Отчёт по лабораторной работе №10

Понятие подпрограммы. Отладчик GDB.

Михаил Александрович Мелкомуков

Содержание

# 1 Цель работы

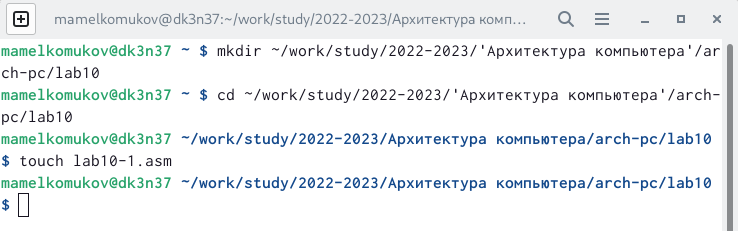
Целью работы является приобретение навыков написания программ с использованием подпрограмм а также знакомство с методами отладки при помощи GDB и его основными возможностями.

# 2 Задание

Узнать понятие подпрограммы, познакомиться с инструкциями call и ret. Ознакомиться с основными возможностями отладчика GDB: как запускать отладчик и выходить из него, как дизассемблировать программы, как устанавливать точки останова, как осуществлять пошаговую отладку, как работать с данными программы в GDB. На практике применить полученные знания.

# 3 Выполнение лабораторной работы

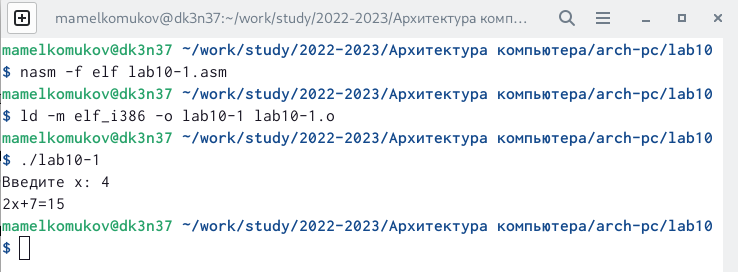
## 3.1 Реализация подпрограмм в NASM



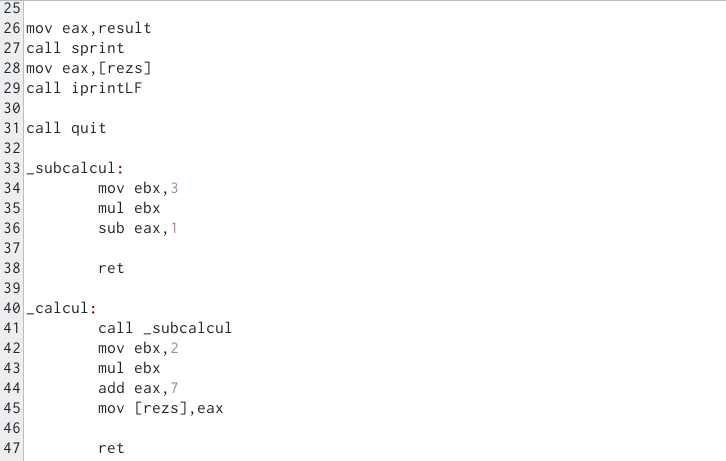
Создали каталог для выполнения лабораторной работы №10, перешли в него и создали файл lab10-1.asm



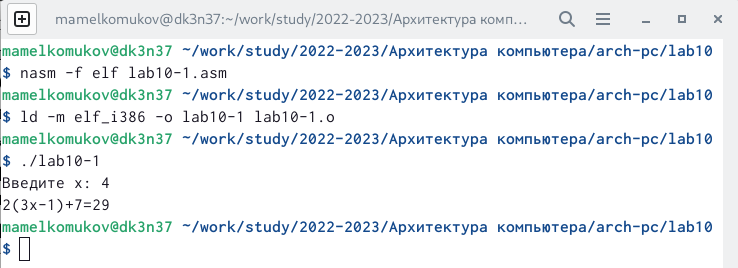
Ввели в файл lab10-1.asm текст программы



Создали исполняемый файл и проверили его работу

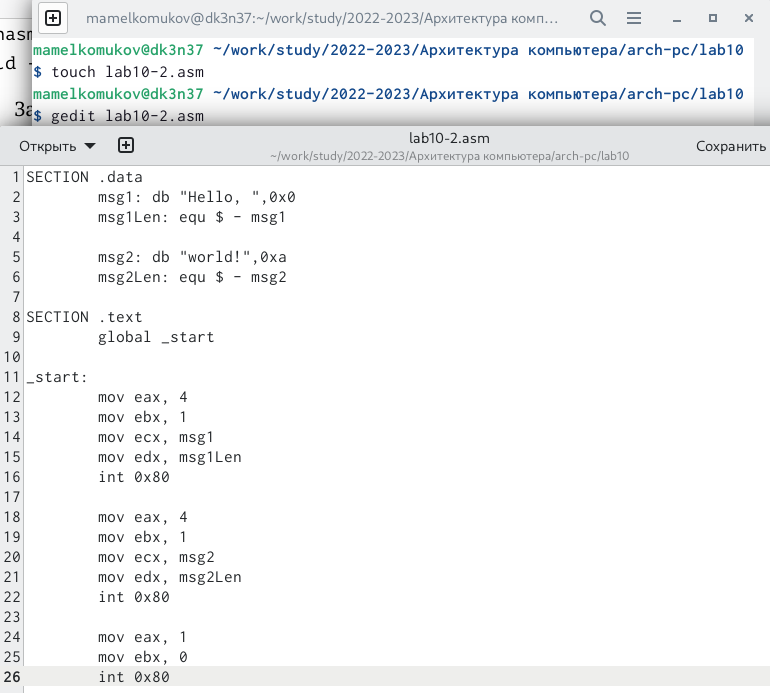


Изменили текст программы, добавив подпрограмму \_subcalcul в подпрограмму \_calcul

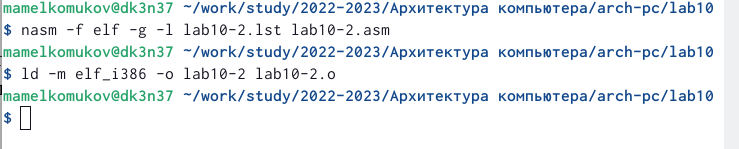


Создали исполняемый файл и проверили его работу

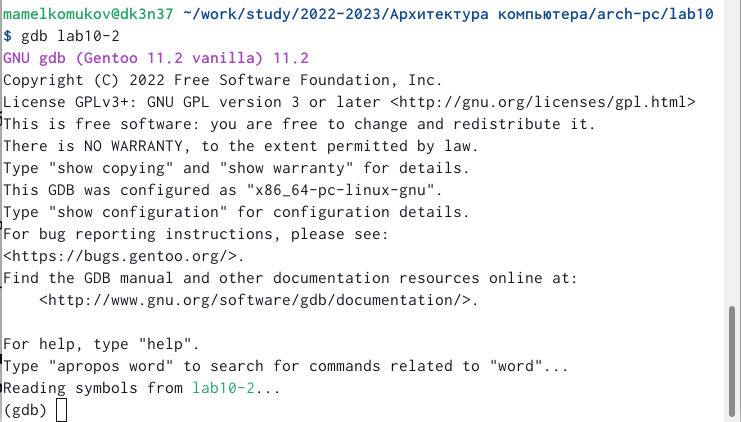
## 3.2 Отладка программ с помощью GDB



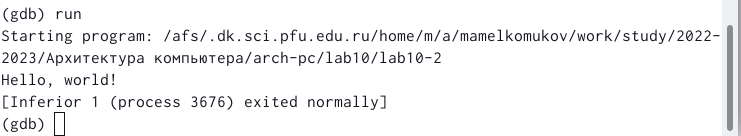
Создали файл lab10-2.asm и ввели в него текст программы



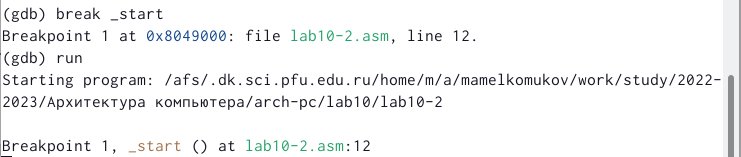
Получили исполняемый файл. Чтобы работать с GDB в исполняемый файл добавили отладочную информацию, для этого трансляцию программ провели с ключом ‘-g’



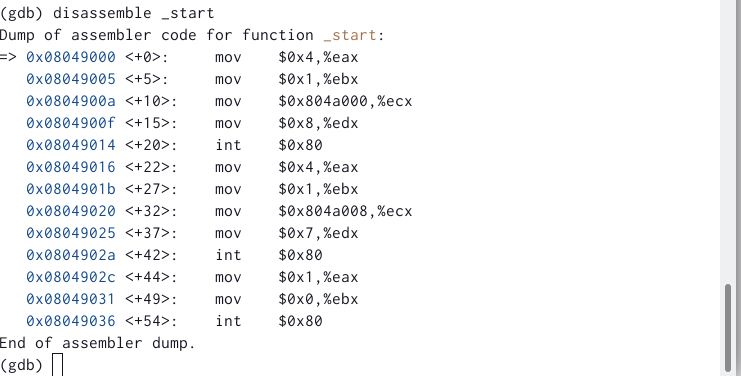
Загрузили исполняемый файл в отладчик gdb



Проверили работу программы, запустив ее в оболочке GDB с помощью команды run

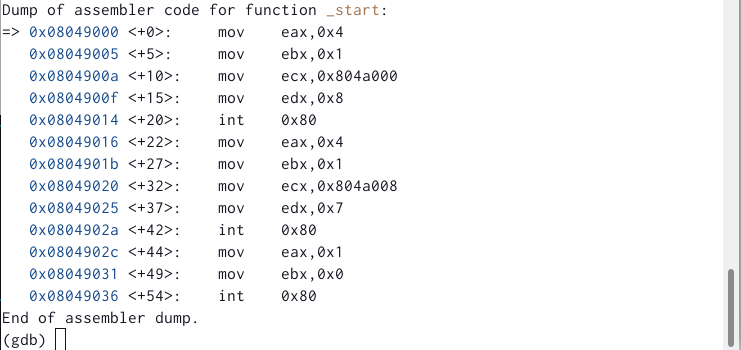


Для более подробного анализа программы установили брейкпоинт на метку \_start и запустили её



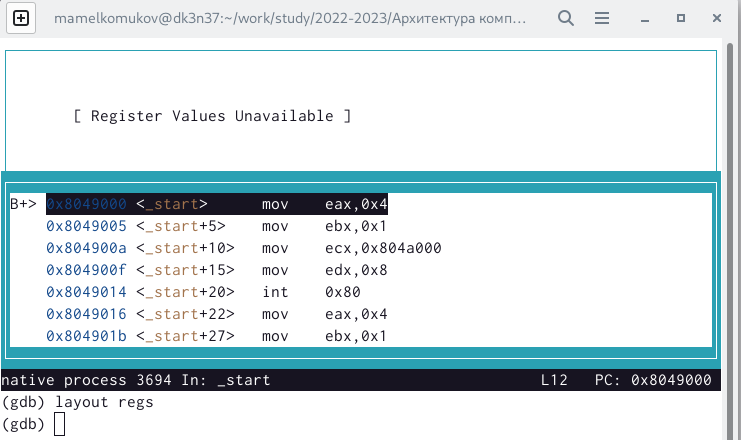
Посмотрели дисассимилированный код программы с помощью команды disassemble

Переключились на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel, после чего снова посмотрели дисассимилированный код программы с помощью команды disassemble.



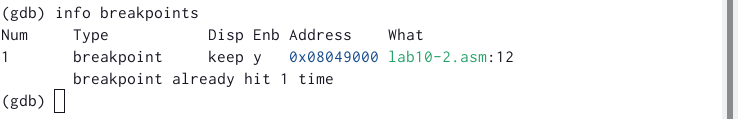
На этот раз дисассимилированный код выглядит следующим образом

В режиме АТТ непосредственные операнды пишутся после ‘$’, в то время как в режиме Intel операнды не выделяются. Регистровые операнды АТТ пишутся после ‘%’, регистровые операнды Intel не выделяются.



Включили режим псевдографики для более удобного анализа программы. Для этого ввели команды: layout asm и layout regs

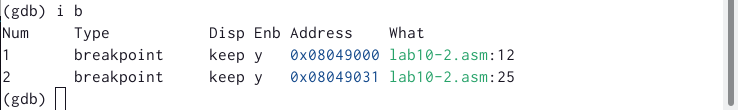
## 3.3 Добавление точек останова



С помощью команды info breakpoints проверили, что на одном из предыдущих шагов была установлена точка останова



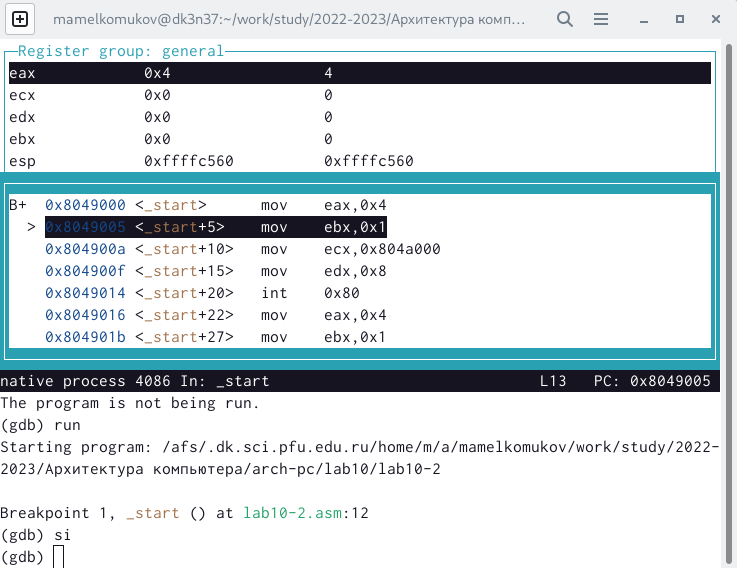
Установили еще одну точку останова по адресу инструкции. Для этого перед адресом добавляем ’\*’



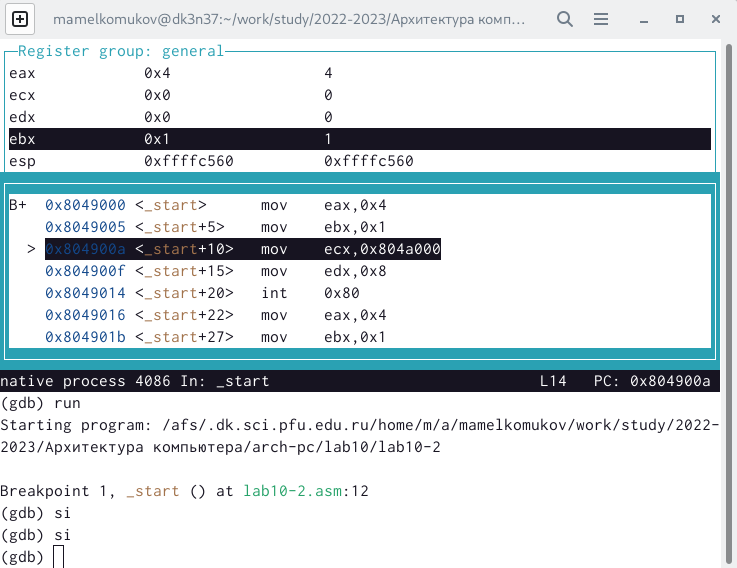
Посмотрели информацию о всех установленных точках останова с помощью команды i b (info breakpoints

## 3.4 Работа с данными программы в GDB

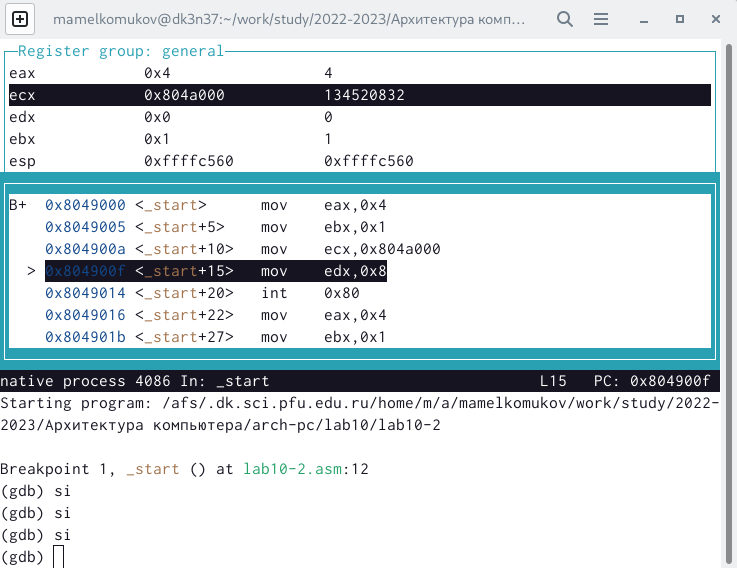
Выполнили 5 инструкций с помощью команды si (stepi) и проследили за изменением значений регистров.



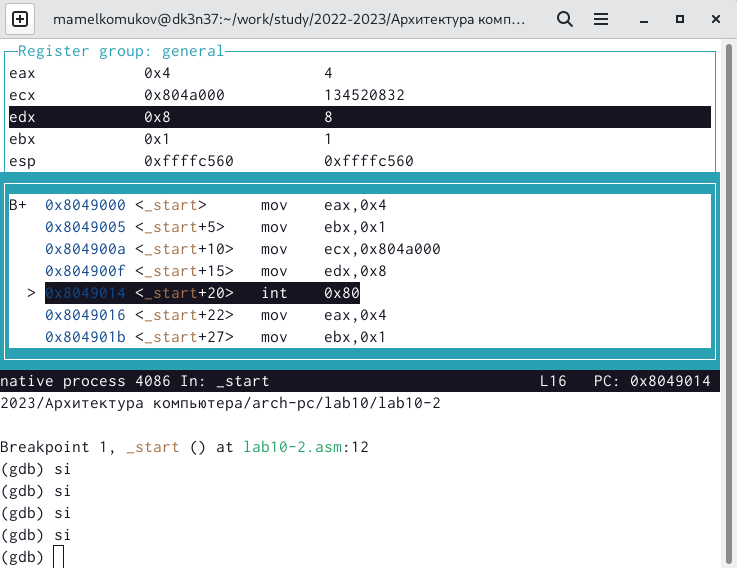
eax присваивается значение 4



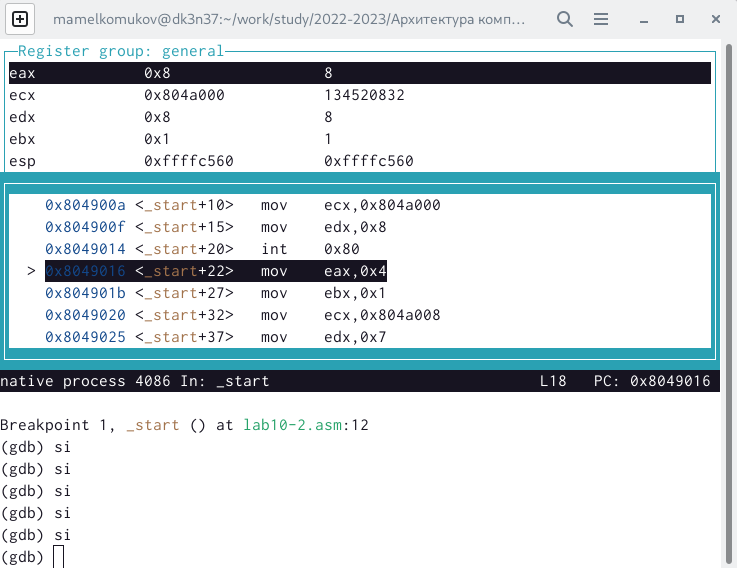
ebx присваивается значение 1



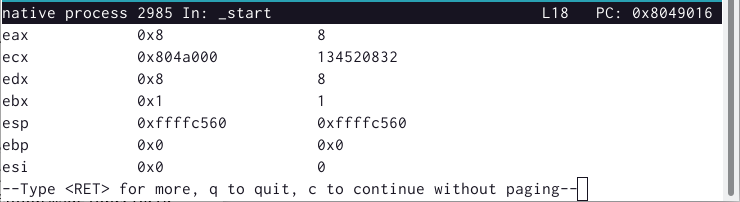
ecx присваивается значение 134520832



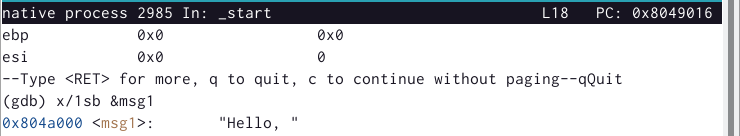
edx присваивается значение 8



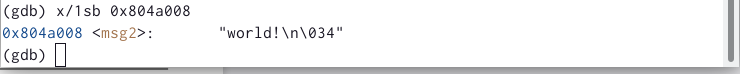
eax увеличивается на 4



Посмотрели содержимое регистров с помощью команды info registers (коротко i r)



С помощью команды x/1sb &msg1 посмотрели значение переменной msg1 по имени



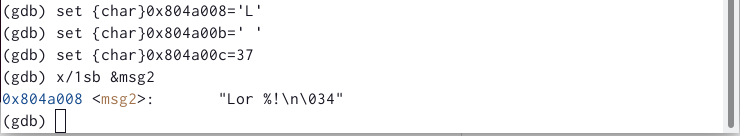
Посмотрели значение переменной msg2 по адресу. Адрес переменной определили по дизассемблированной инструкции

Изменили первый символ переменной msg1 с помощью команды set. Для этого задали ей в качестве аргумента имя регистра. Перед именем регистра поставили префикс $, а перед адресом в фигурных скобках указали тип данных.



С помощью команды x/1sb &msg1 посмотрели значение переменной msg1

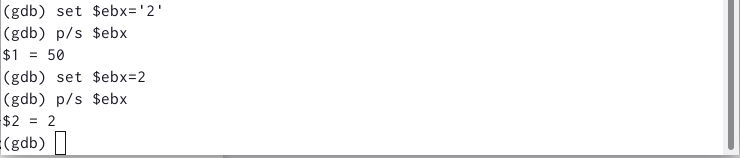
Заменили три символа во второй переменной msg2: первый на ‘L’, четвёртый на ’ ’ и пятый на ‘%’.



Результат изменений переменной msg2



Вывели в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx. Для этого использовали команды p/x, p/t и p/s соответственно. Перед именем регистра ставим префикс ‘$’

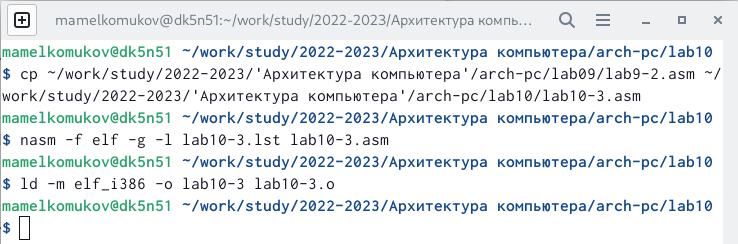


С помощью команды set изменили значение регистра ebx

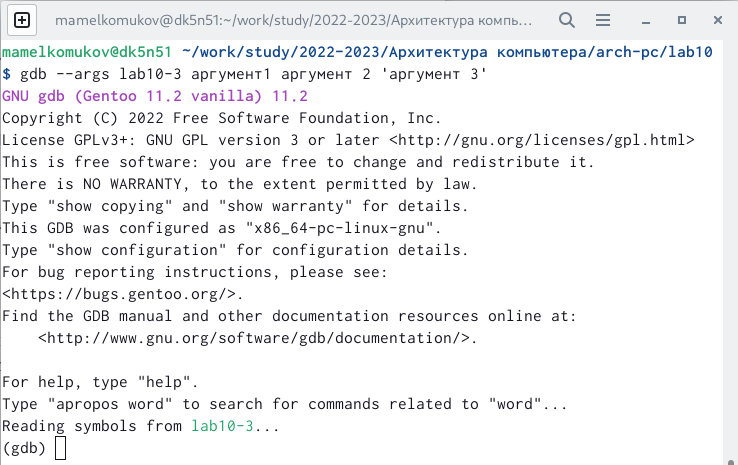
Разница вывода команд p/s $ebx заключается в следующем:

С помощью команды continue (сокращённо c) завершили выполнение программы и вышли из GDB с помощью команды quit (сокращенно q).

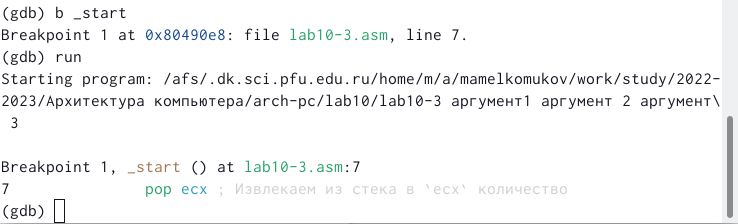
## 3.5 Обработка аргументов командной строки в GDB



Скопировали файл lab9-2.asm, созданный при выполнении лабораторной работы №9 в файл с именем lab10-3.asm и создали исполняемый файл



Для загрузки в gdb программы с аргументами использовали ключ –args. Загрузили исполняемый файл в отладчик и указали аргументы

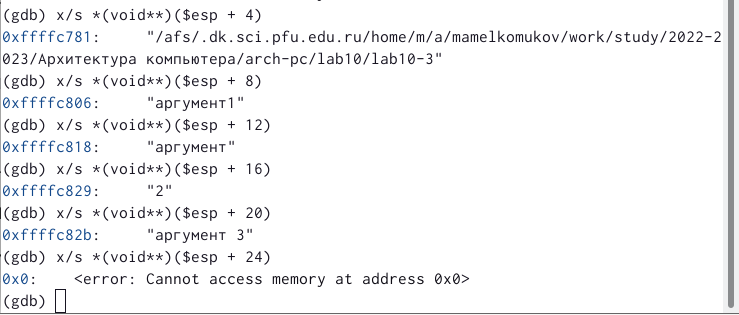


Установили точку останова перед первой инструкцией в программе и запустили ее



Посмотрели на число аргументов командной строки

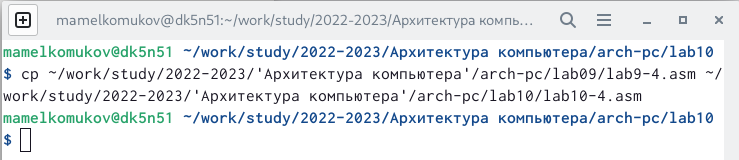
Число аргументов равно 5 – это имя программы lab10-3 и аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’.



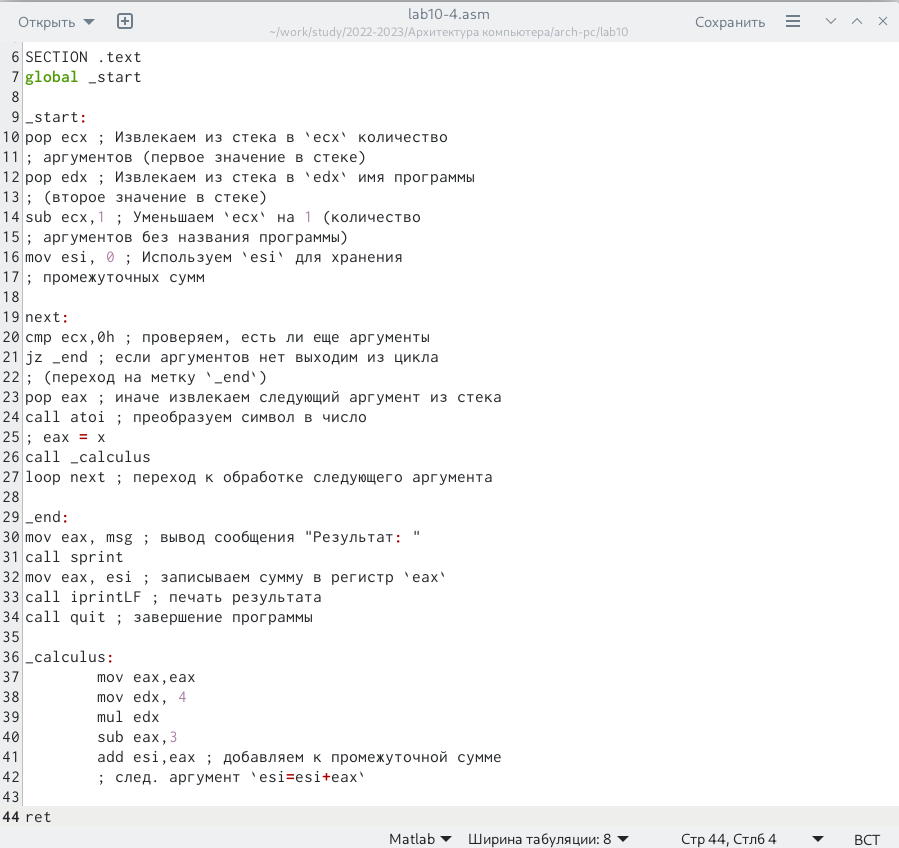
Посмотрели остальные позиции стека

Их адреса располагаются в четырёх байтах друг от друга, так как именно столько занимает элемент стека.

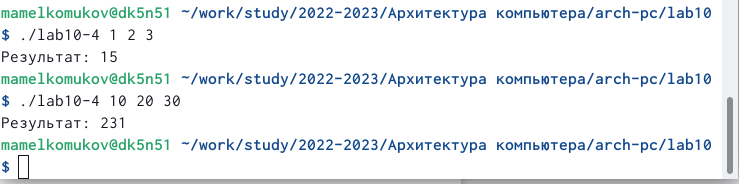
# 4 Задание для самостоятельной работы



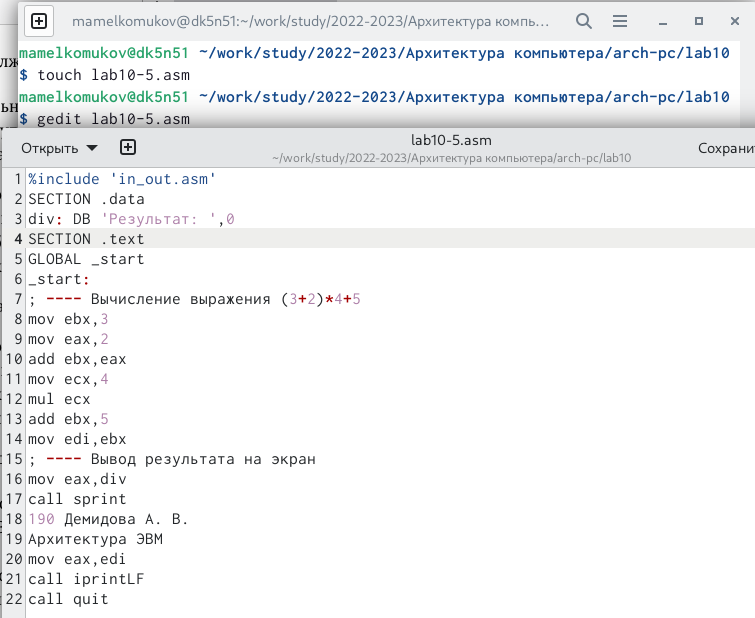
Скопировали программу из лабораторной работы №9



Реализовали вычисление значения функции как подпрограмму

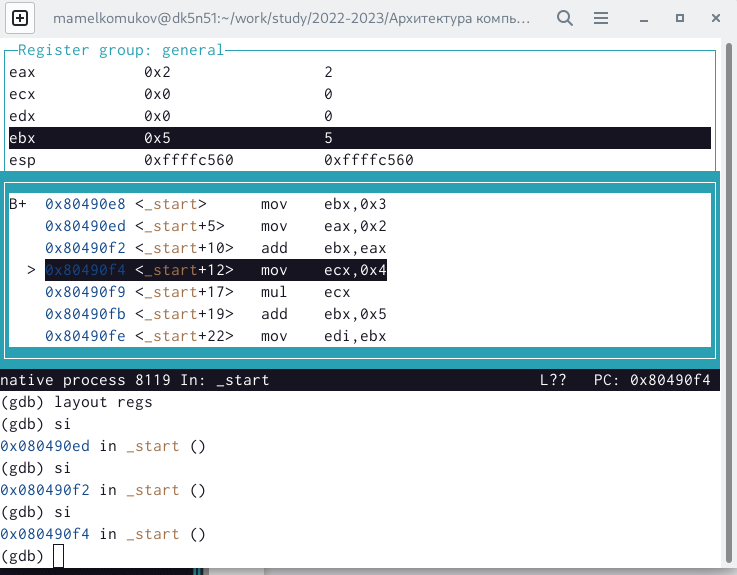


Создали исполняемый файл и проверили его работу

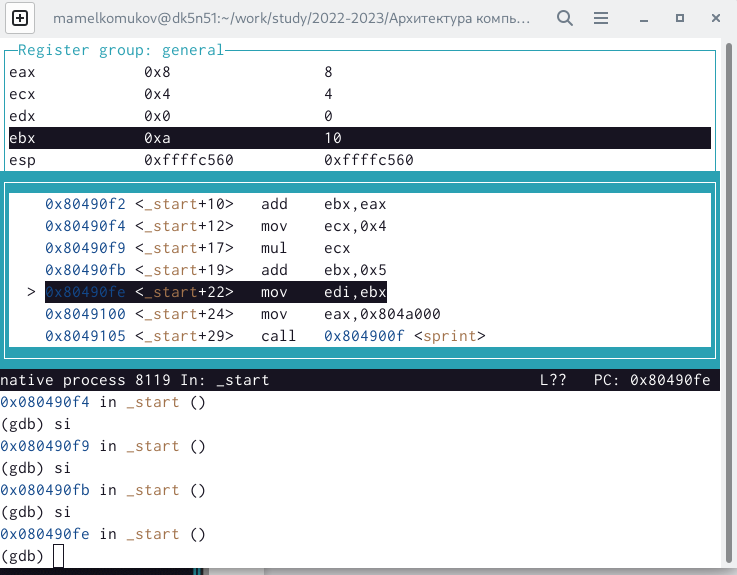


Создали файл lab10-5.asm и ввели в него текст программы

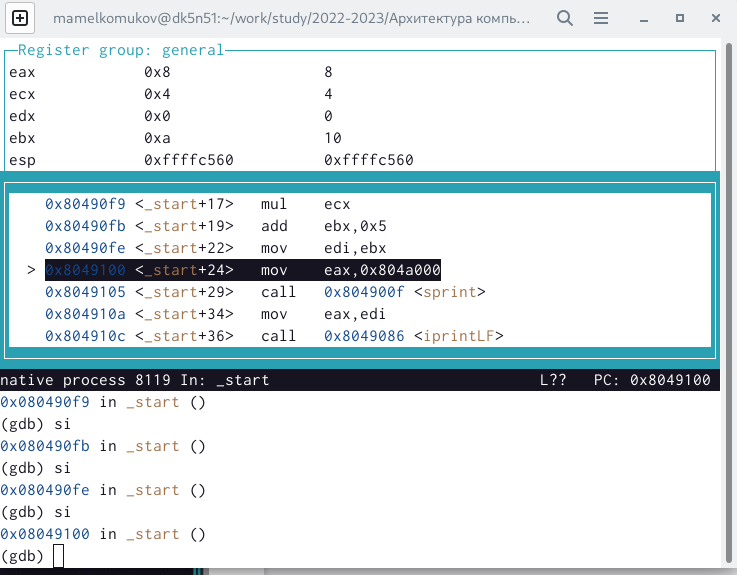
С помощью отладчика GDB проанализировали изменения значений регистров.



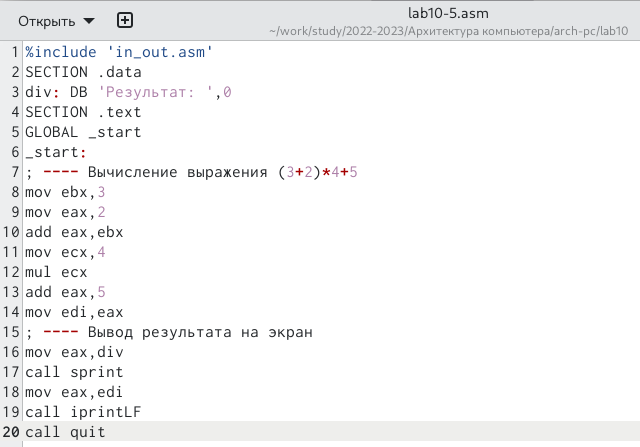
Первая ошибка: сумма элементов eax и ebx записывается в ebx



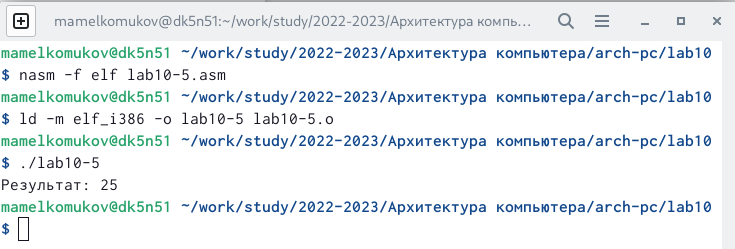
Вторая ошибка: при умножении на 5 меняется не eax, а ebx



Третья ошибка: выводится значение ebx, а не eax



Определили ошибки и исправили программу



Создали исполняемый файл и проверили его работу. Ошибка устранена

# 5 Выводы

Приобрели навыки написания программ с использованием подпрограмм а также познакомились с методами отладки при помощи GDB и его основными возможностями.