

Отчёт по лабораторной работе №9

дисциплина: Архитектура компьютера

Баранова Анна Андреевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	10
4.1	Реализация подпрограмм в NASM	10
4.2	Отладка программ с помощью GDB	13
4.3	Задание для самостоятельной работы	25
5	Выводы	32

List of Figures

4.1	Создание каталога для программ лабораторной работы № 8 и создание в нём файла lab8-1.asm	10
4.2	Создание каталога для программ лабораторной работы № 8 и создание в нём файла lab8-1.asm	10
4.3	Копирование файла in_out.asm в каталог с файлом lab8-1.asm с помощью функциональной клавиши F5	11
4.4	Изменения в файле lab09-1.asm	11
4.5	Создание исполняемого файла и его запуск	12
4.6	Создание исполняемого файла и его запуск	12
4.7	Изменения в файле lab09-1.asm	12
4.8	Создание исполняемого файла и его запуск	13
4.9	Создание файла lab09-2.asm	13
4.10	Создание файла lab09-2.asm	13
4.11	Изменения в файле lab09-2.asm	14
4.12	Получение исполняемого файла	14
4.13	Загрузка исполняемого файла в отладчик gdb и проверка работы программы с помощью команды run	14
4.14	Установка брейкпоинта на метку _start, с которой начинается выполнение любой ассемблерной программы, и её запуск	15
4.15	Просмотр дисассимилированного кода программы с помощью команды disassemble начиная с метки _start	15
4.16	Переключение на отображение команд с Intel'овским синтаксисом	16
4.17	Включение режима псевдографики	16
4.18	Проверка установки точки останова	17
4.19	Установка точки останова	17
4.20	Просмотр информации о всех установленных точках останова	17
4.21	Выполнение инструкции с помощью команды stepi	18
4.22	Выполнение инструкции с помощью команды stepi	18
4.23	Выполнение инструкции с помощью команды stepi	19
4.24	Выполнение инструкции с помощью команды stepi	19
4.25	Выполнение инструкции с помощью команды stepi	20
4.26	Просмотр содержимого регистров с помощью команды info registers	20
4.27	Просмотр значения переменной msg1 по имени	21
4.28	Просмотр значения переменной msg2 по адресу	21
4.29	Изменение первого символа переменной msg1	21
4.30	Изменение первого символа переменной msg2	21
4.31	Вывод в различных форматах значения регистра edx	22

4.32	Изменение значения регистра ebx	23
4.33	Завершение выполнения программы и выход из GDB	23
4.34	Копирование файла lab8-2.asm в файл с именем lab09-3.asm и создание исполняемого файла	23
4.35	Загрузка исполняемого файла в отладчик	24
4.36	Установка точки останова перед первой инструкцией в программе и её запуск	24
4.37	Проверка адреса вершины стека	24
4.38	Просмотр всех позиций стека	25
4.39	Преобразование программы из лабораторной работы №8	25
4.40	Преобразование программы из лабораторной работы №8	26
4.41	Преобразование программы из лабораторной работы №8	27
4.42	Преобразование программы из лабораторной работы №8	27
4.43	Обнаружение ошибки в программе с помощью отладчика GDB и её исправление	27
4.44	Обнаружение ошибки в программе с помощью отладчика GDB и её исправление	28
4.45	Обнаружение ошибки в программе с помощью отладчика GDB и её исправление	28
4.46	Обнаружение ошибки в программе с помощью отладчика GDB и её исправление	28
4.47	Обнаружение ошибки в программе с помощью отладчика GDB и её исправление	29
4.48	Обнаружение ошибки в программе с помощью отладчика GDB и её исправление	29
4.49	Обнаружение ошибки в программе с помощью отладчика GDB и её исправление	30
4.50	Обнаружение ошибки в программе с помощью отладчика GDB и её исправление	30
4.51	Обнаружение ошибки в программе с помощью отладчика GDB и её исправление	31
4.52	Обнаружение ошибки в программе с помощью отладчика GDB и её исправление	31

1 Цель работы

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

2 Задание

В ходе выполнения данной лабораторной работы необходимо изучить:

- Понятие об отладке;
- методы отладки;
- основные возможности отладчика GDB;
- понятие подпрограммы.

Выполнив эту работу, мы приобретём навыки написания программ с использованием подпрограмм. Познакомимся с методами отладки при помощи GDB и его основными возможностями.

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.

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.

GDB (GNU Debugger — отладчик проекта GNU) [1] 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.

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.

4 Выполнение лабораторной работы

4.1 Реализация подпрограмм в NASM

Создадим каталог для выполнения лабораторной работы № 9, перейдём в него и создадим файл lab09-1.asm и также создадим копию файла in_out.asm (рис. 4.1), (рис. 4.2), (рис. 4.3).

```
aabaranova@dk8n69 ~ $ mkdir ~/work/arch-pc/lab09
aabaranova@dk8n69 ~ $ cd ~/work/arch-pc/lab09
aabaranova@dk8n69 ~/work/arch-pc/lab09 $ touch lab09-1.asm
aabaranova@dk8n69 ~/work/arch-pc/lab09 $
```

Рис. 4.1: Создание каталога для программ лабораторной работы № 8 и создание в нём файла lab8-1.asm

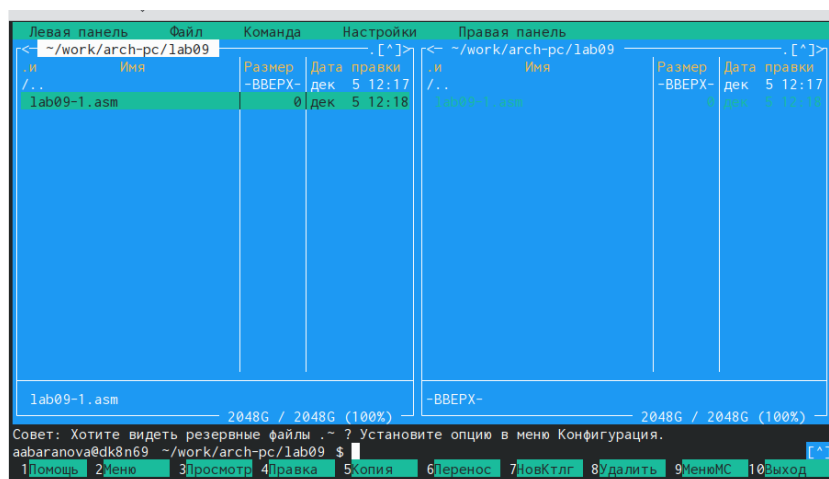


Рис. 4.2: Создание каталога для программ лабораторной работы № 8 и создание в нём файла lab8-1.asm

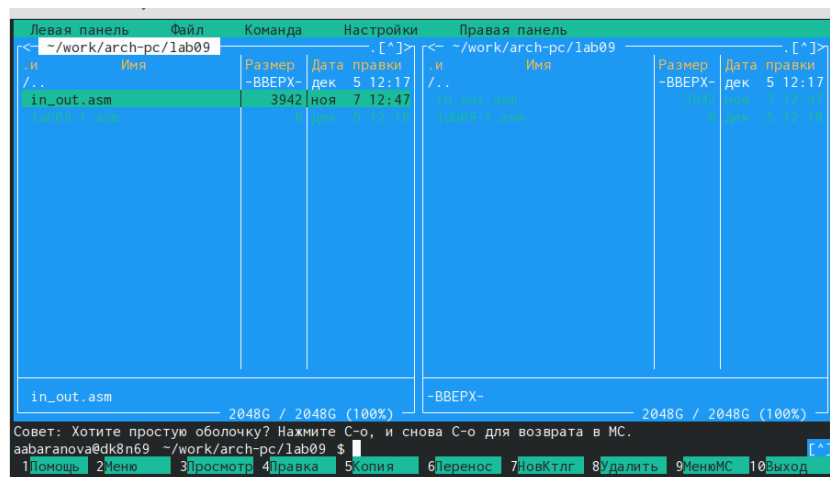


Рис. 4.3: Копирование файла in_out.asm в каталог с файлом lab8-1.asm с помощью функциональной клавиши F5

Введём в файл lab09-1.asm текст программы (рис. 4.4).

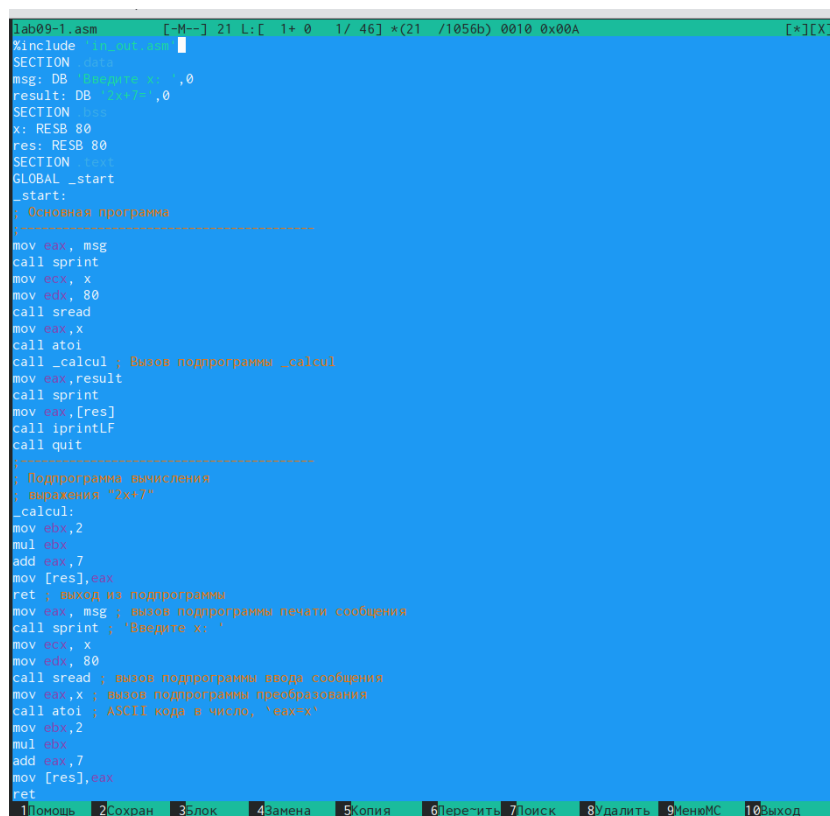


Рис. 4.4: Изменения в файле lab09-1.asm

Создадим исполняемый файл и запустим его (рис. 4.5), (рис. 4.6).

```

aabaranova@dk8n69 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
aabaranova@dk8n69 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
aabaranova@dk8n69 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 2
2x+7=11
aabaranova@dk8n69 ~/work/arch-pc/lab09 $

```

Рис. 4.5: Создание исполняемого файла и его запуск

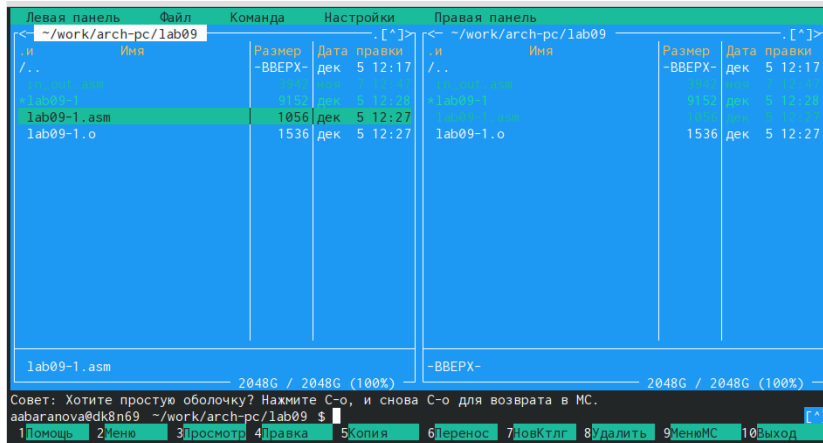


Рис. 4.6: Создание исполняемого файла и его запуск

Изменим текст программы файла lab09-1.asm (рис. 4.7).



Рис. 4.7: Изменения в файле lab09-1.asm

Создадим исполняемый файл и запустим его (рис. 4.8).

```

aabaranova@dk8n69 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
aabaranova@dk8n69 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
aabaranova@dk8n69 ~/work/arch-pc/lab09 $ ./lab09-1
f(x) = 2x+7
g(x) = 3x-1
Введите x: 2
f(g(x))= 17
aabaranova@dk8n69 ~/work/arch-pc/lab09 $

```

Рис. 4.8: Создание исполняемого файла и его запуск

4.2 Отладка программ с помощью GDB

Создадим файл lab09-2.asm в каталоге ~/work/arch-pc/lab09 (рис. 4.9), (рис. 4.10).

```

aabaranova@dk8n69 ~/work/arch-pc/lab09 $ touch lab09-2.asm
aabaranova@dk8n69 ~/work/arch-pc/lab09 $

```

Рис. 4.9: Создание файла lab09-2.asm

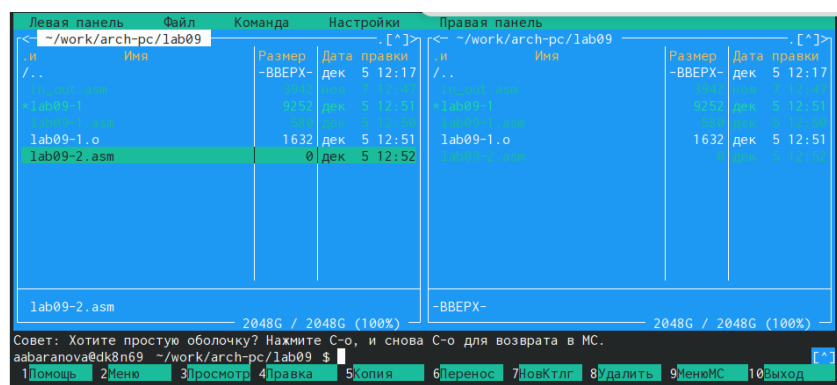


Рис. 4.10: Создание файла lab09-2.asm

Введём в файл lab09-2.asm текст программы (рис. 4.11).

```
lab09-2.asm [-M--] 8 L: [ 1+ 0 1/ 21] *(8 / 293b) 0046 0x02E [*][X]
SECTION .data
msg1: db "Hello, ",0x0
msg1Len: equ $ - msg1
msg2: db "world!",0xa
msg2Len: equ $ - msg2
SECTION .text
global _start
_start:
mov ecx, 4
mov ebx, 1
mov esi, msg1
mov edi, msg1Len
int 0x80
mov ecx, 4
mov ebx, 1
mov esi, msg2
mov edi, msg2Len
int 0x80
mov ecx, 1
mov ebx, 0
```

Рис. 4.11: Изменения в файле lab09-2.asm

Получим исполняемый файл (рис. 4.12).

```
aabaranova@dk8n69 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
aabaranova@dk8n69 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
```

Рис. 4.12: Получение исполняемого файла

Загрузим исполняемый файл в отладчик gdb, проверим работу программы, запустив её в оболочке GDB с помощью команды run (рис. 4.13).

```
aabaranova@dk8n69 ~/work/arch-pc/lab09 $ gdb lab09-2
GNU gdb (Gentoo 14.2 vanilla) 14.2
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-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 lab09-2...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/a/aabaranova/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 7310) exited normally]
(gdb) █
```

Рис. 4.13: Загрузка исполняемого файла в отладчик gdb и проверка работы программы с помощью команды run

Установим брейкпоинт на метку _start, с которой начинается выполнение любой ассемблерной программы, и запустим её (рис. 4.14).

```
(gdb) break _start
Breakpoint 1 at 0x08049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/a/aabaranova/work/arch-pc/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:9
█
```

Рис. 4.14: Установка брейкпоинта на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и её запуск

Посмотрим дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start` (рис. 4.15).

```
(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. 4.15: Просмотр дисассимилированного кода программы с помощью команды `disassemble` начиная с метки `_start`

Переключимся на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`. Отличие отображения синтаксиса машинных команд в режимах ATТ и Intel заключается в командах, в дисассимилированном отображении в командах используют `%` и `$`, а в Intel отображение эти символы не используются (рис. 4.16).

```

(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. 4.16: Переключение на отображение команд с Intel'овским синтаксисом

Включим режим псевдографики для более удобного анализа программы (рис. 4.17).

```

[ Register Values Unavailable ]

B>0x08049000 <_start>  mov     eax,0x4
0x08049005 <_start+5>  mov     ebx,0x1
0x0804900a <_start+10> mov     ecx,0x804a000
0x0804900f <_start+15> mov     edx,0x8
0x08049014 <_start+20> int     0x80
0x08049016 <_start+22> mov     eax,0x4
0x0804901b <_start+27> mov     ebx,0x1
0x08049020 <_start+32> mov     ecx,0x804a008
0x08049025 <_start+37> mov     edx,0x7

native process 7460 In: _start L9 PC: 0x08049000
(gdb) layout regs
(gdb) █

```

Рис. 4.17: Включение режима псевдографики

На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверим это с помощью команды `info breakpoints` (кратко `i b`) (рис.

4.18).

```
(gdb) info breakpoints
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000  lab09-2.asm:9
          breakpoint already hit 1 time
(gdb) █
```

Рис. 4.18: Проверка установки точки останова

Определим адрес предпоследней инструкции (`mov ebx,0x0`) и установим точку останова (рис. 4.19).

```
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) █
```

Рис. 4.19: Установка точки останова

Посмотрим информацию о всех установленных точках останова (рис. 4.20).

```
(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
(gdb) █
```

Рис. 4.20: Просмотр информации о всех установленных точках останова

Выполним 5 инструкций с помощью команды `stepi` (или `si`) и проследим за изменением значений регистров (рис. 4.21), (рис. 4.22), (рис. 4.23), (рис. 4.24), (рис. 4.25).

```

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc220 0xffffc220
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

B+ 0x8049000 <_start>    mov     eax,0x4
>0x8049005 <_start+5>    mov     ebx,0x1
0x804900a <_start+10>    mov     ecx,0x804a000
0x804900f <_start+15>    mov     edx,0x8
0x8049014 <_start+20>    int     0x80
0x8049016 <_start+22>    mov     eax,0x4
0x804901b <_start+27>    mov     ebx,0x1
0x8049020 <_start+32>    mov     ecx,0x804a008
0x8049025 <_start+37>    mov     edx,0x7

native process 7460 In: _start L10 PC: 0x8049005
breakpoint already hit 1 time
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(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
(gdb) si
(gdb)

```

Рис. 4.21: Выполнение инструкций с помощью команды stepi

```

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x1      1
esp      0xffffc220 0xffffc220
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

B+ 0x8049000 <_start>    mov     eax,0x4
0x8049005 <_start+5>    mov     ebx,0x1
>0x804900a <_start+10>    mov     ecx,0x804a000
0x804900f <_start+15>    mov     edx,0x8
0x8049014 <_start+20>    int     0x80
0x8049016 <_start+22>    mov     eax,0x4
0x804901b <_start+27>    mov     ebx,0x1
0x8049020 <_start+32>    mov     ecx,0x804a008
0x8049025 <_start+37>    mov     edx,0x7

native process 7460 In: _start L11 PC: 0x804900a
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(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
(gdb) si
(gdb) si
(gdb)

```

Рис. 4.22: Выполнение инструкций с помощью команды stepi

```

Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x0      0
ebx      0x1      1
esp      0xffffc220 0xffffc220
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

B+ 0x8049000 <_start>    mov     eax,0x4
0x8049005 <_start+5>    mov     ebx,0x1
0x804900a <_start+10>   mov     ecx,0x804a000
>0x804900f <_start+15>   mov     edx,0x8
0x8049014 <_start+20>   int     0x80
0x8049016 <_start+22>   mov     eax,0x4
0x804901b <_start+27>   mov     ebx,0x1
0x8049020 <_start+32>   mov     ecx,0x804a008
0x8049025 <_start+37>   mov     edx,0x7

native process 7460 In: _start L12 PC: 0x804900f
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(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
(gdb) s1
(gdb) s1
(gdb) s1
(gdb)

```

Рис. 4.23: Выполнение инструкций с помощью команды stepi

```

Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc220 0xffffc220
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

B+ 0x8049000 <_start>    mov     eax,0x4
0x8049005 <_start+5>    mov     ebx,0x1
0x804900a <_start+10>   mov     ecx,0x804a000
0x804900f <_start+15>   mov     edx,0x8
>0x8049014 <_start+20>   int     0x80
0x8049016 <_start+22>   mov     eax,0x4
0x804901b <_start+27>   mov     ebx,0x1
0x8049020 <_start+32>   mov     ecx,0x804a008
0x8049025 <_start+37>   mov     edx,0x7

native process 7460 In: _start L13 PC: 0x8049014
(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
(gdb) s1
(gdb) s1
(gdb) s1
(gdb) s1
(gdb)

```

Рис. 4.24: Выполнение инструкций с помощью команды stepi

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc220 0xffffc220
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

B+ 0x8049000 <_start>    mov     eax,0x4
0x8049005 <_start+5>    mov     ebx,0x1
0x804900a <_start+10>   mov     ecx,0x804a000
0x804900f <_start+15>   mov     edx,0x8
0x8049014 <_start+20>   int     0x80
>0x8049016 <_start+22>   mov     eax,0x4
0x804901b <_start+27>   mov     ebx,0x1
0x8049020 <_start+32>   mov     ecx,0x804a008
0x8049025 <_start+37>   mov     edx,0x7

native process 7460 In: _start L14 PC: 0x8049016
Num  Type      Disp Enb Address  What
1    breakpoint keep y 0x8049000 lab09-2.asm:9
      breakpoint already hit 1 time
2    breakpoint keep y 0x8049031 lab09-2.asm:20
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 4.25: Выполнение инструкций с помощью команды stepi

Посмотрим содержимое регистров с помощью команды info registers (или i r) (рис. 4.26).

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc220 0xffffc220
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

B+ 0x8049000 <_start>    mov     eax,0x4
0x8049005 <_start+5>    mov     ebx,0x1
0x804900a <_start+10>   mov     ecx,0x804a000
0x804900f <_start+15>   mov     edx,0x8
0x8049014 <_start+20>   int     0x80
>0x8049016 <_start+22>   mov     eax,0x4
0x804901b <_start+27>   mov     ebx,0x1
0x8049020 <_start+32>   mov     ecx,0x804a008
0x8049025 <_start+37>   mov     edx,0x7

native process 7460 In: _start L14 PC: 0x8049016
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc220 0xffffc220
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 4.26: Просмотр содержимого регистров с помощью команды info registers

Посмотрим значение переменной msg1 по имени (рис. 4.27).

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

Рис. 4.27: Просмотр значения переменной msg1 по имени

Посмотрим значение переменной msg2 по адресу (рис. 4.28).

```
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb) █
```

Рис. 4.28: Просмотр значения переменной msg2 по адресу

Изменим первый символ переменной msg1 (рис. 4.29).

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

Рис. 4.29: Изменение первого символа переменной msg1

Изменим первый символ переменной msg2 (рис. 4.30).

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

Рис. 4.30: Изменение первого символа переменной msg2

Выведем в шестнадцатеричном формате, в двоичном формате и в символьном виде значение регистра edx (рис. 4.31).

```
(gdb) p/s $edx
$5 = 8
(gdb) p/t $edx
$6 = 1000
(gdb) p/x $edx
$7 = 0x8
(gdb) █
```

Рис. 4.31: Вывод в различных форматах значения регистра `edx`

С помощью команды `set` изменим значение регистра `ebx`. Команда выводит два разных значения так как в первый раз вносим значение 2, а во второй раз регистр равен двум, поэтому и значения разные (рис. 4.32).

```

(gdb) set $ebx='2'
(gdb) p/s $ebx
$8 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$9 = 2
(gdb)

```

Рис. 4.32: Изменение значения регистра ebx

Завершим выполнение программы с помощью команды `stepi` (сокращенно `si`) и выйдем из GDB с помощью команды `quit` (сокращенно `q`) (рис. 4.33).

```

(gdb) si
(gdb) quit

```

Рис. 4.33: Завершение выполнения программы и выход из GDB

Скопируем файл `lab8-2.asm`, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки в файл с именем `lab09-3.asm` и создадим исполняемый файл (рис. 4.34).

```

aabaranova@dk8n69 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
aabaranova@dk8n69 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
aabaranova@dk8n69 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
aabaranova@dk8n69 ~/work/arch-pc/lab09 $

```

Рис. 4.34: Копирование файла `lab8-2.asm` в файл с именем `lab09-3.asm` и создание исполняемого файла

Загрузим исполняемый файл в отладчик, указав аргументы (рис. 4.35).

```
aabaranova@dk8n69 ~/work/arch-pc/lab09 $ gdb --args lab09-3 аргумент1 аргумент2 'аргумент3'
GNU gdb (Gentoo 14.2 vanilla) 14.2
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-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 lab09-3...
(gdb)
```

Рис. 4.35: Загрузка исполняемого файла в отладчик

Установим точку останова перед первой инструкцией в программе и запустим её (рис. 4.36).

```
(gdb) b _start
Breakpoint 1 at 0x00490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/aabaranova/work/arch-pc/lab09/lab09-3 аргумент1 аргумент2 аргумент\ 3
Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb)
```

Рис. 4.36: Установка точки останова перед первой инструкцией в программе и её запуск

Проверим адрес вершины стека и убедимся, что там хранится 5 элементов (рис. 4.37).

```
(gdb) x/x $esp
0xffffc1d0:      0x00000005
(gdb)
```

Рис. 4.37: Проверка адреса вершины стека

Посмотрим все позиции стека. По первому адресу хранится адрес, в остальных адресах хранятся элементы. Элементы расположены с интервалом в 4 единицы, так как стек может хранить до 4 байт, и для того чтобы данные сохранялись нормально и без помех, компьютер использует новый стек для новой информации (рис. 4.38).


```

(gdb) x/s *(void**)(esp + 4)
0xffffc46e: "/afs/.dk.sci.pfu.edu.ru/home/a/a/aabaranova/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffc4b5: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffc4c7: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffc4d8: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffc4da: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 4.38: Просмотр всех позиций стека

4.3 Задание для самостоятельной работы

Преобразуем программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму (рис. 4.39), (рис. 4.40), (рис. 4.41), (рис. 4.42).

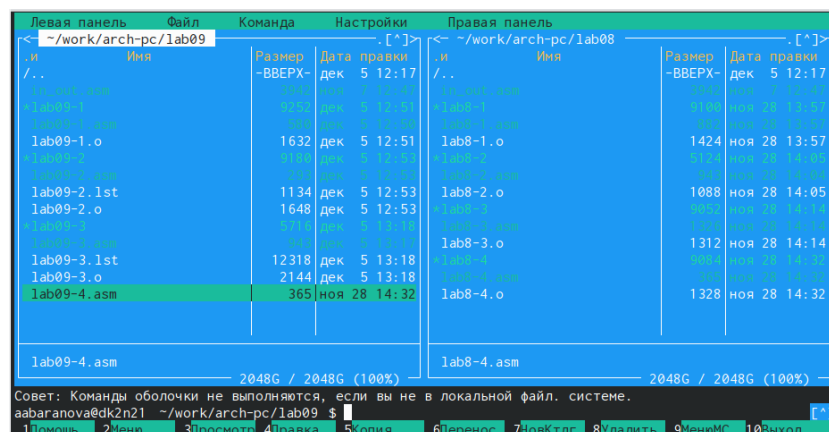


Рис. 4.39: Преобразование программы из лабораторной работы №8

```

lab09-4.asm      [-M--]  3  L:
#include 'in_out.asm'

SECTION .data
prim DB "f(x)=15x-9",0
msg db "Результат: ",0

SECTION .text
global _start

_start:
pop ecx.

pop edx.

sub ecx,1.

mov esi,0

mov eax,prim
call sprintf
next:
cmp ecx, 0
jz _end

pop eax
call atoi
call fir
add esi,eax

loop next

_end:
mov eax,msg
call sprintf
mov eax,esi
call iprintLF
call quit

fir:
mov ebx,15
mul ebx
sub eax,9
ret

```

1 Помощь 2 Сохран 3 Блок

Рис. 4.40: Преобразование программы из лабораторной работы №8

```

aabaranova@dk2n21 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-4.lst lab09-4.asm
aabaranova@dk2n21 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-4 lab09-4.o
aabaranova@dk2n21 ~/work/arch-pc/lab09 $ ./lab09-4 1 2
f(x)=15x-9
Результат: 27
aabaranova@dk2n21 ~/work/arch-pc/lab09 $ ./lab09-4 1 2 3
f(x)=15x-9
Результат: 63
aabaranova@dk2n21 ~/work/arch-pc/lab09 $

```

Рис. 4.41: Преобразование программы из лабораторной работы №8

Левая панель				Правая панель			
Файл		Команда		Файл		Команда	
Имя	Размер	Дата	Правки	Имя	Размер	Дата	Правки
lab09-1.asm	1632	дек 5 12:51		lab09-1.asm	1632	дек 5 12:51	
lab09-1.o	1632	дек 5 12:51		lab09-1.o	1632	дек 5 12:51	
lab09-2.lst	1134	дек 5 12:53		lab09-2.lst	1134	дек 5 12:53	
lab09-2.o	1648	дек 5 12:53		lab09-2.o	1648	дек 5 12:53	
lab09-3.asm	12318	дек 5 13:18		lab09-3.asm	12318	дек 5 13:18	
lab09-3.lst	2144	дек 5 13:18		lab09-3.lst	2144	дек 5 13:18	
lab09-3.o	12786	дек 5 14:22		lab09-3.o	12786	дек 5 14:22	
lab09-4.asm	2448	дек 5 14:22		lab09-4.asm	2448	дек 5 14:22	
lab09-4.lst				lab09-4.lst			
lab09-4.o				lab09-4.o			

Рис. 4.42: Преобразование программы из лабораторной работы №8

В листинге 9.3 приведена программа вычисления выражения $(3 + 2) \cdot 4 + 5$. При запуске данная программа даёт неверный результат. Проверим это. С помощью отладчика GDB, анализируя изменения значений регистров, определим ошибку и исправим её (рис. 4.43), (рис. 4.44), (рис. 4.45), (рис. 4.46), рис. 4.47), рис. 4.48), рис. 4.49), рис. 4.50), рис. 4.51), рис. 4.52).

```

aabaranova@dk2n21 ~/work/arch-pc/lab09 $ touch lab09-5.asm
aabaranova@dk2n21 ~/work/arch-pc/lab09 $

```

Рис. 4.43: Обнаружение ошибки в программе с помощью отладчика GDB и её исправление

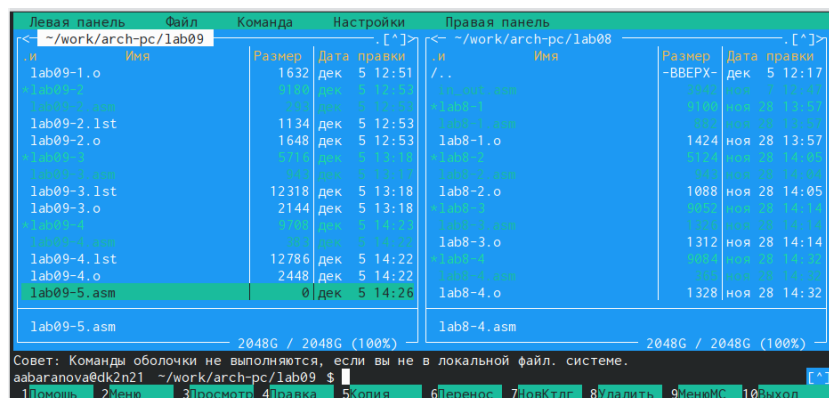


Рис. 4.44: Обнаружение ошибки в программе с помощью отладчика GDB и её исправление

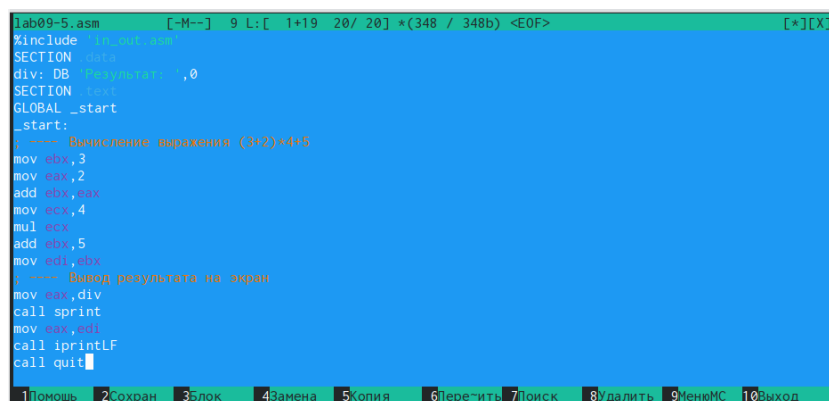


Рис. 4.45: Обнаружение ошибки в программе с помощью отладчика GDB и её исправление

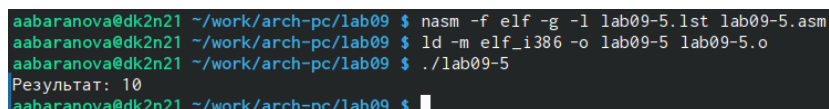


Рис. 4.46: Обнаружение ошибки в программе с помощью отладчика GDB и её исправление

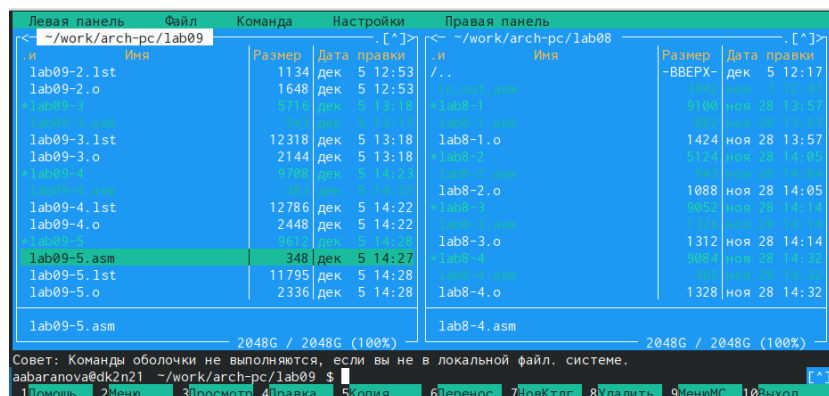


Рис. 4.47: Обнаружение ошибки в программе с помощью отладчика GDB и её исправление

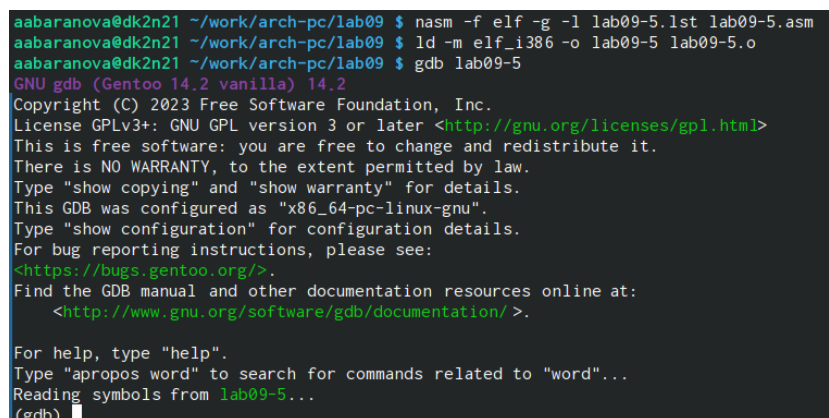


Рис. 4.48: Обнаружение ошибки в программе с помощью отладчика GDB и её исправление

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-5.asm, line 8.
(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/a/aabaranova/work/arch-pc/lab09/lab09-5

Breakpoint 1, _start () at lab09-5.asm:8
8      mov ebx,3
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x080490e8 <+0>:      mov     ebx,0x3
0x080490ed <+5>:      mov     eax,0x2
0x080490f2 <+10>:     add     ebx,eax
0x080490f4 <+12>:     mov     ecx,0x4
0x080490f9 <+17>:     mul     ecx
0x080490fb <+19>:     add     ebx,0x5
0x080490fe <+22>:     mov     edi,ebx
0x08049100 <+24>:     mov     eax,0x804a000
0x08049105 <+29>:     call   0x804900f <sprint>
0x0804910a <+34>:     mov     eax,edi
0x0804910c <+36>:     call   0x8049086 <iprintLF>
0x08049111 <+41>:     call   0x80490db <quit>
End of assembler dump.
(gdb) layout asm

```

Рис. 4.49: Обнаружение ошибки в программе с помощью отладчика GDB и её исправление

```

[ Register Values Unavailable ]

B+>0x80490e8 <_start>      mov     ebx,0x3
0x80490ed <_start+5>      mov     eax,0x2
0x80490f2 <_start+10>     add     eax,ebx
0x80490f4 <_start+12>     mov     ecx,0x4
0x80490f9 <_start+17>     mul     ecx
0x80490fb <_start+19>     add     eax,0x5
0x80490fe <_start+22>     mov     edi,eax
0x8049100 <_start+24>     mov     eax,0x804a000
0x8049105 <_start+29>     call   0x804900f <sprint>
0x804910a <_start+34>     mov     eax,edi
0x804910c <_start+36>     call   0x8049086 <iprintLF>
0x8049111 <_start+41>     call   0x80490db <quit>
0x8049116                      add     BYTE PTR [eax],al
0x8049118                      add     BYTE PTR [eax],al

native process 10423 In: _start
(gdb) layout regs
(gdb)

```

Рис. 4.50: Обнаружение ошибки в программе с помощью отладчика GDB и её исправление

```

Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffc220 0xffffc220
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490fb 0x80490fb <_start+19>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
es       0x2b     43

0x80490e0 <quit+5>    mov     eax,0x1
0x80490e5 <quit+10>   int     0x80
0x80490e7 <quit+12>   ret
B+ 0x80490e8 <_start>   mov     ebx,0x3
0x80490ed <_start+5>   mov     eax,0x2
0x80490f2 <_start+10>  add     ebx,eax
0x80490f4 <_start+12>  mov     ecx,0x4
0x80490f9 <_start+17>  mul     ecx
>0x80490fb <_start+19> add     ebx,0x5
0x80490fe <_start+22>  mov     edi,ebx
0x8049100 <_start+24>  mov     eax,0x804a000
0x8049105 <_start+29>  call   0x804900f <sprint>
0x804910a <_start+34>  mov     eax,edi
0x804910c <_start+36>  call   0x8049086 <iprintf>

native process 7538 In: _start          L13  PC: 0x80490fb
(gdb) layout regs
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 4.51: Обнаружение ошибки в программе с помощью отладчика GDB и её исправление

```

aabaranova@dk2n21 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-5.lst lab09-5.asm
aabaranova@dk2n21 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-5 lab09-5.o
aabaranova@dk2n21 ~/work/arch-pc/lab09 $ gdb lab09-5
GNU gdb (Gentoo 14.2 vanilla) 14.2
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-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 lab09-5...
(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/a/aabaranova/work/arch-pc/lab09/lab09-5
Результат: 25
[Inferior 1 (process 10735) exited normally]
(gdb)

```

Рис. 4.52: Обнаружение ошибки в программе с помощью отладчика GDB и её исправление

5 Выводы

В ходе выполнения данной лабораторной работы были приобретены навыки написания программ с использованием подпрограмм. Также познакомились с методами отладки при помощи GDB и его основными возможностями.