

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

ЛАБОРАТОРНАЯ РАБОТА № 1
по дисциплине «Операционные системы»
Тема: Исследование структур загрузочных модулей.

Студентка гр. 9381

Андрух И.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

Цель работы.

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

Ход работы.

1) На основе шаблона, приведенного в методических указаниях, был написан текст исходного .COM модуля, который определяет тип PC и версию системы. Был получен и “хороший” .COM модуль и “плохой” .EXE модуль. Результаты работы программ представлен ниже.

Файл com.asm

Полученный com.com

```
C:\>exe2bin com.exe com.com

C:\>com.com
PC Type: FC
Modification number: 5.0
OEM: 255
Serial Number: 090000

C:\>_
```

Полученный com.exe

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

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

    47902 + 457308 Bytes symbol space free

    0 Warning Errors
    0 Severe Errors

C:\>link com.obj

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

Run File [COM.EXE]: com
List File [NUL.MAP]: map_com
Libraries [LIB1]:
LINK : warning L4021: no stack segment

C:\>com._
```

```

C:\>com.exe
FC
5 0 255 000000

0¼ PC Type:
5 0 255 000000

0¼ PC Type:
255 000000

000000 0¼ PC Type:

0¼ PC Type:

```

2) Был написан текст программы, построен и отлажен исходный .EXE модуль, который выполняет те же функции, что и модуль .COM. Таким образом, был получен “хороший” .EXE модуль. Результат работы программы представлен ниже.

Файл exe.asm

Полученный exe.exe

```

C:\>link exe.obj
Microsoft (R) Overlay Linker Version 3.64
Copyright (C) Microsoft Corp 1983-1988. All rights reserved.

Run File [EXE.EXE]: exe
List File [NUL.MAP]: map
Libraries [.LIB]:

C:\>exe.exe
PC Type: FC
Modification number: 5.0
OEM: 0
Serial Number: 090000

```

Функции программ

Названия функций	Описание
TETR_TO_HEX	Перевод десятичной цифры в код символа.
BYTE_TO_HEX	Перевод байта в 16-ной с/с в символьный код
WRD_TO_HEX	Перевод слова в 16-ной с/с в символьный код
BYTE_TO_DEC	Перевод байта в 16-ной с/с в символьный код в 10-ной с/с
PRINT_STRING	Вывод строки.

Ответы на контрольные вопросы:

Отличия исходных текстов COM и EXE программ

1. Сколько сегментов должна содержать COM-программа?

Один сегмент, в котором находятся код и данные.

2. EXE программа?

Программы в формате EXE могут иметь любое количество сегментов. EXE-программа предполагает отдельные сегменты для кода, данных и стека.

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

Директива `ORG 100h`, которая задает смещение для всех адресов программы на 256 байт для префикса программного сегмента (PSP).

Также необходима директива `ASSUME`. С помощью директивы `ASSUME` ассемблеру сообщается информация о соответствии между сегментными регистрами, и программными сегментами. Данная директива позволяет ассемблеру проверять допустимость ссылок и самостоятельно вставлять префиксы переопределения сегментов. Позволяет вычислять смещение меток.

Директива `END` – директива для завершения программы.

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

Нет, не все. Нельзя использовать команды, связанные с адресом сегмента, потому что адрес сегмента до загрузки неизвестен, так как в COM-программах в DOS не содержится таблицы настройки, которая содержит описание адресов, зависящих от размещения загрузочного модуля в ОП, потому что подобные адреса в нем запрещены.

Отличия форматов файлов COM и EXE модулей

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

COM-файл состоит из команд, процедур и данных, используемых в программе. Код, данные и стек располагаются в одном сегменте, код начинается с адреса `0h`.

```

C:\MASM\COM.COM
0000000000: E9 AC 00 50 43 20 54 79 70 65 3A 20 20 0D 0A 24 й PC Type: J$
0000000010: 4D 6F 64 69 66 69 63 61 74 69 6F 6E 20 6E 75 6D Modification num
0000000020: 62 65 72 3A 20 20 2E 20 20 0D 0A 24 4F 45 4D 3A ber: . J$OEM:
0000000030: 20 20 20 0D 0A 24 53 65 72 69 61 6C 20 4E 75 6D J$Serial Num
0000000040: 62 65 72 3A 20 20 20 20 20 20 0D 0A 24 B4 09 ber: J$ro
0000000050: CD 21 C3 24 0F 3C 09 76 02 04 07 04 30 C3 51 8A н!Г$<ov♦♦♦0ГQЪ
0000000060: C4 E8 EF FF 86 C4 B1 04 D2 E8 E8 E6 FF 59 C3 53 Дипя†Д±♦ТиижяYTS
0000000070: 8A FC E8 E9 FF 88 25 4F 88 05 5B 88 05 4F 8A C7 32 E4 E8 Ыийя€%O€♣OЪ32ди
0000000080: DC FF 88 25 4F 88 05 5B C3 51 52 50 32 E4 33 D2 Ыя€%O€♣[ГQRP2д3Т
0000000090: B9 0A 00 F7 F1 80 CA 30 88 14 4E 33 D2 3D 0A 00 № чсЪK0€¶N3Т=■
00000000A0: 73 F1 3D 00 00 76 04 0C 30 88 04 58 5A 59 C3 06 sc= v♦*0€♦XZYГ♣
00000000B0: 53 50 BB 00 F0 8E C3 26 A1 FE FF 8A E0 E8 9E FF SP» рТГ&ЎяЪайҺя
00000000C0: 8D 1E 03 01 89 47 09 58 5B 07 8D 16 03 01 E8 7D К▲▼%GoX[•К-▼и}
00000000D0: FF B4 30 CD 21 50 56 8D 36 10 01 83 C6 15 E8 A8 яГОН!PVК6>ѠЪ$иѠ
00000000E0: FF 83 C6 03 8A C4 E8 A0 FF 5E 58 8D 16 10 01 E8 яЎЖ▼Ъди я^ХК-→и
00000000F0: 5C FF 8A C7 8D 36 2C 01 83 C6 07 E8 8B FF 8D 16 \яЪ3К6,ѠЪЖ•ик яК-
0000000100: 2C 01 E8 49 FF 8A C3 E8 54 FF 8D 3E 36 01 83 C7 ,иИяЪГитяК>6ѠЪ3
0000000110: 0F 89 05 8B C1 8D 3E 36 01 83 C7 14 E8 50 FF 8D %♣♣ВК>6ѠЪ3¶иРяК
0000000120: 16 36 01 E8 28 FF 32 C0 B4 4C CD 21 C3 -6и(я2АгLН!Г

```

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

В файле EXE код, данные и стек содержатся в одном сегменте. С 0 адреса располагается управляющая информация для загрузчика, которая содержит заголовок и таблицу настройки адресов, и показывает, что файл является файлом EXE. Код начинается с адреса 300h.

```

C:\MASM\COM.EXE
0000000100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000001A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000001B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000001C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000001D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000001E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000001F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000200: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000210: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000220: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000230: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000240: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000250: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000260: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000270: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000280: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000290: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000002A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000002B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000002C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000002D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000002E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000002F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000300: E9 AC 00 50 43 20 54 79 70 65 3A 20 20 0D 0A 24 й PC Type: J$
0000000310: 4D 6F 64 69 66 69 63 61 74 69 6F 6E 20 6E 75 6D Modification num
0000000320: 62 65 72 3A 20 20 2E 20 20 0D 0A 24 4F 45 4D 3A ber: . J$OEM:
0000000330: 20 20 20 0D 0A 24 53 65 72 69 61 6C 20 4E 75 6D J$Serial Num
0000000340: 62 65 72 3A 20 20 20 20 20 20 0D 0A 24 B4 09 ber: J$ro
0000000350: CD 21 C3 24 0F 3C 09 76 02 04 07 04 30 C3 51 8A н!Г$<ov♦♦♦0ГQЪ

```

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

```

C:\MASM\EXE.EXE
00000000: 4D 5A 3B 01 02 00 01 00 20 00 00 00 FF FF 00 00 MZ;©  яя
00000001: 00 00 D4 68 00 00 05 00 1E 00 00 00 01 00 69 00  Фh  ♣ ▲  i
00000002: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ♣
00000003: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000004: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000005: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000006: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000007: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000008: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000009: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000B: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000D: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000F: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000011: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000012: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000013: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000014: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000015: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000017: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000018: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000019: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001B: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001D: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001F: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020: 50 43 20 54 79 70 65 3A 20 20 0D 0A 24 4D 6F 64 PC Type:  ♣$Mod
00000021: 69 66 69 63 61 74 69 6F 6E 20 6E 75 6D 62 65 72 ification number
00000022: 3A 20 20 2E 20 20 0D 0A 24 4F 45 4D 3A 20 20 20 : .  ♣$OEM:
00000023: 0D 0A 24 53 65 72 69 61 6C 20 4E 75 6D 62 65 72 ♣$Serial Number
00000024: 3A 20 20 20 20 20 20 20 0D 0A 24 00 00 00 00 00 :  ♣$
00000025: EB 62 90 B4 09 CD 21 C3 24 0F 3C 09 76 02 04 07 лбhгoH!Г$*<ov♦•

```

В отличие от плохого, хороший EXE-файл не содержит директивы ORG 100h (которая выделяет память под PSP), поэтому код начинается с адреса 200h. В отличие от «плохого», в «хорошем» EXE данные, стек и код разделены по сегментам.

Загрузка COM модуля в основную память

1. Какой формат загрузки COM модуля? С какого адреса располагается код?
После загрузки COM-программы в память сегментные регистры указывают на начало PSP. Код располагается с адреса 100h.
2. Что располагается с 0 адреса?
С адреса 0 располагается префикс программного сегмента (PSP).
3. Какие значения имеют сегментные регистры? На какие области памяти они указывают?

DS,ES,SS,CS указывают на один и тот же сегмент памяти, поэтому все регистры имеют одно значение и указывают на начало блока PSP.

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

Стек создается автоматически, указатель стека(SP) в конце сегмента(FFFFh). SS указывает на начало PSP(0h). Стек расположен между SS и SP и адреса изменяются от больших к меньшим, то есть от FFFFh к 0000h.

Загрузка «хорошего» EXE модуля в память

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

Сначала создается PSP. Затем определяется длина тела загрузочного модуля, определяется начальный сегмент. Загрузочный модуль считывается в начальный сегмент, таблица настройки считывается в рабочую память, к полю каждого сегмента прибавляется сегментный адрес начального сегмента, определяются значения сегментных регистров. DS и ES указывают на начало PSP (19F5), CS – на начало сегмента команд (1A0A), а SS – на начало сегмента стека (1A05).

DOSBox 0.74-3, Cpu speed: 3000 cycles, Fram...

AX	0000	SI	0000	CS	1A0A	IP	0000	Stack	+0 4350	Flags	7202
BX	0000	DI	0000	DS	19F5				+2 5420		
CX	013B	BP	0000	ES	19F5	HS	19F5		+4 7079	OF	DF
DX	0000	SP	0000	SS	1A05	FS	19F5		+6 3A65	IF	SF
										ZF	AF
										PF	CF

CMD >													
0000	EB62	JMP		0064	0	1	2	3	4	5	6	7	
0002	90	NOP			DS:0000	CD	20	FF	9F	00	EA	F0	FE
0003	B409	MOV		AH,09	DS:0008	AD	DE	1B	05	C5	06	00	00
0005	CD21	INT		21	DS:0010	18	01	10	01	18	01	92	01
0007	C3	RET			DS:0018	01	01	01	00	02	FF	FF	FF
0008	240F	AND		AL,0F	DS:0020	FF	FF	FF	FF	FF	FF	FF	FF
000A	3C09	CMP		AL,09	DS:0028	FF	FF	FF	FF	EB	19	C0	11
000C	7602	JNA		0010	DS:0030	A2	01	14	00	18	00	F5	19
					DS:0038	FF	FF	FF	FF	00	00	00	00
					DS:0040	05	00	00	00	00	00	00	00
					DS:0048	00	00	00	00	00	00	00	00

2	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DS:0000	CD	20	FF	9F	00	EA	F0	FE	AD	DE	1B	05	C5	06	00	00
DS:0010	18	01	10	01	18	01	92	01	01	01	00	02	FF	FF	FF	FF
DS:0020	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	EB	19	C0	11
DS:0030	A2	01	14	00	18	00	F5	19	FF	FF	FF	FF	00	00	00	00
DS:0040	05	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

1 Step 2 ProcStep 3 Retrieve 4 Help ON 5 BRK Menu 6 7 ↑ 8 ↓ 9 ← 10 →

2. На что указывают регистры DS и ES?

Изначально регистры DS и ES указывают на начало сегмента PSP.

3. Как определяется стек?

Стек определяется при объявлении сегмента стека, в котором указывается, сколько памяти необходимо выделить. В регистры SS и SP записываются значения, указанные в заголовке (SS – начало сегмента стека, SP – конец сегмента стека), а к SS прибавляется сегментный адрес начального сегмента.

4. Как определяется точка входа?

С помощью директивы END, операндом которой является адрес, с которого начинается выполнение программы.

Вывод.

Были изучены различия в структурах исходных текстов модулей типов .COM и .EXE, структуры файлов загрузочных модулей и способы их загрузки в основную память.

ПРИЛОЖЕНИЕ А

КОДЫ ИСХОДНЫХ ПРОГРАММ

com.asm

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H
START:    JMP    BEGIN

;ДАННЫЕ
PC_Type      db    'PC Type:  ', 0dh, 0ah, '$'
Mod_numb     db    'Modification number:  ', 0dh, 0ah, '$'
OEM          db    'OEM:  ', 0dh, 0ah, '$'
S_numb       db    'Serial Number:  ', 0dh, 0ah, '$'

;ПРОЦЕДУРЫ
;-----
;печать строки
PRINT_STRING PROC near
    mov     ah, 09h
    int     21h
    ret
PRINT_STRING ENDP

;-----
;перевод десятичной цифры в код символа
TETR_TO_HEX  PROC near
    and     al, 0fh ;логическое умножение всех пар битов
    cmp     al, 09
    jbe     NEXT ;Переход если ниже или равно
    add     al, 07
NEXT: add     al, 30h
    ret
TETR_TO_HEX  ENDP

;-----
;перевод байта 16 с.с в символьный код
;байт в AL переводится в два символа шестнадцатеричного числа в AX
BYTE_TO_HEX  PROC near
    push    cx
    mov     al, ah
    call    TETR_TO_HEX
    xchg    al, ah
    mov     cl, 4
    shr     al, cl ;логический сдвиг вправо
    call    TETR_TO_HEX ;в AL старшая цифра
```

```

        pop        cx                ;в АН младшая
        ret
BYTE_TO_HEX      ENDP

```

```

;-----
-----

```

```

;перевод в 16 с/с 16-ти разрядного числа
;в AX - число, DI - адрес последнего символа

```

```

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

```

```

;-----
-----

```

```

;перевод байта 16 с.с в символьный код 10 с.с
;si - адрес поля младшей цифры

```

```

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_1
        or         al, 30h
        mov        [si], al
end_1:          pop ax
                pop dx

```

```

                pop        cx
                ret
BYTE_TO_DEC          ENDP

```

```

;-----
-----

```

```

BEGIN:

```

```

push es
push bx
push ax
mov  bx, 0F000h
mov  es, bx
mov  ax, es:[0FFFEh]
mov  ah, al
call BYTE_TO_HEX
lea   bx, PC_Type
mov  [bx + 9], ax ;смещение на количество СИМВОЛОВ
pop   ax
pop   bx
pop   es
lea   dx, PC_Type
call PRINT_STRING

```

```

mov  ah, 30h
int   21h

```

```

push ax
push si
lea   si, Mod_numb
add   si, 21
call BYTE_TO_DEC
add   si, 3
mov  al, ah
call   BYTE_TO_DEC
pop   si
pop   ax
lea   dx, Mod_numb
call PRINT_STRING

```

```

mov  al, bh
lea   si, OEM
add   si, 7
call BYTE_TO_DEC
lea   dx, OEM
call PRINT_STRING

```

```

mov  al, bl
call BYTE_TO_HEX
lea   di, S_numb

```

```

add      di, 15
mov      [di], ax
mov      ax, cx
lea      di, S_numb
add      di, 20
call     WRD_TO_HEX
lea      dx, S_numb
call     PRINT_STRING

xor      al, al
mov      ah, 4ch
int      21h
ret
TESTPC   ENDS
          END    START

```

exe.asm

```

AStack   SEGMENT   STACK
AStack   ENDS

```

```

DATA SEGMENT
PC_Type      db      'PC Type:  ', 0dh, 0ah, '$'
Mod_numb      db      'Modification number:  .  ', 0dh, 0ah, '$'
OEM           db      'OEM:  ', 0dh, 0ah, '$'
S_numb        db      'Serial Number:          ', 0dh, 0ah, '$'
DATA ENDS

```

```

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

```

```

START:
      jmp BEGIN

```

```

;ПРОЦЕДУРЫ

```

```

;-----
-

```

```

;печать строки

```

```

PRINT_STRING PROC near
      mov      ah, 09h
      int      21h
      ret

```

```

PRINT_STRING ENDP

```

```

;-----
-

```

```

;перевод десятичной цифры в код символа

```

```

TETR_TO_HEX      PROC near
      and      al, 0fh ;логическое умножение всех пар битов
      cmp      al, 09
      jbe      NEXT ;Переход если ниже или равно
      add      al, 07
NEXT: add      al, 30h
      ret

```

TETR_TO_HEX ENDP

-

;перевод байта 16 с.с в символьный код

;байт в AL переводится в 2 символа шестнадцатеричного числа в AX

BYTE_TO_HEX PROC near

 push cx

 mov al, ah

 call TETR_TO_HEX

 xchg al, ah

 mov cl, 4

 shr al, cl ;логический сдвиг вправо

 call TETR_TO_HEX

 pop cx

 ret

BYTE_TO_HEX ENDP

-

;перевод в 16 с/с 16-ти разрядного числа

;в AX - число, DI - адрес последнего символа

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

-

;перевод байта 16 с.с в символьный код 10 с.с

;si - адрес поля младшей цифры

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_1
        or      al, 30h
        mov     [si], al
end_1:   pop     ax
        pop     dx
        pop     cx
        ret
BYTE_TO_DEC      ENDP

```

```

;-----
-

```

BEGIN:

```

        push    ds
        sub     ax, ax
        push    ax
        mov     ax, DATA
        mov     ds, ax

        push    es
        push    bx
        push    ax
        mov     bx, 0F000h
        mov     es, bx
        mov     ax, es:[0FFFEh]
        mov     ah, al
        call    BYTE_TO_HEX
        lea     bx, PC_Type
        mov     [bx + 9], ax ;смещение на количество СИМВОЛОВ
        pop     ax
        pop     bx
        pop     es
        lea     dx, PC_Type
        call    PRINT_STRING

        mov     ah, 30h
        int     21h

        push    ax
        push    si
        lea     si, Mod_numb
        add     si, 21
        call    BYTE_TO_DEC
        add     si, 3
        mov     al, ah
        call    BYTE_TO_DEC
        pop     si
        pop     ax
        lea     dx, Mod_numb

```

```

call PRINT_STRING

mov  al, bh
lea  si, OEM
add  si, 7
call BYTE_TO_DEC
lea  dx, OEM
call PRINT_STRING

mov  al, bl
call BYTE_TO_HEX
lea  di, S_numb
add  di, 15
mov  [di], ax
mov  ax, cx
lea  di, S_numb
add  di, 20
call WRD_TO_HEX
lea  dx, S_numb
call PRINT_STRING

xor  al, al
mov  ah, 4ch
int  21h
ret

CODE  ENDS

END  START

```