

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 6

Выполнил студент гр. 3530901/00002 _____С.А. Колупаев
(подпись)

Принял старший преподаватель _____Д.С. Степанов
(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 вариант 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 6: Нахождение медианы in-place.

1. Функция на C

Сначала разработаем функцию на C, которая будет реализовывать поиск медианы. Напишем функцию в отдельном файле.

```
#ifndef MEDIAN_H
#define MEDIAN_H

int findMedian(const unsigned *array, size_t size);

#endif
```

Рис.1 Заголовочный файл

```
#include <stddef.h>
#include <stdio.h>
#include "median.h"

static unsigned array[] = { [0]: 2, [1]: 5, [2]: 1, [3]: 6 };
static const size_t length = sizeof(array) / sizeof(array[0]);

int main(void) {
    int i;
    printf(Format: "Input Array: \n");
    for(i = 0; i < length; i++) {
        printf(Format: "%i, ", array[i]);
    }
    int med = findMedian(array, size: length);
    printf(Format: "\nmedian = %i", med);
}
```

Рис.2 Файл main.c

```

#include <stddef.h>
#include "median.h"

int findMedian(const unsigned *array, size_t length) {
    int y = 0;
    int i;
    int j;
    for(i = 0; i < length; i++) {
        int x = 0;
        int z = 0;
        for(j = 0; j < length; j++) {
            if (array[i] > array[j]) {
                x++;
            } else if (array[i] < array[j]) {
                x--;
            } else if (array[i] == array[j]) {
                z++;
            }
        }
        if(z == length) {
            y = z;
        } else if (x == 0) {
            y = array[i];
        } else if (x == 1 || x == -1) {
            y += array[i] / 2;
        }
    }
    return(y);
}

```

Рис.3 Файл findMedian.c

Алгоритм программы:

- 1) Проходимся по всему массиву, сравнивая элементы друг с другом
- 2) В зависимости от сравнения увеличиваем/уменьшаем счётчик
- 3) Подсчитываем счётчик и, если он равен 0(для нечетных массивов)/-1 и 1 (для четных), выводим результат

Результат работы функции:

```
Input Array:  
2, 5, 1, 6,  
median = 3
```

Рис. 4 Результат работы программы

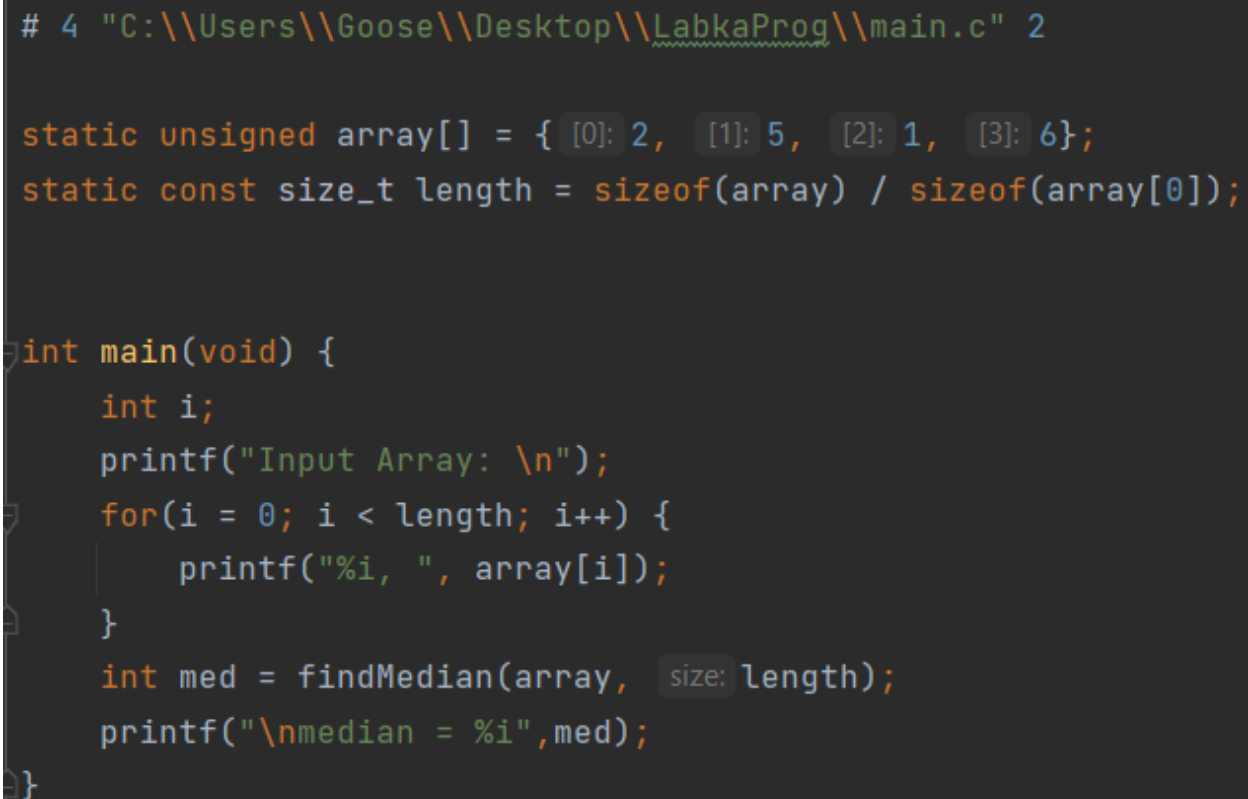
2.Сборка простейшей программы «по шагам»

Препроцессирование

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 -march=rv64iac -mabi=lp64 -O1 -E main.c -o  
main.i -v -E >log_main_pre.txt 2>&1
```

```
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -O1 -E findMedian.c -  
o findMedian.i -v -E >log_findMedian_pre.txt 2>&1
```



```
# 4 "C:\\Users\\Goose\\Desktop\\LabkaProg\\main.c" 2

static unsigned array[] = { [0]: 2, [1]: 5, [2]: 1, [3]: 6};
static const size_t length = sizeof(array) / sizeof(array[0]);

int main(void) {
    int i;
    printf("Input Array: \n");
    for(i = 0; i < length; i++) {
        printf("%i, ", array[i]);
    }
    int med = findMedian(array, size: length);
    printf("\nmedian = %i", med);
}
```

Рис.5. Фрагмент изначального кода в main.i.

```
# 3 "C:\\Users\\Goose\\Desktop\\LabkaProg\\findMedian.c" 2

int findMedian(const unsigned *array, size_t length) {
    int y = 0;
    int i;
    int j;
    for(i = 0; i < length; i++) {
        int x = 0;
        int z = 0;
        for(j = 0; j < length; j++) {
            if (array[i] > array[j]) {
                x++;
            } else if (array[i] < array[j]) {
                x--;
            } else if (array[i] == array[j]) {
                z++;
            }
        }
        if(z == length) {
            y = z;
        } else if (x == 0) {
            y = array[i];
        } else if (x == 1 || x == -1) {
            y += array[i] / 2;
        }
    }
    return(y);
}
```

Рис.6. Фрагмент изначального кода в findMedian.i

Как видно на рис.5 и 6, в созданных файлах после препроцессирования содержится наш изначальный код.

Компиляция

Для выполнения компиляции используем следующие команды:

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

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

Получаем следующие файлы:

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
.LC0:
.string "Input Array: "
.align    3
.LC1:
.string "%i, "
.align    3
.LC2:
.string "\nmedian = %i"
.text
.align    1
.globl    main
.type     main, @function
main:
    addi    sp,sp,-32
    sd     ra,24(sp)
    sd     s0,16(sp)
    sd     s1,8(sp)
    sd     s2,0(sp)
    lui    a0,%hi(.LC0)
    addi    a0,a0,%lo(.LC0)
    call    puts
    lui    s0,%hi(.LANCHOR0)
    addi    s0,s0,%lo(.LANCHOR0)

```

```

        addi    s2,s0,16
        lui    s1,%hi(.LC1)
.L2:
        lw     a1,0(s0)
        addi    a0,s1,%lo(.LC1)
        call    printf
        addi    s0,s0,4
        bne    s0,s2,.L2
        li     a1,4
        lui    a0,%hi(.LANCHOR0)
        addi    a0,a0,%lo(.LANCHOR0)
        call    findMedian
        mv     a1,a0
        lui    a0,%hi(.LC2)
        addi    a0,a0,%lo(.LC2)
        call    printf
        li     a0,0
        ld     ra,24(sp)
        ld     s0,16(sp)
        ld     s1,8(sp)
        ld     s2,0(sp)
        addi    sp,sp,32
        jr     ra
        .size   main, .-main
        .data
        .align  3
        .set    .LANCHOR0,. + 0
        .type   array, @object
        .size   array, 16
array:
        .word   2

```

```
array:
    .word    2
    .word    5
    .word    1
    .word    6
    .ident   "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"
```

II findMedian.s:

```
.file    "findMedian.c"
.option  nopic
.attribute arch, "rv64i2p0_a2p0_c2p0"
.attribute unaligned_access, 0
.attribute stack_align, 16
.text
.align   1
.globl   findMedian
.type    findMedian, @function
findMedian:
    mv    t4,a0
    beq   a1,zero,.L10
    slli   a6,a1,2
    add   a6,a0,a6
    mv    a7,a0
    li    a0,0
    li    t3,0
    j     .L3
.L4:
    bgeu   a3,a4,.L6
    addiw  a2,a2,-1
.L5:
    addi   a5,a5,4
    beq    a5,a6,.L13
.L7:
    lw     a4,0(a5)
    bleu   a3,a4,.L4
    addiw  a2,a2,1
    j     .L5
.L6:
    bne    a3,a4,.L5
```

```

    addiw    t1,t1,1
    j      .L5
.L13:
    beq t1,a1,.L11
    bne a2,zero,.L9
    sxt.w    a0,a3
    j      .L8
.L9:
    addiw    a5,a2,1
    andi     a5,a5,-3
    sxt.w    a5,a5
    bne a5,zero,.L8
    srlw     a3,a3,1
    addw     a0,a3,a0
    j      .L8
.L11:
    mv     a0,t1
.L8:
    addi     a7,a7,4
    beq a7,a6,.L2
.L3:
    lw     a3,0(a7)
    mv     a5,t4
    mv     t1,t3
    mv     a2,t3
    j      .L7
.L10:
    li     a0,0
.L2:
    ret
    .size   findMedian, .-findMedian

```

Получаем инструкции на RISC-V.

Ассемблирование

Используем следующие команды:

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

```
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -v -c findMedian.s -o  
findMedian.o >log_o.txt 2>&1
```

После их выполнения получаем объектные файлы main.o и findMedian.o.
Для их прочтения будем использовать следующие команды:

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

```
riscv64-unknown-elf-objdump -h findMedian.o
```

```
C:\Users\Goose\Desktop\Proga4Lab\bin>riscv64-unknown-elf-objdump -h C:\Users\Goose\Desktop\LabkaProg\main.o  
C:\Users\Goose\Desktop\LabkaProg\main.o:      file format elf64-littleriscv  
Sections:  
Idx Name          Size      VMA           LMA           File off  Algn  
  0 .text          00000070  0000000000000000  0000000000000000  00000040  2**1  
CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE  
  1 .data          00000010  0000000000000000  0000000000000000  000000b0  2**3  
CONTENTS, ALLOC, LOAD, DATA  
  2 .bss           00000000  0000000000000000  0000000000000000  000000c0  2**0  
ALLOC  
  3 .rodata.str1.8  00000025  0000000000000000  0000000000000000  000000c0  2**3  
CONTENTS, ALLOC, LOAD, READONLY, DATA  
  4 .comment        00000031  0000000000000000  0000000000000000  000000e5  2**0  
CONTENTS, READONLY  
  5 .riscv.attributes 00000026  0000000000000000  0000000000000000  00000116  2**0  
CONTENTS, READONLY
```

Рис.7. Хедер файла main.o.

```
C:\Users\Goose\Desktop\LabkaProg\findMedian.o:      file format elf64-littleriscv  
Sections:  
Idx Name          Size      VMA           LMA           File off  Algn  
  0 .text          00000066  0000000000000000  0000000000000000  00000040  2**1  
CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE  
  1 .data          00000000  0000000000000000  0000000000000000  000000a6  2**0  
CONTENTS, ALLOC, LOAD, DATA  
  2 .bss           00000000  0000000000000000  0000000000000000  000000a6  2**0  
ALLOC  
  3 .comment        00000031  0000000000000000  0000000000000000  000000a6  2**0  
CONTENTS, READONLY  
  4 .riscv.attributes 00000026  0000000000000000  0000000000000000  000000d7  2**0  
CONTENTS, READONLY
```

Рис.8 Хедер файла findMedian.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

Далее вводим команду:

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

И можем более подробно рассмотреть секцию .text.

```

0000000000000000 <main>:
  0: 1101          c.addi    sp,-32
  2: ec06          c.sdsp    ra,24(sp)
  4: e822          c.sdsp    s0,16(sp)
  6: e426          c.sdsp    s1,8(sp)
  8: e04a          c.sdsp    s2,0(sp)
 a: 00000537      lui      a0,0x0
 e: 00050513      addi     a0,a0,0 # 0 <main>
12: 00000097      auipc    ra,0x0
16: 000080e7      jalr     ra,0(ra) # 12 <main+0x12>
1a: 00000437      lui      s0,0x0
1e: 00040413      addi     s0,s0,0 # 0 <main>
22: 01040913      addi     s2,s0,16
26: 000004b7      lui      s1,0x0

000000000000002a <.L2>:
2a: 400c          c.lw      a1,0(s0)
2c: 00048513      addi     a0,s1,0 # 0 <main>
30: 00000097      auipc    ra,0x0
34: 000080e7      jalr     ra,0(ra) # 30 <.L2+0x6>
38: 0411          c.addi    s0,4
3a: ff2418e3      bne      s0,s2,2a <.L2>
3e: 4591          c.li      a1,4
40: 00000537      lui      a0,0x0
44: 00050513      addi     a0,a0,0 # 0 <main>
48: 00000097      auipc    ra,0x0
4c: 000080e7      jalr     ra,0(ra) # 48 <.L2+0x1e>
50: 85aa          c.mv      a1,a0
52: 00000537      lui      a0,0x0
56: 00050513      addi     a0,a0,0 # 0 <main>
5a: 00000097      auipc    ra,0x0
5e: 000080e7      jalr     ra,0(ra) # 5a <.L2+0x30>
62: 4501          c.li      a0,0
64: 60e2          c.ldsp    ra,24(sp)
66: 6442          c.ldsp    s0,16(sp)
68: 64a2          c.ldsp    s1,8(sp)
6a: 6902          c.ldsp    s2,0(sp)
6c: 6105          c.addi16sp      sp,32
6e: 8082          c.jr      ra

```

Рис.9. Секция .text файла main.o.

Командой: *riscv64-unknown-elf-objdump -t findMedian.o main.o*

Получаем таблицу символов.

```

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   .data  0000000000000000 .LANCHOR0
0000000000000000 1   0  .data  0000000000000010 array
0000000000000000 1   .rodata.str1.8 0000000000000000 .LC0
0000000000000010 1   .rodata.str1.8 0000000000000000 .LC1
0000000000000018 1   .rodata.str1.8 0000000000000000 .LC2
000000000000002a 1   .text  0000000000000000 .L2
0000000000000000 1   d  .comment 0000000000000000 .comment
0000000000000000 1   d  .riscv.attributes 0000000000000000 .riscv.attributes
0000000000000000 g   F  .text  0000000000000070 main
0000000000000000      *UND*  0000000000000000 puts
0000000000000000      *UND*  0000000000000000 printf
0000000000000000      *UND*  0000000000000000 findMedian

```

Рис.10. Таблица символов

```

RELOCATION RECORDS FOR [.text]:
OFFSET                TYPE                VALUE
000000000000000a R_RISCV_HI20        .LC0
000000000000000a R_RISCV_RELAX       *ABS*
000000000000000e R_RISCV_LO12_I      .LC0
000000000000000e R_RISCV_RELAX       *ABS*
0000000000000012 R_RISCV_CALL        puts
0000000000000012 R_RISCV_RELAX       *ABS*
000000000000001a R_RISCV_HI20        .LANCHOR0
000000000000001a R_RISCV_RELAX       *ABS*
000000000000001e R_RISCV_LO12_I      .LANCHOR0
000000000000001e R_RISCV_RELAX       *ABS*
0000000000000026 R_RISCV_HI20        .LC1
0000000000000026 R_RISCV_RELAX       *ABS*
000000000000002c R_RISCV_LO12_I      .LC1
000000000000002c R_RISCV_RELAX       *ABS*
0000000000000030 R_RISCV_CALL        printf
0000000000000030 R_RISCV_RELAX       *ABS*
0000000000000040 R_RISCV_HI20        .LANCHOR0
0000000000000040 R_RISCV_RELAX       *ABS*
0000000000000044 R_RISCV_LO12_I      .LANCHOR0
0000000000000044 R_RISCV_RELAX       *ABS*
0000000000000048 R_RISCV_CALL        findMedian
0000000000000048 R_RISCV_RELAX       *ABS*
0000000000000052 R_RISCV_HI20        .LC2
0000000000000052 R_RISCV_RELAX       *ABS*
0000000000000056 R_RISCV_LO12_I      .LC2
0000000000000056 R_RISCV_RELAX       *ABS*
000000000000005a R_RISCV_CALL        printf
000000000000005a R_RISCV_RELAX       *ABS*
000000000000003a R_RISCV_BRANCH      .L2

```

Рис.11. Таблица перемещений.

Здесь можно найти записи типа R_RISCV_CALL – сообщения для компоновщика о возможных оптимизациях.

Компоновка

Выполняем компоновку командой:

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

```
findMedian.o -o main.out >log_out.txt 2>&1
```

Мы получили файл main.out, для рассмотрения его секции, используем:

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

```
C:\Users\Goose\Desktop\LabkaProg\main.out:      file format elf64-littleriscv

Disassembly of section .text:

0000000000100b0 <register_fini>:
  100b0: 00000793          addi    a5,zero,0
  100b4: c799             c.beqz  a5,100c2 <register_fini+0x12>
  100b6: 00002517          auipc   a0,0x2
  100ba: 7e850513          addi    a0,a0,2024 # 1289e <__libc_fini_array>
  100be: 4830906f          jal     zero,19d40 <atexit>
  100c2: 8082             c.jr    ra

0000000000100c4 <_start>:
  100c4: 0000f197          auipc   gp,0xf
  100c8: 4bc18193          addi    gp,gp,1212 # 1f580 <__global_pointer$>
  100cc: 76818513          addi    a0,gp,1896 # 1fce8 <_PathLocale>
  100d0: 00010617          auipc   a2,0x10
  100d4: cb060613          addi    a2,a2,-848 # 1fd80 <__BSS_END__>
  100d8: 8e09             c.sub   a2,a0
  100da: 4581             c.li    a1,0
  100dc: 1c2000ef          jal     ra,1029e <memset>
  100e0: 0000a517          auipc   a0,0xa
  100e4: c6050513          addi    a0,a0,-928 # 19d40 <atexit>
  100e8: c519             c.beqz  a0,100f6 <_start+0x32>
  100ea: 00002517          auipc   a0,0x2
  100ee: 7b450513          addi    a0,a0,1972 # 1289e <__libc_fini_array>
  100f2: 44f090ef          jal     ra,19d40 <atexit>
  100f6: 13e000ef          jal     ra,10234 <__libc_init_array>
  100fa: 4502             c.lwsp  a0,0(sp)
```

Рис.12. Исполняемый файл.

```

000000000001cbb2 g      F .text 0000000000000030 .hidden __clzdi2
00000000000104b2 g      F .text 0000000000001cd2 _vfprintf_r
0000000000015f04 g      F .text 0000000000000098 __lo0bits
000000000001b482 g      F .text 0000000000000088 __sigtramp
000000000001b0e0 g      F .text 0000000000000052 wctomb
0000000000016dd4 g      F .text 0000000000000086 frexp
000000000001f2d8 g      O .data 00000000000001a8 __global_locale
0000000000012184 g      F .text 000000000000000e vfprintf
000000000001c9a8 g      F .text 000000000000013c .hidden __trunctfdf2
000000000001a0ca g      F .text 000000000000004e fputwc
000000000001b35c g      F .text 000000000000007c raise
000000000001b540 g      F .text 000000000000002c _close
000000000001287e g      F .text 0000000000000002 __sinit_lock_acquire
0000000000015cc8 g      F .text 0000000000000100 __multadd
0000000000015cb6 g      F .text 0000000000000012 _Bfree

```

Рис.13. Исполняемый файл.

По рисункам 12 и 13 видим, что адресация изменилась на абсолютную.

3.Создание статической библиотеки

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.

Создаем объектный файл `findMedian.o` и собираем в библиотеку командами:

```
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -O1 -c findMedian.c -o findMedian.o
```

```
riscv64-unknown-elf-ar -rsc libStatistics.a findMedian.o
```

И можем получить список символов `libStatistics.a`:

```
riscv64-unknown-elf-nm libStatistics.a
```

```

findMedian.o:
00000000000000062 t .L10
0000000000000004e t .L11
00000000000000030 t .L13
00000000000000064 t .L2
00000000000000056 t .L3
00000000000000012 t .L4
00000000000000018 t .L5
00000000000000028 t .L6
0000000000000001e t .L7
00000000000000050 t .L8
0000000000000003c t .L9
0000000000000000 T findMedian

```

Рис.14. Список символов файла libStatistics.a.

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

Сделаем исполняемый файл тестовой программы:

```

riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -O1 main.c
libStatistics.a -o main.out

```

Для прочтения используем:

```

riscv64-unknown-elf-objdump -t main.out >main.ds

```

305	000000000000159b	g	F .text	0000000000000040	__ascii_mbtowc
306	0000000000001c296	g	F .text	000000000000005d0	.hidden __subtfc3
307	000000000000165e4	g	F .text	00000000000000056	__ulp
308	00000000000012890	g	F .text	0000000000000000e	__fp_unlock_all
309	000000000000101b0	g	F .text	00000000000000066	findMedian
310	0000000000001513c	g	F .text	00000000000000006	localeconv
311	0000000000001530e	g	F .text	00000000000000082	__swatbuf_r
312	0000000000001eb80	g	.data	00000000000000000	__DATA_BEGIN__
313	0000000000001b6b4	g	F .text	0000000000000002c	_write

Рис.15. Таблица символов main.out.

Видим, что в эту таблицу входит содержание объектного файла findMedian.o

Создание make-файлов

Используя примеры с сайта курса, было написано 2 make-файла.

```
# "Фиктивные" цели
.PHONY: all
# файлы для сборки исполняемого файла
objs = main.c findCoeffs.c
CC = riscv64-unknown-elf-gcc
AR = riscv64-unknown-elf-ar
# Параметры компиляции
CFLAGS = -march=rv64iac -mabi=lp64 -O1 -c
# Включаемые файлы следует искать в текущем каталоге
INCLUDES+= -I .
# Утилита make должна искать файлы *.c и *.h в текущей директории
vpath %.c .
vpath %.h .
# Сборка объектных файлов
%.o : %.c
    $(CC) $(CFLAGS) $(INCLUDES) $< -o $@
all : libFindCoeffs.a
libFindCoeffs.a : findCoeffs.o
    $(AR) -rsc $@ $<
    del *.o *.i *.s
```

Рис 16. Содержание файла Makelib

```
# "Фиктивные" цели
.PHONY: all
# файлы для сборки исполняемого файла
objs = main.c libFindCoeffs.a
CC = riscv64-unknown-elf-gcc
# Параметры компиляции
CFLAGS = -march=rv64iac -mabi=lp64 -O1
# Включаемые файлы следует искать в текущем каталоге
INCLUDES+= -I .
# Утилита make должна искать файлы *.c и *.a в текущей директории
vpath %.c .
vpath %.a .
all: main.out
# Сборка исполняемого файла и удаление промежуточных файлов
a.out: $(objs)
    $(CC) $(CFLAGS) $(INCLUDES) $^
    del *.o *.i *.s
```

Рис 17. Содержание файла Makefile

```
C:\Users\Goose\Desktop\SiFive\bin>C:\MinGW\bin\mingw32-make.exe -f C:\Users\Goose\Desktop\SiFive\bin\Makelib  
riscv64-unknown-elf-gcc.exe -MD -O1 -I . -c findMedian.c -o findMedian.o  
riscv64-unknown-elf-ar.exe -rsc libMedian.a findMedian.o
```

Рис.18. Выполнение файла Makelib

Вывод

В ходе выполнения лабораторной работы, была сделана функция и тестирующая ее функция на языке C для нахождения медианны массива. Далее была выполнена сборка по шагам для RISC-V. Была создана библиотека libMedian.a, а также make-файлы для её сборки и сборки тестовой программы с использованием библиотеки.