

# **Лабораторная работа №9**

**Понятие подпрограммы. Отладчик GDB.**

Седохин Даниил Алексеевич

# Содержание

1	Цель работы	4
2	Выполнение лабораторной работы	5
3	Задание для самостоятельной работы	25
4	Выводы	30

## Список иллюстраций

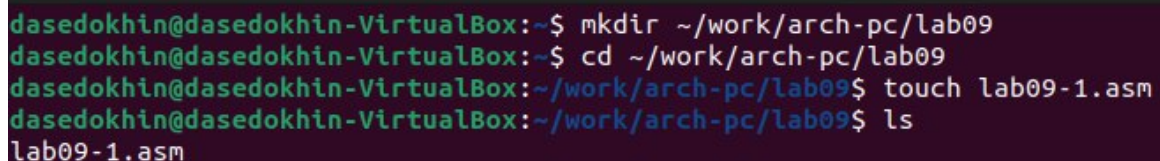
2.1	Создание каталога для лабораторной работы №9 и файла lab9-1.asm	5
2.2	Ввод в файл lab9-1.asm текст программы из листинга 9.1 . . . . .	6
2.3	Создание и проверка исполняемого файла lab9-1.asm . . . . .	7
2.4	Изменение текста программы, добавив команду подпрограммы _subcalcul . . . . .	8
2.5	Создание и проверка исполняемого файла . . . . .	9
2.6	Создание файла lab09-2.asm . . . . .	9
2.7	Ввод текст программы из листинга 9.2. . . . .	10
2.8	Получение исполняемого файла . . . . .	10
2.9	Загрузка файла в отладчик gdb . . . . .	11
2.10	Проверка работы программы . . . . .	11
2.11	Подробный анализ программы _start . . . . .	12
2.12	Дисассимилированный код программы . . . . .	12
2.13	Переключение на отображение команд intel синтаксисом . . . . .	13
2.14	Режим псевдографики . . . . .	14
2.15	layout regs . . . . .	15
2.16	Проверка установки точки _start . . . . .	16
2.17	Определение адреса . . . . .	17
2.18	Просмотр информации о всех установленных точках останова . . . . .	18
2.19	Просмотр значения msg1 и msg2 . . . . .	19
2.20	Изменение первого символа переменной msg1 . . . . .	19
2.21	Изменение первого символа переменной msg2 . . . . .	20
2.22	Изменение значение регистра ebx . . . . .	21
2.23	Копирование файла lab8-2.asm в файл lab09-3.asm . . . . .	22
2.24	Создание исполняемого файла . . . . .	22
2.25	Загрузка исполняемого файла в отладчик . . . . .	23
2.26	Установка точки останова в программе и её запуск . . . . .	23
2.27	Адрес вершины стека . . . . .	24
2.28	Просмотр позиций стека . . . . .	24
3.1	Текст файла lab09-4.asm . . . . .	26
3.2	Создание исполняемого файла и его проверка . . . . .	27
3.3	Текст файла lab09-5.asm . . . . .	27
3.4	Создание и проверка исполняемого файла . . . . .	28
3.5	Правильный код программы lab09-5.asm . . . . .	28
3.6	Создание исполняемого файла и его проверка . . . . .	29

# 1 Цель работы

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

## 2 Выполнение лабораторной работы

- 1) Создадим каталог для выполнения лабораторной работы № 9, перейдём в него и создадим файл lab09-1.asm: (рис. 2.1).



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

Рис. 2.1: Создание каталога для лабораторной работы №9 и файла lab9-1.asm

- 2) В качестве примера рассмотрим программу вычисления арифметического выражения  $f(x) = 2x + 7$  с помощью подпрограммы `_calcul`. В данном примере `x` вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Введём в файл lab9-1.asm текст программы из листинга 9.1. (рис. 2.2).

```

GNU nano 6.2 /home/dasedokhin/work/arch-pc/lab09/lab09-1.asm
#include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+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 "2x+7"
_calcul:
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы

```

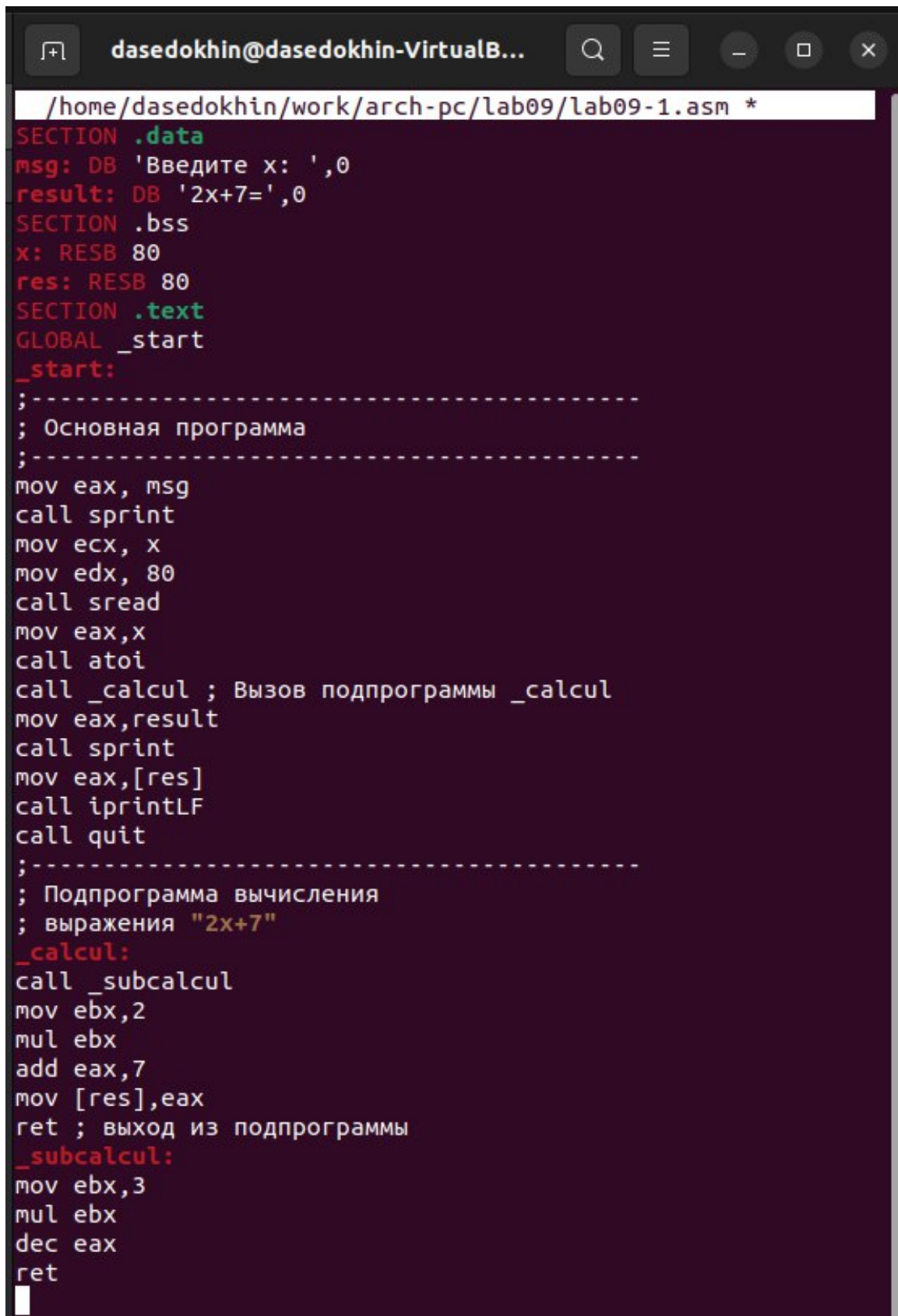
Рис. 2.2: Ввод в файл lab9-1.asm текст программы из листинга 9.1

Создадим исполняемый файл и проверим его работу. (рис. 2.3).

```
dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab09-1
.asm
dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o l
ab09-1 lab09-1.o
dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2x+7=17
dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$
```

Рис. 2.3: Создание и проверка исполняемого файла lab9-1.asm

Изменим текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения  $f(g(x))$ , где  $x$  вводится с клавиатуры,  $f(x) = 2x + 7$ ,  $g(x) = 3x - 1$ . Т.е.  $x$  передается в подпрограмму `_calcul` из нее в подпрограмму `_subcalcul`, где вычисляется выражение  $g(x)$ , результат возвращается в `_calcul` и вычисляется выражение  $f(g(x))$ . Результат возвращается в основную программу для вывода результата на экран. Отредактируем текст программы (рис. 2.4).



```
dasedokhin@dasedokhin-VirtualB...
/home/dasedokhin/work/arch-pc/lab09/lab09-1.asm *
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+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 "2x+7"
_calcul:
call _subcalcul
mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret ; выход из подпрограммы
_subcalcul:
mov ebx,3
mul ebx
dec eax
ret
```

Рис. 2.4: Изменение текста программы, добавив команду подпрограммы `_subcalcul`



Создадим исполняемый файл и проверим его работу. (рис. 2.5).

```
[11]  Установка...
dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$ nasm
-f elf lab09-1.asm
dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$ ld -
m elf_i386 -o lab09-1 lab09-1.o
dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$ ./la
b09-1
Введите x: 2
2x+7=17
dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$
```

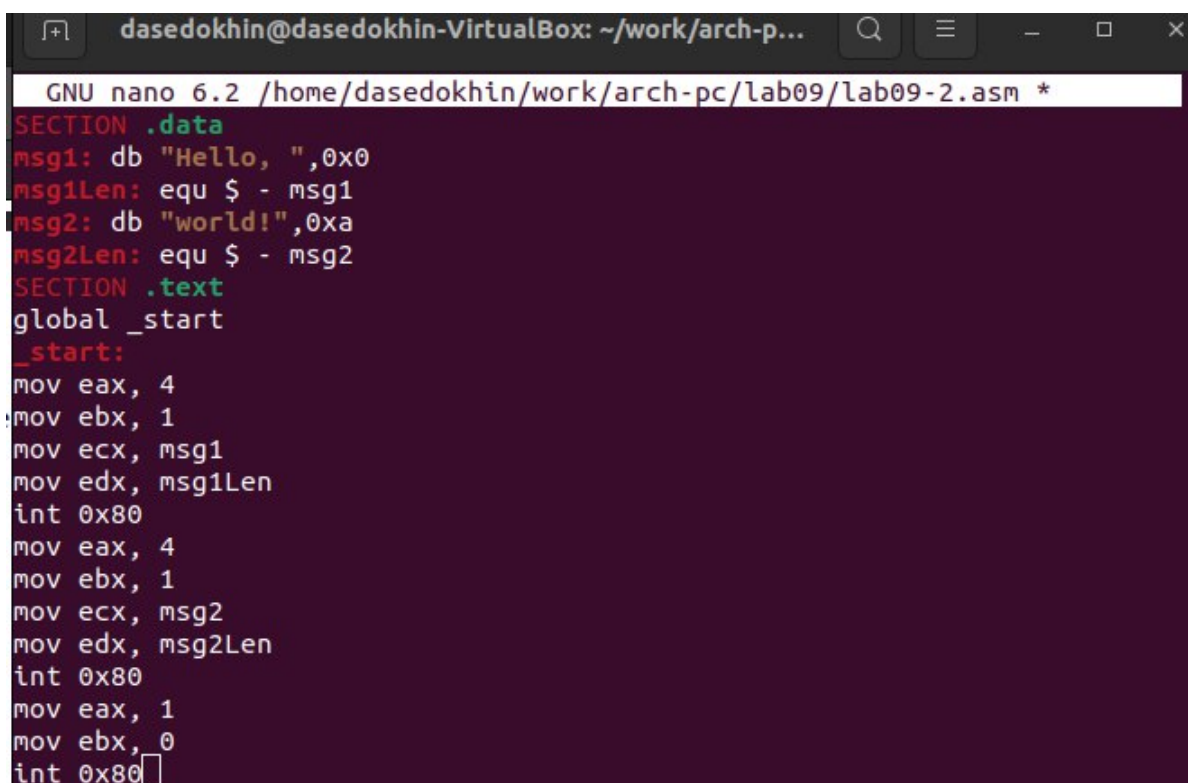
Рис. 2.5: Создание и проверка исполняемого файла

Создадим файл lab09-2.asm с текстом программы из Листинга 9.2. (Программа печати сообщения Hello world!) (рис. 2.6).

```
dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$ touch lab09-2.asm
dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$ ls
in_out.asm lab09-1 lab09-1.asm lab09-1.o lab09-2.asm
dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$
```

Рис. 2.6: Создание файла lab09-2.asm

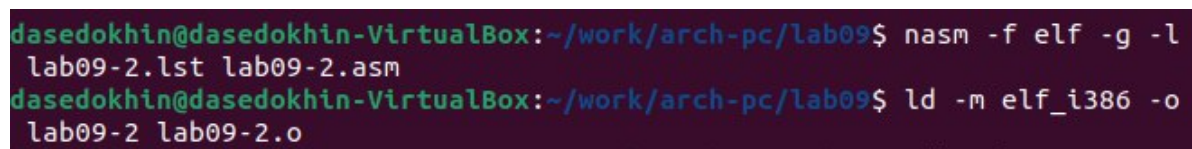
Введём в файл lab09-2.asm текст программы из листинга 9.2. (рис. 2.7).



```
dasedokhin@dasedokhin-VirtualBox: ~/work/arch-p...
GNU nano 6.2 /home/dasedokhin/work/arch-pc/lab09/lab09-2.asm *
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. 2.7: Ввод текст программы из листинга 9.2.

Получим исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом '-g'. (рис. 2.8).



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

Рис. 2.8: Получение исполняемого файла

Загрузим исполняемый файл в отладчик gdb (рис. 2.9).

```

dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
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-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...
(gdb) run

```

Рис. 2.9: Загрузка файла в отладчик gdb

Проверим работу программы, запустив ее в оболочке GDB с помощью команды `run` (со- кращённо `r`): (рис. 2.10).

```

(gdb) run
Starting program: /home/dasedokhin/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 15159) exited normally]

```

Рис. 2.10: Проверка работы программы

Для более подробного анализа программы установим брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустим её. (рис. 2.11).

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

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

Рис. 2.11: Подробный анализ программы \_start

Посмотрим дисассимилированный 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: Дисассимилированный код программы

Переключим на отображение команд с Intel'овским синтаксисом, введя команду `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 отображение команд intel синтаксисом

В АТТ имена регистров начинаются с символа %, а имена операндов с \$, в то время как в intel используется привычный нам синтаксис. Включим режим псевдографики для более удобного анализа программы и выведем значения регистров (рис. 2.14 2.15).



```
dasedokhin@dasedokhin-VirtualBox: ~/work/arch-p...
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
0x804902c <_start+44> mov    eax,0x1
0x8049031 <_start+49> mov    ebx,0x0
0x8049036 <_start+54> int    0x80
0x8049038 add    BYTE PTR [eax],al
0x804903a add    BYTE PTR [eax],al
0x804903c add    BYTE PTR [eax],al
0x804903e add    BYTE PTR [eax],al
0x8049040 add    BYTE PTR [eax],al
0x8049042 add    BYTE PTR [eax],al
0x8049044 add    BYTE PTR [eax],al
0x8049046 add    BYTE PTR [eax],al
0x8049048 add    BYTE PTR [eax],al
0x804904a add    BYTE PTR [eax],al
0x804904c add    BYTE PTR [eax],al
0x804904e add    BYTE PTR [eax],al
0x8049050 add    BYTE PTR [eax],al
0x8049052 add    BYTE PTR [eax],al
native process 15183 In: _start L9 PC: 0x8049000
(gdb)
```

Рис. 2.14: Режим псевдографики

```
dasedokhin@dasedokhin-VirtualBox: ~/work/arch-p...
[ Register Values Unavailable ]

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
0x804902c <_start+44> mov eax,0x1
0x8049031 <_start+49> mov ebx,0x0
0x8049036 <_start+54> int 0x80
0x8049038 add BYTE PTR [eax],al

native process 15183 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb)
```

Рис. 2.15: layout regs

Установить точку останова можно командой `break` (кратко `b`). Типичный аргумент этой команды — место установки. Его можно задать или как номер строки программы (имеет смысл, если есть исходный файл, а программа компилировалась с информацией об отладке), или как имя метки, или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка»: На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверим это с помощью команды `info breakpoints` (кратко `i b`) (рис. 2.16).

```
dasedokhin@dasedokhin-VirtualBox: ~/work/arch-p...
[ Register Values Unavailable ]

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
      0x804902c <_start+44> mov    eax,0x1
      0x8049031 <_start+49> mov    ebx,0x0
      0x8049036 <_start+54> int     0x80
      0x8049038          add    BYTE PTR [eax],al

native process 15183 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) info breakpoints
Num      Type             Disp Enb Address      What
1        breakpoint       keep y  0x08049000 lab09-2.asm:9
        breakpoint already hit 1 time
(gdb)
```

Рис. 2.16: Проверка установки точки \_start

Установим еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей ин-



струкции Определим адрес предпоследней инструкции (mov ebx,0x0) и установим точку останова. (рис. 2.17).

```
(gdb) break *0x8049000
Note: breakpoints 1 and 2 also set at pc 0x8049000.
Breakpoint 3 at 0x8049000: file lab09-2.asm, line 9.
(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  0x08049000  lab09-2.asm:9
3        breakpoint       keep y  0x08049000  lab09-2.asm:9
(gdb) █
```

Рис. 2.17: Определение адреса

Посмотрим информацию о всех установленных точках останова (рис. 2.18).

```

[ Register Values Unavailable ]

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
0x804902c <_start+44>   mov    eax,0x1
0x8049031 <_start+49>   mov    ebx,0x0
0x8049036 <_start+54>   int     0x80
0x8049038             add    BYTE PTR [eax],al

native process 15183 In: _start L9 PC: 0x8049000
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd230 0xffffd230
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
es       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 2.18: Просмотр информации о всех установленных точках останова

С помощью команды `x/` также можно посмотреть содержимое пере-менной.

Посмотрим значение переменной msg1 и msg2 по имени (рис. 2.19).

```
--Type <RET> for more, q to quit, c to continue
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "world!\n\034"
(gdb) █
```

Рис. 2.19: Просмотр значения msg1 и msg2

Изменить значение для регистра или ячейки памяти можно с помощью команды set, задав ей в качестве аргумента имя регистра или адрес. При этом перед именем регистра ставится префикс \$, а перед адресом нужно указать в фигурных скобках тип данных (размер сохраняемого значения; в качестве типа данных можно использовать типы языка Си). Изменим первый символ переменной msg1 (рис. 2.20).

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

Рис. 2.20: Изменение первого символа переменной msg1

Заменяем любой символ во второй переменной msg2. (рис. 2.21).

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

Рис. 2.21: Изменение первого символа переменной msg2

С помощью команды set изменим значение регистра ebx (рис. 2.22).

```
base@oknm@base@oknm-VirtualBox: ~/work/arch-p...
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x32     50
esp      0xffffd230 0xffffd230
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

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
0x804902c <_start+44> mov    eax,0x1
0x8049031 <_start+49> mov    ebx,0x0
0x8049036 <_start+54> int    0x80
0x8049038 add    BYTE PTR [eax],al

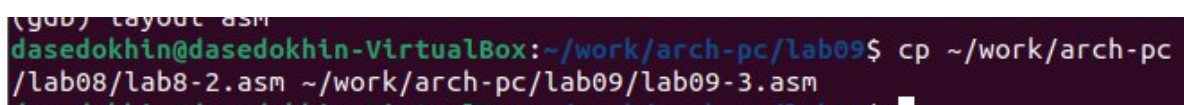
native process 15183 In: _start L9 PC: 0x8049000
(gdb) set {char}&msg1='h'
(gdb) set {char}0x804a001='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hhlllo, "
(gdb) set {char}&msg2='s'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "sorld!\n\034"
(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) p/x $edx
$2 = 0x0
(gdb) p/t $edx
$3 = 0
(gdb)
```

Рис. 2.22: Изменение значение регистра ebx

Разница вывода команд `p/s $ebx` отличается тем, что в первом случае мы переводим символ в его строковый вид, а во втором случае число в строковом виде не изменяется.

Завершим выполнение программы с помощью команды `continue` (сокращенно `c`) или `stepi` (сокращенно `si`) и выйдем из GDB с помощью команды `quit` (сокращенно `q`).

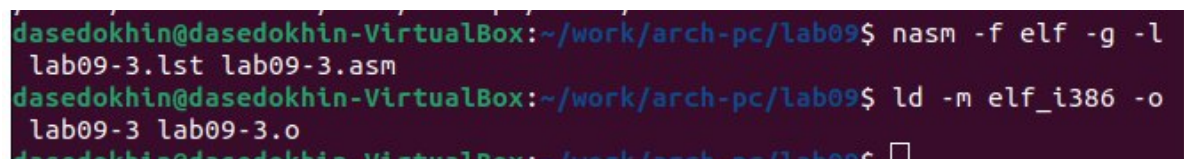
Скопируем файл `lab8-2.asm`, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем `lab09-3.asm` (рис. 2.23).



```
(gdb) layout asm
dasedokhin@dasedokhin-VirtualBox: ~/work/arch-pc/lab09$ cp ~/work/arch-pc/
/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
```

Рис. 2.23: Копирование файла `lab8-2.asm` в файл `lab09-3.asm`

Создадим исполняемый файл (рис. 2.24).



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

Рис. 2.24: Создание исполняемого файла

Для загрузки в `gdb` программы с аргументами необходимо использовать ключ `-args`. Загрузим исполняемый файл в отладчик, указав аргументы (рис. 2.25).



```

dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$ gdb --args lab09-
3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
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-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 lab09-3...
(gdb)

```

Рис. 2.25: Загрузка исполняемого файла в отладчик

Как отмечалось в предыдущей лабораторной работе, при запуске программы аргументы командной строки загружаются в стек. Исследуем расположение аргументов командной строки в стеке после запуска программы с помощью gdb. Для начала установим точку останова перед первой инструкцией в программе и запустим ее. (рис. 2.26).

```

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

Breakpoint 1, _start () at lab09-3.asm:8
8      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) █

```

Рис. 2.26: Установка точки останова в программе и её запуск

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя

программы) (рис. 2.27).

```
(gdb) x/x $esp
0xffffd1f0: 0x00000005
(gdb) █
```

Рис. 2.27: Адрес вершины стека

Как видно, число аргументов равно 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’. Посмотрим остальные позиции стека – по адресу [esp+4] располагается адрес в памяти где находится имя программы, по адресу [esp+8] храниться адрес первого аргумента, по адресу [esp+12] – второго и т.д. (рис. 2.28).

```
(gdb) x/s *(void**)(esp + 4)
0xffffd3a1: "/home/dasedokhin/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd3cd: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd3df: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd3f0: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd3f2: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

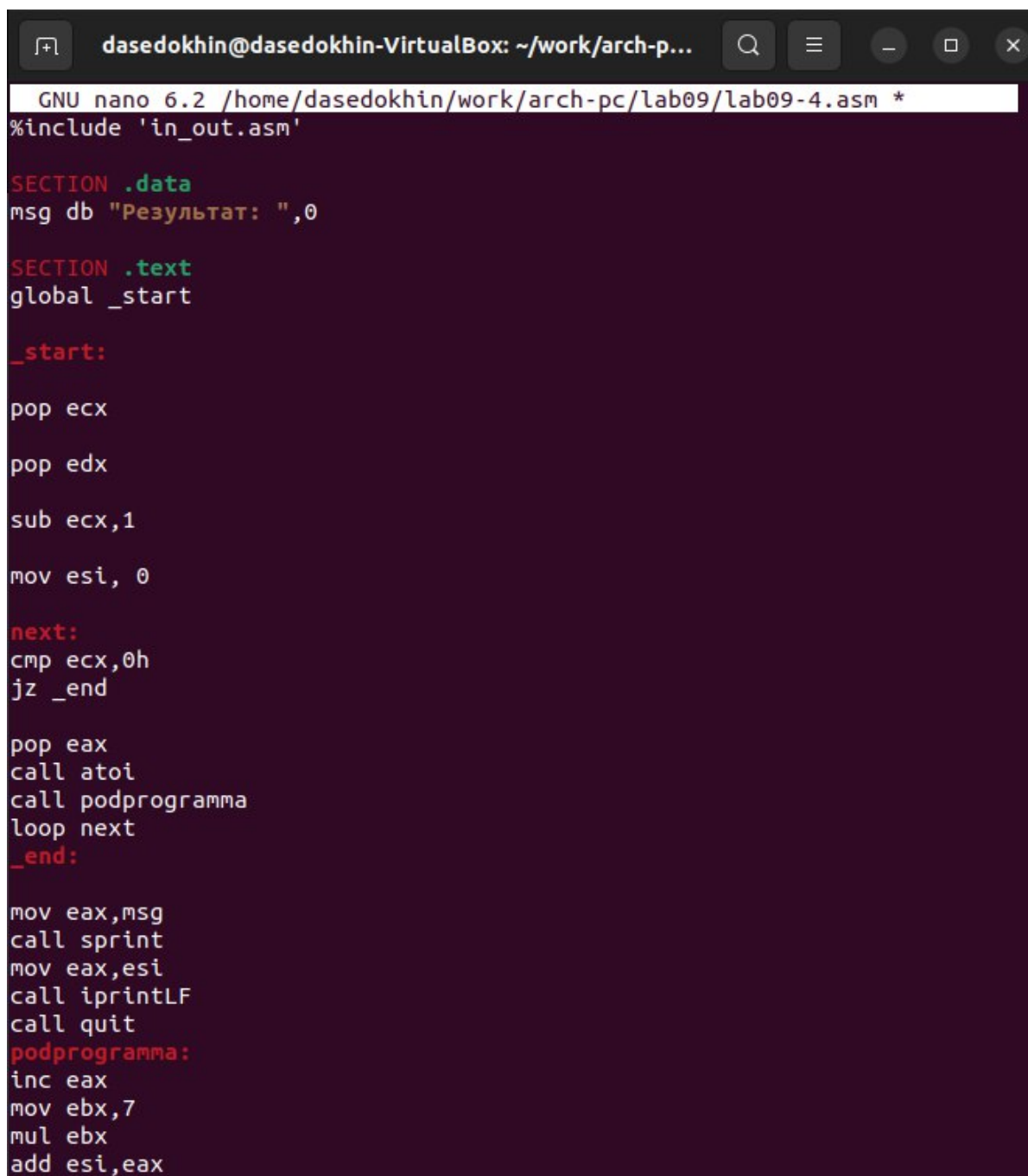
Рис. 2.28: Просмотр позиций стека



### 3 Задание для самостоятельной работы

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

Для этого, сначала создадим файл lab09-4.asm и за основу текста его программы возьмём текст программы из прошлой лабораторной работы файла srab1.asm и отредактируем текст под нашу задачу (рис. 3.1).



```
GNU nano 6.2 /home/dasedokhin/work/arch-pc/lab09/lab09-4.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 podprogramma
loop next
_end:

mov eax,msg
call sprint
mov eax,esi
call iprintLF
call quit
podprogramma:
inc eax
mov ebx,7
mul ebx
add esi,eax
```

Рис. 3.1: Текст файла lab09-4.asm

Создадим исполняемый файл и проверим его работу (рис. 3.2).

```

dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$ ./lab09-4 1 2 3 4
Результат: 98
dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$

```

Рис. 3.2: Создание исполняемого файла и его проверка

Создадим файл lab09-5.asm и введём в него экст программы из листинга 9.3 (рис. 3.3).

```

mc [dasedokhin@dasedokhin-VirtualBox]:~/work/ar...
GNU nano 6.2 /home/dasedokhin/work/arch-pc/lab09/lab09-5.asm *
#include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit

```

Рис. 3.3: Текст файла lab09-5.asm

В листинге 9.3 приведена программа вычисления выражения  $(3 + 2) \times 4 + 5$ . При запуске данная программа дает неверный результат. Проверим, создав исполняемый файл. (рис. 3.4).

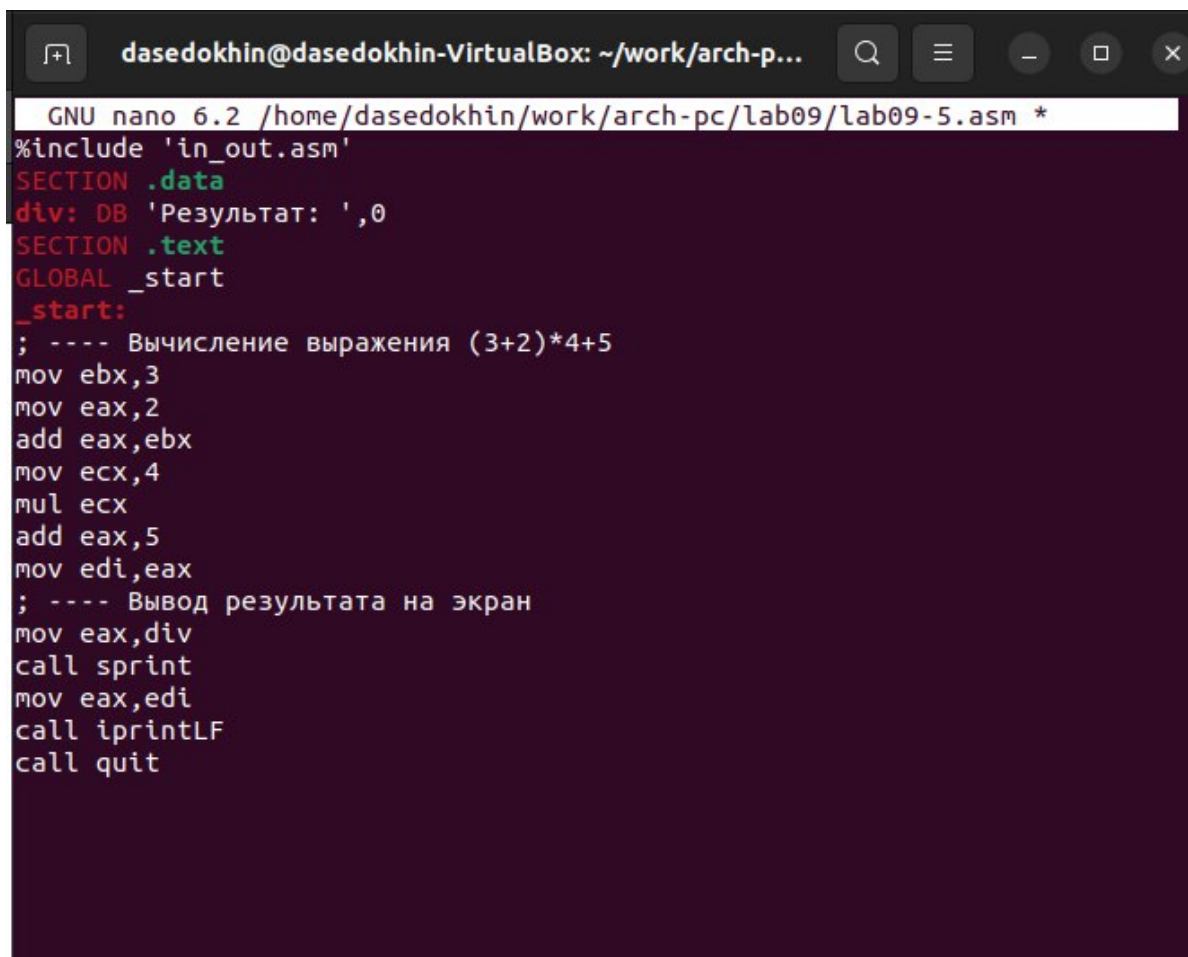
```

dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab09-5.o
dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$ ./lab09-5
Результат: 10
dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$

```

Рис. 3.4: Создание и проверка исполняемого файла

Получим исполняемый файл через `grd`. Проверим с помощью команды `continue` каждую точку останова и значения регистров. На основе этого внесём изменения в программу (рис. 3.5).



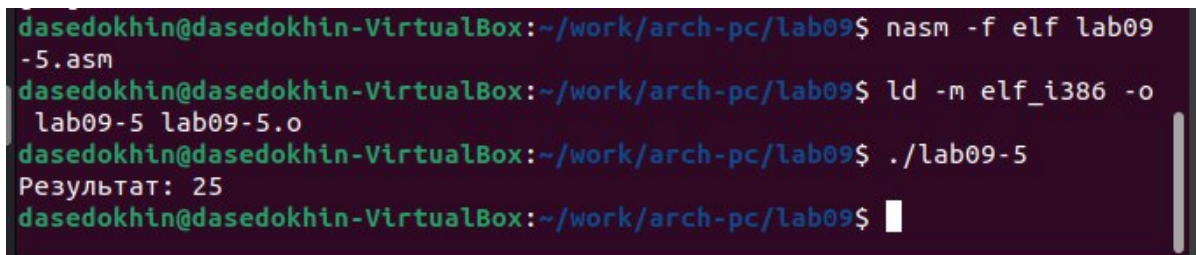
```

GNU nano 6.2 /home/dasedokhin/work/arch-pc/lab09/lab09-5.asm *
%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit

```

Рис. 3.5: Правильный код программы lab09-5.asm

Создадим исполняемый файл и проверим его работу (рис. 3.6).

A terminal window with a dark purple background and green text. The prompt is 'dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09\$'. The first command is 'nasm -f elf lab09-5.asm', followed by 'ld -m elf\_i386 -o lab09-5 lab09-5.o'. The second command is './lab09-5', which outputs 'Результат: 25'. The prompt is then shown again with a cursor.

```
dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab09-5.o
dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$ ./lab09-5
Результат: 25
dasedokhin@dasedokhin-VirtualBox:~/work/arch-pc/lab09$
```

Рис. 3.6: Создание исполняемого файла и его проверка

## 4 Выводы

Я приобрёл навыки написания программ с использованием подпрограмм и ознакомился с методами отладки при помощи GDB и его основными возможностями.