Отчёта по лабораторной работе № 9

Простейший вариант

Сахно Алёна Юрьевна

Содержание

# 1 Цель работы

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

# 2 Задание

1. Теоретическая часть
2. Выполнение лабораторной работы
3. Выполнения самостоятельной работы

# 3 Теоретическое введение

9.2.1. Понятие об отладке

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 пре- рывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль).

Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить доволь- но 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

# 4 Выполнение лабораторной работы

1. Создайте каталог для выполнения лабораторной работы № 9, перейдите в него и со- здайте файл lab09-1.asm:

mkdir ~/work/arch-pc/lab09

cd ~/work/arch-pc/lab09

touch lab09-1.asm

(рис.1 1).

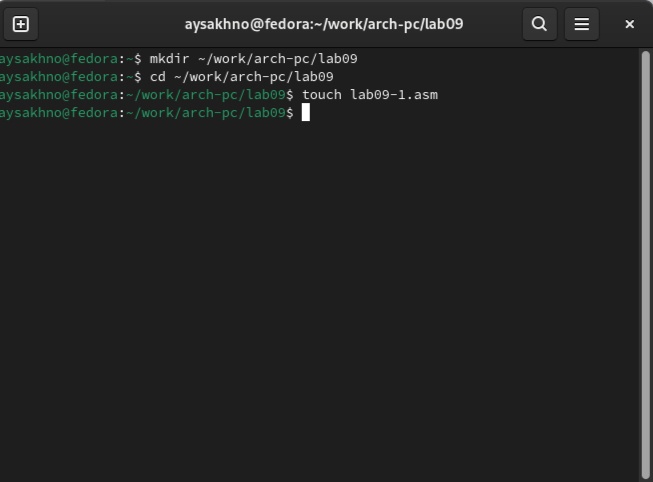


Рис. 1: Задали файл

1. 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 изучите текст программы (Листинг 9.1

(рис.2 2).

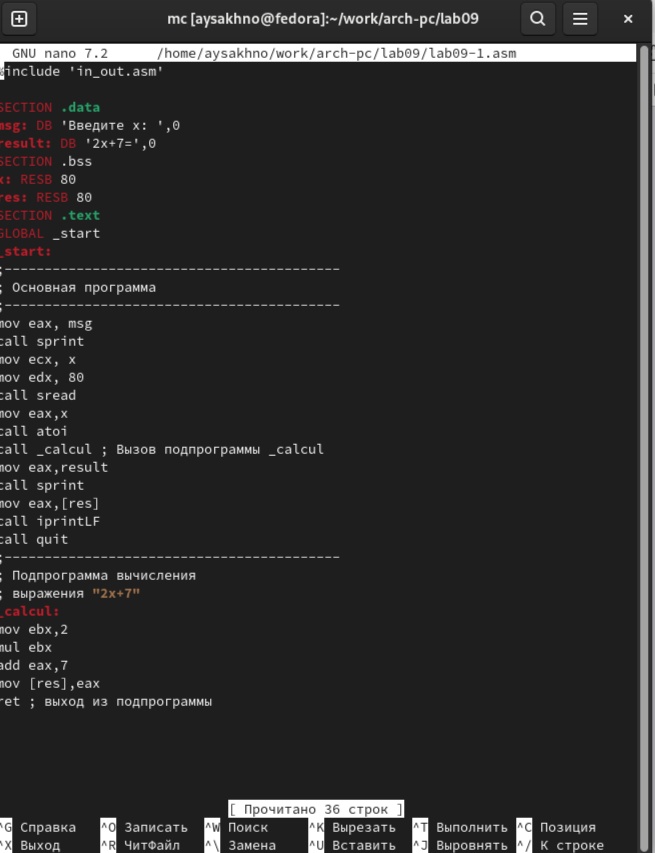


Рис. 2: Листинг 9.1. Пример программы с использованием вызова подпрограммы

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). mov eax, msg ; вызов подпрограммы печати сообщения call sprint ; ‘Введите x:’ mov ecx, x mov edx, 80 call sread ; вызов подпрограммы ввода сообщения mov eax,x ; вызов подпрограммы преобразования call atoi ; ASCII кода в число, eax=x

После следующей инструкции call \_calcul, которая передает управление подпрограмме \_calcul, будут выполнены инструкции подпрограммы:

mov ebx,2 mul ebx add eax,7 mov [res],eax ret

Инструкция ret является последней в подпрограмме и ее исполнение приводит к воз- вращению в основную программу к инструкции, следующей за инструкцией call, которая вызвала данную подпрограмму. Последние строки программы реализую вывод сообщения (call sprint), результата вы- числения (call iprintLF) и завершение программы (call quit). Введите в файл lab09-1.asm текст программы из листинга 9.1. Создайте исполняемый файл и проверьте его работу. Измените 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 в 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 3).

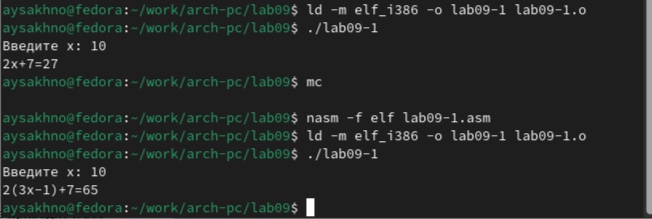


Рис. 3: Результат с изменением

9.4.2. Отладка программам с помощью GDB

Создайте файл lab09-2.asm с текстом программы из Листинга 9.2. (Программа печати сообщения Hello world!):

(рис.4 4).

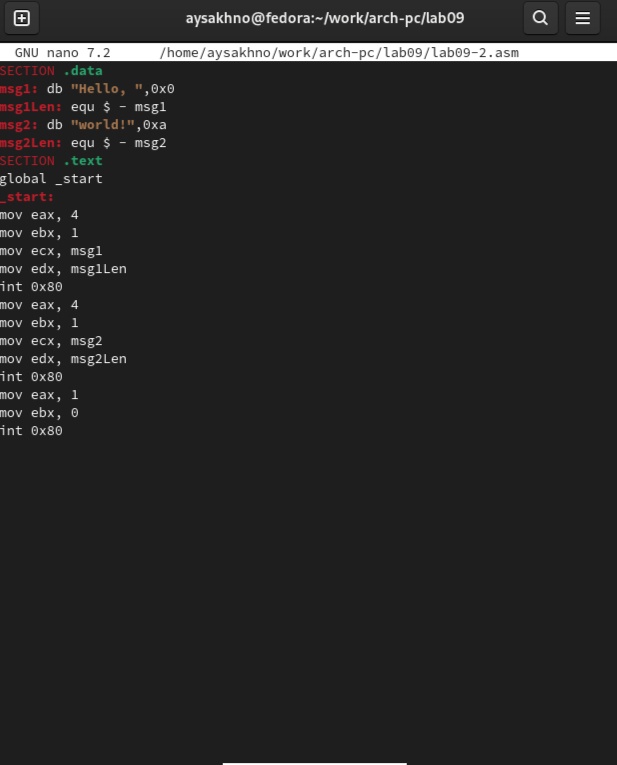


Рис. 4: Листинг 9.2. Программа вывода сообщения Hello world!

Получите исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’.

nasm -f elf -g -l lab09-2.lst lab09-2.asm

ld -m elf\_i386 -o lab09-2 lab09-2.o

Загрузите исполняемый файл в отладчик gdb:

user@dk4n31:~$ gdb lab09-2 Проверьте работу программы, запустив ее в оболочке GDB с помощью команды run (со- кращённо r):

(gdb) run Starting program: ~/work/arch-pc/lab09/lab09-2 Hello, world! [Inferior 1 (process 10220) exited normally] (gdb)

(рис.5 5).

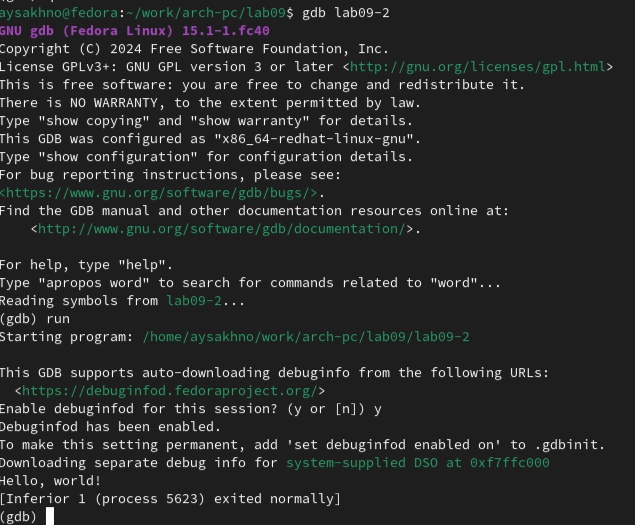


Рис. 5: Проверка работы

Для более подробного анализа программы установите брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запустите её. (gdb) break \_start

Breakpoint 1 at 0x8049000: file lab09-2.asm, line 12. (gdb) run Starting program: ~/work/arch-pc/lab09/lab09-2 Breakpoint 1, \_start () at lab09-2.asm:12 12 mov eax, 4

(рис.6 6).

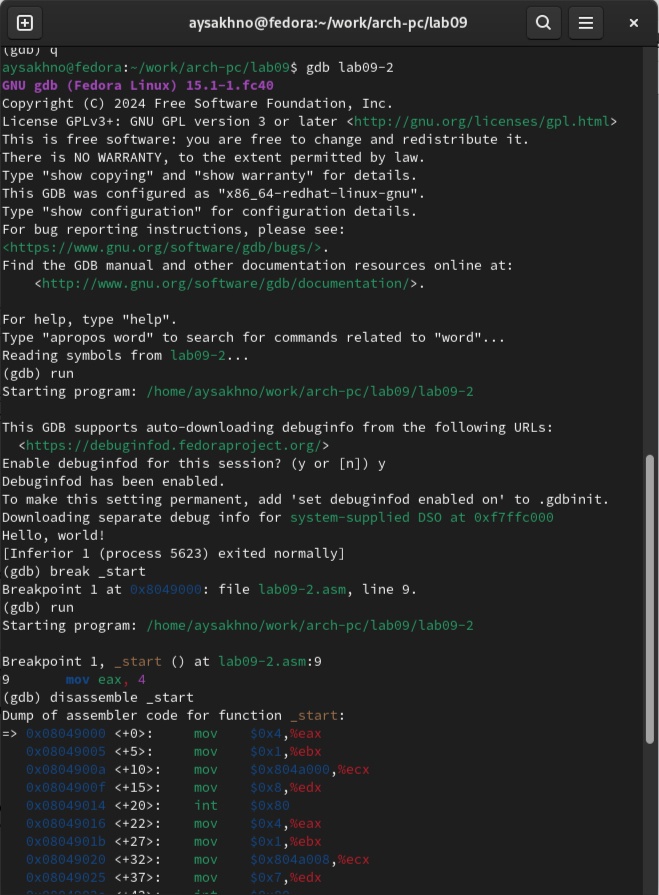


Рис. 6: Установка брейпоинта

Посмотрите дисассимилированный код программы с помощью команды disassemble начиная с метки \_start (gdb) disassemble \_start

Переключитесь на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel (gdb) set disassembly-flavor intel (gdb) disassemble \_start

Перечислите различия отображения синтаксиса машинных команд в режимах ATT и Intel. Включите режим псевдографики для более удобного анализа программы (рис. 9.2): (gdb) layout asm (gdb) layout regs 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.

9.4.2.1. 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). Про- верьте это с помощью команды info breakpoints (кратко i b): (gdb) info breakpoints

(рис.7 7).

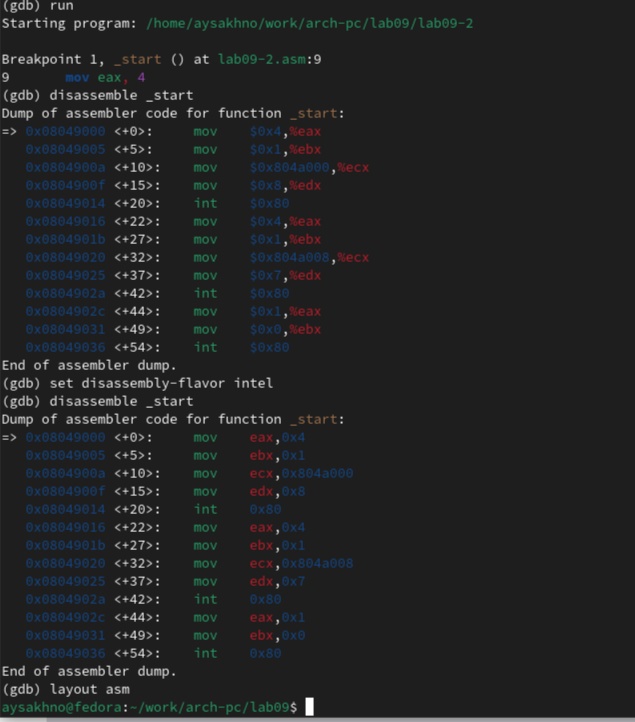


Рис. 7: Проверка

Установим еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции (см. рис. 9.3). Определите адрес предпоследней инструкции (mov ebx,0x0) и установите точку останова. (gdb) break \* Посмотрите информацию о всех установленных точках останова: (gdb) i b

(рис.8 7).

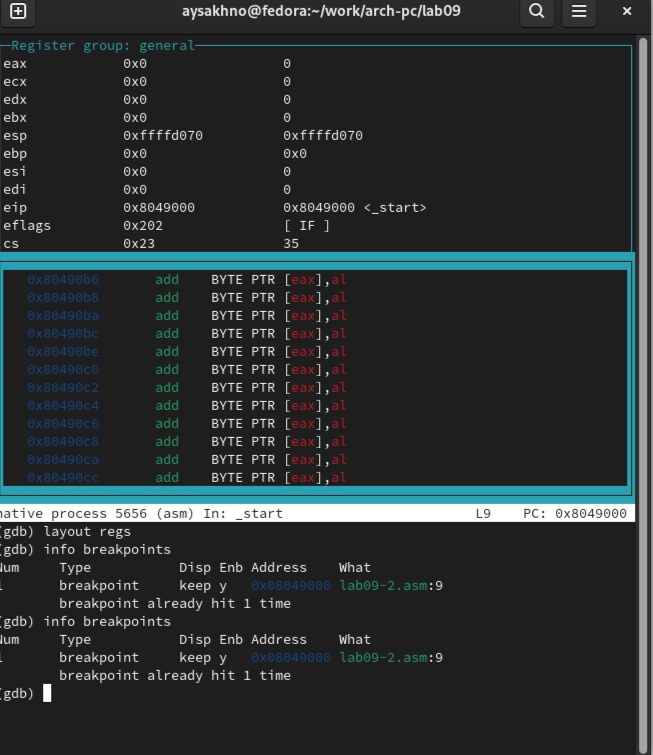


Рис. 8: Установка еще одной точки

9.4.2.2. Работа с данными программы в GDB

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных.

Выполните 5 инструкций с помощью команды stepi (или si) и проследите за изменением значений регистров. Значения каких регистров изменяются? Посмотреть содержимое регистров также можно с помощью команды info registers (или i r).

(gdb) info registers

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 пере- менной. Посмотрите значение переменной msg1 по имени (gdb) x/1sb &msg1 0x804a000 : “Hello,”

(рис.9 9).

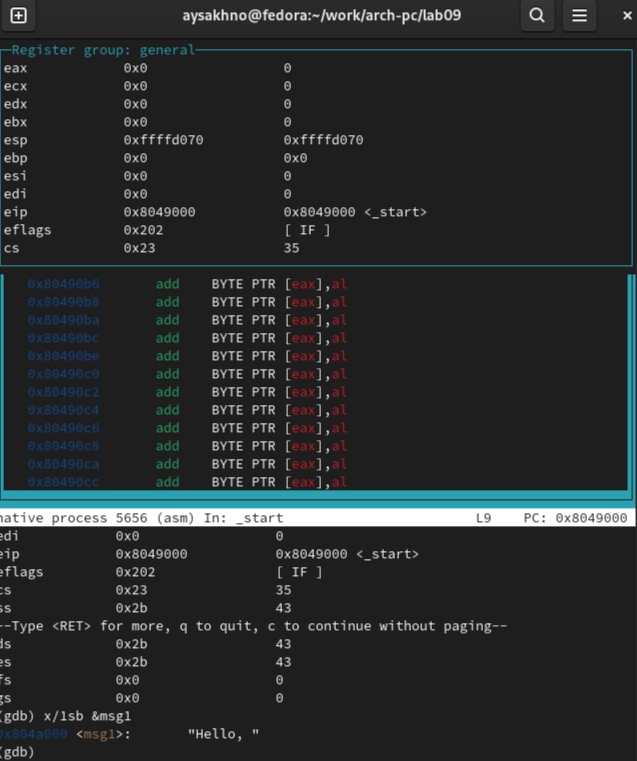


Рис. 9: Значение переменной msg1

Посмотрите значение переменной msg2 по адресу. Адрес переменной можно определить по дизассемблированной инструкции. Посмотрите инструкцию mov ecx,msg2 которая запи- сывает в регистр ecx адрес перемененной msg2 (рис. 9.4) Архитектура ЭВМ

(рис.10 10).

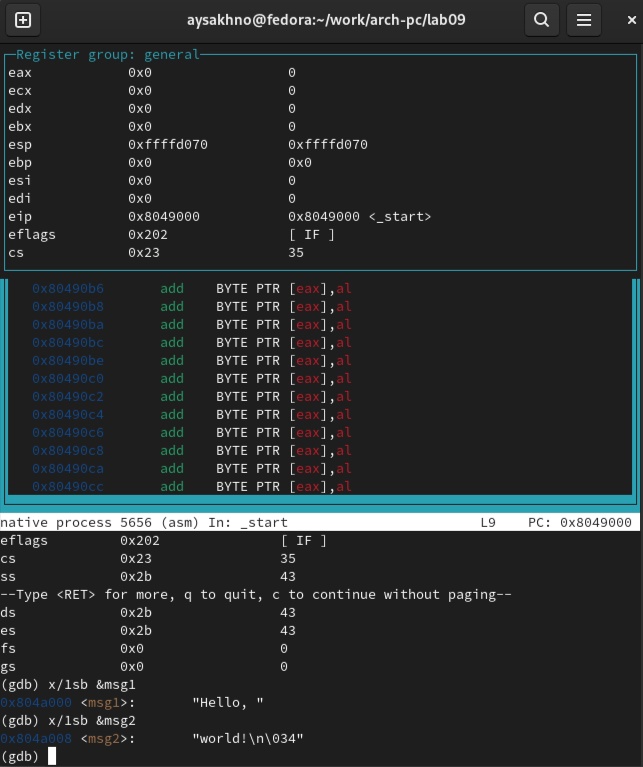


Рис. 10: Значение переменной msg2

Посмотрите значение переменной msg2 по адресу. Адрес переменной можно определить по дизассемблированной инструкции. Посмотрите инструкцию mov ecx,msg2 которая запи- сывает в регистр ecx адрес перемененной msg2 (рис. 9.4). Рис. 9.4. Отображение содержимого памяти 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). Измените первый символ переменной msg1 (рис. 9.5): (gdb) set {char}msg1=‘h’ (gdb) x/1sb &msg1 0x804a000 : “hello,” (gdb)

Замените любой символ во второй переменной msg2. Чтобы посмотреть значения регистров используется команда print /F (перед име- нем регистра обязательно ставится префикс $) (рис. 9.6): p/F $

(рис.11 11).



Рис. 11: Замение любой символ во второй переменной msg2.

Выведете в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx. С помощью команды set измените значение регистра ebx: (gdb) set $ebx=‘2’ (gdb) p/s $ebx

(рис.12 12).

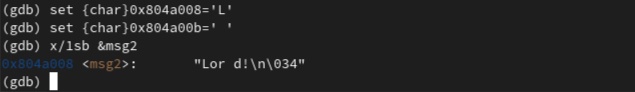


Рис. 12: Вывод а различных формах

$3 = 50 (gdb) set $ebx=2 (gdb) p/s $ebx $4 = 2 (gdb) Объясните разницу вывода команд p/s $ebx. Завершите выполнение программы с помощью команды continue (сокращенно c) или stepi (сокращенно si) и выйдите из GDB с помощью команды quit (сокращенно q).

9.4.2.3. Обработка аргументов командной строки в GDB

Скопируйте 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:

cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm

Создайте исполняемый файл.

nasm -f elf -g -l lab09-3.lst lab09-3.asm ld -m elf\_i386 -o lab09-3 lab09-3.o

Для загрузки в gdb программы с аргументами необходимо использовать ключ –args. Загрузите исполняемый файл в отладчик, указав аргументы:

gdb –args lab09-3 аргумент1 аргумент 2 ‘аргумент 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.

(gdb) b \_start (gdb) run

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

(gdb) x/x $esp 0xffffd200: 0x05

Как видно, число аргументов равно 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’.

Посмотрите 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.

(gdb) x/s \*(void\*\*)($esp + 4) 0xffffd358: "~/lab09-3" (gdb) x/s \*(void\*\*)($esp + 8) 0xffffd3bc: “аргумент1” (gdb) x/s \*(void\*\*)($esp + 12) 0xffffd3ce: "аргумент" (gdb) x/s \*(void\*\*)($esp + 16) 0xffffd3df: “2” (gdb) x/s \*(void\*\*)($esp + 20) 0xffffd3e1: "аргумент 3" (gdb) x/s \*(void\*\*)($esp + 24) 0x0: <error: Cannot access memory at address 0x0> (gdb) Объясните, почему шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12] и т.д.) (рис.13 13).

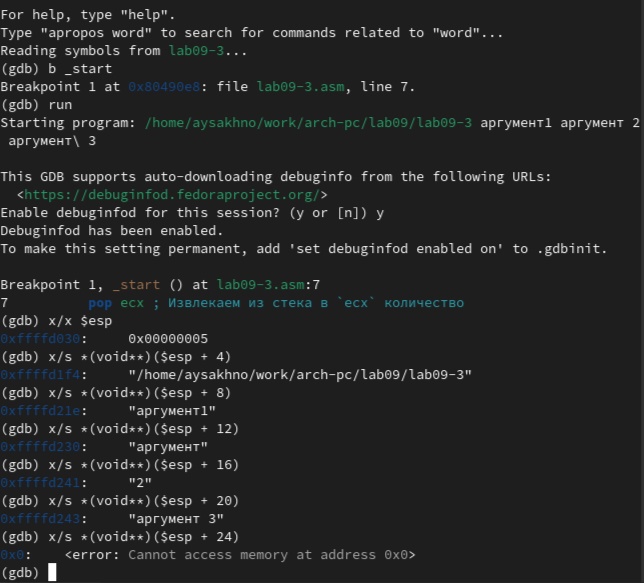


Рис. 13: Объяснение

# 5 9.5. Задание для самостоятельной работы

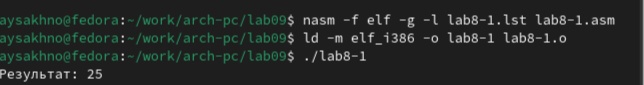
1. Преобразуйте программу из лабораторной работы №8 (Задание №1 для самостоятель- ной работы), реализовав вычисление значения функции 𝑓(𝑥) как подпрограмму.
2. В листинге 9.3 приведена программа вычисления выражения (3 + 2) ∗ 4 + 5. При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее

(рис.14 14).



Рис. 14: Листинг 9.3. Программа вычисления выражения (3 + 2) ∗ 4 + 5

(рис.15 **¿fig:015?**).

 # Выводы

Я приобрела навык написания программ с использованием подпрограмм. Познакомилась с методами отладки при помощи GDB и его основными возможностями

# Список литературы

::: {#refs} ::: https://esystem.rudn.ru/pluginfile.php/2089096/mod\_resource/content/0/%D0%9B%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%20%E2%84%969.%20%D0%9F%D0%BE%D0%BD%D1%8F%D1%82%D0%B8%D0%B5%20%D0%BF%D0%BE%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D1%8B.%20%D0%9E%D1%82%D0%BB%D0%B0%D0%B4%D1%87%D0%B8%D0%BA%20..pdf