

# **Отчёт по лабораторной работе 9**

**Дисциплина: архитектура компьютера**

Айдарбекова Алия НММбд-01-23

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>9</b>
4.1	Реализация подпрограмм в NASM . . . . .	9
4.2	Отладка программ с помощью GDB . . . . .	12
4.3	Задание для самостоятельной работы . . . . .	24
<b>5</b>	<b>Выводы</b>	<b>30</b>

## Список иллюстраций

4.1	Изменение кода lab9-1.asm . . . . .	10
4.2	Компиляция текста программы lab9-1.asm . . . . .	10
4.3	Изменение кода lab9-1.asm . . . . .	11
4.4	Компиляция текста программы lab9-1.asm . . . . .	12
4.5	Изменение кода lab9-2.asm . . . . .	13
4.6	Компиляция текста программы lab9-2.asm в отладчике . . . . .	14
4.7	Дизассемблированный код . . . . .	15
4.8	Дизассемблированный код в режиме интел . . . . .	16
4.9	Точка остановки . . . . .	17
4.10	Изменение регистров . . . . .	18
4.11	Изменение регистров . . . . .	19
4.12	Изменение значения переменной . . . . .	20
4.13	Вывод значения регистра . . . . .	21
4.14	Вывод значения регистра . . . . .	22
4.15	Вывод значения регистра . . . . .	23
4.16	Изменение кода prog-1.asm . . . . .	24
4.17	Компиляция текста программы prog-1.asm . . . . .	25
4.18	Код с ошибкой . . . . .	26
4.19	Отладка . . . . .	27
4.20	Код исправлен . . . . .	28
4.21	Проверка работы . . . . .	29

## **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 Задание

1. Изучение подпрограмм в ассемблере
2. Изучение функций отладчика GDB
3. Изучение передачи аргументов с помощью отладчика
4. Выполнение заданий, рассмотрение примеров
5. Выполнение заданий для самостоятельной работы

### 3 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.

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

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.

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. Если в программе встречается одинаковый участок кода, его можно оформить в виде подпрограммы, а во всех нужных местах поставить её вызов. При этом подпрограмма будет содержаться в коде в одном экземпляре, что позволит уменьшить размер кода всей программы.

Для вызова подпрограммы из основной программы используется инструкция `call`, которая заносит адрес следующей инструкции в стек и загружает в регистр `еір` адрес соответствующей подпрограммы, осуществляя таким образом переход. Затем начинается выполнение подпрограммы, которая, в свою очередь, также может содержать подпрограммы.

Подпрограмма завершается инструкцией `ret`, которая извлекает из стека адрес, занесённый туда соответствующей инструкцией `call`, и заносит его в `еір`. После этого выполнение основной программы возобновится с инструкции, следующей за инструкцией `call`.

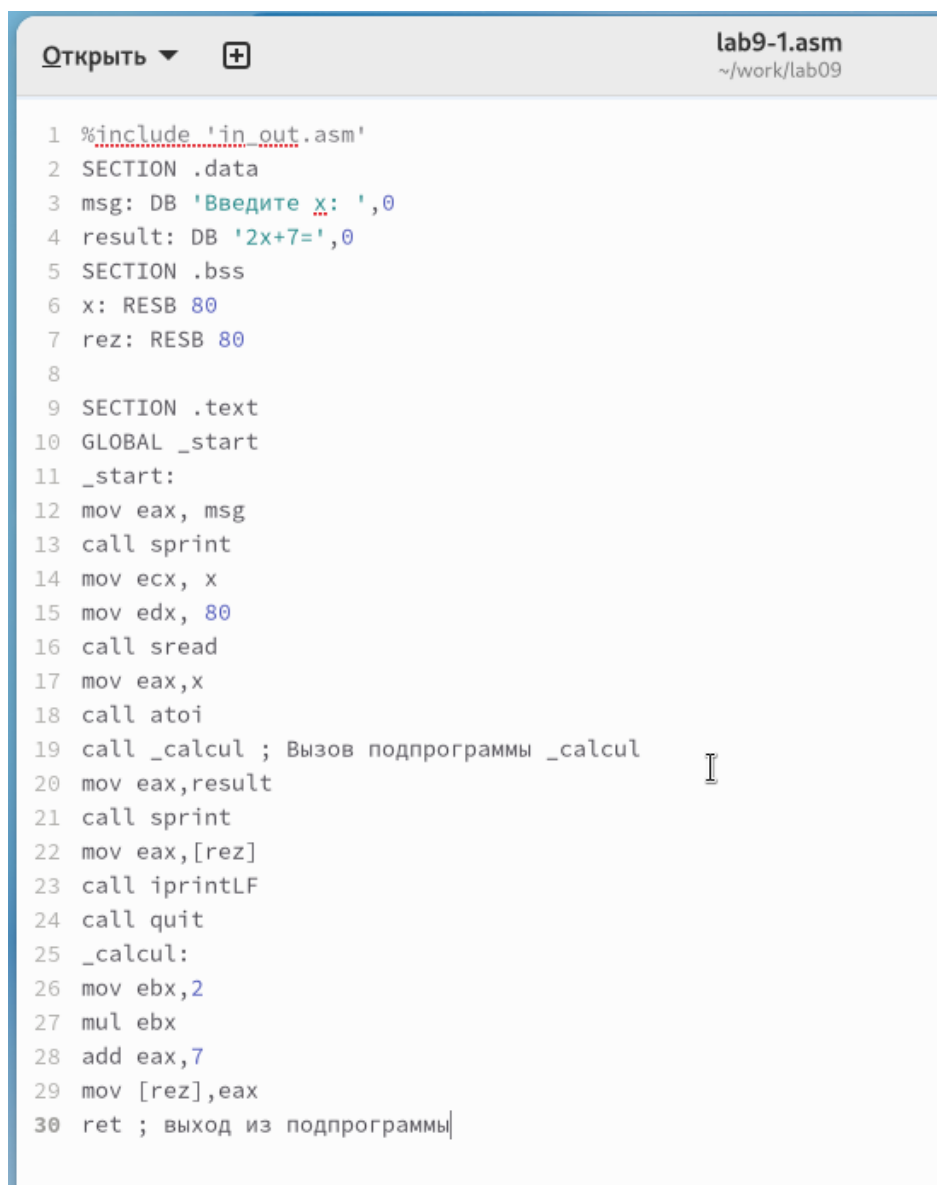


## 4 Выполнение лабораторной работы

### 4.1 Реализация подпрограмм в NASM

Для начала я создала новую директорию и перешла в нее, чтобы выполнить лабораторную работу номер 9. Затем я создала файл с именем lab9-1.asm.

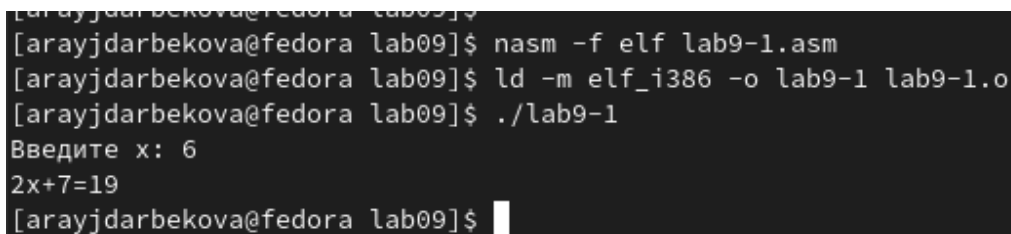
В качестве примера рассмотрим программу, которая вычисляет арифметическое выражение  $f(x) = 2x + 7$  с использованием подпрограммы `calcul`. В данном примере значение переменной `x` вводится с клавиатуры, а само выражение вычисляется внутри подпрограммы. (рис. [4.1]) (рис. [4.2])



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

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
```

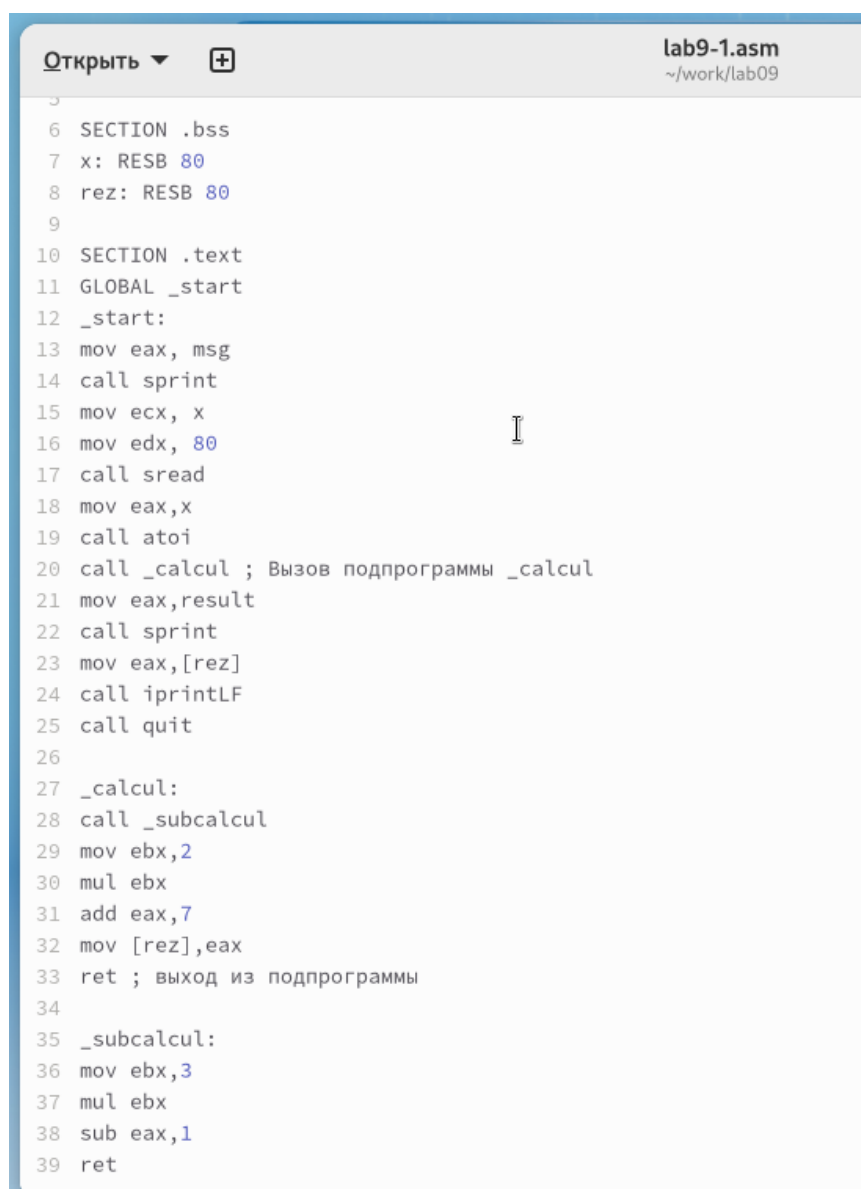
Рис. 4.1: Изменение кода lab9-1.asm



```
[arrayjdarbekova@fedora lab09]$ nasm -f elf lab9-1.asm
[arrayjdarbekova@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[arrayjdarbekova@fedora lab09]$ ./lab9-1
Введите x: 6
2x+7=19
[arrayjdarbekova@fedora lab09]$
```

Рис. 4.2: Компиляция текста программы lab9-1.asm

После этого я внесла изменения в текст программы, добавив подпрограмму `subcalcul` внутрь подпрограммы `calcul`. Это позволяет вычислить составное выражение  $f(g(x))$ , где значение  $x$  также вводится с клавиатуры. Функции определены следующим образом:  $f(x) = 2x + 7$ ,  $g(x) = 3x - 1$ . (рис. [4.3]) (рис. [4.4])



```
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. 4.3: Изменение кода lab9-1.asm

```
[arayjdarbekova@fedora lab09]$  
[arayjdarbekova@fedora lab09]$ nasm -f elf lab9-1.asm  
[arayjdarbekova@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o  
[arayjdarbekova@fedora lab09]$ ./lab9-1  
Введите x: 6  
2(3x-1)+7=41  
[arayjdarbekova@fedora lab09]$
```

Рис. 4.4: Компиляция текста программы lab9-1.asm

## 4.2 Отладка программ с помощью GDB

Я создала файл с именем lab9-2.asm и внесла в него текст программы из Листинга 9.2. Эта программа предназначена для вывода сообщения “Hello world!”. (рис. [4.5])

```
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. 4.5: Изменение кода lab9-2.asm

Далее я скомпилировала файл и получила исполняемый файл. Чтобы добавить отладочную информацию для работы с отладчиком GDB, использовала ключ “-g”.

Затем загрузила полученный исполняемый файл в отладчик GDB и проверила его работу, запустив программу с помощью команды “run” или “r”.(рис. [4.6])

```

[arayjdarbekova@fedora lab09]$ nasm -f elf lab9-1.asm
[arayjdarbekova@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[arayjdarbekova@fedora lab09]$ ./lab9-1
Введите x: 6
2x+7=19
[arayjdarbekova@fedora lab09]$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
[arayjdarbekova@fedora lab09]$ ld -m elf_i386 -o lab9-2 lab9-2.o
[arayjdarbekova@fedora lab09]$ gdb lab9-2
GNU gdb (GDB) Fedora 12.1-2.fc36
Copyright (C) 2022 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/arayjdarbekova/work/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]) n
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
Hello, world!
[Inferior 1 (process 6266) exited normally]
(gdb)

```

Рис. 4.6: Компиляция текста программы lab9-2.asm в отладчике

Для более детального анализа программы я установила точку остановки на метке “start”, с которой начинается выполнение любой ассемблерной программы, и запустила ее. Затем просмотрела дизассемблированный код программы. (рис. [4.7]) (рис. [4.8])

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 11.
(gdb) r
Starting program: /home/arayjdarbekova/work/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. 4.7: Дизассемблированный код

```

(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. 4.8: Дизассемблированный код в режиме интел

Чтобы проверить точку остановки по имени метки “\_start”, я использовала команду “info breakpoints” или “i b”. Затем установила еще одну точку остановки по адресу инструкции, определив адрес предпоследней инструкции “mov ebx, 0x0” (рис. [4.9])



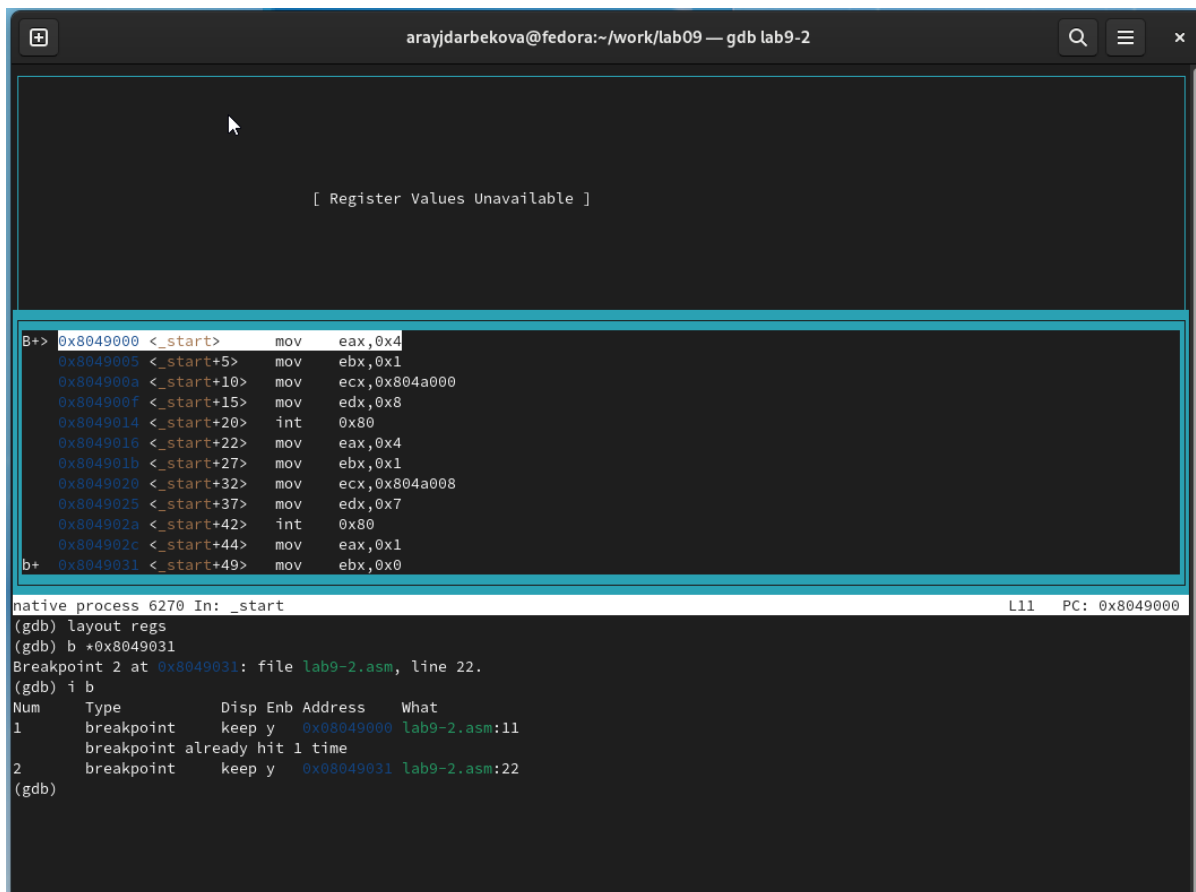


Рис. 4.9: Точка остановки

В отладчике GDB я могу просматривать содержимое ячеек памяти и регистров, а также изменять значения регистров и переменных. Я выполнила 5 инструкций с помощью команды 'stepi' (сокращенно 'si') и отследила изменение значений регистров. (рис. [4.10]) (рис. [4.11])

```
araydarbekova@fedora:~/work/lab09 — gdb lab9-2

Register group: general
eax      0x4      4      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffd200 0xffffd200  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049005 0x8049005 <_start+5>  eflags   0x202    [ IF ]
cs       0x23     35     ss       0x2b     43
ds       0x2b     43     es       0x2b     43
fs       0x0      0      gs       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
0x804902a <_start+42> int 0x80
0x804902c <_start+44> mov eax,0x1
b+ 0x8049031 <_start+49> mov ebx,0x0

native process 6270 In: _start L12 PC: 0x8049005
esp      0xffffd200 0xffffd200
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--es      0x2b     43
fs       0x0      0
gs       0x0      0
(gdb) si
(gdb) 
```

Рис. 4.10: Изменение регистров

```

arraydarbekova@fedora:~/work/lab09 — gdb lab9-2

Register group: general
eax      0x8      8      ecx      0x804a000      134520832
edx      0x8      8      ebx      0x1      1
esp      0xffffd200      0xfffffd200      ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049016      0x8049016 <_start+22>      eflags      0x202      [ IF ]
cs       0x23      35      ss       0x2b      43
ds       0x2b      43      es       0x2b      43
fs       0x0      0      gs       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
0x804902a <_start+42>     int     0x80
0x804902c <_start+44>     mov     eax,0x1
b+ 0x8049031 <_start+49>   mov     ebx,0x0

native process 6270 In: _start      L16      PC: 0x8049016
eip      0x8049000      0x8049000 <_start>
eflags      0x202      [ IF ]
cs       0x23      35
ss       0x2b      43
ds       0x2b      43
--Type <RET> for more, q to quit, c to continue without paging--es      0x2b      43
fs       0x0      0
gs       0x0      0
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 4.11: Изменение регистров

Для просмотра значения переменной `msg1` по имени и получения нужных данных, использовала соответствующую команду.

Для изменения значения регистра или ячейки памяти, использовала команду `set`, указав имя регистра или адрес в качестве аргумента. (рис. [4.12])

```

arayjdarbekova@fedora:~/work/lab09 — gdb lab9-2
Register group: general
eax      0x8      8      ecx      0x804a000      134520832
edx      0x8      8      ebx      0x1      1
esp      0xffffd200      0xffffd200      ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049016      0x8049016 <_start+22>      eflags      0x202      [ IF ]
cs       0x23      35      ss       0x2b      43
ds       0x2b      43      es       0x2b      43
fs       0x0      0      gs       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
0x804902a <_start+42>     int     0x80
0x804902c <_start+44>     mov     eax,0x1
b+ 0x8049031 <_start+49>     mov     ebx,0x0

native process 6270 In: _start      L16      PC: 0x8049016
(gdb) si
(gdb) si
(gdb) x/1sb &msg1
0x804a008 <msg1>:      "Hello, "
0x804a008 <msg2>:      "world!\n\034"
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a008 <msg1>:      "hello, "
(gdb) set {char}0x804a008='L'
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "Lorld!\n\034"
(gdb)

```

Рис. 4.12: Изменение значения переменной

Я успешно изменила первый символ переменной msg1.(рис. [4.13])

```
arayjdarbekova@fedora:~/work/lab09 — gdb lab9-2

Register group: general
eax      0x8      8      ecx      0x804a000    134520832
edx      0x8      8      ebx      0x1      1
esp      0xffffd200  0xffffd200  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049016  0x8049016  <_start+22>  eflags    0x202    [ IF ]
cs       0x23     35      ss       0x2b     43
ds       0x2b     43      es       0x2b     43
fs       0x0      0      gs       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
0x804902a <_start+42>     int     0x80
0x804902c <_start+44>     mov     eax,0x1
b+ 0x8049031 <_start+49>   mov     ebx,0x0

native process 6270 In: _start L16 PC: 0x8049016
$1 = 8
(gdb) p/t $eax
$2 = 1000
(gdb) p/s $ecx
$3 = 134520832
(gdb) p/x $ecx
$4 = 0x804a000
(gdb) p/s $edx
$5 = 8
(gdb) p/t $edx
$6 = 1000
(gdb) p/x $edx
$7 = 0x8
(gdb)
```

Рис. 4.13: Вывод значения регистра

Также, с помощью команды set, я изменила значение регистра ebx на нужное значение. (рис. [4.14])

The screenshot shows a GDB terminal window with the title bar 'arayjdarbekova@fedora: ~/work/lab09 — gdb lab9-2'. The window is divided into three main sections:

- Register group: general:** A table showing the values of general-purpose registers.
 

Register	Value	Register	Value
eax	0x8	ecx	0x804a000
edx	0x8	ebx	0x2
esp	0xffffd200	ebp	0x0
esi	0x0	edi	0x0
eip	0x8049016	eflags	0x202
cs	0x23	ss	0x2b
ds	0x2b	es	0x2b
fs	0x0	gs	0x0
- Assembly code:** A list of instructions with their addresses and disassembled forms. The instruction at address 0x8049016 is highlighted:
 

```

      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
      b+ 0x8049031 <_start+49> mov    ebx,0x0
      
```
- Native process 6270 In: \_start:** A section showing GDB commands and their results:
 

```

      $4 = 0x804a000
      (gdb) p/s $edx
      $5 = 8
      (gdb) p/t $edx
      $6 = 1000
      (gdb) p/x $edx
      $7 = 0x8
      (gdb) set $ebx='2'
      (gdb) p/s $ebx
      $8 = 50
      (gdb) set $ebx=2
      (gdb) p/s $ebx
      $9 = 2
      (gdb)
      
```

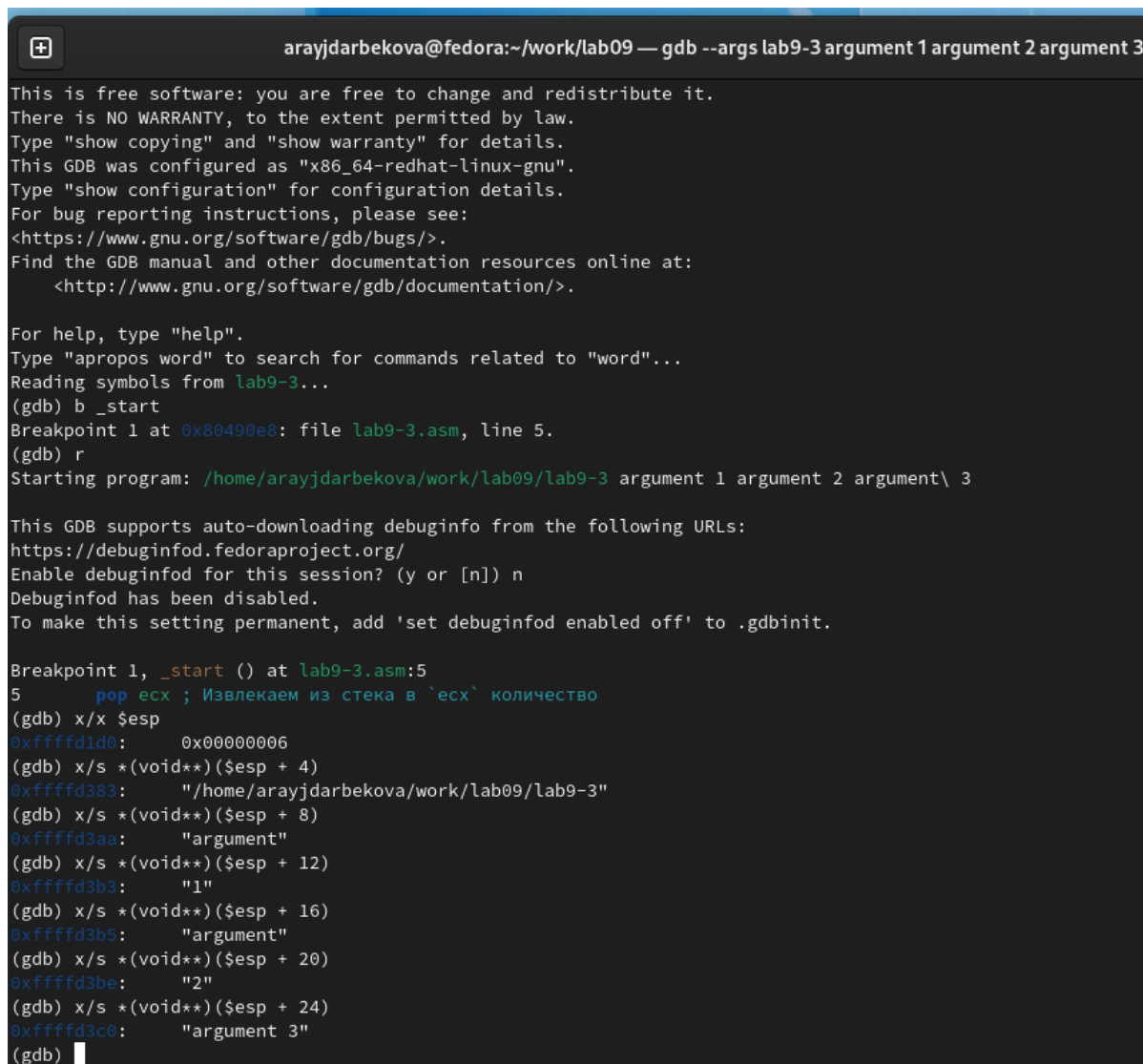
Рис. 4.14: Вывод значения регистра

Я скопировала файл lab8-2.asm, который был создан во время выполнения лабораторной работы №8. Этот файл содержит программу для вывода аргументов командной строки. Затем создала исполняемый файл из скопированного файла.

Для загрузки программы с аргументами в gdb использовала ключ `-args` и загрузила исполняемый файл в отладчик с указанными аргументами. Я установила точку останова перед первой инструкцией программы и запустила ее.

Адрес вершины стека, где хранится количество аргументов командной строки (включая имя программы), хранится в регистре `esp`. По этому адресу я нашла число, указывающее количество аргументов. В данном случае увидела, что количество аргументов равно 5, включая имя программы lab9-3 и сами аргументы: аргумент1, аргумент2 и 'аргумент 3'.

Я также просмотрела остальные позиции стека. По адресу [esp+4] находится адрес в памяти, где располагается имя программы. По адресу [esp+8] хранится адрес первого аргумента, по адресу [esp+12] - второго и так далее. Шаг изменения адреса равен 4, так как каждый следующий адрес на стеке находится на расстоянии 4 байт от предыдущего ([esp+4], [esp+8], [esp+12]). (рис. [4.15])



```
arayjdarbekova@fedora:~/work/lab09 — gdb --args lab9-3 argument 1 argument 2 argument 3
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 lab9-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 5.
(gdb) r
Starting program: /home/arayjdarbekova/work/lab09/lab9-3 argument 1 argument 2 argument\ 3

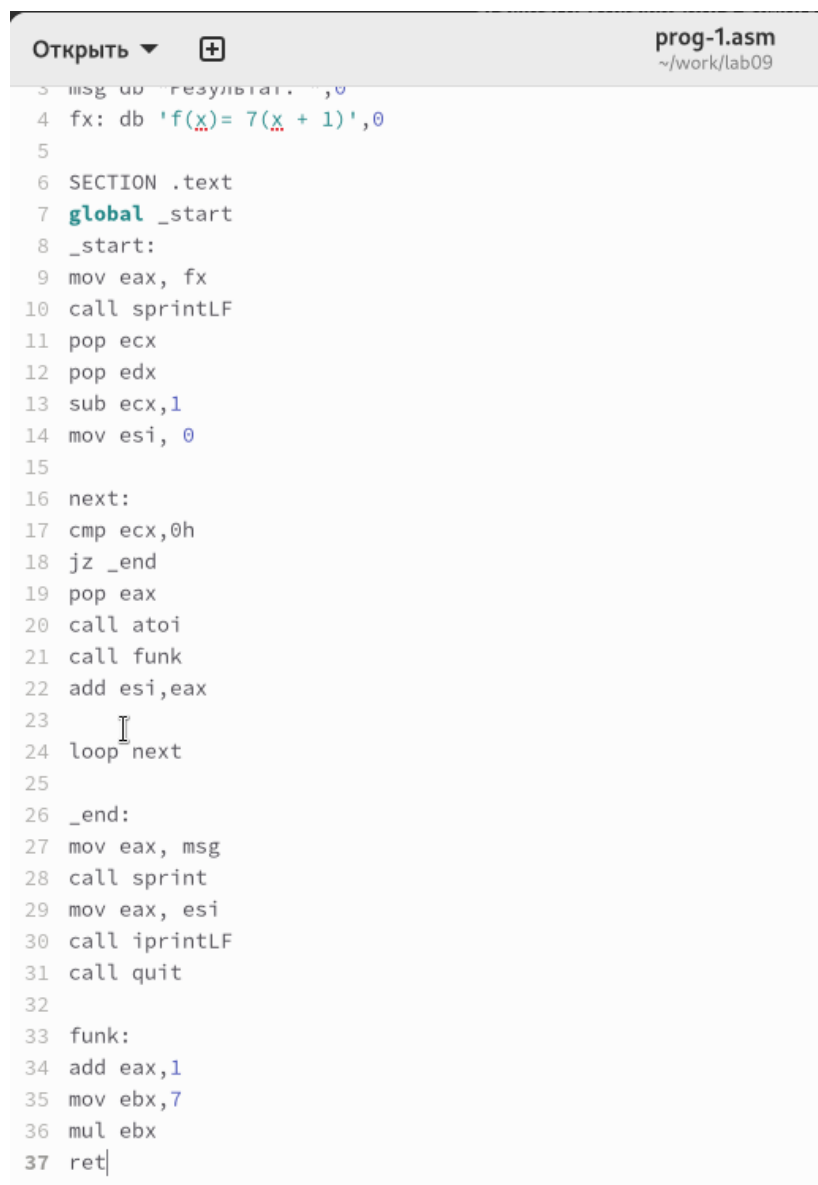
This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
Enable debuginfod for this session? (y or [n]) n
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.

Breakpoint 1, _start () at lab9-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffd1d0: 0x00000006
(gdb) x/s *(void**)(esp + 4)
0xffffd383: "/home/arayjdarbekova/work/lab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd3aa: "argument"
(gdb) x/s *(void**)(esp + 12)
0xffffd3b3: "1"
(gdb) x/s *(void**)(esp + 16)
0xffffd3b5: "argument"
(gdb) x/s *(void**)(esp + 20)
0xffffd3be: "2"
(gdb) x/s *(void**)(esp + 24)
0xffffd3c0: "argument 3"
(gdb) █
```

Рис. 4.15: Вывод значения регистра

## 4.3 Задание для самостоятельной работы

Я переписала программу из лабораторной работы №8, задание №1, чтобы реализовать вычисление значения функции  $f(x)$  как подпрограмму. (рис. [4.16]) (рис. [4.17])



```
Открыть ▾ + prog-1.asm
~/work/lab09
3 msg db "Результат: ",0
4 fx: db 'f(x)= 7(x + 1)',0
5
6 SECTION .text
7 global _start
8 _start:
9 mov eax, fx
10 call sprintf
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 funk
22 add esi,eax
23
24 loop next
25
26 _end:
27 mov eax, msg
28 call sprintf
29 mov eax, esi
30 call iprintLF
31 call quit
32
33 funk:
34 add eax,1
35 mov ebx,7
36 mul ebx
37 ret
```

Рис. 4.16: Изменение кода prog-1.asm



```


[arayjdarbekova@fedora lab09]$
[arayjdarbekova@fedora lab09]$ nasm -f elf prog-1.asm
[arayjdarbekova@fedora lab09]$ ld -m elf_i386 -o prog-1 prog-1.o
[arayjdarbekova@fedora lab09]$ ./prog-1
f(x)= 7(x + 1)
Результат: 0
[arayjdarbekova@fedora lab09]$ ./prog-1 1
f(x)= 7(x + 1)
Результат: 14
[arayjdarbekova@fedora lab09]$ ./prog-1 1 3 6 9 7
f(x)= 7(x + 1)
Результат: 217
[arayjdarbekova@fedora lab09]$

```

Рис. 4.17: Компиляция текста программы prog-1.asm

Приведенный ниже код представляет программу для вычисления выражения  $(3 + 2) * 4 + 5$ . Однако, при запуске программа дает неверный результат.

Я провела анализ изменений значений регистров с помощью отладчика GDB и обнаружила ошибку: перепутан порядок аргументов у инструкции add. Кроме того, заметила, что по окончании работы в регистр edi передается значение ebx вместо eax.(рис. [4.18]) (рис. [4.19])

Открыть ▾ 

prog-2.asm  
~/work/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. 4.18: Код с ошибкой

```

arraydarbekova@fedora:~/work/lab09 — gdb prog-2
eax 0x804a000 134520832 ecx 0x4 4
edx 0x0 0 ebx 0xa 10
esp 0xffffd200 0xffffd200 ebp 0x0 0x0
esi 0x0 0 edi 0xa 10
eip 0x8049105 0x8049105 <_start+29> eflags 0x206 [ PF IF ]
cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43
fs 0x0 0 gs 0x0 0

B+ 0x80490e8 <_start> mov ebx,0x3
0x8049105 <_start+29> call 0x804900f <sprint>
0x804910a <_start+34> add ebx,edi
0x804910c <_start+36> call 0x8049086 <iprintLF>
0x8049111 <_start+41> call 0x80490db <quit>
,0x5

04a000
>
rint>

86 <iprintLF>

native process 6508 In: _start L17 PC: 0x8049105
To makeNo process In: , add 'set debuginfod enabled off' to .gdbinit. L?? PC: ??
Breakpoint 1, _start () at prog-2.asm:8
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) c
Continuing.
Результат: 10
[Inferior 1 (process 6508) exited normally]
(gdb)

```

Рис. 4.19: Отладка

Я исправила код программы, учитывая перепутанный порядок аргументов у инструкции add и правильную передачу значения в регистр edi по окончании работы программы. (рис. [4.20]) (рис. [4.21])

```
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. 4.20: Код исправлен

```
arayjdarbekova@fedora:~/work/lab09 — gdb prog-2
eax 0x19 25 ecx 0x4 4
edx 0x0 0 ebx 0x3 3
esp 0xffffd200 0xffffd200 ebp 0x0 0x0
esi 0x0 0 edi 0x0 0
eip 0x80490fe 0x80490fe <_start+22> eflags 0x202 [ IF ]
cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43
fs 0x0 0 gs 0x0 0

B+ 0x80490e8 <_start> mov ebx,0x3
0x80490fe <_start+22> mov edi,eax04a000
0x8049105 <_start+29> call 0x804900f <sprint>
0x804910a <_start+34> mov eax,edi
0x804910c <_start+36> call 0x8049086 <iprintLF>
0x8049111 <_start+41> call 0x80490db <quit>

native process 6551 In: _start L14 PC: 0x80490fe
https://No process In: ct.org/ L?? PC: ??
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.

Breakpoint 1, _start () at prog-2.asm:8
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) c
Continuing.
Результат: 25
[Inferior 1 (process 6551) exited normally]
(gdb)
```

Рис. 4.21: Проверка работы

## **5 Выводы**

Освоили работу с подпрограммами и отладчиком.