

# **Отчёт по лабораторной работе 9**

**дисциплина: Архитектура компьютера**

**Кылыш Гоктюрк**

# **Содержание**

<b>1 Цель работы</b>	<b>5</b>
<b>2 Выполнение лабораторной работы</b>	<b>6</b>
<b>3 Выводы</b>	<b>27</b>

# Список иллюстраций

2.1 Программа lab9-1.asm . . . . .	7
2.2 Запуск программы lab9-1.asm . . . . .	7
2.3 Программа lab9-1.asm . . . . .	8
2.4 Запуск программы lab9-1.asm . . . . .	9
2.5 Программа lab9-2.asm . . . . .	10
2.6 Запуск программы lab9-2.asm в отладчике . . . . .	11
2.7 Дизассемблированный код . . . . .	12
2.8 Дизассемблированный код в режиме интел . . . . .	13
2.9 Точка остановки . . . . .	14
2.10 Изменение регистров . . . . .	15
2.11 Изменение регистров . . . . .	16
2.12 Изменение значения переменной . . . . .	17
2.13 Вывод значения регистра . . . . .	18
2.14 Вывод значения регистра . . . . .	19
2.15 Вывод значения регистра . . . . .	20
2.16 Программа lab9-4.asm . . . . .	21
2.17 Запуск программы lab9-4.asm . . . . .	22
2.18 Код с ошибкой . . . . .	23
2.19 Отладка . . . . .	24
2.20 Код исправлен . . . . .	25
2.21 Проверка работы . . . . .	26

# **Список таблиц**

# **1 Цель работы**

Целью работы является приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

## **2 Выполнение лабораторной работы**

1. Создал каталог для выполнения лабораторной работы № 9, перешел в него и создал файл lab9-1.asm.
2. В качестве примера рассмотрим программу вычисления арифметического выражения  $f(x) = 2x + 7$  с помощью подпрограммы calcul. В данном примере  $x$  вводится с клавиатуры, а само выражение вычисляется в подпрограмме.

The screenshot shows a terminal window titled "GNU nano 7.2" with the file name "lab9-1.asm". The code is written in NASM assembly language. It includes sections for data and bss, defines variables msg and result, and uses global \_start. The program prompts for input, reads it into eax, calculates 2\*x+7, and prints the result. A comment indicates a call to a subprogram \_calcul. The assembly code is as follows:

```
GNU nano 7.2          lab9-1.asm
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
rez: 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,[rez]
call iprintLF
call quit
_calcul:
mov ebx,2
mul ebx
add eax,7
mov [rez],eax
ret ; выход из подпрограммы
```

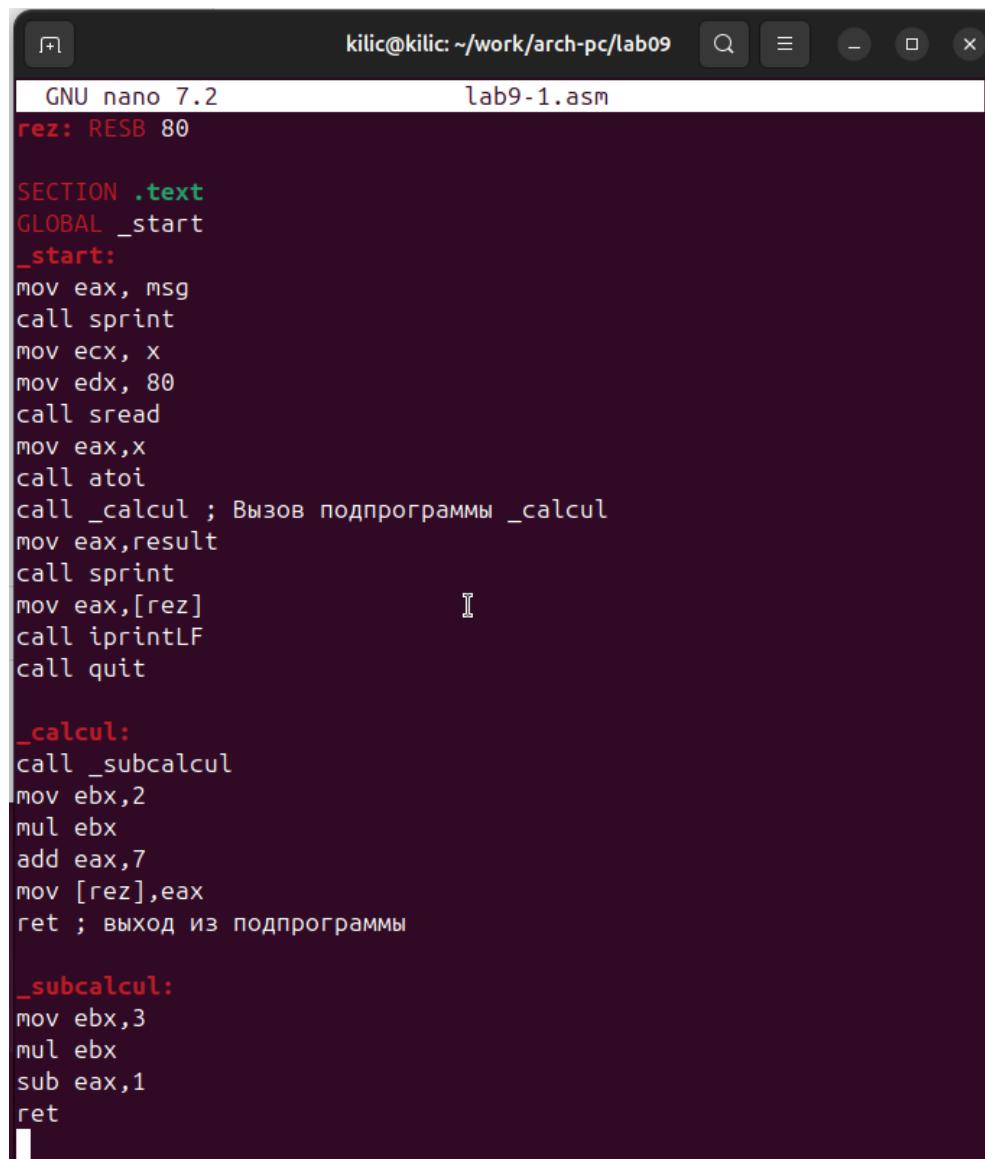
Рисунок 2.1: Программа lab9-1.asm

The screenshot shows a terminal window with the following session:

```
kilic@kilic:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
kilic@kilic:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
kilic@kilic:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 4
2x+7=15
kilic@kilic:~/work/arch-pc/lab09$
```

Рисунок 2.2: Запуск программы lab9-1.asm

3. Изменил текст программы, добавив подпрограмму subcalcul в подпрограмму calcul, для вычисления выражения  $f(g(x))$ , где  $x$  вводится с клавиатуры,  $f(x) = 2x + 7$ ,  $g(x) = 3x - 1$ .



The screenshot shows a terminal window titled "lab9-1.asm" running on a Linux system. The window title bar includes the text "kilic@kilic: ~/work/arch-pc/lab09". The terminal content displays assembly code for a program named "lab9-1.asm". The code defines a global variable "rez" at address 80, and a section ".text" containing the main program logic. It reads a value from standard input, converts it to an integer, calls a subprogram "calcul" (which contains a call to "subcalcul"), prints the result, and then exits. The "calcul" and "subcalcul" subprograms are defined with their own logic for performing the calculations.

```
GNU nano 7.2          lab9-1.asm
rez: 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,[rez]           █
    call iprintLF
    call quit

_calcul:
    call _subcalcul
    mov ebx,2
    mul ebx
    add eax,7
    mov [rez],eax
    ret ; выход из подпрограммы

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

Рисунок 2.3: Программа lab9-1.asm

```
kilic@kilic:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
kilic@kilic:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
kilic@kilic:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 4
2(3x-1)+7=29
kilic@kilic:~/work/arch-pc/lab09$
```

Рисунок 2.4: Запуск программы lab9-1.asm

4. Создал файл lab9-2.asm с текстом программы из Листинга 9.2. (Программа печати сообщения Hello world!).



The screenshot shows a terminal window with the title "kilic@kilic: ~/work/arch-pc/lab09". The file being edited is "lab9-2.asm". The assembly code is as follows:

```
GNU nano 7.2                      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
```

Рисунок 2.5: Программа lab9-2.asm

Получил исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом „-g“.

Загрузил исполняемый файл в отладчик gdb. Проверил работу программы, запустив ее в оболочке GDB с помощью команды run (сокращенно r).

```
kilic@kilic:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
kilic@kilic:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o

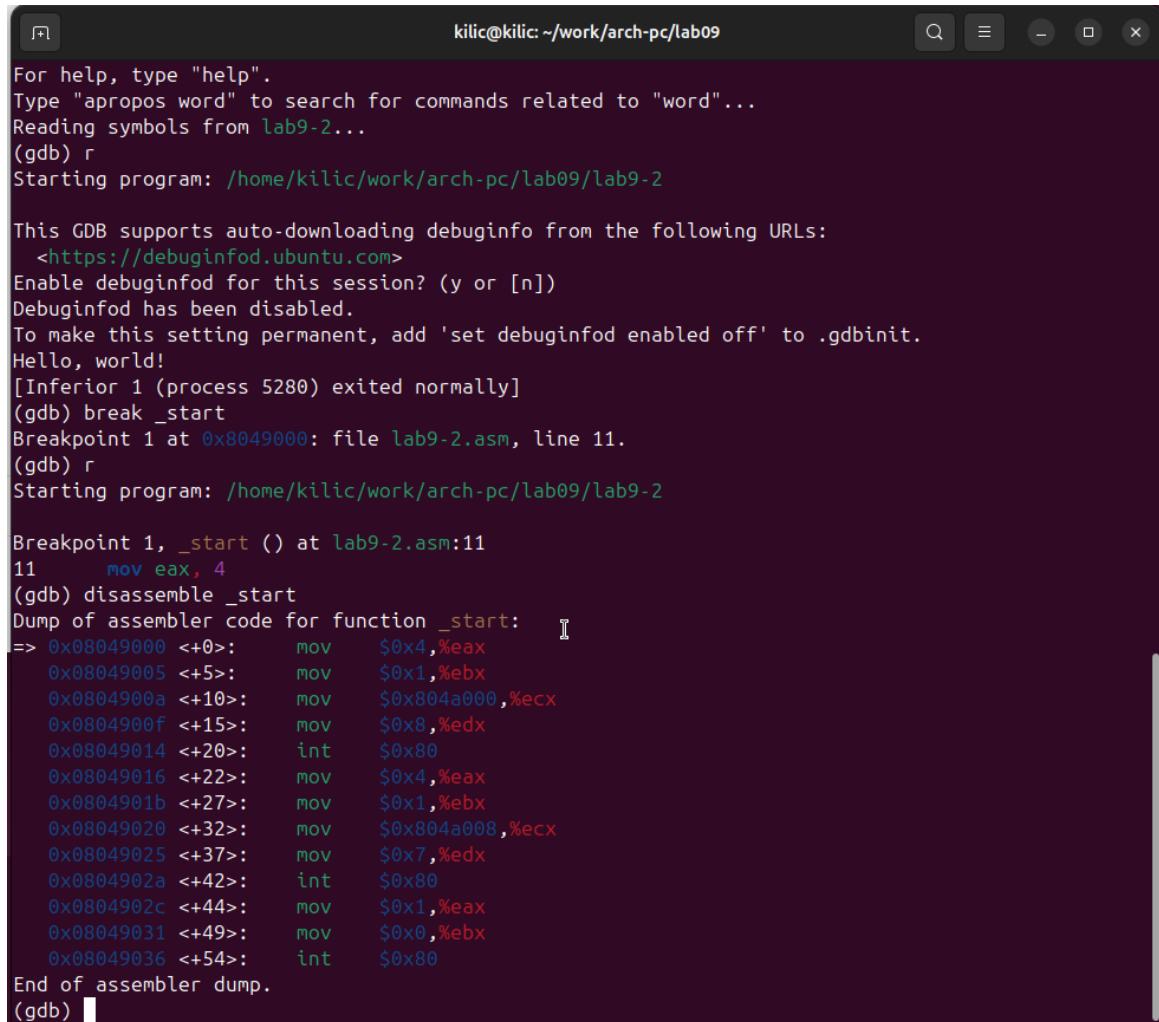
kilic@kilic:~/work/arch-pc/lab09$ gdb lab9-2
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
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) r
Starting program: /home/kilic/work/arch-pc/lab09/lab9-2

This GDB supports auto-downloading debuginfo from the following URL
s:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n])
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to
.gdbinit.
Hello, world!
[Inferior 1 (process 5280) exited normally]
(gdb) █
```

Рисунок 2.6: Запуск программы lab9-2.asm в отладчике

Для более подробного анализа программы установите брейкпоинт на метку start, с которой начинается выполнение любой ассемблерной программы, и запустите её. Посмотрите дизассемблированный код программы.



For help, type "help".  
Type "apropos word" to search for commands related to "word"..."  
Reading symbols from lab9-2...  
(gdb) r  
Starting program: /home/kilic/work/arch-pc/lab09/lab9-2  
  
This GDB supports auto-downloading debuginfo from the following URLs:  
<<https://debuginfod.ubuntu.com>>  
Enable debuginfod for this session? (y or [n])  
Debuginfod has been disabled.  
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.  
Hello, world!  
[Inferior 1 (process 5280) exited normally]  
(gdb) break \_start  
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 11.  
(gdb) r  
Starting program: /home/kilic/work/arch-pc/lab09/lab9-2  
  
Breakpoint 1, \_start () at lab9-2.asm:11  
11 mov eax, 4  
(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) █

Рисунок 2.7: Дизассемблированный код

```
kilic@kilic: ~/work/arch-pc/lab09

Breakpoint 1, _start () at lab9-2.asm:11
11      mov eax, 4
(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) 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)
```

Рисунок 2.8: Дизассемблированный код в режиме интел

На предыдущих шагах была установлена точка остановки по имени метки (`_start`). Проверил это с помощью команды `info breakpoints` (кратко `i b`). Установил еще одну точку остановки по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции. Определил адрес предпоследней инструкции (`mov ebx,0x0`) и установил точку.

The screenshot shows a terminal window with the following content:

```
Register group: general
eax          0x0          0
ecx          0x0          0
edx          0x0          0
ebx          0x0          0
esp 0xfffffcf10 0xfffffcf10
ebp 0x0          0x0
esi          0x0          0
edi          0x0          0
eip 0x8049000 0x8049000 <_start>
eflags       0x202      [ IF ]
```

```
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
```

```
native process 5289 (asm) In: _start
(gdb) layout regs
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 22.
(gdb) i b
Num      Type        Disp Enb Address     What
1        breakpoint  keep y  0x08049000 lab9-2.asm:11
                  breakpoint already hit 1 time
2        breakpoint  keep y  0x08049031 lab9-2.asm:22
(gdb)
```

Рисунок 2.9: Точка остановки

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Выполнил 5 инструкций с помощью команды stepi (или si) и проследил за изменением значений регистров.

```

Register group: general
eax          0x4          4
ecx          0x0          0
edx          0x0          0
ebx          0x0          0
esp          0xfffffcf10    0xfffffcf10
ebp          0x0          0x0
esi          0x0          0
edi          0x0          0
eip          0x8049005     0x8049005 <_start+5>
eflags        0x202        [ IF ]

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

native process 5289 (asm) In: _start
edi          0x0          0
eip          0x8049000     0x8049000 <_start>
eflags        0x202        [ IF ]
cs           0x23         35
--Type <RET> for more, q to quit, c to continue without paging--
ss           0x2b         43
ds           0x2b         43
es           0x2b         43
fs           0x0          0
gs           0x0          0
(gdb) si
(gdb) █

```

Рисунок 2.10: Изменение регистров

The screenshot shows the GDB debugger interface. At the top, it says "kilic@kilic: ~/work/arch-pc/lab09". Below that is a "Register group: general" table:

eax	0x8	8
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x1	1
esp	0xfffffcf10	0xfffffcf10
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x8049016	0x8049016 <_start+22>
eflags	0x202	[ IF ]

Below the registers is the assembly code:

```
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
```

At the bottom, it says "native process 5289 (asm) In: \_start" and "L16 PC: 0x8049016". It also says "--Type <RET> for more, q to quit, c to continue without paging--". The stack information shows:

ss	0x2b	43
ds	0x2b	43
es	0x2b	43
fs	0x0	0
gs	0x0	0

(gdb) si  
(gdb) si  
(gdb) si  
(gdb) si  
(gdb) si  
(gdb) [REDACTED]

Рисунок 2.11: Изменение регистров

Посмотрел значение переменной msg1 по имени. Посмотрел значение переменной msg2 по адресу.

Изменить значение для регистра или ячейки памяти можно с помощью команды set, задав ей в качестве аргумента имя регистра или адрес. Изменил первый символ переменной msg1.

The screenshot shows the GDB debugger interface. At the top, it displays the command `kilic@kili: ~/work/arch-pc/lab09`. Below this is a register dump titled "Register group: general". The registers listed are eax, ecx, edx, ebx, esp, ebp, esi, edi, eip, and eflags. The values for eax, ecx, edx, ebx, esp, ebp, esi, edi, and eip are shown in hexadecimal format. The value for eip is highlighted in yellow and points to the instruction at address 0x8049016. The eflags register shows the value [ IF ].

Below the register dump is the assembly code for the program. The code starts at address 0x8049000 and includes instructions like mov eax, 0x4, mov ebx, 0x1, mov ecx, 0x804a000, etc. The instruction at address 0x8049016 is highlighted in yellow and is labeled <\_start+22>. The assembly code continues with more mov and int instructions.

At the bottom of the screen, there is a terminal window showing GDB commands. It starts with "native process 5289 (asm) In: \_start". Then it shows several GDB commands: si, x/1sb &msg1, set {char}&msg1='h', set {char}0x804a008='L', and x/1sb 0x804a008. The PC is shown as 0x8049016. The terminal window has a light blue background.

Рисунок 2.12: Изменение значения переменной

Вывел в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx.

The screenshot shows a GDB session with the following details:

- Registers:** A table showing general registers:

Register	Value	Description
eax	0x8	8
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x1	1
esp	0xfffffcf10	0xfffffcf10
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x8049016	0x8049016 <_start+22>
eflags	0x202	[ IF ]
- Assembly:** The assembly code for the program, showing instructions from the start of memory up to address 0x8049016:

```
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
```
- Registers View:** A terminal window showing the state of registers:

```
native process 5289 (asm) In: _start
$2 = 1000
(gdb) p/s $ecx
$3 = 134520832
(gdb) p/x $ecx
$4 = 0x804a000
(gdb) p/s $edx
$5 = 8
(gdb) p/t $edx
$6 = 1000
(gdb) p/x $edx
$7 = 0x8
(gdb)
```
- Information:** The top right corner shows "L16 PC: 0x8049016".

Рисунок 2.13: Вывод значения регистра

С помощью команды set изменил значение регистра ebx

The screenshot shows the GDB debugger interface. At the top, it says "kilic@kilic: ~/work/arch-pc/lab09". Below that is a table of general registers:

Register	Value	Description
eax	0x8	8
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x2	2
esp	0xfffffcf10	0xfffffcf10
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x8049016	0x8049016 <_start+22>
eflags	0x202	[ IF ]

Below the registers is the assembly code for the program:

```

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

```

At the bottom, there is a command history and status information:

```

native process 5289 (asm) In: _start
$5 = 8
(gdb) p/t $edx
$6 = 1000
(gdb) p/x $edx
$7 = 0x8
(gdb) set $ebx='2'
(gdb) p/s $ebx
$8 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$9 = 2
(gdb)

```

L16 PC: 0x8049016

Рисунок 2.14: Вывод значения регистра

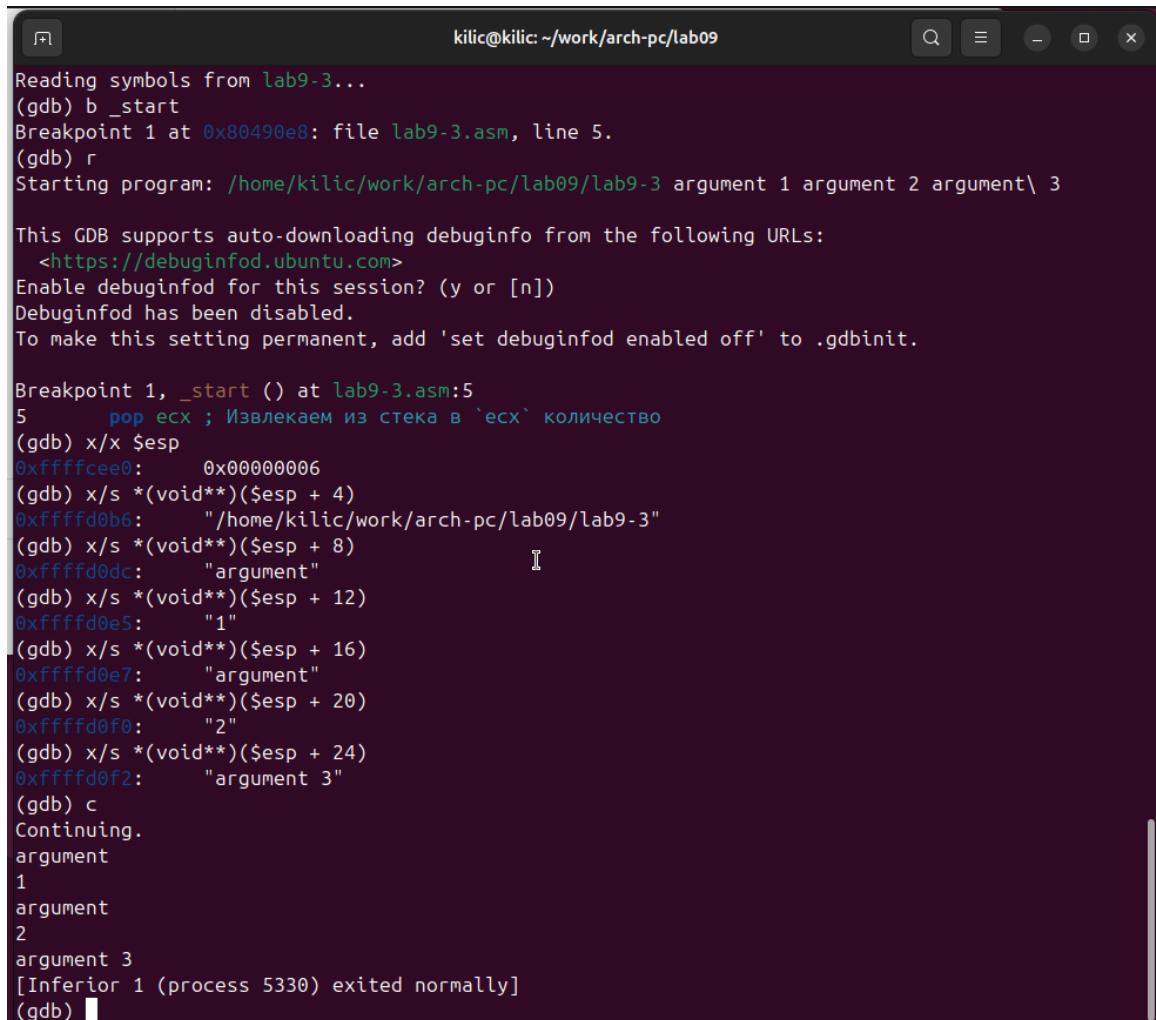
5. Скопировал файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки. Создал исполняемый файл. Для загрузки в gdb программы с аргументами необходимо использовать ключ –args. Загрузил исполняемый файл в отладчик, указав аргументы.

Для начала установил точку останова перед первой инструкцией в программе и запустил ее.

Адрес вершины стека храниться в регистре esp и по этому адресу распола-

гается число равное количеству аргументов командной строки (включая имя программы). Как видно, число аргументов равно 5 – это имя программы lab9-3 и непосредственно аргументы: аргумент1, аргумент, 2 и „аргумент 3“.

Посмотрел остальные позиции стека – по адресу [esp+4] располагается адрес в памяти где находится имя программы, по адресу [esp+8] храниться адрес первого аргумента, по адресу [esp+12] – второго и т.д.



The screenshot shows a terminal window titled "kilic@kilic: ~/work/arch-pc/lab09". The GDB debugger is running on the program "lab9-3". The assembly code at address 0x80490e8 is shown:

```
Reading symbols from lab9-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 5.
(gdb) r
Starting program: /home/kilic/work/arch-pc/lab09/lab9-3 argument 1 argument 2 argument\ 3

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n])
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.

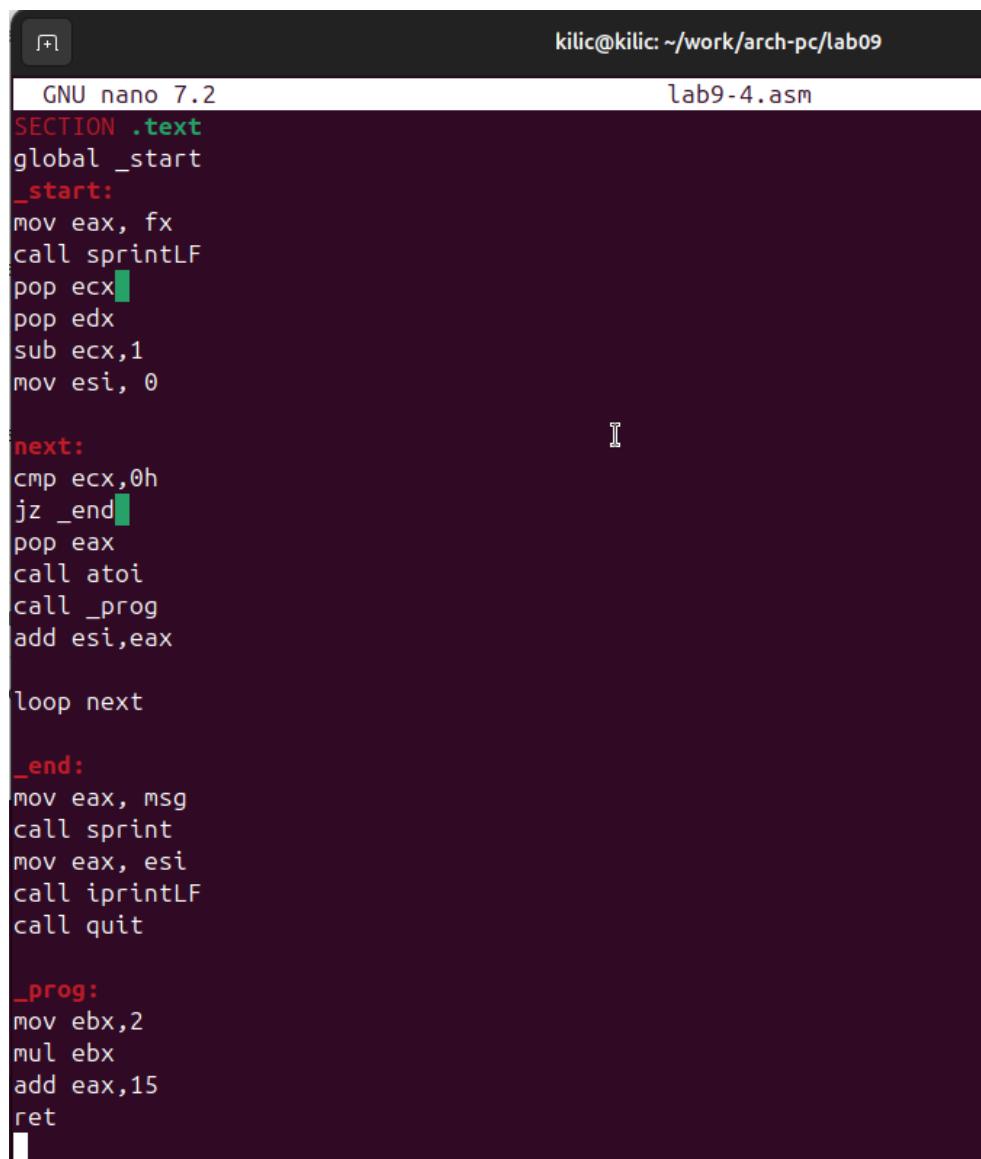
Breakpoint 1, _start () at lab9-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xfffffcee0:    0x00000006
(gdb) x/s *(void**)( $esp + 4)
0xfffffd0b6:    "/home/kilic/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)( $esp + 8)
0xfffffd0dc:    "argument"           I
(gdb) x/s *(void**)( $esp + 12)
0xfffffd0e5:    "1"
(gdb) x/s *(void**)( $esp + 16)
0xfffffd0e7:    "argument"
(gdb) x/s *(void**)( $esp + 20)
0xfffffd0f0:    "2"
(gdb) x/s *(void**)( $esp + 24)
0xfffffd0f2:    "argument 3"
(gdb) c
Continuing.

argument
1
argument
2
argument 3
[Inferior 1 (process 5330) exited normally]
(gdb)
```

Рисунок 2.15: Вывод значения регистра

Объясню, почему шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12] - шаг равен размеру переменной - 4 байтам.

6. Преобразовал программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции  $f(x)$  как подпрограмму.



The screenshot shows a terminal window titled "GNU nano 7.2" with the command "kilic@kilic: ~/work/arch-pc/lab09" at the top. The file being edited is "lab9-4.asm". The assembly code is as follows:

```
SECTION .text
global _start
_start:
    mov eax, fx
    call sprintLF
    pop ecx
    pop edx
    sub ecx,1
    mov esi, 0

    next:
    cmp ecx,0h
    jz _end
    pop eax
    call atoi
    call _prog
    add esi,eax

    loop next

_end:
    mov eax, msg
    call sprint
    mov eax, esi
    call iprintLF
    call quit

_prog:
    mov ebx,2
    mul ebx
    add eax,15
    ret
```

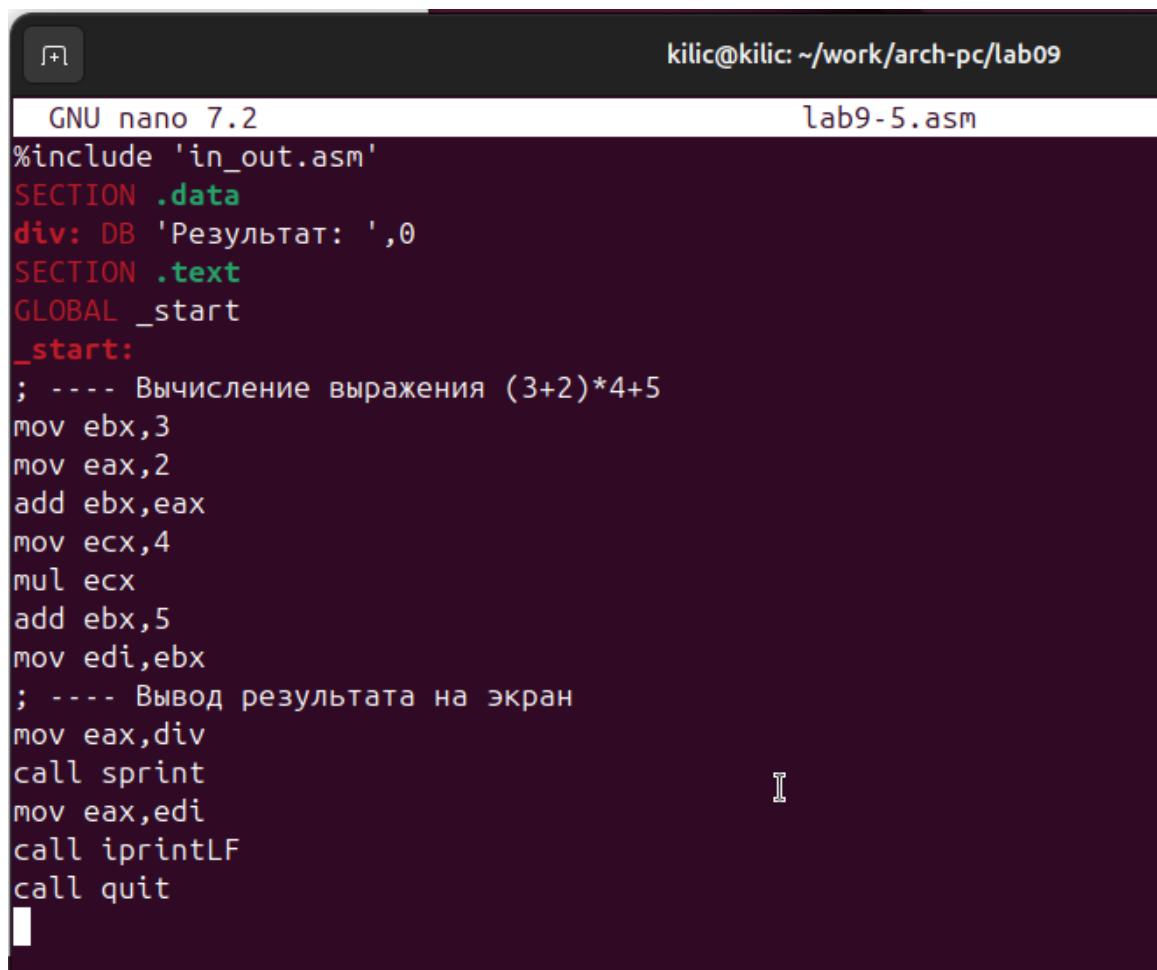
Рисунок 2.16: Программа lab9-4.asm

```
kilic@kilic:~/work/arch-pc/lab09$ nano lab9-4.asm
kilic@kilic:~/work/arch-pc/lab09$ nasm -f elf lab9-4.asm
kilic@kilic:~/work/arch-pc/lab09$ ld -m elf_i386 lab9-4.o -o lab9-4
kilic@kilic:~/work/arch-pc/lab09$ ./lab9-4 1
f(x)= 2x+15
Результат: 17
kilic@kilic:~/work/arch-pc/lab09$ ./lab9-4 3 6 7 9
f(x)= 2x+15
Результат: 110
kilic@kilic:~/work/arch-pc/lab09$
```

Рисунок 2.17: Запуск программы lab9-4.asm

7. В листинге приведена программа вычисления выражения  $(3 + 2) * 4 + 5$ .

При запуске данная программа дает неверный результат. Проверил это. С помощью отладчика GDB, анализируя изменения значений регистров, определю ошибку и исправлю ее.



The screenshot shows a terminal window with the title "klic@klic: ~/work/arch-pc/lab09". The file being edited is "lab9-5.asm". The code is written in assembly language:

```
GNU nano 7.2
lab9-5.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 ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```

Рисунок 2.18: Код с ошибкой

The screenshot shows a terminal window with the following content:

```
Register group: general
eax          0x8          8
ecx          0x4          4
edx          0x0          0
ebx          0xa          10
esp          0xfffffcf10    0xfffffcf10
ebp          0x0          0x0
esi          0x0          0
edi          0xa          10
eip          0x8049100     0x8049100 <_start+24>
eflags        0x206      [ PF IF ]

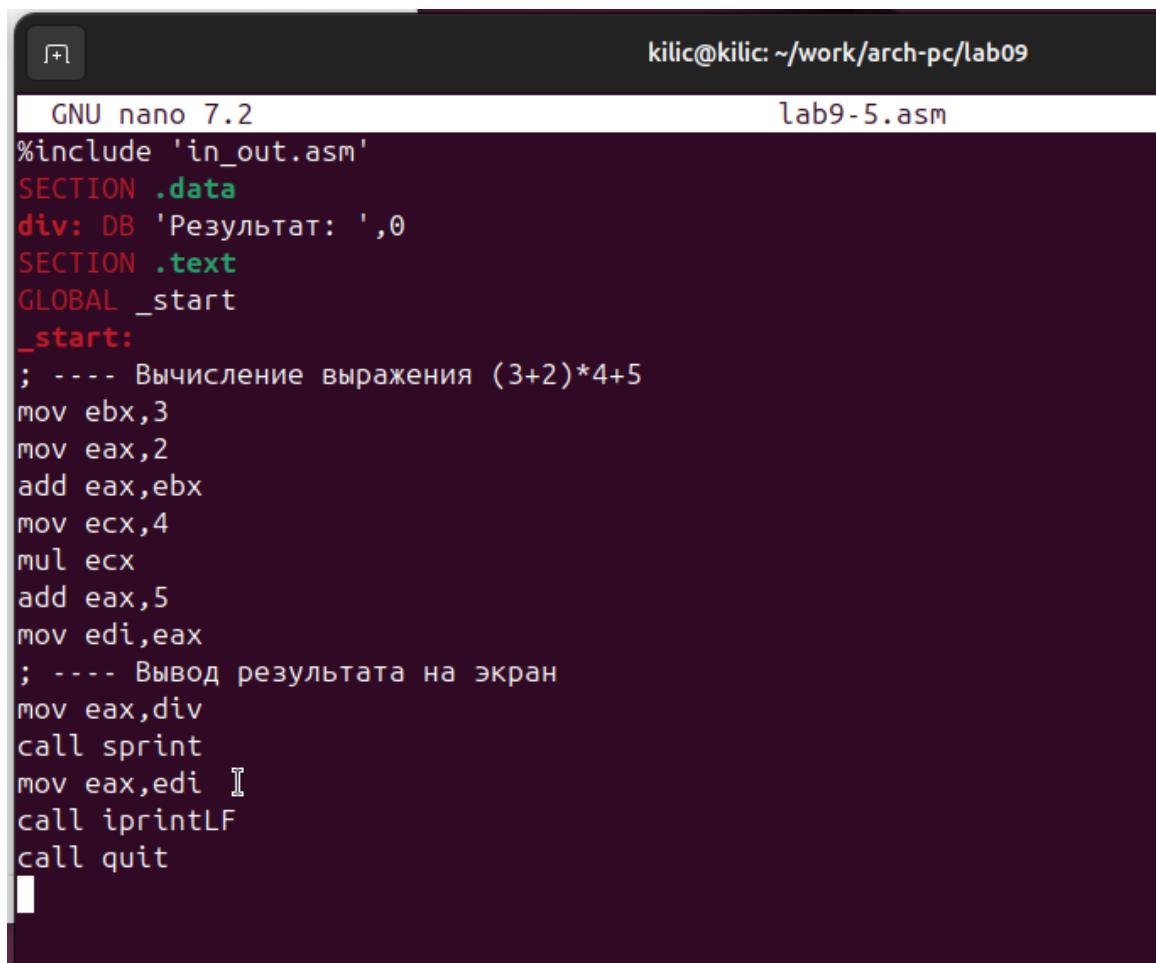
B+ 0x80490e8 <_start>    mov    ebx,0x3
0x80490ed <_start+5>    mov    eax,0x2
0x80490f2 <_start+10>   add    ebx,eax
0x80490f4 <_start+12>   mov    ecx,0x4
0x80490f9 <_start+17>   mul    ecx
0x80490fb <_start+19>   add    ebx,0x5
0x80490fe <_start+22>   mov    edi,ebx
>0x8049100 <_start+24>  mov    eax,0x804a000
0x8049105 <_start+29>   call   0x804900f <sprint>
0x804910a <_start+34>   mov    eax,edi
0x804910c <_start+36>   call   0x8049086 <iprintfLF>

native process 5481 (asm) In: _start
0x080490f2 in _start ()
(gdb) si
0x080490f4 in _start ()
(gdb) si
0x080490f9 in _start ()
(gdb) si
0x080490fb in _start ()
(gdb) si
0x080490fe in _start ()
(gdb) si
0x08049100 in _start ()
(gdb) ■
```

Рисунок 2.19: Отладка

Отмечу, что перепутан порядок аргументов у инструкции add и что по окончании работы в edi отправляется ebx вместо eax

Исправленный код программы



The screenshot shows a terminal window with the title bar "kilic@kilic: ~/work/arch-pc/lab09". The file being edited is "lab9-5.asm". The code is as follows:

```
GNU nano 7.2                               lab9-5.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
```

Рисунок 2.20: Код исправлен

```
kilic@kili:~/work/arch-pc/lab09
eax      5          25
ffffcf10      xffffcf10
[ Register Values Unavailable ]
0490fe      x80490fe <_start+22>
5>
>           ,0x5
[ No Assembly Available ]
04a000      rint>
86 <iprintLF>
native process 5527 (asm) In: _start
0x080490f4 in _start ()
(gdb) si
0x080490f9 in _start ()
(gdb) si
0x080490fb in _start ()
(gdb) si
0x080490fe in _start ()
(gdb) c
Continuing.
Результат: 25
[Inferior 1 (process 5527) exited normally]
(gdb) L?? PC: 0x80490fe
L?? PC: ??
```

Рисунок 2.21: Проверка работы

## **3 Выводы**

Освоили работу с подпрограммами и отладчиком.