

Отчёт по лабораторной работе №9

Дисциплина: Архитектура компьютера

Кузьмина Мария Константиновна

Содержание

1	Цель работы	4
2	Задание	5
3	Выполнение лабораторной работы	6
3.1	Реализация подпрограмм в NASM	6
3.2	Отладка программ с помощью GDB	9
3.3	Выполнение заданий для самостоятельной работы	15
4	Выводы	19

Список иллюстраций

3.1	снимок экрана	6
3.2	снимок экрана	6
3.3	снимок экрана	7
3.4	снимок экрана	7
3.5	снимок экрана	9
3.6	снимок экрана	10
3.7	снимок экрана	10
3.8	снимок экрана	10
3.9	снимок экрана	11
3.10	снимок экрана	11
3.11	снимок экрана	12
3.12	снимок экрана	12
3.13	снимок экрана	13
3.14	снимок экрана	13
3.15	снимок экрана	13
3.16	снимок экрана	13
3.17	снимок экрана	14
3.18	снимок экрана	14
3.19	снимок экрана	15
3.20	снимок экрана	17
3.21	снимок экрана	18

1 Цель работы

Приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

2 Задание

1. Реализация подпрограмм в NASM
2. Отладка программ с помощью GDB
3. Выполнение заданий для самостоятельной работы.

3 Выполнение лабораторной работы

3.1 Реализация подпрограмм в NASM

С помощью `mkdir` создаем директорию для создания файлов лабораторной работы, переходим в созданный каталог(рис. 3.1):

```
mkkuzjmina@VirtualBox:~$ mkdir ~/work/arch-pc/lab09
mkkuzjmina@VirtualBox:~$ cd ~/work/arch-pc/lab09
mkkuzjmina@VirtualBox:~/work/arch-pc/lab09$ touch lab09-1.asm
mkkuzjmina@VirtualBox:~/work/arch-pc/lab09$
```

Рис. 3.1: снимок экрана

Копируем в файл код из листинга и запускаем его, данная программа выполняет вычисление функции (рис. 3.2):

```
mkkuzjmina@VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
mkkuzjmina@VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
mkkuzjmina@VirtualBox:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2x+7=17
```

Рис. 3.2: снимок экрана

Изменяем текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $\varphi(\varphi(x))$, где x вводится с клавиатуры, $\varphi(x) = 2x + 7$, $\varphi(\varphi(x)) = 3x - 1$. Т.е. x передается в подпрограмму `_calcul` из нее в подпрограмму `_subcalcul`, где вычисляется выражение $\varphi(x)$, результат возвращается в `_calcul` и вычисляется выражение $\varphi(\varphi(x))$. (рис. 3.3)

```
*~/work/arch-pc/lab09/lab09-1.asm - Mousepad
Файл  Правка  Поиск  Просмотр  Документ  Помощь
;-----
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax, result
call sprint
mov eax, [res]
call iprintLF
call quit
;-----
; Подпрограмма вычисления
; выражения "2(3x-1)+7"
_calcul:
push eax
call _subcalcul
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
pop eax
ret
_subcalcul:
mov ebx, 3
mul ebx
sub eax, 1
ret ; выход из подпрограммы
```

Рис. 3.3: снимок экрана

(рис. 3.4):

```
mkkuzjmina@VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
mkkuzjmina@VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
mkkuzjmina@VirtualBox:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2(3x-1)+7=35
```

Рис. 3.4: снимок экрана

```
%include 'in_out.asm'

SECTION .data
```

```

msg: DB 'Введите x: ',0
result: DB '2(3x-1)+7=',0
SECTION .bss
x: RESB 80
res: RESB 80
SECTION .text
GLOBAL _start
_start:
;-----
; Основная программа
;-----

mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax,x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax,result
call sprint
mov eax,[res]
call iprintLF
call quit
;-----
; Подпрограмма вычисления
; выражения "2(3x-1)+7"
_calcul:
push eax

```



```

call _subcalcul
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
pop eax
ret
_subcalcul:
mov ebx, 3
mul ebx
sub eax, 1
ret ; выход из подпрограммы

```

3.2 Отладка программ с помощью GDB

Создаем файл lab09-2.asm с текстом программы, в созданный файл копируем программу второго листинга, транслируем с созданием файла листинга и отладки, компонуем и запускаем в отладчике (рис. 3.5):

```

mkkuzjmina@VirtualBox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
mkkuzjmina@VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
mkkuzjmina@VirtualBox:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 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:
<https://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 lab09-2...

```

Рис. 3.5: снимок экрана

Проверяем работу программы, запустив ее в оболочке GDB с помощью команды `run` (рис. 3.6):

```
(gdb) run
Starting program: /home/mkkuzjmina/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 5654) exited normally]
(gdb) Starting program: ~/work/arch-pc/lab09/lab09-2
Undefined command: "Starting". Try "help".
(gdb)
```

Рис. 3.6: снимок экрана

Для более подробного анализа программы устанавливаем брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запускаем её (рис. 3.7):

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/mkkuzjmina/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9
9      mov    eax, 4
(gdb)
```

Рис. 3.7: снимок экрана

Смотрим дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start` (рис. 3.8):

```
(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.
(gdb)
```

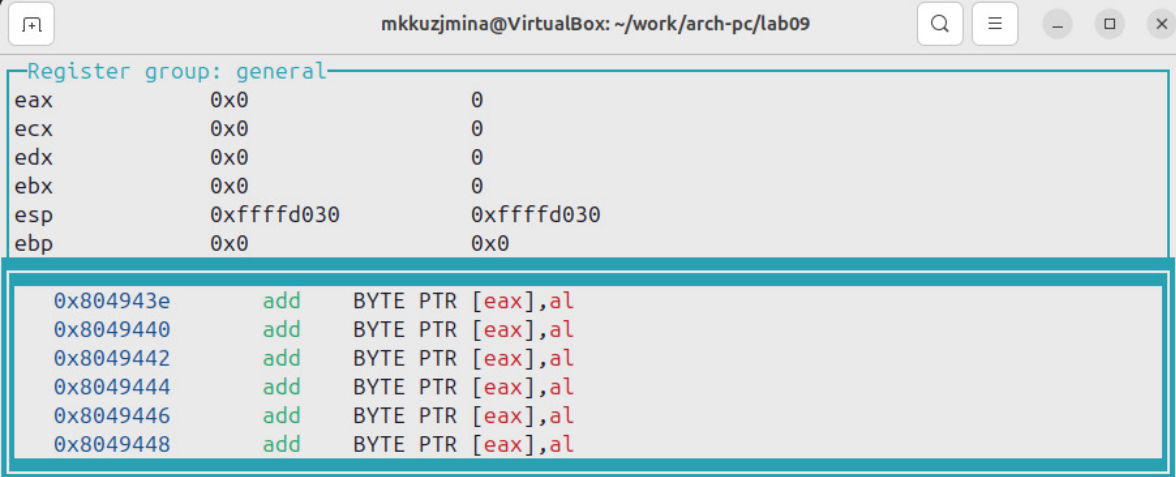
Рис. 3.8: снимок экрана

Переключаемся на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel` (рис. 3.9):

```
(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.
(gdb)
```

Рис. 3.9: снимок экрана

Включаем режим псевдографики для более удобного анализа программы (рис. 3.10):



The screenshot shows the GDB interface with the title bar 'mkuzjmina@VirtualBox: ~/work/arch-pc/lab09'. The 'Register group: general' section displays the following values:

Register	Value
eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xffffd030
ebp	0x0

Below the registers, a list of assembly instructions is shown, highlighted with a blue border:

```
0x804943e  add  BYTE PTR [eax],al
0x8049440  add  BYTE PTR [eax],al
0x8049442  add  BYTE PTR [eax],al
0x8049444  add  BYTE PTR [eax],al
0x8049446  add  BYTE PTR [eax],al
0x8049448  add  BYTE PTR [eax],al
```

At the bottom, the status bar indicates 'native process 5740 (asm) In: _start' with 'L9' and 'PC: 0x8049000'. The command history at the bottom shows '(gdb) layout regs' and '(gdb)'.

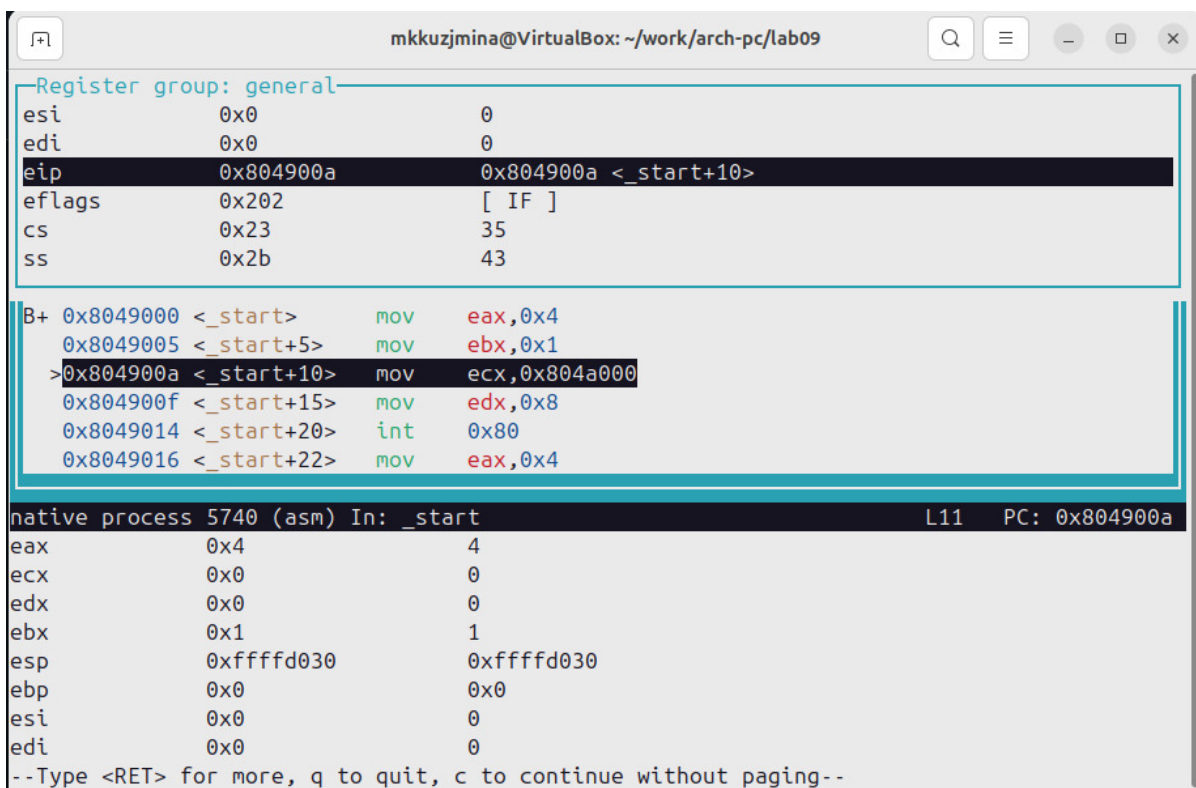
Рис. 3.10: снимок экрана

Устанавливаем точку останова (рис. 3.11):

```
native process 5740 (asm) In: _start PC: 0x0049000
(gdb) break *0x08049031
Breakpoint 2 at 0x08049031: file lab09-2.asm, line 20.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000 lab09-2.asm:9
          breakpoint already hit 1 time
2        breakpoint     keep y   0x08049031 lab09-2.asm:20
(gdb)
```

Рис. 3.11: снимок экрана

Смотрим содержимое регистров с помощью команды info registers (рис. 3.12):



The screenshot shows a debugger window titled 'mkkuzjmina@VirtualBox: ~/work/arch-pc/lab09'. It displays the 'Register group: general' section with the following values:

Register	Value	Comment
esi	0x0	0
edi	0x0	0
eip	0x804900a	0x804900a <_start+10>
eflags	0x202	[IF]
cs	0x23	35
ss	0x2b	43

Below the register list, the assembly code is shown with the current instruction highlighted:

```
B+ 0x8049000 <_start>    mov    eax,0x4
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
```

At the bottom, the 'native process 5740 (asm) In: _start' section shows the state of all registers:

Register	Value	Comment
eax	0x4	4
ecx	0x0	0
edx	0x0	0
ebx	0x1	1
esp	0xffffd030	0xffffd030
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0

The status bar at the bottom indicates 'L11 PC: 0x804900a' and provides instructions: '--Type <RET> for more, q to quit, c to continue without paging--'.

Рис. 3.12: снимок экрана

Смотрим значение переменной msg1 по имени (рис. 3.13):

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

Рис. 3.13: снимок экрана

Изменяем первый символ переменной msg1 (рис. 3.14):

```
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hhllo, "
(gdb) set {char}0x804a008='L'
(gdb) set {char}0x804a00b=' '
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hhllo, "
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "Lor d!\n\034"
(gdb)
```

Рис. 3.14: снимок экрана

Выводим в различных форматах значение регистра edx (рис. 3.15):

```
(gdb) p/s $eax
$3 = 4
(gdb) p/t $eax
$4 = 100
(gdb) p/s $eax
$5 = 4
(gdb) p/s $ecx
$6 = 0
(gdb) p/x $ecx
$7 = 0x0
```

Рис. 3.15: снимок экрана

С помощью команды set меняем содержимое регистра ebx (рис. 3.16):

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$8 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$9 = 2_
```

Рис. 3.16: снимок экрана

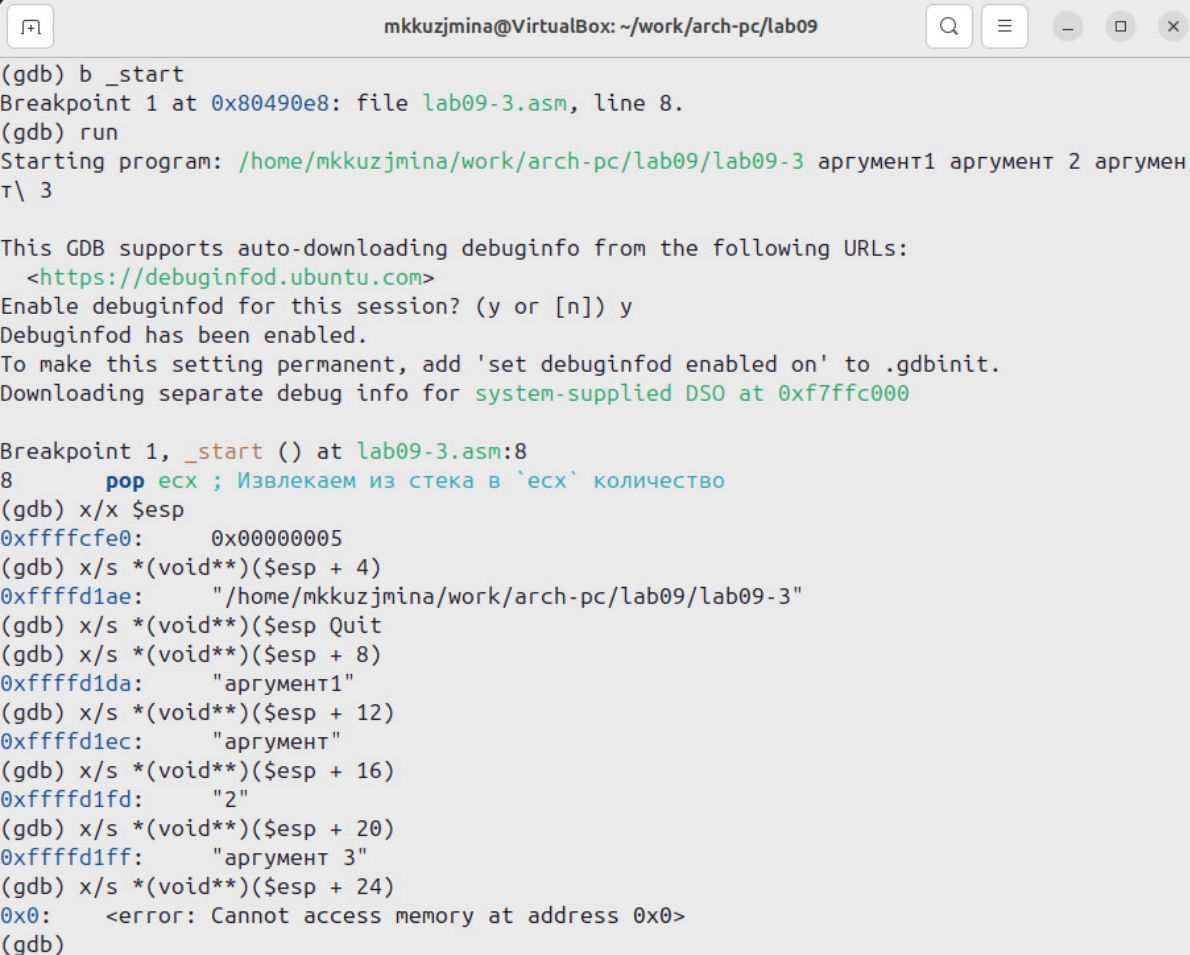
Копируем файл lab8-2.asm, созданный при выполнении лабораторной работы

№8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm (рис. 3.17):

```
mkkuzjmina@VirtualBox:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-
-pc/lab09/lab09-3.asm
mkkuzjmina@VirtualBox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
mkkuzjmina@VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
mkkuzjmina@VirtualBox:~/work/arch-pc/lab09$
```

Рис. 3.17: снимок экрана

Для загрузки в gdb программы с аргументами используем ключ `-args`. Загружаем исполняемый файл в отладчик, указав аргументы. Устанавливаем точку останова перед первой инструкцией в программе и запускаем ее (рис. 3.18):



```
mkkuzjmina@VirtualBox: ~/work/arch-pc/lab09
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 8.
(gdb) run
Starting program: /home/mkkuzjmina/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумен
т\ 3

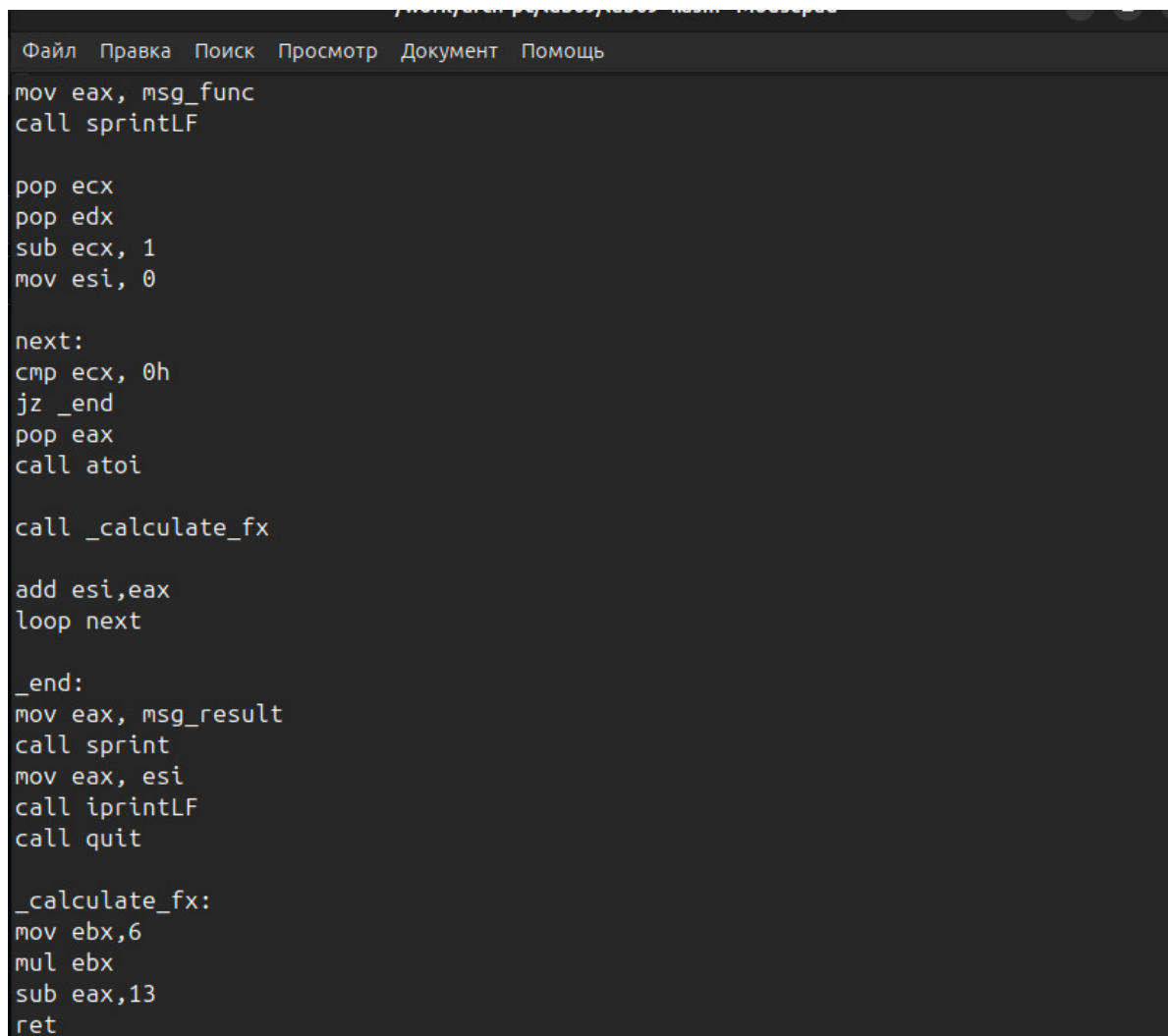
This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000

Breakpoint 1, _start () at lab09-3.asm:8
8      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffcfe0: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffd1ae: "/home/mkkuzjmina/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd1da: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd1ec: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd1fd: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd1ff: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 3.18: снимок экрана

3.3 Выполнение заданий для самостоятельной работы

Преобразовываем программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму (рис. 3.19):



```
Файл  Правка  Поиск  Просмотр  Документ  Помощь
mov eax, msg_func
call sprintfLF

pop ecx
pop edx
sub ecx, 1
mov esi, 0

next:
cmp ecx, 0h
jz _end
pop eax
call atoi

call _calculate_fx

add esi, eax
loop next

_end:
mov eax, msg_result
call sprintf
mov eax, esi
call iprintfLF
call quit

_calculate_fx:
mov ebx, 6
mul ebx
sub eax, 13
ret
```

Рис. 3.19: снимок экрана

```
%include 'in_out.asm'
```

```
SECTION .data
```

```
msg_func db "Функция:  $f(x) = 6x + 13$ ", 0
```

```
msg_result db "Результат: ", 0
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
mov eax, msg_func
```

```
call sprintf
```

```
pop ecx
```

```
pop edx
```

```
sub ecx, 1
```

```
mov esi, 0
```

```
next:
```

```
cmp ecx, 0
```

```
jz _end
```

```
pop eax
```

```
call atoi
```

```
mov ebx, 6
```

```
mul ebx
```

```
add eax, 13
```

```
add esi, eax
```

```
loop next
```

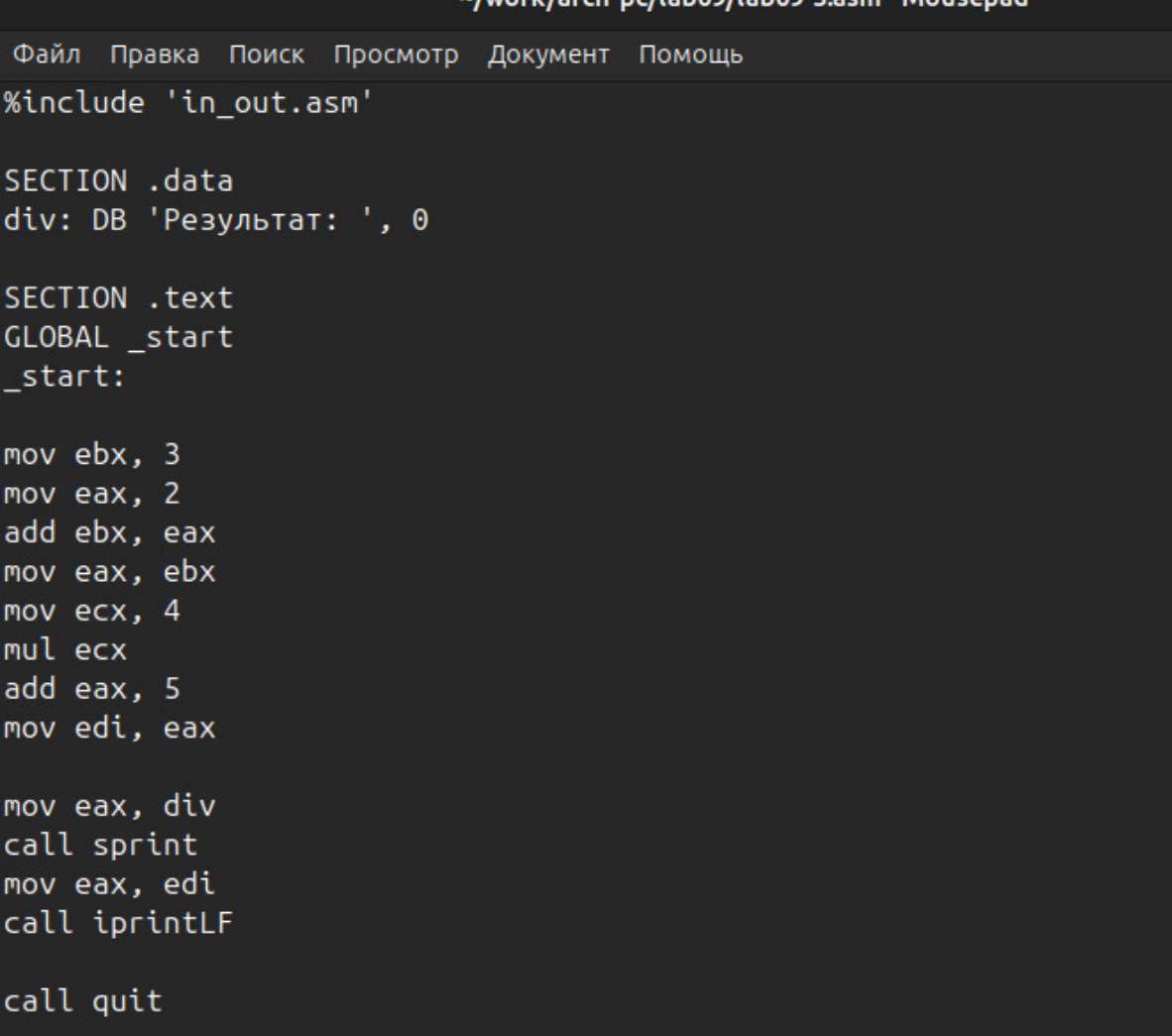
```
_end:
```

```
mov eax, msg_result
```



```
call sprint
mov eax, esi
call iprintLF
call quit
```

Запускаем программу в режиме отладчика и пошагово просматриваем изменение значений регистров через `i`. При выполнении инструкции `mul ecx` можно заметить, что результат умножения записывается в регистр `eax`, но также меняет и `edx`. Значение регистра `ebx` не обновляется напрямую, поэтому результат программа неверно подсчитывает функцию (рис. 3.20):



```
Файл  Правка  Поиск  Просмотр  Документ  Помощь
%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 eax, ebx
mov ecx, 4
mul ecx
add eax, 5
mov edi, eax

mov eax, div
call sprint
mov eax, edi
call iprintLF

call quit
```

Рис. 3.20: снимок экрана

```

#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 eax, ebx
mov ecx, 4
mul ecx
add eax, 5
mov edi, eax

mov eax, div
call sprint
mov eax, edi
call iprintLF

call quit

```

Исправляем найденную ошибку, теперь программа верно считает значение функции (рис. 3.21):



```

mkkuzjmina@VirtualBox:~/work/arch-pc/lab09$ ./lab09-5
Результат: 25

```

Рис. 3.21: снимок экрана

4 Выводы

При выполнении лабораторной работы были приобретены навыки написания программ с использованием подпрограмм. Ознакомились с методами отладки при помощи GDB и его основными возможностями.