Отчёт по лабораторной работе №9

Понятие подпрограммы. Отладчик GDB

Новикова Анастасия Андреевна

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 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 работы №9 и перехожу в него(рис. 1).

Создание каталога

Рис. 1: Создание каталога

Копирую файл in\_out.asm в созданный каталог, так как он понадобится для написания программ (рис. 3).

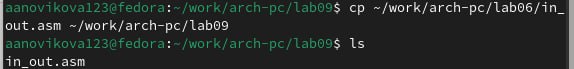


Рис. 2: Копирование файла

Создаю файл lab09-1.asm в новом каталоге (рис. 3).

Создание файла

Рис. 3: Создание файла

Открываю файл в текстовом редакторе и переписываю код программы из листинга 9.1 (рис. 4).

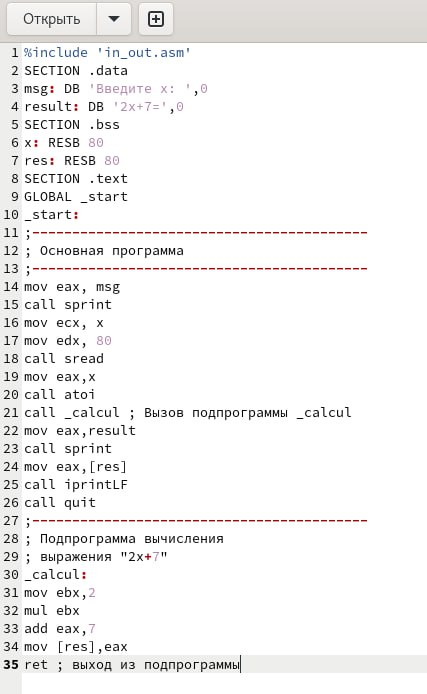


Рис. 4: Редактирование файла

Создаю объектный файл программы и после компановки запускаю его (рис. 5). Код с подпрограммой работает успешно.

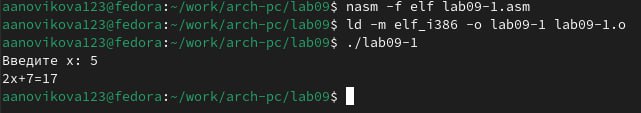


Рис. 5: Запуск программы

Изменяю текст файла,добавив подпрограмму sub\_calcul в подпрограмму \_calcul (рис. 6).

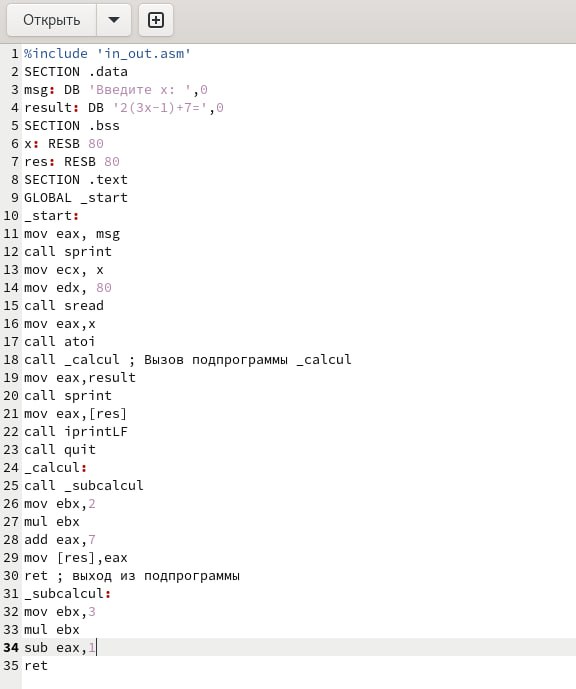


Рис. 6: Редактирование файла

%include 'in\_out.asm'  
SECTION .data  
msg: DB 'Введите x: ',0  
result: DB '2(3x-1)+7=',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 ; Вызов подпрограммы \_calcul  
mov eax,result  
call sprint  
mov eax,[res]  
call iprintLF  
call quit  
\_calcul:  
call \_subcalcul  
mov ebx,2  
mul ebx  
add eax,7  
mov [res],eax  
ret ; выход из подпрограммы  
\_subcalcul:  
mov ebx,3  
mul ebx  
sub eax,1  
ret

Запускаю исполняемый файл (рис. 7).Программа работает верно.

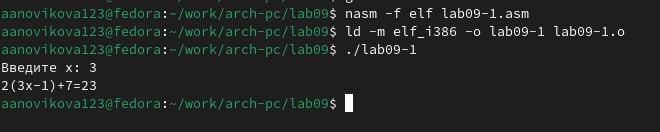


Рис. 7: Запуск программы

## 3.2 Отладка программ с помощью GDB

Создаю файл lab09-2.asm, используя команду touch (рис. 8).

Создание файла

Рис. 8: Создание файла

Записываю код программы из листинга 9.2,который выводит сообщение Hello world (рис. 9).

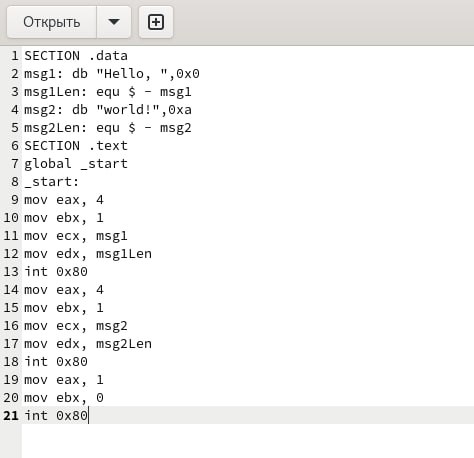


Рис. 9: Редактирование файла

Получаю исполняемый файл. Для работы с GDB провожу трансляцию программ с ключом “-g” и загружаю исполняемый файл в отладчик (рис. 10).

Запуск исполняемого файла

Рис. 10: Запуск исполняемого файла

Проверяю работу программы в оболочке GDB с помощью команды run (рис. 11).

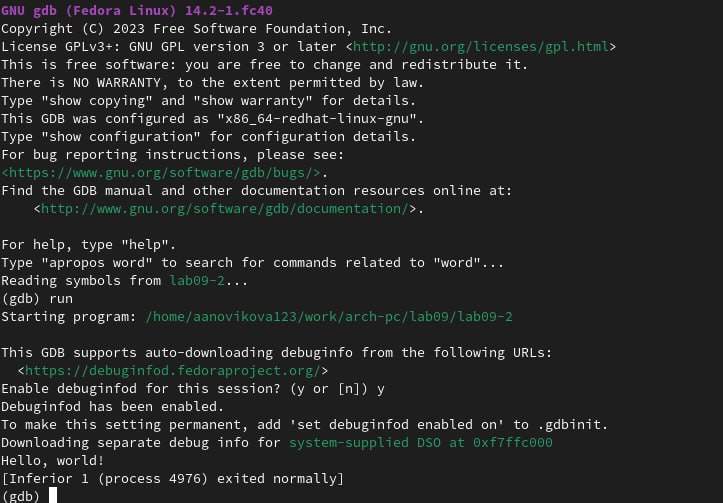


Рис. 11: Запуск программы в отладчике

Для более подробного анализа устанавливаю брейкпоинт на метку \_start,с которой начинается выполнение ассемблерной программы (рис. 12)

Установка брейкпоинта

Рис. 12: Установка брейкпоинта

Запускаю её (рис. 13).

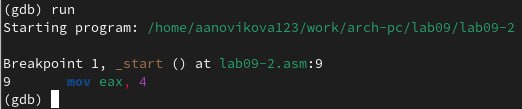


Рис. 13: Запуск

С помощью команды “disassemble \_start” просматриваю дисассимилированный код программы (рис. 14).



Рис. 14: Диссассимилированный код программы

Переключаюсь на отображение команд с Intel’овским синтаксисом, введя команду “set disassembly-flavor intel” (рис. 15).

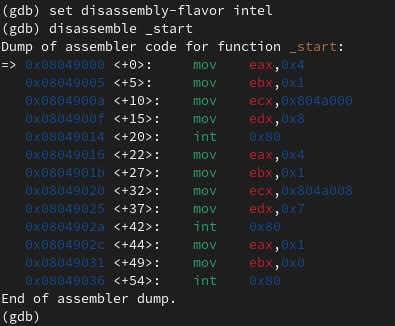


Рис. 15: Отображение с Intel’овским синтаксисом

Основное различие заключается в том, что в режиме Intel пишется сначала сама команда, а потом её машинный код, в то время как в режиме ATT идет сначала машинный код, а только потом сама команда.

## 3.3 Добавление точек останова

Проверяю наличие точки останова с помощью команды info breakpoints (i b) (рис. 16).

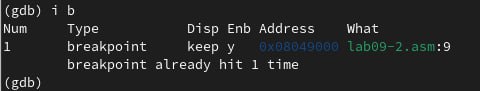


Рис. 16: Точка останова

Устанавливаю ещё одну точку останова по адресу инструкции,которую можно найти в средней части в левом столбце соответствующей инструкции (рис. 17).

Установка точки останова

Рис. 17: Установка точки останова

Просматриваю информацию о точках останова (рис. 18).

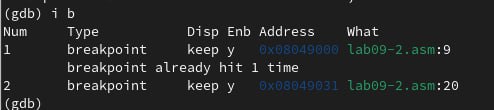


Рис. 18: Точки останова

## 3.4 Работа с данными программы в GDB

Выполняю 5 инструкций командой si(рис. 19). Во время выполнения команд менялись регистры ebx, ecx, edx, eax, eip.

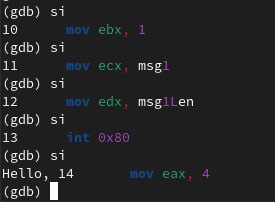


Рис. 19: Отслеживаем регистры

Просматриваю содержимое регистров с помощью команды info registers (рис. 20).

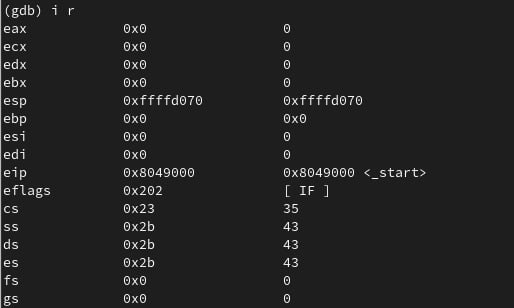


Рис. 20: info register

Узнаю значение переменной msg1 по имени (рис. 21).

Значение переменной по имени

Рис. 21: Значение переменной по имени

Просматриваю значение переменной msg2 по адресу,который можно определить по дизассемблированной инструкции (рис. 22).

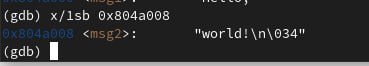


Рис. 22: Значение переменной по адресу

Меняю первый символ переменной msg1 (рис. 23).

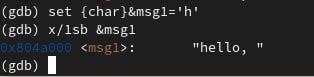


Рис. 23: Изменение переменной

Также меняю первый символ переменной msg2 (рис. 24).

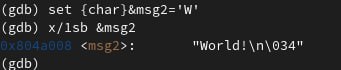


Рис. 24: Изменение второй переменной

Вывожу значение регистра edx в различных форматах (в шестнадцатеричном, двоичном и символьном форматах) (рис. 25).

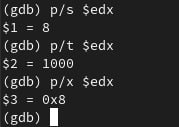


Рис. 25: Изменение значений в разные форматы

С помощью команды set изменяю значение регистра ebx (рис. 26).

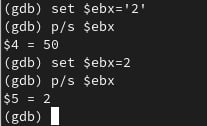


Рис. 26: Изменение значений ebx

Значение регистра отличаются, так как в первом случае мы выводим код символа 2, который в десятичной системе счисления равен 50, а во втором случае выводится число 2,представленное в этой же системе.

## 3.5 Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm,созданный при выполнении лабораторной работы №8, который выводит на экран аргументы, в файл с именем lab09-3.asm (рис. 27).

Копирование файла

Рис. 27: Копирование файла

Создаю исполняемый файл,использую ключ –args для загрузки программы в GDB. Загружаю исполняемый файл,указав аргументы (рис. 28).

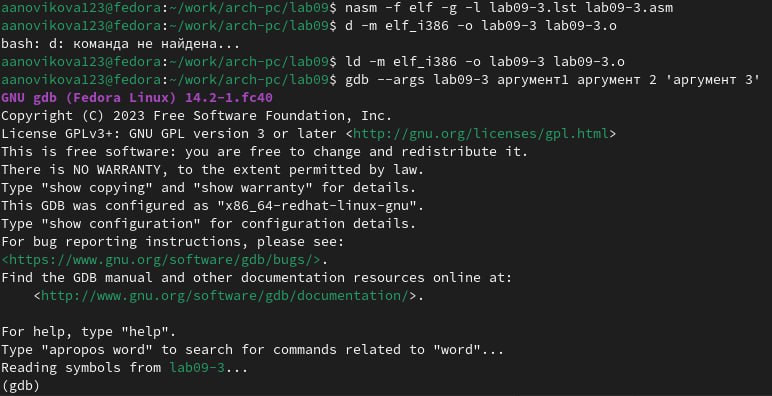


Рис. 28: Создание файла

Устанавливаю точку останова перед первой инструкцией в программе и запускаю её (рис. 29).

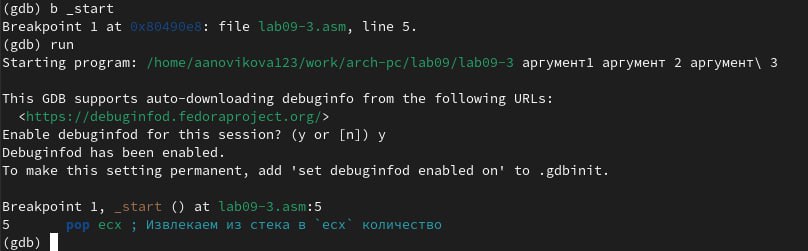


Рис. 29: Запуск программы с точкой останова

Просматриваю адрес вершины стека,который хранится в регистре esp (рис. 30).

Регистр esp

Рис. 30: Регистр esp

Ввожу другие позиции стека- в отличие от адресов, располагается адрес в памяти: имя, первый аргумент, второй и т.д (рис. 31).

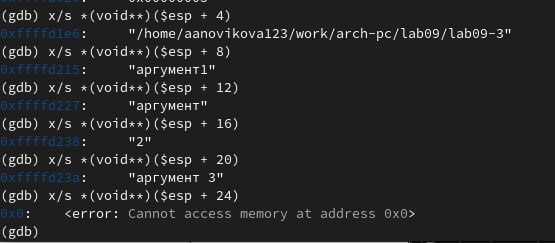


Рис. 31: Позиции стека

Шаг изменения адреса равен 4, потому что адресные регистры имеют размерность 32 бита(4 байта).

## 3.6 Задание для самостоятельной работы

***ЗАДАНИЕ 1***

Создаю файл для первого самостоятельного задания, который будет называться lab09-4.asm (рис. 32).

Создание файла

Рис. 32: Создание файла

Редактирую код программы lab8-4.asm, добавив подпрограмму, которая вычисляет значения функции f(x) (рис. 33).

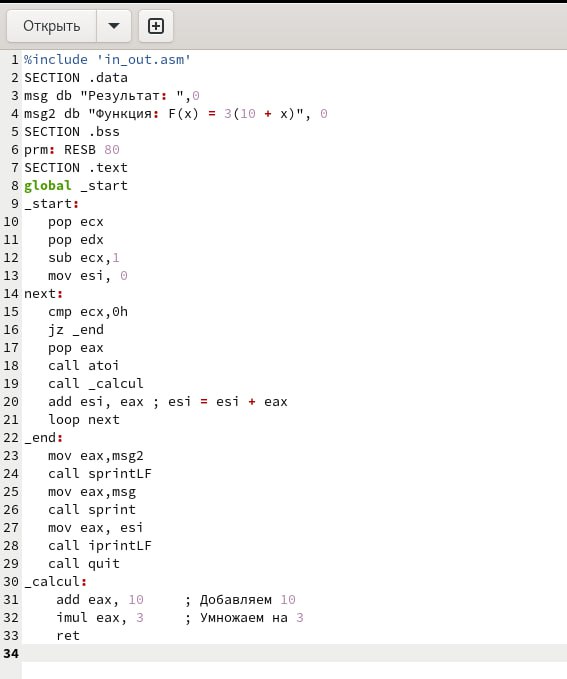


Рис. 33: Редактирование файла

%include 'in\_out.asm'  
SECTION .data  
msg db "Результат: ",0  
msg2 db "Функция: F(x) = 3(10 + x)", 0  
SECTION .bss  
prm: RESB 80  
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 \_calcul  
 add esi, eax ; esi = esi + eax  
 loop next  
\_end:  
 mov eax,msg2  
 call sprintLF  
 mov eax,msg  
 call sprint  
 mov eax, esi  
 call iprintLF  
 call quit  
\_calcul:  
 add eax, 10 ; Добавляем 10  
 imul eax, 3 ; Умножаем на 3  
 ret

Создаю исполняемый файл и ввожу аргументы (рис. 34). Программа работает верно.

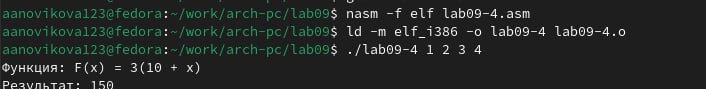


Рис. 34: Запуск программы

***ЗАДАНИЕ 2***

Создаю файл и ввожу код из листинга 9.3 (рис. 35).

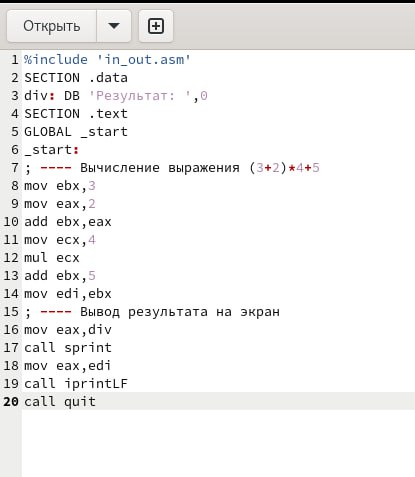


Рис. 35: Редактирование файла

Открываю файл в отладчике GDB и запускаю программу (рис. 36). Программа выдает ответ 10.

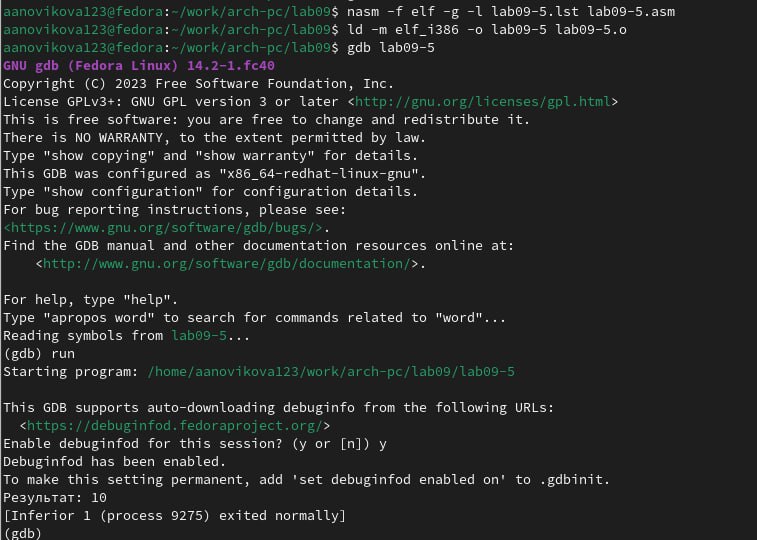


Рис. 36: Запуск программы в отладчике

Просматриваю дисассимилированный код программы, ставлю точку останова перед прибавлением 5 и открываю значения регистров на данном этапе (рис. 37).



Рис. 37: Действия в отладчике

Как можно увидеть, регистр ecx со значением 4 умножается не на ebx, сложенным c eax, а только с eax со значением 2. Значит нужно поменять значения регистров(например присвоить eax значение 3 и просто прибавит 2. После изменений программа будет выглядеть следующим образом:

%include 'in\_out.asm'  
SECTION .data  
div: DB 'Результат: ',0  
SECTION .text  
GLOBAL \_start  
\_start:  
; ---- Вычисление выражения (3+2)\*4+5  
mov eax,3  
mov ebx,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

Пробуем запустить программу (рис. 38). Она работает верно.

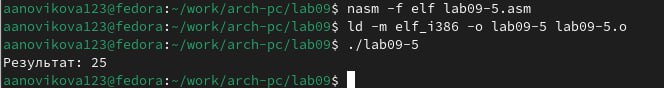


Рис. 38: Запуск программы

# 4 Выводы

В ходе лабораторной работы были приобретены навыки написания программ с использованием подпрограмм. Также было знакомство с методами отладки при помощи GDB и его основными возможностями.