Отчет по лабораторной работе №9

Дисциплина: Архитектура компьютера

Мутаев Муртазаали Магомедович

Содержание

# 1 Цель работы

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

# 2 Задание

1. Реализация подпрограмм в NASM
2. Отладка программ с помощью GDB
   1. Работа с данными программы в GDB
   2. Обработка аргументов командной строки в GDB
3. Задания для самостоятельной работы

# 3 Выполнение лабораторной работы

## 3.1 Реализация подпрограмм в NASM

По базе сначала создаем файл lab9-1.asm в каталоге work/arch-pc/lab09

В качестве примера рассмотрим программу вычисления арифметического выражения *f(x) = 2x + 7* с помощью подпрограммы \_calcul. В данном примере x вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Воспользуемся кодом из Листинга 9.1 (рис. 1):



Рис. 1: Листинг 9.1

Вот такой результат у меня получился (рис. 2):

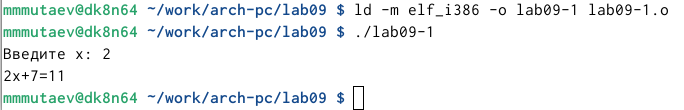


Рис. 2: Результат Листинга 9.1

Попробуем изменить текст программы, добавив подпрограмму \_subcalcul в подпрограмму \_calcul, для вычисления выражения f(g(x)), где x вводится с клавиатуры, f(x) = 2x + 7, g(x) = 3x − 1. Т.е. x передается в подпрограмму \_calcul из нее в подпрограмму \_subcalcul, где вычисляется выражение g(x), результат возвращается в \_calcul и вычисляется выражение f(g(x)). Результат возвращается в основную программу для вывода результата на экран.

Вот такая программа у меня получилась (рис. 3):

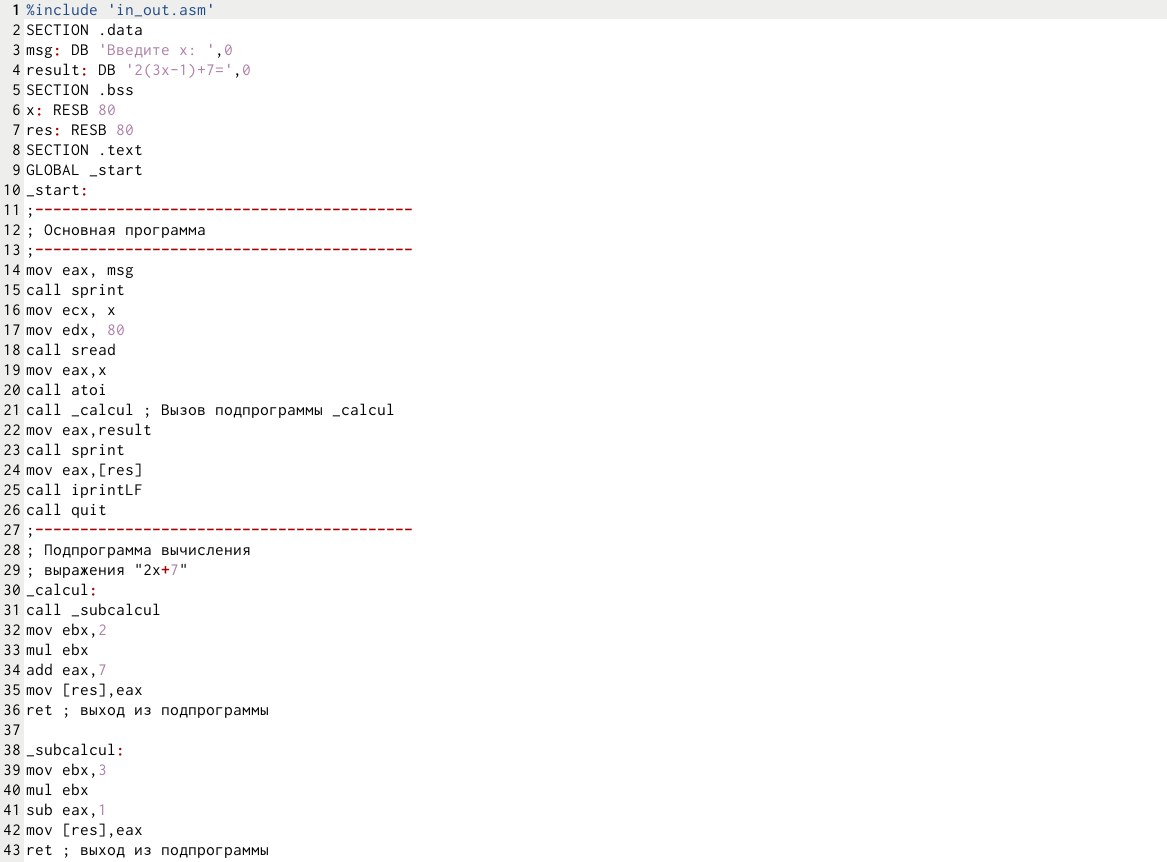


Рис. 3: Измененный Листинг 9.1

У меня получился такой результат (рис. 4):

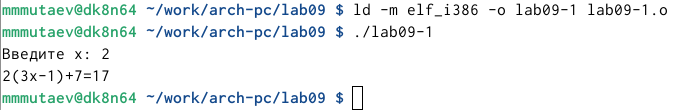


Рис. 4: Результат измененного Листинга 9.1

Как мы видим, результат получился верным, значит программа написана правильно.

## 3.2 Отладка программ с помощью GDB

Воспользуемся кодом из Листинга 9.2 (программы для вывода Hello world!) (рис. 5):

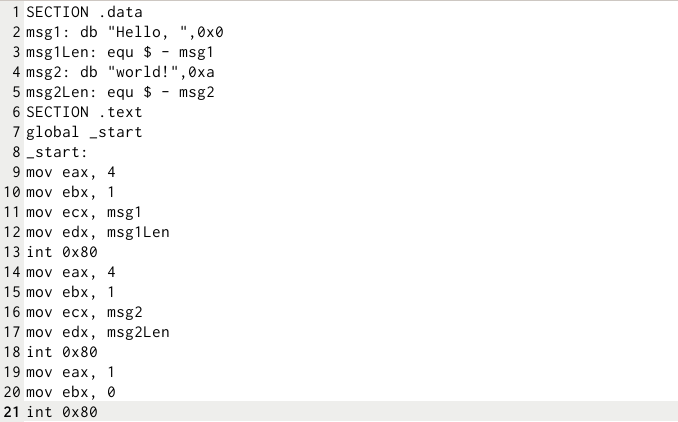


Рис. 5: Листинг 9.2

Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’. После этого я загрузил исполняемый файл в отладчик gdb и запустил его (рис. 6):

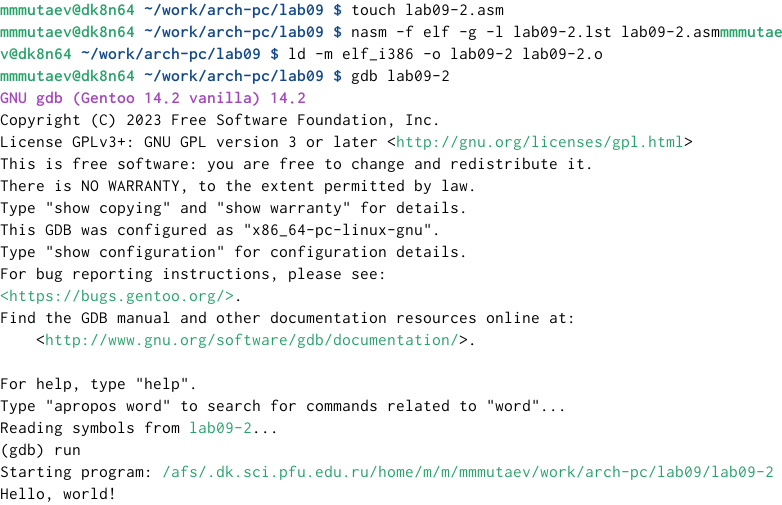


Рис. 6: Отладчик gdb

Для более подробного анализа программы установим брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запустим её (рис. 7):

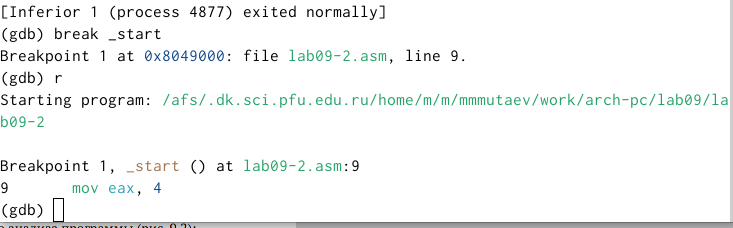


Рис. 7: Breakpoint

Далее просмотрим дисассимилированный код программы с помощью команды disassemble начиная с метки \_start (рис. 8):



Рис. 8: Дисассимпилированный код ATT

Переключимся на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel (рис. 9):

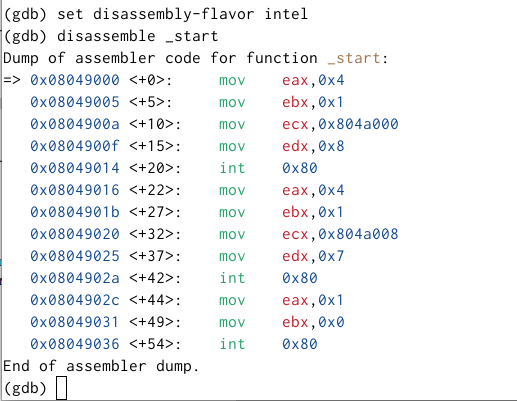


Рис. 9: Дисассимпилированный код Intel

Есть некоторые различия в отображениях в этих режимах, а именно в виде колонки с текстом программы: в ATT’e она выглядит, как *“$0x{операнд},%{регистер}”*, а в Intel - *“{регистер},0х{операнд}”*

Теперь включим режим псевдографики для более удобного анализа программы:

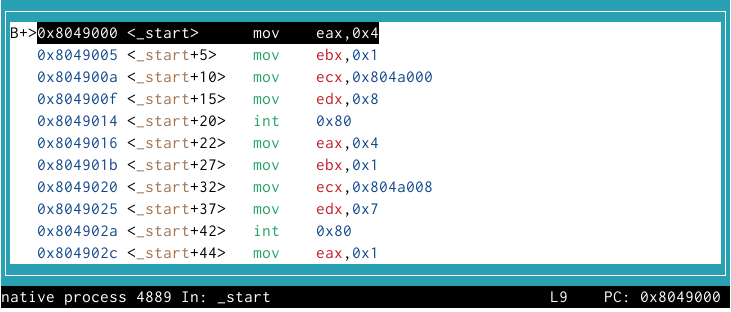


Рис. 10: Псевдографика asm

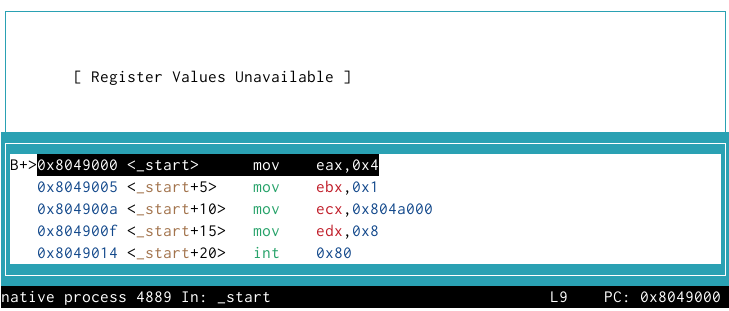


Рис. 11: Псевдографика regs

Установить точку останова можно командой break (кратко b). Типичный аргумент этой команды — место установки. Его можно задать или как номер строки программы (имеет смысл, если есть исходный файл, а программа компилировалась с информацией об отладке), или как имя метки, или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка».

На предыдущих шагах была установлена точка останова по имени метки (\_start). Проверим это с помощью команды info breakpoints (кратко i b) (рис. 12):

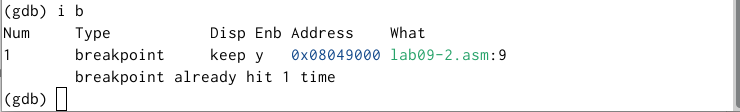


Рис. 12: Точки останова

Установим еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции

Определим адрес предпоследней инструкции (mov ebx,0x0) и установим точку останова (рис. 13):

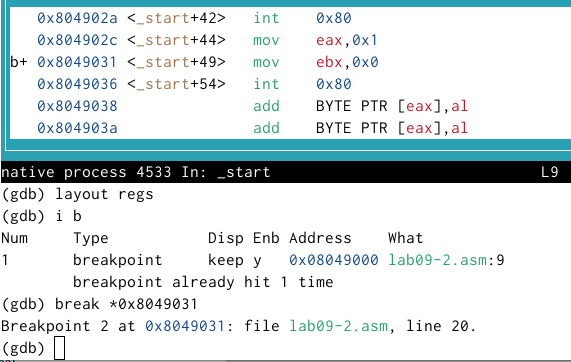


Рис. 13: Установка нового брейкпоинта

Адрес инструкции находился в средней части экрана в левом столбце соответствующей инструкции. Можно заметить, что слева от адреса появился значок *b+*. Вероятно он означает, что здесь поставлен брейкпоинт.

Теперь снова посмотрим информацию о всех установленных точках останова (рис. 14):



Рис. 14: Информация о брейкпоинтах

Как мы видим, информация обновилась.

### 3.2.1 Работа с данными программы в GDB

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Посмотрим содержимое регистров также можно с помощью команды info registers (рис. 15):

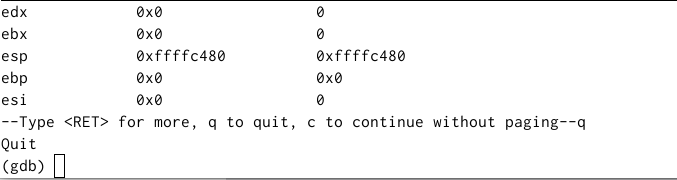


Рис. 15: Содержимое регистров

Для отображения содержимого памяти можно использовать команду x , которая выдаёт содержимое ячейки памяти по указанному адресу. Формат, в котором выводятся данные, можно задать после имени команды через косую черту: x/NFU . С помощью команды x & также можно посмотреть содержимое переменной. Посмотрим значение перменной msg1 по имени (рис. 16):



Рис. 16: Значение переменной msg1

Посмотрим значение переменной msg2 по адресу. Адрес переменной можно определить по дизассемблированной инструкции. Посмотрите инструкцию mov ecx,msg2 которая записывает в регистр ecx адрес перемененной msg2 (рис. 17):

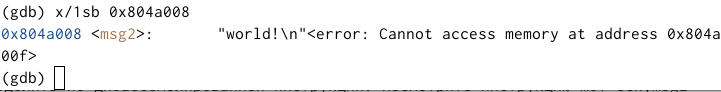


Рис. 17: Значение переменной msg2

Изменить значение для регистра или ячейки памяти можно с помощью команды set, задав ей в качестве аргумента имя регистра или адрес. При этом перед именем регистра ставится префикс $, а перед адресом нужно указать в фигурных скобках тип данных (размер сохраняемого значения; в качестве типа данных можно использовать типы языка Си).

Изменим первый символ переменной msg1 (рис. 18):



Рис. 18: Изменение символа msg1

Заменим любой символ в переменной msg2 (рис. 19):

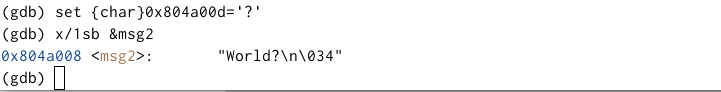


Рис. 19: Изменение символа msg2

Поздравляю, теперь наша программа задает вопрос “hello, World?”

Далее изменим значение регистра ebx и выведемОбработка аргументов командной строки в GDB его значение с помощью print (рис. 20):

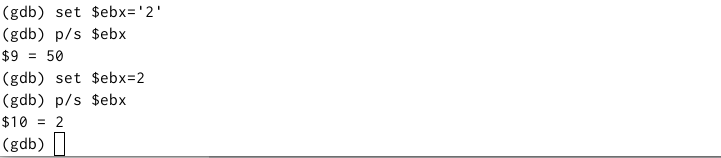


Рис. 20: Print

Можно заметить разницу, в зависимости от введенного значения. В первом случае мы в ebx записываем символ “2”, поэтому принт выводит номер этого символа в таблице ASCII, а во втором случае мы присваеваем ebx значение 2, поэтому он выводит 2.

### 3.2.2 Обработка аргументов командной строки в GDB

Скопируем файл lab8-2.asm в нашу папку и назовем ее lab09-3.asm. Создадим исполняемый файл и загрузим в gdb программу. Для загрузки в gdb программы с аргументами необходимо использовать ключ –args (рис. 21):

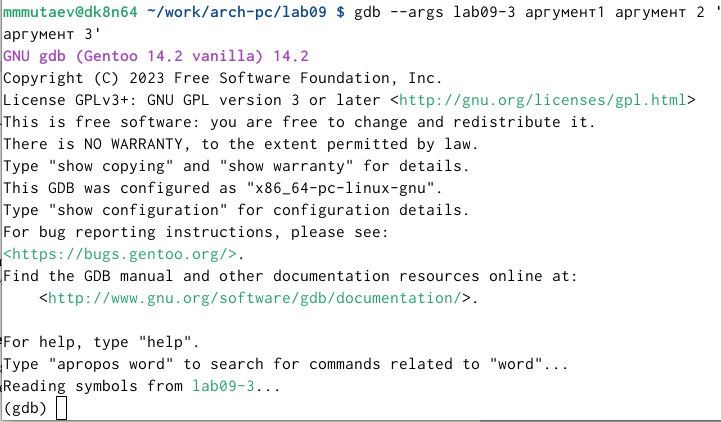


Рис. 21: Загрузка программы с аргументами в GDB

Как отмечалось в предыдущей лабораторной работе, при запуске программы аргументы командной строки загружаются в стек. Исследуем расположение аргументов командной строки в стеке после запуска программы с помощью gdb. Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (рис. 22)

Как видно, число аргументов равно 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’. Посмотрим остальные позиции стека – по адесу [esp+4] располагается адрес в памяти где находиться имя программы, по адесу [esp+8] храниться адрес первого аргумента, по адресу [esp+12] – второго и т.д



Рис. 22: Изучение стека

Предполагаю, что шаг изменения адреса равен 4, потому что на каждый аргумент выделено 4 байта.

## 3.3 Задания для самостоятельной работы

1. Преобразуйте программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму.

Легко! Просто переносим часть с нахождением значения f(x) в подпрограмму. Вот код, который у меня получился (рис. 23):

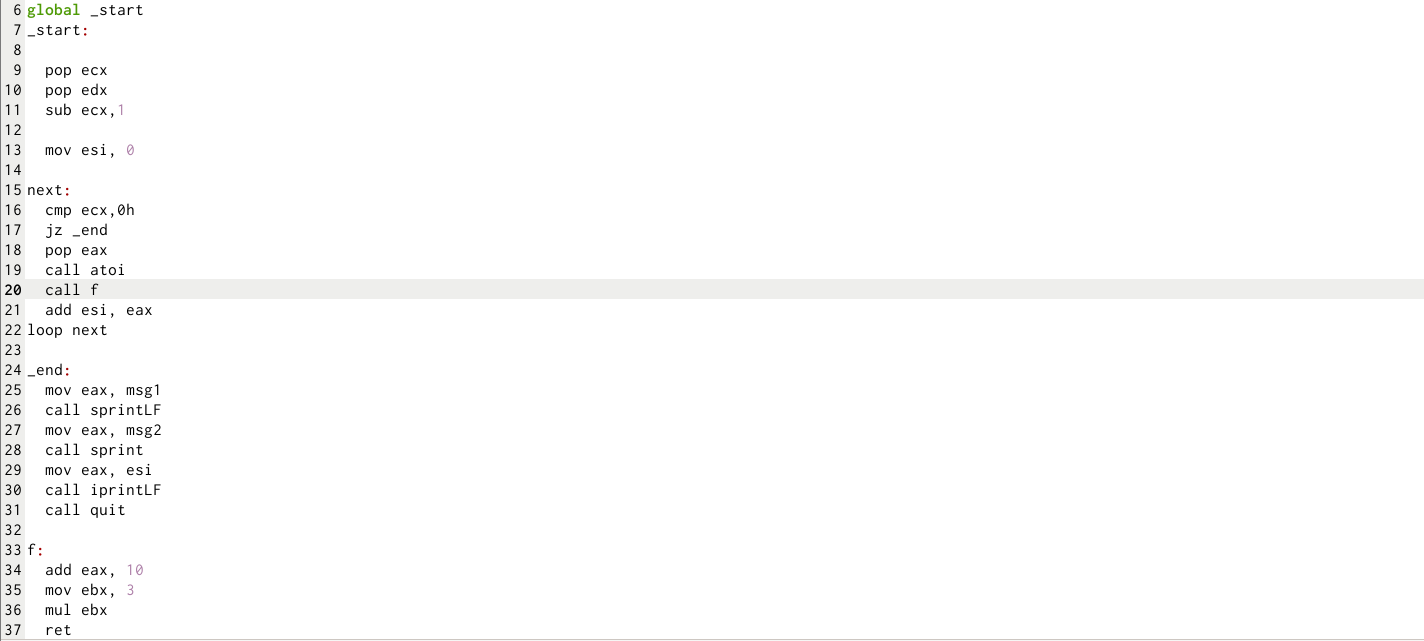


Рис. 23: Самостоятельная работа 1

А вот результат (рис. 24):

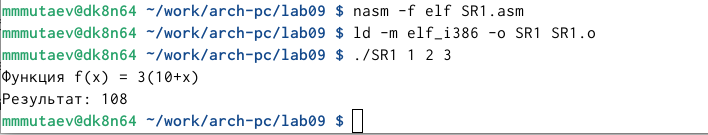


Рис. 24: Результат самостоятельная работа 1

1. В листинге 9.3 приведена программа вычисления выражения (3 + 2) ∗ 4 + 5. При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее.

Ответ действительно неправильный: программа должна выводить 25, а выводит 10

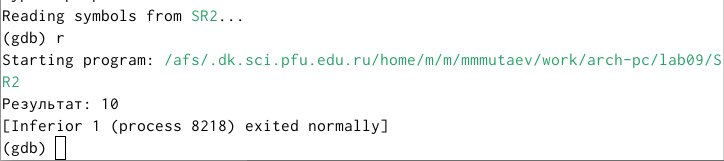


Рис. 25: Работа листинга 9.3

Так как в результат записывается значение регистра ebx, отследим его изменение и изменение других регистров. Изначально ebx = 3

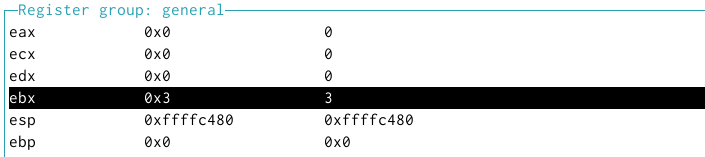


Рис. 26: Step 1

Далее мы прибавили к ebx eax, и у нас ebx = 5

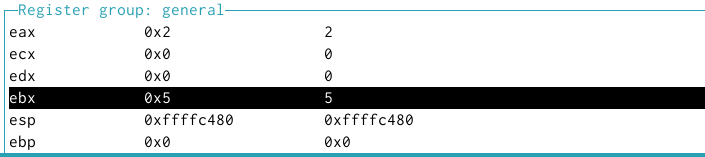


Рис. 27: Step 2

Далее мы присвоили ecx значение 4 и умножили eax на ecx. Возможно, в этом и есть ошибка

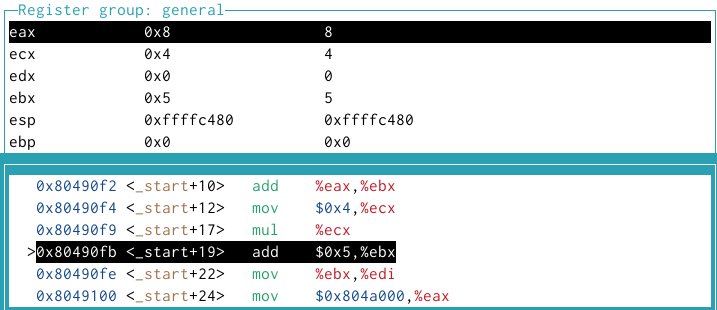


Рис. 28: Step 3

Далее к ebx прибавили 5 (теперь ebx = 10) и записали его значение в edi

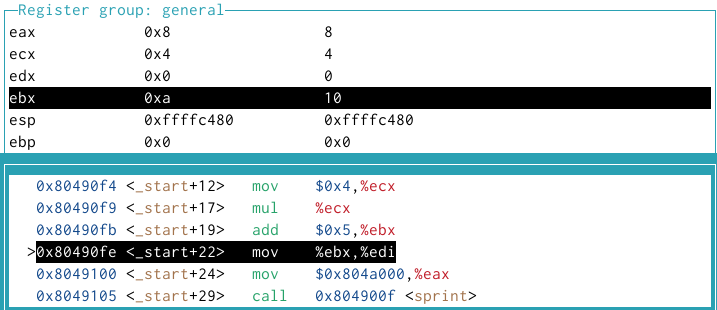


Рис. 29: Step 4

Пошагово проанализировав изменение значений регистров, я выяснил, что ошибкой является то, что мы умножили eax на 4, а должны были ebx. Попытаемся исправить эту ошибку. Вот такая программа у меня получилась (рис. 30):

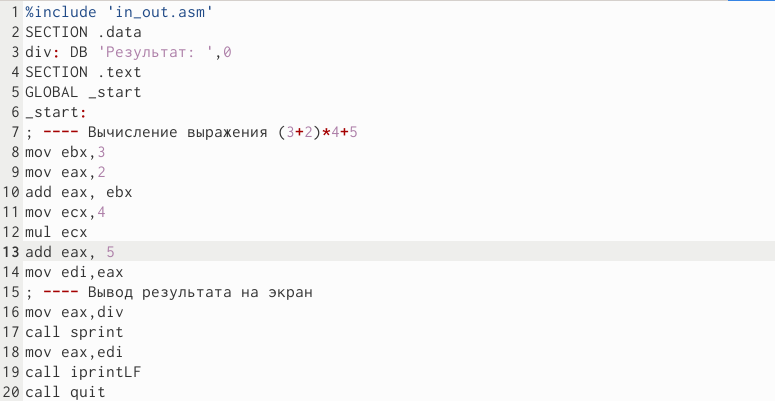


Рис. 30: Самостоятельная работа 2

И вот такой результат (рис. 31):

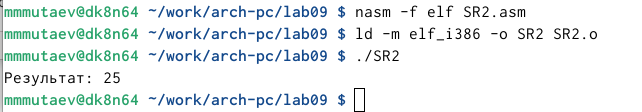


Рис. 31: Результат самостоятельная работа 2

# 4 Выводы

Я приобрел навыки написания программ с использованием подпрограмм и познакомился с методами отладки при помощи GDB и его основными возможностями.