Понятие подпрограммы. Отладчик GDB.

Лабораторная работа №9

Губайдуллина Софья Романовна

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 программам с помощью GDB;
3. Работа с данными программы в GDB;
4. Обработка аргументов командной строки в GDB;
5. Задание для самостоятельной работы.

# 3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа: • обнаружение ошибки; • поиск её местонахождения; • определение причины ошибки; • исправление ошибки. Наиболее популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам. Наиболее популярные виды точек останова - точки Breakpoint и Watchpoint.

GDB (GNU Debugger — отладчик проекта GNU) работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования. Отладчик GDB (как и любой другой отладчик) позволяет увидеть, что происходит «внутри» программы в момент её выполнения или что делает программа в момент сбоя. GDB может выполнять следующие действия: • начать выполнение программы, задав всё, что может повлиять на её поведение; • остановить программу при указанных условиях; • исследовать, что случилось, когда программа остановилась; • изменить программу так, чтобы можно было поэкспериментировать с устранением эффектов одной ошибки и продолжить выявление других. Синтаксис команды для запуска отладчика имеет следующий вид: gdb [опции] [имя\_файла | ID процесса]

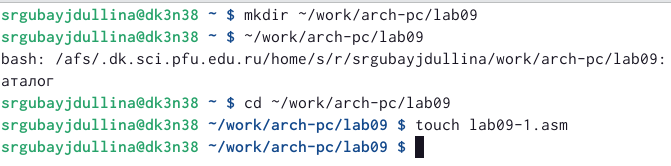
Если есть файл с исходным текстом программы, а в исполняемый файл включена информа- ция о номерах строк исходного кода, то программу можно отлаживать, работая в отладчике непосредственно с её исходным текстом. Чтобы программу можно было отлаживать на уровне строк исходного кода, она должна быть откомпилирована с ключом -g. Посмотреть дизассемблированный код программы можно с помощью команды disassemble : (gdb) disassemble \_start

Установить точку останова можно командой break (кратко b). Типичный аргумент этой команды — место установки. Его можно задать как имя метки или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка»: (gdb) break \* (gdb) b Для продолжения остановленной программы используется команда continue (c) (gdb) с [аргумент]. Команда stepi (кратко sI) позволяет выполнять программу по шагам, т.е. данная команда выполняет ровно одну инструкцию. Команда nexti (или ni) аналогична stepi, но вызов процедуры (функции) трактуется отладчиком как одна инструкция.

Подпрограмма — это, как правило, функционально законченный участок кода, который можно многократно вызывать из разных мест программы. В отличие от простых переходов из подпрограмм существует возврат на команду, следующую за вызовом. Для вызова подпрограммы из основной программы используется инструкция call, которая заносит адрес следующей инструкции в стек и загружает в регистр eip адрес соответствующей подпрограммы, осуществляя таким образом переход. Подпрограмма завершается инструкцией ret, которая извлекает из стека адрес, занесённый туда соответствующей инструкцией call, и заносит его в eip.

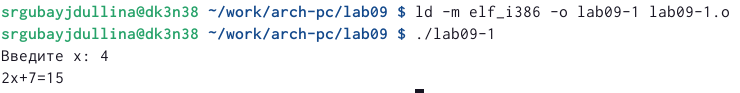
# 4 Выполнение лабораторной работы

1. Начинаю выполнение лабораторной работы с создания нового каталога lab09 и файла lab09-1.asm в нём.(рис. ??).



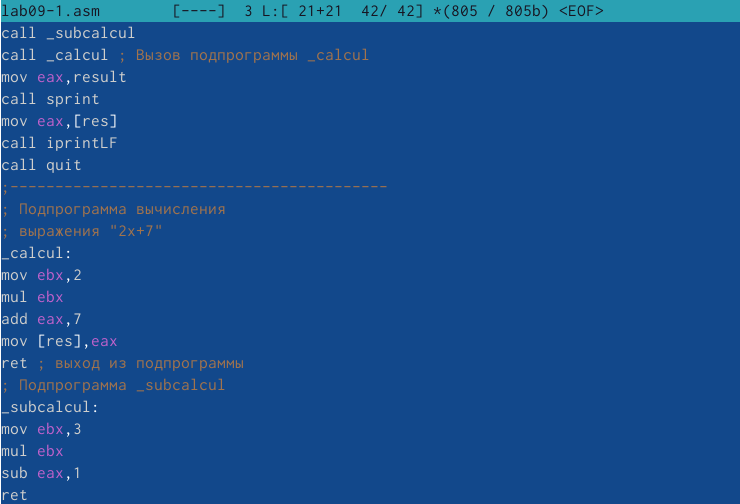
Создание нового файла и каталога

Ввожу в файл lab09-1.asm текст программы из листинга 9.1. Создаю исполняемый файл и проверьте его работу (рис. ??). Файл работает корректно.



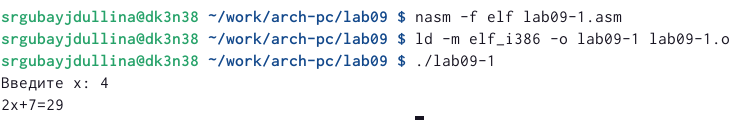
Проверка работы файла lab09-1.asm

Далее мне необходимо поменять текст программы так, чтобы в нем находилась подпрограмма \_subcul для вычисления выражения 𝑓(𝑔(𝑥)), где 𝑥 вводится с клавиатуры, 𝑓(𝑥) = 2𝑥 + 7, 𝑔(𝑥) = 3𝑥 − 1 (рис. ??)



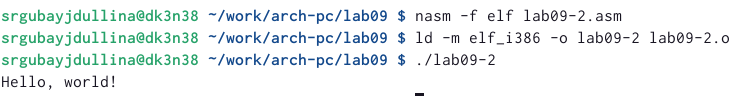
Измененный файл lab09-1.asm с подпрограммой

Создаю исполняемый файл для измененного lab09-1.asm и проверяю правильность работы программы (рис. ??)



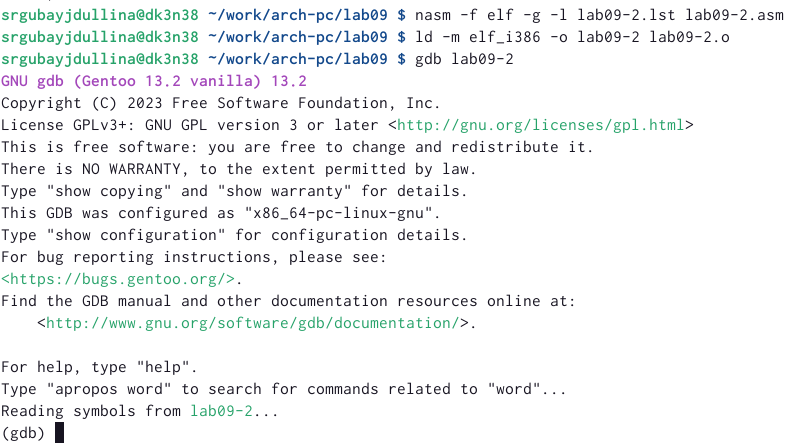
Работа файла lab09-1.asm с подпрограммой

1. Создаю новый файл lab09-2.asm и ввожу туда листинг 9.2. Проверяю работу (рис. ??)

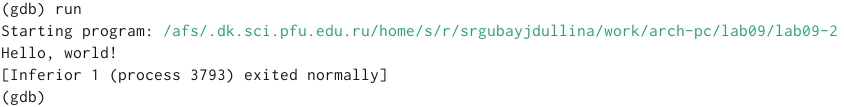


Проверка работы файла lab09-2.asm

Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого транслирую программы с ключом ‘-g’, загружаю исполняемый файл в отладчик gdb (рис. ??) и проверяю работу программы, запустив ее в оболочке GDB с помощью команды run (рис. ??)

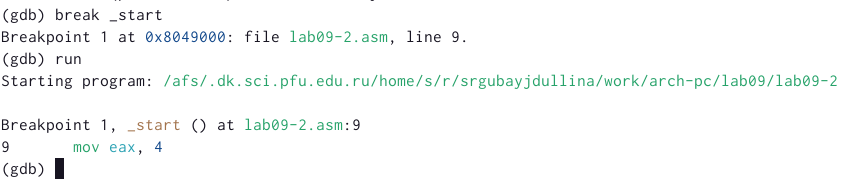


Загрузка файла в отладчик GDB



Запуск при помощи run

Теперь устанавливаю брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запускаю её (рис. ??)

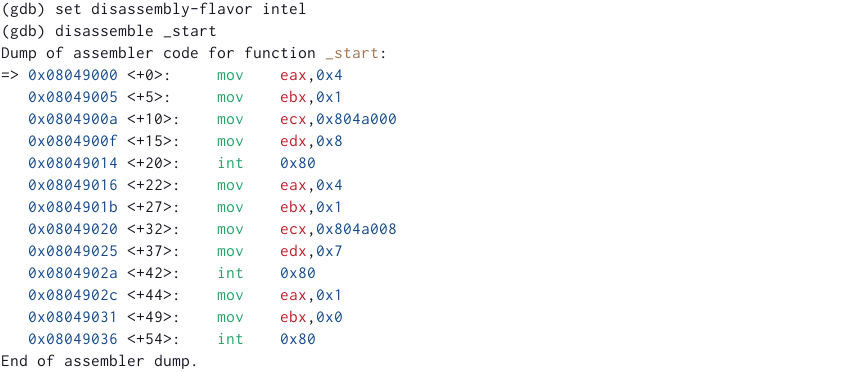


Установка брейпоинта на метку \_start

Анализирую дисассимилированный код программы, начиная с метки \_start (рис. ??), и сравниваю его с выводом отображения команд с Intel’овским синтаксисом (рис. ??).



Дисассимилированный код программы



Отображение команд с Intel-синтаксисом

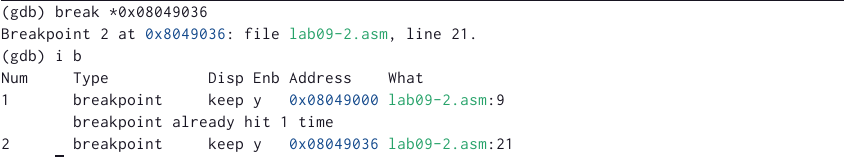
Безусловно, выводы различны. В режиме ATT имена регистров начинатся с символа %, а в синтаксисе Intel - с $.

Далее включаю режим псевдографики для работы в нём при помощи layout asm и layout regs. На предыдущих шагах была установлена точка останова по имени метки (\_start). Проверяю при помощи команды info breakpoints (i b) (рис. ??).



Режим псевдографики

Установливаю еще одну точку останова по адресу инструкции (рис. ??), а затем смотрю информацию обо всех имеющихся точках останова.

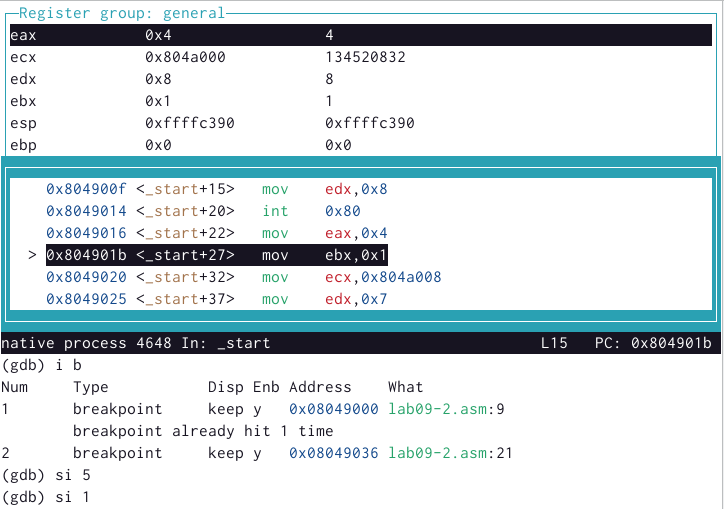


Установка точки брейкпоинт по её адресу

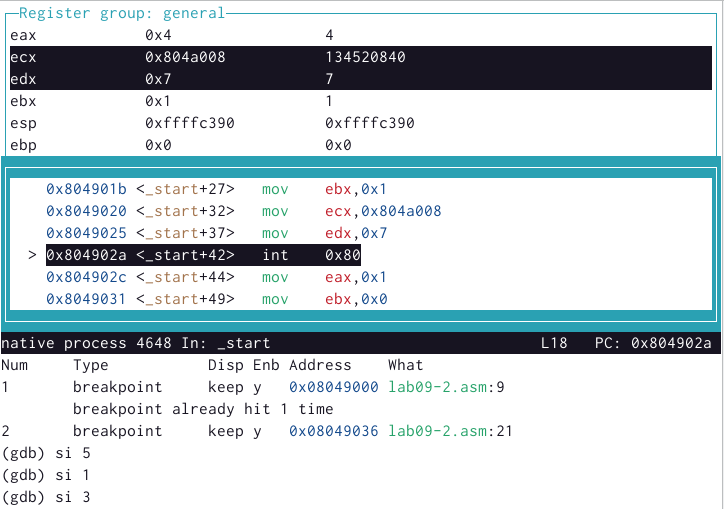
1. Далее по заданию мне необходимо выполнить 5 инструкций с помощью команды stepi (si) и проследить за изменением значений регистров (рис. ??) (рис. ??) (рис. ??) (рис. ??).



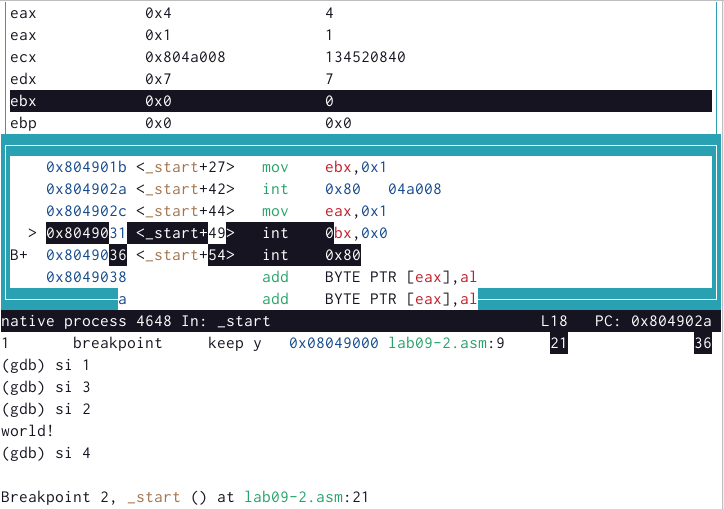
Анализ значений регистров при аргументе si 5



Анализ значений регистров при аргументе si 1



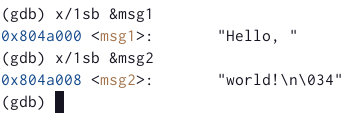
Анализ значений регистров при аргументе si 3



Анализ значений регистров при аргументе si 2 и si 4

Меняются значения регистров eax, ebx, ecx, edx.

Теперь мне необходимо посмотреть значение регистра msg1, а затем значение переменной msg2 (адрес можно узнать по дизассемблированной инструкции) (рис. ??).

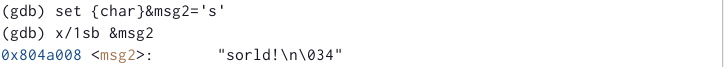


Значение регистра msg1 и переменной msg2

Изменить значение на ‘h’ для регистра msg1 при помощи set (рис. ??). Так же меняю значение регистра msg2 (рис. ??).



Замена значения регистра msg1



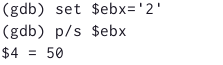
Замена значения регистра msg2

Необходимо вывести регистр edx в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) (рис. ??).

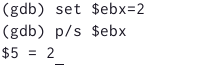


Представление регистра edx в различных форматах

Дважды при помощи команды set меняю значение регистра ebx (рис. ??) (рис. ??).



Замена значения регистра ebx

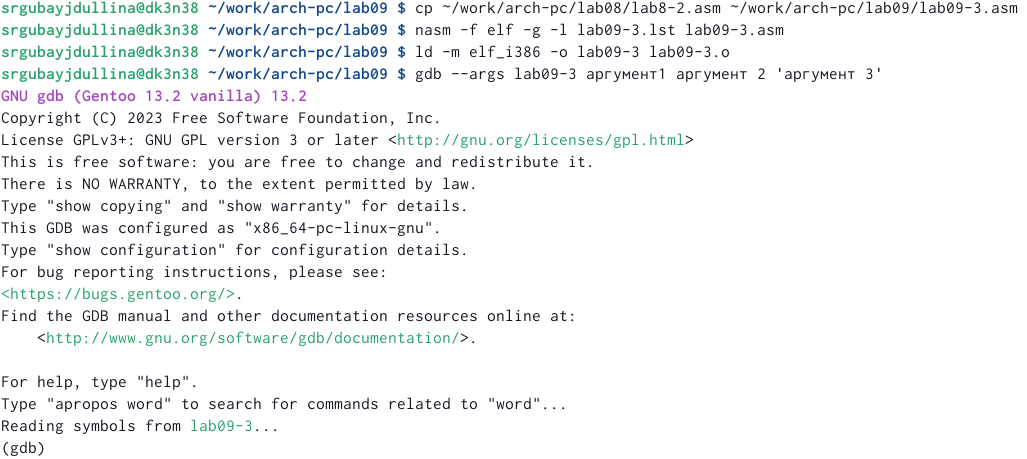


Повторная замена значения регистра ebx

Замена произведена по-разному: в первом случае мы меняем символ в строковый вид, а во втором - число не меняется из строкового вида.

Завершаю выполнение программы с помощью команды continue (c) и выхожу (quit).

1. Компилирую файл lab8-2.asm, созданный при выполнении лабораторной работы №8, в файл с именем lab09-3.asm, создаю исполняемый файл и при помощи ключа –args загружаю исполняемый файл в отладчик (рис. ??)



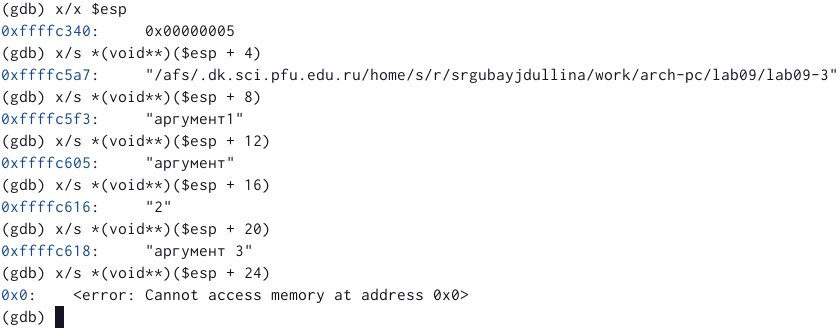
Компиляция и копирование lab08-2.asm в новый lab09-3.asm, загрузка в откладчик

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



Установка точки останова

Просматриваю позиции стека (рис. ??).



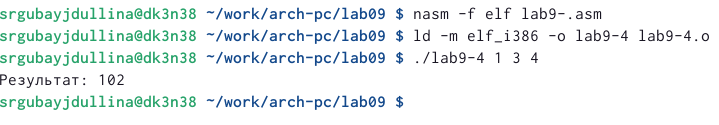
Значения позиций стека по адресам

Адреса меняются с шагом 4, т.к. мы имеем 4 аргумента командной строки.

1. Для начала самостоятельной работы я преобразовываю программу из файла lab08-4.asm таким образов, чтобы вычислить значения функции 𝑓(𝑥) как подпрограмму. Работаю в новом файле lab09-4.asm:

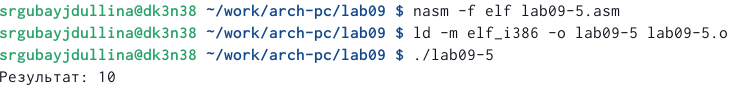
SECTION .data msg db “Результат”, 0 SECTION .text global \_start \_start: pop ecx pop edx sub ecx,1 mov esi,0 call next next: cmp ecx,0h pop eax call atoi mov edi,3 add eax,10 mul edi add esi,eax jz \_end loop next \_end: mov eax,msg call sprint mov eax,esi call iprintLF call quit ret

Запускаю программу (рис. ??).



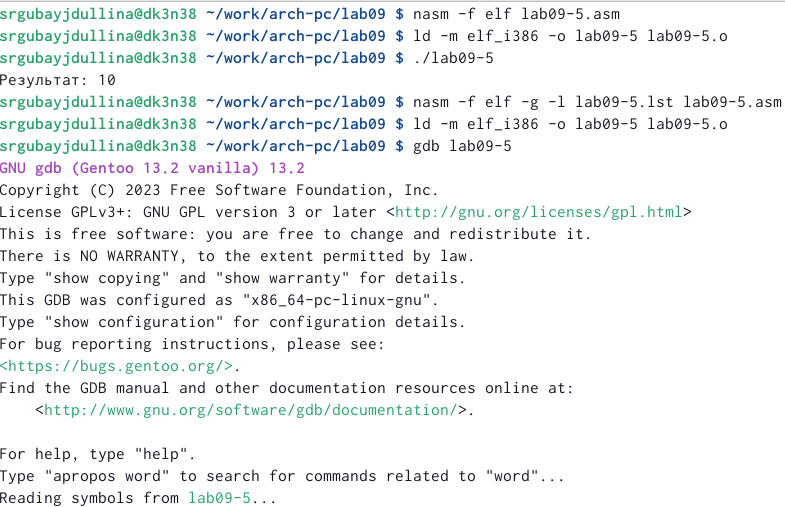
Запуск программы файла lab09-4.asm

Приступаю к выполнению второй части самостоятельной работы, а именно: копирую листинг 9.3 вычисления выражения (3 + 2) ∗ 4 + 5 в новый файл lab09-5.asm. При запуске данная программа дает неверный результат (рис. ??). С помощью отладчика GDB, анализируя изменения значений регистров, необходимо найти ошибку и исправьте ее.

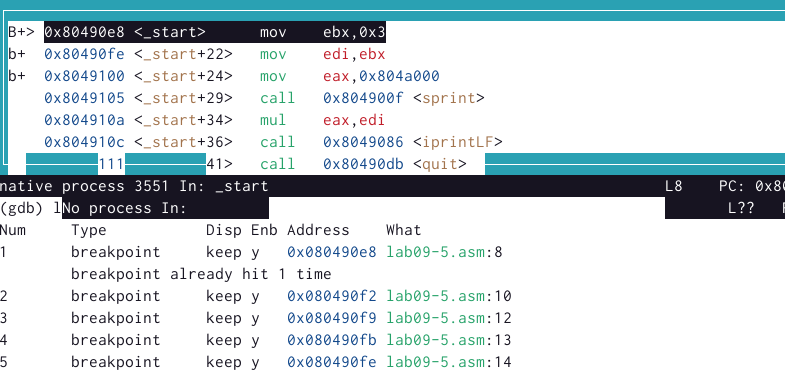


Запуск программы файла lab09-5.asm

Получаю исполняемый файл для GDB и запускаю. Ставлю брейкпоинт на \_start, запускаю и открываю режим псевдографики (рис. ??), и теперь ставлю брейкпоинты по адресу на строчки (рис. ??).

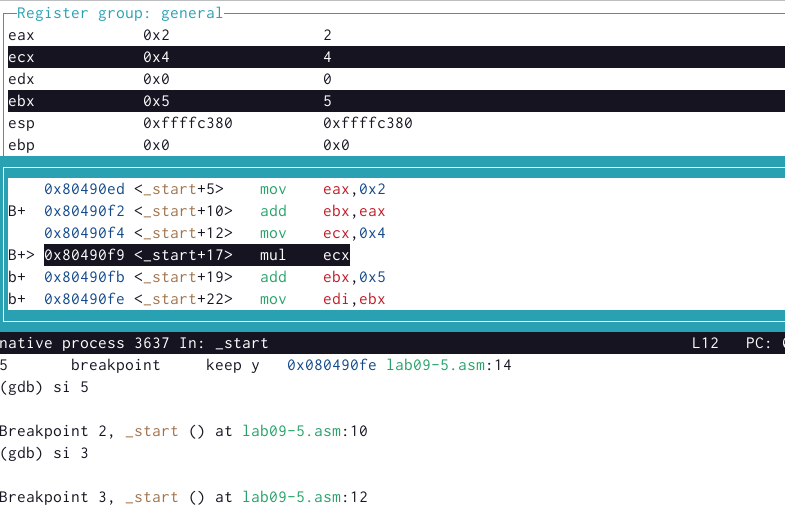


Получение исполняемого файла для запуска GDB

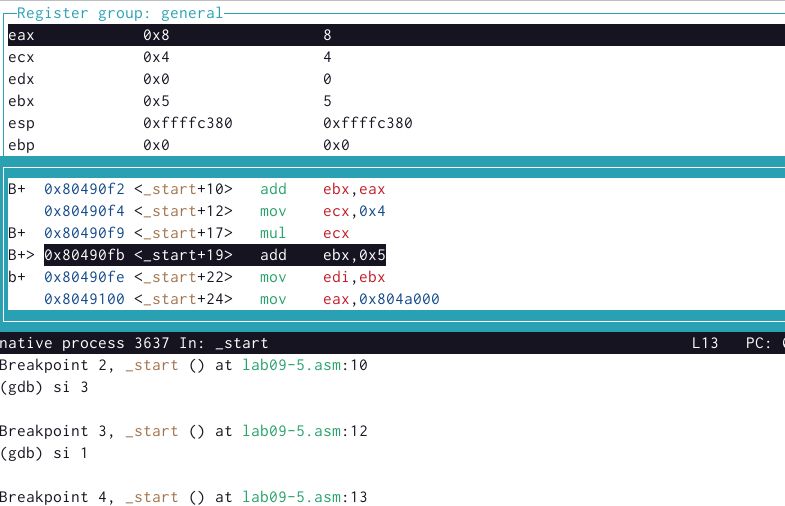


Установка брейкпоинтов на строчки

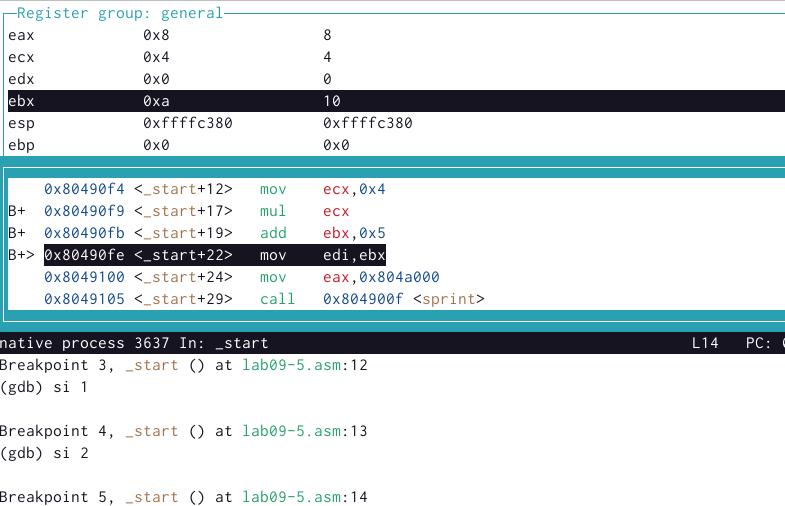
Теперь при помощи si анализирую изменения значения регистров (рис. ??) (рис. ??) (рис. ??) (рис. ??).



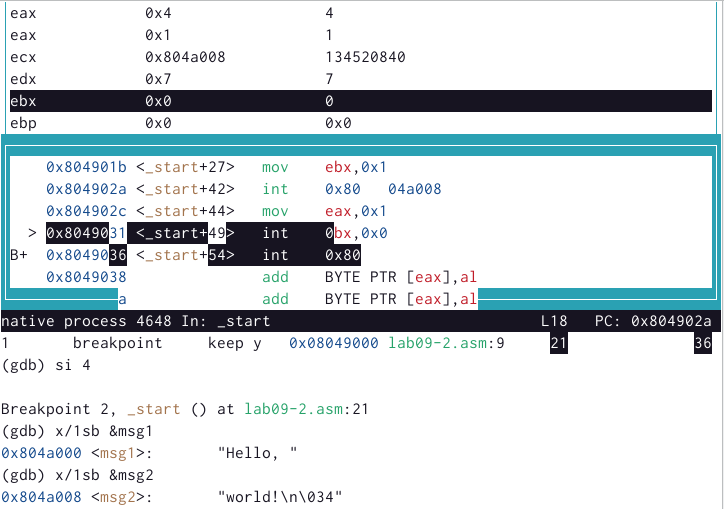
Просмотр значения регистров



Просмотр значения регистров



Просмотр значения регистров

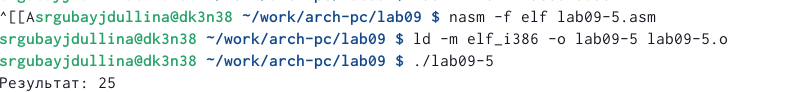


Просмотр значения регистров

Замечаю некоторый ообенности, а именно: происходит умножение 4 на 2 (ecx*eax), вместо 4 на 5 (ecx*ebx), как требуется (регистр ebx показан обнуленным). Функции mov ecx,4; add ebx,eax; mul ecx не связаны друг с другом. Нам необходимо добавить строчку mov edi,ebx. Ещё будет нужно заменить регистр ebx на eax, чтобы связать строчки программы. Теперь программа выглядит таким образом:

%include ‘in\_out.asm’ SECTION .data div: DB ‘Результат’, 0 SECTION .text GLOBAL \_start \_start: mov ebx,3 mov eax,2 add ebx,eax mov eax,ebx mov ecx,4 mul ecx add eax,5 mov edi,eax mov eax,div call sprint mov eax,edi call iprintLF call quit

Запустим программу для проверки (рис. ??). Программа работает корректно.

 # Выводы

В процессе работы над лабораторной работой я приобрела навыков написания программ с использованием подпрограмм, а так же ознакомилась с методами отладки при помощи GDB и его основными возможностями.