Отчёт по лабораторной работе №10

Уткина Алиина Дмитриевна

Содержание

# 1 Цель работы

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

# 2 Выполнение лабораторной работы

## 2.1 Реализация подпрограмм в NASM

Создадим каталог для выполнения лабораторной работы No 10, перейдйдем в него и создадим файл lab10-1.asm.

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. Внимательно изучим текст программы листинга 10.1 и введем его в созданный файл (рис. 1)

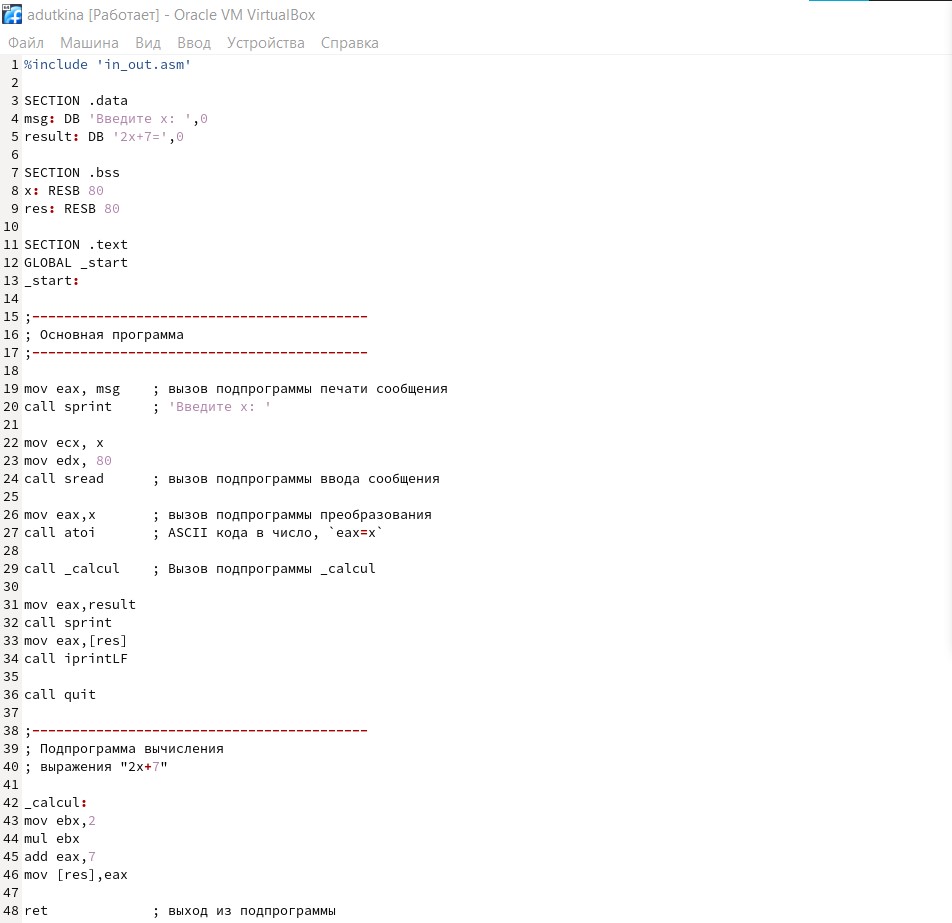


Рис. 1: Пример программы с использованием вызова подпрограммы

Первые строки программы отвечают за вывод сообщения на экран (call sprint), чтение данных введенных с клавиатуры (call sread) и преобразования введенных данных из символьного вида в численный (call atoi). После следующей инструкции call \_calcul, которая передает управление подпрограмме \_calcul, будут выполнены инструкции подпрограммы, написанные до ret. Инструкция ret является последней в подпрограмме и ее исполнение приводит к возвращению в основную программу к инструкции, следующей за инструкцией call, которая вызвала данную подпрограмму. Последние строки программы реализую вывод сообщения (call sprint), результата вычисления (call iprintLF) и завершение программы (call quit).

Создадим исполняемый файл и проверим его работу (рис. 2)



Рис. 2: Результат работы программы с вызовом подпрогрммы

Изменим 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 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 (рис. 3), (рис. 4).



Рис. 3: Добавление подрограммы в подпрограмму

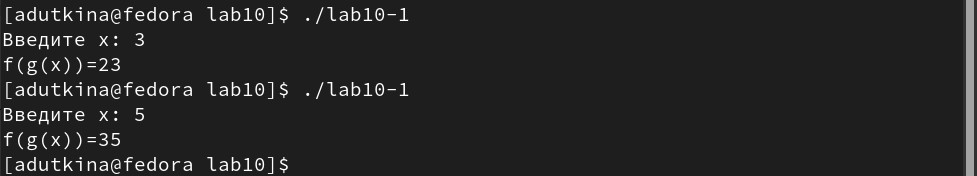


Рис. 4: Результат работы измененной программы

## 2.2 Отладка программам с помощью GDB

Создадим файл lab10-2.asm с текстом программы из Листинга 10.2 (рис. 5).

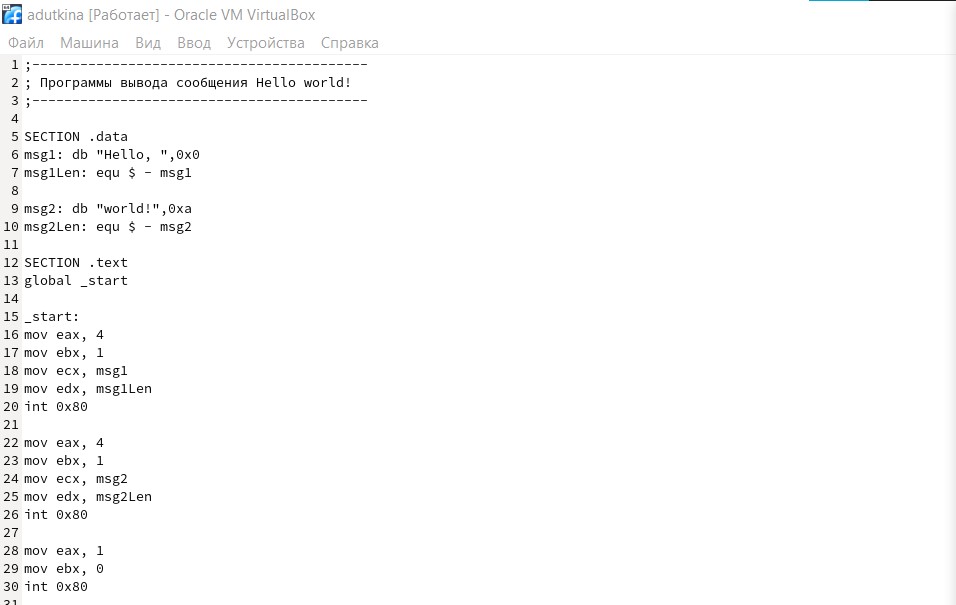


Рис. 5: Программа печати сообщения Hello world!

Получим исполняемый файл. Для работы с GDB в исполняемый файл добавим отладочную информацию, для этого трансляцию программ проводим с ключом ‘-g’, запустим исполняемый файл в отладчик GDB(рис. 6)

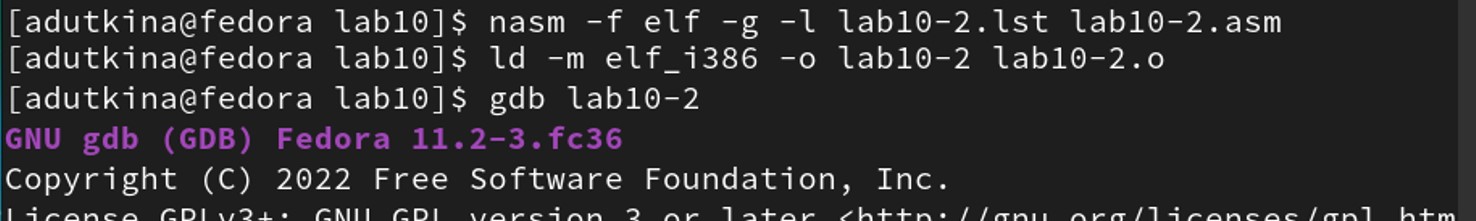


Рис. 6: Трансляция программы для работы с GDB

Проверим работу программы, запустив ее в оболочке GDB с помощью команды run (сокращённо r) (рис. 7).

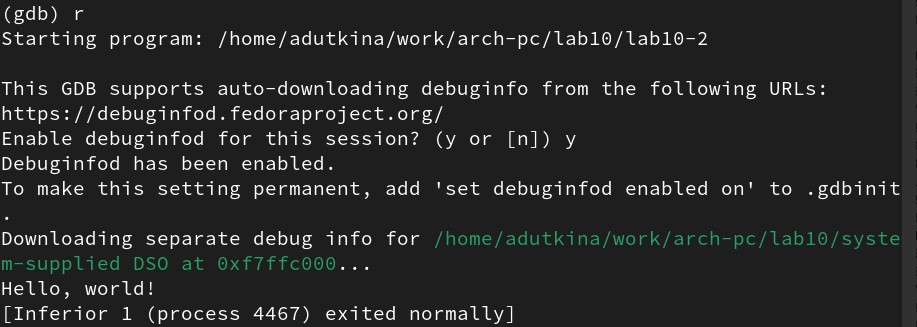


Рис. 7: Запуск программы в оболочке GDB

Для более подробного анализа программы установим брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запустим её. (рис. 8)



Рис. 8: Запуск программы в оболочке GDB с подробным анализом

Посмотрим дисассимилированный код программы с помощью команды disassemble начиная с метки \_start (рис. 9)

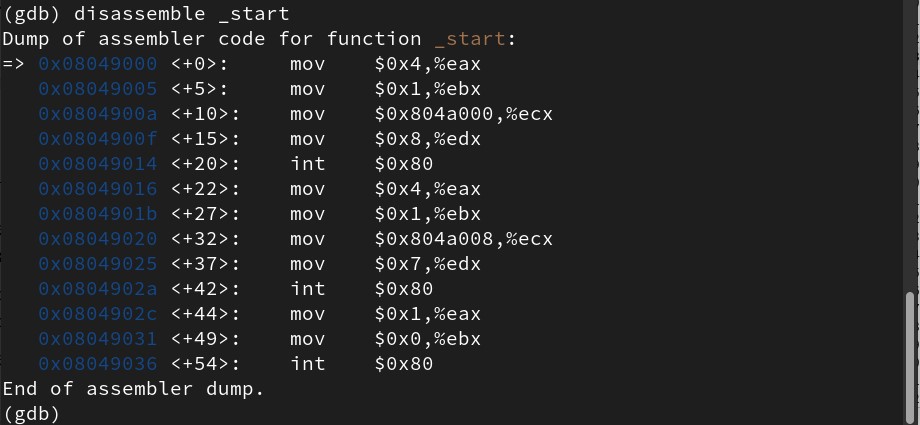


Рис. 9: Просмотр дисассимилированного кода программы

Переключимся на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel (рис. 10).



Рис. 10: Отображение команд с Intel’овским синтаксисом

Различия отображения синтаксиса машинных команд в режимах ATT и Intel в том, что во втором варианте опускается ‘%’ перед именами регистров и инструкции с несколькими операндами перечисляются в разном порядке: Intel в прямом, то есть как записано в программе, а ATT в обратном.

Включим режим псевдографики для более удобного анализа программы (рис. 11)

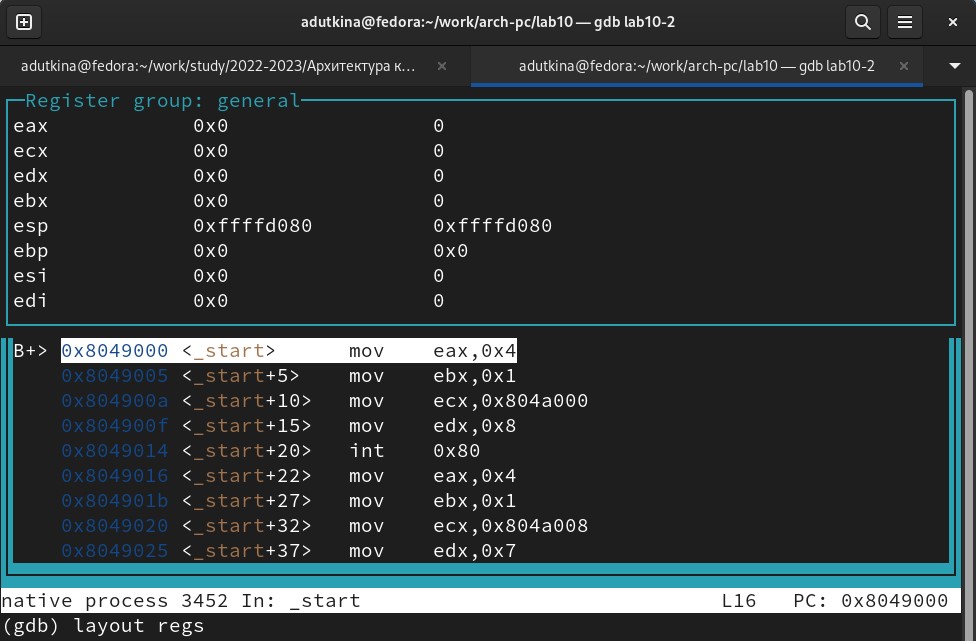


Рис. 11: Режим псевдографики

В этом режиме есть три окна: - В верхней части видны названия регистров и их текущие значения; - В средней части виден результат дисассимилирования программы; - Нижняя часть доступна для ввода команд.

### 2.2.1 Добавление точек останова

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)

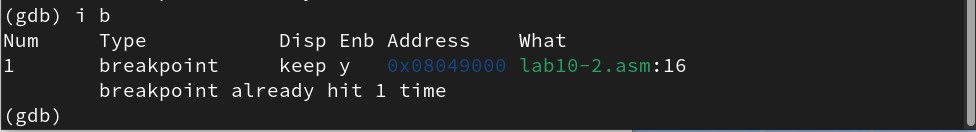


Рис. 12: Проверка установленной метки

Установим еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции. Установим точку останова для предпоследней инструкции (mov ebx,0x0), определив ее адрес (рис. 13)

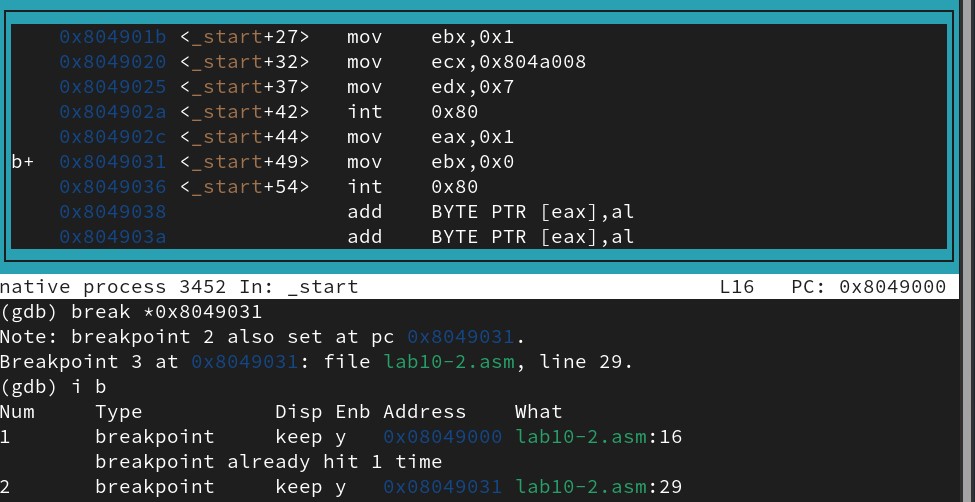


Рис. 13: Установка точки останова

### 2.2.2 Работа с данными программы в GDB

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Выполним 5 инструкций с помощью команды stepi (или si). На этих шагах изменяются значения регистров eax, ebx, ecx, edx и еще раз eax.

Посмотреть содержимое регистров также можно с помощью команды info registers (или i r) (рис. 14)

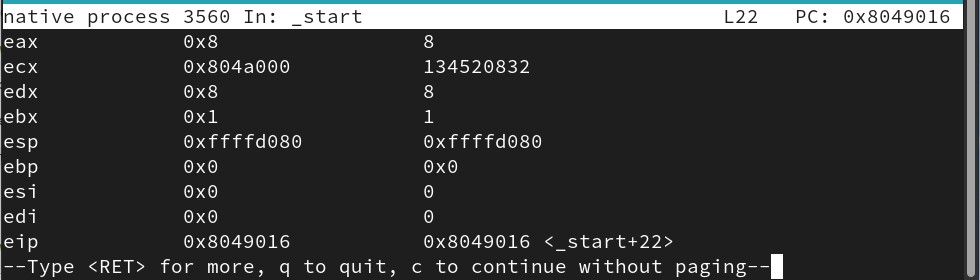


Рис. 14: Просмотр значения регистров

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 переменной msg1 по имени и значение переменной msg2 по адресу (рис. 15).

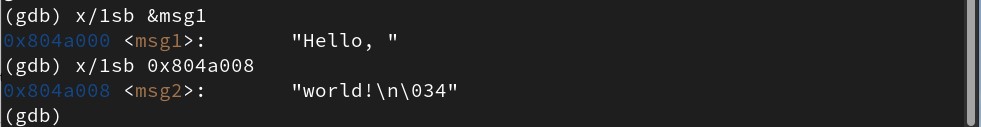


Рис. 15: Просмотр значения переменной

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. 16)

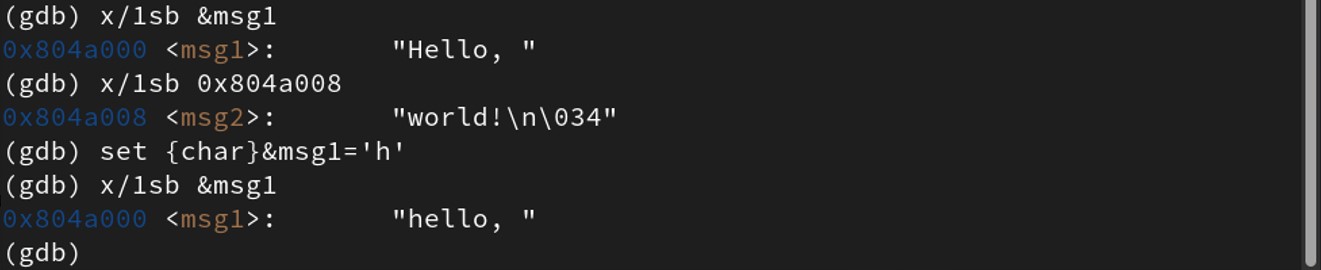


Рис. 16: Изменение символа ппеременной msg1

Заменим второй символ в переменной msg2 на заглавную букву (рис. 17).

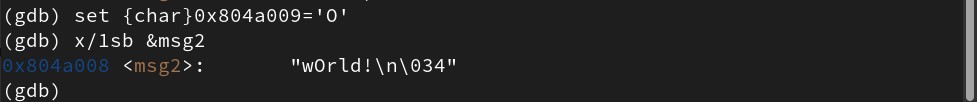


Рис. 17: Изменение символа переменной msg2

Чтобы посмотреть значения регистров используется команда print /F (перед именем регистра обязательно ставится префикс $) (рис. 18)



Рис. 18: Просмотр значения регистра

Выведем в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx. С помощью команды set изменим значение регистра ebx (рис. 19)



Рис. 19: Вывод значения регистра в различных форматах и их изменение

Завершим выполнение программы с помощью команды continue (сокращенно c) и выйдем из GDB с помощью команды quit (сокращенно q).

### 2.2.3 Обработка аргументов командной строки в GDB

Скопируем 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 создадим исполняемый файл.

Для загрузки в gdb программы с аргументами необходимо использовать ключ –args. Загрузим исполняемый файл в отладчик, указав аргументы 1, 2, 3.

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.

Для начала установим точку останова перед первой инструкцией в программе и запустим ее (рис. 20)

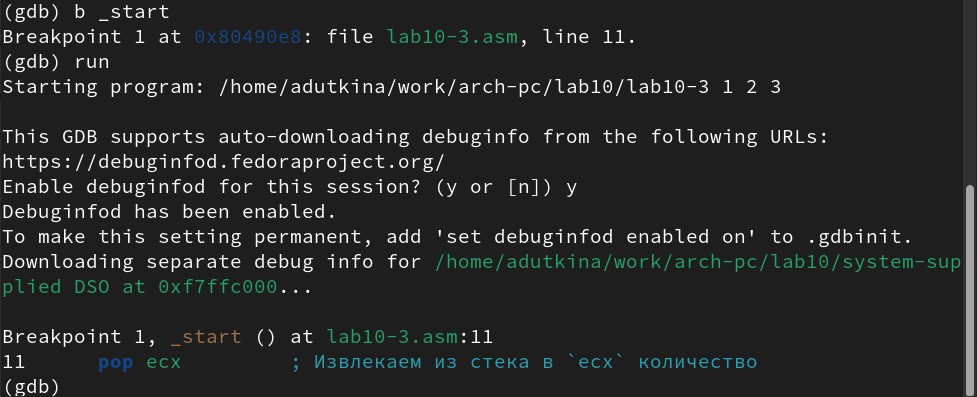


Рис. 20: Работа с программой lab10-3

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы). Как видно, число аргументов равно 4 - расположение программы и три аргумента.

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 аресу [esp+12] – второго и т.д. (рис. 21)

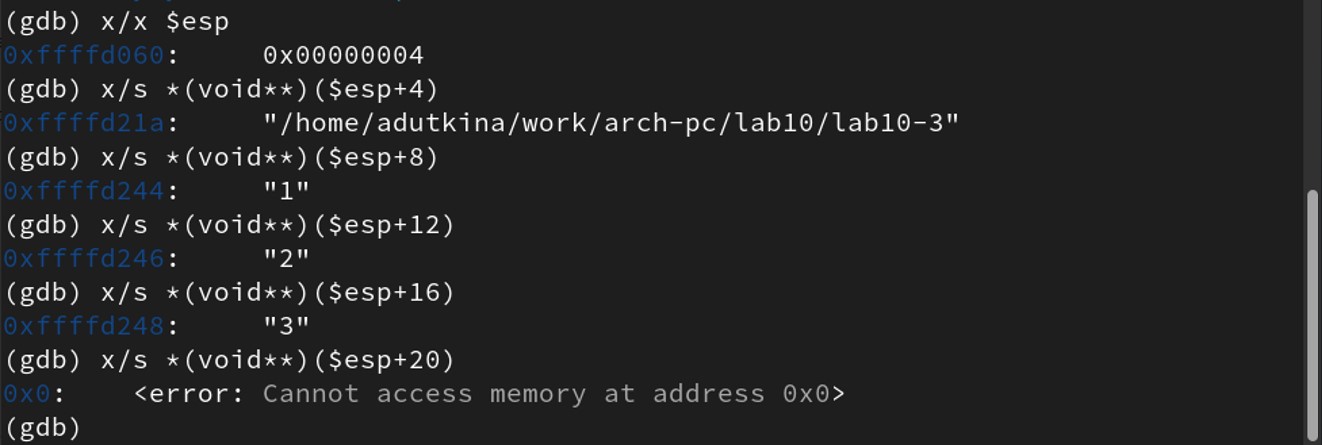


Рис. 21: Просмотр позиций стека

# 3 Выводы

В ходе лабораторной работы были приобретены навыки написания программ с использованием подпрограмм и изучены методы отладки при помощи GDB и его основные возможности.