

Отчёта по лабораторной работе №9

Понятие подпрограммы. Отладчик GDB.

Сибомана Ламек НКАбд-03-24

Содержание

1	Цель работы	5
2	Теоретическое введение	6
2.0.1	Понятие об отладке	6
2.0.2	Методы отладки	7
3	Выполнение лабораторной работы	8
3.1	Реализация подпрограмм в NASM	8
3.2	Отладка программ с помощью GDB	9
3.3	Задание для самостоятельной работы	17
3.3.1	Задание 1	17
4	Выводы	22

Список иллюстраций

3.1	Текст программы lab09-1.asm	8
3.2	Текст программы lab09-1.asm	9
3.3	Создание lab09-2.asm	9
3.4	Текст программы lab09-2.asm	9
3.5	Создание и запуск lab09-2.asm	10
3.6	Создание	10
3.7	Текст программы	10
3.8	Создание	12
3.9	Текст программы	13
3.10	layout and info	13
3.11	текст	14
3.12	Cx/lsb	15
3.13	change of code	15
3.14	change of code	15
3.15	change of code	15
3.16	change of code	16
3.17	change of code	16
3.18	change of code	16
3.19	change of code	17
3.20	Текст программы	18
3.21	change of code	19
3.22	Текст программы	19
3.23	Создание и запуск lab09-5.asm	20
3.24	Текст программы	21
3.25	Создание и запуск lab09-5.asm	21

Список таблиц

1 Цель работы

Приобретение навыков написания программ с использованием циклов и обработкой аргументов командной строки.

2 Теоретическое введение

2.0.1 Понятие об отладке

Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа: • обнаружение ошибки; • поиск её местонахождения; • определение причины ошибки; • исправление ошибки. Можно выделить следующие типы ошибок: • синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка; • семантические ошибки — являются логическими и приводят к тому, что программа запускается, отработывает, но не даёт желаемого результата; • ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают прерывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль). Вторым этапом — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга. Третий этап — выяснение причины ошибки. После определения местонахождения ошибки обычно проще определить причину неправильной работы программы. Последний этап — исправление ошибки. После этого при повторном запуске программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново. |

2.0.2 Методы отладки

Наиболее часто применяют следующие методы отладки:

- создание точек контроля значений на входе и выходе участка программы (например, вывод промежуточных значений на экран — так называемые диагностические сообщения);
- использование специальных программ-отладчиков. Отладчики позволяют управлять ходом выполнения программы, контролировать и изменять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправление. Наиболее популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам. Пошаговое выполнение — это выполнение программы с остановкой после каждой строки, чтобы программист мог проверить значения переменных и выполнить другие действия. Точки останова — это специально отмеченные места в программе, в которых программа-отладчик приостанавливает выполнение программы и ждёт команд. Наиболее популярные виды точек останова:
- Breakpoint — точка останова (остановка происходит, когда выполнение доходит до определённой строки, адреса или процедуры, отмеченной программистом);
- Watchpoint — точка просмотра (выполнение программы приостанавливается, если программа обратилась к определённой переменной: либо считала её значение, либо изменила его). Точки останова устанавливаются в отладчике на время сеанса работы с кодом программы, т.е. они сохраняются до выхода из программы-отладчика или до смены отлаживаемой программы.

3 Выполнение лабораторной работы

3.1 Реализация подпрограмм в NASM

1. Сначала я создал каталог для программ лабораторной работы №9, затем перешёл в него и создал файл lab09-1.asm (рис. fig. ??)

![Создание каталога и файла lab09-1.asm] (image/1.png){#fig:001 width=70%}

Открывал файл в Midnight Commander и заполняем его в соответствии с листингом 9.1 (рис. fig. ??).

![Текст программы lab09-1.asm] (image/2.png){#fig:002 width=70%}

Создал исполняемый файл и проверьте его работу.

![Создание и запуск lab09-1.asm] image/3.png){#fig:003 width=70%}

Изменил текст программы добавив изменение значение регистра esx в цикле. Снова открывал файл для редактирования и изменяем его, добавив изменение значения регистра в цикле (рис. fig. 3.1)

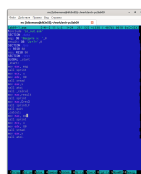


Рис. 3.1: Текст программы lab09-1.asm

Создаем исполняемый файл и запускаем его (рис. fig. ??).

![Создание и запуск lab09-1.asm] image/5.png){#fig:005 width=70%}

3.2 Отладка программ с помощью GDB

Создаем новый файл в каталоге(рис. fig. ??).

```
lsibomana@dk3n55 ~/work/arch-pc/lab09 $ touch lab09-2
lsibomana@dk3n55 ~/work/arch-pc/lab09 $ mc
```

Создаем исполняемый файл и запускаем его

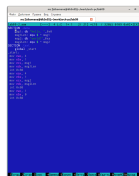


Рис. 3.2: Текст программы lab09-1.asm

Получаем исходный файл с использованием отладчика gdb

```
lsibomana@dk3n55 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
lsibomana@dk3n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
lsibomana@dk3n55 ~/work/arch-pc/lab09 $ gdb lab09-2
GNU gdb (Gentoo 14.2 vanilla) 14.2
Copyright (C) 2023 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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>
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. 3.3: Создание lab09-2.asm

Запускаем команду в отладчике



Рис. 3.4: Текст программы lab09-2.asm

Устанавливаем брейкпоинт на метку _start и запускаем программу

```

(gdb) break _start
Breakpoint 1 at 0x08049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/l/s/lsibomana/work/arch-pc/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
(gdb) █

```

Рис. 3.5: Создание и запуск lab09-2.asm

Смотрим 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. 3.6: Создание

Переключаемся на отображение команд с Intel'овским синтаксисом (рис. fig. ??).

```

(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. 3.7: Текст программы

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. ??).

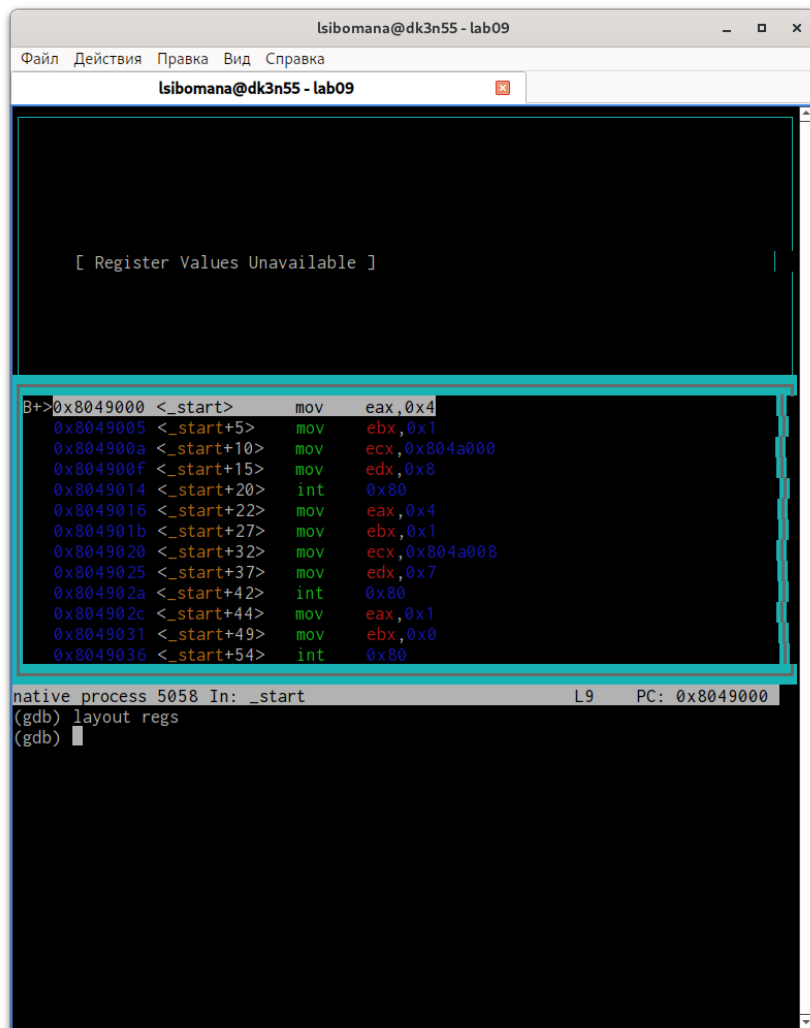


Рис. 3.8: Создание

Проверяем была ли установлена точка останова и устанавливаем точку останова предпоследней инструкции (рис. fig. ??).

```

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

native process 5058 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) info breakpoints
Num      Type             Disp Enb Address      What
1        breakpoint       keep y   0x08049000 lab09-2.asm:9
          breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb)

```

Рис. 3.9: Текст программы

Посмотрим информацию о всех установленных точках останова (рис. fig. ??).

```

(gdb) i b
Num      Type             Disp Enb Address      What
1        breakpoint       keep y   0x08049000 lab09-2.asm:9
          breakpoint already hit 1 time
2        breakpoint       keep y   0x08049031 lab09-2.asm:20
(gdb)

```

Рис. 3.10: layout and info

Выполняем 5 инструкций командой si (рис. fig. ??).

```

lsibomana@dk3n55 - lab09
Файл Действия Правка Вид Справка
lsibomana@dk3n55 - lab09
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc450 0xffffc450
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

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

native process 5058 In: _start L14 PC: 0x8049016
1 breakpoint keep y 0x08049000 lab09-2.asm:9
  breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab09-2.asm:9
  breakpoint already hit 1 time
2 breakpoint keep y 0x08049031 lab09-2.asm:20
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 3.11: текст

Во время выполнения команд менялись регистры: ebx, ecx, edx, eax, eip.

Смотрим значение переменной msg1 по имени (рис. fig. ??).

```

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

```

{#fig:0017width=70%}

Смотрим значение переменной msg2 по адресу (рис. fig. ??).

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

Рис. 3.12: Cx/lb

Изменим первый символ переменной msg1 (рис. fig. ??).

Меняем символ

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

Рис. 3.13: change of code

Изменим первый символ переменной msg2 (рис. fig. ??).

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

Рис. 3.14: change of code

Смотрим значение регистра edx в разных форматах (рис. fig. ??).

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

Изменяем регистр ebx (рис. fig. ??)

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

Рис. 3.15: change of code

Выводится разные значения, так как команда без кеавычек присваивает регистру вводимое значение.

Прописываем команды для завершения программы и выхода из GDB (рис.

```
(gdb) c
Continuing.
World!

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

fig. ??).

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

```
lsibomana@dk3n55 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm ~
/work/arch-pc/lab09-3.asm
lsibomana@dk3n55 ~/work/arch-pc/lab09 $ █
```

Рис. 3.16: change of code

Создаем исполняемый файл и запускаем его в отладчике GDB (рис. fig. ??).

```
lsibomana@dk3n55 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-
3.asm
lsibomana@dk3n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
lsibomana@dk3n55 ~/work/arch-pc/lab09 $ gdb --args lab09-3 2 3 '5'
```

Рис. 3.17: change of code

Установим точку останова перед первой инструкцией в программе и запустим

```
Reading symbols from lab09-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/1/s/lsibomana/work/arch-pc/lab
09/lab09-3 2 3 5

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx
(gdb) x/x $esp
0xfffffc450: 0x00000004
(gdb) █
```

ее (рис. fig. ??).

Смотрим

позиции стека по разным адресам (рис. fig. ??).

```
(gdb) x/x $esp
0xfffffc450: 0x00000004
(gdb) x/s *(void**)( $esp + 4)
0xfffffc6b3: "/afs/.dk.sci.pfu.edu.ru/home/1/s/lsibomana/work/arch-pc/lab
09/lab09-3"
(gdb) x/s *(void**)( $esp + 8)
0xfffffc6f9: "2"
(gdb) x/s *(void**)( $esp + 12)
0xfffffc6fb: "3"
(gdb) x/s *(void**)( $esp + 16)
0xfffffc6fd: "5"
(gdb) x/s *(void**)( $esp + 20)
0x0: <error: Cannot access memory at address 0x0>
(gdb) █
```

Рис. 3.18: change of code

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

3.3 Задание для самостоятельной работы

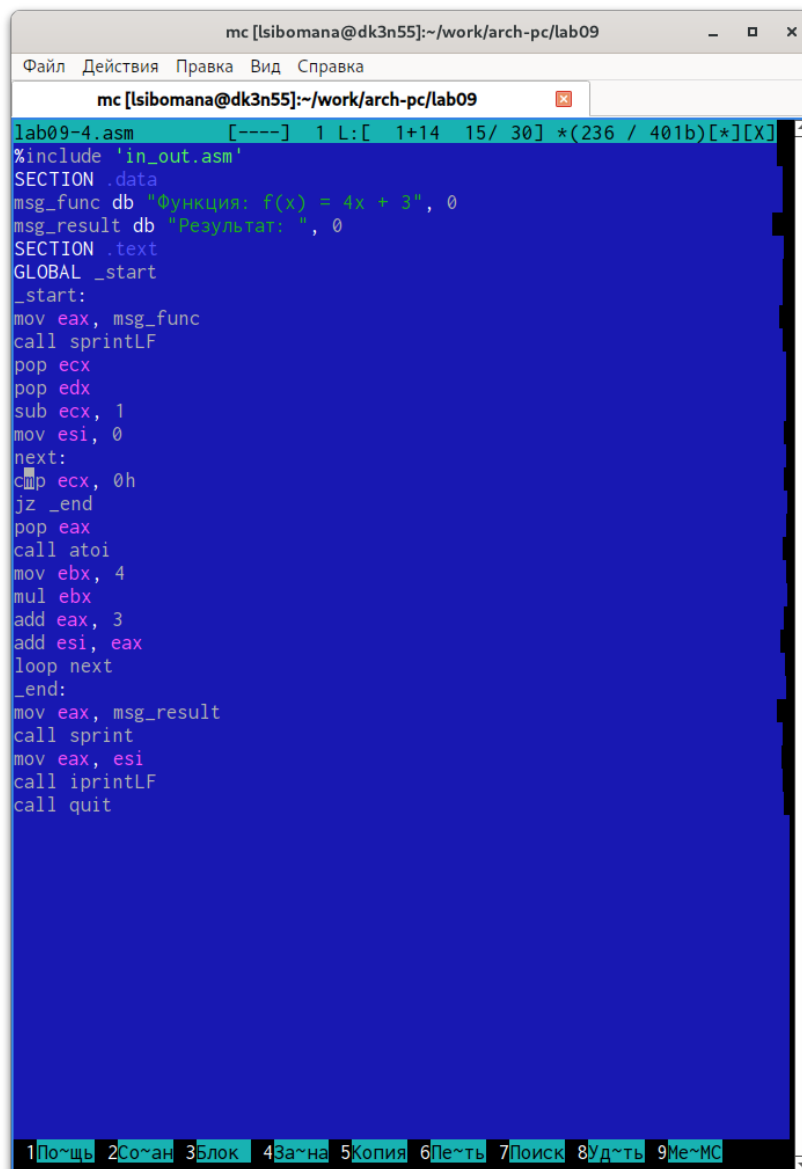
3.3.1 Задание 1

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

```
lsibomana@dk3n55 ~ $ cd ~/work/arch-pc/lab09
lsibomana@dk3n55 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab
8-4.asm ~/work/arch-pc/lab09/lab09-4.asm
lsibomana@dk3n55 ~/work/arch-pc/lab09 $
```

Рис. 3.19: change of code

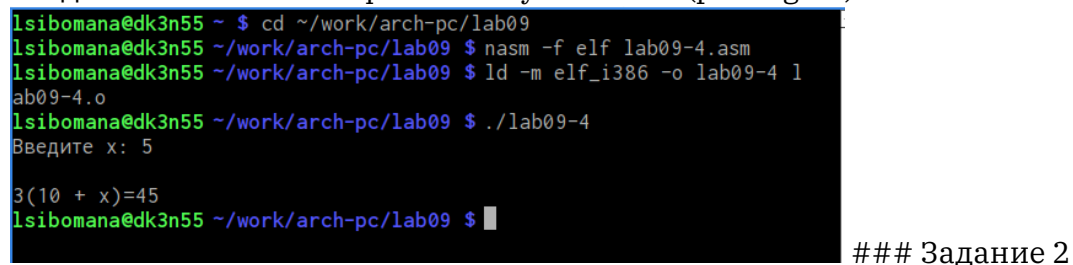
Открываем файл в Midnight Commander и меняем его, создавая подпрограмму (рис. fig. ??).



```
lab09-4.asm [----] 1 L: [ 1+14 15/ 30] *(236 / 401b)[*][X]
#include 'in_out.asm'
SECTION .data
msg_func db "Функция: f(x) = 4x + 3", 0
msg_result db "Результат: ", 0
SECTION .text
GLOBAL _start
_start:
mov eax, msg_func
call sprintf
pop ecx
pop edx
sub ecx, 1
mov esi, 0
next:
cmp ecx, 0h
jz _end
pop eax
call atoi
mov ebx, 4
mul ebx
add eax, 3
add esi, eax
loop next
_end:
mov eax, msg_result
call sprintf
mov eax, esi
call iprintf
call quit
```

Рис. 3.20: Текст программы

Создаем исполняемый файл и запускаем его (рис. fig. ??).



```
lsibomana@dk3n55 ~ $ cd ~/work/arch-pc/lab09
lsibomana@dk3n55 ~/work/arch-pc/lab09 $ nasm -f elf lab09-4.asm
lsibomana@dk3n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-4 l
ab09-4.o
lsibomana@dk3n55 ~/work/arch-pc/lab09 $ ./lab09-4
Введите x: 5
3(10 + x)=45
lsibomana@dk3n55 ~/work/arch-pc/lab09 $
```

Задание 2

Создаем новый файл в директории (рис. fig. ??).

```
lsibomana@dk3n55 ~/work/arch-pc/lab09 $ touch lab09-5.asm
lsibomana@dk3n55 ~/work/arch-pc/lab09 $
```

Рис. 3.21: change of code

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

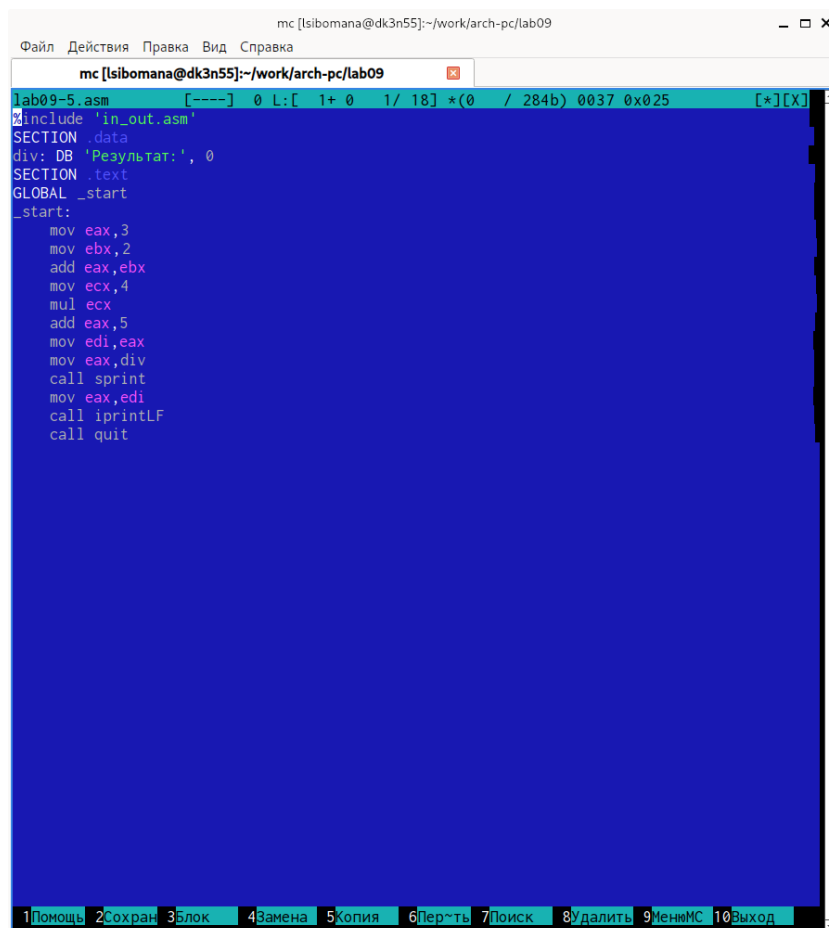


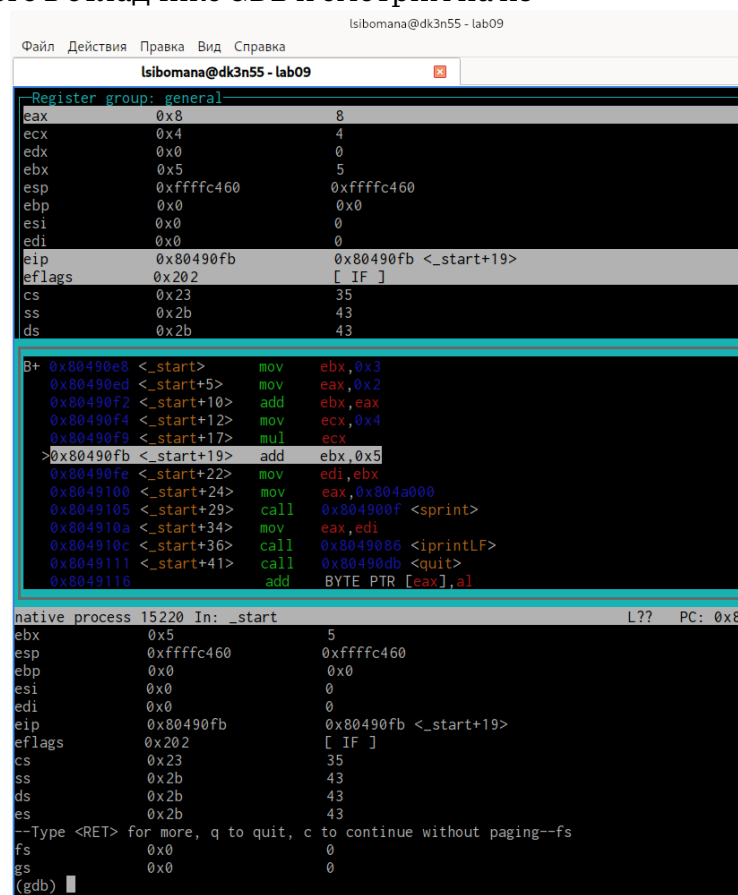
Рис. 3.22: Текст программы

Создаем исполняемый файл и запускаем его (рис. fig. ??).

```
lsibomana@dk3n55 ~/work/arch-pc/lab09 $ nasm -f elf lab09-5.asm
lsibomana@dk3n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-5 lab09-5.o
lsibomana@dk3n55 ~/work/arch-pc/lab09 $ ./lab09-5
Результат: 10
lsibomana@dk3n55 ~/work/arch-pc/lab09 $
```

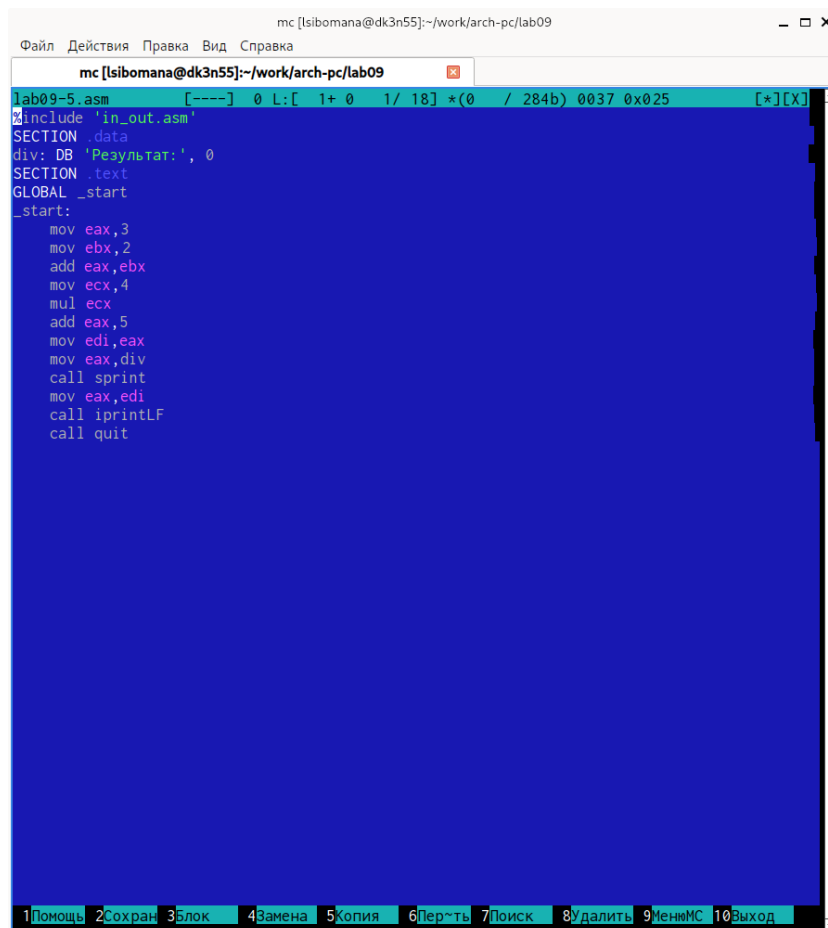
Рис. 3.23: Создание и запуск lab09-5.asm

Создаем исполняемый файл и запускаем его в отладчике GDB и смотрим на из-



менение регистров командой si (рис. fig. ??).

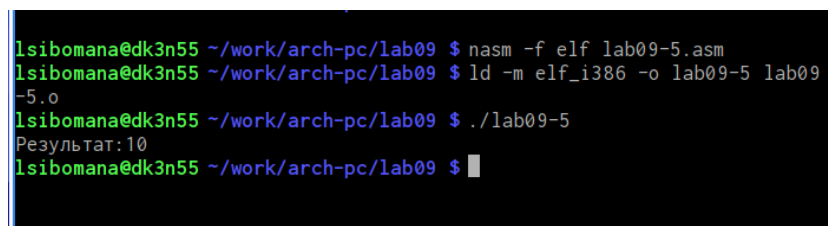
Изменяем программу для корректной работы (рис. fig. ??).



```
mc [lsibomana@dk3n55]:~/work/arch-pc/lab09
lab09-5.asm [----] 0 L: [ 1+ 0 1/ 18] *(0 / 284b) 0037 0x025 [*][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 iprintlnf
    call quit
```

Рис. 3.24: Текст программы

Создаем исполняемый файл и запускаем его (рис. fig. ??).



```
lsibomana@dk3n55 ~/work/arch-pc/lab09 $ nasm -f elf lab09-5.asm
lsibomana@dk3n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-5 lab09-5.o
lsibomana@dk3n55 ~/work/arch-pc/lab09 $ ./lab09-5
Результат:10
lsibomana@dk3n55 ~/work/arch-pc/lab09 $
```

Рис. 3.25: Создание и запуск lab09-5.asm

4 Выводы

Мы познакомились с методами отладки при помощи GDB и его возможностями.