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»

Студентка гр. 9383 ___ Сергиенкова А.А.

Преподаватель ___ Ефремов М.А.

Санкт-Петербург
2020

Цель работы.

Научиться обрабатывать информацию посимвольно на языке ассемблера. Изучить принцип встраивания in-line.

Текст задания (Вариант 19).

Заменить введенные во входной строке латинские буквы на десятичные числа, соответствующие их номеру по алфавиту, остальные символы входной строки передать в выходную строку непосредственно.

Ход работы.

Была реализована программа на языке Ассемблер. Программа посимвольно обрабатывает поступающую информацию. Ввод и вывод строки реализован на ЯВУ, обработка информации реализована на Ассемблере.

Замена введённых латинских букв на десятичные числа, соответствующие их номеру в алфавите осуществлялась с помощью следующих инструкций:

Были использованы следующие команды для выполнения данного задания:

- jne – выполняет переход, если операнды не равны.
- ja – выполняет переход, если первый операнд больше второго. CF = 0 & ZF = 0.
- je – выполняет короткий переход, если первый операнд равен второму. ZF = 1.
- push – команда для записи в стек.
- inc – увеличивает число на единицу.
- pop – команда выталкивания из стека.
- dec – команда производит вычитание 1 из указанного операнда.
- [eax] – адрес.
- cmp – для сравнения двух операндов.
- jmp – безусловный переход. Используется, если для перехода к следующему адресу не нужно делать дополнительных проверок.

- 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).

Тестирование.

Входные данные	Результат
Now or never	141523 1518 12522518
A life is a moment	1 12965 919 1 13151351420
Imagination rules the world	9131791412091514 182112519 2085 231518124

Реализованный код смотреть в приложении А.

Выводы.

Приобретены навыки обработки информации посимвольно, изучен принцип вставки in-line.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ

19lb.asm

```
#include <iostream>
#include <string>
#include<stdio.h>
#include<windows.h>
using namespace std;
void Print(string* a, int len) {
    for (int i = 0; i < len; i++)
        cout « a[i] « endl;
}
int main()
{
    SetConsoleOutputCP(1251);
    SetConsoleCP(1251);
    try {
        const int32_t alphabetSize = 33;
        char rus[] = "abcdefghijklmnopqrstuvwxyz";
        char eng[] =
"_1_2_3_4_5_6_7_8_9_10_11_12_13_14_15_16_17_18_19_20_21_22_23_24_25_26_27";
        string str = "";
        getline(cin,  str);
        char intpustr[80] = "";
        for (int i = 0; i < str.size() && i < 80; i++)
            intpustr[i] = str[i];
        char outpustr[80] = "";
        _asm {
            //Прыжок в точку входа
            jmp START
#pragma region SEARCHRUS

            /// поиск в русском алфавите

            /// символ должен быть в al
            /// вернет позицию символа в eax, если не найдено 100
        SEARCHRUS:
            ///сохранение стека
            push ebx
            //сохраняем регистр в стек
            push ecx
            //сохраняем регистр в стек
            push edx
            //404 - не найдено
            mov ecx, 404
            /// счетчик цикла
            mov ebx, 0
            /// взяли адрес строки с кириллицей
            lea edx, rus
            //цикл
        SEARCHRUS_LOOP: // проходим пока не найдём нужный символ
            //если символ из массива совпал с искомым
            cmp al, [edx]
            //переходим к возврату значения
            je RETURN_SAVE
            //увеличиваем адрес в массиве на 1 байт
            inc ebx
            //увеличиваем адрес в массиве на 1 байт
            inc edx
            //размер алфавита - это условие выхода из цикла
            cmp ebx, 26
            //если счетчик достиг 26 - выходим
            je RETURN
```

```
            //иначе продолжение цикла
            jmp SEARCHRUS_LOOP
            //выход из процедуры
        RETURN_SAVE:
            // сохраняем номер указанного символа в алфавите
            mov eax, ebx
            //переход на восстанолвение стека
            jmp RESTORE_STACK
        RETURN:
            // считаем, что ничего не нашли, индекс 404
            mov eax, 404

            RESTORE_STACK :
            //восстановление регистра из стека
            pop edx
            //восстановление регистра из стека
            pop ecx
            //восстановление регистра из стека
            pop ebx
            //здесь ставим нужную метку возврата
            jmp RET_SEARCHRUS
#pragma endregion

#pragma region SEARCHENG

            /// Ищет замену по указанному индексу, индекс указывается в eax

            /// в edx будет указатель на начало замены(адрес) в eax - длина
замены
        SEARCHENG:
            //проверяем, корректен ли индекс
            cmp eax, 69
            //если он больше 69(длина строки), сразу выходим
            ja SEARCHENG_RETURN
            ///сохранение регистра в стеке
            push ebx
            ///сохранение регистра в стеке
            push ecx
            // счетчик символов
            mov ebx, 0
            // счетчик пройденых литералов
            mov ecx, 0
            /// взяли адрес строки с транскрипциями
            lea edx, eng
            inc eax
            //основной цикл
            SEARCHENG_LOOP :
            //смотрим текущий символ
            //сохраняем eax - там искомый индекс
            push eax
            //отладка
            //mov al, [edx]
            //проверяем, я вляется ли текущий имвол '-'
            mov al, '_'
            //сравнили с текущим символом в массиве(edx - адрес текущего
элемента)
            cmp al, [edx]
            //восстановили искомый индекс
            pop eax
            //если нашли нижнее подчеркивание, следовательно за ним идет
символы замены
            je LITHERAL
            //иначе просто подолжение итерации
            jne NEXT_ITERATION

            LITHERAL :
            //проверили индекс литерала, сопрадает ли с искомым
            //увеличиваем счетчик литералов
            inc ecx
            //проверка
```

```
            cmp ecx, eax
            //если иднексы совпадают - значит нашли
            je SEARCHENG_RETURN_SAVE
            //иначе следующая итерация
            NEXT_ITERATION :
            inc ebx
            inc edx
            //длина строки с литералами, это условие выхода из цикла
            cmp ebx, 69
            //если счетсик дошел до конца строки - выходим
            je SEARCHENG_RETURN
            //иначе продолжение цикла
            jmp SEARCHENG_LOOP
            SEARCHENG_RETURN_SAVE :
            //сохраняем начало замены
            push edx
            //счетчик
            mov ebx, 0
            //смещаемся перед на 1 символ(у нас указатель сейчас на '_')
            inc edx

            //ищем длину замены
        GET_LENGTH:
            //смотрим,что сейчас в указателе
            mov al, [edx]
            //сравниваем с разделителем '_'
            cmp al, '_'
            //если совпадает - конец литерала найден
            je LENGTH_FOUND
            //иначе смещаем адрес
            inc edx
            //увеличиваем счетчик
            inc ebx
            //продолжаем цикл
            jmp GET_LENGTH
        LENGTH_FOUND:
            //если длина найдена, то она лежит в ebx, заносим ее в eax
            mov eax, ebx
            //восстанивливаем адрес начала литерала
            pop edx
            //смещаемся на 1 символ вправо
            inc edx
            //начинаем восстанавливать стек
            jmp SEARCHENG_RESTORE_STACK
        SEARCHENG_RETURN:
            //в регистр с дилной и адресом 404 -инчего не нашли
            //404- ничего не нашли
            mov eax, 404
            //404- ничего не нашли
            mov edx, 404
        SEARCHENG_RESTORE_STACK:
            //восстановление регистра из стека
            pop ecx
            //восстановление регистра из стека
            pop ebx
            //здесь ставим нужную метку возврата
            jmp RET_SERACHENG

#pragma endregion

#pragma region STRING_REPLACE
            /// <summary>
            /// Делает замену в указаной строке
            /// в edx - адрес входной строки
            /// в ecx - адрес выходной строки
            /// </summary>
            /// <returns></returns>
        STRING_REPLACE:
            //сохранение регистра в стеке
            push eax
            //сохранение регистра в стеке
```

```
        push ebx
        //счетчик символов в 0,начальное положение
        mov ebx, 0
        //начало цикла замены
        STRING_REPLACE_LOOP :
        //сохранили текущее положение в строке
        push ebx
        //заносим текущий символ строки в al
        mov al, [edx]
        //делаем поиск кирилического символа
        jmp SEARCHRUS
        //процедура отработает и вернется сюда
    RET_SEARCHRUS:
        //в eax сейчас лежит или индекс или 404(не найдено)
        cmp eax, 404
        //нашли - заменяем
        jne STRING_REPLACE_START_REPLACE
        //не нашли - копируем
        jmp STRING_REPLACE_START_COPY
        //начиаем замену
    STRING_REPLACE_START_REPLACE:
        //индекс для поиска уже в eax
        //сохраним положение в текущей строке
        push edx
        //запускаем процедуру по поиску литерала для замены
        jmp SEARCHENG
        //процедура отработает и вернется сюда
    RET_SERACHENG:
        //сейчас в eax - длина, в edx - указатель на подстроку с заменой
        //копируем все символы, смещаем указатель в ecx
        //запуск замены в самой исходной строке
    STRING_REPLACE_START_REPLACE_LOOP:
        //текущий симвов из литерала
        mov bl, [edx]
        //вставили в строку-копию outstr(они идут параллельно)
        mov[ecx], bl
        //увеличение адреса на 1 байт
        inc ecx
        //увеличение адреса на 1 байт
        inc edx
        //уменьшаем счетчик, в eax длина литерала-замены, например для z -
26, длина 2
        dec eax
        //сравниваем с 0
        cmp eax, 0
        //если больше 0 замена не завершена
        jne STRING_REPLACE_START_REPLACE_LOOP
        //как только все скопировали - идем дальше
        //не забываем восстановиться
        pop edx
        //
        jmp STRING_REPLACE_LOOP_NEXT

    STRING_REPLACE_START_COPY:
        //здесь просто копируем - замены нет
        //взяли из исходной строки символ
        mov bl, [edx]
        //вставили в выходную сртоку
        mov[ecx], bl
        //увеличили адрес в выходой строке
        inc ecx

    STRING_REPLACE_LOOP_NEXT:
        //восстаонвление регистра из стека
        pop ebx
        //увеличение счетчика текущего символа на 1
        inc ebx
        //увеличение адреса на 1
        inc edx
        //проверяем условие выход из цикла: до 80 символов
        cmp ebx, 80
```

```
            //если счетчик достиг конца строки, то выходим
            je STRING_REPLACE_STACK
            //иначе продолжаем цикл
            jmp STRING_REPLACE_LOOP

        STRING_REPLACE_STACK:
            //восстаонвление регистра из стека
            pop ebx
            //восстаонвление регистра из стека
            pop eax
            //возврат в метке
            jmp RET_STRING_REPLACE
#pragma endregion

#pragma region Точка входа
            //здесь мы начинаем
        START:
            //занесли адрес исходной строки в edx
            lea edx, intpustr
            //занесли адрес выходной строки в ecx
            lea ecx, outpustr
            //запустили процедуру замены
            jmp STRING_REPLACE
            //метка возврата
            //нужна чтобы вернуться из подфункции
        RET_STRING_REPLACE:
            //ничего не делаем, замена уже сделана

        }

        printf("Исходная строка:%s\n", intpustr);
        printf("Результат:%s\n", outpustr);
    }
    catch (exception ex) {

    }
}
```