Лабораторная работа №9

Архитектура компьютера

Казначеева Кристина Никитична

Содержание

# 1 Цель работы

Лабораторная работа посвящена практическому освоению программирования с подпрограммами и отладке кода с помощью отладчика GDB.

# 2 Задание

В ходе лабораторной работы мы освоим программирование на NASM с использованием подпрограмм, отладку программ с помощью GDB (включая установку точек останова и работу с данными программы), а также обработку аргументов командной строки. В качестве примеров будут рассмотрены вывод сообщения “Hello, world!” и вычисление заданного выражения.

# 3 Выполнение лабораторной работы

Создадим новый каталог, перейдём в него и создадим файл lab09-1.asm:(рис. 1).

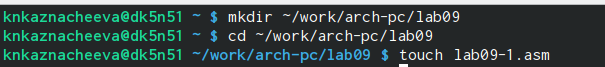


Рис. 1: Создание каталога и файла

Введём в файл lab9-1.asm текст программы с использованием вызова подпрограммы (рис. 2).

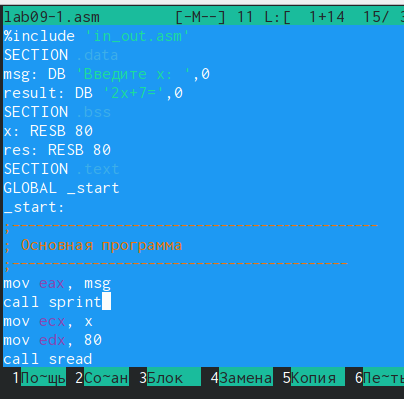


Рис. 2: Ввод текста программы

Создадим исполняемый файл и проверим его работу (рис. 3).

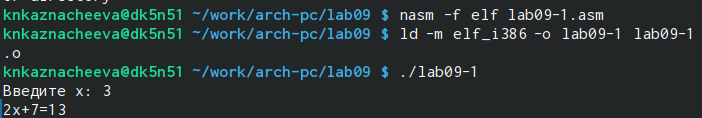


Рис. 3: Проверка работы исходного файла

Затем заменим 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 (рис. 4).

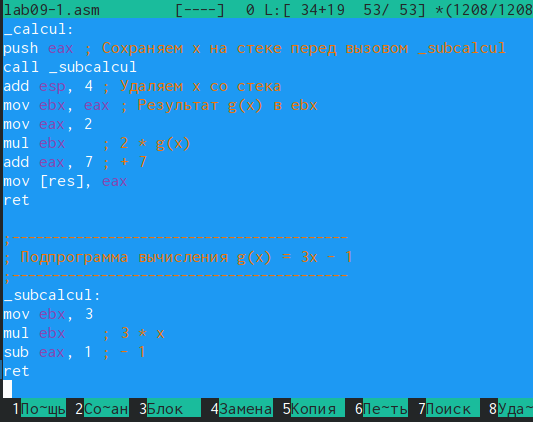


Рис. 4: Изменение текста программы

Запустим исполняемый файл и проверим его (рис. 5).

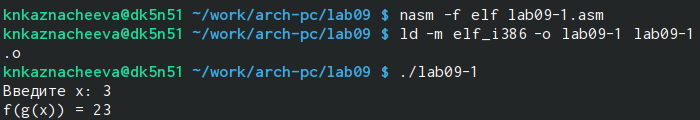


Рис. 5: Проверка работы исходного файла

Создадим файл lab09-2.asm ( (рис. 6).

Рис. 6: Создание файла

Рис. 6: Создание файла

Введём в файл lab9-2.asm текст программы вывода сообщения Hello world (рис. 7).

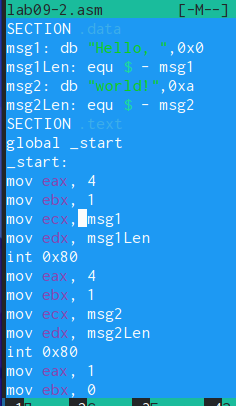


Рис. 7: Ввод текста программы

Для работы с GDB в исполняемый файл добавим отладочную информацию, для этого проведём трансляцию программ необходимо с ключом ‘-g’ (рис. 8):

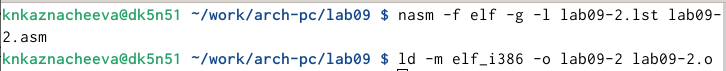


Рис. 8: Трансляция программы

Загрузим исполняемый файл в отладчик gdb (рис. 9).

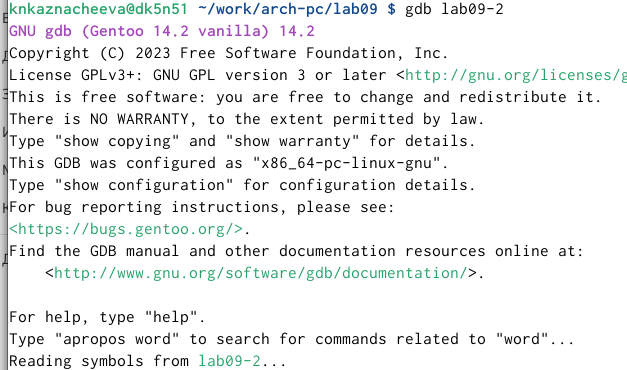


Рис. 9: Загрузка исполняемый файл в отладчик gdb

Проверим работу программы, запустив ее в оболочке GDB с помощью команды run (рис. 10).

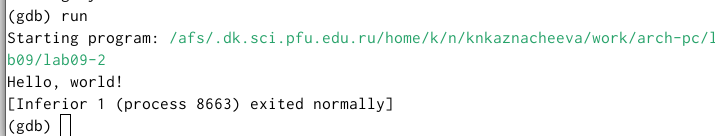


Рис. 10: Проверка работы программы с помощью команды run

Для более подробного анализа программы установим брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы (рис. 11).

Рис. 11: Проверка работы исходного файла

Рис. 11: Проверка работы исходного файла

Затем запустим программу (рис. 12).



Рис. 12: Запуск программы

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

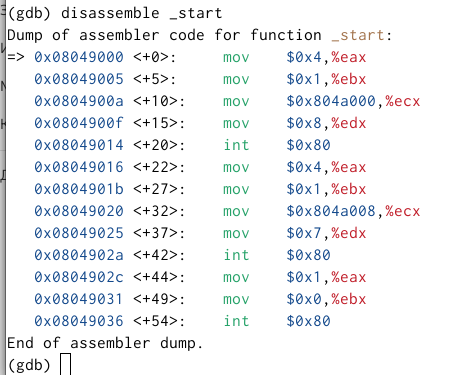


Рис. 13: Команда disassemble

Переключимся на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel (рис. 14).

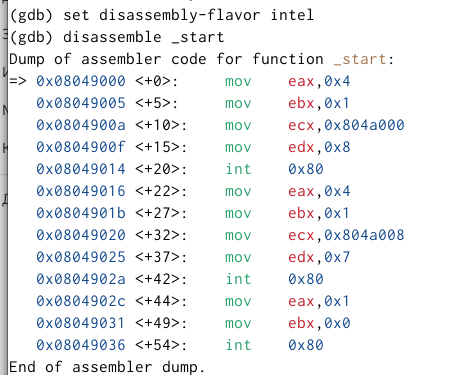


Рис. 14: Переключение на отображение команд с Intel’овским синтаксисом

Включим режим псевдографики для более удобного анализа программы (рис. 15).

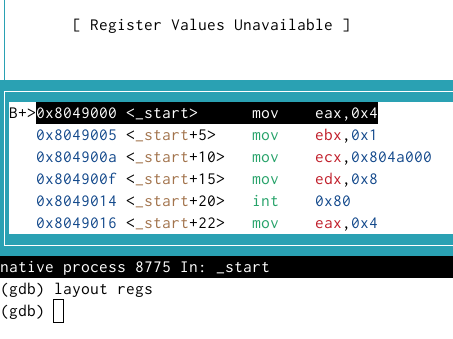


Рис. 15: Изменение текста программы

Основные различия в отображении синтаксиса машинных команд в режимах ATT (AT&T) и Intel заключаются в следующем: • Регистры: \* Intel: Имена регистров пишутся без префикса (например, eax, ebx). \* ATT: Имена регистров предваряются знаком % (например, %eax, %ebx). • Операторы размера: \* Intel: Размер операндов обычно указывается суффиксом (например, mov ax, bx для 16-битных регистров, mov eax, ebx для 32-битных). \* ATT: Размер операндов обычно указывается префиксом перед именем инструкции (например, movw %bx, %ax для 16-битных, movl %ebx, %eax для 32-битных). Также используются префиксы b (байт), w (слово), l (длинное слово - 32 бита), q (четверное слово - 64 бита). • Константы: \* Intel: Константы записываются непосредственно (например, mov eax, 10). \* ATT: Константы предваряются символом $ (например, movl $10, %eax). • Метки: \* Intel: Метки используются непосредственно (например, jmp mylabel). \* ATT: Метки предваряются знаком . (например, .mylabel).

На предыдущих шагах была установлена точка останова по имени метки (\_start). Проверим это с помощью команды info breakpoints (рис. 16).

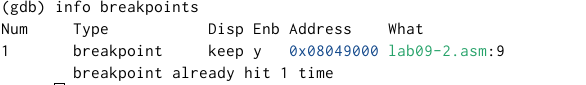


Рис. 16: Команда info breakpoints

Установим ещё одну точку останова по адресу инструкции (рис. 17).

Рис. 17: Установление точки останова по адресу инструкции

Рис. 17: Установление точки останова по адресу инструкции

Посмотрим информацию о всех установленных точках останова (рис. 18).

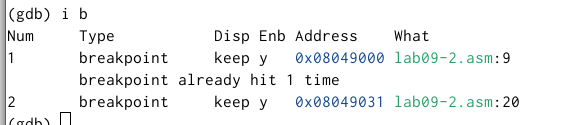


Рис. 18: Команда info breakpoints

Выполним 5 инструкций с помощью команды si и проследим за изменением значений регистров (рис. 19).

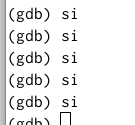


Рис. 19: Команда si

Значения регистров eax, ebx, и ecx изменяются (рис. 20).

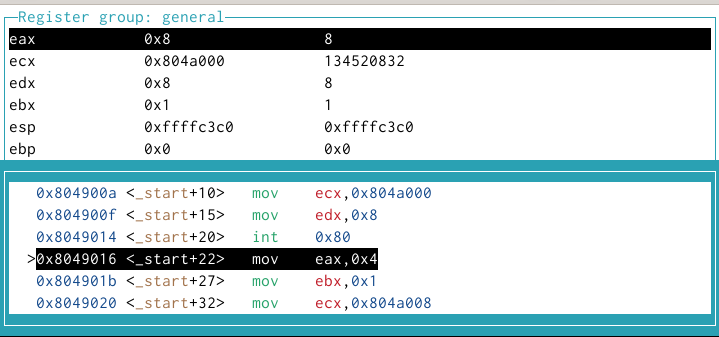


Рис. 20: Команда si

Посмотрим содержимое регистров также можно с помощью команды info registers (рис. 21).

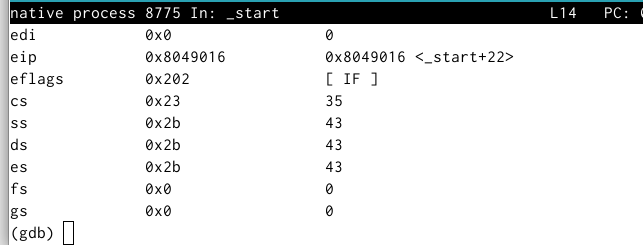


Рис. 21: Команда info registers

Посмотрим значение переменной msg1 по имени (рис. 22).

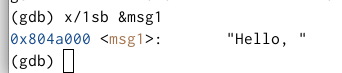


Рис. 22: Просмотр значения переменной msg1 по имени

Посмотрим значение переменной msg2 (рис. 23).

Рис. 23: Просмотр значения переменной msg2 по имени

Рис. 23: Просмотр значения переменной msg2 по имени

Изменим первоый символ переменной msg1 (рис. 24).

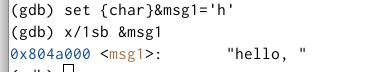


Рис. 24: Изменение символа переменной msg1

Заменим любой символ во второй переменной msg2 (рис. 25).

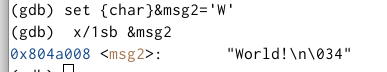


Рис. 25: Изменение символа переменной msg2

Чтобы посмотреть значения регистров воспользуемся командой print /F (рис. 26).

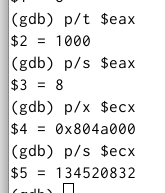


Рис. 26: Команда print

С помощью команды set изменим значение регистра ebx (рис. 27),(рис. 28).

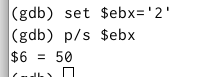


Рис. 27: Команда set

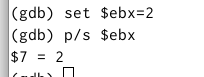


Рис. 28: Команда set

Разница в выводе команды p/s $ebx обусловлена тем, как GDB интерпретирует значение, присвоенное регистру $ebx. • set $ebx=‘2’: Здесь регистру $ebx присваивается строковое значение ‘2’. Символ ’ указывает на строку. • set $ebx=2: Здесь регистру $ebx присваивается числовое значение 2. Отсутствие кавычек указывает на число. Поэтому p/s $ebx отображает числовое значение 2. В сущности, в первом случае мы работаем со строкой, а во втором — с целым числом.

Завершим выполнение программы с помощью команды si и выйдем из GDB с помощью команды quit (рис. 29).

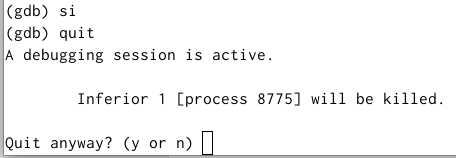


Рис. 29: Завершение выполнения программы и выход из GDB

Скопируем файл lab8-2.asm с программой выводящей на экран аргументы командной строки в файл с именем lab09-3.asm:(рис. 30).

Рис. 30: Копирование файла lab8-2.asm с именем lab09-3.asm

Рис. 30: Копирование файла lab8-2.asm с именем lab09-3.asm

Создадим исполняемый файл (рис. 31).

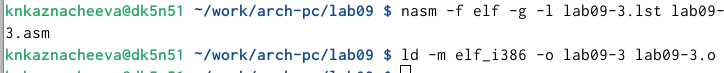


Рис. 31: Создание исполняемого файла

Для загрузки в gdb программы с аргументами необходимо использовать ключ –args. Загрузим исполняемый файл в отладчик, указав аргументы (рис. 32).

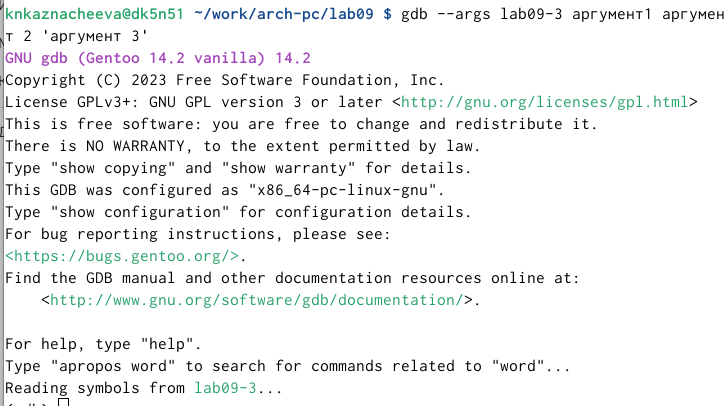


Рис. 32: Загрузка файла в отладчик

Установим точку останова перед первой инструкцией в программе и запустим ее (рис. 33).

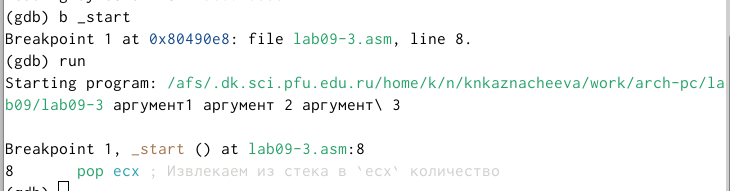


Рис. 33: Создание исполняемого файла

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 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’(рис. 34).

Рис. 34: Команда x/x $esp

Рис. 34: Команда x/x $esp

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. Смещение на 4 байта происходит потому, что в 64-битной архитектуре указатели (в данном случае, указатели на строки аргументов) занимают 8 байт, и стек выравнивается по 8 байтам (для повышения производительности). Это делает адреса аргументов выровненными по 8 байтам, а смещение между ними равно 8 байтам (рис. 35).

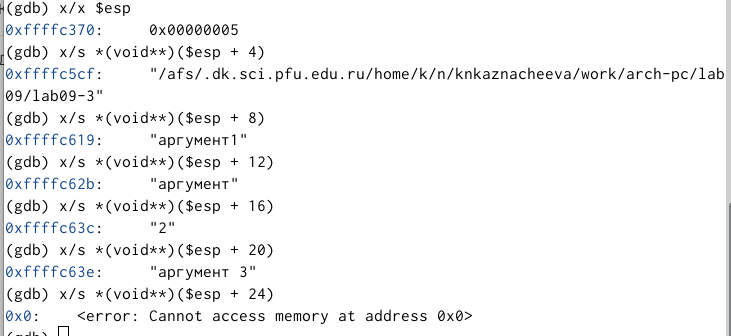


Рис. 35: Просмотр остальных позиций стека

Создадим файл lab09-4.asm (рис. 36).

Рис. 36: Создание файла

Рис. 36: Создание файла

Преобразуем программу из лабораторной работы №8 (Задание №1 для самостоятель- ной работы), реализовав вычисление значения функции f(x)=7(x+1) как подпрограмм (рис. 37).



Рис. 37: Ввод текста программы

Проверим программу исходного файла(рис. 38).

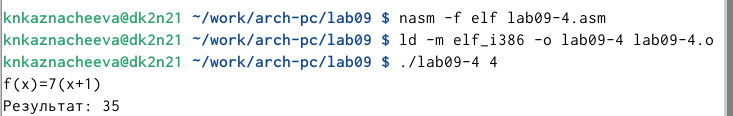


Рис. 38: Запуск программы

Создадим файл lab09-5.asm (рис. 39).

Рис. 39: Создание файла

Рис. 39: Создание файла

Программа вычисления выражения (3 + 2) ∗ 4 + 5 при запуске дает неверный результат. Проверим это. С помощью отладчика GDB, анализируя изменения значений регистров, определим ошибку и исправим ее (рис. 40).

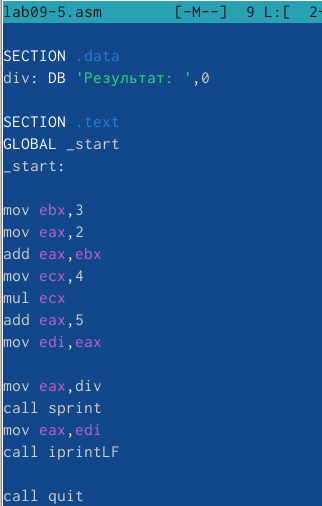


Рис. 40: Ввод текста программы

Проверим программу исходного файла (рис. 41).

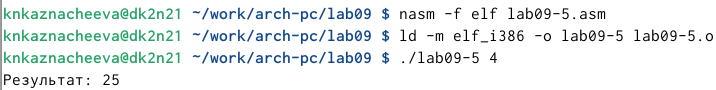


Рис. 41: Запуск программы

# 4 Вывод

В рамках лабораторной работы были изучены и отработаны навыки программирования на языке ассемблера NASM, включающие создание и использование подпрограмм, отладку программ с помощью отладчика GDB (с применением точек останова и анализа содержимого памяти), а также обработку аргументов командной строки. В качестве иллюстративных примеров были использованы вывод сообщения “Hello, world!” и вычисление математического выражения.