Лабораторная работа №10

Архитектура компьютера

Скандарова Полина Юрьевна

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

Для начала необходимо создать каталог для выполнения лабораторной работы №10, перейти в него и создать файл lab10-1.asm. В качестве примера рассматривается программа вычисления арифметического выражения f(x) = 2x + 7 с помощью подпрограммы \_calcul. В данном примере x вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Внимательно изучаю текст программы и ввожу её в файл. (рис. 1)

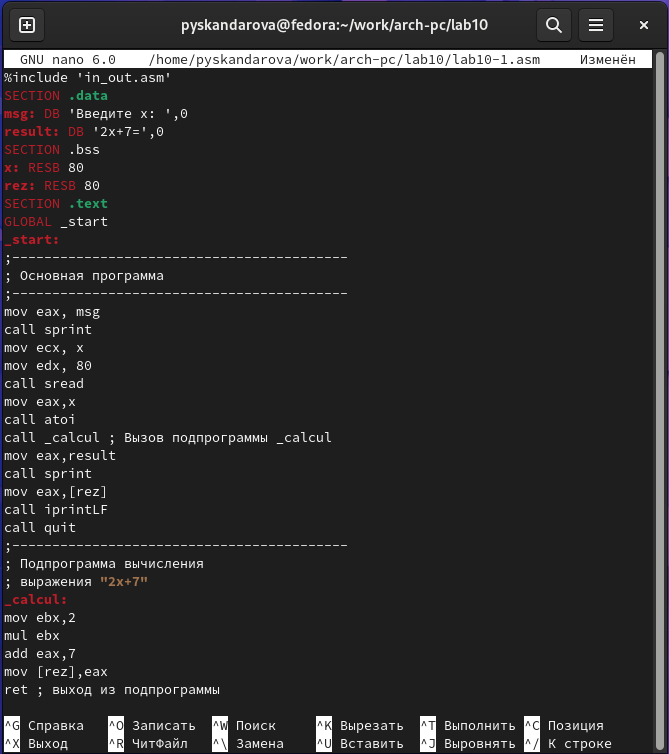


Рис. 1: Текст программы в файле

Создаю исполняемый файл и проверяю его работу. (рис. 2)

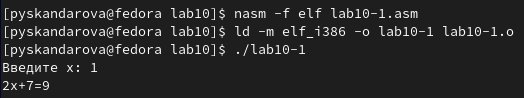


Рис. 2: Проверка работы программы

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. (рис. 3)

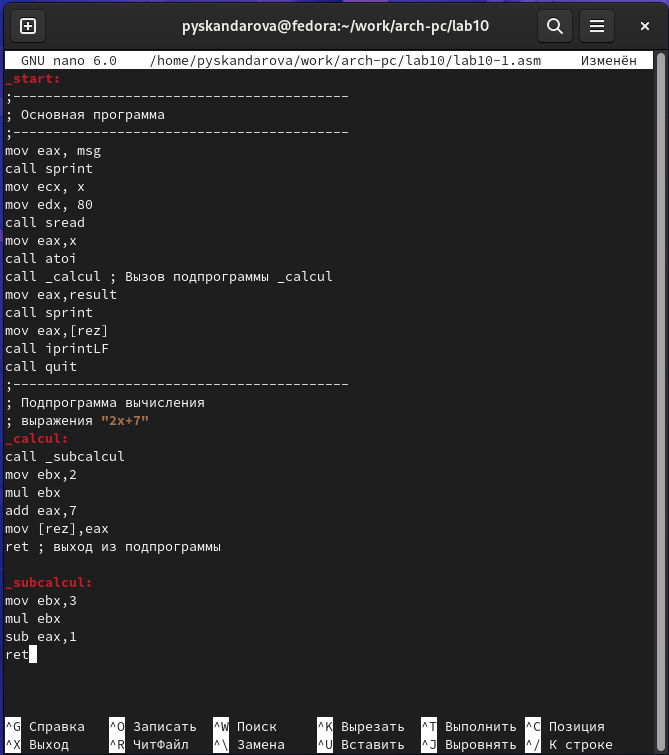


Рис. 3: Текст программы в файле

После чего её проверяю. (рис. 4)

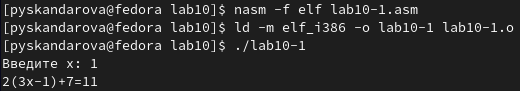


Рис. 4: Проверка работы программы

Дальше создаю файл lab10-2.asm с текстом программы из листинга. (Программа печати сообщения Hello world!). (рис. 5)

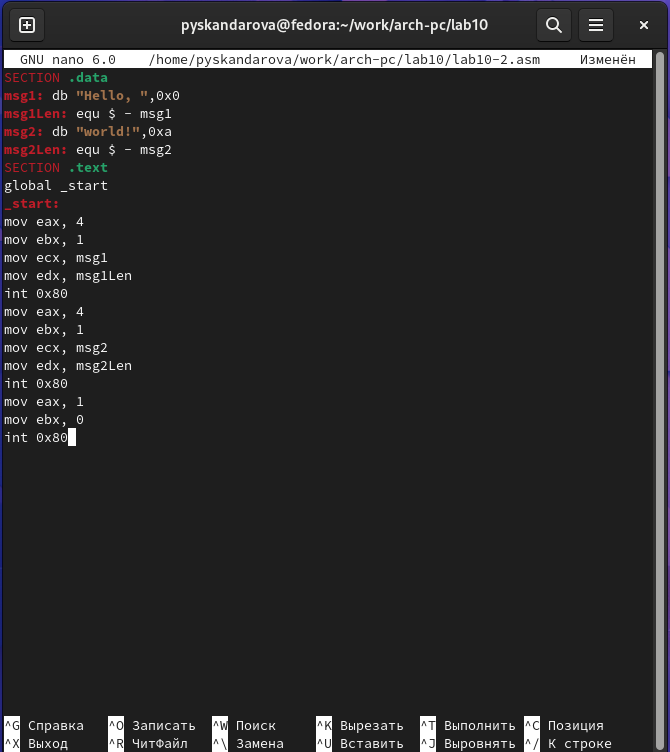


Рис. 5: Текст программы в файле

Получаю исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’. Потом загружаю исполняемый файл в отладчик gdb и проверяю работу программы, запустив ее в оболочке GDB с помощью команды run (сокращённо r).(рис. 6)

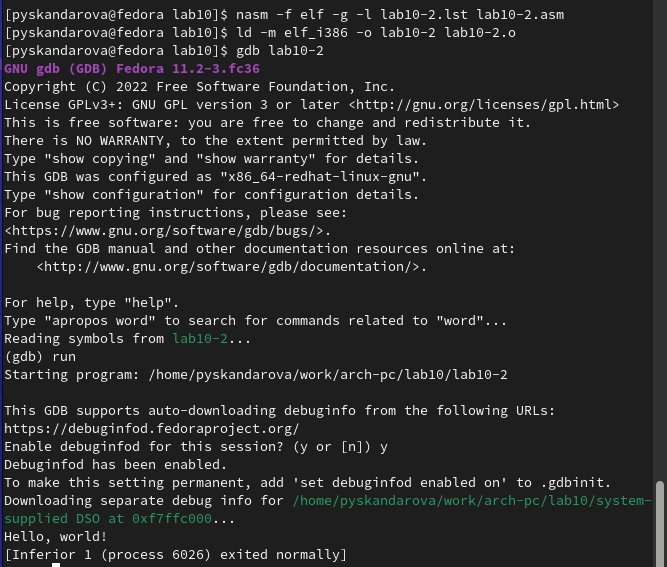


Рис. 6: Загрузка программы в оболочку GDB и её запуск

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

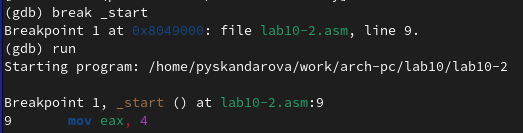


Рис. 7: Запуск программы с брейкпоинтом

Теперь смотрю дисассимилированный код программы с помощью команды disassemble начиная с метки \_start. (рис. 8)



Рис. 8: Дисассимилированный код программы

Переключаюсь на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel. Разница синтаксиса машинных команд в режимах ATT и Intel в том, что в АТТ перед всеми значениями и адресами стоять символы $, а перед регистрами - %, также в АТТ во всех командах mov аргументы стоят в обратном порядке. (рис. 9)

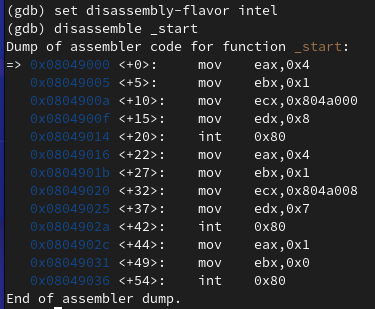


Рис. 9: Дисассимилированный код программы с Intel’овским синтаксисом

Далее включаю режим псевдографики для более удобного анализа программы. (рис. 10)

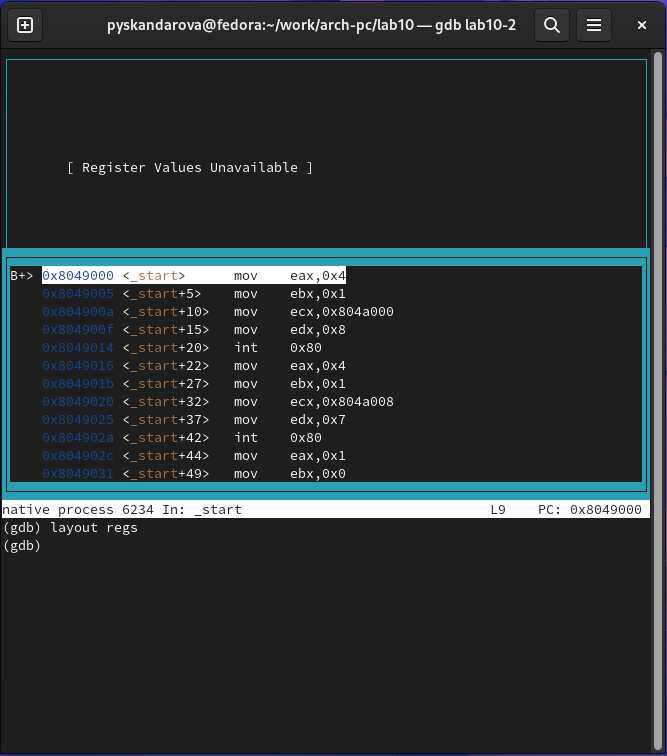


Рис. 10: Режим псевдографики gdb

На предыдущих шагах была установлена точка останова по имени метки (\_start). Проверяю это с помощью команды info breakpoints (кратко i b). (рис. 11)

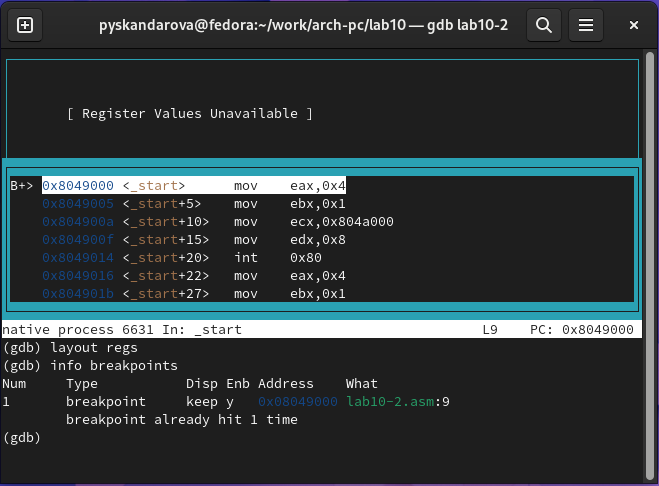


Рис. 11: Информация о точке останова

Устанавливаю еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции. Определяю адрес предпоследней инструкции (mov ebx,0x0) и устаноавливаю точку останова.(рис. 12)

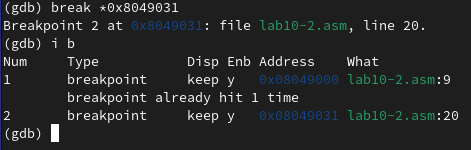


Рис. 12: Новая точка останова и информация о ней

Посмотреть содержимое регистров можно с помощью команды info registers (или i r). (рис. 13)

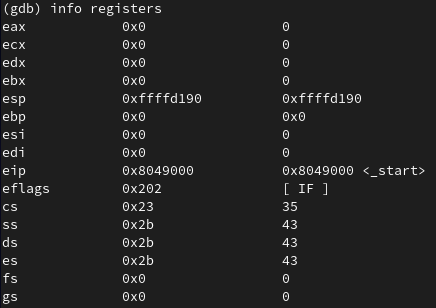


Рис. 13: Результат работы команды info registers

С помощью команды x & можно посмотреть содержимое переменной. Я смотрю значение переменной msg1 по имени. (рис. 14)

Рис. 14: Содержимое переменной msg1

Рис. 14: Содержимое переменной msg1

Значение переменной msg2 смотрю по адресу. (рис. 15)

Рис. 15: Содержимое переменной msg1

Рис. 15: Содержимое переменной msg1

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). Я изменяю первый символ переменной msg1. (рис. 16)

Рис. 16: Изменение содержимого переменной msg1

Рис. 16: Изменение содержимого переменной msg1

Замените любой символ во второй переменной msg2.(рис. 17)

Рис. 17: Изменение содержимого переменной msg2

Рис. 17: Изменение содержимого переменной msg2

Вывожу в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx. (рис. 18)

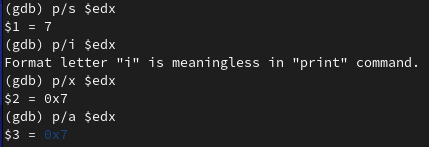


Рис. 18: Значение регистра edx в разных форматах

С помощью команды set изменяю значение регистра ebx. (рис. 19)

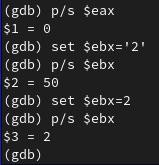


Рис. 19: Меняющиеся значения регистра ebx

В первый раз в регистре находится символ ‘2’ и программа выводит его код, во второй - значение 2. Завершаю выполнение программы с помощью команды continue (сокращенно c) или stepi (сокращенно si) и выхожу из GDB с помощью команды quit (сокращенно q). (рис. 20)

Рис. 20: Выход из GDB

Рис. 20: Выход из GDB

Копирую файл lab9-2.asm, созданный при выполнении лабораторной работы №9, с программой выводящей на экран аргументы командной строки в файл с именем lab10-3.asm и создаю исполняемый файл. Для загрузки в gdb программы с аргументами необходимо использовать ключ –args. Загружаю исполняемый файл в отладчик, указав аргументы. (рис. 21)

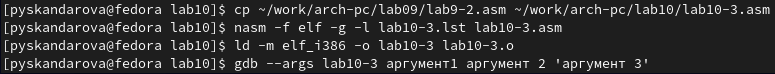


Рис. 21: Загрузка файла lab10-3 в GDB

Как отмечалось в предыдущей лабораторной работе, при запуске программы аргументы командной строки загружаются в стек. Исследую расположение аргументов командной строки в стеке после запуска программы с помощью gdb для начала установив точку останова перед первой инструкцией в программе и запустив ее. (рис. 22)

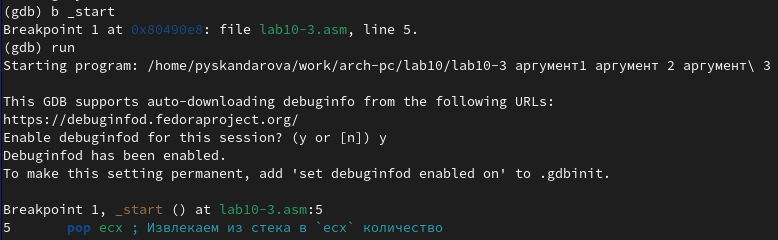


Рис. 22: Запуск файла lab10-3 в GDB

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы). Смотрите остальные позиции стека – по адесу [esp+4] располагается адрес в памяти где находиться имя программы, по адесу [esp+8] храниться адрес первого аргумента, по аресу [esp+12] – второго и т.д.(рис. 23)

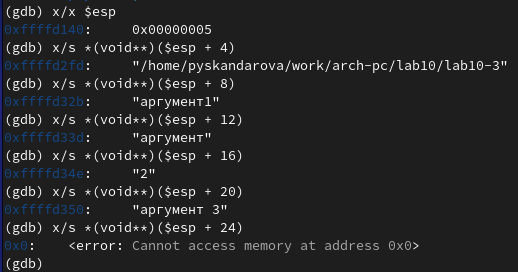


Рис. 23: Аргументы программы по их адресам

Шаг равен 4, так как каждый адрес занимает 4 байта.

# 3 Задание для самостоятельной работы

1. Преобразую программу из лабораторной работы №9 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму. (рис. 24)

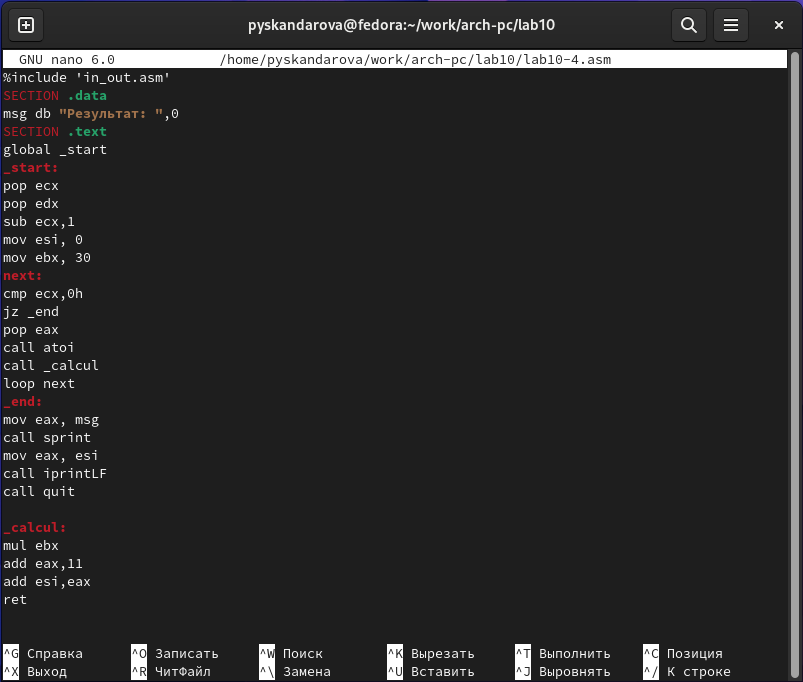


Рис. 24: Изменённая программа в файле

И проверяю её работу. (рис. 25)

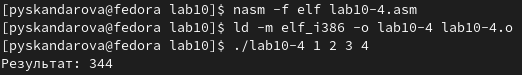


Рис. 25: Проверка работы программы

1. В листинге приведена программа вычисления выражения (3 + 2) \* 4 + 5. При запуске данная программа дает неверный результат. Проверяю это и с помощью отладчика GDB, анализируя изменения значений регистров, определяю ошибку и исправляю ее.(рис. 26)

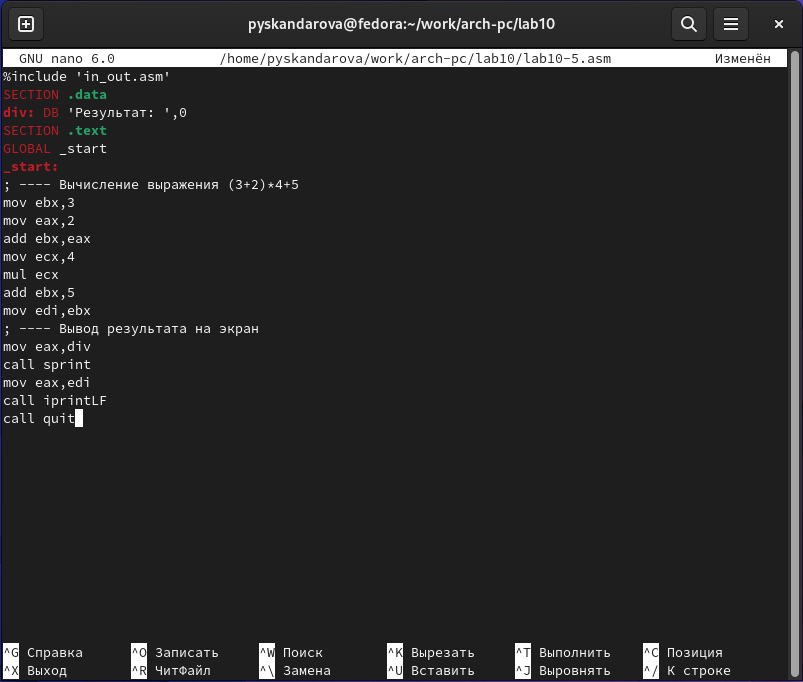


Рис. 26: Программа из листинга в файле

Результат сложения оказывается не в том регистре и не участвует в умножении, сложение и вывод происходят с регистром, где умножения не происходило. (рис. 27) (рис. 28)

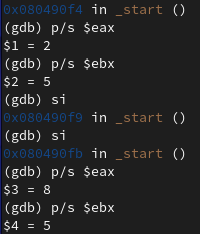


Рис. 27: Значения регистров в процессе выполнения программы

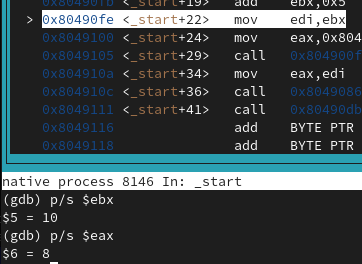


Рис. 28: В выводе участвует регистр ebx вместо регистра eax

Исправляю программу.(рис. 29)

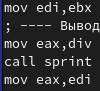


Рис. 29: Исправленная программа в файле

Теперь вывод корректен. (рис. 30)

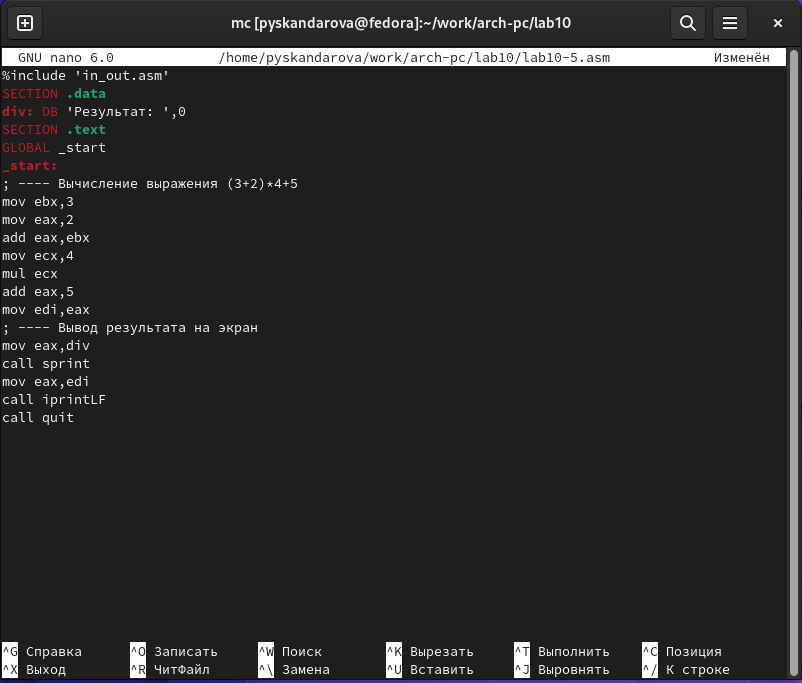


Рис. 30: Результат выполнения исправленной программы

# 4 Выводы

В ходе выполнения лабораторной работы приобретены навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями прошло успешно.