Отчёт по лабораторной работе №10

Дисциплина: Архитектура компьютера

Батова Ирина Сергеевна

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

2 Выполнение лабораторной работы

Сначала создаем каталог ‘lab10’ с помощью команды mkdir, переходим в него с помощью команды cd и создаем в нем файл ‘lab10-1.asm’ с помощью команды touch (рис. 2.1).

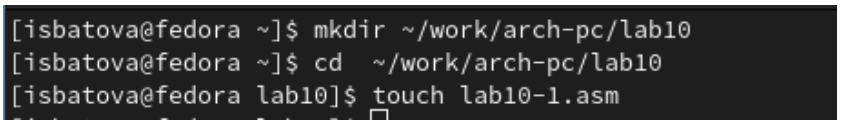

Рис. 2.1: Создание необходимых для работы каталогов и файлов

Открываем файл ‘lab10-1.asm’ и вводим листинг 10.1 из лабораторной работы (рис. 2.2).

```
%include 'in_out.asm'

SECTION .data
        msg: DB 'Введите x: ',0
        result: DB '2x+7= ',0

SECTION .bss
       x: RESB 80
       rez: 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,[rez]
        call iprintLF

        call quit

_calcul:

        mov ebx,2
        mul ebx
        add eax,7
        mov [rez],eax

        ret
```

Рис. 2.2: Ввод листинга 10.1

Создаем исполняемый файл и запускаем его (рис. 2.3). Программа выводит правильный результат, значит, она написана корректно.

```
[isbatova@fedora lab10]$ nasm -f elf lab10-1.asm
[isbatova@fedora lab10]$ ld -m elf_i386 lab10-1.o -o lab10-1
[isbatova@fedora lab10]$ ./lab10-1
Введите x: 3
2x+7= 13
```

Рис. 2.3: Запуск программы из файла ‘lab10-1.asm’

Далее вновь открываем файл ‘lab10-1.asm’ и редактируем его, добавляя еще одну подпрограмму (для вычисления сложной функции). (рис. 2.4).

```
34 _calcul:
35
36         call _subcalcul
37         mov ebx,2
38         mul ebx
39         add eax,7
40         mov [rez],eax
41
42         ret
43
44 _subcalcul:
45
46         mov ecx,3
47         mul ecx
48         sub eax,1
49         mov [rez],eax
50
51         ret
```

Рис. 2.4: Добавление еще одной подпрограммы

Создаем исполняемый файл и запускаем его (рис. 2.5). Программа выводит правильный результат, значит, она написана корректно.

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

Рис. 2.5: Запуск измененной программы из файла ‘lab10-1.asm’

Для дальнейшей работы создаем файл ‘lab10-2.asm’ (рис. 2.6).

```
[isbatova@fedora lab10]$ touch lab10-2.asm
```

Рис. 2.6: Создание файла ‘lab10-2.asm’

Открываем этот файл ‘lab10-2.asm’ и вводим листинг 10.2 из лабораторной работы (рис. 2.7).

```
 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
11 _start:
12         mov eax, 4
13         mov ebx, 1
14         mov ecx, msg1
15         mov edx, msg1Len
16         int 0x80
17
18         mov eax, 4
19         mov ebx, 1
20         mov ecx, msg2
21         mov edx, msg2Len
22         int 0x80
23
24               mov eax,1
25         mov ebx,0
26         int 0x80
```

Рис. 2.7: Ввод листинга 10.2

Создаем исполняемый файл, причем трансляцию выполняем с ключом ‘-g’, чтобы мы могли с этим файлом работать в отладчике (рис. 2.8).

```
[isbatova@fedora lab10]$ nasm -f elf -g -l lab10-2.lst lab10-2.asm
[isbatova@fedora lab10]$ ld -m elf_i386 -o lab10-2 lab10-2.o
```

Рис. 2.8: Создание исполняемого файла 10-2

Загружаем файл в отладчик gdb (рис. 2.9).

Рис. 2.9: Загрузка файла 10-2 в отладчик

Для проверки работы программы запускаем ее командой ‘run’ (рис. 2.10). Видим, что программа работает корректно.

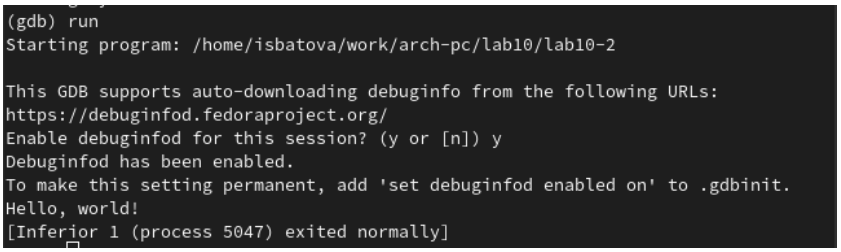

Рис. 2.10: Запуск программы 10-2 в отладчике

Далее устанавливаем брейкпоинт на метку старт и снова запускаем программу (рис. 2.11).

Рис. 2.11: Установка первого брейкпоинта

После этого вводим команду ’disassemble _start’, чтобы посмотреть дисассимилированный код (рис. 2.12).

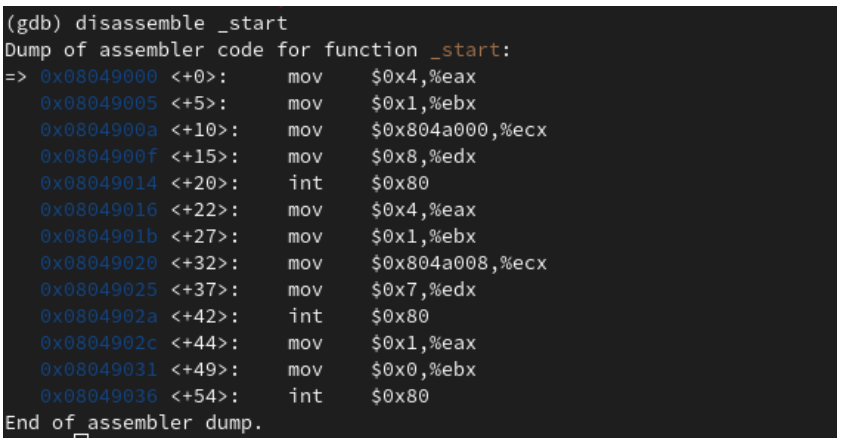

Рис. 2.12: Просмотр дисассимилированного кода

Далее мы переключаемся на отображение команд с intel'овским синтаксисом (команда 'set disassembly-flavor intel') и вновь просматриваем дисассимилированный код (рис. 2.13).

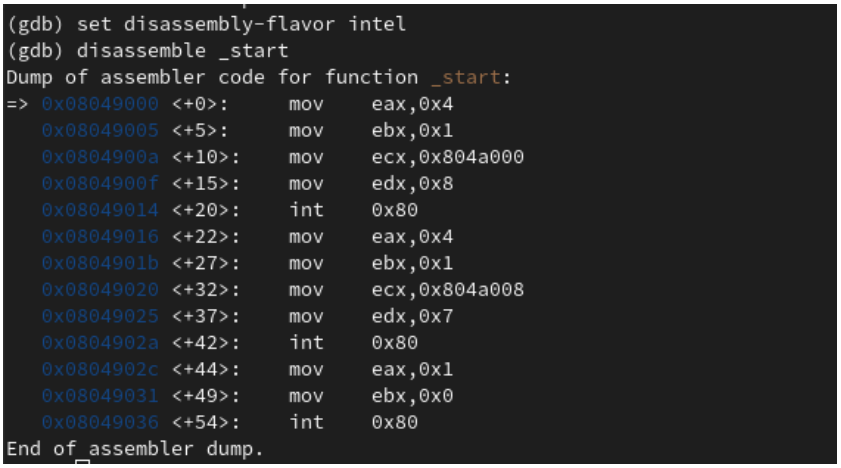

Рис. 2.13: Просмотр intel'овского отображения

Разница между отображениями заключается в удобстве просмотра - в intel'овском синтаксисе отсутствуют знаки '$' и '%', а также идет сначала регистр, а потом его значение. Это делает отображение более наглядным.

После этого вводим команды 'layout asm' и 'layout regs'. Данные команды включают режим псевдографики, что делает анализ программы удобнее (рис. 2.14).

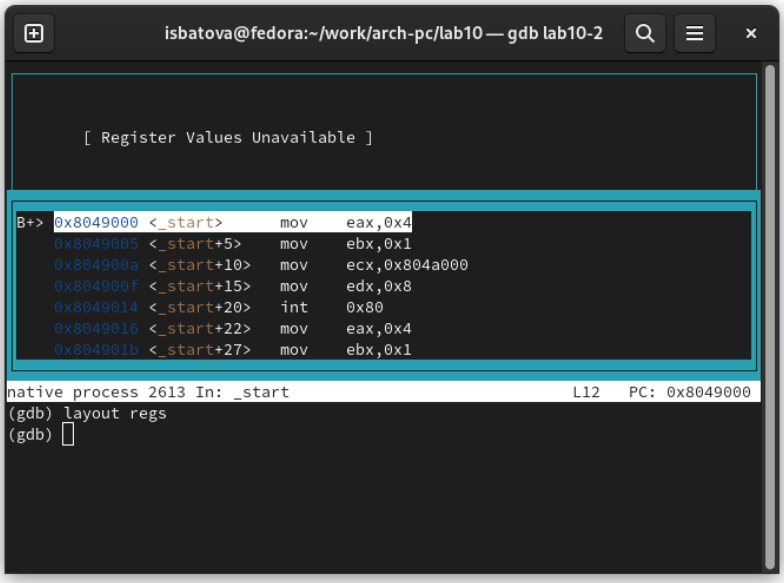

Рис. 2.14: Режим псевдографики

Вводим команду ‘info breakpoints’ для просмотра установленных нами точек останова. Видим одну точку останова, которую мы поставили выше (рис. 2.15).

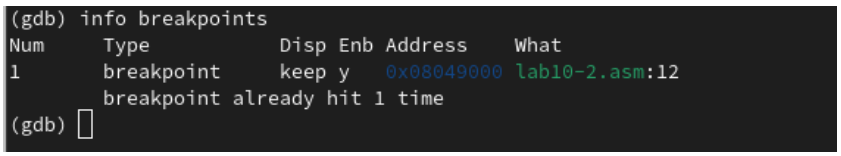

Рис. 2.15: Просмотр брейкпоинтов

После этого нам нужно установить точку останова на предпоследнюю инструкцию. В окне, расположенном посередине, определяем адрес этой инструкции и устанавливаем брейкпоинт по этому адресу. После этого проверяем командой ‘info breakpoints’ наличие двух точек останова (рис. 2.16).

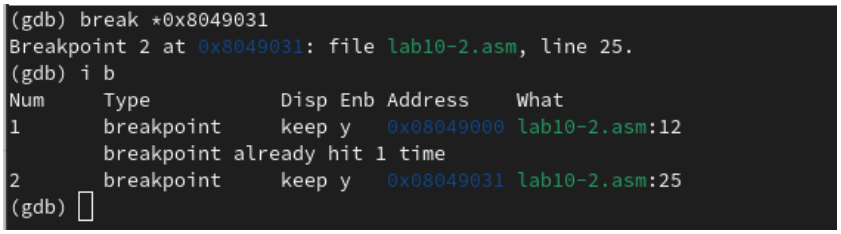

Рис. 2.16: Установка второго брейкпоинта

На этом моменте у нас данные значения регистров (рис. 2.17).

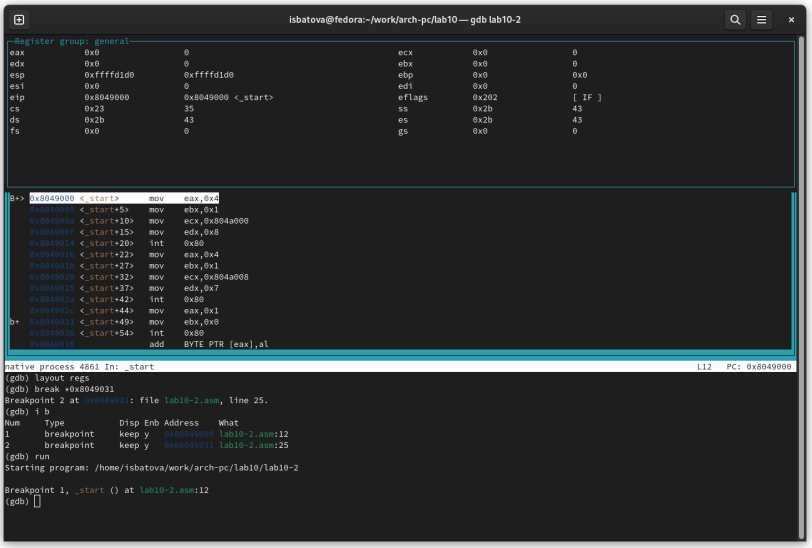

Рис. 2.17: Значения регистров

Далее вводим команду ‘si 5’ и видим, что значения регистров eax, edx и eip меняются (рис. 2.18).

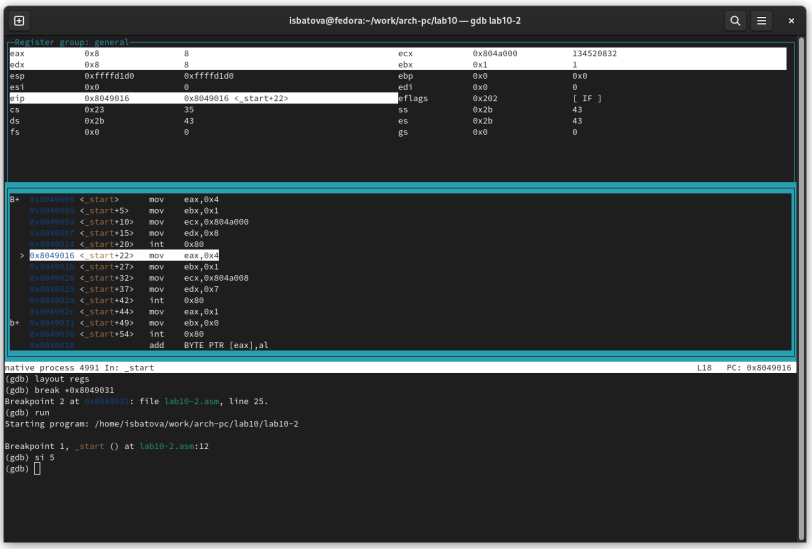

Рис. 2.18: Изменение значений регистров

Просматриваем значения регистров с помощью команды ‘info registers’ (рис. 2.19).

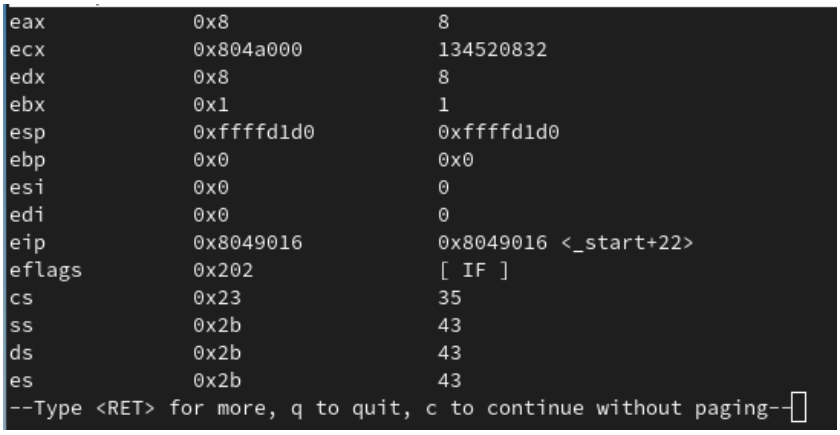

Рис. 2.19: Просмотр значений регистров

Далее мы смотрим содержимое переменной ‘msg1’ (по имени) (рис. 2.20).

Рис. 2.20: Содержимое переменной msg1

После этого с помощью окна посередине определяем адрес переменной ‘msg2’ и смотрим ее содержимое по адресу (рис. 2.21).

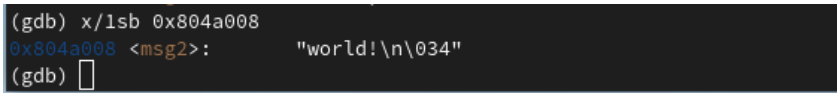

Рис. 2.21: Содержимое переменной msg2

Далее исследуем команду изменения содержимого регистров. Вводим “set {char}&msg1=‘h’ ”. Эта команда заменит нам в регистре msg1 первой символ на h. Проверяем этого, просматривая содержимое регистра (рис. 2.22).

Рис. 2.22: Новое содержимое переменной msg1

Аналогично заменяем первый символ в регистре msg2 на ‘L’ (команда “set {char}&msg2=‘L’ ”) и проверяем это (рис. 2.23).

Рис. 2.23: Новое содержимое переменной msg2

Далее мы рассматриваем команду вывода значения регистров. Выводим в трех различных форматах значение регистра edx (рис. 2.24).

Рис. 2.24: Значение регистра edx

После этого мы изменяем значение регистра ebx с помощью команды ‘set’ (рис. 2.25). Мы получаем разные значения, так как во втором случае мы приравниваем регистр к двойке, а в первом вносим значение 2 в регистра.

Рис. 2.25: Изменение значения регистра ebx

Далее мы завершаем работу в отладчике и выходим из него (команда ‘quit’) (рис. 2.26).

Рис. 2.26: Завершение работы в отладчике

Для дальнейшей работы копируем файл ‘lab9-2.asm’ из девятой лабораторной работы в каталог ‘lab10’ с именем ‘lab10-3.asm’ (рис. 2.27).

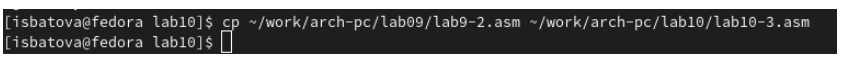

Рис. 2.27: Копирование файла

Создаем исполняемый файл (трансляцию выполняем с ключом ‘-g’ (рис. 2.28).

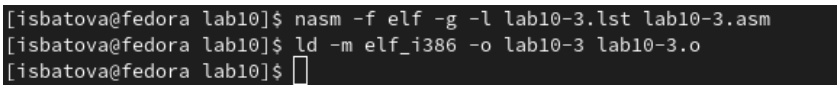

Рис. 2.28: Создание исполняемого файла 10-3

После этого запускаем файл в отладчике с указанием аргументов (рис. 2.29).

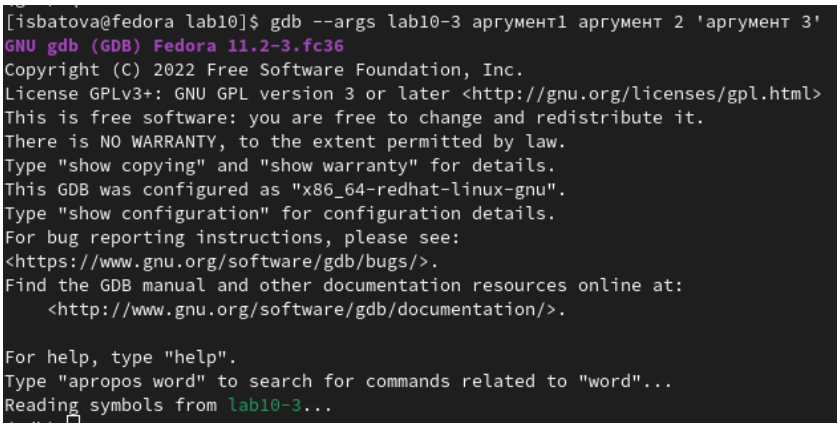

Рис. 2.29: Запуск файла 10-3 в отладчике

Далее устанавливаем брейкпоинт на метку старт и запускаем программу (рис. 2.30).

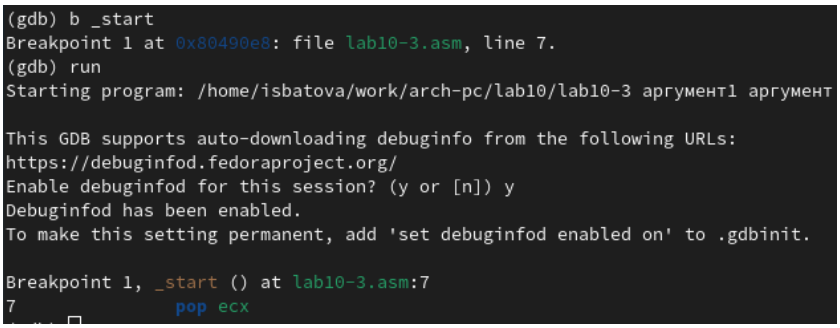

Рис. 2.30: Установка брейкпоинта и запуск

Проверяем адрес вершины стека и наличия там 5 позиций (рис. 2.31).

Рис. 2.31: Адрес вершины стека

После этого просматриваем остальные позиции стека (рис. 2.32). Как мы видим, шаг изменения равен четырем. Так происходит потому, что один стек может хранить до четырех байт, и для каждой позиции используется свой стек.

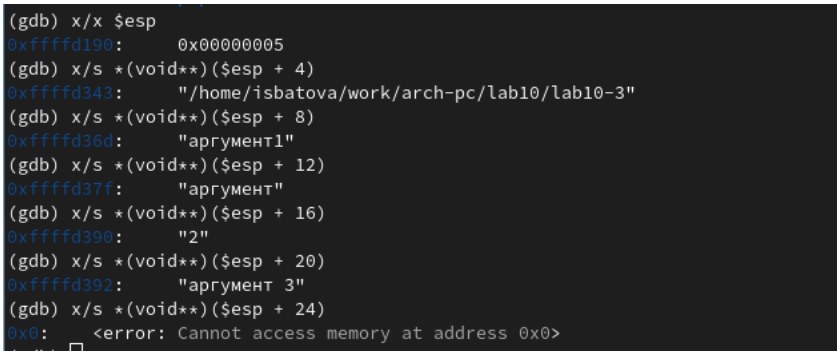

Рис. 2.32: Остальные позиции стека

3 Задание для самостоятельной работы

1. Для выполнения первого задания копируем файл из заданий для самостоятельной работы из лабораторной работы №9 (я копировала с именем ‘lab10-4.asm’). Открываем файл и редактируем его так, чтобы вычисление значений у нас было реализовано как подпрограмма (рис. 3.1).

```
%include 'in_out.asm'

SECTION .data
        primer: DB 'Функция: f(x)=15x+2',0
        rezult: DB 'Результат: ',0

SECTION .text
        global _start

_start:
        mov eax, primer
        call sprintLF

        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,rezult
        call sprint
        mov eax,esi
        call iprintLF
        call quit

_calcul:
        mov ebx,15
        mul ebx
        add eax,2
        ret
```

Рис. 3.1: Редактирование программы

Сохраняем изменения, создаем исполняемый файл и запускаем его (рис. 3.2). Если посчитать аналитически, получается такой же ответ, поэтому программа работает корректно.

```
[isbatova@fedora lab10]$ nasm -f elf lab10-4.asm
[isbatova@fedora lab10]$ ld -m elf_i386 lab10-4.o -o lab10-4
[isbatova@fedora lab10]$ ./lab10-4 1 2 3 4
Функция: f(x)=15x+2
Результат: 158
```

Рис. 3.2: Запуск программы из файла ‘lab10-4.asm’

2. Для начала выполнения второго задания создаем файл ‘lab10-5.asm’ и вводим в него листинг 10.3 - вычисление выражения (3+2)*4+5 (рис. 3.3).

```
%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,ebx

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

Рис. 3.3: Ввод листинга 10.3

Создаем исполняемый файл и запускаем его (рис. 3.4). Если посчитать аналитически, ответ должен быть 25, а программа выдает 10.

```
[isbatova@fedora lab10]$ nasm -f elf lab10-5.asm
[isbatova@fedora lab10]$ ld -m elf_i386 lab10-5.o -o lab10-5
[isbatova@fedora lab10]$ ./lab10-5
Результат: 10
```

Рис. 3.4: Запуск программы из файла ‘lab10-4.asm’

Для поиска ошибки создаем исполняемый файл с ключом ‘-g’ и загружаем файл в отладчик (рис. 3.5).

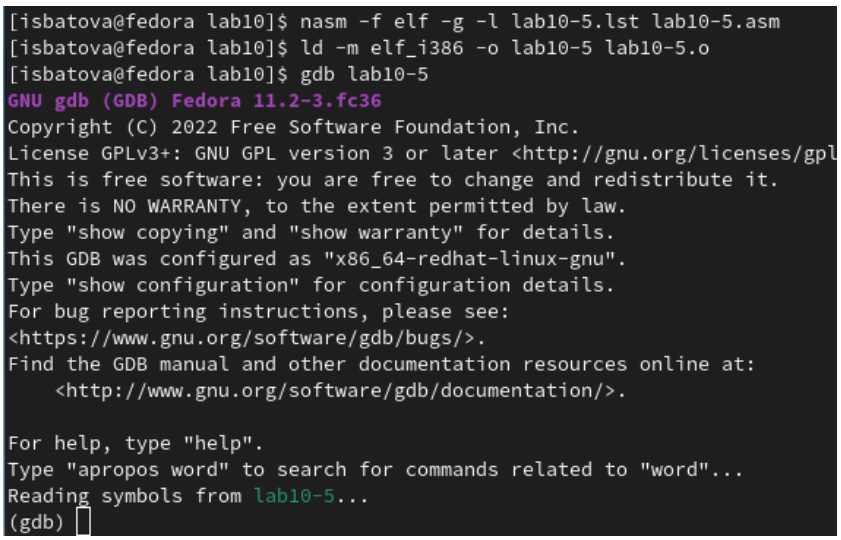

Рис. 3.5: Загрузка программы 10-5 в отладчик

Устанавливаем точку останова на метку start (рис. 3.6) и дисассемилируем код (с intel'овским изображением (рис. 3.7).

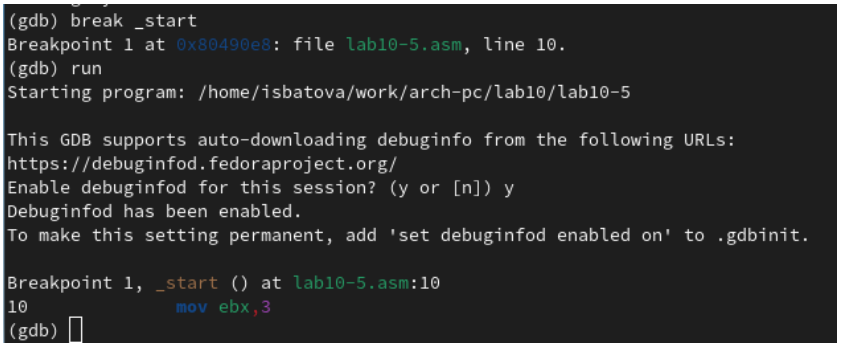

Рис. 3.6: Установка точки останова на метку start

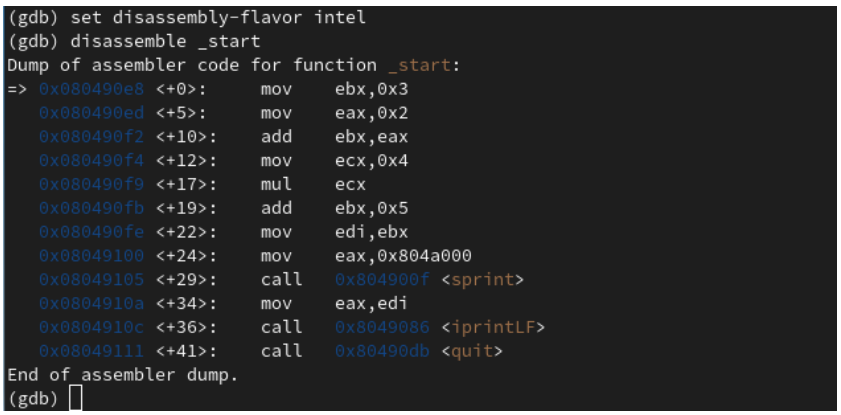

Рис. 3.7: Получение intel'овского отображения

Открываем режим псевдографики. Далее, с помощью команды 'si', выполняем программу по одному шагу и на каждом смотрим значения регистров eax и ebx. Путем анализа понимаем, что на 5 шаге у нас на регистр ecx (то есть на 4) умножается не ebx, а eax, поэтому программа и выводит неверный результат. Чтобы исправить это, мы после прибавления к регистру ebx регистра eax записываем получившееся значение в eax. Далее у нас eax умножается на 4, после чего мы обратно записываем значение регистра eax в регистр ebx (рис. 3.8).

Рис. 3.8: Исправление ошибки

Далее вводим команду 'continue', чтобы наша программа была выполнена до конца и получаем верный результат (рис. 3.9).

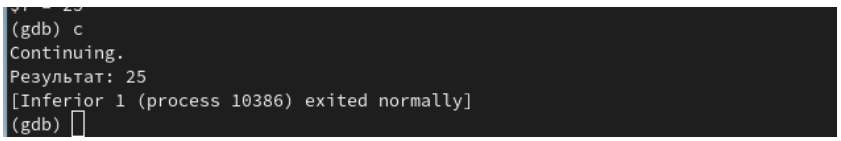

Рис. 3.9: Получение верного результата

4 Выводы

В данной лабораторной работе мной были приобретены навыки написания программ с использованием подпрограмм, а также навыки работы с методами отладки при помощи GDB.