

**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. 9383

Хотяков Е.П.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

### **Цель работы.**

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

### **Задание.**

Шаг 1. Написать текст исходного .COM модуля, который определяет тип РС и версию системы. Результатом выполнения этого шага будет «хороший» .COM модуль, а также необходимо построить «плохой» .EXE, полученный из исходного текста для .COM.

Шаг 2. Написать текст исходного .EXE модуля, который выполняет те же функции, что и модуль в шаге 1 и отладить его.

Шаг 3. Сравнить исходные тексты для .COM и .EXE модулей. Ответить на вопросы «Отличия исходных текстов COM и EXE программ».

Шаг 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».

Шаг 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.

Шаг 6. Открыть отладчик TD.EXE и загрузить «хороший» .EXE. Ответить на контрольные вопросы «Загрузка «хорошего» EXE в основную память».

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

## Выполнение работы.

### Шаг 1:

На первом шаге была написана программа на ассемблере, из которой были получены “хороший” .COM модуль и “плохой” .EXE модуль.

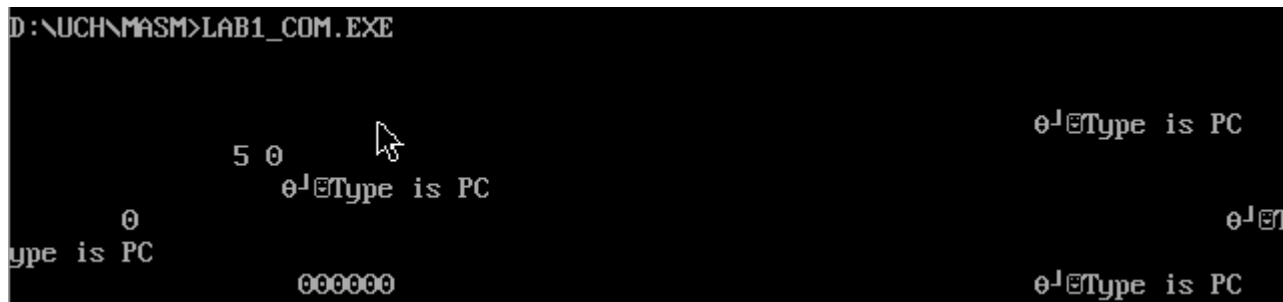


Рисунок 1 - работа плохого .EXE

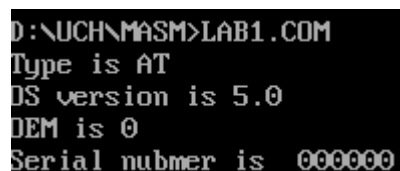


Рисунок 2 - работа хорошего .COM

### Шаг 2:

На втором шаге была написана программа на ассемблере, из которой был получен “хороший” .EXE модуль.

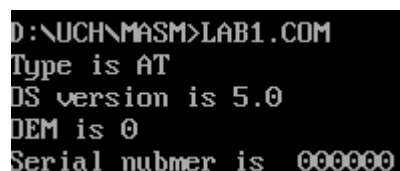


Рисунок 3 - работа хорошего .EXE

Разработанный программный код см. в приложении А.

### Шаг 3:

Ответы на контрольные вопросы “Отличия исходных текстов COM и EXE программ” после сравнения исходных текстов для .COM и .EXE модулей.

1) Сколько сегментов должна содержать COM-программа?

В COM программе должен быть сегмент кода(внутри которого также объявляются и данные). Стек в COM-файле генерируется автоматически. Следовательно сама программа содержит только 1 сегмент.

2) EXE-программа?

В EXE программе же может использовать от одного(не менее одного) сегмента, причем сегмент данных, кода и стека записываются отдельно друг от друга.

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

Для смещения от нулевого адреса на 256 байт(смещения до конца PSP) должна использовать директива `ORG 100h`. Для того, чтобы сегменты данных и кода указывали на один и тот же сегмент необходимо использовать директиву `ASSUME` с параметрами(где вместо `TESTPC` должно быть название сегмента) `CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING`.

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

Нет, не все. Например, команда `MOV AX, CODE` будет некорректна для COM программы, т.к. в момент ассемблирования и редактирования связей сегментное значение для сегмента `CODE` неизвестно(.COM не может предоставить загрузчику перечня всех сегментных ссылок).

#### Шаг 4:

Ответы на контрольные вопросы “Отличия форматов COM и EXE модулей” после просмотра .COM и .EXE файлов в шестнадцатеричном виде:

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

COM-программа начинается с самого верха, с ячейки `0h`, хотя при загрузке модуля установится смещение в `100h`. Сам код располагается в одном сегменте(в котором находятся сегмента кода и данных) и ограничивается 64 Кб памяти.

|                                     |                         |                  |
|-------------------------------------|-------------------------|------------------|
| 0000000000: E9 D9 01 54 79 70 65 20 | 69 73 20 50 43 0D 0A 24 | éÙ@Type is PC    |
| 0000000010: 54 79 70 65 20 69 73 20 | 50 43 2F 58 54 0D 0A 24 | Type is PC/XT    |
| 0000000020: 54 79 70 65 20 69 73 20 | 41 54 0D 0A 24 54 79 70 | Type is AT       |
| 0000000030: 65 20 69 73 20 50 53 32 | 20 6D 6F 64 65 6C 20 33 | e is PS2 model 3 |
| 0000000040: 30 0D 0A 24 54 79 70 65 | 20 69 73 20 50 53 32 20 | Type is PS2      |
| 0000000050: 6D 6F 64 65 6C 20 38 30 | 0D 0A 24 54 79 70 65 20 | model 80         |
| 0000000060: 69 73 20 50 43 20 43 6F | 6E 76 65 72 74 69 62 6C | is PC Convertibl |
| 0000000070: 65 0D 0A 24 54 79 70 65 | 20 69 73 20 50 43 6A 72 | e                |
| 0000000080: 0D 0A 24 45 52 52 4F 52 | 3A 20 4E 6F 20 74 79 70 | ERROR: No typ    |
| 0000000090: 65 20 69 6E 20 74 61 62 | 6C 65 3A 20 0D 0A 24 4F | e in table:      |
| 00000000A0: 53 20 76 65 72 73 69 6F | 6E 20 69 73 20 20 2E 20 | S version is .   |
| 00000000B0: 20 0D 0A 24 4F 45 4D 20 | 69 73 20 20 20 0D 0A 24 | OEM is           |
| 00000000C0: 53 65 72 69 61 6C 20 6E | 75 62 6D 65 72 20 69 73 | Serial number is |
| 00000000D0: 20 20 20 20 20 20 0D 0A | 24 24 0F 3C 09 76 02 04 |                  |
| 00000000E0: 07 04 30 C3 51 8A E0 E8 | EF FF 86 C4 B1 04 D2 E8 |                  |
| 00000000F0: E8 E6 FF 59 C3 53 8A FC | E8 E9 FF 88 25 4F 88 05 |                  |
| 0000000100: 4F 8A C7 E8 DE FF 88 25 | 4F 88 05 5B C3 51 52 32 |                  |
| 0000000110: E4 33 D2 B9 0A 00 F7 F1 | 80 CA 30 88 14 4E 33 D2 |                  |
| 0000000120: 3D 0A 00 73 F1 3C 00 74 | 04 0C 30 88 04 5A 59 C3 |                  |
| 0000000130: B4 09 CD 21 C3 B8 00 F0 | 8E C0 26 A0 FE FF 3C FF |                  |
| 0000000140: 74 22 3C FE 74 24 3C FB | 74 20 3C FC 74 22 3C FA |                  |
| 0000000150: 74 24 3C F8 74 26 3C FD | 74 2E 3C F9 74 24 BA 83 |                  |
| 0000000160: 01 EB 2B 90 BA 03 01 EB | 25 90 BA 10 01 EB 1F 90 |                  |
| 0000000170: BA 20 01 EB 19 90 BA 2D | 01 EB 13 90 BA 44 01 EB |                  |
| 0000000180: 0D 90 BA 5B 01 EB 07 90 | BA 74 01 EB 01 90 E8 9F |                  |
| 0000000190: FF C3 B4 30 CD 21 50 BE | 9F 01 83 C6 0E E8 6D FF |                  |
| 00000001A0: 58 8A C4 83 C6 03 E8 64 | FF BA 9F 01 E8 81 FF BE |                  |
| 00000001B0: B4 01 83 C6 07 8A C7 E8 | 53 FF BA B4 01 E8 70 FF |                  |
| 00000001C0: BF C0 01 83 C7 17 8B C1 | E8 2A FF 8A C3 E8 14 FF |                  |
| 00000001D0: 83 EF 02 89 05 BA C0 01 | E8 55 FF C3 E8 56 FF E8 |                  |
| 00000001E0: B0 FF 32 C0 B4 4C CD 21 |                         |                  |

Рисунок 3 - Шестнадцатеричный вид COM-файла

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

Внутри “плохого” EXE также будет располагаться один сегмент данных. Код(и данные) располагается по адресу 300h. С адреса 0h же располагаются заголовок с MZ-байтами(идентификатор - стандартный формат 16-битных исполняемых файлов с расширением .EXE для DOS) и таблица настроек.



|                                      |                         |                    |
|--------------------------------------|-------------------------|--------------------|
| 0000000000: 4D 5A E8 00 03 00 00 00  | 20 00 00 00 FF FF 00 00 | MZè ♥      ÿÿ      |
| 0000000010: 00 00 5E 2A 00 01 00 00  | 1E 00 00 00 01 00 00 00 | ^* 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 |                    |
| 0000000030: E9 D9 01 54 79 70 65 20  | 69 73 20 50 43 0D 0A 24 | éU0Type is PC      |
| 00000000310: 54 79 70 65 20 69 73 20 | 50 43 2F 58 54 0D 0A 24 | Type is PC/XT      |
| 00000000320: 54 79 70 65 20 69 73 20 | 41 54 0D 0A 24 54 79 70 | Type is AT         |
| 00000000330: 65 20 69 73 20 50 53 32 | 20 6D 6F 64 65 6C 20 33 | e is PS2 model 3   |
| 00000000340: 30 0D 0A 24 54 79 70 65 | 20 69 73 20 50 53 32 20 | 0Type is PS2       |
| 00000000350: 6D 6F 64 65 6C 20 38 30 | 0D 0A 24 54 79 70 65 20 | model 80Type       |
| 00000000360: 69 73 20 50 43 20 43 6F | 6E 76 65 72 74 69 62 6C | is PC Convertibl   |
| 00000000370: 65 0D 0A 24 54 79 70 65 | 20 69 73 20 50 43 6A 72 | eType is PCjr      |
| 00000000380: 0D 0A 24 45 52 52 4F 52 | 3A 20 4E 6F 20 74 79 70 | ERROR: No typ      |
| 00000000390: 65 20 69 6E 20 74 61 62 | 6C 65 3A 20 0D 0A 24 4F | e in table: 0      |
| 000000003A0: 53 20 76 65 72 73 69 6F | 6E 20 69 73 20 20 2E 20 | S version is .     |
| 000000003B0: 20 0D 0A 24 4F 45 4D 20 | 69 73 20 20 20 0D 0A 24 | 0OEM is 0          |
| 000000003C0: 53 65 72 69 61 6C 20 6E | 75 62 6D 65 72 20 69 73 | Serial nubmer is   |
| 000000003D0: 20 20 20 20 20 20 0D 0A | 24 24 0F 3C 09 76 02 04 | 00\$0<ov00         |
| 000000003E0: 07 04 30 C3 51 8A E0 E8 | EF FF 86 C4 B1 04 D2 E8 | •00ÃQŠàèiÿ+Ä±00è   |
| 000000003F0: E8 E6 FF 59 C3 53 8A FC | E8 E9 FF 88 25 4F 88 05 | èæÿYÄSŠüèéÿ~%0^    |
| 00000000400: 4F 8A C7 E8 DE FF 88 25 | 4F 88 05 5B C3 51 52 32 | 0ŠÇèÿ~%0^0[ÃQR2    |
| 00000000410: E4 33 D2 B9 0A 00 F7 F1 | 80 CA 30 88 14 4E 33 D2 | ä30± ÷ñ€Ê0^JN30    |
| 00000000420: 3D 0A 00 73 F1 3C 00 74 | 04 0C 30 88 04 5A 59 C3 | = sñ< t000^ZYÃ     |
| 00000000430: B4 09 CD 21 C3 B8 00 F0 | 8E C0 26 A0 FE FF 3C FF | ´oÍ!Ã. õŽ&À pÿ<ÿ   |
| 00000000440: 74 22 3C FE 74 24 3C FB | 74 20 3C FC 74 22 3C FA | t"<pt\$<ût <ût"<ú  |
| 00000000450: 74 24 3C F8 74 26 3C FD | 74 2E 3C F9 74 24 BA 83 | t\$<øt&<ÿt.<ût\$°f |
| 00000000460: 01 EB 2B 90 BA 03 01 EB | 25 90 BA 10 01 EB 1F 90 | 0è+0°♥0è%0°0è♥0    |
| 00000000470: BA 20 01 EB 19 90 BA 2D | 01 EB 13 90 BA 44 01 EB | ° 0èJ0°-0è!!0°D0è  |
| 00000000480: 0D 90 BA 5B 01 EB 07 90 | BA 74 01 EB 01 90 E8 9F | 00°[0è•0°t0è00èY   |
| 00000000490: FF C3 B4 30 CD 21 50 BE | 9F 01 83 C6 0E E8 6D FF | ÿÃ´0Í!P%ÿ0fÆðemÿ   |
| 000000004A0: 58 8A C4 83 C6 03 E8 64 | FF BA 9F 01 E8 81 FF BE | XŠÄfÆ♥èdÿ°ÿ0è0ÿ%   |
| 000000004B0: B4 01 83 C6 07 8A C7 E8 | 53 FF BA B4 01 E8 70 FF | ´0fÆ•ŠÇèSÿ°´0èpÿ   |
| 000000004C0: BF C0 01 83 C7 17 8B C1 | E8 2A FF 8A C3 E8 14 FF | ¿À0fÇ&◊Áè*yŠÄèÿÿ   |
| 000000004D0: 83 EF 02 89 05 BA C0 01 | E8 55 FF C3 E8 56 FF E8 | fï0%0°À0èUÿÃèVÿè   |

Рисунок 4 - Шестнадцатеричный вид "плохого" EXE-файла

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

В “хорошей” EXE программе сегменты данных, кода и стека разделены друг от друга. В начале файла лежит заголовок, который имеет таблицу для настройки адресов(которая, в отличие от “плохого” EXE-файла, валидна). Адресация кода начинается с 400h, т.к. 200h отводится под заголовок и таблицу настроек и еще 200h под стек.



|                                      |                         |                   |
|--------------------------------------|-------------------------|-------------------|
| 0000000000: 4D 5A F4 01 03 00 01 00  | 20 00 00 00 FF FF 00 00 | MZôø♥ 0      ÿÿ   |
| 0000000010: 00 02 70 5F 03 01 2E 00  | 1E 00 00 00 01 00 04 01 | 0p_♥0. ▲    0 ♦0  |
| 0000000020: 2E 00 00 00 00 00 00 00  | 00 00 00 00 00 00 00 00 | .                 |
| 00000000400: 54 79 70 65 20 69 73 20 | 50 43 0D 0A 24 54 79 70 | Type is PC!\$Type |
| 00000000410: 65 20 69 73 20 50 43 2F | 58 54 0D 0A 24 54 79 70 | e is PC/XT!\$Type |
| 00000000420: 65 20 69 73 20 41 54 0D | 0A 24 54 79 70 65 20 69 | e is AT!\$Type i  |
| 00000000430: 73 20 50 53 32 20 6D 6F | 64 65 6C 20 33 30 0D 0A | s PS2 model 30!\$ |
| 00000000440: 24 54 79 70 65 20 69 73 | 20 50 53 32 20 6D 6F 64 | \$Type is PS2 mod |
| 00000000450: 65 6C 20 38 30 0D 0A 24 | 54 79 70 65 20 69 73 20 | el 80!\$Type is   |
| 00000000460: 50 43 20 43 6F 6E 76 65 | 72 74 69 62 6C 65 0D 0A | PC Convertible!\$ |
| 00000000470: 24 54 79 70 65 20 69 73 | 20 50 43 6A 72 0D 0A 24 | \$Type is PCjr!\$ |
| 00000000480: 45 52 52 4F 52 3A 20 4E | 6F 20 74 79 70 65 20 69 | ERROR: No type i  |
| 00000000490: 6E 20 74 61 62 6C 65 3A | 20 0D 0A 24 4F 53 20 76 | n table: !\$OS v  |
| 000000004A0: 65 72 73 69 6F 6E 20 69 | 73 20 20 2E 20 20 0D 0A | ersion is . !\$   |
| 000000004B0: 24 4F 45 4D 20 69 73 20 | 20 0D 0A 24 53 65 72    | \$OEM is !\$Ser   |
| 000000004C0: 69 61 6C 20 6E 75 62 6D | 65 72 20 69 73 20 20 20 | ial nubmer is     |
| 000000004D0: 20 20 20 0D 0A 24 00 00 | 00 00 00 00 00 00 00 00 | !\$               |
| 000000004E0: 24 0F 3C 09 76 02 04 07 | 04 30 C3 51 8A E0 E8 EF | \$o<ov0♦♦0AQŠàèi  |
| 000000004F0: FF 86 C4 B1 04 D2 E8 E8 | E6 FF 59 C3 53 8A FC E8 | ÿ+Ä±0èèæÿYÄSŠüè   |
| 00000000500: E9 FF 88 25 4F 88 05 4F | 8A C7 E8 DE FF 88 25 4F | éÿ~%0^0ŠÇèpÿ~%0   |
| 00000000510: 88 05 5B C3 51 52 32 E4 | 33 D2 B9 0A 00 F7 F1 80 | ^+ [ÄQR2ä30¹  ÷ñ€ |
| 00000000520: CA 30 88 14 4E 33 D2 3D | 0A 00 73 F1 3C 00 74 04 | Ê0^JN30= sñ< t♦   |
| 00000000530: 0C 30 88 04 5A 59 C3 B4 | 09 CD 21 C3 B8 00 F0 8E | 90^♦ZYÄ^oÍ!Ä. ðŽ  |
| 00000000540: C0 26 A0 FE FF 3C FF 74 | 22 3C FE 74 24 3C FB 74 | À& pÿ<ÿt"<pt\$<ût |
| 00000000550: 20 3C FC 74 22 3C FA 74 | 24 3C F8 74 26 3C FD 74 | <ût"<ût\$<øt&<ÿt  |
| 00000000560: 2E 3C F9 74 24 BA 80 00 | EB 2B 90 BA 00 00 EB 25 | .<ût\$ø€ ë+  ë%   |
| 00000000570: 90 BA 0D 00 EB 1F 90 BA | 1D 00 EB 19 90 BA 2A 00 | ë  ë  *           |
| 00000000580: EB 13 90 BA 41 00 EB 0D | 90 BA 58 00 EB 07 90 BA | ë!!  A ë  X ë  *  |
| 00000000590: 71 00 EB 01 90 E8 9F FF | C3 B4 30 CD 21 50 BE 9C | q ë  èÿÿÄ^0Í!P%æ  |
| 000000005A0: 00 83 C6 0E E8 6D FF 58 | 8A C4 83 C6 03 E8 64 FF | fÄèemÿXSÄfÄ♥èdÿ   |
| 000000005B0: BA 9C 00 E8 81 FF BE B1 | 00 83 C6 07 8A C7 E8 53 | æ è  ÿ%± fÄ.ŠÇèS  |
| 000000005C0: FF BA B1 00 E8 70 FF BF | BD 00 83 C7 17 8B C1 E8 | ÿø± èpÿç% fÇ±<Äè  |
| 000000005D0: 2A FF 8A C3 E8 14 FF 83 | EF 02 89 05 BA BD 00 E8 | *ÿŠÄèÿÿf10%ø% è   |
| 000000005E0: 55 FF C3 B8 20 00 8E D8 | E8 51 FF E8 AB FF 32 C0 | UÿÄ. Ž0èQÿè«ÿ2Ä   |
| 000000005F0: B4 4C CD 21             |                         | ¹Í!               |

Рисунок 5 - Шестнадцатеричный вид "хорошего" EXE-файла

### Шаг 5:

|  |                      |              |     |
|--|----------------------|--------------|-----|
| [CPU 80486]  |                      | [FPU]        |     |
| cs:0100→E9D901                                       | jmp 02DC ↓           | ax 0000      | c=0 |
| cs:0103 54   | push sp              | bx 0000      | z=0 |
| cs:0104 7970   | jns 0176             | cx 0000      | s=0 |
| cs:0106 65206973                                     | and gs:[bx+di+73],dl | dx 0000      | o=0 |
| cs:010A 205043                                       | and [bx+si+43],dl    | si 0000      | p=0 |
| cs:010D 0D0A24                                       | or ax,240A           | di 0000      | a=0 |
| cs:0110 54   | push sp              | bp 0000      | i=1 |
| cs:0111 7970   | jns 0183             | sp FFFE      | d=0 |
| cs:0113 65206973                                     | and gs:[bx+di+73],dl | ds 48DD      |     |
| cs:0117 205043                                       | and [bx+si+43],dl    | es 48DD      |     |
| cs:011A 2F   | das                  | ss 48DD      |     |
| cs:011B 58   | pop ax               | cs 48DD      |     |
| cs:011C 54   | push sp              | ip 0100      |     |
| ds:0000 CD 20 FF 9F 00 EA FF FF = f 0                |                      | ss:0000 20CD |     |
| ds:0008 AD DE E4 01 C9 15 AE 01 i 20 8<0             |                      | ss:FFFE→0000 |     |
| ds:0010 C9 15 80 02 24 10 92 01 8C 05 00 00 00 00 00 |                      |              |     |
| ds:0018 01 01 01 00 02 FF FF FF 00 00 00 00 00 00 00 |                      |              |     |

Ответы на контрольные вопросы “Загрузка COM модуля в основную память”:

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

Сначала определяется участок в оперативной памяти, у которого достаточно места для загрузки COM модуля, после чего он помещается в память, начиная с PSP со смещением в 100h.

2) Что располагается с адреса 0?

С адреса 0 располагается сегмент PSP размером 100h

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

CS, DS, ES и SS указывают на PSP и имеют значения 48DD, а SP указывает на конец сегмента FFFE.

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

В COM-программе стек генерируется автоматически в процессе создания.

Стек занимает всю область сегмента, при этом регистр SS указывает на начало(0h), SP – на конец(FFFEh).

### Шаг 6:

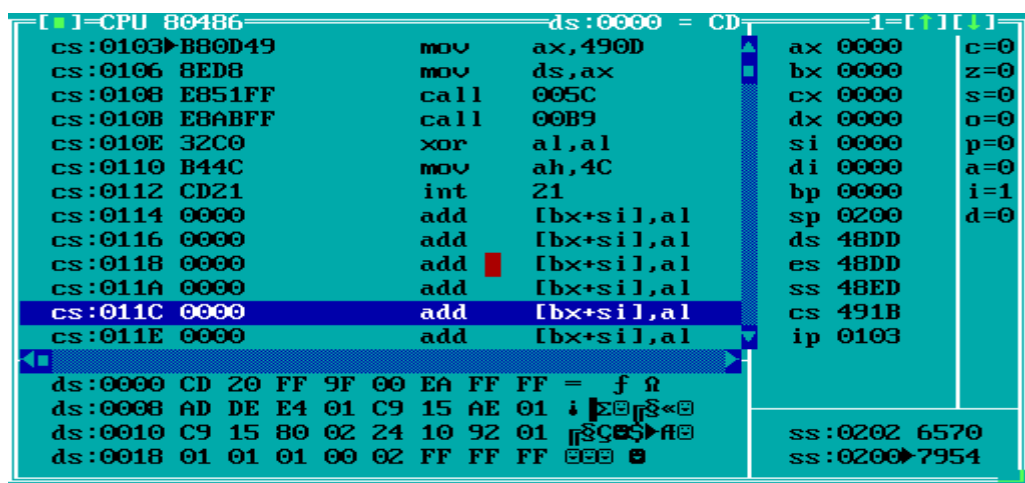


Рисунок 1 - Отладка "хорошего" EXE-файла



Ответы на контрольные вопросы “Загрузка “хорошего” EXE модуля в основную память”:

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

Загрузка файла начинается со смещением 100h(PSP). При этом начинается считывание информации заголовка и выполняется перемещение адресов сегментов: DS и ES указывают на 48DD(начало PSP), SS - 48ED(начало стека), CS – 491B(начало сегмента команд). В IP же загружается смещение точки входа в программу.

2) На что указывают регистры DS и ES?

На начало PSP

3) Как определяется стек?

Стек определяется директивой .stack, при этом SS и SP указывают на начало и конец стека соответственно.

4) Как определяется точка входа?

Точка входа определяется из метки, расположенной после директивы END.

### **Выводы.**

В результате проделанной работы были написаны программы под COM и EXE-структуры, были изучены их различия и различия их загрузки в память

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab1\_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

PC\_TYPE db 'Type is PC',0DH,0AH,'\$'

PC\_XT\_TYPE db 'Type is PC/XT',0DH,0AH,'\$'

AT\_TYPE db 'Type is AT',0DH,0AH,'\$'

PS30\_TYPE db 'Type is PS2 model 30',0DH,0AH,'\$'

PS80\_TYPE db 'Type is PS2 model 80',0DH,0AH,'\$'

PCCON\_TYPE db 'Type is PC Convertible',0DH,0AH,'\$'

PCjr\_TYPE db 'Type is PCjr',0DH,0AH,'\$'

NO\_TYPE db 'ERROR: No type in table: ',0DH,0AH,'\$'

OS\_VERSION db 'OS version is . ',0DH,0AH,'\$'

OEM db 'OEM is ',0DH,0AH,'\$'

SERIAL db 'Serial nubmer is ',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_1
        or AL,30h
        mov [SI],AL

```

```

end_1: pop DX
      pop CX
      ret
BYTE_TO_DEC ENDP

```

WRITE\_STRING PROC near; Вывод строки текста

```

      mov AH,09h
      int 21h
      ret

```

WRITE\_STRING ENDP

WHAT\_TYPE PROC near

```

      mov AX, 0f000h
      mov ES, AX
      mov AL, es:[0ffffh]

```

```

      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 ps30_write
      cmp AL, 0F8h
      je ps80_write
      cmp AL, 0FDh
      je pcjr_write
      cmp AL, 0F9h
      je pccon_write
      mov dx, offset NO_TYPE
      jmp WRITE_STRING_TYPE

```

pc\_write:

```

      mov dx, offset PC_TYPE
      jmp WRITE_STRING_TYPE

```

pc\_xt\_write:

```

      mov dx, offset PC_XT_TYPE

```



```

        jmp WRITE_STRING_TYPE
at_write:
        mov dx, offset AT_TYPE
        jmp WRITE_STRING_TYPE
ps30_write:
        mov dx, offset PS30_TYPE
        jmp WRITE_STRING_TYPE
ps80_write:
        mov dx, offset PS80_TYPE
        jmp WRITE_STRING_TYPE
pccon_write:
        mov dx, offset PCCON_TYPE
        jmp WRITE_STRING_TYPE
pcjr_write:
        mov dx, offset PCjr_TYPE
        jmp WRITE_STRING_TYPE
WRITE_STRING_TYPE:
        CALL WRITE_STRING
        ret
WHAT_TYPE ENDP

```

```

WHAT_OS_VERSION PROC near
        MOV AH, 30h
        INT 21h
        push AX

        mov SI, offset OS_VERSION
        add SI, 14
        call BYTE_TO_DEC
        pop AX
        mov AL, AH
        add SI, 3
        call BYTE_TO_DEC
        mov DX, offset OS_VERSION
        call WRITE_STRING

        mov SI, offset OEM
        add SI, 7
        mov AL, BH
        call BYTE_TO_DEC
        mov DX, offset OEM

```

```

    call WRITE_STRING

    mov DI, offset SERIAL
    add DI, 23
    mov AX, CX
    call WRD_TO_HEX
    mov AL, BL
    call BYTE_TO_HEX
    sub DI, 2
    mov [DI], AX
    mov DX, offset SERIAL
    call WRITE_STRING
    ret
WHAT_OS_VERSION ENDP

;-----
; КОД
BEGIN:
    CALL WHAT_TYPE
    CALL WHAT_OS_VERSION
;.....
; Выход в DOS
    xor AL,AL
    mov AH,4Ch
    int 21H
TESTPC ENDS
    END START ; Конец модуля, START - точка входа

```

Название файла: lab1\_exe.asm

```

AStack SEGMENT STACK
    DW 256 DUP(?)
AStack ENDS
DATA SEGMENT
; Данные
PC_TYPE db 'Type is PC',0DH,0AH,$'
PC_XT_TYPE db 'Type is PC/XT',0DH,0AH,$'
AT_TYPE db 'Type is AT',0DH,0AH,$'
PS30_TYPE db 'Type is PS2 model 30',0DH,0AH,$'
PS80_TYPE db 'Type is PS2 model 80',0DH,0AH,$'
PCCON_TYPE db 'Type is PC Convertible',0DH,0AH,$'
PCjr_TYPE db 'Type is PCjr',0DH,0AH,$'

```

NO\_TYPE db 'ERROR: No type in table: ',0DH,0AH,'\$'

OS\_VERSION db 'OS version is . ',0DH,0AH,'\$'

OEM db 'OEM is ',0DH,0AH,'\$'

SERIAL db 'Serial nubmer is ',0DH,0AH,'\$'

DATA ENDS

CODE SEGMENT

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

; 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

WRITE_STRING PROC near; Вывод строки текста
    mov AH,09h
    int 21h
    ret
WRITE_STRING ENDP

WHAT_TYPE PROC near
    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 ps30_write
cmp AL, 0F8h
je ps80_write
cmp AL, 0FDh
je pcjr_write
cmp AL, 0F9h
je pccon_write
mov dx, offset NO_TYPE
jmp WRITE_STRING_TYPE

```

```

pc_write:
    mov dx, offset PC_TYPE
    jmp WRITE_STRING_TYPE
pc_xt_write:
    mov dx, offset PC_XT_TYPE
    jmp WRITE_STRING_TYPE
at_write:
    mov dx, offset AT_TYPE
    jmp WRITE_STRING_TYPE
ps30_write:
    mov dx, offset PS30_TYPE
    jmp WRITE_STRING_TYPE
ps80_write:
    mov dx, offset PS80_TYPE
    jmp WRITE_STRING_TYPE
pccon_write:
    mov dx, offset PCCON_TYPE
    jmp WRITE_STRING_TYPE

```

```

pcjr_write:
    mov dx, offset PCjr_TYPE
    jmp WRITE_STRING_TYPE
WRITE_STRING_TYPE:
    CALL WRITE_STRING
    ret
WHAT_TYPE ENDP

```

```

WHAT_OS_VERSION PROC near
    MOV AH, 30h
    INT 21h
    push AX

    mov SI, offset OS_VERSION
    add SI, 14
    call BYTE_TO_DEC
    pop AX
    mov AL, AH
    add SI, 3
    call BYTE_TO_DEC
    mov DX, offset OS_VERSION
    call WRITE_STRING

    mov SI, offset OEM
    add SI, 7
    mov AL, BH
    call BYTE_TO_DEC
    mov DX, offset OEM
    call WRITE_STRING

    mov DI, offset SERIAL
    add DI, 23
    mov AX, CX
    call WRD_TO_HEX
    mov AL, BL
    call BYTE_TO_HEX
    sub DI, 2
    mov [DI], AX
    mov DX, offset SERIAL
    call WRITE_STRING
    ret

```

WHAT\_OS\_VERSION ENDP

;-----

; КОД

MAIN PROC FAR

mov AX, DATA

mov DS, AX

call WHAT\_TYPE

call WHAT\_OS\_VERSION

xor AL, AL

mov AH, 4Ch

int 21h

MAIN ENDP

CODE ENDS

END MAIN