Отчёт по лабораторной работе №9

Дисциплина: Архитектура компьютера

Зайцева Ульяна Владимировна

Содержание

# 1 Цель 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 программам с помощью GDB.
3. Добавление точек останова.
4. Работа с данными программы в GDB.
5. Обработка аргументов командной строки в GDB.
6. Задания для самостоятельной 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.

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. 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. 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)

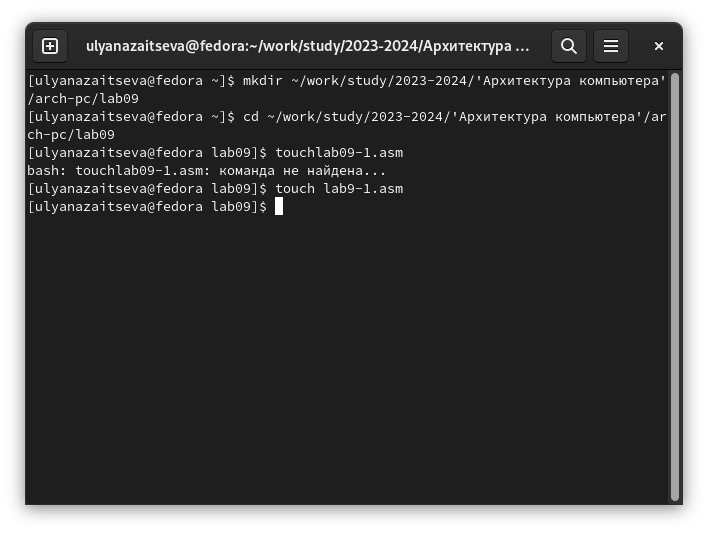
GDB (GNU Debugger — отладчик проекта GNU) [1] работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования. GDB предлагает обширные средства для слежения и контроля за выполнением компьютерных программ. От- ладчик не содержит собственного графического пользовательского 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. Отладчик GDB (как и любой другой отладчик) позволяет увидеть, что происходит «внутри» программы в момент её выполнения или что делает программа в момент сбоя. 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

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.

# 4 Выполнение лабораторной работы

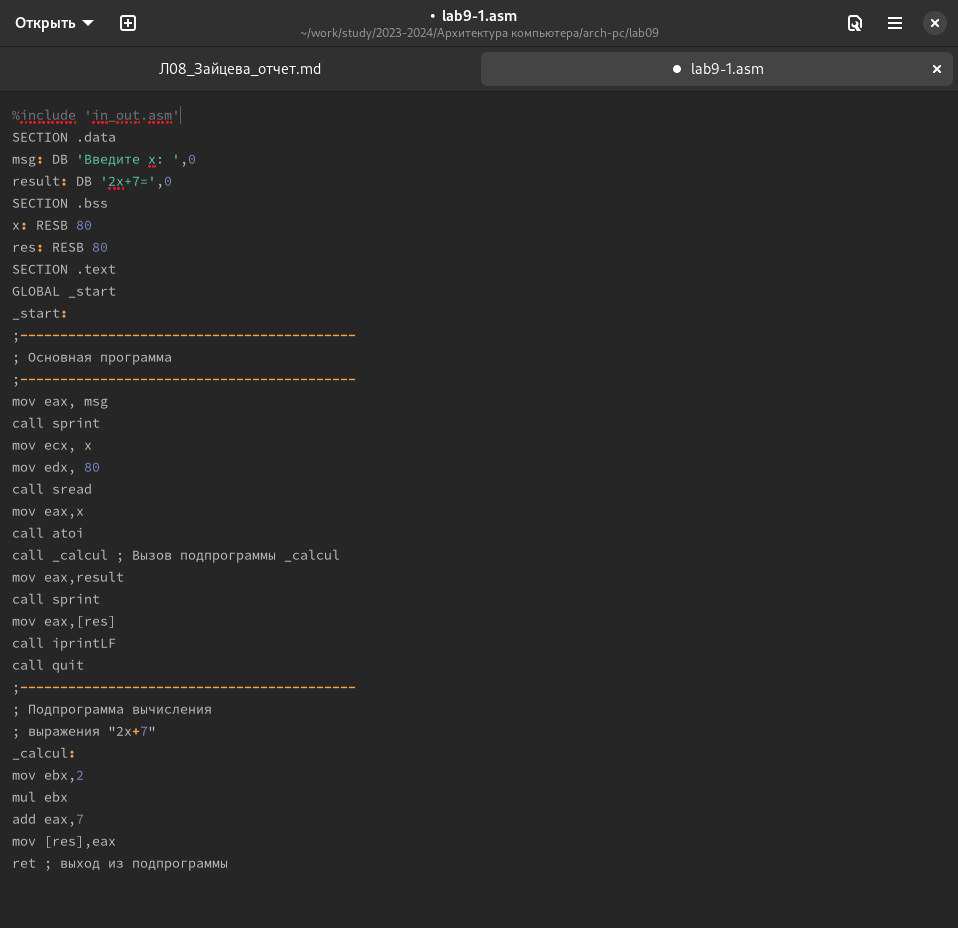
1. Реализация подпрограмм в NASM.

Создаю каталог для выполнения лабораторной работы №9, перейдите в него и создаю файл lab9-1.asm(рис. ??).



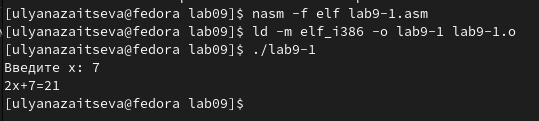
Создание каталога и файла

Ввожу в файл lab9-1.asm текст программы с использованием подпрограммы из листинга 9.1. (рис. ??)



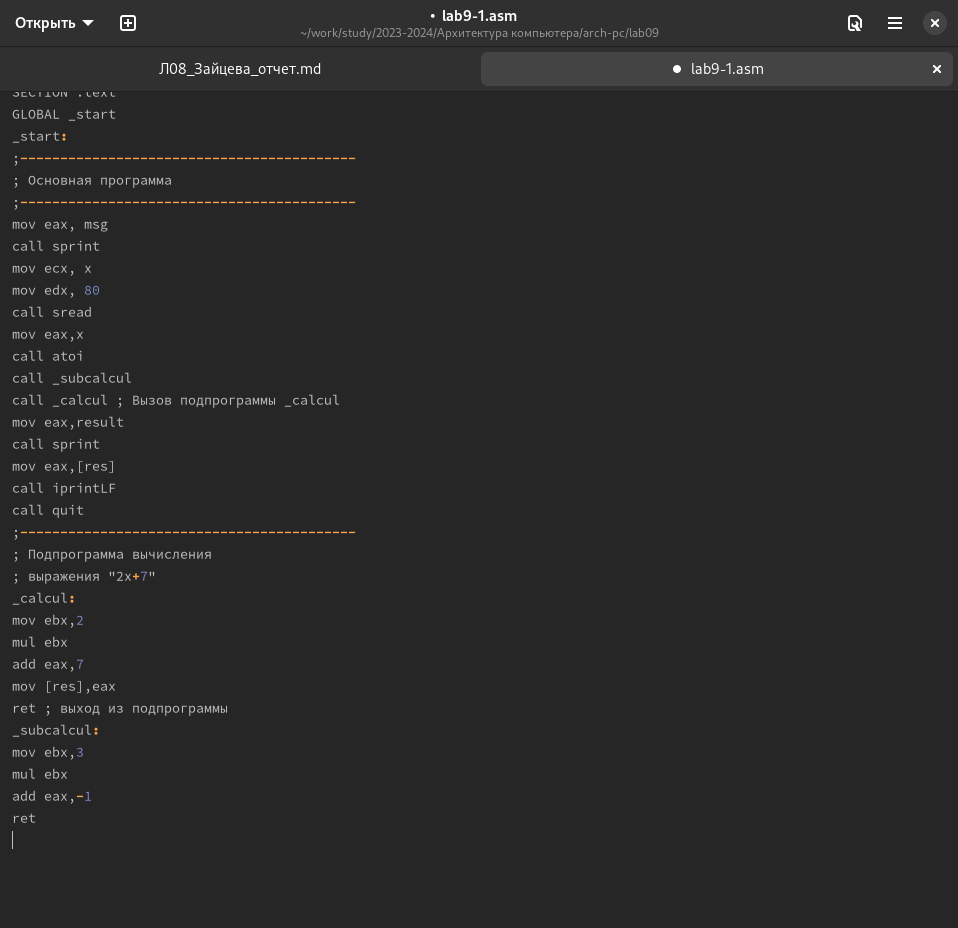
Ввод листинга 9.1

Создаю исполняемый файл и проверяю его работу(рис. ??)



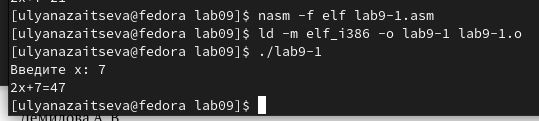
Проверка файла

Изменяю 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. ??)



Изменяю текст программы

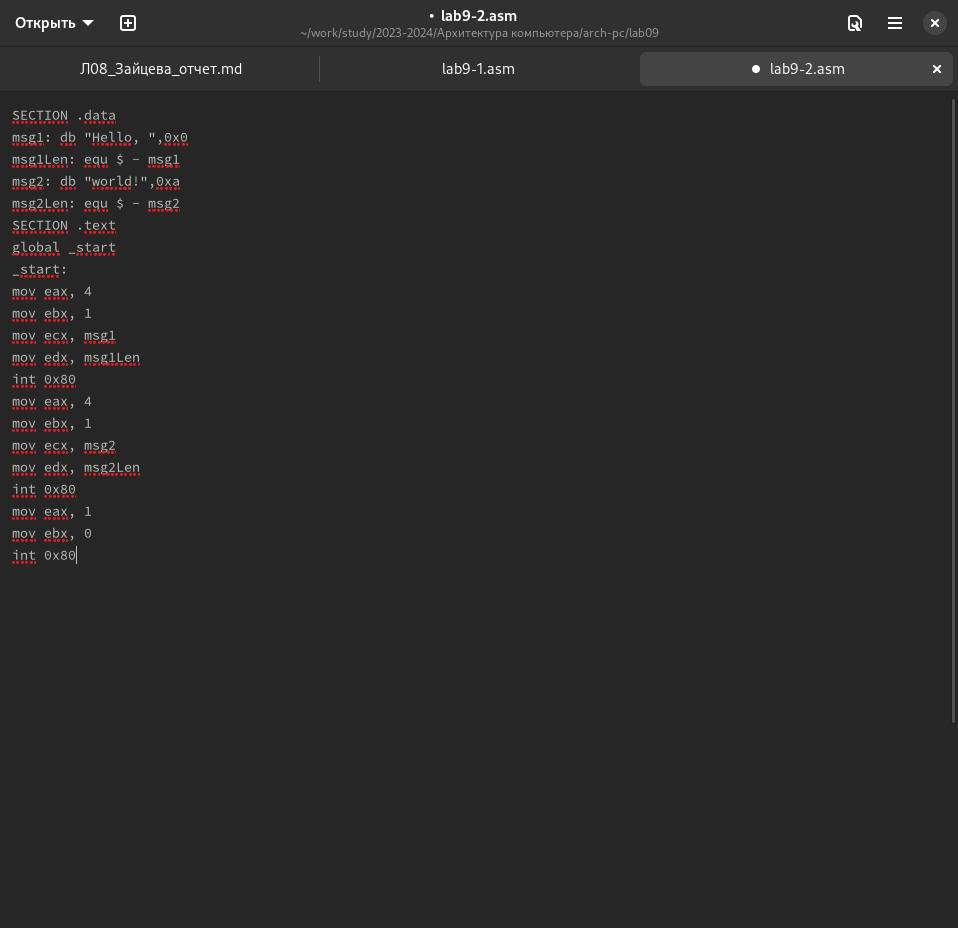
Создаю файл и проверяю его работу(рис. ??)



Проверка работы файла

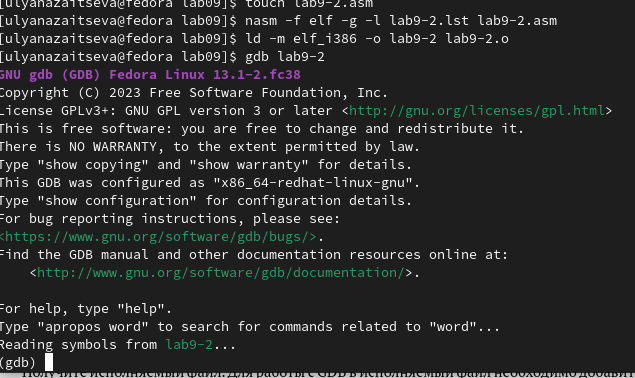
1. Отладка программам с помощью GDB.

Создаю файл lab9-2.asm с текстом программы из Листинга 9.2. (рис. ??)



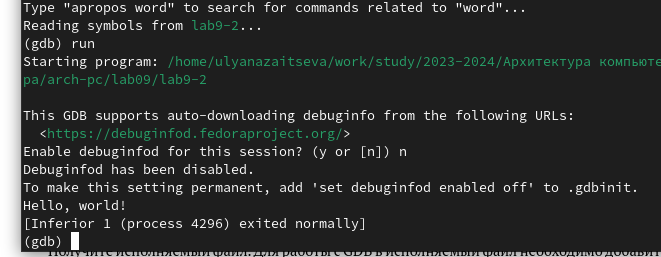
Листинг 9.2

Получаю исполняемый файл для работы с GDB с ключом ‘-g’.(рис. ??)



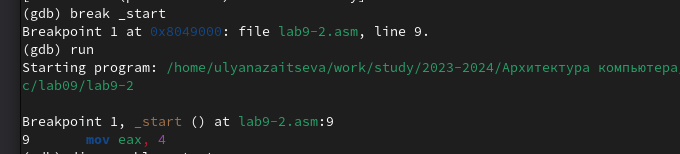
Создаю файл и загружаю в отладчик gdb

Проверяю работу программы, запустив ее в gdb с помощью команды run(рис. ??)



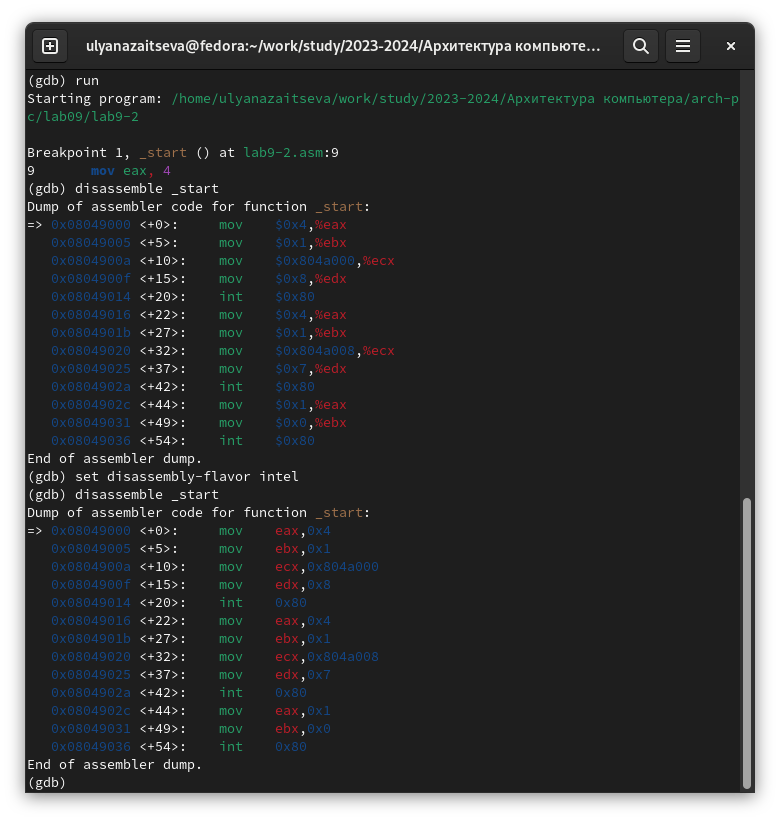
Проверка работы

Для более подробного анализа программы устанавливаю брейкпоинт на метку \_start и запускаю её(рис. ??)



Подробный анализ программы

Просматриваем дисассимилированный код программы с помощью команды disassemble, начиная с метки \_start, и переключаемся на отображение команд с синтаксисом Intel, введя команду set disassembly-flavor intel(рис. ??)

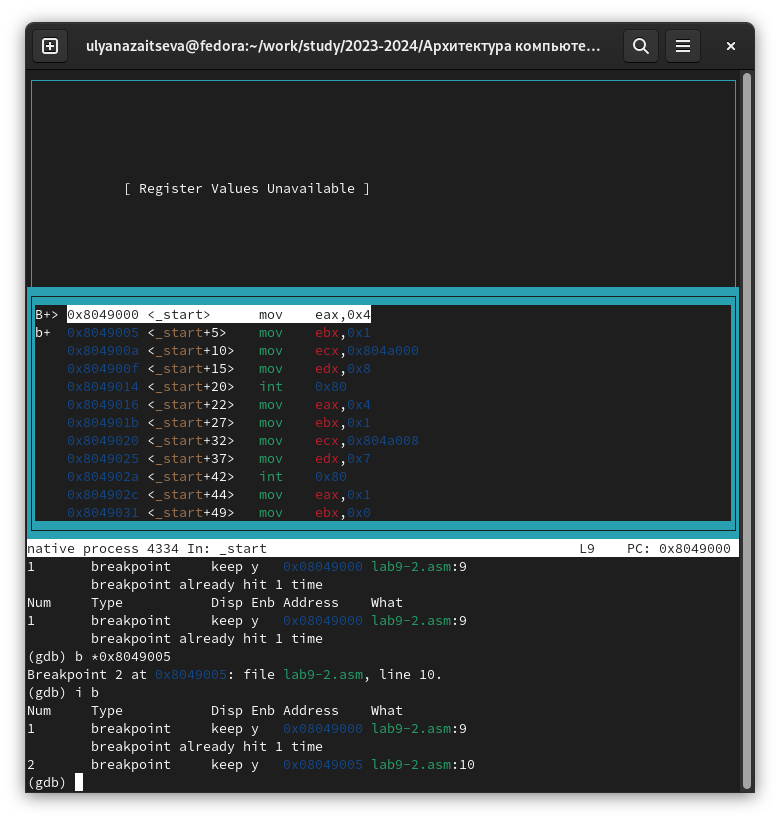


Дисассимилированный код программы

Включаю режим псевдографики для более удобного анализа программы с помощью команд layout asm и layout regs(забыла сделать скрин((()

1. Добавление точек останова.

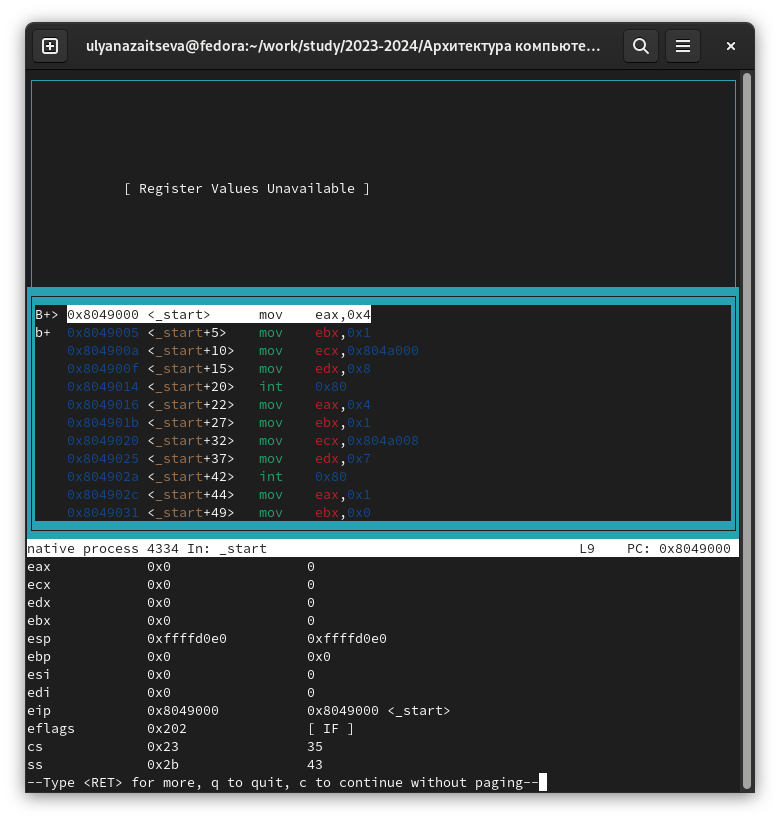
Проверяю, что точка останова по имени метки \_start установлена с помощью команды i b и устанавливаю еще одну точку останова. Просматриваю информацию о всех установленных точках останова.(рис. ??)



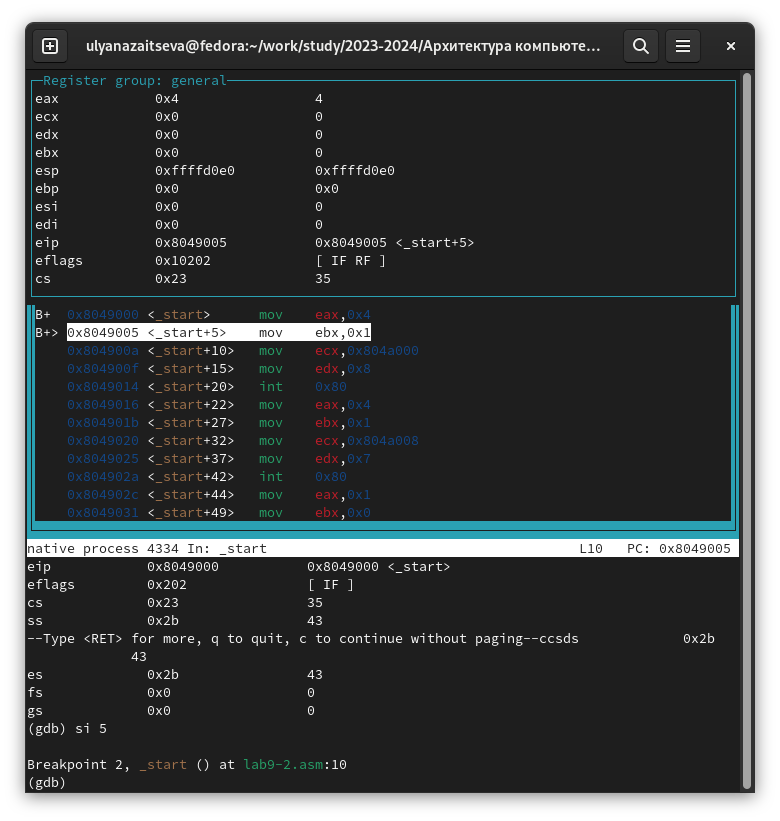
Новая точка останова

1. Работа с данными программы в GDB.

Выполняю инструкции с помощью команды stepi(si) и слежу за изменением значений регистров. (рис. ??)(рис. ??)



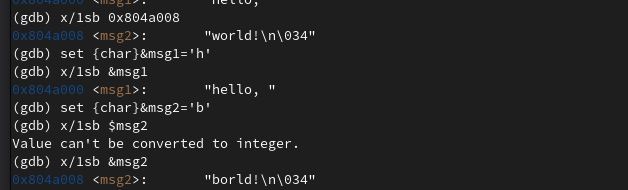
Команда i r



Инструкции si

Изменились значения регистров eax, ecx, edx и ebx.

Просматриваю значение переменной msg1 по имени с помощью команды x/1sb &msg1 и значение переменной msg2 по ее адресу. С помощью команды set изменяю первый символ переменной msg1 и заменяю первый символ в переменной msg2 на b.(рис. ??)



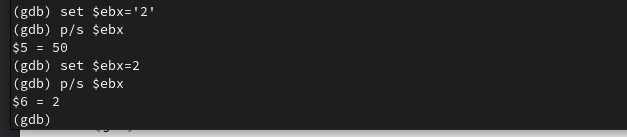
Переменные msg1 и msg2

Вывожу в шестнадцатеричном формате, в двоичном формате и в символьном виде соответственно значение регистра edx с помощью команды print p/F $val.(рис. ??)



Значение регистра edx

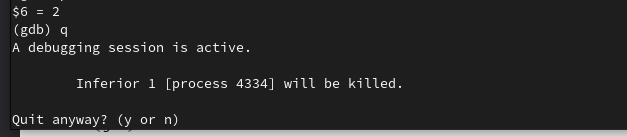
С помощью команды set изменяю значение регистра ebx (рис. ??)



Изменяю значение регистра ebx

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

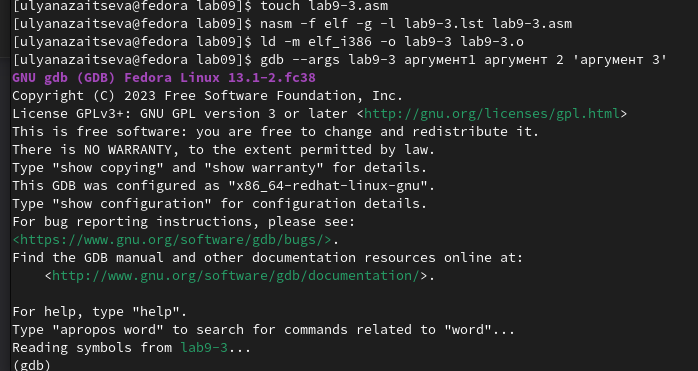
Выхожу из gdb с помощью команды quit. (рис. ??)



Выход из gdb

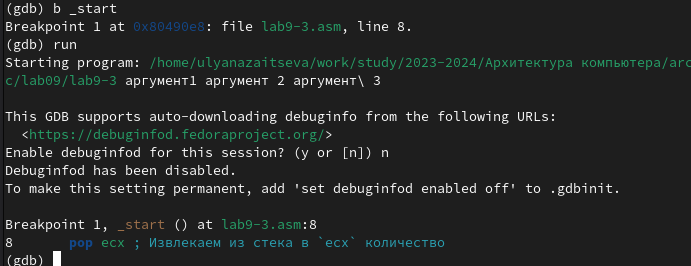
1. Обработка аргументов командной строки в GDB.

Копирую файл lab8-2.asm с программой из листинга 8.2 в файл lab9-3.asm и создаю исполняемый файл.Загружаю исполняемый файл в отладчик gdb, указывая необходимые аргументы с использованием ключа –args.(рис. ??)



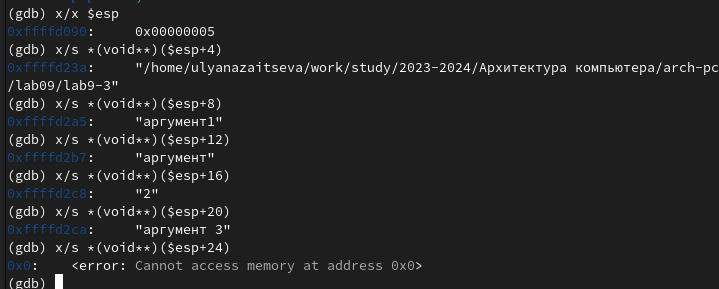
Создание файла и загрузка в gdb

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее.(рис. ??)



Установка точки останова и запуск

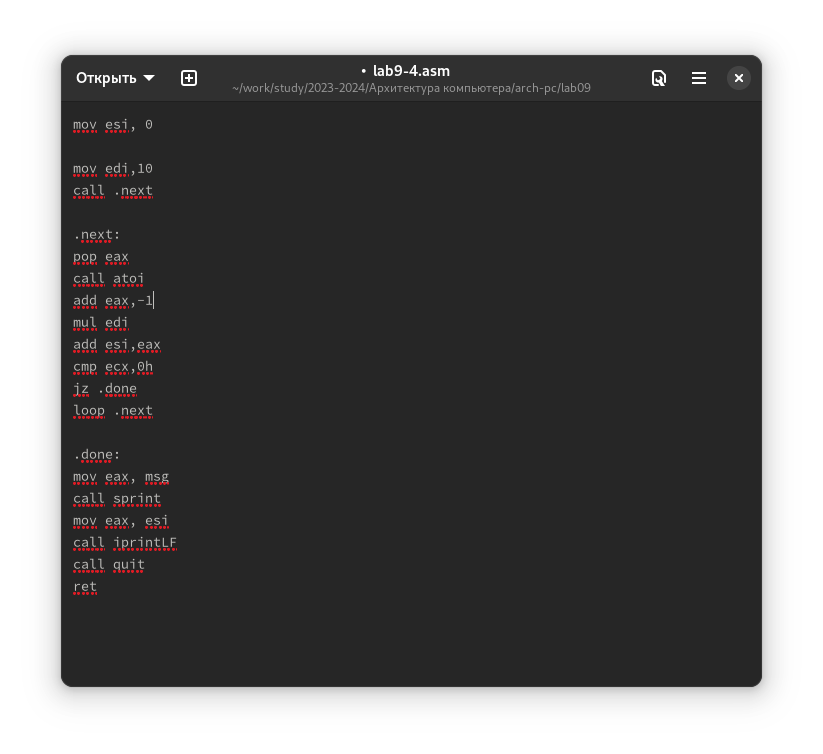
Посматриваю позиции стека по их адресам.(рис. ??)



Позиции стека

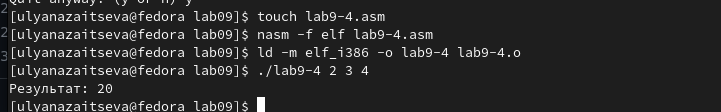
Шаг изменения адреса равен 4, т.к количество аргументов командной строки равно 4.

1. Задания для самостоятельной работы.
2. Преобразовываю программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму. в новом файле lab9-4.asm(рис. ??)



Преобразование программы из прошлой лабораторной работы

Проверяю, что она работает корректно.(рис. ??)



Проверка

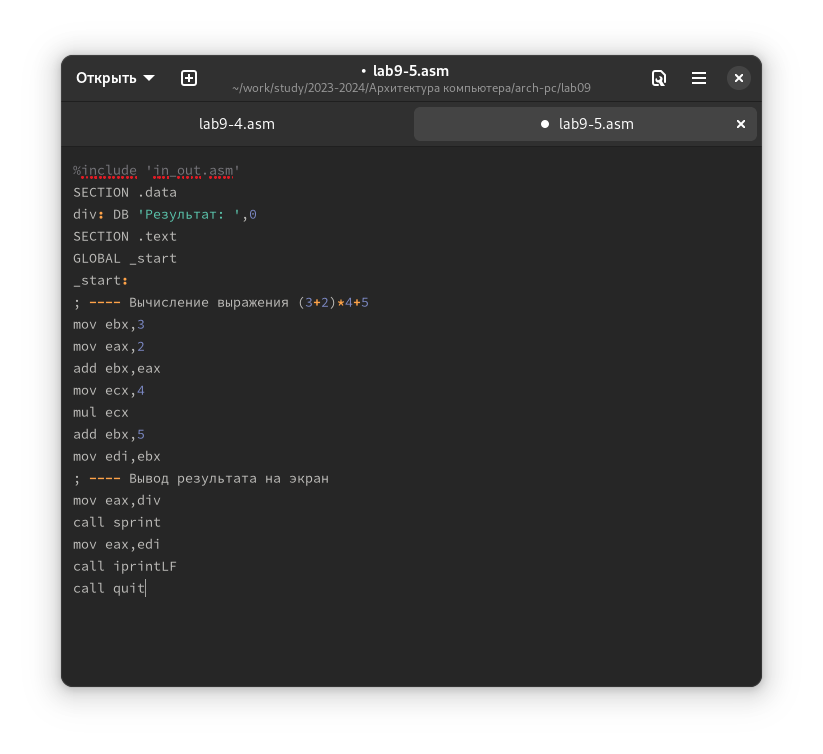
Код: ; 10(x-1) %include ‘in\_out.asm’ SECTION .data msg db “Результат:”,0 SECTION .text global \_start

\_start: pop ecx pop edx sub ecx,1 mov esi, 0 mov edi,10 call .next

.next: pop eax call atoi add eax,-1 mul edi add esi,eax cmp ecx,0h jz .done loop .next

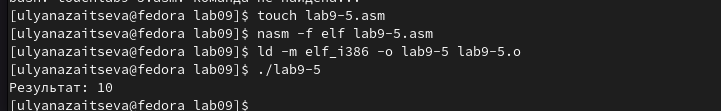
.done: mov eax, msg call sprint mov eax, esi call iprintLF call quit ret

1. Ввожу в файл lab9-5.asm текст программы из листинга 9.3.(рис. ??)



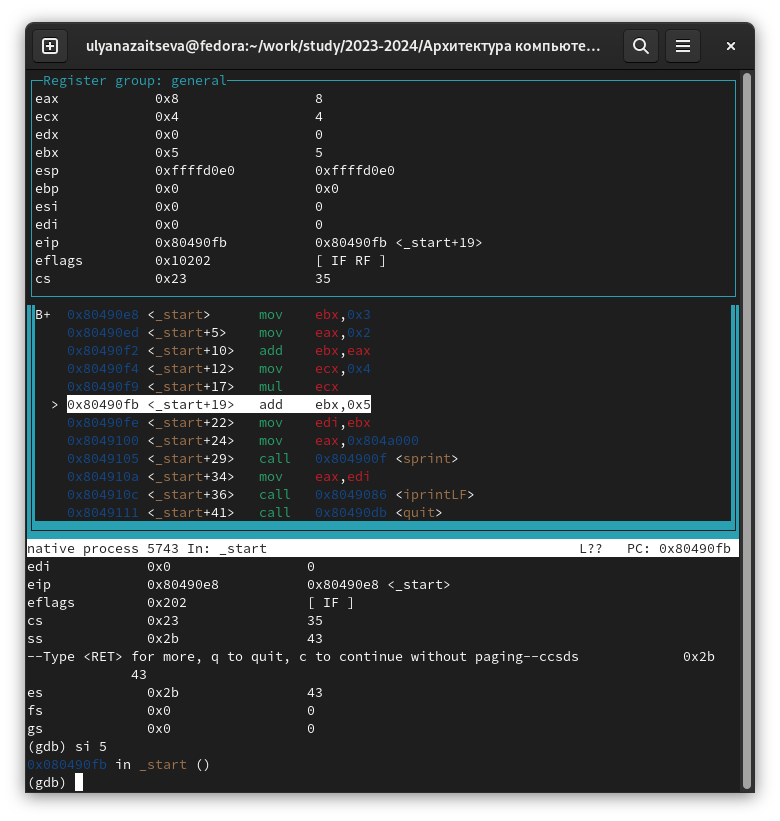
Листинг 9.3

При правильной работе программа должна выводить 25. Создаю исполняемый файл и запускаю его.(рис. ??)



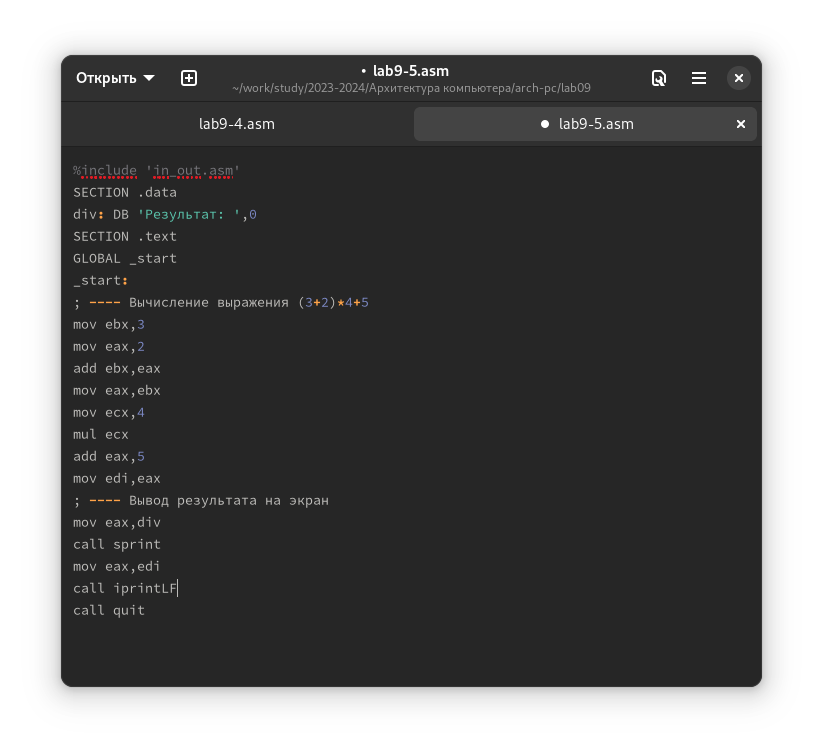
Выполнение программы

В выводе получаем неправильный ответ. Получаю исполняемый файл для работы с gdb, запускаю его и ставлю брейкпоинты для каждой инструкции, связанной с вычислениями. Прохожусь по каждому брейкпоинту и слежу за изменениями значений регистров. При выполнении инструкции mul ecx происходит умножение ecx(4) на eаx(2), вместо умножения 4 на 5 (ebx). Это происходит потому что стоящая перед mov ecx,4 инструкция add ebx,eax не связана с mul ecx, но связана инструкция mov eax,2.(рис. ??)



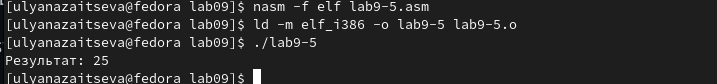
Поиск сбоя

Исправляю ошибку, добавляя после add ebx,eax mov eax,ebx и заменяя ebx на eax в инструкциях add ebx,5 и mov edi,ebx. (рис. ??)



Исправляю ошибку

Запускаю код и проверяю, что программа работает корректно.(рис. ??)



Исправленный код

Код:

%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 ebx,eax mov 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

# 5 Выводы

Во время выполнения данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм и познакомилась с методами отладки при помощи GDB и его основными возможностями.