Отчет по лабораторной работе №10

*дисциплина: Архитектура компьютера*

Галацан Николай, НПИбд-01-22

Содержание

# 1 Цель работы

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

# 2 Выполнение лабораторной работы

1. Ввожу команды для создания каталога лабораторной работы, перехожу в него, создаю файл lab10-1.asm

mkdir ~/work/arch-pc/lab10  
cd ~/work/arch-pc/lab10  
touch lab10-1.asm

1. Ввожу в файл lab10-1.asm текст программы из листинга 10.1, сохраняю файл. (рис. 1).

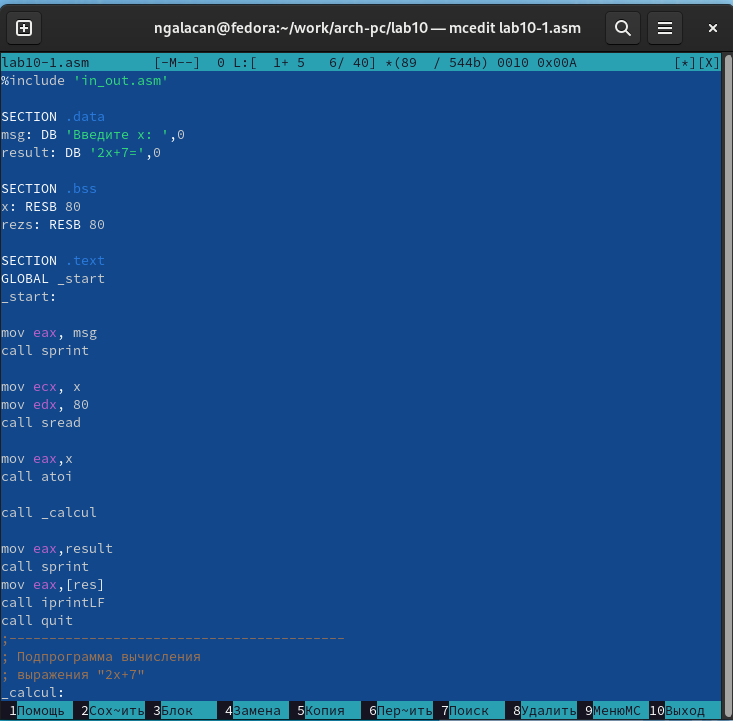


Рис. 1: Редактирование файла lab10-1.asm

Создаю исполняемый файл и запускаю его, предварительно скопировав файл in\_out.asm в соответствующий каталог. Выводится сообщение об ошибке. Исправляю опечатки в имени переменной и вновь запускаю программу. Программа работает верно (рис. 2).

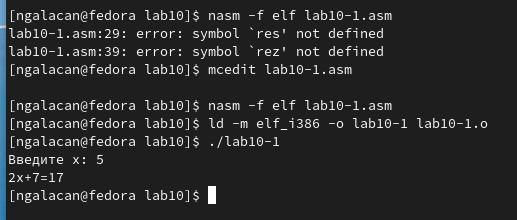


Рис. 2: Запуск исполняемого файла lab 10-1

Далее изменяю текст программы добавив подпрограмму \_subcalcul в \_calcul для вычисления выражения *f(g(x))*, где *х* вводится с клавиатуры. (рис. 3).

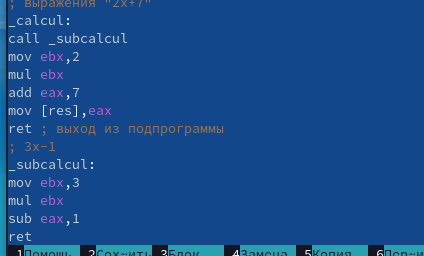


Рис. 3: Добавление подпрограммы в lab10-1.asm

Создаю исполняемый файл и запускаю его (рис. 4).

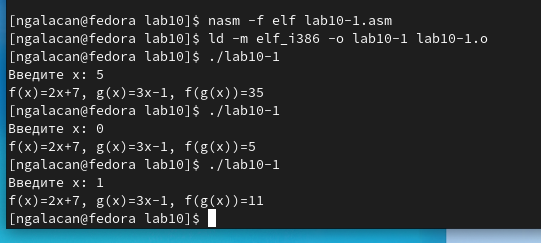


Рис. 4: Запуск измененного исполняемого файла lab 10-1

Убеждаюсь в том, что программа вычисляет верное значение.

Создаю новый файл: touch lab10-2.asm. Ввожу в него текст программы из листинга 10.2 (рис. 5).

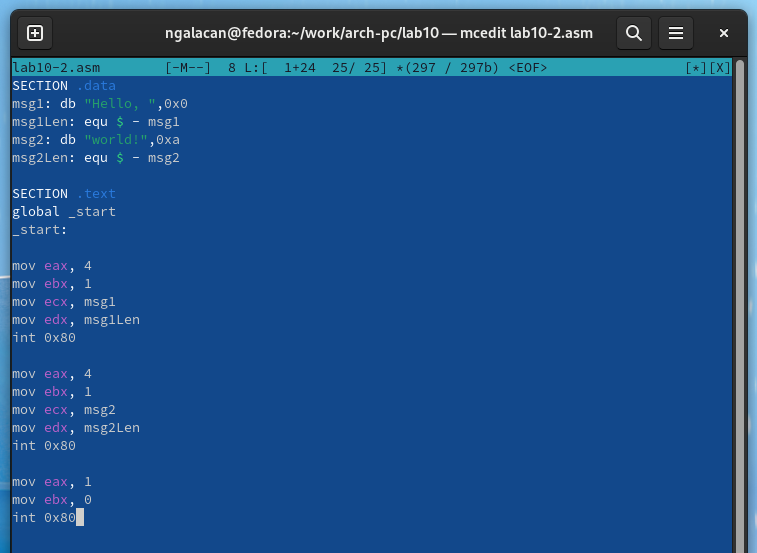


Рис. 5: Редактирование файла lab10-2.asm

Создаю исполняемый файл, с которым можно работать в GDB и загружаю в отладчик:

nasm -f elf -g -l lab10-2.lst lab10-2.asm  
ld -m elf\_i386 -o lab10-2 lab10-2.o  
gdb lab10-2

Запускаю программу в отладчике (рис. 6).

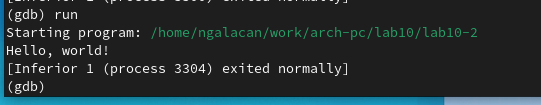


Рис. 6: Запуск исполняемого файла lab10-2 в отладчике

Устанавливаю брейкпоинт на метку \_start и запускаю программу (рис. 7).

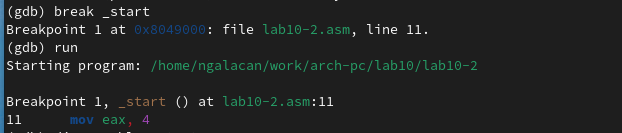


Рис. 7: Установка брейкпоинта и запуск lab10-2

Просматриваю дисассимилированный код программы начиная с \_start. Переключаюсь на отображение команд с синтаксисом Intel (рис. 8).

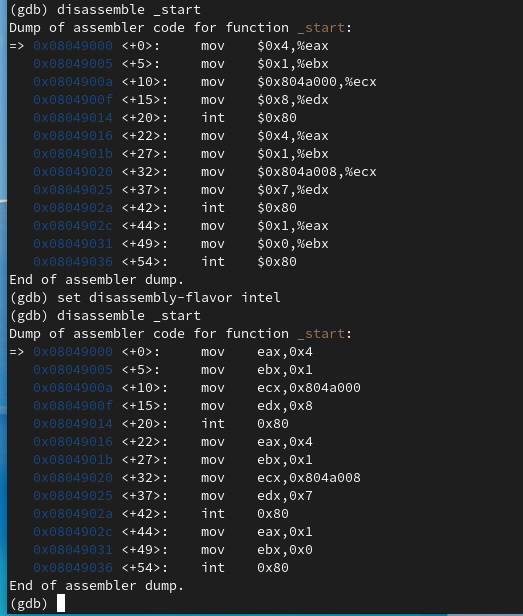


Рис. 8: Промотр дисассимилированного кода программы lab10-2

Различия отображения синтаксиса машинных команд в режимах ATT и Intel заключаются в том, что в режиме Intel регистры и их значения отображаются более привычно и визуально удобно (колонка справа). Сначала указано название регистра, через запятую - его значение. В режиме ATT наоборот: сначала $значение, после - %регистр.

Включаю режим псевографики (рис. 9).

layout asm  
layout regs

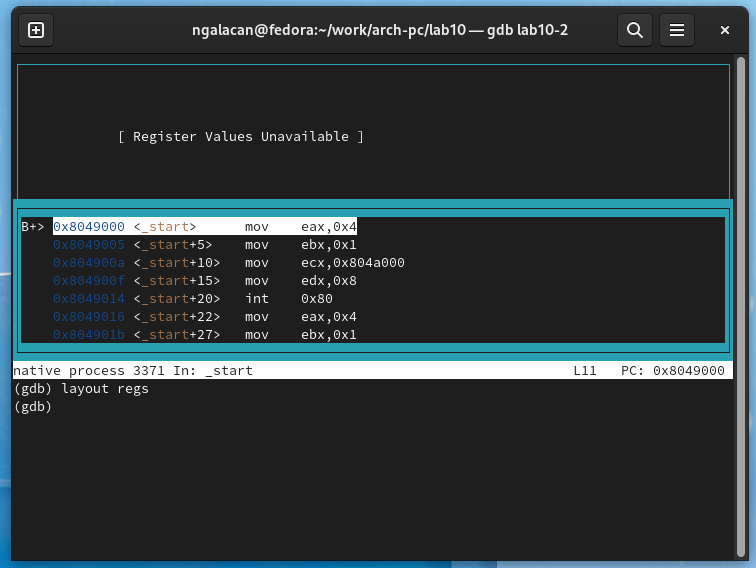


Рис. 9: Режим псевдографики

Проверяю наличие установленной точки останова. Устанавливаю еще одну по адресу инструкции, вновь запрашиваю информацию краткой командой (рис. 10)

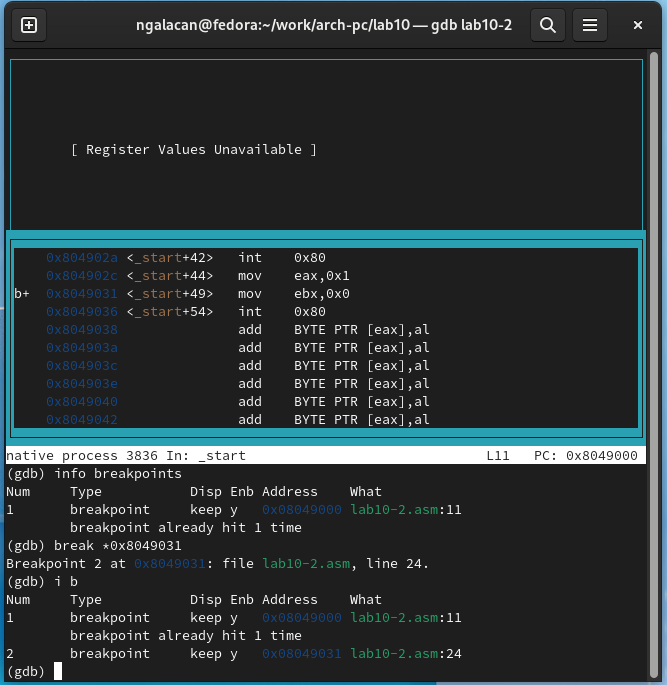


Рис. 10: Информация о точках останова. Установка новой по адресу

Выполнняю 5 инструкций stepi (рис. 11).

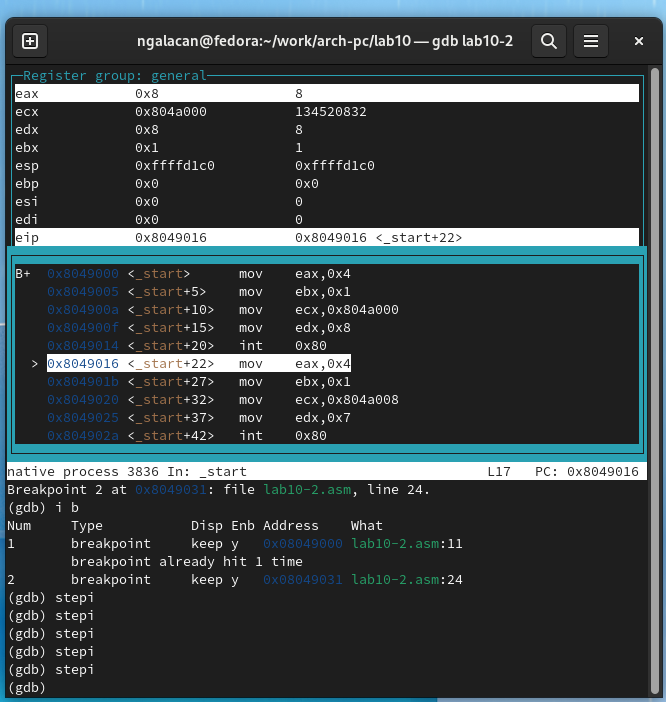


Рис. 11: Выполнение инструкций stepi

Поочередно меняются значения регистров eax, ebx, ecx, edx и снова eax. Изменения соответствуют исходному коду программы.

Просматриваю содержимое регистров, введя info registers (рис. 12).

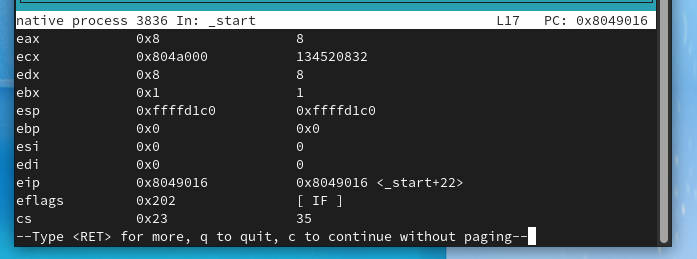


Рис. 12: Применение команды info registers

Просматриваю значение переменной msg1 по имени, msg2 - по адресу из дизассемблированной инструкции (рис. 13).

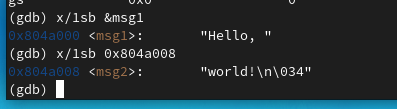


Рис. 13: Применение команды х для просмотра значений переменных

Изменяю первый символ переменной msg1, символ в переменной msg2 с помощью команды set (рис. 14).

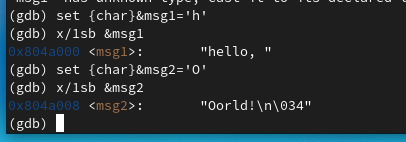


Рис. 14: Применение команды set

Вывожу значение регистра edx в шестнадцатеричном формате (p/x $edx), в двоичном формате (p/t $edx), в символьном формате (p/s $edx) (рис. 15).

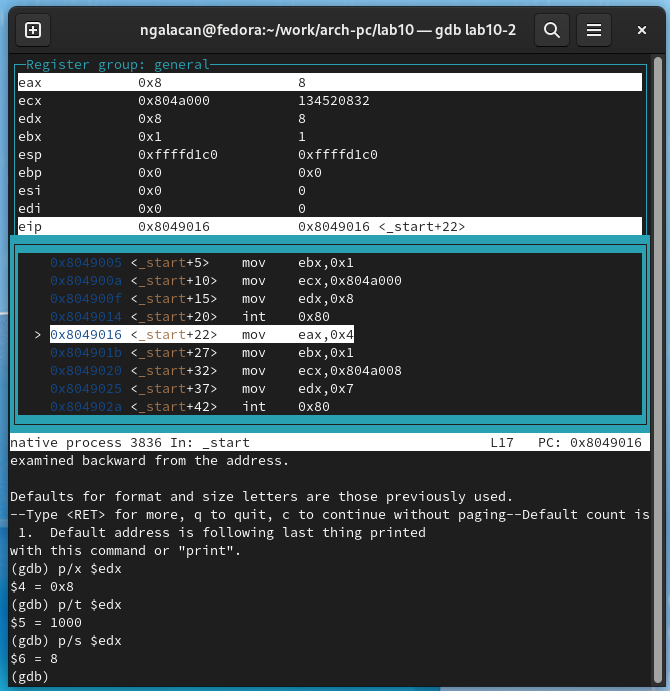


Рис. 15: Применение команды p в различных форматах

С помощью команды set меняю значение регистра ebx, вывожу значение с помощью p/s (рис. 16).

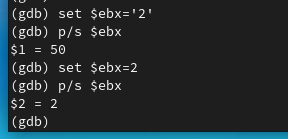


Рис. 16: Изменение значения регистра ebx. Просмотр

В первом случае был введен символ ‘2’ и в качестве значения регистра был выведен номер символа в таблице ASCII (символ 2 имеет номер 50). Во втором случае было введено и выведено само число 2.

Завершаю выполнение программы с помощью команды continue и выхожу из GDB с помощью команды quit.

Копирую файл lab9-2.asm с именем lab10-3.asm в соответствующий ЛР каталог. Создаю исполняемый файл, загружаю программу с аргументами в командной строке в отладчик, указав ключ --args (рис. 17).

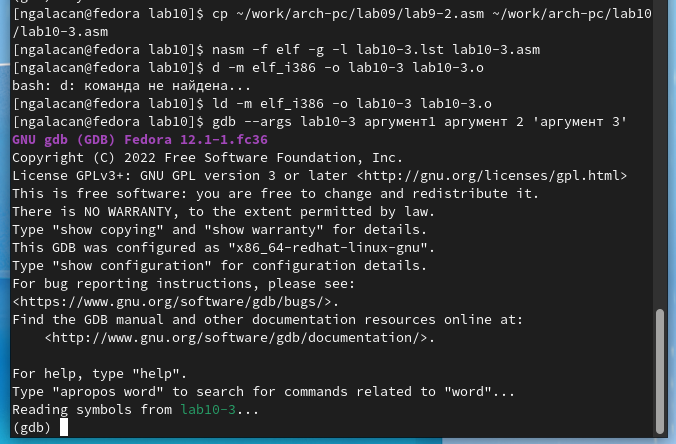


Рис. 17: Создание файла lab10-3.asm. Загрузка в отладчик

Устанавливаю точку останова перед первой инструкцией в программе, запускаю. Программа останавливается на брейкпоинте. Вывожу число аргументов (включая имя программы), которые хранятся в esp (рис. 18).

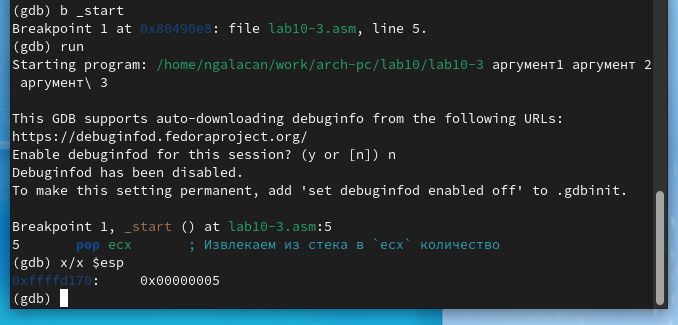


Рис. 18: Установка брейкпоинта и запуск lab10-3. Вывод количества аргументов

Просматриваю остальные позиции стека с шагом измерения адреса 4 (рис. 19).

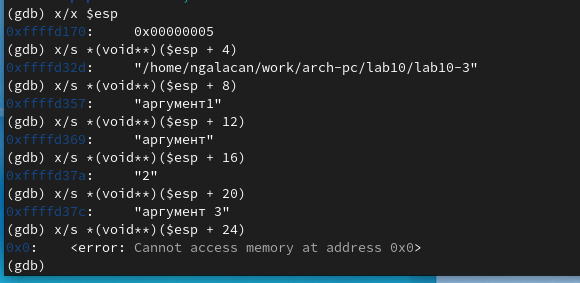


Рис. 19: Просмотр остальных позиций стека

Шаг измерения адреса равен 4, так как при добавлении в стек соответствующее значение помещается в ячейку памяти, на которую указывает регистр esp, после чего значение регистра увеличивается на 4 (т.е. “4” означает размер - 4 байта, что соответствует 32 битам).

# 3 Выполнение заданий для самостоятельной работы

1. Копирую файл lab9-sam.asm с именем lab10-sam-1.asm в соответствующий ЛР каталог. Редактирую программу, реализовав вычисление функции *f(x)=2(x-1)* в виде подпрограммы \_function. Преобразованная программа:

%include 'in\_out.asm'  
SECTION .data  
msg db "Результат: ",0  
fn db "Вариант 4. Функция: f(x)=2(x-1)."  
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 результатов  
  
next:  
cmp ecx,0h  
jz \_end  
  
pop eax ; eax=x  
call atoi  
call \_function  
add esi,eax  
loop next  
  
\_end:  
mov eax,fn  
call sprintLF  
mov eax,msg  
call sprint  
mov eax,esi  
call iprintLF  
call quit  
  
\_function:  
sub eax,1 ; eax=x-1  
mov ebx,2  
mul ebx ; eax=(x-1)\*2  
ret

Внутри цикла после проверки условия извлекается значение из стека, которое преобразуется в целое число. После этого вызывается функция с помощью call. Вычисляется значение функции, которое остается в eax и передается управление программе в том же месте, где был осуществлен вызов функции.

Создаю исполняемый файл и запускаю, введя такие же наборы *х*, как в ЛР №9. Убеждаюсь, что результат совпадает. Следовательно преобразованная программа работает правильно (рис. 20).

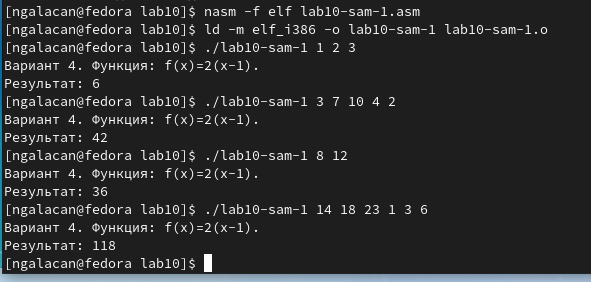


Рис. 20: Запуск исполняемого файла lab10-sam-1

1. Создаю файл lab10-sam-2.asm и ввожу в него программу из листинга 10.3. Создаю исполняемый файл, запускаю. Программа выводит неверный результат (рис. 21).

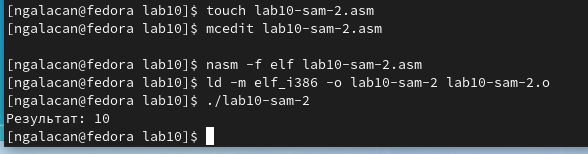


Рис. 21: Запуск исполняемого файла lab10-sam-2

Создаю исполняемый файл, оттранслировав с ключом -g для работы в GDB и загружаю в отладчик (рис. 22).

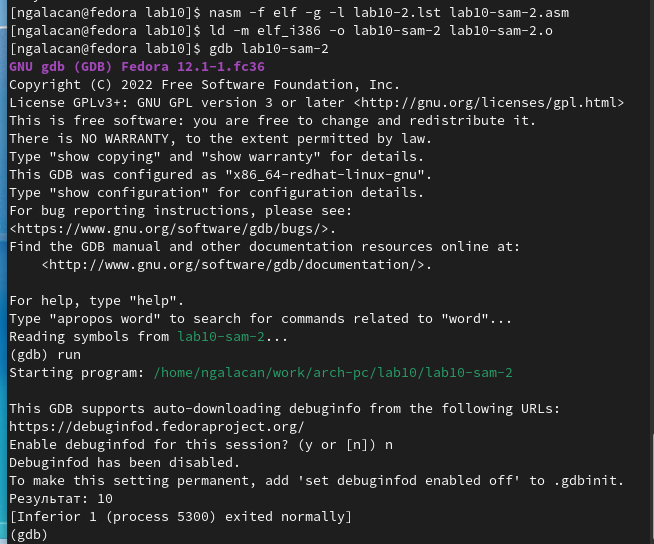


Рис. 22: Загрузка в отладчик lab10-sam-2

Просматриваю дисассимилированный код программы в синтаксисе Intel. Включаю режим псевдографики. Проставляю брейкпоинт на метку \_start и запрашиваю информацию о брейкпоинтах. Запускаю программу, ввожу si для выполнения следующего шага. В регистр ebx помещается сумма значений ebx (3) и eax (2). Значение ebx=5. (рис. 23).

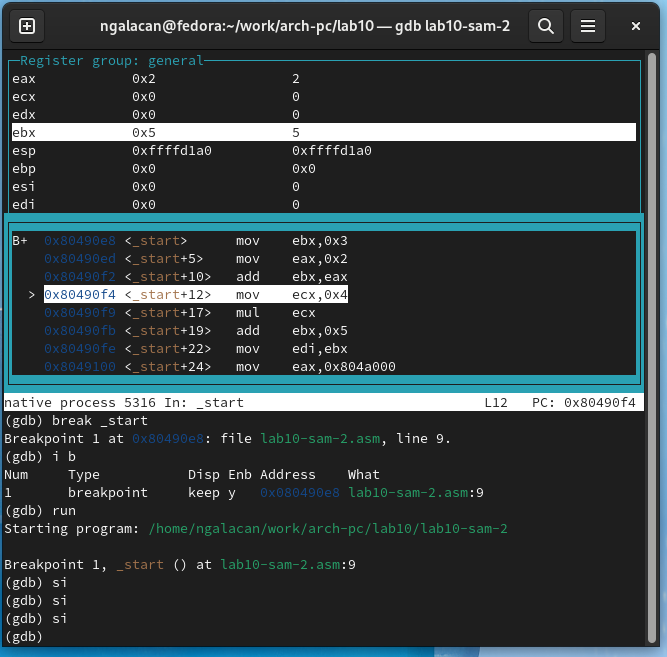


Рис. 23: Установка брейкпоинта в lab10-sam-2 и просмотр. Запуск программы

Выполняю следующие шаги. В ecx помещается 4, выполняется умножение. Результат умножения сохраняется в eax (т.е. 2\*4=8) (рис. 24).

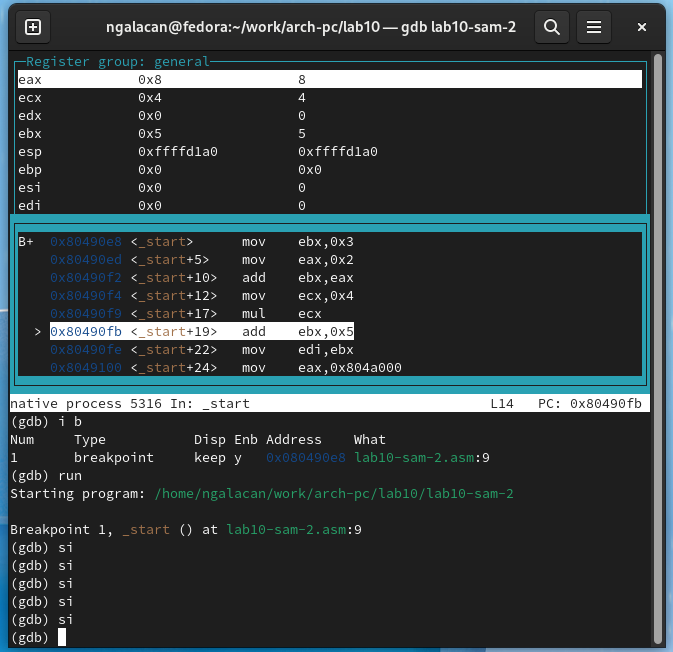


Рис. 24: Отслеживание значений регистров при вычислении выражения (1)

После этого к регистру ebx прибавляется 5, регистр принимает значение 10. Это значение помещается в edi в качестве результата. То есть, произведение, которое сохранилось в eax при вычислении выражение учтено не было, из-за чего выводится неверный результат (рис. 25).

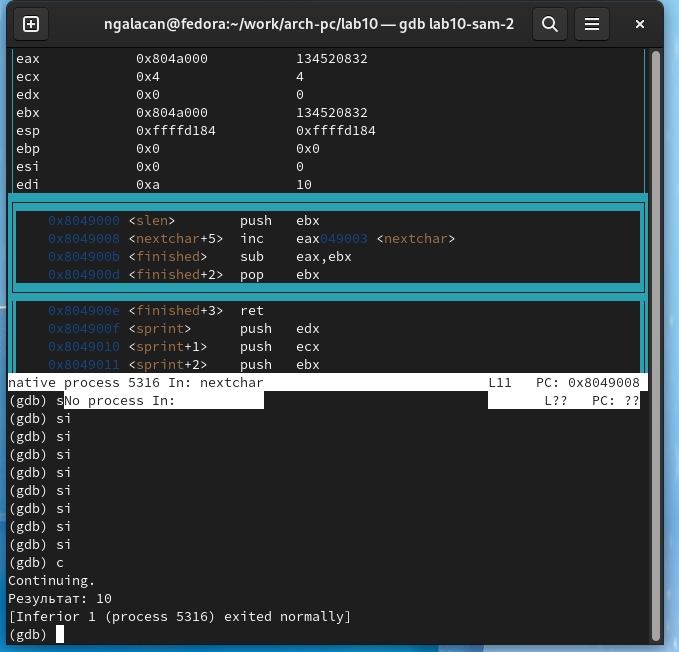


Рис. 25: Отслеживание значений регистров при вычислении выражения (2)

Выхожу из отладчика и вношу изменения в текст программы, чтобы промежуточные результаты сохранялись в регистре eax. Измененная программа:

%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

Аналогично, как описано выше, загружаю программу в отладчик и пошагово отслеживаю значения регистров. Теперь результат сложения в скобках сохраняется в регистр eax. Эта сумма умножается на 4, что выше было помещено в ecx. Произведение сохраняется в eax. После этого к произведению прибавляется 5 (рис. 26).

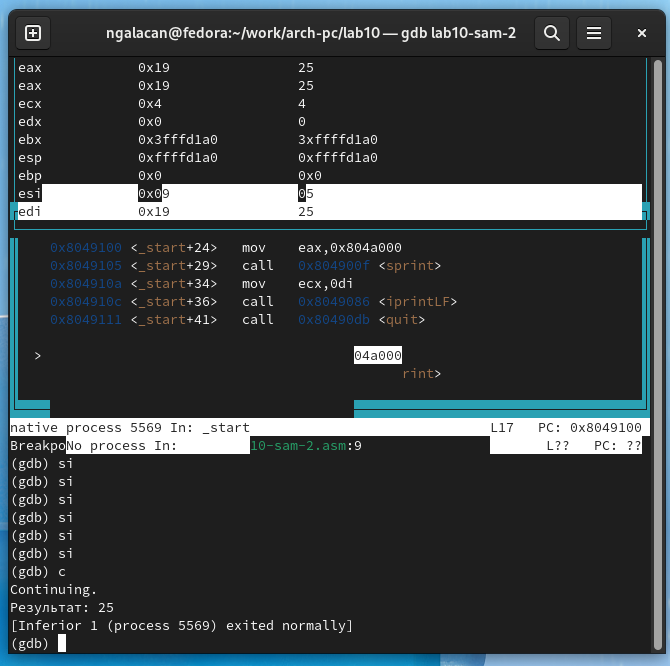


Рис. 26: Отладка измененной программы lab10-sam-2, вывод результата

В процессе первой отладки были выявлены ошибки, которые были исправлены. В процессе второй отладки программа была проверена на правильность работы. Теперь программа выводит верный результат.

# 4 Выводы

Приобретены навыки написания программ с использованием подпрограмм. Были изучены методы отладки при помощи GDB и его возможности.