

Отчет по лабораторной работе №9

Дисциплина: архитектура компьютера

Шабакова Карина Баировна

Содержание

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. 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 местонахождения ошибки обычно проще определить причину неправильной работы программы.

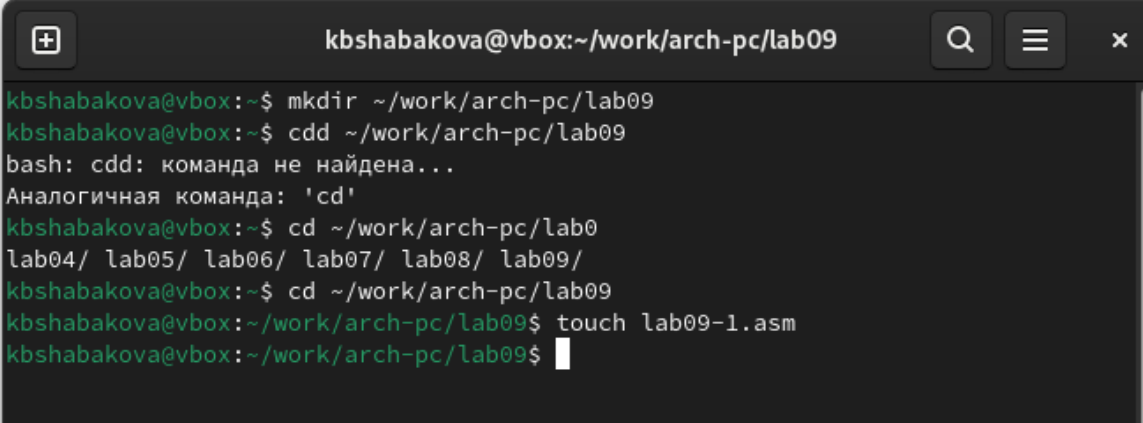
Последний этап — исправление ошибки. После этого при повторном запуске

программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново.

4 Выполнение лабораторной работы

4.1 Релаксация подпрограмм в NASM

Со
зда
ю
ка
та
ло
г
дл
я
вы
по
лн
ен
ия

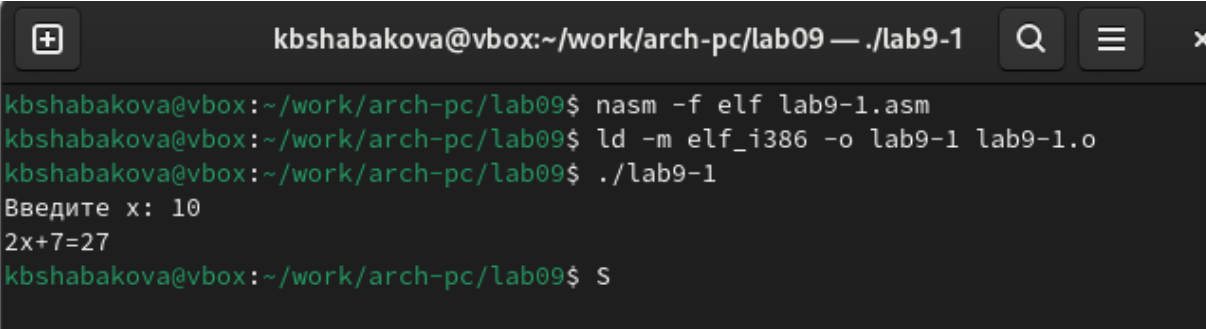


```
kbshabakova@vbox:~/work/arch-pc/lab09
kbshabakova@vbox:~$ mkdir ~/work/arch-pc/lab09
kbshabakova@vbox:~$ cdd ~/work/arch-pc/lab09
bash: cdd: команда не найдена...
Аналогичная команда: 'cd'
kbshabakova@vbox:~$ cd ~/work/arch-pc/lab0
lab04/ lab05/ lab06/ lab07/ lab08/ lab09/
kbshabakova@vbox:~$ cd ~/work/arch-pc/lab09
kbshabakova@vbox:~/work/arch-pc/lab09$ touch lab09-1.asm
kbshabakova@vbox:~/work/arch-pc/lab09$
```

лабораторной работы №9 (рис. 1).

Рис. 1: Создание рабочего каталога

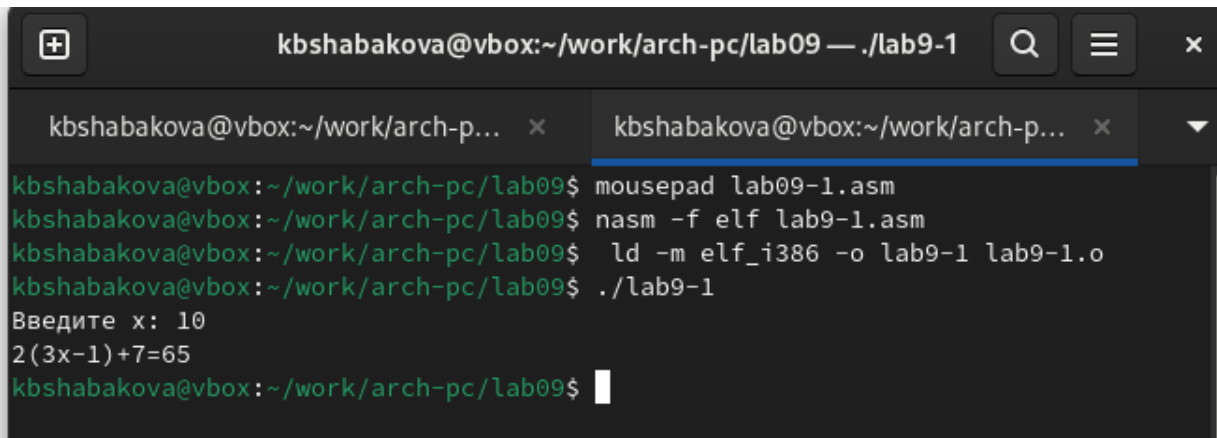
Копирую в файл код из листинга, компилирую и запускаю его, данная программа выполняет вычисление функции (рис. 2).



```
kbshabakova@vbox:~/work/arch-pc/lab09 — ./lab9-1
kbshabakova@vbox:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
kbshabakova@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
kbshabakova@vbox:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 10
2x+7=27
kbshabakova@vbox:~/work/arch-pc/lab09$ S
```

Рис. 2: Запуск программы из листинга

Изменяю текст программы, добавив в нее подпрограмму, теперь она вычисляет значение функции для выражения $f(g(x))$ (рис. 3).



```
kbshabakova@vbox:~/work/arch-pc/lab09 — ./lab9-1
kbshabakova@vbox:~/work/arch-pc/lab09$ mousepad lab09-1.asm
kbshabakova@vbox:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
kbshabakova@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
kbshabakova@vbox:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 10
2(3x-1)+7=65
kbshabakova@vbox:~/work/arch-pc/lab09$
```

Рис. 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:
mov eax, msg
call sprint

mov ecx, x
mov edx, 80
call sread

mov eax, x
call atoi

call _calcul

mov eax, result
call sprint
mov eax, [res]
call iprintLF

call quit

_calcul:
```

```

push eax
call _subcalcul

mov ebx, 2
mul ebx
add eax, 7

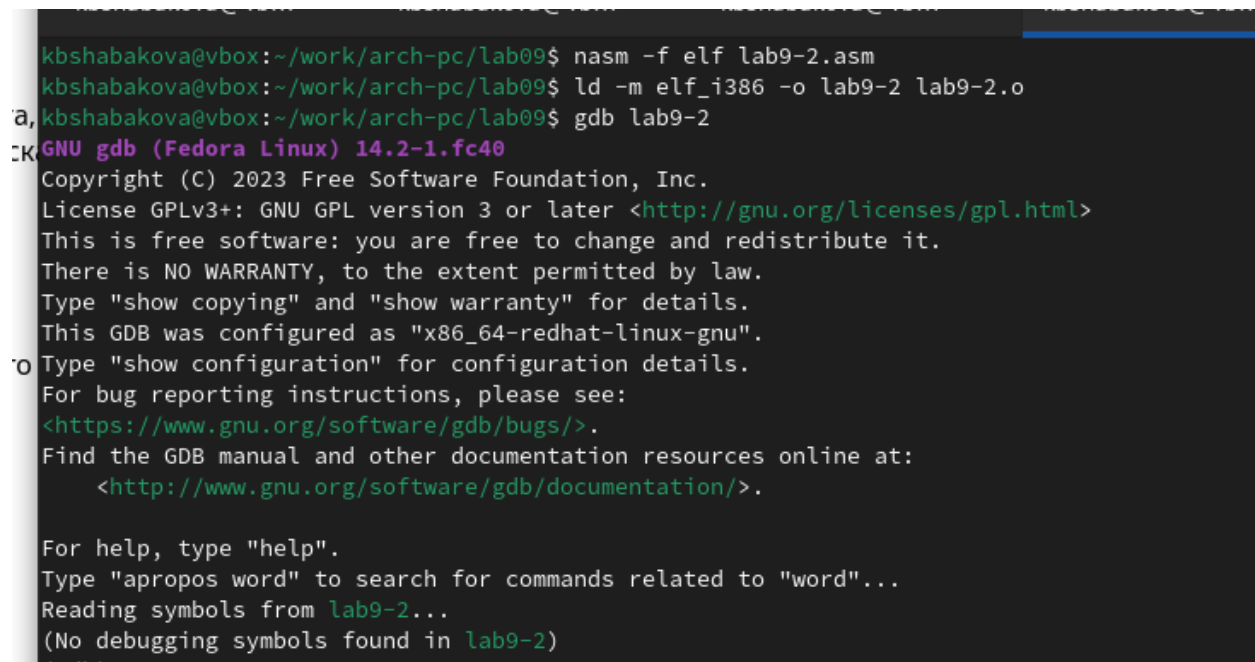
mov [res], eax
pop eax
ret

_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, компону и запускаю в отладчике (рис. 4).



```

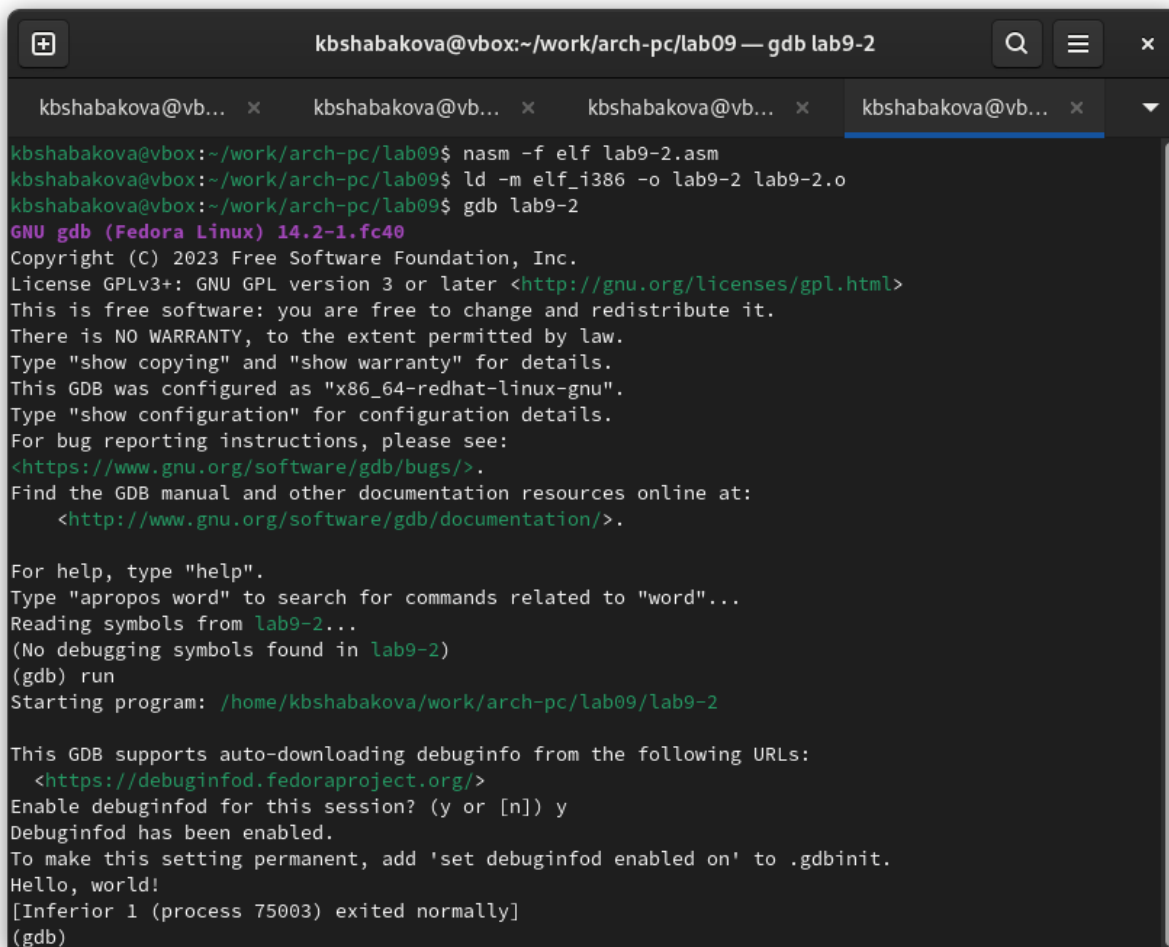
kbshabakova@vbox:~/work/arch-pc/lab09$ nasm -f elf lab9-2.asm
kbshabakova@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
a, kbshabakova@vbox:~/work/arch-pc/lab09$ gdb lab9-2
GNU gdb (Fedora Linux) 14.2-1.fc40
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 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 lab9-2...
(No debugging symbols found in lab9-2)

```

Рис. 4: Запуск программы в отладчике

Запустив программу командой run, я убедилась в том, что она работает исправно (рис. 5).



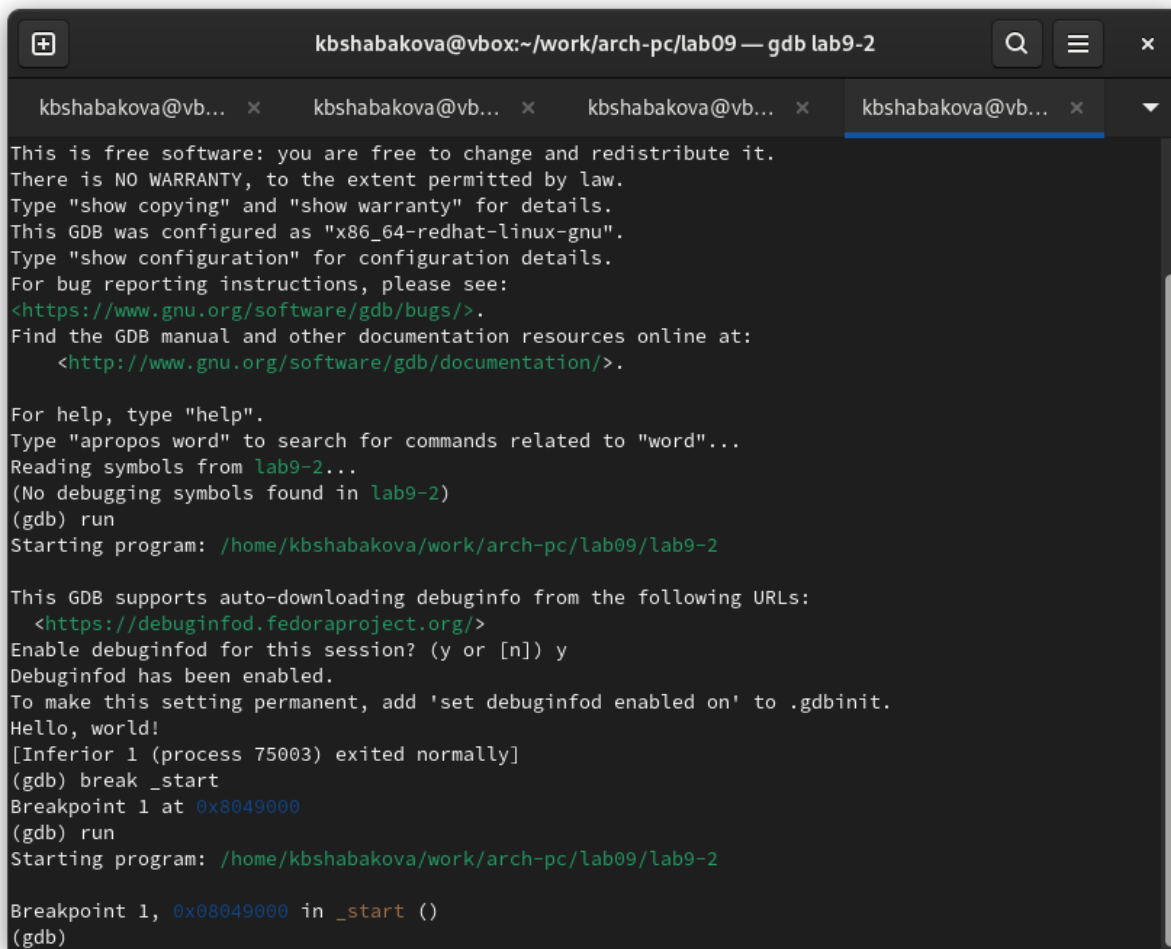
```
kbshabakova@vbox:~/work/arch-pc/lab09 — gdb lab9-2
kbshabakova@vbox:~/work/arch-pc/lab09$ nasm -f elf lab9-2.asm
kbshabakova@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
kbshabakova@vbox:~/work/arch-pc/lab09$ gdb lab9-2
GNU gdb (Fedora Linux) 14.2-1.fc40
Copyright (C) 2023 Free Software 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 lab9-2...
(No debugging symbols found in lab9-2)
(gdb) run
Starting program: /home/kbshabakova/work/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]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Hello, world!
[Inferior 1 (process 75003) exited normally]
(gdb)
```

Рис. 5: Проверка программы отладчиком

Для более подробного анализа программы добавляю брейкпоинт на метку `_start` и снова запускаю отладку (рис. 6).



```
kbshabakova@vbox:~/work/arch-pc/lab09 — gdb lab9-2
kbshabakova@vb... x kbshabakova@vb... x kbshabakova@vb... x kbshabakova@vb... x
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...
(No debugging symbols found in lab9-2)
(gdb) run
Starting program: /home/kbshabakova/work/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]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Hello, world!
[Inferior 1 (process 75003) exited normally]
(gdb) break _start
Breakpoint 1 at 0x08049000
(gdb) run
Starting program: /home/kbshabakova/work/arch-pc/lab09/lab9-2

Breakpoint 1, 0x08049000 in _start ()
(gdb)
```

Рис. 6: Запуск отладчика с брейкпоинтом

Далее смотрю дисассимилированный код программы, перевожу на команд с синтаксисом Intel amd топчик (рис. 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 ah, eax, непосредственные операнды пишутся напрямую), именах регистров (АТТ - имена регистров предваряются символом %, Intel - имена регистров пишутся без префиксов).

```

11      mov eax, 4
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:      mov     $0x4,%eax
0x08049005 <+5>:      mov     $0x1,%ebx
0x0804900a <+10>:     mov     $0x804a000,%ecx
0x0804900f <+15>:     mov     $0x8,%edx
0x08049014 <+20>:     int     $0x80
0x08049016 <+22>:     mov     $0x4,%eax
0x0804901b <+27>:     mov     $0x1,%ebx
0x08049020 <+32>:     mov     $0x804a008,%ecx
0x08049025 <+37>:     mov     $0x7,%edx
0x0804902a <+42>:     int     $0x80
0x0804902c <+44>:     mov     $0x1,%eax
0x08049031 <+49>:     mov     $0x0,%ebx
0x08049036 <+54>:     int     $0x80
End of assembler dump.
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:      mov     eax,0x4
0x08049005 <+5>:      mov     ebx,0x1
0x0804900a <+10>:     mov     ecx,0x804a000
0x0804900f <+15>:     mov     edx,0x8
0x08049014 <+20>:     int     0x80
0x08049016 <+22>:     mov     eax,0x4
0x0804901b <+27>:     mov     ebx,0x1
0x08049020 <+32>:     mov     ecx,0x804a008
0x08049025 <+37>:     mov     edx,0x7
0x0804902a <+42>:     int     0x80
0x0804902c <+44>:     mov     eax,0x1
0x08049031 <+49>:     mov     ebx,0x0
0x08049036 <+54>:     int     0x80
End of assembler dump.
(gdb)

```

Рис. 7: Дисассимилирование программы

Включаю режим псевдографики для более удобного анализа программы (рис. 8).

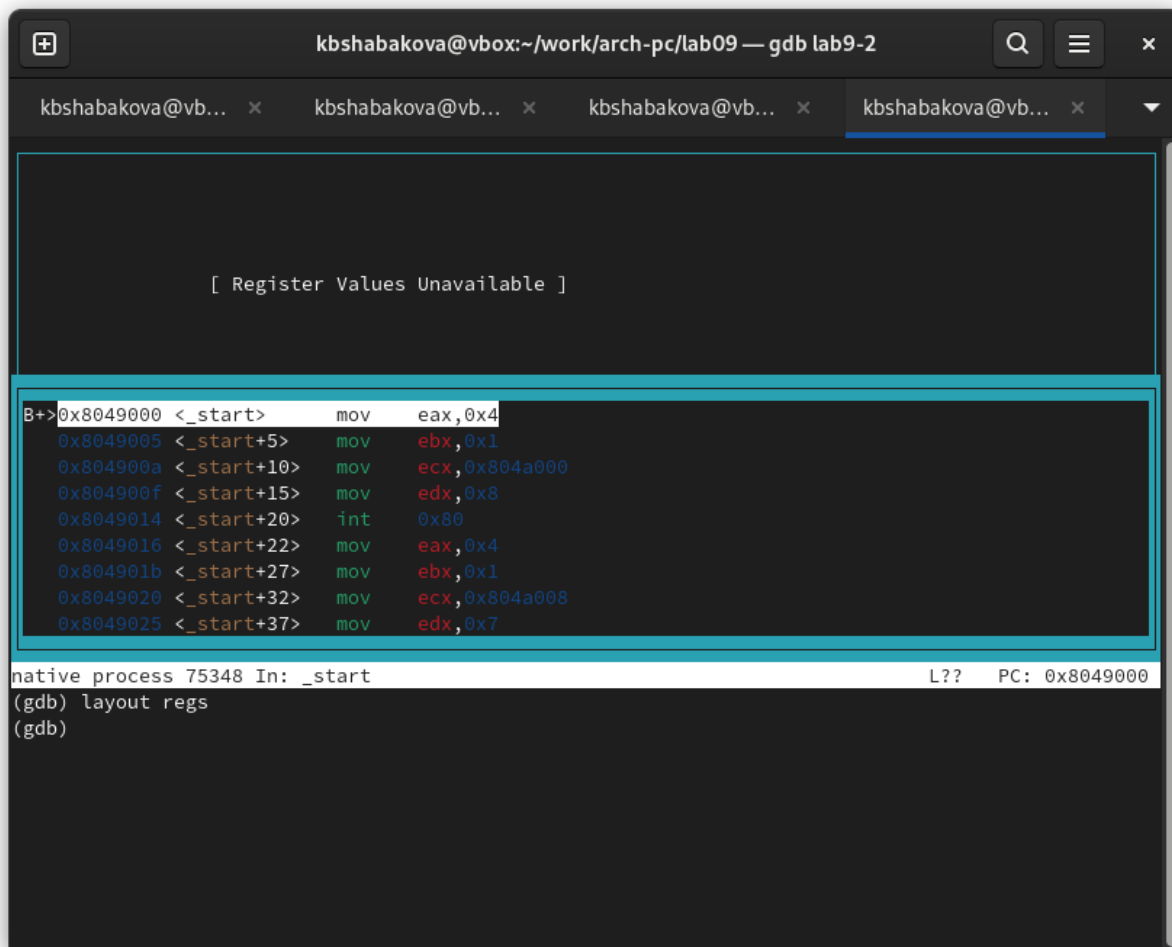


Рис. 8: Режим псевдографики

4.1.2 Добавление точек останова

Проверяю в режиме псевдографики, что брейкпоинт сохранился (рис. 9).


```
kbshabakova@vbox:~/work/arch-pc/lab09 — gdb lab9-2
[ 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
0x8049038          add     BYTE PTR [eax],al
0x804903a          add     BYTE PTR [eax],al

native process 75348 In: _start L?? PC: 0x8049000
1 breakpoint keep y 0x08049000 <_start>
  breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031
(gdb) i b
Num    Type           Disp Enb Address      What
1      breakpoint      keep y 0x08049000 <_start>
      breakpoint already hit 1 time
2      breakpoint      keep y 0x08049031 <_start+49>
(gdb)
```

Рис. 9: Список брейкпоинтов

Устанавливаю еще одну точку останова по адресу инструкции (рис. 10).

The screenshot shows a GDB terminal window with the title bar "kbshabakova@vbox:~/work/arch-pc/lab09 — gdb lab9-2". The window contains several tabs, with the active one showing assembly code. A red box highlights a section of assembly code starting from address 0x804901b. Below the assembly code, the terminal shows the status of a native process (75348) and the management of two breakpoints. The first breakpoint is at address 0x8049000, and the second is at address 0x8049031. The terminal prompt is "(gdb)".

```
[ 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
0x8049038             add     BYTE PTR [eax],al
0x804903a             add     BYTE PTR [eax],al

native process 75348 In: _start                               L??  PC: 0x8049000
1      breakpoint      keep y   0x08049000 <_start>
      breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031
(gdb) i b
Num    Type             Disp Enb Address      What
1      breakpoint      keep y   0x08049000 <_start>
      breakpoint already hit 1 time
2      breakpoint      keep y   0x08049031 <_start+49>
(gdb)
```

Рис. 10: Добавление второй точки останова

4.1.3 Работа с данными программы в GDB

Просматриваю содержимое регистров командой `info registers` (рис. 11).

The screenshot shows a GDB terminal window with the title bar "kbshabakova@vbox:~/work/arch-pc/lab09 — gdb lab9-2". The window contains a list of assembly instructions and a register dump. A red box highlights the assembly instructions, and a blue box highlights the register dump.

```
[ 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
0x8049038             add     BYTE PTR [eax],al
0x804903a             add     BYTE PTR [eax],al
```

```
native process 75348 In: _start L?? PC: 0x8049000
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd0b0 0xffffd0b0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 11: Просмотр содержимого регистров

Смотрю содержимое переменных по имени и по адресу (рис. 12).

The screenshot shows a GDB terminal window with the title bar "kbshabakova@vbox:~/work/arch-pc/lab09 — gdb lab9-2". The window contains two main sections. The top section, titled "Register group: general", displays the values of general-purpose registers:

Register	Value	Value
eax	0x4	4
ecx	0x0	0
edx	0x0	0
ebx	0x0	0
esp	0xffffd0b0	0xffffd0b0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0

. The bottom section displays assembly code starting with "B+ 0x8049000 <_start>". The code includes instructions like "mov eax, 0x4", "mov ebx, 0x1", "mov ecx, 0x804a000", "mov edx, 0x8", "int 0x80", and several more "mov" instructions. Below the assembly code, the GDB prompt shows the current state: "native process 75348 In: _start L?? PC: 0x8049005". It also shows a warning "'msg1' has unknown type; cast it to its declared type" and the command "(gdb) si". The output shows the instruction "0x08049005 in _start ()" and the command "(gdb) x/1sb & msg1". The output shows the memory address "0x804a000:" with the value "Hello, " and the command "(gdb) x/1sb 0x804008". The output shows the memory address "0x804008:" with the error message "<error: Cannot access memory at address 0x804008>" and the command "(gdb) x/1sb 0x804a008". The output shows the memory address "0x804a008:" with the value "world!\n" and the command "(gdb)".

Рис. 12: Просмотр содержимого переменных двумя способами

Меняю содержимое переменных по имени и по адресу (рис. 13).

```

Register group: general
eax      0x8      8      ecx      0x804a000      134520832
edx      0x8      8      ebx      0x1      1
esp      0xffffcf10      0xffffcf10      ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049016      0x8049016 <_start+22>      eflags      0x202      [ IF ]
cs       0x23      35      ss       0x2b      43
ds       0x2b      43      es       0x2b      43
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
0x8049020 <_start+32>     mov     ecx,0x804a008
0x8049025 <_start+37>     mov     edx,0x7
0x804902a <_start+42>     int     0x80

native process 5886 (asm) In: _start      L17      PC: 0x8049016
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "World!\n\034"
(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='x'
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "xorld!\n\034"
(gdb)

```

Рис. 13: Изменение содержимого переменных двумя способами

Вывожу в различных форматах значение регистра edx (рис. 14).

```

Register group: general
eax      0x8      8      ecx      0x804a000      134520832
edx      0x8      8      ebx      0x1      1
esp      0xffffd070      0xffffd070      ebp      0x0      0x0
esi      0x0      0      edi      0x0      0

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

native process 10469 (asm) In: _start      L15      PC: 0x8049016
(gdb) p/t $ecx
$2 = 1000000001001010000000000000
(gdb) p/s $edx
$3 = 8
(gdb) p/t $edx
$4 = 1000
(gdb) p/x $edx
$5 = 0x8
(gdb)

```

Рис. 14: Просмотр значения регистра разными представлениями

С помощью команды set меняю содержимое регистра ebx (рис. 15).

The screenshot shows a GDB terminal window with the title bar 'kbshabakova@vbox: ~/work/arch-pc/lab09 — gdb lab9-2'. The window contains several tabs, with the active one showing the GDB interface. The 'Register group: general' section is expanded, displaying the following register values:

Register	Value (Hex)	Value (Dec)
eax	0x8	8
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x2	2
esp	0xffffd0b0	0xffffd0b0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0

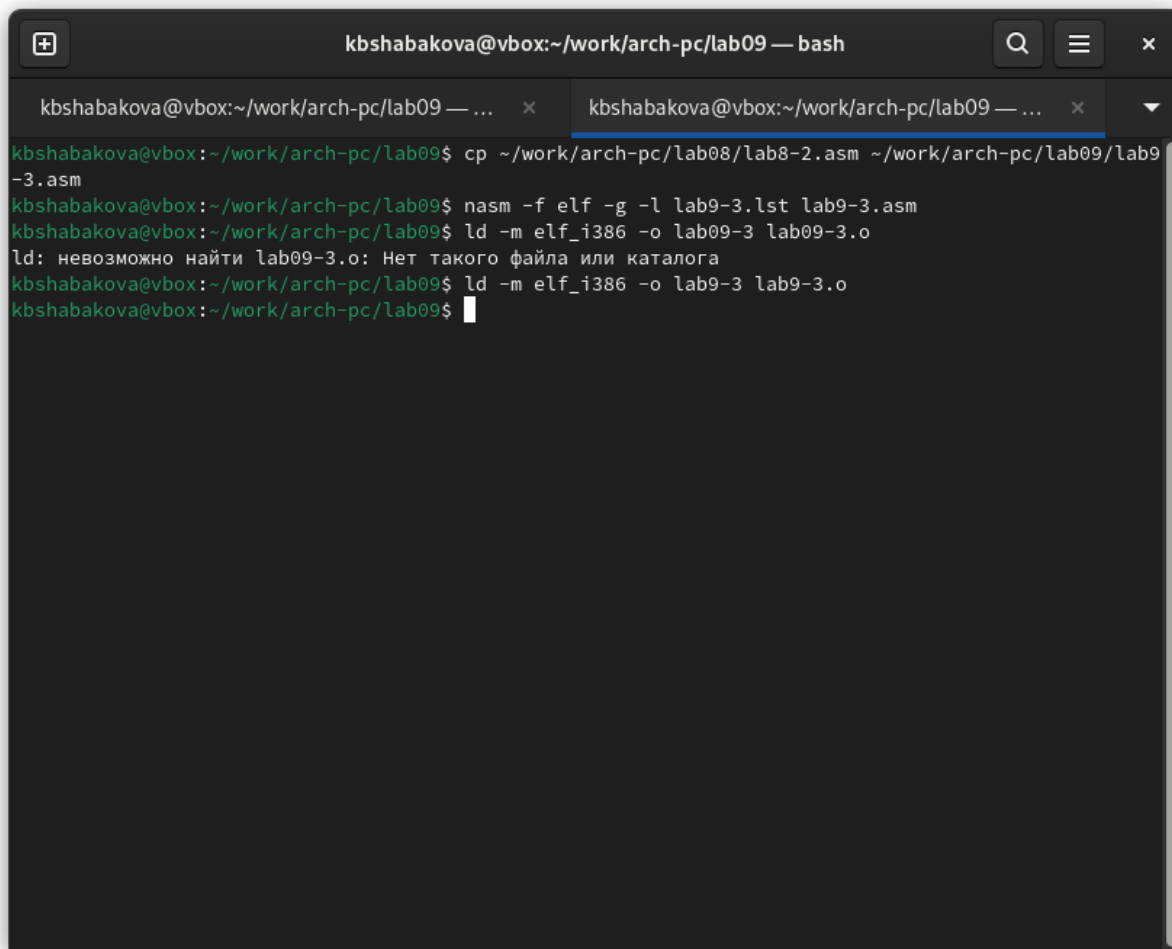
Below the register list, a list of assembly instructions is shown, all of which are 'add BYTE PTR [eax], al' instructions at various memory addresses from 0x80490b4 to 0x80490c4. The GDB console at the bottom shows the following commands and output:

```
native process 75348 In: _start L?? PC: 0x8049005
(gdb) set $ecx=134520832
(gdb) set $ebx='2'
(gdb) p/s
$9 = 0
(gdb) p/s $ebx
$10 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$11 = 2
(gdb)
```

Рис. 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. 16).

A terminal window with a dark background and light green text. The window title is 'kbshabakova@vbox:~/work/arch-pc/lab09 — bash'. It shows a series of commands and their outputs. The first command is 'cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab9-3.asm'. The second is 'nasm -f elf -g -l lab9-3.lst lab9-3.asm'. The third is 'ld -m elf_i386 -o lab09-3 lab09-3.o', which results in an error message in Russian: 'ld: невозможно найти lab09-3.o: Нет такого файла или каталога'. The fourth command is 'ld -m elf_i386 -o lab9-3 lab9-3.o', which appears to succeed. The prompt is currently at the end of the fourth command line.

```
kbshabakova@vbox:~/work/arch-pc/lab09 — bash
kbshabakova@vbox:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab9-3.asm
kbshabakova@vbox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
kbshabakova@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
ld: невозможно найти lab09-3.o: Нет такого файла или каталога
kbshabakova@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-3 lab9-3.o
kbshabakova@vbox:~/work/arch-pc/lab09$
```

Рис. 16: Подготовка новой программы

Запускаю программу с режиме отладки с указанием аргументов, указываю брейкпоинт 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. 17).

```
kbshabakova@vbox:~/work/arch-pc/lab09 — gdb --args lab9-3 arg1 arg2 arg3
kbshabakova@vbox:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab9-3.asm
kbshabakova@vbox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
kbshabakova@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab9-3.o
ld: невозможно найти lab09-3.o: Нет такого файла или каталога
kbshabakova@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-3 lab9-3.o
kbshabakova@vbox:~/work/arch-pc/lab09$ gdb --args lab9-3 arg1 arg2 'arg3'
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 lab9-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 7.
(gdb) run
Starting program: /home/kbshabakova/work/arch-pc/lab09/lab9-3 arg1 arg2 arg3

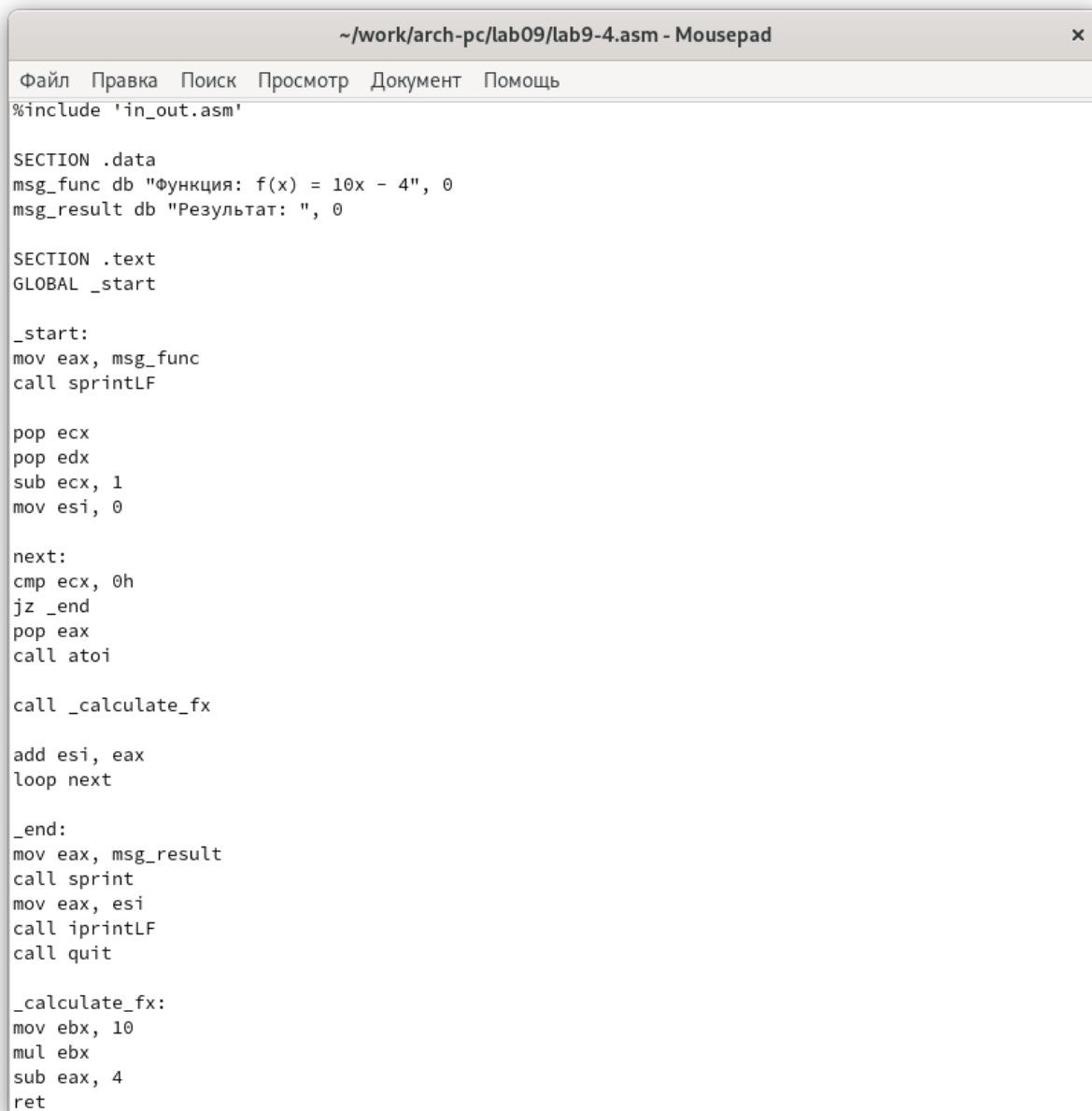
This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) y
```

```
7      pop ecx
(gdb) x/s *(void**)($esp + 4)
0xffffd0d5:  "/home/mazurskiy/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd0ff:  "arg1"
(gdb) x/s *(void**)($esp + 12)
0xffffd104:  "arg"
(gdb) x/s *(void**)($esp + 16)
0xffffd108:  "2"
(gdb) x/s *(void**)($esp + 20)
0xffffd10a:  "arg 3"
(gdb) x/s *(void**)($esp + 24)
0x0:  <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 17: Проверка работы стека

4.2 Задание для самостоятельной работы

1. Меняю программу самостоятельной части предыдущей лабораторной работы с использованием подпрограммы (рис. 18).



```
~/work/arch-pc/lab09/lab9-4.asm - Mousepad
Файл  Правка  Поиск  Просмотр  Документ  Помощь

%include 'in_out.asm'

SECTION .data
msg_func db "Функция: f(x) = 10x - 4", 0
msg_result db "Результат: ", 0

SECTION .text
GLOBAL _start

_start:
mov eax, msg_func
call sprintLF

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 sprint
mov eax, esi
call iprintLF
call quit

_calculate_fx:
mov ebx, 10
mul ebx
sub eax, 4
ret
```

Рис. 18: Измененная программа предыдущей лабораторной работы

Код программы:

```
%include 'in_out.asm'

SECTION .data
msg_func db "Функция: f(x) = 10x - 4", 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, 10
mul ebx
sub eax, 4

```

2. Запускаю программу в режиме отладчика и пошагово через si просматриваю изменение значений регистров через i r. При выполнении инструкции mul ecx можно заметить, что результат умножения записывается в регистр eax, но также меняет и edx. Значение регистра ebx не обновляется напрямую, поэтому результат программа неверно подсчитывает функцию (рис. 19).

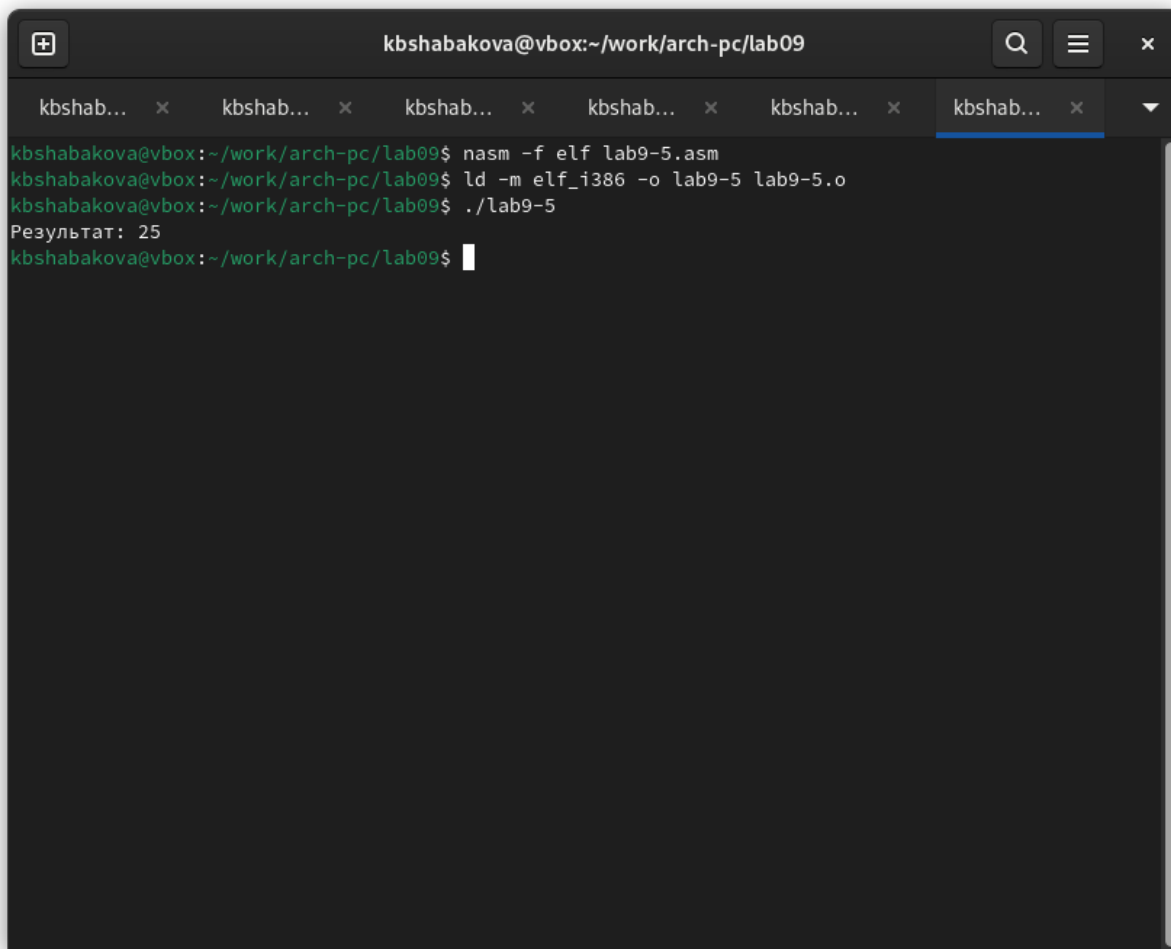
```
--Register group: general--
eax      0x2      2      ecx      0x4      4
edx      0x0      0      ebx      0x5      5
esp      0xffffcf10 0xffffcf10 ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x80490f9 0x80490f9 <_start+17> eflags  0x206    [ PF IF ]
cs       0x23     35     ss       0x2b     43
ds       0x2b     43     es       0x2b     43
fs       0x0      0      gs       0x0      0

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
0x80490fb <_start+19>   add     $0x5,%ebx
0x80490fe <_start+22>   mov     %ebx,%edi
0x8049100 <_start+24>   mov     $0x804a000,%eax
0x8049105 <_start+29>   call    0x804900f <sprint>
0x804910a <_start+34>   mov     %edi,%eax

native process 8526 (asm) In: _start L14 PC: 0x80490f9
eax      0x2      2
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffcf10 0xffffcf10
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490f9 0x80490f9 <_start+17>
eflags   0x206    [ PF IF ]
cs       0x23     35
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 19: Поиск ошибки в программе через пошаговую отладку

Исправляю найденную ошибку, теперь программа верно считает значение функции (рис. 20).



The screenshot shows a terminal window with the title bar 'kbshabakova@vbox:~/work/arch-pc/lab09'. The terminal contains the following text:

```
kbshabakova@vbox:~/work/arch-pc/lab09$ nasm -f elf lab9-5.asm
kbshabakova@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-5 lab9-5.o
kbshabakova@vbox:~/work/arch-pc/lab09$ ./lab9-5
Результат: 25
kbshabakova@vbox:~/work/arch-pc/lab09$
```

Рис. 20: 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 приобрел навыки написания программ с использованием подпрограмм, а так же познакомился с методами отладки при помощи GDB и его основными возможностями.

6 Список литературы

1. Курс на ТУИС
2. Лабораторная работа №9
3. Программирование на языке ассемблера NASM Столяров А. В.