Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра безопасности информационных систем (БИС)

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕР

Отчет по лабораторной работе №2

по дисциплине «Системное программирование»

Студент гр.748

\_\_\_\_\_\_\_\_\_\_\_\_А.А. Мухачева

. .2022

Принял

Преподаватель каф. КИБЭВС

Е.О. Калинин

. .2022

Томск 2022

**Введение**

Целью данной лабораторной работы является знакомство со структурой программы на языке Ассемблер, разновидностями и назначением сегментов, способами организации простых и сложных типов данных, изучение формата и правил работы с транслятором GAS и средствами создания программ на Ассемблере для ОС Linux.

**1 Ход работы**

Запуск программ, работа с отладчиком, сравнение по времени выполнения и по памяти производились в Docker. Поэтому был написан dockerfile, в котором в качестве образа внутри контейнера использовалась Ubuntu, скопированы три программы (hello world на ассемблере, две программы по варианту – на с++ и на ассемблере). Также установлены пакеты для работы с ассемблером и языком высокого уровня, отладчик gdb и текстовый редактор nano.

FROM ubuntu

COPY helloworld.s .

COPY var16.s .

COPY var\_16.cpp .

RUN apt-get update

RUN apt-get install -y nano

RUN apt-get install -y binutils

RUN apt-get update

RUN apt-get install -y gcc-multilib

RUN apt-get install -y g++

RUN apt-get install -y gdb

Далее была выполнена команда docker build –t lab2 . для создания образа на основе Dockerfile. Результат представлен на рисунке 1.1.

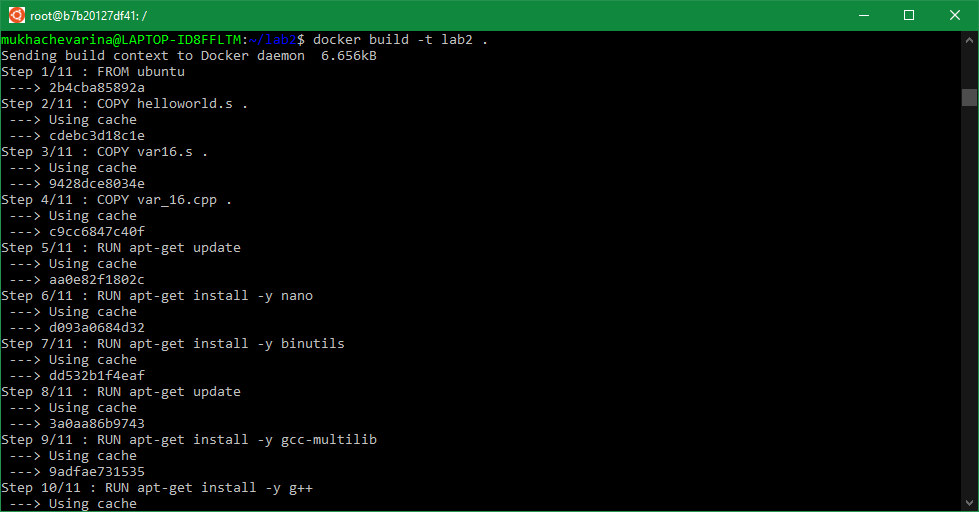


Рисунок 1.1 – Создание образа

Далее посредствам команды docker images был проверен список всех образов, для того чтобы узнать ID созданного образа. И на его основе был запущен контейнер (команда docker run –it lab2).

Внутри контейнера первым делом была написана программа на языке ассемблер, результатом выполнения которой являлась строка «Hello, world» (рисунок 1.2).

Команда для запуска: gcc -c helloworld.s && ld helloworld.o && ./a.out

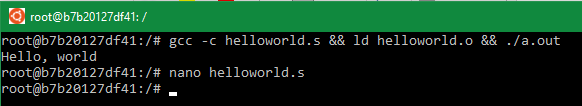


Рисунок 1.2 – Выполнение hello-world на ассемблере

С помощью последовательного выполнения команд gcc –m32 var16.s –o var16 –g и ./var16был скомпилирован код, и запущен файл программы по варианту на ассемблере. Результат представлен на рисунке 1.3. Программа, написанная согласно индивидуальному варианту на ассемблере, представлена в приложении А.

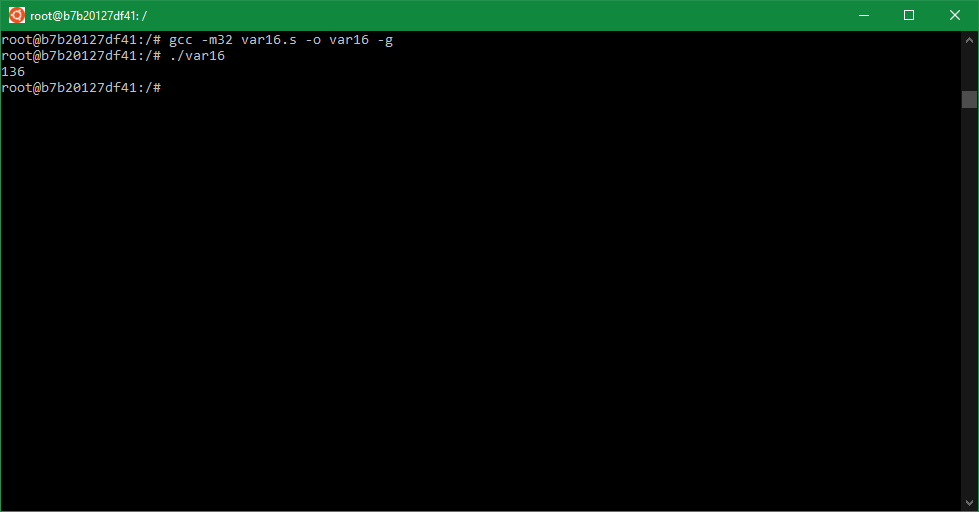


Рисунок 1.3 – Выполнение программы по варианту на ассемблере

Далее на высоком языке программирования была написана программа, выполняющая те же действия, а именно находит сумму элементов массива, которые были сдвинуты логически вправо, в том случае если они имели 1 в 7 бите (рисунок 1.4). Код программы представлен в приложении Б.

Команды: g++ var\_16.cpp -o var\_16; ./var\_16

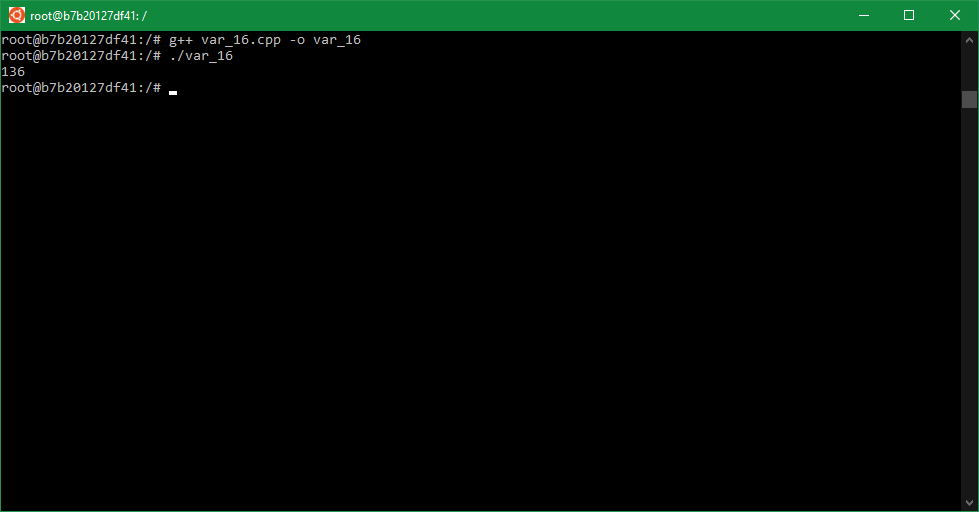


Рисунок 1.4 – Выполнение программы на С++

По итогу обе программы получают одинаковый результат. Далее был запущен отладчик (рисунок 1.5).

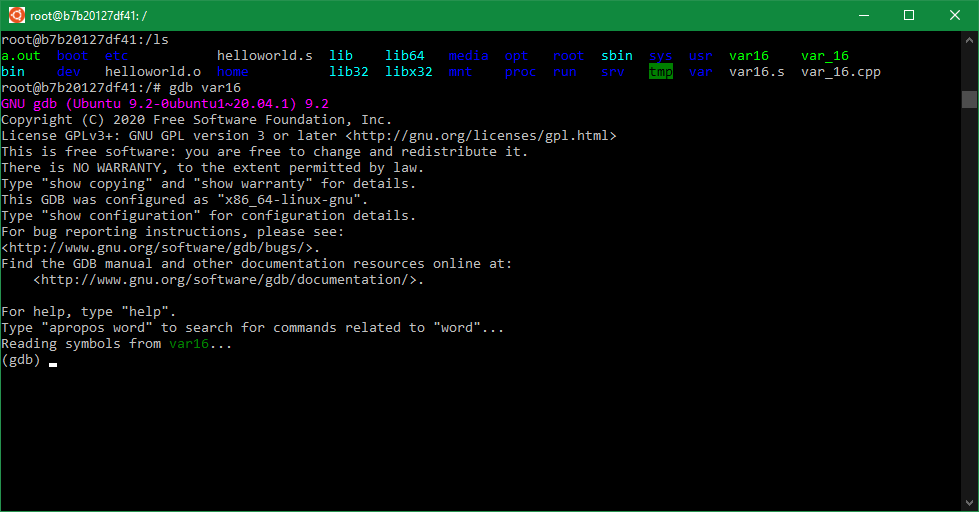


Рисунок 1.5 – Запуск GDB

Далее, при помощи команды break main, была поставлена точка останова в функции main (break main) и запущена на исполнение с помощью команды run (рисунок 1.6).

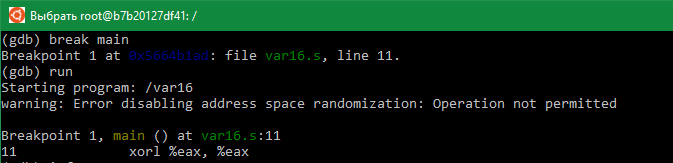


Рисунок 1.6 – Точка останова

Далее была выполнена команда info reg, которая отображает содержимое регистров (рисунок 1.7).

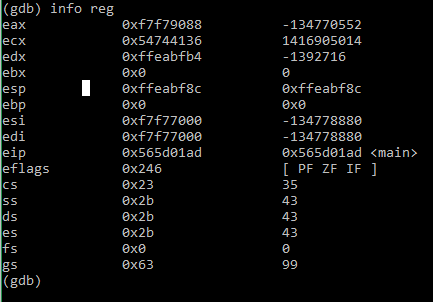


Рисунок 1.7 – Содержимое регистров

Далее было осуществлено передвижение по командам в программе с помощью s. С помощью команды i r eax было определено состояние регистра. Результат представлен на рисунках 1.8 и 1.9.

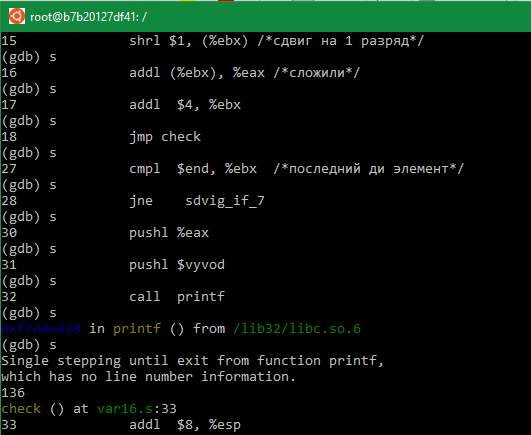


Рисунок 1.8 – Передвижение по программе



Рисунок 1.9 – Предоставление информации о состоянии регистра

В конце была выполнена команда disassemble, было произведено дизассемблирование кода (рисунок 1.10).

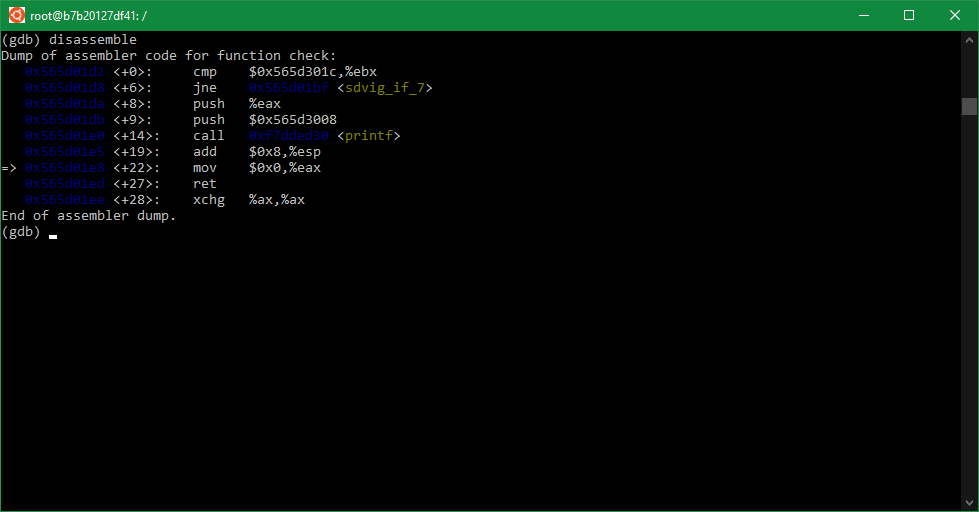


Рисунок 1.10 – Дизассемблирование кода

Далее для программы, написанной на С, была выставлена аналогичная точка останова в функции main и проведено дизассемблирование выбранного участка кода (рисунки 1.11-1.13).

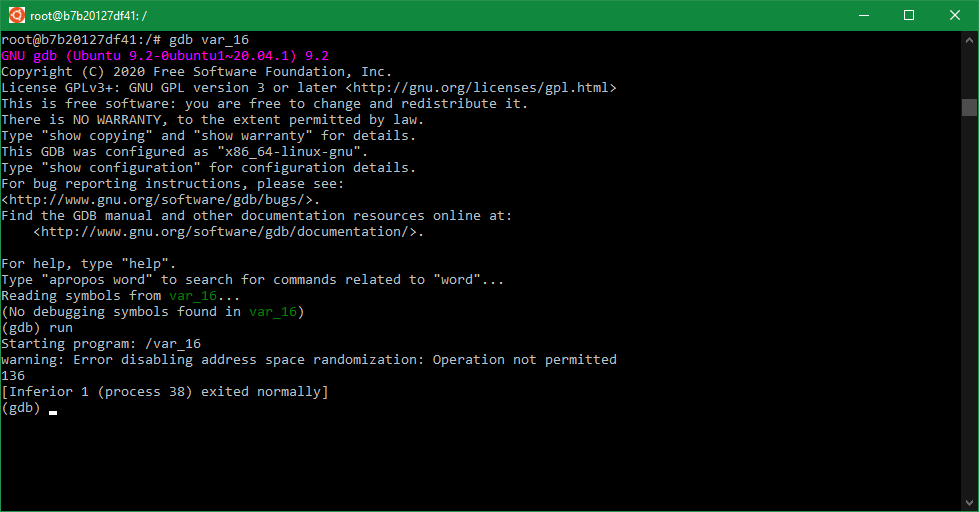


Рисунок 1.11 – Точка останова

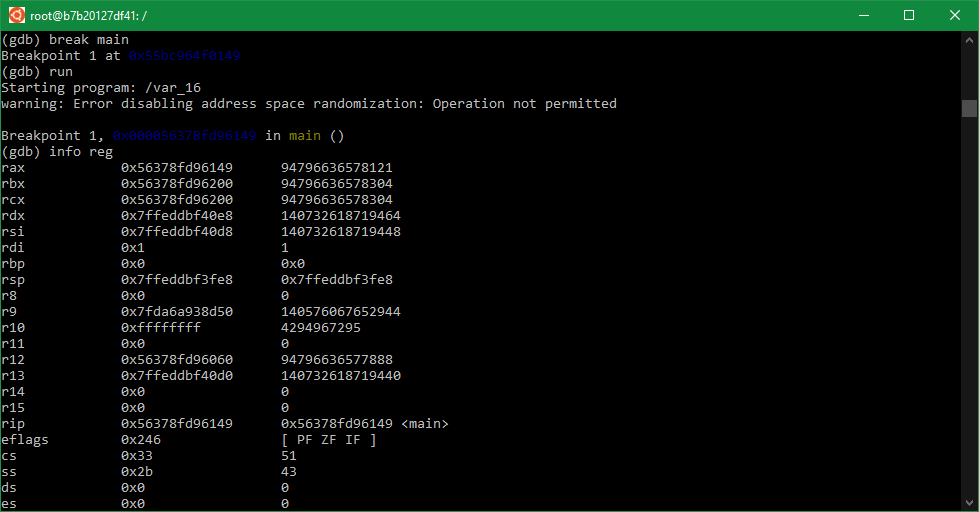


Рисунок 1.12 – Содержимое регистров

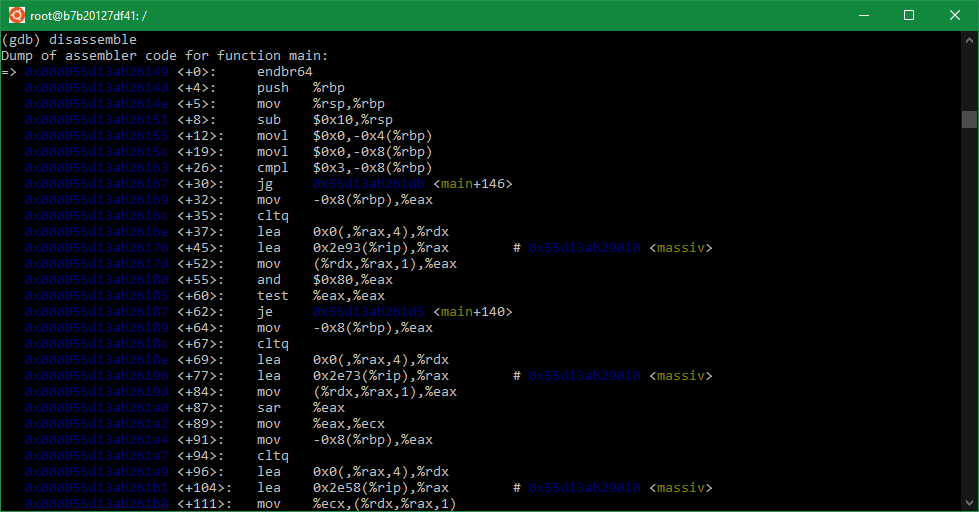


Рисунок 1.13 – Дизассемблирование кода

Далее было выполнено сравнение по времени выполнения программ (рисунок 1.14) и по объему (рисунок 1.15).

Команды: time ./var16; time ./var\_16

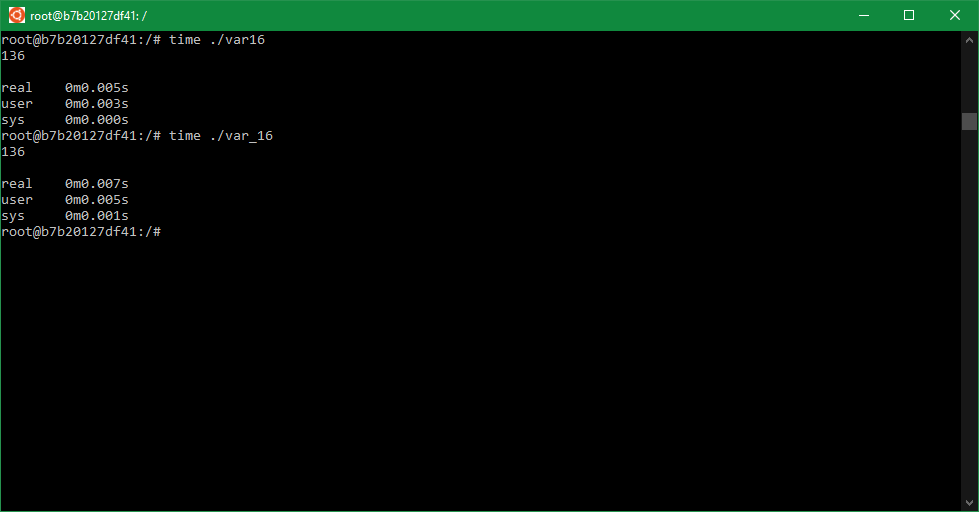


Рисунок 1.14 – Сравнение по времени

Команды: ls -s -h var16; ls -s -h var\_16

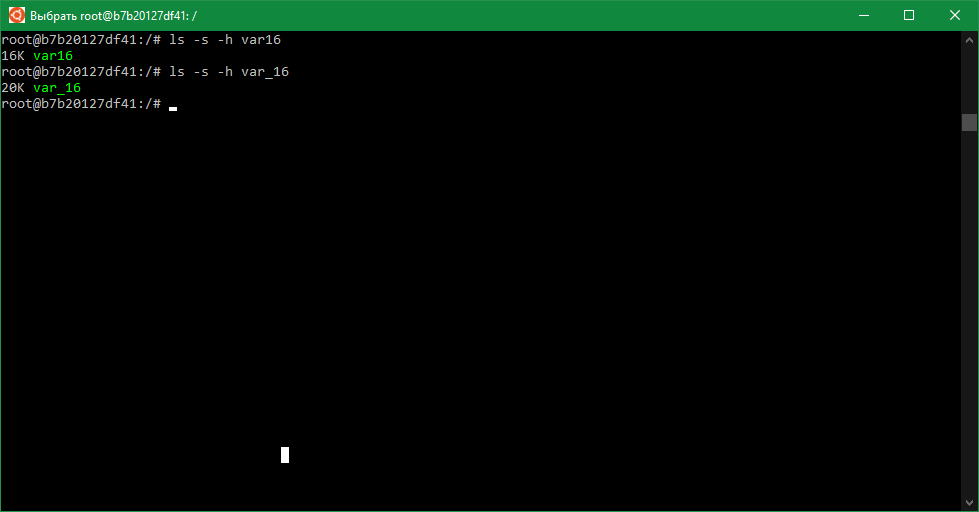


Рисунок 1.15 – Сравнение по объему

Было выяснено, что дизассемблированная программа, написанная на С++ больше по объему, чем код, написанный на Assembler.

**Заключение**

В результате выполнения данной лабораторной работы была изучена структура программ на языке Assembler, изучены форматы и правила работы с компилятором GCC и отладчиком GDB. Так же были написаны программы по индивидуальному заданию на языках Assembler и С++. В результате работы были получены выводы о том, что программа, реализованная на Assembler, исполняется быстрее и имеет меньший объём, нежели программа, написанная на C++.

**Приложение А**

(обязательное)

Код программы на Assembler

.data

vyvod:

.string "%d\n"

massiv:

.long 128, 10, 12, 144 /\*определение содержимого массива\*/

end:

.text

.global main

main:

xorl $eax, %eax /\*eax – будет записана сумма\*/

movl $massiv, %ebx /\*ebx - адрес текущего элемента массива\*/

jmp check

sum\_massiv:

shrl $1, (%ebx) /\* сдвиг логически вправо на 1 разряд\*/

addl (%ebx), %eax /\* складываем результат в %eax\*/

addl $4, %ebx

jne check

sdvig\_if\_7:

movl (%ebx), %ecx

andl $128, %ecx

cmpl $128, %ecx /\* проверка наличия 1 в 7 бите\*/

je sum\_massiv /\* переход если равно\*/

addl $4, %ebx /\*выделяем место и переходим на следующий элемент, путём добавления бит\*/

check:

cmpl $end, %ebx /\*последний ли элемент (адрес)\*/

je sdvig\_if\_7 /\*если это последний элемент, проверяем на наличие 1\*/

pushl %eax

pushl $vyvod

call printf

addl $8, %esp

movl $0, %eax

ret

**Приложение Б**

(обязательное)

Код программы на C++

#include <stdio.h> #include <stdlib.h>

int array [5] = {67,99,3,58, 114};

int main ()

{

int i;

int summ=0;

for (i=0; i!=5; i++)

{ if ((array[i]&128)==128)

{array[i]=array[i]>>1; summ=summ+array[i];

}}

printf ("%i\n", sum); return 0;

}