**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Операционные системы»**

Тема: Исследование структур загрузочных модулей

|  |  |  |
| --- | --- | --- |
| Студент гр. 0304 |  | Шквиря Е. В. |
| Преподаватель |  | Губкин А. Ф. |

Санкт-Петербург

2022

**Цель работы.**

Исследование различий в структурах исходных текстов модулей типов .СОМ и .ЕХЕ, структур файлов загрузочных модулей и способов их загрузки в основную память.

**Порядок выполнения работы.**

«Истина познается в сравнении», как говорили древние. К счастью, у нас есть возможность исследовать в одной системе два различных формата загрузочных модулей, сравнить их и лучше понять как система программирования и управляющая программа обращаются с ними. Система программирования включает компилятор с языка ассемблер (часто называется, просто, ассемблер), который изготавливает объектные модули. Компоновщик (Linker) по совокупности объектных модулей, изготавливает загрузочный модуль, а также, функция ядра – загрузчик, которая помещает программу в основную память и запускает на выполнение. Все эти компоненты согласованно работают для изготовления и выполнения загрузочных модулей разного типа. Для выполнения лабораторной работы сначала нужно изготовить загрузочные модули.

**Шаг 1**. Напишите текст исходного **.СОМ** модуля, который определяет тип РС и версию системы. Это довольно простая задача и для тех, кто уже имеет опыт программирования на ассемблере, это будет небольшой разминкой. Для тех, кто раньше не сталкивался с программированием на ассемблере, это неплохая задача для первого опыта.

За основу возьмите шаблон, приведенный в разделе «Основные сведения». Необходимые сведения о том, как извлечь требуемую информацию, представлены в следующем разделе.

Ассемблерная программа должна читать содержимое предпоследнего байта ROM BIOS, по таблице, сравнивая коды, определять тип РС 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 и серийным номером пользователя. Полученные строки выводятся на экран.

Отладьте полученный исходный модуль.

Результатом выполнения этого шага будет «хороший» **.СОМ** модуль, а также необходимо построить «плохой» **.ЕХЕ**, полученный из исходного текста для **.СОМ** модуля.

**Шаг 2**. Напишите текст исходного **.ЕХЕ** модуля, который выполняет те же функции, что и модуль в Шаге 1 и постройте и отладьте его. Таким образом, будет получен «хороший» **.ЕХЕ**.

**Шаг 3**. Сравните исходные тексты для **.СОМ** и **.ЕХЕ** модулей. Ответьте на контрольные вопросы «Отличия исходных текстов СОМ и ЕХЕ программ».

**Шаг 4**. Запустите FAR и откройте (F3/F4) файл загрузочного модуля **.СОМ** и файл «плохого» **.ЕХЕ** в шестнадцатеричном виде. Затем откройте (F3/F4) файл загрузочного модуля «хорошего» **.ЕХЕ** и сравните его с предыдущими файлами. Ответьте на контрольные вопросы «Отличия форматов файлов СОМ и ЕХЕ модулей».

**Шаг 5**. Откройте отладчик **TD.EXE** и загрузите **.СОМ**. Ответьте на контрольные вопросы «Загрузка СОМ модуля в основную память». Представьте в отчете план загрузки модуля .СОМ в основную память.

**Шаг 6**. Откройте отладчик **TD.EXE** и загрузите «хороший» **.ЕХЕ**. Ответьте на контрольные вопросы «Загрузка «хорошего» ЕХЕ модуля в основную память».

**Шаг 7**. Оформление отчета в соответствии с требованиями. В отчете необходимо привести скриншоты. Для файлов их вид в шестнадцатеричном виде, для загрузочных модулей – в отладчике.

**Необходимые сведения для составления программы**

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

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

**Контрольные вопросы по лабораторной работе №1**

**Отличия исходных текстов СОМ и ЕХЕ программ**

1. Сколько сегментов должна содержать COM-программа?
2. EXE-программа?
3. Какие директивы должны обязательно быть в тексте COM-программы?
4. Все ли форматы команд можно использовать в COM-программе?

**Отличия форматов файлов СОМ и ЕХЕ модулей**

1. Какова структура файла COM? С какого адреса располагается код?
2. Какова структура файла «плохого» EXE? С какого адреса располагается код? Что располагается с адреса 0?
3. Какова структура файла «хорошего» EXE? Чем он отличается от файла  
   «плохого» EXE?

**Загрузка СОМ модуля в основную память**

1. Какой формат загрузки модуля COM? С какого адреса располагается код?
2. Что располагается с адреса 0?
3. Какие значения имеют сегментные регистры? На какие области памяти они указывают?
4. Как определяется стек? Какую область памяти он занимает? Какие адреса?

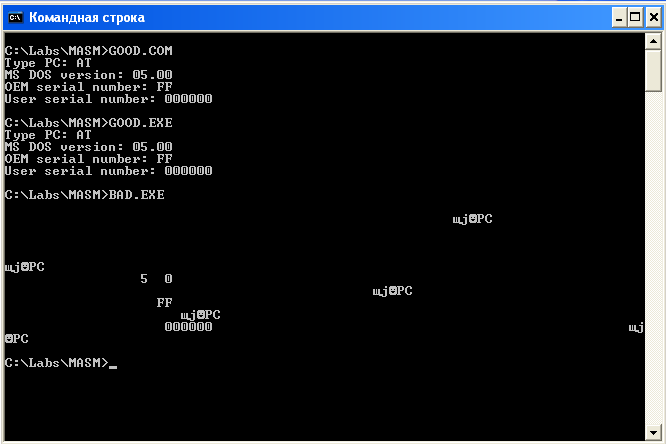
**Загрузка «хорошего» ЕХЕ модуля в основную память**

1. Как загружается «хороший» EXE? Какие значения имеют сегментные регистры?
2. На что указывают регистры DS и ES?
3. Как определяется стек?
4. Как определяется точка входа?

**Выполнение работы.**

1. Для начала был написан код исходного .COM модуля (см. Приложение А, файл *mod\_com.asm*), который выводит нужную информацию о типе IBM PC, о версии и модификации MS DOS и о серийном номере OEM и пользователя. Определение типа IBM PC происходит при помощи считывания информации по адресу 0F000:0FFFEh, а именно – считывание значения предпоследнего байта ROM BIOS. Далее это значение сравнивается по таблице (см. Необходимые сведения для составления программы), и в случае, если такой тип известен – он выводится, иначе выведется сообщение «Unknown type, HEX: XY», где вместо XY подставится 16-ричное представление числа. Версия MS DOS и серийные номера OEM и пользователя определяются посредством функции 30H прерывания 21H.
2. После написания .COM модуля он был скомпилирован при помощи ассемблера MASM в объектный модуль, после чего был создан «плохой» .EXE файл при помощи компоновщика (см. файл *BAD.EXE*). Далее из «плохого» .EXE файла при помощи программы EXE2BIN был получен «хороший» .COM файл (см. файл *GOOD.COM*).
3. Был написан код исходного .EXE модуля (см. Приложение А, файл *mod\_exe.asm*), который выполняет те же действия для вывода информации о типе IBM PC, о версии и модификации MS DOS и о серийном номере OEM и пользователя. После этого .EXE модуль был скомпилирован при помощи ассемблера MASM в объектный модуль, после чего был создан «хороший» .EXE файл при помощи компоновщика (см. файл *GOOD.EXE*).

После запуска всех трёх файлов можно увидеть следующий результат: «хороший» .COM модуль и «хороший» .EXE модуль корректно выводят информацию в читабельном виде, а «плохой» .EXE модуль – в нечитабельном. Результаты приведены на рис. 1

Рис. 1 – Запуск .EXE и .COM модулей

Контрольные вопросы:

**Отличия исходных текстов СОМ и ЕХЕ программ**

1. Любая COM-программа должна содержать только один сегмент, в котором находятся стек, данные и код.
2. Любая EXE-программа должна содержать не менее одного сегмента – сегменты стека, данных и кода.
3. Для любой COM-программы обязательными директивами являются:
4. ORG 100H – нужна для смещения адресации на 256 байт (поэтому 100H), так как там размещается префикс программного сегмента
5. ASSUME – нужен для начальной инициализации регистров CS, DS, ES, SS
6. SEGMENT – нужен для обозначения сегмента
7. Так как в COM-программах есть ограничения на размер файла (максимальный размер – 64 КБ), то программы могут работать только с 16-битными разрядами.

**Сравнение загрузочных модулей**

Рассмотрим 16-ричное представление загрузочного модуля **.COM** и модулей **.EXE**.

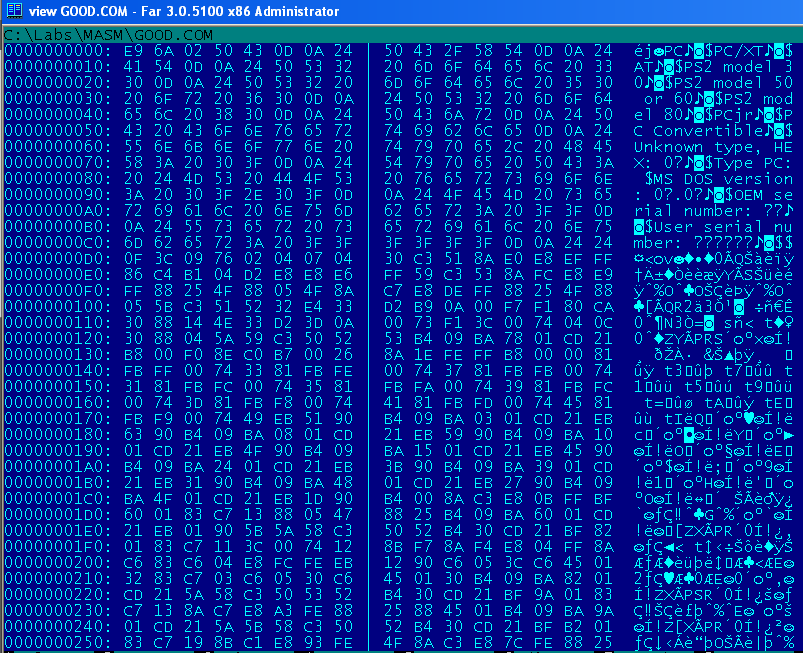


Рис. 2 – 16-ричный вид загрузочного модуля **.COM**

1. Из рисунка 2 можно увидеть, что данные и код располагаются, начиная с адреса 0H. Сам файл в себе содержит данные и команды (например, первая команда E9 отвечает за jmp, благодаря которому происходит переход к метке начала выполнения программы).

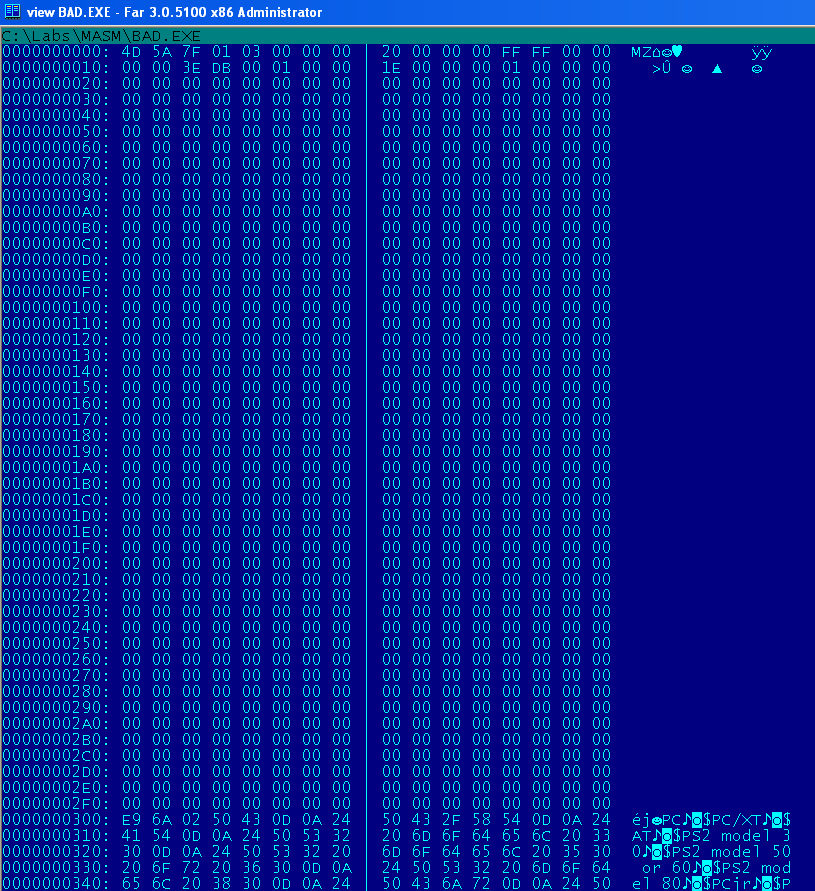


Рис. 3 – 16-ричный вид «плохого» загрузочного модуля **.EXE**

1. Из рисунка 3 можно увидеть, что по адресу 0H располагаются сначала заголовок загрузочного модуля, после него идёт таблица настройки, а уже по адресу 300H начинают идти данные и команды.

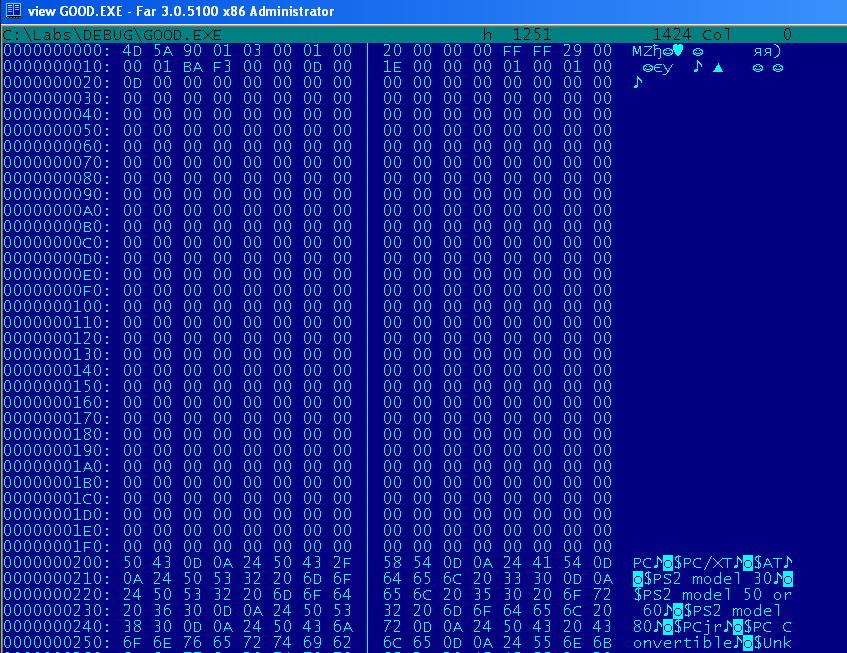
****

Рис. 4 – 16-ричный вид «хорошего» загрузочного модуля .EXE

1. Из рисунка 4 можно увидеть, что, как и в случае с «плохим» модулем .EXE, по адресу 0H расположен заголовок и далее идёт таблица настройки. Но отличием является то, что, во-первых, данные начинаются с адреса 200H, а во-вторых – данные расположены отдельно от команд.

**Загрузка .COM модуля в основную память**

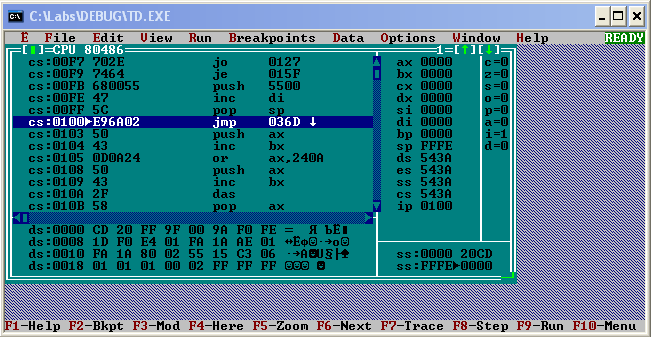
****

Рис. 5 – .COM модуль в отладчике TD.EXE (начиная с адреса 100H)

1. В момент начала работы .COM программы все сегментные регистры содержат адрес префикса программного сегмента (PSP) – 256-байтового блока, который резервируется операционной системой MS DOS перед программой в памяти. Как можно заметить на рисунке 5, код расположен в основной памяти начиная с адреса 100H. Значение регистра IP становится 100H.

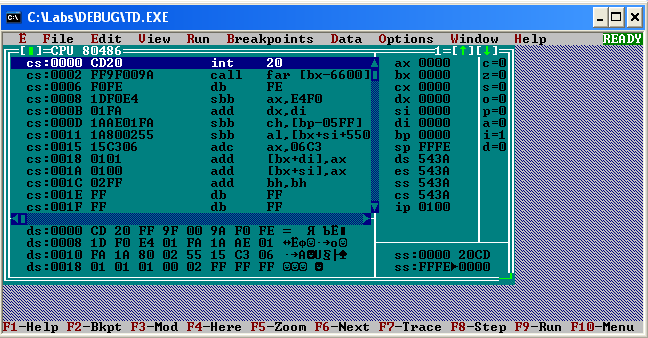


Рис. 6 – .COM модуль в отладчике TD.EXE (начиная с адреса 0H)

1. Начиная с адреса 0H расположен префикс сегмента программы (см. рис. 6), который строится самой системой во время запуска программы и не хранится непосредственно в файле загрузочного модуля. Данный префикс занимает 256 байт.
2. Как можно заметить по рисункам 5 и 6, все сегментные регистры указывают на одну и ту же область памяти. Сегментные регистры содержат адрес префикса программного сегмента, который резервируется DОS.
3. Стек расположен после всех данных кода, причём на его начало указывает регистр SP (см. рис. 6). Стек может расти от указанного в регистре адреса вплоть до 0H. Когда стек достигнет данных кода, то он начнёт их затирать, из-за чего программа станет некорректной.

**Загрузка «хорошего» .ЕХЕ модуля в основную память**

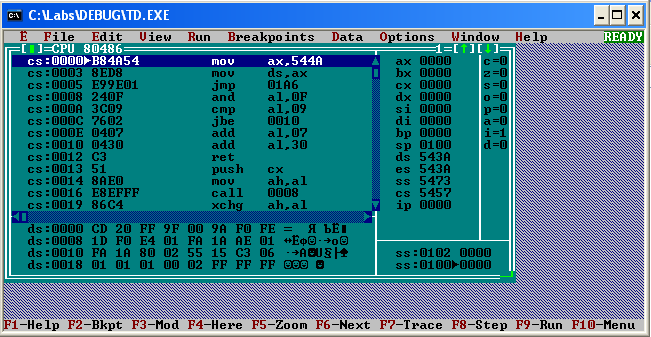
****

Рис. 7 - «Хороший» .EXE модуль в отладчике TD.EXE

1. Загрузка .EXE модуля схожа с загрузкой .COM модуля, но отличием является то, что сегментные регистры будут иметь разное значение, так как в .EXE модуле несколько сегментов. Это можно заметить по рисунку 7. Регистры DS и ES указывают на начало сегмента PSP, регистр SS указывает на начало стека, а CS – на начало сегмента кода.
2. Регистры DS и ES указывают на начало сегмента PSP.
3. В .EXE программах место под стек должно быть задано в программе с помощью указания сегмента стека. При старте программы регистр SP будет хранить в себе размер стека, а регистр SS – адрес начала стека.
4. Точка входа программы определяется при помощи метки, которая в дальнейшем также должна быть указана для директивы END.

**Вывод**

В ходе работы были исследованы файлы .COM модулей и .EXE модулей, выявлены их структуры, отличия друг от друга и способы загрузки данных в память.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

Файл: mod\_com.asm

; Шаблон текста программы на ассемблере для модуля типа .COM

TESTPC SEGMENT

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

ORG 100H

START: JMP BEGIN

; ДАННЫЕ

; Типы PC

TYPE\_PC\_PC db 'PC', 0Dh, 0Ah, '$'

TYPE\_PC\_PC\_XT db 'PC/XT', 0Dh, 0Ah, '$'

TYPE\_PC\_AT db 'AT', 0Dh, 0Ah, '$'

TYPE\_PC\_PS2\_30 db 'PS2 model 30', 0Dh, 0Ah, '$'

TYPE\_PC\_PS2\_50\_60 db 'PS2 model 50 or 60', 0Dh, 0Ah, '$'

TYPE\_PC\_PS2\_80 db 'PS2 model 80', 0Dh, 0Ah, '$'

TYPE\_PC\_PCjr db 'PCjr', 0Dh, 0Ah, '$'

TYPE\_PC\_PC\_CONVERTIBLE db 'PC Convertible', 0Dh, 0Ah, '$'

TYPE\_PC\_UNKNOWN db 'Unknown type, HEX: 0?', 0Dh, 0Ah, '$'

; Информация для вывода

TYPE\_PC\_LABLE db 'Type PC: $'

MS\_DOS\_VERSION db 'MS DOS version: 0?.0?', 0Dh, 0Ah, '$'

OEM\_SERIAL\_NUMBER db 'OEM serial number: ??', 0Dh, 0Ah, '$'

USER\_SERIAL\_NUMBER db 'User serial number: ??????', 0Dh, 0Ah, '$'

;ПРОЦЕДУРЫ

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

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 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

ret

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

ret

WRD\_TO\_HEX ENDP

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

BYTE\_TO\_DEC PROC near

; перевод в 10с/с, SI - адрес поля младшей цифры

push CX

push DX

xor AH,AH

xor DX,DX

mov CX,10

loop\_bd: div CX

or DL,30h

mov [SI],DL

dec SI

xor DX,DX

cmp AX,10

jae loop\_bd

cmp AL,00h

je end\_l

or AL,30h

mov [SI],AL

end\_l: pop DX

pop CX

ret

BYTE\_TO\_DEC ENDP

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

PRINT\_TYPE\_PC PROC near

; TODO Добавить добавление используемых регистров в стек, а в конце - достать их

push ax

push dx

push bx

mov ah, 09

mov dx, offset TYPE\_PC\_LABLE

int 21h

; получение информации о типе PC

mov ax, 0f000H

mov es, ax

mov bh, 0

mov bl, es:[0FFFEH]

mov ax, 0

; выявление типа PC

cmp bx, 0FFh

je PC

cmp bx, 0FEh

je PC\_XT

cmp bx, 0FBh

je PC\_XT

cmp bx, 0FCh

je AT

cmp bx, 0FAh

je PS2\_30

cmp bx, 0FCh

je PS2\_50\_60

cmp bx, 0F8h

je PS2\_80

cmp bx, 0FDh

je PCjr

cmp bx, 0F9h

je PC\_CONVERTIBLE

jmp PC\_UNKNOWN

; вывод информации

PC:

mov ah, 09

mov dx, offset TYPE\_PC\_PC

int 21h

jmp END\_COMP\_TYPE\_PC

PC\_XT:

mov ah, 09

mov dx, offset TYPE\_PC\_PC\_XT

int 21h

jmp END\_COMP\_TYPE\_PC

AT:

mov ah, 09

mov dx, offset TYPE\_PC\_AT

int 21h

jmp END\_COMP\_TYPE\_PC

PS2\_30:

mov ah, 09

mov dx, offset TYPE\_PC\_PS2\_30

int 21h

jmp END\_COMP\_TYPE\_PC

PS2\_50\_60:

mov ah, 09

mov dx, offset TYPE\_PC\_PS2\_50\_60

int 21h

jmp END\_COMP\_TYPE\_PC

PS2\_80:

mov ah, 09

mov dx, offset TYPE\_PC\_PS2\_80

int 21h

jmp END\_COMP\_TYPE\_PC

PCjr:

mov ah, 09

mov dx, offset TYPE\_PC\_PCjr

int 21h

jmp END\_COMP\_TYPE\_PC

PC\_CONVERTIBLE:

mov ah, 09

mov dx, offset TYPE\_PC\_PC\_CONVERTIBLE

int 21h

jmp END\_COMP\_TYPE\_PC

PC\_UNKNOWN:

mov ah, 00h

mov al, bl

call BYTE\_TO\_HEX

mov di, offset TYPE\_PC\_UNKNOWN

add di, 19

mov [di], al

inc di

mov [di], ah

mov ah, 09

mov dx, offset TYPE\_PC\_UNKNOWN

int 21h

jmp END\_COMP\_TYPE\_PC

END\_COMP\_TYPE\_PC:

pop bx

pop dx

pop ax

ret

PRINT\_TYPE\_PC ENDP

PRINT\_MSDOS\_VERS PROC near

push ax

push dx

mov AH,30h

int 21h

mov di, offset MS\_DOS\_VERSION

add di, 17

cmp al, 0

je LESS\_2\_VERS

mov si, di

mov dh, ah

call BYTE\_TO\_DEC

mov al, dh

add si, 4

call BYTE\_TO\_DEC

jmp PRINT\_VERS

LESS\_2\_VERS:

mov byte ptr [di], '<'

mov byte ptr [di+1], '2'

add di, 3

mov byte ptr [di], '0'

mov byte ptr [di+1], '0'

PRINT\_VERS:

mov ah, 09

mov dx, offset MS\_DOS\_VERSION

int 21h

pop dx

pop ax

ret

PRINT\_MSDOS\_VERS ENDP

PRINT\_OEM\_SERIAL\_NUMBER PROC near

push ax

push bx

push dx

mov AH,30h

int 21h

mov di, offset OEM\_SERIAL\_NUMBER

add di, 19

mov al, bh

call BYTE\_TO\_HEX

mov [di], ah

mov [di+1], al

mov ah, 09

mov dx, offset OEM\_SERIAL\_NUMBER

int 21h

pop dx

pop bx

pop ax

ret

PRINT\_OEM\_SERIAL\_NUMBER ENDP

PRINT\_USER\_SERIAL\_NUMBER PROC near

push ax

push dx

mov AH,30h

int 21h

mov di, offset USER\_SERIAL\_NUMBER

add di, 25

mov ax, cx

call WRD\_TO\_HEX

dec di

mov al, bl

call BYTE\_TO\_HEX

mov [di], ah

mov [di-1], al

mov ah, 09

mov dx, offset USER\_SERIAL\_NUMBER

int 21h

pop dx

pop ax

ret

PRINT\_USER\_SERIAL\_NUMBER ENDP

BEGIN:

call PRINT\_TYPE\_PC

call PRINT\_MSDOS\_VERS

call PRINT\_OEM\_SERIAL\_NUMBER

call PRINT\_USER\_SERIAL\_NUMBER

EXIT:

; Выход в DOS

xor AL,AL

mov AH,4Ch

int 21H

TESTPC ENDS

END START ;конец модуля, START - точка входа

Файл: mod\_exe.asm

TESTPC\_data segment

; ДАННЫЕ

; Типы PC

TYPE\_PC\_PC db 'PC', 0Dh, 0Ah, '$'

TYPE\_PC\_PC\_XT db 'PC/XT', 0Dh, 0Ah, '$'

TYPE\_PC\_AT db 'AT', 0Dh, 0Ah, '$'

TYPE\_PC\_PS2\_30 db 'PS2 model 30', 0Dh, 0Ah, '$'

TYPE\_PC\_PS2\_50\_60 db 'PS2 model 50 or 60', 0Dh, 0Ah, '$'

TYPE\_PC\_PS2\_80 db 'PS2 model 80', 0Dh, 0Ah, '$'

TYPE\_PC\_PCjr db 'PCjr', 0Dh, 0Ah, '$'

TYPE\_PC\_PC\_CONVERTIBLE db 'PC Convertible', 0Dh, 0Ah, '$'

TYPE\_PC\_UNKNOWN db 'Unknown type, HEX: 0?', 0Dh, 0Ah, '$'

; Информация для вывода

TYPE\_PC\_LABLE db 'Type PC: $'

MS\_DOS\_VERSION db 'MS DOS version: 0?.0?', 0Dh, 0Ah, '$'

OEM\_SERIAL\_NUMBER db 'OEM serial number: ??', 0Dh, 0Ah, '$'

USER\_SERIAL\_NUMBER db 'User serial number: ??????', 0Dh, 0Ah, '$'

TESTPC\_data ends

TESTPC\_code segment

assume cs: TESTPC\_code

start:

mov ax, testpc\_data

mov ds, ax

assume ds: TESTPC\_data

jmp BEGIN

;ПРОЦЕДУРЫ

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

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 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

ret

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

ret

WRD\_TO\_HEX ENDP

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

BYTE\_TO\_DEC PROC near

; перевод в 10с/с, SI - адрес поля младшей цифры

push CX

push DX

xor AH,AH

xor DX,DX

mov CX,10

loop\_bd: div CX

or DL,30h

mov [SI],DL

dec SI

xor DX,DX

cmp AX,10

jae loop\_bd

cmp AL,00h

je end\_l

or AL,30h

mov [SI],AL

end\_l: pop DX

pop CX

ret

BYTE\_TO\_DEC ENDP

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

PRINT\_TYPE\_PC PROC near

; TODO Добавить добавление используемых регистров в стек, а в конце - достать их

push ax

push dx

push bx

mov ah, 09

mov dx, offset TYPE\_PC\_LABLE

int 21h

; получение информации о типе PC

mov ax, 0f000H

mov es, ax

mov bh, 0

mov bl, es:[0FFFEH]

mov ax, 0

; выявление типа PC

cmp bx, 0FFh

je PC

cmp bx, 0FEh

je PC\_XT

cmp bx, 0FBh

je PC\_XT

cmp bx, 0FCh

je AT

cmp bx, 0FAh

je PS2\_30

cmp bx, 0FCh

je PS2\_50\_60

cmp bx, 0F8h

je PS2\_80

cmp bx, 0FDh

je PCjr

cmp bx, 0F9h

je PC\_CONVERTIBLE

jmp PC\_UNKNOWN

; вывод информации

PC:

mov ah, 09

mov dx, offset TYPE\_PC\_PC

int 21h

jmp END\_COMP\_TYPE\_PC

PC\_XT:

mov ah, 09

mov dx, offset TYPE\_PC\_PC\_XT

int 21h

jmp END\_COMP\_TYPE\_PC

AT:

mov ah, 09

mov dx, offset TYPE\_PC\_AT

int 21h

jmp END\_COMP\_TYPE\_PC

PS2\_30:

mov ah, 09

mov dx, offset TYPE\_PC\_PS2\_30

int 21h

jmp END\_COMP\_TYPE\_PC

PS2\_50\_60:

mov ah, 09

mov dx, offset TYPE\_PC\_PS2\_50\_60

int 21h

jmp END\_COMP\_TYPE\_PC

PS2\_80:

mov ah, 09

mov dx, offset TYPE\_PC\_PS2\_80

int 21h

jmp END\_COMP\_TYPE\_PC

PCjr:

mov ah, 09

mov dx, offset TYPE\_PC\_PCjr

int 21h

jmp END\_COMP\_TYPE\_PC

PC\_CONVERTIBLE:

mov ah, 09

mov dx, offset TYPE\_PC\_PC\_CONVERTIBLE

int 21h

jmp END\_COMP\_TYPE\_PC

PC\_UNKNOWN:

mov ah, 00h

mov al, bl

call BYTE\_TO\_HEX

mov di, offset TYPE\_PC\_UNKNOWN

add di, 19

mov [di], al

inc di

mov [di], ah

mov ah, 09

mov dx, offset TYPE\_PC\_UNKNOWN

int 21h

jmp END\_COMP\_TYPE\_PC

END\_COMP\_TYPE\_PC:

pop bx

pop dx

pop ax

ret

PRINT\_TYPE\_PC ENDP

PRINT\_MSDOS\_VERS PROC near

push ax

push dx

mov AH,30h

int 21h

mov di, offset MS\_DOS\_VERSION

add di, 17

cmp al, 0

je LESS\_2\_VERS

mov si, di

mov dh, ah

call BYTE\_TO\_DEC

mov al, dh

add si, 4

call BYTE\_TO\_DEC

jmp PRINT\_VERS

LESS\_2\_VERS:

mov byte ptr [di], '<'

mov byte ptr [di+1], '2'

add di, 3

mov byte ptr [di], '0'

mov byte ptr [di+1], '0'

PRINT\_VERS:

mov ah, 09

mov dx, offset MS\_DOS\_VERSION

int 21h

pop dx

pop ax

ret

PRINT\_MSDOS\_VERS ENDP

PRINT\_OEM\_SERIAL\_NUMBER PROC near

push ax

push bx

push dx

mov AH,30h

int 21h

mov di, offset OEM\_SERIAL\_NUMBER

add di, 19

mov al, bh

call BYTE\_TO\_HEX

mov [di], ah

mov [di+1], al

mov ah, 09

mov dx, offset OEM\_SERIAL\_NUMBER

int 21h

pop dx

pop bx

pop ax

ret

PRINT\_OEM\_SERIAL\_NUMBER ENDP

PRINT\_USER\_SERIAL\_NUMBER PROC near

push ax

push dx

mov AH,30h

int 21h

mov di, offset USER\_SERIAL\_NUMBER

add di, 25

mov ax, cx

call WRD\_TO\_HEX

dec di

mov al, bl

call BYTE\_TO\_HEX

mov [di], ah

mov [di-1], al

mov ah, 09

mov dx, offset USER\_SERIAL\_NUMBER

int 21h

pop dx

pop ax

ret

PRINT\_USER\_SERIAL\_NUMBER ENDP

BEGIN:

call PRINT\_TYPE\_PC

call PRINT\_MSDOS\_VERS

call PRINT\_OEM\_SERIAL\_NUMBER

call PRINT\_USER\_SERIAL\_NUMBER

EXIT:

; Выход в DOS

xor AL,AL

mov AH,4Ch

int 21H

TESTPC\_code ends

TESTPC\_stack segment stack

db 256 dup (0)

TESTPC\_stack ends

end start