**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 |  | Почаев Н.А. |
| Преподаватель |  | Ефремов М.А. |

Санкт-Петербург

2020

## Цель работы.

Исследование структур данных и работы функций управления памятью ядра операционной системы. Изучение нестраничной памяти и способа управления динамическими разделами.

## 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 ниже:

|  |  |  |
| --- | --- | --- |
| 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. Размер расширенной памяти в Кбайтах можно определить, обращаясь к ячейкам CMOS следующим образом:

mov AL,30h ; запись адреса ячейки CMOS

out 70h,AL

in AL,71h ; чтение младшего байта

mov BL,AL ; размера расширенной памяти

mov AL,31h ; запись адреса ячейки CMOS

out 70h,AL

in AL,71h ; чтение старшего байта

; размера расширенной памяти

## Описание программы.

В результате выполнения данной лабораторной работы была написана программа, содержащая следующие функции, описанные в табл. 1.

Таблица 1 – Функции, реализованный в программе

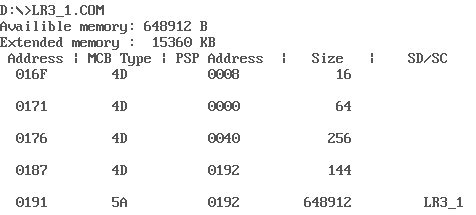
|  |  |
| --- | --- |
| GET\_AVAILABLE\_MEMORY | Получение размера доступной памяти |
| GET\_EXTENDED\_MEMORY | Получение размера расширенной памяти |
| GET\_MCB\_DATA | Получение данных одного MSB блока |
| GET\_ALL\_MSB\_DATA | Получение данных со всех MSB блоков |
| PRINT\_STRING | Вывод строки на экран |

## Выполнение работы.

Выполнение работы производилось на базе операционной системы Windows XP (32 bit), запускаемой в системе виртуализации VMware Workstation, в редакторе Notepad++. Сборка и отладка модулей производились с помощью компилятора MASM и отладчика AFD. Также в работе был использован консольный файловый менеджер Far Manager и HEX-редактор HxD. Для дополнительного тестирования и проверки функциональности программы использовался DOSBox.

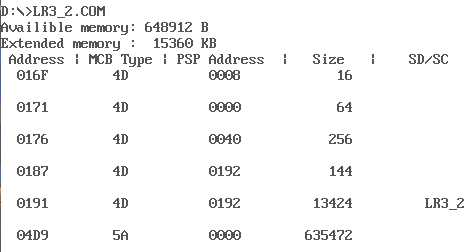
Далее представлены результаты работы программы. Pre.S. (=size para) – размер участка, указанный в байтах.

1. На рис. ниже представлен результат работы модуля LR3\_1.COM



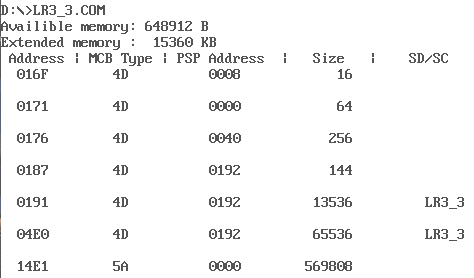
Из него видно, что программа занимает максимум памяти, т.к. при запросе размера доступной памяти мы выделяем, столько памяти, сколько возможно.

1. На рис. ниже представлен результат работы модуля LR3\_2.COM



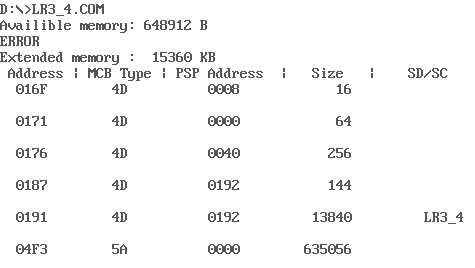
В данном случае память освобождается программой. В итоге её остается столько, сколько занимает программа.

1. На рис. ниже представлен результат работы модуля LR3\_3.COM



В данном случае мы сначала выделяем всю доступную память, потом освобождаем то, что не нужно. Затем запрашиваем блок памяти 64 кб, в итоге система выделяет нам ещё 64 кб памяти.

1. На рис. ниже представлен результат работы модуля LR3\_4.COM



В данном случае мы выделяем всё доступную память, а затем дополнительно запрашиваем 64 кб. В результате чего возникает ошибка. Её причина в том, что в первый раз уже была выделена вся доступная память и больше ОС выдать не может.

## Выводы.

В ходе выполнения лабораторной работы были исследованы структуры данных и работа функций управления памятью ядра ОС.

## Ответы на контрольные вопросы.

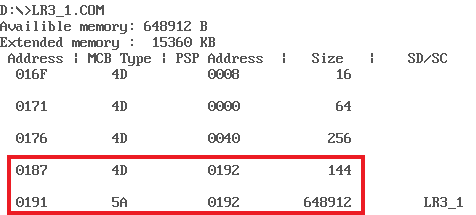
1. **Что означает “доступный объём памяти”?**

Максимальный объем памяти, который может быть доступен для запуска и

выполнения программ. В этом мы убеждаемся в четвёртом пункте данной л.р., когда после выделения всей доступной памяти, мы пытаемся выделить ещё, но получаем ошибку.

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

Данный блок выделен красным на рис. ниже.



В случае LR3\_{1,2,4} MCB блок программы – пятый, в LR3\_3 также 6-ой, для дополнительно выделяемой памяти.

1. Какой размер памяти занимает программа в каждом случае?

* LR3\_1: 648912 байт
* LR3\_2: 13328 байт.
* LR3\_3: 13440 байт (без блока в 64кб).
* LR3\_4: 14048 байт.

# Приложение А Исходный код программы. LR3\_1.ASM

TESTPC SEGMENT

ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING

org 100H

START: JMP BEGIN

A\_MEMORY db 'Availible memory: B ',0Dh,0Ah,'$'

E\_MEMORY db 'Extended memory : KB ',0Dh,0Ah,'$'

TITLE\_LINE db ' Address | MCB Type | PSP Address | Size | SD/SC ',0Dh,0Ah,'$'

LINE db ' ',0Dh,0Ah,'$'

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

WRD\_TO\_HEX PROC near

push BX

mov BH,AH

call BYTE\_TO\_HEX

mov [DI],AH

dec DI

mov [DI],AL

dec DI

mov AL,BH

call BYTE\_TO\_HEX

mov [DI],AH

dec DI

mov [DI],AL

pop BX

ret

WRD\_TO\_HEX ENDP

WRD\_TO\_DEC PROC near

push CX

push DX

mov CX,10

metka1: div CX

or DL,30h

mov [SI],DL

dec SI

xor DX,DX

cmp AX,10

jae metka1

cmp AL,00h

je end\_pr

or AL,30h

mov [SI],AL

end\_pr: pop DX

pop CX

ret

WRD\_TO\_DEC ENDP

BYTE\_TO\_DEC PROC near

push CX

push DX

xor AH,AH

xor DX,DX

mov CX,10

loop\_bd: div CX

or DL,30h

mov [SI],DL

dec SI

xor DX,DX

cmp AX,10

jae loop\_bd

cmp AL,00h

je end\_l

or AL,30h

mov [SI],AL

end\_l: pop DX

pop CX

ret

BYTE\_TO\_DEC ENDP

A\_MEM PROC near

mov ah, 4ah

mov bx, 0ffffh

int 21h

mov ax, 10h

mul bx

mov si, offset A\_MEMORY

add si, 23

call WRD\_TO\_DEC

ret

A\_MEM ENDP

E\_MEM PROC near

xor dx,dx

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 si, offset E\_MEMORY

add si, 23

call WRD\_TO\_DEC

ret

E\_MEM ENDP

MCB\_BLOCK PROC near

mov di, offset LINE

add di, 5

mov ax, es

call WRD\_TO\_HEX

add di,13

xor ah,ah

mov al,es:[0]

call WRD\_TO\_HEX

mov al,20h

mov [di],al

inc di

mov [di],al

mov ax, es:[1]

add di, 16

call WRD\_TO\_HEX

add di, 17

mov ax,es:[3]

mov bx,10h

mul bx

push si

mov si,di

call WRD\_TO\_DEC

pop si

add di, 10

mov bx, 0h

metka:

mov dl, es:[8+bx]

mov [di], dl

inc di

inc bx

cmp bx, 8h

jne metka

mov ax, es:[3]

mov bl, es:[0]

ret

MCB\_BLOCK ENDP

PRINT PROC near

push ax

mov ah,09h

int 21h

pop ax

ret

PRINT ENDP

BEGIN:

call A\_MEM

mov dx, offset A\_MEMORY

call PRINT

call E\_MEM

mov dx, offset E\_MEMORY

call PRINT

mov dx, offset TITLE\_LINE

call PRINT

mov ah, 52h

int 21h

sub bx, 2h

mov es, es:[bx]

OUTPUT\_LINE:

xor ax,ax

xor bx,bx

xor cx,cx

xor dx,dx

xor di,di

call MCB\_BLOCK

mov dx, offset LINE

call PRINT

mov cx,es

add ax,cx

inc ax

mov es,ax

cmp bl,4Dh

je OUTPUT\_LINE

xor al, al

mov ah, 4ch

int 21h

TESTPC ENDS

END START

# Приложение Б Исходный код программы. LR3\_2.ASM

TESTPC SEGMENT

ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING

org 100H

START: JMP BEGIN

A\_MEMORY db 'Availible memory: B ',0Dh,0Ah,'$'

E\_MEMORY db 'Extended memory : KB ',0Dh,0Ah,'$'

TITLE\_LINE db ' Address | MCB Type | PSP Address | Size | SD/SC ',0Dh,0Ah,'$'

LINE db ' ',0Dh,0Ah,'$'

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

WRD\_TO\_HEX PROC near

push BX

mov BH,AH

call BYTE\_TO\_HEX

mov [DI],AH

dec DI

mov [DI],AL

dec DI

mov AL,BH

call BYTE\_TO\_HEX

mov [DI],AH

dec DI

mov [DI],AL

pop BX

ret

WRD\_TO\_HEX ENDP

WRD\_TO\_DEC PROC near

push CX

push DX

mov CX,10

loop\_b: div CX

or DL,30h

mov [SI],DL

dec SI

xor DX,DX

cmp AX,10

jae loop\_b

cmp AL,00h

je endl

or AL,30h

mov [SI],AL

endl: pop DX

pop CX

ret

WRD\_TO\_DEC ENDP

BYTE\_TO\_DEC PROC near

push CX

push DX

xor AH,AH

xor DX,DX

mov CX,10

loop\_bd: div CX

or DL,30h

mov [SI],DL

dec SI

xor DX,DX

cmp AX,10

jae loop\_bd

cmp AL,00h

je end\_l

or AL,30h

mov [SI],AL

end\_l: pop DX

pop CX

ret

BYTE\_TO\_DEC ENDP

A\_MEM PROC near

mov ah, 4ah

mov bx, 0ffffh

int 21h

mov ax, 10h

mul bx

mov si, offset A\_MEMORY

add si, 23

call WRD\_TO\_DEC

ret

A\_MEM ENDP

E\_MEM PROC near

xor dx,dx

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 si, offset E\_MEMORY

add si, 23

call WRD\_TO\_DEC

ret

E\_MEM ENDP

MCB\_BLOCK PROC near

mov di, offset LINE

add di, 5

mov ax, es

call WRD\_TO\_HEX

add di,13

xor ah,ah

mov al,es:[0]

call WRD\_TO\_HEX

mov al,20h

mov [di],al

inc di

mov [di],al

mov ax, es:[1]

add di, 16

call WRD\_TO\_HEX

add di, 17

mov ax,es:[3]

mov bx,10h

mul bx

push si

mov si,di

call WRD\_TO\_DEC

pop si

add di, 10

mov bx, 0h

metka:

mov dl, es:[8+bx]

mov [di], dl

inc di

inc bx

cmp bx, 8h

jne metka

mov ax, es:[3]

mov bl, es:[0]

ret

MCB\_BLOCK ENDP

PRINT PROC near

push ax

mov ah,09h

int 21h

pop ax

ret

PRINT ENDP

BEGIN:

call A\_MEM

mov dx, offset A\_MEMORY

call PRINT

mov ah, 4ah

mov bx, offset end\_size

int 21h

call E\_MEM

mov dx, offset E\_MEMORY

call PRINT

mov dx, offset TITLE\_LINE

call PRINT

mov ah, 52h

int 21h

sub bx, 2h

mov es, es:[bx]

OUTPUT\_LINE:

xor ax,ax

xor bx,bx

xor cx,cx

xor dx,dx

xor di,di

call MCB\_BLOCK

mov dx, offset LINE

call PRINT

mov cx,es

add ax,cx

inc ax

mov es,ax

cmp bl,4Dh

je OUTPUT\_LINE

xor al, al

mov ah, 4ch

int 21h

end\_size db 0

TESTPC ENDS

END START

# Приложение В Исходный код программы. LR3\_3.ASM

TESTPC SEGMENT

ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING

org 100H

START: JMP BEGIN

A\_MEMORY db 'Availible memory: B ',0Dh,0Ah,'$'

E\_MEMORY db 'Extended memory : KB ',0Dh,0Ah,'$'

TITLE\_LINE db ' Address | MCB Type | PSP Address | Size | SD/SC ',0Dh,0Ah,'$'

LINE db ' ',0Dh,0Ah,'$'

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

WRD\_TO\_HEX PROC near

push BX

mov BH,AH

call BYTE\_TO\_HEX

mov [DI],AH

dec DI

mov [DI],AL

dec DI

mov AL,BH

call BYTE\_TO\_HEX

mov [DI],AH

dec DI

mov [DI],AL

pop BX

ret

WRD\_TO\_HEX ENDP

WRD\_TO\_DEC PROC near

push CX

push DX

mov CX,10

loop\_b: div CX

or DL,30h

mov [SI],DL

dec SI

xor DX,DX

cmp AX,10

jae loop\_b

cmp AL,00h

je endl

or AL,30h

mov [SI],AL

endl: pop DX

pop CX

ret

WRD\_TO\_DEC ENDP

BYTE\_TO\_DEC PROC near

push CX

push DX

xor AH,AH

xor DX,DX

mov CX,10

loop\_bd: div CX

or DL,30h

mov [SI],DL

dec SI

xor DX,DX

cmp AX,10

jae loop\_bd

cmp AL,00h

je end\_l

or AL,30h

mov [SI],AL

end\_l: pop DX

pop CX

ret

BYTE\_TO\_DEC ENDP

A\_MEM PROC near

mov ah, 4ah

mov bx, 0ffffh

int 21h

mov ax, 10h

mul bx

mov si, offset A\_MEMORY

add si, 23

call WRD\_TO\_DEC

ret

A\_MEM ENDP

E\_MEM PROC near

xor dx,dx

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 si, offset E\_MEMORY

add si, 23

call WRD\_TO\_DEC

ret

E\_MEM ENDP

MCB\_BLOCK PROC near

mov di, offset LINE

add di, 5

mov ax, es

call WRD\_TO\_HEX

add di,13

xor ah,ah

mov al,es:[0]

call WRD\_TO\_HEX

mov al,20h

mov [di],al

inc di

mov [di],al

mov ax, es:[1]

add di, 16

call WRD\_TO\_HEX

add di, 17

mov ax,es:[3]

mov bx,10h

mul bx

push si

mov si,di

call WRD\_TO\_DEC

pop si

add di, 10

mov bx, 0h

metka:

mov dl, es:[8+bx]

mov [di], dl

inc di

inc bx

cmp bx, 8h

jne metka

mov ax, es:[3]

mov bl, es:[0]

ret

MCB\_BLOCK ENDP

PRINT PROC near

push ax

mov ah,09h

int 21h

pop ax

ret

PRINT ENDP

BEGIN:

call A\_MEM

mov dx, offset A\_MEMORY

call PRINT

mov ah, 4ah

mov bx, offset end\_size

int 21h

mov ah, 48h

mov bx, 1000h

int 21h

call E\_MEM

mov dx, offset E\_MEMORY

call PRINT

mov dx, offset TITLE\_LINE

call PRINT

mov ah, 52h

int 21h

sub bx, 2h

mov es, es:[bx]

OUTPUT\_LINE:

xor ax,ax

xor bx,bx

xor cx,cx

xor dx,dx

xor di,di

call MCB\_BLOCK

mov dx, offset LINE

call PRINT

mov cx,es

add ax,cx

inc ax

mov es,ax

cmp bl,4Dh

je OUTPUT\_LINE

xor al, al

mov ah, 4ch

int 21h

end\_size db 0

TESTPC ENDS

END START

# Приложение Г Исходный код программы. LR3\_4.ASM

TESTPC SEGMENT

ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING

org 100H

START: JMP BEGIN

A\_MEMORY db 'Availible memory: B ',0Dh,0Ah,'$'

E\_MEMORY db 'Extended memory : KB ',0Dh,0Ah,'$'

TITLE\_LINE db ' Address | MCB Type | PSP Address | Size | SD/SC ',0Dh,0Ah,'$'

LINE db ' ',0Dh,0Ah,'$'

ERROR\_LINE db 'ERROR',0Dh,0Ah,'$'

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

WRD\_TO\_HEX PROC near

push BX

mov BH,AH

call BYTE\_TO\_HEX

mov [DI],AH

dec DI

mov [DI],AL

dec DI

mov AL,BH

call BYTE\_TO\_HEX

mov [DI],AH

dec DI

mov [DI],AL

pop BX

ret

WRD\_TO\_HEX ENDP

WRD\_TO\_DEC PROC near

push CX

push DX

mov CX,10

loop\_b: div CX

or DL,30h

mov [SI],DL

dec SI

xor DX,DX

cmp AX,10

jae loop\_b

cmp AL,00h

je endl

or AL,30h

mov [SI],AL

endl: pop DX

pop CX

ret

WRD\_TO\_DEC ENDP

BYTE\_TO\_DEC PROC near

push CX

push DX

xor AH,AH

xor DX,DX

mov CX,10

loop\_bd: div CX

or DL,30h

mov [SI],DL

dec SI

xor DX,DX

cmp AX,10

jae loop\_bd

cmp AL,00h

je end\_l

or AL,30h

mov [SI],AL

end\_l: pop DX

pop CX

ret

BYTE\_TO\_DEC ENDP

A\_MEM PROC near

mov ah, 4ah

mov bx, 0ffffh

int 21h

mov ax, 10h

mul bx

mov si, offset A\_MEMORY

add si, 23

call WRD\_TO\_DEC

ret

A\_MEM ENDP

E\_MEM PROC near

xor dx,dx

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 si, offset E\_MEMORY

add si, 23

call WRD\_TO\_DEC

ret

E\_MEM ENDP

MCB\_BLOCK PROC near

mov di, offset LINE

add di, 5

mov ax, es

call WRD\_TO\_HEX

add di,13

xor ah,ah

mov al,es:[0]

call WRD\_TO\_HEX

mov al,20h

mov [di],al

inc di

mov [di],al

mov ax, es:[1]

add di, 16

call WRD\_TO\_HEX

add di, 17

mov ax,es:[3]

mov bx,10h

mul bx

push si

mov si,di

call WRD\_TO\_DEC

pop si

add di, 10

mov bx, 0h

metka:

mov dl, es:[8+bx]

mov [di], dl

inc di

inc bx

cmp bx, 8h

jne metka

mov ax, es:[3]

mov bl, es:[0]

ret

MCB\_BLOCK ENDP

PRINT PROC near

push ax

mov ah,09h

int 21h

pop ax

ret

PRINT ENDP

BEGIN:

call A\_MEM

mov dx, offset A\_MEMORY

call PRINT

mov ah, 48h

mov bx, 1000h

int 21h

jc err\_1

jmp no\_err\_1

err\_1:

mov dx, offset ERROR\_LINE

call PRINT

no\_err\_1:

mov ah, 4ah

mov bx, offset end\_size

int 21h

call E\_MEM

mov dx, offset E\_MEMORY

call PRINT

mov dx, offset TITLE\_LINE

call PRINT

mov ah, 52h

int 21h

sub bx, 2h

mov es, es:[bx]

OUTPUT\_LINE:

xor ax,ax

xor bx,bx

xor cx,cx

xor dx,dx

xor di,di

call MCB\_BLOCK

mov dx, offset LINE

call PRINT

mov cx,es

add ax,cx

inc ax

mov es,ax

cmp bl,4Dh

je OUTPUT\_LINE

xor al, al

mov ah, 4ch

int 21h

end\_size db 0

TESTPC ENDS

END START