

Отчёта по лабораторной работе №9

Понятие подпрограммы. Отладчик GDB.

Ромицына Анастасия Романовна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Реализация подпрограмм в NASM	6
2.2	Отладка программ с помощью GDB	8
3	Выводы	21

Список иллюстраций

2.1	Создаем каталог с помощью команды <code>mkdir</code> и файл с помощью команды <code>touch</code>	6
2.2	Заполняем файл	7
2.3	Запускаем файл и проверяем его работу	7
2.4	Изменяем файл, добавляя еще одну подпрограмму	8
2.5	Запускаем файл и смотрим на его работу	8
2.6	Создаем файл	8
2.7	Заполняем файл	9
2.8	Загружаем исходный файл в отладчик	9
2.9	Запускаем программу командой <code>run</code>	10
2.10	Запускаем программу с брейкпоинтом	10
2.11	Смотрим дисассимилированный код программы	10
2.12	Переключаемся на синтаксис Intel	11
2.13	Включаем отображение регистров, их значений и результат дисассимилирования программы	11
2.14	Используем команду <code>info breakpoints</code> и создаем новую точку останова	12
2.15	Смотрим информацию	12
2.16	Отслеживаем регистры	13
2.17	Смотрим значение переменной	13
2.18	Смотрим значение переменной	13
2.19	Меняем символ	14
2.20	Меняем символ	14
2.21	Смотрим значение регистра	14
2.22	Изменяем регистр командой <code>set</code>	15
2.23	Прописываем команды <code>c</code> и <code>quit</code>	15
2.24	Копируем файл	15
2.25	Создаем и запускаем в отладчике файл	15
2.26	Устанавливаем точку останова	16
2.27	Изучаем полученные данные	16
2.28	Копируем файл	16
2.29	Изменяем файл	17
2.30	Проверяем работу программы	18
2.31	Создаем файл	18
2.32	Изменяем файл	18
2.33	Создаем и смотрим на работу программы(работает неправильно)	19
2.34	Ищем ошибку регистров в отладчике	19
2.35	Меняем файл	20

2.36 Создаем и запускаем файл(работает корректно)	20
---	----

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 Лабораторных работ, и в нем создаем файл (рис. 2.1).

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

Рис. 2.1: Создаем каталог с помощью команды `mkdir` и файл с помощью команды `touch`

Открываем файл с помощью `gedit` и заполняем его в соответствии с листингом 9.1 (рис. 2.2).

```

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

```

Рис. 2.2: Заполняем файл

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

```

[romitsinaar@fedora lab09]$ nasm -f elf lab09-1.asm
[romitsinaar@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[romitsinaar@fedora lab09]$ ./lab09-1
Введите x: 5
2x+7=17
[romitsinaar@fedora lab09]$

```

Рис. 2.3: Запускаем файл и проверяем его работу

Открываем файл для редактирования и изменяем его, добавив подпрограмму в подпрограмму (рис. 2.4).

```
29 ; Подпрограмма вычисления
30 ; выражения "2x+7"
31 _calcul:
32 call _subcalcul
33 mov ebx,2
34 mul ebx
35 add eax,7
36 mov [res],eax
37 ret
38 _subcalcul:
39     mov ebx,3
40     mul ebx
41     sub eax,1
42     ret
```

Matlab

Рис. 2.4: Изменяем файл, добавляя еще одну подпрограмму

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

```
2x+7=35
[romitsinaar@fedora lab09]$ gedit lab09-1.asm
[romitsinaar@fedora lab09]$ nasm -f elf lab09-1.asm
[romitsinaar@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[romitsinaar@fedora lab09]$ ./lab09-1
Введите x: 5
2(3x-1)+7=35
[romitsinaar@fedora lab09]$
```

Рис. 2.5: Запускаем файл и смотрим на его работу

2.2 Отладка программ с помощью GDB

Создаем новый файл в каталоге(рис. 2.6).

```
[romitsinaar@fedora lab09]$ touch lab09-2.asm
[romitsinaar@fedora lab09]$
```

Рис. 2.6: Создаем файл

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


```

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

```

Рис. 2.7: Заполняем файл

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

```

[romitsinaar@fedora lab09]$ nasm -f elf lab09-2.asm
[romitsinaar@fedora lab09]$ ld -m elf_i386 -o lab09-2 lab09-2.o
[romitsinaar@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. 2.8: Загружаем исходный файл в отладчик

Запускаем команду в отладчике (рис. 2.9).

```
To make this setting permanent, add 'set debuginfod enabled of
Hello, world!
[Inferior 1 (process 11166) exited normally]
(gdb) █
```

Рис. 2.9: Запускаем программу командой run

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

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

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

Рис. 2.10: Запускаем программу с брейкпоином

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

```
(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)
```

Рис. 2.11: Смотрим дисассимилированный код программы

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

```

(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. 2.12: Переключаемся на синтаксис Intel

Включаем режим псевдографики (рис. 2.13).

The screenshot shows a GDB window with the following content:

```

romitsinaar@fedora:~/work/arch-... x  romitsinaar@fedora:~/work/arch-... x
[ Register Values Unavailable ]

B> 0x08049000 <_start>      mov     eax,0x4
    0x08049005 <_start+5>    mov     ebx,0x1
    0x0804900a <_start+10>   mov     ecx,0x804a000
    0x0804900f <_start+15>   mov     edx,0x8
    0x08049014 <_start+20>   int     0x80
    0x08049016 <_start+22>   mov     eax,0x4
    0x0804901b <_start+27>   mov     ebx,0x1

native process 12792 In: _start          L9      PC: 0x08049000
(gdb) layout regs
(gdb)

```

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

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

```
romitsinaar@fedora:~/work/arch-... x  romitsinaar@fedora:~/work/arch-... x
[ Register Values Unavailable ]

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

native process 12792 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) info breakpoints
Num      Type          Disp Enb Address      What
1        breakpoint     keep y  0x08049000 lab09-2.asm:9
          breakpoint already hit 1 time
(gdb) 
```

Рис. 2.14: Используем команду info breakpoints и создаем новую точку останова

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

```
(gdb) i b
Num      Type          Disp Enb Address      What
1        breakpoint     keep y  0x08049000 lab09-2.asm:9
          breakpoint already hit 1 time
2        breakpoint     keep y  0x08049031 lab09-2.asm:20
(gdb) 
```

Рис. 2.15: Смотрим информацию

Выполняем 5 инструкций командой si (рис. 2.16).

```
romitsinaar@fedora:~/work/arch-... x  romitsinaar@fedora:~/work/arch-... x
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1d0 0xffffd1d0
ebp      0x0      0x0

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

native process 18914 In: _start L14 PC: 0x8049016
1 breakpoint keep y 0x08049000 lab09-2.asm:9
  breakpoint already hit 1 time
2 breakpoint keep y 0x08049031 lab09-2.asm:20
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)
```

Рис. 2.16: Отслеживаем регистры

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

```
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb)
```

Рис. 2.17: Смотрим значение переменной

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

```
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n"
(gdb)
```

Рис. 2.18: Смотрим значение переменной

Изменим первый символ переменной msg1 (рис. 2.19).

```
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb)
```

Рис. 2.19: Меняем символ

Изменим первый символ переменной msg2 (рис. 2.20).

```
(gdb) set {char}&msg2='W'
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "World!\n\
(gdb)
```

Рис. 2.20: Меняем символ

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

```
(gdb) p/t $edx
$1 = 1000
(gdb) p/s $edx
$2 = 8
(gdb) p/x $edx
$3 = 0x8
(gdb)
```

Рис. 2.21: Смотрим значение регистра

Изменяем регистр ebx (рис. 2.22).

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
(gdb)
```

Рис. 2.22: Изменяем регистр командой set

Выводятся разные значения, так как команда без кавычек присваивает регистру вводимое значение.

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

```
(gdb) c
Continuing.
World!

Breakpoint 2, _start () at lab09-2.asm:20
(gdb)
```

Рис. 2.23: Прописываем команды c и quit

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

```
[romitsinaar@fedora lab09]$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
[romitsinaar@fedora lab09]$
```

Рис. 2.24: Копируем файл

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

```
[romitsinaar@fedora lab09]$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
[romitsinaar@fedora lab09]$ ld -m elf_i386 -o lab09-3 lab09-3.o
[romitsinaar@fedora lab09]$ gdb --args lab09-3 2 3 '5'
```

Рис. 2.25: Создаем и запускаем в отладчике файл

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

```
(gdb) b _start
Note: breakpoint 1 also set at pc 0x80490e8.
Breakpoint 2 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/romitsinaar/work/arch-pc/lab09/lab09-3 2 3 5

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb) x/x $esp
0xffffd1b0: 0x00000004
(gdb)
```

Рис. 2.26: Устанавливаем точку останова

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

```
(gdb) x/x $esp
0xffffd1b0: 0x00000004
(gdb) x/s *(void**)(esp + 4)
0xffffd35c: "/home/romitsinaar/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd380: "2"
(gdb) x/s *(void**)(esp + 12)
0xffffd38b: "3"
(gdb) x/s *(void**)(esp + 16)
0xffffd38d: "5"
(gdb) x/s *(void**)(esp + 20)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 2.27: Изучаем полученные данные

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

##Задание для самостоятельной работы

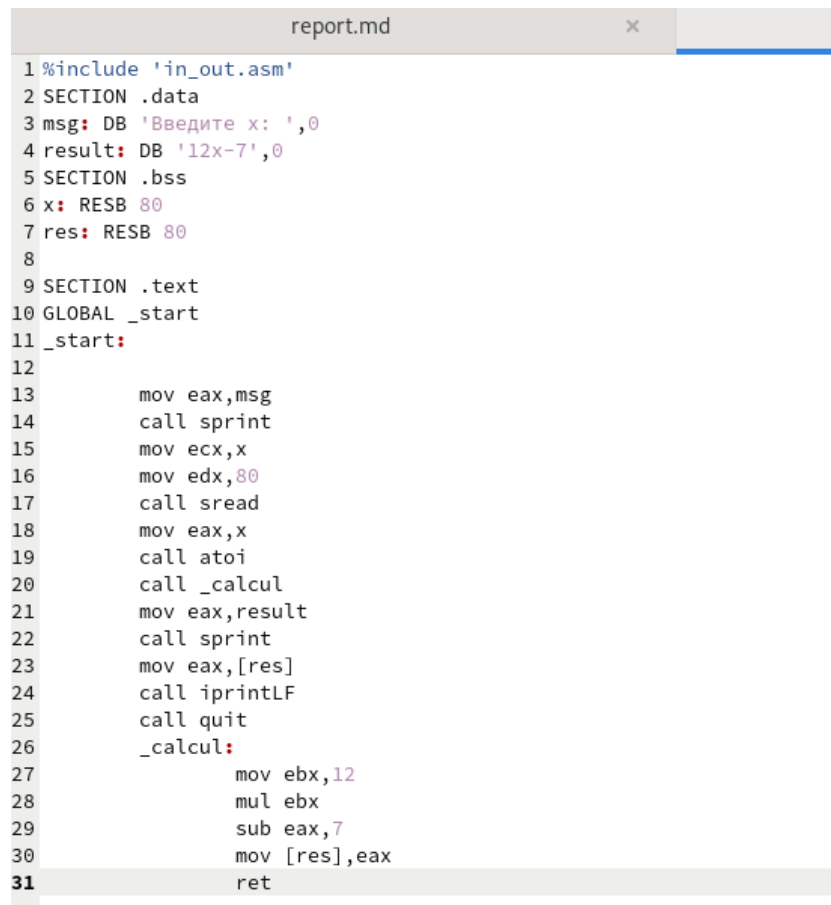
###Задание 1

Копируем файл lab8-4.asm(ср №1 в лабораторной работы 8) в файл с именем lab09-3.asm (рис. 2.28).

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

Рис. 2.28: Копируем файл

Открываем файл с gedit и меняем его, создавая подпрограмму (рис. 2.29).



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '12x-7',0
5 SECTION .bss
6 x: RESB 80
7 res: RESB 80
8
9 SECTION .text
10 GLOBAL _start
11 _start:
12
13     mov eax,msg
14     call sprint
15     mov ecx,x
16     mov edx,80
17     call sread
18     mov eax,x
19     call atoi
20     call _calcul
21     mov eax,result
22     call sprint
23     mov eax,[res]
24     call iprintLF
25     call quit
26     _calcul:
27         mov ebx,12
28         mul ebx
29         sub eax,7
30         mov [res],eax
31     ret
```

Рис. 2.29: Изменяем файл

Создаем исполняемый файл и запускаем его (рис. 2.30).

```
[romitsinaar@fedora lab09]$ nasm -f elf lab09-4.asm
[romitsinaar@fedora lab09]$ ld -m elf_i386 -o lab09-4
[romitsinaar@fedora lab09]$ ./lab09-4
Введите x: 2
12x-7= 17
[romitsinaar@fedora lab09]$
```

Рис. 2.30: Проверяем работу программы

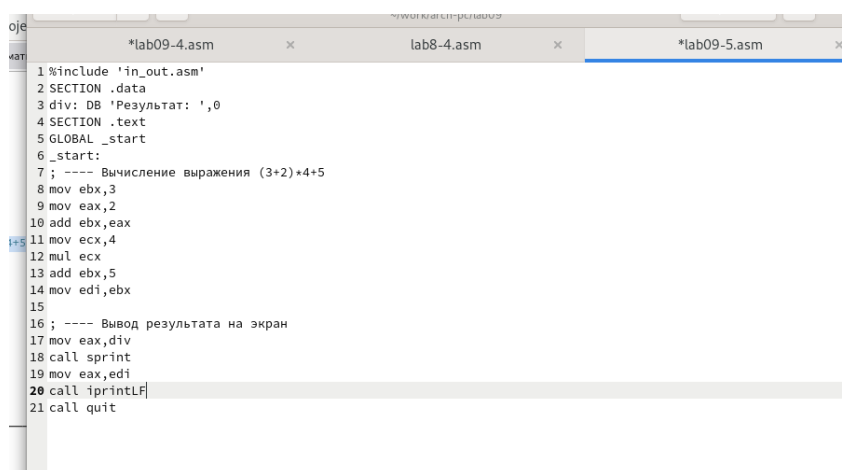
###Задание 2

Создаем новый файл в дирректории (рис. 2.31).

```
[romitsinaar@fedora lab09]$ touch lab09-5.asm
[romitsinaar@fedora lab09]$
```

Рис. 2.31: Создаем файл

Открываем файл с помощью gedit и заполняем его в соответствии с листингом 9.3 (рис. 2.32).



```
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ---- Вычисление выражения (3+2)*4+5
8 mov ebx,3
9 mov eax,2
10 add ebx,eax
11 mov ecx,4
12 mul ecx
13 add ebx,5
14 mov edi,ebx
15
16 ; ---- Вывод результата на экран
17 mov eax,div
18 call sprint
19 mov eax,edi
20 call iprintLF
21 call quit
```

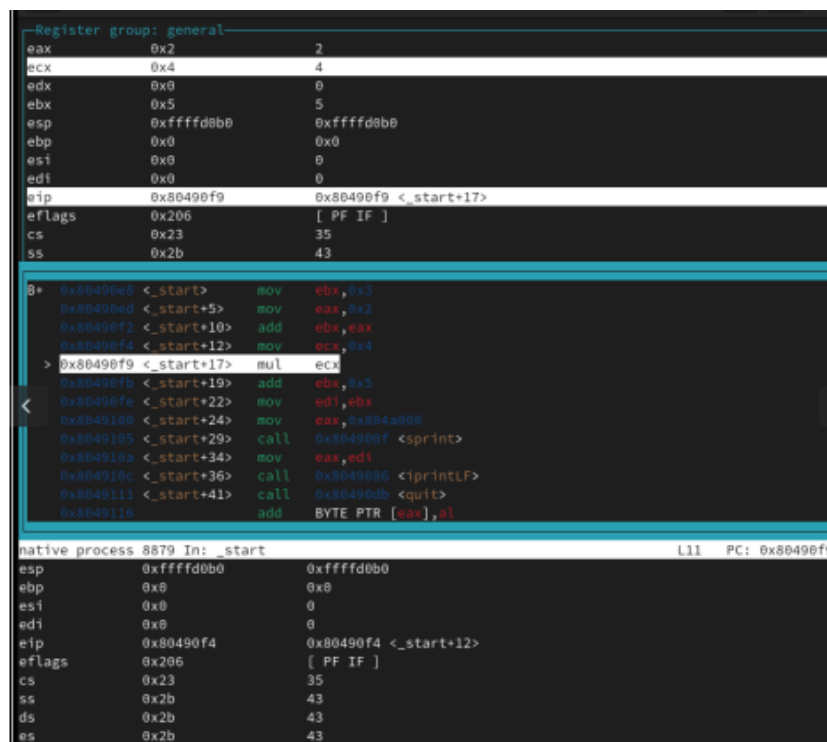
Рис. 2.32: Изменяем файл

Создаем исполняемый файл и запускаем его (рис. 2.33).

```
romitsinaar@fedora lab09]$ nasm -f elf lab09-5.asm
romitsinaar@fedora lab09]$ nasm -m elf_i386 -o lab09-5 lab09-5.o
asm: error: unrecognised option '-m'
romitsinaar@fedora lab09]$ ld -m elf_i386 -o lab09-5 lab09-5.o
romitsinaar@fedora lab09]$ ./lab09-5
результат: 10
romitsinaar@fedora lab09]$
```

Рис. 2.33: Создаем и смотрим на работу программы(работает неправильно)

Создаем исполняемый файл и запускаем его в отладчике gdb и смотрим на изменение регистров командой si (рис. 2.34).



The screenshot shows the GDB interface. At the top, the 'Register group: general' is displayed with values for registers: eax (0x2), ecx (0x4), edx (0x0), ebx (0x5), esp (0xffffd0b0), ebp (0x0), esi (0x0), edi (0x0), eip (0x80490f9), eflags (0x206), cs (0x23), and ss (0x2b). Below this, the assembly code is shown, with the instruction at address 0x80490f9 highlighted: `mul ecx`. The bottom section shows the 'native process 8879 In: _start' with register values: esp (0xffffd0b0), ebp (0x0), esi (0x0), edi (0x0), eip (0x80490f4), eflags (0x206), cs (0x23), ss (0x2b), ds (0x2b), and es (0x2b). The PC is 0x80490f9.

Рис. 2.34: Ищем ошибку регистров в отладчике

Изменяем программу для корректной работы (рис. 2.35).

```

1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ---- Вычисление выражения (3+2)*4+5
8 mov ebx,3
9 mov eax,2
10 add ebx,eax
11 mov ecx,4
12 mul ecx
13 add ebx,5
14 mov edi,eax
15
16 ; ---- Вывод результата на экран
17 mov eax,div
18 call sprint
19 mov eax,edi
20 call iprintLF
21 call quit

```

Рис. 2.35: Меняем файл

Создаем исполняемый файл и запускаем его (рис. 2.36).

```

[romitsinaar@fedora lab09]$ gedit lab09-5.asm
[romitsinaar@fedora lab09]$ nasm -f elf lab09-5.asm
[romitsinaar@fedora lab09]$ ld -m elf_i386 -o lab09-5 lab09-5.o
[romitsinaar@fedora lab09]$ ./lab09-5
Результат: 25
[romitsinaar@fedora lab09]$

```

Рис. 2.36: Создаем и запускаем файл(работает корректно)

3 Выводы

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