Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Отчет по лабораторной работе №2**

Дисциплина: Низкоуровневое программирование

Тема: Программирование RISC-V

Выполнил студент гр. 3530901/10005\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Хворостов С. А.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Коренев Д. А.

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г

Санкт-Петербург

2021

**Задачи:**

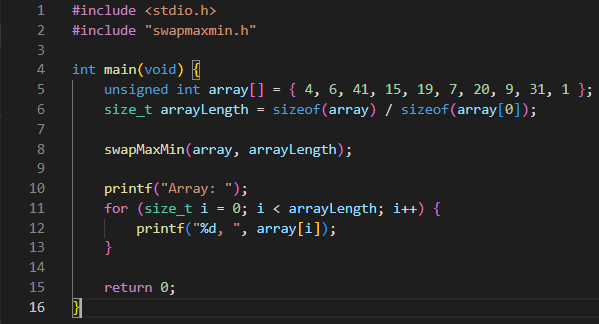
1. На языке C разработать функцию, реализующую определённую вариантом задания функциональность. Поместить определение функции в отдельный исходный файл, оформить заголовочный файл. Разработать тестовую программу на языке C.
2. Собрать программу по «шагам». Проанализировать выход препроцессора и компилятора. Проанализировать состав и содержимое секций, таблицы символов, таблицы перемещений и отладочную информацию, содержащуюся в объектных файлах и исполняемом файле.
3. Выделить разработанную функцию в статическую библиотеку. Разработать make-файлы для сборки библиотеки и использующей её тестовой программы. Проанализировать ход сборки библиотеки и программы, созданные файлы зависимостей.

**Вариант задания:**

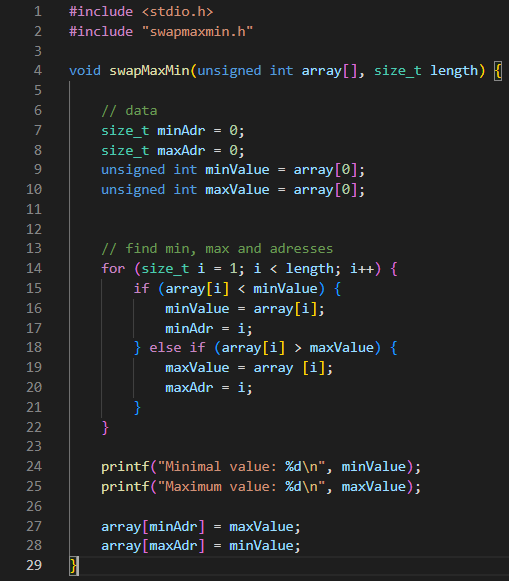
Вариант 15 – поменять в массиве максимальный и минимальный элементы.

**Ход решения:**

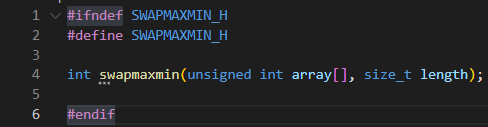
**Текст программы:**



main.c



swapmaxmin.c



swapmaxmin.h

Файл main.c содержит реализацию тестовой программы и вывод в консоль нового массива.

Файл swapmaxmin.c содержит реализацию функции swapMaxMin(), которая меняет в массиве максимальный и минимальный элементы и выводит в консоль минимальное и максимальное значения. Алгоритм перестановки макс и мин элементов следующий.

За первое максимальное и первое минимальное число берется первое число массива. Дальше начинается перебор массива со 2го элемента и если какое-либо из чисел оказывается меньше текущего минимального, то минимальный элемент перезаписывается и сохраняется его адрес. Если какое-либо из чисел оказывается больше максимального, то максимальный элемент перезаписывается и сохраняется его адрес. Далее просто макс элемент записывает в адрес мин и наоборот

**Сборка программ «по шагам», анализ промежуточных и результирующих файлов**

Для сборки выполним программы выполним следующую команду:

*riscv64-unknown-elf-gcc --save-temps -march=rv32i -mabi=ilp32 -O1 -v main.c swapmaxmin.c >log 2>&1*

В результате исполнения приведенной команды в текущем каталоге будут созданы следующие файлы:

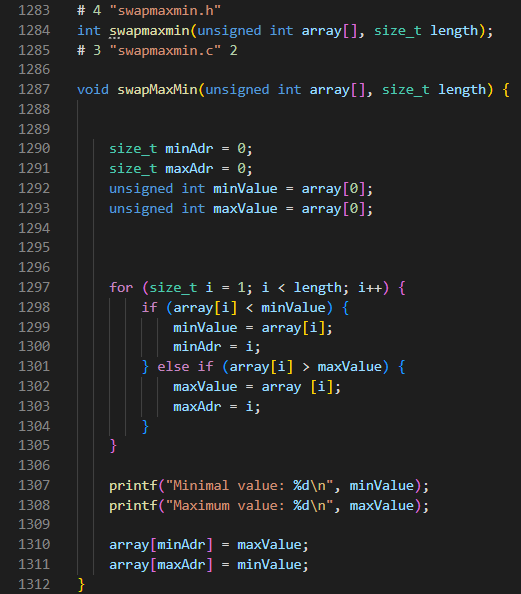
main.i – текстовый файл, содержащий результат препроцессирования файла main.s – текстовый файл, содержащий код на языке ассемблера, сгенерированный компилятором в результате обработки файла “main.i”;

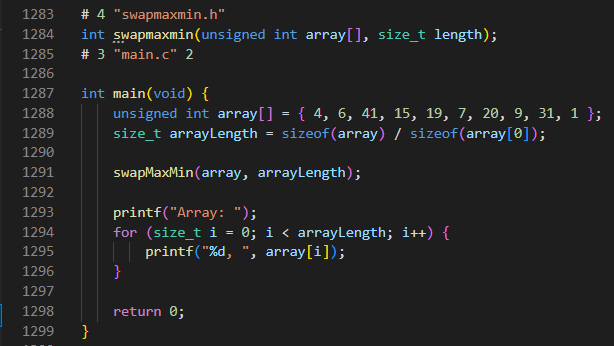
main.o – объектный файл сгенерированный ассемблером в результате обработки файла “main.s”;

a.out – исполняемый файл сгенерированный компоновщиком в результате обработки файла “main.o”, а также других объектных файлов и файлов библиотек, входящих в состав пакета средств разработки.

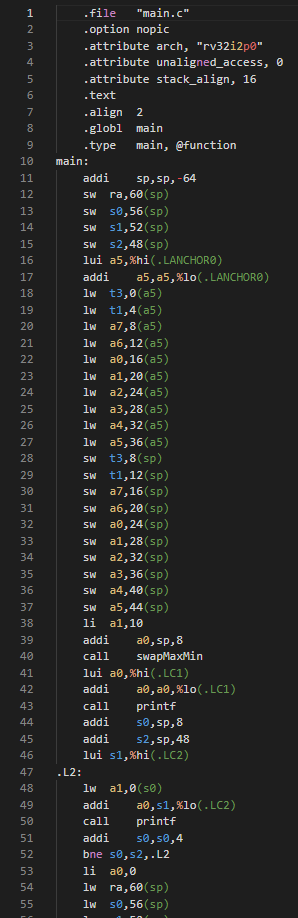
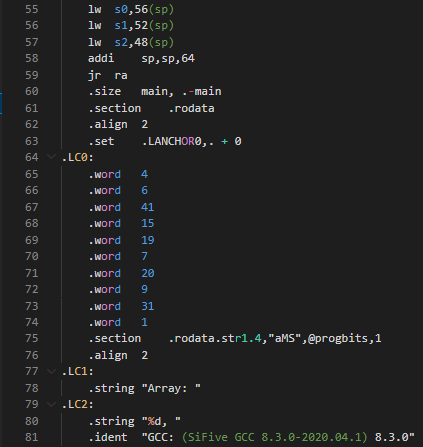
log – текстовый файл, содержащий сообщения компилятора, ассемблера и компоновщика, а также выполняемые команды и дополнительную информацию;

Исходя из вывода лог-файла, процесс сборки простейшей программы состоит из следующих шагов:

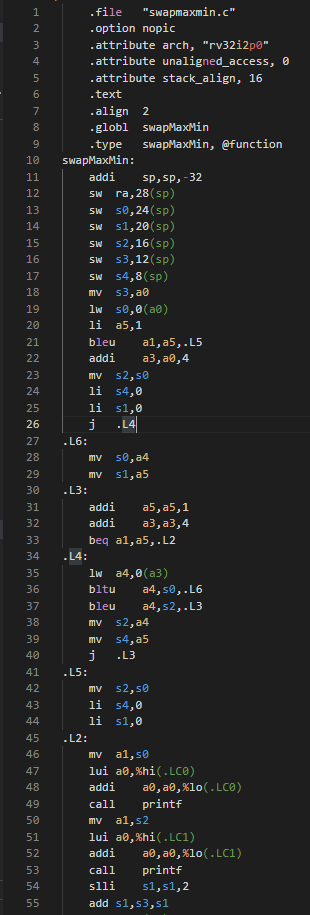
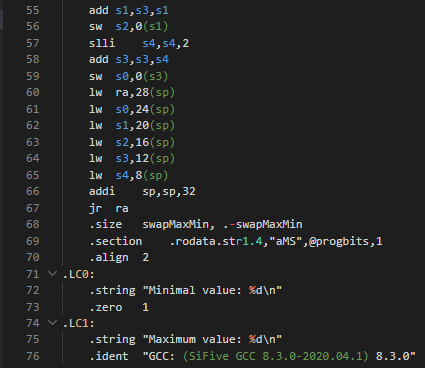
1. Первым делом выполняется препроцессирование исходных файлов main.c и swapmaxmin.c. Результатом работы служат файлы main.i и swapmaxmin.i. В программе используется директива #include<stdio.h>, поэтому в самом файле в начале файла содержится порядка 1200 строк с инструкциями, относящимися к этой директиве, а затем следует код на C, который соответствует программе. Появившиеся нестандартные директивы, начинающиеся с символа “#”, используются для передачи информации об исходном тексте из препроцессора в компилятор. ****   
   Частичное содержимое swapmaxmin.i



Частичное содержимое main.i

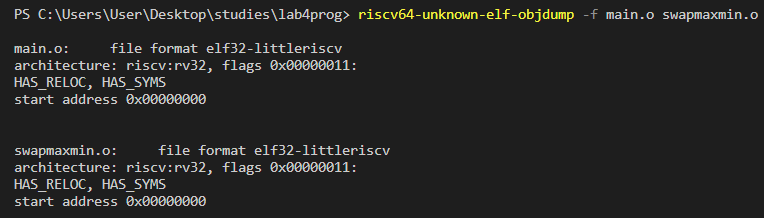
1. Следующим шагом является компиляция файлов main.i и swapmaxmin.i в код на языке ассемблера, содержащийся в файлах main.s и swapmaxmin.s  
    

Cодержимое файла main.s

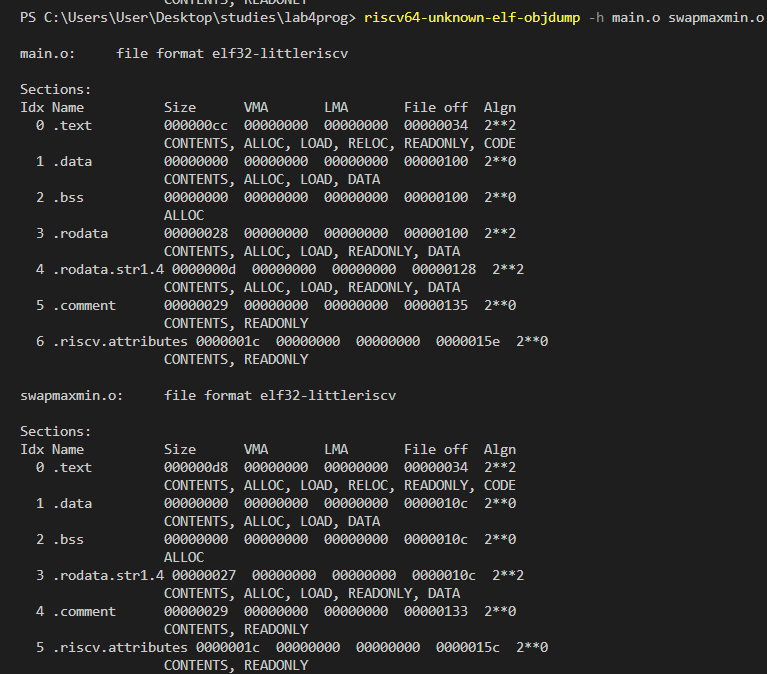
Cодержимое файла swapmaxmin.o

1. Следующим шагом является ассемблирование файлов main.s и swapmaxmin.s в объектные файлы main.o и swapmaxmin.o

В отличие от ранее рассмотренных файлов, объектный фал не является текстовым, и не может быть напрямую выведен на экран в читаемом формате. Для изучения его содержимого воспользуемся командой:  
*riscv64-unknown-elf-objdump -f main.o swapmaxmin.o*  
  


Оба файла имеют формат ELF, являются объектным файлом 32-разрядной архитектуры RISC-V, содержат символы (флаг HAS\_SYMS), содержат таблицу перемещений (флаг HAS\_RELOC). Объектный файл не должен содержать адрес точки входа (адрес, с которого начинается исполнение программы), однако поскольку соответствующее поле присутствует в заголовке файла формата ELF, оно заполняется 0.

Далее выведем все заголовки секций объектных файлов при помощи команды  *riscv64-unknown-elf-objdump -h main.o* *swapmaxmin.o*

**

В файлах содержатся следующие секции

*.text* – секция кода, в которой содержатся коды инструкций

.*data* – секция инициализированных данных

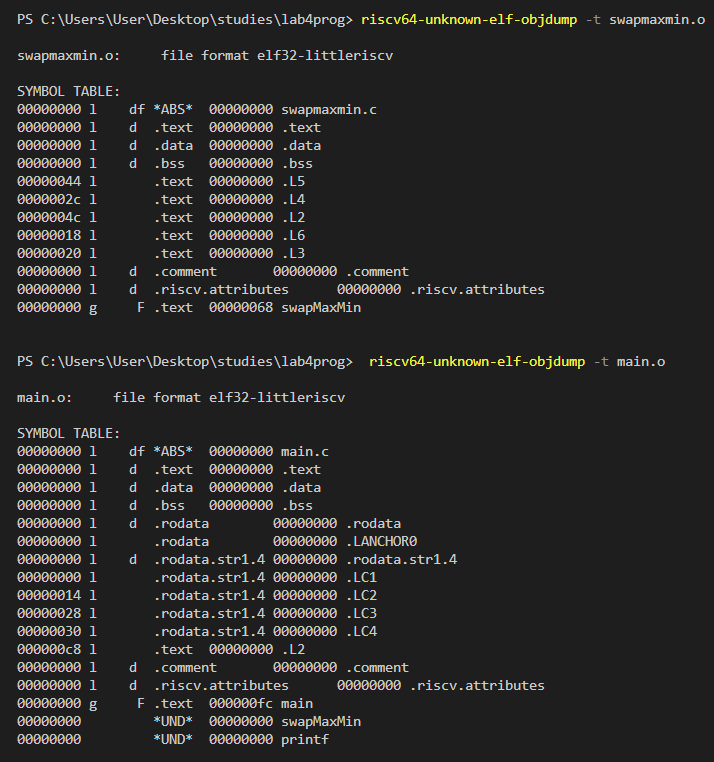
*.bss* – секция неинициализированных статических переменных

*.rodata* – аналог .data для неизменяемых данных

*.comment* – секция данных о версиях размером 12 байт

*.riscv.attributes* – информация про RISC-V

Далее изучим таблицу вывода символов *(команда riscv64-unknown-elf-objdump -t main.o):*

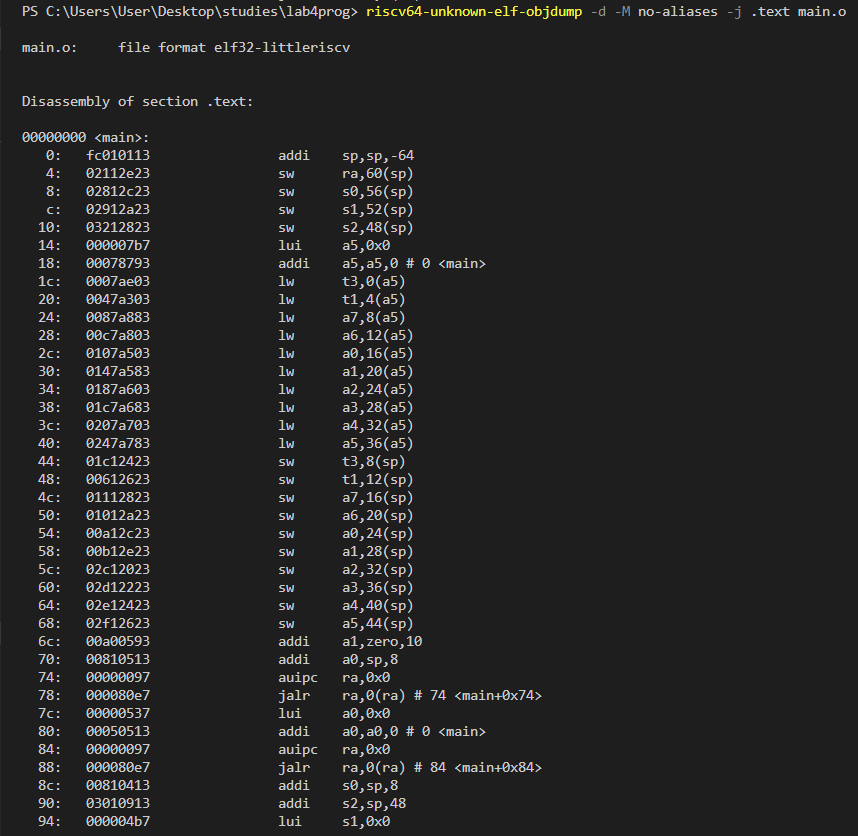


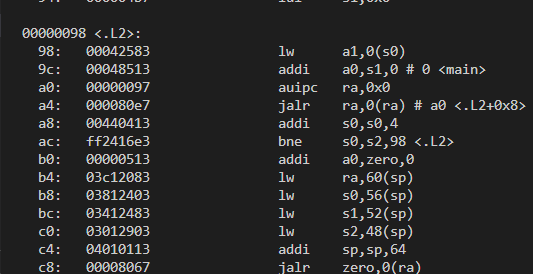
В каждой таблице только один глобальный (флаг “g”) символ типа «функция» (“F”) – main и swapMaxMin соответственно.

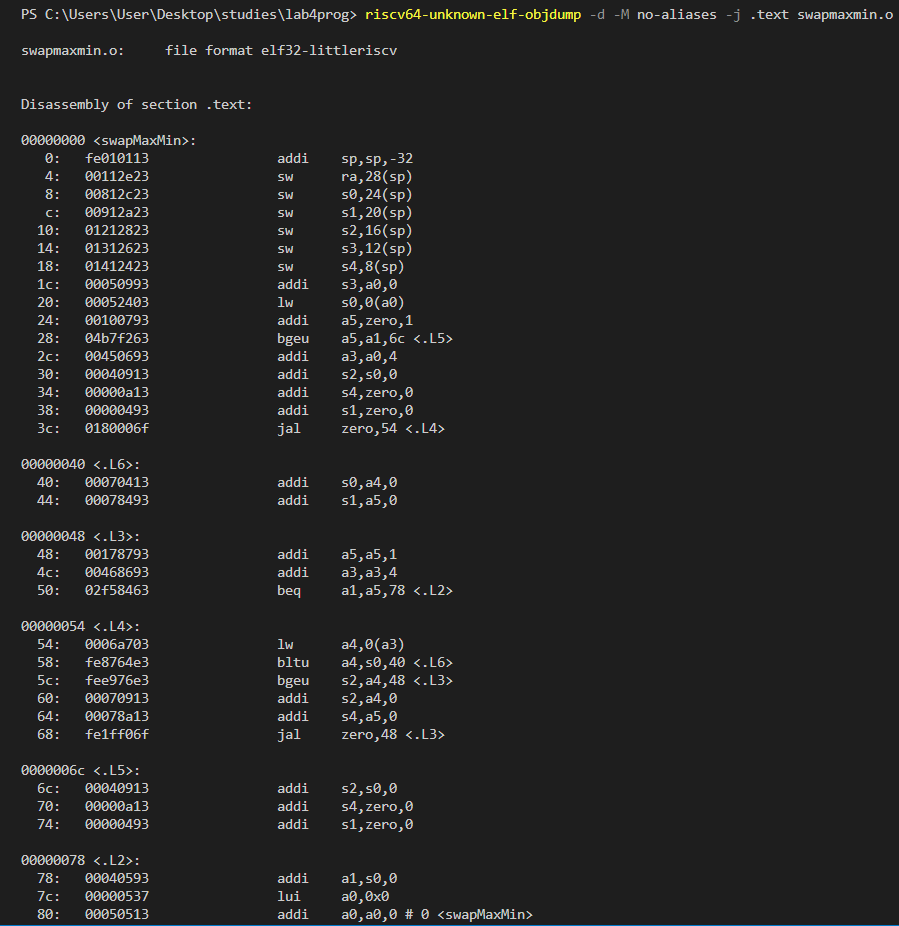
UND означает, что символ printf использовался в ассемблерном коде, из которого был получен данный объектный фал, но не был определён

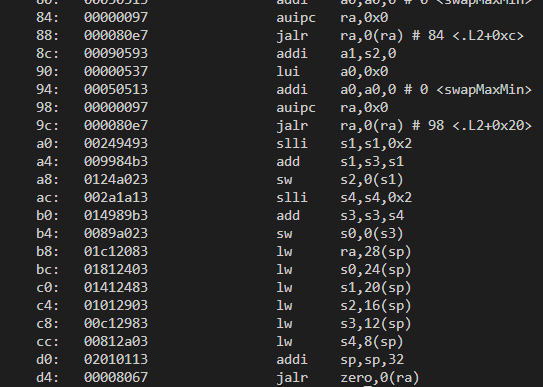
Изучим содержимое секции .text для этого используем команду:

*riscv64-unknown-elf-objdump -d -M no-aliases -j .text main.o*

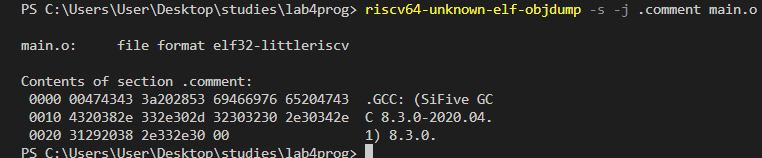
**

**

**

**

Дизассемблированный код практически идентичен сгенерированному (за исключением псевдоинструкций), что логично, учитывая что раздел .text содержит коды инструкций

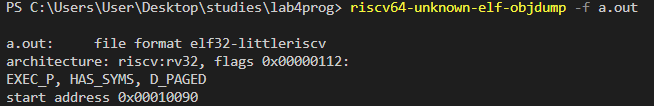
Изучим секцию .comment при помощи команды *riscv64-unknown-elf-objdump -s -j .comment main.o*

Здесь можно увидеть, что секция содержит строку, указанную в директиве ассемблера .ident



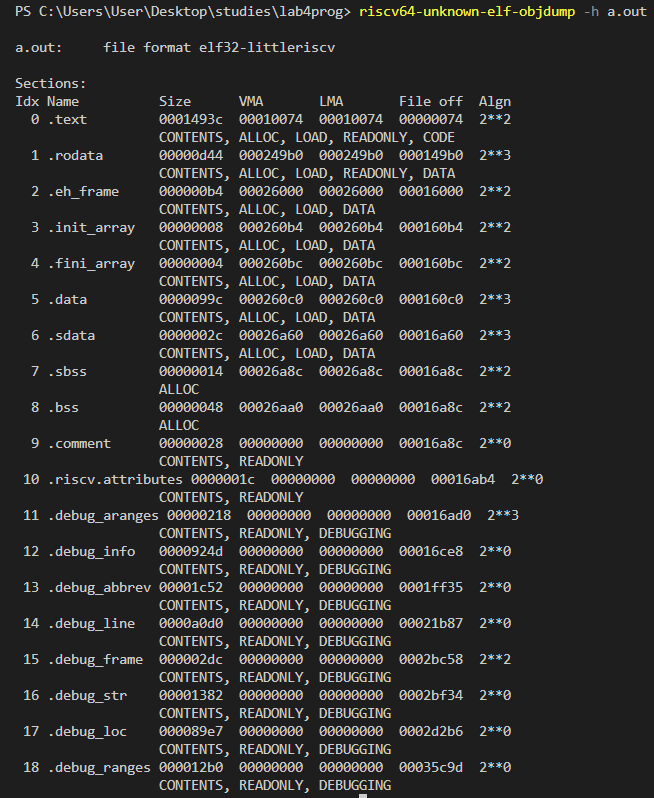
1. Следующим шагом является компоновка и формирование исполняемых фалов программ. В результате получается файл a.out, который также является бинарным, и для его чтения будем пользоваться той же утилитой.

Воспроизведем те же команды, которые исполнялись в прошлом пункте и сравним результаты



Видим, что формат файла – ELF. Также видим наличие флага EXEC\_P – он говорит о том, что файл исполняемый. После загрузки файла его выполнение начинается с адреса 0x00010090

Изучим секции файла:

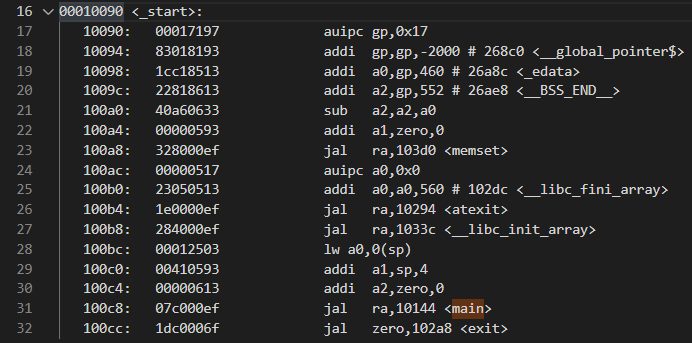


В исполняемом файле производится слияние содержания секций обоих объектных файлов, а также значительное расширяется список секций

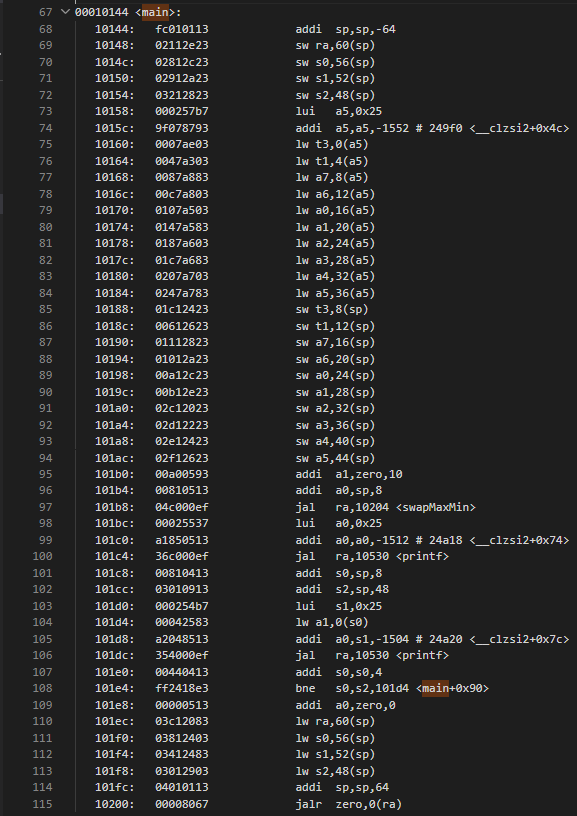
Проанализируем содержимое секции .text исполняемого файла. Для этого воспользуемся командой:

*riscv64-unknown-elf-objdump.exe -j .text -d -M no-aliases a.out >a2*

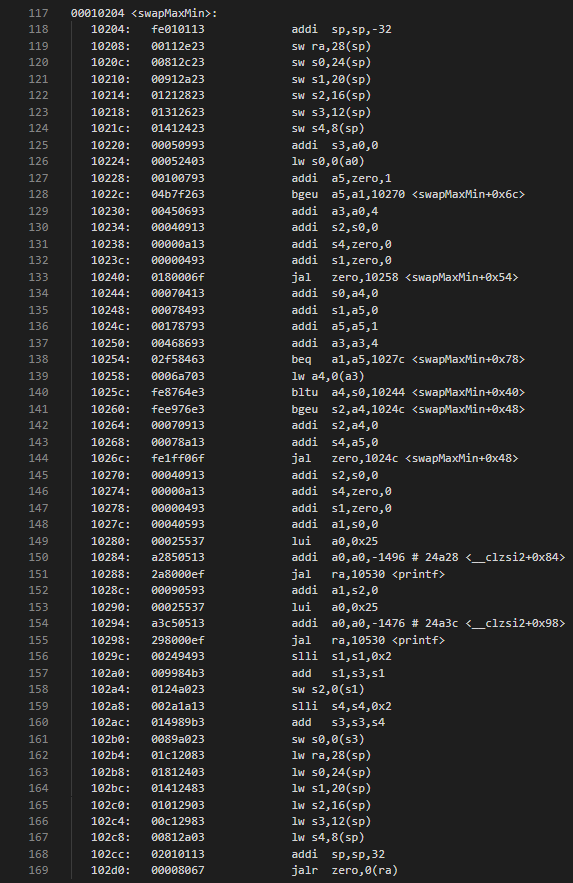
Получили файл a2, изучим его.   
Секция кода теперь содержит большее количество строк (больше 21000 строк), изучим наиболее важные из них.  
Вход в программу осуществляется по метке \_start. Код, начинающийся с метки \_start , обеспечивает инициализацию памяти, регистров процессора и среды времени выполнения, после чего передаёт управление определённой нами функции main.



На 31 строчке управление передается на адрес 10144 что как раз соответствует символу main.



В этом блоке кода можно увидеть что на 97 строчке управление передается на 10204 адрес, что соответствует метке swapMinMax, то есть подпрограмме



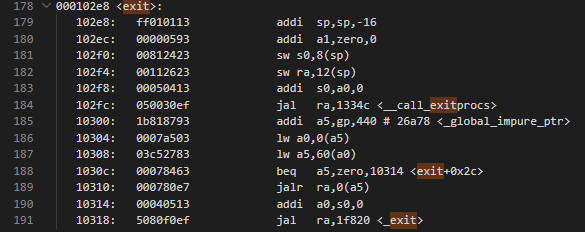
Также в конце блока exit, который вызывается после работы main, можно увидеть переход по метке \_exit 

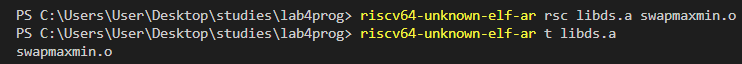
Таблица символов после компоновки также значительно расширилась, получить ее можно при помощи следующей команды (результат в a3)

riscv64-unknown-elf-objdump –t a.out > a3

**Создание и использование статической библиотеки**

Статическая библиотека является, по сути, архивом объектных файлов, среди которых компоновщик выбирает «полезные» для данной программы: объектный файл считается «полезным», если в нем определяется еще не разрешенный компоновщиком символ.

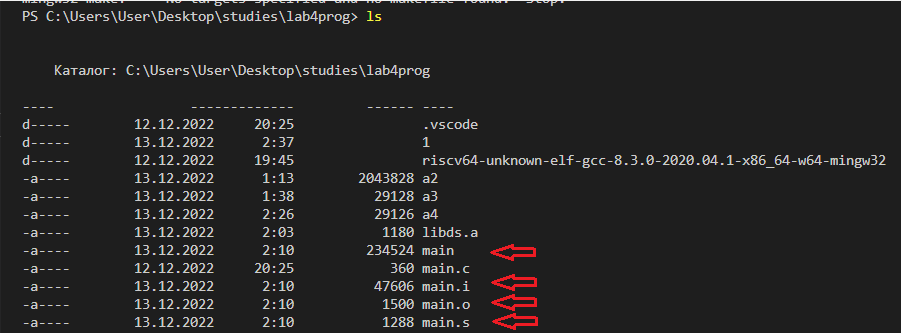
В нашем примере это является единственный файл, содержащий разработанную функцию, которая меняет мин и макс элементы массива местами. Выделим этот файл в статическую библиотеку

****

Сборка проекта при помощи статичной библиотеки выполняется при помощи команды

*riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -O1 --save-temps main.c libds.a -o main*

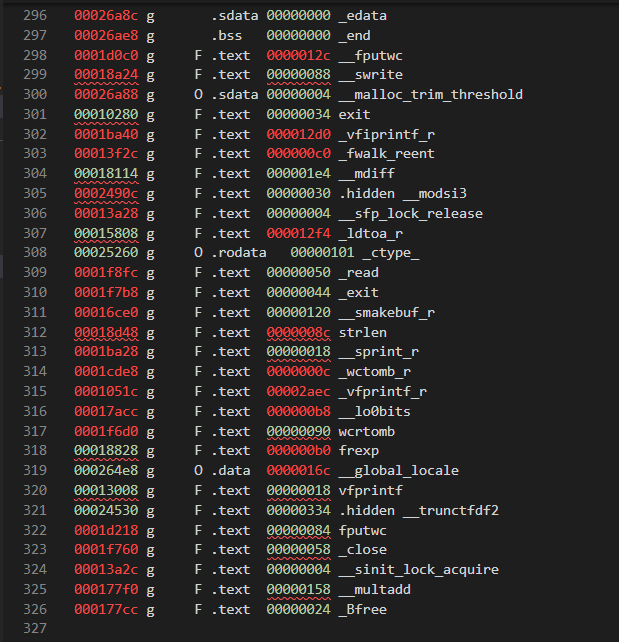
В результате работы в директории появляются файлы main.s main.o main.i и результирующий файл main, это видно на скриншоте (нужные нам файлы созданы одновременно)



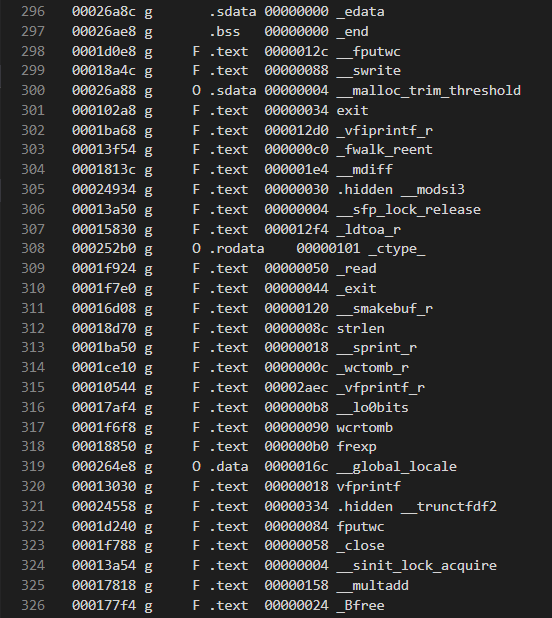
Проанализируем файл main при помощи команды

*riscv64-unknown-elf-objdump -t main > a4*

Сравнивая файлы a4 и a3 (таблица символов при анализе a.out в прошлом пункте) можно сказать, что они, скорее всего, идентичны (по крайней мере по количеству строк, начальной и конечно строчке), а значит сборка работает корректно



Содержимое a4

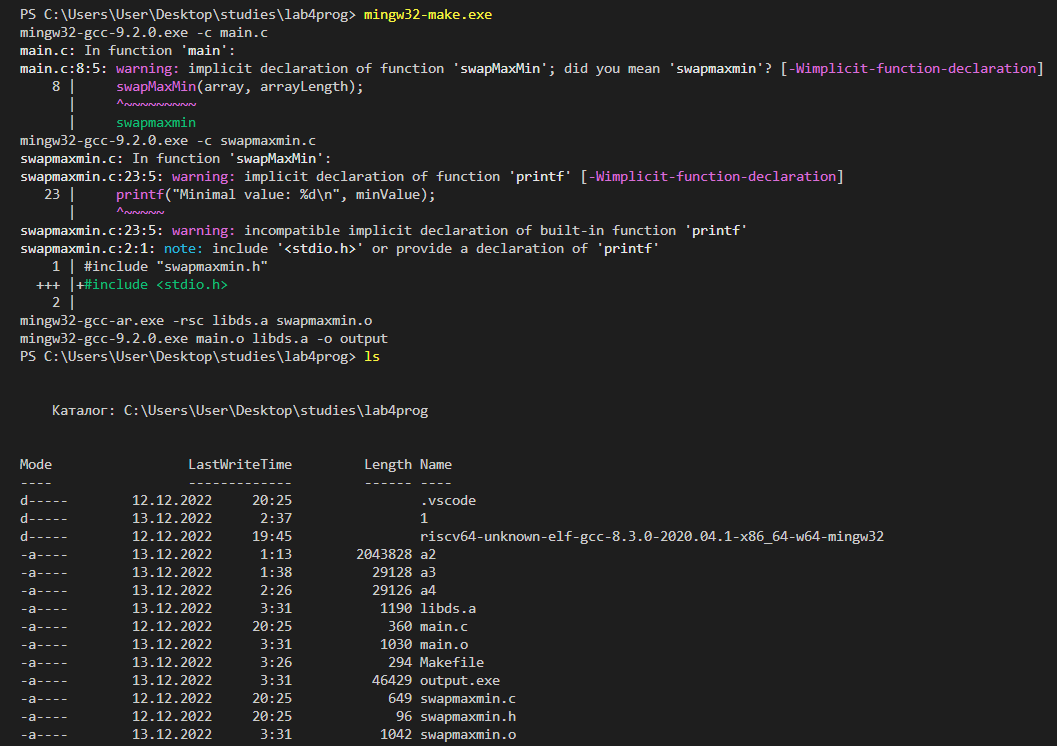


Содержимое a3

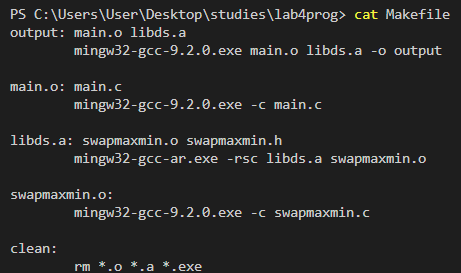
**Запуск Makefile и процесс сборки**

Makefile - это набор инструкций для программы make, которая позволяет собирать проекты, состоящие из большого числа “\*.c” и “\*.h” файлов.

Выполним сборку при помощи makefile



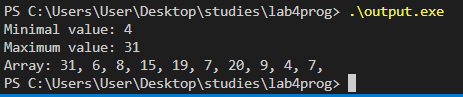
Содержимое makefile



Что происходит в makefile:

1. Создаём объектный файл *main.o* из исходного *main.c*
2. Создаём объектный файл *swapmaxmin.o* из исходного *swapmaxmin.c*
3. Архивируем объектный файл swapmaxmin.o (создаём статическую библиотеку *libds.a*)
4. Компонуем статическую библиотеку *libds.a* с объектным файлом *main.o* и получаем исполняемый файл *output*

Пример работы программы для массива 4, 6, 8, 15, 19, 7, 20, 9, 31, 7



Как видим алгоритм работает верно

**Вывод:** В данной лабораторной работе мы познакомились с процессом сборки проекта на языке C.

Он состоит из:

1. Препроцессирования: исходного *.c* файл препроцессируем в *.i* файл
2. Компиляции: полученный *.i* файл компилируется в файл ассемблера *.s*
3. Ассемблирования: файл *.s* асссемблируется в объектный файл *.o*
4. Компоновки: объектный файл *.o* компонуется в исполняемый файл

Также мы ознакомились в *makefile’ами,* которые упрощают процесс сборки.