli stretli – nebudem ich komentovať.

```
{Progasm.pas}
Uses crt, dos;
Procedure ClearScreen(attr:Byte; znak:char;
blok:word);
assembler;
asm
MOV  AX, $B800
MOV  ES, AX
XOR  DI, DI
MOV  CX, &BLOK
MOV  AH, ATTR
MOV  AL, &ZNAK
REP  STOSW
end;
Procedure Wait; assembler;
asm
MOV AX,0
INT 16H
end;
procedure FastWrite(TxT:String;
Row,Col,Attr:Byte);
assembler;
asm
PUSH    DS
MOV     AX,$B800
MOV     ES,AX
MOV     CH,Row
MOV     BL,Col
XOR     AX,AX
MOV     CL,AL
MOV     BH,AL
DEC     CH
SHR     CX,1
MOV     DI,CX
SHR     DI,1
SHR     DI,1
ADD     DI,CX
DEC     BX
SHL     BX,1
ADD     DI,BX
LDS     SI,DWORD PTR [TxT]
CLD
LODSB
XCHG    AX,CX
JCXZ    @FWExit
MOV     AH,Attr
@FWDisplay:
LODSB
STOSW
LOOP    @FWDisplay
@FWExit:
POP     DS
end;

BEGIN
ClearScreen(78,' ',2000);
FastWrite('Super rychle zobrazenie dosiah-
nete',12,23,78);
FastWrite('len pomocou rutiny
FastWrite',13,26,78);
Wait;
ClearScreen(7,' ',80*25);
END.
```

Posledným spôsobom ako spojiť assembler s jazykom Turbo Pascal je použiť externý modul v assembleri. Procedúry a funkcie napísané v assembleri možno spojiť s programami v Turbo Pascale použitím direktívy kompilátora **$L**. Zdrojový modul napísaný v assembleri musí byť preložený do súboru s príponou **obj** (najlepšie pomocou prekladača TASM). Pri spájaní viacerých modulov treba použiť direktívu **$L** pre každý modul. Najjednoduchšiu štruktúru assemblerovského programu (pre spojenie s Turbo Pascalom) dosiahnete ak použijete direktívu **.model** a segmentové direktívy **.code** a **.data**. Štruktúra by mohla vyzerať napríklad takto:

```
.model TPASCAL
.code ;začiatok code segmentu
assume cs:code,ds:code

paleta   proc far
arg @@par:byte, @@speed:byte
public paleta
...
ret
paleta   endp
.data ;začiatok data segmentu
;oblasť dát
end
```

Po nastavení modelu **TPASCAL** generujú segmentové direktívy **.code** a **.data** korektné mená segmentov pre spojenie s Turbo Pascalom. Okrem mien segmentov sa automaticky, na začiatku procedúry vygenerujú inštrukcie štandardnej vstupnej postupnosti inštrukcií - t.j.

```
push bp
mov bp,sp
```

a na konci inštrukcie štandardnej výstupnej postupnosti inštrukcií

```
pop bp
ret n
```

Procedúry a funkcie napísané v assembleri musia byť v Pascalovskom programe definované s kľúčovým slovom **external** napríklad:

```
procedure
FastWrite(TxT:String;
Row,Col,Attr:Byte); external;
```

Direktíva **external** musí byť uvedená bezprostredne za hlavičkou procedúry alebo funkcie a hovorí, prekladaču, že telo procedúry je definované v oddelenom module. Symboly musia byť v assemblerovskom súbore vyznačené ako **PUBLIC**. Aby sme mohli úspešne písať procedúry a funkcie, musíme vedieť v akých registroch Turbo Pascal očakáva vrátenú hodnotu. Tieto dáta sa predávajú registrom AL, resp. AX. Ukazovateľ je predaný vo formáte segment:offset pomocou registrov DX:AX. Pre reálne čísla pri využívaní programových knižníc sa 6-bajtové číslo predáva cez registre DX, BX a AX.

Ešte pre úplnosť tabuľka typov jazyka Pascal.

Typ	Počet bajtov	Rozsah
byte	1	0 až 255
shortint	1	-128 až 127
integer	2	-32768 až 32767
word	2	0 až 65535
longint	4	-2 147 483 648 až 2 147 483 647
real	6	2.9 * (10-39) až 1.7 * (10+38)
single	4	1.5 * (10-45) až 3.4 * (10+38)
double	8	5.0 * (10-324) až 1.7 * (10+308)
extended	10	3.4 * (10-4932) až 1.1 * (10+4932)
comp	8	-263 + 1 až 263 - 1

Toľko teória, ďalej sa budeme zaoberať príkladom, ktorý ukazuje použitie direktívy **.MODEL TPASCAL**. Najprv zdrojový text modulov.

```
;modul paleta
.model tpascal
.code
assume cs:code,ds:code

paleta   proc far
arg @@par:byte, @@speed:byte, @@a:dword

public paleta
push ds
push bp
push sp
push ss
push di
push si
cmp [@@par],0
jz short on
off:     mov [@@par],62
mov inc_dec,-1
 mov comp,-1
jmp short xloop
on:      mov inc_dec,1
mov comp,64
```

```
xloop:   mov bl,[@@par]
mov bh,63
set:     mov si,offset [@@a]
xor cx,cx

push dx
push ax
MOV DX,03DAh
MOV AH,8
test1:   in AL,DX
TEST AL,AH
JNZ test1
test2:   in AL,DX
TEST AL,AH
JZ test2
pop ax
pop dx
do:    mov dx,03c8h
mov ax,cx
out dx,al
inc dx
lodsb
mul bl
div bh
out dx,al
lodsb
mul bl
div bh
out dx,al
lodsb
mul bl
div bh
out dx,al
inc cx
cmp cx,64
jnz short do

 mov cl,[@@speed] ;spomalenie

xor ch,ch
q1:      push cx
 mov cx,65535
q:       loop q
pop cx
loop q1
add bl,inc_dec
cmp bl,comp
jnz set
pop si
pop di
pop ss
pop sp
pop bp
pop ds
ret
paleta   endp

comp     db          (?)
inc_dec  db          (?)

code     ends
end
```

{Modul Pascalu + Demo program}

```
Uses crt, dos;

type pole1=array[0..192] of byte;

var par,speed : byte;
speed2 : word;
a: pole1;

{$F+}
procedure paleta(par, speed:
byte; a:pole1); external;
{$L paleta.obj}

{$F-}
procedure nacitaj_paletu; assem-
bler;
asm                          mov
ax,1017h
xor bx,bx
mov cx,64
mov dx,offset a
int 10h              end;

Procedure ClearScreen(attr:Byte;
znak:char;
blok:word); assembler;
asm
mov  ax, $b800
mov  es, ax
xor  di, di
mov  cx, &blok
mov  ah, attr
mov  al, &znak
rep  stosw
end;
Procedure Wait; assembler;
asm
mov ax,0
int 16h
end;
Procedure HideCursor; Assembler;

Asm
MOV   ax,$0100
MOV   cx,$2607
INT   $10
end;
Procedure ShowCursor; Assembler;

Asm
MOV   ax,$0100
MOV   cx,$0506
INT   $10
end;
Procedure BoxXY(X1,Y1,X2,Y2,Attr
: Byte);
Assembler;
Asm
mov   ah,$06
mov   al,$00
mov   bh,Attr
mov   ch,Y1
mov   cl,X1
mov   dh,Y2
mov   dl,X2
dec   ch
dec   cl
dec   dh
dec   dl
int   $10
end;
BEGIN {demo program}
HideCursor;
nacitaj_paletu;
paleta(23,1,a); {vypni}
ClearScreen(45,' ',2000);
BoxXY(3,2,78,24,17);
BoxXY(5,4,76,22,34);
BoxXY(7,6,74,20,0);
gotoxy(26,13); {nastav poziciu}

Writeln('Press any key return to
DOS.');
paleta(0,1,a); {zapni}
Wait; {cakaj na klavesu}
paleta(23,1,a); {vypni}
ClearScreen(7,' ',80*25);
paleta(0,1,a); {zapni}
ShowCursor; {zobraz kurzor}
end.
```

Opis programu

Program pozostáva z nasledovných procedúr: **ClearScreen** (slúži na zmazanie obrazovky), **Wait** (čakanie na stlačenie ľubovoľnej klávesy), **HideCursor** (vypne zobrazovanie kurzora), **ShowCursor** (zapne zobrazovanie kurzora), **BoxXY** (procedúra vyplní zadanú plochu na obrazovke farbou) – je tu použitá služba BIOSu INT 10h a jej funkcia 06h, ktorá posúva v aktívnej stránke okno zadané ľavým horným a pravým dolným rohom o zadaný počet riadkov hore, prípadne celé okno zmaže a vyplní ho atribútom, ktorý obsahuje register BH. Procedúra **nacitaj_paletu** – načíta do zadanej vyrovnávacej pamäti obsah zvoleného bloku DAC (Digital to Analog Conversion) registrov. Je ju možné použiť iba u adaptérov VGA. Vstup: AH = 10h, AL = 17h, BX = číslo počiatočného registra, CX = počet registrov, ES:DX = ukazovateľ na vyrovnávaciu pamäť. Výstup: register CX obsahuje počet načítaných trojíc. Vyrovnávacia pamäť musí byť 3*CX bajtov dlhá. Tento krátky podprogram vlastne načíta do poľa, na ktoré ukazujú registre **ES:DX**, hodnoty všetkých DAC registrov. Poslednou ale najdôležitejšou procedúrou je externý modul **paleta**. V Pascalovskom module je táto procedúra ohraničená direktívami prekladača {**$F+**} a {**$F-**}. Od miesta, kde je uvedená direktíva {**$F+**}, je všetkým definovaným podprogramom automatický predpísaný model **FAR**. Pozor! direktíva prekladača {$F-} neznamená pre všetky ďalej definované podprogramy model **NEAR**. V podstate to znamená, že všetky podprogramy deklarované v časti **interface** programovej jednotky dostávajú model **FAR**, ostatné podprogramy (v časti **implementation**) dostávajú model **NEAR**.

A čo je cieľom procedúry? Cieľom procedúry je aby obrazovka pomaly zhasla a zasa sa rozsvietila s novým obsahom. Tento efekt ste už určite videli v rôznych programoch a hudobných demách.

Základný princíp spočíva v zobrazovaní rovnakých farieb s rozdielnou intenzitou. Toto sa dá v plnej miere realizovať až na adaptéri VGA, ktorý poskytuje 64 DAC registrov. Ich nastavenie určuje základnú paletu 64 farieb. každý z týchto registrov sa skladá zo 6-bitových hodnôt. Každá hodnota zodpovedá porade hodnotám jednotlivých zložiek R (červená), G (zelená) a B (modrá). Ak meníme hodnoty R, G, B, ale zároveň zachovávame ich vzájomný pomer (R:G:B), zobrazujeme vlastne tú istú farbu, líšiacu sa iba svojou intenzitou. Ak nastavujeme teda vhodne tieto hodnoty, docielime požadovaného efektu rozsvietenia a zhasnutia. V programe sa to deje násobením hodnôt pôvodných zložiek RGB číslami, ktorých hodnota sa plynule mení v intervale <0,1>. V programe za návestím **set** sa testuje tretí bit Input status registra (3DAh). Tento bit určuje, či sa obraz práve vykresľuje, alebo nie. Dá sa tak zabrániť rušivému "sneženiu" na obrazovke.

Vstupné parametre procedúry: **par** – obsahuje hodnotu 23 (zhasni obrazovku) a hodnotu 0 (rozsvieť obrazovku), **speed** – premenná obsahuje hodnotu na spomalenie programu (týka sa to rýchlejších počítačov, treba si ju nastaviť, pozor! hodnota 0 neznamená, že cyklus neprebehne ani jeden krát, ale naopak 65535 krát, ale to isto už viete.), posledným parametrom je ukazovateľ na pole. Pri vlastných experimentoch nezabudnite vždy na začiatku naplniť pole pomocou procedúry **nacitaj_paletu**. To je nadnes všetko, bohužiaľ rezidentné programy až nabudúce.

Použitá literatúra

[1] P. Cápek, J. Rojko: Turbo assembler 3.0. Grada 1992.

[2] D. Červeňák, F. Novák: Turbo C pre pokročilých, Juventus press 1991.

[3] P. Frič, J. Dudová: Turbo pascal - Príručka programovania, SVŠT 1990.

[4] M. Kvoch: Programovaní v TURBO PASCALU 6.0, Kopp 1992.

[5] M. Kvoch: Programovaní v TURBO PASCALU 7.0, Kopp 1993.

[6] L. Bílek: Plynulé zhasínání a rozsvěcování obrazovky, Bajt 2/92.

[7] M. Dobrodenka: Priama práca s paletou VGA, PC -REVUE 1/96.

Štrnásta časť: Terminate & Stay Resident

Dnes si povieme základné informácie o vytváraní rezidentných programov v assembleri a ukážeme si aj konkrétny príklad.

Rezidentné programy

Vznik rezidentných programov bol ovplyvnený ťažkopádnosťou operačného systému MS DOS. Keďže MS DOS bol jednoúlohový operačný systém, ukázalo sa, že riešiť niektoré problémy pomocou rezidentných programov je výhodné.

„Resident program“ v preklade doslova znamená usadený, bývajúci program. Totiž po ukončení takéhoto programu a po odovzdaní riadenia operačnému systému zostane časť alebo celý tento program v pamäti počítača. Preto môže pracovať ďalej. Samozrejme, za predpokladu, že si pred svojím ukončením zabezpečil nejakú „cestu“, odkiaľ opätovne dostane riadenie – procesor začne vykonávať jeho kód. Z tohto hľadiska môžeme rezidentným programom nazvať „každý“ program, ktorý je ukončený funkciou MS DOS-u Terminate & Stay Resident (TSR - INT 27h) - ukonči a zostaň v pamäti. V tejto pamäti potom čaká, bez toho, aby ovplyvňoval beh iných programov, na svoju aktiváciu. To, že program musí čakať, než mu bude pridelené v istom okamihu riadenie, je dôsledkom jednoúlohového operačného systému MS DOS.

Typy rezidentných programov

1. Ovládače zariadení

Túto skupinu tvoria programy zabezpečujúce priame ovládanie hardvéru periférnych zariadení (napr. ovládač myši, tlačiarne), ďalej programy, ktorých cieľom je zlepšiť a rozšíriť možnosti štandardných zariadení (napr. disková cache, národný ovládač klávesnice atď.). Tieto ovládače sú väčšinou charakteristické tým, že majú príponu **SYS** (printer.sys, ramdrive.sys, driver.sys, himem.sys atď.). Sú to programy, ktoré sú počas behu operačného systému nastálo (rezidentne) umiestnené v operačnej pamäti. Ich úlohou je sprístupňovať operačnému systému i používateľom vstupno-výstupné zariadenia počítača.

2. Pop-up programy

Tieto programy sú charakteristické tým, že sú prístupné po stlačení dopredu nadefinovanej kombinácie klávesov. Do tejto skupiny patria napríklad rôzne slovníky, kalkulačky, zápisníky atď.

3. Bezpečnostné systémy

Bezpečnostné systémy sú typickým príkladom potreby rozšírenia možností systému, či ide o zabezpečenie počítača proti zneužitiu (password), alebo proti deštruktívnej činnosti (formát disku) a pod. Do tejto skupiny tiež patria rôzne antivírusové štíty, ktoré dokážu identifikovať a niekedy aj odstrániť vírus, prípadne naň aspoň upozornia.

4. Rezidentný program typu Vírus

Ďalšou – bohužiaľ, stále početnejšou – skupinou rezidentných programov sú

vírusy. Na prvý pohľad ich nevidieť, teda ak si pozriete výpis pamäte príkazom MEM /C /P, nenájdete ich medzi ostatnými programami v pamäti. Sú totiž veľmi dobre ukryté. Ich prítomnosť zistíte až po použití antivírusového programu alebo podľa zväčšujúcich sa EXE, COM súborov, ktoré vírusy najčastejšie napádajú.

5. Prepínače úloh

Prepínače úloh, tzv. Taskswapery, sú rezidentné programy, ktoré vznikli na základe potreby pracovať s viacerými programami naraz. Ide väčšinou o programy, ktoré umožňujú prepínať procesy (úlohy), ktoré sú v pamäti. Niečo podobné, ako má dnešný operačný systém Windows (kombinácia klávesov ALT+TAB).

Prerušenie

Najprv si vysvetlíme, čo pod pojmom prerušenie vlastne rozumieme. **Prerušenie** je signál, ktorý procesoru prikáže, aby zastavil vykonávanie hlavného programu a začal sa zaoberať iným programom.

Podľa toho, čím je prerušenie generované, rozlišujeme:

a) HARDVÉROVÉ prerušenia

– NON MASCABLE INTERRUPT (NMI) - nemaskovateľné prerušenie

– (INTR) - maskovateľné prerušenie

Vývod NMI je určení pre prerušenie, ktoré nie je možné zakázať. Slúži na signalizáciu havarijných stavov počítača (napr. pokles napájacieho napätia, chyba parity operačnej pamäte...). NMI má vyššiu prioritu ako INTR.

Vývod INTR je väčšinou budený ďalším obvodom - radičom prerušenia 8259. INTR je maskovateľné prerušenie, pretože je ho možné programovo zakázať vynulovaním bitu IF v príznakovom registri procesora. Čiže ak je nastavený na nulu, prerušenie je zakázané, v opačnom prípade je povolené (IF=1). Tento bit je možné nulovať inštrukciou CLI alebo nastaviť na jednotku inštrukciou STI.

b) SOFTVÉROVÉ prerušenia

– vzniknuté inštrukciou INT n, kde n je číslo z rozsahu 0-255

Procesor 8086 rozlišuje 256 možných prerušení. Každému prerušeniu prislúcha 32-bitová logická adresa (segment:offset). Táto adresa sa nazýva vektor prerušenia a ukazuje na miesto v pamäti, kde sa začína podprogram, ktorý sa vykoná pri vyvolaní daného prerušenia. Vektorom prerušenia je vyhradená pamäť od adresy 0000:0000 s dĺžkou 1024 bajtov. Pre každý vektor prerušenia sú rezervované 4 bajty (256*4=1024). V prípade prerušenia je vykonaný podprogram, ktorého logická adresa je v tabuľke vektorov na adrese (číslo **prerušenia * 4**)

Ako už bolo povedané, o prerušenie sa stará tzv. radič prerušení (obvod 8259). Tento obvod rozhoduje o tom, ktoré prerušenie má vyššiu prioritu a v akom poradí budú jednotlivé prerušenia obslúžené. Samozrejme, že najvyššiu prioritu majú prerušenia vznikajúce z kritických chýb, napríklad pokles napájacieho napätia, chyba parity operačnej pamäte a pod. Teraz nasledujú jednotlivé kroky, ktoré procesor vykoná v okamihu vyvolania prerušenia INT x, kde x je číslo z intervalu <0,255>:

1. do zásobníka sa uloží obsah registra F (Flags),
2. dôjde k vynulovaniu príznakov IF (Interrupt enable flag) a TF (Trap flag),
3. do zásobníka sa uloží obsah registra CS (Code segment),
4. obsah adresy (x * 4 +2) sa uloží do registra CS,
5. do zásobníka sa uloží obsah registra IP (Instruction pointer),
6. obsah adresy (x *4) sa uloží do registra IP.

Programovacie jazyky a rezidentné programy

V rámci literatúry o rezidentných programoch sa stále vedie spor o tom, aký programovací jazyk je najvhodnejším prostriedkom na písanie rezidentných programov. Nie je možné prijať tvrdenie, že assembleru v dnešnej dobe definitívne odzvonilo, pretože faktom zostáva, že v niektorých výnimočných prípadoch sa programovaniu na strojovej úrovni nevyhneme. Samozrejme, čas potrebný na vývoj rovnakej aplikácie v jazyku Pascal alebo C je niekoľkonásobne kratší. Rozhodnutie je teda na vás, ktorý programovací jazyk si vyberiete. No hlavným kritériom pri výbere vývojových prostriedkov by mali byť požiadavky, ktoré kladiete na rezidentnú aplikáciu. Je jasné, že pri požiadavke na extrémne krátky kód sa optimalizácii na úrovni assemblera nevyhneme, ale pri potrebe rýchlo vyvinúť zložitejšiu aplikáciu, pri ktorej nám natoľko nezáleží na dĺžke a rýchlosti, môžeme smelo použiť jazyk C alebo Pascal.

Štruktúra rezidentného programu

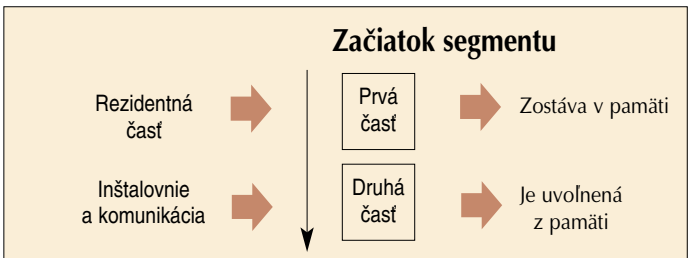
Rezidentný program sa zvyčajne skladá z dvoch častí. **Prvá časť** (tzv. rezidentná časť) zostane v pamäti aj po ukončení programu (pozri službu MS DOS-u – INT 21H, kód 31H). V pamäti je potom tak dlho, kým neresetujete (CTRL+ALT+DEL) počítač, prípadne dokiaľ nie je uvoľnená niektorým zo spôsobov na odstraňovanie rezidentných programov z pamäte. Veľkosť rezidentnej časti závisí od veľkosti zásobníka, veľkosti dát a programového kódu. **Druhá časť** je tá, ktorá je pri ukončení programu uvoľnená z pamäte počítača. Táto časť zvyčajne slúži na inštalovanie samotnej rezidentnej časti a z toho dôvodu nie je potrebné, aby zaberala pamäť. V našom príklade, ktorý si ukážeme, budeme pomocou tejto druhej časti okrem inštalovania aj komunikovať s rezidentnou časťou.

Ak by ste prvú a druhú časť zamenili (pozri obrázok 1), potom by nebolo možné prebytočný kód a dáta uvoľniť z pamäte. Výsledný rezidentný program by tak bol oveľa väčší.

Služby DOS-u

Na tomto mieste si uvedieme základné služby operačného systému MS DOS, ktoré sa používajú pri tvorbe rezidentného programu. Konkrétne ide o tieto služby:

INT 27H – Terminate and Stay Resident (TSR) – táto služba sa používa na ukončenie programu s tým, že jeho časť zostane i po ukončení v pamäti. Veľkosť, ktorá má zostať



Obr. 1 Štruktúra rezidentného programu (Model TINY)

rezidentne v pamäti, je určená offsetom posledného bajtu + 1 miesta, ktoré má byť zachované v pamäti. Takto získaná hodnota sa uloží do registra DX. Okrem toho register CS musí ukazovať na začiatok PSP (Program Segment Prefix). Služba sa používala na ukončovanie programov typu COM, pretože maximálna veľkosť rezidentného kódu bola obmedzená na 64 KB. Neskôr bola služba nahradená rovnakou funkciou DOS-u číslo 31H.

```
Newint8   PROC FAR
          PUSH AX
          ...
          POP AX
Newint8   ENDP
;Koniec rezidentnej časti
Install:  MOV AX,0
          ...
          MOV DX, offset Install+1
          INT 27H
```

INT 21H, funkcia 31H – služba sa používa na ukončenie programu s tým, že jeho časť zostane i po ukončení v pamäti. Služba navyše vracia chybový kód nadradenému programu, ktorý sa nachádza v registri AL. Chybový kód v registri AL predstavuje informáciu o priebehu programu. Veľkosť programu, ktorá má rezidentne zostať v pamäti, sa uloží do registra DX. Ale pozor!!! Do registra DX sa ukladá veľkosť pamäte v paragrafoch. Paragraf má veľkosť 16 bajtov. To znamená, že hodnotu v registri DX musíme ešte vydeliť šestnástimi (pozri príklad v assembleri).

```
MOV DX,(Start-Zaciatok)/16+16+1
MOV AX,3100H
INT 21H
```

V niektorých programoch sa môžete stretnúť aj s takýmto zápisom:

```
MOV DX,(Start-Zaciatok) SHR 4
ADD DX,1
MOV AX,3100H
INT 21H
```

Všimnite, že v obidvoch prípadoch sa pridáva ešte jeden paragraf navyše. To preto, ak by náhodou výsledok nebol deliteľný bezo zvyšku.

Zmenu vektora prerušenia je možné vykonať buď priamo v tabuľke vektorov, alebo pomocou špeciálnych funkcií DOS-u. Pri priamych zásahoch do tabuľky vektorov prerušenia treba obmedziť na minimum nebezpečenstvo, že v priebehu zmeny dôjde k prerušeniu. Vektor by potom mal nesprávnu hodnotu a mohlo by dôjsť k pádu systému. Toto nebezpečenstvo je možné obmedziť inštrukciou CLI, ktorá potlačí všetky prerušenia okrem nemaskovateľného. Po vykonaní zmeny treba prerušenie povoliť inštrukciou STI. Aj keď zakážeme prerušenie, stále existuje nebezpečenstvo nemaskovateľného prerušenia, ktoré vzniká napríklad pri poklese zdroja napätia, chybe parity operačnej pamäte atď. Preto sa odporúča meniť vektory prerušenia pomocou funkcií DOS-u (INT 21H, funkcie 35H a 25H), ktoré obmedzujú nebezpečenstvo prípadného prerušenia na minimálnu možnú mieru.

INT 21H, funkcia 35H – pomocou tejto funkcie možno zistiť adresu programu obsluhy ktoréhokoľvek prerušenia. Pred použitím funkcie musí register AL obsahovať číslo prerušenia, ktorého vektor chceme zistiť. Služba vráti v registroch ES a BX adresu programu obsluhy. Keďže chceme presmerovať adresu programu obsluhy, treba údaje získané v registroch ES a BX uchovať pre prípad, ak by sme chceli neskôr program odinštalovať a všetko vrátiť do pôvodného stavu.

```
MOV AX,351CH                INT 21H            MOV WORD PTR
Staryint,BX                 MOV WORD PTR Staryint+2,ES
```

INT 21H, funkcia 25H – funkcia umožňuje nastaviť ľubovoľný z vektorov prerušenia na určenú adresu. Pred použitím funkcie musí register AL obsahovať číslo prerušenia,

ktorého vektor chceme zmeniť. Registre DS a DX musia obsahovať segmentovú adresu novej obsluhy prerušenia.

```
PUSH CS
POP DS
MOV AX,251CH
MOV DX,OFFSET Novyint
INT 21H
```

Pri používaní tohto prerušenia je potrebné zachovať maximálnu opatrnosť. Nesprávne nastavenie vektorov prerušenia môže viesť k veľmi vážnym kolíznym situáciám. Funkcia 25H nerobí nijaké kontroly čísla prerušenia ani nastavovanej adresy. Používateľské programy by nemali zasahovať do vektorov prerušenia 00H až 1FH, vyhradených programu BIOS a technickému vybaveniu. Výnimkou sú používateľské ovládače zariadení a zložitejšie rezidentné programy, pre ktoré býva zásah do týchto vektorov prerušenia potrebný. V žiadnom prípade však programy nesmú meniť vektory prerušenia rezervované na využitie operačným systémom DOS s číslami 28H až 2EH a 30H až 3FH.

Praktický príklad

Toľko nadnes teórie, teraz sa venujme praxi. Ukážeme si rezidentný program, ktorý zobrazuje do pravého horného rohu obrazovky aktuálny dátum a čas. V programe sa využíva prerušenie INT 1CH. Túto službu volá obslužná rutina systémového časovača INT 08H pri každom tiku (približne 18,2-krát za sekundu, t. j. každých 55 ms). Pretože program obsahuje inštalovanie, komunikáciu s rezidentnou časťou a aj odinštalovanie rezidentnej časti, bude o nejaký ten riadok dlhší ako inokedy. Najprv začneme rezidentnou časťou. Tá po nainštalovaní v pamäti zaberá len 784 bajtov, a ak program trochu upravíte, bude to ešte menej.

```
;————————————————————
; Program: Rezidentne hodiny +
;      + aktualny datum.
;————————————————————
.model tiny
.286
CODE SEGMENT public
assume cs:code,ds:code
org 100h
start1:             jmp start
segadr              equ 0b800h
atribut             equ 079h
atributnavod        equ 7
zaciatok label near
staryint            dw ?
                    dw ?
offadr              dw ?
offadr1             dw ?
offadr2             dw ?
d_on_off            db 1
pat                 db 18
citac               db 0
param               db 0
sekcas              db ?
mincas              db ?
hodcas              db ?
cas                 db 0,0,':',0,0,':',0,0
datum               db 0,0,0,0,0,0,0,0,0,0
@filename:
disp                macro adrstr
                    push cs
                    pop ds
                    mov dx,offset adrstr
                    mov ah,09h
                    int 21h
                    endm
novyint             proc
zac:                push es
                    push ds
                    push di
                    push si
                    push dx
                    push cx
                    push bx
                    push ax
                    push bp
                    pushf
                    push cs
                    pop ds
                    mov al,citac
                    cmp al,0
                    jnz pokr
                    mov ah,0
                    xor dx,dx
                    mov al,sekcas
                    mov ch,5
                    div ch
                    xor al,al
                    cmp ah,0
                    jz pokr1
                    mov pat,18
                    jmp short pokr
pokr1:              mov pat,19
pokr:               mov al,citac
                    inc al
                    cmp al,pat
                    jge sekunda
                    mov citac,al
ukonci:             popf
                    pop bp
                    pop ax
                    pop bx
                    pop cx
                    pop dx
                    pop si
                    pop di
                    pop ds
                    pop es
old:                jmp dword ptr cs:staryint
sekunda:            mov citac,00
                    inc sekcas
                    mov dl,3
                    cmp sekcas,60
                    jl dalsi
                    mov sekcas,00
                    inc mincas
                    cmp mincas,60
                    jl dalsi
                    mov mincas,00
                    inc hodcas
                    push ax
                    push bx
                    push cx
                    mov bl,2
                    call beep
                    pop cx
                    pop bx
                    pop ax
                    cmp hodcas,24
                    jl dalsi
                    mov hodcas,00
                    push dx
                    mov ax,cs
                    mov es,ax
                    mov ax,offset datum
                    mov di,ax
                    call datum1
                    pop dx
dalsi:              mov bl,sekcas
opakuj:             xor bh,bh
                    xor ax,ax
                    mov cx,0010h
tu:                 add bx,bx
                    adc al,al
                    daa
                    loop tu
                    mov bx,ax
                    rol ax,4
                    mov al,bl
                    and al,0fh
                    or ax,3030h
                    push ax
                    dec dl
                    jz zobraz
                    mov bl,mincas
                    xor bh,bh
                    xor ax,ax
                    mov cx,0010h
tu1:                add bx,bx
                    adc al,al
                    daa
                    loop tu1
                    mov bx,ax
                    rol ax,4
                    mov al,bl
                    and al,0fh
                    or ax,3030h
                    push ax
                    dec dl
                    jz zobraz
                    mov bl,hodcas
                    jmp opakuj
zobraz:             pop dx
```

```
                mov cas,dh
                mov cas[1],dl
                pop dx
                mov cas[3],dh
                mov cas[4],dl
                pop dx
                mov cas[6],dh
                mov cas[7],dl
                mov ah,atribut
                mov bx,segadr
                mov es,bx
                mov bx,offadr
                xor si,si
                xor cas+2,':' XOR ' '
                xor cas+5,':' XOR ' '
d_znak:         mov al,cas[si]
                mov word ptr es:[bx],ax
                inc bx
                inc bx
                inc si
                cmp si,8
                jl d_znak
                mov al,[d_on_off]
                cmp al,0
                je off
                mov ah,atribut
                mov bx,segadr
                mov es,bx
                mov bx,offadr1
                xor di,di
d_znak1:        mov al,datum[di]
                mov word ptr es:[bx],ax
                inc bx
                inc bx
                inc di
                cmp di,10
                jl d_znak1
off:            jmp ukonci
novyint         endp
beep            proc near
                cli
                mov al,10110110b
                out 43h,al
                mov ax,533h
                out 42h,al
                mov al,ah
                out 42h,al
                in  al,61h
                mov ah,al
                or  al,03
                out 61h,al
                sub cx,cx
g7:             loop g7
                dec bl
                jnz g7
                mov al,ah
                out 61h,al
                sti
                ret
beep            endp
bcdkonvert      proc near
                push cx
                mov cl,4
                mov ah,al
                shr al,cl
                and ax,0f0fh
                add ax,'00'
                pop cx
                ret
bcdkonvert      endp
datum1          proc near
                push es
                push ds
                push di
                push si
                push dx
                push cx
                push bx
                push ax
                push bp
                pushf
                mov ax,0400h
                mov bx,8Eh
                int 1ah
                mov al,dl
                call bcdkonvert
                stosw
                mov al,'.'
                stosb
                mov al,dh
                call bcdkonvert
                stosw
                mov al,'.'
                stosb
                mov al,ch
                call bcdkonvert
                stosw
                mov al,cl
                call bcdkonvert
                stosw
                popf
                pop bp
                pop ax
                pop bx
                pop cx
                pop dx
                pop si
                pop di
                pop ds
                pop es
                ret
datum1          endp
```

Teraz nasleduje opis rezidentnej časti programu. Program používa model TINY. Na začiatku je definovaných niekoľko premenných, ktoré budeme ďalej využívať. Nasleduje makro pre výpis chybových hlásení na obrazovku. Procedúra **novyint** predstavuje rezidentnú časť, ktorá zostane v pamäti a bude na ňu presmerované prerušenie 1CH. Procedúra sa začína tým, že uchovávame potrebné registre na zásobník. Register AL naplníme počtom tikov časovača. Ak AL neobsahuje nulu, vykoná sa skok na návestie **pokr**. Ďalej vynulujeme register AH aj DX a register AL naplníme počtom sekúnd. Register CH naplníme hodnotou 5 a vykonáme delenie **sekcas mod 5**. Vynulujme register AL a testujme výsledok delenia, ak je to nula, skok na návestie **pokr1**. V opačnom prípade nastav počet tikov na 18 za sekundu. Od návestia **pokr1** a **pokr** sa testuje, či už prešla jedna **sekunda**, ak áno, potom treba zobraziť čas na obrazovku. Ešte vyberieme registre zo zásobníka a skok na pôvodnú obsluhu 1CH. Premenná pat môže nadobúdať hodnoty 19 alebo 18, a to podľa toho, či je počet sekúnd deliteľný bezo zvyšku piatimi. Prečo práve piatimi? Pretože každých 5 sekúnd získame navyše jednu celú sekundu. Počet tikov za jednu sekundu je totiž 18,2, a teda 5*0,2=1 s. Dostali sme sa na návestie sekunda. Vynulujeme čítač a register DL naplníme číslom 3. Toto číslo znamená, že sa budú konvertovať 3 čísla do ASCII (t. j. sekundy, minúty a hodiny). Od tohto návestia sa tiež začína počítanie sekúnd, minút a hodín. Ak prejde 1 hodina, ozve sa krátke cvaknutie, pozri procedúra **beep**. Po zvukovom signáli nasleduje test, či už prešlo 24 hodín, ak áno, vynuluje sa premenná **hodcas** a nastaví sa nový dátum. Od návestia **dalsi** sa uskutočňuje prevod obsahu registra BL na desiatkové číslo v ASCII. Za návestím **zobraz** sa začína výpis času na obrazovku. Využíva sa tu priamy prístup do videoram. Ďalej nasleduje test, či je povolené zobrazovať dátum, ak áno, zobrazí sa na obrazovku o jeden riadok nižšie ako čas. Rezidentná časť okrem novej obsluhy prerušenia obsahuje aj niekoľko procedúr: **beep** – spôsobí krátke cvaknutie, **BCDkonvert** – procedúra prevádza číslo z BCD kódu do ASCII, procedúra **datum1** sa používa na zistenie aktuálneho dátumu. Toľko k rezidentnej časti. Teraz nasleduje výpis inštalačnej a komunikačnej časti programu hodiny.asm. Táto časť bude uvoľnená z pamäte.

```
start:          xor ax,ax
                mov si,80h
                cmp byte ptr [si],0
                jne @1
                disp msg4
                jmp exit
@1:             mov cl,[si]
                xor ch,ch
                inc si
                cld
loadp:          lodsb
                cmp al,"/"
                je findp
                loop loadp
                cmp byte ptr param,1
                jne error1
                jmp instal
help:           jmp help1
instal:         jmp install
findp:          mov byte ptr param,1
                cmp byte ptr [si],"?"
                je help
                and byte ptr [si],0dfh
                cmp byte ptr [si],"O"
                je driver
                cmp byte ptr [si],"H"
                je help
```

```
                cmp byte ptr [si],"I"
                je instal
                cmp byte ptr [si],"D"
                je dateup
error1:         disp msg0
                jmp exit
dateup:         jmp dateup1
error:          mov ax,cs
                mov ds,ax
                mov ah,09h
                int 21h
                mov ax,4c01h
                int 21h
driver:         call je_tam
                jc @2
                mov dx,offset msg1
                jmp error
@2:             mov word ptr es:[@filename],0dedh
                push es
                pop ds
                xor ax,ax
                mov es,ax
                mov ax,ds
                cmp ax,es:[1ch*4+2]
                mov dx,offset msg2
                jne error
                push ds
                pop es
                mov ax,251ch
                mov dx,es:[staryint]
                mov ds,es:[staryint+2]
                int 21h
                push es
                mov ah,49h
                int 21h
                push cs
                pop ds
                disp msg3
                mov ax,4c00h
                int 21h
datedown:       call je_tam
                jc @3
                mov dx,offset msg1
                jmp error
@3:             mov es:[d_on_off],1
                mov ax,4c00h
                int 21h
dateup1:        cmp byte ptr [si+1],"+"
                je datedown
                cmp byte ptr [si+1],"-"
                je dateup2
                jmp error1
dateup2:        call je_tam
                jc @4
                mov dx,offset msg1
                jmp error
@4:             mov es:[d_on_off],0
                mov ax,offset datum
                mov di,ax
                call datum1
                mov ax,4c00h
                int 21h
je_tam          proc near
                mov bx,600h
                mov ax,cs
                cld
j4:             inc bx
                cmp ax,bx
                mov es,bx
                je end_s
                mov si,offset @filename
                mov di,si
                mov cx,16
                rep cmpsb
                or cx,cx
                jnz j4
                stc
                ret
end_s:          mov bx,0c800h
                mov ax,0f700h
                cld
j5:             inc bx
                cmp ax,bx
                mov es,bx
                je end_hled
                mov si,offset @filename
                mov di,si
                mov cx,16
                rep cmpsb
                or cx,cx
                jnz j5
                mov dx,cs
                cmp bx,dx
                je end_hled
                stc
                ret
end_hled:       clc
                ret
je_tam          endp
chyba:          disp msg5
                jmp exit
instal1:        call je_tam
                jc chyba
                mov ax,cs
                mov ds,ax
                mov ah,2ch
                int 21h
                sub dh,1
                mov byte ptr [sekcas],dh
                mov byte ptr [mincas],cl
                mov byte ptr [hodcas],ch
                mov ax,cs
                mov es,ax
                mov ax,offset datum
                mov di,ax
                call datum1
                mov dx,offset sprava
                mov ah,09h
                int 21h
bezpar:         mov ax,0
                mov bx,71
                mov dl,80
                mul dl
                add ax,bx
                mov dx,2
                mul dx
                mov word ptr [offadr],ax
                mov ax,1
                mov bx,70
                mov dl,80
                mul dl
                add ax,bx
                mov dx,2
                mul dx
                push cs
                pop ds
                mov word ptr [offadr1],ax
                mov ax,351ch
                int 21h
                mov word ptr staryint,bx
                mov word ptr staryint+2,es
                mov ax,251ch
                mov dx,offset novyint
                int 21h
                mov ax,word ptr ds:[2ch]
                mov es,ax
                mov ah,49h
                int 21h
                mov dx,(start-zaciatok)/16+16+1
                mov ax,3100h
                int 21h
address         proc near
                mov ax,0
                mov bx,0
                mov dl,80
                mul dl
                add ax,bx
                mov dx,2
                mul dx
                mov word ptr [offadr2],ax
                ret
address         endp
help1:          mov dx,offset navod
                mov ah,09h
                int 21h
exit:           mov ax,4c00h
                int 21h
msg0 db 'Clock - Nespravny parameter !!!',13,10,'$'
msg1 db 'Clock - Program nie je instalovany.',13,10,'$'
msg2 db 'Clock - Program nie je mozne uvolnit z pamäte.',13,10,'$'
msg3 db 'Clock - Program je uvolneny z pamäte.',13,10,'$'
msg4 db 'Clock - Skuste pouzit parameter /? za suborom.','$'
msg5 db 'Clock - Program bol uz raz instalovany.','$'
sprava          db 13,10,13,10,13,10
                db 13,10,'Installed Resident CLOCK  (c) 1996'
                db 13,10,'   Program by Peter Gasparovic'
```

```
                b 13,10,'    Made in Slovakia'
                db 13,10,'$'
                navod
                db 13,10,13,10,13,10
                db 13,10,'Pouzivanie programu:'
                db 13,10
                db 13,10,'hodiny.com /? /h -> zobrazi sa'
                db ' tato informacia'
                db 13,10,'hodiny.com /i -> prebehne'
                db ' instalacia programu'
                db 13,10,'hodiny.com /o -> odinstalovanie'
                db ' programu'
                db 13,10,'hodiny.com /d[+/-] ->'
                db ' [zapne/vypne] zobrazovanie datumu'
                db 13,10,'$'
CODE            ENDS
end             start1
```

Na návestí **start** sa začína obsluha spracovania príkazov v riadku, t. j. parametrov, ktoré zadáte za štartovací súbor a ktoré sprostredkúvajú či už inštalovanie, komunikáciu, alebo odinštalovanie rezidentnej časti programu. Presný tvar príkazového riadka je uložený v štruktúre PSP (Program Segment Prefix) na offsete 81H. Na offsete 80H je uložený počet znakov, ktoré príkazový riadok obsahuje. Ak sa za štartovacím súborom nenachádza nijaký parameter, program vám to oznámi chybovým hlásením **msg4**. Po použití parametra /? alebo /**h**, /**H** za štartovacím súborom sa zobrazí jednoduchý návod, ako program používať (pozri časť programu od návestia **help1**). Zadaním parametra /i prebehne inštalovanie programu do pamäte (návestie **instal1**). Najprv sa otestuje, či už náhodou nie je rezidentná časť inštalovaná, ak áno, potom zobraz chybové hlásenie **msg5**. V prípade, ak sa rezidentná časť ešte nenachádza v pamäti, pokračuje sa inštaláciou. Pomocou INT 21h, služba 2Ch sa zistí aktuálny čas, ďalej aktuálny dátum (pozri procedúru **datum1**) a vypíše sa krátka informácia o programe. Získaný dátum a čas si uložíme do dátovej oblasti rezidentnej časti. Ďalej pokračujeme výpočtom offsetu adresy vo videoram pre reťazec času i dátumu. Obidve hodnoty si opäť uchováme. A konečne sme pri INT 21, službe 35h, ktorá zistí adresu pôvodnej obsluhy prerušenia INT 1Ch. Za ňou nasleduje INT 21, služba 25h, ktorá sa používa na nastavenie novej obsluhy (v našom príklade to bude procedúra **novyint**). Na ešte lepšie využitie pamäte po inštalácii rezidentnej časti programu použijeme INT 21h, službu 49h, ktorá uvoľní blok pamäte pridelený kópii parametrov procesora príkazov obsluhy. Adresa bloku parametrov je zaznamenaná na adrese 2Ch v štruktúre PSP (Program Segment Prefix). Ešte sa vypočíta dĺžka rezidentnej časti a program sa končí s tým, že rezidentná časť zostane v pamäti.

Po zadaní parametra /**o** sa program odinštaluje z pamäte (návestie **driver**). Najprv sa zisťuje, či je už program inštalovaný, ak nie je, vypíše sa chybová správa **msg1**. V opačnom prípade sa pokračuje v odinštalovaní programu. Prerušeniu INT 1Ch sa vráti adresa jeho vlastnej obsluhy, uvoľní sa blok pamäte, ktorý obsadzuje rezidentná časť, a program sa končí tým, že sa na obrazovku vypíše správa **msg3**.

Ešte nám zostal posledný parameter, tým je /d. Pomocou tohto parametra môžete zakázať alebo povoliť zobrazovanie dátumu na obrazovke (návestie **dateup1**). Za parametrom /d ešte môžete uviesť znamienko plus (+) na zapnutie zobrazovania dátumu a znamienko mínus (-) na odstránenie dátumu z obrazovky. Opäť sa tu testuje prítomnosť rezidentnej časti v pamäti (pozri procedúru **je_tam**). Táto procedúra je jeden zo spôsobov, ako nájsť rezidentnú kópiu v pamäti. Pracuje tak, že sa porovnávajú časti programového kódu. Procedúra postupne prechádza jednotlivé segmenty v pamäti a porovnáva prvých 16 bajtov programového kódu, dokiaľ sa nenájde už inštalovaná rezidentná kópia. Porovnanie sa robí tak pre klasickú pamäť (0 – 640 KB), ako aj pre pamäť Upper memory (je to pamäťový priestor medzi 640 KB a 1 MB fyzickej pamäte). Procedúra vracia príznak CF nastavený na 1, ak kópia bola nájdená, inak CF=0.

Na záver tohto opisu chcem ešte dodať, že uvedený spôsob zobrazovania času a dátumu nie je určite jediný a že existujú aj oveľa jednoduchšie algoritmy a spôsoby, ako to dosiahnuť. Program môžete ďalej zlepšovať, doplniť o nové funkcie, záleží len na vás.

Preklad programu

Najprv opíšte celý program (tak ako je uvedený) napr. do súboru hodiny.asm. Spustiteľný COM súbor potom dostanete takto: Tasm.exe hodiny.asm a Tlink.exe hodiny.obj /t. Ak ste pri odpisovaní neurobili chybu, vytvorí sa súbor hodiny.com. Tak a to je nadnes skutočne všetko.

Nabudúce

Budeme pokračovať ďalej v téme o rezidentných programoch. Do tejto časti sa mi už nezmestil PSP (Program Segment Prefix), takže o ňom a iných zaujímavostiach nabudúce.

Literatúra

[1] Brandejs Michal: MS-DOS 6 - kompletní průvodce. Grada 1993.
[2] V. Boukal: BIOS IBM PC. Grada 1992.
[3] P. Heřman: Příručka systémového programátora. Tesla Eltos 1990.
[4] Jan Čáp: Rezidentní programy. Grada 1993.
[5] Zdeněk Kadlec: Tvorba rezidentních programů. Grada 1995.

Pätnásta časť: Terminate & Stay Resident (pokračovanie)

Dnes budeme pokračovať v téme o rezidentných programoch a ukážeme si prepínanie procesov, t. j. rezidentný program typu pop-up.

PSP (Program Segment Prefix)

Pri zavádzaní vykonateľného programu (.COM, .EXE) do pamäte operačný systém vždy vytvorí na začiatku oblasti pamäte, kam umiestni kód, tzv. predponu programového segmentu (Program Segment Prefix - PSP). Program Segment Prefix nie je súčasťou programového súboru. Obsahuje aktuálne údaje o stave operačného systému v okamihu zavádzania programu. Program môže tieto informácie využívať pri svojej práci (pozrite si rezidentný program z predchádzajúceho čísla PC REVUE). Program Segment Prefix má vždy dĺžku 100H (256 bajtov). Obsahuje položky uvedené v tabuľku 1.

Tabuľka 1: Program Segment Prefix –PSP

Offset	Veľkosť (v bajtoch)	Opis
+0	2	Inštrukcia INT 20 (ukončenie programu)
+2	2	Bázová adresa vrcholu pridelenej pamäte
+4	1	Rezervované
+5	5	Inštrukcia CALL na volanie dispečera služieb DOS-u
+0A	4	Vektor prerušenia 22H (ukončovacia adresa)
+0E	4	Vektor prerušenia 23H (obsluha Ctrl-Break)
+12	4	Vektor prerušenia 24H (obsluha chyby periférneho zariadenia)
+16	22	Rezervované pre DOS
+2C	2	Bázová adresa opisu parametrov interpretera príkazov
+2E	34	Rezervované na použitie operačného systému DOS
+50	2	Inštrukcia INT 21h
+52	1	Inštrukcia RETF
+53	2	Rezervované
+55	7	Rozšírenie riadiaceho bloku súboru číslo 1
+5C	9	Riadiaci blok súboru číslo 1
+65	7	Rozšírenie riadiaceho bloku súboru číslo 2
+6C	20	Riadiaci blok súboru číslo 2
+80	128	Disková vyrovnávacia pamäť

V prvých dvoch bajtoch je umiestnená **inštrukcia INT 20h**. Program môže ukončiť prácu skokom na adresu 0 v Program Segment Prefix. To však predpokladá, že segmentový register CS obsahuje bázovú adresu predpony, preto tento spôsob nie je veľmi vhodný. Na ukončenie programu je vhodnejšie použiť funkciu 4Ch, INT 21H.

Bázová adresa vrcholu pridelenej pamäte určuje, koľko pamäte operačný systém pre program vyhradil. Program väčšinou využíva túto informáciu, aby zistil, či má pridelený dostatok miesta pre svoje pracovné oblasti. Údaj však môže byť skreslený v prípade, že programu bola pridelená celá voľná operačná pamäť. Potom bázová adresa vrcholu pridelenej pamäte vždy obsahuje vrchol operačnej pamäte počítača. Napríklad: pre konfiguráciu s operačnou pamäťou 512 KB je bázová adresa 8000h, a to i v prípade, že na horných adresách pamäte je zavedený napr. rezidentný program. Preto je oveľa spoľahlivejšie získavať informácie o pamäti pomocou funkcií DOS-u. Pri zavádzaní programu typu .COM sa vždy vyhradí celá voľná operačná pamäť. Pre programy typu .EXE sa určuje veľkosť pridelenej pamäte pri vytváraní programového súboru spojovacím programom TLINK.

Inštrukcia CALL na volanie dispečera služieb DOS umožňuje volanie obsluhy prerušenia INT 21h. Tento spôsob však dovoľuje používať iba funkcie 00h až 24h. Program zadá číslo funkcie do registra CL a hodnoty ostatných registrov pripraví podľa opisu funkčného volania. Potom odovzdá riadenie na adresu 0005h v predpone programového segmentu inštrukciou CALL NEAR. Tento spôsob volania služieb prerušenia INT 21h je veľmi neobratný, v podstate je využiteľný iba pre programy typu .COM. Okrem toho, že položka slúži ako inštrukcia na volanie dispečera služieb, využíva sa ešte na uchovanie ďalšej informácie. Segmentová adresa dispečera je vždy upravená tak, aby jej relatívna adresa (06H v PSP) obsahovala počet bajtov pridelených programu v úseku pamäte dlhom 64 KB, začínajúcom prvým bajtom predpony programového segmentu. Táto informácia je platná iba do hodnoty FEF0h. Pokiaľ je rovná tejto hodnote, môže byť programu pridelenej viac pamäte než 64 KB. Program si potom musí získať informácie o pridelenej pamäti iným spôsobom.

Vektory prerušenia 22h, 23h a 24h sa ukladajú do predpony programového segmentu pri štarte programu. Po skončení programu sa obnovujú v poli vektorov prerušenia operačného systému DOS práve z týchto troch položiek.

Bázová adresa opisu parametrov interpretera príkazov ukazuje na začiatok kópie opisu parametrov, ktorú operačný systém pripravil pri spustení programu. Kópia opisu parametrov sa vytvára pre každý spúšťaný program znovu. Zavádzač operačného systému DOS ju umiestni do samostatného bloku pamäte. Tento blok sa uvoľní až pri

ukončení práce programu. Kópia opisu parametrov interpretera príkazov opisuje priradenia platné v operačnom systéme v okamihu spustenia programu. Opis parametrov je tvorený skupinou reťazcov ASCIIZ, t. j. reťazcov znakov v kóde ASCII ukončených bajtov s obsahom 00h. Celý opis je ukončený ďalším nulovým bajtom. Každé priradenie v systéme je opísané jedným reťazcom ASCIIZ v tvare meno=hodnota. Prvý reťazec opisu parametrov vždy obsahuje preradenie úplnej špecifikácie súboru, z ktorého bol zavedený aktuálny interpreter príkazov. Špecifikácia je priradená menu COMSPEC. Ďalšie reťazce zodpovedajú naposledy zadaným príkazom operačného systému PATH a PROMPT. Tieto priradenia využíva pri svojej práci operačný systém. Používateľ môže rozšíriť opis parametrov o ďalšie priradenia, ak použije príkaz operačného systému SET. Posledný reťazec opisu parametrov je ukončený dvoma nulami. Za opisom parametrov nasleduje slovo (WORD) obsahujúce hodnotu 0001h (počet ďalších odovzdávaných reťazcov) a reťazec ASCIIZ, ktorý obsahuje úplnú špecifikáciu súboru, z ktorého bol program spustený. Ten je ukončený opäť dvoma bajtmi s obsahom 00h. Celá oblasť kópie opisu parametrov nesmie byť dlhšia než 32 KB.

Je potrebné zdôrazniť, že kópia opisu parametrov je statická. To znamená, že zachytáva stav priradení platných v okamihu spustenia programu. Pokiaľ sa program stane rezidentným alebo bude spúšťať interpreter príkazov ako svoj podproces, môže byť zadaný nový príkaz PATH, SET alebo PROMPT. Potom nebude kópia opisu parametrov ďalej zodpovedať aktuálnemu stavu. Program môže informácie z kópie opisu parametrov využívať a môže ich aj meniť. Pritom však musí pamätať na to, že jeho podprocesy môžu údaje z opisu dediť.

Inštrukcie INT 21h a RETF zabezpečujú volanie obsluhy prerušenia 21h. Program môže týmto spôsobom používať všetky funkčné volania operačného systému DOS. Volanie sa vykoná tak, že program pripraví obsah všetkých registrov okrem AH podľa opisu požadovaného funkčného volania a potom namiesto inštrukcie INT zadá inštrukciu CALL typu FAR, ktorá odovzdá riadenie na adresu 50h v predpone programového segmentu (PSP).

Riadiaci blok súboru číslo 1 a riadiaci blok súboru č. 2, prípadne aj ich rozšírenia sú pripravené na použitie tradičnými funkčnými volaniami na prácu so súbormi. Na začiatku práce operačný systém pripraví do riadiaceho bloku 1 prvý parameter, ktorý bol zadný v príkazovom riadku za menom spusteného programu. Podobne riadiaci blok 2 bude obsahovať meno druhého parametra. Obidva riadiace bloky sú pripravené tak, že sú vyplnené položky zariadení, mená a prípony. To znamená, že môžu byť okamžite otvorené. Operačný systém tým uľahčuje programu prácu s parametrami. Je však potrebné poznamenať, že prvý riadiaci blok prekrýva druhý a druhý prekrýva časť implicitnej diskovej vyrovnávacej pamäte. Preto najlepšia cesta, ako využiť pripravené neotvorené riadiace bloky, je najprv ich presunúť do inej oblasti pamäte a až potom ich otvoriť. Ďalšie obmedzenie sa týka zadania adresárových ciest v parametroch programu. Ak budú zadané, potom neotvorené riadiace bloky nebudú platné (budú obsahovať nezmyselné údaje).

Disková vyrovnávacia pamäť

Pri každom zavádzaní programu systém automaticky nastaví adresu diskovej vyrovnávacej pamäte na prácu so súbormi pomocou tradičných funkčných volaní na adresu 80H vo vytvorenej predpone programového segmentu. Pri začatí práce obsahuje implicitná disková vyrovnávacia pamäť v prvom bajte (adresa 80H) počet znakov zadaných v príkazovom riadku za menom programu. Tieto znaky, tak ako boli zadané sú v PSP od adresy 81H. Pokiaľ príkazový riadok obsahuje znaky <, > alebo |, bude vo vyrovnávacej pamäti príkazový riadok iba od tohto znaku. Presmerovanie štandardného vstupu alebo výstupu a prepojenie je totiž záležitosťou operačného systému a používateľský program sa oň nemusí starať. Programy využívajú obsah vyrovnávacej pamäte po spustení na analýzu zadaných parametrov. Musí sa však počítať s tým, že prvá operácia čítania alebo zápisu pomocou tradičných volaní dáta vo vyrovnávacej pamäti znehodnotí. Program by preto mal buď zmeniť adresu diskovej vyrovnávacej pamäte, alebo príkazový riadok presunúť na iné miesto v pamäti, alebo spracovať všetky parametre ešte pred vykonaním prvého čítania alebo zápisu pomocou tradičného funkčného volania.

Operačná pamäť

Operačný systém DOS v priebehu práce neustále udržuje informácie o pridelenej a voľnej operačnej pamäti. Operačná pamäť sa procesom prideľuje po blokoch. Pamäťový blok je dynamická jednotka, jej dĺžka sa môže meniť. Blok predstavuje vždy súvislý adresný priestor v pamäti. Jeho veľkosť sa určuje v počte paragrafov pamäte. Jeden paragraf je najmenšia veľkosť, o ktorú sa môže blok zmeniť. Pre každý pamäťový blok, či už voľný alebo obsadený, sa udržuje hlavička bloku. Tá je dlhá 16 bajtov (jeden paragraf). Je uložená vždy tesne pred začiatkom bloku. Sú v nej údaje o dĺžke bloku a jeho stave (či je obsadený alebo voľný). V hlavičke bloku pamäte sa udržuje aj ukazovateľ na hlavičku ďalšieho pamäťového bloku. Takto sa neustále udržiava prehľad o voľných a obsadených blokoch v pamäti.

MCB – Memory Control Block (riadiaci blok pamäte). MCB je základná štruktúra správy pamäte v operačnom systéme DOS. Všetka pamäť priamo prístupná systému sa skladá z blokov s hlavičkou MCB. Ukazovateľ na prvý MCB je možné zistiť pomocou prerušenia INT 21h, služba 52h. Služba vráti v registroch ES:BX adresu na štruktúru interných premenných DOS-u. Na adrese ES:[BX-2] sa nachádza adresa prvého MCB. Niektoré rezidentné programy štruktúru interných premenných DOS-u využívajú na odinštalovanie tak, že v nej nájdu prvý MCB, potom prejdú celú reťaz a odstránia bloky patriace vlastnému PSP. Spoliehať sa však na správnosť údajov v nej môže byť riskantné. Ale risk je zisk, a preto aj v našom príklade bude zabudovaný tento spôsob odinštalovania rezidentného programu z pamäte. Nasleduje popis štruktúry MCB.

Adresa	Veľkosť v bajtoch	Opis
+00h	1	Typ – ak je 'M' (4Dh), potom nasleduje ďalší blok, ak je 'Z' (5Ah), potom ide o posledný blok určený na uvoľnenie
+01h	2	Vlastník – adresa PSP vlastníka bloku
+03h	2	Veľkosť – počet paragrafov bloku
+05h	3	Nepoužité
+08h	8	Od verzie DOS 4.0 pri bloku s PSP meno programu
+10h	?	Tu sa začína vlastný blok pamäte veľký (veľkosť*10h) bajtov. Paragraf na tejto adrese je hodnota, ktorú vráti funkcia INT 21, služba 48h

Adresa	Veľkosť v bajtoch	Opis
+00h	1	Typ – ak je 'M' (4Dh), potom nasleduje ďalší blok, ak je 'Z' (5Ah), potom ide o posledný blok určený na uvoľnenie
+01h	2	Začiatok – prvý použiteľný paragraf UMB bloku
+03h	2	Veľkosť – počet paragrafov bloku
+05h	3	Nepoužité
+08h	8	'UMB' alebo 'SM' (Systém Memory) – bloky takto označené sú použité na preklenutie pamätí ROM či VRAM, posledný blok v UMA musí byť z toho dôvodu označený 'SM'
+10h	?	Tu sa začína voľná pamäť UMB bloku, ktorá môže byť využívaná DOS-om. Pokiaľ ide o platný blok, je tu klasický MCB.

UMB (Upper Memory Block) – hlavička UMB je štruktúra podobná MCB, ale podporujúca správu pamäte v UMA. V MS-DOS 5.0 má každý platný UMB vlastný reťazec MCB blokov, rovnako ako konvenčná pamäť. Napriek tomu napríklad EMM386 v DR DOS predlžujú reťazec MCB v konvenčnej pamäti cez celú oblasť UMA. Nasleduje štruktúra hlavičky UMB.

Služby DOS-u

V nasledujúcom texte si opäť uvedieme niekoľko funkcií DOS-u, ktoré sa využívajú pri tvorbe rezidentných programov. INT 21h – služba 48h, INT 21h – služba 49h (uvoľnenie prideleného bloku pamäte), INT 21h – služba 4Ah, INT 21h – služba 62h (čítaj adresu PSP), INT 21h – služba 50h a 51h, INT 21h – služba 1Ah (nastav DTA), INT 21h – služba 2Fh (čítaj DTA).

INT 21h – služba 48h (pridelenie bloku pamäte) – služba pridelí programu blok pamäte, ktorej veľkosť je zadaná v paragrafoch v registri BX. Služba vráti v AX segmentovú časť adresy začiatku bloku (offset = 0). Pokiaľ došlo k chybe, je nastavený príznak CF na 1, register AX obsahuje kód chyby (7 – riadiaci blok je zničený, 8 – nedostatočná kapacita pamäte) a v registri BX je kapacita najväčšieho dostupného bloku v paragrafoch.

INT 21h – služba 49h (uvoľnenie prideleného bloku pamäte) – služba uvoľní blok pamäte, ktorý bol pridelený službou 48h. Bázová adresa začiatku uvoľňovaného bloku sa umiestni do registra ES. Pokiaľ dôjde k chybe, nastaví sa príznak CF a v registri AX vráti chybový kód (7 – riadiaci blok je zničený, 9 – chybná hodnota segmentu v registri ES). Bázová adresa uvoľňovaného bloku v registri ES musí byť nastavená presne na začiatok bloku, inak služba ohlási chybu. Volanie zmení údaje v hlavičke uvoľňovaného bloku a zaradí ho do reťazca voľných blokov v pamäti. Ak sú v pamäti dva voľné bloky bezprostredne za sebou, spojí ich systém do jedného bloku.

INT 21h – služba 4Ah (zmena veľkosti bloku pamäte) – služba umožňuje zmenšiť, prípadne zväčšiť už pridelený blok pamäte. V registri ES musí byť segmentová časť adresy bloku pamäte a v registri BX požadovaná nová dĺžka (v paragrafoch). Pokiaľ došlo k chybe, nastaví sa príznak CF na jedna a register AX obsahuje chybový kód (7 – riadiaci blok je zničený, 8 – nedostatočná kapacita pamäte, 9 – chybná hodnota segmentu v registri ES). V registri BX je kapacita najväčšieho dostupného bloku v paragrafoch.

INT 21h – služba 50h – nastaví adresu PSP (Program Segment Prefix) procesu. Segmentovú časť adresy je potrebné uložiť do registra BX.

INT 21h – služba 51h – je určená na oznámenie segmentovej časti adresy PSP (Program Segment Prefix) bežiaceho procesu. Hodnota je vrátená v registri BX.

INT 21h – služba 62h (čítaj adresu PSP) – zistí bázovú adresu PSP práve prebiehajúceho procesu. Bázová adresa je volajúcemu programu odovzdaná v registri BX. Táto funkcia sa využíva pri práci s programami typu EXE, ktoré sú rozčlenené do viacerých segmentov. Program môže ľubovoľne manipulovať so segmentovými registrami a nemusí uchovávať adresu predpony nastavenú pri spustení v segmentových registroch DS a ES.

INT 21h – služba 1Ah (nastav DTA) – umožňuje nastaviť začiatok diskovej vyrovnávacej pamäte pre prácu so súbormi na ľubovoľnú adresu vnútri prideleného bloku

pamäte. Adresa nového začiatku diskovej vyrovnávacej pamäte sa zadáva v registroch DS (bázová adresa DTA) a DX (relatívna adresa DTA). Segmentová adresa DS:DX musí ukazovať dovnútra aktuálneho bloku pamäte prideleného programu. Dĺžka diskovej vyrovnávacej pamäte sa priamo nezadáva, je však obmedzená koncom segmentu. Disková vyrovnávacia pamäť nemôže v žiadnom prípade presahovať cez hranice prideleného bloku pamäte. Musí byť vždy tvorená jedinou súvislou oblasťou pamäte. Pri spustení každého programu sa implicitne nastavuje disková vyrovnávacia pamäť na adresu 80h v PSP (Program Segment Prefix). Používa sa pre všetky vstupné a výstupné operácie vykonávané pomocou riadiacich blokov súborov až do chvíle, keď program použije funkciu 1Ah. Aby program zistil, kam je práve nastavený začiatok DTA, môže použiť službu 2Fh. Služba 1Ah nevracia nijaký chybový kód. Pokiaľ zadanú adresu DS:DX nie je možné ako začiatok diskovej vyrovnávacej pamäte použiť, spôsobí to väčšinou chybu až pri nasledujúcom pokuse o čítanie alebo zápis.

INT 21h – služba 2Fh (čítaj DTA) – odovzdá v registroch ES:BX súčasnú adresu pracovnej oblasti pre operácie čítania/zápisu disku s FCB (DTA).

Aktivovanie rezidentného programu pomocou HOTKEY

Aktivovanie rezidentného programu je proces, pri ktorom sa preruší vykonávanie práve bežiaceho programu a na základe definovanej udalosti sa riadenie odovzdá rezidentnej časti programu. Definovanou udalosťou sa rozumie napríklad stlačenie klávesu alebo kombinácie klávesov, vyvolanie špecifického prerušenia atď. Vlastná aktivácia rezidentného programu je často viazaná na niekoľko podmienok zároveň. Je to napríklad príznak aktivity DOS-u, kontrola vlastnej aktivity, kontrola semaforov atď. Rezidentný program vlastne čaká až do chvíle, keď budú všetky tieto náležitosti na správne aktivovanie splnené. Vyhnete sa tak možnému pádu systému.

Prepnutie aktívneho procesu

Vzhľadom na to, že operačný systém MS DOS je výhradne jednoúlohový, je potrebné, aby pri aktivovaní rezidentného programu došlo ku korektnému prepnutiu medzi rezidentným programom a práve spustenou aplikáciou. Korektným prepnutím sa rozumie vykonanie niekoľkých dôležitých krokov tak, aby v prípade nového prepnutia naspäť do prerušeného programu mohol ten opäť pokračovať v behu z rovnakého stavu, v akom sa nachádzal pred prerušením. Samotný operačný systém nemá nijaký aparát na prepínanie jednotlivých aplikácií alebo rezidentných programov. Z toho vyplýva, že prepnutie medzi rezidentným programom a práve bežiacou aplikáciou je plne v réžii programátora a jeho rezidentného programu. Nasledujúci opis krokov spĺňa podmienky na správne prepnutie medzi rezidentným programom a prerušenou aplikáciou z hľadiska DOS-u. Kroky, ktoré je potrebné vykonať pri prepnutí medzi procesmi:

1. Čo najskôr nastaviť obsah registrov SS a SP tak, aby ukazovali na zásobník rezidentného programu, a uschovať ich pôvodné hodnoty, ktorými naplníme registre SS a SP opäť pred ukončením danej obsluhy prerušenia.
2. Pokiaľ rezidentný program pracuje so súbormi, musí zistiť adresu PSP aktuálnej aplikácie a nastaviť hodnotu PSP na vlastnú. Adresu PSP aktuálnej aplikácie si opäť treba uchovať na neskoršie použitie.
3. Pri práci so súbormi, keď sa využíva systém FCB alebo handle, je potrebné uschovať a nastaviť vlastnú DTA (Disk Transfer Area).
4. Treba uchovať a nastaviť tie vektory prerušenia, ktorých obsluha je potrebná iba pri behu jadra rezidentného programu. Tento krok sa väčšinou využíva na nastavenie vektorov prerušenia kritickej chyby (INT 24h) a obsluhy stlačenia klávesov CTRL-Break (INT 1Bh).
5. Vykonanie vlastnej akcie rezidentného programu.

Po vykonaní činnosti jadra rezidentného programu treba vykonať rovnakú postupnosť krokov, ale v opačnom poradí. Systém musíme uviesť do pôvodného stavu. To znamená, že musíme vrátiť adresy DTA a PSP a obnoviť obsah registrov SS a SP. Iba po vykonaní týchto krokov môže váš rezidentný program pracovať spoľahlivo a korektne s operačným systémom MS DOS.

Praktický príklad

Ukážeme si rezidentný program typu pop-up, ktorý po stlačení klávesov ATL+A zastaví práve bežiacu aplikáciu a spustí náš proces (procedúru tabuľka). Po stlačení klávesu ESC pokračuje vo vykonávaní prerušenej aplikácie. Program podobne ako v predošlej časti obsahuje inštalovanie (procedúra dos_install), aj odinštalovanie (procedúra deallocate) rezidentnej časti.

```
DOSSEG
.MODEL LARGE
EXTRN     @aplikacia$qv :proc

.DATA
EXTRN     _active_key : WORD
text      db 'ASCII & COLOR TABLE (c) 1997'
          db 32,'Copyright Universal systems'

text1     db 'Press any key'

db 32,'for return.'
.CODE
public    _dos_install
public _tabulka

; stary a novy PSP
_savepsp            DW 0
_mypsp              DW 0

; stary a novy DTA
_intdta     DD 0
_mydta      DD 0


; pointer na disk_free promennu

df_es       DW ?

; povodna obsluha int 13h
_old_disk           DD 0
; povodna obsluha int 16h
_old_vector         DD 0
; povodna obsluha int 1bh
_old1b               DD 0
; povodna obsluha int 28h
_old_28             DD 0

; rezidentne jadro aktivne?
_active             DB 0
; int 13 aktivny
_diskflag           DB 0
_rq

; premenna na uschovanie
; cisla BIOS-funkcie
func                DB ?

; stack preruseneho programu
ss_r                DW ?
sp_r                DW ?
; stack z instalacneho programu
ss_old              DW ?
sp_old              DW ?

r_x       PROC  far
pushf
cmp       [_diskflag],0
jne       near ptr y1
call far ptr dos_free
jne       near ptr y1
call far ptr resident
mov       [_rq],0
jmp       near ptr y2
y1:       mov       [_rq],1
y2:       popf
retf
r_x       ENDP
; Nova obsluha int 16h
_intr_exec          PROC FAR
          push ds
          pushf
          push bp
          mov bp,DGROUP
          mov ds,bp
          pop bp
          cmp [_active],0
          jne l2
          cmp ah,1
          jbe ie1
          cmp ah,10h
          je ie1
          cmp ah,11h
          je ie1
          cmp ax,0F300h
          jne l21
          mov ax,1234h
          jmp l24
l21:      cmp ax,0f400h
          jne l2
          call far ptr change
          je l23
l22:      mov ax,1
l24:      popf
          jmp l1
l23:      mov dx,word ptr [_old_vector]
          mov ds,word ptr [_old_vector+2]
          mov ax,2516h
          int 21h
          mov dx,word ptr [_old_disk]
          mov ds,word ptr [_old_disk+2]
          mov ax,2513h
          int 21h
          mov dx,word ptr [_old_28]
          mov ds,word ptr [_old_28+2]
          mov ax,2528h
          int 21h
          mov ax,[_mypsp]
          call far ptr deallocate
          popf
          xor ax,ax
          jmp l1
l2:       popf
          pushf
          call [_old_vector]
l1:       pop ds
          retf 2
ie1:      cmp [_rq],0
          je near ptr ie2
          call r_x
ie2:      mov [func],ah
          or  ah,ah
          jz ie3
          cmp  ah,10h
          jne  near ptr ie5
ie3:      sti
          pushf
          cli
          call [_old_vector]
          cmp  ax,[_active_key]
          je   ie4
          popf
          pop ds
          iret
ie4:      call r_x
l32:      mov  ah,[func]
          jmp ie3
ie5:      dec [func]
          pushf
          cli
          call [_old_vector]
          jz  ie6
          cmp ax,[_active_key]
          jne ie6
          mov ah,[func]
          pushf
          cli
          call [_old_vector]
          call r_x
ie6:      pop  ds
          pop  ds
          retf 2
_intr_exec          ENDP
resident  PROC FAR
          mov [_active],1
          cli
          mov [ss_r],ss
          mov [sp_r],sp
          mov ss,[ss_old]
          mov sp,[sp_old]
          sti
          push ax bx cx dx es ds si di
          push bp
          pushf
          mov ah,2fh
          mov word ptr [_intdta],bx
```

```
        mov word ptr [_intdta+2],es
        mov ah,1ah
        mov dx,word ptr [_mydta]
        mov ds,word ptr [_mydta+2]
        int 21h
        mov ah,51h
        int 21h
        mov [_savepsp],bx
        mov bx,[_mypsp]
        mov ah,50h
        int 21h
        mov ax,351bh
        int 21h
        mov word ptr [_old1b],bx
        mov word ptr [_old1b+2],es
        mov ax,251bh
        push cs
        pop ds
        mov dx,offset _ctrlbreak
        int 21h
        push bp
        mov bp,DGROUP
        mov ds,bp
        pop bp
        call @aplikacia$qv
        mov ax,251bh
        mov dx,word ptr [_old1b]
        mov ds,word ptr [_old1b+2]
        int 21h
        mov bx,[_savepsp]   ; obnov PSP
        mov ah,50h
        int 21h
        push ds
        mov ah,1ah
        mov dx,word ptr [_intdta]
        mov ds,word ptr [_intdta+2]
        int 21h
        pop ds
        popf
        pop bp
        pop di si ds es dx cx bx ax
        cli
        mov ss,[ss_r]
        mov sp,[sp_r]
        sti
        mov [_active],0
        retf
        resident  ENDP
        deallocate proc far
        push si di
        mov di,ax
        mov ah,52h
        int 21h
        mov si,word ptr es:[bx-2]
@4:     mov bx,si
        mov es,bx
        mov bx,1
        cmp word ptr es:[bx],di
        jne @5
        mov ah,49h
        inc si
        mov es,si
        dec si
        int 21h
@5:     xor ax,ax
        mov bx,si
        mov es,bx
        xor bx,bx
        cmp byte ptr es:[bx],'Z'
        je  @6
        mov bx,si
        mov es,bx
        mov bx,3
        mov ax,word ptr es:[bx]
        add ax,si
        inc ax
@6:     mov si,ax
        or ax,ax
        jne @4
        pop di si
        ret
deallocate endp
change  PROC    FAR
        push    cs
        pop dx
        mov ax,3516h
        int 21h
        mov ax,es
        cmp bx,offset _intr_exec
        jne c1
        cmp dx,ax
        jne c1
        mov ax,3513h
        int 21h
        mov ax,es
        cmp bx,offset _new_disk
        jne c1
        cmp dx,ax
        jne c1
        mov ax,3528h
        int 21h
        mov ax,es
        cmp bx,offset _new_28
        jne c1
        cmp dx,ax
c1:     ret
change  ENDP
        _ctrlbreak          PROC FAR
        iret
        _ctrlbreak          ENDP
        _new_disk           PROC FAR
        mov       [_diskflag],1
        pushf
        call [_old_disk]
        mov       [_diskflag],0
        retf 2              _new_disk
        ENDP
        _new_28   PROC FAR
        pushf
        call [_old_28]
        pushf
        cmp [_active],0
        jne near ptr n1
        cmp [_rq],0
        je  near ptr n1
        cmp [_diskflag],0
        jne near ptr n1
        call far ptr resident
        mov [_rq],0
n1:     popf
        retf 2
_new_28 ENDP
dos_free  PROC FAR
        push es bx
mov     es,[df_es]
mov     bx,[df_bx]
cmp     byte ptr es:[bx],0
pop     bx es
ret     dos_free  ENDP
        _dos_install        PROC FAR
        push si
        mov  si,ss
        mov  [ss_old],si
        mov  si,sp
        mov  [sp_old],si
        pop  si
        push ax es dx bx
        push ds
        push cs
        pop  ds
        mov  ax,3516h
        int  21h
        mov  word ptr [_old_vector],bx
        mov  word ptr [_old_vector+2],es
        mov  dx,offset _intr_exec
        mov  ax,2516h
        int  21h
        mov  ax,3513h
        int  21h
        mov  word ptr [_old_disk],bx
        mov  word ptr [_old_disk+2],es
        mov  dx,offset _new_disk
        mov  ax,2513h
        int  21h
        mov  ax,3528h
        int  21h
        mov  word ptr [_old_28],bx
        mov  word ptr [_old_28+2],es
        mov  dx,offset _new_28
        mov  ax,2528h
        int  21h
        pop  ds
        mov  ah,51h
        int  21h
        mov  [_mypsp],bx
        mov  ah,2fh
        int  21h
        mov  word ptr [_mydta],bx
        mov  word ptr [_mydta+2],es
        mov  ah,34h
        int  21h
        mov  [df_es],es
        mov  [df_bx],bx
        pop  bx dx es ax
retf    _dos_install        ENDP
        vypis     proc near
x10:    movsb
        stosb
        loop x10
        ret
vypis   endp
-----------------------------------------------
transfer2 proc near
push ax sub cx,cx
x0:     sub dx,dx
        div bx
        push dx
        inc cx
        test ax,ax
        jnz x0
        mov ax,word ptr[si+1]
        cmp ax,cx jbe x2
        mov bx,cx sub ax,cx mov cx,ax mov
        dl,48
x1:     mov es:[di],dl
        mov al,[si+0]
        mov es:[di+1],al
        inc di
        inc di
        loop x1
        mov cx,bx
x2:     pop dx
        or dl,'0' cmp dl,'9'
        jbe x3
        add dl,7
x3:     mov es:[di],dl
        mov al,[si+0]
        mov es:[di+1],al
        inc di
        inc di
        loop x2
        pop ax
        ret
color   db ?
null    dw ?
transfer2 endp
poloha2   proc near
        mov dl,80
        mul dl
        add ax,bx
        mov dx,2
        mul dx
        ret
poloha2   endp
uschovaj  proc near
        push ds
        push es
        mov ax,seg screen
        mov es,ax
        mov di,offset screen
        mov cx,80*25*2
        mov ax,0b800h
        mov ds,ax
        xor si,si
skok2:  lodsb
        stosb
        loop skok2
        pop es
        pop ds
        ret
uschovaj  endp
obnov   proc near
        push ds
        push es
        mov ax,0b800h
        mov es,ax
        xor di,di
        mov cx,80*25*2
        mov si,offset screen
        mov ax,seg screen
        mov ds,ax
skok3:  lodsb
        stosb
        loop skok3
        pop es
        pop ds
        ret
obnov   endp
-----------------------------------------------
cls2    proc near
        xor di,di
        mov cx,80*25
        rep stosw
        ret
cls2    endp
screen  db 4000 dup(0)
_tabulka  proc far
        call uschovaj
        mov ax,0b800h
        mov es,ax
```

```
          mov ax,7*256+32
          call cls2
          mov ax,0 ;Riadok.
          mov bx,12 ;Stlpec.
          call poloha2
          mov di,ax
          mov si,offset text
          mov cx,56 ;Dlzka textu.
          mov al,7 ;Farba.
          call vypis
          mov ax,24 ;Riadok.
          mov bx,26 ;Stlpec.
          call poloha2
          mov di,ax
          mov si,offset text1
          mov cx,25 ;Dlzka textu.
          mov al,128+7 ;Farba.
          call vypis
          mov ax,2 ;Riadok.
          mov bx,4 ;Stlpec.
          call poloha2
          mov di,ax
          mov ax,3 ;Riadok.
          mov bx,4 ;Stlpec.
          xor dh,dh
          mov dl,0
          mov si,offset color
          mov cx,3 ;Zarovnanie
          mov [si+1],cx
          mov cx,256
x6:       push cx ax dx bx
          mov es:[di],dl
          inc di
          inc di
          mov al,'-'
          mov es:[di],al
          inc di
          inc di
          mov [si+0],dl
          mov al,dl
          mov bx,10 ;Sustava
          call transfer2
          mov cl,179
          mov es:[di],cl
          inc di
          inc di
          pop bx dx ax cx
          inc dl
          push ax dx
          call poloha2
          mov di,ax
          pop dx ax
          inc ax
          cmp ax,24
          jz posun
navrat:   loop x6
          jmp koniec
posun:    add bx,6
          mov ax,2
          jmp navrat
koniec:   mov ax,0
          int 16h
          call obnov
          retf
_tabulka  endp
          END
```

Opis modulu popup.asm

Program používa model LARGE. Všimnite si, že budeme používať (volať) procedúru z jazyka C – **EXTRN @aplikacia$qv:proc**. Na začiatku je definovaných niekoľko premenných, ktoré budeme ďalej využívať. Po premenných nasleduje **procedúra r-x**, ktorá vykonáva testy bezpečnosti, a pokiaľ sú pozitívne, spustí procedúru **rezident**. V procedúre sa najprv testuje aktivita INT 13h, ďalej naleduje test na voľný DOS (INT 28h). Tento vektor prerušenia používajú rezidentné programy ako jeden z niekoľkých vstupných bodov. To znamená, že sa kontroluje stlačenie aktivačného klávesu. V prípade, že je zistená požiadavka na aktiváciu, nastaví sa príznak aktivácie jadra rezidentného programu. Ak program neprejde cez testy, uloží sa do premennej **rq** hodnota 1, čo znamená, že nie je možné aktivovať procedúru rezident. V opačnom prípade **rq** obsahuje nulu.

Nasleduje procedúra **intr_exec**, čo je vlastne nová obsluha prerušenia INT 16h. Na začiatku procedúry je test na aktivitu rezidenta. Ak je proces práve aktívny, vykoná sa iba stará obsluha INT 16h. Do tejto novej obsluhy sú začlenené aj testy na prítomnosť rezidentného programu v (inštrukcia CMP AX,0F300h), ako aj odinštalovanie programu z pamäte (inštrukcia CMP AX,0F400h).

Procedúra **rezident** má na starosti aktivovanie vlastného procesu, t. j. procesu, ktorý sa spustí po stlačení kombinácie klávesov ALT+A. Na túto procedúru skáče program, ak prejde bezpečnostnými testmi. Na začiatku procedúry sa nastaví premenná active na 1 a zakáže sa prerušenie. Nastavia sa nové hodnoty registrov SS a SP. Povolí sa prerušenie a na zásobník sa uložia všetky potrebné registre. Ďalej sa uchová a nastaví DTA (Disk Transfer Area) aj PSP (Program Segment Prefix). Presmeruje sa vektor prerušenia 1Bh na vlastnú obsluhu (procedúra **ctrlbreak**). Po týchto všetkých nastaveniach môžeme konečne zavolať (odštartovať) náš proces (inštrukcia **call @aplikacia$qv**). Po ukončení procesu sa nastaví INT 1Bh na pôvodnú obsluhu, obnoví sa PSP a DTA. Ďalej sa obnovia registre a zásobník sa uvedie do pôvodného stavu. Nakoniec sa premenná **active** nastaví na hodnotu nula – proces je ukončený.

Procedúra **deallocate** sa používa na odinštalovanie programu z pamäte. Procedúra odstráni z pamäte všetky alokované bloky patriace danému PSP.

Procedúra **change** testuje, či niekto (iný rezidentný program) neprepísal vektory, ktoré používa náš program. Change vracia príznak NZ, ak došlo k zmene nami využívaného vektora. Procedúra **ctrlbreak** je nová obsluha prerušenia INT 1Bh. Procedúra **new_disk** nastavuje premennú **diskflag**, ak je BIOS v diskovej operácii. Je to nová obsluha prerušenia INT 13h. Nasleduje procedúra **new_28** je to nová obsluha prerušenia INT 28H. Aj z tohto miesta môže byť procedúra rezident aktivovaná. Na začiatku procedúry sa zavolá pôvodná obsluha INT 28h, testuje sa, či je rezident aktívny (inštrukcia cmp [active],0), či bola žiadosť o aktiváciu (inštrukcia cmp [rq],0) a či bola disková funkcia aktívna (inštrukcia cmp [_diskflag],0). Ak sú všetky tieto testy negatívne, vykoná sa skok na procedúru rezident. Po návrate z procedúry rezident ešte premennú **rq** nastavíme na hodnotu nula – zrušenie žiadosti o aktiváciu.

Procedúra **dos_free** sa obracia na DOS o informáciu bezpečnosti. Vráti príznak Z, ak je vstup do DOS-u bezpečný. Procedúra dos_install sa používa na inštalovanie nových vektorov prerušenia (INT 13h, INT 16h, INT 28h), uchovanie DTA a PSP a vytvorenie smerníka na DOS-Free. S procedúrami vypis, cls2 a tabuľka ste sa už v tomto seriáli stretli, preto ich nebudem opisovať.

Modul popupx.cpp

Na tomto module niet čo vysvetľovať, sprostredkúva inštalovanie, odinštalovanie programu z pamäte a obsahuje aj procedúru aplikácia, ktorá je volaná z modulu v assembleri (pozri procedúru rezident). Myslím, že pre tých skúsenejších by nemal byť problém prepísať tento modul do assemblera. Modul v jazyku C je výhodnejší pre krátky zdrojový kód.

```
#include <stdio.h>
#include <dos.h>
#include <alloc.h>
#define INIT_KEY 0x1e00 /* Alt-A */
#define STKLN 1024 /* velkost zasobnika */

/* rutina pro nastaveni pointera
obsluhy preruseni, PSP, DTA a DOS_FREE ptr */
extern "C" {void dos_install(void);};
extern "C" {void tabulka(void);};

/*globalne premenne*/
unsigned _stklen    = STKLN,
active_key = INIT_KEY;

union REGS reg;
struct SREGS sreg;

void aplikacia(){
asm {
/*Vypni kurzor*/
mov ah,01h;
mov cx,20h*256+0;
int 10h;
//Vypni kurzor mysi
mov ax,2;
int 33h;
}
/*Volaj hlavnu cast*/
tabulka();
//Zapni kurzor mysi
asm {
mov ax,1;
int 33h;
/*Zapni kurzor*/
mov ah,01h;
mov cx,12h*256+14h;
int 10h;
}
}
void main() {
puts("\nASCII tabulka Pop-up -> resident.");

reg.x.ax=0xf300;
int86(0x16,&reg,&reg);
if(reg.x.ax==0x1234){
reg.x.ax=0xf400;
int86(0x16,&reg,&reg);
if(reg.x.ax)
puts("Program nie je mozne odinstalovat\n");
else  puts("
Program je odinstalovany\n");
}else{
puts("Rezidentna cast bola instalovana.");
puts("
Aktivace ALT-A\n");
/* instaluj nove vektory prerusenia*/
dos_install();
/*ukonci a zostan v pamati*/
keep(1,(int)(*(unsigned *)MK_FP(_psp,2)-_psp-
(unsigned)(farcorele
}
```

O tom, ako vytvoriť projektový súbor sa dozviete práve tu. Spustite program **BC.EXE** (najlepšie BORLAND C++ verziu 3.1, pretože na tejto verzii bol program odladený). V menu kliknite na **PROJECT** a potom na **OPEN PROJECT**. Otvorí sa okno **Open Project File** a od vás sa očakáva, že zadáte názov projektu (napríklad: **rezident.prj**). Po zadaní mena projektu sa v spodnej časti obrazovky otvorí okno aj s vami zadaným názvom projektu. Teraz pomocou klávesu **Insert** vložte najprv program v jazyku C (popupx.cpp), ďalším stlačením modul v assembleri (popup.asm) a nastavte prepínač "·" na program v jazyku C. Ešte skontrolujte pomocou klávesovej skratky **CTRL+O**, či je nastavený správny prekladač pre jazyk C a pre modul v assembleri. Ak nie je, nastavte ho kliknutím na niektorú z volieb v boxe **Project File Translator**. V zásade však platí, že program bc.exe (verzia 3.1) si podľa prípony súboru nastaví všetky potrebné parametre automaticky. Ak však chcete nejaký parameter prekladu zmeniť, musíte to urobiť v okne **Local Option**. Ešte kliknite v hlavnom menu na **OPTIONS/COMPILER/C++ OPTIONS** a v "Use C++ Compiles" nastavte prepínač na **CPP extension**, kliknite na OK a uložte nastavenie **OPTIONS/SAVE**. Teraz už môžete spustiť kompiláciu. Kliknite na menu **Compile** a nastavte sa na položku **Build all**. Ak ste pri odpisovaní zdrojového textu neurobili chyby a všetko je správne nastavené, vytvorí sa súbor s príponou **rezident.exe**.

Literatúra

[1] M. Brandejs: MS-DOS 6 - kompletní průvodce. Grada 1993.
[2] V. Boukal: BIOS IBM PC. Grada 1992.
[3] P. Heřman: Príručka systémového programátora. Tesla Eltos 1990.
[4] J. Čáp: Rezidentní programy. Grada 1993.
[5] Z. Kadlec: Tvorba rezidentních programů. Grada 1995.
[6] D. Jurgens: HelpPC 2.10.
[7] ABSHelp 2.05, ABSoft Olomouc.

Šestnásta časť: Terminate & Stay Resident (pokračovanie)

V nasledujúcom texte si ešte doplníme informácie o niektorých prerušeniach, o ktorých som sa v predchádzajúcom čísle nezmienil. Pôjde o INT 2Fh (komunikácia s rezidentnými procesmi), INT 23h (kláves CTRL-BREAK), INT 24h (obsluha kritickej chyby), INT 28h (voľný DOS), INT 2Dh – AMIS (Alternate Multiplex Interrupt Specification).Tieto prerušenia sa veľmi často využívajú pri tvorbe rezidentných programov.

INT 2FH (komunikácia s rezidentnými procesmi)
Pomocou tohto prerušenia je v operačnom systéme DOS zaistená možnosť komunikácie medzi dvoma procesmi. Používateľský program môže použiť prerušenie 2Fh vždy, keď chce riadiť, modifikovať alebo kontrolovať činnosť niektorého programu, ktorý je rezidentne uložený v operačnej pamäti. Program obsluhy prerušenia INT 2Fh je štandardne upravený na komunikáciu so systémovými rezidentnými procesmi PRINT, ASSIGN a SHARE. Okrem toho je prerušenie 2Fh vhodným prostriedkom pre programátorov, ktorí chcú používať vlastné používateľské rezidentné procesy a mať možnosť s nimi komunikovať. Toto, samozrejme, vyžaduje modifikáciu programu obsluhy prerušenia 2Fh.

Tabuľka 1

Číslo procesu	Rezidentný program
00h	rezervované
01h	PRINT
02h	ASSIGN
03h až 0Fh	rezervované
10h	SHARE
11h až 7Fh	rezervované

Každý rezidentný proces musí mať pridelené **číslo procesu** (pozri tabuľku 1), ktorým sa pri volaní prerušenia 2Fh volajúci program na rezidentný proces odkazuje. Systémovým rezidentným procesom sú vyhradené čísla 00h a 7Fh.

Používateľským rezidentným procesom sú vyhradené čísla procesov 80 až FFh. Číslo procesu sa pri volaní prerušenia 2Fh zadáva v registri AH. Ostatné registre môžu obsahovať ďalšie parametre odovzdávané programu obsluhy prerušenia.

INT 2FH, služba 01h (program PRINT) – pomocou tejto služby je možné komunikovať s programom PRINT. Funkcia, ktorú požadujeme, sa zadáva do registra AL (pozri tabuľku 2).

Tabuľka 2

Reg. AL	Opis
00h	zistí stav procesu
01h	zaradenie súboru do tlačového frontu
02h	vypustenie súboru z tlačového frontu
03h	zrušenie všetkých požiadaviek na tlač
04h	pozastavenie práce PRINT
05h	pokračovanie v práci PRINT

V prípade, že požadovaná funkcia nemôže byť vykonaná, je pri návrate z prerušenia nastavený príznak CF a register AX obsahuje chybový kód (pozri tabuľku 3), ktorý bližšie určuje povahu chyby. Ak je funkcia vykonaná bez chýb, CF je pri návrate z prerušenia nastavený na 0.

Chybový kód je vyvolaný, ak volajúci program požaduje prístup k súboru, ku ktorému je prístup zakázaný. K tejto situácii môže dôjsť, iba ak je v činnosti systém zdieľania súborov SHARE. Vráťme sa teraz však k opisu funkcií.

Pomocou **funkcie 00h** môže používateľský program zistiť, či je rezidentná časť programu PRINT inštalovaná

Tabuľka 3

Chybový kód	Význam
01h	požadovaná funkcia neexistuje
02h	súbor nebol nájdený
03h	cesta nebola nájdená
04h	priveľa otvorených súborov
05h	prístup odmietnutí
08h	tlačový front je plný
09h	program práve pracuje
0Ch	pridlhé meno
0Fh	neznáme zariadenie

v operačnej pamäti. Pri návrate z prerušenia je informácia o stave programu PRINT v registri AL. AL=00h – program PRINT nie je inštalovaný, je možné ho inštalovať, AL=01h – program PRINT nie je inštalovaný, nie je možné ho inštalovať (napr.: málo pamäte), AL=FFh – program PRINT je inštalovaný.

Funkcia 01h zaradí do frontu požiadaviek na tlač súborov novú požiadavku. Dvojica registrov DS:DX musí pri vyvolaní prerušenia obsahovať segmentovú adresu ukazovateľa na meno súboru, ktorého tlač je požadovaná. Ukazovateľ na meno súboru je zostavený z piatich bajtov. Prvý bajt musí mať hodnotu nula, ďalšie dva bajty obsahujú relatívnu a bázovú adresu reťazca znakov v tvare ASCIIZ (reťazec znakov ukončený bajtom s obsahom 0). Reťazec ASCIIZ musí obsahovať celú špecifikáciu súboru vrátane mena zariadenia a kompletnej cesty k súboru. Meno nesmie obsahovať znaky hviezdičkovej a otáznikovej konvencie. Takto zadaný súbor je zaradený na koniec tlačového frontu. Prípadná chyba je indikovaná nastavením príznaku CF.

Funkcia 02h vyradí z frontu požiadaviek na tlač požiadavku na tlačenie zadaného súboru. Registre DS:DX musia obsahovať segmentovú adresu reťazca ASCIIZ, ktorý obsahuje úplnú špecifikáciu súboru, ktorý bude vyradený z frontu požiadaviek. Meno súboru môže obsahovať znaky hviezdičkovej konvencie. Potom budú z frontu vyradené všetky požiadavky na tlač súborov, ktorým zadané meno zodpovedá. Prípadná chyba je indikovaná nastavením príznaku CF.

Funkcia 03h zruší všetky požiadavky na tlač súborov, ktoré sú zaradené vo fronte požiadaviek. Výstup súboru, ktorý sa práve tlačí, je zastavený a na tlačiarni sa vypíše správa o zrušení celého tlačového frontu.

Funkcie 04h umožňuje používateľskému programu zmraziť front požiadaviek na tlač. Výstup práve tlačeného súboru sa zastaví. To poskytuje možnosť prehľadávať a modifikovať front požiadaviek. Pri návrate z prerušenia je v registri DX zaznamenaný počet márnych pokusov programu PRINT o tlač posledného vystupujúceho znaku. Ak je DX nula, nebola zaznamenaná žiadna chyba. Dvojica registrov DS:SI ukazuje na začiatok tlačového frontu. Tlačový front je zložený z položiek. Každá položka je dlhá 64 bajtov a obsahuje úplnú špecifikáciu súboru, ktorý má byť vytlačený. Špecifikácie súborov musia byť uložené ako reťazce ASCIIZ. Po použití tejto funkcie zostane tlačový front zmrazený tak dlho, dokiaľ nie je použitá iná funkcia prerušenia 2Fh komunikujúca s programom PRINT. Výstup súboru, ktorého tlač bola prerušená, pokračuje od miesta prerušenia bez straty znaku.

Funkcia 05 nevykonáva nijakú špeciálnu činnosť, iba vráti volajúcemu programu v registri AX chybový kód. Používa sa na obnovenie normálneho spracovania tlačového frontu po použití funkcie 04.

INT 2FH, služba 06h (program ASSIGN) – služba zistí stav programu ASSIGN. Pred vyvolaním prerušenia musí register AL obsahovať nulu. Pri návrate je informácia o stave programu v registri AL. Návratový kód 00h znamená, že rezidentná časť programu ASSIGN nie je inštalovaná v operačnej pamäti. Kód 01h znamená, že rezidentná časť nie je inštalovaná, ani nie je možné je inštalovať (napr.: z dôvodu nedostatku operačnej pamäte). Návratový kód FFh znamená, že rezidentná časť programu ASSIGN je inštalovaná v pamäti.

INT 2FH, služba 010h (program SHARE) – služba zistí stav programu SHARE (podobne ako pri programe ASSIGN). Pred vyvolaním prerušenia musí byť v registri AL nula. Návratový kód má rovnaký význam ako v programe ASSIGN.

Komunikácia s používateľskými rezidentnými procesmi
Pri programovaní používateľských rezidentných procesov, ktoré budú komunikovať s inými procesmi pomocou prerušenia 2Fh, sa odporúča v čo najväčšej miere dodržovať pravidlá, ktorými sa riadi komunikácia so systémovými rezidentnými programami. To znamená predovšetkým používať na odovzdávanie parametrov a informácií rovnaké registre ako štandardný program obsluhy prerušenia 2Fh a používať rovnaké návratové kódy. Pred vyvolaním prerušenia by register AH mal obsahovať číslo rezidentného procesu a register AL by mal obsahovať požadovanú funkciu. Prípadný chybový kód by mal pri návrate z obsluhy prerušenia byť v registri AX. Dôležité je, aby na komunikáciu s každým používateľským rezidentným procesom bola zabezpečená funkcia zisťujúca stav procesu. Tá by mala mať štandardnú podobu analogickú so systémovými procesmi. Mala by byť volaná hodnotou nula v registri AL a vrátená informácia by takisto mala byť v registri AL. Návratový kód pre inštalovaný rezidentný program by mal byť FFh, v prípade, že program nie je inštalovaný, návratový kód by mal byť 00h. Na volanie funkcií používateľskej obsluhy prerušenia 2Fh sa neodporúča využívať kódy F8h až FFh. Tieto kódy sú vyhradené pre vnútornú potrebu operačného systému DOS. Nová obsluha pre INT 2Fh môže vyzerať napríklad takto:

```
DGROUP    group DATA
...
DS_REG    dw DGROUP
ID_CISLO  db 8Fh
...
newint2f  proc far
push ds
mov ds, cs:DS_REG
cmp ah,ID_CISLO
je x1
pop ds
jmp cs:[oldint2f]
x1:       sti
cmp al,0
je x2
jmp x3
x2:       mov al,-1
x3:       pop ds
iret
newint2f  endp
```

Ak nezodpovedá ID_CISLO, odovzdáme riadenie na pôvodnú obsluhu. ID_CISLO je identifikačné číslo programu. Pomocou inštrukcie mov al,-1 sa testuje prítomnosť rezidentného programu v pamäti.

INT 23H – výstupná adresa pre Ctrl-Break
Adresa v tomto vektore (0000:008c) je adresa, na ktorú bude odovzdané riadenie, keď DOS zistí, že používateľ stlačil kláves Ctrl-Break. Adresa vektora prerušenia INT 23H je skopírovaná do PSP Ctrl-Break adresného poľa prostredníctvom DOS funkcie 26H (vytvorenie PSP) a funkcie 4Ch (EXEC). Pôvodná hodnota rutiny Ctrl-Break je obnovená po ukončení programu z PSP, t. j. otcovská Ctrl-Break rutina bude obnovená po ukončení synovského procesu.

Pre testovanie kláves Break DOS volá prerušenie INT 23H, len čo zistí, že používateľ stlačil Ctrl-Break. Úroveň testovania klávesu Ctrl-Break môže byť zistená alebo nastavená pomocou funkcie 33H takto:
1. Ak Break je ON, DOS testuje Ctrl-Break behom všetkých funkcií okrem 06H a 07H,
2. V prípade, že Break je OFF, DOS testuje Ctrl-Break iba počas vstupných/výstupných operácií na konzolu, tlačiareň a sériové zariadenia.

Normálna rutina DOS-u pre Break spôsobí okamžité zastavenie programu. K vnútornému presmerovaniu Ctrl-Break na náš program je možné uviesť toto:
1. Použite DOS funkciu 25H na nastavenie vektora prerušenia INT 23H tak, aby ukazoval na váš vlastný Ctrl-Break kód.
2. Po vstupe do vášho manipulačného programu pre Ctrl-Break budú všetky registre nastavené tak, ako boli po vstupe do funkcie DOS-u, ktorá testuje Ctrl-Break.
3. Ak chcete ignorovať Ctrl-Break, stačí zavolať IRET.
4. Pokiaľ chcete niečo vykonávať (napr. zastaviť opakujúcu sa operáciu), potom nezabudnite uschovať všetky registre predtým, než akciu začnete, a po vykonaní akcie ich znovu obnovte. Návrat vykonajte pomocou inštrukcie IRET. Ak neexistujú žiadne obmedzenia na to, čo môže vaša Ctrl-Break rutina robiť - všetky DOS funkcie sú povolené. No pokiaľ samotná rutina vykonáva vstupno-výstupné operácie, používateľ opäť stlačí Ctrl-Break, potom dôjde k pádu DOS-u.
5. V prípade, že chcete ukončiť súčasný proces a vrátiť sa do otcovského procesu, potom nastavte príznak prenosu a vráťte sa pomocou FAR IRET. To pre DOS znamená vykonať normálne vyčistenie a návrat do otcovského procesu.
6. Jednoduchý spôsob, ako zabezpečiť, že proces zaznamená výskyt kombinácie klávesov Ctrl-Break, je volať DOS funkciu 0Bh.

INT 24h (obsluha kritickej chyby)

Vektor prerušenia 24h obsahuje adresu programu, ktorý sa začne vykonávať pri vzniku závažnej chyby vstupu alebo výstupu na niektorom periférnom zariadení. Prerušenie 24H je generované iba pri chybách operácií vykonávaných funkčnými volaniami DOS-u (prerušenie INT 21h). Nie je generované pri chybách, ktoré vznikli v priebehu zápisu alebo čítania pomocou prerušenia INT 25h a INT 26h a ani pomocou prerušení obsluhovaných programom BIOS.

Obsluha prerušenia 24h pracuje tak, že najprv prevezme údaje, ktoré nastaví systém pri vzniku chyby v obsluhe prerušenia 21h. Tieto informácie sú v registroch AX, DI, BP a SI. Potom obsluha prerušenia 24h spracuje tieto údaje a podľa nich určí kód akcie, ktorá bude nasledovať. Kód potom odovzdá systému pri ukončení svojej činnosti v registri AL. Systém ihneď po návrate riadenia z obsluhy 21h vykoná akciu, určenú kódom v registri AL.

Pri vyvolaní prerušenia 24h sú obsluhe prerušenia, či už štandardnej, alebo používateľskej, k dispozícii údaje o povahe vzniknutej chyby a o zariadení, ktoré chybu hlási. Register AH obsahuje v najvyššom bite informáciu o druhu zariadenia, ktoré hlási chybu. Pokiaľ je najvyšší bit nastavený na 0, potom ide určite o chybu diskového zariadenia. V prípade, že je najvyšší bit nastavený na 1, mohla chybu spôsobiť buď chybná tabuľka FAT na diskovom zariadení, alebo ju mohlo spôsobiť niektoré znakovo orientované zariadenie. Ktorá z možností nastala, to možno zistiť zo záhlavia ovládača (device header). Segmentová adresa záhlavia ovládača, ktorý hlási chybu, je pri vyvolaní prerušenia 24h uložená v registroch BP:SI. Záhlavie ovládača má tvar uvedený v tabuľke 4.

Tabuľka 4

Relatívna adresa	Dĺžka v bajtoch	Význam
00 – 04	4	Ukazovateľ na ďalšie záhlavie ovládača (obsahuje hodnotu FFFF:FFFF, ak ide o posledné záhlavie)
05 – 06	2	Vlastnosti zariadenia
07 – 08	2	Ukazovateľ na začiatok riadiaceho programu ovládača zariadení
09 – 0A	2	Ukazovateľ na začiatok vykonávajúceho podprogramu ovládača zariadenia
0B –12	8	Meno zariadenia

V položke "Vlastnosti zariadenia" sú údaje o zariadení. Pokiaľ je najvyšší bit položky 0, ide o diskové zariadenie. Ak je 1, patrí záhlavie znakovo orientovanému zariadeniu. V tom prípade bity 0 až 3 (pozri tabuľku 5) určujú, či nie je na zariadenie práve presmerovaná niektorá zo štandardných operácií.

Ďalšie informácie o chybe obsahuje register DI. Chybový kód je v dolnej polovici registra, horná polovica má nedefinovaný obsah. Význam kódu je v tabuľke 6.

Register AX obsahuje pri začatí práce obsluhy prerušenia 24h jednak špeciálne údaje, ak ide o chybu diskového zariadenia, jednak informácie ovplyvňujúce riešenie závažnej chyby štandardnou obsluhou. Štandardná obsluha prerušenia 24h najprv na obrazovku vypíše, ktoré zariadenie chybu hlási, a otázku: **Abort, Retry, Ignore?** Používateľ musí zadať jednu z odpovedí: 1. Stlačiť kláves A, ak má byť požadovaná operácia zrušená, 2. Stlačiť kláves R, ak má byť požadovaná operácia vykonaná znovu, 3. Stlačiť kláves I, ak má program pokračovať, ako keby k chybe nedošlo. Podľa zadanej odpovede sa štandardná obsluha prerušenia rozhodne pre jeden zo spôsobov, ako bude vzniknutú situáciu riešiť. Na výber riešenia má vplyv aj informácia odovzdávaná pri začatí práce programu obsluhy v registri AX. Obsah registra AX pri vyvolaní prerušenia 24h obsahuje tabuľka 7.

Tabuľka 5

Bit	15	…	3	2	1	0	Význam
	1	…	.	.	.	.	Znakovo orientované zariadenie
		…	**1**	.	.	.	Na zariadenie je presmerované zariadenie CLOCK
		…	.	**1**	.	.	Na zariadenie je presmerované zariadenie NUL
		…	.	.	**1**	.	Na zariadenie je presmerovaný štandardný výstup
		…	.	.	.	**1**	Na zariadenie je presmerovaný štandardný vstup
	0	…	.	.	.	.	Diskové zariadenie

Tabuľka 6

Chybový kód	Význam
00	Pokus o zápis na disketu chránenú proti zápisu
01	Neznáma disketová jednotka
02	Disketová jednotka nie je pripravená
03	Neznámy príkaz
04	Chyba pri cyklickej kontrole dát (chyba parity)
05	Požiadavka nesprávne zadaná
06	Chyba pri nastavovaní hlavičky na stopu
07	Neznámy typ média
08	Sektor nebol nájdený
09	Koniec papiera v tlačiarni
0A	Chyba pri zápise
0B	Chyba pri čítaní
0C	Nedefinovaná chyba

Tabuľka 7

Register AH							Register AL								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Význam
.	.	.	.	.	.	.	**0**	.	.	.	.	.	.	.	.	Chyba pri čítaní
.	.	.	.	.	.	.	**1**	.	.	.	.	.	.	.	.	Chyba pri zápise
.	.	.	.	.	**0**	**0**	.	.	.	.	.	.	.	.	.	Oblasť vyhradená DOS-om
.	.	.	.	.	**0**	**1**	.	.	.	.	.	.	.	.	.	Tabuľka obsadenia sektorov
.	.	.	.	.	**1**	**0**	.	.	.	.	.	.	.	.	.	Adresár
.	.	.	.	.	**1**	**1**	.	.	.	.	.	.	.	.	.	Oblasť dát
.	.	.	.	**1**	.	.	.	.	.	.	.	.	.	.	.	Odpoveď "A" povolená
.	.	.	**1**	.	.	.	.	.	.	.	.	.	.	.	.	Odpoveď "R" povolená
.	.	**1**	.	.	.	.	.	.	.	.	.	.	.	.	.	Odpoveď "I" povolená
.	.	.	.	.	.	.	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**		Číslo jednotky (A=0, B=1, ...)

Tabuľka 8

Kód	Význam
00	Chyba je ignorovaná (bude sa pokračovať, ako keby požadovaná operácia prebehla bez chyby)
01	Požadovaná operácia bude vykonaná znovu
02	Program bude ukončený vykonaním prerušenia 23h
03	Požadovaná operácia je zrušená (funkčné volanie sa ukončí, volajúcemu programu sa odovzdá chybový kód)

Bit 15 spolu so záhlavím ovládača na adrese BP:SI určuje typ zariadenia. Bit 14 sa nevyužíva. Register AL a bity 9 a 10 v registri AH majú uvedený význam iba pre závažné chyby diskových zariadení. Pri spracovaní chýb znakovo orientovaných zariadení nemajú žiadny význam. Na základe bitov 11 až 13 a odpovede zadanej používateľom zvolí štandardný program obsluhy jednu z akcií a jej kód (pozri tabuľku 8) odovzdá pri návrate z prerušenia 24h v registri AL. Kódy majú nasledujúci význam.

Pri kolízii odpovede z klávesnice a nastavení bitov 11 až 13 a registra AX sa volí kód takto: 1. Ak je zadaná odpoveď "I" a bit 13 je 0, volí sa kód 3, 2. Ak je zadaná odpoveď "R" a bit 12 je 0, volí sa kód 3, 3. Ak je zadaná odpoveď "A" a bit 11 je 0, volí sa kód 2. V prípade, že zadaná odpoveď nie je v rozpore s nastavením bitov 11 až 13 v registri AX, spôsobí odpoveď "A" kód 2, odpoveď "R" kód 1 a odpoveď "I" kód nula. Z uvedených zásad existujú dve výnimky. Ak je používateľská odpoveď "I" a závažná chyba vznikla na diskovom zariadení ako dôsledok chybnej tabuľky FAT alebo chybného adresára, volí štandardná obsluha kód 3. Kód zvolený štandardnou obsluhou prerušenia 24h vykoná operačný systém DOS bezprostredne po tom, čo mu obsluha vráti riadenie.

Pri zadávaní odpovede "I" sa odporúča veľká opatrnosť, pretože ignorovanie chyby môže v niektorých prípadoch spôsobiť nepredvídanú situáciu. Hlavne pri chybách diskových zariadení by sa mal používateľ tejto odpovedi vyhnúť.

Pri vstupe do programu obsluhy prerušenia 24h obsahuje zásobník presne definovanú postupnosť hodnôt. Prvé štyri bajty uložené v zásobníku tvoria návratovú adresu do programu obsluhy prerušenia INT 21h (registre CS:IP). Nasledujúce dva bajty, to je hodnota stavového registra FLAGS (stav všetkých príznakov) v okamihu vzniku závažnej chyby. Ďalších 18 bajtov v zásobníku obsahuje hodnoty registrov v okamihu vzniku závažnej chyby. Registre sú v tomto poradí (od vrcholu zásobníka) – AX, BX, CX, DX, SI, DI, BP, DS, ES. Nasledujúce 4 bajty na adresách SP+18h a SP + 1Ah tvoria návratovú adresu z obsluhy prerušenia INT 21h do používateľského programu. Na adrese SP + 1Ch je hodnota stavového registra FLAGS pri vstupe do prerušenia INT 21h.

Používateľská obsluha prerušenia 24h

Používateľská obsluha nesmie používať nijaké funkcie DOS-u okrem služieb 01h až 0Ch a služby 59h. Použitie ostatných služieb spôsobí zničenie údajov v zásobníku, čo má nepredvídateľné následky.

Pri tvorbe používateľskej obsluhy prerušenia 24h by mal byť zachovaný nasledujúci postup. Musí byť neustále zabezpečený prístup k pôvodnej hodnote vektora prerušenia 24h, ktorý ukazuje na štandardnú obsluhu. V prípade vzniku prerušenia 24h by používateľská obsluha mala najprv uložiť do zásobníka stavový register. Potom by mala vyvolať pomocou inštrukcie CALL typu FAR a pôvodného vektora prerušenia štandardnú obsluhu prerušenia. Tá zabezpečí výpis správy o chybe a otázky "Abort, Retry, Ignore?" na obrazovke, prijme z klávesnice odpoveď a do registra Al dá kód zvolenej akcie. Potom vráti riadenie používateľskej obsluhe prerušenia pomocou inštrukcie IRET. Až teraz bude program používateľskej obsluhy prerušenia vykonávať svoju vlastnú činnosť. Ukončenie používateľskej obsluhy prerušenia 24h je možné vykonať dvoma spôsobmi: 1. Odovzdať riadenie operačnému systému DOS pomocou inštrukcie IRET, ale v tomto prípade musí zásobník obsahovať rovnaké hodnoty ako pri vyvolaní prerušenia 24h, 2. Vrátiť riadenie priamo používateľskému programu za funkčné volanie, ktoré spôsobilo chybu. Potom je potrebné najprv zo zásobníka vybrať 24 bajtov (obsahujúcich CS, IP a stavový register FLAGS pri vstupe do prerušenia 24h a hodnoty používateľských registrov pri vzniku závažnej chyby). Zásobník bude teraz obsahovať návratovú adresu CS:IP a stavový register FLAGS platné v okamihu funkčného volania. Návrat sa vykoná pomocou inštrukcie IRET. Treba však poznamenať, že druhý spôsob nie je veľmi vhodný, pretože DOS zostáva v nestabilnom stave až do ďalšieho funkčného volania vyššieho než 0Ch. Teraz si uvedieme dva príklady používateľskej obsluhy prerušenia 24h.

```
old       dd ?
newint24  proc  far
pushf
call far old
cmp al,0
...
newint24  endp
```

Najprv si treba uschovať pôvodný vektor prerušenia 24h (pozri premennú old). Ďalej treba uschovať stavový register pre inštrukciu IRET v štandardnej obsluhe prerušenia 24h. Zavoláme štandardnú obsluhu prerušenia 24h. Po tejto inštrukcii už nasleduje iba vlastný kód používateľskej obsluhy prerušenia 24h. Ďalší spôsob, ako riešiť obsluhu prerušenia 24h, môže vyzerať napríklad takto:

```
fail?     equ 8
ignore    equ 0
fail      equ 3
stav      dw 0
s_error   equ 1
ds_reg    dw DGROUP
newint24            proc  far
                    push  ds
                    mov   ds, cs:ds_reg
                    or    stav, s_error
                    pop   ds
                    test  ah, 80h
                    jnz   n2
                    test  ah, fail?
                    jnz   n2
                    mov   al, ignore
                    iret
n2:       mov   al, fail
                    iret
newint24            endp
```

Chyba je zaznamenaná do premennej STAV. V závislosti od povolených služieb definujeme návratovú službu. Preferovanou službou je FAIL, ak nie je možná, treba zvoliť IGNORE. Premenná STAV obsahuje informácie o stave modulu. Pre premennú STAV si môžete definovať rôzne stavy, podľa ktorých budete rozoznávať, napr. či ide o chybu zariadenia, či bol stlačený kláves, prípadne aktivitu a pod.

INT 28H – plánovač DOS-u

Toto prerušenie používa DOS, keď čaká na stlačenie klávesu. Tento vektor je pozastavovaný rezidentným programom PRINT (spooler DOS-u) na účel nájdenia časového intervalu pre načítanie údajov a odoslanie na tlačiareň. Rezidentné programy využívajú INT 28h ako jeden zo vstupných bodov do svojho kódu. To znamená, že dáva pozor na stlačenie aktivačného klávesu. Ak je zistená požiadavka na aktiváciu, nastaví sa príznak aktivácie jadra rezidentného programu. Potom sa v priebehu spracovania INT 28h zavolá pôvodná obsluha tohto prerušenia, čím sa zabezpečí správna funkcia všetkých rezidentných programov, ktoré už tento vektor používajú. V prípade, že je príznak aktivácie nastavený, aktivujeme naše rezidentné jadro. Tu však treba dodržať jednu dôležitú zásadu: takto aktivovaný program nesmie používať funkcie DOS-u menšie alebo rovné 0Ch.

INT 2D – AMIS (Alternate Multiplex Interrupt Specification)

Vektor prerušenia 2Dh je označovaný ako vektor interne používaný operačným systémom DOS. Doteraz nijaká verzia systému DOS tento vektor nevyužívala. Na adrese, kam ukazuje, po naštartovaní systému sa vždy nachádza iba inštrukcia IRET. Uvedená vlastnosť tohto vektora ho predurčuje na ďalšie využitie. Tento vektor bol vybraný ako náhradný komunikačný vektor pre rezidentné programy. Mal by predovšetkým odstrániť nedostatky komunikačného vektora 2Fh. Oproti 2Fh je podstatne rozšírený a obsahuje aj niektoré mechanizmy, používané hlavne novými rezidentnými programami. Podrobnejšie informácie sa o tomto vektore dočítate v literatúre [1] a [2].

Praktický príklad

Ako posledný rezidentný program si ukážeme jednoduchý screen saver. Po vypršaní časového intervalu program zhasne obrazovku a po stlačení ľubovoľného klávesu ju znova zapne.

```
;screen.asm
TIME      equ  2184 ;2 minutes
Model Tiny
CodeSeg
Org 100h
Start:    jmp Main
Counter   dw TIME
text      db "Screen saver has"
db 32,"been installed.$"
NewInt08  proc
cmp [cs:Counter],0
je I08_done
dec [cs:Counter]
jnz I08_done
push ax dx
mov dx,03C4h
mov al,1
out dx,al
inc dx
in al,dx
or al,020h
out dx,al
pop dx ax
I08_done:
db 0EAh
OldInt08
dw 0,0
NewInt08            endp
NewInt09            proc
cmp [cs:Counter],0
jne I09_done
          push ax dx
mov dx,03C4h
mov al,1
out dx,al
inc dx
in al,dx
and al,0DFh
out dx,al
pop dx ax
I09_done:           mov [cs:Counter],TIME

db 0EAh
OldInt09            dw 0,0
NewInt09            endp
Main      proc
mov ax,word ptr ds:[2ch]
mov es,ax
mov ah,49h
int 21h
          mov ax,3508h
int 21h
mov [OldInt08],bx
mov [OldInt08+2],es
mov al,09h
int 21h
mov [OldInt09],bx
mov [OldInt09+2],es
mov ax,2508h
mov dx,offset NewInt08
int 21h
mov al,09h
mov dx,offset NewInt09
```

```
int 21h
mov ah,09h
mov dx,offset text
int 21h
mov dx,offset Main
int 27h
Main      endp
End Start
```

Opis programu

Program **screen.asm** je napísaný pre pamäťový model **TINY**. Na začiatku programu je definovaná premenná TIME, ktorá obsahuje hodnotu, po ktorej vynulovaní program zhasne obrazovku monitora. Celý program pozostáva z troch procedúr: **NewInt08** (nová obsluha prerušenia INT 8), **NewInt09** (nová obsluha prerušenia INT 9) a **Main** (inštalačná časť). Na začiatku procedúry **NewInt08** sa nachádza test, ktorý zisťuje stav premennej **Counter**. Ak je hodnota premennej rovná nule, vykoná sa skok na návestie **I08_done**. V opačnom prípade sa hodnota tejto premennej dekrementuje o 1. Ďalej nasleduje kód, ktorý vypne obrazovku – uschováme hodnoty registrov AX, DX, do registra DX vložíme číslo portu sekvencera a pošleme na tento port príkaz na zhasnutie obrazovky. Obnovíme registre DX a AX. Na návestí I08_done sa nachádza kód inštrukcie **JMP FAR**, za ním sú rezervované 4 bajty na uschovanie pôvodnej hodnoty prerušenia INT 08. Procedúra **NewInt09** obsahuje takmer ten istý kód, s tým rozdielom, že po stlačení ľubovoľného klávesu zapne obrazovku. Procedúra **Main** (na začiatku) uvoľní pamäť prostredia a zistí adresy prerušení INT 08 a INT 09. Pokračujeme nastavením novej obsluhy pre vektory prerušenia INT 08 a INT 09. Nakoniec na obrazovku vypíšeme krátku informáciu o inštalovaní programu a ponecháme v pamäti nové obsluhy vektorov INT 08 a INT 09.

Preklad programu

Najprv opíšte celý program do súboru napr.: **screen.asm**. Spustiteľný COM súbor potom dostanete takto: **Tasm.exe screen.asm** a **Tlink.exe screen.obj /t**. Ak ste pri opisovaní neurobili chybu, vytvorí sa súbor **screen.com**. V prípade, že by ste chceli tento program doplniť o nejakú zaujímavejšiu činnosť ako zhasnutie monitora, napr. nejaké grafické efekty, potom treba skombinovať program z predošlej časti s dnešným. Aj keď to možno nebude jednoduché, vy na to určite prídete.

Literatúra

[1] Jan Čáp: Rezidentní programy. Grada 1993.
[2] Ralf Brown: Interrupt List - Release 33. Pittsburgh PA, U.S.A. 1993.
[3] Zdeněk Kadlec: Tvorba rezidentních programů. Grada 1995.
[4] David Jurgens: HelpPC 2.10.
[5] ABSHelp 2.05, ABSoft Olomouc.

Sedemnásta časť: Štruktúra diskov, BOOT, FAT

Na ukladanie informácií pri práci s operačným systémom DOS slúžia predovšetkým diskové zariadenia. Média používané v týchto zariadeniach sa nazývajú disky. Disky sa delia na diskety a pevné disky. **Disketa** je výmenné diskové médium s menšou kapacitou (zvyčajne 720 KB, 1,2 MB, 1,44 MB, 2,88 MB, 120 MB a pod.). Pevný disk (hard disk) je zariadenie, ktoré má obyčajne väčšiu kapacitu uložených dát. Dnes sú bežne dostupné tieto kapacity: 1,2 GB, 2,5 GB, 3,1 GB, 4,3 GB, 9 GB, 18 GB a pod.

Fyzická štruktúra diskov

Základné prvky fyzickej štruktúry diskov sú totožné pre diskety i pre pevné disky. Dáta sú vždy ukladané v sústredných kružniciach, ktoré sa nazývajú stopy. Každá stopa je rozdelená na sektory. Všetky stopy na médiu obsahujú rovnaký počet sektorov a všetky sektory umožňujú uloženie rovnakého počtu bajtov. Okrem toho môže diskové zariadenie využívať viacej povrchov média. Pri disketách je využívaný jeden alebo dva povrchy. Počet povrchov pevného disku sa môže líšiť podľa typu. Celkové množstvo dát, ktoré je možné na disk uložiť, je závislé od počtu používaných povrchov, počtu stôp na jednom povrchu, počtu sektorov na jednej stope a od veľkosti sektora. Operačný systém DOS používa sektory veľkosti 512 bajtov.

Fyzická organizácia diskety

Systém DOS umožňuje pracovať s disketami priemeru 5 1/4" alebo 3 1/2". Každá disketa musí byť pred použitím naformátovaná. To znamená, že musí byť použitý program, ktorý vykoná rozdelenie záznamovej plochy na jednotlivé sektory. Bežne sú používané rôzne formáty záznamu dát. Jednotlivé typy sa líšia počtom strán, počtom stôp na jednej strane a počtom sektorov dĺžky 512 bajtov na jednej stope. Podľa toho sa líši kapacita jednotlivých typov diskiet.

Fyzická organizácia pevného disku

Na rozdiel od diskiet fyzické rozdelenie pevného disku vždy urobí výrobca a nie je možné ho meniť. Pevné disky rôznych výrobcov sa môžu líšiť počtom používaných povrchov a počtom stôp na jednom povrchu. Každá stopa na ktoromkoľvek povrchu obsahuje 17 sektorov po 512 bajtoch. Všetky stopy, ktoré sú prístupné čítaciemu zariadeniu v danom okamihu, tvoria jeden valec. Sú to všetky stopy, na každom povrchu po jednej, ktoré majú rovnakú vzdialenosť od okraja záznamového média.

Logická štruktúra diskov

Logické rozdelenie všetkých diskov používaných v operačnom systéme DOS je rovnaké. Číslovanie strán, stôp a sektorov sa riadi rovnakými zásadami pri všetkých typoch diskiet a pevných diskov. Niekoľko sektorov je vždy rezervovaných na špeciálne účely a využíva ich operačný systém. Pri práci s diskovými zariadeniami je potrebné rozlišovať dva typy číslovania sektorov. Prvým typom je **fyzické číslovanie**, používané programom BIOS. Skladá sa vždy z troch bajtov: čísla stopy, čísla strany alebo povrchu a čísla sektora. Stopy sú číslované od 0 do 39, prípadne 79 pri disketách, pri pevných diskoch od 0 do 305, prípadne do 614 pri najbežnejších typoch 1 a 2. Strany diskiet majú čísla 0 a 1, povrchy pevných diskov môžu byť číslované napríklad od 0 do 3 (týka sa starých pevných diskov). Sektory na jednej stope sa označujú od 1 do 8 alebo 9, prípadne 15 pri disketách a od 1 do 17 pri pevných diskoch. Pozornosť treba venovať tomu, že stopy a strany sa číslujú od 0, zatiaľ čo sektory vždy od 1. Toto číslovanie sa uplatňuje pri všetkých operáciách s diskmi, ktoré sú vykonávané na úrovni programu BIOS. Napriek tomu všetky operácie s diskmi, vykonávané operačným systémom DOS, vyhľadávajú sektor pomocou tzv. logického čísla sektora.

Logická štruktúra diskety

Každá disketa používaná operačným systémom DOS je rozdelená na štyri oblasti. Prvé tri oblasti slúžia operačnému systému na uchovanie špeciálnych informácií potrebných na prácu s disketou. Sú to: zavádzací záznam (BOOT), tabuľka obsadenia disku (FAT – File Allocation Table) a adresár. Štvrtá oblasť je vyhradená na ukladanie vlastných dát. Veľkosť zavádzacieho záznamu je vždy jeden sektor. Veľkosť ostatných oblastí je rôzna a závisí od použitého formátu diskety. Zavádzací záznam je vždy v sektore s logickým číslom 0. Od logického sektora 1 sú na diskete zaznamenané dve totožné kópie tabuľky FAT. Bezprostredne za nimi začína adresár súborov uložených na diskete. Zostatok diskety zaplňuje dátová oblasť. Sektory dátovej oblasti sú prideľované jednotlivým súborom po tzv. alokačných blokoch (cluster). Veľkosť alokačného bloku sa líši podľa použitého formátu, jeden alebo dva sektory. Celkový počet súborov, ktoré je možné uložiť na disketu za použitia koreňového adresára, závisí od formátu diskety. Počet položiek koreňového adresára závisí od jeho veľkosti.

Tabuľka 1: Veľkosti jednotlivých oblastí a veľkostí alokačných blokov pre používané formáty (čísla udávajú počet sektorov)

Formát	Zavádzač	Tabuľka FAT	Adresár	Data	Alokačný blok
S-8	1	2 x 1	4	323	1
D-8	1	2 x 1	7	630	2
S-9	1	2 x 2	4	351	1
D-9	1	2 x 2	7	708	2
QD-9	1	2 x 4	7	1424	2
QD-15	1	2 x 7	14	2371	1

Logická štruktúra pevného disku

Pretože pevný disk je väčšinou pevnou súčasťou technického vybavenia počítača, je upravený tak, aby ho mohlo využívať viacej operačných systémov. Pritom každý operačný systém má svoju vlastnú štruktúru dát a odlišné ovládanie pevného disku. Logická štruktúra pevného disku je preto dvojúrovňová.

Prvá úroveň tvorí rozdelenie pevného disku na oblasti, z ktorých každá je vyhradená jednému operačnému systému. Veľkosť každej oblasti je voliteľná, počet oblastí sa môže pohybovať v rozmedzí 1 až 4. Logická štruktúra pevného disku na úrovni oblastí je tvorená hlavným zavádzacím záznamom pevného disku a jednou až štyrmi oblasťami. Formát každej oblasti je závislý od operačného systému, ktorý oblasť využíva. Každá oblasť zaberá niekoľko celých stôp, musí sa začínať i končiť na hranici stopy. Stopy jednej oblasti tvoria na pevnom disku súvislú časť. Výnimkou je iba prvá oblasť, ktorá sa nezačína od začiatku stopy, ale až od 1. sektora nultej stopy povrchu 1. Stopa 0 povrchu 0 je vyhradená na systémové účely. Jej prvý sektor obsahuje hlavný zavádzací záznam pevného disku.

Na vytváranie oblastí na pevných diskoch pod operačným systémom DOS sa používa program FDISK. FDISK vytvára oblasť, ktorá môže zaberať časť disku alebo celý disk a označí ju ako oblasť DOS. Takto vytvorenú oblasť DOS je potrebné ešte upraviť programom FORMAT. Program FORMAT vytvorí v oblasti DOS pevného disku logickú štruktúru. Do prvého sektora oblasti DOS umiestni zavádzací záznam, do ďalších sektorov postupne dve kópie FAT a adresár DIR, zostávajúca oblasť je vyhradená pre záznam dát. Veľkosť tabuľky FAT, adresára a aj veľkosť alokačného bloku závisí od typu pevného disku a veľkosti oblasti DOS. Rozdelenie disku na oblasti aj veľkosť jednotlivých oblastí je možné meniť. Treba však počítať so zničením všetkých dát uložených v oblasti, ktorej sa zmena týka. Dnes, samozrejme, už existujú programy, ktoré nám umožnia rozšírenie, resp. skrátenie oblasti bez straty dát (Partition Magic).

Partition Table

Prvý sektor každého pevného disku vždy obsahuje hlavný zavádzací záznam. Jeho súčasťou je krátky úsek strojového kódu, ktorý zabezpečí správne zavedenie operačného systému z pevného disku. Hlavný zavádzací záznam obsahuje tabuľku pevného disku (Partition Table). Tabuľka oblastí pevného disku je umiestnená na konci hlavného zavádzacieho záznamu. Skladá sa zo štyroch položiek po 16 bajtoch a identifikačného slova (veľkosť 2 bajty).

Tabuľka 2: Partition Table – tabuľka oblastí pevného disku

Adresa	Dĺžka	Význam
1BEh	16	Údaje o oblasti 1
1CEh	16	Údaje o oblasti 2
1EEh	16	Údaje o oblasti 4
1FEh	2	identifikačné slovo

Posledné dva bajty hlavného zavádzacieho záznamu tvorí identifikačné slovo, ktoré označuje platnosť hlavného zavádzacieho záznamu. Ak je hodnota identifikačného slova AA55h, potom ide o platný zavádzací záznam. Každá iná hodnota znamená, že záznam je neplatný. Každá položka tabuľky oblastí je vyhradená jednej oblasti a obsahuje nasledujúce informácie:

Príznak aktívnej oblasti je dôležitý pri zavádzaní operačného systému. Pri zavadzaní sa štartovací program najprv obráti na disketové zariadenie A. Ak je v ňom vložená disketa obsahujúca nepoškodený operačný systém, potom je tento systém zavedený. Pokiaľ nie je disketová jednotka A pripravená a v konfigurácii počítača je pevný disk, je riadenie odovzdané hlavnému zavádzaciemu záznamu. Hlavný zavádzací záznam zistí, či je niektorá z oblastí označená ako aktívna. V prípade, že existuje taká oblasť, je z nej zavedený operačný systém. Ak nie je žiadna oblasť označená ako aktívna, vráti sa riadenie štartovaciemu programu. Príznak aktívnej oblasti je pre aktívnu oblasť nastavený na hodnotu 80h (Bootable – aktívna oblasť). Pokiaľ oblasť nie je aktívna, má hodnotu 00h. Na pevnom disku môže byť označená ako aktívna vždy iba jedna z jeho oblastí.

Číslo povrchu počiatočného sektora oblasti určuje, na ktorom povrchu pevného disku je zaznamenaný prvý sektor oblasti. Za normálneho stavu by malo byť nastavené vždy na 0, pretože všetky oblasti sa majú začínať na hranici stopy. Iba oblasť číslo jeden sa začína vždy na povrchu 1 stopy 0. Stopa 0 na nultom povrchu je rezervovaná pre systémové údaje.

Číslo počiatočného sektora a číslo počiatočnej stopy oblasti určujú číslo sektora na stope a číslo stopy pre prvý sektor oblasti. Číslo sektora je uložené na najnižších šiestich bitoch prvého bajtu. Číslo stopy je uložené na zostávajúcich desiatich bitoch tak, že najvýznamnejšie dva bity sú na najvyšších bitoch prvého bajtu a nižších osem bitov je v druhom bajte.

Tento zvláštny spôsob uloženia neumožňuje spracovať informácie ako jedno slovo (word). Číslo sektora bude za normálnych okolností vždy 1, pretože každá oblasť sa vždy začína na hranici stopy.

Príznak operačného systému určuje, či je položka tabuľky obsadená a aký operačným systémom je oblasť pevného disku používaná. Keď položka tabuľky oblastí nie je využitá, má príznak hodnotu 00h. Operačnému systému DOS sú vyhradené hodnoty príznaku 01h a 04h. Hodnota 01h označuje, že v tabuľke obsadenia sektorov disku budú položky dlhé 12 bitov. Hodnota 04h určuje položky dlhé 16 bitov.

Číslo povrchu posledného sektora určuje, na ktorom povrchu je zaznamenaný posledný sektor oblasti. Normálne je nastavené najvyššie číslo povrchu, pretože každá oblasť by sa mala končiť na hranici stopy.

Číslo posledného sektora a číslo poslednej stopy určuje číslo sektora na stope a číslo stopy pre posledný sektor oblasti. Číslo sektora a číslo stopy je uložené rovnako ako pre prvý sektor oblasti. Číslo sektora bude za normálnych okolností nastavené na 11h, pretože každá oblasť by sa mala končiť na hranici stopy.

Logické číslo prvého sektora oblasti udáva vlastne počet sektorov, ktoré majú logické číslo nižšie než prvý sektor oblasti. Hodnota je uložená v dvoch slovách (štyroch bajtoch), pričom menej významné slovo (2 bajty) je prvé.

Tabuľka 3: Položka – tabuľky oblastí

Relatívna adresa	Dĺžka v bajtoch	Význam
00h	1	Príznak aktívnej oblasti.
01h	1	Číslo povrchu počiatočného sektora oblasti.
02h	1	Číslo počiatočného sektora oblasti.
03h	1	Číslo počiatočnej stopy oblasti.
04h	1	Príznak operačného systému.
05h	1	Číslo povrchu posledného sektora oblasti.
06h	1	Číslo posledného sektora oblasti.
07h	1	Číslo poslednej stopy oblasti.
08h	4	Logické číslo prvého sektora oblasti.
0Ch	4	Počet sektorov oblasti.

Tabuľka 4: Číslo počítačového sektora a číslo počítačovej stopy oblasti

	1. bajt								2. bajt								Význam
bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
	.	.	x	x	x	x	x	x	.	.	.	.	.	.	.	.	Číslo sektora
	x	x	.	.	.	.	.	.	x	x	x	x	x	x	x	x	Číslo stopy

Tabuľka 5: Obsah zavádzacieho záznamu

Adresa	Dĺžka v bajtoch	Význam
000h	3	Skoková inštrukcia na začiatok kódu (JMP skok)
003h	8	Identifikácia operačného systému
00Bh	2	Veľkosť sektora
00Dh	1	Veľkosť alokačného bloku (počet sektorov)
00Eh	2	Počet rezervovaných sektorov na začiatku disku
010h	1	Počet kópií tabuľky FAT
011h	2	Počet položiek v adresári
013h	2	Celkový počet sektorov na disku
015h	1	Identifikácia formátu
016h	2	Veľkosť tabuľku FAT v sektoroch
018h	2	Počet sektorov na stope
01Ah	2	Počet povrchov
01Ch	2	Počet skrytých sektorov
skok		Strojový kód zavádzacieho záznamu
1FEh	2	Identifikačné číslo

Počet sektorov oblasti udáva celkový počet sektorov pridelených danej oblasti. Hodnota je uložená v dvoch slovách (štyroch bajtoch), pričom nižšie slovo je prvé. Aby mohol byť pevný disk využívaný operačným systémom DOS, musí obsahovať platný hlavný zavádzací záznam so správne obsadenou tabuľkou oblastí. Musí byť rozdelený na oblasti, z ktorých jedna je označená ako oblasť DOS. Táto oblasť musí byť správne naformátovaná. To znamená, že jej prvé sektory musia obsahovať zavádzací záznam, dve kópie tabuľky FAT a adresár. K takejto oblasti potom DOS pristupuje ako ku každému inému disku vrátane logického číslovania sektorov od 0.

Zavádzací záznam – boot

Zavádzací záznam musí byť obsiahnutý v prvom sektore každého disku, používaného operačným systémom DOS. Pri disketách musí byť v prvom sektore diskety, pri pevných diskoch v prvom sektore oblasti DOS. Zavádzací záznam sa skladá z úseku strojového kódu, z oblasti obsahujúcej údaje o disku a z identifikačného čísla. Na adrese 00h je umiestnená skoková inštrukcia na začiatok úseku kódu zavádzacieho záznamu. Tento krátky program zistí, či je disk systémový. To znamená, či obsahuje systémové súbory, potrebné na zavedenie systému z tohto disku. Podľa toho potom buď začne zvádzanie operačného systému, alebo vypíše na obrazovku správu, že z tohto disku nie je možné operačný systém zaviesť. Na adresách 03h až 1Dh sú uložené parametre disku. Posledné dva bajty prvého sektora sú obsadené identifikačným číslom. Obsah identifikačného čísla AA55h znamená, že zavádzací záznam je platný. V prípade, že má zavádzací záznam inú hodnotu, je zavádzací záznam neplatný.

Tabuľka obsadenia sektorov disku (FAT)

Podobne ako disketa aj oblasť DOS na pevnom disku musí obsahovať od druhého sektora dve totožné kópie tabuľky FAT (File Allocation Table). Tabuľka FAT slúži na označenie miesta na disku, kde sú uložené dáta jednotlivých súborov. Miesto na disku nie je súborom prideľované po jednotlivých sektoroch, ale po alokačných blokoch. Alokačný blok môže byť rôzne veľký podľa typu formátu disku. V tabuľke FAT zodpovedá každému alokačnému bloku jedna položka. Sekvenčné poradie položiek v tabuľke zodpovedá poradiu alokačných blokov v dátovej oblasti na disku. Veľkosť položky tabuľky obsadenia disku je rôzna podľa veľkosti dátovej oblasti. Pre disky s menej než 4080 alokačnými blokmi sa používa dĺžka položky 12 bitov, pre väčšie disky má položka dĺžku 16 bitov. Pre všetky formáty diskiet sa používa tabuľka s 12-bitovými položkami. Jedna 12-bitová položka obsahuje tri hexadecimálne číslice. Položky sú v tabuľke zoradené tak, že vždy v troch bajtoch sú dve položky. Každá trojica bajtov je rozdelená medzi položky tak, že v prvom bajte je nižších osem bitov prvej položky a vyššie štyri bity prvej položky sú v menej významných bitoch druhého bajtu. Vyššie štyri bity druhého bajtu sú najnižšie bity druhej položky. Vyšších osem bitov druhej položky je v treťom bajte.

Dôvodom tohto rozdelenia je snaha o čo najväčšiu úsporu miesta a zároveň jednoduchá manipulácia s položkami pomocou niekoľkých inštrukcií v assembleri.

Podstatne jednoduchšie je rozdelenie tabuľky FAT so 16-bitovými položkami. Každá šestnásťbitová položka je uložená v dvoch bajtoch, pričom vždy nižší bajt položky je uložený ako prvý. Obsahom jednej položky sú štyri hexadecimálne číslice. Zvláštny význam má prvý bajt tabuľky FAT, ktorý obsahuje tzv. identifikačný bajt. Identifikačný bajt určuje, o aký formát disku ide. Identifikačný bajt je súčasťou položky zodpovedajúcej alokačnému bloku 0. Položky tabuľky FAT zodpovedajúce alokačným blokom 0 a 1 sa nevyužívajú. Prvý alokačný blok pridelený dátam niektorého súboru má číslo 2. Položka zodpovedajúca alokačnému bloku 0 obsahuje identifikačný bajt doplnený v horných štyroch bitoch hexadecimálnymi číslicami F tak, aby význam položky bol „posledný obsadený alokačný blok súboru“. Položka zodpovedajúca alokačnému bloku 1 je vždy obsadená hodnotou FFFh, prípadne FFFFh pre 16-bitovú položku.

Ak je hodnota položky tabuľky FAT 000h alebo 0000h pri 16-bitovej položke, potom zodpovedajúci alokačný blok nie je použitý na uloženie dát nijakého z existujúcich súborov a je voľný. Alokačný blok zodpovedajúci položke s hodnotou 000h môže obsahovať dáta predtým zmazaného súboru. Pri mazaní súboru sa všetky položky zod-

Tabuľka 6: Možné hodnoty položiek tabuľky FAT

Položka		Význam
12-bitová	**16-bitová**	
000h	0000h	Voľný alokačný blok
002h – FEFh	0002h – FFEFh	Ukazovateľ na nasledujúci alokačný blok
FF0h – FF6h	FFF0h – FFF6h	Rezervovaný alokačný blok
FF7h	FFF7h	Chybný alokačný blok
FF8h – FFFh	FFF8h – FFFFh	Posledný alokačný blok pridelený súboru

povedajúce alokačným blokom rušeného súboru naplnia hodnotou 000h. Preto je možné dáta zmazaného súboru obnoviť. Ale iba v prípade, že alokačné bloky, v ktorých boli uložené dáta zmazaného súboru, neboli pridelené inému súboru. Hodnota položky FF0h až FF7h, prípadne FFF0h až FFF7h pre 16-bitovú položku znamená, že zodpovedajúci alokačný blok je rezervovaný. Takýto alokačný blok nie je možné prideliť nijakému súboru. Špeciálne hodonota FF7h, resp. FFF7h označuje chybný alokačný blok. Hodnota položky FF8h až FFFh, prípadne FFF8h až FFFFh pre 16-bitovú položku znamená, že zodpovedajúci alokačný blok je posledným alokačným blokom, ktorý ešte patrí danému súboru. Takýto alokačný blok nemusí byť zaplnený dátami celý. Každá iná hodnota položky je číslo, ktoré znamená poradie ďalšej položky v tabuľke FAT. Nová položka, na ktorú ukazuje predchádzajúca položka, zodpovedá alokačnému bloku, v ktorom sú uložené dáta v súbore nasledujúcom za dátami uloženými v alokačnom bloku zodpovedajúcom predchádzajúcej položke. Hodnota položky FF0h až FF6h, prípadne FFF0h až FFF6h znamená, že zodpovedajúci alokačný blok je rezervovaný.

Každému súboru uloženému na disku zodpovedá reťaz alokačných blokov, ktorá je zaznamenaná v tabuľke FAT ako reťaz nadväzujúcich položiek. Ukazovateľ na položku zodpovedajúcu prvému alokačnému bloku priradenému súboru je zaznamenaný v adresári. Táto a každá ďalšia položka reťazca buď ukazuje na nasledujúcu položku, alebo obsahuje značku konca súboru (hodnotu FFF8h – FFFFh). Takto vzniká reťazec položiek zodpovedajúcich reťazcu alokačných blokov prideľovaných postupne danému súboru.

Napríklad v adresári je ako prvá označená položka 007h. To znamená, že začiatok súboru bude uložený v siedmom alokačnom bloku dátovej oblasti disku. Zároveň siedma položka obsahuje ukazovateľ na nasledujúcu položku, napr. 008h. To znamená, že po zaplnení siedmeho alokačného bloku bol súboru pridelený ôsmy alokačný blok. Zápis dát pokračuje doň. Zároveň ôsma položka tabuľky FAT obsahuje hodnotu 00Ah. Ďalším alokačným blokom v reťazci blokov prideľovaných súboru je desiaty alokačný blok. Desiata položka obsahuje hodnotu FFFh. To znamená, že desiaty alokačný blok je posledným alokačným blokom, ktorý je pridelený danému súboru. Z tohto príkladu vyplýva, že alokačné bloky pridelené súboru môžu, ale nemusia tvoriť na disku súvislú oblasť. Ak zapisujeme jeden súbor na novo formátovaný disk, na ktorom doteraz nijaké súbory nie sú, potom bude zaberať súvislú oblasť a budú mu priradené prvé alokačné bloky dátovej oblasti. Podobne to bude i s ďalšími zapisovanými súbormi až do okamihu, keď začneme niektoré súbory na disku mazať. Tým sa uvoľnia alokačné bloky, za ktorými môžu nasledovať obsadené alokačné bloky. Ak potom zapisujeme nejaký súbor na disk, je pravdepodobné, že mu nebudú pridelené alokačné bloky v jednej súvislej oblasti. Tento jav sa nazýva fragmentácia. Čím viacej nesúvislých oblastí na disku súbor zaberá, tým je aj čas potrebný na jeho zavedenie (napr. do pamäte počítača) dlhší.

Pri prideľovaní alokočných blokov a pri vypĺňovaní údajov do tabuľky FAT môžu vzniknúť rôzne chyby. Napríklad reťaz položiek nie je ukončená značkou (FF8h – FFFh) alebo položky reťazca sú prepojené do kruhu. Takisto je častou chybou, že alokačný blok je v tabuľke FAT označený ako pridelený, a pritom nie je súčasťou žiadneho reťazca. Tieto chyby väčšinou vznikajú vinou chyby technického vybavenia počítača, alebo tak, že používateľský program nesprávne manipuluje s dátami pomocou služieb poskytovaných programom BIOS. Typickou situáciou, pri ktorej chyby v tabuľke FAT vznikajú, je násilné ukončenie zápisu súboru na disketu, napr. pri predčasnom vytiahnutí diskety z disketovej jednotky. Operačný systém umožňuje opravu väčšiny takýchto chýb, či už pomocou svojich nástrojov, alebo iných (napr. Norton Utilities). Dosť bolo teórie, poďme do praxe.

Praktický príklad

Dnes si ukážeme program, ktorý dokáže načítať ľubovoľný textový ASCII súbor a zobrazí ho v špeciálnom režime 40 x 128 znakov (grafický režim 640 x 350 bodov – mono).

```
;view.asm
; 128 line wide file viewer
; Autor Tylisha C. Andersen

o   equ <offset>
b   equ <byte ptr>
w   equ <word ptr>

.model tiny
.186

.data ;font 8 x 5, komprimovaný
fontcomp      equ   this byte
          dw 0,0,35584,47779,53757,49147,50525
          dw 20703,-3521,56,15904,14562
          dw 29440,-2113,16,7537,4343,0,8,16
          dw -2049,-4097,255,-6380,40,-5121
          dw 55064,6655,23256,2128,40331,4119
          dw 19204,-2864,4128,53835,9461
          dw 21520,17782,4368,15812,28876,12304
          dw -7236,568,2417,31991,20996
          dw 41108,2600,-9301,37970,18693,18836
          dw 3088,0,14343,28942,-2295,-7041
          dw 2417,4338,8452,18184,1148,15904
          dw 2082,8192,33342,32,8192,16399
          dw 0,41022,40,7200,47183,-2017,-3555
          dw 1080,0,0,16896,32784,2080,421
          dw 64,20992,-2667,2684,6513,2062,8,1060
          dw 4624,11426,20618,8458,32768
          dw 0,8513,16388,17412,17416,4112
          dw 15504,12489,8192,16958,16,0,256
          dw 4,60,0,0,0,4104,17416,4128,11667
          dw 26665,49676,33808,7200,2195
          dw 8233,37662,10520,3080,48977,2081
          dw 34562,51589,3080,14723,18441
          dw 5900,9352,2064,6547,18473,37644
          dw 8477,3080,64,4,8192,512,2048
          dw 40992,8322,2,-8191,120,1092,4164
          dw 37648,8200,1040,11667,22568
          dw 37644,10557,4680,14743,18473
          dw 37660,2337,3136,9623,18473,34588
          dw 2233,7744,47495,16392,37648,2349
          dw 3144,48532,18473,18194,33808
          dw 7200,33809,18473,37900,18865,4688
          dw 8580,16392,-3042,10661,4680
          dw 42452,18537,37650,10533,3144
          dw 14743,16424,37648,9765,1624,14743
          dw 18473,37650,2329,3080,51239,4162
          dw 37892,10661,3144,42388,18474
          dw 37896,12197,4680,39316,18473
          dw 37906,8857,1040,34839,8232,34590
          dw 2081,7232,4224,4226,9986,16904
          dw 7184,147,0,0,0,7680,66,64,0,51460
          dw 3640,9604,18633,28,51492,3136
          dw 42000,18665,14,51236,3192,14417
          dw 8324,8,-6364,33353,9604,18633
          dw 530,33809,7200,33792,2401,33938
          dw 10793,4704,4166,8324,28,10557
          dw 4680,9472,18633,18,51492,3144,9472
          dw 19150,16,-6364,16968,9472,16584
          dw 16,-7904,7216,4418,8388,4,10533
          dw 3656,9472,18474,8,12069,4680
          dw 9472,12329,18,10021,33353,1280,4324
          dw 16670,33824,1056,66,8324,17416
          dw 33800,4128,16563,0,0,18708,7752
          dw 41347,16647,1156,10661,3656,41985
          dw 30920,1804,51588,3640,33796
          dw 14537,1550,51460,3640,1027,14537
          dw 14,-6368,33857,41991,30920,1036
          dw 51364,3192,9222,30920,524,49800
          dw 3600,34823,4290,1550,49672,3600
          dw 41988,30921,25362,51492,4728,41217
          dw 28904,30,-1527,8060,48547
          dw 20554,1814,51620,3144,41988,18633
          dw 1548,51492,3144,42247,18473
          dw 1550,10533,3656,42244,19239,1154
          dw -5723,7752,42244,18473,8478
          dw -6368,33856,14417,41092,21534,18268
          dw 1040,9623,23753,10257,17052
          dw 5137,33793,14537,270,49800,3600
          dw 41985,18633,268,10661,3656,47266
          dw 18441,41490,2980,4712,40017,96
          dw 20736,16412,0,4098,16521,12,2108
          dw 64,15360,2049,37888,16661,1860
          dw 5012,52291,257,16904,1040,10280
          dw 43170,40960,43530,40,658,4233
          dw 27684,9371,14041,30395,47067,8557
          dw 16904,33808,2081,28738,8580,49673
          dw 33904,37970,27045,74,1280,19065
          dw 2304,28866,21124,42373,19049
          dw 37970,10661,74,-6907,19049,34130
          dw 30880,20992,41108,120,2337,28864
          dw 0,512,33904,2081,7232,8448,16392
          dw 124,0,31746,8580,16904,33820,0
          dw 31744,8448,16904,33916,2081,7282
          dw 21124,42388,18989,36946,15536,0
          dw -2800,18989,33106,31920,0,-2815
          dw 19053,36946,11701,74,-4095,124
          dw 33106,28085,8522,-4095,124,37970
          dw 31904,0,-3583,33916,0,32005,21066
          dw 41108,60,2081,7280,0,29192
          dw 33820,0,15621,21066,42388,19069
          dw 2337,31986,8580,16392,112,0,7170
          dw -124,-1,-1,0,-241,-6145,52793
          dw 40051,52793,40051,-25,-3841,0,9544
          dw 55396,-8192,10553,5194,8688,16424
          dw 16,-2795,2600,34967,8329,30
          dw -2524,72,4608,9879,8,41512,2064
          dw 9456,18630,24606,10557,3144,9072
          dw 10261,12315,35080,3176,10752,21735
          dw 4096,-6358,2132,14704,16392
```

```
          dw 24590,10533,4680,15600,15,8448
          dw -4087,7952,6244,16416,24606,153
          dw 7688,34856,4178,8580,18952,2128
          dw 15904,2,20480,18945,40,2129,160
          dw 0,8,16,0,4096,8448,18152,3152
          dw 5216,160,4864,16412,0,7168,14567
          dw 0,0,0

nomem$              db    'not enough memory to load'
                    db   13, 10, '$'
syntax$             db    'syntax:  VIEW <file>'
                    db   13, 10, '$'
notfound$           db    'file not found'
                    db  13, 10, '$'
null_str            db    0

.data?

font                db    2048 dup(?) ;buffer pre font
max_lines           dw    ? ;max. číslo riadkov na zavedenie
num_lines           dw    ? ;počet riadkov v súbore
first_line          dw    ? pointer na prvý riadok
buf_size            equ   8192
                    db    2 dup(?)
filename            db   128 dup(?) ;meno súboru
buffer              db   buf_size dup(?) ;diskový buffer
handle              dw    ? ;file handle
buffer_pos          dw    ? ;pozícia v bufferi
buffer_end          dw    ? ;koniec dát v bufferi

.code
org 100h

start:              jmp   main ;skok na začiatok

m_error:            mov   ah, 9
                    int   21h
m_exit:             mov   ax, 4C00h ;návrat do DOS-u
                    int   21h
m_nomem:            mov   dx, o nomem$
                    jmp   m_error ;chybové správy
m_syntax:           mov   dx, o syntax$
                    jmp   m_error
m_notfound:         mov   dx, o notfound$
                    jmp   m_error

main:               mov   sp, o stack_top ;new stack
                    mov   ah, 4Ah
                    mov   bx, o stack_top+15
                    shr   bx, 4
                    int   21h
                    jc    m_nomem ;out of memory?
                    mov   ah, 48h
                    mov   bx, -1
                    int   21h
                    mov   ah, 48h
                    int   21h
                    mov   first_line, ax
                    shr   bx, 3
                    sub   bx, 40
                    jle   m_nomem  ;out of memory?
                    mov   max_lines, bx
                    lea   cx, [bx+39]
m_cloop:            mov   es, ax
                    mov   b es:[0], 0
                    add   ax, 8
                    loop  m_cloop
                    push  ds  ;es = ds
                    pop   es
                    ;dĺžka príkazového riadka
                    mov   cl, ds:[80h]
                    xor   ch, ch
                    inc   cx
                    mov   di, 81h ;hľadaj prvý znak
                    mov   al, ' '
                    repe  scasb
                    lea   si, [di-1]
                    cmp   b [si], 0Dh
                    je    m_syntax
                    repne scasb
                    mov   b [di-1], 0
                    mov   di, o filename
                    ;konvertuj na plnu cestu
                    mov   ah, 60h
                    int   21h
                    mov   ax, 2020h
                    mov   w filename-2, ax
                    mov   dx, o filename
                    mov   ax, 3D00h ;otvor súbor
                    int   21h
                    jc    m_notfound ;skok pri chybe
                    mov   handle, ax
                    mov   bp, first_line
                    mov   ax, buf_size
                    mov   buffer_pos, ax
                    mov   buffer_end, ax
                    mov   num_lines, 0
m_lineloop:         mov   es, bp
                    xor   di, di
                    mov   cx, 127
m_charloop:         call  getchar
                    jc    m_chardone
                    cmp   al, 13
                    je    m_chardone
                    cmp   al, 10
                    je    m_charloop
                    jcxz  m_charloop
                    stosb
                    dec   cx
                    jmp   m_charloop
m_chardone:         mov   al, 0
                    stosb
                    inc   num_lines
                    jc    m_cont
                    add   bp, 8
                    mov   ax, num_lines
                    cmp   ax, max_lines
                    jb    m_lineloop
m_cont:             mov   bx, handle
                    mov   ah, 3Eh
                    int   21h
                    push  ds
                    pop   es
                    mov   si, o fontcomp
                    mov   di, o font
                    mov   bp, 256
                    jmp   m_decloop
m_decstore:         shr   al, 3
                    and   ax, 1F1Fh
                    stosw
                    ret
m_decloop:          lodsw
                    xchg  bx,ax
                    lodsw
                    xchg  cx,ax
                    lodsb
                    xchg  dx,ax
                    mov   ah, bl
                    mov   al, bh
                    shr   ax, 3
                    call  m_decstore
                    mov   ah, bh
                    mov   al, cl
                    shr   ax, 1
                    call  m_decstore
                    mov   ah, cl
                    mov   al, ch
                    shl   ax, 1
                    call  m_decstore
                    mov   ah, ch
                    mov   al, dl
                    shl   ax, 3
                    call  m_decstore
                    dec   bp
                    jnz   m_decloop
                    mov   ax, 10h
                    int   10h
                    mov   si, o null_str
                    mov   al, -1
                    mov   dx, 42
                    call  putline
                    mov   si, o filename-2
                    mov   dx, -1
                    call  putline
                    mov   bp, -1
                    jmp   k_home
m_done:             mov   ax, 3
                    int   10h
                    jmp   m_exit
k_keyloop:          xor   ah, ah
                    int   16h
                    cmp   ah, 01h
                    je    m_done
                    cmp   ah, 48h
                    je    k_up
                    cmp   ah, 50h
                    je    k_down
                    cmp   ah, 49h
                    je    k_pgup
                    cmp   ah, 51h
                    je    k_pgdn
                    cmp   ah, 47h
                    je    k_home
                    cmp   ah, 4Fh
                    je    k_end
                    jmp   k_keyloop
k_up:               test  bp, bp
```

```
                jz    k_keyloop
                call  scrolldn
                dec   bp
                imul  ax, bp, 8
                add   ax, first_line
                mov   es, ax
                xor   si, si
                mov   dx, 2
                xor   al, al
                call  putline
                jmp   k_keyloop

k_down:         mov   ax, num_lines
                dec   ax
                dec   ax
                cmp   bp, ax
                jge   k_keyloop
                call  scrollup
                inc   bp
                imul  ax, bp, 8
                add   ax, first_line
                add   ax, 39*8
                mov   es, ax
                xor   si, si
                mov   dx, 41
                xor   al, al
                call  putline
                jmp   k_keyloop

k_pgup:         test  bp, bp
                jz    k_keyjmp
                sub   bp, 40
                jnc   k_redraw
                xor   bp, bp
                jmp   k_redraw

k_pgdn:         mov   ax, num_lines
                sub   ax, 41
                cmp   bp, ax
                jge   k_keyjmp
                add   bp, 40
                jmp   k_redraw

k_home:         test  bp, bp
                jz    k_keyjmp
                xor   bp, bp
                jmp   k_redraw

k_end:          mov   ax, num_lines
                sub   ax, 40
                cmp   bp, ax
                jge   k_keyjmp
                mov   bp, ax
                jmp   k_redraw

k_redraw:       imul  di, bp, 8
                add   di, first_line
                mov   dx, 2
                xor   si, si
                xor   al, al

k_rloop:        mov   es, di
                call  putline
                add   di, 8
                inc   dx
                cmp   dx, 42
                jb    k_rloop
k_keyjmp:       jmp   k_keyloop

getchar:        pusha
                mov   si, buffer_pos
                cmp   si, buffer_end
                jb    g_load
                cmp   si, buf_size
                jb    g_done
                mov   ah, 3Fh
                mov   bx, handle
                mov   cx, buf_size
                mov   dx, o buffer
                int   21h
                jc    g_done
                mov   buffer_end, ax
                xor   si, si
                mov   buffer_pos, si
g_load:         mov   al, buffer[si]
                inc   buffer_pos
                clc
g_done:         mov   bp, sp
                mov   [bp+14], al
                popa
                ret

putline:        pusha
                push  ds
                push  es
                pop   ds
                push  0A000h
                pop   es
                imul  dx, 640
                mov   di, dx
                mov   ah, al
                mov   cx, 640/2
                test  dx, dx
                jge   p_clear
                mov   dx, 240
                xor   di, di
                mov   cx, 1120/2
p_clear:        rep   stosw
                mov   di, dx
                mov   bp, ax
                xor   bx, bx
p_yloop:        push  si
                push  di
                xor   dx, dx
                mov   cl, 11
p_xloop:        lodsb
                test  al, al
                jz    p_xdone
                xor   ah, ah
                shl   ax, 3
                add   bx, ax
                mov   ch, cs:font[bx]
                sub   bx, ax
                mov   al, ch
                xor   ah, ah
                shl   ax, cl
                or    dx, ax
                sub   cl, 5
                jnc   p_xloop
                mov   al, dh
                shl   dx, 8
                add   cl, 8
                xor   ax, bp
                stosb
                jmp   p_xloop
p_xdone:        mov   al, dh
                mov   ah, dl
                xor   ax, bp
                stosw
                pop   di
                pop   si
                add   di, 80
                inc   bx
                cmp   bx, 8
                jb    p_yloop
                pop   es
                pop   ds
                popa
                ret

scrolldn:       pusha
                push  ds
                push  es
                mov   ax, 0A000h
                mov   ds, ax
                mov   es, ax
                std
                mov   si, 2*640+24960-2
                mov   di, 3*640+24960-2
                mov   cx, 24960/2
                rep   movsw
                cld
                pop   es
                pop   ds
                popa
                ret

scrollup:       pusha
                push  ds
                push  es
                mov   ax, 0A000h
                mov   ds, ax
                mov   es, ax
                mov   si, 3*640
                mov   di, 2*640
                mov   cx, 24960/2
                rep   movsw
                pop   es
                pop   ds
                popa
                ret

.data?
stack_buf       db    1024 dup(?)
stack_top:
end start
```

Opis programu

Program view.asm je napísaný pre pamäťový model TINY. V dátovej oblasti je definovaný komprimovaný font (matica 8 x 5 bodov), v ktorom bude daný súbor zobrazený. V programe **view.asm** sa využívajú nasledujúce služby operačného systému DOS: **INT 21h – služba 3Dh (otvorenie súboru)** – služba podľa mena v ASCIIZ reťazci špecifikovaného adresou v DS:DX otvorí existujúci súbor. V registri AL sa odovzdávajú nasledujúce atribúty otvorenia súboru:

Pokiaľ došlo k chybe, je nastavený príznak CF a registri AX je kód chyby (2 – súbor nebol nájdený, 3 – cesta nebola nájdená, 4 – nie je voľné číslo súboru, 5 – nepovolený prístup, 0Ch – chybné atribúty otvorenia súboru). V prípade, že sa služba skončila bez chyby, je v registri AX číslo súboru.

Tabuľka 7: Atribúty otvorenia súboru

Bity	Stavy	Význam
7	0	synovský proces súbor dedí
	1	súbor je privátny
6 – 4	000	kompatibilita s FCB
	001	inému procesu je zakázané čítanie aj zápis
	010	inému procesu je zakázaný zápis
	011	inému procesu je zakázané čítanie
	100	nie sú nijaké obmedzenia
3	0	
2 – 0	000	súbor otvorený na čítanie
	001	súbor otvorený na zápis
	010	súbor otvorený na čítanie aj zápis

INT 21h – služba 3Eh – sa používa na zatvorenie súboru, ktorého číslo sa nachádza v registri BX. Ak sa vyskytla nejaká chyba, je príznak CF nastavený na 1 a v registri AX je kód chyby (6 – chybné číslo súboru alebo súbor nie je otvorený).

INT 21h – služba 3Fh – na základe čísla súboru odovzdaného v registri BX z neho prečíta počet bajtov (zadaných v registri CX) a uloží ich do pamäte od adresy v registroch DS:DX. Pokiaľ operácia skončila bez chyby, je príznak CF=0 a v registri AX je počet skutočne prečítaných bajtov. V prípade, že sa vyskytla chyba, je príznak CF=1 a v registri AX je kód chyby (5 – nepovolený prístup, 6 – chybné číslo súboru alebo súbor nie je otvorený).

INT 21h – služba 48h (pridelenie bloku pamäte) – služba pridelí programu blok pamäte, ktorej veľkosť je zadaná v paragrafoch v registri BX. Služba vráti v AX segmentovú časť adresy začiatku bloku (offset = 0). Pokiaľ došlo k chybe, je nastavený príznak CF na 1, register AX obsahuje kód chyby (7 – riadiaci blok je zničený, 8 – nedostatočná kapacita pamäte) a v registri BX je kapacita najväčšieho dostupného bloku v paragrafoch.

INT 21h – služba 4Ah (zmena veľkosti bloku pamäte) – služba umožňuje zmenšiť, prípadne zväčšiť už pridelený blok pamäte. V registri ES musí byť segmentová časť adresy bloku pamäte a v registri BX požadovaná nová dĺžka (v paragrafoch). Pokiaľ došlo k chybe, nastaví sa príznak CF na jedna a register AX obsahuje chybový kód (7 – riadiaci blok je zničený, 8 – nedostatočná kapacita pamäte, 9 – chybná hodnota segmentu v registri ES). V registri BX je kapacita najväčšieho dostupného bloku v paragrafoch.

INT 21h – služba 60h. Služba oznámi skutočné meno zariadenia alebo adresára. Dvojica registrov DS:SI musí ukazovať na pôvodné meno a registre ES:DI na vektor v pamäti, do ktorého sa uloží skutočné meno.

Program štartuje skokom na návestie **MAIN**. Na začiatku si nastavíme nový zásobník a alokujeme pamäť. Z príkazového riadka sa zistí meno zadaného súboru. Otvoríme súbor zadaného mena, prečítame z neho všetok text do buffera a súbor zatvoríme. Pokračujeme dekomprimovaním fontu, nastavením videomódu 0Fh (640 x 350) a zobrazením textu na obrazovku. Od návestia **K_KEYLOOP** sa testuje stlačenie klávesov. V prípade, že je stlačený kláves ESC, program skočí na návestie **M_DONE**, nastaví textový režim 3 (80 x 25 znakov) a vráti sa do DOS-u. Procedúry **K_UP, K_DOWN, K_PGUP, K_PGDN, K_HOME** a **K_END** sú volané vždy po stlačení príslušného klávesu. Používajú sa na pohyb po obsahu daného súboru. Časť zdrojového kódu od návestia **K_REDRAW** sa používa na vykreslenie riadkov na obrazovku. Procedúra **GETCHAR** prečíta jeden znak zo súboru do buffera. Vstup register DS = CS, výstup register AL obsahuje znak. Procedúra PUTLINE zobrazí riadok textu na obrazovku. Vstupné parametre procedúry sú: registre ES:SI musia obsahovať adresu reťazca, ktorý chceme zobraziť, register DX obsahuje hodnotu Y-pozícia, register AL obsahuje hodnotu 0 alebo –1. Táto hodnota určuje, ako bude text zobrazený na obrazovke (0 – znamená biele znaky na čiernom pozadí). Procedúry **SCROLLDN** a **SCROLLUP** umožňujú posun o riadok dole, resp. o riadok hore.

Preklad programu

Najprv opíšte celý program do súboru napr.: **view.asm**. Spustiteľný COM súbor potom dostanete takto: **Tasm.exe view.asm** a **Tlink.exe view.obj /t**. Ak ste pri opisovaní neurobili chybu, vytvorí sa súbor view.com. Program ako parameter očakáva meno súboru (napr. view.com readme.txt), ktorý chcete zobraziť v tomto špeciálnom móde. Tento príklad možno považovať za veľmi dobrý námet, čo sa týka vzdelávania v jazyku assembler. Program nateraz iba zobrazí vami zadaný súbor, vy sa ho však môžete pokúsiť doplniť napr. o editovanie textu, mazanie, presun alebo kopírovanie bloku, použiť myš atď. Námetov sa ponúka veľmi veľa, tak do práce.

Literatúra

[1] Ralf Brown: Interrupt List - Release 33. Pittsburgh PA, U.S.A. 1993.
[2] Zdeněk Kadlec: Tvorba rezidentních programů. Grada 1995.
[3] P. Heřman: Príručka systémového programátora. Tesla Eltos 1990.
[4] David Jurgens: HelpPC 2.10.
[5] ABSHelp 2.05, ABSoft Olomouc.

Osemnásta časť: Podadresár, prerušenie INT 13h

Po naformátovaní disku operačným systémom DOS je vždy vytvorený adresár disku. Jeho veľkosť a umiestnenie záleží na type disku. V adresári sú uchovávané všetky dôležité údaje o súboroch uložených na disku. Každému súboru je vyhradená jedna položka, ktorá má pevnú štruktúru. Nezávisle od formátu disku jedna položka vždy zaberá 32 bajtov. Pre každý súbor je v položke zaznamenané meno, prípona súboru, špeciálne vlastnosti súboru (atribúty), čas a dátum poslednej modifikácie súboru, ukazovateľ na položku v tabuľke FAT, ktorý zodpovedá prvému pridelenému alokačnému bloku, a posledným parametrom je dĺžka súboru v bajtoch. V adresári nenájdete rozmiestnenie a počet alokačných blokov, ktoré má daný súbor pridelené. Tieto údaje obsahuje tabuľka FAT. Okrem položiek obsahujúcich údaje o súboroch môže adresár obsahovať dva odlišné typy položiek. Prvým je položka návestia (volume label) daného disku, druhým je položka obsahujúca údaje o podadresári. Obidva typy majú rovnakú štruktúru ako položky súborov, sú však označené špeciálnym príznakom v poli uchovávajúcom vlastnosti súboru.

Podadresár

Podadresáre je možné vytvárať na každom disku s operačným systémom DOS. Podadresár rovnako ako koreňový adresár je zostavený z položiek uchovávajúcich údaje o súboroch podadresára, prípadne o podadresároch ďalšej úrovne. Takisto zloženie položky podadresára aj význam jednotlivých polí v položke je rovnaký. Na rozdiel od koreňového adresára nie je veľkosť podadresára (a teda ani počet súborov v ňom), obmedzená. Nie je obmedzený ani počet úrovní podadresárov pri vytváraní stromovej štruktúry podadresárov. Jediným obmedzením je kapacita daného disku a maximálna dĺžka cesty – 63 znakov. Každému podadresáru sú rovnako ako súboru prideľované alokačné bloky. Odlišné je uvoľňovanie blokov pri zmenách veľkosti podadresára. Ak je zmenšená veľkosť bežného súboru, sú prebytočné alokačné bloky ihneď uvoľnené. Naproti tomu, ak sú zrušené položky v podadresári, nebudú prebytočné alokačné bloky podadresára uvoľnené, dokiaľ nebude zrušený celý podadresár.

Tabuľka 1: Štruktúra položky adresára

Relatívna adresa			
DEC	**HEX**	**Veľkosť v bajtoch**	**Význam**
0	00	8	Meno súboru (1 až 8 znakov ASCII)
8	08	3	Prípona súboru (0 až 3 znaky ASCII)
11	0B	1	Vlastnosti súboru (atribúty)
12	0C	10	Rezervované
22	16	2	Čas poslednej zmeny súboru
24	18	2	Dátum poslednej zmeny súboru
26	1A	2	Prvý alokačný blok
28	1C	4	Veľkosť súboru

Každý podadresár obsahuje dve zvláštne položky. Prvá z nich je ukazovateľom na tzv. rodičovský adresár, to je adresár, v ktorom je položka definujúca tento podadresár. V poli vyhradenom pre prvý alokačný blok je číslo prvého alokačného bloku rodičovského adresára. Ak je rodičovským adresárom koreňový adresár, je hodnota poľa 0. Táto položka je v poli mena označená znakmi ".." . Druhá položka je v poli mena označená znakom "." a je ukazovateľom na začiatok daného podadresára. V poli prvého alokačného bloku je číslo prvého alokačného bloku pridelené podadresáru, v ktorom sa táto položka nachádza. Koreňový adresár nikdy položky "." a ".." neobsahuje. Nasledujúca tabuľka obsahuje štruktúru položky adresára (každá položka je zložená z ôsmich polí).

Meno súboru sa skladá z 8 znakov ASCII. V prípade, že je meno kratšie než 8 znakov, je sprava doplnené znakmi medzera (ASCII kód 20h alebo 32). Meno súboru musí obsahovať aspoň jeden iný znak, než je medzera. Pokiaľ sú v mene súboru písmená, mali by byť ukladané ako veľké. Väčšina programov totiž automaticky konvertuje malé písmená v menách súborov na veľké. Meno súboru by tiež nemalo obsahovať medzery medzi znakmi vlastného mena. Väčšina príkazov operačného systému DOS neumožňuje takéto meno spracovať. Prvý bajt poľa mena môže obsahovať buď prvý znak mena, alebo niektorú z hodnôt, ktorá má špeciálny význam:

Prvý bajt	Význam
00h	Položka nebola doteraz použitá
05h	Prvý znak mena súboru má hodnotu kódu E5h
E5h	Položka je voľná, súbor bol zmazaný

Hodnota prvého bajtu 00h je dôležitá pri prehľadávaní adresára. Adresár sa prehľadáva sekvenčne od prvej položky a takéto položky sú novým súborom prideľované od prvej voľnej položky v adresári. Preto ak narazíme pri prehľadávaní na položku 00h, môže byť prehľadávanie ukončené. Všetky ďalšie položky až do fyzického konca adresára sú určite neobsadené. Hodnota prvého bajtu 05h umožňuje použiť ako prvý znak mena súboru znak s kódom E5h. Používateľovi je táto zámena ukrytá, prebieha automaticky pri zápise mena súboru. Hodnota prvého bajtu E5h označuje položku, ktorá bola uvoľnená pri zrušení súboru. Zrušenie súboru na disku má za následok iba dve zmeny. V adresári je prvý bajt mena súboru zmenený na E5h a všetky alokačné bloky, ktoré doteraz patrili súboru, sú uvoľnené zmenou hodnoty príslušných položiek v tabuľke FAT.

Menom sa tiež odlišujú dve špeciálne položky, ktoré musí obsahovať každý podadresár. Sú to položky s menami "." a "..". Prvá z nich je ukazovateľom na samotný podadresár v dátovej oblasti, druhá je ukazovateľom na rodičovský adresár.

Prípona je uložená ako tri znaky ASCII. Rovnako ako meno súboru je prípona sprava doplnená medzerami (ASCII kód 20h alebo 32) v prípade, že nezaberá celé tri znaky. Na rozdiel od mena súboru nemusí obsahovať nijaký znak. Pokiaľ sú súčasťou prípony písmená, mali by byť uložené ako veľké a ani prípona by nemala obsahovať uprostred medzery.

Vlastnosti súboru (atribúty) sú uložené ako 8 bitov v jednom bajte. Každý bit reprezentuje jednu vlastnosť súboru alebo zvláštny význam položky adresára.

Bit 0 – ak je nastavený, potom zo súboru, ku ktorému je položka adresára pridelená, je možné iba čítať. Nie je možné doň zapisovať, vykonávať v ňom zmeny alebo ho zmazať. Tieto operácie možno vykonať až po zmene vlastnosti, napríklad pomocou príkazu ATTRIB.

Tabuľka 2: Atribúty súboru

Bit								
7	6	5	4	3	2	1	0	**Význam**
.	.	.	.	.	.	.	**0**	Súbor iba na čítanie (Read-only)
.	.	.	.	.	.	**0**	.	Skrytý súbor (Hidden)
.	.	.	.	.	**0**	.	.	Systémový súbor (System)
.	.	.	.	**0**	.	.	.	Položka návestia (Volume label)
.	.	.	**0**	.	.	.	.	Podadresár (Subdirectory)
.	.	**0**	.	.	.	.	.	Archivačný príznak (Archive)
0	**0**	.	.	.	.	.	.	Rezervované

Bit 1 – ak je nastavený, potom meno súboru ani údaje v ňom nebudú súčasťou bežného výpisu adresára (príkaz DIR). Ani ostatné príkazy operačného systému DOS nevedia pracovať so skrytými súbormi. Hodnota tohto príznaku sa dá meniť iba funkčným volaním.

Bit 2 – Má rovnaký význam ako bit 1. Ide o pozostatok z operačného systému CP/M. Spracovanie súboru je zhodné, či už je nastavený jeden, alebo druhý bit, prípadne obidva.

Bit 3 – Ak je nastavený, potom položka adresára obsahuje návestie disku (volume label). Táto položka môže byť iba v koreňovom adresári. Návestie disku má dĺžku 11 znakov ASCII. Na rozdiel od mena či prípony súboru môže obsahovať veľké aj malá písmená i medzery uprostred. V tejto položke adresára nie sú využívané polia veľkosti súboru ani prvého alokačného bloku. Pole času a dátumu posledného zápisu obsahujú čas aj dátum vytvorenia návestia disku.

Bit 4 – Ak je nastavený, položka obsahuje údaje o podadresári. Podadresáre sú na disku vytvárané rovnako ako dátové súbory. Preto sa aj všetky polia položky využívajú ako pri súboroch. Výnimkou je pole dĺžky súboru, ktoré je nastavené na 0. Dĺžka podadresára sa zistí jedine z počtu pridelených alokačných blokov v tabuľke FAT. Mená podadresárov nie sú súčasťou normálneho výpisu adresára (príkaz DIR).

Bit 5 – Ide o tzv. archivačný príznak, používaný pri zálohovaní a obnove súborov. Význam má iba pre súbory na pevnom disku. Súbory na disketách sa týmto spôsobom väčšinou nezálohujú. Archivačný bit je nastavený, kedykoľvek dôjde k zápisu alebo k zmene súboru. Nuluje sa pri každom zálohovaní súboru.

Čas poslednej zmeny je uložený ako tri celé binárne čísla bez znamienka v dvoch bajtoch slova.

Päťbitová oblasť vyhradená sekundám sa zvyšuje o 1 vždy po dvoch sekundách. Sekunda vytvorenia súboru je uložená násobkom tohto údaju dvoma. Čas poslednej zmeny je v položke vyplnený pri vytváraní súboru a neskôr sa pri každej zmene obsahu súboru nastaví aktuálny čas.

Dátum poslednej zmeny je rovnako ako čas poslednej zmeny uložený v dvoch bajtoch jedného slova ako tri celé binárne čísla bez znamienka.

Rok je v 7-bitovej oblasti uložený ako číslo od 0 do 199 dekadicky. Skutočný rok sa získa pričítaním dekadického čísla 1980. Tak je pokryté rozmedzie rokov 1980 až 2099. Dátum poslednej zmeny súboru sa po prvýkrát v položke adresára vyplní pri vytvorení súboru. Pri každej zmene obsahu súboru sa nahradí aktuálnym dátumom.

Prvý alokačný blok pridelený súboru má dĺžku dva bajty. Je to zároveň poradové číslo položky v tabuľke FAT, ktorá zodpovedá prvému alokačnému bloku súboru. Pri súboroch, ktorým nijaký alokačný blok nebol pridelený, a pri položkách návestia disku je hodnota tohto poľa 0. Zvláštny význam má pole prvého alokačného bloku pri položkách "." a ".." obsiahnutých v každom podadresári. V položke "." obsahuje prvý alokačný blok pridelený podadresáru, v ktorom sa položka nachádza. V položke ".."

Tabuľka 3: Čas poslednej zmeny

Bajt 17H							Bajt 16H									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	**Význam**
0	**0**	**0**	**0**	**0**	.	.	.	.	.	.	.	.	.	.	.	Hodina (0 – 23)
.	.	.	.	.	**0**	**0**	**0**	**0**	**0**	**0**	.	.	.	.	.	Minúta (0 – 59)
.	.	.	.	.	.	.	.	.	.	.	**0**	**0**	**0**	**0**	**0**	Sekunda / 2 (0 – 29)

Tabuľka 4: Dátum poslednej zmeny

Bajt 19H							Bajt 18H								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	**Význam**
0	**0**	**0**	**0**	**0**	**0**	**0**	.	.	.	.	.	.	.	.	.	Rok (0 – 119)
.	.	.	.	.	.	.	**0**	**0**	**0**	**0**	.	.	.	.	.	Mesiac (1 – 12)
.	.	.	.	.	.	.	.	.	.	.	**0**	**0**	**0**	**0**	**0**	Deň (1 – 31)

obsahuje číslo prvého alokačného bloku rodičovského adresára. Ak je hodnota poľa prvého alokačného bloku v položke ".." rovná 0, potom rodičovský adresár je koreňový adresár disku.

Veľkosť súboru v počte bajtov je uložená v štyroch bajtoch ako celé číslo bez znamienka. Veľkosť súboru zvyčajne nezodpovedá presne počtu bajtov pridelených alokačných blokov. Posledný alokačný blok súboru je málokedy využitý celý. Chybou môže dôjsť i k opačnej situácii, keď hodnota udaná v poli veľkosti súboru je väčšia než celková veľkosť pridelených alokačných blokov spolu. Za koniec súboru sa teda považuje buď koniec posledného alokačného bloku, alebo posledný bajt v položke adresára.

Dátová oblasť disku

Všetky dáta súborov a podadresárov sú uschované v dátovej oblasti, ktorá zaberá takmer celý disk okrem prvých niekoľkých sektorov, vyhradených pre zavádzací systém, tabuľky FAT a adresár. Priestor je súborom alebo podadresárom prideľovaný postupne po jednom alokačnom bloku. Počet sektorov jedného alokačného bloku závisí od formátu disku. Pri vytváraní alebo rozširovaní súboru sa pridelí nový alokačný blok až po zaplnení sektorov predchádzajúceho alokačného bloku. Informácie o obsadení alokačných blokov sú uchovávané v tabuľke FAT, kde sú označené aj chybné alokačné bloky. Stratégia prideľovania alokačných blokov je záležitosťou operačného systému DOS.

Výpočet logického čísla sektora

Logické čísla sektorov sa používajú na prerušenia pre priame čítanie z disku (INT 25h) alebo pre priamy zápis na disk (INT 26h). Logické číslo je možné získať týmto postupom (alokačný blok = klaster):

1. odčítame od čísla alokačného bloku 2,
2. výsledok vynásobíme počtom sektorov jedného alokačného bloku,
3. pričítame počet rezervovaných sektorov pre Boot,
4. pričítame (počet kópií tabuľky FAT * veľkosť tabuľky FAT v sektoroch),
5. pričítame (počet položiek v Root Dir * 32 / veľkosť sektorov v bajtoch).

Týmto výpočtom získame logické číslo prvého sektora, ďalší sektor má číslo vždy o jednotku vyššie.

Priame čítanie sektorov z disku (INT 25h)

Prerušenie INT 25h umožňuje priame čítanie sektorov z disku. Prerušenie 25h spolu s prerušením 26h sú v operačnom systéme DOS jediné, ktoré umožňujú jednoduchú prácu so sektormi na disku nezávisle od jeho logickej štruktúry. Pred použitím tohto prerušenia musia registre obsahovať nasledujúce parametre: **AL** – číslo diskového zariadenia (0=A, 1=B, 2=C, ...), **CX** – počet sektorov, ktoré sa majú čítať, **DX** – logické číslo prvého čítaného sektora, **DS** – bázová adresa pamäte pre čítané dáta, **BX** – relatívna adresa pamäte pre čítané dáta.

Prerušenie zmení obsahy všetkých registrov okrem registrov SP, CS, DS, ES a SS. Okrem toho v zásobníku zostane jedna 16–bitová hodnota, ktorá obsahuje stav registra príznakov pred vyvolaním prerušenia. V prípade, že používateľský program bude vyberať zo zásobníka hodnoty uložené pred prerušením, je potrebné z vrcholu zásobníka odstrániť jedno slovo (WORD), inak môže dôjsť ku kolízii. V prípade, že má sektor kapacitu 512 bajtov, a ak sektorov môže byť maximálne 65 536 (veľkosť registra DX), potom je maximálna veľkosť oblasti 32 MB. Táto hranica bola limitom pre všetky verzie DOS-u nižšie ako 4.00. Pretože tento problém bol pri vyššej verzii odstránený, má táto služba od verzie 4.00 navyše ešte jeden variant umožňujúci zadávať číslo sektora väčšie

Tabuľka 5: Rozdelenie disku na oblasti pri súborovom systéme FAT

Veľkosť oblasti v MB	Veľkosť alokačného bloku	Počet sektorov na 1 blok
0 až 127	2 KB	4
128 až 255	4 KB	8
256 až 511	8 KB	16
512 až 1023	16 KB	32
1024 až 2048	32 KB	64
2049 až 4096	64 KB	128

než 65 536. Registre musia obsahovať: **AL** – číslo diskového zariadenia (0=A, 1=B, 2=C, ...), **CX** – 0FFFFh, **DS:BX** – adresa štruktúry, kde štruktúra má dĺžku 10 bajtov s týmto obsahom: **offset 0-3** – 32-bitové číslo prvého sektora, **offset 4-5** – počet čítaných sektorov, **offset 6-7** – offset adresy pamäte pre čítanie/zápis sektorov, **offset 8-9** – segment adresy pamäte pre čítanie/zápis sektorov.

Po návrate z prerušenia je prípadný výskyt chyby indikovaný nastavením príznaku CF na 1. V prípade, že CF=0, čítanie prebehlo bez chýb a na segmentovej adrese, ktorá bola zadaná v registroch DS a BX pri volaní služby, sa začína blok prečítaných dát. V prípade chyby registre AL a AH obsahujú chybový kód. V **registri AL** je chybový kód poskytovaný operačným systémom DOS: **00h** – pokus o zápis na disketu chránenú proti zápisu, **01h** – neplatné číslo zariadenia, neznáma jednotka, **02h** – jednotka nie je pripravená, **03h** – neznámy príkaz, **04h** – chyba parity (CRC) , **06h** – chyba pri nastavovaní hlavy na požadovanú stopu, **07h** – disk má iný formát (HPFS, NTFS, ...), **08h** – sektor nebol nájdený, **0Ah** – chyba pri zápise, **0Bh** – chyba pri čítaní, **0Ch** – všeobecná chyba.

V **registri AH** je chybový kód poskytovaný programom obsluhy periférnych zariadení BIOS: **00h** – neidentifikovateľná chyba, **02h** – nebola nájdená značka začiatku sektora alebo je neplatná, **03h** – pokus o zápis na disketu, ktorá je chránená proti zápisu, **04h** – sektor s daným číslom na disku nie je, **08h** – chyba pri prístupe do operačnej pamäte, **10h** – chyba parity, **20h** – chyba technického vybavenia, **40h** – chyba pri nastavení hlavy na požadovanú stopu, **80h** – jednotka nie je pripravená.

Priami zápis sektorov na disk (INT 26h)

Prerušenie 26h umožňuje priamy zápis jedného alebo viacerých sektorov na určené miesto na disku. Pred samotným použitím prerušenia musia registre obsahovať nasledujúce parametre: **AL** – číslo diskového zariadenia (0=A, 1=B, 2=C, ...), **CX** – počet sektorov, ktoré sa majú zapísať, **DX** – logické číslo prvého zapisovaného sektora, **DS** – bázová (segmentová) adresa pamäte s dátami na zápis, **BX** – relatívna (offsetová) adresa dát na zápis.

Po návrate z prerušenia je prípadná chyba indikovaná nastavením príznaku CF. Ak príznak nie je nastavený, boli všetky dáta zapísané na disk bez chýb. V prípade, že je príznak CF nastavený na 1, znamená to, že došlo k chybe. Registre **AL** a **AH** obsahujú chybový kód, ktorý určuje príčinu chyby. Chybové kódy sú analogické s prerušením INT 25h. Takisto informácie o zmene registrov a zásobníku platia aj pri tomto prerušení (pozri INT 25h).

Deväťnásta časť: Pokračovanie

Zakladné programové vybavenie na obsluhu diskety a disku

Prerušenie INT 13h sprístupňuje najnižšiu úroveň ovládača disketových jednotiek a jednotiek pevných diskov. Tento ovládač je obsiahnutý v ROM-BIOS-e a prípadnom HD-BIOS-e z ROM-modulu na doske riadiacej jednotky pevných diskov. Pri vyvolaní INT 13h musí register AX obsahovať číslo služby v rozmedzí 0 -17h. Väčšina služieb sa používa pre diskety i pevné disky, niektoré však pracujú iba s jedným druhom jednotky. Register DL obsahuje číslo jednotky, s ktorou sa má daná operácia vykonať: bit 7 určuje, či ide o disketu (0) alebo o pevný disk (1); t. j. disketovým jednotkám zodpovedajú čísla 0,1,... a pevným diskom 80h,81h,... ROM-BIOS podporuje obsluhu nanajvýš 4 disketových jednotiek a 4 jednotiek pevných diskov. Po návrate z INT 13h hovorí príznak CF o úspešnosti operácie: 0 znamená úspech, 1 chybu. V druhom prípade obsahuje AH register kód chyby. Nasledujúca tabuľka poskytuje prehľad o jednotlivých službách prerušenia 13h:

INT 13h, služba AH=00h – Reset disketovej alebo diskovej jednotky. Spôsobí rekalibráciu riadiacej jednotky. **Vstup:** register DL – číslo jednotky, ktorá sa má rekalibrovať.

I**NT 13h, služba AH=01h** – Vráť chybový stav poslednej operácie. Register DL obsahuje číslo jednotky. **Vstup:** DL – číslo jednotky, **Výstup:** AH – kód chyby. Po zavolaní tejto služby pre pevný disk sa stav vynuluje, t. j. prípadné ďalšie čítanie môže viesť k chybným výsledkom

INT 13h, služba AH=02h – Čítaj sektory. **Vstup:** DL – číslo jednotky, DH –- číslo hlavy (0:n), CH – spodných 8 bitov čísla cylindra1 (0:n), CL - bity 6 a 7 sú bitmi 9 a 10 čísla cylindra bity 0:5 – číslo prvého sektora (1:n) , AL – počet sektorov (nanajvýš počet sektorov nachádzajúcich sa na 1. stope), ES:BX – adresa vyrovnávacej pamäte sektora(ov). **Výstup:** príznak CF – úspešnosť operácie: 1 – chyba, 0 – úspech, ES:BX – vyrovnávacia pamäť obsahuje dáta prečítané zo sektora(ov), AH – kód chyby, ak CF=1.

INT 13h, služba AH=03h – Píš sektory. Vstup: ako pri funkcii 02h, ES:BX – adresa vyrovnávacej pamäte s dátami, ktoré sa majú zapísať na disk. Výstup: CF – úspešnosť operácie: 1 – chyba, 0 – úspech AH – kód chyby, ak CF=1.

INT 13h, služba AH=04h – Verifikuj sektory. Kontroluje CRC súčtom špecifikované sektory. **Vstup:** ako pri funkcii 02h okrem vyrovnávacej pamäte (neprenášajú sa nijaké dáta). **Výstup:** CF – úspešnosť operácie: 1 – chyba, 0 – úspech, AH – kód chyby, ak CF=1.

INT 13h, služba AH=05h – Formátuj stopu. Vstup: DL,DH,CH – jednotka, hlava, cylinder (pozri funkciu 02h), ES:BX – deskriptor sektorov závislý od typu jednotky a počítača: **Disketa:** pre každý sektor 4-bajtová hodnota 'CHNS' (C-cylinder, H-hlava, N-číslo sektora, S-veľkosť sektora). S má hodnotu 0 pre 128 byte, 1 pre 256 byte, 2 pre 512 byte a 3 pre 1024 byte. **Pevný disk pre AT:** pre každý sektor 2-bajtová hodnota 'FN' (F – príznak, N – číslo sektora). Postupnosť polí N podmieňuje faktor "Interleave", t. j. rozdiel v čísle za sebou fyzicky nasledujúcich sektorov. **Pevný disk pre XT:** vyrovnávacia pamäť ES:BX sa vôbec nevyužíva, namiesto toho AL obsahuje "Interleave" faktor. **Výstup:** CF – úspešnosť operácie: 1 – chyba, 0 – úspech, register AH obsahuje kód chyby, ak CF=1. Pre pevné disky v AT sa pole F príslušného sektora nastaví na 80h, ak je sektor chybný.

INT 13h, služba AH=06h – Formátuj stopu a označ ju ako chybnú. Iba pevný disk a počítač XT. **Vstup:** DL, DH, CH – jednotka, hlava, cylinder (pozri funkciu 02h), register AL – interleave faktor. **Výstup:** CF – úspešnosť operácie: 1 – chyba, 0 – úspech, register AH obsahuje kód chyby, ak CF=1.

INT 13h, služba AH=07h – Formátuje disk od danej stopy. Iba pevný disk a počítač XT. **Vstup:** DL, DH, CH – jednotka, hlava, cylinder (pozri funkciu 02h), register AL – interleave faktor. **Výstup:** CF – úspešnosť operácie: 1 – chyba, 0 – úspech, register AH obsahuje kód chyby, ak CF=1.

INT 13h, služba AH=08h – Vráť fyzické parametre jednotky. Táto služba pracuje pre pevný disk, pri niektorých počítačoch i pre disketu. **Vstup:** DL – číslo jednotky. **Výstup:** DL – počet pripojených diskov primárnej riadiacej jednotky, DH – maximálne číslo hlavy, CH – nižších 8 bitov maximálneho čísla cylindra, CL – bity 6,7 sú bitmi 8,9 maximálneho čísla cylindra, bity 0:5 tvoria maximálne číslo sektora.

INT 13h, služba AH=09h – Iba pre pevný disk. Inicializácia jednotky pevného disku podľa tabuľky pevného disku. **Vstup:** DL – číslo jednotky pevného disku (>80h). **Výstup:** nijaký. Funkcia slúži na inicializáciu riadiacej jednotky daného pevného disku novými hodnotami v tabuľke pevného disku, ktoré mohol používateľ modifikovať. Počítače XT používajú jeden smerník na tabuľku pevných diskov (ukazuje na ňu prerušovací vektor 41h, adresa 0:104h). Tento smerník ukazuje práve na štyri za sebou nasledujúce tabuľky, prepínače na doske riadiacej jednotky pevného disku určujú použitie príslušnej z nich. Ak má register DL platný obsah (>80h), vykoná sa inicializácia oboch pevných diskov z príslušných tabuliek. Počítače AT používajú pre oba pevné disky zvláštny smerník na tabuľku (pre prvý disk vektor prerušenia 41h, pre druhý 46h, adresa 0:118h), ktoré ukazujú každý práve na jednu tabuľku. Funkcia vykoná inicializáciu jednotky špecifikovanej v registri DL.

INT 13h, služba AH=0Ah – "Dlhé" čítanie sektorov. Iba pre pevný disk. Funkcia spôsobí prečítanie sektorov ako v prípade funkcie 02h, ale za dátami vo vyrovnávacej pamäti sa zapíšu ešte kontrolné bajty – tzv. kód ECC slúžiaci na opravu niektorých chýb. Pri bežných radičoch sú to 4 bajty, niektoré radiče však používajú až 6 bajtov. Kód ECC sa v normálnom prípade vyhodnocuje interne riadiacou jednotkou, používateľ však môže prebrať jeho spracovanie sám. **Vstup:** ako funkcia 02h. **Výstup:** ako funkcia 02h, vo vyrovnávacej pamäti sa však za každým prečítaným sektorom nachádza kód ECC.

INT 13h, služba AH=0Bh – "Dlhý" zápis sektorov. Iba pre pevný disk. Funkcia spôsobí zápis sektorov na disk, pričom používateľ pripraví do vyrovnávacej pamäte za každý sektor i kód ECC. **Vstup:** ako funkcia 02h, ES:BX – smerník na vyrovnávaciu pamäť obsahujúcu dáta, za každým sektorom je uložený kód ECC. **Výstup:** ako funkcia 02h

INT 13h, služba AH=0Ch – Prechod hlavičky disku na určitú stopu, tzv. seek. Iba pre pevný disk. **Vstup:** DL, DH, CH – jednotka, hlava, cylinder ako funkcia 02h. **Výstup:** príznak CF=1, ak nastala chyba pri operácii, register AH obsahuje kód chyby, ak CF=1.

INT 13h, služba AH=0Dh – Alternatívny reset pevného disku. Pracuje podobne ako služba 00h, ale iba pre pevné disky. Vstup: register DL musí obsahovať číslo diskovej jednotky (>80h).

INT 13h, služba AH=0Eh – Rezervované na diagnostiku: Čítanie vyrovnávacej pamäte sektora. Iba pre pevný disk a počítač XT.

INT 13h, služba AH=0Fh – Rezervované na diagnostiku: Zápis do vyrovnávacej pamäte sektora. Iba pre pevný disk a počítač XT.

INT 13h, služba AH=10h – Test pripravenosti pevného disku. Iba pre pevný disk. Testuje sa najmä na účel zistenia, či disk dosiahol menovité otáčky a či neprebieha práve proces hľadania stopy (seek). **Vstup:** register DL – číslo jednotky. **Výstup:** AH – kód chyby (0, ak je disk pripravený).

INT 13h, služba AH=11h – Rekalibrácia pevného disku. Iba pre pevný disk. Hlavičky disku prejdú na cylinder 0. **Vstup:** register DL – číslo jednotky. **Výstup:** register AH – chybový kód (0, ak bez chyby).

INT 13h, služba AH=12h – Diagnostika vyrovnávacej pamäte sektora. Iba pre pevný disk a počítače XT.

INT 13h, služba AH=13h – Diagnostika jednotky pevného disku. Iba pre pevný disk a počítače XT. Vykonáva sa test čitateľnosti sektora 1 každej stopy.

INT 13h, služba AH=14h – Diagnostika riadiacej jednotky pevného disku (autotest). Iba pevný disk.

INT 13h, služba AH=15h – Vráť typ jednotky. **Vstup:** register DL – číslo jednotky. **Výstup:** register AH – typ jednotky (00 – nepripojená, 01 – disketová jednotka bez signálu výmeny média, 02 – disketová jednotka so signálom výmeny média, 03 – pevný disk).

INT 13h, služba AH=16h – Zisti, či nastala výmena média (diskety). Iba pre diskety a počítače AT. **Vstup:** register DL – číslo jednotky (<80h). **Výstup:** register AH = 0 a príznak CF=0, ak nenastala výmena diskety, AH = 6 a CF=0, ak nastala výmena diskety. Službu má význam aplikovať iba na disketové jednotky typu 2 zistené službou 15h.

INT 13h, služba AH=17h – Nastavenie typu média pred formátovaním. Iba pre diskety. Má význam aplikovať iba na disketové jednotky typu 2 zistené službou 15h.

Vstup: DL – číslo jednotky (<80h), register AL – typ média: (0 – nepoužitý kód, 1 – 360K disketa v 360K jednotke, 2 – 360K disketa v 1.2M jednotke, 3 – 1.2M disketa v 1.2M jednotke).

INT 13h, služba AH=17h – Vytvorenie tabuľky diskety pre daný typ média. Túto službu neposkytujú všetky verzie ROM-BIOS-ov/HD-BIOS-ov a platí iba pre diskety. **Vstup:** register DL – číslo jednotky (<80h), CH - spodných 8 bitov maximálneho čísla cylindra, CL – bity 6,7 sú bitmi 8,9 maximálneho čísla cylindra, bity 0:5 tvoria maximálne číslo sektora. **Výstup:** register AH – kód chyby, ak CF=1, registre ES:DI obsahujú smerník na tabuľku diskety zodpovedajúcu opísanému médiu.

Pokiaľ pri výkone niektorej služby INT 13h vznikne chyba, mal by používateľ službou **00h** alebo **0Dh** (pre pevný disk) znovu inicializovať jednotku a operáciu zopakovať, a to minimálne 2 – krát. V prípade stálej chyby, ak jej zdrojom nie je nesprávny argument dodaný používateľom, je zrejme spôsobená chybným médiom alebo jednotkou. Funkcie prerušenia 13h vracajú nasledujúce kódy chýb (pozri tabuľku 1).

V prípade, ak chce používateľ bližšie analyzovať zdroj chyby, sú v bloku pamäte na adrese 0:441h uložené tzv. "sense" bajty (pre štandardné riadiace jednotky 4 bajty), ktoré je možné prezrieť.

Pri počítačoch AT sa pri vykonávaní operácie, ktorá očakáva príchod prerušenia od radiča diskety alebo pevného disku, volá prerušenie 15h, služba 90h (zariadenie pracuje) s typom 1 pre disketu, resp. 0 pre pevný disk. Po ukončení operácie sa volá prerušenie 15h, služba 91h. Všetky služby, ktoré vyžadujú menovité otáčky disku (diskety), vykonajú aj po prípadnom spustení motora prerušenie 15h, službu 90h s typom FDh (čakanie na obrátky motora). Účelom tohto protokolu je umožniť vykonávanie inej úlohy počas čakania na I/O operáciu. Po operačným systémom MS-DOS sa táto možnosť nevyužíva, pretože ten nie je reentrantný a nepodporuje multiprogramovanie, preto sú uvedené služby prerušenia 15h nevyužité (vykoná sa hneď IRET).

Služby INT 13h používajú údaje z tabuliek diskiet a tabuliek pevných diskov. Tabuľka diskiet obsahuje niektoré fyzické parametre, ktoré sú potrebné pri programovaní riadiacej jednotky diskiet. Tabuľka diskiet má dĺžku 11 bajtov a smerník na ňu obsahuje vektor prerušenia 1Eh (adresa 0:78h). Má štruktúru podľa tabuľky 2.

Tabuľka pevných diskov sa nachádza na adrese uloženej na smerníku prerušenie 41h (0:104h), prípadne i 46h (0:118h) - pozri popis služby 09h prerušenia 13h. Tabuľka má dĺžku 12 bajtov pre počítače XT, 16 bajtov pre AT a má štruktúru podľa tabuľky 3.

Praktický príklad

Príklad, ktorý si dnes ukážeme, vám umožní zobraziť stromovú štruktúru disku, teda štruktúru adresárov a podadresárov. Aby sa na obrazovke zobrazila čo najväčšia stromová štruktúra, použijeme program view.asm z predošlej časti. Všetko to spojíme pomocou jednoduchého dávkového súboru.

Tabuľka 1: Možné chyby prerušenia INT 13h

Register AH	Opis chýb
00h	Nijaká chyba nenastala
01h	Volaná služba nie je implementovaná
02h	Sektor, na ktorom sa mala vykonať operácia, má chybnú značku adresy (bad address mark)
03h	Disketa je chránená proti zápisu
04h	Žiadaný sektor nebol nájdený (sector not found)
05h	Riadiaca jednotka pevného disku a pevný disk sa nedali inicializovať („resetnúť")
06h	Po poslednej operácii nastala výmena média diskety
07h	Proces inicializácie (služba 09h) nastala chyba
08h	Počas prenosu nastal prebeh DMA – radič DMA si nevyzdvihol úplne alebo dosť rýchlo od radiča jednotky údaje
09h	Počas prenosu DMA nastala chyba stránky (prechod cez hranicu medzi 64 KB blokmi pamäte
0Ah	Číslo sektora je mimo rozsahu (iba AT)
0Bh	Príslušná stopa pevného disku je označená ako chybná
10h	Počas čítania alebo verifikácie nastala chyba CRC súčtu (disketa), resp. neopraviteľná chyba kódu ECC (pevný disk). Ak sa údaje predsa majú prečítať, treba to urobiť službou 0Eh
11h	Iba pevný disk. Počas čítania nastala chyba, ktorú bolo možné opraviť pomocou kódu ECC. V registri AL je počet opravených bitov.
20h	Riadiaca jednotka hlási vnútornú chybu
40h	Hľadaná stopa sa nenašla (track not found)
80h	Pri výkone funkcie došlo k prekročeniu stanoveného časového limitu
AAh	Iba AT. Pevný disk nedosiahol menovité otáčky
BBh	Riadiaca jednotka diskiet zistila neznámu chybu
CCh	Iba AT. Počas zápisu/formátovania nastala chyba
E0h	Iba AT. Nenastala nijaká chyba
FFh	Nastala chyba pri operácii s pevným diskom, ale nebolo možné načítať 4 tzv. "sense" bajty, ktoré hovoria o príčine vzniku chyby

Tabuľka 2

Posunutie	Veľkosť	Obsah
+0	1	bity 0:3 – rýchlosť krokovania (v ms) bity 4:7 – oneskorenie odtiahnutia hlavičiek (v jednotkách 16 ms)
+1	1	bit 0 – príznak použitia DMA (0 – áno) bity 1:7 - rýchlosť pritlačenia hlavičiek (v jednotkách 2 ms)
+2	1	časový interval pred vypnutím motora, ak sa nevykonáva nijaká operácia (v jednotkách 55 ms)
+3	1	veľkosť sektora (0=128, 1=256, 2=512, 3=1024)
+4	1	najväčšie číslo sektora na stope (t. j. počet sektorov na stope)
+5	1	dĺžka medzery (gap) pri čítaní/zápise
+6	1	dĺžka dát pri prenose, ak sa nenastaví iná
+7	1	dĺžka medzery pri formátovaní
+8	1	zapĺňací znak pri formátovaní (obyčajne F6h)
+9	1	ustálenie hlavy po pritiahnutí (v ms)
+A	1	rýchlosť nábehu motora na otáčky (v jednotkách 1/8s)

Tabuľka 3

Posunutie	Veľkosť	Obsah
+0	2	počet cylindrov
+2	1	počet hlavičiek
+3	2	číslo cylindra, od ktorého vyššie sa má zápis vykonávať nižším zápisovým prúdom
+5	2	číslo cylindra, od ktorého sa má vykonávať predkompenzácia
+7	1	maximálny počet bitov, ktorý má byť opravený kódom ECC (pri vyššom počte bitov hlásiť neopraviteľnú chybu)
+8	1	parametre krokovania: bit 7 – zákaz opakovania pri chybe stopy (seek error) bit 6 – zákaz opakovania pri ECC chybe bity 2 až 0 – rýchlosť krokovania
+9	1	štandardný časový interval, počas ktorého sa musí operácia uskutočniť (time out value)
+A	1	časový interval, počas ktorého sa musí uskutočniť operácia formátovania
+B	1	časový interval, počas ktorého sa musí uskutočniť operácia kontroly jednotky
+C	2	Iba AT: parkovacia zóna (cylinder, kam majú byť presunuté hlavičky pred zastavením motora)
+E	1	Iba AT: počet sektorov na stope
+F	1	Iba AT: rezervované pre budúce rozšírenia

```
;VTREEH - Charles Petzold, 1985
CSEG Segment
Assume  CS:CSEG, DS:CSEG, ES:CSEG, SS:CSEG

          Org     005Ch
FCB       Label   Byte
                  Org     0080h
DefaultDTA        Label   Byte
                  Org     0100h
Entry:            Jmp     Begin

SearchAsciiz      db  ?,":\*.*",0
                  db  '(C) Copyright Charles '
                  db 'Petzold, 1985'
DriveError:       db  'Invalid disk drive$'
DosVersError      db 'Requires DOS 2.0 +$'
FirstOrNext       db  0
LevelsIn          dw  0
SearchString      db  "\*.*",0
SearchPointer     dw  3 + Offset SearchAsciiz
DtaPointer        dw  Offset EndProg
DashCount         dw  ?
HiddenMark        db  42

Begin:      Cmp     AL,0FFh
            Jnz     DriveSpecOK
            Lea     DX,DriveError
ErrorExit:  Mov     AH,9
            Int     21h
            Int     20h

DriveSpecOK:      Mov     AH,30h
                  Int     21h
                  Cmp     AL,2
                  Jae     DosVersOK
                  Lea     DX,DosVersError
                  Jmp     ErrorExit
```

```
DosVersOK:          Mov     AL,[FCB]
                    Or      AL,AL
                    Jnz     NotDefault
                    Mov     AH,19h
                    Int     21h
                    Inc     AL
NotDefault:         Mov     DL,AL
                    Add     AL,'@'
                    Mov     [SearchAsciiz],AL
                    Cld

MainLoop:           Mov     DX,[DTAPointer]
                    ov      AH,1Ah
                    Int     21h
                    Mov     BX,[LevelsIn]
                    Add     BX,BX
                    Cmp     [FirstOrNext],0
                    Jnz     FindNextFile
                    Mov Word Ptr [SubDirCounter+BX],0
                    Mov DX,Offset SearchAsciiZ
                    Mov     CX,12h
                    Mov     AH,4Eh
                    Int     21h
                    Jmp     Short TestMatch

FindNextFile:       Mov     AH,4Fh
                    Int     21h
TestMatch:          Jnc     TestAttr
                    Jmp     NoMoreFiles
TestAttr:           Mov SI,[DTAPointer]
                    Cmp Byte Ptr [SI + 21],10h
                    Jz      FoundDirEntry
                    Cmp Byte Ptr [SI + 21],12h
                    Jnz     FindNextFile

FoundDirEntry:      Add     SI,30
                    Cmp     Byte Ptr [SI],'.'
                    Jz      FindNextFile

          Inc Word Ptr [SubDirCounter + BX]
          Mov     CX,[LevelsIn]
          Jcxz    LookAheadSearch
          Cmp Word Ptr [SubDirCounter + BX],1
          Jz      NoSpaceIn
                    Sub     BX,BX
IndentLoop:         Mov     AL,179
                    Test  Word Ptr [SubDirCounter + BX],8000h
                    Jz      GotContinueChar
                    Mov     AL,' '
GotContinueChar:    Call    PrintChar
                    Push    CX
                    Mov     CX,16
BlankLoop:          Mov     AL,' '
                    Call    PrintChar
                    Loop    BlankLoop
                    Pop     CX
                    Inc     BX
                    Inc     BX
                    Loop    IndentLoop
NoSpaceIn:          Cmp  Word Ptr [SubDirCounter + BX],1
                    Jnz     LookAheadSearch
                    Mov     CX,[DashCount]
DashPrint:          Mov     AL,196
                    Call    PrintChar
                    Loop    DashPrint

LookAheadSearch:    Push    SI
                    Mov     SI,[DtaPointer]
                    Mov     DI,Offset DefaultDTA
                    Mov     DX,DI
                    Mov     CX,43
                    Rep     Movsb
                    Pop     SI
                    Mov     AH,1Ah
                    Int     21h

CheckIfAnyMore:     Mov     AH,4Fh
                    Int     21h
                    Jc      CantFindAnother
                    Cmp     Byte Ptr [DefaultDTA + 21],10h
                    Jz      GotOne
                    Cmp     Byte Ptr [DefaultDTA + 21],12h
                    Jnz     CheckIfAnyMore
GotOne:             Mov     AL,194
                    Cmp     Word Ptr [SubDirCounter + BX],1
                    Jz      GotGoodChar
                    Mov     AL,195
                    Jmp     Short GotGoodChar

CantFindAnother:    Mov     AL,196
                    Cmp   Word Ptr [SubDirCounter + BX],1
                    Jz      GotGoodChar
                    Mov     AL,192
                    Or    Word Ptr [SubDirCounter + BX],8000h

GotGoodChar:        Call    PrintChar
                    Mov     AL,196
                    Call    PrintChar
                    Mov     AL,' '
                    Call    PrintChar
                    Mov     CX,13
                    Mov     DI,[SearchPointer]

PrintNameLoop:      Lodsb
                    Or      AL,AL
                    Jz      EndOfName
                    Stosb
                    Call    PrintChar
                    Loop    PrintNameLoop

EndOfName:          Mov     AL,' '
                    Mov     SI,[DTAPointer]
                    Cmp     Byte Ptr [SI + 21],12h
                    Jnz     NowPrint
                    Mov     AL,HiddenMark

NowPrint:           Call    PrintChar
                    Mov     [DashCount],CX

FixUpSearch:        Mov     [SearchPointer],DI
                    Inc     [SearchPointer]
                    Mov     SI,Offset SearchString
                    Mov     CX,5
                    Rep     Movsb
                    Inc    [LevelsIn]
                    Mov     [FirstOrNext],0
                    Add     [DtaPointer],43
                    Jmp     MainLoop

NoMoreFiles:        Cmp     [LevelsIn],0
                    Jz      Terminate
                    Test  Word Ptr [SubDirCounter + BX],7FFFh
                    Jnz     BackUpOneDir
                    Mov     AL,13
                    Call    PrintChar
                    Mov     AL,10
                    Call    PrintChar
BackUpOneDir:       Mov     DI,Offset SearchAsciiz
                    Mov     CX,70
                    Mov     AL,0
                    Repnz   Scasb
                    Dec     DI
                    Mov     CX,64
                    Mov     AL,'\'
                    Std
                    Repnz   Scasb
                    Repnz   Scasb
                    Inc     DI
                    Mov     [SearchPointer],DI
                    Inc     [SearchPointer]
                    Mov     SI,Offset SearchString
                    Mov     CX,5
                    Cld
                    Rep     Movsb
                    Dec     [LevelsIn]
                    Mov     [FirstOrNext],1
                    Sub     [DtaPointer],43
                    Jmp     MainLoop
Terminate:          Int     20h

PrintChar:          Push    DX
                    Mov     DL,AL
                    Mov     AH,2
                    Int     21h
                    Pop     DX
                    Ret

SubDirCounter       Label   Word
EndProg             equ     64 + Offset SubDirCounter
CSEG                EndS
                    End Entry
```

Opis programu

Program je typu COM a začína sa na návesti **Begin**. Vykoná sa test verzie MS-DOS-u pomocou funkcie 30h (INT 21h). Výstupom tejto funkcie je v registri AL hlavné číslo verzie a v registri AH vedľajšie číslo verzie. V prípade, že je verzia MS-DOS-u nižšia ako 2.0, program vypíše chybové hlásenie **Requires DOS 2.0**. Ak testy prebehli bez problémov, pokračuje sa na návesti **DosVersOK**. Z PSP (Program Segment Prefix) zistíme číslo jednotky, ak je to nula, použijeme na zistenie službu 19h. Služba 19h prerušenia INT 21h vráti v registri AL číslo práve nastaveného bežného disku (0=A, 1=B, 2=C, ...). Takto získanú hodnotu zväčšíme o 1, prekonvertujeme na znak ASCII a zapíšeme na začiatok reťazca SearchAsciiz. Od návestia MainLoop sa začína časť zdrojového kódu, ktorá zobrazí celú stromovú štruktúru práve aktívneho disku na obrazovku. Najprv nastavíme DTA pomocou služby **1A (INT 21h)**. Pomocou tejto služby je možné nastaviť adresu oblasti, ktorá je používaná na prenosy dát medzi diskom a pamäťou skupinou služieb FCB. Vykonanie tejto operácie sa vyžaduje pred volaním služieb 4Eh a 4Fh. **Služba 4E** prerušenia INT 21h umožňuje hľadanie súboru zadaného ako reťazec **ASCIIZ**, ktorého adresa je v registroch **DS:DX**. V registri CX sa zadávajú atribúty hľadaného súboru. Ak je **CX=0**, sú do vyhľadávania zahrnuté iba normálne súbory. V prípade, že je nastavený niektorý z bitov 1, 2 alebo 4 (skryté a systémové súbory, podadresáre), potom sú do vyhľadávania okrem bežných súborov zahrnuté aj tie. Ak je nastavený bit 3 (návestie disku), hľadá sa položka s návestím disku. Bity 0 a 5 sú ignorované. Príznak **CF=0**, súbor bol nájdený a DTA obsahuje tieto informácie *(pozri v tab. 4)*.

Tabuľka 4

Offset	Obsah položky DTA
0 – 20	interné premenné služby
21	atribúty nájdeného súboru
22 – 23	čas z adresára
24 – 25	dátum z adresára
26 – 29	veľkosť súboru v bajtoch
30 – 42	meno súboru ako reťazec ASCIIZ

Ak príznak **CF=1**, v registri AX je chybový kód (2 – chybná cesta, 12h – nenájdená zodpovedajúca adresárová položka).

Po vykonaní tejto služby pokračujeme testom atribútov, konkrétne zisťujeme, či ide o adresár (cmp byte ptr [si+21],10h) alebo skrytý adresár (cmp byte ptr [si+21],12h). Ak nie je splnená ani jedna podmienka, pokračuje sa testovaním ďalšieho súboru. Na to sa použije prerušenie **INT 21**, **služba 4Fh**. Táto služba umožňuje hľadať ďalšie súbory po úspešnom dokončení služby 4Eh. Služba 4Fh vráti rovnaké parametre ako služba 4Eh. Od návestia **IndentLoop** sa začína vykresľovanie štruktúry. Pretože na výpis znakov je použité INT 21, službou 02h je možné výpis z obrazovky presmerovať napríklad do súboru.

Preklad programu

Najprv odpíšte celý program do súboru napr.: **vtreeh.asm**. Spustiteľný COM súbor potom dostanete takto: **Tasm.exe vtreeh.asm** a **Tlink.exe vtreeh.obj /t**. Ak ste pri odpisovaní neurobili chybu, vytvorí sa súbor **vtreeh.com**. Teraz do súboru **strom.bat** odpíšte nasledujúcu dávku príkazov.

```
rem ******************
rem súbor strom.bat
rem ******************
@echo Prosim cakaj!
@echo off
vtreeh.com > zmazat.z
view.com zmazat.z
del zmazat.z
cls
```

Ak to všetko máte, súbory **view.com, vtreeh.com, strom.bat** nakopírujte napr. do adresára DOS (predpokladá sa, že v súbore autoexec.bat je nastavená cesta do tohto adresára pomocou príkazu PATH). Teraz kedykoľvek napíše príkaz strom, zobrazí sa vám stromová štruktúra adresárov a podadresárov práve aktívneho disku.

Vysvetlivky

1 Cylinder – valec na disku (zväzok stôp na všetkých diskových plochách, ktoré majú rovnakú polohu).

Literatúra

[1] Ralf Brown: Interrupt List — Release 33. Pittsburgh PA, U.S.A. 1993.
[2] Vlastislav Černý: MS-DOS 6.22. Kopp 1996.
[3] P. Heřman: Príručka systémového programátora. Tesla Eltos 1990.
[4] David Jurgens: HelpPC 2.10.
[5] ABSHelp 2.05. ABSoft Olomouc.
[6] Michal Brandejs: MS-DOS 6, kompletní průvodce. Grada 1993.

Dvadsiata časť: Práca so súbormi

Základným spôsobom uchovávania dát je ich ukladanie do súborov. Vytváranie, hľadanie, mazanie či otváranie a zatváranie, čítanie a zápis súborov patria k najčastejšie používaným operáciám v používateľských programoch.

Funkčné volanie DOS, zamerané na prácu so súbormi, je možné rozdeliť do dvoch skupín podľa spôsobu práce so súbormi. Prvá skupina využíva na ovládanie súboru riadiaci blok súboru (**FCB** – File Control Block). Tieto funkčné volania sú v operačnom systéme DOS zachované z predchádzajúceho operačného systému **CP/M**. Práca s týmito funkčnými volaniami je náročnejšia, pretože používateľ musí sám budovať podstatnú časť riadiaceho bloku súboru. Ďalšou nevýhodou je, že pomocou nich je možné spracúvať iba súbory umiestnené v aktuálnom adresári. Nepodporujú spracovanie stromovej štruktúry adresárov. Dôvodom na zachovanie týchto funkcií v systéme DOS bolo zachovanie kompatibility s prvými verziami DOS-u, ktoré používali iba tieto tradičné funkčné volania.

Druhú skupinu tvoria funkčné volania využívajúce na ovládanie súboru **kanál** (Handle). Tieto rozšírené funkčné volania sú v operačnom systéme DOS hlavným prostriedkom určeným na vykonávanie operácií so súbormi. Ich výhodou je výrazné zjednodušenie práce. Používateľský program sa obracia na súbor iba prostredníctvom čísla kanále, ktoré je každému súboru pridelené operačným systémom pri otvorení alebo vytvorení súboru. Používateľ v tomto prípade nemusí vytvárať nijaký riadiaci blok súboru. Pomocou rozšírených volaní je možné pracovať aj s podadresárovou štruktúrou vrátane vytvárania a rušenia adresárov. Pri tvorbe nových programov pod operačným systémom DOS sa odporúča používať ako prostriedok na prácu so súbormi iba skupinu rozšírených funkčných volaní.

Sekvenčný a priamy prístup k súboru

Tak ako prvá skupina funkčných volaní aj druhá skupina umožňuje dve metódy čítania a zápisu dát do súboru. **Sekvenčný zápis** je spôsob zápisu, pri ktorom sú vždy novo zapisované dáta uložené na koniec súboru. Dáta sú v súbore, do ktorého je zapisované iba sekvenčne, uložené v rovnakom poradí ako boli zapisované. **Sekvenčné čítanie** umožňuje používateľskému programu čítať dáta postupne od začiatku súboru smerom k jeho koncu. Dáta zo súboru vytvoreného metódou sekvenčného zápisu budú čítané v rovnakom poradí, ako boli zapisované.

Priamy zápis, naopak, umožňuje zapisovať dáta na ľubovoľné miesto v súbore. To dovoľuje nielen zapisovať dáta na koniec súboru, ale aj meniť dáta predtým uložené. **Priame čítanie** umožňuje čítať dáta zo súboru v ľubovoľnom poradí. Metóda čítania a zápisu sa neurčuje pri otvorení súboru, ale až pri vlastnom vykonávaní operácie čítania alebo zápisu dát do súboru. Používateľský program môže využívať pri práci s tým istým súborom tak sekvenčný, ako aj priamy prístup, pričom pri zmene prístupu netreba otvorený súbor zatvárať.

Druhá skupina – rozšírené funkčné volania

Najprv si povieme niečo o druhej skupine funkčných volaní, pretože táto skupina sa používa častejšie. Ovládanie súborov pomocou rozšírených funkčných volaní je pre používateľský program veľmi výhodné a ľahké. Pri vytváraní alebo otváraní súboru je mu operačným systémom priradené číslo kanála. Kanál je oblasť v pamäti, vyhradená operačným systémom, kde sa budú uchovávať všetky dôležité údaje o danom súbore. Kanál je úplne pod kontrolou operačného systému a používateľ doň nezasahuje. Od chvíle, keď je súboru priradené číslo kanála, obracia sa používateľský program na súbor iba pomocou čísla kanála. Operačný systém udržuje pre každý otvorený súbor ukazovateľ aktuálnej pozície v súbore. Každý zápis alebo čítanie sa začína od bajtu, na ktorý je nastavený ukazovateľ aktuálnej pozície. Po ukončení operácie je ukazovateľ nastavený na posledný čítaný alebo zapisovaný bajt. Používateľský program môže ukazovateľom v súbore ľubovoľne pohybovať.

Pri otváraní, vytváraní, premenúvaní alebo rušení súboru musí byť zadaná špecifikácia súboru. Pri rozšírených funkčných volaniach sa špecifikácia súboru zadáva vždy ako reťazec ASCIIZ (American Standart Code of International Interchange Zero), t. j. reťazec ukončený bajtom s hodnotou nula. Pri vytváraní súboru je možné zadať vlastnosti súboru. Pri otváraní súboru môžeme určiť prístupové práva, (napríklad môže otvoriť súbor len na čítanie alebo len na zápis) a spôsob zdieľania súboru, ktorý určuje, či môže zo súboru čítať alebo doňho zapisovať aj iný používateľský program než ten, ktorý ho otvoril. Riadenie zdieľania súborov je dôležité napríklad pri vyvolávaní jedného programu druhým. Zdieľanie súborov je možné uplatniť iba za predpokladu, že je povolené operačným systémom.

Špeciálne kanály

Operačný systém má preddefinovaných päť špeciálnych kanálov, ktoré môžu byť využívané používateľskými programami (pozri tabuľku 1).

Napriek tomu, že špeciálne kanály nemusí používateľský program pred použitím otvárať, môže ich kedykoľvek zatvoriť. Štandardné vstupné zariadenie je možné využívať iba na čítanie. Na štandardné výstupné a štandardné chybové zariadenie je povolené iba zapisovať. Štandardné výstupné a vstupné zariadenie možno presmerovať, naopak, štandardné chybové zariadenie nemožno presmerovať. Na zápis a čítanie dát zo všetkých

Tabuľka 1: Špeciálne kanály

Číslo kanála	Zariadenie
00h	Štandardné vstupné zariadenie (stdin)
01h	Štandardné výstupné zariadenie (stdout)
02h	Štandardné chybové zariadenie (stderr)
03h	Štandardné pomocné I/O zariadenie (stdaux)
04h	Štandardné tlačové zariadenie (stdprn)

štandardných zariadení môžeme použiť rovnaké funkcie ako na prácu so súbormi, ktoré boli otvorené pomocou kanála. Štandardné chybové zariadenie je výhodné používať na výstup správ, ktoré nemajú byť zobrazené na obrazovke počítača, nezávisle od presmerovania štandardného výstupu. Zo štandardného pomocného I/O a štandardného tlačového zariadenia možno čítať i zapisovať doň. Štandardné pomocné I/O zariadenie je implicitne smerované na prvý port sériového/paralelného adaptéra COM1. Štandardné tlačové zariadenie je implicitne smerované na paralelnú tlačiareň LPT1. Obidve zariadenia je možné ľubovoľne presmerovať pomocou funkcie DOS-u 46h, prerušenie INT 21h.

Počet súborov, ktoré môžu byť v danej chvíli otvorené pomocou funkčných volaní, závisí od hodnoty premennej FILES. Táto premenná sa nachádza v konfiguračnom súbore config.sys. V premennej FILES sa zadáva celkový počet súborov, ktoré môžu byť zároveň otvorené v celom systéme. Tento počet môže byť až 255 súborov. Ak je v premennej FILES zadaná hodnota 20 a viac, potom jeden proces môže zároveň používať až 20 kanálov, pričom však nesmie celkový počet kanálov otvorených v celom systéme prekročiť hodnotu zadanú vo FILES. Štandardné vstupné, výstupné a chybové zariadenia využívajú jeden spoločný kanál, štandardné tlačové a štandardné pomocné I/O zariadenia disponujú takisto jedným kanálom. Keď konfiguračný súbor config.sys premennú FILES neobsahuje, je počet otvorených súborov v systéme implicitne nastavený na hodnotu 8. Určenie maximálneho počtu kanálov v premennej FILES sa týka iba súborov, ktoré sú otvárané pomocou funkčných volaní. Na súbory otvárané tradičnými volaniami (pomocou FCB) nemá táto premenná nijaký vplyv.

Tabuľka 2: Chybové kódy rozšírených volaní

Chybový kód		
HEX	**DEC**	**Význam**
01	1	Chybné číslo funkčného volania
02	2	Súbor nebol nájdený
03	3	Neplatná adresárová cesta
04	4	Priveľa otvorených súborov
05	5	Nebol povolený prístup k súboru
06	6	Chybné číslo kanála
07	7	Porušenie údajov riadiacich obsadenie pamäte
08	8	Nedostatočná pamäť
09	9	Chybná adresa bloku v pamäti
0A	10	Chybný opis parametrov procesora príkazov
0B	11	Chybný formát
0C	12	Chybný kód prístupu
0D	13	Chybné dáta
0E	14	Rezervované
0F	15	Neznáme označenie jednotky
10	16	Pokus o zrušenie aktuálneho adresára
11	17	Nie je rovnaké zariadenie
12	18	Nie je viac súborov
13	19	Pokus o zápis na disketu chránenú proti zápisu
14	20	Neznáma jednotka
15	21	Jednotka nie je pripravená
16	22	Neznámy príkaz
17	23	Chyba dát
18	24	Požiadavka nesprávne zadaná
19	25	Chyba pri nastavovaní hlavy na požadovanú stopu
1A	26	Neznámy typ média
1B	27	Sektor nebol nájdený
1C	28	V tlačiarni nie je papier
1D	29	Chyba pri zápise
1E	30	Chyba pri čítaní
1F	31	Neidentifikovateľná chyba
20	32	Chyba pri zdieľaní súborov
21	33	Chyba pri pokuse o prístup k zdieľanému súboru
22	34	Nesprávne vykonaná výmena disku
23	35	Nie je voľný riadiaci blok súboru
24	36	Prekročená kapacita vyrovnávacej pamäte pre zdieľanie súborov
25 - 31	37 - 49	Rezervované
3D	61	Zásobník požiadaviek na tlač je plný
3E	62	Nedostatok miesta pre tlač súboru
3F	63	Tlačený súbor bol zrušený
48	72	Presmerovanie disku alebo tlače je zastavené
49 –4F	73 – 79	Rezervované
50	80	Súbor existuje
51	81	Rezervované
52	82	Nie je možné vytvoriť položku v adresári
53	83	Požiadavka bola zrušená obsluhou preruš. 24h
54	84	Priveľa presmerovaní
55	85	Zdvojené presmerovanie
57	87	Nesprávny parameter

Rozšírené funkčné volania na prácu so súbormi

Rozšírené funkčné volania na prácu so súbormi zabezpečujú vytváranie a rušenie adresárov, otváranie a vytváranie súborov, zmenu vlastností súborov, rušenie súborov, čítanie a zápis, posuv ukazovateľa pozície v súbore, vyhľadávanie súborov, otváranie a zatváranie kanálov.

INT 21H, funkcia 39H – Vytvorenie podadresára. Ukazovateľ na reťazec ASCIIZ obsahujúci špecifikáciu vytváraného podadresára sa odovzdá v registroch DS (segmentová adresa reťazca ASCIIZ) a DX (offsetová adresa reťazca ASCIIZ). V prípade chyby je nastavený príznak CF, register AX obsahuje chybový kód (pozri tabuľku 2). Ešte raz pripomenieme, že segmentová adresa v registroch DS:BX musí ukazovať na začiatok reťazca ASCII, ktorý je ukončený bajtom s hodnotou 0.

Reťazec nemusí obsahovať úplnú špecifikáciu nového podadresára vrátane zariadenia a aktuálneho adresára. Zadaná cesta však musí obsahovať iba existujúce adresáre. V prípade, že pri pokuse o vytvorenie adresára dôjde k nejakej chybe, nijaký adresár sa nevytvorí. Vo volajúcom programe môžete použiť funkciu 59h na bližšie určenie príčiny chyby.

```
Príklad:  ...
adresár   db 'C:\temp',0
          ...
          mov ah,39h
          mov dx,seg adresar
          mov ds,dx
          mov dx,offset adresár
          int 21h
          jc chyba
          ...
```

INT 21H, funkcia 3AH – zrušenie podadresára. Ukazovateľ na reťazec ASCIIZ obsahujúci špecifikáciu rušeného podadresára sa odovzdá v registroch DS a DX podobne ako v predchádzajúcom prípade. Reťazec nemusí obsahovať celú špecifikáciu adresára so zadaným zariadením a implicitným adresárom. Zadaná cesta však môže obsahovať iba existujúce podadresáre. Podadresár, ktorý má byť zrušený, nesmie byť aktuálny a musí byť prázdny, t. j. nesmie obsahovať súbory ani podadresáre. V prípade chyby bude nastavený príznak CF a register AX bude obsahovať chybový kód. Ak CF=1 adresár bol zrušený a AX obsahuje kód chyby (napríklad: 3 – cesta nebola nájdená, 5 – nepovolený prístup, 16 – pokus o zrušenie aktuálneho adresára), ak CF=0 adresár nebol zrušený.

```
Príklad:  ...
adresár   db 'C:\temp\prog',0
          ...
          mov ah,3Ah
          mov dx,seg adresar
          mov ds,dx
          mov dx,offset adresár
          int 21h
          jc chyba
          ...
```

INT 21H, funkcia 3BH – zmena implicitného adresára. Ukazovateľ na reťazec je opäť v registroch DS:DX. Špecifikácia adresára nesmie byť dlhšia ako 64 znakov. Tu treba pripomenúť, že implicitný adresár sa neustále udržuje pre každé diskové zariadenie. Použitím tejto funkcie je možné zmeniť implicitný adresár aj na zariadení, ktoré nie je práve aktuálne (aktuálne zariadenie je to, ktorého meno systém vypisuje ako súčasť štandardnej správy v riadku – napríklad C:\TEMP>). V prípade chyby sa nastaví príznak CF a register AX bude obsahovať kód chyby (3 – cesta nebola nájdená).

INT 21H, funkcia 3CH – vytvorenie súboru. Funkcia 3Ch otvára súbor určený na zápis. V prípade, že súbor neexistuje, funkcia ho vytvorí a otvorí. Ak súbor existuje, skráti funkcia jeho dĺžku na 0 bajtov a otvorí ho. Meno súboru sa funkcii odovzdáva ako reťazec ASCIIZ, ktorého adresa sa nachádza v registroch DS:DX. Vlastnosti vytváraného súboru sa odovzdávajú v registri CX (0 – normálny, 1 – iba na čítanie, 2 – skrytý, 4 - systémový). V prípade chyby sa nastaví príznak CF a register AX bude obsahovať kód chyby (3 – cesta nebola nájdená, 4 – nie je voľné číslo súboru, 5 – nepovolený prístup). Ak vytvorenie prebehne bez chyby, bude v registri AX číslo kanála otvoreného súboru (Handle). Reťazec musí obsahovať špecifikáciu súboru. V špecifikácii je možné uvádzať aj podadresáre. Pokiaľ nie je uvedené zariadenie, použije sa aktuálne zariadenie, ak nie je uvedená adresárová cesta od koreňového adresára, použije sa ako východiskový implicitný adresár diskového zariadenia. Zadaná cesta môže obsahovať iba existujúce podadresáre.

Popis práce Funkcie 3Ch: najprv sa prehľadá adresár a zistí, či súbor, ktorý má byť vytvorený, už existuje. Pokiaľ súbor zadaného mena existuje, skráti sa jeho dĺžka v adresári na 0 bajtov, súbor sa otvorí, pridelí sa mu číslo kanála a ukazovateľ aktuálnej pozície v súbore sa nastaví na jeho začiatok. Pokiaľ sa súbor v adresári nenachádza, funkcia ho vytvorí. Vytvorí aj položku v adresári, pridelí súboru vlastnosti určené pri vstupe registrom CX, otvorí ho a priradí súboru číslo kanála, pomocou ktorého sa bude volajúci program na súbor odvolávať. Pri otváraní existujúceho súboru musia byť vlastnosti súboru zadané v registri CX zhodné s vlastnosťami v adresári. V prípade, že sa nezhodujú, súbor nebude otvorený a funkcia ohlási chybu. Chybový kód v tomto prípade bude 05. Vlastnosti sa v registri CX zadávajú v rovnako, ako sú uvedené v položke adresára. Vyžíva sa iba nižší bajt registra CX, vyšší bajt má obsah 0. Po otvorení súboru je možné jeho vlastnosti zmeniť pomocou funkcie 43h.

Pomocou funkcie 3Ch je možné otvárať aj nesúborovo orientované zariadenia – napríklad tlačiareň alebo štandardné vstupné a výstupné zariadenie. Meno zariadenia sa uvedie rovnako ako špecifikácia súboru v reťazci ASCIIZ. Meno zariadenia sa uvádza bez dvojbodky, napríklad PRN. Register CX, ktorý pri otváraní súboru musí obsahovať vlastnosti otváraného súboru, nemá v tomto prípade nijaký význam. Po otvorení pracuje volajúci program so zariadením rovnako, ako keby spracúval súbor. Jediným obmedzením je, že nemožno použiť priamy prístup, dáta musia byť čítané alebo zapisované sekvenčne. V prípade, že pri pokuse o vytvorenie alebo otvorenie súboru vznikne chyba, nedôjde ani k vytvoreniu nového súboru, ani k modifikácii existujúceho súboru.

```
Príklad:            ...
súbor               db 'C:\temp\internet.txt',0
číslo_súboru        dw ?
                    ...
                    mov ah,3Ch
                    mov cx,0
                    mov dx,seg súbor
                    mov ds,dx
                    mov dx,offset súbor
                    int 21h
                    jc chyba
                    mov číslo_súboru,ax
                    ...
```

INT 21h, funkcia 3Dh – Otvorenie súboru – služba podľa mena v reťazci ASCIIZ špecifikovaného adresou v DS:DX otvorí existujúci súbor. V registri AL sa odovzdávajú nasledujúce atribúty otvorenia súboru:

Tabuľka 3: Atribúty otvorenia súboru

Bity	Stavy	Význam
7	0	synovský proces súbor dedí
	1	súbor je privátny
6 - 4	000	kompatibilita s FCB
	001	inému procesu je zakázané čítanie aj zápis
	011	inému procesu je zakázané čítanie
	100	nie sú nijaké obmedzenia
3	0	
2 - 0	000	súbor otvorený na čítanie
	001	súbor otvorený na zápis
	010	súbor otvorený na čítanie aj zápis

Bit 7 registra AL obsahuje tzv. príznak dedičnosti. Má vplyv na spracovanie súboru procesmi, ktoré budú spustené ako podprocesy procesu otvárajúceho súbor. Pokiaľ je príznak dedičnosti nastavený na 1, bude otvorený iba pre volajúci podprogram a pre jeho podprocesy bude uzatvorený. Ak je príznak dedičnosti rovný nule, bude otváraný súbor prístupný pomocou čísla kanála pre volajúci program i pre všetky jeho podprocesy.

Bity 6, 5, 4 určujú režim zdieľania súboru. Majú význam hlavne pri spracovaní súboru inými paralelne bežiacimi procesmi. Režim zdieľania súborov má však zmysel iba vtedy, ak je v činnosti systémový program na zistenie zdieľania súborov **SHARE**. Režim zdieľania súboru určuje, ktoré operácie sa môžu so súborom vykonávať paralelne bežiace programy. Nastavený môže byť jeden z režimov (pozri tabuľku 3).

- *Kompatibilný režim* sa nastavuje automaticky pre súbory, ktoré sú otvárané službami neumožňujúcimi zadanie režimu zdieľania, napríklad funkcia 3Ch alebo služby pracujúce so súborom pomocou riadiaceho bloku súboru (FCB). Pri nastavení kompatibilného režimu zdieľania je súbor možné otvoriť aj niekoľkokrát, no musí byť vždy uvedený kompatibilný režim, inak je ďalšie otvorenie odmietnuté. Pokiaľ sa má kompatibilný režim zmeniť, musí sa najprv súbor uzatvoriť tak, aby nebol ani raz otvorený, potom znovu otvoriť so zadaním požadovaného režimu zdieľania.
- *Zákaz prístupu* spôsobí, že nie je povolené žiadne ďalšie otvorenie súboru v nijakom režime zdieľania ani s nijakým prístupovým právom. Pred ďalším otvorením súboru musí byť takto zdieľaný súbor bezpodmienečne najprv uzatvorený.
- *Zákaz zápisu* spôsobí, že nie je povolené ďalšie otvorenie súboru na zápis.
- *Zákaz čítania*, naopak, znamená, že nie je možné znovu otvoriť súbor na čítanie.
- *Prístup povolený* je režim, ktorý dovoľuje ďalšie otváranie súborov bez obmedzenia tak pre čítanie, ako aj zápis.

Režim zdieľania sa pre každý súbor určuje pri prvom otvorení a zostáva v platnosti až do jeho uzatvorenia. Ak je otvorený viacnásobne, zostáva režim v platnosti až do jeho posledného uzatvorenia. Ďalším zvláštnym prípadom je spracovanie režimu zdieľania pri otváraní súborov označených vlastnosťou read-only (iba na čítanie). Ak je súbor označený read-only otváraný v kompatibilnom režime zdieľania súborov, je ešte pred pokusom o otvorenie režim zdieľania zmenený na "zákaz zápisu". To znamená, že súbor, ktorý bol po prvýkrát otvorený v režime zákaz zápisu, môže byť otvorený po druhýkrát v kompatibilnom režime. Pri súbore, ktorý nie je označený vlastnosťou read-only, by podobná situácia spôsobila vznik závažnej chyby.

Pokiaľ je v platnosti zdieľanie súborov a otvorené súbory sú dedené podprocesmi, ktoré budú spúšťané rodičovským procesom, potom sa dedí pre každý súbor aj režim zdieľania. Rovnako v prípade, že kanál, ktorý je priradený otvorenému súboru, je duplikovaný (funkcie 45h a 46h), sú na nový kanál prenesené aj vlastnosti vyplývajúce z režimu zdieľania súborov.

Bit 3 je rezervovaný pre operačný systém DOS a musí mať vždy v okamihu funkčného volania 3Dh hodnotu 0.

Bity 2, 1 a 0 určujú režim, v ktorom bude súbor spracovaný. Volajúci program si vlastne vyhrádza prístupové práva, ktoré bude uplatňovať pri práci so súborom cez kanál, ktorého číslo práve vykonávaná funkcia 3Dh vráti v registri AX. Môže byť nastavený jeden z režimov (pozri tabuľku 3). Volajúci program môže po otvorení súboru v niektorom režime práce vykonávať iba operácie, ktoré režim zadaný pri otvorení povoľuje.

Pokiaľ nie je v platnosti režim zdieľania súborov, môže volajúci proces a, samozrejme, aj iné procesy otvárať súbory ľubovoľne, s ľubovoľným prístupom aj viacnásobne. Jedinou výnimkou sú súbory označené read-only. Takýto súbor je možné otvárať iba v režime "súbor na čítanie".

Pomocou funkcie 3Dh môžu programy otvárať aj nesúborovo orientované zariadenia, ako tlačiareň alebo štandardné vstupné a výstupné zariadenie. Meno nesúborovo orientovaného zariadenia sa uvedie rovnako ako špecifikácia súboru v reťazci ASCIIZ. Meno zariadenia sa uvádza bez dvojbodky, napríklad PRN. Pre prácu s nesúborovo orientovanými zariadeniami platia všetky zásady vyplývajúce z režimu zdieľania a režimu práce podobne ako pri súboroch. Po otvorení kanála priradeného zariadenia môže volajúci program zapisovať a čítať zo zariadenia rovnako, ako keby pracoval so súborom. Jediný rozdiel je v tom, že nie je možné nastavovať ukazovateľ aktuálnej pozície, to znamená, že nie je možné používať priamy prístup.

Pokiaľ došlo k chybe, je nastavený príznak CF a v registri AX je kód chyby (2 – súbor nebol nájdený, 3 – cesta nebola nájdená, 4 – nie je voľné číslo súboru, 5 – nepovolený prístup, 0Ch – chybné atribúty otvorenia súboru). V prípade, že sa služba skončila bez chyby, je v registri AX číslo súboru. Reťazec musí obsahovať špecifikáciu súboru. V špecifikácii je možné uvádzať aj podadresáre. Pokiaľ nie je uvedené zariadenie, použije sa aktuálne zariadenie, ak nie je uvedená adresárová cesta od koreňového adresára, použije sa ako východiskový implicitný adresár diskového zariadenia. Zadaná cesta môže obsahovať iba existujúce podadresáre. Nasledujúci príklad ukazuje otvorenie súboru.

```
Príklad:            ...
súbor               db 'C:\temp\internet.txt',0
číslo_súboru        dw ?
                    ...
                    mov ah,3Dh
                    mov al,00010010b ;Atribúty
                    mov dx,seg súbor
                    mov ds,dx
                    mov dx,offset súbor
                    int 21h
                    jc chyba
                    mov číslo_súboru,ax
                    ...
```

INT 21h, funkcia 3Eh – uzatvorenie kanála. Služba sa používa na zatvorenie kanála, ktorý bol otvorený pomocou funkcie 3Dh alebo 3Ch alebo vznikol duplikáciou iného kanála. Číslo zatváraného kanála sa musí zadať v registri BX. Ak sa vyskytla nejaká chyba, je príznak CF nastavený na 1 a v registri AX je kód chyby (6 – chybné číslo súboru alebo súbor nie je otvorený). Funkciou 3Eh je možné uzatvoriť všetky kanály, ktoré sú otvorené, vrátane rezervovaných kanálov 0 až 4. Uzatvorením sa kanál uvoľní, nemôže byť ďalej využívaný na prístup k súboru alebo na zariadenie, na ktoré umožňoval prístup pred uzatvorením. Uvoľnený kanál môže byť využitý pre iný súbor alebo zariadenie. Uzatvorenie kanála, ktorý bol využitý na prístup k súboru, má za následok vykonanie zmien v adresári zariadenia, kde je súbor uložený. Obsah položky obsahujúcej dátum a čas sa zmení v prípade, že do súboru bol cez zatváraný kanál vykonaný zápis. Položka obsahujúca dĺžku súboru je prepísaná aktuálnou dĺžkou v okamihu uzatvorenia.

```
Príklad:  ...
          mov ah,3Eh
          mov bx, číslo_súboru
```

```
        int 21h
        jc chyba
        ...
```

INT 21h, funkcia 3Fh – čítanie zo súboru. Funkcia na základe čísla súboru (kanála) odovzdaného v registri BX z neho prečíta počet bajtov zadaných v registri CX a uloží ich do pamäte od adresy určenej registrami DS:DX. Pokiaľ sa operácia skončila bez chyby, je príznak CF=0 a v registri AX je počet skutočne prečítaných bajtov. V prípade, že sa vyskytla chyba, je príznak CF=1 a v registri AX je kód chyby (5 – nepovolený prístup, 6 – chybné číslo súboru alebo súbor nie je otvorený).

Funkciou 3Fh je možné čítať dáta zo všetkých kanálov, ktoré sú otvorené, či už sú priradené otvoreným súborom, alebo nesúborovým zariadenia. Pri každom čítaní zo súboru sa ukazovateľ aktuálnej pozície v súbore posunie o počet skutočne prečítaných bajtov. To znamená, že ak volajúci program číta dáta sekvenčne od začiatku až do konca, nemusí sa vôbec o nastavenie ukazovateľa starať.

Segmentová adresa DS:DX musí ukazovať na dostatočne veľký úsek voľnej pamäte, kam budú čítané dáta umiestnené. Pôvodný obsah pamäte bude prepísaný. Pri čítaní nie je zaručené, že bude prečítaných toľko bajtov, koľko určuje register CX. Môže byť z rôznych dôvodov prečítaných menej bajtov, preto skutočný počet prečítaných bajtov sa po návrate z funkcie 3Fh nachádza v registri AX. Najčastejšie je príčinou rozdielu to, že medzi ukazovateľom pozície v súbore a koncom súboru nie je požadovaný počet bajtov. Pokiaľ je pri návrate z čítania, ktoré prebehlo úspešne (CF=0), v registri AX hodnota 0, znamená to, že súbor bol prečítaný celý, ukazovateľ aktuálnej pozície je nastavený na koniec súboru. Iným prípadom, keď funkcia 3Fh môže prečítať menej znakov, než sa požaduje, je čítanie zo štandardného vstupného a výstupného zariadenia. Jedno volanie môže totiž prečítať najviac jeden riadok zadaný z klávesnice.

```
Príklad 1:          ...
pole                db 1024 dup (?)
                    ...
                    mov ah,3Fh
                    mov bx,číslo_súboru
                    mov cx,1024
                    mov dx,seg pole
                    mov ds,dx
                    mov dx,offset pole
                    int 21h
                    jc chyba
                    mov počet_prečítaných_bajtov,ax
                    ...
alebo
Príklad 2:          ...
buffer              db 1024 dup (?)
                    name      db "test.txt",0
                    ...
read:               lea dx,name
                    mov ax,3D00h
                    int 21h
                    jc error1
                    mov bx,ax
                    mov cx,1024
                    lea dx,buffer
no_all:             mov ah, 3Fh
                    int 21h
                    jc error2
                    cmp ax,0
                    jz go
                    sub cx,ax
                    jz go
                    add dx,ax
                    jmp no_all
                    ...
error1:             ...
error2:             ...
go:                 ...
```

INT 21h, funkcia 40h – zápis do súboru. Funkcia zapíše na daný kanál určený počet bajtov z pamäte. Číslo kanála musí byť v registri BX. Register CX určuje, koľko bajtov má byť zapísaných. Dvojica registrov DS:DX musí obsahovať adresu zapisovaných dát. Ak operácia prebehla úspešne, je CF=0 a v registri AX je počet skutočne zapísaných bajtov. Pokiaľ došlo pri zápise k chybe, nastaví sa príznak CF a v registri AX je kód chyby (5 – nepovolený prístup, 6 – chybné číslo súboru alebo súbor nie je otvorený). Pomocou funkcie 40h je možné zapisovať dáta pomocou všetkých kanálov, ktoré sú práve otvorené, či už sú priradené otvoreným súborom, alebo nesúborovo orientovaným zariadeniam. Súbory, samozrejme, musia byť otvorené v režime práce, ktorý zápis umožňuje, a nesmú byť v adresári označené vlastnosťou read-only. Pri každom zápise do súboru sa ukazovateľ aktuálnej pozície v súbore posunie o počet skutočne zapísaných bajtov. Pokiaľ pri začatí zápisu nie je ukazovateľ nastavený na koniec súboru, budú pôvodné dáta prepísané novo zapisovanými. Ak je ukazovateľ aktuálnej pozície nastavený na koniec súboru, budú novo zapisované dáta pripojené na koniec súboru. Funkcia 40h umožňuje volajúcemu programu skrátiť súbor. Program najprv nastaví ukazovateľ pozície v súbore za posledný bajt, ktorý má v súbore zostať, a potom nastaví v registri CX miesto počtu zapisovaných bajtov hodnotu 00h. Potom vykoná funkciu 40h. Tým sa skráti dĺžka súboru. Dáta, ktoré boli v súbore za ukazovateľom aktuálnej pozície v okamihu skracovania, budú zničené.

Segmentová adresa DS:DX, ako sme už uviedli, musí ukazovať na začiatok dát, ktoré budú zapísané do súboru alebo na zariadenie. Dĺžka dát musí korešpondovať s hodnotou v registri CX. Pri zápise dát však nie je zaručené, že bude zapísaný plný počet znakov. Môže sa stať, že bude zapísaných menej znakov, než udáva register CX. Skutočný počet zapísaných bajtov je po vykonaní funkcie v registri AX. Často je príčinou rozdielu medzi požadovaným a skutočným počtom zapísaných bajtov plné diskové médium. Ak funkcia vráti v AX hodnotu 0, znamená to, že diskové médium je úplne zaplnené. Tu treba zdôrazniť, že v priebehu zápisu do súboru na diskete sa v žiadnom prípade nesmie zmeniť médium. Systém nevykonáva kontrolu, či je vložená tá istá disketa. Zmena diskety môže viesť k poškodeniu dát uložených na médiu. Pokiaľ pri pokuse o zápis vznikne chyba, žiadne dáta nie sú zapísané, ukazovateľ aktuálnej pozície v súbore sa nezmení.

```
Príklad 1:          ...
pole                db 1024 dup (?)
                    ...
                    mov ah,40h
                    mov bx,číslo_súboru
                    mov cx,1024
                    mov dx,seg pole
                    mov ds,dx
                    mov dx,offset pole
                    int 21h
                    jc chyba
                    mov počet_prečítaných_bajtov,ax
                    ...
```

Príklad 2 ukazuje, ako je možné využiť funkciu 40h na zápis na štandardné výstupné zariadenie. Nasledujúci kód vypíše text na obrazovku iba v prípade, ak nie je štandardné výstupné zariadenie presmerované.

```
Príklad 2:          ...
správa              db "Výpis správy na štandardné"
                    db 32,"Input/Output zariadenie"
                    db 0dh, 0ah
                    ...
                    push cs
                    pop ds
                    mov dx,offset správa
                    mov bx,0001h
                    mov cx,32
                    mov ah,40h
                    int 21h
                    ...
```

INT 21h, funkcia 41h – zrušenie súboru. Služba 41h umožňuje zrušiť zadaný súbor. Špecifikácia súboru, ktorý sa má zrušiť sa odovzdá ako reťazec znakov v kóde ASCIIZ, ukončený bajtom s hodnotou 0. Registre DS a DX musia obsahovať adresu reťazca. Pokiaľ súbor nie je možné zrušiť, nastaví sa príznak CF a v registri AX je chybový kód (2 – súbor nenájdený, 5 – nepovolený prístup). Ak špecifikácia súboru neobsahuje zadanie zariadenia, bude súbor hľadaný na aktuálnom zariadení. Ak nie je v špecifikácii zadaná adresárová cesta od koreňového adresára, použije sa ako východiskový implicitný adresár diskového zariadenia. Špecifikácia súboru nesmie obsahovať znak hviezdičkovej a otáznikovej konvencie. Pomocou funkcie 40h nie je možné zrušiť súbor, ktorý je v adresári označený vlastnosťou read-only. V prípade, že dôjde pri pokuse o zrušenie súboru k chybe, súbor sa nezruší ani sa nezmení obsah položky adresára.

```
Príklad:  ...
Súbor     db 'C:\temp\internet.txt',0
          ...
          mov ah,41h
          mov dx,seg súbor
          mov ds,dx
          mov dx,offset súbor
          int 21h
          jc chyba
          ...
```

Literatúra

[1] Ralf Brown: Interrupt List - Release 33. Pittsburgh PA, U.S.A. 1993.
[2] Vlastislav Černý: MS-DOS 6.22. Kopp 1996
[3] P. Heřman: Příručka systémového programátora. Tesla Eltos 1990.
[4] David Jurgens: HelpPC 2.10.
[5] ABSHelp 2.05, ABSoft Olomouc.
[6] Michal Brandejs: MS-DOS 6. Kompletní průvodce. Grada 1993.

Dvadsiataprva časť: Pokračovanie

Pri tvorbe nových programov je vhodné využívať iba funkcie (služby), ktoré patria do skupiny rozšírených volaní. Tradičné (pôvodné) funkcie sa zachovali iba z dôvodu kompatibility programov vytvorených pre prvú verziu operačného systému DOS. Aj keď sa tieto funkcie dnes veľmi nepoužívajú, stále sa s nimi môžete stretnúť v niektorých zdrojových kódoch. Z tohto dôvodu by bolo vhodné aby sme si aj o nich niečo napísali. Okrem toho dokončíme aj opis rozšírených funkčných volaní z minulého čísla.

INT 21h, funkcia 42h – Posun ukazovateľa aktuálnej pozície v súbore. Funkcia umožňuje vykonávať posun ukazovateľa aktuálnej pozície v otvorenom súbore. Register BX musí obsahovať číslo kanálu otvoreného súboru. Dvojica registrov CX a DX sa musí naplniť 32-bitovou hodnotou (myslí sa celé číslo bez znamienka) posunu ukazovateľa v súbore. V prípade, že je hodnota posuvu menšia než 10000H musí register CX vždy obsahovať hodnotu nula.

Register AL určuje metódu, ako bude posuv vykonaný. Funkcia vráti volajúcemu programu novú hodnotu ukazovateľa aktuálnej pozície ako 32-bitové číslo v registroch DX a AX. Pokiaľ dôjde k chybe, nastaví sa príznak CF a v registri AX je miesto hodnoty ukazovateľa chybový kód (1 – chybné číslo funkcie, 6 – chybné číslo súboru alebo súbor nie je otvorený).

Register AL určuje, akým spôsobom sa zaistí nová hodnota ukazovateľa aktuálnej pozície. Ak **AL=0** – posuv sa vykoná od začiatku súboru, **AL=1** – posuv sa vykoná od súčastnej hodnoty ukazovateľa aktuálnej pozície, **AL=2** – posuv sa vykoná od súčastného konca súboru. Pokiaľ použijete metódu posuvu 0 a v registroch CX a DX sa zadá hodnota nula, nastaví sa ukazovateľ na začiatok súboru. Ak použijeme metódu posuvu 2 a v CX a DX bude hodnota 0, nastaví sa ukazovateľ na koniec súboru. Posuv ukazovateľa sa vždy vykonáva smerom ku koncu súboru. Ukazovateľ je možné nastaviť aj za koniec súboru. Tým sa ale ešte nemení pozícia konca súboru. Zmení sa, až keď je vykonaný zápis za koniec súboru. Takto je možné napríklad vyhradiť si priestor v zakladanom súbore, prípadne zisťovať, či je na zariadení dostatok miesta pre predpokladané množstvo údajov. Je treba však pamätať na to, že pokiaľ vykonáme posun ukazovateľa aktuálnej pozície za koniec súboru a nezapíšeme žiadne údaje do priestoru medzi koncom súboru a nastavenou pozíciou, bude tento priestor v súbore obsahovať nedefinované údaje. Funkcia 42h nevykonáva žiadnu kontrolu, či je ukazovateľ pozície nastavený ešte do súboru alebo už ukazuje za koniec súboru. Nekontroluje sa ani, či v prípade zápisu na miesto, kam je nastavený ukazovateľ, je možné údaje uložiť. Môže sa stať, že ukazovateľ aktuálnej pozície je nastavený tak ďaleko za koniec súboru, že zápis do tohto miesta nie je možný, pretože na disku nie je dosť miesta. Táto chyba sa prejaví až pri pokuse o zápis do súboru.

```
Príklad:  ...
          mov ah,42h
          mov al,0
          mov bx,číslo_súboru
          mov cx,0
          mov dx,1024
          int 21h
          jc chyba
          ...
```

INT 21h, funkcia 43h – Zmena vlastností súboru. Služba umožňuje zistiť alebo meniť vlastnosti, ktoré sú priradené súboru v adresári. Špecifikácia súboru, ktorého vlastnosti majú byť zistené alebo zmenené, sa odovzdá funkcii ako reťazec znakov ASCII, ukončený bajtom s hodnotou nula. Registre DS a DX musia obsahovať adresu reťazca ASCIIZ, register AL určuje, či má služba zistiť alebo zmeniť vlastnosti súboru. Register CX obsahuje vlastnosti súboru takto: **5 bit** – archivované, **2 bit** – systémový súbor, **1 bit** – skrytý súbor, **0 bit** – súbor určený iba pre čítanie. Pokiaľ operácia zisťovala atribúty, sú odovzdané v registri CX, pokiaľ je príznak CF=0. Inak je v AX kód chyby (1 – chybné číslo funkcie, 2 – súbor nenájdený, 3 – cesta nenájdená alebo súbor neexistuje, 5 – atribúty nemôžu byť zmenené).

```
Príklad:  ...
súbor     db 'C:\vtipy.txt',0
          ...
          mov ah,43h
          mov al,1
          mov cx,1
          mov dx,seg súbor
          mov ds,dx
          mov dx,offset súbor
          int 21h
          jc chyba
          ...
```

INT 21h, funkcia 44h – Riadenie ovládačov zariadení. Služba je určená na komunikáciu so štandardnými ovládačmi zariadení. Ak došlo ku chybe, je nastavený príznak CF=1 a register AX obsahuje chybový kód (1 – chybné číslo funkcie, 4 - nie je dostupné žiadne číslo súboru, 5 – nepovolený prístup, 6 – chybné číslo súboru alebo súbor nie je otvorený, 0Dh – chybné údaje, 0Fh – chybné číslo disku). Podrobne je táto služba opísaná v literatúre [6]. Nasledujúci príklad nastaví režim binárneho výstupu na konzolu. V tomto režime sa nekontrolujú znaky CTRL+C, CTRL+S, a CTRL+P, čím sa výpis urýchli.

```
Príklad:  ...
          mov ax,4400h
          mov bx,1
          int 21h
          mov dh,0
          or dl,20h
          mov ax,4401h
          int 21h
          ...
```

INT 21h, funkcia 45h – Duplikovanie prideleného kanálu. Funkcia priradí otvorenému kanálu ešte jedno číslo kanálu. Číslo pôvodného kanálu sa funkcii odovzdá v registri BX, číslo novo prideleného kanálu vráti funkcia v registri AX. Službou je možné duplikovať všetky otvorené kanály, vrátane kanálov 0 až 4, ktoré sú vyhradené pre rezervované zariadenia. Ak dôjde ku chybe, nastaví sa príznak CF a v registri AX je chybový kód (4 – nie je dostupné žiadne číslo súboru, 6 – chybné číslo súboru alebo súbor nie je otvorený). Pokiaľ dôjde pri pokuse o otvorenie duplikujúceho kanálu ku chybe, žiadny nový kanál sa neotvorí.

```
Príklad:  ...
          mov ah,45h
          mov bx,číslo_súboru
          int 21h
          jc chyba
          mov bx,ax
          mov ah,3Eh
          int 21h
          jc chyba
          ...
```

INT 21h, funkcia 46h – Vynútené duplikovanie prideleného kanálu. Služba zabezpečí, aby bol k číslu už otvoreného kanálu priradený ešte jeden kanál. Číslo pôvodného kanálu sa zadáva v registri BX, číslo novo prideleného kanálu v registri CX. Ak je príznak CF=1, došlo ku chybe, a v registri AX je chybový kód (4 – nie je dostupné žiadne číslo súboru (kanálu), 6 – chybné číslo súboru alebo súbor nie je otvorený). Pre duplikáciu sa môže použiť ako kanál, ktorý je voľný, tak aj kanál, ktorý je priradený inému súboru alebo zariadeniu. Môže to byť aj kanál rezervovaného zariadenia. Kanál, ktorý je zadaný v registri CX, sa najprv uzatvorí, pokiaľ bol otvorený, a potom sa znovu otvorí a pridelí sa rovnakému zariadeniu alebo súboru, ktorému je pridelený kanál, uvedený v registri BX. V prípade, že je duplikovaný kanál otvorený pre súbor, určuje operačný systém pre pôvodný aj nový kanál len jeden ukazovateľ aktuálnej pozície. Každý posuv ukazovateľa pri čítaní, zápise alebo vyhľadávaní pozície v súbore sa odrazí aj na druhom kanále, otvorenom pre ten istý súbor. Pomocou funkcie 46h je možné napríklad dynamicky presmerovať štandardné zariadenie v priebehu práce programu. Stačí súbor alebo zariadenie, kam má byť presmerovanie vykonané, vynútene duplikovať kanálom, rezervovaným pre štandardné zariadenie. Pôvodne otvorený kanál sa uzatvorí a potom znovu otvorí, ale už pre nové zariadenie alebo súbor. Príklad ukazuje presmerovanie údajov, vystupujúcich na štandardnú tlačiareň do súboru tlac.txt pomocou vynúteného duplikovaného kanálu.

```
Príklad:  ...
súbor     db 'C:\tlac.txt',0
          ...
          mov dx, offset súbor
          mov ax,3D02h
          int 21h
          jc chyba
          mov bx,ax
          mov cx,0004h
          mov ah,46h
          int 21h
          ...
```

INT 21h, funkcia 47h – Zistenie implicitného adresára. Služba umožňuje zistiť úplnú špecifikáciu implicitného adresára ktoréhokoľvek diskového zariadenia. Funkcii sa odovzdá číslo diskového zariadenia v registri DL a adresa pamäti, kam má byť špecifikácia adresára umiestnená v registroch DS:SI. Pokiaľ dôjde ku chybe, nastaví sa príznak CF a v registri AX bude chybový kód. Číslo diskového zariadenia v registri DL označuje zariadenie, na ktorom má byť implicitný adresár zistený (0 – aktuálne zariadenie, 1 – zariadenie A, 2 - B, 3 – C ...). Segmentová adresa DS:SI musí ukazovať na úsek pamäti dlhej 64 bajtov. Špecifikácia nebude obsahovať určenie zariadenia (napr. B:) a nebude obsahovať znak "\" pre označenie koreňového adresára na zariadení. Špecifikácia bude zapísaná do pamäti ako reťazec ASCIIZ. To znamená, že pokiaľ bude implicitným adresárom zariadenia koreňový adresár, vráti funkcia iba bajt 0. V prípade chyby zostane úsek pamäti na adrese DS:SI nezmenený.

```
Príklad:  ...
buffer    db 64 dup(0)
          ...
          mov ah,47h
          mov dl,3
          mov si, seg buffer
          mov ds,si
          mov si, offset buffer
          int 21h
          jc chyba
          ...
```

INT 21h, funkcia 4Eh – Hľadanie súboru. Služba 4E prerušenia INT 21h umožňuje hľadanie súboru zadaného ako ASCIIZ reťazec, ktorého adresa je v registroch DS:DX. V registri CX sa zadávajú atribúty hľadaného súboru. Ak je CX=0, sú do vyhľadávania zahrnuté iba normálne súbory. V prípade, že je nastavený niektorý z bitov 1, 2 alebo 4 (skryté a systémové súbory, podadresáre), potom sú do vyhľadávania okrem bežných súborov zahrnuté aj tieto. Ak je nastavený bit 3 (návestie disku), hľadá sa položka s návestím disku. Bity 0 a 5 sú ignorované. Príznak CF=0, súbor bol nájdený a DTA obsahuje tieto informácie:

Offset	Obsah položky DTA
0 – 20	interné premenné služby
21	atribúty nájdeného súboru
22 – 23	čas z adresára
24 – 25	dátum z adresára
26 – 29	veľkosť súboru v bajtoch
30 – 42	meno súboru ako ASCIIZ reťazec

Ak príznak CF=1, v registri AX je chybový kód (2 – chybná cesta, 12h – nenájdená zodpovedajúca adresárová položka).

```
Príklad:  ...
buffer    db 128 dup(0)
file      db 'c:\zvuk\*.zip',0
          ...
          mov ah,1Ah
          mov dx, seg buffer
          mov ds,dx
          mov dx, offset buffer
          int 21h
          mov ah,4Eh
          mov cx,0
          mov dx, offset file
          int 21h
          jc chyba
          ...
```

INT 21h, funkcia 4Fh – Hľadanie ďalšieho súboru. Služba pokračuje v prehľadávaní adresára hľadá ďalší súbor, ktorého meno a vlastnosti vyhovujú špecifikácii, zadanej v službe 4Eh. Vyrovnávacia pamäť sa nesmie od minuého volania funkcie 4Eh alebo 4Fh zmeniť. Funkcia 4Fh modifikuje obsah vyrovnávacej diskovej pamäti podľa údajov o ďalšom nájdenom súbore. Pokiaľ služba súbor nenašla, alebo pri pokuse o hľadanie došlo ku chybe, nastaví sa príznak CF a v registri AX bude chybový kód. Táto funkcia musí vždy nasledovať až po použití funkcie 4Eh, ktorá vyplní v prvých bajtoch vyrovnávacej pamäte údaje pre ďalšiu funkciu. Ak už raz bola služba 4Eh použitá, môže sa funkcia 4Fh opakovať dovtedy, pokým nie sú spracované všetky mená súborov v adresári, ktoré zodpovedajú špecifikácii, zadanej v 4Eh. Podmienkou hľadania viacerých súborov je, aby sa medzi volaniami nezmenil obsah diskovej vyrovnávacej pamäti. Hľadanie sa vykonáva v rovnakom adresári a na rovnakom zariadení ako pri prvom volaní funkcie 4Eh.

```
Príklad:  ...
mov ah,4Fh
int 21h
jc chyba
...
```

INT 21h, funkcia 56h – Premenovanie súboru. Funkcia umožňuje premenovať súbor v zadanom adresári alebo presunúť adresárovú položku medzi dvoma adresármi na jednom diskovom zariadení. Špecifikácia súboru, ktorý má byť premenovaný alebo presunutý, sa zadáva ako reťazec znakov ASCIIZ. Adresa reťazca sa musí nachádzať v registroch DS:DX. Nová špecifikácia súboru sa tiež odovzdá ako reťazec ASCIIZ. Jeho adresu musí obsahovať dvojica registrov ES:DI. Ak došlo pri premenovávaní ku chybe, je nastavený príznak CF=1 a v registri AX je chybový kód (2 – nenájdený súbor alebo chybná cesta, 3 – cesta nebola nájdená, alebo súbor neexistuje, 5 – nepovolený prístup, 11h – nový súbor nie je na rovnakom zariadení). Ani pôvodné ani nové meno súboru uvedené v špecifikácii nesmie obsahovať znaky (* a ?) hviezdičkovej konvencie. Treba ešte zdôrazniť, že presun sa týka iba položky adresára. Údaje uložené v súbore, zostávajú na rovnakom mieste. Preto nie je možné použiť túto funkciu na presun súborov medzi dvoma zariadeniami.

```
Príklad:  ...
súbor1    db 'C:\vtipy.txt',0
súbor2    db 'C:\humor.txt',0
          ...
          mov ah,56h
          mov dx, seg súbor1
          mov ds,dx
          mov dx,offset súbor1
          mov di, seg súbor2
          mov es,di
          mov Di,offset súbor2
          int 21h
          jc chyba
          ...
```

INT 21h, funkcia 57h – Dátum a čas modifikácie súboru. Služba umožňuje zistiť alebo nastaviť údaje v položke adresára, ktoré sa týkajú dátumu alebo času posledného zápisu do súboru. Údaje je možné zistiť alebo zmeniť iba pre súbory, ktoré sú práve otvorené pomocou kanálov. Funkcii sa odovzdá číslo kanálu otvoreného pre súbor v registri BX. Obsah registra AL rozhoduje o tom či sa budú údaje zisťovať (AL = 0) alebo meniť (AL = 1). Informácie o čase sú vždy v registri CX, informácie o dátume sú v registri DX. Pri vzniku chyby je príznak CF=1 a register AX obsahuje kód chyby (1 – chybné číslo funkcie, 6 – chybné číslo súboru). Systém nevykonáva žiadne kontroly správnosti zadaných údajov. Keď program nastaví nesprávne údaje, môže adresár po uzatvorení súboru obsahovať nezmyselný dátum alebo čas.

```
Príklad:  ...
          mov ah,57h
          mov bx,číslo_súboru
          mov al,0 ;zisťovanie
          int 21h
          jc chyba
          mov dátum,DX
          mov čas,CX
          ...
```

INT 21h, funkcia 5Ah – Vytvorenie dočasného súboru. Služba vytvorí dočasný súbor s takým menom, ktoré nekoliduje s ostatnými menami súborov v adresári. Adresárová cesta k adresáru, v ktorom bude nový súbor vytvorený, sa zadáva ako reťazec ASCIIZ. Jeho adresa musí byť v registroch DS:DX. Register CX musí obsahovať vlastnosti vytváraného dočasného súboru (0 – normálny, 1 – iba pre čítanie, 2 - skrytý, 4 -systémový). Po úspešnom dokončení príznak CF=0, register AX obsahuje číslo kanálu, ktorý bol priradený novému súboru. Registre DS:DX obsahujú adresu ASCIIZ reťazca s kompletným menom súboru. Ak došlo ku chybe, je príznak CF=1 a v registri AX sa nachádza chybový kód (3 – cesta nenájdená, 5 – nepovolený prístup). Dočasný súbor sa pri ukončení práce automaticky nezruší. Volajúci program sa pred ukončením musí postarať o jeho zrušenie napr. službou 41h.

```
Príklad:  ...
file      db 'c:\temp',0
          ...
          mov ah,5Ah
          mov cx,0
          mov dx, seg file
          mov ds,dx
          mov dx, offset file
          int 21h
          jc chyba
          mov číslo_súboru,ax
          ...
```

INT 21h, funkcia 5Bh – Vytvorenie nového súboru. Táto služba sa od služby 3Ch líši iba tým, že súbor, ktorého meno je v ASCIIZ reťazci, nesmie v danom adresári existovať. Pokiaľ existuje, tak sa služba ukončí s chybovým kódom, register AX=50h. Príklad kódu je totožný s funkciou 3Ch.

INT 21h, funkcia 5Ch – Zákaz a povolenie prístupu k súboru. Funkcia umožňuje zakázať alebo povoliť prístup k súboru alebo jeho časti pomocou ostatných kanálov, ktoré sú súboru priradené. Funkciu 5Ch je možné použiť iba v prípade, že je v činnosti systém zdieľania súborov SHARE. Číslo kanálu pridelené súboru, ku ktorého časti má byť zakázaný alebo povolený prístup, sa zadáva do registra BX. Register AL určuje, či bude prístup zakazovaný alebo povoľovaný. Ukazovateľ na relatívnu pozíciu prvého bajtu, ktorého sa operácia týka, sa funkcii odovzdáva v registroch CX a DX. Registre SI a DI musia obsahovať dĺžku oblasti dát v súbore, na ktoré sa zákaz alebo povolenie prístupu bude vzťahovať. Pokiaľ dôjde ku chybe, nastaví sa príznak CF=1 a register AX bude obsahovať chybový kód (1 – chybné číslo funkcie, 6 – chybné číslo súboru, 21h – zamykaná časť je už zamknutá). V registri AL môže byť zadaná iba hodnota 0 alebo 1, iný obsah registra spôsobí chybu. Dvojice registrov CX, DX a SI, DI predstavujú každá 32-bitové celé číslo bez znamienka. Register BX musí obsahovať číslo otvoreného kanálu. Nemá zmysel používať funkciu 5Ch v spojitosti s kanálmi otvorenými pre nesúborové zariadenia.

Zákaz prístupu k súboru alebo jeho časti je možné vykonať iba v prípade, že súbor je otvorený v režimoch zdieľania súborov "prístup povolený" alebo "zákaz čítania" alebo ak je súbor otvorený pre zápis alebo čítanie a zápis v režime zdieľania "zákaz zápisu". Pokus o uzamknutie oblasti údajov súboru, otvorených v iných režimoch zdieľania a prístupu, spôsobí chybu. Ak je oblasť údajov súboru, uzamknutá voči prístupu, potom každý pokus o čítanie alebo zápis údajov do tejto oblasti spôsobí chybu. Chyba pri pokuse o prístup do uzamknutej oblasti sa neohlási hneď. Najprv sa operačný systém pokúsi zopakovať požiadavku niekoľkokrát, vždy po uplynutí určitej doby. Počet pokusov a dĺžka časového úseku sa nastavuje pomocou služby 44h. Táto možnosť sa využíva predovšetkým pri programovaní paralelných procesov.

Pokiaľ program uzamkne časť súboru a duplikuje kanál, ktorý sa na uzamknutie použil, prenáša sa právo prístupu k uzamknutej časti súboru aj na nový kanál. Napriek tomu, ak spustí program podprocesy, právo prístupu k uzamknutým oblastiam súboru sa na ne neprenáša. Program, používajúci uzamykanie súborov musí zabezpečiť, aby nemohlo dôjsť k tomu, že program je ukončený, a pritom zostal otvorený súbor, ktorého časť je uzamknutá voči prístupu. Takáto situácia by mohla mať nepredvídateľné následky.

Povoliť prístup k súboru alebo jeho časti je možné vykonať jedine vtedy keď sa zadá umiestnenie a dĺžka oblasti presne tak, ako keď bola oblasť uzamykaná. Pokus o odomknutie oblasti iným spôsobom, než bola uzamknutá, vedie ku chybe. Ku chybe dôjde aj vtedy, keď sa pokúsime odomknúť časť uzamknutej oblasti alebo pri pokuse odomknúť oblasť, ktorá bola zamknutá ako dve nadväzujúce oblasti. Zákaz prístupu k častiam súboru by sa mal predovšetkým využívať pre koordináciu vstupných a výstupných operácií, paralelne bežiacich procesov.

```
Príklad:  ...
          mov ah,5Ch
          mov al,0
          mov bx,číslo_súboru
          mov cx,0     ;začiatok oblasti
          mov dx,1000h   ; má offset 1000h
          mov si,0
          mov di,2000h ;dĺžka oblasti je 2000h
          int 21h  ;zamkni
          jc chyba
          ...
```

Tradičné funkcie

Pri použití tradičných funkcií pre prácu so súbormi musí pre každý otvorený súbor existovať tzv. riadiaci blok súboru (FCB – File Control Block). Používateľský program musí pri každom tradičnom volaní uviesť ukazovateľ na zodpovedajúci riadiaci blok. Ak je otvorených viac súborov naraz, musí byť ku každému otvorenému súboru priradený vlastný riadiaci blok. Pri otváraní súboru si operačný systém do rezervovaných položiek riadiaceho bloku poznamenáva dôležité informácie o danom súbore. Každý pokus o použitie už otvoreného riadiaceho bloku pre iný súbor vedie k porušeniu týchto informácií. Preto je potrebné pred použitím toho istého riadiaceho bloku pre iný súbor najprv pôvodný súbor uzatvoriť, a až potom zmeniť položky, týkajúce sa špecifikácie súboru. Potom môže byť nový súbor otvorený. Je vhodné uzatvoriť každý súbor ihneď po skončení operácií vstupu a výstupu, ktoré sa ním majú vykonávať. Tým sa obmedzí možnosť vzniku chýb v dôsledku porušenia prístupu k súborom alebo prekročenie najväčšieho možného počtu zároveň otvorených súborov.

Maximálny počet súborov otvorených zároveň pomocou riadiacich blokov súborov závisí od toho, či je inštalovaný v pamäti systém pre zdieľanie súborov (SHARE). Keď nie je zdieľanie súborov v činnosti, nie je obmedzený počet súborov, ktoré môžu byť zároveň otvorené pomocou riadiacich blokov. V opačnom prípade je počet zároveň otvorených súborov obmedzený hodnotou, ktorá je uvedená v konfiguračnom súbore **config.sys** (premenná **FCBS**). Premenná FCBS určuje dve hodnoty – celkový počet súborov, ktoré môžu byť otvorené zároveň pomocou riadiacich blokov, a počet súborov, ktoré sú chránené proti uzatvoreniu. V prípade, že dôjde k prekročeniu maximálneho počtu zároveň otvorených súborov, DOS automaticky uzatvorí najdlhšie otvorený súbor.

Každý zápis alebo čítanie pomocou tradičných funkcií číta alebo zapisuje do súboru **jeden záznam**. Veľkosť záznamu môže používateľský program nastavovať zmenou príslušnej položky v riadiacom bloku. Implicitne je pri otvorení súboru nastavená veľkosť záznamu na 128 bajtov (128 bajtov tvorí jeden blok). V riadiacom bloku súboru sa udržujú ukazovatele aktuálneho bloku a aktuálneho záznamu. Pomocou nich sa systém orientuje v súbore pri sekvenčných zápisoch a čítaniach. Operácie s priamym prístupom sa riadia inou položkou riadiaceho bloku súboru – ukazovateľom relatívneho čísla záznamu od začiatku súboru. Všetky ukazovatele môže nastavovať používateľský program, môžu sa však nastavovať aj niektorým funkčným volaním. Napríklad, pri priamom zápise zo súboru sú na zodpovedajúcu hodnotu nastavené aj ukazovatele aktuálneho záznamu a aktuálneho bloku.

Okrem riadiaceho bloku súboru potrebujú tradičné funkcie k svojej práci diskovú vyrovnávaciu pamäť. Disková vyrovnávacia pamäť (DTA – Disk Transfer Area) musí byť vyhradená vo vnútornej pamäti buď systémom alebo používateľským programom. Všetky presuny údajov pomocou tradičných funkcií sa vykonávajú cez diskovú vyrovnávaciu pamäť.

Riadiaci blok súboru (FCB)

Riadiaci blok obsahuje všetky dôležité údaje o danom súbore. Rozlišujeme **štandardný riadiaci blok**, alebo **rozšírený riadiaci blok súboru**. Štandardný riadiaci blok obsahuje položky podľa tabuľky 1.

Adresa	Dĺžka	Význam
00h	1 B	Číslo jednotky
01h	8 B	Meno súboru alebo rezervované meno
09h	3 B	Prípona mena súboru
0Ch	1 W	Relatívne číslo bloku v súbore
0Eh	1 W	Veľkosť záznamu
10h	2 W	Veľkosť súboru v bajtoch
14h	1 W	Dátum posledného zápisu do súboru
16h	1 W	Čas posledného zápisu do súboru
18h	8 B	Rezervované pre DOS
20h	1 B	Relatívne číslo záznamu v bloku
21h	2 W	Relatívne číslo záznamu od začiatku súboru

Štandardný riadiaci blok súboru (B – byte, W - word)

Číslo jednotky určuje diskové zariadenie, na ktorom sa súbor bude nachádzať. Pred otvorením súboru má číslo jednotky tento význam: **0** – aktuálna disková jednotka, **1** – disková jednotka A, **2** – disková jednotka B, atď. Pri otvorení je číslo jednotky 0 nahradené číslom aktuálnej diskovej jednotky. Ostatné čísla jednotiek zostavajú rovnaké ako pred otvorením.

Meno súboru je dlhé 8 znakov (ASCII). Ak je meno kratšie, je sprava doplnené znakmi medzera (kód 20h) na 8 znakov. Položka môže obsahovať namiesto mena súboru rezervované meno zariadenia. Potom musí byť meno zariadenia uvedené bez dvojbodky (napr. LPT1) a doplnené sprava medzerami.

Prípona mena súboru je dlhá 3 znaky (ASCII). Ak je prípona kratšia, je sprava doplnená na 3 znaky medzerami (kód 20h).

Relatívne číslo bloku v súbore od začiatku súboru predstavuje ukazovateľ na aktuálny blok, v ktorom budú vykonávané operácie vstupu a výstupu. Spolu s relatívnym číslom záznamu v bloku a veľkosťou záznamu má relatívne číslo bloku význam pri sekvenčnom čítaní a zápise. Každý blok súboru obsahuje vždy 128 záznamov. Bloky sú v súbore číslované od 0. Pri otvorení súboru je relatívne číslo bloku nastavené implicitne na 0.

Veľkosť záznamu udáva dĺžku jedného záznamu súboru v bajtoch. Pri otvorení súboru je implicitne nastavená na 128. Ak požaduje používateľský program, inú veľkosť záznamu je treba po otvorení súboru nastaviť priamo v riadiacom bloku. Veľkosť záznamu používa operačný systém pri výpočte všetkých skutočných adries pre čítanie a zápis na disk.

Veľkosť súboru v bajtoch určuje skutočnú veľkosť súboru. Nemusí byť vždy rovnaká ako veľkosť súboru vypočítaná pomocou počtu blokov a veľkosti záznamov. Posledný blok súboru nemusí byť zaplnený celý. Veľkosť súboru je uložená v štyroch bajtoch.

Dátum posledného zápisu do súboru je uložený v dvoch bajtoch jedného slova ako tri celé binárne čísla bez znamienka.

Adresa 15H							Adresa 14H								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Význam
0	**0**	**0**	**0**	**0**	**0**	**0**	.	.	.	.	.	.	.	.	.	Rok (0 – 119)
.	.	.	.	.	.	.	**0**	**0**	**0**	**0**	.	.	.	.	.	Mesiac (1 – 12)
.	.	.	.	.	.	.	.	.	.	.	**0**	**0**	**0**	**0**	**0**	Deň (1 – 31)

Dátum poslednej zmeny

Rok je v 7-bitovej oblasti uložený ako číslo od 0 do 199 dekadicky. Skutočný rok sa získa pričítaním dekadického čísla 1980. Tak je pokryté rozmedzie rokov 1980 až 2099. Položka určuje dátum, kedy boli údaje v súbore naposledy zmenené, prípadne, kedy bol súbor vytvorený.

Čas posledného zápisu do súboru je uložený ako tri celé binárne čísla bez znamienka v dvoch bajtoch slova.

Adresa 17H					Adresa 16H										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Význam
0	**0**	**0**	**0**	**0**	.	.	.	.	.	.	.	.	.	.	.	Hodina (0 – 23)
.	.	.	.	.	**0**	**0**	**0**	**0**	**0**	**0**	.	.	.	.	.	Minúta (0 – 59)
.	.	.	.	.	.	.	.	.	.	.	**0**	**0**	**0**	**0**	**0**	Sekunda / 2 (0 – 29)

Čas poslednej zmeny

Päťbitová oblasť vyhradená sekundám sa zvyšuje o 1 vždy po dvoch sekundách. Sekundu poslednej zmeny je možné zistiť vynásobením tohoto údaju dvoma.

Pre **DOS je rezervovaných** 8 bajtov, do ktorých operačný systém zaznamenáva vlastné údaje o súbore, pre ktoré je riadiaci blok použitý. Tieto údaje sú používané pri riadení operácií zo súborom a používateľský program ich nesmie meniť.

Relatívne číslo záznamu v bloku je zaznamenané v jednom bajte a pohybuje sa v rozmedzí 0 až 127. Predstavuje ukazovateľ na aktuálnom zázname v aktuálnom bloku. kde budú vykonávané operácie sekvenčného vstupu alebo výstupu. Pri otvorení súboru nie je ukazovateľ nastavovaný. Pred začatím sekvenčného čítania alebo zápisu musí byť nastavený používateľským programom na príslušnú hodnotu.

Relatívne číslo záznamu od začiatku súboru predstavuje ukazovateľ na záznam, ktorého sa bude týkať operácia priameho čítania alebo zápisu do súboru. Spolu s veľkosťou záznamu určuje, na ktorom mieste súboru budú operácie s priamym prístupom vykonávané. Záznamy sú číslované v súbore od 0. Ak je veľkosť záznamu menšia než 64 bajtov, používajú sa obidve slová. Ak je veľkosť záznamu väčšia než 64 bajtov, používajú sa prvé tri bajty. Nižšie rády sú zaznamenané v prvom slove. Používateľský program musí tento ukazovateľ nastaviť pred každým vykonaním operácie s priamym prístupom. Pri otvorení súboru nie je ukazovateľ nastavovaný.

Vždy pred otvorením súboru musí byť pripravené miesto pre riadiaci blok a musia v ňom byť vyplnené položky: číslo jednotky, meno súboru a prípona súboru. Takto pripravený riadiaci blok sa nazýva **neotvorený riadiaci blok súboru**. Neotvorený riadiaci blok môže byť zostavený priamo buď používateľským programom alebo funkciou 29h (Spracovanie mena súboru). Po otvorení súboru funkciou 0Fh operačný systém vypĺňa ostatné položky riadiaceho bloku, okrem relatívneho čísla záznamu v bloku a relatívneho čísla záznamu od začiatku súboru. Takýto riadiaci blok sa nazýva **otvorený riadiaci blok súboru**. Používateľský program môže v priebehu spracovania súboru meniť ukazovatele bloku aj záznamu, ukazovateľ záznamu od začiatku súboru a dĺžku záznamu. Zmeniť sa nesmú údaje o dátume a čase posledného záznamu, veľkosť súboru údaje o špecifikácii súboru a údaje o oblasti rezervovanej pre DOS.

Rozšírený riadiaci blok súboru

Rozšírený riadiaci blok súboru sa používa pri vyhľadávaní alebo vytváraní súborov, ktoré majú zvláštne vlastnosti, napríklad systémové alebo skryté súbory. Rozšírený riadiaci blok obsahuje všetky položky štandardného riadiaceho bloku a okrem toho, ešte 7 bajtov tzv. rozšírenie. Rozšírenie je umiestnené pre prvú položku štandardného bloku:

Relatívna adresa	Dĺžka	Význam
00h	1 B	Príznak rozšíreného riadiaceho bloku
01h	5 B	Rezervované pre DOS
06h	1 B	Vlastnosti súboru
07h	37 B	Položky štandardného riadiaceho bloku

Rozšírený riadiaci blok súboru (B - byte)

Príznak rozšíreného riadiaceho bloku určuje, či ide o rozšírený riadiaci blok. Ak je hodnota nastavená na FFh, operačný systém považuje prvých 7 bajtov za rozšírenie riadiaceho bloku. Ak je nastavená iná hodnota, je bajt považovaný za prvý bajt štandardného riadiaceho bloku.

Rezervované pre DOS – pre DOS je rezervovaných 5 bajtov, ktorých obsah je vždy nulový a nesmie byť používateľským programom zmenený.

Vlastnosti súboru (atribúty) sú uložené ako 8 bitov v jednom bajte. Každý bit reprezentuje jednu vlastnosť súboru alebo zvláštny význam položky adresára.

Bit 7	6	5	4	3	2	1	0	Význam
.	.	.	.	.	.	.	**0**	Súbor iba pre čítanie (Read-only)
.	.	.	.	.	.	**0**	.	Skrytý súbor (Hidden)
.	.	.	.	.	**0**	.	.	Systémový súbor (System)
.	.	.	.	**0**	.	.	.	Položka návestia (Volume label)
.	.	.	**0**	.	.	.	.	Podadresár (Subdirectory)
.	.	**0**	.	.	.	.	.	Archivačný príznak (Archive)
0	0	.	.	.	.	.	.	Rezervované

Atribúty súboru

Pri použití rozšíreného riadiaceho bloku musí byť ukazovateľ pri volaní funkcie nastavený na začiatok rozšírenia, a nie na číslo jednotky. Vo väčšine funkčných volaní sa môže používať štandardný, i rozšírené riadiace bloky. Neotvorený rozšírený riadiaci blok musí mať okrem údajov o mene, prípone a jednotke, vyplnenú tiež položku vlastností súboru.

Disková vyrovnávacia pamäť (DTA)

Disková vyrovnávacia pamäť (DTA – Disk Transfer Area) je oblasť pamäti, ktorá sa využíva pri čítaní a zápise pomocou tradičných funkcií. Disková vyrovnávacia pamäť môže byť umiestnená na ľubovoľnom mieste v používateľskom programe. Jej veľkosť a umiestnenie môže používateľ meniť. Pri každom spustení používateľského programu je nastavená implicitná disková vyrovnávacia pamäť na relatívnej adrese 128 v predpone programového segmentu (PSP – Program Segment Prefix). Jej dĺžka je 128 bajtov a používa sa pre všetky operácie čítania a zápisu pomocou tradičných funkcií, pokiaľ používateľský program jej umiestnenie nezmení. Po vykonaní zmeny až do ďalšej zmeny DOS používa novú vyrovnávaciu pamäť. Pri ukončení práce používateľského program sa disková vyrovnávacia pamäť nastaví na pôvodné implicitné hodnoty.

Literatúra

[1] Ralf Brown: Interrupt List - Release 33. Pittsburgh PA, U.S.A. 1993
[2] Vlastislav Černý: MS-DOS 6.22. Kopp 1996
[3] P. Heřman: Príručka systémového programátora. Tesla Eltos 1990
[4] David Jurgens: HelpPC 2.10
[5] ABSHelp 2.05, ABSoft Olomouc
[6] Michal Brandejs: MS-DOS 6, kompletní průvodce. Grada 1993

Dvadsiatadruhá časť:
Dosovské ovládače

Každý z vás sa už určite stretol s pojmom driver (ovládač). Prvým takým ovládačom, s ktorým môžete prísť do styku (napríklad pri optimalizácii súborov config.sys, prípadne autoexec.bat), je HIMEM.SYS[1]. V tejto časti si teda povieme niečo o ovládačoch používaných v DOS-e, hlavne o ich štruktúre a tvorbe.

Zariadenia pripojené k centrálnej jednotke počítača môžeme ovládať niekoľkými spôsobmi. Priamou komunikáciou používateľského programu cez porty zariadení, pomocou prerušenia BIOS alebo využitím INT 21H. Priama komunikácia sa používa pre špeciálne zariadenia, ktoré nie je možné ovládať systémovými prostriedkami. Spôsob priamej komunikácie nie je operačným systémom obmedzený. Pokiaľ sa však majú na ovládanie zariadení použiť systémové prostriedky (INT 21H), je potrebné pri programovaní obsluhy zariadení dodržiavať určité pravidlá. Tie sú dané koncepciou operačného systému a sú použité pri ovládaní štandardných periférnych zariadení (klávesnica, obrazovka, tlačiareň a pod.). Pre každé zariadenie, ktoré má byť ovládané štandardným spôsobom, sa vytvára špeciálny obslužný program – driver (ovládač).

Predstavme si napríklad používateľský program, ktorý požaduje operáciu čítania – vyvolá prerušenie INT 21h. To aktivuje kód pre jeho spracovanie v module DOS (pozri obr. 1). Ten vykoná potrebné kontroly (napr. či je súbor otvorený a pod.). Potom vytvorí požiadavku na vykonanie operácie čítania. Táto požiadavka má štandardný tvar a odovzdá sa príslušnému ovládaču zariadení. Ovládač potom podľa obsahu požiadavky vykoná vlastnú operáciu (napr. čítanie). Pretože sa príslušné operácie vykonávajú pre jednotlivé zariadenia odlišným spôsobom, musí mať každé zariadenie svoj ovládač. Ovládače štandardných systémových zariadení (klávesnica, disk, obrazovka...) sú umiestnené v systémovom module BIOS. Pre ostatné prídavné zariadenia je potrebné napísať ovládače ako samostatné programy. Ovládače komunikujú buď priamo, alebo prostredníctvom podprogramov, ktoré sú súčasťou technického vybavenia počítača (BIOS).

Okrem ovládačov pre nové zariadenia je možné vytvárať aj ovládače pre štandardné zariadenia. Operačný systém umožňuje nahradenie ovládačov používateľskými ovládačmi.

Typy zariadení

Spôsob návrhu ovládača súvisí s typom zariadenia, pre ktoré je určený. Z hľadiska funkcie môžeme zariadenia rozdeliť do dvoch základných skupín:

- znakové zariadenia (sekvenčný prístup),
- blokové zariadenia (priamy prístup).

Znakové zariadenia spracúvajú dáta postupne v sekvenčnom poradí, najčastejšie znak po znaku. Zo systémových zariadení to je napríklad klávesnica a obrazovka (CON), tlačiareň (PRN a LPT) a komunikačné adaptéry (AUX a COM). Do tejto skupiny patria aj niektoré zariadenia, ktoré nepracujú priamo so znakmi, napríklad hodiny reálneho času. Je to dané spôsobom ich ovládania, ktoré je z hľadiska systému zhodné s ovládaním skutočných znakových zariadení. Operačný systém požaduje pre každé znakové zariadenie jeden ovládač a jedinečné meno, t. j. na dve rôzne zariadenia nemôžete odkazovať tým istým menom. Možno namietnete, prečo klávesnica a obrazovka sú označené CON. Je to z toho dôvodu, že sa berú ako jedno zariadenie.

Blokové zariadenia umožňujú operačnému systému spracúvať dáta, ktoré sú na nich uložené v ľubovoľnom poradí. Dáta sa čítajú zo zariadenia alebo sa na zariadenie zapisujú po tzv. blokoch, čo sú obyčajne fyzické sektory. Priamy prístup k ľubovoľnému bloku zariadenia dovoľuje vytváranie adresárov a zložitejších dátových štruktúr. V operačnom systéme sú k dispozícii ovládače na prácu s diskami a disketami, prípadne ovládač na vytvorenie a ovládanie virtuálneho disku.

Na rozdiel od znakových zariadení môže ovládač blokového zariadenia ovládať niekoľko jeho jednotiek. Jednotlivým jednotkám všetkých blokových zariadení, ktoré sú prítomné v systéme, operačný systém automaticky prideľuje mená. Toto sa vykonáva pri zavádzaní operačného systému a ako mená sa používajú veľké písmená A až Z. Najprv sa priradia mená disketovým jednotkám a jednotkám pevných diskov a potom sa postupne priraďujú ďalšie mená pre jednotlivé jednotky blokových zariadení, ktoré boli zavedené do pamäte príkazmi DEVICE. Tieto príkazy sa vkladajú do konfiguračného súboru **config.sys**. Mená sa prídavným zariadeniam prideľujú v poradí, v akom sú uvedené príslušné príkazy DEVICE v konfiguračnom súbore config.sys. V niektorých prípadoch je možné jednej jednotke priradiť aj viacej mien (ovládač DRIVER.SYS).

Tvar a činnosť ovládača

Kód ovládača sa zapisuje obdobným spôsobom ako kód ľubovoľného používateľského programu. Určité obmedzenia sú dané požiadavkami operačného systému na štruktúru ovládača.

Zakladaným tvarom ovládača je program preložený a zostavený do tvaru *.EXE. Pokiaľ tento program spĺňa požadované podmienky, možno ho previesť príkazom EXE2BIN do súboru, ktorý bude obsahovať obraz uloženia programu v pamäti. V tomto tvare sa dodávajú všetky systémové ovládače pre DOS. Výhody: rýchlejšie zavádzanie ovládačov do pamäte, ušetrí sa miesto na systémovom disku. Oproti programovaniu programov v tvare *.COM je tu jedna výnimka. Pretože pri zavádzaní ovládača do pamäte sa nevytvára

predpona programového segmentu PSP (Program Segment Prefix), nesmie byť na začiatku programu použitá direktíva ORG 100H. Kód ovládača sa musí v súbore začínať vždy od adresy 0.

V jednom súbore môže byť obsiahnutý kód jedného alebo niekoľkých nezávislých ovládačov. K zápisu niekoľkých ovládačov do jedného súboru sa pristupuje napr. v prípade, že viacej ovládačov používa rovnaké podprogramy.

Každý ovládač musí obsahovať nasledujúce tri časti:
- záhlavie,
- riadiaci podprogram,
- vykonávací podprogram.

Základný mechanizmus spolupráce systémového modulu DOS s ovládačom je nasledujúci:
- modul DOS vytvorí požiadavku na vykonanie nejakej operácie a adresu tejto požiadavky odovzdá riadiacemu podprogramu ovládača; adresa musí byť uložená v registroch ES:BX
- riadiaci podprogram zaradí požiadavku do frontu požiadaviek alebo iba uchová adresu požiadavky a vráti riadenie modulu DOS,
- modul DOS hneď potom volá vykonávací podprogram ovládača; pritom už neodovzdáva nijaké parametre; úlohou vykonávacieho podprogramu je vykonanie požadovanej operácie a nastavenie prípadných výstupných parametrov pred návratom späť do modulu DOS.

Pri vstupe do kódu ovládača je potrebné uschovať obsahy všetkých registrov, ktoré budú v ovládači použité. Pred návratom do systému treba hodnoty registrov opäť obnoviť. Zásobník používaný modulom DOS má dostatočný priestor na uchovanie všetkých registrov. Pokiaľ však ovládač vyžaduje väčší zásobník, musí si ho vytvoriť aj ovládať sám.

Záhlavie ovládača

Operačný systém vyžaduje, aby každý ovládač mal na svojom začiatku definovanú špeciálnu dátovú oblasť – záhlavie ovládača (pozri tab. 1). Hodnoty uvedené v tejto oblasti využíva operačný systém pri komunikácii s príslušným ovládačom. Záhlavie prvého ovládača v súbore musí byť uvedené na začiatku prvého programového segmentu. Vo výslednom programe, t. j. po preložení TASM, zostavení TLINK a po prevode príkazom EXE2BIN, musí byť uložené od adresy 0. Záhlavie ovládača, riadiace i vykonávacie podprogramy musia byť zapísané v jednom programovom segmente. Z toho vyplýva, že pri zápise ovládača je potrebné ako prvý zapisovať programový segment, na začiatku ktorého je definované záhlavie. Pokiaľ je v jednom súbore obsiahnutých viacej ovládačov, budú ich záhlavia zreťazené v rámci súboru. Na umiestnení zreťazených záhlaví (okrem prvého) potom už nezáleží.

Relatívna adresa	Dĺžka v bajtoch	Význam
0	4	Ukazovateľ na nasledujúci ovládač v reťazci ovládačov
4	2	Vlastnosti ovládača
6	2	Adresa riadiaceho podprogramu
8	2	Adresa vykonávacieho podprogramu
10	8	Meno ovládača / počet jednotiek

Záhlavie ovládača

V posledných 8 bajtov záhlavia obsahuje: ak ide o znakové zariadenie – meno ovládača, ak ide o blokové zariadenie – počet jednotiek, ktoré ovládač obhospodaruje. Táto hodnota je uložená v prvom bajte, ostatných 7 bajtov je nevyužitých, ale treba ich v záhlaví vyhradiť.

Ukazovateľ na nasledujúci ovládač

V tejto položke očakáva operačný systém adresu ďalšieho ovládača vo fronte ovládačov zavedených do operačnej pamäte. Adresa smeruje na záhlavie nasledujúceho ovládača. Do prvých dvoch bajtov sa ukladá relatívna adresa záhlavia ďalšieho ovládača v segmente, do ďalších dvoch bajtov sa ukladá jeho bázová adresa. Obsadenie ukazovateľa závisí od obsahu súboru. Pokiaľ súbor obsahuje iba jeden ovládač, musí byť hodnota ukazovateľa nastavená na –1 (0FFFFh).

Skutočnú adresu ďalšieho ovládača vo fronte ovládačov do tejto položky uloží operačný systém pri zavádzaní ovládača do pamäte. V prípade, ak súbor obsahuje niekoľko ovládačov, musia byť v rámci súboru zreťazené, pričom na –1 (0FFFFh) sa nastaví hodnota ukazovateľa v záhlaví posledného ovládača. Príklad zreťazenia viacerých ovládačov:

```
;začiatok súboru
ovl1        dd ovl2
            ...
ovl2        dd ovl3
            ...
ovl3        dd –1
```

Vlastnosti ovládača

Táto položka slúži na určenie typu a základných vlastností ovládača. Podľa jej obsahu rozhoduje operačný systém o spôsobe spracovania požiadaviek na periférne operácie. Jednotlivé bity majú nasledujúce významy:
- **bit 15** – od hodnoty tohto bitu závisí spôsob komunikácie operačného systému s ovládačom a spôsob použitia ovládača. Bit 15 = 1 – znakovo orientované zariadenie, bit 15 = 0 – blokovo orientované zariadenie,
- **bit 14** – tento bit informuje o tom, či môže ovládač spracúvať sekvencie riadiacich znakov pomocou funkcie 44h. To umožňuje posielanie údajov na zariadenie alebo čítanie údajov bez toho, aby sa vykonal skutočný vstup alebo výstup. Údaje môžu byť použité pre vlastnú potrebu zariadenia, napr.: na definovanie prenosovej rýchlosti a pod. Interpretácia údajov závisí od zariadení; ale v žiadnom prípade sa nesmú údaje chápať ako požiadavky na štandardný vstup a výstup. Bit 14 = 1 – ovládač podporuje spracovanie riadiacich znakov, bit 14 = 0 – ovládač nepodporuje spracovanie riadiacich znakov,
- **bit 13** (pozri tab. 2) – pokiaľ má blokové zariadenie formát IBM (adresáre, bloky atď.), stačí na určenie typu média identifikačný bajt média; v prípade, že zariadenie nemá formát IBM, musí sa na určenie typu média použiť blok BPB. Hlásenie nepripravenosti je používané ovládačmi tlačiarní. Pokiaľ ovládač túto vlastnosť používa a zariadenie nie je v čase odovzdávania znaku pripravené, ohlási sa okamžite chyba,

Stav	Blokové zariadenia		Znakové zariadenia	Stav
0	má štan. formát IBM	**bit 13**	neovláda hlásenia nepripravenosti	0
1	nemá štan. formát IBM		ovláda hlásenia nepripravenosti	1

Možné stavy bitu 13 pre blokové a znakové zariadenia

- **bit 12 = 0** – rezervované,
- **bit 11** – v prípade, že je bit 11 nastavený na 1, obsahuje ovládač kód na spracovanie príkazov číslo 13 (otvorenie), 14 (zatvorenie) a 15 (zistenie výmenného média). Bit 11 = 1 – obsluhuje výmenné média, Bit 11 = 0 – neobsluhuje výmenné média,
- **bity 10 až 7** sú rezervované, ich stav je nulový,
- **bit 6 = 1** – ovláda nastavenie logického zariadenia, bit 6 = 0 – je presným opakom predchádzajúceho stavu,
- **bity 5 a 4** sú rezervované, ich stav je nulový,
- **bit 3** – označenie ovládača, ktorý vykonáva spracovanie času (implicitne je to ovládač CLOCK$3), bit 3 = 1 – zariadenie typu hodiny, bit 3 = 0 – nie je to zariadenie typu hodiny;
- **bit 2** – tento bit má význam pre operačný systém, ktorý podľa neho zisťuje, či sa používa ovládač NUL alebo nie. Bit 2 =1 – aktuálne zariadenie NUL, bit 2 =0 – nie je to aktuálne zariadenie NUL;
- **bit 1 = 1** – aktuálne štandardné vstupné zariadenie, bit 1 = 0 – nie je to aktuálne štandardné vstupné zariadenie;
- **bit 0** – Implicitne je štandardné vstupné a výstupné zariadenie CON. Nastavením bitov 0 a 1 je možné štandardné vstupné a výstupné zariadenie zmeniť.

Stav	Blokové zariadenia		Znakové zariadenia	Stav
0	neovláda spracovanie generovanej sekvencie riadiacich znakov	**bit 0**	nie je to aktuálne štandardné vstupné zariadenie	0
1	ovláda spracovanie generovanej sekvencie riadiacich znakov (prikaz č. 19)		aktuálne štandardné vstupné zariadenie	1

Možné stavy bitu 0 pre blokové a znakové zariadenia

Adresy riadiaceho a vykonávacieho podprogramu

V týchto položkách sú uložené relatívne adresy vstupu do riadiaceho, resp. vykonávacieho podprogramu. Pretože sú tu uložené iba relatívne adresy bez bázových adries segmentov, musí byť riadiaci aj vykonávací podprogram zapísaný v rovnakom segmente ako záhlavie ovládača.

Meno ovládača / počet jednotiek

Pre znakové zariadenie je tu uvedené jeho meno. Toto meno môže mať maximálne 8 znakov; pokiaľ bude kratšie, musí sa doplniť medzerami na dĺžku 8 znakov. Pre blokové zariadenie je v tomto bajte uložený počet jednotiek. Táto hodnota sa dáva do záhlavia ovládača voliteľne, pretože systém túto hodnotu plní pri zavádzaní ovládača hodnotou, ktorá je vrátená inicializačným príkazom.

Riadiaci podprogram

Riadiaci program je volaný modulom DOS na vytvorenie požiadaviek na nejakú operáciu. Volanie tohto podprogramu sa robí inštrukciou CALL typu FAR, pričom register ES obsahuje bázovú a register BX relatívnu adresu požiadavky.

V riadiacom podprograme je možné vytvárať front požiadaviek na dané zariadenie alebo spracovať hodnoty odovzdané v požiadavke. Manipulácia s požiadavkou závisí od typu ovládaného zariadenia a spôsobu jeho ovládania. Vo väčšine prípadov sa iba uloží adresa požiadavky. Tá sa potom použije vo vykonávacej časti. Kód riadiaceho podprogramu musí byť umiestnený v rovnakom programovom segmente ako záhlavie ovládača.

Vykonávací podprogram

Tento podprogram obsahuje kód, ktorý zabezpečí vykonanie operácie určenej požiadavkou, ktorej adresa bola odovzdaná riadiacemu podprogramu. Modul DOS ho

volá bezprostredne po návrate z riadiaceho podprogramu. Na volanie opäť používa inštrukciu CALL typ FAR. Podprogram preto takisto musí byť typu FAR, a to z toho dôvodu, aby prekladač správne vygeneroval návratovú inštrukciu RET.

Na začiatku vykonávacieho podprogramu treba uschovať obsahy všetkých registrov používaných v ďalšom kóde a obnoviť ich tesne pred návratom do modulu DOS. Priebeh vykonávania požadovanej operácie sa riadi obsahom požiadavky alebo hodnotami pripravenými riadiacim podprogramom. Vykonávací podprogram zvyčajne zistí kód požadovaného príkazu a odovzdá riadenie podprogramu, ktorý operáciu určenú týmto príkazom vykoná.

Po vykonaní operácie je potrebné správne nastaviť všetky požadované výstupné hodnoty a obsah stavového slova. Podľa stavového slova zisťuje operačný systém výsledok vykonanej operácie a prípadné chyby, ktoré ovládač zistil.

Literatúra

[1] Ralf Brown: Interrupt List - Release 33. Pittsburgh PA, U.S.A. 1993.
[2] Vlastislav Černý: MS-DOS 6.22. Kopp 1996.
[3] P. Heřman: Príručka systémového programátora. Tesla Eltos 1990.
[4] David Jurgens: HelpPC 2.10.
[5] ABSHelp 2.05, ABSoft Olomouc.
[6] Michal Brandejs: MS-DOS 6, kompletní průvodce. Grada 1993.

Vysvetlivky

1 HIMEM.SYS – Je hlavným správcom dodatočnej pamäte, t. j. pamäte nad 640 KB. Dozerá na to, aby rôzne programy nepoužívali rovnaké bloky pamäte HMA.

Dvadsiatatretia časť:
Dosovské ovládače (2)

Inštalácia ovládačov

Operačný systém inštaluje prídavné ovládače pri svojom štarte. Požiadavka na inštaláciu sa vykonáva vložením príkazov DEVICE do konfiguračného súboru CONFIG.SYS.

Pri inštalácii sa zavedie kód ovládača do operačnej pamäte a bezprostredne potom sa vykoná jeho inicializácia. Pokiaľ prebehne inicializácia bez chýb, zreťazí sa zavedený kód ovládača do frontu už predtým inštalovaných ovládačov. Ovládače sa inštalujú v poradí, v ktorom sú spracúvané jednotlivé príkazy DEVICE. Po zavedení všetkých požadovaných ovládačov je ako prvý vo fronte ovládačov uložený naposledy inštalovaný ovládač a na konci frontu sú umiestnené systémové ovládače (CON, PRN atď.). Blokovým zariadeniam sa pri inštalácii ešte priraďujú mená jednotiek.

Na blokové zariadenia sa operačný systém obracia pomocou mien jednotiek, ktoré im boli priradené pri inštalácii. Na znakové zariadenia sa systém obracia prostredníctvom ich mien (CON, PRN atď.). Pri vykonávaní operácie so znakovým zariadením prehľadáva systém front ovládačov a postupne porovnáva meno požadovaného zariadenia s menami určenými v záhlaviach jednotlivých ovládačov. Na vykonanie operácie sa použije prvý ovládač z frontu, ktorý má požadované meno. Z toho vyplýva, že prídavnými používateľskými ovládačmi je možné nahradiť ovládače systémové. To sa vykoná napríklad vtedy, keď si nainštalujete ovládač ANSI.SYS. Tento ovládač po nainštalovaní nahradí systémový ovládač CON.

Tvar požiadaviek pre ovládač

Na komunikáciu modulu DOS s jednotlivými ovládačmi sa používa dátová štruktúra, ktorá má predpísaný tvar. Túto štruktúru budeme v ďalšom texte nazývať požiadavka. Skladá sa z dvoch častí:

1. z hlavičky (tab. 4),
2. z dát potrebných pre požadovanú operáciu.

Každá požiadavka musí obsahovať hlavičku. Prítomnosť dát a ich tvar závisia od vykonávanej operácie. Hlavička požiadavky je dlhá 13 bajtov.

Relatívna adresa	Dĺžka v bajtoch	Význam
0	1	Dĺžka celej požiadavky, t.j. dĺžka hlavičky + dĺžka potrebných dát
1	1	Číslo jednotky. Používa sa iba pre blokové zariadenia
2	1	Kód príkazu
3	2	Stavové slovo
5	8	Priestor vyhradený pre potreby operačného systému

Hlavička požiadavky

Číslo jednotky označuje jednotku v rámci zariadenia ovládaného daným ovládačom. Ak má napríklad zariadenie dve jednotky, sú pre ne používané čísla 0 a 1. To znamená, že pokiaľ sú pre dané zariadenia pridelené mená jednotiek E a F, bude pri odkaze na jednotku E generované číslo 0, pri odkaze na F číslo 1.

Kód príkazu. Do tejto položky sa ukladá kód určujúci požadovanú operáciu. Príkazy budú opísané ďalej. Teraz si uvedieme iba prehľad ich kódov (pozri tab. 5). Písmeno **B** (v stĺpci Použitie) označuje blokové zariadenie, písmeno **Z** označuje znakové zariadenie.

Kód príkazu	Použitie	Význam
0	BZ	Inicializácia ovládača (INIT)
1	B	Kontrola média
2	B	Odovzdanie bloku parametrov pre BIOS
3	BZ	Vstup riadiacich znakov
4	BZ	Čítanie
5	Z	Nedeštruktívne čítanie bez čakania
6	Z	Stav vstupu
7	Z	Zanedbanie vstupných dát
8	BZ	Zápis
9	BZ	Zápis s kontrolou
10	Z	Stav výstupu
11	Z	Zanedbanie výstupných dát
12	BZ	Výstup riadiacich znakov
13	BZ	Otvorenie súboru
14	BZ	Zatvorenie súboru
15	B	Výmenné médium
19	BZ	Požiadavka na spracovanie generovanej sekvencie riadiacich znakov
23	BZ	Zistenie logického zariadenia
24	BZ	Nastavenie logického zariadenia

Tabuľka 2

Stavové slovo. Prostredníctvom stavového slova informuje ovládač operačný systém o výsledku vykonanej operácie. Jeho obsah sa nastavuje tesne pred návratom z vykonávacieho podprogramu. Pri volaní ovládača má stavové slovo nulový obsah.

Význam jednotlivých bitov stavového slova uvádzame v tabuľke:

Bit	Význam
15	Chybový bit (ERROR). Nastavenie tohto bitu na 1 indikuje, že došlo k chybe. Kód chyby je uložený v bitoch 0 – 7.
14 -10	Rezervované bity.
9	Bit zaneprázdnenia (BUSY). Tento bit nastavujú príkazy 5, 6, 10 a 15.
8	Koniec operácie (DONE). Nastavením na 1 sa oznamuje operačnému systému, že operácia bola ukončená.
7 - 0	Číslo chyby. Tieto bity sú platné iba v prípade, že je bit 15 nastavený na hodnotu 1.

Tabuľka 3: Bity stavového slova

Na označenie chýb zistených ovládačom používa operačný systém nasledujúce čísla (tab. 4):

Kód chyby	Význam
0	Zákaz zápisu na médium
1	Neznáma jednotka
2	Zariadenie nie je pripravené
3	Neznámy príkaz
4	Chyba cyklického kontrolného súčtu (CRC)
5	Chybná dĺžka požiadavky
6	Chyba nastavenia na stopu (SEEK)
7	Neznáme médium
8	Sektor nebol nájdený
9	Koniec papiera v tlačiarni
10	Chyba zápisu
11	Chyba čítania
12	Iná chyba
13	Rezervované
14	Rezervované
15	Nesprávna výmena disku

Tabuľka 4: Chyby zistené ovládačom

Pokiaľ si budete vytvárať vlastný ovládač, treba dodržať uvedené čísla chýb, pretože podľa nich vykonáva systém ďalšiu činnosť.

Príkazy ovládača

Operačný systém komunikuje s ovládačmi pomocou príkazov. Tieto príkazy sa ovládačom odovzdávajú vo forme požiadaviek. Ovládač môže obsahovať kódy na spracovanie všetkých príkazov (tab. 5). Vo väčšine prípadov však bude podporovať iba niektoré príkazy. Je napríklad nezmyselné zisťovať v ovládači klávesnice, či ide o výmenné médium, atď. Pre príkazy, ktoré ovládač nepodporuje, musí ohlásiť ukončenie operácie (bit 8 stavového slova), prípadne vrátiť chybu číslo 3 (tab. 4).

Nasleduje opis jednotlivých príkazov, tak ako sú uvedené v tabuľke 5. V opise každého príkazu bude uvedený formát požiadavky, v ktorej sú uvedené hodnoty požadované pri vstupe do ovládača a hodnoty, ktoré musí ovládač vložiť do požiadavky na výstupe. Tieto hodnoty treba dodržovať, pretože na ich základe rozhoduje operačný systém o ďalších operáciách.

Kód 0 – inicializácia ovládača (INIT)
Príkaz INIT slúži na počiatočnú inicializáciu ovládača a zariadení, t. j. na nastavenie spôsobu práce ovládača, na inicializáciu periférneho zariadenia a pod. Na spresnenie inicializácie slúžia parametre, ktoré sú súčasťou príkazu DEVICE. Inicializácia sa vykonáva bezprostredne po zavedení kódu ovládača do operačnej pamäte. Pretože inicializácia sa robí iba raz, je výhodné tento kód umiestniť na koniec ovládača a po inicializácii ho uvoľniť z pamäte. Požiadavka je dlhá 23 bajtov a má tvar podľa tab. 8.

Relatívna adresa	Dĺžka v bajtoch	Vstup	Výstup
0	13	Hlavička požiadavky	Hlavička požiadavky
13	1	-	Počet jednotiek zariadenia
14	4	-	Koncová adresa rezidentného kódu ovládača (iba pre blokové zariadenia)
18	4	Adresa informácií v config.sys	Adresa poľa ukazovateľov (iba pre blokové zariadenia)
22	1	Číslo zariadenia	-

Tabuľka 8

VSTUP: Pri vstupe do ovládača je v požiadavke adresa informácií v súbore config.sys. Je to adresa, ktorá ukazuje do príslušného konfiguračného príkazu DEVICE, a to bezprostredne za znak '='. Napríklad v príkaze DEVICE=C:\SB16\CTSB16.SYS /UNIT=0 /BLASTER=A:220 I:5 D:1 H:5 ukazuje táto adresa na písmeno 'C' umiestnené za znakom '='. Tento ukazovateľ umožňuje spracovanie parametrov obsiahnutých v príkaze DEVICE. Operačný systém upraví text príkazu DEVICE tak, že za špecifikáciou súboru (CTSB16.SYS) je uložená medzera. Táto medzera nahrádza separátor, ktorý bol uvedený v texte. Text riadka je ukončený znakmi 0Dh a 0Ah. Informácie (parametre) je dovolené z riadka iba čítať. Všetky parametre si musí spracovať inicializačný kód sám, to znamená, že parametre môžu mať ľubovoľný tvar. Jediným obmedzením je, že musia byť zapísané do jedného príkazového riadka. Pokiaľ nie sú v riadku uvedené žiadne parametre, napr.: DEVICE=C:\SB16\CTSB16.SYS, bude riadok ukončený medzerou, ktorou systém nahradí znak 0Dh, a znakom 0Ah.

Číslo zariadenia je uložené v požiadavke na adrese 22 (16h) určuje písmeno pridelené systémom prvej jednotke blokového zariadenia, ktoré bude ovládať inicializovaný ovládač. Číslo 0 zodpovedá menu jednotky A, číslo 1 menu B atď. Toto meno môže byť ovládačom použité napr. vo výpise informačnej správy o zavedenom ovládači.

VÝSTUP: Inicializačný kód musí v požiadavke nastaviť nasledujúce hodnoty:
- stavové slovo v hlavičke požiadavky,
- koncovú adresu kódu, ktorý má zostať rezidentný.

Pre blokové zariadenia treba ešte nastaviť:
- počet jednotiek daného zariadenia (iba pre blokové),
- adresu poľa ukazovateľov BPB (blokov parametrov pre BIOS).

Koncová adresa rezidentného kódu ovládača vymedzuje rozsah kódu, ktorý musí zostať rezidentný v pamäti. Operačná pamäť začínajúca sa na tejto adrese bude uvoľnená na ďalšie použitie. Pokiaľ zavádzaný súbor obsahuje viacej ovládačov, berie systém ako koncovú adresu, tú adresu, ktorú vráti posledný inicializovaný ovládač z daného súboru.

Počet jednotiek blokového zariadenia slúži operačnému systému na vytvorenie potrebných vnútorných tabuliek pre jednotlivé jednotky, na pridelenie mien týmto jednotkám a určenie mena prvej jednotky ďalšieho blokového zariadenia. Táto hodnota tiež prepíše hodnotu určenú explicitne v záhlaví ovládača.

Každá jednotka blokového zariadenia musí mať v kóde ovládača obsiahnutý blok parametrov pre BIOS (BPB). Tieto parametre používa operačný systém pri vykonávaní fyzických periférnych operácií, napr. pri čítaní z diskety.

Príkaz INIT vracia v požiadavke adresu poľa ukazovateľov BPB. V poli je pre každú jednotku uložený ukazovateľ na príslušný BPB. Ako ukazovateľ sa do poľa ukladá iba relatívna adresa v danom segmente. Pole ukazovateľov a vlastné bloky parametrov (BPB) musia byť preto obsiahnuté v rovnakom segmente. Pokiaľ má niekoľko jednotiek rovnaké bloky parametrov BPB, stačí, aby bol v ovládači uvedený iba jeden a všetky príslušné ukazovatele potom budú smerovať naň. Pole ukazovateľov aj jednotlivé tabuľky parametrov pre BIOS musia zostať v pamäti rezidentné. Do požiadavky sa ukladá úplná adresa poľa ukazovateľov, t. j. relatívna aj bázová adresa segmentu.

```
;Príklad tabuľky:
BPB_PTR   DW BPB1 ;prvá jednotka
          DW BPB2 ;druhá jednotka
          ...
BPB1      LABEL BYTE  ;sem treba vložiť hodnoty
          ...         ;pre jednotku 1
          ...
BPB2      LABEL BYTE ;sem treba vložiť hodnoty
          ...         ;pre jednotku 1
          ...
```

Dvadsiataštvrtá časť:
Dosovské ovládače (3)
Príkazy ovládača – pokračovanie

Kód 1 – kontrola média
Funkcia kontroly média sa používa na zistenie, či nebolo od naposledy vykonávanej operácie vymenené médium v disketovej (prípadne inej) jednotke. Požiadavka je dlhá 19 bajtov a má tvar podľa tab. 1.

Relatívna adresa	Dĺžka v bajtoch	Vstup	Výstup
0	13	Hlavička požiadavky	Hlavička požiadavky
13	1	Identifikačný bajt média	-
14	1	-	Informácie o výmene
15	4	-	Ukazovateľ na predchádzajúci identifikátor média

Tabuľka 1

VSTUP: Identifikačný bajt média určuje približný typ použitého média. Operačný systém odovzdáva pri kontrole výmeny média identifikačný bajt naposledy spracúvaného média. Ovládač by mal vykonať kontrolu, či tento identifikačný bajt súhlasí s identifikačným bajtom média vloženého v jednotke. Príklady identifikačného bajtu: F8h – pevný disk, F9h – HD disketa 5 1/4" a pod.

Výstup: Ovládač musí v požiadavke nastaviť nasledujúce hodnoty:
- stavové slovo v hlavičke požiadavky,
- informáciu o výmene média,
- ukazovateľ na identifikáciu predchádzajúceho média v prípade, že bolo médium vymenené.

Informácia o výmene média môže nadobúdať tri hodnoty:
-1 (0FFh) – médium bolo vymenené,
0 – nie je možné určiť, či bolo médium vymenené,
1 – médium nebolo vymenené.

Pri zisťovaní, či bolo médium vymenené, vo väčšine prípadov nie je možné vychádzať iba z porovnania identifikačných slabík médií. V systémovom ovládači je na kontrolu výmeny média použitá nasledujúca metóda:
- médium je pevný disk – vracia sa hodnota 1 (pevný disk sa nevymieňa),
- od poslednej úspešnej operácie s médiom neuplynuli ešte 2 sekundy – vracia sa hodnota 1 (v tak krátkom čase nie je možná výmena média),
- nesúhlasia identifikačné bajty médií - vracia sa hodnota –1 (médium bolo vymenené),
- nesúhlasí identifikácia médií – vracia sa hodnota –1(médium bolo vymenené),
- pokiaľ má zariadenie technické vybavenie na signalizáciu výmeny médií, závisí vrátená hodnota od aktuálnej signalizácie,
- V ostatných prípadoch sa vracia hodnota 0 (ovládač nevie, či bolo médium vymenené).

Pokiaľ je testované výmenné médium a ovládač má vrátiť hodnotu –1 (médium bolo vymenené), musí tiež vrátiť ukazovateľ na identifikáciu predchádzajúceho média. V prípade, že ovládač nemá implementovanú podporu identifikácie média, mal by vrátiť ukazovateľ na textový reťazec "NO NAME" ukončený nulovým bajtom. Ak operačný systém zistí, že bola vykonaná neprípustná výmena média, vygeneruje chybu číslo 15 a spracuje ju rovnakým spôsobom, ako keby ju ohlásil ovládač.

Kód 2 – odovzdanie bloku parametrov pre BIOS
Pri inicializácii fyzických periférnych operácií musí byť známa štruktúra média, s ktorým sa daná operácia uskutočňuje. Je to napríklad kapacita média, počet sektorov na stopu, počet sektorov na alokačný blok atď. Pokiaľ má zariadenie pevné médium (napr. pevný disk), potom sa štruktúra média v priebehu práce systému nemení. Ak však má zariadenie možnosť výmeny rôznych typov médií, môže sa výmenou média zmeniť štruktúra média, a tým aj hodnoty používané pri periférnych operáciách. Na opis štruktúry média slúži blok parametrov pre BIOS (BPB). Tento blok vytvára ovládač a odovzdáva jeho adresu operačnému systému.

Odovzdávanie bloku parametrov pre BIOS požaduje operačný systém v prípade, že bolo vymenené médium, alebo v prípade, že nie je známe, či bolo vymenené a vo vnútorných vyrovnávacích pamätiach nie sú žiadne dáta na zápis. Ovládač musí zistiť typ média vložený v jednotke zariadenia, vytvoriť zodpovedajúci blok (BPB) a vrátiť jeho adresu. Formát požiadavky je dlhý 22 bajtov a má tvar podľa tab. 2.

Relatívna adresa	Dĺžka v bajtoch	Vstup	Výstup
0	13	Hlavička požiadavky	Hlavička požiadavky
13	1	Identifikačný bajt média	Identifikačný bajt média
14	4	Adresa vnútornej vyrovnávacej pamäte	Adresa vnútornej vyrovnávacej pamäte
18	4	-	Adresa bloku parametrov pre BIOS

Tabuľka 2

VSTUP: Identifikačný bajt média určuje typ média, s ktorým bola v danej jednotke vykonaná posledná platná operácia.

Na vykonávanie periférnych operácií si operačný systém udržuje niekoľko vnútorných vyrovnávacích pamätí. V požiadavke sa odovzdáva adresa vnútornej vyrovnávacej pamäte, v ktorej je pre zariadenie formátu IBM načítaný začiatok tabuľky obsadenia sektorov vloženého média. Prvý bajt tabuľky je identifikačný bajt média. Tento bajt je možné v niektorých prípadoch použiť na určenie typu média, a tým aj zodpovedajúceho bloku BPB. Obsah vnútornej vyrovnávacej pamäte ani jej adresu v požiadavke nesmie ovládač meniť. Pokiaľ nie je zariadenie vo formáte IBM, môžu byť vo vyrovnávacej pamäti ľubovoľné dáta.

VÝSTUP:

Ovládač musí v požiadavke nastaviť nasledujúce hodnoty:

- stavové slovo v hlavičke požiadavky,
- nový identifikačný bajt média,
- ukazovateľ na blok parametrov pre BIOS.

Bloky BPB jednotlivých typov médií môžu byť uložené priamo ako dáta v kóde ovládača. Príslušný blok sa potom zvolí podľa zisteného média.

Ovládač diskov operačného systému DOS používa ako BPB hodnoty uložené v sektore, ktorý obsahuje zavádzač systému. Tento sektor sa nahráva na každé médium inicializované operačným systémom. Pri požiadavke na odovzdanie adresy BPB prečíta ovládač sektor zavádzača a vráti systému ukazovateľ na BPB obsiahnutý v prečítanom sektore.

Relatívna adresa	Dĺžka v bajtoch	Význam
0	3	Skok na začiatok kódu zavádzača
3	8	Identifikácia systému
11	2	Počet bajtov na sektor
13	1	Počet sektorov na alokačný blok
14	2	Počet rezervovaných sektorov
16	1	Počet kópií tabuľky FAT
17	2	Maximálny počet položiek v koreňovom adresári
19	2	Celkový počet sektorov na médiu
21	1	Identifikačný bajt média
22	2	Počet sektorov obsadených jednou tabuľkou FAT
		blok BPB
24	2	Počet sektorov na stopu
26	2	Počet povrchov
28	2	Počet skrytých sektorov

Tabuľka 3: Tvar začiatku sektora zavádzača

Tri parametre umiestnené za BPB spresňujú opis štruktúry média. Počet povrchov napr. určuje médiá, ktoré majú rovnakú kapacitu a rôzny počet povrchov. Počet skrytých sektorov umožňuje ovládanie médií, ktorých pamäťový priestor je rozdelený medzi viac operačných systémov. Tieto parametre využíva ovládač pri vykonávaní periférnych operácií. Operačný systém použije iba hodnoty uložené v bloku BPB.

Ovládače, ktoré umožňujú spracovanie identifikácie média, by mali pred odovzdaním adresy BPB tiež prečítať z média jeho identifikáciu. Tým sa zabezpečí správnosť a platnosť vykonanej výmeny médií.

Kódy 3, 4, 8, 9,12 – vstup alebo výstup

Príkazy pre vstupy a výstupy vykonávajú fyzické operácie čítania alebo zápisu. Operácie vstupu a výstupu riadiacich sekvencií je možné vykonávať iba v prípade, že je vo vlastnostiach v záhlaví ovládača nastavený bit 14 na 1. Formát požiadavky je dlhý 26 bajtov a má tvar podľa tab. 4.

Relatívna adresa	Dĺžka v bajtoch	Vstup	Výstup
0	13	Hlavička požiadavky	Hlavička požiadavky
13	1	Identifikačný bajt média	–
14	4	Adresa vyrovnávacej pamäte	Adresa vyrovnávacej pamäte
18	2	Počítadlo bajtov alebo sektorov	Počet prenesených bajtov alebo sektorov
20	2	Číslo počiatočného sektora	–
22	4	–	Adresa identifikácie média (pokiaľ je hlásená chyba 15)

Tabuľka 4

VSTUP: Identifikačný bajt média určuje typ média vloženého v jednotke.

Adresa vyrovnávacej pamäte určuje miesto v pamäti, kam sa majú dáta čítať (pre vstup) alebo odkiaľ majú byť vypísané (výstup).

Počítadlo bajtov alebo sektorov určuje dĺžku čítaných alebo zapisovaných dát. Pre znakové zariadenia sa dĺžka určuje v bajtoch, pre blokové v sektoroch.

Číslo počiatočného sektora určuje logický sektor na médiu, od ktorého sa bude začínať prenos dát (čítanie alebo zápis). Táto položka má význam iba pre blokové zariadenia, pre znakové sa ignoruje.

VÝSTUP: Ovládač musí vykonať požadovanú operáciu a v požiadavke nastaviť nasledujúce hodnoty:

- stavové slovo v hlavičke požiadavky,
- počet skutočne prečítaných (zapísaných) bajtov alebo sektorov,
- pri zistení chyby 15 adresu identifikácie média.

Hodnota počtu skutočne prenesených bajtov musí byť nastavená aj pre vstup alebo výstup sekvencie riadiacich znakov. Pri zistení chyby číslo 15 musí ovládač vrátiť aj adresu identifikácie pôvodného média, t. j. média, pre ktoré boli až doteraz vykonávané periférne operácie. Identifikácia musí byť uložená v pamäti v tvare ASCIIZ2 a operačný systém ju použije vo výpise požiadaviek na výmenu média. Systémový ovládač disku zisťuje nepovolenú výmenu (chybu 15) podľa počtu otvorených súborov. Chyba sa ohlási v prípade, že je nejaký súbor otvorený a od poslednej periférnej operácie došlo k výmene média. Pokiaľ nie je žiadny súbor otvorený, vykoná sa operácia vstupu (výstupu) bez ohľadu na výmenu média.

Kód 5 – nedeštruktívny vstup bez čakania

Tento príkaz umožňuje zistiť, či je na vstupe k dispozícii nejaký znak. Je možné ho použiť iba pre znakové zariadenia. Formát požiadavky je dlhý 14 bajtov a má tvar podľa tab. 5.

Relatívna adresa	Dĺžka v bajtoch	Vstup	Výstup
0	13	Hlavička požiadavky	Hlavička požiadavky
13	1	–	Prečítaný bajt

Tabuľka 5

VÝSTUP: Ovládač musí zistiť, či je k dispozícii nejaký dátový bajt, a v požiadavke musí nastaviť nasledujúce hodnoty:

- stavové slovo v hlavičke požiadavky,
- prečítaný bajt (pokiaľ boli dáta k dispozícii).

Stav vstupu zisťuje operačný systém podľa nastavenia bitu 9 (BUSY) v stavovom bajte. Pokiaľ je hodnota tohto bitu rovná 0, potom sú dáta k dispozícii a v požiadavke je vrátený platný dátový bajt. Obsah vstupnej vyrovnávacej pamäte sa pritom nijako nezmení. K odstráneniu bajtu z vyrovnávacej pamäte dôjde až po prečítaní dát príkazom s kódom 4. Ak je bit 9 vo vrátenom stavovom bajte rovný 1, nie sú žiadne vstupné dáta momentálne k dispozícii.

Kód 6, 10 – stav zariadenia

Príkaz zisťuje aktuálny stav prenosu dát. Možno použiť iba pre znakové zariadenia. Kód 6 sa používa na zistenie stavu vstupného zariadenia, kód 10 na zistenie stavu výstupného zariadenia. Formát požiadavky je dlhý 13 bajtov a má tvar podľa tab. 6.

Relatívna adresa	Dĺžka v bajtoch	Vstup	Výstup
0	13	Hlavička požiadavky	Hlavička požiadavky

Tabuľka 6

VÝSTUP: Ovládač musí zistiť stav prenosu a nastaviť príslušné hodnoty v stavovom slove v hlavičke požiadavky. Stav sa oznamuje hodnotou bitu číslo 9 (BUSY). Jeho nastavenie sa riadi nasledujúcimi pravidlami:

1. Pre výstupné znakové zariadenia – bit sa nastaví na 1 v prípade, že by mala požiadavka na zápis čakať na ukončenie práve vykonávanej požiadavky; nastavenie na hodnotu 0 sa vykoná v prípade, že sa nevykonáva žiadna požiadavka a ďalšia požiadavka na zápis môže byť vykonaná okamžite.
2. Pre vstupné znakové zariadenia - bit sa nastaví na hodnotu 0 v prípade, že je znak prítomný vo vyrovnávacej pamäti; pokiaľ sa má fyzicky čítať, nastaví sa bit na 1. Operačný systém predpokladá, že ovládače znakových zariadení vykonávajú čítanie dopredu – do vyrovnávacej pamäte. Pokiaľ ovládač nemá vyrovnávaciu pamäť, musí sa bit 9 nastaviť vždy na 0, aby systém nečakal na dáta, ktoré majú byť uložené v neexistujúcej vyrovnávacej pamäti.

Kód 7, 11 – zrušenie aktuálnych požiadaviek

Príkaz slúži na zrušenie všetkých inicializovaných, ale ešte nevykonaných požiadaviek. Kód 7 – ruší požiadavky pre vstupné zariadenie, kód 11 – ruší požiadavky pre výstupné zariadenie. Formát požiadavky je dlhý 13 bajtov a má tvar podľa tab.7.

Relatívna adresa	Dĺžka v bajtoch	Vstup	Výstup
0	13	Hlavička požiadavky	Hlavička požiadavky

Tabuľka 7

VÝSTUP: Ovládač musí vykonať požadovanú operáciu a nastaviť stavové slovo v hlavičke požiadavky. Základnou úlohou uvedeného príkazu je zrušenie frontu požiadaviek na dané zariadenie a prípadné zrušenie dát vo vyrovnávacích pamätiach. Napríklad systé-

mový ovládač klávesnice zruší všetky ešte nespracované znaky vo vnútornej vyrovnávacej pamäti.

Kód 13, 14 - otvorenie alebo uzatvorenie
Tieto príkazy sú volané operačným systémom pri otváraní a zatváraní súborov. Používajú sa pre blokové aj pre znakové zariadenia. Formát požiadavky je dlhý 13 bajtov a má tvar podľa tab. 8.

Relatívna adresa	Dĺžka v bajtoch	Vstup	Výstup
0	13	Hlavička požiadavky	Hlavička požiadavky

Tabuľka 8

VÝSTUP: Ovládač musí nastaviť stavové slovo v hlavičke požiadavky. V operačnom systéme používajú tieto príkazy ovládače blokových zariadení s vymeniteľnými médiami (majú nastavený bit 11 vo vlastnostiach v záhlaví ovládača na 1). Ovládač si udržuje počítadlo otvorených súborov. Pri každom otváraní súboru ho zvyšuje o 1, pri zatváraní ho znižuje o 1, t. j. ak nie je hodnota počítadla rovná 0, je otvorený nejaký súbor a médium by sa nemalo v žiadnom prípade vymieňať. Ovládač môže toto počítadlo testovať pri kontrole média, prípadne ohlásiť chybu 15.

Príkazy na otvorenie a zatvorenie je tiež možné využívať pre znakové zaradenia. Otvorenie možno využiť na poslanie inicializačnej postupnosti dát na zariadenie, napr. pre tlačiareň voliť typ písma a pod. Obdobne sa dá využiť uzatvorenie napr. pre stránkovanie. Pre uvedené operácie sa však častejšie používajú sekvencie riadiacich znakov, ktoré umožňujú rozmanitejšie spôsoby ovládania zariadení.

Kód 15 – Zistenie výmenného média
Tento príkaz sa používa iba pre blokové zariadenia a zisťuje, či má dané zariadenie výmenné médiá. Formát požiadavky je dlhý 13 bajtov a má tvar podľa tab. 9.

Relatívna adresa	Dĺžka v bajtoch	Vstup	Výstup
0	13	Hlavička požiadavky	Hlavička požiadavky

Tabuľka 9

VÝSTUP: Ovládač musí nastaviť obsah stavového slova v hlavičke požiadavky. Informácie o type zariadenia sa zobrazia v bite 9 tohto slova. Pre výmenné médiá sa tento bit nastaví na 0, pre pevné médiá na 1. Operačný systém volá tento príkaz prostredníctvom funkčného volania 44h. Toto volanie používajú programy, ktoré potrebujú pre svoju činnosť poznať, či je médium výmenné, alebo nie. Pri návrate do systému sa netestuje chybový bit. Predpokladá sa, že príkaz sa skončí vždy úspešne.

Kód 19 – požiadavka na spracovanie generovanej sekvencie riadiacich znakov
Kód 19 je generovaný operačným systémom pri spracovaní funkčného volania 44h, funkcia 0Dh. Používa sa na vykonávanie špeciálnych operácií. Formát požiadavky je dlhý 23 bajtov a má tvar podľa tab. 10.

Relatívna adresa	Dĺžka v bajtoch	Vstup	Výstup
0	13	Hlavička požiadavky	Hlavička požiadavky
13	1	Základné číslo funkcie	-
14	1	Doplnkové číslo funkcie	-
15	2	Obsah registra SI	-
17	2	Obsah registra DI	-
19	4	Adresa dát	-

Tabuľka 10

VÝSTUP: Ovládač musí vykonať požadovanú funkciu a nastaviť obsah stavového slova v hlavičke požiadavky. Pretože niektoré funkcie umožňujú neštandardné formátovanie médií, musí si ovládač udržovať príslušné tabuľky formátu stôp média. Základné číslo funkcie odovzdávané v požiadavke je pre všetky povolené funkcie rovné 08h.

Kód 23, 24 – zistenie a nastavenie logického zariadenia
Tieto kódy sa používajú v prípadoch, keď je jednej jednotke niektorého zariadenia pridelených viac logických mien. Ako príklad si môžeme uviesť konfiguráciu počítača s jednou disketovou jednotkou. Tejto jednotke sa pri zavádzaní operačného systému pridelia logické mená A a B. Používateľ alebo program môže na jednotku odkazovať obidvoma menami. Meno, ktoré bolo použité pri poslednej operácii s jednotkou, označuje tzv. logické zariadenie.

Operačný systém si udržuje informáciu o logickom zariadení pre každú jednotku blokového zariadenia. Logické zariadenie sa však mení iba pri jednotkách, ktorým je pridelených viac mien. Pokiaľ má jednotka priradené iba jedno meno, potom toto meno označuje logické zariadenie po celý čas práce systému.

Formát požiadavky je dlhý 13 bajtov a má tvar podľa tab. 11.

Relatívna adresa	Dĺžka v bajtoch	Vstup	Výstup
0	13	Hlavička požiadavky	Hlavička požiadavky

Tabuľka 11

VSTUP: Na zisťovanie logického zariadenia je v čísle jednotky uvedené ľubovoľné číslo, ktoré zodpovedá niektorému menu pridelenému danej jednotke zariadenia (0 = implicitná jednotka, 1=A, 2=B atď.). Na nastavenie logického zariadenia je v čísle jednotky uvedené číslo zodpovedajúce požadovanej jednotke (1=A, 2=B atď.).

VÝSTUP:
Ovládač musí vykonať požadovanú operáciu a nastaviť:
- stavové slovo v hlavičke požiadavky,
- aktuálne číslo logického zariadenia; pokiaľ má jednotka iba jedno meno, nastaví sa hodnota 0, inak sa nastaví číslo zodpovedajúce menu danej jednotky (1=A, 2=B atď.).

Praktický príklad
Ako príklad ovládača si ukážeme jednoduchý program **screen.asm**. Tento ovládač umožňuje vkladať medzi dlhé výpisy (napr. výpis obsahu adresárov príkazom DIR alebo výpis obsahu súboru príkazom TYPE a pod.) pauzu. Pokračovať vo výpise môžete až po stlačení ľubovoľného klávesu. Na tento ovládač sa veľmi podobá dosovský príkaz MORE.COM.

Preklad programu
Ovládač SCREEN.SYS dostanete nasledujúcim postupom: **Tasm.exe screen.asm** a **Tlink.exe screen.obj**. Vytvorí sa súbor SCREEN.EXE. Teraz je potrebné príkazom EXE2BIN transformovať vytvorený súbor screen.exe do súboru, ktorý bude obsahovať obraz uloženia programu v pamäti. Funkčnú stánku tohto ovládača môžete vyskúšať po napísaní nasledujúceho riadka do súboru config.sys: **DEVICE C:\SCREEN.SYS**. O správnej inštalácii ovládača v pamäti sa môžete presvedčiť príkazom MEM /c /p, ovládač s názvom screen by sa mal nachádzať vo výpise. V prípade, že sa nenachádza, urobili ste niekde chybu – pozrite si súbor config.sys. Po inštalovaní ovládača skúste zadať napríklad nasledujúce príkazy: **TYPE meno súbor > SCR** alebo **DIR > SCR**. Meno súboru je hocijaký textový súbor. Znaky '> SCR' znamenajú presmerovanie výpisu na náš ovládač SCREEN. SYS.

Diskety k článkom O ASSEMBLERI
Pretože sa seriál už blíži pomaly ku koncu, prepravil som pre prípadných záujemcov výber zdrojových kódov.

Na dvoch disketách sa nachádzajú všetky zdrojové kódy, ktoré boli uverejnené v tomto článku + zdrojové kódy týkajúce sa grafiky, disku, myši a pod. Uvedené diskety je možné získať po zaslaní 120 Sk poštovou poukážkou typu C na adresu:
Peter Gašparovič
Zajačia 23
919 04 Smolenice

Do správy pre prijímateľa uveďte "**Diskety – ASSEMBLER**". Uvedená suma slúži na úhradu nákladov za diskety, kopírovanie, balné a poštovné. Navyše ešte získate 50-stranovú tlačenú príručku s tabuľkami, ktorú určite využijete pri programovaní v assembleri. Príručka obsahuje napr.: Inštrukčný súbor 8086, kódy klávesov, premenné BIOS-u, funkcie DOS, ASCII tabuľku a pod.

Literatúra
[1] Ralf Brown: Interrupt List - Release 33. Pittsburgh PA, U.S.A. 1993.
[2] Vlastislav Černý: MS-DOS 6.22. Kopp 1996.
[3] P. Heřman: Příručka systémového programátora. Tesla Eltos 1990.
[4] David Jurgens: HelpPC 2.10.
[5] ABSHelp 2.05, ABSoft Olomouc.
[6] Michal Brandejs: MS-DOS 6. Kompletní průvodce. Grada 1993.

Vysvetlivky
2 ASCIIZ (American Standart Code of International Interchange Zero) – reťazec zakončený nulou.
3 CLOCK$ – je ovládač hodín, využívaný príkazmi TIME, DATE a programami, ktoré na svoju činnosť potrebujú poznať dátum a čas.

Dvadsiapiata časť:
OPP v Assembleri (DOS)

Skratka OOP znamená Object-Oriented Programming, v doslovnom preklade objektovo orientované programovanie. Program Turbo Assembler (TASM) od verzie 3.0 obsahuje prostriedky na vytváranie objektovo orientovaných programov. Objekt je možné chápať ako zlúčenie dát a procedúr, resp. funkcií, ktoré s týmito dátami pracujú, do jedného celku. Jednotlivé dáta objektu pomenujeme položky a procedúry (funkcie) budeme nazývať metódy. V terminológii OOP sa používajú tri základné princípy:

- zapuzdrenie (encapsulation),
- dedičnosť (inheritance),
- mnohotvárnosť (polymorphism).

Zapuzdrenie (encapsulation) predstavuje ukrytie dát priradených objektom do ich vnútra. K jednotlivým dátam je možné pristupovať len pomocou metód pridelených príslušnému objektu. Táto vlastnosť vedie k tomu, že používanie objektu je nezávislé od vnútornej reprezentácie dát. Pokiaľ sa z nejakého dôvodu vnútorné dáta objektu zmenia, stačí iba upraviť príslušné metódy. Program, ktorý objekt používa, zostane nezmenený.

Dedičnosť (inheritance) znamená, že jednotlivé objekty majú schopnosť dediť vlastnosti od svojich predchodcov, pričom ich môžu modifikovať a vytvárať nové. TASM štandardne podporuje iba jednoduchú dedičnosť, t. j. objekt môže mať iba jeden rodičovský objekt. Objekt, ktorý nemá rodičovský objekt, sa nazýva základný objekt.

Monohotvárnosť (polymorphism) sa viaže na vlastnosť OOP vykonávať metódy definované pod rovnakým menom rôznym spôsobom, závislým od objektu, ktorý metódu volá. Inými slovami, odvodený objekt môže mať metódu rovnakého mena ako rodičovský objekt, ktorá však vykonáva inú činnosť. Polymorfizmus je v TASM realizovaný pomocou virtuálneho volania procedúr. Dôsledkom toho je možnosť pracovať s objektmi, pri ktorých v dobe kompilácie nepoznáme presné typy.

Nasledujúca tabuľka dáva prehľad o výrazoch týkajúcich sa OOP a ich porovnanie s terminológiou iných programovacích jazykov. Je dôležité si uvedomiť rozdiel medzi inštanciou objektu a samotným objektom. Objekt je iba definícia typu. Inštanciou objektu sú konkrétne vytvorené objekty daného typu.

TASM	Turbo Pascal	Borland C++	Preklad
method	Method	member function	metóda
method procedure			procedúra metódy
object	Object	class	objekt
base object	base object	base class	základný objekt
parent object	parent object	parent class	rodičovský objekt
derived object	derived object	derived class	odvodený objekt
field	field	data member	položka
instance	instance	instance	inštancia objektu

Nový typ TABLE

Assembler TASM od verzie 3.0 obsahuje nový dátový typ TABLE. Predstavuje tabuľku hodnôt (najčastejšie ukazovateľov na procedúry). Každá hodnota má svoju veľkosť a môže mať inicializačnú hodnotu. Tento typ bol implementovaný hlavne pre objekty, kde sa používa na reprezentáciu tabuľky metód objektov, je ho však možné využiť aj na iné účely.

Režim IDEAL (jednoriadková verzia):

```
TABLE meno [tbl_prem [,tbl_prem...]]
```

Režim IDEAL (viacriadková verzia):

```
TABLE meno {
        [tbl_prem]
        [tbl_prem]
        ...
        }
```

Režim MASM (jednoriadková verzia):

```
meno TABLE [tbl_prem [,tbl_prem...]]
```

Režim MASM (viacriadková verzia):

```
meno TABLE  {
        [tbl_prem]
        [tbl_prem]
        ...
                }
```

Položky **tbl_prem** majú nasledujúcu syntax:

[VIRTUAL] meno_položky [[počet]]
[:typ [:počet2]] [= výraz]

Identifikátor **meno_položky** je názov položky tabuľky. Položky môžu byť virtuálne (VIRTUAL) alebo statické. Virtuálne položky majú v tabuľke vyhradené miesto a toto miesto má určitý offset od začiatku tabuľky (virtuálne metódy objektov sa volajú nepriamo cez tabuľku). Statické položky nemajú vyhradené žiadne miesto, a teda ani offset (statické metódy sa volajú priamo, nemusia teda byť v tabuľke). Veľkosť tabuľky je daná súčtom veľkostí virtuálnych položiek.

Typ určuje typ položky. Na tomto mieste je možné použiť ľubovoľný dátový typ. Pokiaľ typ nie je uvedený, predpokladá sa typ **WORD**, prípadne v modeli USE32 typ **DWORD**.

Výraz **počet1** umožňuje vytvoriť pole položiek daného typu. Počet položiek v poli udáva práve **počet1**. Pokiaľ sa neuvedie, predpokladá sa jedna položka. Výraz počet2 určuje, koľko položiek daného typu sa má vytvoriť. Implicitná hodnota je 1. Celkové miesto rezervované pre položku sa vypočíta podľa vzorca: **(veľkosť typu dát) *počet1*počet2**. Inicializačnú hodnotu položky určuje parameter výraz. Táto hodnota je väčšinou ukazovateľom na nejakú procedúru. Položka **tbl_prem** môže byť aj názov už existujúcej tabuľky. Týmto spôsobom je možné vkladať tabuľky do seba. Táto vlastnosť sa používa na implementáciu dedičnosti metód objektov. Pokiaľ sú deklarované dve alebo viac položiek s rovnakým názvom, TASM skontroluje, či sú ich veľkosti a typy rovnaké. V prípade, že typ a veľkosť sú totožné, použije TASM inicializačnú hodnotu poslednej deklarovanej položky. Ak typ alebo veľkosť nesúhlasia, ohlási TASM chybu. Táto vlastnosť sa používa na prepísanie inicializačnej hodnoty pri vkladaní tabuliek.

Príklad:

```
Tbl1 TABLE VIRTUAL pol1 = proc1, VIRTUAL pol2 = proc2
Tbl2 TABLE Tbl1, pol1 = proc3
```

Na alokovanie pamäte je možné použiť názov šablóny **meno** rovnakým spôsobom ako ostatné direktívy (DB, DW atď.):

premenná meno_tabuľky inicializácia [,inicializácia...]

Identifikátor **premenná** je názov definovanej premennej. **Meno_tabuľky** je názov šablóny tabuľky. Na inicializáciu vytvorenej premennej je možné použiť tieto hodnoty:

- **?**– všetky položky budú mať nedefinovaný obsah,
- { } – na inicializáciu sa použijú implicitné hodnoty,
- {**položka = hodnota [, položka = hodnota]**} – uvedené položky budú mať zadanú hodnotu, ostatné budú mať implicitné hodnoty.

Príklad:

```
Premenná1 Tbl1 ? ; neinicializovaná tabuľka
Premenná2 Tbl1 { } ; implicitné hodnoty
Premenná3 Tbl2 {pol2 = proc4} ; pol1 implicitná (proc3), pol2 nová hodnota (popzri predchádzajúci príklad)
```

V niektorých prípadoch je vhodné poznať pôvodnú inicializačnú hodnotu položky tabuľky (hodnota uvedená pri definovaní šablóny tabuľky). Túto hodnotu je možné získať zápisom **názov_tabuľky | názov_položky**. Táto vlastnosť sa používa na volanie metód predkov v objektoch.

Príklad:

```
CALL +Tbl2 | pol2 ; volanie procedúry proc2 (pozri prvý príklad)
```

Objekty v TASM

V TASM je objekt realizovaný rozšírením dátového typu STRUC. Deklarácia objektu má nasledujúcu syntax:

Režim IDEAL:

STRUC názov [modifikátory] [názov_rodičovského_objektu] [**METHOD** [položka_tabuľky [položka_tabuľky, ...]]]
položky_štruktúry
ENDS [názov]

Režim MASM:

názov **STRUC** [modifikátory] [názov_rodičovského_objektu] [**METHOD** [položka_tabuľky [položka_tabuľky, ...]]]
položky_štruktúry
[názov] **ENDS**

kde **názov** je názov objektu, **názov_rodičovského_objektu** je nepovinný názov predka. **Položky_tabuľky** opisujú metódy objektu. Syntax zápisu je rovnaká ako pri type **TABLE**. Na zápis je možné použiť aj viacriadkovú syntax. **Položky_štruktúry** opisujú dáta objektu. Syntax zápisu je rovnaká ako pri štandardnom type **STRUC**.

Modifikátory môžu byť nasledujúce:

GLOBAL – spôsobí, že tabuľka virtuálnych metód (Virtual Method Table ďalej len VMT) bude prístupná globálne.

NEAR – ukazovateľ na VMT bude vyjadrený iba offsetom, buď 16, alebo 32 bitov, v závislosti od toho, či je aktuálny model USE16 alebo USE32.
FAR – ukazovateľ na VMT bude vyjadrený offsetom aj segmentom (32 alebo 64 bitov).
Deklaráciou objektu vznikne niekoľko preddefinovaných symbolov:
@Objekt – textové makro, ktoré obsahuje názov aktuálneho objektu, t. j. objektu, ktorý bol deklarovaný ako posledný.
názovobjektu – dátový typ **STRUC**, ktorý obsahuje dátovú štruktúru objektu.
@Table_názovobjektu – dátový typ TABLE, ktorý obsahuje tabuľku metód objektu.
@TableAddr_názovobjektu – návestie inštancie tabuľky virtuálnych metód. Toto návestie je deklarované ako **EXTRN**. Konkrétnu inštanciu VMT je potrebné vytvoriť napríklad pomocou **TBLINST**.
@Mptr_názovobjektu – položka zo štruktúry **názovobjektu** obsahujúca ukazovateľ na VMT objektu.

Príklad deklarácie objektu:

```
STRUC Point GLOBAL METHOD {
            construct:WORD = Point_construct
            destruct:WORD = Point_destruct
            init:WORD = Point_init
virtual     show:WORD = Point_show
}
X           DB 0
Y           DB 0
ENDS
```

Deklaráciou objektu vzniknú dva dátové typy. Typ **STRUC** s názvom Point a typ @Table_Point typu **TABLE**. Typ Point obsahuje okrem položiek X a Y aj položku s názvom **@Mptr_Point**. Táto položka, ktorú TASM automaticky pridá do dátovej štruktúry základného objektu, je ukazovateľ na tabuľku virtuálnych metód. Pozor! Táto položka nie je inicializovaná, pred volaním virtuálnych metód sa musí najprv naplniť ukazovateľom na inštanciu VMT. Veľkosť tohto ukazovateľa ovplyvňujú modifikátory a použitý dátový model. TASM ho štandardne umiestňuje na koniec dátovej štruktúry základného objektu. Pomocou direktívy **TBLPTR** ho môžete umiestniť na ľubovoľné miesto v dátovej štruktúre základného objektu. Potomkovia tento ukazovateľ zdedia od základného objektu. Typ @Table_Point obsahuje tabuľku metód objektu Point. Metódy poznáme statické a virtuálne. Pri volaní statickej metódy sa priamo zavolá procedúra metódy. Virtuálne metódy sa volajú pomocou VMT.

Virtuálne metódy a VMT
Význam virtuálnych metód si ukážeme na príkladoch.

Príklad 1:
Predpokladajme, že máme vyššie deklarovaný objekt Point a z neho odvodíme objekt Line.

```
STRUC Line GLOBAL Point METHOD {
            construct:WORD = Line_construct
            destruct:WORD = Line_destruct
            init:WORD = Line_init
virtual     show:WORD = Line_show
}
Length      DB 0
ENDS
```

Obidva uvedené objekty (Line a Point) majú virtuálnu metódu show, ktorá zobrazí objekt na súradniciach X, Y. Point_show zobrazí bod a Line_show zobrazí čiaru. Objekt Line má navyše položku Length, ktorá určuje dĺžku čiary. Predpokladajme, že píšeme procedúru, ktorá dostane ako parameter ukazovateľ na inštanciu niektorého z týchto dvoch objektov (t. j. na štruktúru dát) a jej úlohou je zobraziť tento objekt pomocou metódy show. Či ide o inštanciu objektu Point alebo Line, to v dobe písania programu nevieme. Pokiaľ je metóda virtuálna, stačí predpokladať, že dostaneme ukazovateľ na inštanciu objektu Point, a vykonať volanie metódy show tohto objektu. Pri behu programu sa potom zavolá procedúra metódy toho objektu, na ktorého inštanciu ukazovateľ ukazuje.

Príklad 2:
Zoznam metód objektu Point doplníme o dve metódy (hide, move)

```
STRUC Point GLOBAL METHOD {
            construct:WORD = Point_construct
            destruct:WORD = Point_destruct
            init:WORD = Point_init
virtual     show:WORD = Point_show
virtual     hide:WORD = Point_hide
            move:WORD = Point_move
}
X           DB 0
Y           DB 0
ENDS
```

a do objektu Line pridáme metódu hide.

```
STRUC Line GLOBAL Point METHOD {
                    construct:WORD = Line_construct
                    destruct:WORD = Line_destruct
                    init:WORD = Line_init
virtual             show:WORD = Line_show
virtual             hide:WORD = Line_hide
}
Length              DB 0
ENDS
```

Metódy hide zmažú príslušný objekt z obrazovky. Metóda move presunie objekt na zadané súradnice tak, že najprv zavolá metódu hide, nastaví nové súradnice objektu a zavolaním metódy show zobrazí objekt na novej pozícii. Objekt Line zdedí metódu move od objektu Point. Pokiaľ zavoláme metódu move objektu Point, dôjde k volaniu procedúr metód Point_hide a Point_show. Ak však zavoláme metódu move pri objekte Line, dôjde k volaniu procedúr metód Line_hide a Line_show. V prípade, že by sme deklarovali metódy show a hide ako statické, došlo by pri volaní metódy move vždy k vyvolaniu Point_hide a Point_show.

V obidvoch prípadoch program až za behu zistí, ktorú konkrétnu procedúru má zavolať. Tomuto spôsobu volania procedúr sa hovorí neskorá väzba (Late binding). Tajomstvo neskorej väzby je v tabuľke VMT. Táto tabuľka obsahuje adresy procedúr virtuálnych metód objektu. Tieto adresy sa môžu pre každý objekt líšiť. Každý objekt (nie inštancia objektu) má preto vlastnú inštanciu VMT. Vtip je v tom, že poloha adries procedúr a metód s rovnakým názvom metódy je vo VMT rovnaká pre všetky objekty danej hierarchie. Napríklad náš objekt Point má adresu procedúry metódy show v prvej položke tabuľky (statické položky v inštancii tabuľky nie sú) a adresu procedúry metódy hide v druhej položke tabuľky. Potom objekt Line a všetky objekty, ktoré sú odvodené od objektu Point, budú mať adresy procedúr metód show a hide na rovnakých miestach v tabuľke ako objekt Point. Informáciu o polohe v tabuľke, t. j. offset od začiatku tabuľky, nesú názvy metód. Volanie virtuálnych metód sa robí nepriamym skokom na procedúru metódy, ktorá je uložená v tabuľke virtuálnych metód. Tým dôjde k vyvolaniu správnej procedúry metódy.

Ukazovateľ na inštanciu objektu ukazuje v skutočnosti na dáta daného objektu. Posledná vec, ktorú treba vyriešiť, je zistiť z tejto dátovej štruktúry adresu VMT. V prechádzajúcom texte sme uviedli, že TASM sám doplní dátovú štruktúru základného objektu o položku **@Mptr_názovobjektu**, ktorá po inicializácii obsahuje adresu VMT. Odvodené položky túto hodnotu zdedia. Poloha tejto položky (t. j. offsetu od začiatku dátovej štruktúry) je rovnaká pre všetky objekty danej hierarchie.

Vytvorenie inštancie VMT musí urobiť programátor. VMT sa väčšinou umiestňuje do dátového segmentu. Na vytvorenie inštancie naposledy deklarovaného objektu (t. j. objektu, ktorého názov obsahuje preddefinované makro **@Objekt**) je možné použiť direktívu **TBLINST**.
Príklad:

```
...
; tu bude deklarácia objektu
DATASEG
TBLINST
...
```

TBLINST je v skutočnosti preddefinované makro, ktoré sa rozloží do nasledujúceho riadka:
@TableAddr_aktuálnyobjekt @Table_aktuálnyobjekt { }

Deklarácie objektov je vhodné umiestniť do samostatných súborov, tie potom začleniť do zdrojového textu pomocou direktívy INCLUDE. Pri vytváraní väčšieho počtu objektov je výhodnejšie umiestniť niekoľko deklarácií (prípadne všetky) do jedného súboru. Direktíva TBLINST vytvorí inštanciu iba posledného deklarovaného objektu. Na vytvorenie inštancie ľubovoľného objektu môžete použiť nasledujúce makro:
Príklad:

```
MACRO TBLINSTO object
@TableAddr_&object @Table_&object { }
ENDM
```

Použitie tohto makra je podobné ako pri **TBLINST**. Zápis **TBLINSTO @Object** je rovnocenný direktíve **TBLINST**.
Príklad:

```
...
INCLUDE <súbor s deklaráciami objektov>
DATASEG
```

```
TBLINSTO názov_objektu1
TBLINSTO názov_objektu2
...
```

Volanie metód – príkaz CALL

Na volanie metód používa TASM rozšírený zápis inštrukcie CALL. Spôsob odovzdávania parametrov závisí od programátora (t. j. je možné použiť zásobník alebo registre). Vo väčšine prípadov sa procedúre metódy odovzdá aspoň jeden parameter – ukazovateľ na inštanciu objektu. Pri volaní metód je potrebné rozlišovať medzi volaním statickej a virtuálnej metódy.

Virtuálne metódy

Volanie virtuálnych metód sa vykonáva nepriamo cez VMT. Pozor! Pred volaním metódy je potrebné inicializovať ukazovateľ na VMT. Syntax zápisu volania virtuálnej metódy je nasledujúci:

CALL inštancia **METHOD** [názov_objektu:] metóda [**USES** [seg:]reg] [jazyk_a_parametre]

kde **inštancia** je ukazovateľ na inštanciu objektu. Môže to byť návestie inštancie objekt alebo dvojica registrov (segmentový a bázový alebo indexový), v ktorých je adresa inštancie. Pokiaľ sa použije režim prekladača **IDEAL**, nemusí sa **názov_objektu** uvádzať. **Seg** je segmentový register DS alebo ES, ktorý sa použije pre offsetovú adresu VMT. V prípade, že sa klauzula **USES** neuvedie, použijú sa registre ES:BX. **Jazyk_a_parametre** určujú volacie konvencie a parametre rovnakým spôsobom ako pri klasickej inštrukcii CALL. V prípade, že ukazovateľ na VMT je typu NEAR (napr. VMT je umiestnená v dátovom segmente a je použitý niektorý z modelov TINY, SMALL alebo MEDIUM), rozvinie sa volanie na takúto postupnosť inštrukcií.

```
MOV reg,[inštancia.ukazovateľ_na_VMT]
CALL [(seg:reg).metóda] ďalšie_prvky
```

Register **seg** teda musí byť naplnený správnou hodnotou ešte pred volaním metódy. Ak ukazovateľ na inštanciu je typu FAR, nahradí sa inštrukcia MOV inštrukciou LDS, prípadne LES v závislosti od špecifikovaného segmentového registra.

Volanie metódy show inštancie MyPoint objektu Point môže vyzerať napríklad takto:

```
CALL MyPoint METHOD Point:show USES DS:BX PASCAL, DS OFFSET MyPoint
[,<ďalšie_parametre>]
```

alebo

```
MOV AX, SEG MyPoint
MOV ES,AX
MOV DI, OFFSET MyPoint
CALL Point PTR ES:DI METHOD Point:show USES DS:BX PASCAL, ES DI [,<ďalšie_parametre>]
```

Statické metódy

Pri vytváraní statickej metódy sa priamo volá procedúra metódy. Volanie môže vyzerať takto:

```
CALL procedúra_metódy [ďalšie_prvky]
```

alebo

```
CALL +@Table_názovobjektu | metóda [ďalšie_prvky]
```

Najvhodnejšie je však použiť rovnaký tvar ako pri volaní virtuálnych metód vrátane klauzuly USES. Špecifikované registre sa síce nepoužijú, bude sa ignorovať aj adresa inštancie objektu, ale pri neskoršej zmene statickej metódy na virtuálnu a naopak už nebudete musieť volanie tejto metódy meniť. Okrem inštrukcie CALL je, samozrejme, možné použiť aj inštrukciu JMP. Rozšírená syntax inštrukcie JMP je rovnaká ako pri inštrukcii CALL. Použitie tohto spôsobu je výhodné hlavne pri implementácii niektorých rekurzívnych algoritmov.

Inštancia objektu

Inštanciu objektu je možné vytvoriť dvojakým spôsobom – buď alokovaním miesta v niektorom segmente rovnakým spôsobom ako pri vytváraní inštancie štruktúry, alebo za behu programu volaním konštruktora. Inštanciu objektu Point prvým spôsobom (statická alokácia) je možné vytvoriť napríklad takto:

```
...
DATASEG
MyPoint             Point {}
YourPoint           Point {X=10, Y=20}
...
```

Konštruktor

Konštruktor je ľubovoľná statická metóda, ktorá alokuje niekde v pamäti dostatočne veľký priestor pre dátovú štruktúru objektu, vykoná prípadnú inicializáciu dát objektu a vráti ukazovateľ na alokovanú oblasť. Alokácia vykonaná týmto spôsobom sa nazýva dynamická alokácia. Okrem toho, že koštruktor musí byť statická metóda, nekladie prekladač TASM na koštruktory nijaké ďalšie podmienky. Názov metódy konštruktora môže byť ľubovoľný. Objekt môže mať aj niekoľko konštruktorov, ale s rôznymi názvami. Spôsob, akým konštruktor vykoná alokáciu pamäte, závisí iba od toho, akým spôsobom program organizuje pamäť. Môže sa použiť aj napr. INT 21h, služba 48h.

Deštruktor

V programe môže nastať aj situácia, že dynamicky alokovaná inštancia objektu sa už ďalej nebude používať, a preto by bolo vhodné uvoľniť pamäť, ktorú táto inštancia zaberá. Na tento účel sa používa metóda nazývaná deštruktor. Jej úlohou je vykonať všetky akcie spojené so zánikom inštancie objektu, hlavne vykonať dealokáciu pamäte. Názov metódy deštruktora môže byť ľubovoľný. Treba zdôrazniť, že prekladač TASM konštruktory a deštruktory nijako nepodporuje. Je iba na programátorovi, ako ich bude realizovať a ako ich bude používať. Pokiaľ sa objekty nebudú dynamicky vytvárať, nemusí objekt obsahovať nijaký konštruktor ani deštruktor.

Inicializácia

S vytvorením inštancie objektu úzko súvisí inicializácia objektu, hlavne inicializácia ukazovateľa na tabuľku virtuálnych metód. Statické objekty je možné inicializovať priamo pri vytváraní inštancie v dátovom segmente, virtuálne metódy môže inicializovať konštruktor. Najvhodnejšie je však vytvoriť pre objekt zvláštnu metódu, napr. s názvom init, ktorá potrebnú inicializáciu vykoná. Vzhľadom na to, že v čase volania tejto metódy nie je ešte inicializovaný ukazovateľ na VMT, musí byť táto metóda statická. Pretože zvlášť veľkú pozornosť treba venovať inicializácii ukazovateľa na VMT, TASM má direktívu **TBLINIT**, ktorá vykoná inicializáciu ukazovateľa na VMT pre zadanú inštanciu objektu. **TBLINIT** predpokladá, že táto inštancia je inštancia aktuálneho typu objektu, t. j. objektu, ktorý bol deklarovaný ako posledný. Syntax direktívy je:

TBLINIT inštancia

kde **inštancia** je ukazovateľ na inštanciu objektu. Môže to byť návestie inštancie objektu alebo dvojice registrov (segmentový a bázový alebo indexový), v ktorých je adresa inštancie. Pokiaľ je ukazovateľ na inštanciu typu NEAR, rozvinie sa direktíva TBLINIT do tvaru:

```
MOV [(inštancia).@Mptr_aktuányobjekt], OFFSET @TableAddr_ aktuányobjekt
```

V prípade, že je ukazovateľ typu FAR, naplní sa aj segmentová časť ukazovateľa. Nasledujúci príklad ukazuje dva možné spôsoby inicializácie ukazovateľa na VMT:

```
...
;deklarácia objektu Point
DATASEG
TBLINST     ;vytvorenie inštancie VMT pre objekt Point
MyPoint     Point {} ;vytvorenie inštancie objektu Point
YourPoint   Point {@Mptr_Point - @TableAddr_Point} ;prvý spôsob
...
CODESEG
...
TBLINIT     MyPoint ;druhý spôsob
...
```

Direktívu TBLINIT môžeme použiť aj v inicializačnej metóde:

```
...
;deklarácia objektu Point
DATASEG
TBLINST
MyPoint     Point {}
...
CODESEG
...
CALL MyPoint METHOD init PASCAL, DS OFFSET MyPoint
...

PROC        Point_init
            ARG @@Point:WORD
            USES DS,BX
            LDS BX, @@Point
            TBLINIT Point PTR DS:BX
            ;inicializácia ostatných dát objektu
            RET
ENDP
```

Použitie direktívy TBLINIT je obmedzené na inštancie aktuálneho typu objektu. Nasledujúce makro umožňuje inicializovať NEAR ukazovateľ na VMT pre objekt ľubovoľného typu.

```
MACRO TBLINITO object, instance
IF SIZE @TableAddr_&object
    MOV [(&object PTR &instance).@Mptr_&object], OFFSET @TableAddr_&object
ENDIF
ENDM
```

Príklad použitia:

```
...
TBLINITO Point, MyPoint
TBLINITO Line, DS:BX
...
```

Praktický príklad

Ukážeme si jednoduchý príklad, na ktorom vidieť praktickú realizáciu programu využívajúceho objekty. Program bude obsahovať deklaráciu dvoch objektov – Point a Line. Ich deklarácia bude odlišná od príkladov v predchádzajúcom texte. Základom programu je procedúra, ktorá dostane ako parameter ukazovateľ na inštanciu niektorého z týchto dvoch objektov. Procedúra nebude vedieť, o ktorý objekt ide, napriek tomu ho zobrazí na obrazovke a bude s ním pohybovať podľa stlačených kurzorových klávesov. Samotný zdrojový kód programu je rozdelený do troch súborov. Súbor OOPTBL.INC obsahuje definície makier TBLINSTO a TBLINITO, ktoré sú v programe používané. Súbor OOPDEMO.ASO obsahuje deklarácie objektov Point a Line. V súbore OOPDEMO.ASM sa nachádza vlastný program a procedúry metód objektov. Program po spustení zobrazí na obrazovke malý štvorček, ktorý reprezentuje bod, po stlačení klávesu ESC sa zobrazí úsečka. Pokúsil som sa do programu pridať časť, ktorá zabezpečuje zväčšenie resp. zmenšenie úsečky, pomocou klávesov '+' a '-' na numerickej klávesnici. Vy sa pokúste túto vlastnosť implemetovať do objektov Point, resp. Line, prípadne sa zamyslite nad realizáciou ďalšieho objektu, napr. štvorca, ktorý by mal stranu a = 10 bodov, pretože len vlastným skúmaním a testovaním zistíte, ako to všetko funguje. Funkčnosť programu bola testovaná v DOS okne programu Windows 95.

Zdrojový kód programu, ako aj preloženú verziu môžete získať na WWW stránke časopisu PC REVUE (http://www.pcrevue.sk), súbor s názvom oopasm.zip.

Záver (zhodnotenie seriálu)

Ťažko povedať, či je v dnešnej dobe ešte potrebné zaoberať sa takým jazykom ako assembler. Veď len taký nástroj ako Delphi umožňuje oveľa rýchlejší vývoj aplikácií. Treba však zdôrazniť, že programátori napr. nástroja Delphi akosi pozabudli na dĺžku aplikácií. Uvediem niekoľko kladov a záporov assemblera, podľa ktorých sa sami môžete rozhodnúť, či ho použijete.

☺ Klady
- malý výsledný kód programu
- programovanie na najnižšej úrovni, t. j. v jazyku procesora
- rýchlosť programov

☹ Zápory
- v niektorých prípadoch veľký zdrojový kód programu
- rastie čas potrebný na vývoj aplikácie
- potrebujete veľa informácií o danom operačnom systéme

Aj keď je assembler taký, aký je, určite vždy nájde použitie pri programovaní kritických častí programu, ktoré majú byť superrýchle, prípade pri tvorbe ovládačov zariadení a pod. Assembler je tiež možné kombinovať s rôznymi jazykmi, napr. Visual C++, Deplhi a pod.

Niekoľko slov k seriálu. Za tie dva roky, čo tento seriál píšem, sa ku mne dostala okrem iného aj informácia, že sa používa ako študijný materiál na vyučovanie assemblera, čo ma, úprimne poviem, teší, aspoň to malo svoj zmysel.

Ak vás zaujala niektorá téma, podrobnejšie informácie treba hľadať v literatúre (napr. v tej, ktorú som takmer vždy za článkom uvádzal), prípadne na sieti internet. Ak budem niekedy v budúcnosti podobný seriál na túto tému pripravovať, bude sa výlučne týkať assemblera pod Windows.

Literatúra:

[1] Ľubor Šešera – Aleš Mičovský: Objektovo-orientovaná tvorba systémov a jazyk C++, Vydavateľstvo PERFEKT, Bratislava 1994.
[2] Ondrej Macko: Turbo Vision – praktický sprievodca. Vydavateľstvo GRADA 1992.
[3] Petr Cápek-Juraj Rojko-Martin Vogel-Jiří Voves: Turbo Assembler 3.0. Vydavateľstvo GRADA 1992.

Peter Gašparovič