Отчёт по лабораторной работе №9

Дисциплина: архитектура компьютера

Репкина Елизавета 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. Релизация 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 Теоретическое введение

Понятие об отладке

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.

Можно выделить следующие типы ошибок:

• синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка;

• семантические ошибки — являются логическими и приводят к тому, что программа запускается, отрабатывает, но не даёт желаемого результата;

• ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают прерывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль).

Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга.

Третий этап — выяснение причины ошибки. После определения местонахождения ошибки обычно проще определить причину неправильной работы программы.

Последний этап — исправление ошибки. После этого при повторном запуске программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново.

Методы отладки

Наиболее часто применяют следующие методы отладки:

• создание точек контроля значений на входе и выходе участка программы (например, вывод промежуточных значений на экран — так называемые диагностические сообщения);

• использование специальных программ-отладчиков.

Отладчики позволяют управлять ходом выполнения программы, контролировать и изменять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправление. Наиболее популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам. Пошаговое выполнение — это выполнение программы с остановкой после каждой строчки, чтобы программист мог проверить значения переменных и выполнить другие действия. Точки останова — это специально отмеченные места в программе, в которых программаотладчик приостанавливает выполнение программы и ждёт команд. Наиболее популярные виды точек останова:

• Breakpoint — точка останова (остановка происходит, когда выполнение доходит до определённой строки, адреса или процедуры, отмеченной программистом);

• Watchpoint — точка просмотра (выполнение программы приостанавливается, если программа обратилась к определённой переменной: либо считала её значение, либо изменила его).

Точки останова устанавливаются в отладчике на время сеанса работы с кодом программы, т.е. они сохраняются до выхода из программы-отладчика или до смены отлаживаемой программы.

# 4 Выполнение лабораторной работы

Реализация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы №9, перехожу в него и создаю файл lab09-1.asm.(рис. 1)

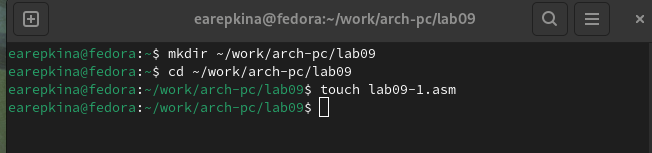


Рис. 1: Выполнение команд

Ввожу в файл lab09-1.asm текст программы из листинга 9.1. (рис. 2)

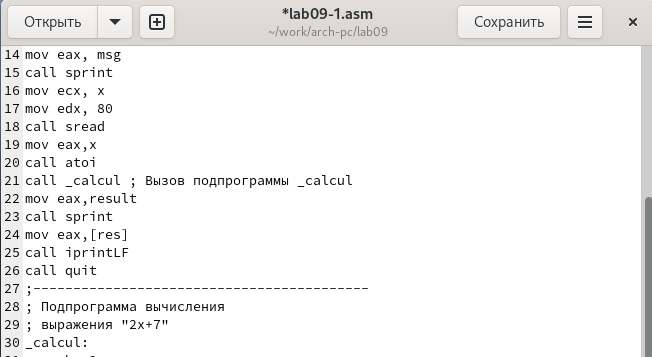


Рис. 2: Редактирование файла

Создаю исполняемый файл и запускаю его.(рис. 3)

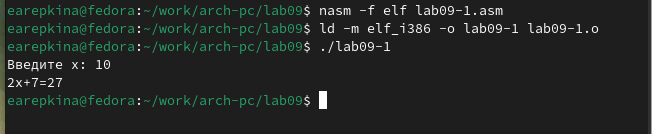


Рис. 3: Запуск файла

Изменяю текст программы, добавляя подпрограмму \_subcalcul в подпрограммы \_calcul (рис. 4)

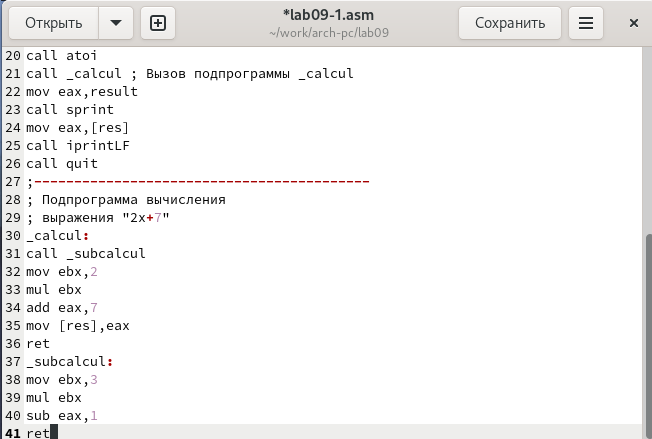


Рис. 4: Редактирование файла

Создаю исполняемый файл и запускаю его.(рис. 5)

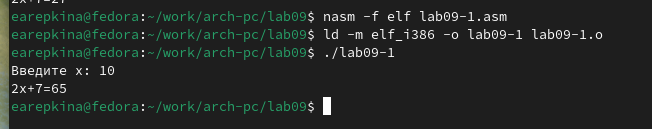


Рис. 5: Запуск файла

Программа работает корректно.

Отладка программам с помощью GDB

Создаю файл lab09-2.asm с текстом программы из Листинга 9.2 (рис. 6)

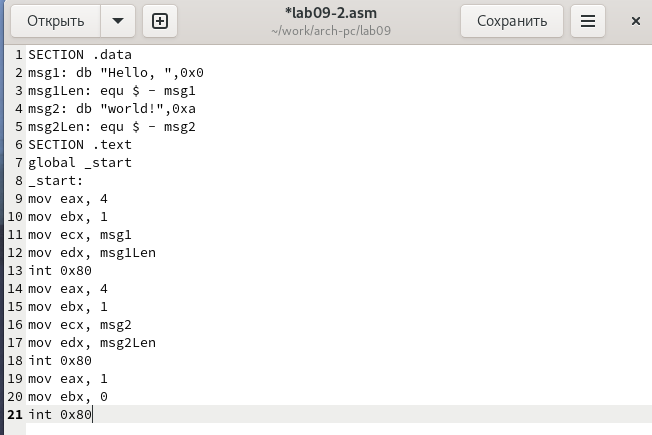


Рис. 6: Создание файла печати сообщения Hello world!

Получаю исполняемый файл.Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’.Далее загружаю исполняемый файл в отладчик gdb и запускаю его с помощью команды run.(рис. 7)

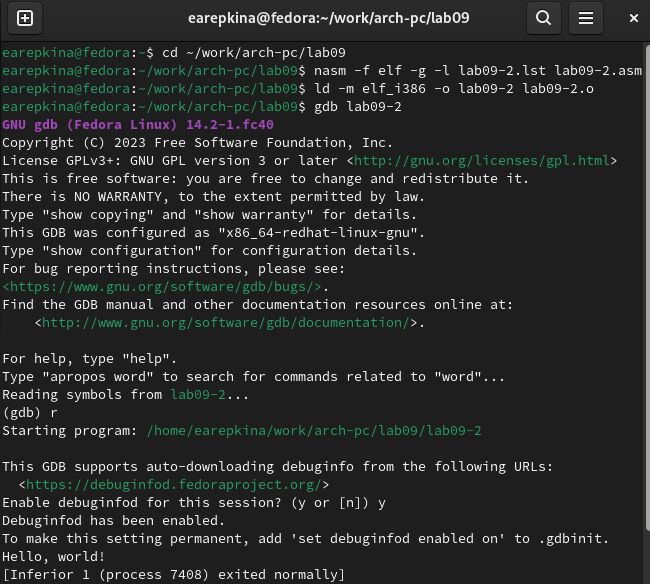


Рис. 7: Получение и загрузка в отлатчик исполняемого файла

Устанавливаю брейкпоинт на метку \_start и запукаю программу (рис. 8)

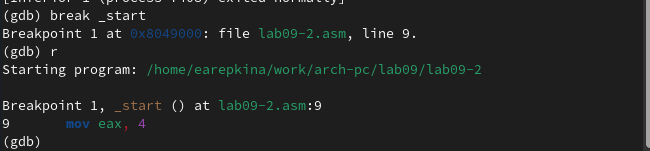


Рис. 8: Просмотр дисассимилированного кода программы

Смотрю дисассимилированный код программы с помощью команды disassemble начинаю с метки \_start (рис. 9)

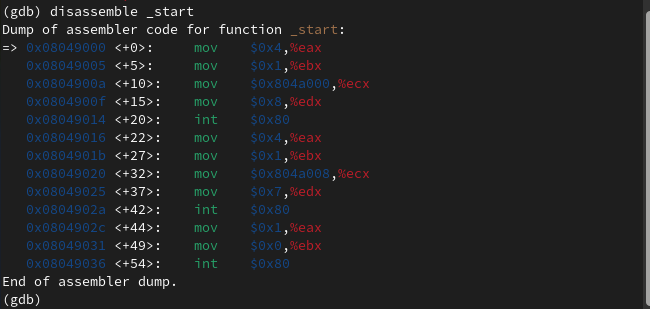


Рис. 9: Просмотр дисассимилированного кода программы

Переключаюсь на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel (рис. 10)



Рис. 10: Переключение на Intel’ский синтаксис

Отличия заключаются в том, что в режиме ATT используются “%” перед перед именами регистров и “$” перед именами операндов, а в режиме Intel используется обычный синтаксис. Включаю режим псевдографики для более удобного анализа программы(рис. 11)

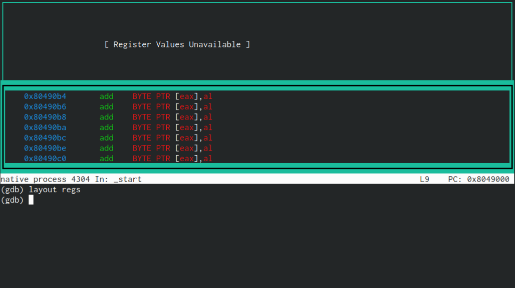


Рис. 11: Включение режима псевдографики

Добавление точек останова Проверяю точку останова по имени метки (рис. 12)

Рис. 12: Проверка точки останова

Рис. 12: Проверка точки останова

Определю адрес пердпоследней инструкции и устанавливаю точку останова. Далее смотрю информацию о всех установленных точках останова (рис. 13)

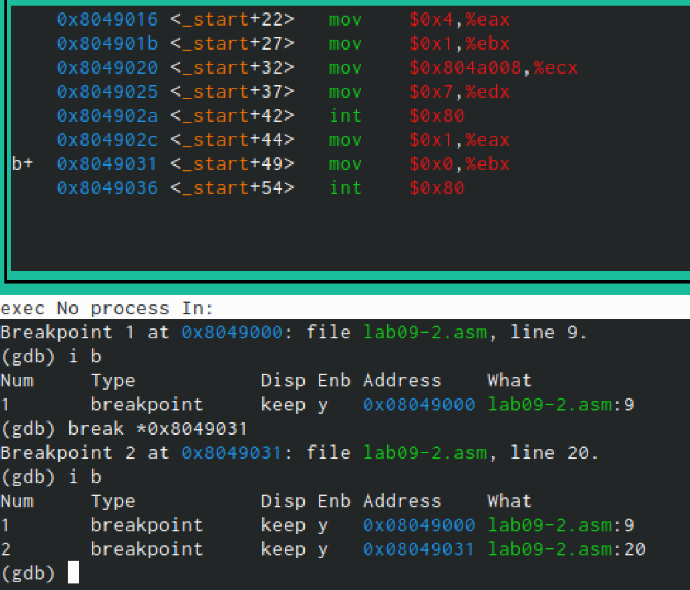


Рис. 13: Установка точки останова и её проверка

Работа с данными программы в GDB

Выполняю 5 инструкций с помощью команды stepi и слежу за изменением регистров. (рис. 14) (рис. 15)

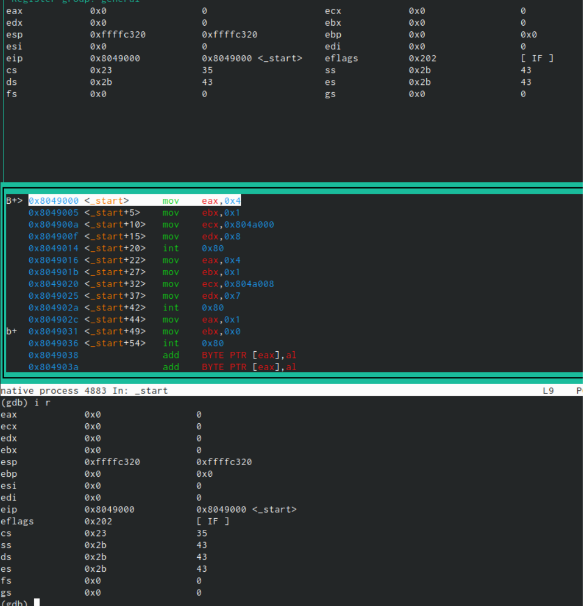


Рис. 14: До использования комадны stepi

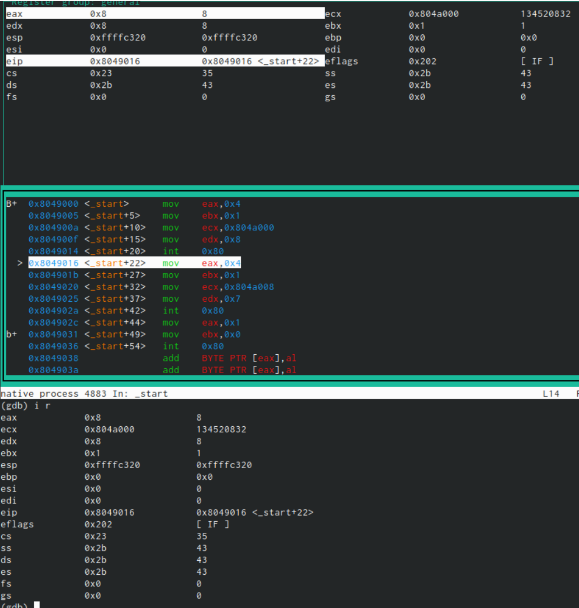


Рис. 15: После использования команды stepi

Изменились регистры eax,ebx,ecx,edx Просматриваю значение переменной msg1 по имени (рис. 16)

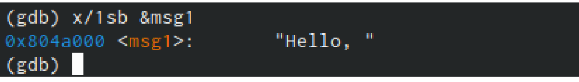


Рис. 16: Просмотр значения переменной msg1

Также просматриваю значение переменной msg2 (рис. 17)

Рис. 17: Просмотр значения переменной msg2

Рис. 17: Просмотр значения переменной msg2

Изменяю первый символ переменной msg1 (рис. 18)

Рис. 18: Изменение первого символа msg1

Рис. 18: Изменение первого символа msg1

Изменяю первый символ переменной msg2. (рис. 19)

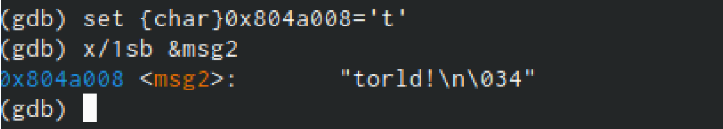


Рис. 19: Изменение первого символа переменной msg2

Вывожу в различных форматах значение регистра ebx (рис. 20)

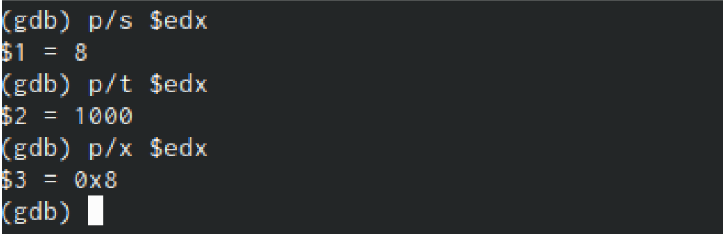


Рис. 20: Вывод значений регистра ebx

С помощью команды set изменяю значение регистра ebx(рис. 21)

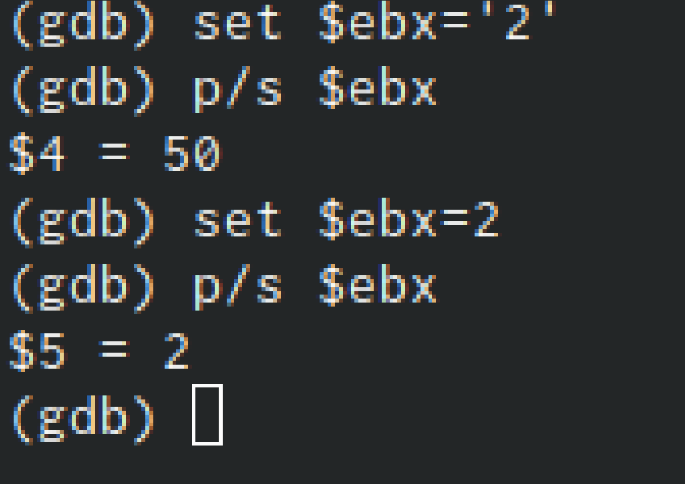


Рис. 21: Изменение значения регистра ebx

Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm (рис. 22)

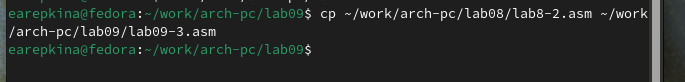


Рис. 22: Копирование файла

Создаю исполняемый файл (рис. 23)

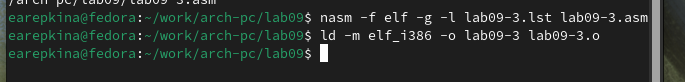


Рис. 23: Создание исполняемого файла

Загружаю исполняемый файл в отладчик, указав агрументы (рис. 24)

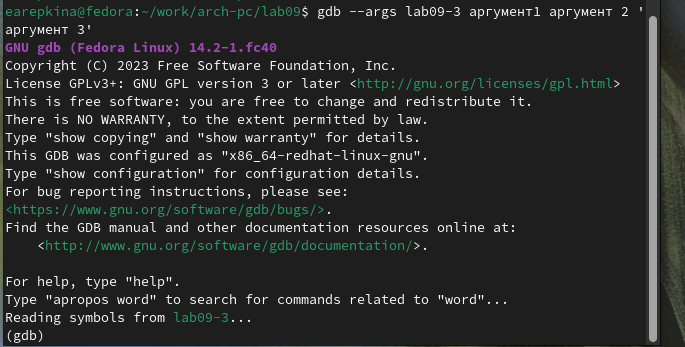


Рис. 24: Загрузка исполняемого файла в отладчик

Для начала устанавливаю точку останова перед первой инструкцией в программе и запускаю её (рис. 25)

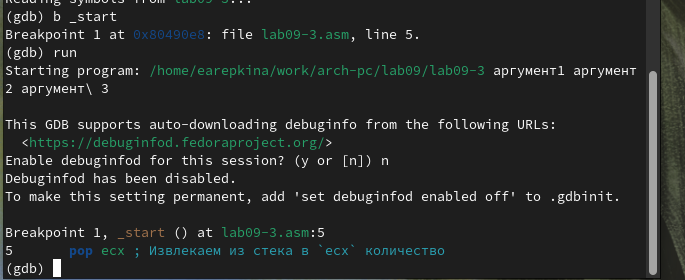


Рис. 25: Установка точки остановы

Просматриваю вершину стека, то есть число аргументов строки(включая имя программы) (рис. 26)

Рис. 26: Просмотр вершины стека

Рис. 26: Просмотр вершины стека

Просматриваю остальные позиции стека. (рис. 27)

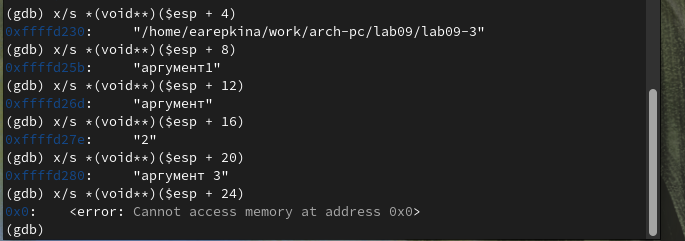


Рис. 27: Просмотр остальных позиций стека

Шаг изменения адреа равен 4, потому что занчение регистра esp в стеке увели- чивается на 4

Задание для самостоятельной работы. 1. Открываю программу из лабораторной работы №8 и начинаю её редактировать (рис. 28)

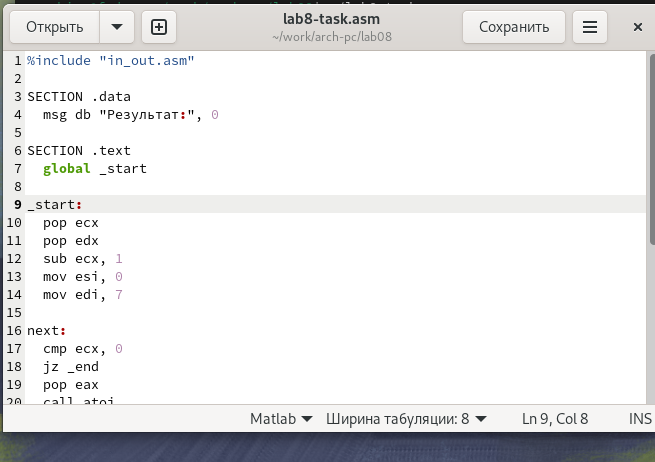


Рис. 28: Редактирование файла

Создаю исполняемый файл и запускаю его, чтобы проверить работу программы (рис. 29)

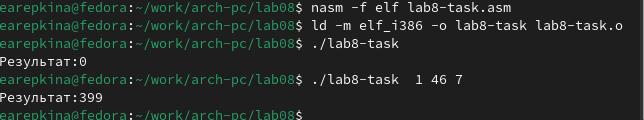


Рис. 29: Запуск исполняемого файла

Программа работает верно.

Текст программы:

%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

mov edi, 7

next:

cmp ecx, 0

jz \_end

pop eax

call atoi

call \_f

add esi, eax

loop next

\_end:

mov eax, msg

call sprint

mov eax, esi

call iprintLF

call quit

\_f:

inc eax

mul edi

ret

1. Создаю файл и ввожу туда текст программы из листинга 9.3 (рис. 30)

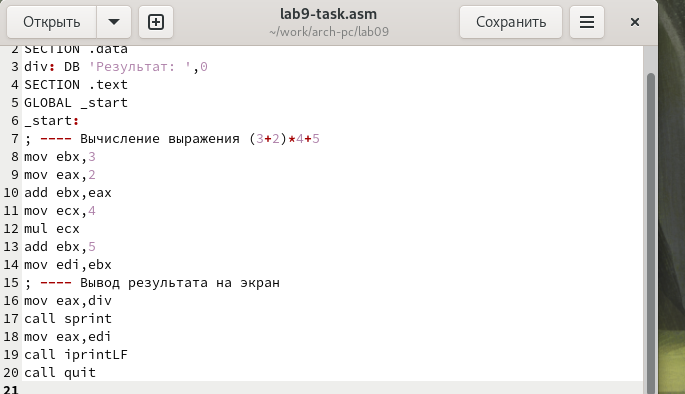


Рис. 30: Редактирование файла

Создаю исполняемый файл и запускаю его, чтобы проверить работу программы (рис. 31)

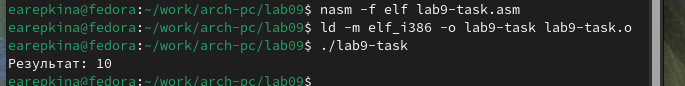


Рис. 31: Запуск исполняемого файла

Убеждаюсь, что программа работает неверно.

Создаю исполняемый файл для работы с GDB и запускаю его через режим отладки. Создаю брейкпойнт и пошагово просматириваю программу (рис. 32)

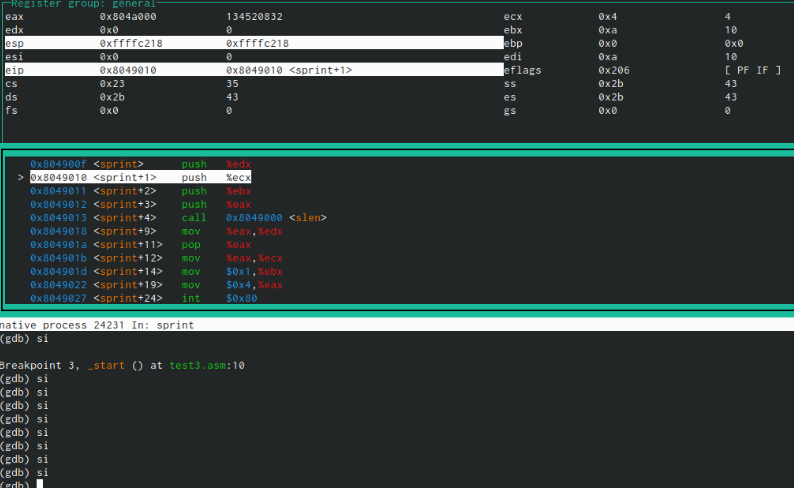


Рис. 32: Просмотр программы

Замечаю, что после выполнения инструкции mul программы умножент 4 на 2 на не на 5, как должно быть.Из-за этого программа выдает неверный результат.

Далле открываю файл с программой и исправляю ошибку (рис. 33)

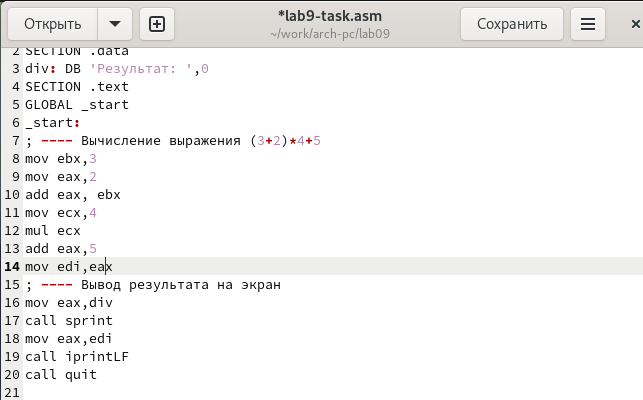


Рис. 33: Редактирование программы

Создаю исполняемый файл и запускаю его, чтобы проверить работу программы (рис. 34)

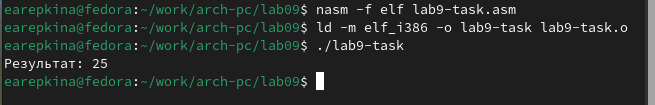


Рис. 34: Запуск исполняемого файла

Теперь программа работает верно.

Текст программы:

%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

# 5 Выводы

После выполнения данной лабораторной работы я приобрелa навыки написания программ с использование подпрограмм и познакомилась 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,
11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM. — 2021. — URL: https://www.opennet.ru/docs/RUS/nasm/.
13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm\_unix.
15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер,
17. — 1120 с. — (Классика Computer Science).