

Отчёт по лабораторной работе 9

Понятие подпрограммы. Отладчик GDB.

Дурдыев Безирген

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение лабораторной работы	7
3.1	Реализация подпрограмм в NASM	7
3.2	Отладка программ с помощью GDB	11
3.3	Задание для самостоятельной работы	22
4	Выводы	29

Список иллюстраций

3.1	Программа в файле lab9-1.asm	8
3.2	Запуск программы lab9-1.asm	9
3.3	Программа в файле lab9-1.asm	10
3.4	Запуск программы lab9-1.asm	10
3.5	Программа в файле lab9-2.asm	11
3.6	Запуск программы lab9-2.asm в отладчике	12
3.7	Дизассимилированный код	13
3.8	Дизассимилированный код в режиме интел	14
3.9	Точка остановки	15
3.10	Изменение регистров	16
3.11	Изменение регистров	17
3.12	Изменение значения переменной	18
3.13	Вывод значения регистра	19
3.14	Вывод значения регистра	20
3.15	Программа в файле lab9-3.asm	21
3.16	Вывод значения регистра	22
3.17	Программа в файле task-1.asm	23
3.18	Запуск программы task-1.asm	24
3.19	Код с ошибкой в файле task-2.asm	25
3.20	Отладка task-2.asm	26
3.21	Код исправлен в файле task-2.asm	27
3.22	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 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа:

- обнаружение ошибки;
- поиск её местонахождения;
- определение причины ошибки;
- исправление ошибки.

GDB (GNU Debugger — отладчик проекта GNU) работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования. GDB предлагает обширные средства для слежения и контроля за выполнением компьютерных программ. Отладчик не содержит собственного графического пользовательского интерфейса и использует стандартный текстовый интерфейс консоли. Однако для GDB существует несколько сторонних графических надстроек, а кроме того, некоторые интегрированные среды разработки используют его в качестве базовой подсистемы отладки. Отладчик GDB (как и любой другой отладчик) позволяет увидеть, что происходит «внутри» программы в момент её выполнения или что делает программа в момент сбоя.

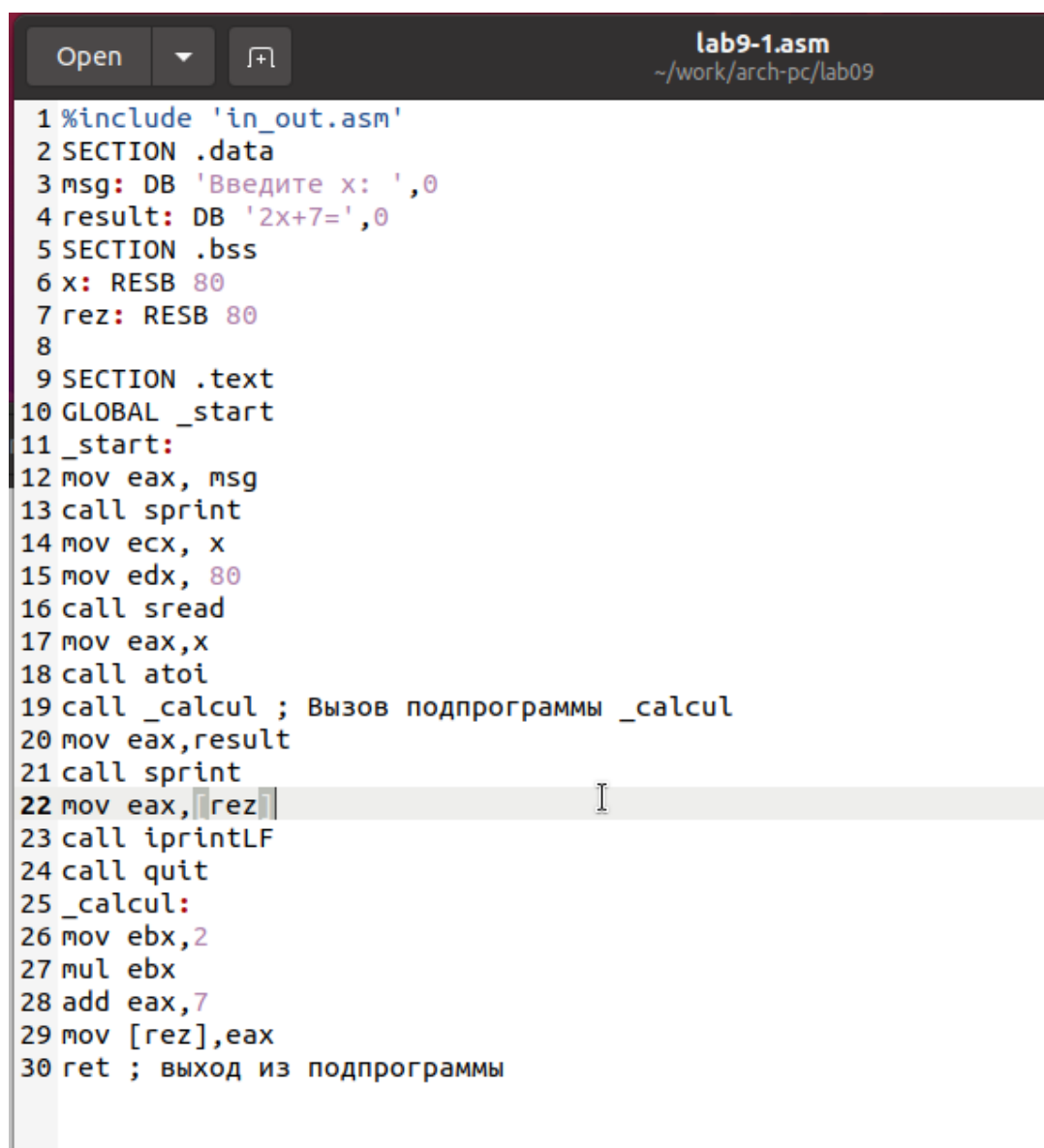
Подпрограмма — это, как правило, функционально законченный участок кода, который можно многократно вызывать из разных мест программы. В отличие от простых переходов из подпрограмм существует возврат на команду, следующую за вызовом

3 Выполнение лабораторной работы

3.1 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.



```
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
```

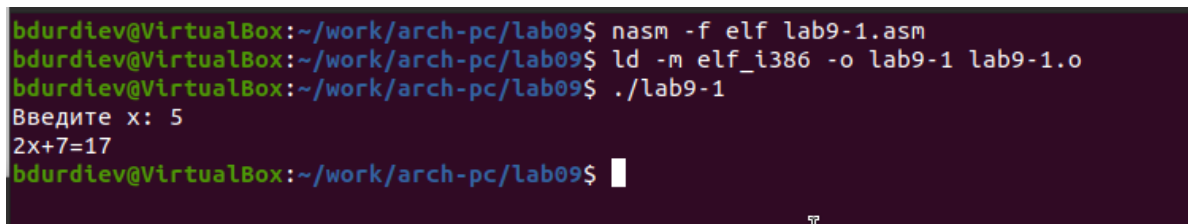
Рис. 3.1: Программа в файле lab9-1.asm

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.

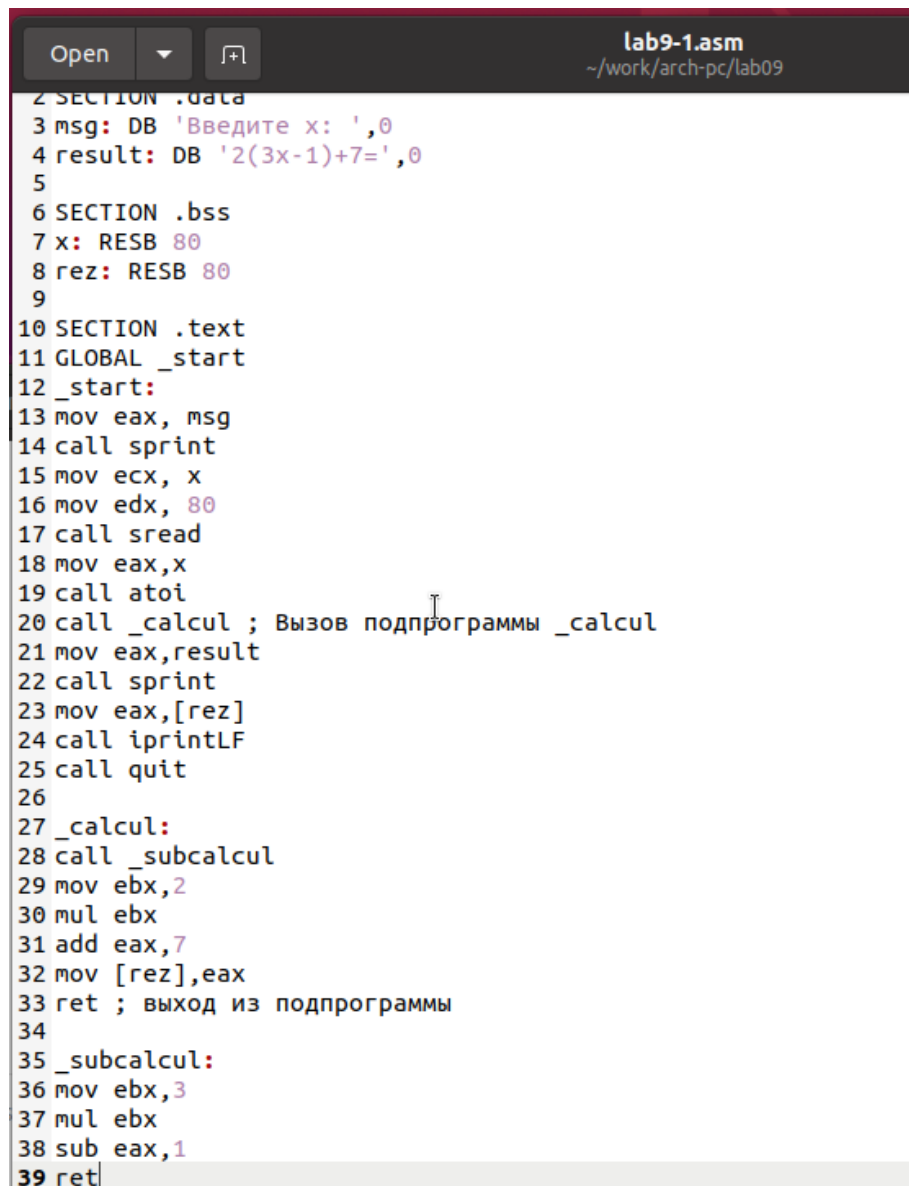
Последние строки программы реализуют вывод сообщения (с помощью вызова `sprint`), вывод результата вычисления (с помощью вызова `iprintLF`) и завершение программы (с помощью вызова `quit`).



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

Рис. 3.2: Запуск программы `lab9-1.asm`

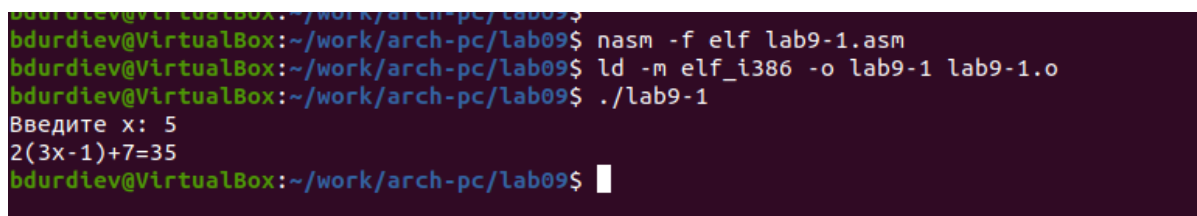
Изменил текст программы, добавив подпрограмму `subcalcul` в подпрограмму `calcul`, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$.



```
lab9-1.asm
~/work/arch-pc/lab09

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

Рис. 3.3: Программа в файле lab9-1.asm

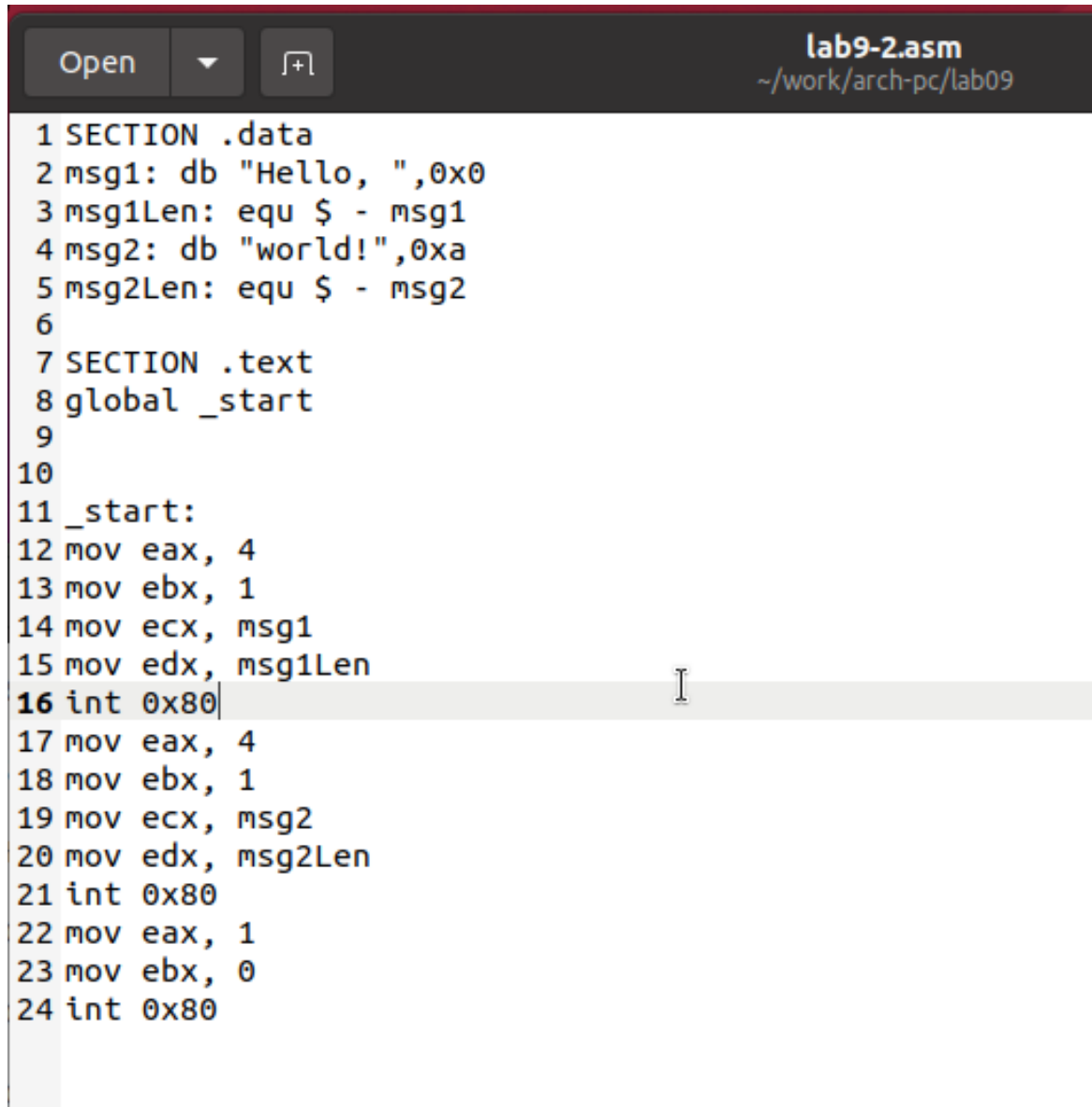


```
bdurdiev@VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
bdurdiev@VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
bdurdiev@VirtualBox:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 5
2(3x-1)+7=35
bdurdiev@VirtualBox:~/work/arch-pc/lab09$
```

Рис. 3.4: Запуск программы lab9-1.asm

3.2 Отладка программ с помощью GDB

Создал 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
11 _start:
12 mov eax, 4
13 mov ebx, 1
14 mov ecx, msg1
15 mov edx, msg1Len
16 int 0x80
17 mov eax, 4
18 mov ebx, 1
19 mov ecx, msg2
20 mov edx, msg2Len
21 int 0x80
22 mov eax, 1
23 mov ebx, 0
24 int 0x80
```

Рис. 3.5: Программа в файле lab9-2.asm

Получил исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необ-

ходимо проводить с ключом '-g'.

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

```
bdurdev@VirtualBox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
bdurdev@VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
bdurdev@VirtualBox:~/work/arch-pc/lab09$
bdurdev@VirtualBox:~/work/arch-pc/lab09$ gdb lab9-2
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 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:
<http://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) run
Starting program: /home/bdurdev/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 5854) exited normally]
(gdb) █
```

Рис. 3.6: 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 дисассимилированный код программы.

```
bdurdiev@VirtualBox: ~/work/arch-pc/lab09
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://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) run
Starting program: /home/bdurdiev/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 5854) exited normally]
(gdb) break _start
Breakpoint 1 at 0x08049000
(gdb) run
Starting program: /home/bdurdiev/work/arch-pc/lab09/lab9-2

Breakpoint 1, 0x08049000 in _start ()
(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. 3.7: Дизассимилированный код

```
bdurdiev@VirtualBox: ~/work/arch-pc/lab09
Starting program: /home/bdurdiev/work/arch-pc/lab09/lab9-2
Breakpoint 1, 0x08049000 in _start ()
(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) █
```

Рис. 3.8: 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 «звёздочка»

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.

```

bdurdiev@VirtualBox: ~/work/arch-pc/lab09

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1e0 0xffffd1e0
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 5858 In: start L?? PC: 0x8049000
(gdb)
(gdb)
(gdb) b *0x8049031Breakpoint 2 at 0x8049031
(gdb) i b
Num   Type             Disp Enb Address      What
1     breakpoint       keep y  0x08049000 <_start>
      breakpoint already hit 1 time
2     breakpoint       keep y  0x08049031 <_start+49>
(gdb)

```

Рис. 3.9: 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.

```
bdurdlev@VirtualBox: ~/work/arch-pc/lab09
Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1e0 0xffffd1e0
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 5858 In: _start L?? PC: 0x8049005
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
0x8049005 in _start ()
(gdb)
```

Рис. 3.10: Изменение регистров


```
bdurdlev@VirtualBox: ~/work/arch-pc/lab09
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 < _start+22>
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 5858 In: _start L?? PC: 0x8049016
gs      0x0      0
(gdb) si
0x08049005 in _start ()
(gdb) si
0x0804900a in _start ()
(gdb) si
0x0804900f in _start ()
(gdb) si
0x08049014 in _start ()
(gdb) si
0x08049016 in _start ()
(gdb) |
```

Рис. 3.11: 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`.

```
bdurdiev@VirtualBox: ~/work/arch-pc/lab09

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 < _start+22>
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 5858 In: _start L?? PC: 0x8049016
0x08049016 in _start ()
(gdb)
(gdb) x/1sb &msg10x804a000 <msg1>:      "Hello, "
(gdb)
(gdb) x/1sb 0x804a0080x804a008 <msg2>:    "world!\n"
(gdb)
(gdb) x/1sb &msg10x804a000 <msg1>:      "hello, "
(gdb)
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "Lor!d!\n"
(gdb)
```

Рис. 3.12: Изменение значения переменной

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

```
bdurdiev@VirtualBox: ~/work/arch-pc/lab09

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 < _start+22>
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 5858 In: _start L?? PC: 0x8049016
(gdb) p/t $eax$2 = 1000
(gdb)
(gdb) p/s $ecx$3 = 134520832
(gdb)
(gdb) p/x $ecx$4 = 0x804a000
(gdb)
(gdb) p/s $edx$5 = 8
(gdb)
(gdb) p/t $edx$6 = 1000
(gdb) p/x $edx
$7 = 0x8
(gdb)
```

Рис. 3.13: Вывод значения регистра

С помощью команды set изменил значение регистра ebx

```
bdurdiev@VirtualBox: ~/work/arch-pc/lab09

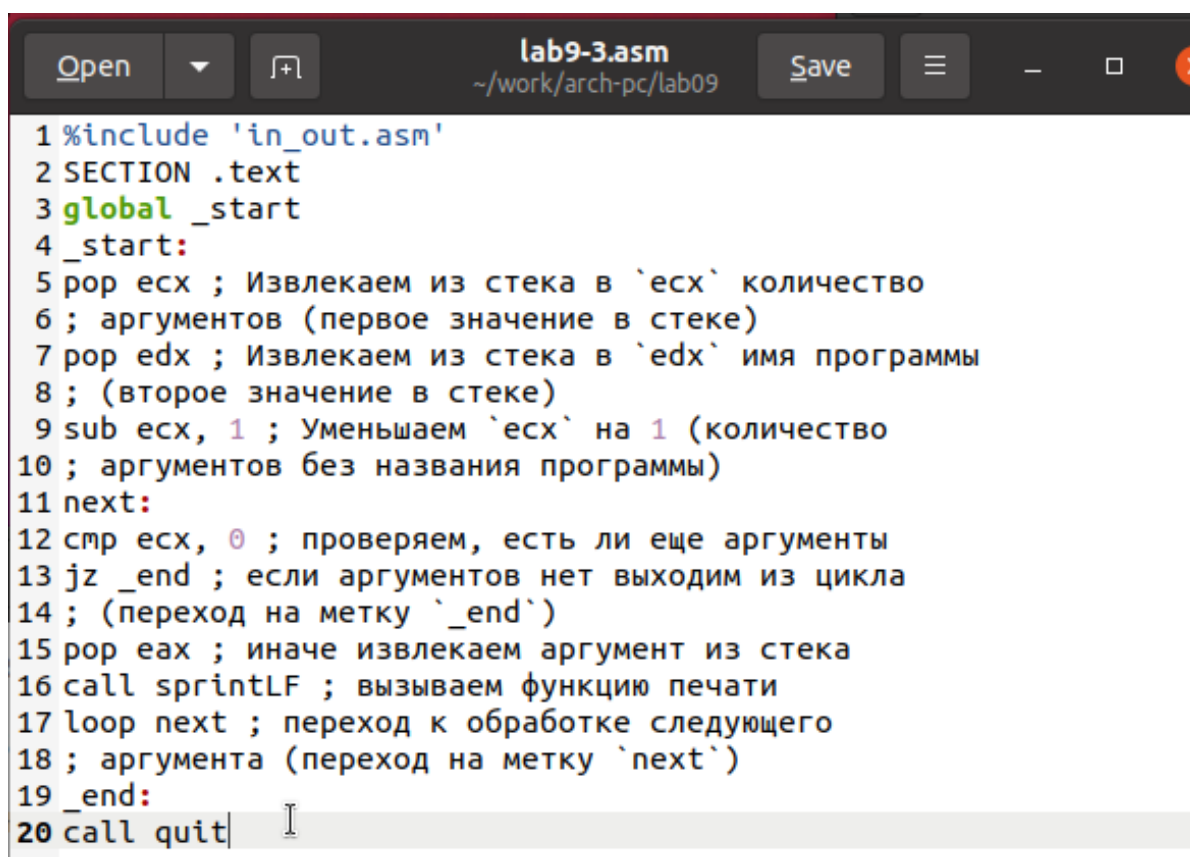
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x2      2
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]

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 5858 In: start L?? PC: 0x8049016
(gdb) p/s $edx$5 = 8
(gdb)
(gdb) p/t $edx$6 = 1000
(gdb) p/x $edx
$7 = 0x8
(gdb)
(gdb)
(gdb) p/s $ebx$8 = 50
(gdb)
(gdb) p/s $ebx
$9 = 2
(gdb) 
```

Рис. 3.14: 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.



```
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. 3.15: Программа в файле lab9-3.asm

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

Адрес вершины стека храниться 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.

```
bdurdiev@VirtualBox: ~/work/arch-pc/lab09
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) b _start
Breakpoint 1 at 0x80490e8
(gdb) run
Starting program: /home/bdurdiev/work/arch-pc/lab09/lab9-3 argument 1 argument 2 argument\ 3

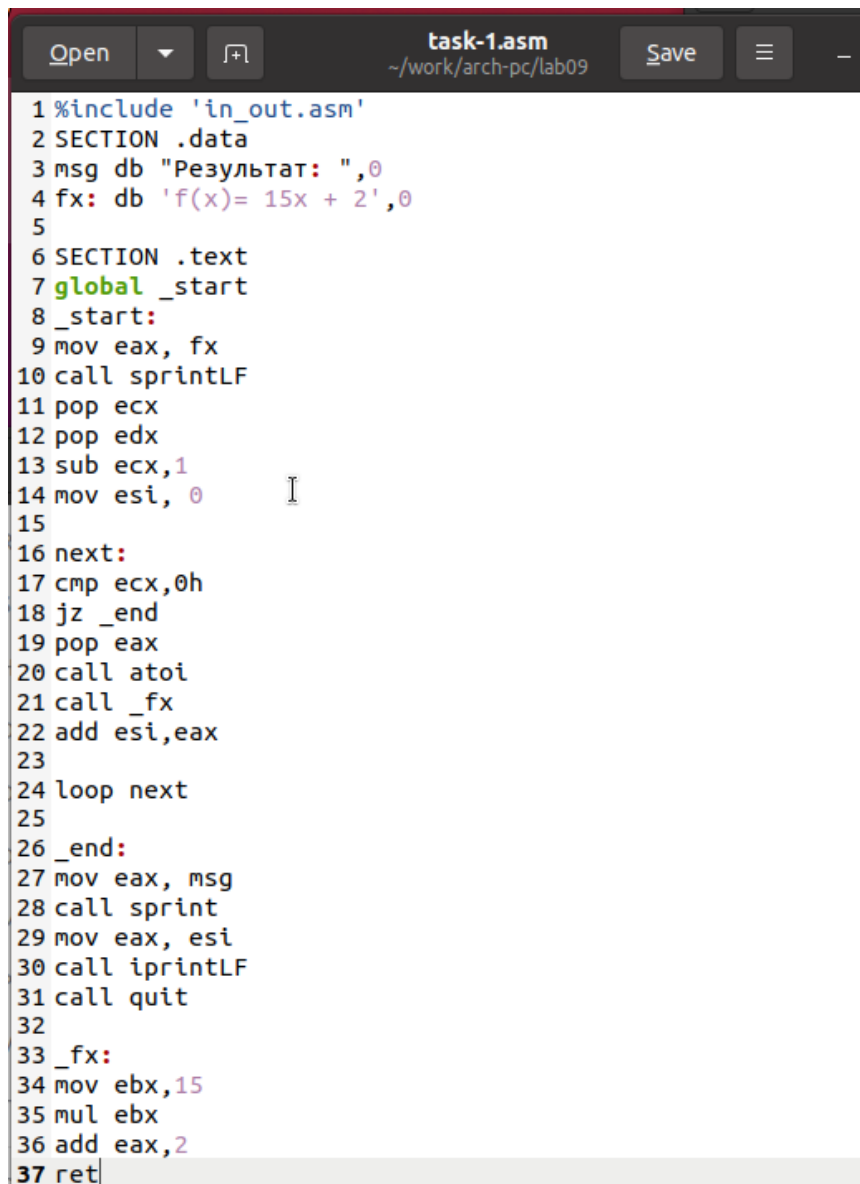
Breakpoint 1, 0x080490e8 in _start ()
(gdb) x/x $esp
0xffffd1a0:    0x00000006
(gdb)
0xffffd1a4:    0xffffd365
(gdb) x/s *(void**)(esp + 4)
0xffffd365:    "/home/bdurdiev/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd38e:    "argument"
(gdb) x/s *(void**)(esp + 12)
0xffffd397:    "1"
(gdb) x/s *(void**)(esp + 16)
0xffffd399:    "argument"
(gdb) x/s *(void**)(esp + 20)
0xffffd3a2:    "2"
(gdb) x/s *(void**)(esp + 24)
0xffffd3a4:    "argument 3"
(gdb) c
Continuing.
argument
1
argument
2
argument 3
[Inferior 1 (process 5871) exited normally]
(gdb)
```

Рис. 3.16: Вывод значения регистра

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

3.3 Задание для самостоятельной работы

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



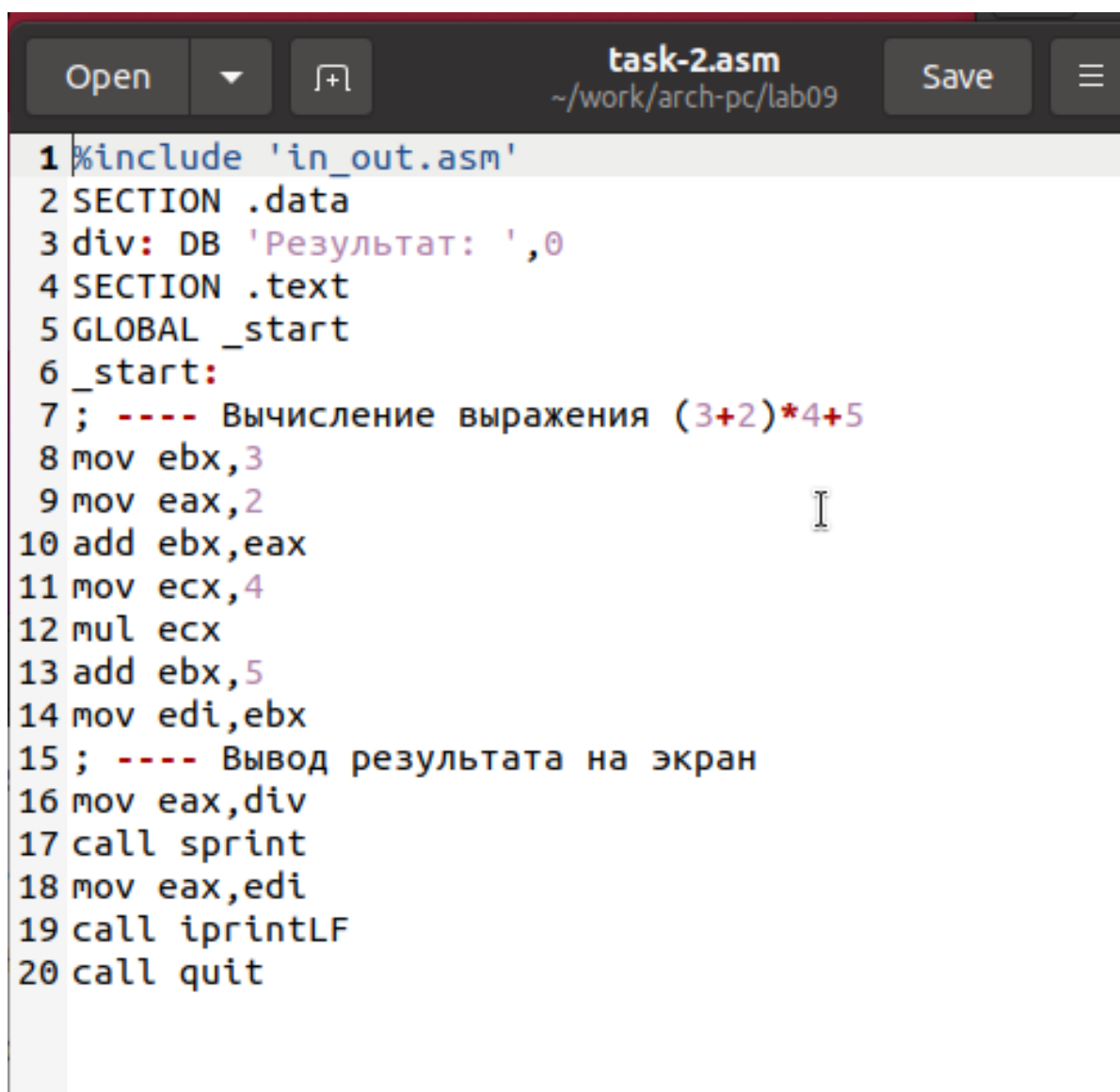
```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 fx: db 'f(x)= 15x + 2',0
5
6 SECTION .text
7 global _start
8 _start:
9 mov eax, fx
10 call sprintLF
11 pop ecx
12 pop edx
13 sub ecx,1
14 mov esi, 0
15
16 next:
17 cmp ecx,0h
18 jz _end
19 pop eax
20 call atoi
21 call _fx
22 add esi,eax
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 _fx:
34 mov ebx,15
35 mul ebx
36 add eax,2
37 ret
```

Рис. 3.17: Программа в файле task-1.asm

```
bdurdiev@VirtualBox:~/work/arch-pc/lab09$ nasm -f elf task-1.asm
bdurdiev@VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o task-1 task-1.o
bdurdiev@VirtualBox:~/work/arch-pc/lab09$ ./task-1
f(x)= 15x + 2
Результат: 0
bdurdiev@VirtualBox:~/work/arch-pc/lab09$ ./task-1 1
f(x)= 15x + 2
Результат: 17
bdurdiev@VirtualBox:~/work/arch-pc/lab09$ ./task-1 1 6 3 1 5 7
f(x)= 15x + 2
Результат: 357
bdurdiev@VirtualBox:~/work/arch-pc/lab09$
```

Рис. 3.18: 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.



```
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. 3.19: Код с ошибкой в файле task-2.asm

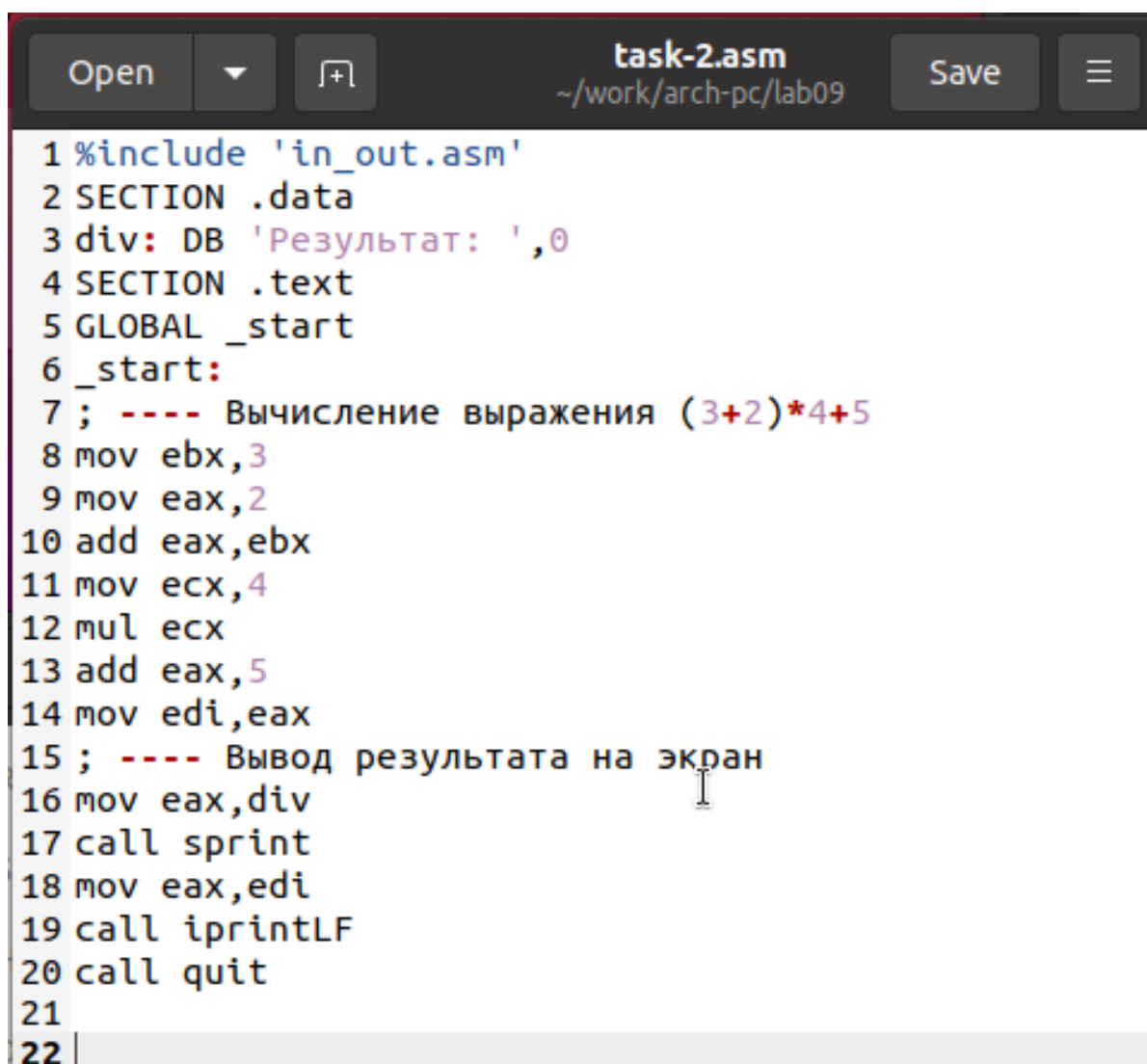
```
bdurdiev@VirtualBox: ~/work/arch-pc/lab09
eax      0x804a000      134520832
ecx      0x4           4
edx      0x0           0
ebx      0xa           10
esp      0xffffd1e0    0xffffd1e0
ebp      0x0           0
esi      0x0           0
edi      0xa           10
eip      0x8049105     0x8049105 < start+29>
eflags   0x206        [ PF IF ]

B+ 0x80490e8 <_start>      mov     ebx,0x3
B+ 0x80490e8 <_start>5>    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,0x5
0x80490fb <_start>19>     add     ebx,0x5
0x80490fe <_start>22>     mov     edi,ebx04a000
>0x8049100 <_start>24>    mov     eax,0x804a000rint>
0x8049105 <_start>29>    call    0x804900f <sprint>
0x804910a <_start>34>    mov     eax,edi86 <iprintLF>
0x804910c <_start>36>    call    0x8049086 <iprintLF>

native process 5902 In: _start      L??   PC: 0x8049105
(gdb) sNo process In:              L??   PC: ??
(gdb) si0x080490fb in _start ()
(gdb) si
0x080490fe in _start ()
(gdb) si
0x08049100 in _start ()
(gdb) si
0x08049105 in _start ()
(gdb) c
Continuing.
Результат: 10
[Inferior 1 (process 5902) exited normally]
(gdb) █
```

Рис. 3.20: 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:



```
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
22
```

Рис. 3.21: Код исправлен в файле task-2.asm

```
bdurdliev@VirtualBox: ~/work/arch-pc/lab09
eax      0x19      25
ecx      0x4       4
edx      0x0       0
ebx      0x3       3
esp      0xffffd1e0 0xffffd1e0
ebp      0x0       0x0
esi      0x0       0
edi      0x0       0
eip      0x80490fe 0x80490fe <_start+22>
eflags   0x202     [ IF ]

B+ 0x80490e8 <_start>      mov     ebx,0x3
B+ 0x80490e8 <_start>+5>   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,0x5
>0x80490fb <_start>+19>    add     eax,0x5
0x80490fe <_start>+22>    mov     edi,eax04a000
0x8049100 <_start>+24>    mov     eax,0x804a000rint>
0x8049105 <_start>+29>    call   0x804900f <sprint>
0x804910a <_start>+34>    mov     eax,edi86 <iprintLF>
0x804910c <_start>+36>    call   0x8049086 <iprintLF>

native process 5913 In: _start L?? PC: 0x80490fe
(gdb) sNo process In: L?? PC: ??
(gdb) si0x080490f4 in _start ()
(gdb)
(gdb) si0x080490f9 in _start ()
(gdb)
(gdb) si0x080490fb in _start ()
(gdb) si
0x080490fe in _start ()
(gdb) c
Continuing.
Результат: 25
[Inferior 1 (process 5913) exited normally]
(gdb)
```

Рис. 3.22: Проверка работы task-2.asm

4 Выводы

Освоили работу с подпрограммами и отладчиком.