Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа информатики и вычислительной техники

**Лабораторная работа № 4**

**Дисциплина**: Низкоуровневое программирование

**Тема:** Раздельная компиляция

Выполнил студент гр. 3530901/10005 Ухов А.Д.

Преподаватель Коренев Д. А.

«30» ноября 2022 г.

Санкт-Петербург

2022

Оглавление

Техническое задание

1. Программа на языке C

2. Сборка программы «по шагам»

Препроцессирование

Компиляция

Ассемблирование

Компоновка

3. Создание статической библиотеки и make-файлов

Вывод

Техническое задание

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.

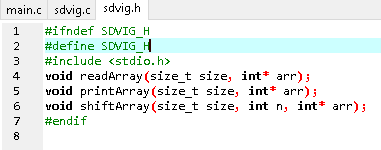
3) Выделить разработанную функцию в статическую библиотеку. Разработать make-файлы для сборки библиотеки и использующей ее тестовой программы. Проанализировать ход сборки библиотеки и программы, созданные файлы зависимостей.

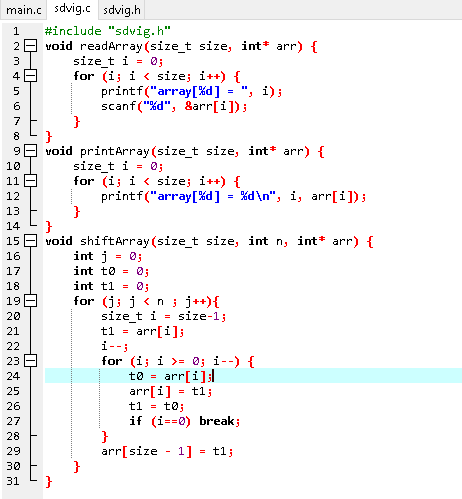
1. **Вариант задания**

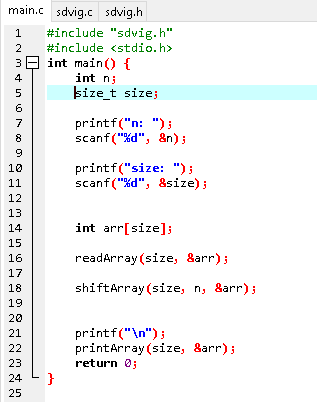
#10 Циклический сдвиг массива чисел на заданное количество разрядов влево

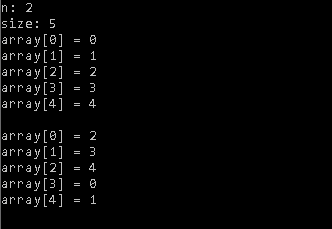
1. **Ход решения**

Листинг 1.1. Заголовочный файл sdvig.h

Листинг 1.2. Основной файл sdvig.c

Листинг 1.3. Тестовая программа main.c



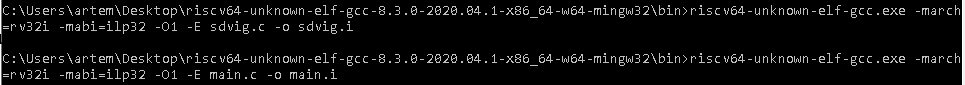


**2. Сборка программы по шагам**

Предпроцесирование

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

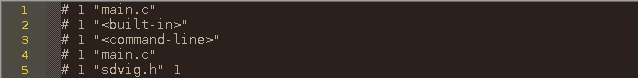
Препроцессирование, компиляция и ассемблирование

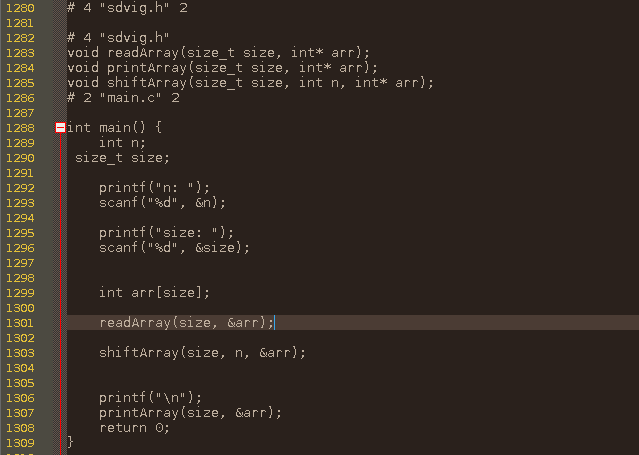


Драйвер компилятора gcc– riscv64-unknown-elf-gcc– запускается с параметрами командной строки “-march=rv32i -mabi=ilp32”, указывающих что целевым является процессор с базовой архитектурой системы команд RV32I;-O1 – указание выполнять простые оптимизации генерируемого кода; -E – указание остановить процесс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

В файлах main.i и sdvig.i содержится результат препроцессирования

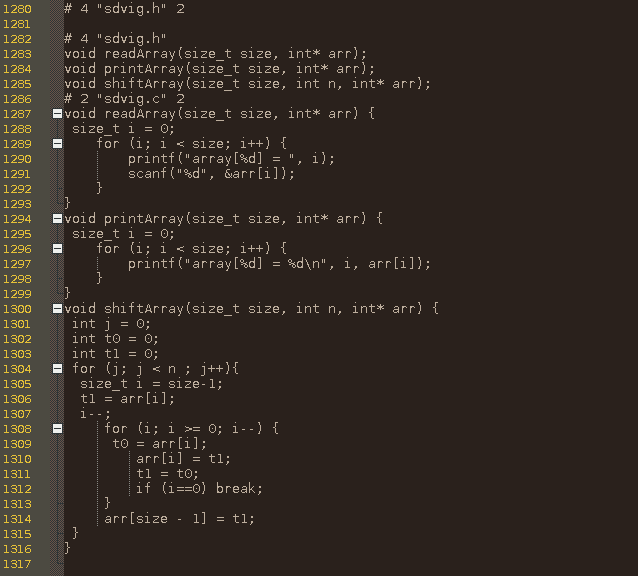
Листинг 2.1. Файл main.i





Листинг 2.2. Файл sdvig.i



Появившиеся нестандартные директивы, начинающиеся с символа “#”, используются для передачи информации об исходном тексте из препроцессора в компилятор.

Компиляция

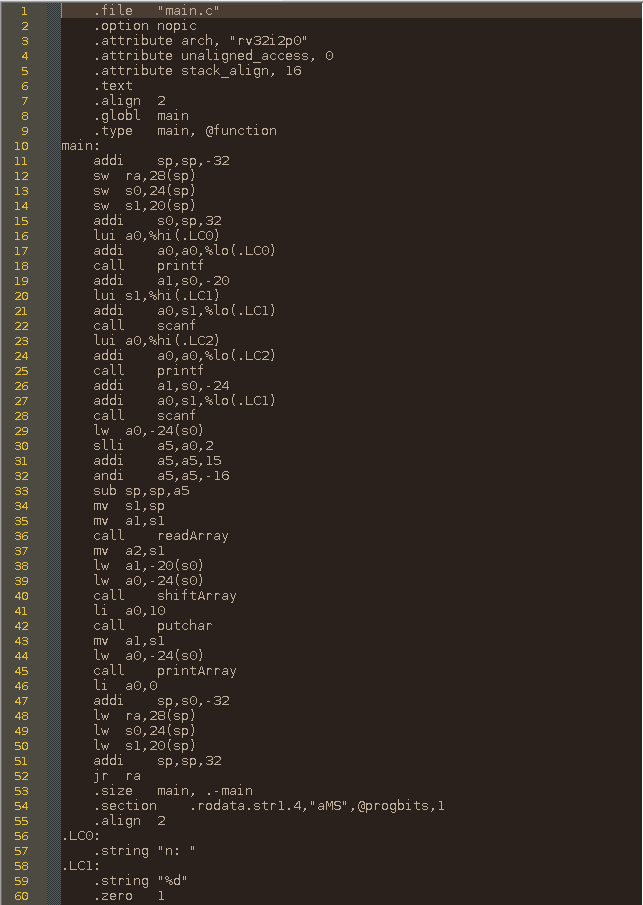
Следующим шагом является компиляция файлов “sdvig.i” и “main.i” в код на языке ассемблера “sdvig.s” и “main.s”:

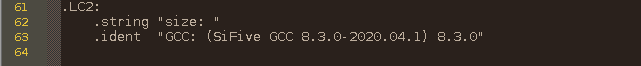




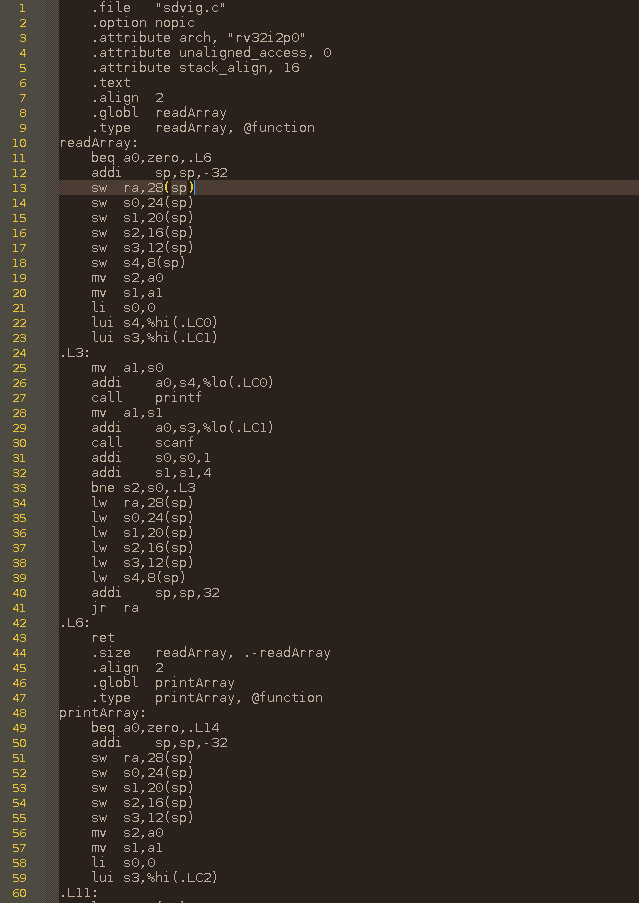
Драйвер компилятора riscv64-unknown-elf-gcc запускается с параметрами командной строки “-march=rv32i -mabi=ilp32”, указывающих что целевым является процессор с базовой архитектурой системы команд RV32I;-O1 – указание выполнять простые оптимизации генерируемого кода; -S – указание остановить процесс сборки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

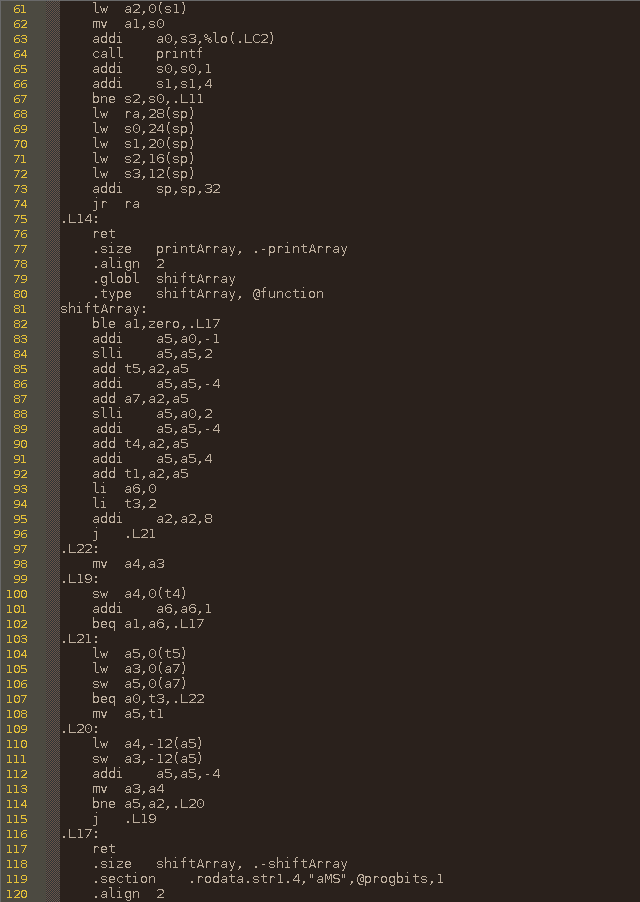
Листинг 2.3. Файл main.s

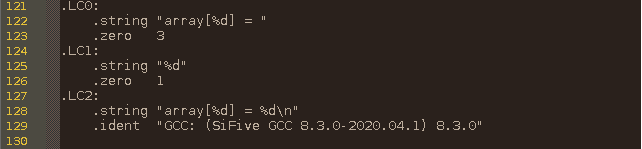




Листинг 2.4. Файл sdvig.s

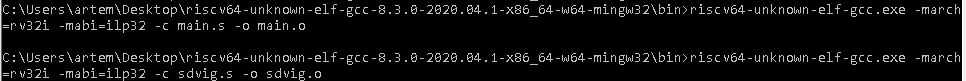


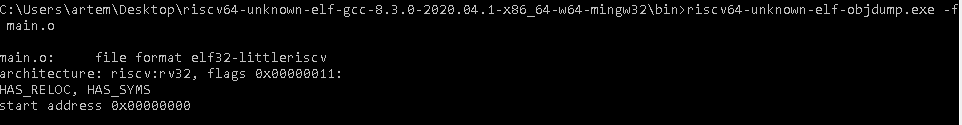


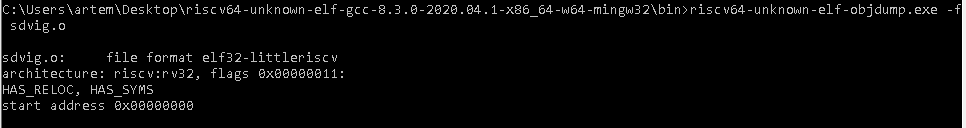


Ассемблирование

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

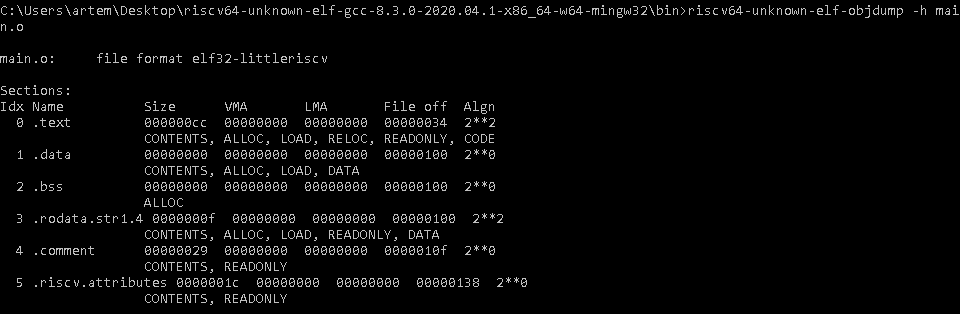
Драйвер компилятора riscv64-unknown-elf-gcc запускается с параметрами командной строки “-march=rv32i -mabi=ilp32”, указывающих что целевым является процессор с базовой архитектурой системы команд RV32I; -c – указание остановить процесс сборки после ассемблирования.

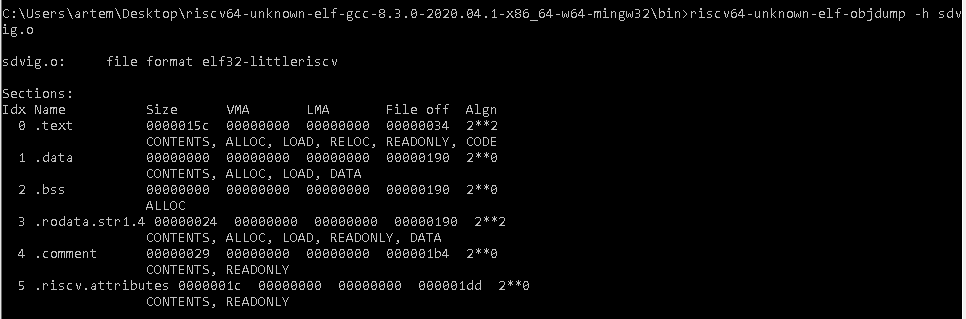
Объектный файл не является текстовым, для изучения его содержимого используемутилиту objdump:



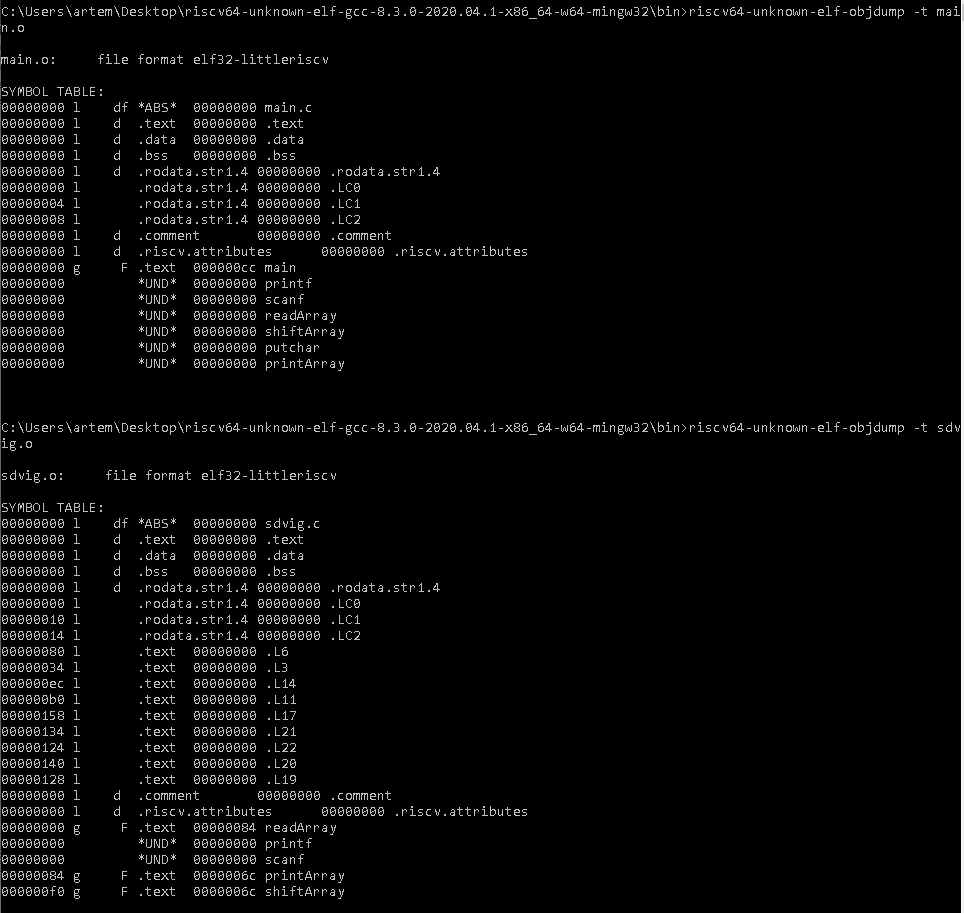
Оба файла содержат таблицу перемещений (в списке флагов есть флага HAS\_RELOC)

Выведем все заголовки секций объектных файлов





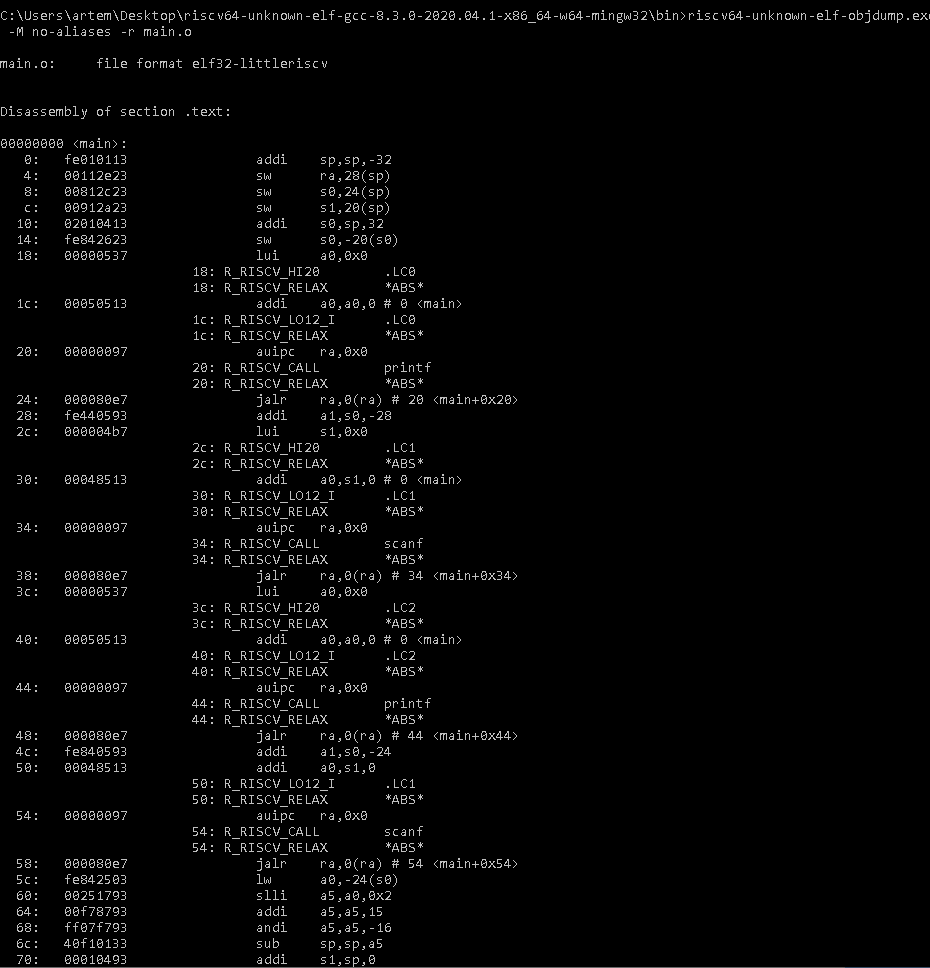
Выведем таблицы символов файлов sdvig.o и main.o

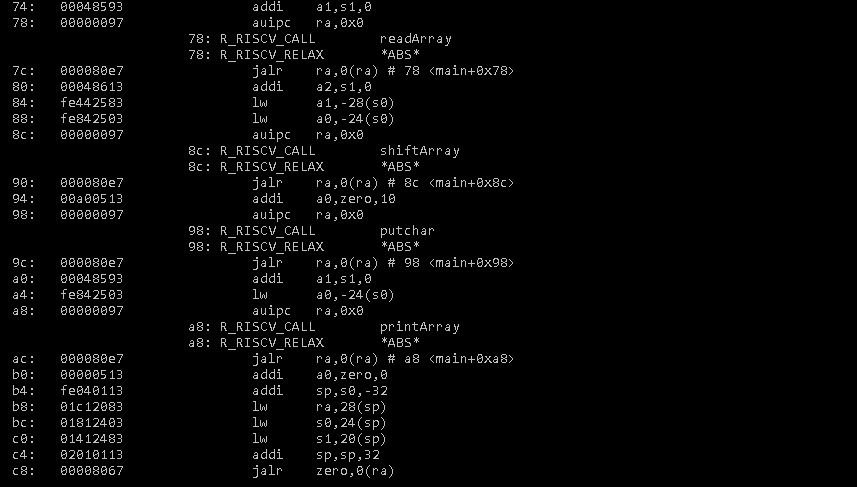


В каждой таблице только одни глобальный (флаг “g”) символ типа «функция» (“F”) – созданные фцнкции

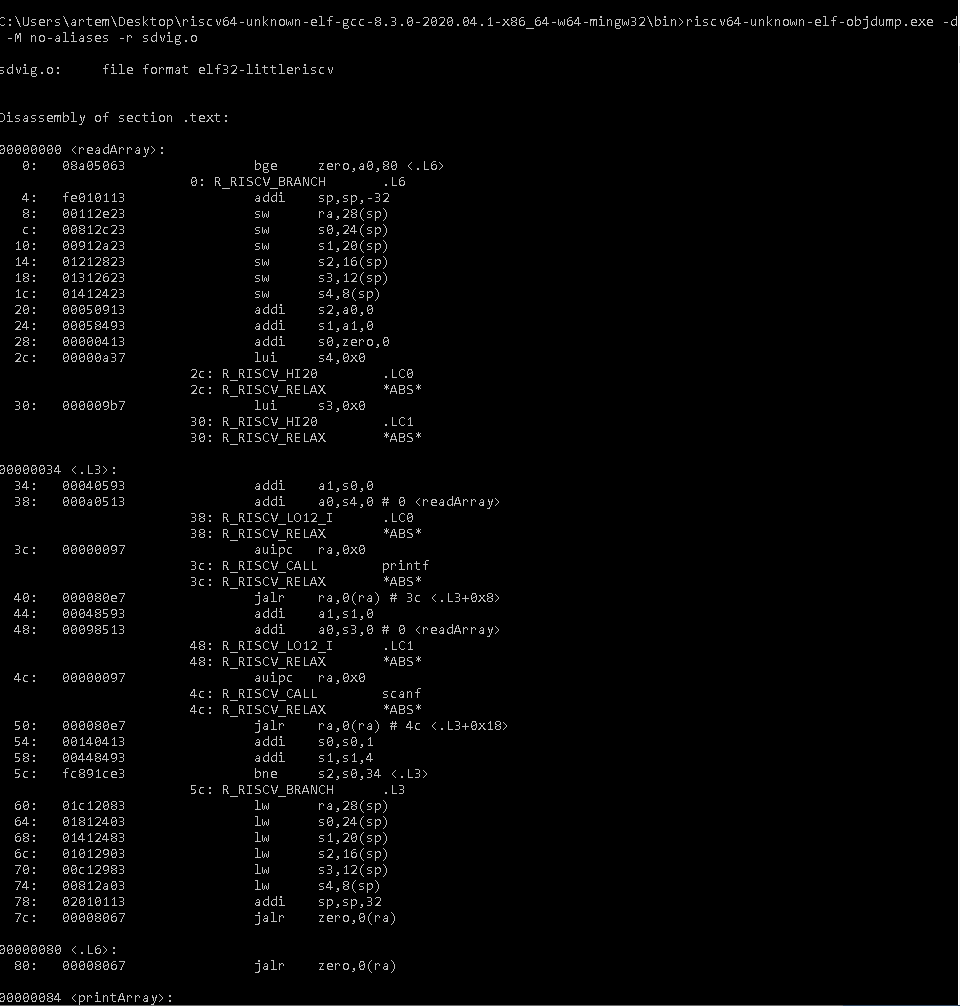
Листинг 2.9. файл main.o

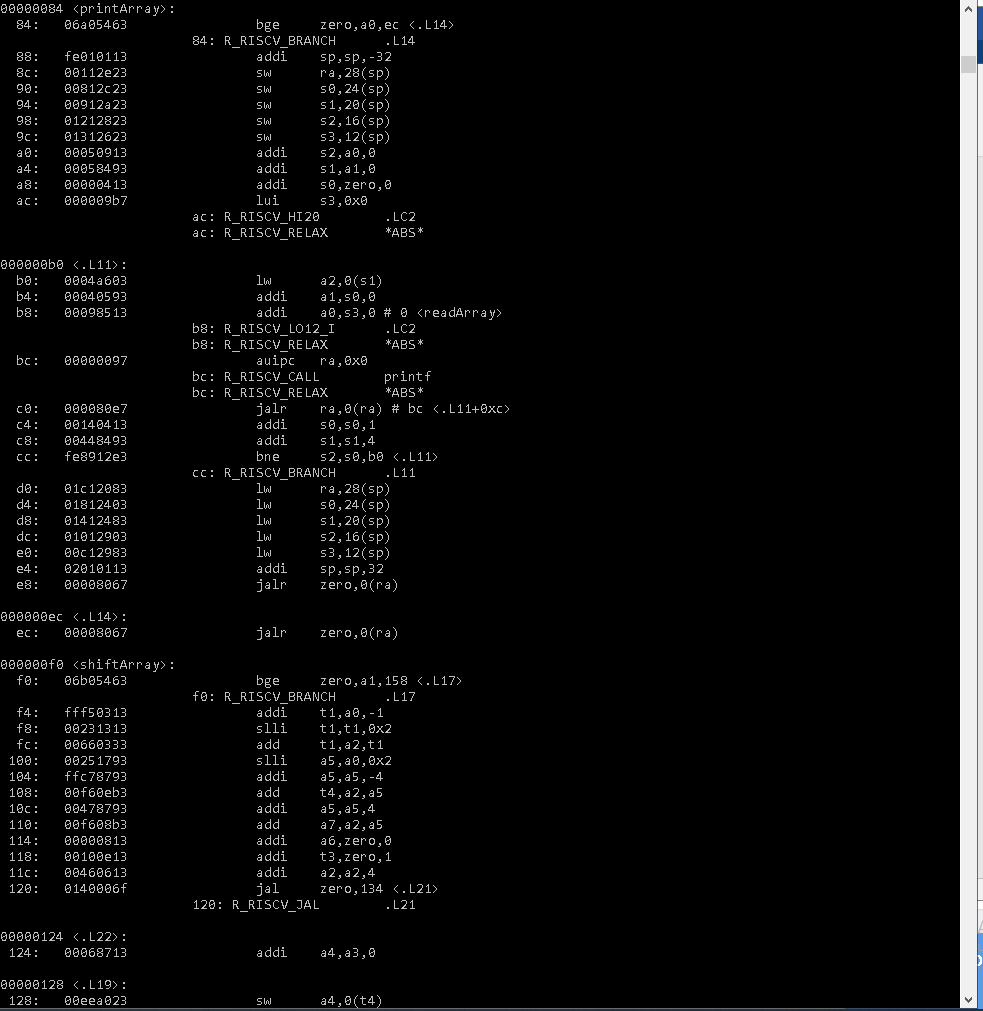
Выведем секции .text объектных файлов – секций кода, в которых содержатся коды инструкций:

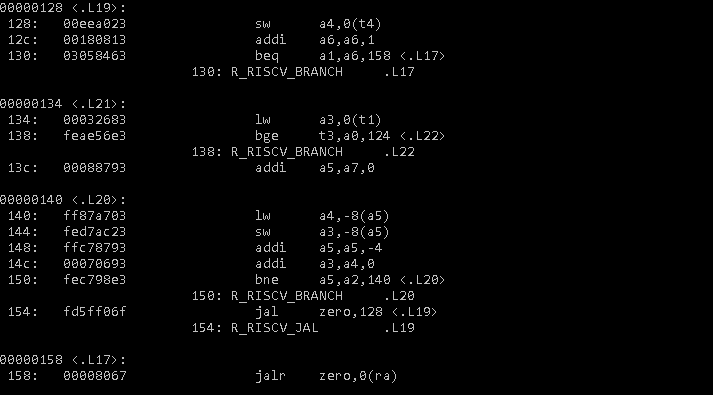




Листинг 2.10. файл sdvig.o

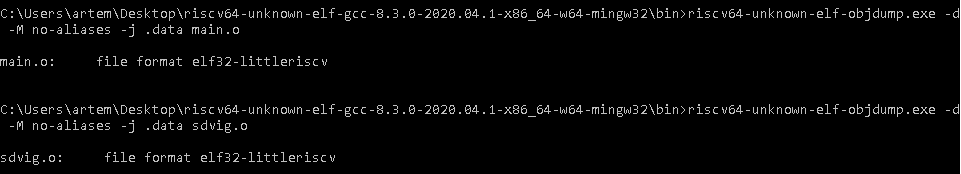




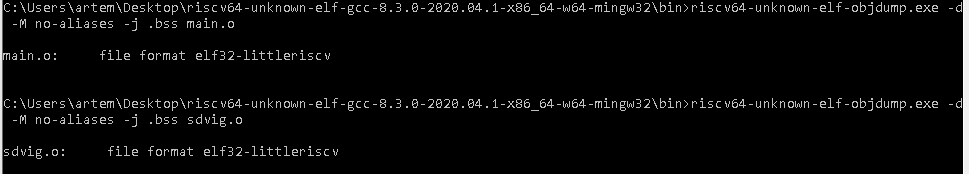


Дизассемблированный код практически идентичен сгенерированному (за исключением псевдоинструкций).

Секции .data объектных файлов – секцииинициализированных данных – не содержат данных, размер секций равен нулю:

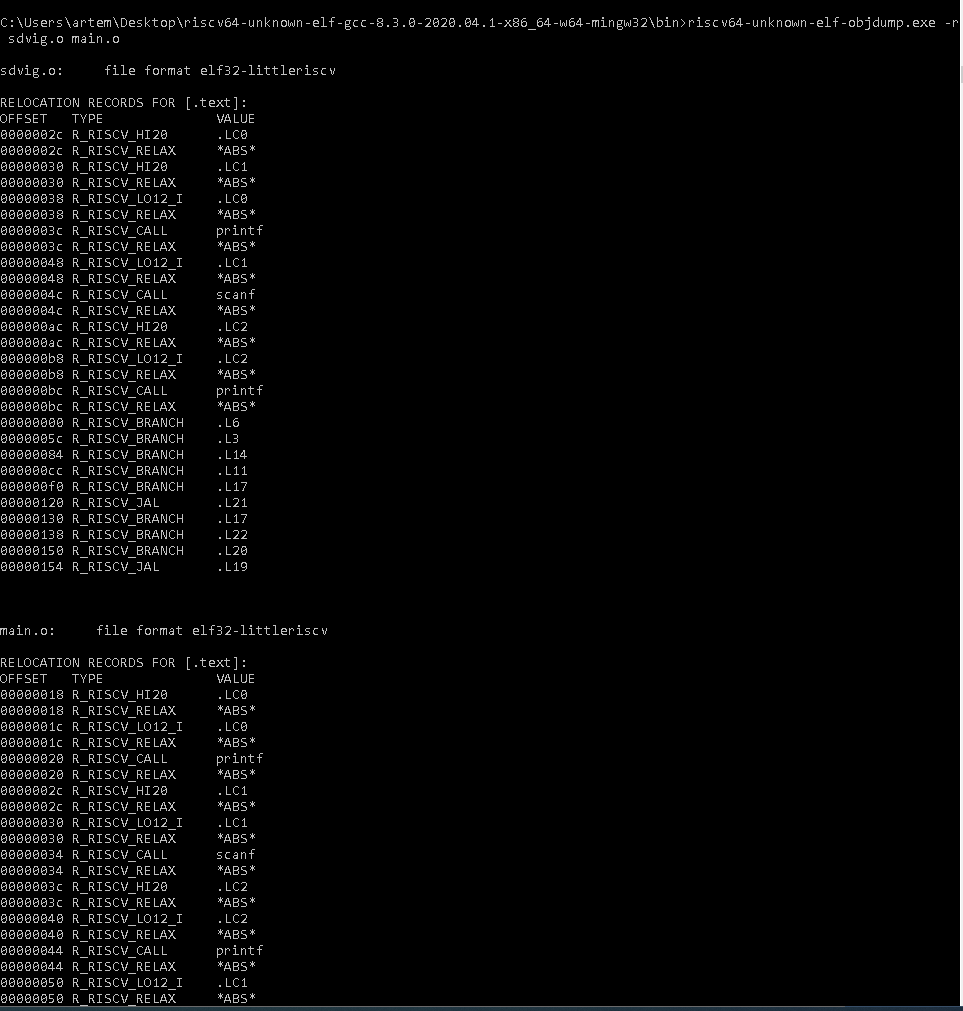


Секции .bss объектных файлов – секции данных, инициализированных нулями – таким же образом пусты:



Листинг 2.11. таблицы перемещений sdvig.o

Выведем таблицы перемещений объектных файлов:



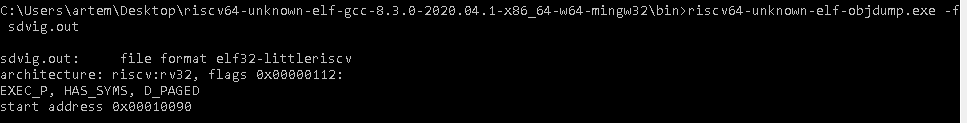


Компановка

Следующим шагом является компоновка и формирование исполняемых фалов программ:

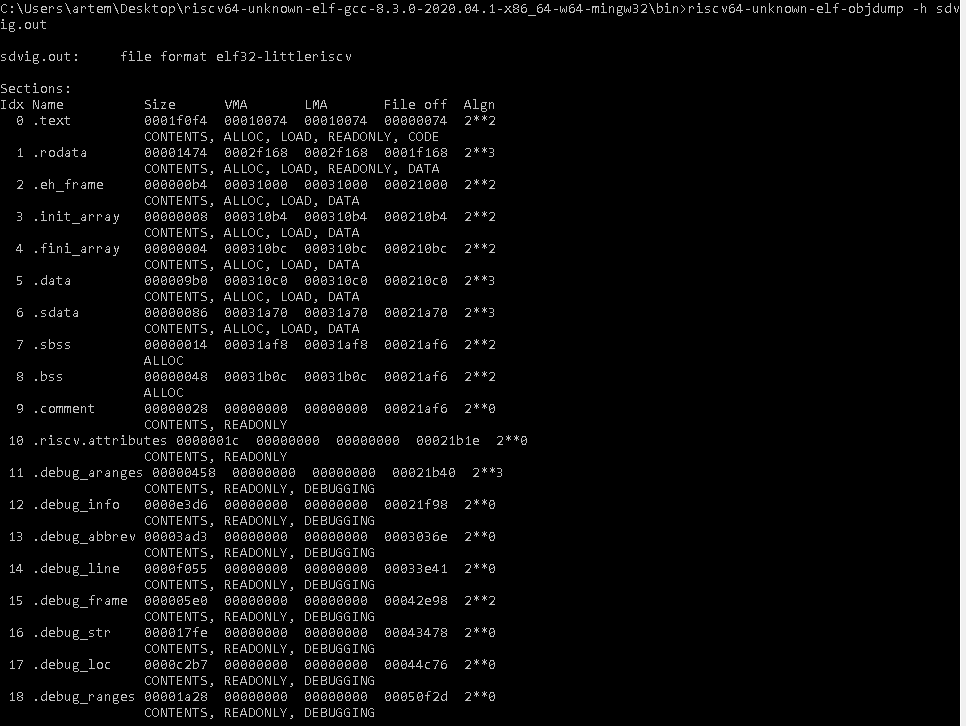


Сформированный компоновщиком файл “sdvig.out”, также является «бинарным»:



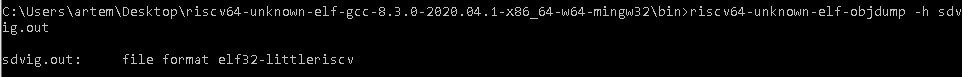
Флаг EXEC\_Pуказывает, что файл действительно является исполняемым, после загрузки его выполнение должно начаться с адреса 0x0001008c (entrypoint).

Перечислим секции исполняемого файла:

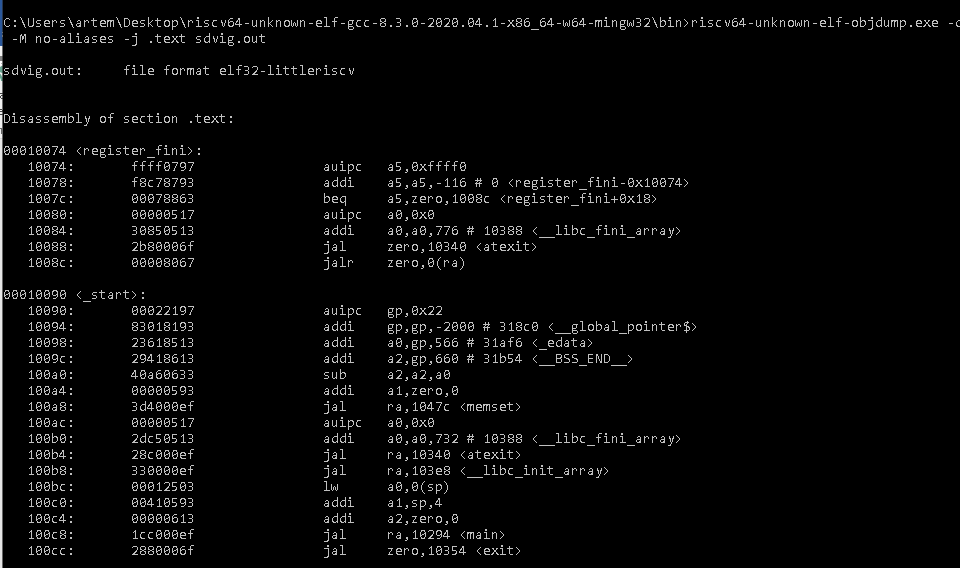


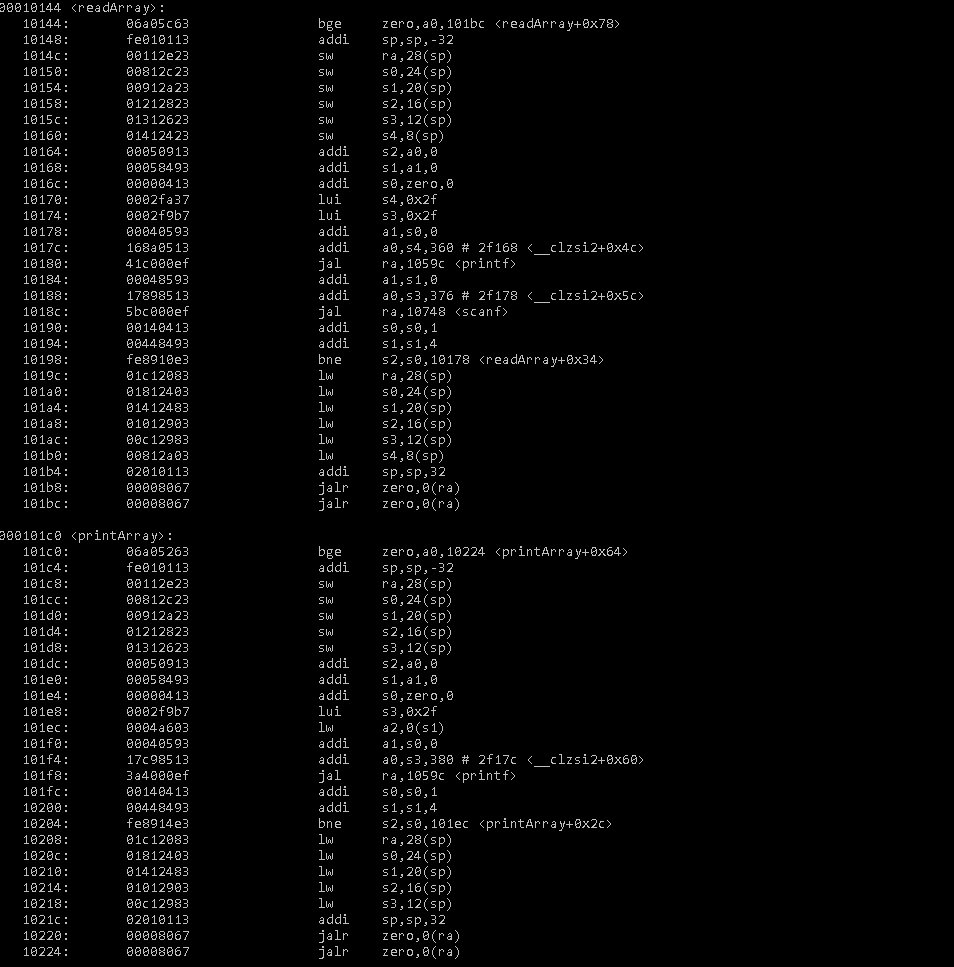
В исполняемом файле действительно производится слияние содержания секций обоих объектных файлов, а также значительное расширение списка секций новыми блоками.

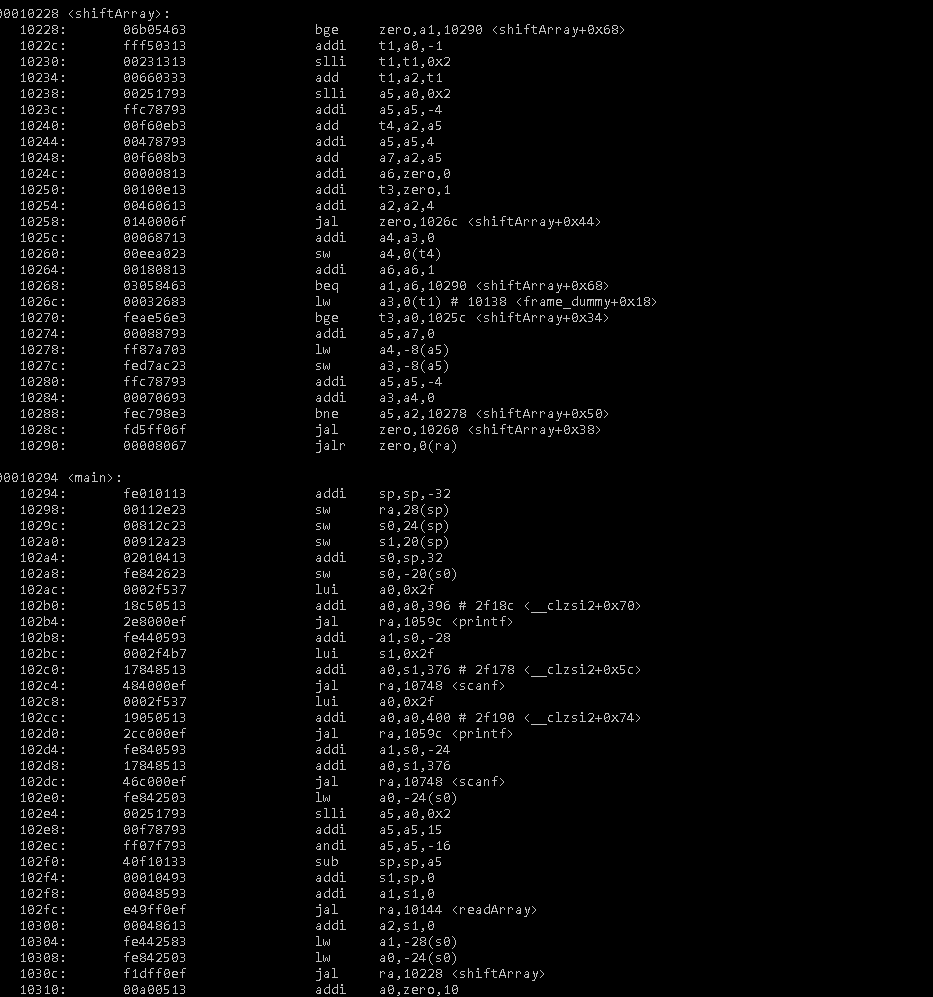
Проанализируем содержимое секции .textисполняемого файла:

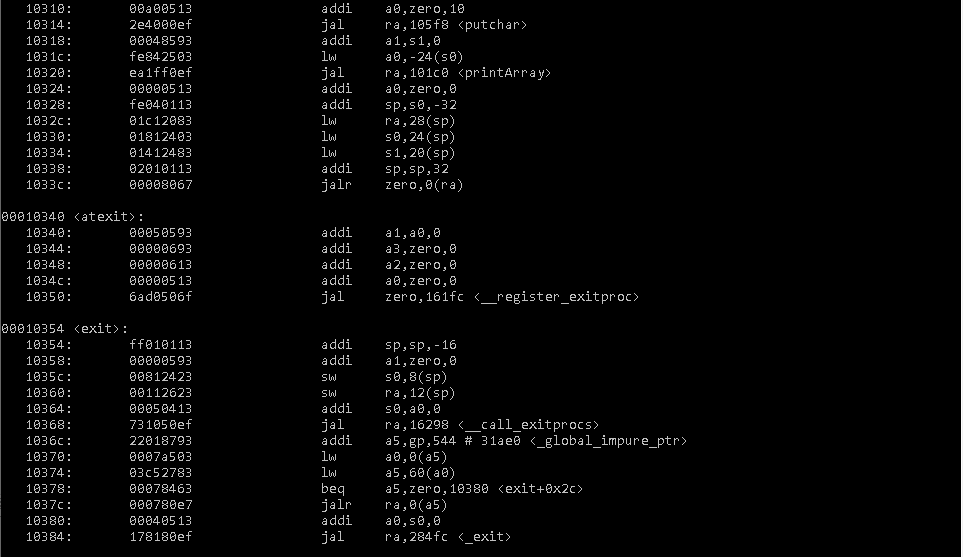


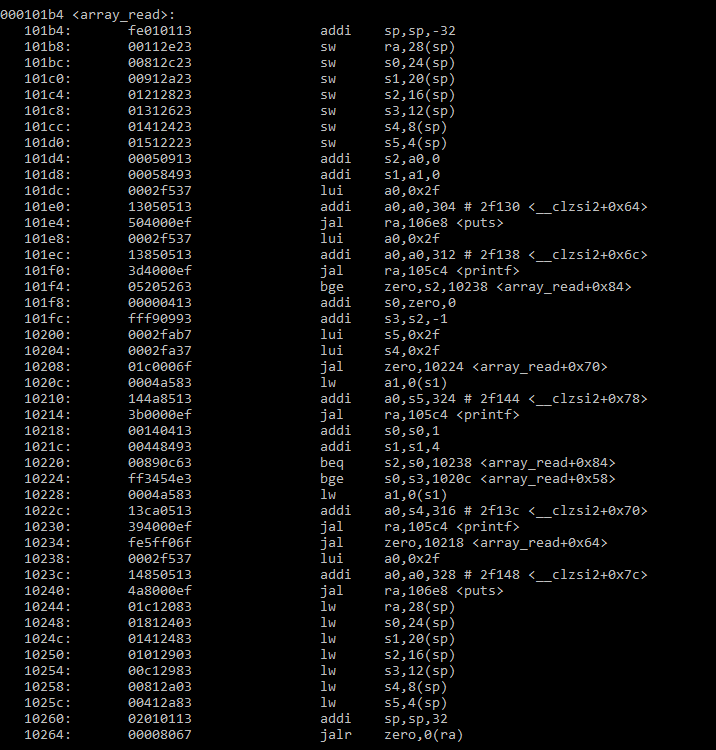
Секция кода теперь содержит намного большее количество строк, поэтому рассмотрим только самые важные участки:

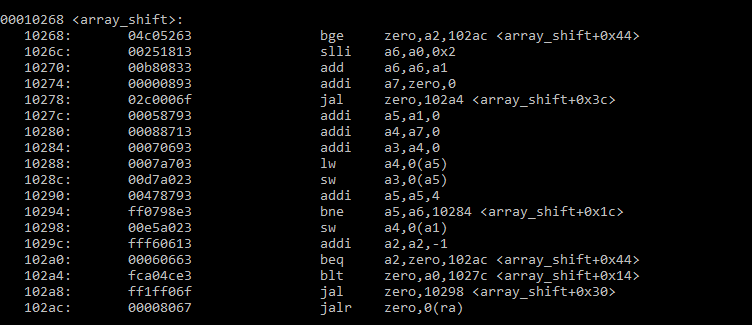


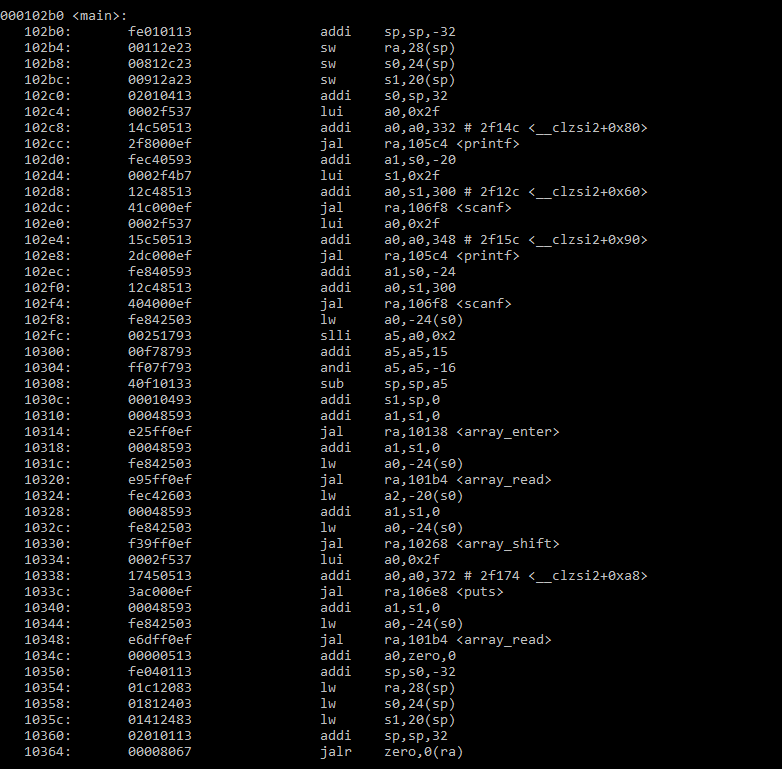












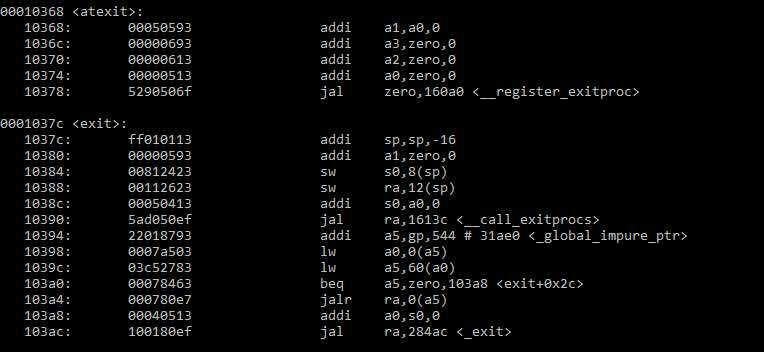
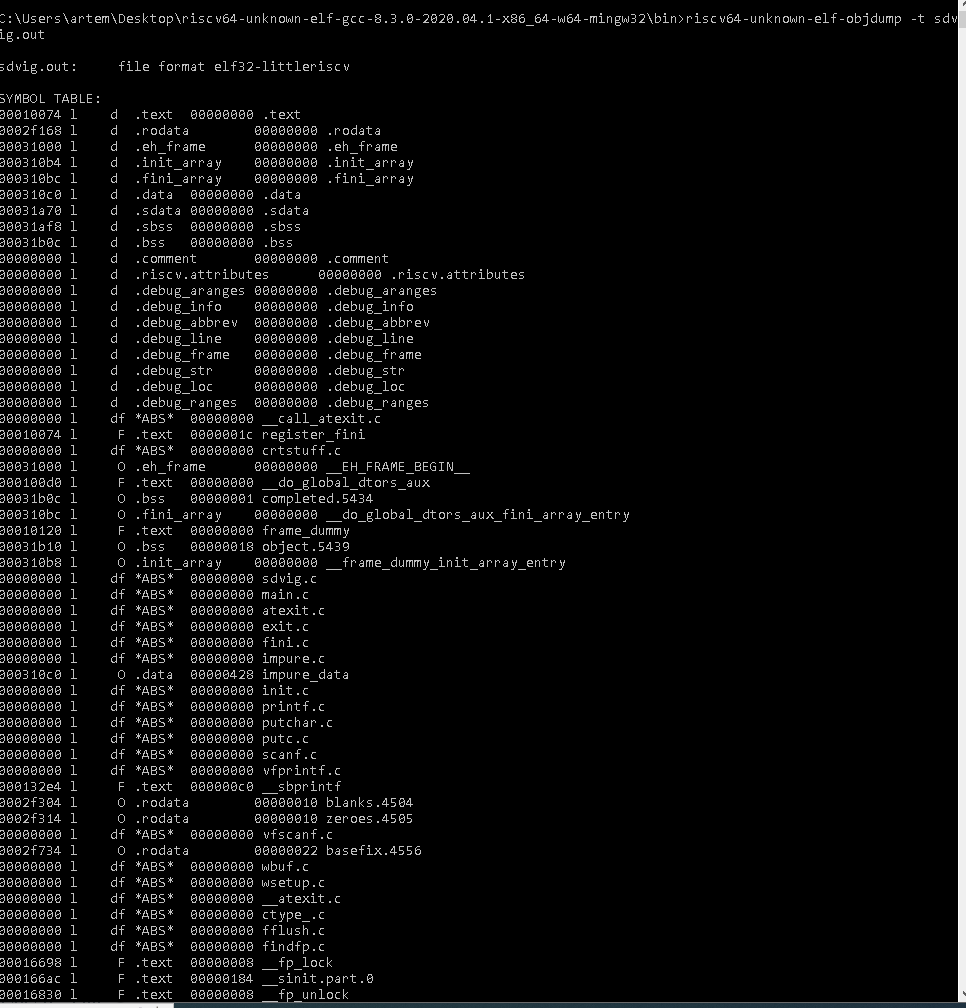


Таблица символов исполняемого файла



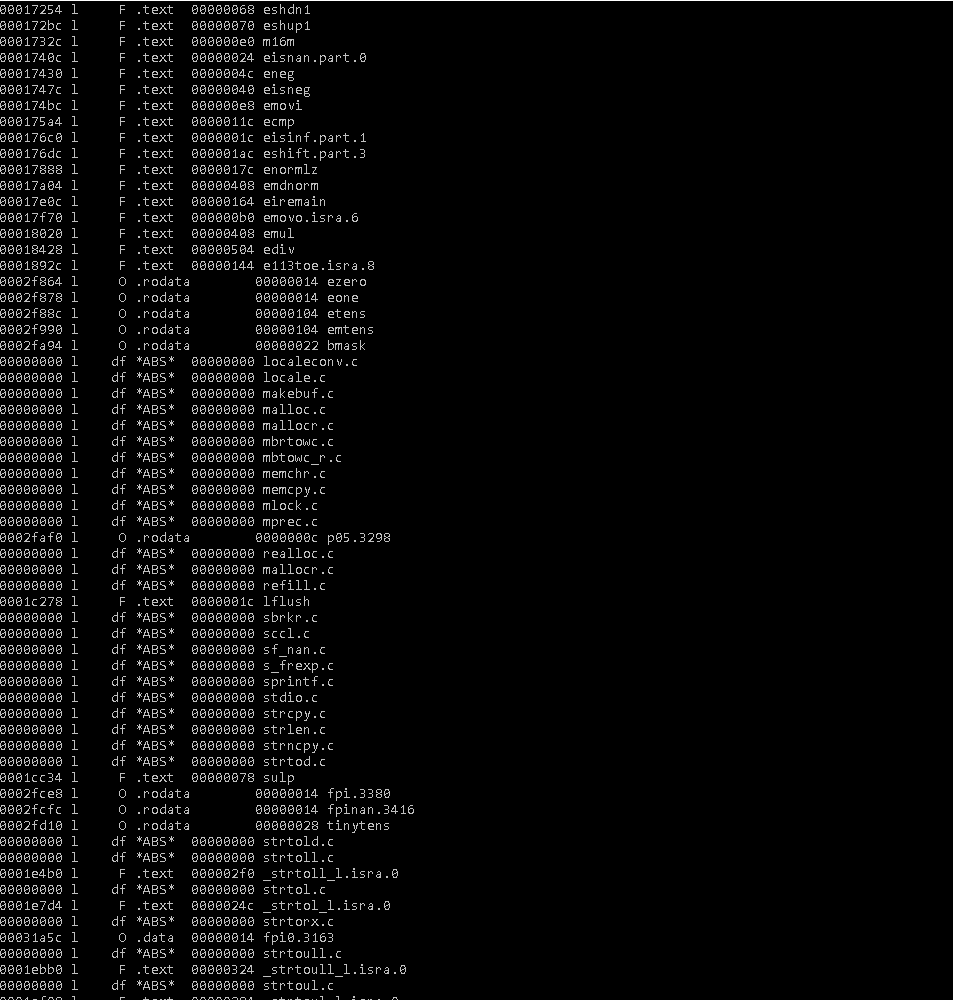


Таблица символов содержит множество дополнительных вхождений, однако в целом определяет все нужные секции, метки и адреса.

Проанализируем таблицу перемещенийисполняемого файла

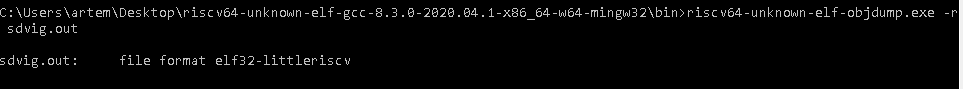
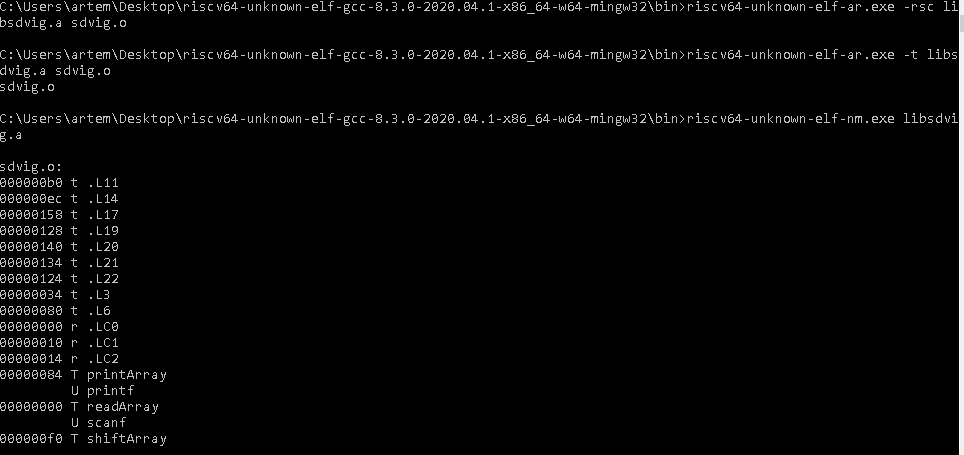


Таблица перемещений оказывается пуста, все необходимые релокации, оптимизации и замены инструкций были успешно проведены компоновщиком.

Итогом сборки программ на языке C по шагам является исполняемый на процессорах архитектуры RISC-V файл, решающий задачу переноса массива на заданное количество разрядов влево.

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

Статическая библиотека (staticlibrary) 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 циклического сдвига массива влево содержится в единственном исходном файле на языке C. Выделим этот файл в статическую библиотеку:



В выводе утилиты “nm” кодом “T” обозначаются символы, определенные в соответствующем объектном файле, кодом “U” - внешние символы.

# Цели

.PHONY: all clean

# Исходные файлы для сборки библиотеки

OBJS= sdvig.c

#Вызываемые приложения

AR = riscv64-unknown-elf-ar.exe

CC = riscv64-unknown-elf-gcc.exe

# Файл библиотеки

MYLIBNAME = shi.a

# Параметры компиляции

CFLAGS= -O1

# файлы искать в данном каталоге

INCLUDES+= -I .

# Make ищет файлы ... .h и ... .c в текущей директории

vpath %.h .

vpath %.c .

# $< = %.c

# $@ = %.o

%.o: %.c

$(CC) -MD $(CFLAGS) $(INCLUDES) -c $< -o $@

# Для того чтобы выполнить задачу "all", требуется построить библиотеку

all: $(MYLIBNAME)

# $^ = (sdvig.o)

$(MYLIBNAME): sdvig.o

$(AR) -rsc $@ $^

Листинг 3.3. Makefile для сборки исполняемого файла

# Цели

.PHONY: all clean

# Исходные файлы для сборки библиотеки

OBJS= main.c \

sdvi.a

#Вызываемые приложения

CC = riscv64-unknown-elf-gcc.exe

# Компиляция

CFLAGS= -O1 --save-temps

#файлы искать в данном каталоге

INCLUDES+= -I .

# Make ищет файлы … .c и … .a в текущей директории

vpath %.c .

vpath %.a .

# Для того чтобы выполнить задачу "all", требуется построить библиотеку

all: a.out

# Сборка файла

a.out: $(OBJS)

$(CC) $(CFLAGS) $(INCLUDES) $^

del \*.o \*.i \*.s \*.d

Листинг 3.4. Запуск Makefile

Запускаем Makefile.win, потом Makefile1.win со сборкой исполняемого файла.

mingw32-make.exe -f Makefile.win Makefile1.win

Листинг 3.5. Таблица символов исполняемого файла, созданного с помощью Makefile

Созданный файл идентичен файлу созданному ранее

Вывод

В ходе выполнения лабораторной работы были закреплены знания языка C, ассемблера RISC-V, получены навыки работы спрепроцессором, компилятором, ассемблером и компоновщиком пакета GCCи драйвером компилятора riscv64-unknown-elf-gcc. Были изучены особенности каждого этапа пошаговой сборки набора программ, а также инструменты, позволяющие выделить разработанные программы в статическую библиотеку и автоматизировать сборку этой библиотеки.