

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. 9381

Преподаватель

Фоминенко А.Н.

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.

Последовательность действий программы.

- 1) Программа получает информацию о типе ПК из предпоследнего байта ROM BIOS по адресу 0F000:0FFFEh.
- 2) В зависимости от значения байта определяется тип, далее он выводится на экран с помощью процедуры PRINT. В случае, если значение не совпадает с табличными, то выводится значение предпоследнего байта ROM BIOS.
- 3) Программа узнаёт версию ОС, серийные номера OEM и пользователя с помощью вызова функции 30h прерывания 21h. Данные заносятся в соответствующие строки и выводятся на экран.
- 4) Программа завершает свою работу.

Ход работы.

- 1) На основе шаблона, приведённого в методических указаниях, был написан текст исходного .COM модуля. Далее с помощью транслятора `masm.exe` и компоновщика `link.exe` был скомпилирован плохой .EXE модуль. При помощи `exe2bin.exe` из плохого .EXE модуля был построен хороший .COM модуль.
- 2) Был написан текст исходного .EXE модуля. При помощи транслятора `masm.exe` и компоновщика `link.exe` был скомпилирован хороший .EXE модуль.
- 3) Проведено сравнение исходных текстов `lab1_com.asm` и `lab1_exe.asm`.

- 4) При помощи программы FAR были открыты в шестнадцатеричном виде файлы загрузочных модулей lab1_com.com, lab1_com.exe и lab1_exe.exe.
- 5) Был исследован загрузочный модуль .COM при помощи отладчика.
- 6) Был исследован загрузочный модуль .EXE при помощи отладчика.

Функции программ.

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

Ответы на контрольные вопросы.

1) Отличия исходных текстов COM и EXE программ.

1. Сколько сегментов должна содержать COM-программа?

Один сегмент.

2. EXE-программа?

Любое количество сегментов. Сегменты кода, данных и стека находятся отдельно.

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

ORG 100H - устанавливает значение программного счётчика в 100h, так как при загрузке COM- файла в память DOS занимает первые 256 байтов блоком данных PSP и располагает код программы после этого блока. Все программы, которые компилируются в файлы типа COM должны начинаться с этой директивы.

Директива ASSUME указывает ассемблеру, с каким сегментом или группой сегментов связан тот или иной сегментный регистр. Она не изменяет значений сегментных регистров, а только позволяет ассемблеру проверять допустимость ссылок и самостоятельно вставлять префиксы переопределения сегментов при необходимости. Без данной директивы программа не скомпилируется, так как ассемблер не будет знать, относительно чего происходит смещение меток.

Директива END, которой завершается любая программа на ассемблере.

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

Нет. Так как в COM программах в DOS отсутствует таблица настроек, которая содержит описание адресов, зависящих от размещения загрузочного модуля в ОП, нельзя использовать команды, связанные с адресом сегмента.

2) Отличия форматов файлов COM и EXE модулей.

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

В COM файле код, данные и стек располагаются в одном сегменте. Код, как и данные начинается с адреса 0h. Файл в 16-ом виде:

0000000000:	E9 42 01 50 43 20 74 79	70 65 20 2D 20 50 43 0D	éB@PC type - PC
0000000010:	0A 24 50 43 20 74 79 70	65 20 2D 20 58 54 0D 0A	\$PC type - XT
0000000020:	24 50 43 20 74 79 70 65	20 2D 20 41 54 0D 0A 24	\$PC type - AT
0000000030:	50 43 20 74 79 70 65 20	2D 20 50 53 32 20 6D 6F	PC type - PS2 mo
0000000040:	64 65 6C 20 33 30 0D 0A	24 50 43 20 74 79 70 65	del 30\$PC type
0000000050:	20 2D 20 50 53 32 20 6D	6F 64 65 6C 20 38 30 0D	- PS2 model 80
0000000060:	0A 24 50 43 20 74 79 70	65 20 2D 20 50 43 6A 72	\$PC type - PCjr
0000000070:	0D 0A 24 50 43 20 74 79	70 65 20 2D 20 50 43 20	\$PC type - PC
0000000080:	D0 A1 6F 6E 76 65 72 74	69 62 6C 65 0D 0A 24 20	ÿionvertible
0000000090:	20 0D 0A 24 4F 53 20 76	65 72 73 69 6F 6E 20 6E	\$OS version n
00000000A0:	75 6D 62 65 72 20 2D 20	20 20 2E 20 20 0D 0A 24	umber - .
00000000B0:	4F 45 4D 20 6E 75 6D 62	65 72 20 2D 20 20 20 0D	OEM number -
00000000C0:	0A 24 32 34 2D 62 69 74	20 75 73 65 72 20 73 65	\$24-bit user se
00000000D0:	72 69 65 73 20 6E 75 6D	62 65 72 20 2D 20 20 20	ries number -
00000000E0:	20 20 20 20 0D 0A 24 24	0F 3C 09 76 02 04 07 04	\$so<ov♦♦
00000000F0:	30 C3 51 8A E0 E8 EF FF	86 C4 B1 04 D2 E8 E8 E6	QŠæiÿ+Ä±ðèæ
0000000100:	FF 59 C3 53 8A FC E8 E9	FF 88 25 4F 88 05 4F 8A	ÿVÄSŠüèÿ~%0^+OŠ
0000000110:	C7 E8 DE FF 88 25 4F 88	05 5B C3 51 52 32 E4 33	Çèÿ~%0^+[ÄQR2ä3
0000000120:	D2 B9 0A 00 F7 F1 80 CA	30 88 14 4E 33 D2 3D 0A	Ò¹ ÷ñ€Ê0^JN30=
0000000130:	00 73 F1 3C 00 74 04 0C	30 88 04 5A 59 C3 50 B4	sñ< t♦00^ZYÄP^
0000000140:	09 CD 21 58 C3 B8 00 F0	8E C0 26 A0 FE FF 3C FF	oÍ!XÄ, ðŽ&Ä þÿ<ÿ
0000000150:	74 1F 3C FE 74 1B 3C FB	74 1D 3C FA 74 1F 3C FC	t<øt<-<ût<-<ût<ü
0000000160:	74 21 3C F8 74 23 3C FD	74 25 3C F9 74 27 EB 38	t!<øt#<ÿt%<ût^è8
0000000170:	90 BA 03 01 EB 32 90 BA	12 01 EB 2C 90 BA 21 01	0°♥0è20°\$0è,0°!0
0000000180:	EB 26 90 BA 30 01 EB 20	90 BA 49 01 EB 1A 90 BA	è&0°00è 0°I0è→0°
0000000190:	62 01 EB 14 90 BA 73 01	EB 0E 90 E8 54 FF BB 8F	b0èJ0°s0è,0èTÿ»0
00000001A0:	01 88 07 88 67 01 8B D3	E8 93 FF B4 30 CD 21 BE	0^•^g0<0è^ÿ^0Í!%
00000001B0:	A9 01 8A D4 E8 64 FF 8A	C2 83 C6 03 E8 5C FF BA	00Š0èdyŠÄfA♥è\ÿ°
00000001C0:	94 01 E8 79 FF 8A C7 E8	28 FF BF BD 01 88 05 88	”0èÿÿŠÇè(ÿ¿%0^+
00000001D0:	65 01 BA B0 01 E8 66 FF	8A C3 E8 15 FF BF DF 01	e0°°0èfyŠÄèÿÿ¿B0
00000001E0:	88 05 88 65 01 8B C1 83	C7 05 E8 16 FF BA C2 01	^+^e0<ÄfÇ+è-ÿ°Ä0
00000001F0:	E8 4B FF 32 C0 B4 4C CD	21	èky2Ä^LÍ!

2. Какова структура файла плохого EXE? С какого адреса располагается код?

Что располагается с адреса 0?

В плохом EXE файле код, данные и стек располагаются в одном сегменте. Данные и код начинаются с адреса 300h. С адреса 0h находится управляющая информация для загрузчика, которая содержит заголовок и таблицу настройки адресов. Это можно проверить, открыв файл в 16-ом виде:

0000000000:	4D 5A F9 00 03 00 00 00	20 00 00 00 FF FF 00 00	MZu ▼	yy
0000000010:	00 00 38 36 00 01 00 00	1E 00 00 00 01 00 00 00	86 0 ▲	0
0000000020:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
0000000030:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		

0000000300:	E9 42 01 50 43 20 74 79	70 65 20 2D 20 50 43 0D	éB0PC type - PC
0000000310:	0A 24 50 43 20 74 79 70	65 20 2D 20 58 54 0D 0A	\$PC type - XT
0000000320:	24 50 43 20 74 79 70 65	20 2D 20 41 54 0D 0A 24	\$PC type - AT
0000000330:	50 43 20 74 79 70 65 20	2D 20 50 53 32 20 6D 6F	PC type - PS2 mo
0000000340:	64 65 6C 20 33 30 0D 0A	24 50 43 20 74 79 70 65	del 30PC type
0000000350:	20 2D 20 50 53 32 20 6D	6F 64 65 6C 20 38 30 0D	- PS2 model 80
0000000360:	0A 24 50 43 20 74 79 70	65 20 2D 20 50 43 6A 72	\$PC type - PCjr
0000000370:	0D 0A 24 50 43 20 74 79	70 65 20 2D 20 50 43 20	PC type - PC
0000000380:	D0 A1 6F 6E 76 65 72 74	69 62 6C 65 0D 0A 24 20	onvertible
0000000390:	20 0D 0A 24 4F 53 20 76	65 72 73 69 6F 6E 20 6E	\$OS version n
00000003A0:	75 6D 62 65 72 20 2D 20	20 20 2E 20 20 0D 0A 24	umber - .
00000003B0:	4F 45 4D 20 6E 75 6D 62	65 72 20 2D 20 20 20 0D	OEM number -
00000003C0:	0A 24 32 34 2D 62 69 74	20 75 73 65 72 20 73 65	\$24-bit user se
00000003D0:	72 69 65 73 20 6E 75 6D	62 65 72 20 2D 20 20 20	ries number -
00000003E0:	20 20 20 20 0D 0A 24 24	0F 3C 09 76 02 04 07 04	\$so<ov
00000003F0:	30 C3 51 8A E0 E8 EF FF	86 C4 B1 04 D2 E8 E8 E6	0AQ\$àèiÿ†Ä±♦òèæ
0000000400:	FF 59 C3 53 8A FC E8 E9	FF 88 25 4F 88 05 4F 8A	yvÄs5uèéÿ%0^+0Š
0000000410:	C7 E8 DE FF 88 25 4F 88	05 5B C3 51 52 32 E4 33	çèpy^%0^+[ÄQR2ãz
0000000420:	D2 B9 0A 00 F7 F1 80 CA	30 88 14 4E 33 D2 3D 0A	Ô± ÷ñ€Ê0^¶N3Ô=
0000000430:	00 73 F1 3C 00 74 04 0C	30 88 04 5A 59 C3 50 B4	sñ< t♦00^♦ZYÄP^
0000000440:	09 CD 21 58 C3 B8 00 F0	8E C0 26 A0 FE FF 3C FF	oÍ!XÃ. ôŽÀ& þÿ<ÿ
0000000450:	74 1F 3C FE 74 1B 3C FB	74 1D 3C FA 74 1F 3C FC	t▼<þt<-<ût<-<ût▼<û
0000000460:	74 21 3C F8 74 23 3C FD	74 25 3C F9 74 27 EB 38	t!<øt#<ÿt%<ût'è8
0000000470:	90 BA 03 01 EB 32 90 BA	12 01 EB 2C 90 BA 21 01	0000000470: 90 BA 03 01 EB 32 90 BA
0000000480:	EB 26 90 BA 30 01 EB 20	90 BA 49 01 EB 1A 90 BA	è&0000 00I0è→00
0000000490:	62 01 EB 14 90 BA 73 01	EB 0E 90 E8 54 FF BB 8F	b000000490: 62 01 EB 14 90 BA 73 01
00000004A0:	01 88 07 88 67 01 8B D3	E8 93 FF B4 30 CD 21 BE	0^•^g0<0è“ÿ^0Í!%
00000004B0:	A9 01 8A D4 E8 64 FF 8A	C2 83 C6 03 E8 5C FF BA	00Š0èdÿŠÄfA♥è\ÿ0
00000004C0:	94 01 E8 79 FF 8A C7 E8	28 FF BF BD 01 88 05 88	”0èÿÿŠÇè(ÿ;X0^+^
00000004D0:	65 01 BA B0 01 E8 66 FF	8A C3 E8 15 FF BF DF 01	e0000004D0: 65 01 BA B0 01 E8 66 FF
00000004E0:	88 05 88 65 01 8B C1 83	C7 05 E8 16 FF BA C2 01	^+^e0<ÁfÇ+è-ÿ0Ä0
00000004F0:	E8 4B FF 32 C0 B4 4C CD	21	èKÿ2Ä^LÍ!

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

В хорошем EXE-файле код, данные и стек находятся в различных сегментах, в отличие от плохого, где они находятся в одном сегменте. Также изменён порядок расположения сегментов в памяти, и стек имеет другой размер - 256 байт(100h). В отличие от плохого, хороший EXE-файл не содержит директивы ORG 100h, которая выделяет память под PSP. Поэтому код начинается с адреса 210h. Файл в 16-ом виде:

0000000000:	4D 5A 0E 00 03 00 01 00	20 00 11 00 FF FF 21 00	MZ
0000000010:	00 01 2C 7D 10 00 00 00	1E 00 00 00 01 00 72 00	0,)> ▲ 0 r
0000000020:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000030:	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	

0000000210:	EB 5F 90 24 0F 3C 09 76	02 04 07 04 30 C3 51 8A	л_ђ\$о<ov0♦♦♦0ГQљ
0000000220:	E0 E8 EF FF 86 C4 B1 04	D2 E8 E8 E6 FF 59 C3 53	аипя†Д±♦ТиижяУГS
0000000230:	8A FC E8 E9 FF 88 25 4F	88 05 4F 8A C7 E8 DE FF	љыйя€%O€#OљзиЮя
0000000240:	88 25 4F 88 05 5B C3 51	52 32 E4 33 D2 B9 0A 00	€%O€# [ГQR2дЗTNљ
0000000250:	F7 F1 80 CA 30 88 14 4E	33 D2 3D 0A 00 73 F1 3C	чсђK0€ђN3T=љ sc<
0000000260:	00 74 04 0C 30 88 04 5A	59 C3 50 B4 09 CD 21 58	t♦00€♦ZYГPгoH!X
0000000270:	C3 B8 12 00 8E D8 B8 00	F0 8E C0 26 A0 FE FF 3C	Гёђ Tљё рђA& юя<
0000000280:	FF 74 1F 3C FE 74 1B 3C	FB 74 1D 3C FA 74 1F 3C	ят▼<ют<<ыт<<ьт▼<
0000000290:	FC 74 21 3C F8 74 23 3C	FD 74 25 3C F9 74 27 EB	ьт!<шт#<эт%<шт'л
00000002A0:	38 90 BA 0A 00 EB 32 90	BA 19 00 EB 2C 90 BA 28	8ђєљ л2ђє↓ л,ђє(
00000002B0:	00 EB 26 90 BA 37 00 EB	20 90 BA 50 00 EB 1A 90	л&ђє7 л ђєР л→ђ
00000002C0:	BA 69 00 EB 14 90 BA 7A	00 EB 0E 90 E8 4F FF BB	єі лђђєz лђђиОя»
00000002D0:	96 00 88 07 88 67 01 8B	D3 E8 8E FF B4 30 CD 21	- €•€g0<Уиђяг0H!
00000002E0:	BE B0 00 8A D4 E8 5F FF	8A C2 83 C6 03 E8 57 FF	s° љфи_яљBђЖ▼иљя
00000002F0:	BA 9B 00 E8 74 FF 8A C7	E8 23 FF BF C4 00 88 05	є> итяљзи#яїД €♦
0000000300:	88 65 01 BA B7 00 E8 61	FF 8A C3 E8 10 FF BF E6	€є0€• иаяљги→яїж
0000000310:	00 88 05 88 65 01 8B C1	83 C7 05 E8 11 FF BA C9	€♦€є0<БfЗ+и<яєЙ
0000000320:	00 E8 46 FF 32 C0 B4 4C	CD 21 50 43 20 74 79 70	иFя2ArLH!PC typ
0000000330:	65 20 2D 20 50 43 0D 0A	24 50 43 20 74 79 70 65	e - PCљ\$PC type
0000000340:	20 2D 20 58 54 0D 0A 24	50 43 20 74 79 70 65 20	- XTљ\$PC type
0000000350:	2D 20 41 54 0D 0A 24 50	43 20 74 79 70 65 20 2D	- ATљ\$PC type -
0000000360:	20 50 53 32 20 6D 6F 64	65 6C 20 33 30 0D 0A 24	PS2 model 30љ\$
0000000370:	50 43 20 74 79 70 65 20	2D 20 50 53 32 20 6D 6F	PC type - PS2 mo
0000000380:	64 65 6C 20 38 30 0D 0A	24 50 43 20 74 79 70 65	del 80љ\$PC type
0000000390:	20 2D 20 50 43 6A 72 0D	0A 24 50 43 20 74 79 70	- PCjrљ\$PC typ
00000003A0:	65 20 2D 20 50 43 20 D0	A1 6F 6E 76 65 72 74 69	e - PC Pђonverti
00000003B0:	62 6C 65 0D 0A 24 20 20	0D 0A 24 4F 53 20 76 65	bleљ\$ љ\$OS ve
00000003C0:	72 73 69 6F 6E 20 6E 75	6D 62 65 72 20 2D 20 20	rsion number -
00000003D0:	20 2E 20 20 0D 0A 24 4F	45 4D 20 6E 75 6D 62 65	. љ\$OEM numbe
00000003E0:	72 20 2D 20 20 20 0D 0A	24 32 34 2D 62 69 74 20	r - љ\$24-bit
00000003F0:	75 73 65 72 20 73 65 72	69 65 73 20 6E 75 6D 62	user series numb
0000000400:	65 72 20 2D 20 20 20 20	20 20 20 0D 0A 24	er - љ\$

3) Загрузка COM модуля в основную память.

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

При загрузке COM файла в память DOS занимает первые 256 байт блоком данных PSP. Код располагается с адреса 100h. После загрузки COM-программы в память сегментные регистры указывают на начало PSP.

AX 0000	SI 0000	CS 119C	IP 0100	Stack +0 0000
BX 0000	DI 0000	DS 119C		+2 0000
CX 01F9	BP 0000	ES 119C	HS 119C	+4 0000
DX 0000	SP FFF5	SS 119C	FS 119C	+6 0000

2. Что располагается с адреса 0?

С адреса 0 располагается префикс программного сегмента PSP.

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

Регистры DS, ES, CS, SS имеют одно значение 119C, так как указывают на один и тот же блок PSP.

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

Стек создаётся автоматически. Регистр SS указывает на начало блока PSP(0h), а SP указывает на конец модуля (FFF5h). То есть стек располагается между адресами SS:0000h и SS:FFF5h и заполняется с конца модуля в сторону уменьшения адресов.

4) Загрузка хорошего EXE модуля в основную память.

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

Сначала создаётся PSP. Затем определяется длина тела загрузочного модуля, определяется начальный сегмент. Далее 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. В результате DS и ES указывают на начало PSP(119C), CS - на начало сегмента команд (11AC), а SS - на начало сегмента стека (11CD). Управление передаётся загруженной задаче по адресу, указанному в заголовке.

AX 0000	SI 0000	CS 11AC	IP 0010	Stack +0 0000
BX 0000	DI 0000	DS 119C		+2 0000
CX 020E	BP 0000	ES 119C	HS 119C	+4 0000
DX 0000	SP 0100	SS 11CD	FS 119C	+6 0000

2. На что указывают регистры DS и ES?

Данные регистры указывают на начало блока PSP.

3. Как определяется стек?

Регистр SS указывает на начало сегмента стека, а SS:SP - на конец сегмента стека.

4. Как определяется точка входа?

Точка входа определяется параметром после директивы END, в качестве

которого нужно передать метку, с которой программа начнёт выполнение команд.

Вывод.

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

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab1_com.asm

```

TESTPC  SEGMENT
        ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
        ORG 100H
START:   JMP BEGIN
; Данные
Type_PC      db 'PC type - PC',0DH,0AH,'$'
Type_XT      db 'PC type - XT',0DH,0AH,'$'
Type_AT      db 'PC type - AT',0DH,0AH,'$'
Type_PS_30   db 'PC type - PS2 model 30',0DH,0AH,'$'
Type_PS_80   db 'PC type - PS2 model 80',0DH,0AH,'$'
Type_PCjr    db 'PC type - PCjr',0DH,0AH,'$'
Type_PCConvert db 'PC type - PC Convertible',0DH,0AH,'$'
Type_Unknown db ' ',0DH,0AH,'$'

OS_Version_Num db 'OS version number - . ',0DH,0AH,'$'
Num_OEM        db 'OEM number - ',0DH,0AH,'$'
User_series_num db '24-bit user series 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 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-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 PROC NEAR ; вывод строки на экран
        push ax
        mov ah, 9h
        int 21h
        pop ax
        ret
PRINT ENDP
;-----
; КОД
BEGIN:
        ;получаем информацию о типе ПК
        mov ax, 0F000h
        mov es, ax

```

```

    mov al, es:[0FFFEh]
    ;сравниваем полученное значение с табличными значениями
    cmp al, 0FFh
    je type_1
    cmp al, 0FEh
    je type_1
    cmp al, 0FBh
    je type_2
    cmp al, 0FAh
    je type_3
    cmp al, 0FCh
    je type_4
    cmp al, 0F8h
    je type_5
    cmp al, 0FDh
    je type_6
    cmp al, 0F9h
    je type_7
    jmp print_type
;выбираем строку соответствующую типу ПК
type_1:
    mov dx, offset Type_PC
    jmp print_type
type_2:
    mov dx, offset Type_XT
    jmp print_type
type_3:
    mov dx, offset Type_AT
    jmp print_type
type_4:
    mov dx, offset Type_PS_30
    jmp print_type
type_5:
    mov dx, offset Type_PS_80
    jmp print_type
type_6:
    mov dx, offset Type_PCjr
    jmp print_type
type_7:
    mov dx, offset Type_PCConvert
    jmp print_type
unknown:
    call BYTE_TO_HEX
    mov bx, offset Type_Unknown
    mov [bx], al
    mov [bx+1], ah
    mov dx, bx
print_type:
    ;печатаем тип ПК
    call PRINT
    ;узнаём информацию о версии ОС и серийные номера OEM и

```

```

пользователя
    mov ah, 30h
    int 21h
    mov si, offset OS_Version_Num + 21
    mov dl, ah
    call BYTE_TO_DEC
    mov al, dl
    add si, 3
    call BYTE_TO_DEC
    ; Выводим на экран версию ОС
    mov dx, offset OS_Version_Num
    call PRINT
    mov al, bh
    call BYTE_TO_HEX
    mov di, offset Num_OEM + 13
    mov [di], al
    mov [di+1], ah
    ; Выводим на экран серийный номер OEM
    mov dx, offset Num_OEM
    call PRINT
    mov al, bl
    call BYTE_TO_HEX
    mov di, offset User_series_num + 29
    mov [di], al
    mov [di+1], ah
    mov ax, cx
    add di, 5
    call WRD_TO_HEX
    ; Выводим на экран серийный номер пользователя
    mov dx, offset User_series_num
    call PRINT
; Выход в DOS
    xor AL,AL
    mov AH,4Ch
    int 21h
TESTPC  ENDS
        END START ; Конец модуля, START - точка входа

```

Имя файла: lab1_exe.asm

```

DOSSEG
.model small
.stack 100h

.data
Type_PC      db 'PC type - PC',0DH,0AH,'$'
Type_XT      db 'PC type - XT',0DH,0AH,'$'
Type_AT      db 'PC type - AT',0DH,0AH,'$'
Type_PS_30   db 'PC type - PS2 model 30',0DH,0AH,'$'
Type_PS_80   db 'PC type - PS2 model 80',0DH,0AH,'$'

```

```

Type_PCjr      db 'PC type - PCjr',0DH,0AH,'$'
Type_PCConvert db 'PC type - PC Convertible',0DH,0AH,'$'
Type_Unknown   db ' ', 0DH, 0AH, '$'

OS_Version_Num db 'OS version number - . ',0DH,0AH,'$'
Num_OEM        db 'OEM number - ',0DH,0AH,'$'
User_series_num db '24-bit user series number - ',0DH,0AH,'$'

.code
START: 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 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-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_1
    or AL,30h
    mov [SI],AL
end_1: pop DX
    pop CX
    ret
BYTE_TO_DEC ENDP

PRINT PROC NEAR    ; вывод строки на экран
    push ax
    mov ah, 9h
    int 21h
    pop ax
    ret
PRINT ENDP
;-----
; КОД
BEGIN:
    mov ax, @data
    mov ds, ax
    ;получаем информацию о типе ПК
    mov ax, 0F000h
    mov es, ax
    mov al, es:[0FFFEh]
    ;сравниваем полученное значение с табличными значениями
    cmp al, 0FFh
    je type_1
    cmp al, 0FEh
    je type_1
    cmp al, 0FBh
    je type_2
    cmp al, 0FAh
    je type_3
    cmp al, 0FCh

```

```

        je type_4
        cmp al, 0F8h
        je type_5
        cmp al, 0FDh
        je type_6
        cmp al, 0F9h
        je type_7
        jmp print_type
;выбираем строку соответствующую типу ПК
type_1:
        mov dx, offset Type_PC
        jmp print_type
type_2:
        mov dx, offset Type_XT
        jmp print_type
type_3:
        mov dx, offset Type_AT
        jmp print_type
type_4:
        mov dx, offset Type_PS_30
        jmp print_type
type_5:
        mov dx, offset Type_PS_80
        jmp print_type
type_6:
        mov dx, offset Type_PCjr
        jmp print_type
type_7:
        mov dx, offset Type_PCConvert
        jmp print_type
unknown:
        call BYTE_TO_HEX
        mov bx, offset Type_Unknown
        mov [bx], al
        mov [bx+1], ah
        mov dx, bx
print_type:
        ;печатаем тип ПК
        call PRINT
        ;узнаём информацию о версии ОС и серийные номера OEM и
пользователя
        mov ah, 30h
        int 21h
        mov si, offset OS_Version_Num + 21
        mov dl, ah
        call BYTE_TO_DEC
        mov al, dl
        add si, 3
        call BYTE_TO_DEC
        ; Выводим на экран версию ОС
        mov dx, offset OS_Version_Num

```



```

call PRINT
mov al, bh
call BYTE_TO_HEX
mov di, offset Num_OEM + 13
mov [di], al
mov [di+1], ah
; Выводим на экран серийный номер OEM
mov dx, offset Num_OEM
call PRINT
mov al, bl
call BYTE_TO_HEX
mov di, offset User_series_num + 29
mov [di], al
mov [di+1], ah
mov ax, cx
add di, 5
call WRD_TO_HEX
; Выводим на экран серийный номер пользователя
mov dx, offset User_series_num
call PRINT
; Выход в DOS
    xor AL,AL
    mov AH,4Ch
    int 21H
END START ; Конец модуля, START - точка входа

```