

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. 736-1

\_\_\_\_\_ Мирошниченко Е.Ю.

\_\_\_\_\_

Принял:

Кандидат технических наук

\_\_\_\_\_ Романов А.С.

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 GNU Assembler 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 GNU Assembler, разработка программы согласно варианту, отладка написанной программы с помощью GDB.

Согласно номеру в списке (14) был выбран следующий вариант задания:

Задача: установить 1 в 4-ых битах всех элементов массива из 10 чисел. Определить сумму элементов полученного массива.

## 2      Ход работы

### 2.1    Теоретические сведения

#### 2.1.1. 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 «Ax» 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 ebx	Увеличить переменную ebx
mov \$0, %eax	Переместить значение 0 в переменную памяти eax
add %eax, %ebx	Добавить содержимое регистра eax в регистр ebx
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

### 2.1.2. 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).

### 2.1.3. 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, представляет собой надстройку над группой компиляторов. В зависимости от расширений имен файлов, передаваемых в качестве параметров, и дополнительных опций, gcc запускает необходимые препроцессоры, компиляторы, линкеры.

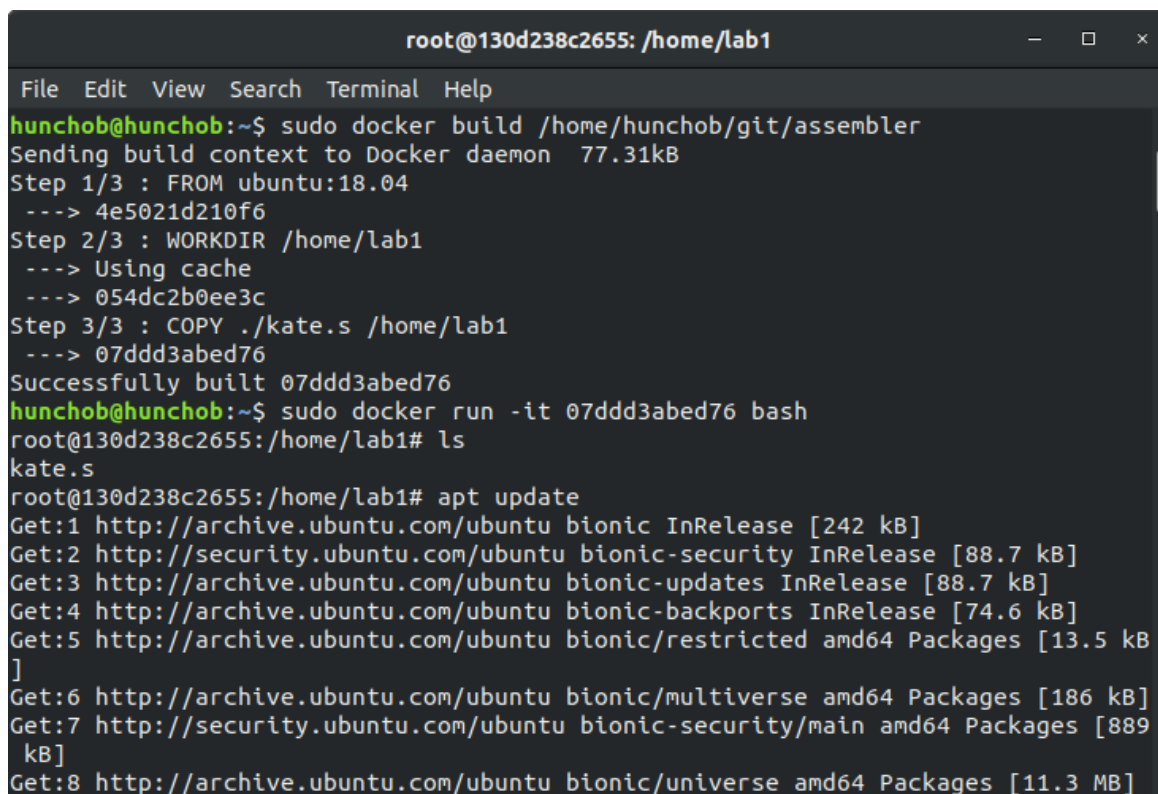
Файлы с расширением .cc или .C рассматриваются, как файлы на языке C++, файлы с расширением .c как программы на языке C, а файлы с расширением .o считаются объектными.

## 2.2 Практическая часть

### 2.2.1. Настройка окружения

В начале выполнения практической работы необходимо настроить окружение.

Для этого необходимо запустить докерконтейнер Ubuntu (рисунок 2.1).



```
root@130d238c2655: /home/lab1
File Edit View Search Terminal Help
hunchob@hunchob:~$ sudo docker build /home/hunchob/git/assembler
Sending build context to Docker daemon 77.31kB
Step 1/3 : FROM ubuntu:18.04
--> 4e5021d210f6
Step 2/3 : WORKDIR /home/lab1
--> Using cache
--> 054dc2b0ee3c
Step 3/3 : COPY ./kate.s /home/lab1
--> 07ddd3abed76
Successfully built 07ddd3abed76
hunchob@hunchob:~$ sudo docker run -it 07ddd3abed76 bash
root@130d238c2655:/home/lab1# ls
kate.s
root@130d238c2655:/home/lab1# apt update
Get:1 http://archive.ubuntu.com/ubuntu bionic InRelease [242 kB]
Get:2 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Get:3 http://archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:4 http://archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:5 http://archive.ubuntu.com/ubuntu bionic/restricted amd64 Packages [13.5 kB]
Get:6 http://archive.ubuntu.com/ubuntu bionic/multiverse amd64 Packages [186 kB]
Get:7 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages [889 kB]
Get:8 http://archive.ubuntu.com/ubuntu bionic/universe amd64 Packages [11.3 MB]
```

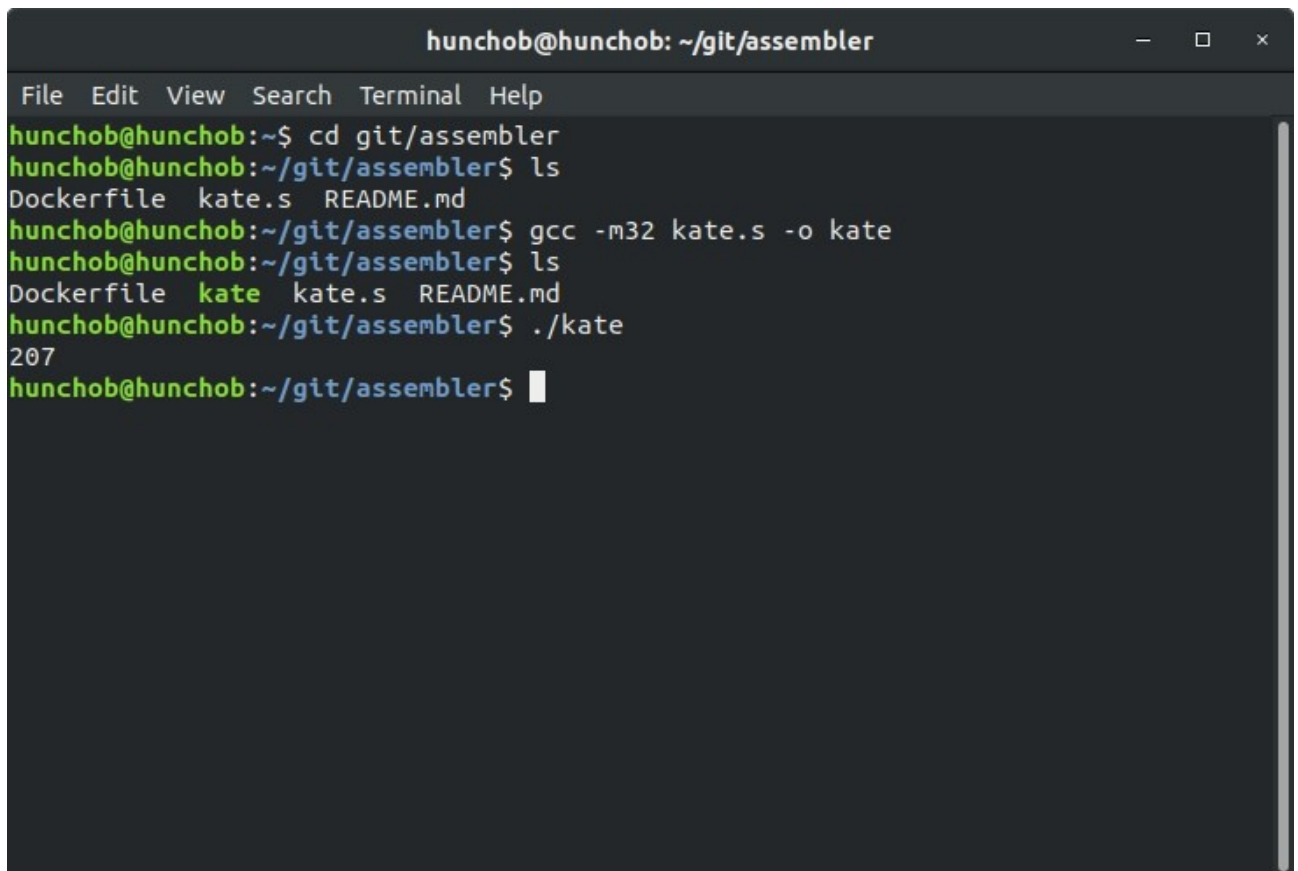
Рисунок 2.1 – Запуск контейнера Ubuntu

Так как дальнейшая работа предполагается с использованием NASM, GCC, GDB, необходимо командой «apt-get install build-essential gdb gcc-multilib» загрузить необходимые компоненты.

### 2.2.2. Основная программа

По заданию необходимо написать программу, условие которой следующее: Коды четных элементов массива необходимо сдвинуть логически влево, а коды нечетных элементов массива арифметически вправо. Листинг программы представлен в приложении А.

Работа программы представлена на рисунке 2.2.



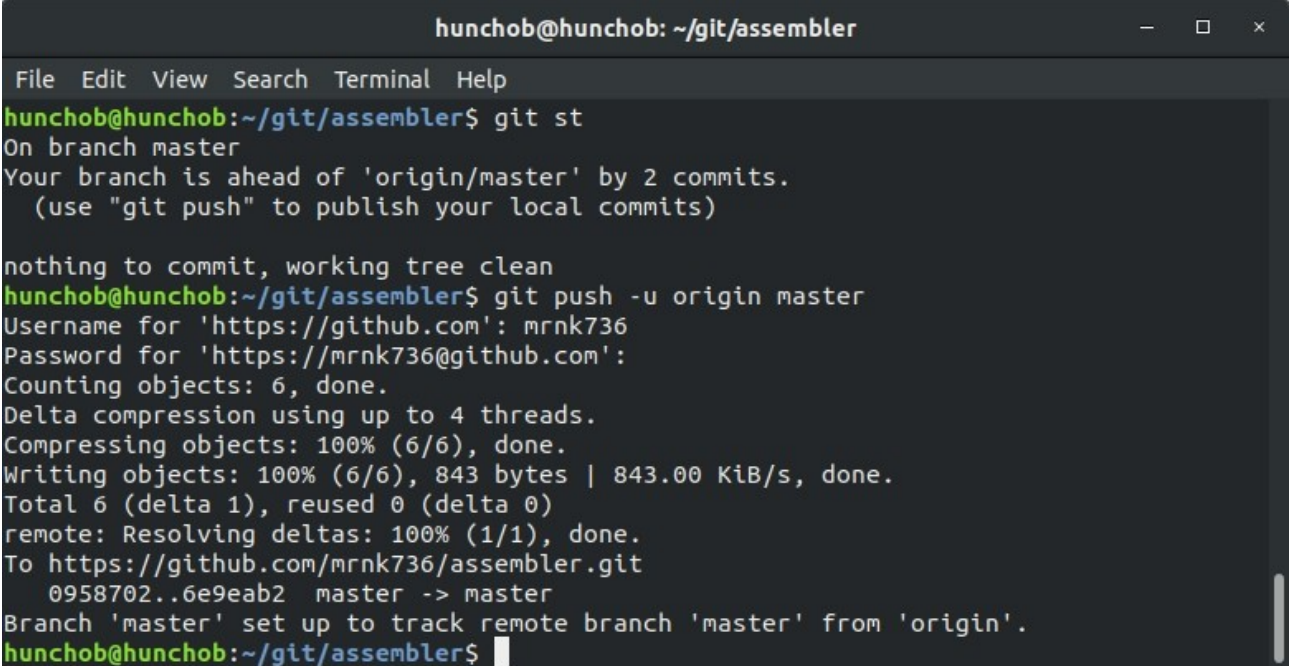
```
hunchob@hunchob: ~/git/assembler
File Edit View Search Terminal Help
hunchob@hunchob:~$ cd git/assembler
hunchob@hunchob:~/git/assembler$ ls
Dockerfile kate.s README.md
hunchob@hunchob:~/git/assembler$ gcc -m32 kate.s -o kate
hunchob@hunchob:~/git/assembler$ ls
Dockerfile kate kate.s README.md
hunchob@hunchob:~/git/assembler$ ./kate
207
hunchob@hunchob:~/git/assembler$
```

Рисунок 2.2 – Работа программы

Далее все исходные коды были помещены в репозиторий на GitHub (рис. 2.3). Ссылка на данный репозиторий представлена ниже:

<https://github.com/mrnk736>



A screenshot of a terminal window titled "hunchob@hunchob: ~/git/assembler". The terminal shows the following sequence of commands and output:

```
File Edit View Search Terminal Help
hunchob@hunchob:~/git/assembler$ git st
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
hunchob@hunchob:~/git/assembler$ git push -u origin master
Username for 'https://github.com': mrnk736
Password for 'https://mrnk736@github.com':
Counting objects: 6, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 843 bytes | 843.00 KiB/s, done.
Total 6 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/mrnk736/assembler.git
   0958702..6e9eab2  master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
hunchob@hunchob:~/git/assembler$
```

Рисунок 2.3 – Работа с системой управления контроля версиями

### 3 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 GNU Assembler.

### Листинг А.1 – Код основной программы (начало)

adding:

inc %ecx

end\_of\_array:

cmpl \$elem\_in\_array, %ecx

jl calculating

pushl %ebx

pushl \$str\_d

call printf

addl \$8, %esp

ret

#### Код Dockerfile

FROM ubuntu:18.04

RUN apt-get install build essential gdb gcc-multilib

WORKDIR /home/lab1

COPY ./kate.s /home/lab1

Листинг А.1 – Код основной программы (окончание)