Отчёт по лабораторной работе №9

дисциплина: Архитектура компьютера

Маслова Анна Павловна

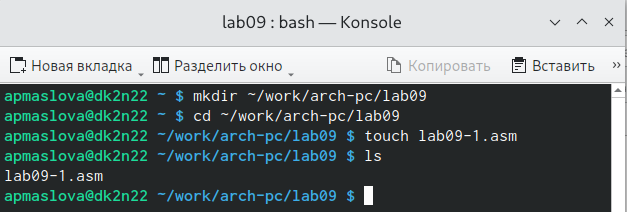
Содержание

# 1 Цель работы

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

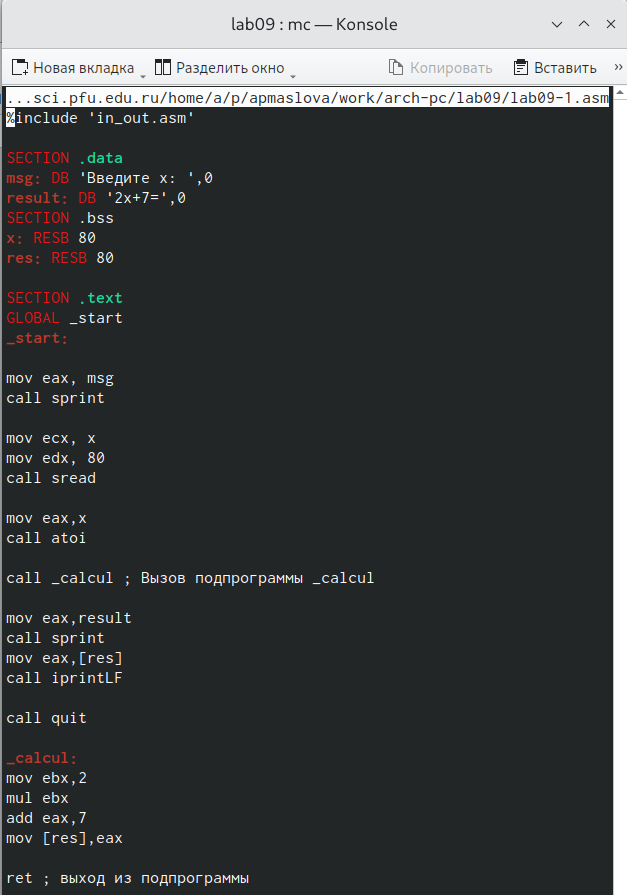
# 2 Выполнение лабораторной работы

Создадим каталог для выполнения лабораторной работы №9, перейдём в него и создадим файл lab09-1.asm. (рис. ??)



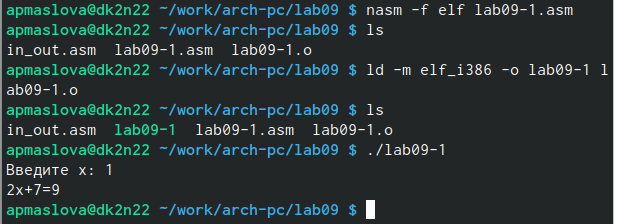
Создание каталога lab09 и файла lab09-1.asm

Рассмотрим программу вычисления арифметического выражения . Введём в файл lab09-1.asm текст этой программы. (рис. ??)



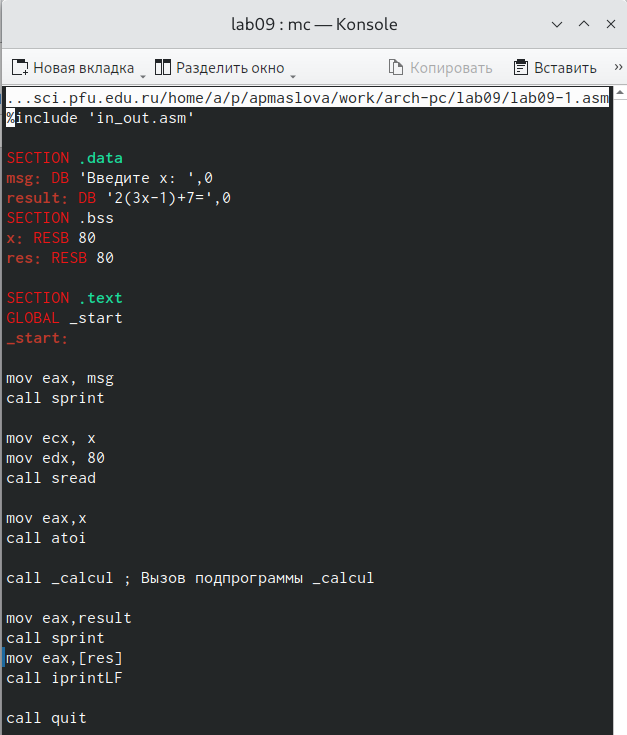
Текст файла lab09-1.asm

Создадим исполняемый файл и проверим его работу. (рис. ??)

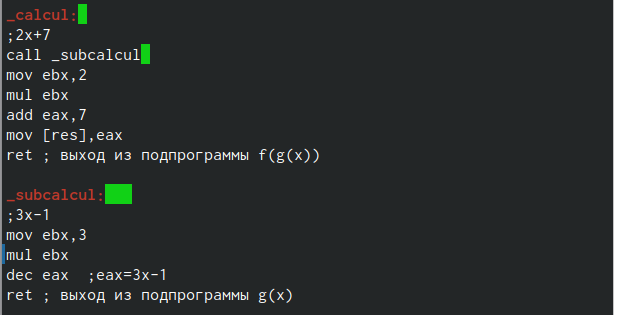


Создание и запуск исполняемого файла lab09-1

Изменим текст программы, добавив подпрограмму \_subcalcul в подпрограмму \_calcul, для вычисления выражения f(g(x)), где x вводится с клавиатуры, f(x) = 2x + 7, g(x) = 3x − 1. Результат возвращается в основную программу для вывода результата на экран. (рис. ?? , рис. ??)

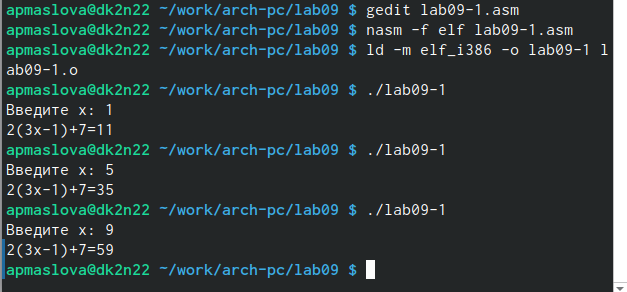


Изменённый текст файла lab09-1.asm



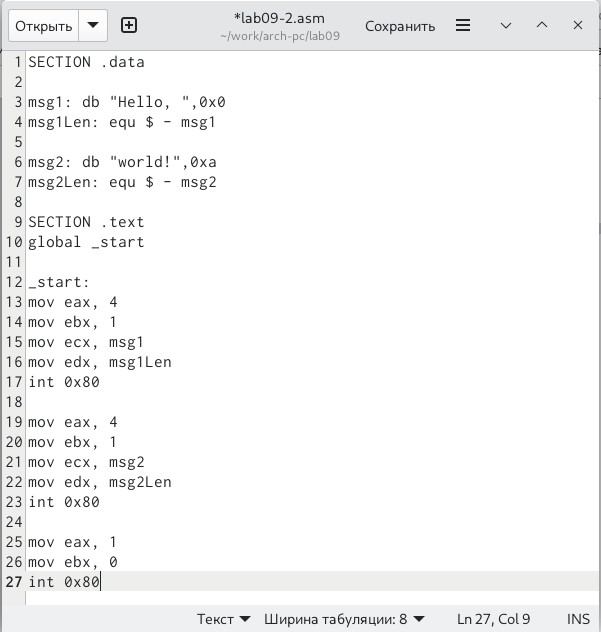
Изменённый текст файла lab09-1.asm

Создадим исполняемый файл и проверим его работу. (рис. ??)



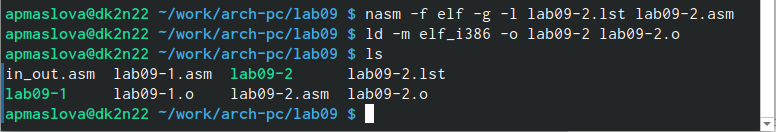
Создание и запуск изменённого исполняемого файла lab09-1

Создадим файл lab09-2.asm с текстом программы печати сообщения *Hello world!* (рис. ??)



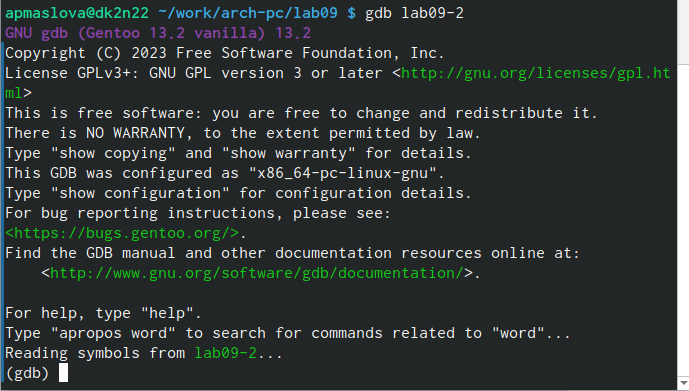
Текст файла lab09-2.asm

Получим исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом -g. (рис. ??)



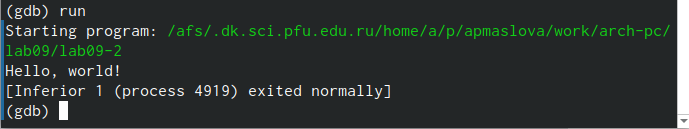
Создание исполняемого файла lab09-2 и файла листинга lab09-2.lst

Загрузим исполняемый файл в отладчик gdb. (рис. ??)



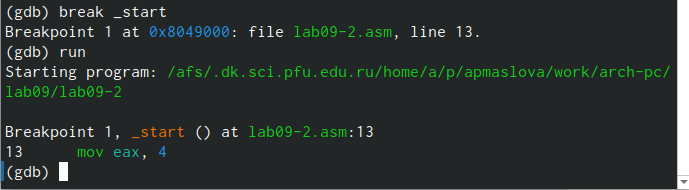
Отладчик gdb

Проверим работу программы, запустив ее в оболочке GDB с помощью команды run (рис. ??).



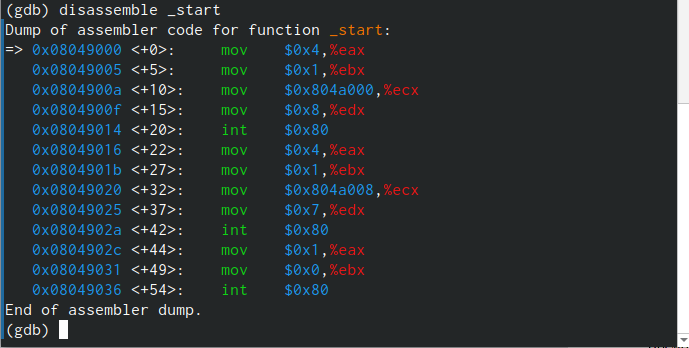
Запуск программы с помощью команды run

Для более подробного анализа программы установим брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запустим её. (рис. ??)



Установка брейкпоинта на метку \_start

Посмотрим дисассимилированный код программы с помощью команды disassemble начиная с метки \_start. (рис. ??)



Дисассимилированный код программы lab09-2

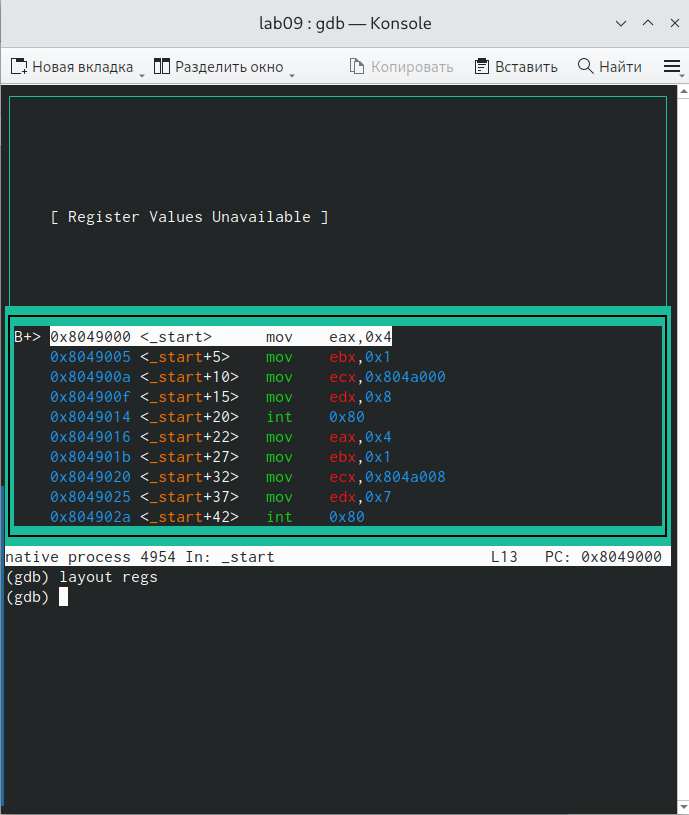
Переключимся на отображение команд с Intel’овским синтаксисом (рис. ??)



Синтаксис Intel

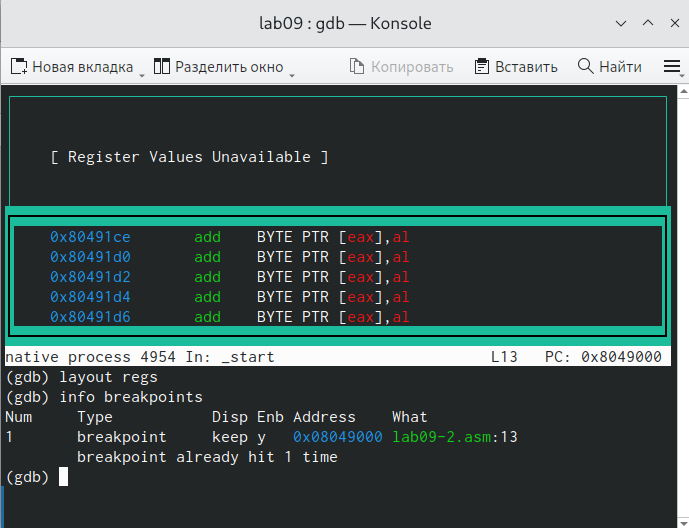
Как мы видим, в синтаксисе Intel после команды выводится регистр, куда помещается значение, а затем адрес помещемого значения или число. А в синтаксисе ATTсначала выводится ссылка на заносимое в регистр значение (для этого перед значением есть символ $), а уже после неё регистр с символом %.

Включим режим псевдографики для более удобного анализа программы (рис. ??)



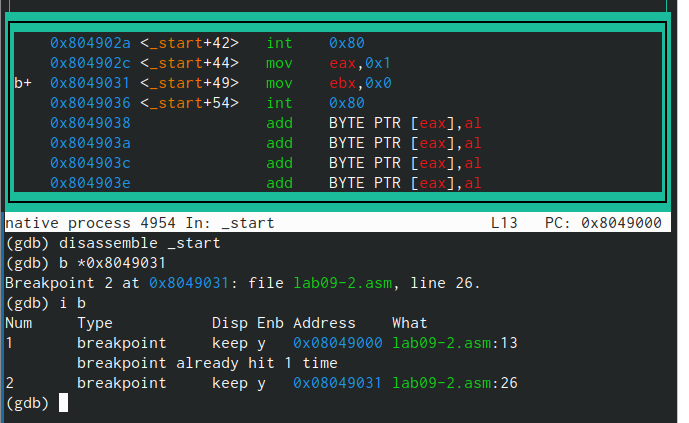
Режим псевдографики

На предыдущих шагах была установлена точка останова по имени метки \_start. Проверим это с помощью команды info breakpoints (рис. ??)



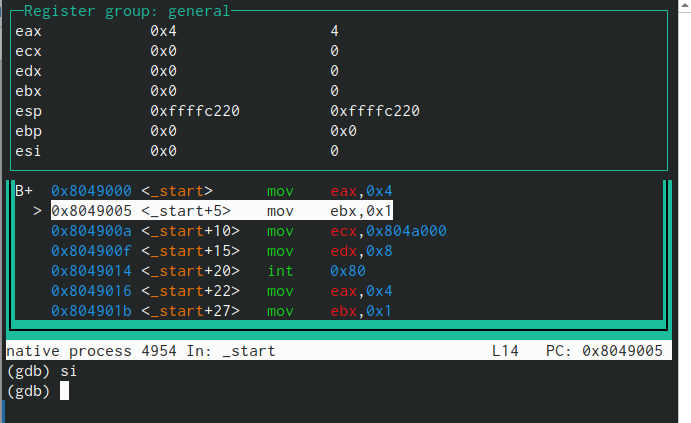
Проверка точек останова

Установим еще одну точку останова по адресу инструкции mov ebx,0x0. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции. В нашем случае адрес следующий: 0x8049031. После этого посмотрим информацию о всех точках останова (рис. ??).

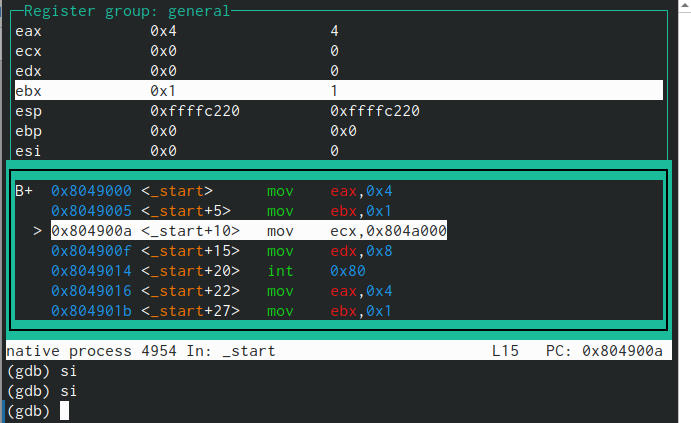


Установка точки останова по адресу

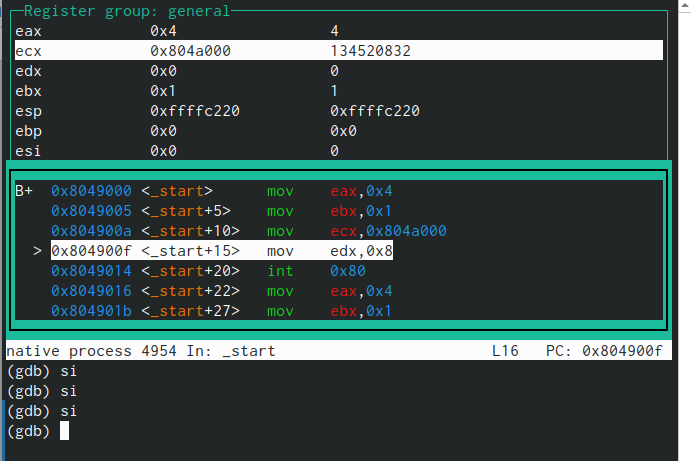
Выполним 5 инструкций с помощью команды stepi (или si) и проследим за изменением значений регистров (рис. ?? - ??).



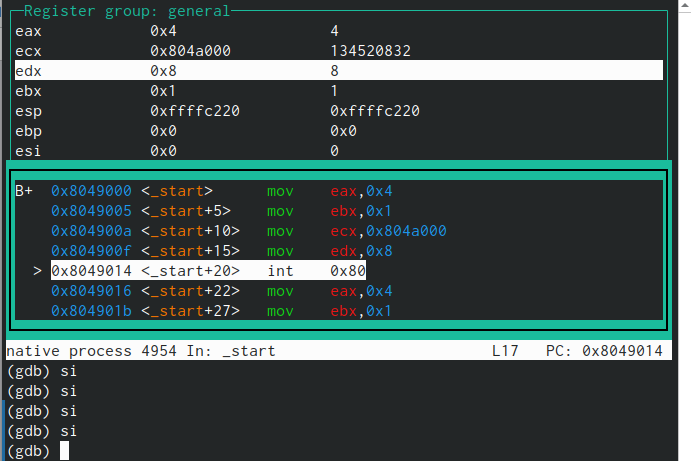
Инструкция stepi



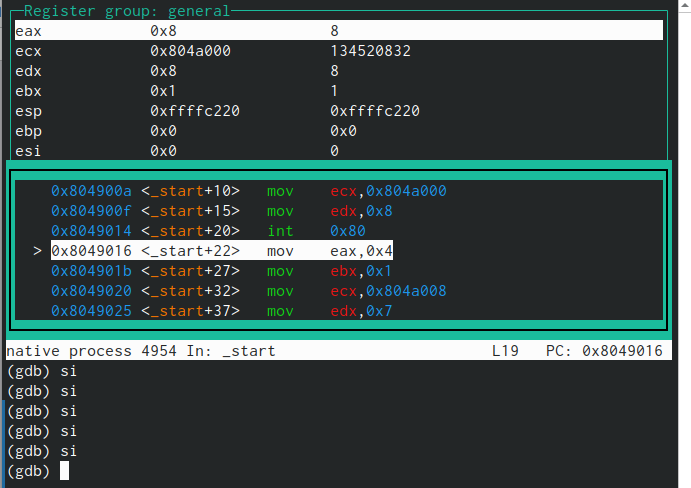
Инструкция stepi



Инструкция stepi



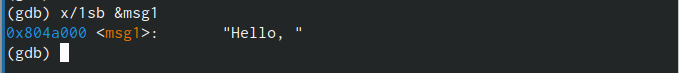
Инструкция stepi



Инструкция stepi

Как мы видим на верхней панели, изменились значения регистров eax, ecx, edx и ebx. Остальные регистры остаются без изменений.

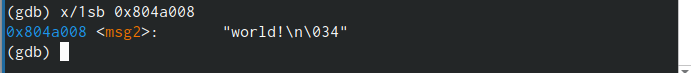
Теперь посмотрим значение переменной msg1 по имени (рис. ??).



Значение переменной msg1 по имени

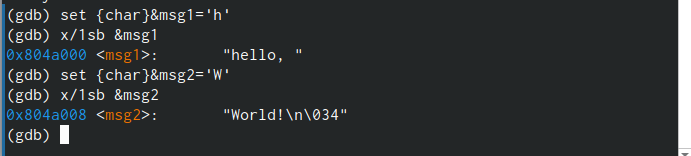
Вывелась строка “Hello,”.

Теперь посмотрим значение переменной msg2 по адресу. Адрес переменной определим по дизассемблированной инструкции. Посмотрим инструкцию mov ecx,msg2 которая записывает в регистр ecx адрес перемененной msg2(рис. ??).



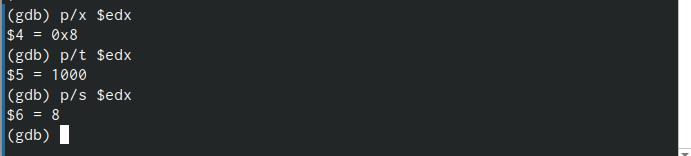
Значение переменной msg2 по адресу

Изменим первый символ переменной msg1 и первый символ переменной msg2(рис. ??).



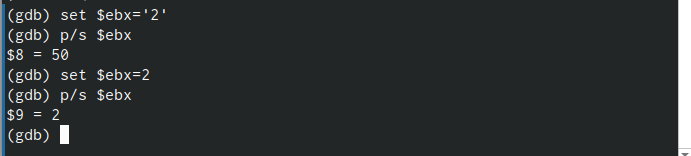
Изменение символов переменных msg1 и msg2

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



Вывод значения регистра edx

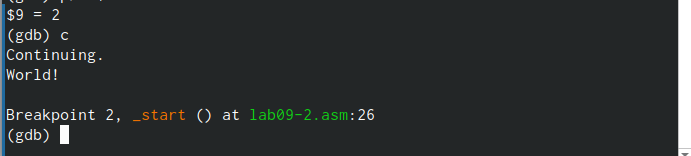
Теперь с помощью команды set изменим значение регистра ebx (рис. ??).



Изменение значения регистра edx

В первом случае мы занесли в регистр ebx символ ‘2’, поэтому после запроса p/s на вывод значения регистра на экране мы увидели код символа “2”, а именно - 50. А в случае, когда в регистр изначально было занесено число 2, а не символ, команда p/s $ebx вывела значение 2.

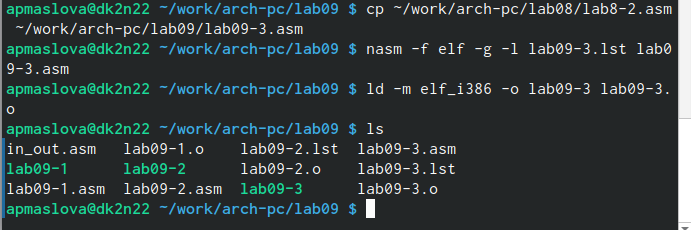
Завершим выполнение программы с помощью команды continue (сокращенно c) (рис. ??).



Завершение выполнения программы

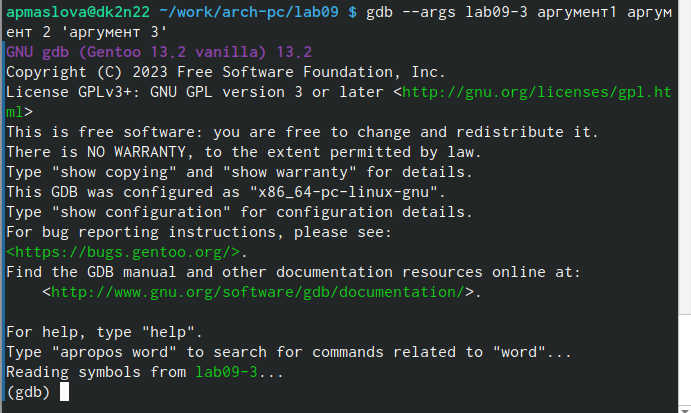
И после этого с помощью команды quit вышли из отладчика gdb.

Скопируем 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 и создадим исполняемый файл с ключом -g и с файлом листинга (рис. ??).



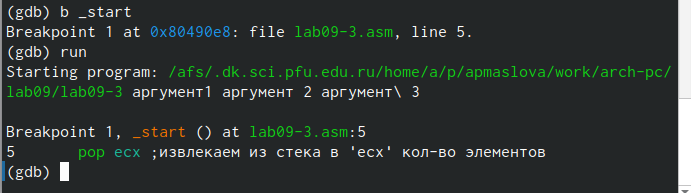
Создание исполняемого файла lab09-3

Для загрузки в gdb программы с аргументами необходимо использовать ключ --args. Загрузим исполняемый файл в отладчик, указав аргументы (рис. ??).



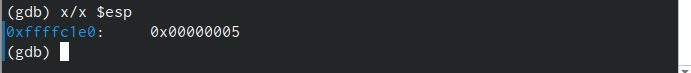
Загрузка программы lab09-3 в gdb

Для начала установим точку останова перед первой инструкцией в программе и запустим её (рис. ??).



Установка точки останова перед первой инструкцией и запуск программы

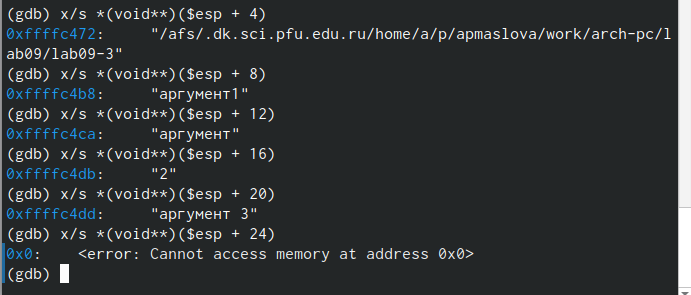
Адрес вершины стека храниться в регистре esp и по этому адресу располагается число, равное количеству аргументов командной строки (включая имя программы) (рис. ??).



Содержимое регистра esp

Как видно, число аргументов равно 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и аргумент 3.

Посмотрим остальные позиции стека (рис. ??).



Содержимое стека

По адесу [esp+4] располагается адрес в памяти, где находиться имя программы, по адесу [esp+8] храниться адрес первого аргумента, по аресу [esp+12] – второго и т.д. Как мы видим, шаг изменения равен 4. Шаг имеет такое значение, потому что при добавлении значения каждого аргумента в стек значение регистра esp увеличивается на 4.

# 3 Выполнение заданий для самостоятельной работы

**Задание №1**

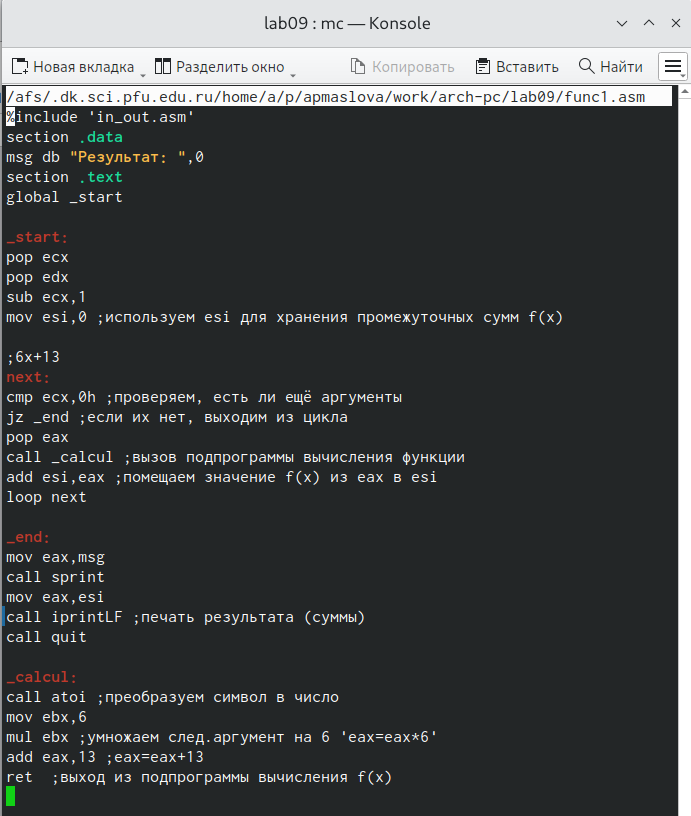
Преобразуем программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции как подпрограмму.

В лабораторной работе №8 мы реализовывали вычисление в цикле следующей функции:

Теперь для вычисления значения этой функции в цикле мы будем вызывать вспомогательную подпрограмму. Создадим файл func1.asm и впишем в него текст программы из листинга 9.4 (рис. ??).

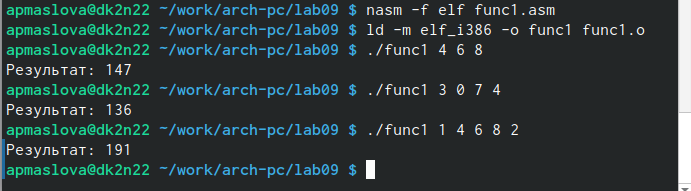
**Листинг 9.4. Программа вычисления суммы значений функции с помощью подпрограммы**

%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 ;используем esi для хранения промежуточных сумм f(x)  
  
;6x+13  
next:  
cmp ecx,0h ;проверяем, есть ли ещё аргументы  
jz \_end ;если их нет, выходим из цикла  
pop eax  
call \_calcul ;вызов подпрограммы вычисления функции  
add esi,eax ;помещаем значение f(x) из eax в esi  
loop next  
  
\_end:  
mov eax,msg  
call sprint  
mov eax,esi  
call iprintLF ;печать результата (суммы)  
call quit  
  
\_calcul:  
call atoi ;преобразуем символ в число  
mov ebx,6  
mul ebx; умножаем след.аргумент на 6 'eax=eax\*6'  
add eax,13 ;eax=eax+13  
ret ;выход из подпрограммы вычисления f(x)



Текст файла func1.asm

Создадим исполняемый файл и проверим его работу с теми же аргументами (рис. ??).

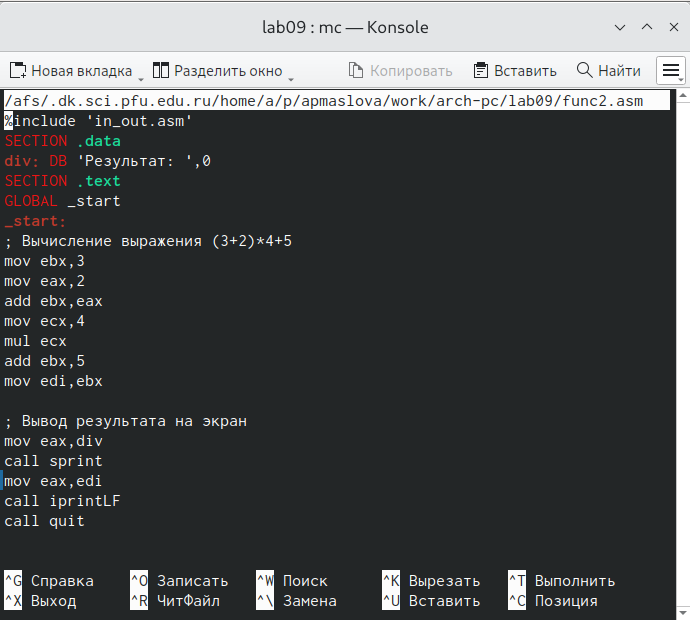


Создание и запуск исполняемого файла func1

Программа работает корректно: выдаются верные значения сумм. Приступим к выполнению следующего задания.

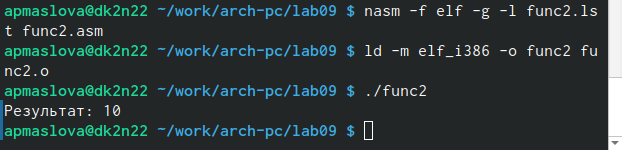
**Задание №2**

Создадим файл func2.asm и впишем в него текст программы вычисления выражения (рис. ??).



Текст файла func2.asm

Создадим исполняемый файл и запустим его (рис. ??).

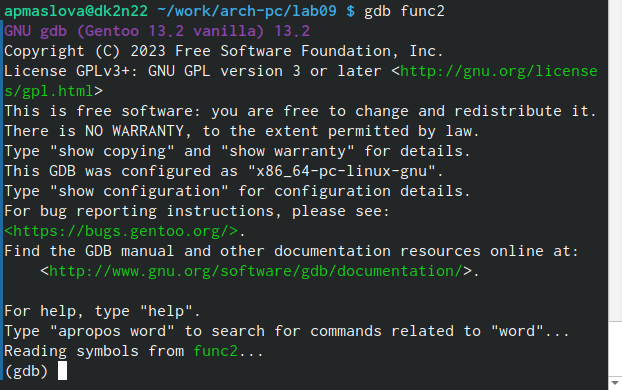


Создание и запуск исполняемого файла func2

Мы проверили, и действительно, при запуске программа даёт неверный результат.

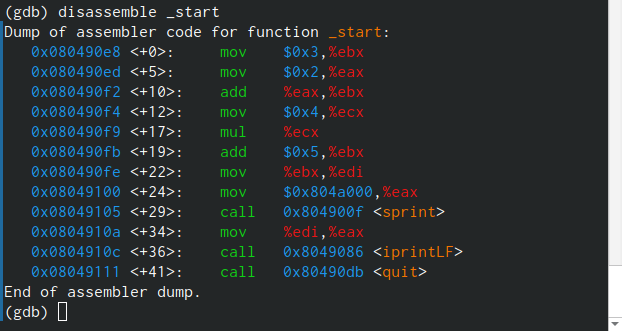
С помощью отладчика GDB проанализируем изменения значений регистров, определим ошибку и исправим её.

Для начала загрузим исполняемый файл в отладчик (рис. ??).



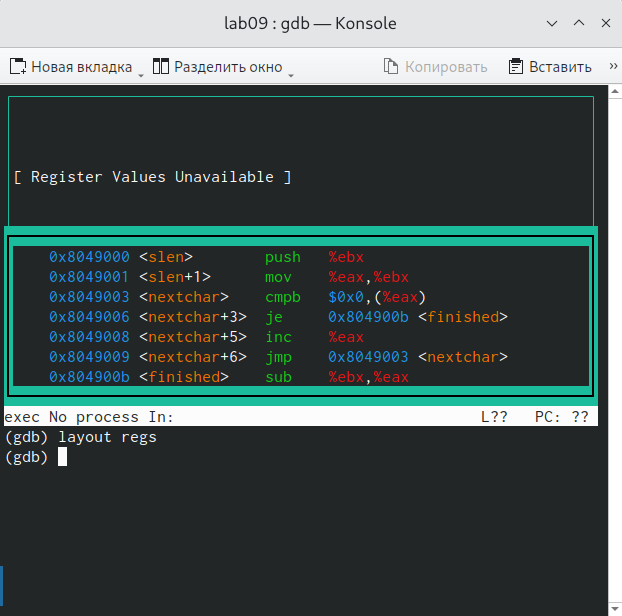
Закрузка файла func2 в gdb

Посмотрим дисассимилированный код программы с помощью команды disassemble, начиная с метки \_start (рис. ??).



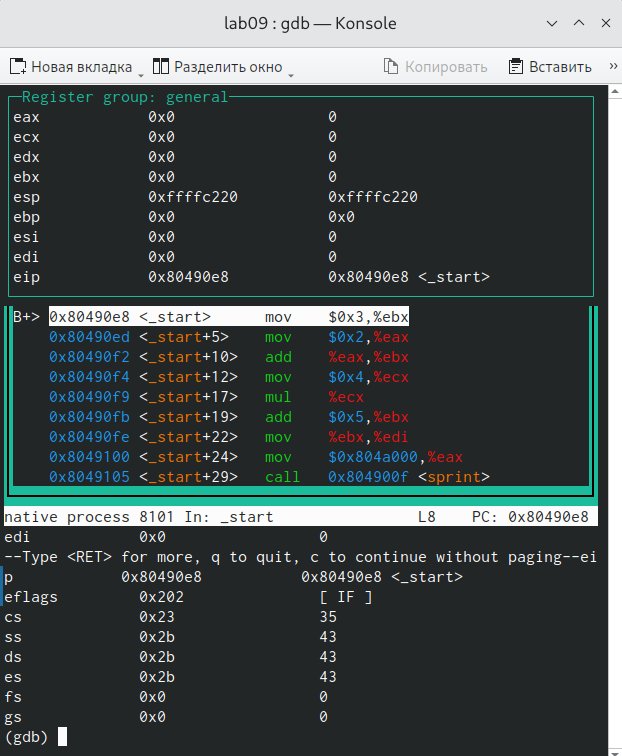
Дисассимилированный код программы func2

Включим режим псевдографики (рис. ??).



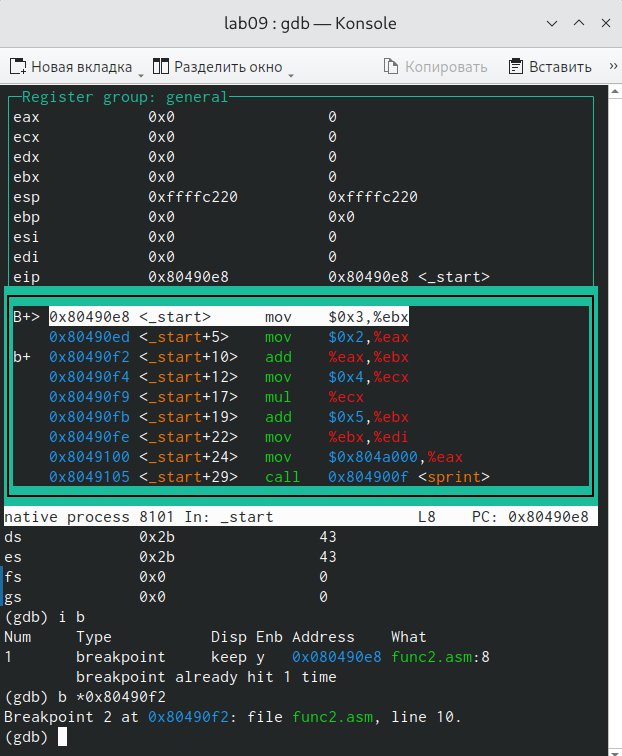
Режим псевдографики для func2

Используем команду для просмотра значений регистров (рис. ??).



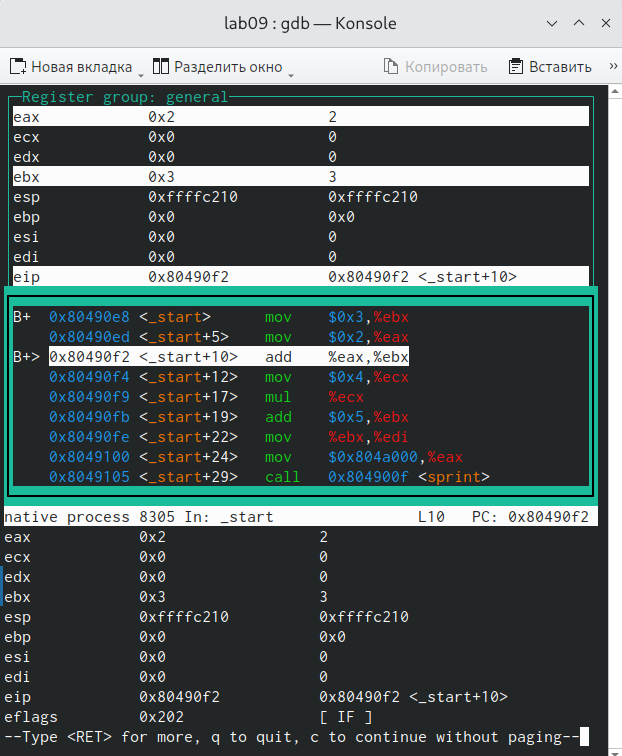
Просмотр значений регистров в func2

С помощью команды break установим точку останова по адресу на инструкции add ebx,eax (рис. ??).



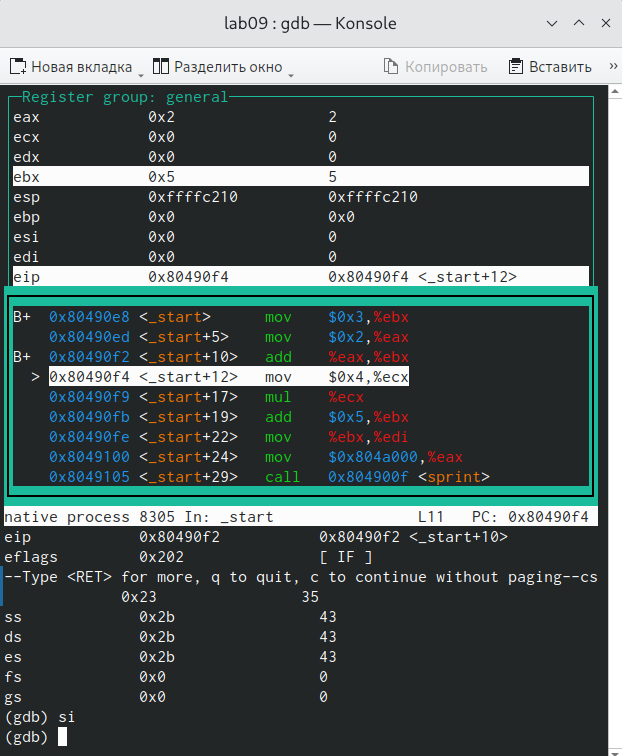
Точка останова на инструкции add

Посмотрим значения регистров на этом этапе с помощью команды i r (рис. ??).



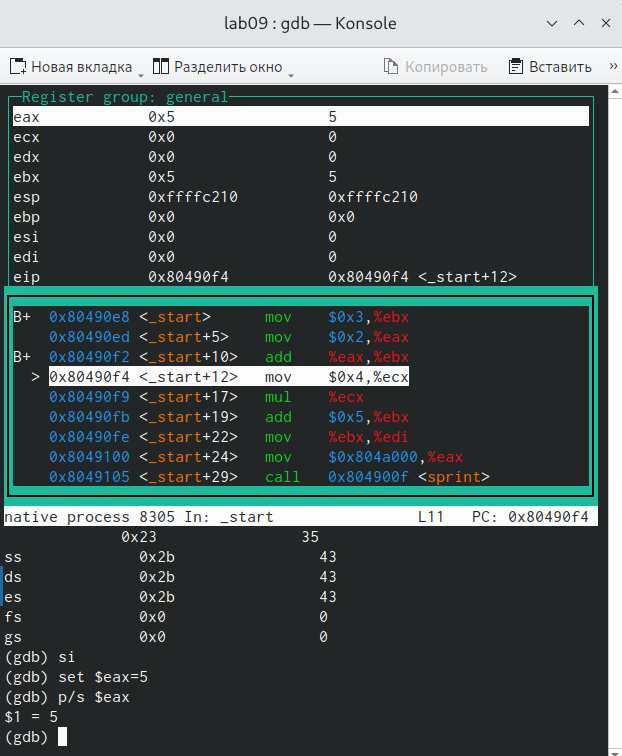
Посмотр значений регистров

Далее с помощью команды si перейдём к следующей инструкции и проследим за изменением значений регистров (рис. ??).



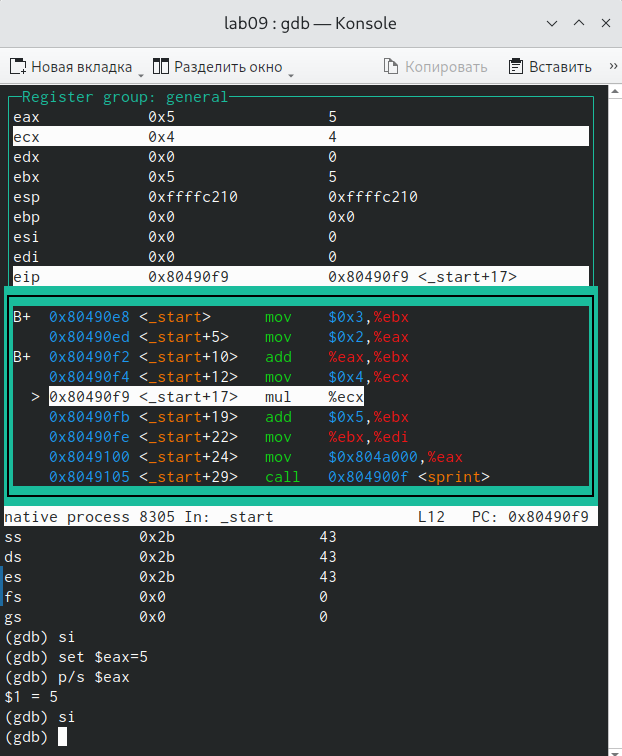
Команда si

Как мы видим, результат суммы чисел и записался в регистр ebx. Это могло послужить проблемой для дальнейшего вычисления произведения. Исправим это, изменив значение регистра eax и занеся в него значение с помощью команды set (рис. ??).



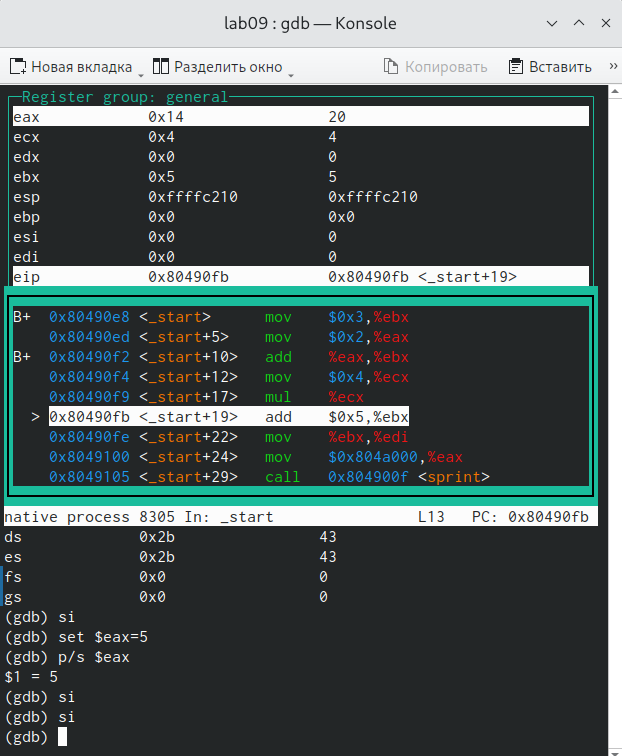
Изменение значения регистра eax с помощью команды set

Переходим к следующей инструкции (рис. ??).



Переход на следующий шаг

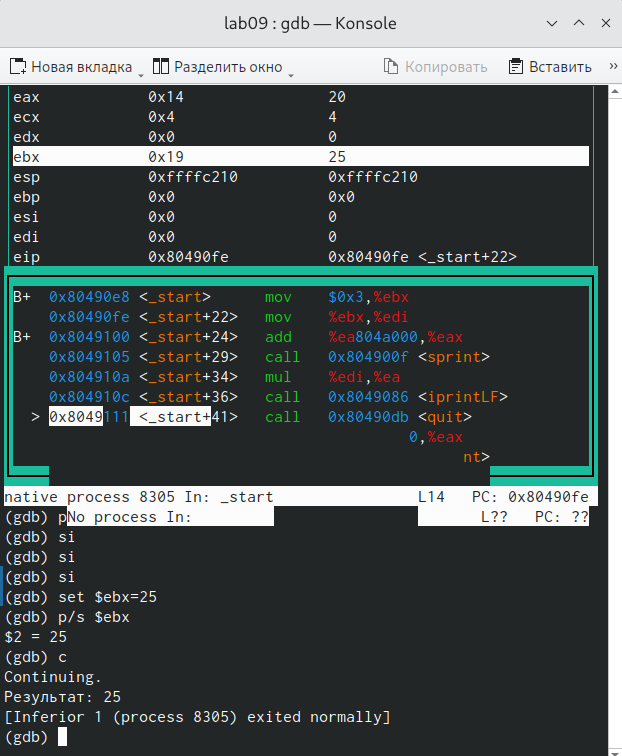
В результате этого шага мы поместили в регистр ecx значение для вычисления произведения. После произведения значения регистров будут следующими: (рис. ??).



Результат вычисления произведения

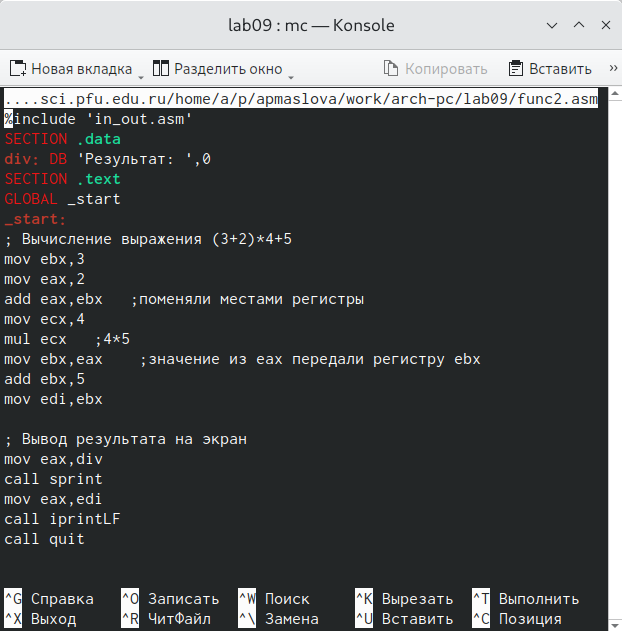
В результате в регистр eax было помещено значение произведения .

Далее к этому значению нужно прибавить . В программе за результат отвечает регистр ebx. Поместим в него значение и запустим программу на вывод конечного результата(рис. ??).



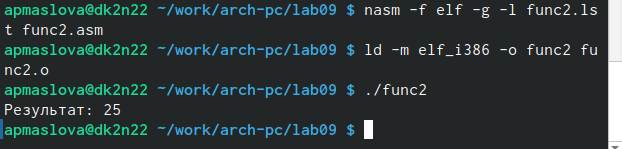
Завершение выполнения программы с помощью команды c

Как мы видим, с учётом всех изменений программа выдаёт верный результат. Теперь изменим код программы в файле func2.asm (рис. ??).



Исправленный текст программы в файле func2

Теперь создадим исполняемый файл и проверим корректность работы программы (рис. ??).



Проверка работы программы func2

Теперь программа выдаёт верный результат.

# 4 Выводы

Мы научились писать программы с использованием подпрограмм. Познакомились с методами отладки при помощи GDB и его основными возможностями.

# Список литературы

1. GDB: The GNU Project Debugger. — URL: https://www.gnu.org/software/gdb/.
2. GNU Bash Manual. — 2016. — URL: https://www.gnu.org/software/bash/manual/.
3. Midnight Commander Development Center. — 2021. — URL: https://midnight-commander. org/.
4. NASM Assembly Language Tutorials. — 2021. — URL: https://asmtutor.com/.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O’Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658.
6. Robbins A. Bash Pocket Reference. — O’Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: https://www.nasm.us/docs.php.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс,