

**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. 8383

\_\_\_\_\_

Бессуднов Г. И.

Преподаватель

\_\_\_\_\_

Ефремов М. А.

Санкт-Петербург

2020

### **Цель работы.**

Исследование структур данных и работы функций управления памятью ядра операционной системы.

### **Основные теоретические сведения.**

Учет занятой и свободной памяти ведется при помощи списка блоков управления памятью MCB. MCB занимает 16 байт (параграф) и располагается всегда с адреса, кратного 16 и находится в адресном пространстве непосредственно перед тем участком памяти, которым он управляет.

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

В результате выполнения функции 52h прерывания int 21h 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.

### **Выполнение работы.**

В ходе выполнения лабораторной работы был написан и отлажен программный модуль типа .COM, который выбирает и печатает на экран следующую информацию:

- 1) Количество доступной памяти;
- 2) Размер расширенной памяти;
- 3) Выводит цепочку блоков управления памяти.

Результат работы программы представлен на рис. 1, исходный код программы в Приложении А.

```

C:\>LR3_1.COM
Aviable memory: 648912 bytes
Extended memory: 15360 kbytes
MCB 1
MS DOS
Size: 16 bytes

MCB 2
Empty area
Size: 64 bytes

MCB 3
0004
Size: 256 bytes

MCB 4
1029
Size: 144 bytes

MCB 5
1029
Size: 648912 bytes
LR3_1

```

Рисунок 1 – Результат работы программы

Далее программа была изменена и было добавлено освобождение памяти. Исходный код программы представлен в Приложении Б, результат работы на рис. 2.

```

C:\>LR3_2.COM
Aviable memory: 648912 bytes
Memory free success
Extended memory: 15360 kbytes
MCB list:
MS DOS
Size: 16 bytes

Empty area
Size: 64 bytes

0004
Size: 256 bytes

1029
Size: 144 bytes

1029
Size: 1392 bytes
LR3_2
Empty area
Size: 647504 bytes
^=/δ. s

```

Рисунок 2 – Результат работы 2 версии программы

Видно, что освобожденная часть памяти находится в новом последнем блоке управления памятью.

Программа была модифицирована еще раз: теперь после освобождения памяти добавлен запрос выделения 64 КБ памяти. Результат работы программы на рис. 3, исходный код в Приложении В.

```

64KB request success
Extended memory: 15360 kbytes
MCB list:
MS DOS
Size: 16 bytes

Empty area
Size: 64 bytes

0004
Size: 256 bytes

1029
Size: 144 bytes

1029
Size: 1456 bytes
LR3_3
1029
Size: 65536 bytes
LR3_3
Empty area
Size: 581888 bytes

```

Рисунок 3 – Результат работы 3 версии программы

Из результатов можно сделать вывод о том, что теперь освобожденная память находится в последнем блоке управления, а в предпоследнем блоке находится выделенная память.

Программа была изменена в последний раз, теперь 64 кБ запрашивается до освобождения памяти. Результат работы приведен на рис. 4, исходный код в Приложении Г. Из результатов видно, что память выделить не удалось.

```

C:\>LR3_4.COM
Aviable memory: 648912 bytes
64KB request unsuccess
Memory free success
Extended memory: 15360 kbytes
MCB list:
MS DOS
Size: 16 bytes

Empty area
Size: 64 bytes

0004
Size: 256 bytes

1029
Size: 144 bytes

1029
Size: 1456 bytes
LR3_4
Empty area
Size: 647440 bytes
LR3_3

```

Рисунок 4 – Результат работы 4 версии программы

## **Контрольные вопросы**

Ниже приведены ответы на контрольные вопросы:

1. Что означает «доступный объем» памяти?

Максимальный объем памяти в системе, который доступен для запуска и исполнения программ.

2. Где МСВ блок вашей программы в списке?

В первой, второй и четвертой версии программы - это 5-ый блок и 4-ый (определено по полю владельца блока). В 3-ей версии еще и 6-ой, так как память под него была выделена в процессе исполнения программы.

3. Какой размер памяти занимает программа в каждом случае?

В первом случае программа занимает - 648912 байт. Во втором случае программа занимает 1392 байт, так как память была успешно освобождена. В третьем - 1456 байт и 65536 байт, так как они были запрошены и успешно выделены. В четвертом случае - 1456 байт, так как запрошенная память выделена не была. Во всех случаях дополнительно еще выделено 144 байта.

## **Выводы.**

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

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПЕРВОЙ ВЕРСИИ ПРОГРАММЫ

```
TESTPC SEGMENT
ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
ORG 100H
START: JMP BEGIN
NEW_LINE db 13, 10, '$'
MES_AVIABLE db "Aviable memory: $"
MES_BYTES db " bytes", 13, 10, '$'
MES_EXTENDED_MEM db "Extended memory: $"
MES_KBYTES db " kbytes", 13, 10, '$'
MES_MCB db "MCB $"
MES_FREE db "Empty area$"
MES_OS_XMS db "OS XMS UMB$"
MES_TOP_MEM db "Top memory$"
MES_DOS db "MS DOS$"
MES_BLOCK db "Control block 386MAX UMB$"
MES_BLOCKED db "Blocked 386MAX$"
MES_386MAX db "386MAX UMB$"
MES_SIZE db 13, 10, "Size: $"

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

WRITE_DEC PROC near
    push AX
    push BX
    push CX
```

```

        push DX
        xor CX,CX
        mov BX,10
loop_bd:
        div BX
        push DX
        xor DX,DX
        inc CX
        cmp AX,0h
        jnz loop_bd
PRINT_num:
        pop DX
        or DL,30h
        call WRITE_SYMBOL
        loop PRINT_num
        pop DX
        pop CX
        pop BX
        pop AX
        ret
WRITE_DEC ENDP

WRITE_SYMBOL PROC near
        push AX
        mov AH, 02H
        int 21H
        pop AX
        ret
WRITE_SYMBOL ENDP

WRITE_STRING PROC near
        push AX
        mov AH, 09H
        int 21H
        pop AX
        ret
WRITE_STRING ENDP

WRITE_HEX PROC near
        push AX
        mov AL, AH
        call BYTE_TO_HEX
        mov DL, AH
        call WRITE_SYMBOL
        mov DL, AL
        call WRITE_SYMBOL
        pop AX
        call BYTE_TO_HEX
        mov DL, AH
        call WRITE_SYMBOL

```

```

    mov DL, AL
    call WRITE_SYMBOL
    ret
WRITE_HEX ENDP

BEGIN:
    ;-----AVAILABLE_MEM-----
    mov DX, offset MES_AVIABLE
    call WRITE_STRING
    mov AH, 4AH
    mov BX, 0FFFFH
    int 21H
    mov AX, BX
    mov BX, 10H
    mul BX
    call WRITE_DEC
    mov DX, offset MES_BYTES
    call WRITE_STRING
    ;-----

    ;-----EXTENDED-----
    mov DX, offset MES_EXTENDED_MEM
    call WRITE_STRING
    mov AL, 30H
    out 70H, AL
    in AL, 71H
    mov BL, AL
    mov AL, 31H
    out 70H, AL
    in AL, 71H
    mov BH, AL
    mov AX, BX
    xor DX, DX
    call WRITE_DEC
    mov DX, offset MES_KBYTES
    call WRITE_STRING
    ;-----

    ;-----MCB-----
    xor CX, CX
    mov AH, 52H
    int 21H
    mov AX, ES:[BX-2]
    mov ES, AX
GET_MCB:
    inc CX
    mov DX, offset MES_MCB
    push CX
    call WRITE_STRING
    xor DX, DX

```



```

    mov AX,CX
    call WRITE_DEC
    mov DX, offset NEW_LINE
    call WRITE_STRING
    xor AX,AX
    mov AL,ES:[0H]
    push AX
    mov AX,ES:[1H]
    cmp AX,0H
    je PRINT_FREE
    cmp AX,6H
    je PRINT_OS_XMS
    cmp AX,7H
    je PRINT_TOP
    cmp AX,8H
    je PRINT_DOS
    cmp AX,0FFFAH
    je PRINT_BLOCK
    cmp AX,0FFFDH
    je PRINT_BLOCKED
    cmp AX,0FFFEH
    je PRINT_386MAX
    xor DX,DX
    call WRITE_HEX
    jmp GET_SIZE
PRINT_FREE:
    mov DX, offset MES_FREE
    jmp PRINT
PRINT_OS_XMS:
    mov DX, offset MES_OS_XMS
    jmp PRINT
PRINT_TOP:
    mov DX, offset MES_TOP_MEM
    jmp PRINT
PRINT_DOS:
    mov DX, offset MES_DOS
    jmp PRINT
PRINT_BLOCK:
    mov DX, offset MES_DOS
    jmp PRINT
PRINT_BLOCKED:
    mov DX, offset MES_DOS
    jmp PRINT
PRINT_386MAX:
    mov DX, offset MES_386MAX
PRINT:
    call WRITE_STRING
GET_SIZE:
    mov DX, offset MES_SIZE
    call WRITE_STRING

```

```

    mov AX,ES:[3H]
    mov BX,10H
    mul BX
    call WRITE_DEC
    mov DX, offset MES_BYTES
    call WRITE_STRING
    xor SI,SI
    mov CX,8
GET_LAST:
    mov DL,ES:[SI+8H]
    call WRITE_SYMBOL
    inc SI
    loop GET_LAST
    mov DX,offset NEW_LINE
    call WRITE_STRING
    mov AX,ES:[3H]
    mov BX,ES
    add BX,AX
    inc BX
    mov ES,BX
    pop AX
    pop CX
    cmp AL,5AH
    je END_PRINT
    jmp GET_MCB
END_PRINT:
    ;-----
    xor AL,AL
    mov AH,4CH
    int 21H
TESTPC ENDS
END START

```

## ПРИЛОЖЕНИЕ Б

### ИСХОДНЫЙ КОД ВТОРОЙ ВЕРСИИ ПРОГРАММЫ

```
TESTPC SEGMENT
ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
ORG 100H
START: JMP BEGIN
NEW_LINE db 13, 10, '$'
MES_AVIABLE db "Aviable memory: $"
MES_BYTES db " bytes", 13, 10, '$'
MES_EXTENDED_MEM db "Extended memory: $"
MES_KBYTES db " kbytes", 13, 10, '$'
MES_MCB db "MCB list: $"
MES_FREE db "Empty area$"
MES_OS_XMS db "OS XMS UMB$"
MES_TOP_MEM db "Top memory$"
MES_DOS db "MS DOS$"
MES_BLOCK db "Control block 386MAX UMB$"
MES_BLOCKED db "Blocked 386MAX$"
MES_386MAX db "386MAX UMB$"
MES_SIZE db 13, 10, "Size: $"
MES_FREE_SUCCES db "Memory free success", 13, 10, '$'
MES_FREE_ERROR db "Memory free unsuccess", 13, 10, '$'

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

WRITE_DEC PROC near
    push AX
    push BX
```

```

        push CX
        push DX
        xor CX,CX
        mov BX,10
loop_bd:
        div BX
        push DX
        xor DX,DX
        inc CX
        cmp AX,0h
        jnz loop_bd
writing_num:
        pop DX
        or DL,30h
        call WRITE_SYMBOL
        loop writing_num
        pop DX
        pop CX
        pop BX
        pop AX
        ret
WRITE_DEC ENDP

WRITE_SYMBOL PROC near
        push AX
        mov AH, 02H
        int 21H
        pop AX
        ret
WRITE_SYMBOL ENDP

WRITE_STRING PROC near
        push AX
        mov AH, 09H
        int 21H
        pop AX
        ret
WRITE_STRING ENDP

WRITE_HEX PROC near
        push AX
        mov AL, AH
        call BYTE_TO_HEX
        mov DL, AH
        call WRITE_SYMBOL
        mov DL, AL
        call WRITE_SYMBOL
        pop AX
        call BYTE_TO_HEX
        mov DL, AH

```

```

    call WRITE_SYMBOL
    mov DL, AL
    call WRITE_SYMBOL
    ret
WRITE_HEX ENDP

BEGIN:
    ;-----AVAILABLE_MEM-----
    mov DX, offset MES_AVIABLE
    call WRITE_STRING
    mov AH, 4AH
    mov BX, 0FFFFH
    int 21H
    mov AX, BX
    mov BX, 10H
    mul BX
    call WRITE_DEC
    mov DX, offset MES_BYTES
    call WRITE_STRING
    ;-----

    mov BX, offset LR3_END
    add BX, 10FH
    shr BX, 1
    shr BX, 1
    shr BX, 1
    shr BX, 1
    mov AH, 4AH
    int 21H
    jnc SUCCESS
    mov DX, offset MES_FREE_ERROR
    call WRITE_STRING
    jmp END_FREE

SUCCESS:
    mov DX, offset MES_FREE_SUCCES
    call WRITE_STRING
END_FREE:

    ;-----EXTENDED_MEM-----
    mov DX, offset MES_EXTENDED_MEM
    call WRITE_STRING
    mov AL, 30H
    out 70H, AL
    in AL, 71H
    mov BL, AL
    mov AL, 31H
    out 70H, AL
    in AL, 71H
    mov BH, AL

```

```

mov AX,BX
xor DX,DX
call WRITE_DEC
mov DX, offset MES_KBYTES
call WRITE_STRING
;-----

;-----MCB-----
xor CX,CX
mov AH,52H
int 21H
mov AX,ES:[BX-2]
mov ES,AX
mov DX, offset MES_MCB
call WRITE_STRING
GET_MCB:
inc CX
push CX
xor DX,DX
mov AX,CX
mov DX, offset NEW_LINE
call WRITE_STRING
xor AX,AX
mov AL,ES:[0H]
push AX
mov AX,ES:[1H]
cmp AX,0H
je PRINTING_FREE
cmp AX,6H
je PRINTING_OS_XMS
cmp AX,7H
je PRINTING_TOP
cmp AX,8H
je PRINTING_DOS
cmp AX,0FFFAH
je PRINTING_BLOCK
cmp AX,0FFFDH
je PRINTING_BLOCKED
cmp AX,0FFFEH
je PRINTING_386MAX
xor DX,DX
call WRITE_HEX
jmp GET_SIZE
PRINTING_FREE:
mov DX, offset MES_FREE
jmp PRINTING
PRINTING_OS_XMS:
mov DX, offset MES_OS_XMS
jmp PRINTING
PRINTING_TOP:

```

```

        mov DX, offset MES_TOP_MEM
        jmp PRINTING
PRINTING_DOS:
        mov DX, offset MES_DOS
        jmp PRINTING
PRINTING_BLOCK:
        mov DX, offset MES_DOS
        jmp PRINTING
PRINTING_BLOCKED:
        mov DX, offset MES_DOS
        jmp PRINTING
PRINTING_386MAX:
        mov DX, offset MES_386MAX
PRINTING:
        call WRITE_STRING
GET_SIZE:
        mov DX, offset MES_SIZE
        call WRITE_STRING
        mov AX,ES:[3H]
        mov BX,10H
        mul BX
        call WRITE_DEC
        mov DX, offset MES_BYTES
        call WRITE_STRING
        xor SI,SI
        mov CX,8
GET_LAST:
        mov DL,ES:[SI+8H]
        call WRITE_SYMBOL
        inc SI
        loop GET_LAST
        mov AX,ES:[3H]
        mov BX,ES
        add BX,AX
        inc BX
        mov ES,BX
        pop AX
        pop CX
        cmp AL,5AH
        je END_PRINTING
        jmp GET_MCB
END_PRINTING:
        ;-----

        xor AL,AL
        mov AH,4CH
        int 21H
        DW 128 dup(0)
LR3_END:
TESTPC ENDS

```

END START

## ПРИЛОЖЕНИЕ В

### ИСХОДНЫЙ КОД ТРЕТЬЕЙ ВЕРСИИ ПРОГРАММЫ

```
TESTPC SEGMENT
ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
ORG 100H
START: JMP BEGIN
NEW_LINE db 13, 10, '$'
MES_AVIABLE db "Aviable memory: $"
MES_BYTES db " bytes", 13, 10, '$'
MES_EXTENDED_MEM db "Extended memory: $"
MES_KBYTES db " kbytes", 13, 10, '$'
MES_MCB db "MCB list: $"
MES_FREE db "Empty area$"
MES_OS_XMS db "OS XMS UMB$"
MES_TOP db "Top memory$"
MES_DOS db "MS DOS$"
MES_BLOCK db "Control block 386MAX UMB$"
MES_BLOCKED db "Blocked 386MAX$"
MES_386MAX db "386MAX UMB$"
MES_SIZE db 13, 10, "Size: $"
MES_FREE_SUCCESS db "Memory free success", 13, 10, '$'
MES_FREE_ERROR db "Memory free unsuccess", 13, 10, '$'
MES_REQUEST_SUCCESS db "64KB request success", 13, 10, '$'
MES_REQUEST_ERROR db "64KB request unsuccess", 13, 10, '$'

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
```



```

WRITE_DEC PROC near
    push AX
    push BX
    push CX
    push DX
    xor CX,CX
    mov BX,10
loop_bd:
    div BX
    push DX
    xor DX,DX
    inc CX
    cmp AX,0h
    jnz loop_bd
writing_num:
    pop DX
    or DL,30h
    call WRITE_SYMBOL
    loop writing_num
    pop DX
    pop CX
    pop BX
    pop AX
    ret
WRITE_DEC ENDP

WRITE_SYMBOL PROC near
    push AX
    mov AH, 02H
    int 21H
    pop AX
    ret
WRITE_SYMBOL ENDP

WRITE_STRING PROC near
    push AX
    mov AH, 09H
    int 21H
    pop AX
    ret
WRITE_STRING ENDP

WRITE_HEX PROC near
    push AX
    mov AL, AH
    call BYTE_TO_HEX
    mov DL, AH
    call WRITE_SYMBOL
    mov DL, AL
    call WRITE_SYMBOL

```

```

    pop AX
    call BYTE_TO_HEX
    mov DL, AH
    call WRITE_SYMBOL
    mov DL, AL
    call WRITE_SYMBOL
    ret
WRITE_HEX ENDP

BEGIN:
    ;-----AVAILABLE_MEM-----
    mov DX, offset MES_AVIABLE
    call WRITE_STRING
    mov AH, 4AH
    mov BX, 0FFFFH
    int 21H
    mov AX, BX
    mov BX, 10H
    mul BX
    call WRITE_DEC
    mov DX, offset MES_BYTES
    call WRITE_STRING
    ;-----

    ;-----FREE_MEM-----
    mov BX, offset TESTPC_END
    add BX, 10FH
    shr BX, 1
    shr BX, 1
    shr BX, 1
    shr BX, 1
    mov AH, 4AH
    int 21H
    jnc SUCCES
    mov DX, offset MES_FREE_ERROR
    call WRITE_STRING
    jmp END_FREE

SUCCES:
    mov DX, offset MES_FREE_SUCCES
    call WRITE_STRING
END_FREE:
    ;-----

    ;-----REQUEST-----
    mov BX, 1000H
    mov AH, 48H
    int 21H
    jnc WRITE_REQUEST
    mov DX, offset MES_REQUEST_ERROR

```

```

        call WRITE_STRING
        jmp END_REQUEST
WRITE_REQUEST:
        mov DX, offset MES_REQUEST_SUCCES
        call WRITE_STRING
END_REQUEST:
        ;-----

        ;-----EXTENDED_MEM-----
        mov DX, offset MES_EXTENDED_MEM
        call WRITE_STRING
        mov AL,30H
        out 70H,AL
        in AL,71H
        mov BL,AL
        mov AL,31H
        out 70H,AL
        in AL,71H
        mov BH,AL
        mov AX,BX
        xor DX,DX
        call WRITE_DEC
        mov DX, offset MES_KBYTES
        call WRITE_STRING
        ;-----

        ;-----MCB-----
        xor CX,CX
        mov AH,52H
        int 21H
        mov AX,ES:[BX-2]
        mov ES,AX
        mov DX, offset MES_MCB
        call WRITE_STRING
GET_MCB:
        inc CX
        push CX
        xor DX,DX
        mov AX,CX
        mov DX, offset NEW_LINE
        call WRITE_STRING
        xor AX,AX
        mov AL,ES:[0H]
        push AX
        mov AX,ES:[1H]
        cmp AX,0H
        je PRINTING_FREE
        cmp AX,6H
        je PRINTING_OS_XMS
        cmp AX,7H

```

```

        je PRINTING_TOP
        cmp AX,8H
        je PRINTING_DOS
        cmp AX,0FFFAH
        je PRINTING_BLOCK
        cmp AX,0FFFDH
        je PRINTING_BLOCKED
        cmp AX,0FFFEH
        je PRINTING_386MAX
        xor DX,DX
        call WRITE_HEX
        jmp GET_SIZE
PRINTING_FREE:
        mov DX, offset MES_FREE
        jmp PRINTING
PRINTING_OS_XMS:
        mov DX, offset MES_OS_XMS
        jmp PRINTING
PRINTING_TOP:
        mov DX, offset MES_TOP
        jmp PRINTING
PRINTING_DOS:
        mov DX, offset MES_DOS
        jmp PRINTING
PRINTING_BLOCK:
        mov DX, offset MES_DOS
        jmp PRINTING
PRINTING_BLOCKED:
        mov DX, offset MES_DOS
        jmp PRINTING
PRINTING_386MAX:
        mov DX, offset MES_386MAX
PRINTING:
        call WRITE_STRING
GET_SIZE:
        mov DX, offset MES_SIZE
        call WRITE_STRING
        mov AX,ES:[3H]
        mov BX,10H
        mul BX
        call WRITE_DEC
        mov DX, offset MES_BYTES
        call WRITE_STRING
        xor SI,SI
        mov CX,8
GET_LAST:
        mov DL,ES:[SI+8H]
        call WRITE_SYMBOL
        inc SI
        loop GET_LAST

```

```

    mov AX,ES:[3H]
    mov BX,ES
    add BX,AX
    inc BX
    mov ES,BX
    pop AX
    pop CX
    cmp AL,5AH
    je END_PRINTING
    jmp GET_MCB
END_PRINTING:
    ;-----
    xor AL,AL
    mov AH,4CH
    int 21H
    DW 128 dup(0)
TESTPC_END:
TESTPC ENDS
END START

```

## ПРИЛОЖЕНИЕ Г

### ИСХОДНЫЙ КОД ЧЕТВЕРТОЙ ВЕРСИИ ПРОГРАММЫ

```
TESTPC SEGMENT
ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
ORG 100H
START: JMP BEGIN
NEW_LINE db 13, 10, '$'
MES_AVIABLE db "Aviable memory: $"
MES_BYTES db " bytes", 13, 10, '$'
MES_EXTENDED_MEM db "Extended memory: $"
MES_KBYTES db " kbytes", 13, 10, '$'
MES_MCB db "MCB list: $"
MES_FREE db "Empty area$"
MES_OS_XMS db "OS XMS UMB$"
MES_TOP db "Top memory$"
MES_DOS db "MS DOS$"
MES_BLOCK db "Control block 386MAX UMB$"
MES_BLOCKED db "Blocked 386MAX$"
MES_386MAX db "386MAX UMB$"
MES_SIZE db 13, 10, "Size: $"
MES_FREE_SUCCES db "Memory free success", 13, 10, '$'
MES_FREE_ERROR db "Memory free unsuccess", 13, 10, '$'
MES_REQUEST_SUCCES db "64KB request success", 13, 10, '$'
MES_REQUEST_ERROR db "64KB request unsuccess", 13, 10, '$'

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
```

```

WRITE_DEC PROC near
    push AX
    push BX
    push CX
    push DX
    xor CX,CX
    mov BX,10
loop_bd:
    div BX
    push DX
    xor DX,DX
    inc CX
    cmp AX,0h
    jnz loop_bd
writing_num:
    pop DX
    or DL,30h
    call WRITE_SYMBOL
    loop writing_num
    pop DX
    pop CX
    pop BX
    pop AX
    ret
WRITE_DEC ENDP

WRITE_SYMBOL PROC near
    push AX
    mov AH, 02H
    int 21H
    pop AX
    ret
WRITE_SYMBOL ENDP

WRITE_STRING PROC near
    push AX
    mov AH, 09H
    int 21H
    pop AX
    ret
WRITE_STRING ENDP

WRITE_HEX PROC near
    push AX
    mov AL, AH
    call BYTE_TO_HEX
    mov DL, AH
    call WRITE_SYMBOL
    mov DL, AL
    call WRITE_SYMBOL

```

```

    pop AX
    call BYTE_TO_HEX
    mov DL, AH
    call WRITE_SYMBOL
    mov DL, AL
    call WRITE_SYMBOL
    ret
WRITE_HEX ENDP

BEGIN:
    ;-----AVAILABLE_MEM-----
    mov DX, offset MES_AVIABLE
    call WRITE_STRING
    mov AH, 4AH
    mov BX, 0FFFFH
    int 21H
    mov AX, BX
    mov BX, 10H
    mul BX
    call WRITE_DEC
    mov DX, offset MES_BYTES
    call WRITE_STRING
    ;-----

    ;-----REQUEST-----
    mov BX, 1000H
    mov AH, 48H
    int 21H
    jnc WRITE_REQUEST
    mov DX, offset MES_REQUEST_ERROR
    call WRITE_STRING
    jmp END_REQUEST
WRITE_REQUEST:
    mov DX, offset MES_REQUEST_SUCCES
    call WRITE_STRING
END_REQUEST:
    ;-----

    ;-----FREE-----
    mov BX, offset LR3_END
    add BX, 10FH
    shr BX, 1
    shr BX, 1
    shr BX, 1
    shr BX, 1
    mov AH, 4AH
    int 21H
    jnc SUCCES
    mov DX, offset MES_FREE_ERROR
    call WRITE_STRING

```



```

        jmp END_FREE

SUCCES:
        mov DX, offset MES_FREE_SUCCES
        call WRITE_STRING
END_FREE:
        ;-----

        ;-----EXTENDED_MEM-----
        mov DX, offset MES_EXTENDED_MEM
        call WRITE_STRING
        mov AL,30H
        out 70H,AL
        in AL,71H
        mov BL,AL
        mov AL,31H
        out 70H,AL
        in AL,71H
        mov BH,AL
        mov AX,BX
        xor DX,DX
        call WRITE_DEC
        mov DX, offset MES_KBYTES
        call WRITE_STRING
        ;-----

        ;-----MCB-----
        xor CX,CX
        mov AH,52H
        int 21H
        mov AX,ES:[BX-2]
        mov ES,AX
        mov DX, offset MES_MCB
        call WRITE_STRING
GET_MCB:
        inc CX
        push CX
        xor DX,DX
        mov AX,CX
        ;call WRITE_DEC
        mov DX, offset NEW_LINE
        call WRITE_STRING
        xor AX,AX
        mov AL,ES:[0H]
        push AX
        mov AX,ES:[1H]
        cmp AX,0H
        je PRINTING_FREE
        cmp AX,6H
        je PRINTING_OS_XMS

```

```

    cmp AX,7H
    je PRINTING_TOP
    cmp AX,8H
    je PRINTING_DOS
    cmp AX,0FFFAH
    je PRINTING_BLOCK
    cmp AX,0FFFDH
    je PRINTING_BLOCKED
    cmp AX,0FFFEH
    je PRINTING_386MAX
    xor DX,DX
    call WRITE_HEX
    jmp GET_SIZE
PRINTING_FREE:
    mov DX, offset MES_FREE
    jmp PRINTING
PRINTING_OS_XMS:
    mov DX, offset MES_OS_XMS
    jmp PRINTING
PRINTING_TOP:
    mov DX, offset MES_TOP
    jmp PRINTING
PRINTING_DOS:
    mov DX, offset MES_DOS
    jmp PRINTING
PRINTING_BLOCK:
    mov DX, offset MES_DOS
    jmp PRINTING
PRINTING_BLOCKED:
    mov DX, offset MES_DOS
    jmp PRINTING
PRINTING_386MAX:
    mov DX, offset MES_386MAX
PRINTING:
    call WRITE_STRING
GET_SIZE:
    mov DX, offset MES_SIZE
    call WRITE_STRING
    mov AX,ES:[3H]
    mov BX,10H
    mul BX
    call WRITE_DEC
    mov DX, offset MES_BYTES
    call WRITE_STRING
    xor SI,SI
    mov CX,8
GET_LAST:
    mov DL,ES:[SI+8H]
    call WRITE_SYMBOL
    inc SI

```

```

    loop GET_LAST
    mov AX,ES:[3H]
    mov BX,ES
    add BX,AX
    inc BX
    mov ES,BX
    pop AX
    pop CX
    cmp AL,5AH
    je END_PRINTING
    ;mov DX,offset NEW_LINE
    ;call WRITE_STRING
    jmp GET_MCB
END_PRINTING:
    ;-----
    xor AL,AL
    mov AH,4CH
    int 21H
    DW 128 dup(0)
LR3_END:
TESTPC ENDS
END START

```