

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего профессионального образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА



Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

ОТЧЁТ

по лабораторной работе №2

"Разработка программы ввода-вывода и обработки последовательности кодов  
на ассемблере "

по дисциплине

**Принципы и методы организации системных программных средств**

(наименование дисциплины)

РУКОВОДИТЕЛЬ:

\_\_\_\_\_  
(подпись)

Викулова Е.Н.  
(фамилия, и.,о.)

СТУДЕНТ:

\_\_\_\_\_  
(подпись)

Халеев А.А.  
(фамилия, и.,о.)

21-ВМз-4  
(шифр группы)

Работа защищена « \_\_\_\_ » \_\_\_\_\_

С оценкой \_\_\_\_\_

Нижний Новгород

2023

## ЦЕЛЬ РАБОТЫ:

Приобретение навыков: разработки *одно-* и *много*сегментных программ на языке ассемблер, использования функций прерываний для организации ввода-вывода, управление трансляцией и компоновкой.

### Вариант №8

Тип: W

Коды, не равные значениям:

$c1 = '01' \sim 3031h$ ;  $c2 = '23' \sim 3233h$ ;  $c3 = '45' \sim 3435h$

### Часть 1:

Написать программу на ассемблере, осуществляющую ввод последовательности символов с клавиатуры, обработку кодов символов в соответствии с заданием и вывод на экран результирующей последовательности.

#### Программа должна содержать:

- ввод последовательности символов с клавиатуры, предваряемый соответствующим текстовым сообщением (в результате ввода формируется статический массив кодов символов, максимальное число вводимых символов выбирается самостоятельно);
- обработку кодов символов в соответствии с заданием, вариант задания, определяющий условия пересылки байтов или слов из входного массива в выходной, взять из лабораторной работы №1;
- вывод на экран сообщения о результате и результирующей последовательности символов.

#### Программа должна быть реализована в 2-х вариантах:

- а) односегментная (.com);
- б) многосегментная (.exe).

### Часть 2:

Написать программу на ассемблере, осуществляющую вывод на экран даты создания BIOS (Аф=0FFFF5h) прямой записью в видеопамять (Аф=B8000h). Использовать точечные директивы (модель памяти, директивы сегментации).

## Теоретическая часть:

### *Односегментные и много сегментные программы:*

#### *Односегментные программы:*

Это программы, которые содержат всю свою информацию (код, данные, стек и т.д.) в одном сегменте памяти. Они обычно используются для простых и маленьких задач. Односегментные программы легче для понимания и отладки, однако они не могут эффективно использовать память.

Односегментные программы обычно используют плоскую модель памяти, где все адреса памяти доступны и равноправны. Это упрощает трансляцию и компоновку, поскольку все символы находятся в одном пространстве адресов. Однако это означает, что программа не может эффективно использовать память, поскольку все данные должны быть загружены в память сразу, даже если они не нужны.

В односегментных программах полные директивы определения сегментов обычно не используются, поскольку вся программа находится в одном сегменте. Точечные директивы могут быть использованы для определения местоположения отдельных элементов данных или функций.

#### *Много сегментные программы:*

Это программы, которые разделяют свою информацию на несколько сегментов памяти. Это позволяет более эффективно использовать память, поскольку каждый сегмент может быть загружен и выгружен независимо. Много сегментные программы сложнее для понимания и отладки, но они могут обрабатывать более сложные и большие задачи.

Много сегментные программы могут использовать сегментированную или страничную модель памяти, что позволяет эффективнее использовать память. Отдельные сегменты могут быть загружены и выгружены по мере необходимости, что позволяет обрабатывать большие объемы данных, которые не умещаются в памяти одновременно.

Трансляция и компоновка много сегментных программ сложнее, поскольку символы могут быть распределены по разным сегментам. Компоновщик должен правильно разрешать ссылки на символы, учитывая их сегментное расположение.

В много сегментных программах используются полные директивы определения сегментов для определения начала и конца каждого сегмента. Точечные директивы могут быть использованы внутри сегментов для определения местоположения отдельных элементов данных или функций.

В заключение, односегментные программы проще в отладке и понимании, но менее эффективны с точки зрения использования памяти. Много сегментные программы сложнее в отладке и понимании, но они могут эффективно обрабатывать большие объемы данных. Выбор между односегментными и много сегментными программами зависит от требований конкретной задачи.

### ***Трансляция и компоновка:***

#### ***Трансляция:***

Это процесс преобразования исходного кода программы в машинный код. Транслятор (как правило, компилятор или интерпретатор) анализирует исходный код и преобразует его в машинный код, который может быть выполнен компьютером.

#### ***Компоновка:***

Это процесс объединения разных частей программы (обычно называемых модулями или файлами) в один исполняемый файл. Компоновщик также решает все ссылки на символы, которые не были определены в файле, в котором они используются.

### ***Модели памяти:***

Модели памяти определяют, как система управляет и обращается к памяти. Примеры моделей памяти включают плоскую модель памяти (все адреса памяти доступны и равноправны), сегментированную модель памяти (память разделена на сегменты, каждый из которых имеет свой собственный набор прав доступа) и страничную модель памяти (память разделена на страницы, которые могут быть независимо загружены и выгружены).

### ***Полные и точечные директивы определения сегментов:***

#### ***Полные директивы определения сегментов:***

Это команды, которые определяют начало и конец сегмента памяти. Они указывают, какой тип информации содержится в сегменте (код, данные, стек и т.д.) и как он должен быть загружен и выгружен.

#### ***Точечные директивы определения сегментов:***

Это команды, которые определяют только одну точку в сегменте

## Часть 1

### а) Одно сегментная программа (com):

#### Структура программы:

Программа состоит из следующих блоков:

1. Секция .data: Здесь объявляются массивы и строки для ввода-вывода данных.
2. Секция .code: Здесь находится исполняемый код программы.
3. Метка start: Начальная точка исполнения программы.

#### Организация ввода-вывода:

Ввод и вывод организованы с помощью прерывания DOS 21h, которое предоставляет различные функции ввода-вывода. Код использует следующие функции:

- Функция 9 (AH=9): выводит строку на экран. Строка должна заканчиваться символом '\$'.
- Функция 10 (AH=10): считывает строку с буфера клавиатуры. Первый байт буфера определяет максимальную длину строки.
- Функция 7 (AH=7): считывает символ с клавиатуры без эхо-вывода.

#### Используемые функции:

В программе используются следующие функции:

- **mov**: Копирует значение из второго операнда в первый.
- **int**: Генерирует прерывание программного обеспечения.
- **xor**: Выполняет операцию исключающего ИЛИ между операндами.
- **div**: Делит двойное слово на байт или слово на слово.
- **cmp**: Сравнивает два операнда.
- **je**: Переходит, если равно (ZF=1).
- **xchg**: Обменивает значения между двумя операндами.
- **add**: Складывает два операнда.
- **loop**: Уменьшает CX и переходит, если CX  $\neq$  0.
- **inc**: Увеличивает значение операнда.

#### Особенности работы с видеопамятью:

Программа не взаимодействует непосредственно с видеопамятью, а использует службы DOS для вывода текста на экран.

## Листинг программы:

### lab2\_com.asm:

```
;пересылка слов(W) из in_str в out_str 21-ВМэ-4 07/12/2023
;слова, не равные '01' ~ 3031h, '23' ~ 3233h, '45' ~ 3435h

.model tiny ; Устанавливает модель памяти, где весь код и данные помещаются в одном 64KB сегменте.

.data
; Объявляем строку mess1, которая содержит текст, символы перевода строки (10) и возврата каретки (13),
;и заканчивается символом '$'
mess1 db '21_vmz_4|A_Khaleev Input:',10,13,'$'

; Объявляем строку mess2, которая начинается с символов перевода строки (10) и возврата каретки (13),
;содержит текст 'Output:', затем еще символы перевода строки и возврата каретки, и заканчивается символом '$'
mess2 db 10,13,'Output:',10,13,'$'

; Объявляем массив in_str, состоящий из 22 элементов, каждый из которых инициализируется вопросительным знаком
(?),
; что означает, что начальные значения не определены
in_str db 22 dup (?)

; Объявляем массив out_str, состоящий из 22 элементов, каждый из которых инициализируется символом '$'
out_str db 22 dup ('$')

; в ассемблере строка заканчивается символом '$'. Это позволяет функциям ввода/вывода DOS распознавать конец
строки

.code
ORG 100h ; COM-программы начинаются с этой точки
start:

; приглашение ввода:
    mov dx,offset mess1 ; Загружаем смещение строки mess1 в регистр DX
    mov ah,9             ; Устанавливаем 9 в регистр AH для функции "вывод строки" в прерывании DOS 21h
    int 21h              ; Вызываем прерывание DOS 21h, выводя строку mess1 на экран

; ввод:
    mov dx,offset in_str ; Загружаем смещение строки in_str в регистр DX
    mov in_str, 16        ; Записываем 16 в первый байт in_str, ограничивая ввод 16 байтами
    mov ah,10             ; Устанавливаем 10 в регистр AH для функции "buffered input" в прерывании DOS 21h
    int 21h              ; Вызываем прерывание DOS 21h, читая введенные данные в буфер in_str

;основная часть (модифицированный код из ЛР1):
    mov si,offset in_str+2 ; первые 2 байта служебные
    mov di,offset out_str
    xor ah, ah ; Очистка старшего байта ah
    mov al,in_str+1 ; помещаем в al количество введенных байт символов
    xor dx, dx ; Очистка dx перед div, так как div использует dx:ax
    mov bl, 2 ; Делитель
    div bl ; ax/bl -> результат в al, остаток в ah
    mov cl, al ; так как оперируем словами данных в регистр счетчика записываем значение деленное на 2
    test ah, ah ; Проверка, есть ли остаток
    jz noRemainder ; Если остатка нет, переход к метке noRemainder
    inc cl ; Если был остаток, увеличиваем cl на 1
noRemainder:
    ; Здесь cl содержит количество слов, учитывая нечетное количество байт
    xor ch,ch ; очистка старшего байта счетчика
cmp_cycle:
    mov ax,[si] ; помещаем первое считанное слово в ax (байты переставлены)
    xchg ah, al ; поменять местами старший и младший байты
    cmp ax,3031h ; 3031h ~ '01'
    je cmp_true ; если равно - пропуск
    cmp ax,3233h ; 3233h ~ '23'
    je cmp_true ; если равно - пропуск
    cmp ax,3435h ; 3435h ~ '45'
    je cmp_true ; если равно - пропуск
    xchg ah, al ; поменять местами старший и младший байты
    mov [di], ax ; запись в выходной массив
    add di, 2 ; увеличиваем выходной указатель на 2 (работаем со словами)
cmp_true:
    add si, 2 ; увеличиваем входной указатель на 2 (работаем со словами)
    loop cmp_cycle ; пока регистр счетчика не равен 0 выполняем цикл cmp_cycle
```

```
; вывод результата:
mov dx,offset mess2 ; Загружаем смещение строки mess2 в регистр DX
mov ah,9 ; Загружаем 9 в регистр AH, что соответствует функции "вывод строки" в прерывании DOS 21h
int 21h ; Вызываем DOS interrupt 21h, что выводит строку mess2 на экран

mov dx,offset out_str ; Загружаем смещение строки out_str в регистр DX
mov ah,9 ; Загружаем 9 в регистр AH, что соответствует функции "вывод строки" в прерывании DOS 21h
int 21h ; Вызываем DOS interrupt 21h, что выводит строку out_str на экран

; задержка (ожидание нажатия клавиши):
mov ah,7 ; Устанавливаем 7 в AH, что соответствует функции "ввод без эха" в прерывании DOS 21h
int 21h ; Вызываем DOS interrupt 21h, что позволяет программе ожидать нажатия клавиши пользователем

; завершение:
mov ax,4c00h ; Загружаем 4C00h в AX, что соответствует функции "завершение программы" в прерывании DOS
21h
int 21h ; Вызываем прерывание DOS 21h, что приводит к завершению программы

end start
```

## lab2\_com.lst:

Turbo Assembler Version 2.51  
lab2\_com.asm

12/09/23 21:44:02

Page 1

```
1          ;пересылка слов(W) из in_str в out_str 21-ВМэ-4   07/12/2023
2          ;слова, не равные '01' ~ 3031h, '23' ~ 3233h, '45' ~ 3435h
3
4 0000          .model tiny ; Устанавливает модель памяти, где весь код и +
5              данные помещаются в одном 64KB сегменте.
6
7 0000          .data
8              ; Объявляем строку mess1, которая содержит текст,      +
9              символы перевода строки (10) и возврата каретки (13),
10             ;и заканчивается символом '$'
11 0000 32 31 5F 76 6D 7A      5F+ mess1 db '21_vmz_4|A_Khaleev Input:',10,13,'$'
12      34 7C 41 5F 4B 68      61+
13      6C 65 65 76 20 49      6E+
14      70 75 74 3A 0A 0D      24
15
16              ; Объявляем строку mess2, которая начинается с      +
17              символов перевода строки (10) и возврата каретки (13),
18              ;содержит текст 'Output:', затем еще символы перевода      +
19              строки и возврата каретки, и заканчивается символом+
20              '$'
21 001C 0A 0D 4F 75 74 70      75+ mess2 db 10,13,'Output:',10,13,'$'
22      74 3A 0A 0D 24
23
24              ; Объявляем массив in_str, состоящий из 22 элементов,      +
25              каждый из которых инициализируется вопросительным +
26              знаком (?),
27              ; что означает, что начальные значения не определены
28 0028 16*(??)      in_str db 22 dup (?)
29
30              ; Объявляем массив out_str, состоящий из 22 элементов,      +
31              каждый из которых инициализируется символом '$'
32 003E 16*(24)      out_str db 22 dup ('$')
33
34              ; в ассемблере строка заканчивается символом '$'. Это +
35              позволяет функциям ввода/вывода DOS распознавать      +
36              конец строки
37
38 0054          .code
39              ORG 100h ; COM-программы начинаются с этой точки
40 0100          start:
41
42
43              ; приглашение ввода:
44 0100 BA 0000r      mov dx,offset mess1 ; Загружаем смещение строки mess1 в      +
45              регистр DX
46 0103 B4 09      mov ah,9 ; Устанавливаем 9 в регистр АН для      +
47              функции "вывод строки" в прерывании DOS 21h
48 0105 CD 21      int 21h ; Вызываем прерывание DOS 21h, выводя      +
49              строку mess1 на экран
50
51              ; ввод:
52 0107 BA 0028r      mov dx,offset in_str ; Загружаем смещение строки in_str в
+
53              регистр DX
54 010A C6 06 0028r 10      mov in_str, 16 ; Записываем 16 в первый байт in_str,
+
55              ограничивая ввод 16 байтами
56 010F B4 0A      mov ah,10 ; Устанавливаем 10 в регистр АН для      +
57              функции "buffered input" в прерывании DOS 21h
```



```
58 0111 CD 21          int 21h          ; Вызываем прерывание DOS 21h, читая      +
59                  введенные данные в   буфер in_str
60
61                  ;основная часть (модифицированный код из  JP1):
62 0113 BE 002Ar        mov si,offset in_str+2 ; первые 2 байта  служебные
63 0116 BF 003Er        mov di,offset out_str
64 0119 32 E4          xor ah, ah        ; Очистка старшего байта ax
65 011B A0 0029r        mov al,in_str+1 ; помещаем      в al количество      +
66                  введенных байт символов
67 011E 33 D2          xor dx, dx        ; Очистка dx перед   div, так как div
+
68                  использует dx:ax
69 0120 B3 02          mov bl, 2        ; Делитель
70 0122 F6 F3          div bl           ; ax/bl -> результат в al, остаток в  ah
71 0124 8A C8          mov cl, al        ; так как оперируем словами данных   в      +
72                  регистр счетчика записываем значение деленное на 2
73 0126 84 E4          test ah, ah      ; Проверка, есть ли остаток
74 0128 74 02          jz noRemainder    ; Если остатка нет, переход к метке      +
75                  noRemainder
76 012A FE C1          inc cl           ; Если был остаток, увеличиваем cl на 1
77 012C                noRemainder:
78                  ; Здесь cl содержит количество слов, учитывая      +
79                  нечетное количество байт
80 012C 32 ED          xor ch, ch        ; очистка старшего байта счетчика
81 012E                cmp_cycle:
82 012E 8B 04          mov ax,[si]      ; помещаем первое считанное слово   в ax      +
83                  (байты переставлены)
84 0130 86 E0          xchg ah, al      ; обменять местами старший и младший      +
85                  байты
86 0132 3D 3031        cmp ax,3031h    ; 3031h ~ '01'
87 0135 74 11          je cmp_true      ; если равно - пропуск
88 0137 3D 3233        cmp ax,3233h    ; 3233h ~ '23'
89 013A 74 0C          je cmp_true      ; если равно - пропуск
90 013C 3D 3435        cmp ax,3435h    ; 3435h ~ '45'
91 013F 74 07          je cmp_true      ; если равно - пропуск
92 0141 86 E0          xchg ah, al      ; обменять местами старший и младший      +
93                  байты
94 0143 89 05          mov [di], ax     ; запись в выходной массив
95 0145 83 C7 02        add di, 2        ; увеличиваем выходной указатель на +
96                  2 (работаем со словами)
97 0148                cmp_true:
98 0148 83 C6 02        add si, 2        ; увеличиваем входной указатель на 2      +
99                  (работаем со словами)
100 014B E2 E1          loop cmp_cycle   ; пока регистр счетчика не равен 0      +
101                  выполняем цикл cmp_cycle
102
103                  ; вывод результата:
104 014D BA 001Cr        mov dx,offset mess2 ; Загружаем смещение строки mess2 в      +
105                  регистр DX
106 0150 B4 09          mov ah,9        ; Загружаем 9 в регистр AH,      что соответствует      +
107                  функции "вывод строки" в прерывании DOS 21h
108 0152 CD 21          int 21h          ; Вызываем DOS interrupt 21h, что   выводит строку mess2
+
109                  на экран
110
111 0154 BA 003Er        mov dx,offset out_str ; Загружаем смещение строки out_str в      +
112                  регистр DX
113 0157 B4 09          mov ah,9        ; Загружаем 9 в регистр AH,      что соответствует      +
114                  функции "вывод строки" в прерывании DOS 21h
```

```
115 0159 CD 21      int 21h    ; Вызываем DOS interrupt 21h, что   выводит строку out_str
+
116                на экран
117
118                ; задержка (ожидание нажатия клавиши):
119 015B B4 07      mov ah,7    ; Устанавливаем 7 в AH, что   соответствует   +
120                функции "ввод без эха" в прерывании DOS 21h
121 015D CD 21      int 21h    ; Вызываем DOS interrupt 21h, что   позволяет   +
122                программе ожидать нажатия клавиши пользователем
123
124                ; завершение:
125 015F B8 4C00    mov ax,4c00h ; Загружаем 4C00h в AX, что соответствует   +
126                функции "завершение программы" в прерывании DOS 21h
127 0162 CD 21      int 21h    ; Вызываем прерывание DOS 21h, что   приводит к   +
128                завершению программы
129
130                end start
```

Symbol Name	Type	Value
??DATE	Text	"12/09/23"
??FILENAME	Text	"lab2_com"
??TIME	Text	"21:44:02"
??VERSION	Number	0205
@CODE	Text	DGROUP
@CODESIZE	Text	0
@CPU	Text	0101H
@CURSEG	Text	_TEXT
@DATA	Text	DGROUP
@DATASIZE	Text	0
@FILENAME	Text	LAB2_COM
@MODEL	Text	1
@WORDSIZE	Text	2
CMP_CYCLE	Near	DGROUP:012E
CMP_TRUE	Near	DGROUP:0148
IN_STR	Byte	DGROUP:0028
MESS1	Byte	DGROUP:0000
MESS2	Byte	DGROUP:001C
NOREMAINDER	Near	DGROUP:012C
OUT_STR	Byte	DGROUP:003E
START	Near	DGROUP:0100

Groups & Segments	Bit	Size	Align	Combine	Class
DGROUP					Group
_DATA	16	0054	Word	Public	DATA
_TEXT	16	0164	Word	Public	CODE

*lab2\_com.map:*

Start	Stop	Length	Name	Class
00000H	00163H	00164H	_TEXT	CODE
00164H	001B7H	00054H	_DATA	DATA

## **б) Много сегментная программа (exe):**

### **Структура программы:**

Программа состоит из трех основных сегментов:

1. Сегмент данных (**dl**), где объявляются и инициализируются переменные.
2. Сегмент кода (**cl**), где находится основная логика программы.
3. Сегмент стека (**st**), где резервируется память для стека.

### **Организация ввода-вывода:**

Ввод и вывод организованы с помощью прерывания DOS 21h, которое предоставляет различные функции ввода-вывода. Код использует следующие функции:

- Функция 9 (AH=9): выводит строку на экран. Строка должна заканчиваться символом '\$'.
- Функция 10 (AH=10): считывает строку с буфера клавиатуры. Первый байт буфера определяет максимальную длину строки.
- Функция 7 (AH=7): считывает символ с клавиатуры без эхо-вывода.

### **Используемые функции:**

В программе используются следующие функции:

- **mov**: Копирует значение из второго операнда в первый.
- **int**: Генерирует прерывание программного обеспечения.
- **xor**: Выполняет операцию исключающего ИЛИ между операндами.
- **div**: Делит двойное слово на байт или слово на слово.
- **cmp**: Сравнивает два операнда.
- **je**: Переходит, если равно (ZF=1).
- **xchg**: Обменивает значения между двумя операндами.
- **add**: Складывает два операнда.
- **loop**: Уменьшает CX и переходит, если CX  $\neq$  0.
- **inc**: Увеличивает значение операнда.

### **Особенности работы с видеопамятью:**

Программа не взаимодействует непосредственно с видеопамятью, а использует службы DOS для вывода текста на экран.

## Листинг программы:

### lab2\_exe.asm:

```
;пересылка слов(W) из in_str в out_str 21-ВМэ-4 07/12/2023
;слова, не равные '01' ~ 3031h, '23' ~ 3233h, '45' ~ 3435h

dl segment para public 'data'
; Объявляем строку mess1, которая содержит текст, символы перевода строки (10) и возврата каретки (13),
;и заканчивается символом '$'
mess1 db '21_vmz_4|A_Khaleev Input:',10,13,'$'

; Объявляем строку mess2, которая начинается с символов перевода строки (10) и возврата каретки (13),
;содержит текст 'Output:', затем еще символы перевода строки и возврата каретки, и заканчивается символом '$'
mess2 db 10,13,'Output:',10,13,'$'

; Объявляем массив in_str, состоящий из 22 элементов, каждый из которых инициализируется вопросительным знаком (?),
; что означает, что начальные значения не определены
in_str db 22 dup (?)

; Объявляем массив out_str, состоящий из 22 элементов, каждый из которых инициализируется символом '$'
out_str db 22 dup ('$')

; в ассемблере строка заканчивается символом '$'. Это позволяет функциям ввода/вывода DOS распознавать конец строки
dl ends

cl segment para public 'code'
assume cs:cl,ds:dl,ss:st1
start:
;устанавливаем явно сегментный регистр данных:
    mov ax,dl
    mov ds,ax

; приглашение ввода:
    mov dx,offset mess1 ; Загружаем смещение строки mess1 в регистр DX
    mov ah,9             ; Устанавливаем 9 в регистр AH для функции "вывод строки" в прерывании DOS 21h
    int 21h              ; Вызываем прерывание DOS 21h, выводя строку mess1 на экран

; ввод:
    mov dx,offset in_str ; Загружаем смещение строки in_str в регистр DX
    mov in_str, 16        ; Записываем 16 в первый байт in_str, ограничивая ввод 16 байтами
    mov ah,10             ; Устанавливаем 10 в регистр AH для функции "buffered input" в прерывании DOS 21h
    int 21h              ; Вызываем прерывание DOS 21h, читая введенные данные в буфер in_str

;основная часть (модифицированный код из LPI):
    mov si,offset in_str+2 ; первые 2 байта служебные
    mov di,offset out_str
    xor ah, ah             ; Очистка старшего байта ax
    mov al,in_str+1         ; помещаем в al количество введенных байт символов
    xor dx, dx             ; Очистка dx перед div, так как div использует dx:ax
    mov bl, 2              ; Делитель
    div bl                  ; ax/bl -> результат в al, остаток в ah
    mov cl, al             ; так как оперируем словами данных в регистр счетчика записываем значение деленное на 2
    test ah, ah            ; Проверка, есть ли остаток
    jz noRemainder         ; Если остатка нет, переход к метке noRemainder
    inc cl                  ; Если был остаток, увеличиваем cl на 1
noRemainder:
    ; Здесь cl содержит количество слов, учитывая нечетное количество байт
    xor ch,ch              ; очистка старшего байта счетчика
cmp_cycle:
    mov ax,[si]            ; помещаем первое считанное слово в ax (байты переставлены)
    xchg ah, al            ; поменять местами старший и младший байты
    cmp ax,3031h           ; 3031h ~ '01'
    je cmp_true            ; если равно - пропуск
    cmp ax,3233h           ; 3233h ~ '23'
    je cmp_true            ; если равно - пропуск
    cmp ax,3435h           ; 3435h ~ '45'
    je cmp_true            ; если равно - пропуск
    xchg ah, al            ; поменять местами старший и младший байты
    mov [di], ax           ; запись в выходной массив
    add di, 2              ; увеличиваем выходной указатель на 2 (работаем со словами)
cmp_true:
    add si, 2              ; увеличиваем входной указатель на 2 (работаем со словами)
    loop cmp_cycle         ; пока регистр счетчика не равен 0 выполняем цикл cmp_cycle
```

```

; вывод результата:
mov dx,offset mess2 ; Загружаем смещение строки mess2 в регистр DX
mov ah,9 ; Загружаем 9 в регистр AH, что соответствует функции "вывод строки" в прерывании DOS 21h
int 21h ; Вызываем DOS interrupt 21h, что выводит строку mess2 на экран

mov dx,offset out_str ; Загружаем смещение строки out_str в регистр DX
mov ah,9 ; Загружаем 9 в регистр AH, что соответствует функции "вывод строки" в прерывании DOS 21h
int 21h ; Вызываем DOS interrupt 21h, что выводит строку out_str на экран

; задержка (ожидание нажатия клавиши):
mov ah,7 ; Устанавливаем 7 в AH, что соответствует функции "ввод без эха" в прерывании DOS 21h
int 21h ; Вызываем DOS interrupt 21h, что позволяет программе ожидать нажатия клавиши пользователем

; завершение:
mov ax,4c00h ; Загружаем 4C00h в AX, что соответствует функции "завершение программы" в прерывании DOS 21h
int 21h ; Вызываем прерывание DOS 21h, что приводит к завершению программы
c1 ends

st1 segment para stack 'stack'
dw 100 dup (?) ; Резервирование памяти под 100 слов (200 байт) для стека. Инициализируются неопределенными
значениями
st1 ends

end start

```

## lab2\_exe.lst:

Turbo Assembler Version 2.51  
lab2\_exe.ASM

12/09/23 18:55:06

Page 1

```
1          ;пересылка слов(W) из in_str в out_str 21-ВМэ-4   07/12/2023
2          ;слова, не равные '01' ~ 3031h, '23' ~ 3233h, '45' ~ 3435h
3
4 0000          d1      segment      para public 'data'
5          ; Объявляем строку mess1, которая содержит текст,      +
6          символы перевода строки (10) и возврата каретки (13),
7          ;и заканчивается символом '$'
8 0000  32 31 5F 76 6D 7A      5F+ mess1 db '21_vmz_4|A_Khaleev Input:',10,13,'$'
9          34 7C 41 5F 4B 68      61+
10         6C 65 65 76 20 49      6E+
11         70 75 74 3A 0A 0D      24
12
13          ; Объявляем строку mess2, которая начинается с      +
14          символов перевода строки (10) и возврата каретки (13),
15          ;содержит текст 'Output:', затем еще символы перевода      +
16          строки и возврата каретки, и заканчивается символом+
17          '$'
18 001C  0A 0D 4F 75 74 70      75+ mess2 db 10,13,'Output:',10,13,'$'
19         74 3A 0A 0D 24
20
21          ; Объявляем массив in_str, состоящий из 22 элементов,      +
22          каждый из которых инициализируется вопросительным +
23          знаком (?),
24          ; что означает, что начальные значения не определены
25 0028  16*(??)      in_str db 22 dup (?)
26
27          ; Объявляем массив out_str, состоящий из 22 элементов,      +
28          каждый из которых инициализируется символом '$'
29 003E  16*(24)      out_str db 22 dup ('$')
30
31          ; в ассемблере строка заканчивается символом '$'. Это +
32          позволяет функциям ввода/вывода DOS распознавать      +
33          конец строки
34 0054          d1      ends
35
36 0000          c1      segment      para public 'code'
37          assume cs:c1,ds:d1,ss:st1
38 0000          start:
39          ;устанавливаем явно сегментный регистр данных:
40 0000  B8 0000s      mov ax,d1
41 0003  8E D8          mov ds,ax
42
43          ; приглашение ввода:
44 0005  BA 0000r      mov dx,offset mess1 ; Загружаем смещение строки mess1 в      +
45          регистр DX
46 0008  B4 09          mov ah,9          ; Устанавливаем 9 в регистр AH для      +
47          функции "вывод строки" в прерывании DOS 21h
48 000A  CD 21          int 21h          ; Вызываем прерывание DOS 21h, выводя      +
49          строку mess1 на экран
50
51          ; ввод:
52 000C  BA 0028r      mov dx,offset in_str ; Загружаем смещение строки in_str в
+
53          регистр DX
54 000F  C6 06 0028r 10      mov in_str, 16      ; Записываем 16 в первый байт in_str,
+
55          ограничивая ввод 16 байтами
56 0014  B4 0A          mov ah,10          ; Устанавливаем 10 в регистр AH для      +
57          функции "buffered input" в прерывании DOS 21h
```

```
58 0016 CD 21          int 21h          ; Вызываем прерывание DOS 21h, читая      +
59                  введенные данные в   буфер in_str
60
61                  ;основная часть (модифицированный код из  JP1):
62 0018 BE 002Ar        mov si,offset in_str+2 ; первые 2 байта  служебные
63 001B BF 003Er        mov di,offset out_str
64 001E 32 E4          xor ah, ah        ; Очистка старшего байта ah
65 0020 A0 0029r        mov al,in_str+1 ; помещаем      в al количество      +
66                  введенных байт символов
67 0023 33 D2          xor dx, dx        ; Очистка dx перед div, так как div      +
68                  использует dx:ax
69 0025 B3 02          mov bl, 2 ; Делитель
70 0027 F6 F3          div bl           ; ax/bl -> результат в al, остаток в ah
71 0029 8A C8          mov cl, al        ; так как оперируем словами данных в      +
72                  регистр счетчика записываем значение деленное на 2
73 002B 84 E4          test ah, ah ; Проверка, есть ли остаток
74 002D 74 02          jz noRemainder ; Если остатка нет, переход к метке      +
75                  noRemainder
76 002F FE C1          inc cl           ; Если был остаток, увеличиваем cl на 1
77 0031                noRemainder:
78                  ; Здесь cl содержит количество слов, учитывая      +
79                  нечетное количество байт
80 0031 32 ED          xor ch, ch ; очистка      старшего байта счетчика
81 0033                cmp_cycle:
82 0033 8B 04          mov ax, [si] ; помещаем первое считанное слово      в ax      +
83                  (байты переставлены)
84 0035 86 E0          xchg ah, al ; обменять местами старший и младший      +
85                  байты
86 0037 3D 3031        cmp ax, 3031h ; 3031h ~ '01'
87 003A 74 11          je cmp_true ; если равно - пропуск
88 003C 3D 3233        cmp ax, 3233h ; 3233h ~ '23'
89 003F 74 0C          je cmp_true ; если равно - пропуск
90 0041 3D 3435        cmp ax, 3435h ; 3435h ~ '45'
91 0044 74 07          je cmp_true ; если равно - пропуск
92 0046 86 E0          xchg ah, al ; обменять местами старший и младший      +
93                  байты
94 0048 89 05          mov [di], ax ; запись в выходной массив
95 004A 83 C7 02        add di, 2 ; увеличиваем выходной указатель на +
96                  2 (работаем со словами)
97 004D                cmp_true:
98 004D 83 C6 02        add si, 2 ; увеличиваем входной указатель на 2      +
99                  (работаем со словами)
100 0050 E2 E1          loop cmp_cycle ; пока регистр счетчика не равен 0      +
101                  выполняем цикл cmp_cycle
102
103                  ; вывод результата:
104 0052 BA 001Cr        mov dx,offset mess2 ; Загружаем смещение строки mess2 в      +
105                  регистр DX
106 0055 B4 09          mov ah, 9 ; Загружаем 9 в регистр AH, что      +
107                  соответствует функции "вывод строки" в прерывании DOS+
108                  21h
109 0057 CD 21          int 21h ; Вызываем DOS interrupt 21h, что выводит строку
+                  mess2 на экран
110
111
112 0059 BA 003Er        mov dx,offset out_str ; Загружаем смещение строки out_str в      +
113                  регистр DX
114 005C B4 09          mov ah, 9 ; Загружаем 9 в регистр AH, что      +
```



```
115          соответствует функции "вывод строки" в прерывании DOS+
116          21h
117 005E CD 21          int 21h    ; Вызываем DOS interrupt    21h, что выводит строку
+
118          out_str на экран
119
120          ; задержка    (ожидание нажатия клавиши):
121 0060 B4 07          mov ah,7    ; Устанавливаем 7 в AH, что соответствует    +
122          функции "ввод без эха" в прерывании DOS 21h
123 0062 CD 21          int 21h    ; Вызываем DOS interrupt    21h, что позволяет
+
124          программе ожидать нажатия клавиши пользователем
125
126          ; завершение:
127 0064 B8 4C00        mov ax,4c00h ; Загружаем 4C00h в AX, что соответствует    +
128          функции "завершение программы" в прерывании DOS 21h
129 0067 CD 21          int 21h    ; Вызываем прерывание DOS 21h, что приводит к +
130          завершению программы
131 0069                c1    ends
132
133 0000                stl segment para stack 'stack'
134 0000 64*(????)      dw 100    dup (?)    ; Резервирование памяти под 100 слов
(200+
135          байт) для стека. Инициализируются неопределенными    +
136          значениями
137 00C8                stl ends
138
139          end start
```

Symbol Name	Type	Value
??DATE	Text	"12/09/23"
??FILENAME	Text	"lab2_exe"
??TIME	Text	"18:55:06"
??VERSION	Number	0205
@CPU	Text	0101H
@CURSEG	Text	ST1
@FILENAME	Text	LAB2_EXE
@WORDSIZE	Text	2
CMP_CYCLE	Near	C1:0033
CMP_TRUE	Near	C1:004D
IN_STR	Byte	D1:0028
MESS1	Byte	D1:0000
MESS2	Byte	D1:001C
NOREMAINDER	Near	C1:0031
OUT_STR	Byte	D1:003E
START	Near	C1:0000

Groups & Segments	Bit	Size	Align	Combine	Class
C1	16	0069	Para	Public	CODE
D1	16	0054	Para	Public	DATA
ST1	16	00C8	Para		Stack

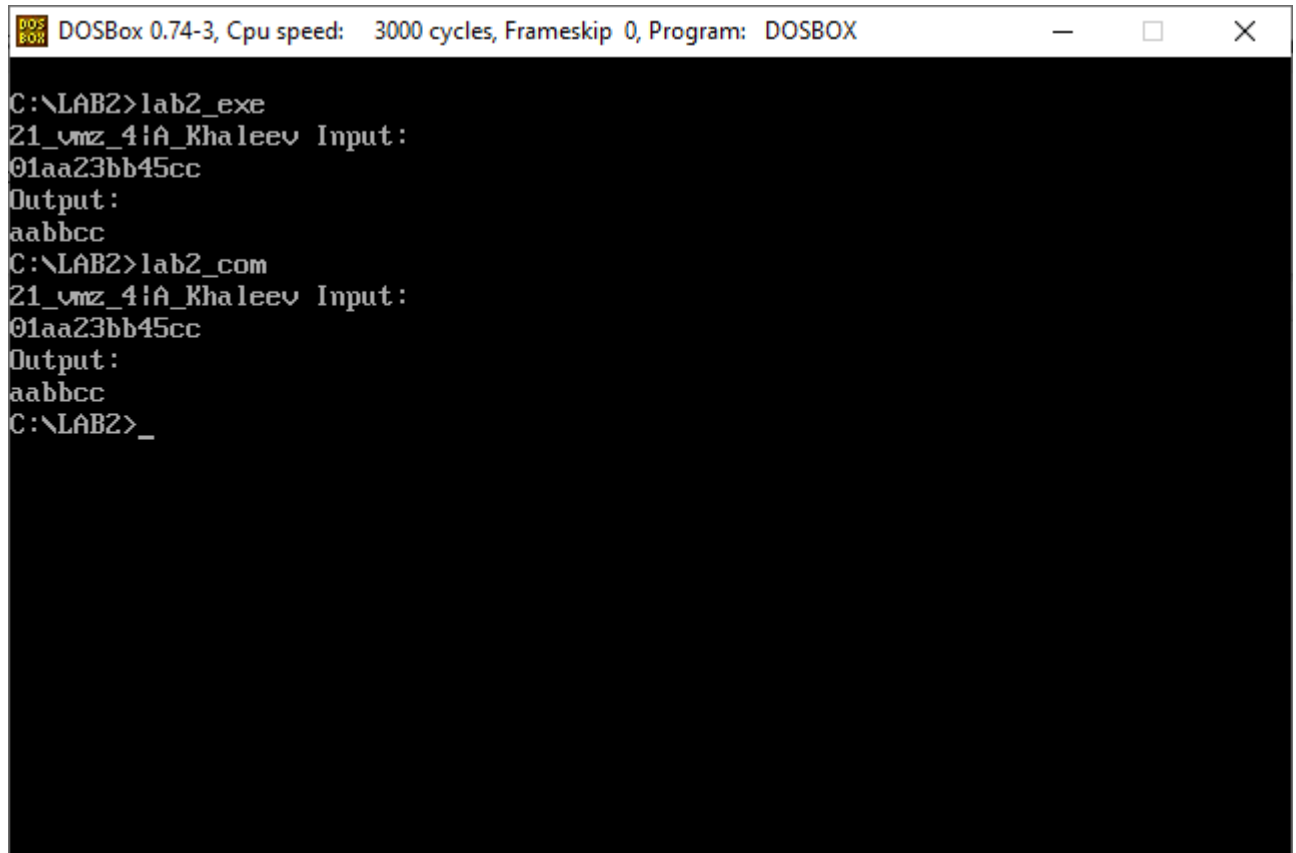
### *lab2\_exe.map:*

Start	Stop	Length	Name	Class
00000H	00053H	00054H	D1	DATA
00060H	000C8H	00069H	C1	CODE
000D0H	00197H	000C8H	ST1	STACK

Program entry point at 0006:0000

## Экранные формы выполнения программы:

Так как логика программ абсолютно одинаковая, ввод и вывод получается идентичным и для exe и для com программы, в отличие от реализации:

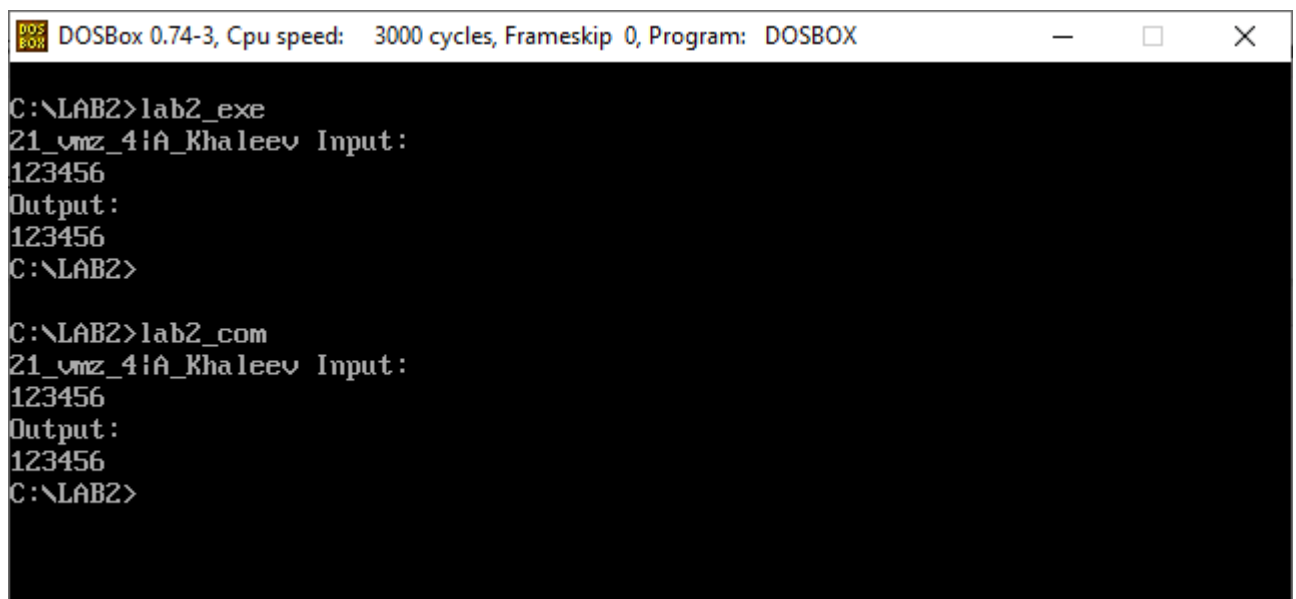


```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
C:\LAB2>lab2.exe
Z1_vnz_4iA_Khaleev Input:
01aa23bb45cc
Output:
aabbcc
C:\LAB2>lab2.com
Z1_vnz_4iA_Khaleev Input:
01aa23bb45cc
Output:
aabbcc
C:\LAB2>_
```

Один ASCII символ занимает один байт, но мы обрабатываем слова, которые занимают 2 байта, и записываем значения, не равные:

$c1 = '01' \sim 3031h$ ;  $c2 = '23' \sim 3233h$ ;  $c3 = '45' \sim 3435h$

Поэтому в выводе из данных на входе оказываются только буквенные символы. Если мы начнем ввод цифр с единицы, то шестнадцатеричные коды двухбайтных слов будут другими, и на вывод попадет вся строка:



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
C:\LAB2>lab2.exe
Z1_vnz_4iA_Khaleev Input:
123456
Output:
123456
C:\LAB2>
C:\LAB2>lab2.com
Z1_vnz_4iA_Khaleev Input:
123456
Output:
123456
C:\LAB2>
```

## Часть 2

Написать программу на ассемблере, осуществляющую вывод на экран даты создания BIOS (Аф=0FFFF5h) прямой записью в видеопамять (Аф=B8000h). Использовать точечные директивы (модель памяти, директивы сегментации).

### Структура программы:

Программа состоит из следующих блоков:

1. Блок макросов:  
В этом блоке определен макрос **anykey**, который используется для ожидания ввода от пользователя без отображения вводимого символа на экране.
2. Секция .code:  
Здесь находится исполняемый код программы.
3. Блок процедур:  
В этом блоке определены процедуры **show\_bios\_str** и **show\_date**, которые используются для вывода строки "BIOS DATE:" и даты BIOS на экран соответственно.
4. Секция .data:  
Здесь объявляются массивы и строки для ввода-вывода данных

### Организация ввода-вывода:

Ввод и вывод данных в этой программе организованы с использованием прерывания DOS 21h, которое предоставляет множество функций для работы с вводом-выводом. В коде программы используются следующие функции:

1. Функция 09 (АН=09): выводит строку на экран. Строка должна заканчиваться символом '\$'.
2. Функция 0A (АН=0A): считывает строку с буфера клавиатуры. Первый байт буфера определяет максимальную длину строки.
3. Функция 07 (АН=07): считывает одиночный символ с клавиатуры без эхо-вывода.

### Используемые функции:

В программе используются следующие функции:

1. **mov**: Перемещает данные между регистрами, из регистра в память и наоборот.
2. **int**: Генерирует программное прерывание.
3. **call**: Вызывает подпрограмму или процедуру.
4. **ret**: Возвращает управление из подпрограммы или процедуры.
5. **inc**: Увеличивает значение операнда на 1.
6. **add**: Выполняет операцию сложения.
7. **loop**: Выполняет циклическое выполнение блока инструкций, пока значение в регистре CX не станет равным нулю.
8. **org**: Определяет начальный адрес текущего сегмента.
9. **proc/endp**: Используются для определения начала и конца процедуры.
10. **macro/endm**: Используются для определения начала и конца макроса.
11. **db/dw**: Определяют байтовые и словесные переменные соответственно.
12. **seg**: Возвращает сегмент указанной переменной.
13. **offset**: Возвращает смещение указанной переменной.
14. **\$**: Возвращает текущий адрес в сегменте.

### Особенности работы с видеопамью:

Программа взаимодействует непосредственно с видеопамью для вывода текста на экран. Видеопамь используется как буфер для вывода символов.

Видеорежим устанавливается в текстовый режим 3 (80x25 символов, 16 цветов) с помощью прерывания BIOS **int 10h**, функция 0.

Сегмент видеопамьи **0B800h** загружается в регистр ES. Смещение DI используется для указания позиции следующего символа на экране. Каждый символ занимает 2 байта: первый байт – ASCII-код символа, второй байт – атрибут символа (цвет и прочее).

Символы строки "BIOS DATE:" и даты BIOS записываются непосредственно в видеопамь по смещению DI с использованием команды **mov es:[di],ax**.

После записи каждого символа смещение DI увеличивается на 2, чтобы перейти к позиции следующего символа.

## Листинг программы:

*lab2\_bios.asm:*

```
.model tiny

; Объявляем макрос, который ожидает нажатия любой клавиши пользователем
anykey macro
    mov ah,7 ; Устанавливаем 7 в АН, что соответствует функции "ввод без эха" в DOS, прерывание 21h
    int 21h ; Вызываем прерывание DOS 21h, что позволяет программе ожидать нажатия клавиши пользователем
endm

.code
org 100h
main:
    mov ax,3 ; Устанавливаем видеорежим
    int 10h ; Вызываем прерывание для установки видеорежима
    mov es,seg_video ; Устанавливаем сегмент видео памяти
    call show_bios_str ; Вызываем процедуру, которая показывает строку "BIOS DATE:"
    call show_date ; Вызываем процедуру, которая показывает дату BIOS
    anykey ; Ожидаем нажатия любой клавиши пользователем
    ret ; Возвращаем управление операционной системе

; Процедура для вывода строки "BIOS DATE:"
show_bios_str proc
    mov cx,N ; Загружаем количество символов в строке
    mov di,696 ; Устанавливаем позицию, где будет выводиться строка на экране
    mov ah,2Eh ; Устанавливаем цвет текста (желтый текст на зеленом фоне: bin = 0010 1110)
    mov si,offset mess ; Загружаем смещение строки
print_loop:
    mov al,[si] ; Загружаем символ из строки
    mov es:[di],ax ; Выводим символ на экран
    add di,2 ; Переходим к следующей позиции на экране
    inc si ; Переходим к следующему символу в строке
    loop print_loop ; Повторяем цикл, пока не будут выведены все символы
    ret
show_bios_str endp

; Процедура для вывода даты BIOS
show_date proc
    mov cx,8 ; Загружаем количество символов в дате BIOS
    mov ds,seg_bios ; Загружаем сегмент BIOS
    mov si,5 ; Загружаем начальное смещение для чтения даты из BIOS
    mov di,720 ; Загружаем позицию, где будет выводиться дата на экране
    mov ah,0Ah ; Устанавливаем цвет даты (зеленый текст на черном фоне: bin = 0000 1010)
date_loop:
    mov al,[si] ; Загружаем байт даты из BIOS
    mov es:[di],ax ; Выводим байт даты на экран
    add di,2 ; Переходим к следующей позиции на экране
    inc si ; Переходим к следующему байту даты в BIOS
    loop date_loop ; Повторяем цикл, пока не будут выведены все символы даты
    ret
show_date endp

; данные:
.data
seg_video dw 0B800h ; Сегмент видео памяти
seg_bios dw 0FFFFh ; Сегмент BIOS
mess db 'BIOS DATE:' ; Строка, которую мы будем выводить на экран
N=$-mess ; Вычисляем количество символов в строке
end main
```

## lab2\_bios.lst:

Turbo Assembler Version 2.51 12/09/23 22:25:27 Page 1  
lab2\_b~1.asm

```
1 0000          .model tiny
2
3          ; Объявляем макрос, который ожидает нажатия любой +
4          клавиши пользователем
5          anykey macro
6          mov ah,7 ; Устанавливаем 7 в AH, что соответствует +
7          функции "ввод без эха" в DOS, прерывание 21h
8          int 21h ; Вызываем прерывание DOS 21h, что позволяет +
9          программе ожидать нажатия клавиши пользователем
10         endm
11
12 0000          .code
13          org 100h
14 0100          main:
15 0100 B8 0003    mov ax,3 ; Устанавливаем видеорежим
16 0103 CD 10     int 10h ; Вызываем прерывание для установки +
17         видеорежима
18 0105 8E 06 0000r mov es,seg_video ; Устанавливаем сегмент видео памяти
19 0109 E8 0008    call show_bios_str; Вызываем процедуру, которая +
20         показывает строку "BIOS DATE:"
21 010C E8 001D    call show_date ; Вызываем процедуру, которая +
22         показывает дату BIOS
23         anykey ; Ожидаем нажатия любой клавиши +
24         пользователем
1 25 010F B4 07    mov ah,7 ; Устанавливаем 7 в AH, что соответствует +
26         функции "ввод без эха" в DOS, прерывание 21h
1 27 0111 CD 21    int 21h ; Вызываем прерывание DOS 21h, что позволяет +
28         программе ожидать нажатия клавиши пользователем
29 0113 C3        ret ; Возвращаем управление операционной +
30         системе
31
32         ; Процедура для вывода строки "BIOS DATE:"
33 0114          show_bios_str proc
34 0114 B9 000A 90    mov cx,N ; Загружаем количество символов в строке
35 0118 BF 02B8      mov di,696 ; Устанавливаем позицию, где будет +
36         выводиться строка на экране
37 011B B4 2E       mov ah,2Eh ; Устанавливаем цвет текста (желтый +
38         текст на зеленом фоне: bin = 0010 1110)
39 011D BE 0004r     mov si,offset mess ; Загружаем смещение строки
40 0120          print_loop: ; Начало цикла печати
41 0120 8A 04        mov al,[si] ; Загружаем символ из строки
42 0122 26: 89 05     mov es:[di],ax ; Выводим символ на экран
43 0125 83 C7 02     add di,2 ; Переходим к следующей позиции на +
44         экране
45 0128 46          inc si ; Переходим к следующему символу в +
46         строке
47 0129 E2 F5       loop print_loop ; Повторяем цикл, пока не будут выведены +
48         все символы
49 012B C3         ret
50 012C          show_bios_str endp
51
52         ; Процедура для вывода даты BIOS
53 012C          show_date proc
54 012C B9 0008      mov cx,8 ; Загружаем количество символов в дате +
55         BIOS
56 012F 8E 1E 0002r   mov ds,seg_bios ; Загружаем сегмент BIOS
57 0133 BE 0005      mov si,5 ; Загружаем начальное смещение для +
```

```
58      чтения даты из      BIOS
59 0136 BF 02D0      mov di,720      ; Загружаем позицию, где будет      +
60      выводиться дата на      экране
61 0139 B4 0A      mov ah,0Ah      ; Устанавливаем цвет даты (зеленый текст+
62      на черном фоне: bin = 0000 1010)
63 013B      date_loop:      ; Начало цикла вывода даты
64 013B 8A 04      mov al,[si]      ; Загружаем байт даты из BIOS
65 013D 26: 89 05      mov es:[di],ax      ; Выводим байт даты на экран
66 0140 83 C7 02      add di, 2      ; Переходим к следующей позиции на      +
67      экране
68 0143 46      inc si      ; Переходим к следующему байту даты в BIOS
69 0144 E2 F5      loop date_loop      ; Повторяем цикл, пока не будут выведены +
70      все символы даты
71 0146 C3      ret
72 0147      show_date endp
73
74      ; данные:
75 0147      .data
76 0000 B800      seg_video dw 0B800h      ; Сегмент видео памяти
77 0002 FFFF      seg_bios dw 0FFFFh      ; Сегмент BIOS
78 0004 42 49 4F 53 20 44      41+ mess db 'BIOS DATE:' ; Строка, которую мы будем выводить на
+
79      54 45 3A      экран
80      = 000A      N=$-mess      ; Вычисляем количество символов в строке
81      end main
```



Symbol Name	Type	Value
??DATE	Text	"12/09/23"
??FILENAME	Text	"lab2_b~1"
??TIME	Text	"22:25:27"
??VERSION	Number	0205
@CODE	Text	DGROUP
@CODESIZE	Text	0
@CPU	Text	0101H
@CURSEG	Text	_DATA
@DATA	Text	DGROUP
@DATASIZE	Text	0
@FILENAME	Text	LAB2_B~1
@MODEL	Text	1
@WORDSIZE	Text	2
DATE_LOOP	Near	DGROUP:013B
MAIN	Near	DGROUP:0100
MESS	Byte	DGROUP:0004
N	Number	000A
PRINT_LOOP	Near	DGROUP:0120
SEG_BIOS	Word	DGROUP:0002
SEG_VIDEO	Word	DGROUP:0000
SHOW_BIOS_STR	Near	DGROUP:0114
SHOW_DATE	Near	DGROUP:012C

## Macro Name

ANYKEY

## Groups &amp; Segments      Bit Size Align   Combine Class

Group	Group	Bit	Size	Align	Combine	Class
DGROUP						
_DATA	16	000E	Word		Public	DATA
_TEXT	16	0147	Word		Public	CODE

*lab2\_bios.map:*

Start	Stop	Length	Name	Class
00000H	00146H	00147H	TEXT	CODE
00148H	00155H	0000EH	_DATA	DATA

Экранная форма выполнения программы:



## Вывод

В ходе лабораторной работы были приобретены навыки:

- 1) разработки *одно-* и *много-* сегментных программ на языке ассемблер
- 2) использования функций прерываний для организации ввода-вывода, управление трансляцией и компоновкой.

Были изучены особенности работы с видеопамью посредством прямой записи, использование точечных директив (модель памяти, директивы сегментации).

Также были написаны, скомпилированы, слинкованы в исполняемые файлы и протестированы 3 программы согласно заданию. Программы работают корректно.