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: Тимаков Макарий Владимирович

Группа: НПИбд 02-25

МОСКВА

2025г.

1.Цель работы

Познакомиться с методами отладки при помощи GDB, его возможностями.

2. Выполнение лабораторной работы

2.1 Реализация подпрограмм в NASM

Создаем каталог для программ ЛБ9, и в нем создаем файл (рис. Рисунок 2.1).

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

Рисунок 2.1: Создаем каталог с помощью команды mkdir и файл с помощью команды touch

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.1 (рис. Рисунок 2.2).

```
%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:
;-----------------------------------------
; Основная программа
;-----------------------------------------
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
;-----------------------------------------
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
mov ebx,2
```

1 Помощь 2 Разверн 3 Выход 4 Hex 5 Перейти 6 7 Поиск 8 Исходный 9 Формат 10 Выход

Рисунок 2.2: Заполняем файл

Создаем исполняемый файл и запускаем его (рис. Рисунок 2.3).

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

Рисунок 2.3: Запускаем файл и проверяем его работу

Снова открываем файл для редактирования и изменяем его, добавив подпрограмму в подпрограмму(по условию) (рис. Рисунок 2.4).

```
%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:
;---------------------------------------
; Основная программа
;---------------------------------------
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
;---------------------------------------
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
call _subcalcul
mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret
_subcalcul:
mov ebx,3
mul ebx
sub eax,1
ret
```


Рисунок 2.4: Изменяем файл, добавляя еще одну подпрограмму

Создаем исполняемый файл и запускаем его (рис. Рисунок 2.5)

```
timakovmv@timakovmv:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
timakovmv@timakovmv:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
timakovmv@timakovmv:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 4
2(3x-1)+7=29
timakovmv@timakovmv:~/work/arch-pc/lab09$
```

Рисунок 2.5: Запускаем файл и смотрим на его работу

Создаем новый файл в каталоге(рис. Рисунок 2.6).

```
2(3x-1)+7=29
timakovmv@timakovmv:~/work/arch-pc/lab09$ touch lab09-2.asm
timakovmv@timakovmv:~/work/arch-pc/lab09$
```

Рисунок 2.6: Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.2 (рис. Рисунок 2.7).

```
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
```

Рисунок 2.7: Заполняем файл

Получаем исходный файл с использованием отладчика gdb (рис. Рисунок 2.8)

```
timakovmv@timakovmv:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
timakovmv@timakovmv:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
timakovmv@timakovmv:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 16.2-3.fc42
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"...
--Type <RET> for more, q to quit, c to continue without paging--
Reading symbols from lab09-2...
(gdb)
```

Рисунок 2.8: Загружаем исходный файл в отладчик

Запускаем команду в отладчике (рис. Рисунок 2.9).

```
(gdb) run
Starting program: /home/timakovmv/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 12284) exited normally]
(gdb)
```

Рисунок 2.9: Запускаем программу командой run

Устанавливаем брейкпоинт на метку _start и запускаем программу (рис. Рисунок 2.10).

```
(gdb) break _start
Breakpoint 1 at 0x8048080: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/timakovmv/work/arch-pc/lab09/lab09-2

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

Рисунок 2.10: Запускаем программу с брейкпоином

Смотрим дисассимилированный код программы с помощью команды disassemble, начиная с метки _start(рис. Рисунок 2.11)

```
(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 2.11: Смотрим дисассимилированный код программы

Переключаемся на отображение команд с Intel'овским синтаксисом (рис. Рисунок 2.12).

3.

```
(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 2.12: Переключаемся на синтаксис Intel

Различия отображения синтаксиса машинных команд в режимах ATT и Intel:

1. Порядок операндов: В ATT синтаксисе порядок операндов обратный, сначала указывается исходный операнд, а затем - результирующий операнд. В Intel синтаксисе порядок обычно прямой, результирующий операнд указывается первым, а исходный — вторым.
2. Разделители: В ATT синтаксисе разделители операндов - запятые. В Intel синтаксисе разделители могут быть запятые или косые черты (/).
3. Префиксы размера операндов: В ATT синтаксисе размер операнда указывается перед операндом с использованием префиксов, таких как «b» (byte), «w» (word), «l» (long) и «q» (quadword). В Intel синтаксисе размер операнда указывается после операнда с использованием суффиксов, таких как «b», «w», «d» и «q».
4. Знак операндов: В ATT синтаксисе операнды с позитивными значениями предваряются символом «".*Intel*"».
5. Обозначение адресов: В ATT синтаксисе адреса указываются в круглых скобках. В Intel синтаксисе адреса указываются без скобок.
6. Обозначение регистров: В ATT синтаксисе обозначение регистра начинается с символа «%». В Intel синтаксисе обозначение регистра может начинаться с символа «R» или «E» (например, «%eax» или «RAX»).

Включаем режим псевдографики (рис. Рисунок 2.13).

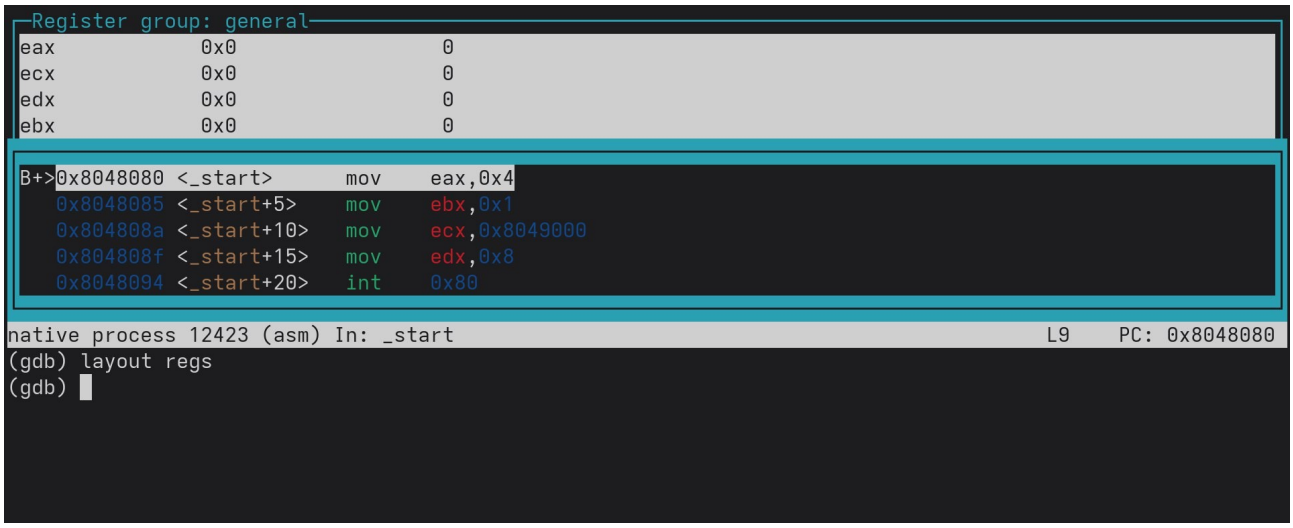

Рисунок 2.13: Включаем отображение регистров, их значений и результат дисассимилирования программы

Проверяем была ли установлена точка останова и устанавливаем точку останова предпоследней инструкции (рис. Рисунок 2.14).

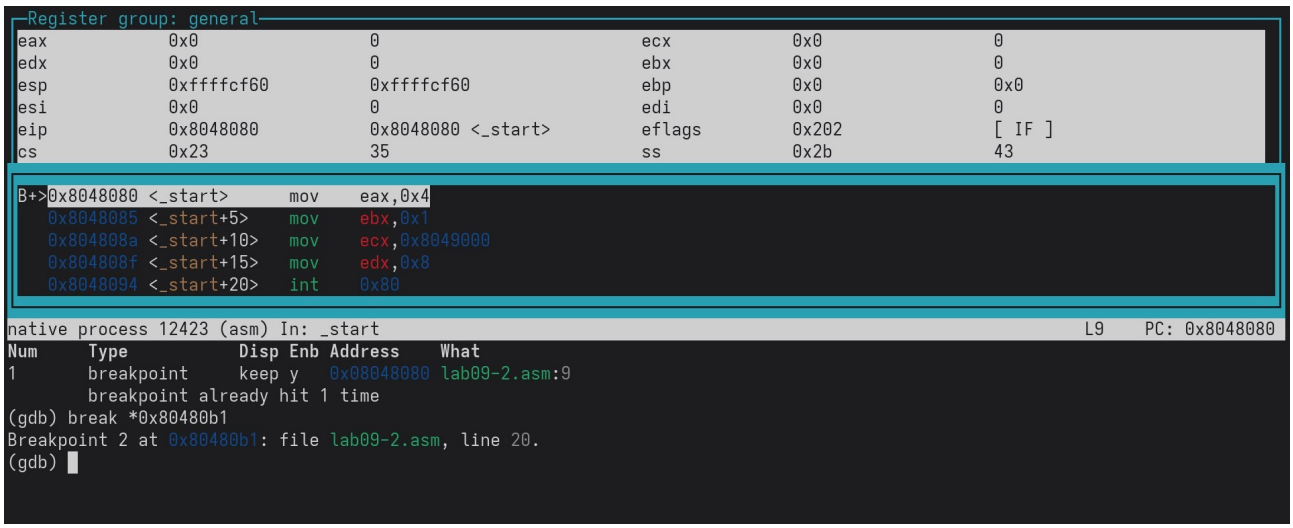

Рисунок 2.14: Используем команду info breakpoints и создаем новую точку останова

Посмотрим информацию о всех установленных точках останова (рис. Рисунок 2.15).

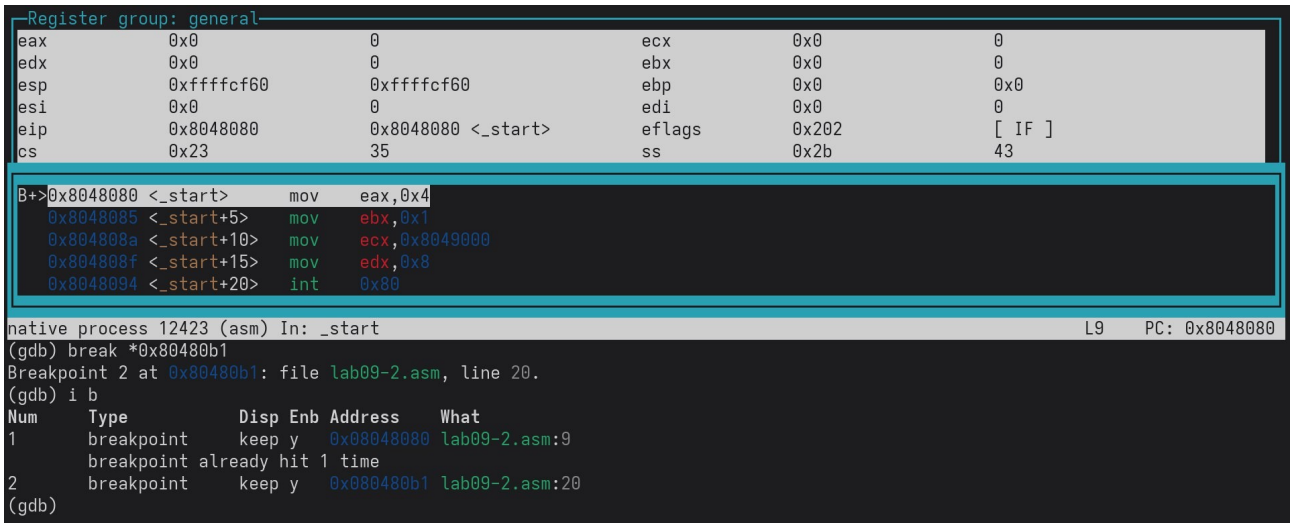

Рисунок 2.15: Смотрим информацию

Выполняем 5 инструкций командой si (рис. Рисунок 2.16)

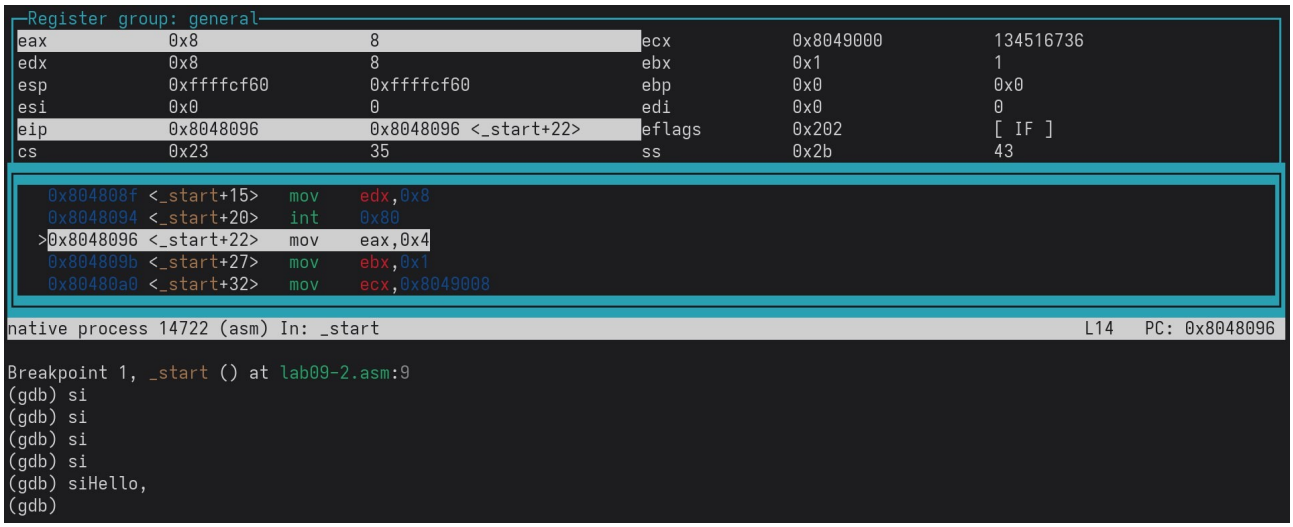

Рисунок 2.16: Отслеживаем регистры

Во время выполнения команд менялись регистры: ebx, ecx, edx,eax, eip. Смотрим значение переменной msg1 по имени (рис. Рисунок 2.17).

Рисунок 2.17: Смотрим значение переменной

Смотрим значение переменной msg2 по адресу (рис. Рисунок 2.18)

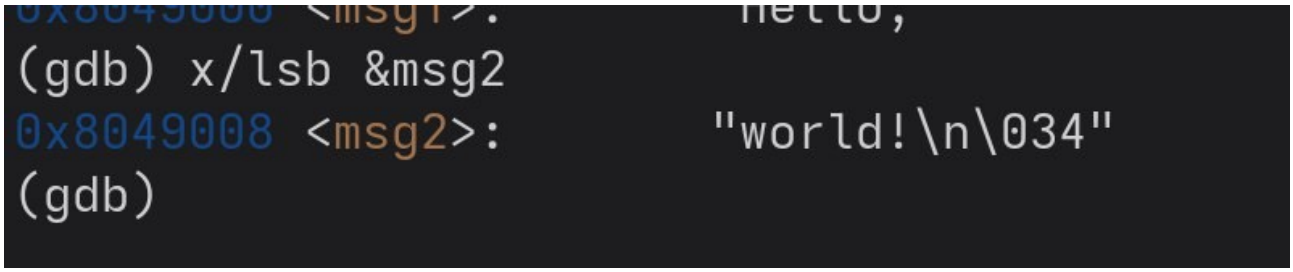

Рисунок 2.18: Смотрим значение переменной

Изменим первый символ переменной msg1 (рис. Рисунок 2.19).

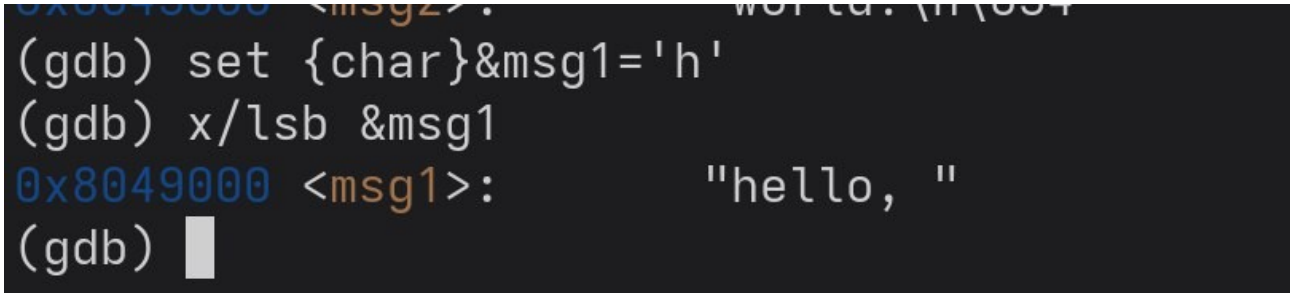

Рисунок 2.19: Меняем символ

Изменим первый символ переменной msg2 (рис. Рисунок 2.20).

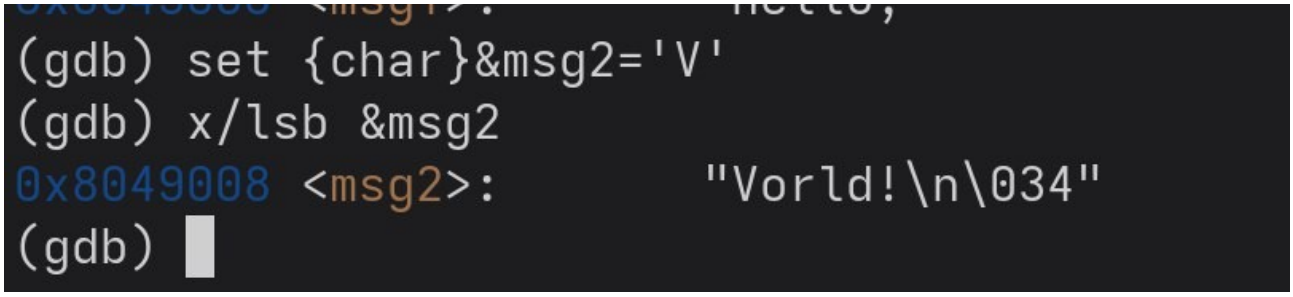

Рисунок 2.20: Меняем символ

Смотрим значение регистра edx в разных форматах (рис. Рисунок 2.21).

```
(gdb) p/s $edx
$2 = 8
(gdb) p/x $edx
$3 = 0x8
(gdb) p/t $edx
$4 = 1000
(gdb)
```

Рисунок 2.21: Смотрим значение регистра

Изменяем регистор ebx (рис. Рисунок 2.22).

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$5 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$6 = 2
(gdb)
```

Рисунок 2.22: Изменяем регистор командой set

Выводится разные значения, так как команда без кеавычек присваивает регистру вводимое значение.

Прописываем команды для завершения программы и выхода из GDB (рис. Рисунок 2.23).

Рисунок 2.23: Прописываем команды c и quit

Копируем файл lab8-2.asm в файл с именем lab09-3.asm (рис. Рисунок 2.24).

Рисунок 2.24: Копируем файл

Создаем исполняемый файл и запускаем его в отладчике GDB (рис. Рисунок 2.25).

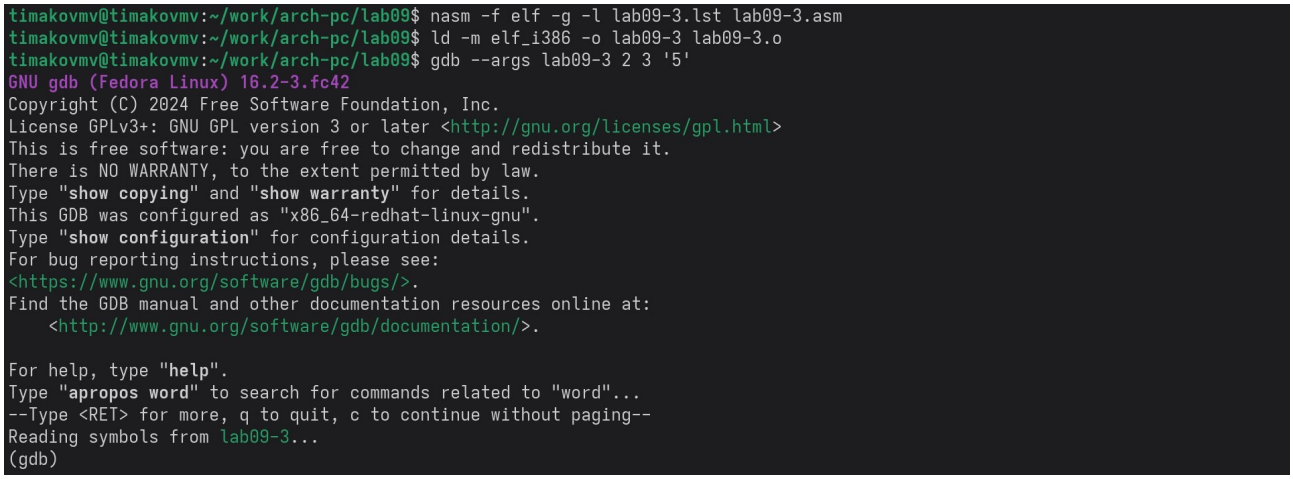

Рисунок 2.25: Создаем и запускаем в отладчике файл

Установим точку останова перед первой инструкцией в программе и запустим ее (рис. Рисунок 2.26).

```
(gdb) b _start
Breakpoint 1 at 0x8048148: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/timakovmv/work/arch-pc/lab09/lab09-3 2 3 5

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:5
5       pop ecx ; Извлекаем из стека в ecx количество
(gdb) x/x $esp
0xffffcf50:     0x00000004
(gdb)
```

Рисунок 2.26: Устанавливаем точку останова

Смотрим позиции стека по разным адресам (рис. Рисунок 2.27).

```
(gdb) x/s *(void**)($esp + 4)
0xffffd139:     "/home/timakovmv/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)

0xffffd164:     "2"
(gdb) x/s *(void**)($esp + 12)

0xffffd166:     "3"
(gdb) x/s *(void**)($esp + 20)

0x0:    <error: Cannot access memory at address 0x0>
(gdb) x/s *(void**)($esp + 16)

0xffffd168:     "5"
(gdb)
```

Рисунок 2.27: Изучаем полученные данные

Шаг изменения адреса равен 4 потому что адресные регистры имеют размерность 32 бита(4 байта).

3.Задание для самостоятельной работы

ЗАДАНИЕ 1

Копируем файл lab8-4.asm(ср №1 в ЛБ8) в файл с именем lab09-4.asm (рис. Рисунок 3.1).

```
timakovmv@timakovmv:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-4.asm ~/work/ar
ch-pc/lab09/lab09-4.asm
timakovmv@timakovmv:~/work/arch-pc/lab09$
```

Рисунок 3.1: Копируем файл

Открываем файл в Midnight Commander и меняем его, создавая подпрограмму (рис. Рисунок 3.2)

```
/home/timakovmv/work/arch-pc/lab09/lab09-4.asm
%include "in_out.asm"

SECTION .data
msg db "Результат: ",0

SECTION .bss
prm: resb 80

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
    add esi, eax
    loop next

_end:
    mov eax, msg
    call sprint
    mov eax, esi
    call iprintLF
    call quit

_calcul:
    mov ebx, 8
    mul ebx
1Помощь  2Разверн  3Выход  4Hex
```

Рисунок 3.2: Изменяем файл

Создаем исполняемый файл и запускаем его (рис. Рисунок 3.3)

```
timakovmv@timakovmv:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
timakovmv@timakovmv:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
timakovmv@timakovmv:~/work/arch-pc/lab09$ ./lab09-4 7 4 2
Результат: 95
timakovmv@timakovmv:~/work/arch-pc/lab09$
```

Рисунок 3.3: Проверяем работу программы

ЗАДАНИЕ 2

Создаем новый файл в дирректории (рис. Рисунок 3.4).

```
timakovmv@timakovmv:~/work/arch-pc/lab09$ touch lab09-5.asm
timakovmv@timakovmv:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
```

Рисунок 3.4: Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.3 (рис. Рисунок 3.5).

```
/home/timakovmv/work/arch-pc/lab09/lab09-5.asm                    348/348  100%
%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 ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
1Помощь 2Разверн 3Выход 4Hex 5Перейти 6  7Поиск 8Исходный 9Формат 10Выход
```

Рисунок 3.5: Заполняем файл

Создаем исполняемый файл и запускаем его (рис. Рисунок 3.6)

```
timakovmv@timakovmv:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
timakovmv@timakovmv:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab09-5.o
timakovmv@timakovmv:~/work/arch-pc/lab09$ ./lab09-5
Результат: 10
timakovmv@timakovmv:~/work/arch-pc/lab09$
```

Рисунок 3.6: Создаем и смотрим на работу программы(работает неправильно)

Создаем исполняемый файл и запускаем его в отладчике GDB и смотрим на изменение решистров командой si (рис. Рисунок 3.7).

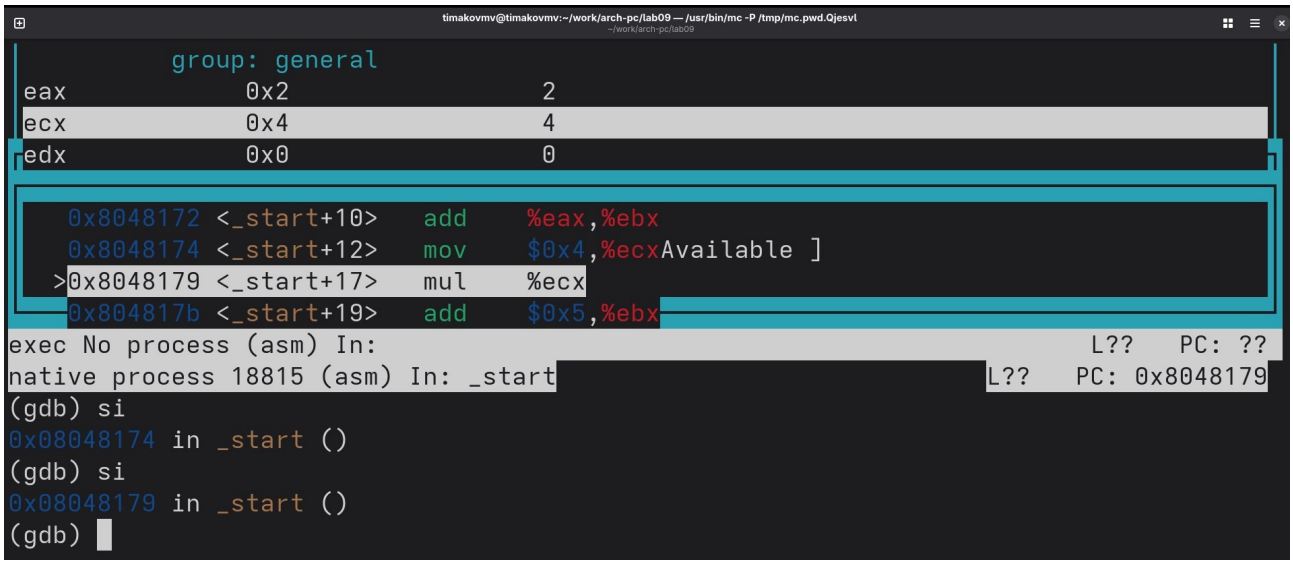

Рисунок 3.7: Ищем ошибку регистров в отладчике

Изменяем программу для корректной работы (рис. Рисунок 3.8)

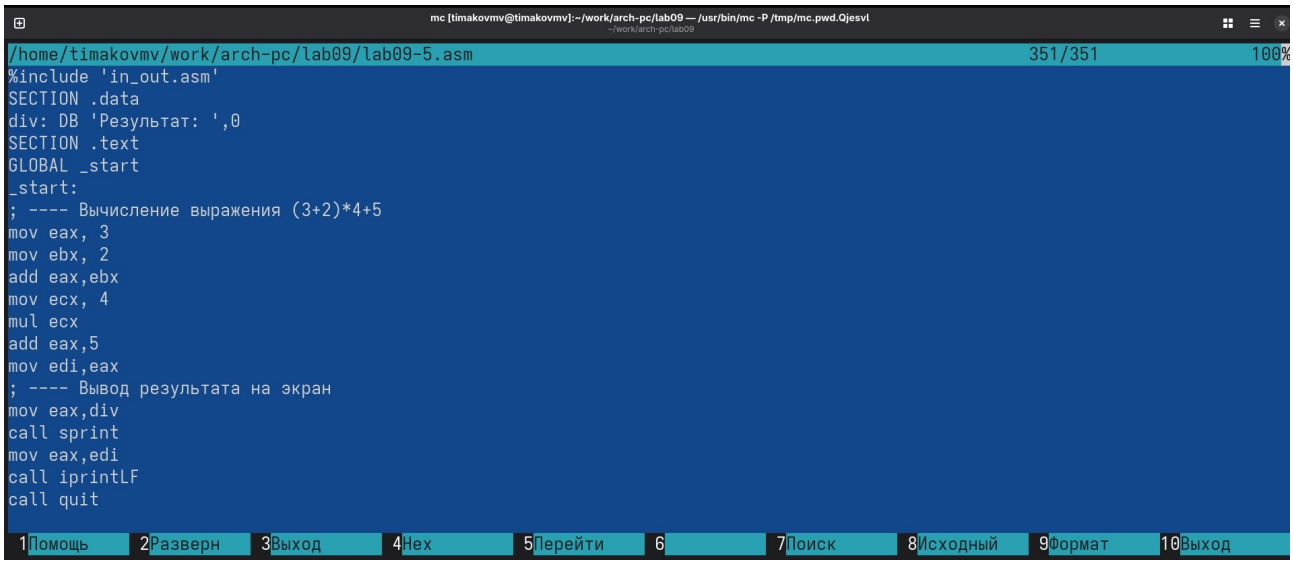

Рисунок 3.8: Меняем файл

Создаем исполняемый файл и запускаем его (рис. Рисунок 3.9)

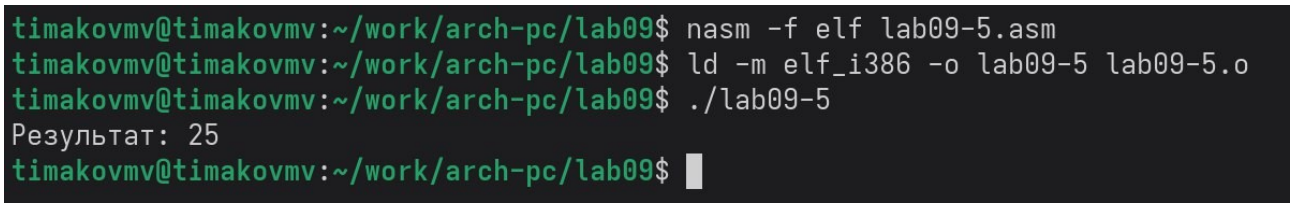

Рисунок 3.9: Создаем и запускаем файл(работает корректно)

4.Выводы

Мы познакомились с методами отладки при помощи GDB и его возможностями.