Отчет по выполнению лабораторной работы №8

Дисциплина: архитектура компьютеров

Намруев Максим Саналович

Содержание

# 1 Цель работы

Приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями

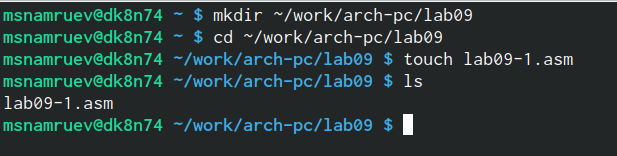
# 2 Задание

1. Релизация подпрограмм в NASM
2. Отладка программ с помощью GDB
3. Добавление точек останова
4. Работа с данными программы в GDB
5. Обработка аргументов командной строки в GDB
6. Задание для самостоятельной работы.

# 3 Выполнение лабораторной работы

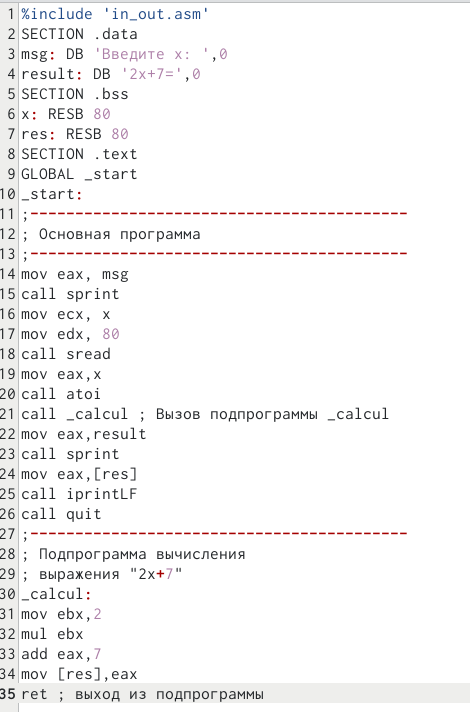
## 3.1 Реализация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы №9, перехожу в него и создаю файл lab09-1.asm. (рис. [??]).



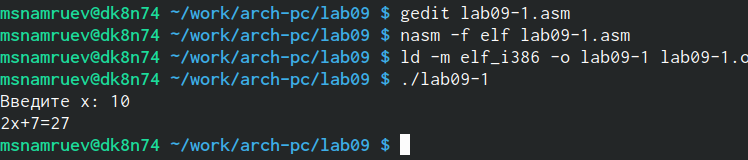
Создание каталога и файла

Ввожу в файл lab09-1.asm текст программы из листинга 9.1.(рис. [??])



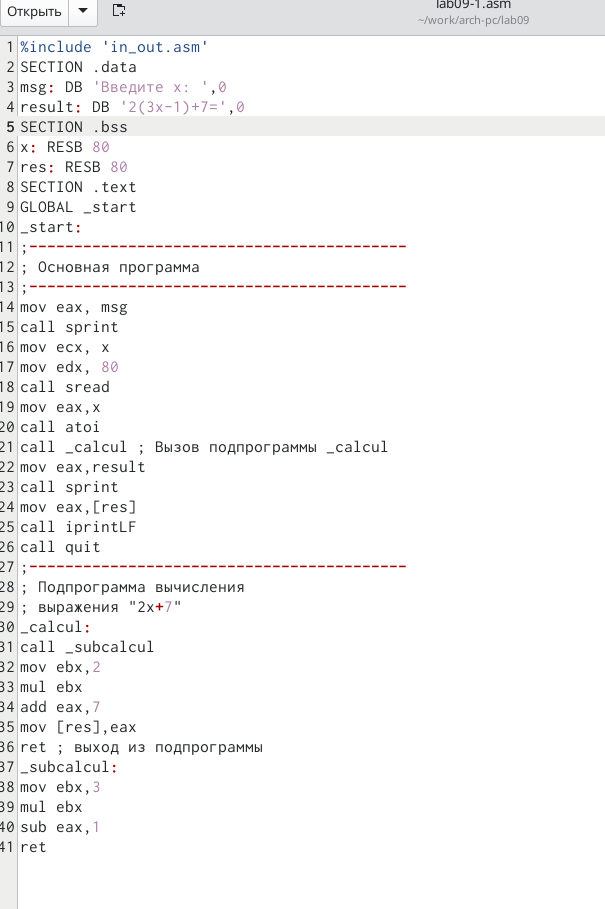
Ввод программы из листинга 9.1

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



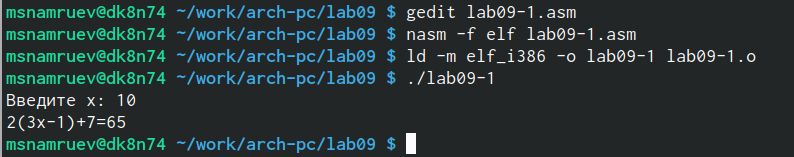
Проверка работы программы

Изменяю текст программы, добавляя подпрограмму \_subcalcul в подпрограммы \_calcul.(рис. [??])



Изменение текста программы

Создаю исполняемый файл и запускаю его.(рис. [??])

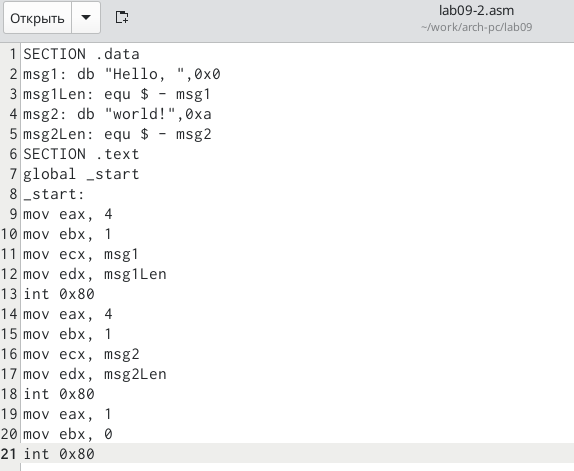


Запуск программы

Программа работает корректно.

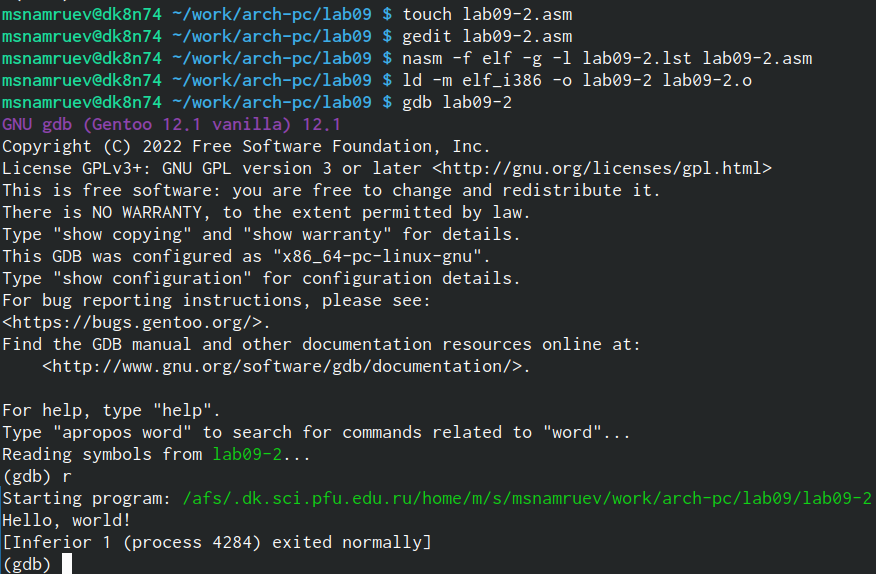
## 3.2 Отладка программам с помощью GDB

Создаю файл lab09-2.asm с текстом программы из Листинга 9.2.(рис. [??])



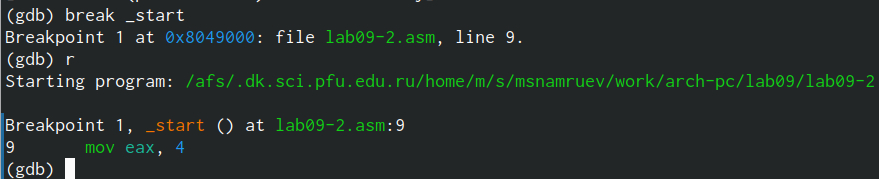
Создание файла печати сообщения Hello world!

Получаю исполняемый файл.Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’.Далее загружаю исполняемый файл в отладчик gdb и запускаю его с помощью команды run.(рис. [??])



Получение и загрузка в отлатчик исполняемого файла

Устанавливаю брейкпоинт на метку \_start и запукаю программу.(рис. [??])



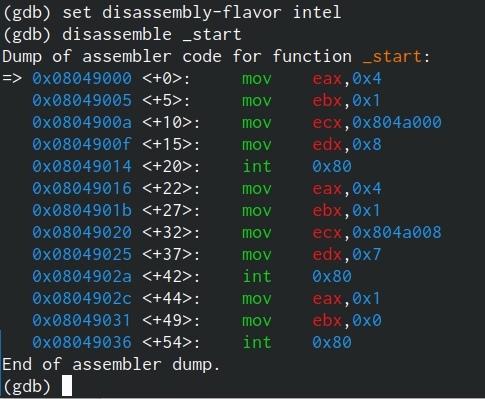
Установление брейкпоинта

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



Просмотр дисассимилированного кода программы

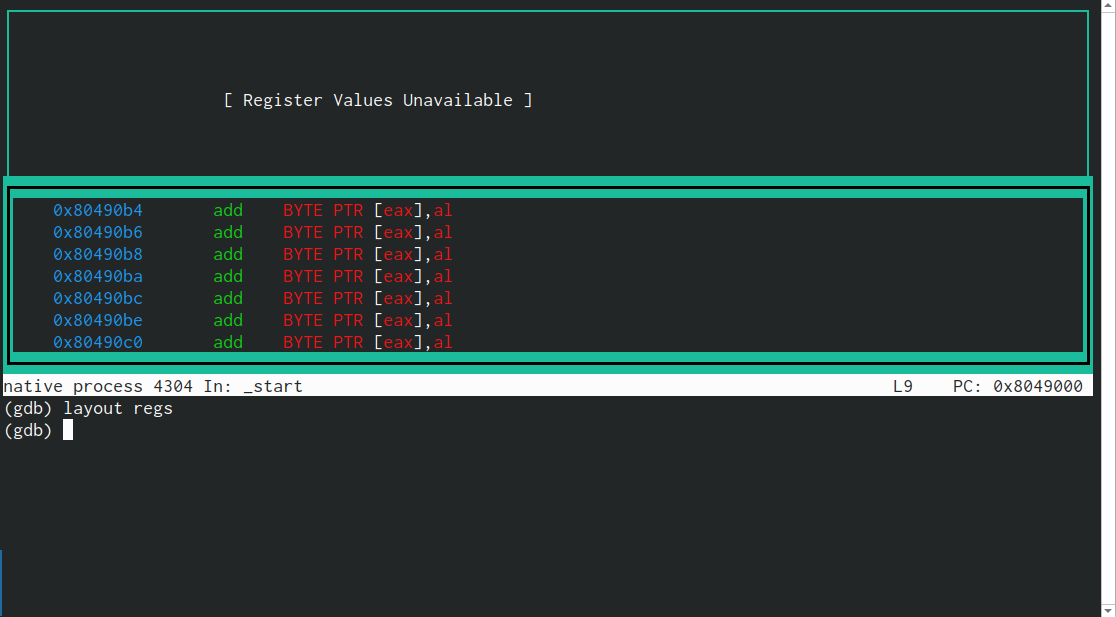
Переключаюсь на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel.(рис. [??])



Переключение на Intel’ский синтаксис

Отличия заключаются в том, что в режиме ATT используются “%” перед перед именами регистров и “$” перед именами операндов, а в режиме Intel используется обычный синтаксис.

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



Включение режима псевдографики

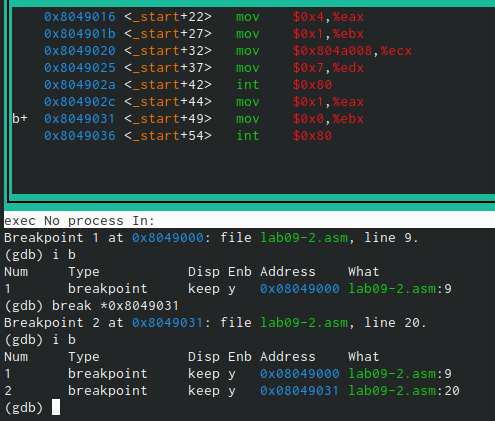
### 3.2.1 Добавление точек останова

Проверяю точку останова по имени метки.(рис. [??])

Проверка точки останова

Проверка точки останова

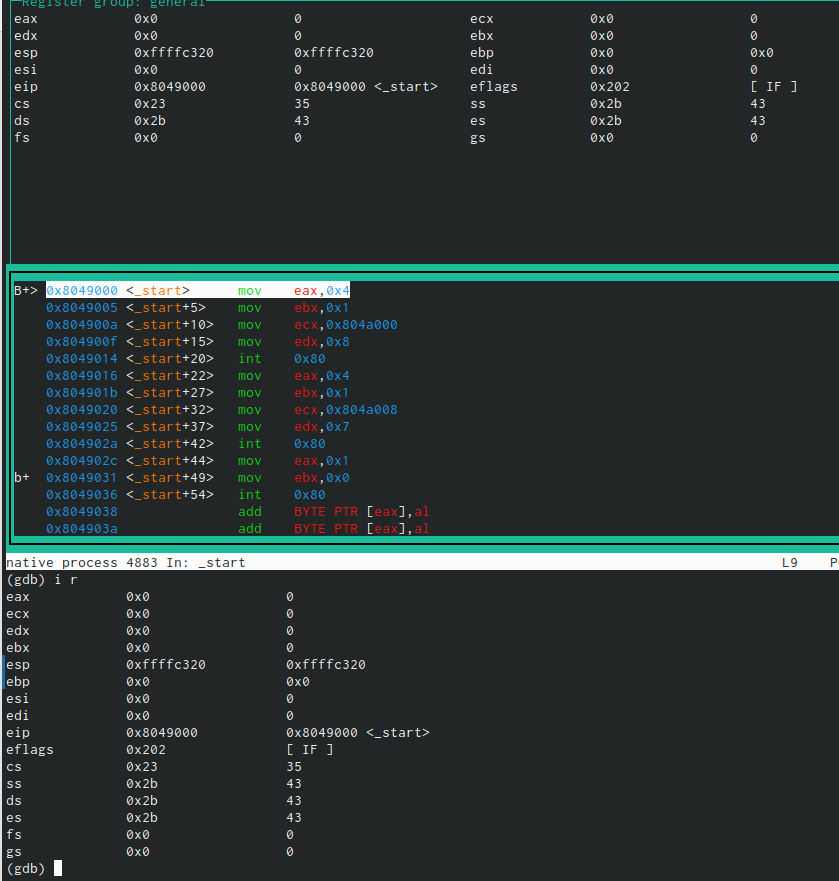
Определю адрес пердпоследней инструкции и устанавливаю точку останова. Далее смотрю информацию о всех установленных точках останова.(рис. [??])



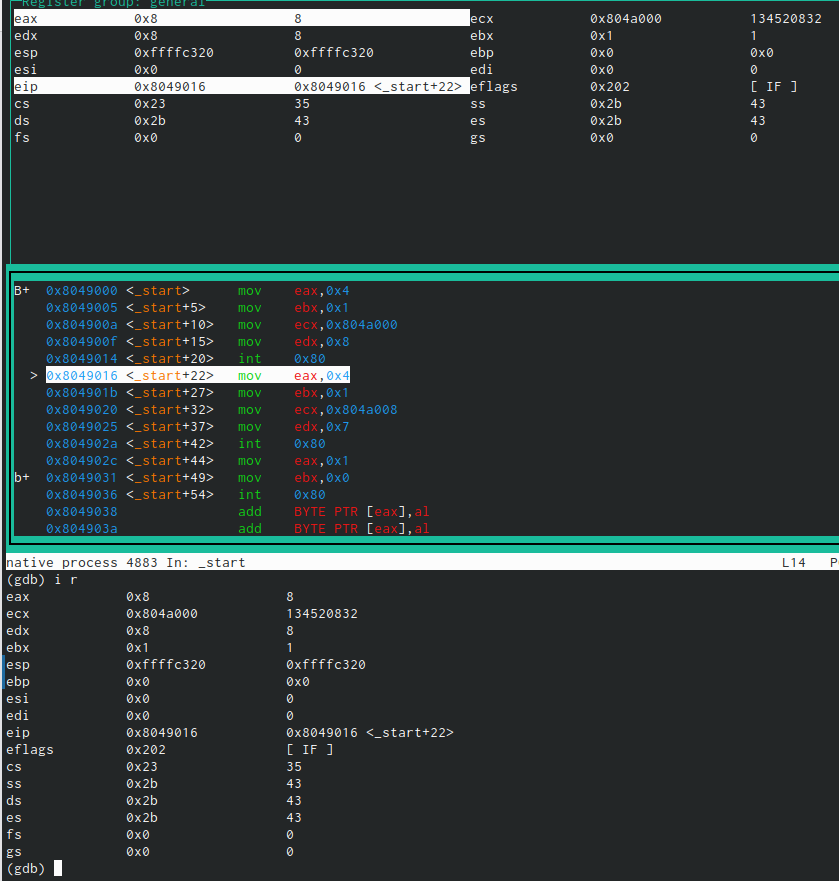
Установка точки останова и её проверка

### 3.2.2 Работа с данными программы в GDB

Выполняю 5 инструкций с помощью команды stepi и слежу за изменением регистров.(рис. [??]) (рис. [??])



До использования комадны stepi



После использования команды stepi

Изменились регистры eax,ebx,ecx,edx

Просматриваю значение переменной msg1 по имени.(рис. [??])

Просмотр значения переменной msg1

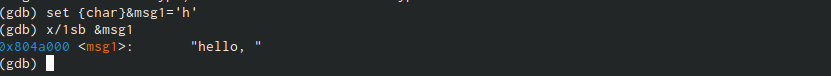
Просмотр значения переменной msg1

Также просматриваю значение переменной msg2.(рис. [??])

Просмотр значения переменной msg2

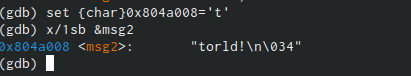
Просмотр значения переменной msg2

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



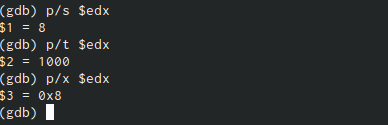
Изменение первого символа msg1

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



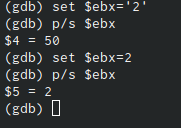
Изменение первого символа переменной msg2

Вывожу в различных форматах значение регистра ebx.(рис. [??])



Вывод значений регистра ebx

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



Изменение значения регистра ebx

Разница в том, что в первый раз мы вводим символ, который преобразуется в число, а во второй раз мы вводим само число.

### 3.2.3 Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm, созданный при выполнении лабораторной работы No8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm.(рис. [??])

Копирование файла

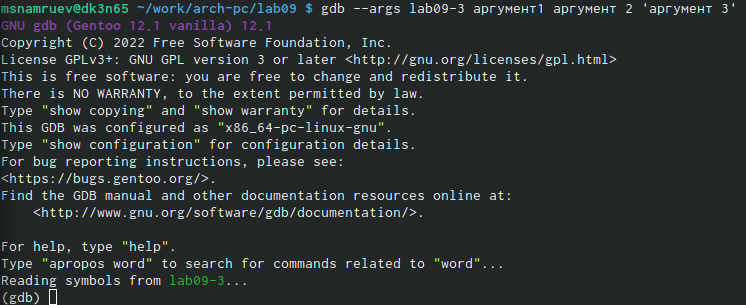
Копирование файла

Создаю исполняемый файл.(рис. [??])

Создание исполняемого файла

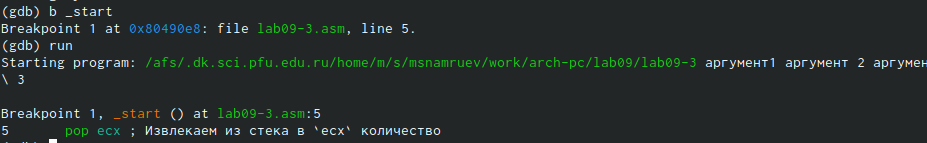
Создание исполняемого файла

Заргужаю исполняемый файл в отладчик, указав агрументы.(рис. [??])



Загрузка исполняемого файла в отладчик

Для начала устанавливаю точку останова перед первой инструкцией в программе и запускаю её.(рис. [??])



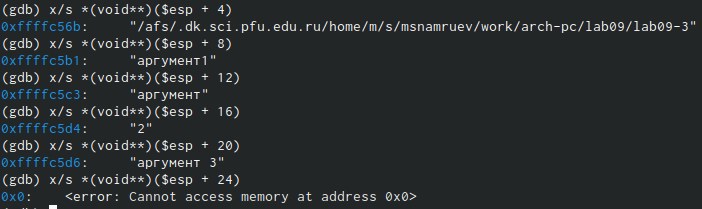
Установка точки остановы

Просматриваю вершину стека, то есть число аргументов строки(включая имя программы).(рис. [??])

Просмотр вершины стека

Просмотр вершины стека

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

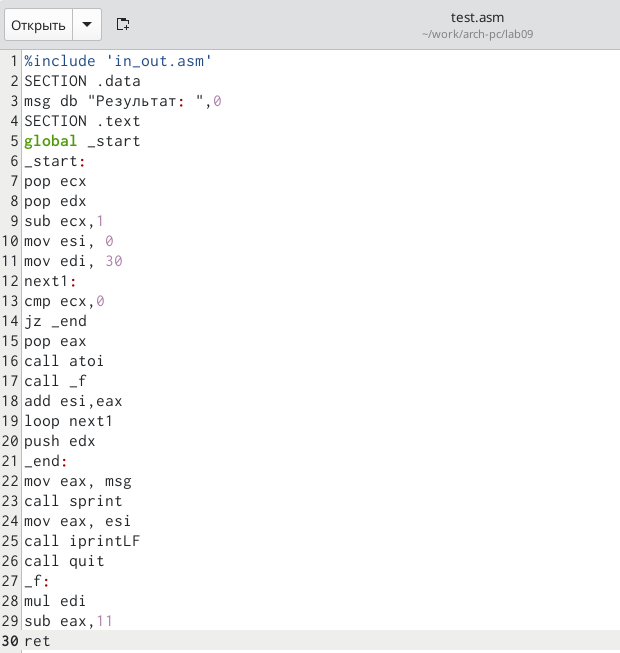


Просмотр остальных позиций стека

Шаг изменения адреа равен 4, потому что занчение регистра esp в стеке увеличивается на 4.

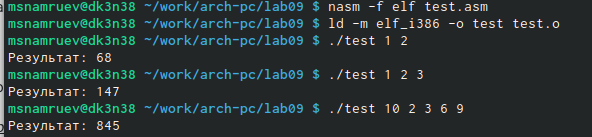
# 4 Задание для самостоятельной работы.

1. Открываю программу из лабораторной работы №8 и начинаю её редактировать.(рис. [??])



Редактирование программы

Создаю исполняемый файл и запускаю его, чтобы проверить работу программы.(рис. [??])



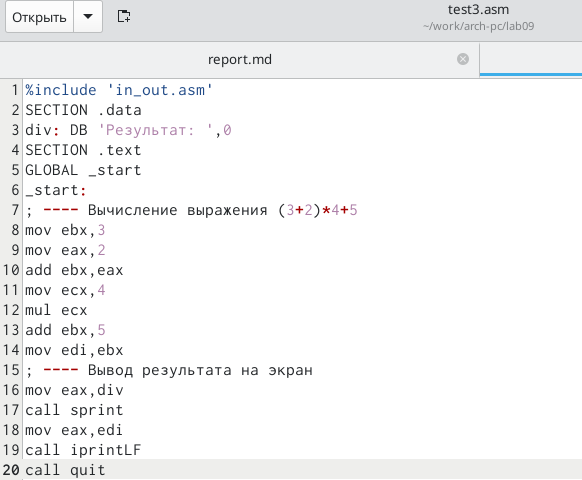
Проверка работы программы

Программа работает верно.

Текст программы:

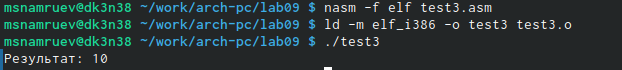
%include ‘in\_out.asm’  
SECTION .data  
msg db “Результат:”,0  
SECTION .text  
global \_start  
\_start:  
pop ecx  
pop edx  
sub ecx,1  
mov esi, 0  
mov edi, 30  
next1:  
cmp ecx,0  
jz \_end  
pop eax  
call atoi  
call \_f  
add esi,eax  
loop next1  
push edx  
\_end:  
mov eax, msg  
call sprint  
mov eax, esi  
call iprintLF  
call quit  
\_f:  
mul edi  
sub eax,11  
ret

1. Создаю файл test3.asm и ввожу туда текст программы из листинга 9.3.(рис. [??])



Ввод программы из листинга 9.3

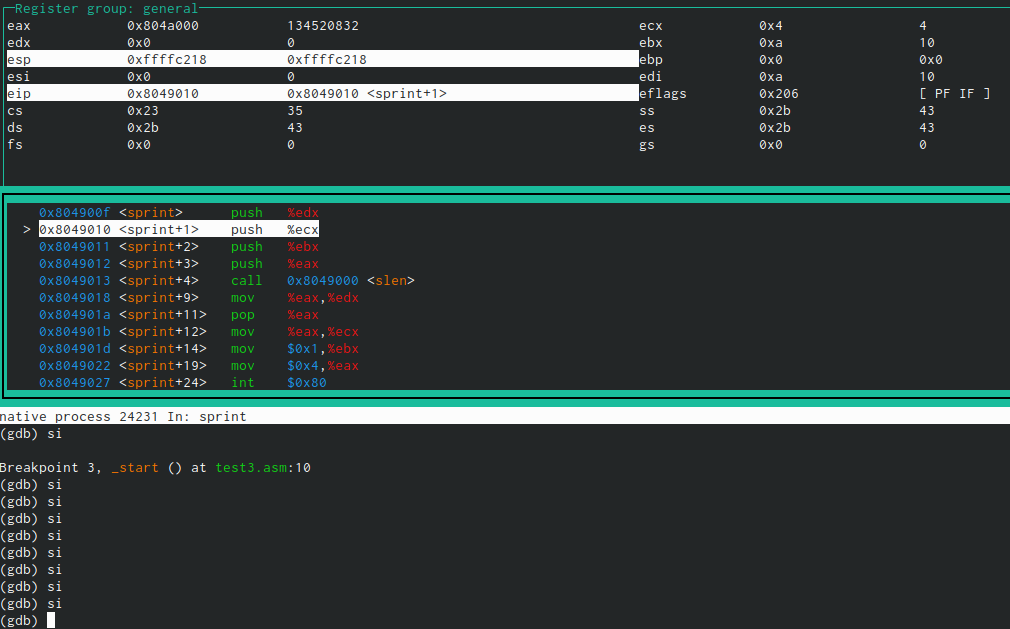
Содзаю исполняемый файл и запускаю его.(рис. [??])



Запуск программы

Убеждаюсь, что программа работает неверно.

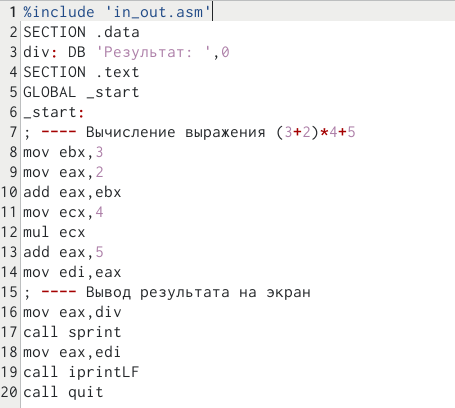
Создаю исполняемый файл для работы с GDB и запускаю его через режим отладки. Создаю брейкпойнт и пошагово просматириваю программу.(рис. [??])



Просмотр программы

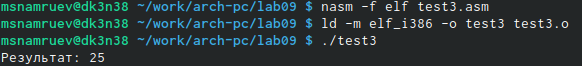
Замечаю, что после выполнения инструкции mul программы умножент 4 на 2 на не на 5, как должно быть.Из-за этого программа выдает неверный результат.

Далле открываю файл с программой и исправляю ошибку.(рис. [??])



Исправление ошибки

Создаю исполняемый файл и запускаю его.(рис. [??])



Просмотр программы

Теперь программа работает верно.

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

После выполнения данной лабораторной работы я приобрел навыки написания программ с использование подпрограмм и познакомился с методами отладки при помощи GDB и его основными возможностями.

# Список литературы