

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

ОТЧЕТ
по лабораторной работе №1
по дисциплине «Операционные системы»
Тема: «Исследование структур заголовочных модулей»

Студент гр. 7381

Трушников А.П.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2019

Цель работы.

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

Основные теоретические положения.

Тип IBM PC узнается путем считывания предпоследнего байта с ROM BIOS. Его значение сравнивается с кодами таблицы, представленной ниже.

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
- AH – номер модификации
- BH – серийный номер OEM
- BL:CH – 24-битовый серийный номер пользователя

Выполнение работы.

Написан текст исходного .COM модуля, который определяет тип PC и версию системы. Для решения поставленной задачи был использован шаблон ассемблерного текста с функциями управляющей программы и процедурами перевода двоичных кодов в символы шестнадцатеричных чисел и десятичное число из раздела “общие сведения” методических указаний. Для того, чтобы

узнать тип IBM PC программа обращается к предпоследнему байту ROM BIOS. Далее полученное значение сравнивается с таблицей. Для определения версии MS DOS используется функция 30h 21h-го прерывания. И в соответствии с полученными данными в регистрах.

Написан текст исходного .EXE модуля с тем же функционалом.

1. Результат выполнения «плохого» .EXE модуля:

```

PC TYPE:
PC TYPE: 5 0 255
000000
PC TYPE: 255 000000
PC TYPE:
000000
PC TYPE:

```

2. Результат выполнения «хорошего» .COM модуля:

```

C:\>COM_CODE.COM
PC TYPE:AT
SYSTEM VERSION: 5.0
OEM number: 255
USER SERIAL NUMBER: 000000

```

3. Результат выполнения «хорошего» .EXE модуля:

```

C:\>EXE_CODE.EXE
PC TYPE:AT
SYSTEM VERSION: 5.0
OEM number: 255
USER SERIAL NUMBER: 000000

```

Отличия исходных текстов COM и EXE файлов

- 1 COM-программа должна содержать один сегмент – сегмент кода.
- 2 EXE-программа должна содержать три сегмента –сегмент стека, сегмент данных, сегмент кода.
- 3 В тексте COM-программы обязательно должна быть директива ASSUME, которая должна указывать, что сегмент кода и данных начинается с одного и того же места. Также должна быть директива ORG, указывающая, сколько места в памяти необходимо зарезервировать под PSP.

4 В COM-программе можно использовать все форматы команд.

2. Цитата:

В COM-программе можно использовать все форматы команд.

Нет. Для демонстрации, предлагаю попробовать получить адрес сегмента TESTPC в ах. Дополните свой ответ результатом с объяснением почему Вы увидели то, что увидели.

Ответ:

- Вот что произошло в результате попытки получить адрес сегмента TESTPC в ах.

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

Ошибка произошла потому что, адреса сегментов определяются загрузчиком в момент загрузки модуля в память, а в модулях типа COM отсутствует таблица настроек, в которой хранится информация о местоположении в файле полей адресов, требующих настройки на адреса основной памяти.

Отличия форматов файлов COM и EXE модулей

- 1 Структура COM-файла очень компактна (сам файл весит гораздо меньше), код располагается с нулевого адреса.

3. Цитата:

Структура COM-файла очень компактна (сам файл весит гораздо меньше)

А есть какие-то ограничения на размер, кстати?

Ответ:

- Есть. В системах DOS и в 8-битной CP/M COM-файл — простой тип исполняемого файла, при выполнении которого данные, код и стек находятся в одном и том же 16-битном сегменте. Поэтому размер файла не может превышать 65280 байт.

2 Структура «плохого» EXE-файла менее компактна, чем у COM-файла. Код располагается с адреса 300h, с нулевого адреса располагается таблица настроек, при помощи которых строится данный EXE-файл.

4. Цитата:

файла. Код располагается с адреса 300h, с нулевого адреса располагается таблица настроек, при помощи которых строится данный EXE-файл.

Во-первых почему с 300? Требуется подтверждать эти утверждения скриншотами. Далее, в начале .exe модуля есть 2 первых байта MZ - это тоже часть таблицы настроек?

Ответ:

- MZ в начале EXE модуля это формат исполняемых файлов MS DOS, он не является частью таблицы настроек адресов, так как таблица состоит из элементов, число которых записано в байтах 06-07.

D:\spbetu_os_2019_7381\Trushnikov\lab1\bad.EXE	h 1251
000000002D0:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000002E0:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000002F0:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000300:	E9 C3 01 50 43 20 54 59 50 45 3A 24 50 43 20 55 йГОPC TYPE:\$PC U
00000000310:	4E 44 45 46 49 4E 45 44 3A 20 20 0D 0A 24 53 59 NDEFINED: J0\$SY
00000000320:	53 54 45 4D 20 56 45 52 53 49 4F 4E 3A 20 20 2E STEM VERSION: .
00000000330:	20 20 0D 0A 24 4F 45 4D 20 6E 75 6D 62 65 72 3A J0\$OEM number:
00000000340:	20 20 20 20 20 20 0D 0A 24 55 53 45 52 20 53 45 J0\$USER SE
00000000350:	52 49 41 4C 20 4E 55 4D 42 45 52 3A 20 20 20 20 RIAL NUMBER:
00000000360:	0D 0A 24 50 43 0D 0A 24 50 43 2F 58 54 0D 0A 24 J0\$PCJ0\$PC/XTJ0\$
00000000370:	41 54 0D 0A 24 50 53 32 20 4D 4F 44 45 4C 20 33 ATJ0\$PS2 MODEL 3
00000000380:	30 0D 0A 24 50 53 32 20 4D 4F 44 45 4C 20 35 30 0J0\$PS2 MODEL 50
00000000390:	20 4F 52 20 36 30 0D 0A 24 50 53 32 20 4D 4F 44 OR 60J0\$PS2 MOD
000000003A0:	45 4C 20 38 30 0D 0A 24 50 53 6A 72 0D 0A 24 50 EL 80J0\$PSjrJ0\$P
000000003B0:	53 20 43 6F 6E 76 65 72 74 69 62 6C 65 0D 0A 24 S ConvertibleJ0\$
000000003C0:	24 0F 3C 09 76 02 04 07 04 30 C3 51 8A E0 E8 EF \$o<ov0♦♦♦0ГQЪаип
000000003D0:	FF 86 C4 B1 04 D2 E8 E8 E6 FF 59 C3 B4 09 CD 21 я†Д♦ТиижаYГГОH!
000000003E0:	C3 53 8A FC E8 E4 FF 88 25 4F 88 05 4F 8A C7 E8 ГSъидя€%O€+Oъ3и
000000003F0:	D9 FF 88 25 4F 88 05 5B C3 51 52 32 E4 33 D2 B9 Щя€%O€+[ГQR2д3T№
00000000400:	0A 00 F7 F1 80 CA 30 88 14 4E 33 D2 3D 0A 00 73 0 чсЪK0€ЅN3T=0 s
00000000410:	F1 3C 00 74 04 0C 30 88 04 5A 59 C3 06 BB 00 F0 с< т♦Q0€♦ZYГ▲» р
00000000420:	8E C3 BB 00 00 26 A1 FE FF 07 C3 50 56 8D 36 1E ЪГ» &Ўюя•ГРVК6▲
00000000430:	01 83 C6 10 E8 C2 FF 83 C6 03 8A C4 E8 BA FF 5E 0ГЖ»иВяГЖ♥ЪДиея^
00000000440:	58 C3 50 53 51 56 8A C3 E8 80 FF 8D 3E 49 01 83 ХГPSQVЪГиЪяК>I0Г
00000000450:	C7 16 89 05 8B C1 8D 3E 49 01 83 C7 1B E8 81 FF 3-%▲<БК>I0Г3<иГя
00000000460:	5E 59 5B 58 C3 50 53 56 8A C7 8D 36 35 01 83 C6 ^YГХГPSVЪ3К650ГЖ

- 4 Стек занимает всё свободное пространство до конца файла (размер .COM файла не может превышать 64 кб), оставшееся после загрузки данных и кода. В данном случае значение регистра SP=FFF5.

Загрузка хорошего EXE-модуля в основную память

- 1 Порядок загрузки EXE-модуля: PSP, сегмент кода, сегмент данных, сегмент стека. Сегментные регистры на момент загрузки программы имеют значения: ES=119C, CS=11B8, DS=119C, SS=11CE. На начальном этапе ES=DS, так как не были выполнены команды “mov ax, data; mov ds, ax”, т.е. в регистр данных не был помещён адрес сегмента данных.
- 2 Регистры DS и ES указывают на начало PSP
- 3 В регистры SS и SP записываются значения, указанные в заголовке, а к SS прибавляется сегментный адрес начального сегмента.

7. Цитата:

В регистры SS и SP записываются значения, указанные в заголовке, а к SS прибавляется сегментный адрес начального сегмента.

О, этот ответ я уже видел в #3, вопросы те же: что за заголовок (у Вас он до этого почти не упоминался, хорошо, что я заметил, кстати), что за "начальный" сегмент?

Ответ:

- Заголовок — это управляющая информация для загрузчика, которая расположена в начале файла. в SS записывается адрес начале сегмента стека, а в SP адрес вершины стека.
- 4 Оператором END start_procedure_name. Эта информация хранится в заголовке модуля.

Выводы.

Исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов загрузки в

основную память. Реализована программа на языке ассемблера позволяющая определить тип IBM PC и тип системы.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ ТЕКСТ .COM МОДУЛЯ

```
TESTPC      SEGMENT

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

              ORG         100H

START: JMP     BEGIN
```

;ДАННЫЕ

```
PC_TYPE      DB          'PC TYPE:', '$'

PC_UNDEFINED  DB          'PC UNDEFINED: ', 0DH, 0AH, '$'

SYSTEM_VERSION DB        'SYSTEM VERSION: . ', 0DH, 0AH, '$'

OEM_NUMBER    DB          'OEM NUMBER:   ', 0DH, 0AH, '$'

SERIAL_NUMBER DB          'USER SERIAL NUMBER: ', 0DH, 0AH, '$'

PC            DB          'PC', 0DH, 0AH, '$'

PCXT          DB          'PC/XT', 0DH, 0AH, '$'

AT            DB          'AT', 0DH, 0AH, '$'

PS2_30        DB          'PS2 MODEL 30', 0DH, 0AH, '$'

PS2_50        DB          'PS2 MODEL 50 OR 60', 0DH, 0AH, '$'

PS2_80        DB          'PS2 MODEL 80', 0DH, 0AH, '$'

PCJR          DB          'PSJR', 0DH, 0AH, '$'

PC_CONVERTIBLE DB        'PS CONVERTIBLE', 0DH, 0AH, '$'
```

```
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


WRITE_MSG      PROC NEAR

        MOV     AH,09H

        INT     21H

        RET

WRITE_MSG      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

```
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

```

```

GET_PC_NUMBER PROC NEAR

```

```

; ФУНКЦИЯ ОПРЕДЕЛЯЮЩАЯ ТИП PC

```

```

PUSH    ES

MOV     BX,0F000H

MOV     ES,BX

MOV     BX,0

MOV     AX,ES:[0FFFEH]

POP     ES

RET

```

```

GET_PC_NUMBER ENDP

```

```

GET_SYS_VER PROC NEAR

```

```

; ФУНКЦИЯ ОПРЕДЕЛЯЮЩАЯ ВЕРСИЮ СИСТЕМЫ

```

```

PUSH    AX

PUSH    SI

```

```

        LEA        SI,SYSTEM_VERSION
        ADD        SI,16
        CALL  BYTE_TO_DEC
        ADD        SI,3
        MOV  AL,AH
        CALL  BYTE_TO_DEC
        POP  SI
        POP  AX
        RET
GET_SYS_VER        ENDP

```

```

GET_SERIAL_NUMPROC NEAR
        PUSH  AX
        PUSH  BX
        PUSH  CX
        PUSH  SI
        MOV   AL,BL
        CALL  BYTE_TO_HEX
        LEA   DI,SERIAL_NUMBER
        ADD   DI,22
        MOV   [DI],AX
        MOV   AX,CX
        LEA   DI,SERIAL_NUMBER
        ADD   DI,27
        CALL  WRD_TO_HEX
        POP   SI
        POP   CX

```

```

        POP        BX

        POP        AX

        RET

GET_SERIAL_NUMENDP

GET_OEM_NUM        PROC NEAR

; ФУНКЦИЯ ОПРЕДЕЛЯЮЩАЯ OEM

        PUSH    AX

        PUSH    BX

        PUSH    SI

        MOV     AL,BH

        LEA     SI,OEM_NUMBER

        ADD     SI,14

        CALL    BYTE_TO_DEC

        POP     SI

        POP     BX

        POP     AX

        RET

GET_OEM_NUM        ENDP

DEFINE_PC_TYPE     PROC NEAR

; ФУНКЦИЯ, ОПРЕДЕЛЯЮЩАЯ ТИП PC

        CMP     AL,0FFh

        JNE     CMP1

        MOV     DX,OFFSET PC

        RET

```

```

CMP1: CMP    AL, 0FEH

        JNE          CMP2

        MOV    DX, OFFSET PCXT

        RET


CMP2: CMP    AL, 0FCH

        JNE          CMP3

        MOV    DX, OFFSET AT

        RET


CMP3: CMP    AL, 0FAH

        JNE          CMP4

        MOV    DX, OFFSET PS2_30

        RET


CMP4: CMP    AL, 0FCH

        JNE          CMP5

        MOV    DX, OFFSET PS2_50

        RET


CMP5: CMP    AL, 0F8H

        JNE          CMP6

        MOV    DX, OFFSET PS2_80

        RET


CMP6: CMP    AL, 0FDH

        JNE          CMP7

```

```

MOV    DX, OFFSET PCJR

RET

CMP7:  CMP    AL, 0F9H

JNE            CMP8

MOV    DX, OFFSET PC_CONVERTIBLE

RET

CMP8:  CALL    BYTE_TO_HEX

LEA            BX, PC_UNDEFINED

MOV        [BX+14], AX

MOV    DX, OFFSET PC_UNDEFINED

RET

DEFINE_PC_TYPE        ENDP

BEGIN:

CALL    GET_PC_NUMBER

MOV    DX, OFFSET PC_TYPE

CALL    WRITE_MSG

CALL    DEFINE_PC_TYPE

CALL    WRITE_MSG

XOR    AX, AX

MOV        AH, 30H

INT        21H

CALL    GET_SYS_VER

CALL    GET_OEM_NUM

```



```

CALL  GET_SERIAL_NUM

MOV      DX, OFFSET SYSTEM_VERSION

CALL  WRITE_MSG

MOV      DX, OFFSET OEM_NUMBER

CALL  WRITE_MSG

MOV      DX, OFFSET SERIAL_NUMBER

CALL  WRITE_MSG

XOR  AL,AL

MOV  AH,4Ch

INT  21H

TESTPC  ENDS

END  START

```