

# **Отчёт по лабораторной работе 9**

**Архитектура компьютеров**

Махкамов Рауфджон НММбд-04-24

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
2.1	Реализация подпрограмм в NASM . . . . .	6
2.2	Отладка программы с помощью GDB . . . . .	9
2.3	Задание для самостоятельной работы . . . . .	19
<b>3</b>	<b>Выводы</b>	<b>25</b>

## Список иллюстраций

2.1	Текст программы lab9-1.asm . . . . .	7
2.2	Запуск программы lab9-1.asm . . . . .	7
2.3	Модифицированная программа lab9-1.asm . . . . .	8
2.4	Запуск модифицированной программы lab9-1.asm . . . . .	9
2.5	Код программы lab9-2.asm . . . . .	10
2.6	Запуск программы lab9-2.asm в GDB . . . . .	11
2.7	Дизассемблированный код программы . . . . .	12
2.8	Дизассемблированный код в Intel-синтаксисе . . . . .	13
2.9	Настройка точки останова . . . . .	14
2.10	Отслеживание изменений регистров . . . . .	15
2.11	Детальный анализ регистров . . . . .	16
2.12	Изменение значения переменной msg1 . . . . .	17
2.13	Просмотр регистра после изменений . . . . .	18
2.14	Анализ стека программы . . . . .	19
2.15	Код программы lab9-prog.asm . . . . .	20
2.16	Запуск программы lab9-prog.asm . . . . .	20
2.17	Код с ошибкой . . . . .	21
2.18	Процесс отладки программы . . . . .	22
2.19	Исправленный код программы . . . . .	23
2.20	Проверка исправленного кода . . . . .	24

## **Список таблиц**

# 1 Цель работы

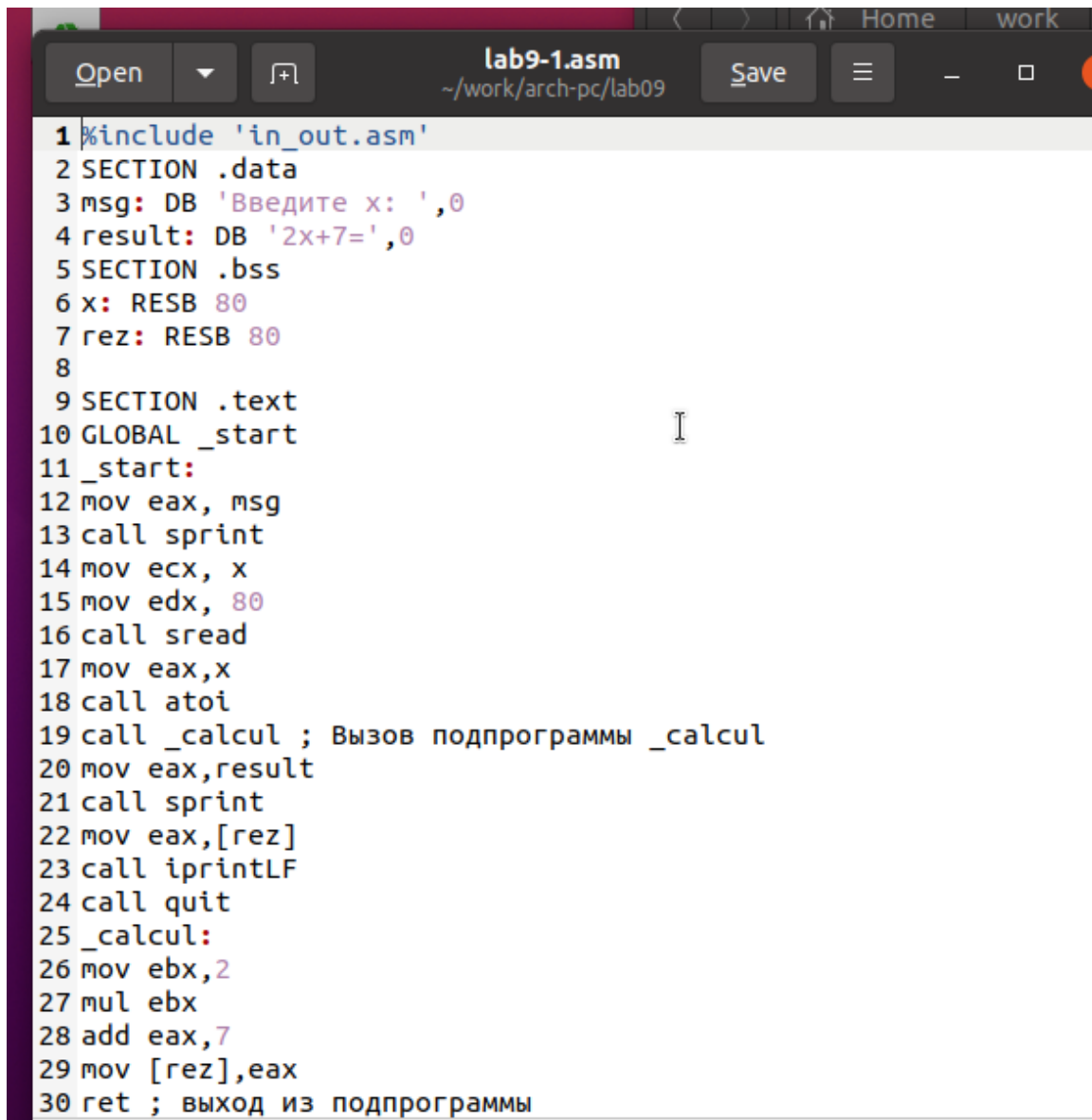
Целью работы является приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

## 2 Выполнение лабораторной работы

### 2.1 Реализация подпрограмм в NASM

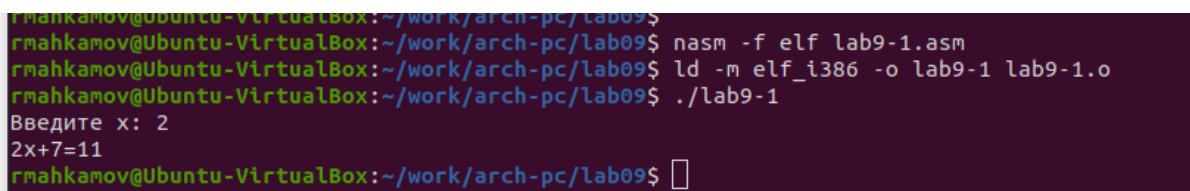
Для выполнения лабораторной работы №9 я создал новую папку и перешел в нее. Затем я создал файл с именем lab9-1.asm.

В качестве примера была рассмотрена программа, которая вычисляет арифметическое выражение  $f(x) = 2x + 7$  с использованием подпрограммы `calcul`. Значение переменной  $x$  вводится с клавиатуры, а вычисление производится внутри подпрограммы. (рис. 2.1) (рис. 2.2)



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2x+7=',0
5 SECTION .bss
6 x: RESB 80
7 rez: RESB 80
8
9 SECTION .text
10 GLOBAL _start
11 _start:
12 mov eax, msg
13 call sprint
14 mov ecx, x
15 mov edx, 80
16 call sread
17 mov eax,x
18 call atoi
19 call _calcul ; Вызов подпрограммы _calcul
20 mov eax,result
21 call sprint
22 mov eax,[rez]
23 call iprintLF
24 call quit
25 _calcul:
26 mov ebx,2
27 mul ebx
28 add eax,7
29 mov [rez],eax
30 ret ; выход из подпрограммы
```

Рис. 2.1: Текст программы lab9-1.asm

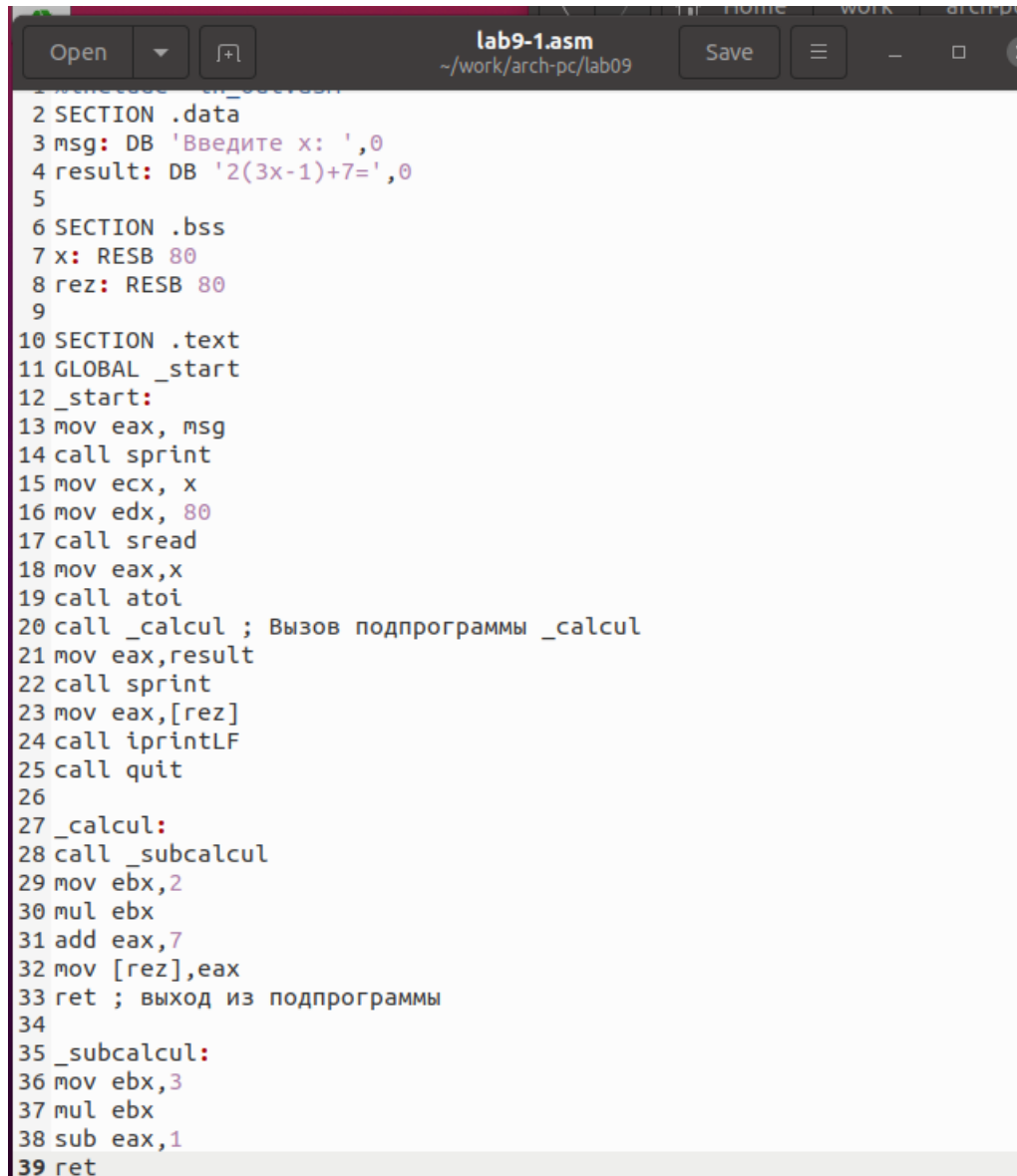


```
rmaikamov@Ubuntu-VirtualBox:~/work/arch-pc/lab09$
rmaikamov@Ubuntu-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
rmaikamov@Ubuntu-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
rmaikamov@Ubuntu-VirtualBox:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 2
2x+7=11
rmaikamov@Ubuntu-VirtualBox:~/work/arch-pc/lab09$
```

Рис. 2.2: Запуск программы lab9-1.asm

Далее я модифицировал программу, добавив подпрограмму subcalcul внутрь

подпрограммы `calcul`. Это позволило вычислять составное выражение  $f(g(x))$ , где  $f(x) = 2x + 7$ , а  $g(x) = 3x - 1$ . Значение  $x$  вводится с клавиатуры. (рис. 2.3) (рис. 2.4)



```
1 #include <unistd.h>
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2(3x-1)+7=',0
5
6 SECTION .bss
7 x: RESB 80
8 rez: RESB 80
9
10 SECTION .text
11 GLOBAL _start
12 _start:
13 mov eax, msg
14 call sprint
15 mov ecx, x
16 mov edx, 80
17 call sread
18 mov eax, x
19 call atoi
20 call _calcul ; Вызов подпрограммы _calcul
21 mov eax, result
22 call sprint
23 mov eax, [rez]
24 call iprintLF
25 call quit
26
27 _calcul:
28 call _subcalcul
29 mov ebx, 2
30 mul ebx
31 add eax, 7
32 mov [rez], eax
33 ret ; выход из подпрограммы
34
35 _subcalcul:
36 mov ebx, 3
37 mul ebx
38 sub eax, 1
39 ret
```

Рис. 2.3: Модифицированная программа `lab9-1.asm`

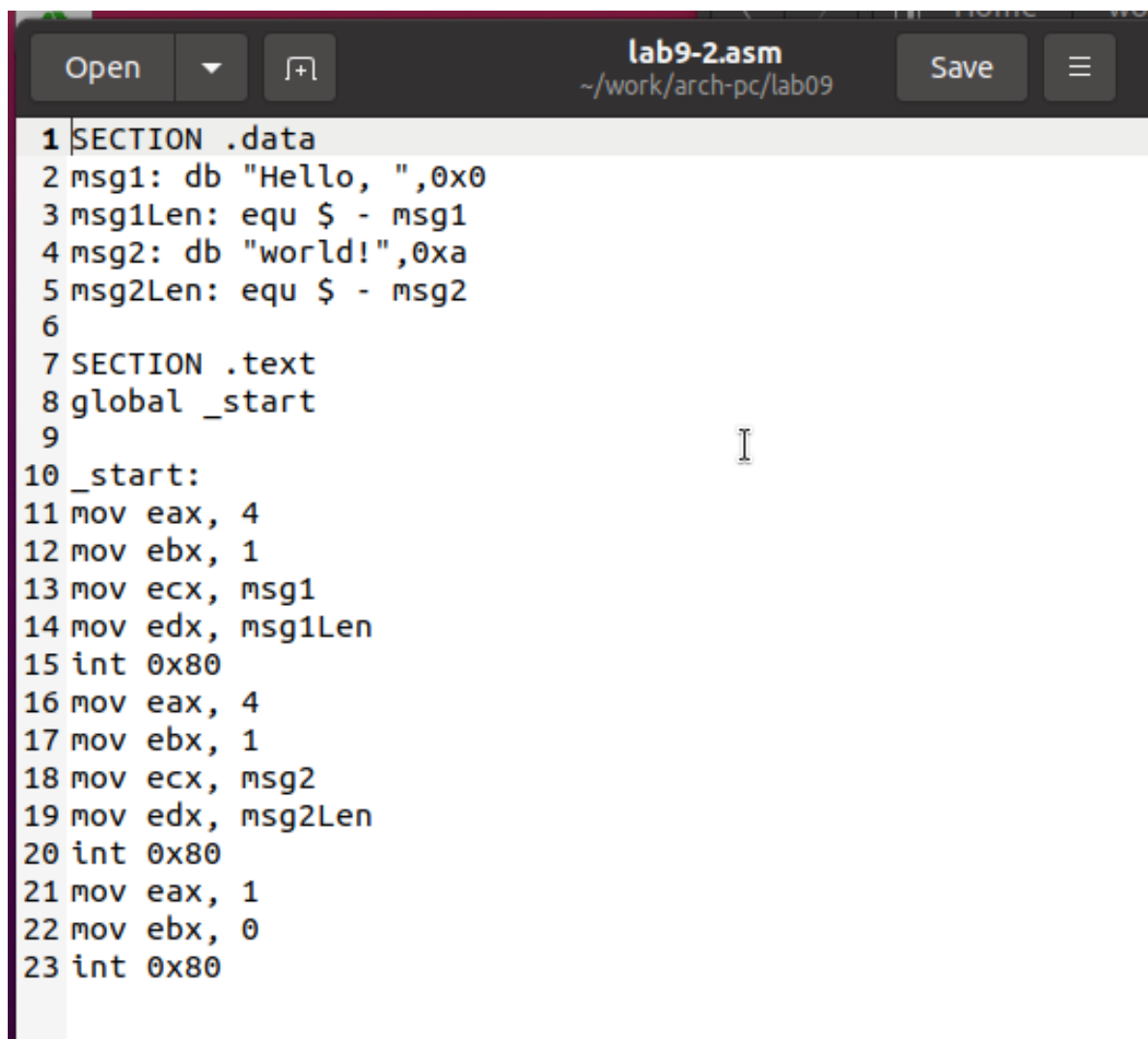


```
rmahkamov@Ubuntu-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
rmahkamov@Ubuntu-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
rmahkamov@Ubuntu-VirtualBox:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 2
2x+7=11
rmahkamov@Ubuntu-VirtualBox:~/work/arch-pc/lab09$
rmahkamov@Ubuntu-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
rmahkamov@Ubuntu-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
rmahkamov@Ubuntu-VirtualBox:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 2
2(3x-1)+7=17
rmahkamov@Ubuntu-VirtualBox:~/work/arch-pc/lab09$
```

Рис. 2.4: Запуск модифицированной программы lab9-1.asm

## 2.2 Отладка программы с помощью GDB

Я создал файл lab9-2.asm, в котором содержится программа из Листинга 9.2. Она отвечает за вывод сообщения “Hello world!” на экран. (рис. 2.5)



```
1 SECTION .data
2 msg1: db "Hello, ",0x0
3 msg1Len: equ $ - msg1
4 msg2: db "world!",0xa
5 msg2Len: equ $ - msg2
6
7 SECTION .text
8 global _start
9
10 _start:
11 mov eax, 4
12 mov ebx, 1
13 mov ecx, msg1
14 mov edx, msg1Len
15 int 0x80
16 mov eax, 4
17 mov ebx, 1
18 mov ecx, msg2
19 mov edx, msg2Len
20 int 0x80
21 mov eax, 1
22 mov ebx, 0
23 int 0x80
```

Рис. 2.5: Код программы lab9-2.asm

После компиляции с ключом `-g` для добавления отладочной информации я загрузил исполняемый файл в GDB. Запустил программу с помощью команды `run` или `r`. (рис. 2.6)

```
rmahkamov@Ubuntu-VirtualBox:~/work/arch-pc/lab09$  
rmahkamov@Ubuntu-VirtualBox:~/work/arch-pc/lab09$ gdb lab9-2  
  
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.2) 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 lab9-2...  
(gdb) run  
Starting program: /home/rmahkamov/work/arch-pc/lab09/lab9-2  
Hello, world!  
[Inferior 1 (process 2657) exited normally]  
(gdb) █
```

Рис. 2.6: Запуск программы lab9-2.asm в GDB

Для анализа программы я установил точку остановки на метке `_start` и запустил выполнение. Затем изучил дизассемблированный код программы. (рис. 2.7) (рис. 2.8)

```
rmahkamov@Ubuntu-VirtualBox: ~/work/arch-pc/lab09
<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 lab9-2...
(gdb) run
Starting program: /home/rmahkamov/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 2657) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000
(gdb) run
Starting program: /home/rmahkamov/work/arch-pc/lab09/lab9-2

Breakpoint 1, 0x8049000 in _start ()
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x8049000 <+0>:    mov     $0x4,%eax
      0x8049005 <+5>:    mov     $0x1,%ebx
      0x804900a <+10>:   mov     $0x804a000,%ecx
      0x804900f <+15>:   mov     $0x8,%edx
      0x8049014 <+20>:   int     $0x80
      0x8049016 <+22>:   mov     $0x4,%eax
      0x804901b <+27>:   mov     $0x1,%ebx
      0x8049020 <+32>:   mov     $0x804a008,%ecx
      0x8049025 <+37>:   mov     $0x7,%edx
      0x804902a <+42>:   int     $0x80
      0x804902c <+44>:   mov     $0x1,%eax
      0x8049031 <+49>:   mov     $0x0,%ebx
      0x8049036 <+54>:   int     $0x80
End of assembler dump.
(gdb) 
```

Рис. 2.7: Дизассемблированный код программы

```
rmahkamov@Ubuntu-VirtualBox: ~/work/arch-pc/lab09
(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) 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. 2.8: Дизассемблированный код в Intel-синтаксисе

Для проверки точки останова я использовал команду `info breakpoints (i b)`. Установил дополнительную точку останова по адресу инструкции `mov ebx, 0x0`. (рис. 2.9)

```
rmahkamov@Ubuntu-VirtualBox: ~/work/arch-pc/lab09

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1d0 0xffffd1d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>

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
0x804901b <_start+27> mov    ebx,0x1
0x8049020 <_start+32> mov    ecx,0x804a008
0x8049025 <_start+37> mov    edx,0x7
0x804902a <_start+42> int    0x80

native process 2669 In: _start L?? PC: 0x8049000
(gdb) layout regs
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031
(gdb) i b
Num    Type           Disp Enb Address      What
1      breakpoint      keep y   0x08049000 <_start>
       breakpoint already hit 1 time
2      breakpoint      keep y   0x08049031 <_start+49>
(gdb) █
```

Рис. 2.9: Настройка точки останова

С помощью команды `stepi (si)` выполнил пошаговую отладку, отслеживая изменения регистров. (рис. 2.10) (рис. 2.11)

```
rmahkamov@Ubuntu-VirtualBox: ~/work/arch-pc/lab09

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1d0 0xffffd1d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049005 0x8049005 <_start+5>

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
0x804901b <_start+27> mov ebx,0x1
0x8049020 <_start+32> mov ecx,0x804a008
0x8049025 <_start+37> mov edx,0x7
0x804902a <_start+42> int 0x80

native process 2669 In: _start L?? PC: 0x8049005
eflags 0x202 [ IF ]
--Type <RET> for more, q to quit, c to continue without paging--
cs      0x23      35
ss      0x2b      43
ds      0x2b      43
es      0x2b      43
fs      0x0      0
gs      0x0      0
(gdb) si
0x8049005 in _start ()
(gdb) 
```

Рис. 2.10: Отслеживание изменений регистров

```
rmahkamov@Ubuntu-VirtualBox: ~/work/arch-pc/lab09

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1d0 0xffffd1d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>

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
0x804901b <_start+27>   mov    ebx,0x1
0x8049020 <_start+32>   mov    ecx,0x804a008
0x8049025 <_start+37>   mov    edx,0x7
0x804902a <_start+42>   int     0x80

native process 2669 In: _start L?? PC: 0x8049016
(gdb) si
0x08049005 in _start ()
(gdb) si
0x0804900a in _start ()
(gdb) si
0x0804900f in _start ()
(gdb) si
0x08049014 in _start ()
(gdb) si
0x08049016 in _start ()
(gdb) 
```

Рис. 2.11: Детальный анализ регистров

Я также просмотрел значение переменной `msg1` по имени и изменил первый символ переменной с помощью команды `set`. (рис. 2.12) (рис. 2.13)



```
rmahkamov@Ubuntu-VirtualBox: ~/work/arch-pc/lab09

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1d0 0xffffd1d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>

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
0x804901b <_start+27>   mov    ebx,0x1
0x8049020 <_start+32>   mov    ecx,0x804a008
0x8049025 <_start+37>   mov    edx,0x7
0x804902a <_start+42>   int     0x80

native process 2669 In: _start L?? PC: 0x8049016
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n"
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb) set {char}0x804a008='L'
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "Lor!d!\n"
(gdb) █
```

Рис. 2.12: Изменение значения переменной msg1

```
rmahkamov@Ubuntu-VirtualBox: ~/work/arch-pc/lab09

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1d0 0xffffd1d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>

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
    0x804901b <_start+27> mov    ebx,0x1
    0x8049020 <_start+32> mov    ecx,0x804a008
    0x8049025 <_start+37> mov    edx,0x7
    0x804902a <_start+42> int     0x80

native process 2669 In: _start L?? PC: 0x8049016
(gdb) p/s $ecx
$3 = 134520832
(gdb) p/x $ecx
$4 = 0x804a000
(gdb) p/s $edx
$5 = 8
(gdb) p/t $edx
$6 = 1000
(gdb) p/x $edx
$7 = 0x8
(gdb) █
```

Рис. 2.13: Просмотр регистра после изменений

Для проверки программы с аргументами я скопировал файл lab8-2.asm из лабораторной работы №8, создал исполняемый файл и загрузил его в GDB с помощью ключа `-args`. Затем исследовал стек, где хранились адреса аргументов. (рис. 2.14)

```
rmahkamov@Ubuntu-VirtualBox: ~/work/arch-pc/lab09
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 lab9-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8
(gdb) run
Starting program: /home/rmahkamov/work/arch-pc/lab09/lab9-3 argument 1 argument 2 argument\
3

Breakpoint 1, 0x080490e8 in _start ()
(gdb) x/x $esp
0xfffffd190: 0x00000006
(gdb)
0xfffffd194: 0xfffffd352
(gdb) x/s *(void**)(esp + 4)
0xfffffd352: "/home/rmahkamov/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xfffffd37c: "argument"
(gdb) x/s *(void**)(esp + 12)
0xfffffd385: "1"
(gdb) x/s *(void**)(esp + 16)
0xfffffd387: "argument"
(gdb) x/s *(void**)(esp + 20)
0xfffffd390: "2"
(gdb) x/s *(void**)(esp + 24)
0xfffffd392: "argument 3"
(gdb)
```

Рис. 2.14: Анализ стека программы

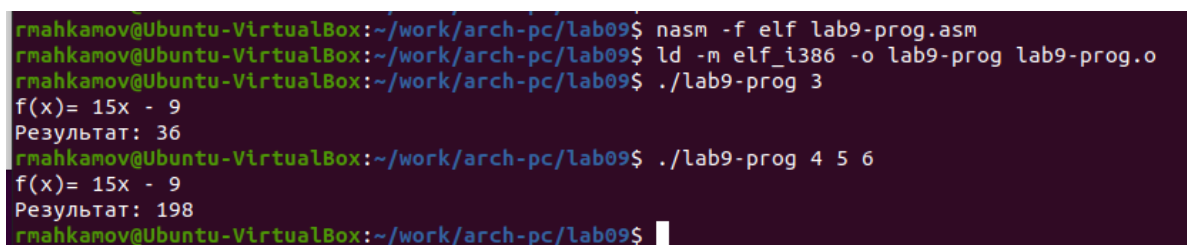
## 2.3 Задание для самостоятельной работы

Я модифицировал программу из лабораторной работы №8, добавив вычисление функции  $f(x)$  в виде подпрограммы. (рис. 2.15) (рис. 2.16)



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 fx: db 'f(x)= 15x - 9',0
5
6 SECTION .text
7 global _start
8 _start:
9 mov eax, fx
10 call sprintf
11 pop ecx
12 pop edx
13 sub ecx,1
14 mov esi, 0
15
16 next:
17 cmp ecx,0h
18 jz _end
19 pop eax
20 call atoi
21 call _calcul
22 add esi,eax
23
24 loop next
25
26 _end:
27 mov eax, msg
28 call sprintf
29 mov eax, esi
30 call iprintLF
31 call quit
32
33 _calcul:
34 mov ebx,15
35 mul ebx
36 sub eax,9
37 ret
```

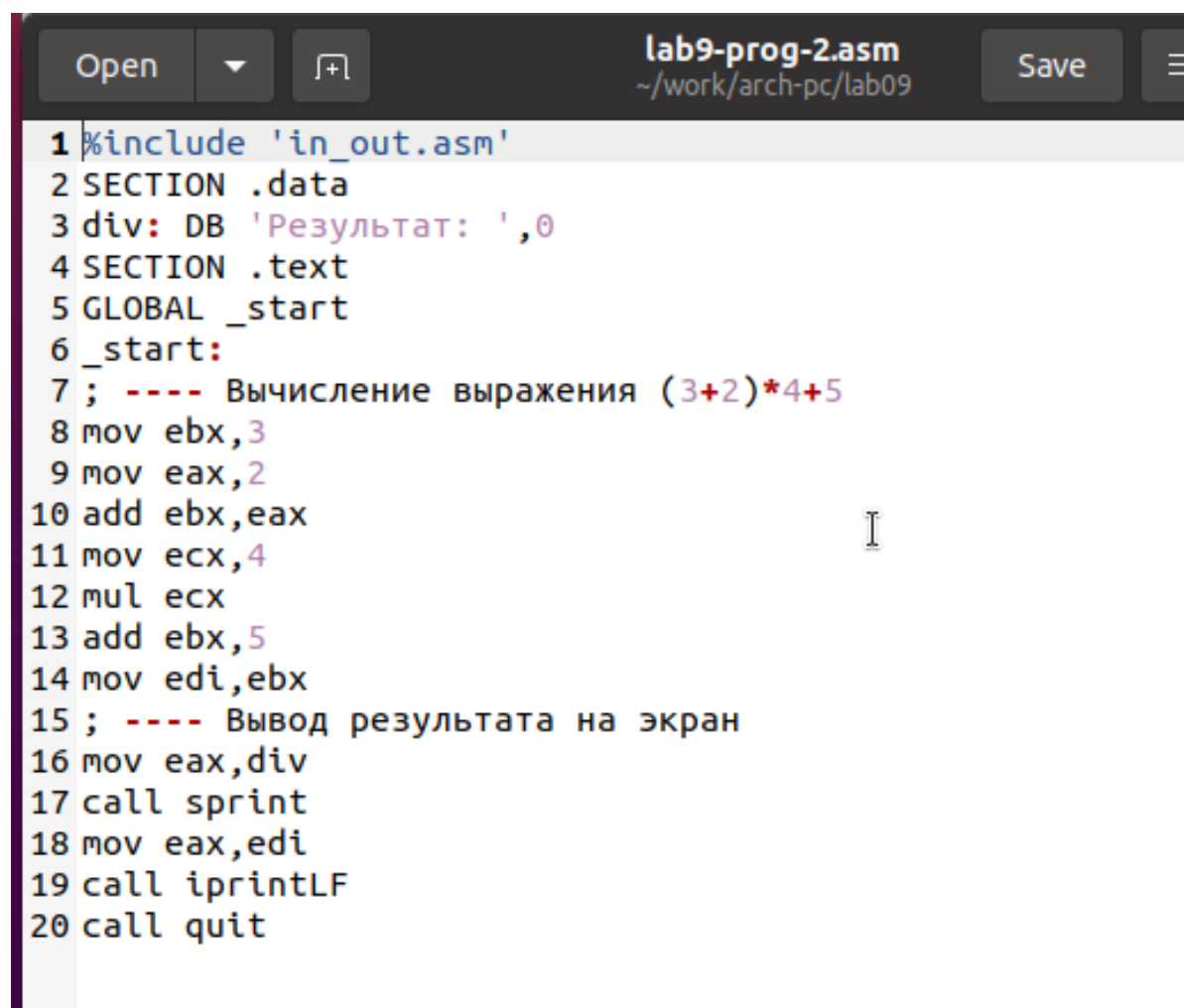
Рис. 2.15: Код программы lab9-prog.asm



```
rmahkamov@Ubuntu-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab9-prog.asm
rmahkamov@Ubuntu-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-prog lab9-prog.o
rmahkamov@Ubuntu-VirtualBox:~/work/arch-pc/lab09$ ./lab9-prog 3
f(x)= 15x - 9
Результат: 36
rmahkamov@Ubuntu-VirtualBox:~/work/arch-pc/lab09$ ./lab9-prog 4 5 6
f(x)= 15x - 9
Результат: 198
rmahkamov@Ubuntu-VirtualBox:~/work/arch-pc/lab09$
```

Рис. 2.16: Запуск программы lab9-prog.asm

При запуске программы я обнаружил ошибку: результат вычислений был неверным. Анализ с помощью GDB показал, что аргументы инструкции `add` перепутаны, а по окончании программы значение регистра `ebx` вместо `eax` отправляется в `edi`. (рис. 2.17) (рис. 2.18)



```
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ---- Вычисление выражения (3+2)*4+5
8 mov ebx,3
9 mov eax,2
10 add ebx,eax
11 mov ecx,4
12 mul ecx
13 add ebx,5
14 mov edi,ebx
15 ; ---- Вывод результата на экран
16 mov eax,div
17 call sprint
18 mov eax,edi
19 call iprintLF
20 call quit
```

Рис. 2.17: Код с ошибкой

```

rmahkamov@Ubuntu-VirtualBox: ~/work/arch-pc/lab09
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd1c0 0xffffd1c0
ebp      0x0      0x0
esi      0x0      0
edi      0xa      10
eip      0x8049100 0x8049100 <_start+24>

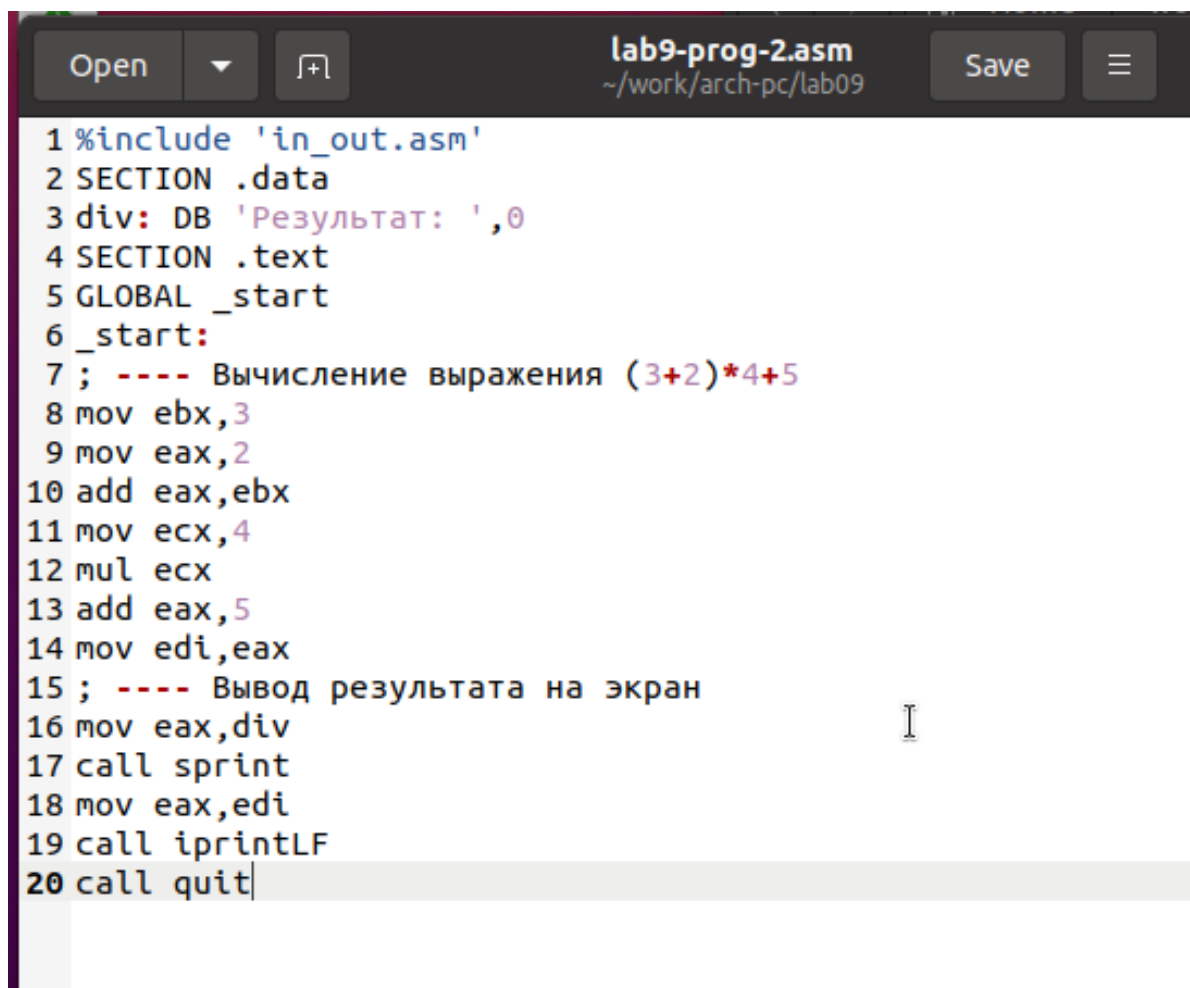
B+ 0x80490e8 <_start>    mov     ebx,0x3
B+ 0x80490e8 <_start>5>  mov     ebx,0x3
0x80490ed <_start+5>    mov     eax,0x2
0x80490f2 <_start+10>   add     ebx,eax
0x80490f4 <_start+12>   mov     ecx,0x4
0x80490f9 <_start+17>   mul     ecx,0x5
0x80490fb <_start+19>   add     ebx,0x5
>0x80490fe <_start+22>   mov     edi,ebx04a000
0x8049100 <_start+24>   mov     eax,0x804a000rint>
0x8049105 <_start+29>   call   0x804900f <sprint>
0x804910a <_start+34>   mov     eax,edi

native process 2727 In: _start L?? PC: 0x8049100
(gdb) sNo process In: L?? PC: ??
(gdb) si
0x080490fb in _start ()
(gdb) si
0x080490fe in _start ()
(gdb) si
0x08049100 in _start ()
(gdb) c
Continuing.
Результат: 10
[Inferior 1 (process 2727) exited normally]
(gdb)

```

Рис. 2.18: Процесс отладки программы

После исправления ошибок я проверил работу программы. (рис. 2.19) (рис. 2.20)



```
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ---- Вычисление выражения (3+2)*4+5
8 mov ebx,3
9 mov eax,2
10 add eax,ebx
11 mov ecx,4
12 mul ecx
13 add eax,5
14 mov edi,eax
15 ; ---- Вывод результата на экран
16 mov eax,div
17 call sprint
18 mov eax,edi
19 call iprintLF
20 call quit
```

Рис. 2.19: Исправленный код программы

```
rmahkamov@Ubuntu-VirtualBox: ~/work/arch-pc/lab09

eax      0x19      25
ecx      0x4       4
edx      0x0       0
ebx      0x3       3
esp      0xffffd1c0 0xffffd1c0
ebp      0x0       0
esi      0x0       0
edi      0x19      25
eip      0x8049100 0x8049100 <_start+24>

B+ 0x80490e8 <_start>    mov     ebx,0x3
B+ 0x80490e8 <_start>5>  mov     ebx,0x3
0x80490ed <_start+5>    mov     eax,0x2
0x80490f2 <_start+10>   add     eax,ebx
0x80490f4 <_start+12>   mov     ecx,0x4
0x80490f9 <_start+17>   mul     ecx,0x5
0x80490fb <_start+19>   add     eax,0x5
>0x80490fe <_start+22>   mov     edi,eax04a000
0x8049100 <_start+24>   mov     eax,0x804a000rint>
0x8049105 <_start+29>   call   0x804900f <sprint>
0x804910a <_start+34>   mov     eax,edi

native process 2739 In: _start L?? PC: 0x8049100
(gdb) sNo process In: L?? PC: ??
(gdb) si
0x080490fb in _start ()
(gdb) si
0x080490fe in _start ()
(gdb) si
0x08049100 in _start ()
(gdb) c
Continuing.
Результат: 25
[Inferior 1 (process 2739) exited normally]
(gdb) █
```

Рис. 2.20: Проверка исправленного кода



## 3 Выводы

Я освоил работу с подпрограммами и отладчиком GDB, научился находить и исправлять ошибки в коде с помощью анализа стеков, регистров и дизассемблированного кода.