

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

Студентка гр. 0382

Деткова А.С.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

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

Задание.

1. Написать текст исходного .COM модуля, который определяет тип РС и версию системы. Отладить полученный исходный модуль. Результатом выполнения этого шага будет «хороший» .COM модуль, а также необходимо построить «плохой» .EXE, полученный из исходного текста для .COM.
2. Написать текст исходного .EXE модуля, который выполняет те же функции, что и модуль в шаге 1 и отладить его. Таким образом, будет получен «хороший» .EXE.
3. Сравнить исходные тексты для .COM и .EXE модулей. Ответить на вопросы «Отличия исходных текстов COM и EXE программ».
4. Запустить FAR и открыть файл загрузочного модуля .COM и файл «плохого» .EXE в шестнадцатеричном виде. Затем открыть файл загрузочного модуля «хорошего» .EXE и сравнить его с предыдущими файлами. Ответить на контрольные вопросы «Отличия форматов файлов COM и EXE модулей».
5. Открыть отладчик TD.EXE и загрузить СО. Ответить на контрольные вопросы «Загрузка COM модуля в основную память». Представить в отчете план загрузки модуля .COM в основную память.
6. Открыть отладчик TD.EXE и загрузить «хороший» .EXE. Ответить на контрольные вопросы «Загрузка «хорошего» EXE в основную память».

7. Оформить отчет в соответствии с требованиями. Привести скриншоты. Для файлов их вид в шестнадцатеричном виде, для загрузочных модулей — в отладчике.

Выполнение работы.

В ходе работы была разработана программа good_com.asm, которая дает («хороший» com-модуль) good_com.com. Данная программа, выводит всю требуемую информацию, в ней использован шаблон из методических указаний. Все функции разобраны и понятны по принципу работы.



```
C:\>good_com.com
IBM PC type - AT.
MS DOS version - 5.0.
OEM number - 255.
User number - 000000h.
```

Рисунок 1: Результат запуска "хорошего" com-модуля

Был создан «плохой» exe-модуль путем линковки good_com.asm без флага -t.

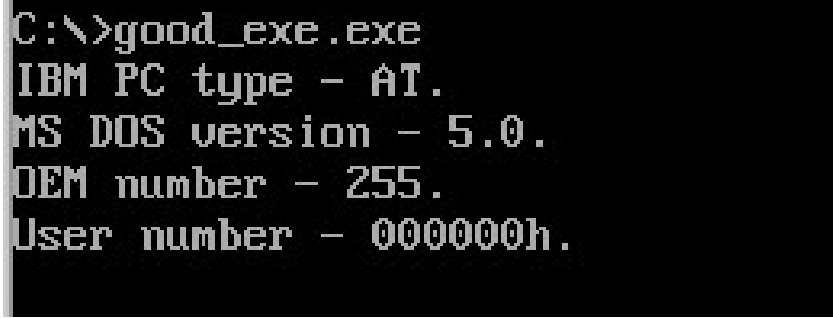


```
C:\>good_com.exe

                    IBM PC type - PC.
                    IBM PC type - PC.
                    5 0
                    255 IBM PC type - PC.
IBM PC type000000
```

Рисунок 2: Результат запуска "плохого" exe-модуля.

Была разработана программа good_exe.asm, которая выполняет те же действия, но она дает «хороший» exe-модуль.



```
C:\>good_exe.exe
IBM PC type - AT.
MS DOS version - 5.0.
OEM number - 255.
User number - 0000000h.
```

Рисунок 3: Результат запуска "хорошего" ехе-модуля

Ответы на контрольные вопросы.

Отличия исходных текстов COM и EXE программ:

1. Сколько сегментов должна содержать COM-программа?

COM-программа содержит один сегмент — сегмент кода. Данные располагаются непосредственно в коде. Стек генерируется автоматически.

2. EXE-программа?

EXE-программа может содержать несколько сегментов (сегмент данных, стека, кода).

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

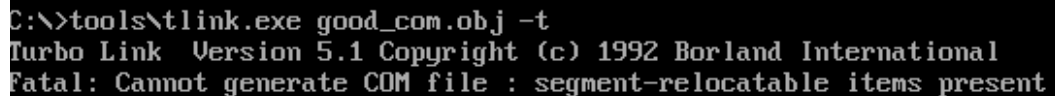
В тексте COM-программы обязательно должна быть директива ASSUME, чтобы соотнести сегменты и сегментные регистры (CS:MainSeg, DS:MainSeg, ES:NOTHING, SS:NOTHING). А также обязательна директива ORG 100H, тк в начале программы идет 256-байтовый блок PSP, загрузчику нужно показать с какого адреса начинается код программы.

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

Т.к. в COM-программе все сегментные регистры определяются в момент запуска программы, а не в момент компиляции (ассемблирования), то невозможно использование, например, таких конструкций:

```
mov ax, DATA
```

```
mov ax, CODE
```



```
C:\>tools\tlink.exe good_com.obj -t
Turbo Link Version 5.1 Copyright (c) 1992 Borland International
Fatal: Cannot generate COM file : segment-relocatable items present
```

Рисунок 4: Ошибка при выполнении неподдерживаемой форматом .COM команды

Отличия форматов файлов COM и EXE модулей:

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

COM файл (рисунок 5) состоит из единственного сегмента кода. Когда программа начинает работать все сегментные регистры указывают на PSP. Код располагается с адреса 0H. При запуске программы в начало будет добавлен сегмент PSP размером 100H (256 байт), поэтому директивой ORG 100H устанавливается смещение на 256 байт, т. е. к любому адресу будет прибавляться смещение 100H.

0000000000:	E9 1A 01 49 42 4D 20 50	43 20 74 79 70 65 20 2D	é→IBM PC type -
0000000010:	20 50 43 2E 0D 0A 24 49	42 4D 20 50 43 20 74 79	PC. IBM PC ty
0000000020:	70 65 20 2D 20 50 43 2F	58 54 2E 0D 0A 24 49 42	pe - PC/XT. IBM
0000000030:	4D 20 50 43 20 74 79 70	65 20 2D 20 41 54 2E 0D	M PC type - AT. I
0000000040:	0A 24 49 42 4D 20 50 43	20 74 79 70 65 20 2D 20	IBM PC type -
0000000050:	50 53 32 20 6D 6F 64 65	6C 20 33 30 2E 0D 0A 24	PS2 model 30. I
0000000060:	49 42 4D 20 50 43 20 74	79 70 65 20 2D 20 50 53	IBM PC type - PS
0000000070:	32 20 6D 6F 64 65 6C 20	35 30 20 6F 72 20 36 30	2 model 50 or 60
0000000080:	2E 0D 0A 24 49 42 4D 20	50 43 20 74 79 70 65 20	. IBM PC type
0000000090:	2D 20 50 53 32 20 6D 6F	64 65 6C 20 38 30 2E 0D	- PS2 model 80. I
00000000A0:	0A 24 49 42 4D 20 50 43	20 74 79 70 65 20 2D 20	IBM PC type -
00000000B0:	50 43 6A 72 2E 0D 0A 24	49 42 4D 20 50 43 20 74	PCjr. IBM PC t
00000000C0:	79 70 65 20 2D 20 50 43	20 43 6F 6E 76 65 72 74	ype - PC Convert
00000000D0:	69 62 6C 65 2E 0D 0A 24	4D 53 20 44 4F 53 20 76	ible. MS DOS v
00000000E0:	65 72 73 69 6F 6E 20 2D	20 20 2E 20 2E 0D 0A 24	ersion - . I
00000000F0:	4F 45 4D 20 6E 75 6D 62	65 72 20 2D 20 20 20 20	OEM number -
0000000100:	2E 0D 0A 24 55 73 65 72	20 6E 75 6D 62 65 72 20	. User number
0000000110:	2D 20 20 20 20 20 20 20	68 2E 0D 0A 24 E8 11 01	- h. I
0000000120:	32 C0 B4 4C CD 21 B4 09	CD 21 C3 51 52 50 32 E4	2À`LÍ!`oÍ!AQRp2ä
0000000130:	33 D2 B9 0A 00 F7 F1 80	CA 30 88 14 4E 33 D2 3D	3D` ÷ñÊ0`JN3D=
0000000140:	0A 00 73 F1 3C 00 74 04	0C 30 88 04 58 5A 59 C3	sñ< t`90`XYZYÄ
0000000150:	24 0F 3C 09 76 02 04 07	04 30 C3 51 8A E0 E8 EF	\$o<ov0`♦0ÄQŠaëi
0000000160:	FF 86 C4 B1 04 D2 E8 E8	E6 FF 59 C3 53 50 8A FC	ÿtÄ±0èèæÿYÄSPŠü
0000000170:	E8 E8 FF 88 25 4F 88 05	4F 8A C7 E8 DD FF 88 25	èèÿ`%0`*0ŠÇèÿÿ`%
0000000180:	4F 88 05 58 5B C3 B8 00	F0 8E C0 26 A0 FE FF 3C	0`*X[Ä, ðŽA& bÿ<
0000000190:	FF 74 20 3C FE 74 22 3C	FB 74 1E 3C FC 74 20 3C	ÿt <þt"<üt▲<üt <
00000001A0:	FA 74 22 3C FC 74 24 3C	F8 74 26 3C FD 74 28 3C	üt"<üt\$<øt&<ÿt(<
00000001B0:	F9 74 2A BA 03 01 E8 2B	90 BA 17 01 E8 25 90 BA	üt*`♥0ë+00\$0ë%00
00000001C0:	2E 01 E8 1F 90 BA 42 01	EB 19 90 BA 60 01 E8 13	.0ëv00B0ë↓00`0ë!!
00000001D0:	90 BA 84 01 E8 0D 90 BA	A2 01 E8 07 90 BA B8 01	00,,0ë,00000ë,00,0
00000001E0:	EB 01 90 E8 40 FF C3 B4	30 CD 21 BE E9 01 E8 3A	ë00ë@ÿÄ`0Í!%é0è:
00000001F0:	FF 8A C4 83 C6 03 E8 32	FF BA D8 01 E8 27 FF C3	ÿŠÄfÆ♥è2ÿ00è`ÿÄ
0000000200:	B4 30 CD 21 8A C7 BE FF	01 E8 1F FF BA F0 01 E8	`0Í!ŠÇÿ0èÿÿ0ð0è
0000000210:	14 FF C3 B4 30 CD 21 8B	C1 BF 17 02 E8 4D FF 8A	ÿÿÄ`0Í!<Ä;±0èMÿŠ
0000000220:	C3 E8 37 FF 4F 88 25 4F	88 05 BA 04 02 E8 F6 FE	Äè7ÿ0`%0`*0♦0èöþ
0000000230:	C3 E8 52 FF E8 B0 FF E8	C6 FF E8 D6 FF C3	ÄèRÿè°ÿèÆÿèöÿÄ

Рисунок 5: Структура «хорошего» COM-файла

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

В «плохом» EXE-файле данные и код располагаются в одном сегменте. Код располагается с адреса 300H. С адреса 0H располагается relocation table (таблица настроек — необходима для загрузки программы).

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

В хорошем EXE-файле (рисунок 8 и 9) данные, стек и код разделены по отдельным сегментам. Вначале файла расположен заголовок и relocation table (вся информация, необходимая для запуска программы). Далее располагается сегмент стека (с 300H по 400H — 256 байт). Далее находится сегмент данных и кода.

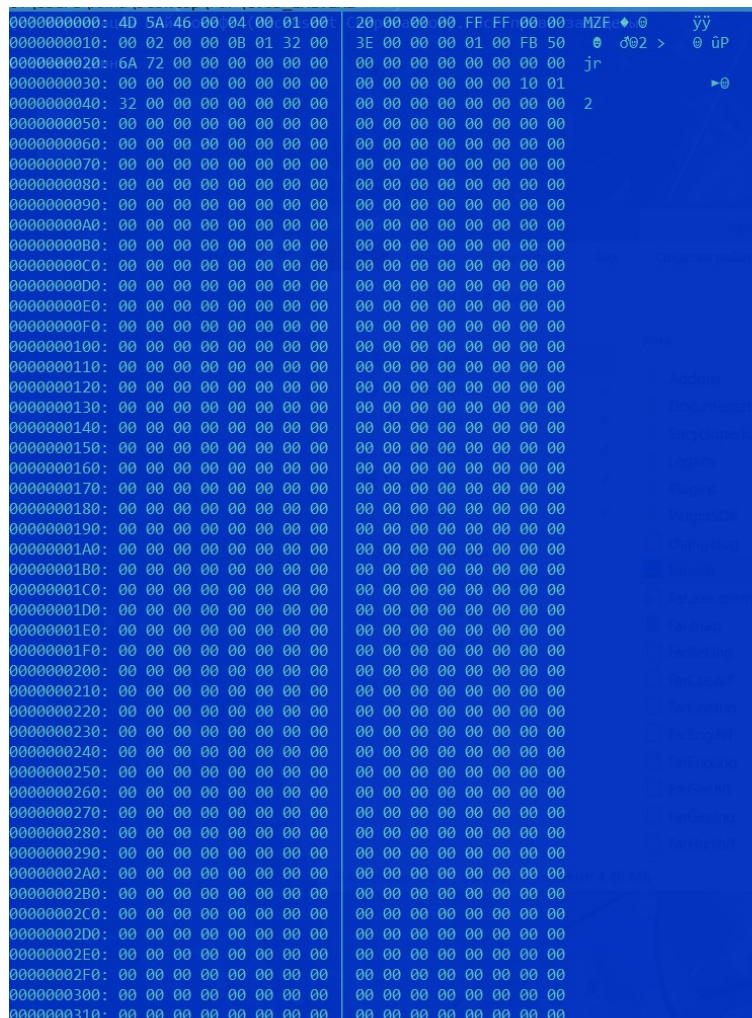


Рисунок 8: Структура "хорошего" EXE-модуля (ч.1)

«Хороший» EXE-файл в 16-ичном виде на рисунке 8 и 9.

Код расположен с адреса CS:0100 = 48DD:0100 (рисунок 10).

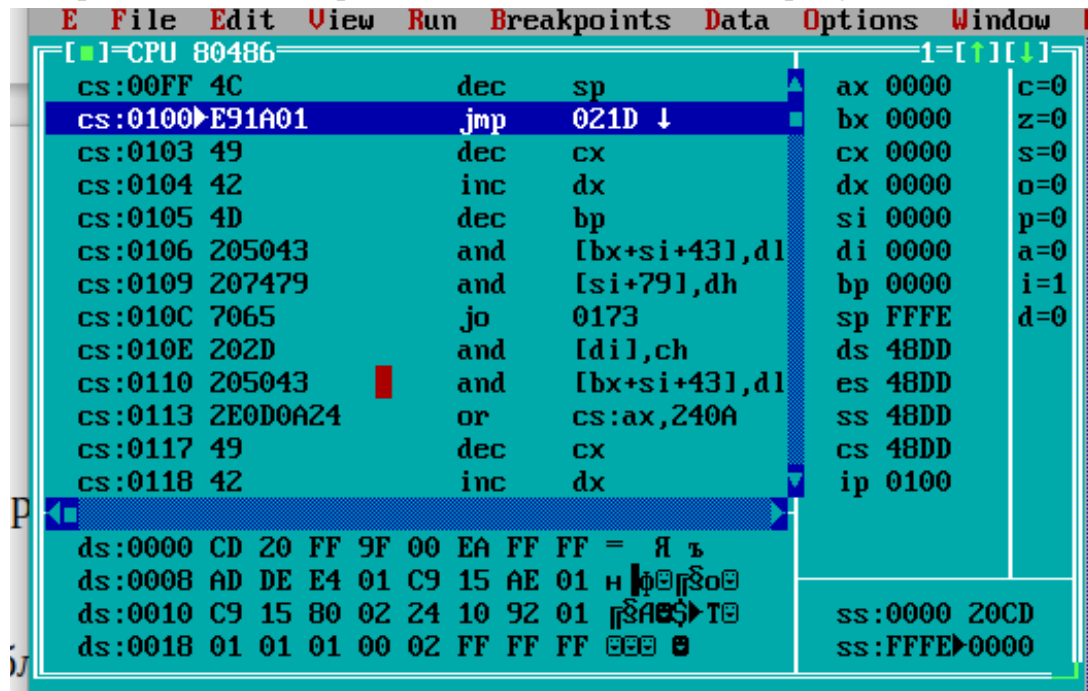


Рисунок 10: COM-модуль в начале загрузки в память

2. Что располагается с адреса 0?

С адреса 0000H располагается сегмент PSP.

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

Все сегментные регистры равны, имеют значение 48DD и указывают на начало PSP (рис. 8).

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

Под стек отведена вся область памяти, которая выделена для программы (можно убедиться в этом, потому что DS:0000 = CD 20 = 20CD = SS:0000). SP = FFFE — указывает на конец стека (последний адрес кратный двум). SS = 48DD — начало стека, начало сегмента. Стек занимает адреса SS:0000 — SS:FFFE (рис. 8).

Загрузка EXE модуля в основную память:

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

При запуске EXE-программы системным загрузчиком выполняются следующие действия:

Определяется сегментный адрес свободного участка памяти, размер которого достаточен для размещения программы. Создается блок памяти для PSP и программы. Генерируется блок PSP. Считывается в память загрузочный модуль в соответствии с информацией в заголовке файла.

DS = ES = 48DD (указывают на PSP), SS = 48ED (указывает на верхушку стека), CS = 491F (указывает на начало кода). См. Рисунок 11.

2. На что указывают регистры DS и ES?

Указывают на PSP.

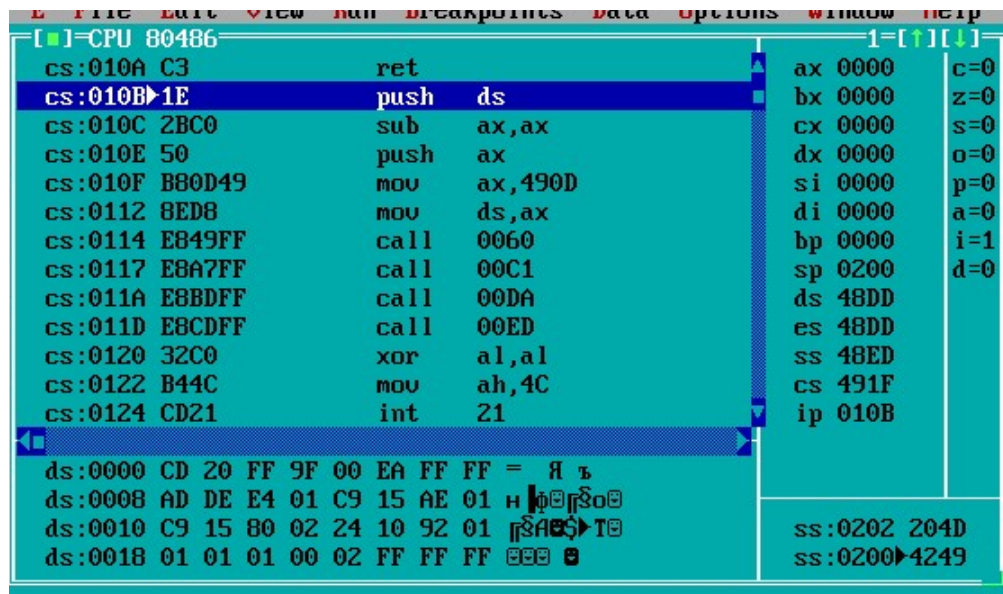


Рисунок 11: EXE-модуль в начале загрузки в память

3. Как определяется стек?

Стек генерируется с помощью описания стекового сегмента в коде, где указывается имя сегмента стека, далее указывается директива

SEGMENT с аргументом STACK. Директива ASSUME соотносит сегмент стека и сегментный регистр SS, который отвечает за стек. SS указывает на начало стека, SP — на конец (SP = 0100H, т. к. размер стека = 256 байт).

4. Как определяется точка входа?

Точка входа определяется директивой END, в аргументах которой указывается метка, с которой начинается программа.

Выводы.

В ходе работы было изучено то, как устроены файлы для COM и EXE исполняемых модулей, а также сами загрузочные модули, их сходства и отличия, способы загрузки в память. Написана программа, которая выводит тип ПК, версию ОС DOS, OEM номер и номер пользователя.

ПРИЛОЖЕНИЕ А

КОД МОДУЛЕЙ

Название файла: good_exe.asm

```
AStack SEGMENT STACK
```

```
DW 256 DUP(?)
```

```
AStack ENDS
```

```
Data SEGMENT
```

```
pc db 'IBM PC type - PC.', 0DH, 0AH, '$'
pcxt db 'IBM PC type - PC/XT.', 0DH, 0AH, '$'
at db 'IBM PC type - AT.', 0DH, 0AH, '$'
ps2_30 db 'IBM PC type - PS2 model 30.', 0DH, 0AH, '$'
ps2_50_60 db 'IBM PC type - PS2 model 50 or 60.', 0DH, 0AH, '$'
ps2_80 db 'IBM PC type - PS2 model 80.', 0DH, 0AH, '$'
pcjr db 'IBM PC type - PCjr.', 0DH, 0AH, '$'
pcconv db 'IBM PC type - PC Convertible.', 0DH, 0AH, '$'

dos_vers db 'MS DOS version - . .', 0DH, 0AH, '$'
oem_num db 'OEM number - .', 0DH, 0AH, '$'
user_num db 'User number - h.', 0DH, 0AH, '$'
```

```
Data ENDS
```

```
MainSeg SEGMENT
```

```
ASSUME CS:MainSeg, DS:Data, ES:NOTHING, SS:AStack
```

```
_print PROC NEAR
```

```
mov AH, 09H
int 21H
ret
```

```
_print ENDP
```

```
byte_to_dec PROC NEAR
```

```
; AH - number, SI - adress of last symbol
```

```
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 AL,00H
    je end_l

    or AL,30H
    mov [SI],AL

end_l:
    pop AX
    pop DX
    pop CX

    ret

byte_to_dec ENDP

tetr_to_hex PROC NEAR

    and AL,0FH      ; save only last part of byte
    cmp AL,09
    jbe next
    add AL,07
next:
    add AL,30H
    ret

tetr_to_hex ENDP

byte_to_hex PROC NEAR

    ; AL - number -> 2 symbols in 16 numb. syst. in AX

    push CX

    mov AH,AL      ; save AL
    call tetr_to_hex
    xchg AL,AH
    mov CL,4
    shr AL,CL
    call tetr_to_hex ; AL - high numb ascii, AH - low numb ascii

    pop CX
    ret

byte_to_hex ENDP

wrd_to_hex PROC NEAR

```



```

; AX - number, DI - last symbol adress

push BX
push AX

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 AX
pop BX
ret

wrd_to_hex ENDP

print_PC_type PROC NEAR

    mov AX,0F000H
    mov ES,AX ; ES -> ROM BIOS
    mov AL,ES:[0FFFEH]

    cmp AL,0FFH
    je _pc_
    cmp AL,0FEH
    je _pc_xt_
    cmp AL,0FBH
    je _pc_xt_
    cmp AL,0FCH
    je _at_
    cmp AL,0FAH
    je _ps2_30_
    cmp AL,0FCH
    je _ps2_50_60_
    cmp AL,0F8H
    je _ps2_80_
    cmp AL,0FDH
    je _pcjr_
    cmp AL,0F9H
    je _pcconv_

_pc_:
    mov DX,offset pc
    jmp _end_

_pc_xt_:
    mov DX,offset pcxt
    jmp _end_

```

```

_at_:
    mov DX,offset at
    jmp _end_

_ps2_30_:
    mov DX,offset ps2_30
    jmp _end_

_ps2_50_60_:
    mov DX,offset ps2_50_60
    jmp _end_

_ps2_80_:
    mov DX,offset ps2_80
    jmp _end_

_pcjr_:
    mov DX,offset pcjr
    jmp _end_

_pcconv_:
    mov DX,offset pcconv
    jmp _end_

_end_:
    call _print
    ret

print_PC_type ENDP

print_dos_version PROC NEAR

    mov AH,30H
    int 21H
    mov SI,offset dos_vers + 17
    call byte_to_dec
    mov AL,AH
    add SI,3
    call byte_to_dec
    mov DX,offset dos_vers
    call _print
    ret

print_dos_version ENDP

print_oem_number PROC NEAR

    mov AH,30H
    int 21H
    mov AL,BH
    mov SI,offset oem_numb + 15
    call byte_to_dec
    mov DX,offset oem_numb
    call _print

```

```

        ret

print_oem_number ENDP

print_user_number PROC NEAR

    mov AH,30H
    int 21H
    mov AX,CX
    mov DI,offset user_numb + 19
    call wrd_to_hex
    mov AL,BL
    call byte_to_hex
    dec DI
    mov [DI],AH
    dec DI
    mov [DI],AL
    mov DX,offset user_numb
    call _print
    ret

print_user_number ENDP

main PROC NEAR

    push DS
    sub AX,AX
    push AX

    mov AX, data
    mov DS,AX

    call print_PC_type
    call print_dos_version
    call print_oem_number
    call print_user_number

    xor AL,AL
    mov AH,4CH
    int 21H

main ENDP

MainSeg ENDS
END main

```

Название файла: good_com.asm

```
MainSeg SEGMENT
    ASSUME CS:MainSeg, DS:MainSeg, ES:NOTHING, SS:NOTHING
    ORG 100H

start:
    jmp begin

data:
    pc db 'IBM PC type - PC.', 0DH, 0AH, '$'
    pcxt db 'IBM PC type - PC/XT.', 0DH, 0AH, '$'
    at db 'IBM PC type - AT.', 0DH, 0AH, '$'
    ps2_30 db 'IBM PC type - PS2 model 30.', 0DH, 0AH, '$'
    ps2_50_60 db 'IBM PC type - PS2 model 50 or 60.', 0DH, 0AH, '$'
    ps2_80 db 'IBM PC type - PS2 model 80.', 0DH, 0AH, '$'
    pcjr db 'IBM PC type - PCjr.', 0DH, 0AH, '$'
    pccnv db 'IBM PC type - PC Convertible.', 0DH, 0AH, '$'

    dos_vers db 'MS DOS version - . .', 0DH, 0AH, '$'
    oem_num db 'OEM number - .', 0DH, 0AH, '$'
    user_num db 'User number - h.', 0DH, 0AH, '$'

begin:
    call main
    xor AL,AL
    mov AH,4CH
    int 21H

_print PROC NEAR

    mov AH, 09H
    int 21H
    ret

_print ENDP

byte_to_dec PROC NEAR

    ; AH - number, SI - adress of last symbol

    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 AL,00H
    je end_l

    or AL,30H
    mov [SI],AL

end_l:
    pop AX
    pop DX
    pop CX

    ret

byte_to_dec ENDP

tetr_to_hex PROC NEAR

    and AL,0FH      ; save only last part of byte
    cmp AL,09
    jbe next
    add AL,07
next:
    add AL,30H
    ret

tetr_to_hex ENDP

byte_to_hex PROC NEAR

    ; AL - number -> 2 symbols in 16 numb. syst. in AX

    push CX

    mov AH,AL      ; save AL
    call tetr_to_hex
    xchg AL,AH
    mov CL,4
    shr AL,CL
    call tetr_to_hex    ; AL - high numb ascii, AH - low numb ascii

    pop CX
    ret

byte_to_hex ENDP

wrd_to_hex PROC NEAR

    ; AX - number, DI - last symbol adress

    push BX

```

```

push AX

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 AX
pop BX
ret

wrd_to_hex ENDP

print_PC_type PROC NEAR

    mov AX,0F000H
    mov ES,AX ; ES -> ROM BIOS
    mov AL,ES:[0FFFEH]

    cmp AL,0FFH
    je _pc_
    cmp AL,0FEH
    je _pc_xt_
    cmp AL,0FBH
    je _pc_xt_
    cmp AL,0FCH
    je _at_
    cmp AL,0FAH
    je _ps2_30_
    cmp AL,0FCH
    je _ps2_50_60_
    cmp AL,0F8H
    je _ps2_80_
    cmp AL,0FDH
    je _pcjr_
    cmp AL,0F9H
    je _pcconv_

_pc_:
    mov DX,offset pc
    jmp _end_

_pc_xt_:
    mov DX,offset pcxt
    jmp _end_

_at_:
    mov DX,offset at

```



```

        jmp _end_

_ps2_30_:
    mov DX,offset ps2_30
    jmp _end_

_ps2_50_60_:
    mov DX,offset ps2_50_60
    jmp _end_

_ps2_80_:
    mov DX,offset ps2_80
    jmp _end_

_pcjr_:
    mov DX,offset pcjr
    jmp _end_

_pcconv_:
    mov DX,offset pcconv
    jmp _end_

_end_:
    call _print
    ret

print_PC_type ENDP

print_dos_version PROC NEAR

    mov AH,30H
    int 21H
    mov SI,offset dos_vers + 17
    call byte_to_dec
    mov AL,AH
    add SI,3
    call byte_to_dec
    mov DX,offset dos_vers
    call _print
    ret

print_dos_version ENDP

print_oem_number PROC NEAR

    mov AH,30H
    int 21H
    mov AL,BH
    mov SI,offset oem_numb + 15
    call byte_to_dec
    mov DX,offset oem_numb
    call _print
    ret

print_oem_number ENDP

```

```

print_user_number PROC NEAR

    mov AH,30H
    int 21H
    mov AX,CX
    mov DI,offset user_num + 19
    call wrd_to_hex
    mov AL,BL
    call byte_to_hex
    dec DI
    mov [DI],AH
    dec DI
    mov [DI],AL
    mov DX,offset user_num
    call _print
    ret

print_user_number ENDP

main PROC NEAR

    call print_PC_type
    call print_dos_version
    call print_oem_number
    call print_user_number
    ret

main ENDP

MainSeg ENDS
END start

```