Отчёт по лабораторной работе №9

Дисциплина: Архитектура компьютера

Буриева Шахзода Акмаловна

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

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

2 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно 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.

Отладчики позволяют управлять ходом выполнения программы, контролировать и из- менять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправление. Наиболее популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам. Пошаговое выполнение — это выполнение программы с остановкой после каждой строчки, чтобы программист мог проверить значения переменных и выполнить другие действия. Точки останова — это специально отмеченные места в программе, в которых программа-отладчик приостанавливает выполнение программы и ждёт команд.

GDB (GNU Debugger — отладчик проекта GNU) работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования. GDB предлагает обширные средства для слежения и контроля за выполнением компьютерных программ. Отладчик не содержит собственного

графического пользовательского интерфейса и использует стандартный текстовый интерфейс консоли. Однако для GDB существует несколько сторонних графических надстроек, а кроме того, некоторые интегрированные среды разработки используют его в качестве базовой подсистемы отладки. Отладчик GDB (как и любой другой отладчик) позволяет увидеть, что происходит «внутри» программы в момент её выполнения или что делает программа в момент сбоя.

сли есть файл с исходным текстом программы, а в исполняемый файл включена информация о номерах строк исходного кода, то программу можно отлаживать, работая в отладчике непосредственно с её исходным текстом. Чтобы программу можно было отлаживать на уровне строк исходного кода, она должна быть откомпилирована с ключом -g.Существует два режима отображения синтаксиса машинных команд: режим Intel, используемый в том числе в NASM, и режим ATT (значительно отличающийся внешне). По умолчанию в дизассемблере GDB принят режим ATT. Переключиться на отображение команд с привычным Intel'овским синтаксисом можно, введя команду set disassembly-flavor intel.

Для продолжения остановленной программы используется команда continue (c) (gdb) c '[аргумент]'. Выполнение программы будет происходить до следующей точки останова. В качестве аргумента может использоваться целое число N.

Подпрограмма — это, как правило, функционально законченный участок кода, который можно многократно вызывать из разных мест программы. В отличие от простых переходов из подпрограмм существует возврат на команду, следующую за вызовом. Если в программе встречается одинаковый участок кода, его можно оформить в виде подпрограммы, а во всех нужных местах поставить её вызов. При этом подпрограмма будет содержаться в коде в одном экземпляре, что позволит уменьшить размер кода всей программы.

Для вызова подпрограммы из основной программы используется инструкция call, которая заносит адрес следующей инструкции в стек и загружает в регистр eip адрес соответствующей подпрограммы, осуществляя таким образом переход. Затем начинается выполнение подпрограммы, которая, в свою очередь, также

может содержать подпрограммы. Подпрограмма завершается инструкцией ret, которая извлекает из стека адрес, занесённый туда соответствующей инструкцией call, и заносит его в eip. После этого выполнение основной программы возобновится с инструкции, следующей за инструкцией call.

3 Выполнение лабораторной работы

Создала каталог для программам лабораторной работы №9, перешла в него и создала файл lab09-1.asm.

Рис. 3.1: Создание каталога lab9,переход в него и создание файла lab09-1.asm

Внимательно изучила текст программы для вычисления арифметического выражения f(x) = 2x + 7 с помощью подпрограммы _calcul.Затем ввела в файл lab09-1.asm текст программы из листинга и ввела его в файл lab09-1.asm

Рис. 3.2: Ввод текста программы в файл lab09-1.asm

Создала исполняемый файл и запустила его.На запрос “Введите x” ввела число 6.На экран вывелось уравнение, результатом которого является число 19 при подстановке вместо x числа 6.Само выражение вычисляется в подпрограмме.

Рис. 3.3: Создание исполняемого файла и его запуск

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.

Рис. 3.4: Изменение текста программы lab09-1.asm

Создала исполняемый файл и запустила его.На запрос “Введите x” ввела число 6. На экран вывелось значние 18.Получается спрева 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)).

Рис. 3.5: Создание исполняемого файла и его запуск

Создала файл lab08-2.asm для выполнения дальнейшей работы.

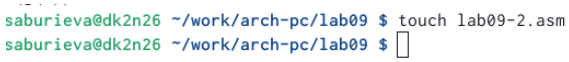

Рис. 3.6: Создание файла lab09-2.asm

Внимательно изучила текст программы из листинга и ввела его в файл lab09-2.asm.

Рис. 3.7: Ввод текста программы в файл lab09-2.asm

Создала исполняемый файл и запустила его.На экран вывелось сообщение “Hello, world!”.

Рис. 3.8: Создание исполняемого файла и его запуск

Для работы с GDB в исполняемый файл добавила отладочную информацию, для этого трансляцию программ проводила с ключом ‘-g’.Загрузила исполняемый файл в отладчик gdb.

```
saburieva@dk2n26 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
saburieva@dk2n26 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
saburieva@dk2n26 ~/work/arch-pc/lab09 $ gdb lab09-2
GNU gdb (Gentoo 13.2 vanilla) 13.2
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...
```

Рис. 3.9: Создание исполняемого файла для работы с GDB и его загрузка

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

```
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/s/a/saburieva/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 10057) exited normally]
```

Рис. 3.10: Проверка работы программы с помощью команды r

Для более подробного анализа программы установила брейкпоинт на метку _start, с которой начинается выполнение любой ассемблерной программы, и запустила её.

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/s/a/saburieva/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9
9       mov eax, 4
(gdb)
```

Рис. 3.11: Установка брейклоинта для подробного анализа программы

Посмотрела дисассимилированный код программы с помощью команды disassemble начиная с метки _start.

Рис. 3.12: Просмотр дисассимилированного кода программы

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

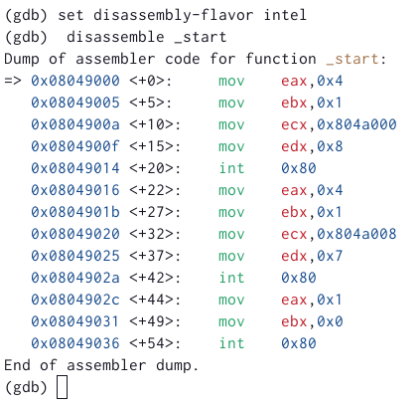

Рис. 3.13: Переключение на отображение команд с Intel'овским синтаксисом

Различие отображения синтаксиса машинных команд в режимах ATT и Intel в том, что в представлении ATT в виде шестнадцатиричного числа записаны первые аргументы всех команд, а в представлении Intel записаны адреса вторых аргументов.

Включила режим псевдографики для более удобного анализа программы.

Рис. 3.14: Включение режима псевдографики с командой layout asm

Рис. 3.15: Включение режима псевдографики с командой layout regs

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

Рис. 3.16: Команда info breakpoints

Установила еще одну точку останова по адресу инструкции. Адрес инструкции посмотрела в средней части экрана в левом столбце.Определила адрес предпоследней инструкции (mov ebx,0x0) и установила точку останова.

Рис. 3.17: Установка еще одной точки останова по адресу инструкции

Посмотрела информацию о всех установленных точках останова.

Рис. 3.18: Просмотр информации с помощью i b

Посмотрела содержимого регистров с помощью команды info registers.

Рис. 3.19: Просмотр содержимого регистров

Посмотрела значение переменной msg1 по имени.

Рис. 3.20: Просмотр значение переменной msg1 по имени

Посмотрела значение переменной msg2 по адресу. Адрес переменной определила по дизассемблированной инструкции. Посмотрела инструкцию mov ecx,msg2 которая записывает в регистр ecx адрес перемененной msg2.

Рис. 3.21: Определение адреса переменной msg2 по дизассемблированной инструкции

```
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:       "world!\n"<error: Cannot access memory at address 0x804a00f>
(gdb)
```

Рис. 3.22: Просмотр значения пременной msg2 по адресу

Изменила первый символ переменной msg1.

```
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:       "hello, "
(gdb)
```

Рис. 3.23: Изменение символа у msg1

Изменила любой символ второй переменной msg2.

```
(gdb) set {char}&msg2='W'
(gdb) x/1sb &msg2
0x804a008 <msg2>:       "World!\n\034"
(gdb)
```

Рис. 3.24: Изменение символа у msg2

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

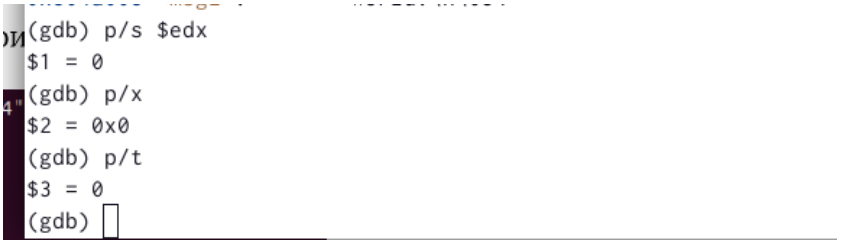

Рис. 3.25: Выведение различных значений регистра edx

С помощью команды set изменила значение регистра ebx.

Рис. 3.26: Изменение значения регистра ebx

Завершила выполнение программы с помощью команды continue и вышла из GDB с помощью команды quit.

Скопировала 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.

Рис. 3.27: Копирование файла

Создала исполняемый файл.

Рис. 3.28: Создание исполняемого файла

Для загрузки в gdb программы с аргументами использовала ключ –args. Загрузила исполняемый файл в отладчик, указав аргументы.

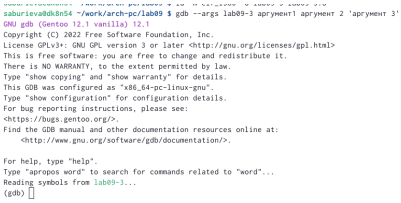

Рис. 3.29: Загрузка исполняемого файла в откладчик

Дальше исследовала расположение аргументов командной строки в стеке после запуска программы с помощью gdb.Для начала установила точку останова перед первой инструкцией в программе и запустила ее.

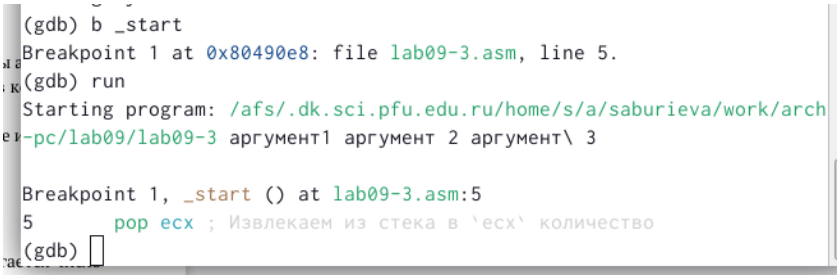

Рис. 3.30: Установка точки останова и ее запуск

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

Рис. 3.31: Просмотр адреса в регистре esp

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.

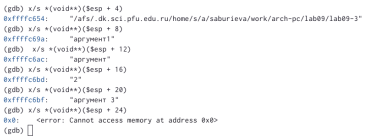

Рис. 3.32: Просмотр адреса остальных позиций стека

Их дареса располгаются в 4 байтах друг от друга,потому что именно столько занимает элемент стека.

4 Выполнение заданий для самостоятельной работы

Создала файл lab09-4.asm для преобразования программы из лабораторной работы №8 (Задание №1 для самостоятель- ной работы).

Рис. 4.1: Создание файла lab09-4.asm

Вводя программу, изменила ее так,чтобы она вычислялась как подпрограмма.

Рис. 4.2: Ввод текста программы как подпрограмма

Создала исполняемый файл и запустила его.

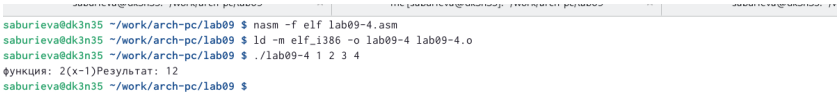

Рис. 4.3: Создание исполняемого файла и его запуск

Создала файл lab09-5.asm для проверки программы вычисления выражения (3 + 2) * 4 + 5.

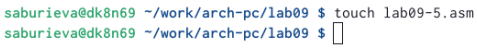

Рис. 4.4: Создание файла lab09-5.asm

Ввела текст программы из листинга для вычисления выражения (3 + 2) * 4 + 5.

Рис. 4.5: Ввод текста программы для проверки

Для работы с GDB в исполняемый файл добавила отладочную информацию, для этого трансляцию программ проводила с ключом ‘-g’.Загрузила исполняемый файл в отладчик gdb.

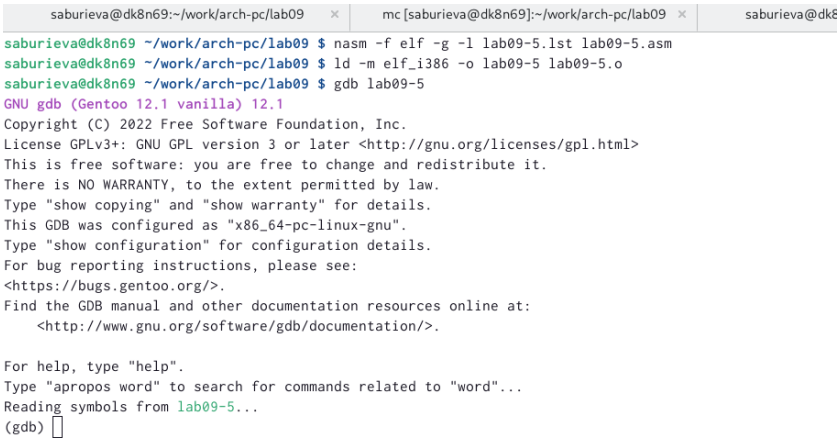

Рис. 4.6: Создание исполняемого файла для работы с GDB и его загрузка

При запуске данная программа дает неверный результат. Проверила это с помощью отладчика GDB, анализируя изменения значений регистров.

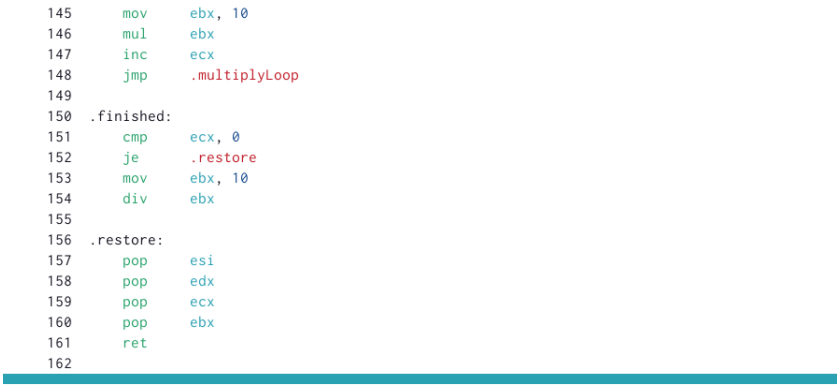

Рис. 4.7: Проверка программы с помощью GDB

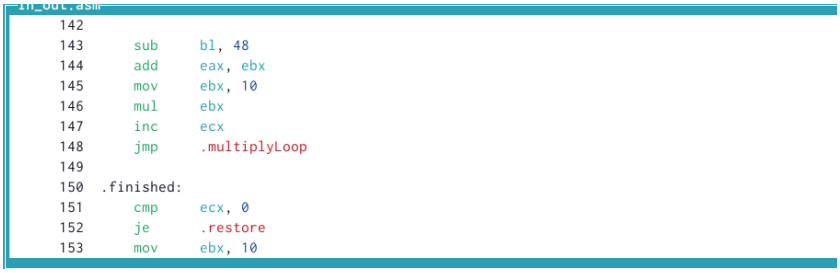

Рис. 4.8: Проверка программы с помощью GDB

Проверив и проанализировав изменения значений регистров, нашла ошибку в следующих строчках.

```
add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx
```

Рис. 4.9: Нахождение ошибки в строках

Далее исправила текст программы правильным образом, чтобы получился верный ответ.

Рис. 4.10: Исправление строк в программе

Создала исполняемый файл и запустила его. Для работы с GDB в исполняемый файл добавила отладочную информацию, для этого трансляцию программ проводила с ключом ‘-g’.Загрузила исполняемый файл в отладчик gdb. В итоге

проверила работу программы, запустив ее в оболочке GDB с помощью команды run.Изменённая программа корректна.

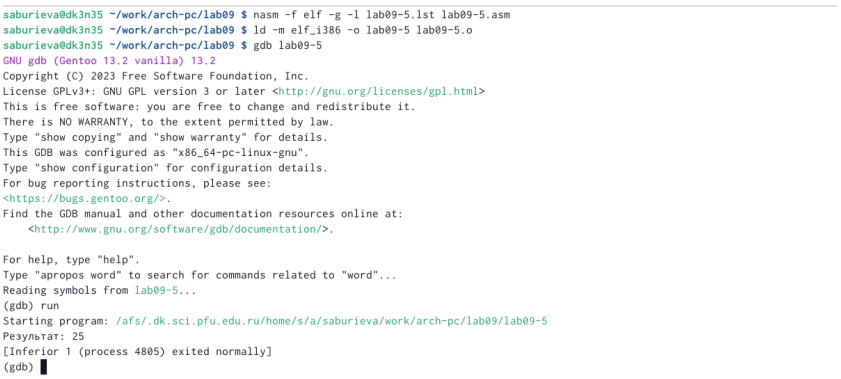

Рис. 4.11: Создание исполняемого файла,загрузка в откладчик gdb и ее запуск

5 Выводы

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

Список литературы