Санкт-Петербургский политехнический университет Петра Великого

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Отчёт по лабораторной работе № 4**

**Дисциплина:** Низкоуровневое программирование

**Тема:** Раздельная компиляция

Выполнил студент гр. 3530901/90003 Зотов М. С.

(подпись)

Преподаватель Алексюк А.О.

(подпись)

“ ” 2021 г.

Санкт-Петербург

2021

**Оглавление**

[1. ТЗ 3](#_Toc69506730)

[2. Метод решения 3](#_Toc69506731)

[3. Ход решения 3](#_Toc69506732)

[3.1 Тексты программ, реализующих определенную вариантом задания функциональность 3](#_Toc69506733)

[3.2 Сборка программ «по шагам», анализ промежуточных и результирующих файлов 4](#_Toc69506734)

[3.3 Формирование статической библиотеки, разработка make-файлов для сборки библиотеки 32](#_Toc69506735)

[4. Выводы 35](#_Toc69506736)

# 

# **1. ТЗ**

1) На языке C разработать функцию, выполняющую *реверс массива чисел*. 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.

# **2. Метод решения**

Алгоритм решения:

1) В начальный момент у нас есть элементы массива *array*. Также есть 2 переменные, содержащие адреса крайнего левого непереставленного элемента (*leftIndex*) и крайнего правого непереставленного элемента (*rightIndex*).

2) В ходе исполнения программы на каждой итерации цикла *while* мы получаем левый элемент по индексу *leftIndex* приравниваем его переменной *temp*. Затем мы выполняем *array[leftIndex] = array[rightIndex]*. После этого в *array[rightIndex]* записываем значение *temp*. Далее увеличиваем/уменьшаем *leftIndex* / *rightIndex* на единицу.

3) Цикл while выполняется, пока *leftIndex* - *rightIndex* < 0.

# **3. Ход решения**

# **3.1 Тексты программ, реализующих определенную вариантом задания функциональность**

*Листинг 1. Программа “reverseProgram.c”*

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

int leftIndex = 0;

int rightIndex = size - 1;

while (leftIndex - rightIndex < 0) {

int temp = array[leftIndex];

array[leftIndex] = array[rightIndex];

array[rightIndex] = temp;

leftIndex++;

rightIndex--;

}

}

*Листинг 2. Заголовочный файл “ reverseProgram.h”*

#ifndef LAB4\_REVERSEPROGRAM\_H

#define LAB4\_REVERSEPROGRAM\_H

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

#endif

*Листинг 3. Программа “test.c”*

#include <stdio.h>

#include "reverseProgram.h"

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

printf("Before: ");

for (int i = 0; i < size; i++)

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

reverse(array, size);

printf("\nAfter: ");

for (int i = 0; i < size; i++)

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

}

int main() {

printf("The first test: \n");

const int sizeForFirstTest = 5;

int arrayForFirstTest[] = { 1, 2, 3, 4, 5 };

test(arrayForFirstTest, sizeForFirstTest);

printf("\nThe second test: \n");

const int sizeForSecondTest = 6;

int arrayForSecondTest[] = { 1, 2, 3, 4, 5, 6 };

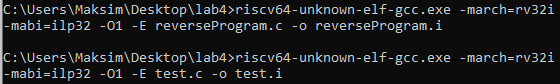
test(arrayForSecondTest, sizeForSecondTest);

return 0;

}

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

Приступим к пошаговой сборке написанных на языке C программ. Первым шагом является препроцессирование файлов исходного текста reverseProgram.c и test. в файлы reverseProgram.i и test.i:



Драйвер компилятора gcc riscv64-unknown-elf-gcc запускается с параметрами командной строки -march=rv32i и -mabi=ilp32, которые указывают, что целевым является процессор с базовой архитектурой системы команд RV32I. Параметр -O1 указывает, что надо выполнять простые оптимизации генерируемого кода, параметр -E – что надо остановить процесс сборки после препроцессирования.

Теперь ознакомимся с результатами работы препроцессора, то есть посмотрим содержимое построенных .i-файлов:

*Листинг 4. Файл “* *reverseProgram.i”*

# 1 "reverseProgram.c"

# 1 "<built-in>"

# 1 "<command-line>"

# 1 "reverseProgram.c"

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

int leftIndex = 0;

int rightIndex = size - 1;

while (leftIndex - rightIndex < 0) {

int temp = array[leftIndex];

array[leftIndex] = array[rightIndex];

array[rightIndex] = temp;

leftIndex++;

rightIndex--;

}

}

*Листинг 5. Файл “test.i”*

# 1 "test.c"

# 1 "<built-in>"

# 1 "<command-line>"

# 1 "test.c"

# 1 "c:\\riscv64\\riscv64-unknown-elf\\include\\stdio.h" 1 3

**…**

**(!) *В программе используется директива #include<stdio.h>, поэтому в самом файле reverseProgram.i в начале файла содержится порядка 1200 строк с инструкциями по линковке stdio.h к проекту, а затем следует код на C, который соответствует программе:***

# 2 "test.c" 2

# 1 "reverseProgram.h" 1

# 4 "reverseProgram.h"

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

# 3 "test.c" 2

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

printf("Before: ");

for (int i = 0; i < size; i++)

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

reverse(array, size);

printf("\nAfter: ");

for (int i = 0; i < size; i++)

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

}

int main() {

printf("The first test: \n");

const int sizeForFirstTest = 5;

int arrayForFirstTest[] = { 1, 2, 3, 4, 5 };

test(arrayForFirstTest, sizeForFirstTest);

printf("\nThe second test: \n");

const int sizeForSecondTest = 6;

int arrayForSecondTest[] = { 1, 2, 3, 4, 5, 6 };

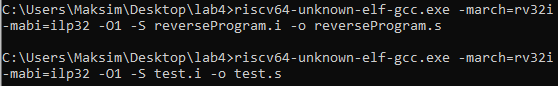
test(arrayForSecondTest, sizeForSecondTest);

return 0;

}

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

Следующим шагом является компиляция файлов reverseProgram.i и test.i в код на языке ассемблера reverseProgram.s и test.s:



Драйвер компилятора riscv64-unknown-elf-gcc запускается с параметрами, один из которых -S. -S указывает, что нужно остановить процесс сборки после компиляции (без запуска ассемблера).

Проанализируем получившийся код на языке ассемблера:

*Листинг 6. Файл “maxPrime.s”*

.file "reverseProgram.c"

.option nopic

.attribute arch, "rv32i2p0"

.attribute unaligned\_access, 0

.attribute stack\_align, 16

.text

.align 2

.globl reverse

.type reverse, @function

reverse:

li a5,1

sub a5,a5,a1

bge a5,zero,.L1

mv a5,a0

slli a4,a1,2

add a0,a0,a4

li a4,3

sub a1,a4,a1

.L3:

lw a4,0(a5)

lw a3,-4(a0)

sw a3,0(a5)

sw a4,-4(a0)

addi a5,a5,4

addi a0,a0,-4

mv a4,a1

addi a1,a1,2

blt a4,zero,.L3

.L1:

ret

.size reverse, .-reverse

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

Как можно видеть, по метке reverse начинается тело самой функции, выполняющей перестановку элементов массива в обратную сторону.

*Листинг 7. Файл “test.s”*

.file "test.c"

.option nopic

.attribute arch, "rv32i2p0"

.attribute unaligned\_access, 0

.attribute stack\_align, 16

.text

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

.align 2

.LC2:

.string "Before: "

.align 2

.LC3:

.string "%d "

.align 2

.LC4:

.string "\nAfter: "

.text

.align 2

.globl test

.type test, @function

test:

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)

sw s6,0(sp)

mv s6,a0

mv s4,a1

lui a0,%hi(.LC2)

addi a0,a0,%lo(.LC2)

call printf

ble s4,zero,.L2

mv s1,s6

mv s2,s6

li s0,0

lui s5,%hi(.LC3)

.L3:

lw a1,0(s2)

addi a0,s5,%lo(.LC3)

call printf

mv s3,s0

addi s0,s0,1

addi s2,s2,4

bne s4,s0,.L3

mv a1,s0

mv a0,s6

call reverse

lui a0,%hi(.LC4)

addi a0,a0,%lo(.LC4)

call printf

li s0,0

lui s2,%hi(.LC3)

.L5:

lw a1,0(s1)

addi a0,s2,%lo(.LC3)

call printf

mv a5,s0

addi s0,s0,1

addi s1,s1,4

bne s3,a5,.L5

.L1:

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)

lw s6,0(sp)

addi sp,sp,32

jr ra

.L2:

mv a1,s4

mv a0,s6

call reverse

lui a0,%hi(.LC4)

addi a0,a0,%lo(.LC4)

call printf

j .L1

.size test, .-test

.section .rodata.str1.4

.align 2

.LC5:

.string "The first test: "

.align 2

.LC6:

.string "\nThe second test: "

.text

.align 2

.globl main

.type main, @function

main:

addi sp,sp,-64

sw ra,60(sp)

sw s0,56(sp)

lui a0,%hi(.LC5)

addi a0,a0,%lo(.LC5)

call puts

lui s0,%hi(.LANCHOR0)

addi s0,s0,%lo(.LANCHOR0)

lw a1,0(s0)

lw a2,4(s0)

lw a3,8(s0)

lw a4,12(s0)

lw a5,16(s0)

sw a1,28(sp)

sw a2,32(sp)

sw a3,36(sp)

sw a4,40(sp)

sw a5,44(sp)

li a1,5

addi a0,sp,28

call test

lui a0,%hi(.LC6)

addi a0,a0,%lo(.LC6)

call puts

lw a0,20(s0)

lw a1,24(s0)

lw a2,28(s0)

lw a3,32(s0)

lw a4,36(s0)

lw a5,40(s0)

sw a0,4(sp)

sw a1,8(sp)

sw a2,12(sp)

sw a3,16(sp)

sw a4,20(sp)

sw a5,24(sp)

li a1,6

addi a0,sp,4

call test

li a0,0

lw ra,60(sp)

lw s0,56(sp)

addi sp,sp,64

jr ra

.size main, .-main

.section .rodata

.align 2

.set .LANCHOR0,. + 0

.LC0:

.word 1

.word 2

.word 3

.word 4

.word 5

.LC1:

.word 1

.word 2

.word 3

.word 4

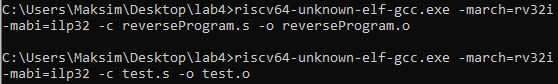
.word 5

.word 6

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

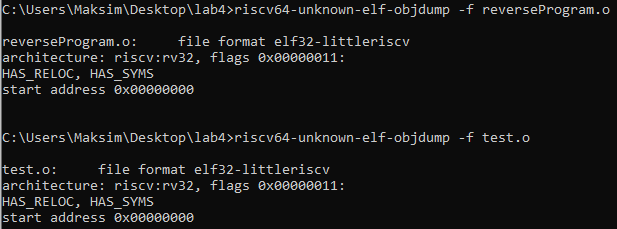
По метке main начинается тестовая программа, вызывающая функцию test, которая внутри себя уже вызывает reverseProgram, однако reverseProgram ещё нигде не определен, потому что не проведено ассемблирование.

Следующим шагом является ассемблирование файлов reverseProgram.s и test.s в объектные файлы reverseProgram.o и test.o:



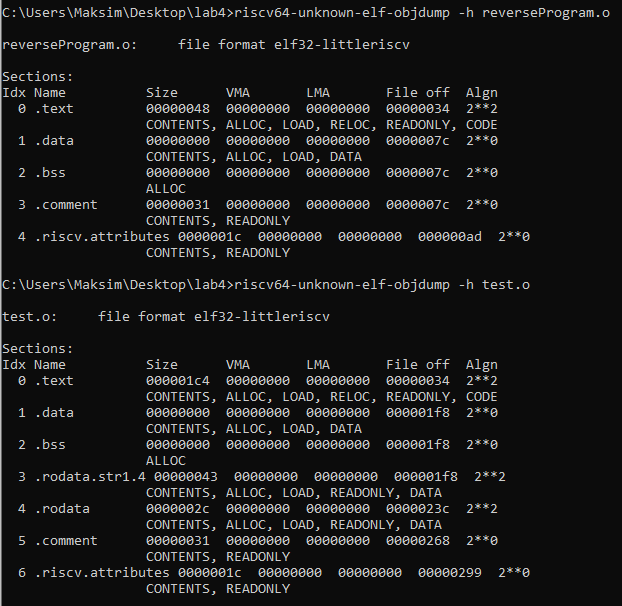
Драйвер компилятора riscv64-unknown-elf-gcc запускается с параметрами командной строки, один из которых -c – указание остановить процесс сборки после ассемблирования.

Так как объектные файлы не являются текстовыми, для просмотра их содержимого используется objdump:

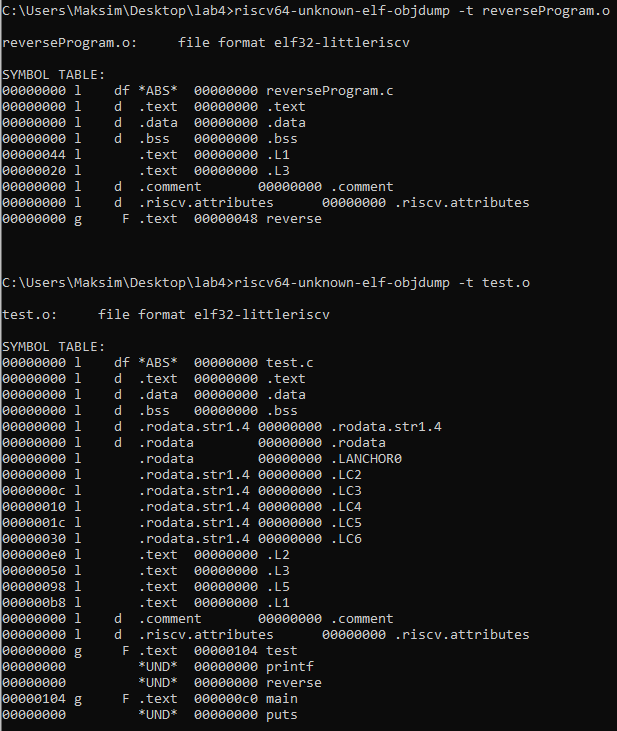


Оба файла содержат таблицу перемещений (в списке флагов есть флага HAS\_RELOC).

Выведем все заголовки секций объектных файлов:

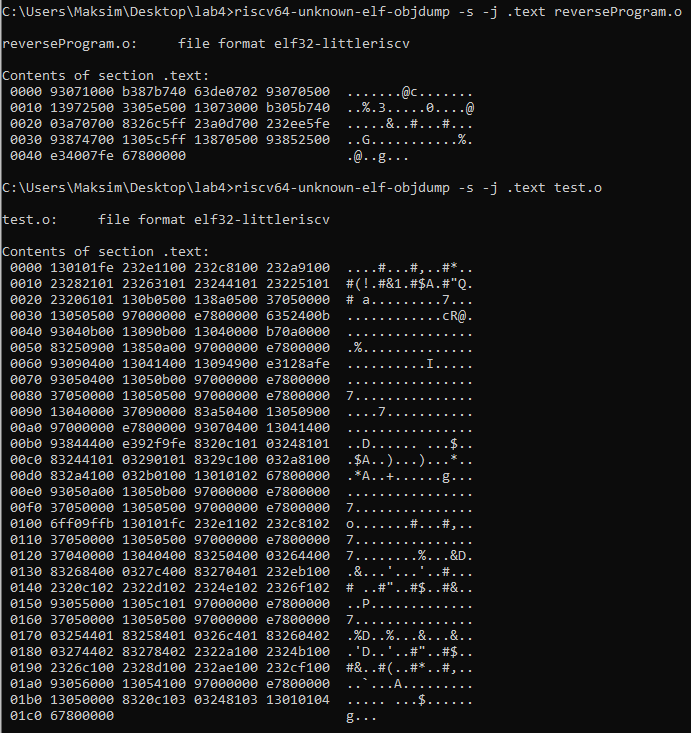


Посмотрим таблицы символов файлов reverseProgram.o и test.o:

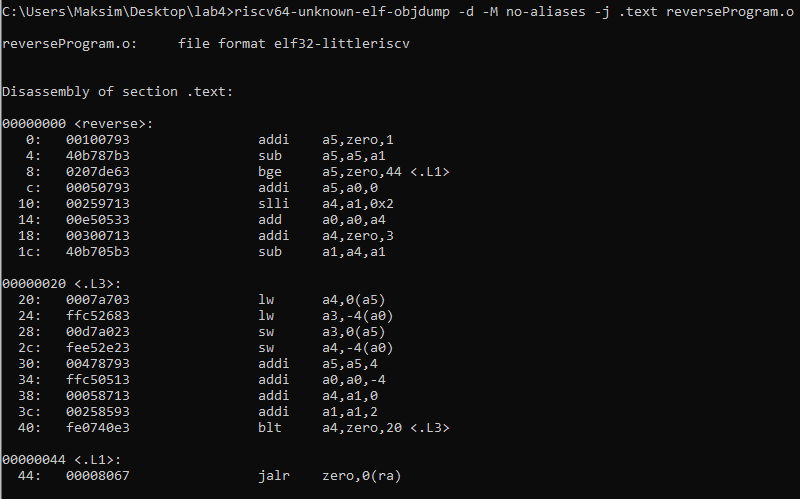


В таблице у reverseProgram.o только одни глобальный (флаг “g”) символ типа «функция» (“F”) – для reverse. В test.o таких символов 2 – для test и main.

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



Процедура декодирования кодов инструкций является «механической» (иначе как бы ее реализовывало техническое устройство – процессор), следовательно, разумно поручить ее выполнение ЭВМ:



Содержимое секции .text удобнее изобразить в виде текста:

test.o: file format elf32-littleriscv

Disassembly of section .text:

00000000 <test>:

0: fe010113 addi sp,sp,-32

4: 00112e23 sw ra,28(sp)

8: 00812c23 sw s0,24(sp)

c: 00912a23 sw s1,20(sp)

10: 01212823 sw s2,16(sp)

14: 01312623 sw s3,12(sp)

18: 01412423 sw s4,8(sp)

1c: 01512223 sw s5,4(sp)

20: 01612023 sw s6,0(sp)

24: 00050b13 addi s6,a0,0

28: 00058a13 addi s4,a1,0

2c: 00000537 lui a0,0x0

30: 00050513 addi a0,a0,0 # 0 <test>

34: 00000097 auipc ra,0x0

38: 000080e7 jalr ra,0(ra) # 34 <test+0x34>

3c: 0b405263 bge zero,s4,e0 <.L2>

40: 000b0493 addi s1,s6,0

44: 000b0913 addi s2,s6,0

48: 00000413 addi s0,zero,0

4c: 00000ab7 lui s5,0x0

00000050 <.L3>:

50: 00092583 lw a1,0(s2)

54: 000a8513 addi a0,s5,0 # 0 <test>

58: 00000097 auipc ra,0x0

5c: 000080e7 jalr ra,0(ra) # 58 <.L3+0x8>

60: 00040993 addi s3,s0,0

64: 00140413 addi s0,s0,1

68: 00490913 addi s2,s2,4

6c: fe8a12e3 bne s4,s0,50 <.L3>

70: 00040593 addi a1,s0,0

74: 000b0513 addi a0,s6,0

78: 00000097 auipc ra,0x0

7c: 000080e7 jalr ra,0(ra) # 78 <.L3+0x28>

80: 00000537 lui a0,0x0

84: 00050513 addi a0,a0,0 # 0 <test>

88: 00000097 auipc ra,0x0

8c: 000080e7 jalr ra,0(ra) # 88 <.L3+0x38>

90: 00000413 addi s0,zero,0

94: 00000937 lui s2,0x0

00000098 <.L5>:

98: 0004a583 lw a1,0(s1)

9c: 00090513 addi a0,s2,0 # 0 <test>

a0: 00000097 auipc ra,0x0

a4: 000080e7 jalr ra,0(ra) # a0 <.L5+0x8>

a8: 00040793 addi a5,s0,0

ac: 00140413 addi s0,s0,1

b0: 00448493 addi s1,s1,4

b4: fef992e3 bne s3,a5,98 <.L5>

000000b8 <.L1>:

b8: 01c12083 lw ra,28(sp)

bc: 01812403 lw s0,24(sp)

c0: 01412483 lw s1,20(sp)

c4: 01012903 lw s2,16(sp)

c8: 00c12983 lw s3,12(sp)

cc: 00812a03 lw s4,8(sp)

d0: 00412a83 lw s5,4(sp)

d4: 00012b03 lw s6,0(sp)

d8: 02010113 addi sp,sp,32

dc: 00008067 jalr zero,0(ra)

000000e0 <.L2>:

e0: 000a0593 addi a1,s4,0

e4: 000b0513 addi a0,s6,0

e8: 00000097 auipc ra,0x0

ec: 000080e7 jalr ra,0(ra) # e8 <.L2+0x8>

f0: 00000537 lui a0,0x0

f4: 00050513 addi a0,a0,0 # 0 <test>

f8: 00000097 auipc ra,0x0

fc: 000080e7 jalr ra,0(ra) # f8 <.L2+0x18>

100: fb9ff06f jal zero,b8 <.L1>

00000104 <main>:

104: fc010113 addi sp,sp,-64

108: 02112e23 sw ra,60(sp)

10c: 02812c23 sw s0,56(sp)

110: 00000537 lui a0,0x0

114: 00050513 addi a0,a0,0 # 0 <test>

118: 00000097 auipc ra,0x0

11c: 000080e7 jalr ra,0(ra) # 118 <main+0x14>

120: 00000437 lui s0,0x0

124: 00040413 addi s0,s0,0 # 0 <test>

128: 00042583 lw a1,0(s0)

12c: 00442603 lw a2,4(s0)

130: 00842683 lw a3,8(s0)

134: 00c42703 lw a4,12(s0)

138: 01042783 lw a5,16(s0)

13c: 00b12e23 sw a1,28(sp)

140: 02c12023 sw a2,32(sp)

144: 02d12223 sw a3,36(sp)

148: 02e12423 sw a4,40(sp)

14c: 02f12623 sw a5,44(sp)

150: 00500593 addi a1,zero,5

154: 01c10513 addi a0,sp,28

158: 00000097 auipc ra,0x0

15c: 000080e7 jalr ra,0(ra) # 158 <main+0x54>

160: 00000537 lui a0,0x0

164: 00050513 addi a0,a0,0 # 0 <test>

168: 00000097 auipc ra,0x0

16c: 000080e7 jalr ra,0(ra) # 168 <main+0x64>

170: 01442503 lw a0,20(s0)

174: 01842583 lw a1,24(s0)

178: 01c42603 lw a2,28(s0)

17c: 02042683 lw a3,32(s0)

180: 02442703 lw a4,36(s0)

184: 02842783 lw a5,40(s0)

188: 00a12223 sw a0,4(sp)

18c: 00b12423 sw a1,8(sp)

190: 00c12623 sw a2,12(sp)

194: 00d12823 sw a3,16(sp)

198: 00e12a23 sw a4,20(sp)

19c: 00f12c23 sw a5,24(sp)

1a0: 00600593 addi a1,zero,6

1a4: 00410513 addi a0,sp,4

1a8: 00000097 auipc ra,0x0

1ac: 000080e7 jalr ra,0(ra) # 1a8 <main+0xa4>

1b0: 00000513 addi a0,zero,0

1b4: 03c12083 lw ra,60(sp)

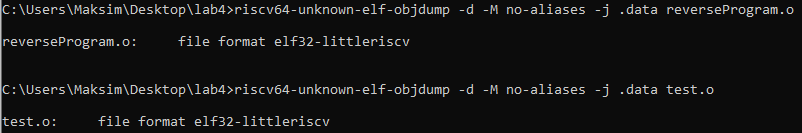
1b8: 03812403 lw s0,56(sp)

1bc: 04010113 addi sp,sp,64

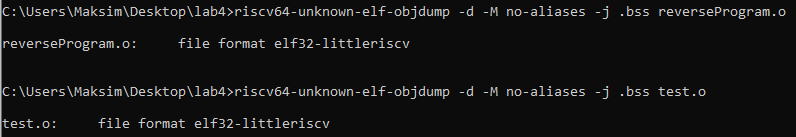
1c0: 00008067 jalr zero,0(ra)

Дизассемблированный код практически идентичен сгенерированному за исключением псевдо-инструкций.

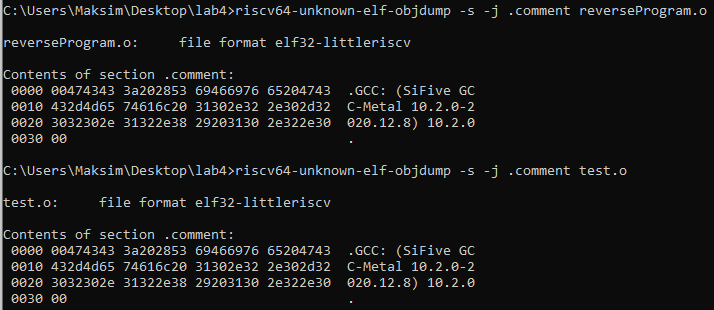
Секции .data объектных файлов – секции инициализированных данных – не содержат данных. Размер секций равен нулю:



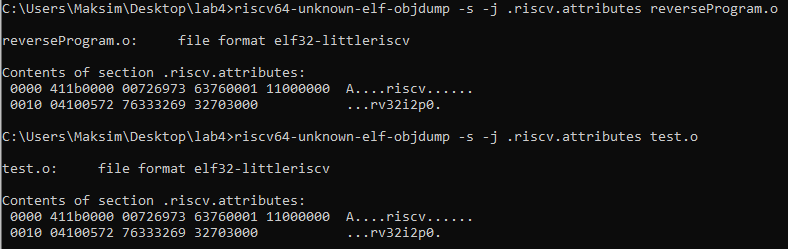
Секции .bss объектных файлов – секции данных, инициализированных нулями – тоже пусты:



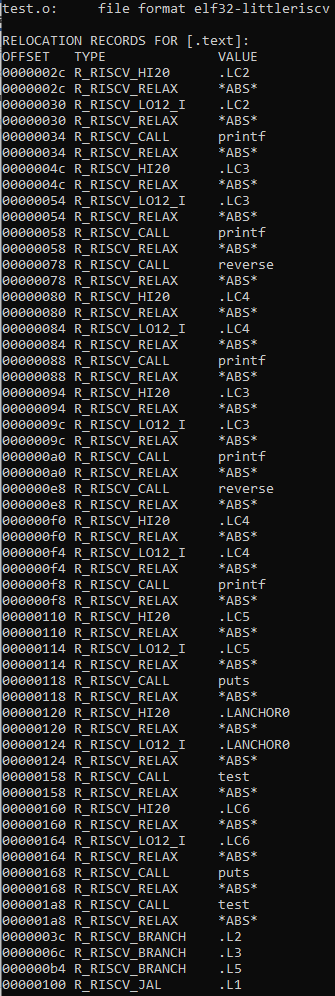
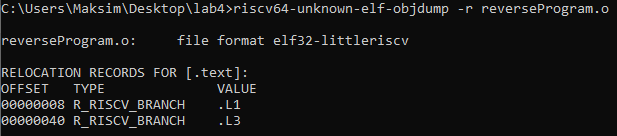
Секция .comment – секция данных о версиях – и для одного и для другого файла содержит одни и те же значения – сведения о GCC от SiFive:



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



Проанализируем таблицы перемещений объектных файлов:



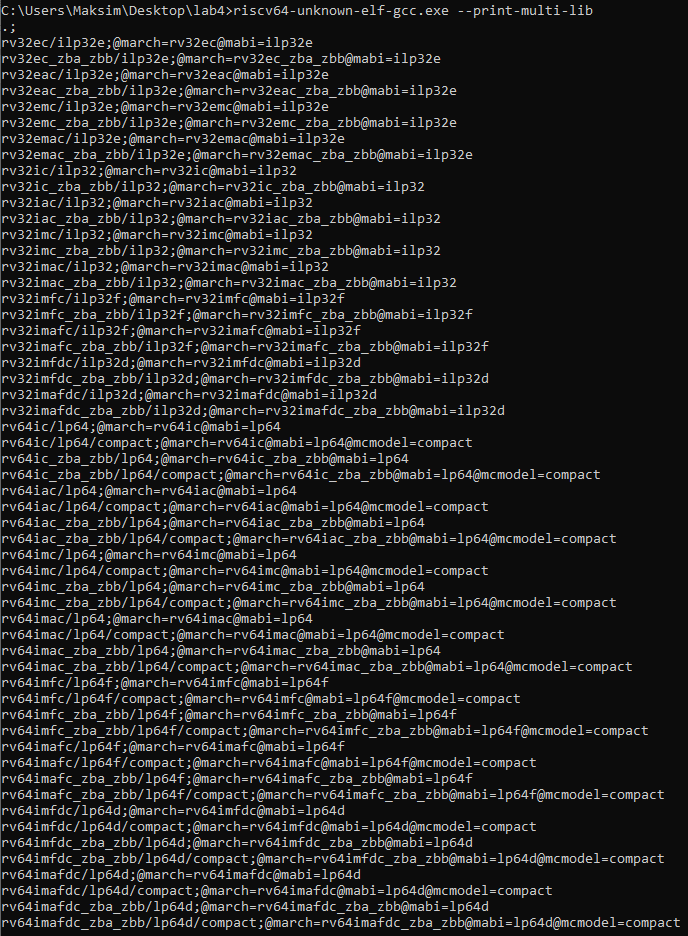
В таблицах перемещения reverseProgram.o содержится информация о переходах (R\_RISCV\_JAL) и ветвлениях(R\_RISCV\_BRANCH). В таблицах перемещения test.o, есть R\_RISCV\_CALL, чтобы информация о переходах соответствовала reverseProgram. Записи типа R\_RISCV\_RELAX заносятся в таблицу перемещений в дополнение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

Следующим шагом является компоновка и формирование исполняемых файлов программ:

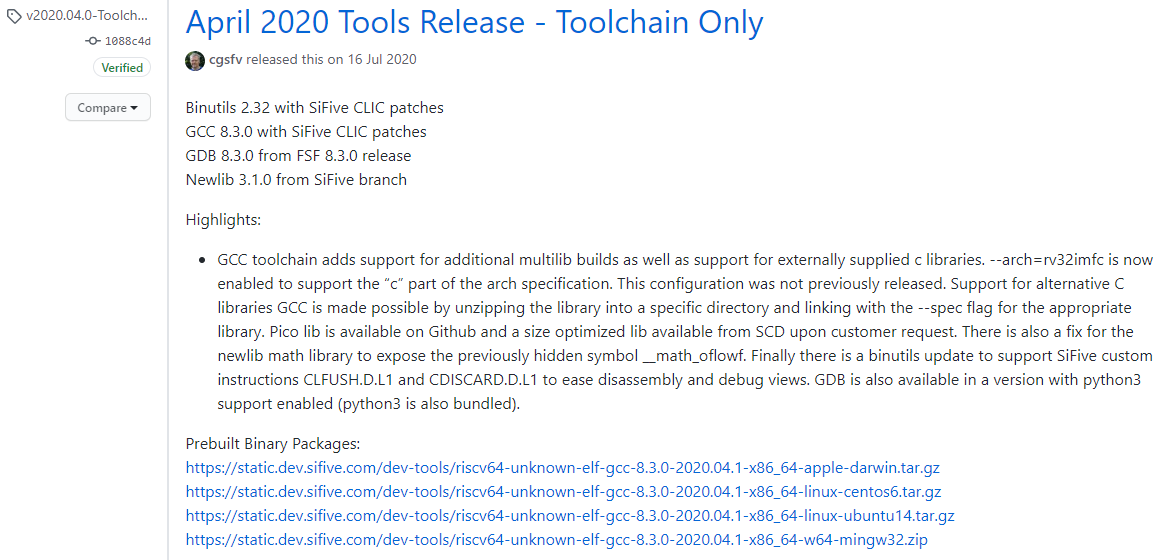


Как можно видеть, мы получили ошибку. Она говорит о том, что отсутствует библиотека для -march=rv32i -mabi=ilp32 (то есть, когда целевым является процессор с базовой архитектурой системы команд RV32I).

Действительно, если мы посмотрим список доступных библиотек, то интересующую нас не найдем:

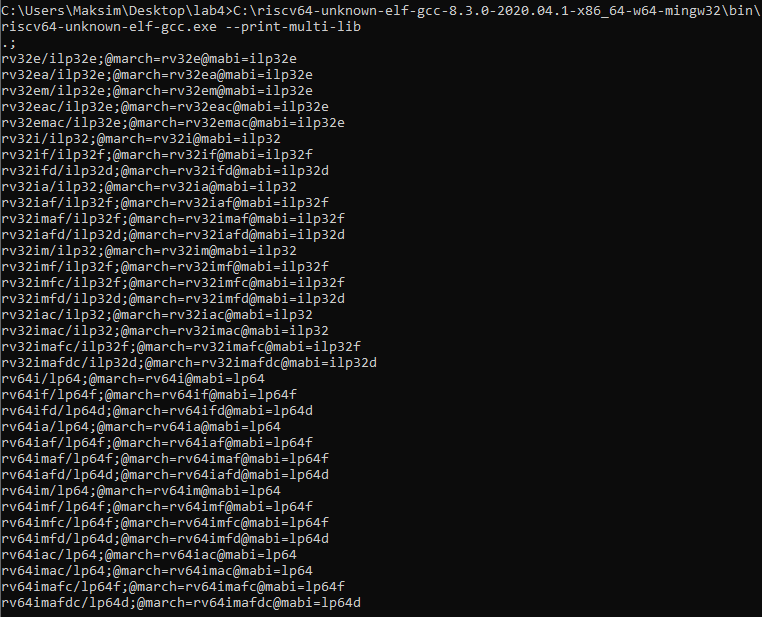


По какой-то причине в версии скачанного с оф. сайта *toolchain-а* эта библиотека отсутствует, однако она есть в более ранних релизах, например, здесь (<https://github.com/sifive/freedom-tools/releases/tag/v2020.04.0-Toolchain.Only>):



Будем теперь пользоваться новой (на самом деле старой) версией.

Посмотрим для начала список доступных библиотек:

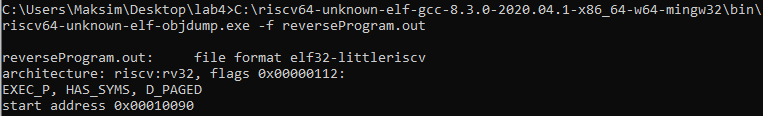


*6-ая* по счёту (rv32i/ilp32;…) как раз та, которая требуется.

Снова попробуем выполнить компоновку и формирование исполняемых файлов программ:

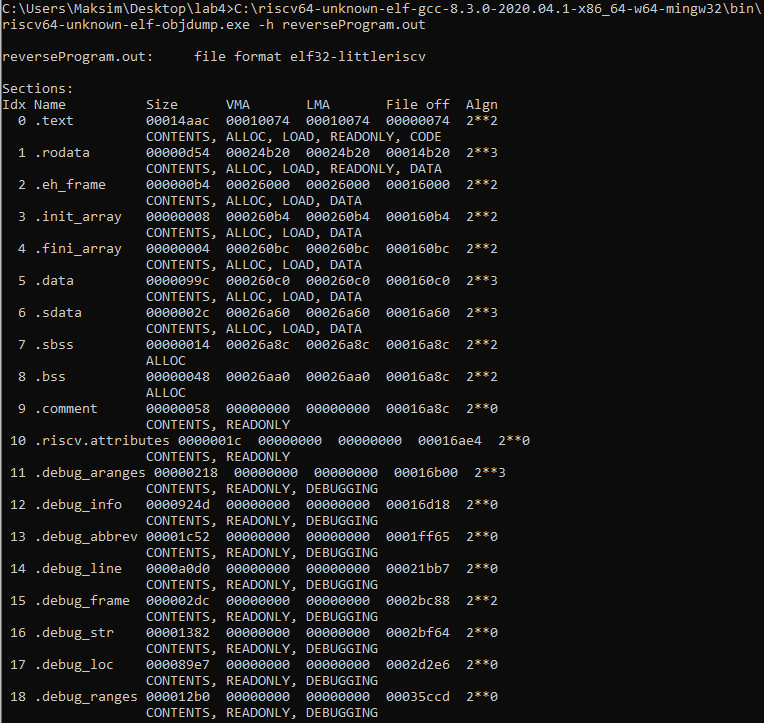


Сформированный компоновщиком файл “reverseProgram.out”, также является «бинарным»:



Флаг EXEC\_P указывает, что файл действительно является исполняемым, после загрузки его выполнение должно начаться с адреса 0x00010090 (entrypoint).

Перечислим секции исполняемого файла:

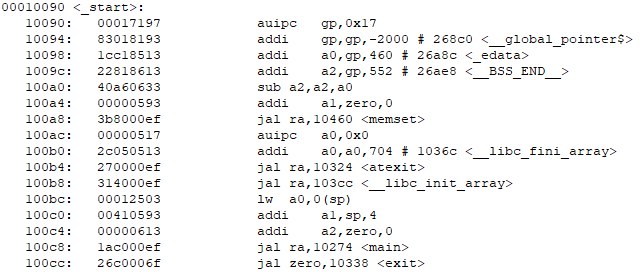


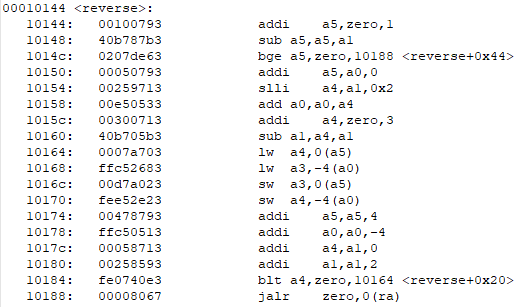
В исполняемом файле производится слияние содержания секций обоих объектных файлов, а также значительное расширение списка секций новыми блоками.

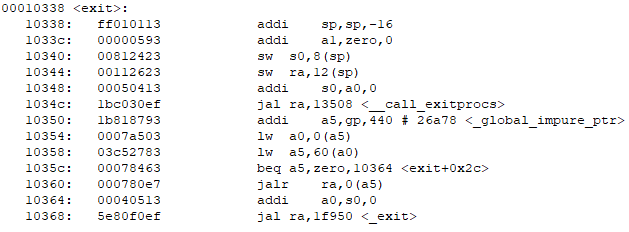
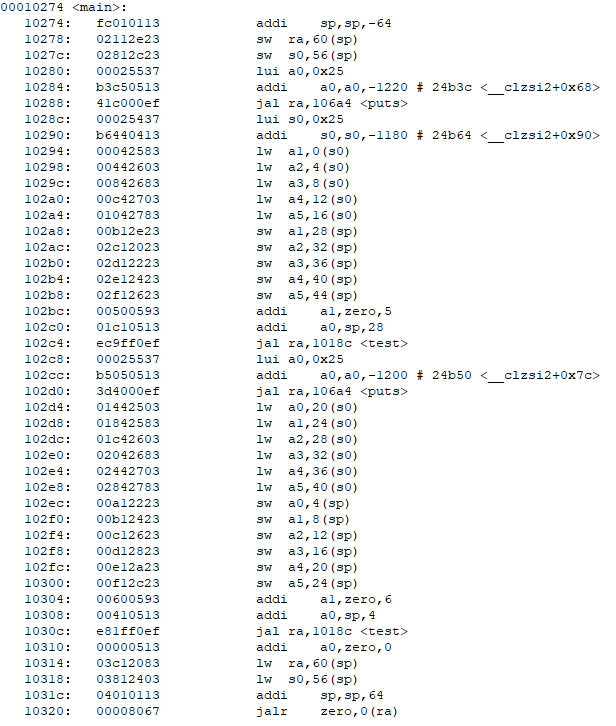
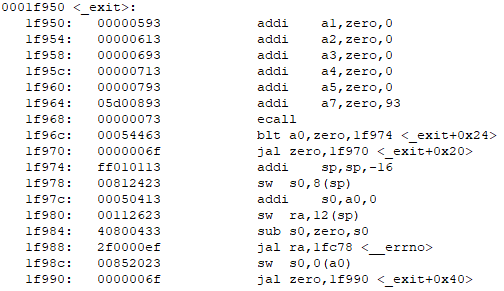
Проанализируем содержимое секции .text исполняемого файла:



Секция кода теперь содержит намного большее количество строк, поэтому мы назначили вывод в *log*. Рассмотрим только самые важные участки:

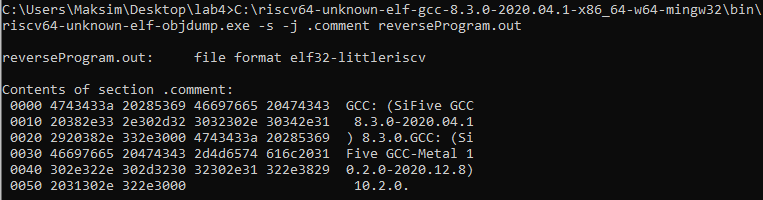




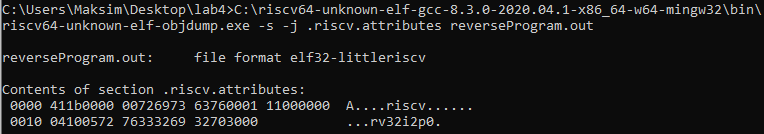
 

Компоновщик все переходы auipc+jalr, заменил на одну инструкцию jal и корректным адресом перехода.

Секция .comment содержит информацию о том, что мы меняли версию GCC:



Секция .riscv.attributes по-прежнему содержит информацию о том, что мы используем архитектуру команд RV32I:



Проанализируем таблицу символов исполняемого файла:

reverseProgram.out: file format elf32-littleriscv

SYMBOL TABLE:

00010074 l d .text 00000000 .text

00024b20 l d .rodata 00000000 .rodata

00026000 l d .eh\_frame 00000000 .eh\_frame

000260b4 l d .init\_array 00000000 .init\_array

000260bc l d .fini\_array 00000000 .fini\_array

000260c0 l d .data 00000000 .data

00026a60 l d .sdata 00000000 .sdata

00026a8c l d .sbss 00000000 .sbss

00026aa0 l d .bss 00000000 .bss

00000000 l d .comment 00000000 .comment

00000000 l d .riscv.attributes 00000000 .riscv.attributes

00000000 l d .debug\_aranges 00000000 .debug\_aranges

00000000 l d .debug\_info 00000000 .debug\_info

00000000 l d .debug\_abbrev 00000000 .debug\_abbrev

00000000 l d .debug\_line 00000000 .debug\_line

00000000 l d .debug\_frame 00000000 .debug\_frame

00000000 l d .debug\_str 00000000 .debug\_str

00000000 l d .debug\_loc 00000000 .debug\_loc

00000000 l d .debug\_ranges 00000000 .debug\_ranges

00000000 l df \*ABS\* 00000000 \_\_call\_atexit.c

00010074 l F .text 0000001c register\_fini

00000000 l df \*ABS\* 00000000 crtstuff.c

00026000 l O .eh\_frame 00000000 \_\_EH\_FRAME\_BEGIN\_\_

000100d0 l F .text 00000000 \_\_do\_global\_dtors\_aux

00026aa0 l O .bss 00000001 completed.5434

000260bc l O .fini\_array 00000000 \_\_do\_global\_dtors\_aux\_fini\_array\_entry

00010120 l F .text 00000000 frame\_dummy

00026aa4 l O .bss 00000018 object.5439

000260b8 l O .init\_array 00000000 \_\_frame\_dummy\_init\_array\_entry

00000000 l df \*ABS\* 00000000 reverseProgram.c

00000000 l df \*ABS\* 00000000 test.c

00000000 l df \*ABS\* 00000000 atexit.c

00000000 l df \*ABS\* 00000000 exit.c

00000000 l df \*ABS\* 00000000 fini.c

00000000 l df \*ABS\* 00000000 impure.c

000260c0 l O .data 00000428 impure\_data

00000000 l df \*ABS\* 00000000 init.c

00000000 l df \*ABS\* 00000000 printf.c

00000000 l df \*ABS\* 00000000 puts.c

00000000 l df \*ABS\* 00000000 strlen.c

00000000 l df \*ABS\* 00000000 vfprintf.c

00013244 l F .text 000000c0 \_\_sbprintf

00024d00 l O .rodata 00000010 blanks.4504

00024d10 l O .rodata 00000010 zeroes.4505

00000000 l df \*ABS\* 00000000 wsetup.c

00000000 l df \*ABS\* 00000000 \_\_atexit.c

00000000 l df \*ABS\* 00000000 fflush.c

00000000 l df \*ABS\* 00000000 findfp.c

00013908 l F .text 00000008 \_\_fp\_lock

0001391c l F .text 00000184 \_\_sinit.part.0

00013aa0 l F .text 00000008 \_\_fp\_unlock

00000000 l df \*ABS\* 00000000 mallocr.c

00000000 l df \*ABS\* 00000000 fvwrite.c

00000000 l df \*ABS\* 00000000 fwalk.c

00000000 l df \*ABS\* 00000000 ldtoa.c

000146cc l F .text 00000068 eshdn1

00014734 l F .text 00000070 eshup1

000147a4 l F .text 000000e0 m16m

00014884 l F .text 00000024 eisnan.part.0

000148a8 l F .text 0000004c eneg

000148f4 l F .text 00000040 eisneg

00014934 l F .text 000000e8 emovi

00014a1c l F .text 0000011c ecmp

00014b38 l F .text 0000001c eisinf.part.1

00014b54 l F .text 000001ac eshift.part.3

00014d00 l F .text 0000017c enormlz

00014e7c l F .text 00000408 emdnorm

00015284 l F .text 00000164 eiremain

000153e8 l F .text 000000b0 emovo.isra.6

00015498 l F .text 00000408 emul

000158a0 l F .text 00000504 ediv

00015da4 l F .text 00000144 e113toe.isra.8

00024d64 l O .rodata 00000014 ezero

00024d78 l O .rodata 00000014 eone

00024d8c l O .rodata 00000104 etens

00024e90 l O .rodata 00000104 emtens

00024f94 l O .rodata 00000022 bmask

00000000 l df \*ABS\* 00000000 localeconv.c

00000000 l df \*ABS\* 00000000 locale.c

00000000 l df \*ABS\* 00000000 makebuf.c

00000000 l df \*ABS\* 00000000 mallocr.c

00000000 l df \*ABS\* 00000000 mbtowc\_r.c

00000000 l df \*ABS\* 00000000 memchr.c

00000000 l df \*ABS\* 00000000 memcpy.c

00000000 l df \*ABS\* 00000000 memmove-stub.c

00000000 l df \*ABS\* 00000000 mlock.c

00000000 l df \*ABS\* 00000000 mprec.c

00024ff0 l O .rodata 0000000c p05.3298

00000000 l df \*ABS\* 00000000 mallocr.c

00000000 l df \*ABS\* 00000000 sbrkr.c

00000000 l df \*ABS\* 00000000 s\_frexp.c

00000000 l df \*ABS\* 00000000 sprintf.c

00000000 l df \*ABS\* 00000000 stdio.c

00000000 l df \*ABS\* 00000000 strcpy.c

00000000 l df \*ABS\* 00000000 strncpy.c

00000000 l df \*ABS\* 00000000 vfprintf.c

00025284 l O .rodata 00000010 blanks.4489

00025294 l O .rodata 00000010 zeroes.4490

00000000 l df \*ABS\* 00000000 vfprintf.c

0001c798 l F .text 000000f0 \_\_sprint\_r.part.0

0001db88 l F .text 000000c0 \_\_sbprintf

00025410 l O .rodata 00000010 blanks.4480

00025420 l O .rodata 00000010 zeroes.4481

00000000 l df \*ABS\* 00000000 wctomb\_r.c

00000000 l df \*ABS\* 00000000 writer.c

00000000 l df \*ABS\* 00000000 mallocr.c

00000000 l df \*ABS\* 00000000 closer.c

00000000 l df \*ABS\* 00000000 ctype\_.c

00000000 l df \*ABS\* 00000000 fclose.c

00000000 l df \*ABS\* 00000000 fputwc.c

00000000 l df \*ABS\* 00000000 fstatr.c

00000000 l df \*ABS\* 00000000 isattyr.c

00000000 l df \*ABS\* 00000000 lseekr.c

00000000 l df \*ABS\* 00000000 readr.c

00000000 l df \*ABS\* 00000000 reent.c

00000000 l df \*ABS\* 00000000 vfprintf.c

000256a0 l O .rodata 00000010 blanks.4466

000256b0 l O .rodata 00000010 zeroes.4467

00000000 l df \*ABS\* 00000000 wbuf.c

00000000 l df \*ABS\* 00000000 wcrtomb.c

00000000 l df \*ABS\* 00000000 sys\_close.c

00000000 l df \*ABS\* 00000000 sys\_exit.c

00000000 l df \*ABS\* 00000000 sys\_fstat.c

00000000 l df \*ABS\* 00000000 sys\_isatty.c

00000000 l df \*ABS\* 00000000 sys\_lseek.c

00000000 l df \*ABS\* 00000000 sys\_read.c

00000000 l df \*ABS\* 00000000 sys\_sbrk.c

00026a9c l O .sbss 00000004 heap\_end.1862

00000000 l df \*ABS\* 00000000 sys\_write.c

00000000 l df \*ABS\* 00000000 sys\_conv\_stat.c

00000000 l df \*ABS\* 00000000 errno.c

00000000 l df \*ABS\* 00000000 libgcc2.c

00000000 l df \*ABS\* 00000000 libgcc2.c

00000000 l df \*ABS\* 00000000 divdf3.c

00000000 l df \*ABS\* 00000000 muldf3.c

00000000 l df \*ABS\* 00000000 eqtf2.c

00000000 l df \*ABS\* 00000000 getf2.c

00000000 l df \*ABS\* 00000000 letf2.c

00000000 l df \*ABS\* 00000000 multf3.c

00000000 l df \*ABS\* 00000000 subtf3.c

00000000 l df \*ABS\* 00000000 fixtfsi.c

00000000 l df \*ABS\* 00000000 floatsitf.c

00000000 l df \*ABS\* 00000000 extenddftf2.c

00000000 l df \*ABS\* 00000000 trunctfdf2.c

00000000 l df \*ABS\* 00000000 libgcc2.c

00000000 l df \*ABS\* 00000000 libgcc2.c

00000000 l df \*ABS\* 00000000 crtstuff.c

000260b0 l O .eh\_frame 00000000 \_\_FRAME\_END\_\_

00000000 l df \*ABS\* 00000000

000260c0 l .fini\_array 00000000 \_\_fini\_array\_end

000260bc l .fini\_array 00000000 \_\_fini\_array\_start

000260bc l .init\_array 00000000 \_\_init\_array\_end

000260b4 l .init\_array 00000000 \_\_preinit\_array\_end

000260b4 l .init\_array 00000000 \_\_init\_array\_start

000260b4 l .init\_array 00000000 \_\_preinit\_array\_start

00018f74 g F .text 0000009c \_mprec\_log10

00019078 g F .text 00000078 \_\_any\_on

0001e158 g F .text 00000054 \_isatty\_r

000250c8 g O .rodata 00000028 \_\_mprec\_tinytens

00019b84 g F .text 000000b0 strcpy

0001e26c g F .text 00000040 cleanup\_glue

000105d4 g F .text 000000d0 \_puts\_r

0001e1ac g F .text 00000060 \_lseek\_r

00021654 g F .text 00000144 .hidden \_\_getf2

00021588 g F .text 000000cc .hidden \_\_eqtf2

00010580 g F .text 00000054 printf

000268c0 g \*ABS\* 00000000 \_\_global\_pointer$

0001f7f0 g F .text 00000078 \_wcrtomb\_r

00019998 g F .text 00000068 \_\_sseek

00013c38 g F .text 00000010 \_\_sinit

0001f658 g F .text 00000184 \_\_swbuf\_r

00017258 g F .text 0000007c \_setlocale\_r

00013aa8 g F .text 00000078 \_\_sfmoreglue

0001803c g F .text 00000004 \_\_malloc\_unlock

00024338 g F .text 00000188 .hidden \_\_floatsitf

00017f18 g F .text 00000120 memmove

00013c24 g F .text 00000014 \_cleanup

00018040 g F .text 000000a8 \_Balloc

00010144 g F .text 00000048 reverse

0001fc78 g F .text 0000000c \_\_errno

0001fbd4 g F .text 000000a4 \_conv\_stat

00017240 g F .text 00000008 \_\_localeconv\_l

0001e0fc g F .text 0000005c \_fstat\_r

00026ae4 g O .bss 00000004 errno

00019908 g F .text 00000008 \_\_seofread

00026a60 g .sdata 00000000 \_\_SDATA\_BEGIN\_\_

00017dfc g F .text 0000011c memcpy

00013910 g F .text 0000000c \_cleanup\_r

00019d00 g F .text 00002a98 \_svfprintf\_r

000106a4 g F .text 00000010 puts

00018ed0 g F .text 000000a4 \_\_ratio

0001053c g F .text 00000044 \_printf\_r

00021798 g F .text 00000144 .hidden \_\_letf2

00024a28 g F .text 00000048 .hidden \_\_udivsi3

0001f994 g F .text 00000070 \_fstat

00025000 g O .rodata 000000c8 \_\_mprec\_tens

00026a98 g O .sbss 00000004 \_\_malloc\_top\_pad

00026a7c g O .sdata 00000000 .hidden \_\_dso\_handle

00017248 g F .text 00000008 \_localeconv\_r

000184a0 g F .text 00000034 \_\_i2b

000140a0 g F .text 000004bc \_\_sfvwrite\_r

000196c0 g F .text 00000054 \_sbrk\_r

0001e20c g F .text 00000060 \_read\_r

0001de00 g F .text 00000110 \_fclose\_r

000138e0 g F .text 00000028 fflush

00026a94 g O .sbss 00000004 \_\_malloc\_max\_sbrked\_mem

00018c74 g F .text 00000118 \_\_b2d

00020258 g F .text 000004f0 .hidden \_\_umoddi3

0001fa04 g F .text 00000040 \_isatty

00026a78 g O .sdata 00000004 \_global\_impure\_ptr

000190f0 g F .text 000005d0 \_realloc\_r

000103cc g F .text 00000094 \_\_libc\_init\_array

0001fc84 g F .text 000005d4 .hidden \_\_udivdi3

0001e04c g F .text 0000002c \_fputwc\_r

000250f0 g O .rodata 00000028 \_\_mprec\_bigtens

00018264 g F .text 00000110 \_\_s2b

0001fae4 g F .text 000000a0 \_sbrk

000189d4 g F .text 0000005c \_\_mcmp

00013c58 g F .text 00000014 \_\_fp\_lock\_all

0001036c g F .text 00000060 \_\_libc\_fini\_array

0001e2ac g F .text 00000110 \_reclaim\_reent

00018374 g F .text 00000074 \_\_hi0bits

000241e8 g F .text 00000150 .hidden \_\_fixtfsi

00018740 g F .text 00000148 \_\_pow5mult

00024a70 g F .text 00000010 .hidden \_\_umodsi3

00024ad4 g F .text 0000004c .hidden \_\_clzsi2

00013c48 g F .text 00000004 \_\_sfp\_lock\_acquire

00017d20 g F .text 000000dc memchr

000197c4 g F .text 0000006c \_sprintf\_r

00013da8 g F .text 000002f8 \_free\_r

000172d4 g F .text 00000008 \_\_locale\_mb\_cur\_max

00013508 g F .text 0000011c \_\_call\_exitprocs

0001018c g F .text 000000e8 test

00026a84 g O .sdata 00000004 \_\_malloc\_sbrk\_base

00010090 g F .text 00000040 \_start

0001fa44 g F .text 00000050 \_lseek

00018888 g F .text 0000014c \_\_lshift

00021798 g F .text 00000144 .hidden \_\_lttf2

00021588 g F .text 000000cc .hidden \_\_netf2

0001e3bc g F .text 000001bc \_\_ssprint\_r

0001346c g F .text 0000009c \_\_register\_exitproc

000171dc g F .text 00000064 \_ldcheck

000184d4 g F .text 0000026c \_\_multiply

000249fc g F .text 00000024 .hidden \_\_mulsi3

00019c34 g F .text 000000cc strncpy

00026abc g O .bss 00000028 \_\_malloc\_current\_mallinfo

00018d8c g F .text 00000144 \_\_d2b

000244c0 g F .text 00000208 .hidden \_\_extenddftf2

0001ddac g F .text 00000054 \_close\_r

00013304 g F .text 00000168 \_\_swsetup\_r

00020748 g F .text 000007d0 .hidden \_\_divdf3

00013b20 g F .text 00000104 \_\_sfp

00019010 g F .text 00000068 \_\_copybits

00026ae8 g .bss 00000000 \_\_BSS\_END\_\_

00026654 g O .data 00000408 \_\_malloc\_av\_

00013c54 g F .text 00000004 \_\_sinit\_lock\_release

00020f18 g F .text 00000670 .hidden \_\_muldf3

000198ac g F .text 0000005c \_\_sread

00018038 g F .text 00000004 \_\_malloc\_lock

00013880 g F .text 00000060 \_fflush\_r

0001dce4 g F .text 000000c8 \_calloc\_r

00026a8c g .sbss 00000000 \_\_bss\_start

00010460 g F .text 000000dc memset

00010274 g F .text 000000b0 main

00026a90 g O .sbss 00000004 \_\_malloc\_max\_total\_mem

0001f7dc g F .text 00000014 \_\_swbuf

00019a00 g F .text 00000008 \_\_sclose

0001df10 g F .text 00000010 fclose

000174e0 g F .text 000007cc \_malloc\_r

0001dc54 g F .text 00000030 \_\_ascii\_wctomb

0001455c g F .text 000000b0 \_fwalk

00017cac g F .text 0000000c \_mbtowc\_r

00024a20 g F .text 00000084 .hidden \_\_divsi3

00013c80 g F .text 00000128 \_malloc\_trim\_r

00019a08 g F .text 0000017c strcmp

0001db70 g F .text 00000018 vfiprintf

000218dc g F .text 0000136c .hidden \_\_multf3

00019830 g F .text 0000007c sprintf

00025774 g O .rodata 00000100 .hidden \_\_clz\_tab

00026a8c g O .sbss 00000004 \_PathLocale

00010324 g F .text 00000014 atexit

0001dc84 g F .text 00000060 \_write\_r

000172dc g F .text 00000014 setlocale

00026a80 g O .sdata 00000004 \_impure\_ptr

00013624 g F .text 0000025c \_\_sflush\_r

00021654 g F .text 00000144 .hidden \_\_gttf2

0001e578 g F .text 000010e0 \_svfiprintf\_r

00017cb8 g F .text 00000068 \_\_ascii\_mbtowc

00022c48 g F .text 000015a0 .hidden \_\_subtf3

00018c14 g F .text 00000060 \_\_ulp

00013c6c g F .text 00000014 \_\_fp\_unlock\_all

00017250 g F .text 00000008 localeconv

000172f0 g F .text 000000d0 \_\_swhatbuf\_r

000260c0 g .data 00000000 \_\_DATA\_BEGIN\_\_

0001fb84 g F .text 00000050 \_write

00026a8c g .sdata 00000000 \_edata

00026ae8 g .bss 00000000 \_end

0001df20 g F .text 0000012c \_\_fputwc

00019910 g F .text 00000088 \_\_swrite

00026a88 g O .sdata 00000004 \_\_malloc\_trim\_threshold

00010338 g F .text 00000034 exit

0001c8a0 g F .text 000012d0 \_vfiprintf\_r

0001460c g F .text 000000c0 \_fwalk\_reent

00018a30 g F .text 000001e4 \_\_mdiff

00024aa4 g F .text 00000030 .hidden \_\_modsi3

00013c4c g F .text 00000004 \_\_sfp\_lock\_release

00015ee8 g F .text 000012f4 \_ldtoa\_r

00025430 g O .rodata 00000101 \_ctype\_

0001fa94 g F .text 00000050 \_read

0001f950 g F .text 00000044 \_exit

000173c0 g F .text 00000120 \_\_smakebuf\_r

000106b4 g F .text 0000008c strlen

0001c888 g F .text 00000018 \_\_sprint\_r

0001dc48 g F .text 0000000c \_wctomb\_r

00010740 g F .text 00002aec \_vfprintf\_r

000183e8 g F .text 000000b8 \_\_lo0bits

0001f868 g F .text 00000090 wcrtomb

00019714 g F .text 000000b0 frexp

000264e8 g O .data 0000016c \_\_global\_locale

0001322c g F .text 00000018 vfprintf

000246c8 g F .text 00000334 .hidden \_\_trunctfdf2

0001e078 g F .text 00000084 fputwc

0001f8f8 g F .text 00000058 \_close

00013c50 g F .text 00000004 \_\_sinit\_lock\_acquire

0001810c g F .text 00000158 \_\_multadd

000180e8 g F .text 00000024 \_Bfree

Таблица символов содержит множество дополнительных вхождений, однако в целом определяет все нужные секции, метки и адреса. Функции reverse (00010144 g F .text 00000048 reverse) ***и*** main (00010274 g F .text 000000b0 main)

также помечены флагом F, но в отличие от стадии ассемблирования все они являются определенными и содержатся по корректным адресам для успешного вызова этих функций из других участков программ.

Проанализируем таблицу перемещений исполняемого файла:

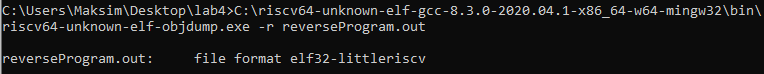
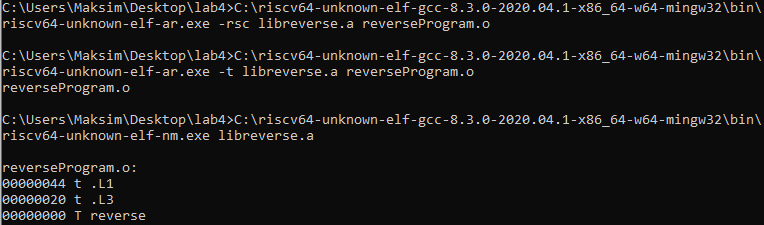


Таблица перемещений оказывается пуста, все необходимые релокации, оптимизации и замены инструкций были успешно проведены компоновщиком.

Итогом сборки программ на языке C по шагам является исполняемый на процессорах архитектуры RISC-V файл, выполняющий реверс массива чисел и проверяющий корректность решения этой задачи на двух тестовых примерах.

# **3.3 Формирование статической библиотеки, разработка make-файлов для сборки библиотеки**

Статическая библиотека 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 реверса массива чисел 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.

Символ функции reverse является основным символом, определяемым в этом объектном файле, остальные символы определяют лишь локальные для этого файла метки.

Поиск библиотеки и сборка программы:

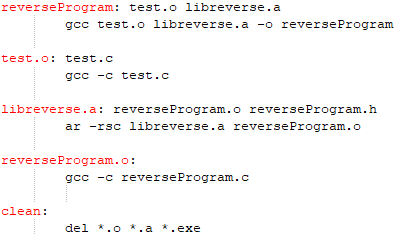


В результате на выходе получим файл reverseProgram.

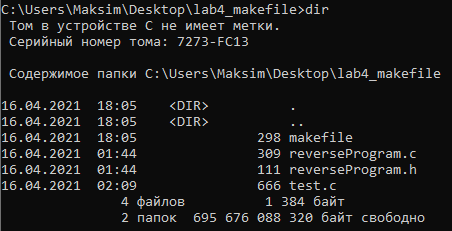
Теперь выполним сборку через *makefile*.

*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).

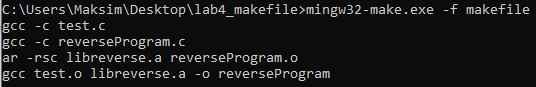
Содержимое *makefile*:



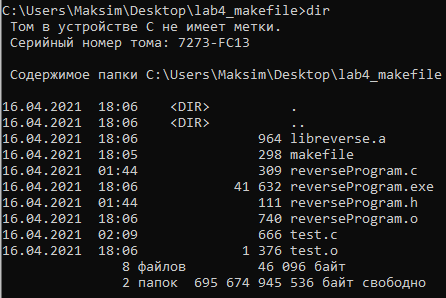
Директория с файлами:



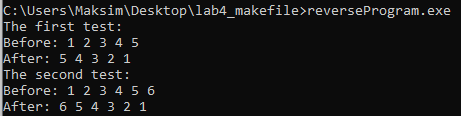
Запуск *makefile* и процесс сборки:



Директория после запуска:



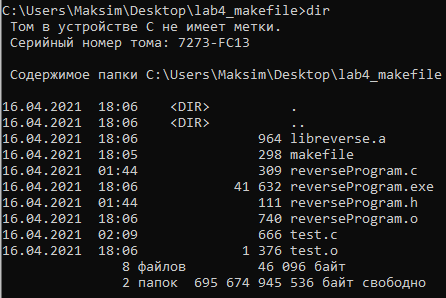
Результат работы *reverseProgram.exe*:



Что происходит в *Makefile*:

1. Создаём объектный файл test.o из исходного test.c
2. Создаём объектный файл reverseProgram.o из исходного reverseProgram.c
3. Архивируем объектный файл reverseProgram.o (создаём статическую библиотеку libreverse.a)
4. Компонуем статическую библиотеку libreverse.a с объектным файлом test.o и получаем исполняемый файл reverseProgram.exe
5. Кроме того, в наш *makefile* мы добавили команду *clean*, которая чистит директорию с файлами от построенных файлов, оставляя только исходные:

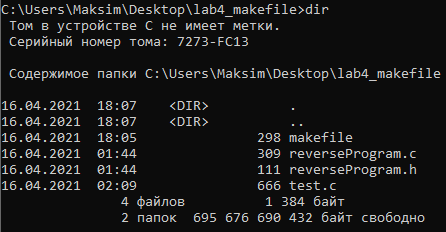
До выполнения:



Вызов clean:



После выполнения:



# **4. Выводы**

В ходе выполнения лабораторной работы 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.

Была реализована поставленная задача – «реверс массива чисел», а затем проведена проверка правильности перевода программы решения этой задачи в набор инструкций, выполняемых процессором.