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. 9383 _____ Моисейченко К. А.

Преподаватель _____ Ефремов М. А.

Санкт-Петербург

2021

Цель работы.

Рассмотреть нестраничную память и способ управления динамическими разделами. Исследование структуры данных и работы функций управления памятью ядра операционной системы

Задание.

Шаг 1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .COM, который выбирает и распечатывает следующую информацию:

1) Количество доступной памяти.

2) Размер расширенной памяти.

3) Выводит цепочку блоков управления памятью.

Адреса при выводе представляются шестнадцатеричными числами. Объем памяти функциями управления памятью выводится в параграфах. Необходимо преобразовать его в байты и выводить в виде десятичных чисел. Последние восемь байт MCB выводятся как символы, не следует преобразовывать их в шестнадцатеричные числа. Запустите программу и внимательно оцените результаты. Сохраните результаты, полученные программой, и включите их в отчет в виде скриншота.

Шаг 2. Измените программу таким образом, чтобы она освобождала память, которую она не занимает. Для этого используйте функцию 4Ah прерывания 21h (пример в разделе «Использование функции 4AH»). Повторите эксперимент, запустив модифицированную программу. Сравните выходные данные с результатами, полученными на предыдущем шаге. Сохраните результаты, полученные программой, и включите их в отчет в виде скриншота.

Шаг 3. Измените программу еще раз таким образом, чтобы после освобождения памяти, программа запрашивала 64Кб памяти функцией 48H прерывания 21H. Повторите эксперимент, запустив модифицированную программу. Сравните выходные данные с результатами, полученными на

предыдущих шагах. Сохраните результаты, полученные программой, и включите их в отчет в виде скриншота.

Шаг 4. Измените первоначальный вариант программы, запросив 64Кб памяти функцией 48H прерывания 21H до освобождения памяти. Обязательно обрабатывайте завершение функций ядра, проверяя флаг CF. Сохраните результаты, полученные программой, и включите их в отчет в виде скриншота.

Шаг 5. Оцените результаты, полученные на предыдущих шагах. Ответьте на контрольные вопросы и оформите отчет

Основные теоретические положения.

Учет занятой и свободной памяти ведется 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 следующую структуру:

Смещение	Длина поля (байт)	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 исключенной 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 нем системный код "SD" - если участок принадлежит MS DOS, то в нем 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 f52h "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 энергонезависимая память, в которой хранится информация о конфигурации ПЭВМ. Объем памяти составляет 64 байта. Размер расширенной памяти в Кбайтах можно определить обращаясь к ячейкам CMOS следующим образом:

```
mov AL,30h              ; запись адреса ячейки CMOS
out 70h,AL in AL,71h    ; чтение младшего байта
mov BL,AL; размера расширенной памяти
mov AL,31h              ; запись адреса ячейки CMOS
out 70h,AL in AL,71h    ; чтение старшего байта
                        ; размера расширенной памяти
```

Контрольные вопросы по лабораторной работе.

1) Что означает "доступный объем памяти"?

2) Где MCB блок Вашей программы в списке?

3) Какой размер памяти занимает программа в каждом случае?

Выполнение работы.

Шаг 1.

Был написан и отлажен программный модуль типа .COM, который выбирает и распечатывает следующую информацию:

1) Количество доступной памяти.

2) Размер расширенной памяти.

3) Выводит цепочку блоков управления памятью.

Адреса при выводе представляются шестнадцатеричными числами. Объем памяти функциями управления памятью выводится в виде десятичных чисел. Последние восемь байт MCB выводятся как символы.

В программе используются следующие функции и процедуры:

TETR_TO_HEX - перевод десятичной цифры в код символа, который записывается в AL.

BYTE_TO_HEX - перевод байта в число шестнадцатеричной СС и его представление в виде двух символов.

WRD_TO_HEX - перевод слова в число шестнадцатеричной СС и его представление в виде четырех символов.

BYTE_TO_DEC - перевод байта в число десятеричной СС и его представление в виде символов.

WRITE_STR - вывод строки на экран

PARAGRAPHS_TO_BYTES - перевод и запись числа шестнадцатеричной СС из регистра ax в десятеричную СС по адресу, хранящемуся в di.

MEMORY_AVAILABLE - запись количества доступной памяти.

MEMORY_EXTENDED - запись количества расширенной памяти.

MCB - печать цепочки блоков управления памятью.

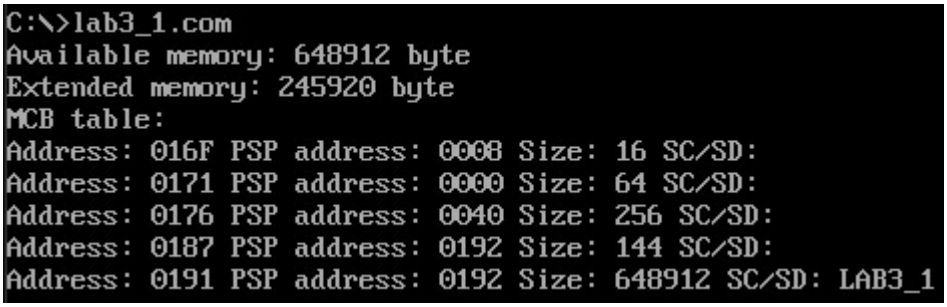

Рисунок 1 - Пример работы программы lab3_1.

Шаг 2.

Программа была изменена таким образом, чтобы она освобождала память, которую она не занимает.

Были добавлены следующие функции и процедуры:

FREE_UNUSED_MEMORY - освобождение неиспользуемой программой памяти.

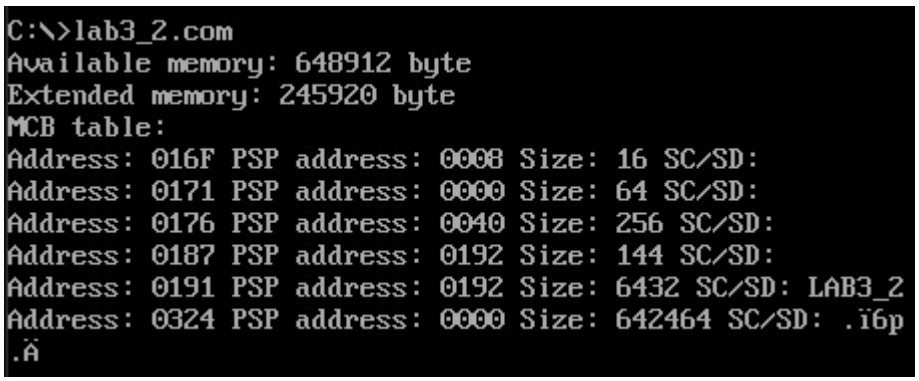

Рисунок 2 - Пример работы программы lab3_2.

Шаг 3.

Программа была изменена таким образом, чтобы после освобождения памяти, программа запрашивала 64Кб памяти функцией 48H прерывания 21H.

Были добавлены следующие функции и процедуры:

MEMORY_REQUEST - запрос 64Кб памяти, печать результата работы на экран.

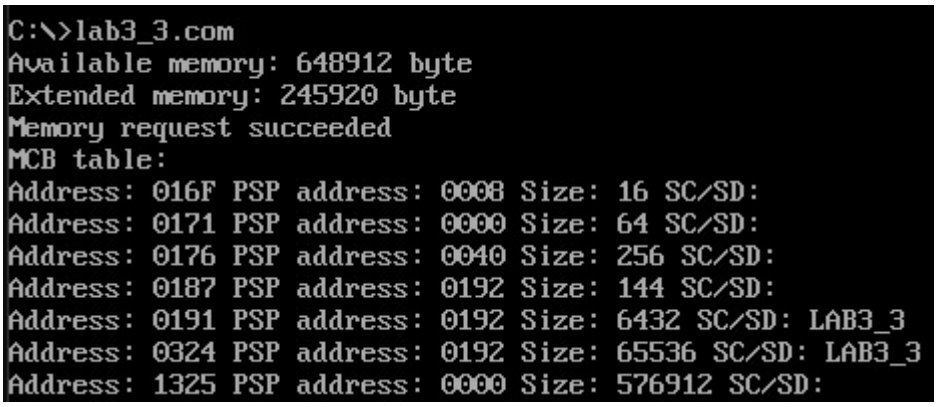

Рисунок 3 - Пример работы программы lab3_3.

Шаг 4.

Первоначальный вариант программы был изменен таким образом, чтобы программа запрашивала 64Кб памяти функцией 48H прерывания 21H до освобождения памяти.

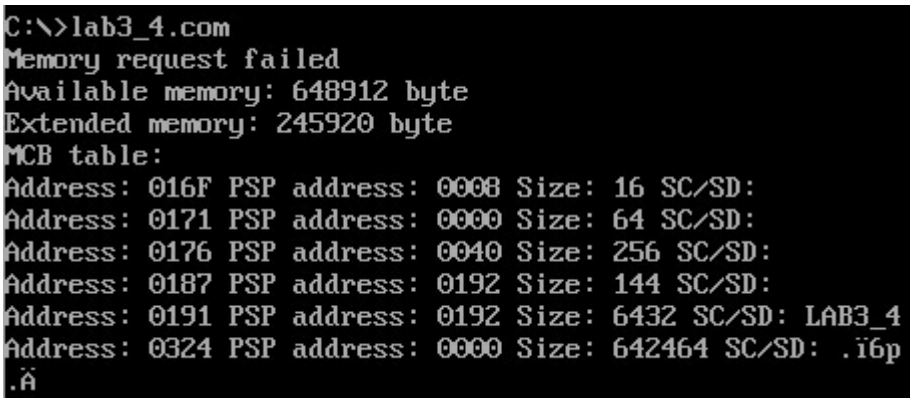

Рисунок 4 - Пример работы программы lab3_4.

Ответы на контрольные вопросы.

1) Что означает "доступный объем памяти"?

Доступный объём памяти - это область оперативной памяти, которая открыта для использования программой.

2) Где MCB блок Вашей программы в списке?

Чтобы ответить на этот вопрос обратимся к рисункам 1-4. На рисунках 1 и 2 мой MCB блок пятый в списке. На рисунке 3 - пятый и специально выделенный шестой. На рисунке 4 - пятый в списке.

3) Какой размер памяти занимает программа в каждом случае?

В первом случае программа занимает весь доступный объём памяти. Во втором случае программа занимает только необходимый объем памяти (6432 байт). В третьем случае программа занимает необходимый объём памяти и запрошенные 64Кб памяти. В четвертом случае программа занимает только необходимый объём памяти, потому что после выделения 64Кб памяти, неиспользуемая память была освобождена.

Выводы.

Были исследованы структуры данных и работа функций управления памятью ядра операционной системы.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab3_1.asm

```
TESTPC SEGMENT
ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
ORG 100H
START: JMP BEGIN
; ДАННЫЕ

AVAILABLE_MEMORY DB 'Available memory: ', '$'
EXTENDED_MEMORY DB 'Extended memory: ', '$'
STRING_BYTE DB ' byte ', '$'
MCB_TABLE DB 'MCB table: ', 0DH, 0AH, '$'
ADDRESS DB 'Address:     ', '$'
PSP_ADDRESS DB 'PSP address:      ', '$'
STRING_SIZE DB 'Size: ', '$'
SC_SD DB 'SC/SD: ', '$'
NEW_STRING DB 0DH,0AH,'$'
SPACE_STRING DB ' ', '$'

;ПРОЦЕДУРЫ
;-----------------------------------------------------
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
; байт в AL переводится в два символа шестн. числа в AX
push CX
mov AH,AL
call TETR_TO_HEX
xchg AL,AH
mov CL,4
shr AL,CL
call TETR_TO_HEX ;в AL старшая цифра
pop CX ;в AH младшая
ret
BYTE_TO_HEX ENDP
;--------------------------------
WRD_TO_HEX PROC near
;перевод в 16 с/с 16-ти разрядного числа
; в AX - число, DI - адрес последнего символа
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
; перевод в 10с/с, SI - адрес поля младшей цифры
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
;--------------------------------

WRITE_STR PROC near
push ax
mov ah,9h
int 21h
pop ax
ret
WRITE_STR ENDP

PARAGRAPHS_TO_BYTES PROC
mov bx, 0ah
xor cx, cx

division_loop:
div bx
push dx
inc cx
sub dx, dx
cmp ax, 0h
jne division_loop

write_symbol:
```

```
pop dx
add dl,30h
mov ah,02h
int 21h

loop write_symbol

ret

PARAGRAPHS_TO_BYTES ENDP


MEMORY_AVAILABLE PROC near
mov dx, offset AVAILABLE_MEMORY
call WRITE_STR
mov ah, 4ah
mov bx, 0ffffh
int 21h
mov ax, bx
mov bx, 16
mul bx
call PARAGRAPHS_TO_BYTES

mov dx, offset STRING_BYTE
call WRITE_STR

mov dx, offset NEW_STRING
call WRITE_STR

ret
MEMORY_AVAILABLE ENDP


MEMORY_EXTENDED proc near
    mov al, 30h
    out 70h, al
    in al, 71h

mov al, 31h
    out 70h, al
    in al, 71h
    mov ah, al
mov bh, al
mov ax, bx

mov dx, offset EXTENDED_MEMORY
call WRITE_STR

mov bx, 010h
mul bx

call PARAGRAPHS_TO_BYTES

mov dx, offset STRING_BYTE
call WRITE_STR
```

```
mov dx, offset NEW_STRING
call WRITE_STR

ret
MEMORY_EXTENDED ENDP


MCB PROC near
mov ah, 52h
int 21h
mov ax, es:[bx-2]
mov es, ax
mov dx, offset MCB_TABLE
call WRITE_STR

MCB_loop:
    mov ax, es                  ;адрес
    mov di, offset ADDRESS
    add di, 12
    call WRD_TO_HEX
    mov dx, offset ADDRESS
    call WRITE_STR
mov dx, offset SPACE_STRING
call WRITE_STR

mov ax, es:[1]              ;psp адрес
mov di, offset PSP_ADDRESS
add di, 16
call WRD_TO_HEX
mov dx, offset PSP_ADDRESS
call WRITE_STR

mov dx, offset STRING_SIZE  ;размер
call WRITE_STR
mov ax, es:[3]
mov di, offset STRING_SIZE
add di, 6
mov bx, 16
mul bx
call PARAGRAPHS_TO_BYTES
mov dx, offset SPACE_STRING
call WRITE_STR

mov bx, 8                    ;SC/SD
mov dx, offset SC_SD
call WRITE_STR
mov cx, 7

SC_SD_loop:
mov dl, es:[bx]
mov ah, 02h
int 21h
inc bx
loop SC_SD_loop
```

```
mov dx, offset NEW_STRING
call WRITE_STR

mov bx, es:[3h]
mov al, es:[0h]
cmp al, 5ah
je MCB_END

mov ax, es
inc ax
add ax, bx
mov es, ax
jmp MCB_loop

MCB_END:
ret

MCB ENDP



BEGIN:
call MEMORY_AVAILABLE
call MEMORY_EXTENDED
call MCB

; Выход в DOS
xor al, al
    mov ah, 4ch
    int 21h

TESTPC ENDS
END START
```

Название файла: lab3_2.asm

```
TESTPC SEGMENT
ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
ORG 100H
START: JMP BEGIN
; ДАННЫЕ

AVAILABLE_MEMORY DB 'Available memory: ', '$'
EXTENDED_MEMORY DB 'Extended memory: ', '$'
STRING_BYTE DB ' byte ', '$'
MCB_TABLE DB 'MCB table: ', 0DH, 0AH, '$'
ADDRESS DB 'Address:     ', '$'
PSP_ADDRESS DB 'PSP address:      ', '$'
STRING_SIZE DB 'Size: ', '$'
SC_SD DB 'SC/SD: ', '$'
NEW_STRING DB 0DH,0AH,'$'
SPACE_STRING DB ' ', '$'
```

```
;ПРОЦЕДУРЫ
;-------------------------------------------------------
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
; байт в AL переводится в два символа шестн. числа в AX
push CX
mov AH,AL
call TETR_TO_HEX
xchg AL,AH
mov CL,4
shr AL,CL
call TETR_TO_HEX ;в AL старшая цифра
pop CX ;в AH младшая
ret
BYTE_TO_HEX ENDP
;--------------------------------
WRD_TO_HEX PROC near
;перевод в 16 с/с 16-ти разрядного числа
; в AX - число, DI - адрес последнего символа
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
; перевод в 10с/с, SI - адрес поля младшей цифры
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
;--------------------------------

WRITE_STR PROC near
push ax
mov ah,9h
int 21h
pop ax
ret
WRITE_STR ENDP

PARAGRAPHS_TO_BYTES PROC
mov bx, 0ah
xor cx, cx

division_loop:
div bx
push dx
inc cx
sub dx, dx
cmp ax, 0h
jne division_loop

write_symbol:
pop dx
add dl,30h
mov ah,02h
int 21h

loop write_symbol

ret

PARAGRAPHS_TO_BYTES ENDP


MEMORY_AVAILABLE PROC near
mov dx, offset AVAILABLE_MEMORY
call WRITE_STR
mov ah, 4ah
mov bx, 0ffffh
int 21h
mov ax, bx
mov bx, 16
mul bx
```

```
call PARAGRAPHS_TO_BYTES

mov dx, offset STRING_BYTE
call WRITE_STR

mov dx, offset NEW_STRING
call WRITE_STR

ret
MEMORY_AVAILABLE ENDP


MEMORY_EXTENDED proc near
    mov al, 30h
    out 70h, al
    in al, 71h

mov al, 31h
    out 70h, al
    in al, 71h
    mov ah, al
mov bh, al
mov ax, bx

mov dx, offset EXTENDED_MEMORY
call WRITE_STR

mov bx, 010h
mul bx

call PARAGRAPHS_TO_BYTES

mov dx, offset STRING_BYTE
call WRITE_STR

mov dx, offset NEW_STRING
call WRITE_STR

ret
MEMORY_EXTENDED ENDP


MCB PROC near
mov ah, 52h
int 21h
mov ax, es:[bx-2]
mov es, ax
mov dx, offset MCB_TABLE
call WRITE_STR

MCB_loop:
    mov ax, es                  ;адрес
    mov di, offset ADDRESS
    add di, 12
    call WRD_TO_HEX
```

```
    mov dx, offset ADDRESS
    call WRITE_STR
mov dx, offset SPACE_STRING
call WRITE_STR

mov ax, es:[1]              ;psp адрес
mov di, offset PSP_ADDRESS
add di, 16
call WRD_TO_HEX
mov dx, offset PSP_ADDRESS
call WRITE_STR

mov dx, offset STRING_SIZE  ;размер
call WRITE_STR
mov ax, es:[3]
mov di, offset STRING_SIZE
add di, 6
mov bx, 16
mul bx
call PARAGRAPHS_TO_BYTES
mov dx, offset SPACE_STRING
call WRITE_STR

mov bx, 8                    ;SC/SD
mov dx, offset SC_SD
call WRITE_STR
mov cx, 7

SC_SD_loop:
mov dl, es:[bx]
mov ah, 02h
int 21h
inc bx
loop SC_SD_loop

mov dx, offset NEW_STRING
call WRITE_STR

mov bx, es:[3h]
mov al, es:[0h]
cmp al, 5ah
je MCB_END

mov ax, es
inc ax
add ax, bx
mov es, ax
jmp MCB_loop

MCB_END:
ret

MCB ENDP
```

```
FREE_UNUSED_MEMORY PROC near
    mov     ax, cs
    mov     es, ax
    mov     bx, offset TESTPC_END
    mov     ax, es
    mov     bx, ax
    mov     ah, 4ah
    int     21h
    ret
FREE_UNUSED_MEMORY ENDP


BEGIN:
call MEMORY_AVAILABLE
call MEMORY_EXTENDED
call FREE_UNUSED_MEMORY
call MCB

; Выход в DOS
xor al, al
    mov ah, 4ch
    int 21h

TESTPC_END:
TESTPC ENDS
END START
```

Название файла: lab3_3.asm

```
TESTPC SEGMENT
ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
ORG 100H
START: JMP BEGIN
; ДАННЫЕ

AVAILABLE_MEMORY DB 'Available memory: ', '$'
EXTENDED_MEMORY DB 'Extended memory: ', '$'
STRING_BYTE DB ' byte ', '$'
MCB_TABLE DB 'MCB table: ', 0DH, 0AH, '$'
ADDRESS DB 'Address:     ', '$'
PSP_ADDRESS DB 'PSP address:      ', '$'
STRING_SIZE DB 'Size: ', '$'
SC_SD DB 'SC/SD: ', '$'
NEW_STRING DB 0DH,0AH,'$'
SPACE_STRING DB ' ', '$'
MEMORY_REQUEST_FAIL DB 'Memory request failed', 0DH, 0AH, '$'
MEMORY_REQUEST_SUCCESS DB 'Memory request succeeded', 0DH, 0AH, '$'

;ПРОЦЕДУРЫ
;-------------------------------------------------
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
; байт в AL переводится в два символа шестн. числа в AX
push CX
mov AH,AL
call TETR_TO_HEX
xchg AL,AH
mov CL,4
shr AL,CL
call TETR_TO_HEX ;в AL старшая цифра
pop CX ;в AH младшая
ret
BYTE_TO_HEX ENDP
;------------------------------
WRD_TO_HEX PROC near
;перевод в 16 с/с 16-ти разрядного числа
; в AX - число, DI - адрес последнего символа
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
; перевод в 10с/с, SI - адрес поля младшей цифры
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
;------------------------------

WRITE_STR PROC near
push ax
mov ah,9h
int 21h
pop ax
ret
WRITE_STR ENDP

PARAGRAPHS_TO_BYTES PROC
mov bx, 0ah
xor cx, cx

division_loop:
div bx
push dx
inc cx
sub dx, dx
cmp ax, 0h
jne division_loop

write_symbol:
pop dx
add dl,30h
mov ah,02h
int 21h

loop write_symbol

ret

PARAGRAPHS_TO_BYTES ENDP


MEMORY_AVAILABLE PROC near
mov dx, offset AVAILABLE_MEMORY
call WRITE_STR
mov ah, 4ah
mov bx, 0ffffh
int 21h
mov ax, bx
mov bx, 16
mul bx
call PARAGRAPHS_TO_BYTES

mov dx, offset STRING_BYTE
call WRITE_STR

mov dx, offset NEW_STRING
```

```
call WRITE_STR

ret
MEMORY_AVAILABLE ENDP


MEMORY_EXTENDED proc near
    mov al, 30h
    out 70h, al
    in al, 71h

mov al, 31h
    out 70h, al
    in al, 71h
    mov ah, al
mov bh, al
mov ax, bx

mov dx, offset EXTENDED_MEMORY
call WRITE_STR

mov bx, 010h
mul bx

call PARAGRAPHS_TO_BYTES

mov dx, offset STRING_BYTE
call WRITE_STR

mov dx, offset NEW_STRING
call WRITE_STR

ret
MEMORY_EXTENDED ENDP


MCB PROC near
mov ah, 52h
int 21h
mov ax, es:[bx-2]
mov es, ax
mov dx, offset MCB_TABLE
call WRITE_STR

MCB_loop:
    mov ax, es                  ;адрес
    mov di, offset ADDRESS
    add di, 12
    call WRD_TO_HEX
    mov dx, offset ADDRESS
    call WRITE_STR
mov dx, offset SPACE_STRING
call WRITE_STR

mov ax, es:[1]              ;psp адрес
```

```
mov di, offset PSP_ADDRESS
add di, 16
call WRD_TO_HEX
mov dx, offset PSP_ADDRESS
call WRITE_STR

mov dx, offset STRING_SIZE  ;размер
call WRITE_STR
mov ax, es:[3]
mov di, offset STRING_SIZE
add di, 6
mov bx, 16
mul bx
call PARAGRAPHS_TO_BYTES
mov dx, offset SPACE_STRING
call WRITE_STR

mov bx, 8                    ;SC/SD
mov dx, offset SC_SD
call WRITE_STR
mov cx, 7

SC_SD_loop:
mov dl, es:[bx]
mov ah, 02h
int 21h
inc bx
loop SC_SD_loop

mov dx, offset NEW_STRING
call WRITE_STR

mov bx, es:[3h]
mov al, es:[0h]
cmp al, 5ah
je MCB_END

mov ax, es
inc ax
add ax, bx
mov es, ax
jmp MCB_loop

MCB_END:
ret

MCB ENDP


FREE_UNUSED_MEMORY PROC near
    mov ax, cs
    mov es, ax
    mov bx, offset TESTPC_END
    mov ax, es
    mov bx, ax
```

```
    mov ah, 4ah
    int 21h
    ret
FREE_UNUSED_MEMORY ENDP


MEMORY_REQUEST PROC near
    mov bx, 1000h
    mov ah, 48h
    int 21h

    jb memory_fail
    jmp memory_success

memory_fail:
    mov dx, offset MEMORY_REQUEST_FAIL
    call WRITE_STR
    jmp memory_request_end

memory_success:
    mov dx, offset MEMORY_REQUEST_SUCCESS
    call WRITE_STR

memory_request_end:
    ret

MEMORY_REQUEST ENDP


BEGIN:
call MEMORY_AVAILABLE
call MEMORY_EXTENDED
call FREE_UNUSED_MEMORY
call MEMORY_REQUEST
call MCB

; Выход в DOS
xor al, al
    mov ah, 4ch
    int 21h

TESTPC_END:
TESTPC ENDS
END START
```

Название файла: lab3_4.asm

```
TESTPC SEGMENT
ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
ORG 100H
START: JMP BEGIN
; ДАННЫЕ
```

```
AVAILABLE_MEMORY DB 'Available memory: ', '$'
EXTENDED_MEMORY DB 'Extended memory: ', '$'
STRING_BYTE DB ' byte ', '$'
MCB_TABLE DB 'MCB table: ', 0DH, 0AH, '$'
ADDRESS DB 'Address:     ', '$'
PSP_ADDRESS DB 'PSP address:      ', '$'
STRING_SIZE DB 'Size: ', '$'
SC_SD DB 'SC/SD: ', '$'
NEW_STRING DB 0DH,0AH,'$'
SPACE_STRING DB ' ', '$'
MEMORY_REQUEST_FAIL DB 'Memory request failed', 0DH, 0AH, '$'
MEMORY_REQUEST_SUCCESS DB 'Memory request succeeded', 0DH, 0AH, '$'

;ПРОЦЕДУРЫ
;--------------------------------------------------
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
; байт в AL переводится в два символа шестн. числа в AX
push CX
mov AH,AL
call TETR_TO_HEX
xchg AL,AH
mov CL,4
shr AL,CL
call TETR_TO_HEX ;в AL старшая цифра
pop CX ;в AH младшая
ret
BYTE_TO_HEX ENDP
;--------------------------------
WRD_TO_HEX PROC near
;перевод в 16 с/с 16-ти разрядного числа
; в AX - число, DI - адрес последнего символа
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
; перевод в 10с/с, SI - адрес поля младшей цифры
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
;--------------------------------

WRITE_STR PROC near
push ax
mov ah,9h
int 21h
pop ax
ret
WRITE_STR ENDP

PARAGRAPHS_TO_BYTES PROC
mov bx, 0ah
xor cx, cx

division_loop:
div bx
push dx
inc cx
sub dx, dx
cmp ax, 0h
jne division_loop

write_symbol:
pop dx
add dl,30h
mov ah,02h
int 21h

loop write_symbol

ret
```

```
PARAGRAPHS_TO_BYTES ENDP


MEMORY_AVAILABLE PROC near
mov dx, offset AVAILABLE_MEMORY
call WRITE_STR
mov ah, 4ah
mov bx, 0ffffh
int 21h
mov ax, bx
mov bx, 16
mul bx
call PARAGRAPHS_TO_BYTES

mov dx, offset STRING_BYTE
call WRITE_STR

mov dx, offset NEW_STRING
call WRITE_STR

ret
MEMORY_AVAILABLE ENDP


MEMORY_EXTENDED proc near
    mov al, 30h
    out 70h, al
    in al, 71h

mov al, 31h
    out 70h, al
    in al, 71h
    mov ah, al
mov bh, al
mov ax, bx

mov dx, offset EXTENDED_MEMORY
call WRITE_STR

mov bx, 010h
mul bx

call PARAGRAPHS_TO_BYTES

mov dx, offset STRING_BYTE
call WRITE_STR

mov dx, offset NEW_STRING
call WRITE_STR

ret
MEMORY_EXTENDED ENDP


MCB PROC near
```

```
mov ah, 52h
int 21h
mov ax, es:[bx-2]
mov es, ax
mov dx, offset MCB_TABLE
call WRITE_STR

MCB_loop:
    mov ax, es                  ;адрес
    mov di, offset ADDRESS
    add di, 12
    call WRD_TO_HEX
    mov dx, offset ADDRESS
    call WRITE_STR
mov dx, offset SPACE_STRING
call WRITE_STR

mov ax, es:[1]              ;psp адрес
mov di, offset PSP_ADDRESS
add di, 16
call WRD_TO_HEX
mov dx, offset PSP_ADDRESS
call WRITE_STR

mov dx, offset STRING_SIZE  ;размер
call WRITE_STR
mov ax, es:[3]
mov di, offset STRING_SIZE
add di, 6
mov bx, 16
mul bx
call PARAGRAPHS_TO_BYTES
mov dx, offset SPACE_STRING
call WRITE_STR

mov bx, 8                    ;SC/SD
mov dx, offset SC_SD
call WRITE_STR
mov cx, 7

SC_SD_loop:
mov dl, es:[bx]
mov ah, 02h
int 21h
inc bx
loop SC_SD_loop

mov dx, offset NEW_STRING
call WRITE_STR

mov bx, es:[3h]
mov al, es:[0h]
cmp al, 5ah
je MCB_END
```

```
mov ax, es
inc ax
add ax, bx
mov es, ax
jmp MCB_loop

MCB_END:
ret

MCB ENDP


FREE_UNUSED_MEMORY PROC near
    mov ax, cs
    mov es, ax
    mov bx, offset TESTPC_END
    mov ax, es
    mov bx, ax
    mov ah, 4ah
    int 21h
    ret
FREE_UNUSED_MEMORY ENDP


MEMORY_REQUEST PROC near
    mov bx, 1000h
    mov ah, 48h
    int 21h

    jb memory_fail
    jmp memory_success

memory_fail:
    mov dx, offset MEMORY_REQUEST_FAIL
    call WRITE_STR
    jmp memory_request_end

memory_success:
    mov dx, offset MEMORY_REQUEST_SUCCESS
    call WRITE_STR

memory_request_end:
    ret

MEMORY_REQUEST ENDP


BEGIN:
call MEMORY_REQUEST
call MEMORY_AVAILABLE
call MEMORY_EXTENDED
call FREE_UNUSED_MEMORY
call MCB

; Выход в DOS
```

```
xor al, al
    mov ah, 4ch
    int 21h

TESTPC_END:
TESTPC ENDS
END START
```