

Отчёт по лабораторной работе №9

Дисциплина: архитектура компьютера

Незами Ахмад Белал

Содержание

| | | |
|---|------------------------------------|----|
| 1 | Цель работы | 5 |
| 2 | Выполнение лабораторной работы | 6 |
| 3 | Задания для самостоятельной работы | 17 |
| 4 | Выводы | 20 |

Список иллюстраций

| | | |
|------|---|----|
| 2.1 | Создаем директорию и файл | 6 |
| 2.2 | Запись программы | 7 |
| 2.3 | Проверка | 7 |
| 2.4 | Создание файла lab09-2.asm | 8 |
| 2.5 | Запись программы | 8 |
| 2.6 | Запуск программы | 9 |
| 2.7 | Поставим точку останова(breakpoint) на метке _start | 10 |
| 2.8 | дизассемблированный код | 11 |
| 2.9 | Команда i b | 12 |
| 2.10 | | 12 |
| 2.11 | | 13 |
| 2.12 | | 13 |
| 2.13 | | 13 |
| 2.14 | | 14 |
| 2.15 | | 15 |
| 2.16 | | 15 |
| 2.17 | | 16 |
| 3.1 | | 17 |
| 3.2 | | 18 |
| 3.3 | Программа работает правильно | 18 |
| 3.4 | Проверка | 19 |

Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

1

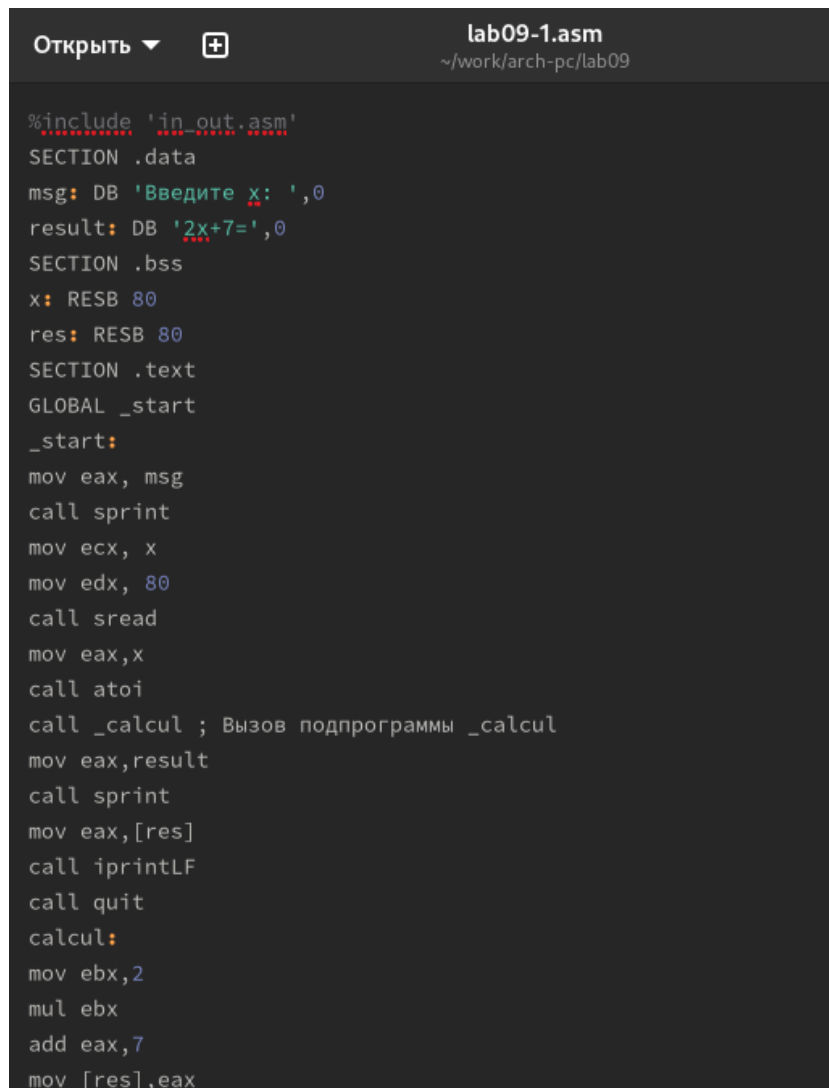
Создадим рабочую директорию и файл.(рис. [2.1])

```
[nezami.a@fedora ~]$ mkdir ~/work/arch-pc/lab09  
[nezami.a@fedora ~]$ cd ~/work/arch-pc/lab09  
[nezami.a@fedora lab09]$ touch lab09-1.asm  
[nezami.a@fedora lab09]$
```

Рис. 2.1: Создаем директорию и файл

2

Напишем программу, имитирующую сложную функцию. Функции назовем _calul и subcalcul.(рис. [2.2])




```
Открыть ▾  lab09-1.asm  
~/work/arch-pc/lab09  
  
%include 'in_out.asm'  
SECTION .data  
msg: DB 'Введите x: ',0  
result: DB '2x+7=',0  
SECTION .bss  
x: RESB 80  
res: RESB 80  
SECTION .text  
GLOBAL _start  
_start:  
mov eax, msg  
call sprint  
mov ecx, x  
mov edx, 80  
call sread  
mov eax, x  
call atoi  
call _calcul ; Вызов подпрограммы _calcul  
mov eax, result  
call sprint  
mov eax, [res]  
call iprintLF  
call quit  
calcul:  
mov ebx, 2  
mul ebx  
add eax, 7  
mov [res], eax
```

Рис. 2.2: Запись программы

3

Проверим ее работу (рис. [-2.3)



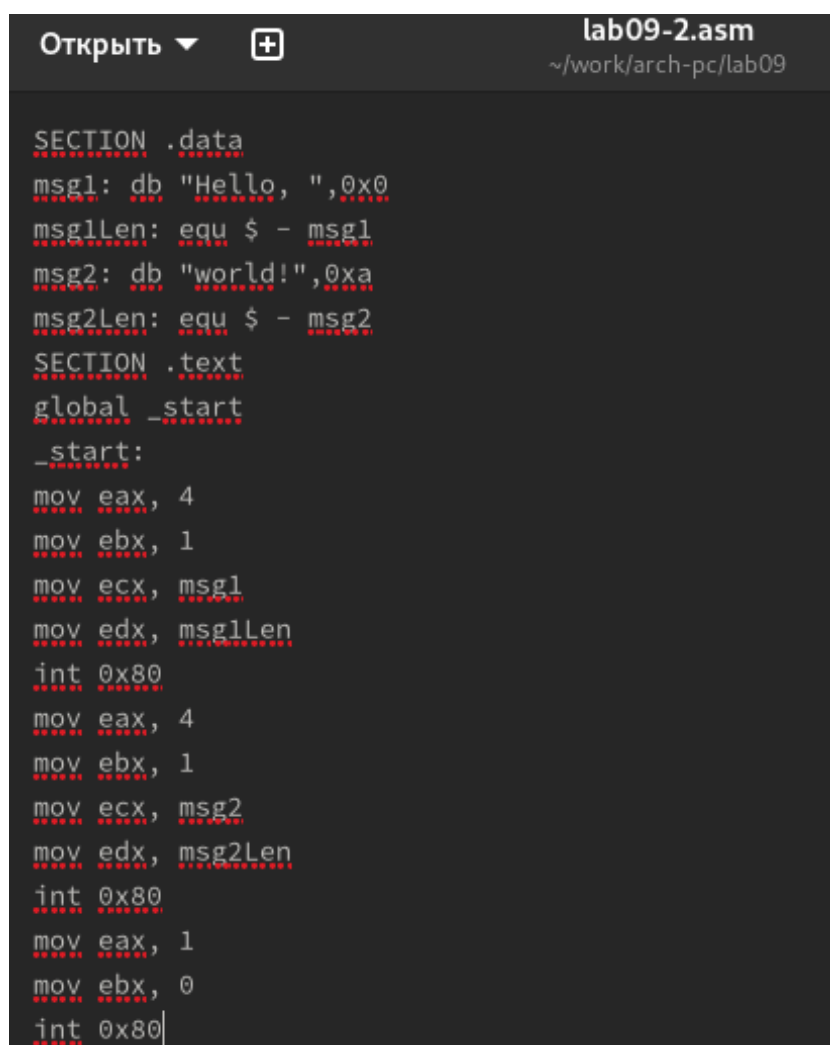
```
Введите x: 1  
2x+7=9
```

Рис. 2.3: Проверка

Создадим файл lab09-2.asm и посмотрим, как она работает. Так же проасSEMBлируем его с другими ключами, чтобы была возможность открыть этот файл через gdb. (рис. [2.4])

```
[nezami.a@fedora lab09]$ touch lab09-2.asm
[nezami.a@fedora lab09]$
```

Рис. 2.4: Создание файла lab09-2.asm



```
lab09-2.asm
~/work/arch-pc/lab09

SECTION .data
msg1: db "Hello, ",0x0
msg1Len: equ $ - msg1
msg2: db "world!",0xa
msg2Len: equ $ - msg2
SECTION .text
global _start
_start:
mov eax, 4
mov ebx, 1
mov ecx, msg1
mov edx, msg1Len
int 0x80
mov eax, 4
mov ebx, 1
mov ecx, msg2
mov edx, msg2Len
int 0x80
mov eax, 1
mov ebx, 0
int 0x80
```

Рис. 2.5: Запись программы

5

Откроем lab09-2 с помощью gdb. Запустим ее там(рис. [2.6])

```
$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
$ ld -m elf_i386 -o lab09-2 lab09-2.o
$ gdb lab09-2
GNU gdb (Ubuntu 13.1-2.fc38)
Copyright (C) 2023 Free Software Foundation, Inc.
This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License as
published by the Free Software Foundation; either version 3 or
later, or (at your option) any later version.
You should have received a copy of the GNU General Public License
along with this program; see the file COPYING. If not, see
<http://www.gnu.org/licenses/gpl.html>.
Please report bugs to bugreport@gnu.org.
For more information about this program, please see:
  <http://www.gnu.org/software/gdb/bugs/>.
Other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

Search for commands related to "word"...
lab09-2...

ayanchberdyev/work/arch-pc/lab09/lab09-2

Downloading debuginfo from the following URLs:
  <http://debuginfod.ubuntu.com/>
Is this session? (y or [n]) y
Loaded.
To make this session permanent, add 'set debuginfod enabled on' to .gdbinit.
Debug info for system-supplied DSO at 0xf7ffc000

(gdb) exited normally]
```

Рис. 2.6: Запуск программы

6

Поставим точку останова(breakpoint) на метке _start. Посмотрим дизассемблированный код, начиная с этой метки. (рис. [2.7])

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1b0 0xffffd1b0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

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

native process 3790 In: _start L9 PC: 0x8049000
The history is empty.
The history is empty.
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/dayanchberdyev/work/arch-pc/lab09/lab09-2

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

Рис. 2.7: Поставим точку останова(breakpoint) на метке _start

7

Так же посмотрим как выглядит дизассемблированный код с синтаксисом Intel (рис. [2.9])

```

[ Register Values Unavailable ]

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
0x8049038             add     BYTE PTR [eax],al

native process 3790 In: _start L9 PC: 0x8049000
(gdb) disassemble _start
(gdb) layout asm
(gdb) layout regs
(gdb) b *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)

```

Рис. 2.8: дизассемблированный код

8

В представлении АТТ в виде 16-ричного числа записаны первые аргументы всех команд, а в представлении intel так записываются адреса вторых аргументов.

Включим режим псевдографики, с помощью которого отображается код программы и содержимое регистров.

Посмотрим информацию о наших точках останова. Сделать это можно коротко командой `i b` (рис. [2.9])

```

[ Register Values Unavailable ]

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
0x8049038             add     BYTE PTR [eax],al

native process 3790 In: _start L9 PC: 0x8049000
(gdb) disassemble _start
(gdb) layout asm
(gdb) layout regs
(gdb) b *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)

```

Рис. 2.9: Команда i b

9

В отладчике можно вывести текущее значение переменных. Сделать это можно например по имени или по адресу (рис. [2.10])

```

(gdb) si
(gdb) x/1sb & msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb)

```

Рис. 2.10:

10

Так же отладчик позволяет менять значения переменных прямо во время выполнения программы (рис. [2.11])

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

Рис. 2.11:

11

Здесь тоже можно обращаться по адресам переменных(рис. [2.12]). здесь был заменен первый символ переменной msg2 на символ отступа.

```
(gdb) set {char}&msg2=9
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "\torld!\n\034"
(gdb)
```

Рис. 2.12:

12

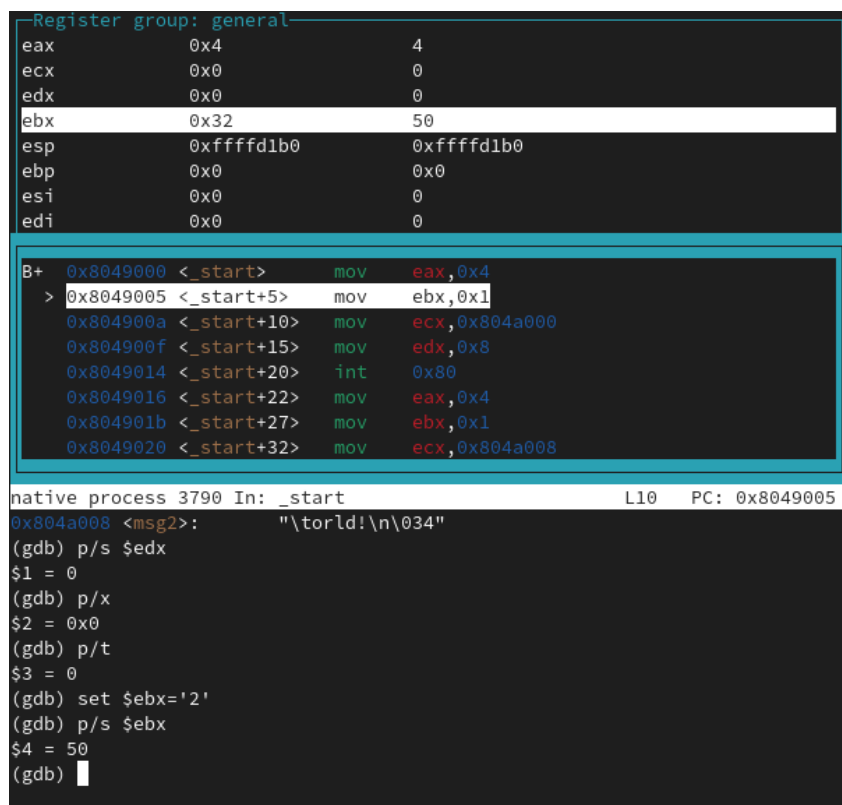
Выводить можно так же содержимое регистров. Выведем значение edx в разных форматах: строчном, 16-ричном, двоичном(рис. [2.14])

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

Рис. 2.13:

13

Как и переменным, регистрам можно задавать значения.(рис. [??])



```
Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x32      50
esp      0xffffd1b0 0xffffd1b0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

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

native process 3790 In: _start L10 PC: 0x8049005
0x804a008 <msg2>: "\world!\n\034"
(gdb) p/s $edx
$1 = 0
(gdb) p/x
$2 = 0x0
(gdb) p/t
$3 = 0
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) 
```

Рис. 2.14:

14

Скопируем файл из лабораторной 9, переименуем и создадим исполняемый файл. Откроем отладчик и зададим аргументы. Создадим точку останова на метке `_start` и запустим программу(рис. [2.15])

```

GNU gdb (GDB) Fedora Linux 13.1-2.fc38
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-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...
(gdb) b _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/dayanchberdyev/work/arch-pc/lab09/lab09-2 arg1 arg 2 arg

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 lab09-2.asm:9
9      mov eax, 4
(gdb)

```

Рис. 2.15:

15

Посмотрим на содержимое того, что расположено по адресу, находящемуся в регистре esp (рис. [2.16])

```

(gdb) x/x $esp
0xffffd190:    0x00000005
(gdb)

```

Рис. 2.16:

16

Далее посмотрим на все остальные аргументы в стеке. Их адреса располагаются в 4 байтах друг от друга (именно столько занимает элемент стека) (рис. [2.17])

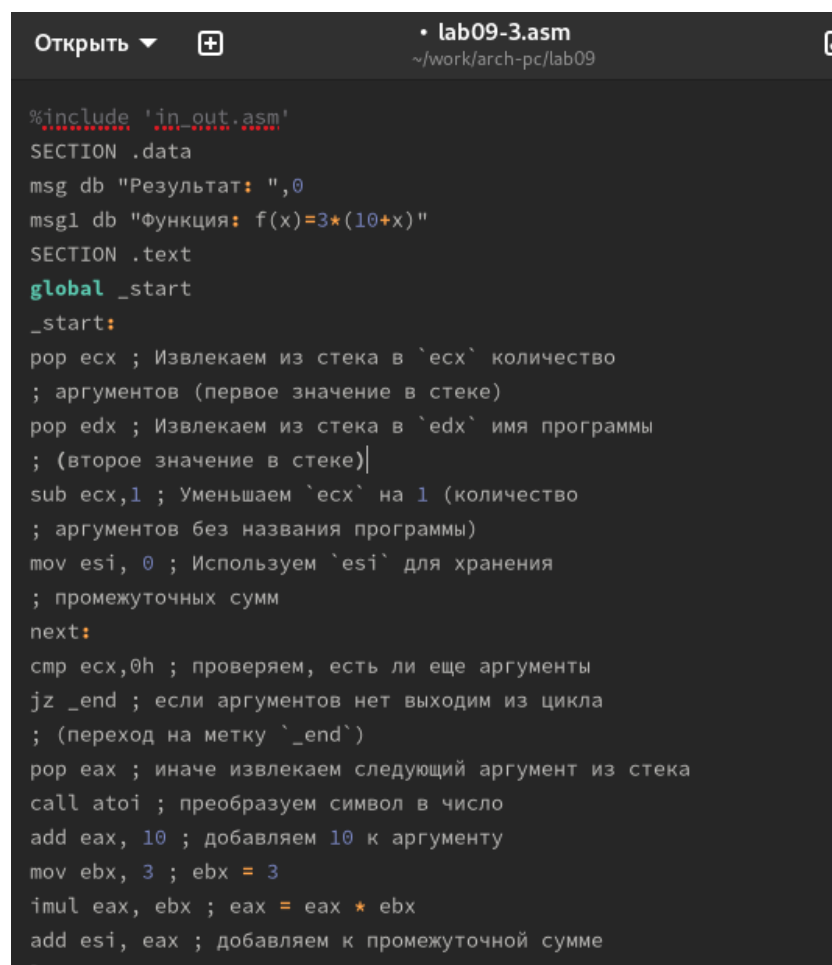
```
(gdb) x/s *(void**)($esp + 4)
0xffffd331:    "/home/dayanchberdyev/work/arch-pc/lab09/lab09-2"
(gdb) x/s *(void**)($esp + 8)
0xffffd361:    "arg1"
(gdb) x/s *(void**)($esp + 12)
0xffffd366:    "arg"
(gdb) x/s *(void**)($esp + 16)
0xffffd36a:    "2"
(gdb) x/s *(void**)($esp + 20)
0xffffd36c:    "arg3"
(gdb) x/s *(void**)($esp + 24)
0x0:    <error: Cannot access memory at address 0x0>
(gdb) █
```

Рис. 2.17:

3 Задания для самостоятельной работы

17

Программа из лабораторной 9, но с использованием подпрограмм (рис. [3.1])



```
Открыть ▾ + lab09-3.asm
~/work/arch-pc/lab09

%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
msg1 db "Функция: f(x)=3*(10+x)"
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
add eax, 10 ; добавляем 10 к аргументу
mov ebx, 3 ; ebx = 3
imul eax, ebx ; eax = eax * ebx
add esi, eax ; добавляем к промежуточной сумме
```

Рис. 3.1:

18

Проверка ее работоспособности(рис. [3.2])

```
Функция: f(x)=3*(10+x)
Результат: 0
[dayanchberdyev@fedora lab09]$ ./lab09-3 1 2 3 4
Функция: f(x)=3*(10+x)
Результат: 150
```


Рис. 3.2:

Просмотр регистров, для поиска ошибки в программе из листинга

Ошибка была в строчках

add ebx,eax mov ecx,4 mul ecx add ebx,5 mov edi,ebx

Правильно работающая программа представлена на (рис. [3.3])

```
Открыть ▾  lab09-3.asm
~/work/arch-pc/lab09

%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
msg1 db "Функция: f(x)=3*(10+x)"
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
    add eax, 10 ; добавляем 10 к аргументу
    mov ebx, 3 ; ebx = 3
    imul eax, ebx ; eax = eax * ebx
    add esi, eax ; добавляем к промежуточной сумме
    loop next ; переход к обработке следующего аргумента
```

Рис. 3.3: Программа работает правильно

Проверка корректности работы программы, после исправлений (рис. [3.4])

[illegible]

Рис. 3.4: Проверка

4 Выводы

В результате выполнения работы, я научился организовывать код в подпрограммы и познакомился с базовыми функциями отладчика gdb.