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 «Конструирование программ и языки программирования»

на тему «Обработка символьных данных»

вариант №6

|  |  |  |
| --- | --- | --- |
| Выполнил  Лагодич И.Р. №250504 |  | 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)**

data segment

str\_input db "Input string: ", "$"

buf db 200 dup ("$")

str db 0Ah, 0Dh, "$"

a\_len dw 0

b\_len dw 0

a\_spos dw 0

a\_epos dw 0

b\_spos dw 0

b\_epos dw 0

i\_spos dw 0

i\_epos dw 0

j\_spos dw 0

j\_epos dw 0

i dw 0

j dw 0

amid dw 0

sub\_len dw 0

cur\_pos dw 0

cur\_word dw 0

words\_num dw 0

a\_buf db 1, "$"

data ends

stack segment

dw 100h dup(0)

stack ends

print\_str macro buf

mov dx, offset str

mov ah, 09h

int 21h

mov ah, 09h

mov dx, offset buf + 2

int 21h

endm

code segment

start:

mov ax, @data

mov ds, ax

lea dx, str\_input

mov ah, 09h

int 21h

mov dx, offset buf

mov ah, 0Ah

int 21h

mov dx, offset str

mov ah, 09h

int 21h

xor al, al

mov si, 0

mov cur\_pos, 2

mov words\_num, 0

words\_counting:

mov si, cur\_pos

mov al, buf[si]

cmp al, 13

je end\_count

cmp al, '$'

je end\_count

inc cur\_pos

cmp al, 'A'

jl words\_counting

cmp al, 'z'

ja words\_counting

cmp al, 'Z'

jle A\_Z\_b

cmp al, 'a'

jae a\_z\_l

A\_Z\_b:

mov si, cur\_pos

mov al, buf[si]

cmp al, 13

je A\_Z\_b\_cont

cmp al, ' '

jne words\_counting

A\_Z\_b\_cont:

inc words\_num

jmp words\_counting

a\_z\_l:

mov si, cur\_pos

mov al, buf[si]

cmp al, 13

je a\_z\_l\_cont

cmp al, ' '

jne words\_counting

a\_z\_l\_cont:

inc words\_num

jmp words\_counting

end\_count:

xor si, si

mov cur\_word, 0

mov cur\_pos, 2

mov a\_len, 0

mov b\_len, 0

mov i, 0

mov j, 0

jmp find\_word1

start\_sorting\_i:

inc i

mov cur\_word, 1

mov si, a\_epos

inc si

mov cur\_pos, si

mov i\_spos, 0

mov j\_epos, 2

mov a\_len, 0

mov a\_spos, 0

mov a\_epos, 0

mov b\_spos, 0

mov b\_epos, 0

mov b\_len, 0

find\_word1:

mov si, i

mov di, words\_num

cmp si, di

je ending

mov si, cur\_pos

mov al, buf[si]

cmp al, 13

je start\_sorting\_i

cmp al, '$'

je start\_sorting\_i

cmp al, ' '

jne word1\_len

inc cur\_pos

jmp find\_word1

start\_sorting\_j:

mov b\_len, 0

mov si, j\_epos

mov cur\_pos, si

find\_word2:

mov si, cur\_word

mov di, words\_num

cmp si, di

ja start\_sorting\_i

mov si, cur\_pos

mov al, buf[si]

cmp al, 13

je start\_sorting\_i

cmp al, '$'

je start\_sorting\_i

cmp al, ' '

jne word2\_len

inc cur\_pos

;inc j\_epos

jmp find\_word2

word1\_len:

inc cur\_word

inc j

mov bx, cur\_pos

mov i\_spos, bx

mov a\_spos, bx

mov si, i\_spos

mov al, buf[si]

while\_len1:

cmp al, ' '

je end\_while\_len1

cmp al, '$'

je end\_while\_len1

cmp al, 13

je end\_while\_len1

mov a\_epos, si

mov i\_epos, si

inc cur\_pos

inc si

inc a\_len

mov al, buf[si]

jmp while\_len1

end\_while\_len1:

jmp find\_word2

word2\_len:

inc cur\_word

inc j

mov bx, cur\_pos

mov j\_spos, bx

mov j\_epos, bx

mov b\_spos, bx

mov si, j\_spos

mov al, buf[si]

while\_len2:

cmp al, ' '

je end\_while\_len2

cmp al, '$'

je end\_while\_len2

cmp al, 13

je end\_while\_len2

mov b\_epos, si

mov j\_epos, si

inc cur\_pos

inc si

inc b\_len

mov al, buf[si]

jmp while\_len2

end\_while\_len2:

comparing:

mov si, a\_len

mov di, b\_len

cmp si, di

je a\_b\_equal

cmp si, di

ja a\_longer

jmp b\_longer

a\_b\_equal:

mov cx, a\_len

mov si, a\_spos

mov di, b\_spos

compare\_a\_b\_equal:

mov al, buf[si]

mov bl, buf[di]

cmp al, bl

jb a\_longer\_sort\_end

cmp al, bl

ja a\_longer\_sort

inc si

inc di

loop compare\_a\_b\_equal

jmp a\_longer\_sort

a\_longer:

mov cx, b\_len

mov si, a\_spos

mov di, b\_spos

compare\_b\_a:

mov al, buf[si]

mov bl, buf[di]

cmp al, bl

jb a\_longer\_sort\_end

cmp al, bl

ja a\_longer\_sort

inc si

inc di

loop compare\_b\_a

ja a\_longer\_sort

b\_longer:

mov cx, a\_len

mov si, a\_spos

mov di, b\_spos

compare\_a\_b:

mov al, buf[si]

mov bl, buf[di]

cmp al, bl

jb b\_longer\_sort\_end

cmp al, bl

ja b\_longer\_sort

inc si

inc di

loop compare\_a\_b

jmp b\_longer\_sort\_end

a\_longer\_sort:

mov cx, a\_len

mov si, a\_spos

a\_sort\_a\_stack:

mov bx, offset buf

mov al, buf[si]

push ax

inc si

loop a\_sort\_a\_stack

mov cx, b\_len

mov si, b\_spos

a\_sort\_b\_stack:

mov bx, offset buf

mov al, buf[si]

push ax

inc si

loop a\_sort\_b\_stack

mov si, a\_epos

mov di, b\_spos

sub di, si

dec di

mov amid, di

cmp di, 0

je a\_longer\_sort\_ab\_len\_swap

mov cx, di

mov di, b\_spos

dec di

a\_longer\_sort\_amid\_stack\_push:

mov bx, offset buf

mov al, [bx+di]

push ax

dec di

loop a\_longer\_sort\_amid\_stack\_push

a\_longer\_sort\_ab\_len\_swap:

mov si, a\_len

mov di, b\_len

sub si, di

mov sub\_len, si

mov di, b\_spos

sub di, si

mov b\_spos, di

mov di, si

mov si, a\_epos

sub si, di

mov a\_epos, si

mov di, amid

cmp di, 0

je a\_longer\_sort\_amid\_shift\_end

mov cx, di

mov si, a\_epos

inc si

a\_longer\_sort\_amid\_shift:

pop ax

mov buf[si], al

inc si

loop a\_longer\_sort\_amid\_shift

a\_longer\_sort\_amid\_shift\_end:

mov si, sub\_len

mov di, b\_len

add di, si

mov b\_len, di

mov di, a\_len

sub di, si

mov a\_len, di

mov cx, a\_len

mov si, a\_epos

a\_longer\_sort\_new\_a:

pop ax

mov buf[si], al

dec si

loop a\_longer\_sort\_new\_a

mov cx, b\_len

mov si, b\_epos

a\_longer\_sort\_new\_b:

pop ax

mov buf[si], al

dec si

loop a\_longer\_sort\_new\_b

a\_longer\_sort\_end:

mov si, b\_epos

inc si

mov j\_epos, si

jmp start\_sorting\_j

b\_longer\_sort:

mov cx, a\_len

mov si, a\_spos

b\_sort\_a\_stack:

mov bx, offset buf

mov al, buf[si]

push ax

inc si

loop b\_sort\_a\_stack

mov cx, b\_len

mov si, b\_spos

b\_sort\_b\_stack:

mov bx, offset buf

mov al, buf[si]

push ax

inc si

loop b\_sort\_b\_stack

mov si, a\_epos

mov di, b\_spos

sub di, si

dec di

mov amid, di

cmp di, 0

je b\_longer\_sort\_ab\_len\_swap

mov cx, di

mov di, b\_spos

dec di

b\_longer\_sort\_amid\_stack\_push:

mov bx, offset buf

mov al, [bx+di]

push ax

dec di

loop b\_longer\_sort\_amid\_stack\_push

b\_longer\_sort\_ab\_len\_swap:

mov si, b\_len

mov di, a\_len

sub si, di

mov sub\_len, si

mov di, b\_spos

add di, si

mov b\_spos, di

mov di, si

mov si, a\_epos

add si, di

mov a\_epos, si

mov di, amid

cmp di, 0

je b\_longer\_sort\_amid\_shift\_end

mov cx, di

mov si, a\_epos

inc si

b\_longer\_sort\_amid\_shift:

pop ax

mov buf[si], al

inc si

loop b\_longer\_sort\_amid\_shift

b\_longer\_sort\_amid\_shift\_end:

mov si, sub\_len

mov di, a\_len

add di, si

mov a\_len, di

mov di, b\_len

sub di, si

mov b\_len, di

mov cx, a\_len

mov si, a\_epos

b\_longer\_sort\_new\_a:

pop ax

mov buf[si], al

dec si

loop b\_longer\_sort\_new\_a

mov cx, b\_len

mov si, b\_epos

b\_longer\_sort\_new\_b:

pop ax

mov buf[si], al

dec si

loop b\_longer\_sort\_new\_b

b\_longer\_sort\_end:

mov si, b\_epos

inc si

mov j\_epos, si

jmp start\_sorting\_j

ending:

print\_str (buf)

mov ax, 4C00h

int 21h

code ends

end start

**Вывод программы**

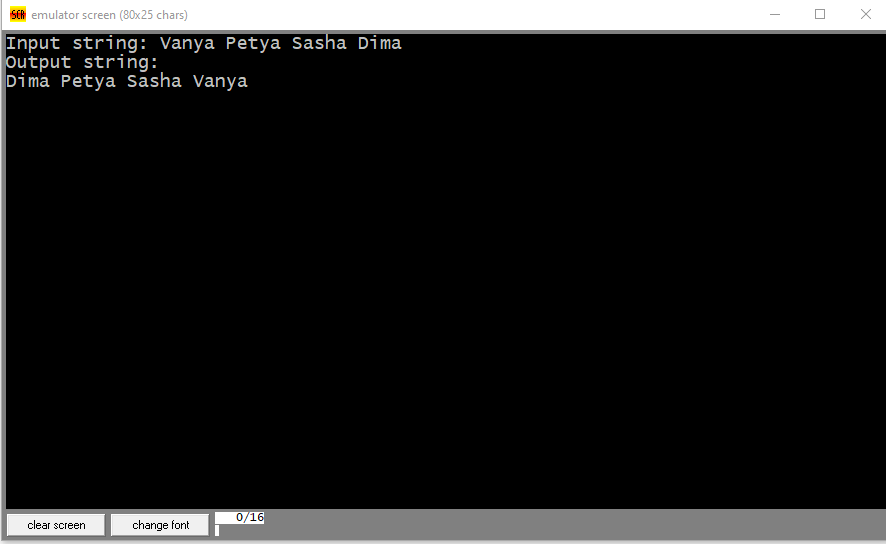
****

Рисунок 1 – Результат работы программы