

Отчёт по лабораторной работе 9

дисциплина: Архитектура компьютера

Тяпкова Альбина НММбд-04-24

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Реализация подпрограмм в NASM	6
2.2	Отладка программ с помощью GDB	10
2.3	Задание для самостоятельной работы	21
3	Выводы	27

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	9
2.5	Программа в файле 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	Программа в файле lab9-3.asm	20
2.16	Вывод значения регистра	21
2.17	Программа в файле task-1.asm	22
2.18	Запуск программы task-1.asm	22
2.19	Код с ошибкой в файле task-2.asm	23
2.20	Отладка task-2.asm	24
2.21	Код исправлен в файле task-2.asm	25
2.22	Проверка работы task-2.asm	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

2.1 Реализация подпрограмм в NASM

Я создала каталог для выполнения лабораторной работы №9 и перешла 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.

```
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 iprintLF
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
```

Рис. 2.1: Программа в файле lab9-1.asm

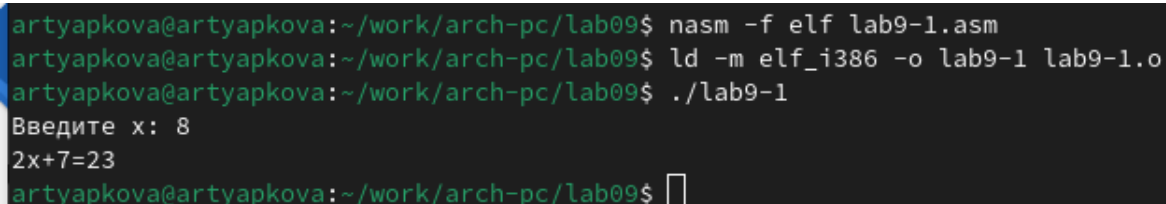
Первые строки программы отвечают за вывод сообщения на экран (с помощью вызова `sprint`), чтение данных, введенных с клавиатуры (с помощью вызова `sread`) и преобразование введенных данных из символьного вида в численный (с помощью вызова `atoi`).

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

Инструкция `ret` является последней в подпрограмме и её выполнение приво-

дит к возврату в основную программу к инструкции, следующей за инструкцией `call`, которая вызвала данную подпрограмму.

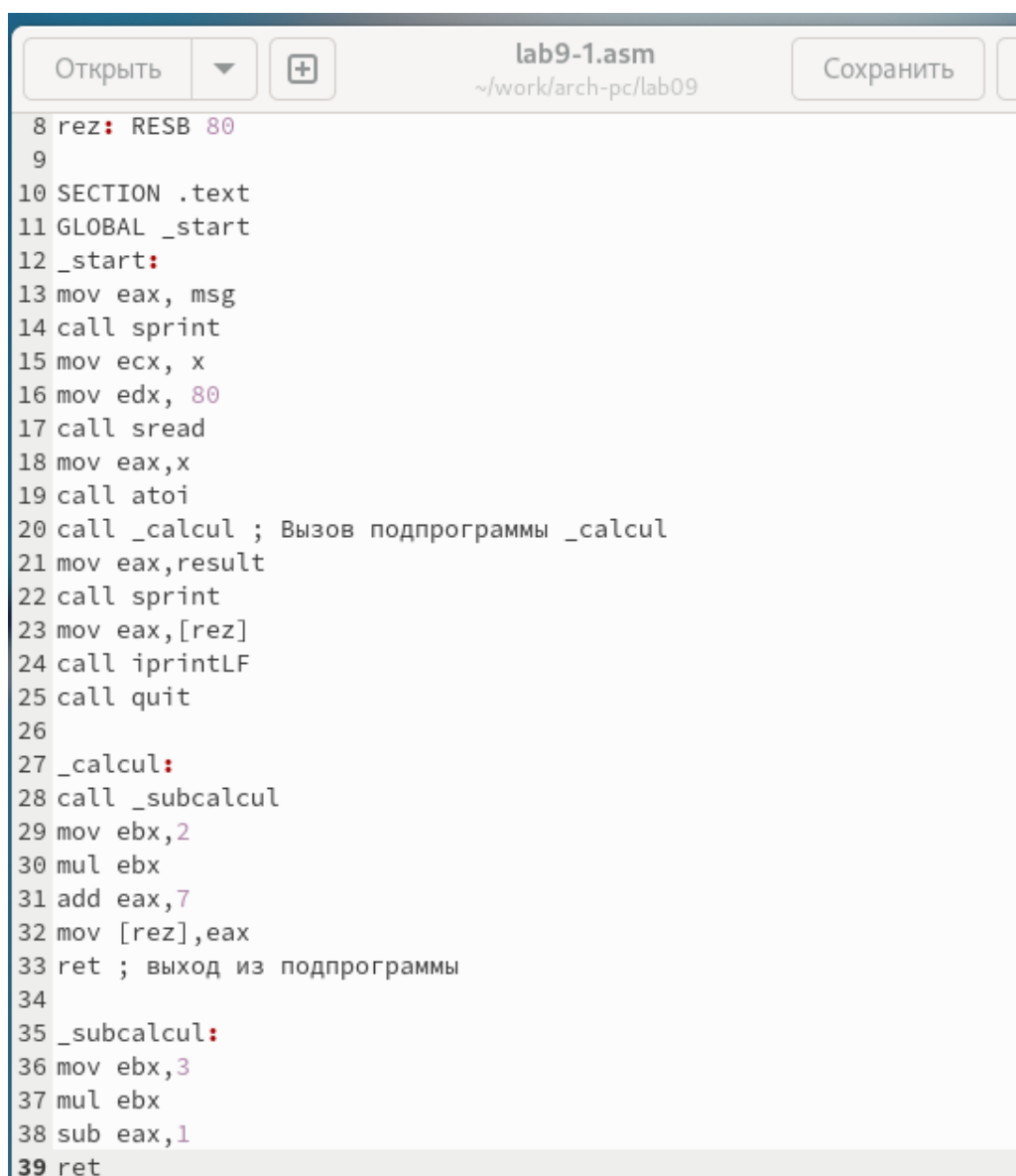
Последние строки программы реализуют вывод сообщения (с помощью вызова `sprint`), вывод результата вычисления (с помощью вызова `iprintLF`) и завершение программы (с помощью вызова `quit`).



```
artyapkova@artyapkova:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
artyapkova@artyapkova:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
artyapkova@artyapkova:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 8
2x+7=23
artyapkova@artyapkova:~/work/arch-pc/lab09$
```

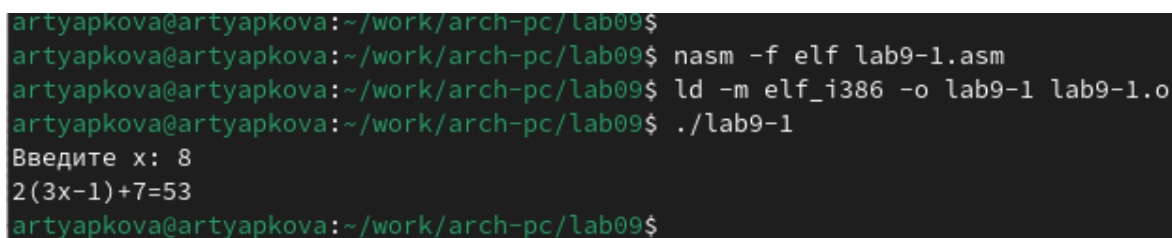
Рис. 2.2: Запуск программы lab9-1.asm

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



```
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
35 _subcalcul:
36 mov ebx, 3
37 mul ebx
38 sub eax, 1
39 ret
```

Рис. 2.3: Программа в файле lab9-1.asm

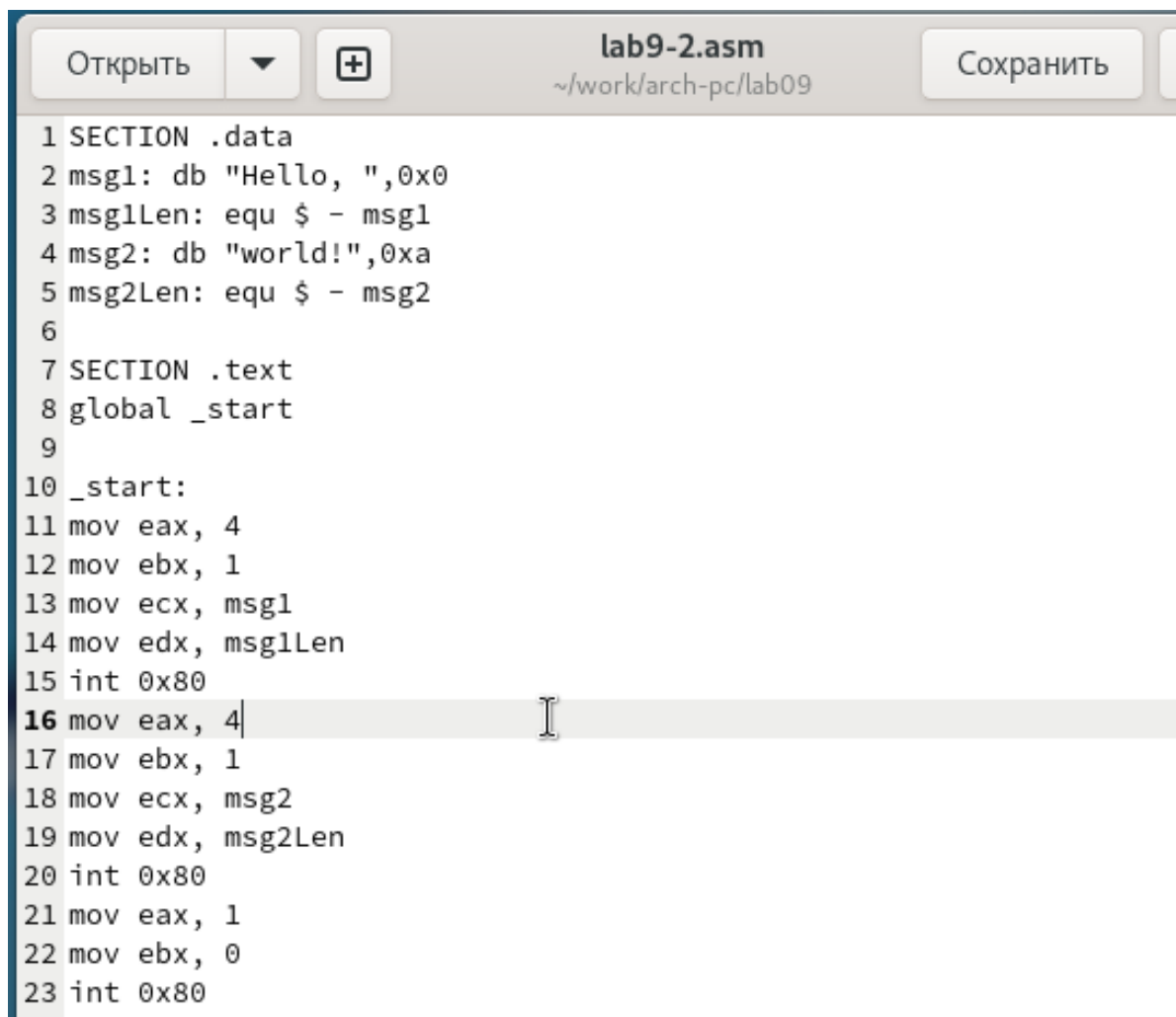


```
artyapkova@artyapkova:~/work/arch-pc/lab09$
artyapkova@artyapkova:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
artyapkova@artyapkova:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
artyapkova@artyapkova:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 8
2(3x-1)+7=53
artyapkova@artyapkova:~/work/arch-pc/lab09$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

Рис. 2.4: Запуск программы lab9-1.asm

2.2 Отладка программ с помощью GDB

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



```
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
```

Рис. 2.5: Программа в файле `lab9-2.asm`

После того как я получила исполняемый файл, для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для чего трансляцию программ следует проводить с ключом `-g`.

Загрузила исполняемый файл в отладчик GDB и проверила работу программы, запустив её в оболочке GDB с помощью команды `run` (сокращенно `r`).

```

artyapkova@artyapkova:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
artyapkova@artyapkova:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
artyapkova@artyapkova:~/work/arch-pc/lab09$ gdb lab9-2
GNU gdb (Fedora Linux) 15.1-1.fc39
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-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab9-2...
(gdb) r
Starting program: /home/artyapkova/work/arch-pc/lab09/lab9-2

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.fedoraproject.org/>
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 3541) exited normally]
(gdb) █

```

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

Для более подробного анализа программы установила брейкпоинт на метку start, с которой начинается выполнение любой ассемблерной программы, и запустила её. Посмотрела дизассемблированный код программы.

```
artyapkova@artyapkova:~/work/arch-pc/lab09 — gdb lab9-2
This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.fedoraproject.org/>
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 3541) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 11.
(gdb) r
Starting program: /home/artyapkova/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) 
```

Рис. 2.7: Дизассемблированный код

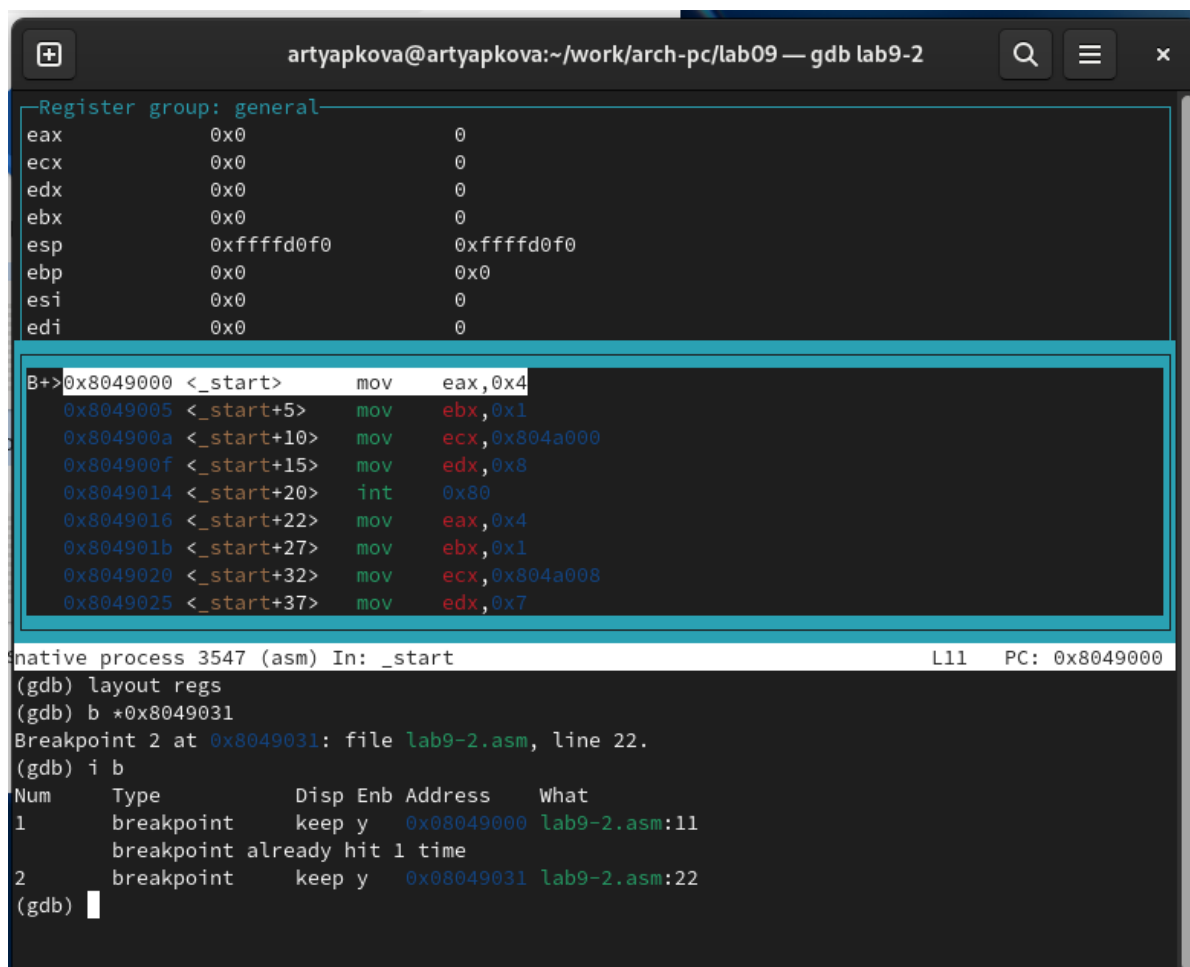
```
artyapkova@artyapkova:~/work/arch-pc/lab09 — gdb lab9-2
=> 0x08049000 <+0>:      mov     $0x4,%eax
0x08049005 <+5>:      mov     $0x1,%ebx
0x0804900a <+10>:     mov     $0x804a000,%ecx
0x0804900f <+15>:     mov     $0x8,%edx
0x08049014 <+20>:     int     $0x80
0x08049016 <+22>:     mov     $0x4,%eax
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) 
```

Рис. 2.8: Дизассемблированный код в режиме интел

Для установки точки останова использовала команду `break` (кратко `b`). Типичный аргумент этой команды — место установки. Его можно задать либо как номер строки программы (если есть исходный файл и программа компилировалась с отладочной информацией), либо как имя метки, или как адрес. Чтобы избежать путаницы с номерами, перед адресом ставится «звездочка».

На предыдущих шагах была установлена точка останова по имени метки `_start`. Проверила это с помощью команды `info breakpoints` (кратко `i b`). Затем установила ещё одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей

инструкции. Определила адрес предпоследней инструкции (`mov ebx,0x0`) и установила точку.



The screenshot shows a GDB terminal window with the title bar "artyapkova@artyapkova:~/work/arch-pc/lab09 — gdb lab9-2". The window is divided into several sections. The top section, titled "Register group: general", lists the values of registers: `eax` (0x0), `ecx` (0x0), `edx` (0x0), `ebx` (0x0), `esp` (0xffffd0f0), `ebp` (0x0), `esi` (0x0), and `edi` (0x0). Below this, a list of assembly instructions is displayed, starting with `B>0x8049000 <_start> mov eax,0x4`. The instructions are: `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`, and `0x8049025 <_start+37> mov edx,0x7`. The bottom section shows the GDB prompt with the command `(gdb) layout regs` and `(gdb) b *0x8049031`. It then displays the message "Breakpoint 2 at 0x8049031: file lab9-2.asm, line 22." and the command `(gdb) i b`. The output shows two breakpoints: `1 breakpoint keep y 0x08049000 lab9-2.asm:11` and `2 breakpoint keep y 0x08049031 lab9-2.asm:22`. The GDB prompt is at the bottom.

```
artyapkova@artyapkova:~/work/arch-pc/lab09 — gdb lab9-2
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd0f0 0xffffd0f0
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
0x8049025 <_start+37> mov edx,0x7

native process 3547 (asm) In: _start L11 PC: 0x8049000
(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) 
```

Рис. 2.9: Точка остановки

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

```
artyapkova@artyapkova:~/work/arch-pc/lab09 — gdb lab9-2
Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd0f0 0xffffd0f0
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
0x8049025 <_start+37>     mov     edx,0x7

native process 3547 (asm) In: _start L12 PC: 0x8049005
eip      0x8049000      0x8049000 <_start>
eflags   0x202          [ IF ]
--Type <RET> for more, q to quit, c to continue without paging--
cs       0x23           35
ss       0x2b           43
ds       0x2b           43
es       0x2b           43
fs       0x0            0
gs       0x0            0
(gdb) si
(gdb) 
```

Рис. 2.10: Изменение регистров

The screenshot shows a GDB terminal window with the title bar "artyapkova@artyapkova:~/work/arch-pc/lab09 — gdb lab9-2". The window is divided into three main sections. The top section, titled "Register group: general", displays the values of general-purpose registers:

Register	Value (hex)	Value (decimal)
eax	0x8	8
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x1	1
esp	0xffffd0f0	0xffffd0f0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0

. The middle section displays assembly code starting from address 0x8049000, with instructions like `mov eax, 0x4`, `mov ebx, 0x1`, `mov ecx, 0x804a000`, `mov edx, 0x8`, `int 0x80`, and subsequent `mov` instructions. The bottom section shows the state of segment registers:

Segment	Value (hex)	Value (decimal)
ss	0x2b	43
ds	0x2b	43
es	0x2b	43
fs	0x0	0
gs	0x0	0

, along with GDB prompts `(gdb) si` repeated five times.

Рис. 2.11: Изменение регистров

Посмотрела значение переменной `msg1` по имени и значение переменной `msg2` по адресу.

Изменить значение для регистра или ячейки памяти можно с помощью команды `set`, указав имя регистра или адрес. Я изменила первый символ переменной `msg1`.


```
artyapkova@artyapkova:~/work/arch-pc/lab09 — gdb lab9-2
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd0f0 0xffffd0f0
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
    0x8049025 <_start+37> mov     edx,0x7

native process 3547 (asm) In: _start L16 PC: 0x8049016
(gdb) x/1sb &msg1
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)
```

Рис. 2.12: Изменение значения переменной

Я вывела значение регистра `edx` в различных форматах (в шестнадцатеричном, двоичном и символьном).

```
artyapkova@artyapkova:~/work/arch-pc/lab09 — gdb lab9-2

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd0f0 0xffffd0f0
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
    0x8049025 <_start+37> mov     edx,0x7

native process 3547 (asm) In: _start L16 PC: 0x8049016
(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)
```

Рис. 2.13: Вывод значения регистра

С помощью команды `set` изменила значение регистра `ebx`.

The screenshot shows a GDB terminal window with the title bar "artyapkova@artyapkova:~/work/arch-pc/lab09 — gdb lab9-2". The window is divided into three main sections. The top section, titled "Register group: general", displays the values of general-purpose registers:

Register	Value (hex)	Value (decimal)
eax	0x8	8
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x2	2
esp	0xffffd0f0	0xffffd0f0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0

. The middle section displays assembly code starting from address 0x8049000, with instructions like `mov eax, 0x4`, `mov ebx, 0x1`, `mov ecx, 0x804a000`, `mov edx, 0x8`, `int 0x80`, and others. The bottom section shows the GDB command prompt with the command `(gdb) p/t $edx` resulting in `$6 = 1000`, and `(gdb) p/x $edx` resulting in `$7 = 0x8`. It also shows `(gdb) set $ebx='2'` and `(gdb) p/s $ebx` resulting in `$8 = 50`, and `(gdb) set $ebx=2` and `(gdb) p/s $ebx` resulting in `$9 = 2`. The status bar at the bottom indicates "native process 3547 (asm) In: _start", "L16", and "PC: 0x8049016".

```
artyapkova@artyapkova:~/work/arch-pc/lab09 — gdb lab9-2

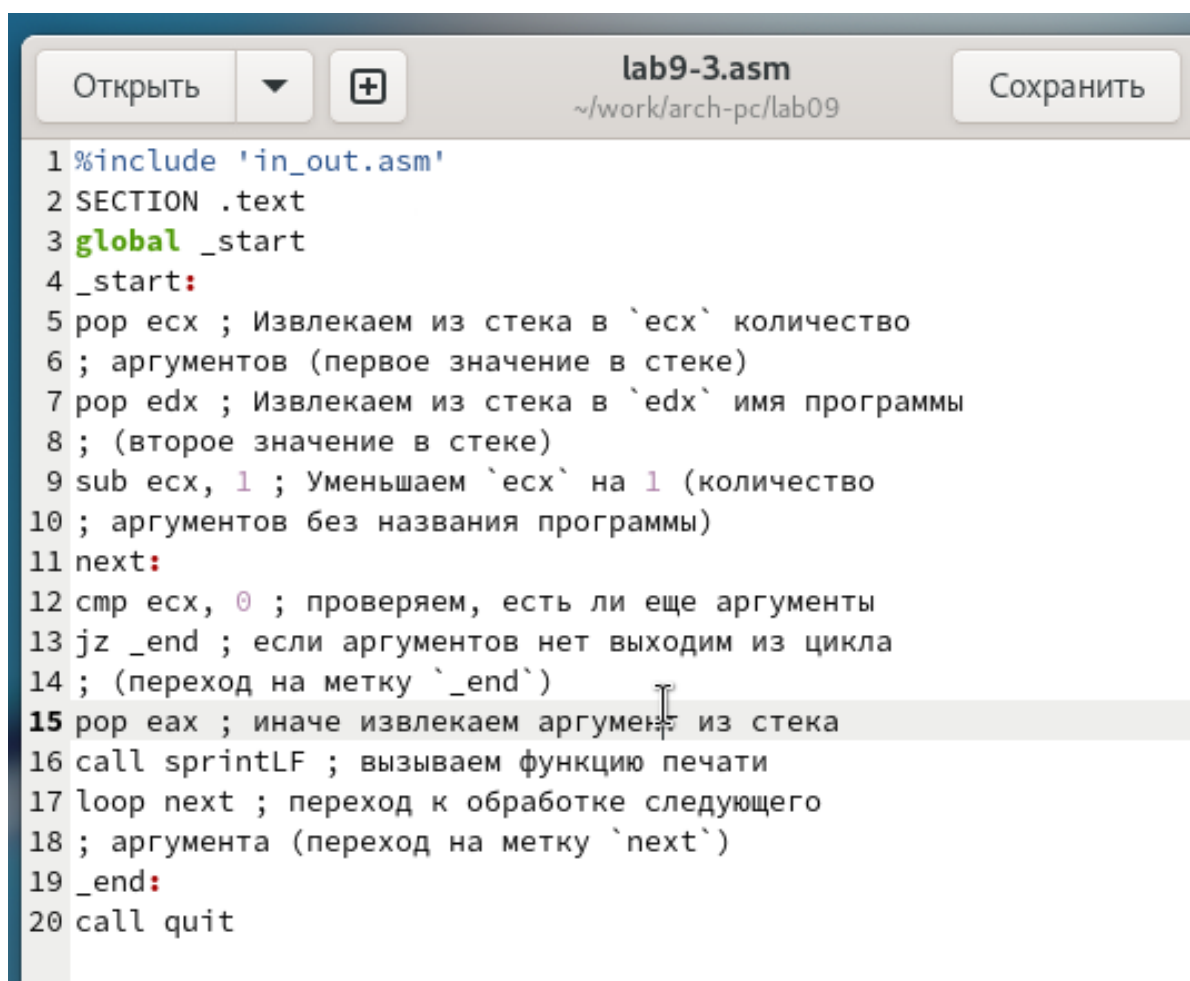
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x2      2
esp      0xffffd0f0 0xffffd0f0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

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
0x8049025 <_start+37>     mov     edx, 0x7

native process 3547 (asm) In: _start      L16      PC: 0x8049016
(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) 
```

Рис. 2.14: Вывод значения регистра

Я скопировала файл `lab8-2.asm`, созданный при выполнении лабораторной работы №8, с программой, выводящей на экран аргументы командной строки. Создала исполняемый файл. Для загрузки программы с аргументами в GDB необходимо использовать ключ `--args`. Загрузила исполняемый файл в отладчик, указав аргументы.



```
1 %include 'in_out.asm'
2 SECTION .text
3 global _start
4 _start:
5 pop ecx ; Извлекаем из стека в `ecx` количество
6 ; аргументов (первое значение в стеке)
7 pop edx ; Извлекаем из стека в `edx` имя программы
8 ; (второе значение в стеке)
9 sub ecx, 1 ; Уменьшаем `ecx` на 1 (количество
10 ; аргументов без названия программы)
11 next:
12 cmp ecx, 0 ; проверяем, есть ли еще аргументы
13 jz _end ; если аргументов нет выходим из цикла
14 ; (переход на метку `_end`)
15 pop eax ; иначе извлекаем аргумент из стека
16 call sprintf ; вызываем функцию печати
17 loop next ; переход к обработке следующего
18 ; аргумента (переход на метку `next`)
19 _end:
20 call quit
```

Рис. 2.15: Программа в файле lab9-3.asm

Для начала установила точку останова перед первой инструкцией в программе и запустила её.

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`, `аргумент2` и `аргумент 3`.

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

```
artyapkova@artyapkova:~/work/arch-pc/lab09 — gdb --args lab9-3 argument 1 a...
Reading symbols from lab9-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 5.
(gdb)
Note: breakpoint 1 also set at pc 0x80490e8.
Breakpoint 2 at 0x80490e8: file lab9-3.asm, line 5.
(gdb) r
Starting program: /home/artyapkova/work/arch-pc/lab09/lab9-3 argument 1 argument 2 argument\ 3

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

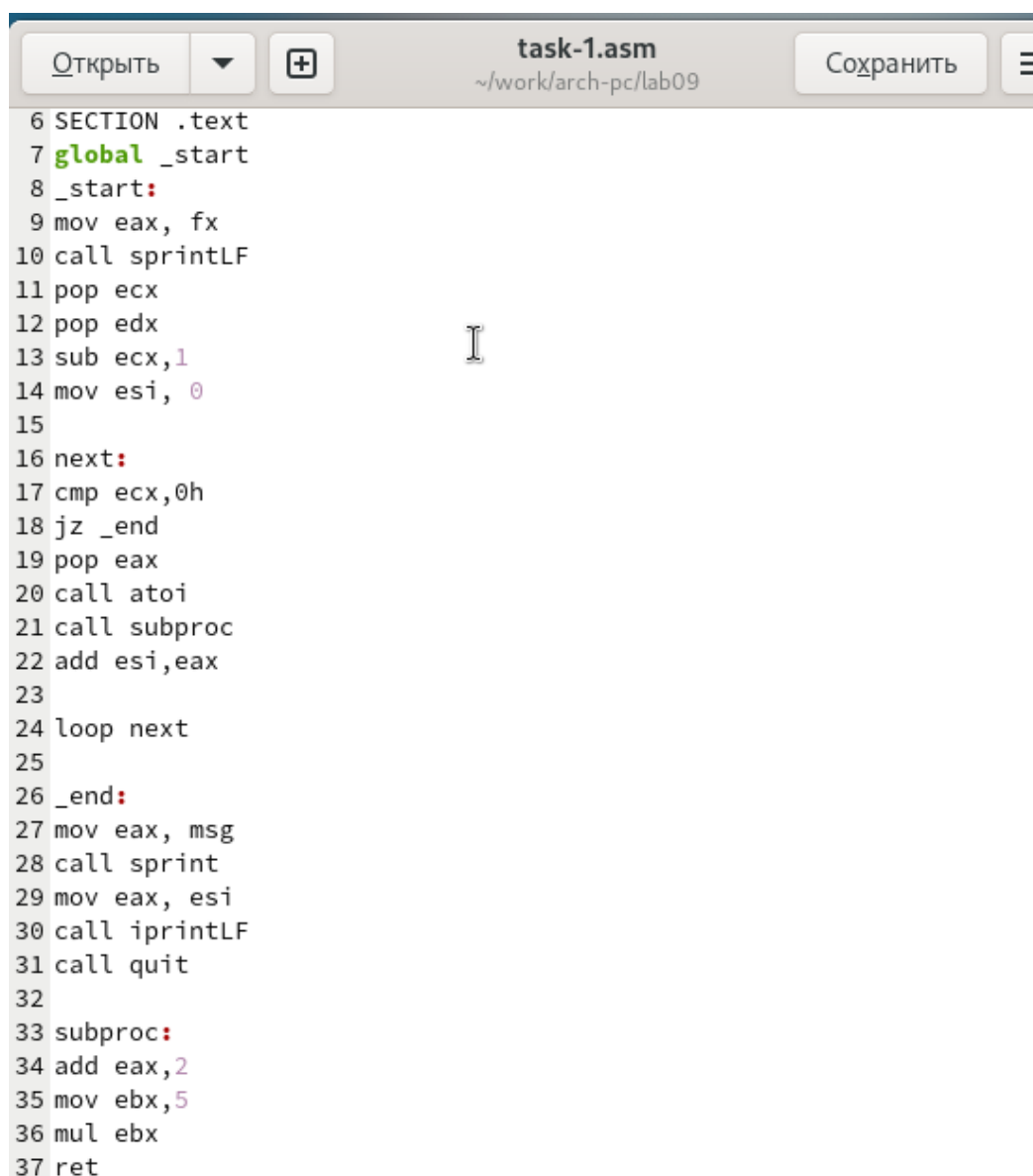
Breakpoint 1, _start () at lab9-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffd0c0:      0x00000006
(gdb) x/s *(void**)(esp + 4)
0xffffd283:      "/home/artyapkova/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd2ae:      "argument"
(gdb) x/s *(void**)(esp + 12)
0xffffd2b7:      "1"
(gdb) x/s *(void**)(esp + 16)
0xffffd2b9:      "argument"
(gdb) x/s *(void**)(esp + 20)
0xffffd2c2:      "2"
(gdb) x/s *(void**)(esp + 24)
0xffffd2c4:      "argument 3"
(gdb)
```

Рис. 2.16: Вывод значения регистра

Объяснила, почему шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12]) — шаг равен размеру переменной (4 байта).

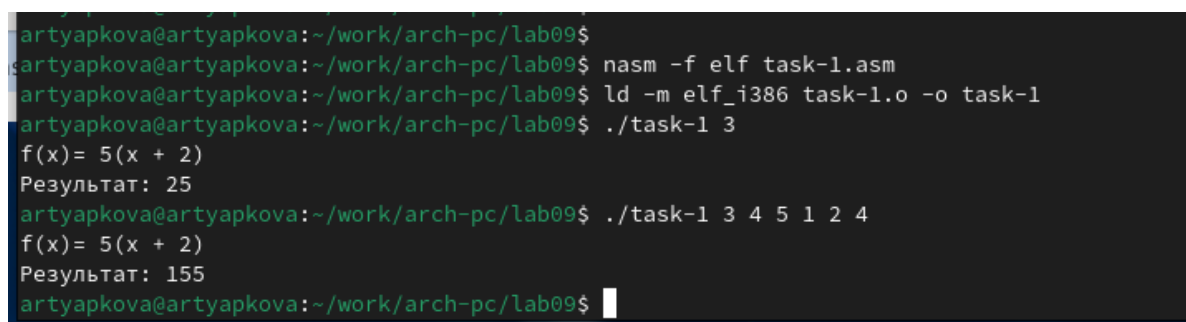
2.3 Задание для самостоятельной работы

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



```
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 subproc
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 subproc:
34 add eax, 2
35 mov ebx, 5
36 mul ebx
37 ret
```

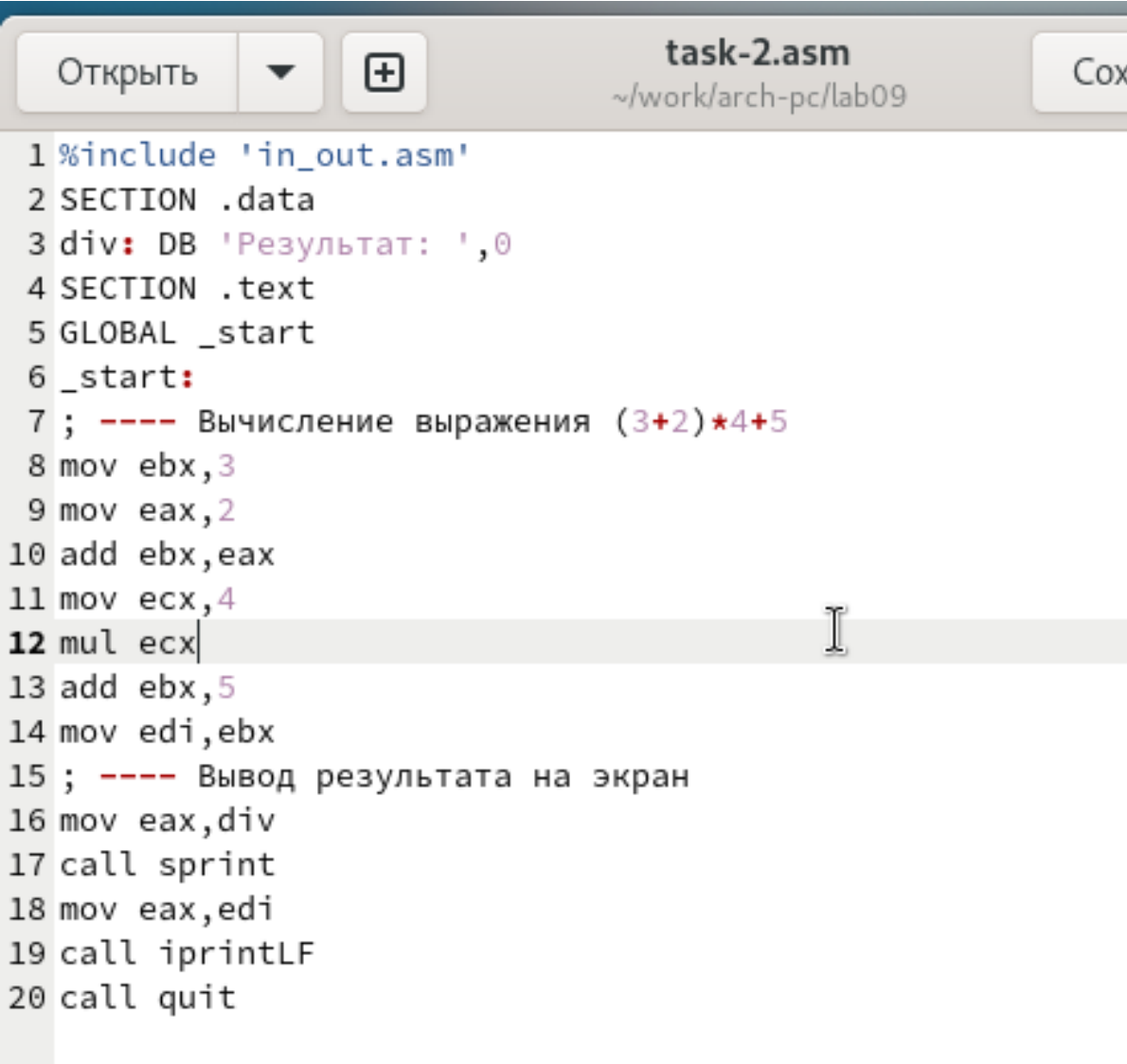
Рис. 2.17: Программа в файле task-1.asm



```
artyapkova@artyapkova:~/work/arch-pc/lab09$
artyapkova@artyapkova:~/work/arch-pc/lab09$ nasm -f elf task-1.asm
artyapkova@artyapkova:~/work/arch-pc/lab09$ ld -m elf_i386 task-1.o -o task-1
artyapkova@artyapkova:~/work/arch-pc/lab09$ ./task-1 3
f(x)= 5(x + 2)
Результат: 25
artyapkova@artyapkova:~/work/arch-pc/lab09$ ./task-1 3 4 5 1 2 4
f(x)= 5(x + 2)
Результат: 155
artyapkova@artyapkova:~/work/arch-pc/lab09$
```

Рис. 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 проверила это и решила использовать отладчик GDB для анализа изменений значений регистров и определения ошибки.



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

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
```

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

The screenshot shows a GDB window titled "artyapkova@artyapkova:~/work/arch-pc/lab09 — gdb task-2". The "Register group: general" pane shows the following values: eax=0x8, ecx=0x4, edx=0x0, ebx=0xa, esp=0xffffd0f0, ebp=0x0, esi=0x0, and edi=0xa. The assembly pane shows the following instructions: 0x80490f4: mov ecx,0x4; 0x80490f9: mul ecx; 0x80490fb: add ebx,0x5; 0x80490fe: mov edi,ebx; 0x8049100: mov eax,0x804a000; 0x8049105: call 0x804900f <sprint>; 0x804910a: mov eax,edi; 0x804910c: call 0x8049086 <iprintfLF>; 0x8049111: call 0x80490db <quit>. The status bar shows "native process 3671 (asm) In: _start L16 PC: 0x8049100". The console shows the command "Breakpoint 1, _start () at task-2.asm:8" and several "si" commands.

```
artyapkova@artyapkova:~/work/arch-pc/lab09 — gdb task-2

Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd0f0 0xffffd0f0
ebp      0x0      0x0
esi      0x0      0
edi      0xa      10

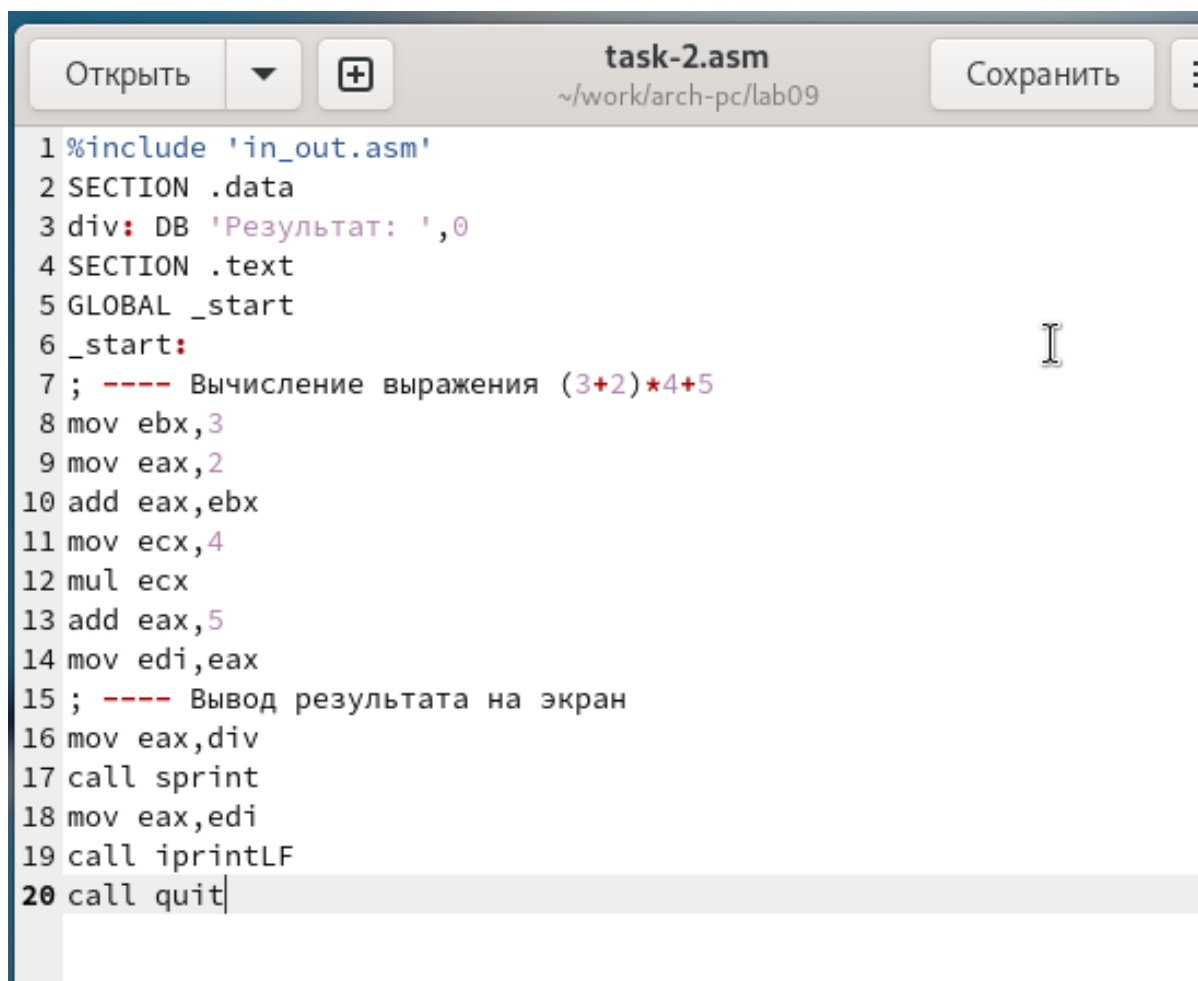
0x80490f4 <_start+12> mov    ecx,0x4
0x80490f9 <_start+17> mul    ecx
0x80490fb <_start+19> add    ebx,0x5
0x80490fe <_start+22> mov    edi,ebx
>0x8049100 <_start+24> mov    eax,0x804a000
0x8049105 <_start+29> call   0x804900f <sprint>
0x804910a <_start+34> mov    eax,edi
0x804910c <_start+36> call   0x8049086 <iprintfLF>
0x8049111 <_start+41> call   0x80490db <quit>

native process 3671 (asm) In: _start L16 PC: 0x8049100
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.

Breakpoint 1, _start () at task-2.asm:8
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
```

Рис. 2.20: Отладка task-2.asm

Я заметила, 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 title bar 'task-2.asm' and the path '~/.work/arch-pc/lab09'. The editor contains 20 lines of assembly code. Line 7 has a comment in Russian: '---- Вычисление выражения (3+2)*4+5'. Line 15 has a comment: '---- Вывод результата на экран'. The code uses standard x86 assembly instructions like 'mov', 'add', 'mul', and 'call' to perform the calculation and output the result. The file 'in_out.asm' is included at the top.

```
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
```

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

The screenshot shows a GDB window titled "artyapkova@artyapkova:~/work/arch-pc/lab09 — gdb task-2". The window is divided into three main sections. The top section, titled "Register group: general", displays the values of general-purpose registers:

Register	Value (hex)	Value (dec)
eax	0x19	25
ecx	0x4	4
edx	0x0	0
ebx	0x3	3
esp	0xffffd0f0	0xffffd0f0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0

. The middle section displays assembly code with addresses and instructions:

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

. The bottom section shows the status of the native process 3720 (asm) in the _start function, with L14 and PC: 0x80490fe. It also displays a message: "Debuginfod has been disabled. To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit." and a breakpoint: "Breakpoint 1, _start () at task-2.asm:8". The GDB prompt "(gdb)" is visible at the bottom.

Рис. 2.22: Проверка работы task-2.asm

3 Выводы

Освоили работу с подпрограммами и отладчиком.