Лабораторная работа №9

Архитектура компьютера

Тойчубекова Асель Нурлановна

Содержание

# 1 Цель работы

Целью лабораторной работы №9 является приобретение навыков написания программ с использованием подпрограмм, также знакомство с методами отладки при помощи GDB и его основными возможностями. # Задание 1. Изучить теоретическое введение:  
- Понятие откладки  
- Методы откладуи  
- Основные возможности откладчика GDB  
- Запуск откладчика GDB;  
- Дизассемблирование программы;  
- Точки останова;  
- Пошаговая откладка;  
- Работа с данными программы в GDB;  
- Понятие подпрограммы;  
- Инструкция call и инструкция ret;  
2. Написать программу с использованием вызова подпрограммы  
3. Выполнить откладку программы с помощью GDB  
4. Выполнить обработку аргументов командной строки в GDB  
5. Задание для самостоятельной работы:  
- Преобразуйте программу из лабораторной работы No8 (Задание No1 для самостоятель- ной работы), реализовав вычисление значения функции f(x) как подпрограмму.  
- В листинге 9.3 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.

# 2 Теоретическое введение

## 2.1 Понятие об отладке

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).

## 2.2 Методы откладки

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 сообще- ния);  
• использование специальных программ-отладчиков.

## 2.3 Основные возможности отладчика GDB

GDB (GNU Debugger — отладчик проекта GNU) [1] 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. От- ладчик не содержит собственного графического пользовательского интерфейса и использует стандартный текстовый интерфейс консоли. Однако для GDB существует несколько сторон- них графических надстроек, а кроме того, некоторые интегрированные среды разработки используют его в качестве базовой подсистемы отладки. GDB может выполнять следующие действия:  
• начать выполнение программы, задав всё, что может повлиять на её поведение;  
• остановить программу при указанных условиях;  
• исследовать, что случилось, когда программа остановилась;  
• изменить программу так, чтобы можно было поэкспериментировать с устранением эффектов одной ошибки и продолжить выявление других.

## 2.4 Запуск отладчика GDB; выполнение программы; выход

Синтаксис команды для запуска отладчика имеет следующий вид:  
gdb опции имя файла|ID процесса  
После запуска gdb выводит текстовое сообщение — так называемое «nice GDB logo». В следующей строке появляется приглашение (gdb) для ввода команд.

Далее приведён список некоторых команд GDB.

Команда run (сокращённо r) — запускает отлаживаемую программу в оболочке GDB.

Если точки останова были заданы, то отладчик останавливается на соответствующей команде и выдаёт номер точки останова, адрес и дополнительную информацию — текущую строку, имя процедуры, и др.

Команда kill (сокращённо k) прекращает отладку программы, после чего следует вопрос о прекращении процесса отладки:  
Kill the program being debugged? (y or n) y

Если в ответ введено y (то есть «да»), отладка программы прекращается. Командой run её можно начать заново, при этом все точки останова (breakpoints), точки просмотра (watchpoints) и точки отлова (catchpoints) сохраняются. Для выхода из отладчика используется команда quit (или сокращённо q)

## 2.5 Дизассемблирование программы

Если 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. Посмотреть дизассемблированный код программы можно с помощью команды disassemble :  
(gdb) disassemble \_start.  
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.

## 2.6 Точки останова

Установить точку останова можно командой break (кратко b). Типичный аргумент этой команды — место установки. Его можно задать как имя метки или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка».

Информацию о всех установленных точках останова можно вывести командой info breakpoint.

Для того чтобы сделать неактивной какую-нибудь ненужную точку останова, можно воспользоваться командой disable, а активировать командой enable.

Если же точка останова в дальнейшем больше не нужна, она может быть удалена с помощью команды delete.

## 2.7 Пошаговая отладка

Для продолжения остановленной программы используется команда continue (c) (gdb) с аргумент. Выполнение программы будет происходить до следующей точки останова. В качестве аргумента может использоваться целое число N, которое указывает отладчику проигнорировать N− 1 точку останова (выполнение остановится на N-й точке). Команда stepi (кратко sI) позволяет выполнять программу по шагам, т.е. данная команда выполняет ровно одну инструкцию.

## 2.8 Работа с данными программы в GDB

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Посмотреть содержимое регистров можно с помощью команды info registers. Для отображения содержимого памяти можно использовать команду x/NFU , выдаёт содержимое ячейки памяти по указанному адресу. NFU задает формат, в котором выводятся данные.

## 2.9 Понятие подпрограммы

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.

## 2.10 Инструкция call и инструкция ret

Для вызова подпрограммы из основной программы используется инструкция call, кото- рая заносит адрес следующей инструкции в стек и загружает в регистр eip адрес соответствующей подпрограммы, осуществляя таким образом переход. Подпрограмма завершается инструкцией ret, которая извлекает из стека адрес, занесённый туда соответствующей инструкцией call, и заносит его в eip. Подпрограмма может вызываться как из внешнего файла, так и быть частью основной программы.

# 3 Выполнение лабораторной работы

Создадим каталог для выполнения лабораторной работы №9, перейдем в него и создадим файл lab09-1.asm.(РИС.1)

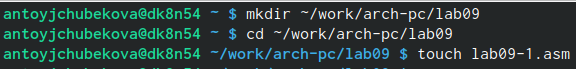


РИС.1 Создание каталога

Введем в файл lab09-1.asm текст программы с использованием вызова подпрограммы.(РИС.2)

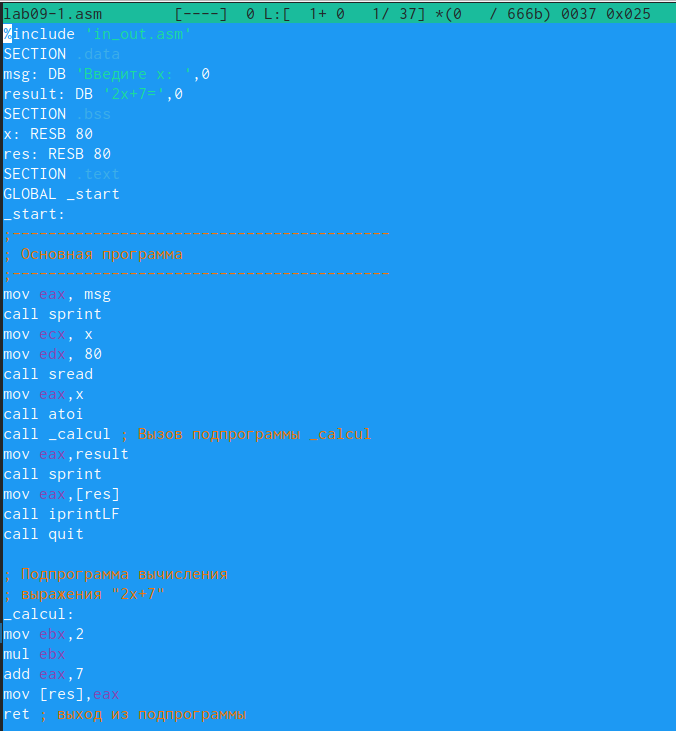


РИС.2 Редактирование файла

Создадим исполняемый файл и запустим его. Мы видим, что программа правильно работает и выдает правильно значение.(РИС.3)

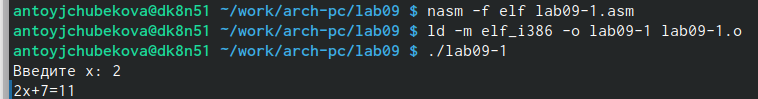


РИС.3 Запуск исполняемого файла

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.(РИС.4)

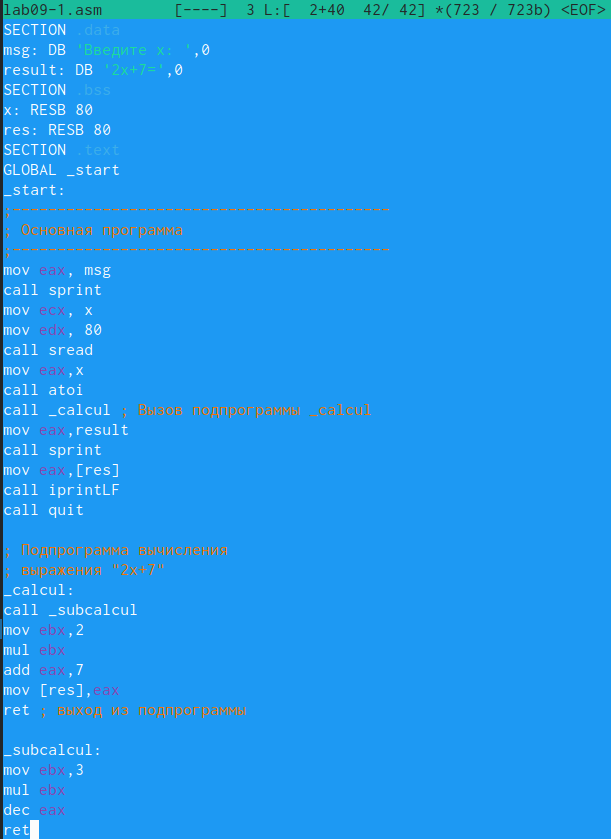


РИС.4 Редактирование файла

Создадим исполняемый файл и запустим его. Проверим работу программы введя число 2 и видим, что все работает правильно.(РИС.5)

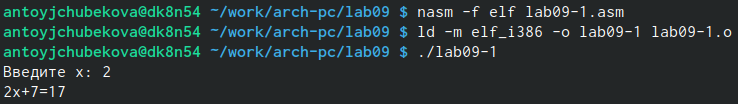


РИС.5 Запуск исполняемого фпйла

Создадим файл lab09-2.asm и введем в него текст программы вывода сообщения Hello world!(РИС.6)

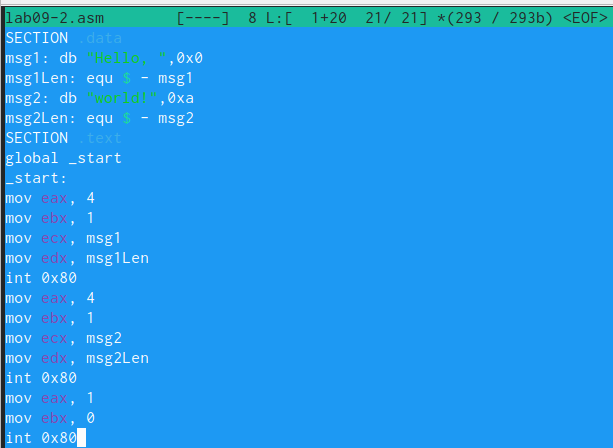


РИС.6 Редактирование файла

Создадим исполняемый файл для работы с GDB с посощью команд:  
nasm -f elf -g -l lab09-2.lst lab09-2.asm  
ld -m elf\_i386 -o lab09-2 lab09-2.o  
Запустим исполняемый файл в отладчик gdb.(РИС.7)

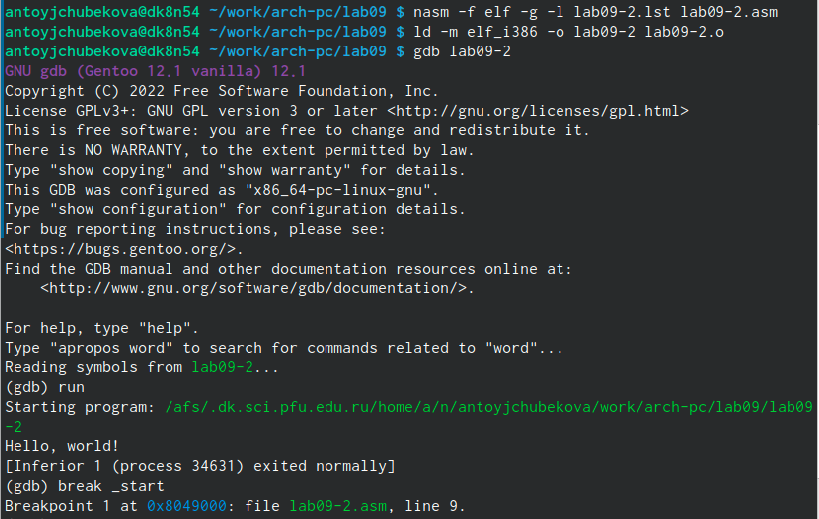


РИС.7 Запус исполняемого файла с gdb

Проверим работу программы, запустив ее в оболоче GDB с помощью команды run(РИС.8)

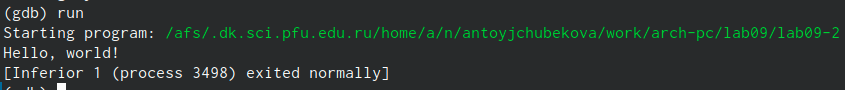


РИС.8 Проверка работы файла с помощью команды run

Для более подробного анализа программы установим breakpoint на метку \_start и запустим ее.(РИС.9)

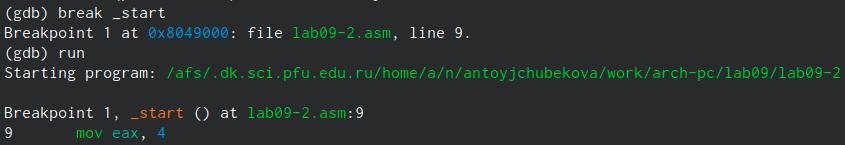


РИС.9 Установка breakpoint и запуск программы

Просмотрим дисассимилированный код программы с помощью команды disassemble начиная с метки \_start. Далее переключимся на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel(РИС.10)

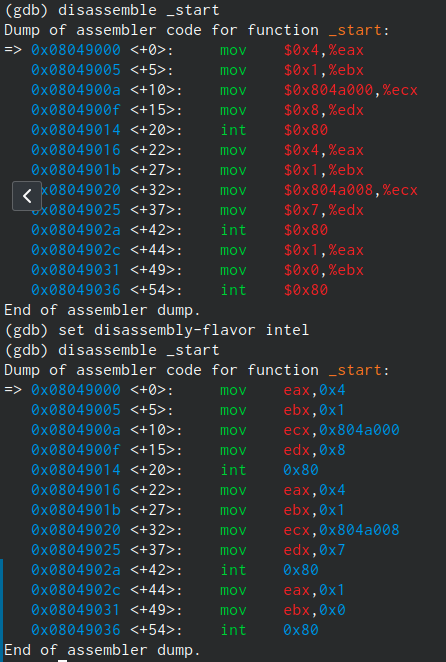


РИС.10 Переключение дисассимилированного кода на intel’овский синтаксис

В пежимк ATT имена регистров начинаются с символа %, а имена опрерандов с $, в то время как в intel используется привычный нам синтаксис.

Включим режим перфографики для более удобного анализа программы.(РИС.11)

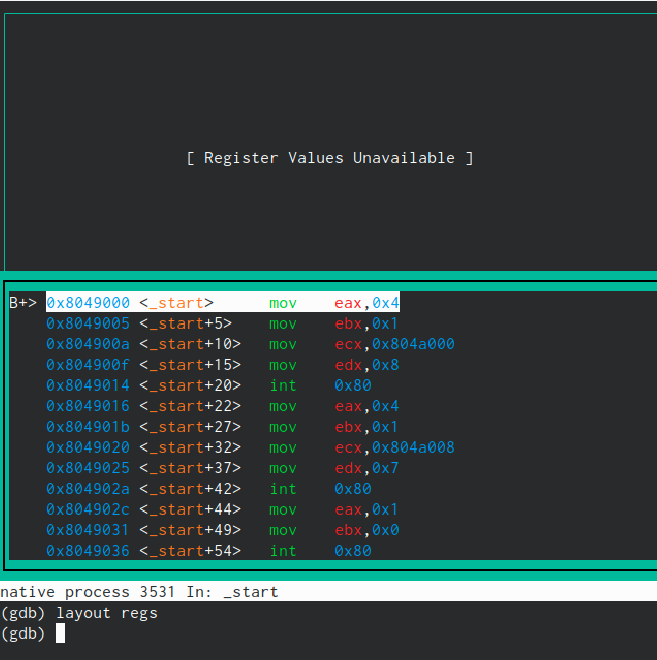


РИС.11 Включение режима псевлографики

Проверим установленный breakpoint по имени метки \_start c помощью команды info breakpoints.(РИС.12)

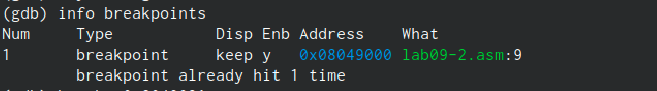


РИС.12 Информация о точке останова

Установим еще одну точку основа по адресу инструкции mov ebx,0x0. Просмотрим информацию о всех установленных точках останова введя i b.(РИС.13)

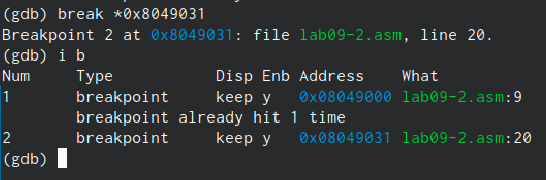


РИС.13 Установка точки останова и просмотр информации о всех точках останова

Выполним 5 инструкций с помощью команды stepi и наблюдаем за изменением значений регистров.(РИС.14) Мы видим, что изменились значение регистров eax,ecx,edx,ebx(РИС.15)

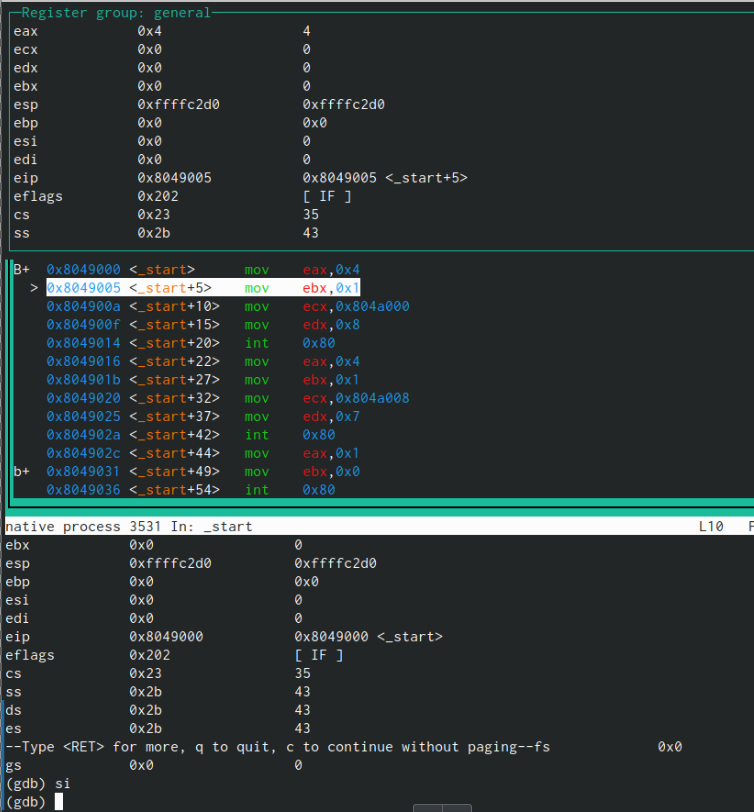


РИС.14 Выполнение 5 инструкций с помощью команды stepi

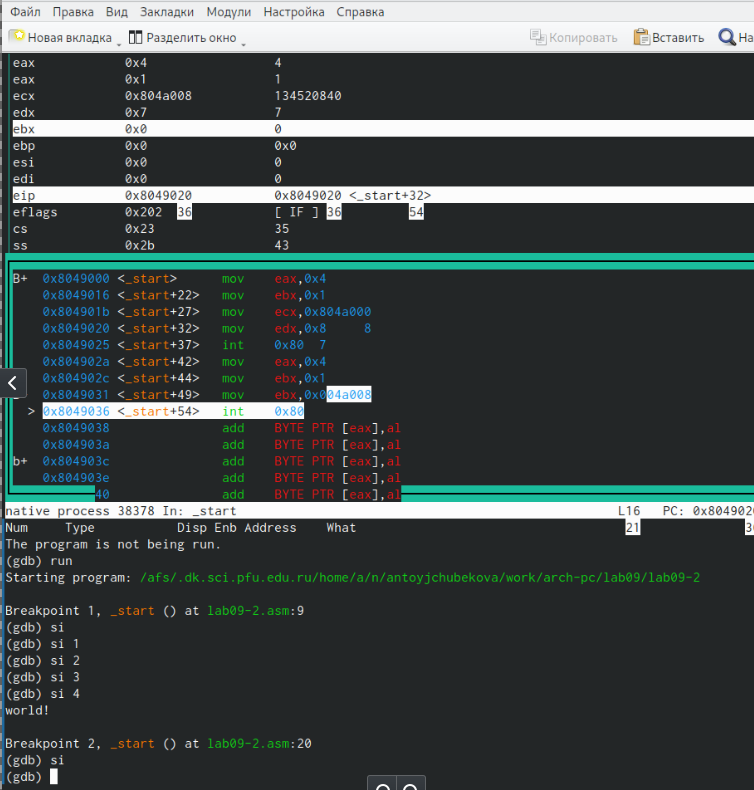


РИС.15 Выполнение 5 инструкций с помощью команды stepi

Просмотрим значение переменной msg1 по имени используя команду x/1sb &msg1, также просмотрим значение переменной msg2 по ее адресу.(РИС.16)

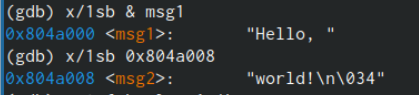


РИС.16 Просмотр значений переменных msg1,msg2

Используя команду set изменим второй символ переменной msg1 на h.(РИС.17)

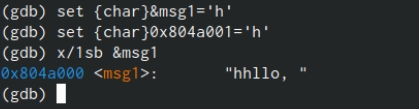


РИС.17 Изменение символа с помощью set

Заменим первый символ в переменной msg2 на f(РИС.18)

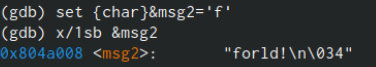


РИС.18 Изменение символа msg2

Выведим значение регистра edx в шестнадцатиричном, двоичном формах и символьном виде.(РИС.19)

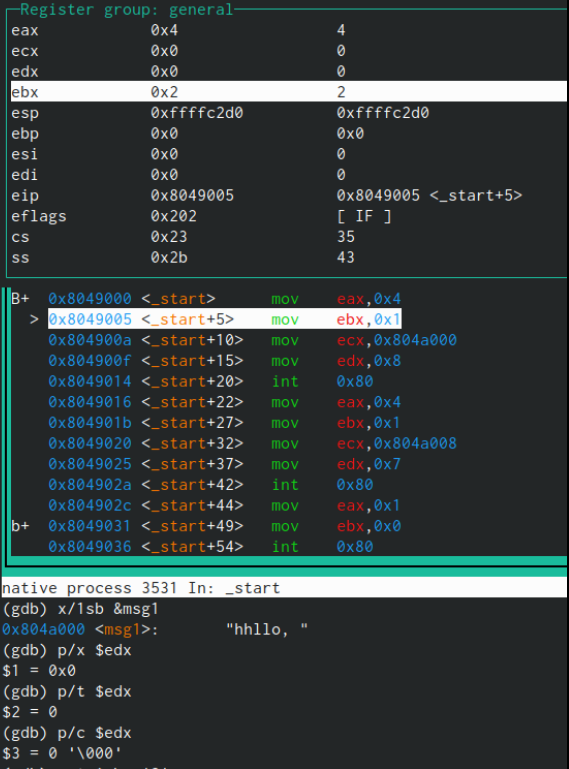


РИС.19 Вывод значения регистра в разных формах

С помощью команды set изменим значение регистра ebx в соответствии с заданием.(РИС.20)

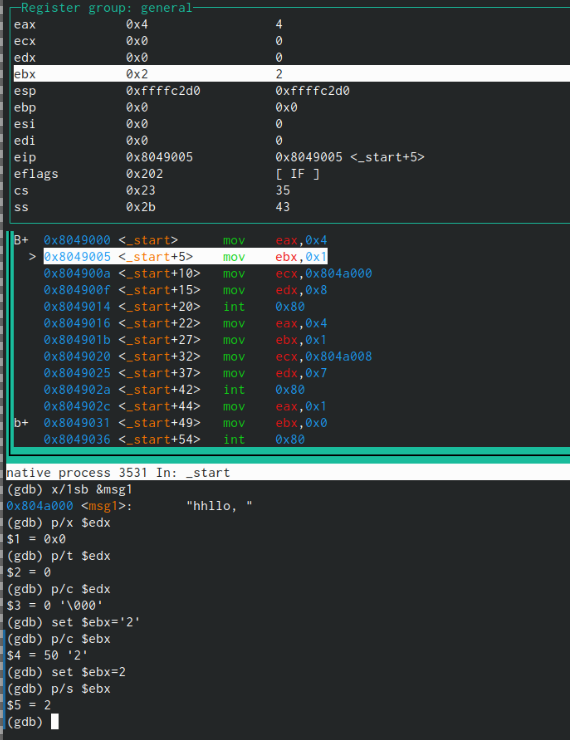


РИС.20 Изменение значений регистра

Разница вывода команд p/s $ebx отличается тем, что в первом случае мы переводим символ в его строковой вид, а во втором случае число в строковом виде не изменяется.

Завершим выполнение программы с помощью команды countinue и выйдем из GDB с помощью команды quit.(РИС.21)

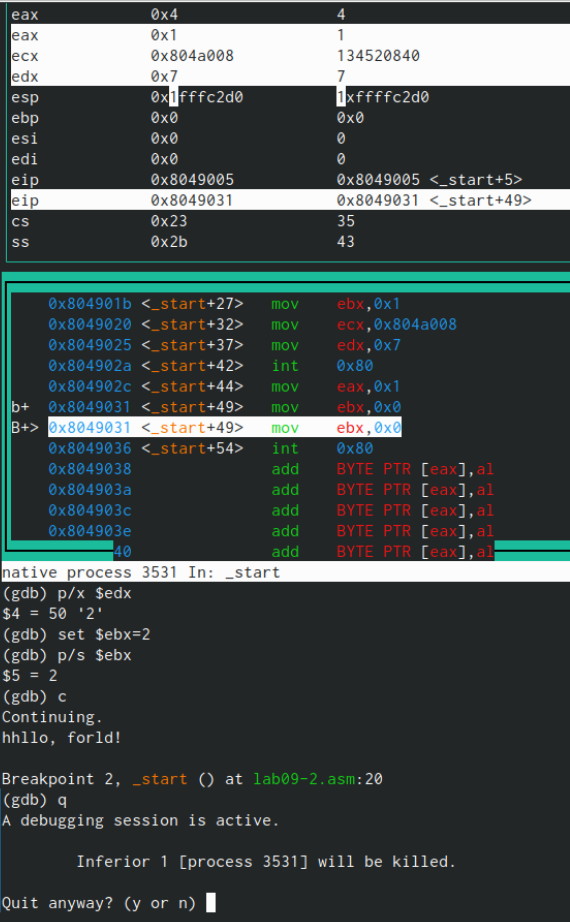


РИС.21 Завершение работы GDB

Скопируем файл lab8-2.asm, созданный при выполнении лабораторной работы № 8 в файл с именем lab09-3.asm.(РИС.22)

РИС.22 Копирование файла

РИС.22 Копирование файла

Создадим исполняемый файл.(РИС.23)

РИС.23 Создание исполняемого файла

РИС.23 Создание исполняемого файла

Загрузим исполняемый файл в отладчик gdb, указав аргументы с использованием ключа –args(РИС.24)

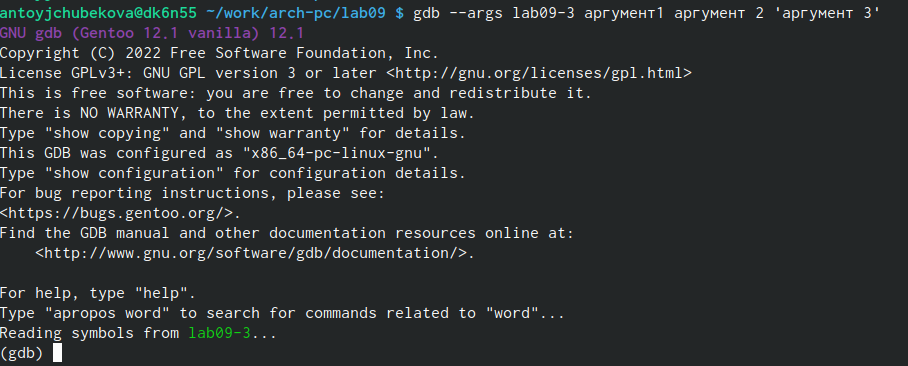


РИС.24 Загрузка файла с аргументами в отладчик

Установим точку останова перед первой инструкцией в программе и запустим ее.(РИС.25)

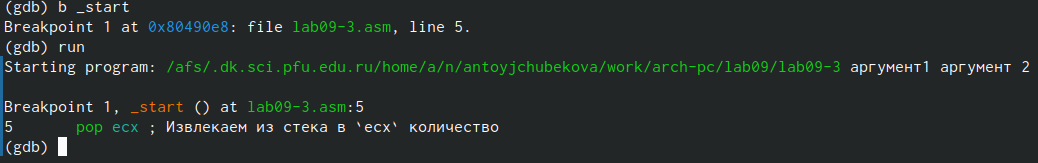


РИС.25 Установка точки останова и запуск программы

Просмотрим вершину стека и остальные позиции стека по их адресам.Шаг изменения адреса равна 4, потому что количество аргументов командной строки равно 4.(РИС.26)

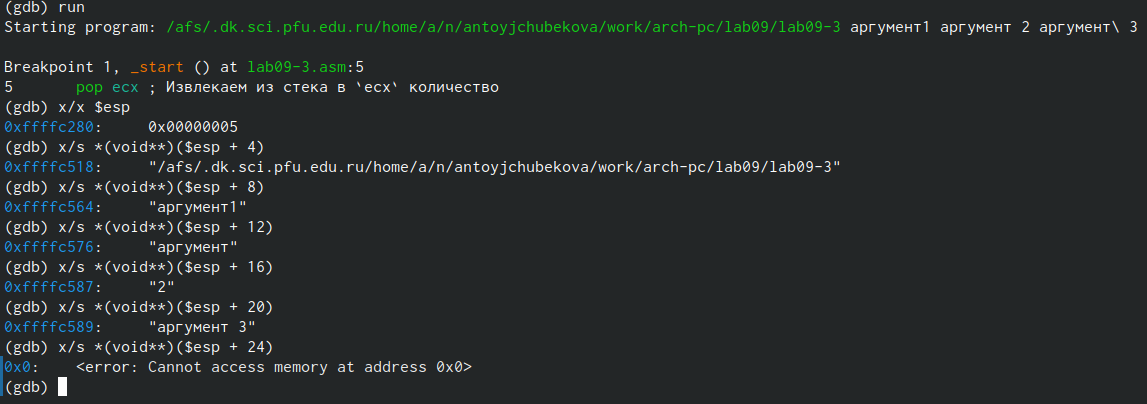


РИС.26 Просмотр значений в стеке

## 3.1 Задание для самостоятельной работы

Скопиров в файл lab08-4.asm в новый файл lab09-4.asm преобразуем программу в этлм файле так, чтобы вычесление значения функции реализовывалась как подпрограмма.(РИС.27) Сама программа выглядит следующим образом:  
%include ‘in\_out.asm’

SECTION .data  
msg db “Результат:”,0

SECTION .text  
global \_start  
\_start:

pop ecx

pop edx

sub ecx,1

mov esi, 0

next:  
cmp ecx,0h  
jz \_end

pop eax  
call atoi  
call podprogramma

loop next  
\_end:

mov eax, msg  
call sprint  
mov eax, esi  
call iprintLF  
call quit

podprogramma:  
inc eax  
mov ebx,7  
mul ebx  
add esi,eax  
ret.



РИС.27 Редактирование файла

Создадим исполняемый файл и запустим его. Введя значения 1 2 3 4, мы видим, что программа работает правильно.(РИС.28)

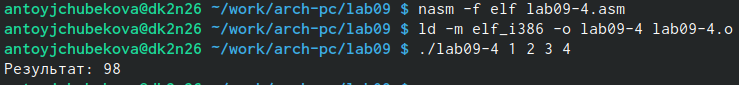


РИС.28 Запуск программы

Создаю файл lab09-5.asm и ввожу в него текс программы вычисления выражения (3 + 2)\*4 + 5(РИС.29)

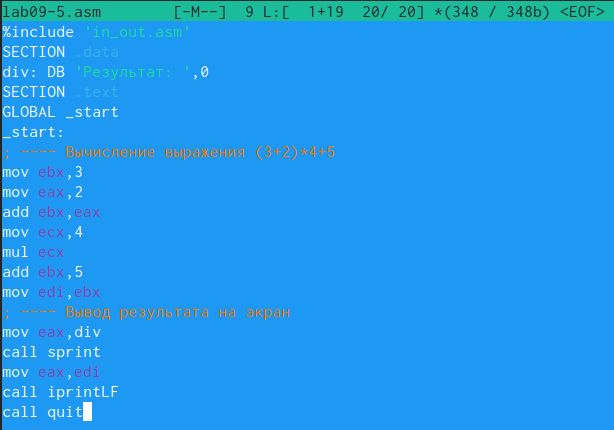


РИС.29 Редактирование файла

Создадим исполняемый файл и запускаем ее. Мы видим, что программа работает неправильно, так как она выводит 10 вместо 25.(РИС.30)

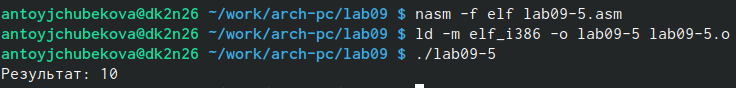


РИС.30 Запуск программы

Получим исполняемый файл для работы с GDB, запустим ее и ставим breakpoint во всех инструкциях, в которых происходит вычесления. С помощью команды continui проверяю каждую точку останова следя за значением регистров. При выполнении инструкции mul ecx, значение регистра eax должно было равняться 20, но она равнялась 8,это происходило потому что после выполнения инструкции add ebx,eax результат сложения был сохранен в регистр ebx, в не врегист eax. В итоге после последней инструкции у насполучилось 10 так как программа прибовляет 5 как раз таки к регистру ebx которое равно 5.(РИС.31) и (РИС.32)

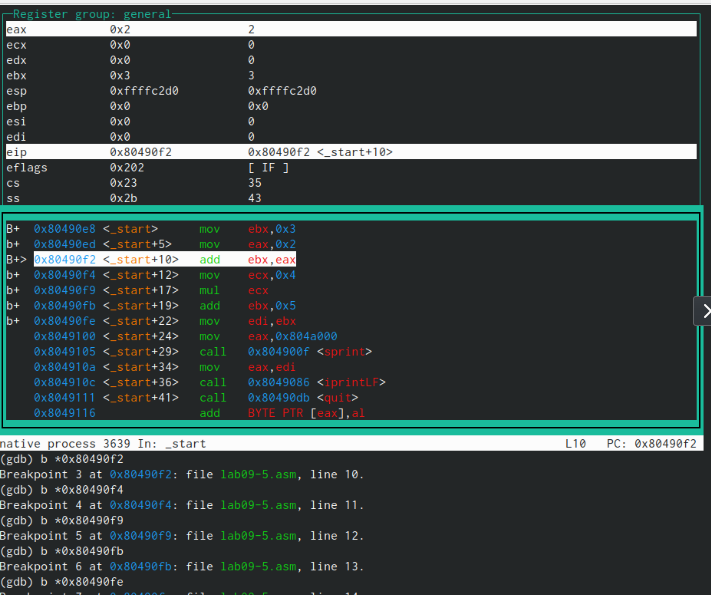


РИС.31 Постановка breakpoint

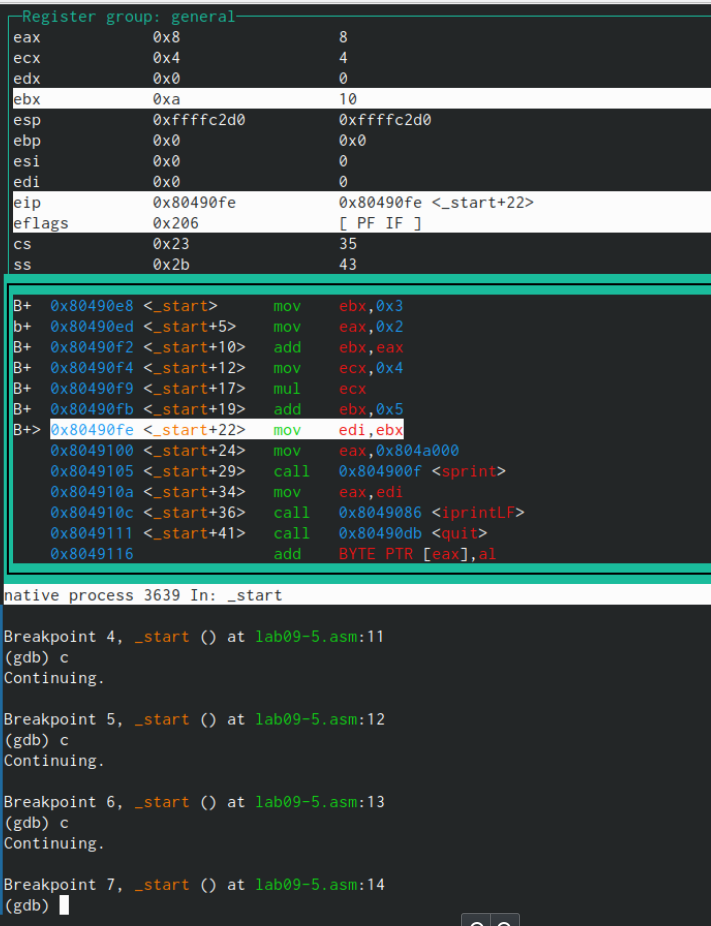


РИС.32 Наблюдения за значениями регистров

Исправляем ошибку, добавляя после add ebx,eax mov eax,ebx и заменяя ebx на eax в инструкциях add ebx,5 и mov edi,ebx.(РИС.33) Правильный код выглядит следующим образом:  
include ‘in\_out.asm’  
SECTION .data  
div: DB ‘Результат:’,0  
SECTION .text  
GLOBAL \_start  
\_start:  
; —- Вычисление выражения (3+2)\*4+5  
mov ebx,3  
mov eax,2  
add eax,ebx  
mov ecx,4  
mul ecx  
add eax,5  
mov edi,eax  
; —- Вывод результата на экран  
mov eax,div  
call sprint  
mov eax,edi  
call iprintLF  
call quit.

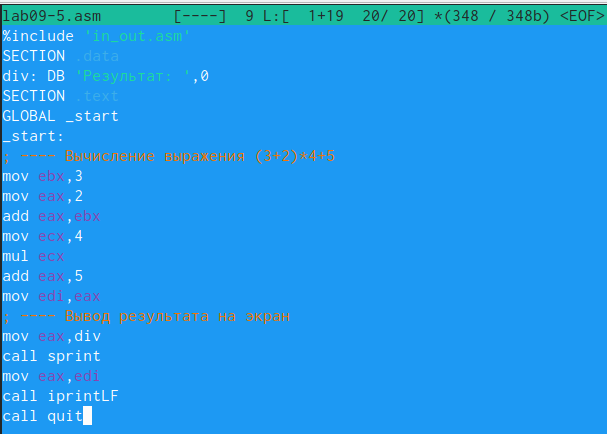


РИС.33 Редактирование файла

Создадим исполняемый файл и запустим его. Мы видим, что программа работает правильно и выводит 25.(РИС.34)

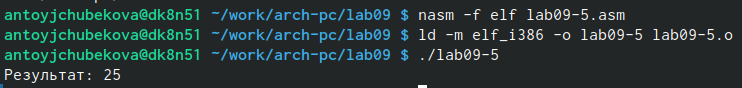


РИС.34 Запуск программы

# 4 Выводы

При выполнении данной лабораторной работы я приобрела навыки написание программ с использованием подпрограмм. Также познакомилась с методом откладки при помощи GDB и его основными возможностями. Используя полученные навыки я отредактировала программу, написанную мной при выполнении лабораторной работы №8 так,чтобы она работала с помощью подпрограмм. Вместе с тем я нашла ошибку в программе используя отладчик GDB.

# Список литературы

-https://esystem.rudn.ru/course/view.php?id=4975.