

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 7

Выполнил студент гр. 3530901/90004 _____ И.А. Сергеев
(подпись)

Принял старший преподаватель _____ А.О. Алексюк
(подпись)

“ ____ ” _____ 2021 г.

Санкт-
Петербург
2021

Цель работы:

1. Изучить методические материалы, опубликованные на сайте курса.
2. Установить пакет средств разработки “SiFive GNU Embedded Toolchain” для RISC-V.
3. На языке C разработать функцию, реализующую определенную вариантом задания функциональность. Поместить определение функции в отдельный исходный файл, оформить заголовочный файл. Разработать тестовую программу на языке C.
4. 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.
5. 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 7: Определение k-й порядковой статистики in-place.

1. Функция на C

Для начала разработаем функцию на C, которая будет реализовывать поиск k-ой порядковой статистики. Напишем функцию в отдельном файле.

На рисунке 1.1 представлен заголовочный файл sort.h.

```
#ifndef STATISTICS_SORT_H
#define STATISTICS_SORT_H
int sort(int array[], int size, int k);
#endif //STATISTICS_SORT_H
```

Рис. 1.1. sort.h

Рисунком 1.2 представлен основной файл sort.c.

```
#include "sort.h"

int sort(int array[], int size, int k) {
    int temp;
    int j;
    for (int i = 1; i < size; i++) {
        temp = array[i];
        j = i - 1;
        while (j >= 0 && array[j] > temp) {
            array[j + 1] = array[j];
            j = j - 1;
        }
        array[j + 1] = temp;
    }
    return array[k-1];
}
```

Рис 1.2. sort.c

Рисунком 1.3 представлена тестовая программа main.c

```

#include <stdio.h>
#include "../sort.h"

int main() {
    int k = 4;
    /* 1 5 6 9 15 */
    int array[] = {15, 6, 9, 1, 5};
    int size = sizeof(array) / sizeof(int);
    for (int i = 0; i < size; i++) {
        printf( _Format: "%u ", array[i]);
    }
    printf( _Format: "\n");
    sort(array, size, k);
    for (int i = 0; i < size; i++) {
        printf( _Format: "%u ", array[i]);
    }
    printf( _Format: "\n%u", sort(array, size, k));
    return 0;
}

```

Рис. 1.3. main.c

2. Препроцессирование

Сборка программы «по шагам». Первым шагом является препроцессирование файлов с исходными текстами. Для этого используется пакет разработки «SiFive GNU Embedded Toolchain». Чтобы это выполнить, необходимо использовать команды:

```

riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -O1 -E main.c -o main.i
riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -O1 -E sort.c -o sort.i

```

Результат препроцессирования содержится в файлах main.i и sort.i. По причине того, что main.c содержит заголовочный файл стандартной библиотеки языка C stdio.h, результат препроцессирования этого файла имеет достаточно много добавочных строк.

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

# 2 "main.c" 2
# 1 "sort.h" 1

# 4 "sort.h"
int sort(int array[], int size, int k);
# 3 "main.c" 2

int main() {
    int k = 4;

    int array[] = {15, 6, 9, 1, 5};
    int size = sizeof(array) / sizeof(int);
    for (int i = 0; i < size; i++) {
        printf("%u ", array[i]);
    }
    printf("\n");
    sort(array, size, k);
    for (int i = 0; i < size; i++) {
        printf("%u ", array[i]);
    }
    printf("\n%u", sort(array, size, k));
    return 0;
}
```

Рис. 2.1. Фрагмент изначального кода в main.i.

```

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

int sort(int array[], int size, int k);
# 2 "sort.c" 2

int sort(int array[], int size, int k) {
    int temp;
    int j;
    for (int i = 1; i < size; i++) {
        temp = array[i];
        j = i - 1;
        while (j >= 0 && array[j] > temp) {
            array[j + 1] = array[j];
            j = j - 1;
        }
        array[j + 1] = temp;
    }
    return array[k-1];
}

```

Рис. 2.2. Фрагмент изначального кода в sort.i.

3. Компиляция

Компиляция осуществляется следующими командами:

```

riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -O1 -S main.i -o main.s
riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -O1 -S sort.i -o sort.s

```

Наибольший интерес представляет файл main.s, так как в нем можно заметить обращение к подпрограмме sort (значение регистра ra, содержащее адрес возврата из main, сохраняется в стеке).

После компиляции получим следующие файлы, содержащие инструкции на RISC-V:

Файл main.s:

```
.file "main.c"
.option nopic
.attribute arch, "rv64i2p0_a2p0_c2p0"
.attribute unaligned_access, 0
.attribute stack_align, 16
.text
.section .rodata.str1.8,"aMS",@progbits,1
.align 3
.LC1:
.string "%u "
.align 3
.LC2:
.string "\n%u"
.text
.align 1
.globl main
.type main, @function
main:
    addi    sp,sp,-80
    sd ra,72(sp)
    sd s0,64(sp)
    sd s1,56(sp)
    sd s2,48(sp)
    sd s3,40(sp)
    lui     a5,%hi(.LANCHOR0)
    addi    a5,a5,%lo(.LANCHOR0)
    ld a4,0(a5)
    sd a4,8(sp)
    ld a4,8(a5)
    sd a4,16(sp)
    lw a5,16(a5)
    sw a5,24(sp)
    addi    s0,sp,8
    addi    s2,sp,28
    mv s1,s0
    lui     s3,%hi(.LC1)
.L2:
    lw a1,0(s1)
    addi    a0,s3,%lo(.LC1)
    call    printf
    addi    s1,s1,4
    bne     s1,s2,.L2
    li a0,10
    call    putchar
    li a2,4
    li a1,5
    addi    a0,sp,8
    call    sort
    lui     s1,%hi(.LC1)
```

```

.L3:
    lw a1,0(s0)
    addi a0,s1,%lo(.LC1)
    call printf
    addi s0,s0,4
    bne s0,s2,.L3
    li a2,4
    li a1,5
    addi a0,sp,8
    call sort
    mv a1,a0
    lui a0,%hi(.LC2)
    addi a0,a0,%lo(.LC2)
    call printf
    li a0,0
    ld ra,72(sp)
    ld s0,64(sp)
    ld s1,56(sp)
    ld s2,48(sp)
    ld s3,40(sp)
    addi sp,sp,80
    jr ra
.size main, .-main
.section .rodata
.align 3
.set .LANCHOR0,. + 0
.LC0:
    .word 15
    .word 6
    .word 9
    .word 1
    .word 5
    .ident "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"

```

Также получим файл с инструкциями для sort.c

Файл sort.s:

```
.file "sort.c"
.option nopic
.attribute arch, "rv64i2p0 a2p0_c2p0"
.attribute unaligned_access, 0
.attribute stack_align, 16
.text
.align 1
.globl sort
.type sort, @function
sort:
    li a5,1
    ble a1,a5,.L2
    mv t1,a0
    addiw t3,a1,-1
    li a7,0
    li a6,-1
    j .L6
.L4:
    addi a4,a4,1
    slli a4,a4,2
    add a4,a0,a4
    sw a1,0(a4)
    addiw a7,a7,1
    addi t1,t1,4
    beq a7,t3,.L2
.L6:
    lw a1,4(t1)
    sext.w a4,a7
    mv a5,t1
    blt a7,zero,.L4
.L3:
    lw a3,0(a5)
    ble a3,a1,.L4
    sw a3,4(a5)
    addiw a4,a4,-1
    addi a5,a5,-4
    bne a4,a6,.L3
    j .L4
.L2:
    slli a5,a2,2
    add a0,a0,a5
    lw a0,-4(a0)
    ret
.size sort, .-sort
.ident "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"
```

4. Ассемблирование

Ассемблирование осуществляется следующими командами:

```
riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -v -c main.s -o main.o  
riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -v -c sort.s -o sort.o
```

После выполнения данных команд у нас появятся объектные файлы main.o и sort.o.

Получим заголовки секций файла main.o с помощью команды:

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

```
Sections:  
Idx Name          Size      VMA               LMA               File off  Algn  
 0 .text          000000a0 0000000000000000 0000000000000000 00000040 2**1  
                CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE  
 1 .data          00000000 0000000000000000 0000000000000000 000000e0 2**0  
                CONTENTS, ALLOC, LOAD, DATA  
 2 .bss           00000000 0000000000000000 0000000000000000 000000e0 2**0  
                ALLOC  
 3 .rodata.str1.8 0000000c 0000000000000000 0000000000000000 000000e0 2**3  
                CONTENTS, ALLOC, LOAD, READONLY, DATA  
 4 .rodata        00000014 0000000000000000 0000000000000000 000000f0 2**3  
                CONTENTS, ALLOC, LOAD, READONLY, DATA  
 5 .comment       00000031 0000000000000000 0000000000000000 00000104 2**0  
                CONTENTS, READONLY  
 6 .riscv.attributes 00000026 0000000000000000 0000000000000000 00000135 2**0  
                CONTENTS, READONLY
```

Рис. 4.1 заголовки секций main.o

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
.rodata	секция данных в формате read-only

Получим таблицу символов файла main.o используя команду:

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

```
SYMBOL TABLE:
0000000000000000 1      df *ABS* 0000000000000000 main.c
0000000000000000 1      d  .text 0000000000000000 .text
0000000000000000 1      d  .data 0000000000000000 .data
0000000000000000 1      d  .bss  0000000000000000 .bss
0000000000000000 1      d  .rodata.str1.8 0000000000000000 .rodata.str1.8
0000000000000000 1      d  .rodata      0000000000000000 .rodata
0000000000000000 1      .rodata      0000000000000000 .LANCHOR0
0000000000000000 1      .rodata.str1.8 0000000000000000 .LC1
0000000000000000 1      .rodata.str1.8 0000000000000000 .LC2
0000000000000008 1      .text 0000000000000000 .L2
000000000000002c 1      .text 0000000000000000 .L3
000000000000005c 1      .text 0000000000000000 .L3
0000000000000000 1      d  .comment      0000000000000000 .comment
0000000000000000 1      d  .riscv.attributes 0000000000000000 .riscv.attributes
0000000000000000 g      F  .text 00000000000000a0 main
0000000000000000      *UND* 0000000000000000 printf
0000000000000000      *UND* 0000000000000000 putchar
0000000000000000      *UND* 0000000000000000 sort
```

Рис. 4.2. Таблица символов файла main.o

В таблице символов main.o имеется запись: символ «sort» типа *UND*. Эта запись означает, что символ «sort» 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. То же самое относится и к символу «printf» и «putchar».

Теперь получим дизассемблированный файл main.o при помощи команды:

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

```

main.o:      file format elf64-littleriscv

Disassembly of section .text:

0000000000000000 <main>:
 0: 715d          c.addi16sp      sp,-80
 2: e486          c.sdsp         ra,72(sp)
 4: e0a2          c.sdsp         s0,64(sp)
 6: fc26          c.sdsp         s1,56(sp)
 8: f84a          c.sdsp         s2,48(sp)
 a: f44e          c.sdsp         s3,40(sp)
 c: 000007b7      lui          a5,0x0
                                c: R_RISCV_HI20 .ANCHOR0
                                c: R_RISCV_RELAX *ABS*
10: 00078793      addi          a5,a5,0 # 0 <main>
                                10: R_RISCV_LO12_I .ANCHOR0
                                10: R_RISCV_RELAX *ABS*
14: 6398          c.ld           a4,0(a5)
16: e43a          c.sdsp         a4,8(sp)
18: 6798          c.ld           a4,8(a5)
1a: e83a          c.sdsp         a4,16(sp)
1c: 4b9c          c.lw           a5,16(a5)
1e: cc3e          c.swsp         a5,24(sp)
20: 0020          c.addi4spn     s0,sp,8
22: 01c10913      addi          s2,sp,28
26: 84a2          c.mv           s1,s0
28: 000009b7      lui           s3,0x0
                                28: R_RISCV_HI20 .LC1
                                28: R_RISCV_RELAX *ABS*

```

Рис. 4.3 Дизассемблированный файл main.o (1)

```

000000000000002c <.L2>:
2c: 408c          c.lw      a1,0(s1)
2e: 00098513     addi      a0,s3,0 # 0 <main>
2e: R_RISCV_LO12_I .LC1
2e: R_RISCV_RELAX  *ABS*
32: 00000097     auipc     ra,0x0
32: R_RISCV_CALL   printf
32: R_RISCV_RELAX  *ABS*
36: 000080e7     jalr      ra,0(ra) # 32 <.L2+0x6>
3a: 0491         c.addi     s1,4
3c: ff2498e3     bne       s1,s2,2c <.L2>
3c: R_RISCV_BRANCH .L2
40: 4529         c.li      a0,10
42: 00000097     auipc     ra,0x0
42: R_RISCV_CALL   putchar
42: R_RISCV_RELAX  *ABS*
46: 000080e7     jalr      ra,0(ra) # 42 <.L2+0x16>
4a: 4609         c.li      a2,2
4c: 4595         c.li      a1,5
4e: 0028         c.addi4spn a0,sp,8
50: 00000097     auipc     ra,0x0
50: R_RISCV_CALL   sort
50: R_RISCV_RELAX  *ABS*
54: 000080e7     jalr      ra,0(ra) # 50 <.L2+0x24>
58: 000004b7     lui       s1,0x0
58: R_RISCV_HI20   .LC1
58: R_RISCV_RELAX  *ABS*

000000000000005c <.L3>:
5c: 400c          c.lw      a1,0(s0)
5e: 00048513     addi      a0,s1,0 # 0 <main>
5e: R_RISCV_LO12_I .LC1
5e: R_RISCV_RELAX  *ABS*
62: 00000097     auipc     ra,0x0
62: R_RISCV_CALL   printf
62: R_RISCV_RELAX  *ABS*
66: 000080e7     jalr      ra,0(ra) # 62 <.L3+0x6>
6a: 0411         c.addi     s0,4
6c: ff2418e3     bne       s0,s2,5c <.L3>
6c: R_RISCV_BRANCH .L3
70: 4609         c.li      a2,2
72: 4595         c.li      a1,5
74: 0028         c.addi4spn a0,sp,8
76: 00000097     auipc     ra,0x0
76: R_RISCV_CALL   sort
76: R_RISCV_RELAX  *ABS*
7a: 000080e7     jalr      ra,0(ra) # 76 <.L3+0x1a>
7e: 85aa         c.mv      a1,a0

```

Рис. 4.4. Дизассемблированный файл main.o (2)

```

80: 00000537          lui      a0,0x0
                        80: R_RISCV_HI20      .LC2
                        80: R_RISCV_RELAX     *ABS*
84: 00050513          addi     a0,a0,0 # 0 <main>
                        84: R_RISCV_LO12_I    .LC2
                        84: R_RISCV_RELAX     *ABS*
88: 00000097          auipc     ra,0x0
                        88: R_RISCV_CALL      printf
                        88: R_RISCV_RELAX     *ABS*
8c: 000080e7          jalr      ra,0(ra) # 88 <.L3+0x2c>
90: 4501             c.li      a0,0
92: 60a6             c.ldsp    ra,72(sp)
94: 6406             c.ldsp    s0,64(sp)
96: 74e2             c.ldsp    s1,56(sp)
98: 7942             c.ldsp    s2,48(sp)
9a: 79a2             c.ldsp    s3,40(sp)
9c: 6161             c.addi16sp    sp,80
9e: 8082             c.jr      ra

```

Рис. 4.5. Дизассемблированный файл main.o (3)

Теперь получим таблицу перемещений файла main.o и sort.o при помощи команды:

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

```

sort.o:      file format elf64-littleriscv

RELOCATION RECORDS FOR [.text]:
OFFSET      TYPE      VALUE
0000000000000002 R_RISCV_BRANCH  .L2
0000000000000010 R_RISCV_RVC_JUMP .L6
000000000000001e R_RISCV_BRANCH  .L2
000000000000002c R_RISCV_BRANCH  .L4
0000000000000032 R_RISCV_BRANCH  .L4
000000000000003c R_RISCV_BRANCH  .L3
0000000000000040 R_RISCV_RVC_JUMP .L4


main.o:      file format elf64-littleriscv

RELOCATION RECORDS FOR [.text]:
OFFSET      TYPE      VALUE
000000000000000c R_RISCV_HI20    .LANCHOR0
000000000000000c R_RISCV_RELAX   *ABS*
0000000000000010 R_RISCV_LO12_I  .LANCHOR0
0000000000000010 R_RISCV_RELAX   *ABS*
0000000000000028 R_RISCV_HI20    .LC1
0000000000000028 R_RISCV_RELAX   *ABS*
000000000000002e R_RISCV_LO12_I  .LC1
000000000000002e R_RISCV_RELAX   *ABS*
0000000000000032 R_RISCV_CALL    printf
0000000000000032 R_RISCV_RELAX   *ABS*
0000000000000042 R_RISCV_CALL    putchar
0000000000000042 R_RISCV_RELAX   *ABS*
0000000000000050 R_RISCV_CALL    sort
0000000000000050 R_RISCV_RELAX   *ABS*
0000000000000058 R_RISCV_HI20    .LC1
0000000000000058 R_RISCV_RELAX   *ABS*
000000000000005e R_RISCV_LO12_I  .LC1
000000000000005e R_RISCV_RELAX   *ABS*
0000000000000062 R_RISCV_CALL    printf
0000000000000062 R_RISCV_RELAX   *ABS*
0000000000000076 R_RISCV_CALL    sort
0000000000000076 R_RISCV_RELAX   *ABS*
0000000000000080 R_RISCV_HI20    .LC2
0000000000000080 R_RISCV_RELAX   *ABS*
0000000000000084 R_RISCV_LO12_I  .LC2
0000000000000084 R_RISCV_RELAX   *ABS*
0000000000000088 R_RISCV_CALL    printf
0000000000000088 R_RISCV_RELAX   *ABS*
000000000000003c R_RISCV_BRANCH  .L2
000000000000006c R_RISCV_BRANCH  .L3

```

Рис. 4.6. Таблица перемещений

В таблице перемещений для main.o наблюдаем вызов метода sort. Записи типа “R_RISCV_RELAX” заносятся в таблицу перемещений в дополнение к записям типа “R_RISCV_CALL” (и некоторым другим) и сообщают

компоновщику, что пара инструкций, обеспечивающих вызов подпрограммы, может быть оптимизирована.

Получим заготовки секций файла sort.o используя команду:

riscv64-unknown-elf-objdump.exe -h sort.o

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

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .text          0000004e  0000000000000000  0000000000000000  00000040  2**1
    CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data          00000000  0000000000000000  0000000000000000  0000008e  2**0
    CONTENTS, ALLOC, LOAD, DATA
  2 .bss           00000000  0000000000000000  0000000000000000  0000008e  2**0
    ALLOC
  3 .comment       00000031  0000000000000000  0000000000000000  0000008e  2**0
    CONTENTS, READONLY
  4 .riscv.attributes 00000026  0000000000000000  0000000000000000  000000bf  2**0
    CONTENTS, READONLY
```

Рис. 4.7. Заготовки секций файла sort.o

Затем получим таблицу символов файла sort.o с помощью команды:

riscv64-unknown-elf-objdump.exe -t sort.o

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

SYMBOL TABLE:
0000000000000000 1      df *ABS* 0000000000000000 sort.c
0000000000000000 1      d  .text 0000000000000000 .text
0000000000000000 1      d  .data 0000000000000000 .data
0000000000000000 1      d  .bss  0000000000000000 .bss
0000000000000042 1      .text 0000000000000000 .L2
0000000000000022 1      .text 0000000000000000 .L6
0000000000000012 1      .text 0000000000000000 .L4
0000000000000030 1      .text 0000000000000000 .L3
0000000000000000 1      d  .comment 0000000000000000 .comment
0000000000000000 1      d  .riscv.attributes 0000000000000000 .riscv.attributes
0000000000000000 g      F  .text 000000000000004c sort
```

Рис. 4.8. Таблица символов файла sort.o

5. Компоновка

Компоновка осуществляется следующей командой:

riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -v main.o sort.o

Исполняемый файл a.out(фрагмент)

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

```
a.out:      file format elf64-littleriscv
...
00000000000010156 <main>:
 10156: 715d          c.addi16sp sp,-80
 10158: e486          c.sdsp ra,72(sp)
 1015a: e0a2          c.sdsp s0,64(sp)
 1015c: fc26          c.sdsp s1,56(sp)
 1015e: f84a          c.sdsp s2,48(sp)
 10160: f44e          c.sdsp s3,40(sp)
 10162: 67f5          c.lui  a5,0x1d
 10164: c5078793      addi   a5,a5,-944 # 1cc50 <__clzdi2+0x42>
 10168: 6398          c.ld   a4,0(a5)
 1016a: e43a          c.sdsp a4,8(sp)
 1016c: 6798          c.ld   a4,8(a5)
 1016e: e83a          c.sdsp a4,16(sp)
 10170: 4b9c          c.lw   a5,16(a5)
 10172: cc3e          c.swsp a5,24(sp)
 10174: 0020          c.addi4spn s0,sp,8
 10176: 01c10913      addi   s2,sp,28
 1017a: 84a2          c.mv   s1,s0
 1017c: 69f5          c.lui  s3,0x1d
 1017e: 408c          c.lw   a1,0(s1)
 10180: c4098513      addi   a0,s3,-960 # 1cc40 <__clzdi2+0x32>
 10184: 1f2000ef      jal    ra,10376 <printf>
 10188: 0491          c.addi s1,4
 1018a: ff249ae3      bne    s1,s2,1017e <main+0x28>
 1018e: 4529          c.li   a0,10
 10190: 216000ef      jal    ra,103a6 <putchar>
```

```

10194: 4609          c.li    a2,2
10196: 4595          c.li    a1,5
10198: 0028          c.addi4spn a0,sp,8
1019a: 03c000ef      jal     ra,101d6 <sort>
1019e: 64f5          c.lui    s1,0x1d
101a0: 400c          c.lw     a1,0(s0)
101a2: c4048513      addi     a0,s1,-960 # 1cc40 <__clzdi2+0x32>
101a6: 1d0000ef      jal     ra,10376 <printf>
101aa: 0411          c.addi   s0,4
101ac: ff241ae3      bne     s0,s2,101a0 <main+0x4a>
101b0: 4609          c.li    a2,2
101b2: 4595          c.li    a1,5
101b4: 0028          c.addi4spn a0,sp,8
101b6: 020000ef      jal     ra,101d6 <sort>
101ba: 85aa          c.mv     a1,a0
101bc: 6575          c.lui    a0,0x1d
101be: c4850513      addi     a0,a0,-952 # 1cc48 <__clzdi2+0x3a>
101c2: 1b4000ef      jal     ra,10376 <printf>
101c6: 4501          c.li    a0,0
101c8: 60a6          c.ldsp   ra,72(sp)
101ca: 6406          c.ldsp   s0,64(sp)
101cc: 74e2          c.ldsp   s1,56(sp)
101ce: 7942          c.ldsp   s2,48(sp)
101d0: 79a2          c.ldsp   s3,40(sp)
101d2: 6161          c.addi16sp sp,80
101d4: 8082          c.jr     ra

00000000000101d6 <sort>:
101d6: 4785          c.li    a5,1
101d8: 04b7d063      bge     a5,a1,10218 <sort+0x42>
101dc: 832a          c.mv     t1,a0
101de: fff58e1b      addiw   t3,a1,-1
101e2: 4881          c.li    a7,0
101e4: 587d          c.li    a6,-1
101e6: a809          c.j     101f8 <sort+0x22>
101e8: 0705          c.addi   a4,1
101ea: 070a          c.slli   a4,0x2
101ec: 972a          c.add    a4,a0
101ee: c30c          c.sw     a1,0(a4)
101f0: 2885          c.addiw  a7,1
101f2: 0311          c.addi   t1,4
101f4: 03c88263      beq     a7,t3,10218 <sort+0x42>
101f8: 00432583      lw      a1,4(t1) # 10150 <frame_dummy+0x16>
101fc: 0008871b      addiw   a4,a7,0
10200: 879a          c.mv     a5,t1
10202: fe08c3e3      blt     a7,zero,101e8 <sort+0x12>
10206: 4394          c.lw     a3,0(a5)
10208: fed5d0e3      bge     a1,a3,101e8 <sort+0x12>
1020c: c3d4          c.sw     a3,4(a5)
1020e: 377d          c.addiw  a4,-1
10210: 17f1          c.addi   a5,-4
10212: ff071ae3      bne     a4,a6,10206 <sort+0x30>
10216: bfc9          c.j     101e8 <sort+0x12>
10218: 00261793      slli    a5,a2,0x2
1021c: 953e          c.add    a0,a5
1021e: 4108          c.lw     a0,0(a0)
10220: 8082          c.jr     ra

```

6. Создание статической библиотеки и make-файлов

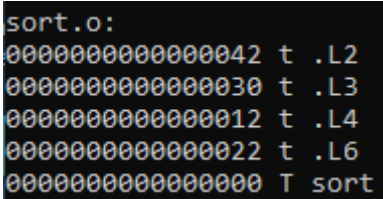
Статическая библиотека (static library) является, по сути, архивом (набором, коллекцией) объектных файлов, среди которых компоновщик выбирает «полезные» для данной программы. Объектный файл считается «полезным», если в нем определяется еще не разрешенный компоновщиком символ.

Выделим функцию `sort` в отдельную статическую библиотеку. Для этого необходимо получить объектный файл `sort.o` и собрать библиотеку.

```
riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -O1 -c sort.c -o sort.o  
riscv64-unknown-elf-ar.exe -rsc lib.a sort.o
```

Рассмотрим список символов библиотеки, используя команду:

```
riscv64-unknown-elf-nm.exe lib.a
```



```
sort.o:  
0000000000000042 t .L2  
0000000000000030 t .L3  
0000000000000012 t .L4  
0000000000000022 t .L6  
0000000000000000 T sort
```

Рис. 6.1. Список символов `lib.a`

В выводе утилиты «`nm`» кодом «`T`» обозначаются символы, определенные в соответствующем объектном файле.

Теперь, имея собранную библиотеку, создадим исполняемый файл тестовой программы «`main.c`», при помощи следующей команды:

```
riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -O1 --save-temps main.c  
lib.a
```

Убедимся, что в состав программы вошло содержание объектного файла `sort.o`, при помощи таблицы символов исполняемого файла

Таблица символов исполняемого файла (фрагмент):

```
riscv64-unknown-elf-objdump.exe -t a.out
```

```
a.out:      file format elf64-littleriscv

SYMBOL TABLE:
0000000000000000 l      df *ABS* 0000000000000000 main.c
0000000000000000 l      df *ABS* 0000000000000000 sort.c
0000000000001cb66 g      F .text 0000000000000014 .hidden __umodsi3
000000000000101d6 g      F .text 000000000000004c sort
00000000000012942 g      F .text 0000000000000002 __sfp_lock_acquire
```

Рис. 6.2. Фрагмент таблицы символов исполняемого файла

Можно заметить, что в состав программы вошло содержимое объектного файла sort.o.

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

Makefile для создания статической библиотеки:

```
# "Фиктивные" цели
.PHONY: all clean
# Исходные файлы, необходимые для сборки библиотеки
OBJS = sort.c \

#Вызываемые приложения
AR = riscv64-unknown-elf-ar.exe
CC = riscv64-unknown-elf-gcc.exe
# Файл библиотеки
MYLIBNAME = lib.a
# Параметры компиляции
CFLAGS= -march=rv64iac -mabi=lp64 -O1
# Включаемые файлы следует искать в текущем каталоге
INCLUDES+= -I .
# Make должна искать файлы *.h и *.c в текущей директории
vpath %.h .
vpath %.c .
# Построение объектного файла из исходного текста
# $< = %.c
# $@ = %.o
%.o: %.c
    $(CC) -MD $(CFLAGS) $(INCLUDES) -c $< -o $@
# Чтобы достичь цели "all", требуется построить библиотеку
all: $(MYLIBNAME)
# $^ = (sort.o)
$(MYLIBNAME): sort.o
    $(AR) -rsc $@ $^
```

Makefile для сборки исполняемого файла:

```
# "Фиктивные" цели
.PHONY: all clean
# Файлы для сборки исполнимого файла
OBJS = main.o \
lib.o
#Вызываемые приложения
CC = riscv64-unknown-elf-gcc.exe
# Параметры компиляции
CFLAGS= -march=rv64iac -mabi=lp64 -O1 --save-temps
# Включаемые файлы следует искать в текущем каталоге
INCLUDES+= -I .
# Make должна искать файлы *.c и *.a в текущей директории
vpath %.c .
vpath %.a .
# Чтобы достичь цели "all", требуется собрать исполнимый файл
all: a.out
# Сборка исполнимого файла и удаление мусора
a.out: $(OBJS)
    $(CC) $(CFLAGS) $(INCLUDES) $^
    del *.o *.i *.s *.d
```

Для запуска Makefile воспользуемся программой mingw32-make.exe

```
mingw32-make.exe -f Makelib
```

```
mingw32-make.exe -f Makeapp
```

Попробуем собрать нашу программу с помощью обычного gcc:

```
C:\Users\Ilya\CLionProjects\statistics\lib>gcc main.c sort.c
C:\Users\Ilya\CLionProjects\statistics\lib>a.exe
15 6 9 1 5
1 5 6 9 15
9
```

Рис. 6.3. Результаты

Результат полностью соответствует ожидаемым.

Вывод

В ходе выполнения лабораторной работы была написана функция и тестирующая ее функция на языке C для поиска k-ой порядковой статистики. Далее была выполнена сборка по шагам для RISC-V. Была создана библиотека lib.a, а также make-файлы для её сборки и сборки тестовой программы с использованием библиотеки.