

Отчет по лабораторной работе №9

Понятие подпрограммы. Отладчик GDB

Мальянц Виктория Кареновна

Содержание

1	Цель работы	6
2	Задание	7
3	Выполнение лабораторной работы	8
3.1	Реализация подпрограмм в NASM	8
3.2	Отладка программ с помощью GDB	12
3.3	Добавление точек останова	16
3.4	Работа с данными программы в GDB	18
3.5	Обработка аргументов командной строки в GDB	25
3.6	Выполнение задания для самостоятельной работы	27
4	Выводы	36

Список иллюстраций

3.1	Создание каталога и файла для программы	8
3.2	Копирование файла	8
3.3	Редактирование файла	9
3.4	Запуск исполняемого файла	9
3.5	Редактирование файла	10
3.6	Запуск исполняемого файла	10
3.7	Создание файла	12
3.8	Редактирование файла	12
3.9	Запуск исполняемого файла	13
3.10	Загрузка исполняемого файла в отладчик GDB	13
3.11	Запуск программы в отладчике GDB	13
3.12	Установка брейкпоинта и запуск программы в отладчике GDB . . .	14
3.13	Дисассимилированный код программы	14
3.14	Дисассимилированный код программы	14
3.15	Включение режима псевдографики	15
3.16	Включение режима псевдографики	15
3.17	Включение режима псевдографики	16
3.18	Список точек останова	17
3.19	Установка точки останова	17
3.20	Список точек останова	18
3.21	step 1	19
3.22	step 2	19
3.23	step 3	20
3.24	step 4	20
3.25	step 5	21
3.26	Содержимое регистров	21
3.27	Содержимое регистров	22
3.28	Просмотр значения переменной msg1 по имени и msg2 по адресу	22
3.29	Изменение первого символа переменной msg1 с помощью команды set	23
3.30	Изменение первого символа переменной msg2 с помощью команды set	24
3.31	Вывод значения регистра edx в различных форматах	24
3.32	Изменение значения регистра ebx	25
3.33	Копирование файла	25
3.34	Создание исполняемого файла	25
3.35	Загрузка исполняемого файла	26

3.36 Установка точки останова	27
3.37 Просмотр позиций стека	27
3.38 Создание файла	28
3.39 Редактирование файла	28
3.40 Запуск исполняемого файла	28
3.41 Создание файла	30
3.42 Редактирование файла	31
3.43 Запуск исполняемого файла	31
3.44 Запуск исполняемого файла и его загрузка в отладчик GDB	32
3.45 Установка брейкпоинта и запуск программы в отладчике GDB	32
3.46 Просмотр значений регистров	33
3.47 Просмотр значений регистров	33
3.48 Редактирование файла	34
3.49 Запуск исполняемого файла	34

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. Выполнение задания для самостоятельной работы

3 Выполнение лабораторной работы

3.1 Реализация подпрограмм в NASM

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

```
vkmaljyanc@vbox:~$ mkdir ~/work/arch-pc/lab09  
vkmaljyanc@vbox:~$ cd ~/work/arch-pc/lab09  
vkmaljyanc@vbox:~/work/arch-pc/lab09$ touch lab09-1.asm
```

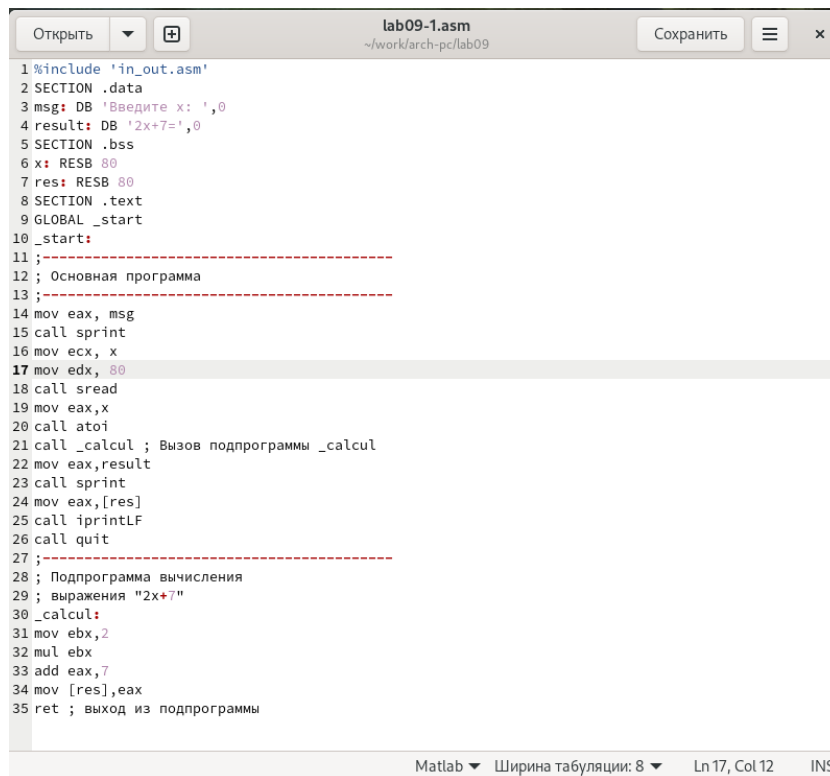
Рис. 3.1: Создание каталога и файла для программы

С помощью команды cp копирую файл in_out.asm (рис. 3.2).

```
vkmaljyanc@vbox:~/work/arch-pc/lab09$ cp ~/Загрузки/in_out.asm in_out.asm
```

Рис. 3.2: Копирование файла

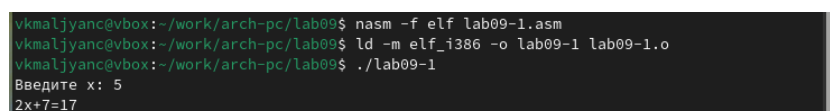
Ввожу в файл lab09-1.asm программу с использованием вызова подпрограммы (рис. 3.3).



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2x+7=',0
5 SECTION .bss
6 x: RESB 80
7 res: RESB 80
8 SECTION .text
9 GLOBAL _start
10 _start:
11 ;-----
12 ; Основная программа
13 ;-----
14 mov eax, msg
15 call sprint
16 mov ecx, x
17 mov edx, 80
18 call sread
19 mov eax, x
20 call atoi
21 call _calcul ; Вызов подпрограммы _calcul
22 mov eax, result
23 call sprint
24 mov eax, [res]
25 call iprintLF
26 call quit
27 ;-----
28 ; Подпрограмма вычисления
29 ; выражения "2x+7"
30 _calcul:
31 mov ebx, 2
32 mul ebx
33 add eax, 7
34 mov [res], eax
35 ret ; выход из подпрограммы
```

Рис. 3.3: Редактирование файла

Создаю исполняемый файл и запускаю его. Проверяю работу исполняемого файла для значения x равного 5. Убеждаюсь в том, что программа работает корректно (рис. 3.4).



```
vkmaljanc@vbox:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
vkmaljanc@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
vkmaljanc@vbox:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2x+7=17
```

Рис. 3.4: Запуск исполняемого файла

Изменяю текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul` (рис. 3.5).

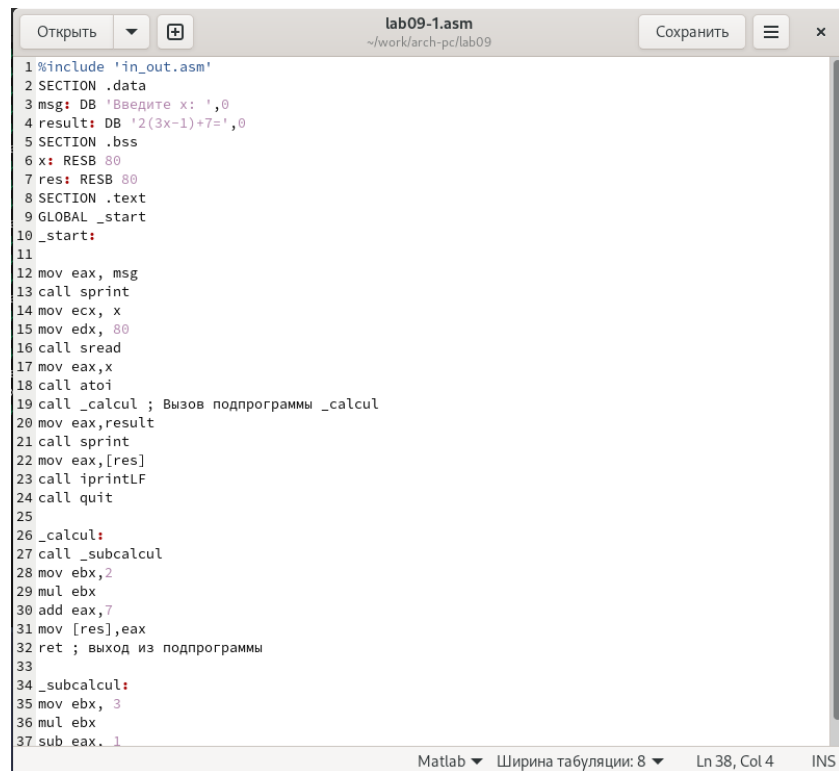


Рис. 3.5: Редактирование файла

Создаю исполняемый файл и запускаю его. Проверяю работу исполняемого файла для значения x равного 5. Убеждаюсь в том, что программа работает корректно (рис. 3.6).

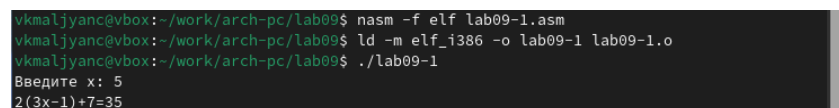


Рис. 3.6: Запуск исполняемого файла

Листинг 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 ; Вызов подпрограммы _calcul
mov eax,result
call sprint
mov eax,[res]
call iprintLF
call quit

_calcul:
call _subcalcul
mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret ; выход из подпрограммы

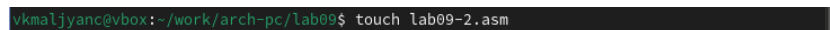
_subcalcul:

```

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

3.2 Отладка программ с помощью GDB

С помощью команды `touch` создаю файл `lab09-2.asm` (рис. 3.7).



```
vkmaljyanc@vbox:~/work/arch-pc/lab09$ touch lab09-2.asm
```

Рис. 3.7: Создание файла

Ввожу в файл `lab09-2.asm` программу вывода сообщения `Hello world!` (рис. 3.8).

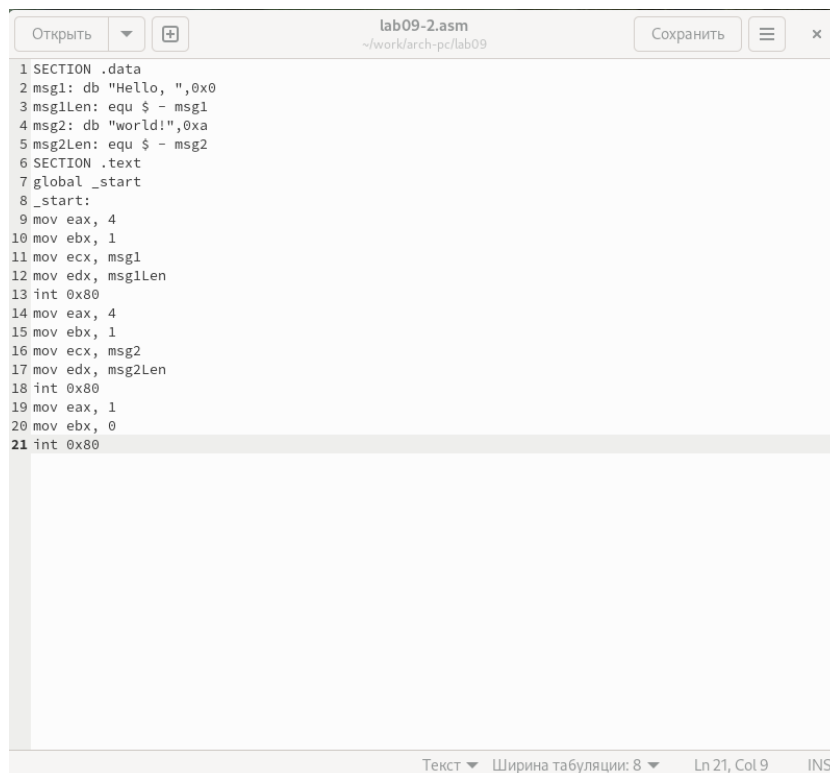


Рис. 3.8: Редактирование файла

В исполняемом файле добавляю отладочную информацию. Провожу трансляцию программы с ключом `'-g'` для работы с GDB (рис. 3.9).

```

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

```

Рис. 3.9: Запуск исполняемого файла

Загружаю исполняемый файл в отладчик GDB (рис. 3.10).

```

vkmaljjanc@vbox:~/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. 3.10: Загрузка исполняемого файла в отладчик GDB

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды `run`. Программа работает корректно (рис. 3.11).

```

vkmaljjanc@vbox:~/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/vkmaljjanc/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.
Hello, world!
[Inferior 1 (process 8829) exited normally]
(gdb)

```

Рис. 3.11: Запуск программы в отладчике GDB

Устанавливаю брейкпоинт на метку `_start` и запускаю программу (рис. 3.12).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/vkmaljanc/work/arch-pc/lab09/lab09-2

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

Рис. 3.12: Установка брейкпоинта и запуск программы в отладчике GDB

Смотрю дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start` (рис. 3.13).

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

Рис. 3.13: Дисассимилированный код программы

Переключаю на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel` (рис. 3.14).

```
(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. 3.14: Дисассимилированный код программы

Различия отображения синтаксиса машинных команд в режимах АТТ и Intel: 1. Различный порядок операндов (в АТТ операнды записываются в порядке источник, назначение, в Intel перанды записываются в порядке назначение, источник)

2. Различия в именах регистров (в АТТ регистры обозначаются с префиксом %, в Intel регистры обозначаются без префикса) 3. Различный размер (в АТТ размер данных указывается с помощью суффиксов, в Intel размер данных определяется контекстом команды)

Включаю режим псевдографики для более удобного анализа программы (рис. 3.15) (рис. 3.16) (рис. 3.17).

```
(gdb) layout asm
```

Рис. 3.15: Включение режима псевдографики

```
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
0x804902c <_start+44> mov    eax,0x1
0x8049031 <_start+49> mov    ebx,0x0
0x8049036 <_start+54> int    0x80
0x8049038 add    BYTE PTR [eax],al
0x804903a add    BYTE PTR [eax],al
0x804903c add    BYTE PTR [eax],al
0x804903e add    BYTE PTR [eax],al
0x8049040 add    BYTE PTR [eax],al

native process 8845 In: _start L9 PC: 0x8049000
(gdb) layout regs
```

Рис. 3.16: Включение режима псевдографики

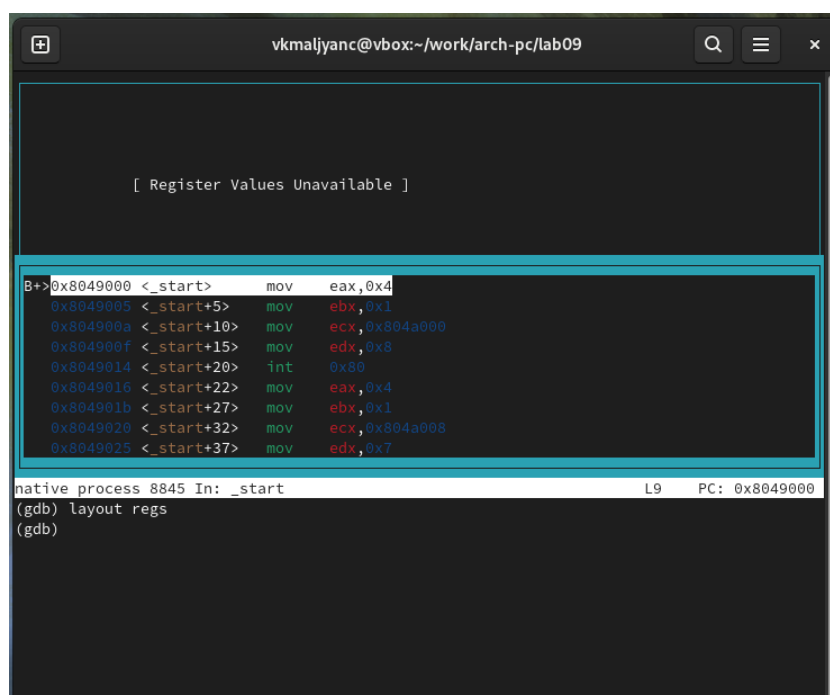


Рис. 3.17: Включение режима псевдографики

3.3 Добавление точек останова

Проверяю наличие точки останова по имени метки (_start) с помощью команды `info breakpoints`. Точка останова существует (рис. 3.18).

The screenshot shows a GDB terminal window with the title bar "vkmaljan@vbox:~/work/arch-pc/lab09". The main display area shows assembly code starting at address 0x8049000. The code includes instructions like `mov eax,0x4`, `mov ebx,0x1`, `mov ecx,0x804a000`, `mov edx,0x8`, `int 0x80`, and several `mov` instructions with offsets. A blue box highlights the assembly code. Below the assembly code, the GDB prompt shows the following commands and output:

```
native process 8845 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) info breakpoints
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
(gdb)
```

Рис. 3.18: Список точек останова

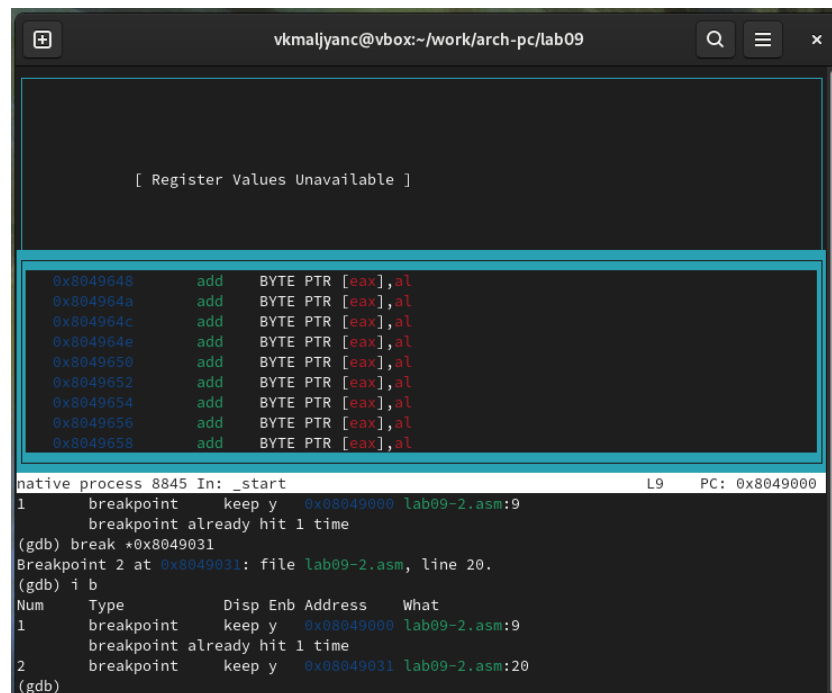
Определяю адрес предпоследней инструкции (`mov ebx,0x0`) и устанавливаю точку останова (рис. 3.19).

The screenshot shows a GDB terminal window with the title bar "vkmaljan@vbox:~/work/arch-pc/lab09". The main display area shows assembly code starting at address 0x8049648. The code includes several `add` instructions: `add BYTE PTR [eax],al`. A blue box highlights the assembly code. Below the assembly code, the GDB prompt shows the following commands and output:

```
native process 8845 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) info breakpoints
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb)
```

Рис. 3.19: Установка точки останова

Просматриваю информацию о всех установленных точках останова (рис. 3.20).



The screenshot shows a GDB terminal window with the title bar 'vkmalijanc@vbox:~/work/arch-pc/lab09'. The main display area is divided into two sections. The top section, titled '[Register Values Unavailable]', is currently empty. The bottom section displays a list of assembly instructions with their addresses and disassembled code:

```
0x8049648  add  BYTE PTR [eax],al
0x804964a  add  BYTE PTR [eax],al
0x804964c  add  BYTE PTR [eax],al
0x804964e  add  BYTE PTR [eax],al
0x8049650  add  BYTE PTR [eax],al
0x8049652  add  BYTE PTR [eax],al
0x8049654  add  BYTE PTR [eax],al
0x8049656  add  BYTE PTR [eax],al
0x8049658  add  BYTE PTR [eax],al
```

Below the assembly list, the GDB status bar shows 'native process 8845 In: _start' and 'L9 PC: 0x8049000'. The command history and output are visible at the bottom:

```
1 breakpoint keep y 0x8049000 lab09-2.asm:9
breakpoint already hit 1 time
(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 0x8049000 lab09-2.asm:9
      breakpoint already hit 1 time
2      breakpoint      keep y 0x8049031 lab09-2.asm:20
(gdb)
```

Рис. 3.20: Список точек останова

3.4 Работа с данными программы в GDB

Выполняю 5 инструкций с помощью команды `stepi`. Изменяются значения регистров: `eax`, `ebx`, `ecx`, `edx` (рис. 3.21) (рис. 3.22) (рис. 3.23) (рис. 3.24) (рис. 3.25).

```

vkmaljanc@vbox:~/work/arch-pc/lab09

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd050 0xffffd050
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

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

native process 8845 In: _start L10 PC: 0x8049005
breakpoint already hit 1 time
(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) stepi
(gdb) info registers

```

Рис. 3.21: step1

```

vkmaljanc@vbox:~/work/arch-pc/lab09

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x1      1
esp      0xffffd050 0xffffd050
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

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

native process 8845 In: _start L11 PC: 0x804900a
--Type <RET> for more, q to quit, c to continue without paging--c
eflags   0x202      [ IF ]
cs       0x23      35
ss       0x2b      43
ds       0x2b      43
es       0x2b      43
fs       0x0      0
gs       0x0      0
(gdb) stepi
(gdb)

```

Рис. 3.22: step2

```

vkmaljanc@vbox:~/work/arch-pc/lab09

Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x0      0
ebx      0x1      1
esp      0xffffd050 0xffffd050
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

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

native process 8845 In: _start L12 PC: 0x804900f
eflags    0x202      [ IF ]
cs        0x23      35
ss        0x2b      43
ds        0x2b      43
es        0x2b      43
fs        0x0      0
gs        0x0      0
(gdb) stepi
(gdb) stepi
(gdb)

```

Рис. 3.23: stepi 3

```

vkmaljanc@vbox:~/work/arch-pc/lab09

Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd050 0xffffd050
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

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

native process 8845 In: _start L13 PC: 0x8049014
cs        0x23      35
ss        0x2b      43
ds        0x2b      43
es        0x2b      43
fs        0x0      0
gs        0x0      0
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb)

```

Рис. 3.24: stepi 4

```

vkmaljan@vbox:~/work/arch-pc/lab09

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd050 0xffffd050
ebp      0x0      0
esi      0x0      0
edi      0x0      0

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

native process 8845 In: _start L14 PC: 0x8049016
ss      0x2b      43
ds      0x2b      43
es      0x2b      43
fs      0x0      0
gs      0x0      0
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb)

```

Рис. 3.25: stepi 5

Просматриваю содержимое регистров с помощью команды info registers (рис. 3.26) (рис. 3.27).

```

vkmaljan@vbox:~/work/arch-pc/lab09

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd050 0xffffd050
ebp      0x0      0
esi      0x0      0
edi      0x0      0

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

native process 8845 In: _start L14 PC: 0x8049016
eip      0x8049016 0x8049016 <_start+22>
--Type <RET> for more, q to quit, c to continue without paging--c
eflags   0x202    [ IF ]
cs       0x23      35
ss       0x2b      43
ds       0x2b      43
es       0x2b      43
fs       0x0      0
gs       0x0      0
(gdb) info registers

```

Рис. 3.26: Содержимое регистров

```

vkmaljan@vbox:~/work/arch-pc/lab09

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd050 0xffffd050
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

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

native process 8845 In: _start L14 PC: 0x8049016
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd050 0xffffd050
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 3.27: Содержимое регистров

Просматриваю значение переменной msg1 по имени и переменной msg2 по адресу (рис. 3.28).

```

vkmaljan@vbox:~/work/arch-pc/lab09

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd050 0xffffd050
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

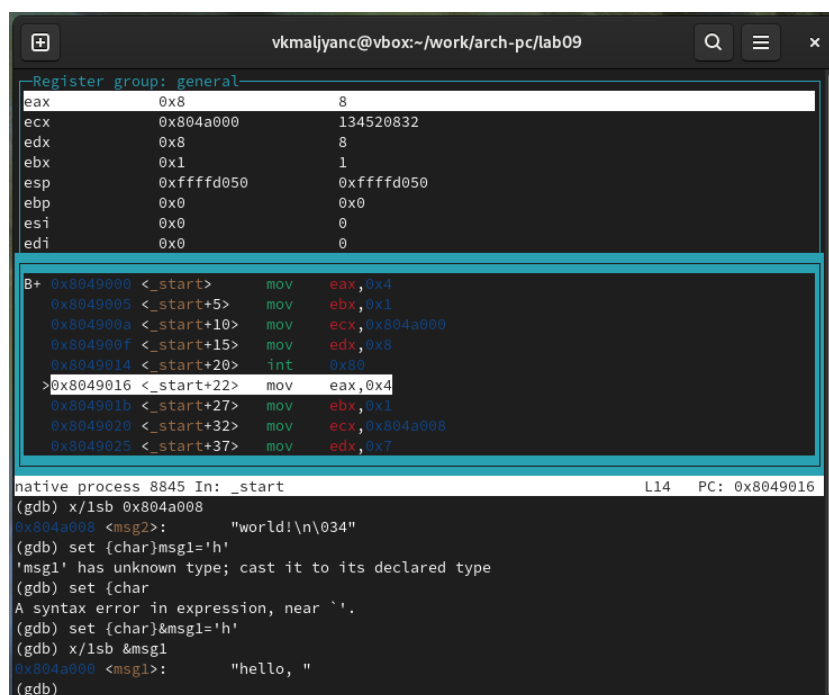
B+ 0x8049000 <_start>    mov     eax,0x4
0x8049005 <_start+5>    mov     ebx,0x1
0x804900a <_start+10>   mov     ecx,0x804a000
0x804900f <_start+15>   mov     edx,0x8
0x8049014 <_start+20>   int     0x80
>0x8049016 <_start+22>   mov     eax,0x4
0x804901b <_start+27>   mov     ebx,0x1
0x8049020 <_start+32>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

native process 8845 In: _start L14 PC: 0x8049016
ss       0x2b      43
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. 3.28: Просмотр значения переменной msg1 по имени и msg2 по адресу

Изменяю первый символ переменной msg1 с помощью команды set (рис. 3.29).



```
vkmaljanc@vbox:~/work/arch-pc/lab09
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd050 0xffffd050
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

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

native process 8845 In: _start L14 PC: 0x8049016
(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}
A syntax error in expression, near `'.
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a008 <msg1>: "hello, "
(gdb)
```

Рис. 3.29: Изменение первого символа переменной msg1 с помощью команды set

Изменяю первый символ переменной msg2 с помощью команды set (рис. 3.30).

```

+ vkmaljanc@vbox:~/work/arch-pc/lab09
-Register group: general-
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd050 0xffffd050
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

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

native process 8845 In: _start L14 PC: 0x8049016
'msg1' has unknown type; cast it to its declared type
(gdb) set {char}
A syntax error in expression, near `'.
(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. 3.30: Изменение первого символа переменной msg2 с помощью команды set

Вывожу в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx (рис. 3.31).

```

+ vkmaljanc@vbox:~/work/arch-pc/lab09
-Register group: general-
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd050 0xffffd050
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

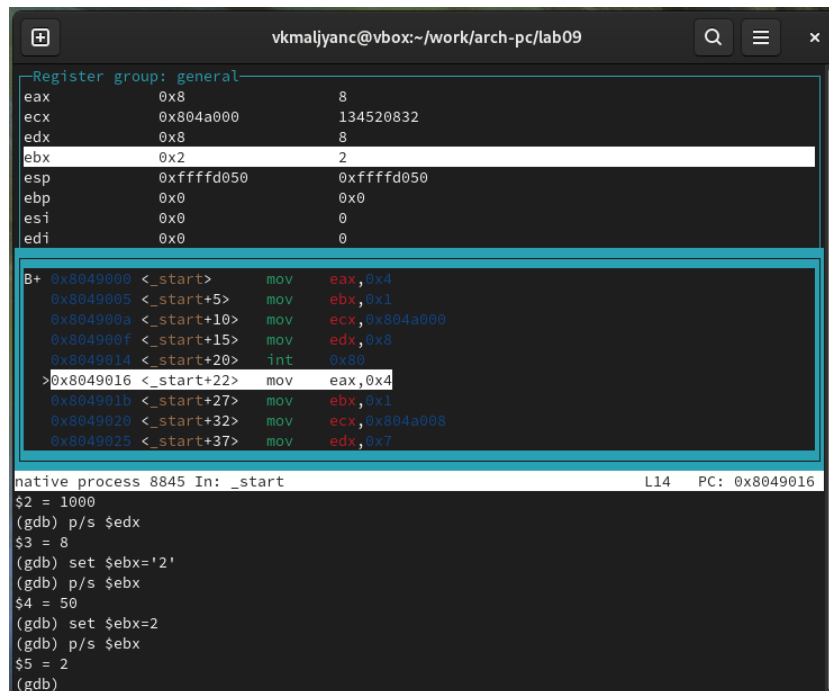
B+ 0x8049000 <_start>    mov     eax,0x4
0x8049005 <_start+5>    mov     ebx,0x1
0x804900a <_start+10>   mov     ecx,0x804a000
0x804900f <_start+15>   mov     edx,0x8
0x8049014 <_start+20>   int     0x80
>0x8049016 <_start+22>   mov     eax,0x4
0x804901b <_start+27>   mov     ebx,0x1
0x8049020 <_start+32>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

native process 8845 In: _start L14 PC: 0x8049016
(gdb) set {char}&msg2='t'
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "torld!\n\034"
(gdb) p/x $edx
$1 = 0x8
(gdb) p/t $edx
$2 = 1000
(gdb) p/s $edx
$3 = 8
(gdb)

```

Рис. 3.31: Вывод значения регистра edx в различных форматах

С помощью команды set изменяю значение регистра ebx (рис. 3.32).



```
vkmaljanc@vbox:~/work/arch-pc/lab09

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x2      2
esp      0xffffd050 0xffffd050
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

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

native process 8845 In: _start      L14  PC: 0x8049016
$2 = 1000
(gdb) p/s $edx
$3 = 8
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
(gdb)
```

Рис. 3.32: Изменение значения регистра ebx

Разница вывода команд p/s \$ebx: при введении команды p/s \$ebx='2' вывод 50, так как в программе '2' рассматривается как символ, в ASCII равен 50, при введении команды p/s \$ebx=2 вывод 2, так как введено число.

3.5 Обработка аргументов командной строки в GDB

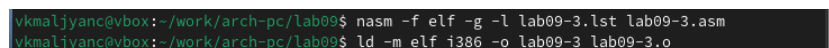
Копирую файл lab8-2.asm в файл с именем lab09-3.asm рис. 3.33).



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

Рис. 3.33: Копирование файла

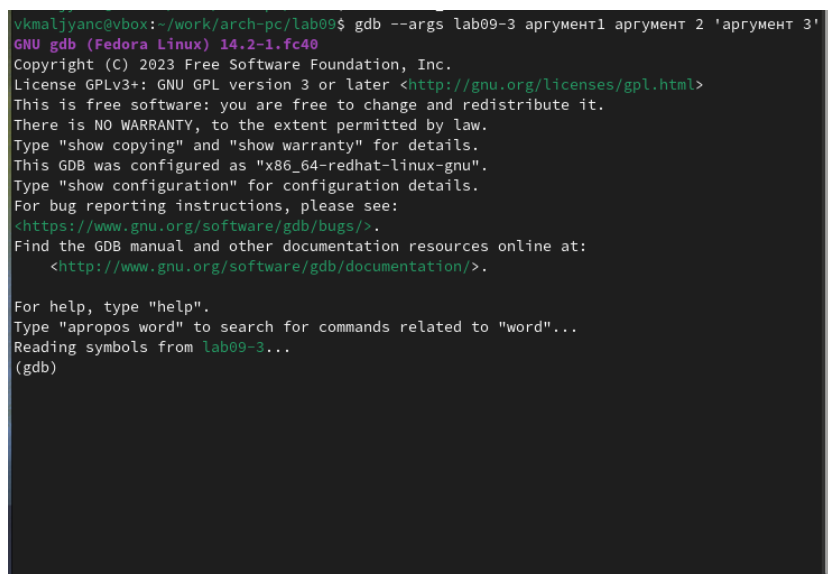
Создаю исполняемый файл 3.34).



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

Рис. 3.34: Создание исполняемого файла

Загружаю исполняемый файл с использованием ключа `--args` в отладчик, указав аргументы 3.35).



```
vkmaljyanc@vbox:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
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 lab09-3...
(gdb)
```

Рис. 3.35: Загрузка исполняемого файла

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее 3.36).

```
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 lab09-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 8.
(gdb) run
Starting program: /home/vkmaljyanc/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумен
т\ 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)
```

Рис. 3.36: Установка точки останова

Просматриваю позиции стека 3.37).

```
(gdb) x/x $esp
0xffffd010: 0x00000005
(gdb) x/s *(void**)($esp + 4)
0xffffd1d2: "/home/vkmaljyanc/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd1fe: "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffd210: "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffd221: "2"
(gdb) x/s *(void**)($esp + 20)
0xffffd223: "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 3.37: Просмотр позиций стека

Шаг изменения адреса равен 4, потому что каждый элемент стека занимает 4 байта.

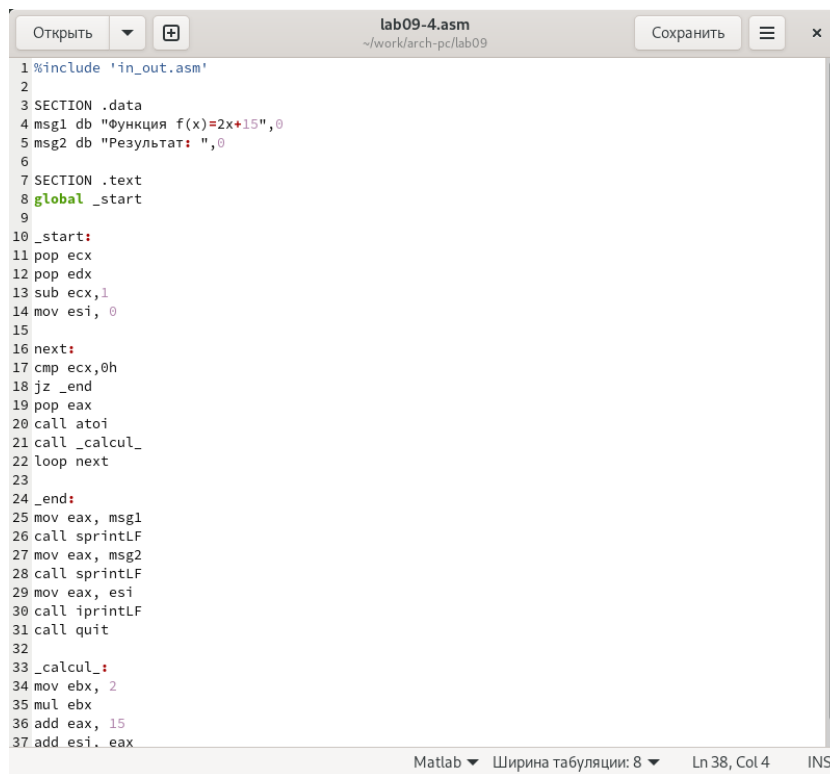
3.6 Выполнение задания для самостоятельной работы

1. С помощью команды touch создаю файл lab09-4.asm (рис. 3.38).

```
vkmaljyanc@vbox:~/work/arch-pc/lab09$ touch lab09-4.asm
```

Рис. 3.38: Создание файла

Преобразовываю программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму (рис. 3.39).



```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg1 db "Функция f(x)=2x+15",0
5 msg2 db "Результат: ",0
6
7 SECTION .text
8 global _start
9
10 _start:
11 pop ecx
12 pop edx
13 sub ecx,1
14 mov esi, 0
15
16 next:
17 cmp ecx,0h
18 jz _end
19 pop eax
20 call atoi
21 call _calcul_
22 loop next
23
24 _end:
25 mov eax, msg1
26 call sprintf
27 mov eax, msg2
28 call sprintf
29 mov eax, esi
30 call sprintf
31 call quit
32
33 _calcul_:
34 mov ebx, 2
35 mul ebx
36 add eax, 15
37 add esi, eax
```

Рис. 3.39: Редактирование файла

Создаю исполняемый файл и запускаю его. Проверяю работу исполняемого файла, указав аргументы: 1 2 3 4 5 (рис. 3.40). Убеждаюсь в том, что программа работает корректно.

```
vkmaljyanc@vbox:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
vkmaljyanc@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
vkmaljyanc@vbox:~/work/arch-pc/lab09$ ./lab09-4 1 2 3 4 5
Функция f(x)=2x+15
Результат:
105
```

Рис. 3.40: Запуск исполняемого файла

Листинг программы:

```

#include 'in_out.asm'

SECTION .data
msg1 db "Функция  $f(x)=2x+15$ ",0
msg2 db "Результат: ",0

SECTION .text
global _start

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

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

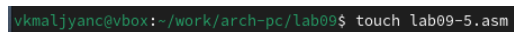
_end:
mov eax, msg1
call sprintf
mov eax, msg2
call sprintf
mov eax, esi

```

```
call iprintLF  
call quit
```

```
_calcul_:  
mov ebx, 2  
mul ebx  
add eax, 15  
add esi, eax  
ret
```

2. С помощью команды `touch` создаю файл `lab09-5.asm` (рис. 3.41).



```
vkmaljyanc@vbox:~/work/arch-pc/lab09$ touch lab09-5.asm
```

Рис. 3.41: Создание файла

Ввожу в файл `lab09-5.asm` программу вычисления выражения $(3+2)*4+5$ (рис. 3.42).

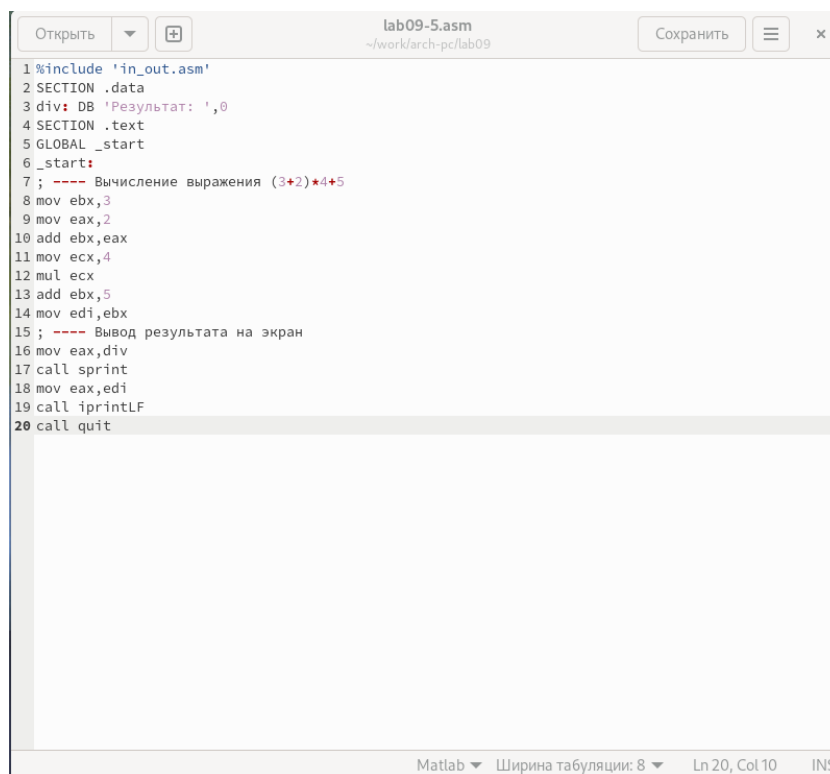


Рис. 3.42: Редактирование файла

Создаю исполняемый файл и запускаю его. Проверяю работу исполняемого файла и убеждаюсь в том, что программа работает некорректно (рис. 3.43).

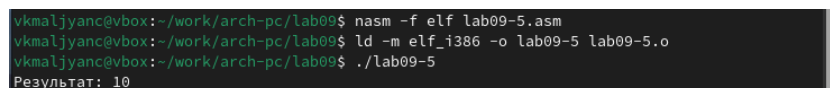


Рис. 3.43: Запуск исполняемого файла

В исполняемом файле добавляю отладочную информацию. Провожу трансляцию программы с ключом '-g' для работы с GDB. Загружаю исполняемый файл в отладчик GDB (рис. 3.44).

```
vkmaljyanc@vbox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-5.lst lab09-5.asm
vkmaljyanc@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab09-5.o
vkmaljyanc@vbox:~/work/arch-pc/lab09$ gdb lab09-5
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 lab09-5...
(gdb)
```

Рис. 3.44: Запуск исполняемого файла и его загрузка в отладчик GDB

Устанавливаю брейкпоинт на метку `_start` и запускаю программу (рис. 3.45).

```
(gdb) break _start
Breakpoint 1 at 0x80490e8: file lab09-5.asm, line 8.
(gdb) run
Starting program: /home/vkmaljyanc/work/arch-pc/lab09/lab09-5

Breakpoint 1, _start () at lab09-5.asm:8
8      mov ebx,3
(gdb)
```

Рис. 3.45: Установка брейкпоинта и запуск программы в отладчике GDB

Включаю режим псевдографики для более удобного анализа программы. С помощью команды `stepi` просматриваю значения регистров. В программе сумму `eax` и `ebx` записывали в `ebx`, но так как `mul ecx` умножает `eax` на `ecx` и записывает результат в `eax`, то на `ecx` умножилась не сумма `eax` и `ebx`, а `eax`, поэтому при выполнении `mul ecx` значение регистра `ebx` не изменилось (рис. 3.46) (рис. 3.47).

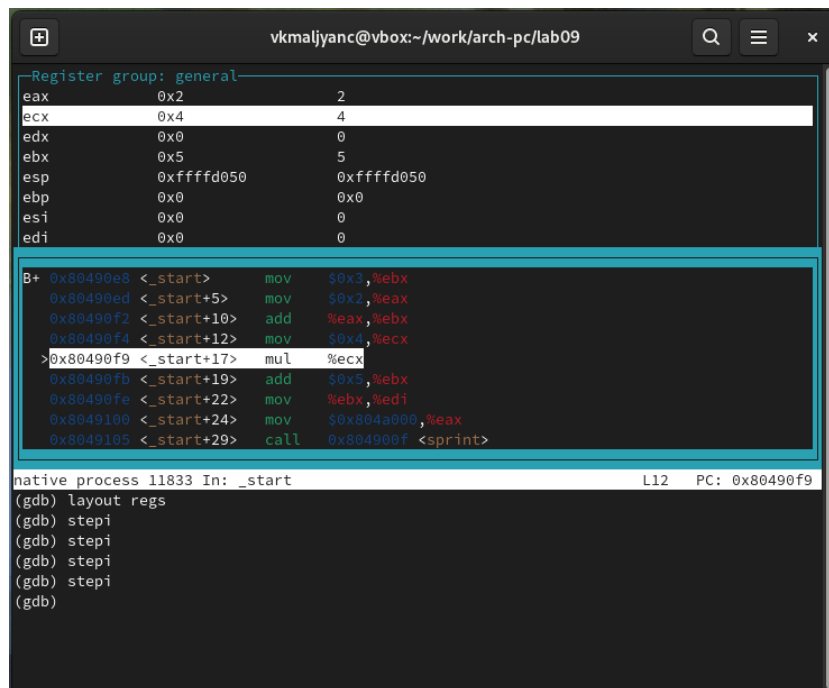


Рис. 3.46: Просмотр значений регистров

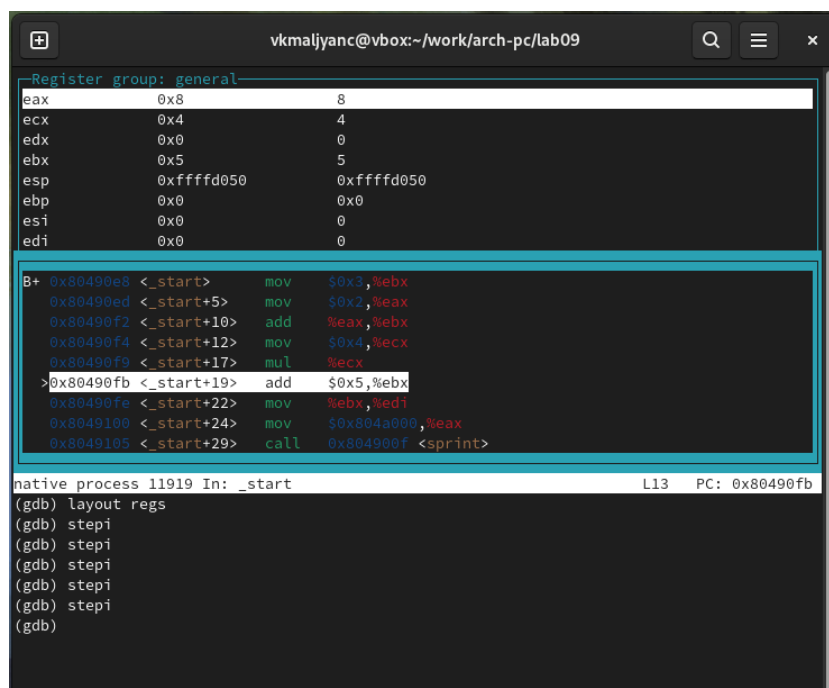


Рис. 3.47: Просмотр значений регистров

Изменяю текст программы, чтобы она работала корректно (рис. 3.48).

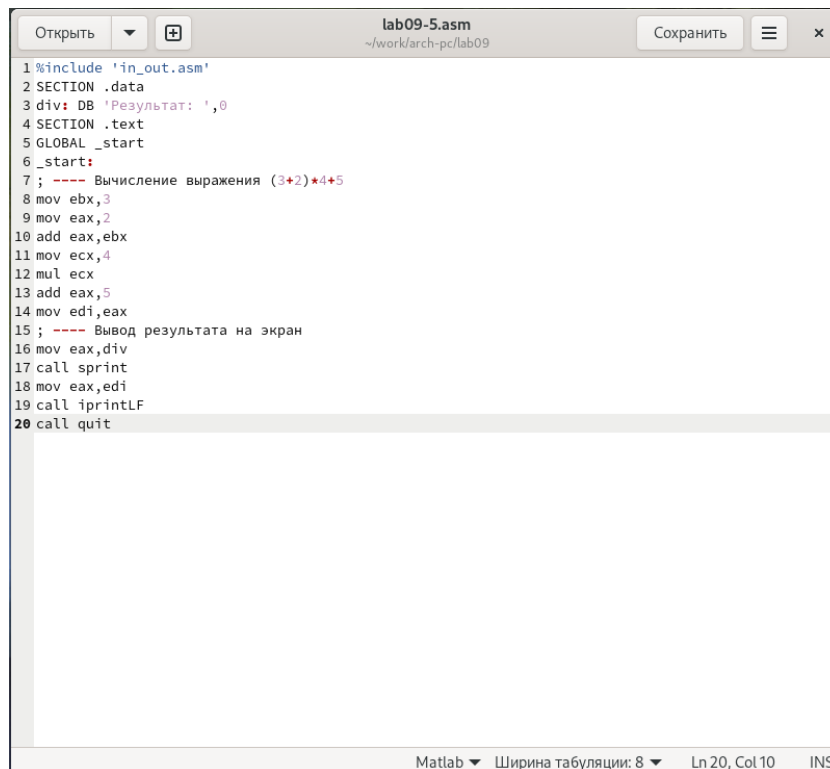


Рис. 3.48: Редактирование файла

Создаю исполняемый файл и запускаю его. Проверяю работу исполняемого файла и убеждаюсь в том, что программа работает корректно (рис. 3.49).

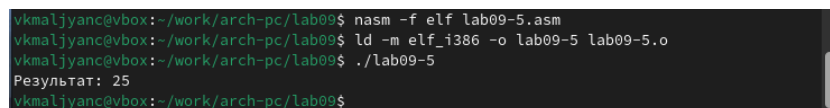


Рис. 3.49: Запуск исполняемого файла

Листинг 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 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
```

4 Выводы

Я приобрела навыки написания программ с использованием подпрограмм. Познакомилась с методами отладки при помощи GDB и его основными возможностями.