Министерство образования Республики Беларусь

Учреждение Образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Лабораторная работа №2

по дисциплине «Программирование на языке ассемблера»

на тему «Обработка символьных данных»

вариант №8

|  |  |  |
| --- | --- | --- |
| Выполнил:  студент гр. 250504  Солодков М. Д |  | Проверил:  Туровец Н.О. |

Минск 2023

**Цель работы:** Ознакомиться с директивами определения данных, изучить команды пересылки данных и передачи управления, изучить строчные операции и прерывания консольного ввода-вывода высокого уровня.

**Теоретические сведения**

Для выполнения работы требуется рассмотреть следующие элементы

языка ассемблера и операционной системы:

1. Директивы определения данных.

Директивы определения данных указывают ассемблеру, что в соответствующем месте программы располагается переменная, определяют тип переменной (байт, слово и т.д.), задают ее начальное значение и ставят в соответствие переменной метку, которая будет использоваться для обращения к этим данным.

Определения данных записываются в общем виде следующим образом:

метка D\* значение

Здесь D\* является определением типа и может быть задано как:

-- DB – байт,

-- DW – слово (2 байта),

-- DD – двойное слово (4 байта),

-- DF – 6 байт (для представления адреса (FAR указатель)),

-- DQ – 8 байт,

-- DT – 10 байт (80-битные данные для FPU).

Для работы с символами и строками символов в данной работе достаточно типа DB. Для работы с числовыми данными (например, индексами символов в строке) лучше использовать тип DW, т.к. длина строки может превышать размер в 255 символов.

2. Команды пересылки данных и способы адресации.

Базовой командой пересылки данных является команда MOV: MOV приемник, источник. Эта команда копирует содержимое источника в приемник, источник при этом не изменяется. Команда MOV действует аналогично операторам присваивания из языков высокого уровня. В качестве источника для MOV могут использоваться: число (непосредственный операнд), регистр общего назначения, сегментный регистр или переменная (операнд, находящийся в памяти). В качестве приемника для MOV могут использоваться: регистр общего назначения, сегментный регистр (кроме CS) или переменная.

Оба операнда должны быть одного и того же размера – байт, слово или

двойное слово. Нельзя выполнять пересылку данных с помощью MOV из одной переменной в другую, из одного сегментного регистра в другой и нельзя помещать в сегментный регистр непосредственный операнд – эти операции выполняют только двумя командами MOV.

Также в данной работе можно использовать стек – это специальным образом организованный участок памяти, используемый для временного хранения переменных, для передачи параметров вызываемым подпрограммам и для сохранения адреса возврата при вызове процедур и прерываний. Данные можно записывать и считывать только с вершины стека. Таким образом, если записать в стек числа 1, 2, 3, то при чтении они будут получаться в обратном порядке — 3, 2, 1.

Для работы со стеком используются команды:

-- PUSH источник – поместить данные в стек,

-- POP приемник – считать данные из стека.

Способы адресации определяют формирование адреса памяти для доступа к данным. Для правильной адресации по умолчанию (без явного указания сегментного регистра) требуется следующее:

-- регистр CS должен указывать на начало сегмента кода – команды переходов всегда используют этот сегментный регистр;

-- регистр SS должен указывать на начало сегмента стека – если для косвенной адресации используется регистр BP, то это адресация к стеку.

-- регистр DS должен указывать на начало сегмента данных – адресация к данным по умолчанию (кроме BP) использует этот сегментный регистр.

3. Команды передачи управления.

Команды передачи управления служат для организации ветвления вычислительного процесса.

Предлагается использовать следующие команды этой группы:

-- безусловный переход (JMP метка) – переход на метку без возврата (от текущего положения до 32768 байт). Для перехода в диапазоне 128 байт от текущего места можно использовать команду JMP SHORT метка.

-- условный переход (Jcc метка, где cc – условие перехода, обычно используется после команды CMP) – переход в зависимости от состояния флагов, которые обычно устанавливаются предыдущей арифметической или логической операцией. Флаги, проверяемые командой, кодируются в ее мнемонике (например, JC – переход, если установлен флаг CF). Сокращения «L» (less – меньше) и «G» (greater – больше) применяются для сравнения целых чисел со знаком, а «A» (above – над) и «B» (below – под) для сравнения целых чисел без знака (см. таблицу 2).

-- переход, если CX = 0 (JCXZ метка).

Для организации условных переходов достаточно часто используется команда сравнения (CMP источник, приемник), которая сравнивает два числа, вычитая второе из первого, но не сохраняет результат, а лишь устанавливает в соответствии с результатом флаги состояния.

4. Строковые операции.

Кроме перечисленных выше базовых команд пересылки данных, для об-

работки строк символов можно использовать специальные строковые операции. Каждая строковая операция, представлена в процессоре двумя видами команд, различающихся по последнему символу мнемоники команды:

-- B (byte) – для обработки строк, состоящих из символов-байтов (как в данной лабораторной работе),

-- W (word) – для обработки строк, состоящих из символов-слов.

Если флаг направления DF перед выполнением команды строковой обработки установлен в 0 (выполнена команда CLD), то значение в индексном регистре автоматически увеличивается, если в 1 (выполнена команда STD) –уменьшается. Индексные регистры уменьшаются или увеличиваются на 1, если команды работают с байтами, или на 2 – при работе со словами.

Строковые операции обеспечивают выполнение следующих операций:

-- сравнение строк (CMPS) – команда сравнивает значение элемента одной строки (DS:SI) со значением элемента второй строки (ES:DI) и устанавливает индексных значения регистров на следующие элементы строк. Сравнение происходит так же, как и по команде сравнения CMP. Результатом операции является установка флагов.

-- сканирование строки (SCAS) – команда производит сравнение содержимого аккумулятора (AL или AX) с байтом или словом памяти, абсолютный адрес которого определяется парой ES:DI, после чего регистр DI устанавливается на следующий символ. Команда SCAS используется обычно для поиска в строке (ES:DI) элемента заданного в аккумуляторе.

-- пересылка строки (MOVS) – пересылает поэлементно строку DS:SI в строку ES:DI и устанавливает значения индексных регистров на следующий элемент строки.

-- запись в строку (STOS) – заполняет строку, содержащуюся по адресу ES:DI, элементом из аккумулятора (AL или AX), не влияет на флаги.

-- чтение из строки (LODS) – записывает в аккумулятор (AL или AX) содержимое ячейки памяти, адрес которой задается регистрами DS:SI, не влияет на флаги.

Команды строковой обработки чаще всего используются с однобайтными префиксами (префиксами повторения), которые обеспечивают многократное автоматическое повторение выполнения команды:

-- повторять, пока равно (REPE),

-- повторять, пока ноль (REPZ),

-- повторять (REP),

-- повторять, пока не равно (REPNE),

-- повторять, пока не ноль (REPNZ).

Префиксы повторения ставятся перед строковыми командами обязательно в той же строке. Префикс использует регистр CX как счетчик циклов. На каждом этапе цикла выполняются следующие действия:

1) если CX=0, то выход из цикла и переход к следующей команде;

2) выполнение заданной строковой операции;

3) уменьшение CX на единицу, флаги при этом не изменяются;

4) выход из цикла, если:

a) условие сравнения не выполняется для SCAS или CMPS;

b) префикс REPE и ZF=0 (последнее сравнение не совпало);

c) префикс REPNE и ZF=1 (последнее сравнение совпало).

5. Прерывания ввода-вывода.

Прерывания ввода-вывода – специальные команды передачи управления, вызывающие функции BIOS или DOS, предоставляющие сервис по работе с аппаратурой ПЭВМ.

Для организации ввода данных с клавиатуры предлагается использовать одну из ниже приведенных функций DOS:

-- Функция DOS 01h (INT 21h) – считать символ из STDIN с эхом, ожиданием и проверкой на Ctrl-Break:

Ввод: АН = 01h

Вывод: AL = ASCII-код символа или 0.

Если AL = 0, то второй вызов этой функции возвратит в AL расширенный ASCII-код символа

Особенности: При чтении с помощью этой функции введенный символ автоматически немедленно отображается на экране (посылается в устройство STDOUT, так что его можно перенаправить в файл). При нажатии Ctrl-C или Ctrl-Break выполняется команда INT 23h. Если нажата клавиша, не соответствующая какомунибудь символу (стрелки, функциональные клавиши Ins, Del и т.д.), то в AL возвращается 0 и функцию надо вызвать еще один раз, чтобы получить расширенный ASCII-код.

-- Функция DOS 06h (INT 21h) – считать символ из STDIN без эха, без ожидания и без проверки на Ctrl-Break.

-- Функция DOS 07h (INT 21h) – считать символ из STDIN без эха, с ожиданием и без проверки на Ctrl-Break.

-- Функция DOS 08h (INT 21h) – считать символ из STDIN без эха, с ожиданием и проверкой на Ctrl-Break.

-- Функция DOS 0Ah (INT 21h) – считать строку символов из STDIN в буфер.

-- Функция DOS 02h (INT 21h) – записать символ в STDOUT с проверкой

на Ctrl-Break.

-- Функция DOS 09h (INT 21h) – записать строку в STDOUT с проверкой

на Ctrl-Break.

-- Функция DOS 40h (INT 21h) – записать строку в файл или устройство.

6. Макросы.

Макросом называется фрагмент программы, который подставляется в код программы всякий раз, когда ассемблер встречает его имя в тексте программы. Макрос начинается именем и директивой MACRO, а заканчивается директивой ENDM. После директивы MACRO могут быть перечислены через запятую идентификаторы параметров, используемых в макросе, что делает макрос гибким средством оформления кода.

**Код программы (.exe)**

.8086

.model small

.stack 100h

.data

input\_str db "Enter the string: $"

input\_word db "Enter the word: $"

word\_input db 16 dup('$')

str db 100 dup('$')

newline db 0Dh, 0Ah, '$'

number\_k db 3

start\_of\_the\_word dw ?

buff1 dw ?

buff2 dw ?

.code

input\_num proc

mov ah, 1

int 21h

sub al, '0'

mov number\_k, al

mov ah, 02h

mov dx, 0Ah

int 21h

mov ah, 02h

mov dx, 0Dh

int 21h

ret

input\_num endp

reverse\_string proc

push bp

mov bp, sp

mov si, [bp + 4]

mov bx, start\_of\_the\_word

mov cx, 100

xor ax, ax

xor dx, dx

loop1:

mov dl, [si + bx]

cmp dl, '$'

je end\_loop

push dx

inc bx

loop loop1

end\_loop:

sub bx, start\_of\_the\_word

mov cx, bx

mov bx, start\_of\_the\_word

mov si, [bp + 4]

loop2:

pop dx

mov [si + bx ], dl

inc bx

loop loop2

pop bp

ret 2

reverse\_string endp

if\_letter proc

cmp al, 'A'

jl not\_letter

cmp al, 'Z'

jbe its\_letter

cmp al, 'a'

jl not\_letter

cmp al, 'z'

jbe its\_letter

not\_letter:

mov al, 0

ret

its\_letter:

mov al, 1

ret

if\_letter endp

find\_k\_word proc

push bp

mov bp, sp

mov ch, number\_k ; number of string to find

mov si, [bp + 4] ; entered string

mov bx, 0 ; index

mov ah, 0 ; counter of words

mov dh, 0 ; flag

find\_loop:

mov al, [si + bx]

mov dl, al

call if\_letter

cmp al, 1

je word\_start

mov dh, 0

continue:

cmp ah, ch

jne not\_k\_word

mov di, bx

pop bp

ret 2

not\_word\_start:

mov dh, 1

inc ah

jmp continue

word\_start:

cmp dh, 0

je not\_word\_start

not\_k\_word:

inc bx

cmp dl, '$'

jne find\_loop

mov di, 65h

pop bp

ret 2

find\_k\_word endp

start:

mov ax, @data

mov ds, ax

mov ah, 09h

mov dx, offset input\_str

int 21h

mov ah, 0Ah

lea dx, str

int 21h

xor ax, ax

mov al, [str + 1]

mov bx, offset str + 2

add bx, ax

mov al, '$'

mov byte [bx - 1], al

mov ah, 09h

mov dx, offset newline

int 21h

mov ah, 09h

mov dx, offset input\_word

int 21h

call input\_num

dec number\_k

push offset str + 2

call find\_k\_word

mov start\_of\_the\_word, bx

inc number\_k

push offset str + 2

call find\_k\_word

mov buff1, bx

mov ax, start\_of\_the\_word

sub bx, ax

mov buff1, bx

push offset str + 2

call reverse\_string

mov al, [str + 1]

xor ah, ah

sub ax, buff1

mov bx, ax

inc bx

inc bx

mov [str + bx], '$'

mov ah, 09h

mov dx, offset newline

int 21h

push offset str + 2

call reverse\_string

mov ah, 09h

lea dx, str + 2

int 21h

mov ax, 4C00h

int 21h

end start

**Вывод программы**

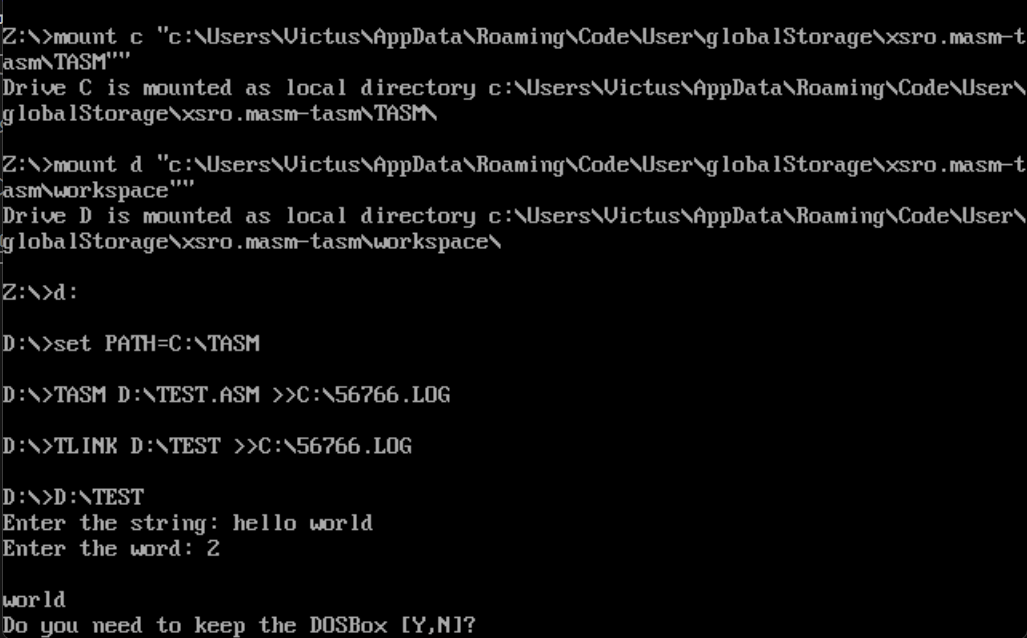


Рисунок 1 – Результат работы программы