

# **Отчет по лабораторной работе №9**

**Дисциплина: архитектура компьютера**

Аннагулыев Арслан

# 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 . . . . .	14
4.1.3	Работа с данными программы в GDB . . . . .	15
4.1.4	Обработка аргументов командной строки в GDB . . . . .	17
4.2	Задание для самостоятельной работы . . . . .	18
<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 . . . . .	8
4.4	Запуск программы в отладчике . . . . .	11
4.5	Проверка программы отладчиком . . . . .	11
4.6	Запуск отладчика с брейкпойнтом . . . . .	12
4.7	Дисассимилирование программы . . . . .	13
4.8	Режим псевдографики . . . . .	14
4.9	Список брейкпойнтов . . . . .	14
4.10	Добавление второй точки останова . . . . .	15
4.11	Просмотр содержимого регистров . . . . .	15
4.12	Просмотр содержимого переменных двумя способами . . . . .	16
4.13	Изменение содержимого переменных двумя способами . . . . .	16
4.14	Просмотр значения регистра разными представлениями . . . . .	17
4.15	Примеры использования команды set . . . . .	17
4.16	Подготовка новой программы . . . . .	18
4.17	Проверка работы стека . . . . .	18
4.18	Измененная программа предыдущей лабораторной работы . . . . .	19
4.19	Поиск ошибки в программе через пошаговую отладку . . . . .	21
4.20	Проверка корректировок в программе . . . . .	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 Релаксация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы №9 (рис. 4.1).

```
vboxuser@rabot:~$ cd work/study/2024-2025/Архитектура\ компьютера/arch-pc/  
vboxuser@rabot:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ mkdir lab09  
vboxuser@rabot:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ touch lab9-1.asm  
vboxuser@rabot:~/work/study/2024-2025/Архитектура компьютера/arch-pc$
```

Рис. 4.1: Создание рабочего каталога

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

```
vboxuser@rabot:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ nasm -f elf lab9-1.asm  
vboxuser@rabot:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o  
vboxuser@rabot:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ ./lab9-1  
Введите x: 10  
2x+7=27
```

Рис. 4.2: Запуск программы из листинга

Изменяю текст программы, добавив в нее подпрограмму, теперь она вычисляет значение функции для выражения  $f(g(x))$  (рис. 4.3).

```
vboxuser@rabot:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ nasm -f elf lab9-1.asm  
vboxuser@rabot:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o  
vboxuser@rabot:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ ./lab9-1  
Введите x: 10  
2(3x-1)+7=65
```

Рис. 4.3: Изменение программы первого листинга

Код программы:

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



**SECTION** .data

msg: DB 'Введите x: ', 0

result: DB '2(3x-1)+7=', 0

**SECTION** .bss

x: RESB 80

res: RESB 80

**SECTION** .text

**GLOBAL** \_start

\_start:

mov eax, msg

call sprint

mov ecx, x

mov edx, 80

call sread

mov eax, x

call atoi

call \_calcul

mov eax, result

call sprint

mov eax, [res]

call iprintLF

call quit

```
_calcul:
push eax
call _subcalcul

mov ebx, 2
mul ebx
add eax, 7

mov [res], eax
pop eax
ret

_subcalcul:
mov ebx, 3
mul ebx
sub eax, 1
ret
```

#### 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 (рис. 4.4).

```
vboxuser@rabort:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ gdb lab9-2
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab9-2...
(gdb) █
```

Рис. 4.4: Запуск программы в отладчике

Запустив программу командой `gdb`, я убедился в том, что она работает исправно (рис. 4.5).

```
vboxuser@rabort:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ gdb lab9-2
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab9-2...
(gdb) run
Starting program: /home/vboxuser/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09/lab9-2

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 346215) exited normally]
(gdb) █
```

Рис. 4.5: Проверка программы отладчиком

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

```

(gdb) run
Starting program: /home/vboxuser/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 346294) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 9.
(gdb) run
Starting program: /home/vboxuser/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:9
9      mov eax, 4
(gdb)

```

Рис. 4.6: Запуск отладчика с брейкпоинтом

Далее смотрю дисассимилированный код программы, перевожу на команд с синтаксисом Intel *amd топчик* (рис. 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 disassemble-flavor intel
No symbol "disassemble" in current context.
(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

```

Рис. 4.7: Дисассимилирование программы

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

```

Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffcf20 0xffffcf20  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

native process 346318 (asm) In: _start      L9      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. 4.9).

```

Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffcf20 0xffffcf20  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

native process 346318 (asm) In: _start      L9      PC: 0x8049000
(gdb) layout regs
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 20.
(gdb) i b
Num   Type      Disp Enb Address  What
1     breakpoint keep  y  0x8049000 lab9-2.asm:9
      breakpoint already hit 1 time
2     breakpoint keep  y  0x8049031 lab9-2.asm:20
(gdb)

```

Рис. 4.9: Список брейкпоинтов

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

```

Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffcf20 0xffffcf20  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

native process 346318 (asm) In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 20.
(gdb) i b
Num   Type             Disp Enb Address      What
1     breakpoint       keep y  0x8049000 lab9-2.asm:9
      breakpoint already hit 1 time
2     breakpoint       keep y  0x8049031 lab9-2.asm:20
(gdb)

```

Рис. 4.10: Добавление второй точки останова

### 4.1.3 Работа с данными программы в GDB

Просматриваю содержимое регистров командой `info registers` (рис. 4.11).

```

Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffcf20 0xffffcf20  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

0x804901b <_start+27> mov    ebx,0x1
0x8049020 <_start+32> mov    ecx,0x804a000
0x8049025 <_start+37> mov    edx,0x7
0x804902a <_start+42> int    0x80
0x804902c <_start+44> mov    eax,0x1
b> 0x8049031 <_start+49> mov    ebx,0x0
0x8049036 <_start+54> int    0x80
0x8049038 add    BYTE PTR [eax],al

native process 346318 (asm) In: _start L9 PC: 0x8049000
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcf20 0xffffcf20
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
--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. 4.12).

```

Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffcf20 0xffffcf20  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

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
b> 0x8049031 <_start+49> mov     ebx,0x0
0x8049036 <_start+54> int     0x80

native process 346318 (asm) In: _start L9 PC: 0x8049000
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
es       0x2b     43
fs       0x0      0
gs       0x0      0
(gdb) x/lsb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/lsb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb)

```

Рис. 4.12: Просмотр содержимого переменных двумя способами

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

```

Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffcf20 0xffffcf20  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

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
b> 0x8049031 <_start+49> mov     ebx,0x0
0x8049036 <_start+54> int     0x80

native process 346318 (asm) In: _start L9 PC: 0x8049000
(gdb) x/lsb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/lsb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb) set {char}&msg1='h'
(gdb) x/lsb &msg1
0x804a000 <msg1>: "hello, "
(gdb) set {char}&msg2='x'
(gdb) x/lsb &msg2
0x804a008 <msg2>: "xorld!\n\034"
(gdb)

```

Рис. 4.13: Изменение содержимого переменных двумя способами

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



```

Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x32      50
esp      0xffffcf20 0xffffcf20  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

native process 346318 (asm) In: _start L9 PC: 0x8049000
$2 = 0
(gdb) set $ebx='2'
(gdb) p/t $ecx
$3 = 0
(gdb) p/t $edx
$4 = 0
(gdb) p/s $edx
$5 = 0
(gdb) p/x $edx
$6 = 0x0
(gdb)

```

Рис. 4.14: Просмотр значения регистра разными представлениями

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

```

Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x2      2
esp      0xffffcf20 0xffffcf20  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

native process 346318 (asm) In: _start L9 PC: 0x8049000
$6 = 0x0
(gdb) set $ebx='2'
(gdb) set $ebx='2'
(gdb) p/s
$7 = 0
(gdb) p/s $ebx
$8 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$9 = 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. 4.16).

```

vboxuser@rabot: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ cp ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08/lab8-2.asm ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09/lab9-3.asm
vboxuser@rabot: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
nasm: fatal: unable to open input file 'lab09-3.asm' No such file or directory
vboxuser@rabot: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ ls
in_out.asm lab9-1 lab9-1.asm lab9-1.o lab9-2 lab9-2.asm lab9-2.lst lab9-2.o lab9-3.asm
vboxuser@rabot: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
vboxuser@rabot: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ ld -m elf_i386 -o lab9-3 lab9-3.o
vboxuser@rabot: ~/work/study/2024-2025/Архитектура компьютера/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. 4.17).

```

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

Breakpoint 1, _start () at lab9-3.asm:7
7      pop ecx
(gdb) ~/home/vboxuser/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09/lab9-3
Undefined command: "~". Try "help".
(gdb) ~/home/vboxuser/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09/lab9-3
Undefined command: "~". Try "help".
(gdb) ~/home/vboxuser/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09/lab9-3
Undefined command: "~". Try "help".
(gdb) дело шло в 4:37
Undefined command: "~". Try "help".
(gdb) x/s *(void**)(esp + 4)
A syntax error in expression, near `)'($esp + 4)'.
(gdb) x/s *(void**)(esp + 4)
0xffffd000: "/home/vboxuser/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0x0: <error: cannot access memory at address 0x0>
(gdb) x/s *(void**)(esp + 12)
0xffffd008: "SHELL=/bin/bash"
(gdb) x/s *(void**)(esp + 16)
0xffffd010: "SESSION_MANAGER=local/rabot:0/tmp/.ICE-unix/2489,unix/rabot:/tmp/.ICE-unix/2489"
(gdb) x/s *(void**)(esp + 20)
0xffffd018: "QT_ACCESSIBILITY=1"
(gdb) x/s *(void**)(esp + 24)
0xffffd020: "COLORTERM=truecolor"
(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. 4.18).



```

File Edit Search View Document Help
%include 'in_out.asm'

SECTION .data
msg_func db "Функция: f(x) = 10x - 4", 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 _calculate_fx

add esi, eax
loop next

_end:
mov eax, msg_result

```

Рис. 4.18: Измененная программа предыдущей лабораторной работы

Код программы:

```

%include 'in_out.asm'

SECTION .data
msg_func db "Функция: f(x) = 10x - 4", 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 _calculate_fx

add esi, eax
loop next

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

_calculate_fx:
mov ebx, 10
mul ebx
sub eax, 4

```

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. 4.19).

```

--Register group: general--
eax    0x0      0      ecx    0x32    50
edx    0x0      0      ebx    0x0     0
esp    0xffffcf20 0xffffcf20 ebp    0x0     0x0
esi    0x0      0      edi    0x0     0
eip    0x32     0x32   eflags 0x202   [ IF ]
cs     0x23     35     ss     0x2b    43
ds     0x2b     43     es     0x2b    43
fs     0x0      0      gs     0x0     0

0x80490ed <_start+5> call 0x80490d2 <sprintf>
0x80490f2 <_start+10> pop  ecx
0x80490f3 <_start+11> pop  edx
0x80490f4 <_start+12> sub  ecx,0x1
0x80490f7 <_start+15> mov  esi,0x0
0x80490fc <next>     cmp  ecx,0x0
0x80490ff <next+3>    je   0x8049110 <_end>
0x8049101 <next+5>    pop  eax
0x8049102 <next+6>    call 0x804909c <atoi>

native process 352896 (asm) In: _start L11 PC: 0x80490e8
eax    0x0      0
ecx    0x32     50
edx    0x0      0
ebx    0x0      0
esp    0xffffcf20 0xffffcf20
ebp    0x0      0x0
esi    0x0      0
edi    0x0      0
eip    0x32     0x32
eflags 0x202    [ IF ]
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 4.19: Поиск ошибки в программе через пошаговую отладку

Исправляю найденную ошибку, теперь программа верно считает значение функции (рис. 4.20).

```

vboxuser@rabot:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab0$ rm lab9-5.asm
vboxuser@rabot:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab0$ nasm -f elf lab9-5.asm
vboxuser@rabot:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab0$ ld -m elf_i386 -o lab9-5 lab9-5.o
vboxuser@rabot:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab0$ ./lab9-5
Результат: 25

```

Рис. 4.20: Проверка корректировок в программе

Код измененной программы:

```

#include 'in_out.asm'

SECTION .data
div: DB 'Результат: ', 0

SECTION .text
GLOBAL _start
_start:

mov ebx, 3
mov eax, 2

```

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

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

```
call quit
```

## **5 Выводы**

В результате выполнения данной лабораторной работы я приобрел навыки написания программ с использованием подпрограмм, а так же познакомился 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.