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

ОТЧЕТ

по практической работе №1

по дисциплине «Операционные системы»

Тема: Исследование структур загрузочных модулей

Студентка гр. 8381 _____ Муковский Д.В.

Преподаватель _____ Губкин А.Ф.

Санкт-Петербург

2020

Цель работы.

Исследование различий в структурах исходных текстов модулей .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.

Необходимые сведения для составления программы.

Тип IBM PC хранится в байте по адресу 0F000:0FFFE, в предпоследнем байте ROM BIOS. Соответствие кода и типа компьютера представлено в таблице 1:

Таблица 1 – Идентификация типа компьютера

Код	Тип компьютера
FF	Оригинальный IBM PC
FE	XT, Portable PC
FD	PCjr
FC	AT
FB	XT с памятью 640 К на мат. плате
FA	PS/2 модель 25 или 30
F9	Convertible PC
F8	PS/2 модели 55SX, 70, 80
9A	Compaq XT, Compaq Plus
30	Sperry PC
2D	Compaq PC, Compaq Deskpro

Для определения версии MS DOS следует воспользоваться функцией 30H прерывания 21H. Входным параметром является номер функции в AH:

```
MOV AH, 30h
INT 21h
```

Выходными параметрами являются:

AL – номер основной версии. Если 0, то <2.0;

AH – номер модификации;

BH – серийный номер OEM (Original Equipment Manufacturer);

BL:CX – 24-битовый серийный номер пользователя.

Постановка задачи.

Требуется реализовать текст исходного .COM модуля, который определяет тип PC и версию системы. Ассемблерная программа должна читать содержимое предпоследнего байта ROM BIOS, по таблице, сравнивая коды, определять тип PC и выводить строку с названием модели. Если код не совпадает ни с одним значением, то двоичный код переводиться в символьную строку, содержащую запись шестнадцатеричного числа и выводиться на экран в виде соответствующего сообщения. Затем определяется версия системы. Ассемблерная программа должна по значениям регистров AL и AH формировать текстовую строку в формате xx.yy, где xx - номер основной версии, а yy - номер модификации в десятичной системе счисления, формировать строки с серийным номером OEM (Original Equipment Manufacturer) и серийным номером пользователя. Полученные строки выводятся на экран.

Далее необходимо отладить полученный исходный модуль и получить «хороший» .COM модуль, а также необходимо построить «плохой» .EXE, полученный из исходного текста для .COM модуля.

Затем нужно написать текст «хорошего» .EXE модуля, который выполняет те же функции, что и модуль .COM, далее его построить, отладить и сравнить исходные тексты для .COM и .EXE модулей.

Выполнение работы.

Первым делом был написан исходный модуль .COM, который определяет тип PC и версию системы. Полученный модуль был скомпилирован, в результате чего был получен “плохой” .EXE модуль, а с помощью программы exe2bin получен “хороший” .COM модуль. После

линковки было выведено предупреждение, что сегмент стека отсутствует, что представлено на рис. 1.

```
Run File [LR1_COM.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
LINK : warning L4021: no stack segment
```

Рисунок 1 – Предупреждение об отсутствие стека

Результат выполнения «плохого» .EXE модуля представлен на рис. 2.

```
S:\>LR1_COM.exe
                                                                  θW⊡PC


θW⊡PC
           5 0
                                                        θW⊡PC
         0
                          θW⊡PC
           00 0000
         θW⊡PC
```

Рисунок 2 – Выполнение «плохого» .EXE модуля

Результат выполнения «хорошего» .COM модуля представлен на рис. 3.

```
S:\>LR1_COM.com
IBM PC TYPE: AT
MSDOS VERSION: 5.0
OEM NUMBER: 0
SERIAL NUMBER: 00 0000
```

Рисунок 3 - Выполнение «хорошего» .COM модуля

Далее был написан исходный .EXE модуль. В результате компилирования был получен «хороший» .EXE модуль. Результат его выполнения представлен на рис. 4.

```
S:\>lr1_exe.exe
IBM PC TYPE: AT
MSDOS VERSION: 5.0
OEM NUMBER: 0
SERIAL NUMBER: 00 0000
```

Рисунок 4 - Выполнение «хорошего» .EXE модуля

Отличия исходных текстов COM и EXE программ.

1. **Сколько сегментов должна содержать COM-программа?**

Один сегмент, в котором размещаются программные коды, данные и стек.

2. **EXE программа?**

Для программного кода, данных и стека предусмотрены отдельные сегменты.

3. **Какие директивы должны обязательно быть в тексте COM программы?**

Директива ORG 100h (смещение 100h), так как при загрузке COM-файла в память, DOS занимает первые 256 байт (100h) блоком данных PSP и располагает код программы только после этого блока.

Директива ASSUME, ставящая в соответствие адрес сегмента программы сегментам кода и данных. Ошибки программы без данной директивы представлены на рис. 5.

```
LR1_COM.asm(71): error A2062: Missing or unreachable CS
LR1_COM.asm(79): error A2062: Missing or unreachable CS
LR1_COM.asm(91): error A2062: Missing or unreachable CS
LR1_COM.asm(100): error A2062: Missing or unreachable CS
LR1_COM.asm(111): error A2062: Missing or unreachable CS
LR1_COM.asm(115): error A2062: Missing or unreachable CS
LR1_COM.asm(119): error A2062: Missing or unreachable CS
LR1_COM.asm(123): error A2062: Missing or unreachable CS
LR1_COM.asm(127): error A2062: Missing or unreachable CS
LR1_COM.asm(131): error A2062: Missing or unreachable CS
LR1_COM.asm(135): error A2062: Missing or unreachable CS
LR1_COM.asm(139): error A2062: Missing or unreachable CS
LR1_COM.asm(143): error A2062: Missing or unreachable CS
LR1_COM.asm(147): error A2062: Missing or unreachable CS
LR1_COM.asm(151): error A2062: Missing or unreachable CS
LR1_COM.asm(184): error A2062: Missing or unreachable CS
LR1_COM.asm(196): error A2062: Missing or unreachable CS
LR1_COM.asm(204): error A2062: Missing or unreachable CS

  49960 + 453205 Bytes symbol space free

      0 Warning Errors
     23 Severe  Errors
```

Рисунок 5 - .COM без директивы ASSUME

4. **Все ли форматы команд можно использовать в COM-программе?**

Нет, так как в отличие от EXE-программы, в которой существует таблица настроек, хранящая информацию о типе адресов и их местоположении в коде, COM-программа ею не располагает. Адреса сегментов определяются загрузчиком в момент запуска программы, а не в момент компиляции. Поэтому в COM-программах невозможно использовать команды вида: mov [регистр], seg [сегмент].

Отличия форматов файлов .COM и .EXE модулей

1. Какова структура файла .COM? С какого адреса располагается код?

Вид файла COM в шестнадцатеричном формате представлен на рис. 6.

Рисунок 6 - .COM модуль в шестнадцатеричном виде

Данный файл состоит из одного сегмента, размер файла не может превышать 64 Кб. Код начинается с адреса 0h, но при запуске модуля устанавливается смещение в 100h.

Сначала идут байты данных, на рис.6 первым выделено слово «PC». А байты, отвечающие за код, начинаются с номера BC и на приведенном рисунке выделена первая команда процедуры TETR_TO_HEX (см. Приложение А).

2. Какова структура файла «плохого» EXE? С какого адреса располагается код? Что располагается с адреса 0?

Вид «плохого» EXE файла в шестнадцатеричном виде представлен на рис. 7.

Рисунок 7 – «плохой» .EXE модуль в шестнадцатеричном виде

В «плохом» .EXE файле код располагается с адреса 300h. С адреса 0h располагается таблица настроек. Также 100h резервируется командой org 100h.

2. Какова структура файла «хорошего» EXE? Чем он отличается от файла «плохого» EXE?

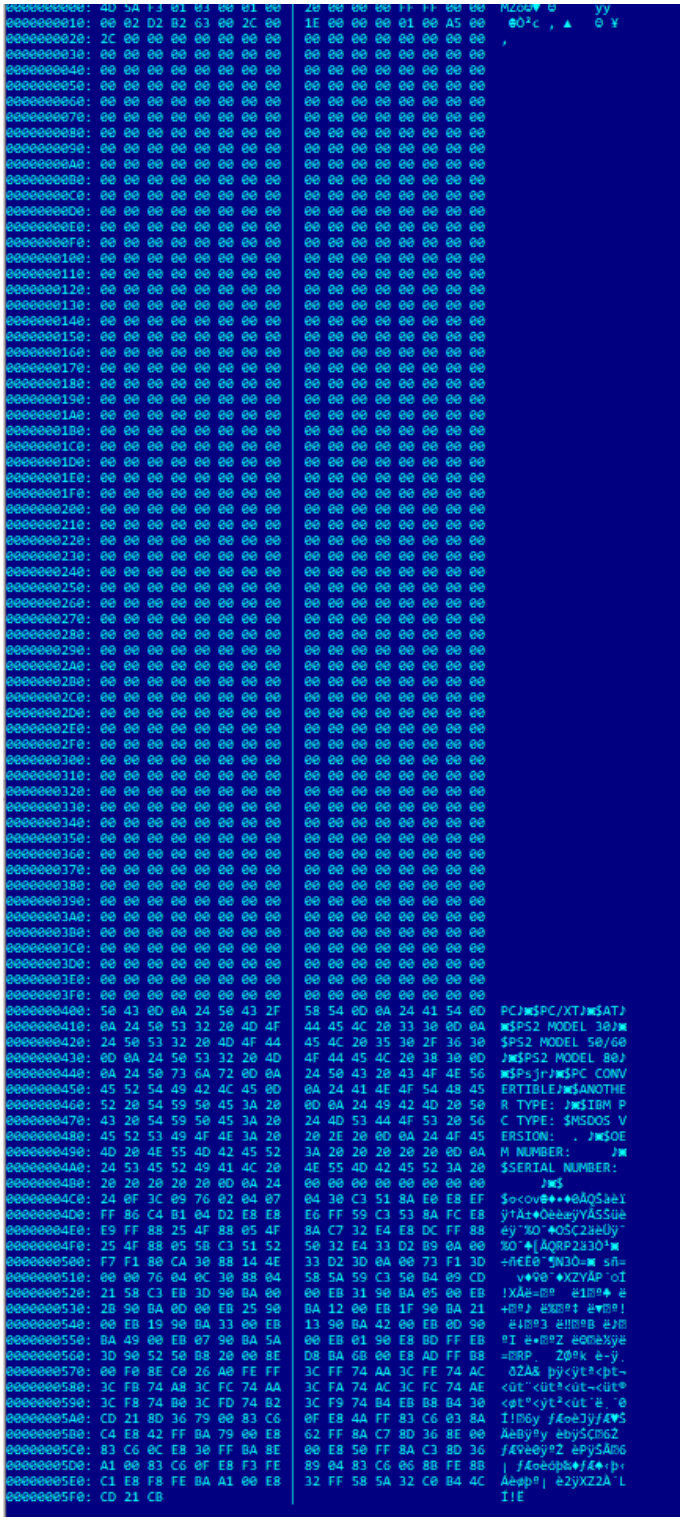

Рисунок 8 – «хороший» .EXE модуль в шестнадцатеричном виде

В «хорошем» EXE с нулевого адреса располагается таблица настроек. Также перед кодом выделена память под стек, в данном случае при размере стека 200h код располагается с адреса 400h. Еще одним отличием «хорошего» модуля является то, что в нем не резервируются дополнительно 100h, требуемые для PSP в .COM файле.

Загрузка COM модуля в основную память

1. Какой формат загрузки COM модуля? С какого адреса располагается код?

Загрузка файла .COM в отладчике Turbo Debugger представлен на рис. 9.

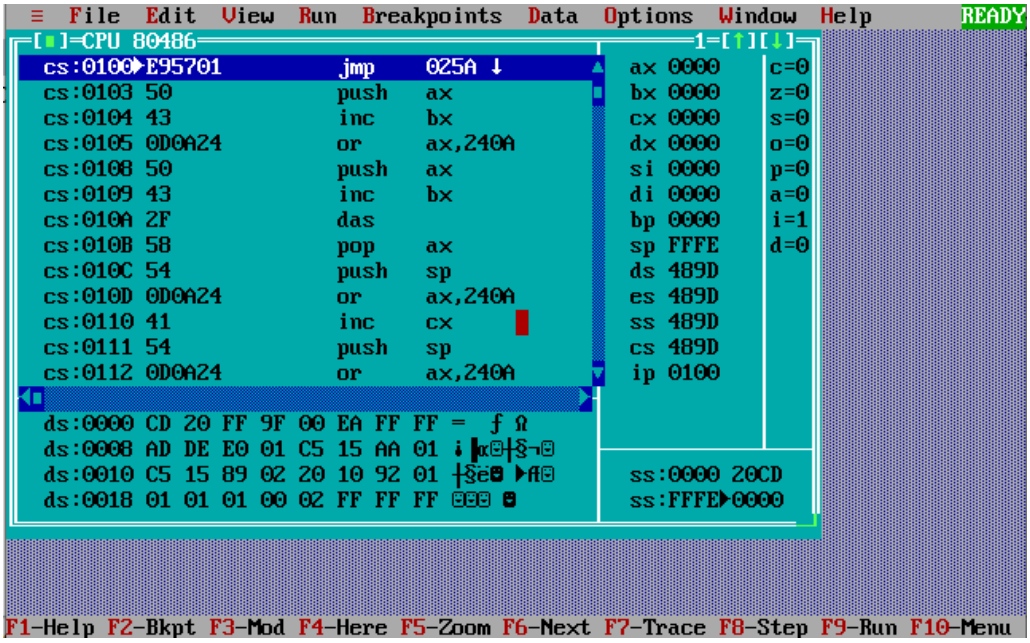

Рисунок 9 – Отладка .COM файла

После загрузки COM-программы в память сегментные регистры указывают на начало PSP. Код располагается с адреса 100h, IP = 0100h.

2. Что располагается с 0 адреса?

С нулевого адреса располагается адрес начала PSP.

3. Какие значения имеют сегментные регистры? На какие области памяти они указывают?

Сегментные регистры имеют значение 489Dh и указывают на PSP.

4. Как определяется стек? Какую область памяти он занимает? Какие адреса?

Стек определяется автоматически, указатель стека устанавливается на конец сегмента. Если для программы размер сегмента 64 кб достаточен, то DOS устанавливает в регистре SP-адрес конца сегмента – FFFEh, что можно увидеть на рис. 9. Адреса расположены в диапазоне от FFFEh до 0000h.

Загрузка «хорошего» EXE модуля в основную память

Загрузка файла .EXE в отладчике Turbo Debugger представлен на рис. 10.

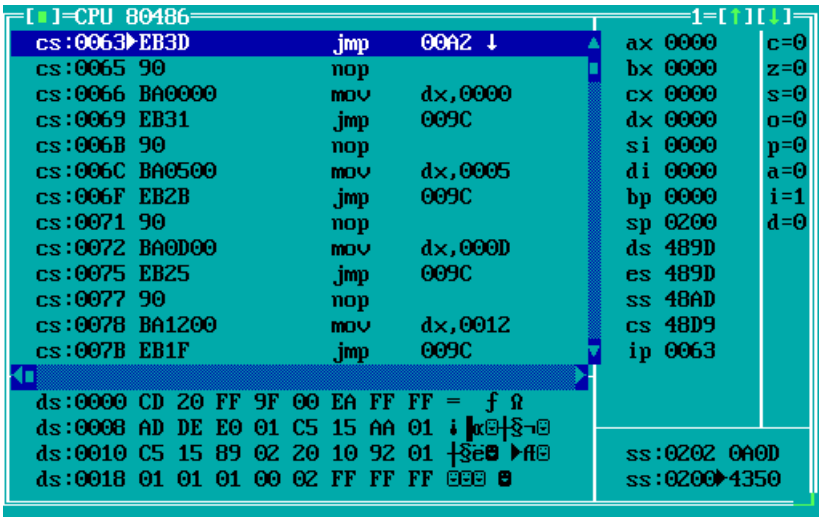

Рисунок 10 – Отладка .EXE файла

1. Как загружается «хороший» EXE? Какие значения имеют сегментные регистры?

Для PSP и программы выделяется блок памяти. После запуска программы DS и ES указывают на начало PSP (489Dh), CS – на начало сегмента команд (48D9), а SS – на начало сегмента стека (48AD). IP имеет ненулевое значение, так как в программе есть дополнительные процедуры, которые расположены до основной.

2. На что указывают регистры DS и ES?

Изначально регистры указывают на начало PSP. Поэтому в начале программы для корректной работы регистр DS инициализируется адресом начала сегмента данных.

3. Как определяется стек?

Стек определяется в исходном коде при помощи директивы. STACK, а при исполнении в регистр SS записывается адрес начала сегмента стека, а в указатель стека SP- указатель на конец стека.

4. Как определяется точка входа?

Смещение точки входа в программу загружается в указатель команд IP и определяется операндом директивы END, который называется точкой входа.

Вывод.

В ходе работы было проведено исследование различий в структурах исходных текстов модулей .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ LR1_COM.ASM

```
LAB SEGMENT
                    ASSUME CS:LAB, DS:LAB, ES:NOTHING, SS:NOTHING
                    ORG 100H
MAIN: JMP BEGIN

;данные
PC                          DB    'PC', 0DH, 0AH,'$'
PC_XT                   DB    'PC/XT', 0DH, 0AH,'$'
AT_                         DB    'AT', 0DH, 0AH, '$'
PS2_30                  DB    'PS2 MODEL 30', 0DH, 0AH, '$'
PS2_5060            DB   'PS2 MODEL 50/60', 0DH, 0AH, '$'
PS2_80                  DB    'PS2 MODEL 80', 0DH, 0AH, '$'
PCJR                    DB    'Psjr', 0DH, 0AH, '$'
PC_CONVERTIBLE      DB   'PC CONVERTIBLE', 0DH, 0AH, '$'
TYPE_ANOTHER DB 'ANOTHER TYPE: ',0DH, 0AH, '$'

IBM_PC_NAME             DB    'IBM PC TYPE: ', '$'
OS_NAME                     DB    'MSDOS VERSION:   . ', 0DH, 0AH,
'$'
OEM_NAME                DB    'OEM NUMBER:     ', 0DH, 0AH, '$'
SERIAL_NAME             DB    'SERIAL NUMBER:      ', 0DH, 0AH, '$'


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
```

```
;байт в al переводится в два символа шест. 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 цифра
                        RET
BYTE_TO_HEX         ENDP

;---------------------------------------------------------------------
-------------

WRD_TO_HEX          PROC NEAR
;перевод в 16 с/с 16 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
                        XOR  AH, AH
                        CALL BYTE_TO_HEX
                        MOV  [DI], AH
                        DEC  DI
                        MOV  [DI], AL
                        POP  BX
                        RET
WRD_TO_HEX          ENDP

;---------------------------------------------------------------------
-------------

BYTE_TO_DEC         PROC NEAR
```

```
;перевод в 10 с/с, si - адрес поля младшей цифры
                        PUSH CX
                        PUSH DX
                        PUSH AX
                        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  AX, 00H
                        JBE  END_L
                        OR AL, 30H
                        MOV  [SI], AL
END_L:
                        POP  AX
                        POP  DX
                        POP  CX
                        RET
BYTE_TO_DEC         ENDP

;-------------------------------------------------------------------
-------------

PRINT PROC NEAR
                        PUSH AX
                        MOV AH, 09H
                        INT 21H
                        POP AX
                        RET
PRINT ENDP

;-------------------------------------------------------------------
-------------
```

```
PC_WRITE:
                        MOV DX, OFFSET PC
                        JMP WRITE

PC_XT_WRITE:
                        MOV DX, OFFSET PC_XT
                        JMP WRITE

AT_WRITE:
                        MOV DX, OFFSET AT_
                        JMP WRITE

PS2_30_WRITE:
                        MOV DX, OFFSET PS2_30
                        JMP WRITE

PS2_5060_WRITE:
                        MOV DX, OFFSET PS2_5060
                        JMP WRITE

PS2_80_WRITE:
                        MOV DX, OFFSET PS2_80
                        JMP WRITE

PCJR_WRITE:
                        MOV DX, OFFSET PCJR
                        JMP WRITE

PC_CONVERTIBLE_WRITE:
                        MOV DX, OFFSET PC_CONVERTIBLE
                        JMP WRITE

PC_ANOTHER_WRITE:
                        MOV DX, OFFSET TYPE_ANOTHER
                        JMP WRITE

WRITE:
```

```
                        CALL PRINT
                        JMP OS

BEGIN:
                        PUSH DX
                        PUSH AX


                        MOV DX, OFFSET IBM_PC_NAME
                        CALL PRINT

                        MOV AX, 0F000H
                        MOV ES, AX
                        MOV AL, ES:[0FFFEH]

                        CMP AL, 0FFH
                        JE   PC_WRITE
                        CMP AL, 0FEH
                        JE PC_XT_WRITE
                        CMP AL, 0FBH
                        JE PC_XT_WRITE
                        CMP AL, 0FCH
                        JE AT_WRITE
                        CMP AL, 0FAH
                        JE PS2_30_WRITE
                        CMP AL, 0FCH
                        JE PS2_5060_WRITE
                        CMP AL, 0F8H
                        JE PS2_80_WRITE
                        CMP AL, 0FDH
                        JE PCJR_WRITE
                        CMP AL, 0F9H
                        JE PC_CONVERTIBLE_WRITE

                        JMP PC_ANOTHER_WRITE

OS:
                        MOV AH, 30H ;дает номер версии dos  al -
основная версия, ah - номер модификации
```

```
                    INT 21H
                    LEA       SI, OS_NAME
                    ADD       SI, 15
                    CALL BYTE_TO_DEC
                    ADD       SI, 3
                    MOV  AL, AH
                    CALL      BYTE_TO_DEC
                    MOV DX, OFFSET OS_NAME
                    CALL PRINT

OEM:
                    MOV  AL, BH ;bh - серийный номер
                    LEA  SI, OEM_NAME
                    ADD  SI, 12
                    CALL BYTE_TO_DEC
                    MOV  DX, OFFSET OEM_NAME
                    CALL PRINT

SERIAL:
                    MOV       AL, BL
                    LEA       SI, SERIAL_NAME
                    ADD       SI, 15
                    CALL BYTE_TO_HEX
                    MOV       [SI], AX
                    ADD       SI, 6
                    MOV       DI, SI
                    MOV  AX, CX
                    CALL WRD_TO_HEX
                    MOV  DX, OFFSET SERIAL_NAME
                    CALL PRINT

                    POP       AX
                    POP  DX
                    XOR       AL, AL
                    MOV  AH, 4CH
                    INT       21H
                    RET
LAB                 ENDS
                    END  MAIN
```

ПРИЛОЖЕНИЕ В
ИСХОДНЫЙ КОД ПРОГРАММЫ LR1_EXE.ASM

```
ASTACK SEGMENT STACK
                    DW 100h DUP(?)
ASTACK ENDS

;данные
DATA SEGMENT
                    PC              DB    'PC', 0DH, 0AH,'$'
                    PC_XT           DB    'PC/XT', 0DH, 0AH,'$'
                    AT_             DB    'AT', 0DH, 0AH, '$'
                    PS2_30          DB    'PS2 MODEL 30', 0DH, 0AH, '$'
                    PS2_5060  DB    'PS2 MODEL 50/60', 0DH, 0AH, '$'
                    PS2_80          DB    'PS2 MODEL 80', 0DH, 0AH, '$'
                    PCJR            DB    'Psjr', 0DH, 0AH, '$'
                    PC_CONVERTIBLE DB    'PC CONVERTIBLE', 0DH, 0AH,
'$'
                    TYPE_ANOTHER DB 'ANOTHER TYPE: ',0DH, 0AH, '$'
                    IBM_PC_NAME          DB    'IBM PC TYPE: ', '$'
                    OS_NAME              DB    'MSDOS VERSION:   . ',
0DH, 0AH, '$'
                    OEM_NAME        DB    'OEM NUMBER:     ', 0DH, 0AH,
'$'
                    SERIAL_NAME          DB    'SERIAL NUMBER:      ',
0DH, 0AH, '$'
DATA ENDS

CODE SEGMENT
                    ASSUME CS:CODE, DS:DATA, SS:ASTACK

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
;байт в al переводится в два символа шест. 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 цифра
                        RET
BYTE_TO_HEX         ENDP

;-------------------------------------------------------------------
-------------

WRD_TO_HEX          PROC NEAR
;перевод в 16 с/с 16 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
                        XOR  AH, AH
                        CALL BYTE_TO_HEX
                        MOV  [DI], AH
                        DEC  DI
                        MOV  [DI], AL
                        POP  BX
                        RET
WRD_TO_HEX          ENDP
```

```
;-------------------------------------------------------------------
-------------

BYTE_TO_DEC         PROC NEAR
;перевод в 10 с/с, si - адрес поля младшей цифры
                        PUSH CX
                        PUSH DX
                        PUSH AX
                        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  AX, 00H
                        JBE  END_L
                        OR AL, 30H
                        MOV  [SI], AL
END_L:
                        POP  AX
                        POP  DX
                        POP  CX
                        RET
BYTE_TO_DEC         ENDP

;-------------------------------------------------------------------
-------------

PRINT PROC NEAR
                        PUSH AX
                        MOV AH, 09H
                        INT 21H
                        POP AX
```

```
                        RET
PRINT ENDP

;-------------------------------------------------------------------------
-------------



MAIN PROC FAR
                    JMP BEGIN
                    PC_WRITE:
                        MOV DX, OFFSET PC
                        JMP WRITE

                    PC_XT_WRITE:
                        MOV DX, OFFSET PC_XT
                        JMP WRITE

                    AT_WRITE:
                        MOV DX, OFFSET AT_
                        JMP WRITE

                    PS2_30_WRITE:
                        MOV DX, OFFSET PS2_30
                        JMP WRITE

                    PS2_5060_WRITE:
                        MOV DX, OFFSET PS2_5060
                        JMP WRITE

                    PS2_80_WRITE:
                        MOV DX, OFFSET PS2_80
                        JMP WRITE

                    PCJR_WRITE:
                        MOV DX, OFFSET PCJR
                        JMP WRITE

                    PC_CONVERTIBLE_WRITE:
```

```
        MOV DX, OFFSET PC_CONVERTIBLE
        JMP WRITE

PC_ANOTHER_WRITE:
        MOV DX, OFFSET TYPE_ANOTHER
        JMP WRITE

WRITE:
        CALL PRINT
        JMP OS


BEGIN:
        PUSH DX
        PUSH AX

        MOV AX, DATA
        MOV DS, AX

        MOV DX, OFFSET IBM_PC_NAME
        CALL PRINT

        MOV AX, 0F000H
        MOV ES, AX
        MOV AL, ES:[0FFFEH]

        CMP AL, 0FFH
        JE   PC_WRITE
        CMP AL, 0FEH
        JE PC_XT_WRITE
        CMP AL, 0FBH
        JE PC_XT_WRITE
        CMP AL, 0FCH
        JE AT_WRITE
        CMP AL, 0FAH
        JE PS2_30_WRITE
        CMP AL, 0FCH
        JE PS2_5060_WRITE
        CMP AL, 0F8H
```

```
        JE PS2_80_WRITE
        CMP AL, 0FDH
        JE PCJR_WRITE
        CMP AL, 0F9H
        JE PC_CONVERTIBLE_WRITE

        JMP PC_ANOTHER_WRITE

    OS:
        MOV AH, 30H ;дает номер версии dos  al - основная версия, ah - номер модификации
        INT 21H
        LEA     SI, OS_NAME
        ADD     SI, 15
        CALL BYTE_TO_DEC
        ADD     SI, 3
        MOV  AL, AH
        CALL    BYTE_TO_DEC
        MOV DX, OFFSET OS_NAME
        CALL PRINT

    OEM:
        MOV  AL, BH ;bh - серийный номер
        LEA  SI, OEM_NAME
        ADD  SI, 12
        CALL BYTE_TO_DEC
        MOV  DX, OFFSET OEM_NAME
        CALL PRINT

    SERIAL:
        MOV     AL, BL
        LEA     SI, SERIAL_NAME
        ADD     SI, 15
        CALL BYTE_TO_HEX
        MOV     [SI], AX
        ADD     SI, 6
        MOV     DI, SI

        MOV  AX, CX
```

```
                    CALL  WRD_TO_HEX
                    MOV   DX, OFFSET SERIAL_NAME
                    CALL  PRINT

                    POP         AX
                    POP   DX
                    XOR         AL, AL
                    MOV   AH, 4CH
                    INT         21H
                    RET

MAIN                ENDP
CODE ENDS
                    END   MAIN
```