Отчёт по лабораторной работе №10

Дисциплина: Архитектура компьютера

Батова Ирина Сергеевна

Содержание

# 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. 1).

Рис. 1: Создание необходимых для работы каталогов и файлов

Рис. 1: Создание необходимых для работы каталогов и файлов

Открываем файл ‘lab10-1.asm’ и вводим листинг 10.1 из лабораторной работы (рис. 2).

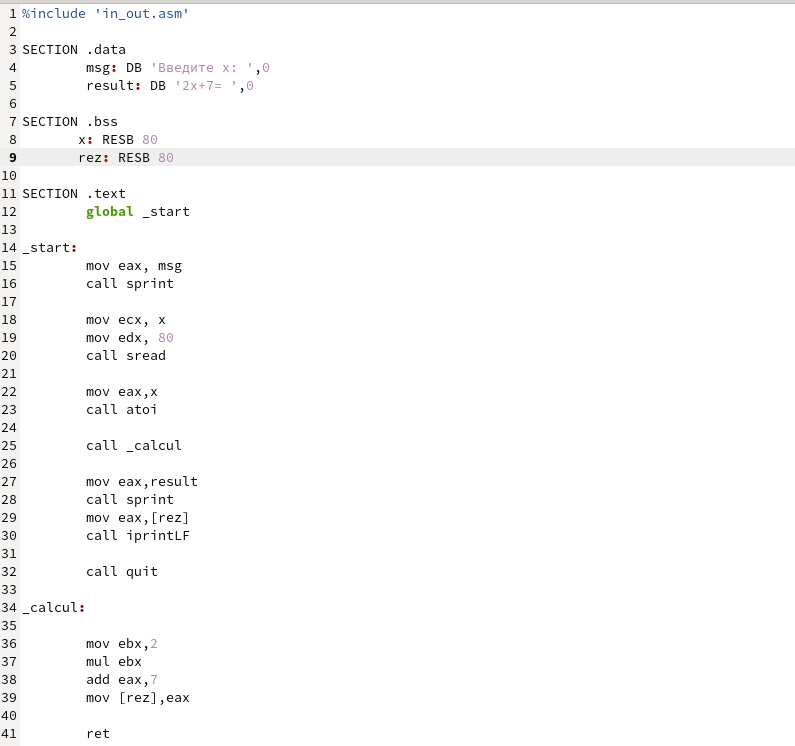


Рис. 2: Ввод листинга 10.1

Создаем исполняемый файл и запускаем его (рис. 3). Программа выводит правильный результат, значит, она написана корректно.

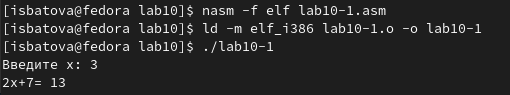


Рис. 3: Запуск программы из файла ‘lab10-1.asm’

Далее вновь открываем файл ‘lab10-1.asm’ и редактируем его, добавляя еще одну подпрограмму (для вычисления сложной функции). (рис. 4).

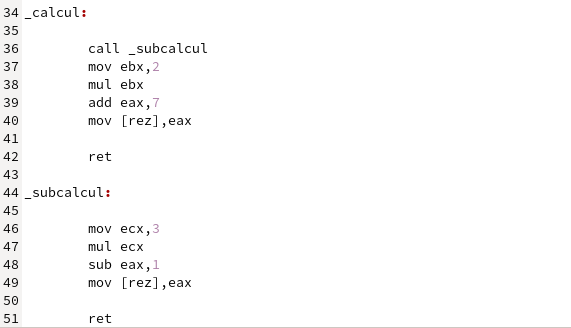


Рис. 4: Добавление еще одной подпрограммы

Создаем исполняемый файл и запускаем его (рис. 5). Программа выводит правильный результат, значит, она написана корректно.

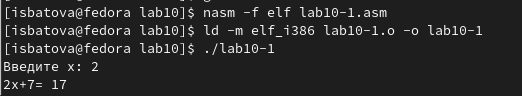


Рис. 5: Запуск измененной программы из файла ‘lab10-1.asm’

Для дальнейшей работы создаем файл ‘lab10-2.asm’ (рис. 6).

Рис. 6: Создание файла ‘lab10-2.asm’

Рис. 6: Создание файла ‘lab10-2.asm’

Открываем этот файл ‘lab10-2.asm’ и вводим листинг 10.2 из лабораторной работы (рис. 7).

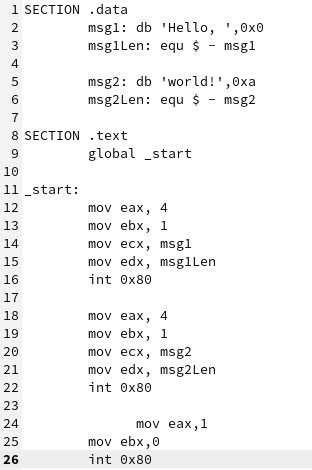


Рис. 7: Ввод листинга 10.2

Создаем исполняемый файл, причем трансляцию выполняем с ключом ‘-g’, чтобы мы могли с этим файлом работать в отладчике (рис. 8).

Рис. 8: Создание исполняемого файла 10-2

Рис. 8: Создание исполняемого файла 10-2

Загружаем файл в отладчик gdb (рис. 9).

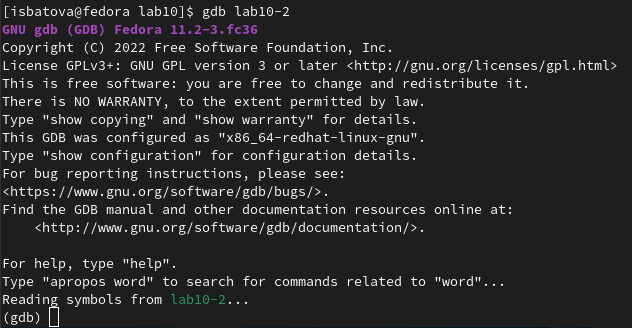


Рис. 9: Загрузка файла 10-2 в отладчик

Для проверки работы программы запускаем ее командой ‘run’ (рис. 10). Видим, что программа работает корректно.

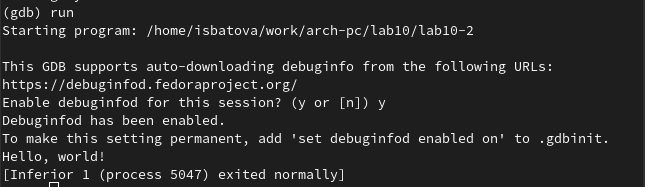


Рис. 10: Запуск программы 10-2 в отладчике

Далее устанавливаем брейкпоинт на метку старт и снова запускаем программу (рис. 11).

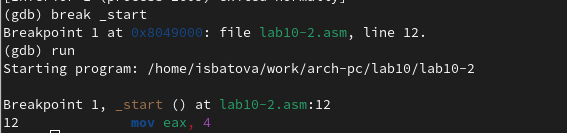


Рис. 11: Установка первого брейкпоинта

После этого вводим команду ’disassemble \_start’, чтобы посмотреть дисассимилированный код (рис. 12).

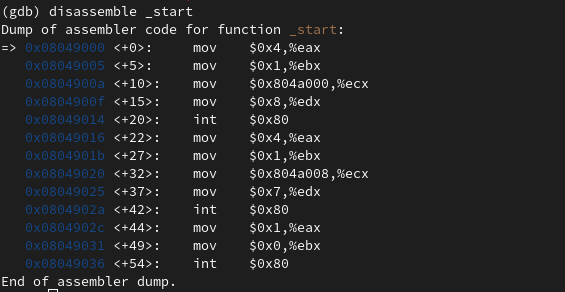


Рис. 12: Просмотр дисассимилированного кода

Далее мы переключаемся на отображение команд с intel’овским синтаксисом (команда ‘set disassembly-flavor intel’) и вновь просматриваем дисассимилированный код (рис. 13).

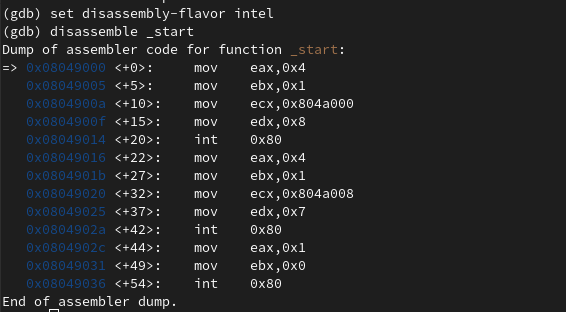


Рис. 13: Просмотр intel’овского отображения

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. 14).

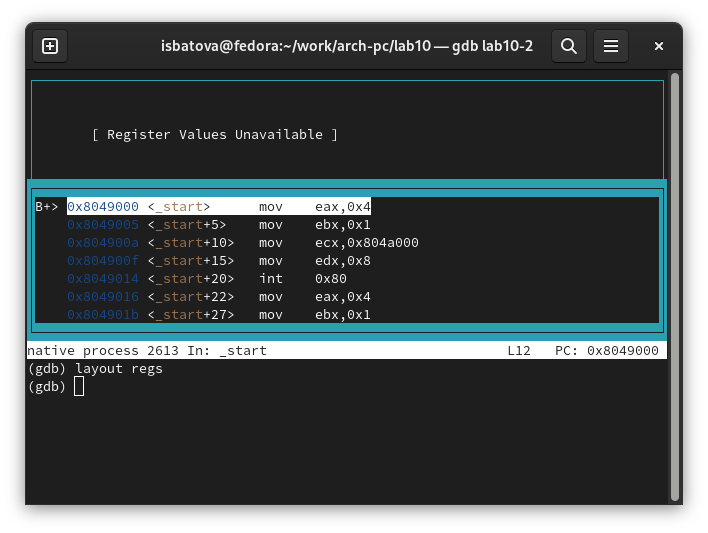


Рис. 14: Режим псевдографики

Вводим команду ‘info breakpoints’ для просмотра установленных нами точек останова. Видим одну точку останова, которую мы поставили выше (рис. 15).

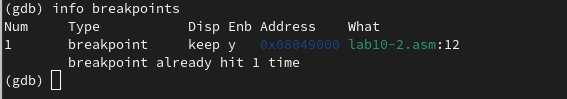


Рис. 15: Просмотр брейкпоинтов

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. 16).

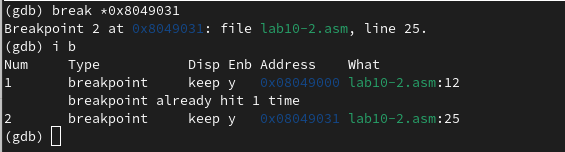


Рис. 16: Установка второго брейкпоинта

На этом моменте у нас данные значения регистров (рис. 17).

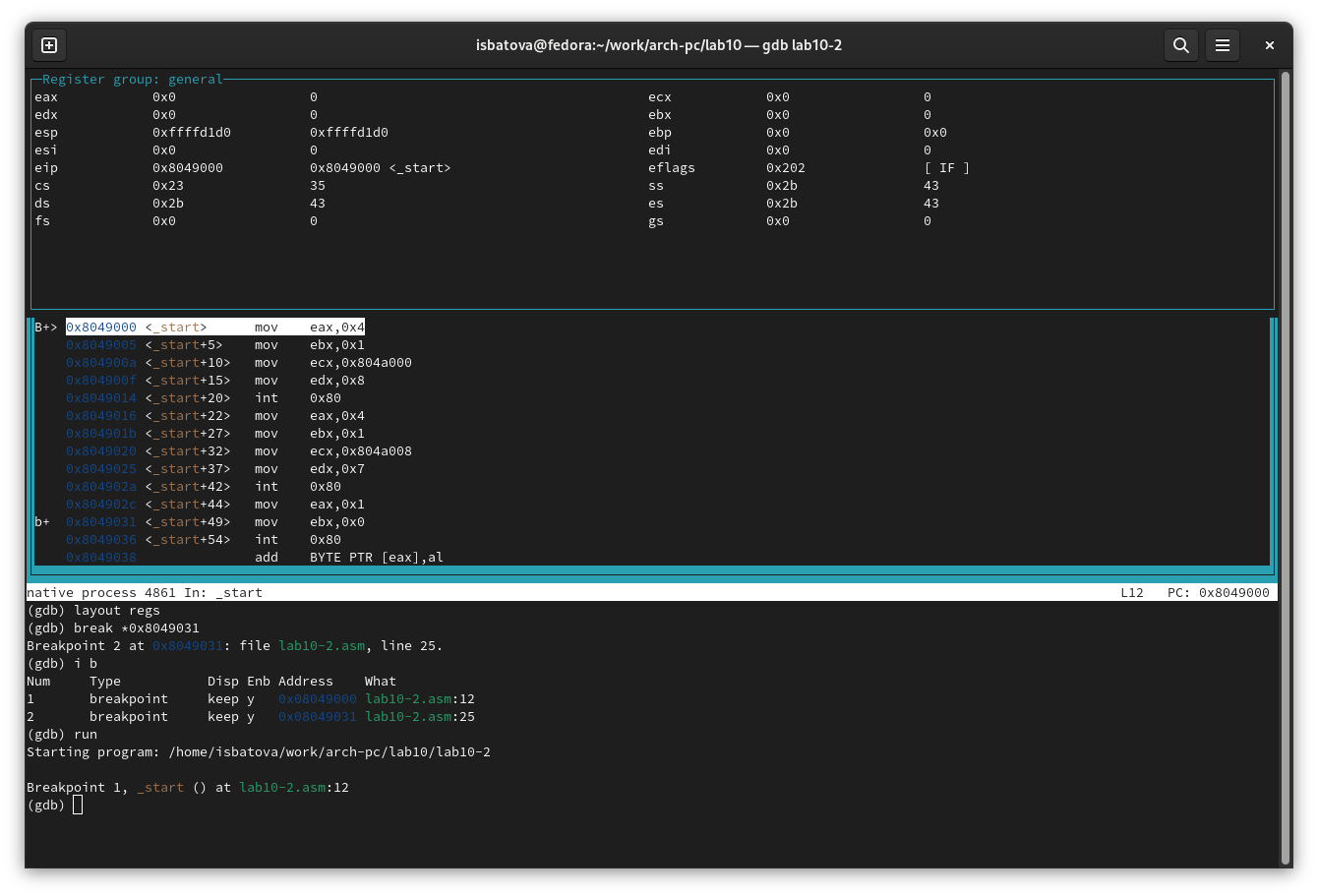


Рис. 17: Значения регистров

Далее вводим команду ‘si 5’ и видим, что значения регистров eax, edx и eip меняются (рис. 18).

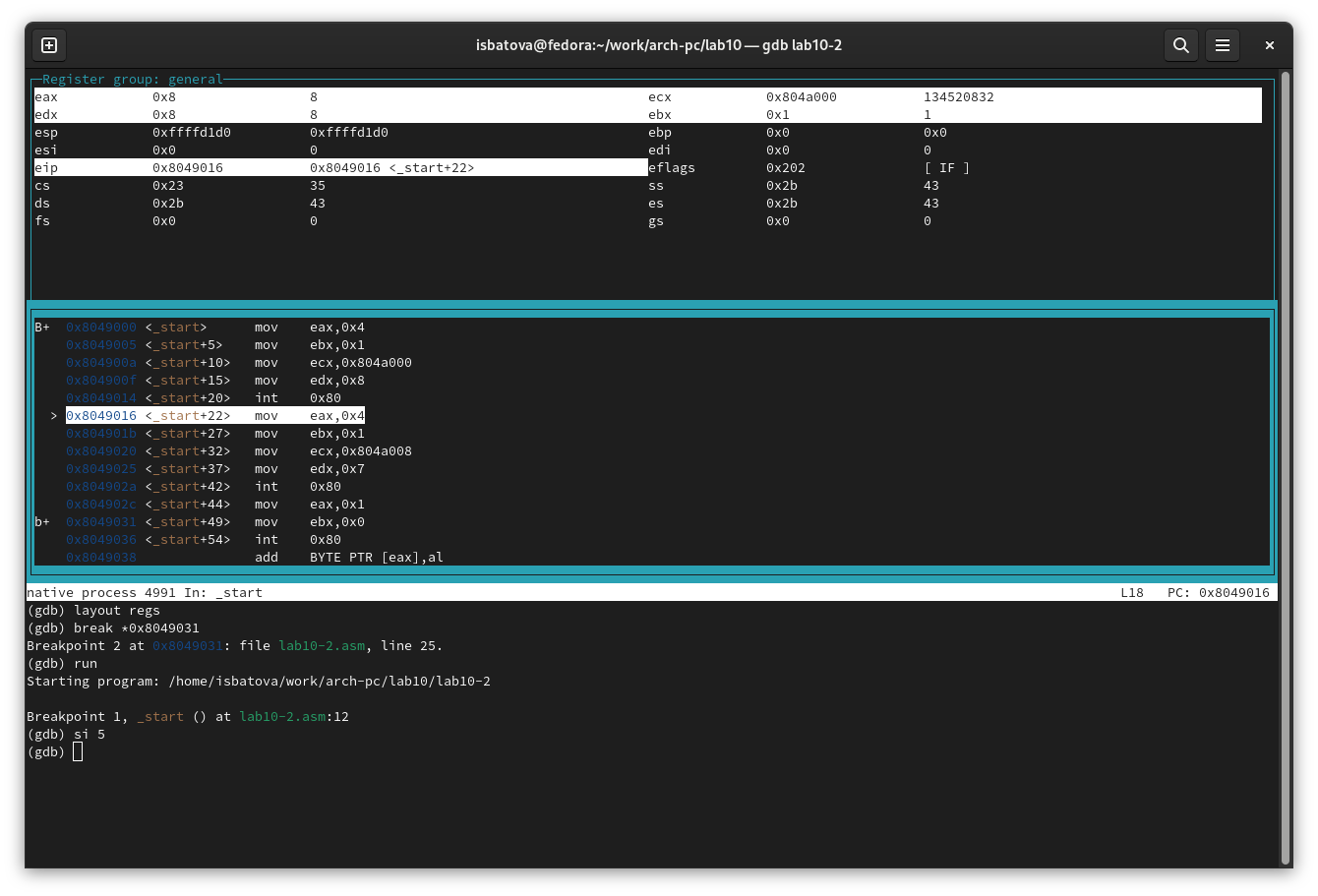


Рис. 18: Изменение значений регистров

Просматриваем значения регистров с помощью команды ‘info registers’ (рис. 19).

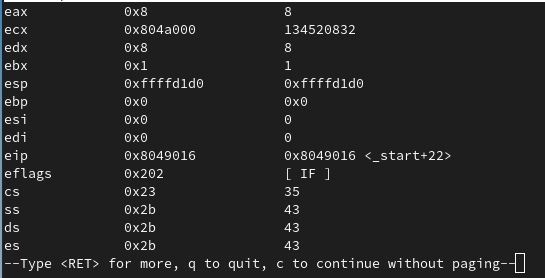


Рис. 19: Просмотр значений регистров

Далее мы смотрим содержимое переменной ‘msg1’ (по имени) (рис. 20).

Рис. 20: Содержимое переменной msg1

Рис. 20: Содержимое переменной msg1

После этого с помощью окна посередине определяем адрес переменной ‘msg2’ и смотрим ее содержимое по адресу (рис. 21).

Рис. 21: Содержимое переменной msg2

Рис. 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. 22).

Рис. 22: Новое содержимое переменной msg1

Рис. 22: Новое содержимое переменной msg1

Аналогично заменяем первый символ в регистре msg2 на ‘L’ (команда “set {char}&msg2=‘L’”) и проверяем это (рис. 23).

Рис. 23: Новое содержимое переменной msg2

Рис. 23: Новое содержимое переменной msg2

Далее мы рассматриваем команду вывода значения регистров. Выводим в трех различных форматах значение регистра edx (рис. 24).



Рис. 24: Значение регистра edx

После этого мы изменяем значение регистра ebx с помощью команды ‘set’ (рис. 25). Мы получаем разные значения, так как во втором случае мы приравниваем регистр к двойке, а в первом вносим значение 2 в регистра.



Рис. 25: Изменение значения регистра ebx

Далее мы завершаем работу в отладчике и выходим из него (команда ‘quit’) (рис. 26).

Рис. 26: Завершение работы в отладчике

Рис. 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. 27).

Рис. 27: Копирование файла

Рис. 27: Копирование файла

Создаем исполняемый файл (трансляцию выполняем с ключом ‘-g’ (рис. 28).

Рис. 28: Создание исполняемого файла 10-3

Рис. 28: Создание исполняемого файла 10-3

После этого запускаем файл в отладчике с указанием аргументов (рис. 29).

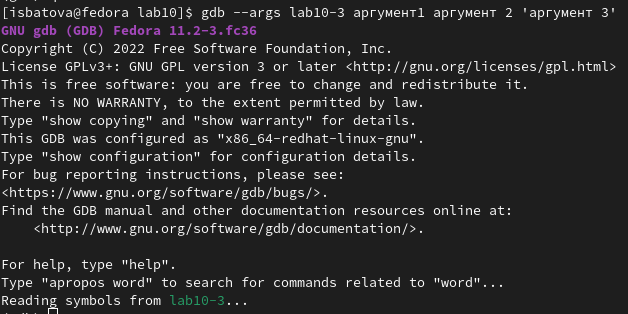


Рис. 29: Запуск файла 10-3 в отладчике

Далее устанавливаем брейкпоинт на метку старт и запускаем программу (рис. 30).

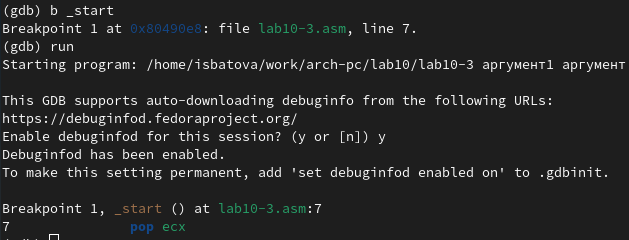


Рис. 30: Установка брейкпоинта и запуск

Проверяем адрес вершины стека и наличия там 5 позиций (рис. 31).

Рис. 31: Адрес вершины стека

Рис. 31: Адрес вершины стека

После этого просматриваем остальные позиции стека (рис. 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.

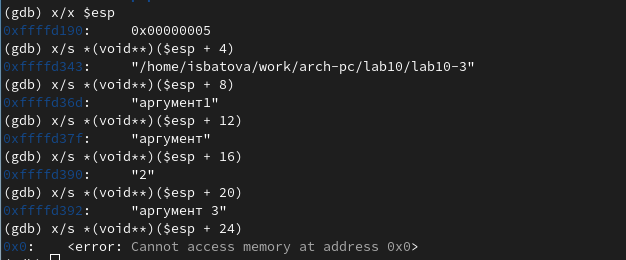


Рис. 32: Остальные позиции стека

# 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. 33).

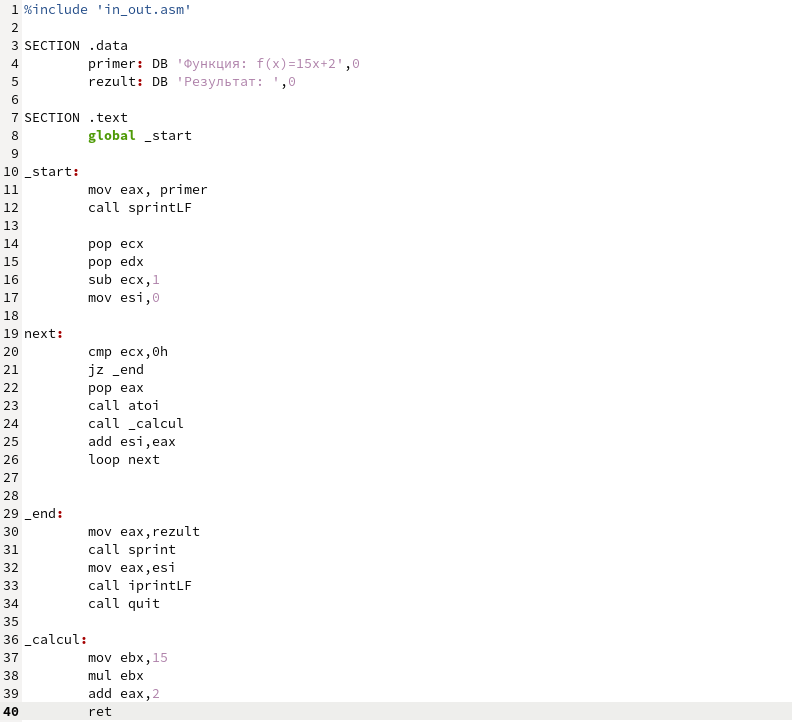


Рис. 33: Редактирование программы

Сохраняем изменения, создаем исполняемый файл и запускаем его (рис. 34). Если посчитать аналитически, получается такой же ответ, поэтому программа работает корректно.

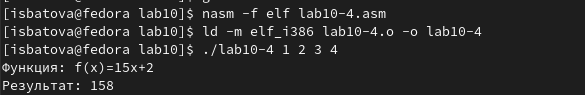


Рис. 34: Запуск программы из файла ‘lab10-4.asm’

1. Для начала выполнения второго задания создаем файл ‘lab10-5.asm’ и вводим в него листинг 10.3 - вычисление выражения (3+2)\*4+5 (рис. 35).

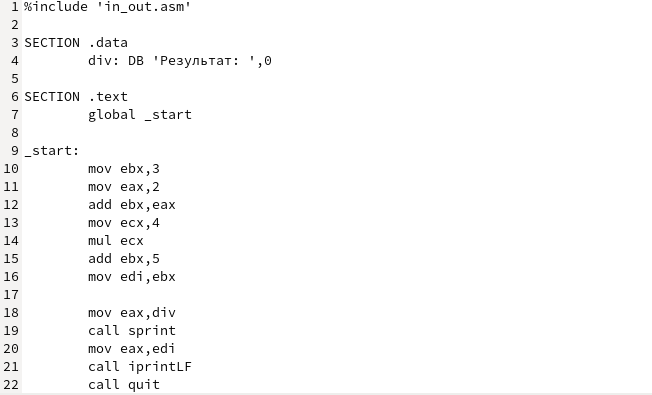


Рис. 35: Ввод листинга 10.3

Cоздаем исполняемый файл и запускаем его (рис. 36). Если посчитать аналитически, ответ должен быть 25, а программа выдает 10.

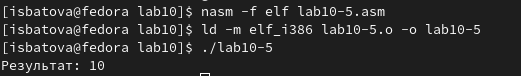


Рис. 36: Запуск программы из файла ‘lab10-4.asm’

Для поиска ошибки создаем исполняемый файл с ключом ‘-g’ и загружаем файл в отладчик (рис. 37).

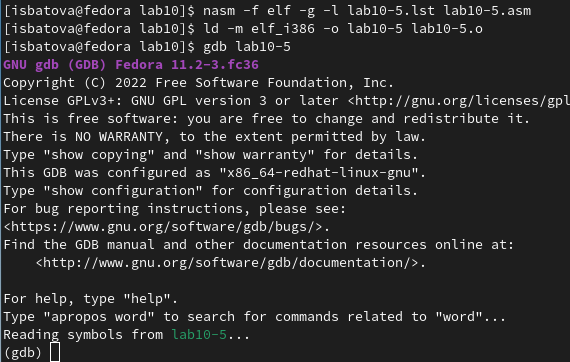


Рис. 37: Загрузка программы 10-5 в отладчик

Устанавливаем точку останова на метку start (рис. 38) и дисассемилируем код (с intel’овским изображением (рис. 39).

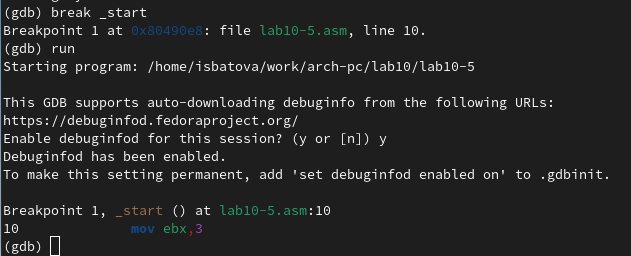


Рис. 38: Установка точки останова на метку start

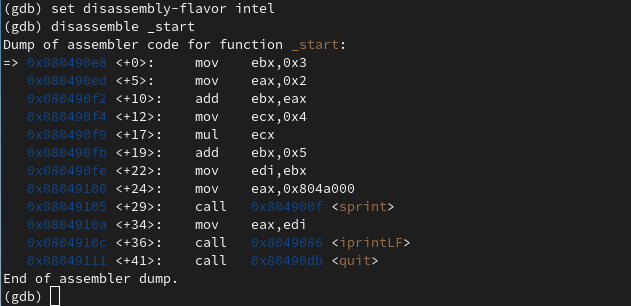


Рис. 39: Получение intel’овского отображения

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. 40).



Рис. 40: Исправление ошибки

Далее вводим команду ‘continue’, чтобы наша программа была выполнена до конца и получаем верный результат (рис. 41).

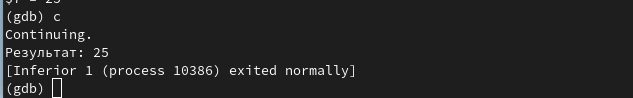


Рис. 41: Получение верного результата

# 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.