

Отчёт по лабораторной работе №10

Компьютерные науки и технология программирования

Сячинова Ксения Ивановна

Содержание

1	Цель работы	6
2	Выполнение лабораторной работы	7

Список иллюстраций

2.1	Создание необходимых файлов	7
2.2	Текст программы	8
2.3	Результат программы	8
2.4	Изменение программы	9
2.5	Изменение программы	9
2.6	Результат	9
2.7	Создание файла	10
2.8	Текст программы	10
2.9	Компиляция программы	10
2.10	Проверка работы программы	10
2.11	Установка брейкпоинта	11
2.12	Дисассимблированный код программы	11
2.13	Переключение на другой синтаксис	12
2.14	Режим псевдографики	12
2.15	Проверка точек	13
2.16	Новая точка останова	13
2.17	Точки останова	13
2.18	Инструкция stepi	14
2.19	Просмотр регистров	14
2.20	Содержимое памяти	14
2.21	Изменение значения регистров	15
2.22	Изменение значения регистров	15
2.23	Просмотр значений	15
2.24	Изменение регистра	15
2.25	Завершение программы	15
2.26	Копирование файла	16
2.27	Создание исполняемого файла	16
2.28	Загрузка в GDB	16
2.29	Точка останова	16
2.30	Адрес вершины стека	17
2.31	Позиции стека	17
2.32	Копирование файла	17
2.33	Текст программы	18
2.34	Текст прогарммы	19
2.35	Проверка файла	19
2.36	Текст программы	20
2.37	Результат программы	20

2.38 GDB	20
2.39 Точка останова	21
2.40 Запуск программы	21
2.41 Проверка программы	21
2.42 Исправление ошибки	22
2.43 Проверка	22

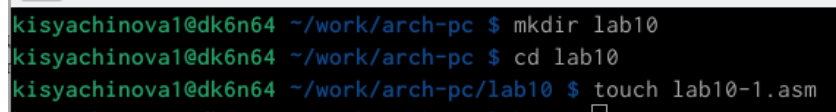
Список таблиц

1 Цель работы

Приобрести навыки написания программ с использованием подпрограмм. познакомиться с методами отладки при помощи GDB и его основными возможностями.

2 Выполнение лабораторной работы

1. Создаём каталог для выполнения лабораторной работы. Переходим в него и создаём файл для первого листинга.(рис. 2.1)



```
kisyachinova1@dk6n64 ~/work/arch-pc $ mkdir lab10
kisyachinova1@dk6n64 ~/work/arch-pc $ cd lab10
kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $ touch lab10-1.asm
```

Рис. 2.1: Создание необходимых файлов

2. Рассмотрим программу вычисления арифметических выражения с помощью подпрограммы '_calcul'. (рис. 2.2), (рис. 2.3)

```

lab10-1.asm      [----]  3 L:1
%include 'in_out.asm'

SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7= ',0

SECTION .bss
x:<---->RESB 80
rez:<-->RESB 80

SECTION .text
GLOBAL _start
_start:

mov eax, msg
call sprint

mov ecx, x
mov edx, 80
call sread

mov eax, x
call atoi

call _calcul

mov eax, result
call sprint
mov eax, [rez]
call iprintLF

call quit

_calcul:
mov ebx, 2
mul ebx
add eax, 7
mov [rez], eax

ret

```

Рис. 2.2: Текст программы

```

kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $ nasm -f elf lab10-1.asm
kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o lab10-1 lab10-1.o
kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $ ./lab10-1
Введите x: 2
2x+7= 11

```

Рис. 2.3: Результат программы

3. Изменим программу, при этом добавим подпрограмму $g(x)=3x-1$. (рис. 2.4), (рис. 2.5), (рис. 2.6)


```

call atoi

call _subcalcul
call _calcul

mov eax,result
call sprint
mov eax,[rez]
call printf

```

Рис. 2.4: Изменение программы

```

_subcalcul:
mov ebx,3
mul ebx
sub eax, 1
mov [rez],eax

ret

```

Рис. 2.5: Изменение программы

```

kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $ nasm -f elf lab10-1.asm
kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o lab10-1 lab10-1.o
kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $ ./lab10-1
Введите x: 1
2x+7= 11
kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $ nasm -f elf lab10-1.asm
kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $ ./lab10-1
Введите x: 2
2x+7= 17

```

Рис. 2.6: Результат

Программа работает корректно. При $x=1$ имеем, что $g(x)=2$, а $f(x)=11$.

4. Создаём файл для второго листинга. Напишем программу печати сообщения Hello world!. (рис. 2.7), (рис. 2.8)

```
kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $ touch lab10-2.asm
```

Рис. 2.7: Создание файла

```
lab10-2.asm [----] 0 L: 1+
SECTION .data
msg1:<----->db "Hello, ",0x0
msg1Len:<-->equ $ - msg1
...
msg2:<----->db "world!",0xa
msg2Len:<-->equ $ - msg2
...
SECTION .text
global _start

_start:
mov eax, 4
mov ebx, 1
mov ecx, msg1
mov edx, msg1Len
int 0x80

mov eax, 4
mov ebx, 1
mov ecx, msg2
mov edx, msg2Len
int 0x80

mov eax, 1
mov ebx, 0
int 0x80
```

Рис. 2.8: Текст программы

Затем получаем исполняемый файл. Для работы с GDB нужно добавить отладочную информацию, для этого трансляцию программы необходимо проводить с ключом “-g”. Загружаем исполняемый файл в отладчик. (рис. 2.9)

```
kisyachinova1@dkin22 ~/work/arch-pc/lab10 $ nasm -f elf -g -l lab10-2.lst lab10-2.asm
kisyachinova1@dkin22 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o lab10-2 lab10-2.o
kisyachinova1@dkin22 ~/work/arch-pc/lab10 $ gdb lab10-2
```

Рис. 2.9: Компиляция программы

Проверим работу программы, запускаем её в оболочке GDB с помощью команды ‘run’. (рис. 2.10)

```
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/k/i/kisyachinova1/work/
arch-pc/lab10/lab10-2
Hello, world!
[Inferior 1 (process 3285) exited normally]
```

Рис. 2.10: Проверка работы программы

Затем установим брейкпоинт на метку `_start`, с которой начинается выполнение программы и запустим её. (рис. 2.11)

```
(gdb) b _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 12.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/k/i/kisyachinova1/work/arc
b10/lab10-2
Breakpoint 1, _start () at lab10-2.asm:12
12      mov eax, 4
```

Рис. 2.11: Установка брейкпоинта

Далее посмотрим дисассимблированный код программы с помощью команды `disassemble` начиная с метки `_start`. (рис. 2.12)

```
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:      mov     $0x4,%eax
      0x08049005 <+5>:      mov     $0x1,%ebx
      0x0804900a <+10>:     mov     $0x804a000,%ecx
      0x0804900f <+15>:     mov     $0x8,%edx
      0x08049014 <+20>:     int     $0x80
      0x08049016 <+22>:     mov     $0x4,%eax
      0x0804901b <+27>:     mov     $0x1,%ebx
      0x08049020 <+32>:     mov     $0x804a008,%ecx
      0x08049025 <+37>:     mov     $0x7,%edx
      0x0804902a <+42>:     int     $0x80
      0x0804902c <+44>:     mov     $0x1,%eax
      0x08049031 <+49>:     mov     $0x0,%ebx
      0x08049036 <+54>:     int     $0x80
End of assembler dump.
```

Рис. 2.12: Дисассимблированный код программы

Переключимся на отображение команд с Intel'овским синтаксисом с помощью команды `set disassembly-flavor intel`. (рис. 2.13)

```

(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:      mov     eax,0x4
    0x08049005 <+5>:      mov     ebx,0x1
    0x0804900a <+10>:     mov     ecx,0x804a000
    0x0804900f <+15>:     mov     edx,0x8
    0x08049014 <+20>:     int     0x80
    0x08049016 <+22>:     mov     eax,0x4
    0x0804901b <+27>:     mov     ebx,0x1
    0x08049020 <+32>:     mov     ecx,0x804a008
    0x08049025 <+37>:     mov     edx,0x7
    0x0804902a <+42>:     int     0x80
    0x0804902c <+44>:     mov     eax,0x1
    0x08049031 <+49>:     mov     ebx,0x0
    0x08049036 <+54>:     int     0x80
End of assembler dump.

```

Рис. 2.13: Переключение на другой синтаксис

Для более удобного анализа программы включаем режим псевдографики.(рис. 2.14)

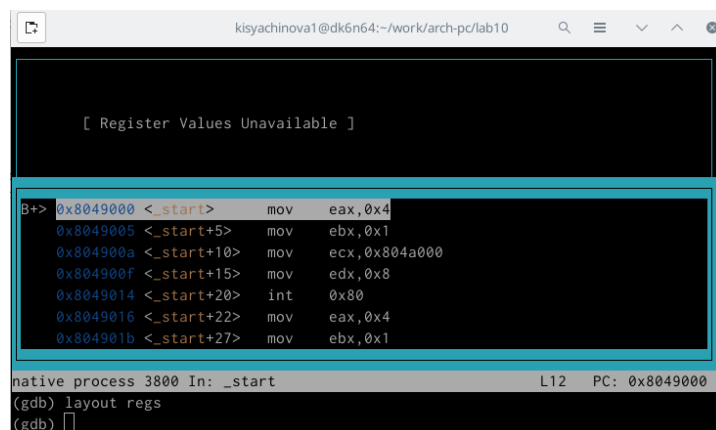


Рис. 2.14: Режим псевдографики

В этом режиме имеем три окна: - В верхней части названия регистров и их текущее значения - В средней части виден результат дисассимилирования программы - Нижняя часть доступная для ввода команд

Основные различия синтаксиса машинных команд в режимах АТТ и Intel заключаются в том, что в синтаксисе Intel не используются символы \$ и %, а так же

операнды меняются местами.

5. Для проверки точек останова используем команду ‘info breakpoints’. (рис. 2.15)

```
native process 3800 In: _start L12 PC: 0x0049000
(gdb) layout regs
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x0049000 lab10-2.asm:12
breakpoint already hit 1 time
```

Рис. 2.15: Проверка точек

Установим ещё одну точку останова по адресу инструкции (ebx, 0x0). (рис. 2.16)

```
(gdb) b *0x0049031
Breakpoint 2 at 0x0049031: file lab10-2.asm, line 25.
```

Рис. 2.16: Новая точка останова

```
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x0049000 lab10-2.asm:12
breakpoint already hit 1 time
2        breakpoint keep y  0x0049031 lab10-2.asm:25
```

Рис. 2.17: Точки останова

6. Выполним 5 инструкций с помощью команды “stepi” и проследим значение регистров.(рис. 2.18). Как мы можем заметить, изменяются значения регистров eax, ebx, ecx, edx.

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc5a0 0xffffc5a0

0x8049005 <_start+5> mov ebx,0x1
0x804900a <_start+10> mov ecx,0x804a000
0x804900f <_start+15> mov edx,0x8
0x8049014 <_start+20> int 0x80
> 0x8049016 <_start+22> mov eax,0x4
0x804901b <_start+27> mov ebx,0x1
0x8049020 <_start+32> mov ecx,0x804a008

native process 3800 In: _start L18 PC: 0x8049016
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stwep
Undefined command: "stwep". Try "help".
(gdb) stepi
(gdb) stepi
(gdb)

```

Рис. 2.18: Инstrukция stepi

Команда 'info registers' (i r) позволяет посмотреть содержимое регистров. (рис. 2.19)

```

eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc5a0 0xffffc5a0
ebp      0x0      0
esi      0x0      0
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 2.19: Просмотр регистров

Команду 'x' можно использовать для отображения памяти. С помощью 'x &' так же можно это сделать.(рис. 2.20)

```

(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x /1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb)

```

Рис. 2.20: Содержимое памяти

Далее воспользуемся командой 's' Приобрести навыки написания программ с использованием подпрограмм. познакомиться с методами отладки при помощи GDB и его основными возможностями.et', которая помогает изменить значение для регистра и ячейки. (рис. 2.21), (рис. 2.22)

```
(gdb) set {char}0x804a000='h'
(gdb) x /1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) set {char}0x804a001='A'
(gdb) x /1sb &msg1
0x804a000 <msg1>:      "hAllo, "
(gdb) □
```

Рис. 2.21: Изменение значения регистров

```
(gdb) set {char}0x804a008='K'
(gdb) x /1sb &msg2
0x804a008 <msg2>:      "Kor1d!\n\034"
(gdb) □
```

Рис. 2.22: Изменение значения регистров

Для просмотра значений используем команду ‘print /F’. (рис. 2.23)

```
(gdb) p/x $edx
$1 = 0x8
(gdb) p/s $edx
$2 = 8
(gdb) p/t $edx
$3 = 1000
(gdb) p/s $eax
$4 = 8
(gdb) □
```

Рис. 2.23: Просмотр значений

С помощью команды ‘set’ можем изменить значение регистра. Изменим значения регистра ‘ebx’. (рис. 2.24)

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$5 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$6 = 2
(gdb) □
```

Рис. 2.24: Изменение регистра

Основное различие в том, что код символа ‘2’ - 110010, а это есть число 50.

Завершим выполнение программы с помощью команды ‘continue’ и выйдем из отладчика с помощью команды ‘q’. (рис. 2.25)

```
Continuing.
[Inferior 1 (process 5038) exited normally]
(gdb) q
kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $ □
```

Рис. 2.25: Завершение программы

7. Копируем файл из лабораторной работы №9 с новым именем.(рис. 2.26)

```
kisyachinova1@dk6n64 ~ $ cp ~/work/arch-pc/lab09/lab9-2.asm ~/work/arch-pc/lab10/lab10-3.asm
kisyachinova1@dk6n64 ~ $ cd work
kisyachinova1@dk6n64 ~/work $ cd arch-pc
kisyachinova1@dk6n64 ~/work/arch-pc $ cd lab10
kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $ ls
in_out.asm  lab10-1.1  lab10-1.asm  lab10-2      lab10-2.1st  lab10-3.asm
lab10-1     lab10-1.1.o  lab10-1.o   lab10-2.asm  lab10-2.o
kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $
```

Рис. 2.26: Копирование файла

Далее создаём исполняемый файл. (рис. 2.27)

```
kisyachinova1@dk6n64 ~ $ cp ~/work/arch-pc/lab09/lab9-2.asm ~/work/arch-pc/lab10/lab10-3.asm
kisyachinova1@dk6n64 ~ $ cd work
kisyachinova1@dk6n64 ~/work $ cd arch-pc
kisyachinova1@dk6n64 ~/work/arch-pc $ cd lab10
kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $ ls
in_out.asm  lab10-1.1  lab10-1.asm  lab10-2      lab10-2.1st  lab10-3.asm
lab10-1     lab10-1.1.o  lab10-1.o   lab10-2.asm  lab10-2.o
kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $
```

Рис. 2.27: Создание исполняемого файла

Для загрузки в GDB программы с аргументами необходимо использовать ключ `-args`. (рис. 2.28)

```
kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $ gdb --args lab10-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Gentoo 11.2 vanilla) 11.2
Copyright (C) 2022 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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
```

Рис. 2.28: Загрузка в GDB

Затем исследуем расположение аргументов командной строки. Для начала установим точку останова и запустим команду. (рис. 2.29)

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab10-3.asm, line 7.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/k/i/kisyachinova1/work/arch-pc/lab10/
lab10-3 аргумент1 аргумент 2 аргумент\ 3

Breakpoint 1, _start () at lab10-3.asm:7
7      pop ecx
```

Рис. 2.29: Точка останова

Адрес вершины стека храниться в регистре `esp` и по этому адресу располагается число равное количеству аргументов командной строки. (рис. 2.30)


```
(gdb) x/x $esp
0xffffc430: 0x00000005
```

Рис. 2.30: Адрес вершины стека

Посмотрим остальные позиции стека – по адресу [esp+4]. Адрес располагается в памяти где находится имя программы, по адресу [esp+8] храниться адрес первого аргумента, по адресу [esp+12] – второго и т.д. (рис. 2.31)

```
(gdb) x/s *(void**)(esp + 4)
0xffffc6c5: "/afs/.dk.sci.pfu.edu.ru/home/k/i/kisyachinova1/work/arch-pc/lab10/lab10-3"
(gdb) x/s *(void**)(esp + 8)
0xffffc70f: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffc721: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffc732: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffc734: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
```

Рис. 2.31: Позиции стека

В данном случае шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12] и т.д.) т.к. в теле цикла next 4 строки ввода.

#Задания для самостоятельной работы.

1. Копируем файл из 9 лабораторной работы. (рис. 2.32)

```
kisyachinova1@dk6n64 ~ $ cp ~/work/arch-pc/lab09/lab9-1.1.asm ~/work/arch-pc/lab10/lab10-1.1.asm
kisyachinova1@dk6n64 ~ $ cd work
kisyachinova1@dk6n64 ~/work $ cd arch-pc
kisyachinova1@dk6n64 ~/work/arch-pc $ cd lab10
kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $ ls
in_out.asm lab10-1.1.asm lab10-1.o lab10-2.lst lab10-3.asm
lab10-1 lab10-1.1.o lab10-2 lab10-2.o lab10-3.lst
lab10-1.1 lab10-1.asm lab10-2.asm lab10-3 lab10-3.o
kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $
```

Рис. 2.32: Копирование файла

Изменим программу из 9 лабораторной с использованием подпрограмм и запустим её. (рис. 2.33)

```

lab10-1.1.asm [----]
%include 'in_out.asm'

SECTION .data
msg db "Результат: ",0

SECTION .text
GLOBAL _start

_start:
    pop ecx
    ....
    pop edx
    ....
    sub ecx,1
    ....
    mov esi, 0
    ....
next:
    cmp ecx,0h
    jz _end
    ....
    pop eax
    call atoi
    call _calcul
    ....
    ....
    loop next

```

Рис. 2.33: Текст программы

```

_end:
    mov eax,msg
    call sprint
    mov eax, esi
    call iprintLF
    call quit
    ....
_calcul:
    mov eax, eax
    mov ebx, 15
    mul ebx
    add eax, 2
    add esi,eax
    ....
    ret

```

Рис. 2.34: Текст прогарммы

```

kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $ nasm -f elf lab10-1.1.asm
kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o lab10-1.1 lab10-1.1.o
kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $ ./lab10-1.1
Результат: 0
kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $ ./lab10-1.1 1 2 3
Результат: 96
kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $ ./lab10-1.1 1 2
Результат: 49

```

Рис. 2.35: Проверка файла

2. Напишем программу из листинга лабораторной работы. Проверяем работу с помощью отладчика.(рис. 2.36), (рис. 2.37)

```
lab10-1.2.asm [----]
#include 'in_out.asm'

SECTION .data
div: DB 'Результат: ',0

SECTION .text
GLOBAL _start
_start:

mov ebx,3
mov eax,2
add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx

mov eax,div
call sprint

mov eax,edi
call iprintLF

call quit
```

Рис. 2.36: Текст программы

```
kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $ nasm -f elf lab10-1.2.asm
kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o lab10-1.2 lab10-1.2.o
kisyachinova1@dk6n64 ~/work/arch-pc/lab10 $ ./lab10-1.2
Результат: 10
```

Рис. 2.37: Результат программы

Видим, что результат программы неверный, т.к $(3+2)*4+5=25$, а не 10. Для устранения ошибки запускаем отладчик.(2.38)

```
kisyachinova1@dk6n65 ~/work/arch-pc/lab10 $ gdb lab10-1.2
GNU gdb (Gentoo 11.2 vanilla) 11.2
Copyright (C) 2022 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.
```

Рис. 2.38: GDB

Установим точку останова. (рис. 2.39)

```
(gdb) b _start
Breakpoint 1 at 0x80490e8
(gdb) i b
Num      Type          Disp Enb Address      What
1        breakpoint    keep y   0x080490e8  <_start>
```

Рис. 2.39: Точка останова

Запускаем код программы.(рис. 2.40)

```
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/k/i/kisyachinova1/work/arch-pc/lab10/lab10-1.2
Breakpoint 1, 0x080490e8 in _start ()
```

Рис. 2.40: Запуск программы

Включаем режим псевдографики и пошагово проходим все строчки.(рис. 2.41)

```
Register group: general
eax      0x804a000      134520832
ecx      0x4           4
edx      0x0           0
ebx      0x804a000      134520832
esp      0xffffc584     0xffffc584

0x8049000 <slen>      push    %ebx
0x8049001 <slen+1>    mov     %eax,%ebx
0x8049003 <nextchar>  cmpb    $0x0,(%eax)
0x8049006 <nextchar+3> je      0x804900b <finished>
> 0x8049008 <nextchar+5> inc     %eax
0x8049009 <nextchar+6> jmp     0x8049003 <nextchar>
0x804900b <finished> sub     %ebx,%eax

native process 7572 In: nextchar      L??  PC: 0x8049008
0x08049001 in slen ()
(gdb) si
0x08049003 in nextchar ()
(gdb) si
0x08049006 in nextchar ()
(gdb) si
0x08049008 in nextchar ()
(gdb) si
```

Рис. 2.41: Проверка программы

Как мы можем заметить, регистр `eax` должен умножаться на 4, но у нас умножался регистр `ebx`. Также, число 5 прибавлялось не к произведению, а только к `ebx`. Исправим ошибки. рис. 2.42)

```

mov ebx,3
mov eax,2
add ebx,eax
mov eax,ebx
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx

```

Рис. 2.42: Исправление ошибки

```

kisyachinova1@dk6n65 ~/work/arch-pc/lab10 $ nasm -f elf lab10-1.2.asm
kisyachinova1@dk6n65 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o lab10-1.2 lab10-1.2.o
kisyachinova1@dk6n65 ~/work/arch-pc/lab10 $ ./lab10-1.2
Результат: 25

```

Рис. 2.43: Проверка

Программа работает корректно.

#Вывод

В ходе выполнения данной лабораторной работы я преобрела навыки написания программ с использованием подпрограмм, а так же познакомилась с методами отладки при помощи GDB и его основными возможностями.