**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.

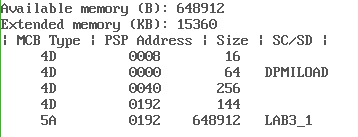


Рисунок 1 – Первый программный модуль.

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.

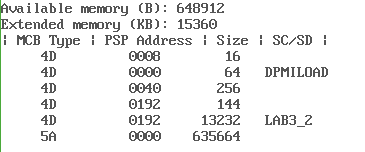


Рисунок 2 – Второй программный модуль.

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).

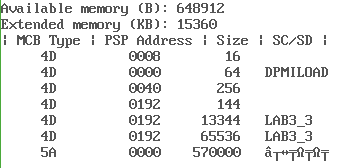


Рисунок 3 – Третий программный модуль.

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).

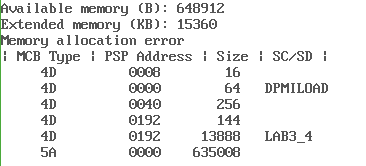


Рисунок 4 – Четвёртый программный модуль.

**Контрольные вопросы.**

В: Что означает «доступный объём памяти»?

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

В: Где MCB блок Вашей программы в списке?

О: В первой программе MCB находится в конце списка. Во второй программе блок расположен в предпоследней строчке списка. Последнюю строчку занимает блок, обозначенный, как пустой участок. В третьей программе MCB находится в пятой строчке списка. После него находятся блок памяти, выделенной по запросу и свободный блок памяти. В четвёртой программе блок MCB 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: байт. Во втором случае: байт. В третьем случае: байт. В четвёртом случае: 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