

# **Отчет по лабораторной работе №9**

**дисциплина: Архитектура компьютера**

Бондарь Татьяна Владимировна

# 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	Реализация подпрограмм в NASM . . . . .	8
4.2	Отладка программ с помощью GDB . . . . .	11
4.3	Добавление точек останова . . . . .	14
4.4	Работа с данными программы в GDB . . . . .	15
4.5	Обработка аргументов командной строки в GDB . . . . .	19
<b>5</b>	<b>Задания для самостоятельной работы</b>	<b>21</b>
<b>6</b>	<b>Выводы</b>	<b>25</b>

# Список иллюстраций

4.1	Переход в каталог и создание файла . . . . .	8
4.2	Программа с использованием подпрограммы . . . . .	9
4.3	Исполнение программы из листинга 9.1 . . . . .	9
4.4	Исправленный текст программы lab09-1.asm . . . . .	10
4.5	Исполнение программы lab09-1 . . . . .	10
4.6	Текст программы из листинга 9.2 . . . . .	11
4.7	Трансляция программы . . . . .	11
4.8	Запуск программы в оболочке отладчика . . . . .	12
4.9	Исполнение программы с брейкпойнт . . . . .	12
4.10	Просмотр дисассимилированного кода программы . . . . .	12
4.11	Просмотр дисассимилированного кода программы с синтаксисом Intel . . . . .	13
4.12	Переход в режим псевдографики . . . . .	14
4.13	Проверка установки точки останова . . . . .	14
4.14	Установка новой точки останова . . . . .	15
4.15	Инструкция stepi . . . . .	16
4.16	Просмотр значения переменной . . . . .	16
4.17	Просмотр значения переменной . . . . .	17
4.18	Изменение переменной . . . . .	17
4.19	Вывод значения регистра . . . . .	18
4.20	Изменение значения регистра . . . . .	18
4.21	Загрузка файла lab09-3 в отладчик . . . . .	19
4.22	Проверка стека . . . . .	20
4.23	Проверка остальных позиций стека . . . . .	20
5.1	Текст программы lab09-4.asm . . . . .	21
5.2	Запуск программы . . . . .	22
5.3	Поиск ошибки в работе программы lab09-5.asm . . . . .	23
5.4	Запуск исправленной программы . . . . .	23
5.5	Исправленный текст программы . . . . .	24

## **Список таблиц**

# **1 *Цель работы***

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

## 2 Задание

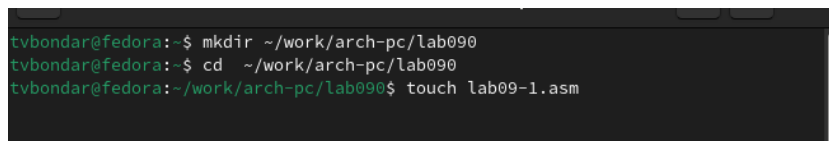
1. Преобразуйте программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции  $\varphi(x)$  как подпрограмму.
2. В листинге 9.3 приведена программа вычисления выражения  $(3 + 2) \cdot 4 + 5$ . При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее.

### **3 Теоретическое введение**

## 4 Выполнение лабораторной работы

### 4.1 Реализация подпрограмм в NASM

1. Создаю каталог для программ лабораторной работы №9, перехожу в него и создаю файл lab09-1.asm.



```
tvbondar@fedora:~$ mkdir ~/work/arch-pc/lab090
tvbondar@fedora:~$ cd ~/work/arch-pc/lab090
tvbondar@fedora:~/work/arch-pc/lab090$ touch lab09-1.asm
```

Рис. 4.1: Переход в каталог и создание файла

2. В качестве примера рассмотрим программу вычисления арифметического выражения  $\square(\square) = 2\square + 7$  с помощью подпрограммы `_calcul`. В данном примере  $\square$  вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Ввожу в файл lab09-1.asm текст программы из листинга 9.1. Запускаю исполняемый файл.



```

;Листинг 9.1. Пример программы с использованием вызова подпрограммы
#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. 4.2: Программа с использованием подпрограммы

```

tvbondar@fedora:~/work/arch-pc/lab090$ nasm -f elf lab09-1.asm
tvbondar@fedora:~/work/arch-pc/lab090$ ld -m elf_i386 lab09-1.o -o lab09-1
tvbondar@fedora:~/work/arch-pc/lab090$ ./lab09-1
Введите x: 9
2x+7=25
tvbondar@fedora:~/work/arch-pc/lab090$

```

Рис. 4.3: Исполнение программы из листинга 9.1

- Изменим текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения  $\square(\square(\square))$ , где  $\square$  вводится с клавиатуры,  $\square(\square) = 2\square + 7$ ,  $\square(\square) = 3\square - 1$ . Запустим исправленную программу. Число проходов цикла не соответствует значению, введенному с клавиатуры.

```

;Рисунок 9.1. Пример программы с использованием вызова подпрограммы
%include "io_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,4
    call atoi
    call _calcul          ; Вызов подпрограммы _calcul
    mov eax,result
    call sprint
    mov eax,[res]
    call jprintlf
    call quit

; Подпрограмма вычисления выражения "2x+7"
_calcul:
    call _subcalcul
    mov ebx,2
    mul ebx
    add eax,7
    mov [res],eax
    ret

; подпрограмма subcalcul
_subcalcul:
    mov ebx,3
    mul ebx
    sub eax,1
    mov [res],eax
    ret

```

Рис. 4.4: Исправленный текст программы lab09-1.asm

```

tvbondar@fedora:~/work/arch-pc/lab090$ nasm -f elf lab09-1.asm
tvbondar@fedora:~/work/arch-pc/lab090$ ld -m elf_i386 lab09-1.o -o lab09-1
tvbondar@fedora:~/work/arch-pc/lab090$ ./lab09-1
Введите x: 1
2x+7=11
tvbondar@fedora:~/work/arch-pc/lab090$ ./lab09-1
Введите x: 6
2x+7=41
tvbondar@fedora:~/work/arch-pc/lab090$

```

Рис. 4.5: Исполнение программы lab09-1

## 4.2 Отладка программ с помощью GDB

4. Создаем файл lab09-2.asm. Вводим в него программу из листинга 9.2. Транслируем текст программы с ключом '-g'. Загружаем исполняемый файл в gdb.

```
;Листинг 9.2. Программа вывода сообщения Hello world!
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. 4.6: Текст программы из листинга 9.2

```
tvbondar@fedora:~/work/arch-pc/lab090$ touch lab09-2.asm
tvbondar@fedora:~/work/arch-pc/lab090$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
tvbondar@fedora:~/work/arch-pc/lab090$ ld -m elf_i386 -o lab09-2 lab09-2.o
tvbondar@fedora:~/work/arch-pc/lab090$ gdb lab09-2
GNU gdb (Fedora Linux) 15.2-2.fc40
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-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. 4.7: Трансляция программы

6. Проверим работу программы, запустив ее в оболочке отладчика. Ошибок не обнаружено.

```
Reading symbols from lab09-2...
(gdb) run
Starting program: /home/tvbondar/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 10335) exited normally]
(gdb)
```

Рис. 4.8: Запуск программы в оболочке отладчика

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

```
(gdb) break _start
Breakpoint 1 at 0x08049000: file lab09-2.asm, line 10.
(gdb) run
Starting program: /home/tvbondar/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:10
10      mov eax, 4
(gdb) █
```

Рис. 4.9: Исполнение программы с брейкпойнт

8. Посмотрим дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start`.

```
(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) █
```

Рис. 4.10: Просмотр дисассимилированного кода программы

9. Переключимся на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`. 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) 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. 4.11: Просмотр дисассимилированного кода программы с синтаксисом Intel

10. Включим режим псевдографики для более удобного анализа программы.

```

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd070 0xffffd070
ebp      0x0      0x0

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

native process 10469 (asm) In: _start L10 PC: 0x8049000
(gdb) layout regs
(gdb)

```

Рис. 4.12: Переход в режим псевдографики

## 4.3 Добавление точек останова

11. Проверим установку точки останова на метке '\_start'.

```

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd070 0xffffd070
ebp      0x0      0x0

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

native process 10469 (asm) In: _start L10 PC: 0x8049000
(gdb) layout regs
(gdb) info breakpoints
Num      Type      Disp Enb Address      What
1        breakpoint keep y 0x08049000 lab09-2.asm:10
          breakpoint already hit 1 time
(gdb)

```

Рис. 4.13: Проверка установки точки останова

12. Установим еще одну точку останова по адресу инструкции. Определим ад-

рес предпоследней инструкции (mov ebx,0x0) и установим точку останова.

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd070 0xffffd070
ebp      0x0      0x0

0x80490a6  add  BYTE PTR [eax],al
0x80490a8  add  BYTE PTR [eax],al
0x80490aa  add  BYTE PTR [eax],al
0x80490ac  add  BYTE PTR [eax],al
0x80490ae  add  BYTE PTR [eax],al
0x80490b0  add  BYTE PTR [eax],al

native process 10469 (asm) In: _start L10 PC: 0x8049000
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 21.
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y 0x08049000 lab09-2.asm:10
          breakpoint already hit 1 time
2        breakpoint keep y 0x08049031 lab09-2.asm:21
(gdb)
```

Рис. 4.14: Установка новой точки останова

## 4.4 Работа с данными программы в GDB

13. Выполним 5 инструкций с помощью команды stepi (или si) и проследим за изменением значений регистров. Изменяются значения регистров eax, ebx, ecx, edx.

```

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

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

native process 10469 (asm) In: _start L15 PC: 0x8049016
breakpoint already hit 1 time
breakpoint keep y 0x08049031 lab09-2.asm:21
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb)

```

Рис. 4.15: Инструкция stepi

14. Посмотрим значение переменной msg1 по имени.

```

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

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

native process 10469 (asm) In: _start L15 PC: 0x8049016
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb)

```

Рис. 4.16: Просмотр значения переменной

15. Просмотрим значение переменной msg2 по адресу.



```

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

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

native process 10469 (asm) In: _start L15 PC: 0x8049016
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb)

```

Рис. 4.17: Просмотр значения переменной

16. Изменим первый символ переменной msg1. Заменяем любой символ во второй переменной msg2.

```

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

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

native process 10469 (asm) In: _start L15 PC: 0x8049016
0x804a000 <msg1>: "Hello, "
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb) set {char}0x804a008='L'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "Lorld!\n\034"
(gdb)

```

Рис. 4.18: Изменение переменной

17. Выведем в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx. С помощью

команды `set` изменим значение регистра `ebx`. В первом случае программа выводит значение кодировки символа '2' в шестнадцатеричной системе, а во втором переводит цифру 2 в шестнадцатеричный вид.

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd070 0xffffd070
ebp      0x0      0x0
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. 4.19: Вывод значения регистра

```

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

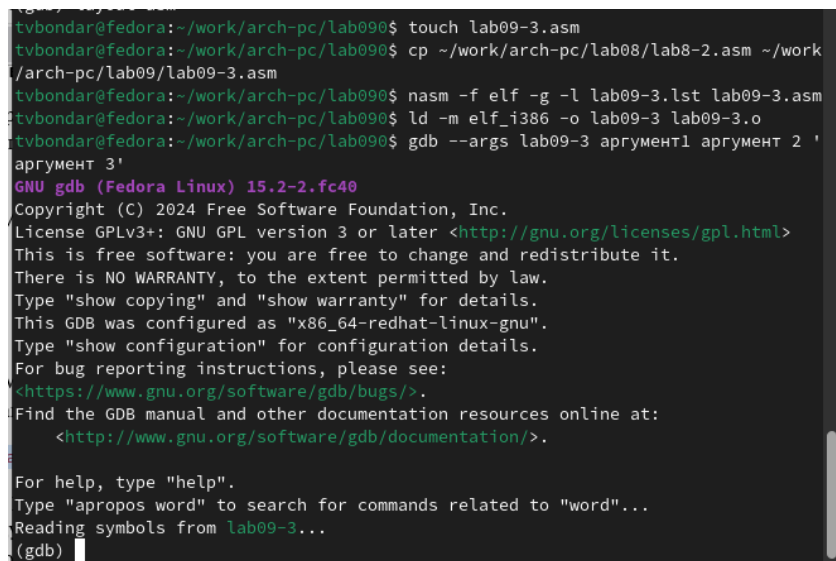
```

Рис. 4.20: Изменение значения регистра

Завершаю выполнение программы и выхожу из отладчика.

## 4.5 Обработка аргументов командной строки в GDB

18. Скопируем файл lab8-2.asm в файл с именем lab09-3.asm. Создадим исполняемый файл. Загрузим исполняемый файл в отладчик, указав аргументы.



```
tvbondar@fedora:~/work/arch-pc/lab09$ touch lab09-3.asm
tvbondar@fedora:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
tvbondar@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
tvbondar@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
tvbondar@fedora:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2 '
аргумент 3'
GNU gdb (Fedora Linux) 15.2-2.fc40
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-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. 4.21: Загрузка файла lab09-3 в отладчик

19. Для начала установим точку останова перед первой инструкцией в программе и запустим ее. Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы). Как видно, число аргументов равно 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент 2 и ‘аргумент 3’.

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 6.
(gdb) run
Starting program: /home/tvbondar/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 аргумент
ов (первое значение в стеке)
(gdb) x/x $esp
0xffffd030:    0x00000005
(gdb)

```

Рис. 4.22: Проверка стека

20. 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. Шаг изменения адреса равен 4 байтам, потому что мы работаем с 32-битной системой (x86), а указатели (`void **`) в такой системе занимают 4 байта. Ошибка `Cannot access memory at address 0x0` на `$esp + 24` указывает на то, что закончились аргументы командной строки.

```

Breakpoint 1, _start () at lab09-3.asm:6
6      pop ecx          ; Извлекаем из стека в `ecx` количество аргумент
ов (первое значение в стеке)
(gdb) x/x $esp
0xffffd030:    0x00000005
(gdb) x/s *(void**)(esp+4)
0xffffd1f3:    "/home/tvbondar/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp+8)
0xffffd21d:    "аргумент1"
(gdb) x/s *(void**)(esp+12)
0xffffd22f:    "аргумент"
(gdb) x/s *(void**)(esp+16)
0xffffd240:    "2"
(gdb) x/s *(void**)(esp+20)
0xffffd242:    "аргумент 3"
(gdb) x/s *(void**)(esp+24)
0x0:    <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 4.23: Проверка остальных позиций стека

## 5 Задания для самостоятельной работы

1. Преобразуем программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции  $f(x) = 15x - 9$  как подпрограмму.

```
;Листинг 8.3. Программа вычисления суммы значений функции  $f(x)=15x-9$ 
%include "io_out.asm"
SECTION .data
    msg1 db "Функция:  $f(x)=15x-9$  ",0
    msg2 db "Результат: ",0
SECTION .text
global _start
_start:
    pop esi             ; Извлекаем из стека в esi количество элементов (первое значение в стеке)
    pop ebx             ; Извлекаем из стека в ebx имя подпрограммы (второе значение в стеке)
    sub esi,1           ; Уменьшаем esi на 1 (количество элементов esi указывает программу)
    mov esi, 0          ; Используем esi для хранения промежуточных сумм
next:
    cmp esi,0          ; Проверим, есть ли еще элементы
    jz _end             ; Если элементов нет, выходим из цикла (переход на метку "_end")
    pop ebx             ; Вновь извлекаем очередной элемент из стека
    push ecx            ; Сохраняем ecx
    call atoi           ; Преобразуем символ в число
    push eax            ; Сохраняем eax
    call _calcul         ; Вызов подпрограммы
    add esi, ebx         ; Складываем результат с текущим значением esi
    pop ebx             ; Восстанавливаем ebx
    pop ecx             ; Восстанавливаем ecx
    loop next           ; Переход к следующему следующему элементу
_end:
    mov eax, msg1       ; Вывод первого сообщения
    call sprint
    mov eax, msg2       ; Вывод второго сообщения
    call sprint
    mov ebx, esi        ; Записываем сумму в регистр ebx
    call jprintlf        ; Печата результат
    call quit           ; Завершение программы

; Подпрограмма вычисления значения  $15x-9$ 
_calcul:
    push esi            ; Сохраняем esi на стек
    push ebx            ; Сохраняем ebx на стек
    mov ebx, esp        ; Устанавливаем ebx указателем на начало локальной области
    mov ebx, 15
    mul ebx             ; Умножаем ebx на 15
    sub ebx, 9          ; Вычитаем 9
    pop ebx             ; Восстанавливаем ebx
    pop esi             ; Восстанавливаем esi
    ret
```

Рис. 5.1: Текст программы lab09-4.asm

```

tvbondar@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
tvbondar@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 lab09-4.o -o lab09-4
tvbondar@fedora:~/work/arch-pc/lab09$ ./lab09-4 1 2 4
Функция: f(x)=15x-9 Результат: 78
tvbondar@fedora:~/work/arch-pc/lab09$

```

Рис. 5.2: Запуск программы

2. В листинге 9.3 приведена программа вычисления выражения  $(3 + 2) \cdot 4 + 5$ . При запуске данная программа дает неверный результат. Ошибка в программе заключается в том, что инструкция `mul esx` умножает значение в регистре `eax` на `esx`, а результат записывает в `eax`. В исправленном варианте мы используем `ebx` для хранения промежуточного результата суммы, `mul esx` умножает `ebx` на `esx`, результат сохраняется в `eax`. Затем к результату в `eax` добавляется 5. Финальный результат сохраняется в `edi` и выводится на экран.

Запускаем программу в режиме отладчика и пошагово через `si` просматриваем 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 программа неверно подсчитывает результат функции. Исправляем найденную ошибку, теперь программа верно считает значение функции.

```

--Register group: general--
eax    0x2      2      ecx    0x4      4
edx    0x0      0      ebx    0x5      5
esp    0xffffcf10 0xffffcf10 ebp    0x0      0x0
esi    0x0      0      edi    0x0      0
eip    0x80490f9 0x80490f9 <_start+17> eflags 0x206    [ PF IF ]
cs     0x23     35     ss     0x2b     43
ds     0x2b     43     es     0x2b     43
fs     0x0      0      gs     0x0      0

b+ 0x80490e5 <_start>    mov    50x3,%ebx
0x80490e6 <_start+5>    mov    50x2,%eax
0x80490f2 <_start+10>   add    %eax,%ebx
0x80490f4 <_start+12>   mov    0x1,%ecx
0x80490f9 <_start+17>   mul    %ecx
0x80490fb <_start+19>   add    %eax,%ebx
0x80490fe <_start+22>   mov    %ebx,%edi
0x8049100 <_start+24>   mov    0x8049000,%eax
0x8049105 <_start+29>   call   0x80490ef <sprintf>
0x804910a <_start+34>   mov    %edi,%eax

native process 8526 (asm) In: _start L14 PC: 0x80490f9
eax    0x2      2
ecx    0x4      4
edx    0x0      0
ebx    0x5      5
esp    0xffffcf10 0xffffcf10
ebp    0x0      0x0
esi    0x0      0
edi    0x0      0
eip    0x80490f9 0x80490f9 <_start+17>
eflags 0x206    [ PF IF ]
cs     0x23     35
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 5.3: Поиск ошибки в работе программы lab09-5.asm

```

tvbondar@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-5.lst lab09-5.asm
tvbondar@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab09-5.o
tvbondar@fedora:~/work/arch-pc/lab09$ ./lab09-5
Результат: 25
tvbondar@fedora:~/work/arch-pc/lab09$

```

Рис. 5.4: Запуск исправленной программы

```

;Листинг 9.3. 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

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

```

Рис. 5.5: Исправленный текст программы



## **6 Выводы**

В результате выполнения лабораторной работы я приобрела навыки написания программ с использованием циклов и обработкой аргументов командной строки в NASM.