Отчёт по лабораторной работе

Дисциплина: Архитектура компьютера

Вакутайпа Милдред

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 Выполнение лабораторной работы

**Реализация подпрограмм в NASM**

Создаю каталог для выполнения лабораторной работы № 9, перехожу0 в него и создаю файл lab09-1.asm:

Рис 1

Рис 1

Ввожу в файл lab09-1.asm текст программы для вычисления арифметического выражения f(x) = 2x + 7 с помощью подпрограммы \_calcul:

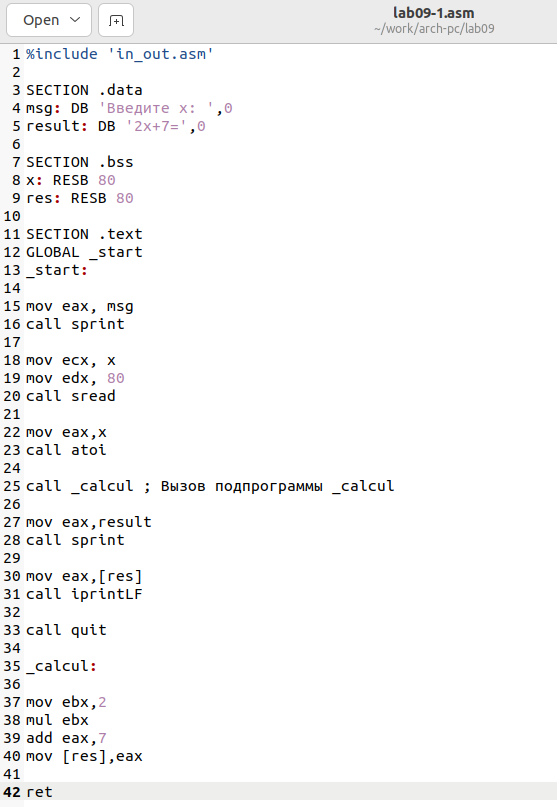


Рис 2

Создаю исполняемый файл и проверяю его работу:

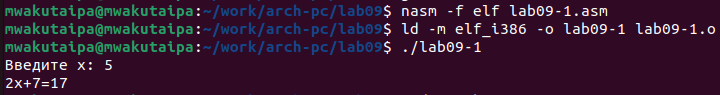


Рис 3

Изменяю текст программы, добавляя подпрограмму \_subcalcul в подпрограмму \_calcul, для вычисления выражения f(g(x)), где x вводится с клавиатуры, f(X) = 2x + 7, g(x) = 3x -1:

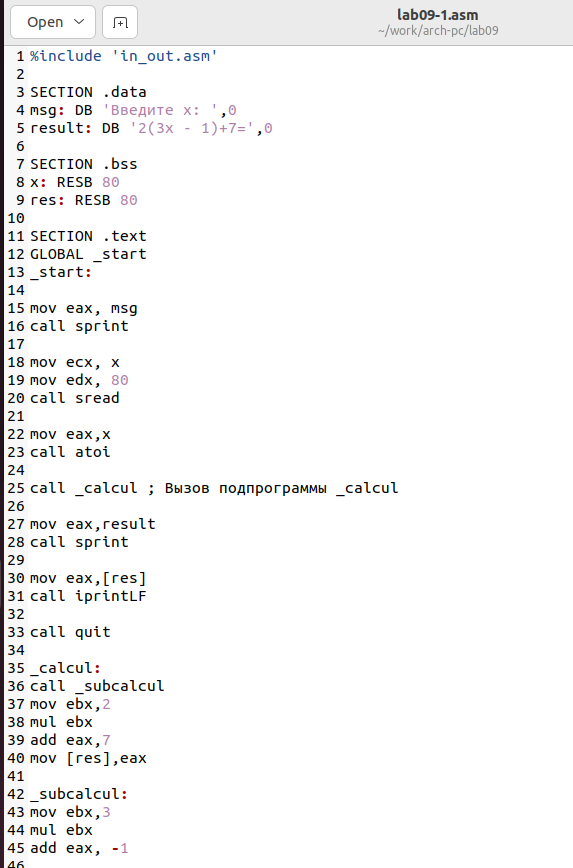


Рис 4

Создаю исполняемый файл и проверяю его работу:

Рис 5

Рис 5

**Отладка программам с помощью GDB**

Создаю файл lab09-2.asm с текстом программы для печати сообщения Hello world!:

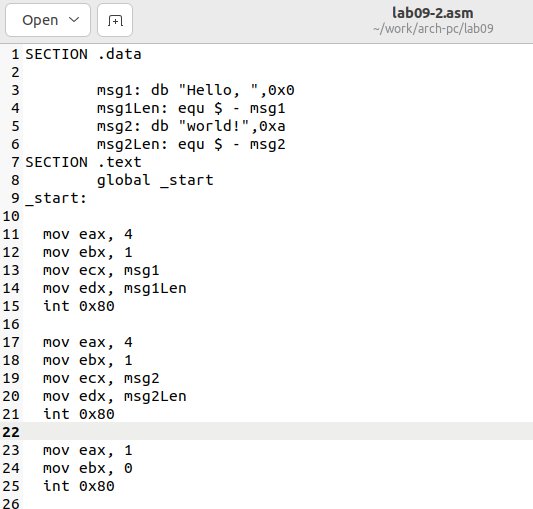


Рис 6

Создаю исполняемый файл добавляя ключ ‘-g’, для работы с GDB:

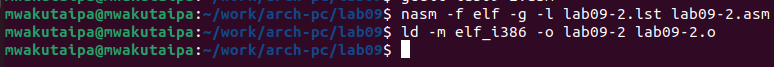


Рис 7

Загружаю исполняемый файл в отладчик gdb:

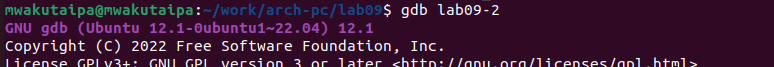


Рис 8

Проверяю работу программы, запуская ее в оболочке GDB с помощью команды run:

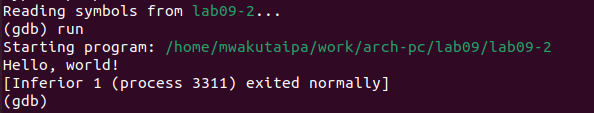


Рис 9

После установки брейкпоинт на метку \_start, запускаю программу для более подробного анализа:

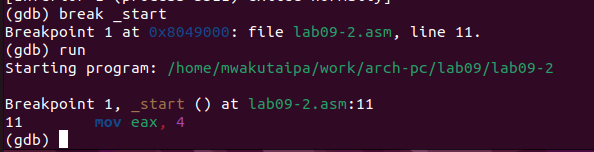


Рис 10

С помощью команды disassemble начиная с метки \_start, смотрю дисассимилированный код программы:

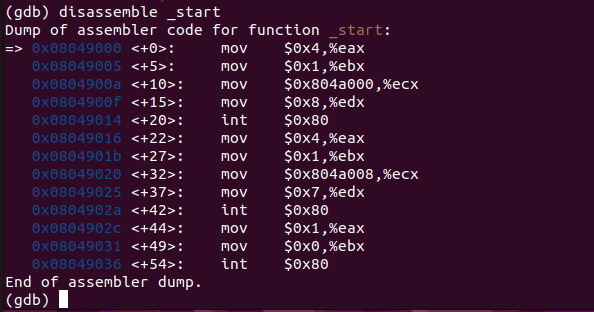


Рис 11

Переключаюсь на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel:

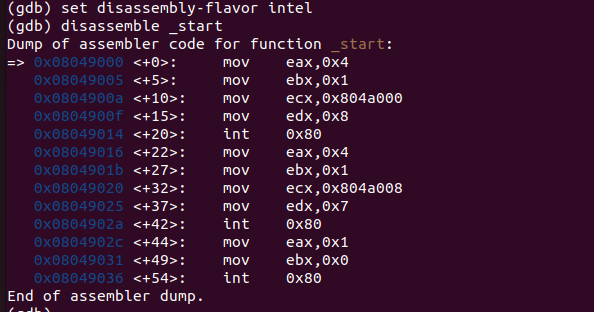


Рис 12

В синтаксе ATT первый операнд является источником, а второй является пунктом назначения(напр. $0x4,%eax). В синтаксе Intel, первый операнд является пунктом назначения, а второй является источником ( напр. eax,0x4).

Включаю режим псевдографики для более удобного анализа программы используя layout asm и layout regs:

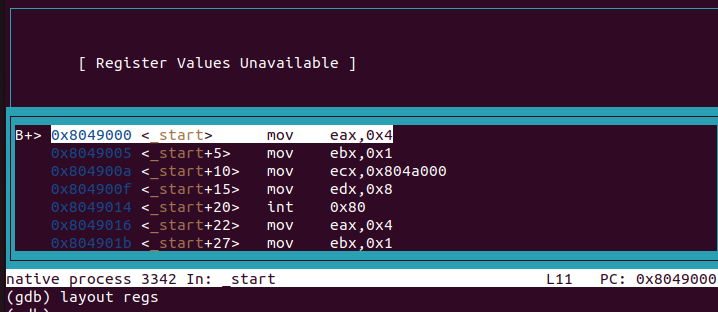


Рис 13

В верхней части должно быть названия регистров и их текущие значения, в средней части виден результат дисассимилирования программы и нижняя часть доступна для ввода команд.

**Добавление точек останова**

На предыдущих шагах была установлена точка останова по имени метки \_start.Проверяю это с помощью команды info breakpoints (i b):

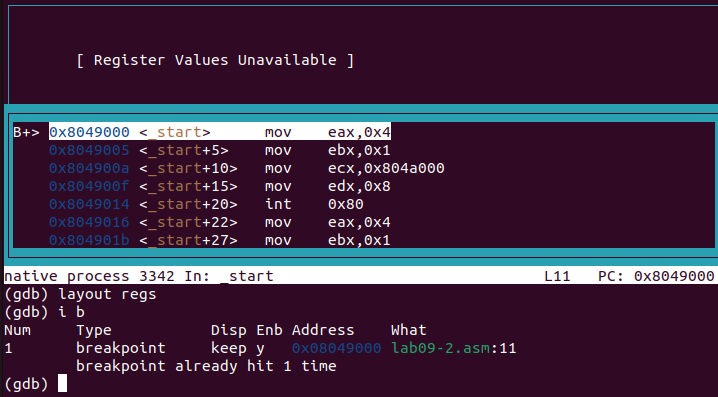


Рис 14

Установливаю еще одну точку останова по адресу инструкции (mov ebx,0x0) используя break \* и смотрю информацию о всех установленных точках останова:

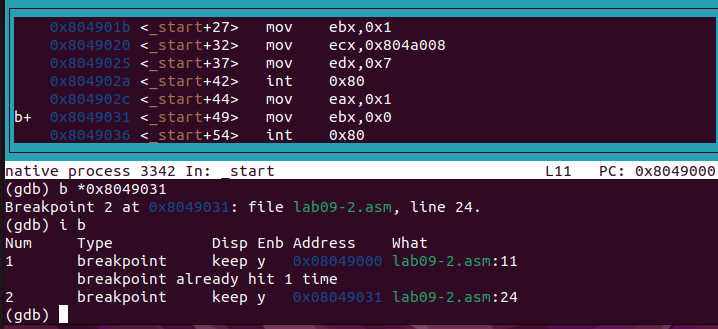


Рис 15

**Работа с данными программы в GDB**

Выполняю 5 инструкций с помощью команды stepi (si):

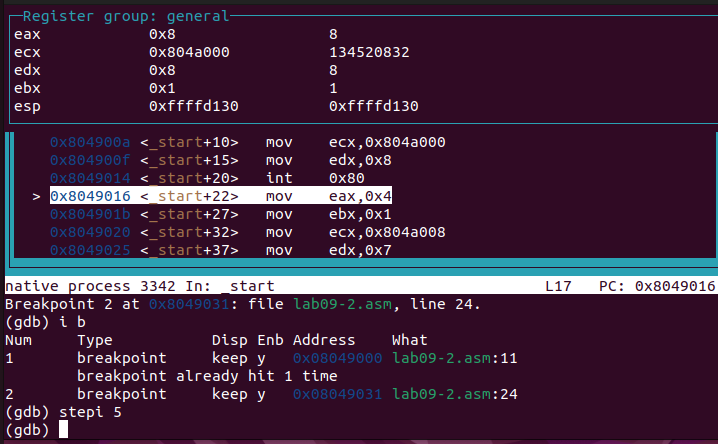


Рис 16

С помощью команды x &, смотрю значение переменной msg1 по имени:

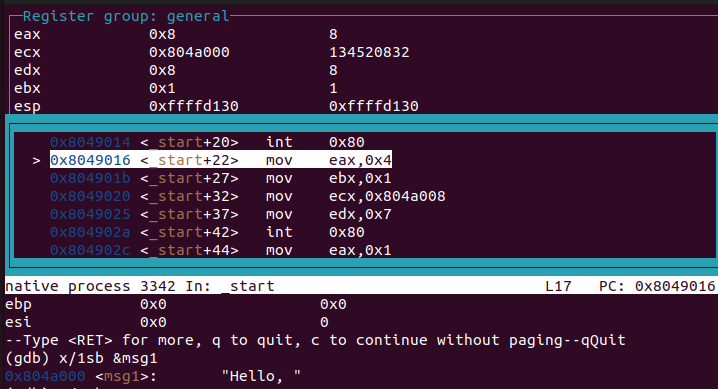


Рис 17

С помощью команды x/NFU , смотрю значение переменной msg2 по адресу:

Рис 18

Рис 18

Изменяю первый символ переменной msg1 и msg2 с помощью команды set:

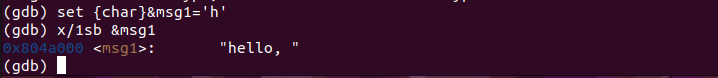


Рис 19

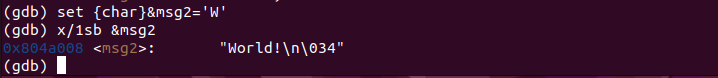


Рис 20

Команда print/F используется для просмотра значений регистров. Вывожу значение регистра edx в двоичном формате, в шестнадцатеричном формате и в символьном виде:



Рис 21

С помощью команды set изменяю значение регистра ebx:



Рис 22

(gdb) set $ebx=‘2’ - изменяет значение ebx на значение символа в двоичном формате 2(50). (gdb) set $ebx=2 изменяет значение ebx на 2.

**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 23

Рис 23

Создаю исполняемый файл:

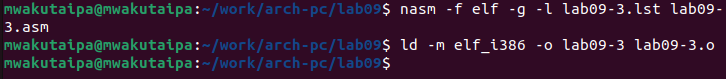


Рис 24

Загружаю исполняемый файл в отладчик с ключом –args, указивая аргументы:

Рис 25

Рис 25

Установливаю точку останова перед первой инструкцией в программе и запускаю ее:

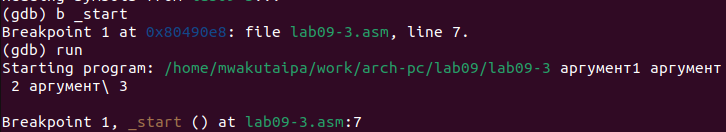


Рис 26

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы), здесь число аргументов равно 5:

Рис 27

Рис 27

Размер шага изменения адреса по умолчанию равен 4 при отладке программы на архитектуре x86. Это связано с тем, что в 32-битных системах адреса памяти обычно представляются как 32-битные числа, и каждый адрес соответствует одному байту:

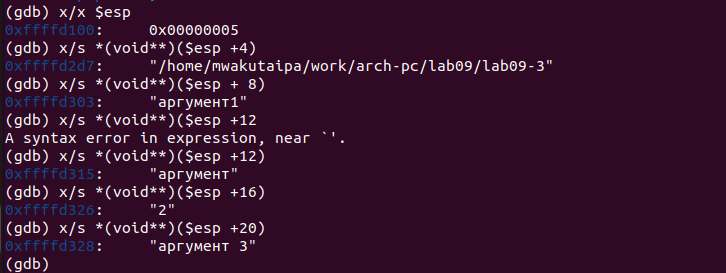


Рис 28

# 4 Выполнение самостоятельной работы

**Задание 1**

Копирую файл task8.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран сумму значений функции f(x) = 5(2 + x) для некоторых значении x (аргументы командной строки), в файл с именем task9-1.asm:

Рис 29

Рис 29

Редактирую программу для вычисление значения функции f(x) как подпрограмму:

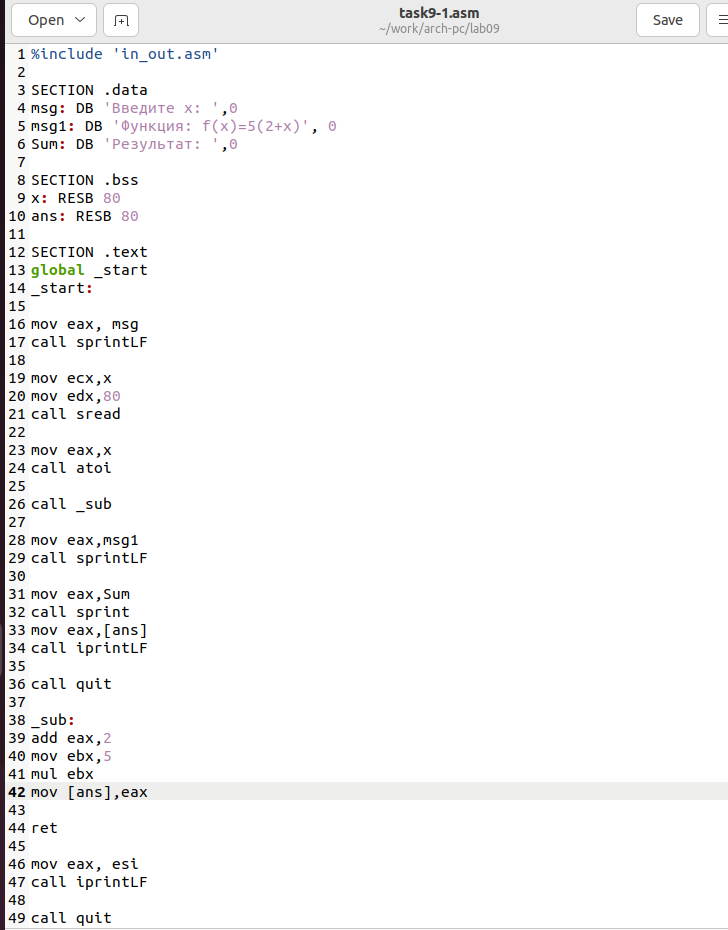


Рис 30

Создаю исполняемый файл и проверяю его работу:

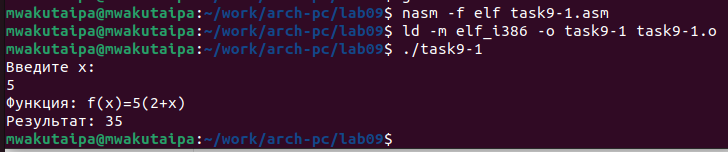


Рис 31

Код программы:

%include 'in\_out.asm'  
  
SECTION .data  
msg: DB 'Введите x: ',0  
msg1: DB 'Функция: f(x)=5(2+x)', 0  
Sum: DB 'Результат: ',0  
  
SECTION .bss  
x: RESB 80  
ans: RESB 80  
  
SECTION .text  
global \_start  
\_start:  
  
mov eax, msg  
call sprintLF  
  
mov ecx,x  
mov edx,80  
call sread  
  
mov eax,x  
call atoi  
  
call \_sub  
  
mov eax,msg1  
call sprintLF  
  
mov eax,Sum  
call sprint  
mov eax,[ans]  
call iprintLF  
  
call quit  
  
\_sub:  
add eax,2  
mov ebx,5  
mul ebx  
mov [ans],eax  
  
ret  
  
mov eax, esi   
call iprintLF  
   
call quit

**Задание 2**

Создаю файл task9-2.asm и вставляю в него программу вычисления выражения(3+2)∗4+5:

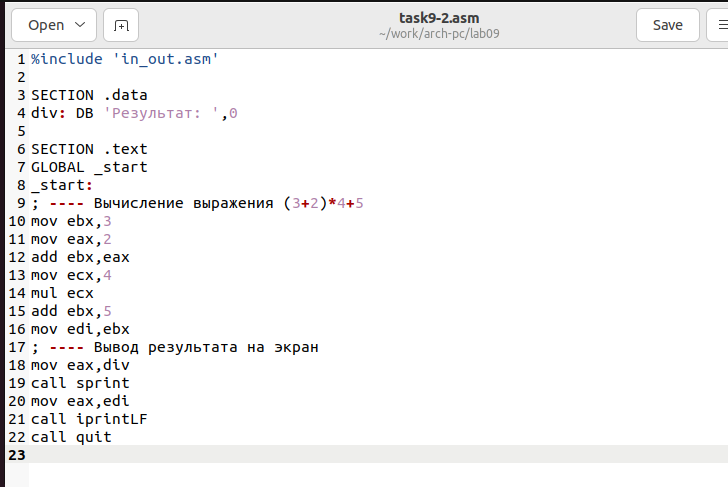


Рис 32

Создаю испольняемый файл:

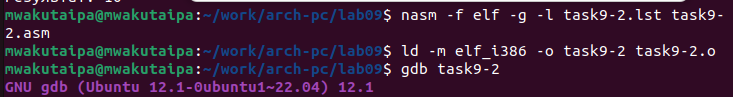


Рис 33

При запуске она выводит неверный результат:

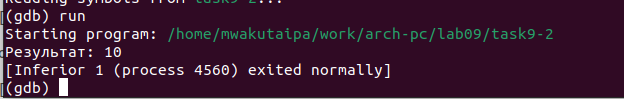


Рис 34

Переключаюсь на отображение команд с Intel’овским синтаксисом:

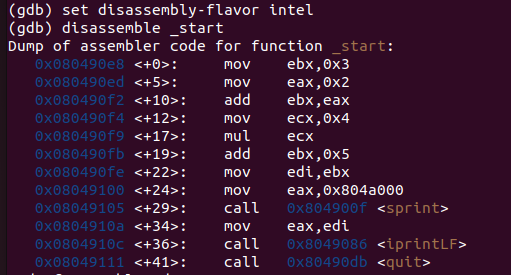


Рис 35

Включаю режим псевдографики для более удобного анализа программы используя layout asm:

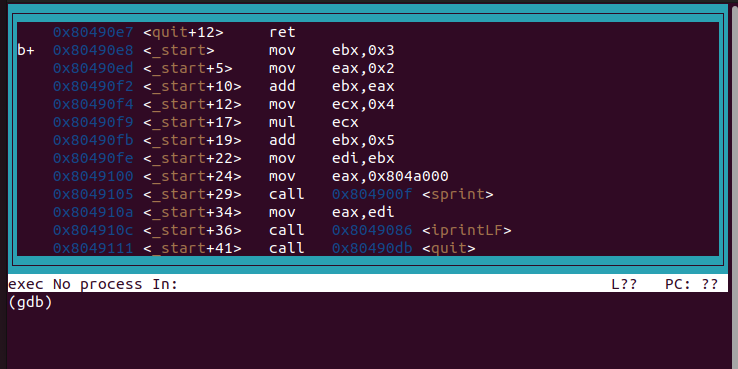


Рис 36

С помощью layout regs, я могу видеть названия регистров и их текущие значения. Выполняю 6 инструкций с помощью команды stepi (si) и при этом замечаю ,что после того как программа суммирует 3 и 2, значение храняется в регистре ebx. Значение регистра ecx(4) умножает на 2(значение регистра eax) а затем суммирует 5 и значение регистра ebx (5):

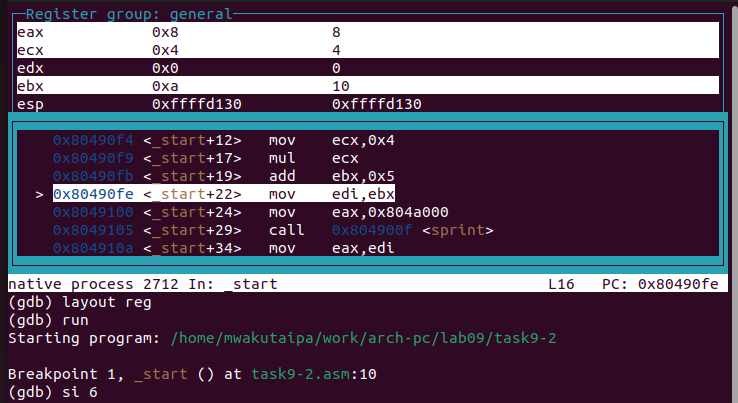


Рис 37

Исправляю код ,чтобы программа выводила правильный ответ:

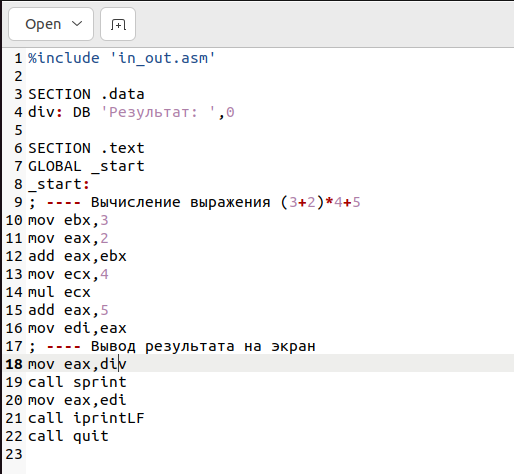


Рис 38

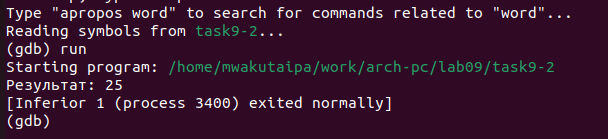


Рис 39

Код 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 работы я освоила написание программ с использованием подпрограмм и знакомила методы отладки при помощи GDB и его основные возможности.

# 6 Список литературы

[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)