

# **Отчёт по лабораторной работе №9**

**Дисциплина: Архитектура компьютера**

Пономарева Татьяна Александровна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>6</b>
<b>2</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
3.1	Реализация подпрограмм в NASM . . . . .	8
3.2	Отладка программ с помощью GDB . . . . .	11
3.3	Добавление точек останова . . . . .	15
3.4	Работа с данными программы в GDB . . . . .	16
3.5	Обработка аргументов командной строки в GDB . . . . .	19
<b>4</b>	<b>Задания для самостоятельной работы</b>	<b>21</b>
<b>5</b>	<b>Выводы</b>	<b>25</b>
	<b>Список литературы</b>	<b>26</b>

# Список иллюстраций

3.1	Терминал. Создание каталога lab09. Создание файла lab09-1.asm .	8
3.2	Окно Midnight Commander. Содержание файла lab09-1.asm . . . . .	8
3.3	Терминал. Терминал. Создание исполняемого файла lab09-1. Проверка работы lab09-1 . . . . .	9
3.4	Терминал. Компиляция исполняемого файла lab09-1. Проверка работы lab09-1 . . . . .	11
3.5	Терминал. Создание файла lab09-2.asm . . . . .	11
3.6	Окно Midnight Commander. Содержание файла lab09-2.asm . . . . .	12
3.7	Терминал. Компиляция исполняемого файла lab09-2 . . . . .	12
3.8	Терминал. Загрузка исполняемого файла lab09-2 в gdb . . . . .	12
3.9	Терминал. Проверка корректности работы исполняемого файла lab09-2 в gdb . . . . .	13
3.10	Терминал. Работа с исполняемым файлом lab09-2 в gdb. Установка брейкпоинта . . . . .	13
3.11	Терминал. Работа с исполняемым файлом lab09-2 в gdb. Команда disassemble . . . . .	13
3.12	Терминал. Работа с исполняемым файлом lab09-2 в gdb. Переключение на отображение команд с синтаксисом Intel . . . . .	14
3.13	Терминал. Работа в gdb. Режим псевдографики gdb . . . . .	15
3.14	Терминал. Работа в gdb. Проверка на точку останова . . . . .	15
3.15	Терминал. Работа в gdb. Установка точки останова по адресу инструкции . . . . .	16
3.16	Терминал. Работа в gdb. Информация о точках останова . . . . .	16
3.17	Терминал. Работа в gdb. Содержимое регистров . . . . .	16
3.18	Терминал. Работа в gdb. Отображение содержимого памяти . . . . .	17
3.19	Терминал. Работа в gdb. Отображение измененного содержимого памяти . . . . .	17
3.20	Терминал. Работа в gdb. Форматы регистра edx . . . . .	18
3.21	Терминал. Работа в gdb. Команда set . . . . .	18
3.22	Терминал. Копирование файла lab8-2.asm в файл lab09-3.asm . . . . .	19
3.23	Терминал. Создание исполняемого файла lab09-3 . . . . .	19
3.24	Терминал. Загрузка исполняемого файла lab09-3 в gdb . . . . .	19
3.25	Терминал. Установка точки останова. Позиции стека . . . . .	20
4.1	Терминал. Компиляция исполняемого файла lab09-4. Проверка работы lab09-4 . . . . .	23
4.2	Терминал. Работа в gdb. Просмотр lab09-5 . . . . .	23

4.3	Терминал. Компиляция исполняемого файла lab09-5. Проверка работы lab09-5 . . . . .	23
-----	------------------------------------------------------------------------------------	----

## **Список таблиц**

# 1 Цель работы

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

## 2 Теоретическое введение

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

## 3 Выполнение лабораторной работы

### 3.1 Реализация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы № 9, перехожу в него и создаю файл lab09-1.asm (рис. 3.1).

```
taponomareva@2c7fe9w:~$ mkdir ~/work/arch-pc/lab09
taponomareva@2c7fe9w:~$ cd ~/work/arch-pc/lab09
taponomareva@2c7fe9w:~/work/arch-pc/lab09$ touch lab09-1.asm
```

Рис. 3.1: Терминал. Создание каталога lab09. Создание файла lab09-1.asm

Ввожу в файл lab09-1.asm текст программы из листинга 9.1 (рис. 3.2).

```
GNU nano 7.2 /home/taponomareva/work/arch-pc/lab09/lab09-1.asm
include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
res: RESB 80
SECTION .text
GLOBAL _start
_start:
;-----
; Основная программа
;-----
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 iprintf
call quit
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы
```

Рис. 3.2: Окно Midnight Commander. Содержание файла lab09-1.asm

Создаю исполняемый файл и проверяю его работу (рис. 3.3). Программа рабо-



тает корректно и при вводе  $x=10$  дает результат 27.

```
taponomareva@2c7fe9w:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
taponomareva@2c7fe9w:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
taponomareva@2c7fe9w:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 10
2x+7=27
```

Рис. 3.3: Терминал. Терминал. Создание исполняемого файла lab09-1. Проверка работы lab09-1

Изменяю текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения  $f(g(x))$ , где  $x$  вводится с клавиатуры.

Текст программы в lab09-1.asm

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

```
SECTION .data
```

```
msg: DB 'Введите x: ', 0
```

```
result: DB '2(3x-1)+7=', 0
```

```
SECTION .bss
```

```
x: RESB 80
```

```
res: RESB 80
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
;-----
```

```
; Основная программа
```

```
;-----
```

```
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 sprintf
mov eax,[res]
call iprintLF

call quit
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
push eax
call _subcalcul

mov ebx, 2
mul ebx
add eax, 7

mov [res],eax
pop eax
ret ; выход из подпрограммы
;-----

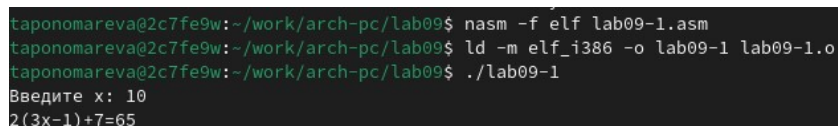
```

```

; Подпрограмма вычисления
; выражения "3x-1"
_subcalcul:
mov ebx, 3
mul ebx
sub eax, 1
ret

```

Компилирую исполняемый файл и проверяю его работу (рис. 3.4). Заметим, что программа работает корректно, выводя 65 при  $x=10$ .



```

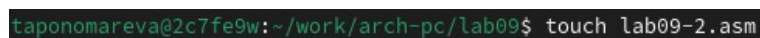
taponomareva@2c7fe9w:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
taponomareva@2c7fe9w:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
taponomareva@2c7fe9w:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 10
2(3x-1)+7=65

```

Рис. 3.4: Терминал. Компиляция исполняемого файла lab09-1. Проверка работы lab09-1

## 3.2 Отладка программ с помощью GDB

Создаю файл lab09-2.asm (рис. 3.5).



```

taponomareva@2c7fe9w:~/work/arch-pc/lab09$ touch lab09-2.asm

```

Рис. 3.5: Терминал. Создание файла lab09-2.asm

Ввожу в файл lab09-2.asm текст программы из листинга 9.2 (рис. 3.6).

```
GNU nano 7.2 /home/taponomareva/work/arch-pc/lab09/lab09-2.asm
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. 3.6: Окно Midnight Commander. Содержание файла lab09-2.asm

Получаю исполняемый файл. Так как для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, то для этого провожу трансляцию программ с ключом -g (рис. 3.7).

```
taponomareva@2c7fe9w: ~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
taponomareva@2c7fe9w: ~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
```

Рис. 3.7: Терминал. Компиляция исполняемого файла lab09-2

Загружаю исполняемый файл в отладчик gdb (рис. 3.8).

```
taponomareva@2c7fe9w: ~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 15.2-1.fc40
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...
(gdb)
```

Рис. 3.8: Терминал. Загрузка исполняемого файла lab09-2 в gdb

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run (рис. 3.9). Программа выводит на экран надпись “Hello, world!”, что указывает на правильность работы программы.

```
(gdb) run

Starting program: /home/taonomareva/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 5239) exited normally]
```

Рис. 3.9: Терминал. Проверка корректности работы исполняемого файла lab09-2 в gdb

Для более подробного анализа программы устанавливаю брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запускаю её (рис. 3.10).

```
(gdb) break _start

Breakpoint 1 at 0x08049000: file lab09-2.asm, line 11.
(gdb) run

Starting program: /home/taonomareva/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:11
11      mov eax, 4
```

Рис. 3.10: Терминал. Работа с исполняемым файлом lab09-2 в gdb. Установка брейкпоинта

Затем рассматриваю дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start` (рис. 3.11).

```
(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.
```

Рис. 3.11: Терминал. Работа с исполняемым файлом lab09-2 в gdb. Команда `disassemble`

Переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel` (рис. 3.12).

```
(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.
```

Рис. 3.12: Терминал. Работа с исполняемым файлом lab09-2 в gdb. Переключение на отображение команд с синтаксисом Intel

Различия отображения синтаксиса машинных команд в режимах АТТ и Intel.

Синтаксис машинных команд в режимах АТТ и Intel различается порядком операндов, обозначениями регистров, префиксами констант и обращением к памяти. В синтаксисе Intel первый операнд — назначение, второй — источник (например, `mov eax, ebx`), тогда как в АТТ — наоборот (`movl %ebx, %eax`). Регистры в АТТ начинаются с %, а в Intel указываются без префиксов. Константы в АТТ начинаются с \$, тогда как в Intel они пишутся без символов.

Включаю режим псевдографики для более удобного анализа программы (рис. 3.13).

```

taponomareva@2c7fe9w:~/work/arch-pc/lab09 — gdb lab09-2
--Register group: general--
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffd030 0xffffd030  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049000 0x8049000 <_start>  eflags    0x202     [ IF ]
cs       0x23     35      ss       0x2b     43
ds       0x2b     43      es       0x2b     43
fs       0x0      0      gs       0x0      0

B->0x8049000 <_start> mov eax,0x4
0x8049005 <_start+5> mov ebx,0x1
0x804900a <_start+10> mov ecx,0x804a000
0x804900f <_start+15> mov edx,0x8
0x8049014 <_start+20> int 0x80
0x8049016 <_start+22> mov eax,0x4
0x804901b <_start+27> mov ebx,0x1
0x8049028 <_start+32> mov ecx,0x804a008
0x8049025 <_start+37> mov edx,0x7
0x804902a <_start+42> int 0x80
0x804902c <_start+44> mov eax,0x1
0x8049031 <_start+49> mov ebx,0x0

native process 5477 (asm) In: _start L11 PC: 0x8049000
(gdb) layout regs
(gdb)

```

Рис. 3.13: Терминал. Работа в gdb. Режим псевдографики gdb

### 3.3 Добавление точек останова

Поскольку на предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверяю это с помощью команды `info breakpoints` (кратко `i b`) (рис. 3.14).

```

taponomareva@2c7fe9w:~/work/arch-pc/lab09 — gdb lab09-2
--Register group: general--
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffd030 0xffffd030  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049000 0x8049000 <_start>  eflags    0x202     [ IF ]
cs       0x23     35      ss       0x2b     43
ds       0x2b     43      es       0x2b     43

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

native process 5576 (asm) In: _start L11 PC: 0x8049000
(gdb) layout regs
(gdb) info breakpoints
Num   Type             Disp Enb Address      What
1      breakpoint       keep y   0x8049000 lab09-2.asm:11
      breakpoint already hit 1 time
(gdb)

```

Рис. 3.14: Терминал. Работа в gdb. Проверка на точку останова

Устанавливаю еще одну точку останова по адресу инструкции. Определяю адрес предпоследней инструкции (`mov ebx,0x0`) и устанавливаю точку останова

при помощи команды `break *` (рис. 3.15).

```
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 24.
```

Рис. 3.15: Терминал. Работа в gdb. Установка точки останова по адресу инструкции

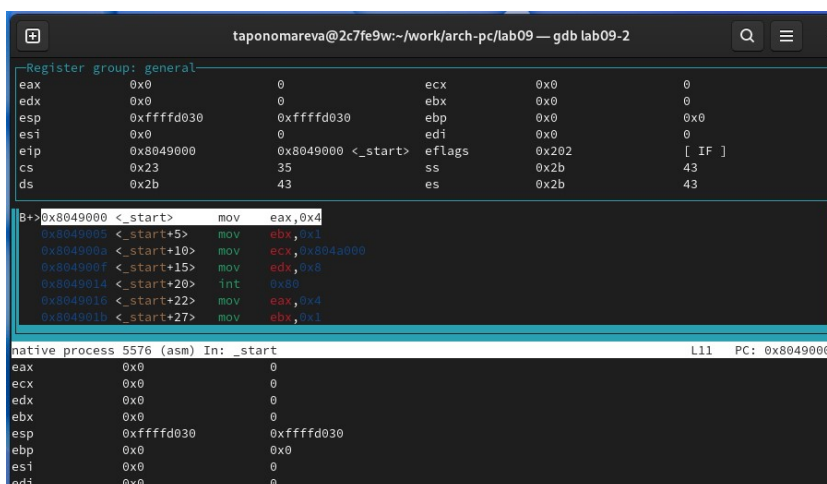
Рассматриваю информацию о всех установленных точках останова (рис. 3.16).

```
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y 0x08049000 lab09-2.asm:11
          breakpoint already hit 1 time
2        breakpoint keep y 0x08049031 lab09-2.asm:24
```

Рис. 3.16: Терминал. Работа в gdb. Информация о точках останова

## 3.4 Работа с данными программы в GDB

Рассматриваю содержимое регистров с помощью команды `info registers` (или `i r`) (рис. 3.17).



The screenshot shows a GDB terminal window with the title "taponomareva@2c7fe9w:~/work/arch-pc/lab09 — gdb lab09-2". The "Register group: general" section displays the following values:

Register	Value	Register	Value	Register	Value
eax	0x0	ecx	0x0		
edx	0x0	ebx	0x0		
esp	0xffffd030	ebp	0x0		
esi	0x0	edi	0x0		
eip	0x8049000	eip	0x8049000	eip	0x8049000
cs	0x23	cs	0x23	cs	0x23
ds	0x2b	ds	0x2b	ds	0x2b

Below the register list, the assembly code is shown for the current instruction:

```
B->0x8049000 <_start> mov eax,0x4
0x8049005 <_start+5> mov ebx,0x1
0x804900a <_start+10> mov ecx,0x804a000
0x804900f <_start+15> mov edx,0x0
0x8049014 <_start+20> int 0x0
0x8049016 <_start+22> mov eax,0x4
0x804901b <_start+27> mov ebx,0x1
```

At the bottom, the "native process 5576 (asm) In: \_start" section shows the current state of the registers:

Register	Value	Register	Value
eax	0x0	ecx	0x0
edx	0x0	ebx	0x0
esp	0xffffd030	ebp	0x0
esi	0x0	edi	0x0

Рис. 3.17: Терминал. Работа в gdb. Содержимое регистров

Смотрю содержимое переменных `msg1` и `msg2` по имени и по адресу (рис. 3.18).



```

taponomareva@2c7fe9w:~/work/arch-pc/lab09 — gdb lab09-2
Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffd030 0xffffd030  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049000 0x8049000 <_start>  eflags    0x202    [ IF ]
cs       0x23     35      ss       0x2b     43
ds       0x2b     43      es       0x2b     43

0x804900f <_start+15> mov     edx,0x5
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

native process 5576 (asm) In: _start L11 PC: 0x8049000
ds      0x2b     43
es      0x2b     43
fs      0x0      0
gs      0x0      0
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb)

```

Рис. 3.18: Терминал. Работа в gdb. Отображение содержимого памяти

Меняю содержимое переменных msg1 и msg2 по имени и по адресу (рис. 3.19).

```

taponomareva@2c7fe9w:~/work/arch-pc/lab09 — gdb lab09-2
Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffd030 0xffffd030  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049000 0x8049000 <_start>  eflags    0x202    [ IF ]
cs       0x23     35      ss       0x2b     43
ds       0x2b     43      es       0x2b     43

0x804900f <_start+15> mov     edx,0x5
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

native process 5576 (asm) In: _start L11 PC: 0x8049000
(gdb) set {char}msg1='h'
'msg1' has unknown type; cast it to its declared type
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb) set {char}&msg2='k'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "korld!\n\034"
(gdb)

```

Рис. 3.19: Терминал. Работа в gdb. Отображение измененного содержимого памяти

Вывожу в различных форматах значение регистра edx (рис. 3.20).

```

taponomareva@2c7fe9w:~/work/arch-pc/lab09 — gdb lab09-2

Register group: general
eax      0x4      4
ecx      0x804a008 134520840
edx      0x7      7
ebx      0x1      1
esp      0xffffd030 0xffffd030
ebp      0x0      0
esi      0x0      0

B+ 0x8049000 <_start> mov eax, 4
0x8049005 <_start+5> mov ebx, 1
0x804900a <_start+10> mov ecx, 0x804a008
0x804900f <_start+15> mov edx, 0x8
0x8049014 <_start+20> int 0x80
0x8049016 <_start+22> mov eax, 0x4
0x804901b <_start+27> mov ebx, 1

native process 5576 (asm) In: _start L21 PC: 0x804902a
(gdb) p/x $edx
A syntax error in expression, near '$edx'.
(gdb) p/s $edx
$7 = 7
(gdb) p/t $edx
$8 = 111
(gdb) p/x $edx
$9 = 0x7
(gdb)

```

Рис. 3.20: Терминал. Работа в gdb. Форматы регистра edx

С помощью команды `set` изменяю значение регистра `ebx` (рис. 3.21). В первом случае `'2'` - это символ с ASCII-кодом 50. Команда `p/s $ebx` интерпретирует содержимое регистра `$ebx` как строку символов. Результат равен 50, потому что ASCII-код символа `'2'` соответствует 50. Во втором случае `2` - это целое число, интерпретируемое как указатель в памяти, который не ведет к осмысленной строке.

```

taponomareva@2c7fe9w:~/work/arch-pc/lab09 — gdb lab09-2

Register group: general
eax      0x4      4
ecx      0x804a008 134520840
edx      0x7      7
ebx      0x2      2
esp      0xffffd030 0xffffd030
ebp      0x0      0
esi      0x0      0

B+ 0x8049000 <_start> mov eax, 4
0x8049005 <_start+5> mov ebx, 1
0x804900a <_start+10> mov ecx, 0x804a008
0x804900f <_start+15> mov edx, 0x8
0x8049014 <_start+20> int 0x80
0x8049016 <_start+22> mov eax, 0x4
0x804901b <_start+27> mov ebx, 1

native process 5576 (asm) In: _start L21 PC: 0x804902a
(gdb) p/x $edx
$9 = 0x7
(gdb) set $ebx='2'
(gdb) p/s $ebx
$10 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$11 = 2
(gdb)

```

Рис. 3.21: Терминал. Работа в gdb. Команда `set`

Завершаю выполнение программы с помощью команды `continue` (сокращенно

с) или `stepi` (сокращенно `si`) и выхожу из GDB с помощью команды `quit` (сокращенно `q`).

### 3.5 Обработка аргументов командной строки в GDB

Скопирую файл `lab8-2.asm`, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем `lab09-3.asm` (рис. 3.22).

```
taponomareva@2c7fe9w:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
```

Рис. 3.22: Терминал. Копирование файла `lab8-2.asm` в файл `lab09-3.asm`

Создаю исполняемый файл (рис. 3.23).

```
taponomareva@2c7fe9w:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
taponomareva@2c7fe9w:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
```

Рис. 3.23: Терминал. Создание исполняемого файла `lab09-3`

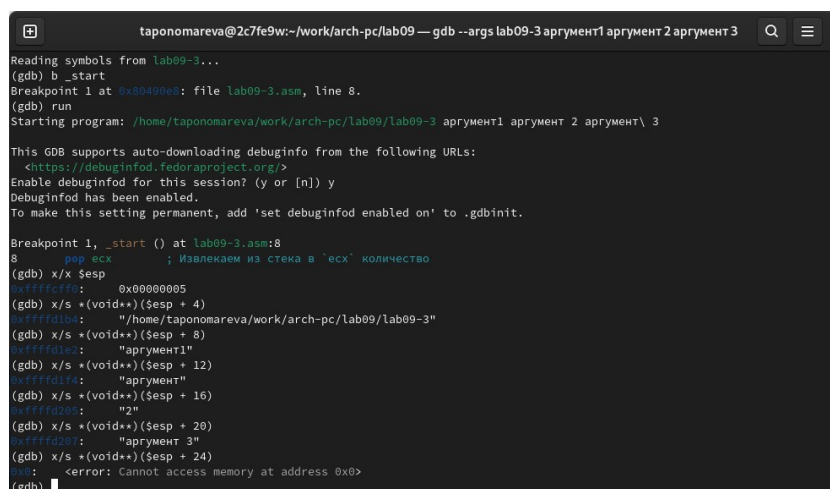
Для загрузки в `gdb` программы с аргументами необходимо использовать ключ `-args`. Загружаю исполняемый файл в отладчик, указав аргументы (рис. 3.24).

```
taponomareva@2c7fe9w:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Fedora Linux) 15.2-1.fc40
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-3...
(gdb)
```

Рис. 3.24: Терминал. Загрузка исполняемого файла `lab09-3` в `gdb`

Для начала устанавливаю точку останова перед первой инструкцией в программе и запускаю ее (рис. 3.25). Затем рассматриваю остальные позиции стека. Шаг изменения адреса равен 4, поскольку это соответствует размеру одного элемента

типа `void*` на 32-битных системах с размером указателя в 4 байта. Это означает, что при обращении к следующему элементу в стеке необходимо сдвинуть указатель на 4 байта.



```
taponomareva@2c7fe9w:~/work/arch-pc/lab09 — gdb --args lab09-3 аргумент1 аргумент2 аргумент3
Reading symbols from lab09-3...
(gdb) b _start
Breakpoint 1 at 0x80490e5: file lab09-3.asm, line 8.
(gdb) run
Starting program: /home/taponomareva/work/arch-pc/lab09/lab09-3 аргумент1 аргумент2 аргумент3

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 lab09-3.asm:8
8      pop ecx          ; Извлекаем из стека в 'ecx' количество
(gdb) x/x $esp
0xffffd1f4: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffd1d4: "/home/taponomareva/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd1c2: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd1f4: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd205: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd207: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 3.25: Терминал. Установка точки останова. Позиции стека

## 4 Задания для самостоятельной работы

- 1) Преобразовываю программу из лабораторной работы №8(Задание №1 для самостоятельной работы), реализовав вычисление значения функции  $f(x)=10x-5$  как подпрограмму.

Код программы файла lab09-4.asm

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

```
SECTION .data
```

```
msg_function db "Функция:  $f(x) = 10x - 5$ ", 0
```

```
msg_res db "Результат: ", 0
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
    mov eax, msg_function
```

```
    call sprintf
```

```
    pop ecx
```

```
    pop edx
```

```

    sub ecx, 1
    mov esi, 0

next:
    cmp ecx, 0h
    jz _end
    pop eax
    call atoi

    call _solve_f

    add esi, eax
    loop next

_end:
    mov eax, msg_res
    call sprint
    mov eax, esi
    call iprintLF
    call quit

_solve_f:
    mov ebx, 10          ; Загружаем множитель 10 в ebx
    mul ebx              ; Умножаем eax на 10, результат в eax (eax = eax * 10)
    sub eax, 5           ; Вычитаем 5 из eax (eax = eax - 5)
    ret                  ; Возвращаемся из подпрограммы

```

Компилирую исполняемый файл и проверяю его работу (рис. 4.1). Программа работает корректно, т.к.  $101-5 + 102-5+10*3-5 = 5+15+25 = 45$ .

```

taponomareva@2c7fe9w:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
taponomareva@2c7fe9w:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
taponomareva@2c7fe9w:~/work/arch-pc/lab09$ ./lab09-4 1 2 3
Функция: f(x) = 10x - 5
Результат: 45

```

Рис. 4.1: Терминал. Компиляция исполняемого файла lab09-4. Проверка работы lab09-4

- 2) Проанализировав программу при помощи отладчика GDB, можно заметить, что ошибка заключается в неправильном использовании регистров при выполнении арифметических операций. В инструкции `mul` `ecx` результат умножения сохраняется в регистрах `edx` и `eax`, но программа ожидает, что результат будет в `ebx`, что приводит к потере данных и неправильному результату (рис. 4.2).

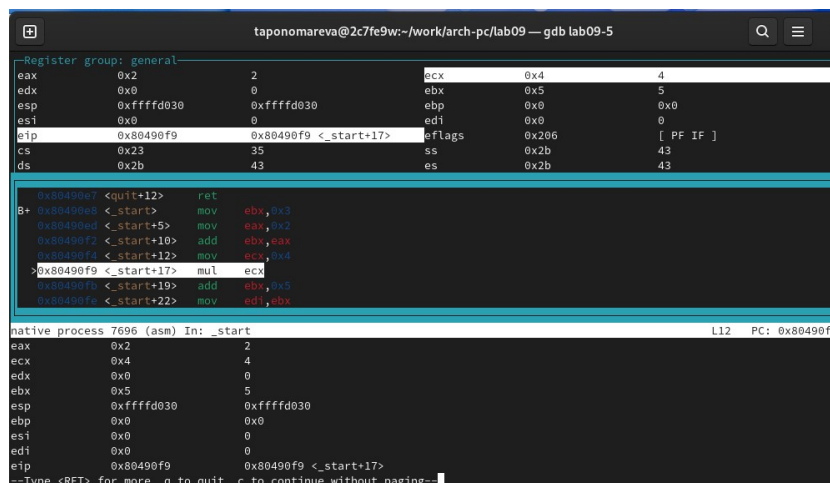


Рис. 4.2: Терминал. Работа в gdb. Просмотр lab09-5

После исправления кода получаем верный ответ (рис. 4.3).

```

taponomareva@2c7fe9w:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
taponomareva@2c7fe9w:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab09-5.o
taponomareva@2c7fe9w:~/work/arch-pc/lab09$ ./lab09-5
Результат: 25

```

Рис. 4.3: Терминал. Компиляция исполняемого файла lab09-5. Проверка работы lab09-5

Исправленный код

```

#include 'in_out.asm'

SECTION .data
div: DB 'Результат: ',0

SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov eax,3
add eax,2
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

```

Загружаю отчет на GitHub.



## **5 Выводы**

В ходе выполнения лабораторной работы были приобретены навыки написания программ с использованием подпрограмм. Было произведено знакомство с методами отладки при помощи GDB и его основными возможностями.

## **Список литературы**

1. Курс на ТУИС
2. Лабораторная работа №9