

Отчёт по лабораторной работе №9

Дисциплина: архитектура компьютера

Барбакова Алиса Саяновна

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	8
4.1	Релазиация подпрограмм в NASM	8
4.1.1	Отладка программ с помощью GDB	11
4.1.2	Добавление точек останова	16
4.1.3	Работа с данными программы в GDB	18
4.1.4	Обработка аргументов командной строки в GDB . . .	23
4.2	Задание для самостоятельной работы	24
5	Выводы	30
	Список литературы	31

Список иллюстраций

4.1	Создание рабочего каталога	8
4.2	Запуск программы из листинга	9
4.3	Изменение программы первого листинга	9
4.4	Запуск программы в отладчике	12
4.5	Проверка программы отладчиком	13
4.6	Запуск отладчика с брейкпойнтом	14
4.7	Дисассимилирование программы	15
4.8	Режим псевдографики	16
4.9	Список брейкпойнтов	17
4.10	Добавление второй точки остановки	18
4.11	Просмотр содержимого регистров	19
4.12	Просмотр содержимого переменных двумя способами . .	20
4.13	Изменение содержимого переменных двумя способами . .	21
4.14	Просмотр значения регистра разными представлениями .	22
4.15	Примеры использования команды set	23
4.16	Подготовка новой программы	23
4.17	Проверка работы стека	24
4.18	Измененная программа предыдущей лабораторной работы	25
4.19	Работа программы	27
4.20	Поиск ошибки в программе через пошаговую отладку . . .	28
4.21	Проверка корректировок в программе	28

1 Цель работы

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

2 Задание

1. Реализация подпрограмм в NASM
2. Отладка программ с помощью GDB
3. Самостоятельное выполнение заданий по материалам лабораторной работы

3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа:

- обнаружение ошибки;
- поиск её местонахождения;
- определение причины ошибки;
- исправление ошибки.

Можно выделить следующие типы ошибок:

- синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка;
- семантические ошибки — являются логическими и приводят к тому, что программа запускается, отработывает, но не даёт желаемого результата;
- ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают прерывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль).

Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга.

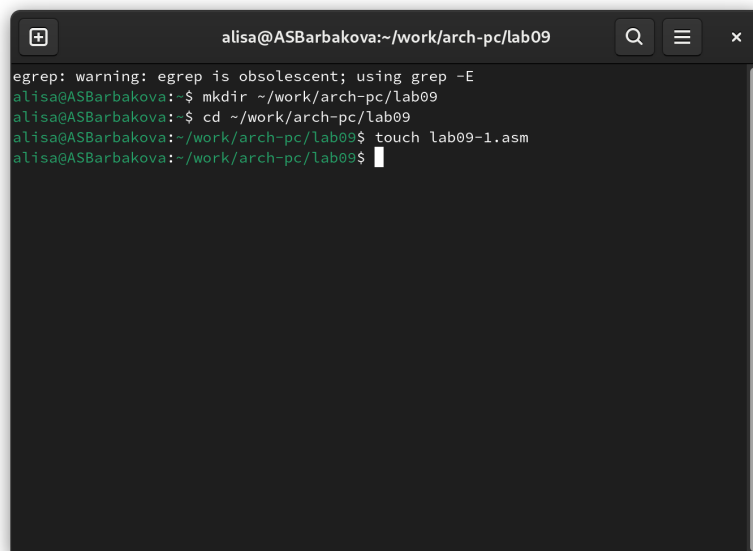
Третий этап — выяснение причины ошибки. После определения место-

нахождения 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.

4 Выполнение лабораторной работы

4.1 Релазиация подпрограмм в NASM

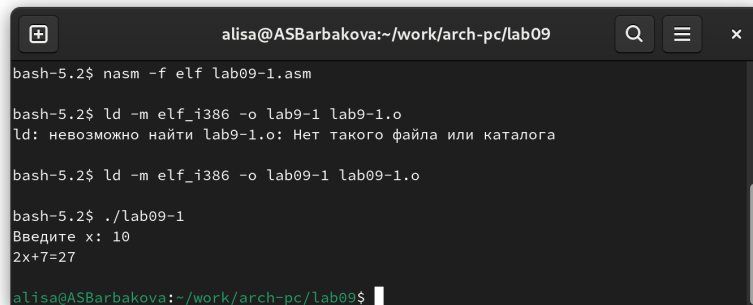
Создаю каталог для выполнения лабораторной работы №9 (рис. 4.1).



```
alisa@ASBarbakova:~/work/arch-pc/lab09
egrep: warning: egrep is obsolescent; using grep -E
alisa@ASBarbakova:~$ mkdir ~/work/arch-pc/lab09
alisa@ASBarbakova:~$ cd ~/work/arch-pc/lab09
alisa@ASBarbakova:~/work/arch-pc/lab09$ touch lab09-1.asm
alisa@ASBarbakova:~/work/arch-pc/lab09$
```

Рис. 4.1: Создание рабочего каталога

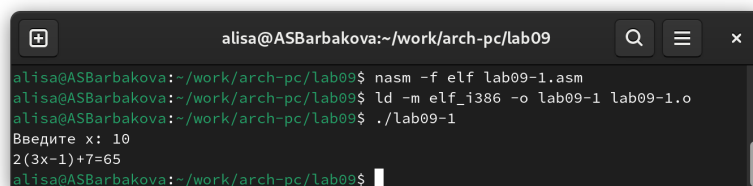
Копирую в файл код из листинга, компилирую и запускаю его, данная программа выполняет вычисление функции (рис. -fig. 4.2).



```
alisa@ASBarbakova:~/work/arch-pc/lab09
bash-5.2$ nasm -f elf lab09-1.asm
bash-5.2$ ld -m elf_i386 -o lab9-1 lab9-1.o
ld: невозможно найти lab9-1.o: Нет такого файла или каталога
bash-5.2$ ld -m elf_i386 -o lab09-1 lab09-1.o
bash-5.2$ ./lab09-1
Введите x: 10
2x+7=27
alisa@ASBarbakova:~/work/arch-pc/lab09$
```

Рис. 4.2: Запуск программы из листинга

Изменяю текст программы, добавив в неё подпрограмму, теперь она вычисляет значение функции для выражения $f(g(x))$ (рис. -fig. 4.3).



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

Рис. 4.3: Изменение программы первого листинга

Код программы:

```
%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 sprint
mov eax, [res]
call iprintLF
call quit
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:

```

```
push eax
call _subcalcul
```

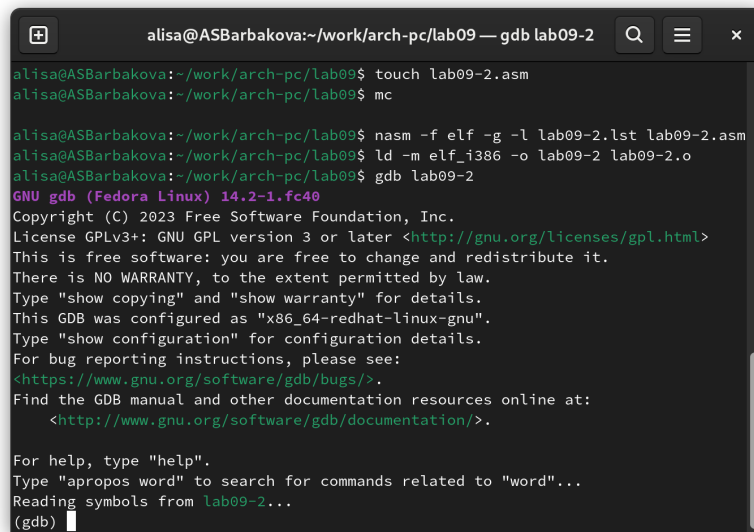
```
mov ebx, 2
mul ebx
add eax, 7
```

```
mov [res], eax
pop eax
ret
```

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

4.1.1 Отладка программ с помощью GDB

В созданный файл копирую программу второго листинга, транслирую с созданием файла листинга и отладки, компоную и запускаю в отладчике (рис. -fig. 4.4).

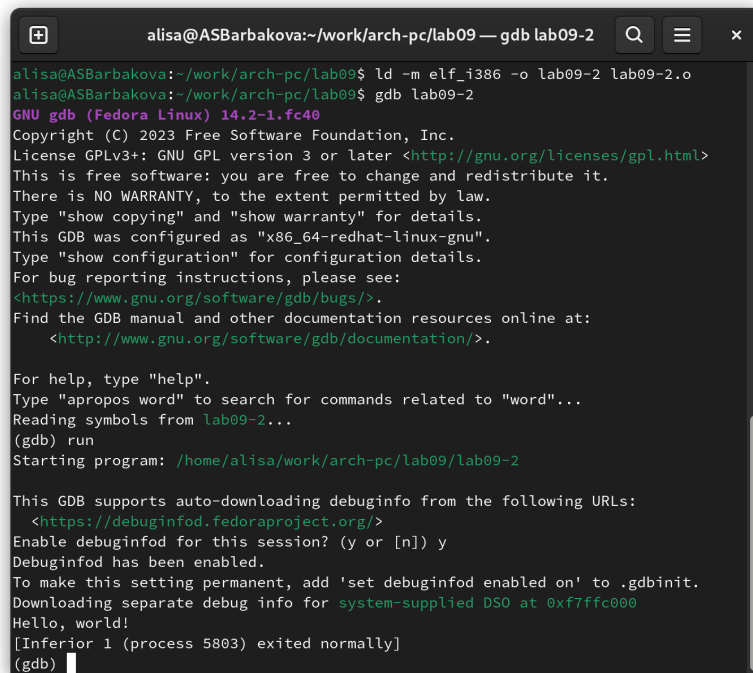


```
alisa@ASBarbakova:~/work/arch-pc/lab09 — gdb lab09-2
alisa@ASBarbakova:~/work/arch-pc/lab09$ touch lab09-2.asm
alisa@ASBarbakova:~/work/arch-pc/lab09$ mc
alisa@ASBarbakova:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
alisa@ASBarbakova:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
alisa@ASBarbakova:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 14.2-1.fc40
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 lab09-2...
(gdb) 
```

Рис. 4.4: Запуск программы в отладчике

Запустив программу командой `run`, я убедилась в том, что она работает исправно (рис. -fig. 4.5).



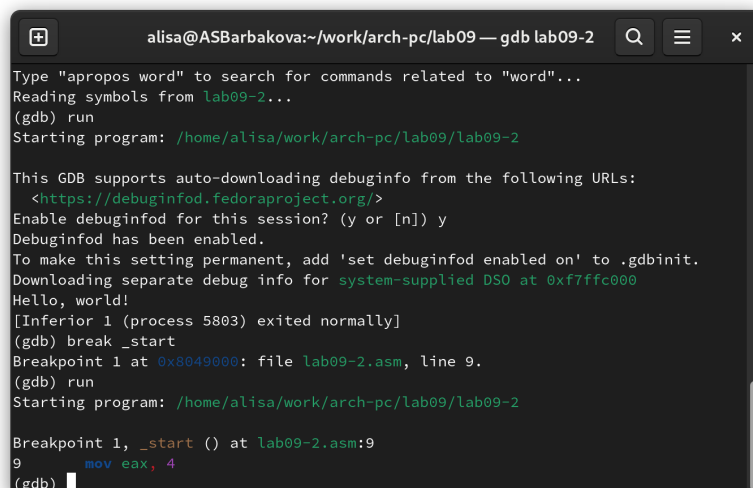
```
alisa@ASBarbakova:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
alisa@ASBarbakova:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 14.2-1.fc40
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is 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 lab09-2...
(gdb) run
Starting program: /home/alisa/work/arch-pc/lab09/lab09-2

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.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 5803) exited normally]
(gdb)
```

Рис. 4.5: Проверка программы отладчиком

Для более подробного анализа программы добавляю breakpoint на метку `_start` и снова запускаю отладку (рис. -fig. 4.6).



```
alisa@ASBarbakova:~/work/arch-pc/lab09 — gdb lab09-2
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...
(gdb) run
Starting program: /home/alisa/work/arch-pc/lab09/lab09-2

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.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 5803) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/alisa/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
(gdb) 
```

Рис. 4.6: Запуск отладчика с брейкпоинтом

Далее смотрю дисассимилированный код программы, перевожу на команд с синтаксисом Intel (рис. -fig. 4.7).

Различия между синтаксисом АТТ и Intel заключаются в порядке операндов (АТТ - Операнд источника указан первым. Intel - Операнд назначения указан первым), их размере (АТТ - размер операндов указывается явно с помощью суффиксов, непосредственные операнды предваряются символом \$; Intel - Размер операндов неявно определяется контекстом, как ах, еах, непосредственные операнды пишутся напрямую), именах регистров (АТТ - имена регистров предваряются символом %, Intel - имена регистров пишутся без префиксов).

```
alisa@ASBarbakova:~/work/arch-pc/lab09 — gdb lab09-2
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:    mov     $0x4,%eax
0x08049005 <+5>:    mov     $0x1,%ebx
0x0804900a <+10>:   mov     $0x804a000,%ecx
0x0804900f <+15>:   mov     $0x8,%edx
0x08049014 <+20>:   int     $0x80
0x08049016 <+22>:   mov     $0x4,%eax
0x0804901b <+27>:   mov     $0x1,%ebx
0x08049020 <+32>:   mov     $0x804a008,%ecx
0x08049025 <+37>:   mov     $0x7,%edx
0x0804902a <+42>:   int     $0x80
0x0804902c <+44>:   mov     $0x1,%eax
0x08049031 <+49>:   mov     $0x0,%ebx
0x08049036 <+54>:   int     $0x80
End of assembler dump.
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:    mov     eax,0x4
0x08049005 <+5>:    mov     ebx,0x1
0x0804900a <+10>:   mov     ecx,0x804a000
0x0804900f <+15>:   mov     edx,0x8
0x08049014 <+20>:   int     0x80
0x08049016 <+22>:   mov     eax,0x4
0x0804901b <+27>:   mov     ebx,0x1
0x08049020 <+32>:   mov     ecx,0x804a008
0x08049025 <+37>:   mov     edx,0x7
0x0804902a <+42>:   int     0x80
0x0804902c <+44>:   mov     eax,0x1
0x08049031 <+49>:   mov     ebx,0x0
0x08049036 <+54>:   int     0x80
End of assembler dump.
(gdb) 
```

Рис. 4.7: Дисассимилирование программы

Включаю режим псевдографики для более удобного анализа программы (рис. -fig. 4.8).

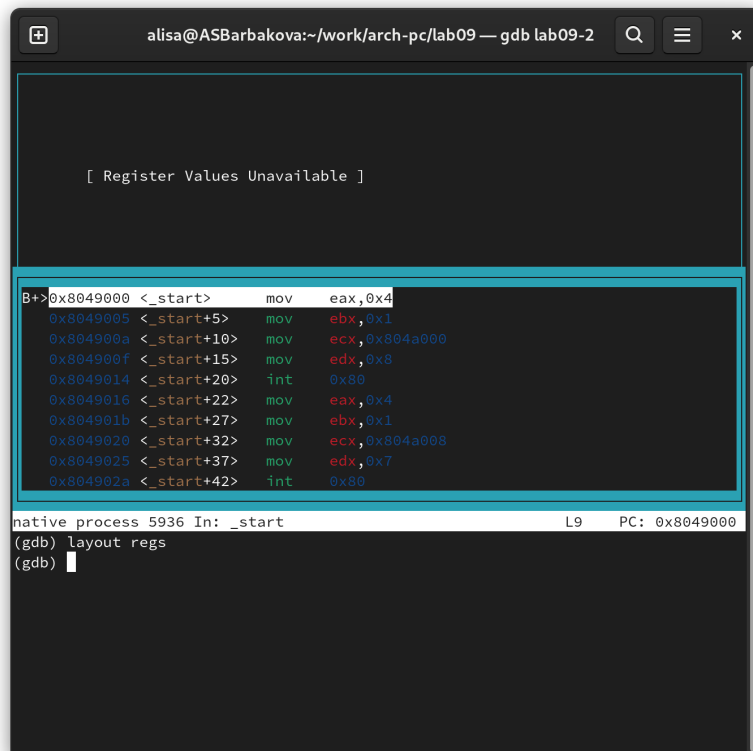


Рис. 4.8: Режим псевдографики

4.1.2 Добавление точек останова

Проверяю в режиме псевдографики, что breakpoint сохранился (рис. 4.9).


```
alisa@ASBarbakova:~/work/arch-pc/lab09 — gdb lab09-2
[ Register Values Unavailable ]

0x804901b <_start+27> mov    ebx,0x1
0x8049020 <_start+32> mov    ecx,0x804a008
0x8049025 <_start+37> mov    edx,0x7
0x804902a <_start+42> int     0x80
0x804902c <_start+44> mov    eax,0x1
b+ 0x8049031 <_start+49> mov    ebx,0x0
0x8049036 <_start+54> int     0x80

native process 5936 In: _start L9 PC: 0x8049000
(gdb) break *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) 
```

Рис. 4.9: Список брейкпоинтов

Устанавливаю еще одну точку остановки по адресу инструкции (рис. -fig. 4.10).

The screenshot shows a GDB terminal window with the title bar "alisa@ASBarbakova:~/work/arch-pc/lab09 — gdb lab09-2". The main display area shows assembly code with addresses and instructions. A light blue box highlights a section of the code starting at address 0x8049020. Below the assembly code, the status bar indicates "native process 5936 In: _start L9 PC: 0x8049000". The command prompt "(gdb)" is followed by the command "break *0x8049031", which sets a breakpoint at address 0x8049031. The output shows "Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20." followed by the command "(gdb) i b", which lists the breakpoints. The output shows two breakpoints: one at 0x8049000 (line 9) and one at 0x8049031 (line 20). The second breakpoint is marked as "already hit 1 time".

```
[ Register Values Unavailable ]

0x804901b <_start+27> mov     ebx,0x1
0x8049020 <_start+32> mov     ecx,0x804a008
0x8049025 <_start+37> mov     edx,0x7
0x804902a <_start+42> int     0x80
0x804902c <_start+44> mov     eax,0x1
b+ 0x8049031 <_start+49> mov     ebx,0x0
0x8049036 <_start+54> int     0x80

native process 5936 In: _start L9 PC: 0x8049000
(gdb) break *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)
```

Рис. 4.10: Добавление второй точки остановки

4.1.3 Работа с данными программы в GDB

Просматриваю содержимое регистров командой `info registers(i r)` (рис. -fig. 4.11).

The screenshot shows a GDB window titled "alisa@ASBarbakova:~/work/arch-pc/lab09 — gdb lab09-2". The window is divided into two main sections. The top section, titled "[Register Values Unavailable]", is currently empty. The bottom section displays assembly code for a native process (ID 5936) at the start of the program (In: _start). The assembly code is as follows:

```
0x804901b <_start+27> mov     ebx,0x1
0x8049020 <_start+32> mov     ecx,0x804a008
0x8049025 <_start+37> mov     edx,0x7
0x804902a <_start+42> int     0x80
0x804902c <_start+44> mov     eax,0x1
b+ 0x8049031 <_start+49> mov     ebx,0x0
0x8049036 <_start+54> int     0x80
```

Below the assembly code, the register values are listed:

Register	Value
eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xffffd0f0
ebp	0x0
esi	0x0
edi	0x0
eip	0x8049000

The PC (Program Counter) is 0x8049000. The instruction pointer (eip) is 0x8049000, which corresponds to the instruction at address 0x8049000, labeled as "<_start>". The instruction pointer is currently pointing to the instruction at address 0x8049000, which is the first instruction in the list shown.

Рис. 4.11: Просмотр содержимого регистров

Смотрю содержимое переменных по имени и по адресу (рис. -fig. 4.12).

```
alisa@ASBarbakova:~/work/arch-pc/lab09 — gdb lab09-2
Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd0f0 0xffffd0f0
ebp      0x0      0x0
esi      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

native process 5936 In: _start L10 PC: 0x8049005
Quit
(gdb) layout split
(gdb) layout asm
(gdb) layout regs
(gdb) si
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb) 
```

Рис. 4.12: Просмотр содержимого переменных двумя способами

Меняю содержимое переменных по имени и по адресу(H – h, W – T)
(рис. -fig. 4.13).

```
alisa@ASBarbakova:~/work/arch-pc/lab09 — gdb lab09-2
Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd0f0 0xffffd0f0
ebp      0x0      0
esi      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

native process 5936 In: _start L10 PC: 0x8049005
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) set {char}&msg2='T'
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "Torld!\n\034"
(gdb) 
```

Рис. 4.13: Изменение содержимого переменных двумя способами

Вывожу в различных форматах значение регистра edx (рис. -fig. 4.14).

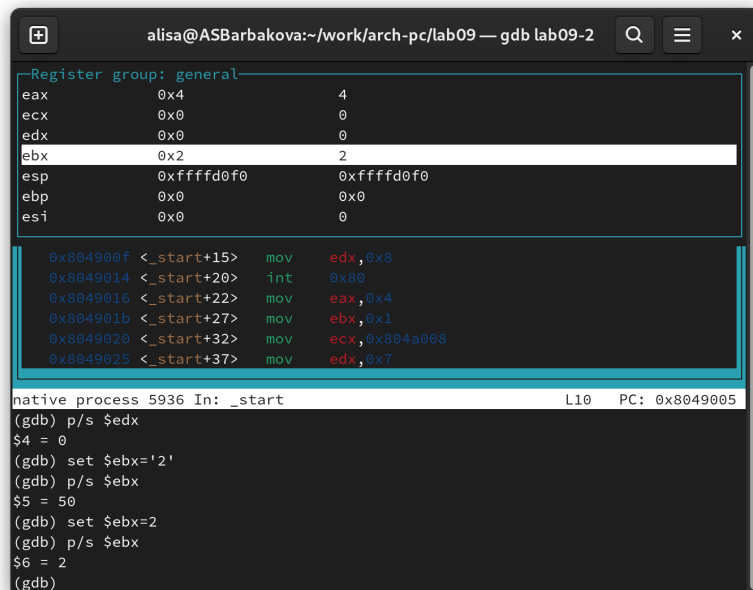
```
alisa@ASBarbakova:~/work/arch-pc/lab09 — gdb lab09-2
Register group: general
eax      0x1      1
ecx      0x804a008 134520840
edx      0x7      7
ebx      0x1      1
esp      0xffffd0f0 0xffffd0f0
ebp      0x0      0x0
esi      0x0      0

0x8049025 <_start+37> mov     edx,0x7
0x804902a <_start+42> int     0x80
0x804902c <_start+44> mov     eax,0x1
B->0x8049031 <_start+49> mov     ebx,0x0
0x8049036 <_start+54> int     0x80
0x8049038 add     BYTE PTR [eax],al
0x804903a add     BYTE PTR [eax],al

native process 5936 In: _start L20 PC: 0x8049031
(gdb) p/t $ecx
$7 = 1000000001001010000000001000
(gdb) p/s $edx
$8 = 7
(gdb) p/t $edx
$9 = 111
(gdb) p/x $edx
$10 = 0x7
(gdb)
```

Рис. 4.14: Просмотр значения регистра разными представлениями

С помощью команды `set` меняю содержимое регистра `ebx` (рис. -fig. 4.15).



The screenshot shows a GDB terminal window with the title bar "alisa@ASBarbakova:~/work/arch-pc/lab09 — gdb lab09-2". The main content is divided into two sections. The top section, titled "Register group: general", displays the values of several registers:

eax	0x4	4
ecx	0x0	0
edx	0x0	0
ebx	0x2	2
esp	0xffffd0f0	0xffffd0f0
ebp	0x0	0x0
esi	0x0	0

. The bottom section shows assembly code with addresses and instructions:

```
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
```

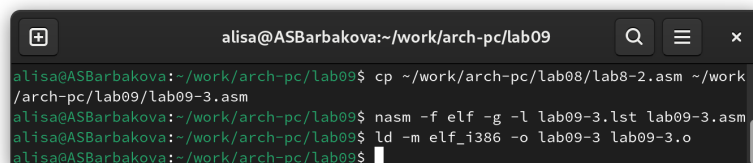
. Below the assembly code, the status bar indicates "native process 5936 In: _start" and "L10 PC: 0x8049005". The bottom part of the window shows GDB commands and their output:

```
(gdb) p/s $edx
$4 = 0
(gdb) set $ebx='2'
(gdb) p/s $ebx
$5 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$6 = 2
(gdb)
```

Рис. 4.15: Примеры использования команды set

4.1.4 Обработка аргументов командной строки в GDB

Копирую программу из предыдущей лабораторной работы в текущий каталог и создаю исполняемый файл с файлом листинга и отладки (рис. -fig. 4.16).



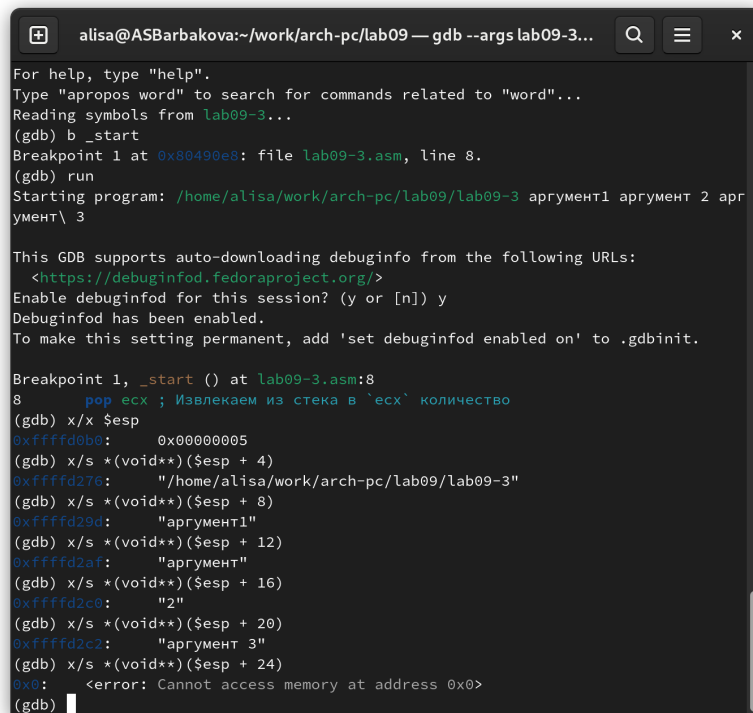
The screenshot shows a terminal window with the title bar "alisa@ASBarbakova:~/work/arch-pc/lab09". The terminal displays the following commands and their output:

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

Рис. 4.16: Подготовка новой программы

Запускаю программу с режиме отладки с указанием аргументов, ука-

зываю и запускаю отладку. Проверяю работу стека, изменяя аргумент команды просмотра регистра `esp` на `+4`, число обусловлено разрядностью системы, а указатель `void` занимает как раз 4 байта. Ошибка при аргументе `+24` означает, что аргументы на вход программы закончились. (рис. -fig. 4.17).



```
alisa@ASBarbakova:~/work/arch-pc/lab09 — gdb --args lab09-3...
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 8.
(gdb) run
Starting program: /home/alisa/work/arch-pc/lab09/lab09-3 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 lab09-3.asm:8
8      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffd0b8: 0x00000005
(gdb) x/s *(void*)($esp + 4)
0xffffd276: "/home/alisa/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void*)($esp + 8)
0xffffd29d: "аргумент1"
(gdb) x/s *(void*)($esp + 12)
0xffffd2af: "аргумент"
(gdb) x/s *(void*)($esp + 16)
0xffffd2c0: "2"
(gdb) x/s *(void*)($esp + 20)
0xffffd2c2: "аргумент 3"
(gdb) x/s *(void*)($esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 4.17: Проверка работы стека

4.2 Задание для самостоятельной работы

1. Меняю программу самостоятельной части лабораторной работы №8(вариант 8) с использованием подпрограммы (рис. -fig. 4.18).


```
lab09-4.asm [----] 7 L: [ 11+30 41/ 41] *(549 / 549b) <EOF> [*] [X]
mov eax, msg_func
call sprintf

pop ecx
pop edx
sub ecx, 1
mov esi, 0

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

add esi, eax
loop next

_end:
mov eax, msg_result
call sprintf
mov eax, esi
call iprintLF
call quit
....
_calculate_fx:
mov ebx, 2
mul ebx
add eax, 7
ret
```

Рис. 4.18: Измененная программа предыдущей лабораторной работы

Код программы:

```
%include 'in_out.asm'

.
SECTION .data.
msg_func db "Функция: f(x) = 7 + 2x", 0.
msg_result db "Результат: ", 0.

SECTION .text.
GLOBAL _start.

_start:
```

```
mov eax, msg_func
```

```
call sprintf
```

```
pop ecx
```

```
pop edx
```

```
sub ecx, 1
```

```
mov esi, 0
```

```
next:
```

```
cmp ecx, 0h
```

```
jz _end
```

```
pop eax
```

```
call atoi
```

```
....
```

```
call _calculate_fx
```

```
add esi, eax
```

```
loop next
```

```
_end:
```

```
mov eax, msg_result
```

```
call sprintf
```

```
mov eax, esi
```

```
call iprintLF
```

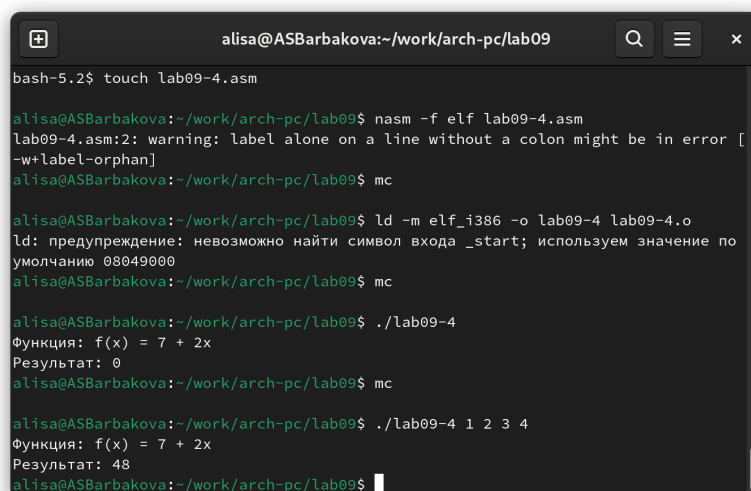
```
call quit
```

```
....
```

```
_calculate_fx:
```

```
mov ebx, 2
mul ebx
add eax, 7
ret
```

Запускаю исполняемый файл, вижу, что программа работает корректно (рис. -fig. 4.19).



```
alisa@ASBarbakova:~/work/arch-pc/lab09
bash-5.2$ touch lab09-4.asm

alisa@ASBarbakova:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
lab09-4.asm:2: warning: label alone on a line without a colon might be in error [-w+label-orphan]
alisa@ASBarbakova:~/work/arch-pc/lab09$ mc

alisa@ASBarbakova:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
ld: предупреждение: невозможно найти символ входа _start; используем значение по умолчанию 08049000
alisa@ASBarbakova:~/work/arch-pc/lab09$ mc

alisa@ASBarbakova:~/work/arch-pc/lab09$ ./lab09-4
Функция: f(x) = 7 + 2x
Результат: 0
alisa@ASBarbakova:~/work/arch-pc/lab09$ mc

alisa@ASBarbakova:~/work/arch-pc/lab09$ ./lab09-4 1 2 3 4
Функция: f(x) = 7 + 2x
Результат: 48
alisa@ASBarbakova:~/work/arch-pc/lab09$
```

Рис. 4.19: Работа программы

2. Запускаю программу в режиме отладчика и через si просматриваю изменение значений регистров через i r. При выполнении инструкции mul esx можно заметить, что результат умножения записывается в регистр eax, но также меняет и edx. Значение регистра ebx не обновляется напрямую, поэтому результат программы неверно подсчитывает функцию (рис. -fig. 4.20).

```
alisa@ASBarbakova:~/work/arch-pc/lab09 — gdb lab09-5
Register group: general
eax      0x2      2
ecx      0x0      0
edx      0x0      0
ebx      0x3      3
esp      0xffffd0f0 0xffffd0f0

B+ 0x80490e8 <_start>   mov     $0x3,%ebx
0x80490ed <_start+5>   mov     $0x2,%eax
>0x80490f2 <_start+10> add     %eax,%ebx
0x80490f4 <_start+12>   mov     $0x4,%ecx
0x80490f9 <_start+17>   mul     %ecx

native process 16049 In: _start L10 PC: 0x80490f2

Inferior 1 [process 16049] will be killed.

Quit anyway? (y or n) n
Not confirmed.
(gdb) si
(gdb)
```

Рис. 4.20: Поиск ошибки в программе через пошаговую отладку

Исправляю найденную ошибку, теперь программа верно считает значение функции (рис. -fig. 4.21).

```
alisa@ASBarbakova:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
alisa@ASBarbakova:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab09-5.o
alisa@ASBarbakova:~/work/arch-pc/lab09$ ./lab09-5
Результат: 25
alisa@ASBarbakova:~/work/arch-pc/lab09$
```

Рис. 4.21: Проверка корректировок в программе

Код измененной программы:

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

```
SECTION .data
```

```
div: DB 'Результат: ', 0
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
mov ebx, 3
```

```
mov eax, 2
```

```
add ebx, eax
```

```
mov eax, ebx
```

```
mov ecx, 4
```

```
mul ecx
```

```
add eax, 5
```

```
mov edi, eax
```

```
mov eax, div
```

```
call sprint
```

```
mov eax, edi
```

```
call iprintLF
```

```
call quit
```

5 Выводы

В результате выполнения данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм, а так же познакомилась с методами отладки при помощи GDB и с его основными возможностями.

Список литературы

1. Курс на ТУИС
2. Лабораторная работа №9