Отчёт по лабораторной работе №9

Дисциплина: Архитектура компьютера

Бызова Мария Олеговна

Содержание

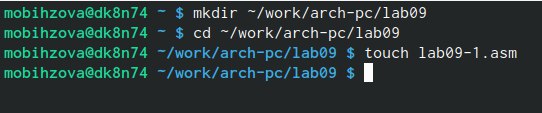
# 1 Цель работы

Целью лабораторной работы является приобретение навыков написания программ с использованием подпрограмм, знакомство с методами отладки при помощи GDB и его основными возможностями.

# 2 Выполнение лабораторной работы

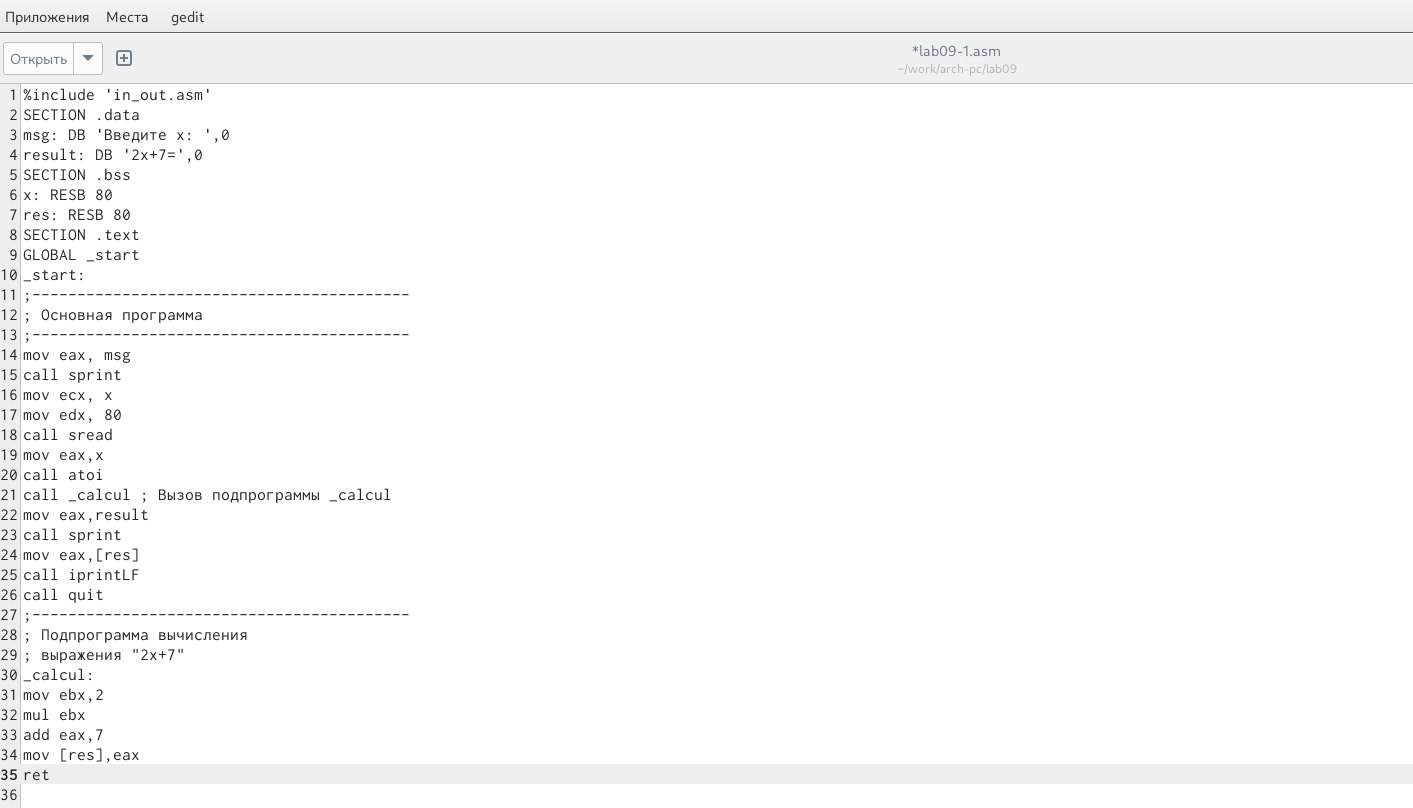
## 2.1 Реализация подпрограмм в NASM

1. С помощью утилиты mkdir создаем директорию, в которой будем создавать файлы с программами для лабораторной работы №9. Переходим в созданный каталог с помощью утилиты cd. С помощью утилиты touch создаем файл lab09-1.asm (рис. [??]).



Создание необходимой директории и файла

1. Внимательно изучив текст программы из листинга 9.1, вводим его в файл lab09-1.asm (рис. [??]).

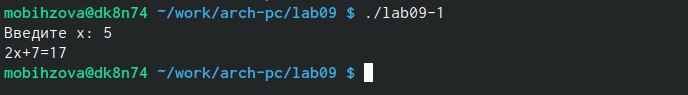


Редактирование файла

Далее создаем исполняемый файл и запускаем его (рис. [??], [??]).

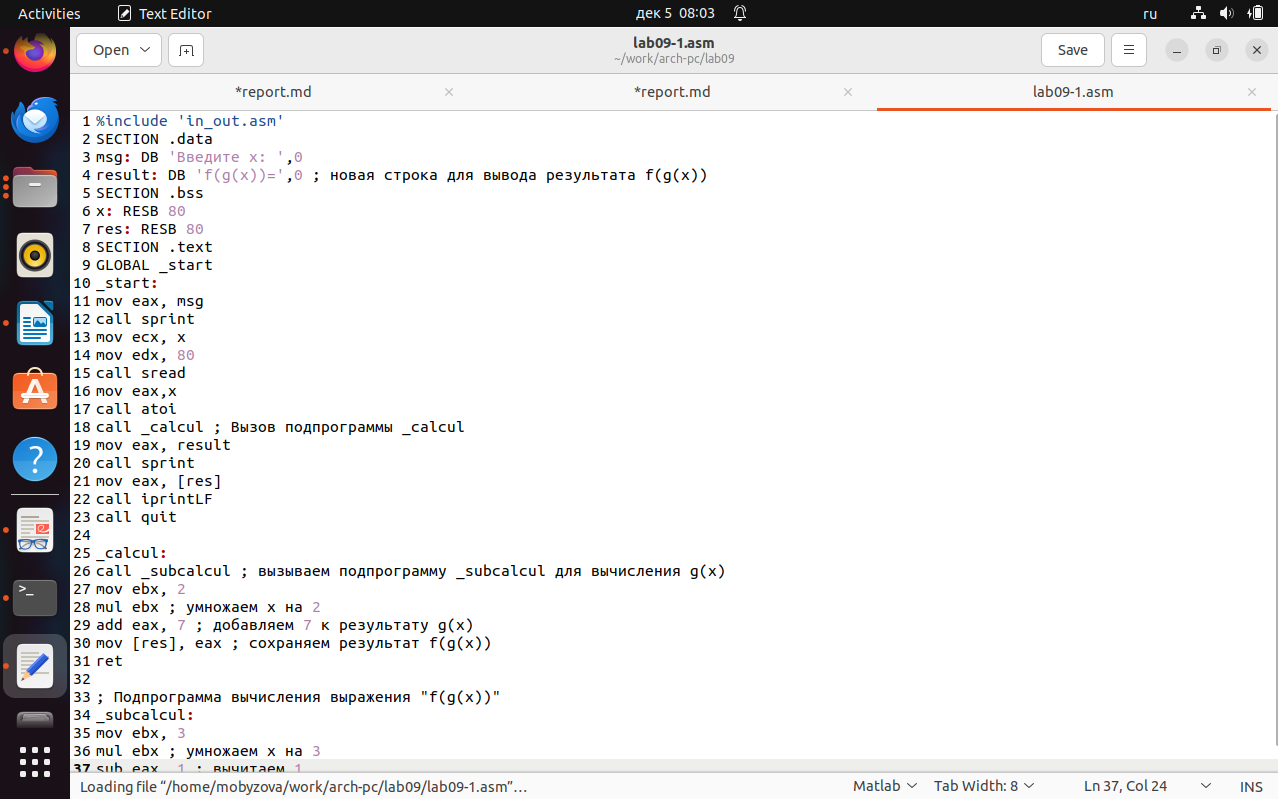
Создание исполняемого файла

Создание исполняемого файла



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. [??]).

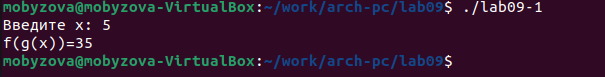


Редактирование файла

Далее создаем исполняемый файл и запускаем его (рис. [??], [??]).

Создание исполняемого файла

Создание исполняемого файла



Запуск исполняемого файла

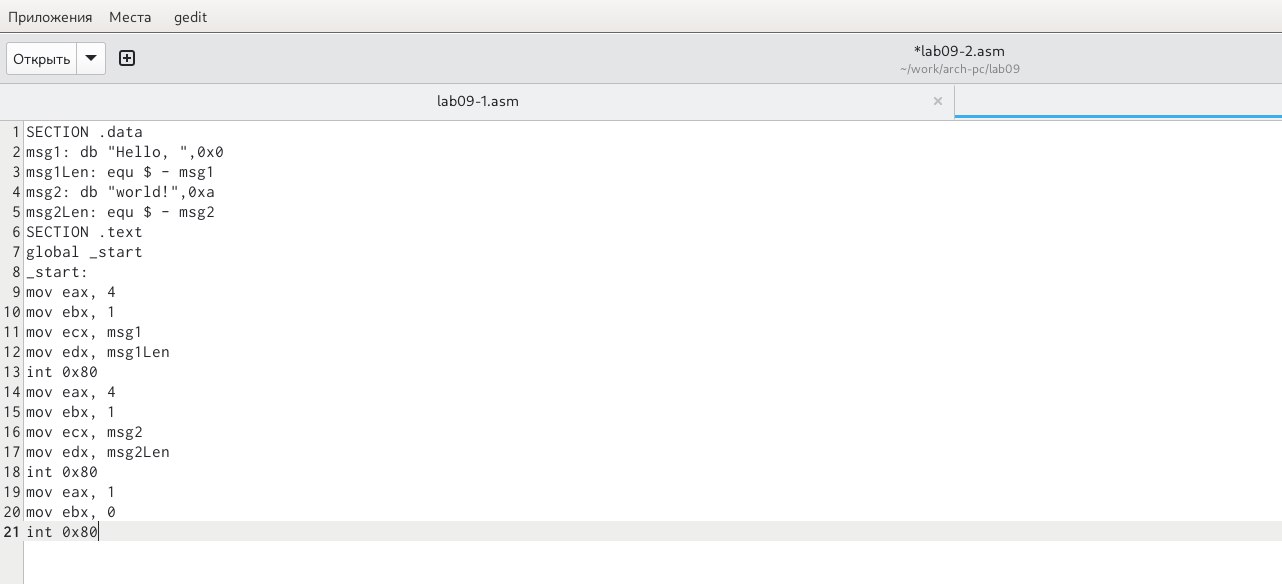
## 2.2 Отладка программам с помощью GDB

Создаем новый файл lab09-2.asm в каталоге (рис. [??]).

Создание необходимого файла

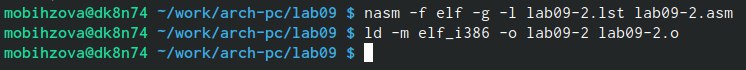
Создание необходимого файла

Открываем файл и заполняем его в соответствии с листингом 9.2 (рис. [??]).



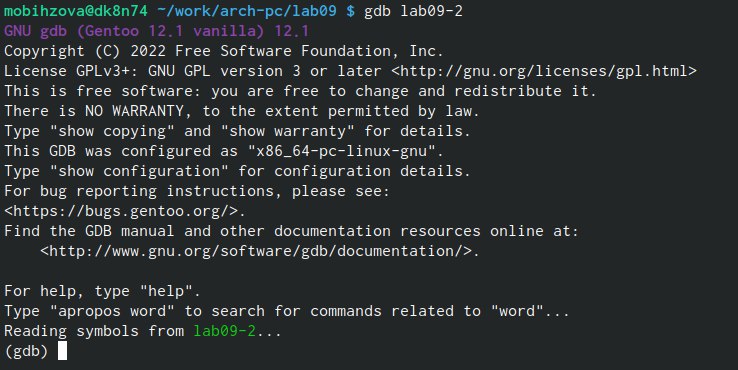
Редактирование файла

Получим исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’ (рис. [??]).



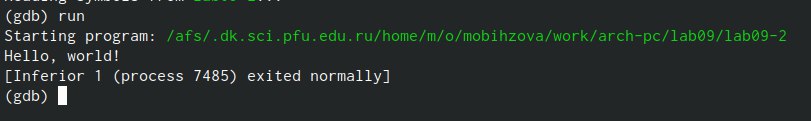
Получение исполняемого файла

Загрузим исполняемый файл в отладчик gdb (рис. [??]).



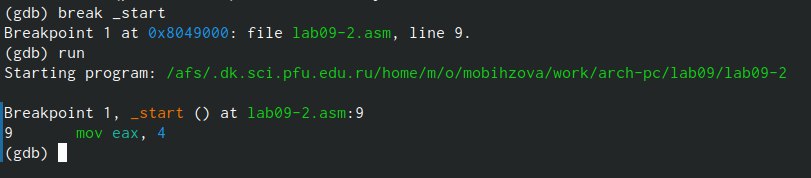
Загрузка исполняемого файла в отладчик gdb

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



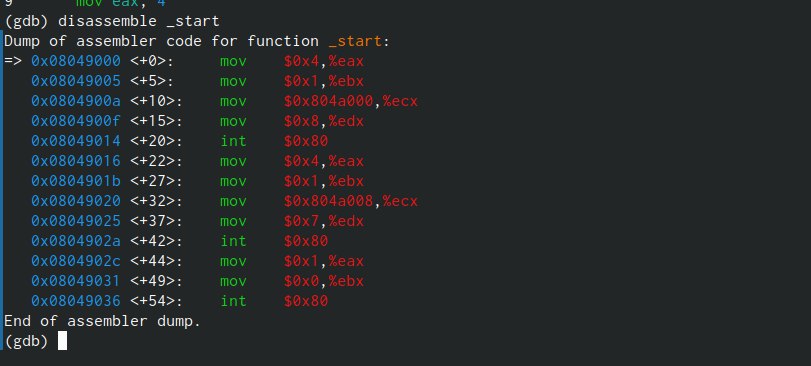
Проверка работы программы

Для более подробного анализа программы установим брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запустим её (рис. [??]).



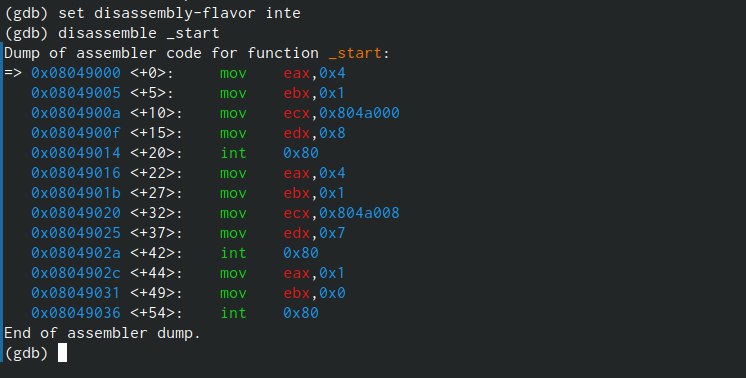
Запуск программы с брэйкпоинтом

Посмотрим дисассимилированный код программы с помощью команды disassemble начиная с метки \_start (рис. [??]).



Дисассимилированный код программы

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



Синтаксис Intel

Различия отображения синтаксиса машинных команд в режимах ATT и Intel:

1. Порядок операндов: В ATT синтаксисе порядок операндов обратный, сначала указывается исходный операнд, а затем - результирующий операнд. В Intel синтаксисе порядок обычно прямой, результирующий операнд указывается первым, а исходный - вторым.
2. Разделители: В ATT синтаксисе разделители операндов - запятые. В Intel синтаксисе разделители могут быть запятые или косые черты (/).
3. Префиксы размера операндов: В ATT синтаксисе размер операнда указывается перед операндом с использованием префиксов, таких как “b” (byte), “w” (word), “l” (long) и “q” (quadword). В Intel синтаксисе размер операнда указывается после операнда с использованием суффиксов, таких как “b”, “w”, “d” и “q”.
4. Знак операндов: В ATT синтаксисе операнды с позитивными значениями предваряются символом “$". В Intel синтаксисе операнды с позитивными значениями могут быть указаны без символа "$”.
5. Обозначение адресов: В ATT синтаксисе адреса указываются в круглых скобках. В Intel синтаксисе адреса указываются без скобок.
6. Обозначение регистров: В ATT синтаксисе обозначение регистра начинается с символа “%”. В Intel синтаксисе обозначение регистра может начинаться с символа “R” или “E” (например, “%eax” или “RAX”).

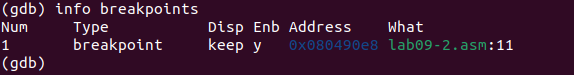
Включим режим псевдографики для более удобного анализа программы (рис. [??]).



Режим псевдографики

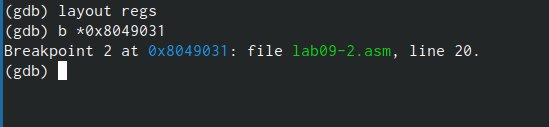
### 2.2.1 Добавление точек останова

Проверим установленные точки останова с помощью команды info breakpoints (рис. [??]).



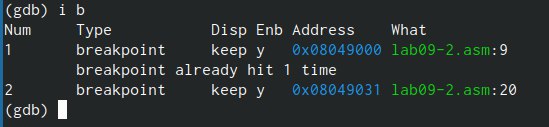
Информация о точках останова

Определим адрес предпоследней инструкции (mov ebx,0x0) и установим точку останова (рис. [??]).



Установка точки останова

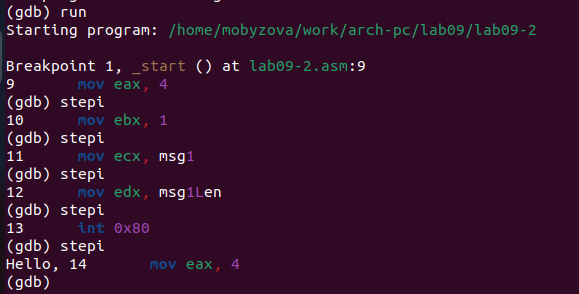
Посмотрим информацию о всех установленных точках останова (рис. [??]).



Информация о точках останова

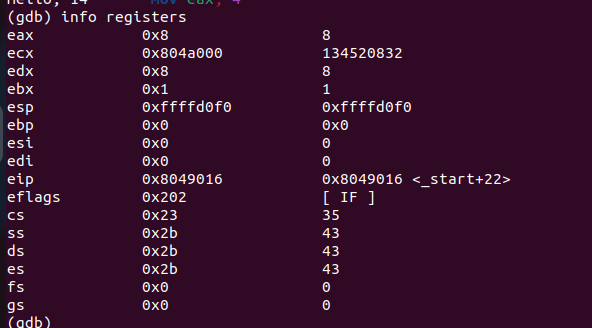
### 2.2.2 Работа с данными программы в GDB

Выполните 5 инструкций с помощью команды stepi (или si) (рис. [??]).



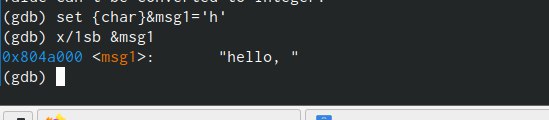
Выполняем 5 инструкций командой stepi

Посмотрев содержимое регистров с помощью команды info registers (или i r), замечаем, что во время выполнения команд менялись регистры: ebx, ecx, edx, eax, eip (рис. [??]).



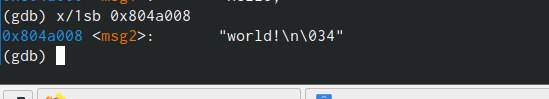
Информация о регистрах

С помощью команды x & также можно посмотреть содержимое переменной. Посмотрим значение переменной msg1 по имени (рис. [??]).



Значение переменной

Смотрим значение переменной msg2 по адресу (рис. [??]).



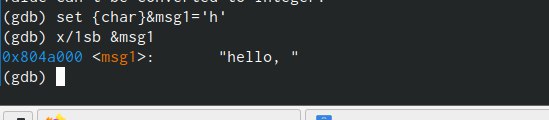
Значение переменной

Посмотрим инструкцию mov ecx, msg2 которая записывает в регистр ecx адрес перемененной msg2 (рис. [??]).

Просмотр инструкции

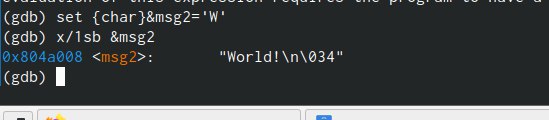
Просмотр инструкции

Изменим первый символ переменной msg1 (рис. [??]).



Изменение первого символа переменной msg1

Изменим первый символ переменной msg2 (рис. [??]).



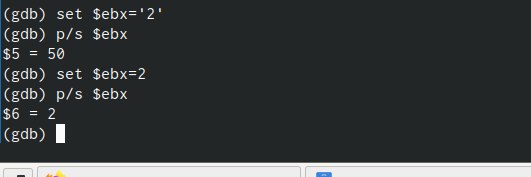
Изменение первого символа переменной msg2

Выведем в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx (рис. [??]).



Значение регистра edx

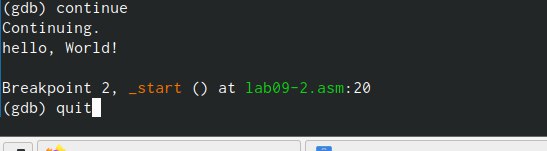
Изменим регистр ebx (рис. [??]).



Изменение регистра ebx

Выводятся разные значения, так как именно команда без кавычек присваивает регистру вводимое значение.

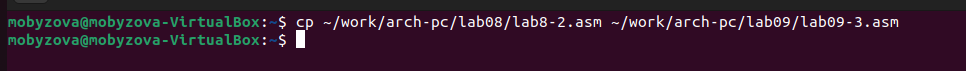
Завершим выполнение программы с помощью команды continue (сокращенно c) и выйдем из GDB с помощью команды quit (сокращенно q) (рис. [??]).



Завершение программы и выход из GDB

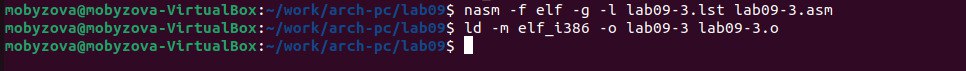
### 2.2.3 Обработка аргументов командной строки в GDB

Скопируем 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. [??]).



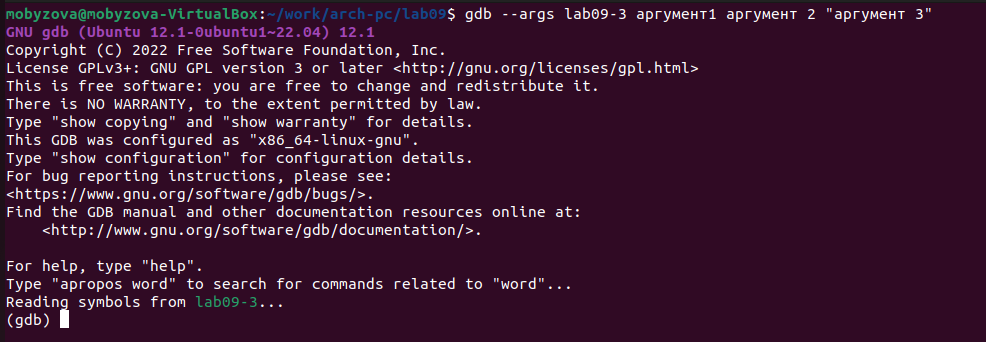
Копирование файла

Далее создадим исполняемый файл (рис. [??]).



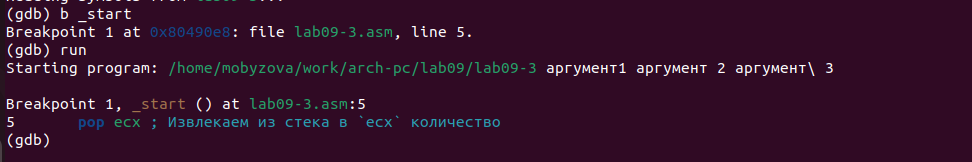
Создание исполняемого файла

Загрузим исполняемый файл в отладчик, указав аргументы (рис. [??]).



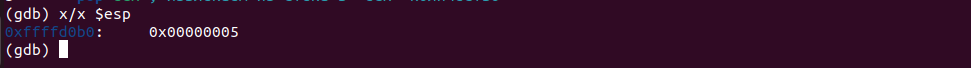
Загрузка исполняемого файла в отладчик

Для начала установим точку останова перед первой инструкцией в программе и запустим ее (рис. [??]).



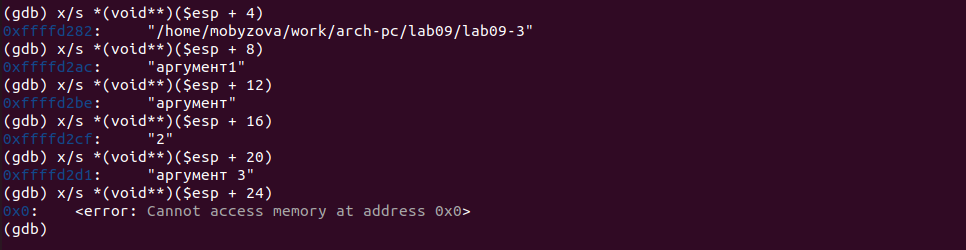
Установка точки останова и запуск программы

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы) (рис. [??]).



Адрес вершины стека

Посмотрим остальные позиции стека (рис. [??]).



Просмотр остальных позиций стека

Шаг изменения адреса равен 4, потому что в большинстве архитектур процессоров размер слова (или размер указателя) составляет 4 байта. Это означает, что каждый раз, когда мы обращаемся к следующему элементу в стеке, мы увеличиваем адрес на 4, чтобы перейти к следующему слову данных. Таким образом, шаг изменения адреса равен 4 для обеспечения корректного доступа к данным в стеке.

## 2.3 Выполнение заданий для самостоятельной работы

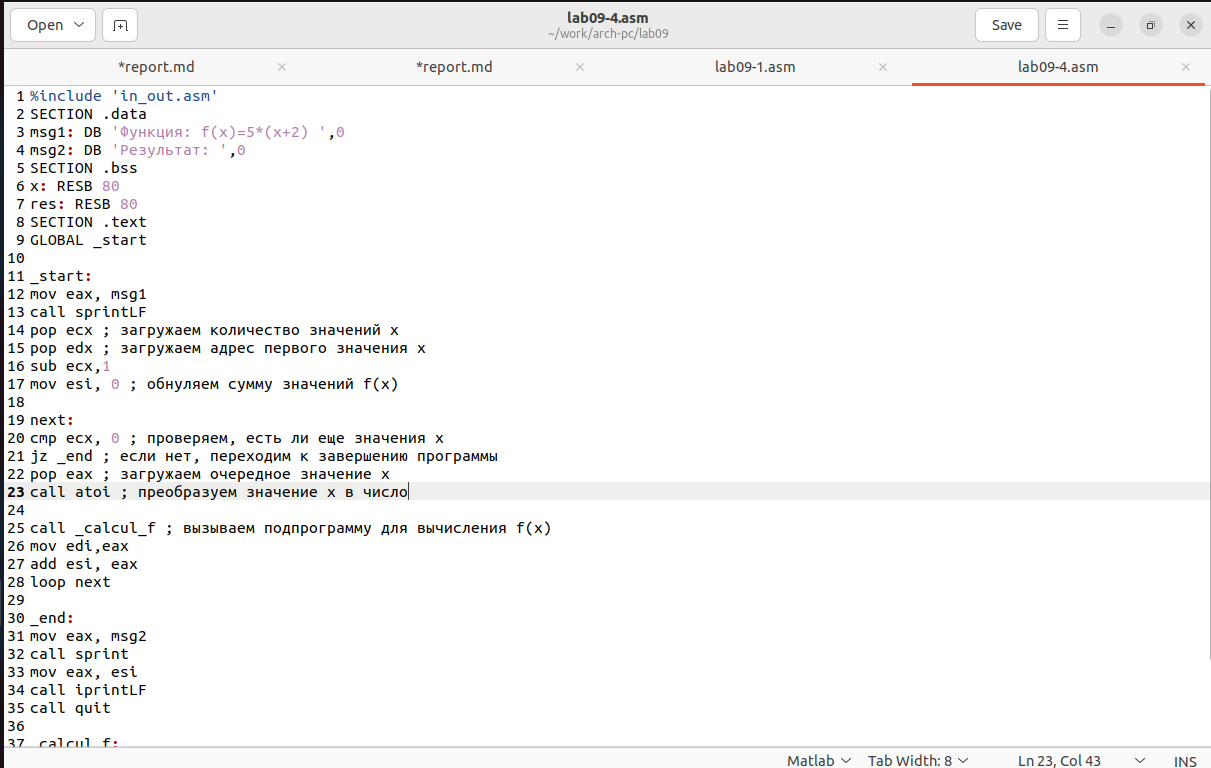
1. Преобразуем программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму.

Создаем файл для выполнения задания (рис. [??]).

Создание файла

Создание файла

Вставляем отредактированную программу из лабораторной работы №8 с добавлением подпрограммы (рис. [??]).

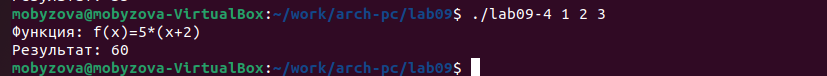


Редактирование файла

Далее создаем исполняемый файл и запускаем его (рис. [??], [??]).

Создание исполняемого файла

Создание исполняемого файла



Запуск исполняемого файла

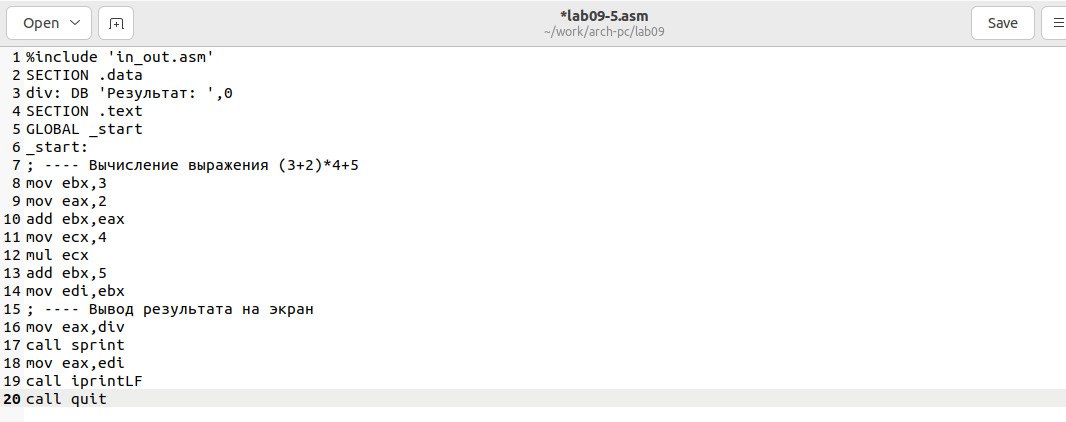
1. В листинге 9.3 приведена программа вычисления выражения (3 + 2) \* 4 + 5. При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее.

Создаем файл для выполнения задания (рис. [??]).

Создание файла

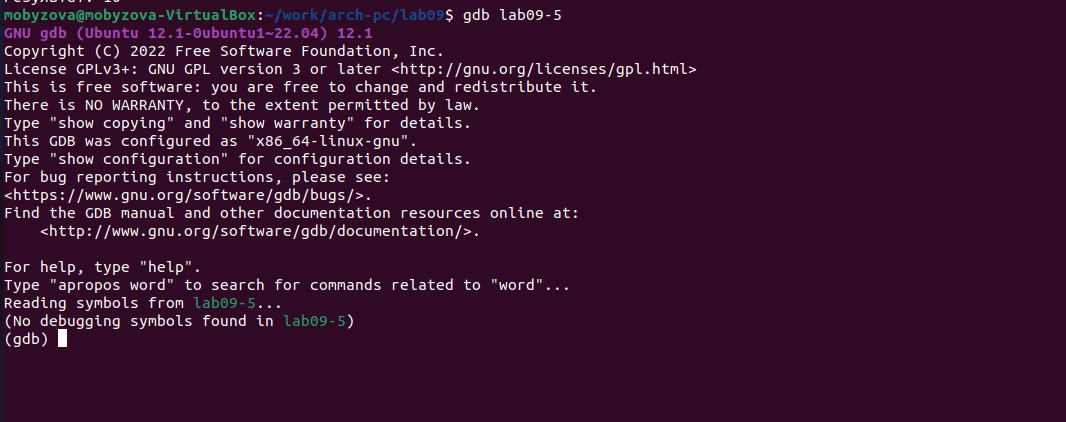
Создание файла

Вставляем программу (рис. [??]).

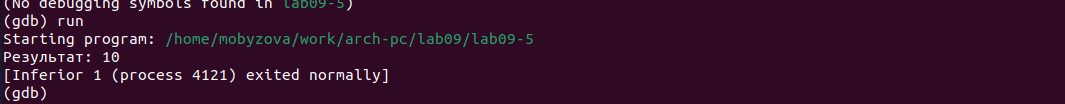


Редактирование файла

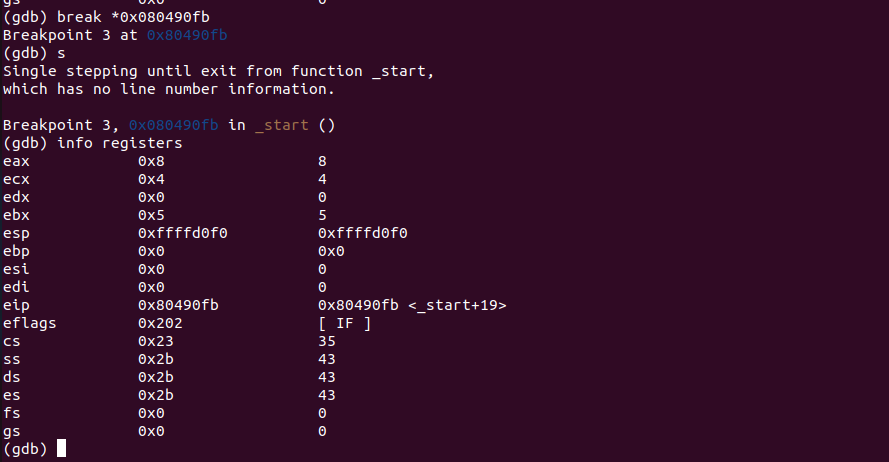
Создаем исполняемый файл и запускаем его в отладчике GDB. Смотрим на изменение регистров по ходу программы (рис. [??], [??], [??]).



Загрузка файла в отладчик



Запуск программы



Изменение регистров

Обнаружив ошибку неправильной записи регистров, корректируем программу (рис. [??]).



Редактирование файла

Далее создаем исполняемый файл и запускаем его (рис. [??], [??]).

Создание исполняемого файла

Создание исполняемого файла

Запуск исполняемого файла

Запуск исполняемого файла

**Листинг №1. Изменённая программа**

%include 'in\_out.asm'   
SECTION .data   
msg: DB 'Введите x: ',0   
result: DB 'f(g(x))=',0 ; новая строка для вывода результата f(g(x))   
SECTION .bss   
x: RESB 80   
res: RESB 80   
SECTION .text   
GLOBAL \_start   
\_start:   
mov eax, msg   
call sprint   
mov ecx, x   
mov edx, 80   
call sread   
mov eax,x   
call atoi   
call \_calcul ; Вызов подпрограммы \_calcul  
mov eax, result  
call sprint  
mov eax, [res]  
call iprintLF  
call quit  
  
\_calcul:  
call \_subcalcul ; вызываем подпрограмму \_subcalcul для вычисления g(x)  
mov ebx, 2  
mul ebx ; умножаем x на 2  
add eax, 7 ; добавляем 7 к результату g(x)  
mov [res], eax ; сохраняем результат f(g(x))  
ret  
  
; Подпрограмма вычисления выражения "f(g(x))"  
\_subcalcul:  
mov ebx, 3  
mul ebx ; умножаем x на 3  
sub eax, 1 ; вычитаем 1  
ret

**Листинг №2. Задание для самостоятельной работы №1**

%include 'in\_out.asm'   
SECTION .data  
msg1: DB 'Функция: f(x)=5\*(x+2) ',0  
msg2: DB 'Результат: ',0  
SECTION .bss  
x: RESB 80  
res: RESB 80  
SECTION .text  
GLOBAL \_start  
  
\_start:  
mov eax, msg1  
call sprintLF  
pop ecx ; загружаем количество значений x  
pop edx ; загружаем адрес первого значения x  
sub ecx,1  
mov esi, 0 ; обнуляем сумму значений f(x)  
  
next:  
cmp ecx, 0 ; проверяем, есть ли еще значения x  
jz \_end ; если нет, переходим к завершению программы  
pop eax ; загружаем очередное значение x  
call atoi ; преобразуем значение x в число  
  
call \_calcul\_f ; вызываем подпрограмму для вычисления f(x)  
mov edi,eax  
add esi, eax  
loop next  
  
\_end:  
mov eax, msg2  
call sprint  
mov eax, esi  
call iprintLF  
call quit  
  
\_calcul\_f:  
add eax, 2 ; прибавляем 2 к x  
mov ebx, 5  
mul ebx ; умножаем результат на 5  
ret

**Листинг №3. Задание для самостоятельной работы №2**

%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  
; ----  
mov eax,div  
call sprint  
mov eax,edi  
call iprintLF  
call quit

# 3 Выводы

В ходе выполнения лабораторной работы мы приобрели навыки написания программ с использованием подпрограмм, познакомились с методами отладки при помощи GDB и его основными возможностями.