Санкт-Петербургский политехнический университет Петра Великого

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Отчет по лабораторной работе №4**

**Дисциплина:** Низкоуровневое программирование

**Тема:** Раздельная компиляция

Выполнил

студент гр. 3530901/90003 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Руднев А.К.

(подпись)

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Алексюк А.О.

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Санкт-Петербург   
2021

**Оглавление**

[**1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ** 3](#_Toc69842835)

[**2. МЕТОД РЕШЕНИЯ** 3](#_Toc69842836)

[**3 РЕШЕНИЕ** 3](#_Toc69842837)

[**3.1 ПРОГРАММА НА С** 3](#_Toc69842838)

[**3.2 АНАЛИЗ ВЫХОДА ПРЕПРОЦЕССОРА** 4](#_Toc69842839)

[**3.3 АНАЛИХ ВЫХОДА КОМПИЛЯТОРА** 5](#_Toc69842840)

[**3.4 АНАЛИХ СОСТАВА И СОДЕРЖИМОГО ТАБЛИЦЫ СИМВОЛОВ, ТАБЛИЦЫ ПЕРЕМЕЩЕНИЙ И ОТЛАДОЧНОЙ ИНФОРМАЦИИ, СОДЕРЖАЩИХСЯ В ИСПОЛНИТЕЛЬНОМ И ОБЪЕКТНЫХ ФАЙЛАХ** 8](#_Toc69842841)

[**3.5 СОДЕРЖИМОЕ ТАБЛИЦЫ ПЕРЕМЕЩЕНИЙ** 12](#_Toc69842842)

[**3.6 РЕЗУЛЬТАТ КОМПОНОВКИ** 13](#_Toc69842843)

[**3.7 АНАЛИЗ ОТЛАДОЧНОЙ ИНФОРМАЦИИ** 15](#_Toc69842844)

[**3.8 ВЫДЕЛЕНИЕ РАЗРАБОТАННОЙ ФУНКЦИИ В СТАТИЧСЕКУЮ БИБЛИОТЕКУ** 17](#_Toc69842845)

[**4. РЕЗУЛЬТАТЫ** 18](#_Toc69842846)

# **1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

1) На языке программирования C разработать функцию, которая выполняет поиск медианы массива in-place. Поместить определение функции в отдельный исходный файл и оформить его. 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 и исполняемом файлах.

3) Выделить разработанную функцию в статическую библиотеку. Разработать make-файлы для сборки библиотеки и использующей её тестовой программы. Проанализировать хорд сборки библиотеки и программы, созданные файлы зависимостей.

# **2. МЕТОД РЕШЕНИЯ**

1) Берем i элемент массива и поочередно вычитаем из него другие элементы массива

2) Если при вычитании получилось отрицательное число, то min + 1, иначе max + 1.

3) Если max = min, то медианы найдена.

4) Изменяем i, прибавляя единицу и возвращаемся к пункту 1

5) Если размер массива – значение четное, то выводит наибольшая медиана.

# **3 РЕШЕНИЕ**

## **3.1 ПРОГРАММА НА С**

Напишу программу на языке C (рис. 1-3):

|  |
| --- |
| #ifndef LOWLEVEL\_MEDIAN\_H #define LOWLEVEL\_MEDIAN\_H #include <stdio.h>  int Median(int a[], int length);  #endif //LOWLEVEL\_MEDIAN\_H  Рис.1 - Median.h |
| #include "Median.h"  int Median(int a[], int length) {  int result = 0;  for(int i = 0; i < length; i++) {  int smaller = 0;  int bigger = 0;  for (int j = 0; j < length; j++) {  if (a[i] - a[j] > 0) bigger++;  if (a[i] - a[j] < 0) smaller++;  }  if ((bigger - smaller <= 1) &&  (bigger - smaller >= -1)) result = a[i];  }  return result; }  Рис. 2 - Median.c  #include <stdio.h> #include "Median.h"  int main() {  int a[5] = {0, 2, 4, 1, 3};  int length = sizeof(a) / sizeof(int);  printf("Array: [");  for (int i = 0; i < length; i++) {  printf("%d", a[i]);  if (i != length - 1) printf(", ");  }  printf("]\n");  printf("result=%d", Median(a, length)); }  Рис. 3 - main.c |

Проведу сборку программы по этапам (рис. 4). Для этого использовался дистрибутив «SiFive».

|  |
| --- |
| Рисунок 4 – Сборка программы «по шагам» |

## **3.2 АНАЛИЗ ВЫХОДА ПРЕПРОЦЕССОРА**

В заголовочном файле прописаны директивы, которые определяют вставку стандартной библиотеки ввода-вывода языка C. Пользовательская часть кода практически не изменяется. Часть файла main.i приведена на рисунке 5.

|  |
| --- |
| Рисунок 5 – Пользовательская часть кода main.i выхода препроцессора |

Также происходит и с функцией Median.h. Пользовательская часть кода Median.i практически не изменяется. Пользовательская часть кода выхода препроцессора приведена на рисунке 6.

|  |
| --- |
| Рисунок 6 – Пользовательская часть кода Median.i выхода препроцессора |

## **3.3 АНАЛИХ ВЫХОДА КОМПИЛЯТОРА**

|  |
| --- |
| .file "main.c"  .option nopic  .attribute arch, "rv64i2p0\_m2p0\_a2p0\_f2p0\_d2p0\_c2p0"  .attribute unaligned\_access, 0  .attribute stack\_align, 16  .text  .section .rodata.str1.8,"aMS",@progbits,1  .align 3  .LC1:  .string "Array: ["  .align 3  .LC2:  .string "%d"  .align 3  .LC3:  .string ", "  .align 3  .LC4:  .string "]"  .align 3  .LC5:  .string "result=%d"  .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)  lui a0,%hi(.LC1)  addi a0,a0,%lo(.LC1)  call printf  lw a1,8(sp)  lui a0,%hi(.LC2)  addi a0,a0,%lo(.LC2)  call printf  addi s0,sp,8  addi s3,sp,24  lui s2,%hi(.LC3)  lui s1,%hi(.LC2)  .L2:  addi a0,s2,%lo(.LC3)  call printf  lw a1,4(s0)  addi a0,s1,%lo(.LC2)  call printf  addi s0,s0,4  bne s0,s3,.L2  lui a0,%hi(.LC4)  addi a0,a0,%lo(.LC4)  call puts  li a1,5  addi a0,sp,8  call Median  mv a1,a0  lui a0,%hi(.LC5)  addi a0,a0,%lo(.LC5)  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 0  .word 2  .word 4  .word 1  .word 3  .ident "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"  Рисунок 7 – Выход компилятора файла main.s |
| .file "Median.c"  .option nopic  .attribute arch, "rv64i2p0\_m2p0\_a2p0\_f2p0\_d2p0\_c2p0"  .attribute unaligned\_access, 0  .attribute stack\_align, 16  .text  .align 1  .globl Median  .type Median, @function  Median:  mv t3,a0  ble a1,zero,.L8  addiw a2,a1,-1  slli a5,a2,32  srli a2,a5,30  addi a5,a0,4  add a2,a2,a5  mv a7,a0  li a0,0  li t1,0  li t4,2  j .L3  .L4:  bge a5,zero,.L5  addiw a6,a6,1  .L5:  addi a4,a4,4  beq a4,a2,.L11  .L6:  lw a5,0(a4)  subw a5,a1,a5  ble a5,zero,.L4  addiw a3,a3,1  j .L5  .L11:  subw a3,a3,a6  addiw a3,a3,1  bgtu a3,t4,.L7  mv a0,a1  .L7:  addi a7,a7,4  beq a7,a2,.L2  .L3:  lw a1,0(a7)  mv a4,t3  mv a3,t1  mv a6,t1  j .L6  .L8:  li a0,0  .L2:  ret  .size Median, .-Median  .ident "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"  Рисунок 8 – Выход компилятора файла Median.s |

Из полученных данных видно, что в программе main выполняется обращение к подпрограмме Median (call Median, значение регистра ra, которое содержит адрес возврата из main, было сохранено на время вызова в стеке).

## **3.4 АНАЛИХ СОСТАВА И СОДЕРЖИМОГО ТАБЛИЦЫ СИМВОЛОВ, ТАБЛИЦЫ ПЕРЕМЕЩЕНИЙ И ОТЛАДОЧНОЙ ИНФОРМАЦИИ, СОДЕРЖАЩИХСЯ В ИСПОЛНИТЕЛЬНОМ И ОБЪЕКТНЫХ ФАЙЛАХ**

**1.** Ранее уже были сформированы ассемблером объектные файлы main.o и Median.o, которые содержат коды инструкций, а также таблицу символов и перемещений. Воспользуюсь утилитой objdump, чтобы исследовать их содержимое. Содержимое бинарных файлов в текстовом виде приведено на рисунке 9 и 10.

|  |
| --- |
| Рисунок 9 – Содержимое main.o |
| Рисунок 10 – Содержимое Median.o |

Можно увидеть, что в файлах есть несколько сегментов:

0.text – сегмент кода;

1.data – сегмент инициализированных данных;

2.bss – сегмент данных, которые были инициализированы нулями;

3.rodata.str1.8 – часть сегмента rodata, который служит для хранения дополнительной информации для компоновщика;

4.rodata – сегмент неизменяемых данных;

5.comment – сегмент данных о версиях;

6.riscv.attributes – сегмент, хранящий атрибуты для указания определенных свойств функции.

Значения в колонках приведены в 16-разрядной системе счисления.

**2.** На рисунках 11 и 12 приведены таблицы символов объектных файлов main.o и Median.o соответственно.

|  |
| --- |
| Рисунок 11 – Таблица символов для файла main.o |
| Рисунок 12 – Таблица символов для файла Median.o |

В таблицах можно заметить один глобальный флаг – символ типа «Function» для main и для Median. Символы типа «UND» - неопределенные – они означают, что в ассемблерном коде использовались данные символы, но они не были определены. Символы определяются в другом месте, поэтому ассемблер отразил это так в таблице.

Содержимое сегмента .text для файлов main.o и Median.o приведено на рисунках 13 и 14 соответственно.

|  |
| --- |
| Рисунок 13 – Содержимое сегмента .text файла main.o |

Исходя их полученных результатов можно провести аналогию между объектным файлом main.o и файлом main.s: псевдоинструкция вызова подпрограммы Median была переведена ассемблером в пару инструкций, приведенных на рисунке 15.

|  |
| --- |
| Рисунок 15 – Полученные инструкции ассемблером |

|  |
| --- |
| Рисунок 14 – Содержимое сегмента .text файла Median.o |

Секции .data объектных файлов – сегмент инициализированных данных – не содержат данных, размер секций, как было выведено выше, равен нулю. Данные представлены на рисунках 15 и 16.

|  |
| --- |
| Рисунок 15 – Содержимое сегмента .data файла main.o |
| Рисунок 16 – Содержимое сегмента .data файла Median.o |

Секции .bss объектных файлов – секции данных, инициализированных нулями – таким же образом пусты. Данные представлены на рисунках 17 и 18

|  |
| --- |
| Рисунок 17 – Содержимое сегмента .bss файла main.o |
| Рисунок 18 – Содержимое сегмента .bss файла Median.o |

Секция .comment – секция данных о версиях – и для одного и для другого файла содержит одни и те же значения – сведения о GCC версии 8.3.0 от SiFive. Данные представлены на рисунках 19 и 20.

|  |
| --- |
| Рисунок 19 – Содержимое сегмента .comment файла main.o |
| Рисунок 20 – Содержимое сегмента .comment файла Median.o |

Секция .riscv.attributes обоих объектных файлов содержит одну и ту же информацию об используемой архитектуре команд RV32I. Данные представлены на рисунке 21 и 22.

|  |
| --- |
| Рисунок 21 – Содержимое сегмента .riscv.attributes файла main.o |
| Рисунок 22 - Содержимое сегмента .riscv.attributes файла Median.o |

## **3.5 СОДЕРЖИМОЕ ТАБЛИЦЫ ПЕРЕМЕЩЕНИЙ**

Содержимое таблицы перемещений представлено на рисунках 23 и 24 для файлов main и Median соответственно. В таблицах перемещения Median.o содержится информация о переходах (R\_RISCV\_JAL) и ветвлениях (R\_RISCV\_BRANCH). В таблицах перемещения main.o, есть R\_RISCV\_CALL, чтобы информация о переходах соответствовала Median. Записи типа R\_RISCV\_RELAX заносятся в таблицу перемещений в дополнение к записям типа R\_RISCV\_CAL.

|  |
| --- |
| Рисунок 23 – Содержимое таблицы перемещений для файла main.o |
| Рисунок 24 – Содержимое таблицы перемещений для файла Median.o |

## **3.6 РЕЗУЛЬТАТ КОМПОНОВКИ**

Для компоновки файлов использую следующие команды. Команды приведены на рисунке 25.

|  |
| --- |
| Рисунок 25 – Команды для компоновки |

Содержимое файла mainC.c приведено на рисунке 26. Компоновщик все переходы auipc+jalr, заменил на одну инструкцию jal и корректным адресом перехода.

|  |
| --- |
| 0000000000010188 <main>:  10188: 715d c.addi16sp sp,-80  1018a: e486 c.sdsp ra,72(sp)  1018c: e0a2 c.sdsp s0,64(sp)  1018e: fc26 c.sdsp s1,56(sp)  10190: f84a c.sdsp s2,48(sp)  10192: f44e c.sdsp s3,40(sp)  10194: 0001d7b7 lui a5,0x1d  10198: 28878793 addi a5,a5,648 # 1d288 <\_\_clzdi2+0x70>  1019c: 6398 c.ld a4,0(a5)  1019e: e43a c.sdsp a4,8(sp)  101a0: 6798 c.ld a4,8(a5)  101a2: e83a c.sdsp a4,16(sp)  101a4: 4b9c c.lw a5,16(a5)  101a6: cc3e c.swsp a5,24(sp)  101a8: 0001d537 lui a0,0x1d  101ac: 25050513 addi a0,a0,592 # 1d250 <\_\_clzdi2+0x38>  101b0: 00000097 auipc ra,0x0  101b4: 248080e7 jalr ra,584(ra) # 103f8 <printf>  101b8: 45a2 c.lwsp a1,8(sp)  101ba: 0001d537 lui a0,0x1d  101be: 26050513 addi a0,a0,608 # 1d260 <\_\_clzdi2+0x48>  101c2: 00000097 auipc ra,0x0  101c6: 236080e7 jalr ra,566(ra) # 103f8 <printf>  101ca: 0020 c.addi4spn s0,sp,8  101cc: 01810993 addi s3,sp,24  101d0: 0001d937 lui s2,0x1d  101d4: 0001d4b7 lui s1,0x1d  101d8: 26890513 addi a0,s2,616 # 1d268 <\_\_clzdi2+0x50>  101dc: 00000097 auipc ra,0x0  101e0: 21c080e7 jalr ra,540(ra) # 103f8 <printf>  101e4: 404c c.lw a1,4(s0)  101e6: 26048513 addi a0,s1,608 # 1d260 <\_\_clzdi2+0x48>  101ea: 00000097 auipc ra,0x0  101ee: 20e080e7 jalr ra,526(ra) # 103f8 <printf>  101f2: 0411 c.addi s0,4  101f4: ff3412e3 bne s0,s3,101d8 <main+0x50>  101f8: 0001d537 lui a0,0x1d  101fc: 27050513 addi a0,a0,624 # 1d270 <\_\_clzdi2+0x58>  10200: 00000097 auipc ra,0x0  10204: 2b8080e7 jalr ra,696(ra) # 104b8 <puts>  10208: 4595 c.li a1,5  1020a: 0028 c.addi4spn a0,sp,8  1020c: 00000097 auipc ra,0x0  10210: 02a080e7 jalr ra,42(ra) # 10236 <Median>  10214: 85aa c.mv a1,a0  10216: 0001d537 lui a0,0x1d  1021a: 27850513 addi a0,a0,632 # 1d278 <\_\_clzdi2+0x60>  1021e: 00000097 auipc ra,0x0  10222: 1da080e7 jalr ra,474(ra) # 103f8 <printf>  10226: 4501 c.li a0,0  10228: 60a6 c.ldsp ra,72(sp)  1022a: 6406 c.ldsp s0,64(sp)  1022c: 74e2 c.ldsp s1,56(sp)  1022e: 7942 c.ldsp s2,48(sp)  10230: 79a2 c.ldsp s3,40(sp)  10232: 6161 c.addi16sp sp,80  10234: 8082 c.jr ra  0000000000010236 <Median>:  10236: 8e2a c.mv t3,a0  10238: 04b05c63 bge zero,a1,10290 <Median+0x5a>  1023c: fff5861b addiw a2,a1,-1  10240: 02061793 slli a5,a2,0x20  10244: 01e7d613 srli a2,a5,0x1e  10248: 00450793 addi a5,a0,4  1024c: 963e c.add a2,a5  1024e: 88aa c.mv a7,a0  10250: 4501 c.li a0,0  10252: 4301 c.li t1,0  10254: 4e89 c.li t4,2  10256: a03d c.j 10284 <Median+0x4e>  10258: 0007d363 bge a5,zero,1025e <Median+0x28>  1025c: 2805 c.addiw a6,1  1025e: 0711 c.addi a4,4  10260: 00c70963 beq a4,a2,10272 <Median+0x3c>  10264: 431c c.lw a5,0(a4)  10266: 40f587bb subw a5,a1,a5  1026a: fef057e3 bge zero,a5,10258 <Median+0x22>  1026e: 2685 c.addiw a3,1  10270: b7fd c.j 1025e <Median+0x28>  10272: 410686bb subw a3,a3,a6  10276: 2685 c.addiw a3,1  10278: 00dee363 bltu t4,a3,1027e <Median+0x48>  1027c: 852e c.mv a0,a1  1027e: 0891 c.addi a7,4  10280: 00c88963 beq a7,a2,10292 <Median+0x5c>  10284: 0008a583 lw a1,0(a7)  10288: 8772 c.mv a4,t3  1028a: 869a c.mv a3,t1  1028c: 881a c.mv a6,t1  1028e: bfd9 c.j 10264 <Median+0x2e>  10290: 4501 c.li a0,0  10292: 8082 c.jr ra  Рисунок 26 – Часть содержимого файла компоновки |

## **3.7 АНАЛИЗ ОТЛАДОЧНОЙ ИНФОРМАЦИИ**

Перечислю сегменты исполняемого файла на рисунке 27.

|  |
| --- |
| Рисунок 27 – Список сегментов исполняемого файла |

Файл, который был сформирован, содержит информацию для отладки (в сегментах .debug), а также полную таблицу символов и сведения о версиях средств разработки:

11.debug\_abbrev – сегмент с сокращениями, используемыми в .debug\_info;

12.debug\_aranges – сегмент с таблицей поиска для сопоставления адресов с единицами компиляции;

13.debug\_frame – сегмент с информацией о кадре вызова;

14.debug\_info – сегмент основной информации;

15.debug\_line – сегмент информация о номере строки;

16.debug\_loc – сегмент со списком местоположений, используемых в атрибутах DW\_AT\_location;

17.debug\_ranges – сегмент с диапазонами адресов, используемых в атрибутах DW\_AT\_ranges;

18.debug\_str – сегмент с таблицей строк, используемой в .debug\_info.

Таблица символов содержит множество дополнительных вхождений, однако в целом определяет все нужные секции, метки и адреса. Функции Median и main так же помечены флагом F, но в отличие от стадии ассемблирования, все они являются определенными и содержатся по корректным адресам для успешного вызова этих функций из других участков программ.

Проанализируем таблицу перемещений исполняемого файла, которая приведена на рисунке 28. Таблица перемещений оказывается пуста, все необходимые релокации, оптимизации и замены инструкций были успешно проведены компоновщиком.

|  |
| --- |
| Рисунок 28 – Содержимое таблицы перемещений файла main |

Итогом сборки программ на языке C «по шагам» является исполняемый на процессорах архитектуры RISC-V файл, решающий задачу поиска максимального простого числа до числа и проверяющий корректность решения этой задачи на тестовом примере.

## **3.8 ВЫДЕЛЕНИЕ РАЗРАБОТАННОЙ ФУНКЦИИ В СТАТИЧСЕКУЮ БИБЛИОТЕКУ**

Статическая библиотека (staticlibrary) 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. Разработанная функция поиска максимального простого числа до числа n содержится в единственном исходном файле на языке C. Выделим этот файл в статическую библиотеку. Результаты приведены на рисунках 29 и 30.

|  |
| --- |
| Рисунок 29 – Команды выделения функции в стат. Библиотеку |
| Рисунок 30 – Список символов библиотеки |

В выводе утилиты «nm» кодом «T» обозначаются символы, определенные в соответствующем объектном файле, кодом “U” - внешние символы. Для нашей библиотеки нет внешних символов.

Символ функции Median является основным символом, определяемым в этом объектном файле, остальные символы определяют лишь локальные для этого файла метки.

Создам make-файл, который приведен на рисунке 31.

|  |
| --- |
| Рисунок 31 – make-файл |

В make-файле создается объектный файл main.o из исходного, создается объектный файл Median.o из исходного, происходит архивация объектного файла Median.o, а именно создается статическая библиотека, происходит компоновка статической библиотеки Medianlib.o с объектным файлом main.o, и получается исполняемый файл output.

Результаты работы представлены на рисунке 32.

|  |
| --- |
| Рисунок 32 – Компоновка библиотеки и получение исполняемого файла output |

# **4. РЕЗУЛЬТАТЫ**

В ходе выполнения работы был исследован процесс сборки проекта на языке С. Были разобраны составные части процесса сборки проекта: препроцессирование файла, компиляция файла в файл ассемблера, ассемблирование в исполняемый файл, компоновка объектного файла в исполняемый файл. Также были рассмотрена упрощенная версия сборки при помощи make-файлов, которые содержат инструкции для корректной сборки программы.