

# **Отчёт по лабораторной работе №9**

**Дисциплина: архитектура компьютеров и операционные системы**

Шибаета Алесандра Алексеевна

# 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>10</b>
4.1	Реализация подпрограмм в NASM . . . . .	10
4.2	Отладка программ с помощью GDB . . . . .	12
4.2.1	Добавление точек останова . . . . .	15
4.2.2	Работа с данными программы в GDB . . . . .	16
4.2.3	Обработка аргументов командной строки в GDB . . . . .	21
4.3	Задания для самостоятельной работы . . . . .	23
<b>5</b>	<b>Выводы</b>	<b>29</b>
<b>6</b>	<b>Список литературы</b>	<b>30</b>

## Список иллюстраций

4.1	Создание файлов для лабораторной работы . . . . .	10
4.2	Запуск исполняемого файла . . . . .	10
4.3	Изменение текста программы согласно заданию . . . . .	11
4.4	Запуск исполняемого файла . . . . .	11
4.5	Ввод текста программы из листинга 9.2 . . . . .	12
4.6	Получение исполняемого файла . . . . .	12
4.7	Загрузка исполняемого файла в отладчик . . . . .	13
4.8	Проверка работы файла с помощью команды run . . . . .	13
4.9	Установка брейкпоинта и запуск программы . . . . .	13
4.10	Использование команд disassemble и disassembly-flavor intel . . .	14
4.11	Включение режима псевдографики . . . . .	15
4.12	Установление точек останова и просмотр информации о них . . .	16
4.13	До использования команды stepi . . . . .	17
4.14	После использования команды stepi . . . . .	17
4.15	Просмотр значений переменных . . . . .	18
4.16	Использование команды set . . . . .	18
4.17	Вывод значения регистра в разных представлениях . . . . .	19
4.18	Использование команды set для изменения значения регистра . .	20
4.19	Завершение работы GDB . . . . .	21
4.20	Создание файла . . . . .	21
4.21	Загрузка файла с аргументами в отладчик . . . . .	22
4.22	Установление точки останова и запуск программы . . . . .	22
4.23	Просмотр значений, введенных в стек . . . . .	22
4.24	Написание кода подпрограммы . . . . .	23
4.25	Запуск программы и проверка его вывода . . . . .	23
4.26	Ввод текста программы из листинга 9.3 . . . . .	25
4.27	Создание и запуск исполняемого файла . . . . .	25
4.28	Нахождение причины ошибки . . . . .	26
4.29	Неверное изменение регистра . . . . .	26
4.30	Исправление ошибки . . . . .	27
4.31	Ошибка исправлена . . . . .	27

## 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. Реализация подпрограмм в NASM.
2. Отладка программ с помощью GDB.
3. Добавление точек останова.
4. Работа с данными программы в GDB.
5. Обработка аргументов командной строки в GDB.
6. Задания для самостоятельной работы.

### 3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. 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.

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.

Команда `run` (сокращённо `r`) — запускает отлаживаемую программу в оболочке GDB.

Команда `kill` (сокращённо `k`) прекращает отладку программы, после чего следует вопрос о прекращении процесса отладки. Если в ответ введено `y` (то есть «да»), отладка программы прекращается. Командой `run` её можно начать

заново, при этом все точки останова (breakpoints), точки просмотра (watchpoints) и точки отлова (catchpoints) сохраняются.

Для выхода из отладчика используется команда quit (или сокращённо q).

Если есть файл с исходным текстом программы, а в исполняемый файл включена информация о номерах строк исходного кода, то программу можно отлаживать, работая в отладчике непосредственно с её исходным текстом. Чтобы программу можно было отлаживать на уровне строк исходного кода, она должна быть откомпилирована с ключом -g.

Установить точку останова можно командой break (кратко b). Типичный аргумент этой команды — место установки. Его можно задать как имя метки или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка».

Информацию о всех установленных точках останова можно вывести командой info (кратко i).

Для того чтобы сделать неактивной какую-нибудь ненужную точку останова, можно воспользоваться командой disable.

Обратно точка останова активируется командой enable.

Если же точка останова в дальнейшем больше не нужна, она может быть удалена с помощью команды delete.

Для продолжения остановленной программы используется команда continue (c). Выполнение программы будет происходить до следующей точки останова. В качестве аргумента может использоваться целое число N, которое указывает отладчику проигнорировать N – 1 точку останова (выполнение остановится на N-й точке).

Команда stepi (кратко sl) позволяет выполнять программу по шагам, т.е. данная команда выполняет ровно одну инструкцию.

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

Создаю каталог для выполнения лабораторной работы № 9, перехожу в него и создаю файл lab9-1.asm. (рис. 4.1)

```
[aashibaeva@fedora ~]$ mkdir ~/work/study/2023-2024/'Архитектура компьютера'/arch-pc/lab09
[aashibaeva@fedora ~]$ cd ~/work/study/2023-2024/'Архитектура компьютера'/arch-pc/lab09
[aashibaeva@fedora lab09]$ touch lab9-1.asm
[aashibaeva@fedora lab09]$
```

Рис. 4.1: Создание файлов для лабораторной работы

Ввожу в файл lab09-1.asm текст программы с использованием подпрограммы из листинга 9.1. (рис. ??)

![Ввод текста программы из листинга 9.1](image/2.jpg{#fig:002 width=70%})

Создаю исполняемый файл и проверяю его работу. (рис. 4.2)

```
[aashibaeva@fedora lab09]$ nasm -f elf lab9-1.asm
[aashibaeva@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[aashibaeva@fedora lab09]$
[aashibaeva@fedora lab09]$ ./lab9-1
Введите x: 4
2x+7=15
[aashibaeva@fedora lab09]$
```

Рис. 4.2: Запуск исполняемого файла

Изменяю текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul` для вычисления выражения  $f(g(x))$ , где  $x$  вводится с клавиатуры,  $f(x) = 2x + 7$ ,  $g(x) = 3x - 1$ . (рис. 4.3)

```
lab9-1.asm      [----]  3 L: [ 17+23  40/ 40] *(754 / 754b) <EOF>
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:
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы
_subcalcul:
mov ebx, 3
mul ebx
add eax, -1
ret
1 Помощь 2 Сохранить 3 Блок 4 Замена 5 Копия 6 Переименовать 7 Поиск 8 Удалить
```

Рис. 4.3: Изменение текста программы согласно заданию

Создаю исполняемый файл и проверяю его работу. (рис. 4.4)

```
2x+7=15
[aashibaeva@fedora lab09]$ mc

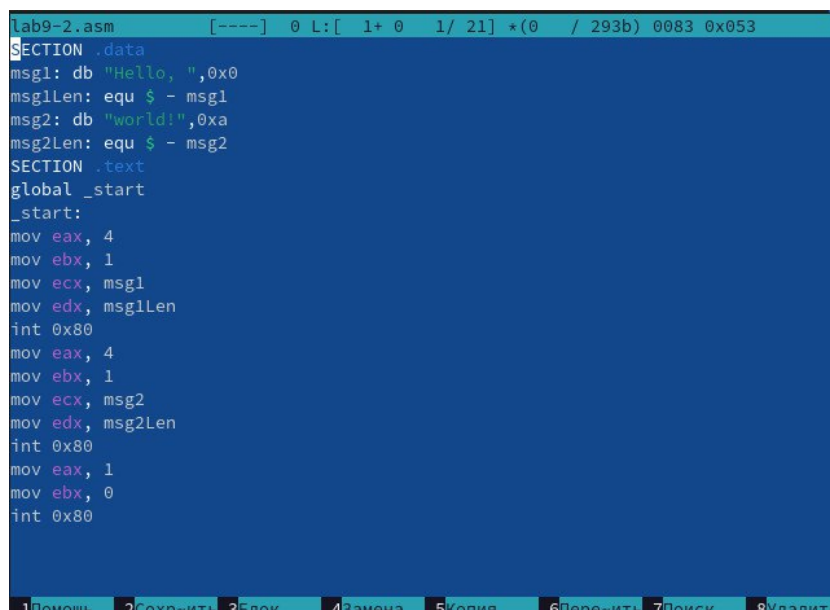
[aashibaeva@fedora lab09]$ nasm -f elf lab9-1.asm
[aashibaeva@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[aashibaeva@fedora lab09]$ ./lab9-1
Введите x: 4
2x+7=15
[aashibaeva@fedora lab09]$ nasm -f elf lab9-1.asm
[aashibaeva@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[aashibaeva@fedora lab09]$ ./lab9-1
Введите x: 7
2x+7=21
[aashibaeva@fedora lab09]$ mc

[aashibaeva@fedora lab09]$ nasm -f elf lab9-1.asm
[aashibaeva@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[aashibaeva@fedora lab09]$ ./lab9-1
Введите x: 7
2x+7=47
[aashibaeva@fedora lab09]$ nasm -f elf lab9-1.asm
[aashibaeva@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[aashibaeva@fedora lab09]$ ./lab9-1
Введите x: 4
2x+7=29
[aashibaeva@fedora lab09]$
```

Рис. 4.4: Запуск исполняемого файла

## 4.2 Отладка программ с помощью GDB

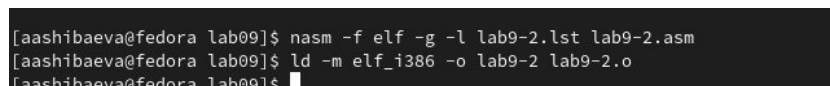
Создаю файл lab9-2.asm с текстом программы из Листинга 9.2. (рис. 4.5)



```
lab9-2.asm      [----]  0 L: [ 1+ 0  1/ 21] *(0  / 293b) 0083 0x053
SECTION .data
msg1: db "Hello, ",0x0
msg1len: equ $ - msg1
msg2: db "world!",0xa
msg2len: equ $ - msg2
SECTION .text
global _start
_start:
mov eax, 4
mov ebx, 1
mov ecx, msg1
mov edx, msg1len
int 0x80
mov eax, 4
mov ebx, 1
mov ecx, msg2
mov edx, msg2len
int 0x80
mov eax, 1
mov ebx, 0
int 0x80
```

Рис. 4.5: Ввод текста программы из листинга 9.2

Получаю исполняемый файл для работы с GDB с ключом ‘-g’. (рис. 4.6)



```
[aashibaeva@fedora lab09]$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
[aashibaeva@fedora lab09]$ ld -m elf_i386 -o lab9-2 lab9-2.o
[aashibaeva@fedora lab09]$
```

Рис. 4.6: Получение исполняемого файла

Загружаю исполняемый файл в отладчик gdb. (рис. 4.7)

```
aashibaeva@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 ...
aashibaeva... x aashibaeva... x aashibaeva... x aashibaeva... x aashibaeva... x
[aashibaeva@fedora report]$ cd .
[aashibaeva@fedora report]$ cd .
bash: cd: слишком много аргументов
[aashibaeva@fedora report]$ cd ..
[aashibaeva@fedora lab08]$ cd ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09
[aashibaeva@fedora lab09]$ gdb lab9-2
GNU gdb (GDB) Fedora Linux 13.2-4.fc38
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab9-2...
(gdb)
```

Рис. 4.7: Загрузка исполняемого файла в отладчик

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды `run`. (рис. 4.8)

```
(gdb) run
Starting program: /home/aashibaeva/work/study/2023-2024/Архитектура компьютера/arch-pc/
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 345648) exited normally]
(gdb)
```

Рис. 4.8: Проверка работы файла с помощью команды `run`

Для более подробного анализа программы устанавливаю брейкпоинт на метку `_start` и запускаю её. (рис. 4.9)

```
(gdb) break _start
Breakpoint 1 at 0x4010e0: file lab9-2.asm, line 9.
(gdb) run
Starting program: /home/aashibaeva/work/study/2023-2024/Архитектура компьютера/arch-pc/
lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:9
9      mov eax, 4
(gdb)
```

Рис. 4.9: Установка брейкпоинта и запуск программы

Просматриваю дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start`, и переключаюсь на отображение команд с синтаксисом Intel, введя команду `set disassembly-flavor intel`. (рис. 4.10)

```
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x004010e0 <+0>:    mov     $0x4,%eax
      0x004010e5 <+5>:    mov     $0x1,%ebx
      0x004010ea <+10>:   mov     $0x402118,%ecx
      0x004010ef <+15>:   mov     $0x8,%edx
      0x004010f4 <+20>:   int     $0x80
      0x004010f6 <+22>:   mov     $0x4,%eax
      0x004010fb <+27>:   mov     $0x1,%ebx
      0x00401100 <+32>:   mov     $0x402120,%ecx
      0x00401105 <+37>:   mov     $0x7,%edx
      0x0040110a <+42>:   int     $0x80
      0x0040110c <+44>:   mov     $0x1,%eax
      0x00401111 <+49>:   mov     $0x0,%ebx
      0x00401116 <+54>:   int     $0x80
End of assembler dump.
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x004010e0 <+0>:    mov     eax,0x4
      0x004010e5 <+5>:    mov     ebx,0x1
      0x004010ea <+10>:   mov     ecx,0x402118
      0x004010ef <+15>:   mov     edx,0x8
      0x004010f4 <+20>:   int     0x80
      0x004010f6 <+22>:   mov     eax,0x4
      0x004010fb <+27>:   mov     ebx,0x1
      0x00401100 <+32>:   mov     ecx,0x402120
      0x00401105 <+37>:   mov     edx,0x7
      0x0040110a <+42>:   int     0x80
      0x0040110c <+44>:   mov     eax,0x1
      0x00401111 <+49>:   mov     ebx,0x0
      0x00401116 <+54>:   int     0x80
End of assembler dump.
(gdb)
```

Рис. 4.10: Использование команд `disassemble` и `disassembly-flavor intel`

В режиме АТТ имена регистров начинаются с символа `%`, а имена операндов с `$`, в то время как в Intel используется привычный нам синтаксис.

Включаю режим псевдографики для более удобного анализа программы с помощью команд `layout asm` и `layout regs`. (рис. 4.11)

```
[ Register Values Unavailable ]

0x401479      add     BYTE PTR [eax],al
0x40147b      add     BYTE PTR [ecx],al
0x40147d      add     BYTE PTR [eax],al
0x40147f      add     BYTE PTR [eax],al
0x401481      add     BYTE PTR [eax],al
0x401483      add     BYTE PTR [eax],al
0x401485      add     BYTE PTR [eax],al
0x401487      add     ah,ah
0x401489      add     DWORD PTR [eax],eax
0x40148b      add     BYTE PTR ds:0x0,b1
0x401491      add     BYTE PTR [eax],al

native process 346053 In: _start          L9    PC: 0x4010e0
(gdb) layout regs
(gdb) █
```

Рис. 4.11: Включение режима псевдографики

#### 4.2.1 Добавление точек останова

Проверяю, что точка останова по имени метки `_start` установлена с помощью команды `info breakpoints` и устанавливаю еще одну точку останова по адресу инструкции `mov ebx,0x0`. Просматриваю информацию о всех установленных точках останова. (рис. 4.12)

```
0x402913    add    BYTE PTR [eax],al
0x402915    add    BYTE PTR [eax],al
0x402917    add    BYTE PTR [eax],al
0x402919    add    BYTE PTR [eax],al
0x40291b    add    BYTE PTR [eax],al
0x40291d    add    BYTE PTR [eax],al
0x40291f    add    BYTE PTR [eax],al
0x402921    add    BYTE PTR [eax],al
0x402923    add    BYTE PTR [eax],al
0x402925    add    BYTE PTR [eax],al
0x402927    add    BYTE PTR [eax],al

native process 346053 In: _start L9 PC: 0x4010e0
(gdb) i b
Num      Type      Disp Enb Address  What
1        breakpoint keep y  0x004010e0 lab9-2.asm:9
        breakpoint already hit 1 time
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031
(gdb) i b
Num      Type      Disp Enb Address  What
1        breakpoint keep y  0x004010e0 lab9-2.asm:9
        breakpoint already hit 1 time
2        breakpoint keep y  0x08049031
(gdb)
```

Рис. 4.12: Установление точек останова и просмотр информации о них

## 4.2.2 Работа с данными программы в GDB

Выполняю 5 инструкций с помощью команды `stepi` и слежу за изменением значений регистров. (рис. 4.13)



```

0x402913      add     BYTE PTR [eax],al
0x402915      add     BYTE PTR [eax],al
0x402917      add     BYTE PTR [eax],al
0x402919      add     BYTE PTR [eax],al
0x40291b      add     BYTE PTR [eax],al
0x40291d      add     BYTE PTR [eax],al
0x40291f      add     BYTE PTR [eax],al
0x402921      add     BYTE PTR [eax],al
0x402923      add     BYTE PTR [eax],al
0x402925      add     BYTE PTR [eax],al
0x402927      add     BYTE PTR [eax],al

```

Register	Value	Comment
eax	0x0	0
ecx	0x0	0
edx	0x0	0
ebx	0x0	0
esp	0xffffd0a0	0xffffd0a0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x4010e0	0x4010e0 <_start>
eflags	0x202	[ IF ]
cs	0x23	35

--Type <RET> for more, q to quit, c to continue without paging--

Рис. 4.13: До использования команды stepi

(рис. 4.14)

```

aashibaeva@fedora:~/work/study/2023-2024/Архитектура компьюте...
aashiba... x aashiba... x aashiba... x aashiba... x aashiba... x
Register group: general
eax      0x8      8
ecx      0x402118 4202776
edx      0x8      8
ebx      0x1      1
esp      0xffffd0a0 0xffffd0a0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x4010f6 0x4010f6 <_start+22>
eflags   0x202    [ IF ]

```

```

B+ 0x4010e0 <_start>      mov     $0x4,%eax
0x4010e5 <_start+5>      mov     $0x1,%ebx
0x4010ea <_start+10>     mov     $0x402118,%ecx
0x4010ef <_start+15>     mov     $0x8,%edx
0x4010f4 <_start+20>     int     $0x80
> 0x4010f6 <_start+22>   mov     $0x4,%eax
0x4010fb <_start+27>     mov     $0x1,%ebx
0x401100 <_start+32>     mov     $0x402120,%ecx
0x401105 <_start+37>     mov     $0x7,%edx
0x40110a <_start+42>     int     $0x80
0x40110c <_start+44>     mov     $0x1,%eax

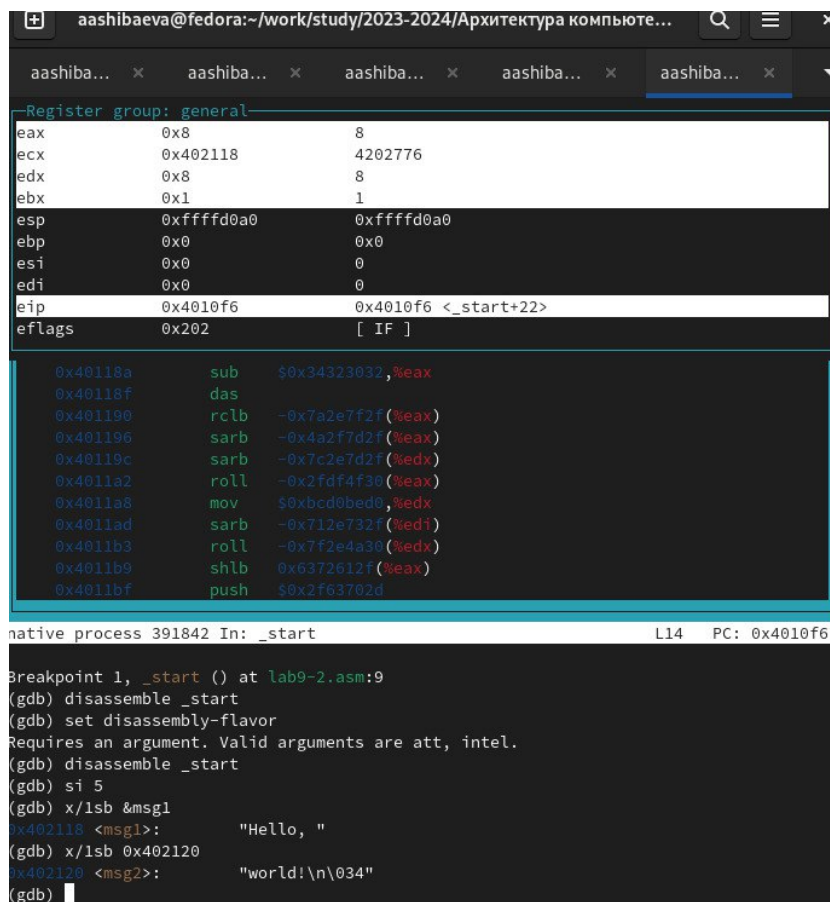
```

native process 391842 In: \_start L14 PC: 0x4010f6

Рис. 4.14: После использования команды stepi

Изменились значения регистров eax, ecx, edx и ebx.

Просматриваю значение переменной `msg1` по имени с помощью команды `x/1sb &msg1` и значение переменной `msg2` по ее адресу. (рис. 4.15)



```
aashibaeva@fedora:~/work/study/2023-2024/Архитектура компьюте...
aashiba... x aashiba... x aashiba... x aashiba... x aashiba... x
Register group: general
eax      0x8      8
ecx      0x402118 4202776
edx      0x8      8
ebx      0x1      1
esp      0xffffd0a0 0xffffd0a0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x4010f6 0x4010f6 <_start+22>
eflags   0x202    [ IF ]

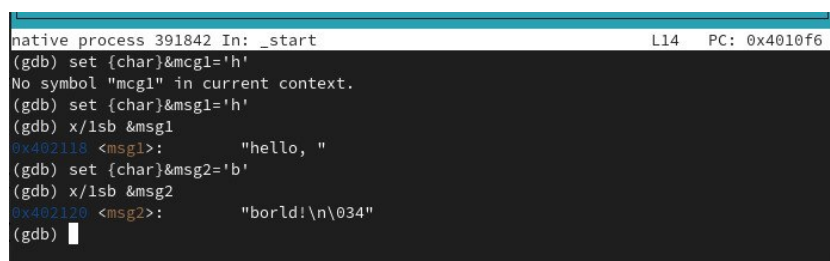
0x40118a    sub    $0x34323032,%eax
0x40118f    das
0x401190    rclb   -0x7a2e7f2f(%eax)
0x401196    sarb   -0x4a2f7d2f(%eax)
0x40119c    sarb   -0x7c2e7d2f(%edx)
0x4011a2    roll   -0x2fdf4f30(%eax)
0x4011a8    mov    $0xbcd0bed0,%edx
0x4011ad    sarb   -0x712e732f(%edi)
0x4011b3    roll   -0x7f2e4a30(%edx)
0x4011b9    shlb   0x6372612f(%eax)
0x4011bf    push   $0x2f63702d

native process 391842 In: _start L14 PC: 0x4010f6

Breakpoint 1, _start () at lab9-2.asm:9
(gdb) disassemble _start
(gdb) set disassembly-flavor
Requires an argument. Valid arguments are att, intel.
(gdb) disassemble _start
(gdb) si 5
(gdb) x/1sb &msg1
0x402118 <msg1>:      "Hello, "
(gdb) x/1sb 0x402120
0x402120 <msg2>:      "world!\n\034"
(gdb)
```

Рис. 4.15: Просмотр значений переменных

С помощью команды `set` изменяю первый символ переменной `msg1` и заменяю первый символ в переменной `msg2`. (рис. 4.16)



```
native process 391842 In: _start L14 PC: 0x4010f6
(gdb) set {char}&msg1='h'
No symbol "mcg1" in current context.
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x402118 <msg1>:      "hello, "
(gdb) set {char}&msg2='b'
(gdb) x/1sb &msg2
0x402120 <msg2>:      "borld!\n\034"
(gdb)
```

Рис. 4.16: Использование команды `set`

Вывожу в шестнадцатеричном формате, в двоичном формате и в символьном виде соответственно значение регистра `edx` с помощью команды `print p/F $val`. (рис. 4.17)

The screenshot shows a GDB window with the following content:

Register group: general		
<code>eax</code>	<code>0x8</code>	<code>8</code>
<code>ecx</code>	<code>0x402118</code>	<code>4202776</code>
<code>edx</code>	<code>0x8</code>	<code>8</code>
<code>ebx</code>	<code>0x1</code>	<code>1</code>
<code>esp</code>	<code>0xffffd0a0</code>	<code>0xffffd0a0</code>
<code>ebp</code>	<code>0x0</code>	<code>0x0</code>
<code>esi</code>	<code>0x0</code>	<code>0</code>
<code>edi</code>	<code>0x0</code>	<code>0</code>

```

0x40118f      das
0x401190      rclb    -0x7a2e7f2f(%eax)
0x401196      sarb    -0x4a2f7d2f(%eax)
0x40119c      sarb    -0x7c2e7d2f(%edx)
0x4011a2      roll    -0x2fdf4f30(%eax)
0x4011a8      mov     $0xbcd0bed0,%edx
0x4011ad      sarb    -0x712e732f(%edi)
  
```

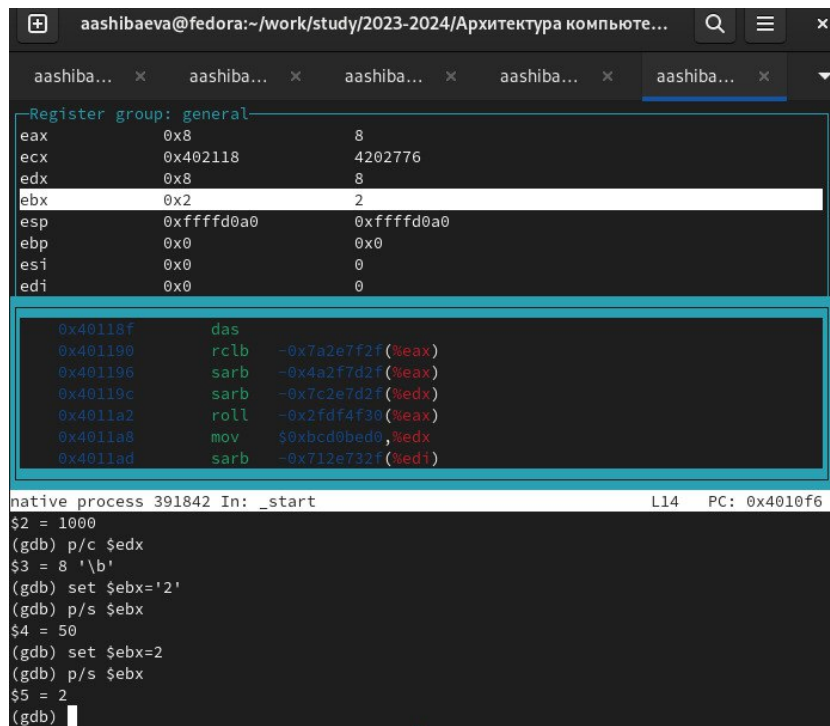
  

```

native process 391842 In: _start          L14  PC: 0x4010f6
(gdb) set (char*)&msg2='b'
(gdb) x/1sb &msg2
0x402120 <msg2>:      "borld!\n\034"
(gdb) p/x $edx
$1 = 0x8
(gdb) p/t $edx
$2 = 1000
(gdb) p/c $edx
$3 = 8 '\b'
(gdb)
  
```

Рис. 4.17: Вывод значения регистра в разных представлениях

С помощью команды `set` изменяю значение регистра `ebx` в соответствии с заданием. (рис. 4.18)



The screenshot shows a GDB interface with the following content:

```
Register group: general
eax      0x8      8
ecx      0x402118  4202776
edx      0x8      8
ebx      0x2      2
esp      0xffffd0a0 0xffffd0a0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
```

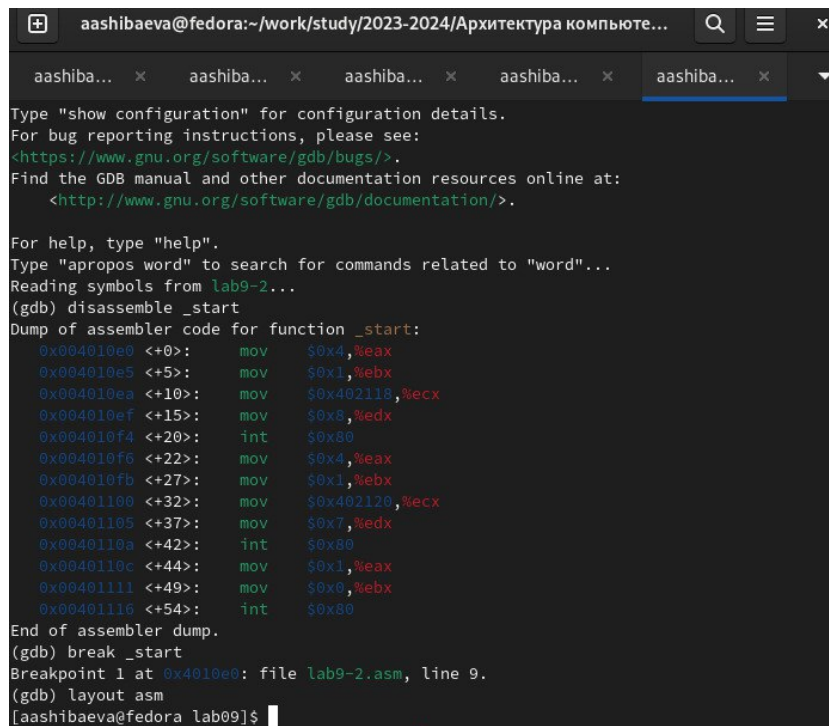
```
0x40118f      das
0x401190      rclb  -0x7a2e7f2f(%eax)
0x401196      sarb  -0x4a2f7d2f(%eax)
0x40119c      sarb  -0x7c2e7d2f(%edx)
0x4011a2      roll  -0x2fd4f30(%eax)
0x4011a8      mov   $0xbcd0bed0,%edx
0x4011ad      sarb  -0x712e732f(%edi)
```

```
native process 391842 In: _start      L14      PC: 0x4010f6
$2 = 1000
(gdb) p/c $edx
$3 = 8 '\b'
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
(gdb)
```

Рис. 4.18: Использование команды set для изменения значения регистра

Разница вывода команд p/s \$ebx отличается тем, что в первом случае мы переводим символ в его строковый вид, а во втором случае число в строковом виде не изменяется.

Завершаю выполнение программы с помощью команды continue и выхожу из GDB с помощью команды quit. (рис. 4.19)



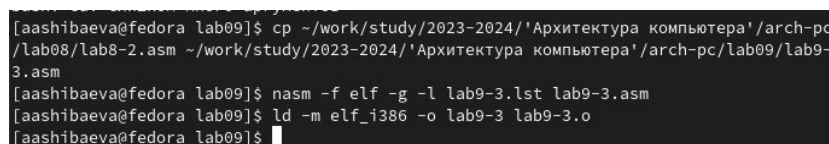
```
aashibaeva@fedora: ~/work/study/2023-2024/Архитектура компьюте...
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab9-2...
(gdb) disassemble _start
Dump of assembler code for function _start:
   0x004010e0 <+0>:  mov     $0x4,%eax
   0x004010e5 <+5>:  mov     $0x1,%ebx
   0x004010ea <+10>: mov     $0x402118,%ecx
   0x004010ef <+15>: mov     $0x8,%edx
   0x004010f4 <+20>:  int     $0x80
   0x004010f6 <+22>: mov     $0x4,%eax
   0x004010fb <+27>: mov     $0x1,%ebx
   0x00401100 <+32>: mov     $0x402120,%ecx
   0x00401105 <+37>: mov     $0x7,%edx
   0x0040110a <+42>:  int     $0x80
   0x0040110c <+44>: mov     $0x1,%eax
   0x00401111 <+49>: mov     $0x0,%ebx
   0x00401116 <+54>:  int     $0x80
End of assembler dump.
(gdb) break _start
Breakpoint 1 at 0x4010e0: file lab9-2.asm, line 9.
(gdb) layout asm
[aashibaeva@fedora lab09]$
```

Рис. 4.19: Завершение работы GDB

### 4.2.3 Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm с программой из листинга 8.2 в файл с именем lab9-3.asm и создаю исполняемый файл. (рис. 4.20)



```
[aashibaeva@fedora lab09]$ cp ~/work/study/2023-2024/Архитектура компьюте...
~/work/study/2023-2024/Архитектура компьюте.../arch-pc/lab08/lab8-2.asm
~/work/study/2023-2024/Архитектура компьюте.../arch-pc/lab09/lab9-3.asm
[aashibaeva@fedora lab09]$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
[aashibaeva@fedora lab09]$ ld -m elf_i386 -o lab9-3 lab9-3.o
[aashibaeva@fedora lab09]$
```

Рис. 4.20: Создание файла

Загружаю исполняемый файл в отладчик gdb, указывая необходимые аргументы с использованием ключа -args. (рис. 4.21)

```
[aashibaeva@fedora lab09]$ gdb --args lab9-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (GDB) Fedora Linux 13.2-4.fc38
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab9-3...
(gdb)
```

Рис. 4.21: Загрузка файла с аргументами в отладчик

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее. (рис. 4.22)

```
(gdb) b _start
Breakpoint 1 at 0x4011a3: file lab9-3.asm, line 5.
(gdb) run
Starting program: /home/aashibaeva/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09/lab9-3 аргумент1 аргумент 2 аргумент\ 3

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

Breakpoint 1, _start () at lab9-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb)
```

Рис. 4.22: Установление точки останова и запуск программы

Посматриваю вершину стека и позиции стека по их адресам. (рис. 4.23)

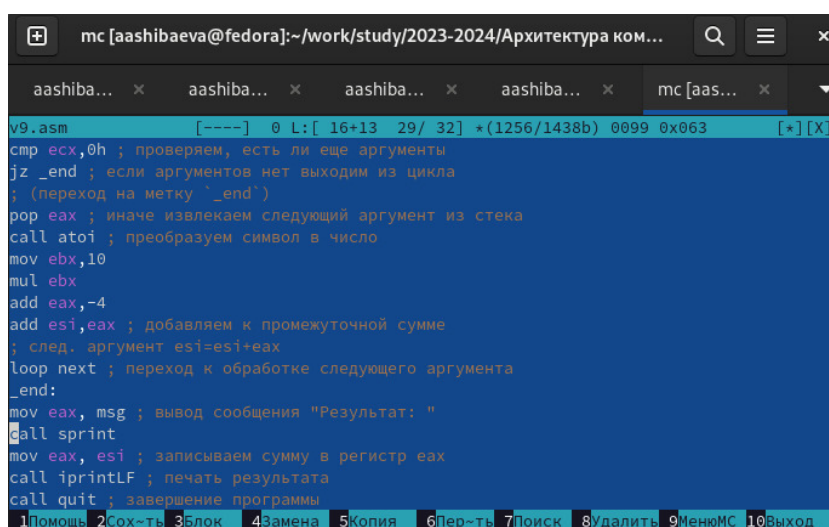
```
(gdb) x/x $esp
0xffffd068: 0x00000005
(gdb) x/s *(void**)($esp+4)
0xffffd216: "/home/aashibaeva/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)($esp+8)
0xffffd27d: "аргумент1"
(gdb) x/s *(void**)($esp+12)
0xffffd28f: "аргумент"
(gdb) x/s *(void**)($esp+16)
0xffffd2a0: "2"
(gdb) x/s *(void**)($esp+20)
0xffffd2a2: "аргумент 3"
(gdb) x/s *(void**)($esp+24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 4.23: Просмотр значений, введенных в стек

Шаг изменения адреса равен 4, т.к количество аргументов командной строки равно 4.

### 4.3 Задания для самостоятельной работы

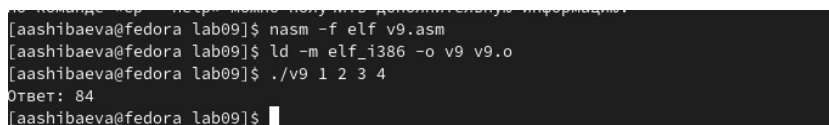
1. Преобразовываю программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции  $f(x)$  как подпрограмму. (рис. 4.24)



```
v9.asm [----] 0 L: [ 16+13 29/ 32] *(1256/1438b) 0099 0x063 [*] [X]
cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку '_end')
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
mov ebx,10
mul ebx
add ebx,-4
add esi,eax ; добавляем к промежуточной сумме
; след. аргумент esi=esi+eax
loop next ; переход к обработке следующего аргумента
_end:
mov eax,msg ; вывод сообщения "Результат: "
call sprint
mov eax,esi ; записываем сумму в регистр eax
call iprintf ; печать результата
call quit ; завершение программы
1Помощь 2Сох-ть 3Блок 4Замена 5Копия 6Пер-ть 7Поиск 8Удалить 9МенюМС 10Выход
```

Рис. 4.24: Написание кода подпрограммы

Запускаю код и проверяю, что она работает корректно. (рис. 4.25)



```
[aashibaeva@fedora lab09]$ nasm -f elf v9.asm
[aashibaeva@fedora lab09]$ ld -m elf_i386 -o v9 v9.o
[aashibaeva@fedora lab09]$ ./v9 1 2 3 4
Ответ: 84
[aashibaeva@fedora lab09]$
```

Рис. 4.25: Запуск программы и проверка его вывода

Код программы:

%include 'in\_out.asm'

```

SECTION .data
msg db "Ответ:",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 ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
mov ebx,10
mul ebx
add eax,-4
add esi,eax ; добавляем к промежуточной сумме
; след. аргумент esi=esi+eax
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg ; вывод сообщения "Результат:"
call sprint
mov eax, esi ; записываем сумму в регистр eax

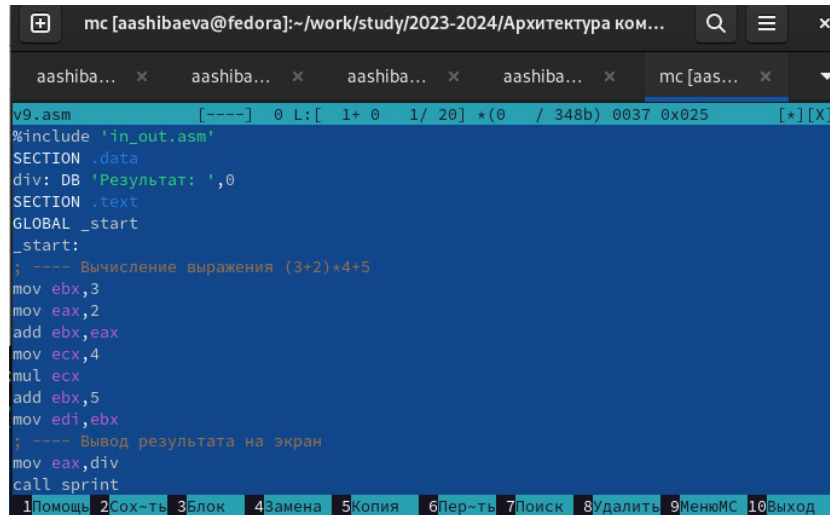
```



call iprintLF ; печать результата

call quit ; завершение программы

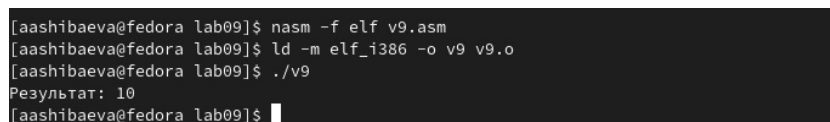
2. Ввожу в файл task1.asm текст программы из листинга 9.3. (рис. 4.26)



```
v9.asm [----] 0 L: [ 1+ 0 1/ 20] *(0 / 348b) 0037 0x025 [*] [X]
#include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx
; ---- Вывод результата на экран
mov eax,div
call sprint
1Помощь 2Сох-ть 3Блок 4Замена 5Копия 6Пер-ть 7Поиск 8Удалить 9МенюМС 10Выход
```

Рис. 4.26: Ввод текста программы из листинга 9.3

При корректной работе программы должно выводиться “25”. Создаю исполняемый файл и запускаю его. (рис. 4.27)



```
[aashibaeva@fedora lab09]$ nasm -f elf v9.asm
[aashibaeva@fedora lab09]$ ld -m elf_i386 -o v9 v9.o
[aashibaeva@fedora lab09]$ ./v9
Результат: 10
[aashibaeva@fedora lab09]$
```

Рис. 4.27: Создание и запуск исполняемого файла

Видим, что в выводе мы получаем неправильный ответ.

Получаю исполняемый файл для работы с GDB, запускаю его и ставлю брейкпоинты для каждой инструкции, связанной с вычислениями. С помощью команды continue прохожусь по каждому брейкпоинту и слежу за изменениями значений регистров.

При выполнении инструкции mul ecx происходит умножение ecx на eax, то есть 4 на 2, вместо умножения 4 на 5 (регистр ebx). Происходит это из-за того, что

стоящая перед `mov ecx,4` инструкция `add ebx,eax` не связана с `mul ecx`, но связана инструкция `mov eax,2`. (рис. 4.28)

```

eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd160 0xffffd160
ebp      0x0      0x0

B+ 0x80490f4 <_start+12> mov    ecx,0x4
B+ 0x80490f9 <_start+17> mul    ecx
B+ 0x80490fb <_start+19> add    ebx,0x5
b+ 0x80490fe <_start+22> mov    edi,ebx
0x8049100 <_start+24> mov    eax,0x804a000

native process 14573 In: _start L13 PC: 0x80490
Continuing.

Breakpoint 5, _start () at task2.asm:12
(gdb) c
Continuing.

Breakpoint 6, _start () at task2.asm:13
(gdb)

```

Рис. 4.28: Нахождение причины ошибки

Из-за этого мы получаем неправильный ответ. (рис. 4.29)

```

eax      0x0      0
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd160 0xffffd160
ebp      0x0      0x0

B+ 0x80490f9 <_start+17> mul    ecx
B+ 0x80490fb <_start+19> add    ebx,0x5
B+ 0x80490fe <_start+22> mov    edi,ebx
0x8049100 <_start+24> mov    eax,0x804a000
0x8049105 <_start+29> call   0x804900f <sprint>

native process 14573 In: _start L14 PC: 0x80490
Continuing.

Breakpoint 6, _start () at task2.asm:13
(gdb) c
Continuing.

Breakpoint 7, _start () at task2.asm:14
(gdb)

```

Рис. 4.29: Неверное изменение регистра

Исправляем ошибку, добавляя после `add ebx,eax` `mov eax,ebx` и заменяя `ebx` на `eax` в инструкциях `add ebx,5` и `mov edi,ebx`. (рис. 4.30)

```

v9.asm [-M--] 10 L:[ 3+14 17/ 21] *(311 / 360b) 0118 0x076
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add ebx,eax
mov eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax
; ---- Вывод результата на экран
mov eax,edi
call sprint
mov eax,edi

```

Рис. 4.30: Исправление ошибки

Также, вместо того, чтобы изменять значение `eax`, можно было изменять значение неиспользованного регистра `edx`.

Создаем исполняемый файл и запускаем его. Убеждаемся, что ошибка исправлена. (рис. 4.31)

```

[aashibaeva@fedora lab09]$ nasm -f elf v9.asm
[aashibaeva@fedora lab09]$ ld -m elf_i386 -o v9 v9.o
[aashibaeva@fedora lab09]$ ./v9
Результат: 25

```

Рис. 4.31: Ошибка исправлена

Код программы:

```
%include 'in_out.asm'
```

```
SECTION .data
```

```
div: DB 'Результат:',0
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
; -- Вычисление выражения (3+2)*4+5
```

```
mov ebx,3
```

```
mov eax,2
add ebx,eax
mov eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax
; -- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```

## 5 Выводы

Во время выполнения данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм и ознакомилась с методами отладки при помощи GDB и его основными возможностями.

## 6 Список литературы

1. GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.
2. GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.
3. Midnight Commander Development Center. — 2021. — URL: <https://midnight-commander.org/>.
4. NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005 — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658>.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.
11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.
13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: [http://www.stolyarov.info/books/asm\\_unix](http://www.stolyarov.info/books/asm_unix).

15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).