Архитектура компьютера

Отчёт по лабораторной работе №9

Арбатова Варвара Петровна

Содержание

# Цель работы

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

# Задание

1. Создайте каталог для выполнения лабораторной работы № 9, перейдите в него и создайте файл lab09-1.asm
2. Внимательно изучите текст программы (Листинг 9.1). Введите в файл lab09-1.asm текст программы из листинга 9.1. Создайте исполняемый файл и проверьте его работу
3. Создайте файл lab09-2.asm с текстом программы из Листинга 9.2
4. Проверьте работу программы, запустив ее в оболочке GDB

# Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа: • обнаружение ошибки; • поиск её местонахождения; • определение причины ошибки; • исправление ошибки. Можно выделить следующие типы ошибок: • синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка; • семантические ошибки — являются логическими и приводят к тому, что программа запускается, отрабатывает, но не даёт желаемого результата; • ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают прерывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль). Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга. Третий этап — выяснение причины ошибки. После определения местонахождения ошибки обычно проще определить причину неправильной работы программы. Последний этап — исправление ошибки. После этого при повторном запуске программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново. Наиболее часто применяют следующие методы отладки: • создание точек контроля значений на входе и выходе участка программы (например, вывод промежуточных значений на экран — так называемые диагностические сообщения); • использование специальных программ-отладчиков. Отладчики позволяют управлять ходом выполнения программы, контролировать и изменять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправление. Наиболее популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам. Пошаговое выполнение — это выполнение программы с остановкой после каждой строчки, чтобы программист мог проверить значения переменных и выполнить другие действия. Точки останова — это специально отмеченные места в программе, в которых программа отладчик приостанавливает выполнение программы и ждёт команд. Наиболее популярные виды точек останова: • Breakpoint — точка останова (остановка происходит, когда выполнение доходит до определённой строки, адреса или процедуры, отмеченной программистом); • Watchpoint — точка просмотра (выполнение программы приостанавливается, если программа обратилась к определённой переменной: либо считала её значение, либо изменила его). Точки останова устанавливаются в отладчике на время сеанса работы с кодом программы, т.е. они сохраняются до выхода из программы-отладчика или до смены отлаживаемой программы.

# Выполнение лабораторной работы

1. Создаю каталог для выполнения лабораторной работы № 9, перехожу в него и создаю файл lab09-1.asm



Figure 1: 1)Создание файла

1. Ввожу в файл lab09-1.asm текст программы из листинга 9.1. Создаю исполняемый файл и проверяю его работу (2-3)

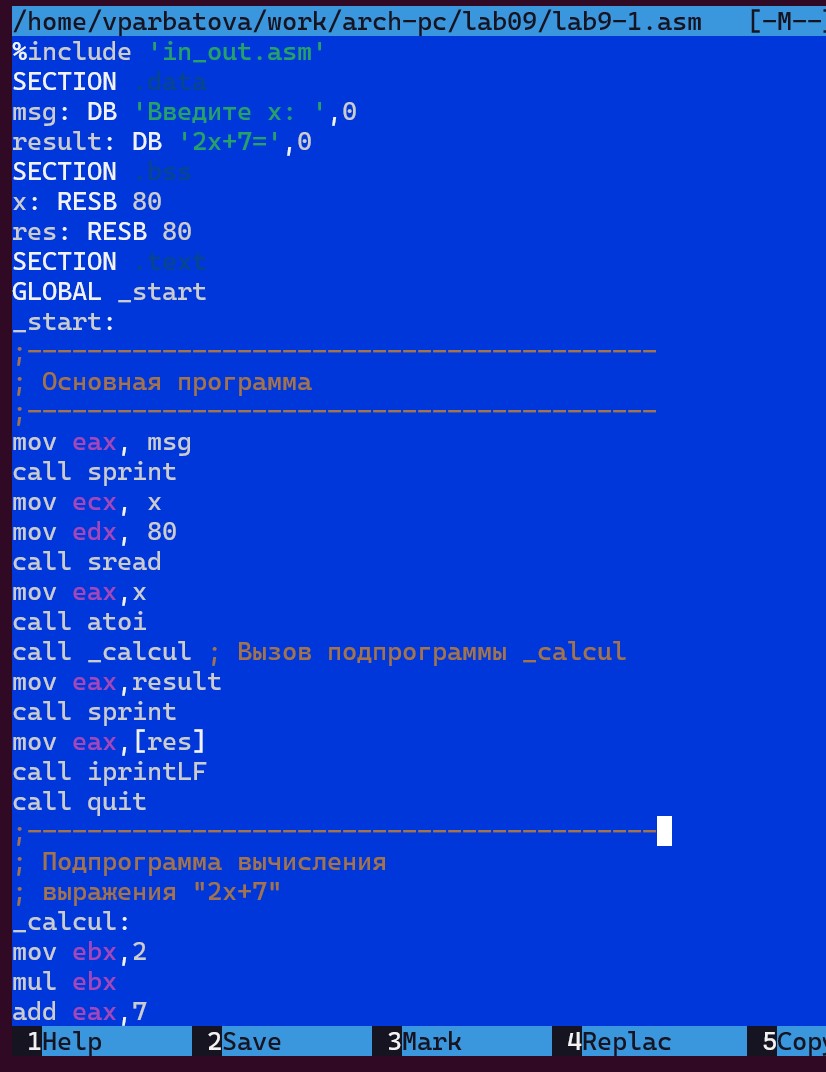


Figure 2: 2)Текст файла



Figure 3: 3)Создание файла

1. Изменяю 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. Создаю файл и проверяю его работу (4-5)

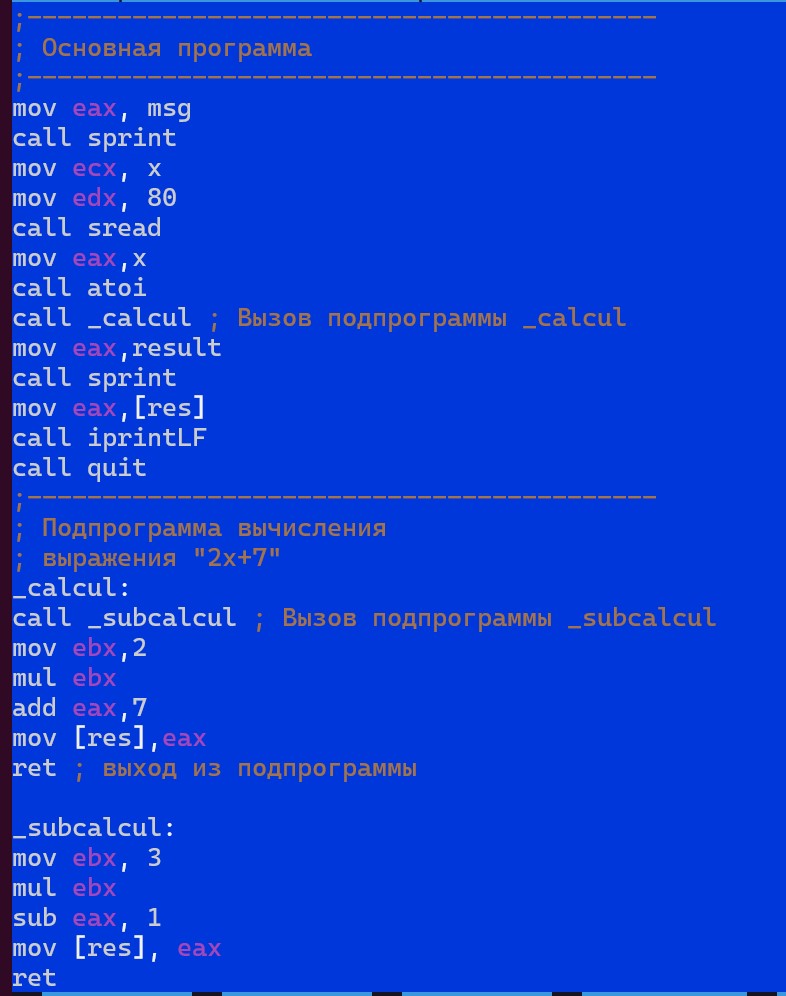


Figure 4: 4)Текст программы изменённый

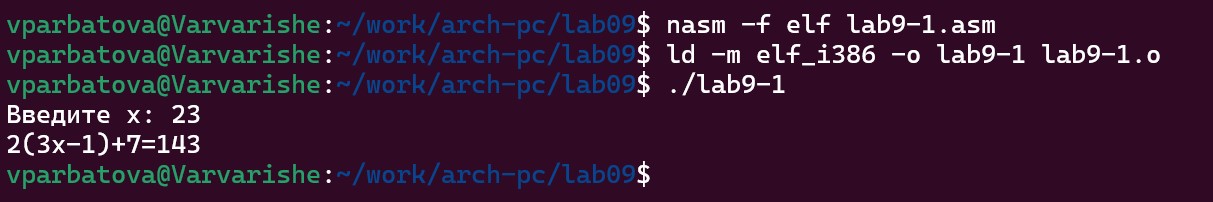


Figure 5: 5)Работа файла

1. Создаю файл lab09-2.asm с текстом программы из Листинга 9.2. (6-8)



Figure 6: 6) Создание файла

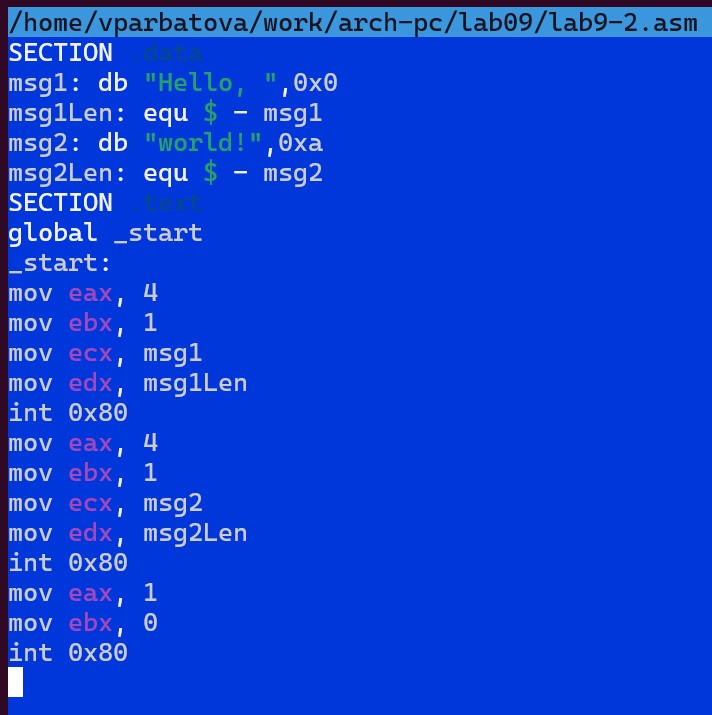


Figure 7: 7) Текст файла

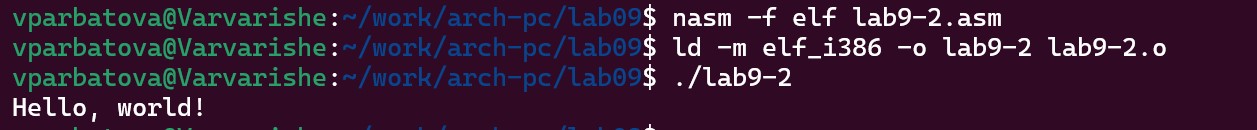


Figure 8: 8) Работа файла

1. Получаю исполняемый файл.Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’. Скачиваю gdb и запускаю его (9-10)

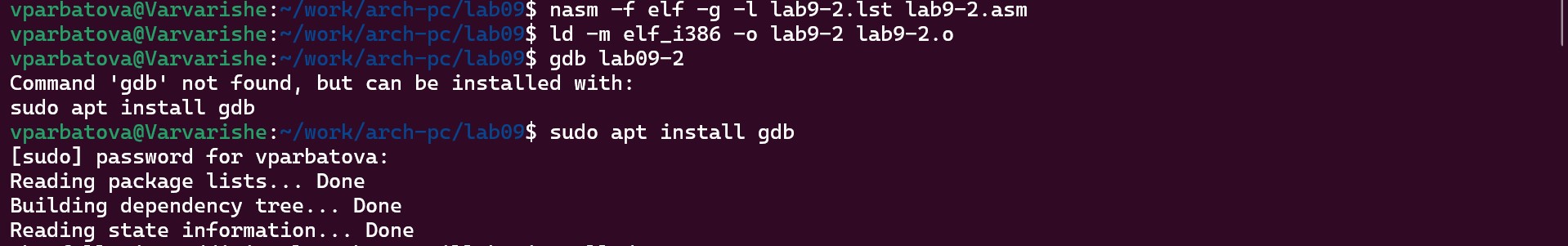


Figure 9: 9) Скачиваю gdb

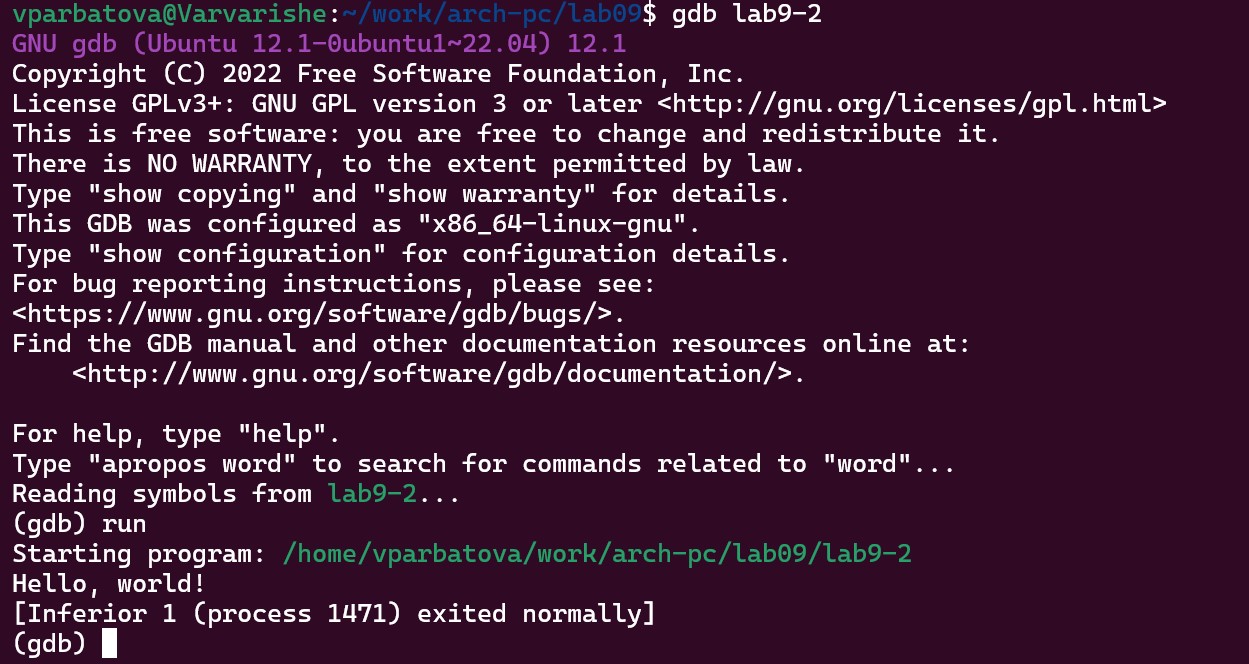


Figure 10: 10) Запуск gdb и файла

1. Для более подробного анализа программы установливаю брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запускаю её

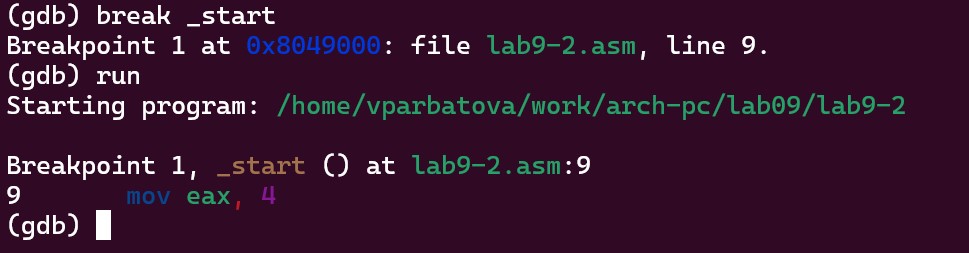


Figure 11: 11) Установка break\_point и запуск программы с ней

1. Смотрю дисассимилированный код программы с помощью команды disassemble начиная с метки \_start

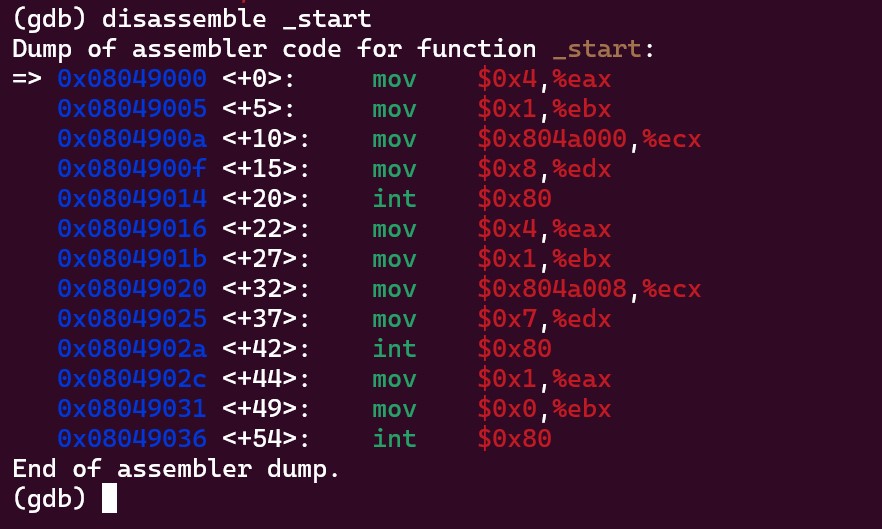


Figure 12: 12)Дисассимилированный код

1. Переключаюсь на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel



Figure 13: 13)Отображение команд с Intel’овским синтаксисом

1. Включаю режим псевдографики для более удобного анализа программы

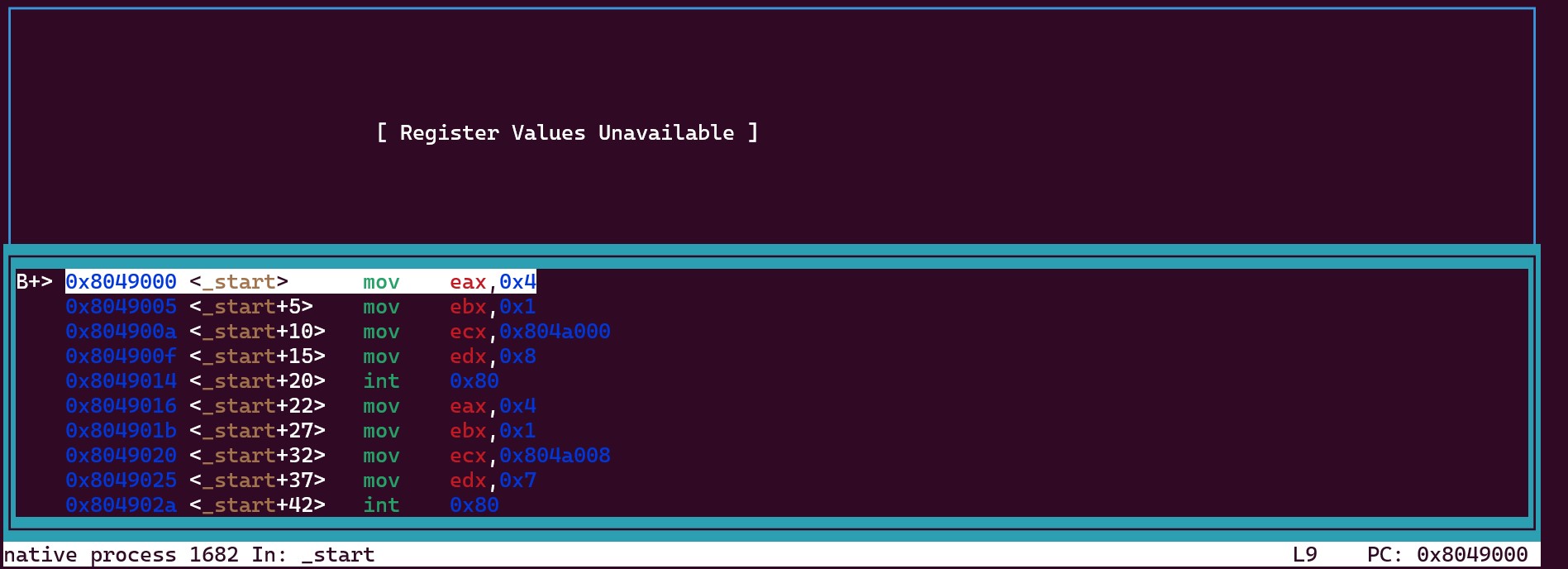


Figure 14: 14)Режим псевдографики

1. С помощью info breakpoints узнаю информацию об установленных точках останова

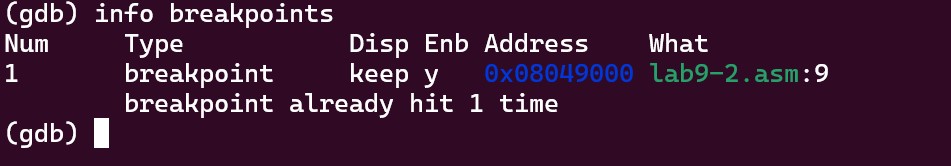


Figure 15: 15)info breakpoints

1. Определяю адрес предпоследней инструкции (mov ebx,0x0) и устанавливаю точку останова. Смотрю информацию об установленных точках останова (16-17)

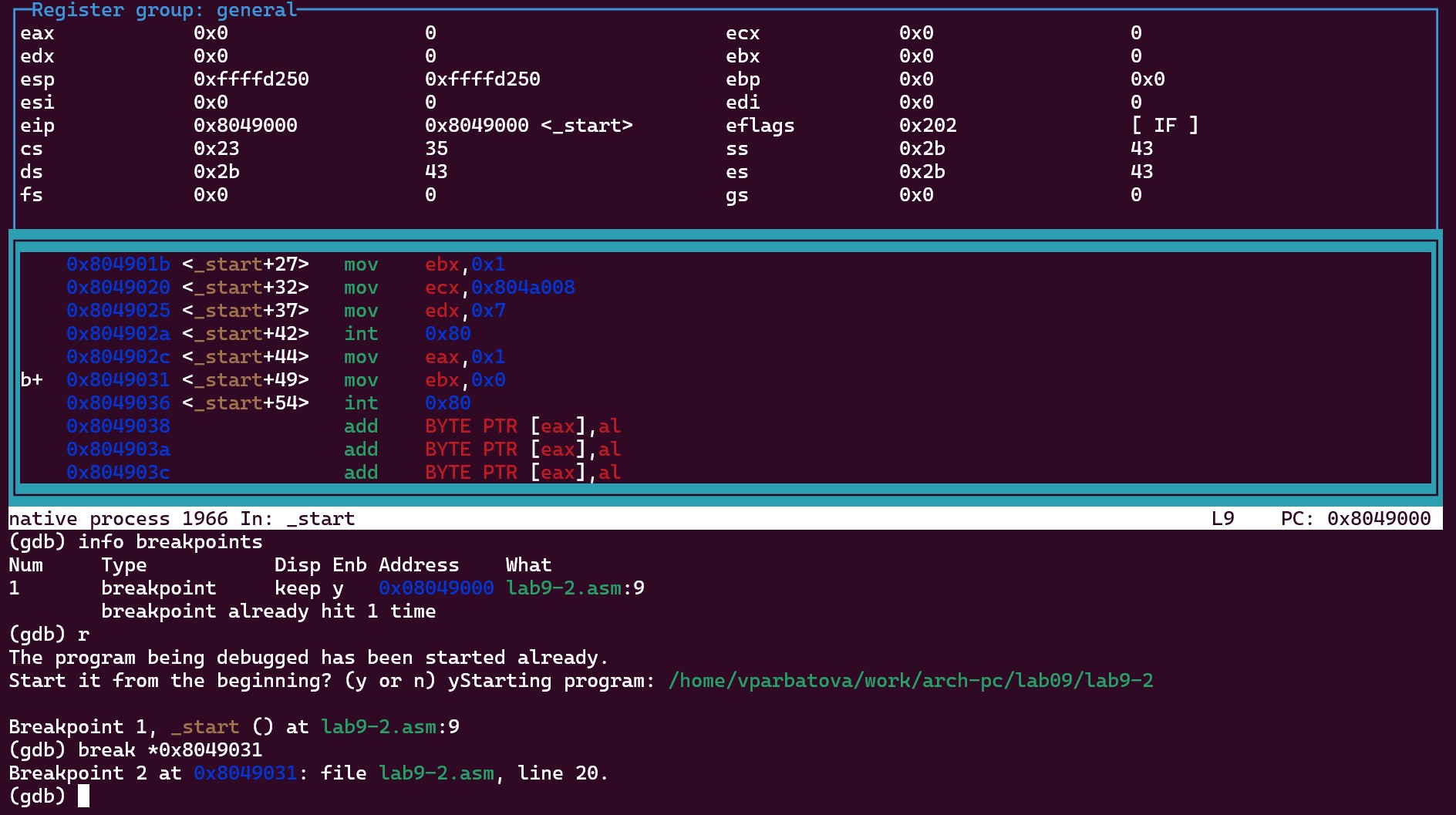


Figure 16: 16)Установка точки останова

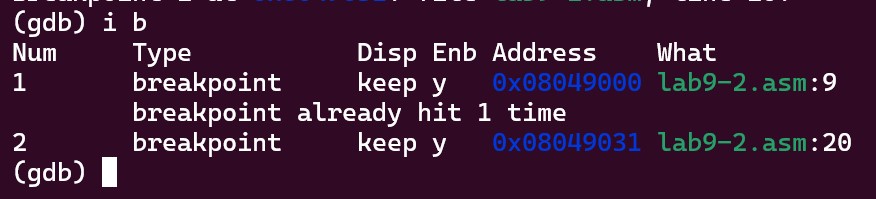


Figure 17: 17)Вывод информации о точках останова

1. Выполняю 5 инструкций с помощью команды stepi (или si) и слежу за изменением значений регистров. Значения регистров eax, edx, ecx, esp, eip, cs, ds, ebx, ss, eflags, es изменяются

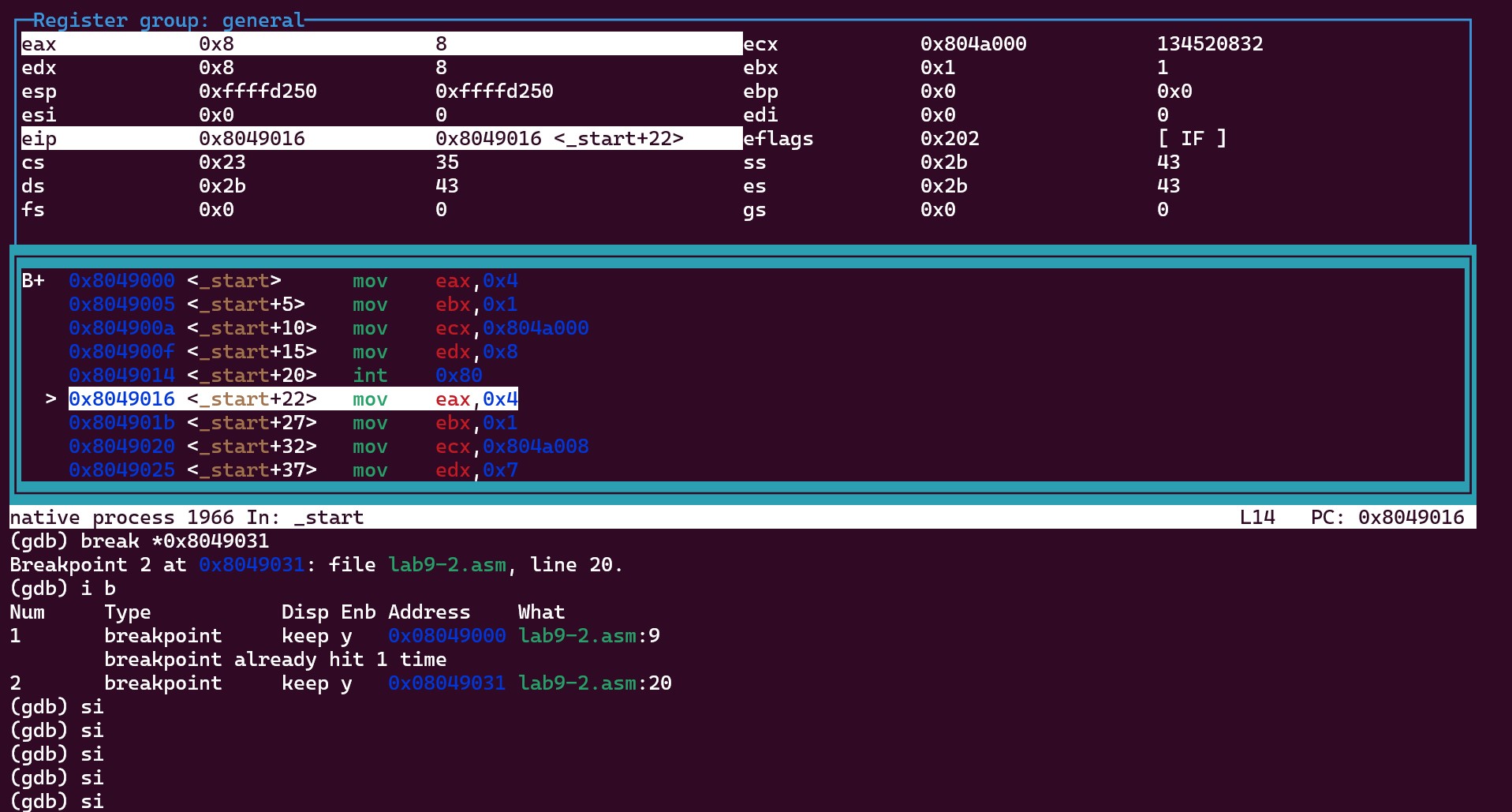


Figure 18: 18)5 инструкций si

1. Посмотреть содержимое регистров также можно с помощью команды info registers

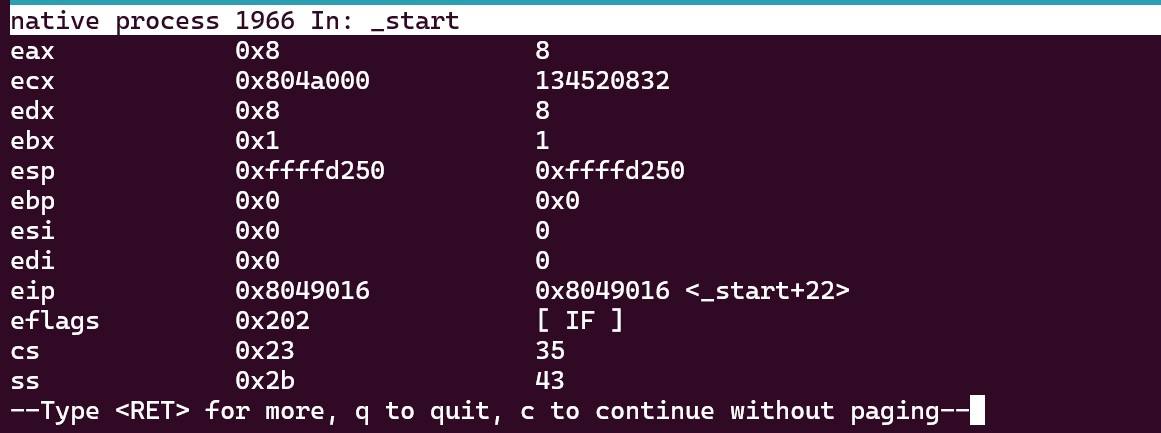


Figure 19: 19)info registers

1. Смотрю значение переменной msg1 по имени и переменной msg2 по адресу (20-21)

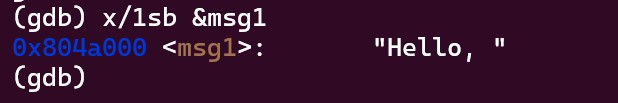


Figure 20: 20)Значение переменной msg1 по имени



Figure 21: 21)Значение переменной msg2 по адресу

1. Изменяю первый символ переменной msg1

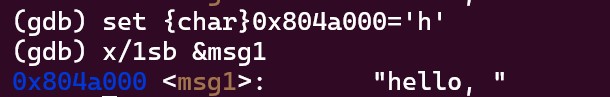


Figure 22: 22)Изменение первого символа переменной msg1

1. Заменяю символ во второй переменной msg2

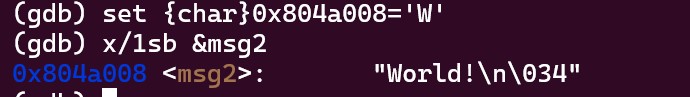


Figure 23: 23)Изменение символа второй переменной

1. Вывожу в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx.

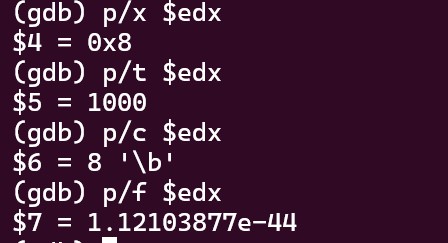


Figure 24: 24)Значение регистра edx

1. С помощью команды set изменяю значение регистра ebx. В первом случае выводит значение символа (его код), во втором - число

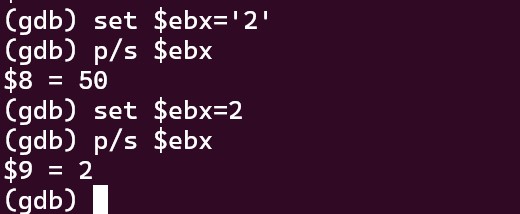


Figure 25: 25)Изменяю значение регистра ebx

1. Завершаю выполнение программы с помощью команды continue (сокращенно c) и выхожу из GDB с помощью команды quit (сокращенно q)

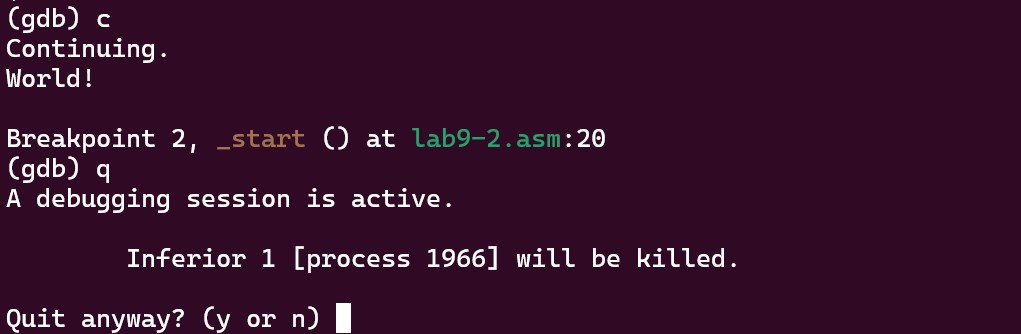


Figure 26: 26)Завершаю программу

1. Копирую файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm, создаю исполняемый файл, загружаю исполняемый файл в отладчик, указав аргументы:

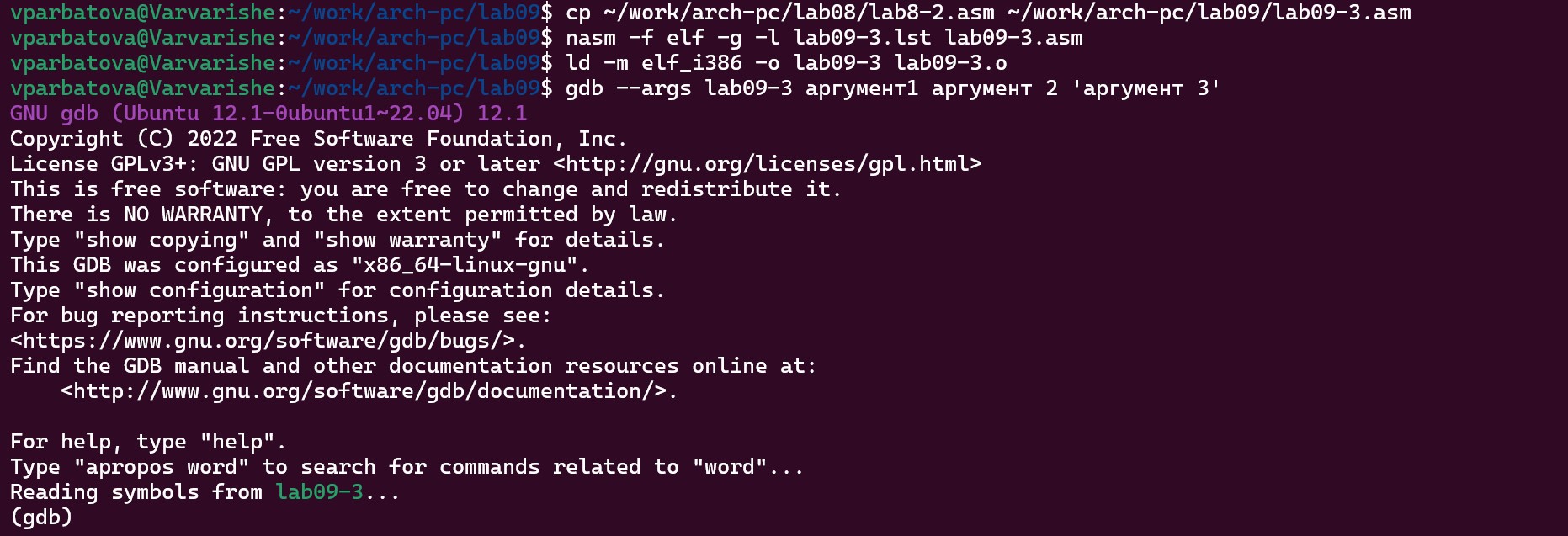


Figure 27: 27)Копирование, создание, загрузка файла

1. Для начала уставливаю точку останова перед первой инструкцией в программе и запускаю её.

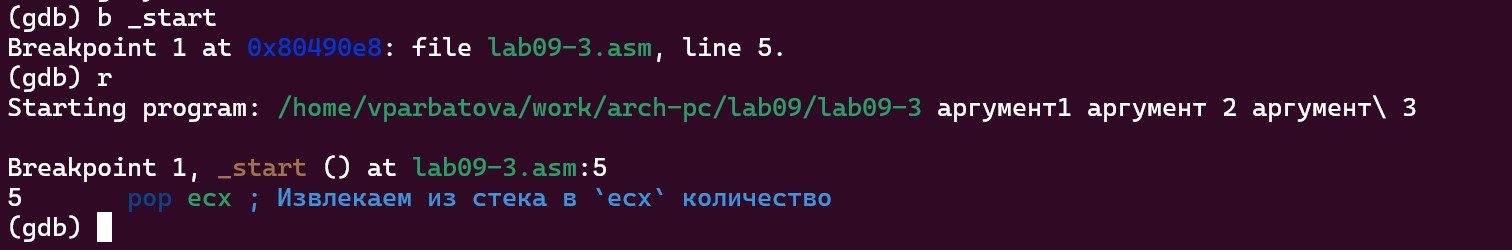


Figure 28: 28)Точка установа и запуск

1. Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы)

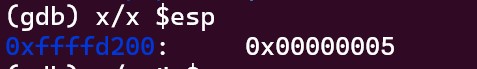


Figure 29: 29)Регистр esp

1. Смотрю остальные позиции стека. Шаг изменения равен 4 потому что шаг - int, а под этот тип данных выделяется 4 байта



Figure 30: 30)Остальные позиции стека

# Выполнение заданий для самостоятельной работы

1. Копирую файл

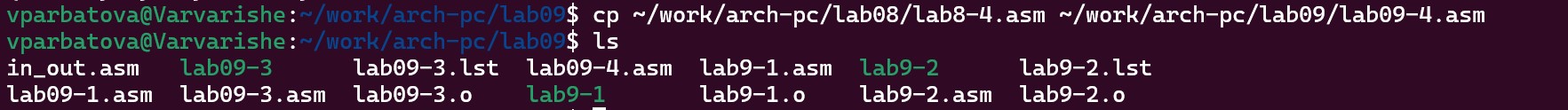


Figure 31: Копирование файла

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

Текст программы:

%include ‘in\_out.asm’ SECTION .data msg db “Результат:”,0 msg1 db “Функция: f(x) = 2x + 15”,0 SECTION .text global \_start \_start: pop ecx ; Извлекаем из стека в ecx количество ; аргументов (первое значение в стеке) pop edx ; Извлекаем из стека в edx имя программы ; (второе значение в стеке) sub ecx,1 ; Уменьшаем ecx на 1 (количество ; аргументов без названия программы) mov esi, 0 ; Используем esi для хранения ; промежуточных сумм next: cmp ecx,0h ; проверяем, есть ли еще аргументы jz \_end ; если аргументов нет выходим из цикла ; (переход на метку \_end) pop eax ; иначе извлекаем следующий аргумент из стека call atoi ; преобразуем символ в число call \_function loop next \_end: mov eax, msg1 call sprintLF mov eax, msg ; вывод сообщения “Результат:” call sprint mov eax, esi ; записываем сумму в регистр eax call iprintLF ; печать результата call quit ; завершение программыmcedit lab09-4.asm \_function: mov ebx, 2 mul ebx add eax, 15 add esi,eax ; добавляем к промежуточной сумме ; след. аргумент esi=esi+eax ret

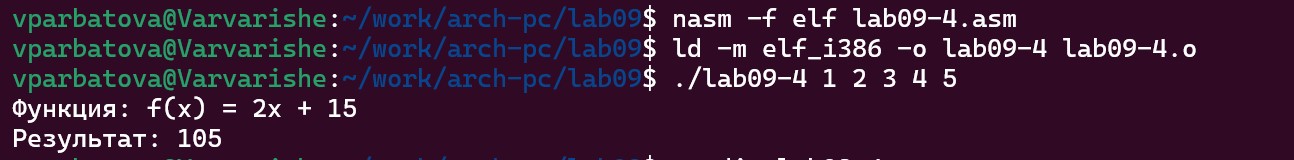


Figure 32: Работа файла

1. Создаю файл, ввожу туда текст программы

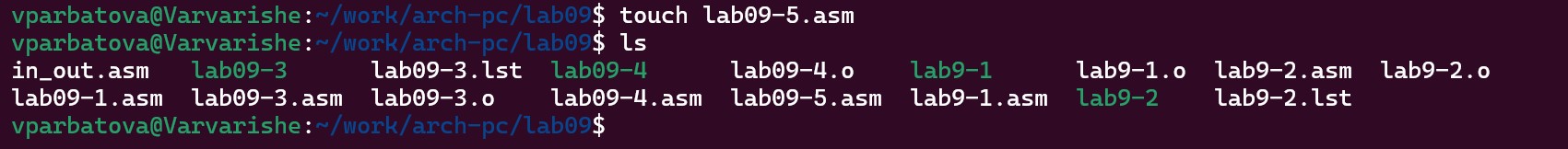


Figure 33: Создание файла

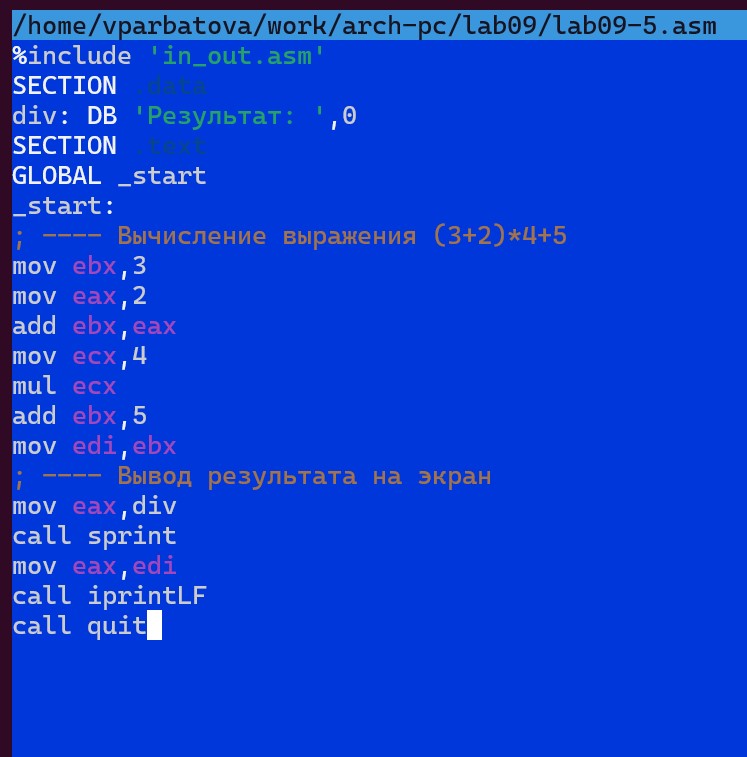


Figure 34: Текст файла

1. Передаю файл на отладку

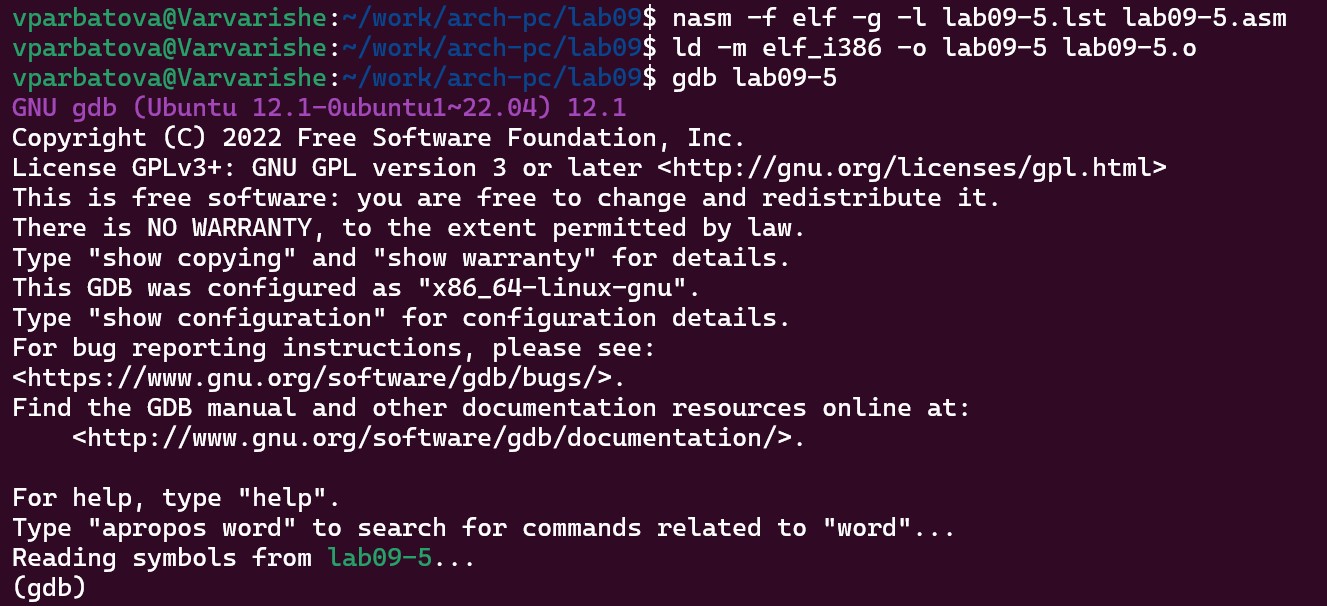


Figure 35: Передача файла

1. Устанавливаю точки останова

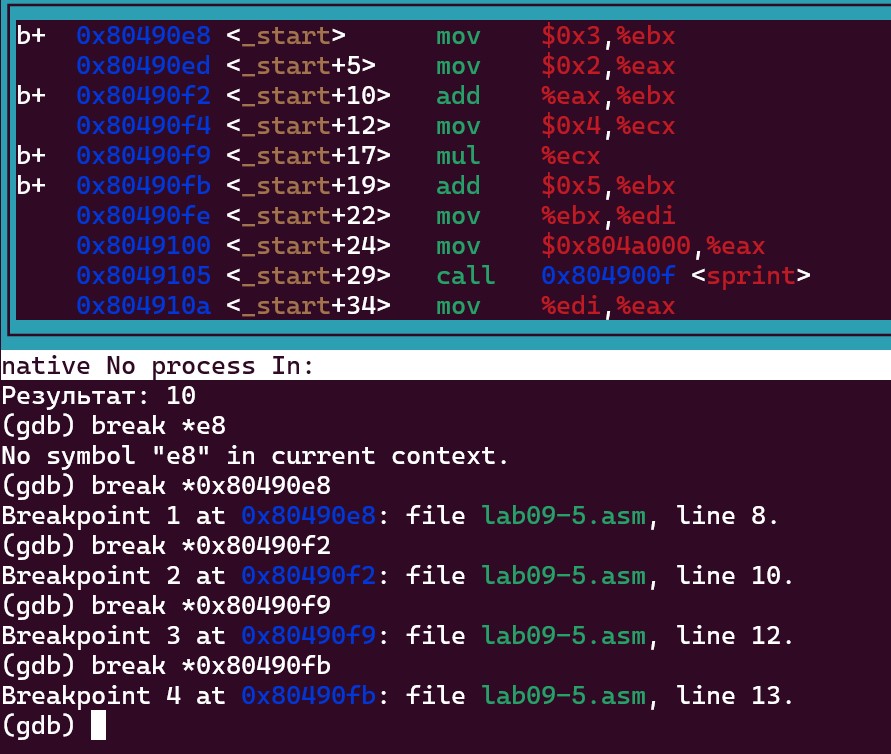


Figure 36: Точки останова

1. Прохожу по программе обращая внимание на несостыковки с логикой. Таким образом нашла 3 ошибки

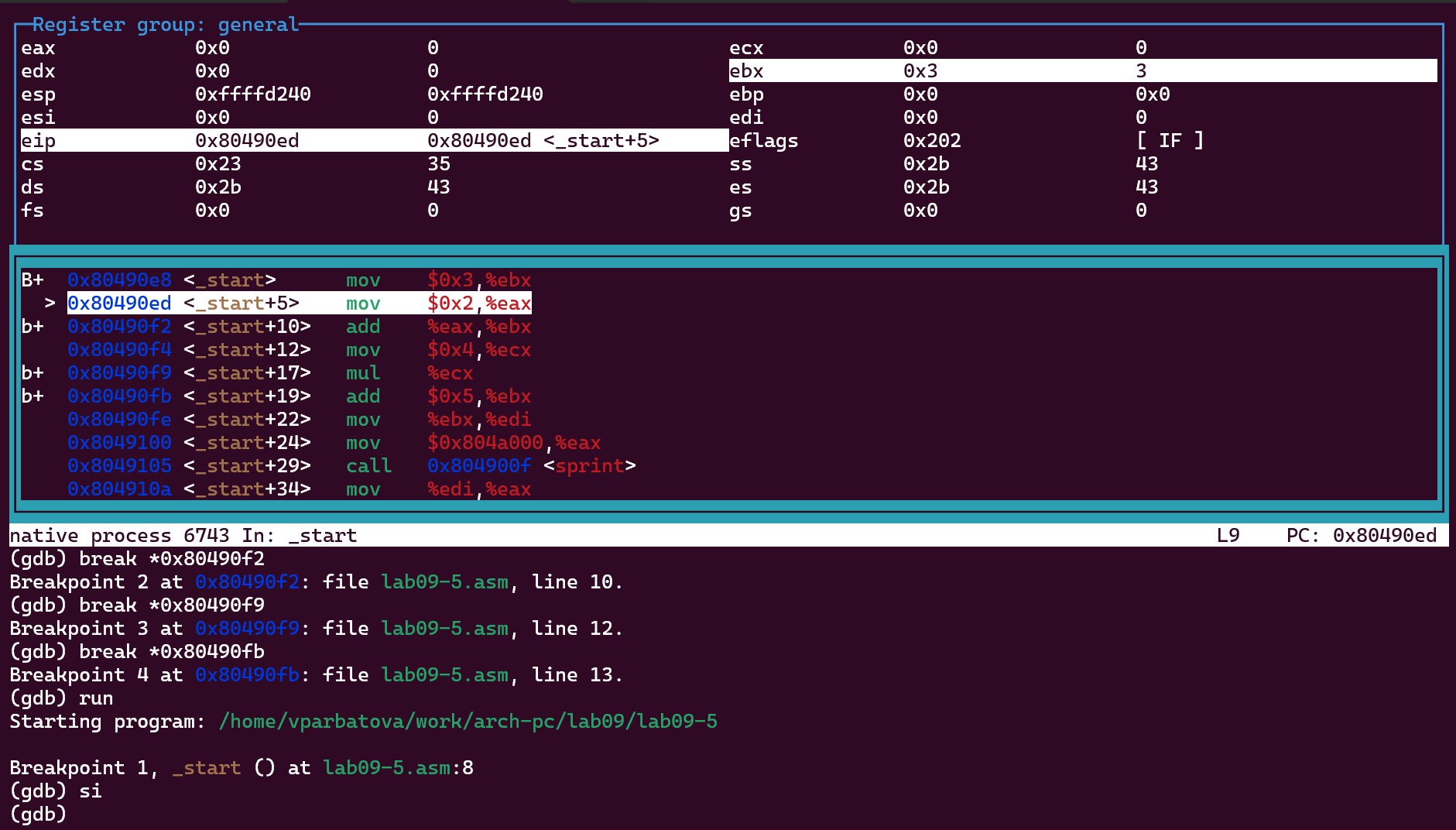


Figure 37: Отладка файла

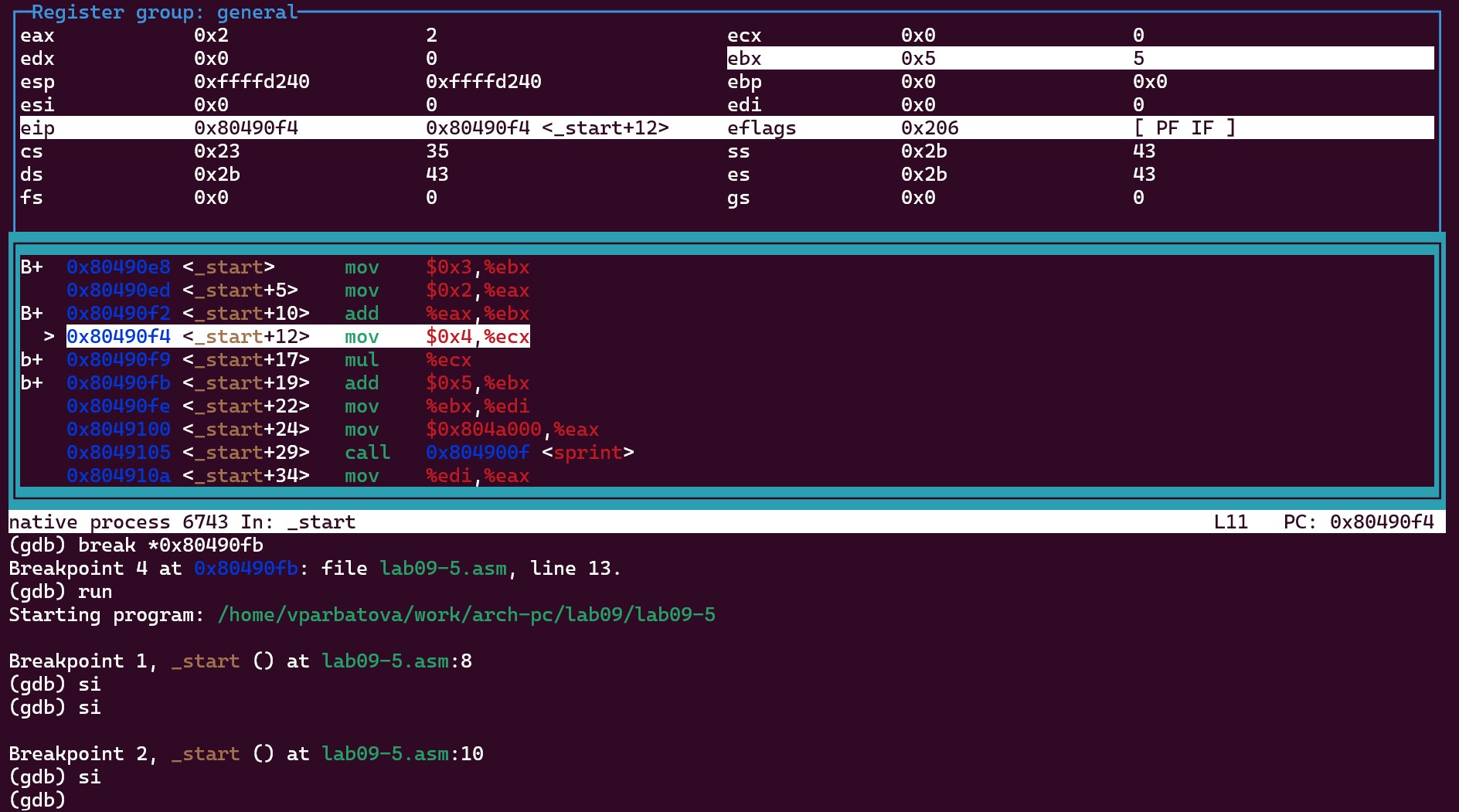


Figure 38: Первая ошибка

1. Исправляю ошибки

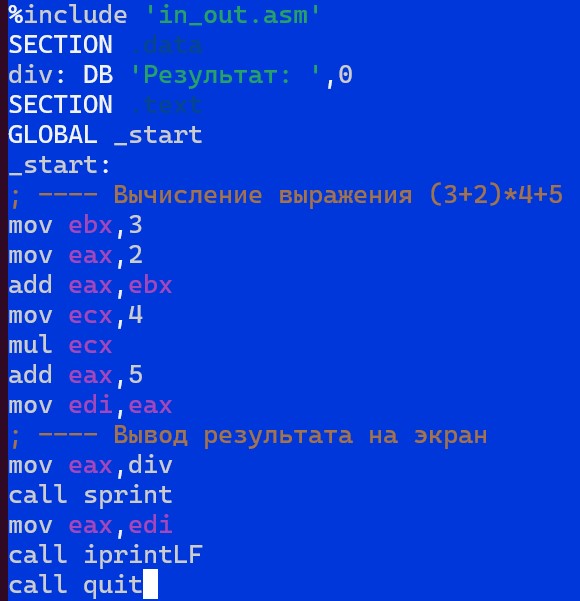


Figure 39: Исправленный текст файла

1. Работа файла

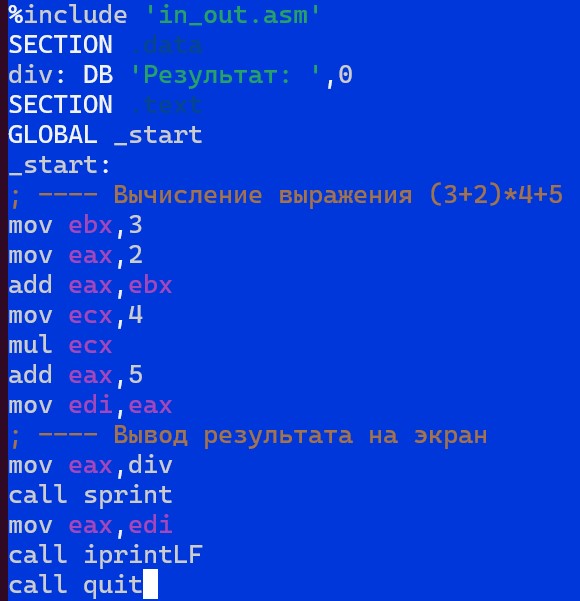


Figure 40: Работа файла

# Выводы

Приобрела навыки написания программ с использованием подпрограмм. Познакомилась с методами отладки при помощи GDB и его основными возможностями.

# Список литературы