**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Организация ЭВМ и систем»**

Тема: Представление и обработка символьной информации с использованием строковых команд

|  |  |  |
| --- | --- | --- |
| Студент гр. 8383 |  | Ларин А. |
| Преподаватель |  | Ефремов М.А, |

Санкт-Петербург

2019

**Цель работы.**

Научится обрабатывать строки на языке ассемблера. Научится использовать для этого строковые команды. Использовать полученные знания для написания обработчика строк согласно заданию.

**Основные теоретические положения.**

Часто, однако, бывает необходимо

переслать или сравнить поля данных, которые превышают по длине одно слово. Например, необходимо сравнить описания или имена для того, чтобы отсортировать их в восходящей последовательности. Элементы такого формата известны как строковые данные и могут являться как символьными, так и числовыми. Для обработки строковых данных ассемблер имеет пять команд обработки строк:

MOVS - переслать один байт или одно слово из одной области памяти в другую;

LODS - загрузить из памяти один байт в регистр AL или одно слово в регистр AX;

STOS - записать содержимое регистра AL или AX в память; - сравнить содержимое двух областей памяти, размером в один байт или в одно слово;

SCAS - сравнить содержимое регистра AL или AX с содержимым памяти.

Префикс REP позволяет этим командам обрабатывать строки любой длины.

Свойства операций над строками

Цепочечная команда может быть закодирована для повторяющейся обpаботки одного байта или одного слова за одно выполнение. Например, можно выбрать "байтовую" команду для обработки строки с нечетным числом байт или "двухбайтовую" команду для обработки четного числа байт. Ниже перечислены регистры, участвующие в цепочечных командах (для однобайтовых и двухбайтовых вариантов). Предположим, что регистры DI и SI содержат необходимые адреса:

|  |  |  |  |
| --- | --- | --- | --- |
| Команда | Операнды | Байт | Слово |
| MOVS | DI,SI | MOVSB | MOVSW |
| LODS | AL,SI или AX,SI | LODSB | LODSW |
| STOS | DI,AL или DI,AX | STOSB | STOSW |
| CMPS | SI,DI | CMPSB | CMPSW |
| SCAS | DI,AL или DI,AX | SCASB | SCASW |

REP: префикс посторения цепочной команды

Несмотря на то, что цепочечные команды имеют отношение к одному байту или одному слову, префикс REP обеспечивает повторение команды несколько раз. Префикс кодируется непосредственно перед цепочечной командой, например, REP MOVSB. Для использования префикса REP необходимо установить начальное значение в регистре CX. При выполнении цепочечной команды с префиксом REP происходит уменьшение на 1 значения в регистре CX до нуля. Таким образом, можно обрабатывать строки любой длины.

Флаг направления определяет направление повторяющейся операции:

- для направления слева направо необходимо с помощью команды CLD установить флаг DF в 0;

- для направления справа налево необходимо с помощью команды STD установить флаг DF в 1.

При выполнении команд CMPS и SCAS возможна установка флагов состояния, так чтобы операция могла прекратиться сразу после обнаружения необходимого условия. Ниже приведены модификации префикса REP для этих целей:

REP - повторять операцию, пока CX не равно 0;

REPZ или REPE - повторять операцию, пока флаг ZF показывает "равно ли ноль". Прекратить операцию при флаге ZF, указывающему на не равно или не ноль или при CX равном 0;

REPNE или REPNZ - повторять операцию, пока флаг ZF показывает "не равно или не ноль". Прекратить операцию при флаге ZF, указывающему на "равно или нуль" или при CX равным 0.

Для процессоров 8086, 80286 и 80386, обрабатывающих слово за oдно выполнение, использование цепочечных команд, где это возможно, приводит к повышению эффективности работы программы.

CMPS: сравнение строк

Команда CMPS сравнивает содержимое одной области памяти (адресуемой регистрами DS:SI) с содержимыми другой области (адресуемой как ES:DI). В зависимости от флага DF команда CMPS также увеличивает или уменьшает адреса в регистрах SI и DI на 1 для байта или на 2 для слова. Команда CMPS устанавливает флаги AF, CF, OF, PF, SF и ZF. При использовании префикса REP в регистре CX должна находиться длина сравниваемых полей. Команда CMPS может сравнивать любое число байт или слов.

SCAS: Сканирование строк

Команда SCAS отличается от команды CMPS тем, что сканирует

(просматривает) строку на определенное значение байта или слова. Команда

SCAS сравнивает содержимое области памяти (адресуемой pегистрами ES:DI) с

содержимым регистра AL или AX. В зависимости от значения флага DF команда

SCAS также увеличивает или уменьшает адрес в регистре DI на 1 для байта

или на 2 для слова. Команда SCAS устанавливает флаги AF, CF, OF, PF, SF и

ZF. При использовании префикса REP и значения длины в регистре CX команда

SCAS может сканировать строки любой длины.

Команда SCAS особенно полезна, например, в текстовых редакторах, где

программа должна сканировать строки, выполняя поиск знаков пунктуации:

точек, запятых и пробелов.

**Задание**

Разработать программу обработки символьной информации, реализующую функции:

- инициализация (вывод титульной таблички с указанием вида преобразования и автора программы) - на ЯВУ;

- ввода строки символов, длиной не более Nmax (<=80), с клавиатуры в заданную область памяти - на ЯВУ; если длина строки превышает Nmax, остальные символы следует игнорировать;

- выполнение заданного в таблице 5 преобразования исходной строки с записью результата в выходную строку - на Ассемблере;

- вывода результирующей строки символов на экран и ее запись в файл - на ЯВУ.

Ассемблерную часть программы включить в программу нa ЯВУ по принципу встраивания (in-line).

Таблица 5.

Варианты заданий вида преобразования

Вариант 13. Формирование номера введенной русской буквы по алфавиту и номера позиции его первого вхождения во входной строке и выдача их на экран.

**Выполнение**

Вариант 13

Программа реализована на ЯВУ С++. Функционал ЯВУ используется для объявления переменных, ввода/вывода информации из стандартного потока.

Программа начинается с объявления буферов-строк \_term и \_str –входных данных. Далее объявляются и инициализируются в -1 переменные alphaNum и firstОcc, предназначенные для хранения места кирилического символа в алфавите и индекс его первого вхождения. Далее на экран выводится сообщение, заполняются строки, и начинается исполнение кода ассемблера.

В первую очередь в ассемблерном коде определяется длина строки последовательным сравнением символов строки с терминальными. Далее запускается цикл посимвольной обработки строки. При помощи серии проверок определяется принадлежность символа к кирилице, определяется и сохраняется его место в алфавите и индекс вхождения. Далее при помощи ЯВУ результаты выводятся на экран

**Тестирование.**

|  |  |
| --- | --- |
| Mnogo russкih bуkv | Первое вхождение русского символа: 10  Номер по алфавиту: 12 |
| CTPAHHAR CTPOKA | Русских букв нет |
| Шнат !5 Тн!5 | Первое вхождение русского символа: 0  Номер по алфавиту: 26 |
| 6ёквё ‘ё’ мёжёт с лёгкёстьё зёмёнёть лёбёё дрёгёё нёчтёжнёё глёснёё бёквё | Первое вхождение русского символа: 1  Номер по алфавиту: 7 |

**Выводы.**

В результате работы были разобраны некоторые базовые концепции языка ассемблера. Были изучены строковые команды ассемблера, освоена программная обработка строк. Была написана рабочая программа для обработки строк.

Приложение

LR4.cpp

#include "pch.h"

#include <iostream>

#include <stdio.h>

#define N 80

#define T 0

using namespace std;

int main()

{

system("chcp 1251 > nul");

char \_str[N + 1];

char msg[] = "ЛР4. Ларин Антон 8383. 13) Формирование номера введенной русской буквы по алфавиту и номера позиции его первого вхождения во входной строке и выдача их на экран.\n";

cout << msg;

int i = 0;

char alphaNum=-1;

int firstOcc=-1;

cin.getline(\_str, N);

\_asm {

sub eax, eax;

mov al, T; al <- termenate symbol

mov ecx,N;

lea edi,\_str;

repne scas; cmps;

sub ecx, N;

not ecx; ecx < -length(\_str)

mov edx, ecx; edx <- ecx (length)

sub edi,edi

cycle:

mov edi, edx; edi <- current index defined by counter

sub edi, ecx;

mov al, \_str[edi]; al <- current symbol

cmp al, 'Ё'; Self-explanatory

je isYo;

cmp al, 'ё';

je isYo;

cmp al, 'А'; In bounds. Otherwise - not cyrilic

jl notCyrilic;

cmp al, 'я';

jg notCyrilic;

isCyrilic:

mov firstOcc, edi; firstOcc <- current sybmol index (first cyrilic occurance)

cmp al, 'а'; а+ are lower, Я- are upper

jge isLower;

isUpper:

cmp al, 'Е';

jle beforeYo;

inc al; Letters after ё are one position bigger

beforeYo:

sub al, 'А'; Get alphabetical position

inc al; Muggles are counting from one

mov alphaNum, al; alphaNum <-alphabetical position

jmp hooray; Thie end

isLower:

cmp al, 'е';

jle beforeYo\_;

inc al;

beforeYo\_:

sub al, 'а';

inc al;

mov alphaNum, al;

jmp hooray;

isYo:;Dё yёё rёmёmbёr mё?

mov alphaNum, 7; Truly magic number

mov firstOcc, edi;

jmp hooray;

notCyrilic:

loop cycle;

hooray:

}

if (firstOcc != -1) {

cout << "Первое вхождение русского символа: " << firstOcc << endl;

cout << "Номер по алфавиту: " << (int)alphaNum<< endl;;

}

else {

cout << "Русских букв нет" << endl;

}

return 0;

}