

Архитектура компьютера

Отчёт по лабораторной работе №9

Ибрахим Мохсейн Алькамаль

Содержание

1	Цель работы	6
2	Задание	7
3	Теоретическое введение	8
4	Выполнение лабораторной работы	10
5	Выполнение заданий для самостоятельной работы	22
6	Выводы	29
	Список литературы	30

Список иллюстраций

4.1	1)Создание файла	10
4.2	2)Текст файла	11
4.3	3)Создание файла	11
4.4	4)Текст программы изменённый	12
4.5	5)Работа файла	12
4.6	6) Создание файла	12
4.7	7) Текст файла	13
4.8	8) Работа файла	13
4.9	9) Запуск gdb и файла	14
4.10	10) Запуск gdb и файла	14
4.11	11) Установка break_point и запуск программы с ней	14
4.12	12)Дисассимилированный код	15
4.13	13)Отображение команд с Intel'овским синтаксисом	15
4.14	14)Режим псевдографики	16
4.15	15)info breakpoints	16
4.16	16)Установка точки останова	17
4.17	17)Вывод информации о точках останова	17
4.18	18)5 инструкций si	18
4.19	19)info registers	18
4.20	20)Значение переменной msg1 по имени	18
4.21	21)Значение переменной msg2 по адресу	19
4.22	22)Изменение первого символа переменной msg1	19
4.23	23)Изменение символа второй переменной	19
4.24	25)Изменяю значение регистра ebx	19
4.25	26)Завершаю программу	20
4.26	27)Копирование, создание, загрузка файла	20
4.27	28)Точка установа и запуск	21
4.28	29)Регистр esp	21
4.29	30)Остальные позиции стека	21
5.1	Копирование файла	22
5.2	Работа файла	24
5.3	Создание файла	24
5.4	Текст файла	24
5.5	Передача файла	25
5.6	Точки останова	25
5.7	Отладка файла	26

5.8	Исправленный текст файла	27
5.9	Работа файла	28

Список таблиц

1 Цель работы

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

2 Задание

1. Создайте каталог для выполнения лабораторной работы № 9, перейдите в него и создайте файл lab09-1.asm
2. Внимательно изучите текст программы (Листинг 9.1). Введите в файл lab09-1.asm текст программы из листинга 9.1. Создайте исполняемый файл и проверьте его работу
3. Создайте файл lab09-2.asm с текстом программы из Листинга 9.2
4. Проверьте работу программы, запустив ее в оболочке GDB

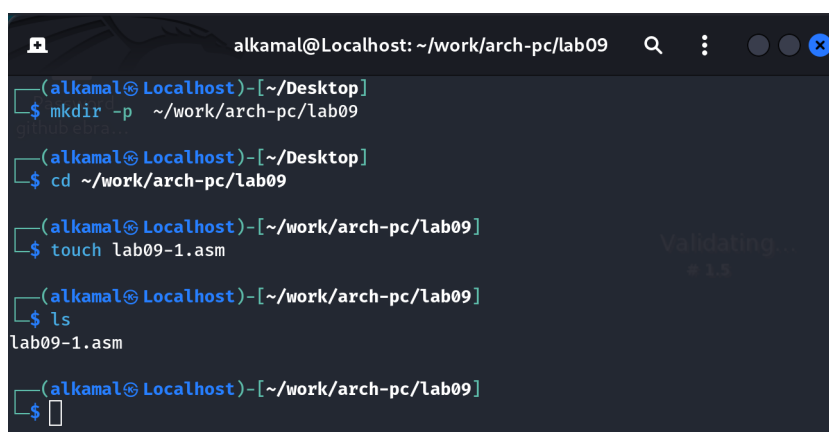
3 Теоретическое введение

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. 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 быст-

рее 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 строчки, 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.

4 Выполнение лабораторной работы

- 1) Создаю каталог для выполнения лабораторной работы № 9, перехожу в него и создаю файл lab09-1.asm



```
alkamal@Localhost: ~/work/arch-pc/lab09
(alkamal@Localhost)~[~/Desktop]
$ mkdir -p ~/work/arch-pc/lab09
(alkamal@Localhost)~[~/Desktop]
$ cd ~/work/arch-pc/lab09
(alkamal@Localhost)~[~/work/arch-pc/lab09]
$ touch lab09-1.asm
(alkamal@Localhost)~[~/work/arch-pc/lab09]
$ ls
lab09-1.asm
(alkamal@Localhost)~[~/work/arch-pc/lab09]
$
```

Рис. 4.1: 1)Создание файла

- 2) Ввожу в файл lab09-1.asm текст программы из листинга 9.1. Создаю исполняемый файл и проверяю его работу (2-3)

```

mc [alkamal@Localhost]:~/work/arch-pc/lab09
GNU nano 7.2 /home/alkamal/work/arch-pc/lab09/lab09-1.asm
#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:
;-----
; Основная программа
;-----
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
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы

```

Рис. 4.2: 2)Текст файла

```

(alkamal@Localhost)-[~/work/arch-pc/lab09]
$ nasm -f elf lab09-1.asm

(alkamal@Localhost)-[~/work/arch-pc/lab09]
$ ld -m elf_i386 -o lab09-1 lab09-1.o

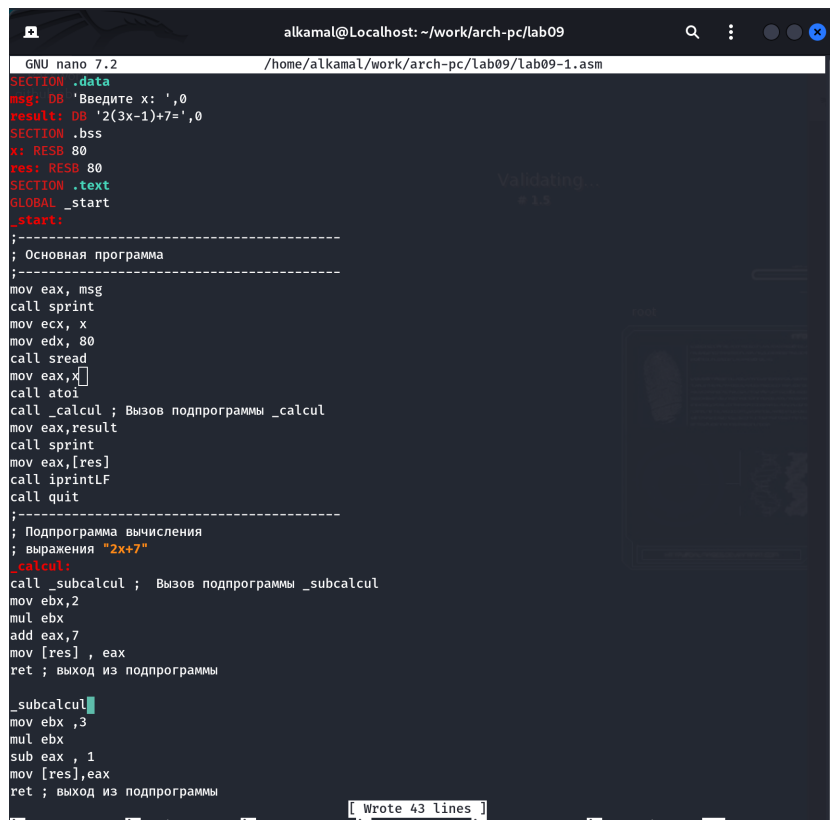
(alkamal@Localhost)-[~/work/arch-pc/lab09]
$ ./lab09-1
Введите x: 23
2x+7=53

(alkamal@Localhost)-[~/work/arch-pc/lab09]
$ 

```

Рис. 4.3: 3)Создание файла

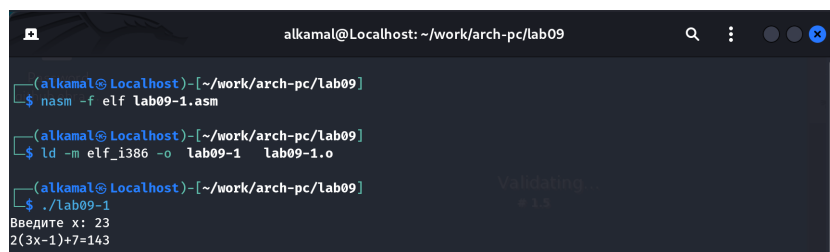
- 3) Изменяю текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $\boxed{x}(\boxed{x}(\boxed{x}))$, где \boxed{x} вводится с клавиатуры, $\boxed{x}(\boxed{x}) = 2\boxed{x} + 7$, $\boxed{x}(\boxed{x}) = 3\boxed{x} - 1$. Создаю файл и проверяю его работу (4-5)



```
alkamal@Localhost: ~/work/arch-pc/lab09
GNU nano 7.2 /home/alkamal/work/arch-pc/lab09/lab09-1.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:
;-----
; Основная программа
;-----
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
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
call _subcalcul ; Вызов подпрограммы _subcalcul
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы

_subcalcul
mov ebx, 3
mul ebx
sub eax, 1
mov [res], eax
ret ; выход из подпрограммы
```

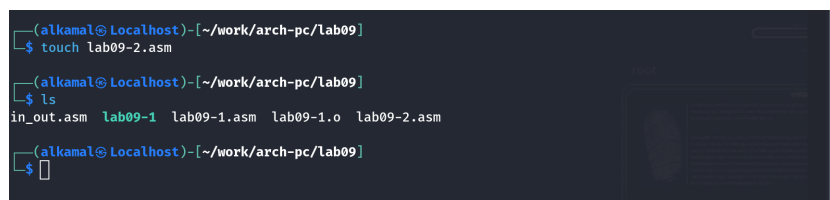
Рис. 4.4: 4)Текст программы изменённый



```
alkamal@Localhost: ~/work/arch-pc/lab09
$ nasm -f elf lab09-1.asm
$ ld -m elf_i386 -o lab09-1 lab09-1.o
$ ./lab09-1
Введите x: 23
2(3x-1)+7=143
```

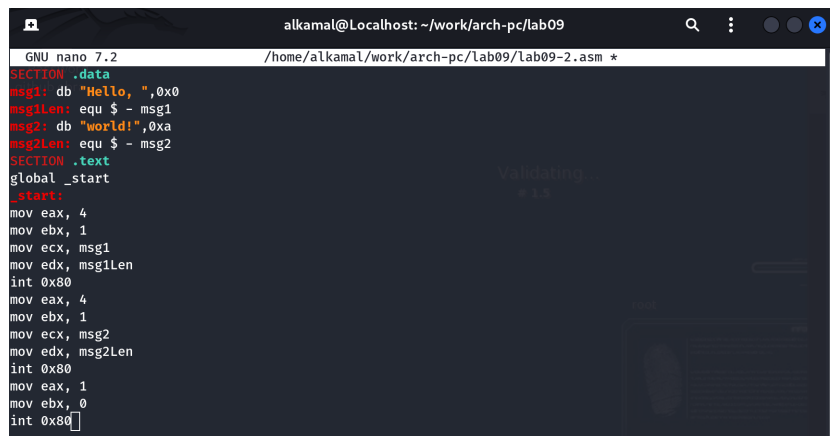
Рис. 4.5: 5)Работа файла

4) Создаю файл lab09-2.asm с текстом программы из Листинга 9.2. (6-8)



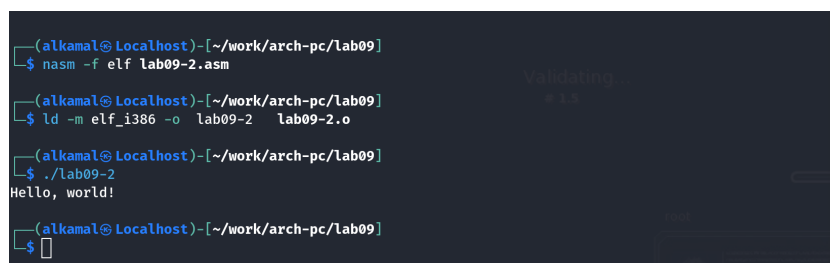
```
alkamal@Localhost: ~/work/arch-pc/lab09
$ touch lab09-2.asm
$ ls
in_out.asm lab09-1 lab09-1.asm lab09-1.o lab09-2.asm
$
```

Рис. 4.6: 6) Создание файла



```
alkamal@Localhost: ~/work/arch-pc/lab09
GNU nano 7.2 /home/alkamal/work/arch-pc/lab09/lab09-2.asm *
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. 4.7: 7) Текст файла



```
alkamal@Localhost)-[~/work/arch-pc/lab09]
$ nasm -f elf lab09-2.asm

alkamal@Localhost)-[~/work/arch-pc/lab09]
$ ld -m elf_i386 -o lab09-2 lab09-2.o

alkamal@Localhost)-[~/work/arch-pc/lab09]
$ ./lab09-2
Hello, world!

alkamal@Localhost)-[~/work/arch-pc/lab09]
$
```

Рис. 4.8: 8) Работа файла

- 5) Получаю исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить(9)

```
alkamal@Localhost: ~/work/arch-pc/lab09
(alkamal@Localhost)-[~/work/arch-pc/lab09]
$ nasm -f elf -g -l lab09-1.lst lab09-2.asm
Validating
x 13

(alkamal@Localhost)-[~/work/arch-pc/lab09]
$ ld -m elf_i386 -o lab09-2 lab09-2.o

(alkamal@Localhost)-[~/work/arch-pc/lab09]
$ gdb lab09-2
GNU gdb (Debian 13.2-1) 13.2
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-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. 4.9: 9) Запуск gdb и файла

```
(gdb) run
Starting program: /home/alkamal/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 4045) exited normally]
(gdb) 
```

Рис. 4.10: 10) Запуск gdb и файла

- 6) Для более подробного анализа программы устанавливаю брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запускаю её

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/alkamal/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
(gdb) 
```

Рис. 4.11: 11) Установка break_point и запуск программы с ней

- 7) Смотрю 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. 4.12: 12) Дисассимилированный код

- 8) Переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`

```

(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. 4.13: 13) Отображение команд с Intel'овским синтаксисом

- 9) Включаю режим псевдографики для более удобного анализа программы

```

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcf30 0xffffcf30
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
0x804902a <_start+42> int    0x80

native process 6095 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /home/alkamal/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9
(gdb)

```

Рис. 4.14: 14)Режим псевдографики

10) С помощью info breakpoints узнаю информацию об установленных точках останова

```

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

```

Рис. 4.15: 15)info breakpoints

11) Определяю адрес предпоследней инструкции (mov ebx,0x0) и устанавливаю точку останова. Смотрю информацию об установленных точках останова (16-17)


```

alkamal@Localhost: ~/work/arch-pc/lab09

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcf30 0xffffcf30
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
0x804902a <_start+42> int    0x80

native process 6173 In: _start L9 PC: 0x8049000
1 breakpoint keep y 0x08049000 lab09-2.asm:9
  breakpoint already hit 1 time
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /home/alkamal/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb)

```

Рис. 4.16: 16) Установка точки останова

```

(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. 4.17: 17) Вывод информации о точках останова

- 12) Выполняю 5 инструкций с помощью команды stepi (или si) и слежу за изменением значений регистров. Значения регистров eax, edx, ecx, esp, eip, cs, ds, ebx, ss, eflags, es изменяются

```

alkamal@Localhost: ~/work/arch-pc/lab09
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcf30 0xffffcf30
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
0x804902a <_start+42> int    0x80

native process 6173 In: _start L9 PC: 0x8049000
(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) break *0x8049031
Note: breakpoint 2 also set at pc 0x8049031.
Breakpoint 3 at 0x8049031: file lab09-2.asm, line 20.
(gdb)

```

Рис. 4.18: 18)5 инструкций si

- 13) Посмотреть содержимое регистров также можно с помощью команды info registers

```

native process 6173 In: _start L9 PC: 0x8049000
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcf30 0xffffcf30
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 4.19: 19)info registers

- 14) Смотрю значение переменной msg1 по имени и переменной msg2 по адресу (20-21)

```

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

```

Рис. 4.20: 20)Значение переменной msg1 по имени

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

Рис. 4.21: 21)Значение переменной msg2 по адресу

15) Изменяю первый символ переменной msg1

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

Рис. 4.22: 22)Изменение первого символа переменной msg1

16) Заменяю символ во второй переменной msg2

```
(gdb) set {char}0x804a000='w'
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "world!\n\034"
(gdb) □
```

Рис. 4.23: 23)Изменение символа второй переменной

18) С помощью команды set изменяю значение регистра ebx. В первом случае выводит значение символа (его код), во втором - число

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

Рис. 4.24: 25)Изменяю значение регистра ebx

- 19) Завершаю выполнение программы с помощью команды `continue` (сокращенно `c`) и выхожу из GDB с помощью команды `quit` (сокращенно `q`)

```
(gdb) c
Continuing.
wello, world!

Breakpoint 2, _start () at lab09-2.asm:20
(gdb) q
A debugging session is active.

    Inferior 1 [process 6173] will be killed.

Quit anyway? (y or n) ☐
```

Рис. 4.25: 26)Завершаю программу

- 20) Копирую файл `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`, создаю исполняемый файл, загружаю исполняемый файл в отладчик, указав аргументы:

```
(alkamal@localhost)-[~/work/arch-pc/lab09]
$ cp ~/work/arch-pc/lab08/lab08-2.asm ~/work/arch-pc/lab09/lab09-3.asm

(alkamal@localhost)-[~/work/arch-pc/lab09]
$ nasm -f elf -g -l lab09-3.lst lab09-3.asm

(alkamal@localhost)-[~/work/arch-pc/lab09]
$ ld -m elf_i386 -o lab09-3 lab09-3.o

(alkamal@localhost)-[~/work/arch-pc/lab09]
$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Debian 13.2-1) 13.2
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-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. 4.26: 27)Копирование, создание, загрузка файла

- 21) Для начала устанавливаю точку останова перед первой инструкцией в программе и запускаю её.

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) r
Starting program: /home/alkamal/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3
Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb)

```

Рис. 4.27: 28)Точка установка и запуск

- 22) Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы)

```

(gdb) x/x $esp
0xffffcef0: 0x00000005
(gdb)

```

Рис. 4.28: 29)Регистр esp

- 23) Смотрю остальные позиции стека. Шаг изменения равен 4 потому что шаг - int, а под этот тип данных выделяется 4 байта

```

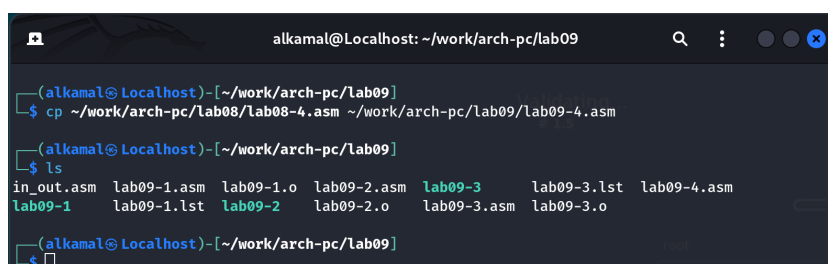
(gdb) x/s *(void**)(esp + 4)
0xffffd0e4: "/home/alkamal/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd10d: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd11f: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd130: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd132: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 4.29: 30)Остальные позиции стека

5 Выполнение заданий для самостоятельной работы

1) Копирую файл



```
alkamal@Localhost: ~/work/arch-pc/lab09
(alkamal@Localhost)~[~/work/arch-pc/lab09]
$ cp ~/work/arch-pc/lab08/lab08-4.asm ~/work/arch-pc/lab09/lab09-4.asm
(alkamal@Localhost)~[~/work/arch-pc/lab09]
$ ls
in_out.asm  lab09-1.asm  lab09-1.o  lab09-2.asm  lab09-3  lab09-3.lst  lab09-4.asm
lab09-1     lab09-1.lst  lab09-2   lab09-2.o    lab09-3.asm  lab09-3.o
```

Рис. 5.1: Копирование файла

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

Текст программы:

```
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
msg1 db "Функция:  $f(x) = 2x + 15$ ",0
SECTION .text
global _start
_start:
```

```

pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (второе значение в стеке)
sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
; аргументов без названия программы)
mov esi, 0 ; Используем `esi` для хранения
; промежуточных сумм
next:
cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
call _function
loop next
_end:
mov eax, msg1
call sprintf
mov eax, msg ; вывод сообщения "Результат: "
call sprintf
mov eax, esi ; записываем сумму в регистр `eax`
call iprintf ; печать результата
call quit ; завершение программы
csedit lab09-4.asm
_function:
mov ebx, 2
mul ebx
add eax, 15
add esi,eax ; добавляем к промежуточной сумме

```

; след. аргумент `esi=esi+eax`

ret

```
(alkamal@Localhost)-[~/work/arch-pc/lab09]
$ nasm -f elf lab09-4.asm
(alkamal@Localhost)-[~/work/arch-pc/lab09]
$ ld -m elf_i386 -o lab09-4 lab09-4.o
(alkamal@Localhost)-[~/work/arch-pc/lab09]
$ ./lab09-4 1 2 3 4 5
Функция: f(x) = 2x + 15
Результат: 105
(alkamal@Localhost)-[~/work/arch-pc/lab09]
$
```

Рис. 5.2: Работа файла

3) Создаю файл, ввожу туда текст программы

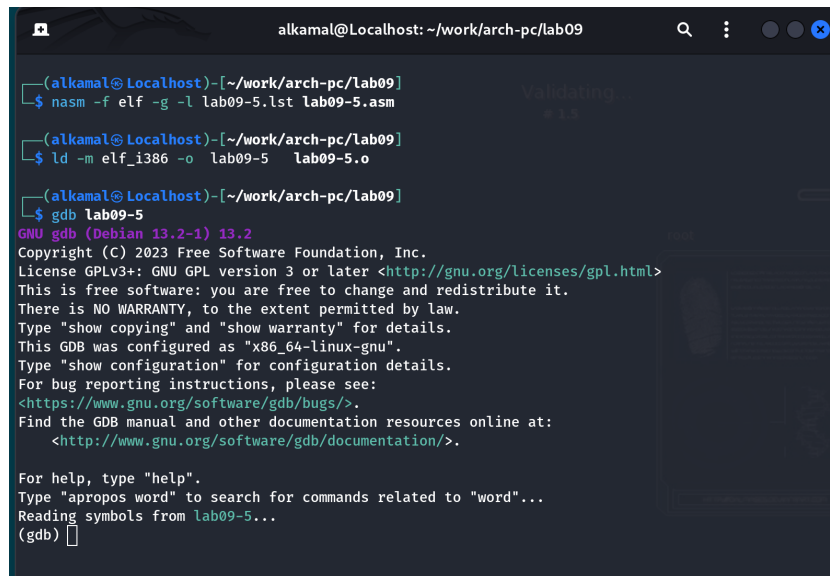
```
alkamal@Localhost: ~/work/arch-pc/lab09
(alkamal@Localhost)-[~/work/arch-pc/lab09]
$ touch lab09-5.asm
(alkamal@Localhost)-[~/work/arch-pc/lab09]
$ ls
in_out.asm  lab09-1.lst  lab09-2.asm  lab09-3.asm  lab09-4  lab09-5.asm
lab09-1  lab09-1.o  lab09-2.o  lab09-3.lst  lab09-4.asm
lab09-1.asm  lab09-2  lab09-3  lab09-3.o  lab09-4.o
(alkamal@Localhost)-[~/work/arch-pc/lab09]
$
```

Рис. 5.3: Создание файла

```
GNU nano 7.2 /home/alkamal/work/arch-pc/lab09/lab09-5.asm
#include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```

Рис. 5.4: Текст файла

4) Передаю файл на отладку



```
alkamal@Localhost: ~/work/arch-pc/lab09
(alkamal@Localhost)~-[~/work/arch-pc/lab09]
$ nasm -f elf -g -l lab09-5.lst lab09-5.asm
Validating
* 1.5

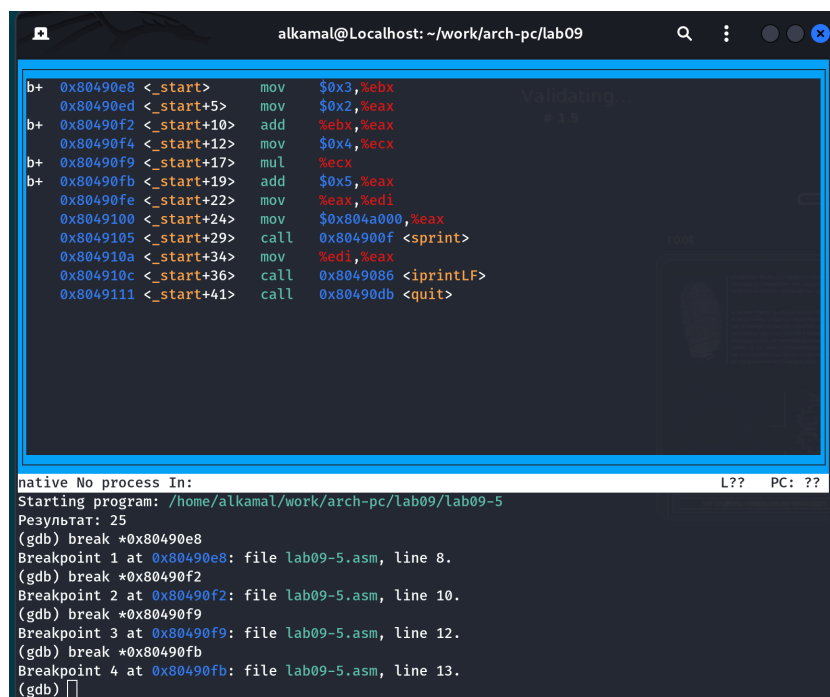
(alkamal@Localhost)~-[~/work/arch-pc/lab09]
$ ld -m elf_i386 -o lab09-5 lab09-5.o

(alkamal@Localhost)~-[~/work/arch-pc/lab09]
$ gdb lab09-5
GNU gdb (Debian 13.2-1) 13.2
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-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 lab09-5...
(gdb) 
```

Рис. 5.5: Передача файла

5) Устанавливаю точки останова

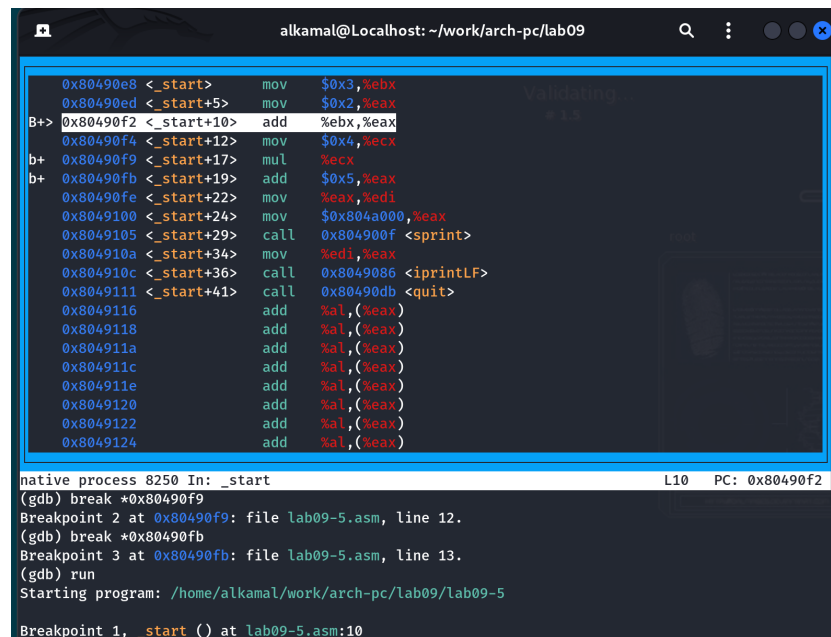


```
alkamal@Localhost: ~/work/arch-pc/lab09
b+ 0x80490e8 <_start>    mov     $0x3,%ebx
0x80490ed <_start+5>    mov     $0x2,%eax
b+ 0x80490f2 <_start+10> add     %ebx,%eax
0x80490f4 <_start+12>    mov     $0x4,%ecx
b+ 0x80490f9 <_start+17>    mul     %ecx
b+ 0x80490fb <_start+19>    add     $0x5,%eax
0x80490fe <_start+22>    mov     %eax,%edi
0x8049100 <_start+24>    mov     $0x804a000,%eax
0x8049105 <_start+29>    call    0x804900f <sprint>
0x804910a <_start+34>    mov     %edi,%eax
0x804910c <_start+36>    call    0x8049086 <iprintLF>
0x8049111 <_start+41>    call    0x80490db <quit>
Validating
* 1.5

native No process in:
Starting program: /home/alkamal/work/arch-pc/lab09/lab09-5
Результат: 25
(gdb) break *0x80490e8
Breakpoint 1 at 0x80490e8: file lab09-5.asm, line 8.
(gdb) break *0x80490f2
Breakpoint 2 at 0x80490f2: file lab09-5.asm, line 10.
(gdb) break *0x80490f9
Breakpoint 3 at 0x80490f9: file lab09-5.asm, line 12.
(gdb) break *0x80490fb
Breakpoint 4 at 0x80490fb: file lab09-5.asm, line 13.
(gdb) 
```

Рис. 5.6: Точки останова

- 6) Прохожу по программе обращая внимание на несостыковки с логикой. Таким образом нашла 3 ошибки



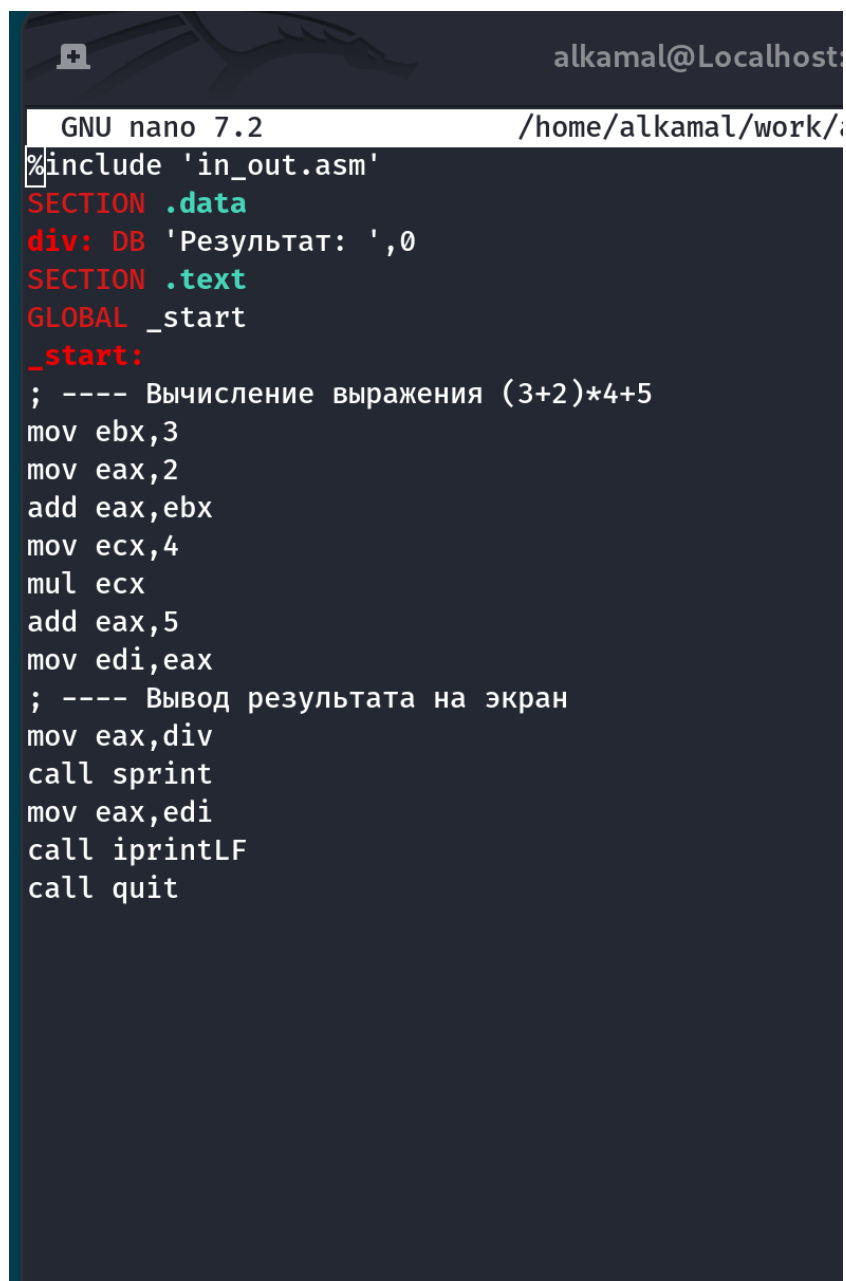
The screenshot shows a GDB debugger window with the following assembly code and breakpoints:

```
0x80490e8 <_start> mov $0x3,%ebx
0x80490ed <_start+5> mov $0x2,%eax
B-> 0x80490f2 <_start+10> add %ebx,%eax
0x80490f4 <_start+12> mov $0x4,%ecx
b+ 0x80490f9 <_start+17> mul %ecx
b+ 0x80490fb <_start+19> add $0x5,%eax
0x80490fe <_start+22> mov %eax,%edi
0x8049100 <_start+24> mov $0x804a000,%eax
0x8049105 <_start+29> call 0x804900f <sprint>
0x804910a <_start+34> mov %edi,%eax
0x804910c <_start+36> call 0x8049086 <iprintLF>
0x8049111 <_start+41> call 0x80490db <quit>
0x8049116 add %al,(%eax)
0x8049118 add %al,(%eax)
0x804911a add %al,(%eax)
0x804911c add %al,(%eax)
0x804911e add %al,(%eax)
0x8049120 add %al,(%eax)
0x8049122 add %al,(%eax)
0x8049124 add %al,(%eax)
```

Breakpoints are set at 0x80490f9, 0x80490fb, and 0x80490fb. The program is running at address 0x80490f2.

Рис. 5.7: Отладка файла

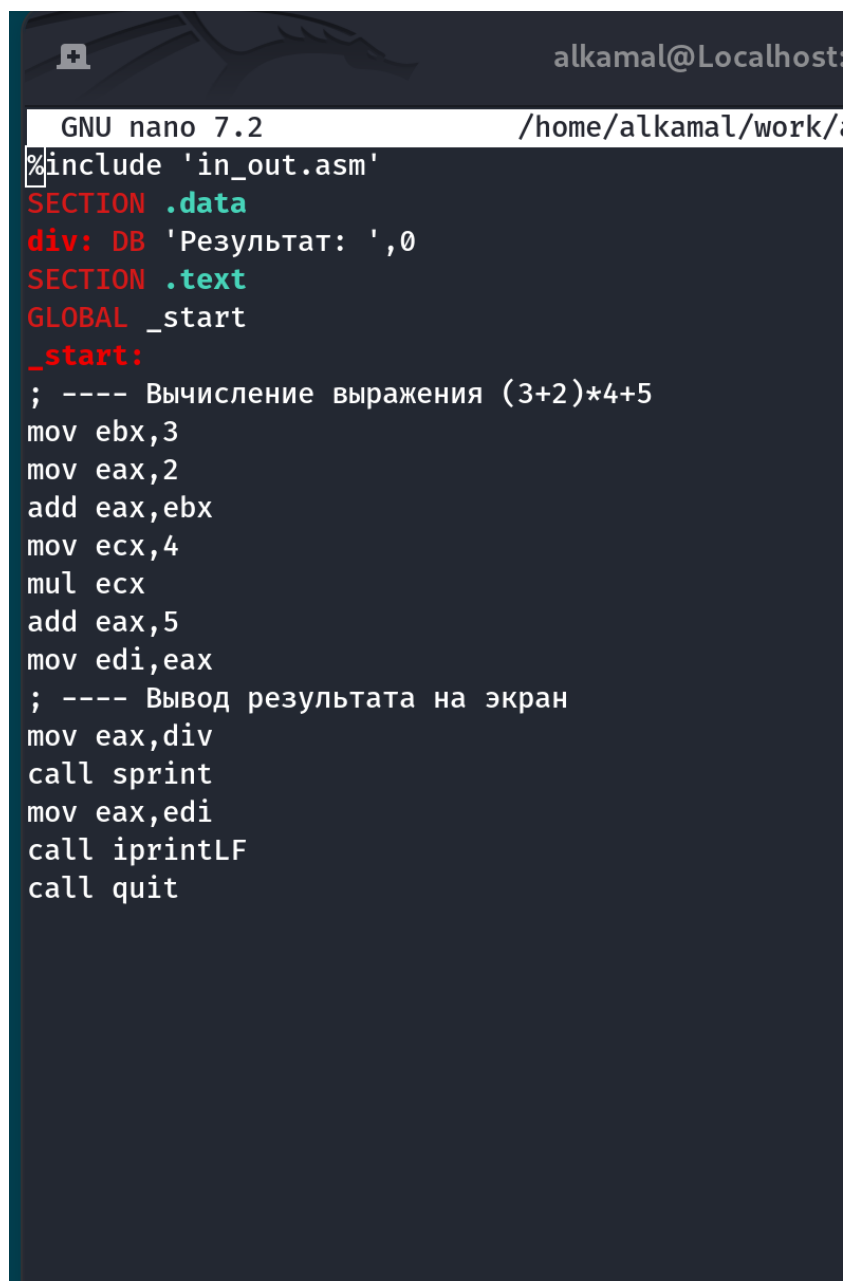
- 7) Исправляю ошибки



```
alkamal@Localhost:~$ nano /home/alkamal/work/
GNU nano 7.2 /home/alkamal/work/
%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax
; ---- Вывод результата на экран
mov eax,div
call sprintf
mov eax,edi
call iprintLF
call quit
```

Рис. 5.8: Исправленный текст файла

8) Работа файла



```
alkamal@Localhost:~$ nano /home/alkamal/work/
GNU nano 7.2 /home/alkamal/work/
%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax
; ---- Вывод результата на экран
mov eax,div
call sprintf
mov eax,edi
call iprintLF
call quit
```

Рис. 5.9: Работа файла

6 Выводы

Приобрела навыки написания программ с использованием подпрограмм. Познакомилась с методами отладки при помощи GDB и его основными возможностями.

Список литературы