Отчет по лабораторной работе No 9

Дисциплины: Архитектура компьютера

Pакутуманандзара Цантамписедрана Сарубиди

Содержание

Список иллюстраций

Список таблиц

# 1 Цель работы

Цель лабораторной работы – приобретение навыков написания программ с использованием подпрограмм. Введение в методы отладки с использованием GDB и его основные возможности.

Цель данного ш

# 2 Задание

1. Реализация подпрограмм в NASM
2. Отладка программам с помощью GDB
3. Задание для самостоятельной работы

# 3 Выполнение лабораторной работы

1. Реализация подпрограмм в NASM

Я создам каталог для lab09, зайду в него и создам файл lab09-1.asm(рис 1)

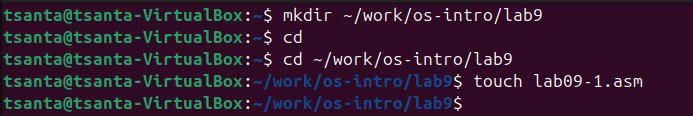


Рис. 1: рис 1

Рассмотрим программу вычисления арифметического выражения f(x) = 2x + 7 с помощью подпрограммы \_calcul. В этом примере x вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Я скопирую текст программы ниже и скопирую его в созданный мной файл(рис 2)

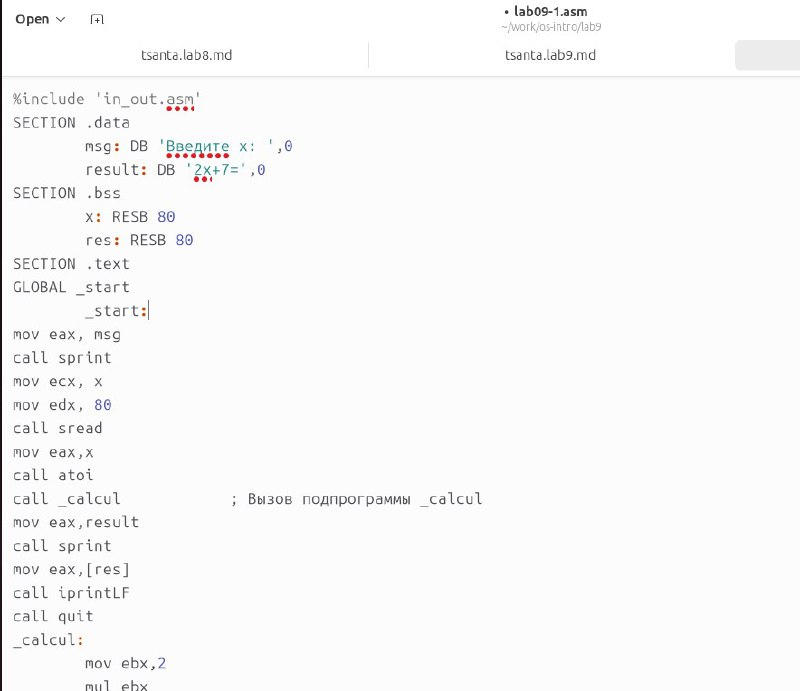


Рис. 2: рис 2

Я создам исполняемый файл и проверю его работу(рис 3)



Рис. 3: рис 3

Я отредактирую программу так, чтобы она решала функцию f(g(x)), где f(x)=2x+7 и g(x)=3x-1(рис 4)

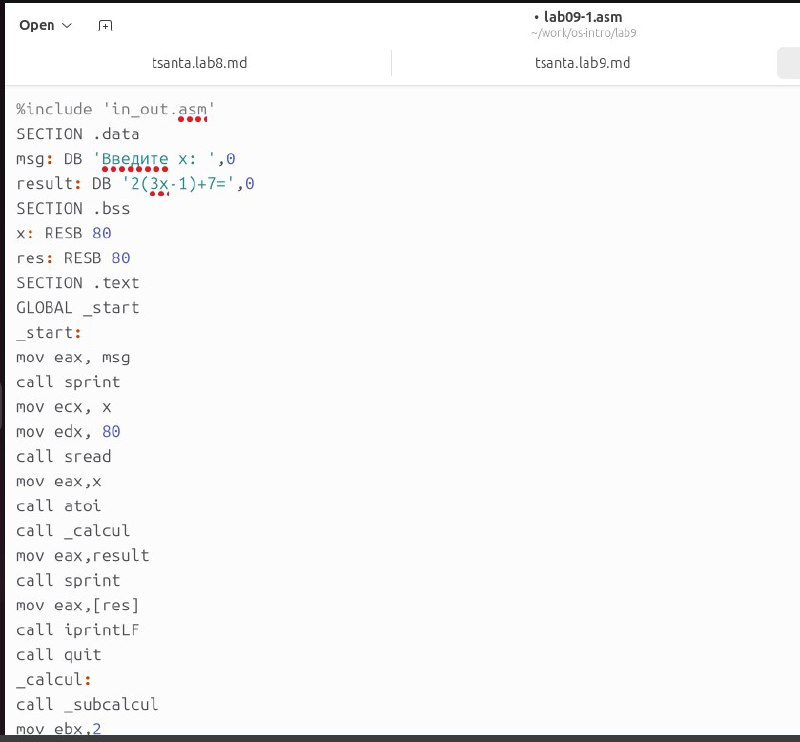


Рис. 4: рис 4

Я создам исполняемый файл и проверю его работу(рис 5)

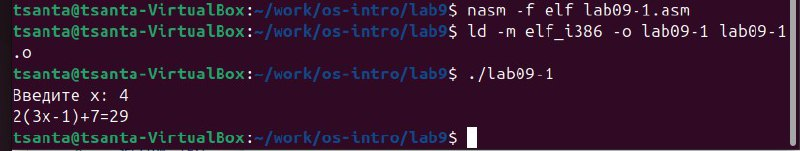


Рис. 5: рис 5

1. Отладка программам с помощью GDB

Я создам новый файл lab09-2asm и скопирую в него данную программу(рис 6)

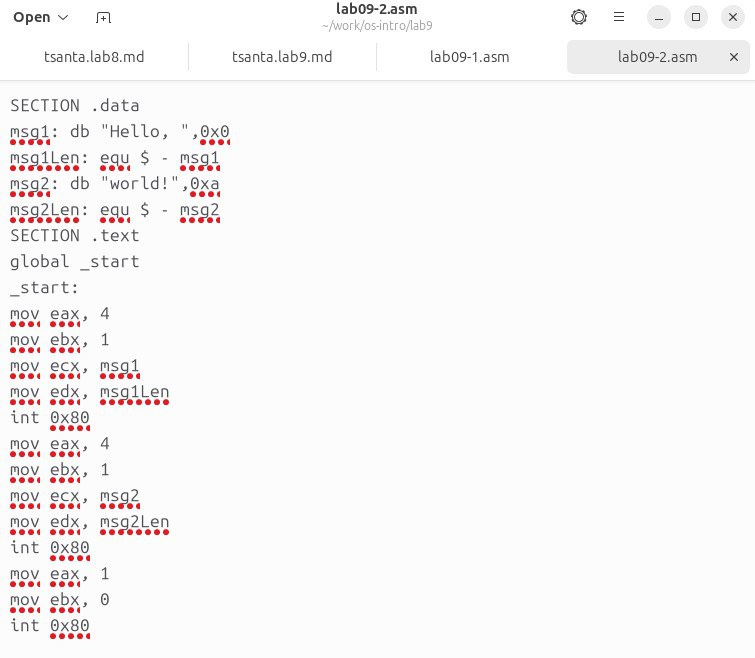


Рис. 6: рис 6

Я создам исполняемый файл и запущу его с помощью отладчика GDB. Чтобы работать с GDB, мне нужно добавить в исполняемый файл отладочную информацию; для этого программы необходимо переводить с ключом «-g»(рис 7)

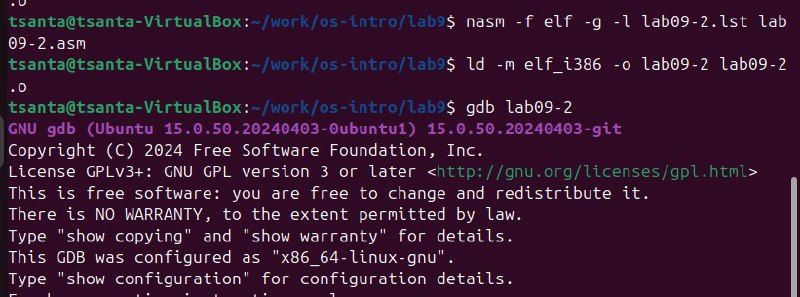
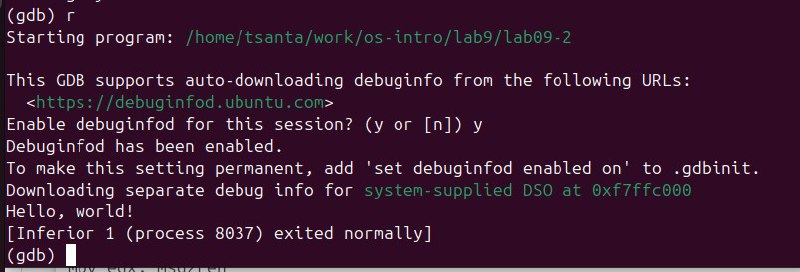


Рис. 7: рис 7

Я протестирую программу, запустив ее в оболочке GDB с помощью команды запуска(рис 8)

Р

Для более детального анализа программы я поставлю точку останова на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запущу ее(рис 9)



Рис. 8: рис 9

Я буду просматривать дизассемблированный код программы с помощью команды дизассемблирования, начиная с метки \_start(рис 10)

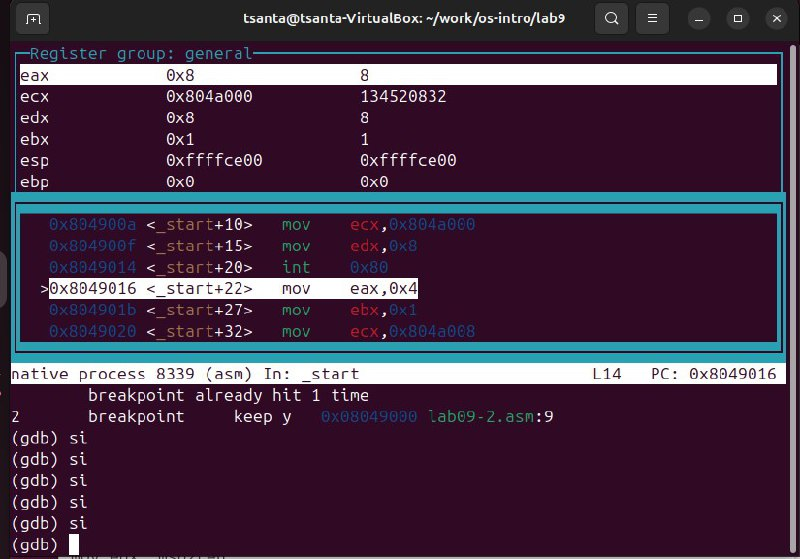


Рис. 9: рис 10

Я переключусь на отображение команд с синтаксисом Intel, введя команду set disassembly-flavor intel(рис 11)



Рис. 10: рис 11

Для более удобного анализа программы включу режим псевдографики(рис 12)

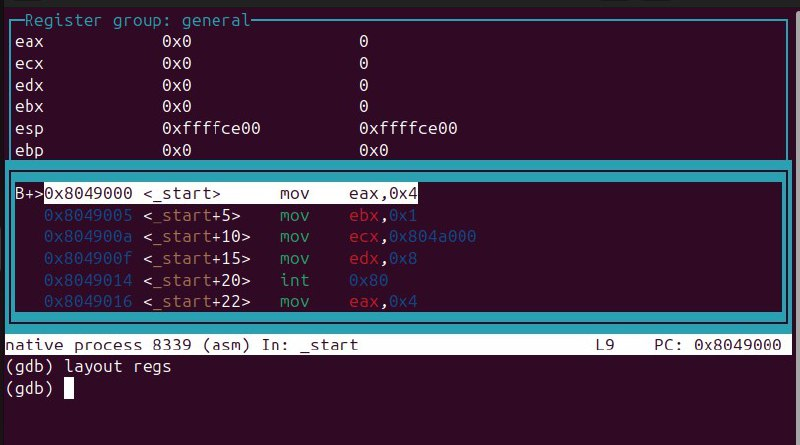


Рис. 11: рис 12

В Intel все начинается с адреса, затем с источника, а в ATT наоборот

2.1. Добавление точек останова

На предыдущих шагах точка останова была установлена ​​по имени метки (\_start). Я проверю это с помощью команды info Breakpoints (сокращенно i b)(рис 13)

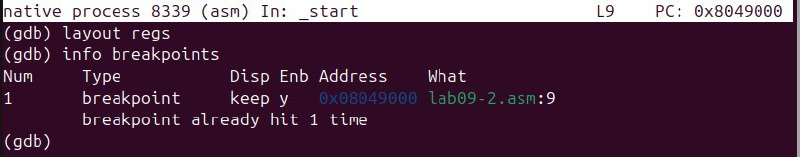


Рис. 12: рис 13

Я поставлю еще одну точку останова по адресу инструкции(рис 14)

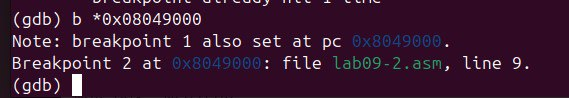


Рис. 13: рис 14

Теперь я посмотрю информацию обо всех установленных точках останова.

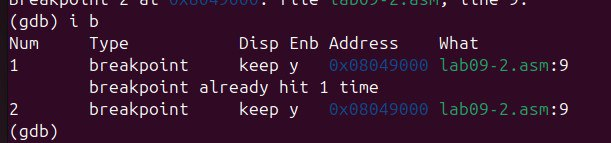


Рис. 14: рис 15

2.2. Работа с данными программы в GDB

Я выполню 5 инструкций с помощью команды Stepi (или Si)(рис 16)

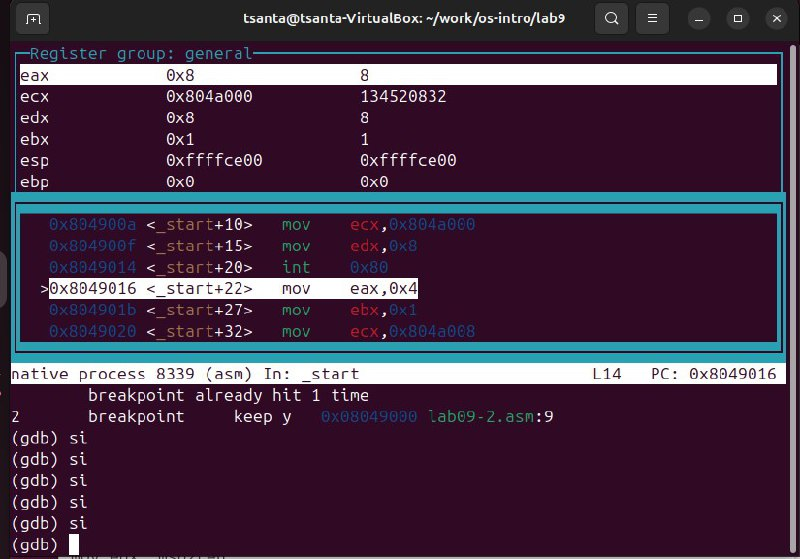


Рис. 15: рис 16

Значения eax,ecx,esp и edx изменились

Содержимое регистров также можно просмотреть с помощью команды info Registers(рис 17)

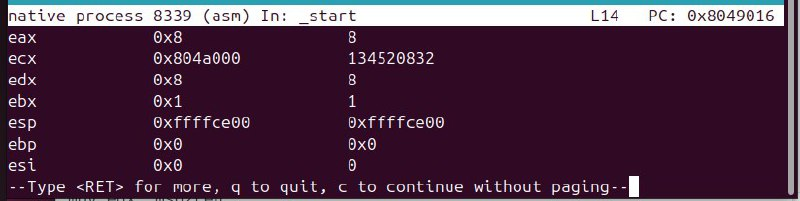


Рис. 16: рис 17

С помощью команды x & можно посмотреть содержимое переменной.Я поищу значение переменной msg1 по имени(рис 18)

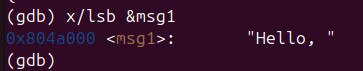


Рис. 17: рис 18

Теперь я посмотрю на значение переменной msg2 по адресу. Адрес переменной можно определить из дизассемблированной инструкции(рис 19)

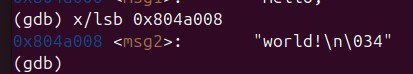


Рис. 18: рис 19

Вы можете изменить значение регистра или ячейки памяти с помощью команды set, передав ей имя или адрес регистра в качестве аргумента. Я изменю первый символ переменной msg1(рис 20)



Рис. 19: рис 20

Теперь я заменю символ во второй переменной msg2(рис 21)

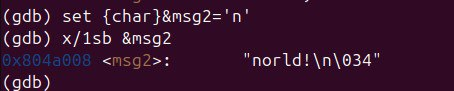


Рис. 20: рис 21

Я буду использовать команду set для изменения значения регистра ebx(рис 22)



Рис. 21: рис 22

2.3. Обработка аргументов командной строки в GDB

Я скопирую файл lab8-2.asm, созданный во время лабораторной работы 8, с помощью программы, которая печатает аргументы командной строки, в файл с именем lab09-3.asm(рис 23)

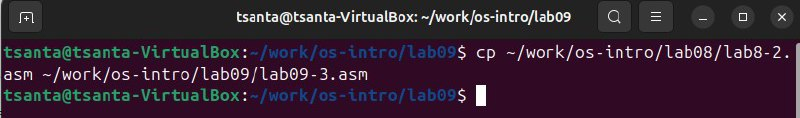


Рис. 22: рис 23

Я создам исполняемый файл и загрузим исполняемый файл в отладчик с аргументами, для загрузки программ с аргументами в gdb я буду использовать ключ –args(рис 24)



Рис. 23: рис 24

Сначала я установлю точку останова перед первой инструкцией в программе и запущу ее(рис 25)

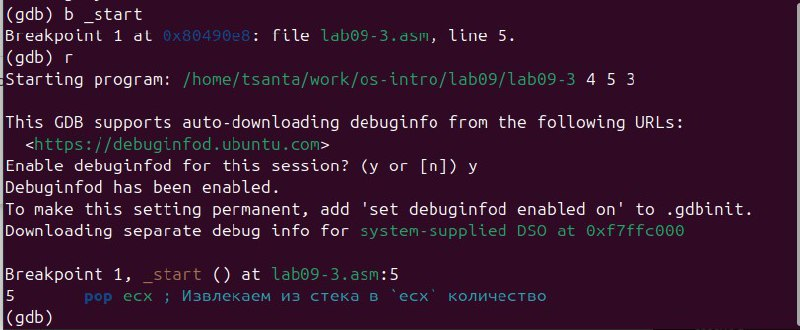


Рис. 24: рис 25

Адрес вершины стека хранится в регистре esp и по этому адресу можно увидеть число, равное количеству аргументов командной строки (включая имя программы)(рис 26)



Рис. 25: рис 26

Как видите, количество аргументов равно 4 — это название программы lab09-3 и сами аргументы: аргумент1, аргумент 2 и ‘аргумент 3’

Я взгляну на оставшиеся позиции стека — адрес [esp+4] хранит адрес в памяти, где находится имя программы, адрес [esp+8] хранит адрес первого аргумента, адрес [esp+12] ] сохраняет второй аргумент и т. д(рис 27)

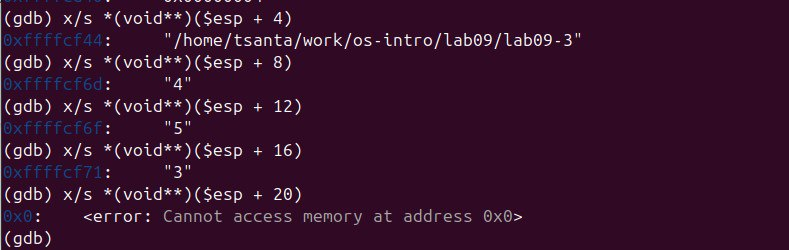


Рис. 26: рис 27

В 32-битных компьютерах информация хранится именно так: первая память выделяется 4 бита, а вторая — 4х2

1. Задание для самостоятельной работы

* Я создам новый файл с именем lab09-4.asm(рис 28)

Рис. 27: рис 28

Рис. 27: рис 28

Используя функцию (f(x) = 15x-9) которая была у меня при выполнении lab08- задание 1, я напишу программу, которая вычисляет значение функции как подпрограмму(рис 29)

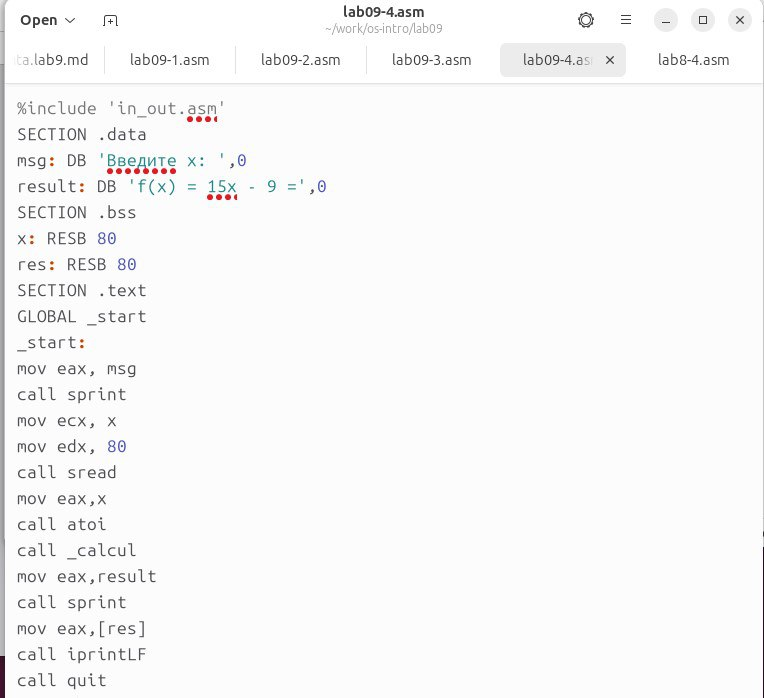


Рис. 28: рис 29

Я создам исполняемый файл и запущу его(рис 30)

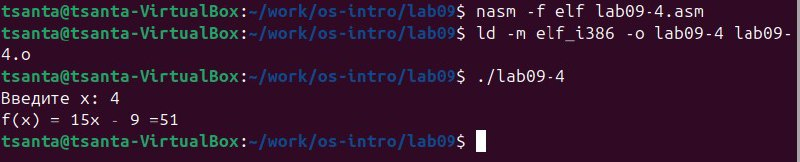


Рис. 29: рис 30

Я создам новый файл с именем lab09-5.asm(рис 31)

Рис. 30: рис 31

Рис. 30: рис 31

Я скопирую данную программу, которая вычисляет значение (3+2)\*4+5(рис 32)

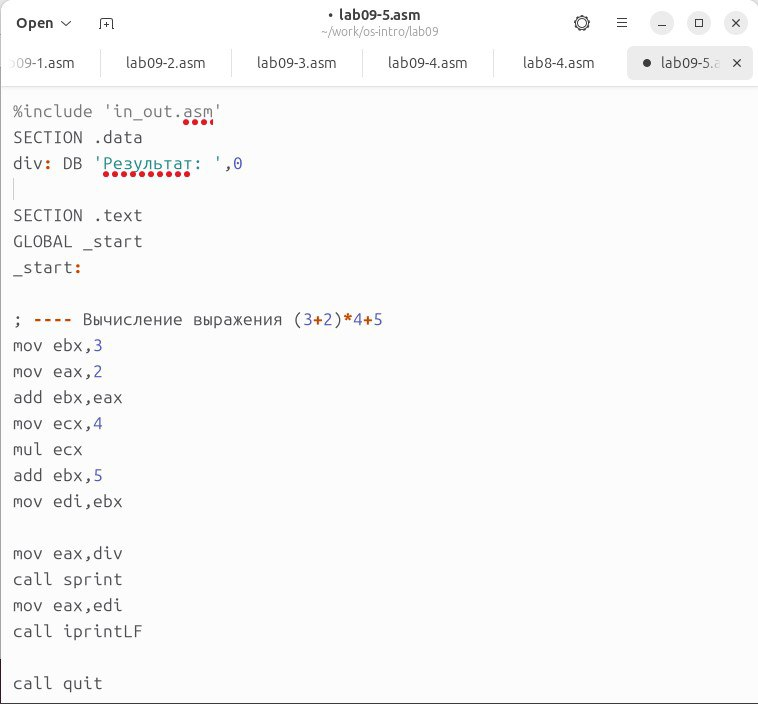


Рис. 31: рис 32

Я создам исполняемый файл и запущу его с помощью GDB(рис 33)

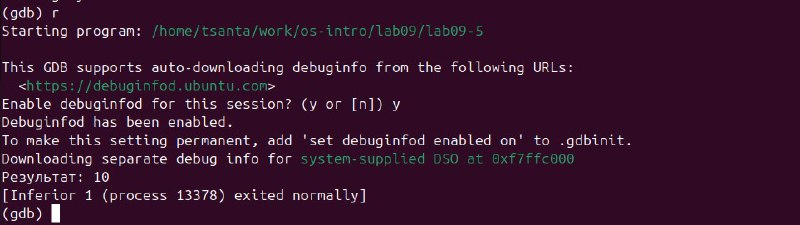


Рис. 32: рис 33

Теперь я проверю, где ошибка: первый шаг нашей программы — сложить ebx, равный 3, и eax, равный 2, что делает ebx=5, затем она перемещает 4 в ecx и по умолчанию умножает ecx на eax. что дает eax 8. В-третьих, он добавит ebx к ebx, в результате чего получится 10(рис 34).

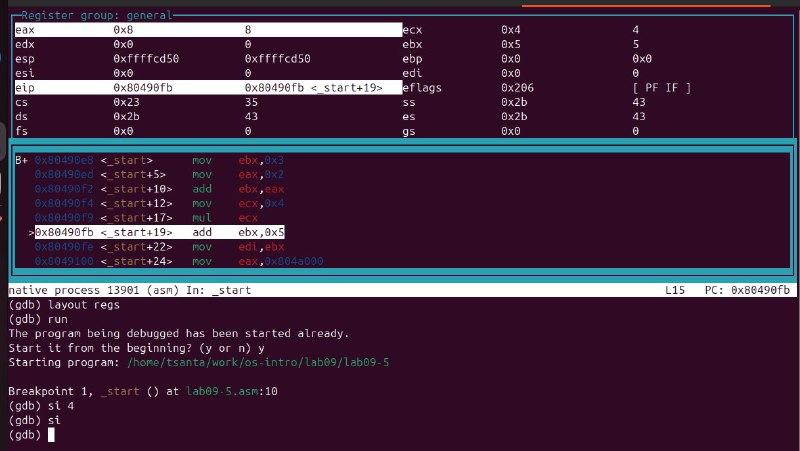


Рис. 33: рис 34

Я изменю программу так, чтобы она давала мне правильный ответ(рис 35)

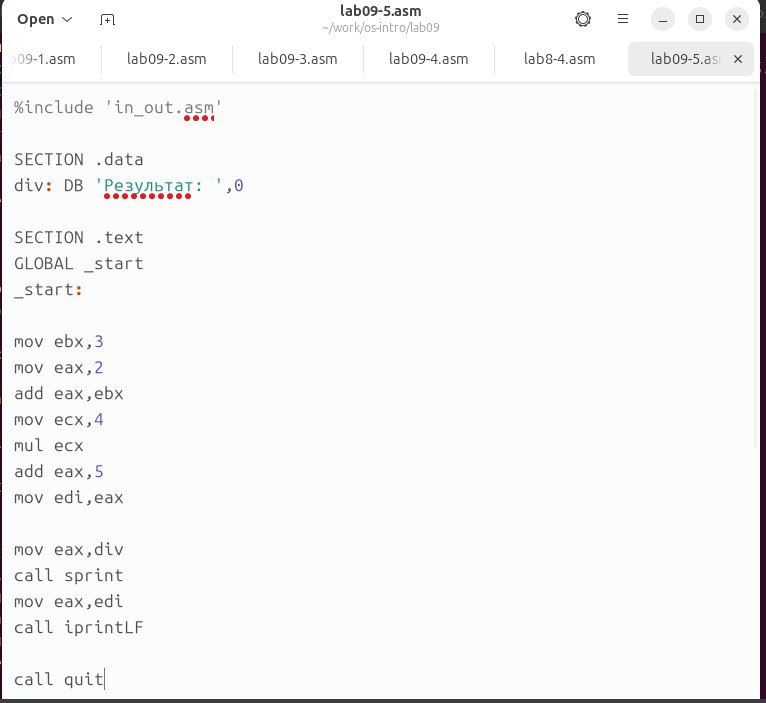


Рис. 34: рис 35

Теперь я создам исполняемый файл и запущу его(рис 36)

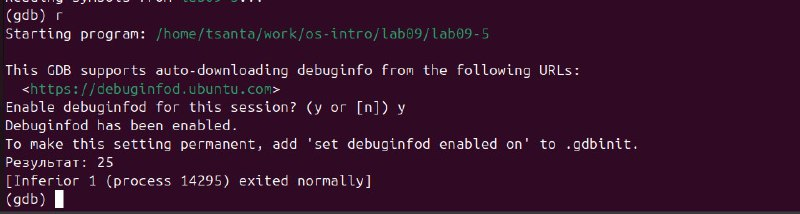


Рис. 35: рис 36

Текстовая программа для самостоятельной работы 1

%include 'in\_out.asm'  
SECTION .data  
msg: DB 'Введите x: ',0  
result: DB 'f(x) = 15x - 9 =',0  
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   
mov eax,result  
call sprint  
mov eax,[res]  
call iprintLF  
call quit  
\_calcul:  
 mov ebx, 15   
 mul ebx   
 sub eax, 9   
mov [res],eax  
ret

Текстовая программа для самостоятельной работы 2

%include 'in\_out.asm'  
  
SECTION .data  
div: DB 'Результат: ',0  
  
SECTION .text  
GLOBAL \_start  
\_start:  
  
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

# 4 Выводы

В ходе лабораторной работы я приобрел навыки написания программ с использованием подпрограмм. А также введение в методы отладки с использованием GDB и его основные возможности.

# Список литературы

1. Архитектура ЭВМ