Отчёт по лабораторной работе №10

Понятие подпрограммы. Отладчик GDB

Аскеров Александр Эдуардович

Содержание

[1 Цель работы 1](#__RefHeading___Toc378_2168413795)

[2 Выполнение лабораторной работы 1](#__RefHeading___Toc380_2168413795)

[2.1 Реализация подпрограмм в NASM 1](#__RefHeading___Toc382_2168413795)

[2.2 Отладка программам с помощью GDB 2](#__RefHeading___Toc384_2168413795)

[2.2.1 Добавление точек останова 5](#__RefHeading___Toc386_2168413795)

[2.2.2 Работа с данными программы в GDB 6](#__RefHeading___Toc388_2168413795)

[2.2.3 Обработка аргументов командной строки в GDB 9](#__RefHeading___Toc390_2168413795)

[2.3 Задания для самостоятельной работы 11](#__RefHeading___Toc392_2168413795)

[3 Выводы 13](#__RefHeading___Toc394_2168413795)

# 1 Цель работы

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

# 2 Выполнение лабораторной работы

## 2.1 Реализация подпрограмм в NASM

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

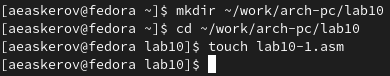


Рис. 1: Создание каталога lab10 и файла lab10-1.asm

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 изучим текст программы.

Введём в файл lab10-1.asm текст программы из листинга 10.1. Создадим исполняемый файл и проверим его работу.

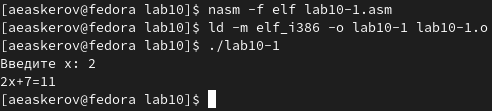


Рис. 2: Проверка работы программы lab10-1.asm

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 передаётся в подпрограмму \_calcul из неё 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.

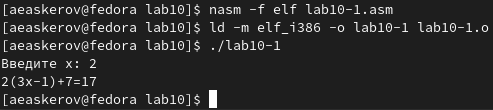


Рис. 3: Проверка работы программы lab10-1.asm (вывод f(g(x)))

## 2.2 Отладка программам с помощью GDB

Создадим файл lab10-2.asm с текстом программы из Листинга 10.2. (Программа печати сообщения Hello world!).

Рис. 4: Создание файла lab10-2.asm

Рис. 4: Создание файла lab10-2.asm

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

Рис. 5: Трансляция с ключом –g

Рис. 5: Трансляция с ключом –g

Загрузим исполняемый файл в отладчик gdb.

Рис. 6: Загрузка исполняемого файла в отладчик gdb

Рис. 6: Загрузка исполняемого файла в отладчик gdb

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

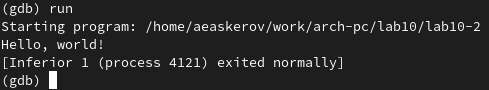


Рис. 7: Проверим работу программы lab10-2.asm в оболочке GDB

Для более подробного анализа программы установим брейкпойнт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запустим её.

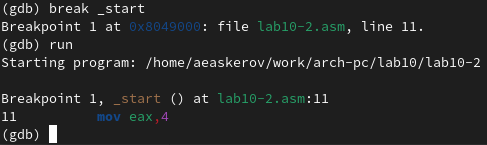


Рис. 8: Более подробный анализ программы с брейкпойнт

Посмотрим дизассемблированный код программы с помощью команды disassemble, начиная с метки \_start.



Рис. 9: Дизассемблированный код программы

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

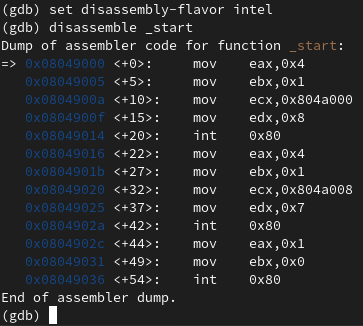


Рис. 10: Переключение на отображение команд с Intel’овским синтаксисом

Перечислим различия отображения синтаксиса машинных команд в режимах ATT и Intel.

В ATT сначала записывается адрес, потом регистр, перед адресом регистра ставится $, перед названием регистра %. В Intel сначала регистр, потом адрес и перед ними ничего не ставится.

Включим режим псевдографики для более удобного анализа программы.

Рис. 11: Включение режима псевдографики

Рис. 11: Включение режима псевдографики

### 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).

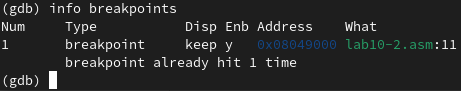


Рис. 12: Проверка наличия установки точки останова

Установим ещё одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции. Определим адрес предпоследней инструкции (mov ebx,0x0) и установим точку останова.

Рис. 13: Установка точки останова

Рис. 13: Установка точки останова

Посмотрим информацию о всех установленных точках останова.

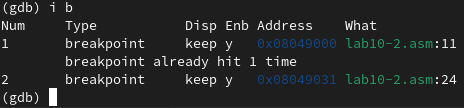


Рис. 14: Информация о всех установленных точках останова

### 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) и проследим за изменением значений регистров. Изменяются значения регистров eax, ebx, ecx и edx.

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

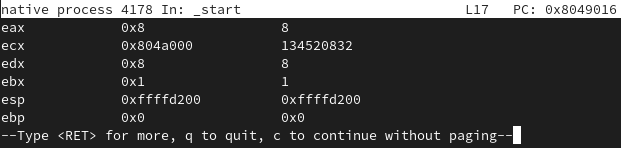


Рис. 15: Просмотр содержимого регистров

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.

Посмотрите значение переменной msg1.

Рис. 16: Просмотр значения msg1

Рис. 16: Просмотр значения msg1

Посмотрим значение переменной msg2 по адресу. Адрес переменной можно определить по дизассемблированной инструкции. Посмотрим инструкцию mov ecx,msg2 которая записывает в регистр ecx адрес перемененной msg2.

Рис. 17: Просмотр значения msg2

Рис. 17: Просмотр значения msg2

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.

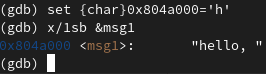


Рис. 18: Изменение первого символа переменной msg1

Заменим любой символ во второй переменной msg2.

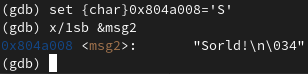


Рис. 19: Изменение первого символа переменной msg2

Чтобы посмотреть значения регистров используется команда print /F val (перед именем регистра обязательно ставится префикс $): p/F $<регистр>.

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

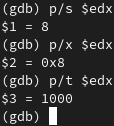


Рис. 20: Вывод в различных форматах значения регистра edx

С помощью команды set изменим значение регистра ebx.

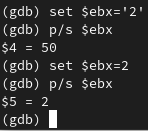


Рис. 21: Изменение значения регистра ebx с помощью команды set

Разница в выводе команд p/s $ebx заключается в том, что 50 – это номер символа 2 в таблице ASCII, в то время как 2 – это просто число 2.

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



Рис. 22: Завершение выполнения программы с помощью команды continue

### 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 (Листинг 9.2), в файл с именем lab10-3.asm.

Рис. 23: Копирование lab9-2.asm в каталог lab10 под именем lab10-3.asm

Рис. 23: Копирование lab9-2.asm в каталог lab10 под именем lab10-3.asm

Создадим исполняемый файл.

Рис. 24: Создание исполняемого файла

Рис. 24: Создание исполняемого файла

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

Рис. 25: Загрузка исполняемого файла в отладчик с указыванием аргументов

Рис. 25: Загрузка исполняемого файла в отладчик с указыванием аргументов

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 её.

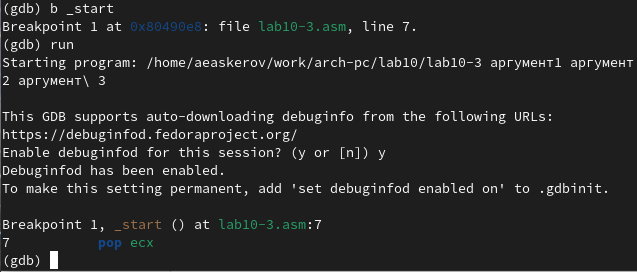


Рис. 26: Установка точки и её запуск

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

Рис. 27: Адрес вершины стека

Рис. 27: Адрес вершины стека

Как видно, число аргументов равно 5 – это имя программы lab10-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’. Посмотрим остальные позиции стека – по адесу [esp + 4] располагается адрес в памяти, где находится имя программы, по адесу [esp + 8] хранится адрес первого аргумента, по аресу [esp + 12] – второго и т.д.

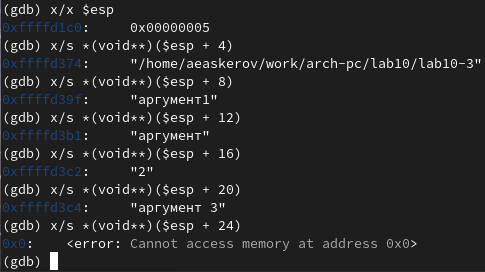


Рис. 28: Просмотр остальных позиций стека

Шаг равен четырём из-за того, что на позицию, которая хранит элемент стека, выделяется 4 байта.

## 2.3 Задания для самостоятельной работы

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

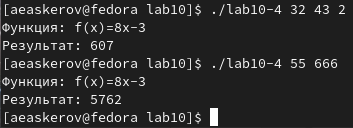


Рис. 29: Результат работы программы lab10-4.asm

1. В листинге 10.3 приведена программа вычисления выражения (3 + 2) \* 4 + 5. При запуске данная программа даёт неверный результат. Проверим это. С помощью отладчика GDB, анализируя изменения значений регистров, определим ошибку и исправим её.

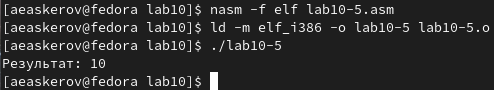


Рис. 30: Проверка работы программы lab10-5.asm (неисправленной)

Результат действительно не соответствует ожидаемому.

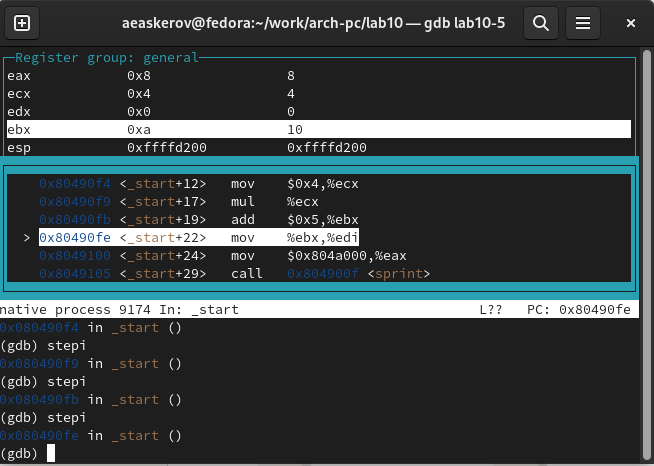


Рис. 31: Работа с отладчиком

С помощью GDB были пошагово просмотрены шаги выполнения программы. Выяснилось, что регистр ecx умножается на изначальное значение регистра eax, а не на значение, полученное после сложения eax и ebx. Причина этого в том, что результат сложения сохраняется в ebx, в то время как ecx умножается на eax, который остался равным двум. Потом, вместо того чтобы прибавить 5 к eax (который должен был бы равняться 20, а равняется 2), 5 прибавляется к регистру ebx, равному пяти. Таким образом, мы получаем 8 и, впоследствии, 10, вместо 20 и, впоследствии, 25. Также нужно помещать в регистр edi, предназначенный для хранения ответа, регистр eax, а не регистр ebx.

Теперь исправим выявленные ошибки и получим верный ответ.

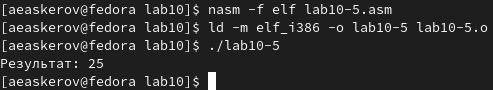


Рис. 32: Проверка работы программы lab10-5.asm (исправленной)

# 3 Выводы

Приобретены навыки написания программ с использованием подпрограмм. Изучены методы отладки при помощи GDB и его основные возможности.