

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

“ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ” (ТУСУР)

Кафедра комплексной информационной безопасности электронно-
вычислительных систем (КИБЭВС)

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ
АССЕМБЛЕР

Отчёт по лабораторной работе №2
по дисциплине «Системное программирование»

Студент гр.718

_____Р.Д. Сахарбеков

«__»_____2022

Принял

М.н.с ИСИБ

_____Е.О. Калинин

«__»_____2022

Томск 2022

Введение

Целью работы является познакомиться со структурой программы на языке Ассемблер, разновидностями и назначением сегментов, способами организации простых и сложных типов данных.

Задание:

- подготовить образ операционной системы Linux для Docker. Установить компилятор GCC и другие необходимые пакеты. Скомпилировать простейшую программу на Ассемблере с помощью компилятора GCC;
- воспользоваться отладчиком GDB и научиться пользоваться представляемой им информацией;
- получить индивидуальное задание у преподавателя и реализовать соответствующую программу на Ассемблере и на языке высокого уровня. Дизассемблировать обе программы, провести сравнительный анализ скорости работы программ, объема полученного дизассемблированного кода, попробовать оптимизировать программы. Сделать выводы.

Вариант 21.

Задача: дан массив из 10 слов. Определить минимальный и максимальный элементы массива.

2 ХОД РАБОТЫ

Для начала командой «*sudo apt install gcc-multilib*» был установлен основной пакет компилятора GCC для работы с Ассемблером (рисунок 2.1).

```
ruslan718@ruslan718-VirtualBox:~$ sudo apt install gcc-multilib
[sudo] пароль для ruslan718:
Чтение списков пакетов... Готово
Построение дерева зависимостей
Чтение информации о состоянии... Готово
Будут установлены следующие дополнительные пакеты:
gcc-9-multilib lib32asan5 lib32atomic1 lib32gcc-9-dev
lib32gcc-s1 lib32gomp1 lib32itm1 lib32quadmath0 lib32stdc++6
lib32ubsan1 libc6-dev-i386 libc6-dev-x32 libc6-i386 libc6-x32
```

Рисунок 2.1 – Установка пакетов компилятора

Была создана и написана тестовая программа на языке Ассемблер (рисунок 2.2 – 2.3).

```
ruslan718@ruslan718-VirtualBox:~$ nano test_proga.s
```

Рисунок 2.2 – Создание программы

```
GNU nano 4.8 test_proga.s
.data
    hello_str:
        .string "Привет мир\n"
        .set hl, .-hello_str - 1
.text
.globl main
.type main, @function
main:
    movl $4, %eax
    movl $1, %ebx
    movl $hello_str, %ecx
    movl $hl, %edx

    int $0x80
    movl $1, %eax
    movl $0, %ebx
    int $0x80
.size main, . - main
```

Рисунок 2.3 – Тестовая программа

Запустим программу на отладку и скомпилируем. (рисунок 2.4).

```
ruslan718@ruslan718-VirtualBox:~$ gcc -m32 -fno-pie -no-pie test_proga.s -o test_proga -g
ruslan718@ruslan718-VirtualBox:~$ sudo ./test_proga
Привет мир
```

Рисунок 2.4 – Работа программы

Удалена отладочная информация командой `strip prog1` (рисунок 2.5).

```
ruslan718@ruslan718-VirtualBox:~$ strip test_prog1
```

Рисунок 2.5 – Удаление отладочной информации

Был получен физический адрес точки входа командой `objdump`, запущенной с ключом `-f` «***objdump -f name***» (рисунок 2.6).

```
ruslan718@ruslan718-VirtualBox:~$ objdump -f test_prog1

test_prog1:      формат файла elf32-i386
архитектура: i386, флаги 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
начальный адрес 0x08049050
```

Рисунок 2.6 – Адрес

Запустим программу на отладку, указав в качестве параметра её имя (`gdb prog1`). Также была установлена точка наблюдения на данный адрес с помощью команды «***break address***», Далее программы была запущена и завершена успешно «***run***» (рисунок 2.7).

```
ruslan718@ruslan718-VirtualBox:~$ gdb test_prog1
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from test_prog1...
(No debugging symbols found in test_prog1)
(gdb) break 0x08049050
Function "0x08049050" not defined.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 1 (0x08049050) pending.
(gdb) run
Starting program: /home/ruslan718/test_prog1
Привет мир
[Inferior 1 (process 8336) exited normally]
(gdb) q
ruslan718@ruslan718-VirtualBox:~$
```

Рисунок 2.7 – Запуск отладчика

Создадим файл с помощью команды «*nano lab2.c*» где расширение «с» для языка программирования С (Си) и пропишем команду «*gcc lab.c -o lab2C*», чтобы скомпилировать файл на С (рисунок 2.8). Смотреть Приложение А.

```
ruslan718@ruslan718-VirtualBox:~$ nano lab2.c
GNU nano 4.8 lab2.c
#include <stdio.h>

int main ()
{
    printf("Hello, World!");
    return 0;
}

ruslan718@ruslan718-VirtualBox:~$ gcc lab2.c -o lab2C
ruslan718@ruslan718-VirtualBox:~$ ./lab2C
Hello, World!
ruslan718@ruslan718-VirtualBox:~$
```

Рисунок 2.8 – Проверка работоспособности «Hello World» для С

Теперь создадим файл «*nano lab2.s*», где расширение «s» для Ассемблера и пропишем команду «*gcc -m32 -fno-pie lab2.s -o lab2S -g*» для отладки и компиляции файла на Ассемблере (рисунок 2.9). Смотреть Приложение А.

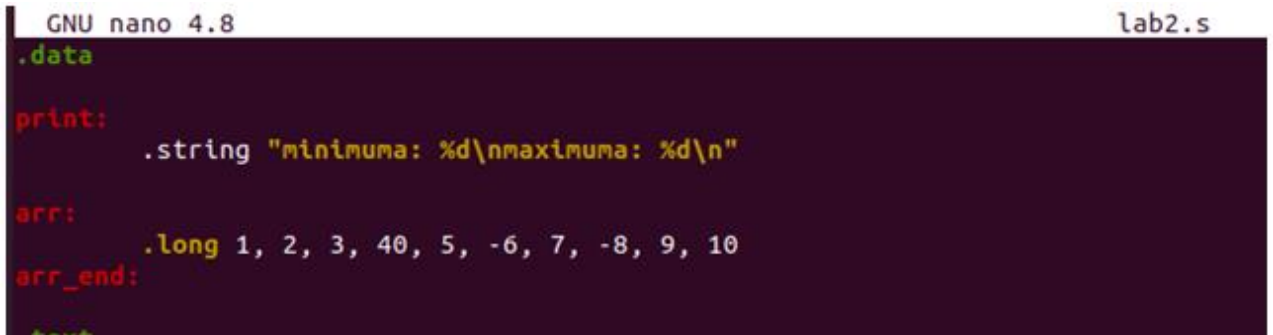
```
ruslan718@ruslan718-VirtualBox:~$ nano lab2.s
GNU nano 4.8 lab2.s
.data
    hello_str:
        .string "Привет мир\n"
        .set hl, .-hello_str - 1
.text
.globl main
.type main, @function
main:
    movl $4, %eax
    movl $1, %ebx
    movl $hello_str, %ecx
    movl $hl, %edx

    int $0x80
    movl $1, %eax
    movl $0, %ebx
    int $0x80
.size main, . - main

ruslan718@ruslan718-VirtualBox:~$ gcc -m32 -fno-pie lab2.s -o lab2S -g
ruslan718@ruslan718-VirtualBox:~$ ./lab2S
Привет мир
ruslan718@ruslan718-VirtualBox:~$
```

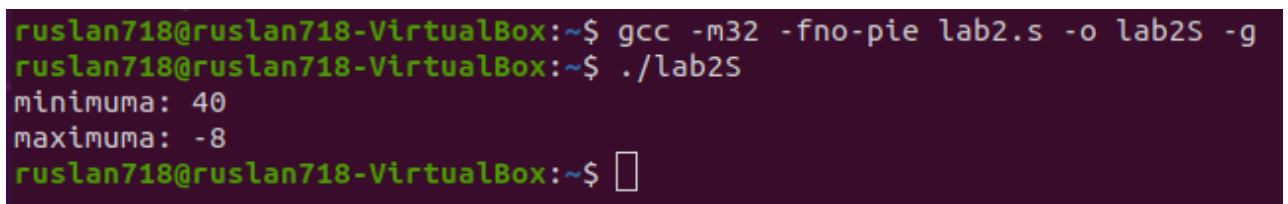
Рисунок 2.9 – Проверка работоспособности «Hello World» для Ассемблера

Далее напишем программу для Ассемблера, которая задана по варианту №21, и проверим ее работоспособность (рисунок 2.10 – 2.11). Листинг программы представлен в приложении Б.



```
GNU nano 4.8 lab2.s
.data
print:
    .string "minimuma: %d\nmaximuma: %d\n"
arr:
    .long 1, 2, 3, 40, 5, -6, 7, -8, 9, 10
arr_end:
.text
```

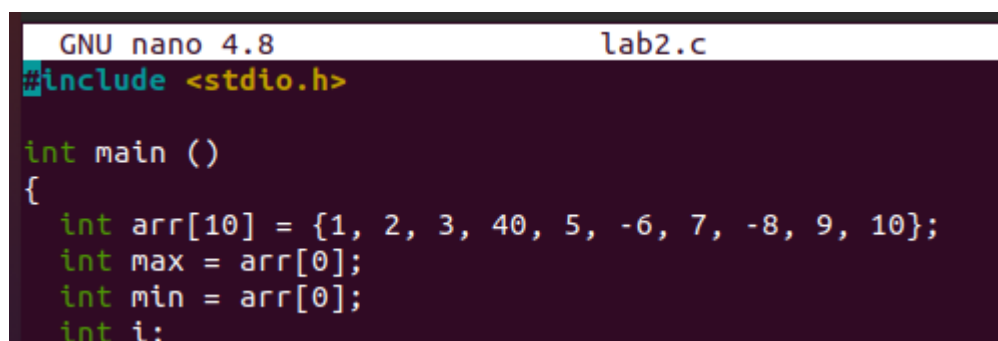
Рисунок 2.10 – Листинг программы на Ассемблере



```
ruslan718@ruslan718-VirtualBox:~$ gcc -m32 -fno-pie lab2.s -o lab2S -g
ruslan718@ruslan718-VirtualBox:~$ ./lab2S
minimuma: 40
maximuma: -8
ruslan718@ruslan718-VirtualBox:~$
```

Рисунок 2.11 – Результат программы на Ассемблере

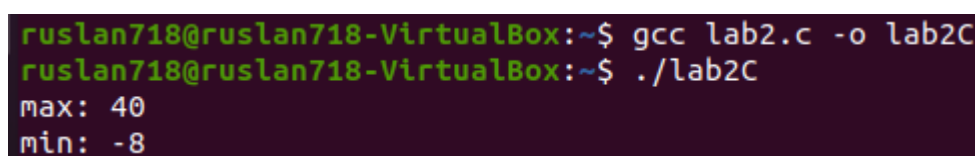
Напишем программу по заданному варианту на языке С (рисунок 2.12 – 2.13). Листинг программы представлен в приложении Б.



```
GNU nano 4.8 lab2.c
#include <stdio.h>

int main ()
{
    int arr[10] = {1, 2, 3, 40, 5, -6, 7, -8, 9, 10};
    int max = arr[0];
    int min = arr[0];
    int i;
```

Рисунок 2.12 – Листинг программы на С



```
ruslan718@ruslan718-VirtualBox:~$ gcc lab2.c -o lab2C
ruslan718@ruslan718-VirtualBox:~$ ./lab2C
max: 40
min: -8
```

Рисунок 2.13 – Результат программы на С

Так же по заданию нужно было провести анализ двух написанных программ. Пропишем команду «*du --bytes name*», чтобы узнать их дисковое пространство (рисунок 2.14) и можно увидеть, что код на Ассемблере в 3 раза больше, чем код на С.

```
ruslan718@ruslan718-VirtualBox:~$ du --bytes lab2.c
368      lab2.c
ruslan718@ruslan718-VirtualBox:~$ du --bytes lab2.s
1003     lab2.s
```

Рисунок 2.14 – Вес программного кода

Напишем команду «*time ./name*», чтобы узнать скорость выполнения программ. Можем заметить, что по скорости выполнения кода на С выполняется намного быстрее, чем на Ассемблере (рисунок 2.15).

```
ruslan718@ruslan718-VirtualBox:~$ time ./lab2C
max: 40
min: -8

real    0m0,016s
user    0m0,003s
sys     0m0,001s
ruslan718@ruslan718-VirtualBox:~$ time ./lab2S
minimum: 40
maximum: -8

real    0m0,003s
user    0m0,002s
sys     0m0,001s
ruslan718@ruslan718-VirtualBox:~$
```

Рисунок 2.15 – Скорость выполнения программ

Далее проведем дизассемблирование программ с помощью команды «*objdump -D name*» (рисунок 2.16 – 2.17).


```

ruslan718@ruslan718-VirtualBox:~$ objdump -D lab2S

lab2S:      формат файла elf32-i386

Дизассемблирование раздела .interp:

00000194 <.interp>:
194:    2f                      das
195:    6c                      insb    (%dx),%es:(%edi)
196:    69 62 2f 6c 64 2d 6c    imul    $0x6c2d646c,0x2f(%edx),%esp
19d:    69 6e 75 78 2e 73 6f    imul    $0x6f732e78,0x75(%esi),%ebp
1a4:    2e 32 00                xor     %cs:(%eax),%al

```

Рисунок 2.16 – Дизассемблирование программы на Ассемблере

```

ruslan718@ruslan718-VirtualBox:~$ objdump -D lab2C

lab2C:      формат файла elf64-x86-64

Дизассемблирование раздела .interp:

00000000000000318 <.interp>:
318:    2f                      (bad)
319:    6c                      insb    (%dx),%es:(%rdi)
31a:    69 62 36 34 2f 6c 64    imul    $0x646c2f34,0x36(%rdx),%esp
31b:    2d 6e 60 60 75         sub     $0x75606060,%eax

```

Рисунок 2.17 – Дизассемблирование программы на С

Сравним вес полученных файлов (рисунок 2.18) и можно заметить, что после дизассемблирования исполняемый файл на языке С стал больше весить.

```

ruslan718@ruslan718-VirtualBox:~$ objdump -D lab2S > lab2S_dump
ruslan718@ruslan718-VirtualBox:~$ objdump -D lab2C > lab2C_dump
ruslan718@ruslan718-VirtualBox:~$ du --bytes lab2S_dump
39547    lab2S_dump
ruslan718@ruslan718-VirtualBox:~$ du --bytes lab2C_dump
45172    lab2C_dump
ruslan718@ruslan718-VirtualBox:~$

```

Рисунок 2.18 – Сравнение программного кода после дизассемблирования

Был собран и запущен образ на основе файла «Dockerfile» (рисунок 2.19 – 2.21). Были использованы команды: «*docker build -t test .*» и «*docker run -it test*».

GNU nano 4.8	Dockerfile
FROM ubuntu	
RUN apt-get update	
RUN apt install gcc gdb gcc-multilib nano -y	
COPY lab2.s .	
RUN gcc -m32 lab2.s -o lab2S	
CMD ./lab2S	

Рисунок 2.19 – «Dockerfile»

```

ruslan718@ruslan718-VirtualBox:~$ docker build -t test .
Sending build context to Docker daemon 37.18MB
Step 1/6 : FROM ubuntu
--> 54c9d81cbb44
Step 2/6 : RUN apt-get update
--> Using cache
--> 0b896e6da4f5
Step 3/6 : RUN apt install gcc gdb gcc-multilib nano -y
--> Using cache
--> 6d39f2f62c2f
Step 4/6 : COPY lab2.s .
--> a09d3285dfb7
Step 5/6 : RUN gcc -m32 lab2.s -o lab2S
--> Running in ade142b73d4f
Removing intermediate container ade142b73d4f
--> bdbfa907d1ad
Step 6/6 : CMD ./lab2S
--> Running in 7d1165ef696c
Removing intermediate container 7d1165ef696c
--> c198ccada165
Successfully built c198ccada165
Successfully tagged test:latest

```

Рисунок 2.20 – Сборка образа lab2_test на основе образа Ubuntu и Dockerfile

```

ruslan718@ruslan718-VirtualBox:~$ docker run -it test
minimума: 40
maximума: -8
ruslan718@ruslan718-VirtualBox:~$ █

```

Рисунок 2.21 – Создание и запуск контейнера на основе собранного образа lab2_test

Заключение

В результате выполнения данной лабораторной работы было произведено ознакомление со структурой программы на языке Ассемблер, разновидностями и назначением сегментов, способами организации простых и сложных типов данных, изучение форматов и правил работы с синтаксисами AT&T, ознакомление с возможностями GCC для работы с Ассемблером и средствами создания программ на Ассемблере для ОС Linux.

Приложение А
(обязательное)

<https://github.com/ruslanushka/sp/tree/master/2>