

# **Отчёт по лабораторной работе 9**

**Архитектура компьютера**

Сайд Курбанов

# **Содержание**

<b>1 Цель работы</b>	<b>5</b>
<b>2 Выполнение лабораторной работы</b>	<b>6</b>
2.1 Реализация подпрограмм в NASM . . . . .	6
2.2 Отладка программам с помощью GDB . . . . .	10
2.3 Задание для самостоятельной работы . . . . .	22
<b>3 Выводы</b>	<b>29</b>

# Список иллюстраций

2.1 Программа в файле lab9-1.asm . . . . .	7
2.2 Запуск программы lab9-1.asm . . . . .	8
2.3 Программа в файле lab9-1.asm . . . . .	9
2.4 Запуск программы lab9-1.asm . . . . .	10
2.5 Программа в файле lab9-2.asm . . . . .	11
2.6 Запуск программы lab9-2.asm в отладчике . . . . .	12
2.7 Дизассимилированный код . . . . .	13
2.8 Дизассимилированный код в режиме интел . . . . .	14
2.9 Точка остановки . . . . .	15
2.10 Изменение регистров . . . . .	16
2.11 Изменение регистров . . . . .	17
2.12 Изменение значения переменной . . . . .	18
2.13 Вывод значения регистра . . . . .	19
2.14 Вывод значения регистра . . . . .	20
2.15 Программа в файле lab9-3.asm . . . . .	21
2.16 Вывод значения регистра . . . . .	22
2.17 Программа в файле task-1.asm . . . . .	23
2.18 Запуск программы task-1.asm . . . . .	24
2.19 Код с ошибкой в файле task-2.asm . . . . .	25
2.20 Отладка task-2.asm . . . . .	26
2.21 Код исправлен в файле task-2.asm . . . . .	27
2.22 Проверка работы task-2.asm . . . . .	28

# **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**

### **2.1 Реализация подпрограмм в NASM**

Я создал каталог для выполнения лабораторной работы №9 и перешел в него.

В качестве примера рассмотрим программу, которая вычисляет арифметическое выражение  $f(x) = 2x + 7$  с использованием подпрограммы `calcul`. В этом примере значение  $x$  вводится с клавиатуры, а само выражение вычисляется в подпрограмме.

The screenshot shows a text editor window with the title bar "lab9-1.asm". Below the title bar is a toolbar with icons for file operations. The main area displays assembly code with line numbers on the left. The code includes sections for data (.data), bss (.bss), and text (.text). It uses standard assembly instructions like mov, call, and add, along with specific functions like sprint, sread, atoi, and iprintf. A note in line 19 indicates a call to a subroutine named \_calcul.

```
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2x+7=',0
5 SECTION .bss
6 x: RESB 80
7 rez: RESB 80
8
9 SECTION .text
10 GLOBAL _start
11 _start:
12 mov eax, msg
13 call sprint
14 mov ecx, x
15 mov edx, 80
16 call sread
17 mov eax,x
18 call atoi
19 call _calcul ; Вызов подпрограммы _calcul
20 mov eax,result
21 call sprint|
22 mov eax,[rez]
23 call iprintf
24 call quit
25 _calcul:
26 mov ebx,2
27 mul ebx
28 add eax,7
29 mov [rez],eax
30 ret ; выход из подпрограммы
31
```

Рисунок 2.1: Программа в файле lab9-1.asm

Первые строки программы отвечают за вывод сообщения на экран (с помощью вызова sprint), чтение данных, введенных с клавиатуры (с помощью вызова sread) и преобразование введенных данных из символьного вида в численный (с помощью вызова atoi).

После инструкции call \_calcul, которая передает управление подпрограмме \_calcul, будут выполнены инструкции, содержащиеся в подпрограмме.

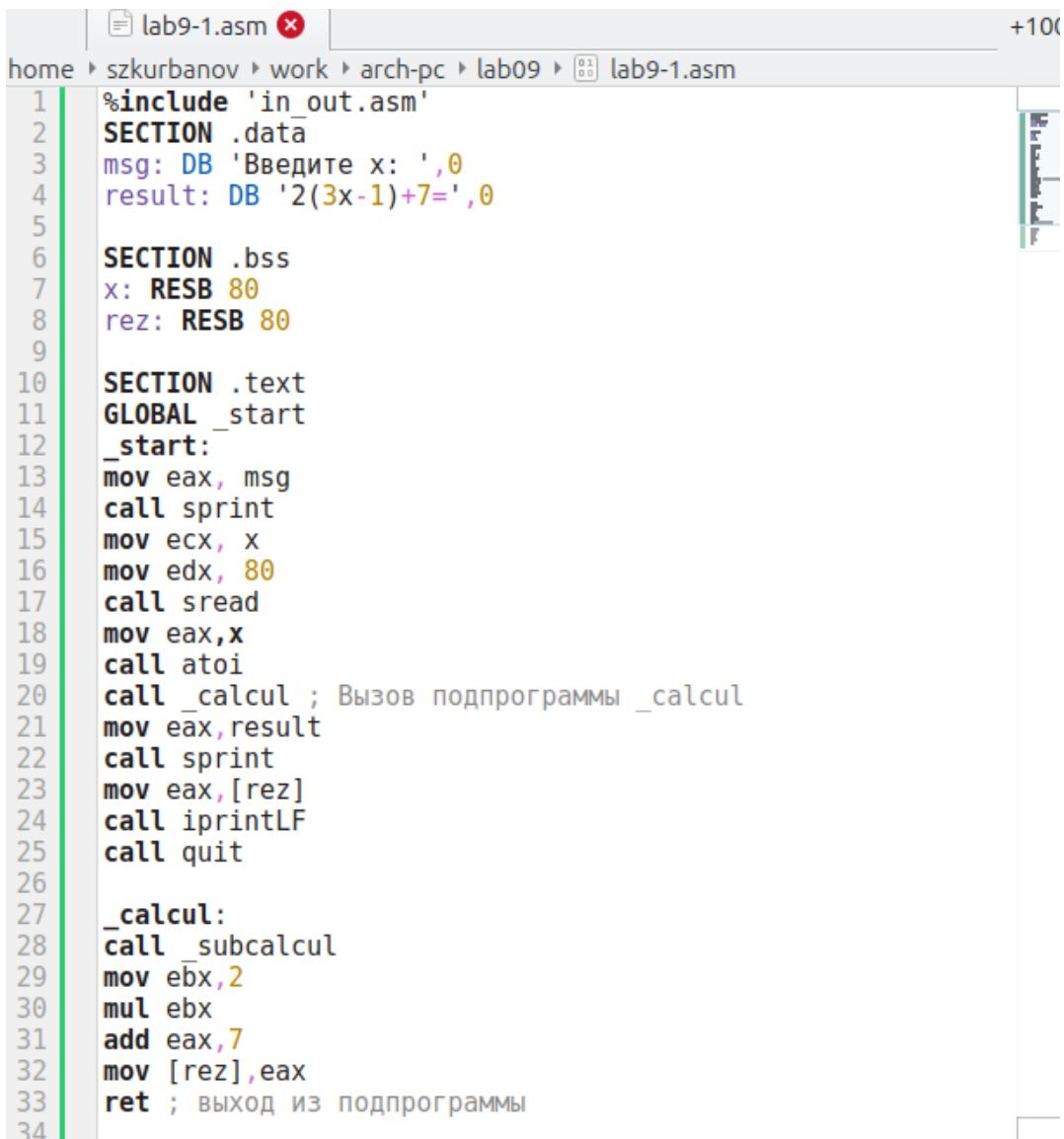
Инструкция `ret` является последней в подпрограмме и ее выполнение приводит к возврату в основную программу к инструкции, следующей за инструкцией `call`, которая вызвала данную подпрограмму.

Последние строки программы реализуют вывод сообщения (с помощью вызова `sprint`), вывод результата вычисления (с помощью вызова `iprintLF`) и завершение программы (с помощью вызова `quit`).

```
szkurbanov@said:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
szkurbanov@said:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
szkurbanov@said:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 3
2x+7=13
szkurbanov@said:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 4
2x+7=15
szkurbanov@said:~/work/arch-pc/lab09$
```

Рисунок 2.2: Запуск программы `lab9-1.asm`

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 Microsoft Notepad window with the file 'lab9-1.asm' open. The window title bar reads 'lab9-1.asm'. The file path in the title bar is 'home > szkurbanov > work > arch-pc > lab09 > lab9-1.asm'. The status bar at the bottom right shows '+100'. The code itself is as follows:

```
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2(3x-1)+7=',0
5
6 SECTION .bss
7 x: RESB 80
8 rez: RESB 80
9
10 SECTION .text
11 GLOBAL _start
12 _start:
13 mov eax, msg
14 call sprint
15 mov ecx, x
16 mov edx, 80
17 call sread
18 mov eax,x
19 call atoi
20 call _calcul ; Вызов подпрограммы _calcul
21 mov eax,result
22 call sprint
23 mov eax,[rez]
24 call iprintLF
25 call quit
26
27 _calcul:
28 call subcalcul
29 mov ebx,2
30 mul ebx
31 add eax,7
32 mov [rez],eax
33 ret ; выход из подпрограммы
34
```

Рисунок 2.3: Программа в файле lab9-1.asm

```
szkurbanov@said:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
szkurbanov@said:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
szkurbanov@said:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 3
2(3x-1)+7=23
szkurbanov@said:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 4
2(3x-1)+7=29
szkurbanov@said:~/work/arch-pc/lab09$
```

Рисунок 2.4: Запуск программы lab9-1.asm

## 2.2 Отладка программам с помощью GDB

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

```
home › szkurbanov › work › arch-pc › lab09 › lab9-2.asm
1 SECTION .data
2 msg1: db "Hello, ",0x0
3 msg1Len: equ $ - msg1
4 msg2: db "world!",0xa
5 msg2Len: equ $ - msg2
6
7 SECTION .text
8 global _start
9
10 _start:
11 mov eax, 4
12 mov ebx, 1
13 mov ecx, msg1
14 mov edx, msg1Len
15 int 0x80
16 mov eax, 4
17 mov ebx, 1
18 mov ecx, msg2
19 mov edx, msg2Len
20 int 0x80
21 mov eax, 1
22 mov ebx, 0
23 int 0x80|
```

Рисунок 2.5: Программа в файле 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 (сокращённо r).

```
szkurbanov@said:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
szkurbanov@said:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
szkurbanov@said:~/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/szkurbanov/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 11514) exited normally]
(gdb) 
```

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

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

```
szkurbanov@said: ~/work/arch-pc/lab09

<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 11514) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 11.
(gdb) r
Starting program: /home/szkurbanov/wrk/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: Дизассимилированный код

```
szkurbanov@said: ~/work/arch-pc/lab09
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    I
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: Дизассимилированный код в режиме интел

Установить точку останова можно командой `break` (кратко `b`). Типичный аргумент этой команды — место установки. Его можно задать или как номер строки программы (имеет смысл, если есть исходный файл, а программа компилировалась с информацией об отладке), или как имя метки, или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звездочка»

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 мож-

но 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:

```
szkurbanov@said:~/work/arch-pc/lab09
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp 0xffffcef0 0xffffcef0
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 11522 (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.

The screenshot shows a terminal window with the following content:

```
szkurbanov@said:~/work/arch-pc/lab09
Register group: general
eax      0x4          4
ecx      0x0          0
edx      0x0          0
ebx      0x0          0
esp      0xffffcef0    0xffffcef0
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 11522 (asm) In: _start
--Type <RET> for more, q to quit, c to continue without paging--
eflags     0x202        [ IF ]
cs         0x23         35
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 displays the command `szkurbanov@saид: ~/work/arch-pc/lab09`. Below this is a table titled "Register group: general" showing the following register values:

Register	Value	Description
eax	0x8	8
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x1	1
esp	0xfffffcf0	0xfffffcf0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0

Below the registers is a block of 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
```

At the bottom, there is a native process dump and some GDB commands:

```
native process 11522 (asm) In: _start
ds      0x2b      43
es      0x2b      43
fs      0x0       0
gs      0x0       0
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
```

The status bar at the bottom right shows "L16 PC: 0x8049016".

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 a terminal window with the following content:

```
Register group: general
eax          0x8          8
ecx          0x804a000    134520832
edx          0x8          8
ebx          0x1          1
esp          0xfffffcf0    0xfffffcf0
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 11522 (asm) In: _start
L16   PC: 0x8049016
0x804a000 <msg1>:        "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:        "world!\n\034"
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:        "hello, "
(gdb) set {char}0x804a008='L'
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:        "Lorld!\n\034"
(gdb)
```

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 terminal window with the following content:

```
szkurbanov@said:~/work/arch-pc/lab09
Register group: general
eax          0x8          8
ecx          0x804a000    134520832
edx          0x8          8
ebx          0x1          1
esp          0xffffcef0  0xffffcef0
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 11522 (asm) In: _start
$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)
```

Рисунок 2.13: Вывод значения регистра

С помощью команды set изменил значение регистра ebx

The screenshot shows a terminal window with the following content:

```
szkurbanov@said: ~/work/arch-pc/lab09
Register group: general
eax          0x8          8
ecx          0x804a000    134520832
edx          0x8          8
ebx          0x2          2
esp          0xfffffcf0    0xfffffcf0
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 11522 (asm) In: _start
$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)
```

Рисунок 2.14: Вывод значения регистра

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.

The screenshot shows a code editor window with the file 'lab9-3.asm' open. The file path is 'home > szkurbanov > work > arch-pc > lab09 > lab9-3.asm'. The code is written in assembly language and includes comments in Russian explaining the purpose of each instruction. The code starts with an include directive, defines a section, and sets up a global variable '\_start'. It then enters a loop at '\_start' where it processes arguments from the stack. It checks if there are more arguments using 'cmp ecx, 0' and 'jz \_end'. If there are, it prints the argument using 'call sprintLF' and loops back to '\_start'. If there are no more arguments, it exits the program using 'call quit'. The assembly code is as follows:

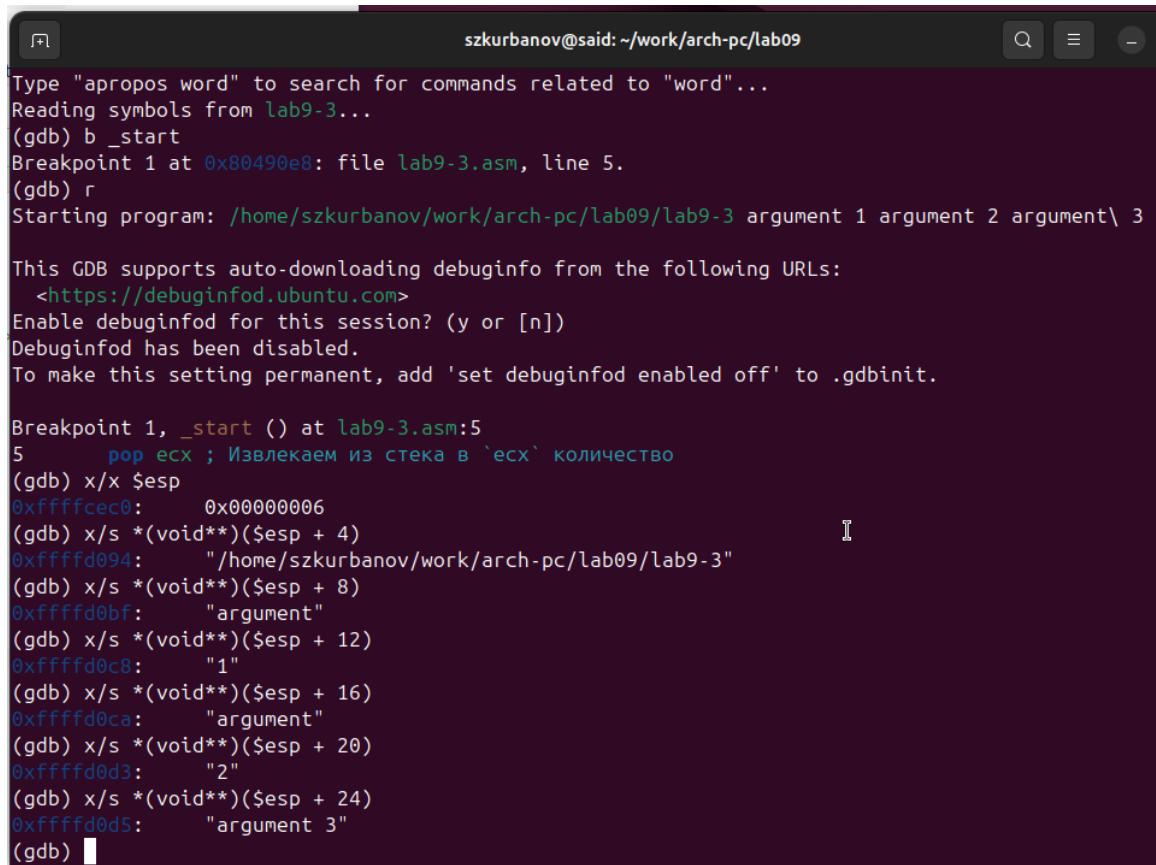
```
%include 'in_out.asm'
SECTION .text
global _start
_start:
    pop ecx ; Извлекаем из стека в `ecx` количество
    ; аргументов (первое значение в стеке)
    pop edx ; Извлекаем из стека в `edx` имя программы
    ; (второе значение в стеке)
    sub ecx, 1 ; Уменьшаем `ecx` на 1 (количество
    ; аргументов без названия программы)
next:
    cmp ecx, 0 ; проверяем, есть ли еще аргументы
    jz _end ; если аргументов нет выходим из цикла
    ; (переход на метку `_end`)
    pop eax ; иначе извлекаем аргумент из стека
    call sprintLF ; вызываем функцию печати
    loop next ; переход к обработке следующего
    ; аргумента (переход на метку `next`)
_end:
    call quit
```

Рисунок 2.15: Программа в файле lab9-3.asm

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

Адрес вершины стека хранится в регистре esp и по этому адресу располагается 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 with the following content:

```
szkurbanov@said: ~/work/arch-pc/lab09
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab9-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 5.
(gdb) r
Starting program: /home/szkurbanov/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.

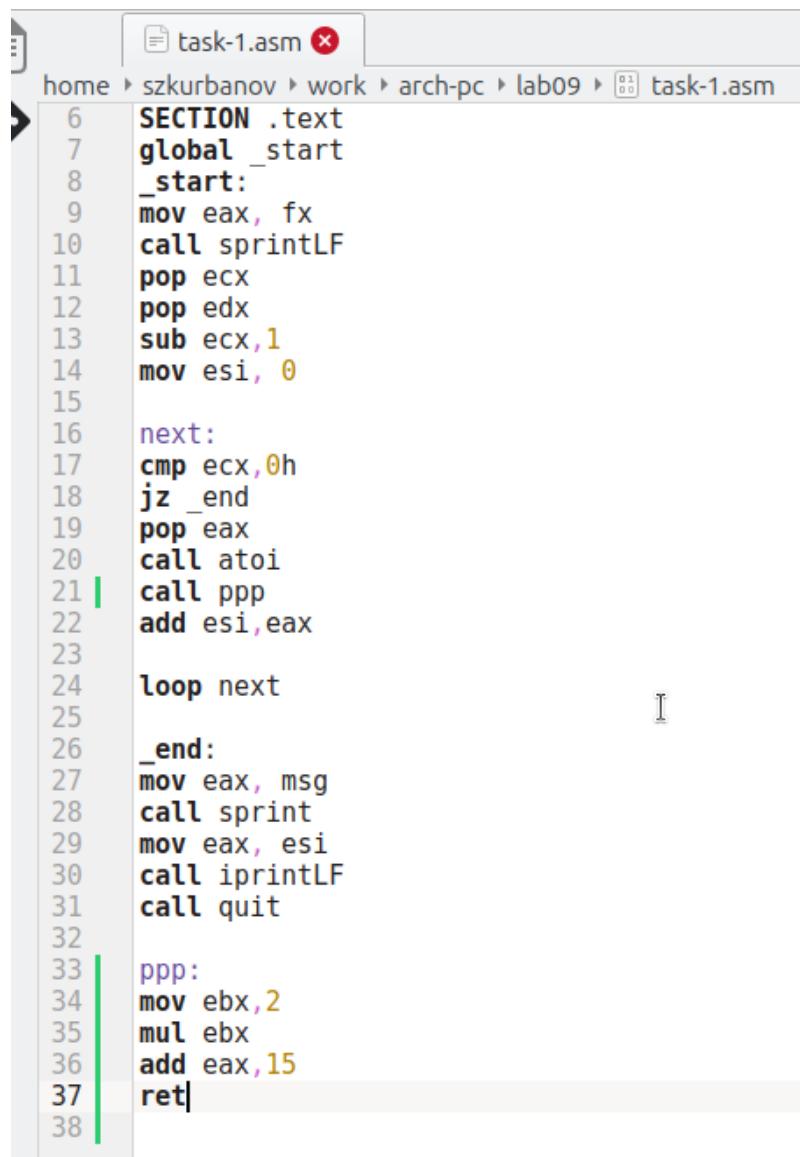
Breakpoint 1, _start () at lab9-3.asm:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb) x/x $esp
0xfffffec0: 0x00000006
(gdb) x/s *(void**)( $esp + 4)
0xfffffd094: "/home/szkurbanov/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)( $esp + 8)
0xfffffd0bf: "argument"
(gdb) x/s *(void**)( $esp + 12)
0xfffffd0c8: "1"
(gdb) x/s *(void**)( $esp + 16)
0xfffffd0ca: "argument"
(gdb) x/s *(void**)( $esp + 20)
0xfffffd0d3: "2"
(gdb) x/s *(void**)( $esp + 24)
0xfffffd0d5: "argument 3"
(gdb)
```

Рисунок 2.16: Вывод значения регистра

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

## 2.3 Задание для самостоятельной работы

Я переписал программу из лабораторной работы №8, чтобы вычислить значение функции  $f(x)$  в виде подпрограммы.



The screenshot shows a code editor window with the file "task-1.asm" open. The file path is displayed at the top: "home > szkurbanov > work > arch-pc > lab09 > task-1.asm". The assembly code is as follows:

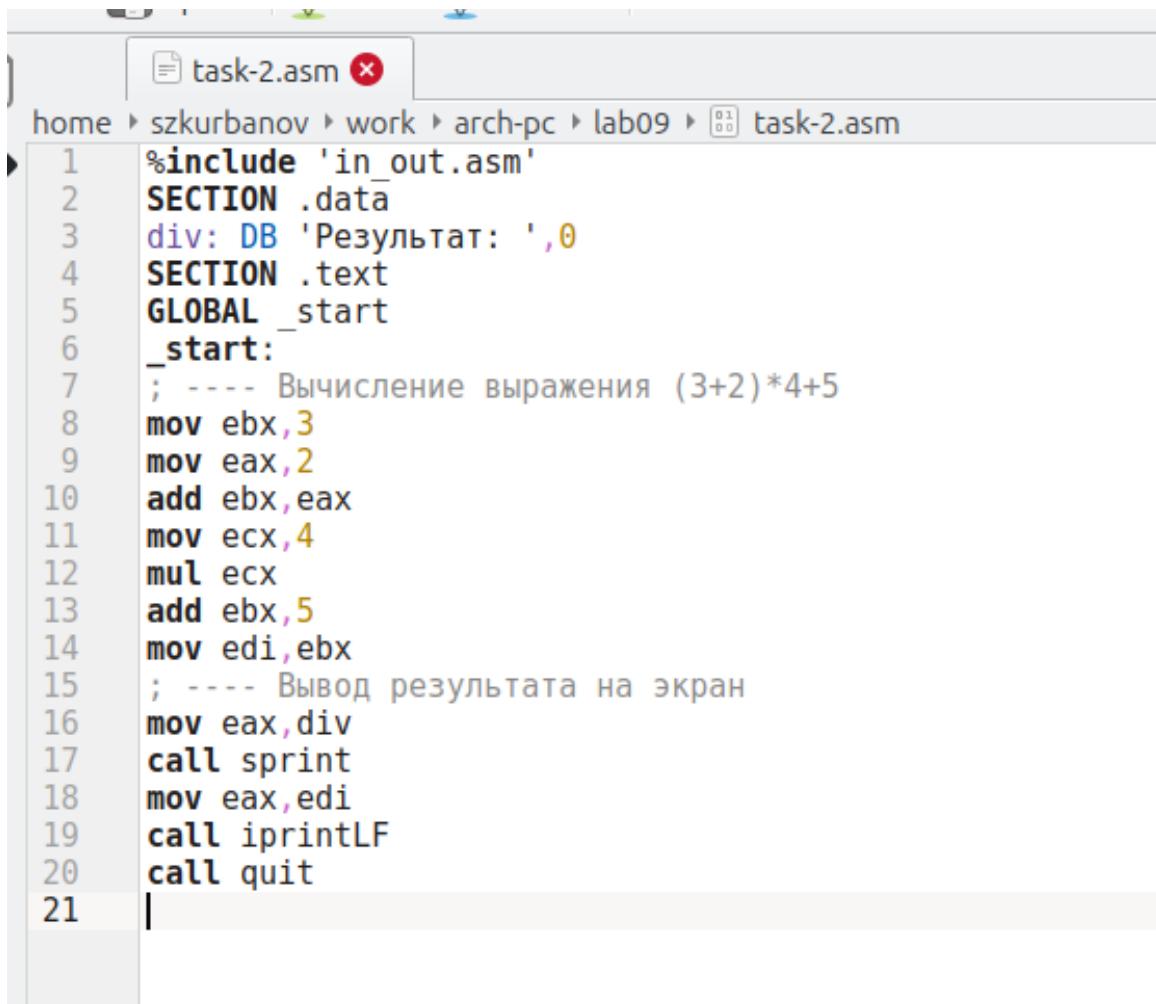
```
6 SECTION .text
7 global _start
8 _start:
9     mov eax, fx
10    call sprintLF
11    pop ecx
12    pop edx
13    sub ecx, 1
14    mov esi, 0
15
16 next:
17    cmp ecx, 0h
18    jz _end
19    pop eax
20    call atoi
21    call ppp
22    add esi, eax
23
24 loop next
25
26 _end:
27    mov eax, msg
28    call sprint
29    mov eax, esi
30    call iprintLF
31    call quit
32
33 ppp:
34    mov ebx, 2
35    mul ebx
36    add eax, 15
37    ret
38
```

Рисунок 2.17: Программа в файле task-1.asm

```
szkurbanov@said:~/work/arch-pc/lab09$ nasm -f elf task-1.asm
szkurbanov@said:~/work/arch-pc/lab09$ ld -m elf_i386 task-1.o -o task-1
szkurbanov@said:~/work/arch-pc/lab09$ ./task-1
f(x)= 2x + 15
Результат: 0
szkurbanov@said:~/work/arch-pc/lab09$ ./task-1 1
f(x)= 2x + 15
Результат: 17
szkurbanov@said:~/work/arch-pc/lab09$ ./task-1 1 3 4 5 6
f(x)= 2x + 15
Результат: 113
szkurbanov@said:~/work/arch-pc/lab09$ █
```

Рисунок 2.18: Запуск программы task-1.asm

Приведенный ниже листинг программы вычисляет выражение  $(3+2)*4+5$ . Однако, при запуске, программа дает неверный результат. Я проверил это и решил использовать отладчик GDB для анализа изменений значений регистров и определения ошибки.



The screenshot shows a Windows Notepad window with the file "task-2.asm" open. The code is written in assembly language. There is a syntax error in line 15 where the label "div:" is used as a comment. The code is as follows:

```
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ----- Вычисление выражения (3+2)*4+5
8 mov ebx,3
9 mov eax,2
10 add ebx,eax
11 mov ecx,4
12 mul ecx
13 add ebx,5
14 mov edi,ebx
15 ; ----- Вывод результата на экран
16 mov eax,div
17 call sprint
18 mov eax,edi
19 call iprintLF
20 call quit
21 |
```

Рисунок 2.19: Код с ошибкой в файле task-2.asm

The screenshot shows a terminal window titled "szkurbanov@said: ~/work/arch-pc/lab09". It displays assembly code and register values. The assembly code includes calls to `sprint`, `iprintLF`, and `quit`. The registers show `eax` containing `134520832` and `edi` containing `0`. The stack shows the command history of the debugger.

```
eax      s              134520832
        0
ffffcef0          fffffcef0
[ Register Values Unavailable ]
        0
0x8049105 <_start+29>  call   0x804900f <sprint>
0x804910c <_start+36>  call   0x8049086 <iprintLF>
0x8049111 <_start+41>  call   0x80490db <quit>

native process 11635 (asm) In: _start
(gdb) s
No process (asm) In:
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) c
Continuing.
Результат: 10
[Inferior 1 (process 11635) exited normally]
(gdb)
```

Рисунок 2.20: Отладка task-2.asm

Я заметил, что порядок аргументов в инструкции add был перепутан и что при завершении работы, вместо eax, значение отправлялось в edi. Вот исправленный код программы:

The screenshot shows a code editor window with the file 'task-2.asm' open. The file path is 'home > szkurbanov > work > arch-pc > lab09 > task-2.asm'. The code is written in assembly language and performs the calculation  $(3+2)*4+5$  and prints the result '27' to the screen. The code includes directives for including files, defining sections (.data and .text), and global symbols. It uses various CPU registers (eax, ebx, ecx, edx) and system calls for output.

```
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ----- Вычисление выражения (3+2)*4+5
8 mov ebx,3
9 mov eax,2
10 add eax,ebx
11 mov ecx,4
12 mul ecx
13 add eax,5
14 mov edi,eax
15 ; ----- Вывод результата на экран
16 mov eax,div
17 call sprint
18 mov eax,edi
19 call iprintLF
20 call quit
21
```

Рисунок 2.21: Код исправлен в файле task-2.asm

The screenshot shows a terminal window with the following content:

```
szkurbanov@said: ~/work/arch-pc/lab09
eax      $25
ffffcef0          xffffcef0
[ Register Values Unavailable ]

0x80490fe <_start+22>    mov    edi,ea4
0x8049100 <_start+24>    mul    eax,0x804a000
0x8049105 <_start+29>    call   0x804900f <sprint>
>0x804910a <_start+34>   mov    edi,edi
0x804910c <_start+36>    call   0x8049086 <iprintLF>
0x8049111 <_start+41>    call   0x80490db <quit>t

native process 11658 (asm) In: _start
BreakpoNo process (asm) In: :8
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) c
Continuing.
Результат: 25
[Inferior 1 (process 11658) exited normally]
(gdb) L14 PC: 0x80490fe
L?? PC: ??
```

Рисунок 2.22: Проверка работы task-2.asm

## **3 Выводы**

Освоили работу с подпрограммами и отладчиком.