

ОтчеТт по лабораторной работе №9

дисциплина: Архитектура компьютера

Дельгадильо Валерия

Содержание

1	Цель работы	6
2	Теоретическое введение	7
2.1	Понятие об отладке	7
2.2	Методы отладки	8
3	Лабораторной работы	10
3.1	Реализация подпрограмм в NASM	10
3.2	Отладка программ с помощью GDB	13
3.3	Добавление точек останова	17
3.4	Работа с данными программы в GDB	19
4	Задание для самостоятельной работы	25
5	Выводы	31
6	Список литературы	32

Список иллюстраций

3.1	10
3.2	11
3.3	11
3.4	12
3.5	12
3.6	13
3.7	13
3.8	14
3.9	14
3.10	15
3.11	15
3.12	16
3.13	17
3.14	18
3.15	18
3.16	18
3.17	19
3.18	19
3.19	20
3.20	21
3.21	21
3.22	21
3.23	22
3.24	22
3.25	22
3.26	23
3.27	23
3.28	24
3.29	24
4.1	26
4.2	26
4.3	27
4.4	27
4.5	28
4.6	28
4.7	28

4.8	29
4.9	29
4.10	30

Список таблиц

1 Цель работы

Приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

2 Теоретическое введение

2.1 Понятие об отладке

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.

Можно выделить следующие типы ошибок:

- синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка;
- семантические ошибки — являются логическими и приводят к тому, что программа запускается, отработывает, но не даёт желаемого результата;
- ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают прерывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль).

Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга.

Третий этап — выяснение причины ошибки. После определения местонахождения ошибки обычно проще определить причину неправильной работы про-

граммы.

Последний этап — исправление ошибки. После этого при повторном запуске программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново.

2.2 Методы отладки

Наиболее часто применяют следующие методы отладки:

- создание точек контроля значений на входе и выходе участка программы (например, вывод промежуточных значений на экран — так называемые диагностические сообщения);
- использование специальных программ-отладчиков.

Отладчики позволяют управлять ходом выполнения программы, контролировать и изменять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправление. Наиболее популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам.

Пошаговое выполнение — это выполнение программы с остановкой после каждой строки, чтобы программист мог проверить значения переменных и выполнить другие действия.

Точки останова — это специально отмеченные места в программе, в которых программа-отладчик приостанавливает выполнение программы и ждёт команд. Наиболее популярные виды точек останова:

- Breakpoint — точка останова (остановка происходит, когда выполнение доходит до определённой строки, адреса или процедуры, отмеченной программистом);
- Watchpoint — точка просмотра (выполнение программы приостанавливается, если программа обратилась к определённой переменной: либо считала её значение, либо изменила его).

Точки останова устанавливаются в отладчике на время сеанса работы с кодом

программы, т.е. они сохраняются до выхода из программы-отладчика или до смены отлаживаемой программы.

3 Лабораторной работы

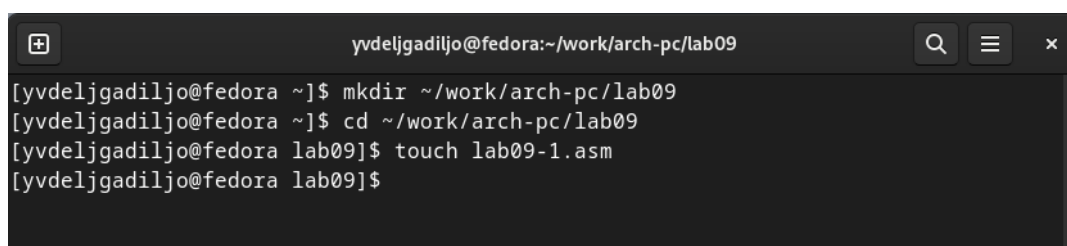
3.1 Реализация подпрограмм в NASM

Создайте каталог для выполнения лабораторной работы № 9, перейдите в него и создайте файл lab09-1.asm:

```
mkdir ~/work/arch-pc/lab09
```

```
cd ~/work/arch-pc/lab09
```

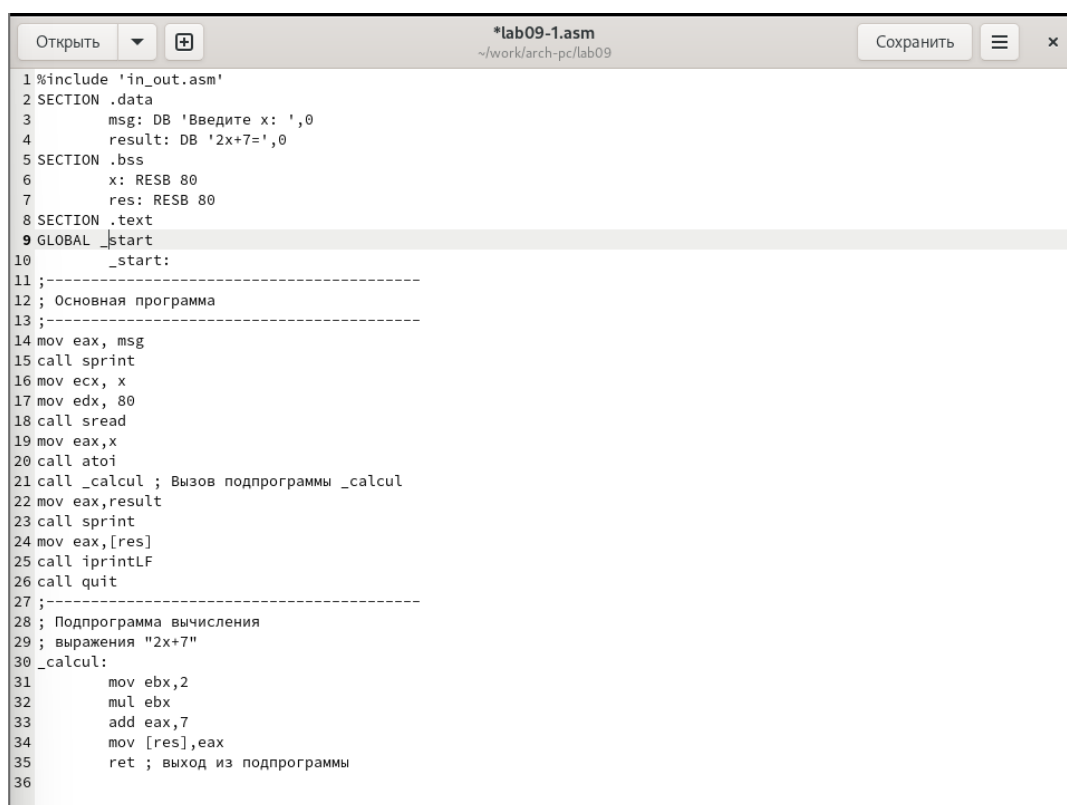
```
touch lab09-1.asm
```

A screenshot of a terminal window with a dark background. The title bar shows the user 'yvdeljgatiljo' on a 'fedora' machine, in the directory '~/work/arch-pc/lab09'. The terminal contains four lines of text: the first line shows the command 'mkdir ~/work/arch-pc/lab09' being executed; the second line shows 'cd ~/work/arch-pc/lab09'; the third line shows 'touch lab09-1.asm'; and the fourth line shows the prompt after the file has been created.

```
yvdeljgatiljo@fedora:~/work/arch-pc/lab09
[yvdeljgatiljo@fedora ~]$ mkdir ~/work/arch-pc/lab09
[yvdeljgatiljo@fedora ~]$ cd ~/work/arch-pc/lab09
[yvdeljgatiljo@fedora lab09]$ touch lab09-1.asm
[yvdeljgatiljo@fedora lab09]$
```

Рис. 3.1:

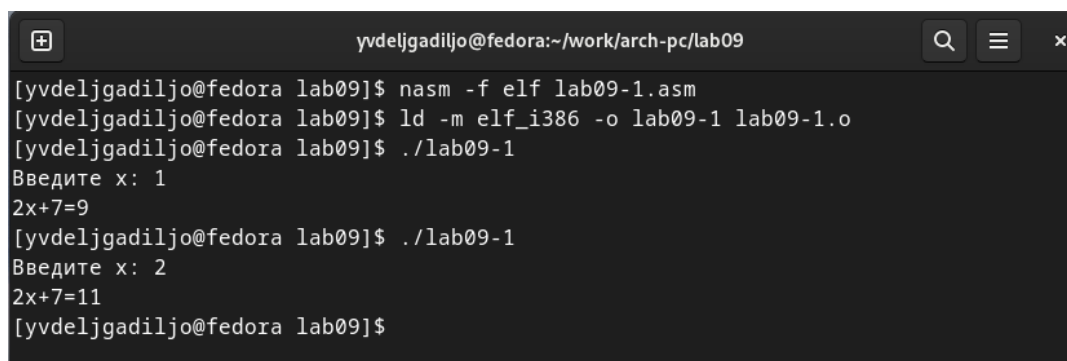
В качестве примера рассмотрим программу вычисления арифметического выражения $f(x) = 2x + 7$ с помощью подпрограммы `_calcul`. В данном примере x вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Внимательно изучите текст программы (Листинг 9.1)



```
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     res: RESB 80
8 SECTION .text
9 GLOBAL _start
10 _start:
11 ;-----
12 ; Основная программа
13 ;-----
14 mov eax, msg
15 call sprint
16 mov ecx, x
17 mov edx, 80
18 call sread
19 mov eax, x
20 call atoi
21 call _calcul ; Вызов подпрограммы _calcul
22 mov eax, result
23 call sprint
24 mov eax, [res]
25 call iprintLF
26 call quit
27 ;-----
28 ; Подпрограмма вычисления
29 ; выражения "2x+7"
30 _calcul:
31     mov ebx, 2
32     mul ebx
33     add eax, 7
34     mov [res], eax
35     ret ; выход из подпрограммы
36
```

Рис. 3.2:

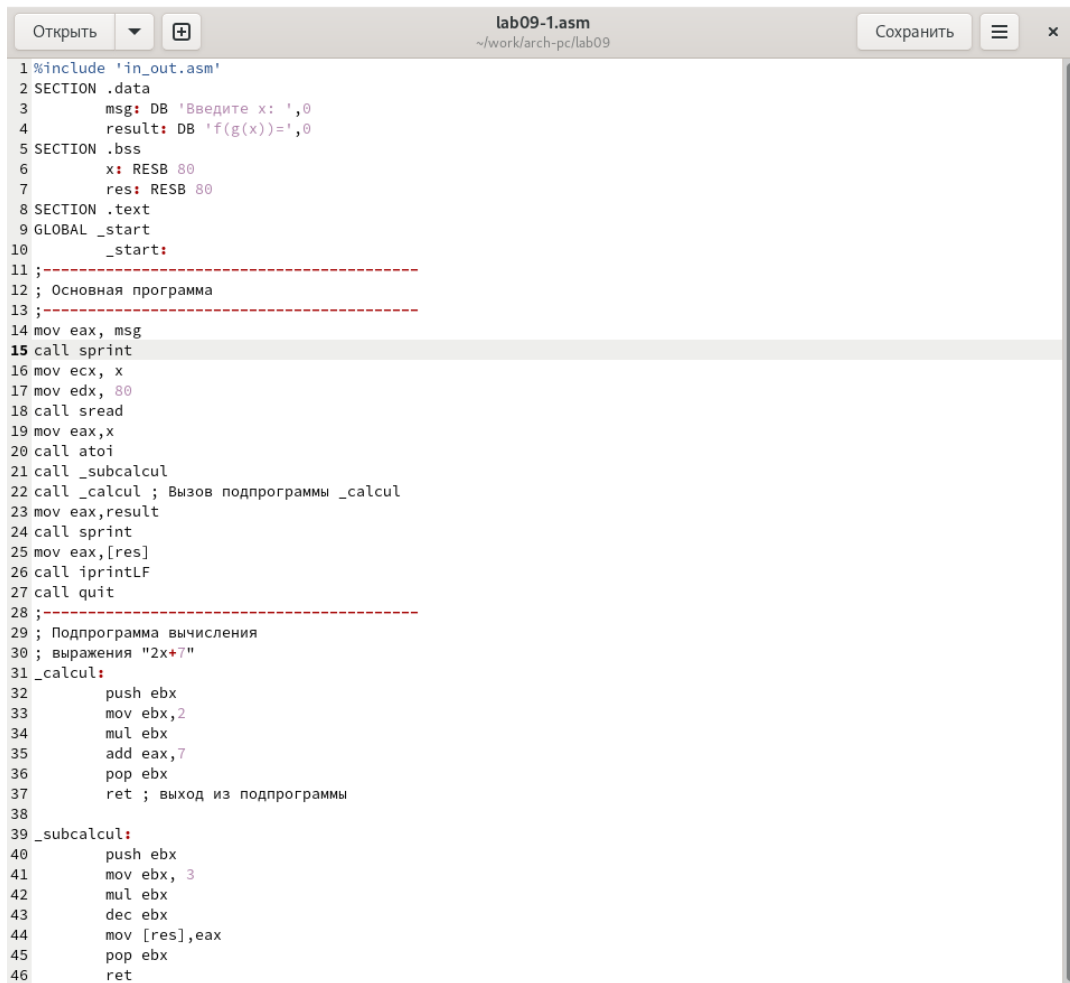
Создайте исполняемый файл и проверьте его работу.



```
[yvdeljgatiljo@fedora lab09]$ nasm -f elf lab09-1.asm
[yvdeljgatiljo@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[yvdeljgatiljo@fedora lab09]$ ./lab09-1
Введите x: 1
2x+7=9
[yvdeljgatiljo@fedora lab09]$ ./lab09-1
Введите x: 2
2x+7=11
[yvdeljgatiljo@fedora lab09]$
```

Рис. 3.3:

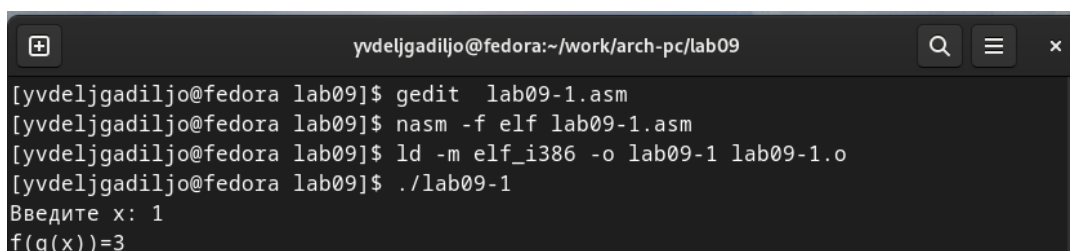
Измените текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$.



```
1 %include 'in_out.asm'
2 SECTION .data
3     msg: DB 'Введите x: ',0
4     result: DB 'f(g(x))=',0
5 SECTION .bss
6     x: RESB 80
7     res: RESB 80
8 SECTION .text
9 GLOBAL _start
10 _start:
11 ;-----
12 ; Основная программа
13 ;-----
14 mov eax, msg
15 call sprint
16 mov ecx, x
17 mov edx, 80
18 call sread
19 mov eax, x
20 call atoi
21 call _subcalcul
22 call _calcul ; Вызов подпрограммы _calcul
23 mov eax, result
24 call sprint
25 mov eax, [res]
26 call iprintfLF
27 call quit
28 ;-----
29 ; Подпрограмма вычисления
30 ; выражения "2x+7"
31 _calcul:
32     push ebx
33     mov ebx, 2
34     mul ebx
35     add eax, 7
36     pop ebx
37     ret ; выход из подпрограммы
38
39 _subcalcul:
40     push ebx
41     mov ebx, 3
42     mul ebx
43     dec ebx
44     mov [res], eax
45     pop ebx
46     ret
```

Рис. 3.4:

Создайте исполняемый файл и проверьте его работу.

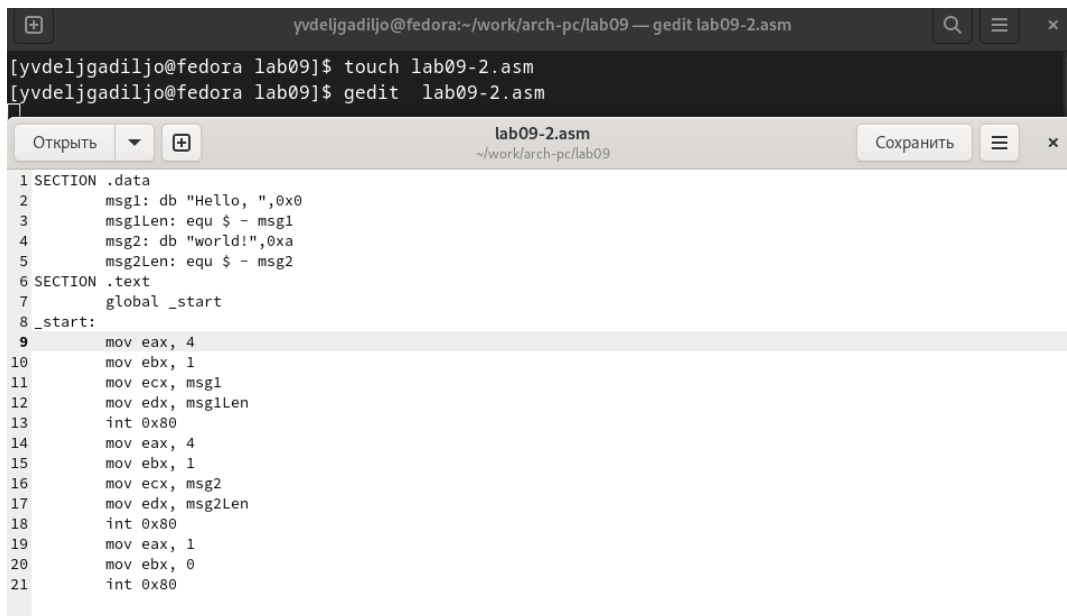


```
yvdeljgadijo@fedora:~/work/arch-pc/lab09
[yvdeljgadijo@fedora lab09]$ gedit lab09-1.asm
[yvdeljgadijo@fedora lab09]$ nasm -f elf lab09-1.asm
[yvdeljgadijo@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[yvdeljgadijo@fedora lab09]$ ./lab09-1
Введите x: 1
f(g(x))=3
```

Рис. 3.5:

3.2 Отладка программ с помощью GDB

Создайте файл lab09-2.asm с текстом программы из Листинга 9.2. (Программа печати сообщения Hello world!):



```
[yvdeljgatiljo@fedora lab09]$ touch lab09-2.asm
[yvdeljgatiljo@fedora lab09]$ gedit lab09-2.asm

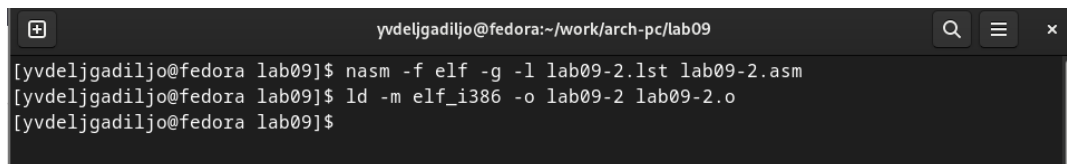
1 SECTION .data
2     msg1: db "Hello, ",0x0
3     msg1Len: equ $ - msg1
4     msg2: db "world!",0xa
5     msg2Len: equ $ - msg2
6 SECTION .text
7     global _start
8 _start:
9     mov eax, 4
10    mov ebx, 1
11    mov ecx, msg1
12    mov edx, msg1Len
13    int 0x80
14    mov eax, 4
15    mov ebx, 1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

Рис. 3.6:

Получите исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’.

```
nasm -f elf -g -l lab09-2.lst lab09-2.asm
```

```
ld -m elf_i386 -o lab09-2 lab09-2.o
```

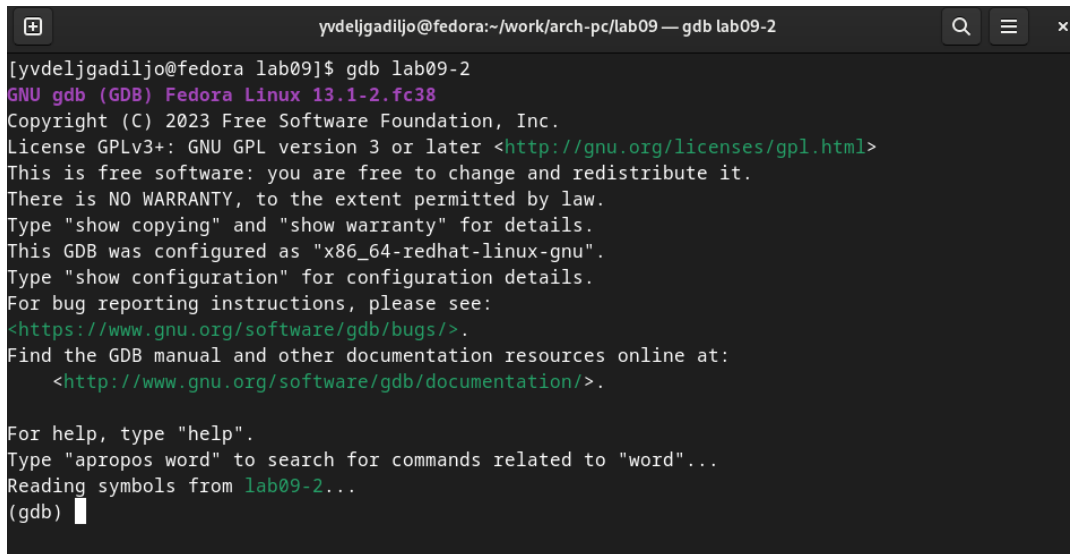


```
[yvdeljgatiljo@fedora lab09]$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
[yvdeljgatiljo@fedora lab09]$ ld -m elf_i386 -o lab09-2 lab09-2.o
[yvdeljgatiljo@fedora lab09]$
```

Рис. 3.7:

Загрузите исполняемый файл в отладчик gdb:

user@dk4n31:~\$ gdb lab09-2



```
[yvdeljgatiljo@fedora lab09]$ gdb lab09-2
GNU gdb (GDB) Fedora Linux 13.1-2.fc38
Copyright (C) 2023 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 lab09-2...
(gdb)
```

Рис. 3.8:

Проверьте работу программы, запустив ее в оболочке GDB с помощью команды `run` (сокращённо `r`):

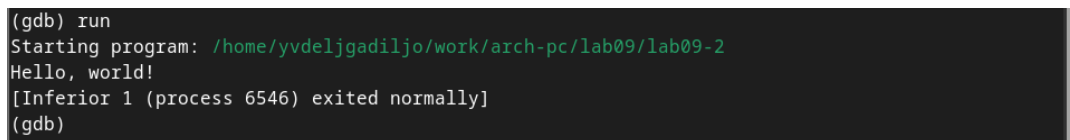
(gdb) run

Starting program: ~/work/arch-pc/lab09/lab09-2

Hello, world!

[Inferior 1 (process 10220) exited normally]

(gdb)



```
(gdb) run
Starting program: /home/yvdeljgatiljo/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 6546) exited normally]
(gdb)
```

Рис. 3.9:

Для более подробного анализа программы установите брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустите её.

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

Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
(gdb)
```

Рис. 3.10:

Посмотрите дисассимилированный код программы с помощью команды `disassemble`

начиная с метки `_start`

`(gdb) disassemble _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. 3.11:

Переключитесь на отображение команд с Intel'овским синтаксисом, введя команду `set`

`disassembly-flavor intel`

`(gdb) set disassembly-flavor intel`

`(gdb) disassemble _start`

```

(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. 3.12:

Перечислите различия отображения синтаксиса машинных команд в режимах АТТ и Intel.

Включите режим псевдографики для более удобного анализа программы:

(gdb) layout asm

(gdb) layout regs

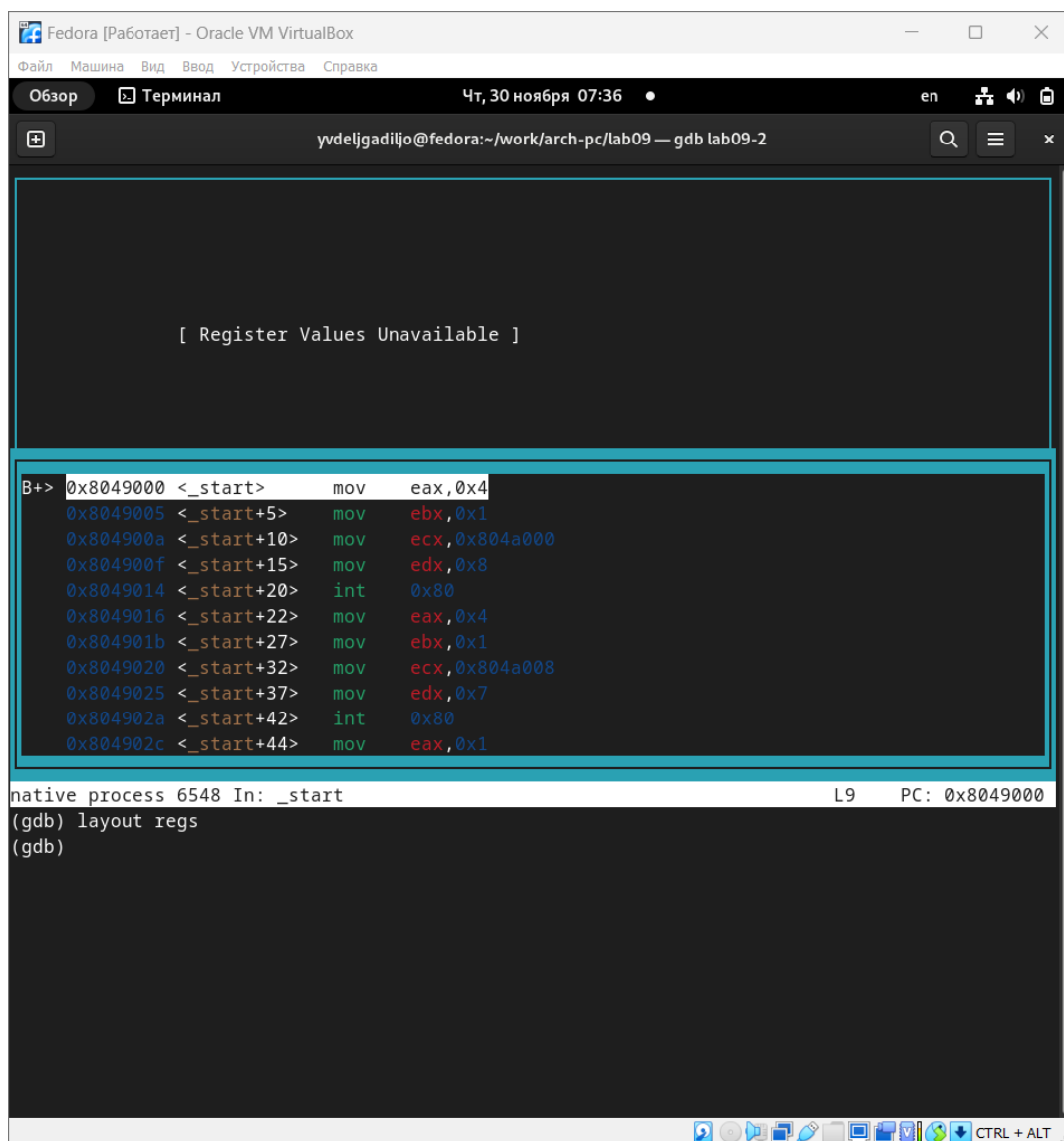


Рис. 3.13:

3.3 Добавление точек останова

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 останова по имени метки (_start). Проверьте это с помощью команды `info breakpoints` (кратко `i b`): `(gdb) info breakpoints`

```
native process 6548 In: _start L9 PC: 0x8049000
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y 0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
```

Рис. 3.14:

Установим еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции.

Определите адрес предпоследней инструкции (`mov ebx,0x0`) и установите точку останова.

`(gdb) break *<адрес>`

```
b+ 0x8049031 <_start+49> mov ebx,0x0
native process 6548 In: _start L9 PC: 0x8049000
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y 0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
```

Рис. 3.15:

Посмотрите информацию о всех установленных точках останова:

`(gdb) i b`

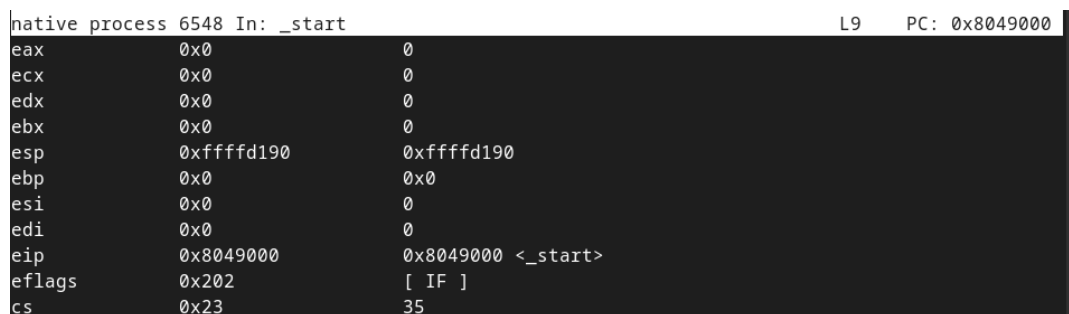
```
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y 0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
2        breakpoint      keep y 0x08049031 lab09-2.asm:20
```

Рис. 3.16:

3.4 Работа с данными программы в GDB

Посмотреть содержимое регистров также можно с помощью команды `info registers` (или `i r`).

(gdb) info registers



The screenshot shows the output of the 'info registers' command in GDB. It lists various registers and their values. The registers are: eax, ecx, edx, ebx, esp, ebp, esi, edi, eip, eflags, and cs. The values are displayed in two columns. The PC (Program Counter) is shown as 0x8049000.

Register	Value	Comment
eax	0x0	
ecx	0x0	
edx	0x0	
ebx	0x0	
esp	0xffffd190	0xffffd190
ebp	0x0	0x0
esi	0x0	
edi	0x0	
eip	0x8049000	0x8049000 <_start>
eflags	0x202	[IF]
cs	0x23	35

Рис. 3.17:

Для отображения содержимого памяти можно использовать команду `x <адрес>`, которая выдаёт содержимое ячейки памяти по указанному адресу. Формат, в котором выводятся данные, можно задать после имени команды через косую черту: `x/NFU <адрес>`.

С помощью команды `x &<имя переменной>` также можно посмотреть содержимое переменной.

Посмотрите значение переменной `msg1` по имени

(gdb) x/1sb &msg1

0x804a000 <msg1>: "Hello,"



The screenshot shows the output of the 'x/1sb &msg1' command in GDB. It displays the memory address 0x804a000 and the value 'Hello, '.

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

Рис. 3.18:

Посмотрите значение переменной `msg2` по адресу. Адрес переменной можно определить по дизассемблированной инструкции. Посмотрите инструкцию `mov ecx,msg2` которая записывает в регистр `ecx` адрес переменной `msg2`.

```

Register group: general
eax      0x4          4
ecx      0x804a000    134520832
edx      0x0          0
ebx      0x1          1
esp      0xffffd190   0xffffd190
ebp      0x0          0x0
esi      0x0          0
edi      0x0          0
eip      0x804900f    0x804900f <_start+15>
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 6548 In: _start L12 PC: 0x804900f
eip      0x8049000    0x8049000 <_start>
eflags   0x202        [ IF ]
cs       0x23         35
--Type <RET> for more, q to quit, c to continue without paging--qQuit
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) si
(gdb) si
(gdb) si
(gdb) x/1sb 0x804a000
0x804a000 <msg1>: "Hello, "
(gdb)

```

Рис. 3.19:

Изменить значение для регистра или ячейки памяти можно с помощью команды `set`,

задав ей в качестве аргумента имя регистра или адрес. При этом перед именем регистра ставится префикс `$`, а перед адресом нужно указать в фигурных скобках тип данных (размер сохраняемого значения; в качестве типа данных можно использовать типы языка Си).

Измените первый символ переменной `msg1`:

```
(gdb) set {char}msg1='h'
```

```
(gdb) x/1sb &msg1
```

```
0x804a000 <msg1>: "hello, "
```

(gdb)

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

Рис. 3.20:

Замените любой символ во второй переменной msg2.

```
(gdb) set {char}&msg2=9
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "\torld!\n\034"
(gdb)
```

Рис. 3.21:

Чтобы посмотреть значения регистров используется команда `print /F <val>` (перед именем регистра обязательно ставится префикс \$):

`p/F`

`$<регистр>`

```
native process 6548 In: _start L12 PC: 0x804900f
(gdb) set $ebx=2
(gdb) p/s $ebx
$2 = 2
(gdb) p/s $eax
$3 = 4
(gdb) p/t $eax
$4 = 100
(gdb) p/s $ecx
$5 = 134520832
(gdb) p/x $ecx
$6 = 0x804a000
(gdb)
```

Рис. 3.22:

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

С помощью команды `set` измените значение регистра `ebx`:

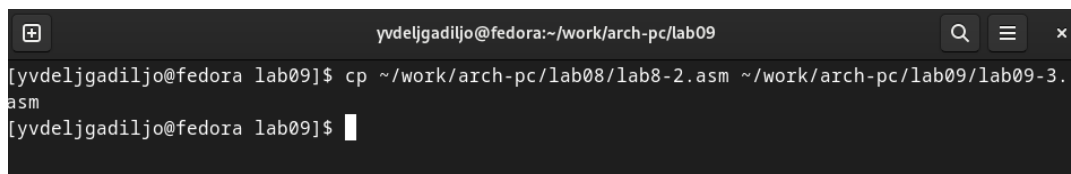
```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$2 = 2
(gdb)
```

Рис. 3.23:

Завершим работу в gdb командами continue, она закончит выполнение программы, и exit, она завершит сеанс gdb.

1.4.3. Обработка аргументов командной строки в GDBСкопируйте файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm:

```
cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
```



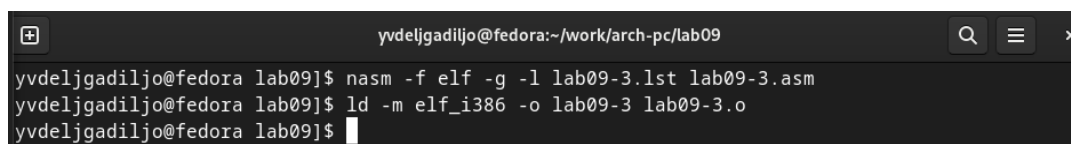
```
yvdeljgatiljo@fedora:~/work/arch-pc/lab09
[yvdeljgatiljo@fedora lab09]$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
[yvdeljgatiljo@fedora lab09]$
```

Рис. 3.24:

Создайте исполняемый файл.

```
nasm -f elf -g -l lab09-3.lst lab09-3.asm
```

```
ld -m elf_i386 -o lab09-3 lab09-3.o
```



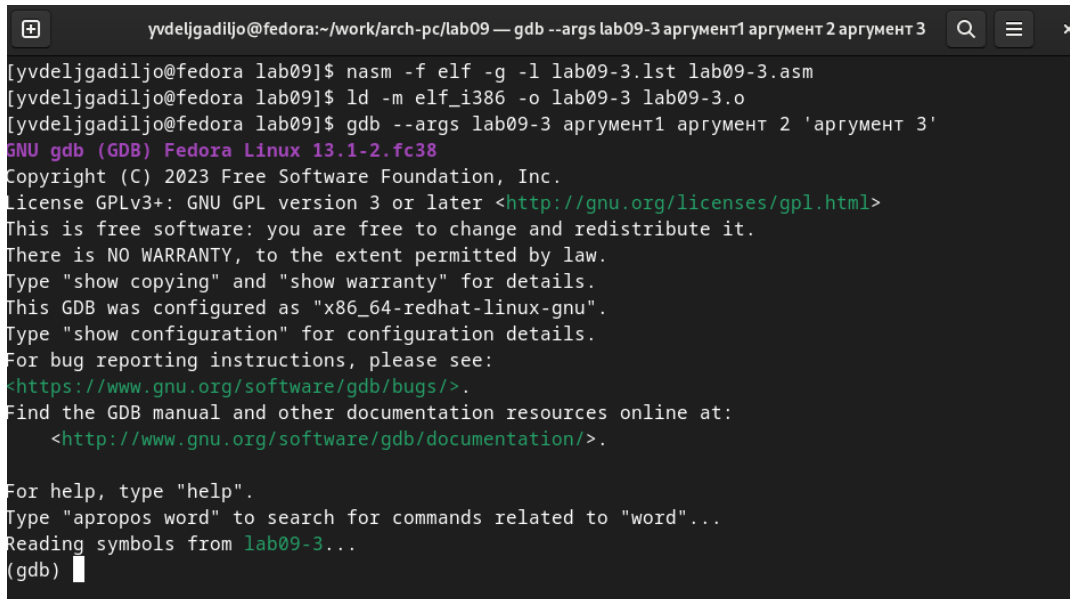
```
yvdeljgatiljo@fedora:~/work/arch-pc/lab09
[yvdeljgatiljo@fedora lab09]$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
[yvdeljgatiljo@fedora lab09]$ ld -m elf_i386 -o lab09-3 lab09-3.o
[yvdeljgatiljo@fedora lab09]$
```

Рис. 3.25:

Для загрузки в gdb программы с аргументами необходимо использовать ключ --args.

Загрузите исполняемый файл в отладчик, указав аргументы:

`gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'`



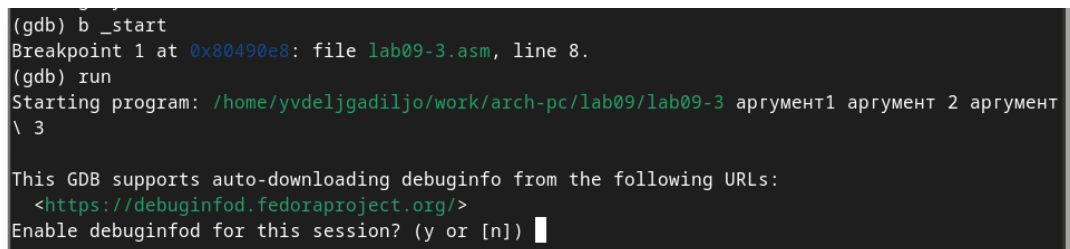
```
yvdeljgatiljo@fedora:~/work/arch-pc/lab09 — gdb --args lab09-3 аргумент1 аргумент2 аргумент3
[yvdeljgatiljo@fedora lab09]$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
[yvdeljgatiljo@fedora lab09]$ ld -m elf_i386 -o lab09-3 lab09-3.o
[yvdeljgatiljo@fedora lab09]$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (GDB) Fedora Linux 13.1-2.fc38
Copyright (C) 2023 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 lab09-3...
(gdb) █
```

Рис. 3.26:

Для начала установим точку останова перед первой инструкцией в программе и запустим ее.

(gdb) `b _start`

(gdb) `run`



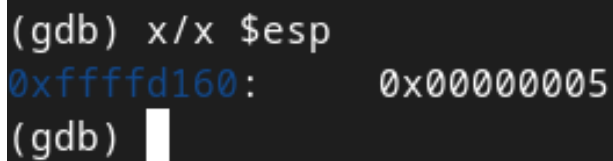
```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 8.
(gdb) run
Starting program: /home/yvdeljgatiljo/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент
\ 3

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) █
```

Рис. 3.27:

Адрес вершины стека храниться в регистре `esp` и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы):

```
(gdb) x/x $esp
0xffffd200: 0x05
```

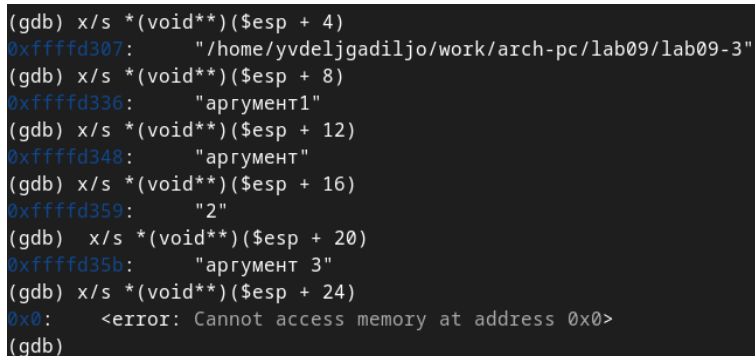


```
(gdb) x/x $esp
0xffffd160: 0x00000005
(gdb) █
```

Рис. 3.28:

Как видно, число аргументов равно 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и 'аргумент 3'.

Посмотрите 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.



```
(gdb) x/s *(void**)(esp + 4)
0xffffd307: "/home/yvdeljgadiljo/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd336: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd348: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd359: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd35b: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 3.29:

Объясните, почему шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12] и т.д.).

Их адреса располагаются в 4 байтах друг от друга (именно столько занимает элемент стека).

4 Задание для самостоятельной работы

Преобразуйте программу из лабораторной работы N8 (Задание N1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму.

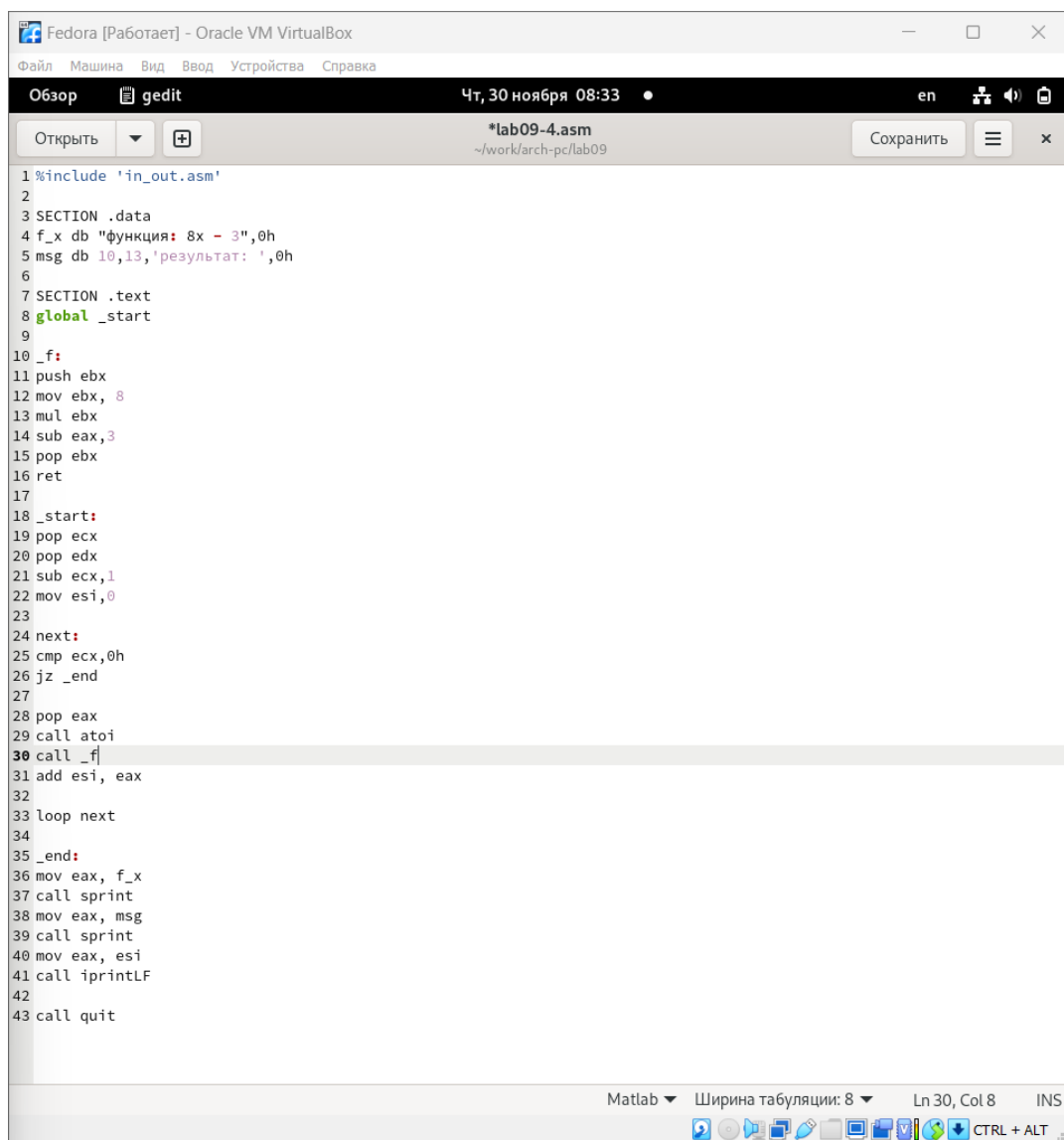


Рис. 4.1:

И проверка ее работоспособности

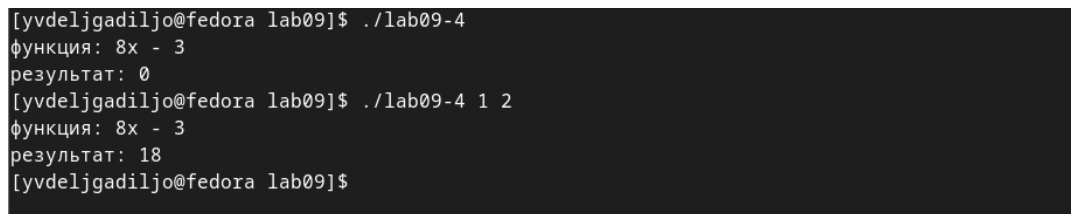


Рис. 4.2:

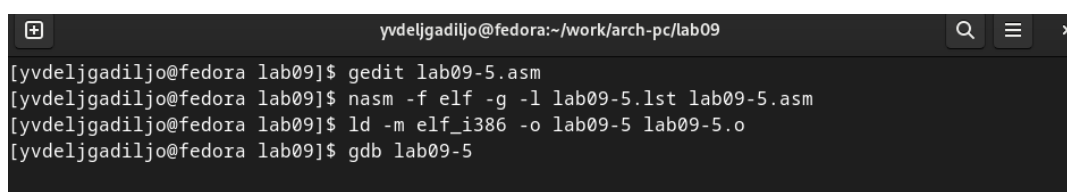
В листинге 9.3 приведена программа вычисления выражения $(3 + 2) \times 4 + 5$. При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее.



```
Открыть ▾ + lab09-5.asm
~/work/arch-pc/lab09

%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
```

Рис. 4.3:



```
yvdeljgatiljo@fedora:~/work/arch-pc/lab09

[yvdeljgatiljo@fedora lab09]$ gedit lab09-5.asm
[yvdeljgatiljo@fedora lab09]$ nasm -f elf -g -l lab09-5.lst lab09-5.asm
[yvdeljgatiljo@fedora lab09]$ ld -m elf_i386 -o lab09-5 lab09-5.o
[yvdeljgatiljo@fedora lab09]$ gdb lab09-5
```

Рис. 4.4:

```
(gdb) break _start
Breakpoint 1 at 0x80490e8: file lab09-5.asm, line 11.
(gdb) run
Starting program: /home/yvdeljgadiljo/work/arch-pc/lab09/lab09-5

Breakpoint 1, _start () at lab09-5.asm:11
warning: Source file is more recent than executable.
11      mov ecx,4
```

Рис. 4.5:

Просмотр регистров, для поиска ошибки в программе из листинга 10.3

```
Register group: general
eax      0x2      2
ecx      0x0      0
edx      0x0      0
ebx      0x5      5
esp      0xffffd0b0 0xffffd0b0
ebp      0x0      0x0

self10-1.asm
  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
```

Рис. 4.6:

```
Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa     10
esp      0xffffd0b0 0xffffd0b0
ebp      0x0      0x0

self10-1.asm
 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
```

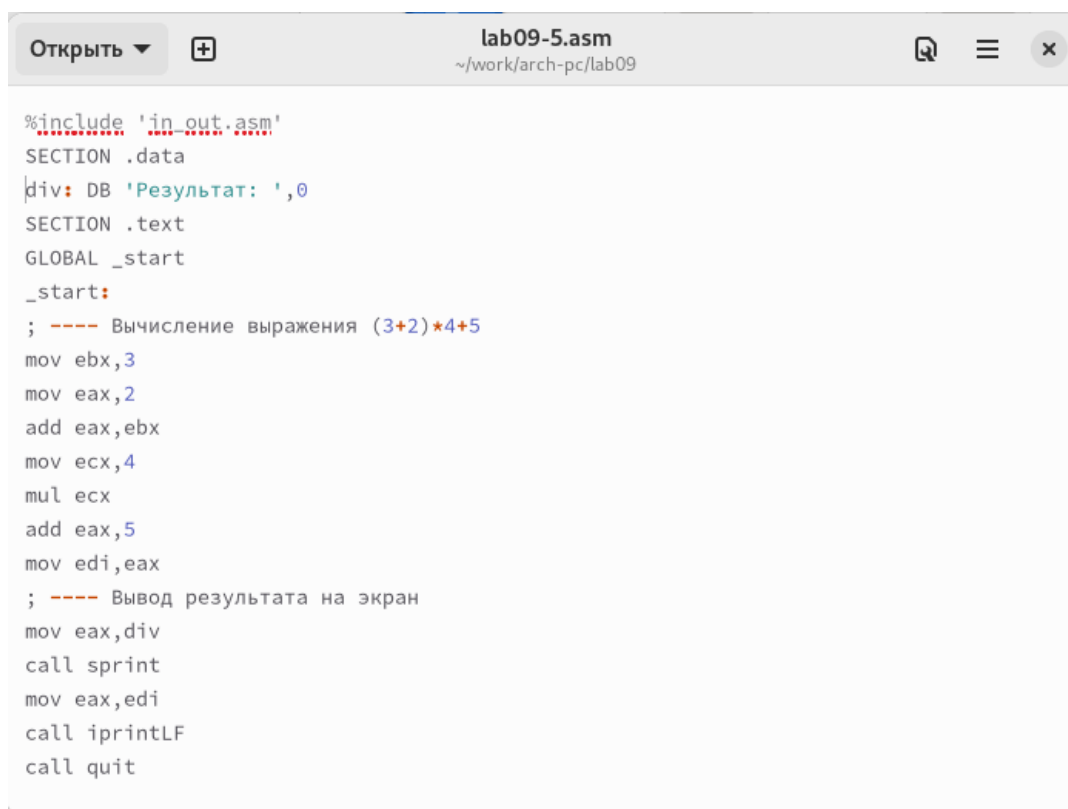
Рис. 4.7:

Ошибка была в строках

```
add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx
```

Рис. 4.8:

Правильно работающая программа представлена

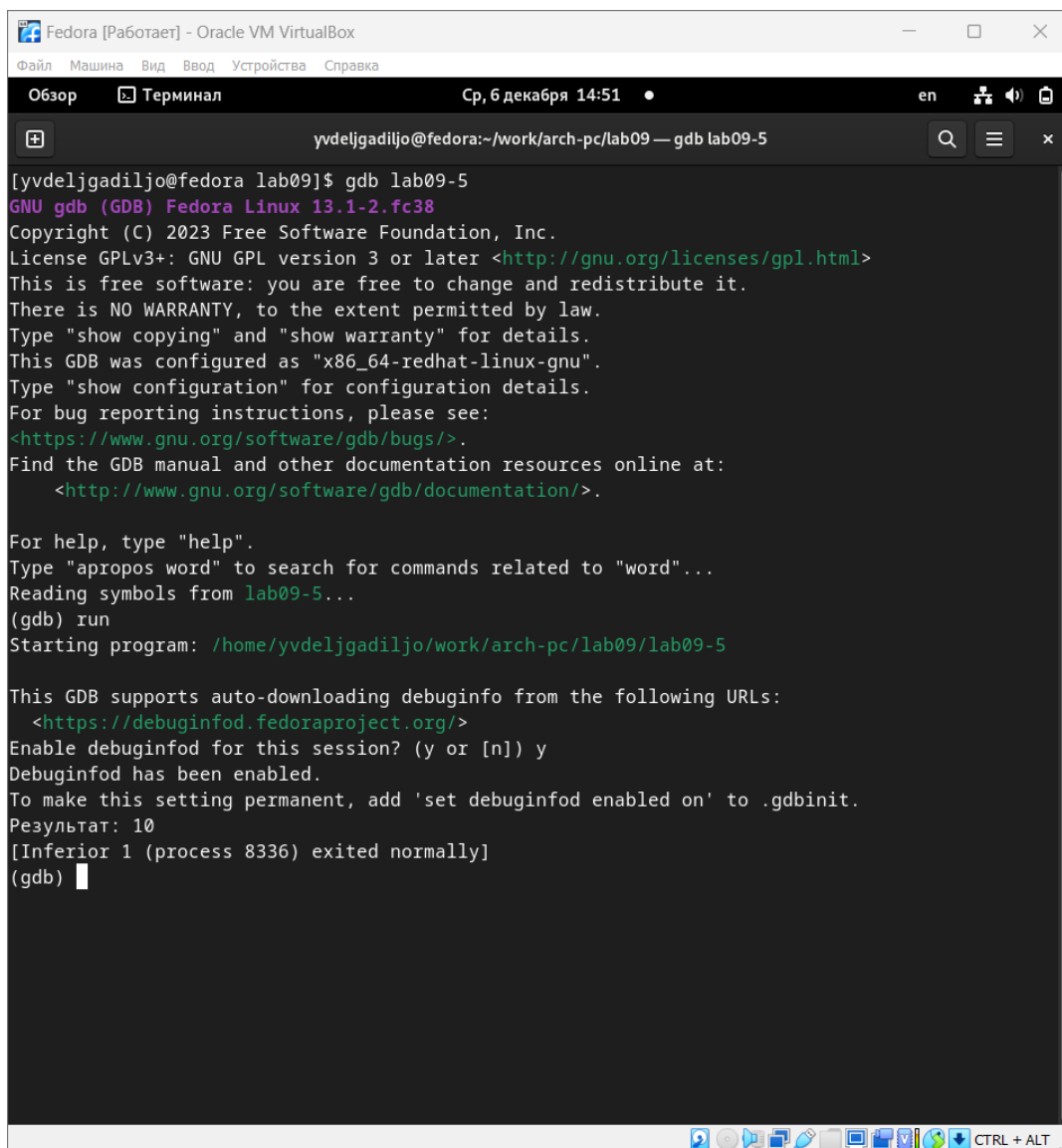


```
Открыть ▾ + lab09-5.asm
~/work/arch-pc/lab09

%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
```

Рис. 4.9:

Проверка корректности работы программы, после исправлений



```
Fedora [Работает] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка
Обзор  Терминал  Ср, 6 декабря 14:51  en
yvdeljgadiljo@fedora:~/work/arch-pc/lab09 — gdb lab09-5

[yvdeljgadiljo@fedora lab09]$ gdb lab09-5
GNU gdb (GDB) Fedora Linux 13.1-2.fc38
Copyright (C) 2023 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 lab09-5...
(gdb) run
Starting program: /home/yvdeljgadiljo/work/arch-pc/lab09/lab09-5

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Результат: 10
[Inferior 1 (process 8336) exited normally]
(gdb)
```

Рис. 4.10:

5 Выводы

В результате выполнения работы, я научился организовывать код в подпрограммы и познакомился с базовыми функциями отладчика gdb.

6 Список литературы

- GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.
- GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.
- Midnight Commander Development Center. — 2021. — URL: <https://midnight-commander.org/>.
- NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.
- Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658>.
- Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
- The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.
- Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
- Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
- Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.
- Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
- Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.

- Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
- Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm_unix.
- Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
- Таненбаум Э., Бос Х. Современные операционн