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.

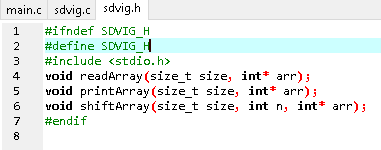
3) Выделить разработанную функцию в статическую библиотеку. Разработать make-файлы для сборки библиотеки и использующей ее тестовой программы. Проанализировать ход сборки библиотеки и программы, созданные файлы зависимостей.

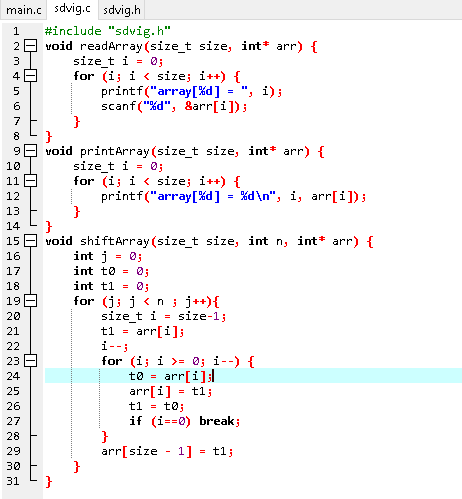
1. **Вариант задания**

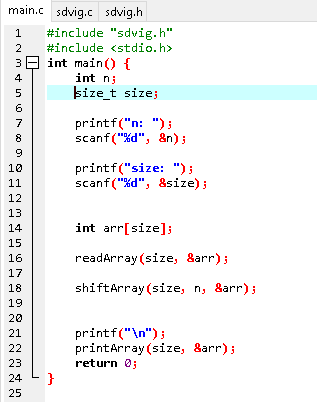
#10 Циклический сдвиг массива чисел на заданное количество разрядов влево

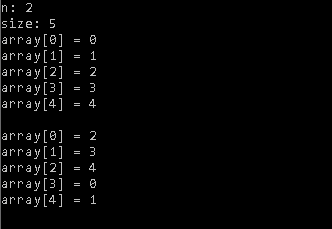
1. **Ход решения**

Листинг 1.1. Заголовочный файл sdvig.h

Листинг 1.2. Основной файл sdvig.c

Листинг 1.3. Тестовая программа main.c



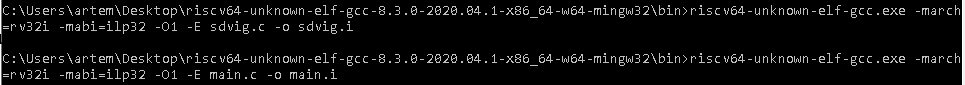


**2. Сборка программы по шагам**

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

# 1 "main.c"

# 1 "<built-in>"

# 1 "<command-line>"

# 1 "main.c"

# 1 "sdvig.h" 1

void readArray(size\_t size, int\* arr);

void printArray(size\_t size, int\* arr);

void shiftArray(size\_t size, int n, int\* arr);

# 2 "main.c" 2

int main() {

int n;

size\_t size;

printf("n: ");

scanf("%d", &n);

printf("size: ");

scanf("%d", &size);

const int getSize(int n){

return size \* sizeof(int);

}

int arr[size];

readArray(size, &arr);

shiftArray(size, n, &arr);

printf("\n");

printArray(size, &arr);

return 0;

}

Листинг 2.2. Файл sdvig.i

# 1 "sdvig.c"

# 1 "<built-in>"

# 1 "<command-line>"

# 1 "sdvig.c"

# 1 "sdvig.h" 1

void readArray(size\_t size, int\* arr);

void printArray(size\_t size, int\* arr);

void shiftArray(size\_t size, int n, int\* arr);

# 2 "sdvig.c" 2

void readArray(size\_t size, int\* arr) {

size\_t i = 0;

for (i; i < size; i++) {

printf("array[%d] = ", i);

scanf("%d", &arr[i]);

}

}

void printArray(size\_t size, int\* arr) {

size\_t i = 0;

for (i; i < size; ++i) {

printf("array[%d] = %d\n", i, arr[i]);

}

}

void shiftArray(size\_t size, int n, int\* arr) {

int j = 0;

int t0 = 0;

int t1 = 0;

for (j; j < n ; j++){

size\_t i = size - 1;

t1 = arr[i];

i--;

for (i; i >= 0; i--) {

t0 = arr[i];

arr[i] = t1;

t1 = t0;

}

arr[size - 1] = t1;

}

}

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

.file "main.c"

.option nopic

.attribute arch, "rv32i2p0"

.attribute unaligned\_access, 0

.attribute stack\_align, 16

.text

.section .rodata.str1.4,"aMS",@progbits,1

.align 2

.LC0:

.string "n: "

.align 2

.LC1:

.string "%d"

.align 2

.LC2:

.string "size: "

.text

.align 2

.globl main

.type main, @function

main:

addi sp,sp,-32

sw ra,28(sp)

sw s0,24(sp)

sw s1,20(sp)

addi s0,sp,32

sw s0,-20(s0)

lui a0,%hi(.LC0)

addi a0,a0,%lo(.LC0)

call printf

addi a1,s0,-28

lui s1,%hi(.LC1)

addi a0,s1,%lo(.LC1)

call scanf

lui a0,%hi(.LC2)

addi a0,a0,%lo(.LC2)

call printf

addi a1,s0,-24

addi a0,s1,%lo(.LC1)

call scanf

lw a0,-24(s0)

slli a5,a0,2

addi a5,a5,15

andi a5,a5,-16

sub sp,sp,a5

mv s1,sp

mv a1,s1

call readArray

mv a2,s1

lw a1,-28(s0)

lw a0,-24(s0)

call shiftArray

li a0,10

call putchar

mv a1,s1

lw a0,-24(s0)

call printArray

li a0,0

addi sp,s0,-32

lw ra,28(sp)

lw s0,24(sp)

lw s1,20(sp)

addi sp,sp,32

jr ra

.size main, .-main

.ident "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"

Листинг 2.4. Файл sdvig.s

.file "sdvig.c"

.option nopic

.attribute arch, "rv32i2p0"

.attribute unaligned\_access, 0

.attribute stack\_align, 16

.text

.section .rodata.str1.4,"aMS",@progbits,1

.align 2

.LC0:

.string "array[%d] = "

.align 2

.LC1:

.string "%d"

.text

.align 2

.globl readArray

.type readArray, @function

readArray:

ble a0,zero,.L6

addi sp,sp,-32

sw ra,28(sp)

sw s0,24(sp)

sw s1,20(sp)

sw s2,16(sp)

sw s3,12(sp)

sw s4,8(sp)

mv s2,a0

mv s1,a1

li s0,0

lui s4,%hi(.LC0)

lui s3,%hi(.LC1)

.L3:

mv a1,s0

addi a0,s4,%lo(.LC0)

call printf

mv a1,s1

addi a0,s3,%lo(.LC1)

call scanf

addi s0,s0,1

addi s1,s1,4

bne s2,s0,.L3

lw ra,28(sp)

lw s0,24(sp)

lw s1,20(sp)

lw s2,16(sp)

lw s3,12(sp)

lw s4,8(sp)

addi sp,sp,32

jr ra

.L6:

ret

.size readArray, .-readArray

.section .rodata.str1.4

.align 2

.LC2:

.string "array[%d] = %d\n"

.text

.align 2

.globl printArray

.type printArray, @function

printArray:

ble a0,zero,.L14

addi sp,sp,-32

sw ra,28(sp)

sw s0,24(sp)

sw s1,20(sp)

sw s2,16(sp)

sw s3,12(sp)

mv s2,a0

mv s1,a1

li s0,0

lui s3,%hi(.LC2)

.L11:

lw a2,0(s1)

mv a1,s0

addi a0,s3,%lo(.LC2)

call printf

addi s0,s0,1

addi s1,s1,4

bne s2,s0,.L11

lw ra,28(sp)

lw s0,24(sp)

lw s1,20(sp)

lw s2,16(sp)

lw s3,12(sp)

addi sp,sp,32

jr ra

.L14:

ret

.size printArray, .-printArray

.align 2

.globl shiftArray

.type shiftArray, @function

shiftArray:

ble a1,zero,.L17

addi t1,a0,-1

slli t1,t1,2

add t1,a2,t1

slli a5,a0,2

addi a5,a5,-4

add t4,a2,a5

addi a5,a5,4

add a7,a2,a5

li a6,0

li t3,1

addi a2,a2,4

.L21:

lw a4,0(t1)

ble a0,t3,.L19

mv a5,a7

.L20:

mv a3,a4

lw a4,-8(a5)

sw a3,-8(a5)

addi a5,a5,-4

bne a5,a2,.L20

.L19:

sw a4,0(t4)

addi a6,a6,1

bne a1,a6,.L21

.L17:

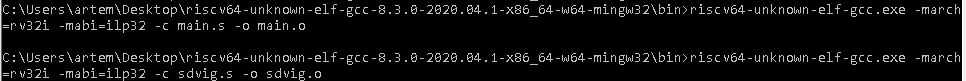
ret

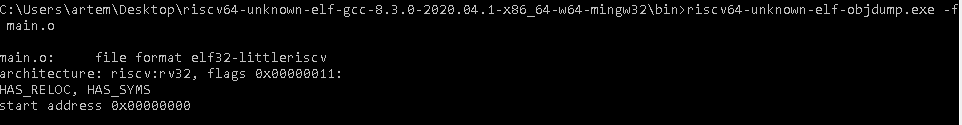
.size shiftArray, .-shiftArray

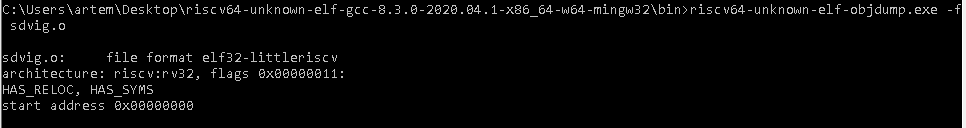
.ident "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"

Ассемблирование

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

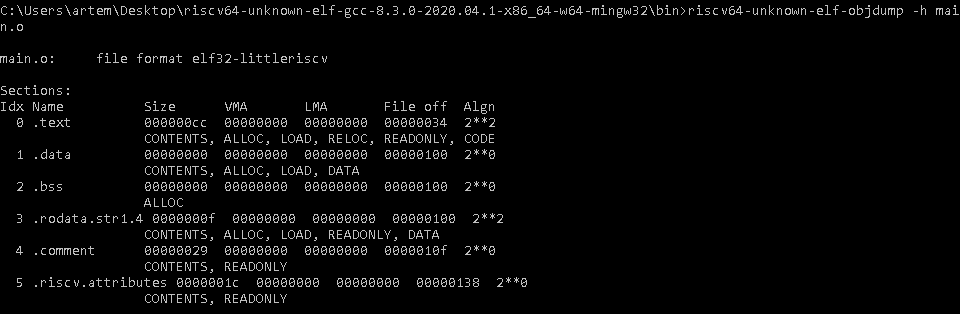
Драйвер компилятора riscv64-unknown-elf-gcc запускается с параметрами командной строки “-march=rv32i -mabi=ilp32”, указывающих что целевым является процессор с базовой архитектурой системы команд RV32I; -c – указание остановить процесс сборки после ассемблирования.

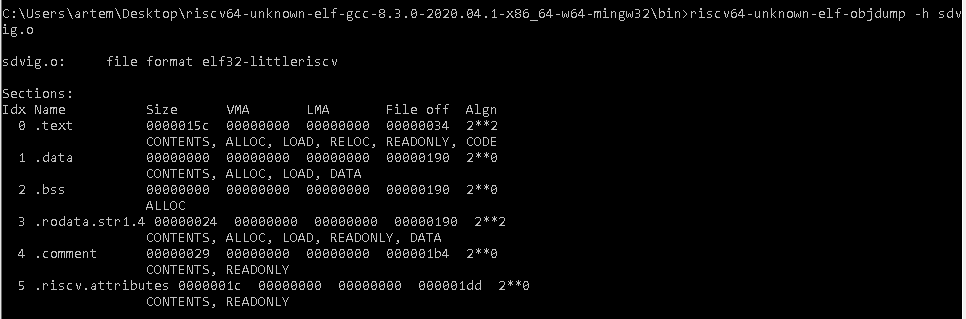
Объектный файл не является текстовым, для изучения его содержимого используемутилиту objdump:



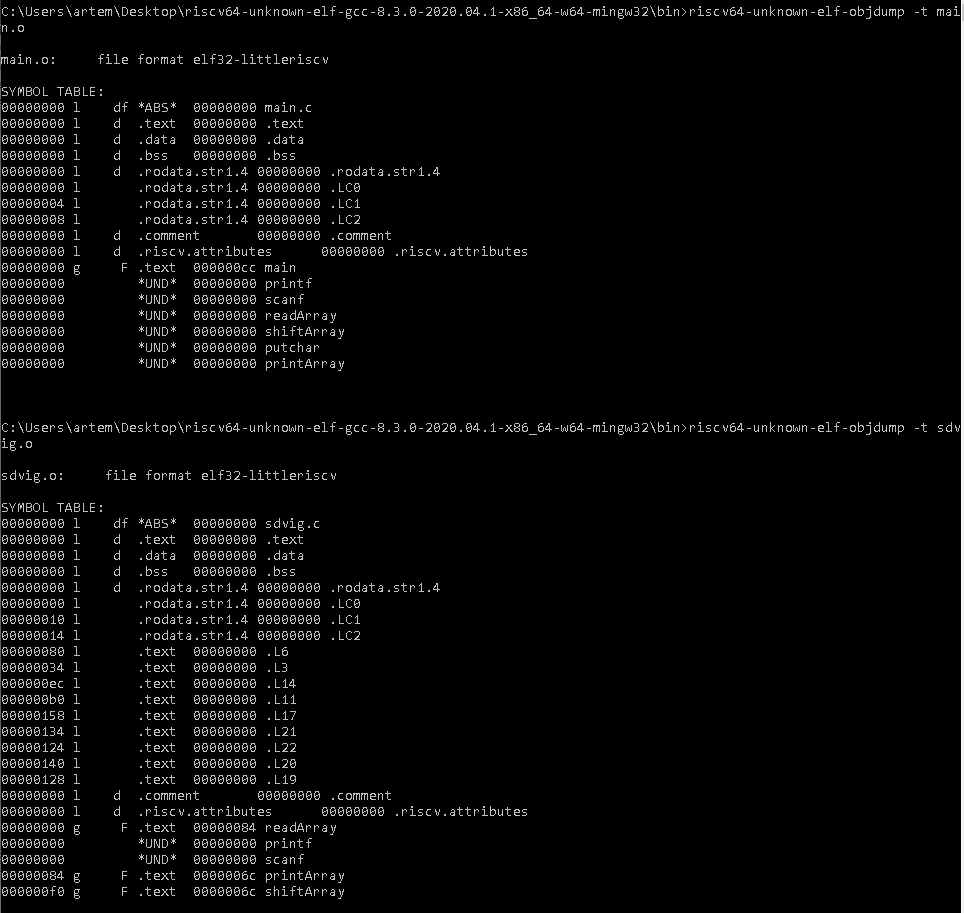
Оба файла содержат таблицу перемещений (в списке флагов есть флага HAS\_RELOC)

Выведем все заголовки секций объектных файлов





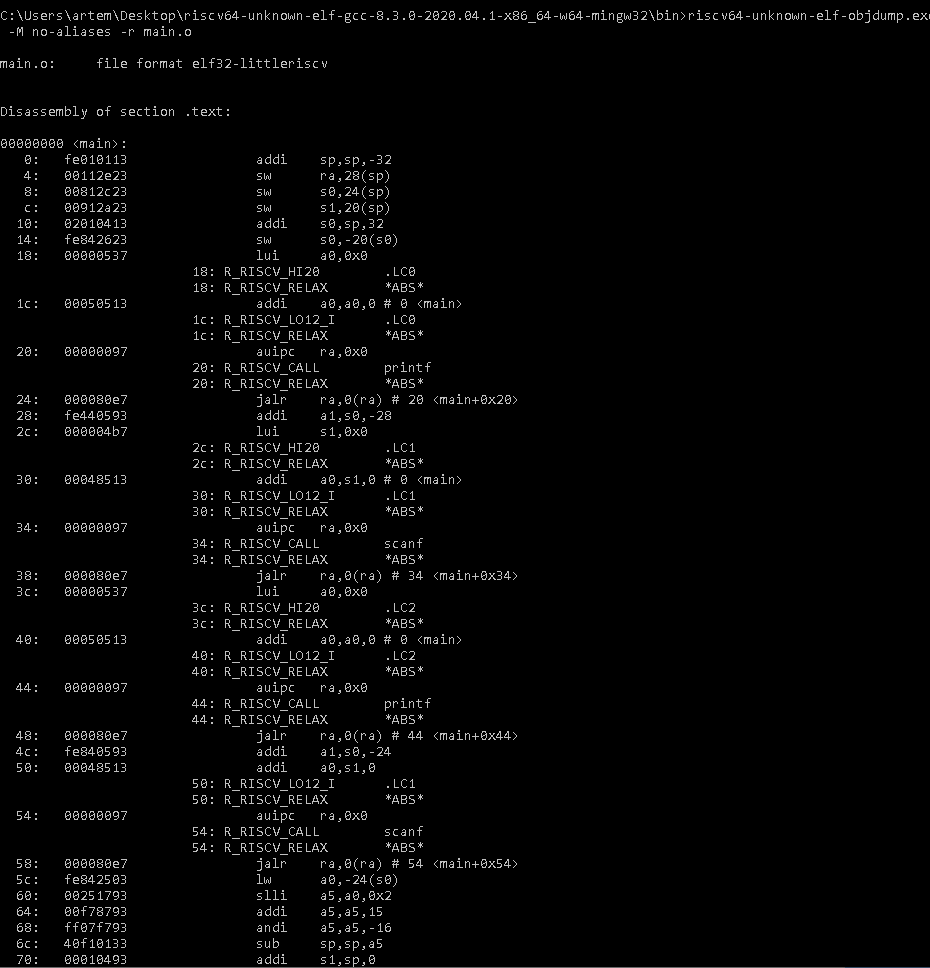
Выведем таблицы символов файлов sdvig.o и main.o

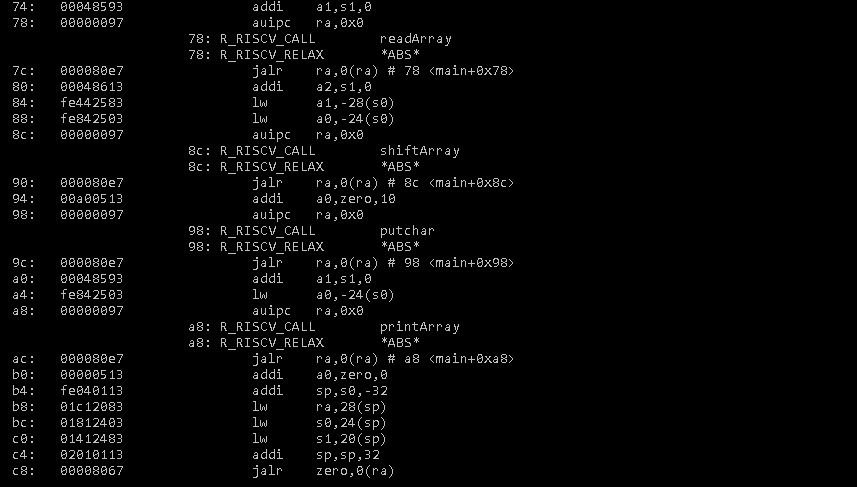


В каждой таблице только одни глобальный (флаг “g”) символ типа «функция» (“F”) – созданные фцнкции

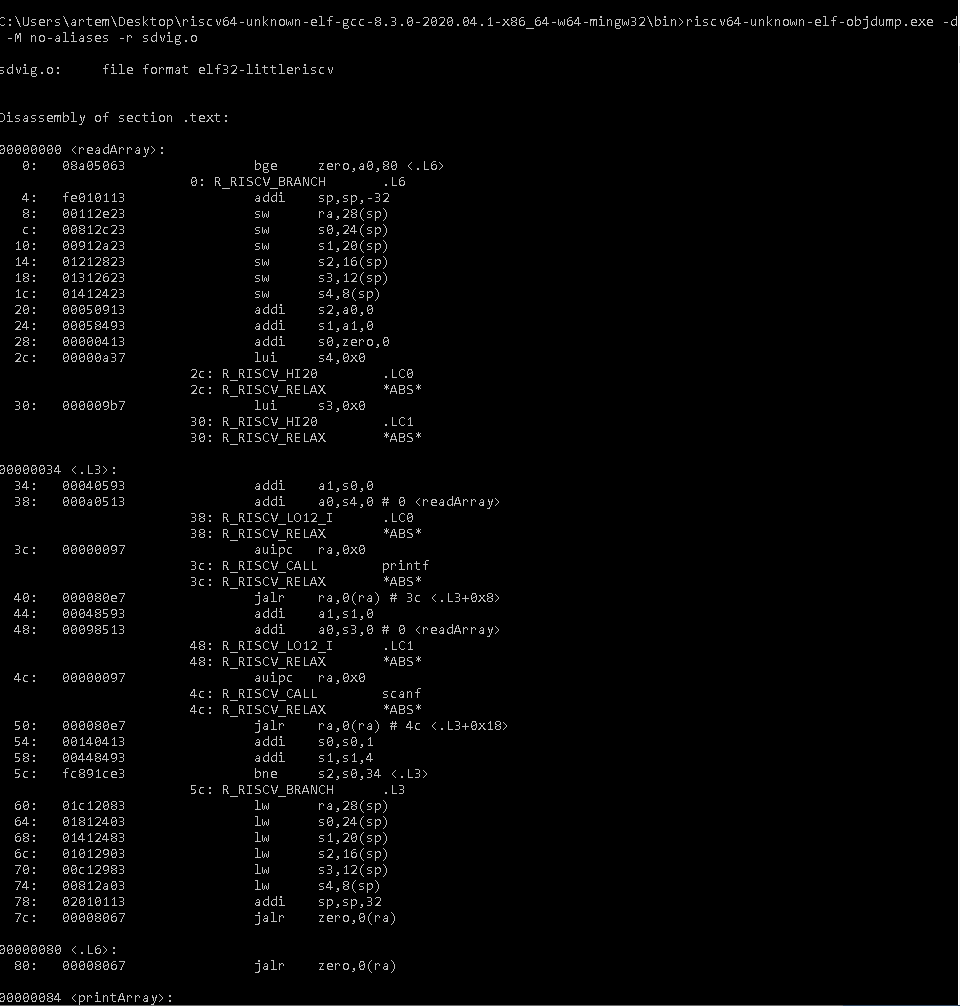
Листинг 2.9. файл main.o

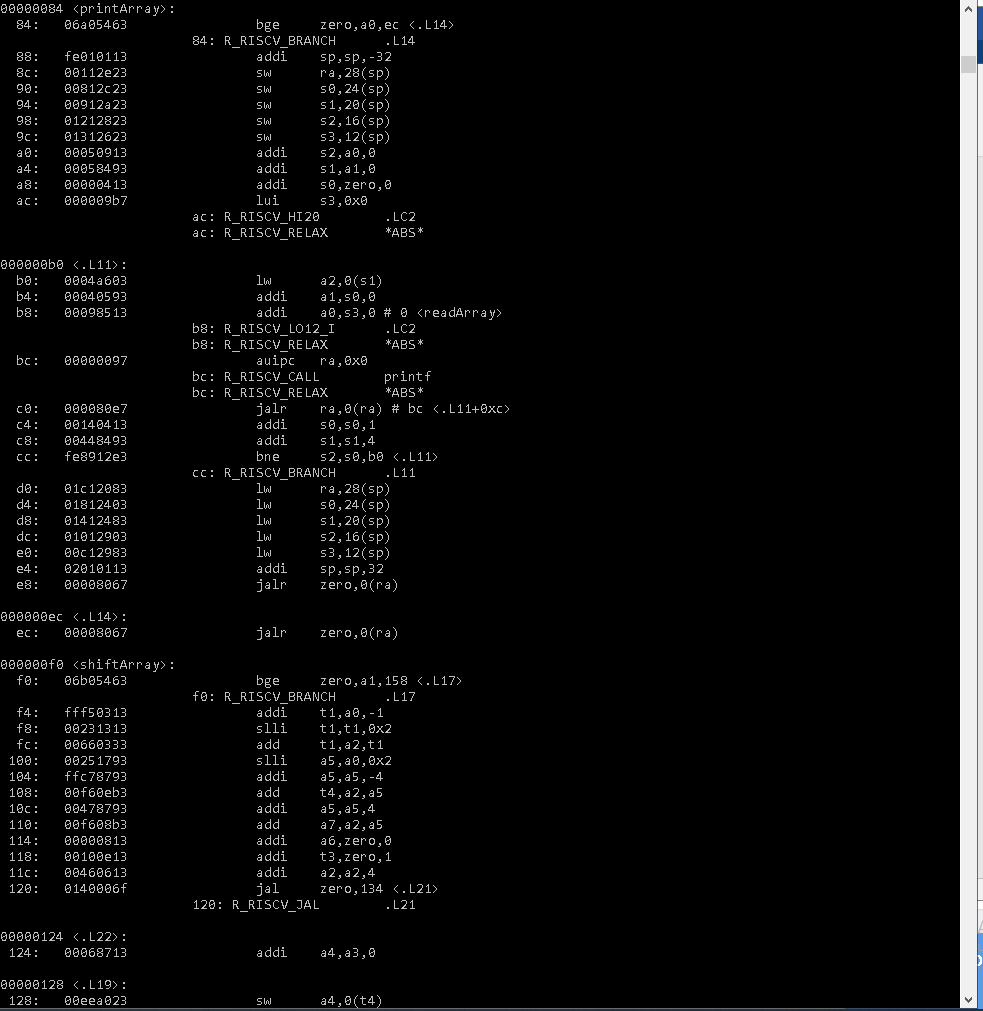
Выведем секции .text объектных файлов – секций кода, в которых содержатся коды инструкций:

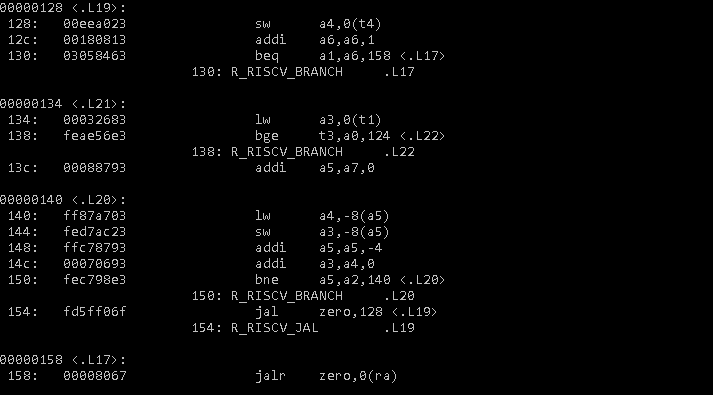




Листинг 2.10. файл sdvig.o

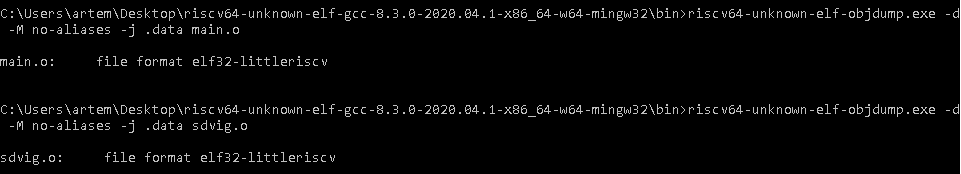




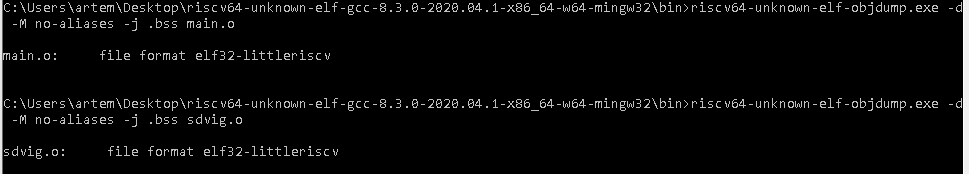


Дизассемблированный код практически идентичен сгенерированному (за исключением псевдоинструкций).

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

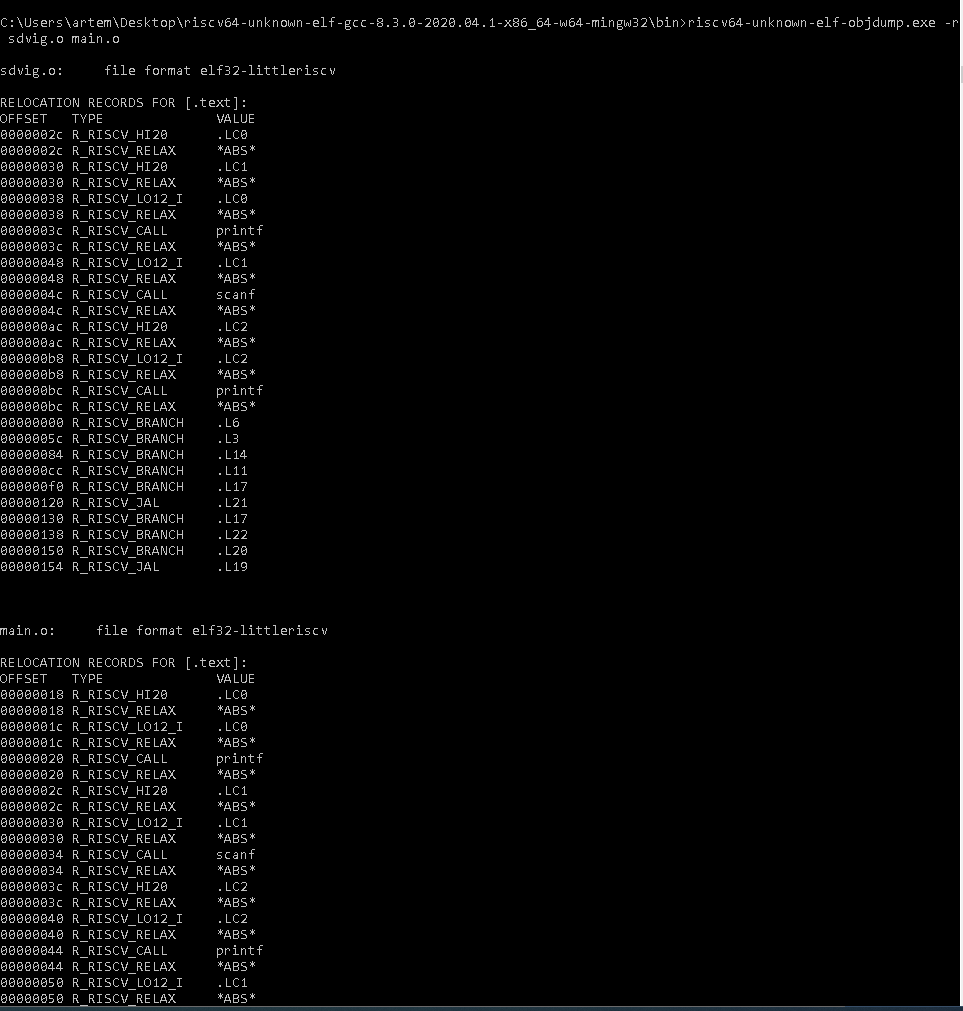


Секции .bss объектных файлов – секции данных, инициализированных нулями – таким же образом пусты:



Листинг 2.11. таблицы перемещений sdvig.o

Выведем таблицы перемещений объектных файлов:



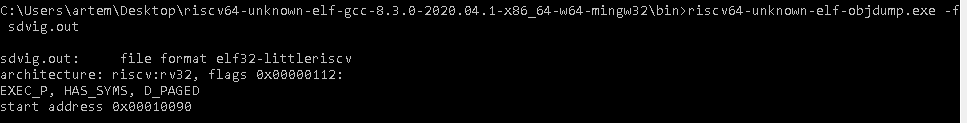


Компановка

Следующим шагом является компоновка и формирование исполняемых фалов программ:

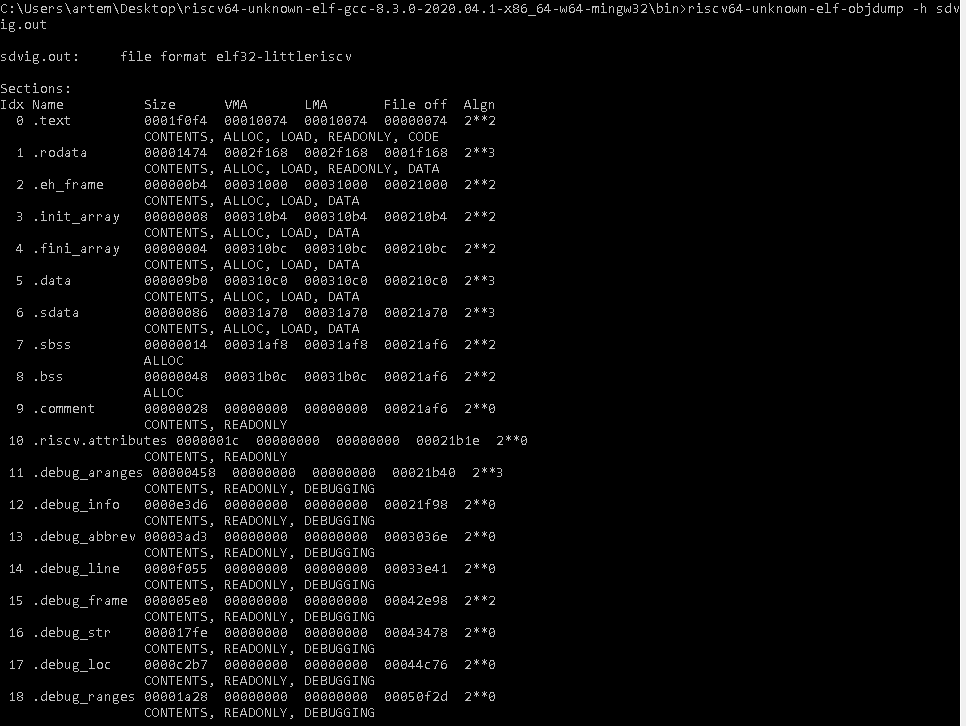


Сформированный компоновщиком файл “sdvig.out”, также является «бинарным»:



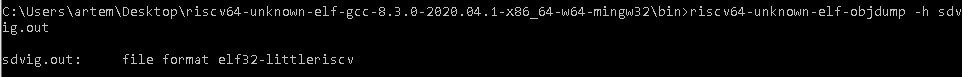
Флаг EXEC\_Pуказывает, что файл действительно является исполняемым, после загрузки его выполнение должно начаться с адреса 0x0001008c (entrypoint).

Перечислим секции исполняемого файла:

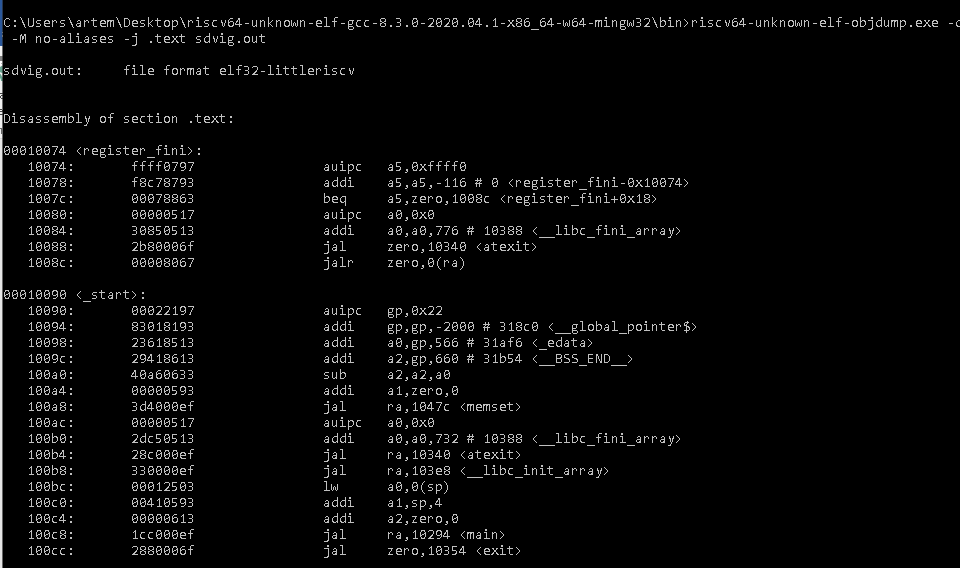


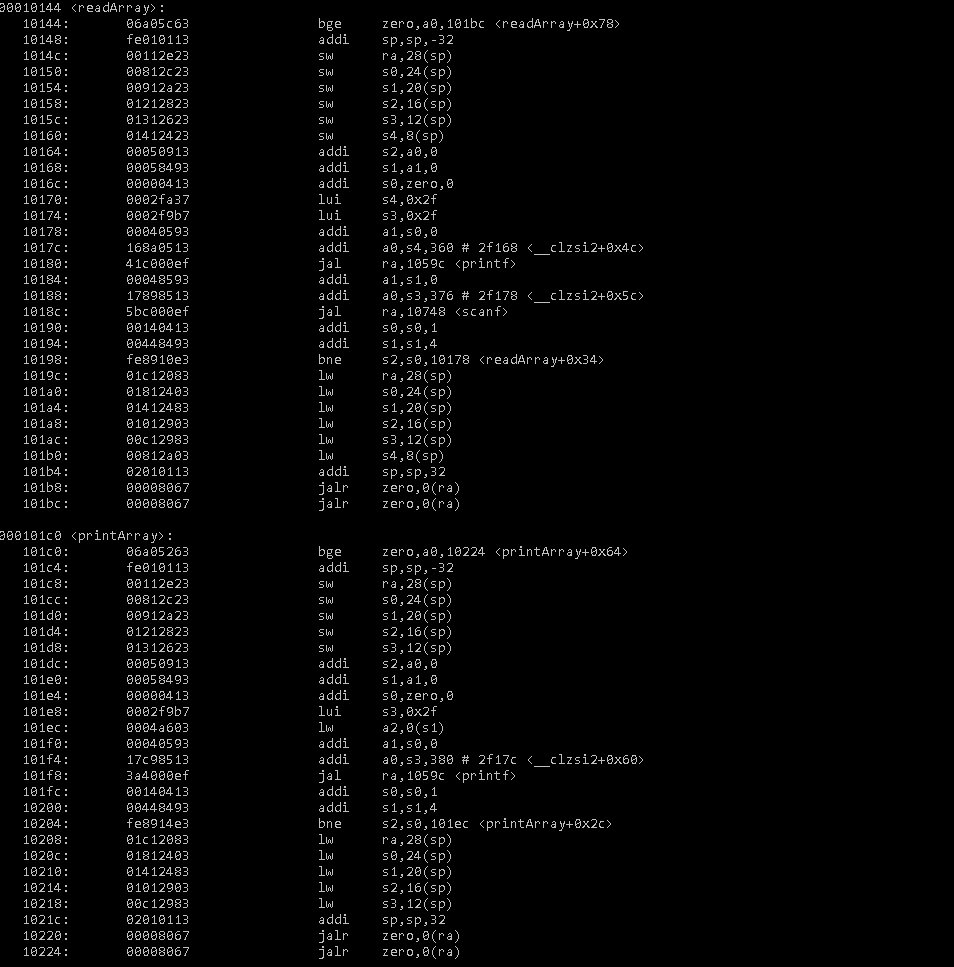
В исполняемом файле действительно производится слияние содержания секций обоих объектных файлов, а также значительное расширение списка секций новыми блоками.

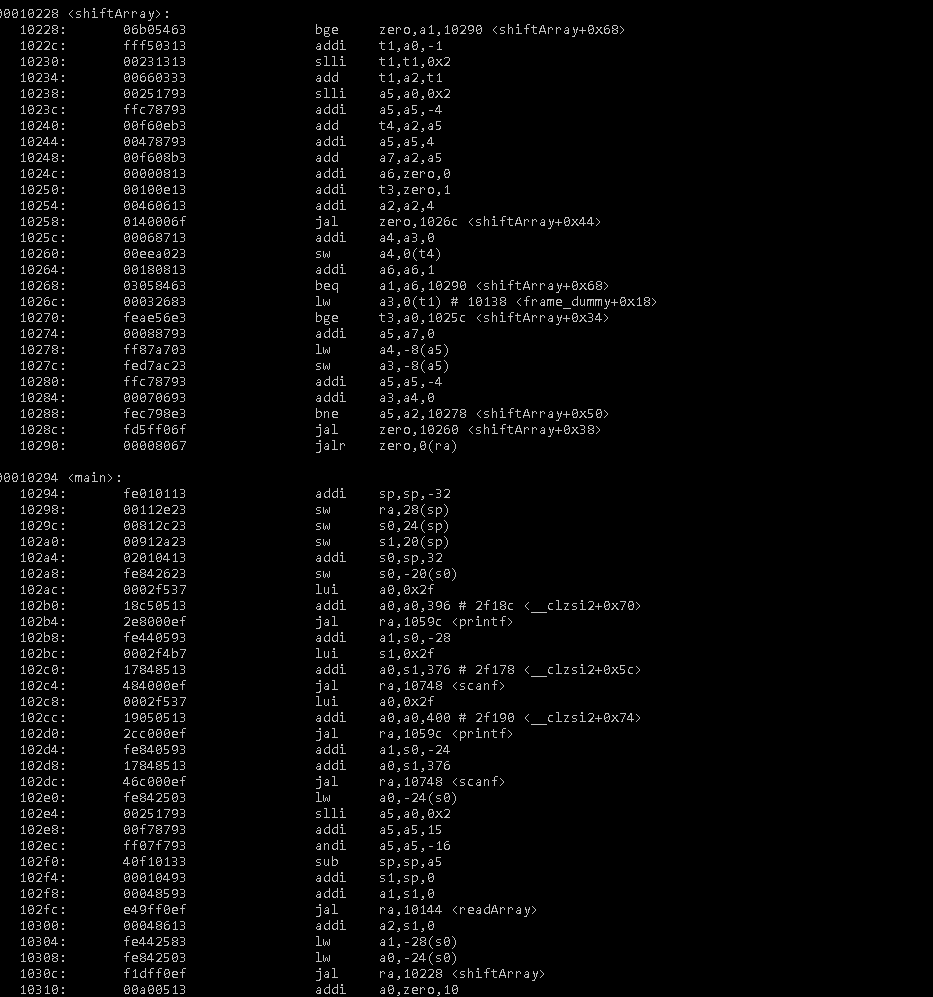
Проанализируем содержимое секции .textисполняемого файла:

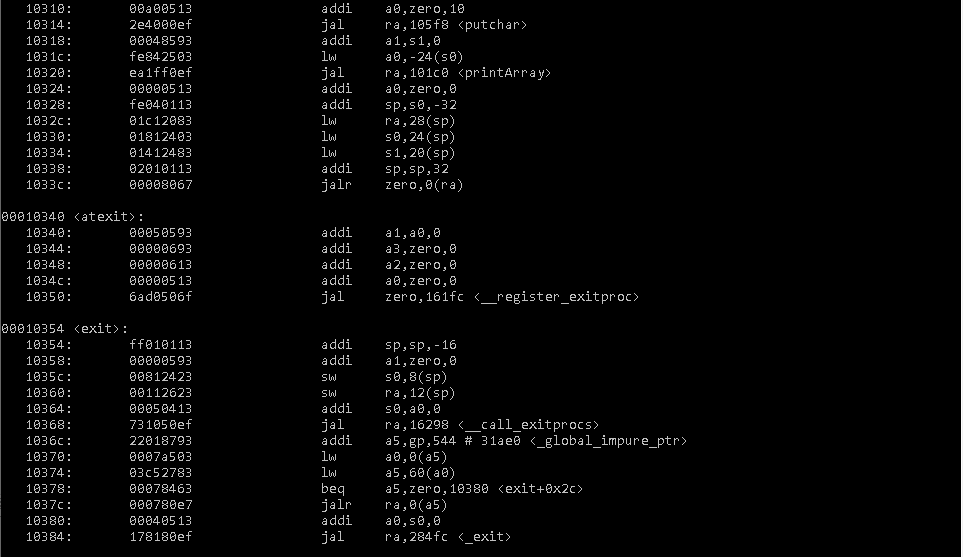


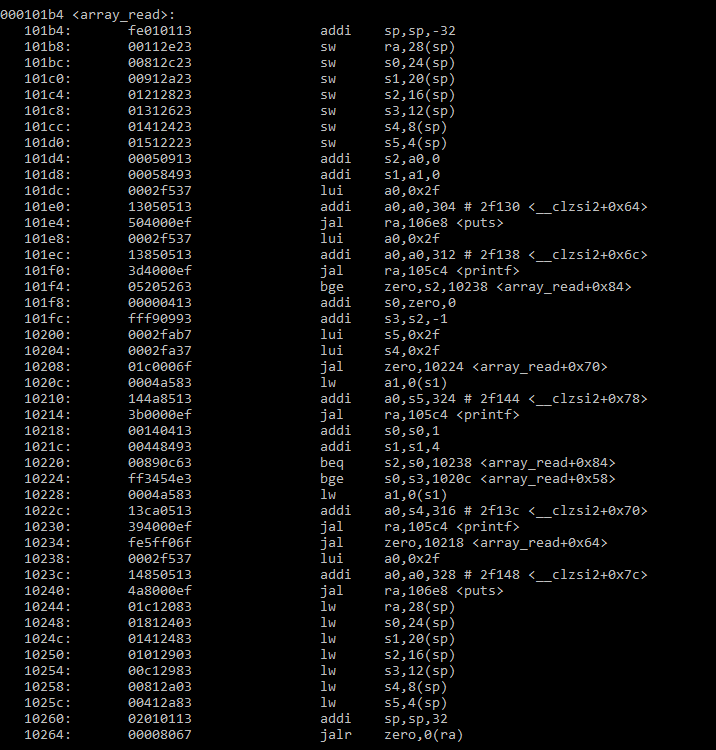
Секция кода теперь содержит намного большее количество строк, поэтому рассмотрим только самые важные участки:

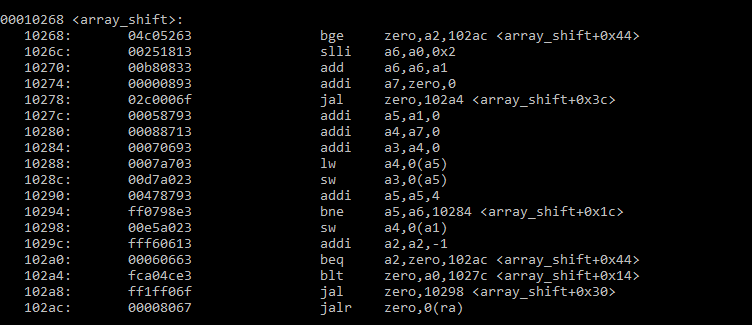


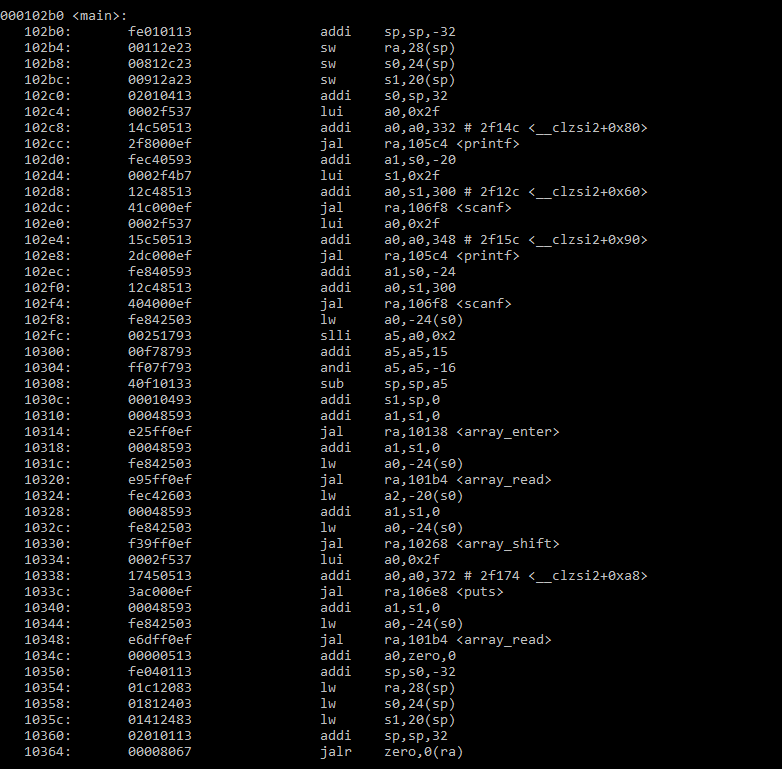












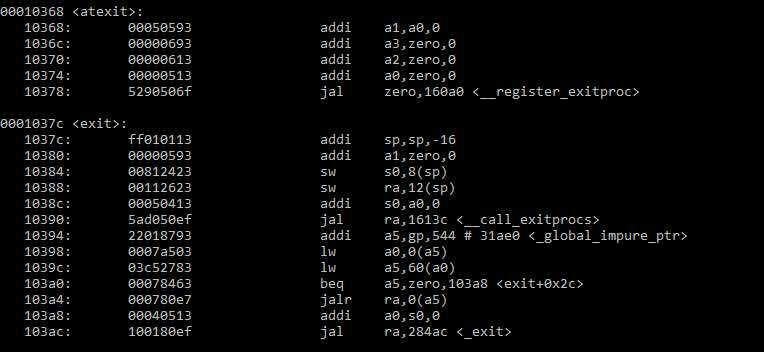
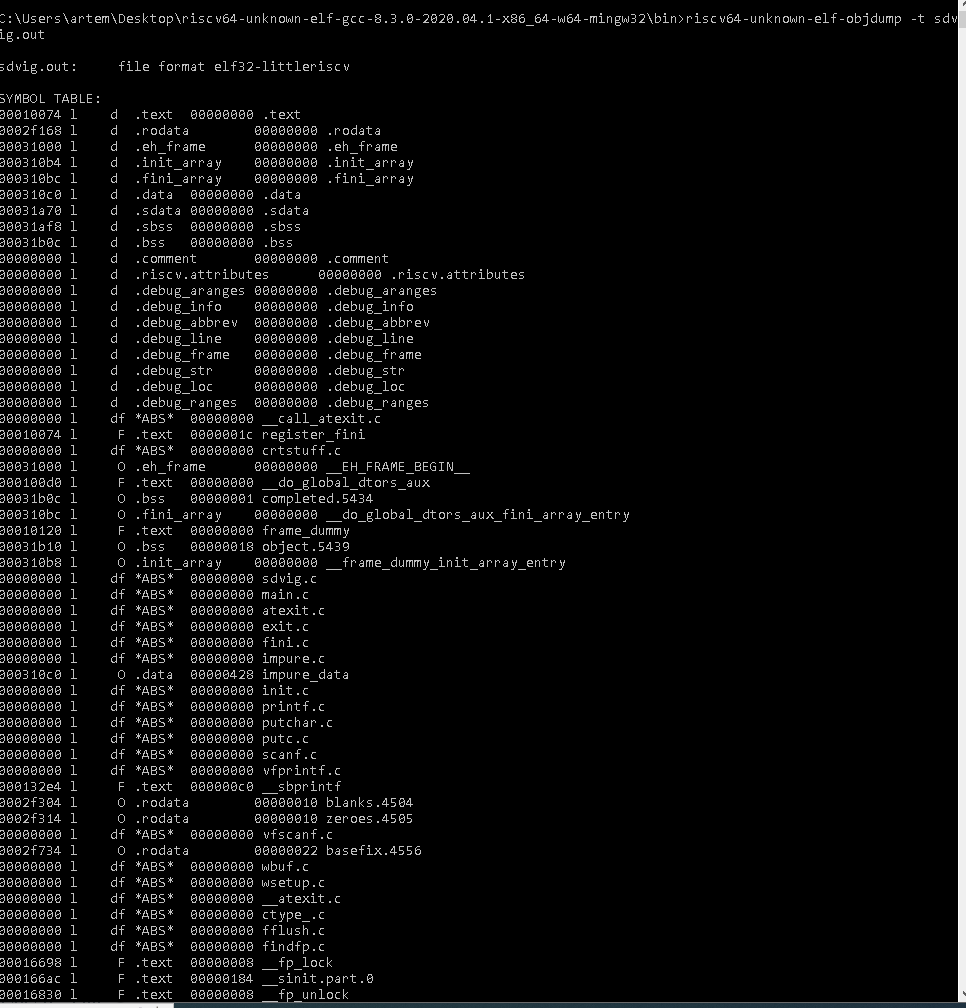


Таблица символов исполняемого файла



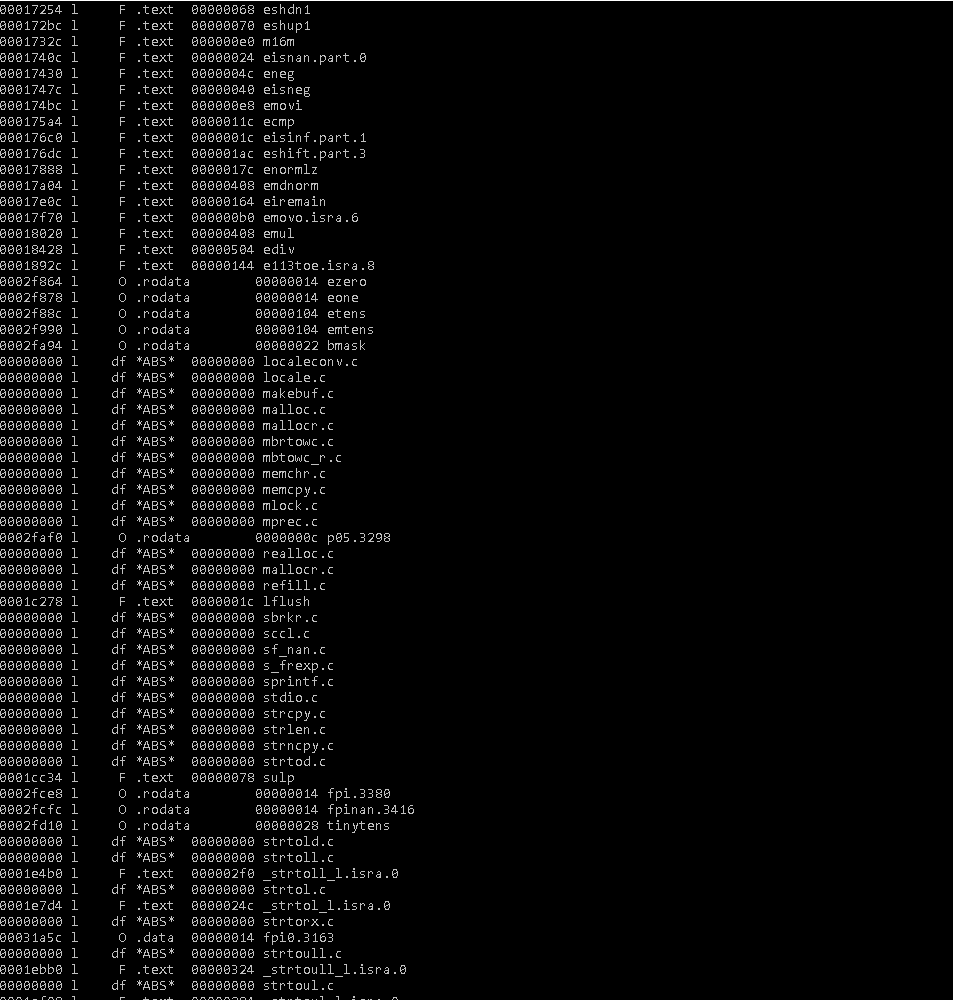


Таблица символов содержит множество дополнительных вхождений, однако в целом определяет все нужные секции, метки и адреса.

Проанализируем таблицу перемещенийисполняемого файла

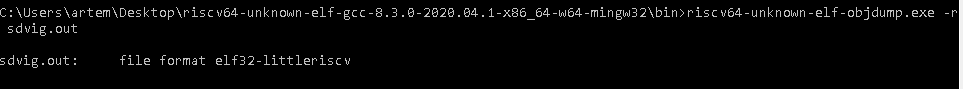
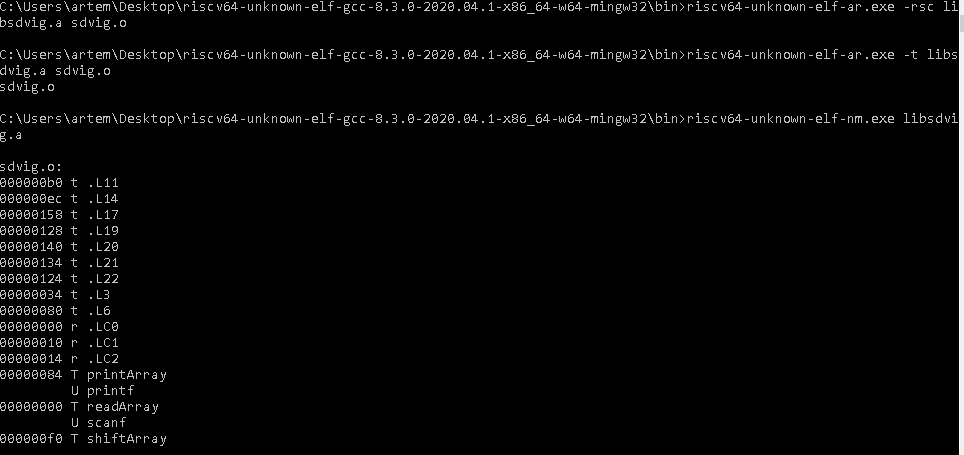


Таблица перемещений оказывается пуста, все необходимые релокации, оптимизации и замены инструкций были успешно проведены компоновщиком.

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 вправно 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.