

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: 4

Выполнил студент гр. 3530901/00002 _____ Н.А. Васихин
(подпись)

Принял старший преподаватель _____ Д.С. Степанов
(подпись)

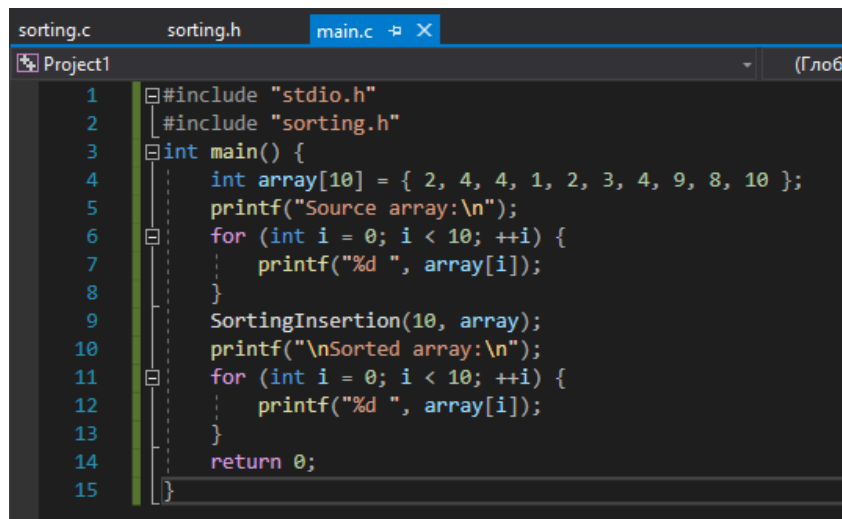
“ ____ ” _____ 2021 г.

Санкт-Петербург
2021

Постановка задачи

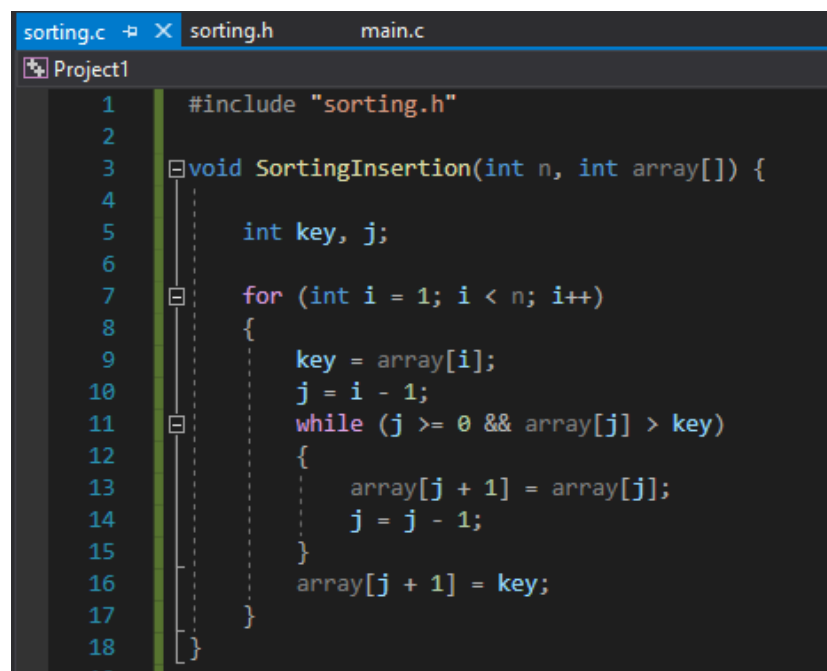
1. Изучить методические материалы, опубликованные на сайте курса.
2. Установить пакет средств разработки “SiFive GNU Embedded Toolchain” для RISC-V.
3. На языке C разработать функцию, реализующую сортировку массива вставкой. Поместить определение функции в отдельный исходный файл, оформить заголовочный 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 исполнимом 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.

1. Программа на языке C:



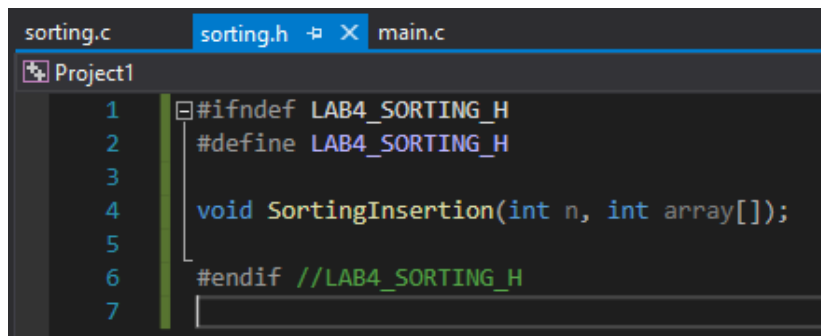
```
1 #include "stdio.h"
2 #include "sorting.h"
3 int main() {
4     int array[10] = { 2, 4, 4, 1, 2, 3, 4, 9, 8, 10 };
5     printf("Source array:\n");
6     for (int i = 0; i < 10; ++i) {
7         printf("%d ", array[i]);
8     }
9     SortingInsertion(10, array);
10    printf("\nSorted array:\n");
11    for (int i = 0; i < 10; ++i) {
12        printf("%d ", array[i]);
13    }
14    return 0;
15 }
```

Рис 1.1 Файл тестовой программы main.c



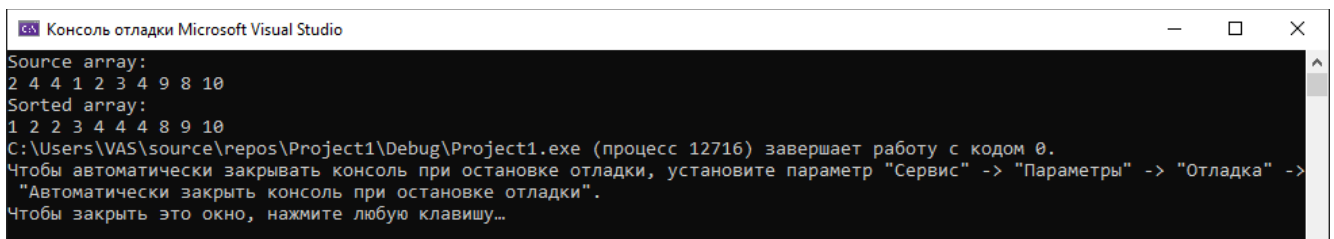
```
1 #include "sorting.h"
2
3 void SortingInsertion(int n, int array[]) {
4     int key, j;
5
6     for (int i = 1; i < n; i++)
7     {
8         key = array[i];
9         j = i - 1;
10        while (j >= 0 && array[j] > key)
11        {
12            array[j + 1] = array[j];
13            j = j - 1;
14        }
15        array[j + 1] = key;
16    }
17 }
18 }
```

Рис 1.2 Файл функции сортировки вставкой sorting.c



```
1  #ifndef LAB4_SORTING_H
2  #define LAB4_SORTING_H
3
4  void SortingInsertion(int n, int array[]);
5
6  #endif //LAB4_SORTING_H
7
```

Рис 1.3 Заголовочный файл sorting.h



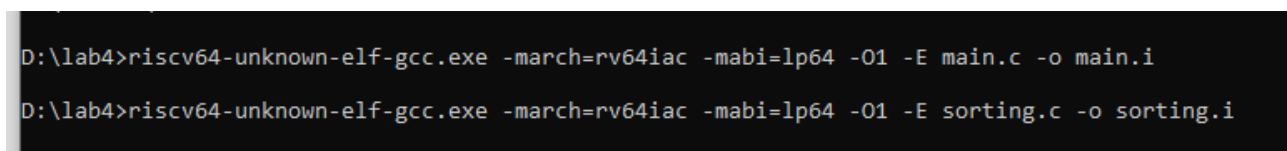
```
Source array:
2 4 4 1 2 3 4 9 8 10
Sorted array:
1 2 2 3 4 4 4 8 9 10
C:\Users\VAS\source\repos\Project1\Debug\Project1.exe (процесс 12716) завершает работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, установите параметр "Сервис" -> "Параметры" -> "Отладка" ->
"Автоматически закрыть консоль при остановке отладки".
Чтобы закрыть это окно, нажмите любую клавишу...
```

Рис. 1.4 Результат работы программы

2. Сборка программы “по шагам”

Препроцессирование

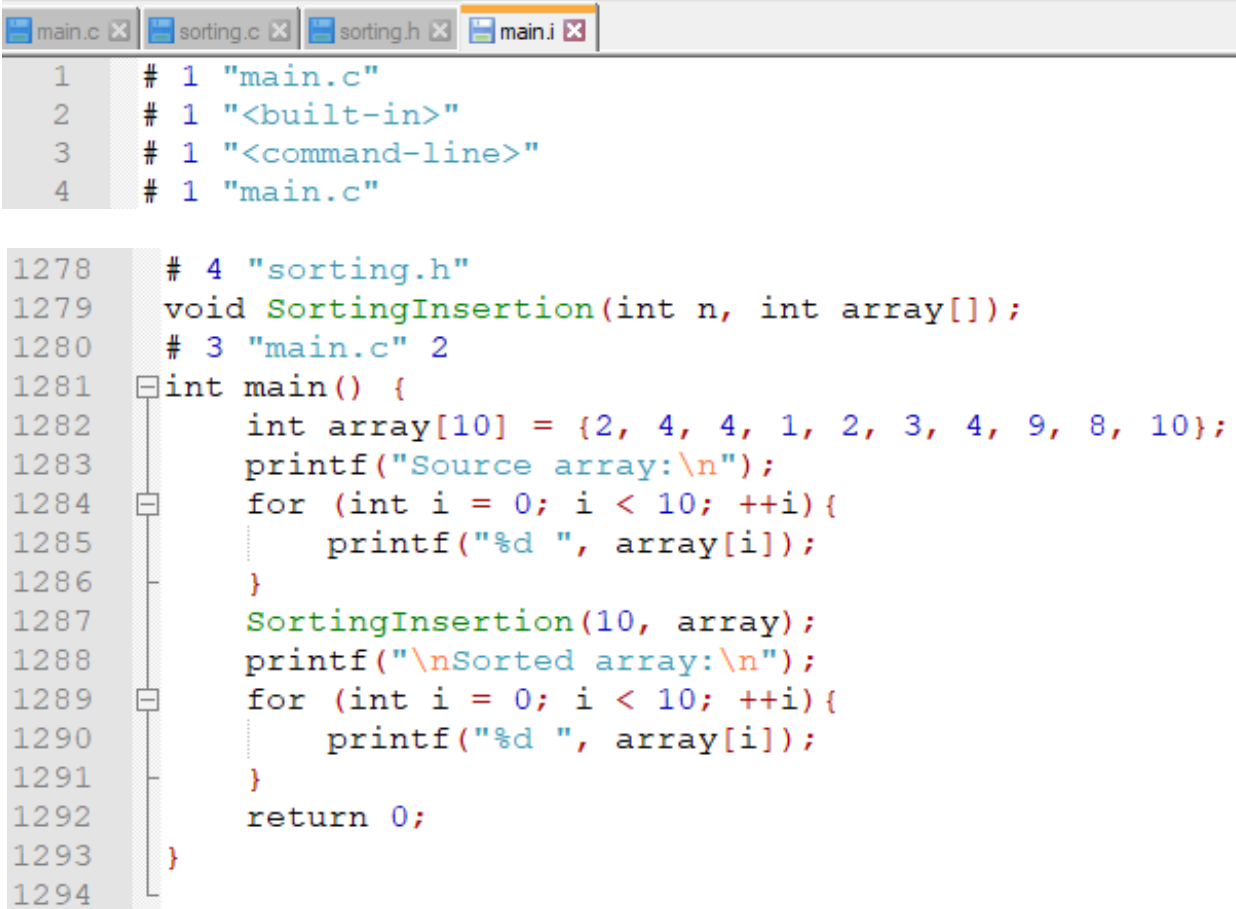
Используя пакет разработки “SiFive GNU Embedded Toolchain” выполним препроцессирование файлов, используя следующие команды:



```
D:\lab4>riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -O1 -E main.c -o main.i
D:\lab4>riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -O1 -E sorting.c -o sorting.i
```

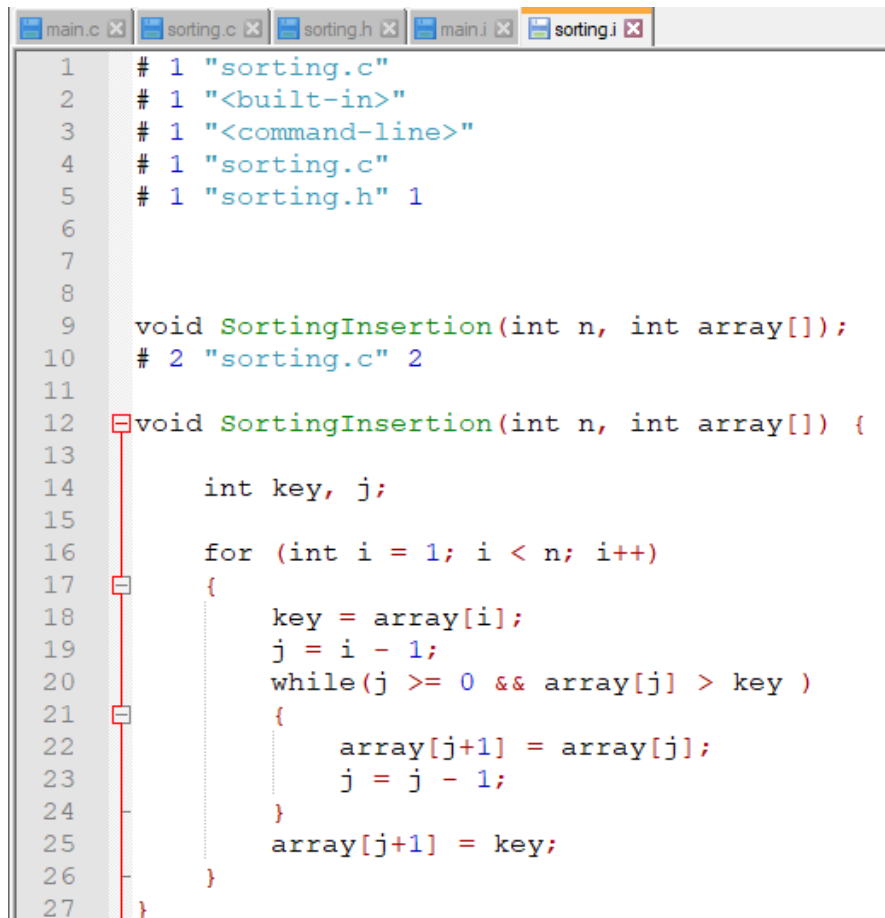
Параметр *-E* указывает на то, что обработка файлов должна происходить только препроцессором. Параметр *-o* отвечает за название результирующего файла, то есть результат препроцессирования содержится в файлах *main.i* и *sorting.i*. В связи с тем, что в файле тестовой программы мы использовали стандартную библиотеку языка C “*stdio.h*” для вывода на консоль значений массива, результирующий файл препроцессирования имеет много добавочных строк.

Листинг 2.1. Файл main.i (фрагмент)



```
1      # 1 "main.c"
2      # 1 "<built-in>"
3      # 1 "<command-line>"
4      # 1 "main.c"

1278   # 4 "sorting.h"
1279   void SortingInsertion(int n, int array[]);
1280   # 3 "main.c" 2
1281   int main() {
1282       int array[10] = {2, 4, 4, 1, 2, 3, 4, 9, 8, 10};
1283       printf("Source array:\n");
1284       for (int i = 0; i < 10; ++i){
1285           printf("%d ", array[i]);
1286       }
1287       SortingInsertion(10, array);
1288       printf("\nSorted array:\n");
1289       for (int i = 0; i < 10; ++i){
1290           printf("%d ", array[i]);
1291       }
1292       return 0;
1293   }
1294
```



```

1  # 1 "sorting.c"
2  # 1 "<built-in>"
3  # 1 "<command-line>"
4  # 1 "sorting.c"
5  # 1 "sorting.h" 1
6
7
8
9  void SortingInsertion(int n, int array[]);
10 # 2 "sorting.c" 2
11
12 void SortingInsertion(int n, int array[]) {
13
14     int key, j;
15
16     for (int i = 1; i < n; i++)
17     {
18         key = array[i];
19         j = i - 1;
20         while(j >= 0 && array[j] > key )
21         {
22             array[j+1] = array[j];
23             j = j - 1;
24         }
25         array[j+1] = key;
26     }
27 }

```

Появившиеся нестандартные директивы, начинающиеся с символа “#”, используются для передачи информации об исходном тексте из препроцессора в компилятор. Например, последняя директива «# 1 “main.c”» информирует компилятор о том, что следующая строка является результатом обработки строки 1 исходного файла “main.c”. Также мы видим, что в данных файлах содержится информация из заголовочного файла.

Компиляция

Компиляция осуществляется следующими командами:

```

D:\lab4>riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -O1 -S main.i -o main.s
D:\lab4>riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -O1 -S sorting.i -o sorting.s

```

```

.file "main.c"
.option nopic
.attribute arch, "rv64i2p0_a2p0_c2p0"
.attribute unaligned_access, 0
.attribute stack_align, 16
.text
.align      1
.globl      main
.type main, @function
main:
    addi sp,sp,-96
    sd    ra,88(sp)
    sd    s0,80(sp)
    sd    s1,72(sp)
    sd    s2,64(sp)
    sd    s3,56(sp)
    lui   a5,%hi(.LANCHOR0)
    addi  a5,a5,%lo(.LANCHOR0)
    ld    a1,0(a5)
    ld    a2,8(a5)
    ld    a3,16(a5)
    ld    a4,24(a5)
    ld    a5,32(a5)
    sd    a1,8(sp)
    sd    a2,16(sp)
    sd    a3,24(sp)
    sd    a4,32(sp)
    sd    a5,40(sp)
    lui   a0,%hi(.LC1)
    addi  a0,a0,%lo(.LC1)
    call  puts
    addi  s0,sp,8
    addi  s2,sp,48
    mv    s1,s0
    lui   s3,%hi(.LC2)
.L2:
    lw    a1,0(s1)
    addi  a0,s3,%lo(.LC2)
    call  printf
    addi  s1,s1,4
    bne   s1,s2,.L2
    addi  a1,sp,8
    li    a0,10
    call  SortingInsertion
    lui   a0,%hi(.LC3)
    addi  a0,a0,%lo(.LC3)
    call  puts
    lui   s1,%hi(.LC2)
.L3:

```

```

    lw    a1,0(s0)
    addi  a0,s1,%lo(.LC2)
    call  printf
    addi  s0,s0,4
    bne   s0,s2,.L3
    li    a0,0
    ld    ra,88(sp)
    ld    s0,80(sp)
    ld    s1,72(sp)
    ld    s2,64(sp)
    ld    s3,56(sp)
    addi  sp,sp,96
    jr    ra
.size main, .-main
.section .rodata
.align   3
.set    .LANCHOR0, . + 0
.LC0:
    .word 2
    .word 4
    .word 4
    .word 1
    .word 2
    .word 3
    .word 4
    .word 9
    .word 8
    .word 10
    .section .rodata.str1.8,"aMS",@progbits,1
    .align   3
.LC1:
    .string  "Source array:"
    .zero 2
.LC2:
    .string  "%d "
    .zero 4
.LC3:
    .string  "\nSorted array:"
    .ident   "GCC: (SiFive GCC 8.3.0-2020.04.1) 8.3.0"

```

Листинг 2.4. Файл sorting.s

```

.file "sorting.c"
.option nopic
.attribute arch, "rv64i2p0_a2p0_c2p0"
.attribute unaligned_access, 0
.attribute stack_align, 16
.text
.align 1
.globl SortingInsertion

```



```

.type   SortingInsertion, @function
SortingInsertion:
    li    a5,1
    ble    a0,a5,.L1
    addi   a6,a1,4
    addiw  t1,a0,-1
    li    a7,0
    li    a0,-1
    j      .L5
.L3:
    addi   a5,a5,1
    slli   a5,a5,2
    add    a5,a1,a5
    sw     a2,0(a5)
    addiw  a7,a7,1
    addi   a6,a6,4
    beq    a7,t1,.L1
.L5:
    lw     a2,0(a6)
    sext.w a5,a7
    blt    a5,zero,.L3
    lw     a3,-4(a6)
    bge    a2,a3,.L3
    mv     a4,a6
.L4:
    sw     a3,0(a4)
    addiw  a5,a5,-1
    beq    a5,a0,.L3
    addi   a4,a4,-4
    lw     a3,-4(a4)
    bgt    a3,a2,.L4
    j      .L3
.L1:
    ret
.size   SortingInsertion, .-SortingInsertion
.ident  "GCC: (SiFive GCC 8.3.0-2020.04.1) 8.3.0"

```

Ассемблирование

Ассемблирование осуществляется следующими командами:

```
D:\lab4>riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -v -c main.s -o main.o
```

```
D:\lab4>riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -v -c sorting.s -o sorting.o
```

На выходе мы получаем два бинарных файла “main.o” и “sorting.o”. Для их прочтения используем программу из пакета разработки.

Листинг 2.5. Заголовки секций файла main.o

```
D:\lab4>riscv64-unknown-elf-objdump.exe -h main.o

main.o:      file format elf64-littleriscv

Sections:
Idx Name          Size      VMA           LMA           File off  Algn
 0 .text          0000009c  0000000000000000 0000000000000000 00000040 2**1
                CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
 1 .data          00000000  0000000000000000 0000000000000000 000000dc 2**0
                CONTENTS, ALLOC, LOAD, DATA
 2 .bss           00000000  0000000000000000 0000000000000000 000000dc 2**0
                ALLOC
 3 .rodata        00000028  0000000000000000 0000000000000000 000000e0 2**3
                CONTENTS, ALLOC, LOAD, READONLY, DATA
 4 .rodata.str1.8 00000027  0000000000000000 0000000000000000 00000108 2**3
                CONTENTS, ALLOC, LOAD, READONLY, DATA
 5 .comment       00000029  0000000000000000 0000000000000000 0000012f 2**0
                CONTENTS, READONLY
 6 .riscv.attributes 00000026  0000000000000000 0000000000000000 00000158 2**0
                CONTENTS, READONLY
```

Листинг 2.6. Заголовки секций файла sorting.o

```
D:\lab4>riscv64-unknown-elf-objdump.exe -h sorting.o

sorting.o:    file format elf64-littleriscv

Sections:
Idx Name          Size      VMA           LMA           File off  Algn
 0 .text          00000050  0000000000000000 0000000000000000 00000040 2**1
                CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
 1 .data          00000000  0000000000000000 0000000000000000 00000090 2**0
                CONTENTS, ALLOC, LOAD, DATA
 2 .bss           00000000  0000000000000000 0000000000000000 00000090 2**0
                ALLOC
 3 .comment       00000029  0000000000000000 0000000000000000 00000090 2**0
                CONTENTS, READONLY
 4 .riscv.attributes 00000026  0000000000000000 0000000000000000 000000b9 2**0
                CONTENTS, READONLY
```

Секции:

- .text* - скомпилированный машинный код;
- .data* - секция инициализированных данных;
- .rodata* - аналог *.data* для неизменяемых данных;
- .bss* - секция данных, инициализированных нулями;
- .comment* — информация о версии компилятора;

вывод `objdump` нам также сообщает, что RISC-V является little-endian

архитектурой, произведем декодирование кода, чтобы рассмотреть секцию *.text* подробнее, с помощью команды:

```
D:\lab4>riscv64-unknown-elf-objdump -d -M no-aliases -j .text main.o
```

Опция “-d” инициирует процесс дизассемблирования, опция “-M no-aliases” требует использовать в выводе только инструкции системы команд (но не псевдоинструкции ассемблера)

Листинг 2.7. Дизассемблированный файл main.o

```
main.o:      file format elf64-littleriscv
Disassembly of section .text:
0000000000000000 <main>:
 0:   711d                c.addi16sp      sp,-96
 2:   ec86                c.sdsp   ra,88(sp)
 4:   e8a2                c.sdsp   s0,80(sp)
 6:   e4a6                c.sdsp   s1,72(sp)
 8:   e0ca                c.sdsp   s2,64(sp)
 a:   fc4e                c.sdsp   s3,56(sp)
 c:   000007b7            lui      a5,0x0
10:   00078793            addi     a5,a5,0 # 0 <main>
14:   638c                c.ld     a1,0(a5)
16:   6790                c.ld     a2,8(a5)
18:   6b94                c.ld     a3,16(a5)
1a:   6f98                c.ld     a4,24(a5)
1c:   739c                c.ld     a5,32(a5)
1e:   e42e                c.sdsp   a1,8(sp)
20:   e832                c.sdsp   a2,16(sp)
22:   ec36                c.sdsp   a3,24(sp)
24:   f03a                c.sdsp   a4,32(sp)
26:   f43e                c.sdsp   a5,40(sp)
28:   00000537            lui      a0,0x0
2c:   00050513            addi     a0,a0,0 # 0 <main>
30:   00000097            auipc    ra,0x0
34:   000080e7            jalr     ra,0(ra) # 30 <main+0x30>
38:   0020                c.addi4spn   s0,sp,8
3a:   03010913            addi     s2,sp,48
3e:   84a2                c.mv     s1,s0
40:   000009b7            lui      s3,0x0

0000000000000044 <.L2>:
44:   408c                c.lw     a1,0(s1)
46:   00098513            addi     a0,s3,0 # 0 <main>
4a:   00000097            auipc    ra,0x0
4e:   000080e7            jalr     ra,0(ra) # 4a <.L2+0x6>
52:   0491                c.addi    s1,4
54:   ff2498e3            bne      s1,s2,44 <.L2>
58:   002c                c.addi4spn   a1,sp,8
5a:   4529                c.li     a0,10
5c:   00000097            auipc    ra,0x0
60:   000080e7            jalr     ra,0(ra) # 5c <.L2+0x18>
```

```

64: 00000537      lui      a0,0x0
68: 00050513      addi     a0,a0,0 # 0 <main>
6c: 00000097      auipc    ra,0x0
70: 000080e7      jalr     ra,0(ra) # 6c <.L2+0x28>
74: 000004b7      lui      s1,0x0

0000000000000078 <.L3>:
78: 400c          c.lw      a1,0(s0)
7a: 00048513      addi     a0,s1,0 # 0 <main>
7e: 00000097      auipc    ra,0x0
82: 000080e7      jalr     ra,0(ra) # 7e <.L3+0x6>
86: 0411          c.addi    s0,4
88: ff2418e3      bne      s0,s2,78 <.L3>
8c: 4501          c.li      a0,0
8e: 60e6          c.ldsp    ra,88(sp)
90: 6446          c.ldsp    s0,80(sp)
92: 64a6          c.ldsp    s1,72(sp)
94: 6906          c.ldsp    s2,64(sp)
96: 79e2          c.ldsp    s3,56(sp)
98: 6125          c.addi16sp      sp,96
9a: 8082          c.jr      ra

```

Мы видим, как происходит выход из подпрограммы “*c.jr ra*”, также мы видим, как сочетание инструкций *auipc* и *jalr* заменяют псевдоинструкцию *call*.

Рассмотрим таблицу символов и таблицу перемещений с помощью команд:

```
D:\lab4>riscv64-unknown-elf-objdump -t main.o sorting.o
```

```
D:\lab4>riscv64-unknown-elf-objdump -r main.o sorting.o
```

Листинг 2.8. Таблица символов

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

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      0000000000000000 .rodata
0000000000000000 1      .rodata      0000000000000000 .LANCHOR0
0000000000000000 1      d  .rodata.str1.8 0000000000000000 .rodata.str1.8
0000000000000000 1      .rodata.str1.8 0000000000000000 .LC1

```

```

000000000000000010 1      .rodata.str1.8 0000000000000000 .LC2
000000000000000018 1      .rodata.str1.8 0000000000000000 .LC3
000000000000000044 1      .text 0000000000000000 .L2
000000000000000078 1      .text 0000000000000000 .L3
000000000000000000 1      d .comment 0000000000000000 .comment
000000000000000000 1      d .riscv.attributes
000000000000000000 .riscv.attributes
000000000000000000 g      F .text 000000000000009c main
000000000000000000      *UND* 0000000000000000 puts
000000000000000000      *UND* 0000000000000000 printf
000000000000000000      *UND* 0000000000000000 SortingInsertion

```

sorting.o: file format elf64-littleriscv

SYMBOL TABLE:

```

000000000000000000 1      df *ABS* 0000000000000000 sorting.c
000000000000000000 1      d .text 0000000000000000 .text
000000000000000000 1      d .data 0000000000000000 .data
000000000000000000 1      d .bss 0000000000000000 .bss
00000000000000004e 1      .text 0000000000000000 .L1
000000000000000024 1      .text 0000000000000000 .L5
000000000000000014 1      .text 0000000000000000 .L3
00000000000000003a 1      .text 0000000000000000 .L4
000000000000000000 1      d .comment 0000000000000000 .comment
000000000000000000 1      d .riscv.attributes
000000000000000000 .riscv.attributes
000000000000000000 g      F .text 0000000000000050 SortingInsertion

```

В таблице символов main.o имеется запись: символ “puts” типа *UND*. Эта запись означает, что символ “puts” 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 “SortingInsertion”

Листинг 2.9. Таблица перемещений

main.o: file format elf64-littleriscv

RELOCATION RECORDS FOR [.text]:

OFFSET	TYPE	VALUE
0000000000000000c	R_RISCV_HI20	.LANCHOR0
0000000000000000c	R_RISCV_RELAX	*ABS*

00000000000000010	R_RISCV_L012_I	.LANCHOR0
00000000000000010	R_RISCV_RELAX	*ABS*
00000000000000028	R_RISCV_HI20	.LC1
00000000000000028	R_RISCV_RELAX	*ABS*
0000000000000002c	R_RISCV_L012_I	.LC1
0000000000000002c	R_RISCV_RELAX	*ABS*
00000000000000030	R_RISCV_CALL	puts
00000000000000030	R_RISCV_RELAX	*ABS*
00000000000000040	R_RISCV_HI20	.LC2
00000000000000040	R_RISCV_RELAX	*ABS*
00000000000000046	R_RISCV_L012_I	.LC2
00000000000000046	R_RISCV_RELAX	*ABS*
0000000000000004a	R_RISCV_CALL	printf
0000000000000004a	R_RISCV_RELAX	*ABS*
0000000000000005c	R_RISCV_CALL	SortingInsertion
0000000000000005c	R_RISCV_RELAX	*ABS*
00000000000000064	R_RISCV_HI20	.LC3
00000000000000064	R_RISCV_RELAX	*ABS*
00000000000000068	R_RISCV_L012_I	.LC3
00000000000000068	R_RISCV_RELAX	*ABS*
0000000000000006c	R_RISCV_CALL	puts
0000000000000006c	R_RISCV_RELAX	*ABS*
00000000000000074	R_RISCV_HI20	.LC2
00000000000000074	R_RISCV_RELAX	*ABS*
0000000000000007a	R_RISCV_L012_I	.LC2
0000000000000007a	R_RISCV_RELAX	*ABS*
0000000000000007e	R_RISCV_CALL	printf
0000000000000007e	R_RISCV_RELAX	*ABS*
00000000000000054	R_RISCV_BRANCH	.L2
00000000000000088	R_RISCV_BRANCH	.L3

sorting.o: file format elf64-littleriscv

RELOCATION RECORDS FOR [.text]:

OFFSET	TYPE	VALUE
0000000000000002	R_RISCV_BRANCH	.L1
0000000000000012	R_RISCV_RVC_JUMP	.L5
0000000000000020	R_RISCV_BRANCH	.L1
000000000000002c	R_RISCV_BRANCH	.L3
0000000000000034	R_RISCV_BRANCH	.L3
000000000000003e	R_RISCV_BRANCH	.L3
0000000000000048	R_RISCV_BRANCH	.L4
000000000000004c	R_RISCV_RVC_JUMP	.L3

Здесь содержится информация обо всех «неоконченных» инструкциях.

Записи типа “R_RISCV_RELAX” заносятся в таблицу перемещений в дополнение к записям типа “R_RISCV_CALL” и сообщают компоновщику, что пара инструкций, обеспечивающих вызов подпрограммы может быть оптимизирована.

Компиляция

Выполним компоновку следующей командой:

```
D:\lab4>riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -v main.o sorting.o
```

Листинг 2.10. Фрагмент исполняемого файла

```
D:\lab4>riscv64-unknown-elf-objdump -j .text -d -M no-aliases a.out >a.ds
```

```
0000000000010158 <main>:
 10158: 711d          c.addi16sp sp,-96
 1015a: ec86          c.sdsp      ra,88(sp)
 1015c: e8a2          c.sdsp      s0,80(sp)
 1015e: e4a6          c.sdsp      s1,72(sp)
 10160: e0ca          c.sdsp      s2,64(sp)
 10162: fc4e          c.sdsp      s3,56(sp)
 10164: 67f1          c.lui a5,0x1c
 10166: 57078793      addi a5,a5,1392 # 1c570 <__clzdi2+0x36>
 1016a: 638c          c.ld a1,0(a5)
 1016c: 6790          c.ld a2,8(a5)
 1016e: 6b94          c.ld a3,16(a5)
```

10170:	6f98	c.ld a4,24(a5)
10172:	739c	c.ld a5,32(a5)
10174:	e42e	c.sdsp a1,8(sp)
10176:	e832	c.sdsp a2,16(sp)
10178:	ec36	c.sdsp a3,24(sp)
1017a:	f03a	c.sdsp a4,32(sp)
1017c:	f43e	c.sdsp a5,40(sp)
1017e:	6571	c.lui a0,0x1c
10180:	59850513	addi a0,a0,1432 # 1c598 <__clzdi2+0x5e>
10184:	2f0000ef	jal ra,10474 <puts>
10188:	0020	c.addi4spn s0,sp,8
1018a:	03010913	addi s2,sp,48
1018e:	84a2	c.mv s1,s0
10190:	69f1	c.lui s3,0x1c
10192:	408c	c.lw a1,0(s1)
10194:	5a898513	addi a0,s3,1448 # 1c5a8 <__clzdi2+0x6e>
10198:	22a000ef	jal ra,103c2 <printf>
1019c:	0491	c.addi s1,4
1019e:	ff249ae3	bne s1,s2,10192 <main+0x3a>
101a2:	002c	c.addi4spn a1,sp,8
101a4:	4529	c.li a0,10
101a6:	030000ef	jal ra,101d6 <SortingInsertion>
101aa:	6571	c.lui a0,0x1c
101ac:	5b050513	addi a0,a0,1456 # 1c5b0 <__clzdi2+0x76>
101b0:	2c4000ef	jal ra,10474 <puts>
101b4:	64f1	c.lui s1,0x1c
101b6:	400c	c.lw a1,0(s0)
101b8:	5a848513	addi a0,s1,1448 # 1c5a8 <__clzdi2+0x6e>
101bc:	206000ef	jal ra,103c2 <printf>
101c0:	0411	c.addi s0,4
101c2:	ff241ae3	bne s0,s2,101b6 <main+0x5e>
101c6:	4501	c.li a0,0
101c8:	60e6	c.ldsp ra,88(sp)
101ca:	6446	c.ldsp s0,80(sp)
101cc:	64a6	c.ldsp s1,72(sp)
101ce:	6906	c.ldsp s2,64(sp)
101d0:	79e2	c.ldsp s3,56(sp)
101d2:	6125	c.addi16sp sp,96
101d4:	8082	c.jr ra

0000000000101d6 <SortingInsertion>:

101d6:	4785	c.li a5,1
101d8:	04a7d663	bge a5,a0,10224 <SortingInsertion+0x4e>
101dc:	00458813	addi a6,a1,4
101e0:	fff5031b	addiw t1,a0,-1
101e4:	4881	c.li a7,0
101e6:	557d	c.li a0,-1
101e8:	a809	c.j 101fa <SortingInsertion+0x24>
101ea:	0785	c.addi a5,1
101ec:	078a	c.slli a5,0x2
101ee:	97ae	c.add a5,a1

101f0:	c390	c.sw a2,0(a5)
101f2:	2885	c.addiw a7,1
101f4:	0811	c.addi a6,4
101f6:	02688763	beq a7,t1,10224 <SortingInsertion+0x4e>
101fa:	00082603	lw a2,0(a6)
101fe:	0008879b	addiw a5,a7,0
10202:	fe07c4e3	blt a5,zero,101ea <SortingInsertion+0x14>
10206:	ffc82683	lw a3,-4(a6)
1020a:	fed650e3	bge a2,a3,101ea <SortingInsertion+0x14>
1020e:	8742	c.mv a4,a6
10210:	c314	c.sw a3,0(a4)
10212:	37fd	c.addiw a5,-1
10214:	fca78be3	beq a5,a0,101ea <SortingInsertion+0x14>
10218:	1771	c.addi a4,-4
1021a:	ffc72683	lw a3,-4(a4)
1021e:	fed649e3	blt a2,a3,10210 <SortingInsertion+0x3a>
10222:	b7e1	c.j 101ea <SortingInsertion+0x14>
10224:	8082	c.jr ra

Мы видим, что адресация для вызовов функций изменилась на абсолютную.

3.Создание статической библиотеки

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

```
D:\lab4>riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -O1 -c sorting.c -o sorting.o
D:\lab4>riscv64-unknown-elf-ar -rsc lib.a sorting.o
```

Рассмотрим список символов библиотеки:

Листинг 3.1 Список символов `lib.a`

```
D:\lab4>riscv64-unknown-elf-nm lib.a

sorting.o:
000000000000004e t .L1
0000000000000014 t .L3
000000000000003a t .L4
0000000000000024 t .L5
0000000000000000 T SortingInsertion
```

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

Теперь, имея собранную библиотеку, создадим исполняемый файл тестовой программы ‘*main.c*’ с помощью следующей команды:

```
D:\lab4>riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -O1 main.c lib.a -o main.out
```

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

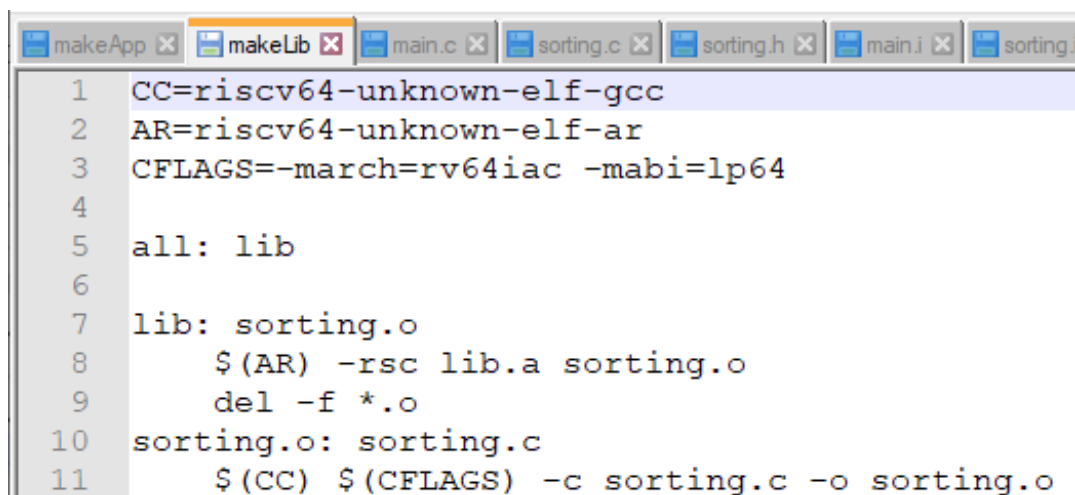
Листинг 3.1 Фрагмент списка символов `main.out`.

```
D:\lab4>riscv64-unknown-elf-objdump -t main.out >main.ds
```

266	0000000000001f4c0	g	O .sbss	0000000000000008	__malloc_max_total_mem
267	0000000000001ada2	g	F .text	000000000000000c	__swbuf
268	00000000000016e2e	g	F .text	0000000000000008	__sclose
269	00000000000019cba	g	F .text	000000000000000a	fclose
270	00000000000015308	g	F .text	0000000000000060	_malloc_r
271	00000000000019adc	g	F .text	0000000000000024	__ascii_wctomb
272	000000000000101d6	g	F .text	0000000000000050	SortingInsertion
273	00000000000012f74	g	F .text	0000000000000008	_fwalk
274	00000000000015968	g	F .text	000000000000000a	_mbtowc_r

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

Листинг 3.2. Makefile для создания статической библиотеки “`makeLib`”



```
1 CC=riscv64-unknown-elf-gcc
2 AR=riscv64-unknown-elf-ar
3 CFLAGS=-march=rv64iac -mabi=lp64
4
5 all: lib
6
7 lib: sorting.o
8     $(AR) -rsc lib.a sorting.o
9     del -f *.o
10 sorting.o: sorting.c
11     $(CC) $(CFLAGS) -c sorting.c -o sorting.o
```

Листинг 3.3. Makefile для сборки исполняемого файла “makeApp”

```
makeApp x makeLib x main.c x sorting.c x sorting.h x main.i x sorting.i x
1 TARGET=main.out
2 CC=riscv64-unknown-elf-gcc
3 CFLAGS=-march=rv64iac -mabi=lp64
4
5 all:
6     make -f makeLib
7     $(CC) $(CFLAGS) main.c lib.a -o $(TARGET)
8     del -f *.o *.a
```

Теперь с помощью GNU make выполним сначала makeLib, а затем makeApp, для создания библиотеки.

```
D:\lab4>mingw32-make -f makeLib
riscv64-unknown-elf-ar -rsc lib.a sorting.o
del -f *.o

D:\lab4>mingw32-make -f makeApp
mingw32-make -f makeLib
mingw32-make[1]: Entering directory 'D:/lab4'
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -c sorting.c -o sorting.o
riscv64-unknown-elf-ar -rsc lib.a sorting.o
del -f *.o
mingw32-make[1]: Leaving directory 'D:/lab4'
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 main.c lib.a -o main.out
del -f *.o *.a
```

Рис. 3.1 Выполнение make файлов.

Посмотрим таблицу символов полученного с помощью makefile исполняемого файла:

Листинг 3.3 Фрагмент списка символов main.out (makefile).

```
D:\lab4>riscv64-unknown-elf-objdump -t main.out >main.ds
```

270	000000000000153bc	g	F .text	0000000000000660	_malloc_r
271	00000000000019b90	g	F .text	0000000000000024	__ascii_wctomb
272	00000000000010224	g	F .text	00000000000000b6	SortingInsertion
273	00000000000013028	g	F .text	000000000000008c	_fwalk
274	00000000000015a1c	g	F .text	000000000000000a	_mbtowc_r
...

Мы видим, что исполняемый файл аналогичен созданному в терминале файлу.

Вывод

В ходе лабораторной работы изучена пошаговая компиляция программы на языке С. Также была создана статическая библиотека и произведена сборка программы с помощью Makefile.