

**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

Преподаватель

\_\_\_\_\_

\_\_\_\_\_

Почаев Н.А.

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

```

D:\>LR3_1.COM
Available memory: 648912 B
Extended memory : 15360 KB
Address | MCB Type | PSP Address | Size | SD/SC
016F    4D    0008        16
0171    4D    0000        64
0176    4D    0040       256
0187    4D    0192       144
0191    5A    0192    648912    LR3_1

```

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

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

```
D:\>LR3_2.COM
Available memory: 648912 B
Extended memory : 15360 KB
```

Address	MCB Type	PSP	Address	Size	SD/SC
016F	4D	0008		16	
0171	4D	0000		64	
0176	4D	0040		256	
0187	4D	0192		144	
0191	4D	0192		13424	LR3_2
04D9	5A	0000		635472	

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

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

```
D:\>LR3_3.COM
Available memory: 648912 B
Extended memory : 15360 KB
```

Address	MCB Type	PSP	Address	Size	SD/SC
016F	4D	0008		16	
0171	4D	0000		64	
0176	4D	0040		256	
0187	4D	0192		144	
0191	4D	0192		13536	LR3_3
04E0	4D	0192		65536	LR3_3
14E1	5A	0000		569808	

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.

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

```
D:\>LR3_4.COM
Available memory: 648912 B
ERROR
Extended memory : 15360 KB
Address | MCB Type | PSP Address | Size | SD/SC
016F    4D    0008        16
0171    4D    0000        64
0176    4D    0040       256
0187    4D    0192       144
0191    4D    0192     13840    LR3_4
04F3    5A    0000     635056
```

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 четвертом пункте данной л.р., когда после выделения всей доступной памяти, мы пытаемся выделить ещё, но получаем ошибку.

### 2. Где MCB блок Вашей программы в списке?

Данный блок выделен красным на рис. ниже.

```
D:\>LR3_1.COM
Availible memory: 648912 B
Extended memory : 15360 KB
```

Address	MCB Type	PSP	Address	Size	SD/SC
016F	4D		0008	16	
0171	4D		0000	64	
0176	4D		0040	256	
0187	4D		0192	144	
0191	5A		0192	648912	LR3_1

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

```
D:\>LR3_3.COM
Availible memory: 648912 B
Extended memory : 15360 KB
```

Address	MCB Type	PSP	Address	Size	SD/SC
016F	4D		0008	16	
0171	4D		0000	64	
0176	4D		0040	256	
0187	4D		0192	144	
0191	4D		0192	13536	LR3_3
04E0	4D		0192	65536	LR3_3
14E1	5A		0000	569808	

Принадлежность определяется по сегментной компоненте адреса резидента – владельца блока (по смещению в один байт располагается в MCB).

### 3. Какой размер памяти занимает программа в каждом случае?

- LR3\_1:  $648912 + 144$  байт 4-ого блока, т.к. его сегментная компонента совпадает с 5-м блоком (далее аналогично)
- LR3\_2:  $13328 + 144$  байт.
- LR3\_3:  $13440 + 144$  байт (без блока в 64кб), а также занимает 65536 байт после выделения (шестой блоков).
- LR3\_4:  $14048 + 144$  байт после неудачной попытки выделения дополнительной памяти и удачного освобождения.



# 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_1:      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_1
    or AL,30h
    mov [SI],AL
end_1: 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_1
        or AL,30h
        mov [SI],AL
end_1:      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_1
    or AL,30h
    mov [SI],AL
end_1: 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