Отчет по лабораторной работе №9

Дисциплина: Архитектура компьютера

Краснова Камилла Геннадьевна

Содержание

# 1 Цель работы

Цель данной лабораторной работы - приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

# 2 Задание

1. Реализация подпрограмм в NASM
2. Отладка программ с помощью GDB
3. Самостоятельное выполнение заданий по материалам лабораторной работы

# 3 Теоретическое введение

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.

Наиболее часто применяют следующие методы отладки: • создание точек контроля значений на входе и выходе участка программы (например, вывод промежуточных значений на экран — так называемые диагностические сообщения);

• использование специальных программ-отладчиков. Отладчики позволяют управлять ходом выполнения программы, контролировать и изменять 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. Наиболее популярные виды точек останова: • Breakpoint — точка останова (остановка происходит, когда выполнение доходит до определённой строки, адреса или процедуры, отмеченной программистом);

• Watchpoint — точка просмотра (выполнение программы приостанавливается, если программа обратилась к определённой переменной: либо считала её значение, либо изменила его).

Точки останова устанавливаются в отладчике на время сеанса работы с кодом програм- мы, т.е. они сохраняются до выхода из программы-отладчика или до смены отлаживаемой программы.

# 4 Выполнение лабораторной работы

## 4.1 Реализация подпрограмм в NASM

С помощью утилиты mkdir создаю директорию, в которой буду создавать файлы с программами для лабораторной работы №9. Перехожу в созданный каталог с помощью утилиты cd и с помощью утилиты touch создаю файл lab09-1.asm.(рис. 1).

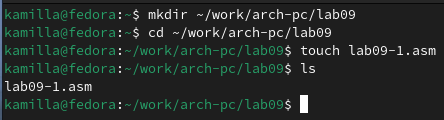


Рис. 1: Создание директории

Открываю созданный файл lab09-1.asm, вставляю в него программу (рис. 2).

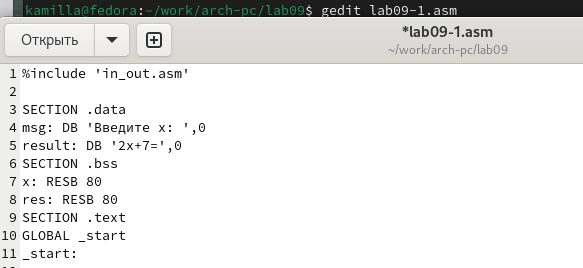


Рис. 2: Редактирование файла

Создаю исполняемый файл программы и запускаю его (рис. 3).

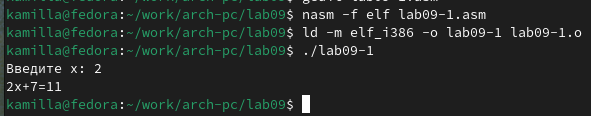


Рис. 3: Запуск исполняемого файла

Изменяю текст программы , добавляя подпрограмму \_subcalcul в подпрограмму \_calcul(рис. 4).

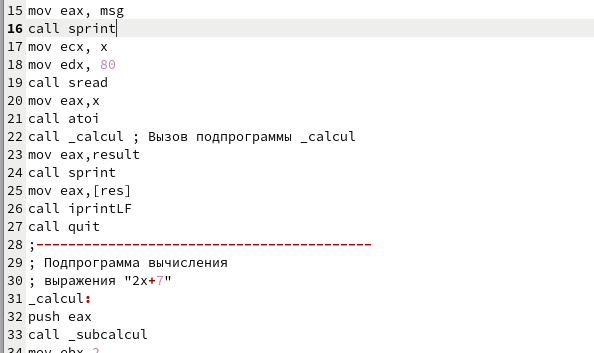


Рис. 4: Редактирование файла

Создаю новый исполняемый файл программы и запускаю его (рис. 5).

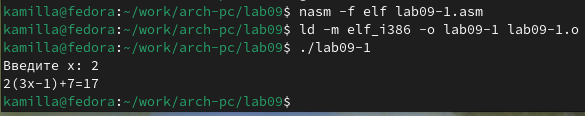


Рис. 5: Запуск исполняемого файла

## 4.2 Отладка программам с помощью GDB

Создаю файл lab09-2.asm (рис. 6).

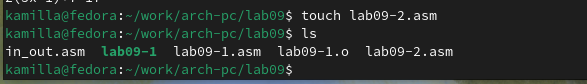


Рис. 6: Создание файла

Открываю созданный файл lab09-2.asm, вставляю в него программу (рис. 7).

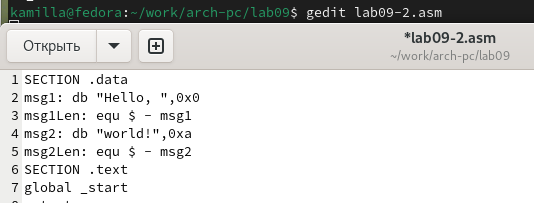


Рис. 7: Редактирование файла

Получаю исполняемый файл. Трансляцию программ провожу с ключом ‘-g’ (рис. 8).

Рис. 8: Получение исполняемого файла

Рис. 8: Получение исполняемого файла

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

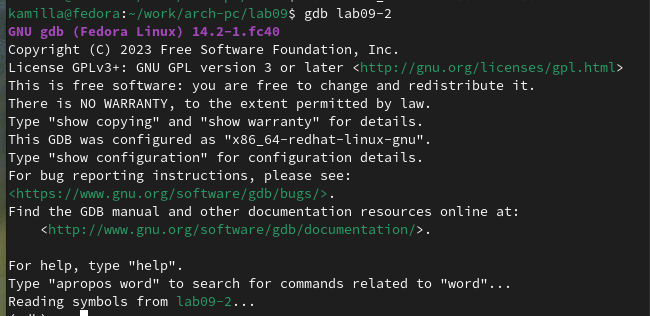


Рис. 9: Загрузка файла

Проверяю работу программы, запустив ее в оболочке GDB (рис. 10).

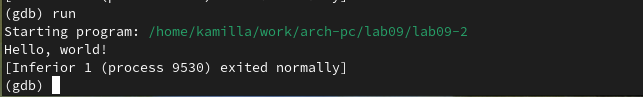


Рис. 10: Проверка работы программы

Устанавливаю брекпоинт на метку \_start и запускаю программу (рис. 11).

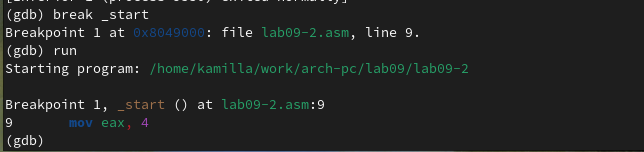


Рис. 11: Запуск отладчика с брекпоинтом

Далее смотрю дисассимилированный код программы (рис. 12).

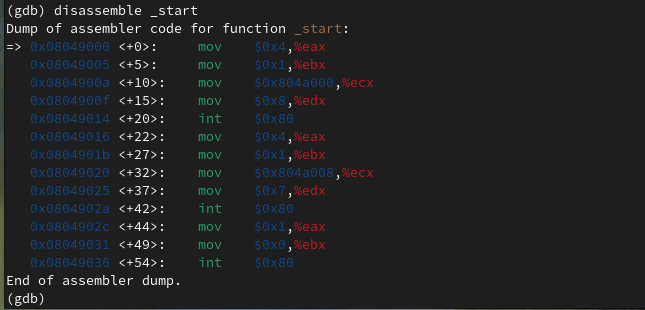


Рис. 12: Дисассимилирование программы

Переключаюсь на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel (рис. 13).



Рис. 13: Переключение отображения команд

Включаю режим псевдографики (рис. 14) и (рис. 15).

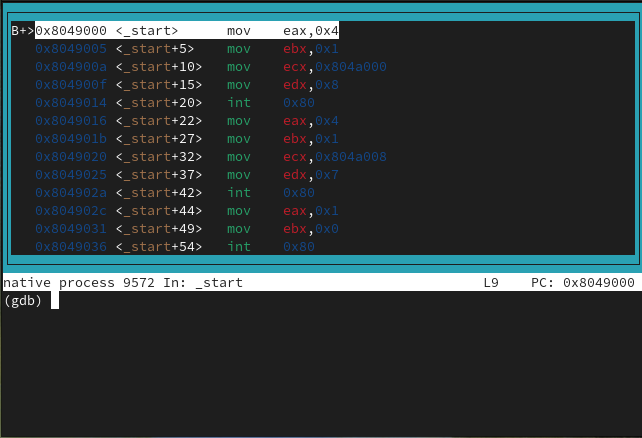


Рис. 14: Режим псевдографики

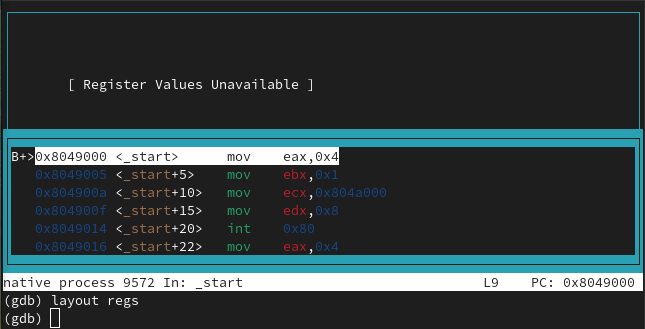


Рис. 15: Режим псевдографики

Различия между синтаксисом ATT и Intel заключаются в порядке операндов (ATT - Операнд источника указан первым. Intel - Операнд назначения указан первым), их размере (ATT - pазмер операндов указывается явно с помощью суффиксов, непосредственные операнды предваряются символом $; Intel - Размер операндов неявно определяется контекстом, как ax, eax, непосредственные операнды пишутся напрямую), именах регистров(ATT - имена регистров предваряются символом %, Intel - имена регистров пишутся без префиксов).

## 4.3 Добавление точек останова

Проверяю установку точки останова по имени метки (рис. 16).

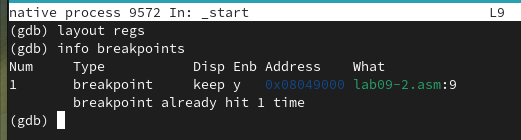


Рис. 16: Проверка установки точки

Устанавливаю точку останова по адресу инструкции (рис. 17).

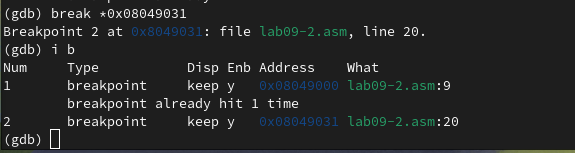


Рис. 17: Добавление точки останова

## 4.4 Работа с данными программы в GDB

Просматриваю содержимое регистров командой info registers (рис. 18).

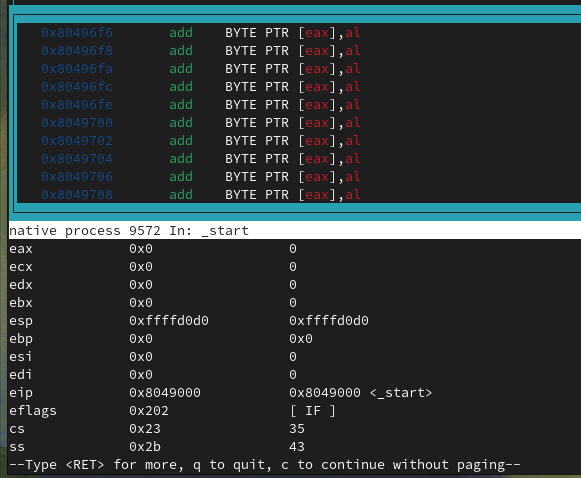


Рис. 18: Просмотр содержимого

Смотрю содержимое переменных по имени и по адресу (рис. 19).

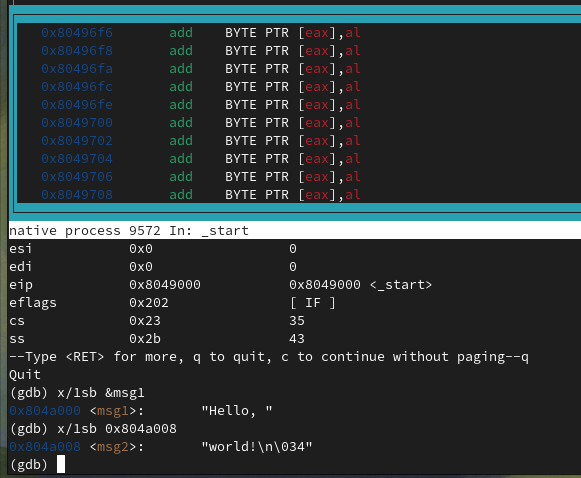
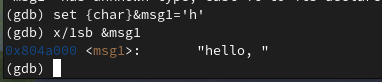
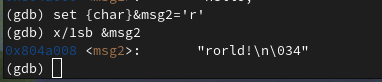


Рис. 19: Просмотр переменных

Меняю содержимое переменных по имени и по адресу (рис. **¿fig:020?**) и (рис. **¿fig:021?**).

Вывожу в различных форматах значение регистра edx (рис. 20).



Рис. 20: Просмотр значения регистра разными представлениями

С помощью команды set меняю содержимое регистра ebx (рис. 21).

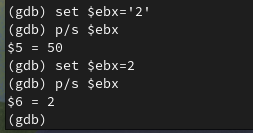


Рис. 21: Примеры использования команды set

## 4.5 Обработка аргументов командной строки в GDB

Копирую программу из предыдущей лабораторной работы в текущий каталог и и создаю исполняемый файл с файлом листинга и отладки (рис. 22).

Рис. 22: Подготовка новой программы

Рис. 22: Подготовка новой программы

Запускаю программу с режиме отладки с указанием аргументов (рис. 23).

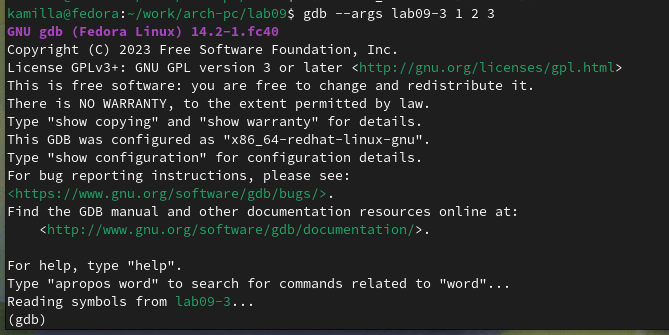


Рис. 23: Запуск в режиме отладки

Указываю брейкпопнт и запускаю отладку. (рис. **¿fig:06?**).

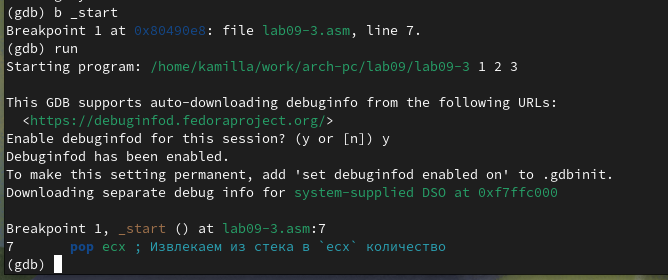


Рис. 24: Запуск отладки

Проверяю работу стека, изменяя аргумент команды просмотра регистра esp на +4, число обусловлено разрядностью системы, а указатель void занимает как раз 4 байта. Ошибка при аргументе +20 означает, что аргументы на вход программы закончились. (рис. 25).

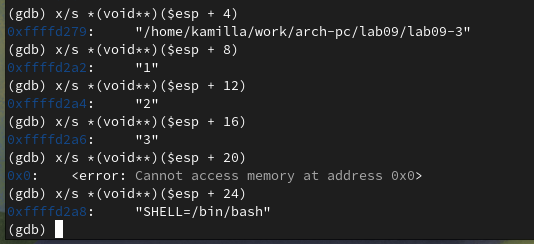


Рис. 25: Проверка работы стека

## 4.6 Задание для самостоятельной работы

1. Меняю программу самостоятельной части предыдущей лабораторной работы с использованием подпрограммы (рис. 26).

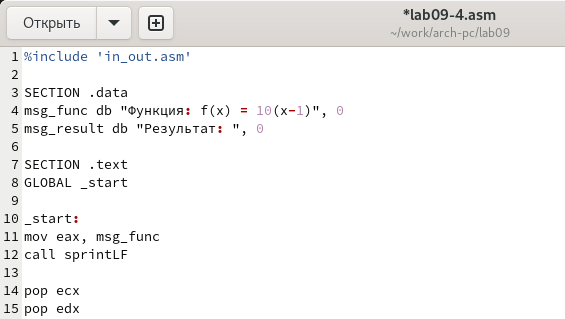


Рис. 26: Измененная программа предыдущей лабораторной работы

Код программы:

%include 'in\_out.asm'  
  
SECTION .data  
msg\_func db "Функция: f(x) = 10x - 4", 0  
msg\_result db "Результат: ", 0  
  
SECTION .text  
GLOBAL \_start  
  
\_start:  
mov eax, msg\_func  
call sprintLF  
  
pop ecx  
pop edx  
sub ecx, 1  
mov esi, 0  
  
next:  
cmp ecx, 0h  
jz \_end  
pop eax  
call atoi  
  
call \_calculate\_fx  
  
add esi, eax  
loop next  
  
\_end:   
mov eax, msg\_result  
call sprint  
mov eax, esi  
call iprintLF  
call quit  
  
\_calculate\_fx:  
mov ebx, 10  
mul ebx  
sub eax, 4

1. Запускаю программу в режике отладичка и просматриваю изменение значений регистров через i r. При выполнении инструкции mul ecx можно заметить, что результат умножения записывается в регистр eax, но также меняет и edx. Значение регистра ebx не обновляется напрямую, поэтому программа неверно подсчитывает функцию (рис. 27).

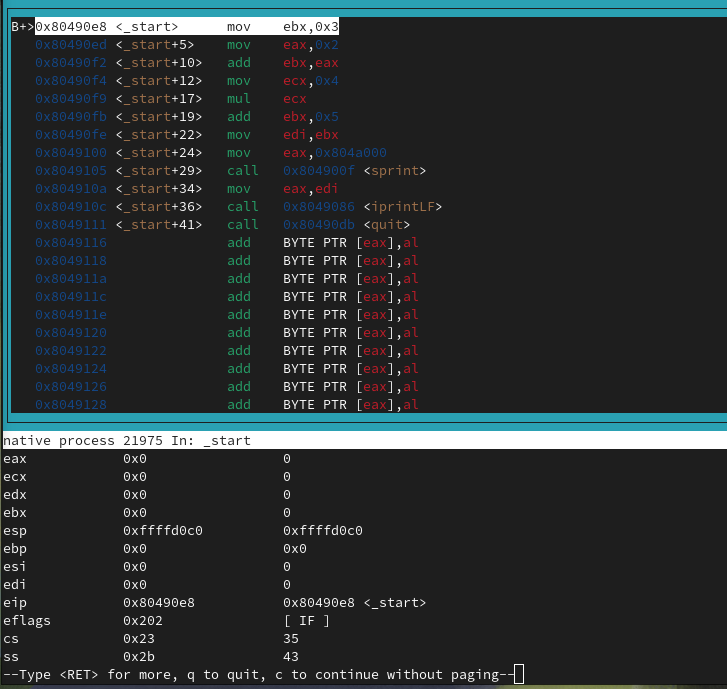


Рис. 27: Поиск ошибки в программе через пошаговую отладку

Исправляю найденную ошибку, теперь программа верно считает значение функции (рис. 28).

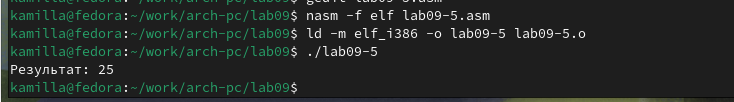


Рис. 28: Проверка корректировок в программе

Код измененной программы:

%include 'in\_out.asm'  
  
SECTION .data  
div: DB 'Результат: ', 0  
  
SECTION .text  
GLOBAL \_start  
\_start:  
  
mov ebx, 3  
mov eax, 2  
add ebx, eax  
mov eax, ebx  
mov ecx, 4  
mul ecx  
add eax, 5  
mov edi, eax  
  
mov eax, div  
call sprint  
mov eax, edi  
call iprintLF  
  
call quit

# 5 Выводы

При выполнении данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм, а так же познакомилась с методами отладки при поомщи GDB и его основными возможностями.

# 6 Список литературы

1. [Лабораторная работа №9](https://esystem.rudn.ru/pluginfile.php/2089096/mod_resource/content/0/Лабораторная%20работа%20№9.%20Понятие%20подпрограммы.%20Отладчик%20.pdf)