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

Вариант 6.

Выполнил студент гр. 3530901/90004 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Р.И.Галиева

(подпись)

Принял преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.О. Алексюк

(подпись)

“ ” 2021 г.

Санкт-Петербург

2021

**Задача**

1. 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.
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 задания**

По варианту номер 6 необходимо реализовать нахождение медианы массива in-place.

**Решение**

Медиана – элемент массива, который находится ровно посередине после сортировки. Для выполнения поставленной задачи программа проходится по массиву, сравнивая каждое число с каждым, прибавляя к счетчику 1, если второе число больше текущего; вычитая 1, если число меньше текущего и не делая ничего, если они равны. В конце каждого прохода проверяется значение счетчика, если он меньше 0 или больше 1 (количество чисел больше текущего меньше, чем количество чисел меньше текущего и наоборот, соответственно), то программа за место текущего числа берет следующее из массива и начинает проход заново, если значение счетчика равно 0 или 1 (0, когда в массиве нечетное количество элементов и 1, когда - четное), то медиана найдена и программа, записав ее, завершает работу.

1. **Написание программы на языке С**

Была написана программа, которая находит медиану массива. Функция помещена в отдельный файл, написан заголовочный файл.

#ifndef MEDIAN\_H  
#define MEDIAN\_H  
int median(unsigned \*array, size\_t size);  
  
#endif

Рис.1 Заголовочный файл

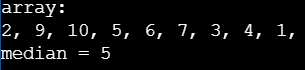
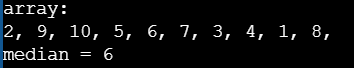
#include <stddef.h>  
#include <stdio.h>  
#include "median.h"  
  
static unsigned array[] = {2, 9, 10, 5, 6, 7, 3, 4, 1, 8};  
static const size\_t array\_length = sizeof(array) / sizeof(array[0]);  
  
int main(void) {  
 printf("array: \n");  
 for (size\_t i = 0; i < array\_length; i++) {  
 printf("%i, ", array[i]);  
 }  
 printf("\n");  
 int temp = median(array, array\_length);  
 printf("median = %i", temp);  
 return 0;  
}

Рис.2 Текст программы

#include <stddef.h>  
#include "median.h"  
  
int median(unsigned \*array, size\_t array\_length)  
{  
 int count = 0;  
 for (size\_t i = 0; i < array\_length; i++)  
 {  
 for (size\_t j = 0; j < array\_length; j++)  
 {  
 if (array[i] > array[j])  
 {  
 count++;  
 }  
 if (array[i] < array[j])  
 {  
 count--;  
 }  
 }  
 if ((count == 0) | (count == 1))  
 {  
 return(array[i]);  
 break;  
 }  
 count = 0;  
 }  
 return -1;  
}

Рис.3Функция median

Произведем компиляцию программы:



а) б)

Рис. 4 Результат исполнения программы(а)n-четное б) n-нечетное)

1. **Сборка программы «по шагам»**

**Препроцессирование.**

Выполним препроцессирование файлов с помощью пакета разработки «SiFive GNU Embedded Toolchain» для RISK-V. Для этого необходимо выполнить следующие команды:

riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -O1 -v -E main.c -o main.i>log\_main\_prepr.txt 2>&1

riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -O1 -v -E median.c -o median.i >log\_median\_prepr.txt 2>&1

**-march=rv64iac -mabi=lp64** – объявляем, что целевым является процессор с базовой архитектурой системы команд RV64IAC

**-O1**–выполняются простые оптимизации генерируемого кода

**-v**–печатаются (в стандартный поток ошибок) выполняемые драйвером команды, а также дополнительную информацию

**>**- печатать в файл

**-E**– обработка файлов будет выполнятся только препроцессором

Рассмотрим результаты препроцессирования.

|  |  |
| --- | --- |
|  |  |
| Рис. 5 Файл main.i | Рис. 6 Файл median.i |

main.i – текстовый файл, содержащий результат препроцессирования файла “main.c” .

В связи с тем, что в файле тестовой программы мы использовали стандартную библиотеку языка C “stdio.h” для вывода на консоль значений массива, результирующий файл препроцессирования имеет много добавочных строк.

**Компиляция**

Выполним компиляцию файлов с помощью пакета разработки «SiFive GNU Embedded Toolchain» для RISK-V. Для этого необходимо выполнить следующие команды:

riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -O1 -v -S –fpreprocessed median.i -omedian.s>log\_median\_comp.txt 2>&1

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

На данном шаге выполняется компиляция файла “main.i”, уже обработанного препроцессором (опция “-fpreprocessed”). Получаем код на языке ассемблер.

.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 "array: "  
 .align 3  
.LC1:  
 .string "%i, "  
 .align 3  
.LC2:  
 .string "median = %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,s0,40  
 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 a0,10  
 call putchar  
 li a1,10  
 lui a0,%hi(.LANCHOR0)  
 addi a0,a0,%lo(.LANCHOR0)  
 call median  
 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, 40  
array:  
 .word 2  
 .word 9  
 .word 10  
 .word 5  
 .word 6  
 .word 7  
 .word 3  
 .word 4  
 .word 1  
 .word 8  
 .ident "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"

Рис. 7 файл main.s

.file "median.c"  
 .option nopic  
 .attribute arch, "rv64i2p0\_a2p0\_c2p0"  
 .attribute unaligned\_access, 0  
 .attribute stack\_align, 16  
 .text  
 .align 1  
 .globl median  
 .type median, @function  
median:  
 beq a1,zero,.L8  
 slli a6,a1,2  
 add a6,a6,a0  
 mv t1,a0  
 li a7,0  
 li t4,0  
 li t3,1  
 j .L3  
.L5:  
 addi a5,a5,4   
 beq a5,a6,.L11   
.L6:  
 lw a4,0(a5)  
 bleu a2,a4,.L4   
 addiw a3,a3,1   
.L4:  
 bgeu a2,a4,.L5   
 addiw a3,a3,-1   
 j .L5  
.L11:  
 sext.w a3,a3  
 bleu a3,t3,.L12  
 addi a7,a7,1  
 addi t1,t1,4  
 beq a1,a7,.L9  
.L3:  
 lw a2,0(t1)  
 mv a5,a0  
 mv a3,t4  
 j .L6  
.L12:  
 slli a7,a7,2  
 add a7,a0,a7  
 lw a0,0(a7)  
 ret  
.L8:  
 li a0,-1  
 ret  
.L9:  
 li a0,-1  
 ret  
 .size median, .-median  
 .ident "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"

Рис. 8 файл median.s

main.s – текстовый файл, содержащий код на языке ассемблера, сгенерированный компилятором в результате обработки файла “main.i”

**Объектный файл**

Выполним ассемблирование для получения объектных файлов программы.

main.o – объектный файл (object file), сгенерированный ассемблером в результате обработки файла “main.s”.

Для этого исполним следующие команды:

riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -v -c main.s -o main.o >log\_obj\_main.txt 2>&1

riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -v -c median.s -o median.o >log\_obj\_median.txt 2>&1

Сформированный ассемблером объектный файл “main.o” содержит коды инструкций, таблицу символов и таблицу перемещений (relocations). В отличие от ранее рассмотренных файлов, объектный файл не является текстовым, для изучения его содержимого используем утилиту objdump, отображающую содержимое бинарных файлов в текстовом виде. Как известно, содержательная часть объектного файла разбита на «разделы», называемые обычно секциями (section). Следующая команда обеспечивает отображение заголовков секций файла “main.o”:

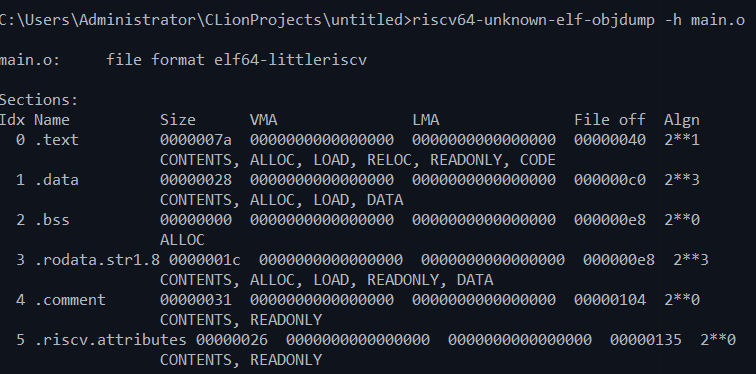


Рис. 9 Файл main.o.

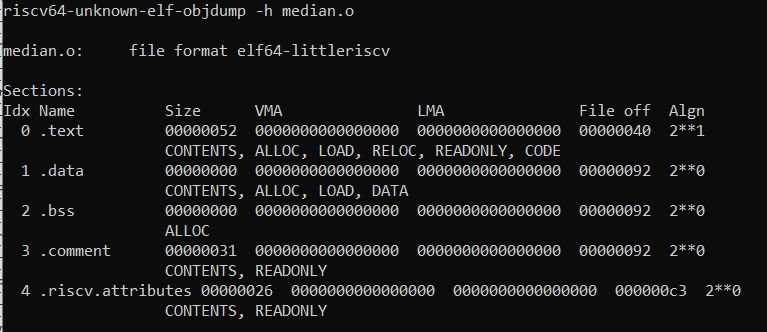


Рис. 10 Файл median.o.

.text – секция кода, содержащая коды инструкций

.data – секция инициализированных данных

.bss – секция данных, инициализированных нулями

.comment – секция данных о версиях

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

Опция “-d” инициирует процесс дизассемблирования (disassemble),

опция “-M no-aliases” требует использовать в выводе только инструкции системы команд (но не псевдоинструкции ассемблера).

Вывод утилиты:

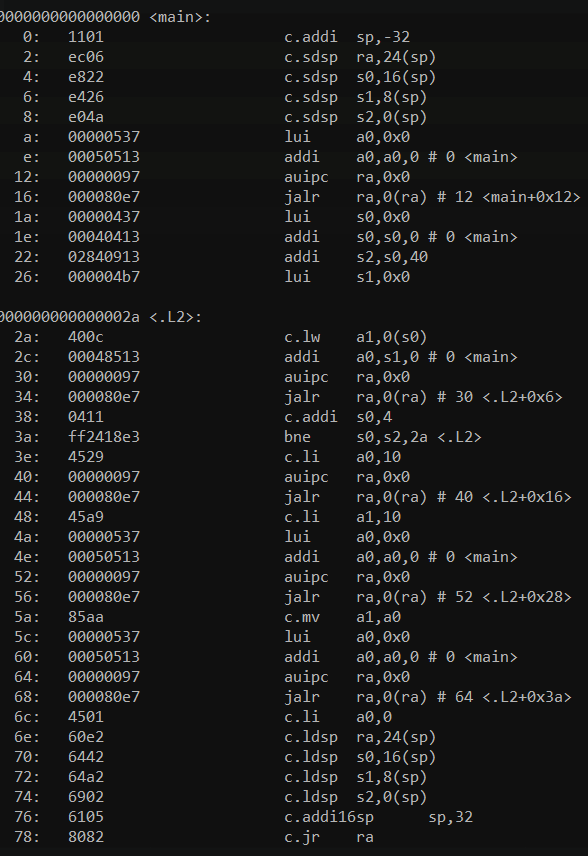


Рис. 11 дизассемблированный файл main.o

Рассмотрим таблицу символов, выполнив команду:

Командой objdump с флагом -t получаем список символов для объектных файлов

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

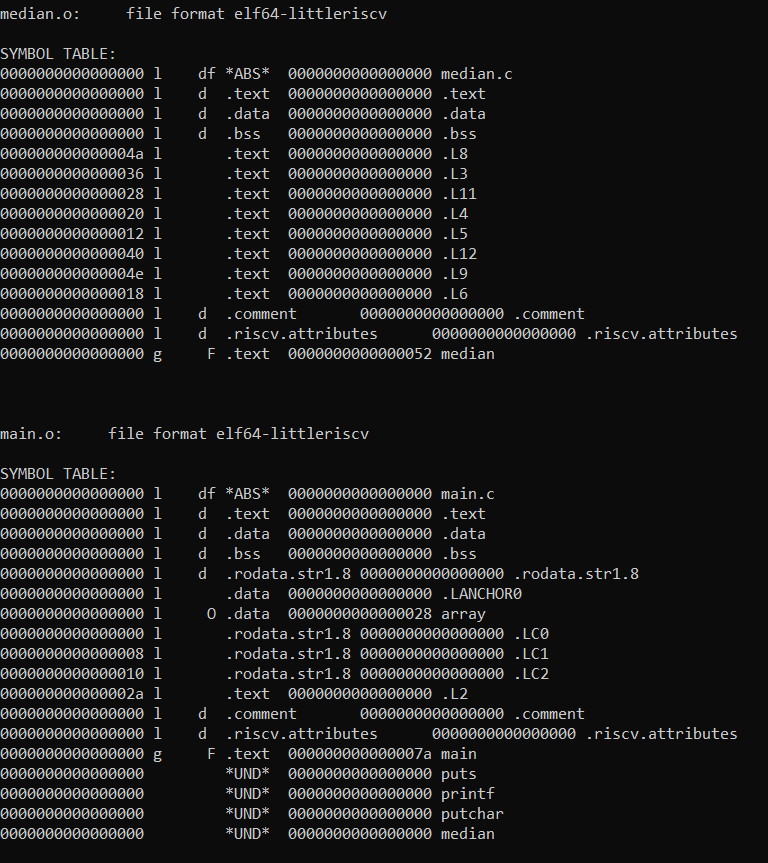


Рис. 12 Таблица символов.

В таблице символов“main.o” имеется интересная запись: символ “zero” типа “\*UND\*” (undefined – не определен).Эта запись означает, что символ “zero” 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, получить которую можно по команде:

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

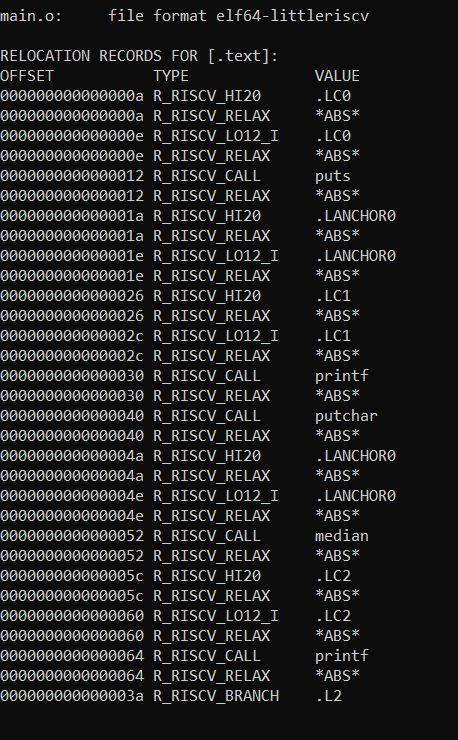
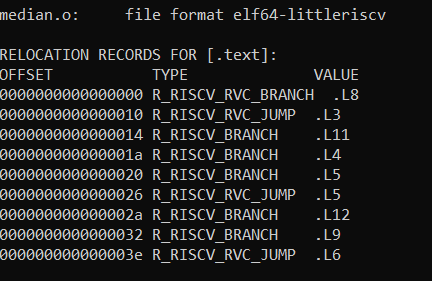


Рис. 13 Таблица перемещений

В таблице перемещений для main.o наблюдаем вызов метода median. 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.

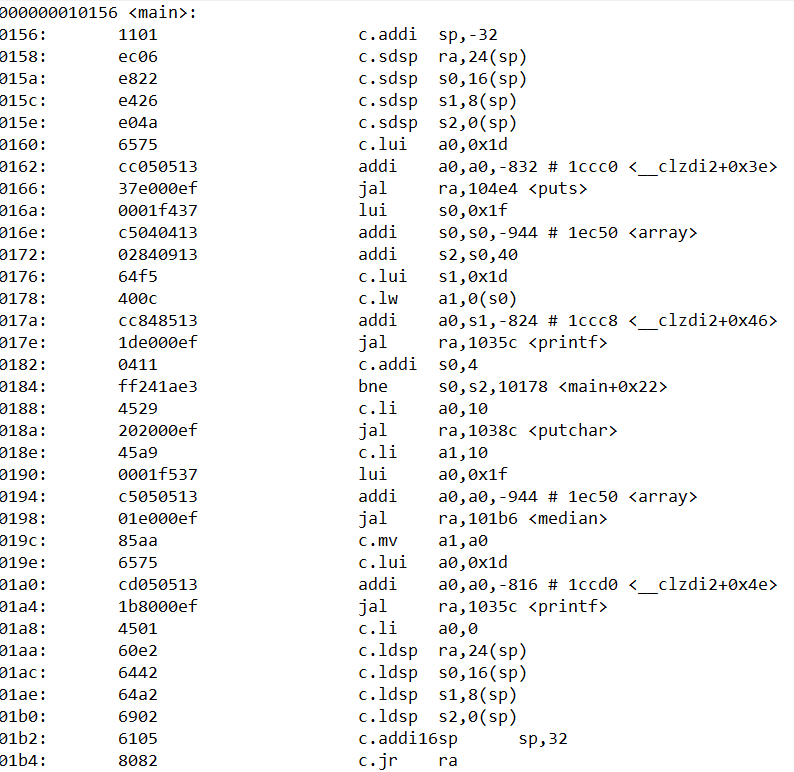
**Компоновка**

Выполним компоновку командой

riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 –v main.o median.o -omain.out>log\_out.txt 2>&1

В результате выполнения этой команды был получен файл main.out – исполняемый бинарный файл. Рассмотрим его секцию кода с помощью команды:

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



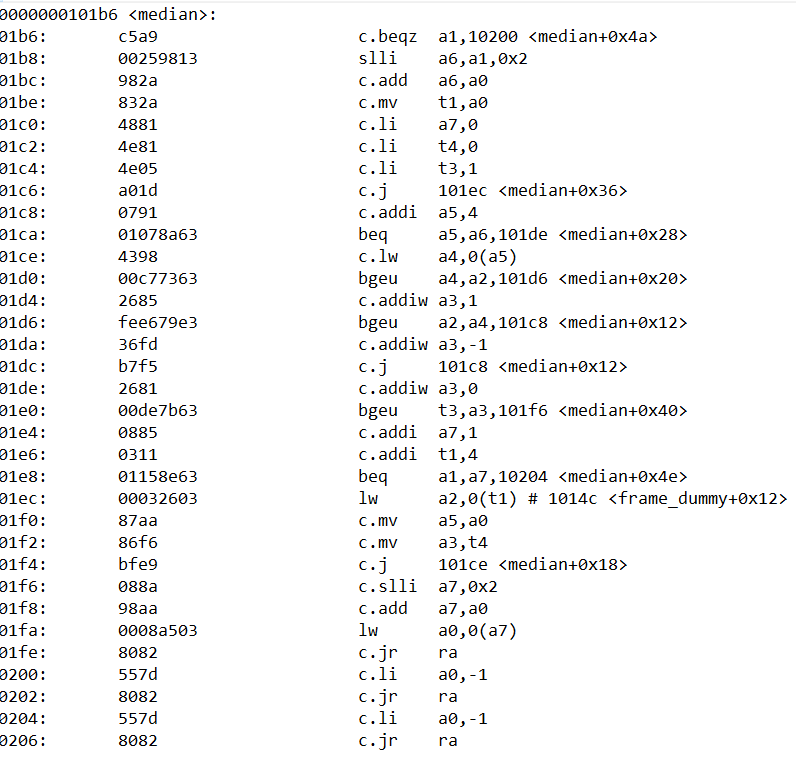


Рис. 14 содержимое файла a.ds

1. **Создание статической библиотеки**

Выделим функцию median в отдельную статическую библиотеку. Для этого надо получить объектный файл median.o и собрать библиотеку.

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

riscv64-unknown-elf-ar -rsc libMedian.a median.o

Рассмотрим список символов libMedian.a с помощью команды

riscv64-unknown-elf-nm libMedian.a

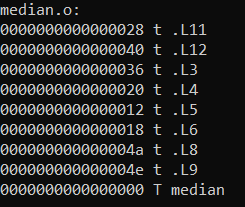


Рис. 15 Cписок символов

В выводе утилиты “nm” кодом “T” обозначаются символы, определенные в соответствующем объектном файле. Используя собранную библиотеку, произведём исполняемый файл тестовой программы с помощью команды:

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

Посмотрим содержимое таблицы символов исполняемого файла с помощью команды:

riscv64-unknown-elf-objdump -t main.out >main.ds и убедимся, что там есть функция median

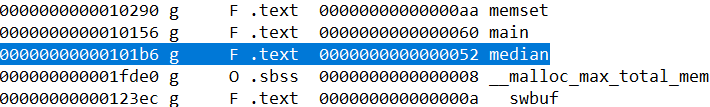
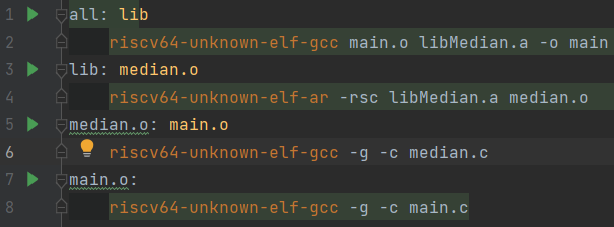


Рис. 16 Таблица символов исполняемого файла

Создание make-файлов

Чтобы автоматизировать процесс сборки библиотеки и приложения напишем make-файл.



**Вывод:**

В ходе выполнения работы была написана программа на языке С, была выполнена сборка этой программы по шагам для архитектуры RISC-V, была создана статическая библиотека. Также были написаны два make-файла для автоматизированной сборки библиотеки и тестовой программы.