

Отчёта по лабораторной работе №9

Понятие подпрограммы. Отладчик GDB.

Сибомана Ламек НКАбд-03-24

Содержание

1	Цель работы	1
2	Теоретическое введение.....	1
3	Выполнение лабораторной работы	2
4	Выводы	13

1 Цель работы

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

2 Теоретическое введение

###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). Второй этап — 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. |

###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

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

Открывал файл в Midnight Commander и заполняю его в соответствии с листингом 9.1 (рис. fig. 002?).

! [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 значения регистра ехх в цикле. Снова открывал файл для редактирования и изменяю его, добавив изменение значения регистра в цикле (рис. fig. 1)



Рис. 1: Текст программы lab09-1.asm

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

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

##Отладка программ с помощью GDB

Создаем новый файл в каталоге(рис. fig. 006?).

```
lsibomana@edk3n55 ~/work/arch-pc/lab09 $ touch lab09-2.asm
lsibomana@edk3n55 ~/work/arch-pc/lab09 $ mc
```

Создаем исполняемый файл и запускаем его



Рис. 2: Текст программы lab09-1.asm

Получаем исходный файл с использованием отладчика gdb

```
lsibomana@edk3n55 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
lsibomana@edk3n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
lsibomana@edk3n55 ~/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.
Type "show reporting instructions" for reporting instructions.
For bug reporting instructions, please see:
<http://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: Создание lab09-2.asm

Запускаем команду в отладчике



Рис. 4: Текст программы lab09-2.asm

Устанавливаем брейкпоинт на метку _start и запускаем программу

```

(gdb) break _start
Breakpoint 1 at 0x00400000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/1/s/lsibomana/work/arch-pc/la
b09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 1
(gdb) █

```

Рис. 5: Создание и запуск lab09-2.asm

Смотрим дисассемблированный код программы с помощью команды disassemble, начиная с метки _start

```

(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x00400000 <+0>:    mov     $0x4, %eax
0x00400005 <+5>:    mov     $0x1, %ebx
0x0040000a <+10>:   mov     $0x00400000, %ecx
0x0040000f <+15>:   mov     $0x0, %edx
0x00400014 <+20>:   int     $0x80
0x0040001b <+22>:   mov     $0x4, %eax
0x0040001e <+27>:   mov     $0x1, %ebx
0x00400023 <+32>:   mov     $0x00400000, %ecx
0x00400028 <+37>:   mov     $0x7, %edx
0x0040002d <+42>:   int     $0x80
0x00400032 <+44>:   mov     $0x1, %eax
0x00400035 <+49>:   mov     $0x0, %ebx
0x00400038 <+54>:   int     $0x80
End of assembler dump.
(gdb) █

```

Рис. 6: Создание

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:
=> 0x00400000 <+0>:    mov     eax, 4
0x00400005 <+5>:    mov     ebx, 1
0x0040000a <+10>:   mov     ecx, 0x00400000
0x0040000f <+15>:   mov     edx, 0
0x00400014 <+20>:   int     80h
0x0040001b <+22>:   mov     eax, 4
0x0040001e <+27>:   mov     ebx, 1
0x00400023 <+32>:   mov     ecx, 0x00400000
0x00400028 <+37>:   mov     edx, 7
0x0040002d <+42>:   int     80h
0x00400032 <+44>:   mov     eax, 1
0x00400035 <+49>:   mov     ebx, 0
0x00400038 <+54>:   int     80h
End of assembler dump.
(gdb) █

```

Рис. 7: Текст программы

Различия отображения синтаксиса машинных команд в режимах ATТ 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 (/).

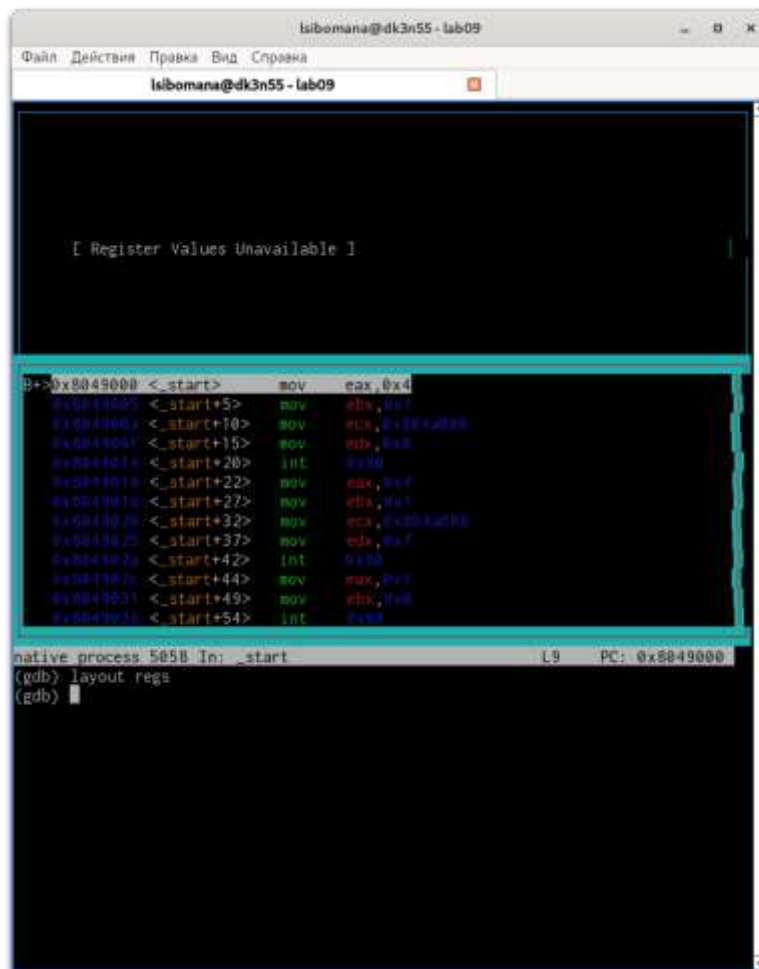
3. Префиксы размера операндов: В АТТ синтаксисе размер операнда указывается перед операндом с использованием префиксов, таких как “b” (byte), “w” (word), “l” (long) и “q” (quadword). В Intel синтаксисе размер операнда указывается после операнда с использованием суффиксов, таких как “b”, “w”, “d” и “q”.

4. Знак операндов: В АТТ синтаксисе операнды с позитивными значениями предваряются символом “\$”. В Intel синтаксисе операнды с позитивными значениями могут быть указаны без символа “\$”.

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”).

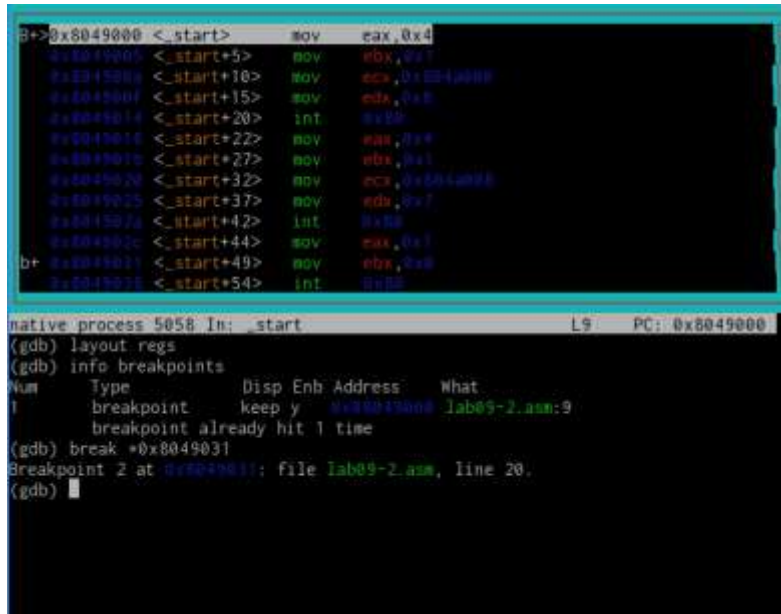
Включаем режим псевдографики (рис. fig. 013?).



The screenshot shows a GDB terminal window with the title "lsibomana@dk3n55 - lab09". The window has a menu bar with "Файл", "Действия", "Правка", "Вид", and "Справка". Below the menu bar, there is a text area displaying "[Register Values Unavailable]". Below this, a list of assembly instructions is shown, each with a memory address, a comment, and the instruction itself. The instructions are:
0x00490000 <_start> mov eax, 0x4
0x00490005 <_start+5> mov ebx, 0x1
0x00490006 <_start+10> mov ecx, 0x00490000
0x00490007 <_start+15> mov edx, 0x0
0x00490008 <_start+20> int 0x0
0x00490009 <_start+22> mov eax, 0x0
0x0049000a <_start+27> mov ebx, 0x1
0x0049000b <_start+32> mov ecx, 0x00490000
0x0049000c <_start+37> mov edx, 0x0
0x0049000d <_start+42> int 0x0
0x0049000e <_start+44> mov eax, 0x0
0x0049000f <_start+49> mov ebx, 0x0
0x00490010 <_start+54> int 0x0
At the bottom of the window, the status bar shows "native process 505B In: _start L9 PC: 0x00490000". Below the status bar, the text "(gdb) layout regs" and "(gdb) " is visible.

Рис. 8: Создание

Проверяем была ли установлена точка останова и устанавливаем точку останова предпоследней инструкции (рис. fig. 014?).

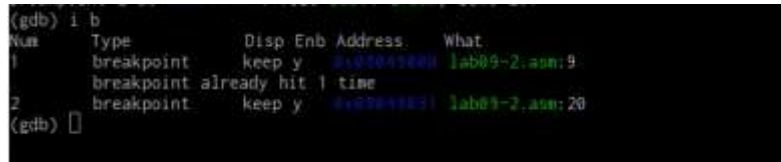


```
gdb> 0x8049000 <_start> mov eax,0x4
0x8049005 <_start+5> mov ebx,0x1
0x8049009 <_start+9> mov ecx,0x8049000
0x804900f <_start+15> mov edx,0x5
0x8049014 <_start+20> int $0
0x804901c <_start+28> mov eax,0x1
0x804901e <_start+30> mov ebx,0x1
0x8049020 <_start+32> mov ecx,0x8049000
0x8049025 <_start+37> mov edx,0x7
0x804902a <_start+42> int $0
0x804902c <_start+44> mov eax,0x1
br 0x8049031 <_start+49> mov ebx,0x0
0x8049035 <_start+54> int $0

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

Рис. 9: Текст программы

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



```
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x8049000 lab09-2.asm:9
breakpoint already hit 1 time
2 breakpoint keep y 0x8049031 lab09-2.asm:20
(gdb)
```

Рис. 10: layout and info

Выполняем 5 инструкций командой si (рис. fig. 016?).

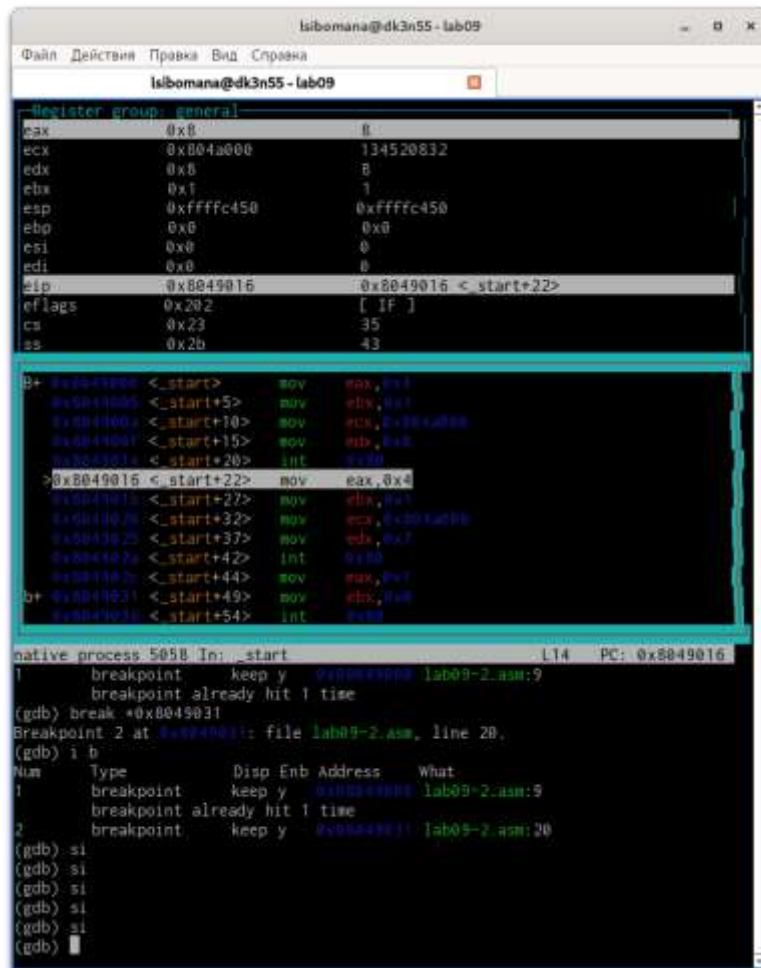
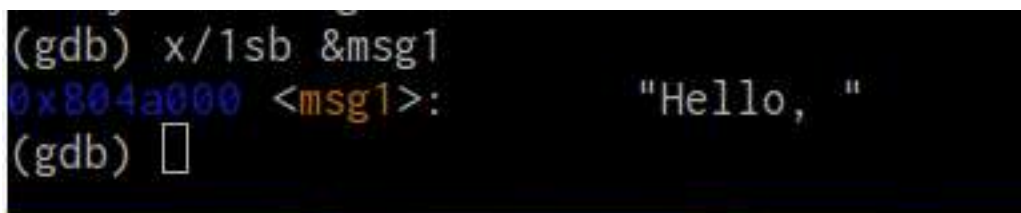


Рис. 11: текст

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

Смотрим значение переменной msg1 по имени (рис. fig. 017?).



{#fig:0017width=70%}

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

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

Рис. 12: Cx/lb

Изменим первый символ переменной msg1 (рис. fig. 019?).

Меняем символ

```
(gdb) set (char)&msg1='h'
(gdb) x/lb &msg1
0x804a008 <msg1>: "hello, "
(gdb)
```

Рис. 13: change of code

Изменим первый символ переменной msg2 (рис. fig. 020?).

```
(gdb) set (char)&msg2='L'
(gdb) x/lb &msg2
0x804a009 <msg2>: "Lorld!\n\034"
(gdb)
```

Рис. 14: change of code

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

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

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. 15: change of code

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)
```

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


```
lsibomana@dk3n55 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab0-2.asm ~/work/arch-pc/lab09-3.asm
lsibomana@dk3n55 ~/work/arch-pc/lab09 $
```

Рис. 16: change of code

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

```
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. 17: change of code

Установим точку останова перед первой инструкцией в программе и запустим ее (рис.

```
(gdb) b _start
Breakpoint 1 at 0x00400005: file lab09-3.asm, line 5.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/1/s/lsibomana/work/arch-pc/lab09/lab09-3 2 3 5

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx
(gdb) x/x $esp
0xffff450: 0x00000004
(gdb)
```

fig. 026?). Смотрим позиции стека по разным адресам (рис. fig. 027?).

```
(gdb) x/x $esp
0xffff450: 0x00000004
(gdb) x/s *(void**)(esp + 4)
0xffff454: "/afs/.dk.sci.pfu.edu.ru/home/1/s/lsibomana/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffff458: "2"
(gdb) x/s*(void**)(esp + 12)
0xffff45c: "3"
(gdb) x/s*(void**)(esp + 16)
0xffff460: "5"
(gdb) x/s*(void**)(esp + 20)
00: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 18: change of code

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?).

```
lsibomana@dk3n55 ~ $ cd ~/work/arch-pc/lab09
lsibomana@dk3n55 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-4.asm ~/work/arch-pc/lab09/lab09-4.asm
lsibomana@dk3n55 ~/work/arch-pc/lab09 $
```

Рис. 19: change of code

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

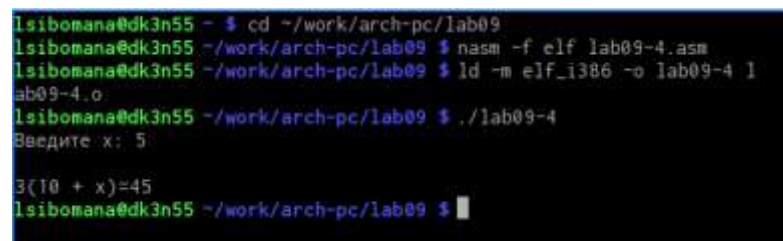


```

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. 20: Текст программы

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



```

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 lab09-4.o
lsibomana@dk3n55 ~/work/arch-pc/lab09 $ ./lab09-4
Введите x: 5

3(10 + x)=45
lsibomana@dk3n55 ~/work/arch-pc/lab09 $
  
```

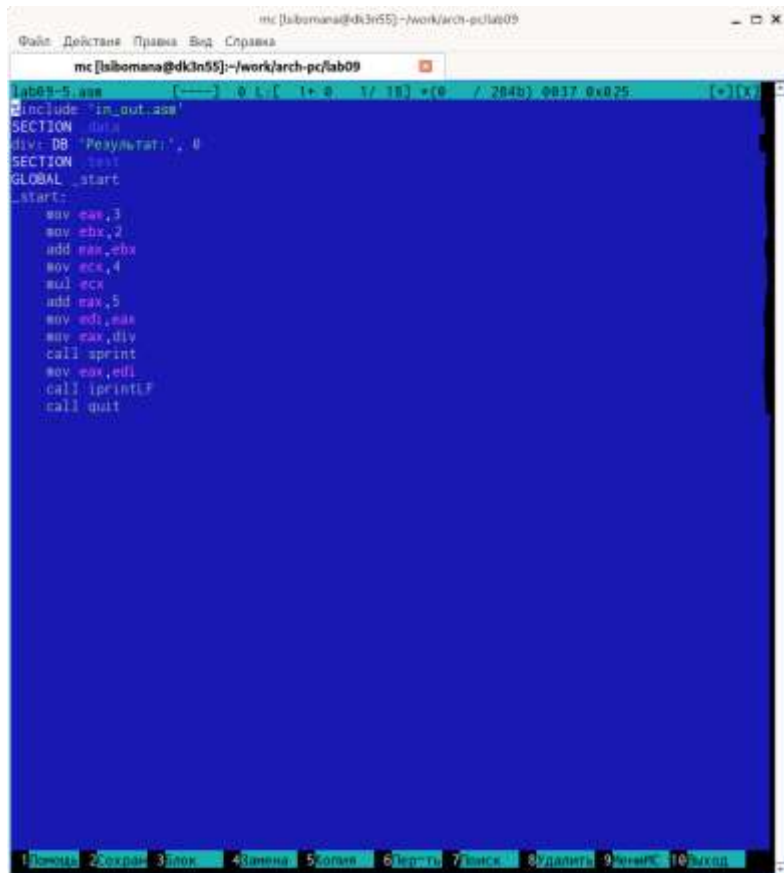
###Задание 2

Создаем новый файл в директории (рис. fig. 031?).

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

Рис. 21: change of code

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.3 (рис. fig. 2fig:032?).



```
lab09-5.asm
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 sprintf
    mov eax,edi
    call iprintf
    call quit
```

Рис. 22: Текст программы

Создаем исполняемый файл и запускаем его (рис. fig. 2fig:033?).

```
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. 23: Создание и запуск lab09-5.asm

```

lab0mana@dk3n55 - lab09
File  Действия  Прошел  Вид  Стрелки
lab0mana@dk3n55 - lab09

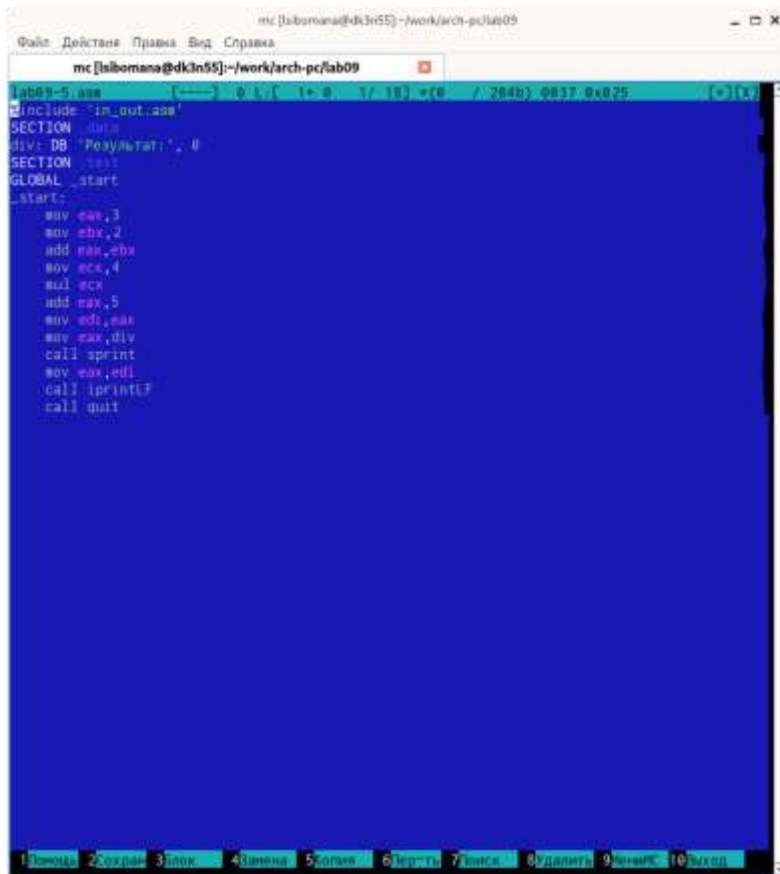
Register group: general
eax 0x5 0
ecx 0x4 4
edx 0x8 0
ebx 0x5 5
esp 0xffffc460 0xffffc460
ebp 0x0 0x0
esi 0x0 0
edi 0x0 0
eip 0x80490fb 0x80490fb <_start+19>
eflags 0x202 [ IF ]
cs 0x23 35
ss 0x2b 43
ds 0x2b 43

0x80490fb <_start> mov ebx,0x5
0x80490fd <_start+9> mov eax,0x0
0x80490ff <_start+10> add ebx,eax
0x8049101 <_start+12> mov ecx,0x0
0x8049103 <_start+13> esi ecx
0x80490fb <_start+19> add ebx,0xc
0x8049105 <_start+22> mov edi,ebx
0x8049107 <_start+24> mov eax,0x0
0x8049109 <_start+29> call _expn0000 <_print>
0x804910b <_start+34> mov eax,edi
0x804910d <_start+36> call _expn0000 <_printf>
0x804910f <_start+41> call _expn0000 <_quit>
0x8049111 <_start+46> add BYTE PTR [eax],al

native process 15220 In: _start L?? PC: 0x80490fb
eax 0x5 5
esp 0xffffc460 0xffffc460
ebp 0x0 0x0
esi 0x0 0
edi 0x0 0
eip 0x80490fb 0x80490fb <_start+19>
eflags 0x202 [ IF ]
cs 0x23 35
ss 0x2b 43
ds 0x2b 43
es 0x2b 43

--type <RET> for more, q to quit, c to continue without paging--fs
fs 0x0 0
gs 0x0 0
(gdb)

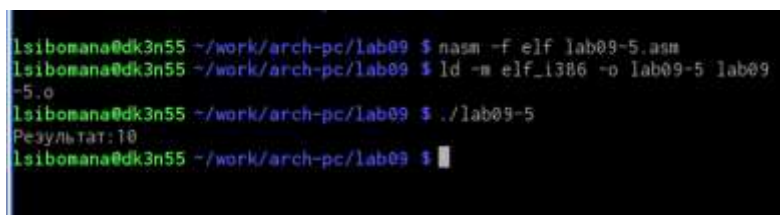
```



```
lab09-5.asm
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 sprintf
    mov eax, edi
    call iprintlf
    call quit
```

Рис. 24: Текст программы

Создаем исполняемый файл и запускаем его (рис. fig. 2fig:036?).



```
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. 25: Создание и запуск lab09-5.asm

4 Выводы

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

#Содержание отчёта

Отчёт должен включать: • Титульный лист с указанием номера лабораторной работы и ФИО студента. • Формулировка цели работы. • Описание результатов выполнения лабораторной работы: – описание выполняемого задания; – скриншоты (снимки экрана), фиксирующие выполнение заданий лабораторной работы; – комментарии и выводы по

результатам выполнения заданий. • Описание результатов выполнения заданий для самостоятельной работы: – описание выполняемого задания; – скриншоты (снимки экрана), фиксирующие выполнение заданий; – комментарии и выводы по результатам выполнения заданий; – листинги написанных программ (текст программ). • Выводы, согласованные с целью работы. Отчёт по выполнению лабораторной работы оформляется в формате Markdown. В качестве отчёта необходимо предоставить отчёты в 3 форматах: pdf, docx и md. А также файлы с исходными текстами написанных при выполнении лабораторной работы программ (файлы *.asm). Файлы необходимо загрузить на странице курса в ТУИС в задание к соответствующей лабораторной работе и загрузить на Github.