**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.

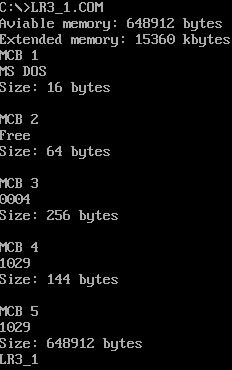


Рисунок 1 – Результат работы программы

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

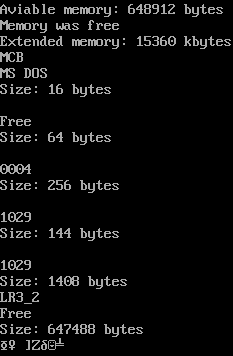


Рисунок 2 – Результат работы 2 версии программы

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.

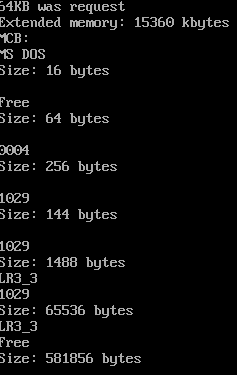


Рисунок 3 – Результат работы 3 версии программы

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.

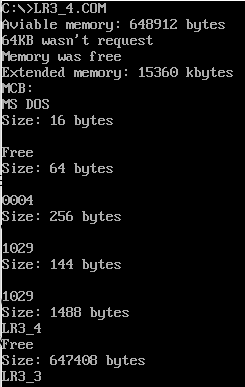


Рисунок 4 – Результат работы 4 версии программы

Память не была выделена.

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

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

1. *Где MCB блок вашей программы в списке?*

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

1. *Какой размер памяти занимает программа в каждом случае?*

В первой версии вся доступная память (648 912 байт). Во второй и четвертой версиях 1408 и 1488 байт соответственно, т.к. была освобождена память. В третьей версии 1488 + 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