

# Отчёт по лабораторной работе №9

Дисциплина: Архитектура компьютера

Студент: Верниковская Екатерина Андреевна

# Содержание

Цель работы	6
Задание	7
Выполнение лабораторной работы	8
Реализация подпрограмм в NASM . . . . .	8
Отладка программ с помощью GDB . . . . .	14
Добавление точек останова . . . . .	21
Работа с данными программы в GDB . . . . .	22
Обработка аргументов командной строки в GDB . . . . .	31
Задание для самостоятельной работы . . . . .	33
Выводы	41

## Список таблиц

## Список иллюстраций

1	Создание первого файла . . . . .	8
2	Копирование файла «in_out.asm» . . . . .	9
3	Ввод текста программы с подпрограммой _calcul . . . . .	11
4	Создание исполняемого файла и его запуск . . . . .	11
5	Изменение программы . . . . .	13
6	Исполняемый файл + запуск . . . . .	14
7	Создание файла «lab9-2.asm» . . . . .	14
8	Ввод текста программы . . . . .	15
9	Создание исполняемого файла с ключом '-g' . . . . .	15
10	Загрузка исполняемого файла в gdb . . . . .	16
11	Проверка работы программы в оболочке gdb . . . . .	16
12	Установка брейкпоинта и запуск . . . . .	16
13	Дисассимилированный код . . . . .	17
14	Отображение команд с Intel'овским синтаксисом . . . . .	18
15	Использование команды 'layout asm' . . . . .	20
16	Использование команды 'layout regs' . . . . .	21
17	Использование команды 'info breakpoints' . . . . .	21
18	Установка точки останова . . . . .	22
19	Информация о всех точек останова . . . . .	22
20	Команда 'stepi' 1 . . . . .	23
21	Команда 'stepi' 2 . . . . .	24
22	Команда 'stepi' 3 . . . . .	25
23	Команда 'stepi' 4 . . . . .	26
24	Команда 'stepi' 5 . . . . .	27
25	Смотрим содержимое регистров . . . . .	27
26	Значение переменной msg1 по имени . . . . .	28
27	Значение переменной msg2 по адресу . . . . .	28
28	Инструкцию «mov ecx,msg2» . . . . .	28
29	Изменение первого символа переменной msg1 . . . . .	28
30	Изменение первого символа переменной ms g2 . . . . .	29
31	Значение регистра edx в различных форматах . . . . .	29
32	Изменение значение регистра ebx 1 . . . . .	29
33	Изменение значение регистра ebx 2 . . . . .	30
34	Завершение программы . . . . .	30
35	Выход 1 . . . . .	30
36	Выход 2 . . . . .	31
37	Создание файла «lab9-3.asm» . . . . .	31

38	Создание исполняемого файла . . . . .	31
39	Загрузка в gdb с использованием ‘-args’ . . . . .	32
40	Точка останова + запуск программы . . . . .	32
41	Адрес вершины стека . . . . .	32
42	Остальные позиции стека . . . . .	33
43	Создание файла «lab9-4.asm» . . . . .	33
44	Написание программы . . . . .	35
45	Создание исполняемого файла и его запуск . . . . .	35
46	Создание файла «lab9-5.asm» . . . . .	35
47	Ввод текста программы . . . . .	37
48	Создание исполняемого файла . . . . .	37
49	Загрузка исполняемого файла в gdb . . . . .	37
50	Проверка работы программы в оболочке gdb . . . . .	38
51	Ищем ошибку регистров . . . . .	38
52	Исправленная программа . . . . .	39
53	Проверка программы . . . . .	40

## Цель работы

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

# Задание

1. Создать каталог для программ лабораторной работы №9 и в нём создать файл «lab9-1.asm».
2. Ввести в файл «lab9-1.asm» определённый текст программы с подпрограммой `_calcul`. Создать исполняемый файл и запустить его.
3. Изменить текст программы, добавив подпрограмму `_subcalcul`. Снова создать исполняемый файл и запустить его.
4. Опять изменить текст программы, добавив команды `push` и `pop`, создать исполняемый файл и запустить его.
5. Создать файл «lab9-2.asm» и ввести в него определённый текст программы (Программа печати сообщения Hello world!).
6. Создать исполняемый файл с отладочной информацией.
7. Сделать задания, которые помогут лучше понять структуру GDB.
8. Преобразовать программу из лабораторной работы №8 (задание №1 для самостоятельной работы), реализовав вычисление значения функции  $f(x)$  как подпрограмму.

Задание №1 для самостоятельной работы: Написать программу, которая находит сумму значений функции  $f(x)$  для  $x = x_1, x_2, \dots, x_n$ . Программа должна выводить значение  $f(x_1) + f(x_2) + \dots + f(x_n)$ . Вид функции брать из определённой таблицы, в соответствии с полученным вариантом (В нашем случае это 17 вариант).

9. Создать файл и ввести в него программу, которая вычисляет выражение  $(3+2)*4+5$ . Спомощью отладчика GDB определить ошибку и исправить её.

# Выполнение лабораторной работы

## Реализация подпрограмм в NASM

В созданном каталоге «~/work/arch-pc/lab09» создаём файл «lab9-1.asm» (рис. [-@fig:001])

```
eavernikovskaya@ubuntu-katerok:~$ mkdir ~/work/arch-pc/lab09
eavernikovskaya@ubuntu-katerok:~$ cd ~/work/arch-pc/lab09
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ touch lab9-1.asm
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ ls
lab9-1.asm
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$
```

Рис. 1: Создание первого файла

Копируем из каталога «~/work/arch-pc/lab08» файл «in\_out.asm» (рис. [-@fig:002])



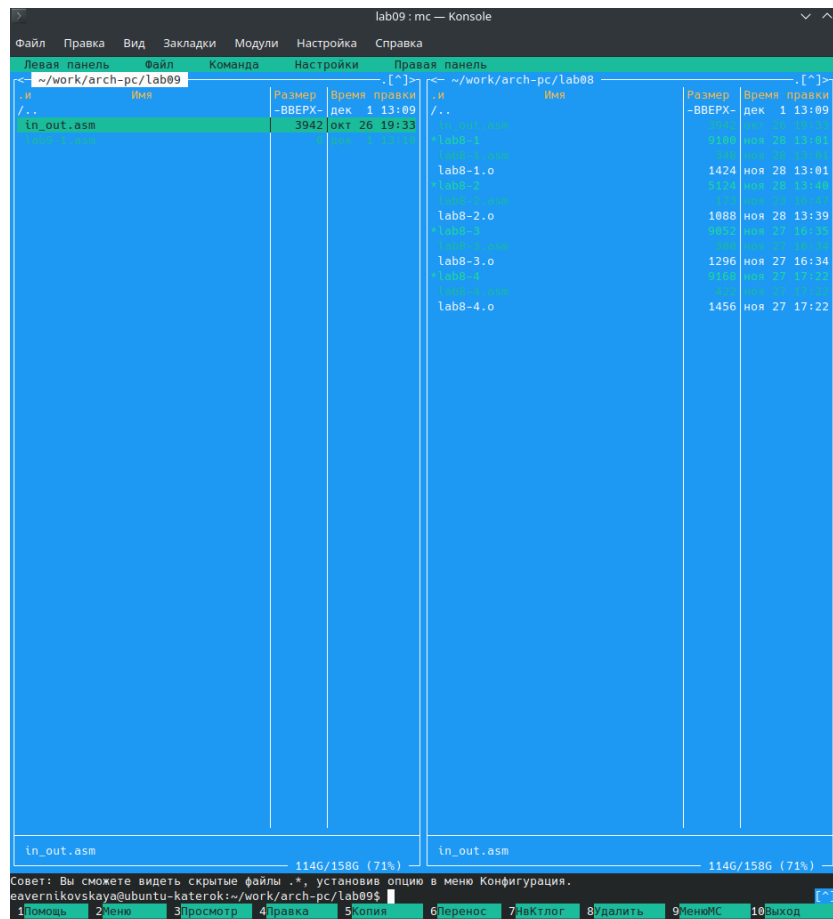


Рис. 2: Копирование файла «in\_out.asm»

Вводим нужный текст программы, которая вычисляет арифмитическое выражение  $f(x)=2x+7$  с помощью подпрограммы `_calcul` (рис. [-@fig:003])

Текст программы:

```
%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
    mul ebx
    add eax,7
    mov [res],eax
    ret

```

```

lab09 : mc — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
GNU nano 6.2 /home/eavernikovskaya/work/arch-pc/lab09/lab9-1.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:
    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
        mul ebx
        add eax,7
        mov [res],eax
        ret

```

Рис. 3: Ввод текста программы с подпрограммой `_calcul`

Создаём исполняемый файл и запускаем его (рис. [-@fig:004])

```

eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 5
2x+7=17
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 2
2x+7=11
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$

```

Рис. 4: Создание исполняемого файла и его запуск

Изменяем текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения  $f(g(x))$ .  $f(x)=2x+7$  а  $g(x)=3x-1$  (рис. [-@fig:005])

Изменённый текст программы:

```

%include 'in_out.asm'

SECTION .data
msg1: DB 'f(x)=2x+7; g(x)=3x-1 ',0
msg: DB 'Введите x: ',0

```

```
result: DB 'f(g(x))=',0
```

```
SECTION .bss
```

```
x: RESB 80
```

```
res: RESB 80
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
    mov eax,msg1
```

```
    call sprintLF
```

```
    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:
```

```
    push eax
```

```
    call _subcalcul
```

```
    pop ecx
```

```
    mov ebx,2
```

```
    mul ebx
```

```
    add eax,7
```

```

    mov [res],eax

    ret

_subcalcul:
    mov ebx,3
    mul ebx
    sub eax,1
    ret

```

```

lab09: mc — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
GNU nano 6.2 /home/eavernikovskaya/work/arch-pc/lab09/lab9-1.asm
#include 'in_out.asm'
SECTION .data
msg1: DB 'f(x)=2x+7; g(x)=3x-1',0
msg: DB 'Введите x: ',0
result: DB 'f(g(x))=',0
SECTION .bss
x: RESB 80
res: RESB 80
SECTION .text
GLOBAL _start
_start:
    mov eax,msg1
    call sprintf
    mov eax,msg
    call sprintf
    mov ecx,x
    mov edx,80
    call sread
    mov eax,x
    call atoi
    call _calcul
    mov eax,result
    call sprintf
    mov eax,[res]
    call iprintLF
    call quit
    _calcul:
        push eax
        call _subcalcul
        pop ecx
        mov ebx,2
        mul ebx
        add eax,7
        mov [res],eax
        ret
    _subcalcul:
        mov ebx,3
        mul ebx
        sub eax,1
        ret

```

Рис. 5: Изменение программы

Снова создаём исполняемый файл и запускаем его (рис. [-@fig:006])

```

eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ ./lab9-1
f(x)=2x+7; g(x)=3x-1
Введите x: 2
f(g(x))=17
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ ./lab9-1
f(x)=2x+7; g(x)=3x-1
Введите x: 8
f(g(x))=53
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$

```

Рис. 6: Исполняемый файл + запуск

## Отладка программ с помощью GDB

Создаём файл «lab9-2.asm» (рис. [-@fig:007])

```

eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ touch lab9-2.asm
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ ls
in_out.asm lab9-1 lab9-1.asm lab9-1.o lab9-2.asm
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$

```

Рис. 7: Создание файла «lab9-2.asm»

Вводим текст программы - программа печати сообщения Hello world! (рис. [-@fig:008])

Текст программы:

```

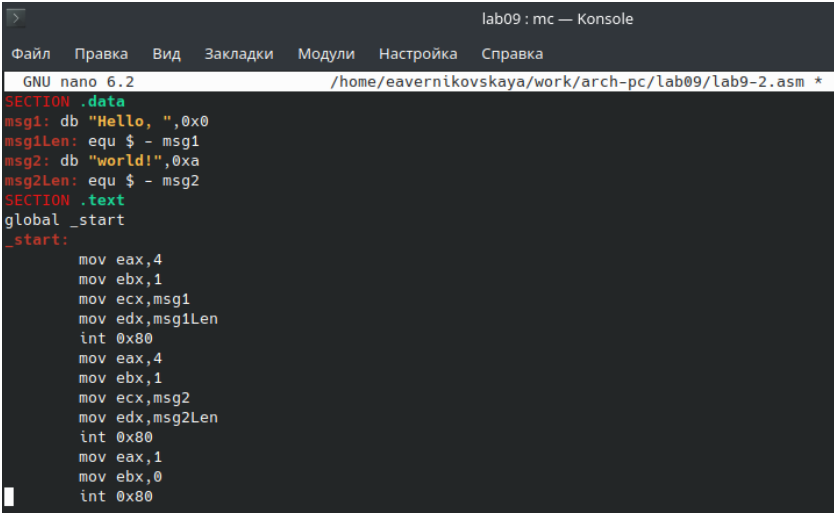
SECTION .data
msg1: db "Hello, ",0x0
msg1Len: equ $ - msg1
msg2: db "world!",0xa
msg2Len: equ $ - msg2
SECTION .text
global _start
_start:
    mov eax,4
    mov ebx,1
    mov ecx,msg1

```

```

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

```



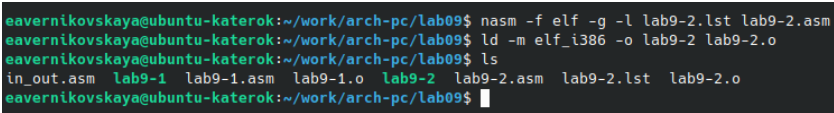
```

lab09 : mc — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
GNU nano 6.2 /home/eavernikovskaya/work/arch-pc/lab09/lab9-2.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. 8: Ввод текста программы

Создаём исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию. Для этого трансляцию программ мы проводим с ключом ‘-g’ (рис. [-@fig:009])



```

eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ ls
in_out.asm  lab9-1  lab9-1.asm  lab9-1.o  lab9-2  lab9-2.asm  lab9-2.lst  lab9-2.o
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$

```

Рис. 9: Создание исполняемого файла с ключом ‘-g’

Загружаем исполняемый файл в отладчик gdb (рис. [-@fig:010])

```
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ gdb lab9-2
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 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 lab9-2...
(gdb) █
```

Рис. 10: Загрузка исполняемого файла в gdb

Проверяем работу программы, запустив её в оболочке GDB с помощью команды ‘run’ (рис. [-@fig:011])

```
(gdb) run
Starting program: /home/eavernikovskaya/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 6932) exited normally]
(gdb) █
```

Рис. 11: Проверка работы программы в оболочке gdb

Для более подробного анализа программы устанавливаем брейкпоинт на метку «\_start», с которой начинается выполнение любой ассемблерной программы, и запускаем её (рис. [-@fig:012])

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 9.
(gdb) run
Starting program: /home/eavernikovskaya/work/arch-pc/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:9
9      mov eax,4
(gdb) █
```

Рис. 12: Установка брейкпоинта и запуск



Смотрим дисассимилированный код программы с помощью команды 'disassemble' начиная с метки «\_start» (рис. [-@fig:013])

```
(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. 13: Дисассимилированный код

Переключаемся на отображение команд с Intel'овским синтаксисом, введя команду 'set disassembly-flavor intel' (рис. [-@fig:014])

```

(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. 14: Отображение команд с Intel'овским синтаксисом

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 “\$”.
- 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 для более удобного анализа программы (рис. [-@fig:015]), (рис. [-@fig:016])

The screenshot shows a GDB console window titled 'lab09: gdb — Konsole'. The menu bar includes 'Файл', 'Правка', 'Вид', 'Закладки', 'Модули', 'Настройка', and 'Справка'. The main area displays '[ Register Values Unavailable ]'. Below this, a list of assembly instructions is shown, each with its address, a comment, and the instruction itself. The instructions are:   
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   
0x8049031 <\_start+49> mov ebx,0x0   
0x8049036 <\_start+54> int 0x80   
0x8049038 add BYTE PTR [eax],al   
0x804903a add BYTE PTR [eax],al   
0x804903c add BYTE PTR [eax],al   
0x804903e add BYTE PTR [eax],al   
0x8049040 add BYTE PTR [eax],al   
At the bottom, the status bar shows 'native process 7067 In: \_start', '(gdb) layout regs', 'L9', and 'PC: 0x8049000'.

```
lab09: gdb — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка

[ Register Values Unavailable ]

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
0x8049031 <_start+49> mov ebx,0x0
0x8049036 <_start+54> int 0x80
0x8049038 add BYTE PTR [eax],al
0x804903a add BYTE PTR [eax],al
0x804903c add BYTE PTR [eax],al
0x804903e add BYTE PTR [eax],al
0x8049040 add BYTE PTR [eax],al

native process 7067 In: _start          L9  PC: 0x8049000
(gdb) layout regs
(gdb) 
```

Рис. 15: Использование команды 'layout asm'

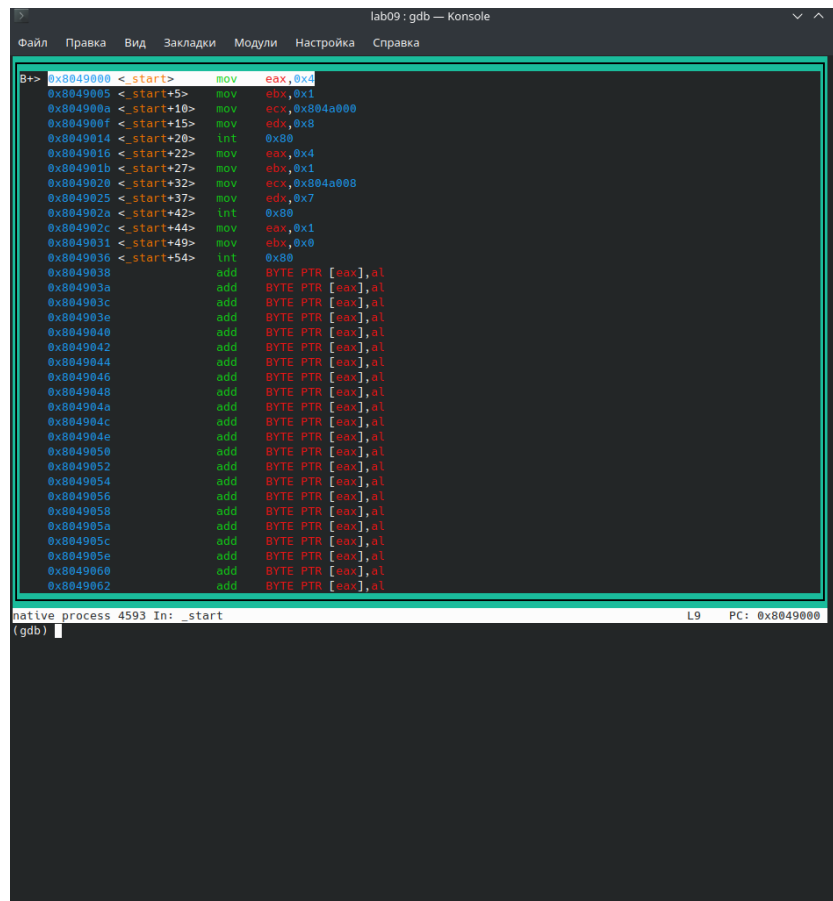


Рис. 16: Использование команды 'layout regs'

## Добавление точек останова

Проверяем с помощью команды 'info breakpoints' была ли установлена точка останова по имени на предыдущих шагах «\_start» (рис. [-@fig:017])

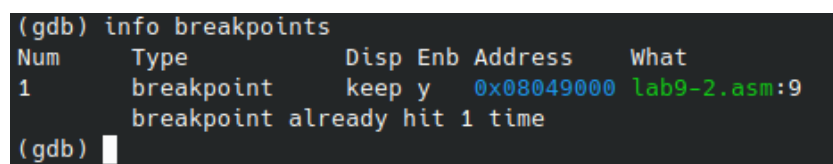


Рис. 17: Использование команды 'info breakpoints'

Определяем адрес предпоследней инструкции (mov ebx,0x0) и устанавливаем точку

останова (рис. [-@fig:018])

```
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 20.
(gdb) █
```

Рис. 18: Установка точки останова

Смотрим информацию о всех установленных точках останова (рис. [-@fig:019])

```
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000 lab9-2.asm:9
          breakpoint already hit 1 time
2        breakpoint     keep y   0x08049031 lab9-2.asm:20
(gdb) █
```

Рис. 19: Информация о всех точек останова

## Работа с данными программы в GDB

Выполняем 5 инструкций с помощью команды 'stepi' (рис. [-@fig:020]), (рис. [-@fig:021]), (рис. [-@fig:022]), (рис. [-@fig:023]), (рис. [-@fig:024])

```

lab09: gdb — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка

Register group: general-
eax      0x4      4      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffd010  0xffffd010  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049005  0x8049005 <_start+5>  eflags   0x202    [ IF ]
cs       0x23     35      ss       0x2b     43
ds       0x2b     43      es       0x2b     43
fs       0x0      0      gs       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
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
0x804903a      add     BYTE PTR [eax],al
0x804903c      add     BYTE PTR [eax],al
0x804903e      add     BYTE PTR [eax],al
0x8049040      add     BYTE PTR [eax],al

native process 4018 In: _start      L10  PC: 0x8049005
(gdb) layout regs
(gdb) info breakpoints
Num   Type             Disp Enb Address      What
1     breakpoint       keep y  0x8049000  lab9-2.asm:9
      breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 20.
(gdb) i b
Num   Type             Disp Enb Address      What
1     breakpoint       keep y  0x8049000  lab9-2.asm:9
      breakpoint already hit 1 time
2     breakpoint       keep y  0x8049031  lab9-2.asm:20
(gdb) stepi
(gdb)

```

Рис. 20: Команда 'stepi' 1

```

lab09: gdb — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка

--Register group: general--
eax      0x4      4      ecx      0x0      0
edx      0x0      0      ebx      0x1      1
esp      0xffffd010 0xffffd010  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x0804900a 0x0804900a <start+10>  eflags    0x202    [ IF ]
cs       0x23     35     ss       0x2b     43
ds       0x2b     43     es       0x2b     43
fs       0x0      0      gs       0x0      0

B+  0x08049000 <start>    mov    eax,0x4
0x08049005 <start+5>    mov    ebx,0x1
>  0x0804900a <start+10> mov    ecx,0x0804a000
0x0804900f <start+15>    mov    edx,0x0
0x08049014 <start+20>    int     0x80
0x08049016 <start+22>    mov    eax,0x4
0x0804901b <start+27>    mov    ebx,0x1
0x08049020 <start+32>    mov    ecx,0x0804a008
0x08049025 <start+37>    mov    edx,0x7
0x0804902a <start+42>    int     0x80
0x0804902c <start+44>    mov    eax,0x1
0x08049031 <start+49>    mov    ebx,0x0
b+  0x08049036 <start+54>    int     0x80
0x08049038      add    BYTE PTR [eax],al
0x0804903a      add    BYTE PTR [eax],al
0x0804903c      add    BYTE PTR [eax],al
0x0804903e      add    BYTE PTR [eax],al
0x08049040      add    BYTE PTR [eax],al

native process 4018 In: _start                               L11  PC: 0x0804900a
(gdb) layout regs
(gdb) info breakpoints
Num   Type             Disp Enb Address      What
1     breakpoint       keep y  0x08049000 lab9-2.asm:9
      breakpoint already hit 1 time
(gdb) break *0x08049031
breakpoint 2 at 0x08049031: file lab9-2.asm, line 20.
(gdb) i b
Num   Type             Disp Enb Address      What
1     breakpoint       keep y  0x08049000 lab9-2.asm:9
      breakpoint already hit 1 time
2     breakpoint       keep y  0x08049031 lab9-2.asm:20
(gdb) stepi
(gdb) stepi
(gdb)

```

Рис. 21: Команда 'stepi' 2



```

lab09: gdb — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка

--Register group: general--
eax      0x4      4      ecx      0x804a000    134520832
edx      0x0      0      ebx      0x1      1
esp      0xffffd010 0xffffd010  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x804900f 0x804900f <_start+15>  eflags   0x202    [ IF ]
cs       0x23     35     ss       0x2b     43
ds       0x2b     43     es       0x2b     43
fs       0x0      0      gs       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,0x0
0x8049014 <_start+20>   int     0x80
0x8049016 <_start+22>   mov     eax,0x4
0x804901b <_start+27>   mov     ebx,0x1
0x8049020 <_start+32>   mov     ecx,0x804a000
0x8049025 <_start+37>   mov     edx,0x7
0x804902a <_start+42>   int     0x60
0x804902c <_start+44>   mov     eax,0x1
b+ 0x8049031 <_start+49> mov     ebx,0x0
0x8049036 <_start+54>   int     0x80
0x8049038             add     BYTE PTR [eax],al
0x804903a             add     BYTE PTR [eax],al
0x804903c             add     BYTE PTR [eax],al
0x804903e             add     BYTE PTR [eax],al
0x8049040             add     BYTE PTR [eax],al

native process 4018 In: _start                               L12  PC: 0x804900f
(gdb) layout regs
(gdb) info breakpoints
Num   Type             Disp Enb Address      What
1     breakpoint       keep y  0x08049000 lab9-2.asm:9
      breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 20.
(gdb) i b
Num   Type             Disp Enb Address      What
1     breakpoint       keep y  0x08049000 lab9-2.asm:9
      breakpoint already hit 1 time
2     breakpoint       keep y  0x08049031 lab9-2.asm:20
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb)

```

Рис. 22: Команда 'stepi' 3

```

lab09: gdb — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка

--Register group: general--
eax      0x4          4          ecx      0x804a000      134520832
edx      0x8          8          ebx      0x1          1
esp      0xffffd010    0xffffd010    ebp      0x0          0x0
esi      0x0          0          edi      0x0          0
ebp      0x8049014      0x8049014 <_start+20>  eflags      0x202      [ IF ]
cs       0x23         35          ss       0x2b         43
ds       0x2b         43          es       0x2b         43
fs       0x0          0          gs       0x0          0

0+  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
0x8049031 <_start+49>     mov     ebx,0x0
b+  0x8049036 <_start+54>    int     0x80
0x8049038      add     BYTE PTR [eax],al
0x804903a      add     BYTE PTR [eax],al
0x804903c      add     BYTE PTR [eax],al
0x804903e      add     BYTE PTR [eax],al
0x8049040      add     BYTE PTR [eax],al

native process 4018 In: _start                               L13    PC: 0x8049014
(gdb) layout regs
(gdb) info breakpoints
Num   Type             Disp Enb Address      What
1     breakpoint       keep y   0x08049000  lab9-2.asm:9
      breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 20.
(gdb) i b
Num   Type             Disp Enb Address      What
1     breakpoint       keep y   0x08049000  lab9-2.asm:9
      breakpoint already hit 1 time
2     breakpoint       keep y   0x08049031  lab9-2.asm:20
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb)

```

Рис. 23: Команда 'stepi' 4

```

lab09: gdb — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка

--Register group: general--
eax      0x8      8      ecx      0x804a000      134520832
edx      0x3      3      ebx      0x1      1
esp      0xffffd010  0xffffd010  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049016  0x8049016  <_start+22>  eflags      0x202      [ IF ]
cs       0x23      35      ss       0x2b      43
ds       0x2b      43      es       0x2b      43
fs       0x0      0      gs       0x0      0

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,0x804a000
0x8049025 <_start+37>   mov     edx,0x7
0x804902a <_start+42>   int     0x80
0x804902c <_start+44>   mov     ecx,0x1
0x8049031 <_start+49>   mov     ebx,0x0
b+ 0x8049036 <_start+54>   int     0x80
0x8049038      add     BYTE PTR [eax],al
0x804903a      add     BYTE PTR [eax],al
0x804903c      add     BYTE PTR [eax],al
0x804903e      add     BYTE PTR [eax],al
0x8049040      add     BYTE PTR [eax],al

native process 4593 In: _start      L14      PC: 0x8049016
(gdb) layout regs
(gdb) info breakpoints
Num   Type             Disp Enb Address      What
1     breakpoint       keep y  0x8049000  lab9-2.asm:9
      breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 20.
(gdb) i b
Num   Type             Disp Enb Address      What
1     breakpoint       keep y  0x8049000  lab9-2.asm:9
      breakpoint already hit 1 time
2     breakpoint       keep y  0x8049031  lab9-2.asm:20
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb)

```

Рис. 24: Команда 'stepi' 5

Вопрос: значения каких регистров изменяются?

Ответ: во время выполнения команд менялись регистры ebx, ecx, edx, eip и eax.

Смотрим содержимое регистров с помощью команды 'info registers' (рис. [-@fig:025])

```

(gdb) info registers
eax      0x8      8
ecx      0x804a000  134520832
edx      0x3      3
ebx      0x1      1
esp      0xffffd010  0xffffd010
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016  0x8049016 <_start+22>
eflags   0x202      [ IF ]
cs       0x23      35
ss       0x2b      43
ds       0x2b      43
es       0x2b      43
fs       0x0      0
gs       0x0      0
(gdb)

```

Рис. 25: Смотрим содержимое регистров

Смотрим значение переменной `msg1` по имени (рис. [-@fig:026])

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

Рис. 26: Значение переменной `msg1` по имени

Смотрим значение переменной `msg2` по адресу (рис. [-@fig:027])

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

Рис. 27: Значение переменной `msg2` по адресу

Смотрим инструкцию «`mov ecx,msg2`», которая записывает в регистр `ecx` адрес переменной `msg2` (рис. [-@fig:028])

```
(gdb) x/1sb 0x8049020
0x8049020 <_start+32>:  "\271\b\240\004\b\272\a"
(gdb) █
```

Рис. 28: Инструкцию «`mov ecx,msg2`»

Изменяем первый символ переменной `msg1` с помощью команды ‘`set`’ (большую букву меняем на маленькую) (рис. [-@fig:029])

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

Рис. 29: Изменение первого символа переменной `msg1`

Заменяю первый символ во второй переменной msg2 (маленькую букву меняю на большую) (рис. [-@fig:030])

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

Рис. 30: Изменение первого символа переменной msg2

Смотрим значение регистра edx в различных форматах (сначала в двоичном, потом шестнадцатеричном, а затем в символьном) (рис. [-@fig:031])

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

Рис. 31: Значение регистра edx в различных форматах

С помощью команды 'set' изменяю значение регистра ebx (рис. [-@fig:032]), (рис. [-@fig:033])

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

Рис. 32: Изменение значения регистра ebx

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

Рис. 33: Изменение значение регистра ebx 2

Вопрос: в чём разница вывода команд 'p/s \$ebx'?

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

Завершаем выполнение программы с помощью команды 'continue' (рис. [-@fig:034])

```
(gdb) continue
Continuing.
World!

Breakpoint 2, _start () at lab9-2.asm:20
(gdb) █
```

Рис. 34: Завершение программы

Выходим из GDB с помощью команды 'quit' (рис. [-@fig:035]), (рис. [-@fig:036])

```
(gdb) continue
Continuing.
World!

Breakpoint 2, _start () at lab9-2.asm:20
(gdb) quit█
```

Рис. 35: Выход 1

```
(gdb) quit
A debugging session is active.

        Inferior 1 [process 4593] will be killed.

Quit anyway? (y or n) y
```

Рис. 36: Выход 2

## Обработка аргументов командной строки в GDB

Копируем файл «lab8-2.asm», созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки в файл с именем «lab9-3.asm» (рис. [-@fig:037])

```
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab9-3.asm
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ ls
in_out.asm lab9-1 lab9-1.asm lab9-1.o lab9-2 lab9-2.asm lab9-2.lst lab9-2.o lab9-3.asm
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$
```

Рис. 37: Создание файла «lab9-3.asm»

Создаём исполняемый файл (рис. [-@fig:038])

```
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-3 lab9-3.o
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ ls
in_out.asm lab9-1.asm lab9-2 lab9-2.lst lab9-3 lab9-3.lst
lab9-1 lab9-1.o lab9-2.asm lab9-2.o lab9-3.asm lab9-3.o
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$
```

Рис. 38: Создание исполняемого файла

Загружаем программу с аргументами в GDB. Для этого используем ключ ‘-args’ (рис. [-@fig:039])

```
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ gdb --args lab9-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 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 lab9-3...
(gdb) █
```

Рис. 39: Загрузка в gdb с использованием ‘--args’

Устанавливаем точку останова перед первой инструкцией в программе и запускаем её (рис. [-@fig:040])

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 5.
(gdb) run
Starting program: /home/eavernikovskaya/work/arch-pc/lab09/lab9-3 аргумент1 аргумент 2 аргумент\ 3

Breakpoint 1, _start () at lab9-3.asm:5
5      pop ecx
(gdb) █
```

Рис. 40: Точка останова + запуск программы

Изучаем адрес вершины стека (рис. [-@fig:041])

```
(gdb) x/x $esp
0xffffcfc0: 0x00000005
(gdb) █
```

Рис. 41: Адрес вершины стека

Смотрим остальные позиции стека (рис. [-@fig:042])



```

(gdb) x/s *(void**)(esp + 4)
0xffffd1ab:    "/home/eavernikovskaya/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd1db:    "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd1ed:    "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd1fe:    "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd200:    "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0:    <error: Cannot access memory at address 0x0>
(gdb) █

```

Рис. 42: Остальные позиции стека

Вопрос: почему шаг изменения адреса равен 4?

Ответ: шаг изменения адреса равен 4, потому что адресные регистры увеличиваются на 4 при выполнении инструкций в режиме 32-битных (4 байта) процессоров x86. Поэтому при выполнении каждой инструкции адрес следующей инструкции увеличивается на 4. А, например, в случае использования режима 64-битных (8 байтаов) процессоров x86 адреса будут увеличиваться на 8.

## Задание для самостоятельной работы

Создаём файл «lab9-4.asm» (рис. [-@fig:043])

```

eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab08$ cd ~/work/arch-pc/lab09
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ touch lab9-4.asm

```

Рис. 43: Создание файла «lab9-4.asm»

Переписываем программу из лабораторной работы №8, реализовав вычисления функции  $f(x)$  как подпрограмму (рис. [-@fig:044])

Текст изменённой программы:

```

#include 'in_out.asm'
SECTION .data
msg1: DB 'Функция: f(x)=10(x-1)',0

```

```
msg2 db 'Результат: ',0
```

```
SECTION .bss
```

```
fx: RESB 80
```

```
SECTION .text
```

```
global _start
```

```
_start:
```

```
    mov eax,msg1
```

```
    call sprintLF
```

```
    pop ecx
```

```
    pop edx
```

```
    sub ecx,1
```

```
    mov esi,10
```

```
next:
```

```
    cmp ecx,0h
```

```
    jz _end
```

```
    pop eax
```

```
    call atoi
```

```
    call _calcul_f
```

```
    add [fx],eax
```

```
    loop next
```

```
_end:
```

```
    mov eax,msg2
```

```
    call sprint
```

```
    mov eax,[fx]
```

```
    call iprintLF
```

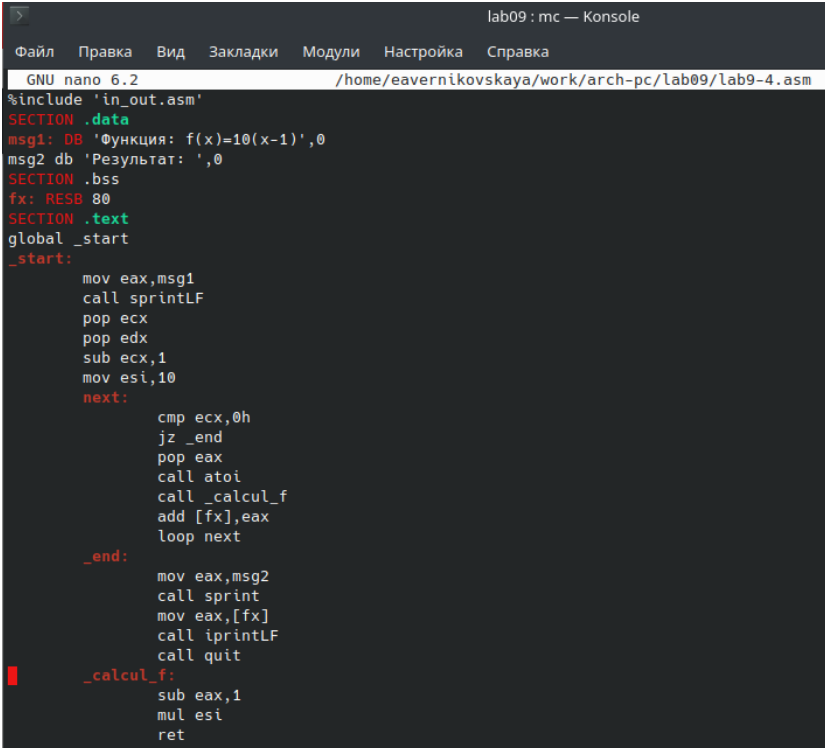
```
    call quit
```

```
_calcul_f:
```

```
    sub eax,1
```

```
    mul esi
```

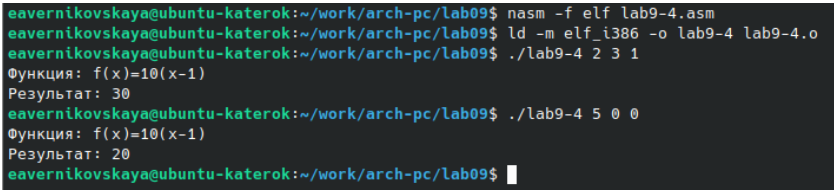
ret



```
lab09: mc — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
GNU nano 6.2 /home/eavernikovskaya/work/arch-pc/lab09/lab9-4.asm
#include 'in_out.asm'
SECTION .data
msg1: DB 'Функция: f(x)=10(x-1)',0
msg2 db 'Результат: ',0
SECTION .bss
fx: RESB 80
SECTION .text
global _start
_start:
    mov eax,msg1
    call sprintLF
    pop ecx
    pop edx
    sub ecx,1
    mov esi,10
next:
    cmp ecx,0h
    jz _end
    pop eax
    call atoi
    call _calcul_f
    add [fx],eax
    loop next
_end:
    mov eax,msg2
    call sprint
    mov eax,[fx]
    call iprintLF
    call quit
_calcul_f:
    sub eax,1
    mul esi
    ret
```

Рис. 44: Написание программы

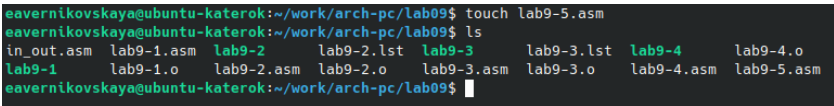
Создаём исполняемый файл и запускаем его (рис. [-@fig:045])



```
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ nasm -f elf lab9-4.asm
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-4 lab9-4.o
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ ./lab9-4 2 3 1
Функция: f(x)=10(x-1)
Результат: 30
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ ./lab9-4 5 0 0
Функция: f(x)=10(x-1)
Результат: 20
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$
```

Рис. 45: Создание исполняемого файла и его запуск

Создаём файл «lab9-5.asm» (рис. [-@fig:046])



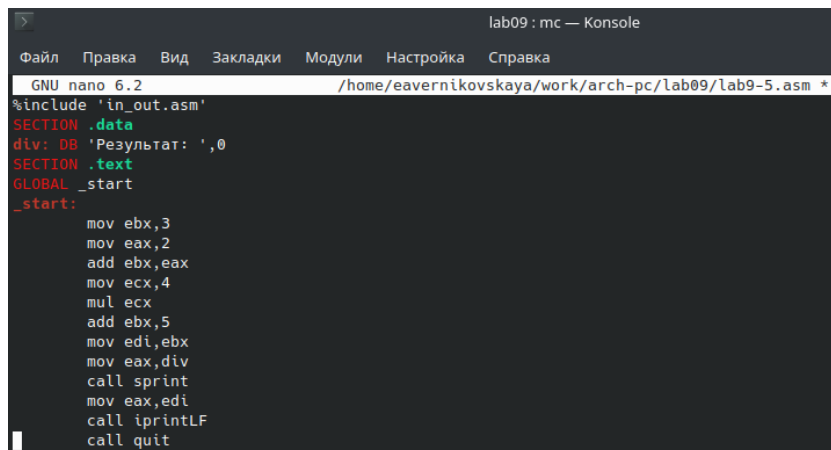
```
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ touch lab9-5.asm
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ ls
in_out.asm  lab9-1.asm  lab9-2      lab9-2.lst  lab9-3      lab9-3.lst  lab9-4      lab9-4.o
lab9-1      lab9-1.o    lab9-2.asm  lab9-2.o    lab9-3.asm  lab9-3.o    lab9-4.asm  lab9-5.asm
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$
```

Рис. 46: Создание файла «lab9-5.asm»

Вводим текст программы, которая вычисляет выражение  $(3+2)*4+5$  (рис. [-@fig:047])

Текст программы:

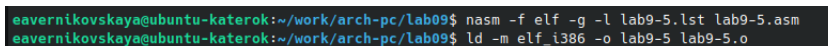
```
%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
    mov ebx,3
    mov eax,2
    add ebx,eax
    mov ecx,4
    mul ecx
    add ebx,5
    mov edi,ebx
    mov eax,div
    call sprint
    mov eax,edi
    call iprintLF
    call quit
```



```
GNU nano 6.2 /home/eavernikovskaya/work/arch-pc/lab09/lab9-5.asm *
%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
    mov ebx,3
    mov eax,2
    add ebx,eax
    mov ecx,4
    mul ecx
    add ebx,5
    mov edi,ebx
    mov eax,div
    call sprint
    mov eax,edi
    call iprintLF
    call quit
```

Рис. 47: Ввод текста программы

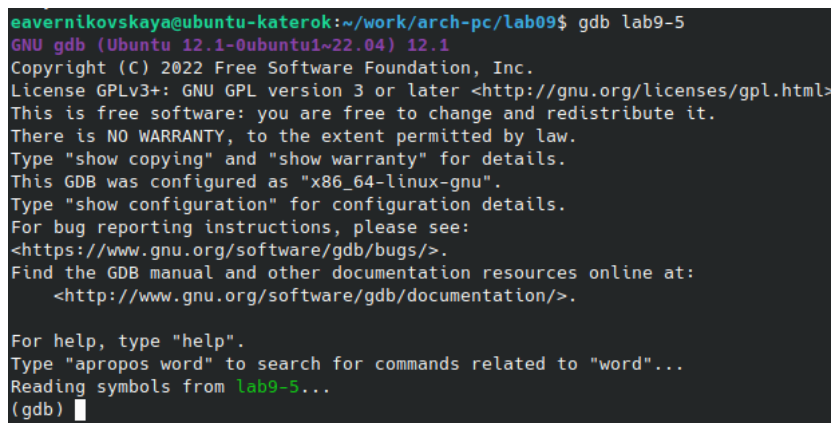
Создаём исполняемый файл (рис. [-@fig:048])



```
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-5.lst lab9-5.asm
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-5 lab9-5.o
```

Рис. 48: Создание исполняемого файла

Загружаем исполняемый файл в отладчик gdb (рис. [-@fig:049])



```
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ gdb lab9-5
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 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 lab9-5...
(gdb)
```

Рис. 49: Загрузка исполняемого файла в gdb

Проверяем работу программы, запустив её в оболочке GDB с помощью команды ‘run’ (рис. [-@fig:050])

```
(gdb) run
Starting program: /home/eavernikovskaya/work/arch-pc/lab09/lab9-5
Результат: 10
[Inferior 1 (process 5243) exited normally]
(gdb) █
```

Рис. 50: Проверка работы программы в оболочке gdb

При запуске программа даёт неверный результат.

Запускаем исполняемый файл в отладчике GDB и смотрим на изменения регистров с помощью команды ‘stepi’ (рис. [-@fig:051])

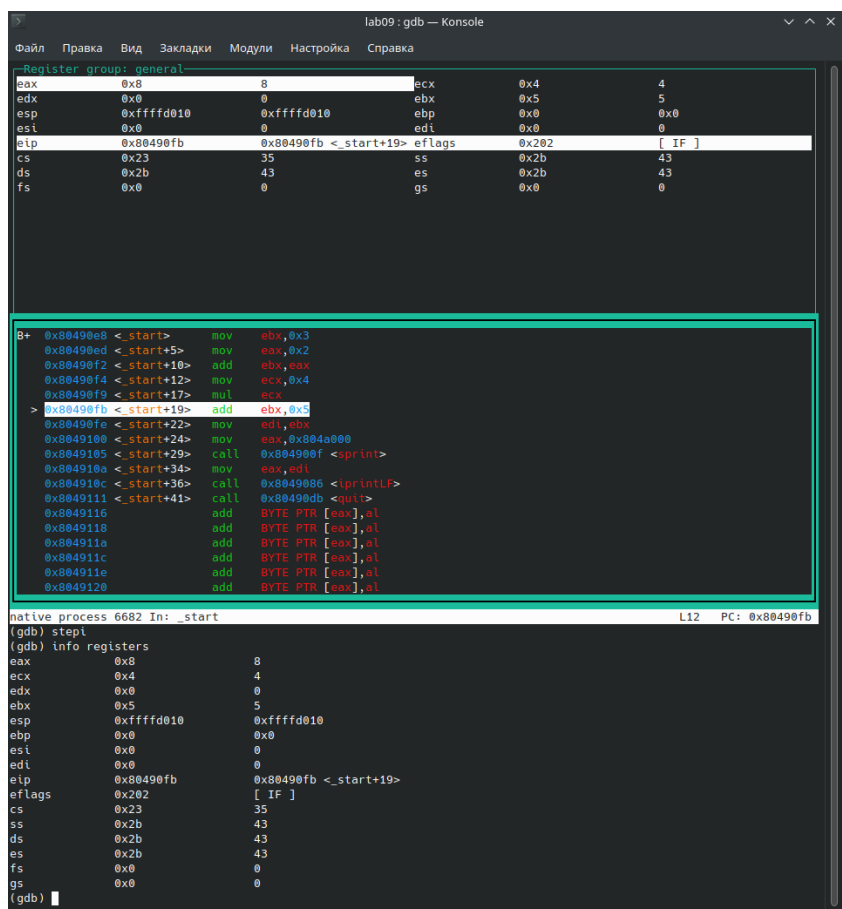


Рис. 51: Ищем ошибку регистров

Обнаружив ошибку, исправляем программу (рис. [-@fig:052])

Текст исправленной программы:

```

#include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
    mov ebx,3
    mov eax,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

```

```

lab09 : mc — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
GNU nano 6.2 /home/eavernikovskaya/work/arch-pc/lab09/lab9-5.asm
#include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
    mov ebx,3
    mov eax,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. 52: Исправленная программа

Проверяем работу исправленной программы (рис. [-@fig:053])

```
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ nasm -f elf lab9-5.asm
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-5 lab9-5.o
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ ./lab9-5
Результат: 25
eavernikovskaya@ubuntu-katerok:~/work/arch-pc/lab09$ █
```

Рис. 53: Проверка программы



## Выводы

В ходе выполнения лабораторной работы мы приобрели навыки написания программ с использованием подпрограмм. Также мы познакомились с методами отладки при помощи GDB и его основными возможностями.