

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №10

дисциплина: Архитектура компьютера

Шурыгин Илья Максимович

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Задание для самостоятельной работы:	16
5	Выводы	20

Список иллюстраций

3.1	Создали каталог lab10 в Midnight Commander и файл lab10-1.asm .	7
3.2	Вычисляет $f(x) = 2x + 7$	7
3.3	Код измененной программы	8
3.4	Вычисляет $f(g(x))$	8
3.5	Создание исполняемого файла	9
3.6	Запуск программы lab10-2.asm	9
3.7	Устанавливаем брейкпоинт	9
3.8	Дисассимилированный код программы	10
3.9	Отображение команд с Intel'овским синтаксисом	10
3.10	Проверяем брейкпоинт	11
3.11	Устанавливаем брейкпоинт	11
3.12	Адрес предпоследней инструкции	12
3.13	Значения переменных msg1 и msg2	12
3.14	Заменяем символы	13
3.15	Вывод значения регистра edx	13
3.16	Замена значения регистра ebx	13
3.17	Загрузка исполняемого файла в отладчик	14
3.18	Установили точку останова и проверили число аргументов	14
3.19	Просмотр позиций стека	15
4.1	Результат работы программы	16
4.2	Код программы	17
4.3	Окно отладчика	18
4.4	Вывод программы	18
4.5	Код программы	19

Список таблиц

1 Цель работы

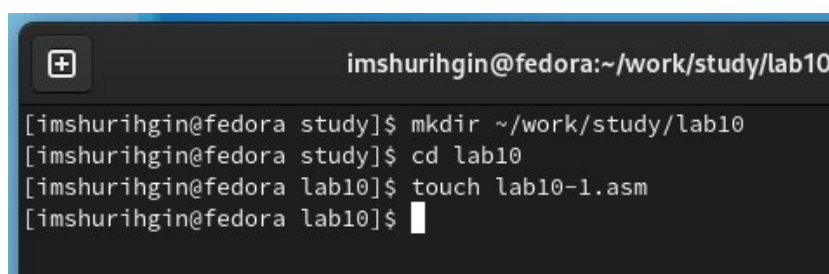
Приобрети навык написания программ с использованием подпрограмм. Познакомиться с методами отладки при помощи GDB и его основными возможностями.

2 Задание

Необходимо написать программы, которые вычисляют значение функции, а затем проверить их работу в отладчике.

3 Выполнение лабораторной работы

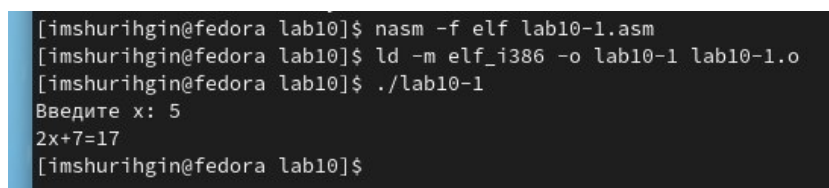
1. Откроем Midnight Commander и перейдем в каталог `~/work/study`. Создадим каталог для программ лабораторной работы N10, перейдем в него и создадим файл `lab10-1.asm`. (рис. 3.1)



```
imshurihgin@fedora:~/work/study/lab10
[imshurihgin@fedora study]$ mkdir ~/work/study/lab10
[imshurihgin@fedora study]$ cd lab10
[imshurihgin@fedora lab10]$ touch lab10-1.asm
[imshurihgin@fedora lab10]$
```

Рис. 3.1: Создали каталог `lab10` в Midnight Commander и файл `lab10-1.asm`

2. Запишем в файл `lab10-1.asm` текст программы из листинга 10.1. Она вычисляет $f(x) = 2x + 7$ с помощью подпрограммы `_calcul`. Изменим текст программы. Теперь программа вычисляет выражение $f(g(x))$, где $f(x) = 2x + 7$, $g(x) = 3x - 1$. (рис. 3.2)(рис. 3.3)(рис. 3.4)



```
[imshurihgin@fedora lab10]$ nasm -f elf lab10-1.asm
[imshurihgin@fedora lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[imshurihgin@fedora lab10]$ ./lab10-1
Введите x: 5
2x+7=17
[imshurihgin@fedora lab10]$
```

Рис. 3.2: Вычисляет $f(x) = 2x + 7$

```

;-----
;
; Подпрограмма вычисления
; выражения "2(3x-1)+7"
_calcul:
call _subcalcul
mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret ; выход из подпрограммы
_subcalcul:
mov ebx,3
mul ebx
dec eax
ret
1Помощь 2Сохранить 3Блок 4Замена 5К

```

Рис. 3.3: Код измененной программы

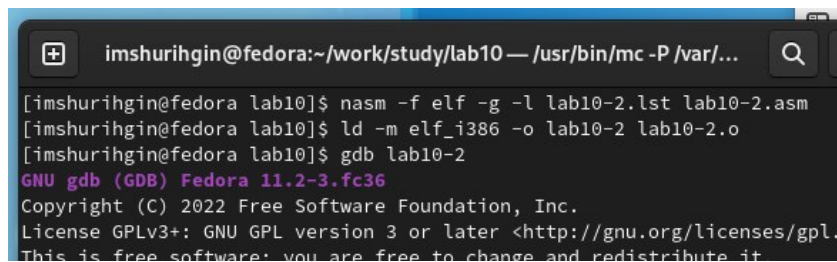
```

imshurighin@fedora:~/work/study/lab10
[imshurighin@fedora lab10]$ nasm -f elf lab10-1.asm
[imshurighin@fedora lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[imshurighin@fedora lab10]$ ./lab10-1
Введите x: 5
2(3x-1)+7=35
[imshurighin@fedora lab10]$

```

Рис. 3.4: Вычисляет $f(g(x))$

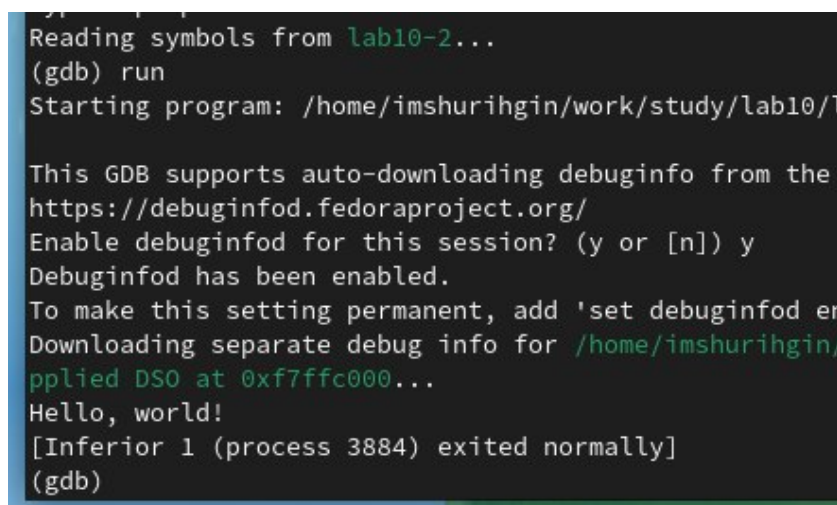
3. Создадим файл lab10-2.asm с текстом программы из Листинга 10.2.(Программа печати сообщения Hello world!). Для работы с GDB в исполняемый файл добавим отладочную информацию, для этого трансляцию программ необходимо проводить с ключом '-g'.(рис. 3.5)



```
imshurihgin@fedora:~/work/study/lab10 — /usr/bin/mc -P /var/...
[imshurihgin@fedora lab10]$ nasm -f elf -g -l lab10-2.lst lab10-2.asm
[imshurihgin@fedora lab10]$ ld -m elf_i386 -o lab10-2 lab10-2.o
[imshurihgin@fedora lab10]$ gdb lab10-2
GNU gdb (GDB) Fedora 11.2-3.fc36
Copyright (C) 2022 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.
```

Рис. 3.5: Создание исполняемого файла

4. Запустим программу в оболочке GDB с помощью команды run.(рис. 3.6)

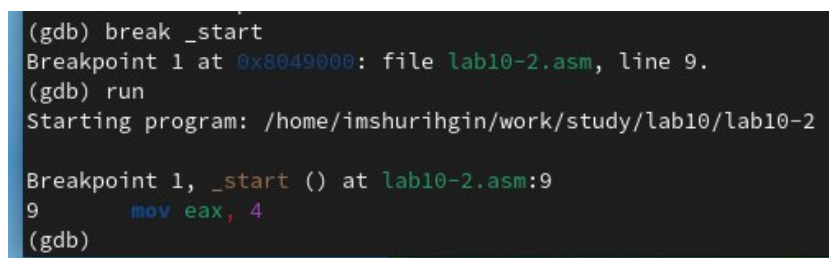


```
Reading symbols from lab10-2...
(gdb) run
Starting program: /home/imshurihgin/work/study/lab10/lab10-2

This GDB supports auto-downloading debuginfo from the
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' to your .gdbinit
Downloading separate debug info for /home/imshurihgin/work/study/lab10/lab10-2
Applied DSO at 0xf7ffc000...
Hello, world!
[Inferior 1 (process 3884) exited normally]
(gdb)
```

Рис. 3.6: Запуск программы lab10-2.asm

5. Установим брейкпоинт на метку `_start` и посмотрим дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start`.(рис. 3.7)(рис. 3.8)



```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 9.
(gdb) run
Starting program: /home/imshurihgin/work/study/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:9
9      mov eax, 4
(gdb)
```

Рис. 3.7: Устанавливаем брейкпоинт

```

(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. 3.8: Дисассимилированный код программы

6. Переключимся на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel.`(рис. 3.9)

```

(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. 3.9: Отображение команд с Intel'овским синтаксисом

7. Включим режим псевдографики для более удобного анализа программы и проверим, что была установлена точка останова(`_start`) с помощью команды `info breakpoints.`(рис. 3.10)

```
native process 3938 In: _start
(gdb) layout regs
(gdb) info breakpoints
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000 lab10-2.asm:9
        breakpoint already hit 1 time
(gdb)
```

Рис. 3.10: Проверяем брейкпоинт

8. Установим еще одну точку останова по адресу инструкции. Определим адрес предпоследней инструкции (`mov ebx,0x0`) и установим точку останова. (рис. 3.11)(рис. 3.12)

```
(gdb) break *0x08049031
Breakpoint 4 at 0x08049031: file lab10-2.asm, line 20.
(gdb) i b
Num      Type           Disp Enb Address      What
3        breakpoint     keep y   0x08049000 lab10-2.asm:9
4        breakpoint     keep y   0x08049031 lab10-2.asm:20
(gdb)
```

Рис. 3.11: Устанавливаем брейкпоинт

```

3
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      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

native process 5471 In: _start
edx      0x8      8
ebx      0x1      1
esp      0xffffd1e0 0xffffd1e0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049014 0x8049014 <_start+20>
eflags   0x202    [ IF ]
--Type <RET> for more, q to quit, c to continue without paging
(gdb) si
(gdb)

```

Рис. 3.12: Адрес предпоследней инструкции

- Посмотрим значение переменной msg1 по имени, а значение переменной msg2 по адресу. Затем изменим первый символ переменной msg1 и любой символ во второй переменной msg2.(рис. 3.13)(рис. 3.14)

```

(gdb) si
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb)

```

Рис. 3.13: Значения переменных msg1 и msg2

```

(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) set {char}&msg2='L'
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "Lor!d!\n\034"
(gdb)

```

Рис. 3.14: Заменяем символы

10. Выведем в различных форматах значение регистра `edx`, а затем с помощью команды `set` изменим значение регистра `ebx`. Завершим выполнение программы с помощью команды `continue` или `stepi` и выйдем из GDB с помощью команды `quit`. (рис. 3.15) (рис. 3.16)

```

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

```

Рис. 3.15: Вывод значения регистра `edx`

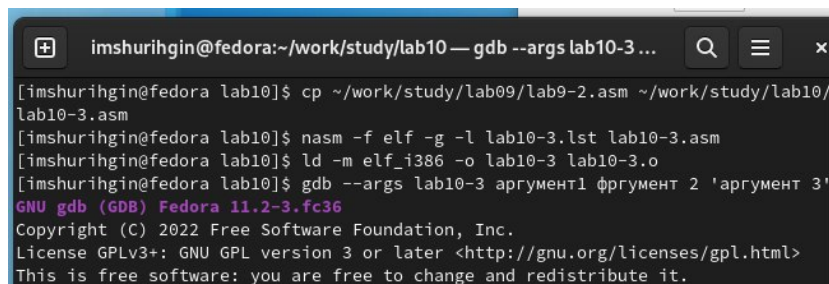
```

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

```

Рис. 3.16: Замена значения регистра `ebx`

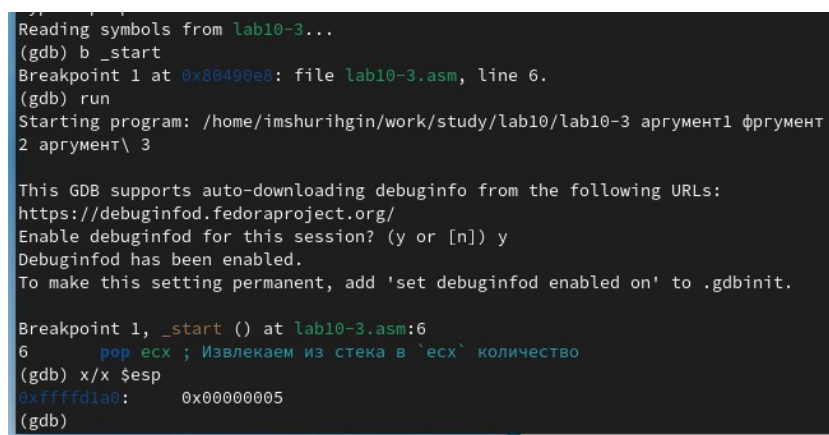
11. Скопируем файл `lab9-2.asm`, созданный при выполнении лабораторной работы N9, создадим исполняемый файл и загрузим исполняемый файл в отладчик, указав аргументы. (рис. 3.17)



```
imshurihgin@fedora:~/work/study/lab10 — gdb --args lab10-3 ...
[imshurihgin@fedora lab10]$ cp ~/work/study/lab09/lab9-2.asm ~/work/study/lab10/lab10-3.asm
[imshurihgin@fedora lab10]$ nasm -f elf -g -l lab10-3.lst lab10-3.asm
[imshurihgin@fedora lab10]$ ld -m elf_i386 -o lab10-3 lab10-3.o
[imshurihgin@fedora lab10]$ gdb --args lab10-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (GDB) Fedora 11.2-3.fc36
Copyright (C) 2022 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.
```

Рис. 3.17: Загрузка исполняемого файла в отладчик

12. Установим точку останова перед первой инструкцией в программе и запустим ее. Проверим число аргументов командной строки, которое располагается в регистре esp.(рис. 3.18)



```
Reading symbols from lab10-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab10-3.asm, line 6.
(gdb) run
Starting program: /home/imshurihgin/work/study/lab10/lab10-3 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 lab10-3.asm:6
6      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffd1a0: 0x00000005
(gdb)
```

Рис. 3.18: Установили точку останова и проверили число аргументов

13. Посмотрим остальные позиции стека – по адресу (esp+4) располагается адрес в памяти, где находится имя программы, по адресу (esp+8) храниться адрес первого аргумента и т.д.(рис. 3.19)

```

Breakpoint 1, _start () at lab10-3.asm:6
6      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffd1a0: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffd356: "/home/imshurihgin/work/study/lab10/lab10-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd381: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd393: "фргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd3a4: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd3a6: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

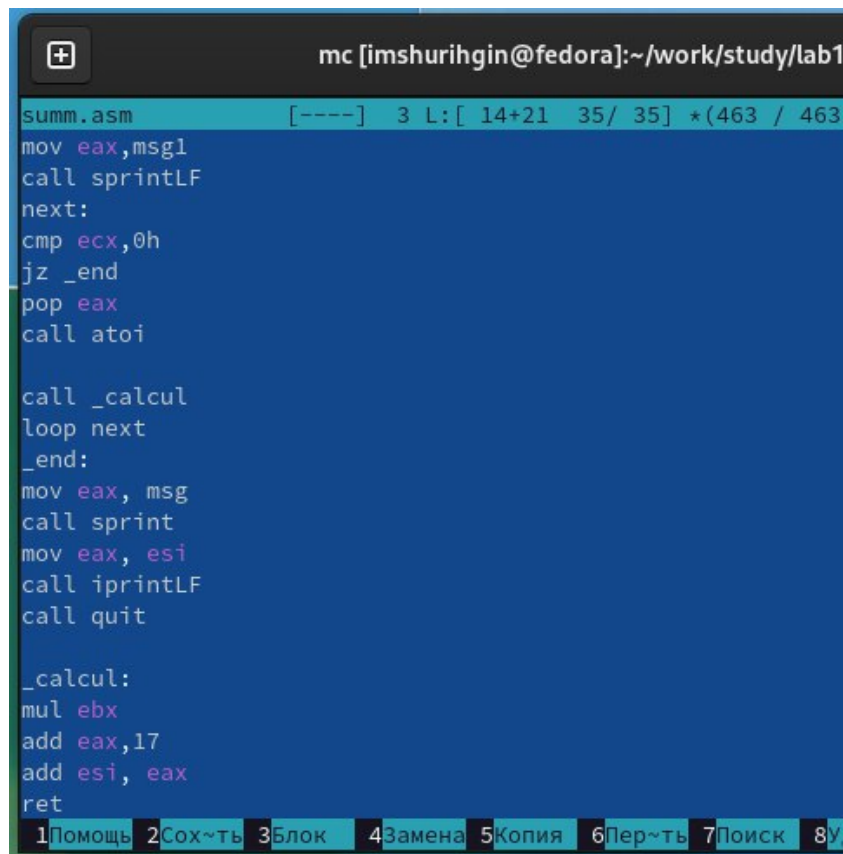
Рис. 3.19: Просмотр позиций стека

4 Задание для самостоятельной работы:

1. Преобразуем программу из лабораторной работы N9 (Задание N1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму.(рис. 4.1)(рис. 4.2)

```
[imshurihgin@fedora lab10]$ nasm -f elf summ.asm
[imshurihgin@fedora lab10]$ ld -m elf_i386 -o summ summ.o
[imshurihgin@fedora lab10]$ ./summ 1 2
Функция:  $f(x)=17+5x$ 
Результат: 49
[imshurihgin@fedora lab10]$ ./summ 1 2 3 4
Функция:  $f(x)=17+5x$ 
Результат: 118
[imshurihgin@fedora lab10]$
```

Рис. 4.1: Результат работы программы



```
mc [imshurhgin@fedora]:~/work/study/lab1
summ.asm [----] 3 L: [ 14+21 35/ 35] *(463 / 463
mov eax,msg1
call printf
next:
cmp ecx,0h
jz _end
pop eax
call atoi

call _calcul
loop next
_end:
mov eax, msg
call printf
mov eax, esi
call printf
call quit

_calcul:
mul ebx
add eax,17
add esi, eax
ret

1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перейти 7Поиск 8Удалить
```

Рис. 4.2: Код программы

2. Нужно исправить работу программы, вычисляющей выражение: $(3+2)*4+5$ с помощью отладчика GDB, анализируя изменения значений регистров. В отладчике видно, что программа умножает `eax` на 4, вместо того, чтобы умножать значение в регистре `ebx`, где лежит $(3+2)$. (рис. 4.3)(рис. 4.4)(рис. 4.5)

```
imshurihgin@fedora:~/work/study/lab10 — /usr/bin/mc -P /v

Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd1c0 0xffffd1c0

0x80490f2 <_start+10> add    %eax,%ebx
0x80490f4 <_start+12> mov    $0x4,%ecx
0x80490f9 <_start+17> mul    %ecx
> 0x80490fb <_start+19> add    $0x5,%ebx
0x80490fe <_start+22> mov    %ebx,%edi
0x8049100 <_start+24> mov    $0x804a000,%eax
0x8049105 <_start+29> call   0x804900f <sprint>

native process 3438 In: _start

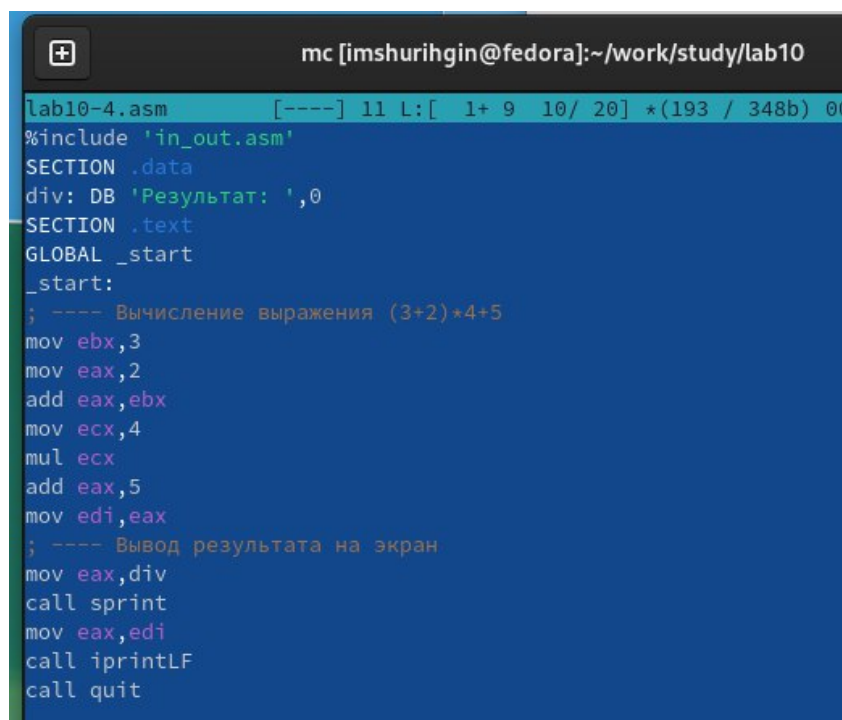
Breakpoint 1, _start () at lab10-4.asm:8
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) 
```

Рис. 4.3: Окно отладчика

```
imshurihgin@fedora:~/work/study/lab10

[imshurihgin@fedora lab10]$ nasm -f elf -g -l lab10-4.lst lab10-4.asm
[imshurihgin@fedora lab10]$ ld -m elf_i386 -o lab10-4 lab10-4.o
[imshurihgin@fedora lab10]$ ./lab10-4
Результат: 25
[imshurihgin@fedora lab10]$ 
```

Рис. 4.4: Вывод программы



```
mc [imshurihgin@fedora]:~/work/study/lab10
lab10-4.asm [----] 11 L:[ 1+ 9 10/ 20] *(193 / 348b) 00
#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 eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```

Рис. 4.5: Код программы

5 Выводы

В данной лабораторной работе я научился писать простые программы на языке ассемблера NASM, а именно: создал программу которая вычисляет значение функции с помощью подпрограммы. А также научился находить ошибки в коде программ с помощью отладчика GDB.