

# **Отчет по лабораторной работе №9**

**Дисциплина: архитектура компьютера**

Панина Жанна Валерьевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
4.1	Релаксация подпрограмм в NASM . . . . .	8
4.1.1	Отладка программ с помощью GDB . . . . .	10
4.1.2	Добавление точек останова . . . . .	12
4.1.3	Работа с данными программы в GDB . . . . .	13
4.1.4	Обработка аргументов командной строки в GDB . . . . .	16
4.2	Задание для самостоятельной работы . . . . .	17
<b>5</b>	<b>Выводы</b>	<b>23</b>
<b>6</b>	<b>Список литературы</b>	<b>24</b>

## Список иллюстраций

4.1	Создание рабочего каталога . . . . .	8
4.2	Запуск программы из листинга . . . . .	8
4.3	Изменение программы первого листинга . . . . .	9
4.4	Запуск изменённой программы . . . . .	9
4.5	Проверка программы отладчиком . . . . .	10
4.6	Запуск отладчика с брейкпойнтом . . . . .	10
4.7	Дисассимилирование программы . . . . .	11
4.8	Режим псевдографики . . . . .	12
4.9	Список брейкпойнтов . . . . .	12
4.10	Добавление второй точки останова . . . . .	13
4.11	Просмотр содержимого регистров . . . . .	13
4.12	Просмотр содержимого переменных двумя способами . . . . .	14
4.13	Изменение содержимого переменных двумя способами . . . . .	14
4.14	Просмотр значения регистра разными представлениями . . . . .	15
4.15	Примеры использования команды set . . . . .	15
4.16	Подготовка новой программы . . . . .	16
4.17	Проверка работы стека . . . . .	17
4.18	Измененная программа предыдущей лабораторной работы . . . . .	18
4.19	Запуск программы . . . . .	18
4.20	Поиск ошибки в программе через пошаговую отладку . . . . .	21
4.21	Проверка корректировок в программме . . . . .	21

## **Список таблиц**

# 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. Самостоятельное выполнение заданий по материалам лабораторной работы

### 3 Теоретическое введение

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).

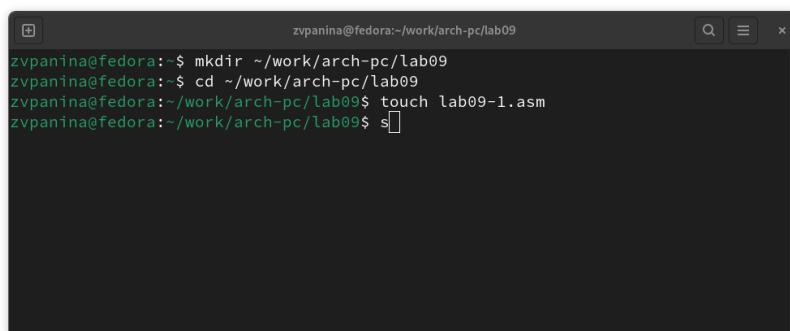
Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга.

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.

## 4 Выполнение лабораторной работы

### 4.1 Релазация подпрограмм в NASM

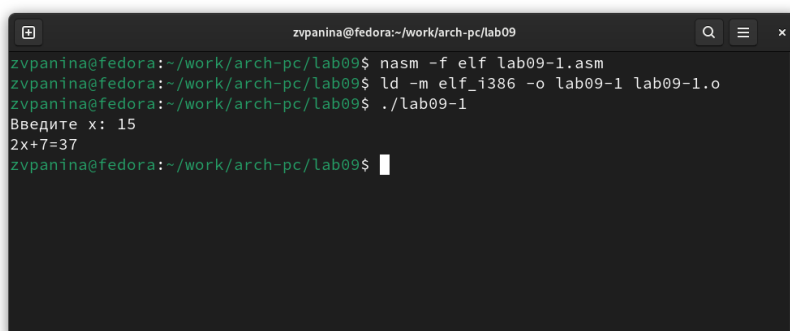
Создаю каталог для выполнения лабораторной работы №9, перехожу в него и создаю файл lab09-1.asm (рис. -fig. 4.1).



```
zvpanina@fedora:~/work/arch-pc/lab09
zvpanina@fedora:~$ mkdir ~/work/arch-pc/lab09
zvpanina@fedora:~$ cd ~/work/arch-pc/lab09
zvpanina@fedora:~/work/arch-pc/lab09$ touch lab09-1.asm
zvpanina@fedora:~/work/arch-pc/lab09$ s
```

Рис. 4.1: Создание рабочего каталога

Копирую в файл код из листинга, компилирую и запускаю его, данная программа выполняет вычисление функции (рис. -fig. 4.2).

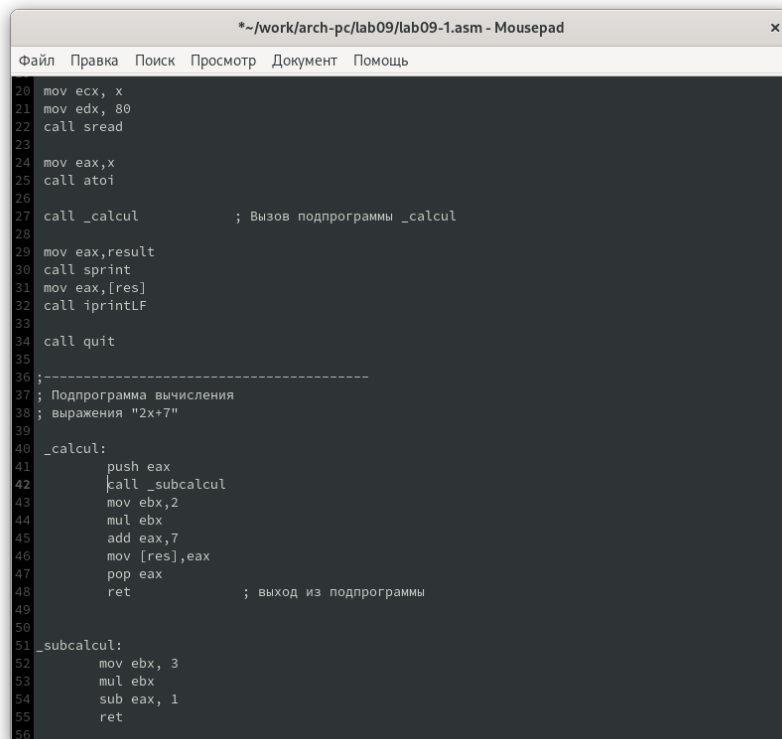


```
zvpanina@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
zvpanina@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
zvpanina@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 15
2x+7=37
zvpanina@fedora:~/work/arch-pc/lab09$
```

Рис. 4.2: Запуск программы из листинга



Изменяю текст программы, добавив в нее подпрограмму `_subcalcul` (рис. - fig. 4.3).

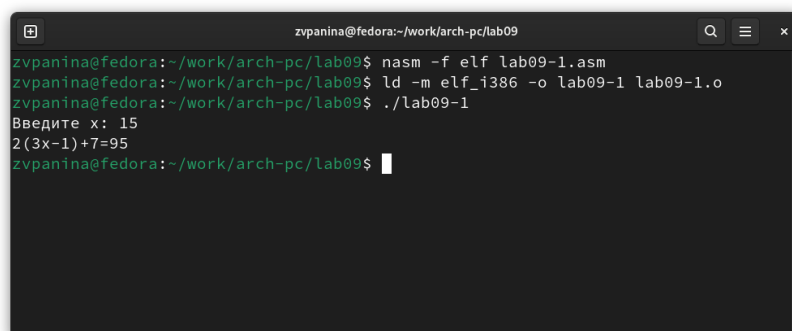


```
*~/work/arch-pc/lab09/lab09-1.asm - Mousepad
Файл  Правка  Поиск  Просмотр  Документ  Помощь

20 mov ecx, x
21 mov edx, 80
22 call sread
23
24 mov eax, x
25 call atoi
26
27 call _calcul          ; Вызов подпрограммы _calcul
28
29 mov eax, result
30 call sprint
31 mov eax, [res]
32 call iprintLF
33
34 call quit
35
36 ;-----
37 ; Подпрограмма вычисления
38 ; выражения "2x+7"
39
40 _calcul:
41     push eax
42     call _subcalcul
43     mov ebx, 2
44     mul ebx
45     add eax, 7
46     mov [res], eax
47     pop eax
48     ret          ; выход из подпрограммы
49
50
51 _subcalcul:
52     mov ebx, 3
53     mul ebx
54     sub eax, 1
55     ret
56
```

Рис. 4.3: Изменение программы первого листинга

Теперь она вычисляет значение функции для выражения  $f(g(x))$  (рис. -fig. 4.4).

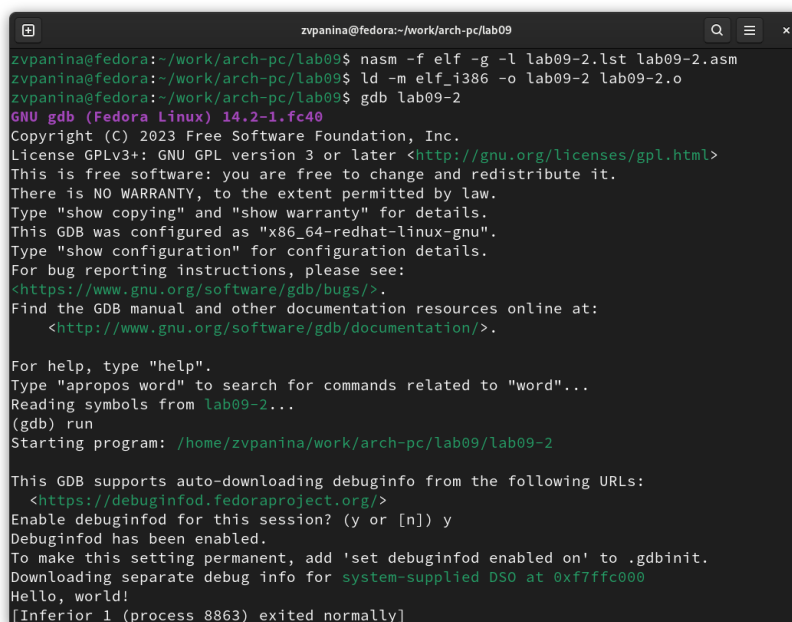


```
zvpanina@fedora:~/work/arch-pc/lab09
zvpanina@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
zvpanina@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
zvpanina@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 15
2(3x-1)+7=95
zvpanina@fedora:~/work/arch-pc/lab09$
```

Рис. 4.4: Запуск изменённой программы

### 4.1.1 Отладка программ с помощью GDB

В созданный файл копирую программу второго листинга, транслирую с созданием файла листинга и отладки, компоную и запускаю в отладчике. Запустив программу командой run, я убедилась в том, что она работает исправно (рис. -fig. 4.5).



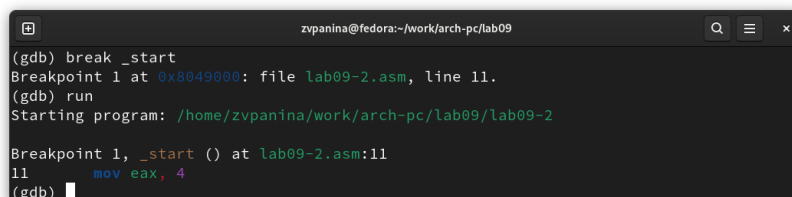
```
zypanina@fedora:~/work/arch-pc/lab09
zypanina@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
zypanina@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
zypanina@fedora:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 14.2-1.fc40
Copyright (C) 2023 Free 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.
Type "show 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
Starting program: /home/zypanina/work/arch-pc/lab09/lab09-2

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.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 8863) exited normally]
```

Рис. 4.5: Проверка программы отладчиком

Для более подробного анализа программы добавляю брейкпоинт на метку \_start и снова запускаю отладку (рис. -fig. 4.6).



```
zypanina@fedora:~/work/arch-pc/lab09
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 11.
(gdb) run
Starting program: /home/zypanina/work/arch-pc/lab09/lab09-2

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

Рис. 4.6: Запуск отладчика с брейкпоинтом

Далее смотрю дисассимилированный код программы, переключаюсь на отображение команд с синтаксисом Intel (рис. -fig. 4.7).

Различия между синтаксисом АТТ и Intel заключаются в порядке операндов (АТТ - Операнд источника указан первым. Intel - Операнд назначения указан первым), их размере (АТТ - размер операндов указывается явно с помощью суффиксов, непосредственные операнды предваряются символом \$; Intel - Размер операндов неявно определяется контекстом, как ax, eax, непосредственные операнды пишутся напрямую), именах регистров (АТТ - имена регистров предваряются символом %, Intel - имена регистров пишутся без префиксов).

```

(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. 4.7: Дисассимилирование программы

Включаю режим псевдографики для более удобного анализа программы (рис. -fig. 4.8).

```

Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffcf10 0xffffcf10  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
fs       0x0      0      gs       0x0      0

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,0x804a000
0x8049025 <_start+37> mov edx,0x7
0x804902a <_start+42> int 0x80

native process 5886 (asm) In: _start L11 PC: 0x8049000
(gdb) layout regs
(gdb)

```

Рис. 4.8: Режим псевдографики

## 4.1.2 Добавление точек останова

Проверяю в режиме псевдографики, что брейкпоинт сохранился (рис. -fig. 4.9).

```

Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffcf10 0xffffcf10  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
fs       0x0      0      gs       0x0      0

0x80496f2 add BYTE PTR [eax],al
0x80496f4 add BYTE PTR [eax],al
0x80496f6 add BYTE PTR [eax],al
0x80496f8 add BYTE PTR [eax],al
0x80496fa add BYTE PTR [eax],al
0x80496fc add BYTE PTR [eax],al
0x80496fe add BYTE PTR [eax],al
0x8049700 add BYTE PTR [eax],al
0x8049702 add BYTE PTR [eax],al
0x8049704 add BYTE PTR [eax],al

native process 5886 (asm) In: _start L11 PC: 0x8049000
breakpoint already hit 1 time
(gdb) layout asm
(gdb) layout regs
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 24.
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x8049000 lab9-2.asm:11
breakpoint already hit 1 time
2 breakpoint keep y 0x8049031 lab9-2.asm:24
(gdb)

```

Рис. 4.9: Список брейкпоинтов

Устанавливаю еще одну точку останова по адресу инструкции (рис. -fig. 4.10).

```

Register group: general
eax 0x0 0 0 ecx 0x0 0
edx 0x0 0 0 ebx 0x0 0
esp 0xffffcf10 0xffffcf10 ebp 0x0 0x0
esi 0x0 0 0 edi 0x0 0
eip 0x8049000 0x8049000 <_start> eflags 0x202 [ IF ]
cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43
fs 0x0 0 gs 0x0 0

0x80496f2 add BYTE PTR [eax],al
0x80496f4 add BYTE PTR [eax],al
0x80496f6 add BYTE PTR [eax],al
0x80496f8 add BYTE PTR [eax],al
0x80496fa add BYTE PTR [eax],al
0x80496fc add BYTE PTR [eax],al
0x80496fe add BYTE PTR [eax],al
0x8049700 add BYTE PTR [eax],al
0x8049702 add BYTE PTR [eax],al
0x8049704 add BYTE PTR [eax],al

native process 5886 (asm) In: _start L11 PC: 0x8049000
(gdb) breakpoint already hit 1 time
(gdb) layout asm
(gdb) layout regs
(gdb) layout regs
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 24.
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x8049000 lab9-2.asm:11
breakpoint already hit 1 time
2 breakpoint keep y 0x8049031 lab9-2.asm:24
(gdb)

```

Рис. 4.10: Добавление второй точки останова

### 4.1.3 Работа с данными программы в GDB

Просматриваю содержимое регистров командой info registers (рис. -fig. 4.11).

```

Register group: general
eax 0x0 8 ecx 0x804a000 134520832
edx 0x0 8 ebx 0x1 1
esp 0xffffcf10 0xffffcf10 ebp 0x0 0x0
esi 0x0 0 edi 0x0 0
eip 0x8049016 0x8049016 <_start+22> eflags 0x202 [ IF ]
cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43
fs 0x0 0 gs 0x0 0

B* 0x8049000 <_start> mov eax,0x4
0x8049005 <_start+5> mov ebx,0x1
0x804900a <_start+10> mov ecx,0x804a000
0x804900f <_start+15> mov edx,0x0
0x8049014 <_start+20> int 0x0
>0x8049016 <_start+22> mov eax,0x4
0x804901b <_start+27> mov ebx,0x1
0x8049020 <_start+32> mov ecx,0x804a000
0x8049025 <_start+37> mov edx,0x7
0x804902a <_start+42> int 0x0

native process 5886 (asm) In: _start L17 PC: 0x8049016
eax 0x0 8
ecx 0x804a000 134520832
edx 0x0 8
ebx 0x1 1
esp 0xffffcf10 0xffffcf10
ebp 0x0 0x0
esi 0x0 0
edi 0x0 0
eip 0x8049016 0x8049016 <_start+22>
eflags 0x202 [ IF ]
cs 0x23 35
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 4.11: Просмотр содержимого регистров

Смотрю содержимое переменных по имени и по адресу (рис. -fig. 4.12).

```

Register group: general
eax 0x8 8 ecx 0x804a000 134520832
edx 0x8 8 ebx 0x1 1
esp 0xffffcf10 0xffffcf10 ebp 0x0 0x0
esi 0x0 0 edi 0x0 0
eip 0x8049016 0x8049016 <_start+22> eflags 0x202 [ IF ]
cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43
fs 0x0 0 gs 0x0 0

B+ 0x8049000 <_start> mov eax,0x4
0x8049005 <_start+5> mov ebx,0x1
0x804900a <_start+10> mov ecx,0x804a008
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 5886 (asm) In: _start L17 PC: 0x8049016
esi 0x0 0
edi 0x0 0
eip 0x8049016 0x8049016 <_start+22>
eflags 0x202 [ IF ]
cs 0x23 35
--Type <RET> for more, q to quit, c to continue without paging--
Quit
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "World!\n\034"
(gdb)

```

Рис. 4.12: Просмотр содержимого переменных двумя способами

Меняю содержимое переменных по имени и по адресу (рис. -fig. 4.13).

```

Register group: general
eax 0x8 8 ecx 0x804a000 134520832
edx 0x8 8 ebx 0x1 1
esp 0xffffcf10 0xffffcf10 ebp 0x0 0x0
esi 0x0 0 edi 0x0 0
eip 0x8049016 0x8049016 <_start+22> eflags 0x202 [ IF ]
cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43
fs 0x0 0 gs 0x0 0

B+ 0x8049000 <_start> mov eax,0x4
0x8049005 <_start+5> mov ebx,0x1
0x804900a <_start+10> mov ecx,0x804a008
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 5886 (asm) In: _start L17 PC: 0x8049016
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "World!\n\034"
(gdb) set {char}msg1='h'
'msg1' has unknown type; cast it to its declared type
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb) set {char}&msg2='x'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "xorld!\n\034"
(gdb)

```

Рис. 4.13: Изменение содержимого переменных двумя способами

Вывожу в различных форматах значение регистра edx (рис. -fig. 4.14).

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd070 0xffffd070
ebp      0x0      0
esi      0x0      0

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

native process 10469 (asm) In: _start      L15    PC: 0x8049016
(gdb) p/t $ecx
$2 = 100000000100101000000000000000
(gdb) p/s $edx
$3 = 8
(gdb) p/t $edx
$4 = 1000
(gdb) p/x $edx
$5 = 0x8
(gdb)

```

Рис. 4.14: Просмотр значения регистра разными представлениями

С помощью команды set меняю содержимое регистра ebx (рис. -fig. 4.15).

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x2      2
esp      0xffffd070 0xffffd070
ebp      0x0      0
esi      0x0      0

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

native process 10469 (asm) In: _start      L15    PC: 0x8049016
(gdb) set $ebx='2'
(gdb) p/s
$6 = 8
(gdb) p/s $ebx
$7 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$8 = 2
(gdb)

```

Рис. 4.15: Примеры использования команды set

#### 4.1.4 Обработка аргументов командной строки в GDB

Копирую программу из предыдущей лабораторной работы в текущий каталог и создаю исполняемый файл с файлом листинга и отладки (рис. -fig. 4.16).

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

Рис. 4.16: Подготовка новой программы

Запускаю программу с режиме отладки с указанием аргументов, указываю брейкпоинт и запускаю отладку. Проверяю работу стека, изменяя аргумент команды просмотра регистра `esp` на `+4`, число обусловлено разрядностью системы, а указатель `void` занимает как раз 4 байта, ошибка при аргументе `+24` означает, что аргументы на вход программы закончились. (рис. -fig. 4.17).



```
zspanina@fedora:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Fedora Linux) 14.2-1.fc40
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) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 10.
(gdb) run
Starting program: /home/zspanina/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:10
10      pop ecx          ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffd070: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffd23c: "/home/zspanina/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd260: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd278: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd280: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd28b: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 4.17: Проверка работы стека

## 4.2 Задание для самостоятельной работы

1. Меняю программу самостоятельной части предыдущей лабораторной работы с использованием подпрограммы (рис. -fig. 4.18).

```

1 ;-----
2 ; Программа вычисления суммы значений функции f(x) для x = x1, x2, ..., xn
3 ;-----
4 %include 'in_out.asm'
5
6 SECTION .data
7 msg_func db "Функция: f(x) = 15x + 2", 0
8 msg_result db "Результат: ", 0
9
10 SECTION .text
11 GLOBAL _start
12
13 _start:
14 mov eax, msg_func
15 call sprintf
16
17 pop ecx
18 pop edx
19 sub ecx, 1
20 mov esi, 0
21
22 next:
23 cmp ecx, 0h
24 jz _end
25 pop eax
26 call atoi
27
28 call _calcul
29
30 add esi, eax
31 loop next
32
33 _end:
34 mov eax, msg_result
35 call sprintf
36 mov eax, esi
37 call iprintf
38 call quit
39
40 _calcul:
41 mov ebx, 15
42 mul ebx
43 add eax, 2
44 ret
45

```

Рис. 4.18: Измененная программа предыдущей лабораторной работы

Запускаю исполняемый файл и убеждаюсь, что ошибок в коде нет (рис. -fig. 4.19).

```

zvpanina@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
zvpanina@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
zvpanina@fedora:~/work/arch-pc/lab09$ ./lab09-4 1 2 3 4
Функция: f(x) = 15x + 2
Результат: 158
zvpanina@fedora:~/work/arch-pc/lab09$

```

Рис. 4.19: Запуск программы

Код программы:

```

;-----
; Программа вычисления суммы значений функции f(x) для x = x1, x2, ..., xn
;-----
%include 'in_out.asm'

```

## SECTION .data

msg\_func db "Функция:  $f(x) = 15x + 2$ ", 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, 0h

jz \_end

pop eax

call atoi

call \_calcul

add esi, eax

loop next

\_end:

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

```
_calcul:
mov ebx, 15
mul ebx
add eax, 2
ret
```

2. Запускаю программу в режиме отладчика и пошагово через si просматриваю изменение значений регистров через i r. При выполнении инструкции mul esx 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. -fig. 4.20).

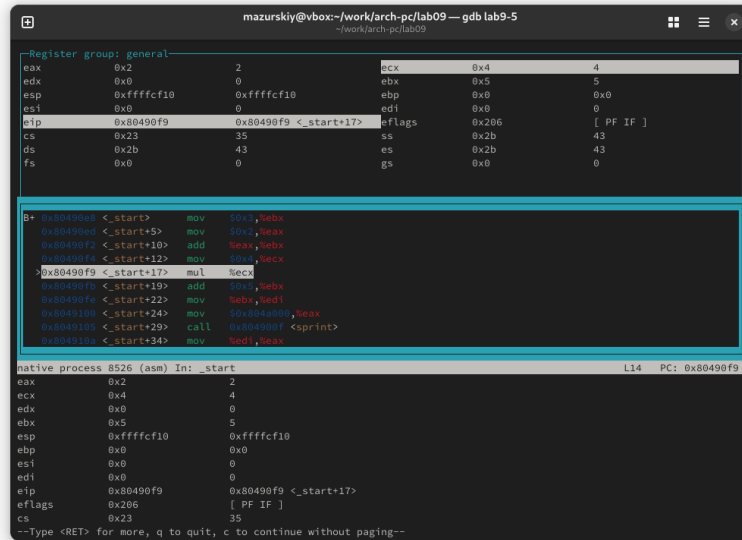


Рис. 4.20: Поиск ошибки в программе через пошаговую отладку

Исправляю найденную ошибку, теперь программа верно считает значение функции (рис. -fig. 4.21).

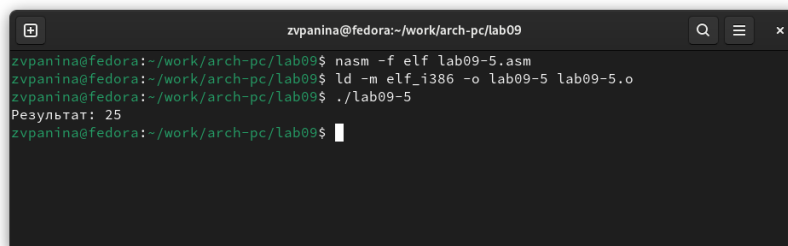


Рис. 4.21: Проверка корректировок в программе

Код измененной программы:

```

;-----
; Программа вычисления выражения (3+2)*4+5
;-----
#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 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**

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

## **6 Список литературы**

1. Курс на ТУИС
2. Лабораторная работа №9
3. Программирование на языке ассемблера NASM Столяров А. В.