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. 8381 _____ Облизанов А.Д.

Преподаватель _____ 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 представлена на рис. 1.

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 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.

Написание работы производилось на базе операционной системы Windows 10 в редакторе Visual Code. Сборка, отладка производились на базе эмулятора DOSBox 0.74-3.

Был написан текст исходного .COM модуля с именем lr31.com, который обеспечивает вывод на экран

- Размера доступной памяти (в килобайтах)
- Размера расширенной памяти (в килобайтах)
- Данных с MCB блоков:
    - «Владелец» участка (или адрес PSP владельца участка)
    - Размер участка памяти
    - SD/SC/Другое (последние 8 байт MCB)

Вывод программы представлен на рис. 2.

```
D:\>lr31.com
Available memory: 640 kbytes
Extended memory: 15360 kbytes
MCB number: 1
Block is  MS DOS    Area size: 16

MCB number: 2
Block is  free    Area size: 64

MCB number: 3
Block is 0040    Area size: 256

MCB number: 4
Block is 0192    Area size: 144

MCB number: 5
Block is 0192    Area size: 648912
LR31
```

Рисунок 2 – Выполнение lr31.com

Как видно из рисунка, программа занимает всю доступную память.

Далее был написан текст исходного модуля lr32.com, в котором программа освобождает память, которую не занимает. Результат выполнения программы представлен на рис. 3.

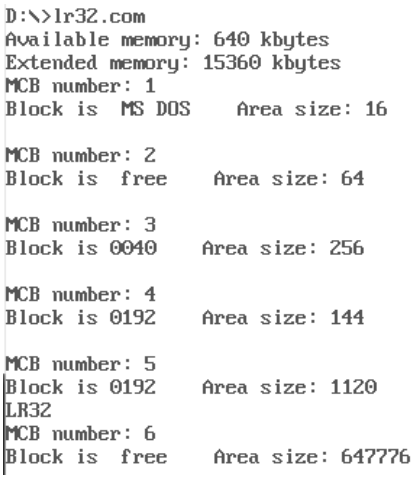

Рисунок 3 – Выполнение lr32.com

Как видно из рисунка, программа не занимает всю память. Освобожденная память относится к шестому блоку управления памятью, который является свободным.

Далее был написан текст исходного модуля lr33.com, в котором программа освобождает память, которую не занимает, а после этого запрашивает дополнительно 64 кбайт. Выполнение показано на рис. 4.

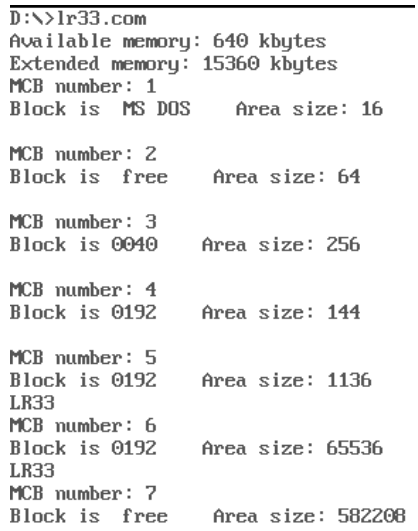

Рисунок 4 – Выполнение lr33.com

Как видно из рисунка, дополнительно выделенная память для программы относится к 6-му блоку, а к 5-му блоку относится память, выделенная программе после освобождения памяти. Сегментный адрес PSP владельца участка памяти 5-го и 6-го блока совпадают.

Далее был написан текст исходного модуля lr34.com, в котором программа сначала запрашивает дополнительно 64 кбайт, а затем освобождает память. Выполнение показано на рис. 5.

```
D:\>lr34.com
Available memory: 640 kbytes
Extended memory: 15360 kbytes
MCB number: 1
Block is  MS DOS    Area size: 16

MCB number: 2
Block is  free    Area size: 64

MCB number: 3
Block is 0040    Area size: 256

MCB number: 4
Block is 0192    Area size: 144

MCB number: 5
Block is 0192    Area size: 1136
LR34
MCB number: 6
Block is  free    Area size: 647760
```

Рисунок 5 – Выполнение lr34.com

Как видно из рисунка, дополнительная память не была выделена для программы. Выделенная под программу память относится к 5-му блоку и имеет объем меньше 64 кбайт.

Контрольные вопросы

1. Что означает «доступный объем» памяти?

Максимальный объем памяти, который может быть доступен для запуска и выполнения программ.

Примечание: в модуле lr34.com программа изначально занимает весь доступный объем памяти, из-за чего невозможно выделить дополнительно 64 кбайт.

2. **Где MCB блок вашей программы в списке?**
    - Lr31.com, lr32.com, lr34.com – MCB блок программы пятый
    - Lr33.com – MCB блок программы пятый, но также шестой для дополнительно выделенных во время работы программы 64 кбайт.
3. **Какой размер памяти занимает программа в каждом случае?**
    - Lr31.com. Программа занимает 648 912 байт, то есть всю доступную ей память.
    - Lr32.com. Программа занимает 1120 байт после освобождения памяти. Для освобождения в конец программы была добавлена метка, по смещению которой можно определить объем памяти, необходимый для программы, добавляя некоторой запас.
    - Lr33.com. Программа занимает 1120 байт после освобождения памяти (пятый блок), а также занимает дополнительно 65536 байт после выделения (шестой блок).
    - Lr34.com. Программа занимает 1120 байт после неудачной попытки выделения дополнительной памяти и (удачного) освобождения. Причина того, что память не была выделена, в 1-м контрольном вопросе.

Выводы.

В ходе выполнения лабораторной работы был изучен список блоков управления памятью, а также 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

    ;DATA

AVL_MEMORY_INFO     db  "Available memory: $"
EXT_MEMORY_INFO     db  "Extended memory: $"

MCB_NUM_INFO            db  "MCB number: $"
AREA_SIZE_INFO          db  "    Area size: $"

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

OWNER_INFO              db  0Dh, 0Ah, "Block is $"
OWNER_FREE              db  " free$"
OWNER_XMS               db  " OS XMS UMB$"
OWNER_TM                db  " driver's top memory$"
OWNER_DOS               db  " MS DOS$"
OWNER_386CB             db  " busy by 386MAX UMB$"
OWNER_386B              db  " blocked by 386MAX$"
OWNER_386               db  " 386MAX UMB$"


    ; There is some CUSTOM procedures...

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

KBYTES_PRINT    PROC    near
        push    DX
        mov     DX, offset KBYTES
        call    PRINT_STRING
        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

    ; here is the action...

PRINT_AVL_MEMORY  PROC NEAR
          push    AX
          push    BX
          push    DX
          push    SI

          xor     AX, AX
            int   12h

          mov     DX, offset AVL_MEMORY_INFO
          call    PRINT_STRING
            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 EXT_MEMORY_INFO
            call  PRINT_STRING
            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 MCB_NUM_INFO
      push  CX
      call  PRINT_STRING
      mov   AX, CX
      xor   DX, DX
      call  DEC_WORD_PRINT
OWNER_START:
      mov   DX, offset OWNER_INFO
      call  PRINT_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_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 OWNER_FREE
      jmp   OWNER_END
PRINT_XMS:
      mov   DX, offset OWNER_XMS
      jmp   OWNER_END
PRINT_TM:
      mov   DX, offset OWNER_TM
      jmp   OWNER_END
PRINT_DOS:
      mov   DX, offset OWNER_DOS
```

```
      jmp   OWNER_END
PRINT_386CB:
      mov   DX, offset OWNER_386CB
      jmp   OWNER_END
PRINT_386B:
      mov   DX, offset OWNER_386B
      jmp   OWNER_END
PRINT_386:
      mov   DX, offset OWNER_386
OWNER_END:
      call  PRINT_STRING

AREA_SIZE_START:
      mov   DX, offset AREA_SIZE_INFO
      call  PRINT_STRING
      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  PRINT_STRING

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    PRINT_STRING
      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

    ;DATA

AVL_MEMORY_INFO     db  "Available memory: $"
EXT_MEMORY_INFO     db  "Extended memory: $"

MCB_NUM_INFO            db  "MCB number: $"
AREA_SIZE_INFO          db  "    Area size: $"

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

OWNER_INFO              db  0Dh, 0Ah, "Block is $"
OWNER_FREE              db  " free$"
OWNER_XMS               db  " OS XMS UMB$"
OWNER_TM                db  " driver's top memory$"
OWNER_DOS               db  " MS DOS$"
OWNER_386CB             db  " busy by 386MAX UMB$"
OWNER_386B              db  " blocked by 386MAX$"
OWNER_386               db  " 386MAX UMB$"


    ; There is some CUSTOM procedures...

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

KBYTES_PRINT    PROC    near
        push    DX
        mov     DX, offset KBYTES
        call    PRINT_STRING
        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
```

```
    ; here is the action...

PRINT_AVL_MEMORY  PROC NEAR
          push    AX
          push    BX
          push    DX
          push    SI

          xor     AX, AX
            int   12h

          mov     DX, offset AVL_MEMORY_INFO
          call    PRINT_STRING
            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 EXT_MEMORY_INFO
            call  PRINT_STRING
            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 MCB_NUM_INFO
            push  CX
            call  PRINT_STRING
            mov   AX, CX
            xor   DX, DX
            call  DEC_WORD_PRINT
      OWNER_START:
            mov   DX, offset OWNER_INFO
            call  PRINT_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_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 OWNER_FREE
      jmp   OWNER_END
PRINT_XMS:
      mov   DX, offset OWNER_XMS
      jmp   OWNER_END
PRINT_TM:
      mov   DX, offset OWNER_TM
      jmp   OWNER_END
PRINT_DOS:
      mov   DX, offset OWNER_DOS
      jmp   OWNER_END
PRINT_386CB:
      mov   DX, offset OWNER_386CB
      jmp   OWNER_END
PRINT_386B:
      mov   DX, offset OWNER_386B
      jmp   OWNER_END
PRINT_386:
      mov   DX, offset OWNER_386
OWNER_END:
      call  PRINT_STRING

AREA_SIZE_START:
      mov   DX, offset AREA_SIZE_INFO
      call  PRINT_STRING
      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  PRINT_STRING

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    PRINT_STRING
            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

    ;DATA

AVL_MEMORY_INFO     db  "Available memory: $"
EXT_MEMORY_INFO     db  "Extended memory: $"

MCB_NUM_INFO            db  "MCB number: $"
AREA_SIZE_INFO          db  "    Area size: $"

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

OWNER_INFO              db  0Dh, 0Ah, "Block is $"
OWNER_FREE              db  " free$"
OWNER_XMS               db  " OS XMS UMB$"
OWNER_TM                db  " driver's top memory$"
OWNER_DOS               db  " MS DOS$"
OWNER_386CB             db  " busy by 386MAX UMB$"
OWNER_386B              db  " blocked by 386MAX$"
OWNER_386               db  " 386MAX UMB$"


    ; There is some CUSTOM procedures...

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

KBYTES_PRINT    PROC    near
        push    DX
        mov     DX, offset KBYTES
        call    PRINT_STRING
        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

    ; here is the action...

PRINT_AVL_MEMORY  PROC NEAR
          push    AX
          push    BX
          push    DX
          push    SI

          xor     AX, AX
            int   12h

          mov     DX, offset AVL_MEMORY_INFO
          call    PRINT_STRING
            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 EXT_MEMORY_INFO
            call  PRINT_STRING
            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 MCB_NUM_INFO
            push  CX
            call  PRINT_STRING
            mov   AX, CX
            xor   DX, DX
            call  DEC_WORD_PRINT
      OWNER_START:
            mov   DX, offset OWNER_INFO
```

```
      call  PRINT_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_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 OWNER_FREE
      jmp   OWNER_END
PRINT_XMS:
      mov   DX, offset OWNER_XMS
      jmp   OWNER_END
PRINT_TM:
      mov   DX, offset OWNER_TM
      jmp   OWNER_END
PRINT_DOS:
      mov   DX, offset OWNER_DOS
      jmp   OWNER_END
PRINT_386CB:
      mov   DX, offset OWNER_386CB
      jmp   OWNER_END
PRINT_386B:
      mov   DX, offset OWNER_386B
      jmp   OWNER_END
PRINT_386:
      mov   DX, offset OWNER_386
OWNER_END:
      call  PRINT_STRING
```

```
      AREA_SIZE_START:
            mov   DX, offset AREA_SIZE_INFO
            call  PRINT_STRING
            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  PRINT_STRING

      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    PRINT_STRING
            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

    ;DATA

AVL_MEMORY_INFO     db  "Available memory: $"
EXT_MEMORY_INFO     db  "Extended memory: $"

MCB_NUM_INFO            db  "MCB number: $"
AREA_SIZE_INFO          db  "    Area size: $"

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

OWNER_INFO              db  0Dh, 0Ah, "Block is $"
OWNER_FREE              db  " free$"
OWNER_XMS               db  " OS XMS UMB$"
OWNER_TM                db  " driver's top memory$"
OWNER_DOS               db  " MS DOS$"
OWNER_386CB             db  " busy by 386MAX UMB$"
OWNER_386B              db  " blocked by 386MAX$"
OWNER_386               db  " 386MAX UMB$"


    ; There is some CUSTOM procedures...

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

KBYTES_PRINT    PROC    near
        push    DX
        mov     DX, offset KBYTES
        call    PRINT_STRING
        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

    ; here is the action...

PRINT_AVL_MEMORY  PROC NEAR
          push    AX
          push    BX
          push    DX
          push    SI

          xor     AX, AX
            int   12h

          mov     DX, offset AVL_MEMORY_INFO
          call    PRINT_STRING
            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 EXT_MEMORY_INFO
            call  PRINT_STRING
            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 MCB_NUM_INFO
            push  CX
            call  PRINT_STRING
            mov   AX, CX
            xor   DX, DX
            call  DEC_WORD_PRINT
      OWNER_START:
            mov   DX, offset OWNER_INFO
```

```
      call  PRINT_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_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 OWNER_FREE
      jmp   OWNER_END
PRINT_XMS:
      mov   DX, offset OWNER_XMS
      jmp   OWNER_END
PRINT_TM:
      mov   DX, offset OWNER_TM
      jmp   OWNER_END
PRINT_DOS:
      mov   DX, offset OWNER_DOS
      jmp   OWNER_END
PRINT_386CB:
      mov   DX, offset OWNER_386CB
      jmp   OWNER_END
PRINT_386B:
      mov   DX, offset OWNER_386B
      jmp   OWNER_END
PRINT_386:
      mov   DX, offset OWNER_386
OWNER_END:
      call  PRINT_STRING
```

```
      AREA_SIZE_START:
            mov   DX, offset AREA_SIZE_INFO
            call  PRINT_STRING
            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  PRINT_STRING

      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    PRINT_STRING
            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
```