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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Операционные системы»

Тема: Исследование организации управления основной памятью

Студент гр. 8383 _____ Костарев К.В.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

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

Основные теоретические сведения.

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

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

В результате выполнения функции 52h прерывания int 21h ES:[BX] будет указывать на список списков. Слово по адресу ES:[BX-2] и есть адрес самого первого MCB.

Размер расширенной памяти находится в ячейках 30h, 31h CMOS.

Выполнение работы.

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

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

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

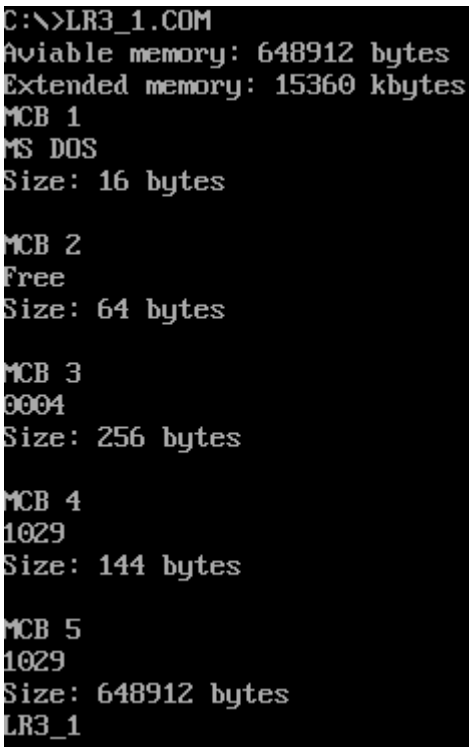

Рисунок 1 – Результат работы программы

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

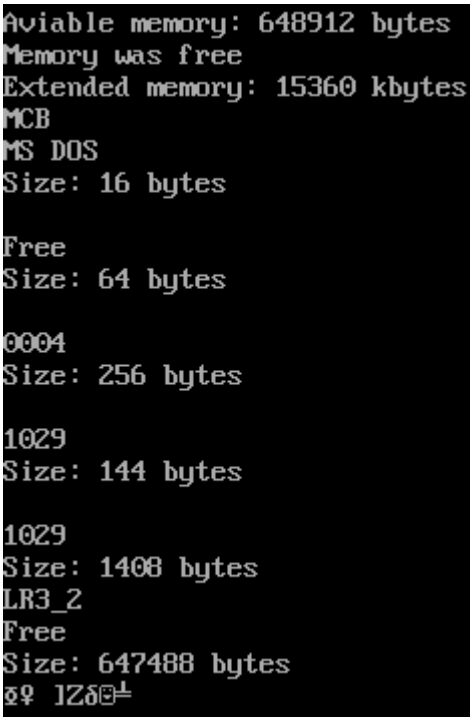

Рисунок 2 – Результат работы 2 версии программы

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

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

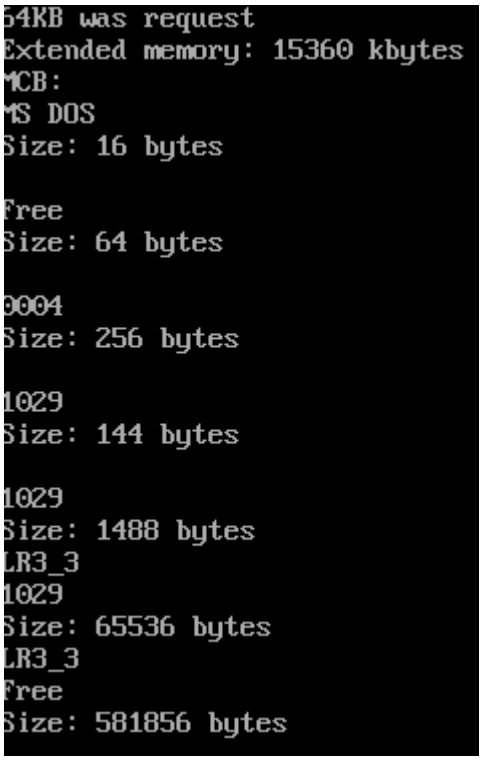

Рисунок 3 – Результат работы 3 версии программы

Можно отметить, что был создан еще один блок управления памятью, и теперь в предпоследнем находится выделенная память, а в последнем освобожденная.

Программа была изменена в последний раз, теперь 64 кБ запрашивается до освобождения памяти. Результат работы приведен на рис. 4, исходный код в Приложении Г.

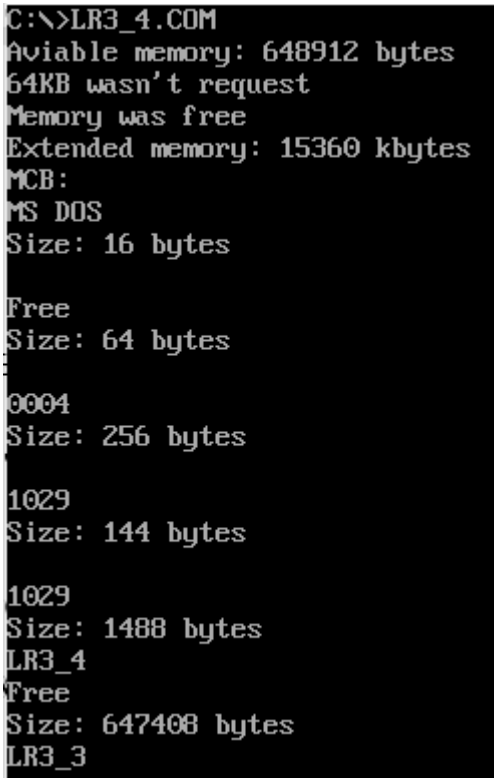

Рисунок 4 – Результат работы 4 версии программы

Память не была выделена.

1) *Что означает «доступный объем» памяти?*

   Объем памяти в системе, доступный для запуска и выполнения программ.

2) *Где MCB блок вашей программы в списке?*

   В первой, второй и четвертой версиях программы это четвертый и пятый блок. Т.к. в третьей версии запрашивается 64 кБ памяти, то в этой версии еще и шестой.

3) *Какой размер памяти занимает программа в каждом случае?*

   В первой версии вся доступная память (648 912+144 = 649 556 байт). Во второй и четвертой версиях 1408+144 = 1552 и 1488+144 = 1632 байт соответственно, т.к. была освобождена память. В третьей версии 1488 + 144 + 65536 байт, т.к. было запрошено 64 кБ доступной памяти.

Выводы.

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

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПЕРВОЙ ВЕРСИИ ПРОГРАММЫ

```
LR3 SEGMENT
ASSUME CS:LR3, DS:LR3, ES:NOTHING, SS:NOTHING
ORG 100H
START: JMP BEGIN
ENTER db 13, 10, '$'
AVIABLE db "Aviable memory: $"
BYTES db " bytes", 13, 10, '$'
EXTENDED db "Extended memory: $"
KBYTES db " kbytes", 13, 10, '$'
MCB db "MCB $"
FREE db "Free$"
OS_XMS db "OS XMS UMB$"
TOP db "Top memory$"
DOS db "MS DOS$"
BLOCK db "Control block 386MAX UMB$"
BLOCKED db "Blocked 386MAX$"
S_386MAX db "386MAX UMB$"
S_SIZE db 13, 10, "Size: $"
TETR_TO_HEX PROC near
    and AL,0Fh
    cmp AL,09
    jbe NEXT
    add AL,07
    NEXT: add AL,30h
    ret
TETR_TO_HEX ENDP

BYTE_TO_HEX PROC near
    push CX
    mov AH,AL
    call TETR_TO_HEX
    xchg AL,AH
    mov CL,4
    shr AL,CL
    call TETR_TO_HEX
    pop CX
    ret
BYTE_TO_HEX ENDP

WRITE_DEC PROC near
    push AX
    push BX
    push CX
    push DX
    xor CX,CX
    mov BX,10
```

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

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

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

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

```
WRITE_HEX ENDP

WRITE_AVIABLE_MEMORY PROC near
    push AX
    push BX
    push DX
    mov DX, offset AVIABLE
    call WRITE_STRING
    mov AH,4AH
    mov BX,0FFFFH
    int 21H
    mov AX,BX
    mov BX,10H
    mul BX
    call WRITE_DEC
    mov DX, offset BYTES
    call WRITE_STRING
    pop DX
    pop BX
    pop AX
    ret
WRITE_AVIABLE_MEMORY ENDP

WRITE_EXTENDED_MEMORY PROC near
    push AX
    push BX
    push DX
    mov DX, offset EXTENDED
    call WRITE_STRING
    mov AL,30H
    out 70H,AL
    in AL,71H
    mov BL,AL
    mov AL,31H
    out 70H,AL
    in AL,71H
    mov BH,AL
    mov AX,BX
    xor DX,DX
    call WRITE_DEC
    mov DX, offset KBYTES
    call WRITE_STRING
    pop DX
    pop BX
    pop AX
    ret
WRITE_EXTENDED_MEMORY ENDP

WRITE_MCB PROC near
    push AX
```

```
    push CX
    push DX
    push ES
    push SI
    xor CX,CX
    mov AH,52H
    int 21H
    mov AX,ES:[BX-2]
    mov ES,AX
GET_MCB:
    inc CX
    mov DX, offset MCB
    push CX
    call WRITE_STRING
    xor DX,DX
    mov AX,CX
    call WRITE_DEC
    mov DX, offset ENTER
    call WRITE_STRING
    xor AX,AX
    mov AL,ES:[0H]
    push AX
    mov AX,ES:[1H]
    cmp AX,0H
    je WRITING_FREE
    cmp AX,6H
    je WRITING_OS_XMS
    cmp AX,7H
    je WRITING_TOP
    cmp AX,8H
    je WRITING_DOS
    cmp AX,0FFFAH
    je WRITING_BLOCK
    cmp AX,0FFFDH
    je WRITING_BLOCKED
    cmp AX,0FFFEH
    je WRITING_386MAX
    xor DX,DX
    call WRITE_HEX
    jmp GET_SIZE
WRITING_FREE:
    mov DX, offset FREE
    jmp WRITING
WRITING_OS_XMS:
    mov DX, offset OS_XMS
    jmp WRITING
WRITING_TOP:
    mov DX, offset TOP
    jmp WRITING
WRITING_DOS:
```

```
    mov DX, offset DOS
    jmp WRITING
WRITING_BLOCK:
    mov DX, offset DOS
    jmp WRITING
WRITING_BLOCKED:
    mov DX, offset DOS
    jmp WRITING
WRITING_386MAX:
    mov DX, offset S_386MAX
WRITING:
    call WRITE_STRING
GET_SIZE:
    mov DX, offset S_SIZE
    call WRITE_STRING
    mov AX,ES:[3H]
    mov BX,10H
    mul BX
    call WRITE_DEC
    mov DX, offset BYTES
    call WRITE_STRING
    xor SI,SI
    mov CX,8
GET_LAST:
    mov DL,ES:[SI+8H]
    call WRITE_SYMBOL
    inc SI
    loop GET_LAST
    mov DX,offset ENTER
    call WRITE_STRING
    mov AX,ES:[3H]
    mov BX,ES
    add BX,AX
    inc BX
    mov ES,BX
    pop AX
    pop CX
    cmp AL,5AH
    je END_WRITING
    jmp GET_MCB
END_WRITING:
    pop SI
    pop ES
    pop DX
    pop CX
    pop AX
    ret
WRITE_MCB ENDP

BEGIN:
```

```
    call WRITE_AVIABLE_MEMORY
    call WRITE_EXTENDED_MEMORY
    call WRITE_MCB
    xor AL,AL
    mov AH,4CH
    int 21H
LR3 ENDS
END START
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ВТОРОЙ ВЕРСИИ ПРОГРАММЫ

```
LR3 SEGMENT
ASSUME CS:LR3, DS:LR3, ES:NOTHING, SS:NOTHING
ORG 100H
START: JMP BEGIN
ENTER db 13, 10, '$'
AVIABLE db "Aviable memory: $"
BYTES db " bytes", 13, 10, '$'
EXTENDED db "Extended memory: $"
KBYTES db " kbytes", 13, 10, '$'
MCB db "MCB: $"
FREE db "Free$"
OS_XMS db "OS XMS UMB$"
TOP db "Top memory$"
DOS db "MS DOS$"
BLOCK db "Control block 386MAX UMB$"
BLOCKED db "Blocked 386MAX$"
S_386MAX db "386MAX UMB$"
S_SIZE db 13, 10, "Size: $"
FREE_SUCCES db "Memory was free", 13, 10, '$'
FREE_ERROR db "Memory wasn't free", 13, 10, '$'
TETR_TO_HEX PROC near
    and AL,0Fh
    cmp AL,09
    jbe NEXT
    add AL,07
    NEXT: add AL,30h
    ret
TETR_TO_HEX ENDP

BYTE_TO_HEX PROC near
    push CX
    mov AH,AL
    call TETR_TO_HEX
    xchg AL,AH
    mov CL,4
    shr AL,CL
    call TETR_TO_HEX
    pop CX
    ret
BYTE_TO_HEX ENDP

WRITE_DEC PROC near
    push AX
    push BX
    push CX
    push DX
    xor CX,CX
```

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

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

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

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

```
    ret
WRITE_HEX ENDP

WRITE_AVIABLE_MEMORY PROC near
    push AX
    push BX
    push DX
    mov DX, offset AVIABLE
    call WRITE_STRING
    mov AH,4AH
    mov BX,0FFFFH
    int 21H
    mov AX,BX
    mov BX,10H
    mul BX
    call WRITE_DEC
    mov DX, offset BYTES
    call WRITE_STRING
    pop DX
    pop BX
    pop AX
    ret
WRITE_AVIABLE_MEMORY ENDP

WRITE_EXTENDED_MEMORY PROC near
    push AX
    push BX
    push DX
    mov DX, offset EXTENDED
    call WRITE_STRING
    mov AL,30H
    out 70H,AL
    in AL,71H
    mov BL,AL
    mov AL,31H
    out 70H,AL
    in AL,71H
    mov BH,AL
    mov AX,BX
    xor DX,DX
    call WRITE_DEC
    mov DX, offset KBYTES
    call WRITE_STRING
    pop DX
    pop BX
    pop AX
    ret
WRITE_EXTENDED_MEMORY ENDP

WRITE_MCB PROC near
```

```
    push AX
    push CX
    push DX
    push ES
    push SI
    xor CX,CX
    mov AH,52H
    int 21H
    mov AX,ES:[BX-2]
    mov ES,AX
    mov DX, offset MCB
    call WRITE_STRING
GET_MCB:
    inc CX
    push CX
    xor DX,DX
    mov AX,CX
    ;call WRITE_DEC
    mov DX, offset ENTER
    call WRITE_STRING
    xor AX,AX
    mov AL,ES:[0H]
    push AX
    mov AX,ES:[1H]
    cmp AX,0H
    je WRITING_FREE
    cmp AX,6H
    je WRITING_OS_XMS
    cmp AX,7H
    je WRITING_TOP
    cmp AX,8H
    je WRITING_DOS
    cmp AX,0FFFAH
    je WRITING_BLOCK
    cmp AX,0FFFDH
    je WRITING_BLOCKED
    cmp AX,0FFFEH
    je WRITING_386MAX
    xor DX,DX
    call WRITE_HEX
    jmp GET_SIZE
WRITING_FREE:
    mov DX, offset FREE
    jmp WRITING
WRITING_OS_XMS:
    mov DX, offset OS_XMS
    jmp WRITING
WRITING_TOP:
    mov DX, offset TOP
    jmp WRITING
```

```
WRITING_DOS:
    mov DX, offset DOS
    jmp WRITING
WRITING_BLOCK:
    mov DX, offset DOS
    jmp WRITING
WRITING_BLOCKED:
    mov DX, offset DOS
    jmp WRITING
WRITING_386MAX:
    mov DX, offset S_386MAX
WRITING:
    call WRITE_STRING
GET_SIZE:
    mov DX, offset S_SIZE
    call WRITE_STRING
    mov AX,ES:[3H]
    mov BX,10H
    mul BX
    call WRITE_DEC
    mov DX, offset BYTES
    call WRITE_STRING
    xor SI,SI
    mov CX,8
GET_LAST:
    mov DL,ES:[SI+8H]
    call WRITE_SYMBOL
    inc SI
    loop GET_LAST
    mov AX,ES:[3H]
    mov BX,ES
    add BX,AX
    inc BX
    mov ES,BX
    pop AX
    pop CX
    cmp AL,5AH
    je END_WRITING
    ;mov DX,offset ENTER
    ;call WRITE_STRING
    jmp GET_MCB
END_WRITING:
    pop SI
    pop ES
    pop DX
    pop CX
    pop AX
    ret
WRITE_MCB ENDP
```

```
FREE_MEMORY PROC near
    push AX
     push BX
     push DX
     mov BX, offset LR3_END
     add BX, 10FH
     shr BX, 1
    shr BX, 1
    shr BX, 1
    shr BX, 1
     mov AH, 4AH
    int 21H
     jnc SUCCES
     mov DX, offset FREE_ERROR
     call WRITE_STRING
     jmp END_FREE

SUCCES:
     mov DX, offset FREE_SUCCES
     call WRITE_STRING
END_FREE:
     pop DX
     pop BX
     pop AX
    ret
FREE_MEMORY ENDP

BEGIN:
    call WRITE_AVIABLE_MEMORY
    call FREE_MEMORY
    call WRITE_EXTENDED_MEMORY
    call WRITE_MCB
    xor AL,AL
    mov AH,4CH
    int 21H
    DW 128 dup(0)
LR3_END:
LR3 ENDS
END START
```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД ТРЕТЬЕЙ ВЕРСИИ ПРОГРАММЫ

```
LR3 SEGMENT
ASSUME CS:LR3, DS:LR3, ES:NOTHING, SS:NOTHING
ORG 100H
START: JMP BEGIN
ENTER db 13, 10, '$'
AVIABLE db "Aviable memory: $"
BYTES db " bytes", 13, 10, '$'
EXTENDED db "Extended memory: $"
KBYTES db " kbytes", 13, 10, '$'
MCB db "MCB: $"
FREE db "Free$"
OS_XMS db "OS XMS UMB$"
TOP db "Top memory$"
DOS db "MS DOS$"
BLOCK db "Control block 386MAX UMB$"
BLOCKED db "Blocked 386MAX$"
S_386MAX db "386MAX UMB$"
S_SIZE db 13, 10, "Size: $"
FREE_SUCCES db "Memory was free", 13, 10, '$'
FREE_ERROR db "Memory wasn't free", 13, 10, '$'
REQUEST_SUCCES db "64KB was request", 13, 10, '$'
REQUEST_ERROR db "64KB wasn't request", 13, 10, '$'
TETR_TO_HEX PROC near
    and AL,0Fh
    cmp AL,09
    jbe NEXT
    add AL,07
    NEXT: add AL,30h
    ret
TETR_TO_HEX ENDP

BYTE_TO_HEX PROC near
    push CX
    mov AH,AL
    call TETR_TO_HEX
    xchg AL,AH
    mov CL,4
    shr AL,CL
    call TETR_TO_HEX
    pop CX
    ret
BYTE_TO_HEX ENDP

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

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

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

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

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

```
    mov DL, AL
    call WRITE_SYMBOL
    ret
WRITE_HEX ENDP

WRITE_AVIABLE_MEMORY PROC near
    push AX
    push BX
    push DX
    mov DX, offset AVIABLE
    call WRITE_STRING
    mov AH,4AH
    mov BX,0FFFFH
    int 21H
    mov AX,BX
    mov BX,10H
    mul BX
    call WRITE_DEC
    mov DX, offset BYTES
    call WRITE_STRING
    pop DX
    pop BX
    pop AX
    ret
WRITE_AVIABLE_MEMORY ENDP

WRITE_EXTENDED_MEMORY PROC near
    push AX
    push BX
    push DX
    mov DX, offset EXTENDED
    call WRITE_STRING
    mov AL,30H
    out 70H,AL
    in AL,71H
    mov BL,AL
    mov AL,31H
    out 70H,AL
    in AL,71H
    mov BH,AL
    mov AX,BX
    xor DX,DX
    call WRITE_DEC
    mov DX, offset KBYTES
    call WRITE_STRING
    pop DX
    pop BX
    pop AX
    ret
WRITE_EXTENDED_MEMORY ENDP
```

```
WRITE_MCB PROC near
    push AX
    push CX
    push DX
    push ES
    push SI
    xor CX,CX
    mov AH,52H
    int 21H
    mov AX,ES:[BX-2]
    mov ES,AX
    mov DX, offset MCB
    call WRITE_STRING
GET_MCB:
    inc CX
    push CX
    xor DX,DX
    mov AX,CX
    ;call WRITE_DEC
    mov DX, offset ENTER
    call WRITE_STRING
    xor AX,AX
    mov AL,ES:[0H]
    push AX
    mov AX,ES:[1H]
    cmp AX,0H
    je WRITING_FREE
    cmp AX,6H
    je WRITING_OS_XMS
    cmp AX,7H
    je WRITING_TOP
    cmp AX,8H
    je WRITING_DOS
    cmp AX,0FFFAH
    je WRITING_BLOCK
    cmp AX,0FFFDH
    je WRITING_BLOCKED
    cmp AX,0FFFEH
    je WRITING_386MAX
    xor DX,DX
    call WRITE_HEX
    jmp GET_SIZE
WRITING_FREE:
    mov DX, offset FREE
    jmp WRITING
WRITING_OS_XMS:
    mov DX, offset OS_XMS
    jmp WRITING
WRITING_TOP:
```

```
    mov DX, offset TOP
    jmp WRITING
WRITING_DOS:
    mov DX, offset DOS
    jmp WRITING
WRITING_BLOCK:
    mov DX, offset DOS
    jmp WRITING
WRITING_BLOCKED:
    mov DX, offset DOS
    jmp WRITING
WRITING_386MAX:
    mov DX, offset S_386MAX
WRITING:
    call WRITE_STRING
GET_SIZE:
    mov DX, offset S_SIZE
    call WRITE_STRING
    mov AX,ES:[3H]
    mov BX,10H
    mul BX
    call WRITE_DEC
    mov DX, offset BYTES
    call WRITE_STRING
    xor SI,SI
    mov CX,8
GET_LAST:
    mov DL,ES:[SI+8H]
    call WRITE_SYMBOL
    inc SI
    loop GET_LAST
    mov AX,ES:[3H]
    mov BX,ES
    add BX,AX
    inc BX
    mov ES,BX
    pop AX
    pop CX
    cmp AL,5AH
    je END_WRITING
    ;mov DX,offset ENTER
    ;call WRITE_STRING
    jmp GET_MCB
END_WRITING:
    pop SI
    pop ES
    pop DX
    pop CX
    pop AX
    ret
```

```
WRITE_MCB ENDP

FREE_MEMORY PROC near
    push AX
     push BX
     push DX
     mov BX, offset LR3_END
     add BX, 10FH
     shr BX, 1
    shr BX, 1
    shr BX, 1
    shr BX, 1
     mov AH, 4AH
    int 21H
     jnc SUCCES
     mov DX, offset FREE_ERROR
     call WRITE_STRING
     jmp END_FREE

SUCCES:
     mov DX, offset FREE_SUCCES
     call WRITE_STRING
END_FREE:
     pop DX
     pop BX
     pop AX
    ret
FREE_MEMORY ENDP

REQUEST_64KB PROC near
    push AX
     push BX
     push DX
     mov BX, 1000H
     mov AH, 48H
     int 21H
     jnc WRITE_REQUEST
     mov DX, offset REQUEST_ERROR
     call WRITE_STRING
     jmp END_REQUEST
WRITE_REQUEST:
     mov DX, offset REQUEST_SUCCES
     call WRITE_STRING
END_REQUEST:
     pop DX
     pop BX
     pop AX
    ret
REQUEST_64KB ENDP
```

```
BEGIN:
    call WRITE_AVIABLE_MEMORY
    call FREE_MEMORY
    call REQUEST_64KB
    call WRITE_EXTENDED_MEMORY
    call WRITE_MCB
    xor AL,AL
    mov AH,4CH
    int 21H
    DW 128 dup(0)
LR3_END:
LR3 ENDS
END START
```

ПРИЛОЖЕНИЕ Г

ИСХОДНЫЙ КОД ЧЕТВЕРТОЙ ВЕРСИИ ПРОГРАММЫ

```
LR3 SEGMENT
ASSUME CS:LR3, DS:LR3, ES:NOTHING, SS:NOTHING
ORG 100H
START: JMP BEGIN
ENTER db 13, 10, '$'
AVIABLE db "Aviable memory: $"
BYTES db " bytes", 13, 10, '$'
EXTENDED db "Extended memory: $"
KBYTES db " kbytes", 13, 10, '$'
MCB db "MCB: $"
FREE db "Free$"
OS_XMS db "OS XMS UMB$"
TOP db "Top memory$"
DOS db "MS DOS$"
BLOCK db "Control block 386MAX UMB$"
BLOCKED db "Blocked 386MAX$"
S_386MAX db "386MAX UMB$"
S_SIZE db 13, 10, "Size: $"
FREE_SUCCES db "Memory was free", 13, 10, '$'
FREE_ERROR db "Memory wasn't free", 13, 10, '$'
REQUEST_SUCCES db "64KB was request", 13, 10, '$'
REQUEST_ERROR db "64KB wasn't request", 13, 10, '$'
TETR_TO_HEX PROC near
    and AL,0Fh
    cmp AL,09
    jbe NEXT
    add AL,07
    NEXT: add AL,30h
    ret
TETR_TO_HEX ENDP

BYTE_TO_HEX PROC near
    push CX
    mov AH,AL
    call TETR_TO_HEX
    xchg AL,AH
    mov CL,4
    shr AL,CL
    call TETR_TO_HEX
    pop CX
    ret
BYTE_TO_HEX ENDP

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

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

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

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

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

```
    mov DL, AL
    call WRITE_SYMBOL
    ret
WRITE_HEX ENDP

WRITE_AVIABLE_MEMORY PROC near
    push AX
    push BX
    push DX
    mov DX, offset AVIABLE
    call WRITE_STRING
    mov AH,4AH
    mov BX,0FFFFH
    int 21H
    mov AX,BX
    mov BX,10H
    mul BX
    call WRITE_DEC
    mov DX, offset BYTES
    call WRITE_STRING
    pop DX
    pop BX
    pop AX
    ret
WRITE_AVIABLE_MEMORY ENDP

WRITE_EXTENDED_MEMORY PROC near
    push AX
    push BX
    push DX
    mov DX, offset EXTENDED
    call WRITE_STRING
    mov AL,30H
    out 70H,AL
    in AL,71H
    mov BL,AL
    mov AL,31H
    out 70H,AL
    in AL,71H
    mov BH,AL
    mov AX,BX
    xor DX,DX
    call WRITE_DEC
    mov DX, offset KBYTES
    call WRITE_STRING
    pop DX
    pop BX
    pop AX
    ret
WRITE_EXTENDED_MEMORY ENDP
```

```
WRITE_MCB PROC near
    push AX
    push CX
    push DX
    push ES
    push SI
    xor CX,CX
    mov AH,52H
    int 21H
    mov AX,ES:[BX-2]
    mov ES,AX
    mov DX, offset MCB
    call WRITE_STRING
GET_MCB:
    inc CX
    push CX
    xor DX,DX
    mov AX,CX
    ;call WRITE_DEC
    mov DX, offset ENTER
    call WRITE_STRING
    xor AX,AX
    mov AL,ES:[0H]
    push AX
    mov AX,ES:[1H]
    cmp AX,0H
    je WRITING_FREE
    cmp AX,6H
    je WRITING_OS_XMS
    cmp AX,7H
    je WRITING_TOP
    cmp AX,8H
    je WRITING_DOS
    cmp AX,0FFFAH
    je WRITING_BLOCK
    cmp AX,0FFFDH
    je WRITING_BLOCKED
    cmp AX,0FFFEH
    je WRITING_386MAX
    xor DX,DX
    call WRITE_HEX
    jmp GET_SIZE
WRITING_FREE:
    mov DX, offset FREE
    jmp WRITING
WRITING_OS_XMS:
    mov DX, offset OS_XMS
    jmp WRITING
WRITING_TOP:
```

```
    mov DX, offset TOP
    jmp WRITING
WRITING_DOS:
    mov DX, offset DOS
    jmp WRITING
WRITING_BLOCK:
    mov DX, offset DOS
    jmp WRITING
WRITING_BLOCKED:
    mov DX, offset DOS
    jmp WRITING
WRITING_386MAX:
    mov DX, offset S_386MAX
WRITING:
    call WRITE_STRING
GET_SIZE:
    mov DX, offset S_SIZE
    call WRITE_STRING
    mov AX,ES:[3H]
    mov BX,10H
    mul BX
    call WRITE_DEC
    mov DX, offset BYTES
    call WRITE_STRING
    xor SI,SI
    mov CX,8
GET_LAST:
    mov DL,ES:[SI+8H]
    call WRITE_SYMBOL
    inc SI
    loop GET_LAST
    mov AX,ES:[3H]
    mov BX,ES
    add BX,AX
    inc BX
    mov ES,BX
    pop AX
    pop CX
    cmp AL,5AH
    je END_WRITING
    ;mov DX,offset ENTER
    ;call WRITE_STRING
    jmp GET_MCB
END_WRITING:
    pop SI
    pop ES
    pop DX
    pop CX
    pop AX
    ret
```

```
WRITE_MCB ENDP

FREE_MEMORY PROC near
    push AX
     push BX
     push DX
     mov BX, offset LR3_END
     add BX, 10FH
     shr BX, 1
    shr BX, 1
    shr BX, 1
    shr BX, 1
     mov AH, 4AH
    int 21H
     jnc SUCCES
     mov DX, offset FREE_ERROR
     call WRITE_STRING
     jmp END_FREE

SUCCES:
     mov DX, offset FREE_SUCCES
     call WRITE_STRING
END_FREE:
     pop DX
     pop BX
     pop AX
    ret
FREE_MEMORY ENDP

REQUEST_64KB PROC near
    push AX
     push BX
     push DX
     mov BX, 1000H
     mov AH, 48H
     int 21H
     jnc WRITE_REQUEST
     mov DX, offset REQUEST_ERROR
     call WRITE_STRING
     jmp END_REQUEST
WRITE_REQUEST:
     mov DX, offset REQUEST_SUCCES
     call WRITE_STRING
END_REQUEST:
     pop DX
     pop BX
     pop AX
    ret
REQUEST_64KB ENDP
```

```
BEGIN:
    call WRITE_AVIABLE_MEMORY
    call REQUEST_64KB
    call FREE_MEMORY
    call WRITE_EXTENDED_MEMORY
    call WRITE_MCB
    xor AL,AL
    mov AH,4CH
    int 21H
    DW 128 dup(0)
LR3_END:
LR3 ENDS
END START
```