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

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Операционные системы»
Тема: Исследование структур загрузочных модулей

Студент гр. 8381 _____ Почаев Н.А.

Преподаватель _____ Ефремов М.А.

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.

Задание.

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

Таблица 1 – Соответствие типа и кода PC

Тип IBM PC	Код
PC	FF
PC/XT	FE, FB
AT	FC
PS2 модель 30	FA
PS2 модель 50 или 60	FC
PS2 модель 80	F8
PCjr	FD
PC Convertible	F9

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 символьную 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.

Выполнение работы производилось на базе операционной системы Windows 7 (32 bit) 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.

Был написан текст исходного .COM модуля, который определяет тип PC и информацию о системе. Полученный модуль был отлажен, и в результате были получены «плохой» .EXE модуль и, с помощью программы EXE2BIN,

«хороший» .COM модуль. Во время линковки было выведено предупреждение об отсутствии сегмента стека, представленное на рис. 1.

```
E:\lr1>masm COM.asm
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988.  All rights reserved.

Object filename [COM.OBJ]:
Source listing  [NUL.LST]:
Cross-reference [NUL.CRF]:

  49088 + 436651 Bytes symbol space free

       0 Warning Errors
       0 Severe  Errors

E:\lr1>link COM.OBJ

Microsoft (R) Overlay Linker  Version 3.64
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.

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

Рисунок 1 – Предупреждение во время линковки

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

```
E:\lr1>com.exe

   щ▬☺PC


щ▬☺PC
                5 0
                                                                  щ▬☺PC
          255
                                                щ▬☺PC
                00 0000
                  щ▬☺PC
```

Рисунок 2 – Вывод «плохого» .EXE модуля

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

```
E:\lr1>com.com
IBM PC type is: AT
MSDOS version is: 5.0
OEM number is255
Serial number is: 00 0000
```

Рисунок 3 – Вывод «хорошего» .COM модуля

Был написан текст исходного .EXE модуля, который выполняет те же функции, что и .COM модуль. В результате постройки и отладки был получен «хороший» .EXE модуль. Результат его выполнения представлен на рис. 4.

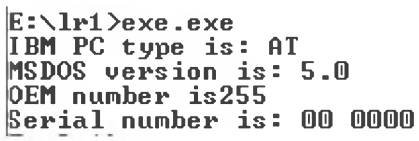


Рисунок 4 – Вывод «хорошего» .EXE модуля

Отличия исходных текстов COM и EXE программ.

1) Сколько сегментов должна содержать COM программа?

.COM - программы содержат только один сегмент. Модель памяти tiny - код, данные и стек объединены в один физический сегмент, максимальный размер которого не мог превышать 64 Кбайта без 256 байтов (последние требуются для создания префикса программного сегмента (PSP)).

2) EXE программа?

EXE-программа может содержать несколько сегментов. При использовании модели памяти small, в программе должен содержаться один сегмент данных и один сегмент кода в разных физических сегментах и каждый из них не может превосходить 64 Кбайта. При других моделях памяти (например large) есть возможность использования нескольких сегментов данных и (или) нескольких сегментов кода.

Помимо этого, в программе должен быть описан сегмент стека (до 64 Кбайт, содержит адреса возврата как для программы (для возврата в операционную систему), так и для вызовов подпрограмм (для возврата в главную программу), а также используется для передачи параметров в процедуры). Использует операционная система при обработке прерываний. Регистр сегмента стека (SS) адресует данный сегмент. Адрес текущей вершины стека задается регистрами SS:ESP.

3) Какие директивы должны обязательно быть в тексте COM-программы?

При запуске COM-программы первые 100h байт необходимо зарезервировать для префикса программного сегмента (PSP). Для этого используется директива ORG, которая устанавливает относительный адрес для начала выполнения программы: ORG 100h.

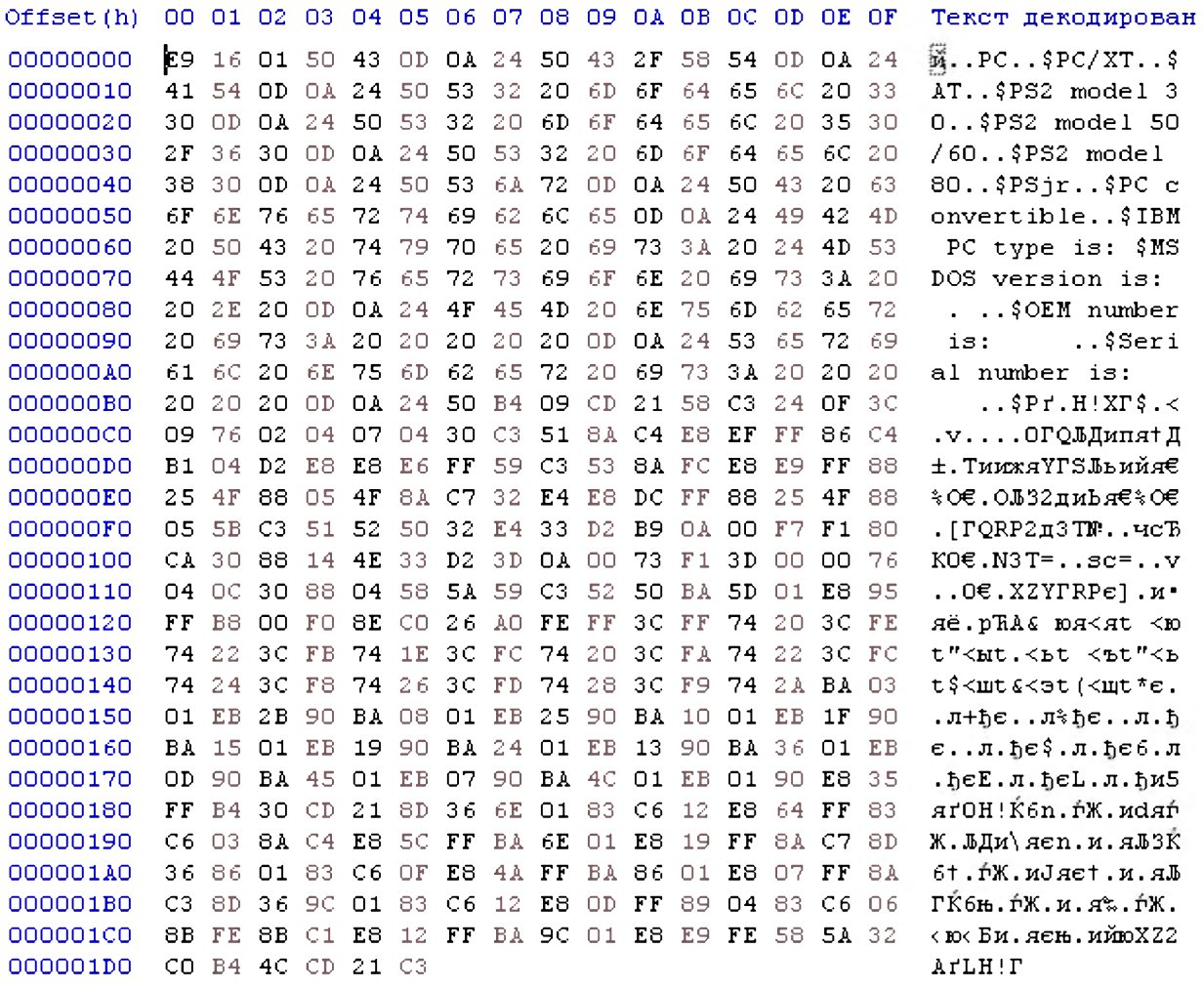
4) Все ли форматы команд можно использовать в COM-программе?

В .COM файле отсутствует таблица настройки с информацией о типе адресов и их местоположении в коде. Поэтому нельзя использовать команды, связанные с адресом сегмента, так как адрес сегмента неизвестен вплоть до загрузки этого сегмента в память. Загрузчику необходима информация о местоположении в файле загрузочного модуля полей адресов.

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. 5.



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Текст декодирован
00000000	E9	16	01	50	43	0D	0A	24	50	43	2F	58	54	0D	0A	24	й..PC..$PC/XT..$
00000010	41	54	0D	0A	24	50	53	32	20	6D	6F	64	65	6C	20	33	AT..$PS2 model 3
00000020	30	0D	0A	24	50	53	32	20	6D	6F	64	65	6C	20	35	30	0..$PS2 model 50
00000030	2F	36	30	0D	0A	24	50	53	32	20	6D	6F	64	65	6C	20	/60..$PS2 model
00000040	38	30	0D	0A	24	50	53	6A	72	0D	0A	24	50	43	20	63	80..$PSjr..$PC c
00000050	6F	6E	76	65	72	74	69	62	6C	65	0D	0A	24	49	42	4D	onvertible..$IBM
00000060	20	50	43	20	74	79	70	65	20	69	73	3A	20	24	4D	53	PC type is: $MS
00000070	44	4F	53	20	76	65	72	73	69	6F	6E	20	69	73	3A	20	DOS version is:
00000080	20	2E	20	0D	0A	24	4F	45	4D	20	6E	75	6D	62	65	72	. ..$OEM number
00000090	20	69	73	3A	20	20	20	20	20	0D	0A	24	53	65	72	69	is: ..$Seri
000000A0	61	6C	20	6E	75	6D	62	65	72	20	69	73	3A	20	20	20	al number is:
000000B0	20	20	20	0D	0A	24	50	B4	09	CD	21	58	C3	24	0F	3C	..$Pґ.H!XГ$.<
000000C0	09	76	02	04	07	04	30	C3	51	8A	C4	E8	EF	FF	86	C4	.v....0ГQЉДипяtД
000000D0	B1	04	D2	E8	E8	E6	FF	59	C3	53	8A	FC	E8	E9	FF	88	±.ТиижяYГSЉьийяЂ
000000E0	25	4F	88	05	4F	8A	C7	32	E4	E8	DC	FF	88	25	4F	88	%O€.OЉ32диЬяЂ%O€
000000F0	05	5B	C3	51	52	50	32	E4	33	D2	B9	0A	00	F7	F1	80	.[ГQRP2д3Т№..чсЂ
00000100	CA	30	88	14	4E	33	D2	3D	0A	00	73	F1	3D	00	00	76	К0€.N3Т=..sc=..v
00000110	04	0C	30	88	04	58	5A	59	C3	52	50	BA	5D	01	E8	95	..0€.XZYГRPє].и•
00000120	FF	B8	00	F0	8E	C0	26	A0	FE	FF	3C	FF	74	20	3C	FE	яё.рЋА& юя<яt <ю
00000130	74	22	3C	FB	74	1E	3C	FC	74	20	3C	FA	74	22	3C	FC	t"<ыt.<ьt <ъt"<ь
00000140	74	24	3C	F8	74	26	3C	FD	74	28	3C	F9	74	2A	BA	03	t$<шt&<эt(<щt*є.
00000150	01	EB	2B	90	BA	08	01	EB	25	90	BA	10	01	EB	1F	90	.л+ђє..л%ђє..л.ђ
00000160	BA	15	01	EB	19	90	BA	24	01	EB	13	90	BA	36	01	EB	є..л.ђє$.л.ђє6.л
00000170	0D	90	BA	45	01	EB	07	90	BA	4C	01	EB	01	90	E8	35	.ђєE.л.ђєL.л.ђи5
00000180	FF	B4	30	CD	21	8D	36	6E	01	83	C6	12	E8	64	FF	83	яґ0Н!Ќ6n.ѓЖ.идяѓ
00000190	C6	03	8A	C4	E8	5C	FF	BA	6E	01	E8	19	FF	8A	C7	8D	Ж.ЉДи\яєn.и.яЉЗЌ
000001A0	36	86	01	83	C6	0F	E8	4A	FF	BA	86	01	E8	07	FF	8A	6†.ѓЖ.иJяє†.и.яЉ
000001B0	C3	8D	36	9C	01	83	C6	12	E8	0D	FF	89	04	83	C6	06	ГЌ6њ.ѓЖ.и.я‰.ѓЖ.
000001C0	8B	FE	8B	C1	E8	12	FF	BA	9C	01	E8	E9	FE	58	5A	32	‹ю‹Би.яєњ.ийюXZ2
000001D0	C0	B4	4C	CD	21	C3											АґLH!Г

Рисунок 5 – Вид COM файла в шестнадцатеричном виде

COM-файл состоит из одного сегмента, а размер файла не превышает 64 КБ. Код располагается с нулевого адреса, что видно на рис. 5.

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

Вид «плохого» EXE файла в шестнадцатеричном формате представлен на рис. 6.

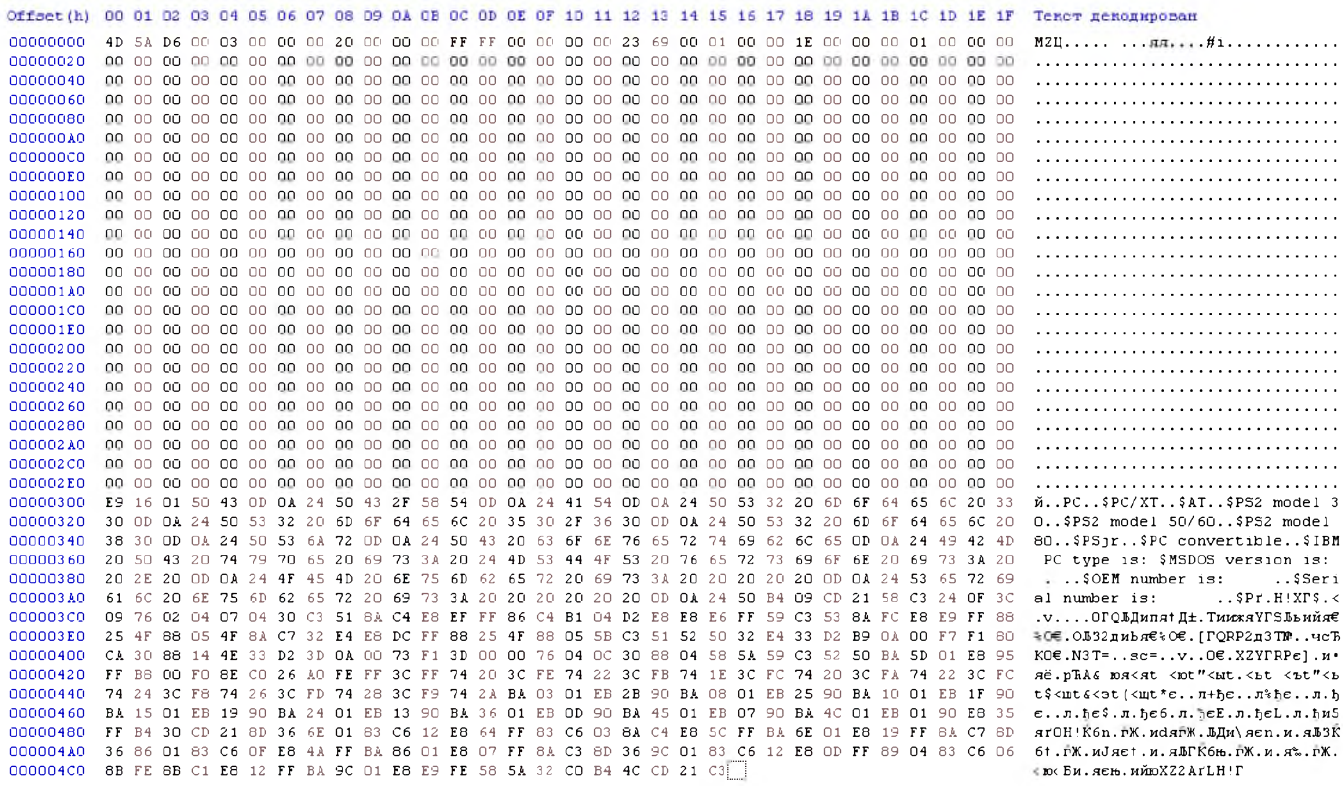


Рисунок 6 – Вид «плохого» EXE файла

В «плохом» EXE-файле код располагается с адреса 300h. С нулевого адреса располагается управляющая информация для загрузчика, образующая заголовок.

Также 100h резервируются командой ORG 100h (что вызовет отличие в структуре «плохого» и «хорошего» EXE).

3) Какова структура файла «хорошего» EXE? Чем он отличается от файла «плохого» EXE?

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

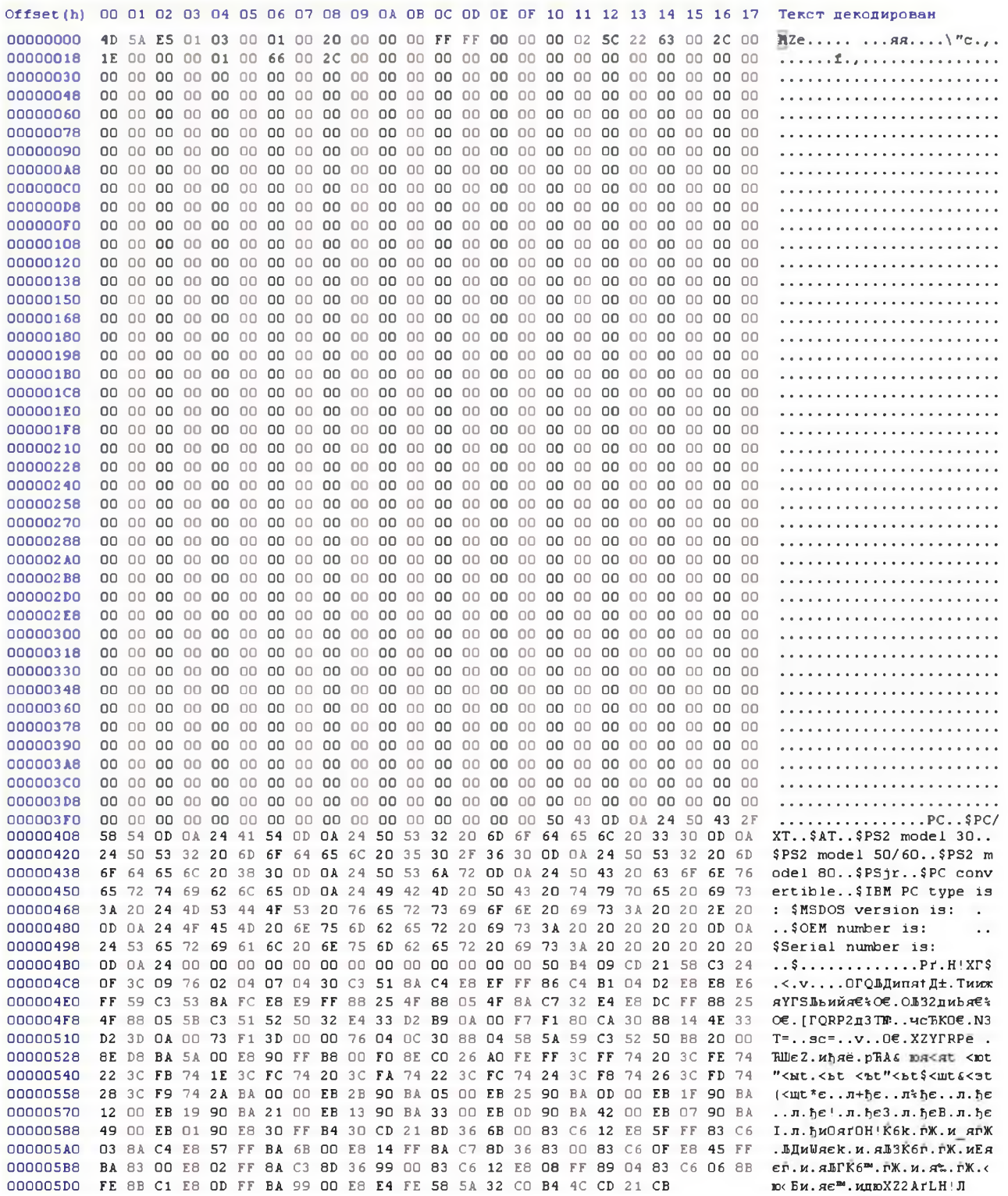


Рисунок 7 – Вид «хорошего» EXE файла

В «хорошем» EXE с нулевого адреса также располагается заголовок (например, 00-01 - 4D5A - сигнатура файла .EXE). Также перед кодом располагается сегмент стека. Так, при размере стека 200h код располагается с

адреса 400h. Если из исходного текста .EXE-программы убрать сегмент стека, то код будет располагаться с адреса 200h. Отличие от «плохого» EXE в том, что в «хорошем» не резервируется дополнительно 100h, которые в COM файле требовались для PSP, поэтому адреса начала кода отличаются на $100h + S$, где S – размер стека.

Загрузка COM модуля в основную память.

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

Запуск файла .COM в отладчике AFD.EXE представлен на рис. 8.

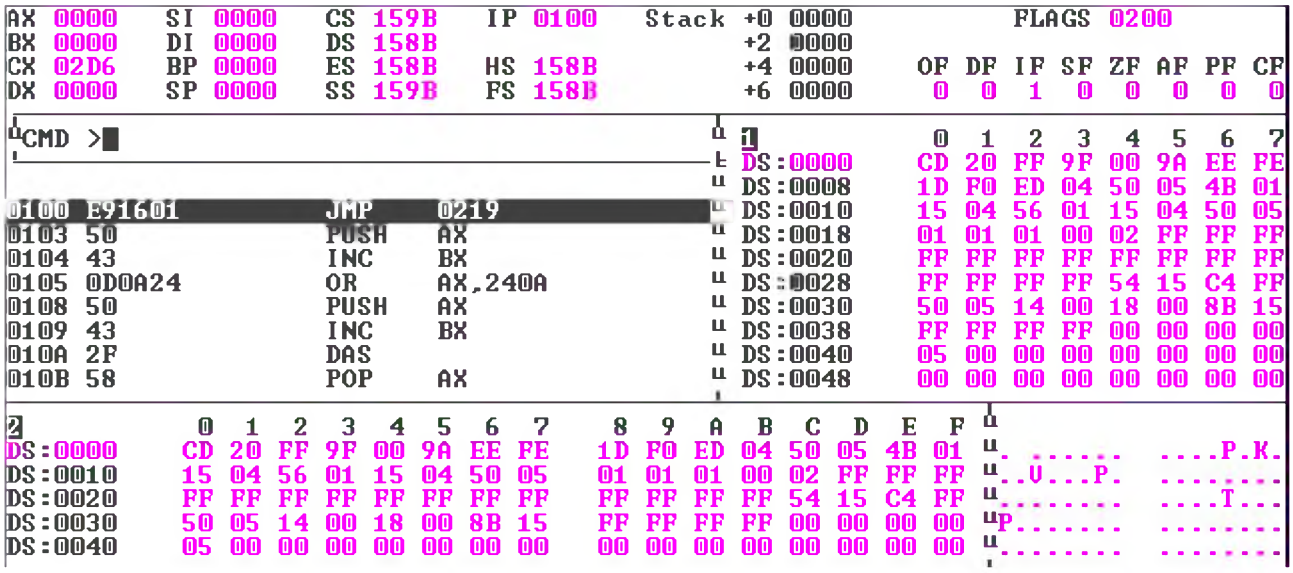


Рисунок 8 – Отладка файла .COM

Определяется сегментный адрес свободного участка ОП, в который можно загрузить программу. Создается блок памяти. В поля PSP заносятся значения. Загружается COM файл со смещением 100h. Сегментные регистры устанавливаются на адрес сегмента PSP, регистр SP указывает на конец сегмента, туда записывается 0000h. С ростом стека значение SP будет уменьшаться. Счетчик команд принимает значение 100h. Программа запускается.

2) Что располагается с адреса 0?

С нулевого адреса (0h) располагается сегмент PSP.

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

Сегментные регистры имеют значения 48DDh и указывают на PSP.

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

В COM модуле нельзя объявить стек, он создается автоматически. На рис. 8 видно, что SP имеет указывает на FFFEh. Стек занимает оставшуюся память (из 64 Кб), а его адреса изменяются от больших к меньшим, то есть от FFFEh к 0000h.

Загрузка «хорошего» EXE. модуля в основную память.

Запуск «хорошего» EXE модуля в отладчике AFD.EXE представлен на рис. 9.

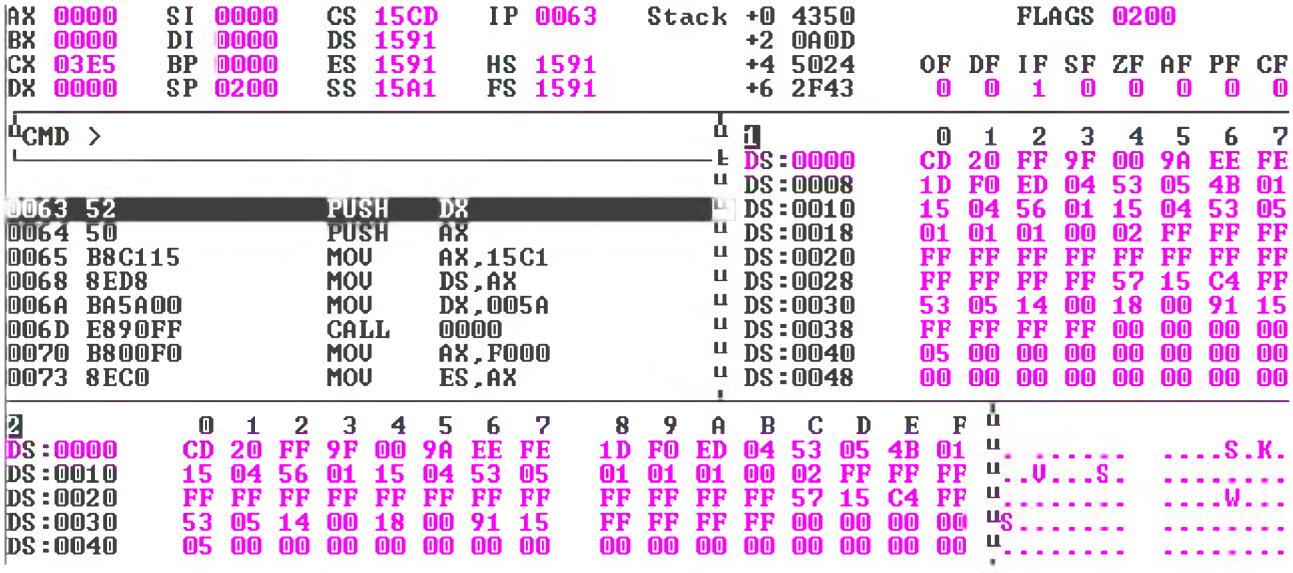


Рисунок 9 – Отладка «хорошего» EXE модуля

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

Определяется сегментный адрес свободного участка ОП, в который можно загрузить программу. Создается блок памяти для PSP и программы. После запуска программы DS и ES указывают на начало PSP (48DDh), CS – на начало сегмента команд (4919h), а SS – на начало сегмента стека (48EDh). IP имеет ненулевое значение, так как в программе есть дополнительные процедуры, расположенные до основной.

В PSP заносятся соответствующие значения. В рабочую область загрузчика считывается форматированная часть заголовка файла. Определяется смещение

начала загрузочного модуля в EXE файле. Вычисляется сегментный адрес (START_SEG) для загрузки. В память считывается загрузочный модуль. Таблица настройки порциями считывается в рабочую память. Для каждого элемента таблицы настройки к полю сегмента прибавляется сегментный адрес начального сегмента (в результате элемент таблицы указывает на нужное слово в памяти). Управление передается загруженной задаче по адресу из заголовка.

2) На что указывают регистры DS и ES?

Изначально регистры DS и ES указывают на начало сегмента PSP. Именно поэтому в начале программы для корректной работы с данными необходимо загрузить в DS адрес сегмента данных.

3) Как определяется стек?

Стек может быть объявлен при помощи директивы ASSUME, которая устанавливает сегментный регистр SS на начало сегмента стека, а также задает значение SP, указанное в заголовке. Также стек может быть объявлен с помощью директивы STACK. Если стек не объявлять, то он будет создан автоматически таким же образом, как в COM-модуле. Вид программы EXE модуля без объявленного стека после команды push в отладчике представлен на рис. 10.

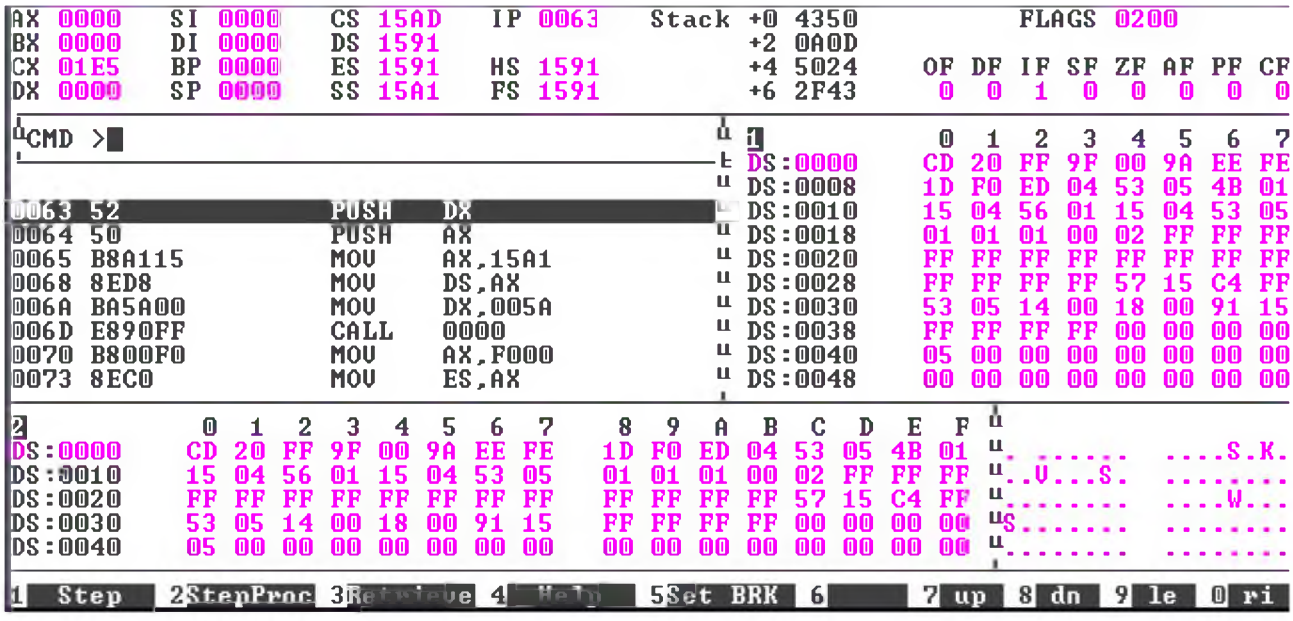


Рисунок 10 – Отладка EXE модуля без объявленного стека

4) Как определяется точка входа?

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

Операндом является функция или метка, с которой необходимо начать программу.

Выводы.

В ходе выполнения лабораторной работы были исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. COM.ASM

```
TESTPC SEGMENT
            ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
            ORG 100H    ; резервирование места для PSP
START:      JMP   BEGIN

; DATA SEGMENT
PC_TYPE                 db    'PC', 0DH, 0AH,'$'
PC_XT_TYPE        db    'PC/XT', 0dh, 0ah,'$'
AT_TYPE                 db    'AT', 0dh, 0ah, '$'
PS2_30_TYPE       db    'PS2 model 30', 0dh, 0ah, '$'
PS2_5060_TYPE     db    'PS2 model 50/60', 0dh, 0ah, '$'
PS2_80_TYPE       db    'PS2 model 80', 0dh, 0ah, '$'
PCjr_TYPE         db    'PSjr', 0dh, 0ah, '$'
PC_CONVERTIBLE    db    'PC convertible', 0dh, 0ah, '$'

IBM_PC_NAME       db    'IBM PC type is: ', '$'
OS_NAME                 db    'MSDOS version is:  . ', 0dh, 0ah, '$'
OEM_NAME          db    'OEM number is:     ', 0dh, 0ah, '$'
SERIAL_NAME       db    'Serial number is:      ', 0dh, 0ah, '$'
; DATA ENDS

; CODE SEGMENT


PRINT_STRING      PROC  near
            push  AX
            mov   AH, 09h
            int         21h
            pop   AX
            ret
PRINT_STRING      ENDP
;-------------------------------------------------------------------------------
--
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         al, ah
            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
            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
            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

BEGIN:
;PC INFO OUT
            push DX
            push AX

            mov DX, offset IBM_PC_NAME
            call PRINT_STRING

            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

PC_WRITE:
            mov DX, offset PC_TYPE
            jmp TYPE_WRITE

PC_XT_WRITE:
            mov DX, offset PC_XT_TYPE
            jmp TYPE_WRITE

AT_WRITE:
```

```
            mov DX, offset AT_TYPE
            jmp TYPE_WRITE

PS2_30_WRITE:
            mov DX, offset PS2_30_TYPE
            jmp TYPE_WRITE

PS2_5060_WRITE:
            mov DX, offset PS2_5060_TYPE
            jmp TYPE_WRITE

PS2_80_WRITE:
            mov DX, offset PS2_80_TYPE
            jmp TYPE_WRITE

PCjr_WRITE:
            mov DX, offset PCjr_TYPE
            jmp TYPE_WRITE

PC_CONVERTIBLE_WRITE:
            mov DX, offset PC_CONVERTIBLE
            jmp TYPE_WRITE

TYPE_WRITE:
            call PRINT_STRING

OS_INFO_GET:
            mov AH, 30h
            int 21h

OS_VERSION_SET:
            lea         SI, OS_NAME
            add         SI, 18
            call  BYTE_TO_DEC
            add         SI, 3
            mov   AL, AH
            call        BYTE_TO_DEC

OS_VERSION_WRITE:
            mov DX, offset OS_NAME
            call PRINT_STRING

OEM_SET:
            mov   AL, BH
            lea   SI, OEM_NAME
            add   SI, 15
            call  BYTE_TO_DEC
```

```
OEM_WRITE:
            mov   DX, offset OEM_NAME
            call  PRINT_STRING

SERIAL_SET:
            mov         AL, BL
            lea         SI, SERIAL_NAME
            add         SI, 18
            call  BYTE_TO_HEX
            mov         [SI], AX
            add         SI, 6
            mov         DI, SI
            mov   AX, CX
            call  WRD_TO_HEX

SERIAL_WRITE:
            mov   DX, offset SERIAL_NAME
            call  PRINT_STRING

ENDING:
            pop         AX
            pop   DX

            xor         AL, AL
            mov   AH, 4ch
            int         21h
            ret
TESTPC      ENDS
            END   START
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ. EXE.ASM

```
AStack      SEGMENT  STACK
                ; резервирование 2 байт по адресу 100h,
                ; DUP(?) - резервирование памяти без инициализации
                DW 100h DUP(?)
AStack      ENDS

DATA SEGMENT
      PC_TYPE                db    'PC', 0DH, 0AH,'$'
      PC_XT_TYPE        db    'PC/XT', 0dh, 0ah,'$'
      AT_TYPE                db    'AT', 0dh, 0ah, '$'
      PS2_30_TYPE       db    'PS2 model 30', 0dh, 0ah, '$'
      PS2_5060_TYPE     db    'PS2 model 50/60', 0dh, 0ah, '$'
      PS2_80_TYPE       db    'PS2 model 80', 0dh, 0ah, '$'
      PCjr_TYPE         db    'PSjr', 0dh, 0ah, '$'
      PC_CONVERTIBLE    db    'PC convertible', 0dh, 0ah, '$'

      IBM_PC_NAME       db    'IBM PC type is: ', '$'
      OS_NAME                db    'MSDOS version is:  . ', 0dh, 0ah, '$'
      OEM_NAME          db    'OEM number is:     ', 0dh, 0ah, '$'
      SERIAL_NAME       db    'Serial number is:      ', 0dh, 0ah, '$'
DATA ENDS

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

PRINT_STRING PROC near
            push  AX
            mov   AH, 09h
            int         21h
            pop   AX
            ret                     ; возврат из ближней процедуры
PRINT_STRING      ENDP
;-------------------------------------------------------------------------
--
TETR_TO_HEX       PROC  near
            and         al, 0fh
            cmp         al, 09
            jbe         NEXT  ; короткий переход, если первый операнд МЕНЬШЕ или
РАВЕН
                                ; второму операнду при выполнении операции
сравнения
                                ; с помощью команды CMP.
            add         al, 07
NEXT: add         al, 30h
```

```
            ret
TETR_TO_HEX       ENDP
;-------------------------------------------------------------------------------
--
BYTE_TO_HEX       PROC  near
            push  cx
            mov         al, ah
            call  TETR_TO_HEX
            xchg  al, ah      ; обмен значений двух операндов
            mov         cl, 4
            shr         al, cl      ; логический сдвиг вправо всех битов операнд
            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                ; декремент
            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
            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

Main  PROC  far
;PC INFO OUT
            push DX
            push AX

            mov ax, DATA
            mov   ds, ax

            mov DX, offset IBM_PC_NAME
            call PRINT_STRING

            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

PC_WRITE:
```

```
            mov DX, offset PC_TYPE
            jmp TYPE_WRITE

PC_XT_WRITE:
            mov DX, offset PC_XT_TYPE
            jmp TYPE_WRITE

AT_WRITE:
            mov DX, offset AT_TYPE
            jmp TYPE_WRITE

PS2_30_WRITE:
            mov DX, offset PS2_30_TYPE
            jmp TYPE_WRITE

PS2_5060_WRITE:
            mov DX, offset PS2_5060_TYPE
            jmp TYPE_WRITE

PS2_80_WRITE:
            mov DX, offset PS2_80_TYPE
            jmp TYPE_WRITE

PCjr_WRITE:
            mov DX, offset PCjr_TYPE
            jmp TYPE_WRITE

PC_CONVERTIBLE_WRITE:
            mov DX, offset PC_CONVERTIBLE
            jmp TYPE_WRITE

TYPE_WRITE:
            call PRINT_STRING

OS_INFO_GET:
            mov AH, 30h
            int 21h

OS_VERSION_SET:
            lea         SI, OS_NAME
            add         SI, 18
            call  BYTE_TO_DEC
            add         SI, 3
            mov   AL, AH
            call        BYTE_TO_DEC

OS_VERSION_WRITE:
```

```
            mov DX, offset OS_NAME
            call PRINT_STRING

OEM_SET:
            mov   AL, BH
            lea   SI, OEM_NAME
            add   SI, 15
            call  BYTE_TO_DEC

OEM_WRITE:
            mov   DX, offset OEM_NAME
            call  PRINT_STRING

SERIAL_SET:
            mov         AL, BL
            lea         SI, SERIAL_NAME
            add         SI, 18
            call  BYTE_TO_HEX
            mov         [SI], AX
            add         SI, 6
            mov         DI, SI
            mov   AX, CX
            call  WRD_TO_HEX

SERIAL_WRITE:
            mov   DX, offset SERIAL_NAME
            call  PRINT_STRING

ENDING:
            pop         AX
            pop   DX

            xor         AL, AL
            mov   AH, 4ch
            int         21h
            ret

Main  ENDP
CODE  ENDS
            END   Main
```