

Лабораторная работа №9.

Понятие подпрограммы. Отладчик GDB.

Жукова София Викторовна

Содержание

Цель работы	6
Выполнение лабораторной работы	7
Отладка программ с помощью GDB	9
Выводы	23

Список иллюстраций

1	Создаем каталог и файл	7
2	Заполняем файл	8
3	Запускаем файл и проверяем его работу	8
4	Изменяем файл	9
5	Запускаем файл и смотрим на его работу	9
6	Создаем файл	10
7	Заполняем файл	10
8	Загружаем исходный файл в отладчик	11
9	Запускаем программу командой run	11
10	Запускаем программу с брейкпоинтом	11
11	Смотрим дисассимилированный код программы	12
12	Переключаемся на синтаксис Intel	12
13	Включаем отображение регистров, их значений и результат дисассимилирования программы	13
14	Используем команду info breakpoints и создаем новую точку останова	14
15	Смотрим информацию	14
16	Отслеживаем регистры	15
17	Смотрим значение переменной	15
18	Смотрим значение переменной	15
19	Меняем символ	16
20	Меняем символ	16
21	Смотрим значение регистра	16
22	Изменяем регистр командой set	17
23	Прописываем команды с и quit	17
24	Копируем файл	17
25	Создаем и запускаем в отладчике файл	18
26	Устанавливаем точку останова	18
27	Изучаем полученные данные	18
28	Копируем файл	19
29	Изменяем файл	19
30	Проверяем работу программы	20
31	Создаем файл	20
32	Изменяем файл	20
33	Создаем и смотрим на работу программы(работает неправильно)	21
34	Ищем ошибку регистров в отладчике	21
35	Меняем файл	22

36	Создаем и запускаем файл	22
----	------------------------------------	----

Список таблиц

Цель работы

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

Выполнение лабораторной работы

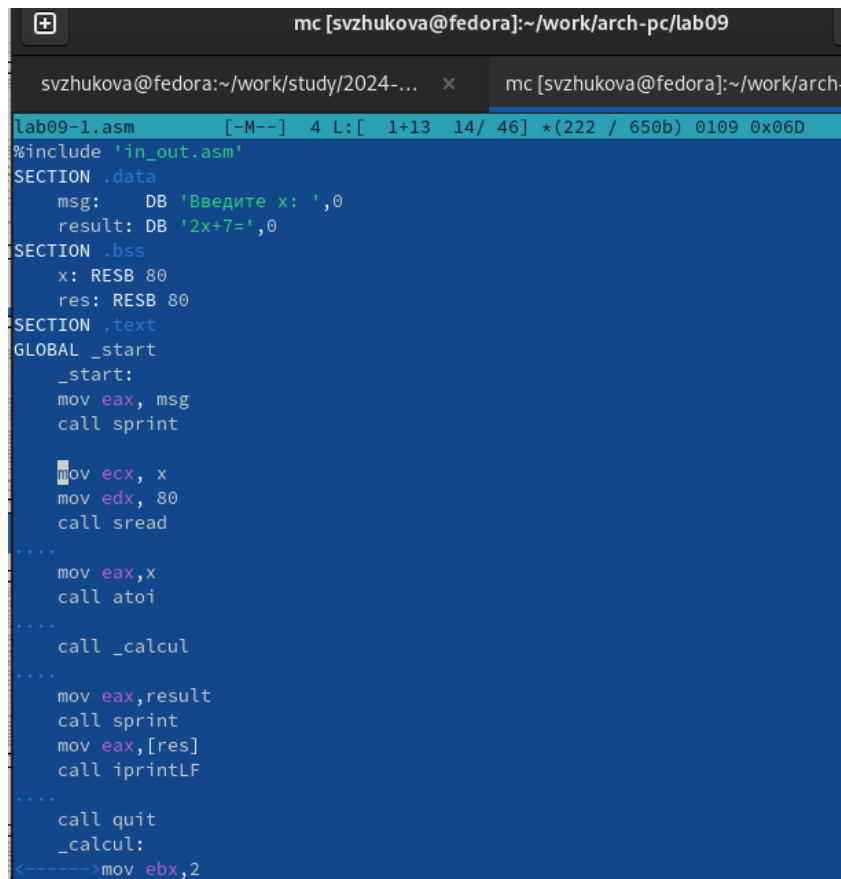
1. Реализация подпрограмм в NASM

Создадим каталог для выполнения лабораторной работы № 9, перейдем в него и создадим файл lab09-1.asm: (рис. [-@fig:001]).

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

Рис. 1: Создаем каталог и файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.1 (рис. [-@fig:002]).



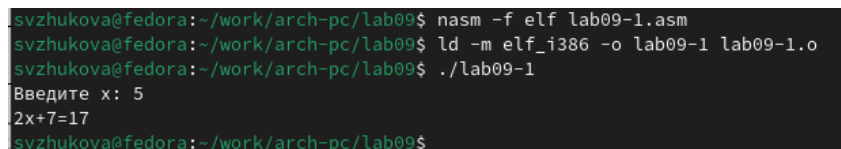
```
mc [svzhukova@fedora]:~/work/arch-pc/lab09
svzhukova@fedora:~/work/study/2024-... x mc [svzhukova@fedora]:~/work/arch-
lab09-1.asm [-M--] 4 L: [ 1+13 14/ 46] *(222 / 650b) 0109 0x06D
#include 'in_out.asm'
SECTION .data
    msg: DB 'Введите x: ',0
    result: DB '2x+7=',0
SECTION .bss
    x: RESB 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
    ....
    mov eax,result
    call sprint
    mov eax,[res]
    call iprintLF
    ....
    call quit
    _calcul:
<----->mov ebx,2
```

Рис. 2: Заполняем файл

Создаем исполняемый файл и запускаем его (рис. [-@fig:003]).



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

Рис. 3: Запускаем файл и проверяем его работу

Снова открываем файл для редактирования и изменяем его, добавив подпрограмму в подпрограмму (рис. [-@fig:004]).


```
lab09-1.asm [----] 12 L: [ 3+14 17/ 35] *(291 / 552b) 0105 0x06
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:
<-----> call _subcalcul
<-----> mov ebx, 2
<-----> mul ebx
<-----> add eax, 7
<-----> mov [res], eax
<-----> ret
<-----> _subcalcul:
<-----> mov ebx, 3
<-----> mul ebx
```

Рис. 4: Изменяем файл

Создаем исполняемый файл и запускаем его (рис. @fig:005).

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

Рис. 5: Запускаем файл и смотрим на его работу

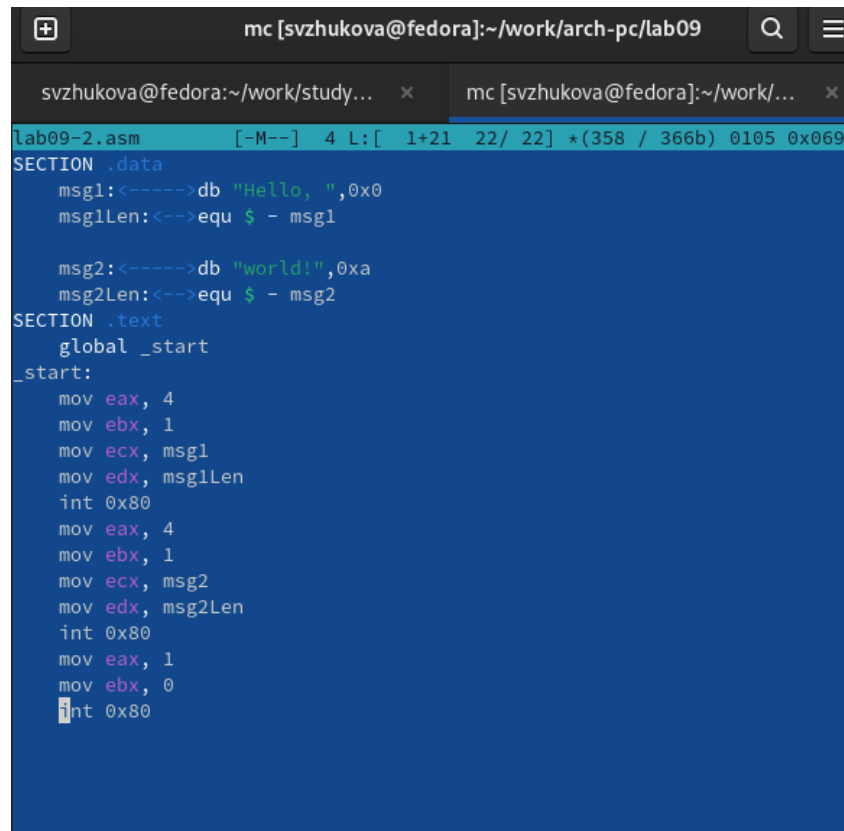
Отладка программ с помощью GDB

Создаем новый файл в каталоге(рис. @fig:006).

```
svzhukova@fedora:~/work/arch-pc/lab09$ touch lab09-2.asm
svzhukova@fedora:~/work/arch-pc/lab09$
```

Рис. 6: Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.2 (рис. @fig:007).



```
mc [svzhukova@fedora]:~/work/arch-pc/lab09
lab09-2.asm [-M--] 4 L: [ 1+21 22/ 22] *(358 / 366b) 0105 0x069
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. 7: Заполняем файл

Получаем исходный файл с использованием отладчика gdb (рис. @fig:008).

```
svzhukova@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
svzhukova@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
svzhukova@fedora:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 15.2-1.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...
```

Рис. 8: Загружаем исходный файл в отладчик

Запускаем команду в отладчике (рис. @fig:009).

```
(gdb) r
Starting program: /home/svzhukova/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 47.71 K separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 8189) exited normally]
(gdb)
```

Рис. 9: Запускаем программу командой run

Устанавливаем брейкпоинт на метку _start и запускаем программу (рис. @fig:010).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 10.
(gdb) run
Starting program: /home/svzhukova/work/arch-pc/lab09/lab09-2

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

Рис. 10: Запускаем программу с брейкпоинтом

Смотрим дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start`(рис. @fig:011).

```
(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. 11: Смотрим дисассимилированный код программы

Переключаемся на отображение команд с Intel'овским синтаксисом (рис. @fig:012).

```
(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. 12: Переключаемся на синтаксис Intel

Различия отображения синтаксиса машинных команд в режимах АТТ и Intel:

1.Порядок операндов: В АТТ синтаксисе порядок операндов обратный, сначала указывается исходный операнд, а затем - результирующий операнд. В Intel синтаксисе порядок обычно прямой, результирующий операнд указывается первым, а исходный - вторым.

2.Разделители: В АТТ синтаксисе разделители операндов - запятые. В Intel синтаксисе разделители могут быть запятые или косые черты (/).

3.Префиксы размера операндов: В АТТ синтаксисе размер операнда указывается перед операндом с использованием префиксов, таких как “b” (byte), “w” (word), “l” (long) и “q” (quadword). В Intel синтаксисе размер операнда указывается после операнда с использованием суффиксов, таких как “b”, “w”, “d” и “q”.

4.Знак операндов: В АТТ синтаксисе операнды с позитивными значениями предваряются символом “*.Intel*”.

5.Обозначение адресов: В АТТ синтаксисе адреса указываются в круглых скобках. В Intel синтаксисе адреса указываются без скобок.

6.Обозначение регистров: В АТТ синтаксисе обозначение регистра начинается с символа “%”. В Intel синтаксисе обозначение регистра может начинаться с символа “R” или “E” (например, “%eax” или “RAX”).

Включаем режим псевдографики (рис. @fig:013).

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd050 0xffffd050
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>

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

native process 8537 (asm) In: _start      L10  PC: 0x8049000
(gdb) layout regs
(gdb) █
```

Рис. 13: Включаем отображение регистров, их значений и результат дисассимилирования программы

Проверяем была ли установлена точка останова и устанавливаем точку остано-

ва предпоследней инструкции (рис. @fig:014).

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd050 0xffffd050
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>

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

native process 8537 (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. 14: Используем команду info breakpoints и создаем новую точку останова

Посмотрим информацию о всех установленных точках останова (рис. @fig:015).

```
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y 0x08049000 lab09-2.asm:10
breakpoint already hit 1 time
(gdb)
```

Рис. 15: Смотрим информацию

Выполняем 5 инструкций командой si (рис. @fig:016).

```

eax          0x0          0
ecx          0x0          0
edx          0x0          0
ebx          0x0          0
esp          0xffffd050    0xffffd050
ebp          0x0          0x0
esi          0x0          0
edi          0x0          0
eip          0x8049005      0x8049005 <_start+5>

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

native process 8537 (asm) In: _start          L11    PC: 0x8049005
eip          0x8049000      0x8049000 <_start>
eflags       0x202          [ IF ]
--Type <RET> for more, q to quit, c to continue without paging--
cs           0x23           35
ss           0x2b           43
ds           0x2b           43
es           0x2b           43
fs           0x0            0
gs           0x0            0
(gdb) si

```

Рис. 16: Отслеживаем регистры

Во время выполнения команд менялись регистры: ebx, ecx, edx, eax, eip. Смотрим значение переменной msg1 по имени (рис. @fig:017).

```

(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb)

```

Рис. 17: Смотрим значение переменной

Смотрим значение переменной msg2 по адресу (рис. @fig:018).

```

(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb)

```

Рис. 18: Смотрим значение переменной

Изменим первый символ переменной msg1 (рис. @fig:019).

```
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb)
```

Рис. 19: Меняем символ

Изменим первый символ переменной msg2 (рис. @fig:020).

```
(gdb) set {char}&msg2='L'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "Lorld!\n\034"
(gdb)
```

Рис. 20: Меняем символ

Смотрим значение регистра edx в разных форматах (рис. @fig:021).

```
(gdb) p/t $edx
$1 = 0
(gdb) p/s $edx
$2 = 0
(gdb) p/x $edx
$3 = 0x0
(gdb)
```

Рис. 21: Смотрим значение регистра

Изменяем регистр ebx (рис. @fig:022).


```

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

```

Рис. 22: Изменяем регистр командой set

Выводятся разные значения, так как команда без кавычек присваивает регистру вводимое значение.

Прописываем команды для завершения программы и выхода из GDB (рис. @fig:023).

```

(gdb) c
Continuing.
World!

Breakpoint 2, _start () at lab09-2.asm:20
(gdb)

```

Рис. 23: Прописываем команды c и quit

3. Обработка аргументов командной строки в GDB

Копируем файл lab8-2.asm в файл с именем lab09-3.asm (рис. @fig:024).

```

svzhukova@fedora:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
svzhukova@fedora:~/work/arch-pc/lab09$

```

Рис. 24: Копируем файл

Создаем исполняемый файл и запускаем его в отладчике GDB (рис. @fig:025).

```
svzhukova@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
svzhukova@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
svzhukova@fedora:~/work/arch-pc/lab09$ gdb --args lab09-3 2 3 '5'
```

Рис. 25: Создаем и запускаем в отладчике файл

Установим точку останова перед первой инструкцией в программе и запустим ее (рис. @fig:026).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) r
Starting program: /home/svzhukova/work/arch-pc/lab09/lab09-3 2 3 5

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) x/x $esp
0xffffd040:    0x00000004
(gdb)
```

Рис. 26: Устанавливаем точку останова

Смотрим позиции стека по разным адресам (рис. @fig:027).

```
(gdb) x/x $esp
0xffffd040:    0x00000004
(gdb) x/s *(void**)(esp + 4)
0xffffd206:    "/home/svzhukova/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd231:    "2"
(gdb) x/s *(void**)(esp + 12)
0xffffd233:    "3"
(gdb) x/s *(void**)(esp + 16)
0xffffd235:    "5"
(gdb) x/s *(void**)(esp + 20)
0x0:    <error: Cannot access memory at address 0x0>
(gdb) x/s *(void**)(esp + 24)
0xffffd237:    "SHELL=/bin/bash"
(gdb)
```

Рис. 27: Изучаем полученные данные

Шаг изменения адреса равен 4 потому что адресные регистры имеют размерность 32 бита(4 байта).

##Задание для самостоятельной работы

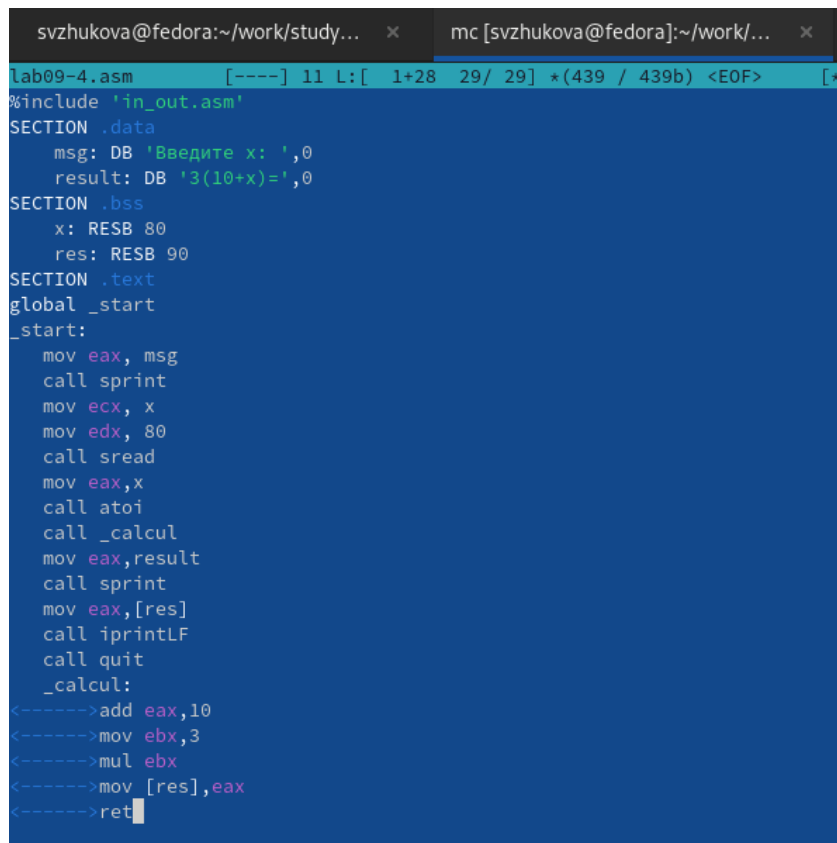
Задание 1

Копируем файл lab8-4.asm(ср №1 в ЛБ8) в файл с именем lab09-3.asm (рис. @fig:028).

```
svzhukova@fedora:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-4.asm ~/work/arch-pc/lab09/lab09-4.asm
svzhukova@fedora:~/work/arch-pc/lab09$
```

Рис. 28: Копируем файл

Открываем файл в Midnight Commander и меняем его, создавая подпрограмму (рис. @fig:029).



```
svzhukova@fedora:~/work/study... x mc [svzhukova@fedora]:~/work/... x
lab09-4.asm [----] 11 L: [ 1+28 29/ 29] *(439 / 439b) <EOF> [*]
#include 'in_out.asm'
SECTION .data
    msg: DB 'Введите x: ',0
    result: DB '3(10+x)=',0
SECTION .bss
    x: RESB 80
    res: RESB 90
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:
<----->add eax,10
<----->mov ebx,3
<----->mul ebx
<----->mov [res],eax
<----->ret
```

Рис. 29: Изменяем файл

Создаем исполняемый файл и запускаем его (рис. @fig:030).

```
svzhukova@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
svzhukova@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
svzhukova@fedora:~/work/arch-pc/lab09$ ./lab09-4
Введите x: 5
3(10+x)=45
svzhukova@fedora:~/work/arch-pc/lab09$
```

Рис. 30: Проверяем работу программы

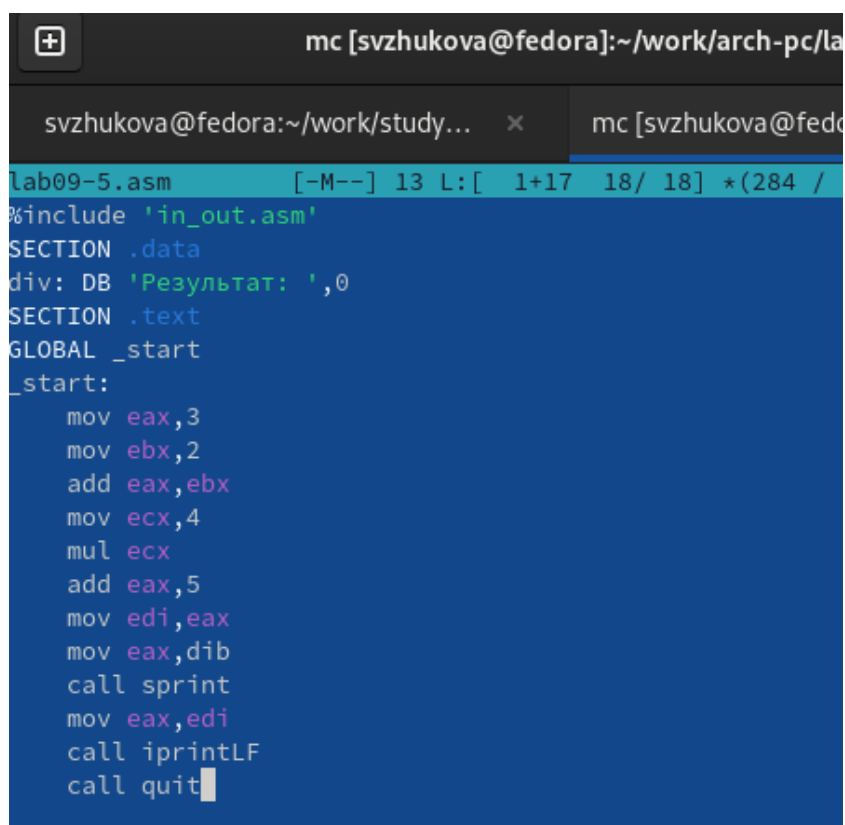
###Задание 2

Создаем новый файл в дирректории (рис. @fig:031).

```
svzhukova@fedora:~/work/arch-pc/lab09$ touch lab09-5.asm
svzhukova@fedora:~/work/arch-pc/lab09$
```

Рис. 31: Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.3 (рис. @fig:032).



```
lab09-5.asm [-M--] 13 L: [ 1+17 18/ 18] *(284 /
%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
    mov eax,3
    mov ebx,2
    add eax,ebx
    mov ecx,4
    mul ecx
    add eax,5
    mov edi,eax
    mov eax,dib
    call sprint
    mov eax,edi
    call iprintLF
    call quit
```

Рис. 32: Изменяем файл

Создаем исполняемый файл и запускаем его (рис. @fig:033).

```
svzhukova@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
lab09-5.asm:14: error: symbol `dib' not defined
svzhukova@fedora:~/work/arch-pc/lab09$ mc
```

Рис. 33: Создаем и смотрим на работу программы(работает неправильно)

Создаем исполняемый файл и запускаем его в отладчике GDB и смотрим на изменение регистров командой si (рис. @fig:034).

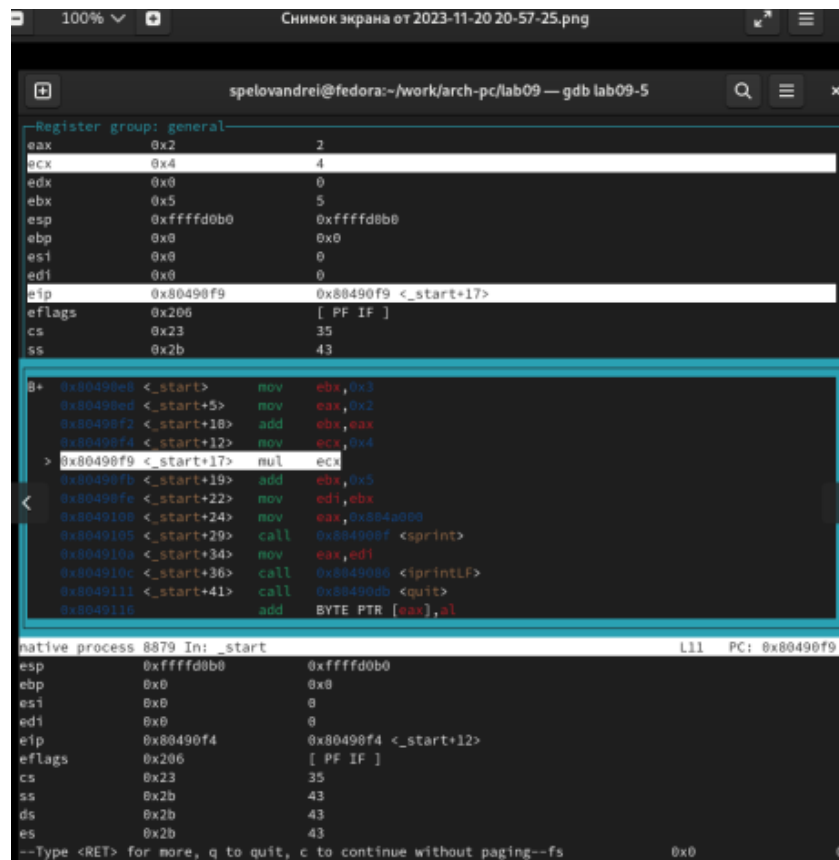
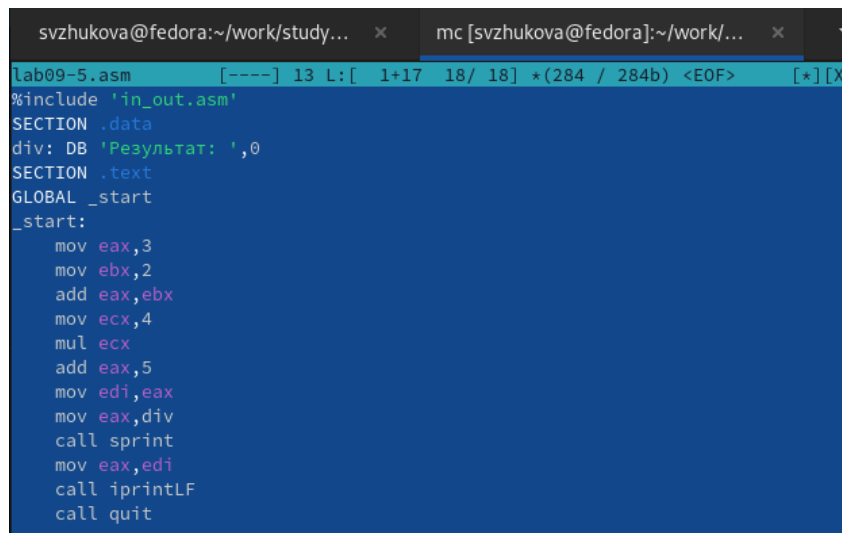


Рис. 34: Ищем ошибку регистров в отладчике

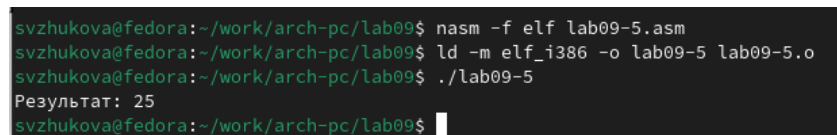
Изменяем программу для корректной работы (рис. @fig:035).



```
lab09-5.asm [----] 13 L: [ 1+17 18/ 18] *(284 / 284b) <EOF> [*][X]
#include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
    mov eax,3
    mov ebx,2
    add 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. 35: Меняем файл

Создаем исполняемый файл и запускаем его (рис. @fig:036).



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

Рис. 36: Создаем и запускаем файл

Выводы

Мы приобрели навыки написания программ с использованием подпрограмм. Познакомились с методами отладки при помощи GDB и его основными возможностями.