МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

# Кафедра проектирования информационно-компьютерных систем

Отчет по предмету

Программирование на языке ассемблера

Лабораторная работа №2

**«ОБРАБОТКА СИМВОЛЬНЫХ ДАННЫХ»**

|  |  |
| --- | --- |
| **Выполнил:**  Студент группы 150501  Смоленский Н.О. | **Проверил:**  Туровец Н.О. |

Минск 2022

**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:**

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).

Поле значения в определении переменной может содержать одно или несколько чисел, строк символов (взятых в одиночные («'») или двойные кавычки («"»)), операторов ? и DUP, разделенных запятыми.

Имя переменной будет соответствовать адресу первого из указанных значений. Если вместо точного значения указан символ «?», то такая переменная считается неинициализированной и ее значение на момент запуска программы может оказаться любым. Если нужно заполнить участок памяти повторяющимися данными, то вместо указания точного значения используется специальный оператор DUP, имеющий формат

*счетчик* DUP *(значение)*

где счетчик – число задающее число элементов.

Директива EQU предназначена для описания констант:

*имя* EQU *операнд*

*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.

Основные способы адресации данных

|  |  |  |
| --- | --- | --- |
| Способ адресации | Пример | Действие |
| Регистровая | MOV AX,BX | AX = BX |
| Непосредственная | MOV AX,4 | AX = 4 |
| Прямая | MOV AX,ES:0001  MOV AX,var | AX = слово по адресу ES:0001  AX = слово по адресу DS:var |
| Косвенная (базовая  или индексная) | MOV AX,[BX] MOV AX,ES:[BX] | AX = слово по адресу DS:BX AX = слово по адресу ES:BX |
| Относительная косвенная | MOV AX,[BX+2] MOV AX,2[BX]  MOV AX,[BX]+2 | AX = слово по адресу DS:(BX+2) |
| По базе с индекси- рованием | MOV AX,[BX+SI+2] MOV AX,2[BX][SI]  MOV AX,[BX][SI]+2  MOV AX,[BX+2][SI] | AX = слово по адресу DS:(BX+SI+2) |

Операторы адресации:

-- SEG выражение – сегментный адрес;

-- OFFSET выражение – смещение;

-- THIS тип – текущий адрес (TASM и MASM);

-- тип PTR выражение – переопределение типа;

-- SMALL выражение – 16-ти битное смещение (TASM);

-- SHORT выражение — 8-ми битное смещение.

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.

-- 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.

Команды условных переходов

|  |  |  |
| --- | --- | --- |
| Команда | Условие для команды CMP | Реальное условие сравнения |
| JA  JNBE | если выше  если не ниже или равно | CF = 0 и ZF = 0 |
| JAE JNB  JNC | если выше или равно если не ниже  если нет переноса | CF = 0 |
| JB JNAE  JC | если ниже  если не выше или равно если перенос | CF = 1 |
| JBE JNA | если ниже или равно  если не выше | CF = 1 и ZF = 1 |
| JE  JZ | если равно  если ноль | ZF = 1 |
| JG  JNLE | если больше  если не меньше или равно | ZF = 0 и SF = OF |
| JGE  JNL | если больше или равно  если не меньше | SF = OF |
| JL  JNGE | если меньше  если не больше или равно | SF <> OF |
| JLE  JNG | если меньше или равно  если не больше | ZF = 1 и SF <> OF |
| JNE JNZ | если не равно  если не ноль | ZF = 0 |
| JNO | если нет переполнения | OF = 0 |
| JO | если есть переполнение | OF = 1 |
| JNP  JPO | если нет четности  если нечетное | PF = 0 |
| JP  JPE | если есть четность  если четное | PF = 1 |
| JNS | если нет знака | SF = 0 |
| JS | если есть знак | SF = 1 |

-- цикл (LOOPxx метка) – организация циклов в программах, используя регистр CX в качестве счетчика цикла. Команда LOOP уменьшает содержимое CX на 1 и передает управление на указанную метку, если содержимое CX не равно 0, в противном случае выполняется команда, следующая за LOOP. Команда LOOPE (цикл пока равно) завершает цикл, если регистр CX достиг нуля или если не установлен флаг нуля. Команда LOOPNE (цикл пока не равно) осуществляет выход из цикла, если регистр CX достиг нуля или установлен флаг нуля.

4. *Строковые операции.*

Кроме перечисленных выше базовых команд пересылки данных, для обработки 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 дан- ной 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:
   1. условие сравнения не выполняется для SCAS или CMPS;
   2. префикс REPE и ZF=0 (последнее сравнение не совпало);
   3. префикс REPNE и ZF=1 (последнее сравнение совпало);
5. изменение значения индексных регистров в соответствии со значением флага направления DF и переход на начало цикла.

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.

-- Функция DOS *06h* (INT 21h) – считать символ из STDIN без эха, без ожидания 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.

Для вывода данных на консоль предлагается использовать одну из ниже приведенных функций DOS:

-- Функция DOS *02h* (INT 21h) – записать символ в STDOUT с проверкой на Ctrl-Break.

-- Функция DOS *06h* (INT 21h) – записать символ в STDOUT без проверки 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:**

*.model tiny*

*.code*

*org 100h*

*jmp start*

*Output\_string proc*

*mov ah,9*

*int 21h*

*ret*

*Output\_string endp*

*Input\_string macro str*

*lea dx,str*

*mov ah,0Ah*

*int 21h*

*endm*

*Exit proc*

*mov dx,offset newl*

*call Output\_string*

*mov dx,offset result*

*call Output\_string*

*mov dx,offset string[2]*

*call Output\_string*

*mov dx,offset newl*

*call Output\_string*

*mov ax,4C00h*

*int 21h*

*ret*

*Exit endp*

*Empty\_exit proc*

*mov dx,offset newl*

*call Output\_string*

*mov ax,4C00h*

*int 21h*

*ret*

*Empty\_exit endp*

*Delete proc*

*inc si*

*mov dl,[si]*

*dec si*

*mov [si],dl*

*inc si*

*cmp [si],0Dh*

*jne call Delete*

*mov si,bx*

*cmp [si],' '*

*jne call Check*

*ret*

*Delete endp*

*Check proc*

*cmp [si],0Dh*

*jne call Delete*

*ret*

*Check endp*

*start:*

*mov dx,offset msg1*

*call Output\_string*

*Input\_string string*

*mov dx,offset newl*

*call Output\_string*

*mov dx,offset msg3*

*call Output\_string*

*Input\_string word*

*cmp string[1],0*

*je isEmpty*

*cmp word[1],0*

*je isEmpty*

*isEmpty:*

*mov ah,[string[1]]*

*cmp ah,0*

*je String\_is\_empty*

*mov ah,[word[1]]*

*cmp ah,0*

*je Word\_is\_empty*

*jne Find*

*String\_is\_empty:*

*mov dx,offset newl*

*call Output\_string*

*mov dx,offset msg6*

*call Output\_string*

*call Empty\_exit*

*Word\_is\_empty:*

*mov dx,offset newl*

*call Output\_string*

*mov dx,offset msg7*

*call Output\_string*

*call Empty\_exit*

*Find:*

*mov si,offset string[1]*

*mov di,offset word[2]*

*jmp Find\_symbol*

*Find\_symbol:*

*inc si*

*cmp [si],0Dh*

*je notFound*

*mov dl,[si]*

*cmp [si],' '*

*je Find\_symbol*

*cmp dl,0Dh*

*je call Exit*

*cmp dl,[di]*

*jne Skip\_word*

*mov bx,si*

*jmp Find\_word*

*loop Find\_symbol*

*Skip\_word:*

*cmp [si],' '*

*je Find\_symbol*

*cmp [si],0Dh*

*je notFound*

*inc si*

*loop Skip\_word*

*Find\_word:*

*inc si*

*inc di*

*cmp [di],0Dh*

*je isEnd*

*mov dl,[si]*

*cmp dl,[di]*

*je loop Find\_word*

*jmp Next\_word*

*isEnd:*

*cmp [si],' '*

*je Found*

*cmp [si],0Dh*

*je Found*

*Next\_word:*

*mov di,offset word[2]*

*jmp Skip\_word*

*notFound:*

*mov dx,offset newl*

*call Output\_string*

*mov dx,offset msg5*

*call Output\_string*

*call Exit*

*Found:*

*inc si*

*mov dx,offset newl*

*call Output\_string*

*jmp Delete\_word*

*Delete\_word:*

*dec bx*

*mov si,bx*

*call Delete*

*call Exit*

*string db 200,200 dup ('$')*

*word db 200,200 dup ('$')*

*msg1 db "Enter the string: $"*

*msg2 db "Entered string: $"*

*msg3 db "Enter the word to delete: $"*

*msg4 db "Entered word: $"*

*msg5 db "Word not found!$"*

*msg6 db "String is empty$"*

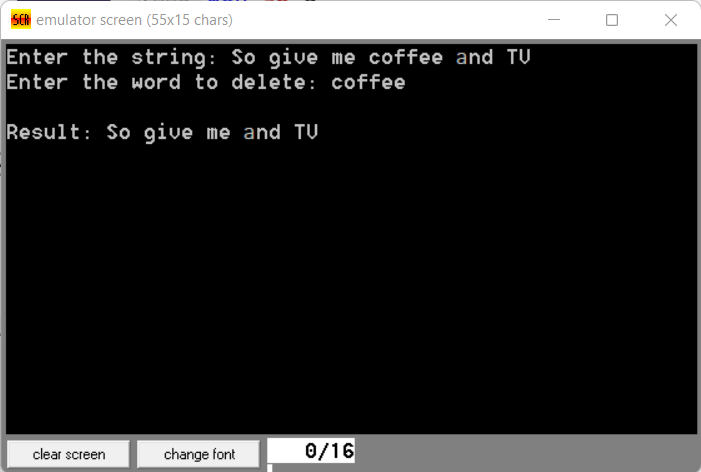
*msg7 db "Word is empty$"*

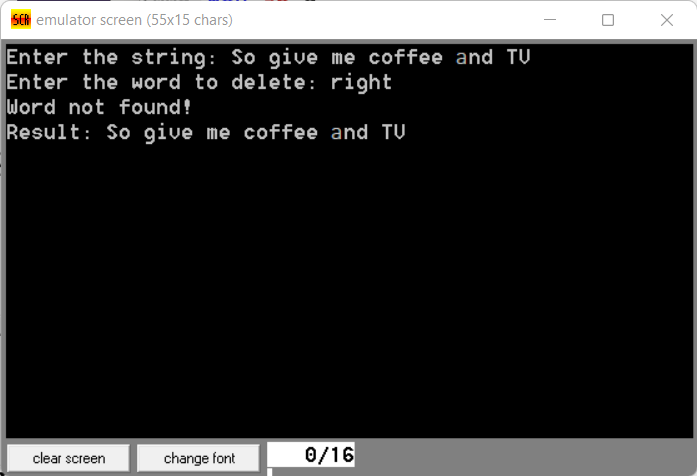
*result db "Result: $"*

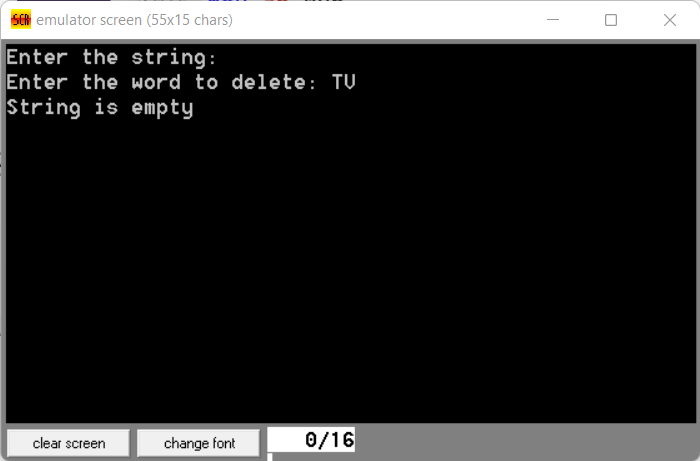
*newl db 0Dh,0Ah,'$'*

*end start*

**Результат работы:**

****

****

****

**Вывод**

В ходе выполнения лабораторной работы было проведено ознакомление с директивами определения данных, командами пересылки данных и передачи управления, изучены строчные операции и прерывания консольного ввода-вывода.