Лабораторная работа №10

Архитектура компьютера

Голованова Мария Константиновна

Содержание

# 1 Цель работы

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

# 2 Задание

Создать программы, вычисляющие результат с использованием подпрограмм. Исследовать программы с помощью GDB.

# 3 Теоретическое введение

Подпрограмма — часть компьютерной программы, содержащая описание определённого набора действий, которая может быть многократно вызвана из разных частей программы.

Отладка — процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа: обнаружение ошибки, поиск её местонахождения, определение причины ошибки и исправление ошибки. Наиболее часто применяют такие методы отладки, как создание точек контроля значений на входе и выходе участка программы (например, вывод промежуточных значений на экран) или использование специальных программ-отладчиков. Отладчики позволяют управлять ходом выполнения программы, контролировать и изменять данные, что помогает быстрее найти место ошибки в программе и ускорить её исправление. Они позволяют увидеть, что происходит «внутри» программы в момент её выполнения или что делает программа в момент сбоя. Наиболее известным отладчиком для Linux является программа GNU GDB (GNU Debugger — отладчик проекта GNU). Он работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования. Самые популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам.

# 4 Выполнение лабораторной работы

## 4.1 Реализация подпрограмм в NASM

Я создала каталог для выполнения лабораторной работы No 10, перешла в него и создала файл lab10-1.asm (рис. 1).

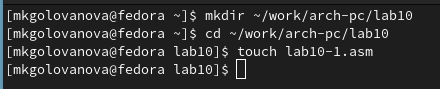


Рис. 1: Создание каталога для программам лабораторной работы №10 и файла lab10-1.asm

В качестве примера я рассмотрела 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. Я внимательно изучила текст программы из листинга 10.1, ввела его в файл lab10-1.asm, создала исполняемый файл и проверила его работу (рис. 2, рис. 3).

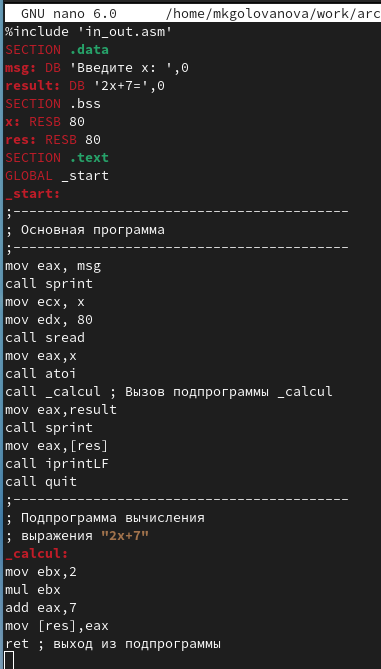


Рис. 2: Введение текста программы из листинга 10.1

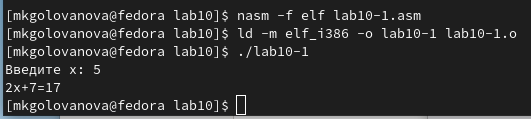


Рис. 3: Создание и проверка исполняемого файла lab10-1

Я изменила 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. 4, рис. 5, рис. 6).

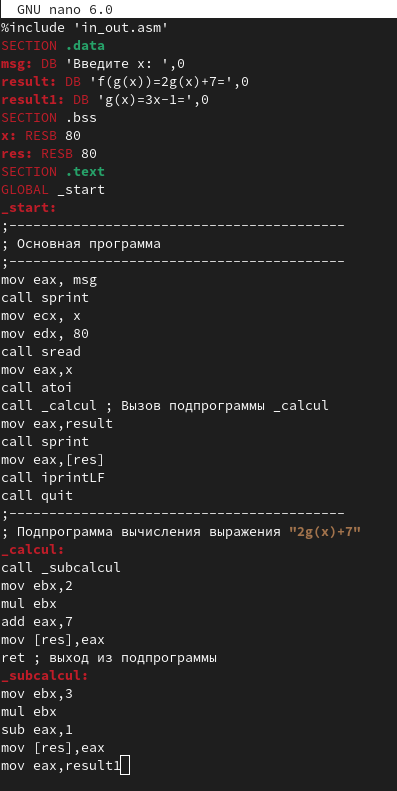


Рис. 4: Изменение текста программы в файле lab10-1.asm

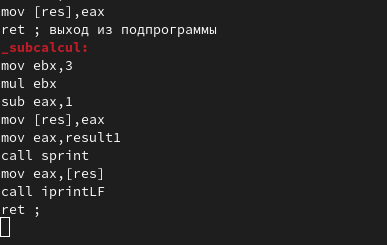


Рис. 5: Изменение текста программы в файле lab10-1.asm

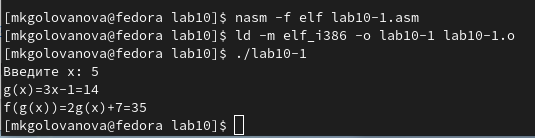


Рис. 6: Результат работы изменённого файла lab10-1.asm

## 4.2 Отладка программам с помощью GDB

Я создала файл lab10-2.asm и ввела в него текст программы из Листинга 10.2 (Программа печати сообщения Hello world!) (рис. 7, рис. 8).

Рис. 7: Создание файла lab10-2.asm

Рис. 7: Создание файла lab10-2.asm

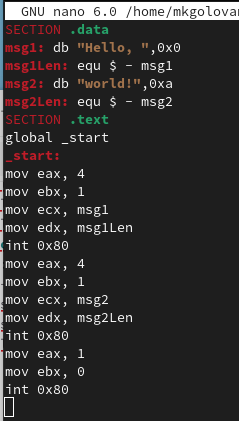


Рис. 8: Введение текста программы из листинга 10.2

Я получила исполняемый файл (рис. 9). Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’.

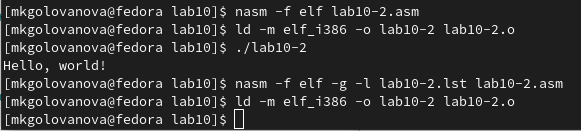


Рис. 9: Создание исполняемого файла lab10-2

Я загрузила исполняемый файл в отладчик gdb (рис. 10).



Рис. 10: Загрузка исполняемого файла в отладчик gdb

Я проверила работу программы, запустив ее в оболочке GDB с помощью команды run (сокращённо r) (рис. 11).

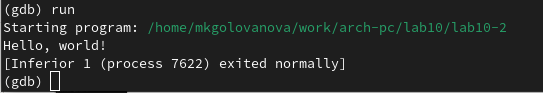


Рис. 11: Проверка работы программы в оболочке GDB

Для более подробного анализа программы я установила брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запустила программу (рис. 12).

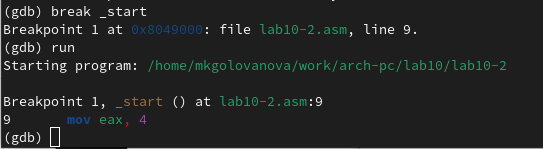


Рис. 12: Установка брейкпоинта на метку \_start и запуск программы

Я посмотрела дисассимилированный код программы с помощью команды disassemble начиная с метки \_start (рис. 13).



Рис. 13: Просмотр дисассимилированного кода программы начиная с метки \_start с помощью команды disassemble

Я переключилась на отображение команд с синтаксисом Intel, введя команду set disassembly-flavor intel (рис. 14).

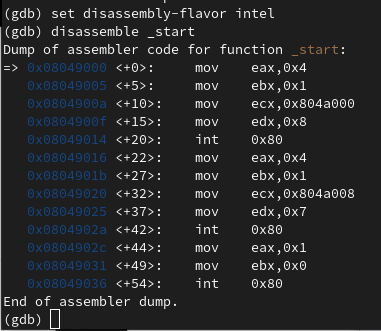


Рис. 14: Переключение на отображение команд с синтаксисом Intel

При отображении синтаксиса машинных команд регистр и его значение в режиме ATT указывается как $,%, а в режиме Intel - ,.

Я включила режим псевдографики для более удобного анализа программы (рис. 15): (gdb) layout asm (gdb) layout regs

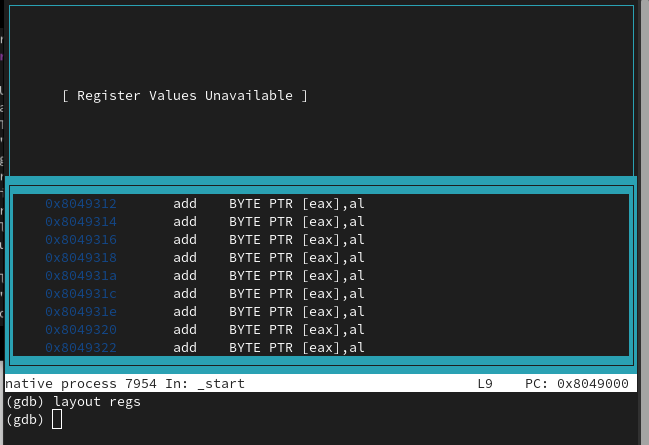


Рис. 15: Режим псевдографики

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.

### 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). Я проверила это с помощью команды info breakpoints (кратко i b). Я определила адрес предпоследней инструкции (mov ebx,0x0), установила точку останова и посмотрела информацию о всех установленных точках останова (рис. 16).

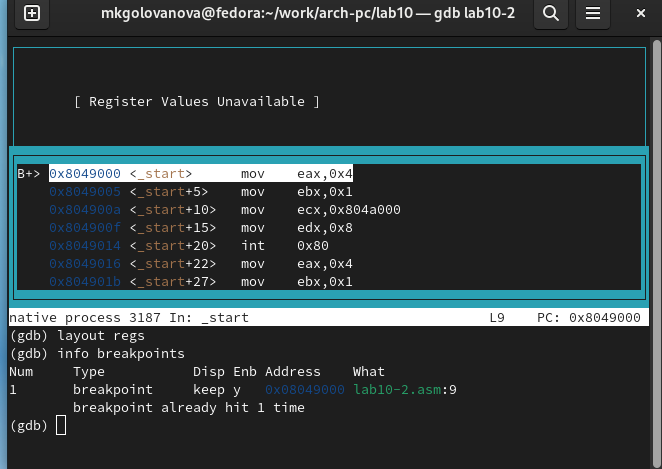


Рис. 16: Проверка наличия точки останова, установка новой точки останова, проверка наличия точек

### 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. Я выполнила 5 инструкций с помощью команды stepi (или si) и проследила за изменением значений регистров. Значения изменяются в регистрах eax, ebx, ecx, edx, eax (повторно) (рис. 17).

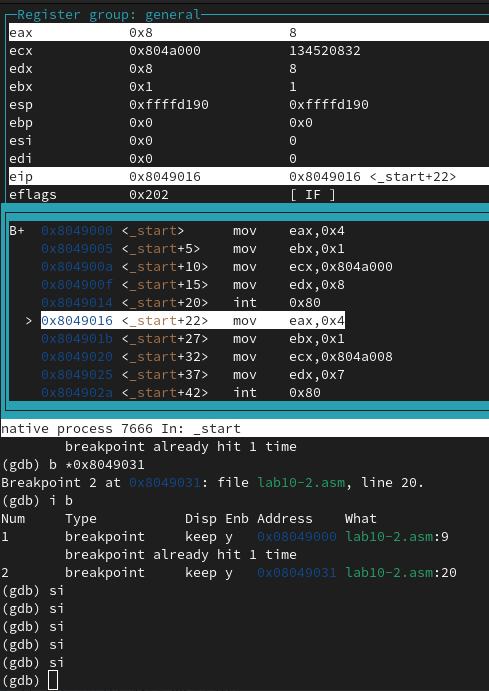


Рис. 17: Выполнение 5 инструкций с помощью команды stepi

Посмотреть содержимое регистров также можно с помощью команды info registers (или i r) (рис. 18).

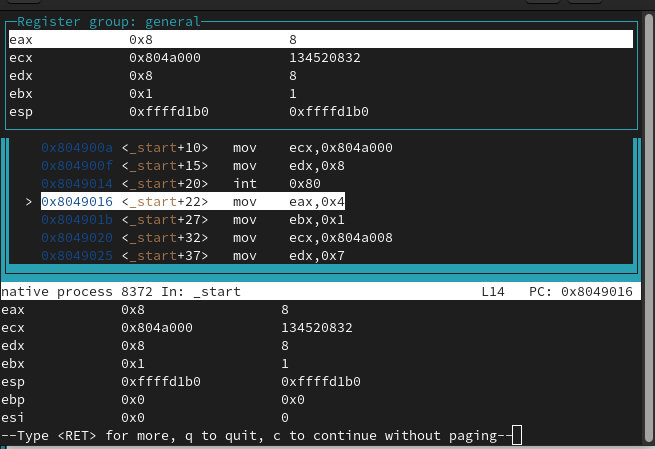


Рис. 18: Просмотр содержимого регистров с помощью команды info registers

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 переменной. Я посмотрела значение переменной msg1 по имени (рис. 19).

Рис. 19: Просмотр значения переменной msg1 по имени

Рис. 19: Просмотр значения переменной msg1 по имени

Я посмотрела значение переменной msg2 по адресу. Адрес переменной можно определить по дизассемблированной инструкции. Я посмотрела инструкцию mov ecx,msg2, которая записывает в регистр ecx адрес перемененной msg2 (рис. 20)

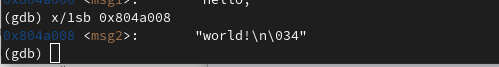


Рис. 20: Просмотр значения переменной msg2 по адресу инструкции mov ecx,msg2

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). Я изменила первый символ переменной msg1 (рис. 21)

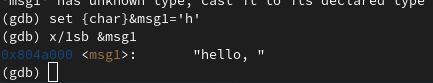


Рис. 21: Изменение первого символа переменной msg1

Я заменила первый символ во второй переменной msg2 (рис. 22)

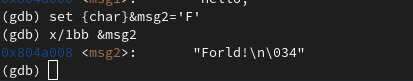


Рис. 22: Изменение символа в переменной msg2

Я вывела в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx (рис. 23).

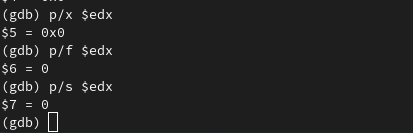


Рис. 23: Выведение значение регистра edx в различных форматах

С помощью команды set я изменила значение регистра ebx (рис. 24).

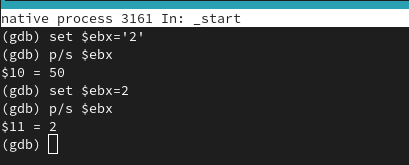


Рис. 24: Изменение значения регистра ebx

Разницу вывода команд p/s $ebx вызвана тем, что в первом случае значение регистра вводится как символ, а во втором - как цифра. Я завершила выполнение программы с помощью команды continue (сокращенно c) и вышла из GDB с помощью команды quit (сокращенно q)(рис. 25).

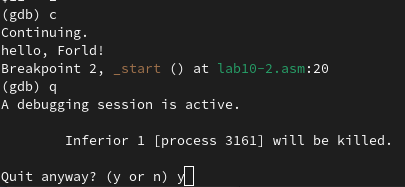


Рис. 25: Завершение выполнения программы и выход из GDB

### 4.2.3 Обработка аргументов командной строки в GDB

Я скопировала файл lab9-2.asm, созданный при выполнении лабораторной работы No9, с программой, выводящей на экран аргументы командной строки (Листинг 9.2), в файл с именем lab10-3.asm и создала исполняемый файл (рис. 26).

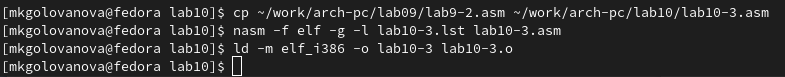


Рис. 26: Копирование файла lab9-2.asm в файл с именем lab10-3.asm и создание исполняемого файла

Для загрузки в gdb программы с аргументами необходимо использовать ключ –args. Загрузите исполняемый файл в отладчик, указав аргументы: gdb –args lab10-3 аргумент1 аргумент 2 ‘аргумент 3’ (рис. 27).

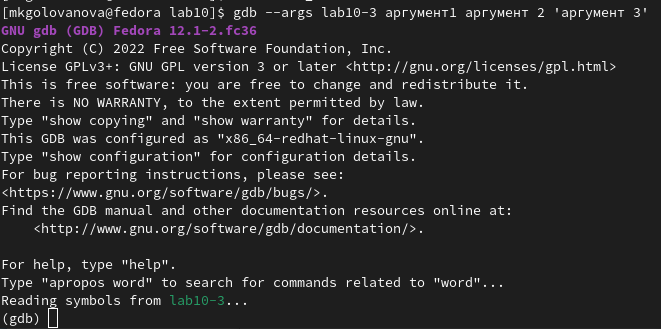


Рис. 27: Загрузка исполняемого файл в отладчик с указанием аргументов

Как отмечалось в предыдущей лабораторной работе, при запуске программы аргументы командной строки загружаются в стек. Я исследовала расположение аргументов командной строки в стеке после запуска программы с помощью gdb. Я установила точку останова перед первой инструкцией в программе и запустила её (рис. 28).

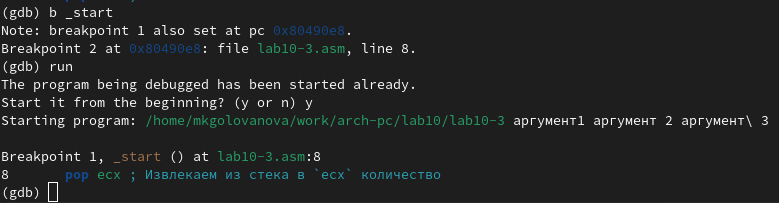


Рис. 28: Установка точки останова перед первой инструкцией и запуск программы

Адрес вершины стека храниться в регистре esp, и по этому адресу располагается число, равное количеству аргументов командной строки (включая имя программы). Число аргументов равно 5 – это имя программы lab10-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’. Я посмотрела 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. 29).

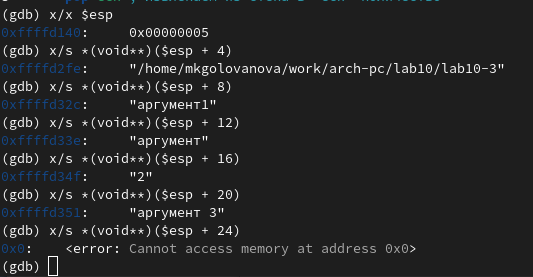


Рис. 29: Просмотр позиций стека

Шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12] и т.д.), потому что количество команд для выведения каждого аргумента равно 4.

# 5 Выполнение самостоятельной работы

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

В листинге 10.3 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.

# 6 Выводы

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