Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

# Отчет по лабораторной работе №4

по дисциплине «Низкоуровневое программирование»

**Раздельная компиляция**

**Работу выполнил:**
Ильин В.П.
Группа:
35300901/10005
**Преподаватель:**
Коренев Д.А.

Санкт-Петербург
2022

# Содержание

# 1. Техническое задание

На языке C разработать функцию, реализующую задачу поиска $k$-й порядковой статистики массива. Поместить определение функции в отдельный исходный файл, оформить заголовочный файл. Разработать тестовую программу на языке C.

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.

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. Разработанные исходные тексты

```c
// find.c
#include "find.h"

unsigned find_order_statistic( unsigned *array, size_t array_length, unsigned k ) {
    for (size_t j = 0; j < array_length; j++) {
        for (size_t i = 1; i < array_length; i++) {
            if (array[i] < array[i - 1]) {
                const unsigned t = array[i - 1];
                array[i - 1] = array[i];
                array[i] = t;
            }
        }
    }

    return array[k - 1];
}
```

Листинг 1: Определение функции

```c
// main.c
#include <stdio.h>
#include "find.h"

void test( unsigned *array, size_t array_length, unsigned k ) {
    printf("%d-th order statistic of { ", k);
    for (size_t i = 0; i < array_length; i++) {
        printf("%d ", array[i]);
    }
    printf("} is %d\n", find_order_statistic(array, array_length, k));
}

int main( void ) {
    unsigned array[] = {4, 3, 5, 7, 2, 8};
    unsigned array_s[] = {11, 7, -4, 0};
    const unsigned array_length = sizeof(array) / sizeof(array[0]);
    const unsigned array_length_s = sizeof(array_s) / sizeof(array_s[0]);

```

```
19      test(array, array_length, /*k*/3);
20      test(array_s, array_length_s, /*k*/1);
21      return 0;
22  }
```

Листинг 2: Тестовая программа

```
1  // find.h
2  #ifndef __FIND_H_
3  #define __FIND_H_
4  #include <stddef.h>
5  unsigned find_order_statistic( unsigned *array, size_t array_length, unsigned k );
6  void test( unsigned *array, size_t array_length, unsigned k );
7  #endif /*__FIND_H_*/
```

Листинг 3: Заголовочный файл

# 3. Сборка программы по шагам

## 3.1. Препроцессирование

Первым шагом является препроцессирование файлов исходного кода `find.c` и `main.c` в файлы `find.i` и `main.i` соответственно. Для этого выполним следующие команды:

```
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -O1 -E find.c -o find.i
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -O1 -E main.c -o main.i
```

Драйвер компилятора запускается с командами `-march=rv32i -mabi=ilp32`, указывающими, что целевым является процессор с базовой архитектурой системы команд `RV32I`. Опция `-O1` указывает выполнять базовые оптимизации, а `-E` — остановить процесс сборки после препроцессирования. Ниже представлены фрагменты полученных файлов. В полной версии файла `main.i` перечислены все определения из библиотеки `<stdio.h>`.

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

# 1 "find.h" 1



# 1 "r:\\gcc\\lib\\gcc\\riscv64-unknown-elf\\8.3.0\\include\\stddef.h" 1 3 4
# 149 "r:\\gcc\\lib\\gcc\\riscv64-unknown-elf\\8.3.0\\include\\stddef.h" 3 4

# 149 "r:\\gcc\\lib\\gcc\\riscv64-unknown-elf\\8.3.0\\include\\stddef.h" 3 4
typedef int ptrdiff_t;
# 216 "r:\\gcc\\lib\\gcc\\riscv64-unknown-elf\\8.3.0\\include\\stddef.h" 3 4
typedef unsigned int size_t;
# 328 "r:\\gcc\\lib\\gcc\\riscv64-unknown-elf\\8.3.0\\include\\stddef.h" 3 4
typedef int wchar_t;
# 426 "r:\\gcc\\lib\\gcc\\riscv64-unknown-elf\\8.3.0\\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))));
# 437 "r:\\gcc\\lib\\gcc\\riscv64-unknown-elf\\8.3.0\\include\\stddef.h" 3 4
} max_align_t;
# 5 "find.h" 2

# 5 "find.h"
unsigned find_order_statistic( unsigned *array, size_t array_length, unsigned k );
void test( unsigned *array, size_t array_length, unsigned k );
# 3 "find.c" 2

unsigned find_order_statistic( unsigned *array, size_t array_length, unsigned k ) {
 for (size_t j = 0; j < array_length; j++) {
  for (size_t i = 1; i < array_length; i++) {
   if (array[i] < array[i - 1]) {
    const unsigned t = array[i - 1];
    array[i - 1] = array[i];
    array[i] = t;
   }
  }
 }

 return array[k - 1];
}
```

Листинг 4: find.i

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

# 1 "r:\\gcc\\riscv64-unknown-elf\\include\\stdio.h" 1 3
```

```
# 29 "r:\\gcc\\riscv64-unknown-elf\\include\\stdio.h" 3
# 1 "r:\\gcc\\riscv64-unknown-elf\\include\\_ansi.h" 1 3
# 10 "r:\\gcc\\riscv64-unknown-elf\\include\\_ansi.h" 3
# 1 "r:\\gcc\\riscv64-unknown-elf\\include\\newlib.h" 1 3
# 14 "r:\\gcc\\riscv64-unknown-elf\\include\\newlib.h" 3
# 1 "r:\\gcc\\riscv64-unknown-elf\\include\\_newlib_version.h" 1 3
# 15 "r:\\gcc\\riscv64-unknown-elf\\include\\newlib.h" 2 3
# 11 "r:\\gcc\\riscv64-unknown-elf\\include\\_ansi.h" 2 3
# 1 "r:\\gcc\\riscv64-unknown-elf\\include\\sys\\config.h" 1 3



# 1 "r:\\gcc\\riscv64-unknown-elf\\include\\machine\\ieeefp.h" 1 3
# 5 "r:\\gcc\\riscv64-unknown-elf\\include\\sys\\config.h" 2 3
# 1 "r:\\gcc\\riscv64-unknown-elf\\include\\sys\\features.h" 1 3
# 6 "r:\\gcc\\riscv64-unknown-elf\\include\\sys\\config.h" 2 3
# 12 "r:\\gcc\\riscv64-unknown-elf\\include\\_ansi.h" 2 3
# 30 "r:\\gcc\\riscv64-unknown-elf\\include\\stdio.h" 2 3




# 1 "r:\\gcc\\riscv64-unknown-elf\\include\\sys\\cdefs.h" 1 3
# 45 "r:\\gcc\\riscv64-unknown-elf\\include\\sys\\cdefs.h" 3
# 1 "r:\\gcc\\riscv64-unknown-elf\\include\\machine\\_default_types.h" 1 3
# 41 "r:\\gcc\\riscv64-unknown-elf\\include\\machine\\_default_types.h" 3


...


# 797 "r:\\gcc\\riscv64-unknown-elf\\include\\stdio.h" 3

# 3 "main.c" 2
# 1 "find.h" 1



# 1 "r:\\gcc\\lib\\gcc\\riscv64-unknown-elf\\8.3.0\\include\\stddef.h" 1 3 4
# 5 "find.h" 2

# 5 "find.h"
unsigned find_order_statistic( unsigned *array, size_t array_length, unsigned k );
void test( unsigned *array, size_t array_length, unsigned k );
# 4 "main.c" 2

void test( unsigned *array, size_t array_length, unsigned k ) {
 printf("%d-th order statistic of { ", k);
 for (size_t i = 0; i < array_length; i++) {
  printf("%d ", array[i]);
 }
 printf("} is %d\n", find_order_statistic(array, array_length, k));
```

```
60 }
61
62 int main( void ) {
63  unsigned array[] = {4, 3, 5, 7, 2, 8};
64  unsigned array_s[] = {11, 7, -4, 0};
65  const unsigned array_length = sizeof(array) / sizeof(array[0]);
66  const unsigned array_length_s = sizeof(array_s) / sizeof(array_s[0]);
67
68     test(array, array_length, 3);
69     test(array_s, array_length_s, 1);
70     return 0;
71 }
```

Листинг 5: main.i (фрагмент)

## 3.2. Компиляция

Далее необходимо выполнить компиляцию получившихся препроцессированных файлов, сохранив результат - сгенерированный код на языке ассемблера. Для этого воспользуемся командами:

```
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -O1 -S -fpreprocessed find.i
↪  -o find.s
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -O1 -S -fpreprocessed main.i
↪  -o main.s
```

Для останова процесса сборки после компиляции используется опция `-S`. Флаг `-fpreprocessed` (для *драйвера* компилятора его указывать необязательно) говорит драйверу компилятора о том, что переданный файл уже был препроцессирован, и повторять те же действия не стоит.

```
    .file   "find.c"
    .option nopic
    .attribute arch, "rv32i2p0"
    .attribute unaligned_access, 0
    .attribute stack_align, 16
    .text
    .align  2
    .globl  find_order_statistic
    .type   find_order_statistic, @function
find_order_statistic:
    beq a1,zero,.L2
    slli    a6,a1,2
    add a6,a0,a6
    li  a7,0
    li  t1,1
    j   .L3
.L4:
    addi    a5,a5,4
    beq a5,a6,.L7
.L5:
    lw  a3,0(a5)
    lw  a4,-4(a5)
    bgeu    a3,a4,.L4
```

```
    sw  a3,-4(a5)
    sw  a4,0(a5)
    j   .L4
.L7:
    addi    a7,a7,1
    beq a1,a7,.L2
.L3:
    addi    a5,a0,4
    bgtu    a1,t1,.L5
    j   .L7
.L2:
    slli    a2,a2,2
    add a0,a0,a2
    lw  a0,-4(a0)
    ret
    .size   find_order_statistic, .-find_order_statistic
    .ident  "GCC: (SiFive GCC 8.3.0-2020.04.1) 8.3.0"
```

Листинг 6: find.s

```
    .file   "main.c"
    .option nopic
    .attribute arch, "rv32i2p0"
    .attribute unaligned_access, 0
    .attribute stack_align, 16
    .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)
    mv  s4,a0
    mv  s3,a1
    mv  s5,a2
    mv  a1,a2
    lui a0,%hi(.LC2)
    addi    a0,a0,%lo(.LC2)
    call    printf
    beq s3,zero,.L2
    mv  s0,s4
    slli    s1,s3,2
    add s1,s1,s4
    lui s2,%hi(.LC3)
```

```
.L3:
    lw  a1,0(s0)
    addi    a0,s2,%lo(.LC3)
    call    printf
    addi    s0,s0,4
    bne s0,s1,.L3
.L2:
    mv  a2,s5
    mv  a1,s3
    mv  a0,s4
    call    find_order_statistic
    mv  a1,a0
    lui a0,%hi(.LC4)
    addi    a0,a0,%lo(.LC4)
    call    printf
    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   test, .-test
    .align  2
    .globl  main
    .type   main, @function
main:
    addi    sp,sp,-64
    sw  ra,60(sp)
    lui a5,%hi(.LANCHOR0)
    addi    a5,a5,%lo(.LANCHOR0)
    lw  a6,0(a5)
    lw  a0,4(a5)
    lw  a1,8(a5)
    lw  a2,12(a5)
    lw  a3,16(a5)
    lw  a4,20(a5)
    sw  a6,24(sp)
    sw  a0,28(sp)
    sw  a1,32(sp)
    sw  a2,36(sp)
    sw  a3,40(sp)
    sw  a4,44(sp)
    lw  a2,24(a5)
    lw  a3,28(a5)
    lw  a4,32(a5)
    lw  a5,36(a5)
    sw  a2,8(sp)
    sw  a3,12(sp)
    sw  a4,16(sp)
    sw  a5,20(sp)
```

```
    li  a2,3
    li  a1,6
    addi    a0,sp,24
    call    test
    li  a2,1
    li  a1,4
    addi    a0,sp,8
    call    test
    li  a0,0
    lw  ra,60(sp)
    addi    sp,sp,64
    jr  ra
    .size   main, .-main
    .section    .rodata
    .align  2
    .set    .LANCHOR0,. + 0
.LC0:
    .word   4
    .word   3
    .word   5
    .word   7
    .word   2
    .word   8
.LC1:
    .word   11
    .word   7
    .word   -4
    .word   0
    .section    .rodata.str1.4,"aMS",@progbits,1
    .align  2
.LC2:
    .string "%d-th order statistic of { "
.LC3:
    .string "%d "
.LC4:
    .string "} is %d\n"
    .ident  "GCC: (SiFive GCC 8.3.0-2020.04.1) 8.3.0"
```

Листинг 7: main.s

Прокомментируем содержимое полученных файлов. В `find.s` определяется подпрограмма `find_order_statistic`, выполняющая функциональность задачи. В файле `main.s` определяются две подпрограммы: `test` и `main`. Вместе они обеспечивают тестирование функциональной подпрограммы. В `main` создаются два тестовых набора данных и для каждого из них вызывается подпрограмма `test`. В ней несколько раз вызывается библиотечная подпрограмма `printf` для вывода текста на экран, а также тестируемая `find_order_statistic`

## 3.3. Ассемблирование

Для ассемблирования выполним следующие команды:

```
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -c find.s -o find.o
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -c main.s -o main.o
```

Опция `-c` останавливает процесс сборки после ассемблирования. Стоит отметить, что в этих командах опция `-O1` уже не используется, так как ассемблер (обычно) не выполняет оптимизацию.

### 3.3.1. Анализ результата

Ненадолго прервем сборку и изучим полученные результаты. Для начала, рассмотрим содержимое таблиц символов полученных объектных файлов. Для этого воспользуемся другой утилитой:

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

SYMBOL TABLE:
00000000 l    df *ABS*  00000000 find.c
00000000 l    d  .text  00000000 .text
00000000 l    d  .data  00000000 .data
00000000 l    d  .bss   00000000 .bss
0000004c l       .text  00000000 .L2
00000040 l       .text  00000000 .L3
00000038 l       .text  00000000 .L7
00000018 l       .text  00000000 .L4
00000020 l       .text  00000000 .L5
00000000 l    d  .comment       00000000 .comment
00000000 l    d  .riscv.attributes      00000000 .riscv.attributes
00000000 g     F .text  0000005c find_order_statistic



main.o:     file format elf32-littleriscv

SYMBOL TABLE:
00000000 l    df *ABS*  00000000 main.c
00000000 l    d  .text  00000000 .text
00000000 l    d  .data  00000000 .data
00000000 l    d  .bss   00000000 .bss
00000000 l    d  .rodata        00000000 .rodata
00000000 l       .rodata        00000000 .LANCHOR0
00000000 l    d  .rodata.str1.4 00000000 .rodata.str1.4
00000000 l       .rodata.str1.4 00000000 .LC2
0000001c l       .rodata.str1.4 00000000 .LC3
00000020 l       .rodata.str1.4 00000000 .LC4
0000006c l       .text  00000000 .L2
00000054 l       .text  00000000 .L3
00000000 l    d  .comment       00000000 .comment
00000000 l    d  .riscv.attributes      00000000 .riscv.attributes
00000000 g     F .text  000000b8 test
00000000         *UND*  00000000 printf
00000000         *UND*  00000000 find_order_statistic
000000b8 g     F .text  00000098 main
```

Из таблицы символов видно, что символы `printf` и `find_order_statistic` имеют тип `*UND*`. Это означает, что они использовались в ассемблерном коде, из которого был полу-

чен данный объектный файл, но не были определены; ассемблер сделал вывод о том, что символы должны быть определены где-то еще, и таким образом это отразил.

Теперь проанализируем содержимое секций. Используем ту же утилиту с опциями `-d`, позволяющей дизассемблировать содержимое файла и `-M no-aliases`, требующей использовать в выводе только инструкции системы команд (но не псевдоинструкции):

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

find.o:     file format elf32-littleriscv


Disassembly of section .text:

00000000 <find_order_statistic>:
   0:   04058663                beq     a1,zero,4c <.L2>
   4:   00259813                slli    a6,a1,0x2
   8:   01050833                add     a6,a0,a6
   c:   00000893                addi    a7,zero,0
  10:   00100313                addi    t1,zero,1
  14:   02c0006f                jal     zero,40 <.L3>

00000018 <.L4>:
  18:   00478793                addi    a5,a5,4
  1c:   01078e63                beq     a5,a6,38 <.L7>

00000020 <.L5>:
  20:   0007a683                lw      a3,0(a5)
  24:   ffc7a703                lw      a4,-4(a5)
  28:   fee6f8e3                bgeu    a3,a4,18 <.L4>
  2c:   fed7ae23                sw      a3,-4(a5)
  30:   00e7a023                sw      a4,0(a5)
  34:   fe5ff06f                jal     zero,18 <.L4>

00000038 <.L7>:
  38:   00188893                addi    a7,a7,1
  3c:   01158863                beq     a1,a7,4c <.L2>

00000040 <.L3>:
  40:   00450793                addi    a5,a0,4
  44:   fcb36ee3                bltu    t1,a1,20 <.L5>
  48:   ff1ff06f                jal     zero,38 <.L7>

0000004c <.L2>:
  4c:   00261613                slli    a2,a2,0x2
  50:   00c50533                add     a0,a0,a2
  54:   ffc52503                lw      a0,-4(a0)
  58:   00008067                jalr    zero,0(ra)

main.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:   00050a13                addi    s4,a0,0
  24:   00058993                addi    s3,a1,0
  28:   00060a93                addi    s5,a2,0
  2c:   00060593                addi    a1,a2,0
  30:   00000537                lui     a0,0x0
  34:   00050513                addi    a0,a0,0 # 0 <test>
  38:   00000097                auipc   ra,0x0
  3c:   000080e7                jalr    ra,0(ra) # 38 <test+0x38>
  40:   02098663                beq     s3,zero,6c <.L2>
  44:   000a0413                addi    s0,s4,0
  48:   00299493                slli    s1,s3,0x2
  4c:   014484b3                add     s1,s1,s4
  50:   00000937                lui     s2,0x0

00000054 <.L3>:
  54:   00042583                lw      a1,0(s0)
  58:   00090513                addi    a0,s2,0 # 0 <test>
  5c:   00000097                auipc   ra,0x0
  60:   000080e7                jalr    ra,0(ra) # 5c <.L3+0x8>
  64:   00440413                addi    s0,s0,4
  68:   fe9416e3                bne     s0,s1,54 <.L3>

0000006c <.L2>:
  6c:   000a8613                addi    a2,s5,0
  70:   00098593                addi    a1,s3,0
  74:   000a0513                addi    a0,s4,0
  78:   00000097                auipc   ra,0x0
  7c:   000080e7                jalr    ra,0(ra) # 78 <.L2+0xc>
  80:   00050593                addi    a1,a0,0
  84:   00000537                lui     a0,0x0
  88:   00050513                addi    a0,a0,0 # 0 <test>
  8c:   00000097                auipc   ra,0x0
  90:   000080e7                jalr    ra,0(ra) # 8c <.L2+0x20>
  94:   01c12083                lw      ra,28(sp)
  98:   01812403                lw      s0,24(sp)
  9c:   01412483                lw      s1,20(sp)
  a0:   01012903                lw      s2,16(sp)
  a4:   00c12983                lw      s3,12(sp)
  a8:   00812a03                lw      s4,8(sp)
```

```
  ac:   00412a83                lw      s5,4(sp)
  b0:   02010113                addi    sp,sp,32
  b4:   00008067                jalr    zero,0(ra)

000000b8 <main>:
  b8:   fc010113                addi    sp,sp,-64
  bc:   02112e23                sw      ra,60(sp)
  c0:   000007b7                lui     a5,0x0
  c4:   00078793                addi    a5,a5,0 # 0 <test>
  c8:   0007a803                lw      a6,0(a5)
  cc:   0047a503                lw      a0,4(a5)
  d0:   0087a583                lw      a1,8(a5)
  d4:   00c7a603                lw      a2,12(a5)
  d8:   0107a683                lw      a3,16(a5)
  dc:   0147a703                lw      a4,20(a5)
  e0:   01012c23                sw      a6,24(sp)
  e4:   00a12e23                sw      a0,28(sp)
  e8:   02b12023                sw      a1,32(sp)
  ec:   02c12223                sw      a2,36(sp)
  f0:   02d12423                sw      a3,40(sp)
  f4:   02e12623                sw      a4,44(sp)
  f8:   0187a603                lw      a2,24(a5)
  fc:   01c7a683                lw      a3,28(a5)
 100:   0207a703                lw      a4,32(a5)
 104:   0247a783                lw      a5,36(a5)
 108:   00c12423                sw      a2,8(sp)
 10c:   00d12623                sw      a3,12(sp)
 110:   00e12823                sw      a4,16(sp)
 114:   00f12a23                sw      a5,20(sp)
 118:   00300613                addi    a2,zero,3
 11c:   00600593                addi    a1,zero,6
 120:   01810513                addi    a0,sp,24
 124:   00000097                auipc   ra,0x0
 128:   000080e7                jalr    ra,0(ra) # 124 <main+0x6c>
 12c:   00100613                addi    a2,zero,1
 130:   00400593                addi    a1,zero,4
 134:   00810513                addi    a0,sp,8
 138:   00000097                auipc   ra,0x0
 13c:   000080e7                jalr    ra,0(ra) # 138 <main+0x80>
 140:   00000513                addi    a0,zero,0
 144:   03c12083                lw      ra,60(sp)
 148:   04010113                addi    sp,sp,64
 14c:   00008067                jalr    zero,0(ra)
```

Отсюда видно, что вызовы подпрограмм, например, первый вызов `printf` внутри `test`, из псевдоинструкции `call printf` транслируется в:

```
 38:   00000097                auipc   ra,0x0
```

```
 3c:   000080e7                jalr    ra,0(ra) # 38 <test+0x38>
```

Ассемблер не имел возможности корректно определить целевой адрес перехода, поэтому сформировал некорректные нулевые значения и передал задачу исправления этих инструкций компоновщику.

Далее рассмотрим таблицу перемещений. Она содержит информацию обо всех "неоконченных" инструкциях, которая передается компоновщику. Выполним команду с опцией `-r`, выводящую таблицу перемещений:

```
riscv64-unknown-elf-objdump -r find.o main.o

RELOCATION RECORDS FOR [.text]:
OFFSET   TYPE              VALUE
00000000 R_RISCV_BRANCH    .L2
00000014 R_RISCV_JAL       .L3
0000001c R_RISCV_BRANCH    .L7
00000028 R_RISCV_BRANCH    .L4
00000034 R_RISCV_JAL       .L4
0000003c R_RISCV_BRANCH    .L2
00000044 R_RISCV_BRANCH    .L5
00000048 R_RISCV_JAL       .L7



main.o:     file format elf32-littleriscv

RELOCATION RECORDS FOR [.text]:
OFFSET   TYPE              VALUE
00000030 R_RISCV_HI20      .LC2
00000030 R_RISCV_RELAX     *ABS*
00000034 R_RISCV_LO12_I    .LC2
00000034 R_RISCV_RELAX     *ABS*
00000038 R_RISCV_CALL      printf
00000038 R_RISCV_RELAX     *ABS*
00000050 R_RISCV_HI20      .LC3
00000050 R_RISCV_RELAX     *ABS*
00000058 R_RISCV_LO12_I    .LC3
00000058 R_RISCV_RELAX     *ABS*
0000005c R_RISCV_CALL      printf
0000005c R_RISCV_RELAX     *ABS*
00000078 R_RISCV_CALL      find_order_statistic
00000078 R_RISCV_RELAX     *ABS*
00000084 R_RISCV_HI20      .LC4
00000084 R_RISCV_RELAX     *ABS*
00000088 R_RISCV_LO12_I    .LC4
00000088 R_RISCV_RELAX     *ABS*
0000008c R_RISCV_CALL      printf
0000008c R_RISCV_RELAX     *ABS*
000000c0 R_RISCV_HI20      .LANCHOR0
000000c0 R_RISCV_RELAX     *ABS*
000000c4 R_RISCV_LO12_I    .LANCHOR0
```

```
000000c4 R_RISCV_RELAX     *ABS*
00000124 R_RISCV_CALL      test
00000124 R_RISCV_RELAX     *ABS*
00000138 R_RISCV_CALL      test
00000138 R_RISCV_RELAX     *ABS*
00000040 R_RISCV_BRANCH    .L2
00000068 R_RISCV_BRANCH    .L3
```

Записи типа `R_RISCV_RELAX` необходимы для оптимизации, а все остальные содержат информацию для выполнения необходимых замен. Дело в том, что, если точки вызова подпрограммы и сама подпрограмма находятся друг от друга достаточно близко, то вместо использования пары инструкций `auipc+jalr`, позволяющей выполнять переходы в любую точку программы, можно ограничиться использованием инструкции `jal`, операнд которой занимает всего 20 бит.

## 3.4. Компоновка

Для компоновки используем следующую команду:

```
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 find.o main.o -o main.out
```

В ней нет опции `-W1,--no-relax`, а значит оптимизация, рассмотренная ранее, будет выполнена. Заметим, что компонуются файлы *совместно*, а не отдельно, как это было на предыдущих этапах.

Посмотрим на результат компоновки, выполнив следующую команду:

```
riscv64-unknown-elf-objdump -j .text -d -M no-aliases main.out >main.ds
```

В файле `main.ds` интересны только некоторые фрагменты, описывающие разработанные функции:

```
00010144 <find_order_statistic>:
   10144:        04058663                beq        a1,zero,10190
    ↪  <find_order_statistic+0x4c>
   10148:        00259813                slli        a6,a1,0x2
   1014c:        01050833                add        a6,a0,a6
   10150:        00000893                addi        a7,zero,0
   10154:        00100313                addi        t1,zero,1
   10158:        02c0006f                jal        zero,10184
    ↪  <find_order_statistic+0x40>
   1015c:        00478793                addi        a5,a5,4
   10160:        01078e63                beq        a5,a6,1017c
    ↪  <find_order_statistic+0x38>
   10164:        0007a683                lw        a3,0(a5)
   10168:        ffc7a703                lw        a4,-4(a5)
   1016c:        fee6f8e3                bgeu        a3,a4,1015c
    ↪  <find_order_statistic+0x18>
   10170:        fed7ae23                sw        a3,-4(a5)
   10174:        00e7a023                sw        a4,0(a5)
   10178:        fe5ff06f                jal        zero,1015c
    ↪  <find_order_statistic+0x18>
   1017c:        00188893                addi        a7,a7,1
```

```
   10180:        01158863                beq        a1,a7,10190
    ↪ <find_order_statistic+0x4c>
   10184:        00450793                addi        a5,a0,4
   10188:        fcb36ee3                bltu        t1,a1,10164
    ↪ <find_order_statistic+0x20>
   1018c:        ff1ff06f                jal        zero,1017c
    ↪ <find_order_statistic+0x38>
   10190:        00261613                slli        a2,a2,0x2
   10194:        00c50533                add        a0,a0,a2
   10198:        ffc52503                lw        a0,-4(a0)
   1019c:        00008067                jalr        zero,0(ra)

000101a0 <test>:
   101a0:        fe010113                addi        sp,sp,-32
   101a4:        00112e23                sw        ra,28(sp)
   101a8:        00812c23                sw        s0,24(sp)
   101ac:        00912a23                sw        s1,20(sp)
   101b0:        01212823                sw        s2,16(sp)
   101b4:        01312623                sw        s3,12(sp)
   101b8:        01412423                sw        s4,8(sp)
   101bc:        01512223                sw        s5,4(sp)
   101c0:        00050a13                addi        s4,a0,0
   101c4:        00058993                addi        s3,a1,0
   101c8:        00060a93                addi        s5,a2,0
   101cc:        00060593                addi        a1,a2,0
   101d0:        00025537                lui        a0,0x25
   101d4:        a2050513                addi        a0,a0,-1504 # 24a20
    ↪ <__clzsi2+0x78>
   101d8:        35c000ef                jal        ra,10534 <printf>
   101dc:        02098463                beq        s3,zero,10204
    ↪ <test+0x64>
   101e0:        000a0413                addi        s0,s4,0
   101e4:        00299493                slli        s1,s3,0x2
   101e8:        014484b3                add        s1,s1,s4
   101ec:        00025937                lui        s2,0x25
   101f0:        00042583                lw        a1,0(s0)
   101f4:        a3c90513                addi        a0,s2,-1476 # 24a3c
    ↪ <__clzsi2+0x94>
   101f8:        33c000ef                jal        ra,10534 <printf>
   101fc:        00440413                addi        s0,s0,4
   10200:        fe9418e3                bne        s0,s1,101f0 <test+0x50>
   10204:        000a8613                addi        a2,s5,0
   10208:        00098593                addi        a1,s3,0
   1020c:        000a0513                addi        a0,s4,0
   10210:        f35ff0ef                jal        ra,10144
    ↪ <find_order_statistic>
   10214:        00050593                addi        a1,a0,0
   10218:        00025537                lui        a0,0x25
   1021c:        a4050513                addi        a0,a0,-1472 # 24a40
    ↪ <__clzsi2+0x98>
```

```
   10220:       314000ef                jal      ra,10534 <printf>
   10224:       01c12083                lw       ra,28(sp)
   10228:       01812403                lw       s0,24(sp)
   1022c:       01412483                lw       s1,20(sp)
   10230:       01012903                lw       s2,16(sp)
   10234:       00c12983                lw       s3,12(sp)
   10238:       00812a03                lw       s4,8(sp)
   1023c:       00412a83                lw       s5,4(sp)
   10240:       02010113                addi     sp,sp,32
   10244:       00008067                jalr     zero,0(ra)

00010248 <main>:
   10248:       fc010113                addi     sp,sp,-64
   1024c:       02112e23                sw       ra,60(sp)
   10250:       000257b7                lui      a5,0x25
   10254:       9f878793                addi     a5,a5,-1544 # 249f8
    ↪  <__clzsi2+0x50>
   10258:       0007a803                lw       a6,0(a5)
   1025c:       0047a503                lw       a0,4(a5)
   10260:       0087a583                lw       a1,8(a5)
   10264:       00c7a603                lw       a2,12(a5)
   10268:       0107a683                lw       a3,16(a5)
   1026c:       0147a703                lw       a4,20(a5)
   10270:       01012c23                sw       a6,24(sp)
   10274:       00a12e23                sw       a0,28(sp)
   10278:       02b12023                sw       a1,32(sp)
   1027c:       02c12223                sw       a2,36(sp)
   10280:       02d12423                sw       a3,40(sp)
   10284:       02e12623                sw       a4,44(sp)
   10288:       0187a603                lw       a2,24(a5)
   1028c:       01c7a683                lw       a3,28(a5)
   10290:       0207a703                lw       a4,32(a5)
   10294:       0247a783                lw       a5,36(a5)
   10298:       00c12423                sw       a2,8(sp)
   1029c:       00d12623                sw       a3,12(sp)
   102a0:       00e12823                sw       a4,16(sp)
   102a4:       00f12a23                sw       a5,20(sp)
   102a8:       00300613                addi     a2,zero,3
   102ac:       00600593                addi     a1,zero,6
   102b0:       01810513                addi     a0,sp,24
   102b4:       eedff0ef                jal      ra,101a0 <test>
   102b8:       00100613                addi     a2,zero,1
   102bc:       00400593                addi     a1,zero,4
   102c0:       00810513                addi     a0,sp,8
   102c4:       eddff0ef                jal      ra,101a0 <test>
   102c8:       00000513                addi     a0,zero,0
   102cc:       03c12083                lw       ra,60(sp)
   102d0:       04010113                addi     sp,sp,64
   102d4:       00008067                jalr     zero,0(ra)
```

Во-первых, можно заметить, что пары `auipc+jalr` были успешно оптимизированы до `jal`, позволив сократить число инструкций. Во-вторых, в результирующих инструкциях теперь верные адреса.

# 4. Создание статической библиотеки

В некоторых случаях, при большом количестве объектных файлов, удобно использовать так называемые статические библиотеки. По сути, это архивы объектных файлов, из которых компоновщик выбирает нужные. В случае данной задачи это не столь необходимо, поскольку используется только 2 объектных файла, а в библиотеку и то имеет смысл поместить лишь один. Несмотря на это, создадим статическую библиотеку.

Для этого, используем соответствующую утилиту, которой передадим сформированный ранее объектный файл:

```
riscv64-unknown-elf-ar -rsc libfind.a find.o
```

Результирующим файлом является `libfind.a`, рассмотрим его содержимое:

```
R:\study\lowlevelprog\c>riscv64-unknown-elf-ar -t libfind.a
find.o
```

Видим, что в ней действительно находится переданный файл. Теперь используем полученную библиотеку для сборки:

```
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -O1 --save-temps main.c
↪  libfind.a -o mainwlib.out
```

Таблица символов полученного файла будет содержать запись о разработанной функции, а также множество символов из подключенной библиотеки `<stdio.h>`:

```
    ...
0001027c g     F .text         0000005c find_order_statistic
    ...
```

# 5. Разработка Make-файла

Для упрощения процесса сборки проекта удобно использовать так называемые make-файлы. По сути, они содержат консольные команды, необходимые для сборки проекта. Ниже представлен разработанный для данного проекта make-file, в котором создано две зависимости: для сборки проекта и удаления всех файлов, кроме исходных.

```
1  all: output
2
3  output: main.o findlib.a
4      mingw32-gcc-6.3.0 main.o findlib.a -o output
5
6  main.o: main.c
7      mingw32-gcc-6.3.0 -c main.c
8
9  findlib.a: find.o find.h
10     mingw32-gcc-ar -rsc findlib.a find.o
11
12 find.o: find.c
```

```
13      mingw32-gcc-6.3.0 -c find.c
14
15  clean:
16      del -rf *.a *.o *.exe output
```

Листинг 8: Make-файл

Для его использования достаточно вызвать утилиту `mingw32-make` и передать ей имя make-файла.

```
R:\study\lowlevelprog\c>dir
 Том в устройстве R имеет метку R
 Серийный номер тома: 0E47-15BB

 Содержимое папки R:\study\lowlevelprog\c

14.12.2022  02:39    <DIR>          .
12.12.2022  13:47    <DIR>          ..
13.12.2022  00:11               374 find.c
13.12.2022  00:11               238 find.h
13.12.2022  18:46               653 main.c
12.12.2022  22:55               290 Makefile
               4 файлов          1 555 байт
               2 папок  469 587 951 616 байт свободно

R:\study\lowlevelprog\c>mingw32-make -f Makefile
mingw32-gcc-6.3.0 -c main.c
mingw32-gcc-6.3.0 -c find.c
mingw32-gcc-ar -rsc findlib.a find.o
mingw32-gcc-6.3.0 main.o findlib.a -o output

R:\study\lowlevelprog\c>dir
 Том в устройстве R имеет метку R
 Серийный номер тома: 0E47-15BB

 Содержимое папки R:\study\lowlevelprog\c

14.12.2022  02:39    <DIR>          .
12.12.2022  13:47    <DIR>          ..
13.12.2022  00:11               374 find.c
13.12.2022  00:11               238 find.h
14.12.2022  02:39               858 find.o
14.12.2022  02:39             1 016 findlib.a
13.12.2022  18:46               653 main.c
14.12.2022  02:39             1 252 main.o
12.12.2022  22:55               290 Makefile
14.12.2022  02:39            41 570 output.exe
               8 файлов         46 251 байт
               2 папок  469 587 894 272 байт свободно
```

Рис. 5.1: Использование make-файла

Вызывая в командной строке сформированный файл `output.exe`, видим ожидаемый вывод:

```
R:\study\lowlevelprog\c>output.exe
3-th order statistic of { 4 3 5 7 2 8 } is 4
1-th order statistic of { 11 7 -4 0 } is 0
```

# 6. Вывод

В ходе выполнения лабораторной работы, была реализована программа поиска $k$-й порядковой статистики массива. Программа была разделена на 3 файла: реализацию функциональной подпрограммы, тестовую подпрограмму, вызывающую функицональную, а также заголовочный файл. Проект был собран ‘по шагам”, во время сборки были проанализированы промежуточные файлы. Были разработаны и протестированы статическая библиотека, а также make-файл, упрощающие сборку программ.