Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Лабораторная работа № 4**

**Дисциплина**: Низкоуровневое программирование

**Тема:** Раздельная компиляция

Вариант №3

Выполнил студент гр. 3530901/10003 Дедяев Д.М.

Преподаватель Коренев Д. А.

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

Санкт-Петербург

2022

**Оглавление:**

Формулировка задачи … 3

Вариант задания … 3

Ход решения … 3

Сборка программы по шагам … 5

Препроцессирование … 5

Компиляция … 7

Ассемблирование … 10

Компоновка … 16

Формирование статической библиотеки, разработка make-файлов для сборки библиотеки … 19

Вывод … 21

1. **Формулировка задачи**

1) На языке C разработать функцию, реализующую определенную 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 ее тестовой программы. Проанализировать ход сборки библиотеки и программы, созданные файлы зависимостей.

1. **Вариант задания**

Написать программу для нахождения суммы последовательности чисел в унитарном коде.

1. **Ход решения**

Листинг 1.1. Заголовочный файл findMin.h

Изображение выглядит как текст

Автоматически созданное описание

Листинг 1.2. Основной файл findMin.c

Изображение выглядит как текст, экран, снимок экрана, серебряный

Автоматически созданное описание

Листинг 1.3. Тестовая программа main.c

Изображение выглядит как текст, экран, снимок экрана

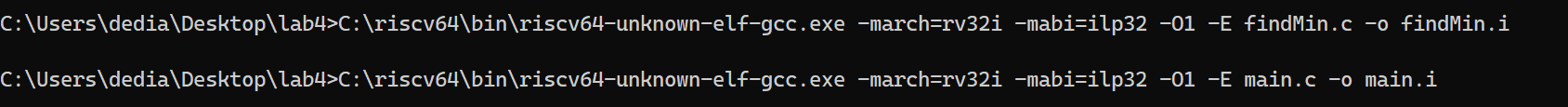
Автоматически созданное описание

В файле “findMin.c” реализована функция findMin(), в которую передаётся массив num и его длина size.  
 Вначале в функции инициализируется переменная minEl, в которую помещается первый элемент массива num. Далее реализован цикл, в ходе которого мы проходим по всем элементам массива и сравниваем их со значением minEl. Если num[i] <minEl, мы присваиваем minEl значение num[i].  
 После реализован ещё один цикл, в ходе которого мы снова проходимся по массиву и вычитаем значение minEl из каждого элемента.

Заголовочный файл “findMin.h” содержит в себе определение функции findMin(). В дальнейшем для использования этой функции в другой программе необходимо организовывать подключение этого заголовочного файла, а также компиляцию исходного файла “findMin.c” вместе с использующей ее программой.

1. **Сборка программы по шагам** 
   1. **Препроцессирование**

Начнем сборку созданных программ на языке C по шагам. Первым шагом является препроцессирование файлов исходного текста “findMin.c” и “main.c” в файлы “findMin.i” и “main.i”:



Драйвер компилятора gcc– riscv64-unknown-elf-gcc– запускается с параметрами командной строки “-march=rv32i -mabi=ilp32”, указывающих что целевым является процессор с базовой архитектурой системы команд RV32I; -O1 – указание выполнять простые оптимизации генерируемого кода; -E – указание остановить процесс сборки после препроцессирования.

Поскольку в данном примере не использовались директивы препроцессора (начинающиеся в языке C с символа “#”), результат работы препроцессора (файл “main.i”) мало отличается от исходного файла “main.c”:

Листинг 2.1. Файл main.i (фрагменты)



Изображение выглядит как текст, экран, снимок экрана

Автоматически созданное описание

Листинг 2.2. Файл findMin.i

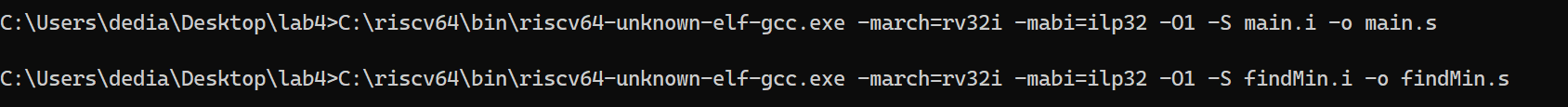
Изображение выглядит как текст

Автоматически созданное описание

Появившиеся нестандартные директивы, начинающиеся с символа “#”, используются для передачи информации об исходном тексте из препроцессора в компилятор. Так, в файле “main.i” четвертая директива «# 1 “main.c”» информирует компилятор о том, что следующая строка является результатом обработки строки 1 исходного файла “main.c”. В этой строке стояла команда #include "findMin.h", поэтому препроцессор произвел вставку содержимого этого заголовочного файла, то есть определение функции findMin(). Далее же начинается описание самого содержимого файла, что происходит после директивы # 2 "main.c" 2. Исходный код тестирующей функции main() после работы препроцессора остался без изменений, как и исходный код функции findMin().

* 1. **Компиляция**

Следующим шагом является компиляция файлов “findMin.i” и “main.i” в код на языке ассемблера “findMin.s” и “main.s”:



Драйвер компилятора riscv64-unknown-elf-gcc запускается с параметрами командной строки “-march=rv32i -mabi=ilp32”, указывающих что целевым является процессор с базовой архитектурой системы команд RV32I; -O1 – указание выполнять простые оптимизации генерируемого кода; -S – указание остановить процесс сборки после компиляции (без запуска ассемблера).

Проанализируем получившийся код на языке ассемблера:

Листинг 2.3. Файл findMin.s

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

По метке findMin начинается тело самой функции.

Листинг 2.4. Файл main.s

Изображение выглядит как текст

Автоматически созданное описание

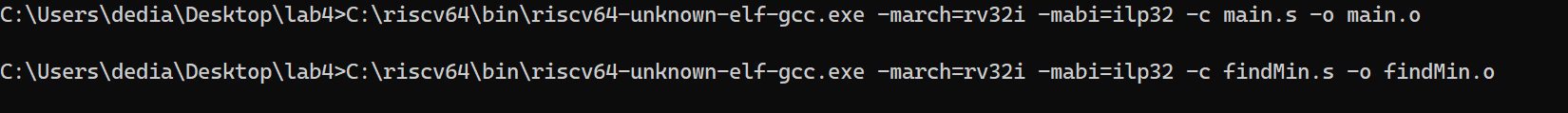
Изображение выглядит как текст

Автоматически созданное описание

По метке main начинается тестовая программа, вызывающая функцию findMin().

* 1. **Ассемблирование**

Следующим шагом является ассемблирование файлов “findMin.s” и “main.s” в объектные файлы “findMin.o” и “main.o”:



Драйвер компилятора riscv64-unknown-elf-gcc запускается с параметрами командной строки “-march=rv32i -mabi=ilp32”, указывающих что целевым является процессор с базовой архитектурой системы команд RV32I; -c – указание остановить процесс сборки после ассемблирования.

Объектный файл не является текстовым, для изучения его содержимого используем утилиту objdump:

Изображение выглядит как текст

Автоматически созданное описание

Оба файла содержат таблицу перемещений (в списке флагов есть флага HAS\_RELOC).

Выведем все заголовки секций объектных файлов (команда riscv64-unknown-elf-objdump -h findMin.o):

Листинг 2.5. Заголовки секций файлов main.o и findMin.o

Изображение выглядит как текст

Автоматически созданное описание

Далее представлены таблицы символов файлов findMin.o и main.o:

Листинг 2.6. Таблица символов файлов main.o и findMin.o

Изображение выглядит как текст

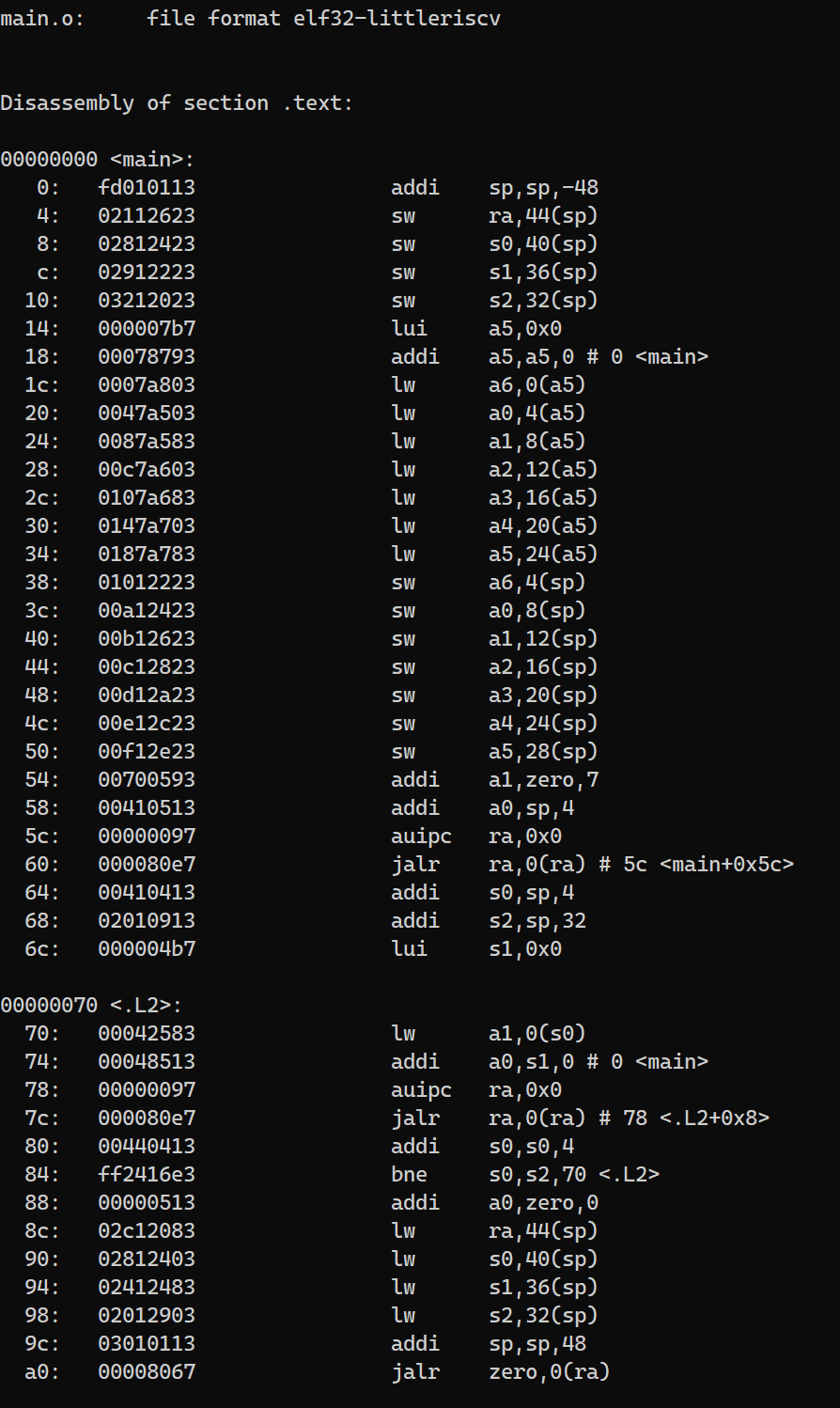
Автоматически созданное описание

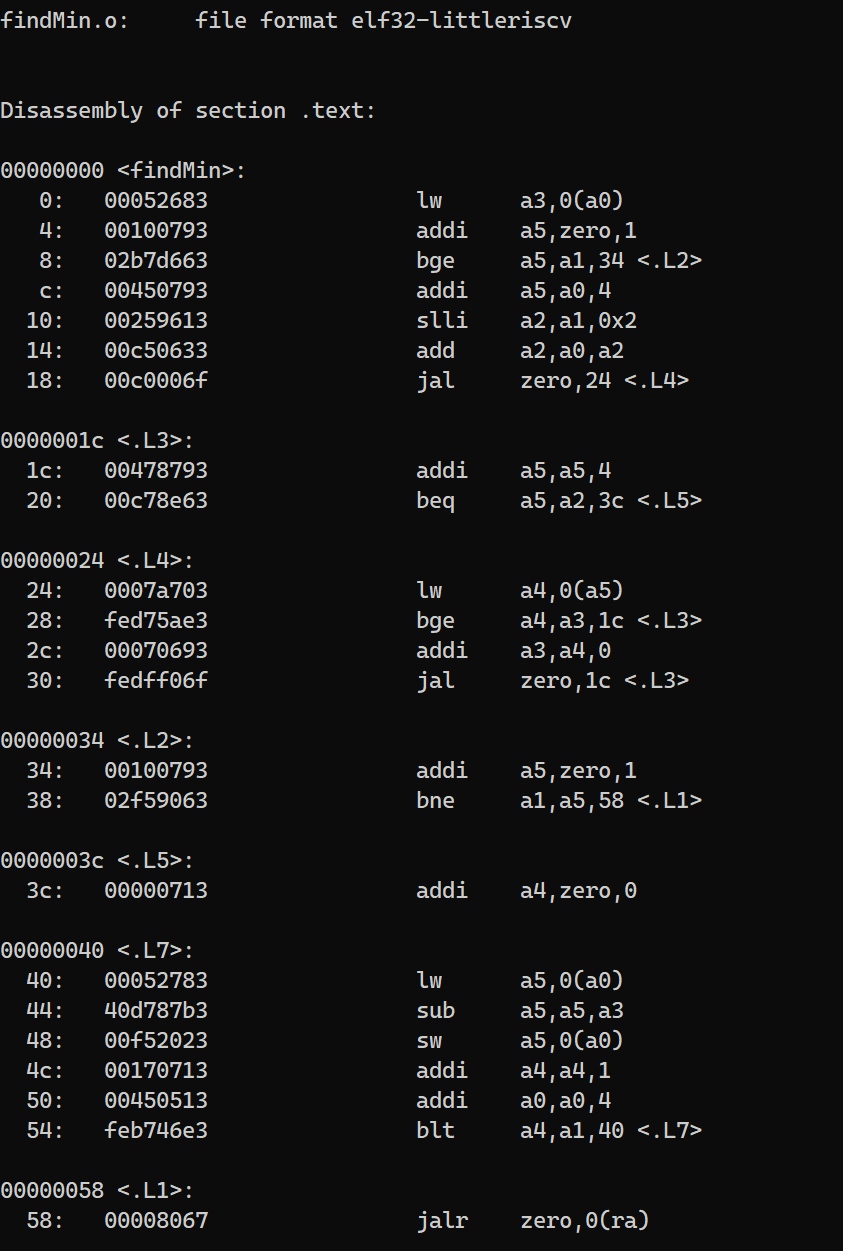
В таблице символов main.o имеется запись: символ “findMin” типа \*UND\*. Эта запись означает, что символ “findMin” использовался в ассемблерном коде, из которого был получен данный объектный файл, но не был определен; ассемблер сделал вывод о том, что символ должен быть определен где-то еще (адреса вызовов этих функций будут подставлены во время компоновки при помощи таблицы перемещений) и отразил это в таблице символов. То же самое относится и к символу “printf”.

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

Проанализируем секции .text объектных файлов – секций кода, в которых содержатся коды инструкций:

Листинг 2.7. Секции .text файлов findMin.o и main.o





Дизассемблированный код практически идентичен сгенерированному (за исключением псевдоинструкций).

Секции .data объектных файлов – секции инициализированных данных – не содержат данных, размер секций, как было выведено выше, равен нулю:

Листинг 2.8. Секции .data файлов findMin.o и main.o

Изображение выглядит как текст

Автоматически созданное описание

Секции .bss объектных файлов – секции данных, инициализированных нулями – таким же образом пусты:

Листинг 2.9. Секции .bss файлов findMin.o и main.o

Изображение выглядит как текст

Автоматически созданное описание

Секция .comment – секция данных о версиях – и для одного и для другого файла содержит одни и те же значения – сведения о GCC версии 8.3.0 от SiFive:

Листинг 2.10. Секции .comment файлов findMin.o и main.o

Изображение выглядит как текст

Автоматически созданное описание

Секция .riscv.attributes и обоих объектных файлов содержит одну и ту же информацию об используемой архитектуре команд RV32I:

Листинг 2.11. Секции . riscv.attributes файлов findMin.o и main.o

Изображение выглядит как текст, снимок экрана, монитор

Автоматически созданное описание

Проанализируем таблицы перемещений объектных файлов:

Листинг 2.12. Таблицы перемещений файлов findMin.o и main.o

Изображение выглядит как текст

Автоматически созданное описание

В таблицах перемещения findMin.o содержится информация о переходах (R\_RISCV\_JAL) и ветвлениях(R\_RISCV\_BRANCH). В таблицах перемещения main.o, есть R\_RISCV\_CALL, чтобы информация о переходах соответствовала findMIn. Записи типа R\_RISCV\_RELAX заносятся в таблицу перемещений в дополнение к записям типа R\_RISCV\_CAL.

* 1. **Компоновка**

Компоновка осуществляется следующей командой:

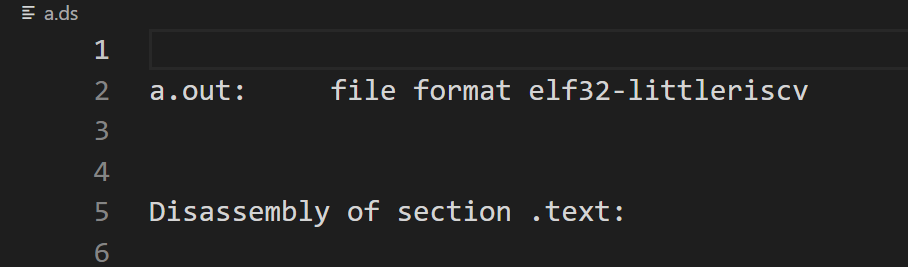


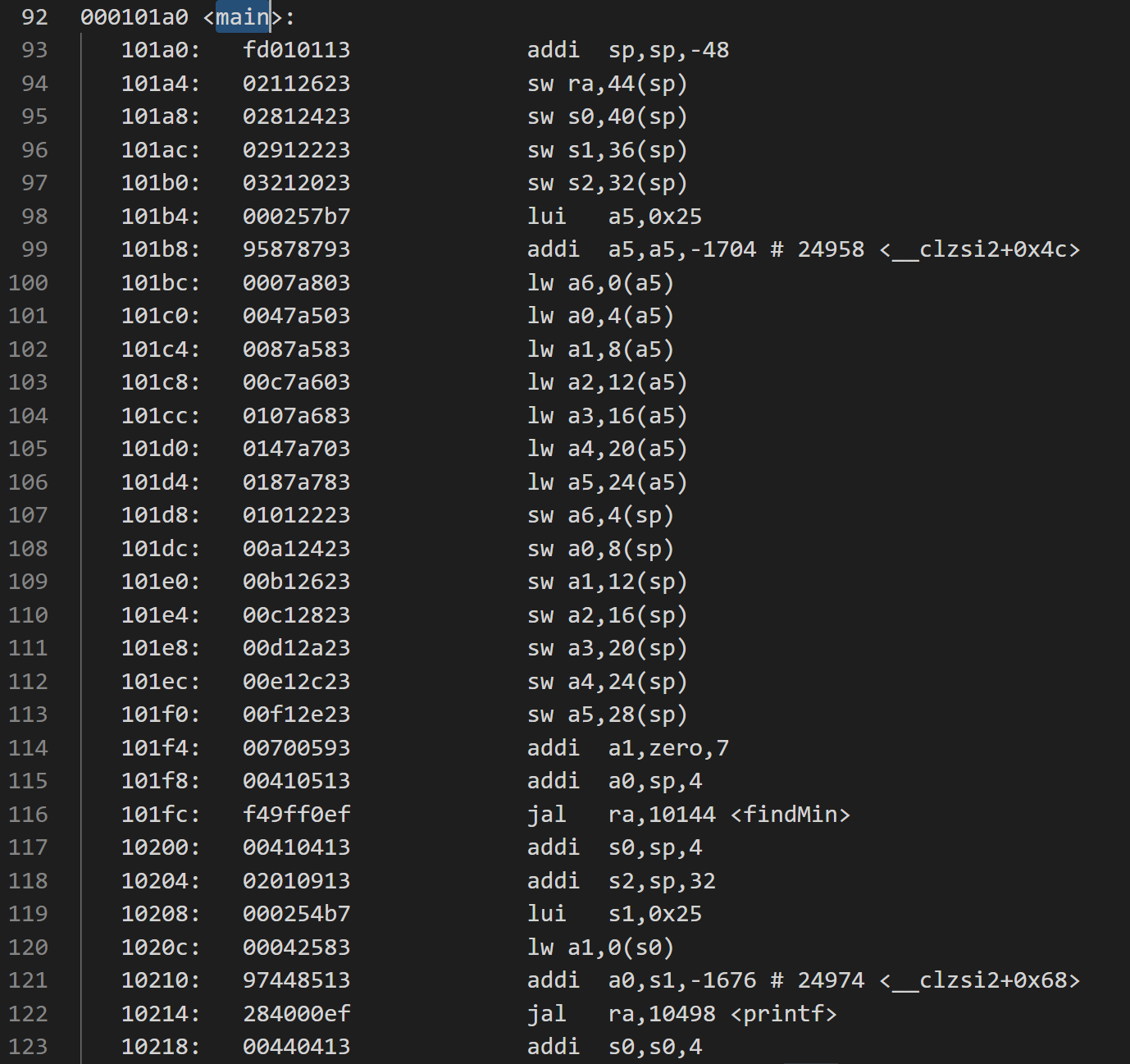
В результате получаем исполняемый файл main, скомпилированный под архитектуру risc-v. Произведём дизассемблирование полученного бинарного файла и обратим внимание на метки, соответствующие функциям.

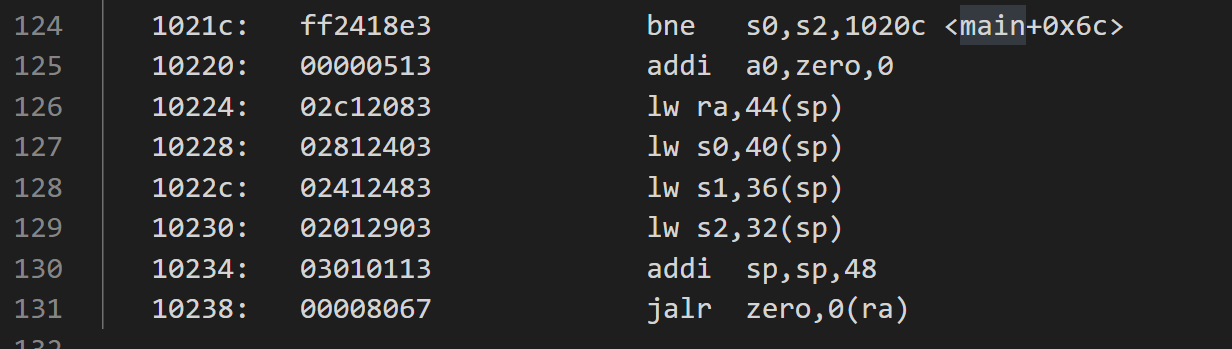
Команда, используемая для дизассемблирования:

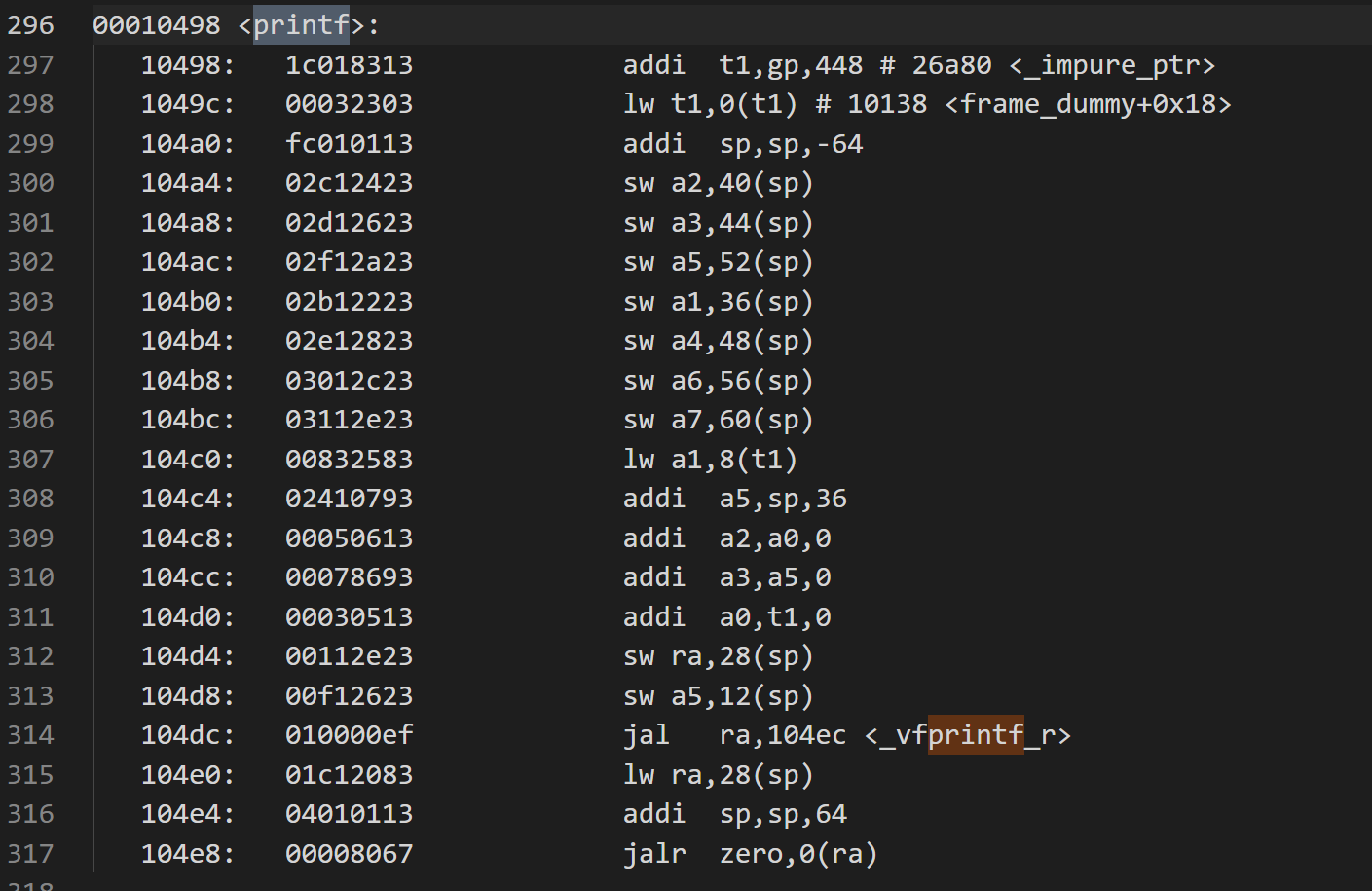


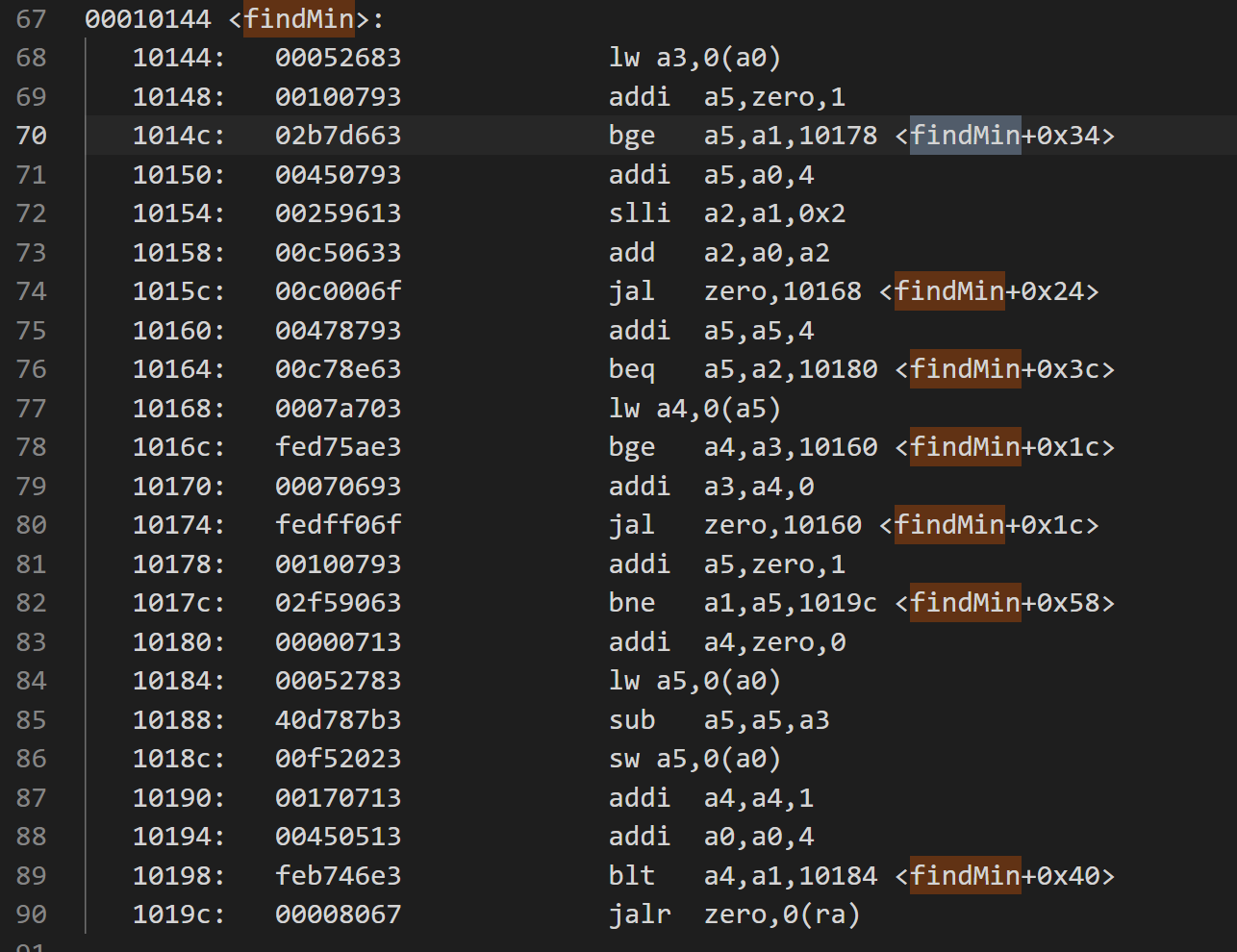
Листинг 2.13. Исполняемый файл a.ds (фрагмент)











В дизассемблированном файле видим, что при компоновке была произведена «склейка» всего исполняемого кода из всех компонуемых объектных файлов, а также были просчитаны и расставлены все адреса.

Проанализируем таблицу перемещений исполняемого файла:

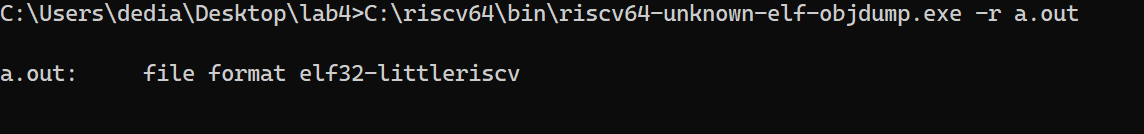
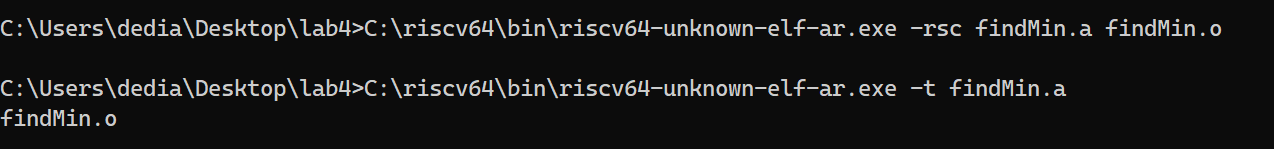


Таблица перемещений оказывается пуста, все необходимые релокации, оптимизации и замены инструкций были успешно проведены компоновщиком.

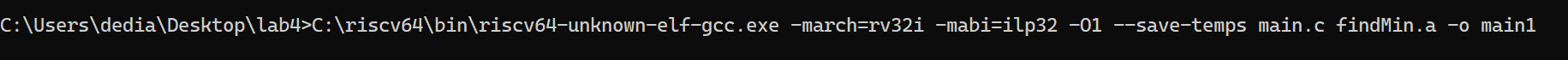
Итогом сборки программ на языке C по шагам является исполняемый на процессорах архитектуры RISC-V файл, решающий заданную вариантом задачу и проверяющий корректность решения этой задачи на тестовом примере.

1. **Формирование статической библиотеки, разработка make-файлов для сборки библиотеки**

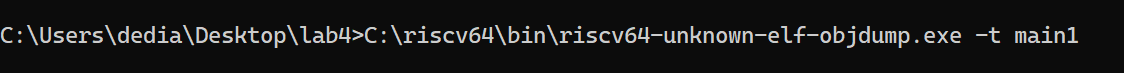
Статическая библиотека (static library) является, по сути, архивом (набором, коллекцией) объектных файлов, среди которых компоновщик выбирает «полезные» для данной программы: объектный файл считается «полезным», если в нем определяется еще не разрешенный компоновщиком символ. Разработанная функция поиска максимального простого числа до числа n содержится в единственном исходном файле на языке C. Выделим этот файл в статическую библиотеку:



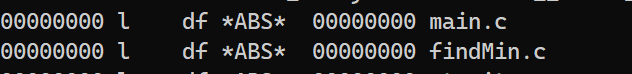
Используя получившуюся библиотеку, соберем исполняемый файл программы следующей командой:



Изучим таблицы символов полученных исполняемых файлов:



Листинг 3.1. Таблица символов исполняемого файла (фрагмент)



Можно заметить, что в состав программы вошло содержимое объектного файла findMin.o

Процесс выполнения команд выше можно заменить make-файлами, которые произведут создание библиотеки и сборку программы.

**Makefile** - это набор инструкций для программы make, которая позволяет собирать проекты, состоящие из большого числа “\*.c” и “\*.h” файлов. Обычно эта программа используется в связке с системами сборки, например cmake, позволяя вести проекты модульно (т.е. проект с включенными подпроектами).

Сборка с помощью Makefile:

Листинг 4.1. Сборка с помощью Makefile

Изображение выглядит как текст

Автоматически созданное описание

Демонстрация работы програмы:

Листинг 4.2. Результат вывода программы

Изображение выглядит как текст

Автоматически созданное описание

Цель для Make – очистить:

Листинг 4.3. Результат выполнения clean

Изображение выглядит как текст

Автоматически созданное описание

Листинг 4.4. Makefile

Изображение выглядит как текст

Автоматически созданное описание

Что происходит в Makefile:

1. Создаём объектный файл *main.o* из исходного *main.c*
2. Создаём объектный файл *findMin.o* из исходного *findMin.c*
3. Архивируем объектный файл findMin.o (создаём статическую библиотеку *findMin.a*)
4. Компонуем статическую библиотеку *findMin.a* с объектным файлом *main.o* и получаем исполняемый файл *output*
5. **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. Утилита Make позволяет собирать проекты, состоящие из большого количества файлов, вместо использования PS/SH скриптов, и прописывания файлов вручную.