Лабораторная работа №9

Понятие подпрограммы

Софич Андрей Геннадьевич

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 (рис. ??).

Создание каталога

Создание каталога

Перехожу в созданную директорию (рис. ??).

Перемещение по директории

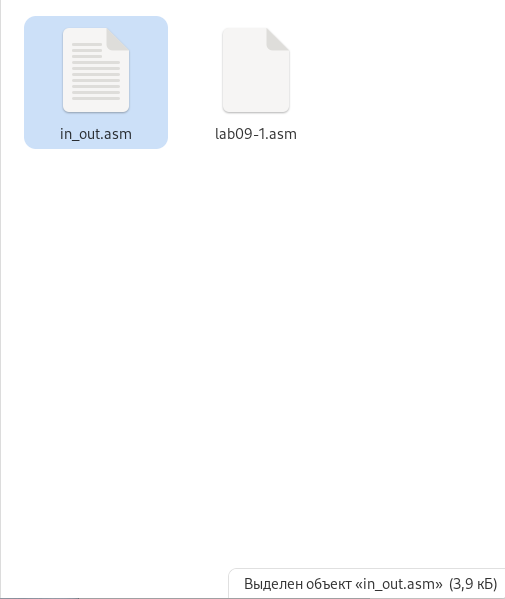
Перемещение по директории

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

Создание файла

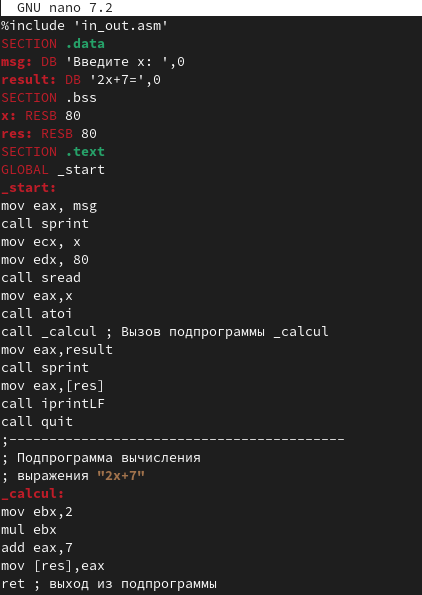
Создание файла

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



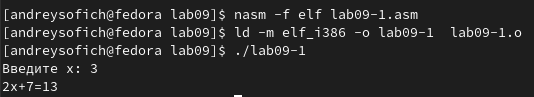
Копирование файла

Открываю файл в GNU nano и переписываю код программы из листинга 9.1 (рис. ??).



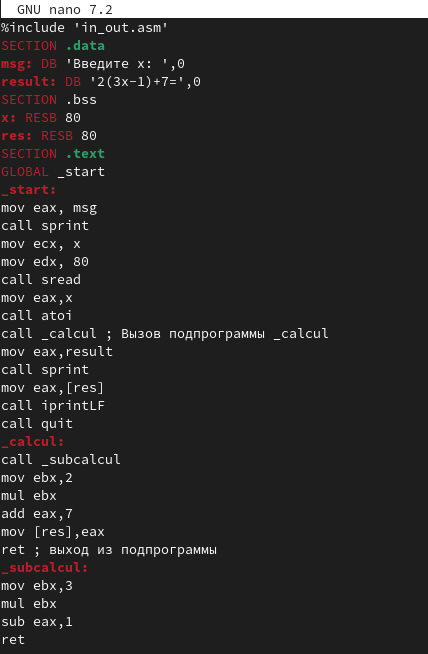
Редактирование файла

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



Запуск программы

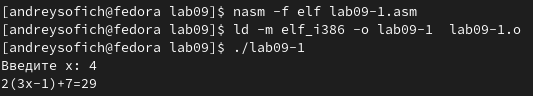
Изменяю текст файла,добавив подпрограмму sub\_calcul в подпрограмму \_calcul (рис. ??).



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. ??).Программа работает верно.



Запуск программы

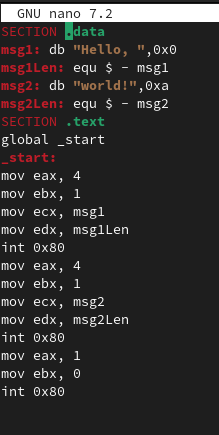
## 3.2 Отладка программ с помощью GDB

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

Создание файла

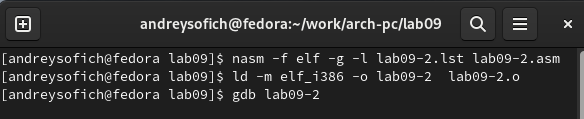
Создание файла

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



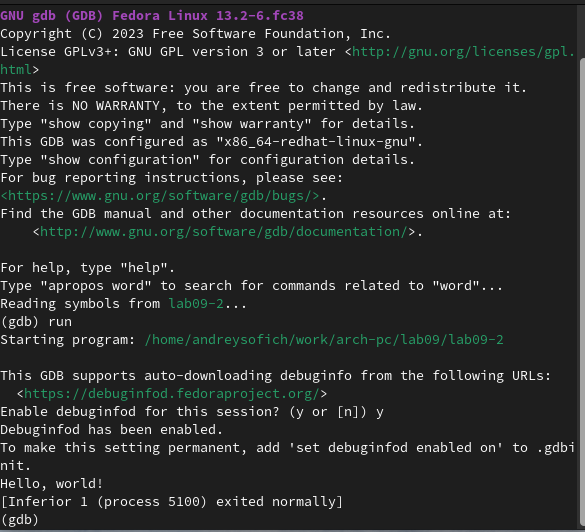
Редактирование файла

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



Запуск исполняемого файла

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



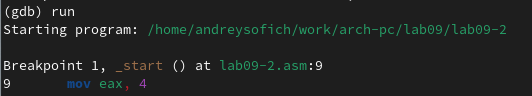
Запуск программы в отладчике

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

Установка брейкпоинта

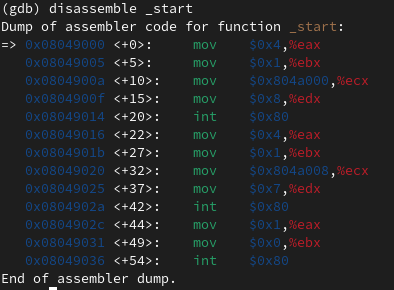
Установка брейкпоинта

Запускаю её (рис. ??).



Запуск

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



Диссассимилированный код программы

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

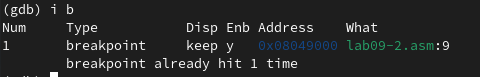


Отображение с Intel’овским синтаксисом

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. ??).



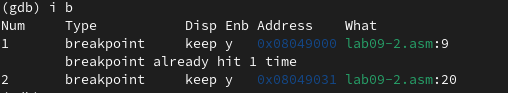
Точка останова

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

Установка точки останова

Установка точки останова

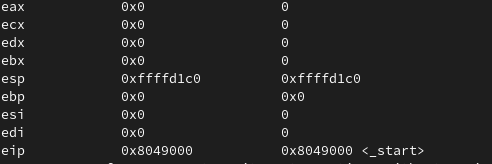
Просматриваю информацию о точках останова (рис. ??).



Точки останова

## 3.4 Работа с данными программы в GDB

Просматриваю содержимое регистров с помощью команды info register (i r) (рис. ??).



info register

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

Значение переменной по имени

Значение переменной по имени

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

Значение переменной по адресу

Значение переменной по адресу

Меняю первый символ переменной msg1 (рис. ??).

Изменение переменной

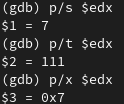
Изменение переменной

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

Изменение второй переменной

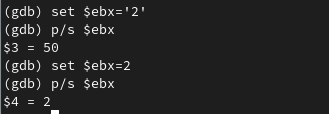
Изменение второй переменной

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



Изменение значений в разные форматы

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



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.

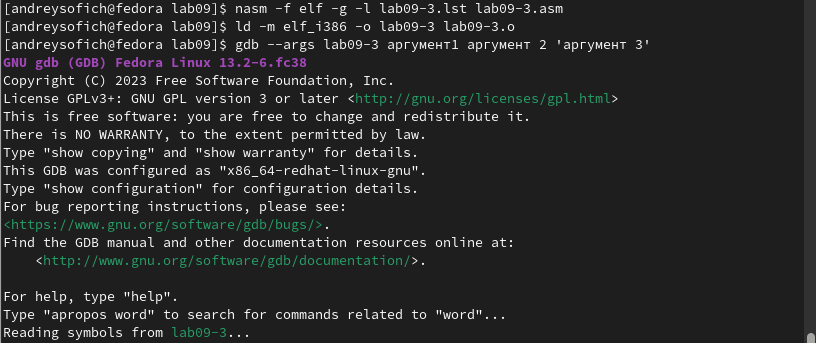
## 3.5 Обработка аргументов командной строки в GDB

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

Копирование файла

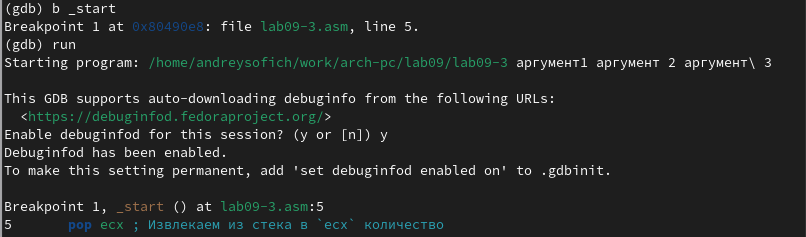
Копирование файла

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



Создание файла

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



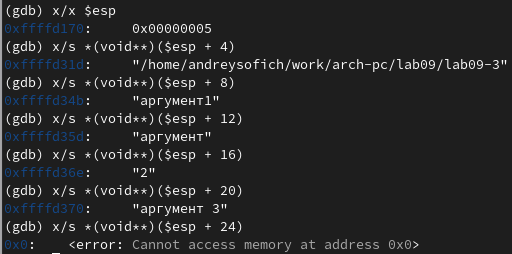
Запуск программы с точкой останова

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

Регистр esp

Регистр esp

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



Позиции стека

Количество аргументов командной строки 4,следовательно и шаг равен четырем.

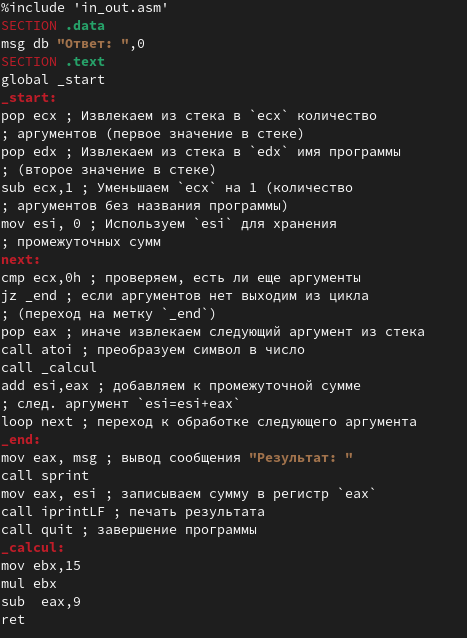
## 3.6 Задание для самостоятельной работы

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

Создание файла

Создание файла

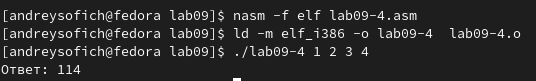
Редактирую код программы lab8-4.asm,добавив подпрограмму,которая вычисляет значения функции f(x) (рис. ??).



Редактирование файла

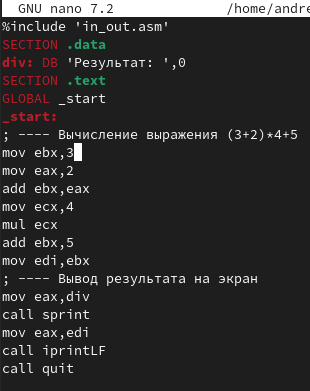
%include 'in\_out.asm'  
SECTION .data  
msg db "Ответ: ",0  
SECTION .text  
global \_start  
\_start:  
pop ecx ; Извлекаем из стека в `ecx` количество  
; аргументов (первое значение в стеке)  
pop edx ; Извлекаем из стека в `edx` имя программы  
; (второе значение в стеке)  
sub ecx,1 ; Уменьшаем `ecx` на 1 (количество  
; аргументов без названия программы)  
mov esi, 0 ; Используем `esi` для хранения  
; промежуточных сумм  
next:  
cmp ecx,0h ; проверяем, есть ли еще аргументы  
jz \_end ; если аргументов нет выходим из цикла  
; (переход на метку `\_end`)  
pop eax ; иначе извлекаем следующий аргумент из стека  
call atoi ; преобразуем символ в число  
call \_calcul  
add esi,eax ; добавляем к промежуточной сумме  
; след. аргумент `esi=esi+eax`  
loop next ; переход к обработке следующего аргумента  
\_end:  
mov eax, msg ; вывод сообщения "Результат: "  
call sprint  
mov eax, esi ; записываем сумму в регистр `eax`  
call iprintLF ; печать результата  
call quit ; завершение программы  
\_calcul:  
mov ebx,15  
mul ebx  
sub eax,9  
ret

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



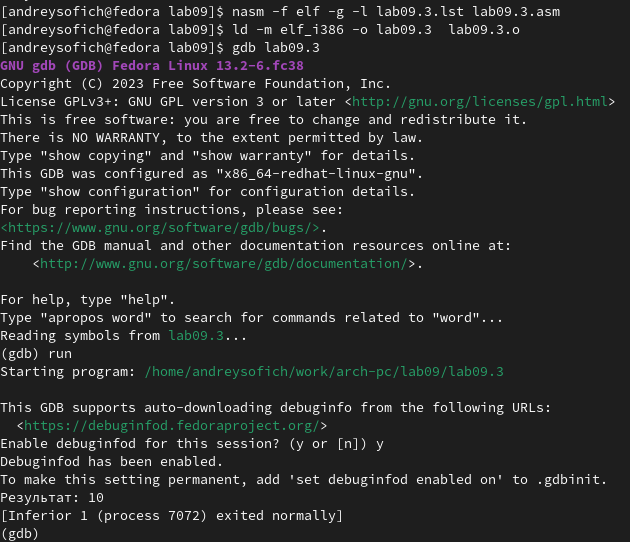
Запуск программы

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



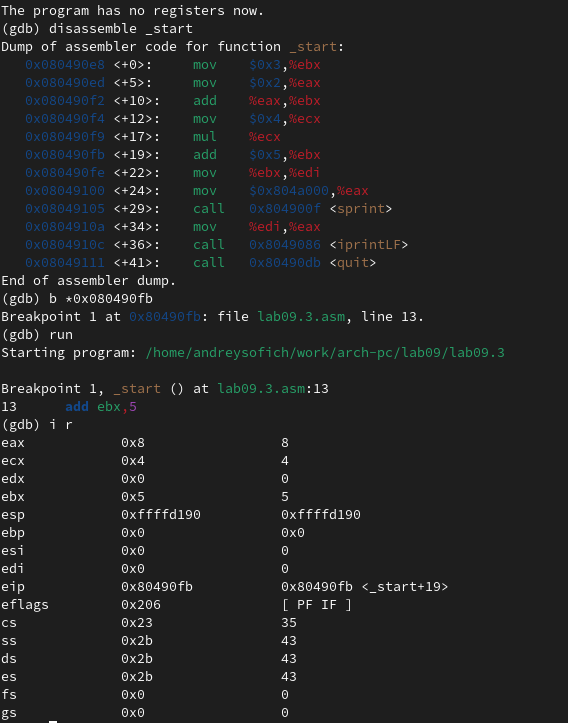
Редактирование файла

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



Запуск программы в отладчике

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

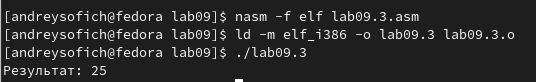


Действия в отладчике

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 выглядить 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. ??). Она работает верно.



Запуск программы

# 4 Выводы

В данной работе я приобрел навыки написания программ с подпрограммами и познакомился с методами отладки при помощи GDB.

# Список литературы

Лабораторная работа №9