

# **Отчёт по лабораторной работе №8**

**Дисциплина: Архитектура компьютера**

Осина Виктория Александровна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>6</b>
<b>2</b>	<b>Задание</b>	<b>7</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>8</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>11</b>
4.1	Реализация подпрограмм в NASM . . . . .	11
4.2	Отладка программ с помощью GDB . . . . .	14
4.3	Обработка аргументов командной строки в GDB . . . . .	22
4.4	Выполнение задания для самостоятельной работы . . . . .	24
<b>5</b>	<b>Выводы</b>	<b>32</b>
<b>6</b>	<b>Список литературы</b>	<b>33</b>

## Список иллюстраций

4.1	Создание каталога lab09 и файла lab9-1.asm в нем . . . . .	11
4.2	Копирование файла in_out.asm . . . . .	11
4.3	Проверка, что файл находится в нужном каталоге . . . . .	12
4.4	Открытие файла в редакторе . . . . .	12
4.5	Ввод текста программы в файл . . . . .	12
4.6	Создание исполняемого файла и его запуск . . . . .	13
4.7	Изменение текста программы . . . . .	13
4.8	Создание исполняемого файла и его запуск . . . . .	14
4.9	Создание файла lab9-2.asm и его открытие в редакторе . . . . .	14
4.10	Ввод текста программы . . . . .	14
4.11	Создание исполняемого файла и его запуск . . . . .	15
4.12	Загрузка исполняемого файла в отладчик gdb . . . . .	15
4.13	Проверка работы программы . . . . .	15
4.14	Установка брейкпоинта . . . . .	15
4.15	Просмотр дисассимилированного кода программы . . . . .	16
4.16	Переключение на режим Intel . . . . .	16
4.17	Просмотр дисассимилированного кода программы . . . . .	17
4.18	Включение режима псевдографики . . . . .	17
4.19	Проверка наличия брейкпоинта . . . . .	18
4.20	Установка еще одной точки останова . . . . .	18
4.21	Просмотр содержимого регистров . . . . .	18
4.22	Выполнение 5 инструкций stepi . . . . .	19
4.23	Просмотр содержимого регистров . . . . .	19
4.24	Просмотр значения переменной msg1 . . . . .	19
4.25	Просмотр значения переменной msg2 . . . . .	20
4.26	Изменение символа . . . . .	20
4.27	Изменение символа . . . . .	20
4.28	Вывод в различных форматах . . . . .	20
4.29	Изменение значения регистра . . . . .	21
4.30	Завершение программы . . . . .	21
4.31	Выход из gdb . . . . .	22
4.32	Копирование файла . . . . .	22
4.33	Создание исполняемого файла . . . . .	22
4.34	Загрузка исполняемого файла в отладчик . . . . .	23
4.35	Установка точки останова . . . . .	23
4.36	Просмотр содержимого регистра . . . . .	23
4.37	Просмотр остальных позиций стека . . . . .	24

4.38 Преобразование программы . . . . .	25
4.39 Создание исполняемого файла и его запуск . . . . .	25
4.40 Создание файла var7-2.asm и открытие его в редакторе . . . . .	26
4.41 Ввод текста программы . . . . .	27
4.42 Создание исполняемого файла и его запуск . . . . .	27
4.43 Получение исполняемого файла . . . . .	28
4.44 Запуск программы и установка брейкпоинта . . . . .	28
4.45 Просмотр изменения регистра . . . . .	29
4.46 Просмотр изменения регистра . . . . .	29
4.47 Изменение текста программы . . . . .	30
4.48 Создание исполняемого файла и его запуск . . . . .	30

## Список таблиц

# 1 Цель работы

Целью данной лабораторной работы является 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. Обработка аргументов командной строки в GDB
4. Выполнение задания для самостоятельной работы

## 3 Теоретическое введение

### Понятие об отладке

Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа: \* обнаружение ошибки; \* поиск её местонахождения; \* определение причины ошибки; \* исправление ошибки.

Можно выделить следующие типы ошибок: \* синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка; \* семантические ошибки — являются логическими и приводят к тому, что программа запускается, отработывает, но не даёт желаемого результата; \* ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают прерывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль).

Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга.

Третий этап — выяснение причины ошибки. После определения местонахождения ошибки обычно проще определить причину неправильной работы программы.

Последний этап — исправление ошибки. После этого при повторном запуске программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново.

### Методы отладки



Наиболее часто применяют следующие методы отладки: \* создание точек контроля значений на входе и выходе участка программы (например, вывод промежуточных значений на экран — так называемые диагностические сообщения); \* использование специальных программ-отладчиков. Отладчики позволяют управлять ходом выполнения программы, контролировать и изменять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправление. Наиболее популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам.

Пошаговое выполнение — это выполнение программы с остановкой после каждой строчки, чтобы программист мог проверить значения переменных и выполнить другие действия.

Точки останова — это специально отмеченные места в программе, в которых программа-отладчик приостанавливает выполнение программы и ждёт команд. Наиболее популярные виды точек останова: \* Breakpoint — точка останова (остановка происходит, когда выполнение доходит до определённой строки, адреса или процедуры, отмеченной программистом); \* Watchpoint — точка просмотра (выполнение программы приостанавливается, если программа обратилась к определённой переменной: либо считала её значение, либо изменила его).

Точки останова устанавливаются в отладчике на время сеанса работы с кодом программы, т.е. они сохраняются до выхода из программы-отладчика или до смены отлаживаемой программы.

### **Запуск отладчика GDB**

Команда run (сокращённо r) — запускает отлаживаемую программу в оболочке GDB. Если точки останова не были установлены, то программа выполняется и выводятся сообщения:

```
(gdb) run
Starting program: test
Program exited normally.
(gdb)
```

Для выхода из отладчика используется команда `quit` (или сокращённо `q`): `(gdb)`

`q`

### **Дизассемблирование программы**

Посмотреть дизассемблированный код программы можно с помощью команды `disassemble`: `(gdb) disassemble _start`

### **Точки останова**

Установить точку останова можно командой `break` (кратко `b`). Типичный аргумент этой команды — место установки. Его можно задать как имя метки или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка»: `(gdb) break *<адрес>` `(gdb) b <метка>` Информацию о всех установленных точках останова можно вывести командой `info` (кратко `i`): `(gdb) info breakpoints` `(gdb) i b`

Если же точка останова в дальнейшем больше не нужна, она может быть удалена с помощью команды `delete`: `(gdb) delete breakpoint <номер точки останова>`

### **Пошаговая отладка**

Команда `stepi` (кратко `si`) позволяет выполнять программу по шагам, т.е. данная команда выполняет ровно одну инструкцию `(gdb) si [аргумент]`

При указании в качестве аргумента целого числа  $\times$  отладчик выполнит команду `step`  $\times$  раз при условии, что не будет точек останова или выполнение программы не прервётся по другим причинам.

Команда `nexti` (или `ni`) аналогична `stepi`, но вызов процедуры (функции) трактуется отладчиком как одна инструкция: `(gdb) ni [аргумент]`

## 4 Выполнение лабораторной работы

### 4.1 Реализация подпрограмм в NASM

Создаю каталог lab09 для программ лабораторной работы №9, перехожу в него и создаю файл lab9-1.asm, проверяю, что файл создан (рис. 4.1)

```
[vaosina@fedora ~]$ mkdir ~/work/arch-pc/lab09
[vaosina@fedora ~]$ cd ~/work/arch-pc/lab09
[vaosina@fedora lab09]$ touch lab09-1.asm
```

Рис. 4.1: Создание каталога lab09 и файла lab9-1.asm в нем

Перед работой с программами копирую файл in\_out.asm в каталог и проверяю, что файл находится в нужном каталоге (рис. 4.2) (рис. 4.3)

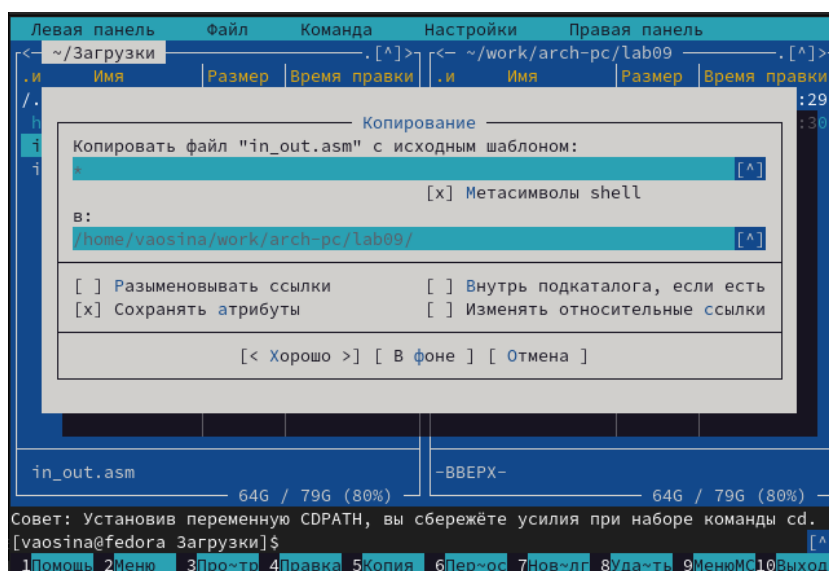


Рис. 4.2: Копирование файла in\_out.asm

Настройки		Правая панель	
← ~/work/arch-pc/lab09		. [^]>	
.и	Имя	Размер	Время правки
/..		-ВВЕРХ-	дек 5 12:29
in_out.asm		3942	ноя 11 14:25
lab09-1.asm		0	дек 5 12:30

Рис. 4.3: Проверка, что файл находится в нужном каталоге

Открываю lab9-1.asm в редакторе и ввожу в него текст программы вычисления арифметического выражения  $f(x) = 2x + 7$  (рис. 4.4) и (рис. 4.5).

```
[vaosina@fedora lab09]$ gedit lab09-1.asm
```

Рис. 4.4: Открытие файла в редакторе

```

Открыть  *lab09-1.asm
~/work/arch-pc/lab09
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 res: RESB 80
8 SECTION .text
9 GLOBAL _start
10 _start:
11
12 ;-----
13 ; Основная программа
14 ;-----
15 mov eax, msg
16 call sprint
17 mov ecx, x
18 mov edx, 80
19 call sread
20 mov eax, x
21 call atoi
22 call _calcul ; Вызов подпрограммы _calcul
23 mov eax, result
24 call sprint
25 mov eax, [res]
26 call iprintf
27 call quit
28
29 ;-----
30 ; Подпрограмма вычисления
31 ; выражения "2x+7"
32 ;-----
33 _calcul:
34 mov ebx, 2
35 mul ebx
36 add eax, 7
37 mov [res], eax
38 ret ; выход из подпрограммы

```

Рис. 4.5: Ввод текста программы в файл

Создаю исполняемый файл и запускаю его (рис. 4.6). Программа работает корректно.

```
[vaosina@fedora lab09]$ nasm -f elf lab09-1.asm
[vaosina@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[vaosina@fedora lab09]$ ./lab09-1
Введите x: 4
2x+7=15
```

Рис. 4.6: Создание исполняемого файла и его запуск

Изменяю текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения  $f(g(x))$ , где  $x$  вводится с клавиатуры,  $f(x) = 2x + 7$ ,  $g(x) = 3x - 1$ . (рис. 4.7)

```
29 ;-----
30 ; Подпрограмма вычисления
31 ; выражения f(g(x)), где
32 ; f(x) = 2x+7
33 ;-----
34 _calcul:
35 call _subcalcul
36 mov ebx,2
37 mul ebx
38 add eax,7
39 mov [res],eax
40 ret ; выход из подпрограммы
41
42 ;-----
43 ; Подпрограмма вычисления
44 ; выражения g(x)=3x-1
45 ;-----
46 _subcalcul:
47 mov ebx, 3
48 mul ebx
49 add eax, -1
50 ret
```

Рис. 4.7: Изменение текста программы

Создаю исполняемый файл и запускаю его. (рис. 4.8). Проверяю работу программы на нескольких значениях  $x$ . Программа работает корректно.

```
[vaosina@fedora lab09]$ gedit lab09-1.asm
[vaosina@fedora lab09]$ nasm -f elf lab09-1.asm
[vaosina@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[vaosina@fedora lab09]$ ./lab09-1
Введите x: 4
2(3x-1)+7=29
[vaosina@fedora lab09]$ ./lab09-1
Введите x: 1
2(3x-1)+7=11
```

Рис. 4.8: Создание исполняемого файла и его запуск

## 4.2 Отладка программ с помощью GDB

Создаю файл lab9-2.asm в каталоге ~/work/arch-pc/lab09 и открываю его в редакторе (рис. 4.9).

```
[vaosina@fedora lab09]$ touch lab09-2.asm
[vaosina@fedora lab09]$ gedit lab09-2.asm
```

Рис. 4.9: Создание файла lab9-2.asm и его открытие в редакторе

Ввожу в него текст программы печати сообщения Hello world!. (рис. 4.10)

```
Открыть [icon] *lab09-2.asm
~/work/arch-pc/lab09

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
17 mov eax, 4
18 mov ebx, 1
19 mov ecx, msg2
20 mov edx, msg2Len
21 int 0x80
22
23 mov eax, 1
24 mov ebx, 0
25 int 0x80
```

Рис. 4.10: Ввод текста программы

Получаю исполняемый файл с добавлением отладочной информации, для этого трансляцию программы провожу с ключом -g. (рис. 4.11).

```
[vaosina@fedora lab09]$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
[vaosina@fedora lab09]$ ld -m elf_i386 -o lab09-2 lab09-2.o
```

Рис. 4.11: Создание исполняемого файла и его запуск

Загружаю исполняемый файл в отладчик gdb. (рис. 4.12).

```
[vaosina@fedora lab09]$ gdb lab09-2
GNU gdb (GDB) Fedora Linux 13.1-2.fc38
Copyright (C) 2023 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-redhat-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)
```

Рис. 4.12: Загрузка исполняемого файла в отладчик gdb

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run.(рис. 4.13).

```
(gdb) run
Starting program: /home/vaosina/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 57502) exited normally]
(gdb)
```

Рис. 4.13: Проверка работы программы

Для более подробного анализа программы устанавливаю брейкпоинт на метку \_start и запускаю программу. (рис. 4.14).

```
(gdb) break _start
Breakpoint 1 at 0x4010e0: file lab09-2.asm, line 11.
(gdb) r
Starting program: /home/vaosina/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:11
11      mov eax, 4
(gdb)
```

Рис. 4.14: Установка брейкпоинта

Смотрю дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start`. (рис. 4.15)

```
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x004010e0 <+0>:    mov     $0x4,%eax
      0x004010e5 <+5>:    mov     $0x1,%ebx
      0x004010ea <+10>:   mov     $0x402118,%ecx
      0x004010ef <+15>:   mov     $0x8,%edx
      0x004010f4 <+20>:   int     $0x80
      0x004010f6 <+22>:   mov     $0x4,%eax
      0x004010fb <+27>:   mov     $0x1,%ebx
      0x00401100 <+32>:   mov     $0x402120,%ecx
      0x00401105 <+37>:   mov     $0x7,%edx
      0x0040110a <+42>:   int     $0x80
      0x0040110c <+44>:   mov     $0x1,%eax
      0x00401111 <+49>:   mov     $0x0,%ebx
      0x00401116 <+54>:   int     $0x80
End of assembler dump.
(gdb) █
```

Рис. 4.15: Просмотр дисассимилированного кода программы

Переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`, и смотрю дисассимилированный код. (рис. 4.16) и (рис. 4.17)

Различия отображения синтаксиса машинных команд в режимах АТТ и Intel состоят в том, что в режиме АТТ перед названиями регистров стоит `$`, а перед операндами `%`, а еще после переключения на режим Intel, регистры и операнды поменялись местами.

```
(gdb) set disassembly-flavor intel
(gdb) █
```

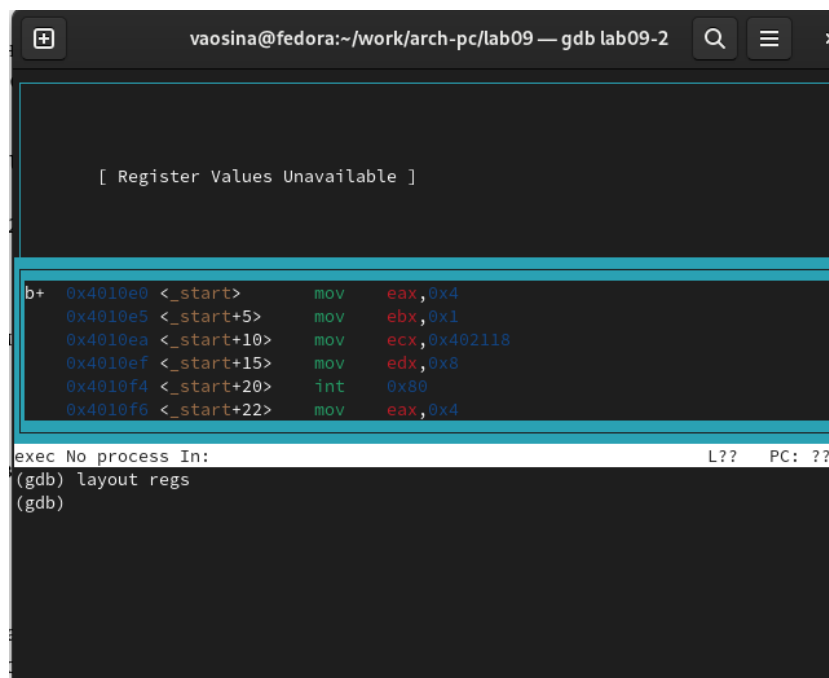
Рис. 4.16: Переключение на режим Intel



```
(gdb) disassemble _start
Dump of assembler code for function _start:
0x004010e0 <+0>:    mov     eax,0x4
0x004010e5 <+5>:    mov     ebx,0x1
0x004010ea <+10>:   mov     ecx,0x402118
0x004010ef <+15>:   mov     edx,0x8
0x004010f4 <+20>:   int     0x80
0x004010f6 <+22>:   mov     eax,0x4
0x004010fb <+27>:   mov     ebx,0x1
0x00401100 <+32>:   mov     ecx,0x402120
0x00401105 <+37>:   mov     edx,0x7
0x0040110a <+42>:   int     0x80
0x0040110c <+44>:   mov     eax,0x1
0x00401111 <+49>:   mov     ebx,0x0
0x00401116 <+54>:   int     0x80
End of assembler dump.
(gdb)
```

Рис. 4.17: Просмотр дисассимилированного кода программы

Включаю режим псевдографики для более удобного анализа программы. (рис. 4.18)



```
vaosina@fedora:~/work/arch-pc/lab09 — gdb lab09-2
[ Register Values Unavailable ]
b+ 0x4010e0 <_start>    mov     eax,0x4
0x4010e5 <_start+5>    mov     ebx,0x1
0x4010ea <_start+10>   mov     ecx,0x402118
0x4010ef <_start+15>   mov     edx,0x8
0x4010f4 <_start+20>   int     0x80
0x4010f6 <_start+22>   mov     eax,0x4
exec No process In: L?? PC: ??
(gdb) layout regs
(gdb)
```

Рис. 4.18: Включение режима псевдографики

На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверяю это с помощью команды `info breakpoints` (кратко `i b`). (рис. 4.19)

```
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x004010e0 lab09-2.asm:11
          breakpoint already hit 1 time
(gdb)
```

Рис. 4.19: Проверка наличия брейкпоинта

Устанавливаю еще одну точку останова по адресу инструкции `mov ebx,0x0` и Смотрю информацию о всех установленных точках останова. (рис. 4.20)

```
0x40110a <_start+42>  int    0x80
0x40110c <_start+44>  mov     eax,0x1
b+ 0x401111 <_start+49>  mov     ebx,0x0
0x401116 <_start+54>  int    0x80
0x401118             dec     eax
0x401119             gs ins BYTE PTR es:[edi],dx

native process 59226 In: _start L11 PC: 0x4010e0
(gdb) b *0x401111
Breakpoint 2 at 0x401111: file lab09-2.asm, line 24.
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x004010e0 lab09-2.asm:11
          breakpoint already hit 1 time
2        breakpoint keep y  0x00401111 lab09-2.asm:24
(gdb)
```

Рис. 4.20: Установка еще одной точки останова

Смотрю содержимое регистров с помощью команды `info registers` (или `i r`). (рис. 4.21)

```
native process 59226 In: _start L11 PC: 0x4010e0
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 4.21: Просмотр содержимого регистров

Выполняю 5 инструкций с помощью команды `stepi` (или `si`). (рис. 4.22) и (рис. 4.23)

Изменились значения регистров `eax`, `ecx`, `edx`, `ebx`.

```

vaosina@fedora:~/work/arch-pc/lab09 — gdb lab09-2
Register group: general
eax      0x8      8
ecx      0x402118  4202776
edx      0x8      8
ebx      0x1      1
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0

0x4010ea <_start+10> mov ecx,0x402118
0x4010ef <_start+15> mov edx,0x8
0x4010f4 <_start+20> int 0x80
> 0x4010f6 <_start+22> mov eax,0x4
0x4010fb <_start+27> mov ebx,0x1
0x401100 <_start+32> mov ecx,0x402120

native process 59226 In: _start L17 PC: 0x4010f6
cs      0x23      35
ss      0x2b      43
ds      0x2b      43
es      0x2b      43
fs      0x0      0
gs      0x0      0
(gdb) si 5
(gdb)

```

Рис. 4.22: Выполнение 5 инструкций stepi

```

native process 59226 In: _start L17 PC: 0x4010f6
eax      0x8      8
ecx      0x402118  4202776
edx      0x8      8
ebx      0x1      1
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 4.23: Просмотр содержимого регистров

Смотрю значение переменной msg1 по имени. (рис. 4.24)

```

native process 59226 In: _start L17 PC: 0x4010f6
ss      0x2b      43
ds      0x2b      43
es      0x2b      43
fs      0x0      0
gs      0x0      0
(gdb) x/1sb &msg1
0x402118 <msg1>: "Hello, "
(gdb)

```

Рис. 4.24: Просмотр значения переменной msg1

Смотрю значение переменной msg2 по адресу, который определяю по ди-

засемблированной инструкции. (на скрине я сначала случайно ввела не тот адрес)(рис. 4.25)

```
> 0x4010f6 <_start+22> mov    eax,0x4
0x4010fb <_start+27> mov    ebx,0x1
0x401100 <_start+32> mov    ecx,0x402120

native process 59226 In: _start L17 PC: 0x4010f6
gs      0x0      0
(gdb) x/1sb &msg1
0x402118 <msg1>:      "Hello, "
(gdb) x/1sb 0x4010f6
0x4010f6 <_start+22>: "\270\004"
(gdb) x/1sb 0x402120
0x402120 <msg2>:      "world!\n\034"
(gdb)
```

Рис. 4.25: Просмотр значения переменной msg2

Изменяю первый символ переменной msg1. (рис. 4.26)

```
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x402118 <msg1>:      "hello, "
(gdb)
```

Рис. 4.26: Изменение символа

Изменяю символ переменной msg2. (рис. 4.27)

```
(gdb) set {char}&msg2='o'
(gdb) x/1sb &msg2
0x402120 <msg2>:      "oorld!\n\034"
(gdb)
```

Рис. 4.27: Изменение символа

Вывожу в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx. (рис. 4.28)

```
(gdb) p/x $edx
$1 = 0x7
(gdb) p/t $edx
$2 = 111
(gdb) p/c $edx
$3 = 7 '\a'
(gdb)
```

Рис. 4.28: Вывод в различных форматах

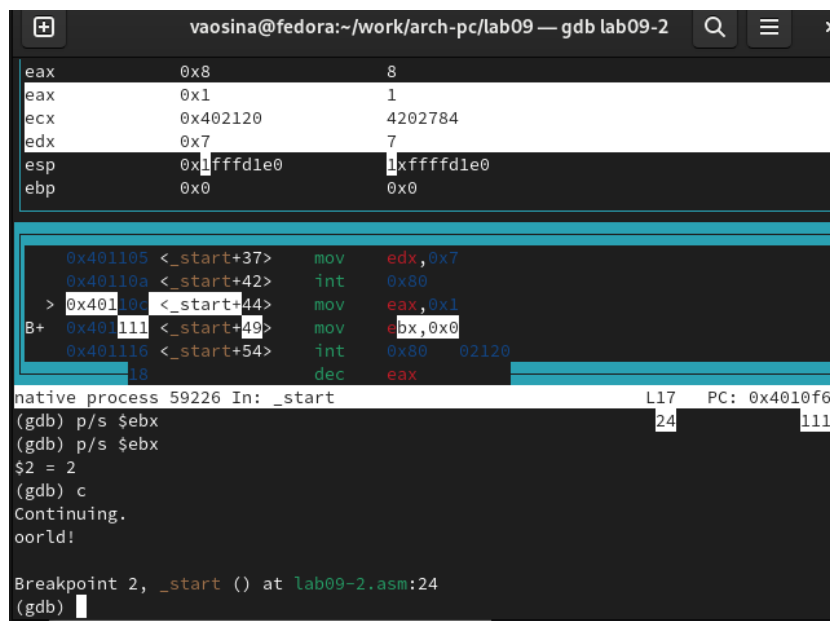
С помощью команды `set` изменяю значение регистра `ebx`. (рис. 4.29)

Разница вывода состоит в том, что в первом случае символ переводится в строковый вид.

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$2 = 2
(gdb)
```

Рис. 4.29: Изменение значения регистра

Завершаю выполнение программы с помощью команды `continue` (сокращенно `c`) и выхожу из GDB с помощью команды `quit` (сокращенно `q`). (рис. 4.30) и (рис. 4.31)



```
vaosina@fedora:~/work/arch-pc/lab09 — gdb lab09-2
eax      0x8      8
ecx      0x402120 4202784
edx      0x7      7
esp      0xffffd1e0 1fffffd1e0
ebp      0x0      0x0

0x401105 <_start+37>  mov     edx,0x7
0x40110a <_start+42>  int     0x80
> 0x40110e <_start+44>  mov     eax,0x1
B+ 0x401111 <_start+49>  mov     ebx,0x0
0x40111b <_start+54>  int     0x80  02120
0x401118      dec     eax

native process 59226 In: _start L17 PC: 0x4010f6
(gdb) p/s $ebx
24 111
(gdb) p/s $ebx
$2 = 2
(gdb) c
Continuing.
oorld!

Breakpoint 2, _start () at lab09-2.asm:24
(gdb)
```

Рис. 4.30: Завершение программы

```
(gdb) q
A debugging session is active.

    Inferior 1 [process 59226] will be killed.

Quit anyway? (y or n)
```

Рис. 4.31: Выход из gdb

## 4.3 Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки в файл с именем lab09-3.asm. (рис. 4.32)

```
[vaosina@fedora lab09]$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/
lab09-3.asm
[vaosina@fedora lab09]$ ls
in_out.asm lab09-1.asm lab09-2 lab09-2.lst lab09-3.asm
lab09-1 lab09-1.o lab09-2.asm lab09-2.o
[vaosina@fedora lab09]$
```

Рис. 4.32: Копирование файла

Создаю исполняемый файл (рис. 4.33)

```
[vaosina@fedora lab09]$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
[vaosina@fedora lab09]$ ld -m elf_i386 -o lab09-3 lab09-3.o
```

Рис. 4.33: Создание исполняемого файла

Загружаю исполняемый файл в отладчик, указав аргументы. (рис. 4.34)

```
[vaosina@fedora lab09]$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (GDB) Fedora Linux 13.1-2.fc38
Copyright (C) 2023 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-redhat-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. 4.34: Загрузка исполняемого файла в отладчик

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее. (рис. 4.35)

```
(gdb) b _start
Breakpoint 1 at 0x4011a8: file lab09-3.asm, line 6.
(gdb) run
Starting program: /home/vaosina/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2
аргумент\ 3

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.fedoraproject.org/>
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.

Breakpoint 1, _start () at lab09-3.asm:6
6      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb)
```

Рис. 4.35: Установка точки останова

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы). (рис. 4.36)

Как видно, число аргументов равно 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’.

```
0      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xfffffd180: 0x00000005
(gdb)
```

Рис. 4.36: Просмотр содержимого регистра

Смотрю 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. 4.37)

Шаг изменения адреса равен 4, т.к. 4 - это отводимый размер памяти на ячейку и мы смотрим содержимое ячеек.

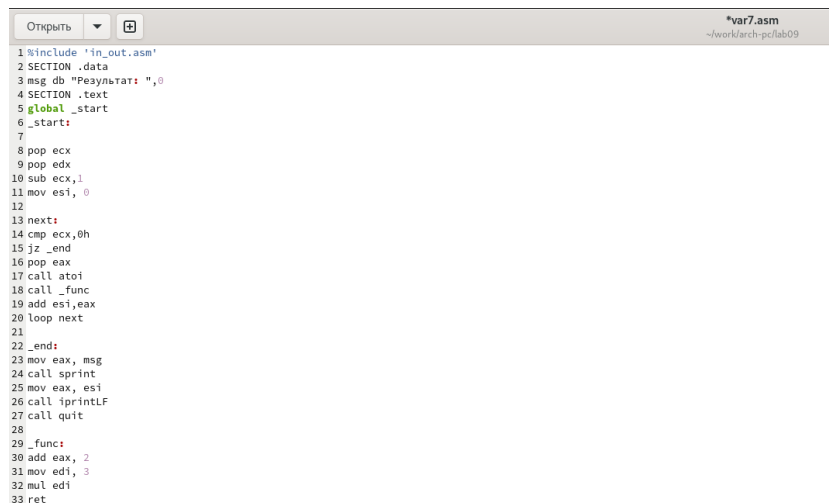
```
(gdb) x/s *(void**)(esp+4)
0xffffd32d:  "/home/vaosina/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp+8)
0xffffd358:  "аргумент1"
(gdb) x/s *(void**)(esp+12)
0xffffd368:  "аргумент"
(gdb) x/s *(void**)(esp+16)
0xffffd378:  "2"
(gdb) x/s *(void**)(esp+20)
0xffffd37b:  "аргумент 3"
(gdb) x/s *(void**)(esp+24)
0x0: <error: Cannot access memory at address 0x0>
(gdb) □
```

Рис. 4.37: Просмотр остальных позиций стека

## 4.4 Выполнение задания для самостоятельной работы

Преобразовываю программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции  $f(x)$  как подпрограмму. (рис. 4.38)






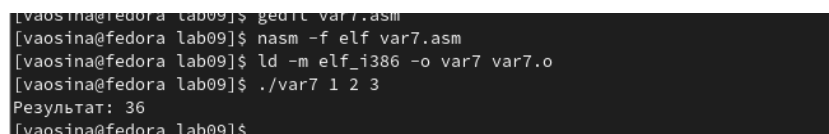
```
Открыть ▼  *var7.asm  
~./work/arch-pc/lab09  
1 %include 'in_out.asm'  
2 SECTION .data  
3 msg db "Результат: ",0  
4 SECTION .text  
5 global _start  
6 _start:  
7  
8 pop ecx  
9 pop edx  
10 sub ecx,1  
11 mov esi, 0  
12  
13 next:  
14 cmp ecx,0h  
15 jz _end  
16 pop eax  
17 call atoi  
18 call _func  
19 add esi,eax  
20 loop next  
21  
22 _end:  
23 mov eax, msg  
24 call sprint  
25 mov eax, esi  
26 call iprintf  
27 call quit  
28  
29 _func:  
30 add eax, 2  
31 mov edi, 3  
32 mul edi  
33 ret
```

Рис. 4.38: Преобразование программы

Создаю исполняемый файл и запускаю его. (рис. 4.39) Программа работает корректно.



```
[vaosina@fedora lab09]$ gedit var7.asm  
[vaosina@fedora lab09]$ nasm -f elf var7.asm  
[vaosina@fedora lab09]$ ld -m elf_i386 -o var7 var7.o  
[vaosina@fedora lab09]$ ./var7 1 2 3  
Результат: 36  
[vaosina@fedora lab09]$
```

Рис. 4.39: Создание исполняемого файла и его запуск

Текст программы:

```
%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

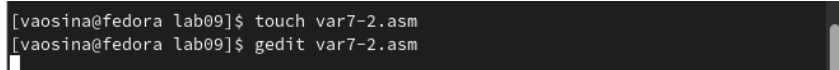
next:
cmp ecx,0h
jz _end
pop eax
call atoi
call _func
add esi,eax
loop next

_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit

_func:
add eax, 2
mov edi, 3
mul edi
ret

```

Создаю файл var7-2.asm и открываю его в редакторе. (рис. 4.40)



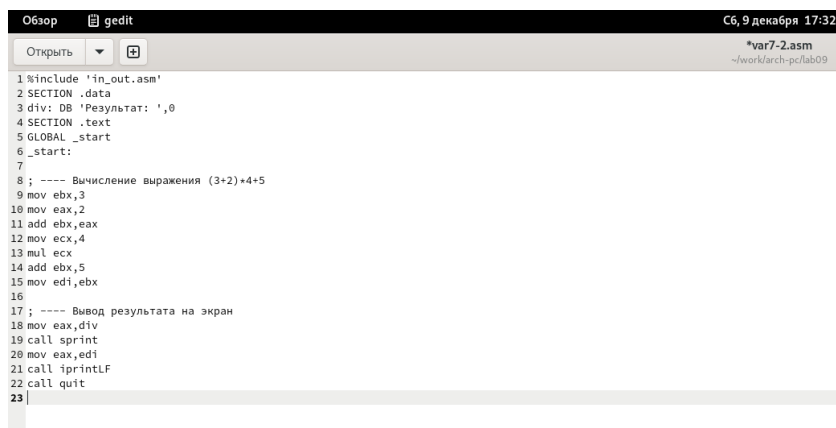
```

[vaosina@fedora lab09]$ touch var7-2.asm
[vaosina@fedora lab09]$ gedit var7-2.asm

```

Рис. 4.40: Создание файла var7-2.asm и открытие его в редакторе

Ввожу текст программы вычисления выражения  $(3+2)*4+5$  из листинга 9.3. (рис. 4.41)

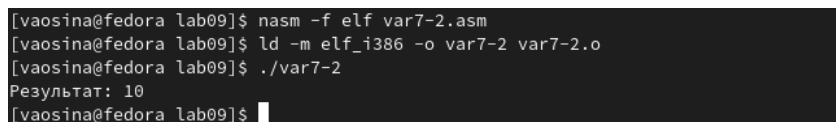


```
Обзор gedit Сб, 9 декабря 17:32
Открыть *var7-2.asm ~/work/arch-pc/lab09

1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7
8 ; ---- Вычисление выражения (3+2)*4+5
9 mov ebx,3
10 mov eax,2
11 add ebx,eax
12 mov ecx,4
13 mul ecx
14 add ebx,5
15 mov edi,ebx
16
17 ; ---- Вывод результата на экран
18 mov eax,div
19 call sprint
20 mov eax,edi
21 call iprintLF
22 call quit
23
```

Рис. 4.41: Ввод текста программы

Создаю исполняемый файл и запускаю его. (рис. 4.42) Программа действительно работает некорректно.



```
[vaosina@fedora lab09]$ nasm -f elf var7-2.asm
[vaosina@fedora lab09]$ ld -m elf_i386 -o var7-2 var7-2.o
[vaosina@fedora lab09]$ ./var7-2
Результат: 10
[vaosina@fedora lab09]$
```

Рис. 4.42: Создание исполняемого файла и его запуск

Получаю исполняемый файл с добавлением отладочной информации, для этого трансляцию программы провожу с ключом -g и загружаю исполняемый файл в отладчик gdb. (рис. 4.43)

```
[vaosina@fedora lab09]$ nasm -f elf -g -l var7-2.lst var7-2.asm
[vaosina@fedora lab09]$ ld -m elf_i386 -o var7-2 var7-2.o
[vaosina@fedora lab09]$ gdb var7-2
GNU gdb (GDB) Fedora Linux 13.1-2.fc38
Copyright (C) 2023 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-redhat-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 var7-2...
(gdb)
```

Рис. 4.43: Получение исполняемого файла

Запускаю программу и устанавливаю точку останова на метке `_start`, чтобы было удобнее анализировать изменения регистров. (рис. 4.44)

```
(gdb) run
Starting program: /home/vaosina/work/arch-pc/lab09/var7-2

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.fedoraproject.org/>
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.
Результат: 10
[Inferior 1 (process 63366) exited normally]
(gdb) break _start
Breakpoint 1 at 0x4011c8: file var7-2.asm, line 9.
(gdb)
```

Рис. 4.44: Запуск программы и установка брейкпоинта

Первую ошибку я замечаю при изменении регистра `eax` на 8, такого быть не должно. Мы должны были получить при умножении 20. (рис. 4.45)

```

vaosina@fedora:~/work/arch-pc/lab09 — gdb var7-2
eax      0x0      0
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd180 0xffffd180
ebp      0x0      0x0

0x4011d2 <_start+10> add    %eax,%ebx
0x4011d4 <_start+12> mov    $0x4,%ecx
0x4011d9 <_start+17> mul    %ecx
> 0x4011db <_start+19> add    $0x5,%ebx
0x4011de <_start+22> mov    %ebx,%edi
0x4011e0 <_start+24> mov    $0x4021f8,%eax
0x4011e6 <_start+29> call   0x4010ef<_sprint>

native process 63405 In: _start L9 PC: 0x4011c8
(gdb) rprocess 63439 In: _start L14 PC: 0x4011db

Breakpoint 1, _start () at var7-2.asm:9
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 4.45: Просмотр изменения регистра

Далее ошибка при последнем действии, т.к. 5 складывается со значением ebx, в котором тоже 5, в итоге получаем 10, в результате и выводится это значение. (рис. 4.46)

```

vaosina@fedora:~/work/arch-pc/lab09 — gdb var7-2
eax      0x0      0
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd180 0xffffd180
ebp      0x0      0x0

0x4011d4 <_start+12> mov    $0x4,%ecx
0x4011d9 <_start+17> mul    %ecx
0x4011db <_start+19> add    $0x5,%ebx
> 0x4011de <_start+22> mov    %ebx,%edi
0x4011e0 <_start+24> mov    $0x4021f8,%eax
0x4011e6 <_start+29> call   0x4010ef<_sprint>

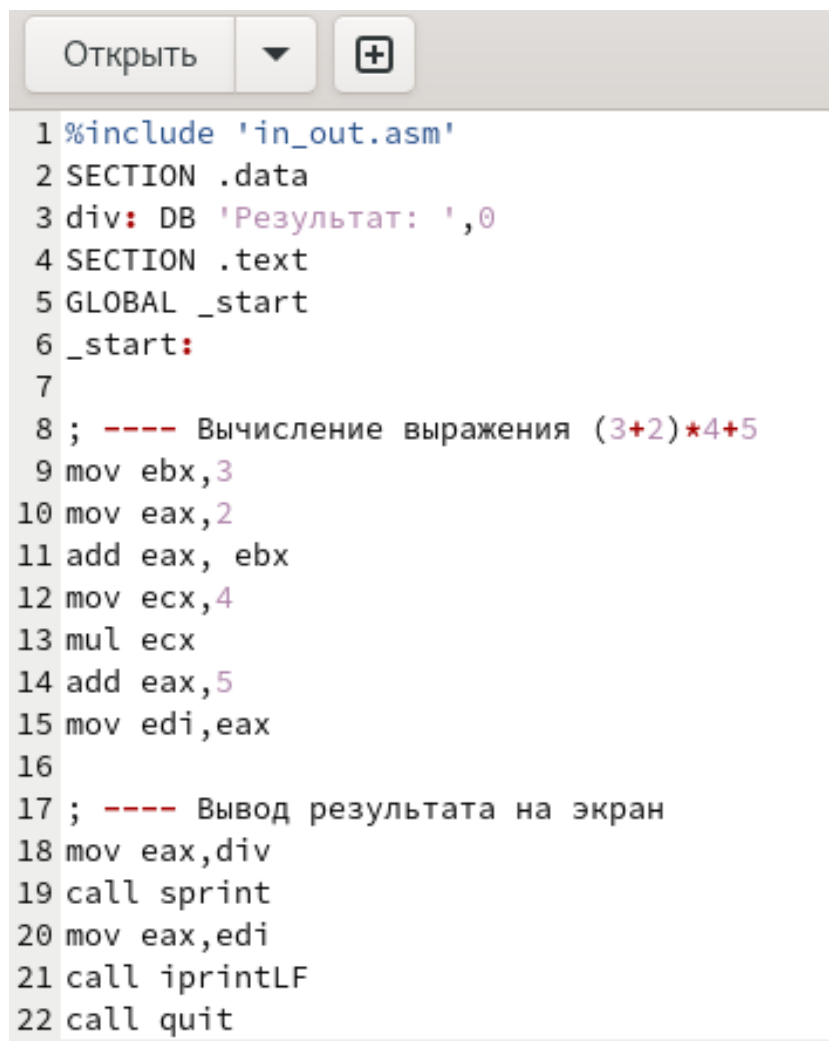
native process 63405 In: _start L9 PC: 0x4011c8
(gdb) rprocess 63439 In: _start L15 PC: 0x4011de

Breakpoint 1, _start () at var7-2.asm:9
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 4.46: Просмотр изменения регистра

Меняю текст программы таким образом, чтобы она работала корректно. (рис. 4.47)

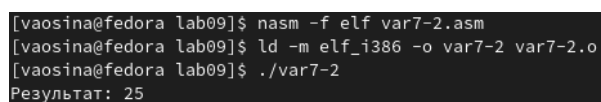


```
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7
8 ; ---- Вычисление выражения (3+2)*4+5
9 mov ebx,3
10 mov eax,2
11 add eax, ebx
12 mov ecx,4
13 mul ecx
14 add eax,5
15 mov edi,eax
16
17 ; ---- Вывод результата на экран
18 mov eax,div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

Рис. 4.47: Изменение текста программы

Создаю исполняемый файл и запускаю его. (рис. 4.48)

Теперь программа работает корректно.



```
[vaosina@fedora lab09]$ nasm -f elf var7-2.asm
[vaosina@fedora lab09]$ ld -m elf_i386 -o var7-2 var7-2.o
[vaosina@fedora lab09]$ ./var7-2
Результат: 25
```

Рис. 4.48: Создание исполняемого файла и его запуск

### Текст программы:

```
%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
```

## 5 Выводы

При выполнении данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм и познакомилась с методами отладки при помощи GDB и его основными возможностями.



## **6 Список литературы**

### **1. Архитектура ЭВМ**