РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 9

дисциплина: Архитектура компьютера

Студент: Володина Алиса Алексеевна

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. **Обнаружение ошибки** — понимание того, что программа работает некорректно.
2. **Поиск места ошибки** — локализация фрагмента кода, вызывающего проблему.
3. **Определение причины** — анализ, почему в данном месте возникает ошибка.
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

Реализация подпрограмм в NASM

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

Рисунок 0.1(создание каталога)

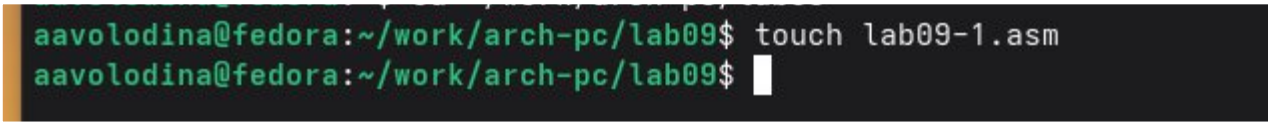

Рисунок 0.2(создание файла)

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

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

ret
```

Рисунок 0.3(текст программы)

Создадим исполняемый файл и проверим его работу

```
aavolodina@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
aavolodina@fedora:~/work/arch-pc/lab09$
```

Рисунок 0.4(создание исполняемого файла)

```
aavolodina@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab0
aavolodina@fedora:~/work/arch-pc/lab09$
```

Рисунок 0.5(создание исполняемого файла)

```
aavolodina@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x:
```

Рисунок 0.6(запуск исполняемого файла)

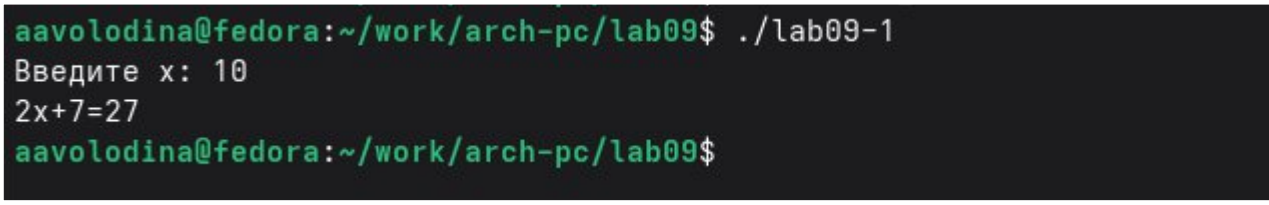

Рисунок 0.7(проверка работы программы)

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.

```
%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
```

Рисунок 0.8 (программа с добавлением подпрограммы)

Создадим исполняемый файл и проверим его работу

```
aavolodina@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
aavolodina@fedora:~/work/arch-pc/lab09$
```

Рисунок 0.9 (создание исполняемого файла)

```
aavolodina@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab
aavolodina@fedora:~/work/arch-pc/lab09$
```

Рисунок 0.10 (создание исполняемого файла)

```
aavolodina@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x:
```

Рисунок 0.11 (запуск исполняемого файла)

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

Рисунок 0.12 (проверка работы программы)

Отладка программам с помощью GDB

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

```
aavolodina@fedora:~/work/arch-pc/lab09$ touch lab09-2.asm
aavolodina@fedora:~/work/arch-pc/lab09$
```

Рисунок 0.13 (создание файла)

```
 1 SECTION .data
 2 msg1: db 'Hello, ', 0x0
 3 msg1Len: equ $ -msg1
 4
 5 msg2: db 'World!', 0xa
 6 msg2Len: equ $ -msg2
 7
 8 SECTION .text
 9 global _start
10 _start:
11 mov eax, 4
12 mov ebx, 1
13 mov ecx, msg1
14 mov edx, msg1Len
15 int 0x80
16
17 mov eax, 4
18 mov ebx, 1
19 mov ecx, msg2
20 mov edx, msg2Len
21 int 0x80
22
23 mov eax, 1
24 mov ebx, 0
25 int 0x80
```

Рисунок 0.14 (текст программы)

Получим исполняемый файл

```
aavolodina@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
aavolodina@fedora:~/work/arch-pc/lab09$
```

Рисунок 0.15 (создание исполняемого файла)

```
aavolodina@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
aavolodina@fedora:~/work/arch-pc/lab09$
```

Рисунок 0.16 (создание исполняемого файла)

Загрузим исполняемый файл в отладчик gdb

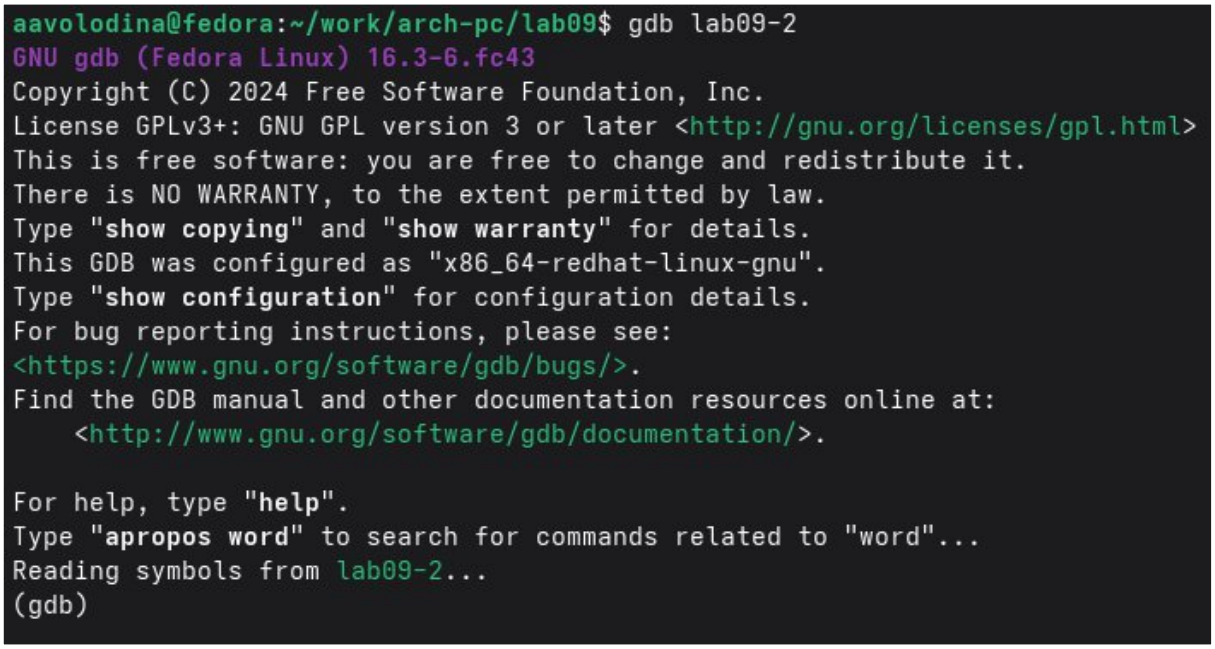

Рисунок 0.17 (Загрузка исполняемого файла в отладчик gdb)

Проверьте работу программы,запустив ее в оболочке GDB с помощью команды run

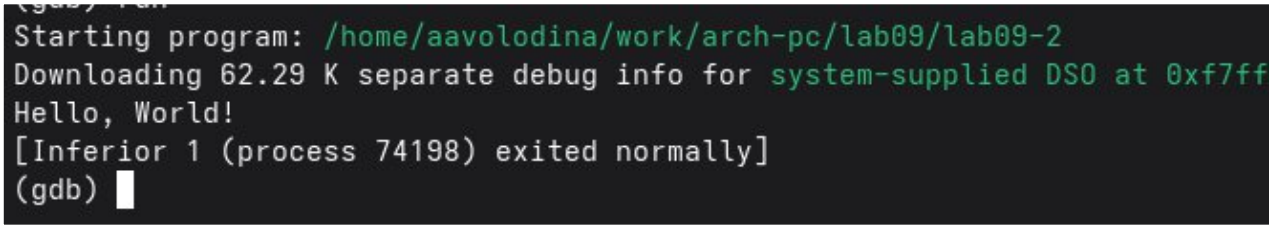

Рисунок 0.18(запуск программы при помощи run)

Для более подробного анализа программы установим брейкпоинт на метку _start, с которой начинается выполнение любой ассемблерной программы, и запустим её

Рисунок 0.19(установка брейкпоинта)

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

Breakpoint 1, _start () at lab09-2.asm:12
12      mov eax, 4
(gdb)
```

Рисунок 0.20(запуск программы)

Посмотрим дисассимилированный код программы с помощью команды disassemble начиная с метки _start

```
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08048080 <+0>:     mov    $0x4,%eax
   0x08048085 <+5>:     mov    $0x1,%ebx
   0x0804808a <+10>:    mov    $0x8049000,%ecx
   0x0804808f <+15>:    mov    $0x8,%edx
   0x08048094 <+20>:    int    $0x80
   0x08048096 <+22>:    mov    $0x4,%eax
   0x0804809b <+27>:    mov    $0x1,%ebx
   0x080480a0 <+32>:    mov    $0x8049008,%ecx
   0x080480a5 <+37>:    mov    $0x7,%edx
   0x080480aa <+42>:    int    $0x80
   0x080480ac <+44>:    mov    $0x1,%eax
   0x080480b1 <+49>:    mov    $0x0,%ebx
   0x080480b6 <+54>:    int    $0x80
End of assembler dump.
(gdb)
```

Рисунок 0.21(просмотр дисассимилированного кода программы)

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:
=> 0x08048080 <+0>:     mov    eax,0x4
   0x08048085 <+5>:     mov    ebx,0x1
   0x0804808a <+10>:    mov    ecx,0x8049000
   0x0804808f <+15>:    mov    edx,0x8
   0x08048094 <+20>:    int    0x80
   0x08048096 <+22>:    mov    eax,0x4
   0x0804809b <+27>:    mov    ebx,0x1
   0x080480a0 <+32>:    mov    ecx,0x8049008
   0x080480a5 <+37>:    mov    edx,0x7
   0x080480aa <+42>:    int    0x80
   0x080480ac <+44>:    mov    eax,0x1
   0x080480b1 <+49>:    mov    ebx,0x0
   0x080480b6 <+54>:    int    0x80
End of assembler dump.
(gdb)
```

Рисунок 0.22(переключение на отображение команд с Intel'овским синтаксисом)

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

```
B+>0x8048080 <_start>       mov    eax,0x4
   0x8048085 <_start+5>     mov    ebx,0x1
   0x804808a <_start+10>    mov    ecx,0x8049000
   0x804808f <_start+15>    mov    edx,0x8
   0x8048094 <_start+20>    int    0x80
   0x8048096 <_start+22>    mov    eax,0x4
   0x804809b <_start+27>    mov    ebx,0x1
   0x80480a0 <_start+32>    mov    ecx,0x8049008
   0x80480a5 <_start+37>    mov    edx,0x7
   0x80480aa <_start+42>    int    0x80
   0x80480ac <_start+44>    mov    eax,0x1
   0x80480b1 <_start+49>    mov    ebx,0x0
   0x80480b6 <_start+54>    int    0x80
   0x80480b8                add    BYTE PTR [eax],al
   0x80480ba                add    BYTE PTR [eax],al
   0x80480bc                add    BYTE PTR [eax],al
   0x80480be                add    BYTE PTR [eax],al
   0x80480c0                add    BYTE PTR [eax],al
   0x80480c2                add    BYTE PTR [eax],al
   0x80480c4                add    BYTE PTR [eax],al
   0x80480c6                add    BYTE PTR [eax],al
   0x80480c8                add    BYTE PTR [eax],al
   0x80480ca                add    BYTE PTR [eax],al
native process 74285 (asm) In: _start
(gdb)
```

Рисунок 0.23 (layout asm)

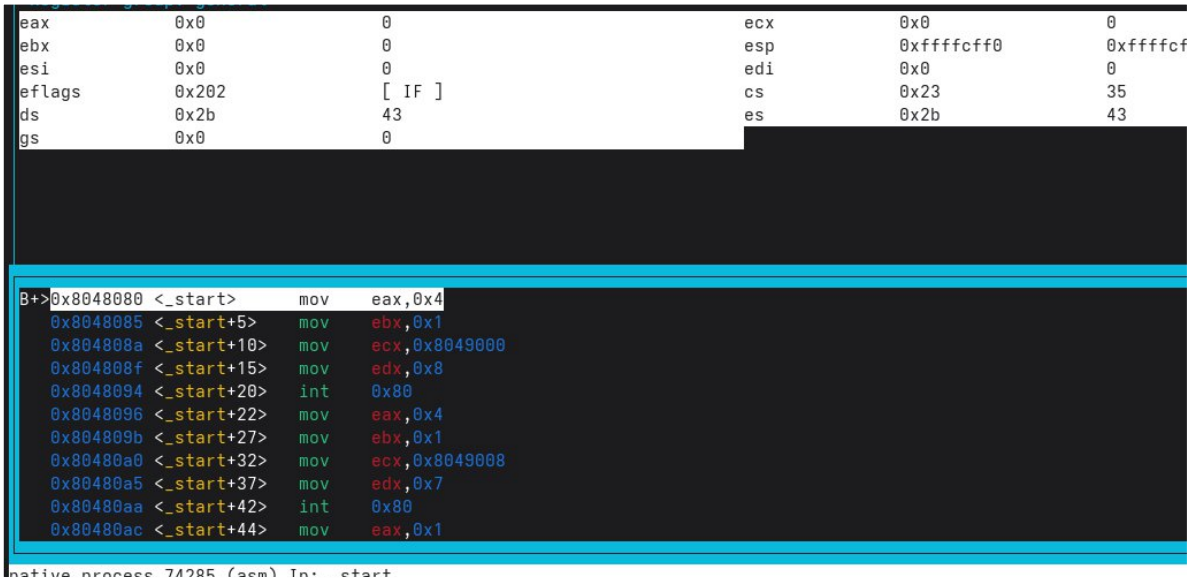

Рисунок 0.24 (layout regs)

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

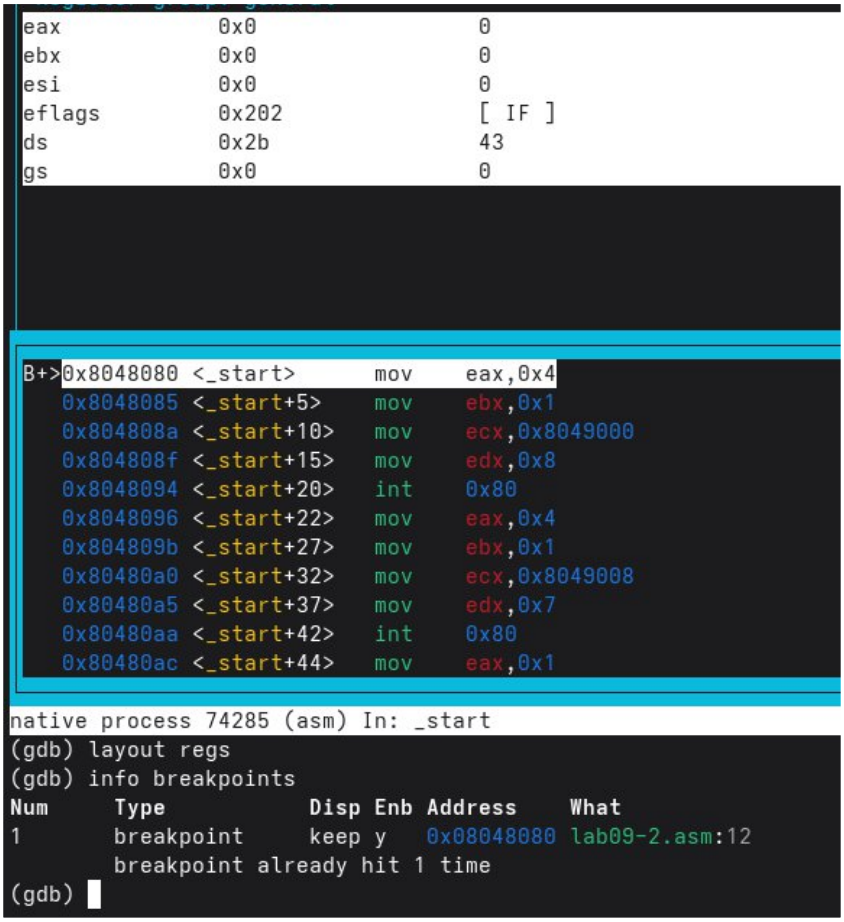

Рисунок 0.25 (info breakpoints)

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.

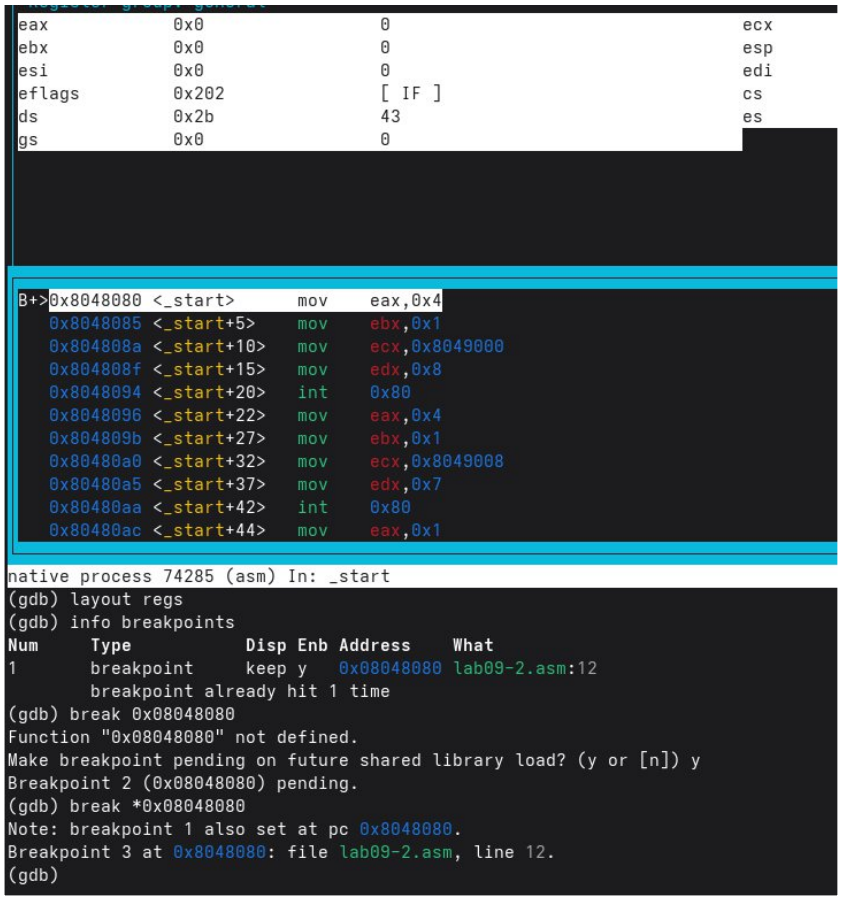

Рисунок 0.26(установка точки останова)

Посмотрим информацию о всех установленных точках останова

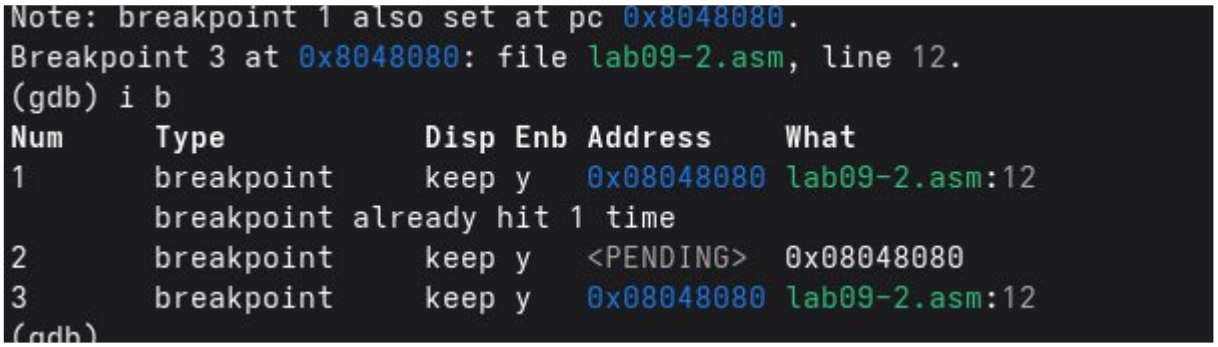

Рисунок 0.27(просмотр информации о установленных точках)

Посмотрим содержимое регистров также можно с помощью команды info registers

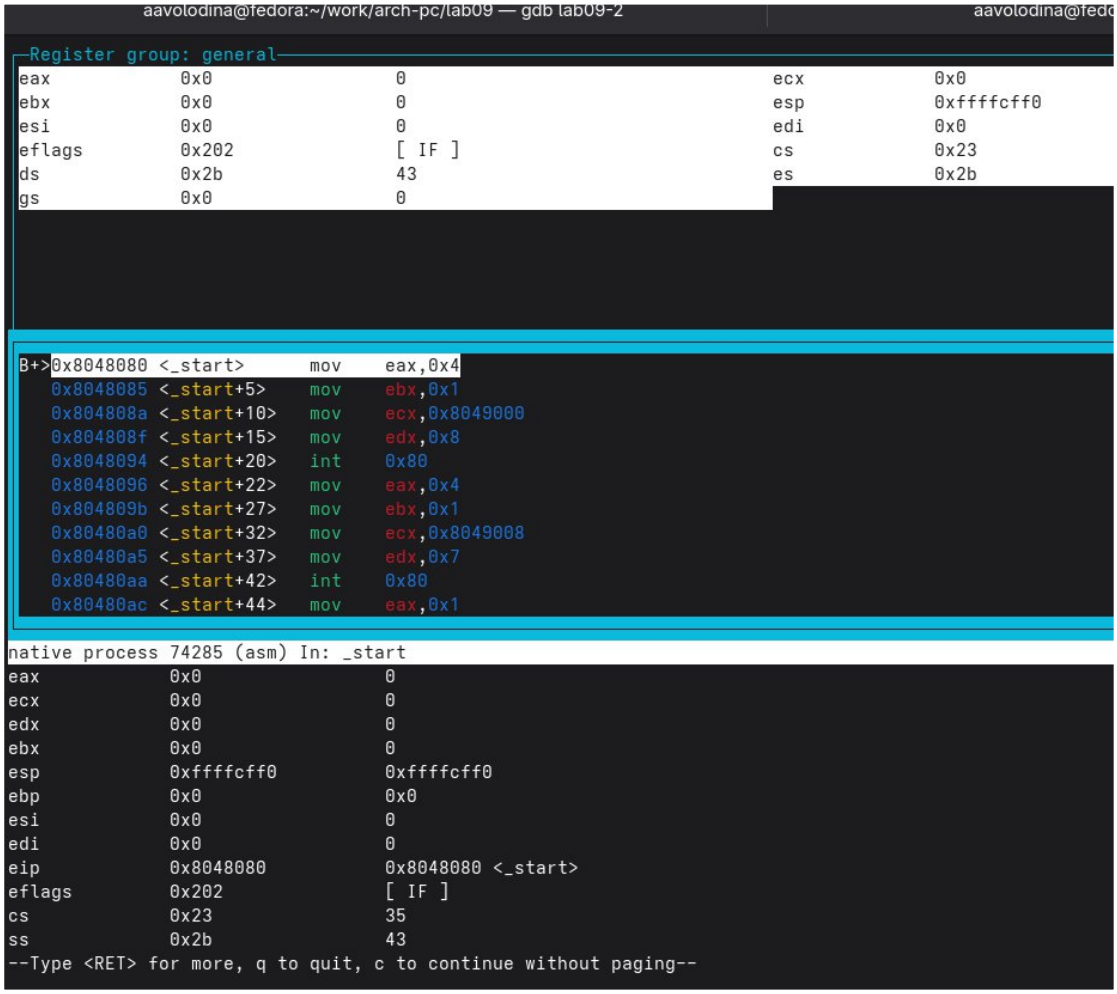

Рисунок 0.28 (просмотр содержимого регистров)

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

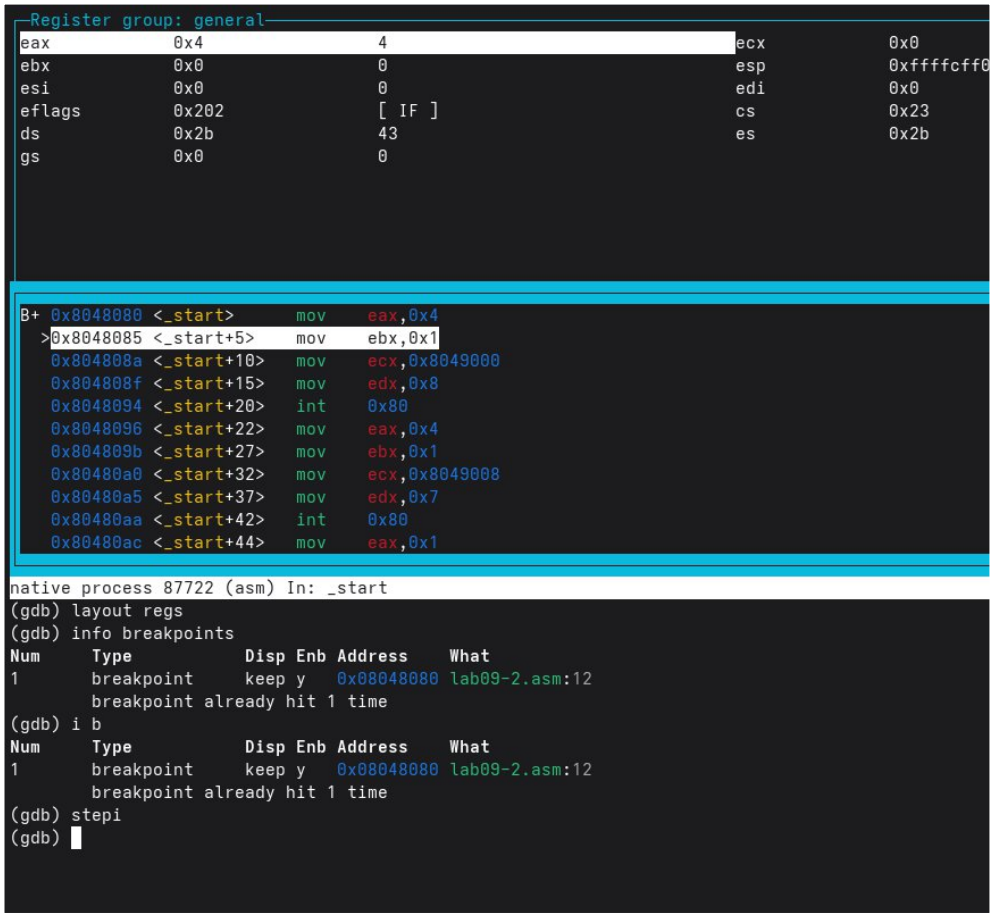

Рисунок 0.29(stepi)

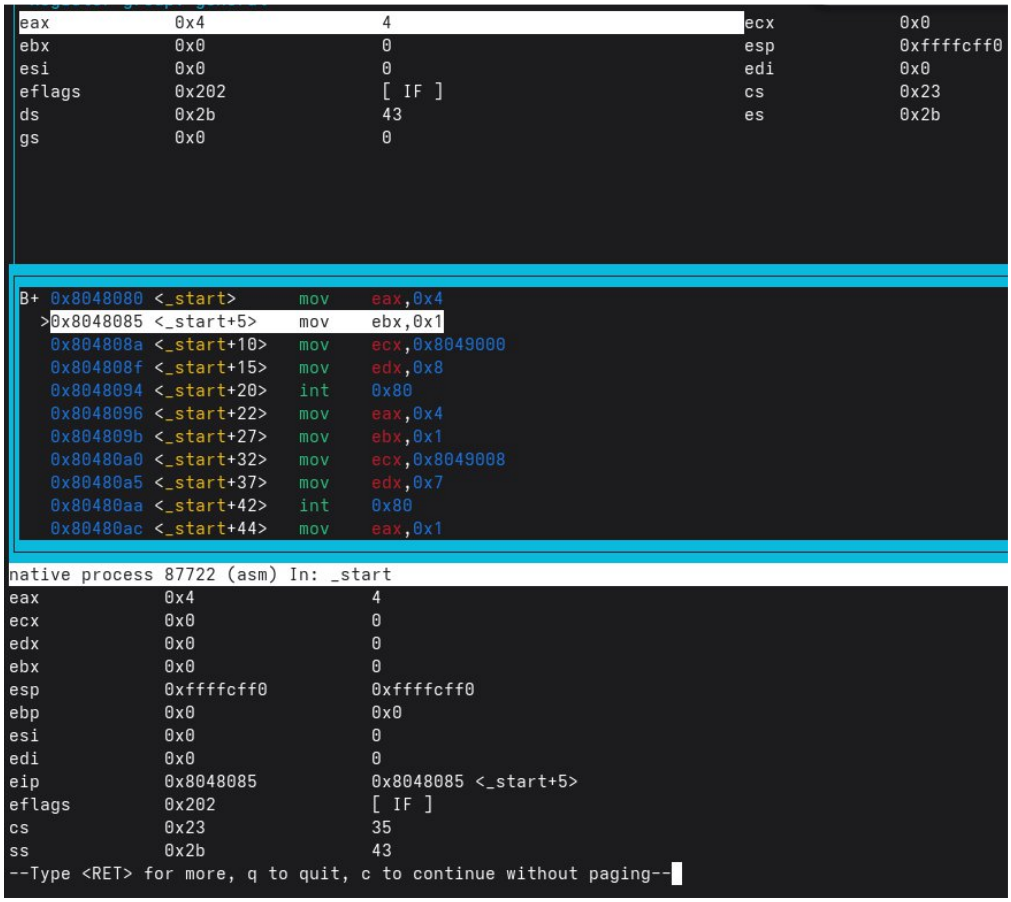

Рисунок 0.30(info registers)

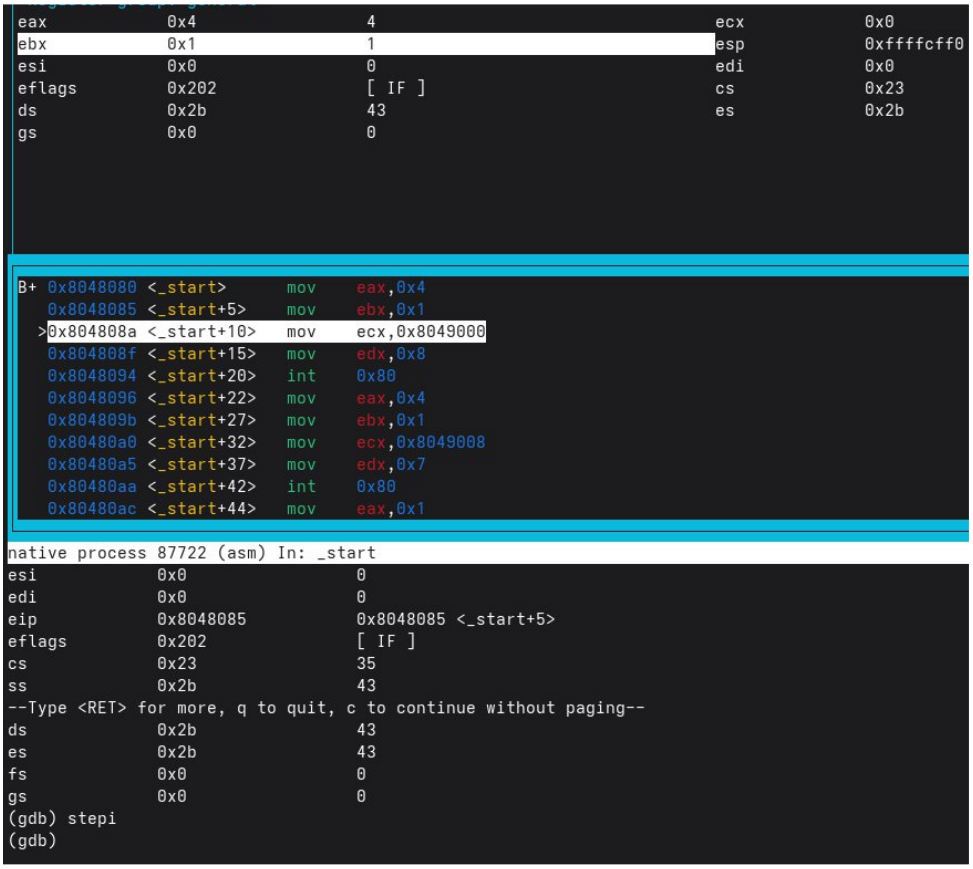

Рисунок 0.31(stepi)

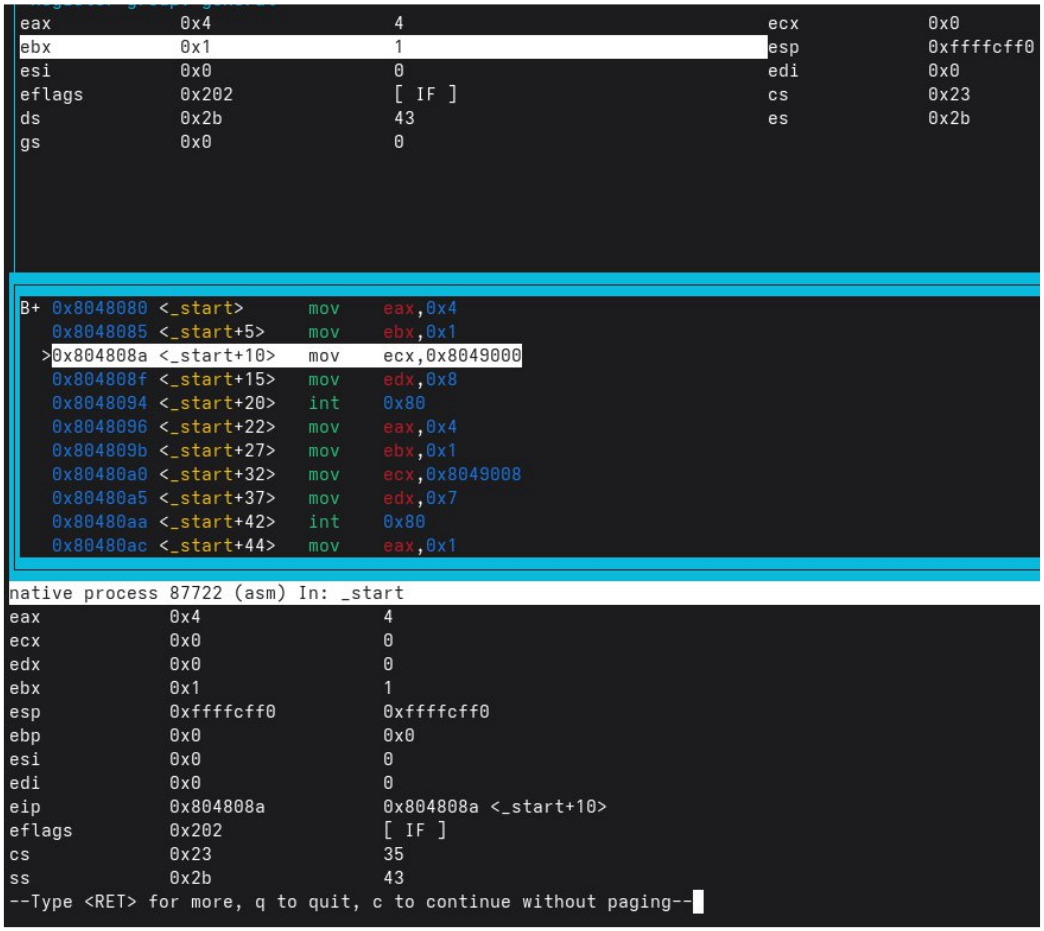

Рисунок 0.32(info registers)

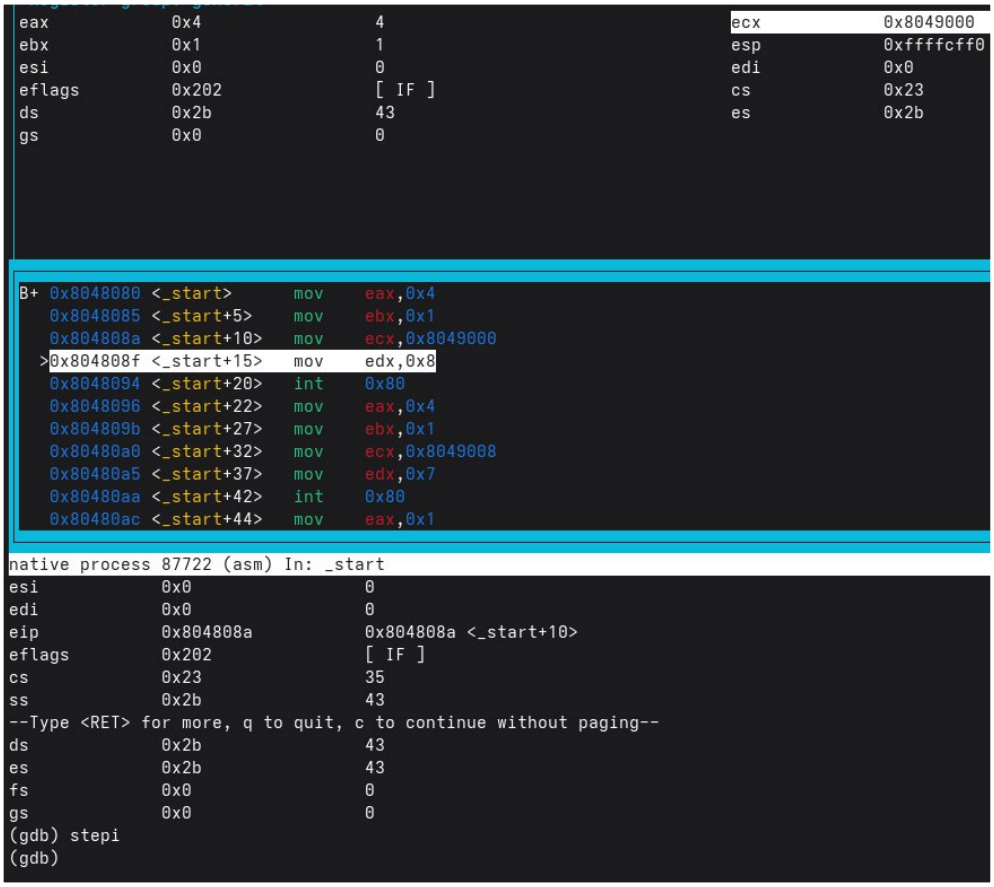

Рисунок 0.33(stepi)

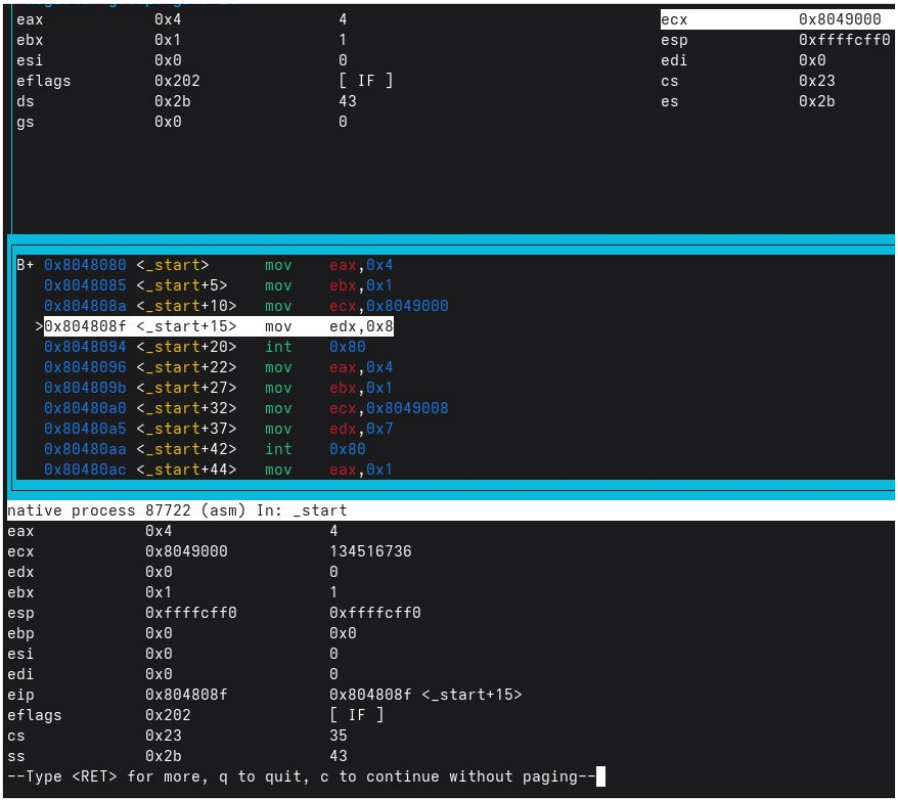

Рисунок 0.34(info registers)

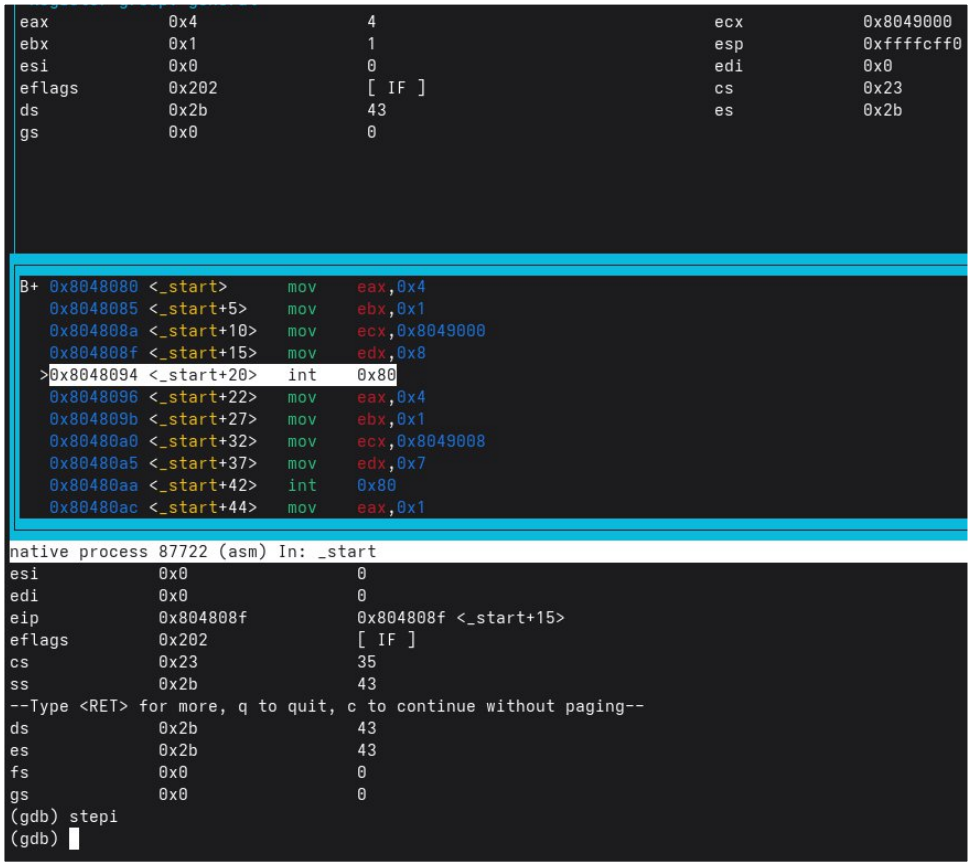

Рисунок 0.35(stepi)

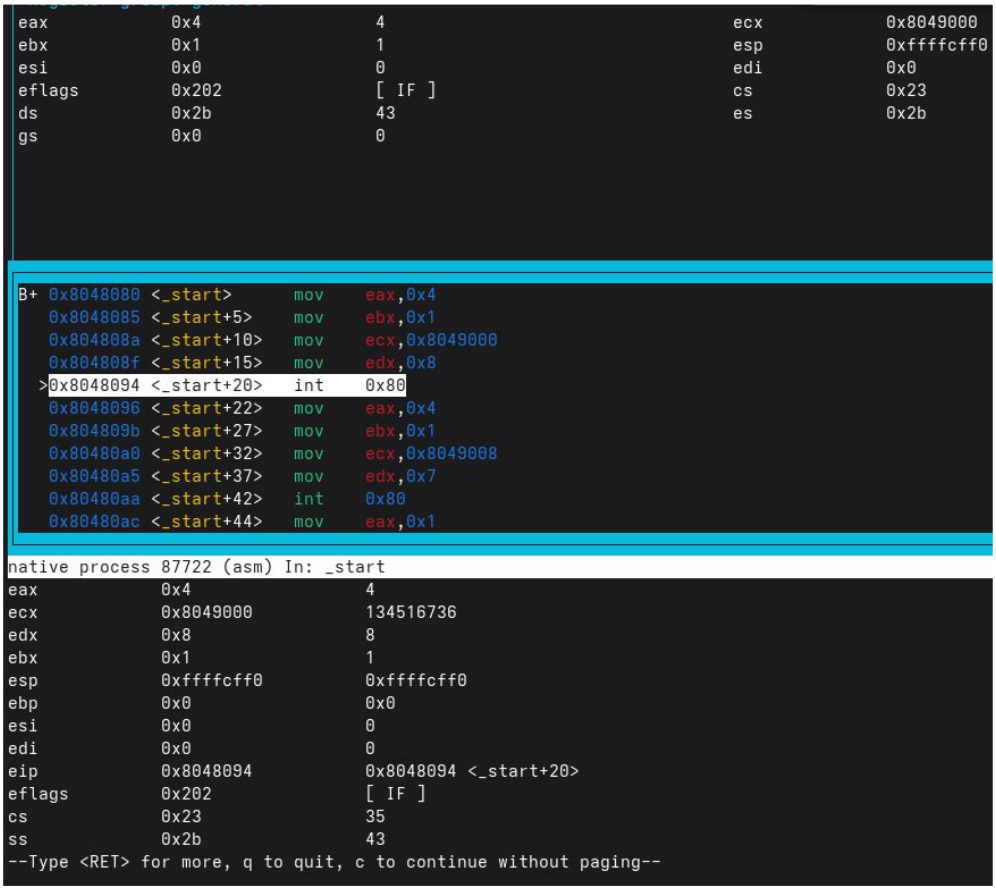

Рисунок 0.36(info registers)

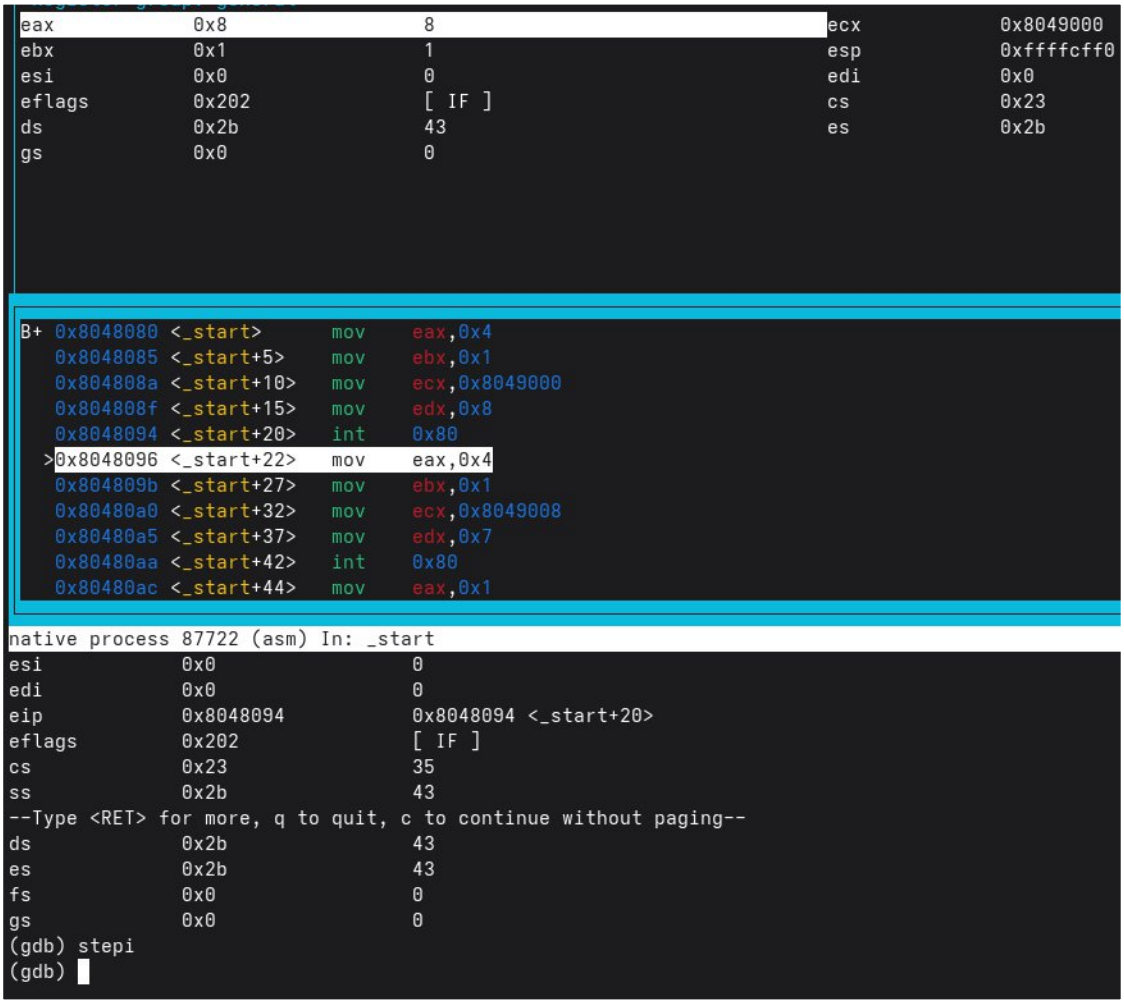

Рисунок 0.37(stepi)

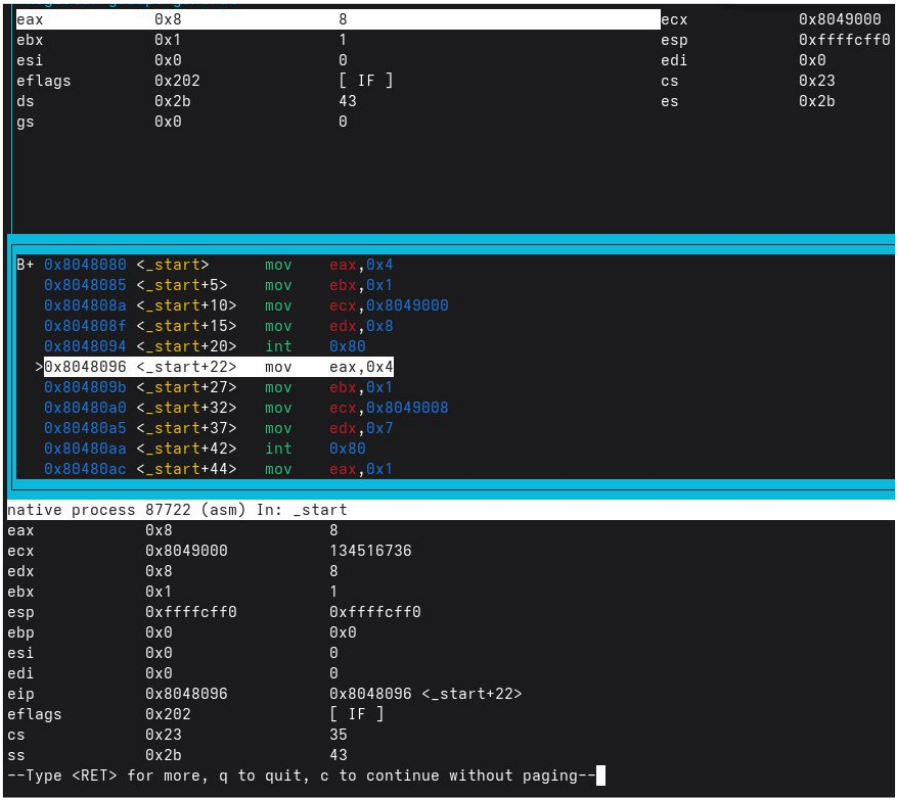

Рисунок 0.38(info registers)

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

```
(gdb) x/1sb &msg1
0x8049000 <msg1>:       "Hello, "
(gdb)
```

Рисунок 0.39(значение переменной msg1)

Посмотрим значение переменной msg2 по адресу

```
(gdb) x/1sb 0x8049008
0x8049008 <msg2>:       "World!\n\034"
(gdb)
```

Рисунок 0.40(значение переменной msg2)

Изменим первый символ переменной msg1

```
native process 74285 (asm) In: _start
0x804a008:      <error: Cannot access memory at address 0x804a008>
(gdb) x/1sb 0x8049000
0x8049000 <msg1>:       "Hello, "
(gdb) x/1sb 0x8049008
0x8049008 <msg2>:       "World!\n\034"
(gdb) set {char}msg1='h'
'msg1' has unknown type; cast it to its declared type
(gdb) set {char}msg1='h'
'msg1' has unknown type; cast it to its declared type
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x8049000 <msg1>:       "hello, "
(gdb)
```

Рисунок 0.41(изменение первого символа msg1)

Заменим любой символ во второй переменной msg2.

```
(gdb) set {char}&msg2='u'
(gdb) x/1sb &mag2
No symbol "mag2" in current context.
(gdb) x/1sb &msg2
0x8049008 <msg2>:       "uorld!\n\034"
(gdb)
```

Рисунок 0.42 (изменение символа msg2)

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

```
native process 76534 (asm) In: _start
(gdb) set $ebx='2'
(gdb) p/s $ebx
$2 = 50
(gdb) p/t $ecx
$3 = 0
$4 = 0
(gdb) p/s $edx
$5 = 0
(gdb) p/t $edx
$6 = 0
(gdb) p/x $edx
$7 = 0x0
(gdb)
```

Рисунок 0.43(значение регистра edx в разных форматах)

С помощью команды set изменим значение регистра ebx

```
native process 76534 (asm) In: _start
(gdb) pNo process (asm) In:
(gdb) p/x $edx
$7 = 0x0
(gdb) set $ebx='2'
(gdb) p/s $ebx
$8 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$9 = 2
```

Рисунок 0.44 (изменение значения регистра ebx)

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

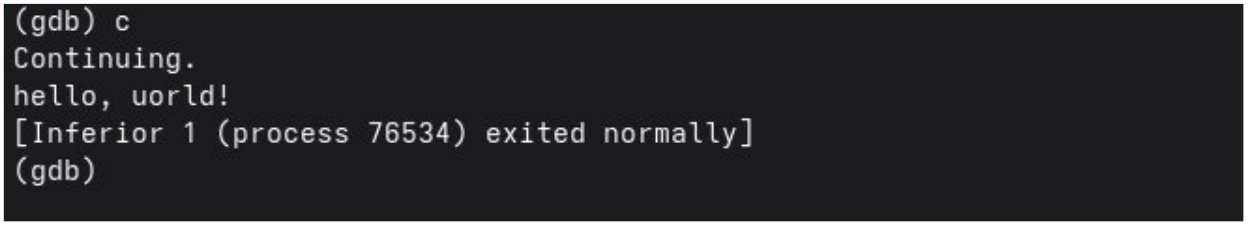

Рисунок 0.45(завершение выполнения программы)

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

Рисунок 0.46(Создание копии файла)

Создадим исполняемый файл

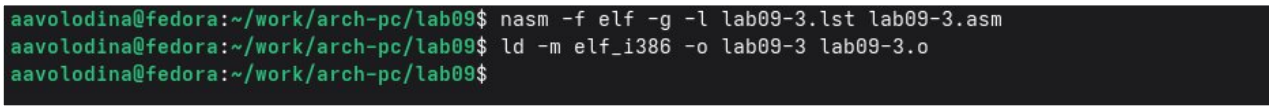

Рисунок 0.47(создание исполняемого файла)

Загрузим исполняемый файл в отладчик, указав аргументы:

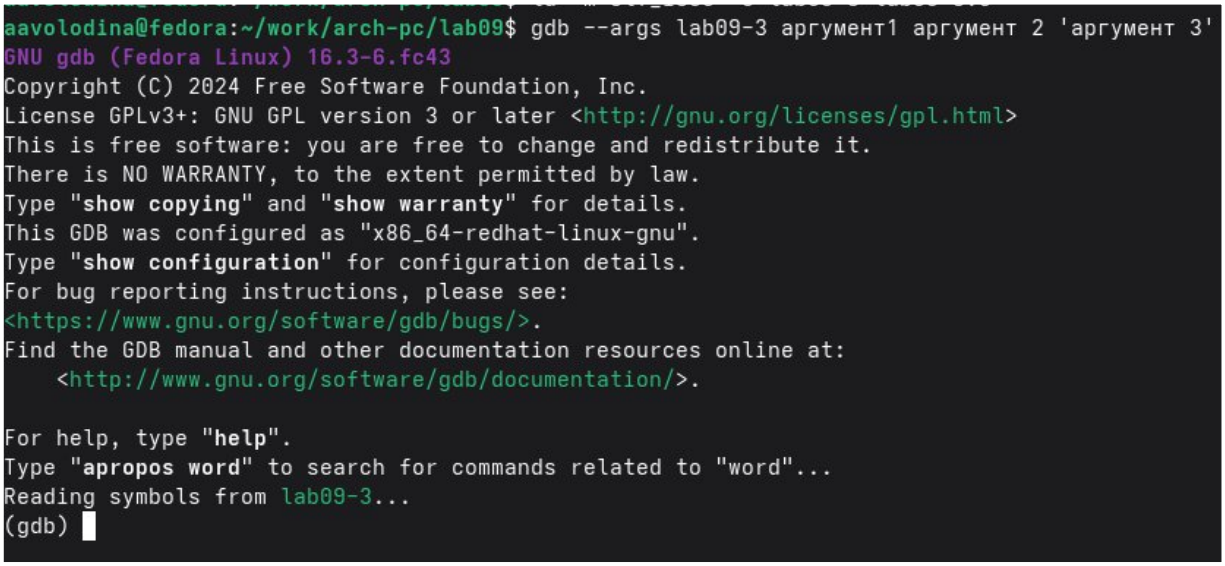

Рисунок 0.48(загрузка исполняемого файла в отладчик)

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

Рисунок 0.49(установка точки перед первой инструкцией)

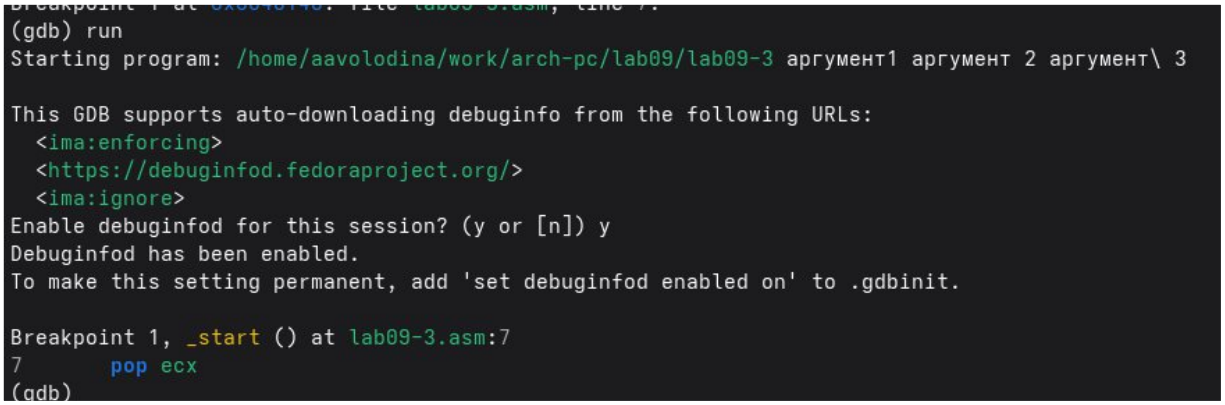

Рисунок 0.50(запуск)

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.

```
        pop ecx
(gdb) x/x $esp
0xffffcfb0:     0x00000005
(gdb) x/s *(void**)($esp + 4)
0xffffd182:     "/home/aavolodina/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd1ae:     "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffd1c0:     "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffd1d1:     "2"
(gdb) x/s *(void**)($esp + 20)
0xffffd1d3:     "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0:    <error: Cannot access memory at address 0x0>
(gdb)
```

Рисунок 0.51(Просмотр остальных позиций стека)

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

Задание для самостоятельной работы

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

```
aavolodina@fedora:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-4.asm ~/work/arch-pc/lab09/lab09-4.asm
aavolodina@fedora:~/work/arch-pc/lab09$
```

Рисунок 0.52 (создание копии файла)

Открыть

```
%include 'in_out.asm'

SECTION .data
msg_func db "Функция: f(x) = 3x - 1", 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, 3
mul ebx
sub eax, 1
ret
```

Рисунок 0.53(текст программы)

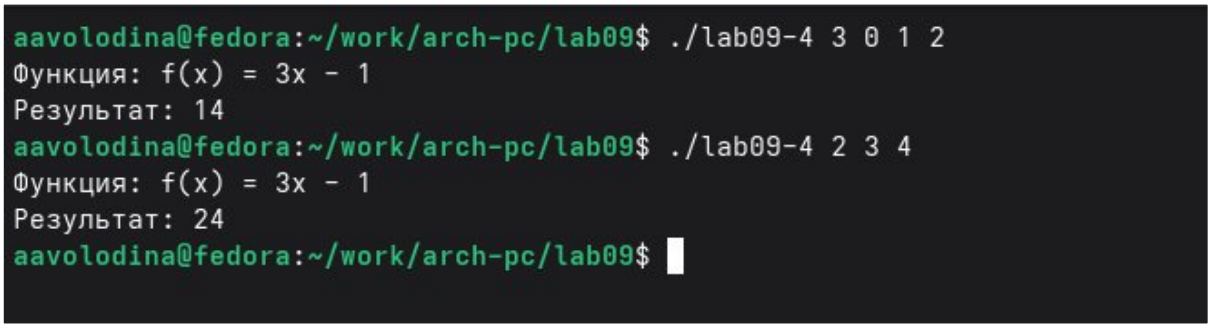

Рисунок 0.54(проверка работы программы)

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

Рисунок 0.55(создание файла)

```
%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 ecx, 4
mul ecx
add ebx, 5
mov edi, eax

mov eax, div
call sprint
mov eax, edi
call iprintLF

call quit
```

Рисунок 0.56(текст программы)

```
aavolodina@fedora:~/work/arch-pc/lab09$ nasm -f elf -g lab09-5.asm -o lab09-5.o
```

Рисунок 0.57(создание исполняемого файла)

```
aavolodina@fedora:~/work/arch-pc/lab09$ gdb lab09-5
GNU gdb (Fedora Linux) 16.3-6.fc43
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-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)
```

Рисунок 0.58(загрузка исполняемого файла в отладчик)

```
(gdb) break _start
Breakpoint 1 at 0x8048168: file lab09-5.asm, line 10.
(gdb)
```

Рисунок 0.59(установка брейкпоинта)

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

This GDB supports auto-downloading debuginfo from the following URLs:
  <ima:enforcing>
  <https://debuginfod.fedoraproject.org/>
  <ima:ignore>
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 lab09-5.asm:10
10      mov ebx, 3
(gdb)
```

Рисунок 0.60(запуск программы)

```
(gdb) print $eax
$1 = 0
(gdb) continue
Continuing.
Результат: 8
[Inferior 1 (process 82215) exited normally]
(gdb)
```

Рисунок 0.61(Результат до исправления кода)

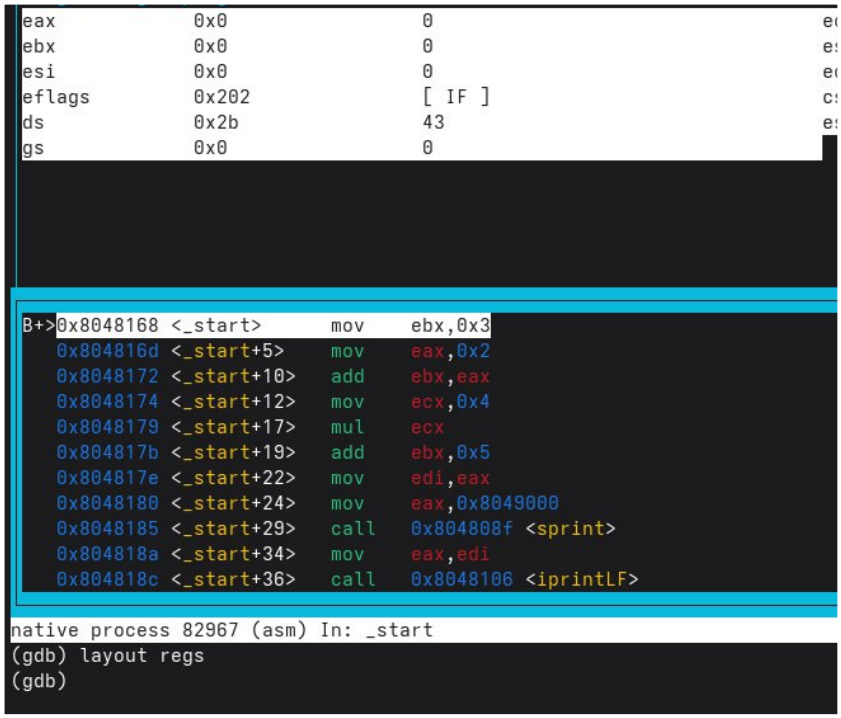

Рисунок 0.62(layout regs)

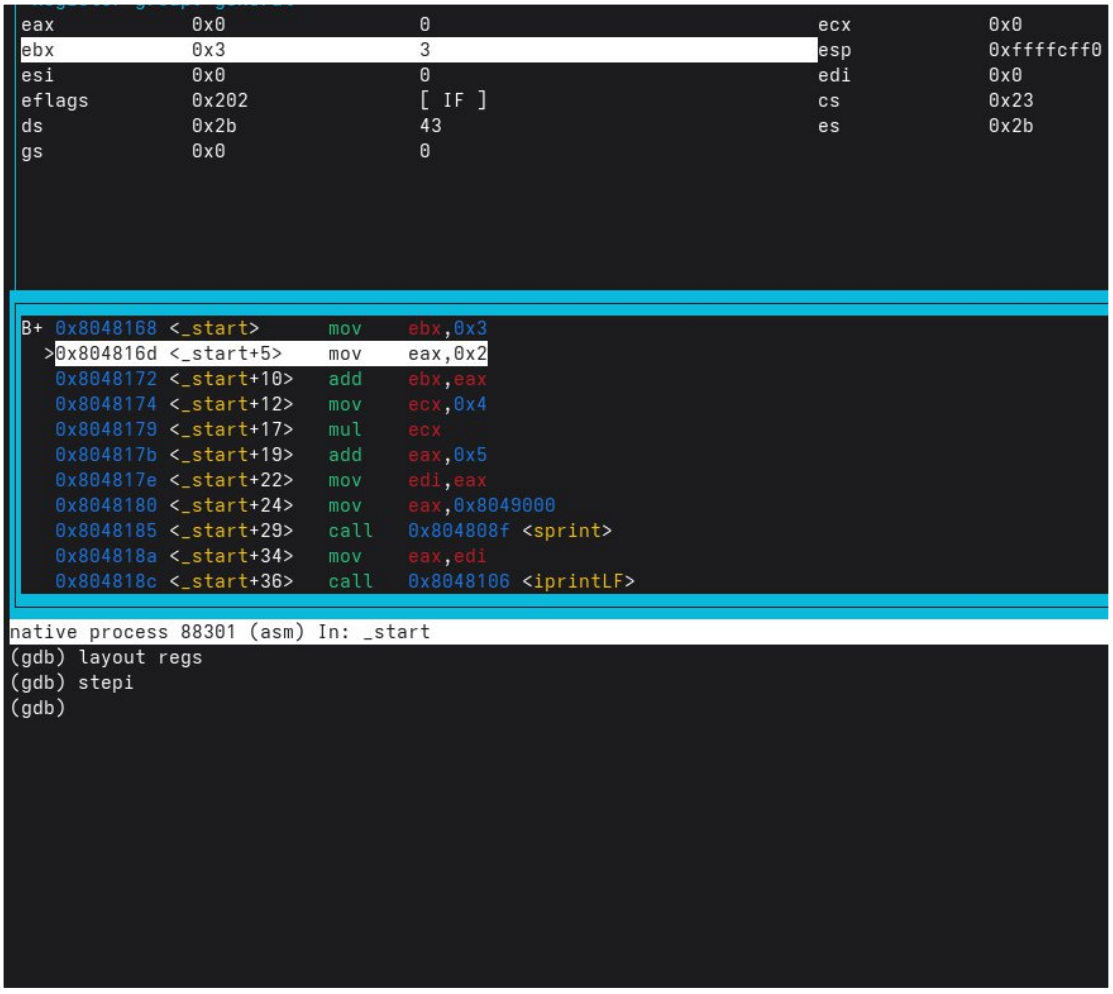

Рисунок 0.63(первый stepi)

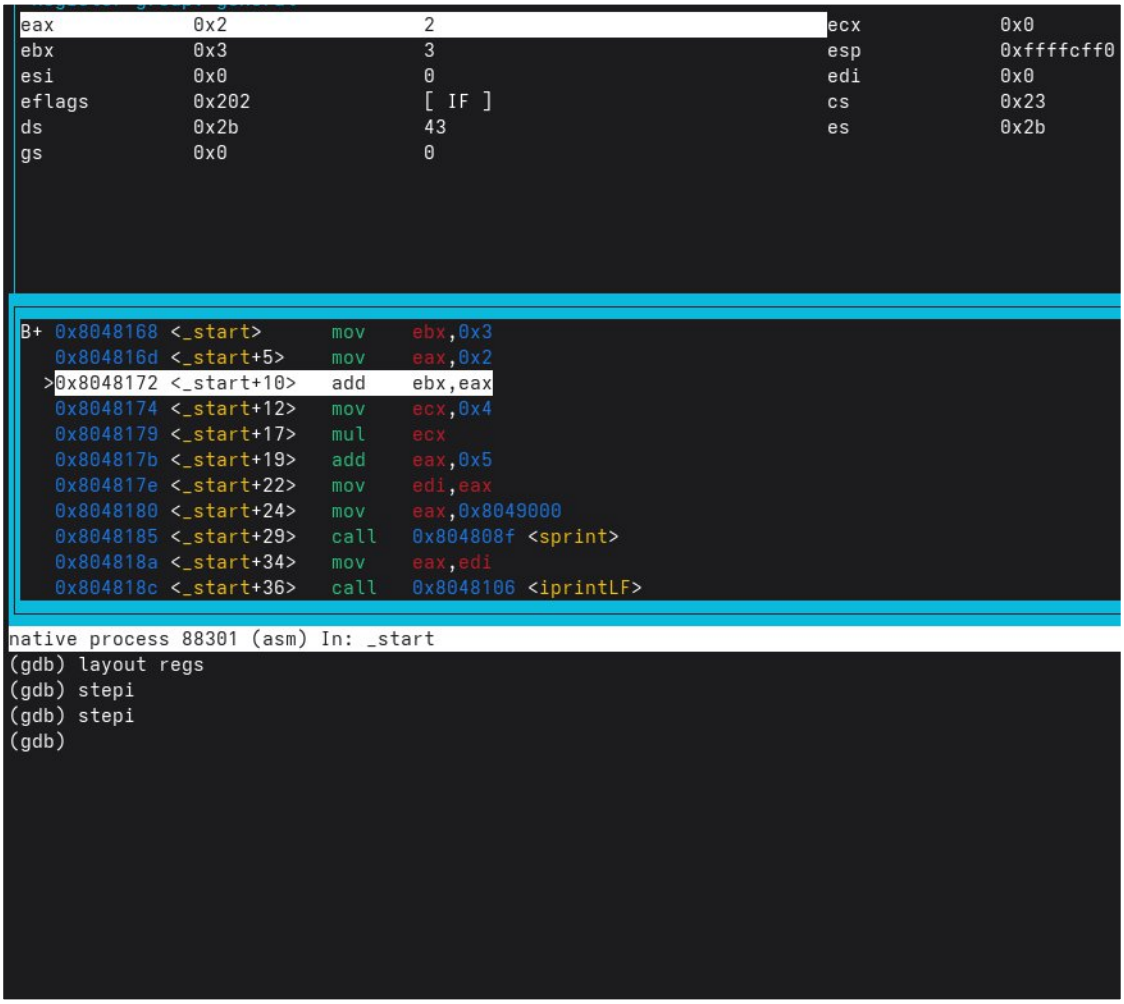

Рисунок 0.64(второй stepi)

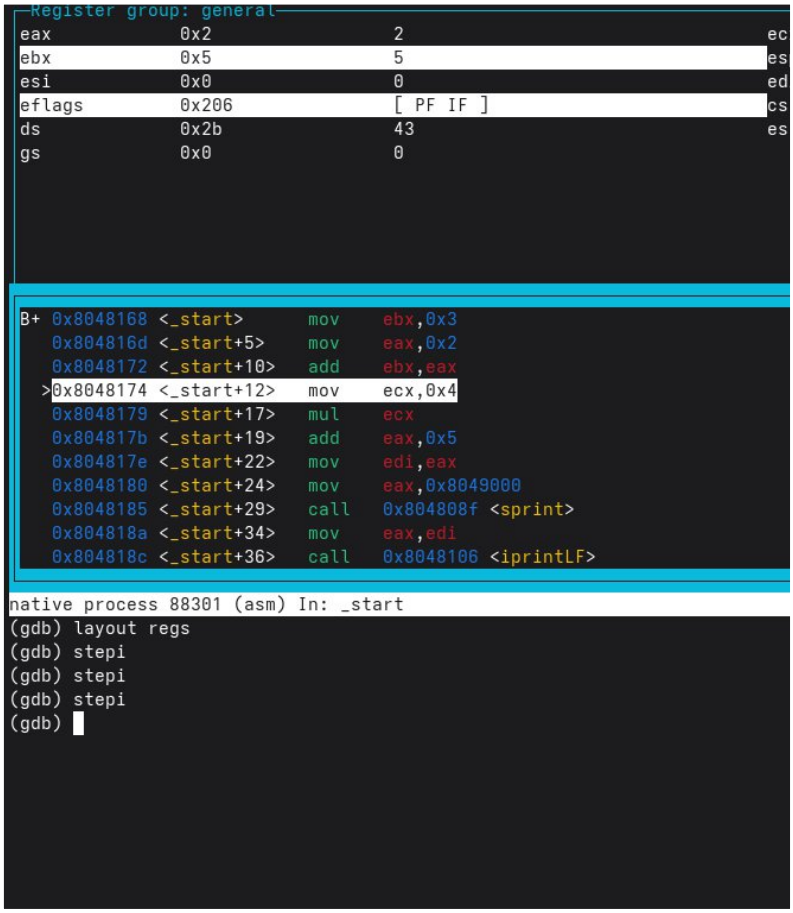

Рисунок 0.65(третий stepi)

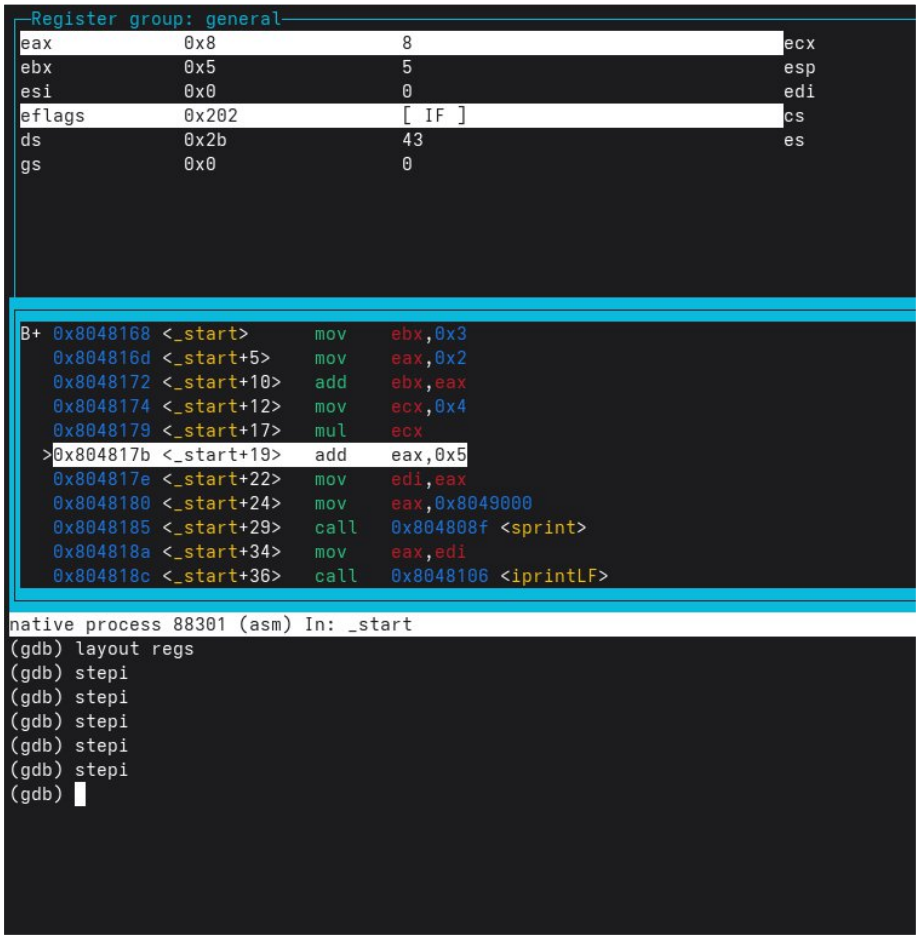

Рисунок 0.66(stepi находим ошибку)

```
%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
```

Рисунок 0.67(исправленный код)

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

Рисунок 0.68(результат после исправления)

5 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://midnight-commander. 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.

9. Колдаев В. Д.,Лупин С. А.Архитектура ЭВМ.—М. : Форум,2018.

10. Куляс О. Л.,Никитин К. А. Курс программирования наASSEMBLER.—М. : Солон-Пресс, 2017.

11. НовожиловО.П.Архитектура ЭВМисистем.—М.:Юрайт,2016.

12. Расширенныйассемблер:NASM.—2021.—URL:https://www.opennet.ru/docs/RUS/nasm/.

13. Робачевский А.,Немнюгин С.,Стесик О. Операционная система UNIX.—2-е изд.—БХВ Петербург, 2010.—656 с.—ISBN 978-5-94157-538-1.

14. СтоляровА.Программированиенаязыкеассемблера NASMдляОСUnix.—2-еизд.— М. : МАКСПресс,2011.—URL:http://www.stolyarov.info/books/asm_unix.

15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).

16. Таненбаум Э.,Бос Х. Современные операционные системы.—4-е изд.—СПб.: Питер, 2015. —1120 с.—(Классика Computer Science)