РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №9

***Дисциплина*: Архитектура компьютеров**

Студент: Болдырева Дельгир

Группа: НКАбд-01-25

Москва

2025 г.

Оглавление

1 Цель работы

Приобретение навыков написания программ с использованием подпрограмм.

Знакомство с методами отладки при помощи GDB и его основными

Возможностями

2 Задания

Реализация подпрограмм в NASM, Отладка программам с помощью GDB,

Задание для самостоятельной работы

3 Теоретическое введение

Отладка — процесс поиска и исправления ошибок в программе. Как правило, он состоит из четырех основных этапов:

1. Работает некорректно.

вызывающего проблему.

Обнаружение ошибки — понимание того, что программа

2. Поиск места ошибки — локализация фрагмента кода,

3. возникает ошибка.

Определение причины — анализ, почему в данном месте

4. Исправление ошибки — изменение кода для устранения

проблемы, после чего цикл может повториться для других ошибок.

Можно выделить три основных типа ошибок:

- Синтаксические ошибки — нарушение правил языка программирования; обнаруживаются на этапе трансляции (компиляции/интерпретации).
- Семантические (логические) ошибки — программа запускается и выполняется, но выдает непредусмотренный или неверный результат из-за ошибочной логики.
- Ошибки в процессе выполнения — возникают во время работы программы и могут приводить к её аварийному завершению (например, деление на ноль или переполнение памяти).

Наиболее сложным часто оказывается второй этап — поиск места ошибки.

Для этого полезно разбивать программу на части и проверять их отдельно, используя методы отладки (например, вывод промежуточных значений, пошаговое выполнение, тестирование модулей).

После локализации ошибки обычно проще понять её причину и внести необходимые исправления. Однако после исправления одной ошибки могут обнаруживаться другие, и процесс отладки продолжается.

4 Выполнение работы

4.1 Реализация подпрограмм в Nasm

Создадим каталог для выполнения лабораторной работы №9, перейдем в него и создадим файл lab09-1.asm

Рис.4.1.1

В качестве примера рассмотрим программу вычисления арифметического выражения $f(x) = 2x + 7$ с помощью подпрограммы _calcul. В данном примере x вводится с клавиатуры, а само выражение вычисляется в подпрограмме.

Внимательно изучим текст программы и перепишем его

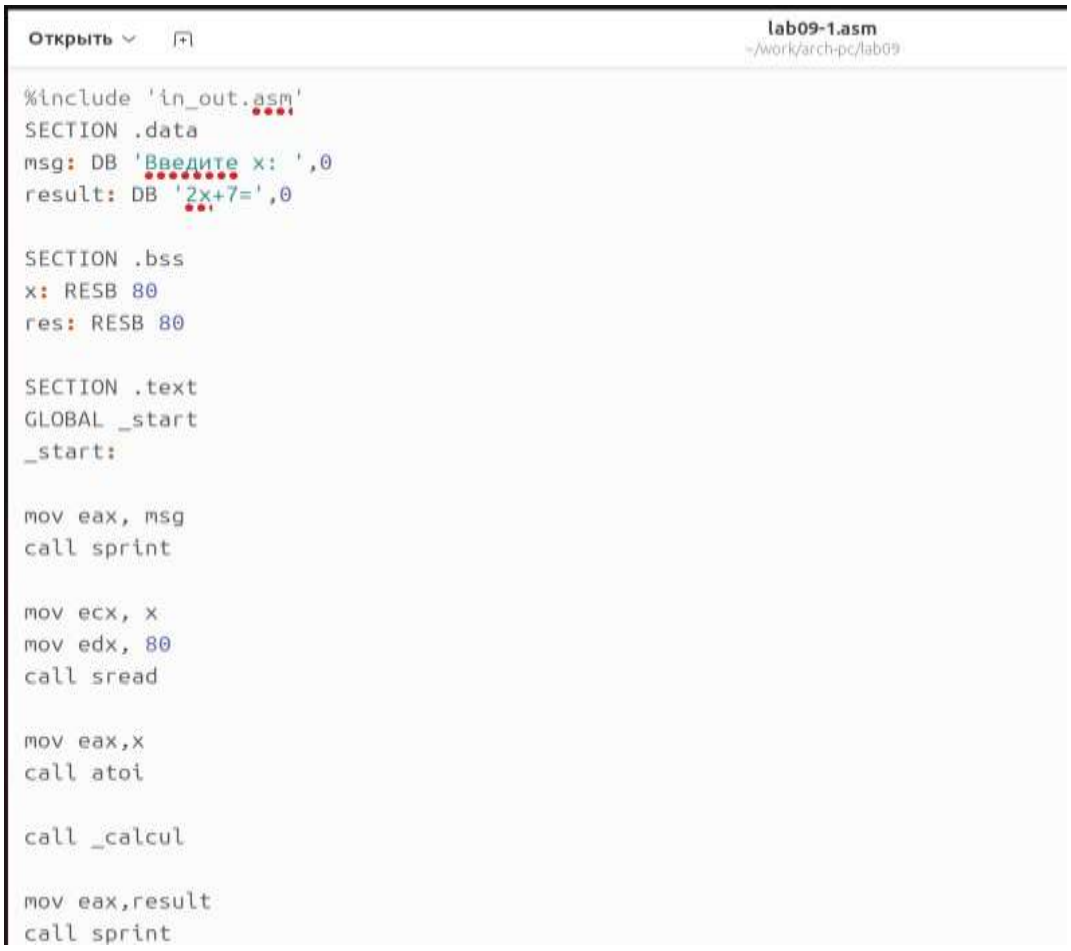


Рис.4.1.2

Создадим исполняемый файл и проверим его работу

```
boldyrevadelgir@deboldihreva1:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
boldyrevadelgir@deboldihreva1:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
boldyrevadelgir@deboldihreva1:~/work/arch-pc/lab09$ ./lab09-1
```

```
Введите x: 10
2x+7=27
boldyrevadelgir@deboldihreva1:~/work/arch-pc/lab09$
```

Рис.4.1.3

Изменим текст программы, добавив подпрограмму _subcalcul в подпрограмму _calcul, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$.Т.е. x передается в подпрограмму _calcul из нее в подпрограмму _subcalcul, где вычисляется выражение $g(x)$, результат возвращается в _calcul и вычисляется выражение $f(g(x))$. Результат возвращается в основную программу для вывода результата на экран

Открыть ▾ lab09-1.asm

```
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2(3x-1)+7=',0

SECTION .bss
x: RESB 80
res: RESB 80

SECTION .text
GLOBAL _start
_start:
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
```

Рис.4.1.4

Создадим исполняемый файл и проверим его работу

```
boldyrevadelgir@deboldihreva1:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
boldyrevadelgir@deboldihreva1:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
boldyrevadelgir@deboldihreva1:~/work/arch-pc/lab09$ ./lab09-1
```

```
Введите x: 10
2(3x-1)+7=65
boldyrevadelgir@deboldihreva1:~/work/arch-pc/lab09$
```

Рис.4.1.5

4.2 Отладка программам с помощью GDB

Создадим файл lab09-2.asm с текстом программы из Листинга

```
boldyrevadelgir@deboldihreva1:~/work/arch-pc/lab09$ touch lab09-2.asm
boldyrevadelgir@deboldihreva1:~/work/arch-pc/lab09$
```

Открыть

lab09-2.asm
~/work/arch-pc/lab09

lab09-1.asm | lab09-2.asm

```
SECTION .data
msg1: db "Hello, ",0x0
msg1Len: equ $ - msg1

msg2: db "world!",0xa
msg2Len: equ $ - msg2

SECTION .text
global _start
_start:
mov eax, 4
mov ebx, 1
mov ecx, msg1
mov edx, msg1Len
int 0x80

mov eax, 4
mov ebx, 1
mov ecx, msg2
mov edx, msg2Len
int 0x80

mov eax, 1
mov ebx, 0
int 0x80
```

Рис.4.2.1

Загрузим исполняемый файл в отладчик gdb

```
boldyrevadelgir@deboldihreva1:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

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

Рис.4.2.2

Проверьте работу программы,запустив ее в оболочке GDB с помощью

команды run

```
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 4696) exited normally]
(gdb)
```

Рис.4.2.3

Для более подробного анализа программы установим брейкпоинт на метку

_start, с которой начинается выполнение любой ассемблерной программы, и

запустим её

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 11.
(gdb)
```

```
(gdb) run
Starting program: /home/boldyrevadelgir/work/arch-pc/lab09/lab09-2

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

Рис.4.2.4

Посмотрим дисассимилированный код программы с помощью команды

disassemble начиная с метки _start

```
(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.4.2.5

Переключимся на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel

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

Рис.4.2.6

Перечислим различия отображения синтаксиса машинных команд в режимах ATT и Intel. Включим режим псевдографики для более удобного анализа Программы

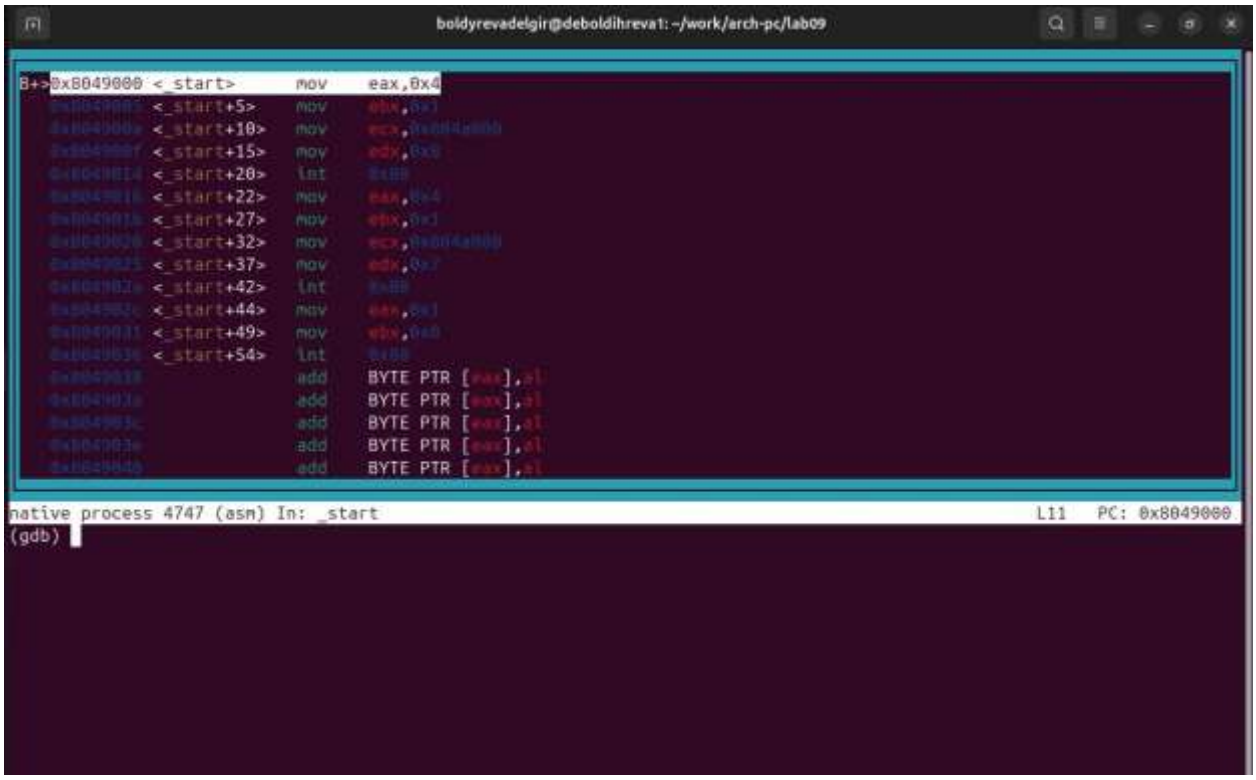


Рис.4.2.7

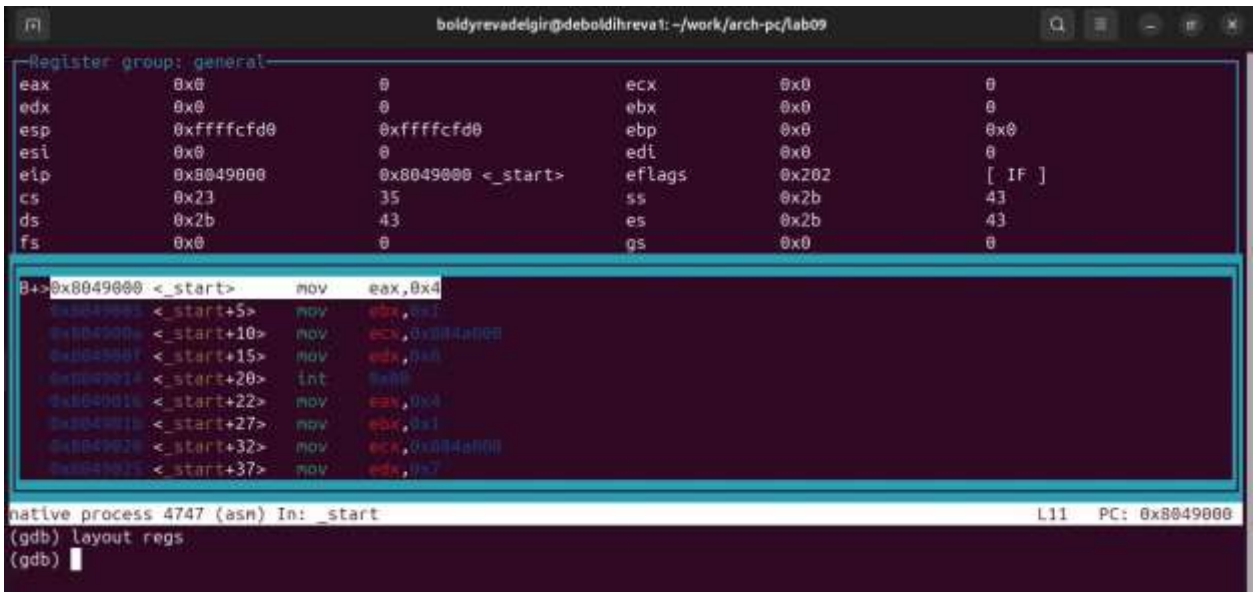


Рис.4.2.7

На предыдущих шагах была установлена точка останова по имени метки(_start).Про верьте это с помощью команды info breakpoints(кратко i b)

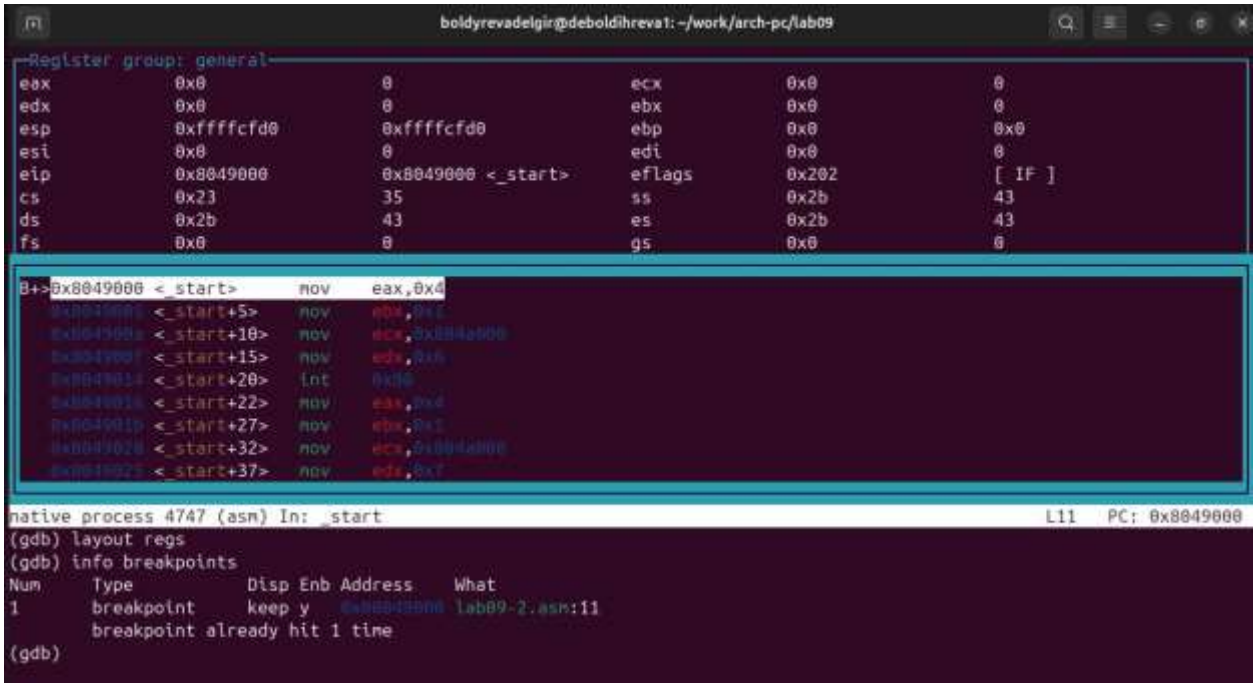


Рис.4.2.8

Установим еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции. Определите адрес предпоследней инструкции (mov ebx,0x0) и установим точку останова.

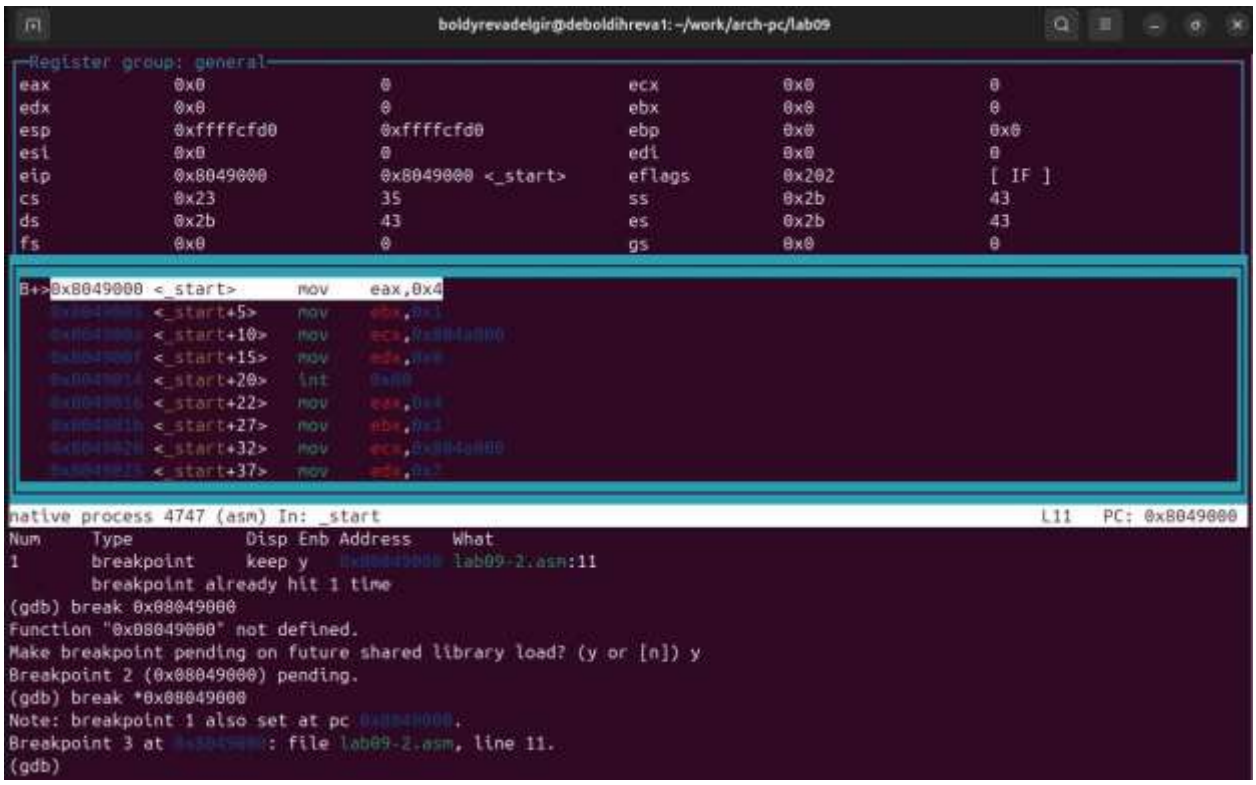


Рис.4.2.9

Посмотрим информацию о всех установленных точках останова

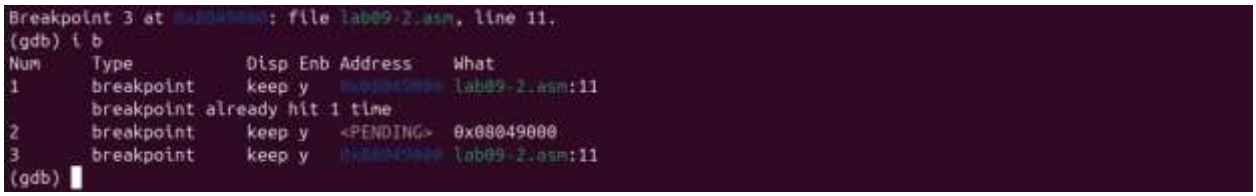
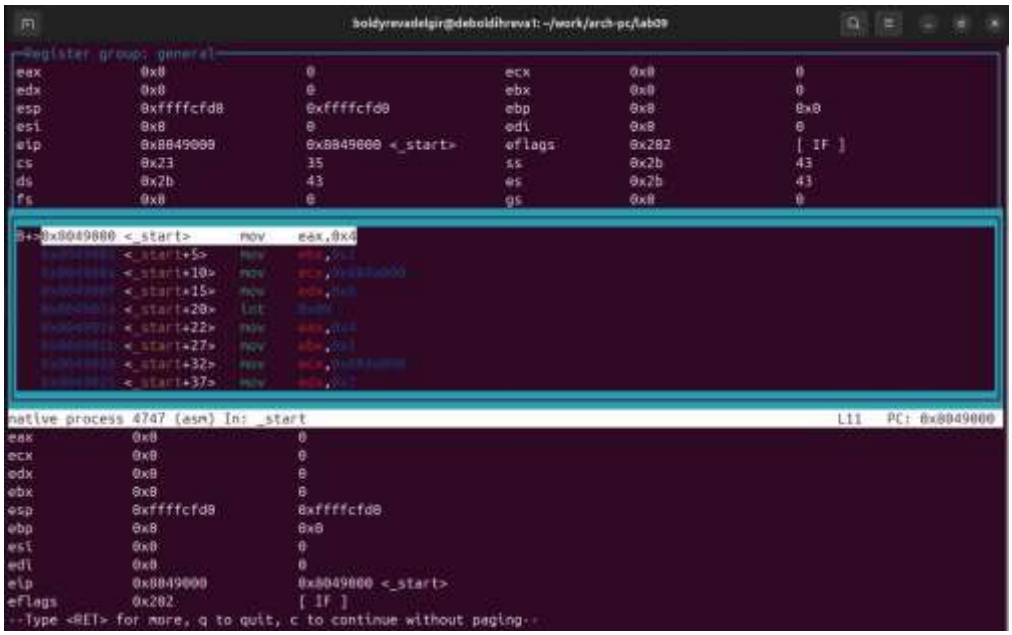


Рис.4.2.10

Посмотрим содержимое регистров также можно с помощью команды info

Registers



Выполним 5 инструкций с помощью команды stepi(или si) и проследите за изменением значений регистров.

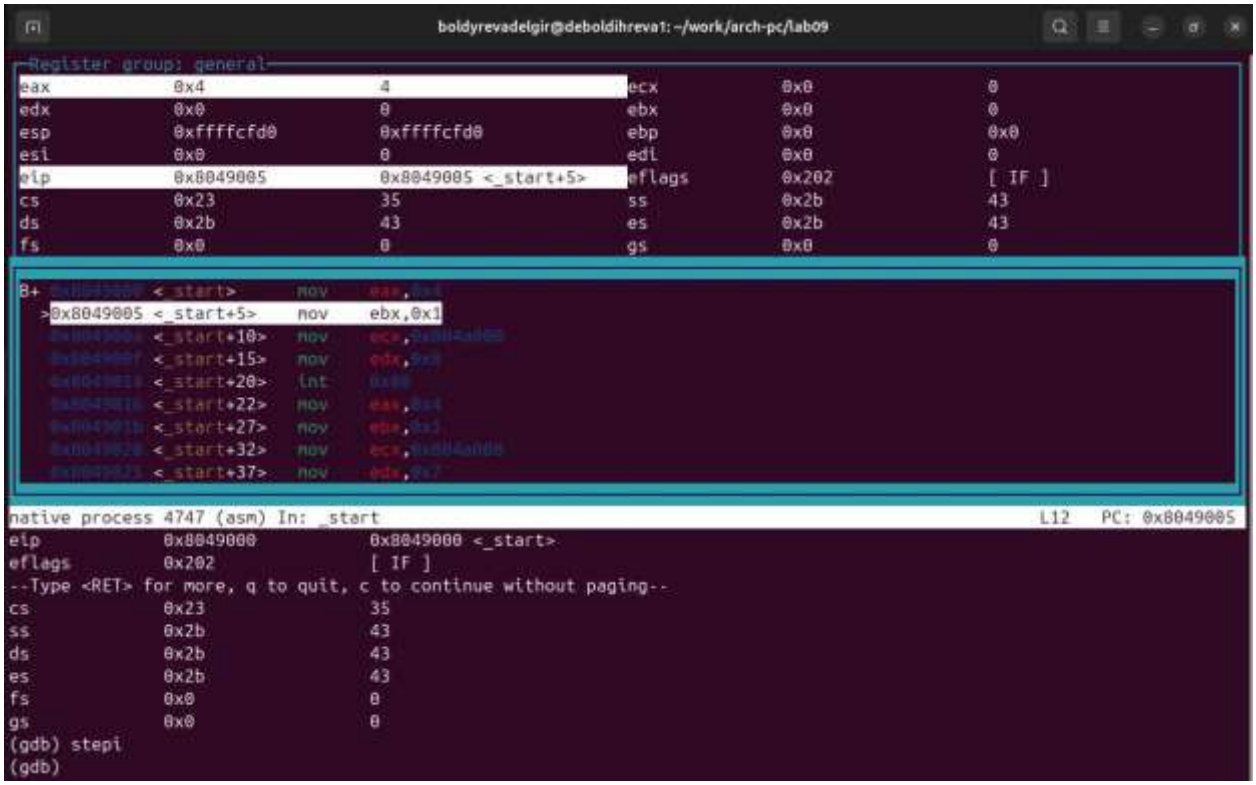


Рис.4.2.11(stepi)

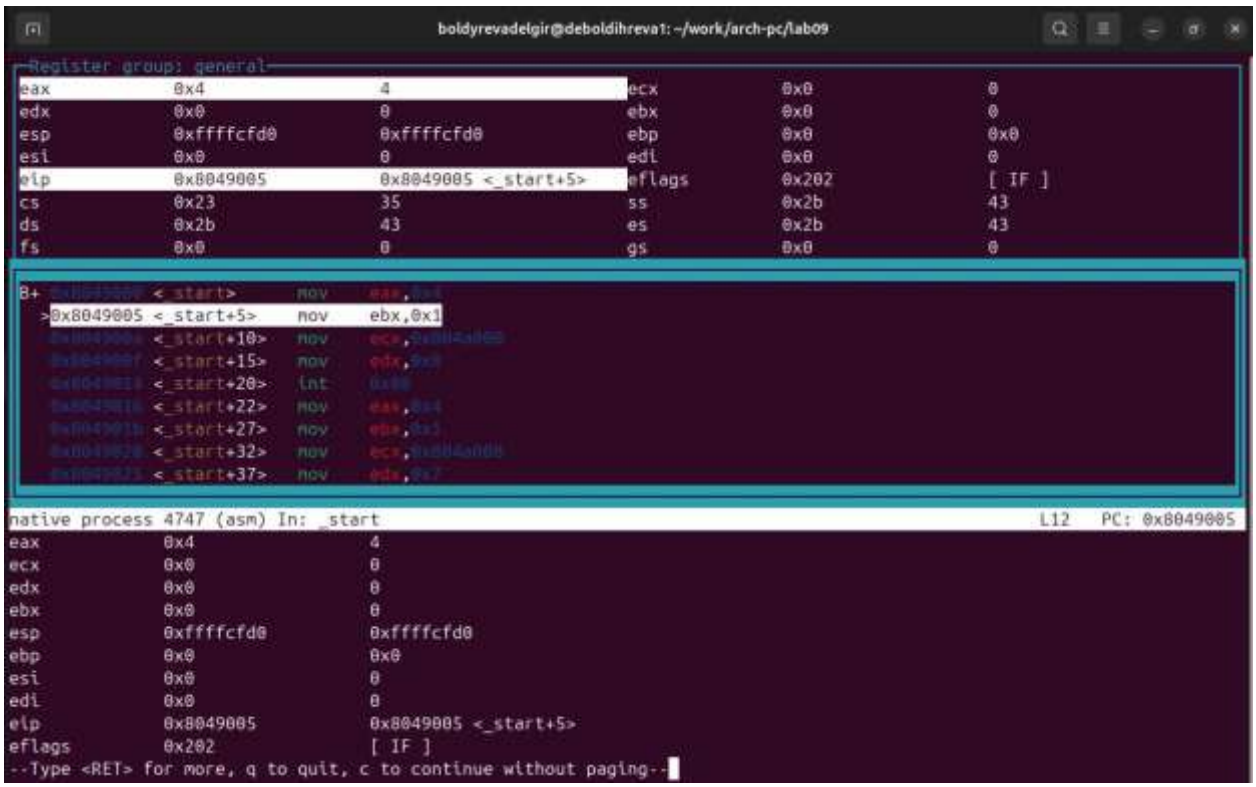


Рис.4.2.12(info registers)

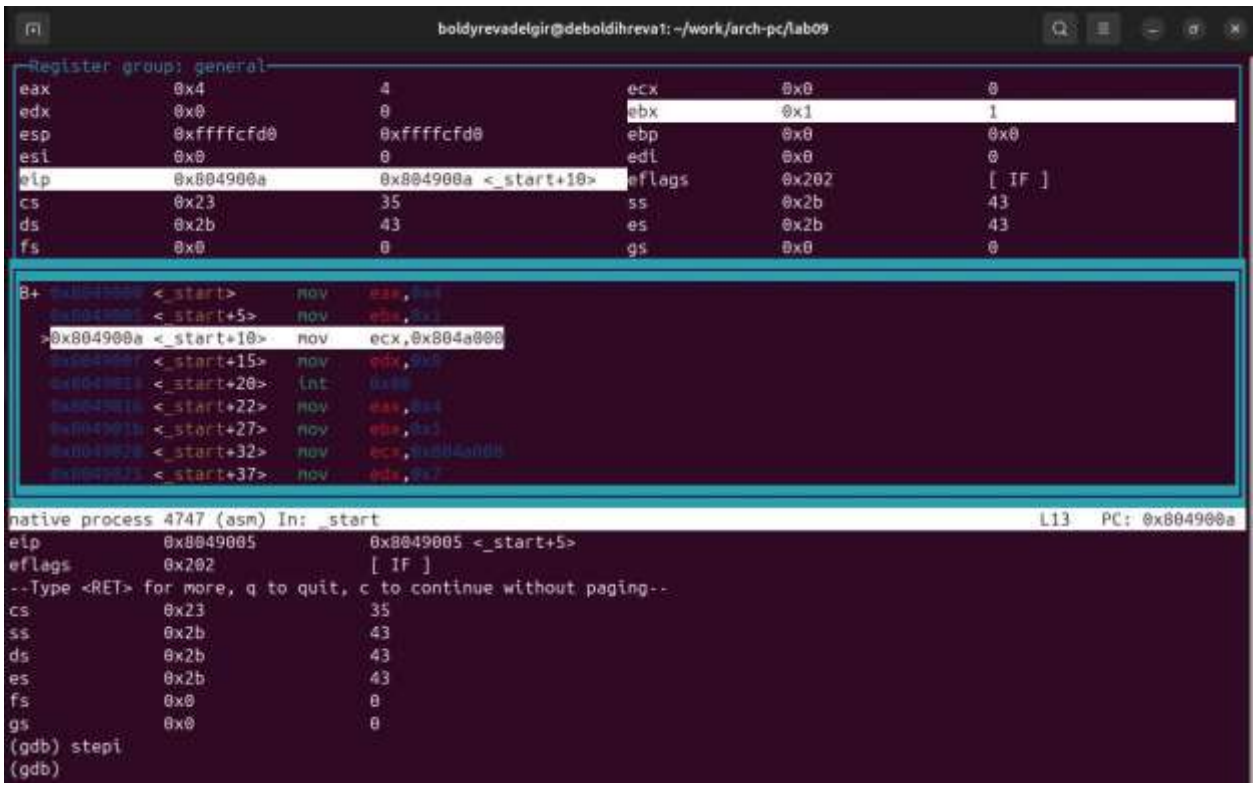


Рис.4.2.13(stepi)

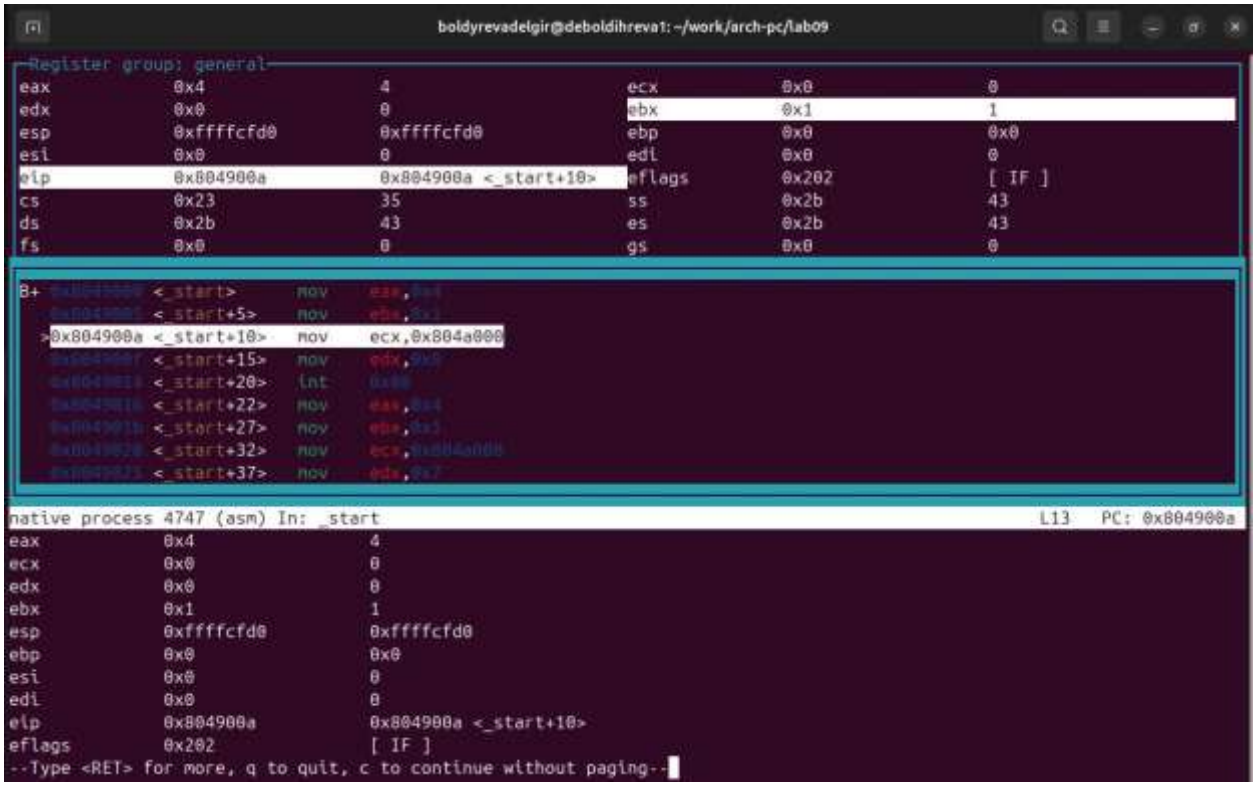


Рис.4.2.14(info registers)

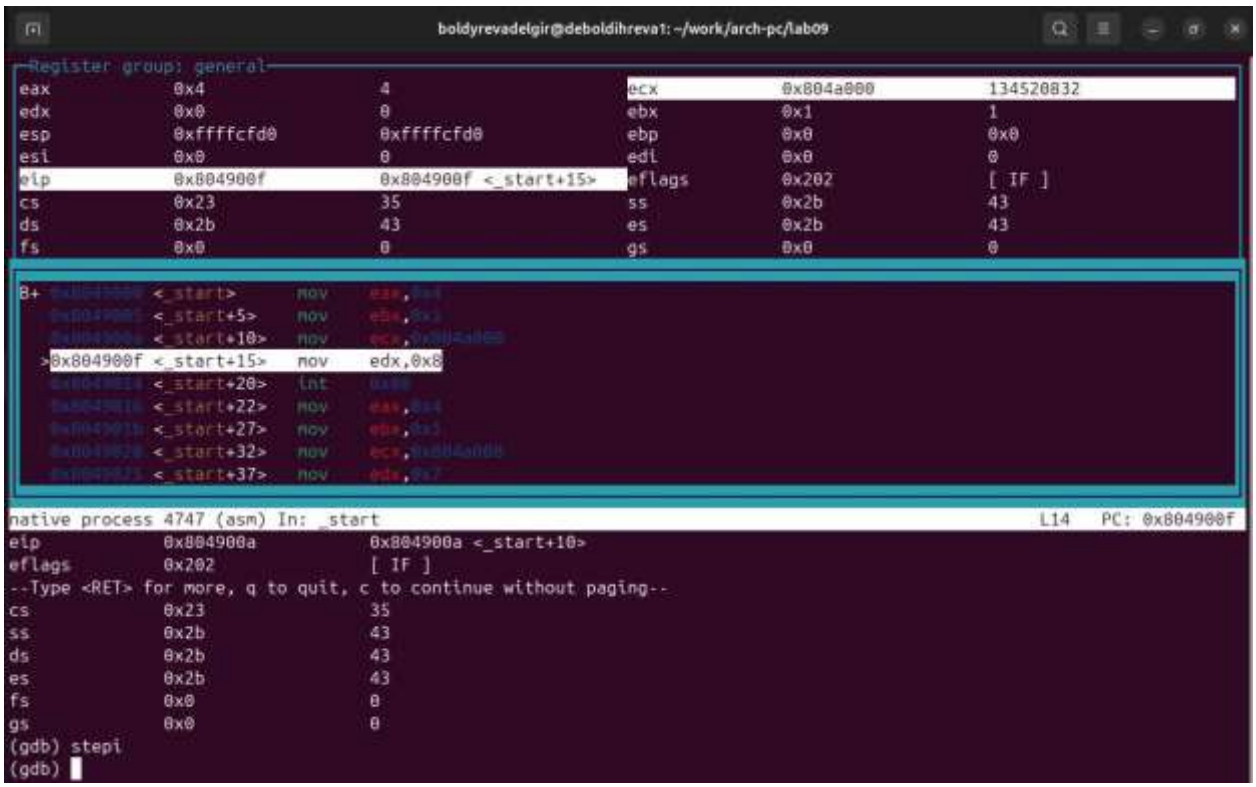


Рис.4.2.15(stepi)

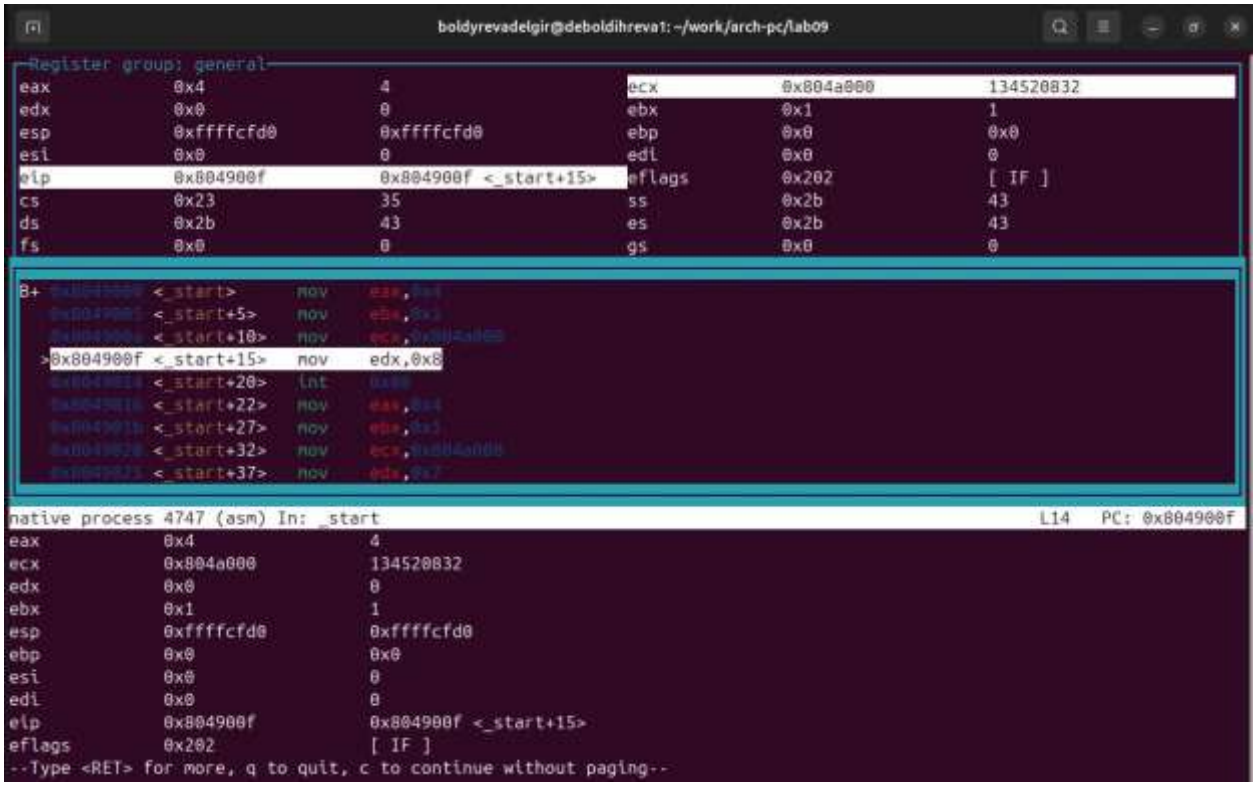


Рис.4.2.16(info registers)

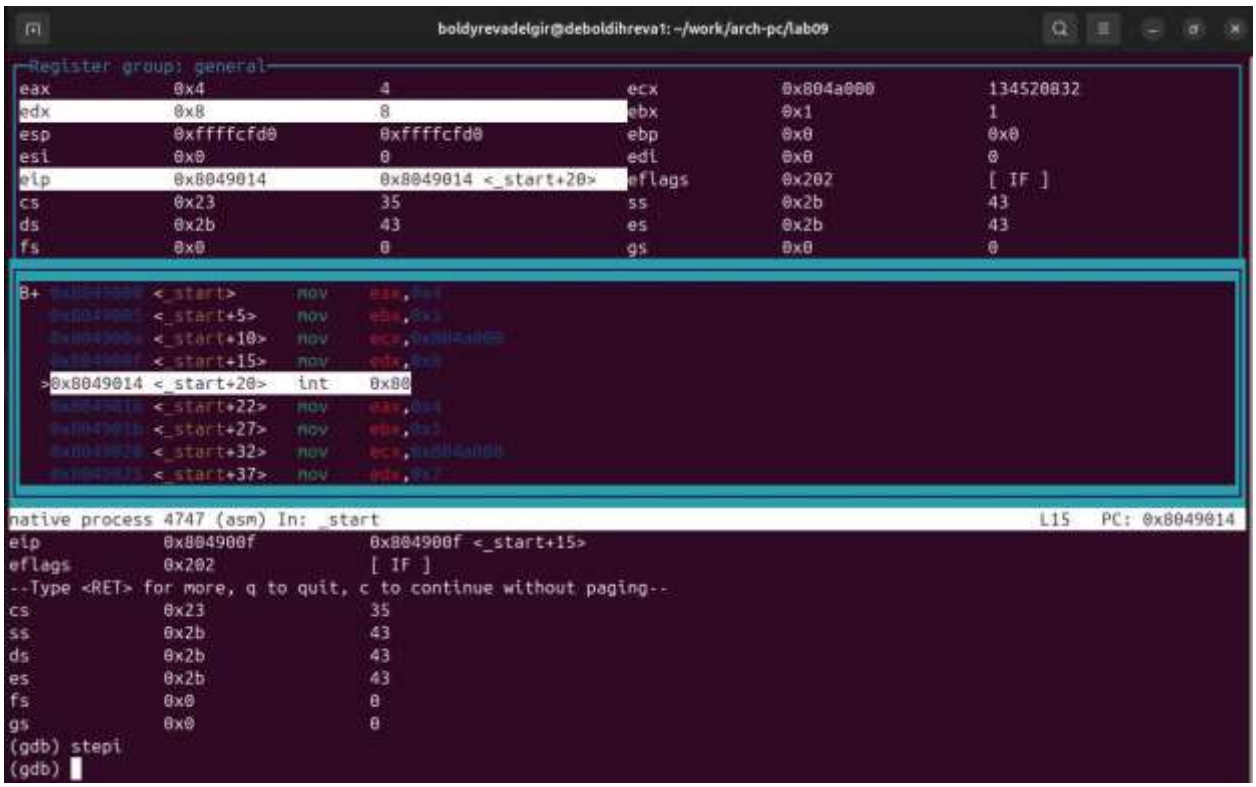


Рис.4.2.17(stepi)

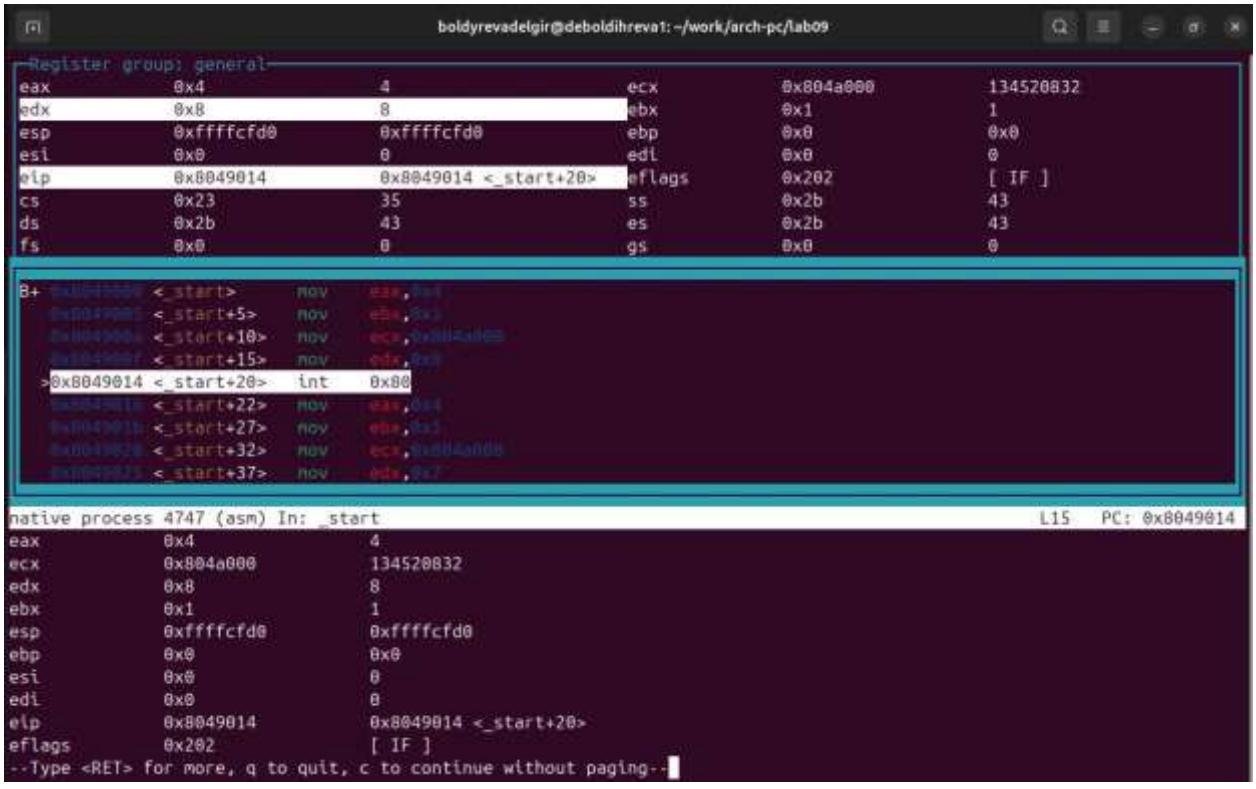


Рис.4.2.18(info registers)

При выполнении команды stepi (одной инструкции процессора) обычно изменяются:

1. RIP/EIP -указатель на следующую инструкцию (всегда меняется)
2. RAX/EAX -часто используется для вычислений и возврата значений
3. RSP/ESP -указатель стека (меняется при push/pop)
4. Флаги (ZF, CF, SF, OF) -после арифметических операций

Посмотрим значение переменной msg1 по имени

Рис.4.2.19

Посмотрим значение переменной msg2 по адресу

Рис.4.2.20

Изменим первый символ переменной msg1

Рис.4.2.21

Заменим любой символ во второй переменной msg2

Рис.4.2.22

Выведем в различных форматах (в шестнадцатеричном формате, вдвоичном формате и в символьном виде) значение регистра edx.

С помощью команды set изменим значение регистра ebx

```
(gdb) p/t $edx
$4 = 1000
(gdb) p/x $edx
$5 = 0x8
(gdb) set $ebx='2'
(gdb) p/s $ebx
$6 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$7 = 2
(gdb)
```

Рис.4.2.23

Завершим выполнение программы с помощью команды continue (сокращенно c) или stepi (сокращенно si) и выйдите из GDB с помощью команды quit(сокращенно q)

```
(gdb) c
Continuing.
hello, world!
[Inferior 1 (process 4747) exited normally]
(gdb)
```

Рис.4.2.24

4.3 Обработка аргументов командной строки в GDB

Скопируем файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки в файл с именем lab09-3.asm

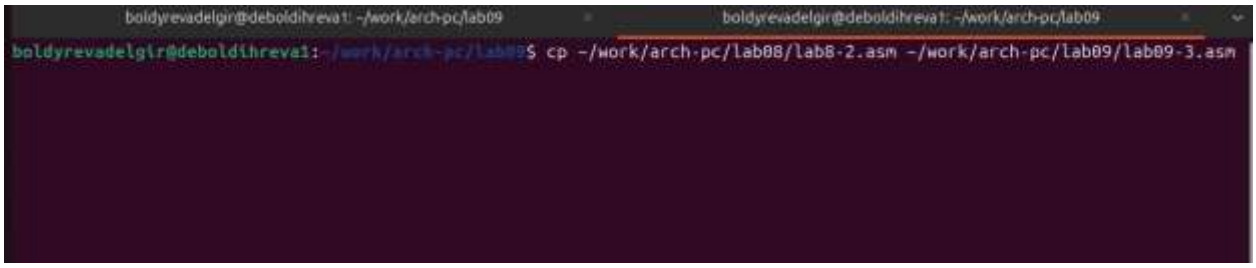


Рис.4.3.1

Создадим исполняемый файл

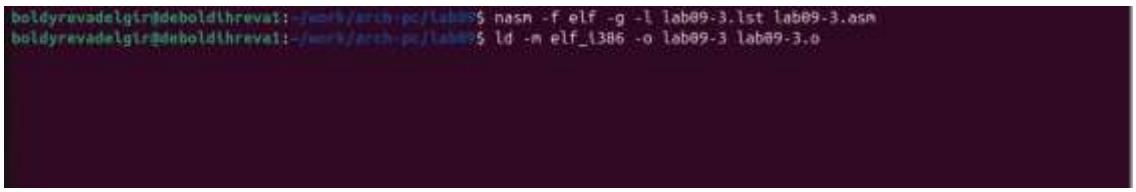


Рис.4.3.2

Загрузим исполняемый файл в отладчик, указав аргументы

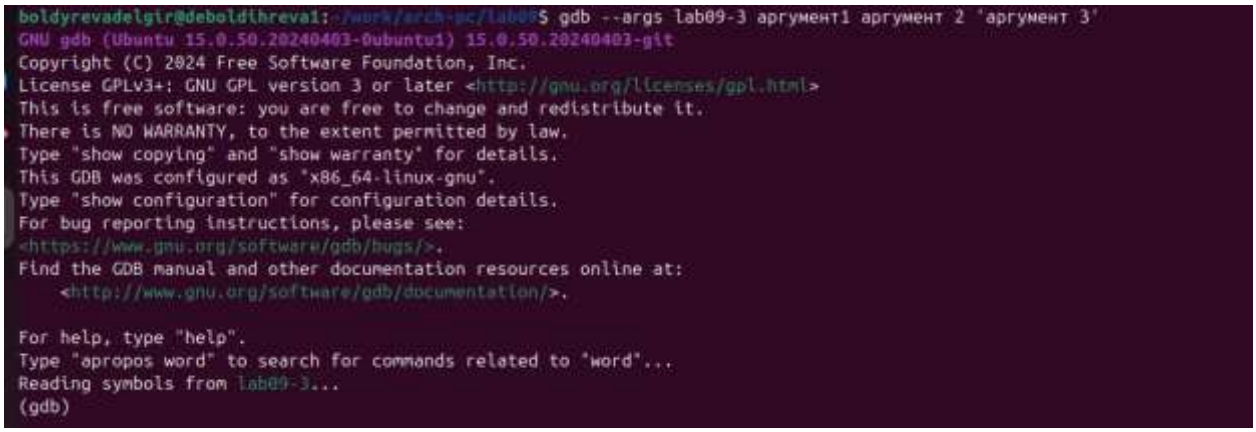


Рис.4.3.3

Установим точку останова перед первой инструкцией в программе и запустим ее.

Рис.4.3.4

Рис.4.3.5

Посмотрим остальные позиции стека–по адесу [esp+4] располагается адрес в памяти где находиться имя программы,по адесу [esp+8] храниться адрес первого аргумента,по аресу [esp+12]–второго ит.д

Рис.4.3.6

Каждый элемент этого массива — это адрес (указатель), который в 32-битной системе занимает 4 байта. Поэтому чтобы перейти, нужно прибавить 4 байта.

5 Задание для самостоятельной работы

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

Рис.5.1

Рис.5.2

Рис.5.3

В листинге приведена программа вычисления выражения (3+2)∗4+5.Призапуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее

```
boldyrevadelgir@deboldihreva1:~/work/arch-pc/lab09$ touch lab09-5
boldyrevadelgir@deboldihreva1:~/work/arch-pc/lab09$ gedit lab09-5
```

Рис.5.4

```
boldyrevadelgir@deboldihreva1:~/work/arch-pc/lab09$ nasm -f elf -g lab09-5.asm -o lab09-5.o
boldyrevadelgir@deboldihreva1:~/work/arch-pc/lab09$ ld -m elf_i386 lab09-5.o -o lab09-5
boldyrevadelgir@deboldihreva1:~/work/arch-pc/lab09$ gdb lab09-5
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-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...
```

Рис.5.5

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-5.asm, line 10.
(gdb)
```

Рис.5.6

```
(gdb) run
Starting program: /home/boldyrevadelgir/work/arch-pc/lab09/lab09-5

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000

Breakpoint 1, _start () at lab09-5.asm:10
10      mov ebx, 3
(gdb)
```

Рис.5.7

```
(gdb) print $eax
$1 = 0
(gdb) continue
Continuing.
Результат: 8
[Inferior 1 (process 5983) exited normally]
(gdb)
```

Рис.5.8

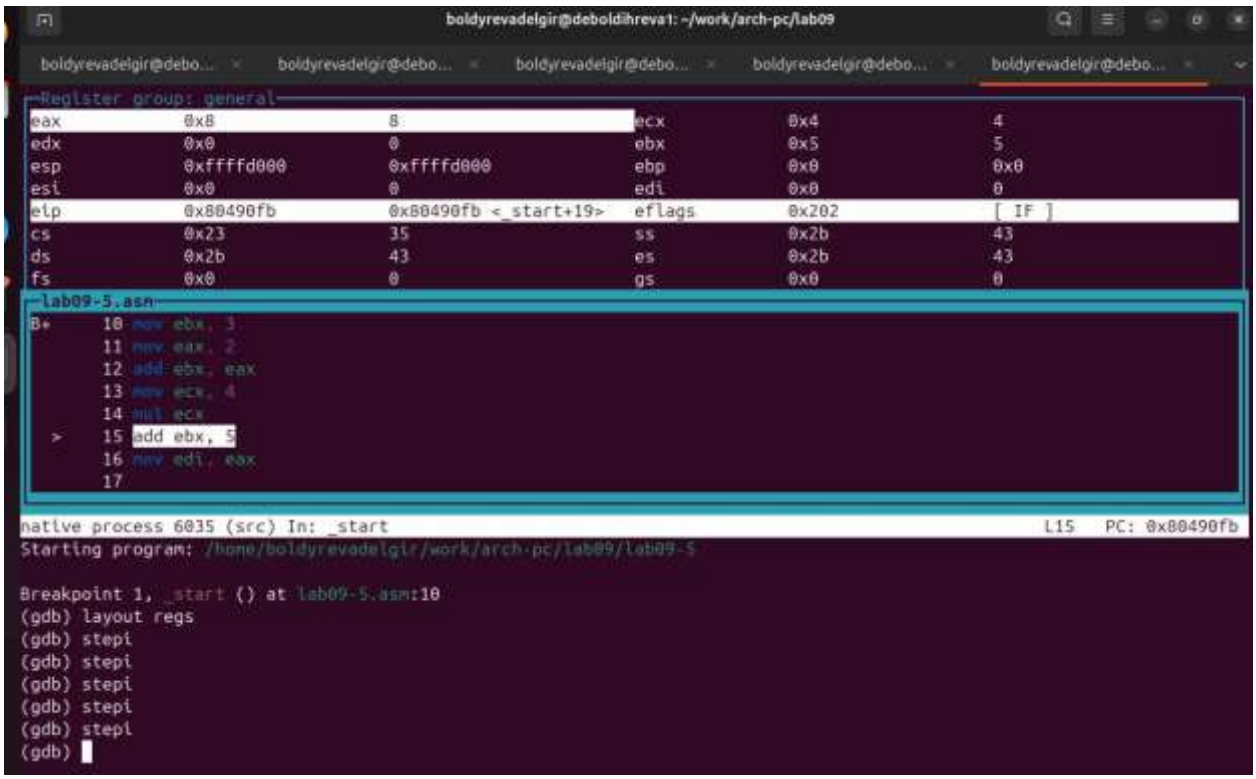


Рис.5.9

Рис.5.10

```
boldyrevadelgir@deboldihreva1:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
boldyrevadelgir@deboldihreva1:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab09-5.o
boldyrevadelgir@deboldihreva1:~/work/arch-pc/lab09$ ./lab09-5
Результат: 20
boldyrevadelgir@deboldihreva1:~/work/arch-pc/lab09$
```

Рис.5.11

6 Выводы

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

Список литературы

1. GDB:TheGNUProjectDebugger.—URL:https://www.gnu.org/software/gdb/.

2. GNUBashManual.—2016.—URL:https://www.gnu.org/software/bash/manual/.

3. Midnight CommanderDevelopment Center.—2021.—URL: https://midnightcommander. org/.

4. NASMAssemblyLanguageTutorials.—2021.—URL: https://asmtutor.com/.

5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 c. —(In a Nutshell). —ISBN 0596009658.—URL: http://www.amazon.com/Learning bash-Shell-Programming-Nutshell/dp/0596009658.

6. RobbinsA. Bash Pocket Reference.—O'Reilly Media,2016.—156 c.—ISBN 978-1491941591.

7. TheNASMdocumentation.—2021.—URL:https://www.nasm.us/docs.php.

8. Zarrelli G. Mastering Bash.—Packt Publishing,2017.—502 c.—ISBN 9781784396879.