Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Факультет безопасности (ФБ)

Кафедра комплексной информационной безопасности электронно-вычислительных систем (КИБЭВС)

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕР

Отчет по лабораторной работе № 2

по дисциплине «Системное программирование»

Выполнил:

Студент гр.726

\_\_\_\_\_\_\_\_\_\_ М.И. Васильева

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

Принял:

к.т.н. доцент кафедры КИБЭВС

\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_ Я.Е. Мещеряков

оценка

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

дата

1. Введение

Цель работы: познакомиться со структурой программы на языке Ассемблер, разновидностями и назначением сегментов, способами организации простых и сложных типов данных, изучить форматы и правила работы с транслятором NASM, компоновщиком GCC и отладчиком GDB.

Ход работы: изучение теоретической части, подготовка образа операционной системы Linux для Docker, настройка окружающей среды, позволяющей программировать на языке Ассемблер (NASM), разработка простой программы на языке Ассемблер, компилирование написанной программы с помощью NASM, разработка программы согласно варианту, отладка написанной программы с помощью GDB.

Согласно номеру в списке (8) были выбран следующий вариант задания:

Дан массив из 10 слов. Инвертировать все отрицательные числа и найти сумму элементов полученного массива.

1. Ход работы
   1. Теоретические сведения
      1. Ассемблер

Язык Ассемблера – язык программирования, который представляет собой символьную форму машинного языка с рядом возможностей языков высокого уровня.

В 1949 году Морис Уилкс создал программу, которая позволяла писать команды в удобной для человека форме, и сама переводила их в машинный код. Уилкс называл программу собирающей системой или «ассемблером» от английского глагола «assemble» - собирать.

Вместо двоичного кода в языке ассемблера использовались буквы, цифры или сокращения, которые отражали суть команды. Например, команда «Mov Ax, 6» на языке ассемблера означала «передвинь в ячейку памяти «Ах» число 6». Такой новый подход существенно упростил написание программ, так как запоминать команды стало значительно проще, чем запоминать ряды единиц и нулей.

На сегодняшний день существует множество языков ассемблера, подходящих под конкретную модель процессора и удобство разработчика. Однако большинство разработчиков придерживаются общих традиционных подходов. Основные такие стандарты – Intel-синтаксис и AT&T – синтаксис.

Общий формат записи инструкций одинаков для обоих стандартов:

[метка:] опкод. [операнды] [; комментарий], где «опкод.» и есть собственно ассемблерная команда, мнемоника инструкции процессору. В качестве операндов могут выступать константы, названия регистров, адреса в оперативной памяти и так далее. Различия между стандартами Intel и AT&T касаются в основном порядка перечисления операндов и их синтаксиса при разных методах адресации.

Ниже приведены некоторые примеры типичных операндов языка ассемблера (таблица 2.1).

Таблица 2.1 – Примеры операндов

|  |  |
| --- | --- |
| Пример | Описание |
| inc count | Увеличить переменную памяти count |
| mov total, 48 | Переместить значение 48 в переменную памяти total |
| add ah, bh | Добавить содержимое регистра bh в регистр ah |
| and mask1, 128 | Выполнить операцию and на переменной mask1 и 128 |
| add marks, 10 | Добавить 10 к переменной marks |
| mov al, 48 | Перенести значение 10 в регистр al |

* + 1. NASM

NASM – это свободный ассемблер для архитектуры Intel x86. Используется для написания 16-, 32- и 64-разрядных программ. В NASM используется Intel-синтаксис записи инструкций. Предложение языка ассемблера NASM (строка программы) может состоять из следующих элементов: метка инструкция операнды комментарии.

Операнды отделяются между собой запятой. Перед строкой и после инструкции можно использовать любое количество пробельных символов. Комментарий начинается с точки с запятой, а концом комментария считается конец строки. В качестве инструкции может использоваться команда или псевдокоманда (директива компилятора).

NASM компилирует программы под различные операционные системы в пределах x86-совместимых процессоров. Компиляция программ в NASM состоит из двух этапов. Первый — ассемблирование, второй — компоновка. На этапе ассемблирования создаётся объектный код. В нём содержится машинный код программы и данные, в соответствии с исходным кодом, но идентификаторы (переменные, символы) пока не привязаны к адресам памяти. На этапе компоновки из одного или нескольких объектных модулей создаётся исполняемый файл (программа).

* + 1. GCC

GCC – это набор компиляторов для различных языков программирования, разработанный в рамках проекта GNU. GCC является свободным программным обеспечением, распространяется фондом свободного программного обеспечения (FSF) на условиях GNU GPL и GNU LGPL. Он используется как стандартный компилятор для свободных UNIX-подобных операционных систем.

Будучи официальным компилятором системы GNU, GCC также является главным компилятором для сборки ряда других операционных систем; среди них различные варианты Linux и BSD, а также ReactOS, Mac OS X, OpenSolaris, NeXTSTEP, BeOS и Haiku.

Программа gcc, запускаемая из командной строки, представяляет собой надстройку над группой компиляторов. В зависимости от расширений имен файлов, передаваемых в качестве параметров, и дополнительных опций, gcc запускает необходимые препроцессоры, компиляторы, линкеры.

Файлы с расширением .cc или .C рассматриваются, как файлы на языке C++, файлы с расширением .c как программы на языке C, а файлы c расширением .o считаются объектными.

* 1. Практическая часть
     1. Настройка окружения

В начале выполнения практической работы необходимо настроить окружение.

Для этого необходимо запустить докерконтейнер Debian (рисунок 2.1).



Рисунок 2.1 – Запуск контейнера Debian

Так как дальнейшая работа предполагается с использованием NASM, GCC, GDB, необходимо командой «apt-get install …» загрузить необходимое компоненты (рисунок 2.2).

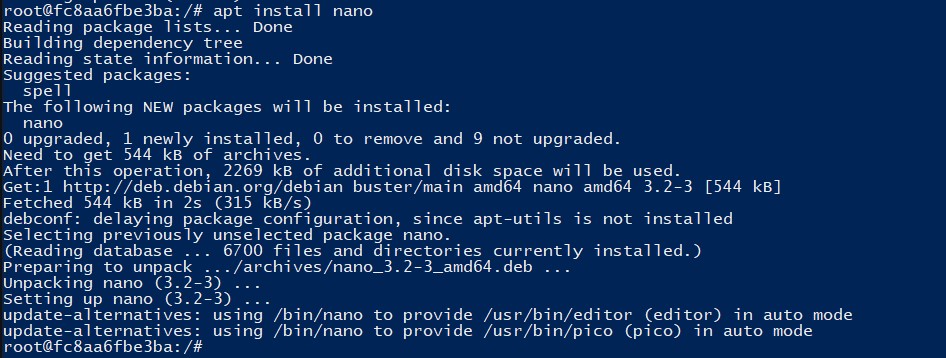


Рисунок 2.2 – Загрузка необходимых компонентов (NASM)

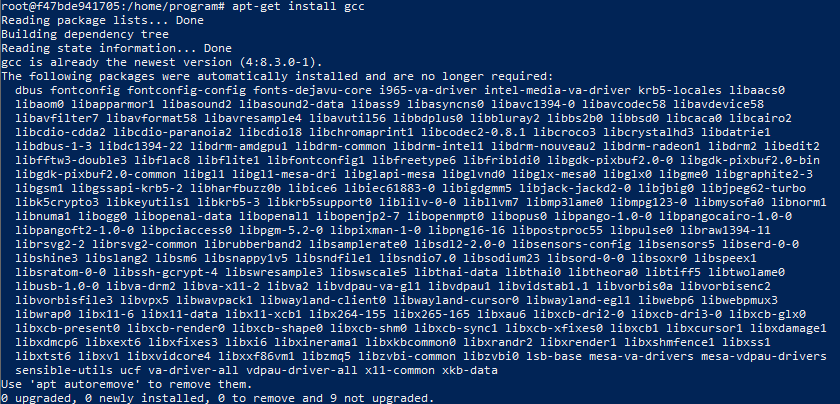


Рисунок 2.3 – Загрузка необходимых компонентов (GCC)

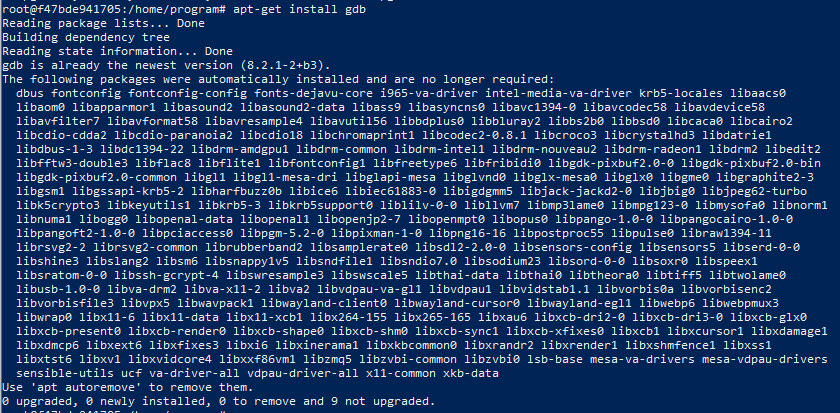


Рисунок 2.4 – Загрузка необходимых компонентов (GDB)

* + 1. Простая программа

Необходимо реализовать простую программу для ознакомления с установленными компонентами.

В качестве постой программы был выбран функционал, выводящий в командную строку «Hello 726». Листинг программы приведен в приложении А.

Результат работы программы представлен на рисунке 2.5.

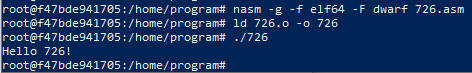


Рисунок 2.5 – Работа программы

Первая команда необходима для ассемблирования файла, которая создает объектный файл программы. Вторая команда выполняет линкование файла, данной командой части программы доводятся до конца и связываются между собой в исполняемый файл формата, понятного данной операционной системе.

* + 1. Основная программа

По заданию необходимо написать программу, которая считает сумму 10 элементов массива, и если есть отрицательные числа, то инвертирует их. Листинг программы представлен в приложении Б.

Работа программы представлена на рисунке 2.6.



Рисунок 2.6 – Работа программы

Согласно заданию не было необходимости выводить сумму элементов массива на экран. Результат выполнения программы можно просмотреть с использованием отладчика GDB.

На рисунке 2.7 представлен результат работы программы, который выводится с помощью отладчика.

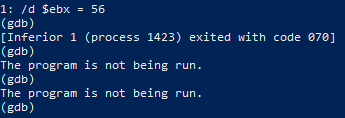


Рисунок 2.7 – Результат работы программы (GDB)

Ebx представляет собой регистр, в который заносится сумма эллементов массива. Для того, чтобы просматривать именно это регистр необходимо в отладчике gdb написать «display/d $ebx», тогда на каждом шаге выполнения программы будет выводиться значение регистра.

1. Заключение

В ходе работы был изучен язык программирования Ассемблер и отладчик GDB, написана программа на Ассемблере. Изучена работа GDB отладчика для ОС Linux. Также были получены навыки компилирования программ с помощью NASM.

Ссылка на репозиторий - <https://github.com/mariaigorevna> . Название – Assembler.

Приложение А (обязательное)

Код простой программы

section .text

global \_start

\_start:

mov edx,len

mov ecx, msg

mov ebx, 1

mov eax, 4

int 0x80

mov eax, 1

int 0x80

section .data

msg db 'Hello 726!', 0xa

len equ $ - msg

Листинг А.1 – Код простой программы

Приложение Б (обязательное)

Код основной программы

global \_start

\_start:

mass:

push rbp

mov rbp, rsp

mov DWORD [rbp-48], 1

mov DWORD [rbp-44], -3

mov DWORD [rbp-40], 3

mov DWORD [rbp-36], 4

mov DWORD [rbp-32], 5

mov DWORD [rbp-28], 6

mov DWORD [rbp-24], -7

mov DWORD [rbp-20], 8

mov DWORD [rbp-16], 9

mov DWORD [rbp-12], 10

mov DWORD [rbp-8], 0

mov DWORD [rbp-4], 0

main:

cmp DWORD [rbp-4], 9

Листинг Б.1 – Код основной программы (начало)

jg exit

mov eax, DWORD [rbp-4]

cdqe

mov eax, DWORD [rbp-48+rax\*4]

test eax, eax

jns sum

mov eax, DWORD [rbp-4]

cdqe

mov eax, DWORD [rbp-48+rax\*4]

neg eax

mov edx, eax

mov eax, DWORD [rbp-4]

cdqe

mov DWORD [rbp-48+rax\*4], edx

sum:

mov eax, DWORD [rbp-4]

cdqe

mov eax, DWORD [rbp-48+rax\*4]

add DWORD [rbp-8], eax

mov ebx, DWORD [rbp-8]

add DWORD [rbp-4], 1

Листинг Б.1 – Код основной программы (продолжение)

jmp main

exit:

mov eax, 0

pop rbp

mov eax, 1

int 0x80

jmp .L4

\_exit:

mov eax, 0

pop rbp

mov eax, 1

int 0x80

Листинг Б.1 – Код основной программы (окончание)