

**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук**

**Кафедра прикладной информатики и теории вероятностей**

**ОТЧЕТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2**

*дисциплина:* Архитектура компьютера

Студент: Безходарнова А.В

Группа: НКАбд-01-25

**МОСКВА**

2025 г.

## СОДЕРЖАНИЕ

1.Цель работы .....	3
2. Выполнение лабораторной работы.....	4
2.1 Реализация подпрограмм в NASM .....	4
2.2 Отладка программ с помощью GDB .....	7
2.3 Добавление точек останова .....	13
2.4 Работа с данными программы в GDB .....	15
2.5 Обработка аргументов командной строки в GDB.....	18
3. Задание для самостоятельной работы.....	20
4. Выводы .....	23

## **1.Цель работы**

Целью данной работы является приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

## 2. Выполнение лабораторной работы

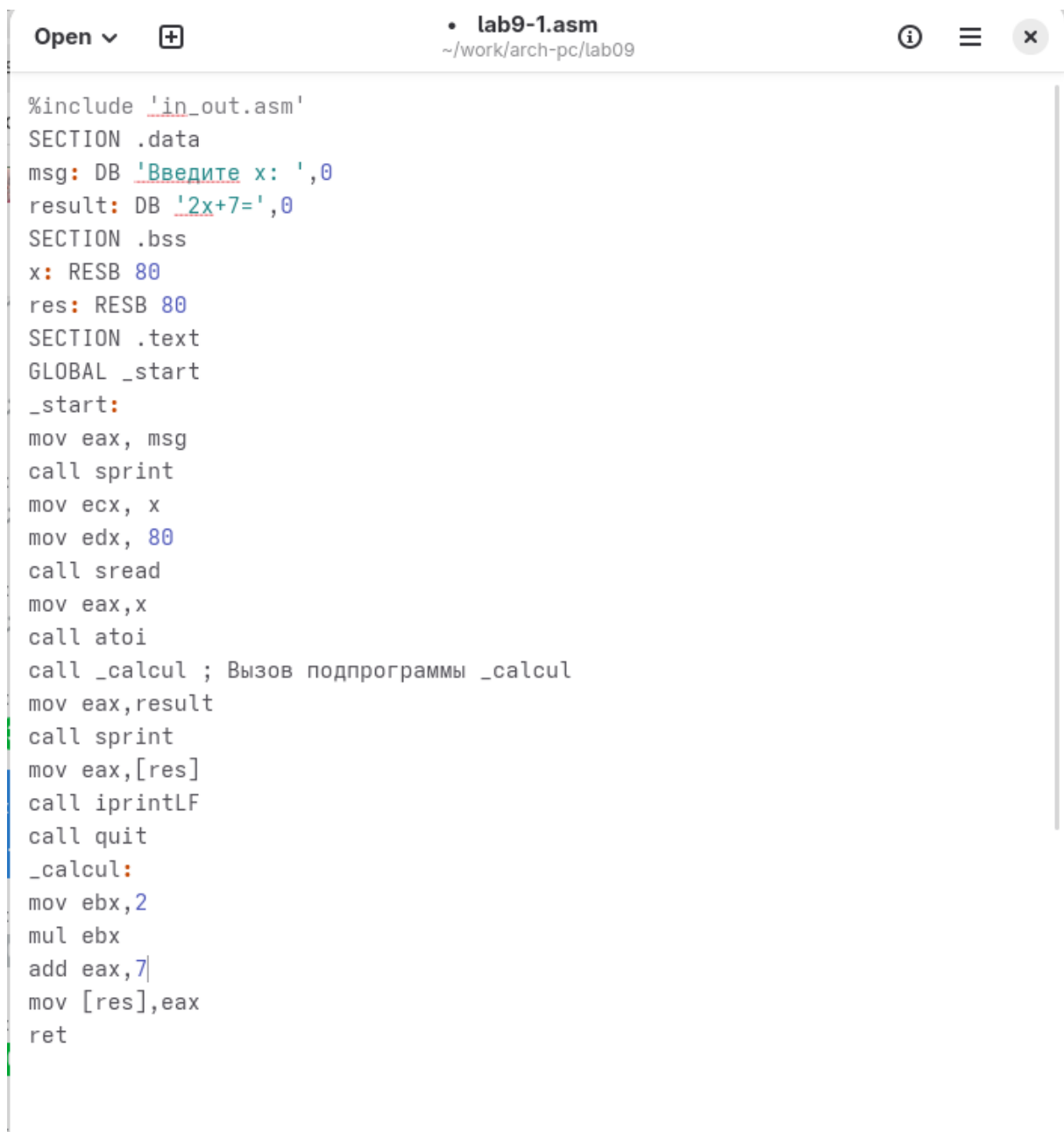
### 2.1 Реализация подпрограмм в NASM

Создаю каталог для программ лабораторной работы №9, перейдя в него создаю файл lab9-1.asm:.(рис.2.1.1)

```
avbezhodarnova1@avbezhodarnova1:~$ mkdir ~/work/arch-pc/lab09
avbezhodarnova1@avbezhodarnova1:~$ cd ~/work/arch-pc/lab09
avbezhodarnova1@avbezhodarnova1:~/work/arch-pc/lab09$ touch lab9-1.asm
avbezhodarnova1@avbezhodarnova1:~/work/arch-pc/lab09$
```

Рис. 2.1.1. Создание файла

В созданном файле ввожу программу из листинга. (рис.2.1.3)



```
Open ▾ [+]  
• lab9-1.asm  
~/work/arch-pc/lab09  
%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:  
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  
_calcul:  
mov ebx, 2  
mul ebx  
add eax, 7  
mov [res], eax  
ret
```

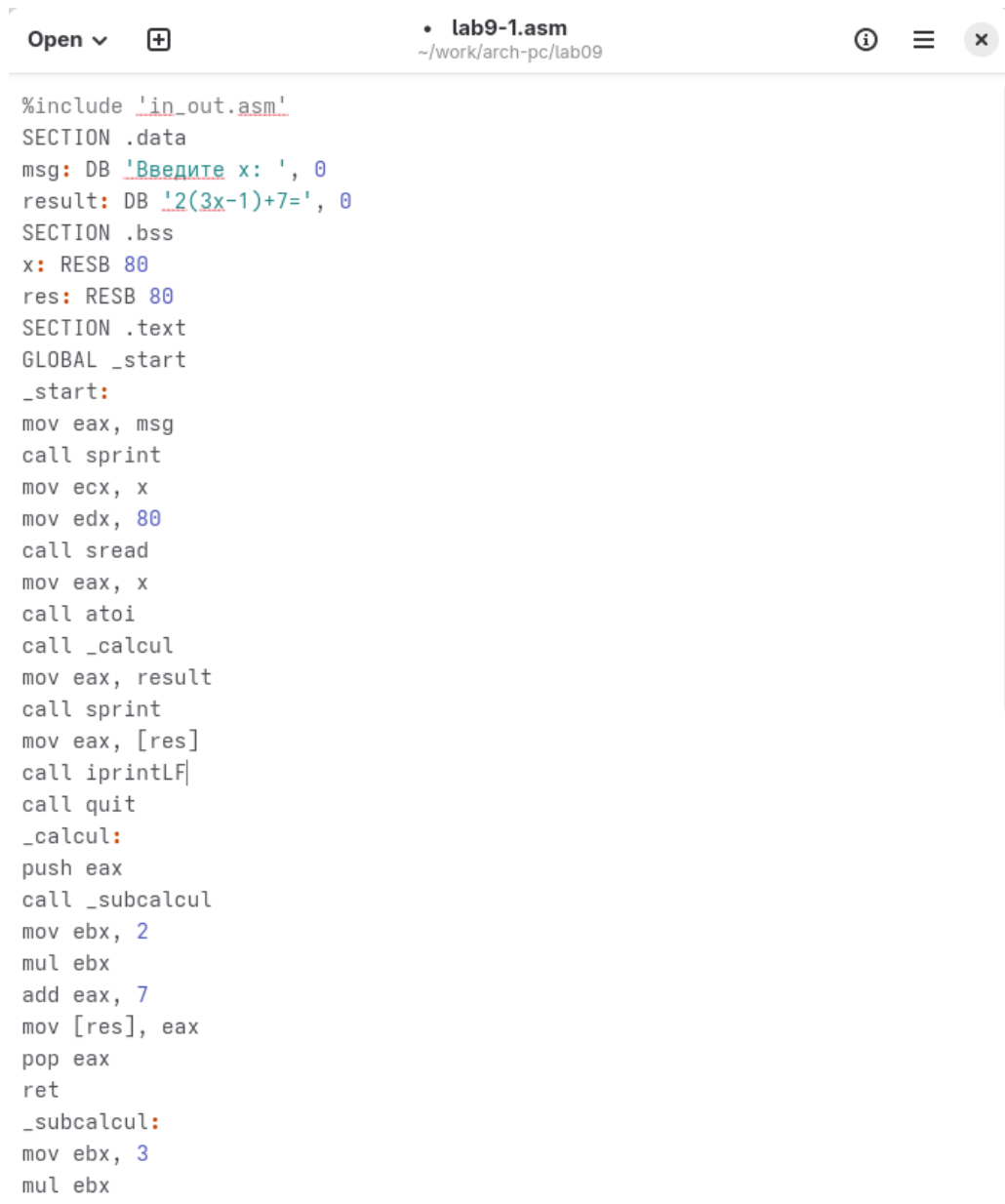
Рис. 2.1.2 Загрузка текста из листинга

Далее запускаю исходный код, который вычисляет значение функции.  
(рис.2.1.3)

```
avbezhodarnova1@avbezhodarnova1:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
avbezhodarnova1@avbezhodarnova1:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
ld: cannot find lab9-1.o: No such file or directory
avbezhodarnova1@avbezhodarnova1:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
avbezhodarnova1@avbezhodarnova1:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 5
2x+7=17
avbezhodarnova1@avbezhodarnova1:~/work/arch-pc/lab09$
```

Рис. 2.1.3 Запуск программы

Изменяю текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения  $f(g(x))$ , где  $x$  вводится с клавиатуры,  $f(x) = 2x + 7$ ,  $g(x) = 3x - 1$ . (Рис.2.1.4) и (рис.2.1.5)




```
• lab9-1.asm
~/work/arch-pc/lab09

Open ▾ ⊕ ⓘ ≡ ✕

#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
```

Рис.2.1.4 Изменение текста

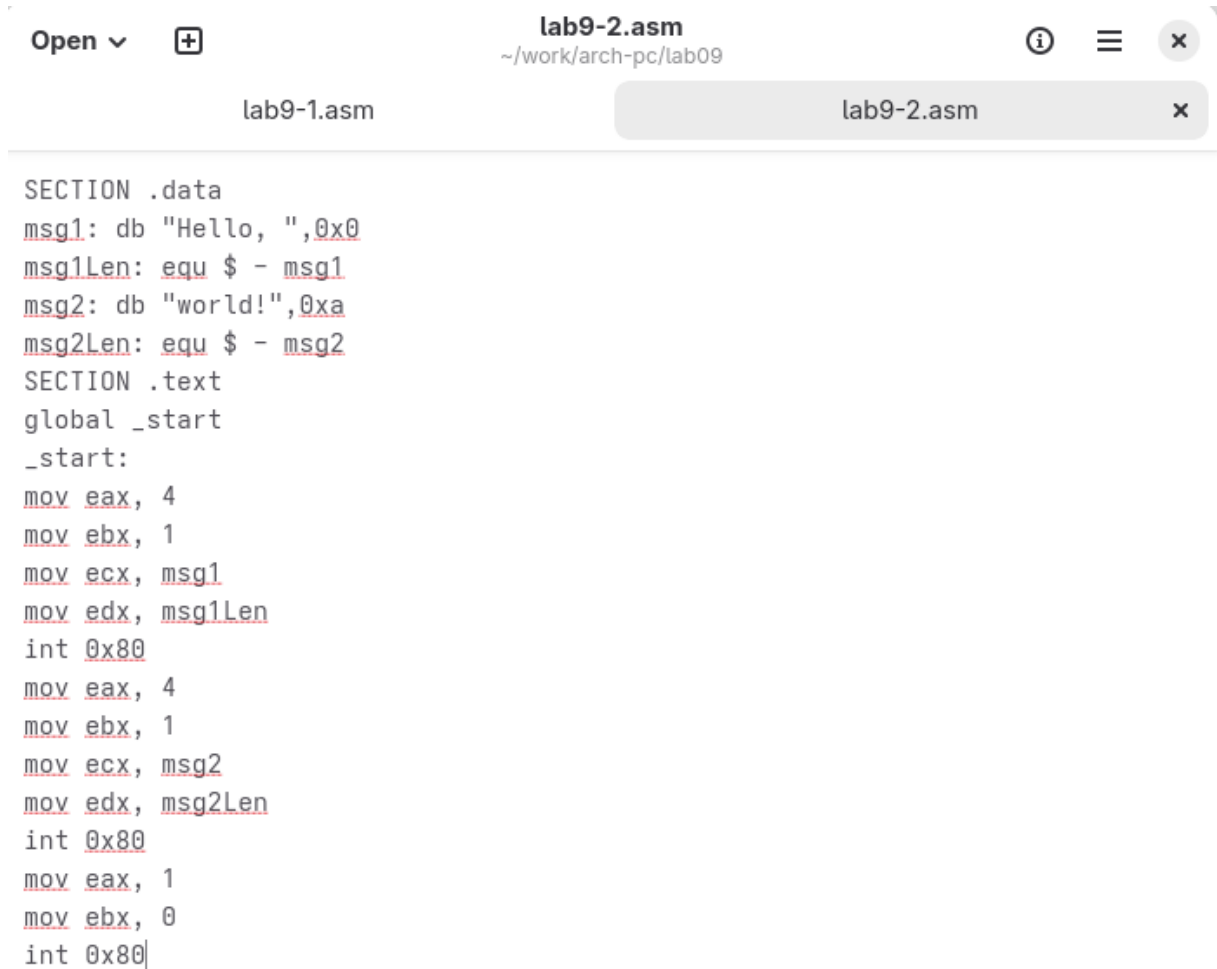


```
avbezkhodarnova1@avbezkhodarnova1:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
avbezkhodarnova1@avbezkhodarnova1:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
avbezkhodarnova1@avbezkhodarnova1:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 5
2(3x-1)+7=35
avbezkhodarnova1@avbezkhodarnova1:~/work/arch-pc/lab09$
```

Рис.2.1.5 Запуск программы

## 2.2 Отладка программ с помощью GDB

Создаю новый файл и ввожу в него код из листинга. (рис. 2.2.1)



```
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.2.2.1 Ввод кода из листинга.

Запускаю программу и получаю исполняемый файл. (Рис.2.2.2)

```

avbezhkodarnova1@avbezhkodarnova1:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
avbezhkodarnova1@avbezhkodarnova1:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
avbezhkodarnova1@avbezhkodarnova1:~/work/arch-pc/lab09$ gdb lab9-2
GNU gdb (Fedora Linux) 16.2-3.fc42
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 lab9-2...
(gdb) █

```

Рис. 2.2.2 Запуск программы

Проверяю работу программы, запуская ее с помощью run. (Рис.2.2.3)

```

avbezhkodarnova1@avbezhkodarnova1:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2
avbezhkodarnova1@avbezhkodarnova1:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
avbezhkodarnova1@avbezhkodarnova1:~/work/arch-pc/lab09$ gdb lab9-2
GNU gdb (Fedora Linux) 16.2-3.fc42
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 lab9-2...
(gdb) run
Starting program: /home/avbezhkodarnova1/work/arch-pc/lab09/lab9-2

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 7351) exited normally]
(gdb) █

```

Рис. 2.2.3 Работа программы.

Далее ставлю брейкпоинт для корректной работы программы и запускаю. (Рис.2.2.4)



```

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 lab9-2...
(gdb) run
Starting program: /home/avbezhodarnova1/work/arch-pc/lab09/lab9-2

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 7351) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8048080: file lab9-2.asm, line 9.
(gdb) run
Starting program: /home/avbezhodarnova1/work/arch-pc/lab09/lab9-2

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

```

Рис.2.2.4 Запуск программы.

Далее я смотрю дисассимилированный код программы с помощью команды `disassemble`, а после переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду `setdisassembly-flavor intel`. (рис.2.2.5) 1.

Различия между синтаксисом АТТ и Intel заключаются в следующем: в синтаксисе АТ&Т первым указывается исходный операнд, а вторым — целевой. Например, инструкция `addl %eax, %ebx` означает сложение содержимого регистра EAX с регистром EBX и сохранение результата в EBX. В синтаксисе Intel порядок обратный: сначала указывается операнд-назначение, затем операнд-источник. Та же операция записывается как `add`

ebx, eax. В АТТ размер обрабатываемых данных задаётся явно через суффиксы в мнемонике команды. В Intel размер данных определяется неявно — по используемым регистрам или явным указанием типа (например, byte ptr, dword ptr). В АТТ непосредственные значения предваряются символом \$. Например, `movl $100, %eax` загружает число 100 в регистр EAX. В Intel непосредственные значения записываются без дополнительных символов: `mov eax, 100`. В АТТ для обращения к памяти используются круглые скобки: `(%eax)`. В Intel адресация памяти обозначается квадратными скобками: `[eax]`, а сложные адреса имеют вид `[база + индекс * множитель + смещение]`.

```

(gdb) run
Starting program: /home/avbezhodarnova1/work/arch-pc/lab09/lab9
Breakpoint 1, _start () at lab9-2.asm:9
9      mov eax, 4
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08048080 <+0>:      mov     $0x4,%eax
      0x08048085 <+5>:      mov     $0x1,%ebx
      0x0804808a <+10>:     mov     $0x8049000,%ecx
      0x0804808f <+15>:     mov     $0x8,%edx
      0x08048094 <+20>:     int     $0x80
      0x08048096 <+22>:     mov     $0x4,%eax
      0x0804809b <+27>:     mov     $0x1,%ebx
      0x080480a0 <+32>:     mov     $0x8049008,%ecx
      0x080480a5 <+37>:     mov     $0x7,%edx
      0x080480aa <+42>:     int     $0x80
      0x080480ac <+44>:     mov     $0x1,%eax
      0x080480b1 <+49>:     mov     $0x0,%ebx
      0x080480b6 <+54>:     int     $0x80
End of assembler dump.
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08048080 <+0>:      mov     eax,0x4
      0x08048085 <+5>:      mov     ebx,0x1
      0x0804808a <+10>:     mov     ecx,0x8049000
      0x0804808f <+15>:     mov     edx,0x8
      0x08048094 <+20>:     int     0x80
      0x08048096 <+22>:     mov     eax,0x4
      0x0804809b <+27>:     mov     ebx,0x1
      0x080480a0 <+32>:     mov     ecx,0x8049008
      0x080480a5 <+37>:     mov     edx,0x7

```

Рис.2.2.5 Изменение текста листинга.

Включаю режим псевдографики. (Рис.2.2.6)

```
avbezkhodarnova1@localhost-live:~/work/arch-pc/lab09 — gdb lab9-2
~/work/arch-pc/lab09

Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffcf50 0xffffcf50  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8048080 0x8048080 <_start>  eflags    0x202    [ IF ]
cs       0x23     35     ss       0x2b     43
ds       0x2b     43     es       0x2b     43
fs       0x0      0      gs       0x0      0

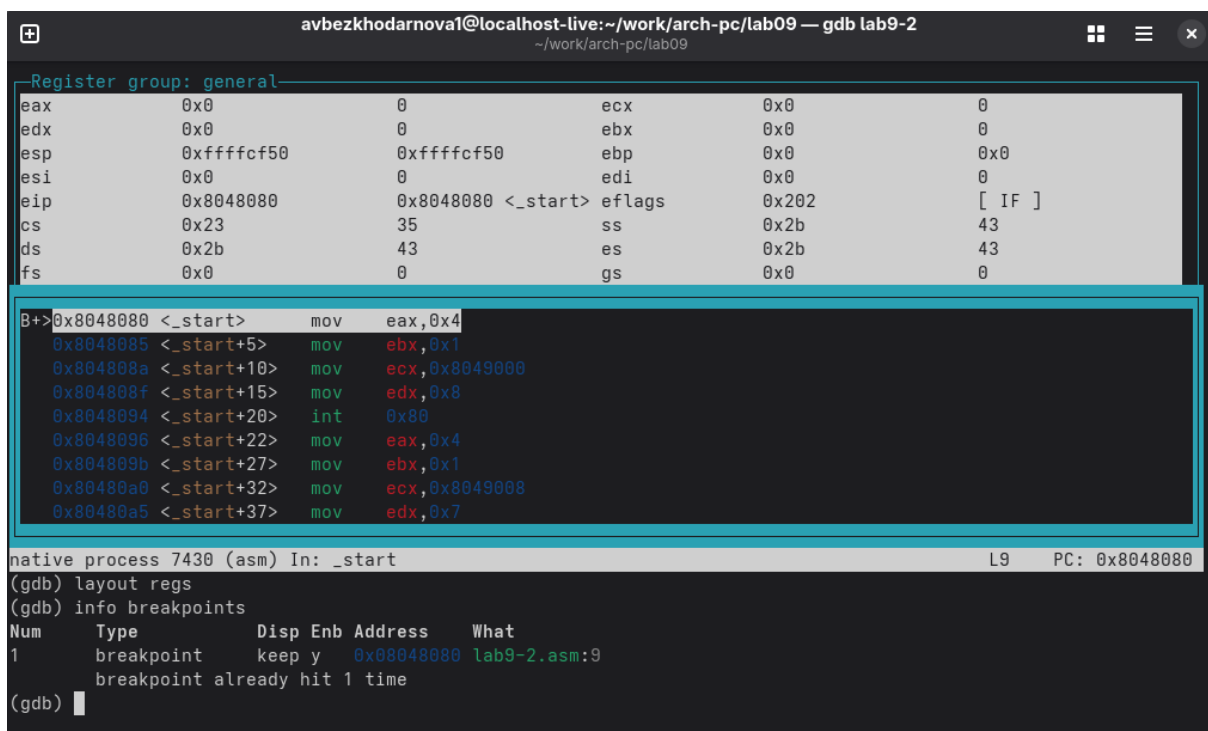
B+>0x8048080 <_start> mov eax,0x4
0x8048085 <_start+5> mov ebx,0x1
0x804808a <_start+10> mov ecx,0x8049000
0x804808f <_start+15> mov edx,0x8
0x8048094 <_start+20> int 0x80
0x8048096 <_start+22> mov eax,0x4
0x804809b <_start+27> mov ebx,0x1
0x80480a0 <_start+32> mov ecx,0x8049008
0x80480a5 <_start+37> mov edx,0x7

native process 7430 (asm) In: _start L9 PC: 0x8048080
(gdb) layout regs
(gdb)
```

Рис.2.2.6 Режим псевдографики.

## 2.3 Добавление точек останова

Проверяю, установлена ли точка останова с помощью команды `info breakpoints`. (Рис.2.3.1)



```
avbezkhodarnova1@localhost-live:~/work/arch-pc/lab09 — gdb lab9-2
~/work/arch-pc/lab09

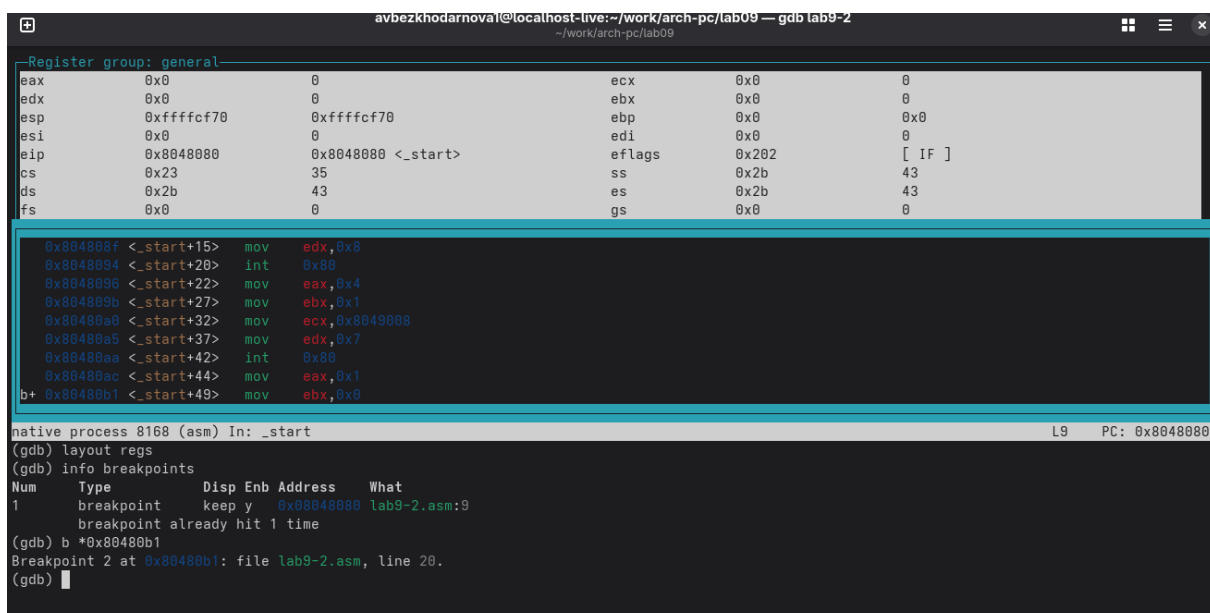
Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffcf50  0xffffcf50  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8048080  0x8048080 <_start>  eflags    0x202      [ IF ]
cs       0x23     35      ss       0x2b     43
ds       0x2b     43      es       0x2b     43
fs       0x0      0      gs       0x0      0

B+>0x8048080 <_start> mov eax,0x4
0x8048085 <_start+5> mov ebx,0x1
0x804808a <_start+10> mov ecx,0x8049000
0x804808f <_start+15> mov edx,0x8
0x8048094 <_start+20> int 0x80
0x8048096 <_start+22> mov eax,0x4
0x804809b <_start+27> mov ebx,0x1
0x80480a0 <_start+32> mov ecx,0x8049008
0x80480a5 <_start+37> mov edx,0x7

native process 7430 (asm) In: _start L9 PC: 0x8048080
(gdb) layout regs
(gdb) info breakpoints
Num      Type      Disp Enb Address      What
1        breakpoint keep y 0x8048080 lab9-2.asm:9
        breakpoint already hit 1 time
(gdb)
```

Рис.2.3.1 Проверка установки

Далее устанавливаю еще одну точку останова по адресу. Для этого смотрю, адрес инструкции и задаю точку останова. (рис.2.3.2)



```
avbezkhodarnova1@localhost-live:~/work/arch-pc/lab09 — gdb lab9-2
~/work/arch-pc/lab09

Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffcf70  0xffffcf70  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8048080  0x8048080 <_start>  eflags    0x202      [ IF ]
cs       0x23     35      ss       0x2b     43
ds       0x2b     43      es       0x2b     43
fs       0x0      0      gs       0x0      0

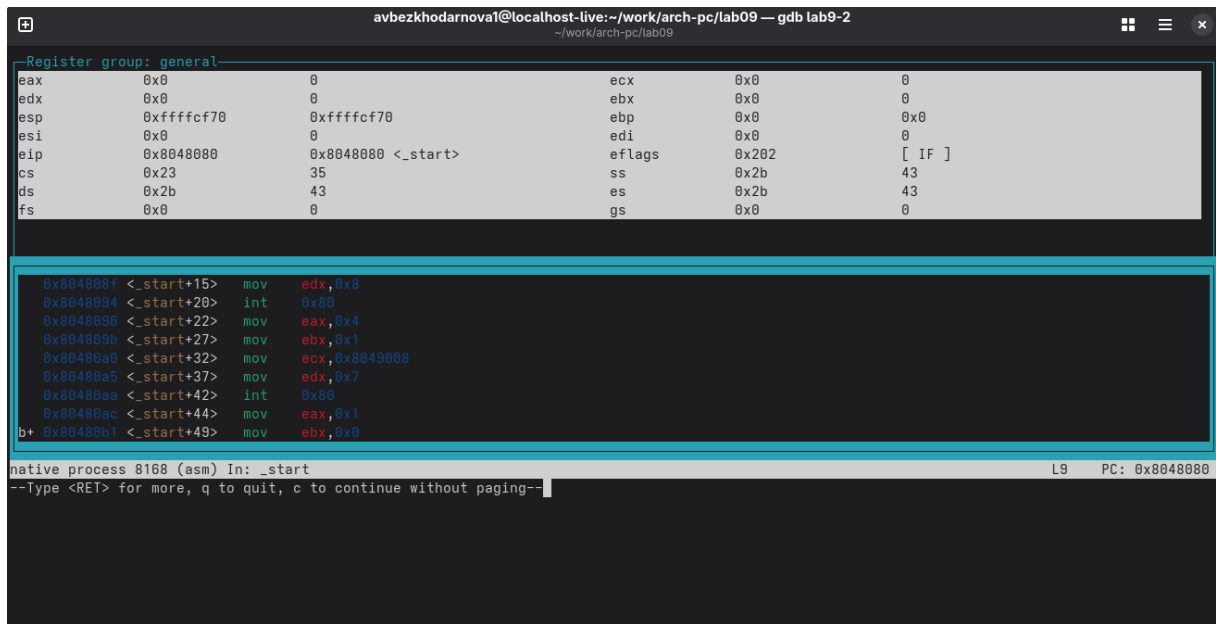
0x804808f <_start+15> mov edx,0x8
0x8048094 <_start+20> int 0x80
0x8048096 <_start+22> mov eax,0x4
0x804809b <_start+27> mov ebx,0x1
0x80480a0 <_start+32> mov ecx,0x8049008
0x80480a5 <_start+37> mov edx,0x7
0x80480aa <_start+42> int 0x80
0x80480ac <_start+44> mov eax,0x1
b+ 0x80480b1 <_start+49> mov ebx,0x0

native process 8168 (asm) In: _start L9 PC: 0x8048080
(gdb) layout regs
(gdb) info breakpoints
Num      Type      Disp Enb Address      What
1        breakpoint keep y 0x8048080 lab9-2.asm:9
        breakpoint already hit 1 time
(gdb) b *0x80480b1
Breakpoint 2 at 0x80480b1: file lab9-2.asm, line 20.
(gdb)
```

Рис.2.3.2 задача точек останова

## 2.4 Работа с данными программы в GDB

Смотрю содержимое регистров с помощью команды. (рис.2.4.1)



The screenshot shows the GDB interface with the 'Register group: general' window. The registers and their values are as follows:

Register	Value
eax	0x0
edx	0x0
esp	0xffffcf70
esi	0x0
eip	0x8048080
cs	0x23
ds	0x2b
fs	0x0
ecx	0x0
ebx	0x0
ebp	0xffffcf70
edi	0x0
eflags	0x202 [ IF ]
ss	0x2b
es	0x2b
gs	0x0

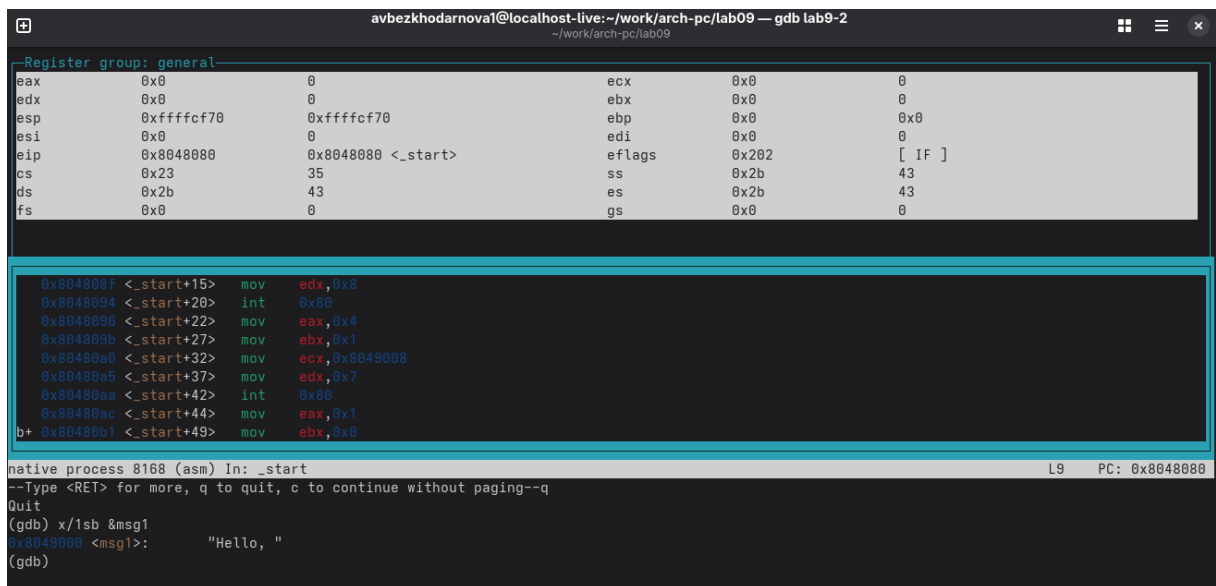
Below the register window, the assembly code is displayed:

```
0x804808f <_start+15> mov    edx,0x8
0x8048094 <_start+20> int    0x80
0x8048096 <_start+22> mov    eax,0x4
0x804809b <_start+27> mov    ebx,0x1
0x80480a0 <_start+32> mov    ecx,0x8049008
0x80480a5 <_start+37> mov    edx,0x7
0x80480aa <_start+42> int    0x80
0x80480ac <_start+44> mov    eax,0x1
b+ 0x80480b1 <_start+49> mov    ebx,0x0
```

The status bar at the bottom indicates: native process 8168 (asm) In: \_start L9 PC: 0x8048080.

Рис.2.4.1 просмотр регистров

Смотрю содержимое переменной. (Рис.2.4.2)



The screenshot shows the GDB interface with the 'Register group: general' window. The registers and their values are as follows:

Register	Value
eax	0x0
edx	0x0
esp	0xffffcf70
esi	0x0
eip	0x8048080
cs	0x23
ds	0x2b
fs	0x0
ecx	0x0
ebx	0x0
ebp	0xffffcf70
edi	0x0
eflags	0x202 [ IF ]
ss	0x2b
es	0x2b
gs	0x0

Below the register window, the assembly code is displayed:

```
0x804808f <_start+15> mov    edx,0x8
0x8048094 <_start+20> int    0x80
0x8048096 <_start+22> mov    eax,0x4
0x804809b <_start+27> mov    ebx,0x1
0x80480a0 <_start+32> mov    ecx,0x8049008
0x80480a5 <_start+37> mov    edx,0x7
0x80480aa <_start+42> int    0x80
0x80480ac <_start+44> mov    eax,0x1
b+ 0x80480b1 <_start+49> mov    ebx,0x0
```

The status bar at the bottom indicates: native process 8168 (asm) In: \_start L9 PC: 0x8048080.

Below the assembly code, the command prompt shows the following commands and output:

```
--Type <RET> for more, q to quit, c to continue without paging--
Quit
(gdb) x/1sb &msg1
0x8049008 <msg1>: "Hello, "
(gdb)
```

Рис.2.4.2 просмотр содержимого переменной

Далее я смотрю адрес переменной, но теперь определяю Адрес переменной можно по дизассемблированной инструкции. (Рис.2.4.3)

The screenshot shows the GDB interface with the 'Register group: general' expanded. It lists registers like eax, edx, esp, esi, eip, cs, ds, fs and their values. Below, assembly code is displayed for the range 0x804808f to 0x80480b1. The code includes instructions like 'mov', 'int', and 'mov' with their operands. At the bottom, the native process 8168 (asm) is shown with the instruction pointer at 0x8048080.

```

Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffcf70 0xffffcf70  ebp      0x0      0
esi      0x0      0      edi      0x0      0
eip      0x8048080 0x8048080 <_start>  eflags    0x202    [ IF ]
cs       0x23     35     ss       0x2b     43
ds       0x2b     43     es       0x2b     43
fs       0x0      0      gs       0x0      0

0x804808f <_start+15> mov    edx,0x8
0x8048094 <_start+20> int    0x80
0x8048096 <_start+22> mov    eax,0x4
0x804809b <_start+27> mov    ebx,0x1
0x80480a0 <_start+32> mov    ecx,0x8049008
0x80480a5 <_start+37> mov    edx,0x7
0x80480aa <_start+42> int    0x80
0x80480ac <_start+44> mov    eax,0x1
b+ 0x80480b1 <_start+49> mov    ebx,0x0

native process 8168 (asm) In: _start
--Type <RET> for more, q to quit, c to continue without paging--q
Quit
(gdb) x/1sb &msg1
0x8049000 <msg1>:      "Hello, "
(gdb) x/1sb 0x8049008
0x8049008 <msg2>:      "world!\n\034"
(gdb)

```

Рис.2.4.3 Просмотр содержимого переменной.

Изменяю значение для регистра или ячейки памяти с помощью команды set. (рис.2.4.4)

The screenshot shows the GDB interface where the user uses the 'set' command to modify the content of the 'msg1' variable. The initial state shows 'Hello, ' and after the command 'set {char}msg1='h'', it changes to 'hello, '.

```

native process 8168 (asm) In: _start
--Type <RET> for more, q to quit, c to continue without paging--q
Quit
(gdb) x/1sb &msg1
0x8049000 <msg1>:      "Hello, "
(gdb) x/1sb 0x8049008
0x8049008 <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
0x8049000 <msg1>:      "hello, "
(gdb)

```

Рис.2.4.4 Изменение значение регистра.

Меняю символ в переменной msg2. (Рис.2.4.5)

The screenshot shows the GDB interface where the user uses the 'set' command to change the first character of the 'msg2' variable from 'w' to 'a'. The string changes from 'world!' to 'aorld!'.

```

(gdb) set {char}&msg2='a'
(gdb) x/1sb &msg2
0x8049008 <msg2>:      "aorld!\n\034"
(gdb)

```

Рис.2.4.5 Изменение символа в переменной

Вывожу в различных форматах значение регистра edx. (Рис.2.4.6)



```

(gdb) p/x $edx
$1 = 0x0
(gdb) p/t $edx
$2 = 0
(gdb) p/c $edx
$3 = 0 '\000'
(gdb)

```

Рис.2.4.6 Значения регистра

Далее с помощью команды `set` изменяю значения регистра `ebx`.  
(Рис.2.4.7) и (Рис.2.4.8)

```

(gdb) set $ebx='2'
(gdb) p/s ebx
No symbol "ebx" in current context.
(gdb) p/s $ebx
$4 = 50
(gdb)

```

Рис.2.4.7 Изменение регистра

```

$5 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$6 = 2
(gdb)

```

Рис.2.4.8 Изменение регистра

Кавычки `'2'` дают код символа (50), а просто `2` даёт число 2. Команда `p/s` показывает число, а не символ.

## 2.5 Обработка аргументов командной строки в GDB

Копирую файл из лабораторной работы №8 и создаю исполняемый файл, а также задаю аргументы, использую ключ. (Рис.2.5.1)

```
avbezkhodarnova1@avbezkhodarnova1:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
avbezkhodarnova1@avbezkhodarnova1:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
avbezkhodarnova1@avbezkhodarnova1:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
avbezkhodarnova1@avbezkhodarnova1:~/work/arch-pc/lab09$ gdb --args lab09-3 arg1 arg2 'arg3'
GNU gdb (Fedora Linux) 16.2-3.fc42
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.2.5.1 Копирование файла.

Далее устанавливаю точку останова перед первой инструкцией в программе и запускаю ее. (Рис.2.5.2)

```
GNU gdb (Fedora Linux) 16.2-3.fc42
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) b _start
Breakpoint 1 at 0x8048148: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/avbezkhodarnova1/work/arch-pc/lab09/lab09-3 arg1 arg2 arg3

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:5
5      pop ecx
(gdb) █
```

Рис.2.5.2 Запуск программы

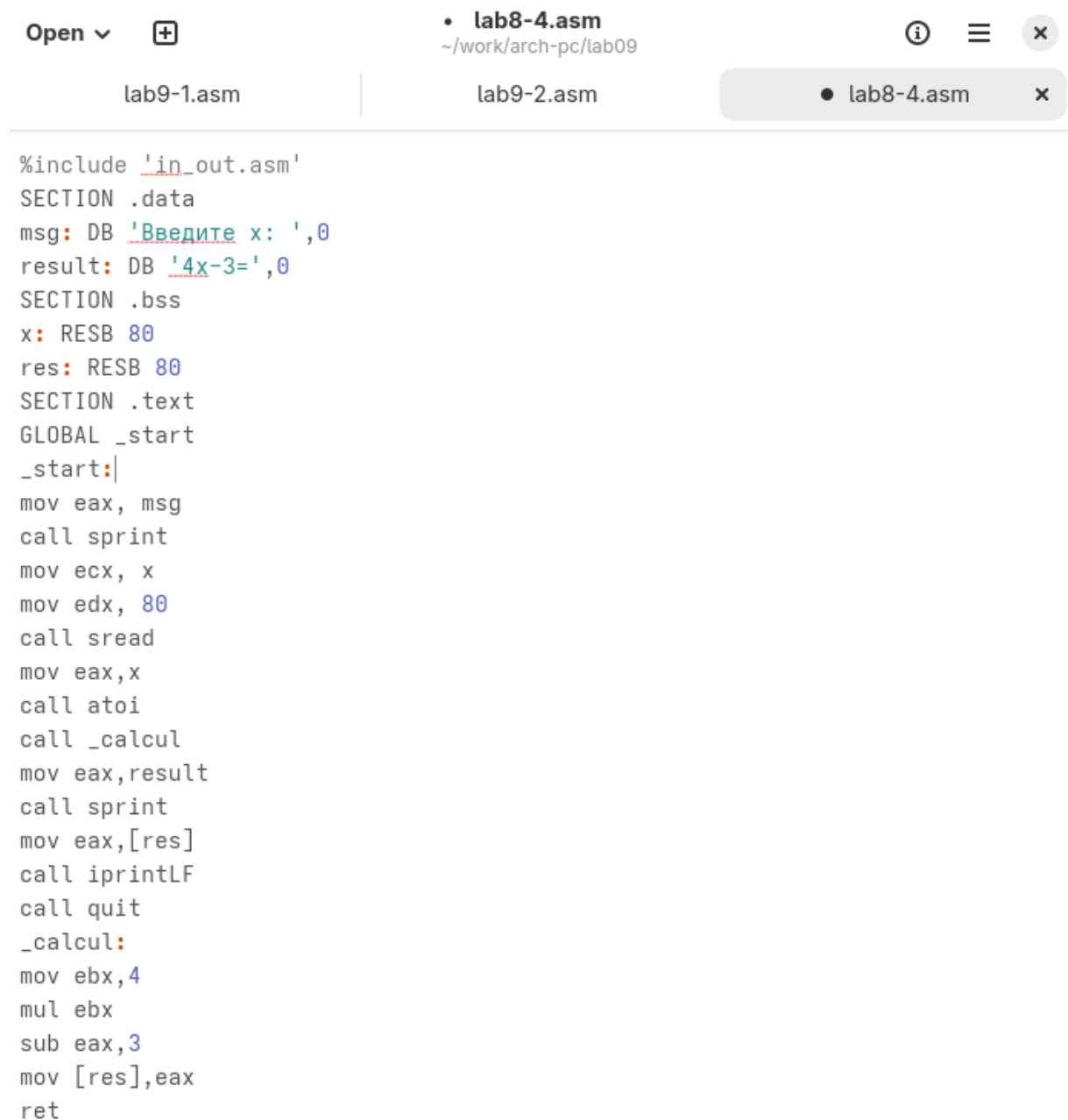
Смотрю остальные позиции стека – по адресу [esp+4] располагается адрес в памяти где находится имя программы, по адресу [esp+8] храниться адрес первого аргумента, по адресу [esp+12] – второго и т.д. (Рис.2.5.3)

```
0xffffcf50: 0x00000004
(gdb) x/x $esp+4
0xffffcf54: 0xffffd12e
(gdb) x/x $esp+8
0xffffcf58: 0xffffd160
(gdb) x/x $esp+12
0xffffcf5c: 0xffffd165
(gdb) x/s *(void**)(esp+4)
0xffffd12e: "/home/avbezhkodarnova1/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp+8)
0xffffd160: "arg1"
(gdb) x/s *(void**)(esp+12)
0xffffd165: "arg2"
(gdb) x/s *(void**)(esp+16)
0xffffd16a: "arg3"
(gdb) x/s *(void**)(esp+20)
0x0: <error: Cannot access memory at address 0x0>
(gdb) x/s *(void**)(esp+24)
0xffffd16f: "SHELL=/bin/bash"
(gdb)
```

Рис.2.5.3 Проверка работы

### 3. Задание для самостоятельной работы

Преобразую программу из лабораторной работы №8, реализовав вычисление значения функции  $f(x)$  как подпрограмму. (Рис.3.1)



```
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '4x-3=',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:
mov ebx, 4
mul ebx
sub eax, 3
mov [res], eax
ret
```

Рис.3.1 Редактирование файла.

Далее я загружаю программу вычисления, с помощью отладчика нахожу ошибки и исправляю их. (Рис.3.2), (Рис.3.3), (Рис.3.4), (Рис.3.5)

```

avbezhodarnova1@localhost-live:~/work/arch-pc/lab09 — gdb lab9-5
~/work/arch-pc/lab09

0x8048159 <atoi.restore+3> pop    ebx
0x804815a <atoi.restore+4> ret
0x804815b <quit> mov    ebx,0x0
0x8048160 <quit+5> mov    eax,0x1
0x8048165 <quit+10> int    0x80
0x8048167 <quit+12> ret
B+ 0x8048168 <_start> mov    ebx,0x3
0x804816d <_start+5> mov    eax,0x2
0x8048172 <_start+10> add    ebx,eax
0x8048174 <_start+12> mov    ecx,0x4
>0x8048179 <_start+17> mul    ecx
0x804817b <_start+19> add    ebx,0x5
0x804817e <_start+22> mov    edi,ebx
0x8048180 <_start+24> mov    eax,0x049000
0x8048185 <_start+29> call   0x804800f <sprint>
0x804818a <_start+34> mov    eax,edi
0x804818c <_start+36> call   0x8048106 <printf@plt>
0x8048191 <_start+41> call   0x804815b <quit>
0x8048196 add    BYTE PTR [eax],al
0x8048198 add    BYTE PTR [eax],al

native process 10901 (asm) In: _start L11 PC: 0x8048179
eax 0x2 2
ecx 0x4 4
edx 0x0 0
ebx 0x5 5
esp 0xffffcf70 0xffffcf70
ebp 0x0 0
esi 0x0 0
edi 0x0 0
eip 0x8048179 0x8048179 <_start+17>
eflags 0x206 [ PF IF ]
cs 0x23 35

```

Рис.3.2 Запуск программы.

```

avbezhodarnova1@localhost-live:~/work/arch-pc/lab09 — gdb lab9-5
~/work/arch-pc/lab09

--Register group: general--
eax 0x8 8 ecx 0x4 4
edx 0x0 0 ebx 0x5 5
esp 0xffffcf70 0xffffcf70 ebp 0x0 0x0
esi 0x0 0 edi 0x0 0
eip 0x804817b 0x804817b <_start+19> eflags 0x206 [ PF IF ]
cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43
fs 0x0 0 gs 0x0 0

0x8048160 <quit+5> mov    eax,0x1
0x8048165 <quit+10> int    0x80
0x8048167 <quit+12> ret
B+ 0x8048168 <_start> mov    ebx,0x3
0x804816d <_start+5> mov    eax,0x2
0x8048172 <_start+10> add    ebx,eax
0x8048174 <_start+12> mov    ecx,0x4
0x8048179 <_start+17> mul    ecx
>0x804817b <_start+19> add    ebx,0x5
0x804817e <_start+22> mov    edi,ebx

native process 10901 (asm) In: _start L12 PC: 0x804817b
edx 0x0 0
ebx 0x5 5
esp 0xffffcf70 0xffffcf70
ebp 0x0 0x0
esi 0x0 0
edi 0x0 0
eip 0x804817b 0x804817b <_start+19>
eflags 0x206 [ PF IF ]
cs 0x23 35
--Type <RET> for more, q to quit, c to continue without paging--q
Quit
(gdb)

```

Рис.3.3



Рис.3.4

```
avbezhodarnova1@avbezhodarnova1:~/work/arch-pc/lab09$ nasm -f elf lab9-5.asm
avbezhodarnova1@avbezhodarnova1:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-5 lab9-5.o
avbezhodarnova1@avbezhodarnova1:~/work/arch-pc/lab09$ ./lab9-5
Результат: 25
avbezhodarnova1@avbezhodarnova1:~/work/arch-pc/lab09$
```

Рис.3.5

#### **4. Выводы**

В ходе выполнения данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм и также познакомилась с методами отладки при помощи GDB и его основными возможностями.