Отчёт по лабораторной работе №8

Дисциплина: Архитектура компьютера

Осина Виктория Александровна

Содержание

# 1 Цель работы

Целью данной лабораторной работы является 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. Обработка аргументов командной строки в GDB
4. Выполнение задания для самостоятельной работы

# 3 Теоретическое введение

**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).

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.

**Методы отладки**

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).

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.

**Запуск отладчика GDB**

Команда run (сокращённо r) — запускает отлаживаемую программу в оболочке GDB. Если точки останова не были установлены, то программа выполняется и выводятся сооб- щения:

(gdb) run  
Starting program: test  
Program exited normally.  
(gdb)

Для выхода из отладчика используется команда quit (или сокращённо q): (gdb) q

**Дизассемблирование программы**

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

**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»: (gdb) break \*<адрес> (gdb) b <метка> Информацию о всех установленных точках останова можно вывести командой info (крат- ко i): (gdb) info breakpoints (gdb) i b

Если же точка останова в дальнейшем больше не нужна, она может быть удалена с помощью команды delete: (gdb) delete breakpoint <номер точки останова>

**Пошаговая отладка**

Команда stepi (кратко sI) позволяет выполнять программу по шагам, т.е. данная команда выполняет ровно одну инструкцию (gdb) si [аргумент]

При указании в качестве аргумента целого числа 𝑁 отладчик выполнит команду step 𝑁 раз при условии, что не будет точек останова или выполнение программы не прервётся по другим причинам.

Команда nexti (или ni) аналогична stepi, но вызов процедуры (функции) трактуется отладчиком как одна инструкция: (gdb) ni [аргумент]

# 4 Выполнение лабораторной работы

## 4.1 Реализация подпрограмм в NASM

Создаю каталог lab09 для программ лабораторной работы №9, перехожу в него и создаю файл lab9-1.asm, проверяю, что файл создан (рис. [1](#fig:001))

Figure 1: Создание каталога lab09 и файла lab9-1.asm в нем

Figure 1: Создание каталога lab09 и файла lab9-1.asm в нем

Перед работой с программами копирую файл in\_out.asm в каталог и проверяю, что файл находится в нужном каталоге (рис. [2](#fig:002)) (рис. [3](#fig:003))

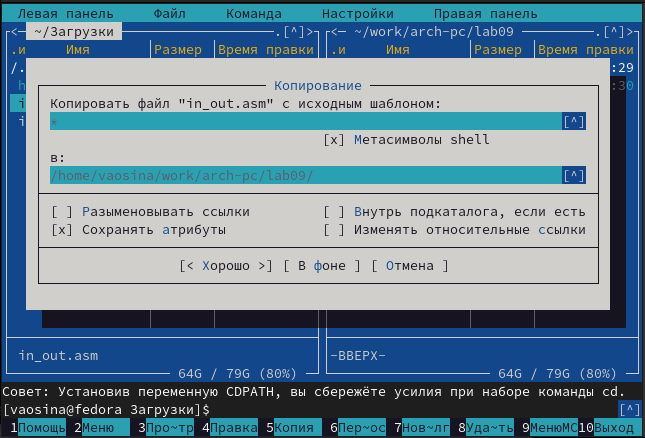


Figure 2: Копирование файла in\_out.asm

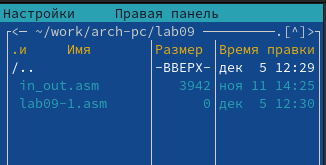


Figure 3: Проверка, что файл находится в нужном каталоге

Открываю lab9-1.asm в редакторе и ввожу в него текст программы вычисления арифметического выражения f(x) = 2x + 7 (рис. [4](#fig:004)) и (рис. [5](#fig:005)).

Figure 4: Открытие файла в редакторе

Figure 4: Открытие файла в редакторе

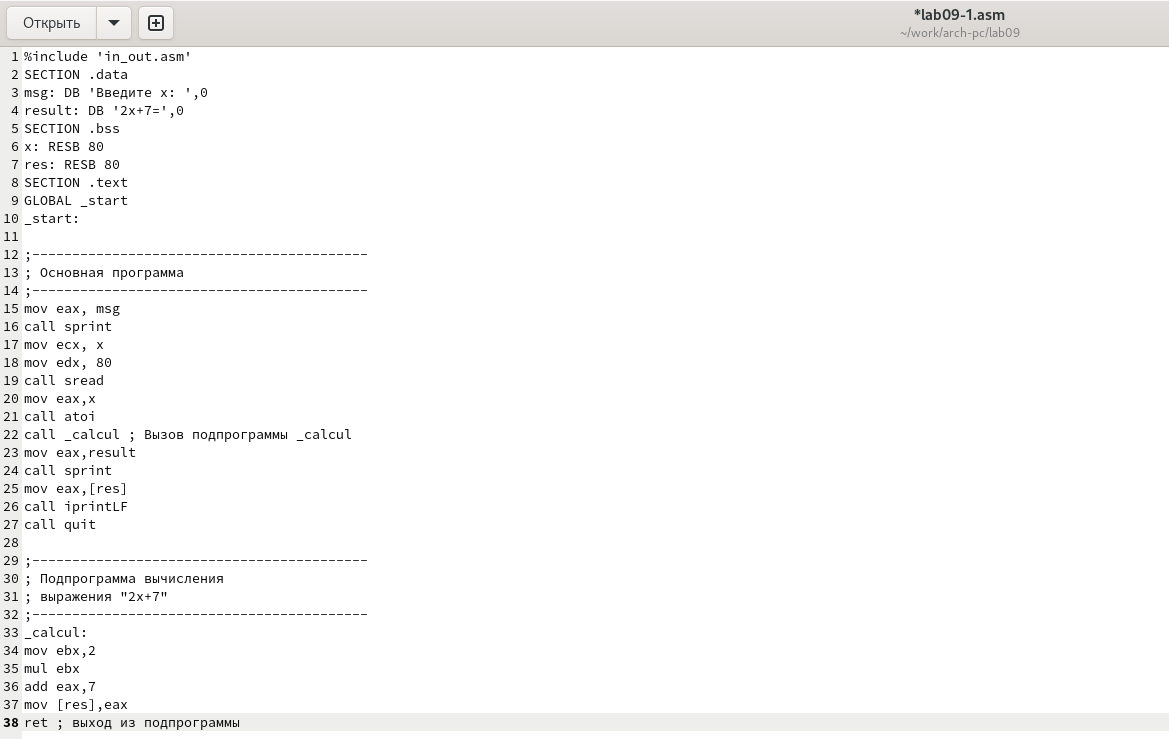


Figure 5: Ввод текста программы в файл

Создаю исполняемый файл и запускаю его (рис. [6](#fig:006)). Программа работает корректно.

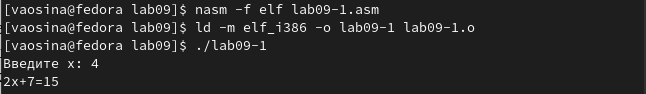


Figure 6: Создание исполняемого файла и его запуск

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. [7](#fig:007))

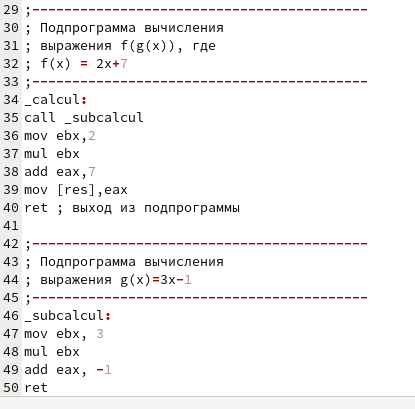


Figure 7: Изменение текста программы

Создаю исполняемый файл и запускаю его. (рис. [8](#fig:008)). Проверяю работу программы на нескольких значениях х. Программа работает корректно.

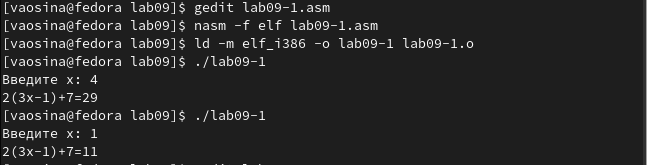


Figure 8: Создание исполняемого файла и его запуск

## 4.2 Отладка программам с помощью GDB

Создаю файл lab9-2.asm в каталоге ~/work/arch-pc/lab09 и открываю его в редакторе (рис. [9](#fig:009)).

Figure 9: Создание файла lab9-2.asm и его открытие в редакторе

Figure 9: Создание файла lab9-2.asm и его открытие в редакторе

Ввожу в него текст программы печати сообщения Hello world!. (рис. [10](#fig:010))



Figure 10: Ввод текста программы

Получаю исполняемый файл с добавлением отладочной информации, для этого трансляцию программы провожу с ключом -g. (рис. [11](#fig:011)).

Figure 11: Создание исполняемого файла и его запуск

Figure 11: Создание исполняемого файла и его запуск

Загружаю исполняемый файл в отладчик gdb. (рис. [12](#fig:012)).

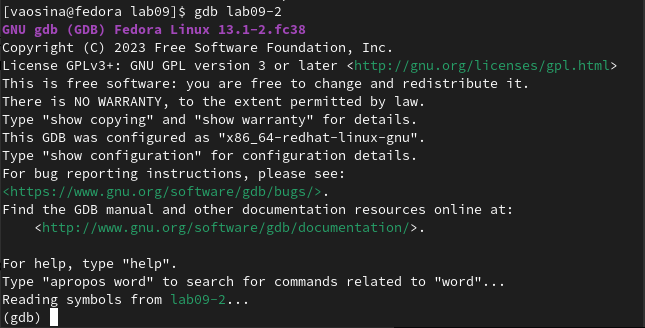


Figure 12: Загрузка исполняемого файла в отладчик gdb

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run.(рис. [13](#fig:013)).

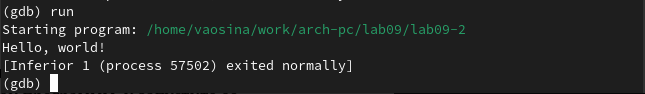


Figure 13: Проверка работы программы

Для более подробного анализа программы устанавливаю брейкпоинт на метку \_start и запускаю программу. (рис. [14](#fig:014)).

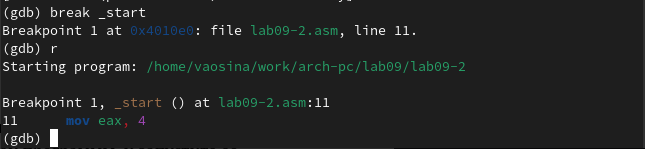


Figure 14: Установка брейкпоинта

Смотрю дисассимилированный код программы с помощью команды disassemble начиная с метки \_start. (рис. [15](#fig:015))

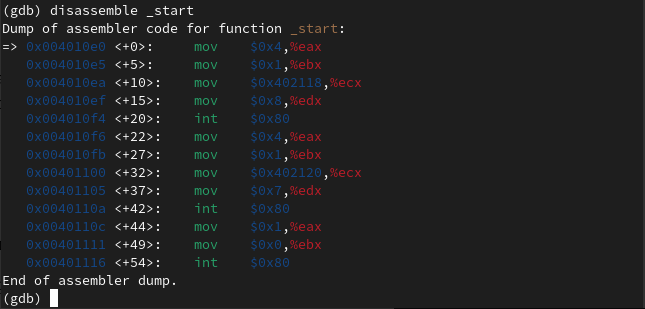


Figure 15: Просмотр дисассимилированного кода программы

Переключаюсь на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel, и смотрю дисассимилированный код. (рис. [16](#fig:016)) и (рис. [17](#fig:017))

Различия отображения синтаксиса машинных команд в режимах ATT и Intel состоят в том, что в режиме АТТ перед названиями регистров стоит $, а перед операндами %, а еще после переключения на режим Intel, регистры и операнды поменялись местами.

Figure 16: Переключение на режим Intel

Figure 16: Переключение на режим Intel



Figure 17: Просмотр дисассимилированного кода программы

Включаю режим псевдографики для более удобного анализа программы. (рис. [18](#fig:018))

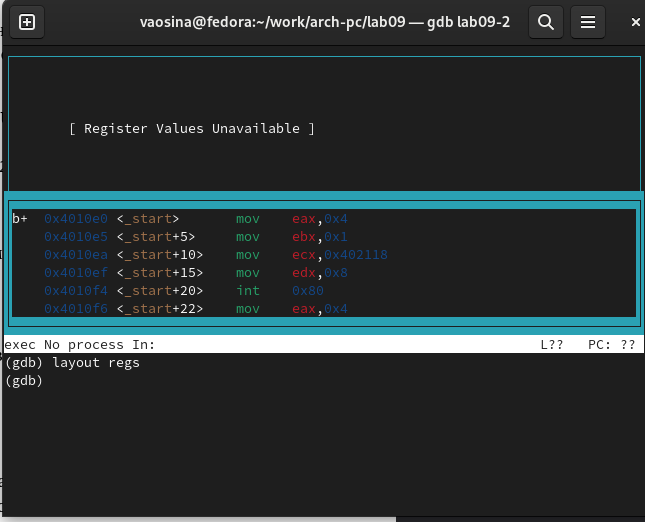


Figure 18: Включение режима псевдографики

На предыдущих шагах была установлена точка останова по имени метки (\_start). Проверяю это с помощью команды info breakpoints (кратко i b). (рис. [19](#fig:019))

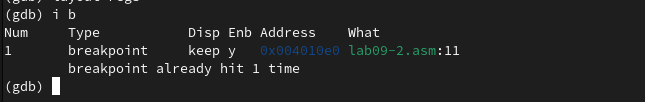


Figure 19: Проверка наличия брейкпоинта

Устанавливаю еще одну точку останова по адресу инструкции mov ebx,0x0 и Смотрю информацию о всех установленных точках останова. (рис. [20](#fig:020))

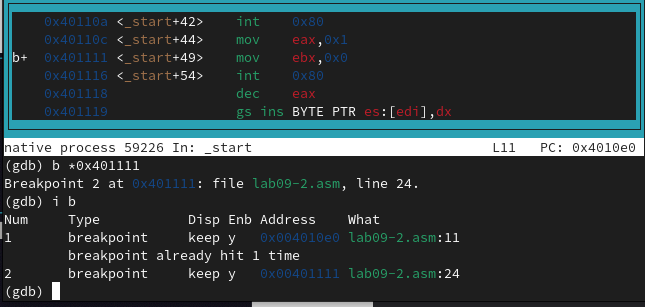


Figure 20: Установка еще одной точки установа

Смотрю содержимое регистров с помощью команды info registers (или i r). (рис. [21](#fig:021))

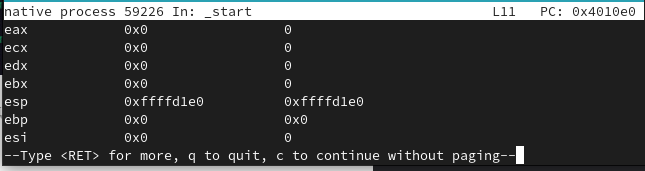


Figure 21: Просмотр содержимого регистров

Выполняю 5 инструкций с помощью команды stepi (или si). (рис. [22](#fig:022)) и (рис. [23](#fig:023))

Изменились значения регистров eax, ecx, edx, ebx.

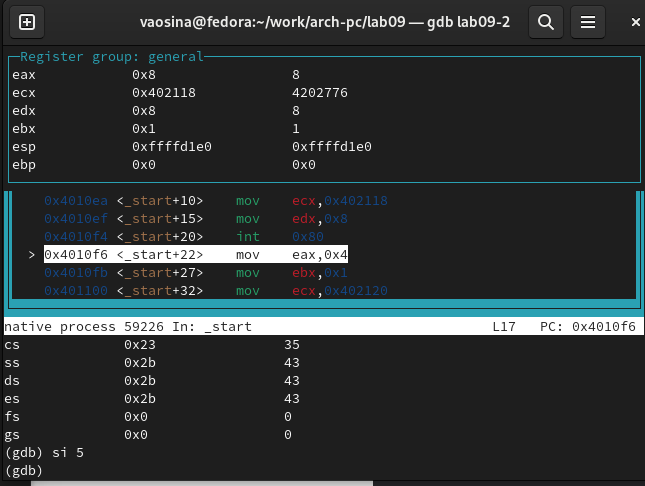


Figure 22: Выполнение 5 инструкций stepi

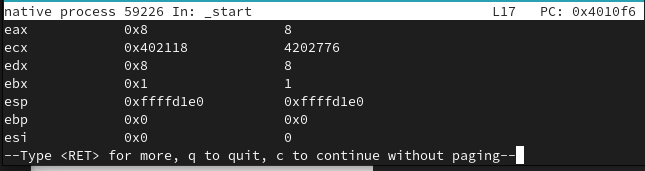


Figure 23: Просмотр содержимого регистров

Смотрю значение переменной msg1 по имени. (рис. [24](#fig:024))

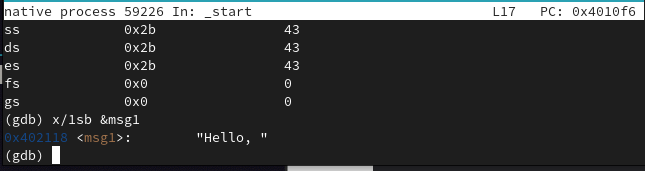


Figure 24: Просмотр значения переменной msg1

Смотрю значение переменной msg2 по адресу, который определяю по дизассемблированной инструкции. (на скрине я сначала случайно ввела не тот адрес)(рис. [25](#fig:025))

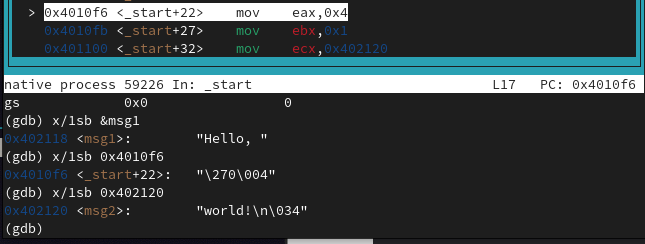


Figure 25: Просмотр значения переменной msg2

Изменяю первый символ переменной msg1. (рис. [26](#fig:026))

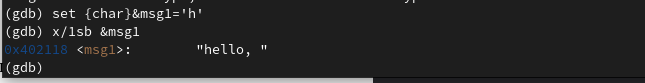


Figure 26: Изменение символа

Изменяю символ переменной msg2. (рис. [27](#fig:027))

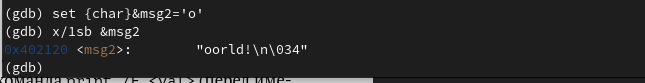


Figure 27: Изменение символа

Вывожу в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx. (рис. [28](#fig:050))



Figure 28: Ввыод в различных форматах

С помощью команды set изменяю значение регистра ebx. (рис. [29](#fig:028))

Разница вывода состоит в том, что в первом случае символ переводится в строковый вид.

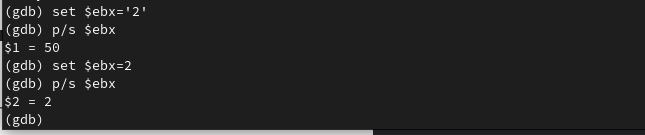


Figure 29: Изменение значения регистра

Завершаю выполнение программы с помощью команды continue (сокращенно c) и выхожу из GDB с помощью команды quit (сокращенно q). (рис. [30](#fig:029)) и (рис. [31](#fig:030))

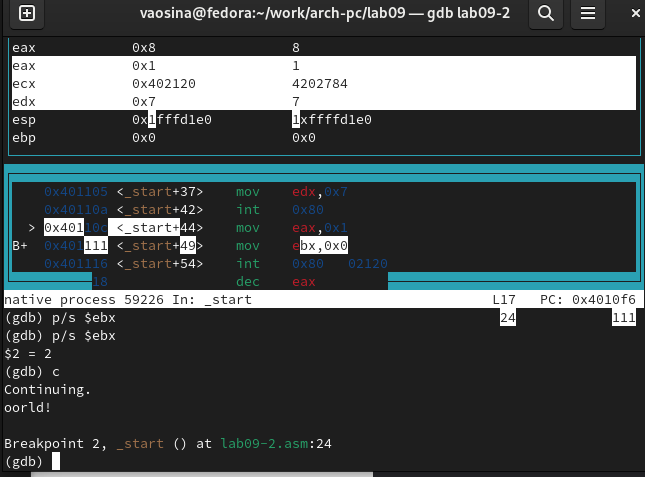


Figure 30: Завершение программы

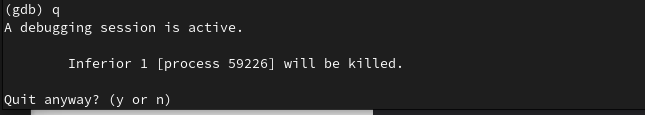


Figure 31: Выход из gdb

## 4.3 Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки в файл с именем lab09-3.asm. (рис. [32](#fig:032))

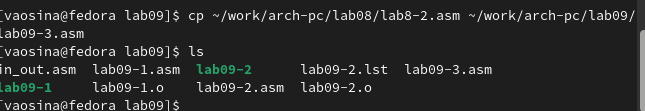


Figure 32: Копирование файла

Создаю исполняемый файл (рис. [33](#fig:033))

Figure 33: Создание исполняемого файла

Figure 33: Создание исполняемого файла

Загружаю исполняемый файл в отладчик, указав аргументы. (рис. [34](#fig:034))

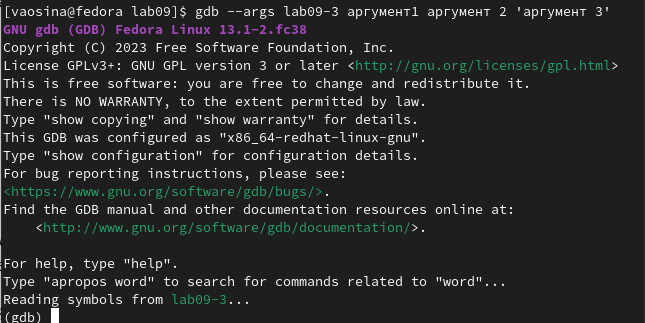


Figure 34: Загрузка исполняемого файла в отладчик

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее. (рис. [35](#fig:035))

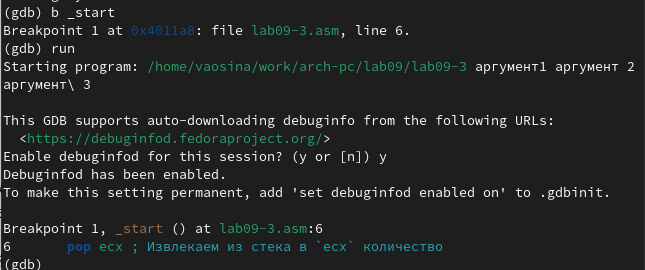


Figure 35: Установка точки останова

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы).(рис. [36](#fig:036))

Как видно, число аргументов равно 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’.

Figure 36: Просмотр содержимого регистра

Figure 36: Просмотр содержимого регистра

Смотрю 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. [37](#fig:037))

Шаг изменения адреса равен 4, т.к. 4 - это отводимый размер памяти на ячейку и мы смотрим содержимое ячеек.

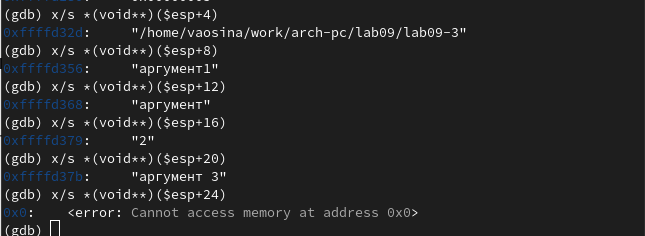


Figure 37: Просмотр остальных позиций стека

## 4.4 Выполнение задания для самостоятельной работы

Преобразовываю программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму. (рис. [38](#fig:038))

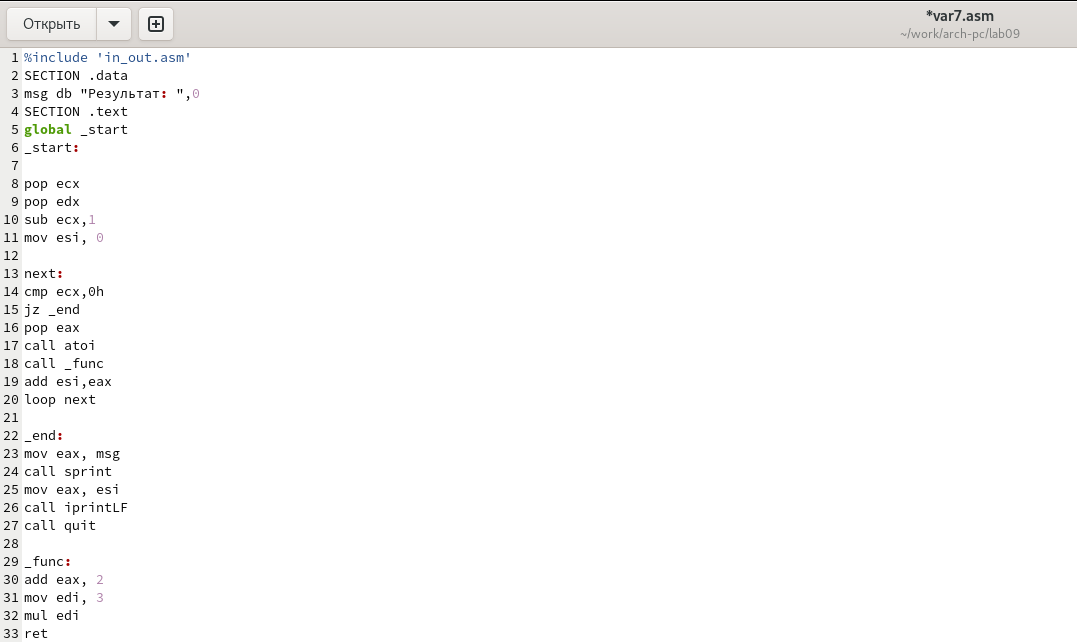


Figure 38: Преобразование программы

Создаю исполняемый файл и запускаю его. (рис. [39](#fig:039)) Программа работает корректно.

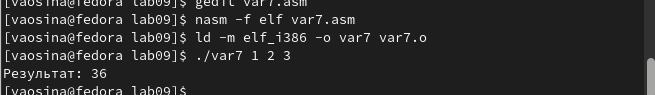


Figure 39: Создание исполняемого файла и его запуск

Текст программы:

%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 \_func  
add esi,eax  
loop next  
  
\_end:  
mov eax, msg  
call sprint  
mov eax, esi  
call iprintLF  
call quit  
  
\_func:  
add eax, 2  
mov edi, 3  
mul edi  
ret

Создаю файл var7-2.asm и открываю его в редакторе. (рис. [40](#fig:040))

Figure 40: Создание файла var7-2.asm и открытие его в редакторе

Figure 40: Создание файла var7-2.asm и открытие его в редакторе

Ввожу текст программы вычисления выражения (3+2)\*4+5 из листинга 9.3. (рис. [41](#fig:041))

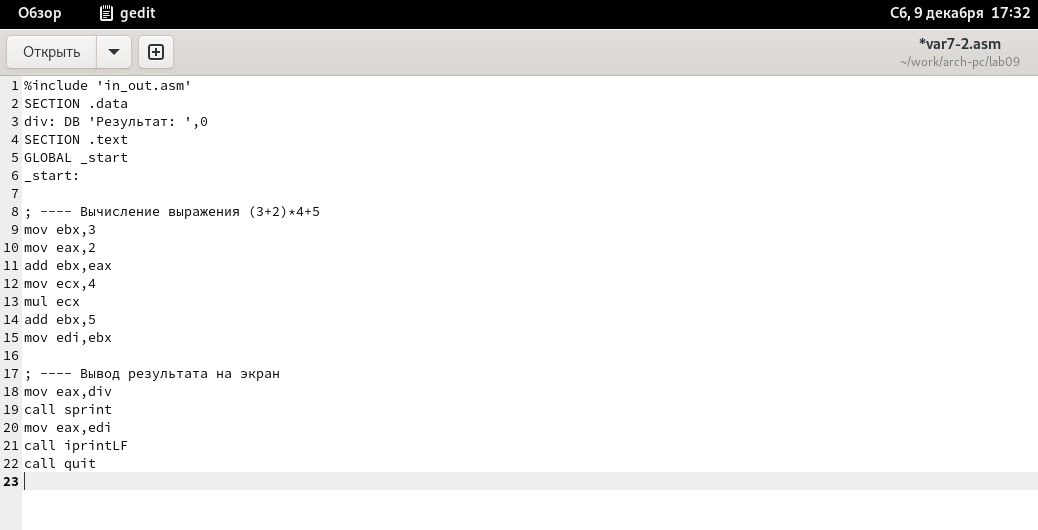


Figure 41: Ввод текста программы

Создаю исполняемый файл и запускаю его. (рис. [42](#fig:042)) Программа действительно работает некорректно.

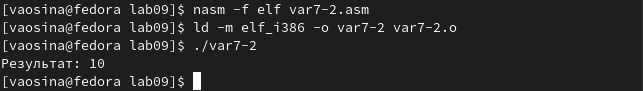


Figure 42: Создание исполняемого файла и его запуск

Получаю исполняемый файл с добавлением отладочной информации, для этого трансляцию программы провожу с ключом -g и загружаю исполняемый файл в отладчик gdb. (рис. [43](#fig:043))

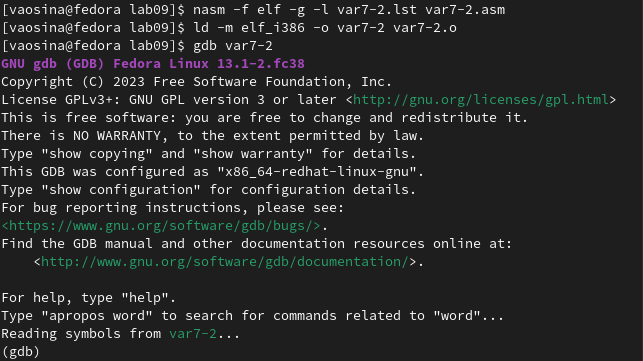


Figure 43: Получение исполняемого файла

Запускаю программу и устанавливаю точку останова на метке \_start, чтобы было удобнее анализировать изменения регистров. (рис. [44](#fig:044))

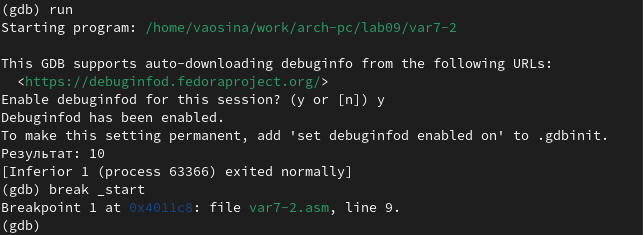


Figure 44: Запуск программы и установка брейкпоинта

Первую ошибку я замечаю при изменении регистра eax на 8, такого быть не должно. Мы должны были получить при умножении 20. (рис. [45](#fig:045))

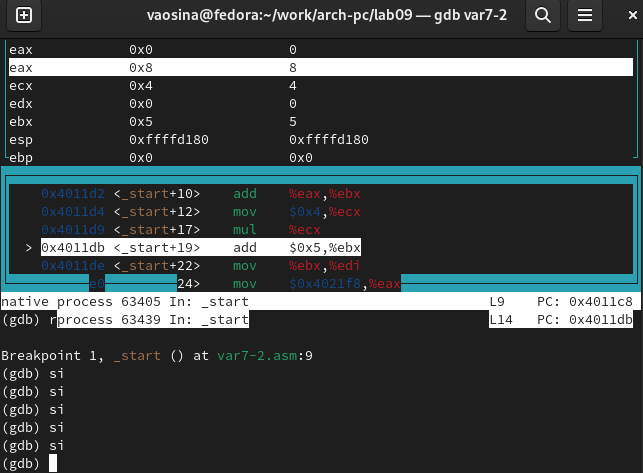


Figure 45: Просмотр изменения регистра

Далее ошибка при последнем действии, т.к. 5 складывается со значением ebx, в котором тоже 5, в итоге получаем 10, в результате и выводится это значение. (рис. [46](#fig:046))

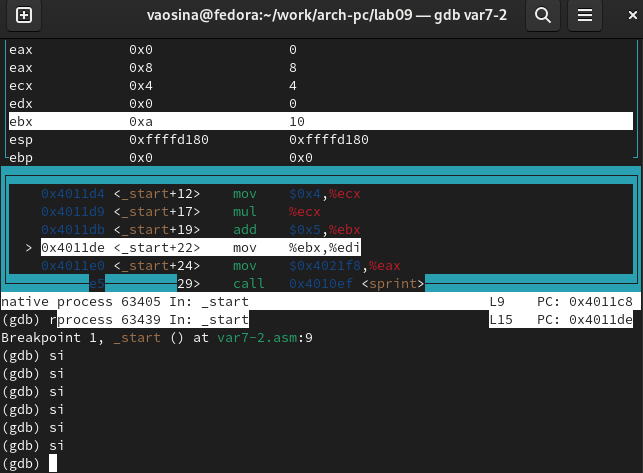


Figure 46: Просмотр изменения регистра

Меняю текст программы таким образом, чтобы она работала корректно. (рис. [47](#fig:047))

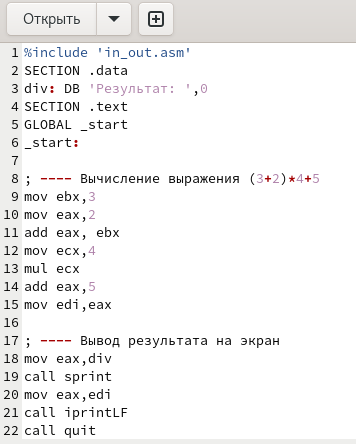


Figure 47: Изменение текста программы

Создаю исполняемый файл и запускаю его. (рис. [48](#fig:048))

Теперь программа работает корректно.

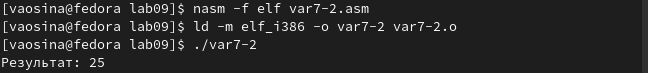


Figure 48: Создание исполняемого файла и его запуск

Текст программы:

%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

# 5 Выводы

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

# 6 Список литературы

1. [Архитектура ЭВМ](https://esystem.rudn.ru/pluginfile.php/2089096/mod_resource/content/0/%D0%9B%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%20%E2%84%969.%20%D0%9F%D0%BE%D0%BD%D1%8F%D1%82%D0%B8%D0%B5%20%D0%BF%D0%BE%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D1%8B.%20%D0%9E%D1%82%D0%BB%D0%B0%D0%B4%D1%87%D0%B8%D0%BA%20..pdf)