**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Операционные системы»**

Тема: **Исследование структур загрузочных модулей**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8383 |  | Костарев К.В. |
| Преподаватель |  | Ефремов М.А. |

Санкт-Петербург

2020

**Цель работы.**

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

**Выполнение работы.**

В первую очередь был написан текст исходного .COM модуля для программы, определяющей тип PC и версию системы. Для выполнения этой задачи ассемблерная программа читает содержимое предпоследнего байта ROM BIOS, и в соответствие с табл. 1 определяет тип PC и выводит строку с названием модели. Если код не совпадает ни с одним значением, то двоичный код переводится в символьную строку, содержащую запись шестнадцатеричного числа и выводится на экран.

Таблица 1 – Соответствие кода и типа PC

|  |  |
| --- | --- |
| Тип | Код |
| PC | FF |
| PC/XT | FE, FB |
| AT | FC |
| PS2 модель 30 | FA |
| PS2 модель 50 или 60 | FC |
| PS2 модель 80 | F8 |
| PC Convertiable | F9 |
| PCjr | FD |

Для решения задачи определения версии системы ассемблерная программа по значению регистров AL и AH, после выполнения функции 30H прерывания 21H, формирует текстовую строку в формате x.y, где x – номер основной версии, а y – номер модификации, числа в десятичной системе счисления. Также по значению регистров BH и BL:CX формируются строки со значением OEM и серийного номера пользователя.

Результатом линковки были получены «хороший» .COM и «плохой» .EXE модули.

Далее был написан текст исходного «хорошего» .EXE модуля, который выполняет те же задачи, что и .COM и «плохой» .EXE, с последующей линковкой.

Код исходных модулей представлен в Приложении А.

Отличия исходных текстов COM и EXE программ:

1. *Сколько сегментов должна содержать COM-программа?*

COM-программа содержит только один сегмент.

1. *EXE-программа?*

Тем временем EXE может содержать произвольное число сегментов.

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

ORG 100H, которая необходима для размещения PSP, и Assume для инициализации регистров.

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

В COM отсутствует таблица настроек, а следовательно для них некорректно указание адреса сегмента.

После этого в FAR были открыты файлы .COM, «плохого» и «хорошего» .EXE модулей в шестнадцатеричном виде с последующим их сравнением. Содержание файлов представлено в Приложении Б.

Отличия форматов файлов COM и EXE модулей:

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

Файл COM содержит только лишь сегмент с кодом и данными, в памяти код начинается с адреса 100H.

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

«Плохой» EXE тоже содержит только сегмент с кодом и данными, но начинается с адреса 300H. С адреса 0H располагается заголовок и таблица настроек.

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

В «плохом» EXE только один сегмент, и при этом файл занимает больше памяти, т.к. по умолчанию память до адреса 300H зарезервирована под таблицу релокации. В начале файлов информация отличается.

Далее в отладчике TD был загружен .COM файл, представленный на рис. 1.

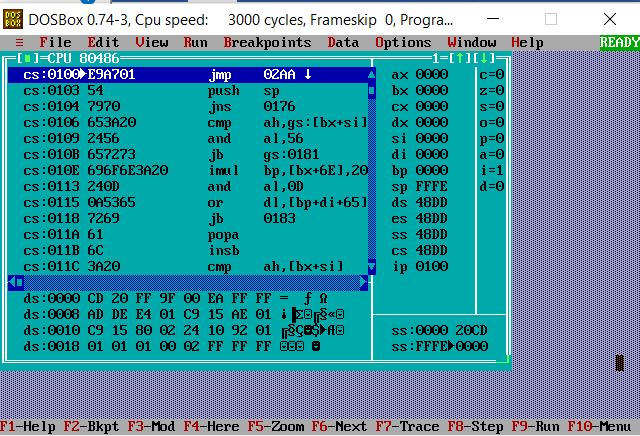


Рисунок 1 – COM файл, запущенный в отладчике TD

Загрузка COM модуля в основную память:

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

Код и данные располагаются с адреса 100H.

1. *Что располагается с адреса 0?*

При загрузке ОС с адреса 0H располагается PSP.

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

При запуске COM файла все сегментные регистры содержат адрес PSP.

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

Стек занимает всю доступную память вместе с кодом и данными. При загрузке в SP записывается FFFE, а в BP 0000.

Также в отладчике TD был загружен и «хороший» EXE файл, который представлен на рис. 2.

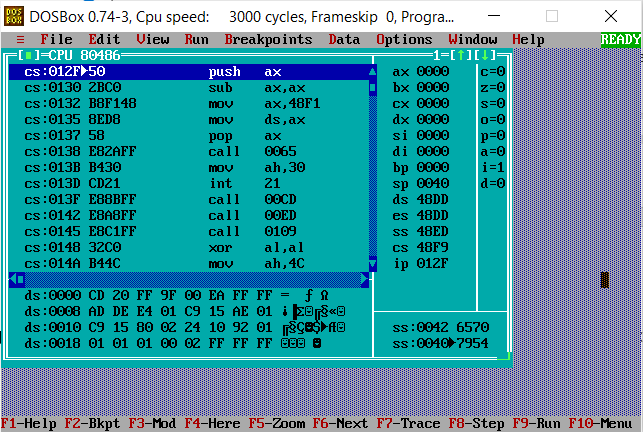


Рисунок 2 – «хороший» EXE файл, запущенный в отладчике TD

Загрузка «хорошего» EXE модуля в основную память:

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

В сегментные регистры CS и SS записываются адреса начала соответствующего сегмента при загрузке, а в DS и ES адрес начала PSP.

1. *На что указывают регистры DS и ES?*

DS и ES указывают на начало PSP.

1. *Как определяется стек?*

Память под стек выделяется в соответствии с моделью памяти в программе, если только это не прописано в самой программе. При ее загрузке инициализируются регистры BP и SP.

1. *Как определяется точка входа?*

Точкой входа является начало сегмента кода CS, если она не указана явно. В программе точка входа указывается в конце директивой END <точка входа>.

**Выводы.**

В ходе выполнения данной лабораторной работы были исследованы различия в структурах исходных текстов модулей типов .COM и .EXE, структур файлов загрузочных модулей и способов их загрузки в основную память. Для исследования были написаны две программы для COM и EXE модулей, решающие одну и ту же задачу определения типа PC, версии и серийных номеров OEM и пользователя.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД РЕАЛИЗОВАННЫХ ПРОГРАММ

COM модуль:

TESTPC SEGMENT

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

ORG 100H

START: JMP BEGIN

TYPE\_PC db 'Type: ', '$'

VERSION db 'Version: ', '$'

SERIAL db 0DH, 0AH, 'Serial: ', '$'

OEM db 0DH, 0AH, 'OEM: ', '$'

NEW\_STR db " $"

NEW\_STR2 db " $"

STR\_PC db 'PC',0DH,0AH,'$'

STR\_PCXT db 'PC/XT',0DH,0AH,'$'

STR\_AT db 'AT',0DH,0AH,'$'

STR\_PS230 db 'PS2 model 30',0DH,0AH,'$'

STR\_PS280 db 'PS2 model 80',0DH,0AH,'$'

STR\_PCJR db 'PCjr',0DH,0AH,'$'

STR\_PCCO db 'PC 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

push CX

mov AH,AL

call TETR\_TO\_HEX

xchg AL,AH

mov CL,4

shr AL,CL

call TETR\_TO\_HEX

pop CX

ret

BYTE\_TO\_HEX ENDP

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

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

WRITE\_NUMBER PROC near

push AX

mov AH, 02H

int 21H

pop AX

ret

WRITE\_NUMBER ENDP

WRITE\_STRING PROC near

push AX

mov AH, 09H

int 21H

pop AX

ret

WRITE\_STRING ENDP

WRITE\_TYPE PROC near

push AX

push BX

push DX

push ES

mov DX,offset TYPE\_PC

call WRITE\_STRING

mov BX,0F000H

mov ES,BX

mov AL,ES:[0FFFEH]

call BYTE\_TO\_HEX

cmp AX, 0FFH

je printPC

cmp AX, 0FEH

je printPCXT

cmp AX, 0FCH

je printAT

cmp AX, 0FAH

je printPS230

cmp AX, 0F8H

je printPS280

cmp AX, 0FDH

je printPCjr

cmp AX, 0F9H

je printPCCo

printPC:

mov DX,offset STR\_PC

jmp printSTR

printPCXT:

mov DX,offset STR\_PCXT

jmp printSTR

printAT:

mov DX,offset STR\_AT

jmp printSTR

printPS230:

mov DX,offset STR\_PS230

jmp printSTR

printPS280:

mov DX,offset STR\_PS280

jmp printSTR

printPCjr:

mov DX,offset STR\_PCJR

jmp printSTR

printPCCo:

mov DX,offset STR\_PCCO

printSTR:

call WRITE\_STRING

pop ES

pop DX

pop BX

pop AX

ret

WRITE\_TYPE ENDP

WRITE\_VERSION PROC near

push AX

push DX

mov DX, offset VERSION

call WRITE\_STRING

mov DL, AL

add DL, '0'

call WRITE\_NUMBER

mov DL, '.'

call WRITE\_NUMBER

mov DL, AH

add DL, '0'

call WRITE\_NUMBER

pop DX

pop AX

ret

WRITE\_VERSION ENDP

WRITE\_OEM PROC near

push AX

push DX

mov DX, offset OEM

call WRITE\_STRING

mov SI, offset NEW\_STR

add SI, 2

mov AL, BH

call BYTE\_TO\_DEC

mov DX, offset NEW\_STR

call WRITE\_STRING

pop DX

pop AX

ret

WRITE\_OEM ENDP

WRITE\_SERIAL PROC near

push AX

push DX

mov DX, offset SERIAL

call WRITE\_STRING

mov DI, offset NEW\_STR2

add DI, 6

mov AX, CX

call WRD\_TO\_HEX

mov AL, BL

call BYTE\_TO\_HEX

sub DI, 2

mov [DI], AX

mov dx, offset NEW\_STR2

call WRITE\_STRING

pop DX

pop AX

ret

WRITE\_SERIAL ENDP

BEGIN:

call WRITE\_TYPE

mov AH, 30H

int 21H

call WRITE\_VERSION

call WRITE\_OEM

call WRITE\_SERIAL

xor AL,AL

mov AH,4Ch

int 21H

TESTPC ENDS

END START

EXE модуль:

AStack SEGMENT STACK

DW 20h DUP(?)

AStack ENDS

DATA SEGMENT

TYPE\_PC db 'Type: ', '$'

VERSION db 'Version: ', '$'

SERIAL db 0DH, 0AH, 'Serial: ', '$'

OEM db 0DH, 0AH, 'OEM: ', '$'

NEW\_STR db " $"

NEW\_STR2 db " $"

STR\_PC db 'PC',0DH,0AH,'$'

STR\_PCXT db 'PC/XT',0DH,0AH,'$'

STR\_AT db 'AT',0DH,0AH,'$'

STR\_PS230 db 'PS2 model 30',0DH,0AH,'$'

STR\_PS280 db 'PS2 model 80',0DH,0AH,'$'

STR\_PCJR db 'PCjr',0DH,0AH,'$'

STR\_PCCO db 'PC Convertible',0DH,0AH,'$'

DATA ENDS

CODE SEGMENT

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

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

push CX

mov AH,AL

call TETR\_TO\_HEX

xchg AL,AH

mov CL,4

shr AL,CL

call TETR\_TO\_HEX

pop CX

ret

BYTE\_TO\_HEX ENDP

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

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

WRITE\_NUMBER PROC near

push AX

mov AH, 02H

int 21H

pop AX

ret

WRITE\_NUMBER ENDP

WRITE\_STRING PROC near

push AX

mov AH, 09H

int 21H

pop AX

ret

WRITE\_STRING ENDP

WRITE\_TYPE PROC near

push AX

push BX

push DX

push ES

mov DX,offset TYPE\_PC

call WRITE\_STRING

mov BX,0F000H

mov ES,BX

mov AL,ES:[0FFFEH]

call BYTE\_TO\_HEX

cmp AX, 0FFH

je printPC

cmp AX, 0FEH

je printPCXT

cmp AX, 0FCH

je printAT

cmp AX, 0FAH

je printPS230

cmp AX, 0F8H

je printPS280

cmp AX, 0FDH

je printPCjr

cmp AX, 0F9H

je printPCCo

printPC:

mov DX,offset STR\_PC

jmp printSTR

printPCXT:

mov DX,offset STR\_PCXT

jmp printSTR

printAT:

mov DX,offset STR\_AT

jmp printSTR

printPS230:

mov DX,offset STR\_PS230

jmp printSTR

printPS280:

mov DX,offset STR\_PS280

jmp printSTR

printPCjr:

mov DX,offset STR\_PCJR

jmp printSTR

printPCCo:

mov DX,offset STR\_PCCO

printSTR:

call WRITE\_STRING

pop ES

pop DX

pop BX

pop AX

ret

WRITE\_TYPE ENDP

WRITE\_VERSION PROC near

push AX

push DX

mov DX, offset VERSION

call WRITE\_STRING

mov DL, AL

add DL, '0'

call WRITE\_NUMBER

mov DL, '.'

call WRITE\_NUMBER

mov DL, AH

add DL, '0'

call WRITE\_NUMBER

pop DX

pop AX

ret

WRITE\_VERSION ENDP

WRITE\_OEM PROC near

push AX

push DX

mov DX, offset OEM

call WRITE\_STRING

mov SI, offset NEW\_STR

add SI, 2

mov AL, BH

call BYTE\_TO\_DEC

mov DX, offset NEW\_STR

call WRITE\_STRING

pop DX

pop AX

ret

WRITE\_OEM ENDP

WRITE\_SERIAL PROC near

push AX

push DX

mov DX, offset SERIAL

call WRITE\_STRING

mov DI, offset NEW\_STR2

add DI, 6

mov AX, CX

call WRD\_TO\_HEX

mov AL, BL

call BYTE\_TO\_HEX

sub DI, 2

mov [DI], AX

mov dx, offset NEW\_STR2

call WRITE\_STRING

pop DX

pop AX

ret

WRITE\_SERIAL ENDP

MAIN PROC FAR

push AX

sub AX,AX

mov AX,DATA

mov DS,AX

pop AX

call WRITE\_TYPE

mov AH, 30H

int 21H

call WRITE\_VERSION

call WRITE\_OEM

call WRITE\_SERIAL

xor AL,AL

mov AH,4Ch

int 21H

MAIN ENDP

CODE ENDS

END MAIN

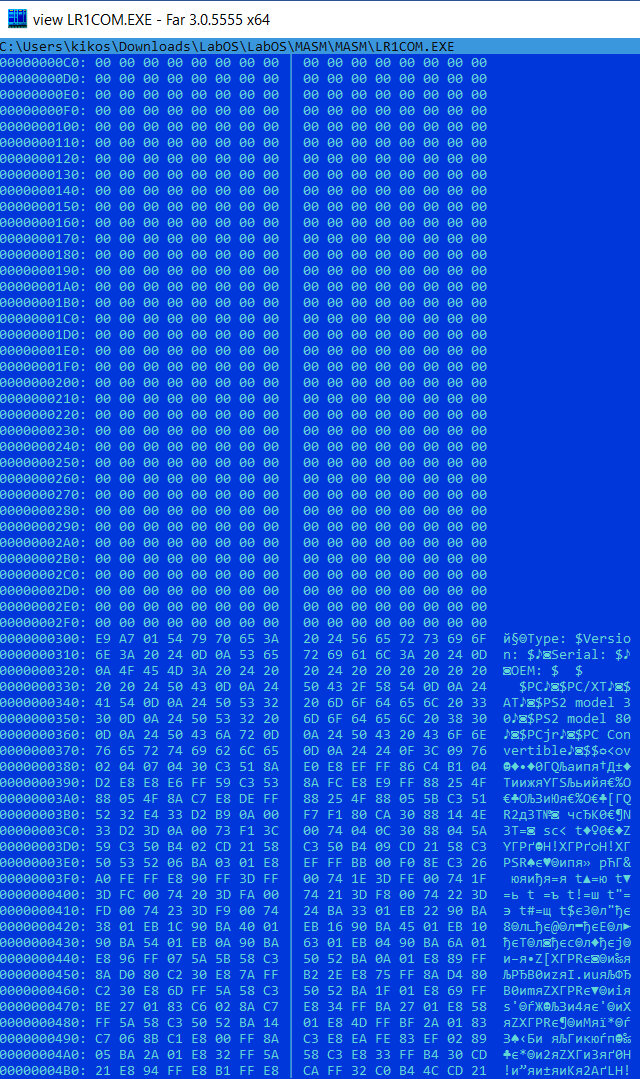
**ПРИЛОЖЕНИЕ Б**

**СОДЕРЖАНИЕ ФАЙЛОВ ПРОГРАММ В ШЕСТНАДЦАТЕРИЧНОМ ВИДЕ**

.COM:



«Плохой» .EXE:



«Хороший» .EXE:

