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. 8304 _____ Сергеев А.Д.

Преподаватель _____ Губкин А.Ф.

Санкт-Петербург
2019

Цель работы.

Исследование структур данных и работы функций управления памятью операционной системы.

Постановка задачи:

Шаг 1.

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.

Шаг 2.

Изменить программу таким образом, чтобы она освобождала память, которую не занимает.

Шаг 3.

Изменить программу ещё раз таким образом, чтобы после освобождения памяти, программа запрашивала 64Кб памяти.

Шаг 2.

Изменить первоначальный вариант программы, запросив 64Кб памяти до освобождения памяти. Обработать завершение функций ядра, проверяя флаг CF.

Необходимые сведения для составления программы.

Учёт занятой и свободной памяти ведётся при помощи списка блоков управления памятью MCB (Memory Control Block). MCB занимает 16 байт (параграф) и располагается всегда с адреса кратного 16 (адреса 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).

Размер расширенной памяти находится в ячейках 30h, 31h CMOS. CMOS это энергонезависимая память, в которой хранится информация о конфигурации ПЭВМ.

Описание программы.

В результате выполнения лабораторной работы была написана программа, описание функций которой представлено ниже.

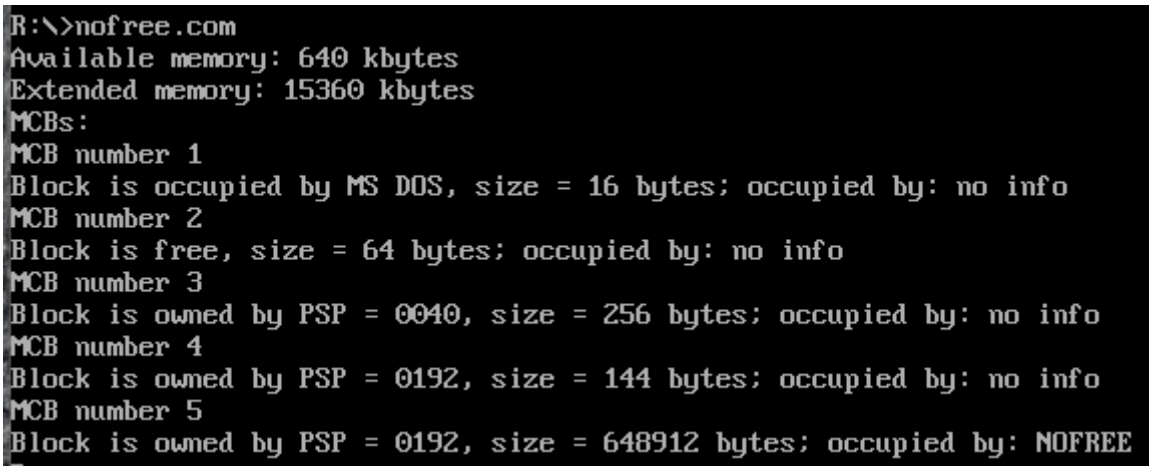
- DEC_WORD_PRINT - вывод содержимого AX в десятичной системе счисления;
- HEX_BYTE_PRINT - вспомогательная для HEX_WORD_PRINT процедура;
- HEX_WORD_PRINT - вывод содержимого AX в шестнадцатиричной системе счисления;
- PRINT_STRING - вывод строки из DX на экран;

- PRINT_AVL_MEMORY - вывод количества доступной памяти (в килобайтах);
- PRINT_EXT_MEMORY - вывод количества расширенной памяти (в килобайтах).
- PRINT_MCB - вывод списка MCB;
- FREE_MEM (для пунктов 2 - 4) - освобождение незанятой памяти;
- ALLOC_MEM (для пунктов 3 - 4) - выделение 64Кб памяти.

Ход работы

Написание исходного кода производилось в редакторе Atom на базе операционной системы Windows 10, сборка и отладка производились в эмуляторе DOSBox.

Модуль под названием NOFREE.COM выводит на экран размер доступной и расширенной памяти, а также данные MCB блоков.



Программа занимает всю доступную память.

Модуль под названием FREE.COM освобождает память, которая не занята программой.

```
R:\>free.com
Available memory: 640 kbytes
Extended memory: 15360 kbytes
MCBs:
MCB number 1
Block is occupied by MS DOS, size = 16 bytes; occupied by: no info
MCB number 2
Block is free, size = 64 bytes; occupied by: no info
MCB number 3
Block is owned by PSP = 0040, size = 256 bytes; occupied by: no info
MCB number 4
Block is owned by PSP = 0192, size = 144 bytes; occupied by: no info
MCB number 5
Block is owned by PSP = 0192, size = 1072 bytes; occupied by: FREE
MCB number 6
Block is free, size = 647824 bytes; occupied by: δ╥t♦♥4δ↑
```

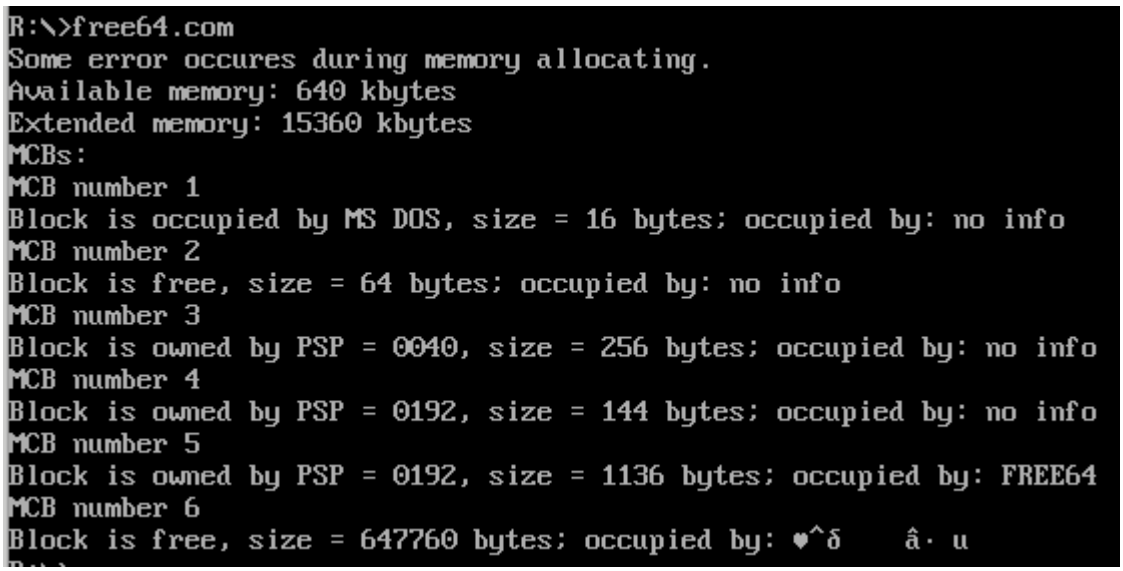
Теперь программа занимает не всю память, освобождённая память относится к новому, свободному блоку.

Модуль под названием NOFREE64.COM освобождает память, которая не занята программой, а потом запрашивает 64Кб памяти.

```
R:\>nofree64.com
Available memory: 640 kbytes
Extended memory: 15360 kbytes
MCBs:
MCB number 1
Block is occupied by MS DOS, size = 16 bytes; occupied by: no info
MCB number 2
Block is free, size = 64 bytes; occupied by: no info
MCB number 3
Block is owned by PSP = 0040, size = 256 bytes; occupied by: no info
MCB number 4
Block is owned by PSP = 0192, size = 144 bytes; occupied by: no info
MCB number 5
Block is owned by PSP = 0192, size = 1136 bytes; occupied by: NOFREE64
MCB number 6
Block is owned by PSP = 0192, size = 65536 bytes; occupied by: NOFREE64
MCB number 7
Block is free, size = 582208 bytes; occupied by: nteger d
```

Дополнительная память была выделена отдельным блоком, а блок, который появился после освобождения незанятой памяти, остался неизменным.

Модуль под названием FREE64.COM сначала запрашивает 64 Кб памяти, а потом освобождает незанятое.



Дополнительная память не была выделена, так как на момент попытки выделения вся доступная память уже принадлежит программе, о чём свидетельствует сообщение в начале вывода. Под программу выделено меньше 64Кб.

Вывод.

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

Контрольные вопросы.

Что означает «доступный объём» памяти:

Максимальный объём памяти, доступный для запуска и выполнения программ.

Где МСВ блок Вашей программы в списке:

Принадлежность блока памяти можно определить, по адресу его владельца (расположенному со смещением в один байт в МСВ). Также название модуля-владельца может содержаться в последних восьми байтах МСВ. Как видно из вывода программы, ей обычно принадлежит два блока, первый из которых имеет фиксированный размер в 144 байта, а второй зависит от размера исходного кода. Также программе будет принадлежать запрошенная и выделенная память.

Какой размер памяти программа занимает в каждом случае:

Без освобождения памяти программа занимает 144 байта и всю свободную память. После освобождения она занимает144 байта + около 1000 байтов (в каждом случае это зависит от исходного кода), для измерения размера вычисляется ближайший к концу модуля адрес конца параграфа и к нему добавляется ещё 16 байт. Также при запросе и успешном выделении дополнительной памяти, программа также занимает и её.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. NOFREE.ASM

```
LAB segment
     ASSUME cs: LAB, ds: LAB, es: NOTHING, ss: NOTHING
     org 100h

START: jmp BEGIN

AVL_MEMORY_INFO     db  "Available memory: $"
EXT_MEMORY_INFO     db  "Extended memory: $"
MCB_INFO            db  "MCBs:", 0Dh, 0Ah, "$"

MCB_NUM_INFO        db  "MCB number $"
AREA_SIZE_INFO           db  ", size = $"
OCCUPANT_INFO       db  "; occupied by: $"

END_LINE            db  0Dh, 0Ah, "$"
BYTES               db  " bytes$"
KBYTES              db  " kbytes", 0Dh, 0Ah, "$"

OWNER_INFO               db  0Dh, 0Ah, "Block is $"
OWNER_UNKNOWN       db  "owned by PSP = $"
OWNER_FREE               db  "free$"
OWNER_XMS                db  "occupied by OS XMS UMB$"
```

```
OWNER_TM           db  "occupied by driver's top memory$"
OWNER_DOS               db  "occupied by MS DOS$"
OWNER_386CB        db  "busy by 386MAX UMB$"
OWNER_386B              db  "blocked by 386MAX$"
OWNER_386               db  "occupied by 386MAX UMB$"

OCCUPANT_UNKNOWN   db  "no info$"

;-----------------------------------------------------------------
----------------

PRINT_STRING PROC NEAR ; (DX : String)
     push      ax

     mov  ah, 09h
     int       21h

     pop  ax
     ret
PRINT_STRING ENDP

DEC_WORD_PRINT PROC NEAR ; (AX : short)
     push      ax
     push      cx
```

```
        push dx
        push bx

        mov  bx, 10
        xor  cx, cx

NUM:
        div  bx
        push dx
        xor  dx, dx
        inc  cx
        cmp  ax, 0h
        jnz  NUM

PRINT_NUM:
        pop  dx
        or        dl, 30h
        mov  ah, 02h
        int  21h
        loop      PRINT_NUM

        pop  bx
        pop  dx
        pop  cx
```

```
     pop  ax
     ret
DEC_WORD_PRINT ENDP

HEX_BYTE_PRINT PROC NEAR ; (AL : byte)
     push      ax
     push      bx
     push      dx

     mov  ah, 0
     mov  bl, 10h
     div  bl
     mov  dx, ax
     mov  ah, 02h
     cmp  dl, 0Ah
     jl        PRINT
     add  dl, 07h

PRINT:
     add  dl, '0'
     int  21h;

     mov  dl, dh
     cmp  dl, 0Ah
```

```
     jl        PRINT_EXT
     add  dl, 07h

PRINT_EXT:
     add  dl, '0'
     int  21h;

     pop  dx
     pop  bx
     pop  ax
     ret
HEX_BYTE_PRINT ENDP

HEX_WORD_PRINT PROC NEAR ; (AX : short)
     push      ax
     push      ax
     mov  al, ah
     call      HEX_BYTE_PRINT
     pop  ax
     call      HEX_BYTE_PRINT
     pop  ax
     ret
HEX_WORD_PRINT ENDP
```

```
;-------------------------------------------------------------------
----------------


PRINT_AVL_MEMORY PROC NEAR
     push    ax
     push    bx
     push    dx

     mov     dx, offset AVL_MEMORY_INFO
     call    PRINT_STRING

     xor     ax, ax
     int  12h
     xor     dx, dx
     call    DEC_WORD_PRINT

    mov     dx, offset KBYTES
    call    PRINT_STRING

     pop     dx
     pop     bx
     pop     ax
     ret
PRINT_AVL_MEMORY ENDP
```

```
PRINT_EXT_MEMORY PROC NEAR
     push      ax
     push      bx
     push      dx

    mov   dx, offset EXT_MEMORY_INFO
     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
     xor     dx, dx
     call      DEC_WORD_PRINT

    mov     dx, offset KBYTES
    call    PRINT_STRING
```

```
     pop  dx
     pop  bx
     pop  ax
     ret
PRINT_EXT_MEMORY ENDP

PRINT_MCB PROC near
     push      ax
     push      bx
     push      cx
     push      dx
     push      es
     push      si

    mov     dx, offset MCB_INFO
     call    PRINT_STRING

     mov  ah, 52h
     int  21h
     mov  ax, es:[bx-2]
     mov  es, ax
     xor  cx, cx
    push    cx
```

```
NEXT_MCB:
    pop     cx
    mov   dx, offset MCB_NUM_INFO
    call  PRINT_STRING
    inc   cx
     mov  ax, cx
     xor     dx, dx
     call DEC_WORD_PRINT
    push  cx

OWNER_START:
     mov  dx, offset OWNER_INFO
     call      PRINT_STRING
     mov  ax, es:[1h]

     cmp  ax, 0h
     je        PRINT_FREE
     cmp  ax, 6h
     je        PRINT_XMS
     cmp  ax, 7h
     je        PRINT_TM
     cmp  ax, 8h
     je        PRINT_DOS
     cmp  ax, 0FFFAh
```

```
     je        PRINT_386CB
     cmp  ax, 0FFFDh
     je        PRINT_386B
     cmp  ax, 0FFFEh
     je        PRINT_386

    mov   dx, offset OWNER_UNKNOWN
    call  PRINT_STRING
     call      HEX_WORD_PRINT
     jmp  AREA_SIZE

PRINT_FREE:
     mov  dx, offset OWNER_FREE
    call  PRINT_STRING
     jmp  AREA_SIZE
PRINT_XMS:
     mov  dx, offset OWNER_XMS
    call  PRINT_STRING
     jmp  AREA_SIZE
PRINT_TM:
     mov  dx, offset OWNER_TM
    call  PRINT_STRING
     jmp  AREA_SIZE
PRINT_DOS:
```

```
     mov  dx, offset OWNER_DOS
    call  PRINT_STRING
     jmp  AREA_SIZE
PRINT_386CB:
     mov  dx, offset OWNER_386CB
    call  PRINT_STRING
     jmp  AREA_SIZE
PRINT_386B:
     mov  dx, offset OWNER_386B
    call  PRINT_STRING
     jmp  AREA_SIZE
PRINT_386:
     mov  dx, offset OWNER_386
     call      PRINT_STRING

AREA_SIZE:
     mov  dx, offset AREA_SIZE_INFO
     call      PRINT_STRING

     mov  ax, es:[3h]
     mov  bx, 10h
     mul  bx
     call      DEC_WORD_PRINT
```

```
    mov     dx, offset BYTES
    call    PRINT_STRING

    mov   dx, offset OCCUPANT_INFO
     call      PRINT_STRING

    mov     ax, es:[8h]
    cmp     ax, 0h
    je      NO_OCCUPANT

     mov  cx, 8
     xor  si, si
OCCUPANT:
     mov     dl, es:[si + 8h]
    mov     ah, 02h
     int     21h
     inc     si
     loop    OCCUPANT
    jmp     PROCEEDING

NO_OCCUPANT:
    mov   dx, offset OCCUPANT_UNKNOWN
    call  PRINT_STRING
```

```
PROCEEDING:
    xor   ax, ax
     mov  al, es:[0h]
    cmp     al, 5Ah
     je      ENDING

     mov  ax, es:[3h]
     mov     bx, es
     add     bx, ax
     inc     bx
     mov     es, bx

    mov     dx, offset END_LINE
     call    PRINT_STRING
     jmp     NEXT_MCB

ENDING:
     pop       cx
     pop     si
     pop     es
     pop     dx
     pop     cx
     pop     bx
     pop     ax
```

```
     ret
PRINT_MCB ENDP

BEGIN:
    call    PRINT_AVL_MEMORY
     call PRINT_EXT_MEMORY
    call    PRINT_MCB

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

PROGRAMM_END:

LAB ends
end START
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ. FREE.ASM

```
LAB segment
     ASSUME cs: LAB, ds: LAB, es: NOTHING, ss: NOTHING
     org 100h

START: jmp BEGIN

AVL_MEMORY_INFO     db  "Available memory: $"
EXT_MEMORY_INFO     db  "Extended memory: $"
MCB_INFO            db  "MCBs:", 0Dh, 0Ah, "$"

MCB_NUM_INFO        db  "MCB number $"
AREA_SIZE_INFO          db  ", size = $"
OCCUPANT_INFO       db  "; occupied by: $"

END_LINE            db  0Dh, 0Ah, "$"
BYTES               db  " bytes$"
KBYTES              db  " kbytes", 0Dh, 0Ah, "$"

OWNER_INFO              db  0Dh, 0Ah, "Block is $"
OWNER_UNKNOWN       db  "owned by PSP = $"
OWNER_FREE              db  "free$"
OWNER_XMS               db  "occupied by OS XMS UMB$"
```

```
    OWNER_TM            db  "occupied by driver's top memory$"
    OWNER_DOS               db  "occupied by MS DOS$"
    OWNER_386CB         db  "busy by 386MAX UMB$"
    OWNER_386B              db  "blocked by 386MAX$"
    OWNER_386               db  "occupied by 386MAX UMB$"

    OCCUPANT_UNKNOWN    db  "no info$"

    FREEING_ERROR       db  "Some error occures during memory
freeing.", 0Dh, 0Ah, "$"

    ;-------------------------------------------------------------
---------------------

    PRINT_STRING PROC NEAR ; (DX : String)
        push      ax

        mov  ah, 09h
        int       21h

        pop  ax
        ret
    PRINT_STRING ENDP

    DEC_WORD_PRINT PROC NEAR ; (AX : short)
```

```
        push      ax
        push      cx
        push dx
        push bx

        mov  bx, 10
        xor  cx, cx

NUM:
        div  bx
        push dx
        xor  dx, dx
        inc  cx
        cmp  ax, 0h
        jnz  NUM

PRINT_NUM:
        pop  dx
        or        dl, 30h
        mov  ah, 02h
        int  21h
        loop      PRINT_NUM

        pop  bx
```

```
     pop  dx
     pop  cx
     pop  ax
     ret
DEC_WORD_PRINT ENDP

HEX_BYTE_PRINT PROC NEAR ; (AL : byte)
     push      ax
     push      bx
     push      dx

     mov  ah, 0
     mov  bl, 10h
     div  bl
     mov  dx, ax
     mov  ah, 02h
     cmp  dl, 0Ah
     jl        PRINT
     add  dl, 07h

PRINT:
     add  dl, '0'
     int  21h;
```

```
    mov  dl, dh
    cmp  dl, 0Ah
    jl        PRINT_EXT
    add  dl, 07h

PRINT_EXT:
    add  dl, '0'
    int  21h;

    pop  dx
    pop  bx
    pop  ax
    ret
HEX_BYTE_PRINT ENDP

HEX_WORD_PRINT PROC NEAR ; (AX : short)
    push      ax
    push      ax
    mov  al, ah
    call      HEX_BYTE_PRINT
    pop  ax
    call      HEX_BYTE_PRINT
    pop  ax
    ret
```

```
     HEX_WORD_PRINT ENDP

     ;-------------------------------------------------------------
---------------------


     FREE_MEM PROC NEAR
          push      ax
          push      bx
          push      cx
          push      dx

          mov  bx, offset PROGRAMM_END
          mov  cl, 4
          shr  bx, cl
          add  bx, 1
          mov  ah, 4Ah
          int  21h
          jnc       FREE_SUCCESS

     FREE_ERROR:
          mov dx, offset FREEING_ERROR
          call PRINT_STRING

     FREE_SUCCESS:
```

```
     pop  dx
     pop       cx
     pop  bx
     pop  ax
     ret
FREE_MEM ENDP

PRINT_AVL_MEMORY PROC NEAR
     push    ax
     push    bx
     push    dx

     mov     dx, offset AVL_MEMORY_INFO
     call    PRINT_STRING

     xor     ax, ax
     int  12h
     xor     dx, dx
     call    DEC_WORD_PRINT

    mov     dx, offset KBYTES
    call    PRINT_STRING

     pop     dx
```

```
     pop     bx
     pop     ax
     ret
PRINT_AVL_MEMORY ENDP

PRINT_EXT_MEMORY PROC NEAR
     push      ax
     push      bx
     push      dx

    mov   dx, offset EXT_MEMORY_INFO
     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
     xor     dx, dx
     call      DEC_WORD_PRINT
```

```
    mov     dx, offset KBYTES
    call    PRINT_STRING

     pop  dx
     pop  bx
     pop  ax
     ret
PRINT_EXT_MEMORY ENDP

PRINT_MCB PROC near
     push      ax
     push      bx
     push      cx
     push      dx
     push      es
     push      si

    mov     dx, offset MCB_INFO
     call    PRINT_STRING

     mov  ah, 52h
     int  21h
     mov  ax, es:[bx-2]
```

```
     mov  es, ax
     xor  cx, cx
    push    cx

NEXT_MCB:
    pop     cx
    mov   dx, offset MCB_NUM_INFO
    call  PRINT_STRING
    inc   cx
     mov  ax, cx
     xor     dx, dx
     call DEC_WORD_PRINT
    push  cx

OWNER_START:
     mov  dx, offset OWNER_INFO
     call      PRINT_STRING
     mov  ax, es:[1h]

     cmp  ax, 0h
     je        PRINT_FREE
     cmp  ax, 6h
     je        PRINT_XMS
     cmp  ax, 7h
```

```
     je        PRINT_TM
     cmp  ax, 8h
     je        PRINT_DOS
     cmp  ax, 0FFFAh
     je        PRINT_386CB
     cmp  ax, 0FFFDh
     je        PRINT_386B
     cmp  ax, 0FFFEh
     je        PRINT_386

    mov   dx, offset OWNER_UNKNOWN
    call  PRINT_STRING
     call      HEX_WORD_PRINT
     jmp  AREA_SIZE

PRINT_FREE:
     mov  dx, offset OWNER_FREE
    call  PRINT_STRING
     jmp  AREA_SIZE
PRINT_XMS:
     mov  dx, offset OWNER_XMS
    call  PRINT_STRING
     jmp  AREA_SIZE
PRINT_TM:
```

```
     mov  dx, offset OWNER_TM
    call  PRINT_STRING
     jmp  AREA_SIZE
PRINT_DOS:
     mov  dx, offset OWNER_DOS
    call  PRINT_STRING
     jmp  AREA_SIZE
PRINT_386CB:
     mov  dx, offset OWNER_386CB
    call  PRINT_STRING
     jmp  AREA_SIZE
PRINT_386B:
     mov  dx, offset OWNER_386B
    call  PRINT_STRING
     jmp  AREA_SIZE
PRINT_386:
     mov  dx, offset OWNER_386
     call      PRINT_STRING

AREA_SIZE:
     mov  dx, offset AREA_SIZE_INFO
     call      PRINT_STRING

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

```
     mov  bx, 10h
     mul  bx
     call      DEC_WORD_PRINT

    mov     dx, offset BYTES
    call    PRINT_STRING

    mov   dx, offset OCCUPANT_INFO
     call      PRINT_STRING

    mov     ax, es:[8h]
    cmp     ax, 0h
    je      NO_OCCUPANT

     mov  cx, 8
     xor  si, si
OCCUPANT:
     mov     dl, es:[si + 8h]
    mov     ah, 02h
     int     21h
     inc     si
     loop    OCCUPANT
    jmp     PROCEEDING
```

```
NO_OCCUPANT:
    mov   dx, offset OCCUPANT_UNKNOWN
    call  PRINT_STRING

PROCEEDING:
    xor   ax, ax
     mov  al, es:[0h]
    cmp     al, 5Ah
     je      ENDING

     mov     ax, es:[3h]
     mov     bx, es
     add     bx, ax
     inc     bx
     mov     es, bx

    mov     dx, offset END_LINE
     call    PRINT_STRING
     jmp     NEXT_MCB

ENDING:
     pop  cx
     pop     si
     pop     es
```

```
     pop     dx
     pop     cx
     pop     bx
     pop     ax
     ret
PRINT_MCB ENDP

BEGIN:
     call      FREE_MEM

    call    PRINT_AVL_MEMORY
     call PRINT_EXT_MEMORY
    call    PRINT_MCB

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

PROGRAMM_END:

LAB ends
end START
```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД ПРОГРАММЫ. NOFREE64.ASM

```
LAB segment
     ASSUME cs: LAB, ds: LAB, es: NOTHING, ss: NOTHING
     org 100h

START: jmp BEGIN

AVL_MEMORY_INFO     db  "Available memory: $"
EXT_MEMORY_INFO     db  "Extended memory: $"
MCB_INFO            db  "MCBs:", 0Dh, 0Ah, "$"

MCB_NUM_INFO        db  "MCB number $"
AREA_SIZE_INFO           db  ", size = $"
OCCUPANT_INFO       db  "; occupied by: $"

END_LINE            db  0Dh, 0Ah, "$"
BYTES               db  " bytes$"
KBYTES              db  " kbytes", 0Dh, 0Ah, "$"

OWNER_INFO               db  0Dh, 0Ah, "Block is $"
OWNER_UNKNOWN       db  "owned by PSP = $"
OWNER_FREE               db  "free$"
OWNER_XMS                db  "occupied by OS XMS UMB$"
```

```
OWNER_TM            db  "occupied by driver's top memory$"
OWNER_DOS                db  "occupied by MS DOS$"
OWNER_386CB         db  "busy by 386MAX UMB$"
OWNER_386B               db  "blocked by 386MAX$"
OWNER_386                db  "occupied by 386MAX UMB$"

OCCUPANT_UNKNOWN    db  "no info$"

FREEING_ERROR       db  "Some error occures during memory
freeing.", 0Dh, 0Ah, "$"
ALLOCATING_ERROR    db  "Some error occures during memory
allocating.", 0Dh, 0Ah, "$"

;-----------------------------------------------------------------
----------------

PRINT_STRING PROC NEAR ; (DX : String)
     push      ax

     mov  ah, 09h
     int       21h

     pop  ax
     ret
PRINT_STRING ENDP
```

```
DEC_WORD_PRINT PROC NEAR ; (AX : short)
     push      ax
     push      cx
     push dx
     push bx

     mov  bx, 10
     xor  cx, cx

NUM:
     div  bx
     push dx
     xor  dx, dx
     inc  cx
     cmp  ax, 0h
     jnz  NUM

PRINT_NUM:
     pop  dx
     or        dl, 30h
     mov  ah, 02h
     int  21h
     loop      PRINT_NUM
```

```
     pop  bx
     pop  dx
     pop  cx
     pop  ax
     ret
DEC_WORD_PRINT ENDP

HEX_BYTE_PRINT PROC NEAR ; (AL : byte)
     push      ax
     push      bx
     push      dx

     mov  ah, 0
     mov  bl, 10h
     div  bl
     mov  dx, ax
     mov  ah, 02h
     cmp  dl, 0Ah
     jl        PRINT
     add  dl, 07h

PRINT:
     add  dl, '0'
```

```
     int  21h;


     mov  dl, dh
     cmp  dl, 0Ah
     jl        PRINT_EXT
     add  dl, 07h


PRINT_EXT:
     add  dl, '0'
     int  21h;


     pop  dx
     pop  bx
     pop  ax
     ret
HEX_BYTE_PRINT ENDP


HEX_WORD_PRINT PROC NEAR ; (AX : short)
     push      ax
     push      ax
     mov  al, ah
     call      HEX_BYTE_PRINT
     pop  ax
     call      HEX_BYTE_PRINT
```

```
     pop  ax
     ret
HEX_WORD_PRINT ENDP

;-----------------------------------------------------------------
---------------

FREE_MEM PROC NEAR
     push      ax
     push      bx
     push      cx
     push      dx

     mov  bx, offset PROGRAMM_END
     mov  cl, 4
     shr  bx, cl
     add  bx, 1
     mov  ah, 4Ah
     int  21h
     jnc       FREE_SUCCESS

FREE_ERROR:
     mov dx, offset FREEING_ERROR
     call PRINT_STRING
```

```
FREE_SUCCESS:
     pop  dx
     pop       cx
     pop  bx
     pop  ax
     ret
FREE_MEM ENDP


ALLOC_MEM PROC NEAR
    push  ax
    push  bx
    push  cx
    push  dx

    mov     bx, 1000h
    mov     ah, 48h
    int     21h
    jnc        FREE_SUCCESS

ALLOC_ERROR:
    mov    dx, offset ALLOCATING_ERROR
    call   PRINT_STRING
```

```
ALLOC_SUCCESS:
    pop   dx
    pop        cx
    pop   bx
    pop   ax
    ret
ALLOC_MEM ENDP


PRINT_AVL_MEMORY PROC NEAR
     push    ax
     push    bx
     push    dx

     mov     dx, offset AVL_MEMORY_INFO
     call    PRINT_STRING

     xor     ax, ax
     int  12h
     xor     dx, dx
     call    DEC_WORD_PRINT

    mov     dx, offset KBYTES
    call    PRINT_STRING
```

```
     pop     dx
     pop     bx
     pop     ax
     ret
PRINT_AVL_MEMORY ENDP

PRINT_EXT_MEMORY PROC NEAR
     push      ax
     push      bx
     push      dx

    mov   dx, offset EXT_MEMORY_INFO
     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
     xor     dx, dx
```

```
     call      DEC_WORD_PRINT

    mov     dx, offset KBYTES
    call    PRINT_STRING

     pop  dx
     pop  bx
     pop  ax
     ret
PRINT_EXT_MEMORY ENDP

PRINT_MCB PROC near
     push      ax
     push      bx
     push      cx
     push      dx
     push      es
     push      si

    mov     dx, offset MCB_INFO
     call    PRINT_STRING

     mov  ah, 52h
     int  21h
```

```
     mov  ax, es:[bx-2]
     mov  es, ax
     xor  cx, cx
    push    cx

NEXT_MCB:
    pop     cx
    mov   dx, offset MCB_NUM_INFO
    call  PRINT_STRING
    inc   cx
     mov  ax, cx
     xor     dx, dx
     call DEC_WORD_PRINT
    push  cx

OWNER_START:
     mov  dx, offset OWNER_INFO
     call      PRINT_STRING
     mov  ax, es:[1h]

     cmp  ax, 0h
     je        PRINT_FREE
     cmp  ax, 6h
     je        PRINT_XMS
```

```
     cmp  ax, 7h
     je        PRINT_TM
     cmp  ax, 8h
     je        PRINT_DOS
     cmp  ax, 0FFFAh
     je        PRINT_386CB
     cmp  ax, 0FFFDh
     je        PRINT_386B
     cmp  ax, 0FFFEh
     je        PRINT_386

    mov   dx, offset OWNER_UNKNOWN
    call  PRINT_STRING
     call      HEX_WORD_PRINT
     jmp  AREA_SIZE

PRINT_FREE:
     mov  dx, offset OWNER_FREE
    call  PRINT_STRING
     jmp  AREA_SIZE
PRINT_XMS:
     mov  dx, offset OWNER_XMS
    call  PRINT_STRING
     jmp  AREA_SIZE
```

```
PRINT_TM:
     mov  dx, offset OWNER_TM
    call  PRINT_STRING
     jmp  AREA_SIZE
PRINT_DOS:
     mov  dx, offset OWNER_DOS
    call  PRINT_STRING
     jmp  AREA_SIZE
PRINT_386CB:
     mov  dx, offset OWNER_386CB
    call  PRINT_STRING
     jmp  AREA_SIZE
PRINT_386B:
     mov  dx, offset OWNER_386B
    call  PRINT_STRING
     jmp  AREA_SIZE
PRINT_386:
     mov  dx, offset OWNER_386
     call      PRINT_STRING


AREA_SIZE:
     mov  dx, offset AREA_SIZE_INFO
     call      PRINT_STRING
```

```
     mov  ax, es:[3h]
     mov  bx, 10h
     mul  bx
     call      DEC_WORD_PRINT

    mov     dx, offset BYTES
    call    PRINT_STRING

    mov   dx, offset OCCUPANT_INFO
     call      PRINT_STRING

    mov     ax, es:[8h]
    cmp     ax, 0h
    je      NO_OCCUPANT

     mov  cx, 8
     xor  si, si
OCCUPANT:
     mov     dl, es:[si + 8h]
    mov     ah, 02h
     int     21h
     inc     si
     loop    OCCUPANT
    jmp     PROCEEDING
```

```
NO_OCCUPANT:
    mov   dx, offset OCCUPANT_UNKNOWN
    call  PRINT_STRING

PROCEEDING:
    xor   ax, ax
     mov  al, es:[0h]
    cmp     al, 5Ah
     je      ENDING

     mov     ax, es:[3h]
     mov     bx, es
     add     bx, ax
     inc     bx
     mov     es, bx

    mov     dx, offset END_LINE
     call    PRINT_STRING
     jmp     NEXT_MCB

ENDING:
     pop  cx
     pop     si
```

```
     pop     es
     pop     dx
     pop     cx
     pop     bx
     pop     ax
     ret
PRINT_MCB ENDP

BEGIN:
     call      FREE_MEM
    call    ALLOC_MEM

    call    PRINT_AVL_MEMORY
     call PRINT_EXT_MEMORY
    call    PRINT_MCB

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

PROGRAMM_END:

LAB ends
end START
```

ПРИЛОЖЕНИЕ Г

ИСХОДНЫЙ КОД ПРОГРАММЫ. FREE64.ASM

```
LAB segment
     ASSUME cs: LAB, ds: LAB, es: NOTHING, ss: NOTHING
     org 100h

START: jmp BEGIN

AVL_MEMORY_INFO     db  "Available memory: $"
EXT_MEMORY_INFO     db  "Extended memory: $"
MCB_INFO            db  "MCBs:", 0Dh, 0Ah, "$"

MCB_NUM_INFO        db  "MCB number $"
AREA_SIZE_INFO           db  ", size = $"
OCCUPANT_INFO       db  "; occupied by: $"

END_LINE            db  0Dh, 0Ah, "$"
BYTES               db  " bytes$"
KBYTES              db  " kbytes", 0Dh, 0Ah, "$"

OWNER_INFO               db  0Dh, 0Ah, "Block is $"
OWNER_UNKNOWN       db  "owned by PSP = $"
OWNER_FREE               db  "free$"
OWNER_XMS                db  "occupied by OS XMS UMB$"
```

```
OWNER_TM            db  "occupied by driver's top memory$"
OWNER_DOS                db  "occupied by MS DOS$"
OWNER_386CB         db  "busy by 386MAX UMB$"
OWNER_386B               db  "blocked by 386MAX$"
OWNER_386                db  "occupied by 386MAX UMB$"

OCCUPANT_UNKNOWN    db  "no info$"

FREEING_ERROR       db  "Some error occures during memory
freeing.", 0Dh, 0Ah, "$"
ALLOCATING_ERROR    db  "Some error occures during memory
allocating.", 0Dh, 0Ah, "$"

;-----------------------------------------------------------------
----------------

PRINT_STRING PROC NEAR ; (DX : String)
     push      ax

     mov  ah, 09h
     int       21h

     pop  ax
     ret
PRINT_STRING ENDP
```

```
DEC_WORD_PRINT PROC NEAR ; (AX : short)
     push      ax
     push      cx
     push dx
     push bx

     mov  bx, 10
     xor  cx, cx

NUM:
     div  bx
     push dx
     xor  dx, dx
     inc  cx
     cmp  ax, 0h
     jnz  NUM

PRINT_NUM:
     pop  dx
     or        dl, 30h
     mov  ah, 02h
     int  21h
     loop      PRINT_NUM
```

```
     pop  bx
     pop  dx
     pop  cx
     pop  ax
     ret
DEC_WORD_PRINT ENDP

HEX_BYTE_PRINT PROC NEAR ; (AL : byte)
     push      ax
     push      bx
     push      dx

     mov  ah, 0
     mov  bl, 10h
     div  bl
     mov  dx, ax
     mov  ah, 02h
     cmp  dl, 0Ah
     jl        PRINT
     add  dl, 07h

PRINT:
     add  dl, '0'
```

```
     int  21h;


     mov  dl, dh
     cmp  dl, 0Ah
     jl        PRINT_EXT
     add  dl, 07h


PRINT_EXT:
     add  dl, '0'
     int  21h;


     pop  dx
     pop  bx
     pop  ax
     ret
HEX_BYTE_PRINT ENDP


HEX_WORD_PRINT PROC NEAR ; (AX : short)
     push      ax
     push      ax
     mov  al, ah
     call      HEX_BYTE_PRINT
     pop  ax
     call      HEX_BYTE_PRINT
```

```
     pop  ax
     ret
HEX_WORD_PRINT ENDP

;-----------------------------------------------------------------
---------------

FREE_MEM PROC NEAR
     push      ax
     push      bx
     push      cx
     push      dx

     mov  bx, offset PROGRAMM_END
     mov  cl, 4
     shr  bx, cl
     add  bx, 1
     mov  ah, 4Ah
     int  21h
     jnc       FREE_SUCCESS

FREE_ERROR:
     mov dx, offset FREEING_ERROR
     call PRINT_STRING
```

```
FREE_SUCCESS:
     pop  dx
     pop       cx
     pop  bx
     pop  ax
     ret
FREE_MEM ENDP


ALLOC_MEM PROC NEAR
     push      ax
     push      bx
     push      cx
     push      dx


     mov     bx, 1000h
     mov     ah, 48h
     int     21h
     jnc       FREE_SUCCESS


ALLOC_ERROR:
     mov    dx, offset ALLOCATING_ERROR
     call   PRINT_STRING
```

```
ALLOC_SUCCESS:
     pop  dx
     pop       cx
     pop  bx
     pop  ax
     ret
ALLOC_MEM ENDP


PRINT_AVL_MEMORY PROC NEAR
     push    ax
     push    bx
     push    dx

     mov     dx, offset AVL_MEMORY_INFO
     call    PRINT_STRING

     xor     ax, ax
     int  12h
     xor     dx, dx
     call    DEC_WORD_PRINT

    mov     dx, offset KBYTES
    call    PRINT_STRING
```

```
     pop     dx
     pop     bx
     pop     ax
     ret
PRINT_AVL_MEMORY ENDP

PRINT_EXT_MEMORY PROC NEAR
     push      ax
     push      bx
     push      dx

    mov   dx, offset EXT_MEMORY_INFO
     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
     xor     dx, dx
```

```
     call      DEC_WORD_PRINT

    mov     dx, offset KBYTES
    call    PRINT_STRING

     pop  dx
     pop  bx
     pop  ax
     ret
PRINT_EXT_MEMORY ENDP

PRINT_MCB PROC near
     push      ax
     push      bx
     push      cx
     push      dx
     push      es
     push      si

    mov     dx, offset MCB_INFO
     call    PRINT_STRING

     mov  ah, 52h
     int  21h
```

```
     mov  ax, es:[bx-2]
     mov  es, ax
     xor  cx, cx
    push    cx

NEXT_MCB:
    pop     cx
    mov   dx, offset MCB_NUM_INFO
    call  PRINT_STRING
    inc   cx
     mov  ax, cx
     xor     dx, dx
     call DEC_WORD_PRINT
    push  cx

OWNER_START:
     mov  dx, offset OWNER_INFO
     call      PRINT_STRING
     mov  ax, es:[1h]

     cmp  ax, 0h
     je        PRINT_FREE
     cmp  ax, 6h
     je        PRINT_XMS
```

```
     cmp  ax, 7h
     je        PRINT_TM
     cmp  ax, 8h
     je        PRINT_DOS
     cmp  ax, 0FFFAh
     je        PRINT_386CB
     cmp  ax, 0FFFDh
     je        PRINT_386B
     cmp  ax, 0FFFEh
     je        PRINT_386

    mov   dx, offset OWNER_UNKNOWN
    call  PRINT_STRING
     call      HEX_WORD_PRINT
     jmp  AREA_SIZE

PRINT_FREE:
     mov  dx, offset OWNER_FREE
    call  PRINT_STRING
     jmp  AREA_SIZE
PRINT_XMS:
     mov  dx, offset OWNER_XMS
    call  PRINT_STRING
     jmp  AREA_SIZE
```

```
PRINT_TM:
     mov  dx, offset OWNER_TM
    call  PRINT_STRING
     jmp  AREA_SIZE
PRINT_DOS:
     mov  dx, offset OWNER_DOS
    call  PRINT_STRING
     jmp  AREA_SIZE
PRINT_386CB:
     mov  dx, offset OWNER_386CB
    call  PRINT_STRING
     jmp  AREA_SIZE
PRINT_386B:
     mov  dx, offset OWNER_386B
    call  PRINT_STRING
     jmp  AREA_SIZE
PRINT_386:
     mov  dx, offset OWNER_386
     call      PRINT_STRING


AREA_SIZE:
     mov  dx, offset AREA_SIZE_INFO
     call      PRINT_STRING
```

```
     mov  ax, es:[3h]
     mov  bx, 10h
     mul  bx
     call      DEC_WORD_PRINT

    mov     dx, offset BYTES
    call    PRINT_STRING

    mov   dx, offset OCCUPANT_INFO
     call      PRINT_STRING

    mov     ax, es:[8h]
    cmp     ax, 0h
    je      NO_OCCUPANT

     mov  cx, 8
     xor  si, si
OCCUPANT:
     mov     dl, es:[si + 8h]
    mov     ah, 02h
     int     21h
     inc     si
     loop    OCCUPANT
    jmp     PROCEEDING
```

```
NO_OCCUPANT:
    mov   dx, offset OCCUPANT_UNKNOWN
    call  PRINT_STRING

PROCEEDING:
    xor   ax, ax
     mov  al, es:[0h]
    cmp     al, 5Ah
     je      ENDING

     mov     ax, es:[3h]
     mov     bx, es
     add     bx, ax
     inc     bx
     mov     es, bx

    mov     dx, offset END_LINE
     call    PRINT_STRING
     jmp     NEXT_MCB

ENDING:
     pop  cx
     pop     si
```

```
     pop     es
     pop     dx
     pop     cx
     pop     bx
     pop     ax
     ret
PRINT_MCB ENDP

BEGIN:
    call    ALLOC_MEM
     call      FREE_MEM

    call    PRINT_AVL_MEMORY
     call PRINT_EXT_MEMORY
    call    PRINT_MCB

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

PROGRAMM_END:

LAB ends
end START
```