

# **ОТЧЕТ по лабораторной работе №9**

Полякова Юлия Александровна

# **Содержание**

<b>1 Цель работы</b>	<b>5</b>
<b>2 Результаты выполнения лабораторной работы</b>	<b>6</b>
<b>3 Результаты выполнения заданий для самостоятельной работы</b>	<b>20</b>
<b>4 Вывод</b>	<b>23</b>

# Список иллюстраций

2.1 Создание каталога и lab09-1.asm . . . . .	6
2.2 Запуск lab09-1.asm из листинга 9.1. . . . .	6
2.3 Измененный lab09-1.asm . . . . .	7
2.4 Запуск измененного lab09-1.asm . . . . .	8
2.5 lab09-2.asm . . . . .	8
2.6 Загрузка файла в отладчик . . . . .	9
2.7 Запуск командой run . . . . .	9
2.8 Ставим брейкпоинт . . . . .	9
2.9 Дисассимилированный код . . . . .	10
2.10 Переключаемся на Intel'овский синтаксис . . . . .	10
2.11 Режим псевдографики . . . . .	11
2.12 Окно с регистрами . . . . .	12
2.13 Проверяем информацию о брейкпоинте . . . . .	12
2.14 Брейкпоинт по адресу . . . . .	12
2.15 Первое si . . . . .	13
2.16 Второе si . . . . .	13
2.17 Третье si . . . . .	14
2.18 Четвертое si . . . . .	14
2.19 Пятое si . . . . .	15
2.20 Просмотр значений регистров . . . . .	15
2.21 Смотрим значения переменных . . . . .	15
2.22 Изменения значения регистра . . . . .	16
2.23 Вывод значения . . . . .	16
2.24 Меняем значение . . . . .	17
2.25 Отладка с аргументами программы из лаб. 8 . . . . .	18
2.26 Брейкпоинт и запуск . . . . .	18
2.27 Значения из стека . . . . .	19
3.1 Программа с вычислением в _calc . . . . .	21
3.2 Запуск программы . . . . .	22
3.3 Данная программа . . . . .	22

# **Список таблиц**

# **1 Цель работы**

Приобретение навыков написания программ с использованием подпрограмм.  
Знакомство с методами отладки при помощи GDB и его основными возможностя-  
ми.

## 2 Результаты выполнения лабораторной работы

1. Создаем каталог для выполнения лабораторной работы № 9, переходим в него и создаем файл lab09-1.asm. Вводим в файл lab09-1.asm текст программы из листинга 9.1. (Рис. 1).

uaPolyakova@fedora: ~\$ mkdir ~/work/arch-pc/lab09  
uaPolyakova@fedora: ~\$ cd ~/work/arch-pc/lab09  
uaPolyakova@fedora:~/work/arch-pc/lab09\$ touch lab09-1.asm  
uaPolyakova@fedora:~/work/arch-pc/lab09\$ gedit lab09-1.asm

Листинг 9.1. Пример программы с использованием вызова подпрограмм

```
%include 'in_out.asm'

SECTION .data
msg1 DB 'Введите х: ',0
result1 DB '2x+7= ',0

SECTION .bss
x1 RESB 8
res1 RESB 8

SECTION .text
GLOBAL _start
```

uaPolyakova@fedora:~/work/arch-pc/lab09\$

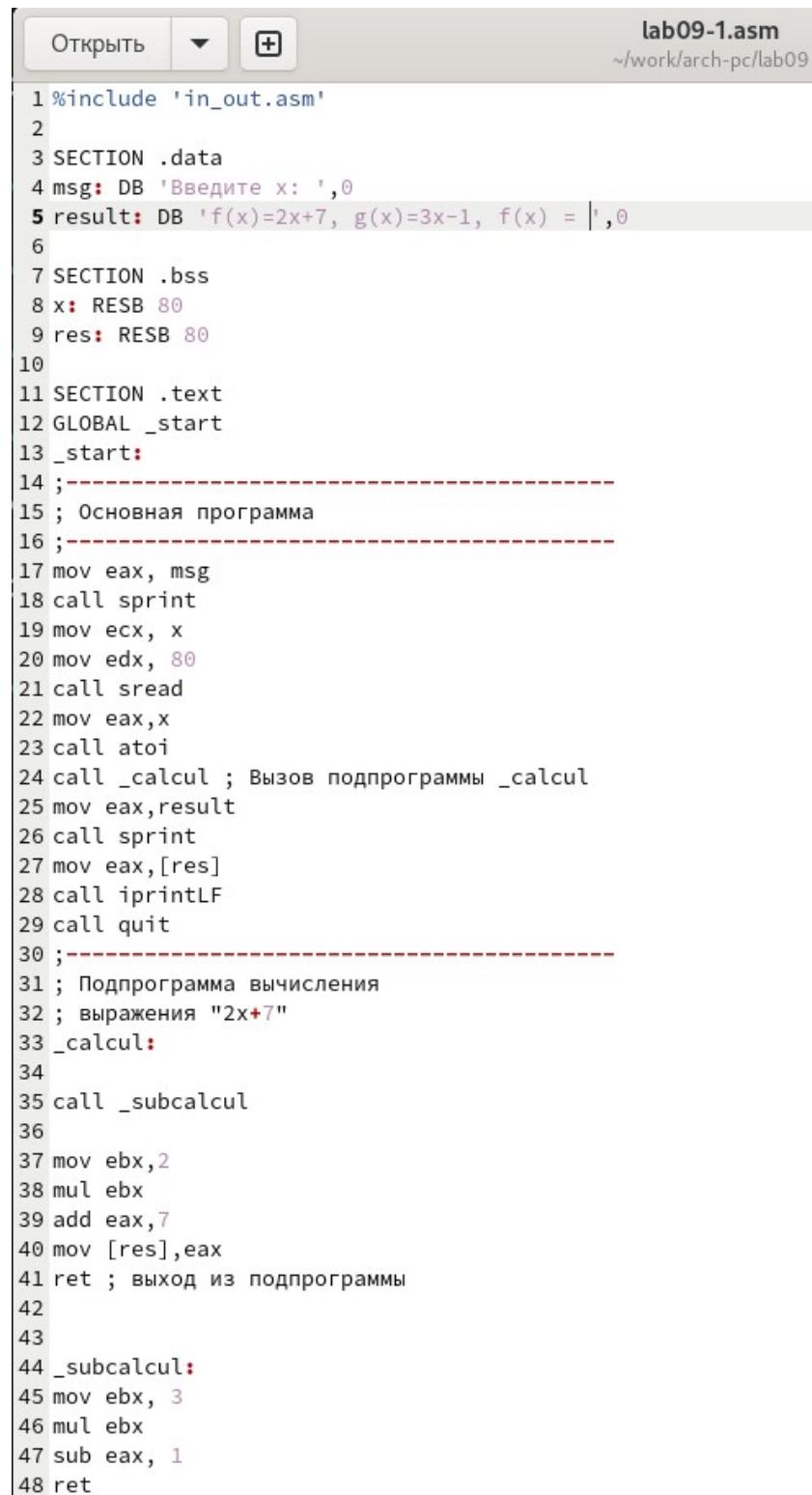
Рис. 2.1: Создание каталога и lab09-1.asm

2. Создаем исполняемый файл и запускаем его (Рис. 2).

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

Рис. 2.2: Запуск lab09-1.asm из листинга 9.1.

3. Изменяем текст программы, чтобы вычисление  $g(x)$  было в \_calcul (Рис. 3).



The screenshot shows a Windows Notepad window with the title bar 'lab09-1.asm' and the path ' ~/work/arch-pc/lab09'. The code is written in assembly language:

```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg: DB 'Введите x: ',0
5 result: DB 'f(x)=2x+7, g(x)=3x-1, f(x) = |',0
6
7 SECTION .bss
8 x: RESB 80
9 res: RESB 80
10
11 SECTION .text
12 GLOBAL _start
13 _start:
14 ;-----
15 ; Основная программа
16 ;-----
17 mov eax, msg
18 call sprint
19 mov ecx, x
20 mov edx, 80
21 call sread
22 mov eax,x
23 call atoi
24 call _calcul ; Вызов подпрограммы _calcul
25 mov eax,result
26 call sprint
27 mov eax,[res]
28 call iprintLF
29 call quit
30 ;-----
31 ; Подпрограмма вычисления
32 ; выражения "2x+7"
33 _calcul:
34
35 call _subcalcul
36
37 mov ebx,2
38 mul ebx
39 add eax,7
40 mov [res],eax
41 ret ; выход из подпрограммы
42
43
44 _subcalcul:
45 mov ebx, 3
46 mul ebx
47 sub eax, 1
48 ret
```

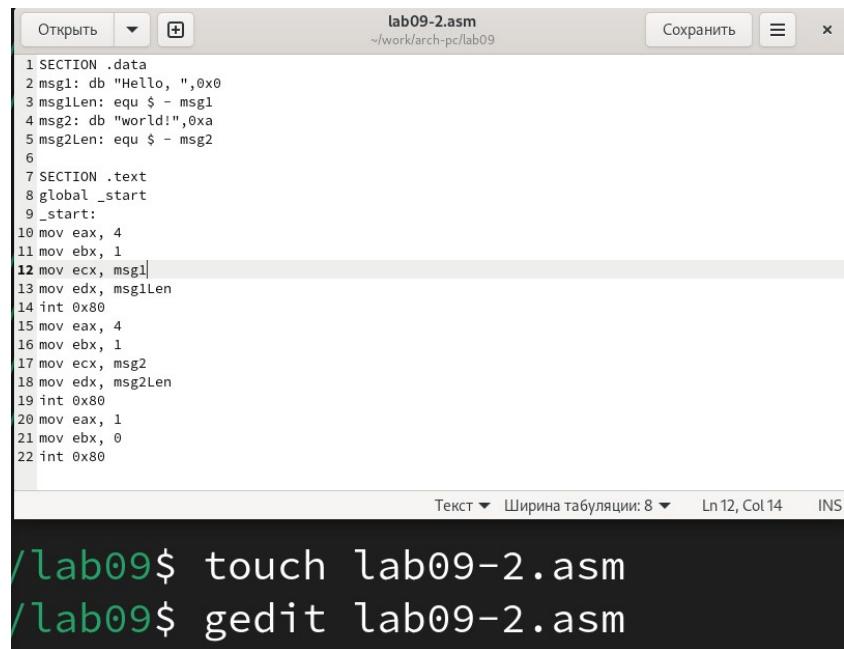
Рис. 2.3: Измененный lab09-1.asm

4. Создаем исполняемый файл и запускаем его (Рис. 4)

```
uapolyakova@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
uapolyakova@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
uapolyakova@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 2
f(x)=2x+7, g(x)=3x-1, f(x) = 17
uapolyakova@fedora:~/work/arch-pc/lab09$
```

Рис. 2.4: Запуск измененного lab09-1.asm

5. Создаем файл lab09-2.asm с текстом программы из Листинга 9.2. (Рис. 5).



```
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 _start:
10 mov eax, 4
11 mov ebx, 1
12 mov ecx, msg1
13 mov edx, msg1Len
14 int 0x80
15 mov eax, 4
16 mov ebx, 1
17 mov ecx, msg2
18 mov edx, msg2Len
19 int 0x80
20 mov eax, 1
21 mov ebx, 0
22 int 0x80
```

Текст ▾ Ширина табуляции: 8 ▾ Ln 12, Col 14 INS

```
/lab09$ touch lab09-2.asm
/lab09$ gedit lab09-2.asm
```

Рис. 2.5: lab09-2.asm

6. Транслируем с аргументом отладки, загружаем файл в отладчик (Рис. 6).

```
uapolyakova@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
uapolyakova@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
uapolyakova@fedora:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 15.2-3.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. 2.6: Загрузка файла в отладчик

## 7. Запускаем программу командой run (Рис. 7).

```
(gdb) run
Starting program: /home/uapolyakova/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.
Download failed: Нет маршрута до узла. Continuing without separate debug info for system-supplied DSO at 0xf7ffc000.
Hello, world!
[Inferior 1 (process 4836) exited normally]
(gdb) 
```

Рис. 2.7: Запуск командой run

## 8. Ставим брейкпоинт на \_start (Рис. 8).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 10.
(gdb) run
Starting program: /home/uapolyakova/work/arch-pc/lab09/lab09-2
Download failed: Нет маршрута до узла. Continuing without separate debug info for system-supplied DSO at 0xf7ffc000.

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

Рис. 2.8: Ставим брейкпоинт

## 9. Смотрим дисассимилированный код программы с помощью команды disassemble начиная с метки \_start (Рис. 9)

```
(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. 2.9: Дисассимилированный код

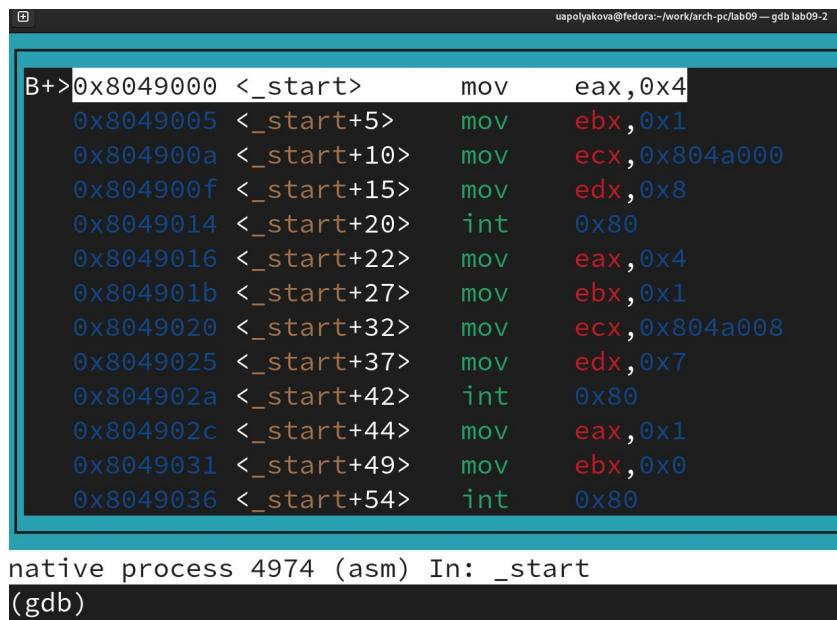
10. Переключаемся на отображение команд с Intel'овским синтаксисом (Рис. 10).

```
(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. 2.10: Переключаемся на Intel'овский синтаксис

В Intel'овском синтаксисе регистр идет первым аргументом и опускаются знаки (\$, %)

11. Включаем режим псевдографики (Рис. 11).



The screenshot shows the GDB assembly dump window. The assembly code is displayed in Intel syntax, where registers are used as arguments. The code starts at address 0x8049000 and includes instructions like mov eax, 0x4, mov ebx, 0x1, etc. The window title is "native process 4974 (asm) In: \_start (gdb)".

```
B+>0x8049000 <_start>    mov    eax, 0x4
    0x8049005 <_start+5>    mov    ebx, 0x1
    0x804900a <_start+10>   mov    ecx, 0x804a000
    0x804900f <_start+15>   mov    edx, 0x8
    0x8049014 <_start+20>   int    0x80
    0x8049016 <_start+22>   mov    eax, 0x4
    0x804901b <_start+27>   mov    ebx, 0x1
    0x8049020 <_start+32>   mov    ecx, 0x804a008
    0x8049025 <_start+37>   mov    edx, 0x7
    0x804902a <_start+42>   int    0x80
    0x804902c <_start+44>   mov    eax, 0x1
    0x8049031 <_start+49>   mov    ebx, 0x0
    0x8049036 <_start+54>   int    0x80

native process 4974 (asm) In: _start
(gdb)
```

Рис. 2.11: Режим псевдографики

12. Включаем окно с регистрами (Рис. 12).

```

Register group: general
eax          0x0          0
ecx          0x0          0
edx          0x0          0
ebx          0x0          0
esp 0xfffffd030 0xfffffd030
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 4974 (asm) In: _start
(gdb) layout regs
(gdb)

```

Рис. 2.12: Окно с регистрами

13. Проверяем информацию о установленном ранее брейкпоинте (Рис. 13).

```

(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. 2.13: Проверяем информацию о брейкпоинте

14. Устанавливаем брейкпойнт по адресу и проверяем его (Рис. 14).

```

(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. 2.14: Брейкпойнт по адресу

15. Первое si (Рис. 15).

```
B+ 0x8049000 <_start>      mov    eax,0x4
px 0x8049005 <_start+5>    mov    ebx,0x1
-1 0x804900a <_start+10>   mov    ecx,0x80
-c 0x804900f <_start+15>   mov    edx,0x8
нг 0x8049014 <_start+20>   int    0x80
gд 0x8049016 <_start+22>   mov    eax,0x4
ут

native process 4974 (asm) In: _start
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm
(gdb) i b
Num      Type            Disp Enb Address
1        breakpoint      keep y  0x08049000
|        breakpoint already hit 1 time
2        breakpoint      keep y  0x08049031
(gdb) si
(gdb) █
```

Рис. 2.15: Первое si

16. Второе si (Рис. 16).

```
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 4974 (asm) In: _start
Breakpoint 2 at 0x8049031: file lab09-2.asm, line
(gdb) i b
Num      Type            Disp Enb Address      What
1        breakpoint      keep y  0x08049000  lab09-
|        breakpoint already hit 1 time
2        breakpoint      keep y  0x08049031  lab09-
(gdb) si
(gdb) si
(gdb) █
```

Рис. 2.16: Второе si

17. Третье si (Рис. 17).

```
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 4974 (asm) In: _start
(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) si
(gdb) si
(gdb) si
(gdb)
```

Рис. 2.17: Третье si

18. Четвертое si (рис. 18).

```
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 4974 (asm) In: _start
Num      Type            Disp Enb Address     What
1        breakpoint       keep y  0x08049000 lab09-
                                breakpoint already hit 1 time
2        breakpoint       keep y  0x08049031 lab09-
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)
```

Рис. 2.18: Четвертое si

19. Пятое si (рис. 19).

```
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 4974 (asm) In: _start
1      breakpoint    keep y  0x08049000 lab09-
      breakpoint already hit 1 time
2      breakpoint    keep y  0x08049031 lab09-
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
```

Рис. 2.19: Пятое si

20. Просматриваем значение всех регистров (Рис. 20)

```
eax          0x8          8
ecx          0x804a000    134520832
edx          0x8          8
ebx          0x1          1
esp          0xfffffd030  0xfffffd030
ebp          0x0          0x0
esi          0x0          0
edi          0x0          0
--Type <RET> for more, q to quit, c to continue without paging--■
```

Рис. 2.20: Просмотр значений регистров

21. Смотрим значение msg1, а msg2 по адресу (Рис. 21)

```
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb)
```

Рис. 2.21: Смотрим значения переменных

22. Изменяем два символа в предыдущих переменных (Рис. 22)

```
(gdb) set {char}&msg1='h'  
(gdb) x/1sb &msg1  
0x804a000 <msg1>:      "hello, "  
(gdb) set {char}&msg2='0'  
(gdb) x/1sb &msg2  
0x804a008 <msg2>:      "Oorlд!\n\034"  
(gdb) █
```

Рис. 2.22: Изменения значения регистра

23. Выводим значение edx в разных форматах (Рис. 23)

```
(gdb) p/x $edx  
$1 = 0x8  
(gdb) p/t $edx  
$2 = 1000  
(gdb) p/s $edx  
$3 = 8  
(gdb)
```

Рис. 2.23: Вывод значения

24. Меняем значение edx (Рис. 24)

```
(gdb) set $ebx='2'  
(gdb) p/s $ebx  
$4 = 50  
(gdb) set $ebx=2  
(gdb) p/s $ebx  
$5 = 2  
(gdb)
```

Рис. 2.24: Меняем значение

В первом случае выводится номер символа, во втором число.

Завершаем работу программы.

25. Копируем lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm. Создаем исполняемый файл и загружаем его в отладчик с аргументами (Рис. 25)

```
uapolyakova@fedora:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
uapolyakova@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
uapolyakova@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
uapolyakova@fedora:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент3'
GNU gdb (Fedora Linux) 15.2-3.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. 2.25: Отладка с аргументами программы из лаб. 8

## 26. Ставим брейкпоинт и запускаем (Рис. 26)

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 6.
(gdb) run
Starting program: /home/uapolyakova/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2
аргумент3

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.
Download failed: Нет маршрута до узла. Continuing without separate debug info for system-supplied DSO at 0xf7ffc000.

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

Рис. 2.26: Брейкпоинт и запуск

## 27. Смотрим значения из стека (Рис. 27)

```
(gdb) x/x $esp
0xfffffcfe0:      0x00000005
(gdb) x/s *(void**)( $esp + 4)
0xfffffd1a9:      "/home/uapolyakova/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)( $esp + 8)
0xfffffd1d6:      "аргумент1"
(gdb) x/s *(void**)( $esp + 12)
0xfffffd1e8:      "аргумент"
(gdb) x/s *(void**)( $esp + 16)
0xfffffd1f0:      "2"
(gdb) x/s *(void**)( $esp + 20)
0xfffffd1fb:      "аргумент3"
(gdb) x/s *(void**)( $esp + 24)
0x0:   <error: Cannot access memory at address 0x0>
(gdb) █
```

Рис. 2.27: Значения из стека

Шаг равен 4, чтобы учесть размер операнда.

### **3 Результаты выполнения заданий для самостоятельной работы**

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

```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg1 db "Функция: f(x)=10x-5",0
5 msg2 db "Результат: ",0
6
7 SECTION .text
8 global _start
9 _start:
10 pop ecx      ; Извлекаем из стека в `ecx` количество
11           ; аргументов (первое значение в стеке)
12 pop edx      ; Извлекаем из стека в `edx` имя программы
13           ; (второе значение в стеке)
14 sub ecx,1    ; Уменьшаем `ecx` на 1 (количество
15           ; аргументов без названия программы)
16 mov esi, 0    ; Используем `esi` для хранения результата
17
18 next:
19 cmp ecx,0h    ; проверяем, есть ли еще аргументы
20 jz _end       ; если аргументов нет выходим из цикла
21           ; (переход на метку `_end`)
22 pop eax      ; иначе извлекаем следующий аргумент из стека
23 call atoi     ; преобразуем символ в число
24
25 call _calcul
26
27 loop next    ; переход к обработке следующего аргумента
28
29 _end:
30 mov eax, msg1 ; вывод сообщения "Функция: f(x)=10x-5"
31 call sprintLF
32 mov eax, msg2 ; вывод сообщения "Результат: "
33 call sprint
34 mov eax, esi  ; записываем результат в регистр `eax`
35 call iprintfLF ; печать результата
36 call quit     ; завершение программы
37
38 _calcul:
39 mov ebx, 10
40 mul ebx
41 sub eax, 5
42 add esi, eax
43 ret
```

Рис. 3.1: Программа с вычислением в \_calc

2. Создаем исполняемый файл и запускаем.

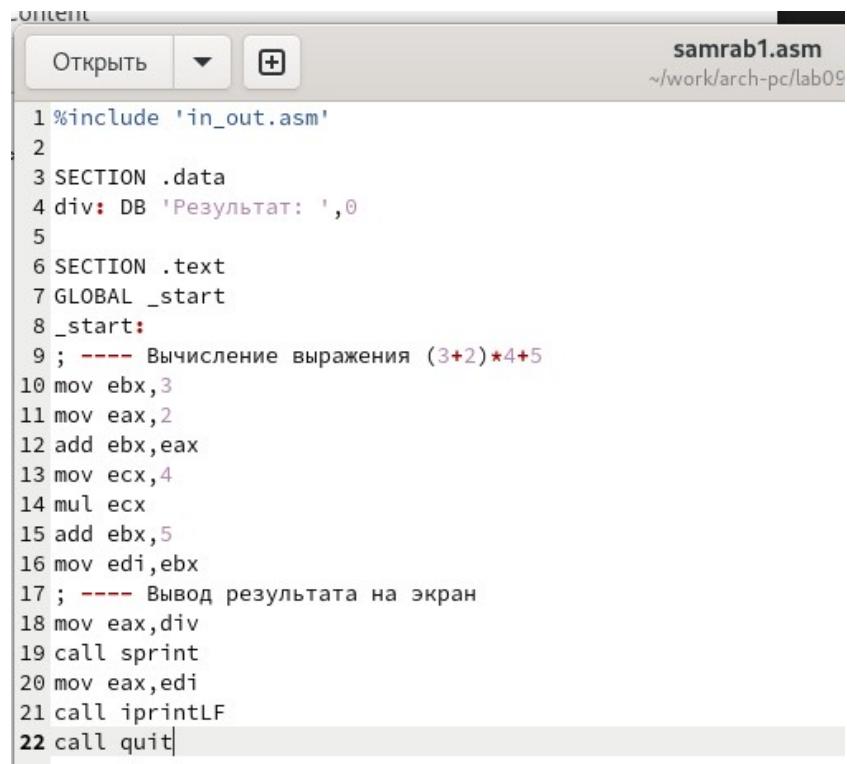
```

uapolyakova@fedora:~/work/arch-pc/lab09$ nasm -f elf samrab.asm
uapolyakova@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o samrab samrab.o
uapolyakova@fedora:~/work/arch-pc/lab09$ ./samrab 1 2 3
Функция: f(x)=10x-5
Результат: 45
uapolyakova@fedora:~/work/arch-pc/lab09$ 

```

Рис. 3.2: Запуск программы

3. Проводим отладку данной в условии программы.



The screenshot shows the QEMU debugger interface with the assembly code for `samrab1.asm`. The code is as follows:

```

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

```

Рис. 3.3: Данная программа

Ошибка в том, что `mul` увеличивает `eax` на `ecx`, и этот результат никуда не записывается. Мы все время работаем с `ebx` и получается  $3 + 2 + 5 = 10$ .

## **4 Вывод**

Были приобретены навыки написания программ с использованием подпрограмм. Были изучены методы отладки при помощи GDB и его основные возможности.