

# Шаблон отчёта по лабораторной работе

9

Баптишта Матеуж

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
3.1	Реализация подпрограмм в NASM. . . . .	7
3.2	Отладка программ с помощью GDB. ....	10
3.3	Добавление точек останова. ....	13
3.4	Работа с данными программы в GDB. ....	14
3.5	Обработка аргументов командной строки в GDB. ....	17
3.6	Задания для самостоятельной работы. ....	19
<b>4</b>	<b>Выводы</b>	<b>20</b>
	<b>Список литературы</b>	<b>21</b>

## Список иллюстраций

3.1	создание файлов для лабораторной работы . . . . .	7
3.2	ввод текста программы . . . . .	8
3.3	запуск исполняемого файла . . . . .	8
3.4	изменение текста программы . . . . .	9
3.5	запуск исполняемого файла .....	10
3.6	ввод текста программы .....	10
3.7	получение исполняемого файла .....	11
3.8	загрузка исполняемого файла в отладчике .....	11
3.9	проверка работы файла с помощью команды run .....	11
3.10	установка брейкпоинта и запуск программы .....	11
3.11	использование команд disassemble и set disassembly-flavor intel .	12
3.12	включение режима псевдографики .....	13
3.13	установление точек останова .....	13
3.14	до использования команды stepi .....	14
3.15	после использования команды stepi .....	14
3.16	просмотр значений переменных .....	15
3.17	использование команды set .....	15
3.18	вывод значения регистра .....	16
3.19	использование команды set для изменения значения регистра .....	16
3.20	завершение работы .....	17
3.21	создание файла .....	17
3.22	загрузка исполняемого файла в отладчике .....	18
3.23	установление точек останова .....	18
3.24	просмотр значений и введение в стек .....	18
3.25	запуск программы .....	19
3.26	запуск программы .....	19

## Список таблиц

# 1 Цель работы

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

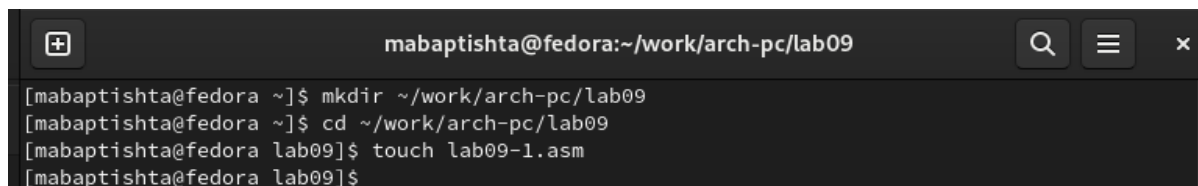
## 2 Задание

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

## 3 Выполнение лабораторной работы

### 3.1 Реализация подпрограмм в NASM.

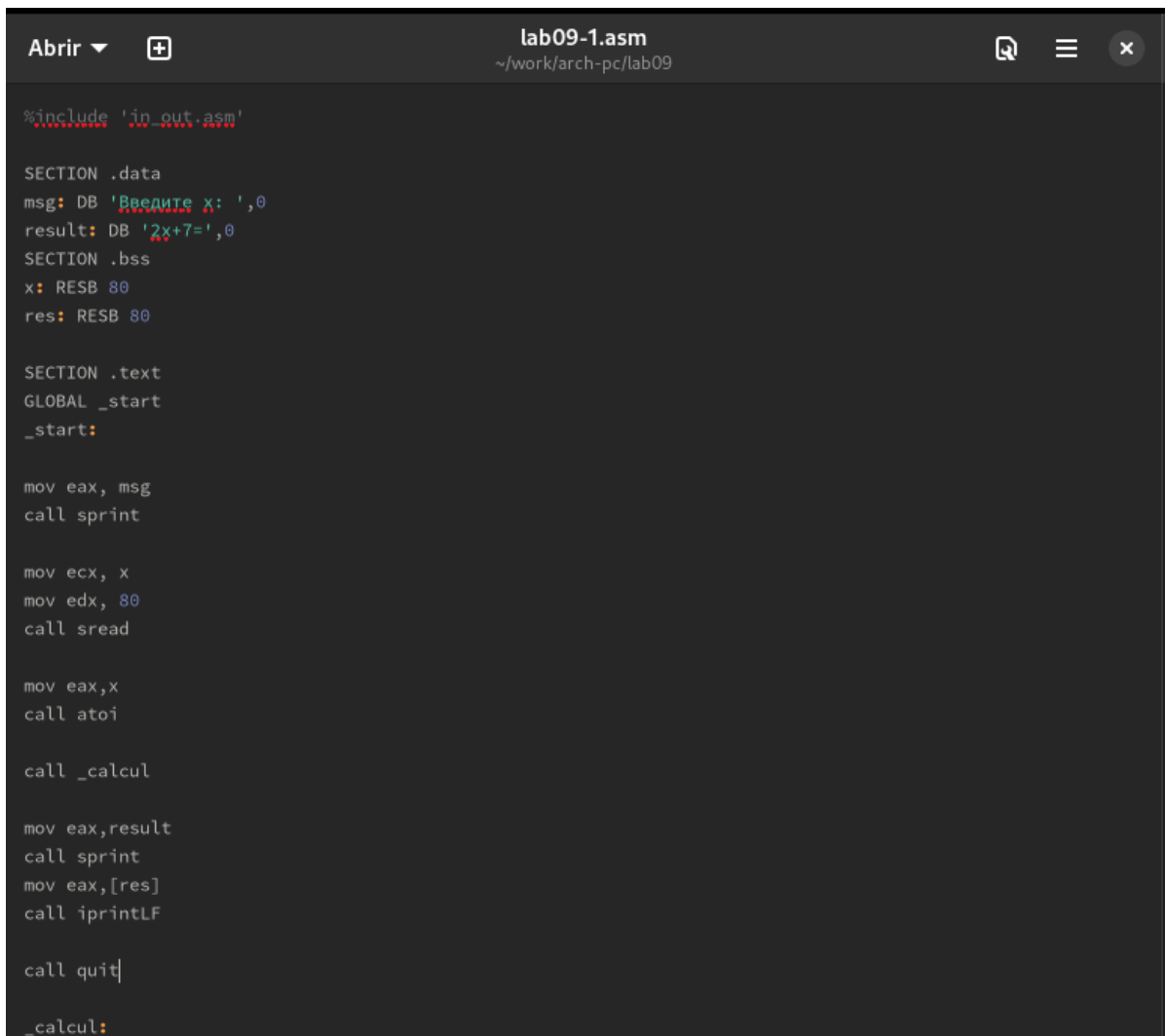
- Создаю каталог для выполнения лабораторной работы № 9, перехожу в него и создаю файл lab09-1.asm. (рис. [3.1]).



```
mabaptishta@fedora:~/work/arch-pc/lab09
[mabaptishta@fedora ~]$ mkdir ~/work/arch-pc/lab09
[mabaptishta@fedora ~]$ cd ~/work/arch-pc/lab09
[mabaptishta@fedora lab09]$ touch lab09-1.asm
[mabaptishta@fedora lab09]$
```

Рис. 3.1: создание файлов для лабораторной работы

- Ввожу в файл lab09-1.asm текст программы с использованием подпрограммы из листинга 9.1. (рис. [3.2]).



```
lab09-1.asm
~/work/arch-pc/lab09

#include 'in_out.asm'

SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
res: RESB 80

SECTION .text
GLOBAL _start
_start:

mov eax, msg
call sprint

mov ecx, x
mov edx, 80
call sread

mov eax, x
call atoi

call _calcul

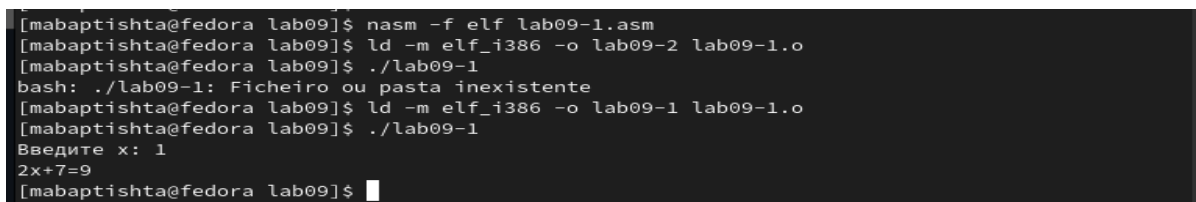
mov eax, result
call sprint
mov eax, [res]
call iprintLF

call quit

_calcul:
```

Рис. 3.2: ввод текста программы

- Создаю исполняемый файл и проверяю его работу. (рис. [3.3]).



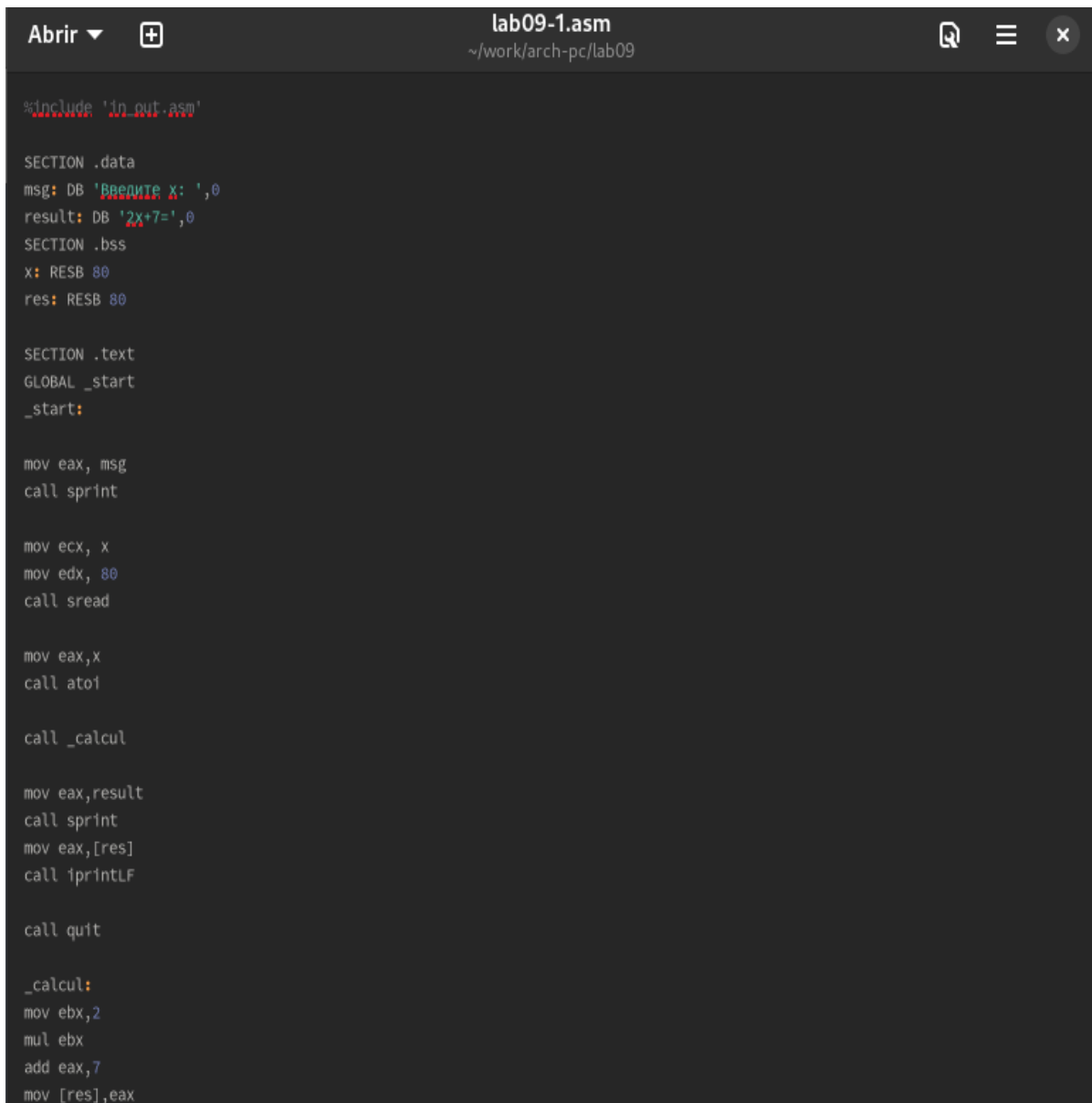
```
[mabaptishta@fedora lab09]$ nasm -f elf lab09-1.asm
[mabaptishta@fedora lab09]$ ld -m elf_i386 -o lab09-2 lab09-1.o
[mabaptishta@fedora lab09]$ ./lab09-1
bash: ./lab09-1: Fichero ou pasta inexistente
[mabaptishta@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[mabaptishta@fedora lab09]$ ./lab09-1
Введите x: 1
2x+7=9
[mabaptishta@fedora lab09]$
```

Рис. 3.3: запуск исполняемого файла

- Изменяю текст программы, добавив подпрограмму \_subcalcul в подпрограм-



му `_calcul` для вычисления выражения  $f(g(x))$ , где  $x$  вводится с клавиатуры,  $f(x) = 2x + 7$ ,  $g(x) = 3x - 1$ . (рис. [3.4]).



```
lab09-1.asm
~/work/arch-pc/lab09

#include 'in_out.asm'

SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
res: RESB 80

SECTION .text
GLOBAL _start
_start:

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 iprintfLF

call quit

_calcul:
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
```

Рис. 3.4: изменение текста программы

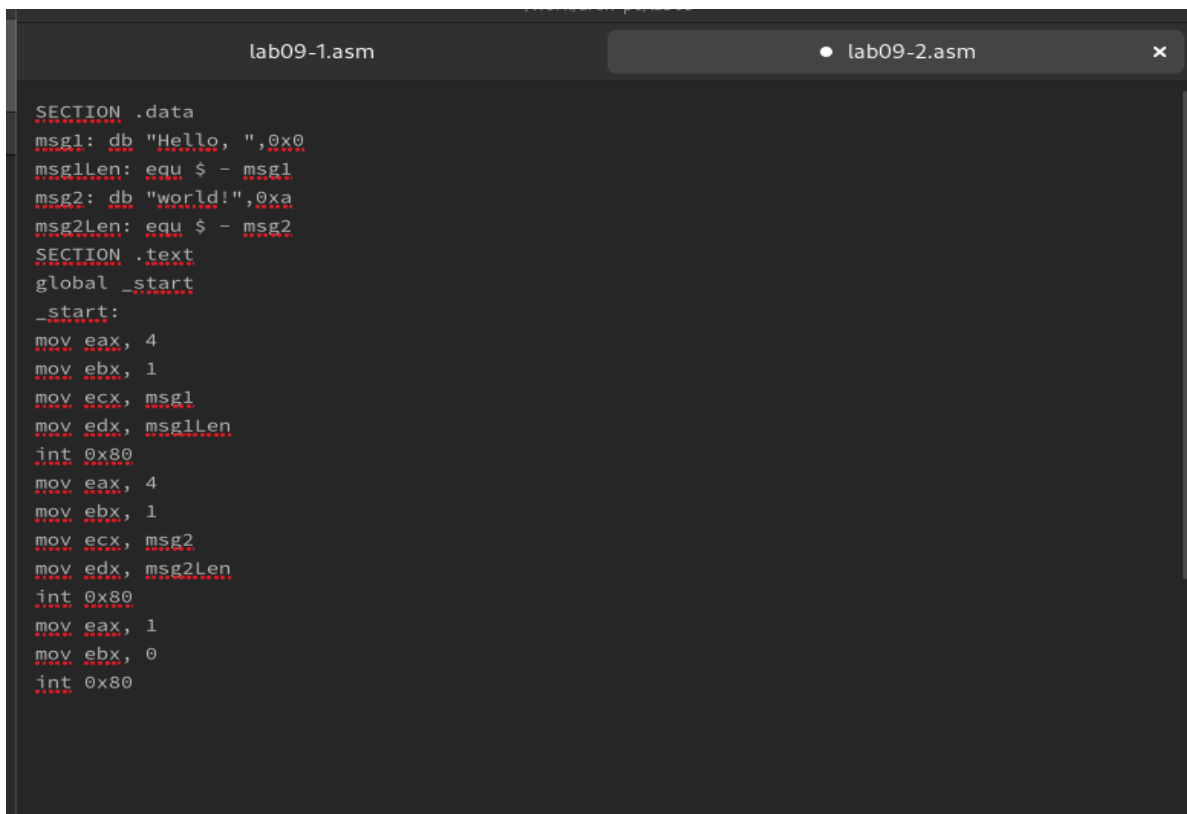
- Создаю исполняемый файл и проверяю его работу. (рис. [3.5]).

```
[mabaptishta@fedora lab09]$ nasm -f elf lab09-1.asm
[mabaptishta@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[mabaptishta@fedora lab09]$ ./lab09-1
Введите x: 1
2x+7=9
[mabaptishta@fedora lab09]$
```

Рис. 3.5: запуск исполняемого файла

## 3.2 Отладка программ с помощью GDB.

- На этом шаге мы создали файл lab09-2.asm с текстом программы из листинга 9.2. (рис. [3.6]).



```
lab09-1.asm  lab09-2.asm x
SECTION .data
msg1: db "Hello, ",0x0
msg1len: equ $ - msg1
msg2: db "world!",0xa
msg2len: equ $ - msg2
SECTION .text
global _start
_start:
mov eax, 4
mov ebx, 1
mov ecx, msg1
mov edx, msg1len
int 0x80
mov eax, 4
mov ebx, 1
mov ecx, msg2
mov edx, msg2len
int 0x80
mov eax, 1
mov ebx, 0
int 0x80
```

Рис. 3.6: ввод текста программы

- Получаю исполняемый файл для работы с GDB с ключом '-g'. (рис. [3.7]).

```
[mabaptishta@fedora lab09]$ touch lab09-2.asm
[mabaptishta@fedora lab09]$ nasm -f elf lab09-2.asm
[mabaptishta@fedora lab09]$ ld -m elf_i386 -o lab09-2 lab09-2.o
[mabaptishta@fedora lab09]$ ./lab09-2
Hello, world!
[mabaptishta@fedora lab09]$
```

Рис. 3.7: получение исполняемого файла

- Загружаю исполняемый файл в отладчик gdb.(рис. [3.8]).

```
[mabaptishta@fedora lab09]$ gdb lab09-2
GNU gdb (GDB) Fedora Linux 13.1-2.fc38
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...
(No debugging symbols found in lab09-2)
(gdb)
```

Рис. 3.8: загрузка исполняемого файла в отладчике

- Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run. (рис. [3.9]).

```
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/s/e/serazanacua/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 4245) exited normally]
(gdb)
```

Рис. 3.9: проверка работы файла с помощью команды run

- Для более подробного анализа программы устанавливаю брейкпоинт на метку \_start и запускаю её.(рис. [3.10]).

```
(gdb) break _start
Breakpoint 1 at 0x8049000
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/s/e/serazanacua/work/arch-pc/lab09/lab09-2
Breakpoint 1, 0x08049000 in _start ()
```

Рис. 3.10: установка брейкпоинта и запуск программы

- Просматриваю дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start`, и переключаюсь на отображение команд с синтаксисом Intel, введя команду `set disassembly-flavor intel`. (рис. [3.11]).

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

Рис. 3.11: использование команд `disassemble` и `set disassembly-flavor intel`

- Разница в синтаксисе между AT&T и INTEL заключается в том, что AT&T использует синтаксис `mov $0x4,%eax`, который популярен среди пользователей Linux, с другой стороны, INTEL использует синтаксис `mov eax,0x4`, который является популярен среди пользователей Windows.
- Включаю режим псевдографики для более удобного анализа программы с помощью команд `layout asm` и `layout regs`. (рис. [3.12]).

```

[ Register Values Unavailable ]

B> 0x0049000 <_start> mov     eax,0x4
0x0049005 <_start+5> mov     ebx,0x1
0x004900a <_start+10> mov     ecx,0x304a000
0x004900f <_start+15> mov     edx,0x3
0x0049014 <_start+20> int     0x0
0x0049016 <_start+22> mov     eax,0x4
0x004901b <_start+27> mov     ebx,0x1
0x0049020 <_start+32> mov     ecx,0x304a008
0x0049025 <_start+37> mov     edx,0x7
0x004902a <_start+42> int     0x0
0x004902c <_start+44> mov     eax,0x1
0x0049031 <_start+49> mov     ebx,0x0
0x0049035 <_start+54> int     0x0

native process 4281 In: _start
(gdb) layout regs
(gdb)

```

Рис. 3.12: включение режима псевдографики

### 3.3 Добавление точек останова.

- Проверяю, что точка останова по имени метки `_start` установлена с помощью команды `info breakpoints` и устанавливаю еще одну точку останова по адресу инструкции `mov ebx,0x0`. Просматриваю информацию о всех установленных точках останова.(рис. [3.13]).

```

[ Register Values Unavailable ]

0x0049004 add     BYTE PTR [eax],al
0x0049006 add     BYTE PTR [eax],al
0x0049008 add     BYTE PTR [eax],al
0x004900a add     BYTE PTR [eax],al
0x004900c add     BYTE PTR [eax],al
0x004900e add     BYTE PTR [eax],al
0x0049008 add     BYTE PTR [eax],al
0x0049002 add     BYTE PTR [eax],al
0x0049004 add     BYTE PTR [eax],al
0x0049006 add     BYTE PTR [eax],al
0x0049008 add     BYTE PTR [eax],al
0x004900a add     BYTE PTR [eax],al
0x004900c add     BYTE PTR [eax],al

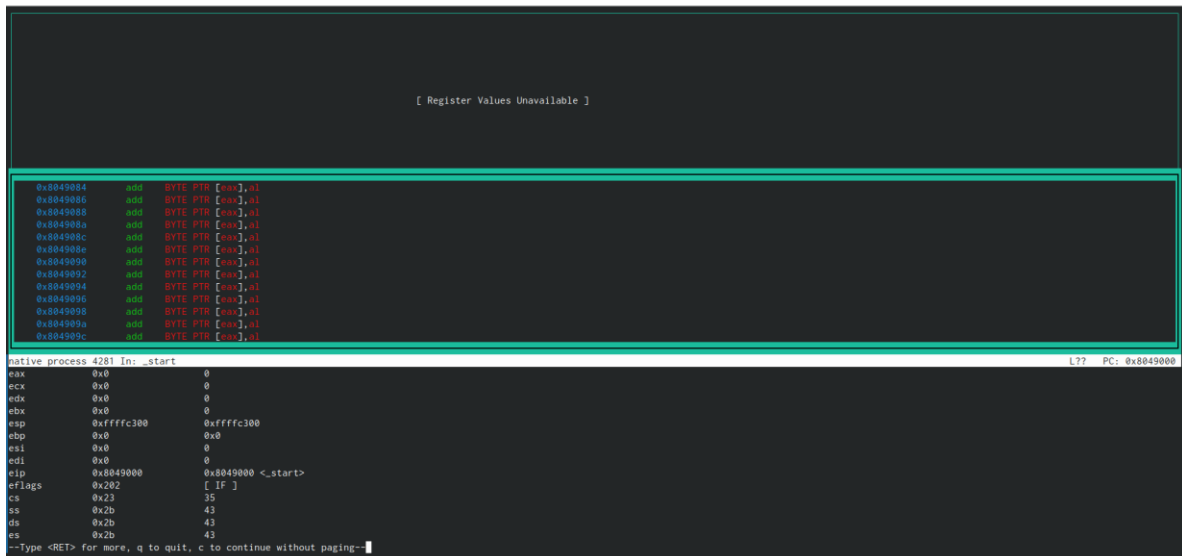
native process 4281 In: _start
L77 PC: 0x0049000
Num Type Disp Enb Address What
1 breakpoint keep y 0x0049000 <_start>
breakpoint already hit 1 time
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x0049000 <_start>
breakpoint already hit 1 time
(gdb) b *0x0049031
Breakpoint 2 at 0x0049031
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x0049000 <_start>
breakpoint already hit 1 time
2 breakpoint keep y 0x0049031 <_start+49>
(gdb)

```

Рис. 3.13: установление точек останова

## 3.4 Работа с данными программы в GDB.

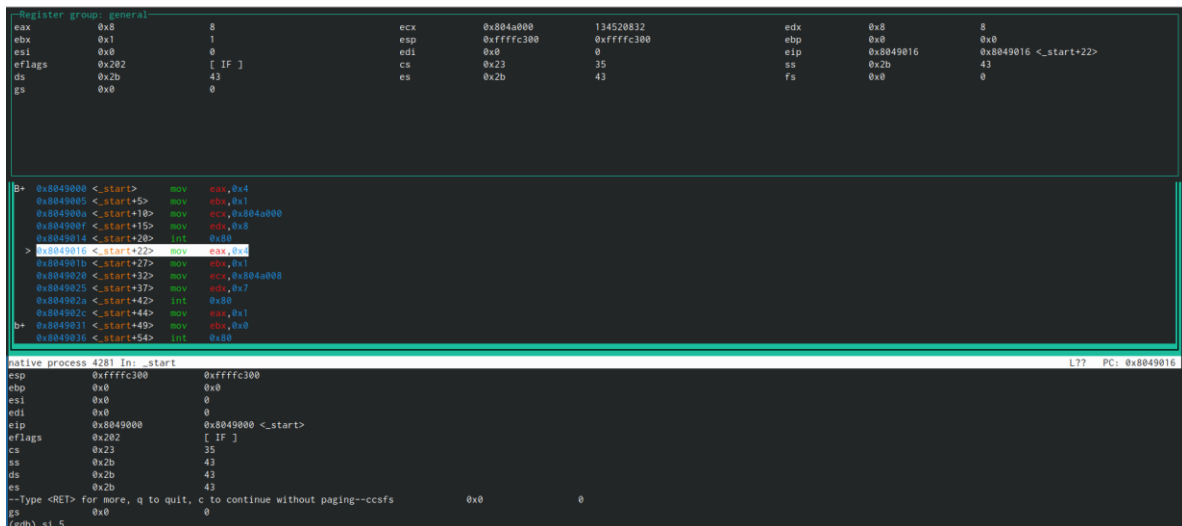
- Выполняю 5 инструкций с помощью команды `stepi` и слежу за изменением значений регистров. (рис. [3.14]).



The screenshot shows the GDB interface. At the top, a message says "[ Register Values Unavailable ]". Below it, a list of assembly instructions is displayed, each preceded by an address and the instruction itself. The instructions are: `add BYTE PTR [eax], al` at addresses `0x8049084` through `0x804909c`. Below the instructions, the register values are listed: `eax 0x0 0`, `ecx 0x0 0`, `edx 0x0 0`, `ebx 0x0 0`, `esp 0xffffc300 0xffffc300`, `ebp 0x0 0x0`, `esi 0x0 0`, `edi 0x0 0`, `eip 0x8049000 0x8049000 <_start>`, `eflags 0x202 [ IF ]`, `cs 0x23 35`, `ss 0x2b 43`, `ds 0x2b 43`, `es 0x2b 43`. At the bottom, there is a prompt: `--Type <RET> for more, q to quit, c to continue without paging--`.

Рис. 3.14: до использования команды `stepi`

(рис. [3.15]).



The screenshot shows the GDB interface after the `stepi` command. The register values are now updated: `eax 0x8 8`, `ecx 0x0 0`, `edx 0x8 8`, `ebx 0x1 1`, `esp 0xffffc300 0xffffc300`, `ebp 0x0 0`, `esi 0x0 0`, `edi 0x0 0`, `eip 0x8049000 0x8049000 <_start>`, `eflags 0x202 [ IF ]`, `cs 0x23 35`, `ss 0x2b 43`, `ds 0x2b 43`, `es 0x2b 43`. The assembly instructions are also updated: `mov eax, 0x4` at `0x8049000`, `mov ebx, 0x1` at `0x8049005`, `mov ecx, 0x804a000` at `0x804900a`, `mov edx, 0x8` at `0x804900f`, `int 0x80` at `0x8049014`, `mov eax, 0x4` at `0x8049016`, `mov ebx, 0x1` at `0x804901b`, `mov ecx, 0x804a008` at `0x8049020`, `mov edx, 0x7` at `0x8049025`, `int 0x80` at `0x804902a`, `mov eax, 0x1` at `0x804902c`, `mov ebx, 0x0` at `0x8049031`, and `int 0x80` at `0x8049036`. At the bottom, there is a prompt: `--Type <RET> for more, q to quit, c to continue without paging--ccsfs 0x0 0`.

Рис. 3.15: после использования команды `stepi`

- Просматриваю значение переменной msg1 по имени с помощью команды x/1sb &msg1 и значение переменной msg2 по ее адресу.(рис. [3.16]).

```

--Register group: general
eax 0x6 8 ecx 0x804a000 134520832 edx 0x6 8
ebx 0x1 1 esp 0xffffc300 0xffffc300 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,0x6
0x8049014 <_start+20> int 0x0
> 0x8049016 <_start+22> mov eax,0x6
0x804901b <_start+27> mov ebx,0x1
0x8049020 <_start+32> mov ecx,0x804a008
0x8049025 <_start+37> mov edx,0x7
0x804902a <_start+42> int 0x0
0x804902c <_start+44> mov eax,0x1
b> 0x8049031 <_start+49> mov ebx,0x0
0x8049036 <_start+54> int 0x0

Active process 4201 Im: _start
L77 PC: 0x8049016
rip 0x8049000 0x8049000 <_start>
eflags 0x202 [ IF ]
cs 0x23 35
ss 0x2b 43
ds 0x2b 43
es 0x2b 43
fs 0x0 0
gs 0x0 0
--Type <RET> for more, q to quit, c to continue without paging--ccsfs 0x0 0
(gdb) si 5
Hello, 0x8049016 in _start ()
(gdb) x/1sb &msg1
0x804a000: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008: "world!\n"
(gdb)

```

Рис. 3.16: просмотр значений переменных

- С помощью команды set изменяю первый символ переменной msg1 и заменяю первый символ в переменной msg2.(рис. [3.17]).

```

(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000: "hello, "
(gdb) set {char}&msg2='b'
(gdb) x/1sb &msg2
0x804a008: "borld!\n"
(gdb)

```

Рис. 3.17: использование команды set

- Вывожу в шестнадцатеричном формате, в двоичном формате и в символьном виде соответственно значение регистра edx с помощью команды print p/F \$val. (рис. [3.18]).

```

-Register group: general
eax      0x8      8      ecx      0x804a000      134520832      edx      0x8      8
ebx      0x1      1      esp      0xffffc300      0xffffc300      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
0x804902c <_start+44> mov     eax,0x1
b+ 0x8049031 <_start+49> mov     ebx,0x0
0x8049036 <_start+54> int     0x80

native process 4281 in: _start
(gdb) set (char*)msg1="h"
'msg1' has unknown type; cast it to its declared type
(gdb) set (char*)msg1="h"
(gdb) x/10b &msg1
0x804a000: "hello, "
(gdb) set (char*)msg2="b"
(gdb) x/10b &msg2
0x804a008: "borld\n"
(gdb) p/x $edx
$1 = 0x8
(gdb) p/t $edx
$2 = 1000
(gdb) p/c $edx
$3 = 8 '\b'
(gdb)

```

Рис. 3.18: вывод значения регистра

- С помощью команды set изменяю значение регистра ebx в соответствии с заданием. (рис. [3.19]).

```

-Register group: general
eax      0x8      8      ecx      0x804a000      134520832      edx      0x8      8
ebx      0x2      2      esp      0xffffc300      0xffffc300      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
0x804902c <_start+44> mov     eax,0x1
b+ 0x8049031 <_start+49> mov     ebx,0x0
0x8049036 <_start+54> int     0x80

native process 4281 in: _start
(gdb) x/10b &msg2
0x804a008: "borld\n"
(gdb) p/x $edx
$1 = 0x8
(gdb) p/t $edx
$2 = 1000
(gdb) p/c $edx
$3 = 8 '\b'
(gdb) set $ebx="2"
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
(gdb)

```

Рис. 3.19: использование команды set для изменения значения регистра

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



виде не изменяется.

- Завершаю выполнение программы с помощью команды `continue` и выхожу из GDB с помощью команды `quit`. (рис. [3.20]).

```

eax    0x8      8      ecx    ecx    0x804a000    134520832    edx    0x8      8
rax    0x1      1      ecx    ecx    0x804a000    134520840    edx    0x7      7
esi    0x0      0      edi    edi    0x0      0      eip    0x8049016    0x8049016 < .start+22>
eflags 0x202    [ IF ]    cs     cs     0x23     16      rip    0x8049031    0x8049031 < .start+40>
ds     0x2b    43      es     es     0x2b    43      fs     0x0      0
gs     0x0      0

B> 0x8049000 < .start> mov    eax,0x4
0x8049016 < .start+22> mov    eax,eax
0x804901b < .start+27> mov    ecx,0x804a000
0x8049020 < .start+32> mov    edx,0x8      8
0x8049025 < .start+37> int    0x0      7
> 0x8049031 < .start+40> mov    eax,0x4
0x8049031 < .start+40> mov    ebx,0x804a000
B> 0x8049031 < .start+40> mov    ebx,0x804a000
0x8049036 < .start+45> int    0x0
0x8049038 add    BYTE PTR [eax],0x1
0x804903a add    BYTE PTR [eax],0x1
B> 0x804903c add    BYTE PTR [eax],0x1
0x804903e add    BYTE PTR [eax],0x1
40 add    BYTE PTR [eax],0x1

native process 4281 in: .start
(gdb) p/x $edx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
(gdb) c
Continuing.
boid!

Breakpoint 2, 0x8049031 in .start ()
(gdb) q
A debugging session is active.

    Inferior 1 [process 4281] will be killed.

Quit anyway? (y or n)

```

Рис. 3.20: завершение работы

### 3.5 Обработка аргументов командной строки в GDB.

- Копирую файл lab8-2.asm с программой из листинга 8.2 в файл с именем lab09-3.asm и создаю исполняемый файл. (рис. [3.21]).

```
[mabaptishta@fedora lab09]$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
[mabaptishta@fedora lab09]$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
[mabaptishta@fedora lab09]$ ld -m elf_i386 -o lab09-3 lab09-3.o
[mabaptishta@fedora lab09]$
```

Рис. 3.21: создание файла

- Загружаю исполняемый файл в отладчик gdb, указывая необходимые аргументы с использованием ключа `-args`. (рис. [3.22]).

```
[mabaptishta@fedora lab09]$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (GDB) Fedora Linux 13.1-2.fc38
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-3...
(gdb)
```

Рис. 3.22: загрузка исполняемого файла в отладчике

- Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее.(рис. [3.23]).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 8.
(gdb) run
Starting program: /home/mabaptishta/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
(gdb)
```

Рис. 3.23: установление точек останова

- Посматриваю вершину стека и позиции стека по их адресам. (рис. [3.24]).

```
(gdb) x/x $esp
0xffffd170: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffd310: "/home/mabaptishta/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd33d: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd34f: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd360: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd362: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 3.24: просмотр значений и введение в стек

### 3.6 Задания для самостоятельной работы.

- 1) Преобразовываю программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции  $f(x)$  как подпрограмму.
- Запускаю код и проверяю, что она работает корректно. (рис. [3.25]).

```
[mabaptishta@fedora lab09]$ cp ~/work/arch-pc/lab08/lab8-4.asm ~/work/arch-pc/lab09/test1.asm
[mabaptishta@fedora lab09]$ nasm -f elf test1.asm
[mabaptishta@fedora lab09]$ ld -m elf_i386 -o test1 test1.o
[mabaptishta@fedora lab09]$ ./test1 2 5 7
Функция :  $f(x) = 30x - 11$ 
Результат : 387
[mabaptishta@fedora lab09]$
```

Рис. 3.25: запуск программы

- 2) Ввожу в файл task1.asm текст программы из листинга 9.3
- При корректной работе программы должно выводиться “25”. Создаю исполняемый файл и запускаю его.(рис. [3.26]).

```
[mabaptishta@fedora lab09]$ nasm -f elf test2.asm
[mabaptishta@fedora lab09]$ ld -m elf_i386 -o test2 test2.o
[mabaptishta@fedora lab09]$ ./test2
Результат: 10
[mabaptishta@fedora lab09]$
```

Рис. 3.26: запуск программы

## 4 Выводы

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

## Список литературы

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

- е изд.— М.: МАКС Пресс, 2011.— URL: [http://www.stolyarov.info/books/asm\\_unix](http://www.stolyarov.info/books/asm_unix).
16. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
17. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер,
18. — 1120 с. — (Классика Computer Science).