Санкт-Петербургский политехнический университет Петра Великого

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Отчёт по лабораторной работе №4**

**Дисциплина**: Низкоуровневое программирование

**Тема**: раздельная компиляция

Выполнил студент гр. 3530901/90003 В.Д. Коротких

(подпись)

Преподаватель А. О. Алексюк

(подпись)

“ ” 2021 г.

Санкт-Петербург

2021

Оглавление

[1. Техническое задание 3](#_Toc69584074)

[2. Метод решения 3](#_Toc69584075)

[3. Решение 3](#_Toc69584076)

[3.1. Анализ выхода препроцессора: 5](#_Toc69584077)

[3.2. Анализ выхода компилятора: 6](#_Toc69584078)

[3.3. Анализ состава и содержимого секций, таблицы символов, таблицы перемещений и отладочной информации, содержащейся в объектных файлах и исполняемом файле: 8](#_Toc69584079)

[3.4. Содержимое таблицы перемещений: 12](#_Toc69584086)

[3.5. Результат компоновки 14](#_Toc69584087)

[3.6. Анализ отладочной информации 16](#_Toc69584158)

[3.7. Выделение разработанной функции в статическую библиотеку 17](#_Toc69584160)

[3.8. Создание и использование полученной статической библиотеки 17](#_Toc69584166)

[4. Результаты 19](#_Toc69584180)

# Техническое задание

1. На языке С разработать функцию, реализующую поиск k-ой порядковой статистики in-place. Поместить определение функции в отдельный исходный файл, оформить заголовочный файл. Разработать тестовую программу на языке С.
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.

# Метод решения

Поиск k-ой порядковой статистики выполнен на основе сортировки обменом: упорядочиваются первые k элементов, после чего k-ый (максимальный на текущий момент) элемент выводится в консоль.

# Решение

Напишем программу на языке СИ:

kSearch.h

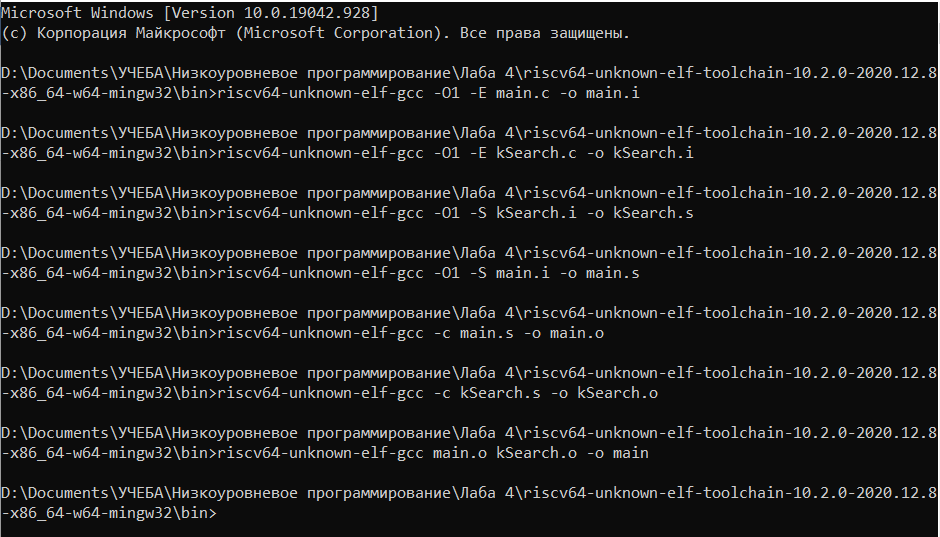
|  |
| --- |
| #ifndef LOWLEVEL\_KSEARCH\_H  #define LOWLEVEL\_KSEARCH\_H  #include <stdio.h>  int kSearch(int a[], int length, int k);  #endif // LOWLEVEL\_KSEARCH\_H |

kSearch.c

|  |
| --- |
| #include "kSearch.h"  int kSearch(int a[], int length, int k) {  int min;  for(int i = 0; i < k; i++) {  min = a[i];  for (int j = i + 1; j < length; j++) {  if (a[j] < min) {  int tmp = min;  min = a[j];  a[j] = tmp;  }  }  a[i] = min;  }  return min;  } |

main.c

|  |
| --- |
| #include "kSearch.h"  int main() {  int a[11] = {1, 4, 8, 8, 5, 3, 9, 2, 10, 0, 7};  int length = sizeof(a) / sizeof(int);  int k = 8;  printf("Array: [");  for(int i = 0; i < length; i++) {  printf("%d", a[i]);  if (i != length - 1) printf(", ");  }  printf("]\nk=%d\n", k);  printf("result=%d", kSearch(a, length, k));  return 0;  } |



**Рис.1.** Сборка программы по этапам

## Анализ выхода препроцессора:

Директивы, прописанные в заголовочном файле, определяют вставку стандартной библиотеки ввода-вывода языка Си. Пользовательская часть кода практически не меняется:

Часть файла main.i

|  |
| --- |
| # 4 "kSearch.h" 2  # 4 "kSearch.h"  int kSearch(int a[], int length, int k);  # 2 "main.c" 2  int main() {  int a[11] = {1, 4, 8, 8, 5, 3, 9, 2, 10, 0, 7};  int length = sizeof(a) / sizeof(int);  int k = 8;  printf("Array: [");  for(int i = 0; i < length; i++) {  printf("%d", a[i]);  if (i != length - 1) printf(", ");  }  printf("]\nk=%d\n", k);  printf("result=%d", kSearch(a, length, k));  return 0;  } |

Аналогично происходит препроцессирование функции:

Часть файла kSearch.i:

|  |
| --- |
| # 4 "kSearch.h" 2  # 4 "kSearch.h"  int kSearch(int a[], int length, int k);  # 2 "kSearch.c" 2  int kSearch(int a[], int length, int k) {  int min;  for(int i = 0; i < k; i++) {  min = a[i];  for (int j = i + 1; j < length; j++) {  if (a[j] < min) {  int tmp = min;  min = a[j];  a[j] = tmp;  }  }  a[i] = min;  }  return min;  } |

## Анализ выхода компилятора:

main.s

|  |
| --- |
| .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 "]\nk=%d\n"  .align 3  .LC5:  .string "result=%d"  .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 a0,0(a5)  ld a1,8(a5)  ld a2,16(a5)  ld a3,24(a5)  ld a4,32(a5)  sd a0,0(sp)  sd a1,8(sp)  sd a2,16(sp)  sd a3,24(sp)  sd a4,32(sp)  lw a5,40(a5)  sw a5,40(sp)  lui a0,%hi(.LC1)  addi a0,a0,%lo(.LC1)  call printf  lw a1,0(sp)  lui a0,%hi(.LC2)  addi a0,a0,%lo(.LC2)  call printf  addi s0,sp,4  addi s3,sp,44  lui s2,%hi(.LC3)  lui s1,%hi(.LC2)  .L2:  addi a0,s2,%lo(.LC3)  call printf  lw a1,0(s0)  addi a0,s1,%lo(.LC2)  call printf  addi s0,s0,4  bne s0,s3,.L2  li a1,8  lui a0,%hi(.LC4)  addi a0,a0,%lo(.LC4)  call printf  li a2,8  li a1,11  mv a0,sp  call kSearch  mv a1,a0  lui a0,%hi(.LC5)  addi a0,a0,%lo(.LC5)  call printf  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 1  .word 4  .word 8  .word 8  .word 5  .word 3  .word 9  .word 2  .word 10  .word 0  .word 7  .ident "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0" |

kSearch.s

|  |
| --- |
| .file "kSearch.c"  .option nopic  .attribute arch, "rv64i2p0\_m2p0\_a2p0\_f2p0\_d2p0\_c2p0"  .attribute unaligned\_access, 0  .attribute stack\_align, 16  .text  .align 1  .globl kSearch  .type kSearch, @function  kSearch:  ble a2,zero,.L7  addi a6,a0,4  li t1,0  li a7,0  addiw t5,a1,-1  addi t4,a0,8  j .L6  .L4:  addi a5,a5,4  beq a5,a3,.L3  .L5:  lw a4,0(a5)  bge a4,a0,.L4  sw a0,0(a5)  mv a0,a4  j .L4  .L3:  sw a0,-4(t3)  addi t1,t1,1  addi a6,a6,4  beq a2,a7,.L10  .L6:  mv t3,a6  lw a0,-4(a6)  addiw a5,a7,1  sext.w a7,a5  bge a7,a1,.L3  subw a3,t5,a5  slli a3,a3,32  srli a3,a3,32  add a3,a3,t1  slli a3,a3,2  add a3,a3,t4  mv a5,a6  j .L5  .L10:  ret  .L7:  li a0,0  ret  .size kSearch, .-kSearch  .ident "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0" |

В программе main выполняется обращение к подпрограмме kSeacrh (значение регистра ra, содержащее адрес возврата из main, сохраняется на время вызова в стеке). Следует отметить, что символ kSeacrh используется в файле main.s, но никак не определяется.

## Анализ состава и содержимого секций, таблицы символов, таблицы перемещений и отладочной информации, содержащейся в объектных файлах и исполняемом файле:

Сформированный ассемблером объектный файл main.o и kSearch.o должны содержать коды инструкций, таблицу символов и таблицу перемещений. 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:

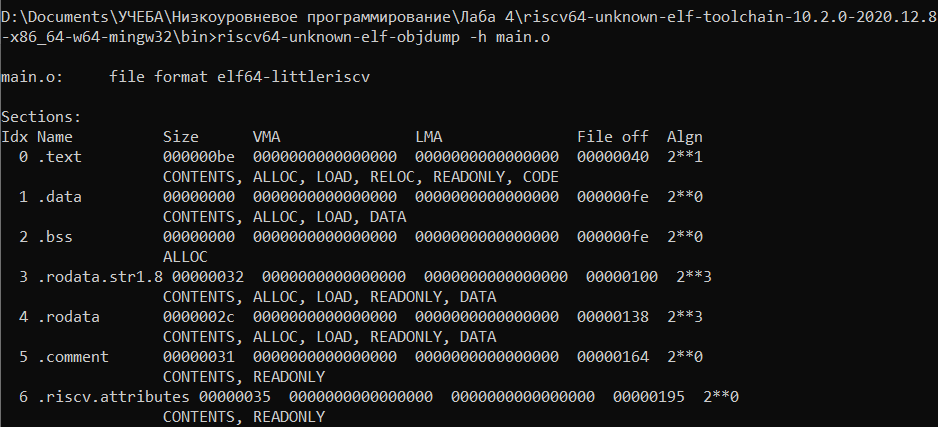


Рис.2. Содержимое заголовков секций main.o

В файле имеются следующие секции:

•.text – секция кода;

•.data – секция инициализированных данных;

•.bss – секция данных, инициализированных нулями;

•.rodata – секция неизменяемых данных;

•.rodata.str1.4 –подсекция неизменяемых данных, используется компилятором для хранения дополнительной информации (например, о типе данных) для компоновщика;

•.comment – секция данных о версиях;

•.riscv.attributes – атрибуты для указания определенных свойств функции (в помощь компилятору для проверок и оптимизации кода).

Значения в столбце size приведены в 16-ричной системе счисления.

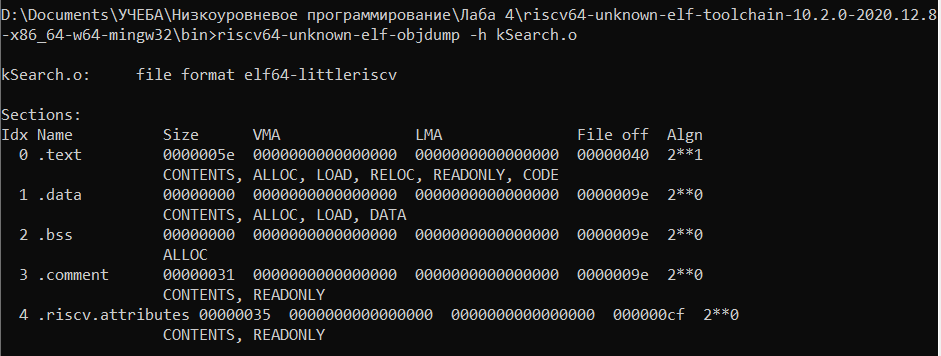


Рис.3. Содержимое заголовков секций kSearch.o

Tаблицы символов объектных файлов main.o и kSearch.o:

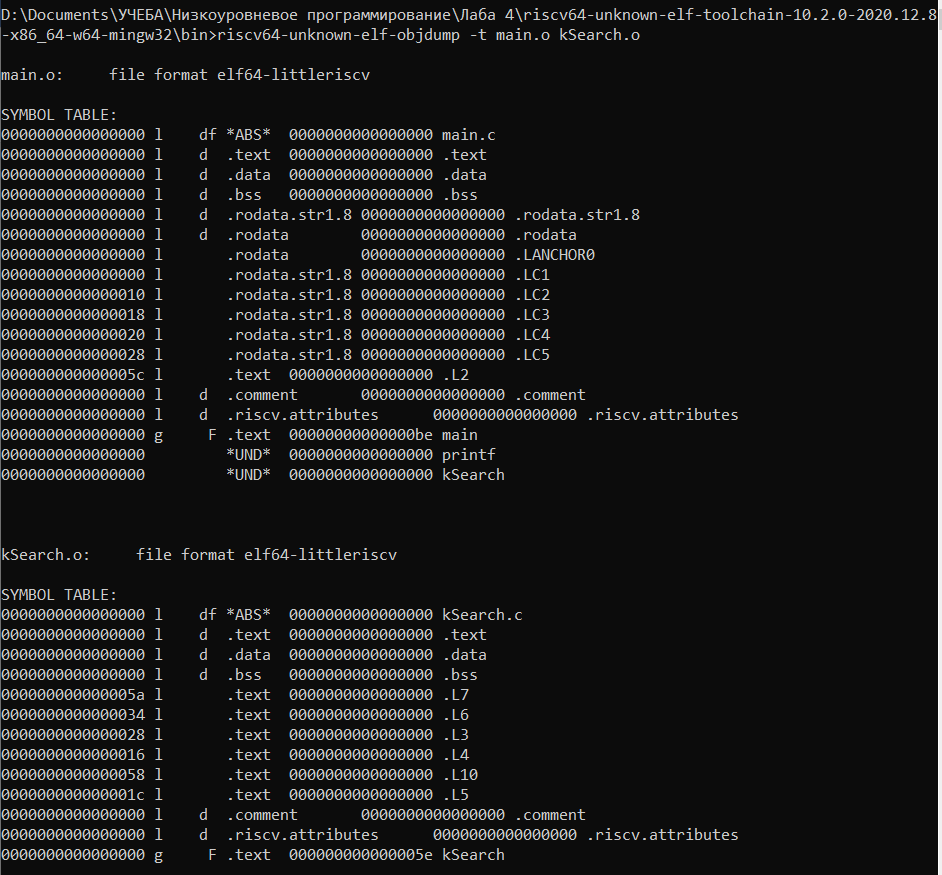


Рис.4. Содержимое таблиц символов объектных файлов

Как и ожидалось таблица содержит 2 глобальные (флаг g) функции (флаг F) – main и kSearch, также два неопределенных (UND) символа.

UND означает, что символы printf и kSearch 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.

Изучим содержимое секции .text объектных файлов main.o и kSearch.o:

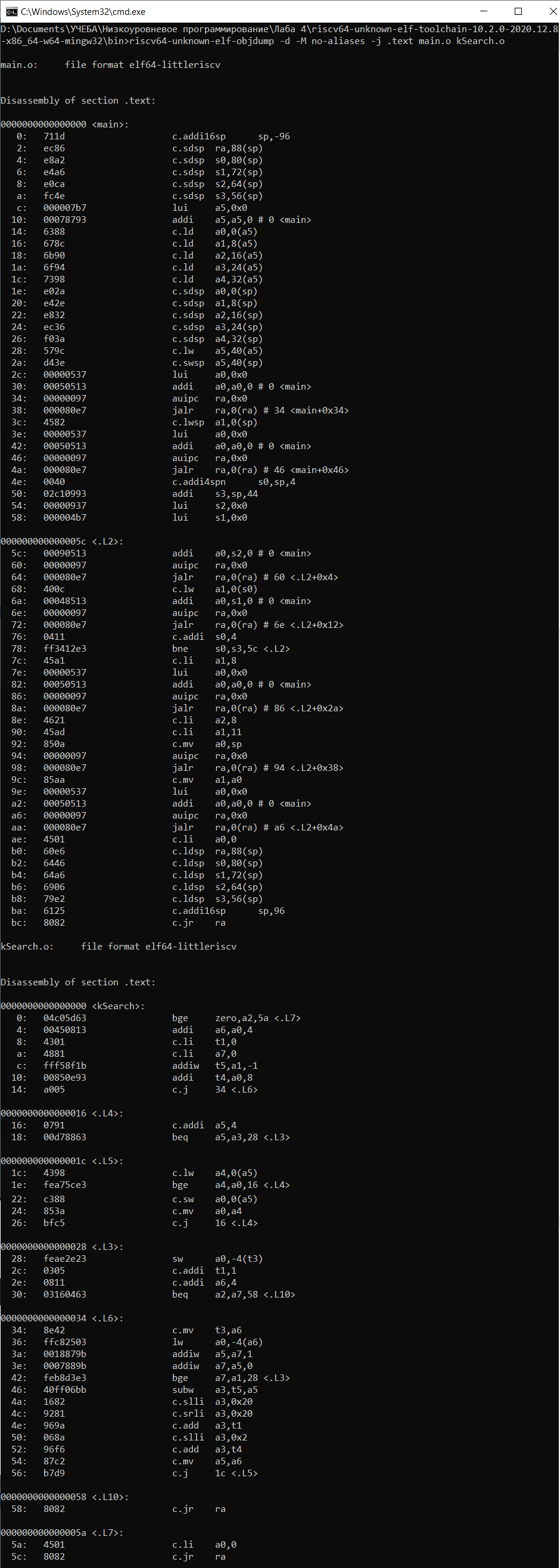


Рис.5. Cодержимое секции .text объектного файла main.o

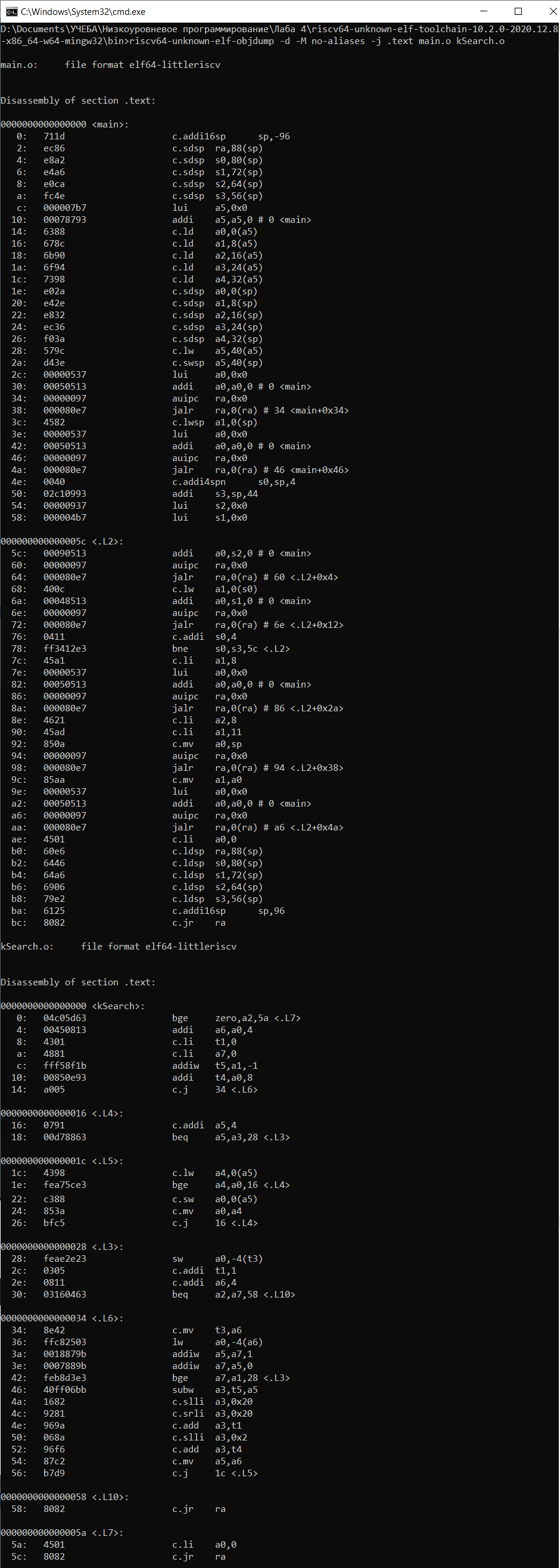
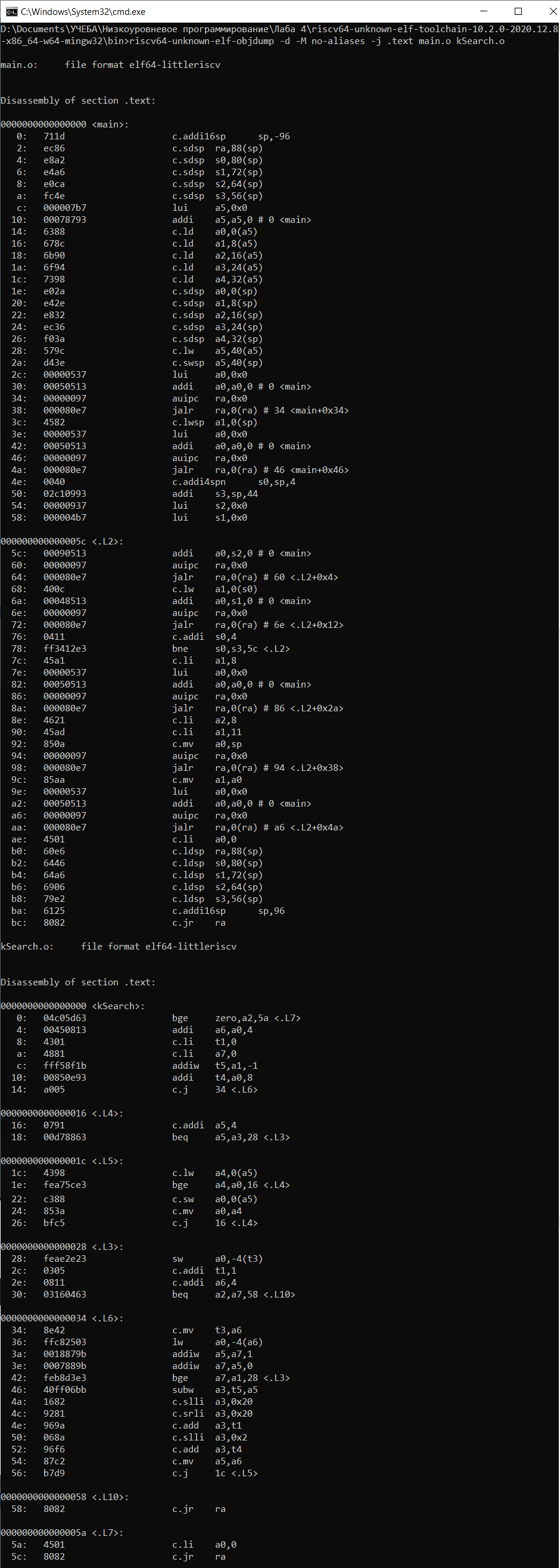


Рис.6. Cодержимое секции .text объектного файла kSearch.o

Результат дизассемблирования kSearch.o интереса не представляет, в отличие от результата дизассемблирования main.o: сравнивая его с main.s, можно понять, что псевдоинструкция вызова подпрограммы kSearch, транслировалась ассемблером в следующую пару инструкций:



Результатом выполнения этой пары инструкций станет переход на адрес .L2+0x38 (5c+38=94) - произойдет зацикливание. Это показано в выводе дизассемблера. Загадочное поведение ассемблера объясняется очень просто: ассемблер не имел возможности определить целевой адрес перехода (кроме того, что этот адрес обозначен символом kSearch), поэтому не мог сформировать корректную инструкцию (пару инструкций) передачи управления. В результате была сформирована пара инструкций с некорректными (нулевыми) значениями непосредственных операндов. Для получения исполняемого кода эта пара инструкций должна быть исправлена компоновщиком.



## Содержимое таблицы перемещений:

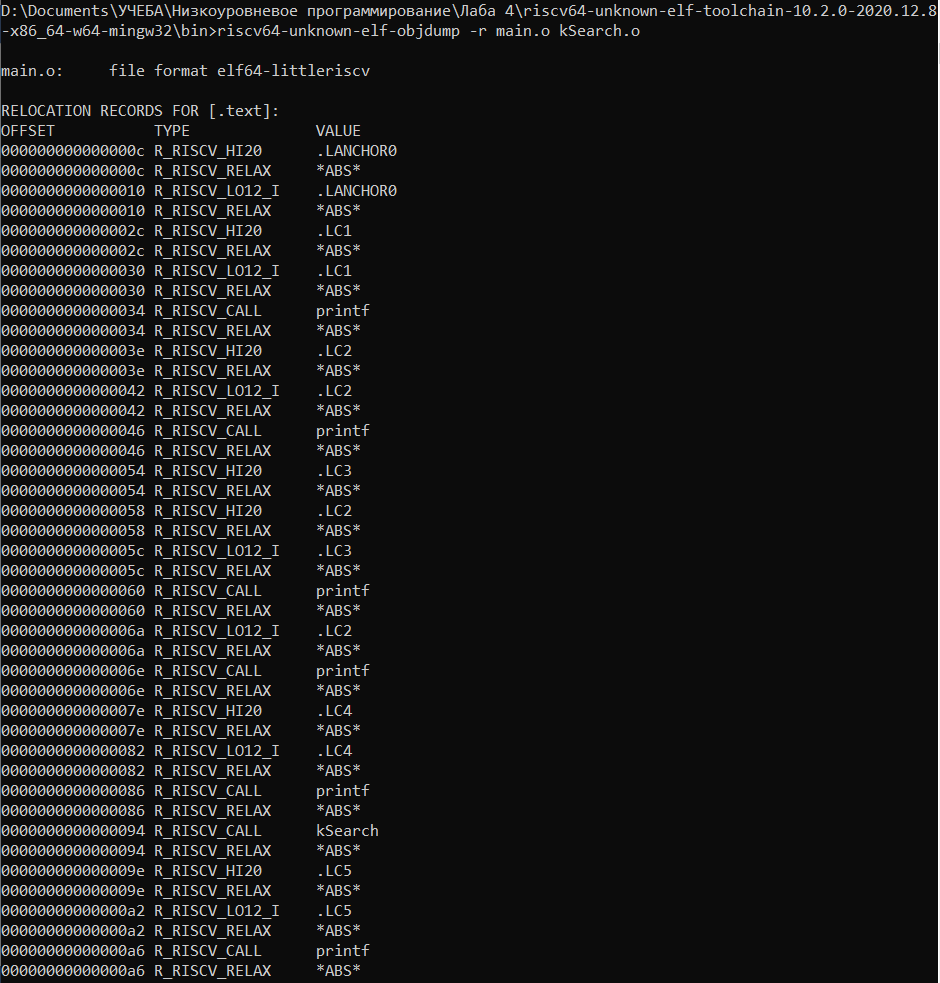


Рис.7. Таблица перемещений main.o

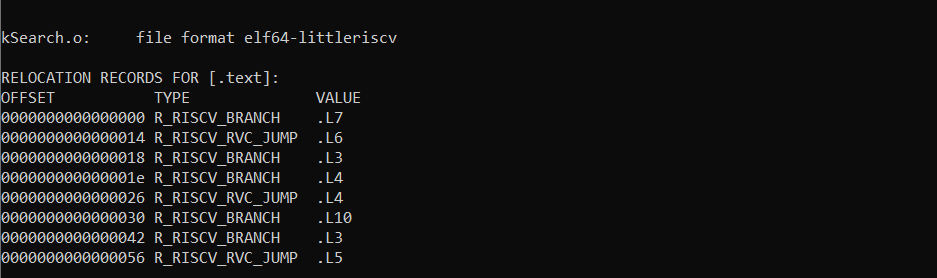
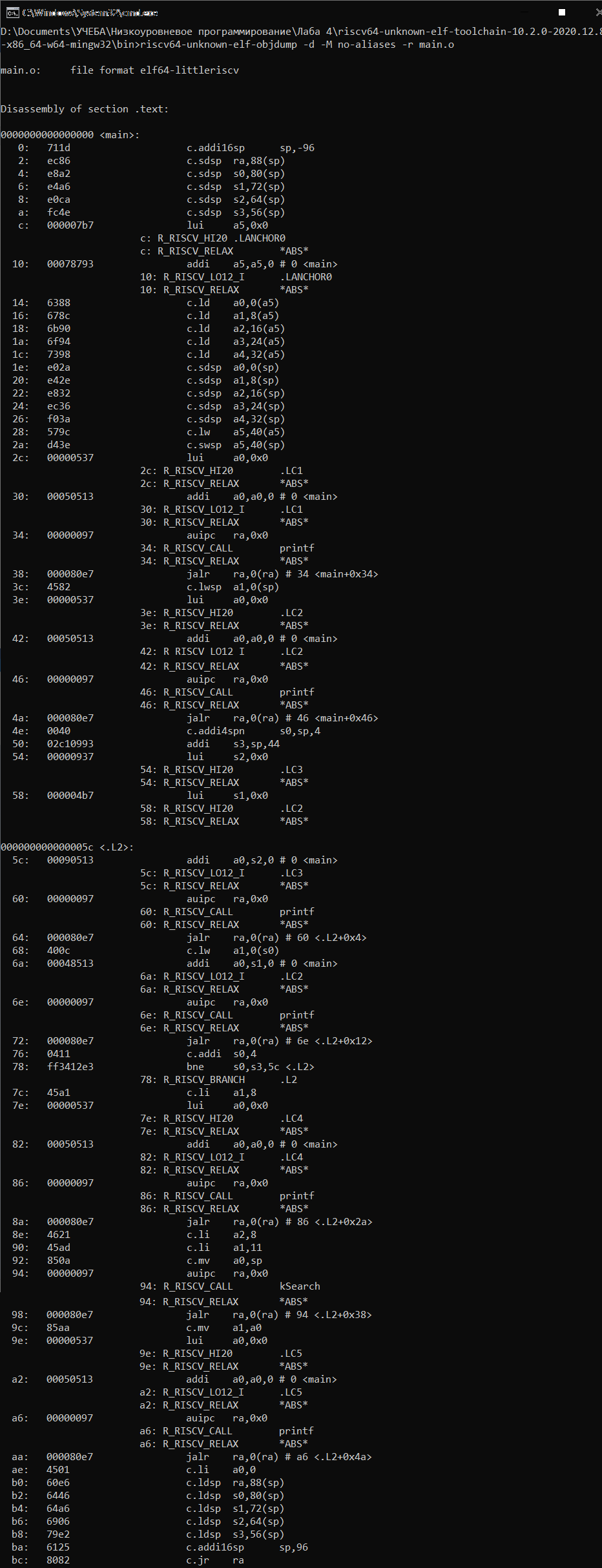


Рис.8. Таблица перемещений kSearch.o

Информация обо всех «неоконченных» инструкциях передается ассемблером компоновщику посредством таблицы перемещений.

Содержимое kSearch.o не требует модификации, поэтому не содержит записей о перемещениях (relocation entries). В файле же main.o имеется 7 записей, среди которых есть запись относящаяся к адресу 94 (как мы видели выше, по этому адресу в main.o находится инструкция пары auipc+jalr). Дизассемблирование и вывод таблицы перемещений можно совместить



Для того чтобы внести необходимые исправления, требуется знать, что исправить, как исправить и какой символ следует использовать, именно эта информация и содержится в записях о перемещениях. Так, в первой записи таблице перемещений указано, что по адресу 2a следует исправить пару инструкций (тип перемещения “R\_RISCV\_CALL”) так, чтобы результат соответствовал вызову подпрограммы kSearch. Типы перемещений специфичны для каждой архитектуры системы команд и обычно определены в ABI (Application Binary Interface).

Вторая запись таблицы перемещений специфична для средств разработки RISC-V. 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.

## Результат компоновки

riscv64-unknown-elf-gcc -Wl,--no-relax main.o kSearch.o -o main

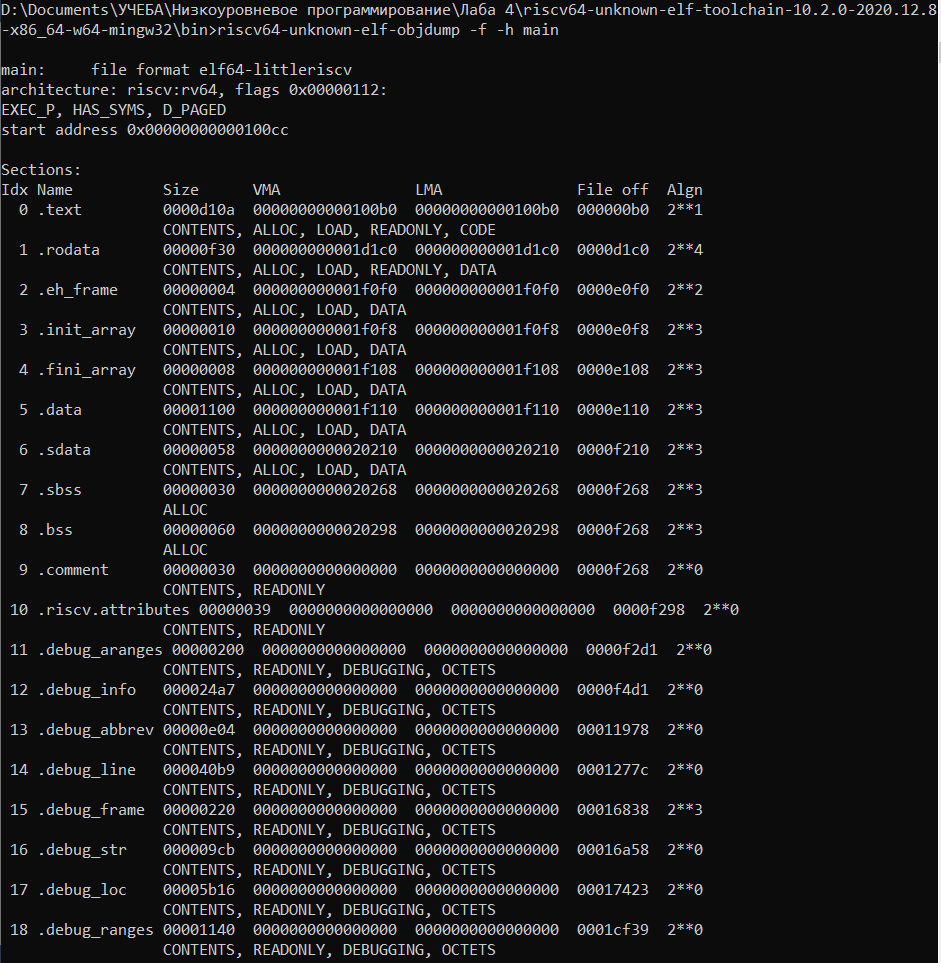
riscv64-unknown-elf-objdump -j .text -d -M no-aliases main >main.ds

main.ds (строки 79-180)

|  |
| --- |
| 0000000000010188 <main>:  10188: 711d c.addi16sp sp,-96  1018a: ec86 c.sdsp ra,88(sp)  1018c: e8a2 c.sdsp s0,80(sp)  1018e: e4a6 c.sdsp s1,72(sp)  10190: e0ca c.sdsp s2,64(sp)  10192: fc4e c.sdsp s3,56(sp)  10194: 0001d7b7 lui a5,0x1d  10198: 1f878793 addi a5,a5,504 # 1d1f8 <\_\_clzdi2+0x6e>  1019c: 6388 c.ld a0,0(a5)  1019e: 678c c.ld a1,8(a5)  101a0: 6b90 c.ld a2,16(a5)  101a2: 6f94 c.ld a3,24(a5)  101a4: 7398 c.ld a4,32(a5)  101a6: e02a c.sdsp a0,0(sp)  101a8: e42e c.sdsp a1,8(sp)  101aa: e832 c.sdsp a2,16(sp)  101ac: ec36 c.sdsp a3,24(sp)  101ae: f03a c.sdsp a4,32(sp)  101b0: 579c c.lw a5,40(a5)  101b2: d43e c.swsp a5,40(sp)  101b4: 0001d537 lui a0,0x1d  101b8: 1c050513 addi a0,a0,448 # 1d1c0 <\_\_clzdi2+0x36>  101bc: 00000097 auipc ra,0x0  101c0: 24c080e7 jalr ra,588(ra) # 10408 <printf>  101c4: 4582 c.lwsp a1,0(sp)  101c6: 0001d537 lui a0,0x1d  101ca: 1d050513 addi a0,a0,464 # 1d1d0 <\_\_clzdi2+0x46>  101ce: 00000097 auipc ra,0x0  101d2: 23a080e7 jalr ra,570(ra) # 10408 <printf>  101d6: 0040 c.addi4spn s0,sp,4  101d8: 02c10993 addi s3,sp,44  101dc: 0001d937 lui s2,0x1d  101e0: 0001d4b7 lui s1,0x1d  101e4: 1d890513 addi a0,s2,472 # 1d1d8 <\_\_clzdi2+0x4e>  101e8: 00000097 auipc ra,0x0  101ec: 220080e7 jalr ra,544(ra) # 10408 <printf>  101f0: 400c c.lw a1,0(s0)  101f2: 1d048513 addi a0,s1,464 # 1d1d0 <\_\_clzdi2+0x46>  101f6: 00000097 auipc ra,0x0  101fa: 212080e7 jalr ra,530(ra) # 10408 <printf>  101fe: 0411 c.addi s0,4  10200: ff3412e3 bne s0,s3,101e4 <main+0x5c>  10204: 45a1 c.li a1,8  10206: 0001d537 lui a0,0x1d  1020a: 1e050513 addi a0,a0,480 # 1d1e0 <\_\_clzdi2+0x56>  1020e: 00000097 auipc ra,0x0  10212: 1fa080e7 jalr ra,506(ra) # 10408 <printf>  10216: 4621 c.li a2,8  10218: 45ad c.li a1,11  1021a: 850a c.mv a0,sp  1021c: 00000097 auipc ra,0x0  10220: 02a080e7 jalr ra,42(ra) # 10246 <kSearch>  10224: 85aa c.mv a1,a0  10226: 0001d537 lui a0,0x1d  1022a: 1e850513 addi a0,a0,488 # 1d1e8 <\_\_clzdi2+0x5e>  1022e: 00000097 auipc ra,0x0  10232: 1da080e7 jalr ra,474(ra) # 10408 <printf>  10236: 4501 c.li a0,0  10238: 60e6 c.ldsp ra,88(sp)  1023a: 6446 c.ldsp s0,80(sp)  1023c: 64a6 c.ldsp s1,72(sp)  1023e: 6906 c.ldsp s2,64(sp)  10240: 79e2 c.ldsp s3,56(sp)  10242: 6125 c.addi16sp sp,96  10244: 8082 c.jr ra  0000000000010246 <kSearch>:  10246: 04c05d63 bge zero,a2,102a0 <kSearch+0x5a>  1024a: 00450813 addi a6,a0,4  1024e: 4301 c.li t1,0  10250: 4881 c.li a7,0  10252: fff58f1b addiw t5,a1,-1  10256: 00850e93 addi t4,a0,8  1025a: a005 c.j 1027a <kSearch+0x34>  1025c: 0791 c.addi a5,4  1025e: 00d78863 beq a5,a3,1026e <kSearch+0x28>  10262: 4398 c.lw a4,0(a5)  10264: fea75ce3 bge a4,a0,1025c <kSearch+0x16>  10268: c388 c.sw a0,0(a5)  1026a: 853a c.mv a0,a4  1026c: bfc5 c.j 1025c <kSearch+0x16>  1026e: feae2e23 sw a0,-4(t3)  10272: 0305 c.addi t1,1  10274: 0811 c.addi a6,4  10276: 03160463 beq a2,a7,1029e <kSearch+0x58>  1027a: 8e42 c.mv t3,a6  1027c: ffc82503 lw a0,-4(a6)  10280: 0018879b addiw a5,a7,1  10284: 0007889b addiw a7,a5,0  10288: feb8d3e3 bge a7,a1,1026e <kSearch+0x28>  1028c: 40ff06bb subw a3,t5,a5  10290: 1682 c.slli a3,0x20  10292: 9281 c.srli a3,0x20  10294: 969a c.add a3,t1  10296: 068a c.slli a3,0x2  10298: 96f6 c.add a3,t4  1029a: 87c2 c.mv a5,a6  1029c: b7d9 c.j 10262 <kSearch+0x1c>  1029e: 8082 c.jr ra  102a0: 4501 c.li a0,0  102a2: 8082 c.jr ra |

Прежде всего можно видеть, что в результат компоновки попало содержимое обоих объектных файлов – main.o и kSeacrh.o. Инструкции подпрограммы kSeacrh начинаются с адреса 1024616, и пара инструкций auipc+jalr, вызывающих подпрограмму kSeacrh соответствующим образом откорректированы.

## Анализ отладочной информации

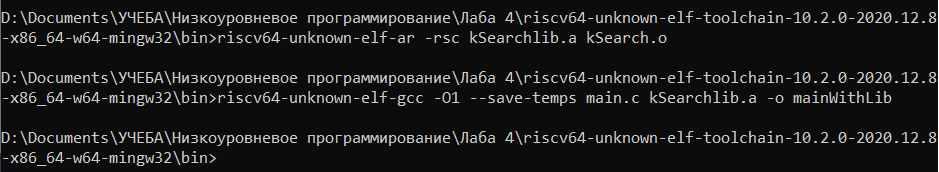


Сформированный исполняемый файл содержит информацию для отладки (в секциях ***.debug…***), полную таблицу символов и сведения о версиях средств разработки.

Встреченные разделы DWARF:

* ***.debug\_abbrev*** – сокращения , используемые в ***.debug\_info*** разделе;
* ***.debug\_aranges***– таблица поиска для сопоставления адресов с единицами компиляции;
* ***.debug\_frame*** – информация о кадре вызова;
* ***.debug\_info*** – раздел основной информации DWARF;
* ***.debug\_line*** – информация о номере строки;
* ***.debug\_loc*** – списки местоположений, используемые в атрибутах ***DW\_AT\_location***;
* ***.debug\_ranges*** – диапазоны адресов, используемые в атрибутах ***DW\_AT\_ranges***;
* ***.debug\_str*** – таблица строк, используемая в ***.debug\_info***.

## Выделение разработанной функции в статическую библиотеку

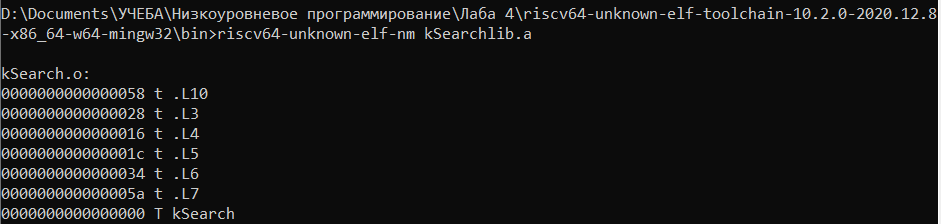


|  |
| --- |
| …  00000000000102d0 g F .text 00000000000000aa memset  0000000000010156 g F .text 0000000000000094 main  …  00000000000101ea g F .text 000000000000005e kSearch  … |

Таблица символов полученного исполняемого файла

Как и следовало ожидать, в состав исполняемого файла вошло содержимое всех объектных файлов, указанных в команде сборки.

## Создание и использование полученной статической библиотеки



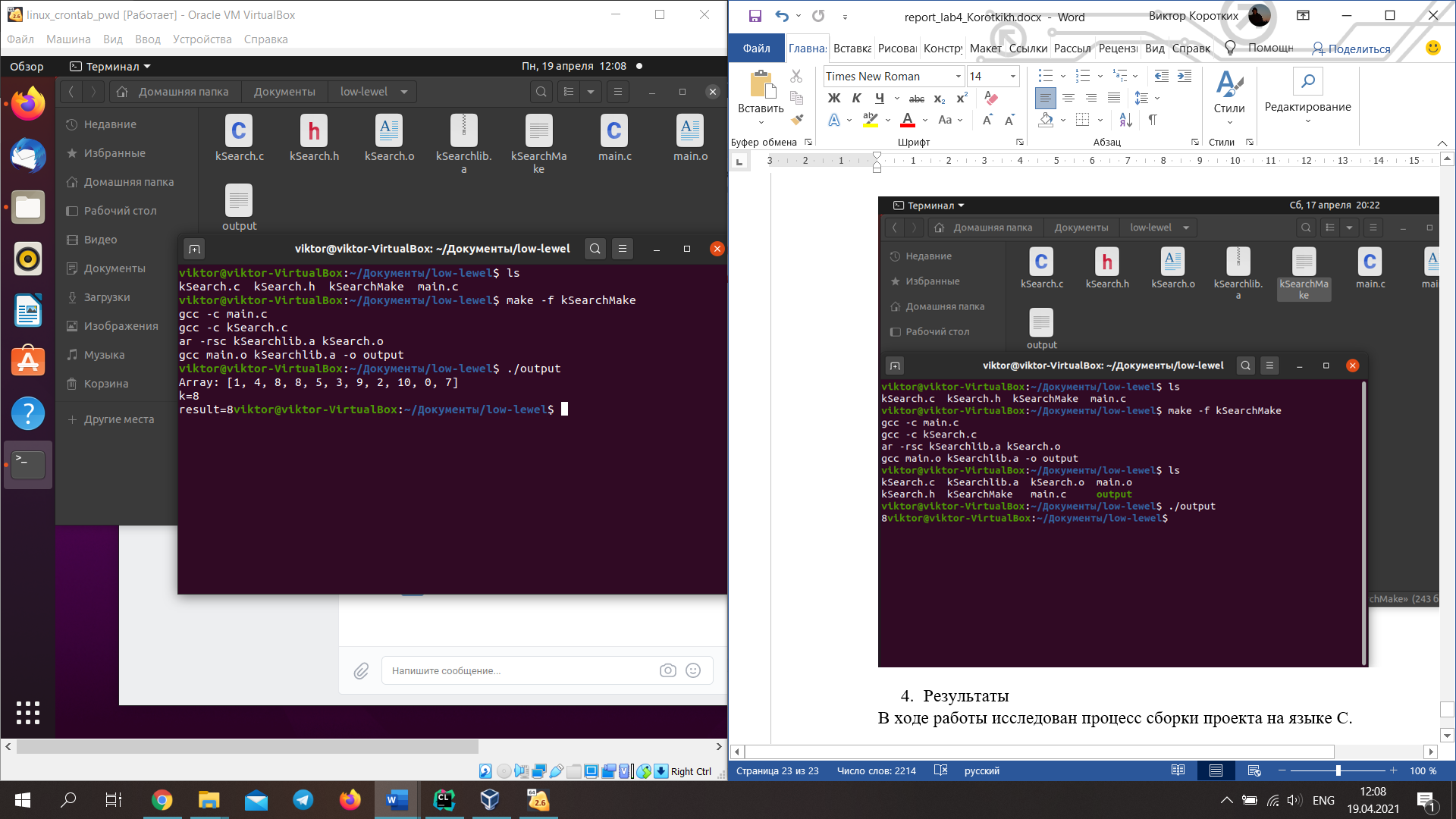
Список символов библиотеки

Создадим make-файл:

|  |
| --- |
| output: main.o kSearchlib.a  gcc main.o kSearchlib.a -o output    main.o: main.c  gcc -c main.c    kSearchlib.a: kSearch.o kSearch.h  ar -rsc kSearchlib.a kSearch.o    kSearch.o:  gcc -c kSearch.c    clean:  rm \*.o \*.a output |

Что происходит в ***makefile***:

* Создаём объектный файл ***main.o*** из исходного ***main.c***;
* Создаём объектный файл ***kSearch.o*** из исходного ***kSearch.c***;
* Архивируем объектный файл ***kSearch.o*** (создаём статическую библиотеку ***kSearchlib.a***);
* Компонуем статическую библиотеку ***kSearchlib.a*** с объектным файлом ***main.o***, получаем исполняемый файл ***output***.



# Результаты

В ходе работы исследован процесс сборки проекта на языке C.

Он состоит из:

• Препроцессирования исходного <filename>.c в <filename>.i;

• Компиляции полученного <filename>.i в файл ассемблера <filename>.s;

• Ассемблирования <filename>.s в объектный файл <filename>.o;

• Компоновки объектного файла <filename>.o в исполняемый файл.

Также были рассмотрены makefile’ы, которые существенно упрощают процесс сборки.

Вместо того, чтобы поочередно набирать команды в терминале, используется единственная команда make, которая по инструкциям в makefile’е собирает программу в автоматическом режиме.