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 Довлатов И.М.

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**

#17. Циклический сдвиг массива чисел на заданное количество разрядов вправо

1. **Ход решения**

Листинг 1.1. Заголовочный файл shift.h

#ifndef SHIFT\_H

#define SHIFT\_H

void array\_enter(int size, int\* array);

void array\_read(int size, int\* array);

void array\_shift(int size, int\* array, int delta);

#endif

Листинг 1.2. Основной файл shift.c

#include "shift.h"

void array\_enter(int size, int\* array) {

int i = 0;

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

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

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

}

}

void array\_read(int size, int\* array) {

int j = 0;

printf("Array\n");

printf("[ ");

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

if(j<(size - 1)) {

printf("%d,\t",array[j]);

}else{

printf("%d ",array[j]);

}

}

printf("]\n");

}

void array\_shift(int size, int\* array, int delta){

int current\_element;

while(delta > 0){

int i = 0;

int prev\_element = 0;

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

current\_element = array[i];

array[i] = prev\_element;

prev\_element = current\_element;

}

array[0] = current\_element;

delta--;

}

}

Листинг 1.3. Тестовая программа main.c

#include <stdio.h>

#include <stdlib.h>

#include "shift.h"

int main() {

int delta;

int array\_size;

printf("Enter delta = :");

scanf("%d",&delta);

printf("Enter array size = :");

scanf("%d",&array\_size);

int array[array\_size];

array\_enter(array\_size, &array);

array\_read(array\_size, &array);

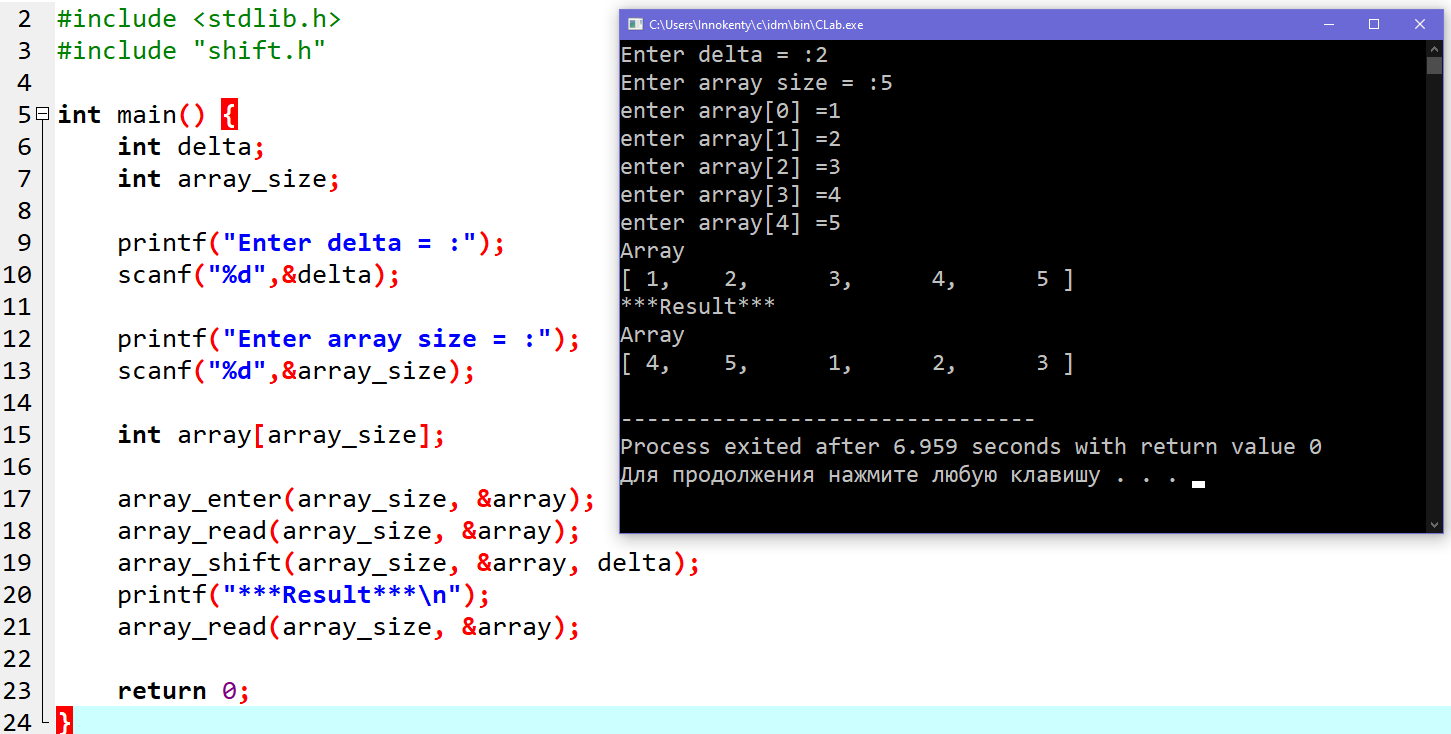
array\_shift(array\_size, &array, delta);

printf("\*\*\*Result\*\*\*\n");

array\_read(array\_size, &array);

return 0;

}



**2. Сборка программы по шагам**

Предпроцесирование

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

Препроцессирование, компиляция и ассемблирование

riscv64-unknown-elf-gcc.exe –march=rv32i –mabi=ilp32 -O1 -E shift.c –o shift.i

riscv64-unknown-elf-gcc.exe –march=rv32i –mabi=ilp32 -O1 -E main.c –o 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.

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

Листинг 2.1. Файл main.i

# 3 "main.c" 2

# 1 "shift.h" 1

# 3 "shift.h"

void array\_enter(int size, int\* array);

void array\_read(int size, int\* array);

void array\_shift(int size, int\* array, int delta);

# 4 "main.c" 2

int main() {

int delta;

int array\_size;

printf("Enter delta = :");

scanf("%d",&delta);

printf("Enter array size = :");

scanf("%d",&array\_size);

int array[array\_size];

array\_enter(array\_size, &array);

array\_read(array\_size, &array);

array\_shift(array\_size, &array, delta);

printf("\*\*\*Result\*\*\*\n");

array\_read(array\_size, &array);

return 0;

}

Листинг 2.2. Файл shift.i

# 1 "shift.c"

# 1 "<built-in>"

# 1 "<command-line>"

# 1 "shift.c"

# 1 "shift.h" 1

void array\_enter(int size, int\* array);

void array\_read(int size, int\* array);

void array\_shift(int size, int\* array, int delta);

# 2 "shift.c" 2

void array\_enter(int size, int\* array) {

int i = 0;

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

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

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

}

}

void array\_read(int size, int\* array) {

int j = 0;

printf("Array\n");

printf("[ ");

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

if(j<(size - 1)) {

printf("%d,\t",array[j]);

}else{

printf("%d ",array[j]);

}

}

printf("]\n");

}

void array\_shift(int size, int\* array, int delta){

int current\_element;

while(delta > 0){

int i = 0;

int prev\_element = 0;

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

current\_element = array[i];

array[i] = prev\_element;

prev\_element = current\_element;

}

array[0] = current\_element;

delta--;

}

}

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

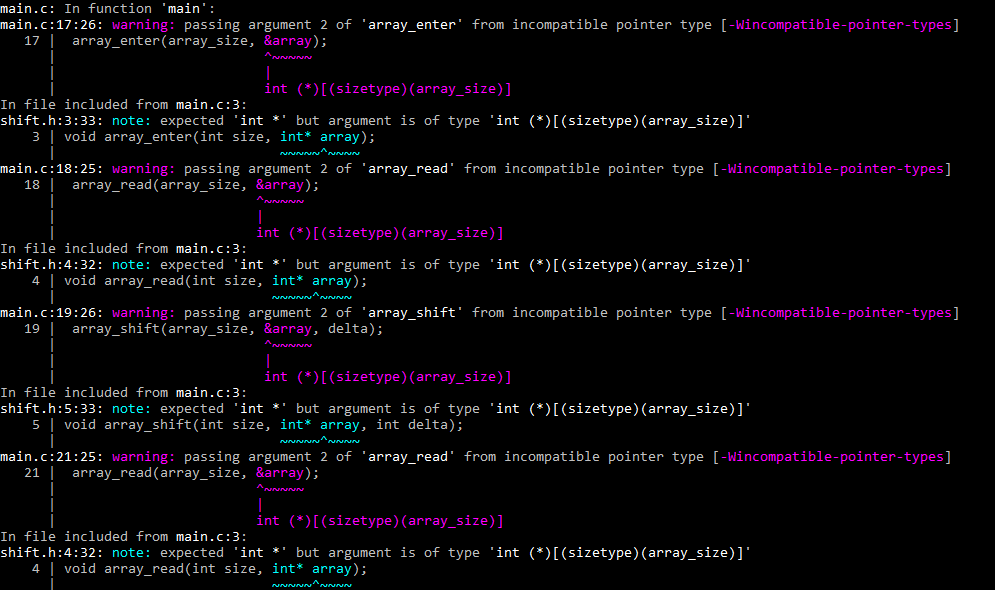
Компиляция

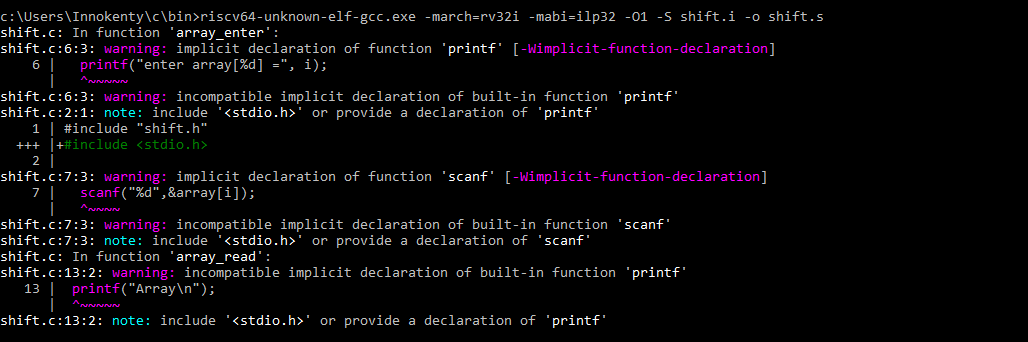
Следующим шагом является компиляция файлов “shift.i” и “main.i” в код на языке ассемблера “shift.s” и “main.s”:

riscv64-unknown-elf-gcc.exe –march=rv32i –mabi=ilp32 -O1 -S main.i –o main.s

riscv64-unknown-elf-gcc.exe –march=rv32i –mabi=ilp32 -O1 -S shift.i –o shift.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 "Enter delta = :"

.align 2

.LC1:

.string "%d"

.align 2

.LC2:

.string "Enter array size = :"

.align 2

.LC3:

.string "\*\*\*Result\*\*\*"

.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

lui a0,%hi(.LC0)

addi a0,a0,%lo(.LC0)

call printf

addi a1,s0,-20

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 array\_enter

mv a1,s1

lw a0,-24(s0)

call array\_read

lw a2,-20(s0)

mv a1,s1

lw a0,-24(s0)

call array\_shift

lui a0,%hi(.LC3)

addi a0,a0,%lo(.LC3)

call puts

mv a1,s1

lw a0,-24(s0)

call array\_read

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 shift.s

.file "shift.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 "enter array[%d] ="

.align 2

.LC1:

.string "%d"

.text

.align 2

.globl array\_enter

.type array\_enter, @function

array\_enter:

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 array\_enter, .-array\_enter

.section .rodata.str1.4

.align 2

.LC2:

.string "Array"

.align 2

.LC3:

.string "[ "

.align 2

.LC4:

.string "%d,\t"

.align 2

.LC5:

.string "%d "

.align 2

.LC6:

.string "]"

.text

.align 2

.globl array\_read

.type array\_read, @function

array\_read:

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)

sw s5,4(sp)

mv s2,a0

mv s1,a1

lui a0,%hi(.LC2)

addi a0,a0,%lo(.LC2)

call puts

lui a0,%hi(.LC3)

addi a0,a0,%lo(.LC3)

call printf

ble s2,zero,.L10

li s0,0

addi s3,s2,-1

lui s5,%hi(.LC5)

lui s4,%hi(.LC4)

j .L13

.L11:

lw a1,0(s1)

addi a0,s5,%lo(.LC5)

call printf

.L12:

addi s0,s0,1

addi s1,s1,4

beq s2,s0,.L10

.L13:

ble s3,s0,.L11

lw a1,0(s1)

addi a0,s4,%lo(.LC4)

call printf

j .L12

.L10:

lui a0,%hi(.LC6)

addi a0,a0,%lo(.LC6)

call puts

lw ra,28(sp)

lw s0,24(sp)

lw s1,20(sp)

lw s2,16(sp)

lw s3,12(sp)

lw s4,8(sp)

lw s5,4(sp)

addi sp,sp,32

jr ra

.size array\_read, .-array\_read

.align 2

.globl array\_shift

.type array\_shift, @function

array\_shift:

ble a2,zero,.L16

slli a6,a0,2

add a6,a6,a1

li a7,0

j .L18

.L21:

mv a5,a1

mv a4,a7

.L19:

mv a3,a4

lw a4,0(a5)

sw a3,0(a5)

addi a5,a5,4

bne a5,a6,.L19

.L20:

sw a4,0(a1)

addi a2,a2,-1

beq a2,zero,.L16

.L18:

bgt a0,zero,.L21

j .L20

.L16:

ret

.size array\_shift, .-array\_shift

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

Ассемблирование

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

riscv64-unknown-elf-gcc.exe –march=rv32i –mabi=ilp32 -c main.s –o main.o

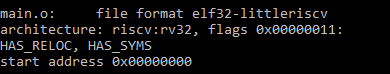
riscv64-unknown-elf-gcc.exe –march=rv32i –mabi=ilp32 -c shift.s –o shift.o

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.

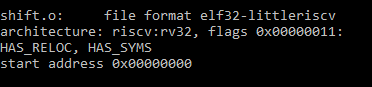
Листинг 2.5. Файл main.o

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

riscv64-unknown-elf-objdump.exe -f main.o



riscv64-unknown-elf-objdump.exe -f shift.o

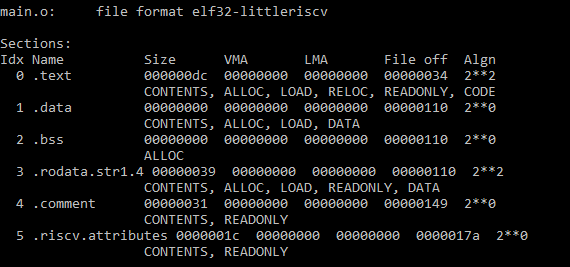


Оба файла содержат таблицу перемещений (в списке флагов есть флага HAS\_RELOC)

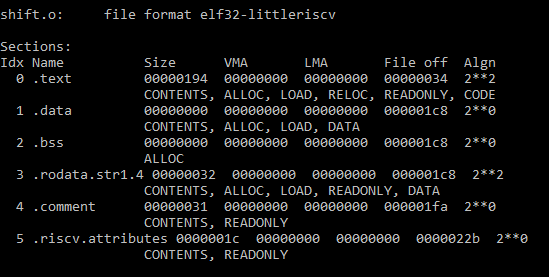
Листинг 2.6. Заголовки секций файла main.o

Выведем все заголовки секций объектных файлов

riscv64-unknown-elf-objdump -h main.o



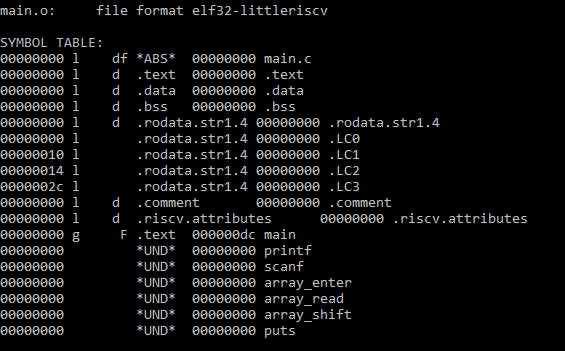
riscv64-unknown-elf-objdump -h shift.o



Выведем таблицы символов файлов shift.o и main.o

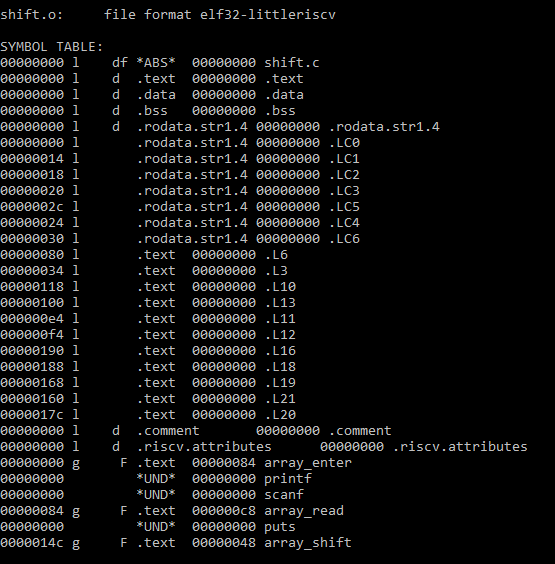
riscv64-unknown-elf-objdump -t main.o

Листинг 2.7. Таблица символов файла main.o



Листинг 2.8. Таблица символов файла shift.o

riscv64-unknown-elf-objdump -t shift.o

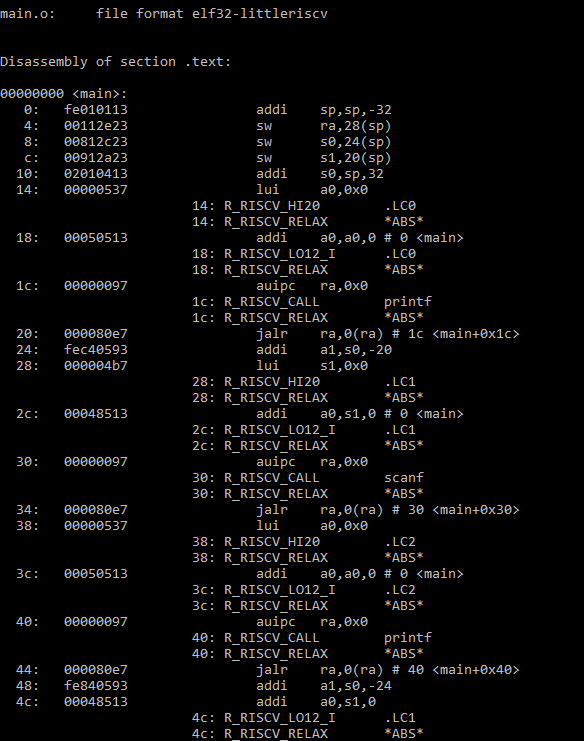


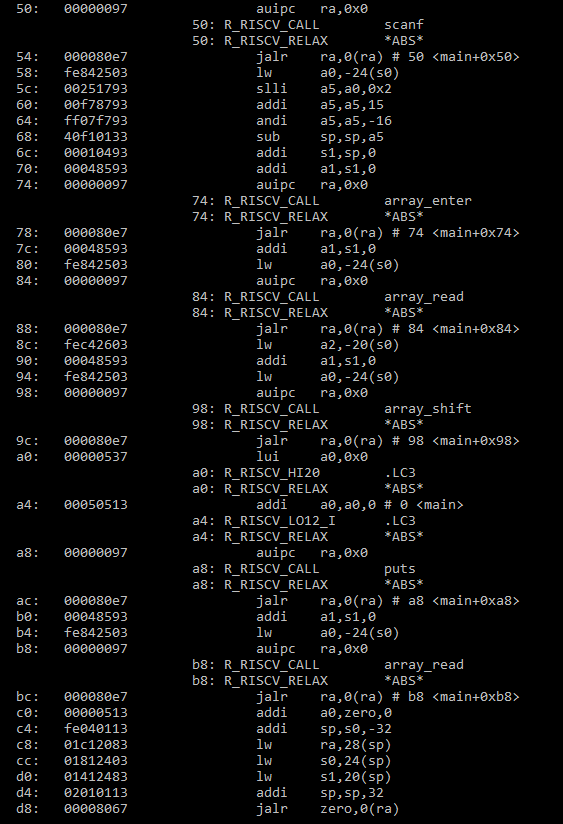
В каждой таблице только одни глобальный (флаг “g”) символ типа «функция» (“F”) – “shift” и “main” соответственно.

Листинг 2.9. файл main.o

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

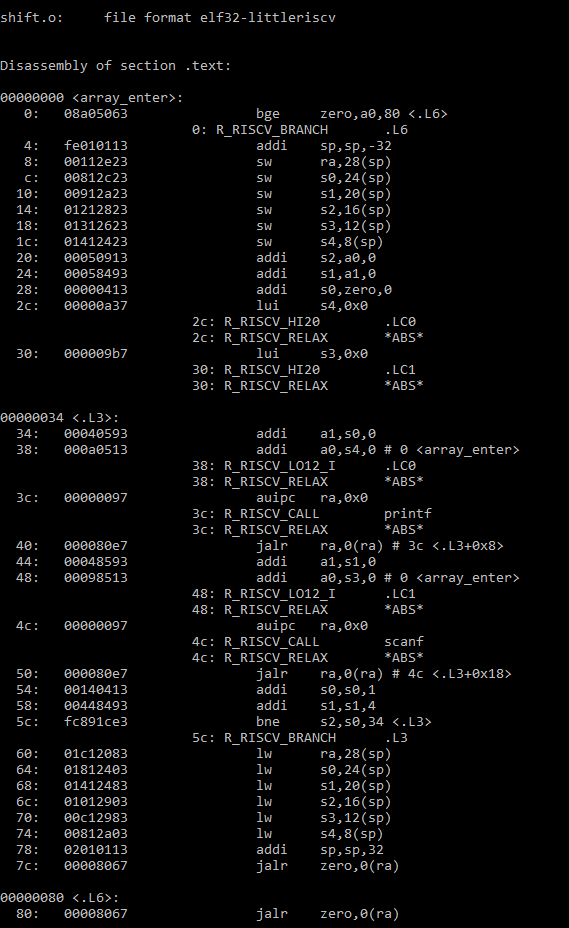
riscv64-unknown-elf-objdump.exe -d -M no-aliases -r main.o

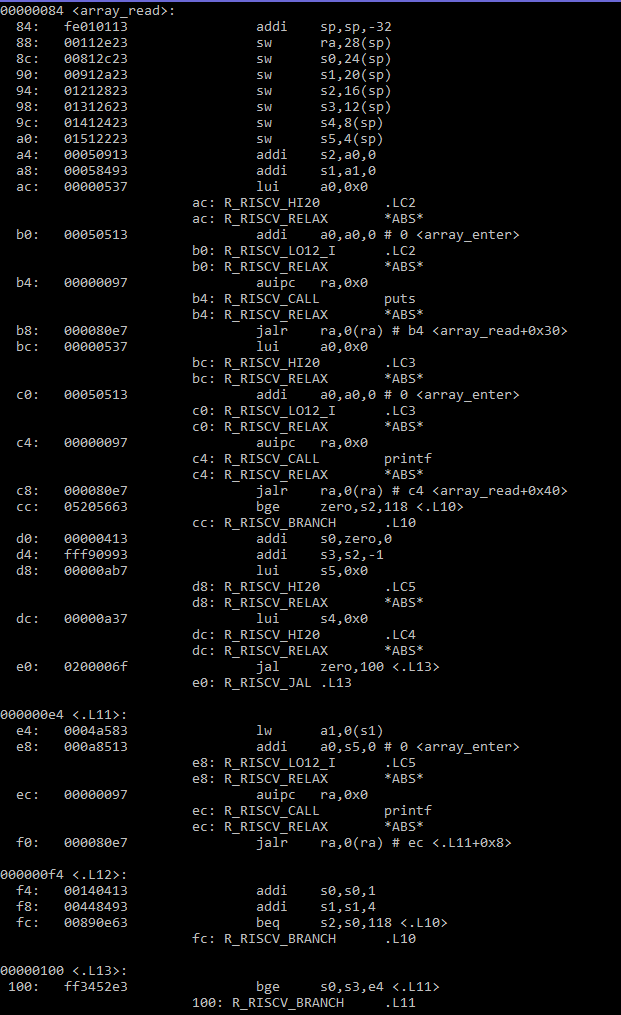


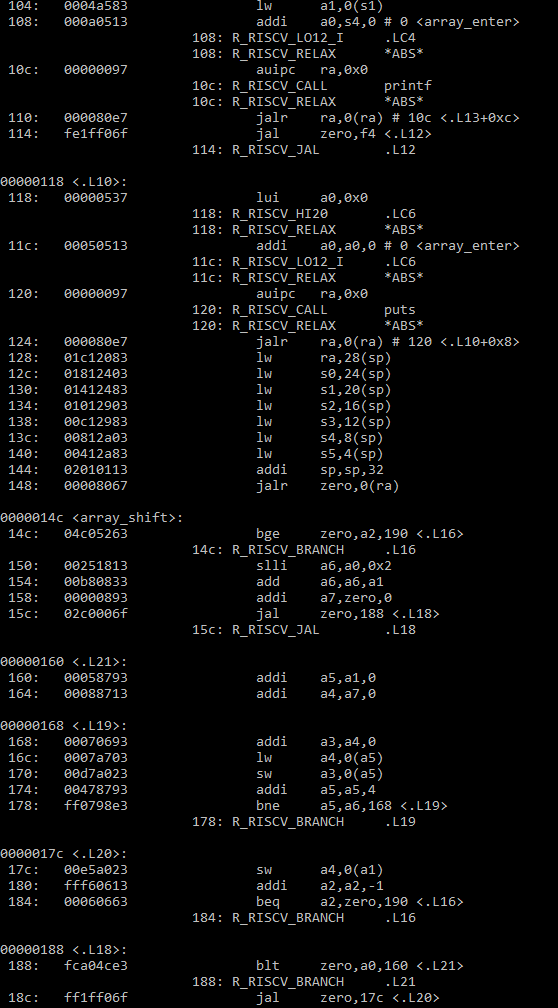


Листинг 2.10. файл shift.o

riscv64-unknown-elf-objdump.exe -d -M no-aliases -r main.o







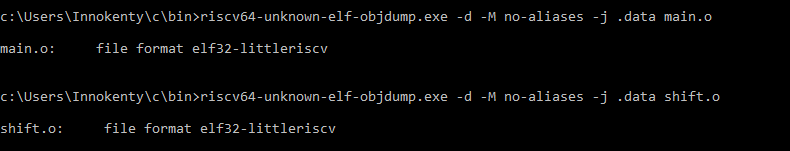


Дизассемблированный код практически идентичен сгенерированному (за исключением псевдоинструкций).

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

riscv64-unknown-elf-objdump.exe -d -M no-aliases –j .data main.o

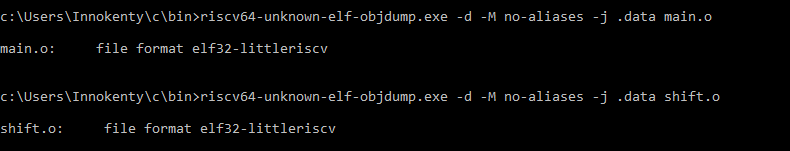
riscv64-unknown-elf-objdump.exe -d -M no-aliases –j .data shift.o



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

riscv64-unknown-elf-objdump.exe -d -M no-aliases –j .bss main.o

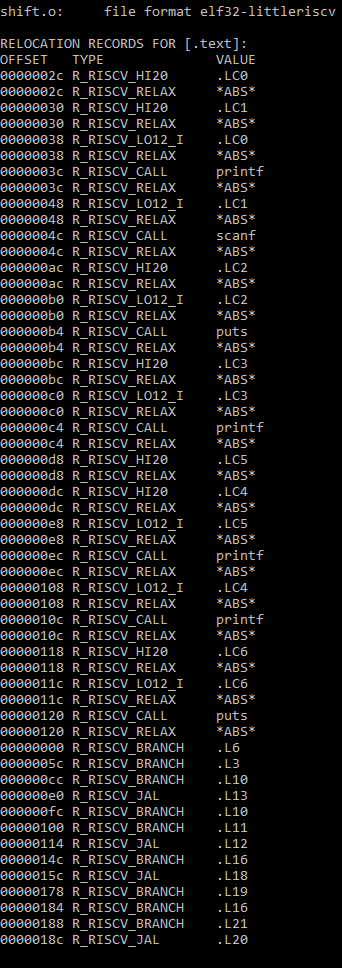
riscv64-unknown-elf-objdump.exe -d -M no-aliases –j .bss shift.o



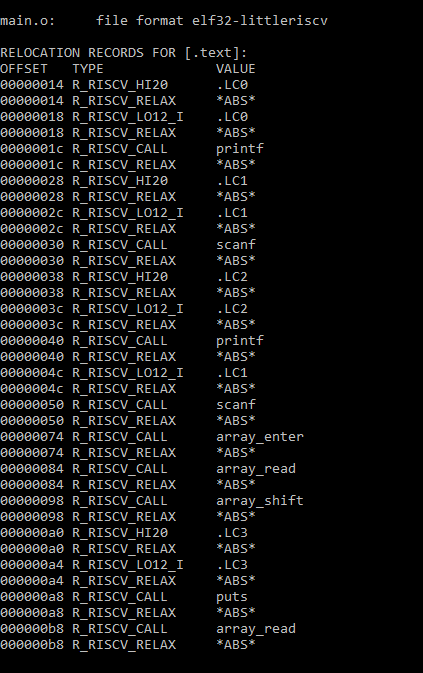
Листинг 2.11. таблицы перемещений shift.o

Выведем таблицы перемещений объектных файлов:

riscv64-unknown-elf-objdump.exe -r shift.o main.o



Листинг 2.12. таблицы перемещений main.o



В таблицах перемещения shift.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 shift. Записи типа R\_RISCV\_RELAX заносятся в таблицу перемещений в дополнение к записям типа R\_RISCV\_CAL.

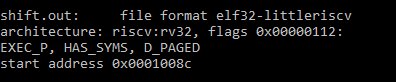
Компановка

Следующим шагом является компоновка и формирование исполняемых фалов программ:



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

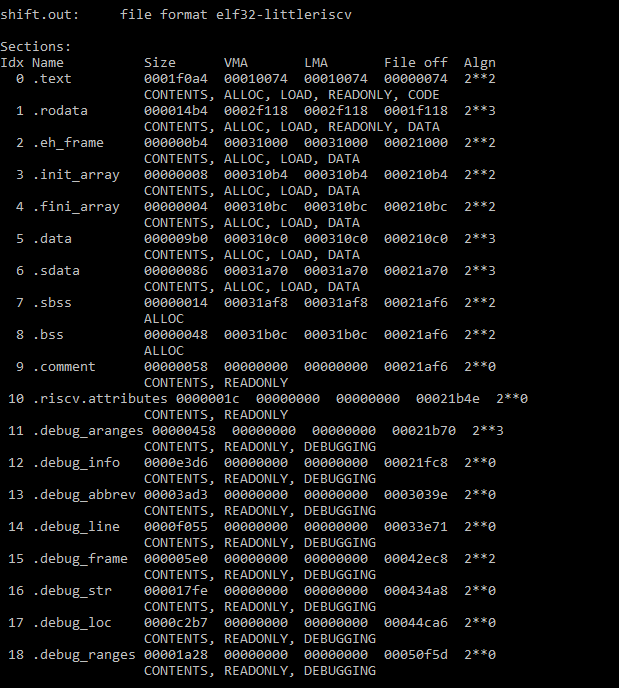
riscv64-unknown-elf-objdump.exe -f shift.out



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

Перечислим секции исполняемого файла:

riscv64-unknown-elf-objdump -h shift.out

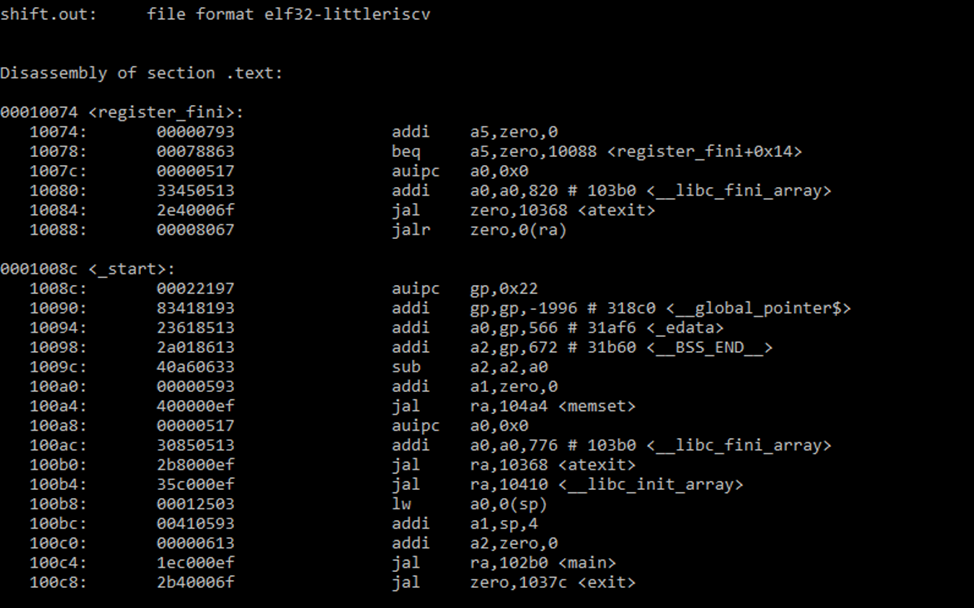


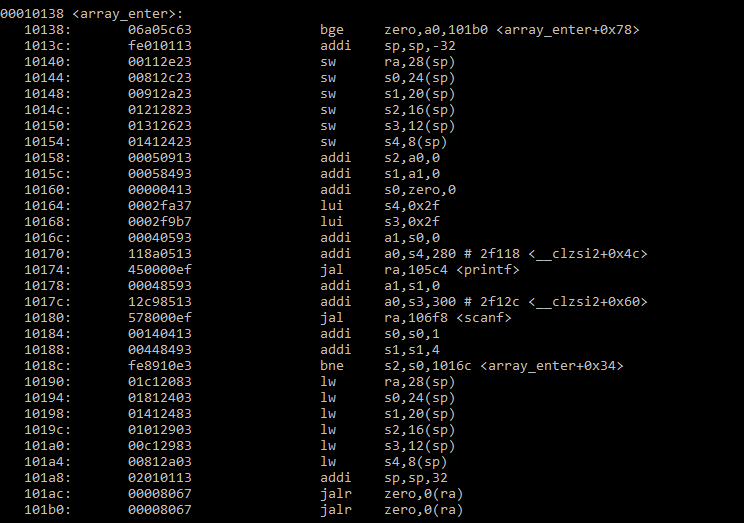
В исполняемом файле действительно производится слияние содержания секций обоих объектных файлов, а также значительное расширение списка секций новыми блоками.

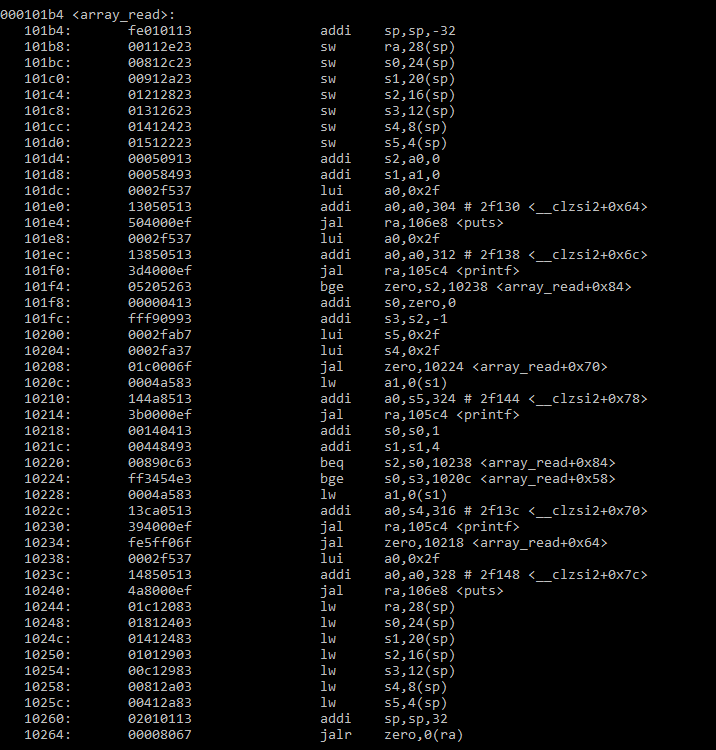
Проанализируем содержимое секции .textисполняемого файла:

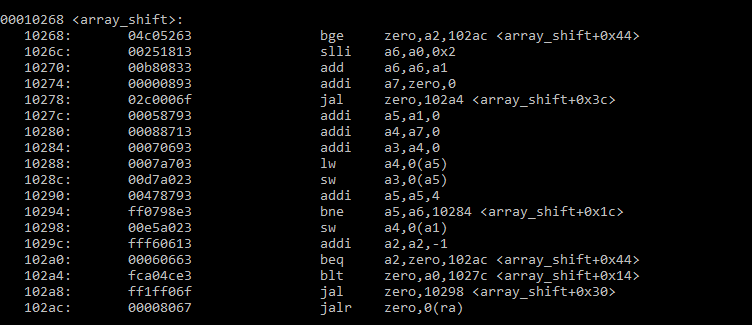
riscv64-unknown-elf-objdump.exe –d –M no-aliases –j .text shift.out

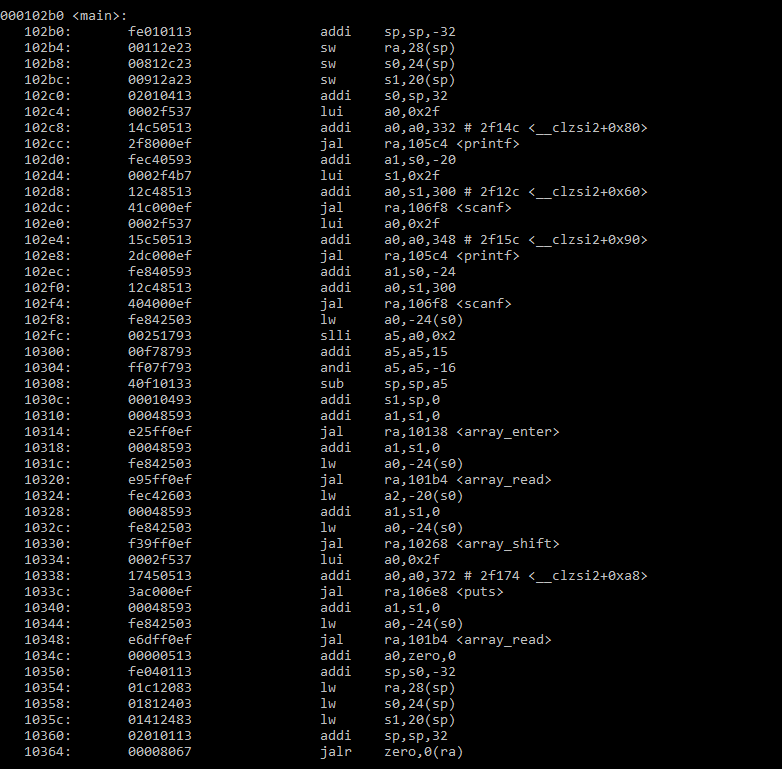
Секция кода теперь содержит намного большее количество строк, поэтому рассмотрим только самые важные участки:

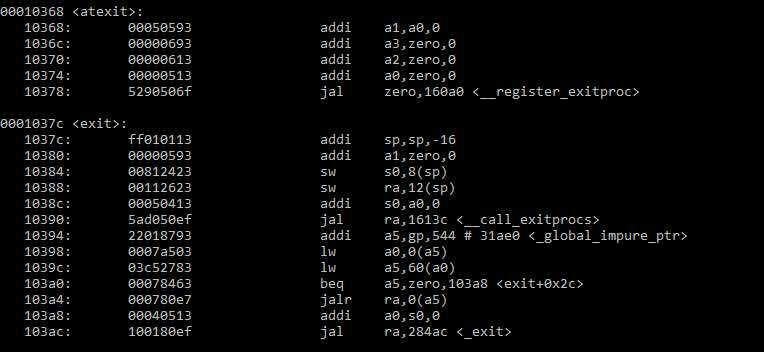








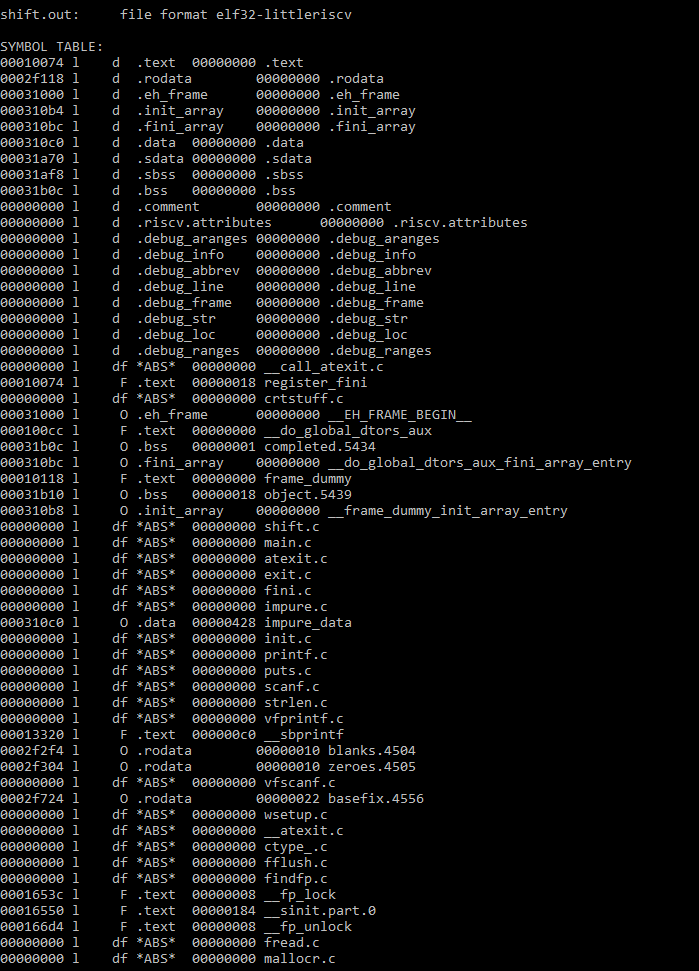




Компоновщик все переходы auipc+jalr, заменил на одну инструкцию jal и корректным адресом перехода. Выведем таблицу символов исполняемого файла:

riscv64-unknown-elf-objdump -t shift.out

Листинг 2.13. Таблица символов исполняемого файла



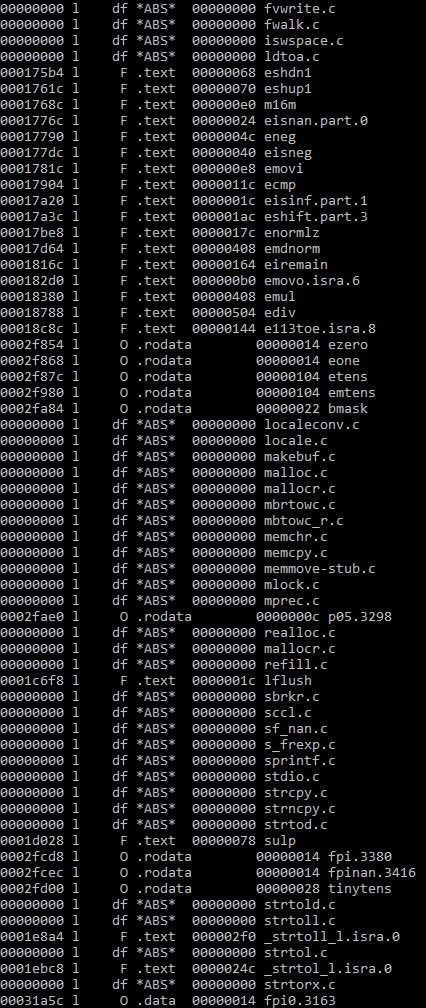


Таблица символов содержит множество дополнительных вхождений, однако в целом определяет все нужные секции, метки и адреса.

Проанализируем таблицу перемещенийисполняемого файла

riscv64-unknown-elf-objdump.exe -r shift.out

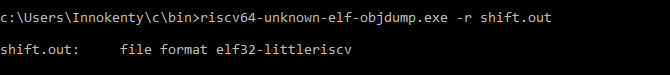


Таблица перемещений оказывается пуста, все необходимые релокации, оптимизации и замены инструкций были успешно проведены компоновщиком.

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

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:

Получим объектный файл shift.o

**riscv64-unknown-elf-ar.exe -rsc libshift.a shift.o**

Параметры:

-r – заменить старые файлы с такими названиями (adder.o), если они уже есть в архиве;

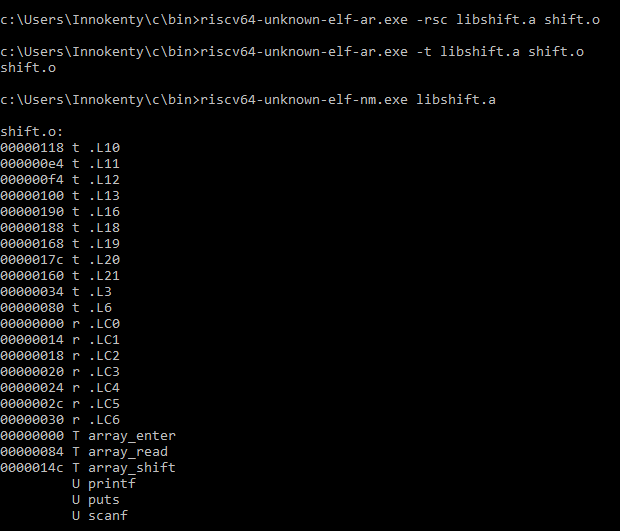
-s – записать «index» в архив. Index – это список всех символов, объявленных во включенных в архив объектных файлах, и его присутствие ускоряет линковку;

-с – создать архив, если его еще не было.

**riscv64-unknown-elf-ar.exe -t libshift.a shift.o**

**riscv64-unknown-elf-nm.exe libshift.a**

Листинг 3.1. файл shift.o



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

Листинг 3.2. Makefile для создания статической библиотеки

# Цели

.PHONY: all clean

# Исходные файлы для сборки библиотеки

OBJS= shift.c

#Вызываемые приложения

AR = riscv64-unknown-elf-ar.exe

CC = riscv64-unknown-elf-gcc.exe

# Файл библиотеки

MYLIBNAME = libshift.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)

# $^ = (shift.o)

$(MYLIBNAME): shift.o

$(AR) -rsc $@ $^

Листинг 3.3. Makefile для сборки исполняемого файла

# Цели

.PHONY: all clean

# Исходные файлы для сборки библиотеки

OBJS= main.c \

libshift.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 данной лабораторной работы была разработана программа на языке C, программа соответствует заданному варианту «#17 Циклический сдвиг массива чисел на заданное количество разрядов вправо»

Также были закреплены знания языка C, ассемблера RISC-V, получены навыки работы с препроцессором, компилятором, ассемблером и компоновщиком пакета GCC и 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. В ходе лабораторной работы были изучены особенности каждого этапа пошаговой сборки набора программ, а также инструменты, позволяющие выделить разработанные программы в статическую библиотеку и автоматизировать сборку этой библиотеки.