Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

Отчёт по лабораторной работе № 4

Дисциплина: Низкоуровневое программирование

Тема: раздельная компиляция

Вариант: 12

Выполнил студент гр. 3530901/90002 _____Г. А. Сухоруков
(подпись)

Принял старший преподаватель _____Д. С. Степанов
(подпись)

“_” _____ 2021 г.

Санкт-Петербург
2021

Формулировка задачи

1. На языке С разработать функцию, реализующую определенную вариантом задания функциональность. Поместить определение функции в отдельный исходный файл, оформить заголовочный файл. Разработать тестовую программу на языке С.
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.

Вариант №12 – Формирование последовательности чисел в коде Грея заданной разрядности.

1. Программа, реализующая функциональность задания

Листинг 1.1. Заголовочный файл *grayCode.h*

```
#ifndef GRAYCODE_H_
#define GRAYCODE_H_

extern int grayCode( int n );

#endif // GRAYCODE_H_
```

Листинг 1.2. Основной файл *grayCode.c*

```
#include "grayCode.h"
#include <stddef.h>
#include "string.h"
#include <stdio.h>

int main( int n ) {
    int m = 1;
    for (int i = 0; i < n; i++) {
        m *= 2;
    }
  unsigned grayCode[m][n];
    grayCode[0][n - 1] = 0;
```

```
    grayCode[1][n - 1] = 1;
    int p = 2;
    for (int i = 1; i < n; i++) {
        int t = p - 1;
        int k = p;
        p *= 2;
        for (; k < p; k++) {
            for (int row = 0; row < n; row++) {
                grayCode[k][row] = grayCode[t][row];
            }

            grayCode[t][n - i - 1] = 0;
            grayCode[k][n - i - 1] = 1;
            t--;
        }
    }
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d", grayCode[i][j]);
        }
   printf(" ");
    }
    return 0;
}
```

Листинг 1.3. Тестовая программа *main.c*

```
#include "grayCode.h"
#include <stdio.h>

int main( void ) {
    grayCode(3);
    printf("\n");
    grayCode(2);
    return 0;
}
```

2. Сборка программы «по шагам»

2.1. Препроцессирование

Выполним сборку программы по шагам. Для выполнения отдельных шагов мы будем по-прежнему драйвер компилятора (а не обращаться к ассемблеру или компоновщику напрямую), и контролировать его действия.

Используя пакет разработки “SiFive GNU Embedded Toolchain” для RISC-V , первым шагом выполним препроцессирование файлов. Для этого выполним следующие команды:

```
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -O1 -v -E main.c -o main.i >log_pre_main.txt 2>&1
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -O1 -v -E grayCode.c -o grayCode.i >log_pre_ grayCode.txt 2>&1
```

Разберём параметры запуска.

-march=rv64iac -mabi=lp64	целевым является процессор с базовой архитектурой системы команд RV32I
-O1	выполнять простые оптимизации генерируемого кода
-v	печатать (в стандартный поток ошибок) выполняемые драйвером команды, а также дополнительную информацию.
>	Выводы печати в файлы
-o	Output file
-E	Выполнять обработку файлов только препроцессором
2>&1	поток вывода ошибок (2 – стандартный «номер» этого потока) «связывается» с поток вывода («номер» 1), т.е. сообщения об ошибках (и информация, вывод которой вызван использованием флага “-v”, см.выше) также выводятся в файл

Результат препроцессирования содержится в файле *grayCode.i* и *main.i.* По причине того, что исходные файлы содержат заголовочные файлы нескольких стандартных библиотек С, результат препроцессирования отличается от исходных файлов и имеет достаточно много добавочных строк, среди которых и исходные программы. Также можно заметить, что препроцессор включил содержимое файла *grayCode.h.*

Листинг 2.1. Выход препроцессора (фрагменты)

```
// файл grayCode.i
# 1 "grayCode.c"
```

```
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "grayCode.c"
# 1 "grayCode.h" 1



extern int grayCode( int n );
# 2 "grayCode.c" 2
# 1 "c:\\users\\gleb\\desktop\\riscv64-unknown-elf-toolchain-10.2.0-2020.12.8-x86_64-w64-
mingw32\\riscv64-unknown-elf-toolchain-10.2.0-2020.12.8-x86_64-w64-
mingw32\\lib\\gcc\\riscv64-unknown-elf\\10.2.0\\include\\stddef.h" 1 3 4
# 143 "c:\\users\\gleb\\desktop\\riscv64-unknown-elf-toolchain-10.2.0-2020.12.8-x86_64-
w64-mingw32\\riscv64-unknown-elf-toolchain-10.2.0-2020.12.8-x86_64-w64-
mingw32\\lib\\gcc\\riscv64-unknown-elf\\10.2.0\\include\\stddef.h" 3 4

# 143 "c:\\users\\gleb\\desktop\\riscv64-unknown-elf-toolchain-10.2.0-2020.12.8-x86_64-
w64-mingw32\\riscv64-unknown-elf-toolchain-10.2.0-2020.12.8-x86_64-w64-
mingw32\\lib\\gcc\\riscv64-unknown-elf\\10.2.0\\include\\stddef.h" 3 4
typedef long int ptrdiff_t;
# 209 "c:\\users\\gleb\\desktop\\riscv64-unknown-elf-toolchain-10.2.0-2020.12.8-x86_64-
w64-mingw32\\riscv64-unknown-elf-toolchain-10.2.0-2020.12.8-x86_64-w64-
mingw32\\lib\\gcc\\riscv64-unknown-elf\\10.2.0\\include\\stddef.h" 3 4
typedef long unsigned int size_t;
# 321 "c:\\users\\gleb\\desktop\\riscv64-unknown-elf-toolchain-10.2.0-2020.12.8-x86_64-
w64-mingw32\\riscv64-unknown-elf-toolchain-10.2.0-2020.12.8-x86_64-w64-
mingw32\\lib\\gcc\\riscv64-unknown-elf\\10.2.0\\include\\stddef.h" 3 4
typedef int wchar_t;
# 415 "c:\\users\\gleb\\desktop\\riscv64-unknown-elf-toolchain-10.2.0-2020.12.8-x86_64-
w64-mingw32\\riscv64-unknown-elf-toolchain-10.2.0-2020.12.8-x86_64-w64-
mingw32\\lib\\gcc\\riscv64-unknown-elf\\10.2.0\\include\\stddef.h" 3 4
typedef struct {
  long long __max_align_ll __attribute__((__aligned__(__alignof__(long long))));
  long double __max_align_ld __attribute__((__aligned__(__alignof__(long double))));
# 426 "c:\\users\\gleb\\desktop\\riscv64-unknown-elf-toolchain-10.2.0-2020.12.8-x86_64-
w64-mingw32\\riscv64-unknown-elf-toolchain-10.2.0-2020.12.8-x86_64-w64-
mingw32\\lib\\gcc\\riscv64-unknown-elf\\10.2.0\\include\\stddef.h" 3 4
} max_align_t;
# 3 "grayCode.c" 2


# 4 "grayCode.c"
int main( int n ) {
    int m = 1;
    for (int i = 0; i < n; i++) {
        m *= 2;
    }

    unsigned grayCode[m][n];
    grayCode[0][n - 1] = 0;
    grayCode[1][n - 1] = 1;
    int p = 2;
    for (int i = 1; i < n; i++) {
        int t = p - 1;
        int k = p;
        p *= 2;
        for (; k < p; k++) {
            for (int row = 0; row < n; row++) {
                grayCode[k][row] = grayCode[t][row];
            }

            grayCode[t][n - i - 1] = 0;
            grayCode[k][n - i - 1] = 1;
            t--;
        }
    }
    return 0;
}
```

```
// файл main.i
# 1 "main.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "main.c"
# 1 "grayCode.h" 1



extern int grayCode( int n );
# 2 "main.c" 2

int main( void ) {
    grayCode(3);
    grayCode(2);

    return 0;
}
```

2.2. Компиляция

Для компиляции препроцессированных файлов используем следующие команды:

```
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -O1 -v -S -fpreprocessed main.i -o main.s >log_comp_main.txt 2>&1

riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -O1 -v -S -fpreprocessed grayCode.i -o grayCode.s >log_comp_grayCode.txt 2>&1
```

Наибольший интерес представляет файл *main.s*, так как в нем можно заметить обращение к подпрограмме *grayCode* (значение регистра *ra*, содержащее адрес возврата из *main*, сохраняется на время вызова в стеке). Следует отметить, что символ *grayCode* используется в файле, но никак не определяется.

Листинг 2.2. Выход компилятора *main.s*

```
        .file   "main.c"
        .option nopic
        .attribute arch, "rv64i2p0_a2p0_c2p0"
        .attribute unaligned_access, 0
        .attribute stack_align, 16
        .text
        .align 1
        .globl main
        .type   main, @function
main:
        addi    sp,sp,-16
        sd      ra,8(sp)
        li      a0,3
        call    grayCode
        li      a0,2
        call    grayCode
        li      a0,0
        ld      ra,8(sp)
        addi    sp,sp,16
        jr      ra
        .size   main, .-main
        .ident "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"
```

Листинг 2.3. Выход компилятора *grayCode.s*

```
        .file   "grayCode.c"
        .option nopic
        .attribute arch, "rv64i2p0_a2p0_c2p0"
        .attribute unaligned_access, 0
        .attribute stack_align, 16
        .text
        .align 1
        .globl main
        .type   main, @function
main:
        mv      a4,a0
        ble     a0,zero,.L2
        li      a5,0
.L3:
        addiw   a5,a5,1
        bne     a4,a5,.L3
.L2:
        li      a5,1
        ble     a4,a5,.L4
        li      a2,2
        li      a0,1
        li      a1,0
.L8:
        mv      a3,a2
        slliw   a2,a2,1
        ble     a2,a3,.L6
.L5:
        mv      a5,a1
.L7:
        addiw   a5,a5,1
        bne     a4,a5,.L7
        addiw   a3,a3,1
        bne     a2,a3,.L5
.L6:
        addiw   a0,a0,1
        bne     a4,a0,.L8
.L4:
        li      a0,0
        ret
        .size   main, .-main
        .ident "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"
```

2.3. Выход ассемблера – объектный файл

Выполним ассемблирование для получения объектных файлов программы.

```
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -v -c main.s -o main.o
>log_as_main.txt 2>&1
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -v -c grayCode.s -o
grayCode.o >log_as_grayCode.txt 2>&1
```

На выходе получаем файлы “reverse.o” и “main.o”. В отличие от ранее рассмотренных файлов, объектные файлы не является текстовыми, для изучения их содержимого используем утилиту objdump, отображающую содержимое бинарных файлов в текстовом виде:

Листинг 2.4. Хедер файла *main.o*

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

```
main.o:     file format elf64-littleriscv

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .text         00000020  0000000000000000  0000000000000000  00000040  2**1
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  0000000000000000  0000000000000000  00000060  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  0000000000000000  0000000000000000  00000060  2**0
                  ALLOC
  3 .comment      00000031  0000000000000000  0000000000000000  00000060  2**0
                  CONTENTS, READONLY
  4 .riscv.attributes 00000026  0000000000000000  0000000000000000  00000091  2**0
                  CONTENTS, READONLY
```

Листинг 2.5. Хедер файла *grayCode.o*

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

```
grayCode.o:     file format elf64-littleriscv

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .text         0000003c  0000000000000000  0000000000000000  00000040  2**1
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  0000000000000000  0000000000000000  0000007c  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  0000000000000000  0000000000000000  0000007c  2**0
                  ALLOC
  3 .comment      00000031  0000000000000000  0000000000000000  0000007c  2**0
                  CONTENTS, READONLY
  4 .riscv.attributes 00000026  0000000000000000  0000000000000000  000000ad  2**0
                  CONTENTS, READONLY
```

Вся информация размещается в секциях.

Секция	Назначение
.text	секция кода, в которой содержатся коды инструкций
.data	секция инициализированных данных
.bss	секция данных, инициализированных нулями
.comment	секция данных о версиях размером 12 байт

Также в начале вывода пишут о формате файла “elf” и о том, что используется архитектура little-endian RISC-V.

Рассмотрим некоторые секции поближе.

Листинг 2.6. Дизассемблированный файл *main.o*

riscv64-unknown-elf-objdump -d -M no-aliases -j .text main.o

```
main.o:     file format elf64-littleriscv


Disassembly of section .text:

0000000000000000 <main>:
   0:   1141                    c.addi  sp,-16
   2:   e406                    c.sdsp  ra,8(sp)
   4:   450d                    c.li    a0,3
   6:   00000097                auipc   ra,0x0
   a:   000080e7                jalr    ra,0(ra) # 6 <main+0x6>
   e:   4509                    c.li    a0,2
  10:   00000097                auipc   ra,0x0
  14:   000080e7                jalr    ra,0(ra) # 10 <main+0x10>
  18:   4501                    c.li    a0,0
  1a:   60a2                    c.ldsp  ra,8(sp)
  1c:   0141                    c.addi  sp,16
  1e:   8082                    c.jr    ra
```

Можно заметить псеводинстуркцию call, которая здесь записана комбинацей инструкций auipc + jalr. Также наблюдается выход из метода main.

Листинг 2.7. Содержание секции .comment

riscv64-unknown-elf-objdump –j -s .comment main.o

```
main.o:     file format elf64-littleriscv

Contents of section .text:
 0000 411106e4 0d459700 0000e780 00000945  A....E.........E
 0010 97000000 e7800000 0145a260 41018280  .........E.`A...
Contents of section .comment:
 0000 00474343 3a202853 69466976 65204743  .GCC: (SiFive GC
 0010 432d4d65 74616c20 31302e32 2e302d32  C-Metal 10.2.0-2
 0020 3032302e 31322e38 29203130 2e322e30  020.12.8) 10.2.0
 0030 00                                   .
Contents of section .riscv.attributes:
 0000 41250000 00726973 63760001 1b000000  A%...riscv......
 0010 04100572 76363469 3270305f 61327030  ...rv64i2p0_a2p0
 0020 5f633270 3000                        _c2p0.
```

Тут ничего особенного не наблюдается, всё как в main.s.

Рассмотрим таблицу символов:

Листинг 2.8. Таблица символов

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

```
grayCode.o:     file format elf64-littleriscv

SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 grayCode.c
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000000 l    d  .data  0000000000000000 .data
0000000000000000 l    d  .bss   0000000000000000 .bss
000000000000000e l       .text  0000000000000000 .L2
0000000000000008 l       .text  0000000000000000 .L3
0000000000000038 l       .text  0000000000000000 .L4
0000000000000032 l       .text  0000000000000000 .L6
0000000000000026 l       .text  0000000000000000 .L7
0000000000000024 l       .text  0000000000000000 .L5
000000000000001a l       .text  0000000000000000 .L8
0000000000000000 l    d  .comment       0000000000000000 .comment
0000000000000000 l    d  .riscv.attributes      0000000000000000 .riscv.attributes
0000000000000000 g     F .text  000000000000003c main



main.o:     file format elf64-littleriscv

SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 main.c
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000000 l    d  .data  0000000000000000 .data
0000000000000000 l    d  .bss   0000000000000000 .bss
0000000000000000 l    d  .comment       0000000000000000 .comment
0000000000000000 l    d  .riscv.attributes      0000000000000000 .riscv.attributes
0000000000000000 g     F .text  0000000000000020 main
0000000000000000         *UND*  0000000000000000 grayCode
```

В таблице символов *main.o* имеется запись: символ “grayCode” типа *UND*. Эта запись означает, что символ “grayCode” использовался в ассемблерном коде, из которого был получен данный объектный файл, но не был определен; ассемблер сделал вывод о том, что символ должен быть определен где-то еще, и отразил это в таблице символов.

Информация обо всех «неоконченных» инструкциях передается ассемблером компоновщику посредством **таблицы перемещений**:

Листинг 2.9. Таблица перемещений

riscv64-unknown-elf-objdump -r grayCode.o main.o

```
grayCode.o:     file format elf64-littleriscv

RELOCATION RECORDS FOR [.text]:
OFFSET           TYPE              VALUE
0000000000000002 R_RISCV_BRANCH    .L2
000000000000000a R_RISCV_BRANCH    .L3
0000000000000010 R_RISCV_BRANCH    .L4
0000000000000020 R_RISCV_BRANCH    .L6
0000000000000028 R_RISCV_BRANCH    .L7
000000000000002e R_RISCV_BRANCH    .L5
0000000000000034 R_RISCV_BRANCH    .L8



main.o:     file format elf64-littleriscv

RELOCATION RECORDS FOR [.text]:
OFFSET           TYPE              VALUE
0000000000000006 R_RISCV_CALL      grayCode
0000000000000006 R_RISCV_RELAX     *ABS*
0000000000000010 R_RISCV_CALL      grayCode
0000000000000010 R_RISCV_RELAX     *ABS*
```

В таблице перемещений для main.o наблюдаем вызов метода grayCode. Записи типа “R_RISCV_RELAX” заносятся в таблицу перемещений в дополнение к записям типа “R_RISCV_CALL” и сообщают компоновщику, что пара инструкций, обеспечивающих вызов подпрограммы, может быть оптимизирована.

2.4. Компоновка

Рассмотрим результат такой оптимизации в нашей программе.

Листинг 2.10. Результат компоновки *a.out*

```
00010188 <main>:
   10188:       ff010113        addi    sp,sp,-16
   1018c:       00112623        sw      ra,12(sp)
   10190:       00300513        addi    a0,zero,3
   10194:       024000ef        jal     ra,101b8 <grayCode>
   10198:       00a00513        addi    a0,zero,10
   1019c:       538000ef        jal     ra,106d4 <putchar>
   101a0:       00200513        addi    a0,zero,2
   101a4:       014000ef        jal     ra,101b8 <grayCode>
   101a8:       00000513        addi    a0,zero,0
   101ac:       00c12083        lw      ra,12(sp)
   101b0:       01010113        addi    sp,sp,16
   101b4:       00008067        jalr    zero,0(ra) # 10174
<frame_dummy+0x20>

000101b8 <grayCode>:
   101b8:       fb010113        addi    sp,sp,-80
```

```
101bc:      04112623      sw      ra,76(sp)
                                  ...
103e8:      05010113      addi    sp,sp,80
103ec:      00008067      jalr    zero,0(ra)
```

Можно видеть, что подпрограмма main имеет на одну инструкцию меньше: пара инструкций auipc+jalr заменена компоновщиком после оптимизации одной инструкцией jal.

3. Создание статической библиотеки

Выделим из программы *grayCode.c* функцию возведения двойки в степень *pow2(n)* в отдельную программу и объединим эти программы в статическую библиотеку gray, тестовую программу main оставим без изменений.

Листинг 3.1. Заголовочный файл *grayCode.h*

```
#ifndef GRAYCODE_H_
#define GRAYCODE_H_

extern int grayCode( int n );

#endif // GRAYCODE_H_
```

Листинг 3.2. Основной файл *grayCode.c*

```
#include "grayCode.h"
#include <stddef.h>
#include "string.h"
#include <stdio.h>
#include "pow2.h"

int grayCode( int n ) {
    int m = pow2(n);

    unsigned grayCode[m][n];
    grayCode[0][n - 1] = 0;
    grayCode[1][n - 1] = 1;

    int p = 2;
    for (int i = 1; i < n; i++) {
        int t = p - 1;
        int k = p;
        p *= 2;

        for (; k < p; k++) {
            for (int row = 0; row < n; row++) {
```

```
                grayCode[k][row] = grayCode[t][row];
            }

            grayCode[t][n - i - 1] = 0;
            grayCode[k][n - i - 1] = 1;
            t--;
        }
    }


    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d", grayCode[i][j]);
        }
        printf(" ");
    }

    return 0;
}
```

Листинг 3.3. Заголовочный файл *pow2.h*

```
#ifndef POW2_H_
#define POW2_H_

extern int pow2( int pow );

#endif // POW2_H_
```

Листинг 3.4. Основной файл *pow2.c*

```
#include "pow2.h"
#include <stdio.h>

int pow2 ( int pow ) {
    int result = 1;
    for (int i = 0; i < pow; i++) {
        result *= 2;
    }

    return result;
}
```

Для создания статической библиотеки получим объектные файлы всех используемых программ:

```
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -O1 -c grayCode.c -o grayCode.o
```

```
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -O1 -c pow2.c -o pow2.o
```

Объединим их в одну библиотеку gray:

```
riscv64-unknown-elf-ar –rsc libgray.a pow2.o grayCode.o
```

Используем получившуюся библиотеку для сборки программ:

```
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -O1 --save-temps main.c libgrayCode.a -o main
```

Листинг 3.5. Таблица символов исполняемого файла

```
main:     file format elf32-littleriscv

SYMBOL TABLE:
00010074 l    d  .text      00000000 .text
                                    ...
000190f0 g     F .text      00000cac _vfiprintf_r
00013414 g     F .text      000000b4 _fwalk_reent
000165fc g     F .text      00000170 __mdiff
00021408 g     F .text      00000030 .hidden __modsi3
00012fdc g     F .text      00000004 __sfp_lock_release
00014818 g     F .text      00000bf8 _ldtoa_r
000101b8 g     F .text      0000020c grayCode
00021eac g     O .rodata    00000101 _ctype_
0001c408 g     F .text      0000004c _read
0001c2d4 g     F .text      00000044 _exit
00015550 g     F .text      000000f0 __smakebuf_r
00016fc8 g     F .text      0000001c strlen
0001a8fc g     F .text      00000008 __locale_ctype_ptr_l
00018ff4 g     F .text      000000fc __sprint_r
000103c4 g     F .text      00000028 pow2
                                    ...
```

Легко заметить, что в состав программы main вошло содержимое объектных файлов *grayCode.o* и *pow2.o*.

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

Листинг 3.5. make-файл для создания библиотеки

```
CC=riscv64-unknown-elf-gcc
AR=riscv64-unknown-elf-ar
CFLAGS=-march=rv32i -mabi=ilp32 -O1

all: libgray

libgray: pow2.o grayCode.o
	$(AR) -rsc libgray.a pow2.o grayCode.o
pow2.o: pow2.c
	$(CC) $(CFLAGS) -c pow2.c -o pow2.o

grayCode.o: grayCode.c
	$(CC) $(CFLAGS) -c grayCode.c -o grayCode.o

clean:
	rm -f *.o *.a
```

Листинг 3.6. make-файл для сборки

```
TARGET=main
CC=riscv64-unknown-elf-gcc
CFLAGS=-march=rv32i -mabi=ilp32 -O1

main:
	$(CC) $(CFLAGS) main.c libgray.a -o $(TARGET)

clean:
	rm -f *.o *.a $(TARGET)
```

Вывод

В ходе выполнения лабораторной работы была написана программа на языке C с заданной функциональностью. После была выполнена сборка этой программы по шагам для архитектуры команд RISC-V с помощью пакета разработки “SiFive GNU Embedded Toolchain” для RISC-V. Были проанализированы выводы препроцессора, компилятора и линковщика последовательно отдельно друг от друга. Была создана своя статически линкуемая библиотека

libgrayCode.a. Были написаны make-файлы для её сборки, а также для сборки тестовой программы с использованием библиотеки.