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 загрузочных модулей

Студент гр. 8381 _____ Нгуен Ш. Х.

Преподаватель _____ Ефмиров 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.

Процедуры, используемые в работе.

Название процедуры	Описание процедуры
OS_VER_PROC	Процедура распознавания и вывода версии ОС
PC_TYPE_PROC	Процедура распознавания и вывода типа ПК
WRITE	Процедура вывода содержимого по смещению DX
WRITE_OS_VERSION	Процедура вывода версии ОС
WRITE_DEC	Процедура вывода слова AX в 10-иричной с.с.
WRITE_SERIAL	Процедура вывода серийного номера пользователя
WRITE_HEX_BYTE	Процедура вывода байта AL в 16-иричной с.с.
WRITE_DEC_BYTE	Процедура вывода байта AL в 10-иричной с.с. в формате, требуемым заданием

Таблица 1 - Процедуры, используемые в работе

Ход работы.

Был написан текст исходного .COM модуля, который определяет тип ПК и версию системы. Код программы представлен в приложении А. После написания текста, из него были построены **.COM** модуль (результат выполнения которого показан на рис.

1) а также «плохой» **.EXE** модуль (результат выполнения показан на рис. 2). В ходе линковки **.EXE** модуля LINK-ер выдал предупреждении об отсутствии стека (см. рис. 3).

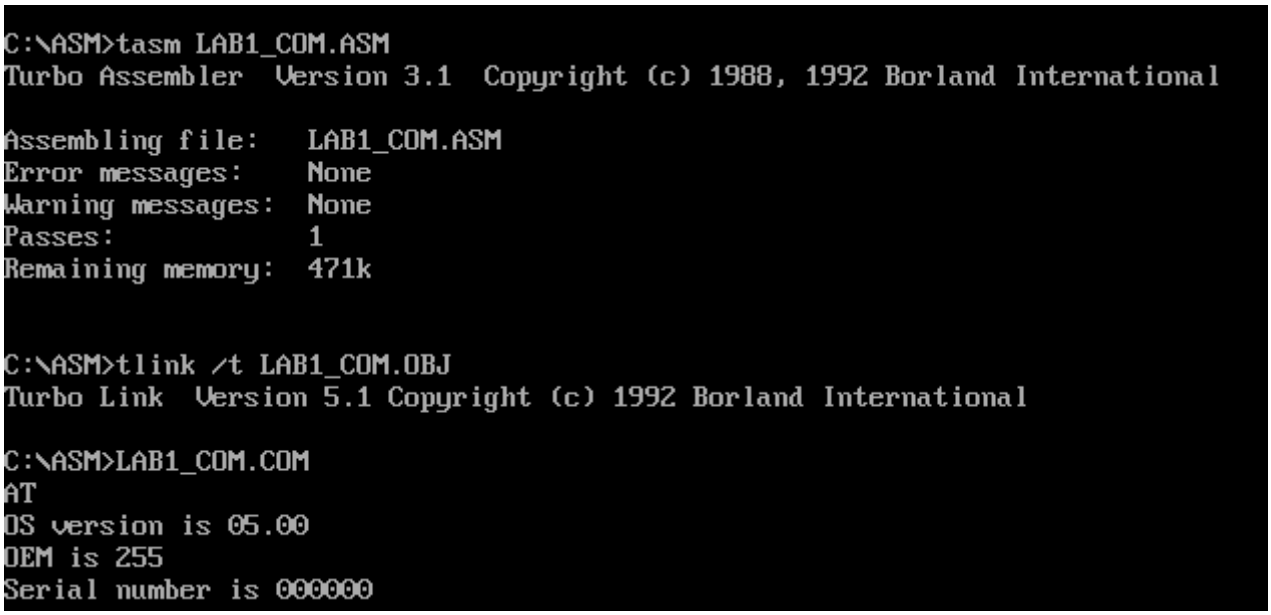

Рисунок 1 – Результат выполнения **.COM** модуля

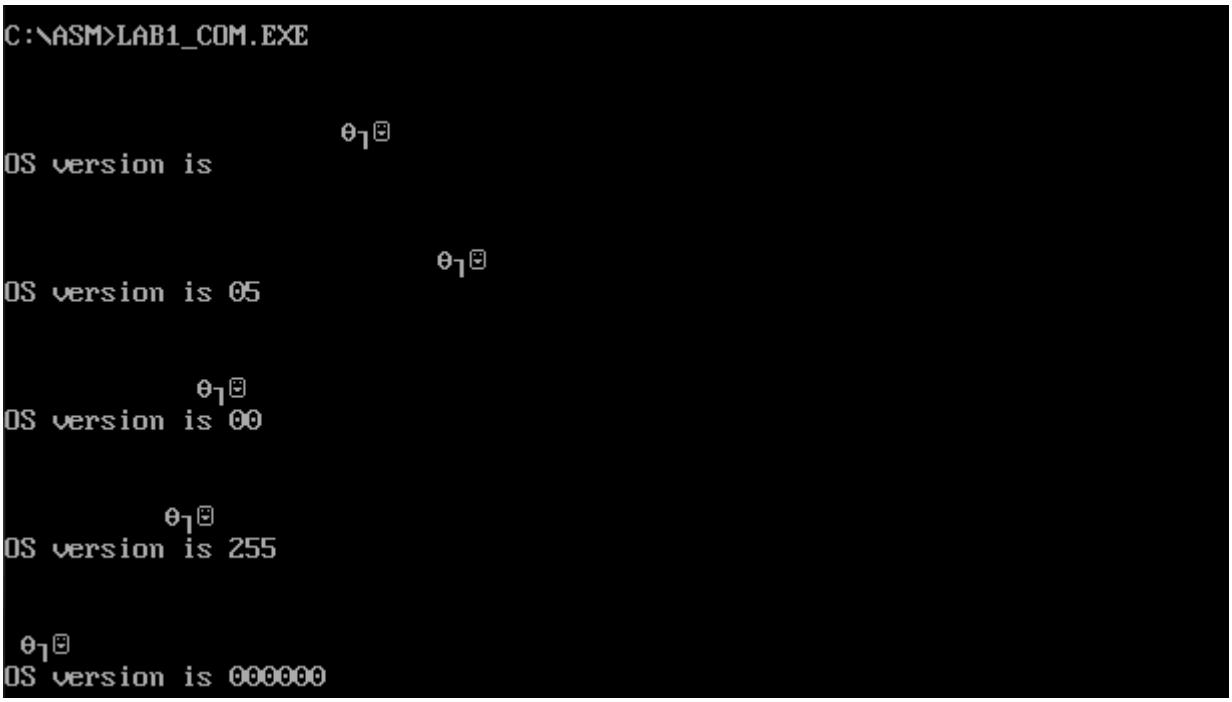

Рисунок 2 – Результат выполнения «плохого» **.EXE** модуля

Рисунок 3 – Предупреждение во время линковки

Был написан текст для «хорошего» **.EXE** модуля. Код программы представлен в приложении Б. После написания текста, из него был построен **.EXE** модуль (результат выполнения которого показан на рис. 4).

Рисунок 4 – Результат выполнения «хорошего» **.EXE** модуля

Отличия исходных текстов COM и EXE программ

1) Сколько сегментов должна содержать COM-программа?

COM программа должна содержать ровно один сегмент.

2) EXE-программа?

EXE программа может содержать сколько угодно сегментов (не меньше одного).

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

Обязательная должна быть диркетива ORG 100h. Она нужна по той причине, что при загрузке модуля в ОП в начале COM-программы определяется 256-байтовый (100h) префикс программного сегмента, так что адресация имеет смещение в 256 байт от нулевого адреса. Это смещение мы и задаём директивой ORG 100h (в отличие от EXE-программы, где PSP расположен вне кодового сегмента и, следовательно, явно определять смещение там не нужно).

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

В COM программах нельзя использовать команды вида *mov <регистр> <сегмент>,* так как в этих программах используется только один сегмент и надобности в этих командах нет. Также запрещены 64-битные команды.

При помощи приложения Far были открыты все созданные файлы загрузочных модулей в шестнадцатеричном виде. Результаты представлены на рис. 5 – 7.

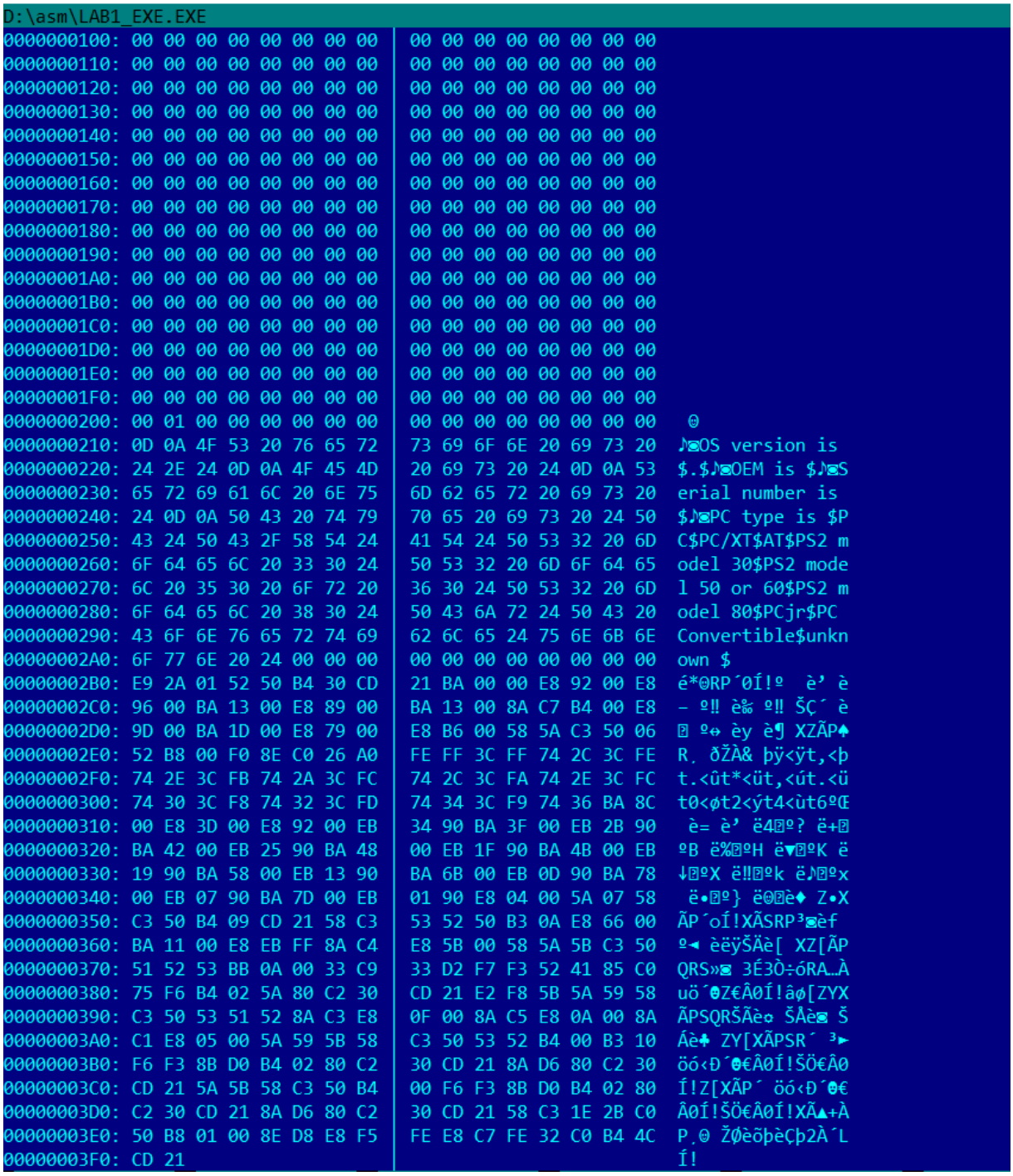

Рисунок 5 – Содержимое файла «хорошего» EXE модуля

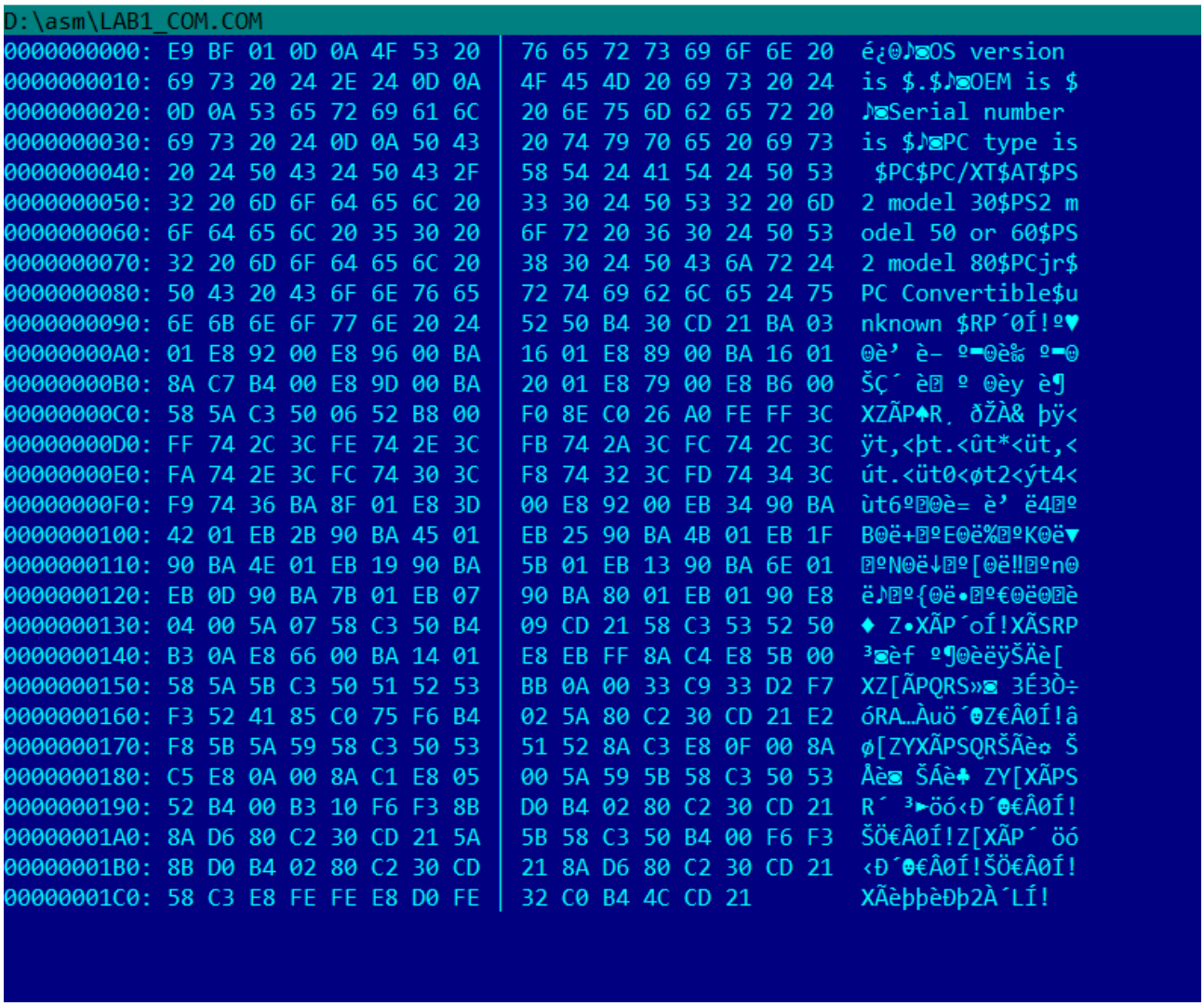

Рисунок 6 – Содержисое файла COM модуля

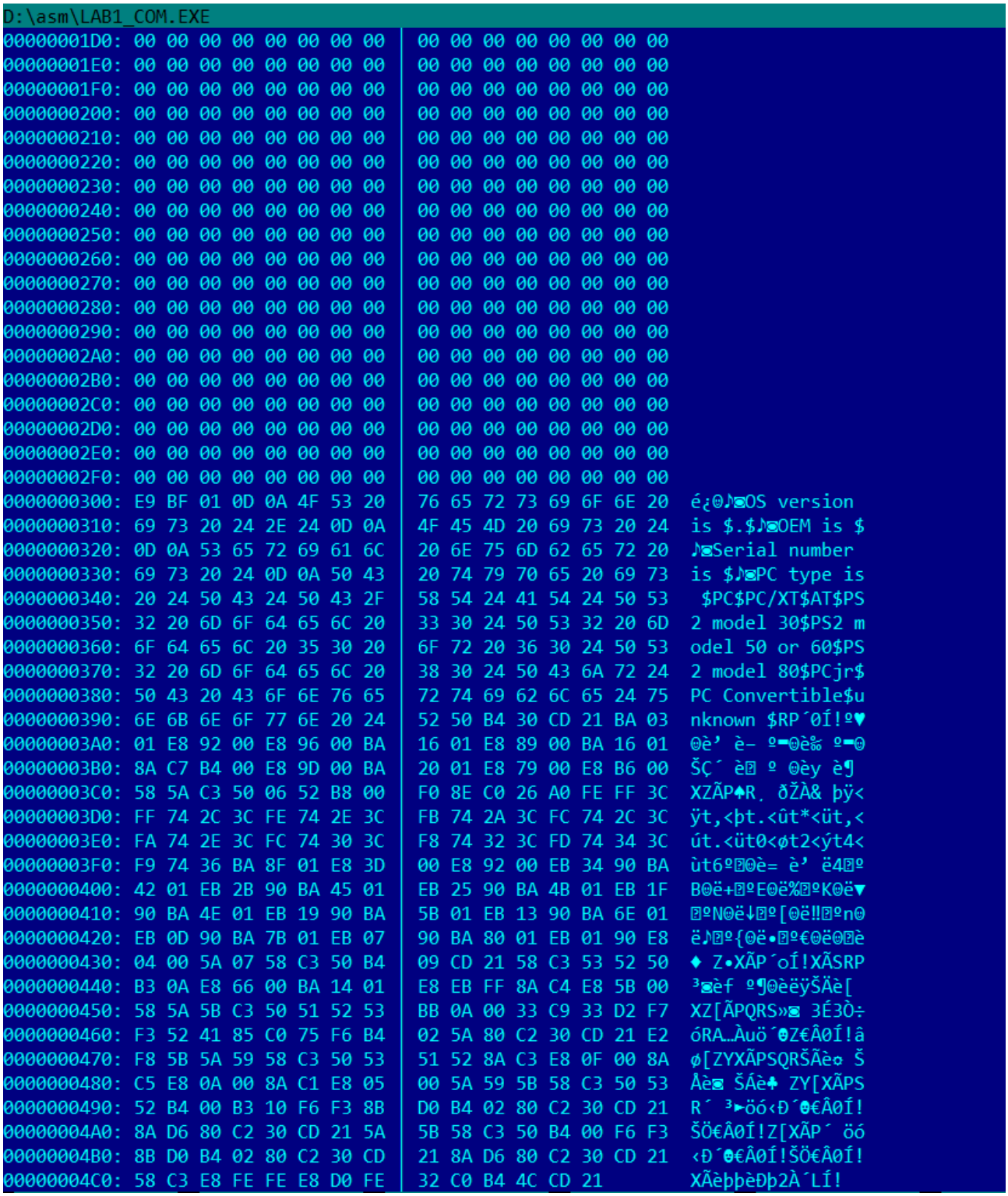

Рисунок 7 – Содержимое файла «плохого» EXE модуля

Отличие форматов файлов COM и EXE модулей

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

COM файл содержит только машинный код и данные программы. Код начинается с адреса 0h, но при загрузке происходит смещение на 100h.

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

В «плохом» EXE модуле машинный код и данные содержатся в одном сегменте. С адреса 0h идёт таблица настроек (Relocation table). Код располагается с адреса 300h.

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

В «хорошем» EXE модуле данные, стэк и машинный код находятся в разных сегментах. От «плохого» EXE он так же отличается наличием стека.

При помощи отладчика TD COM файл был загружен в основную память. Результат продемонстрирован на рис. 8.

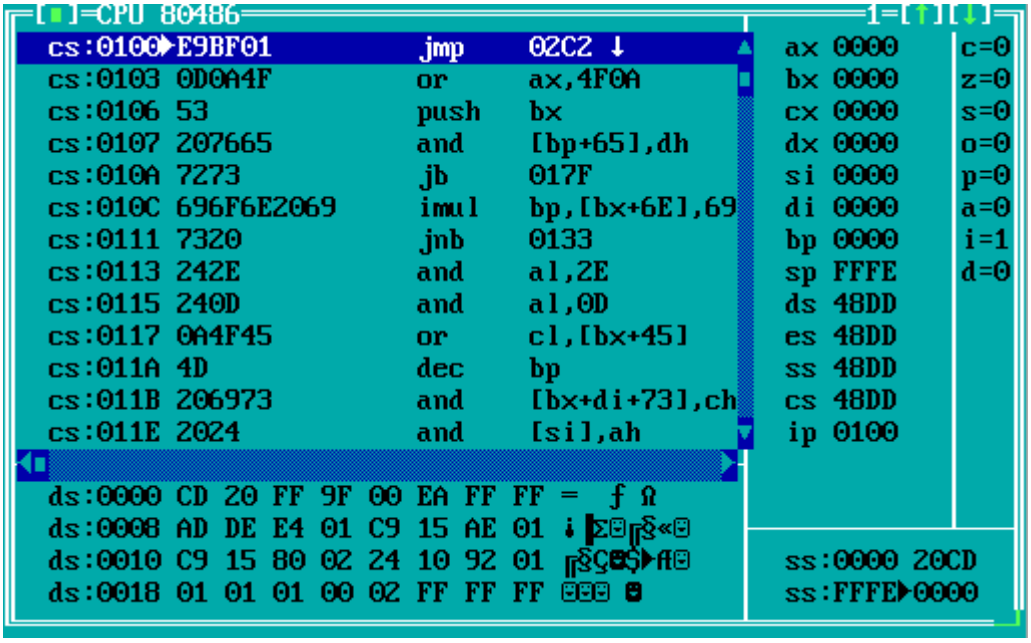

Рисунок 8 – Результат загрузки COM файла в память

Загрузка COM модуля в основную память

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

Определяется сегментный адрес свободного участка ОП, в который можно загрузить программу. Создается блок памяти. В поля PSP заносятся значения. Загружается COM файл со смещением 100h. Сегментные регистры устанавливаются на адрес сегмента PSP, регистр SP указывает на конец сегмента, туда записывается 0000h. С ростом стека значение SP будет уменьшаться. Счетчик команд принимает значение 100h. Программа запускается.

2) **Что располагается с адреса 0?**

Со смещения 0h в программном сегменте начинается PSP.

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

Сегментные регистры указывают на начало адрес сегмента PSP.

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

Регистр SP указывает на конец стека (FFFFh), SS – на начало (0h). Адреса расположены в диапазоне 0h – FFFEh (FFFEh, т.к. это последний адрес, кратный двум).

При помощи отладчика в основную память был записан так же «хороший» EXE файл. Результат продемонстрирован на рис. 9.

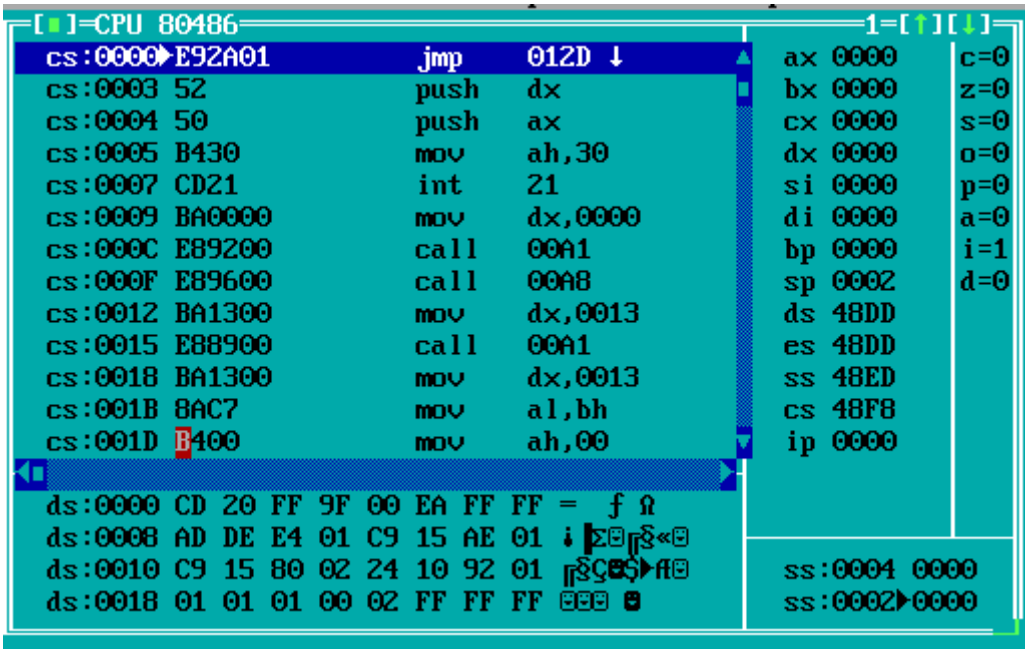

Рисунок 9 – Результат загрузки EXE файла в память

Загрузка «хорошего» EXE модуля в основную память

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

Для PSP и программы выделяется блок памяти. После запуска программы DS и ES указывают на начало PSP (48DDh), CS – на начало сегмента команд (4919h), а SS – на начало сегмента стека (48EDh). IP имеет ненулевое значение, так как в программе есть дополнительные процедуры, расположенные до основной.

2) **На что указывают регистры DS и ES?**

Регистры ES и DS указывают на начало PSP.

3) **Как определяется стек?**

Стек определяется при помощи директивы ASSUME CS:MYSTACK, где MYSTACK – сегмент, отведенный под стек:

```
MYSTACK SEGMENT STACK
    DW 100H
MYSTACK ENDS
```

4) Как определяется точка входа?

Смещение точки входа в программу загружается в указатель команд IP и определяется операндом директивы END, который называется точкой входа. Операндом является функция или метка, с которой необходимо начать программу.

Выводы.

В ходе выполнения 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.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ ДЛЯ COM ФАЙЛА

```
LAB1  SEGMENT
            ASSUME  CS:LAB1, DS:LAB1, ES:NOTHING, SS:NOTHING
            ORG 100H
START: JMP BEGIN
;DATA
OS_VERSION_NUMBER DB 13, 10, "OS VERSION IS $"
DOT DB ".$"
OEM DB 13, 10, "OEM IS $"
SERIAL DB 13, 10, "SERIAL NUMBER IS $"
TYPE_PC_STR DB 13, 10, "PC TYPE IS $"
TYPE_PC DB "PC$"
TYPE_PC_XT DB "PC/XT$"
TYPE_AT DB "AT$"
TYPE_PS2_M30 DB "PS2 MODEL 30$"
TYPE_PS2_M5060 DB "PS2 MODEL 50 OR 60$"
TYPE_PS2M80 DB "PS2 MODEL 80$"
TYPE_PC_JR DB "PCJR$"
TYPE_PC_CONV DB "PC CONVERTIBLE$"
TYPE_UNKNOWN DB "UNKNOWN $"
;PROCEDURES
;_____________________________________________________________
OS_VER_PROC PROC
      PUSH DX
      PUSH AX
      MOV AH, 30H ;GETTING INFO
      INT 21H

      MOV DX, OFFSET OS_VERSION_NUMBER
      CALL WRITE
      CALL WRITE_OS_VERSION
      MOV DX, OFFSET OEM
      CALL WRITE
      MOV AL, BH
      MOV AH, 0
      CALL WRITE_DEC
      MOV DX, OFFSET SERIAL
      CALL WRITE
      CALL WRITE_SERIAL
```

```
      POP AX
      POP DX
      RET
OS_VER_PROC ENDP
;_____________________________________________________________________
PC_TYPE_PROC PROC
      PUSH AX
      PUSH ES
      PUSH DX

      MOV AX, 0F000H
      MOV ES, AX
      MOV AL, ES:[0FFFEH]
      CMP AL, 0FFH
      JE PC
      CMP AL, 0FEH
      JE XT
      CMP AL, 0FBH
      JE XT
      CMP AL, 0FCH
      JE PCAT
      CMP AL, 0FAH
      JE PS30
      CMP AL, 0FCH;
      JE PS50
      CMP AL, 0F8H
      JE PS80
      CMP AL, 0FDH
      JE JR
      CMP AL, 0F9H
      JE CONV
      ;DEFOULT
      MOV DX, OFFSET TYPE_UNKNOWN
      CALL WRITE
      CALL WRITE_HEX_BYTE
      JMP FINISH
PC:
      MOV DX, OFFSET TYPE_PC
      JMP RES
XT:
```

```
      MOV DX, OFFSET TYPE_PC_XT
      JMP RES
PCAT:
      MOV DX, OFFSET TYPE_AT
      JMP RES
PS30:
      MOV DX, OFFSET TYPE_PS2_M30
      JMP RES
PS50:
      MOV DX, OFFSET TYPE_PS2_M5060
      JMP RES
PS80:
      MOV DX, OFFSET TYPE_PS2M80
      JMP RES
JR:
      MOV DX, OFFSET TYPE_PC_JR
      JMP RES
CONV:
      MOV DX, OFFSET TYPE_PC_CONV
      JMP RES
RES:
      CALL WRITE

FINISH:
      POP DX
      POP ES
      POP AX
      RET
PC_TYPE_PROC ENDP
;_____________________________________________________________________
WRITE PROC
      PUSH AX
      MOV AH, 9H
      INT 21H
      POP AX
      RET
WRITE ENDP
;_____________________________________________________________________
WRITE_OS_VERSION PROC
      PUSH BX
      PUSH DX
```

```
        PUSH AX

        MOV BL, 10
        ;VER
        CALL WRITE_DEC_BYTE
        ;DOT
        MOV DX, OFFSET DOT
        CALL WRITE
        ;MOD
        MOV AL, AH
        CALL WRITE_DEC_BYTE

        POP AX
        POP DX
        POP BX
        RET
WRITE_OS_VERSION ENDP
;_____________________________________________________________________
WRITE_DEC PROC
        PUSH AX
        PUSH CX
        PUSH DX
        PUSH BX

        MOV BX, 10
        XOR CX, CX
GETTING_NUMS:
        XOR DX, DX
        DIV BX
        PUSH DX
        INC CX
        TEST AX, AX
        JNZ GETTING_NUMS
        MOV AH, 02H
WRITING:
        POP DX
        ADD DL, '0'
        INT 21H
        LOOP WRITING

        POP BX
```

```
        POP DX
        POP CX
        POP AX
        RET
WRITE_DEC ENDP
;_____________________________________________________________________
WRITE_SERIAL PROC
        PUSH AX
        PUSH BX
        PUSH CX
        PUSH DX

        ;FIRST BYTE
        MOV AL, BL
        CALL WRITE_HEX_BYTE
        ;SECOND BYTE
        MOV AL, CH
        CALL WRITE_HEX_BYTE
        ;THIRD BYTE
        MOV AL, CL
        CALL WRITE_HEX_BYTE

        POP DX
        POP CX
        POP BX
        POP AX
        RET
WRITE_SERIAL ENDP
;_____________________________________________________________________
WRITE_HEX_BYTE PROC
        PUSH AX
        PUSH BX
        PUSH DX

        MOV AH, 0
        MOV BL, 16
        DIV BL
        MOV DX, AX
        MOV AH, 02H
        ADD DL, '0'
        INT 21H;
```

```
            MOV DL, DH
            ADD DL, '0'
            INT 21H;

            POP DX
            POP BX
            POP AX
            RET
      WRITE_HEX_BYTE ENDP
      ;_____________________________________________________________________
      WRITE_DEC_BYTE PROC
            PUSH AX
            MOV AH, 0
            DIV BL
            MOV DX, AX
            MOV AH, 02H
            ADD DL, '0'
            INT 21H
            MOV DL, DH
            ADD DL, '0'
            INT 21H
            POP AX
            RET
      WRITE_DEC_BYTE ENDP
      ;_____________________________________________________________________

      BEGIN:
            CALL PC_TYPE_PROC
            CALL OS_VER_PROC

            ;TO DOS
            XOR AL, AL
            MOV AH, 4CH
            INT 21H
      LAB1 ENDS
END START
```

ПРИЛОЖЕНИЕ Б

КОД ПРОГРАММЫ ДЛЯ EXE ФАЙЛА

```
MYSTACK SEGMENT STACK
	DW 100H
MYSTACK ENDS

DATA SEGMENT
	;DATA
	OS_VERSION_NUMBER DB 13, 10, "OS VERSION IS $"
	DOT DB ".$"
	OEM DB 13, 10, "OEM IS $"
	SERIAL DB 13, 10, "SERIAL NUMBER IS $"
	TYPE_PC_STR DB 13, 10, "PC TYPE IS $"
	TYPE_PC DB "PC$"
	TYPE_PC_XT DB "PC/XT$"
	TYPE_AT DB "AT$"
	TYPE_PS2_M30 DB "PS2 MODEL 30$"
	TYPE_PS2_M5060 DB "PS2 MODEL 50 OR 60$"
	TYPE_PS2M80 DB "PS2 MODEL 80$"
	TYPE_PC_JR DB "PCJR$"
	TYPE_PC_CONV DB "PC CONVERTIBLE$"
	TYPE_UNKNOWN DB "UNKNOWN $"
DATA ENDS

LAB1  SEGMENT
		ASSUME  CS:LAB1, DS:DATA, ES:NOTHING, SS:MYSTACK
START: JMP BEGIN

;PROCEDURES
;_____________________________________________________________________
OS_VER_PROC PROC
	PUSH DX
	PUSH AX
	MOV AH, 30H ;GETTING INFO
	INT 21H

	MOV DX, OFFSET OS_VERSION_NUMBER
	CALL WRITE
	CALL WRITE_OS_VERSION
	MOV DX, OFFSET OEM
```

```
        CALL WRITE
        MOV AL, BH
        MOV AH, 0
        CALL WRITE_DEC
        MOV DX, OFFSET SERIAL
        CALL WRITE
        CALL WRITE_SERIAL

        POP AX
        POP DX
        RET
OS_VER_PROC ENDP
;_____________________________________________________________________
PC_TYPE_PROC PROC
        PUSH AX
        PUSH ES
        PUSH DX

        MOV AX, 0F000H
        MOV ES, AX
        MOV AL, ES:[0FFFEH]
        CMP AL, 0FFH
        JE PC
        CMP AL, 0FEH
        JE XT
        CMP AL, 0FBH
        JE XT
        CMP AL, 0FCH
        JE PCAT
        CMP AL, 0FAH
        JE PS30
        CMP AL, 0FCH;ОПЕЧАТКА В МЕТОДЕ
        JE PS50
        CMP AL, 0F8H
        JE PS80
        CMP AL, 0FDH
        JE JR
        CMP AL, 0F9H
        JE CONV
        ;DEFOULT
        MOV DX, OFFSET TYPE_UNKNOWN
```

```
      CALL WRITE
      CALL WRITE_HEX_BYTE
      JMP FINISH
PC:
      MOV DX, OFFSET TYPE_PC
      JMP RES
XT:
      MOV DX, OFFSET TYPE_PC_XT
      JMP RES
PCAT:
      MOV DX, OFFSET TYPE_AT
      JMP RES
PS30:
      MOV DX, OFFSET TYPE_PS2_M30
      JMP RES
PS50:
      MOV DX, OFFSET TYPE_PS2_M5060
      JMP RES
PS80:
      MOV DX, OFFSET TYPE_PS2M80
      JMP RES
JR:
      MOV DX, OFFSET TYPE_PC_JR
      JMP RES
CONV:
      MOV DX, OFFSET TYPE_PC_CONV
      JMP RES
RES:
      CALL WRITE

FINISH:
      POP DX
      POP ES
      POP AX
      RET
PC_TYPE_PROC ENDP
;_____________________________________________________________________
WRITE PROC
      PUSH AX
      MOV AH, 9H
      INT 21H
```

```
	POP AX
	RET
WRITE ENDP
;_____________________________________________________________________
WRITE_OS_VERSION PROC
	PUSH BX
	PUSH DX
	PUSH AX

	MOV BL, 10
	;VER
	CALL WRITE_DEC_BYTE
	;DOT
	MOV DX, OFFSET DOT
	CALL WRITE
	;MOD
	MOV AL, AH
	CALL WRITE_DEC_BYTE

	POP AX
	POP DX
	POP BX
	RET
WRITE_OS_VERSION ENDP
;_____________________________________________________________________
WRITE_DEC PROC
	PUSH AX
	PUSH CX
	PUSH DX
	PUSH BX

	MOV BX, 10
	XOR CX, CX
GETTING_NUMS:
	XOR DX, DX
	DIV BX
	PUSH DX
	INC CX
	TEST AX, AX
	JNZ GETTING_NUMS
	MOV AH, 02H
```

```
WRITING:
	POP DX
	ADD DL, '0'
	INT 21H
	LOOP WRITING

	POP BX
	POP DX
	POP CX
	POP AX
	RET
WRITE_DEC ENDP
;_____________________________________________________________
WRITE_SERIAL PROC
	PUSH AX
	PUSH BX
	PUSH CX
	PUSH DX

	;FIRST BYTE
	MOV AL, BL
	CALL WRITE_HEX_BYTE
	;SECOND BYTE
	MOV AL, CH
	CALL WRITE_HEX_BYTE
	;THIRD BYTE
	MOV AL, CL
	CALL WRITE_HEX_BYTE

	POP DX
	POP CX
	POP BX
	POP AX
	RET
WRITE_SERIAL ENDP
;_____________________________________________________________
WRITE_HEX_BYTE PROC
	PUSH AX
	PUSH BX
	PUSH DX
```

```
      MOV AH, 0
      MOV BL, 16
      DIV BL
      MOV DX, AX
      MOV AH, 02H
      ADD DL, '0'
      INT 21H;
      MOV DL, DH
      ADD DL, '0'
      INT 21H;

      POP DX
      POP BX
      POP AX
      RET
WRITE_HEX_BYTE ENDP
;_____________________________________________________________________
WRITE_DEC_BYTE PROC
      PUSH AX
      MOV AH, 0
      DIV BL
      MOV DX, AX
      MOV AH, 02H
      ADD DL, '0'
      INT 21H
      MOV DL, DH
      ADD DL, '0'
      INT 21H
      POP AX
      RET
WRITE_DEC_BYTE ENDP
;_____________________________________________________________________

BEGIN:
      PUSH  DS
    SUB   AX,AX
    PUSH  AX
    MOV   AX,DATA
    MOV   DS,AX

      CALL PC_TYPE_PROC
```

```
            CALL OS_VER_PROC

            ;TO DOS
            XOR AL, AL
            MOV AH, 4CH
            INT 21H
      LAB1 ENDS
END START
```