

Отчёт по лабораторной работе № 4

Дисциплина: Низкоуровневое программирование

Тема: Раздельная компиляция

Вариант: 14

Выполнил студент гр. 3530901/90002 _____ С.А. Федоров
(подпись)

Принял преподаватель _____ Д.С. Степанов
(подпись)

“ _____ ” _____ 2021 г.

Цели работы:

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.

Начальные данные для 14 варианта

Определение наиболее часто встречающегося в массиве значения.

1. Подробный алгоритм для решения поставленной задачи

- 1) Реализовать функцию для сортировки массива. Была выбрана сортировка пузырьком. В этой сортировке требуется написать два цикла. Один цикл от 0 до размера массива, а второй от 0 до размера массива минус текущий индекс первого цикла. Так после очередного прохода по внутреннему цикл последние элементы массива уже будут отсортированы и их уже не надо проверять еще раз.
- 2) Реализовать функцию для итерации по отсортированному массиву, чтобы найти в нем наиболее часто встречающийся элемент (если таких элементов несколько, то взять наибольший по значению). Для этого требуется в цикле сравнивать два соседних элемента и если они отличаются, то сравнивать текущее количество элементов с количеством элементов результата. Если текущее значение больше или равно, то обновлять результат. Также при любом исходе требуется обновить счетчик количества текущего элемента. Если элементы при проверке оказались равными, то увеличить счетчик количества текущего элемента.

2. Реализация программы на С

Была написана программа на языке программирования С, которая реализует поиск наиболее часто встречающегося значения в массиве. Функции сортировки и поиска были помещены в отдельный файл, также был реализован Header-файл.

```
1  #ifndef LAB4_MAIN_H
2  #define LAB4_MAIN_H
3  |
4  ↔ void sort(unsigned *array, size_t size);
5  ↔ unsigned findElement(const unsigned *array, size_t size);
6
7  #endif
8
```

Рис.1. Листинг Header-файла.

В Header-файле определяются две функции (сортировка массива и поиск элемента по массиву), для их дальнейшего использования в тестовой программе.

```
1  #include <stddef.h>
2  #include <stdio.h>
3  #include "main.h"
4
5  //Создаем массив
6  static unsigned array[] = {2, 5, 2, 3, 1, 1};
7
8  //Найдем размерность массива
9  static const size_t array_length = sizeof(array) / sizeof(array[0]);
10
11
12  int main() {
13
14      // Выведем изначальный массив на экран
15      printf(_Format: "\nOriginal array\n");
16      for (size_t i = 0; i < array_length; i++) {
17          printf(_Format: "%u ", array[i]);
18      }
19
20      //Сделаем сортировку
21      sort(array, array_length);
22
23      //Выведем отсортированный массив
24      printf(_Format: "\n\nSorted array\n");
25      for (size_t i = 0; i < array_length; i++) {
26          printf(_Format: "%u ", array[i]);
27      }
28
29      //Найдем наиболее часто встречающееся значение
30      printf(_Format: "\n\nMost common meaning\n");
31      unsigned element = findElement(array, array_length);
32      printf(_Format: "%u ", element);
33
34  }
```

Рис.2. Листинг файла тестовой программы main.c

К проекту были подключены две стандартные библиотеки. Библиотека “stddef.h” используется для определения типа size_t (беззнаковый тип данных, размер выбирается таким образом, чтобы в него можно было записать максимальный размер теоретически возможного массива любого типа). Вторая библиотека “stdio.h” требуется для вывода в консоль.

```

1  #include <stddef.h>
2  #include "main.h"
3
4  void sort(unsigned *array, size_t size) {
5      for (size_t i = 0; i < size - 1; i++) {
6          for (size_t j = 0; j < size - 1 - i; j++) {
7              if (array[j] > array[j + 1]) {
8                  unsigned save = array[j];
9                  array[j] = array[j + 1];
10                 array[j + 1] = save;
11             }
12         }
13     }
14 }
15
16
17 unsigned findElement(const unsigned *array, size_t size) {
18     unsigned result;
19     size_t resSize = 0;
20     size_t currentSize = 1;
21
22     for (size_t i = 0; i < size - 1; i++) {
23         if (array[i] != array[i + 1]) {
24             if (currentSize >= resSize) {
25                 resSize = currentSize;
26                 result = array[i];
27             }
28             currentSize = 1;
29         } else {
30             currentSize++;
31         }
32     }
33
34     //Проверка краевого случая
35     if (currentSize >= resSize) {
36         result = array[size - 1];
37     }
38     return result;
39 }

```

Рис.3. Листинг файла с search.

```

Original array
2 5 2 3 1 1

Sorted array
1 1 2 2 3 5

Most common meaning
2

```

Рис.4. Результаты работы программы.

Исходя из рис.4, видно, что программа работает корректно и находит наибольшее наиболее часто встречающееся значение

3. Сборка программы “по шагам”

Для выполнения отдельных шагов требуется запускать драйвер компилятора и контролировать его действия при помощи флага “-v”.

3.1. Препроцессирование

Препроцессирование (в языке Си/Си++). Механизм, просматривающий входной ".c/.cpp" файл, исполняющий в нём директивы препроцессора, включающий в него содержимое других файлов, указанных в директивах #include и прочее. В результате получается файл, который не содержит директив препроцессора, все используемые макросы раскрыты, вместо директив #include подставлено содержимое соответствующих файлов. Файл с результатом препроцессирования обычно имеет суффикс ".i". Результат препроцессирования называется единицей трансляции

Первым шагом при сборке программы является препроцессирование файлов при помощи пакета разработки “SiFive GNU Embedded Toolchain” для RISC-V. Для этого потребуется выполнить следующие 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 search.c -o search.i -v  
-E >log_search_pre.txt 2>&1
```

Программа **riscv64-unknown-elf-gcc** является драйвером компилятора **gcc** (compiler driver), в данном случае она запускается со следующими параметрами командной строки:

- **--save-temps** – сохранять промежуточные (intermediate, temporary) файлы, создаваемые в процессе сборки;
- **-march=rv32i -mabi=ilp32** – целевым является процессор с базовой архитектурой системы команд RV32I;
- **-O1** – выполнять простые оптимизации генерируемого кода (мы используем эту опцию в примерах, потому что обычно генерируемый код получается более простым);
- **-v** – печатать (в стандартный поток ошибок) выполняемые драйвером команды, а также дополнительную информацию.

- **>log** - вместо печати в консоли вывод программы направляется в файл с именем "log" (если файл не существует, он создается; если файл существует, его содержимое будет утеряно);
- **2>&1** – поток вывода ошибок (2 – стандартный «номер» этого потока) «связывается» с поток вывода («номер» 1), т.е. сообщения об ошибках и информация, которая выводится флагом "-v, также выводятся в файл "log";
- **-E** – обработка файлов будет выполняться только процессором.

Теперь можно посмотреть на результат препроцессирования в двух созданных файлах. Результат имеет достаточно много строк, которые при написании явно не указывались. Эти строки связаны с файлами стандартной библиотеки языка С, которые мы указывали в нашей программе.

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

# 4 "main.h"
void sort(unsigned *array, size_t size);
unsigned findElement(const unsigned *array, size_t size);
# 4 "main.c" 2

static unsigned array[] = {2, 5, 2, 3, 1, 1};

static const size_t array_length = sizeof(array) / sizeof(array[0]);

int main() {

    printf("\nOriginal array\n");
    for (size_t i = 0; i < array_length; i++) {
        printf("%u ", array[i]);
    }

    sort(array, array_length);

    printf("\n\nSorted array\n");
    for (size_t i = 0; i < array_length; i++) {
        printf("%u ", array[i]);
    }

    printf("\n\nMost common meaning\n");
    unsigned element = findElement(array, array_length);
    printf("%u ", element);
}
```

Рис.5. Содержание файла main.i

```

# 1 "search.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "search.c"
void sort(unsigned *array, size_t size);
unsigned findElement(const unsigned *array, size_t size);
# 3 "search.c" 2

void sort(unsigned *array, size_t size) {
    for (size_t i = 0; i < size - 1; i++) {
        for (size_t j = 0; j < size - 1 - i; j++) {
            if (array[j] > array[j + 1]) {
                unsigned save = array[j];
                array[j] = array[j + 1];
                array[j + 1] = save;
            }
        }
    }
}

unsigned findElement(const unsigned *array, size_t size) {
    unsigned result;
    size_t resSize = 0;
    size_t currentSize = 1;

    for (size_t i = 0; i < size - 1; i++) {
        if (array[i] != array[i + 1]) {
            if (currentSize >= resSize) {
                resSize = currentSize;
                result = array[i];
            }
            currentSize = 1;
        } else {
            currentSize++;
        }
    }

    if (currentSize >= resSize) {
        result = array[size - 1];
    }
}

```

Рис.6. Содержание файла search.i

Символом “#” обозначаются директивы, которые используются они для передачи информации об исходном тексте из препроцессора в компилятор.

3.2. Компиляция

Теперь требуется выполнить компиляцию, для этого потребуется выполнить 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  
search.i -o search.s >log_s_search.txt 2>&1
```

Ниже приведены результаты компиляции (содержание созданных файлов).

```
1 | .file "main.c"
2 | .option nopic
3 | .attribute arch, "rv64i2p0_a2p0_c2p0"
4 | .attribute unaligned_access, 0
5 | .attribute stack_align, 16
6 | .text
7 | .section .rodata.str1.8,"aMS",@progbits,1
8 | .align 3
9 | .LC0:
10 | .string "\n0original array"
11 | .align 3
12 | .LC1:
13 | .string "%u "
14 | .align 3
15 | .LC2:
16 | .string "\n\nSorted array"
17 | .align 3
18 | .LC3:
19 | .string "\n\nMost common meaning"
20 | .text
21 | .align 1
22 | .globl main
23 | .type main, @function
24 | main:
25 |     addi    sp,sp,-48
26 |     sd     ra,40(sp)
27 |     sd     s0,32(sp)
28 |     sd     s1,24(sp)
29 |     sd     s2,16(sp)
30 |     sd     s3,8(sp)
31 |     lui    a0,%hi(.LC0)
32 |     addi    a0,a0,%lo(.LC0)
33 |     call    puts
34 |     lui    s0,%hi(.LANCHOR0)
35 |     addi    s1,s0,%lo(.LANCHOR0)
36 |     addi    s2,s1,24
37 |     addi    s0,s0,%lo(.LANCHOR0)
38 |     lui    s3,%hi(.LC1)
39 | .L2:
40 |     lw     a1,0(s0)
```

Рис.7. Содержание файла main.c (часть 1)

```

41      addi    a0,s3,%lo(.LC1)
42      call     printf
43      addi    s0,s0,4
44      bne     s0,s2,.L2
45      li      a1,6
46      lui     a0,%hi(.LANCHOR0)
47      addi    a0,a0,%lo(.LANCHOR0)
48      call     sort
49      lui     a0,%hi(.LC2)
50      addi    a0,a0,%lo(.LC2)
51      call     puts
52      lui     s0,%hi(.LC1)
53  .L3:
54      lw      a1,0(s1)
55      addi    a0,s0,%lo(.LC1)
56      call     printf
57      addi    s1,s1,4
58      bne     s1,s2,.L3
59      lui     a0,%hi(.LC3)
60      addi    a0,a0,%lo(.LC3)
61      call     puts
62      li      a1,6
63      lui     a0,%hi(.LANCHOR0)
64      addi    a0,a0,%lo(.LANCHOR0)
65      call     findElement
66      sext.w   a1,a0
67      lui     a0,%hi(.LC1)
68      addi    a0,a0,%lo(.LC1)
69      call     printf
70      li      a0,0
71      ld      ra,40(sp)
72      ld      s0,32(sp)
73      ld      s1,24(sp)
74      ld      s2,16(sp)
75      ld      s3,8(sp)
76      addi    sp,sp,48
77      jr      ra
78      .size   main,.-main
79      .data
80      .align  3
81      .set    .LANCHOR0,. + 0
82      .type   array,@object
83      .size   array, 24
84  array:
85      .word   2
86      .word   5
87      .word   2
88      .word   3
89      .word   1
90      .word   1
91      .ident  "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"

```

Рис.8. Содержание файла main.c (часть 2)

```

1      .file    "search.c"
2      .option  nopic
3      .attribute arch, "rv64i2p0_a2p0_c2p0"
4      .attribute unaligned_access, 0
5      .attribute stack_align, 16
6      .text
7      .align   1
8      .globl   sort
9      .type    sort, @function
10     sort:
11         addi   a1,a1,-1
12         beq    a1,zero,.L1
13         slli    a2,a1,2
14         add    a2,a0,a2
15         j      .L3
16     .L4:
17         addi   a5,a5,4
18         beq    a5,a2,.L11
19     .L5:
20         lw     a4,0(a5)
21         lw     a3,4(a5)
22         bleu    a4,a3,.L4
23         sw     a3,0(a5)
24         sw     a4,4(a5)
25         j      .L4
26     .L11:
27         addi   a1,a1,-1
28         addi   a2,a2,-4
29         beq    a1,zero,.L1
30     .L3:|
31         mv     a5,a0
32         bne    a1,zero,.L5
33         addi   a1,a1,-1
34         addi   a2,a2,-4
35         j      .L3
36     .L1:
37         ret
38         .size   sort, .-sort
39         .align   1
40         .globl   findElement

```

Рис.9. Содержание файла search.c (часть 1)

```

41      .type    findElement, @function
42  findElement:
43      li      a5,1
44      beq     a1,a5,.L13
45      addi    a5,a0,4
46      slli    a6,a1,2
47      add     a6,a0,a6
48      li      a4,1
49      li      a7,0
50      j       .L16
51  .L14:
52      addi    a4,a4,1
53  .L15:
54      addi    a5,a5,4
55      beq     a5,a6,.L20
56  .L16:
57      lw      a3,-4(a5)
58      lw      a2,0(a5)
59      beq     a2,a3,.L14
60      bgtu    a7,a4,.L18
61      mv      a7,a4
62      mv      t1,a3
63      li      a4,1
64      j       .L15
65  .L18:
66      li      a4,1
67      j       .L15
68  .L20:
69      bleu    a7,a4,.L13
70  .L17:
71      mv      a0,t1
72      ret
73  .L13:
74      slli    a1,a1,2
75      add     a1,a0,a1
76      lw      t1,-4(a1)
77      j       .L17
78      .size   findElement, .-findElement
79      .ident   "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"

```

Рис.10. Содержание файла search.c (часть 2)

Самые интересные моменты выделены красным цветом. Например, можно заметить, как в файле search.s реализуется цикл for через инструкции RISC-V. Видно, что тестовая программа вызывает search через псевдофункцию call. Также снизу видна метка на наш массив array.

3.3. Объектный файл

Объектный файл – это файл с промежуточным представлением отдельного модуля программы, полученный в результате обработки исходного кода компилятором. Объектный файл содержит в себе особым образом подготовленный код (часто называемый двоичным или бинарным), который может быть объединён с другими объектными файлами при помощи редактора связей (компоновщика) для получения готового исполнимого модуля либо библиотеки

Выполним ассемблирование для получения объектных файлов программы, для этого потребуется выполнить 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 search.s -o search.o  
>log_o.txt 2>&1
```

Теперь у нас появились файлы main.o и search.o. Эти файлы являются бинарным, поэтому для их просмотра требуется ввести команду из пакета разработки:

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

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

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .text          000000c4  0000000000000000  0000000000000000  00000040  2**1
    CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data          00000018  0000000000000000  0000000000000000  00000108  2**3
    CONTENTS, ALLOC, LOAD, DATA
  2 .bss           00000000  0000000000000000  0000000000000000  00000120  2**0
    ALLOC
  3 .rodata.str1.8 0000003e  0000000000000000  0000000000000000  00000120  2**3
    CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .comment        00000031  0000000000000000  0000000000000000  0000015e  2**0
    CONTENTS, READONLY
  5 .riscv.attributes 00000026  0000000000000000  0000000000000000  0000018f  2**
*0
    CONTENTS, READONLY
```

Рис.11. Header файла main.o

Теперь введем команду для файла search.o

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

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

Sections:
Idx Name          Size      UMA              LMA              File off  Algn
 0  .text          000000c4  0000000000000000  0000000000000000  000000040  2**1
    CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
 1  .data          00000018  0000000000000000  0000000000000000  000000108  2**3
    CONTENTS, ALLOC, LOAD, DATA
 2  .bss           00000000  0000000000000000  0000000000000000  000000120  2**0
    ALLOC
 3  .rodata.str1.8 0000003e  0000000000000000  0000000000000000  000000120  2**3
    CONTENTS, ALLOC, LOAD, READONLY, DATA
 4  .comment       00000031  0000000000000000  0000000000000000  00000015e  2**0
    CONTENTS, READONLY
 5  .riscv.attributes 00000026  0000000000000000  0000000000000000  00000018f  2**0
    CONTENTS, READONLY
```

Рис.12. Header файла search.o

Вся информация размещается в секциях:

- .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

Так же в начале выводе пишут о формате файла “elf” и о том, что используется архитектур alittle-endian RISC-V. Рассмотрим некоторые секции поближе, для этого введем команду:

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

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

Disassembly of section .text:

0000000000000000 <main>:
 0: 7179          c.addi16sp    sp,-48
 2: f406          c.sdsp      ra,40(sp)
 4: f022          c.sdsp      s0,32(sp)
 6: ec26          c.sdsp      s1,24(sp)
 8: e84a          c.sdsp      s2,16(sp)
 a: e44e          c.sdsp      s3,8(sp)
 c: 000000537     lui        a0,0x0
10: 00050513     addi       a0,a0,0 # 0 <main>
14: 00000097     auipc      ra,0x0
18: 000080e7     jalr       ra,0(ra) # 14 <main+0x14>
1c: 000000437     lui        s0,0x0
20: 00040493     addi       s1,s0,0 # 0 <main>
24: 01848913     addi       s2,s1,24
28: 00040413     addi       s0,s0,0
2c: 0000009b7     lui        s3,0x0
```

Рис.13. Дизассемблированный файл main.o (часть 1)

```

00000000000000030 <.L2>:
30: 400c          c.lw      a1,0(s0)
32: 000098513     addi      a0,s3,0 # 0 <main>
36: 000000097     auipc     ra,0x0
3a: 000080e7     jalr      ra,0(ra) # 36 <.L2+0x6>
3e: 0411          c.addi    s0,4
40: ff2418e3     bne       s0,s2,30 <.L2>
44: 4599          c.li      a1,6
46: 000000537     lui       a0,0x0
4a: 00050513     addi      a0,a0,0 # 0 <main>
4e: 000000097     auipc     ra,0x0
52: 000080e7     jalr      ra,0(ra) # 4e <.L2+0x1e>
56: 000000537     lui       a0,0x0
5a: 00050513     addi      a0,a0,0 # 0 <main>
5e: 000000097     auipc     ra,0x0
62: 000080e7     jalr      ra,0(ra) # 5e <.L2+0x2e>
66: 000000437     lui       s0,0x0

0000000000000006a <.L3>:
6a: 408c          c.lw      a1,0(s1)
6c: 00040513     addi      a0,s0,0 # 0 <main>
70: 000000097     auipc     ra,0x0
74: 000080e7     jalr      ra,0(ra) # 70 <.L3+0x6>
78: 0491          c.addi    s1,4
7a: ff2498e3     bne       s1,s2,6a <.L3>
7e: 000000537     lui       a0,0x0
82: 00050513     addi      a0,a0,0 # 0 <main>
86: 000000097     auipc     ra,0x0
8a: 000080e7     jalr      ra,0(ra) # 86 <.L3+0x1c>
8e: 4599          c.li      a1,6
90: 000000537     lui       a0,0x0
94: 00050513     addi      a0,a0,0 # 0 <main>
98: 000000097     auipc     ra,0x0
9c: 000080e7     jalr      ra,0(ra) # 98 <.L3+0x2e>
a0: 0005059b     addiw     a1,a0,0
a4: 000000537     lui       a0,0x0
a8: 00050513     addi      a0,a0,0 # 0 <main>
ac: 000000097     auipc     ra,0x0
b0: 000080e7     jalr      ra,0(ra) # ac <.L3+0x42>
b4: 4501          c.li      a0,0
b6: 70a2          c.ldsp    ra,40(sp)
b8: 7402          c.ldsp    s0,32(sp)
ba: 64e2          c.ldsp    s1,24(sp)
bc: 6942          c.ldsp    s2,16(sp)
be: 69a2          c.ldsp    s3,8(sp)
c0: 6145          c.addi16sp sp,48
c2: 8082          c.jr      ra

```

Рис.14. Дизассемблированный файл main.o (часть 2)

На рис.13- рис.14. можно заметить комбинации инструкций auipc+jalr, которые на самом деле являются одной из псевдофункций call

Рассмотрим содержание секции comment, для этого введем команду:

```
riscv64-unknown-elf-objdump -s -j .comment main.o
```

```

main.o:          file format elf64-littleriscv

Contents of section .comment:
0000 00474343 3a202853 69466976 65204743      .GCC: (SiFive GC
0010 432d4d65 74616c20 31302e32 2e302d32      C-Metal 10.2.0-2
0020 3032302e 31322e38 29203130 2e322e30      020.12.8) 10.2.0
0030 00

```

Рис.15. Содержание секции comment.

Рассмотрим таблицу символов, для этого введем команду:

```
riscv64-unknown-elf-objdump -t search.o main.o
```

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

SYMBOL TABLE:
0000000000000000 1      df *ABS* 0000000000000000 search.c
0000000000000000 1      d  .text 0000000000000000 .text
0000000000000000 1      d  .data 0000000000000000 .data
0000000000000000 1      d  .bss 0000000000000000 .bss
0000000000000030 1      .text 0000000000000000 .L1
0000000000000026 1      .text 0000000000000000 .L3
0000000000000050 1      .text 0000000000000000 .L11
000000000000000c 1      .text 0000000000000000 .L4
0000000000000012 1      .text 0000000000000000 .L5
0000000000000072 1      .text 0000000000000000 .L13
0000000000000050 1      .text 0000000000000000 .L16
000000000000006a 1      .text 0000000000000000 .L20
0000000000000048 1      .text 0000000000000000 .L14
0000000000000066 1      .text 0000000000000000 .L18
000000000000004a 1      .text 0000000000000000 .L15
000000000000006e 1      .text 0000000000000000 .L17
0000000000000000 1      d  .comment 0000000000000000 .comment
0000000000000000 1      d  .riscv.attributes 0000000000000000 .riscv.attributes
0000000000000000 g      F  .text 0000000000000032 sort
0000000000000032 g      F  .text 000000000000004a findElement

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.str1.8 0000000000000000 .rodata.str1.8
0000000000000000 1      .data 0000000000000000 .LANCHOR0
0000000000000000 1      0  .data 0000000000000018 array
0000000000000000 1      .rodata.str1.8 0000000000000000 .LC0
0000000000000010 1      .rodata.str1.8 0000000000000000 .LC1
0000000000000018 1      .rodata.str1.8 0000000000000000 .LC2
0000000000000028 1      .rodata.str1.8 0000000000000000 .LC3
0000000000000030 1      .text 0000000000000000 .L2
000000000000006a 1      .text 0000000000000000 .L3
0000000000000000 1      d  .comment 0000000000000000 .comment
0000000000000000 1      d  .riscv.attributes 0000000000000000 .riscv.attributes
0000000000000000 g      F  .text 00000000000000c4 main
0000000000000000      *UND* 0000000000000000 puts
0000000000000000      *UND* 0000000000000000 printf
0000000000000000      *UND* 0000000000000000 sort
0000000000000000      *UND* 0000000000000000 findElement
```

Рис.16. 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. Для ее отображения введем команду:

```
riscv64-unknown-elf-objdump -r search.o main.o
```



```

search.o:      file format elf64-littleriscv

RELOCATION RECORDS FOR [.text]:
OFFSET      TYPE      VALUE
0000000000000002 R_RISCV_RVC_BRANCH .L1
000000000000000a R_RISCV_RVC_JUMP   .L3
000000000000000e R_RISCV_BRANCH     .L11
0000000000000016 R_RISCV_BRANCH     .L4
000000000000001e R_RISCV_RVC_JUMP   .L4
0000000000000024 R_RISCV_RVC_BRANCH .L1
0000000000000028 R_RISCV_RVC_BRANCH .L5
000000000000002e R_RISCV_RVC_JUMP   .L3
0000000000000034 R_RISCV_BRANCH     .L13
0000000000000046 R_RISCV_RVC_JUMP   .L16
000000000000004c R_RISCV_BRANCH     .L20
0000000000000056 R_RISCV_BRANCH     .L14
000000000000005a R_RISCV_BRANCH     .L18
0000000000000064 R_RISCV_RVC_JUMP   .L15
0000000000000068 R_RISCV_RVC_JUMP   .L15
000000000000006a R_RISCV_BRANCH     .L13
000000000000007a R_RISCV_RVC_JUMP   .L17

main.o:      file format elf64-littleriscv

RELOCATION RECORDS FOR [.text]:
OFFSET      TYPE      VALUE
000000000000000c R_RISCV_HI20       .LC0
000000000000000c R_RISCV_RELAX      *ABS*
0000000000000010 R_RISCV_LO12_I     .LC0
0000000000000010 R_RISCV_RELAX      *ABS*
0000000000000014 R_RISCV_CALL       puts
0000000000000014 R_RISCV_RELAX      *ABS*
000000000000001c R_RISCV_HI20       .LANCHORO
000000000000001c R_RISCV_RELAX      *ABS*
0000000000000020 R_RISCV_LO12_I     .LANCHORO
0000000000000020 R_RISCV_RELAX      *ABS*
0000000000000028 R_RISCV_LO12_I     .LANCHORO
0000000000000028 R_RISCV_RELAX      *ABS*
000000000000002c R_RISCV_HI20       .LC1
000000000000002c R_RISCV_RELAX      *ABS*
0000000000000032 R_RISCV_LO12_I     .LC1
0000000000000032 R_RISCV_RELAX      *ABS*
0000000000000036 R_RISCV_CALL       printf
0000000000000036 R_RISCV_RELAX      *ABS*
0000000000000046 R_RISCV_HI20       .LANCHORO
0000000000000046 R_RISCV_RELAX      *ABS*
000000000000004a R_RISCV_LO12_I     .LANCHORO
000000000000004a R_RISCV_RELAX      *ABS*
000000000000004e R_RISCV_CALL       sort
000000000000004e R_RISCV_RELAX      *ABS*
0000000000000056 R_RISCV_HI20       .LC2
0000000000000056 R_RISCV_RELAX      *ABS*
000000000000005a R_RISCV_LO12_I     .LC2
000000000000005a R_RISCV_RELAX      *ABS*
000000000000005e R_RISCV_CALL       puts
000000000000005e R_RISCV_RELAX      *ABS*
0000000000000066 R_RISCV_HI20       .LC1
0000000000000066 R_RISCV_RELAX      *ABS*
000000000000006c R_RISCV_LO12_I     .LC1
000000000000006c R_RISCV_RELAX      *ABS*
0000000000000070 R_RISCV_CALL       printf
0000000000000070 R_RISCV_RELAX      *ABS*
000000000000007e R_RISCV_HI20       .LC3
000000000000007e R_RISCV_RELAX      *ABS*
0000000000000082 R_RISCV_LO12_I     .LC3
0000000000000082 R_RISCV_RELAX      *ABS*
0000000000000086 R_RISCV_CALL       puts
0000000000000086 R_RISCV_RELAX      *ABS*
0000000000000090 R_RISCV_HI20       .LANCHORO
0000000000000090 R_RISCV_RELAX      *ABS*
0000000000000094 R_RISCV_LO12_I     .LANCHORO
0000000000000094 R_RISCV_RELAX      *ABS*
0000000000000098 R_RISCV_CALL       findElement
0000000000000098 R_RISCV_RELAX      *ABS*
00000000000000a4 R_RISCV_HI20       .LC1
00000000000000a4 R_RISCV_RELAX      *ABS*
00000000000000a8 R_RISCV_LO12_I     .LC1
00000000000000a8 R_RISCV_RELAX      *ABS*
00000000000000ac R_RISCV_CALL       printf
00000000000000ac R_RISCV_RELAX      *ABS*
0000000000000040 R_RISCV_BRANCH     .L2
000000000000007a R_RISCV_BRANCH     .L3

```

Рис.17. Таблица перемещений.

В таблице перемещений для main.o наблюдаем вызов методов sort и findElement. 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.

3.4. Компоновка

Выполним компоновку при помощи команды:

```
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -v main.o search.o -o main.out  
>log_out.txt 2>&1
```

В результате выполнения этой команды был создан файл main.out – исполняемый бинарный файл. Для рассмотрения его секций кода введем команду:

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

```
67 0000000000010156 <main>:  
68 10156: 7179          c.addi16sp sp,-48  
69 10158: f406          c.sdsp ra,40(sp)  
70 1015a: f022          c.sdsp s0,32(sp)  
71 1015c: ec26          c.sdsp s1,24(sp)  
72 1015e: e84a          c.sdsp s2,16(sp)  
73 10160: e44e          c.sdsp s3,8(sp)  
74 10162: 6575          c.lui a0,0x1d  
75 10164: c4050513      addi a0,a0,-960 # 1cc40 <__clzdi2+0x3a>  
76 10168: 302000ef      jal ra,1046a <puts>  
77 1016c: 0001f437      lui s0,0x1f  
78 10170: bf040493      addi s1,s0,-1040 # 1ebf0 <array>  
79 10174: 01848913      addi s2,s1,24  
80 10178: bf040413      addi s0,s0,-1040  
81 1017c: 69f5          c.lui s3,0x1d  
82 1017e: 400c          c.lw a1,0(s0)  
83 10180: c5098513      addi a0,s3,-944 # 1cc50 <__clzdi2+0x4a>  
84 10184: 23a000ef      jal ra,103be <printf>  
85 10188: 0411          c.addi s0,4  
86 1018a: ff241ae3      bne s0,s2,1017e <main+0x28>  
87 1018e: 4599          c.li a1,6  
88 10190: 0001f537      lui a0,0x1f  
89 10194: bf050513      addi a0,a0,-1040 # 1ebf0 <array>  
90 10198: 056000ef      jal ra,101ee <sort>  
91 1019c: 6575          c.lui a0,0x1d  
92 1019e: c5850513      addi a0,a0,-936 # 1cc58 <__clzdi2+0x52>  
93 101a2: 2c8000ef      jal ra,1046a <puts>  
94 101a6: 6475          c.lui s0,0x1d  
95 101a8: 408c          c.lw a1,0(s1)  
96 101aa: c5040513      addi a0,s0,-944 # 1cc50 <__clzdi2+0x4a>  
97 101ae: 210000ef      jal ra,103be <printf>  
98 101b2: 0491          c.addi s1,4  
99 101b4: ff249ae3      bne s1,s2,101a8 <main+0x52>  
100 101b8: 6575          c.lui a0,0x1d  
101 101ba: c6850513      addi a0,a0,-920 # 1cc68 <__clzdi2+0x62>  
102 101be: 2ac000ef      jal ra,1046a <puts>  
103 101c2: 4599          c.li a1,6  
104 101c4: 0001f537      lui a0,0x1f  
105 101c8: bf050513      addi a0,a0,-1040 # 1ebf0 <array>  
106 101cc: 054000ef      jal ra,10220 <findElement>  
107 101d0: 0005059b      addiw a1,a0,0  
108 101d4: 6575          c.lui a0,0x1d  
109 101d6: c5050513      addi a0,a0,-944 # 1cc50 <__clzdi2+0x4a>  
110 101da: 1e4000ef      jal ra,103be <printf>  
111 101de: 4501          c.li a0,0  
112 101e0: 70a2          c.ldsp ra,40(sp)  
113 101e2: 7402          c.ldsp s0,32(sp)  
114 101e4: 64e2          c.ldsp s1,24(sp)  
115 101e6: 6942          c.ldsp s2,16(sp)  
116 101e8: 69a2          c.ldsp s3,8(sp)  
117 101ea: 6145          c.addi16sp sp,48  
118 101ec: 8082          c.jr ra
```

Рис.18. Исполняемый файл (часть 1).

```

120 00000000000101ee <sort>:
121 101ee: 15fd c.addi a1,-1
122 101f0: c59d c.beqz a1,1021e <sort+0x30>
123 101f2: 00259613 slli a2,a1,0x2
124 101f6: 962a c.add a2,a0
125 101f8: a831 c.j 10214 <sort+0x26>
126 101fa: 0791 c.addi a5,4
127 101fc: 00c78963 beq a5,a2,1020e <sort+0x20>
128 10200: 4398 c.lw a4,0(a5)
129 10202: 43d4 c.lw a3,4(a5)
130 10204: fee6f3e3 bgeu a3,a4,101fa <sort+0xc>
131 10208: c394 c.sw a3,0(a5)
132 1020a: c3d8 c.sw a4,4(a5)
133 1020c: b7fd c.j 101fa <sort+0xc>
134 1020e: 15fd c.addi a1,-1
135 10210: 1671 c.addi a2,-4
136 10212: c591 c.beqz a1,1021e <sort+0x30>
137 10214: 87aa c.mv a5,a0
138 10216: f5ed c.bnez a1,10200 <sort+0x12>
139 10218: 15fd c.addi a1,-1
140 1021a: 1671 c.addi a2,-4
141 1021c: bfe5 c.j 10214 <sort+0x26>
142 1021e: 8082 c.jr ra
143
144 0000000000010220 <findElement>:
145 10220: 4785 c.li a5,1
146 10222: 02f58f63 beq a1,a5,10260 <findElement+0x40>
147 10226: 00450793 addi a5,a0,4
148 1022a: 00259813 slli a6,a1,0x2
149 1022e: 982a c.add a6,a0
150 10230: 4705 c.li a4,1
151 10232: 4881 c.li a7,0
152 10234: a029 c.j 1023e <findElement+0x1e>
153 10236: 0705 c.addi a4,1
154 10238: 0791 c.addi a5,4
155 1023a: 01078f63 beq a5,a6,10258 <findElement+0x38>
156 1023e: ffc7a683 lw a3,-4(a5)
157 10242: 4390 c.lw a2,0(a5)
158 10244: fed609e3 beq a2,a3,10236 <findElement+0x16>
159 10248: 01176663 bltu a4,a7,10254 <findElement+0x34>
160 1024c: 88ba c.mv a7,a4
161 1024e: 8336 c.mv t1,a3
162 10250: 4705 c.li a4,1
163 10252: b7dd c.j 10238 <findElement+0x18>
164 10254: 4705 c.li a4,1
165 10256: b7cd c.j 10238 <findElement+0x18>
166 10258: 01177463 bgeu a4,a7,10260 <findElement+0x40>
167 1025c: 851a c.mv a0,t1
168 1025e: 8082 c.jr ra
169 10260: 058a c.slli a1,0x2
170 10262: 95aa c.add a1,a0
171 10264: ffc5a303 lw t1,-4(a1)
172 10268: bfd5 c.j 1025c <findElement+0x3c>

```

Рис.19. Исполняемый файл (часть 2).

Адресация для вызовов функций изменилась на абсолютную.

4. 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.

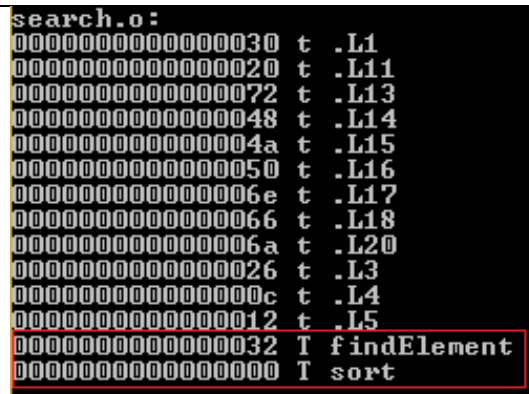
Для создания статической библиотеки требуется выделить функцию `search` в отдельную статическую библиотеку. Для этого надо получить объектный файл `search.o` и собрать библиотеку.

Выполним следующие команды:

```
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -O1 -c search.c -o search.o  
riscv64-unknown-elf-ar -rsc libSearch.a search.o
```

Теперь можно рассмотреть список символов нашей библиотеки, для этого введем команду:

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



```
search.o:  
000000000000000030 t .L1  
000000000000000020 t .L11  
000000000000000072 t .L13  
000000000000000048 t .L14  
00000000000000004a t .L15  
000000000000000050 t .L16  
00000000000000006e t .L17  
000000000000000066 t .L18  
00000000000000006a t .L20  
000000000000000026 t .L3  
00000000000000000c t .L4  
000000000000000012 t .L5  
000000000000000032 T findElement  
000000000000000000 T sort
```

Рис.20. Список символов `libSearch.a`

В выводе утилиты “nm” кодом “T” обозначаются символы, определенные в соответствующем объектном файле. Символ функций `findElement` и `sort` является основным символом, определяемым в этом объектном файле. Используя собранную библиотеку, произведём исполняемый файл тестовой программы, для этого введем команду:

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

Теперь проверим таблицу символов на наличие функции `search` при помощи команды:

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

34	00000000000000000000	1	df	*ABS*	00000000000000000000	main.c
35	00000000000001ebf0	1	O	.data	00000000000000000018	array
36	00000000000000000000	1	df	*ABS*	00000000000000000000	search.c
37	00000000000000000000	1	df	*ABS*	00000000000000000000	exit.c
38	00000000000000000000	1	df	*ABS*	00000000000000000000	impure.c
39	00000000000001ec08	1	O	.data	00000000000000000748	impure_data

Рис.21. Таблица символов main.out

4.1. Создание make-файлов

Для того, чтобы автоматизировать процесс сборки библиотеки и приложения напишем make-файлы. Используя пример с сайта курса, были написаны следующие файлы:

```

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: search.o
8     $(AR) -rsc libSearch.a search.o
9     del -f *.o
10 search.o: search.c
11     $(CC) $(CFLAGS) -c search.c -o search.o

```

Рис.21. Содержимое файла make_lib

```

1 TARGET=main
2 CC=riscv64-unknown-elf-gcc
3 CFLAGS=-march=rv64iac -mabi=lp64
4
5 all:
6     make -f make_lib
7     $(CC) $(CFLAGS) main.c libSearch.a -o $(TARGET)
8     del -f *.o *.a

```

Рис.22. Содержимое файла make_app

Для создания библиотеки необходимо выполнить `make_lib`, а для приложения `make_app`.

```
C:\vichmat_labs\lab4\lib>make -f make_lib
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -c search.c -o search.o
riscv64-unknown-elf-ar -rsc libSearch.a search.o
del -f *.o

C:\vichmat_labs\lab4\lib>dir
Том в устройстве C не имеет метки.
Серийный номер тома: 7EBD-7F2A

Содержимое папки C:\vichmat_labs\lab4\lib

26.04.2021 01:38 <DIR> .
26.04.2021 01:38 <DIR> ..
26.04.2021 01:38 2 246 libSearch.a
25.04.2021 20:26 950 main.c
25.04.2021 20:11 156 main.h
26.04.2021 01:26 172 make_app
26.04.2021 01:26 228 make_lib
25.04.2021 20:31 998 search.c
6 файлов 4 750 байт
2 папок 75 022 495 744 байт свободно
```

Рис.23. Выполнение make файлов (часть 1)

```
C:\vichmat_labs\lab4\lib>make -f make_app
make -f make_lib
make[1]: Entering directory 'C:/vichmat_labs/lab4/lib'
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -c search.c -o search.o
riscv64-unknown-elf-ar -rsc libSearch.a search.o
del -f *.o
make[1]: Leaving directory 'C:/vichmat_labs/lab4/lib'
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 main.c libSearch.a -o main
del -f *.o *.a

C:\vichmat_labs\lab4\lib>dir
Том в устройстве C не имеет метки.
Серийный номер тома: 7EBD-7F2A

Содержимое папки C:\vichmat_labs\lab4\lib

26.04.2021 01:38 <DIR> .
26.04.2021 01:38 <DIR> ..
26.04.2021 01:38 143 528 main
25.04.2021 20:26 950 main.c
25.04.2021 20:11 156 main.h
26.04.2021 01:26 172 make_app
26.04.2021 01:26 228 make_lib
25.04.2021 20:31 998 search.c
6 файлов 146 032 байт
2 папок 75 022 548 992 байт свободно
```

Рис.24. Выполнение make файлов (часть 2)

5. Вывод

В данной лабораторной работе я познакомился с языком программирования C, а именно написал программу на этом языке с заданной функциональностью (нахождение в массиве наибольшего наиболее часто встречающегося элемента). После этого была выполнена сборка “по шагам” для архитектуры команд RISC-V. Был произведен анализ выводов препроцессора, компилятора и линковщика. Также для автоматизации процесса сборки была создана собственная статически линкуемая библиотека libSearch.a. Были написаны make-файлы для её сборки (make_lib и make_app), а также сборки тестовой программы с использованием библиотеки.

Список использованных источников

<https://www.sifive.com/software>

<http://kspt.icc.spbstu.ru/media/files/2018/lowlevelprog/cle.pdf>

<http://kspt.icc.spbstu.ru/media/files/2020/lowlevelprog/lab4.pdf>