

# **Отчет по лабораторной работе 10**

**Понятие подпрограммы. Отладчик GDB.**

Гисматуллин Артём Вадимович НПИбд-01-22

# 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>
3.1	Понятие об отладке . . . . .	7
3.2	Методы отладки . . . . .	8
3.3	Основные возможности отладчика GDB . . . . .	9
3.3.1	Понятие подпрограммы . . . . .	9
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>11</b>
<b>5</b>	<b>Выполнение задания для самостоятельной работы</b>	<b>25</b>
<b>6</b>	<b>Выводы</b>	<b>30</b>
	<b>Список литературы</b>	<b>31</b>

## Список иллюстраций

4.1	Командная строка. каталог ~/work/arch-pc/lab10 . . . . .	11
4.2	Командная строка. Выполнение программы lab10-1.asm . . . . .	12
4.3	Командная строка. Выполнение измененной программы lab10-1.asm	13
4.4	Командная строка. Отладка программы lab10-2.asm . . . . .	14
4.5	Командная строка. Отладка программы lab10-2.asm 2 . . . . .	15
4.6	Командная строка. Intel синтаксис . . . . .	16
4.7	Командная строка. Режим псевдографики . . . . .	17
4.8	Командная строка. Выполнение инструкций. . . . .	18
4.9	Командная строка. Просмотр значения переменной . . . . .	19
4.10	Командная строка. Просмотр и изменение значения переменных	20
4.11	Командная строка. Значения регистров eax и edx . . . . .	21
4.12	Командная строка. Изменение значения регистра ebx . . . . .	22
4.13	Командная строка. Запуск в gdb lab10-3.asm . . . . .	23
4.14	Командная строка. Просмотр содержимого очереди . . . . .	24
5.1	Командная строка. Программа lab10-4.asm . . . . .	26
5.2	Командная строка. Отладка программы lab10-5.asm . . . . .	27
5.3	Командная строка. Отладка программы lab10-5.asm 2 . . . . .	28
5.4	Командная строка. Выполнение верной программы lab10-5.asm .	29

## **Список таблиц**

# 1 Цель работы

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

## 2 Задание

1. Рассмотрите программу вычисления выражения  $f(x) = 2x + 7$  с помощью подпрограммы. Добавьте в нее подпрограмму вычисления выражения  $f(g(x))$ .
2. Проведите отладку программы листинга 10.2. Выполните соответствующие задания по изучению процесса отладки (точки останова, работа с данными программами в gdb, обработка аргументов).
3. Преобразовать программу из лабораторной работы №9, реализовав вычисления как подпрограмму.
4. С помощью отладчика gdb определить ошибку программы листинга 10.3.

## 3 Теоретическое введение

### 3.1 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.

## 3.2 Методы отладки

Наиболее часто применяют следующие методы отладки:

- создание точек контроля значений на входе и выходе участка программы;
- 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 строчки, 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.



## 3.3 Основные возможности отладчика GDB

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. GDB может выполнять следующие действия:

- начать выполнение программы, задав всё, что может повлиять на её поведение;
- остановить программу при указанных условиях;
- исследовать, что случилось, когда программа остановилась;
- изменить программу так, чтобы можно было поэкспериментировать с устранением эффектов одной ошибки и продолжить выявление других.

### 3.3.1 Понятие подпрограммы

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.

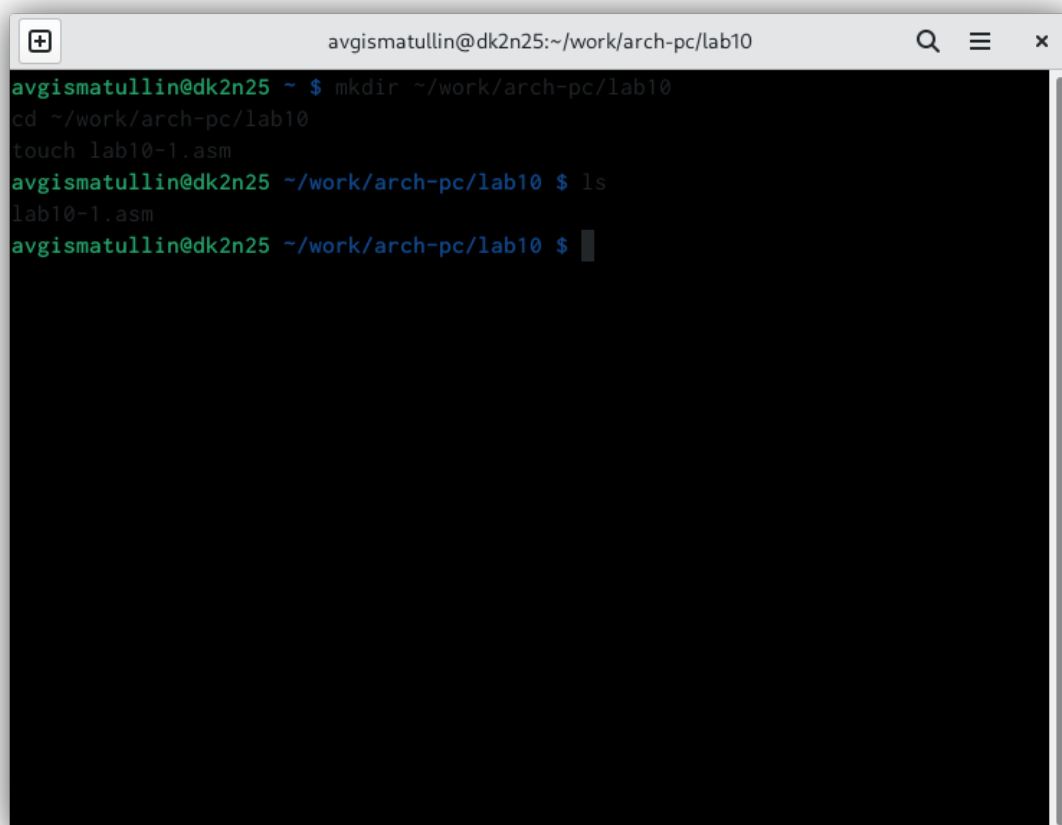
Для вызова подпрограммы из основной программы используется инструкция `call`, которая заносит адрес следующей инструкции в стек и загружает в регистр `еір` адрес соответствующей подпрограммы, осуществляя таким образом переход. Затем начинается выполнение подпрограммы, которая, в свою очередь, также может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

Подпрограмма может вызываться как из внешнего файла, так и быть частью основной программы.

## 4 Выполнение лабораторной работы

1. Создаем каталог для программ лабораторной работы No 10, перейдем в него и создаем файл lab10-1.asm. (рис. 4.1)

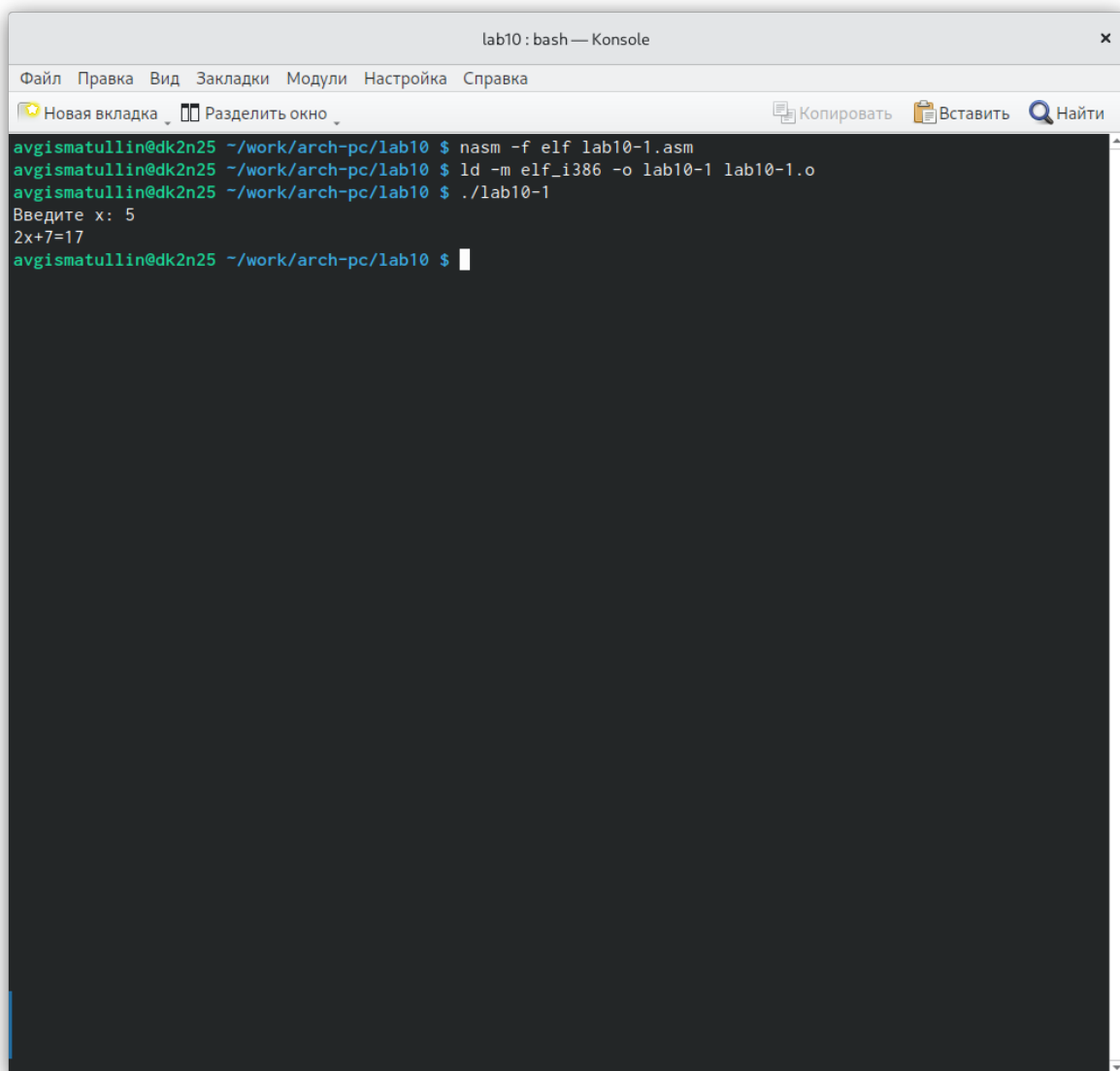


```
avgismatullin@dk2n25:~/work/arch-pc/lab10
avgismatullin@dk2n25 ~ $ mkdir ~/work/arch-pc/lab10
cd ~/work/arch-pc/lab10
touch lab10-1.asm
avgismatullin@dk2n25 ~/work/arch-pc/lab10 $ ls
lab10-1.asm
avgismatullin@dk2n25 ~/work/arch-pc/lab10 $
```

Рис. 4.1: Командная строка. каталог ~/work/arch-pc/lab10

2. Ввели в файл lab10-1.asm текст программы листинга, создали исполняемый

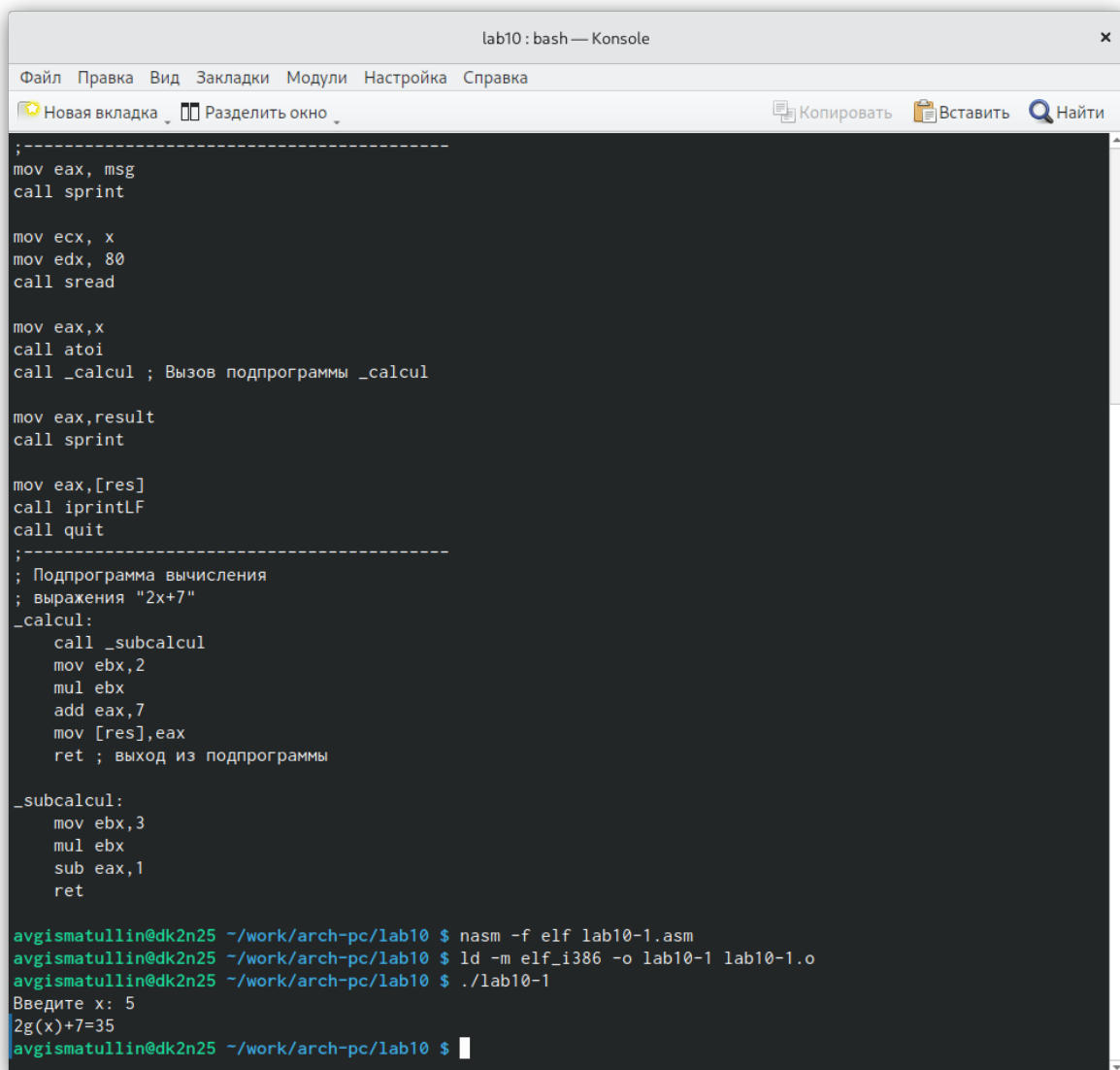
файл и посмотрели результат программы. (рис. 4.2)



```
lab10: bash — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
Новая вкладка  Разделить окно  Копировать  Вставить  Найти
avgismatullin@dk2n25 ~/work/arch-pc/lab10 $ nasm -f elf lab10-1.asm
avgismatullin@dk2n25 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o lab10-1 lab10-1.o
avgismatullin@dk2n25 ~/work/arch-pc/lab10 $ ./lab10-1
Введите x: 5
2x+7=17
avgismatullin@dk2n25 ~/work/arch-pc/lab10 $
```

Рис. 4.2: Командная строка. Выполнение программы lab10-1.asm

Далее добавляем подпрограмму для вычисления функции  $g(x) = 3x - 1$ , и выполняем программу. Результат: (рис. 4.3)



```
lab10: bash — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
Новая вкладка  Разделить окно  Копировать  Вставить  Найти

;-----
mov eax, msg
call sprint

mov ecx, x
mov edx, 80
call sread

mov eax, x
call atoi
call _calcul ; Вызов подпрограммы _calcul

mov eax, result
call sprint

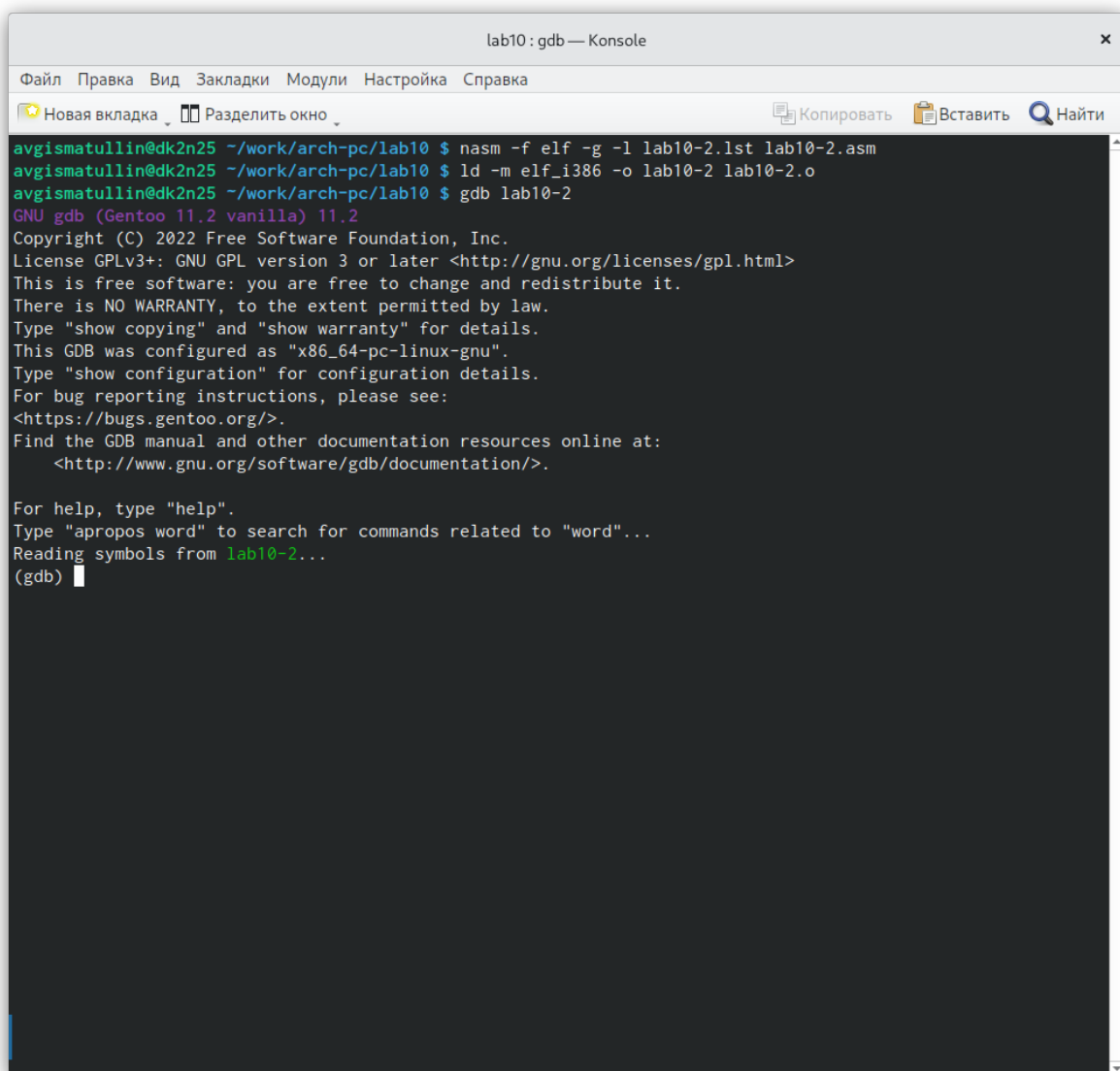
mov eax, [res]
call iprintLF
call quit
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
    call _subcalcul
    mov ebx, 2
    mul ebx
    add eax, 7
    mov [res], eax
    ret ; выход из подпрограммы

_subcalcul:
    mov ebx, 3
    mul ebx
    sub eax, 1
    ret

avgismatullin@dk2n25 ~/work/arch-pc/lab10 $ nasm -f elf lab10-1.asm
avgismatullin@dk2n25 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o lab10-1 lab10-1.o
avgismatullin@dk2n25 ~/work/arch-pc/lab10 $ ./lab10-1
Введите x: 5
2g(x)+7=35
avgismatullin@dk2n25 ~/work/arch-pc/lab10 $
```

Рис. 4.3: Командная строка. Выполнение измененной программы lab10-1.asm

3. Создаем файл lab10-2.asm с текстом программы из Листинга 10.2, получаем исполняемый файл с ключом '-g'. (рис. 4.4)

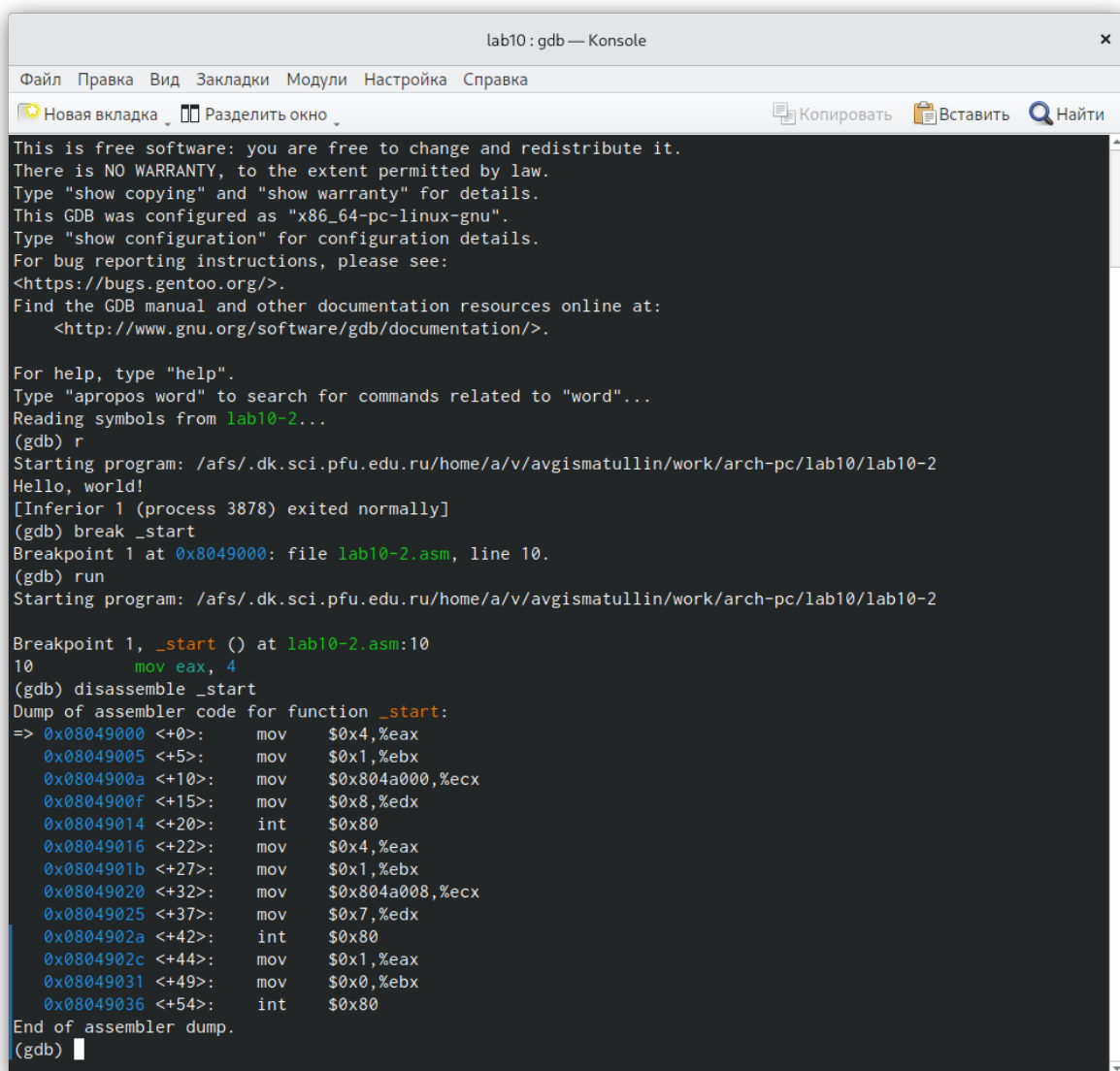


```
lab10: gdb — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
* Новая вкладка  Разделить окно  Копировать  Вставить  Найти
avgismatullin@dk2n25 ~/work/arch-pc/lab10 $ nasm -f elf -g -l lab10-2.lst lab10-2.asm
avgismatullin@dk2n25 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o lab10-2 lab10-2.o
avgismatullin@dk2n25 ~/work/arch-pc/lab10 $ gdb lab10-2
GNU gdb (Gentoo 11.2 vanilla) 11.2
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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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 lab10-2...
(gdb) 
```

Рис. 4.4: Командная строка. Отладка программы lab10-2.asm

Проверяем работу программы, запустив ее в оболочке GDB с помощью команды `run`, устанавливаем брейкпоинт на метку `_start`, посмотрели дисассимилированный код программы с помощью команды `disassemble` (рис. 4.5)



```
lab10: gdb — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
* Новая вкладка  Разделить окно  Копировать  Вставить  Найти

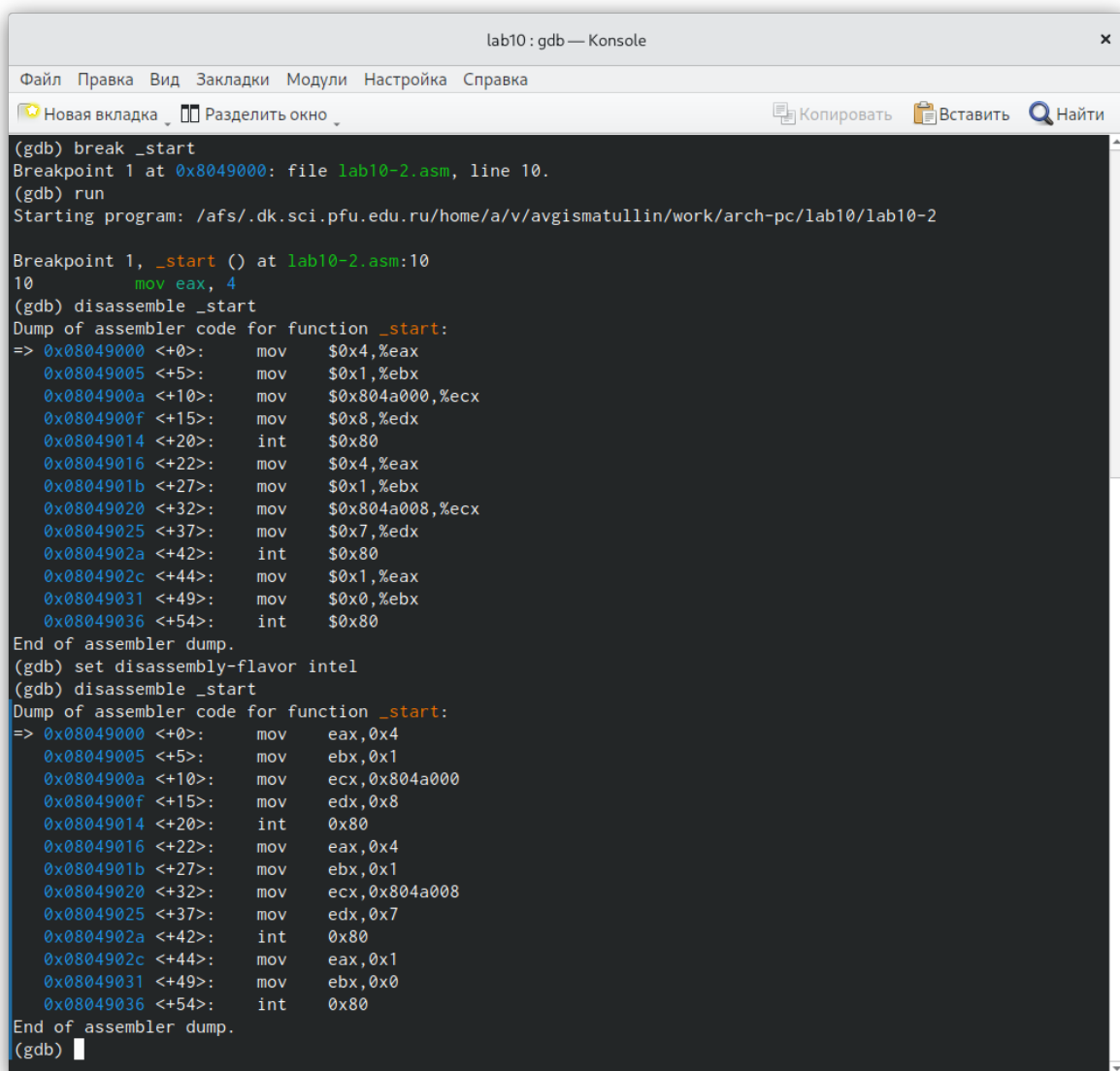
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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 lab10-2...
(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/v/avgismatullin/work/arch-pc/lab10/lab10-2
Hello, world!
[Inferior 1 (process 3878) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 10.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/v/avgismatullin/work/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:10
10      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. 4.5: Командная строка. Отладка программы lab10-2.asm 2

Затем переключаем на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel` (рис. 4.6)



```
lab10: gdb — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
Новая вкладка  Разделить окно  Копировать  Вставить  Найти
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 10.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/v/avgismatullin/work/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:10
10      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) 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. 4.6: Командная строка. Intel синтаксис

Различия отображения синтаксиса машинных команд в режимах АТТ и Intel отличаются тем, что название регистра и адрес в режиме Intel меняются местами и изменяются так, что читаемость листинга отладки становится лучше.

Затем включаем режим псевдографики для более удобного анализа программы: (рис. 4.7)



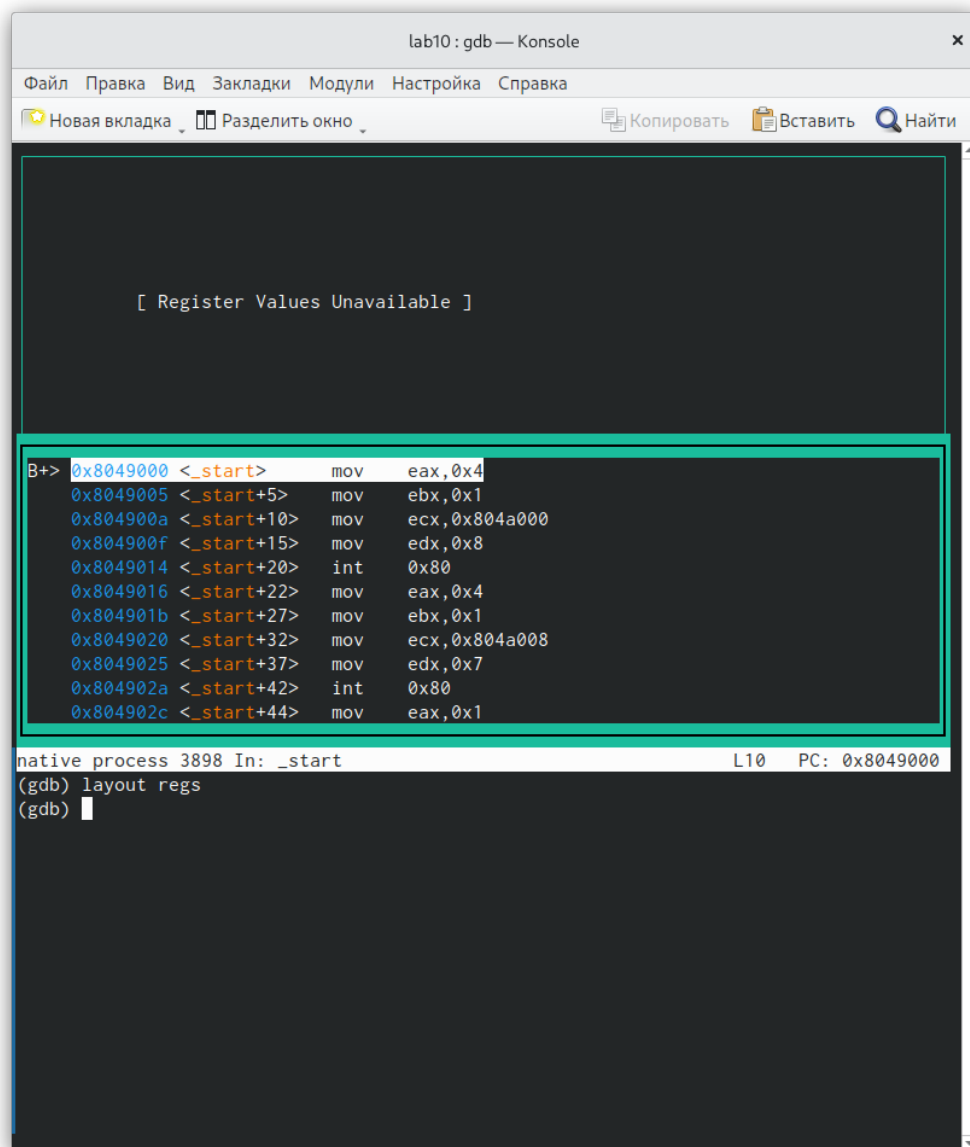


Рис. 4.7: Командная строка. Режим псевдографики

4. Далее устанавливаем еще одну точку останова по адресу инструкции, посмотрим информацию о всех установленных точках останова, а после начинаем работу с данными программы, выполняя инструкции командой `stepi (si)`: (рис. 4.8)

The screenshot shows a GDB console window titled "lab10: gdb — Konsole". The menu bar includes "Файл", "Правка", "Вид", "Закладки", "Модули", "Настройка", and "Справка". The toolbar has icons for "Новая вкладка", "Разделить окно", "Копировать", "Вставить", and "Найти".

The "Register group: general" section displays the following register values:

Register	Value	Comment
eax	0x8	8
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x1	1
esp	0xffffc4a0	0xffffc4a0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x8049016	0x8049016 <_start+22>
eflags	0x202	[ IF ]
cs	0x23	35

The assembly window shows instructions starting from address 0x8049000. The instruction at 0x8049016, `mov eax, 0x4`, is highlighted with a green box. The instruction list includes:

- 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

The status bar shows "native process 3898 In: \_start L16 PC: 0x8049016". The command history shows:

```
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab10-2.asm, line 23.
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab10-2.asm:10
breakpoint already hit 1 time
2 breakpoint keep y 0x08049031 lab10-2.asm:23
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
```

Рис. 4.8: Командная строка. Выполнение инструкций.

Можно проследить, как изменяются значения регистров `eax`, `ebx`, `ecx`, `edx`, `eip` и др.

- После просмотра содержимого регистров, смотрим значение переменной `msg1`. (рис. 4.9)

The screenshot shows a GDB console window titled "lab10 : gdb — Konsole". The menu bar includes "Файл", "Правка", "Вид", "Закладки", "Модули", "Настройка", and "Справка". The toolbar has icons for "Новая вкладка", "Разделить окно", "Копировать", "Вставить", and "Найти".

The "Register group: general" section displays the following register values:

Register	Value	Comment
ebx	0x1	1
esp	0xffffc4a0	0xffffc4a0
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

The assembly code window shows the following instructions:

```
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
```

The "native process 3898 In: \_start" section shows the following register values:

Register	Value	Comment
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

The console output shows the following commands and results:

```
--Type <RET> for more, q to quit, c to continue without paging--cfs      0x0
0
gs      0x0      0
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) 
```

Рис. 4.9: Командная строка. Просмотр значения переменной

6. Смотрим значение переменной `msg1` по адресу, изменяем значение для этой переменной, смотрим значение переменной `msg2` и проводим с ним изменения первого символа: (рис. 4.8)

```
lab10: gdb — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
Новая вкладка  Разделить окно  Копировать  Вставить  Найти

Register group: general
ebx      0x1      1
esp      0xffffc4a0  0xffffc4a0
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

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
0x8049036 <_start+54> int 0x80
0x8049038 add BYTE PTR [eax],al
0x804903a add BYTE PTR [eax],al
0x804903c add BYTE PTR [eax],al
0x804903e add BYTE PTR [eax],al

native process 3898 In: _start L16 PC: 0x8049016
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb) x/1sb &msg2
0x804a008 <msg2>: "world!\n\034"
(gdb) set {char}&msg2='W'
'msg2' has unknown type; cast it to its declared type
(gdb) set {char}&msg2='W'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "World!\n\034"
(gdb) 
```

Рис. 4.10: Командная строка. Просмотр и изменение значения переменных

7. Далее выводим в различных форматах значение регистров eax и edx. (рис. 4.11)

The screenshot shows a GDB console window titled 'lab10: gdb — Konsole'. The window has a menu bar with 'Файл', 'Правка', 'Вид', 'Закладки', 'Модули', 'Настройка', and 'Справка'. Below the menu bar is a toolbar with icons for 'Новая вкладка', 'Разделить окно', 'Копировать', 'Вставить', and 'Найти'. The main content area is divided into two sections. The top section, titled 'Register group: general', displays the values of various registers: 

Register	Value	Comment
ebx	0x1	1
esp	0xffffc4a0	0xffffc4a0
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

. The bottom section, titled 'b+', displays assembly code: 

```
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
0x8049036 <_start+54> int     0x80
0x8049038          add     BYTE PTR [eax],al
0x804903a          add     BYTE PTR [eax],al
0x804903c          add     BYTE PTR [eax],al
0x804903e          add     BYTE PTR [eax],al
```

. At the bottom of the window, the status bar shows 'native process 3898 In: \_start L16 PC: 0x8049016'. The command prompt shows the following sequence of commands and their outputs: 

```
(gdb) p/s $eax
$1 = 8
(gdb) p/t $eax
$2 = 1000
(gdb) p/s $edx
$3 = 8
(gdb) p/t #edx
Invalid character '#' in expression.
(gdb) p/t $edx
$4 = 1000
(gdb) p/s $edx
$5 = 8
(gdb) p/x $edx
$6 = 0x8
(gdb)
```

Рис. 4.11: Командная строка. Значения регистров eax и edx

Более того, значения регистров можно изменять с помощью команды set: (рис. 4.12)

```
lab10: gdb — Konsole
Файл Правка Вид Закладки Модули Настройка Справка
Новая вкладка Разделить окно Копировать Вставить Найти

Register group: general
ebx      0x2      2
esp      0xffffc4a0 0xffffc4a0
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

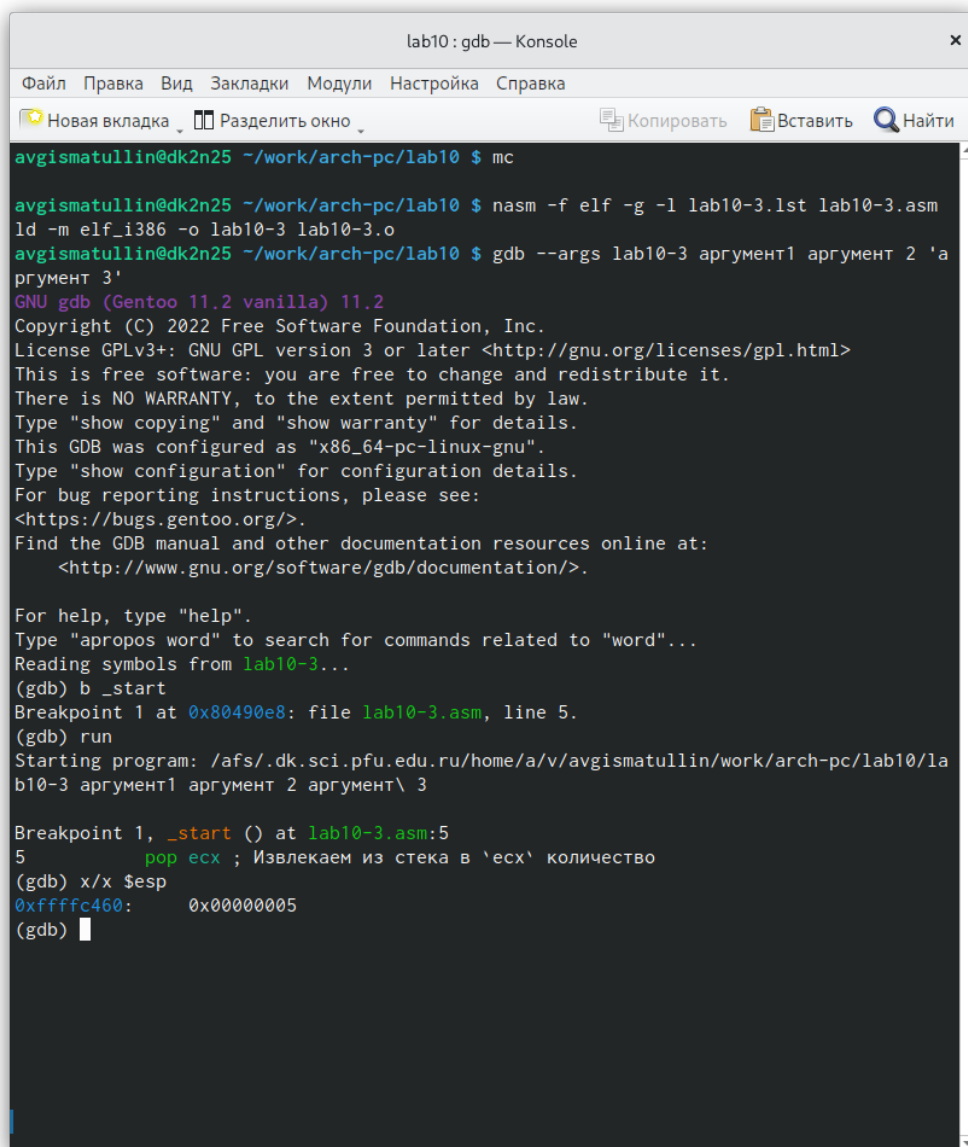
0x804901b <_start+27> mov ebx, 0x1
0x8049020 <_start+32> mov ecx, 0x804a008
0x8049025 <_start+37> mov edx, 0x7
0x804902a <_start+42> int 0x80
0x804902c <_start+44> mov eax, 0x1
b+ 0x8049031 <_start+49> mov ebx, 0x0
0x8049036 <_start+54> int 0x80
0x8049038 add BYTE PTR [eax], al
0x804903a add BYTE PTR [eax], al
0x804903c add BYTE PTR [eax], al
0x804903e add BYTE PTR [eax], al

native process 3898 In: _start L16 PC: 0x8049016
(gdb) p/t $edx
$4 = 1000
(gdb) p/s $edx
$5 = 8
(gdb) p/x $edx
$6 = 0x8
(gdb) set $ebx=2
Unmatched single quote.
(gdb) set $ebx='2'
$7 = 50
(gdb) set $ebx=2
$8 = 2
(gdb) 
```

Рис. 4.12: Командная строка. Изменение значения регистра ebx

Разница между выводом p/s для '2' и 2 заключается в том, что для данных символов возвращается число из таблицы символов Unicode представляющее их позицию, а строковые объекты и числа имеют разные значения.

8. Завершив выполнение программы, копируем файл lab9-2.asm в lab10-3.asm и запускаем отладчик в программе с аргументами. Устанавливаем брейкпоинт и смотрим количество аргументов командной строки: (рис. 4.13)



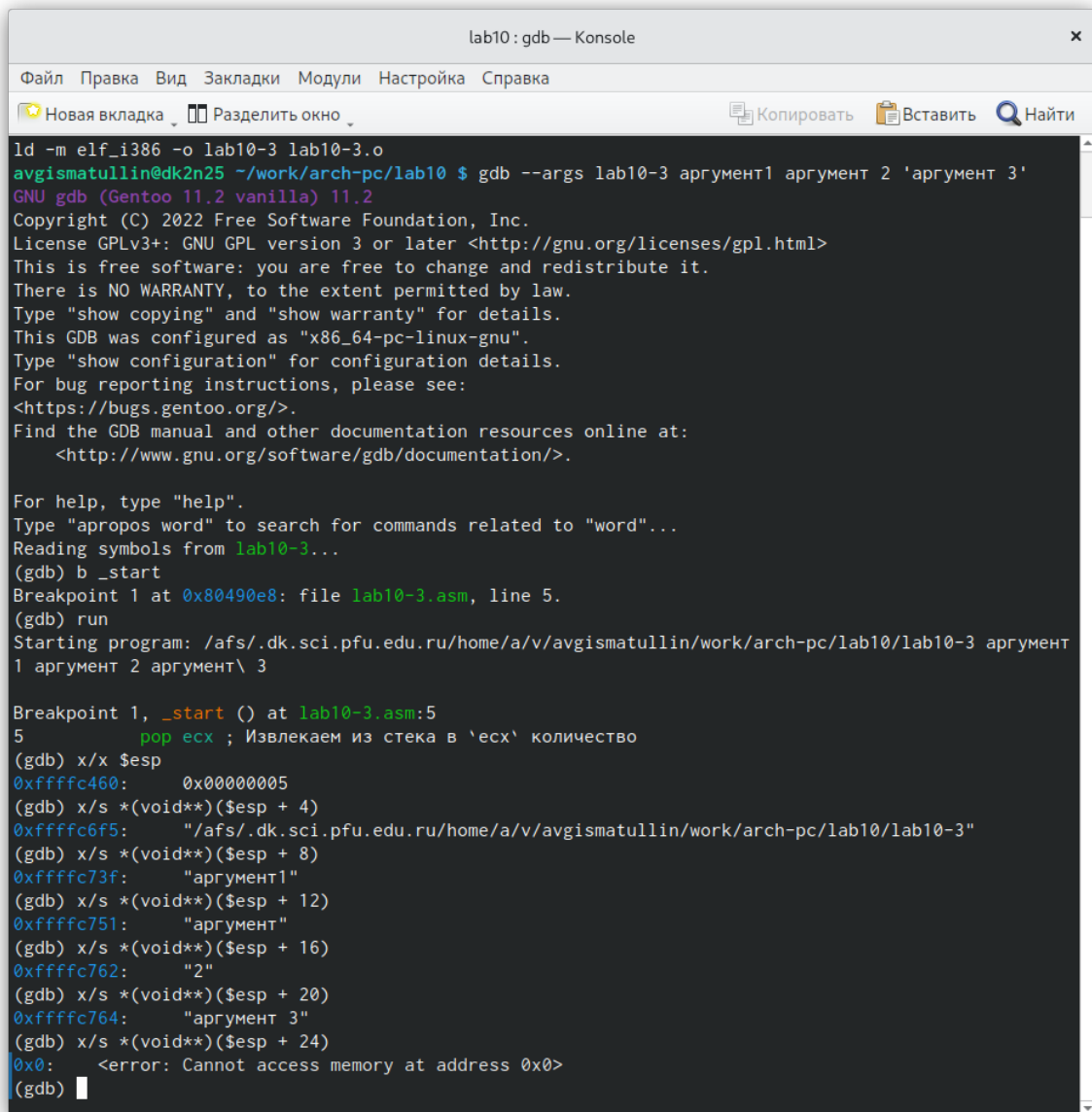
```
lab10: gdb — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
Новая вкладка  Разделить окно  Копировать  Вставить  Найти
avgismatullin@dk2n25 ~/work/arch-pc/lab10 $ mc
avgismatullin@dk2n25 ~/work/arch-pc/lab10 $ nasm -f elf -g -l lab10-3.lst lab10-3.asm
ld -m elf_i386 -o lab10-3 lab10-3.o
avgismatullin@dk2n25 ~/work/arch-pc/lab10 $ gdb --args lab10-3 аргумент1 аргумент 2 'а
ргумент 3'
GNU gdb (Gentoo 11.2 vanilla) 11.2
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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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 lab10-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab10-3.asm, line 5.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/v/avgismatullin/work/arch-pc/lab10/la
b10-3 аргумент1 аргумент 2 аргумент\ 3

Breakpoint 1, _start () at lab10-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffc460: 0x00000005
(gdb) |
```

Рис. 4.13: Командная строка. Запуск в gdb lab10-3.asm

9. С помощью команды `x/s *(void**)( $\$esp + n$ )` можно посмотреть, какие аргументы находятся в стеке: (рис. 4.14)



```
lab10: gdb — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
Новая вкладка  Разделить окно  Копировать  Вставить  Найти
ld -m elf_i386 -o lab10-3 lab10-3.o
avgismatullin@dk2n25 ~/work/arch-pc/lab10 $ gdb --args lab10-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Gentoo 11.2 vanilla) 11.2
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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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 lab10-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab10-3.asm, line 5.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/v/avgismatullin/work/arch-pc/lab10/lab10-3 аргумент
1 аргумент 2 аргумент\ 3

Breakpoint 1, _start () at lab10-3.asm:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb) x/x $esp
0xfffffc460:  0x00000005
(gdb) x/s *(void**)($esp + 4)
0xfffffc6f5:  "/afs/.dk.sci.pfu.edu.ru/home/a/v/avgismatullin/work/arch-pc/lab10/lab10-3"
(gdb) x/s *(void**)($esp + 8)
0xfffffc73f:  "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xfffffc751:  "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xfffffc762:  "2"
(gdb) x/s *(void**)($esp + 20)
0xfffffc764:  "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0:  <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 4.14: Командная строка. Просмотр содержимого очереди

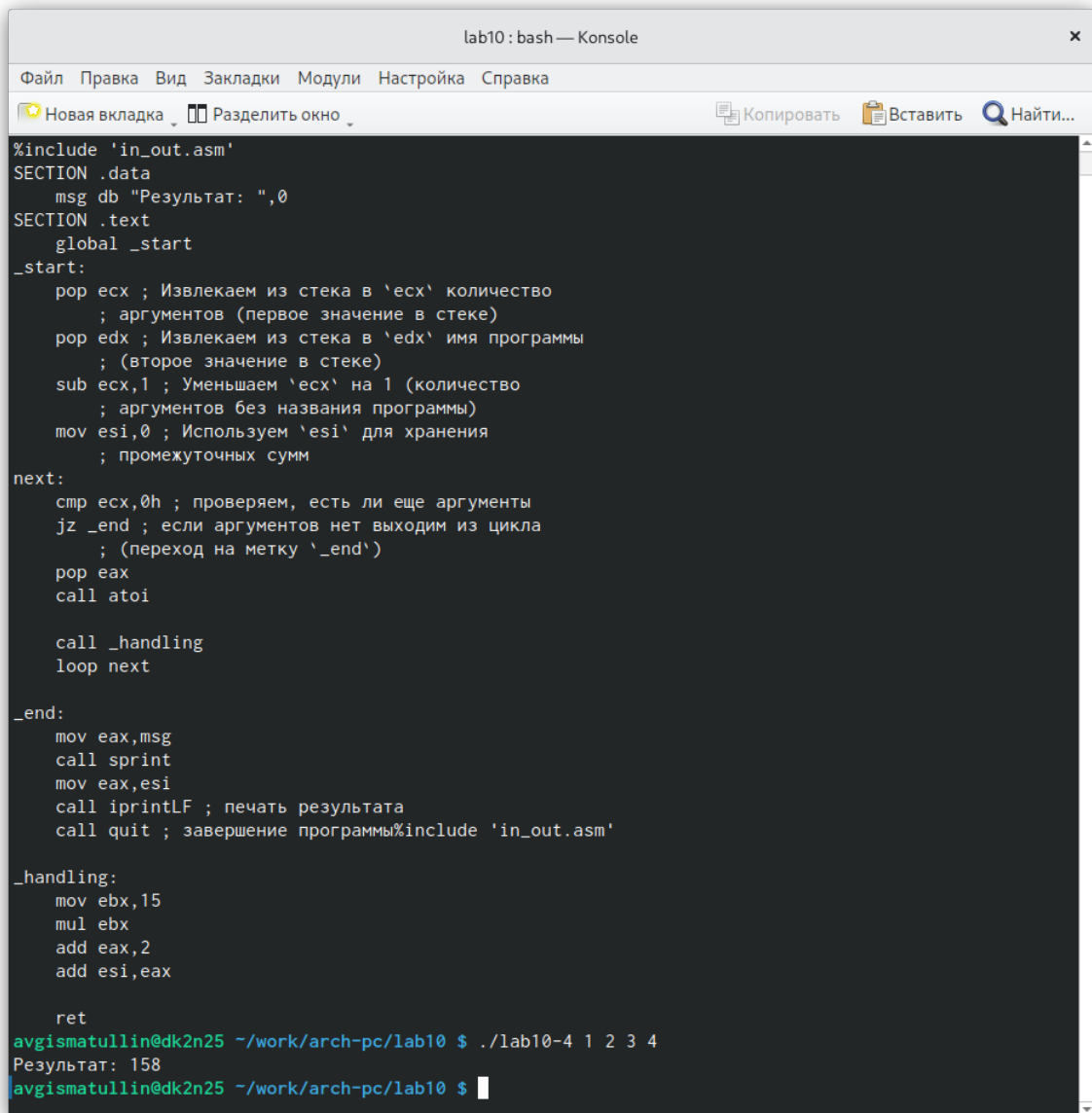
Примечательно следующее: адрес выглядит, как  $[\$esp + n]$ , где  $n$  кратно 4. Это вызвано тем, что  $n$  - число байт, на которое смещен элемент относительно вершины стека.



## 5 Выполнение задания для самостоятельной работы

Суть задания для самостоятельной работы заключается в том, чтобы преобразовать программу из лабораторной работы №9, реализовав вычисление значения функции как подпрограмму, а также изучить листинг 10.3 с помощью отладчика gdb и определить в нем ошибку.

1. Создадим файл lab10-4.asm для создания подпрограммы для функции. Скопируем программу из файла lab9-4.asm, цель которой была вычисление сумм значений функций  $f(x) = 15x + 2$ . Перепишем ее, добавив подпрограмму. Результат получился следующим: (рис. 5.1)



```
lab10: bash — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
Новая вкладка  Разделить окно  Копировать  Вставить  Найти...

%include 'in_out.asm'
SECTION .data
    msg db "Результат: ",0
SECTION .text
    global _start
_start:
    pop ecx ; Извлекаем из стека в 'ecx' количество
             ; аргументов (первое значение в стеке)
    pop edx ; Извлекаем из стека в 'edx' имя программы
             ; (второе значение в стеке)
    sub ecx,1 ; Уменьшаем 'ecx' на 1 (количество
             ; аргументов без названия программы)
    mov esi,0 ; Используем 'esi' для хранения
             ; промежуточных сумм
next:
    cmp ecx,0h ; проверяем, есть ли еще аргументы
    jz _end ; если аргументов нет выходим из цикла
             ; (переход на метку '_end')
    pop eax
    call atoi

    call _handling
    loop next

_end:
    mov eax,msg
    call sprint
    mov eax,esi
    call iprintLF ; печать результата
    call quit ; завершение программы%include 'in_out.asm'

_handling:
    mov ebx,15
    mul ebx
    add eax,2
    add esi,eax

    ret
avgismatullin@dk2n25 ~/work/arch-pc/lab10 $ ./lab10-4 1 2 3 4
Результат: 158
avgismatullin@dk2n25 ~/work/arch-pc/lab10 $
```

Рис. 5.1: Командная строка. Программа lab10-4.asm

Ответ получился верным.

2. Скопируем программу листинга 10.3 в файл lab10-5.asm, запустим его в gdb и проследим за изменением значений регистров. (рис. 5.2)

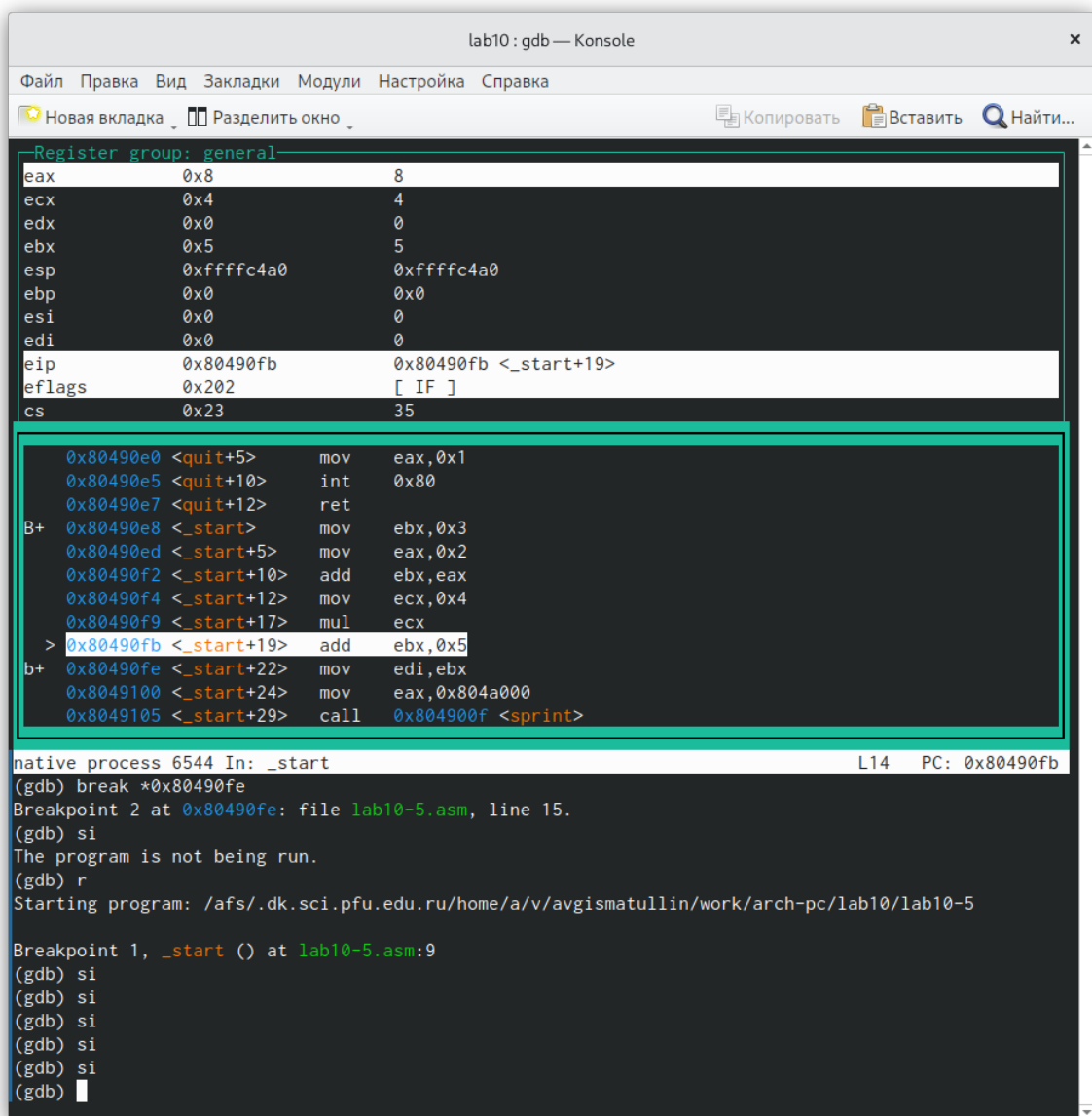


Рис. 5.2: Командная строка. Отладка программы lab10-5.asm

Здесь сразу можно обратить внимание, что значение из регистра `eax` прибавляется к `ebx`, а не наоборот. Ввиду этого на данный момент значение в `eax` равно 8. Проходим далее: (рис. 5.3)

The screenshot shows a GDB console window titled "lab10: gdb — Konsole". The menu bar includes "Файл", "Правка", "Вид", "Закладки", "Модули", "Настройка", and "Справка". The toolbar has buttons for "Новая вкладка", "Разделить окно", "Копировать", "Вставить", and "Найти...".

The "Register group: general" section displays the following register values:

Register	Value (Hex)	Value (Dec)
eax	0x804a000	134520832
ecx	0x4	4
edx	0x0	0
ebx	0x8	8
esp	0xffffc49c	0xffffc49c
ebp	0x0	0x0
esi	0x0	0
edi	0x8	8
eip	0x804900f	0x804900f <sprint>
eflags	0x202	[ IF ]
cs	0x23	35

Below the registers, a list of assembly instructions is shown, each with its address and operation:

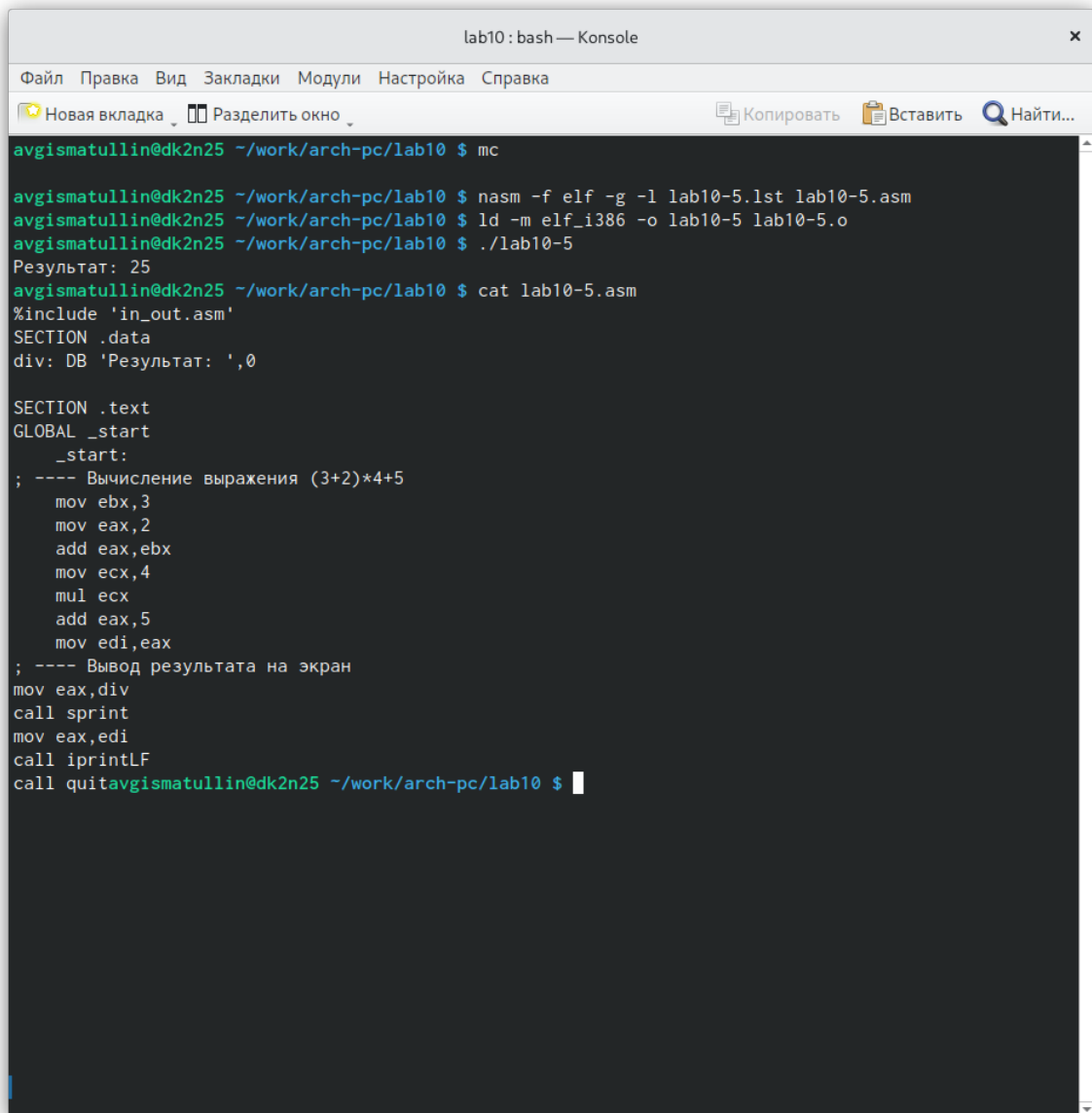
```
0x80491ec    add    BYTE PTR [eax],al
0x80491ee    add    BYTE PTR [eax],al
0x80491f0    add    BYTE PTR [eax],al
0x80491f2    add    BYTE PTR [eax],al
0x80491f4    add    BYTE PTR [eax],al
0x80491f6    add    BYTE PTR [eax],al
0x80491f8    add    BYTE PTR [eax],al
0x80491fa    add    BYTE PTR [eax],al
0x80491fc    add    BYTE PTR [eax],al
0x80491fe    add    BYTE PTR [eax],al
0x8049200    add    BYTE PTR [eax],al
0x8049202    add    BYTE PTR [eax],al
```

The status bar at the bottom indicates "native process 6670 In: sprint" and "L24 PC: 0x804900f". The command history shows the following commands:

```
(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/v/avgismatullin/work/arch-pc/lab10/lab10-5
Breakpoint 1, _start () at lab10-5.asm:9
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
```

Рис. 5.3: Командная строка. Отладка программы lab10-5.asm 2

Здесь тоже можно сделать вывод, что значения прибавляются не к тому регистру, а edi запоминает число из ebx, что тоже неверно. Основываясь на всех этих недочетах, я переписал программу следующим образом: (рис. 5.4)



```
lab10: bash — Konsole
Файл Правка Вид Закладки Модули Настройка Справка
Новая вкладка Разделить окно Копировать Вставить Найти...
avgismatullin@dk2n25 ~/work/arch-pc/lab10 $ mc
avgismatullin@dk2n25 ~/work/arch-pc/lab10 $ nasm -f elf -g -l lab10-5.lst lab10-5.asm
avgismatullin@dk2n25 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o lab10-5 lab10-5.o
avgismatullin@dk2n25 ~/work/arch-pc/lab10 $ ./lab10-5
Результат: 25
avgismatullin@dk2n25 ~/work/arch-pc/lab10 $ cat lab10-5.asm
%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 quitavgismatullin@dk2n25 ~/work/arch-pc/lab10 $
```

Рис. 5.4: Командная строка. Выполнение верной программы lab10-5.asm

## 6 Выводы

В ходе работы я приобрел навыки написания программ с использованием подпрограмм, познакомился с методами отладки при помощи GDB и его основными возможностями.

## **Список литературы**