

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МО ЭВМ**

**ОТЧЕТ**  
**по лабораторной работе №3**  
**по дисциплине «Операционные системы»**  
**Тема: «Исследование организации управления основной памятью»**

Студент гр. 7381

Минуллин М.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2019

### **Цель работы.**

Для исследования организации управления памятью необходимо ориентироваться на тип основной памяти, реализованной в компьютере и способ организации, принятый в ОС. В лабораторной работе рассматривается нестраничная память и способ управления динамическими разделами. Для реализации управления памятью в этом случае строится список занятых и свободных участков памяти. Функции ядра, обеспечивающие управление основной памятью, просматривают и преобразуют этот список.

В лабораторной работе исследуются структуры данных и работа функций управления памятью ядра операционной системы.

### **Необходимые сведения для составления программы.**

Учёт занятой и свободной памяти ведётся при помощи списка блоков управления памятью MCB (Memory Control Block). MCB занимает 16 байт (параграф) и располагается всегда с адреса, кратного 16 (адрес сегмента ОП) и находится в адресном пространстве непосредственно перед тем участком памяти, которым он управляет.

MCB имеет структуру, представленную в табл. 1.

Таблица 1 – Структура Memory Control Block.

Смещение	Длина поля (байты)	Содержимое поля
00h	1	Тип MCB: 5Ah, если последний в списке; 4Dh, если не последний.
01h	2	Сегментный адрес PSP владельца участка памяти, либо: 0000h – свободный участок; 0006h – участок принадлежит драйверу OS XMS UMB; 0007h – участок является исключённой верхней памятью драйверов; 0008h – участок принадлежит MS DOS; FFFAh – участок занят управляющим блоком 386MAX UMB; FFFDh – участок заблокирован 386MAX; FFFEh – участок принадлежит 386MAX UMB;
03h	2	Размер участка в параграфах

05h	3	Зарезервирован
08h	8	«SC», если участок принадлежит MS DOS, то в нём системный код; «SD», если участок принадлежит MS DOS, то в нём системные данные.

По сегментному адресу и размеру участка памяти, контролируемого этим MCB можно определить местоположение следующего MCB в списке.

Адрес первого MCB хранится во внутренней структуре MS DOS, называемой «List of Lists» (список списков). Доступ к указателю на эту структуру можно получить, используя функцию 52h – «Get List of Lists» int 21h. В результате выполнения этой функции ES:BX будет указывать на список списков. Слово по адресу ES:[BX-2] и есть адрес самого первого MCB.

Размер расширенной памяти находится в ячейках 30h, 31h CMOS. CMOS – это энергозависимая память, в которой хранится информация о конфигурации ПЭВМ. Объём памяти составляет 64 байта. Размер расширенной памяти в Кбайтах можно определить, обращаясь к ячейкам CMOS определённым образом.

### Ход работы.

Написан и отлажен программный .COM модуль, который определяется и распечатывает информацию о количестве доступной памяти, размере расширенной памяти и блоках управления памятью. Результат работы модуля представлен на рис. 1.

```

Available memory (B): 648912
Extended memory (KB): 15360
| MCB Type | PSP Address | Size | SC/SD |
|-----|-----|-----|-----|
| 4D       | 0008       | 16   |       |
| 4D       | 0000       | 64   | DPMILOAD
| 4D       | 0040       | 256  |
| 4D       | 0192       | 144  |
| 5A       | 0192       | 648912 | LAB3_1

```

Рисунок 1 – Первый программный модуль.

Первый модуль модифицирован таким образом, чтобы программа освобождала память, которую она не занимает, используя функцию 4Ah

прерывания 21h. Результатом стал второй модуль, создающий новый блок, обозначаемый как пустой. Результаты можно увидеть на рис. 2.

```
Available memory (B): 648912
Extended memory (KB): 15360
| MCB Type | PSP Address | Size | SC/SD |
| 4D       | 0008       | 16  |       |
| 4D       | 0000       | 64  | DPMILOAD
| 4D       | 0040       | 256 |
| 4D       | 0192       | 144 |
| 4D       | 0192       | 13232 | LAB3_2
| 5A       | 0000       | 635664
```

Рисунок 2 – Второй программный модуль.

Второй модуль модифицирован таким образом, что после выполнения он запрашивает 64 Кбайта памяти. Для этого используется функция 48h прерывания 21h. Результатом выступил новый участок памяти, указанного размера (см. рис. 3).

```
Available memory (B): 648912
Extended memory (KB): 15360
| MCB Type | PSP Address | Size | SC/SD |
| 4D       | 0008       | 16  |
| 4D       | 0000       | 64  | DPMILOAD
| 4D       | 0040       | 256 |
| 4D       | 0192       | 144 |
| 4D       | 0192       | 13344 | LAB3_3
| 4D       | 0192       | 65536 | LAB3_3
| 5A       | 0000       | 570000 | â†“â†“â†“â†“
```

Рисунок 3 – Третий программный модуль.

Четвёртый программный модуль запрашивает 64 Кбайта памяти до освобождения памяти. Возникает ошибка, сопровождаемая сообщением о том, что память уже была выделена программе и выделение ещё 64 Кбайт память невозможно (результат представлен на рис. 4).

```
Available memory (B): 648912
Extended memory (KB): 15360
Memory allocation error
| MCB Type | PSP Address | Size | SC/SD |
| 4D       | 0008       | 16  |
| 4D       | 0000       | 64  | DPMILOAD
| 4D       | 0040       | 256 |
| 4D       | 0192       | 144 |
| 4D       | 0192       | 13888 | LAB3_4
| 5A       | 0000       | 635008
```

Рисунок 4 – Четвёртый программный модуль.

### **Контрольные вопросы.**

В: Что означает «доступный объём памяти»?

О: Доступный объём памяти – максимальный объём памяти, выделенный программе операционной системой.

В: Где МСВ блок Вашей программы в списке?

О: В первой программе МСВ находится в предпоследней строчке списка размером в 144 байта. Во второй программе блок расположен в предпоследней строчке списка. Последнюю строчку занимает блок, обозначенный, как пустой участок. В третьей программе МСВ находится в пятой строчке списка. После него находятся блок памяти, выделенной по запросу и свободный блок памяти. В четвёртой программе блок МСВ расположен в предпоследней строчке списка. Последнюю строку занимает блок, обозначенный как пустой участок.

В: Какой размер памяти занимает программа в каждом случае?

О: В первом случае: 648912 байт. Во втором случае:  $648912 - 635664 - 16 = 13232$  байт. В третьем случае:  $648912 - 570000 - 65536 - 2 \times 16 = 13344$  байт. В четвёртом случае:  $648912 - 635008 - 16 = 13888$  байт.

### **Выводы.**

В ходе выполнения лабораторной работы были исследованы организация управления основной памятью, структуры данных и работа функций управления памятью ядра операционной системы.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ ТЕКСТ ПЕРВОГО МОДУЛЯ

```
testpc segment
    assume cs:testpc, ds: testpc, es:nothing, ss:nothing

org 100h

start: jmp begin

AVAILABLE_MEM    db 'Available memory (B):      ', 10, 13, '$'
EXTENDED_MEM     db 'Extended memory (KB):      ', 10, 13, '$'
TABLE_TITLE      db '| MCB Type | PSP Address | Size | SC/SD |',
10, 13, '$'
TABLE_MCB_DATA   db '
', 10, 13, '$'

PRINT            proc    near
    push    ax
    mov     ah, 09h
    int     21h
    pop     ax
    ret
PRINT            endp

TETR_TO_HEX      proc    near
    and     al,0Fh
    cmp     al,09
    jbe     NEXT
    add     al,07
NEXT:
    add     al,30h
    ret
TETR_TO_HEX      endp

BYTE_TO_HEX      proc    near
    push    cx
    mov     ah, al
    call    TETR_TO_HEX
    xchg    al, ah
    mov     cl, 4
    shr     al, cl
    call    TETR_TO_HEX
    pop     cx
    ret
BYTE_TO_HEX      endp

WRD_TO_HEX       proc    near
```

```

    push    BX
    mov     BH,ah
    call    BYTE_TO_HEX
    mov     [di], ah
    dec     di
    mov     [di], al
    dec     di
    mov     al, BH
    call    BYTE_TO_HEX
    mov     [di], ah
    dec     di
    mov     [di], al
    pop     BX
    ret
WRD_TO_HEX    endp

BYTE_TO_DEC   proc near
    push    cx
    push    dx
    xor     ah, ah
    xor     dx, dx
    mov     cx, 10
loop_bd:
    div     cx
    or      dl, 30h
    mov     [si], dl
    dec     si
    xor     dx, dx
    cmp     ax, 10
    jae     loop_bd
    cmp     al, 00h
    je      end_l
    or      al, 30h
    mov     [si], al
end_l:
    pop     dx
    pop     cx
    ret
BYTE_TO_DEC   endp

WRD_TO_DEC    proc      near
    push    cx
    push    dx
    mov     cx, 10
loop_b:
    div     cx
    or      dl, 30h
    mov     [si], dl
    dec     si
    xor     dx, dx
    cmp     ax, 10

```

```

        jae    loop_b
        cmp    al, 00h
        je     endl
        or     al, 30h
        mov    [si], al
endl:
        pop    dx
        pop    cx
        ret
WRD_TO_DEC  endp

GET_AVAILABLE_MEM proc  near
        push    ax
        push    bx
        push    dx
        push    si
        xor     ax, ax
        mov     ah, 04Ah
        mov     bx, 0FFFFh
        int     21h
        mov     ax, 10h
        mul     bx
        mov     si, offset AVAILABLE_MEM
        add     si, 27
        call    WRD_TO_DEC
        mov     dx, offset AVAILABLE_MEM
        call    PRINT
        pop     si
        pop     dx
        pop     bx
        pop     ax
        ret
GET_AVAILABLE_MEM  endp

GET_EXTENDED_MEM  proc  near
        push    ax
        push    bx
        push    dx
        push    si
        xor     dx, dx
        mov     al, 30h
        out     70h, al
        in      al, 71h
        mov     bl, al
        mov     al, 31h
        out     70h, al
        in      al, 71h
        mov     ah, al
        mov     al, bl
        mov     si, offset EXTENDED_MEM
        add     si, 26

```



```

    call    WRD_TO_DEC
    mov     dx, offset EXTENDED_MEM
    call    PRINT
    pop     si
    pop     dx
    pop     bx
    pop     ax
    ret
GET_EXTENDED_MEM endp

GET_MCB_TYPE proc near
    push    ax
    push    di
    mov     di, offset TABLE_MCB_DATA
    add     di, 5
    xor     ah, ah
    mov     al, es:[00h]
    call    BYTE_TO_HEX
    mov     [di], al
    inc     di
    mov     [di], ah
    pop     di
    pop     ax
    ret
GET_MCB_TYPE endp

GET_PSP_ADDRESS proc near
    push    ax
    push    di
    mov     di, offset TABLE_MCB_DATA
    mov     ax, es:[01h]
    add     di, 19
    call    WRD_TO_HEX
    pop     di
    pop     ax
    ret
GET_PSP_ADDRESS endp

GET_MCB_SIZE proc near
    push    ax
    push    bx
    push    di
    push    si
    mov     di, offset TABLE_MCB_DATA
    mov     ax, es:[03h]
    mov     bx, 10h
    mul     bx
    add     di, 29
    mov     si, di
    call    WRD_TO_DEC
    pop     si

```

```

        pop    di
        pop    bx
        pop    ax
        ret
GET_MCB_SIZE endp

GET_SC_SD proc near
    push    bx
    push    dx
    push    di
    mov     di, offset TABLE_MCB_DATA
    add     di, 33
    mov     bx, 0h
GET_8_BYTES:
    mov     dl, es:[bx + 8]
    mov     [di], dl
    inc     di
    inc     bx
    cmp     bx, 8h
    jne     GET_8_BYTES
    pop     di
    pop     dx
    pop     bx
    ret
GET_SC_SD endp

GET_MCB_DATA proc near
    mov     ah, 52h
    int     21h
    sub     bx, 2h
    mov     es, es:[bx]
FOR_EACH_MCB:
    call    GET_MCB_TYPE
    call    GET_PSP_ADDRESS
    call    GET_MCB_SIZE
    call    GET_SC_SD
    mov     ax, es:[03h]
    mov     bl, es:[00h]
    mov     dx, offset TABLE_MCB_DATA
    call    PRINT
    mov     cx, es
    add     ax, cx
    inc     ax
    mov     es, ax
    cmp     bl, 4Dh
    je      FOR_EACH_MCB

    ret
GET_MCB_DATA endp

```

```
begin:
```

```
    call    GET_AVAILABLE_MEM
    call    GET_EXTENDED_MEM
    mov     dx, offset TABLE_TITLE
    call    PRINT
    call    GET_MCB_DATA
    xor     al, al
    mov     ah, 4ch
    int     21h
testpc ends

end start
```

## ПРИЛОЖЕНИЕ Б

### ИСХОДНЫЙ ТЕКСТ ВТОРОГО МОДУЛЯ

```
testpc segment
    assume cs:testpc, ds: testpc, es:nothing, ss:nothing

org 100h

start: jmp begin

AVAILABLE_MEM    db 'Available memory (B):          ', 10, 13, '$'
EXTENDED_MEM     db 'Extended memory (KB):          ', 10, 13, '$'
TABLE_TITLE      db '| MCB Type | PSP Address | Size | SC/SD |', 10,
13, '$'
TABLE_MCB_DATA   db '
', 10, 13, '$'

PRINT    proc    near
    push    ax
    mov     ah, 09h
    int     21h
    pop     ax
    ret
PRINT    endp

TETR_TO_HEX      proc    near
    and     al, 0Fh
    cmp     al, 09
    jbe     NEXT
    add     al, 07
NEXT:
    add     al, 30h
    ret
TETR_TO_HEX      endp

BYTE_TO_HEX      proc    near
    push    cx
    mov     ah, al
    call    TETR_TO_HEX
    xchg    al, ah
    mov     cl, 4
    shr     al, cl
    call    TETR_TO_HEX
    pop     cx
    ret
BYTE_TO_HEX      endp

WRD_TO_HEX      proc    near
```

```

    push    bx
    mov     bh, ah
    call    BYTE_TO_HEX
    mov     [di], ah
    dec     di
    mov     [di], al
    dec     di
    mov     al, bh
    call    BYTE_TO_HEX
    mov     [di], ah
    dec     di
    mov     [di], al
    pop     bx
    ret
WRD_TO_HEX    endp

BYTE_TO_DEC    proc    near
    push    cx
    push    dx
    xor     ah, ah
    xor     dx, dx
    mov     cx, 10
loop_bd:
    div     cx
    or      dl, 30h
    mov     [si], dl
    dec     si
    xor     dx, dx
    cmp     ax, 10
    jae     loop_bd
    cmp     al, 00h
    je      end_l
    or      al, 30h
    mov     [si], al
end_l:
    pop     dx
    pop     cx
    ret
BYTE_TO_DEC    endp

WRD_TO_DEC    proc    near
    push    cx
    push    dx
    mov     cx, 10
loop_b:
    div     cx
    or      dl, 30h
    mov     [si], dl
    dec     si
    xor     dx, dx
    cmp     ax, 10

```

```

        jae    loop_b
        cmp    al, 00h
        je     endl
        or     al, 30h
        mov    [si], al
endl:
        pop    dx
        pop    cx
        ret
WRD_TO_DEC  endp

GET_AVAILABLE_MEM  proc    near
        push    ax
        push    bx
        push    dx
        push    si
        xor     ax, ax
        mov     ah, 04Ah
        mov     bx, 0FFFFh
        int     21h
        mov     ax, 10h
        mul     bx
        mov     si, offset AVAILABLE_MEM
        add     si, 27
        call    WRD_TO_DEC
        mov     dx, offset AVAILABLE_MEM
        call    PRINT
        pop     si
        pop     dx
        pop     bx
        pop     ax
        ret
GET_AVAILABLE_MEM  endp

GET_EXTENDED_MEM  proc  near
        push    ax
        push    bx
        push    dx
        push    si
        xor     dx, dx
        mov     al, 30h
        out     70h, al
        in      al, 71h
        mov     bl, al
        mov     al, 31h
        out     70h, al
        in      al, 71h
        mov     ah, al
        mov     al, bl
        mov     si, offset EXTENDED_MEM
        add     si, 26

```

```

        call    WRD_TO_DEC
        mov     dx, offset EXTENDED_MEM
        call    PRINT
        pop     si
        pop     dx
        pop     bx
        pop     ax
        ret
GET_EXTENDED_MEM endp

GET_MCB_TYPE    proc    near
        push    ax
        push    di
        mov     di, offset TABLE_MCB_DATA
        add     di, 5
        xor     ah, ah
        mov     al, es:[00h]
        call    BYTE_TO_HEX
        mov     [di], al
        inc     di
        mov     [di], ah
        pop     di
        pop     ax
        ret
GET_MCB_TYPE    endp

GET_PSP_ADDRESS proc    near
        push    ax
        push    di
        mov     di, offset TABLE_MCB_DATA
        mov     ax, es:[01h]
        add     di, 19
        call    WRD_TO_HEX
        pop     di
        pop     ax
        ret
GET_PSP_ADDRESS endp

GET_MCB_SIZE     proc    near
        push    ax
        push    bx
        push    di
        push    si
        mov     di, offset TABLE_MCB_DATA
        mov     ax, es:[03h]
        mov     bx, 10h
        mul     bx
        add     di, 29
        mov     si, di
        call    WRD_TO_DEC
        pop     si

```

```

        pop     di
        pop     bx
        pop     ax
        ret
GET_MCB_SIZE     endp

GET_SC_SD        proc     near
        push    bx
        push    dx
        push    di
        mov     di, offset TABLE_MCB_DATA
        add     di, 33
        mov     bx, 0h
GET_8_BYTES:
        mov     dl, es:[bx + 8]
        mov     [di], dl
        inc     di
        inc     bx
        cmp     bx, 8h
        jne     GET_8_BYTES
        pop     di
        pop     dx
        pop     bx
        ret
GET_SC_SD        endp

GET_MCB_DATA     proc     near
        mov     ah, 52h
        int     21h
        sub     bx, 2h
        mov     es, es:[bx]

FOR_EACH_MCB:
        call    GET_MCB_TYPE
        call    GET_PSP_ADDRESS
        call    GET_MCB_SIZE
        call    GET_SC_SD
        mov     ax, es:[03h]
        mov     bl, es:[00h]
        mov     dx, offset TABLE_MCB_DATA
        call    PRINT
        mov     cx, es
        add     ax, cx
        inc     ax
        mov     es, ax
        cmp     bl, 4Dh
        je      FOR_EACH_MCB
        xor     al, al
        mov     ah, 4ch
        int     21h
GET_MCB_DATA     endp

```



```
begin:
    call    GET_AVAILABLE_MEM
    call    GET_EXTENDED_MEM
    mov     ah, 4ah
    mov     bx, offset END_OF_PROGRAMM
    int     21h
    mov     dx, offset TABLE_TITLE
    call    PRINT
    call    GET_MCB_DATA
    xor     al, al
    mov     ah, 4Ch
    int     21h
    END_OF_PROGRAMM db 0
testpc ends

end start
```

## ПРИЛОЖЕНИЕ В

### ИСХОДНЫЙ ТЕКСТ ТРЕТЬЕГО МОДУЛЯ

```
testpc segment
    assume cs:testpc, ds: testpc, es:nothing, ss:nothing

org 100h

start: jmp begin

AVAILABLE_MEM    db 'Available memory (B):          ', 10, 13, '$'
EXTENDED_MEM     db 'Extended memory (KB):          ', 10, 13, '$'
TABLE_TITLE      db '| MCB Type | PSP Address | Size | SC/SD |', 10,
13, '$'
TABLE_MCB_DATA   db '
', 10, 13, '$'

PRINT    proc    near
    push    ax
    mov     ah, 09h
    int     21h
    pop     ax
    ret
PRINT    endp

TETR_TO_HEX      proc    near
    and     al, 0Fh
    cmp     al, 09
    jbe     NEXT
    add     al, 07
NEXT:
    add     al, 30h
    ret
TETR_TO_HEX      endp

BYTE_TO_HEX      proc    near
    push    cx
    mov     ah, al
    call    TETR_TO_HEX
    xchg    al, ah
    mov     cl, 4
    shr     al, cl
    call    TETR_TO_HEX
    pop     cx
    ret
BYTE_TO_HEX      endp

WRD_TO_HEX      proc    near
```

```

    push    bx
    mov     bh, ah
    call    BYTE_TO_HEX
    mov     [di], ah
    dec     di
    mov     [di], al
    dec     di
    mov     al, bh
    call    BYTE_TO_HEX
    mov     [di], ah
    dec     di
    mov     [di], al
    pop     bx
    ret
WRD_TO_HEX endp

```

```

BYTE_TO_DEC proc near
    push    cx
    push    dx
    xor     ah, ah
    xor     dx, dx
    mov     cx, 10
loop_bd:
    div     cx
    or      dl, 30h
    mov     [si], dl
    dec     si
    xor     dx, dx
    cmp     ax, 10
    jae     loop_bd
    cmp     al, 00h
    je      end_l
    or      al, 30h
    mov     [si], al
end_l:
    pop     dx
    pop     cx
    ret
BYTE_TO_DEC endp

```

```

WRD_TO_DEC proc near
    push    cx
    push    dx
    mov     cx, 10
loop_b:
    div     cx
    or      dl, 30h
    mov     [si], dl
    dec     si
    xor     dx, dx
    cmp     ax, 10

```

```

        jae    loop_b
        cmp    al, 00h
        je     endl
        or     al, 30h
        mov    [si], al
endl:
        pop    dx
        pop    cx
        ret
WRD_TO_DEC endp

GET_AVAILABLE_MEM    proc    near
        push    ax
        push    bx
        push    dx
        push    si
        xor     ax, ax
        mov     ah, 04Ah
        mov     bx, 0FFFFh
        int     21h
        mov     ax, 10h
        mul     bx
        mov     si, offset AVAILABLE_MEM
        add     si, 27
        call    WRD_TO_DEC
        mov     dx, offset AVAILABLE_MEM
        call    PRINT
        pop     si
        pop     dx
        pop     bx
        pop     ax
        ret
GET_AVAILABLE_MEM    endp

GET_EXTENDED_MEM    proc    near
        push    ax
        push    bx
        push    dx
        push    si
        xor     dx, dx
        mov     al, 30h
        out     70h, al
        in      al, 71h
        mov     bl, al
        mov     al, 31h
        out     70h, al
        in      al, 71h
        mov     ah, al
        mov     al, bl
        mov     si, offset EXTENDED_MEM
        add     si, 26

```

```

    call    WRD_TO_DEC
    mov     dx, offset EXTENDED_MEM
    call    PRINT
    pop     si
    pop     dx
    pop     bx
    pop     ax
    ret
GET_EXTENDED_MEM    endp

GET_MCB_TYPE        proc    near
    push    ax
    push    di
    mov     di, offset TABLE_MCB_DATA
    add     di, 5
    xor     ah, ah
    mov     al, es:[00h]
    call    BYTE_TO_HEX
    mov     [di], al
    inc     di
    mov     [di], ah
    pop     di
    pop     ax
    ret
GET_MCB_TYPE        endp

GET_PSP_ADDRESS     proc    near
    push    ax
    push    di
    mov     di, offset TABLE_MCB_DATA
    mov     ax, es:[01h]
    add     di, 19
    call    WRD_TO_HEX
    pop     di
    pop     ax
    ret
GET_PSP_ADDRESS     endp

GET_MCB_SIZE        proc    near
    push    ax
    push    bx
    push    di
    push    si
    mov     di, offset TABLE_MCB_DATA
    mov     ax, es:[03h]
    mov     bx, 10h
    mul     bx
    add     di, 29
    mov     si, di
    call    WRD_TO_DEC
    pop     si

```

```

        pop    di
        pop    bx
        pop    ax
        ret
GET_MCB_SIZE endp

GET_SC_SD    proc    near
        push    bx
        push    dx
        push    di
        mov     di, offset TABLE_MCB_DATA
        add     di, 33
        mov     bx, 0h
GET_8_BYTES:
        mov     dl, es:[bx + 8]
        mov     [di], dl
        inc     di
        inc     bx
        cmp     bx, 8h
        jne     GET_8_BYTES
        pop     di
        pop     dx
        pop     bx
        ret
GET_SC_SD    endp

GET_MCB_DATA    proc    near
        mov     ah, 52h
        int     21h
        sub     bx, 2h
        mov     es, es:[bx]
FOR_EACH_MCB:
        call    GET_MCB_TYPE
        call    GET_PSP_ADDRESS
        call    GET_MCB_SIZE
        call    GET_SC_SD
        mov     ax, es:[03h]
        mov     bl, es:[00h]
        mov     dx, offset TABLE_MCB_DATA
        call    PRINT
        mov     cx, es
        add     ax, cx
        inc     ax
        mov     es, ax
        cmp     bl, 4Dh
        je     FOR_EACH_MCB
        xor     al, al
        mov     ah, 4ch
        int     21h
GET_MCB_DATA    endp

```

```

begin:
    call    GET_AVAILABLE_MEM
    call    GET_EXTENDED_MEM
    mov     ah, 4ah
    mov     bx, offset END_OF_PROGRAMM
    int     21h
    mov     ah, 48h
    mov     bx, 1000h
    int     21h
    mov     dx, offset TABLE_TITLE
    call    PRINT
    call    GET_MCB_DATA
    xor     al, al
    mov     ah, 4Ch
    int     21h
    END_OF_PROGRAMM db 0
testpc ends

end start

```

## ПРИЛОЖЕНИЕ Г

### ИСХОДНЫЙ ТЕКСТ ЧЕТВЁРТОГО МОДУЛЯ

```
testpc segment
    assume cs:testpc, ds: testpc, es:nothing, ss:nothing

org 100h

start: jmp begin

AVAILABLE_MEM    db 'Available memory (B):          ', 10, 13, '$'
EXTENDED_MEM     db 'Extended memory (KB):          ', 10, 13, '$'
TABLE_TITLE      db '| MCB Type | PSP Address | Size | SC/SD |', 10,
13, '$'
TABLE_MCB_DATA   db '
', 10, 13, '$'
ERROR_MEM        db 'Memory allocation error', 10, 13, '$'

PRINT    proc    near
    push    ax
    mov     ah, 09h
    int     21h
    pop     ax
    ret
PRINT    endp

TETR_TO_HEX      proc    near
    and     al, 0Fh
    cmp     al, 09
    jbe     NEXT
    add     al, 07
NEXT:
    add     al, 30h
    ret
TETR_TO_HEX      endp

BYTE_TO_HEX      proc    near
    push    cx
    mov     ah, al
    call    TETR_TO_HEX
    xchg    al, ah
    mov     cl, 4
    shr     al, cl
    call    TETR_TO_HEX
    pop     cx
    ret
BYTE_TO_HEX      endp
```



```

WRD_TO_HEX    proc      near
    push      bx
    mov       bh, ah
    call      BYTE_TO_HEX
    mov       [di], ah
    dec       di
    mov       [di], al
    dec       di
    mov       al, bh
    call      BYTE_TO_HEX
    mov       [di], ah
    dec       di
    mov       [di], al
    pop       bx
    ret
WRD_TO_HEX    endp

```

```

BYTE_TO_DEC   proc      near
    push      cx
    push      dx
    xor       ah, ah
    xor       dx, dx
    mov       cx, 10
loop_bd:
    div       cx
    or        dl, 30h
    mov       [si], dl
    dec       si
    xor       dx, dx
    cmp       ax, 10
    jae       loop_bd
    cmp       al, 00h
    je        end_l
    or        al, 30h
    mov       [si], al
end_l:
    pop       dx
    pop       cx
    ret
BYTE_TO_DEC   endp

```

```

WRD_TO_DEC    proc      near
    push      cx
    push      dx
    mov       cx, 10
loop_b:
    div       cx
    or        dl, 30h
    mov       [si], dl
    dec       si
    xor       dx, dx

```

```

        cmp     ax, 10
        jae     loop_b
        cmp     al, 00h
        je      endl
        or      al, 30h
        mov     [si], al
endl:
        pop     dx
        pop     cx
        ret
WRD_TO_DEC endp

GET_AVAILABLE_MEM    proc    near
    push     ax
    push     bx
    push     dx
    push     si
    xor      ax, ax
    mov      ah, 04Ah
    mov      bx, 0FFFFh
    int      21h
    mov      ax, 10h
    mul      bx
    mov      si, offset AVAILABLE_MEM
    add      si, 27
    call     WRD_TO_DEC
    mov      dx, offset AVAILABLE_MEM
    call     PRINT
    pop      si
    pop      dx
    pop      bx
    pop      ax
    ret
GET_AVAILABLE_MEM    endp

GET_EXTENDED_MEM     proc    near
    push     ax
    push     bx
    push     dx
    push     si
    xor      dx, dx
    mov      al, 30h
    out      70h, al
    in       al, 71h
    mov      bl, al
    mov      al, 31h
    out      70h, al
    in       al, 71h
    mov      ah, al
    mov      al, bl
    mov      si, offset EXTENDED_MEM

```

```

        add     si, 26
        call    WRD_TO_DEC
        mov     dx, offset EXTENDED_MEM
        call    PRINT
        pop     si
        pop     dx
        pop     bx
        pop     ax
        ret
GET_EXTENDED_MEM    endp

GET_MCB_TYPE        proc    near
        push    ax
        push    di
        mov     di, offset TABLE_MCB_DATA
        add     di, 5
        xor     ah, ah
        mov     al, es:[00h]
        call    BYTE_TO_HEX
        mov     [di], al
        inc     di
        mov     [di], ah
        pop     di
        pop     ax
        ret
GET_MCB_TYPE        endp

GET_PSP_ADDRESS     proc    near
        push    ax
        push    di
        mov     di, offset TABLE_MCB_DATA
        mov     ax, es:[01h]
        add     di, 19
        call    WRD_TO_HEX
        pop     di
        pop     ax
        ret
GET_PSP_ADDRESS     endp

GET_MCB_SIZE        proc    near
        push    ax
        push    bx
        push    di
        push    si
        mov     di, offset TABLE_MCB_DATA
        mov     ax, es:[03h]
        mov     bx, 10h
        mul     bx
        add     di, 29
        mov     si, di
        call    WRD_TO_DEC

```

```

        pop     si
        pop     di
        pop     bx
        pop     ax
        ret
GET_MCB_SIZE      endp

GET_SC_SD        proc      near
        push    bx
        push    dx
        push    di
        mov     di, offset TABLE_MCB_DATA
        add     di, 33
        mov     bx, 0h
GET_8_BYTES:
        mov     dl, es:[bx + 8]
        mov     [di], dl
        inc     di
        inc     bx
        cmp     bx, 8h
        jne     GET_8_BYTES
        pop     di
        pop     dx
        pop     bx
        ret
GET_SC_SD        endp

GET_MCB_DATA     proc      near
        mov     ah, 52h
        int     21h
        sub     bx, 2h
        mov     es, es:[bx]
FOR_EACH_MCB:
        call    GET_MCB_TYPE
        call    GET_PSP_ADDRESS
        call    GET_MCB_SIZE
        call    GET_SC_SD
        mov     ax, es:[03h]
        mov     bl, es:[00h]
        mov     dx, offset TABLE_MCB_DATA
        call    PRINT
        mov     cx, es
        add     ax, cx
        inc     ax
        mov     es, ax
        cmp     bl, 4Dh
        je      FOR_EACH_MCB
        xor     al, al
        mov     ah, 4ch
        int     21h
GET_MCB_DATA     endp

```

```

begin:
    call    GET_AVAILABLE_MEM
    call    GET_EXTENDED_MEM
    mov     ah, 48h
    mov     bx, 1000h
    int     21h
    jnc     not_error
    mov     dx, offset ERROR_MEM
    call    PRINT
not_error:
    mov     bx, offset END_OF_PROGRAMM
    mov     ah, 4ah
    int     21h
    mov     dx, offset TABLE_TITLE
    call    PRINT
    call    GET_MCB_DATA
    xor     al, al
    mov     ah, 4Ch
    int     21h
    END_OF_PROGRAMM db 0
testpc ends

end start

```