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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Организация ЭВМ и систем»

Тема: «Исследование организации управления основной памятью»

Студент гр. 8381 _____ Переверзев Д.Е.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2019

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

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

Рисунок 1 – Структура MCB

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 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:

Выполнение работы.

Количество доступной памяти было найдено с помощью функции 4Ah прерывания 21h (т.к. был занесен заведомо больший размер памяти, чем может предоставить ОС, то в регистр BX возвратился размер доступной памяти в параграфах).

Размер расширенной памяти был определен с помощью обращения к ячейкам 30h, 31h CMOS.

Адрес первого блока управления памятью MCB) был получен с помощью функции 52h . В результате выполнения этой функции ES:BX указывает на список списков, а слово по адресу ES:[BX-2] и есть адрес самого первого блока. Тип MCB находится по смещению 00h, владелец участка памяти по смещению 01h, размер участка в параграфах по смещению 03h и по смещению 08h находятся последние 8 байт MCB, в которых могут находится системные данные.Адрес следующего блока памяти вычисляется с помощью адреса текущего блока и его размера.

Результат работы программы показан на рис. 2.

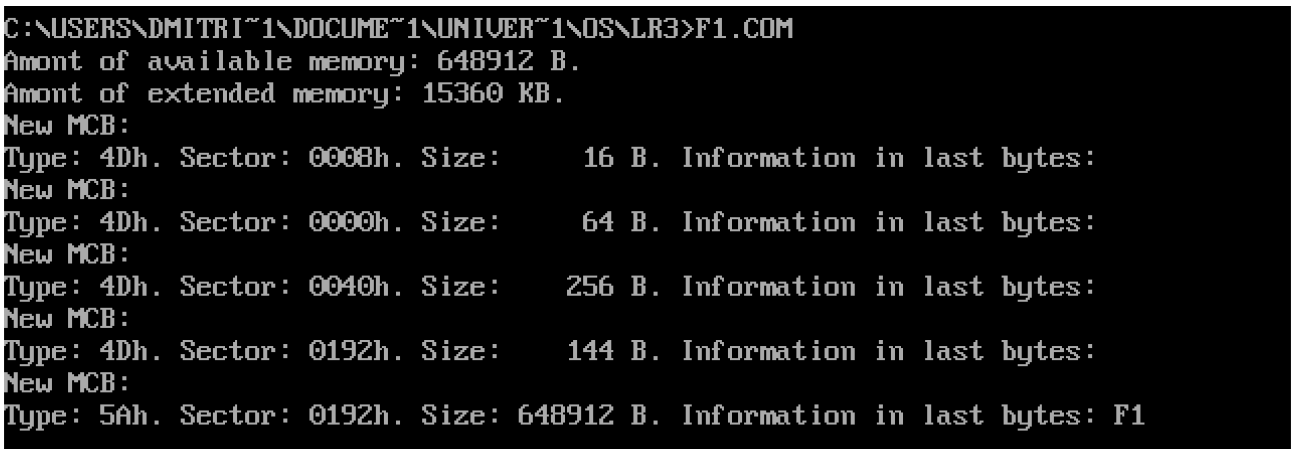

Рисунок 2 – Вывод программы f1.com

Первый блок принадлежит MS DOS т. к. в поле сектор содержит 0008h), второй блок свободный содержит 0000h). Владельцы третьего, четвертого и пятого блоков имеют сегментные адреса PSP 0040h, 0192h соответственно.

Затем программа была изменена таким образом, чтобы она освобождала память, которую она не занимает , так же с помощью функции 4Ah прерывания 21h . Результат работы программы представлен на рис. 3.

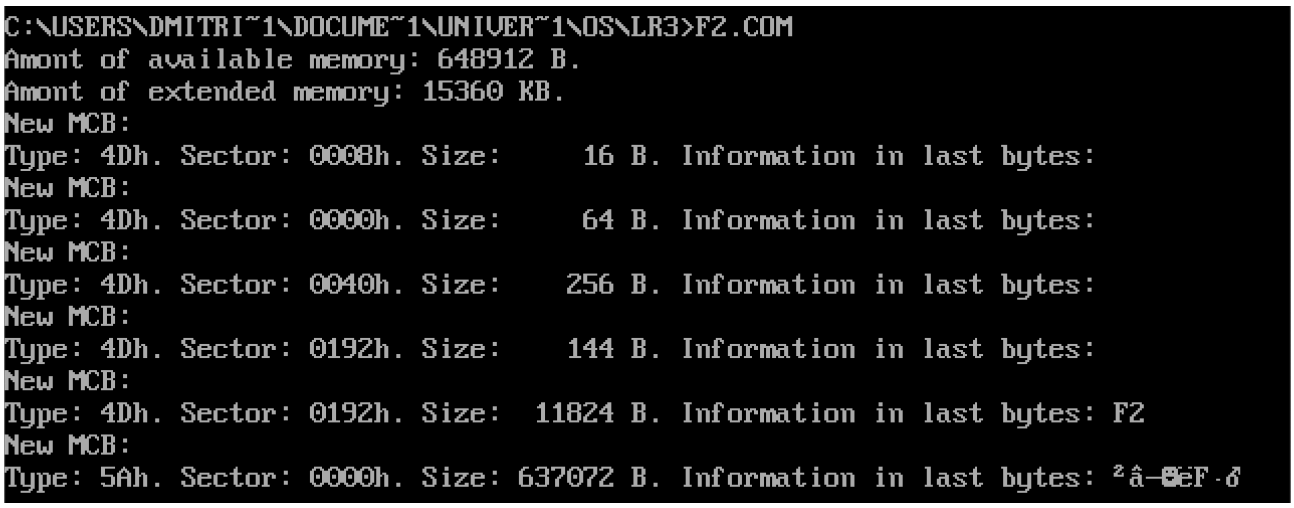

Рисунок 3 – Вывод программы f2.com

Как видно из рисунка, был создан новый пустой блок (шестой).

Далее программа была изменена так, чтобы после освобождения памяти она запрашивала 64Кб памяти функцией 4Ah . Результат работы программы представлен на рис. 4

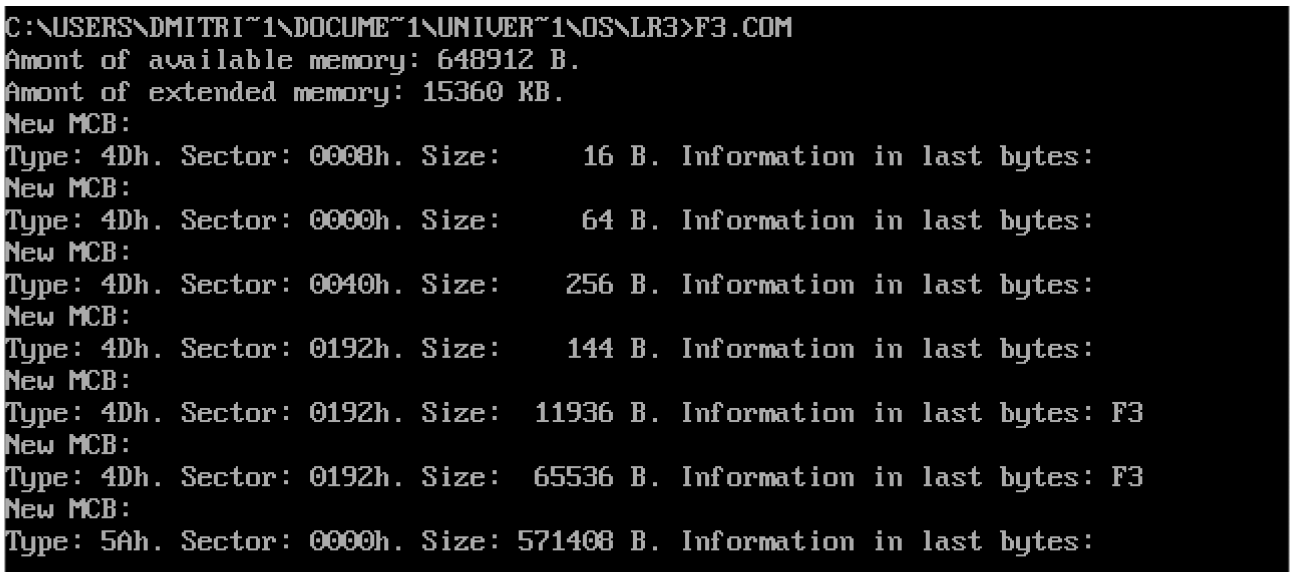

Рисунок 4 – Вывод программы f3.com

В конце был изменен начальный вариант программы запросом 64Кб памяти функцией 48h прерывания 21h до освобождения памяти. Результат работы программы представлен на рис. 5.

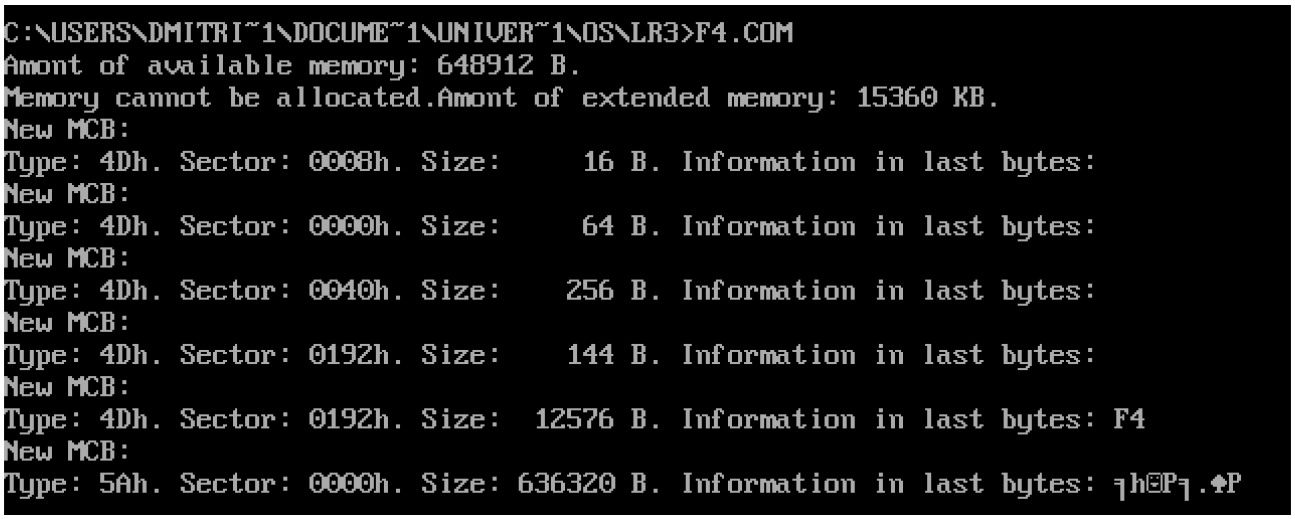

Рисунок 4 – Вывод программы f3.com

Программе не удалось запросить 64Кб до освобождения. Во второй строке было выведено сообщение об этом.

Контрольные вопросы

1. Что означает «доступный объём памяти»?

Максимальный объем памяти, который может использовать программа.

2. Где MCB блок Вашей программы в списке?

В первой и второй программе — это четвертые и пятые блоки (в обоих программах), в третьей — четвертый, пятый и шестой блок, в четвертой программе — четвертый и пятый блоки.
Эти блоки имеют одного владельца с сегментным адресом PSP 0192h и в последних 8 байтах пятых блоков (в случае с третьей программой — еще и в шестом блоке) содержится имя файла программы, которой принадлежит блок.

3. Какой размер памяти занимает программа в каждом случае?

- Первая программа занимает 649 056 байт.
- Вторая программа занимает 11 968 байт.
- Третья программа занимает 11 936 байт + дополнительно 65 536 байт, которые мы запросили.
- Четвертая программа занимает 12 720 байт.

Вывод

В результате выполнения данной лабораторной работы был изучен список блоков управления памятью, а также методы выделения и освобождения памяти для программы.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. F1.ASM

```
LAB3    SEGMENT
        ASSUME CS:LAB3, DS:LAB3, ES:NOTHING, SS:NOTHING
        ORG 100H
START:  JMP BEGIN

AvailableMemory db 'Amont of available memory:        B.', 0DH,0AH, '$'
ExtendedMemory db 'Amont of extended  memory:       KB.', 0DH,0AH, '$'
Mcbline db 'New MCB:', 0DH,0AH, 'Type: $'
Mcbsector db 'h. Sector: $'
Mcbsize db 'h. Size:        B$'
LastBytes db '. Information in last bytes: $'

BEGIN:
        mov ah, 4ah
        mov bx, 0ffffh
        int 21h
        mov ax, bx
        mov bx, 16
        mul bx
        lea si, AvailableMemory + 32
        call WRD_TO_DEC
        lea dx, AvailableMemory
        call WRITE
        xor ax, ax
        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 bh, al
        mov ax, bx
        lea si, ExtendedMemory + 30
        call WRD_TO_DEC
        lea dx, ExtendedMemory
        call WRITE
        xor ax, ax
        mov ah, 52h
        int 21h
        mov cx, es:[bx-2]
        mov es, cx
mcb:
        lea dx, Mcbline
        call WRITE
        mov al, es:[00h]
        call WRITE_BYTE
        lea dx, Mcbsector
        call WRITE
        mov ax, es:[01h]
        mov ch, ah
        mov ah, al
        mov al, ch
        call WRITE_BYTE
        mov ch, ah
        mov ah, al
        mov al, ch
        call WRITE_BYTE
        mov ax, es:[03h]
        mov bx, 10h
```

```
        mul bx
        mov si, offset Mcbsize
        add si, 14
        call WRD_TO_DEC
        mov dx, offset Mcbsize
        call WRITE
        lea dx, LastBytes
        call WRITE
        mov bx, 0
last:
        mov dl, es:[bx+08h]
        mov ah, 02h
        int 21h
        inc bx
        cmp bx, 8
        jl last
        lea dx, db 0DH,0AH, '$'
        call WRITE
        mov al, es:[00h]
        cmp al, 5Ah
        je endmcb
        xor cx, cx
        mov cx, es:[03h]
        mov bx, es
        add bx, cx
        inc bx
        mov es, bx
        jmp mcb

endmcb:
        xor al, al
        mov ah, 4Ch
```

```
        int 21h

WRITE_BYTE PROC near
        push ax
        push dx
        push cx
        call BYTE_TO_HEX
        xor cx, cx
        mov ch, ah
        mov dl, al
        mov ah, 02h
        int 21h
        mov dl, ch
        mov ah, 02h
        int 21h
        pop cx
        pop dx
        pop ax
        ret
WRITE_BYTE ENDP

WRITE PROC near
        mov ah, 09
        int 21h
WRITE 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_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

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

LAB3    ENDS
        END START
```

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. F2.ASM

```
LAB3    SEGMENT
        ASSUME CS:LAB3, DS:LAB3, ES:NOTHING, SS:NOTHING
        ORG 100H
START:  JMP BEGIN

AvailableMemory db 'Amont of available memory:        B.', 0DH,0AH, '$'
ExtendedMemory db 'Amont of extended  memory:       KB.', 0DH,0AH, '$'
Mcbline db 'New MCB:', 0DH,0AH, 'Type: $'
Mcbsector db 'h. Sector: $'
Mcbsize db 'h. Size:        B$'
LastBytes db '. Information in last bytes: $'

BEGIN:
        mov ah, 4ah
        mov bx, 0ffffh
        int 21h
        mov ax, bx
        mov bx, 16
        mul bx
        lea si, AvailableMemory + 32
        call WRD_TO_DEC
        lea dx, AvailableMemory
        call WRITE
        mov ah, 4ah
        mov bx, offset LAB_END
        int 21h
        xor ax, ax
        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 bh, al
        mov ax, bx
        lea si, ExtendedMemory + 30
        call WRD_TO_DEC
        lea dx, ExtendedMemory
        call WRITE
        xor ax, ax
        mov ah, 52h
        int 21h
        mov cx, es:[bx-2]
        mov es, cx
mcb:
        lea dx, Mcbline
        call WRITE
        mov al, es:[00h]
        call WRITE_BYTE
        lea dx, Mcbsector
        call WRITE
        mov ax, es:[01h]
        mov ch, ah
        mov ah, al
        mov al, ch
        call WRITE_BYTE
        mov ch, ah
        mov ah, al
        mov al, ch
```

```
        call WRITE_BYTE
        mov ax, es:[03h]
        mov bx, 10h
        mul bx
        mov si, offset Mcbsize
        add si, 14
        call WRD_TO_DEC
        mov dx, offset Mcbsize
        call WRITE
        lea dx, LastBytes
        call WRITE
        xor bx, bx
last:
        mov dl, es:[bx+08h]
        mov ah, 02h
        int 21h
        inc bx
        cmp bx, 8
        jl last
        lea dx, db 0DH,0AH, '$'
        call WRITE
        mov al, es:[00h]
        cmp al, 5Ah
        je endmcb
        xor cx, cx
        mov cx, es:[03h]
        mov bx, es
        add bx, cx
        inc bx
        mov es, bx
        jmp mcb
```

```
endmcb:
        xor al, al
        mov ah, 4Ch
        int 21h

WRITE_BYTE PROC near
        push ax
        push dx
        push cx
        call BYTE_TO_HEX
        xor cx, cx
        mov ch, ah
        mov dl, al
        mov ah, 02h
        int 21h
        mov dl, ch
        mov ah, 02h
        int 21h
        pop cx
        pop dx
        pop ax
        ret
WRITE_BYTE ENDP

WRITE PROC near
        mov ah, 09
        int 21h
WRITE 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_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
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

LAB_END:
LAB3    ENDS
        END START
```

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. F3.ASM

```
LAB3    SEGMENT
        ASSUME CS:LAB3, DS:LAB3, ES:NOTHING, SS:NOTHING
        ORG 100H
START:  JMP BEGIN

AvailableMemory db 'Amont of available memory:        B.', 0DH,0AH, '$'
ExtendedMemory db 'Amont of extended  memory:       KB.', 0DH,0AH, '$'
Mcbline db 'New MCB:', 0DH,0AH, 'Type: $'
Mcbsector db 'h. Sector: $'
Mcbsize db 'h. Size:        B$'
LastBytes db '. Information in last bytes: $'

BEGIN:
        mov ah, 4ah
        mov bx, 0ffffh
        int 21h
        mov ax, bx
        mov bx, 16
        mul bx
        lea si, AvailableMemory + 32
        call WRD_TO_DEC
        lea dx, AvailableMemory
        call WRITE
        mov bx, offset LAB_END
        mov ah, 4ah
        int 21h
        mov bx, 1000h
        mov ah, 48h
        int 21h
```

```
        xor ax, ax
        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 bh, al
        mov ax, bx
        lea si, ExtendedMemory + 30
        call WRD_TO_DEC
        lea dx, ExtendedMemory
        call WRITE
        xor ax, ax
        mov ah, 52h
        int 21h
        mov cx, es:[bx-2]
        mov es, cx
mcb:
        lea dx, Mcbline
        call WRITE
        mov al, es:[00h]
        call WRITE_BYTE
        lea dx, Mcbsector
        call WRITE
        mov ax, es:[01h]
        mov ch, ah
        mov ah, al
        mov al, ch
        call WRITE_BYTE
```

```
        mov ch, ah
        mov ah, al
        mov al, ch
        call WRITE_BYTE
        mov ax, es:[03h]
        mov bx, 10h
        mul bx
        mov si, offset Mcbsize
        add si, 14
        call WRD_TO_DEC
        mov dx, offset Mcbsize
        call WRITE
        lea dx, LastBytes
        call WRITE
        xor bx, bx
last:
        mov dl, es:[bx+08h]
        mov ah, 02h
        int 21h
        inc bx
        cmp bx, 8
        jl last
        lea dx, db 0DH,0AH, '$'
        call WRITE
        mov al, es:[00h]
        cmp al, 5Ah
        je endmcb
        xor cx, cx
        mov cx, es:[03h]
        mov bx, es
        add bx, cx
        inc bx
```

```
        mov es, bx
        jmp mcb

endmcb:
        xor al, al
        mov ah, 4Ch
        int 21h

WRITE_BYTE PROC near
        push ax
        push dx
        push cx
        call BYTE_TO_HEX
        xor cx, cx
        mov ch, ah
        mov dl, al
        mov ah, 02h
        int 21h
        mov dl, ch
        mov ah, 02h
        int 21h
        pop cx
        pop dx
        pop ax
        ret
WRITE_BYTE ENDP

WRITE PROC near
        mov ah, 09
        int 21h
WRITE 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_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

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

LAB_END:
LAB3    ENDS
        END START
```

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. F4.ASM

```
LAB3    SEGMENT
        ASSUME CS:LAB3, DS:LAB3, ES:NOTHING, SS:NOTHING
        ORG 100H
START:  JMP BEGIN

AvailableMemory db 'Amont of available memory:         B.', 0DH,0AH, '$'
ExtendedMemory db 'Amont of extended  memory:        KB.', 0DH,0AH, '$'
Mcbline db 'New MCB:', 0DH,0AH, 'Type: $'
Mcbsector db 'h. Sector: $'
Mcbsize db 'h. Size:         B$'
LastBytes db '. Information in last bytes: $'
MemError db 'Memory cannot be allocated.$'

BEGIN:
        mov ah, 4ah
        mov bx, 0ffffh
        int 21h
        mov ax, bx
        mov bx, 16
        mul bx
        lea si, AvailableMemory + 32
        call WRD_TO_DEC
        lea dx, AvailableMemory
        call WRITE
        mov ah, 48h
        mov bx, 1000h
        int 21h
        jc  allocError
        jmp continue
```

```
allocError:
        lea dx, MemError
        call WRITE

continue:
        mov ah, 4ah
        mov bx, offset LAB_END
        int 21h
        xor ax, ax
        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 bh, al
        mov ax, bx
        lea si, ExtendedMemory + 30
        call WRD_TO_DEC
        lea dx, ExtendedMemory
        call WRITE
        xor ax, ax
        mov ah, 52h
        int 21h
        mov cx, es:[bx-2]
        mov es, cx

mcb:
        lea dx, Mcbline
```

```
        call WRITE
        mov al, es:[00h]
        call WRITE_BYTE
        lea dx, Mcbsector
        call WRITE
        mov ax, es:[01h]
        mov ch, ah
        mov ah, al
        mov al, ch
        call WRITE_BYTE
        mov ch, ah
        mov ah, al
        mov al, ch
        call WRITE_BYTE
        mov ax, es:[03h]
        mov bx, 10h
        mul bx
        mov si, offset Mcbsize
        add si, 14
        call WRD_TO_DEC
        mov dx, offset Mcbsize
        call WRITE
        lea dx, LastBytes
        call WRITE
        xor bx, bx
last:
        mov dl, es:[bx+08h]
        mov ah, 02h
        int 21h
        inc bx
        cmp bx, 8
        jl last
```

```
        lea dx, db 0DH,0AH, '$'
        call WRITE
        mov al, es:[00h]
        cmp al, 5Ah
        je endmcb
        xor cx, cx
        mov cx, es:[03h]
        mov bx, es
        add bx, cx
        inc bx
        mov es, bx
        jmp mcb

endmcb:
        xor al, al
        mov ah, 4Ch
        int 21h

WRITE_BYTE PROC near
        push ax
        push dx
        push cx
        call BYTE_TO_HEX
        xor cx, cx
        mov ch, ah
        mov dl, al
        mov ah, 02h
        int 21h
        mov dl, ch
        mov ah, 02h
        int 21h
        pop cx
```

```
        pop dx
        pop ax
        ret
WRITE_BYTE    ENDP

WRITE PROC near
        mov ah, 09
        int 21h
WRITE 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_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

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

LAB_END:
LAB3    ENDS
        END START
```