Шаблон отчёта по лабораторной работе

№10

Сильвен Макс Грегор Филс , НКАбд-03-22

Содержание

# 1 Цель работы

* В этой лабораторной работе мы научимся писать программы с использованием подпрограмм и познакомимся со способами отладки с использованием GDB и его основными функциями

# 2 Выполнение лабораторной работы :

## 2.1 Реализация циклов в NASM :

* Здесь мы начали с создания каталога для программаы лабораторной работы No10, а затем переместились в десятой каталог лаборатории “~/work/arch- pc/lab10”, после чего мы создали файл “lab10-1.asm”. (рис. 1)

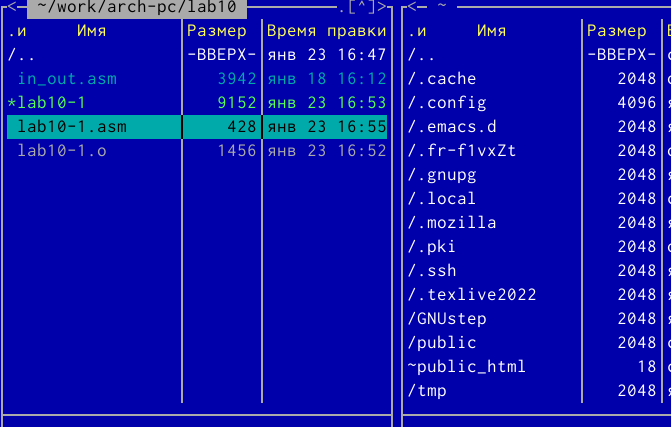


Рис. 1: Ресунок

* Затем мы заполнили код нашей программы в файле lab10-1.asm.(рис. 2)

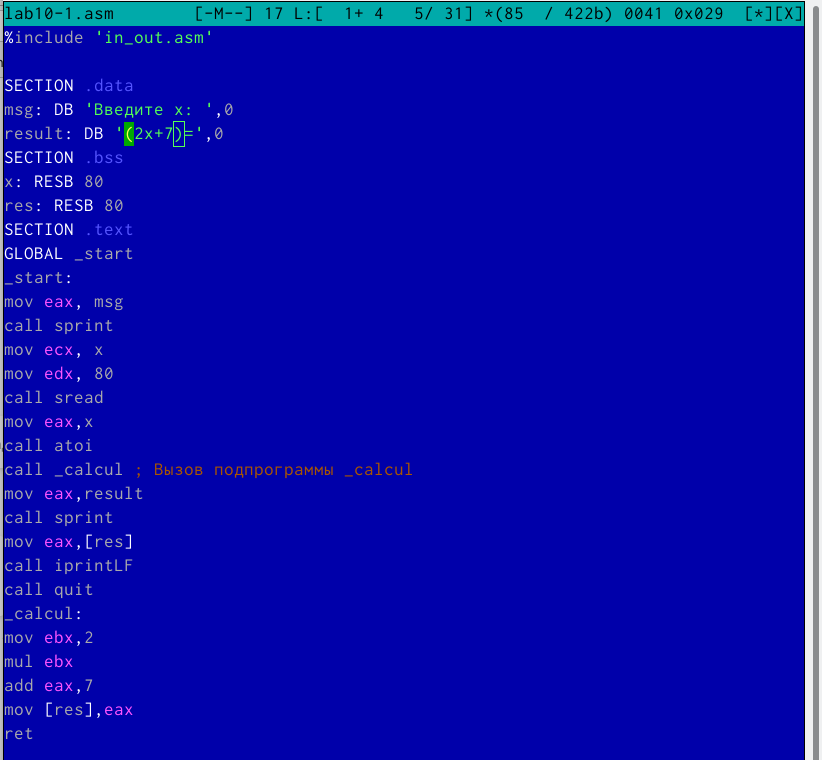


Рис. 2: Ресунок

* После этого мы скомпилировали файл, создали исполняемый файл и про- верили его работу.(рис. 3)

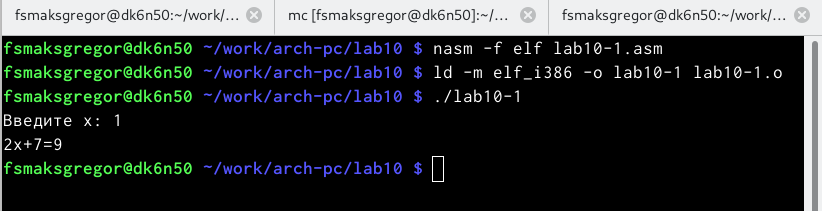


Рис. 3: Ресунок

* Мы внесли изменения в наш код ,чтобы она вычислила это уравнение f(g(x)), где x вводится с клавиатуры и f(x) = 2x + 7, g(x) = 3x − 1 а затем создали исполняемый файл.(рис. 4)(рис. 5)

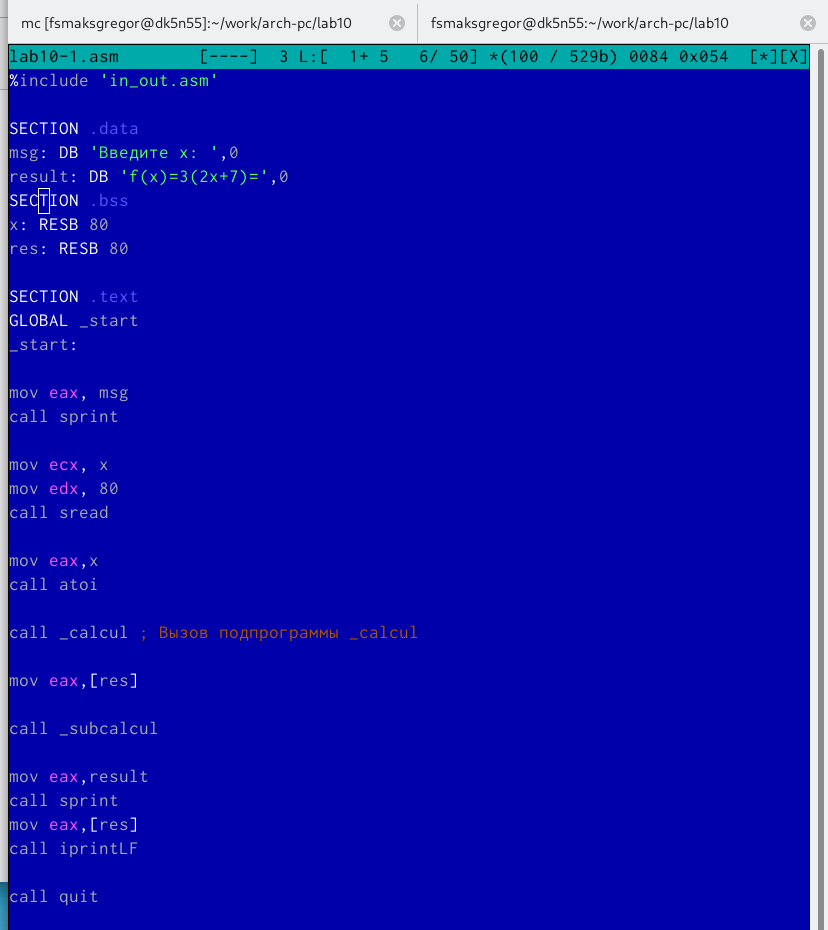


Рис. 4: Ресунок

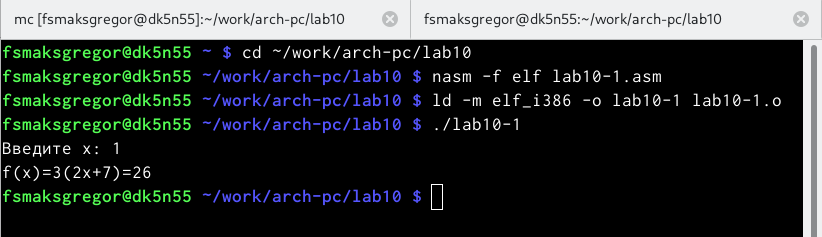


Рис. 5: Ресунок

## 2.2 Отладка программам с помощью GDB :

* На этом шаге мы создали файл lab10-2.asm с текстом программы из ли- стинга 10.2.(рис. 6)

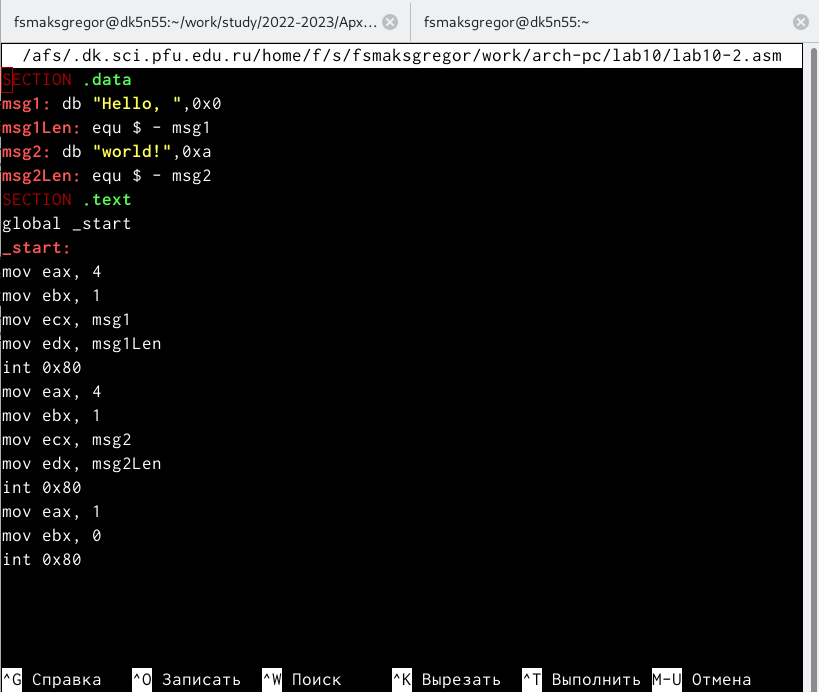


Рис. 6: Ресунок

* После этого мы скомпилировали файл, создали исполняемый файл.Затем мы загрузили исполняемый файл в отладчик GDM. (рис. 7)

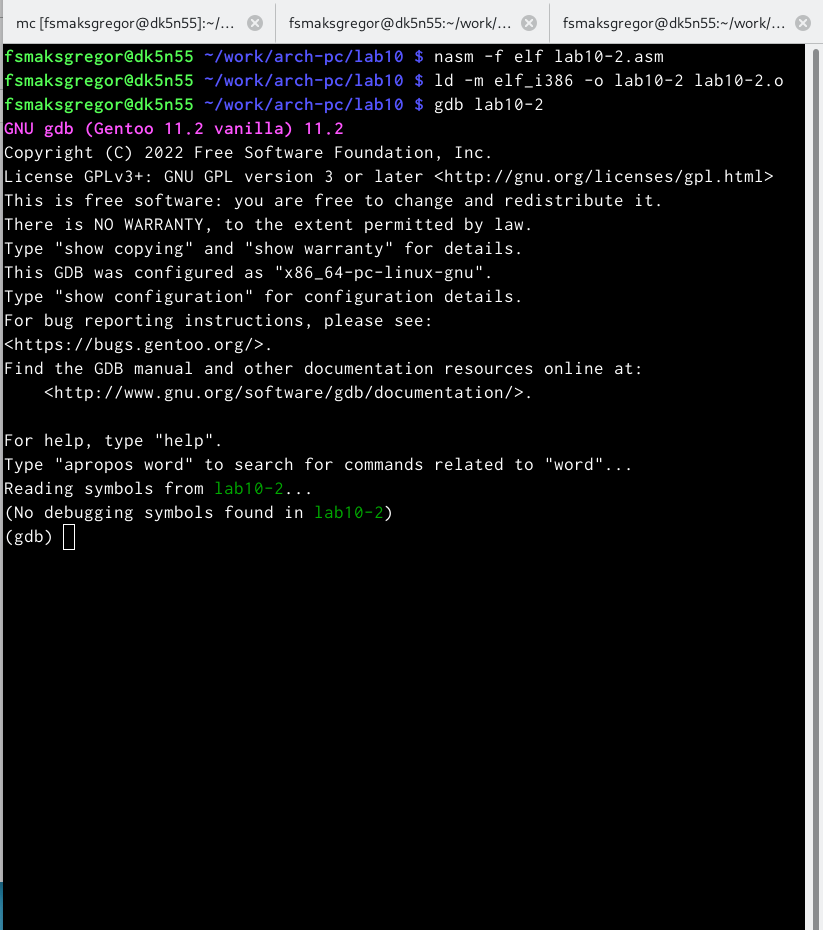


Рис. 7: Ресунок

* затем мы проверили работу программы, запустив ее в оболочке GDB с помощью команды run.(рис. 8)

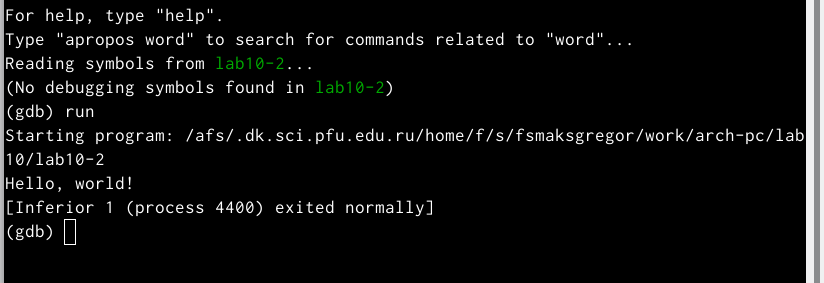


Рис. 8: Ресунок

* затем мы установили точку останова на метке \*\*\_start\*\*, которая запускает выполнение любой программы на ассемблере, и запустили ее.(рис. 9)

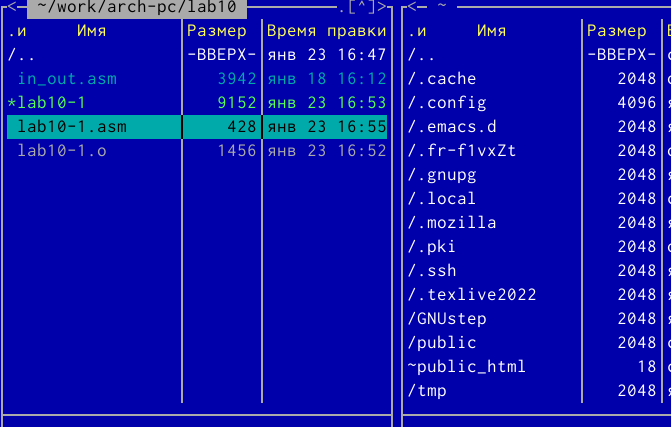


Рис. 9: Ресунок

* Затем мы просмотрели разобранный программный код, используя команду disassemble, начинающуюся с метки \*\*\_start\*\*. (рис. 10)

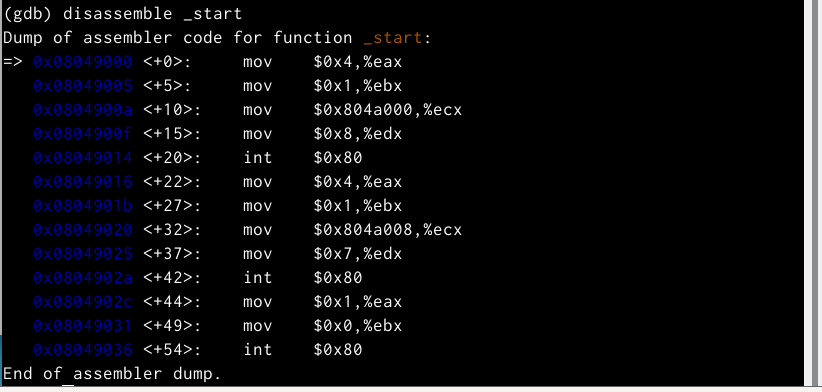


Рис. 10: Ресунок

* после этого мы переключились на отображение команд с синтаксисом Intel, введя команду set disassembly-flavor intel.(рис. 11)



Рис. 11: Ресунок

* Разница в синтаксисе между AT&T и INTEL заключается в том, что AT&T использует синтаксис mov $0x4,%eax, который популярен среди пользова- телей Linux, с другой стороны, INTEL использует синтаксис mov eax,0x4 , который является популярен среди пользователей Windows.
* Затем мы включили псевдографический режим для более удобного анализа программы. (рис. 12)

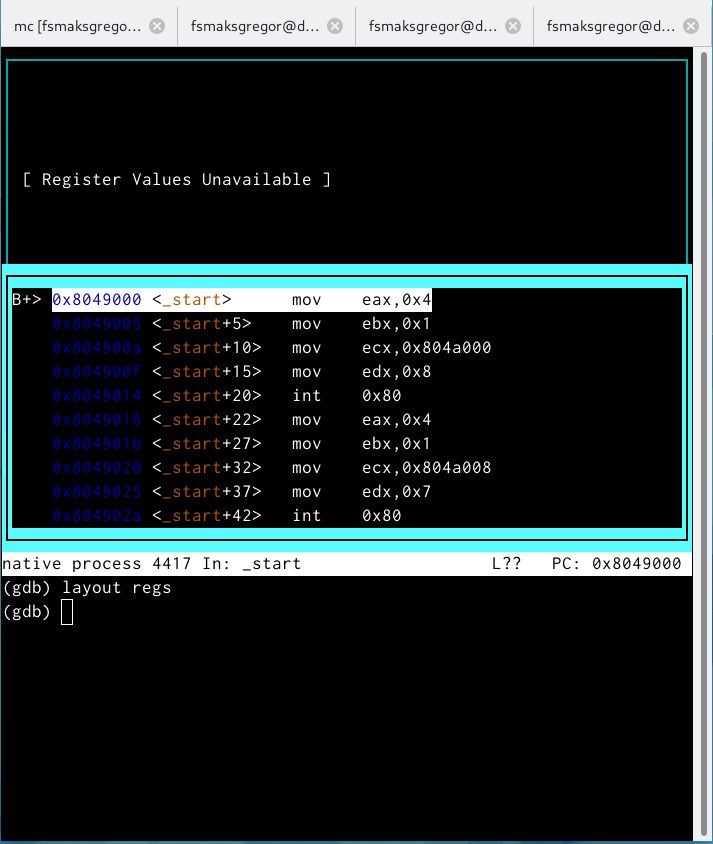


Рис. 12: Ресунок

## 2.3 Добавление точек останова :

* Мы проверили точку останова с помощью информационных точек останова.(рис. 13)

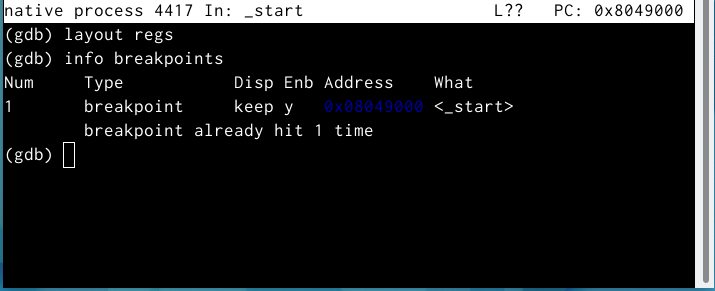


Рис. 13: Ресунок

* Mы определили адрес предпоследней инструкции (mov ebx,0x0) и устано- вили точку останова.(рис. 14)

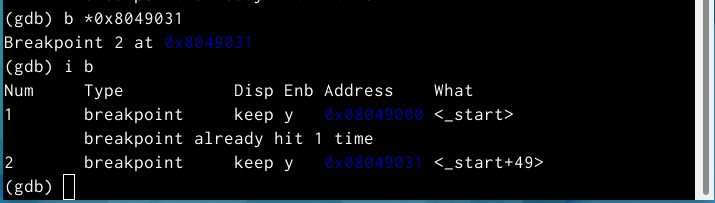


Рис. 14: Ресунок

## 2.4 Работа с данными программы в GDB :

* На этом шаге мы следовали 5 инструкциям, используя командный шаг i, и отслеживали изменение значений регистров, но перед этим мы проверили предыдущие значения регистров.(рис. 15)(рис. **¿fig:16?**)

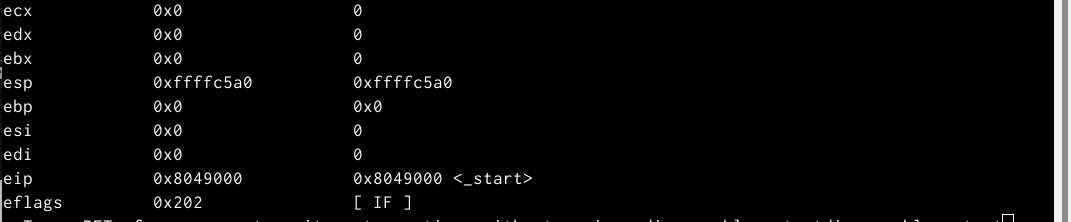
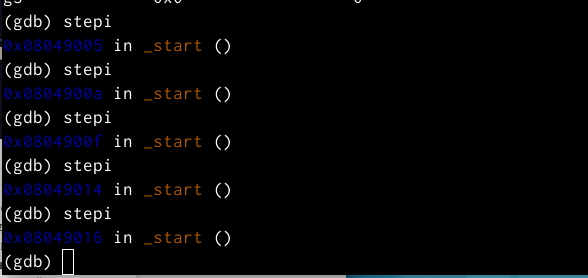
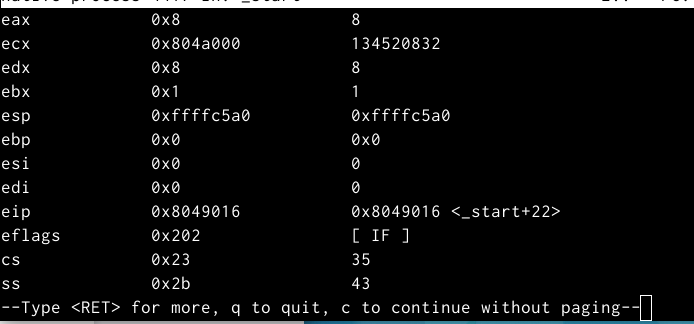


Рис. 15: Ресунок

* После проверки мы видим, что регистры : eax,ecx,edx,ebx,esp изменили свое значение.
* Mы рассмотрели значение переменной msg1 по имени, используя команду x/1sb.(рис. 16)

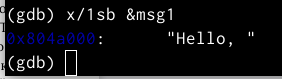


Рис. 16: Ресунок

* Здесь мы рассмотрели значение переменной msg2, используя адрес.(рис. 17)

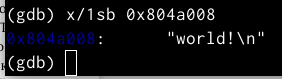


Рис. 17: Ресунок

* Здесь мы изменили первую букву переменной msg1, которая имеет тип char.(рис. 18)

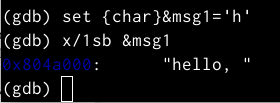


Рис. 18: Ресунок

* После этого мы изменили первую букву переменной msg2.(рис. 19)

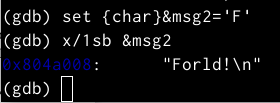


Рис. 19: Ресунок

* Затем мы выводим значение регистра edx в различных форматах (шестна- дцатеричном, двоичном и символьном).(рис. 20)



Рис. 20: Ресунок

* Используя команду set, мы изменили значение регистра ebx, когда раз, введя ‘2’, а в другой раз, введя 2.(рис. 21)

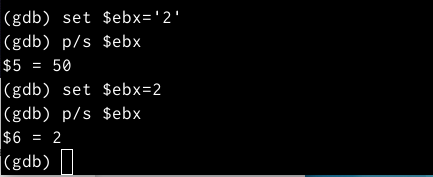


Рис. 21: Ресунок

* но когда мы напечатали значение регистра, мы получили значение 50 и это потому, что машина интерпретировала введенное значение как символ, и в таблице ASCII символ ‘2’ имеет значение 50 в десятичной системе, но когда мы ввели значение 2 машина интерпретировала 2 как число в десятичной системе.
* Наконец, мы завершили программу с помощью stepi и вышли из GDB с помощью команды quit.(рис. 22)

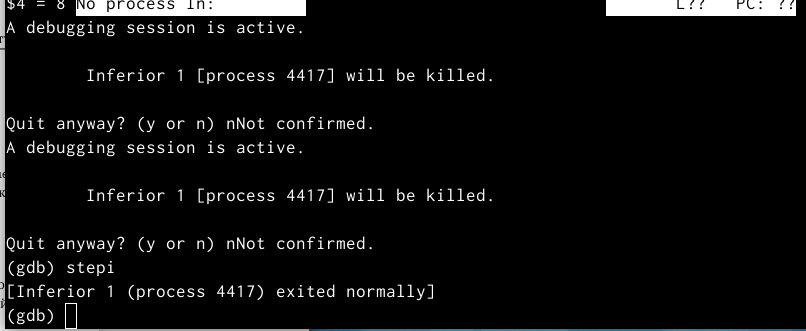


Рис. 22: Ресунок

## 2.5 Обработка аргументов командной строки в GDB :

* На этом этапе мы скопировали файл lab9-2.asm, созданный при выполне- нии лабораторной работы No9 с программой, отображающей аргументы командной строки на экране (листинг 9.2), в файл с именем lab 10-3.asm, а затем мы скомпилировали этот файл и установил точку останова в \*\*\_start\*\* и запустил отладчик.(рис. 23)

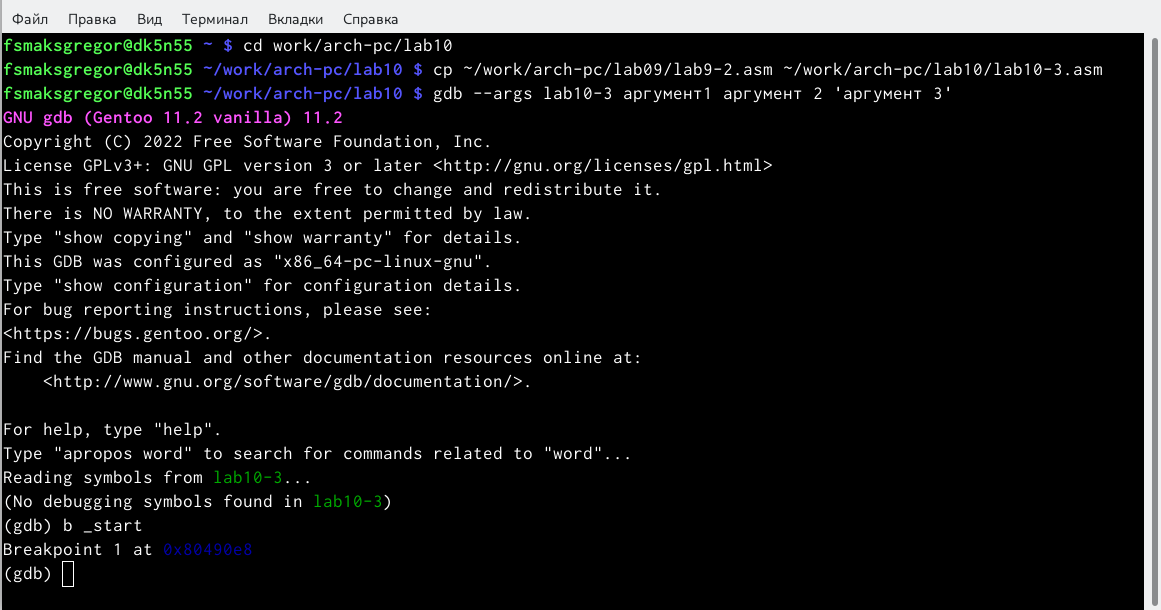


Рис. 23: Ресунок

* Затем мы посмотрели на остальные позиции стека – адрес в памяти, где находится имя программы, находится в [esp + 4], адрес первого аргумента хранится в [esp +8], в [esp +12].(рис. 24)

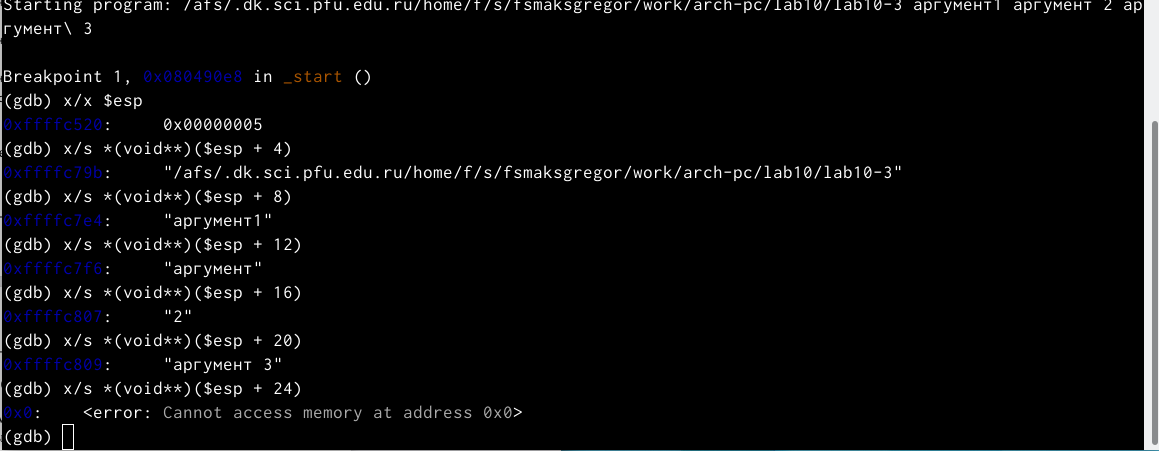


Рис. 24: Ресунок

* Шаг изменения адреса равен 4, потому что размер регистра esp равен 32би- там = 4 байтам, а количество памяти равно количеству аргументов плюс имя программы, поэтому мы получили 5 шагов с 4 байтами для каждого шага.

## 2.6 Выводы по результатам выполнения заданий :

* В этой части работы мы узнали, как работать с отладчиком GDB, и получили более близкое представление о том, как работают подпрограммы.

# 3 Задание для самостоятельной работы :

* Преобразуйте программу из лабораторной работы №9 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму. (рис. 25 26)

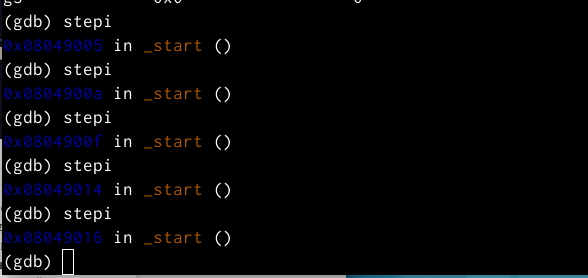


Рис. 25: Файл lab10-4.asm

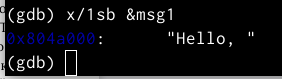


Рис. 26: Работа программы lab10-4.asm

1. В листинге 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.(рис. 27 28 29 **¿fig:021?**)

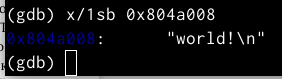


Рис. 27: код с ошибкой

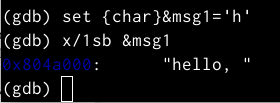


Рис. 28: отладка

Отметим, что перепутан порядок аргументов у инструкции add и что по окончании работы в edi отправляется ebx вместо eax

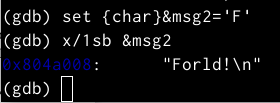


Рис. 29: код исправлен

## 3.1 Выводы по результатам выполнения заданий :

* В этой части мы узнали, как превратить программу в подпрограмму, но у нас возникла проблема с подпрограммой atoi , поэтому мы не смогли вычислить результат.

# 4 Выводы, согласованные с целью работы :

* В этой лабораторной работе мы научимся писать программы с использова- нием подпрограмм и познакомимся со способами отладки с использовани- ем GDB и его основными функциями.

# Список литературы