

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

ОТЧЕТ
по лабораторной работе №7
по дисциплине «Операционные системы»
Тема: Построение модуля оверлейной структуры

Студентка гр. 8383

Максимова А.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Исследование возможности построения загрузочного модуля оверлейной структуры.

Основные теоретические положения.

Для организации программы, имеющей оверлейную структуру, используется функция 4B03h прерывания int 21h. Эта функция позволяет в отведенную область памяти, начинающуюся с адреса сегмента, загрузить программу, находящуюся в файле на диске. Передача управления загруженной программе этой функцией не осуществляется и префикс сегмента программы (PSP) не создается.

Перед загрузкой оверлея вызывающая программа должна освободить память по функции 4Ah прерывания int 21h. Затем определить размер оверлея. Это можно сделать с помощью функции 4Eh прерывания int 21h. Перед обращением к функции необходимо определить область памяти размером в 43 байта под буфер DTA, которую функция заполнит, если файл будет найден.

Процедуры, используемые в программе.

Таблица 1 - Процедуры

Название:	Предназначение:
FREE_MEMORY	Процедура, используемая для освобождения места в памяти, которое не требуется программе. В случае если освобождение не может быть выполнено, выводится ошибка.

CATCH_ERROR	Процедура, используемая для определения ошибки, возникшей при освобождении памяти, определяемой по коду в AX.
GET_PROG_PATH	Процедура, используемая для записи местоположения вызываемого модуля.
GET_SIZE_OVERLAY	Процедура, используемая для записи размера оверлейного модуля.
GET_SIZE_FILE	Процедура для работы с DTA.
FIND_ERROR	Процедура, используемая для отлавливания ошибок, возникших при обращении к функции 4Eh.
LOADER	Процедура, используемая для загрузки оверлейного модуля.
CATCH_LOAD_ERROR	Процедура, используемая для отлавливания ошибок, возникших в процессе загрузки оверлейного модуля.
CALL_OVERLAY	Вспомогательная процедура для поочередного вызова процедур, отвечающих за освобождение памяти, получение пути, выполнение загрузки оверлейного модуля.
PRINTF	Процедура печати.
BEGIN	Головная процедура.
TETR_TO_HEX	Процедура перевода тетрады в шестнадцатеричную цифру (символ)
BYTE_TO_HEX	Процедура перевода байта AL в два символа шестнадцатеричного числа
WRD_TO_HEX	Процедура перевода слова в шестнадцатеричную систему счисления

Выполнение работы.

Шаг 1.

Был написан и отлажен программный модуль типа **.EXE**, который выполняет следующие функции:

- 1) Освобождает память для загрузки оверлеев.
- 2) Читает размер файла оверлея и запрашивает объем памяти, достаточный для его загрузки.
- 3) Файл оверлейного сегмента загружается и выполняется.
- 4) Освобождается память, отведенная для оверлейного сегмента.
- 5) Затем действия 1) - 4) выполняются для следующего оверлейного сегмента.

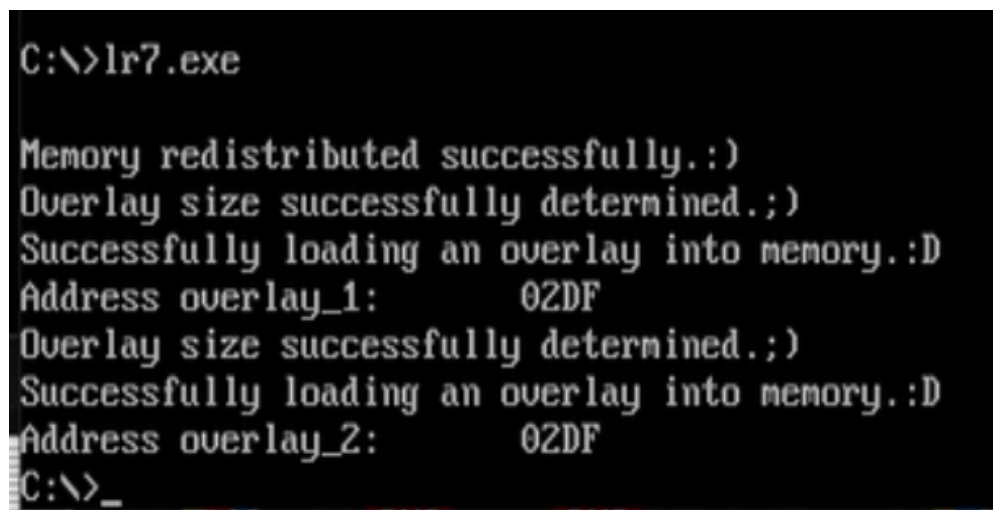
Код программы см. в приложении А.

Шаг 2.

Были написаны и отлажены оверлейные сегменты, выводящие адрес сегмента, в который каждый из них загружен. Код оверлеев см. в приложении Б и В.

Шаг 3.

Приложение, находящееся в одном каталоге с оверлеями, было запущено. Результат работы приложения см. на рис. 1.



```
C:\>lr7.exe

Memory redistributed successfully.:)
Overlay size successfully determined.;)
Successfully loading an overlay into memory.:D
Address overlay_1:      02DF
Overlay size successfully determined.;)
Successfully loading an overlay into memory.:D
Address overlay_2:      02DF
C:\>_
```

Рисунок 1 – Запуск программы

Шаг 4.

Запуск приложения из другого каталога. Результат работы приложения см. на рис. 2.



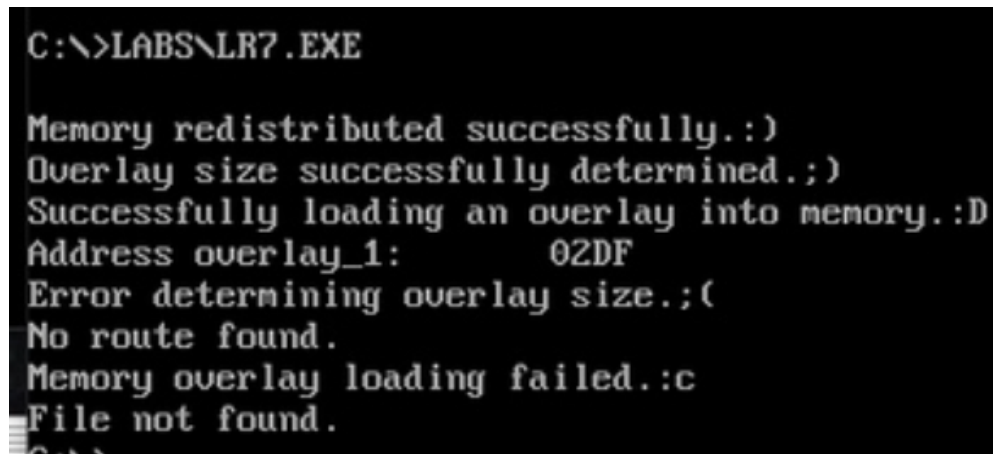
```
C:\>LABS\LR7.EXE

Memory redistributed successfully. :)
Overlay size successfully determined. ;)
Successfully loading an overlay into memory. :D
Address overlay_1:      02DF
Overlay size successfully determined. ;)
Successfully loading an overlay into memory. :D
Address overlay_2:      02DF
```

Рисунок 2 – Запуск программы из другой директории

Шаг 5.

Один из оверлеев был перенесен в другой каталог, запущенное приложение закончилось аварийно. Результат работы приложения см. на рис. 3.



```
C:\>LABS\LR7.EXE

Memory redistributed successfully. :)
Overlay size successfully determined. ;)
Successfully loading an overlay into memory. :D
Address overlay_1:      02DF
Error determining overlay size. :(
No route found.
Memory overlay loading failed. :c
File not found.
```

Рисунок 3 – Запуск программы, когда второй оверлей не находится в текущем каталоге

Ответы на контрольные вопросы.

- 1) Как должна быть устроена программа, если в качестве оверлейного сегмента использовать **.COM** модули?

В программе следует учесть, что для всех **.COM** программ код начинается с адреса 100h (первые 256 байт занимает PSP), то есть обращение к оверлейному модулю должно происходить со смещением равным 100h.

Выводы.

В результате выполнения лабораторной работы была изучена возможность построения загрузочного модуля оверлейной структуры. Был реализован такой модуль, а также два оверлея.

ПРИЛОЖЕНИЕ А

Содержимое файлы LR7.ASM

```
*COMMENT *
Максимова Анастасия, группа 8383 - лабораторная 7
*
;-----
AStack SEGMENT STACK
        DW 256 dup(?)
AStack ENDS
;-----
DATA    SEGMENT                                ;ДАННЫЕ
EOF      EQU      '$'
;-----
        KEEP_PSP      DW      0
        THIS_          DW      0

        OVERLAY_OFFSET DW      0
        ADDRESS_OVERSEG DW      0
        ADDRESS_OVERLAY DD      0
        BUFFER_DTA     DB      43 dup(?) ;область памяти, которую
заполнит функция, если файл будет найден
;-----поиск пути
        ERROR_MEM2     DB      0DH, 0AH, 'Memory allocation
error.', EOF
        ERROR_MEM_FLAG2 DB      0
;-----
        OVERLAY_PATH    DB      64 dup(?)
        OVERLAY1        DB      'OVR1.OVL', 0
        OVERLAY2        DB      'OVR2.OVL', 0
;-----выделение памяти
        ERROR_MEM       DB      0DH, 0AH, 'Memory block size not
changed.:(' , EOF
```

```

NOT_ERROR_MEM DB 0DH, 0AH, 'Memory redistributed
successfully.:)', EOF

ERROR_MEM_7 DB 0DH, 0AH, 'Memory control
block destroyed.', EOF
ERROR_MEM_8 DB 0DH, 0AH, 'Not enough memory
to execute function.', EOF
ERROR_MEM_9 DB 0DH, 0AH, 'Invalid memory
block address.', EOF

ERROR_MEM_FLAG DB 0
;-----
GET_SIZE_NOT_ERROR DB 0DH, 0AH, 'Overlay size
successfully determined.;)', EOF
GET_SIZE_ERROR DB 0DH, 0AH, 'Error determining
overlay size.;(', EOF
ERROR_2 DB 0DH, 0AH, 'File not found.',
EOF
ERROR_3 DB 0DH, 0AH, 'No route found.',
EOF
;-----определение размера памяти
LOAD_ERROR_FLAG DB 0
LOADER_ERROR DB 0DH, 0AH, 'Memory overlay
loading failed.:c', EOF
LOADER_NOT_ERROR DB 0DH, 0AH, 'Successfully loading an
overlay into memory.:D', EOF

LOAD_ERROR1 DB 0DH, 0AH, 'Nonexistent
function.', EOF
LOAD_ERROR2 DB 0DH, 0AH, 'File not
found.', EOF
LOAD_ERROR3 DB 0DH, 0AH, 'No route
found.', EOF

```



```

        LOAD_ERROR4          DB          0DH, 0AH, 'Too many open
files.', EOF
        LOAD_ERROR5          DB          0DH, 0AH, 'No access.',
EOF
        LOAD_ERROR8          DB          0DH,      0AH,      'Little
memory.', EOF
        LOAD_ERROR10         DB          0DH,      0AH,      'Wrong
environment.', EOF
;-----
        DATA_END_FLAG      DB          0
;-----
DATA      ENDS
;-----
CODE      SEGMENT
        ASSUME CS:CODE, DS:DATA, SS:AStack, ES:NOTHING
;-----
PRINTF      PROC  NEAR

        push    AX
        mov     AH, 09h
        int     21h
        pop     AX
        retn

PRINTF      ENDP
;-----
FREE_MEMORY      PROC  NEAR                                ;уменьшить
отведенный блок памяти

        push    AX
        push    BX
        push    CX
        push    DX

```

```

mov     BX, offset CODE_END_FLAG    ;определить
размер памяти необходимый программе
mov     AX, offset DATA_END_FLAG   ;положить    в
BX число параграфов
add     BX, AX
mov     CL, 4                        ;деление
на 16
shr     BX, CL
add     BX, 100h
;дополнительная память

xor     AL, AL
mov     AH, 4Ah
;4Ah - изменить размер блока памяти
int     21h                          ;BX
- новый размер в 16байтных параграфах

;если освобождение памяти не может быть выполнено
;устанавливается флаг переноса CF = 1 и AX
заносится код ошибки (7,8,9)

jnc     NOT_ERROR
;переход, если перенос не установлен
mov     DX, offset ERROR_MEM
call    PRINTF
call    CATCH_ERROR

NOT_ERROR:
mov     DX, offset NOT_ERROR_MEM
call    PRINTF

pop     DX
pop     CX
pop     BX

```

```

                                pop      AX
                                retn
FREE_MEMORY      ENDP
;-----
CATCH_ERROR      PROC NEAR                                ;отлавливание
ошибок при работе с памятью
                                push     AX
                                push     DX

                                mov      ERROR_MEM_FLAG, 1      ;если ошибки с
памятью, то продолжение работы программы не имеет смысла (begin)

                                cmp      AX, 7
                                jne      NEXT_ERROR
                                mov      DX, offset ERROR_MEM_7
                                jmp      END_CATCH

NEXT_ERROR:
                                cmp      AX, 8
                                jne      NEXT_ERROR2
                                mov      DX, offset ERROR_MEM_8
                                jmp      END_CATCH

NEXT_ERROR2:
                                cmp      AX, 9
                                mov      DX, offset ERROR_MEM_9

END_CATCH:
                                call     PRINTF

                                pop      DX
                                pop      AX

CATCH_ERROR      ENDP

```

```

;-----
GET_PROG_PATH      PROC NEAR

    ;2)путь к файлу

    push    AX
    push    ES
    push    DI
    push    SI
    push    CX

    mov     OVERLAY_OFFSET, AX
    mov     AX, KEEP_PSP
    mov     ES, AX

    mov     ES, ES:[002Ch]
    xor     DI, DI
    xor     SI, SI

FIND_PATH:
    mov     AX, ES:[DI]
    inc     DI
    cmp     AX, 0000h                                ;если
встретили два нулевых байта подряд
    je      NEXT_STEP
    jmp     FIND_PATH

NEXT_STEP:
    inc     DI
    mov     AL, ES:[DI]
    cmp                                           AL,    01h
;после располагается маршрут
    jne     NEXT_STEP
    add     DI, 2

```

```

                                mov     SI, offset OVERLAY_PATH

WRITTING:

                                mov     AL, ES:[DI]
                                cmp     AL, 00h                                ;the
end

                                je      FIND_NAME

;последний слэш запоминаем

                                cmp     AL, '\'
                                jne     CONTINUE_
                                mov     THIS_, SI

CONTINUE_:

                                mov     [SI], AL
                                inc     DI
                                inc     SI
                                jmp     WRITTING

FIND_NAME:

                                mov     SI, THIS_
                                inc     SI

                                mov     DI, OVERLAY_OFFSET
                                mov     CX, 8

                                xor     AL, AL

REPEAT_:

                                mov     AL, [DI]
                                mov     [SI], AL
                                inc     DI
                                inc     SI
                                loop    REPEAT_

END_FIND:

```

```

                                pop     CX
                                pop     SI
                                pop     DI
                                pop     ES
                                pop     AX
                                retn
GET_PROG_PATH                 ENDP
;-----
GET_SIZE_OVERLAY PROC NEAR                                ; (3) определение
размера оверлея
                                push    AX
                                push    DX
                                push    CX
                                ;1
установить адрес DTA
                                xor     AL, AL
                                mov     AH, 1Ah
                                mov     DX, offset BUFFER_DTA
                                int     21h
                                ;2
определить размер оверлея используя 4Eh
                                xor     AL, AL
                                mov     AH, 4Eh
                                mov     DX, offset OVERLAY_PATH ;DS:DX
адрес строки ASCIIIZ с именем файла
                                mov     CX, 0                ;CX
атрибут файла для сравнения (для файла 0)
                                int     21h
                                jnc     SIZE_NOT_ERROR        ;переход, если
перенос не установлен

```

```

mov     DX, offset GET_SIZE_ERROR

;CF = 1

call    PRINTF

call    FIND_ERROR
jmp     END_GET_SIZE

SIZE_NOT_ERROR:

mov     DX, offset GET_SIZE_NOT_ERROR
call    PRINTF
call    GET_SIZE_FILE

;CF = 0
END_GET_SIZE:

pop     CX
pop     DX
pop     AX
retn

GET_SIZE_OVERLAY    ENDP

;-----
GET_SIZE_FILE       PROC NEAR ;CF = -0
push     DI
push     AX
push     BX
push     CX
push     DX

mov     DI, offset BUFFER_DTA

mov     BX, [DI + 1Ah]

;размер файла

mov     AX, [DI + 1Ch] ;размер
памяти в байтах

;перевести в параграфы
xor     CH, CH

```

```

mov     CL, 4
shr     BX, CL
mov     CL, 12
shl     AX, CL

add     BX, AX
add     BX, 2

xor     AL, AL
mov     AH, 48h
;распределить память
int     21h

jnc     SEGMENT_SET                ;переход,
если перенос не установлен

mov     DX, offset ERROR_MEM2      ;CF = 1
call    PRINTF

mov     ERROR_MEM_FLAG2, 1
jmp     BYE

SEGMENT_SET:
mov     ADDRESS_OVERSEG, AX

BYE:
pop     DX
pop     CX
pop     BX
pop     AX
pop     DI
retn

GET_SIZE_FILE      ENDP
;-----
FIND_ERROR         PROC NEAR

```



```

                                push    AX
                                push    DX

                                cmp     AX, 2
                                jne     NEXT
                                mov     DX, offset ERROR_2
                                jmp     ENDF

NEXT:

                                cmp     AX, 3
                                mov     DX, offset ERROR_3

ENDF:

                                call    PRINTF
                                pop     DX
                                pop     AX
                                retn

FIND_ERROR                     ENDP
;-----
LOADER      PROC  NEAR

                                push    AX
                                push    ES
                                push    BX
                                push    DX

                                mov     AX, DATA
                                mov     ES, AX

                                mov     DX, offset OVERLAY_PATH    ;DS:DX - указывает
на строку, содержащую путь к оверлею

                                mov     BX, offset ADDRESS_OVERSEG ;ES:BX - сегментный
адрес загрузки программы

                                mov     AX, 4B03h
                                int     21h

```

```

                                jnc         DONE

                                mov         DX, offset LOADER_ERROR
                                call        PRINTF

                                mov         LOAD_ERROR_FLAG, 1

                                call        CATCH_LOAD_ERROR
                                jmp         ENDDD

DONE:                                ;CF =
0
                                mov         DX, offset LOADER_NOT_ERROR
                                call        PRINTF

                                mov         AX, ADDRESS_OVERSEG
                                mov         ES, AX

                                mov         WORD PTR ADDRESS_OVERLAY + 2, AX
                                call        ADDRESS_OVERLAY

                                mov         ES, AX
                                xor         AL, AL
                                mov         AH, 49h
                                int         21h

ENDDD:

                                pop         DX
                                pop         BX
                                pop         ES
                                pop         AX
                                retn

LOADER        ENDP

;-----
CATCH_LOAD_ERROR PROC NEAR

```

```

push AX
push DX

cmp     AX, 1
jne     NOT_1
mov     DX, offset LOAD_ERROR1
jmp     END_CATCH_ER

NOT_1:

cmp     AX, 2
jne     NOT_2
mov     DX, offset LOAD_ERROR2
jmp     END_CATCH_ER

NOT_2:

cmp     AX, 3
jne     NOT_3
mov     DX, offset LOAD_ERROR3
jmp     END_CATCH_ER

NOT_3:

cmp     AX, 4
jne     NOT_4
mov     DX, offset LOAD_ERROR4
jmp     END_CATCH_ER

NOT_4:

cmp     AX, 5
jne     NOT_5
mov     DX, offset LOAD_ERROR5
jmp     END_CATCH_ER

NOT_5:

cmp     AX, 8
jne     NOT_8
mov     DX, offset LOAD_ERROR8
jmp     END_CATCH_ER

NOT_8:

cmp     AX, 10

```

```

                                mov         DX, offset LOAD_ERROR10

END_CATCH_ER:

                                call  PRINTF
                                pop     DX
                                pop     AX
                                retn

CATCH_LOAD_ERROR ENDP
;-----
CALL_OVERLAY      PROC  NEAR

                                call     GET_PROG_PATH                ;2)
получили путь

                                call     GET_SIZE_OVERLAY
;3)определение размера оверлея

                                cmp      ERROR_MEM_FLAG2, 1
                                jne      CONT
                                jmp      EEND

CONT:

                                call     LOADER

EEND:

                                retn

CALL_OVERLAY      ENDP
;-----
BEGIN            PROC  FAR

                                push  DS
                                xor   AX, AX
                                push  AX

                                mov   AX, DATA
                                mov   DS, AX

                                mov   KEEP_PSP, ES

```

```

;-----
        call    FREE_MEMORY                                ;1)перед
загрузкой оверлей программа должна освободить память

        cmp     ERROR_MEM_FLAG, 1                        ;если не
получилось - заканчиваем программу
        jne     CONTINUE
        jmp     END_BEGIN

;-----
CONTINUE:
        mov     AX, offset OVERLAY1
        ;программа 1
        call    CALL_OVERLAY

;-----
        cmp     LOAD_ERROR_FLAG, 1                      ;ошибка при
загрузке первого
        je      END_BEGIN

;-----
CONTINUE2:
        mov     AX, offset OVERLAY2
        ;программа 2
        call    CALL_OVERLAY

;-----
END_BEGIN:
        xor     AL, AL                                    ;выход
в DOS

        mov     AH, 4Ch
        int     21h

BEGIN    ENDP

;-----
CODE_END_FLAG:

```

CODE ENDS
 END BEGIN

ПРИЛОЖЕНИЕ Б

Содержимое файлы OVR1.ASM

COMMENT *

Максимова Анастасия, группа 8383 - лабораторная 7

*

CODE SEGMENT

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

;-----

BEGIN PROC FAR

push DS

push AX

push DI

push DX

mov AX, CS

mov DS, AX

mov DI, offset SMS1

add DI, 30

call WRD_TO_HEX

mov DX, offset SMS1

call PRINTF

END_BEGIN:

pop DX

pop DI

pop AX

pop DS

retf

BEGIN ENDP

```

;-----
EOF            EQU            '$'
SETPR          EQU            30
SMS1           DB             0DH, 0AH, 'Address overlay_1:
', EOF
;-----

PRINTF         PROC NEAR

                push    AX
                mov     AH, 09h
                int     21h
                pop     AX
                retn

PRINTF         ENDP
;-----

TETR_TO_HEX    PROC NEAR

                and     AL, 0Fh

                cmp     AL, 09

                jbe     NEXT

                add     AL, 07

NEXT:

                add     AL, 30h

                retn

TETR_TO_HEX    ENDP
;-----

BYTE_TO_HEX    PROC NEAR
                ;байт в AL переводится в два символа шестн. числа в AX
                push    CX

```



```

        mov     AH, AL
        call    TETR_TO_HEX
        xchg    AL, AH
        mov     CL, 4
        shr     AL, CL
        call    TETR_TO_HEX
; в AL - старшая цифра
        pop     CX
; в AH - младшая
        retn
BYTE_TO_HEX    ENDP
;-----
WRD_TO_HEX     PROC NEAR
; перевод в 16 с/с 16-ти разрядного числа

        ; в AX - число, DI - адрес последнего символа
        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
        retn
WRD_TO_HEX     ENDP
;-----
CODE    ENDS
                END        BEGIN

```

ПРИЛОЖЕНИЕ В

Содержимое файлы OVR2.ASM

COMMENT *

МАКСИМОВА АНАСТАСИЯ, ГРУППА 8383 - ЛАБОРАТОРНАЯ 7

*

CODE SEGMENT

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

;-----

BEGIN PROC FAR

 PUSH DS

 PUSH AX

 PUSH DI

 PUSH DX

 MOV AX, CS

 MOV DS, AX

 MOV DI, OFFSET SMS2

 ADD DI, 30

 CALL WRD_TO_HEX

 MOV DX, OFFSET SMS2

 CALL PRINTF

END_BEGIN:

 POP DX

 POP DI

 POP AX

 POP DS

 RETF

BEGIN ENDP

```

;-----
EOF            EQU            '$'
SETPR          EQU            30
SMS2           DB             0DH, 0AH, 'ADDRESS OVERLAY_2:
                ', EOF
;-----

PRINTF         PROC NEAR

                PUSH         AX
                MOV          AH, 09H
                INT          21H
                POP          AX
                RETN

PRINTF         ENDP
;-----

TETR_TO_HEX    PROC NEAR

                AND          AL, 0FH

                CMP          AL, 09

                JBE          NEXT

                ADD          AL, 07

NEXT:

                ADD          AL, 30H

                RETN

TETR_TO_HEX    ENDP
;-----

BYTE_TO_HEX    PROC NEAR
                ;БАЙТ В AL ПЕРЕВОДИТСЯ В ДВА СИМВОЛА ШЕСТН. ЧИСЛА В AX

```

```

        PUSH    CX
        MOV     AH, AL
        CALL    TETR_TO_HEX
        XCHG    AL, AH
        MOV     CL, 4
        SHR     AL, CL
        CALL    TETR_TO_HEX
    ;B AL - СТАРШАЯ ЦИФРА
        POP     CX
    ;B AH - МЛАДШАЯ
        RETN
BYTE_TO_HEX    ENDP
;-----
WRD_TO_HEX     PROC NEAR
    ;ПЕРЕВОД В 16 С/С 16-ТИ РАЗРЯДНОГО ЧИСЛА

    ;B AX - ЧИСЛО, DI - АДРЕС ПОСЛЕДНЕГО СИМВОЛА
    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
    RETN
WRD_TO_HEX     ENDP
;-----
CODE    ENDS

```

END

BEGIN