МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждения высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ

**ЛАБОРАТОРНАЯ РАБОТА № 3**

по дисциплине

**«Машинно-ориентированное программирование»**

на тему:

**«Использование логических команд»**

*Вариант № 2*

Выполнил:

Студент группы

КТбо2-8

|  |  |  |
| --- | --- | --- |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Жалнин Д. И. |
|  | *подпись* |  |
|  |  |  |

Проверил:

ассистент кафедры

МОП ЭВМ

|  |  |  |
| --- | --- | --- |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Гуляев Н. А. |
|  | *подпись* |  |

Оценка

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Таганрог 2020

# **1 ЦЕЛЬ РАБОТЫ**

* 1. **Дидактическая цель работы**

Ознакомление с основными методами составления программ, используемых циклические алгоритмы работы, на языке «Ассемблер», а также с наиболее часто используемыми для этого мнемониками.

**1.2 Практическая цель работы**

В рамках лабораторной работы необходимо разработать программу на языке ассемблера, алгоритм которой выполняет задачу согласно описанному индивидуальному заданию, скомпилировать и запустить код программы с помощью программного пакета «TASM».

# **2 ВАРИАНТ ЗАДАНИЯ**

**2.1 Общие требования**

Для всех вариантов требуется выполнить разработку программного модуля при помощи СРПО «Turbo Assembler», реализующего некоторую обработку массива, состоящего из байтов или 2-байтных слов. Выполнить отладку разработанного программного модуля при помощи программного отладчика «Turbo Debugger».

**2.2 Индивидуальное задание, вариант № 3**

Дан массив из 8 байтов. Рассматривая его как массив из 64 бит, посчитать количество единиц.

# **3 ХОД РАБОТЫ**

**3.1 Описание высокоуровневой реализации**

В ход работы была составлена программа на языке программирования «Ассемблер», используя модель памяти «**small**», в котором допускается наличие одного сегмента кода и единственного сегмента данных. Размер стека программы – 256 байт.

Перед началом основного сегмента кода происходит инициализация:

1. Переменной «**COUNT**», которая хранит в себе количество найденных по ходу работы программы единиц.
2. Массива «**ARR**», представляющий из себя массив из 8 байт.

Алгоритм работы программы:

1. В регистр **CX**, который является счетчиком для цикла **loop** помещается число 8, что является длиной обрабатываемого массива: **mov CX 8**.
2. В регистр **BX** помещается указатель на первый элемент массива **ARR**: **lea BX ARR**.
3. Устанавливаем метку **loop1** для работы внешнего цикла.
4. Так как при вызове команды **loop**, переменной счетчиком будет являться регистр CX, для реализации внутреннего цикла необходимо сохранять положение счетчика внешнего цикла. Сохранение происходит в стеке программы с помощью команды

**push CX**.

1. После сохранения значения регистра **CX**, в него помещается число 8 – количество бит в одном, рассматриваемом во внутреннем цикле байте:

**mov CX, 8**.

1. Устанавливается метка **loop2** для внутреннего цикла.
2. В регистр **AX** помещаем число 1 с помощью команды

**mov AX,1**.

1. Производим логическую операцию «и» с регистром **AX** и значением массива, указатель на которое хранится в регистре **BX**:

**and AX, [BX]**.

1. Если первый байт элемента массива равен 1, то в **AX** останется переданная в него 1, если же нет, то значением регистра будет являться 0. Помещаем в переменную **COUNT** значение регистра **AX**:

**add COUNT, AX**.

1. В регистр **AL** вносим значение массива по указателю в **BX**, производим логический сдвиг вправо на 1 единицу и вносим значение из **AL** обратно в **BX**:

**mov AL, [BX]**

**shr AL, 1**

**mov [BX], AL**.

1. Переходим к метке loop2 с помощью команды **loop loop2** и повторяем шаги 7-11 пока значения счетчика в регистре **CX** не будет равно 0 – выполняем цикл 8 раз.
2. Смещаем указатель на элемент массива в регистре **AX** на 1:

**add BX, 1**.

1. Достаем из стека сохраненной значение и передаем его в регистр **CX**:

**pop CX**.

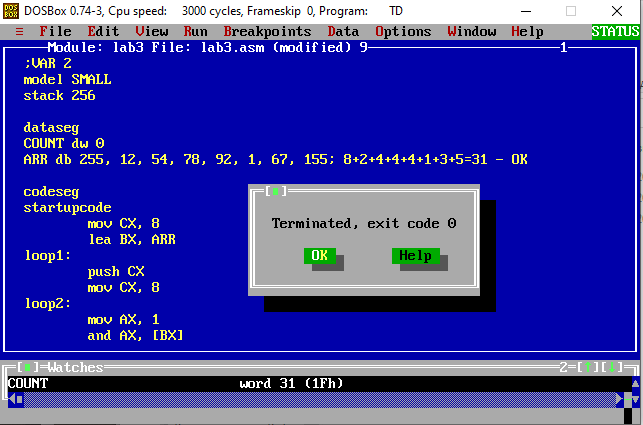
1. Переходим к метке **loop1** командой **loop loop1**. Повторяем шаги 3-14 еще 7 раз.

В конце работы программы в переменной **COUNT** будет находиться число, равное количеству единичных битов в последовательности байтов **ARR**.

Полный листинг программы расположен в *«Приложение А».*

**3.4 Описание полученных результатов**

Программный модуль был скомпилирован, запущен и отлажен в рамках среды «DOS BOX». При вызове «TASM» были заданы ключи «-L -ZI», которые позволили получить отладочные файлы. При вызове «TLINK» были использованы ключи «-V». С целью отладки был запущен отладчик «TD», в котором было проведено пошаговое исполнение программы. На *рисунках 1 и 2* можно наблюдать результат выполнения программы в отладчике – значение переменной, которую необходимо вычислить расположено в секции «Watches». Результат работы программы совпал с ожидаемым – программа отработала корректно.

**

*Рисунок 1 – результат работы программы в «Turbo Debugger»*

# **4 ВЫВОДЫ**

**4.1 Полученные знания, навыки, умения**

В ходе выполнения лабораторной работы была разработан и отлажен программный модуль, который считает количество единиц в массиве байт из 8 элементов. Получены навыки работы со стеком, с вложенными циклами и с основным логическими командами в программах на языке «Ассемблер».

**ПРИЛОЖЕНИЕ А**

model SMALL

stack 256

dataseg

COUNT dw 0

ARR db 255, 12, 54, 78, 92, 1, 67, 155; 8+2+4+4+4+1+3+5=31 - OK

codeseg

startupcode

mov CX, 8

lea BX, ARR

loop1:

push CX

mov CX, 8

loop2:

mov AX, 1

and AX, [BX]

add COUNT, AX

mov AL, [BX]

shr AL, 1

mov [BX], AL

loop loop2

add BX, 1

pop CX

loop loop1

QUIT: exitcode 0

end