

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

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

Студент гр. 8382

Янкин Д.О.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

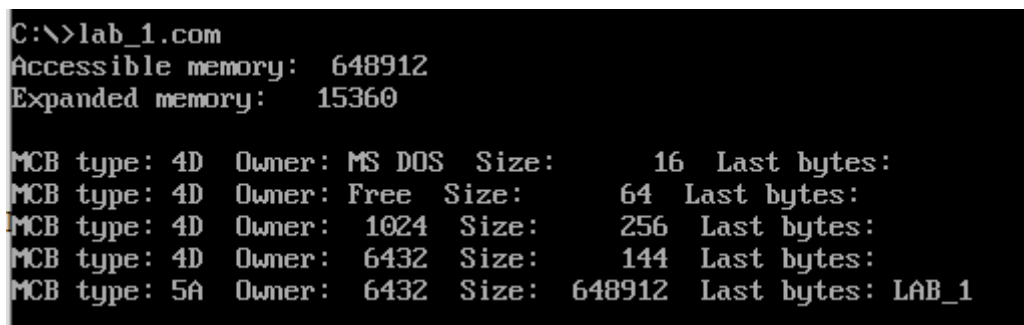
Цель работы.

Для исследования организации управления памятью необходимо ориентироваться на тип основной памяти, реализованный 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 преобразовывают этот список.

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

Ход работы.

Написан код программы, выводящей размер доступной памяти, расширенной памяти и список MCB (см. рисунок 1).



```
C:\>lab_1.com
Accessible memory: 648912
Expanded memory: 15360

MCB type: 4D Owner: MS DOS Size: 16 Last bytes:
MCB type: 4D Owner: Free Size: 64 Last bytes:
MCB type: 4D Owner: 1024 Size: 256 Last bytes:
MCB type: 4D Owner: 6432 Size: 144 Last bytes:
MCB type: 5A Owner: 6432 Size: 648912 Last bytes: LAB_1
```

Рисунок 1. Результат работы первой программы

Предыдущая программа модифицирована так, чтобы освободить память, которую она не занимает (см. рисунок 2). Так как в com программах дно стека лежит со смещением FFFFh от начала сегмента, то программа занимает 64 КБ.

```

C:\>lab_2.com
Accessible memory: 648912
Expanded memory: 15360

Free: OK

MCB type: 4D Owner: MS DOS Size: 16 Last bytes:
MCB type: 4D Owner: Free Size: 64 Last bytes:
MCB type: 4D Owner: 1024 Size: 256 Last bytes:
MCB type: 4D Owner: 6432 Size: 144 Last bytes:
MCB type: 4D Owner: 6432 Size: 65536 Last bytes: LAB_2
MCB type: 5A Owner: Free Size: 583360 Last bytes: ength

```

Рисунок 2. Результат работы второй программы

Предыдущая программа модифицирована так, чтобы после освобождения неиспользуемой памяти дополнительно запрашивать блок памяти размером 64 Кб (см. рисунок 3).

```

C:\>lab_3.com
Accessible memory: 648912
Expanded memory: 15360

Free: OK
Allocation: OK

MCB type: 4D Owner: MS DOS Size: 16 Last bytes:
MCB type: 4D Owner: Free Size: 64 Last bytes:
MCB type: 4D Owner: 1024 Size: 256 Last bytes:
MCB type: 4D Owner: 6432 Size: 144 Last bytes:
MCB type: 4D Owner: 6432 Size: 65536 Last bytes: LAB_3
MCB type: 4D Owner: 6432 Size: 65536 Last bytes: LAB_3
MCB type: 5A Owner: Free Size: 517808 Last bytes:

```

Рисунок 3. Результат работы третьей программы

Изначальная программа модифицирована так, чтобы сначала запрашивать новый блок памяти размером 64 КБ, а только после этого освобождать неиспользуемую (см. рисунок 4).

```

C:\>lab_4.com
Accessible memory: 648912
Expanded memory: 15360

Allocation: ERROR
Free: OK

MCB type: 4D Owner: MS DOS Size: 16 Last bytes:
MCB type: 4D Owner: Free Size: 64 Last bytes:
MCB type: 4D Owner: 1024 Size: 256 Last bytes:
MCB type: 4D Owner: 6432 Size: 144 Last bytes:
MCB type: 4D Owner: 6432 Size: 65536 Last bytes: LAB_4
MCB type: 5A Owner: Free Size: 583360 Last bytes: ength

```

Рисунок 4. Результат работы четвертой программы

Контрольные вопросы.

- 1) Что такое «доступный объем памяти»?

Максимальный объем ОП, который может быть выделен программе.

- 2) Где MCB блок вашей программы в списке?

На скриншотах это те блоки, у которых в поле «Last bytes» указано «LAB_#», где # – номер программы.

Во второй программе неиспользуемая память была освобождена, поэтому был создан новый блок, помеченный свободным.

В третьей программе был запрошен дополнительный участок памяти, поэтому моя программа имеет два MCB блока в списке.

В четвертой программе запрос на выделение памяти был осуществлен в тот момент, когда достаточного объема свободной памяти не было, поэтому моя программа не получила второй MCB блок.

- 3) Какой размер памяти занимает программа в каждом случае?

В первом случае программа заняла весь свободный участок памяти в 648912 байт.

Во втором случае – 64 КБ, так как лишняя память была освобождена.

В третьем – 128 КБ, так как были дополнительно запрошены 64 КБ.

В четвертом – 64 КБ, так как выделение дополнительной памяти было неудачным.

Выводы.

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

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ LAB_1.ASM

TESTPC SEGMENT

ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING

ORG 100H

START: JMP BEGIN

; ДАННЫЕ

ACCESSIBLE_MEMORY_MESSAGE db 'Accessible memory: ', '\$'

ACCESSIBLE_MEMORY db ' ', 13, 10, '\$'

EXPANDED_MEMORY_MESSAGE db 'Expanded memory: ', '\$'

EXPANDED_MEMORY db ' ', 13, 10, '\$'

MCB_TYPE_MESSAGE db 'MCB type: ', '\$'

MCB_TYPE db ' ', '\$'

OWNER_MESSAGE db 'Owner: ', '\$'

OWNER_FREE_MESSAGE db 'Free', '\$'

OWNER_XMSUMB_MESSAGE db 'OS XMS UMB', '\$'

OWNER_DRIVER_MESSAGE db 'Исключенная верхняя память
драйвера; даже гуглить правильный перевод не буду', '\$'

OWNER_MSDOS_MESSAGE db 'MS DOS', '\$'

OWNER_CONTROLLER_MESSAGE db '386MAX UMB Controller', '\$'

OWNER_BLOCKED_MESSAGE db 'Blocked by 386MAX UMB', '\$'

OWNER_MAXUMB_MESSAGE db '386MAX UMB', '\$'

OWNER_UNKNOWN db ' ', '\$'

BLOCK_SIZE_MESSAGE db 'Size: ', '\$'

BLOCK_SIZE db ' ', '\$'

LAST_BYTES_MESSAGE db 'Last bytes: ', '\$'

```

FREE_OK_MESSAGE          db 'Free: OK', 13, 10, '$'
FREE_ERROR_MESSAGE       db 'Free: ERROR', 13, 10, '$'

```

```

ALLOCATION_OK_MESSAGE     db 'Allocation: OK', 13, 10, '$'
ALLOCATION_ERROR_MESSAGE   db 'Allocation: ERROR', 13,
10, '$'

```

```

; ПРОЦЕДУРЫ

```

```

;-----

```

```

TETR_TO_HEX PROC near

```

```

; младшая шестн. цифра AL в шестн. цифру ASCII

```

```

    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 шестн. числа ASCII 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

```

```

    ;xchg al, ah          ;; а теперь наоборот!

```

```

    ret

```

```

BYTE_TO_HEX ENDP

```

```

;-----
WRD_TO_HEX PROC near
; перевод в 16 с/с 16-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 10 с/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_1
        or AL,30h
        mov [SI],AL
        dec si

end_1:   pop DX
        pop CX
        ret
BYTE_TO_DEC ENDP

```

```

;-----
WRD_TO_DEC PROC near
        push ax
        push bx

        mov bx, 10

div_loop:
        div bx
        add dl, 30h
        mov [si], dl
        dec si
        mov dx, 0
        cmp ax, 0
        jne div_loop

        pop bx
        pop ax
        ret
WRD_TO_DEC ENDP

```

```

;-----
PRINT_STRING PROC near
; Просто выводит строку с уже указанным в dx смещением, очень
сложная функция
    push ax

    mov ah, 09h
    int 21h

    pop ax
    ret
PRINT_STRING ENDP

;-----
PRINT_WORD PROC near
; Выводит регистр AX
    push ax
    push dx

    mov dl, ah
    mov ah, 02h
    int 21h

    mov dl, al
    int 21h

    pop dx
    pop ax
    ret
PRINT_WORD ENDP

;-----

```

```

PRINT_ENDL PROC near
; Выводит 13, 10
    push ax
    push dx

    mov dl, 13
    mov ah, 02h
    int 21h

    mov dl, 10
    int 21h

    pop dx
    pop ax
    ret
PRINT_ENDL ENDP

```

```

PRINT_DSPACE PROC near
; Выводит два пробела
    push ax
    push dx

    mov dl, ' '
    mov ah, 02h
    int 21h

    mov dl, ' '
    int 21h

    pop dx
    pop ax
    ret
PRINT_DSPACE ENDP

```

```

;-----
; КОД
BEGIN:

    ; Размер доступной памяти
    mov dx, offset ACCESSIBLE_MEMORY_MESSAGE
    call PRINT_STRING

    mov bx, 0FFFFh
    mov ah, 4Ah
    int 21h

    mov ax, bx
    mov bx, 16
    mul bx

    mov si, offset ACCESSIBLE_MEMORY
    add si, 6
    call WRD_TO_DEC
    mov dx, offset ACCESSIBLE_MEMORY
    call PRINT_STRING

    ; Размер расширенной памяти
    mov dx, offset EXPANDED_MEMORY_MESSAGE
    call PRINT_STRING

    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 EXPANDED_MEMORY
```

```
add si, 6
```

```
mov dx, 0
```

```
call WRD_TO_DEC
```

```
mov dx, offset EXPANDED_MEMORY
```

```
call PRINT_STRING
```

```
call PRINT_ENDL
```

```
; Цепочка блоков управления памятью
```

```
mov ah, 52h
```

```
int 21h
```

```
mov es, es:[bx - 2]
```

```
MCB_loop:
```

```
; Вывод типа MCB
```

```
mov dx, offset MCB_TYPE_MESSAGE
```

```
call PRINT_STRING
```

```
mov al, es:[0]
```

```
call BYTE_TO_HEX
```

```
xchg ah, al
```

```
mov dl, ah
```

```
mov ah, 02h
```

```
push ax
```

```
int 21h
```

```
pop ax
```

```
mov dl, al
```

```
int 21h
```

```

call PRINT_DSPACE

; Вывод владельца
mov dx, offset OWNER_MESSAGE
call PRINT_STRING

mov ax, es:[1]
check_0:
    cmp ax, 0000h
    jne check_1
    mov dx, offset OWNER_FREE_MESSAGE
    call PRINT_STRING
    jmp owner_end
check_1:
    cmp ax, 0006h
    jne check_2
    mov dx, offset OWNER_XMSUMB_MESSAGE
    call PRINT_STRING
    jmp owner_end
check_2:
    cmp ax, 0007h
    jne check_3
    mov dx, offset OWNER_DRIVER_MESSAGE
    call PRINT_STRING
    jmp owner_end
check_3:
    cmp ax, 0008h
    jne check_4
    mov dx, offset OWNER_MSDOS_MESSAGE
    call PRINT_STRING
    jmp owner_end
check_4:
    cmp ax, 0FFFAh

```

```

        jne check_5
        mov dx, offset OWNER_CONTROLLER_MESSAGE
        call PRINT_STRING
        jmp owner_end

check_5:
        cmp ax, 0FFFDh
        jne check_6
        mov dx, offset OWNER_BLOCKED_MESSAGE
        call PRINT_STRING
        jmp owner_end

check_6:
        cmp ax, 0FFFEh
        jne check_last
        mov dx, offset OWNER_MAXUMB_MESSAGE
        call PRINT_STRING
        jmp owner_end

check_last:
        mov dx, 16
        mul dx
        mov si, offset OWNER_UNKNOWN
        add si, 4
        call WRD_TO_DEC
        mov dx, offset OWNER_UNKNOWN
        call PRINT_STRING

owner_end:
        call PRINT_DSPACE
        jmp block_size_label

jump_up:
        jmp MCB_loop

block_size_label:
        ; Вывод размера участка

```

```
mov dx, offset BLOCK_SIZE_MESSAGE
call PRINT_STRING
```

```
mov ax, es:[3]
mov bx, 10h
mul bx
mov si, offset BLOCK_SIZE
add si, 6
call WRD_TO_DEC
mov dx, offset BLOCK_SIZE
call PRINT_STRING
call PRINT_DSPACE
```

```
; Вывод последних байт
mov dx, offset LAST_BYTES_MESSAGE
call PRINT_STRING
```

```
mov cx, 8
mov si, 8h
mov ah, 02h
```

```
last_bytes_loop:
    mov dl, es:[si]
    int 21h
    inc si
    loop last_bytes_loop
    call PRINT_ENDL
```

```
mov al, es:[0]
cmp al, 5Ah
je exit
mov ax, es
add ax, es:[3]
inc ax
```



```
        mov es, ax
        jmp jump_up

exit:
        ret

TESTPC  ENDS
        END START ; конец модуля, START – точка входа
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ LAB_2.ASM

TESTPC SEGMENT

ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING

ORG 100H

START: JMP BEGIN

; ДАННЫЕ

ACCESSIBLE_MEMORY_MESSAGE db 'Accessible memory: ', '\$'

ACCESSIBLE_MEMORY db ' ', 13, 10, '\$'

EXPANDED_MEMORY_MESSAGE db 'Expanded memory: ', '\$'

EXPANDED_MEMORY db ' ', 13, 10, '\$'

MCB_TYPE_MESSAGE db 'MCB type: ', '\$'

MCB_TYPE db ' ', '\$'

OWNER_MESSAGE db 'Owner: ', '\$'

OWNER_FREE_MESSAGE db 'Free', '\$'

OWNER_XMSUMB_MESSAGE db 'OS XMS UMB', '\$'

OWNER_DRIVER_MESSAGE db 'Исключенная верхняя память
драйвера; даже гуглить правильный перевод не буду', '\$'

OWNER_MSDOS_MESSAGE db 'MS DOS', '\$'

OWNER_CONTROLLER_MESSAGE db '386MAX UMB Controller', '\$'

OWNER_BLOCKED_MESSAGE db 'Blocked by 386MAX UMB', '\$'

OWNER_MAXUMB_MESSAGE db '386MAX UMB', '\$'

OWNER_UNKNOWN db ' ', '\$'

BLOCK_SIZE_MESSAGE db 'Size: ', '\$'

BLOCK_SIZE db ' ', '\$'

LAST_BYTES_MESSAGE db 'Last bytes: ', '\$'

```

FREE_OK_MESSAGE                db 'Free: OK', 13, 10, '$'
FREE_ERROR_MESSAGE            db 'Free: ERROR', 13, 10, '$'

```

```

ALLOCATION_OK_MESSAGE          db 'Allocation: OK', 13, 10, '$'
ALLOCATION_ERROR_MESSAGE       db 'Allocation: ERROR', 13,
10, '$'

```

```

; ПРОЦЕДУРЫ

```

```

;-----

```

```

TETR_TO_HEX PROC near

```

```

; младшая шестн. цифра AL в шестн. цифру ASCII

```

```

    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 шестн. числа ASCII 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

```

```

    ;xchg al, ah        ;; а теперь наоборот!

```

```

    ret

```

```

BYTE_TO_HEX ENDP

```

```

;-----
WRD_TO_HEX PROC near
; перевод в 16 с/с 16-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 10 с/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_1
        or AL,30h
        mov [SI],AL
        dec si

end_1:   pop DX
        pop CX
        ret
BYTE_TO_DEC ENDP

```

```

;-----
WRD_TO_DEC PROC near
        push ax
        push bx

        mov bx, 10

div_loop:
        div bx
        add dl, 30h
        mov [si], dl
        dec si
        mov dx, 0
        cmp ax, 0
        jne div_loop

        pop bx
        pop ax
        ret
WRD_TO_DEC ENDP

```

```

;-----
PRINT_STRING PROC near
; Просто выводит строку с уже указанным в dx смещением, очень
сложная функция
    push ax

    mov ah, 09h
    int 21h

    pop ax
    ret
PRINT_STRING ENDP

;-----
PRINT_WORD PROC near
; Выводит регистр AX
    push ax
    push dx

    mov dl, ah
    mov ah, 02h
    int 21h

    mov dl, al
    int 21h

    pop dx
    pop ax
    ret
PRINT_WORD ENDP

;-----

```

```

PRINT_ENDL PROC near
; Выводит 13, 10
    push ax
    push dx

    mov dl, 13
    mov ah, 02h
    int 21h

    mov dl, 10
    int 21h

    pop dx
    pop ax
    ret
PRINT_ENDL ENDP

```

```

PRINT_DSPACE PROC near
; Выводит два пробела
    push ax
    push dx

    mov dl, ' '
    mov ah, 02h
    int 21h

    mov dl, ' '
    int 21h

    pop dx
    pop ax
    ret
PRINT_DSPACE ENDP

```

```

;-----
; КОД
BEGIN:

    ; Размер доступной памяти
    mov dx, offset ACCESSIBLE_MEMORY_MESSAGE
    call PRINT_STRING

    mov bx, 0FFFFh
    mov ah, 4Ah
    int 21h

    mov ax, bx
    mov bx, 16
    mul bx

    mov si, offset ACCESSIBLE_MEMORY
    add si, 6
    call WRD_TO_DEC
    mov dx, offset ACCESSIBLE_MEMORY
    call PRINT_STRING

    ; Размер расширенной памяти
    mov dx, offset EXPANDED_MEMORY_MESSAGE
    call PRINT_STRING

    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 EXPANDED_MEMORY
```

```
add si, 6
```

```
mov dx, 0
```

```
call WRD_TO_DEC
```

```
mov dx, offset EXPANDED_MEMORY
```

```
call PRINT_STRING
```

```
call PRINT_ENDL
```

```
; Цепочка блоков управления памятью
```

```
mov ah, 52h
```

```
int 21h
```

```
mov es, es:[bx - 2]
```

```
MCB_loop:
```

```
; Вывод типа MCB
```

```
mov dx, offset MCB_TYPE_MESSAGE
```

```
call PRINT_STRING
```

```
mov al, es:[0]
```

```
call BYTE_TO_HEX
```

```
xchg ah, al
```

```
mov dl, ah
```

```
mov ah, 02h
```

```
push ax
```

```
int 21h
```

```
pop ax
```

```
mov dl, al
```

```
int 21h
```

```

call PRINT_DSPACE

; Вывод владельца
mov dx, offset OWNER_MESSAGE
call PRINT_STRING

mov ax, es:[1]
check_0:
    cmp ax, 0000h
    jne check_1
    mov dx, offset OWNER_FREE_MESSAGE
    call PRINT_STRING
    jmp owner_end
check_1:
    cmp ax, 0006h
    jne check_2
    mov dx, offset OWNER_XMSUMB_MESSAGE
    call PRINT_STRING
    jmp owner_end
check_2:
    cmp ax, 0007h
    jne check_3
    mov dx, offset OWNER_DRIVER_MESSAGE
    call PRINT_STRING
    jmp owner_end
check_3:
    cmp ax, 0008h
    jne check_4
    mov dx, offset OWNER_MSDOS_MESSAGE
    call PRINT_STRING
    jmp owner_end
check_4:
    cmp ax, 0FFFAh

```

```

        jne check_5
        mov dx, offset OWNER_CONTROLLER_MESSAGE
        call PRINT_STRING
        jmp owner_end

check_5:
        cmp ax, 0FFFDh
        jne check_6
        mov dx, offset OWNER_BLOCKED_MESSAGE
        call PRINT_STRING
        jmp owner_end

check_6:
        cmp ax, 0FFFEh
        jne check_last
        mov dx, offset OWNER_MAXUMB_MESSAGE
        call PRINT_STRING
        jmp owner_end

check_last:
        mov dx, 16
        mul dx
        mov si, offset OWNER_UNKNOWN
        add si, 4
        call WRD_TO_DEC
        mov dx, offset OWNER_UNKNOWN
        call PRINT_STRING

owner_end:
        call PRINT_DSPACE
        jmp block_size_label

jump_up:
        jmp MCB_loop

block_size_label:
        ; Вывод размера участка

```

```
mov dx, offset BLOCK_SIZE_MESSAGE
call PRINT_STRING
```

```
mov ax, es:[3]
mov bx, 10h
mul bx
mov si, offset BLOCK_SIZE
add si, 6
call WRD_TO_DEC
mov dx, offset BLOCK_SIZE
call PRINT_STRING
call PRINT_DSPACE
```

```
; Вывод последних байт
mov dx, offset LAST_BYTES_MESSAGE
call PRINT_STRING
```

```
mov cx, 8
mov si, 8h
mov ah, 02h
```

```
last_bytes_loop:
    mov dl, es:[si]
    int 21h
    inc si
    loop last_bytes_loop
    call PRINT_ENDL
```

```
mov al, es:[0]
cmp al, 5Ah
je exit
mov ax, es
add ax, es:[3]
inc ax
```

```
        mov es, ax
        jmp jump_up

exit:

        ret

TESTPC  ENDS

        END START ; конец модуля, START – точка входа
```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД ПРОГРАММЫ LAB_3.ASM

TESTPC SEGMENT

ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING

ORG 100H

START: JMP BEGIN

; ДАННЫЕ

ACCESSIBLE_MEMORY_MESSAGE db 'Accessible memory: ', '\$'

ACCESSIBLE_MEMORY db ' ', 13, 10, '\$'

EXPANDED_MEMORY_MESSAGE db 'Expanded memory: ', '\$'

EXPANDED_MEMORY db ' ', 13, 10, '\$'

MCB_TYPE_MESSAGE db 'MCB type: ', '\$'

MCB_TYPE db ' ', '\$'

OWNER_MESSAGE db 'Owner: ', '\$'

OWNER_FREE_MESSAGE db 'Free', '\$'

OWNER_XMSUMB_MESSAGE db 'OS XMS UMB', '\$'

OWNER_DRIVER_MESSAGE db 'Исключенная верхняя память
драйвера; даже гуглить правильный перевод не буду', '\$'

OWNER_MSDOS_MESSAGE db 'MS DOS', '\$'

OWNER_CONTROLLER_MESSAGE db '386MAX UMB Controller', '\$'

OWNER_BLOCKED_MESSAGE db 'Blocked by 386MAX UMB', '\$'

OWNER_MAXUMB_MESSAGE db '386MAX UMB', '\$'

OWNER_UNKNOWN db ' ', '\$'

BLOCK_SIZE_MESSAGE db 'Size: ', '\$'

BLOCK_SIZE db ' ', '\$'

LAST_BYTES_MESSAGE db 'Last bytes: ', '\$'

```

FREE_OK_MESSAGE                db 'Free: OK', 13, 10, '$'
FREE_ERROR_MESSAGE             db 'Free: ERROR', 13, 10, '$'

```

```

ALLOCATION_OK_MESSAGE           db 'Allocation: OK', 13, 10, '$'
ALLOCATION_ERROR_MESSAGE        db 'Allocation: ERROR', 13,
10, '$'

```

```

; ПРОЦЕДУРЫ

```

```

;-----

```

```

TETR_TO_HEX PROC near

```

```

; младшая шестн. цифра AL в шестн. цифру ASCII

```

```

    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 шестн. числа ASCII 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

```

```

    ;xchg al, ah        ;; а теперь наоборот!

```

```

    ret

```

```

BYTE_TO_HEX ENDP

```

```

;-----
WRD_TO_HEX PROC near
; перевод в 16 с/с 16-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 10 с/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_1
        or AL,30h
        mov [SI],AL
        dec si

end_1:   pop DX
        pop CX
        ret
BYTE_TO_DEC ENDP

```

```

;-----
WRD_TO_DEC PROC near
        push ax
        push bx

        mov bx, 10

div_loop:
        div bx
        add dl, 30h
        mov [si], dl
        dec si
        mov dx, 0
        cmp ax, 0
        jne div_loop

        pop bx
        pop ax
        ret
WRD_TO_DEC ENDP

```

```

;-----
PRINT_STRING PROC near
; Просто выводит строку с уже указанным в dx смещением, очень
сложная функция
    push ax

    mov ah, 09h
    int 21h

    pop ax
    ret
PRINT_STRING ENDP

;-----
PRINT_WORD PROC near
; Выводит регистр AX
    push ax
    push dx

    mov dl, ah
    mov ah, 02h
    int 21h

    mov dl, al
    int 21h

    pop dx
    pop ax
    ret
PRINT_WORD ENDP

;-----

```

```

PRINT_ENDL PROC near
; Выводит 13, 10
    push ax
    push dx

    mov dl, 13
    mov ah, 02h
    int 21h

    mov dl, 10
    int 21h

    pop dx
    pop ax
    ret
PRINT_ENDL ENDP

```

```

PRINT_DSPACE PROC near
; Выводит два пробела
    push ax
    push dx

    mov dl, ' '
    mov ah, 02h
    int 21h

    mov dl, ' '
    int 21h

    pop dx
    pop ax
    ret
PRINT_DSPACE ENDP

```

```

;-----
; КОД
BEGIN:

    ; Размер доступной памяти
    mov dx, offset ACCESSIBLE_MEMORY_MESSAGE
    call PRINT_STRING

    mov bx, 0FFFFh
    mov ah, 4Ah
    int 21h

    mov ax, bx
    mov bx, 16
    mul bx

    mov si, offset ACCESSIBLE_MEMORY
    add si, 6
    call WRD_TO_DEC
    mov dx, offset ACCESSIBLE_MEMORY
    call PRINT_STRING

    ; Размер расширенной памяти
    mov dx, offset EXPANDED_MEMORY_MESSAGE
    call PRINT_STRING

    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 EXPANDED_MEMORY
        add si, 6
        mov dx, 0
        call WRD_TO_DEC
        mov dx, offset EXPANDED_MEMORY
        call PRINT_STRING
        call PRINT_ENDL


        ; Освобождение
        ; В конце сегмента, вроде, стек должен валяться,
        ; поэтому 65536/16 = 4096 параграфов
        mov bx, 4096
        mov ah, 4Ah
        int 21h

        jc free_error

free_ok:
        mov dx, offset FREE_OK_MESSAGE
        jmp print_free_message

free_error:
        mov dx, offset FREE_ERROR_MESSAGE

print_free_message:
        call PRINT_STRING

```

```

; Запрос новых 64 Кб = 4096 параграфов
mov bx, 4096
mov ah, 48h
int 21h

jc allocation_error
allocation_ok:
    mov dx, offset ALLOCATION_OK_MESSAGE
    jmp print_allocation_message

allocation_error:
    mov dx, offset ALLOCATION_ERROR_MESSAGE

print_allocation_message:
    call PRINT_STRING
    call PRINT_ENDL


; Цепочка блоков управления памятью
mov ah, 52h
int 21h
mov es, es:[bx - 2]

MCB_loop:
    ; Вывод типа MCB
    mov dx, offset MCB_TYPE_MESSAGE
    call PRINT_STRING

    mov al, es:[0]
    call BYTE_TO_HEX
    xchg ah, al
    mov dl, ah
    mov ah, 02h

```

```

        push ax
        int 21h
        pop ax
        mov dl, al
        int 21h
        call PRINT_DSPACE

        ; Вывод владельца
        mov dx, offset OWNER_MESSAGE
        call PRINT_STRING

        mov ax, es:[1]
check_0:
        cmp ax, 0000h
        jne check_1
        mov dx, offset OWNER_FREE_MESSAGE
        call PRINT_STRING
        jmp owner_end
check_1:
        cmp ax, 0006h
        jne check_2
        mov dx, offset OWNER_XMSUMB_MESSAGE
        call PRINT_STRING
        jmp owner_end
check_2:
        cmp ax, 0007h
        jne check_3
        mov dx, offset OWNER_DRIVER_MESSAGE
        call PRINT_STRING
        jmp owner_end
check_3:
        cmp ax, 0008h
        jne check_4

```

```

        mov dx, offset OWNER_MSDOS_MESSAGE
        call PRINT_STRING
        jmp owner_end

check_4:
        cmp ax, 0FFFAh
        jne check_5
        mov dx, offset OWNER_CONTROLLER_MESSAGE
        call PRINT_STRING
        jmp owner_end

check_5:
        cmp ax, 0FFFDh
        jne check_6
        mov dx, offset OWNER_BLOCKED_MESSAGE
        call PRINT_STRING
        jmp owner_end

check_6:
        cmp ax, 0FFFEh
        jne check_last
        mov dx, offset OWNER_MAXUMB_MESSAGE
        call PRINT_STRING
        jmp owner_end

check_last:
        mov dx, 16
        mul dx
        mov si, offset OWNER_UNKNOWN
        add si, 4
        call WRD_TO_DEC
        mov dx, offset OWNER_UNKNOWN
        call PRINT_STRING

owner_end:
        call PRINT_DSPACE
        jmp block_size_label

jump_up:

```



```
jmp MCB_loop
```

```
block_size_label:
```

```
    ; Вывод размера участка
```

```
    mov dx, offset BLOCK_SIZE_MESSAGE
```

```
    call PRINT_STRING
```

```
    mov ax, es:[3]
```

```
    mov bx, 10h
```

```
    mul bx
```

```
    mov si, offset BLOCK_SIZE
```

```
    add si, 6
```

```
    call WRD_TO_DEC
```

```
    mov dx, offset BLOCK_SIZE
```

```
    call PRINT_STRING
```

```
    call PRINT_DSPACE
```

```
    ; Вывод последних байт
```

```
    mov dx, offset LAST_BYTES_MESSAGE
```

```
    call PRINT_STRING
```

```
    mov cx, 8
```

```
    mov si, 8h
```

```
    mov ah, 02h
```

```
last_bytes_loop:
```

```
    mov dl, es:[si]
```

```
    int 21h
```

```
    inc si
```

```
    loop last_bytes_loop
```

```
    call PRINT_ENDL
```

```
    mov al, es:[0]
```

```

        cmp al, 5Ah
        je exit
        mov ax, es
        add ax, es:[3]
        inc ax
        mov es, ax
        jmp jump_up

exit:
        ret

TESTPC  ENDS
        END START ; конец модуля, START – точка входа

```

ПРИЛОЖЕНИЕ Г

ИСХОДНЫЙ КОД ПРОГРАММЫ LAB_4.ASM

TESTPC SEGMENT

ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING

ORG 100H

START: JMP BEGIN

; ДАННЫЕ

ACCESSIBLE_MEMORY_MESSAGE db 'Accessible memory: ', '\$'

ACCESSIBLE_MEMORY db ' ', 13, 10, '\$'

EXPANDED_MEMORY_MESSAGE db 'Expanded memory: ', '\$'

EXPANDED_MEMORY db ' ', 13, 10, '\$'

MCB_TYPE_MESSAGE db 'MCB type: ', '\$'

MCB_TYPE db ' ', '\$'

OWNER_MESSAGE db 'Owner: ', '\$'

OWNER_FREE_MESSAGE db 'Free', '\$'

OWNER_XMSUMB_MESSAGE db 'OS XMS UMB', '\$'

OWNER_DRIVER_MESSAGE db 'Исключенная верхняя память
драйвера; даже гуглить правильный перевод не буду', '\$'

OWNER_MSDOS_MESSAGE db 'MS DOS', '\$'

OWNER_CONTROLLER_MESSAGE db '386MAX UMB Controller', '\$'

OWNER_BLOCKED_MESSAGE db 'Blocked by 386MAX UMB', '\$'

OWNER_MAXUMB_MESSAGE db '386MAX UMB', '\$'

OWNER_UNKNOWN db ' ', '\$'

BLOCK_SIZE_MESSAGE db 'Size: ', '\$'

BLOCK_SIZE db ' ', '\$'

LAST_BYTES_MESSAGE db 'Last bytes: ', '\$'

```

FREE_OK_MESSAGE                db 'Free: OK', 13, 10, '$'
FREE_ERROR_MESSAGE             db 'Free: ERROR', 13, 10, '$'

```

```

ALLOCATION_OK_MESSAGE           db 'Allocation: OK', 13, 10, '$'
ALLOCATION_ERROR_MESSAGE        db 'Allocation: ERROR', 13,
10, '$'

```

```

; ПРОЦЕДУРЫ

```

```

;-----

```

```

TETR_TO_HEX PROC near

```

```

; младшая шестн. цифра AL в шестн. цифру ASCII

```

```

    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 шестн. числа ASCII 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

```

```

    ;xchg al, ah        ;; а теперь наоборот!

```

```

    ret

```

```

BYTE_TO_HEX ENDP

```

```

;-----
WRD_TO_HEX PROC near
; перевод в 16 с/с 16-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 10 с/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_1
        or AL,30h
        mov [SI],AL
        dec si

end_1:   pop DX
        pop CX
        ret
BYTE_TO_DEC ENDP

```

```

;-----
WRD_TO_DEC PROC near
        push ax
        push bx

        mov bx, 10

div_loop:
        div bx
        add dl, 30h
        mov [si], dl
        dec si
        mov dx, 0
        cmp ax, 0
        jne div_loop

        pop bx
        pop ax
        ret
WRD_TO_DEC ENDP

```

```

;-----
PRINT_STRING PROC near
; Просто выводит строку с уже указанным в dx смещением, очень
сложная функция
    push ax

    mov ah, 09h
    int 21h

    pop ax
    ret
PRINT_STRING ENDP

;-----
PRINT_WORD PROC near
; Выводит регистр AX
    push ax
    push dx

    mov dl, ah
    mov ah, 02h
    int 21h

    mov dl, al
    int 21h

    pop dx
    pop ax
    ret
PRINT_WORD ENDP

;-----

```

```
PRINT_ENDL PROC near
; Выводит 13, 10
    push ax
    push dx

    mov dl, 13
    mov ah, 02h
    int 21h

    mov dl, 10
    int 21h

    pop dx
    pop ax
    ret
PRINT_ENDL ENDP
```

```
PRINT_DSPACE PROC near
; Выводит два пробела
    push ax
    push dx

    mov dl, ' '
    mov ah, 02h
    int 21h

    mov dl, ' '
    int 21h

    pop dx
    pop ax
    ret
PRINT_DSPACE ENDP
```



```

;-----
; КОД
BEGIN:

    ; Размер доступной памяти
    mov dx, offset ACCESSIBLE_MEMORY_MESSAGE
    call PRINT_STRING

    mov bx, 0FFFFh
    mov ah, 4Ah
    int 21h

    mov ax, bx
    mov bx, 16
    mul bx

    mov si, offset ACCESSIBLE_MEMORY
    add si, 6
    call WRD_TO_DEC
    mov dx, offset ACCESSIBLE_MEMORY
    call PRINT_STRING

    ; Размер расширенной памяти памяти
    mov dx, offset EXPANDED_MEMORY_MESSAGE
    call PRINT_STRING

    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 EXPANDED_MEMORY
    add si, 6
    mov dx, 0
    call WRD_TO_DEC
    mov dx, offset EXPANDED_MEMORY
    call PRINT_STRING
    call PRINT_ENDL

; Запрос новых 64 Кб = 4096 параграфов
    mov bx, 4096
    mov ah, 48h
    int 21h

    jc allocation_error
allocation_ok:
    mov dx, offset ALLOCATION_OK_MESSAGE
    jmp print_allocation_message

allocation_error:
    mov dx, offset ALLOCATION_ERROR_MESSAGE

print_allocation_message:
    call PRINT_STRING

; Освобождение
; В конце сегмента, вроде, стек должен валяться,

```

```

; поэтому 65536/16 = 4096 параграфов
mov bx, 4096
mov ah, 4Ah
int 21h

jc free_error
free_ok:
mov dx, offset FREE_OK_MESSAGE
jmp print_free_message

free_error:
mov dx, offset FREE_ERROR_MESSAGE

print_free_message:
call PRINT_STRING
call PRINT_ENDL

; Цепочка блоков управления памятью
mov ah, 52h
int 21h
mov es, es:[bx - 2]

MCB_loop:
; Вывод типа MCB
mov dx, offset MCB_TYPE_MESSAGE
call PRINT_STRING

mov al, es:[0]
call BYTE_TO_HEX
xchg ah, al
mov dl, ah
mov ah, 02h

```

```

        push ax
        int 21h
        pop ax
        mov dl, al
        int 21h
        call PRINT_DSPACE

        ; Вывод владельца
        mov dx, offset OWNER_MESSAGE
        call PRINT_STRING

        mov ax, es:[1]
check_0:
        cmp ax, 0000h
        jne check_1
        mov dx, offset OWNER_FREE_MESSAGE
        call PRINT_STRING
        jmp owner_end
check_1:
        cmp ax, 0006h
        jne check_2
        mov dx, offset OWNER_XMSUMB_MESSAGE
        call PRINT_STRING
        jmp owner_end
check_2:
        cmp ax, 0007h
        jne check_3
        mov dx, offset OWNER_DRIVER_MESSAGE
        call PRINT_STRING
        jmp owner_end
check_3:
        cmp ax, 0008h
        jne check_4

```

```

        mov dx, offset OWNER_MSDOS_MESSAGE
        call PRINT_STRING
        jmp owner_end

check_4:
        cmp ax, 0FFFAh
        jne check_5
        mov dx, offset OWNER_CONTROLLER_MESSAGE
        call PRINT_STRING
        jmp owner_end

check_5:
        cmp ax, 0FFFDh
        jne check_6
        mov dx, offset OWNER_BLOCKED_MESSAGE
        call PRINT_STRING
        jmp owner_end

check_6:
        cmp ax, 0FFFEh
        jne check_last
        mov dx, offset OWNER_MAXUMB_MESSAGE
        call PRINT_STRING
        jmp owner_end

check_last:
        mov dx, 16
        mul dx
        mov si, offset OWNER_UNKNOWN
        add si, 4
        call WRD_TO_DEC
        mov dx, offset OWNER_UNKNOWN
        call PRINT_STRING

owner_end:
        call PRINT_DSPACE
        jmp block_size_label

jump_up:

```

```
jmp MCB_loop
```

```
block_size_label:
```

```
    ; Вывод размера участка
```

```
    mov dx, offset BLOCK_SIZE_MESSAGE
```

```
    call PRINT_STRING
```

```
    mov ax, es:[3]
```

```
    mov bx, 10h
```

```
    mul bx
```

```
    mov si, offset BLOCK_SIZE
```

```
    add si, 6
```

```
    call WRD_TO_DEC
```

```
    mov dx, offset BLOCK_SIZE
```

```
    call PRINT_STRING
```

```
    call PRINT_DSPACE
```

```
    ; Вывод последних байт
```

```
    mov dx, offset LAST_BYTES_MESSAGE
```

```
    call PRINT_STRING
```

```
    mov cx, 8
```

```
    mov si, 8h
```

```
    mov ah, 02h
```

```
last_bytes_loop:
```

```
    mov dl, es:[si]
```

```
    int 21h
```

```
    inc si
```

```
    loop last_bytes_loop
```

```
    call PRINT_ENDL
```

```
    mov al, es:[0]
```

```

        cmp al, 5Ah
        je exit
        mov ax, es
        add ax, es:[3]
        inc ax
        mov es, ax
        jmp jump_up

exit:
        ret

TESTPC  ENDS
        END START ; конец модуля, START – точка входа

```