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

# ОТЧЕТ

**по лабораторной работе №4**
**по дисциплине «Организация ЭВМ и систем»**
**ТЕМА: «Представление и обработка символьной информации с использованием строковых команд»**

Студент гр. 9383 _____ Рыбников Р.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2020

**Цель работы.**

Научиться обрабатывать символьную информацию с помощью встраивания in-line.

**Текст задания (Вариант 17).**

Преобразование введенных во входной строке русских букв в латинские в соответствие с правилами транслитерации, остальные символы входной строки передаются в выходную строку непосредственно

**Ход работы.**

Были использованы следующие команды для выполнения данного задания:

1. lea – вычисляет эффективный адрес источника(справа) и помещает его в приёмник (слева). Источником может быть только переменная (ячейка памяти), а приёмником только регистр (не сегментный). Эффективный адрес это действующий (текущий) адрес (база + смещение + индекс). Если адрес 32-разрядный, а приёмник 16-разрядный, то старшая половина вычисленного адреса теряется, иначе вычисленное смещение дополняется нулями. Команда lea может определять смещение входе выполнения программы, в отличии от команды offset (при компиляции).
2. je – Jump if equal (x = y) ZF = 1
3. inc байт – команда "инкремент" выполняет прибавление "1" к указанной переменной и влияет на флаги. Начальное значение 0FFH перейдет в 00H. Эта команда допускает четыре режима адресации: к аккумулятору, регистровый, прямой, косвенно-регистровый
4. dec байт – команда «декремент» производит вычитание «1» из указанного операнда. Начальное значение 00H перейдет в 0FFH. Команда dec не влияет на флаги. Этой командой допускается четыре режима адресации операнда: к аккумулятору, регистровый, прямой, косвенно-регистровый
5. [eax] – адрес

Программа вычисляет адрес входной и выходной строки, вызывает функцию замены символов, в регистре edx находится входная строка, в ecx – выходная строка. Фиксируется текущий символ и помещается в регистр al. Далее происходит сравнение со всеми символами которые подразумевают однозначную замену (1 символ в 1 символ). После этого, если эти символы обнаружены не были, рассматриваются символы которые подразумевают неоднозначную замену (ё, ч, ш, щ, ж). Программа завершит обработку строки, когда встретит символ «\0». После конца работы функции замены символов, программа выведет результат работы.

**Тестирование.**

| Входные данные | Результат |
|---|---|
| Lol kek cheburek | Лол кек чебурек |
| | |
| Yo zh sh shh | Ё ж ш щ |

Реализованный код смотреть в приложении А.

**Выводы.**

Приобртены навыки обработки символьной информации, изучен принцип in-line.

# ПРИЛОЖЕНИЕ А
# КОД ПРОГРАММЫ

**main.cpp**

```
#include <iostream>
#include <string>
#include <stdio.h>
#include <windows.h>

using namespace std;

void Init() {
    cout <<
"_____
_____" << endl;
    cout << "Работу выполнил: студент группы 9383, Рыбников
Роман." << endl;
    cout << "Задание 17: Преобразование введенных во входной
строке латинских букв " << endl;
    cout << "в русские в соответствие с правилами транслитерации,
остальные символы" << endl;
    cout << "входной строки передаются в выходную строку
непосредственно." << endl;
    cout <<
"_____
_____" << endl;
}


void Print(string* a, int len) {
    for (int i = 0; i < len; i++)
        cout << a[i] << endl;
}
int main(){

    SetConsoleOutputCP(1251);
    SetConsoleCP(1251);

        Init();
        cout << endl << endl;
        cout << "Введите строку" << endl;
        const int32_t alphabetSize = 33;
        char eng[] =
"a_b_v_g_d_e_yo_zh_z_i_j_k_l_m_n_o_p_r_s_t_u_f_x_c_ch_sh_shh_''_y_
'_e'_yu_ya\0";
        char rus[] = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя\0";
        int rusLen = strlen(rus);
        int engLen = strlen(eng);
        string str = "";
```

```
        getline(cin, str);
        char instr[80] = "";
        for (int i = 0; i < str.size() && i < 80; i++)
            instr[i] = str[i];
        char outstr[80] = "";
        _asm {
                //Прыжок в точку входа
                jmp START

#pragma region ENG_RUS_REPLACER
                // Конвертирует транслит в кирилическую строку
                // edx - входная строка
                // ecx - выходная
                ENG_RUS_REPLACER:
                ENG_RUS_REPLACER_LOOP:
                //берем текущий символ
                mov al, [edx]

#pragma region Сравнения
                //сравниваем
                cmp al, 39
                //прыжок к замене
                je ENG_RUS_REPLACER_LOOP_LSIGN
                //сравниваем
                cmp al, 'a'
                //прыжок к замене
                je ENG_RUS_REPLACER_LOOP_A
                //сравниваем
                cmp al, 'b'
                //прыжок к замене
                je ENG_RUS_REPLACER_LOOP_B
                //сравниваем
                cmp al, 'v'
                //прыжок к замене
                je ENG_RUS_REPLACER_LOOP_V
                //сравниваем
                cmp al, 'c'
                //прыжок к замене
                je ENG_RUS_REPLACER_LOOP_C
                //сравниваем
                cmp al, 'g'
                //прыжок к замене
                je ENG_RUS_REPLACER_LOOP_G
                //сравниваем
                cmp al, 'd'
                //прыжок к замене
                je ENG_RUS_REPLACER_LOOP_D
                //сравниваем
                cmp al, 'e'
                //прыжок к замене
                je ENG_RUS_REPLACER_LOOP_E
                //сравниваем
                cmp al, 'y'
```

```
//прыжок к замене
je ENG_RUS_REPLACER_LOOP_Y
//сравниваем
cmp al, 'z'
//прыжок к замене
je ENG_RUS_REPLACER_LOOP_Z
//сравниваем
cmp al, 'i'
//прыжок к замене
je ENG_RUS_REPLACER_LOOP_I
//сравниваем
cmp al, 'j'
//прыжок к замене
je ENG_RUS_REPLACER_LOOP_J
//сравниваем
cmp al, 'k'
//прыжок к замене
je ENG_RUS_REPLACER_LOOP_K
//сравниваем
cmp al, 'l'
//прыжок к замене
je ENG_RUS_REPLACER_LOOP_L
//сравниваем
cmp al, 'm'
//прыжок к замене
je ENG_RUS_REPLACER_LOOP_M
//сравниваем
cmp al, 'n'
//прыжок к замене
je ENG_RUS_REPLACER_LOOP_N
//сравниваем
cmp al, 'o'
//прыжок к замене
je ENG_RUS_REPLACER_LOOP_O
//сравниваем
cmp al, 'p'
//прыжок к замене
je ENG_RUS_REPLACER_LOOP_P
//сравниваем
cmp al, 'r'
//прыжок к замене
je ENG_RUS_REPLACER_LOOP_R
//сравниваем
cmp al, 's'
//прыжок к замене
je ENG_RUS_REPLACER_LOOP_S
//сравниваем
cmp al, 't'
//прыжок к замене
je ENG_RUS_REPLACER_LOOP_T
//сравниваем
cmp al, 'u'
//прыжок к замене
```

```
                je ENG_RUS_REPLACER_LOOP_U
                //сравниваем
                cmp al, 'f'
                //прыжок к замене
                je ENG_RUS_REPLACER_LOOP_F
                //сравниваем
                cmp al, 'x'
                //прыжок к замене
                je ENG_RUS_REPLACER_LOOP_X
                //сравниваем
                cmp al, 'c'
                //прыжок к замене
                je ENG_RUS_REPLACER_LOOP_C
                //сравниваем
                cmp al, 's'
                //прыжок к замене
                je ENG_RUS_REPLACER_LOOP_S
                //замены не было , следующий символ
                jmp ENG_RUS_REPLACER_LOOP_NEXT_NO_REPLACE
#pragma endregion

#pragma region C
                ENG_RUS_REPLACER_LOOP_C :
                //вставка нужного символа в регистр
                mov al, 'ц'
                //вставляем символ в выходную строку
                mov[ecx], al
                //смещаем ардес
                inc edx
                //заносим следующий символ в регистр
                mov al, [edx]
                //сравниваем
                cmp al, 'h'
                //не равно - одинарный литерал, равно 2-3 значный
                jne ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE_COM-
PLEX_2
                //вставка нужного символа в регистр
                mov al, 'ч'
                //вставляем символ в выходную строку
                mov[ecx], al
                //следующая итерация, переход к следующему символу
                jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE
#pragma endregion

#pragma region Y
                ENG_RUS_REPLACER_LOOP_Y :
                //вставка нужного символа в регистр
                mov al, 'ы'
                //вставляем символ в выходную строку
                mov[ecx], al
                //смещаем ардес
                inc edx
                //заносим следующий символ в регистр
```

```
                mov al, [edx]
                //сравниваем
                cmp al, 'o'
                //заносим в регистр нужный символ
                mov al, 'ё'
                //записываем в текущую позицию в строке
                mov[ecx], al
                je ENG_RUS_REPLACER_LOOP_Y_DONE
                //заносим следующий символ в регистр
                mov al, [edx]
                //сравниваем
                cmp al, 'u'
                //заносим в регистр нужный символ
                mov al, 'ю'
                //записываем в текущую позицию в строке
                mov[ecx], al
                je ENG_RUS_REPLACER_LOOP_Y_DONE
                //заносим следующий символ в регистр
                mov al, [edx]
                //сравниваем
                cmp al, 'a'
                //заносим в регистр нужный символ
                mov al, 'я'
                //записываем в текущую позицию в строке
                mov[ecx], al
                je ENG_RUS_REPLACER_LOOP_Y_DONE
                jne ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE_COM-
PLEX_2
                ENG_RUS_REPLACER_LOOP_Y_DONE :
                jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE

#pragma endregion

#pragma region S
                ENG_RUS_REPLACER_LOOP_S :
                //вставка нужного символа в регистр
                mov al, 'c'
                //вставляем символ в выходную строку
                mov[ecx], al
                //смещаем ардес
                inc edx
                //заносим следующий символ в регистр
                mov al, [edx]
                //сравниваем
                cmp al, 'h'
                jne ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE_COM-
PLEX_2
                //заносим нужный символ
                mov al, 'ш'
                //заносим в выходную строку
                mov[ecx], al
                //вставляем символ в выходную строку
                mov[ecx], al
```

```
                //смещаем ардес
                inc edx
                //заносим следующий символ в регистр
                mov al, [edx]
                //сравниваем
                cmp al, 'h'
                jne ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE_COM-
PLEX_2
                //заносим нужный символ
                mov al, 'щ'
                //заносим в выходную строку
                mov[ecx], al
                jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE
#pragma endregion

#pragma region E
                ENG_RUS_REPLACER_LOOP_E :
                //вставка нужного символа в регистр
                mov al, 'е'
                //вставляем символ в выходную строку
                mov[ecx], al
                //смещаем ардес
                inc edx
                //заносим следующий символ в регистр
                mov al, [edx]
                //сравниваем
                cmp al, 39
                //не равно - одинарный литерал, равно 2-3 значный
                jne ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE_COM-
PLEX_2
                //вставка нужного символа в регистр
                mov al, 'э'
                //вставляем символ в выходную строку
                mov[ecx], al
                //следующая итерация, переход к следующему символу
                jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE
                jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE
#pragma endregion

#pragma region Z
                ENG_RUS_REPLACER_LOOP_Z :
                //вставка нужного символа в регистр
                mov al, 'з'
                //вставляем символ в выходную строку
                mov[ecx], al
                //смещаем ардес
                inc edx
                //заносим следующий символ в регистр
                mov al, [edx]
                //сравниваем
                cmp al, 'h'
                //не равно - одинарный литерал, равно 2-3 значный
```

```
                jne ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE_COM-
PLEX_2
                //вставка нужного символа в регистр
                mov al, 'ж'
                //вставляем символ в выходную строку
                mov[ecx], al
                //следующая итерация, переход к следующему символу
                jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE
                jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE
#pragma endregion

#pragma region Одноразрядные
                ENG_RUS_REPLACER_LOOP_LSIGN :
                mov al, 'ь'
                //вставляем символ в выходную строку
                mov[ecx], al
                //переход к следующей итерации
                jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE
                ENG_RUS_REPLACER_LOOP_A :
                //вставка нужного символа в регистр
                mov al, 'а'
                //вставляем символ в выходную строку
                mov[ecx], al
                //переход к следующей итерации
                jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE
                ENG_RUS_REPLACER_LOOP_B :
                //вставка нужного символа в регистр
                mov al, 'б'
                //вставляем символ в выходную строку
                mov[ecx], al
                //переход к следующей итерации
                jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE
                ENG_RUS_REPLACER_LOOP_V :
                //вставка нужного символа в регистр
                mov al, 'в'
                //вставляем символ в выходную строку
                mov[ecx], al
                //переход к следующей итерации
                jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE
                ENG_RUS_REPLACER_LOOP_G :
                //вставка нужного символа в регистр
                mov al, 'г'
                //вставляем символ в выходную строку
                mov[ecx], al
                //переход к следующей итерации
                jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE
                ENG_RUS_REPLACER_LOOP_D :
                //вставка нужного символа в регистр
                mov al, 'д'
                //вставляем символ в выходную строку
                mov[ecx], al
                //переход к следующей итерации
                jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE
```

```
ENG_RUS_REPLACER_LOOP_I :
//вставка нужного символа в регистр
mov al, 'и'
//вставляем символ в выходную строку
mov[ecx], al
//переход к следующей итерации
jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE
ENG_RUS_REPLACER_LOOP_J :
//вставка нужного символа в регистр
mov al, 'й'
//вставляем символ в выходную строку
mov[ecx], al
//переход к следующей итерации
jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE
ENG_RUS_REPLACER_LOOP_K :
//вставка нужного символа в регистр
mov al, 'к'
//вставляем символ в выходную строку
mov[ecx], al
//переход к следующей итерации
jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE
ENG_RUS_REPLACER_LOOP_L :
//вставка нужного символа в регистр
mov al, 'л'
//вставляем символ в выходную строку
mov[ecx], al
//переход к следующей итерации
jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE
ENG_RUS_REPLACER_LOOP_M :
//вставка нужного символа в регистр
mov al, 'м'
//вставляем символ в выходную строку
mov[ecx], al
//переход к следующей итерации
jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE
ENG_RUS_REPLACER_LOOP_N :
//вставка нужного символа в регистр
mov al, 'н'
//вставляем символ в выходную строку
mov[ecx], al
//переход к следующей итерации
jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE
ENG_RUS_REPLACER_LOOP_O :
//вставка нужного символа в регистр
mov al, 'о'
//вставляем символ в выходную строку
mov[ecx], al
//переход к следующей итерации
jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE
ENG_RUS_REPLACER_LOOP_P :
//вставка нужного символа в регистр
mov al, 'п'
//вставляем символ в выходную строку
```

```
                mov[ecx], al
                //переход к следующей итерации
                jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE
                ENG_RUS_REPLACER_LOOP_R :
                //вставка нужного символа в регистр
                mov al, 'р'
                //вставляем символ в выходную строку
                mov[ecx], al
                //переход к следующей итерации
                jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE
                ENG_RUS_REPLACER_LOOP_T :
                //вставка нужного символа в регистр
                mov al, 'т'
                //вставляем символ в выходную строку
                mov[ecx], al
                //переход к следующей итерации
                jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE
                ENG_RUS_REPLACER_LOOP_U :
                //вставка нужного символа в регистр
                mov al, 'у'
                //вставляем символ в выходную строку
                mov[ecx], al
                //переход к следующей итерации
                jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE
                ENG_RUS_REPLACER_LOOP_F :
                //вставка нужного символа в регистр
                mov al, 'ф'
                //вставляем символ в выходную строку
                mov[ecx], al
                //переход к следующей итерации
                jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE
                ENG_RUS_REPLACER_LOOP_X :
                //вставка нужного символа в регистр
                mov al, 'х'
                //вставляем символ в выходную строку
                mov[ecx], al
                //переход к следующей итерации
                jmp ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE


#pragma endregion
                ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE_COMPLEX_3:
                //вернем адрес входной строки на 1 позицию назад
                dec edx
                //вернем адрес входной строки на 1 позицию назад
                dec edx
                //следующая итерация
                JMP ENG_RUS_REPLACER_LOOP_NEXT

                ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE_COMPLEX_2
:
                //вернем адрес входной строки на 1 позицию назад
                dec edx
```

```
                //следующая итерация
                JMP ENG_RUS_REPLACER_LOOP_NEXT

                ENG_RUS_REPLACER_LOOP_NEXT_NO_REPLACE :
                //вставляем текущий символ как есть
                mov[ecx], al
                //следующая итерация
                JMP ENG_RUS_REPLACER_LOOP_NEXT

                ENG_RUS_REPLACER_LOOP_NEXT_REPLACE_DONE :
                // тут просто смещаем строки посимвольно и
проверяем дальше
                ENG_RUS_REPLACER_LOOP_NEXT:
                //смещаем указатель в выходной строке
                inc ecx
                //смещаем указатель в входной строке
                inc edx
                //проверяем текущий символ на конец строки
                mov al, [edx]
                //если стоит конец строки - выходим
                cmp al, '\0'
                //нет -продолжаем
                JNE ENG_RUS_REPLACER_LOOP
                //возврат во внешний код
                //pop edx
                RET

#pragma endregion

#pragma region Test
                START :
                //адрес входной строки
                lea edx, instr
                //адрес выходной строки
                lea ecx, outstr
                //саппуск замены
                CALL ENG_RUS_REPLACER
                FINISH :
                mov eax, 0
#pragma endregion
        }
        cout << "String before: " << instr << endl;
        cout << "String after: " << outstr << endl;
    return 0;
}
```