

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

Студентка гр. 8381

Преподаватель

Ивлева О.А.

Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Исследование структур данных и работы 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. Структура MCB:

Смещение	Длина поля (байт)	Содержимое поля
00h	1	тип MCB: 5Ah, если последний в списке, 4Dh, если не последний
01h	2	Сегментный адрес PSP владельца участка памяти, либо 0000h - свободный участок, 0006h - участок принадлежит драйверу OS XMS UMB 0007h - участок является исключенной верхней памятью драйверов 0008h - участок принадлежит MS DOS FFFAh - участок занят управляющим блоком 386MAX UMB FFFDh - участок заблокирован 386MAX FFFEh - участок принадлежит 386MAX UMB
03h	2	Размер участка в параграфах
05h	3	Зарезервирован
08h	8	"SC" - если участок принадлежит MS DOS, то в нем системный код "SD" - если участок принадлежит MS DOS, то в нем системные данные

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

Адрес первого MCB хранится во внутренней структуре MS DOS, называемой "List of Lists" (список списков). Доступ к указателю на эту структуру можно получить, используя функцию f52h "Get List of Lists" int 21h. В результате

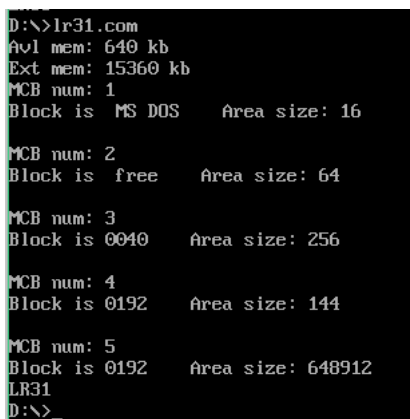
выполнения этой функции ES:BX будет указывать на список списков. Слово по адресу ES:[BX-2] и есть адрес самого первого MCB.

Размер расширенной памяти находится в ячейках 30h, 31h CMOS. CMOS это энергонезависимая память, в которой хранится информация о конфигурации ПЭВМ. Объем памяти составляет 64 байта.

Выполнение работы.

Сборка и отладка производились на базе эмулятора DOSBox 0.74-3.

Был написан текст исходного .COM модуля с именем lr31.com, который выводит блоки памяти. Наша программа (0192) до очистки памяти занимает 144+648912 байт.



```
D:\>lr31.com
Tot mem: 640 kb
Ext mem: 15360 kb
MCB num: 1
Block is MS DOS Area size: 16
MCB num: 2
Block is free Area size: 64
MCB num: 3
Block is 0040 Area size: 256
MCB num: 4
Block is 0192 Area size: 144
MCB num: 5
Block is 0192 Area size: 648912
LR31
D:\>
```

Рисунок 1 – lr31.com

Был написан текст исходного .COM модуля с именем lr32.com, который выводит блоки памяти. Наша программа (0192) с очисткой памяти занимает 144 + 1104 байт. Также присутствует 6-ой блок освобожденной памяти (free), который занимает 647792 байта.

```

D:\>lr32.com
Avl mem: 640 kb
Ext mem: 15360 kb
MCB num: 1
Block is MS DOS Area size: 16

MCB num: 2
Block is free Area size: 64

MCB num: 3
Block is 0040 Area size: 256

MCB num: 4
Block is 0192 Area size: 144

MCB num: 5
Block is 0192 Area size: 1104
LR32
MCB num: 6
Block is free Area size: 647792
D:\>

```

Рисунок 2 – lr32.com

Был написан текст исходного .COM модуля с именем lr32.com, который выводит блоки памяти. В нем программа сначала освобождает память, а потом запрашивает 64 кбайт. Наша программа (0192) занимает 144 + 1120 + 65536 байт. Также присутствует 7-ой блок освобожденной памяти (free), который занимает 582224 байта.

```

D:\>lr33.com
Avl mem: 640 kb
Ext mem: 15360 kb
MCB num: 1
Block is MS DOS Area size: 16

MCB num: 2
Block is free Area size: 64

MCB num: 3
Block is 0040 Area size: 256

MCB num: 4
Block is 0192 Area size: 144

MCB num: 5
Block is 0192 Area size: 1120
LR33
MCB num: 6
Block is 0192 Area size: 65536
LR33
MCB num: 7
Block is free Area size: 582224
D:\>_

```

Рисунок 3 –lr33.com

Был написан текст исходного .COM модуля с именем lr34.com, который выводит блоки памяти. В нем программа сначала запрашивает дополнительные 64кб, которые не могут выделиться, так как программа занимает всю свободную память, а затем происходит освобождение памяти (6 блок).

```

D:\>lr34.com
Avl mem: 640 kb
Ext mem: 15360 kb
MCB num: 1
Block is MS DOS      Area size: 16

MCB num: 2
Block is free        Area size: 64

MCB num: 3
Block is 0040        Area size: 256

MCB num: 4
Block is 0192        Area size: 144

MCB num: 5
Block is 0192        Area size: 1120
LR34
MCB num: 6
Block is free        Area size: 647776
LR33
D:\>_

```

Рисунок 4 –lr34.com

Контрольные вопросы

1. Что означает «доступный объем» памяти?

Максимальный объем памяти, который доступен программе.

2. Где МСВ блок вашей программы в списке?

(0192)

Lr31, lr32, lr34 – 4 и 5 блок.

Lr33 – 4, 5, 6 блоки

3. Какой размер памяти занимает программа в каждом случае?

Lr31: 144+648912

Lr32: 144 + 1104

Lr33: 144 + 1120 + 65536

Lr34: 144 + 1120

Выводы.

В ходе выполнения лабораторной работы были изучены методы выделения и освобождения памяти для программы.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. LR31.ASM

LAB segment

```
ASSUME CS: LAB, DS: LAB, ES: NOTHING, SS: NOTHING
org 100h
```

START: jmp BEGIN

```
STR_AVL_MEMORY      db  "Avl mem: $"
STR_EXT_MEMORY      db  "Ext mem: $"
STR_MCB_NUM         db  "MCB num: $"
STR_AREA_SIZE       db  "   Area size: $"
END_LINE            db  0Dh, 0Ah, "$"
KBYTES              db  " kb", 0Dh, 0Ah, "$"
STR_TM              db  " driver's top memory$"
STR_DOS             db  " MS DOS$"
STR_INFO            db  0Dh, 0Ah, "Block is $"
STR_FREE            db  " free$"
STR_XMS             db  " OS XMS UMB$"
STR_386CB           db  " busy by 386MAX UMB$"
STR_386B            db  " blocked by 386MAX$"
STR_386             db  " 386MAX UMB$"
```

```
WRITE      PROC  near
            push  AX
            mov   AH, 09h
            int   21h
            pop   AX
            ret
WRITE      ENDP
```

```
KBYTES_PRINT  PROC      near
            push  DX
            mov   DX, offset KBYTES
            call  WRITE
            pop   DX
            ret
KBYTES_PRINT  ENDP
```

```
DEC_WORD_PRINT  PROC
            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
        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
        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
    push    SI

    xor     AX, AX
    int     12h

    mov     DX, offset STR_AVL_MEMORY
    call    WRITE
    xor     DX, DX
    call    DEC_WORD_PRINT
    call    KBYTES_PRINT

    pop     SI
    pop     DX
    pop     BX
    pop     AX
    ret
PRINT_AVL_MEMORY ENDP

PRINT_EXT_MEMORY PROC NEAR
    push    AX
    push    BX
    push    DX
    push    SI

    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     DX, offset STR_EXT_MEMORY
    call    WRITE
    xor     DX, DX
    call    DEC_WORD_PRINT
    call    KBYTES_PRINT

    pop     SI
    pop     DX

```



```

        pop    BX
        pop    AX

    ret
PRINT_EXT_MEMORY ENDP

PRINT_MCB PROC
    push    AX
    push    BX
    push    CX
    push    DX
    push    ES
    push    SI

    mov     AH, 52h
    int     21h
    mov     AX, ES:[BX-2]
    mov     ES, AX
    xor     CX, CX
NEXT_MCB:
    inc     CX
    mov     DX, offset STR_MCB_NUM
    push    CX
    call    WRITE
    mov     AX, CX
    xor     DX, DX
    call    DEC_WORD_PRINT
OWNER_START:
    mov     DX, offset STR_INFO
    call    WRITE
    xor     AX, AX
    mov     AL, ES:[0h]
    push    AX
    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
    xor     DX, DX
    call    HEX_WORD_PRINT
    jmp     AREA_SIZE_START

PRINT_FREE:

```

```

        mov     DX, offset STR_FREE
        jmp     OWNER_END
PRINT_XMS:
        mov     DX, offset STR_XMS
        jmp     OWNER_END
PRINT_TM:
        mov     DX, offset STR_TM
        jmp     OWNER_END
PRINT_DOS:
        mov     DX, offset STR_DOS
        jmp     OWNER_END
PRINT_386CB:
        mov     DX, offset STR_386CB
        jmp     OWNER_END
PRINT_386B:
        mov     DX, offset STR_386B
        jmp     OWNER_END
PRINT_386:
        mov     DX, offset STR_386
OWNER_END:
        call    WRITE

AREA_SIZE_START:
        mov     DX, offset STR_AREA_SIZE
        call    WRITE
        mov     AX, ES:[3h]
        mov     BX, 10h
        mul     BX
        call    DEC_WORD_PRINT

        mov     CX, 8
        xor     SI, SI
        mov     DX, offset END_LINE
        call    WRITE

LAST_BYTES_START:
        mov     DL, ES:[SI + 8h]
        mov     AH, 02h
        int     21h
        inc     SI
        loop    LAST_BYTES_START

        mov     AX, ES:[3h]
        mov     BX, ES
        add     BX, AX
        inc     BX
        mov     ES, BX
        pop     AX
        pop     CX
        cmp     AL, 5Ah
        je      ENDING
        mov     DX, offset END_LINE
        call    WRITE
        jmp     NEXT_MCB

```

```

ENDING:
    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

PROG_END:

LAB    ends
end    START

```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ. LR32.ASM

LAB segment

ASSUME CS: LAB, DS: LAB, ES: NOTHING, SS: NOTHING
org 100h

START: jmp BEGIN

STR_AVL_MEMORY	db	"Avl mem: \$"
STR_EXT_MEMORY	db	"Ext mem: \$"
STR_MCB_NUM	db	"MCB num: \$"
STR_AREA_SIZE	db	" Area size: \$"
END_LINE	db	0Dh, 0Ah, "\$"
KBYTES	db	" kb", 0Dh, 0Ah, "\$"
STR_TM	db	" driver's top memory\$"
STR_DOS	db	" MS DOS\$"
STR_INFO	db	0Dh, 0Ah, "Block is \$"
STR_FREE	db	" free\$"
STR_XMS	db	" OS XMS UMB\$"
STR_386CB	db	" busy by 386MAX UMB\$"
STR_386B	db	" blocked by 386MAX\$"
STR_386	db	" 386MAX UMB\$"

WRITE	PROC	near	
	push	AX	
	mov	AH, 09h	
	int	21h	
	pop	AX	
	ret		

WRITE ENDP

KBYTES_PRINT	PROC	near	
	push	DX	
	mov	DX, offset KBYTES	
	call	WRITE	
	pop	DX	
	ret		

KBYTES_PRINT ENDP

DEC_WORD_PRINT	PROC ; IN: AX	
	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
        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
    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
    push    AX
    push    BX
    push    DX

    mov     BX, offset PROG_END
    add     BX, 100h
    shr     BX, 1
    shr     BX, 1
    shr     BX, 1
    shr     BX, 1 ; to paragraph
    mov     AH, 4Ah
    int     21h

    pop     DX
    pop     BX
    pop     AX

    ret
FREE_MEM    ENDP

PRINT_AVL_MEMORY    PROC NEAR
    push    AX
    push    BX
    push    DX
    push    SI

    xor     AX, AX

```

```

        int    12h

        mov     DX, offset STR_AVL_MEMORY
        call    WRITE
        xor     DX, DX
        call    DEC_WORD_PRINT
        call    KBYTES_PRINT

        pop     SI
        pop     DX
        pop     BX
        pop     AX
    ret
PRINT_AVL_MEMORY ENDP

PRINT_EXT_MEMORY PROC NEAR
    push  AX
    push  BX
    push  DX
    push  SI

    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     DX, offset STR_EXT_MEMORY
    call    WRITE
    xor     DX, DX
    call    DEC_WORD_PRINT
    call    KBYTES_PRINT

    pop     SI
    pop     DX
    pop     BX
    pop     AX

    ret
PRINT_EXT_MEMORY ENDP

PRINT_MCB      PROC

```

```

push  AX
push  BX
push  CX
push  DX
push  ES
push  SI

mov   AH, 52h
int   21h
mov   AX, ES:[BX-2]
mov   ES, AX
xor   CX, CX
NEXT_MCB:
inc   CX
mov   DX, offset STR_MCB_NUM
push  CX
call  WRITE
mov   AX, CX
xor   DX, DX
call  DEC_WORD_PRINT
OWNER_START:
mov   DX, offset STR_INFO
call  WRITE
xor   AX, AX
mov   AL, ES:[0h]
push  AX
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
xor   DX, DX
call  HEX_WORD_PRINT
jmp   AREA_SIZE_START

PRINT_FREE:

```



```

        mov     DX, offset STR_FREE
        jmp     OWNER_END
PRINT_XMS:
        mov     DX, offset STR_XMS
        jmp     OWNER_END
PRINT_TM:
        mov     DX, offset STR_TM
        jmp     OWNER_END
PRINT_DOS:
        mov     DX, offset STR_DOS
        jmp     OWNER_END
PRINT_386CB:
        mov     DX, offset STR_386CB
        jmp     OWNER_END
PRINT_386B:
        mov     DX, offset STR_386B
        jmp     OWNER_END
PRINT_386:
        mov     DX, offset STR_386
OWNER_END:
        call    WRITE

AREA_SIZE_START:
        mov     DX, offset STR_AREA_SIZE
        call    WRITE
        mov     AX, ES:[3h]
        mov     BX, 10h
        mul     BX
        call    DEC_WORD_PRINT

        mov     CX, 8
        xor     SI, SI
        mov     DX, offset END_LINE
        call    WRITE

LAST_BYTES_START:
        mov     DL, ES:[SI + 8h]
        mov     AH, 02h
        int     21h
        inc     SI
        loop    LAST_BYTES_START

        mov     AX, ES:[3h]
        mov     BX, ES
        add     BX, AX
        inc     BX
        mov     ES, BX

```

```

        pop    AX
        pop    CX
        cmp    AL, 5Ah
        je     ENDING
        mov    DX, offset END_LINE
        call   WRITE
        jmp    NEXT_MCB

ENDING:
        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

PROG_END:

LAB      ends
end      START

```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД ПРОГРАММЫ. LR33.ASM

LAB segment

```
ASSUME CS: LAB, DS: LAB, ES: NOTHING, SS: NOTHING
org 100h
```

START: jmp BEGIN

```
STR_AVL_MEMORY      db  "Avl mem: $"
STR_EXT_MEMORY      db  "Ext mem: $"
STR_MCB_NUM         db  "MCB num: $"
STR_AREA_SIZE       db  "   Area size: $"
END_LINE            db  0Dh, 0Ah, "$"
KBYTES              db  " kb", 0Dh, 0Ah, "$"
STR_TM              db  " driver's top memory$"
STR_DOS             db  " MS DOS$"
STR_INFO            db  0Dh, 0Ah, "Block is $"
STR_FREE            db  " free$"
STR_XMS             db  " OS XMS UMB$"
STR_386CB           db  " busy by 386MAX UMB$"
STR_386B            db  " blocked by 386MAX$"
STR_386             db  " 386MAX UMB$"
```

```
WRITE      PROC  near
            push  AX
            mov   AH, 09h
            int   21h
            pop   AX
            ret
WRITE      ENDP
```

```
KBYTES_PRINT  PROC      near
            push  DX
            mov   DX, offset KBYTES
            call  WRITE
            pop   DX
            ret
KBYTES_PRINT  ENDP
```

```
DEC_WORD_PRINT  PROC ; IN: AX
            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
        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
        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
    push AX
    push BX
    push DX

    mov BX, offset PROG_END
    add BX, 100h
    shr BX, 1
    shr BX, 1
    shr BX, 1
    shr BX, 1 ; to paragraph
    mov AH, 4Ah
    int 21h

    pop DX
    pop BX
    pop AX
    ret
FREE_MEM    ENDP

ADD_MEM    PROC
    push AX
    push BX
    push DX

    mov BX, 1000h
    mov AH, 48h
    int 21h

    pop DX
    pop BX
    pop AX
    ret
ADD_MEM    ENDP

PRINT_AVL_MEMORY    PROC NEAR
    push AX
    push BX
    push DX
    push SI

    xor AX, AX
    int 12h

    mov DX, offset STR_AVL_MEMORY
    call WRITE
    xor     DX, DX

```

```

        call    DEC_WORD_PRINT
    call    KBYTES_PRINT

        pop     SI
        pop     DX
        pop     BX
        pop     AX
    ret
PRINT_AVL_MEMORY ENDP

PRINT_EXT_MEMORY PROC NEAR
    push    AX
    push    BX
    push    DX
    push    SI

    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     DX, offset STR_EXT_MEMORY
    call    WRITE
    xor     DX, DX
    call    DEC_WORD_PRINT
    call    KBYTES_PRINT

    pop     SI
    pop     DX
    pop     BX
    pop     AX

    ret
PRINT_EXT_MEMORY ENDP

PRINT_MCB PROC
    push    AX
    push    BX
    push    CX
    push    DX
    push    ES
    push    SI

    mov     AH, 52h
    int     21h
    mov     AX, ES:[BX-2]
    mov     ES, AX
    xor     CX, CX

```

```

NEXT_MCB:
    inc     CX
    mov     DX, offset STR_MCB_NUM
    push    CX
    call    WRITE
    mov     AX, CX
    xor     DX, DX
    call    DEC_WORD_PRINT
OWNER_START:
    mov     DX, offset STR_INFO
    call    WRITE
    xor     AX, AX
    mov     AL, ES:[0h]
    push    AX
    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
    xor     DX, DX
    call    HEX_WORD_PRINT
    jmp     AREA_SIZE_START

PRINT_FREE:
    mov     DX, offset STR_FREE
    jmp     OWNER_END
PRINT_XMS:
    mov     DX, offset STR_XMS
    jmp     OWNER_END
PRINT_TM:
    mov     DX, offset STR_TM
    jmp     OWNER_END
PRINT_DOS:
    mov     DX, offset STR_DOS
    jmp     OWNER_END
PRINT_386CB:
    mov     DX, offset STR_386CB
    jmp     OWNER_END
PRINT_386B:
    mov     DX, offset STR_386B
    jmp     OWNER_END
PRINT_386:
    mov     DX, offset STR_386

```

```

OWNER_END:
    call WRITE

AREA_SIZE_START:
    mov DX, offset STR_AREA_SIZE
    call WRITE
    mov AX, ES:[3h]
    mov BX, 10h
    mul BX
    call DEC_WORD_PRINT

    mov CX, 8
    xor SI, SI
    mov DX, offset END_LINE
    call WRITE

LAST_BYTES_START:
    mov DL, ES:[SI + 8h]
    mov AH, 02h
    int 21h
    inc SI
    loop LAST_BYTES_START

    mov AX, ES:[3h]
    mov BX, ES
    add BX, AX
    inc BX
    mov ES, BX
    pop AX
    pop CX
    cmp AL, 5Ah
    je ENDING
    mov DX, offset END_LINE
    call WRITE
    jmp NEXT_MCB

ENDING:
    pop SI
    pop ES
    pop DX
    pop CX
    pop BX
    pop AX
    ret
PRINT_MCB      ENDP

BEGIN:
    call FREE_MEM
    call ADD_MEM

    call PRINT_AVL_MEMORY
    call PRINT_EXT_MEMORY
    call PRINT_MCB

```



```
xor    AL, AL
mov     AH, 4Ch
int     21h
```

PROG_END:

```
LAB     ends
end      START
```

ПРИЛОЖЕНИЕ Д

ИСХОДНЫЙ КОД ПРОГРАММЫ. LR34.ASM

LAB segment

ASSUME CS: LAB, DS: LAB, ES: NOTHING, SS: NOTHING
org 100h

START: jmp BEGIN

STR_AVL_MEMORY	db	"Avl mem: \$"
STR_EXT_MEMORY	db	"Ext mem: \$"
STR_MCB_NUM	db	"MCB num: \$"
STR_AREA_SIZE	db	" Area size: \$"
END_LINE	db	0Dh, 0Ah, "\$"
KBYTES	db	" kb", 0Dh, 0Ah, "\$"
STR_TM	db	" driver's top memory\$"
STR_DOS	db	" MS DOS\$"
STR_INFO	db	0Dh, 0Ah, "Block is \$"
STR_FREE	db	" free\$"
STR_XMS	db	" OS XMS UMB\$"
STR_386CB	db	" busy by 386MAX UMB\$"
STR_386B	db	" blocked by 386MAX\$"
STR_386	db	" 386MAX UMB\$"

WRITE	PROC	near	
	push	AX	
	mov	AH, 09h	
	int	21h	
	pop	AX	
	ret		

WRITE ENDP

KBYTES_PRINT	PROC	near	
	push	DX	
	mov	DX, offset KBYTES	
	call	WRITE	
	pop	DX	
	ret		

KBYTES_PRINT ENDP

DEC_WORD_PRINT	PROC ; IN: AX	
	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
        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
    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
    push    AX
    push    BX
    push    DX

    mov     BX, offset PROG_END
    add     BX, 100h
    shr     BX, 1
    shr     BX, 1
    shr     BX, 1
    shr     BX, 1 ; to paragraph
    mov     AH, 4Ah
    int     21h

    pop     DX
    pop     BX
    pop     AX
    ret
FREE_MEM    ENDP

ADD_MEM    PROC
    push    AX
    push    BX
    push    DX

    mov     BX, 1000h
    mov     AH, 48h

```

```

        int      21h

        pop      DX
        pop      BX
        pop      AX

    ret
ADD_MEM      ENDP

PRINT_AVL_MEMORY PROC NEAR
    push    AX
    push    BX
    push    DX
    push    SI

    xor     AX, AX
    int     12h

    mov     DX, offset STR_AVL_MEMORY
    call    WRITE
    xor     DX, DX
    call    DEC_WORD_PRINT
    call    KBYTES_PRINT

    pop     SI
    pop     DX
    pop     BX
    pop     AX

    ret
PRINT_AVL_MEMORY ENDP

PRINT_EXT_MEMORY PROC NEAR
    push    AX
    push    BX
    push    DX
    push    SI

    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     DX, offset STR_EXT_MEMORY
    call    WRITE
    xor     DX, DX
    call    DEC_WORD_PRINT
    call    KBYTES_PRINT

    pop     SI
    pop     DX
    pop     BX
    pop     AX

    ret
PRINT_EXT_MEMORY ENDP

PRINT_MCB      PROC
    push    AX
    push    BX
    push    CX
    push    DX
    push    ES
    push    SI

    mov     AH, 52h
    int     21h
    mov     AX, ES:[BX-2]
    mov     ES, AX
    xor     CX, CX
NEXT_MCB:
    inc     CX
    mov     DX, offset STR_MCB_NUM
    push    CX
    call    WRITE
    mov     AX, CX
    xor     DX, DX
    call    DEC_WORD_PRINT
OWNER_START:
    mov     DX, offset STR_INFO
    call    WRITE
    xor     AX, AX
    mov     AL, ES:[0h]
    push    AX
    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
        xor     DX, DX
        call    HEX_WORD_PRINT
        jmp     AREA_SIZE_START

PRINT_FREE:
        mov     DX, offset STR_FREE
        jmp     OWNER_END

PRINT_XMS:
        mov     DX, offset STR_XMS
        jmp     OWNER_END

PRINT_TM:
        mov     DX, offset STR_TM
        jmp     OWNER_END

PRINT_DOS:
        mov     DX, offset STR_DOS
        jmp     OWNER_END

PRINT_386CB:
        mov     DX, offset STR_386CB
        jmp     OWNER_END

PRINT_386B:
        mov     DX, offset STR_386B
        jmp     OWNER_END

PRINT_386:
        mov     DX, offset STR_386

OWNER_END:
        call    WRITE

AREA_SIZE_START:
        mov     DX, offset STR_AREA_SIZE
        call    WRITE
        mov     AX, ES:[3h]
        mov     BX, 10h
        mul     BX
        call    DEC_WORD_PRINT

        mov     CX, 8
        xor     SI, SI

```

```

        mov     DX, offset END_LINE
        call    WRITE

LAST_BYTES_START:
        mov     DL, ES:[SI + 8h]
        mov     AH, 02h
        int     21h
        inc     SI
        loop    LAST_BYTES_START

        mov     AX, ES:[3h]
        mov     BX, ES
        add     BX, AX
        inc     BX
        mov     ES, BX
        pop     AX
        pop     CX
        cmp     AL, 5Ah
        je      ENDING
        mov     DX, offset END_LINE
        call    WRITE
        jmp     NEXT_MCB

ENDING:
        pop     SI
        pop     ES
        pop     DX
        pop     CX
        pop     BX
        pop     AX
        ret
PRINT_MCB      ENDP

BEGIN:
        call    ADD_MEM
        call    FREE_MEM

        call    PRINT_AVL_MEMORY
        call    PRINT_EXT_MEMORY
        call    PRINT_MCB

        xor     AL, AL
        mov     AH, 4Ch
        int     21h

```


PROG_END:

LAB	ends
end	START