

**Отчёт по лабораторной работе № 4**

Дисциплина: Низкоуровневое программирование

Тема: раздельная компиляция

Вариант: 5

Выполнил студент гр. 3530901/90002 \_\_\_\_\_ Е. В. Бурков  
(подпись)

Принял преподаватель \_\_\_\_\_ Д. С. Степанов  
(подпись)

“ \_\_\_\_\_ ” \_\_\_\_\_ 2021 г.

## Формулировка задачи

1. На языке C разработать функцию, реализующую определенную вариантом задания функциональность. Поместить определение функции в отдельный исходный файл, оформить заголовочный файл. Разработать тестовую программу на языке C.
2. Собрать программу «по шагам». Проанализировать выход препроцессора и компилятора. Проанализировать состав и содержимое секций, таблицы символов, таблицы перемещений и отладочную информацию, содержащуюся в объектных файлах и исполняемом файле.
3. Выделить разработанную функцию в статическую библиотеку. Разработать make-файлы для сборки библиотеки и использующей ее тестовой программы. Проанализировать ход сборки библиотеки и программы, созданные файлы зависимостей.

## Вариант задания

По варианту номер 5 необходимо реализовать сортировку обменом чисел in-place. Сортировка обменом является простым алгоритмом. Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются  $N - 1$  раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде — отсюда и название алгоритма).

# 1. Написание программы на языке C

Согласно заданию, была написана программа, сортирующая массив пузырьком. Функция помещена в отдельный файл, оформлен заголовочный файл.

Листинг 1.1 Заголовочный файл bubble.h

```
#ifndef BUBBLE_H
#define BUBBLE_H

void bubble_sort(unsigned *array, size_t size);

#endif
```

В заголовочном файле отображаем функцию сортировки для её использования в тестовой программе.

Листинг 1.2 Файл тестовой программы main.c

```
#include <stddef.h>
#include <stdio.h>
#include "bubble.h"

static unsigned array[] = {
    1337, 1488, 228, 42, 2048, 0
};

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

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

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

Были импортированы стандартные библиотеки “**stddef.h**” и “**stdio.h**”. Первая требуется для определения типа `size_t`. Вторая используется для вывода на консоль.

```
#include <stdio.h>
#include "bubble.h"

void bubble_sort(unsigned *array, size_t size) {

    static unsigned f = 0;

    for (size_t i = 0; i < size - 1; i++) {
        if (f == 1) {
            return;
        }
        f = 1; // If array sorted - quit
        for (size_t j = 0; j < size - i - 1; j++) {
            if (array[j] > array[j + 1]) {
                const unsigned temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
                f = 0;
            }
        }
    }
}
```

Произведём компиляцию программы и посмотрим на результат исполнения:

```
D:\projects\lowlevelprog\lowlevelprog\lab4\app>untitled.exe
1337 1488 228 42 2048 0
0 42 228 1337 1488 2048
```

Рис. 1.1 Вызов программы

На первой строчке указан неотсортированный массив, а на второй уже отсортированный.

## 2. Сборка программы “по шагам”

### Преппроцессирование<sup>1</sup>

Используя пакет разработки “SiFive GNU Embedded Toolchain” для RISC-V выполним преппроцессирование файлов. Для этого выполним следующие команды:

```
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -O1 -E main.c -o main.i -v -E  
>log_main_pre.txt 2>&1  
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -O1 -E bubble.c -o bubble.i -v -E  
>log_bubble_pre.txt 2>&1
```

Разберём параметры запуска.

|                          |  |
|--------------------------|--|
| -march=rv32i -mabi=ilp32 | целевым является процессор с базовой архитектурой системы команд RV32I |
| -O1                      | выполнять простые оптимизации генерируемого кода                       |
| >                        | Выводы печати в файлы  |
| -o                       | Output file  |
| -E                       | Выполнять обработку файлов только преппроцессором                      |

Посмотрим на результаты преппроцессирования. Результат имеет достаточно много строк, которые при написании явно не указывались. Эти строки связаны с файлами стандартной библиотеки языка C, которые мы указывали в нашей программе.

---

<sup>1</sup> Преппроцессирование (в языке Си/Си++). Механизм, просматривающий входной ".c/.cpp" файл, исполняющий в нём директивы преппроцессора, включающий в него содержимое других файлов, указанных в директивах #include и прочее. В результате получается файл, который не содержит директив преппроцессора, все используемые макросы раскрыты, вместо директив #include подставлено содержимое соответствующих файлов. Файл с результатом преппроцессирования обычно имеет суффикс ".i". Результат преппроцессирования называется единицей трансляции.

```

# 1 "main.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "main.c"
.....
# 5 "bubble.h"
void bubble_sort(unsigned *array, size_t size);
# 4 "main.c" 2

static unsigned array[] = {
    1337, 1488, 228, 42, 2048, 0
};

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

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

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

```

```

# 5 "bubble.h"
void bubble_sort(unsigned *array, size_t size);
# 3 "bubble.c" 2

void bubble_sort(unsigned *array, size_t size) {

    static unsigned f = 0;

    for (size_t i = 0; i < size - 1; i++) {
        if (f == 1) {
            return;
        }
        f = 1;
    }
}

```

```
    for (size_t j = 0; j < size - i - 1; j++) {  
        if (array[j] > array[j + 1]) {  
            const unsigned temp = array[j];  
            array[j] = array[j + 1];  
            array[j + 1] = temp;  
            f = 0;  
        }  
    }  
}
```

Видно, что в данных файлах содержится информация из заголовочного файла.

## Компиляция

Для компиляции препроцессированных файлов используем следующие команды:

```
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -O1 -v -S -fpreprocessed main.i -  
o main.s >log_s_main.txt 2>&1  
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -O1 -v -S -fpreprocessed  
bubble.i -o bubble.s >log_s_bubble.txt 2>&1
```

Ниже приведены файлы – результаты компиляции:

Листинг 2.3 Файл main.s

```
.file "main.c"  
.option nopic  
.text  
.align 2  
.globl main  
.type main, @function  
main:  
    addi    sp,sp,-32  
    sw      ra,28(sp)  
    sw      s0,24(sp)  
    sw      s1,20(sp)  
    sw      s2,16(sp)  
    sw      s3,12(sp)  
    lui     s0,%hi(.LANCHOR0)  
    addi    s1,s0,%lo(.LANCHOR0)  
    addi    s2,s1,24  
    addi    s0,s0,%lo(.LANCHOR0)  
    lui     s3,%hi(.LC0)  
.L2:  
    lw      a1,0(s0)  
    addi    a0,s3,%lo(.LC0)  
    call    printf  
    addi    s0,s0,4  
    bne     s0,s2,.L2  
    li      a0,10  
    call    putchar  
    li      a1,6  
    lui     a0,%hi(.LANCHOR0)  
    addi    a0,a0,%lo(.LANCHOR0)  
    call    bubble_sort  
    lui     s0,%hi(.LC0)  
.L3:
```



```

lw    a1,0(s1)
addi  a0,s0,%lo(.LC0)
call  printf
addi  s1,s1,4
bne   s1,s2,.L3
li     a0,0
lw     ra,28(sp)
lw     s0,24(sp)
lw     s1,20(sp)
lw     s2,16(sp)
lw     s3,12(sp)
addi  sp,sp,32
jr     ra
.size  main,.-main
.data
.align 2
.set   .LANCHOR0,. + 0
.type  array, @object
.size  array, 24
array:
.word 1337
.word 1488
.word 228
.word 42
.word 2048
.word 0
.section .rodata.str1.4,"aMS",@progbits,1
.align 2
.LC0:
.string "%u "
.ident "GCC: (SiFive GCC 8.2.0-2019.05.3) 8.2.0"

```

#### Листинг 2.4 Файл bubble.s

```

.file "bubble.c"
.option nopic
.text
.align 2
.globl bubble_sort
.type bubble_sort, @function
bubble_sort:
addi  t3,a1,-1
beqz  t3,.L1
lui   a5,%hi(f.2552)
lw     a4,%lo(f.2552)(a5)
li     a5,1

```

```

    beq    a4,a5,.L1
    slli   a2,a1,2
    add    a2,a0,a2
    li     a7,0
    li     t1,1
    li     a6,0
    j      .L3
.L11:
    lui    a5,%hi(f.2552)
    li     a4,1
    sw     a4,%lo(f.2552)(a5)
    ret
.L5:
    addi   a5,a5,4
.L4:
    beq    a5,a2,.L9
    lw     a4,-4(a5)
    lw     a3,0(a5)
    bleu   a4,a3,.L5
    sw     a3,-4(a5)
    sw     a4,0(a5)
    mv     a1,a6
    j      .L5
.L9:
    addi   a7,a7,1
    beq    a7,t3,.L10
    addi   a2,a2,-4
    beq    a1,t1,.L11
.L3:
    addi   a5,a0,4
    mv     a1,t1
    j      .L4
.L10:
    lui    a5,%hi(f.2552)
    sw     a1,%lo(f.2552)(a5)
.L1:
    ret
.size    bubble_sort,.-bubble_sort
.section .sbss,"aw",@nobits
.align   2
.type    f.2552, @object
.size    f.2552, 4
f.2552:
    .zero 4
    .ident "GCC: (SiFive GCC 8.2.0-2019.05.3) 8.2.0"

```

Наиболее интересные куски кода выделены красным цветом.

Можно заметить, как реализуется цикл for через инструкции RISC-V. Заметим, что тестовая программа действительно вызывает bubble\_sort через псевдоинструкцию call. Так же снизу имеем метку на наш массив array. В файле bubble.s видно, как программа меняет элементы местами.

## Объектный файл<sup>2</sup>

Выполним ассемблирование для получения объектных файлов программы.

```
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -v -c main.s -o main.o >log_o.txt 2>&1
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -v -c bubble.s -o bubble.o >log_o.txt 2>&1
```

На выходе получаем файлы “bubble.o” и “main.o”. Данные файлы являются бинарными, поэтому используем программу из пакета разработки для их прочтения.

### Листинг 2.5 Хедер файла main.o

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

main.o: file format elf32-littleriscv

Sections:

| Idx | Name              | Size     | VMA      | LMA      | File off                                     | Algn |
|-----|-------------------|----------|----------|----------|--|------|
| 0   | .text             | 000000a0 | 00000000 | 00000000 | 00000034                                     | 2**2 |
|     |                   |          |          |          | CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE |      |
| 1   | .data             | 00000018 | 00000000 | 00000000 | 000000d4                                     | 2**2 |
|     |                   |          |          |          | CONTENTS, ALLOC, LOAD, DATA                  |      |
| 2   | .bss              | 00000000 | 00000000 | 00000000 | 000000ec                                     | 2**0 |
|     |                   |          |          |          | ALLOC  |      |
| 3   | .rodata.str1.4    | 00000004 | 00000000 | 00000000 | 000000ec                                     | 2**2 |
|     |                   |          |          |          | CONTENTS, ALLOC, LOAD, READONLY, DATA        |      |
| 4   | .comment          | 00000029 | 00000000 | 00000000 | 000000f0                                     | 2**0 |
|     |                   |          |          |          | CONTENTS, READONLY                           |      |
| 5   | .riscv.attributes | 0000001a | 00000000 | 00000000 | 00000119                                     | 2**0 |
|     |                   |          |          |          | CONTENTS, READONLY                           |      |

<sup>2</sup> Объектный файл— 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.

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

```
bubble.o:      file format elf32-littleriscv
```

```
Sections:
```

| Idx | Name   | Size     | VMA      | LMA      | File off | Algn |
|-----|--|----------|----------|----------|----------|------|
| 0   | .text  | 0000008c | 00000000 | 00000000 | 00000034 | 2**2 |
|     | CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE |          |          |          |          |      |
| 1   | .data  | 00000000 | 00000000 | 00000000 | 000000c0 | 2**0 |
|     | CONTENTS, ALLOC, LOAD, DATA                  |          |          |          |          |      |
| 2   | .bss   | 00000000 | 00000000 | 00000000 | 000000c0 | 2**0 |
|     | ALLOC  |          |          |          |          |      |
| 3   | .sbss  | 00000004 | 00000000 | 00000000 | 000000c0 | 2**2 |
|     | ALLOC  |          |          |          |          |      |
| 4   | .comment                                     | 00000029 | 00000000 | 00000000 | 000000c0 | 2**0 |
|     | CONTENTS, READONLY                           |          |          |          |          |      |
| 5   | .riscv.attributes                            | 0000001a | 00000000 | 00000000 | 000000e9 | 2**0 |
|     | CONTENTS, READONLY                           |          |          |          |          |      |

Вся информация размещается в секциях.

| Секция   | Назначение  |
|----------|---|
| .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 использует архитектура little-endian RISC-V. Рассмотрим некоторые секции поближе.

### Листинг 2.7 Дизассемблированный файл main.o

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

```
main.o:      file format elf32-littleriscv
```

Disassembly of section .text:

00000000 <main>:

```

0:  fe010113      addi    sp,sp,-32
4:  00112e23      sw      ra,28(sp)
8:  00812c23      sw      s0,24(sp)
c:  00912a23      sw      s1,20(sp)
10: 01212823      sw      s2,16(sp)
14: 01312623      sw      s3,12(sp)
18: 00000437      lui     s0,0x0
1c: 00040493      addi    s1,s0,0 # 0 <main>
20: 01848913      addi    s2,s1,24
24: 00040413      addi    s0,s0,0
28: 000009b7      lui     s3,0x0
```

0000002c <.L2>:

```

2c: 00042583      lw      a1,0(s0)
30: 00098513      addi    a0,s3,0 # 0 <main>
34: 00000097      auipc   ra,0x0
38: 000080e7      jalr    ra,0(ra) # 34 <.L2+0x8>
3c: 00440413      addi    s0,s0,4
40: ff2416e3      bne     s0,s2,2c <.L2>
44: 00a00513      addi    a0,zero,10
48: 00000097      auipc   ra,0x0
4c: 000080e7      jalr    ra,0(ra) # 48 <.L2+0x1c>
50: 00600593      addi    a1,zero,6
54: 00000537      lui     a0,0x0
58: 00050513      addi    a0,a0,0 # 0 <main>
5c: 00000097      auipc   ra,0x0
60: 000080e7      jalr    ra,0(ra) # 5c <.L2+0x30>
64: 00000437      lui     s0,0x0
```

00000068 <.L3>:

```

68: 0004a583      lw      a1,0(s1)
6c: 00040513      addi    a0,s0,0 # 0 <main>
70: 00000097      auipc   ra,0x0
74: 000080e7      jalr    ra,0(ra) # 70 <.L3+0x8>
```

|     |          |      |                |
|-----|----------|------|----------------|
| 78: | 00448493 | addi | s1,s1,4        |
| 7c: | ff2496e3 | bne  | s1,s2,68 <.L3> |
| 80: | 00000513 | addi | a0,zero,0      |
| 84: | 01c12083 | lw   | ra,28(sp)      |
| 88: | 01812403 | lw   | s0,24(sp)      |
| 8c: | 01412483 | lw   | s1,20(sp)      |
| 90: | 01012903 | lw   | s2,16(sp)      |
| 94: | 00c12983 | lw   | s3,12(sp)      |
| 98: | 02010113 | addi | sp,sp,32       |
| 9c: | 00008067 | jalr | zero,0(ra)     |

Можно заметить комбинации инструкций auipc + jalr, которые на самом деле являются одной псевдоинструкцией call. Так же наблюдается выход из метода main.

| Листинг 2.7 Содержание секции .comment            |                               |
|---|-------------------------------|
| riscv64-unknown-elf-objdump -s -j .comment main.o |                               |
| main.o:   | file format elf32-littleriscv |
| Contents of section .comment:                     |                               |
| 0000 00474343 3a202853 69466976 65204743          | .GCC: (SiFive GC              |
| 0010 4320382e 322e302d 32303139 2e30352e          | C 8.2.0-2019.05.              |
| 0020 33292038 2e322e30 00                         | 3) 8.2.0.                     |

Тут ничего особенного не наблюдается, всё как в main.s.

Рассмотрим таблицу символов:

| Листинг 2.8 Таблица символов                   |                               |
|--|-------------------------------|
| riscv64-unknown-elf-objdump -t bubble.o main.o |                               |
| bubble.o:                                      | file format elf32-littleriscv |
| SYMBOL TABLE:                                  |                               |
| 00000000 1                                     | df *ABS* 00000000 bubble.c    |
| 00000000 1                                     | d .text 00000000 .text        |
| 00000000 1                                     | d .data 00000000 .data        |
| 00000000 1                                     | d .bss 00000000 .bss          |
| 00000000 1                                     | O .sbss 00000004 f.2552       |
| 00000000 1                                     | d .sbss 00000000 .sbss        |
| 00000088 1                                     | .text 00000000 .L1            |
| 00000074 1                                     | .text 00000000 .L3            |
| 00000064 1                                     | .text 00000000 .L9            |
| 00000040 1                                     | .text 00000000 .L5            |
| 00000080 1                                     | .text 00000000 .L10           |

```

00000030 l      .text 00000000 .L11
00000044 l      .text 00000000 .L4
00000000 l      d .comment      00000000 .comment
00000000 l      d .riscv.attributes 00000000 .riscv.attributes
00000000 g      F .text 0000008c bubble_sort

```

```
main.o:      file format elf32-littleriscv
```

#### SYMBOL TABLE:

```

00000000 l      df *ABS* 00000000 main.c
00000000 l      d .text 00000000 .text
00000000 l      d .data 00000000 .data
00000000 l      d .bss 00000000 .bss
00000000 l      .data 00000000 .LANCHOR0
00000000 l      O .data 00000018 array
00000000 l      d .rodata.str1.4 00000000 .rodata.str1.4
00000000 l      .rodata.str1.4 00000000 .LC0
0000002c l      .text 00000000 .L2
00000068 l      .text 00000000 .L3
00000000 l      d .comment      00000000 .comment
00000000 l      d .riscv.attributes 00000000 .riscv.attributes
00000000 g      F .text 000000a0 main
00000000      *UND* 00000000 printf
00000000      *UND* 00000000 putchar
00000000      *UND* 00000000 bubble_sort

```

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**:

Листинг 2.9 Таблица перемещений

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

```
bubble.o:      file format elf32-littleriscv
```

#### RELOCATION RECORDS FOR [.text]:

| OFFSET   | TYPE           | VALUE  |
|----------|----------------|--------|
| 00000008 | R_RISCV_HI20   | f.2552 |
| 00000008 | R_RISCV_RELAX  | *ABS*  |
| 0000000c | R_RISCV_LO12_I | f.2552 |

```

0000000c R_RISCV_RELAX    *ABS*
00000030 R_RISCV_HI20     f.2552
00000030 R_RISCV_RELAX    *ABS*
00000038 R_RISCV_LO12_S   f.2552
00000038 R_RISCV_RELAX    *ABS*
00000080 R_RISCV_HI20     f.2552
00000080 R_RISCV_RELAX    *ABS*
00000084 R_RISCV_LO12_S   f.2552
00000084 R_RISCV_RELAX    *ABS*
00000004 R_RISCV_BRANCH    .L1
00000014 R_RISCV_BRANCH    .L1
0000002c R_RISCV_JAL       .L3
00000044 R_RISCV_BRANCH    .L9
00000050 R_RISCV_BRANCH    .L5
00000060 R_RISCV_JAL       .L5
00000068 R_RISCV_BRANCH    .L10
00000070 R_RISCV_BRANCH    .L11
0000007c R_RISCV_JAL       .L4

```

main.o: file format elf32-littleriscv

#### RELOCATION RECORDS FOR [.text]:

| OFFSET   | TYPE           | VALUE       |
|----------|----------------|-------------|
| 00000018 | R_RISCV_HI20   | .LANCHOR0   |
| 00000018 | R_RISCV_RELAX  | *ABS*       |
| 0000001c | R_RISCV_LO12_I | .LANCHOR0   |
| 0000001c | R_RISCV_RELAX  | *ABS*       |
| 00000024 | R_RISCV_LO12_I | .LANCHOR0   |
| 00000024 | R_RISCV_RELAX  | *ABS*       |
| 00000028 | R_RISCV_HI20   | .LC0        |
| 00000028 | R_RISCV_RELAX  | *ABS*       |
| 00000030 | R_RISCV_LO12_I | .LC0        |
| 00000030 | R_RISCV_RELAX  | *ABS*       |
| 00000034 | R_RISCV_CALL   | printf      |
| 00000034 | R_RISCV_RELAX  | *ABS*       |
| 00000048 | R_RISCV_CALL   | putchar     |
| 00000048 | R_RISCV_RELAX  | *ABS*       |
| 00000054 | R_RISCV_HI20   | .LANCHOR0   |
| 00000054 | R_RISCV_RELAX  | *ABS*       |
| 00000058 | R_RISCV_LO12_I | .LANCHOR0   |
| 00000058 | R_RISCV_RELAX  | *ABS*       |
| 0000005c | R_RISCV_CALL   | bubble_sort |
| 0000005c | R_RISCV_RELAX  | *ABS*       |
| 00000064 | R_RISCV_HI20   | .LC0        |
| 00000064 | R_RISCV_RELAX  | *ABS*       |
| 0000006c | R_RISCV_LO12_I | .LC0        |
| 0000006c | R_RISCV_RELAX  | *ABS*       |
| 00000070 | R_RISCV_CALL   | printf      |
| 00000070 | R_RISCV_RELAX  | *ABS*       |
| 00000040 | R_RISCV_BRANCH | .L2         |
| 0000007c | R_RISCV_BRANCH | .L3         |



В таблице перемещений для main.o наблюдаем вызов метода bubble\_sort. Записи типа “R\_RISCV\_RELAX” заносятся в таблицу перемещений в дополнение к записям типа “R\_RISCV\_CALL” (и некоторым другим) и сообщают компоновщику, что пара инструкций, обеспечивающих вызов подпрограммы, может быть оптимизирована.

### Компоновка

Выполним компоновку следующей командой:

```
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -v main.o bubble.o -o main.out
>log_out.txt 2>&1
```

В результате выполнения команды получаем main.out – исполняемый бинарный файл. Изучим его секцию кода.

#### Листинг 2.10 Исполняемый файл (часть)

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

```
00010188 <main>:
 10188: fe010113      addi    sp,sp,-32
 1018c: 00112e23      sw      ra,28(sp)
 10190: 00812c23      sw      s0,24(sp)
 10194: 00912a23      sw      s1,20(sp)
 10198: 01212823      sw      s2,16(sp)
 1019c: 01312623      sw      s3,12(sp)
 101a0: 00022437      lui     s0,0x22
 101a4: 0c040493      addi    s1,s0,192 # 220c0 <__DATA_BEGIN__>
 101a8: 01848913      addi    s2,s1,24
 101ac: 0c040413      addi    s0,s0,192
 101b0: 000219b7      lui     s3,0x21
 101b4: 00042583      lw      a1,0(s0)
 101b8: 31898513      addi    a0,s3,792 # 21318 <__clzsi2+0x50>
101bc: 344000ef      jal     ra,10500 <printf>
 101c0: 00440413      addi    s0,s0,4
 101c4: ff2418e3      bne     s0,s2,101b4 <main+0x2c>
 101c8: 00a00513      addi    a0,zero,10
 101cc: 38c000ef      jal     ra,10558 <putchar>
 101d0: 00600593      addi    a1,zero,6
 101d4: 00022537      lui     a0,0x22
 101d8: 0c050513      addi    a0,a0,192 # 220c0 <__DATA_BEGIN__>
101dc: 03c000ef      jal     ra,10218 <bubble_sort>
 101e0: 00021437      lui     s0,0x21
 101e4: 0004a583      lw      a1,0(s1)
 101e8: 31840513      addi    a0,s0,792 # 21318 <__clzsi2+0x50>
101ec: 314000ef      jal     ra,10500 <printf>
 101f0: 00448493      addi    s1,s1,4
 101f4: ff2498e3      bne     s1,s2,101e4 <main+0x5c>
```

|                         |          |      |                                       |
|-------------------------|----------|------|---------------------------------------|
| 101f8:                  | 00000513 | addi | a0,zero,0                             |
| 101fc:                  | 01c12083 | lw   | ra,28(sp)                             |
| 10200:                  | 01812403 | lw   | s0,24(sp)                             |
| 10204:                  | 01412483 | lw   | s1,20(sp)                             |
| 10208:                  | 01012903 | lw   | s2,16(sp)                             |
| 1020c:                  | 00c12983 | lw   | s3,12(sp)                             |
| 10210:                  | 02010113 | addi | sp,sp,32                              |
| 10214:                  | 00008067 | jalr | zero,0(ra) # 10174 <frame_dummy+0x20> |
| 00010218 <bubble_sort>: |          |      |                                       |
| 10218:                  | fff58e13 | addi | t3,a1,-1                              |
| 1021c:                  | 060e0c63 | beq  | t3,zero,10294 <bubble_sort+0x7c>      |
| 10220:                  | 1e41a703 | lw   | a4,484(gp) # 22aa4 <_edata>           |
| 10224:                  | 00100793 | addi | a5,zero,1                             |
| 10228:                  | 06f70663 | beq  | a4,a5,10294 <bubble_sort+0x7c>        |
| 1022c:                  | 00259613 | slli | a2,a1,0x2                             |
| 10230:                  | 00c50633 | add  | a2,a0,a2                              |
| 10234:                  | 00000893 | addi | a7,zero,0                             |
| 10238:                  | 00100313 | addi | t1,zero,1                             |
| 1023c:                  | 00000813 | addi | a6,zero,0                             |
| 10240:                  | 0440006f | jal  | zero,10284 <bubble_sort+0x6c>         |
| 10244:                  | 00100713 | addi | a4,zero,1                             |
| 10248:                  | 1ee1a223 | sw   | a4,484(gp) # 22aa4 <_edata>           |
| 1024c:                  | 00008067 | jalr | zero,0(ra)                            |
| 10250:                  | 00478793 | addi | a5,a5,4                               |
| 10254:                  | 02c78063 | beq  | a5,a2,10274 <bubble_sort+0x5c>        |
| 10258:                  | ffc7a703 | lw   | a4,-4(a5)                             |
| 1025c:                  | 0007a683 | lw   | a3,0(a5)                              |
| 10260:                  | fee6f8e3 | bgeu | a3,a4,10250 <bubble_sort+0x38>        |
| 10264:                  | fed7ae23 | sw   | a3,-4(a5)                             |
| 10268:                  | 00e7a023 | sw   | a4,0(a5)                              |
| 1026c:                  | 00080593 | addi | a1,a6,0                               |
| 10270:                  | fe1ff06f | jal  | zero,10250 <bubble_sort+0x38>         |
| 10274:                  | 00188893 | addi | a7,a7,1                               |
| 10278:                  | 01c88c63 | beq  | a7,t3,10290 <bubble_sort+0x78>        |
| 1027c:                  | ffc60613 | addi | a2,a2,-4                              |
| 10280:                  | fc6582e3 | beq  | a1,t1,10244 <bubble_sort+0x2c>        |
| 10284:                  | 00450793 | addi | a5,a0,4                               |
| 10288:                  | 00030593 | addi | a1,t1,0                               |
| 1028c:                  | fc9ff06f | jal  | zero,10254 <bubble_sort+0x3c>         |
| 10290:                  | 1eb1a223 | sw   | a1,484(gp) # 22aa4 <_edata>           |
| 10294:                  | 00008067 | jalr | zero,0(ra)                            |

Адресация для вызовов функций изменилась на абсолютную.

### 3. Создание статической библиотеки<sup>3</sup>

Выделим функция `bubble_sort` в отдельную статическую библиотеку. Для этого надо получить объектный файл `bubble.o` и собрать библиотеку.

```
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -O1 -c bubble.c -o bubble.o  
riscv64-unknown-elf-ar -rsc libBubble.a bubble.o
```

Рассмотрим список символов библиотеки:

| Листинг 3.1 Список символов libBubble.a  |
|--|
| <pre>riscv64-unknown-elf-nm libBubble.a<br/><br/>bubble.o:<br/>00000088 t .L1<br/>00000080 t .L10<br/>00000030 t .L11<br/>00000074 t .L3<br/>00000044 t .L4<br/>00000040 t .L5<br/>00000064 t .L9<br/>00000000 T bubble_sort<br/>00000000 s f.2552</pre> |

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

Используя собранную библиотеку, произведём исполняемый файл тестовой программы.

```
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -O1 main.c libBubble.a -o  
main.out
```

---

<sup>3</sup> 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.

Посмотрим таблицу символов исполняемого файла и убедимся, что там находится наша функция.

Листинг 3.2 Таблица символов main.out

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

```
321 00023b84 g      .bss  00000000 _end
322 0001a3bc g      F .text 00000114 __fputwc
323 00010278 g      F .text 00000080 bubble_sort
324 00017020 g      F .text 00000084 __swrite
325 00023b20 g      O .sdata 00000004 malloc trim thre
```

В состав нашей программы вошло содержание объектного файла bubble.o.

### Создание make-файлов

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

#### make\_lib

```
CC=riscv64-unknown-elf-gcc
AR=riscv64-unknown-elf-ar
CFLAGS=-march=rv32i -mabi=ilp32
```

```
all: lib
```

```
lib: bubble.o
```

```
$(AR) -rsc libBubble.a bubble.o
```

```
del -f *.o
```

```
bubble.o: bubble.c
```

```
$(CC) $(CFLAGS) -c bubble.c -o bubble.o
```

#### make\_app

```
TARGET=main
```

```
CC=riscv64-unknown-elf-gcc
```

```
CFLAGS=-march=rv32i -mabi=ilp32
```

```
all:
```

```
make -f make_lib
```

```
$(CC) $(CFLAGS) main.c libBubble.a -o $(TARGET)
```

```
del -f *.o *.a
```

Для создания библиотеки необходимо выполнить make\_lib, а для приложения make\_app.

```

Содержимое папки D:\projects\lowlevelprog\lowlevelprog\lab4\lib

16.04.2021  19:49    <DIR>          .
16.04.2021  19:49    <DIR>          ..
16.04.2021  17:38             530 bubble.c
16.04.2021  17:39             99 bubble.h
16.04.2021  17:42            978 main.c
16.04.2021  19:48            175 make_app
16.04.2021  19:45            236 make_lib
                5 файлов                2 018 байт
                2 папок   906 127 167 488 байт свободно

D:\projects\lowlevelprog\lowlevelprog\lab4\lib>make -f make_lib
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -c bubble.c -o bubble.o
riscv64-unknown-elf-ar -rsc libBubble.a bubble.o
del -f *.o

D:\projects\lowlevelprog\lowlevelprog\lab4\lib>dir
Том в устройстве D не имеет метки.
Серийный номер тома: 9242-2F94

Содержимое папки D:\projects\lowlevelprog\lowlevelprog\lab4\lib

16.04.2021  19:51    <DIR>          .
16.04.2021  19:51    <DIR>          ..
16.04.2021  17:38             530 bubble.c
16.04.2021  17:39             99 bubble.h
16.04.2021  19:51             1 588 libBubble.a
16.04.2021  17:42            978 main.c
16.04.2021  19:48            175 make_app
16.04.2021  19:45            236 make_lib
                6 файлов                3 606 байт
                2 папок   906 127 163 392 байт свободно

D:\projects\lowlevelprog\lowlevelprog\lab4\lib>

D:\projects\lowlevelprog\lowlevelprog\lab4\lib>make -f make_app
make -f make_lib
make[1]: Entering directory 'D:/projects/lowlevelprog/lowlevelprog/lab4/lib'
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -c bubble.c -o bubble.o
riscv64-unknown-elf-ar -rsc libBubble.a bubble.o
del -f *.o
make[1]: Leaving directory 'D:/projects/lowlevelprog/lowlevelprog/lab4/lib'
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 main.c libBubble.a -o main
del -f *.o *.a

D:\projects\lowlevelprog\lowlevelprog\lab4\lib>dir
Том в устройстве D не имеет метки.
Серийный номер тома: 9242-2F94

Содержимое папки D:\projects\lowlevelprog\lowlevelprog\lab4\lib

16.04.2021  19:51    <DIR>          .
16.04.2021  19:51    <DIR>          ..
16.04.2021  17:38             530 bubble.c
16.04.2021  17:39             99 bubble.h
16.04.2021  19:51            218 864 main
16.04.2021  17:42            978 main.c
16.04.2021  19:48            175 make_app
16.04.2021  19:45            236 make_lib
                6 файлов                220 882 байт
                2 папок   906 126 946 304 байт свободно

D:\projects\lowlevelprog\lowlevelprog\lab4\lib>

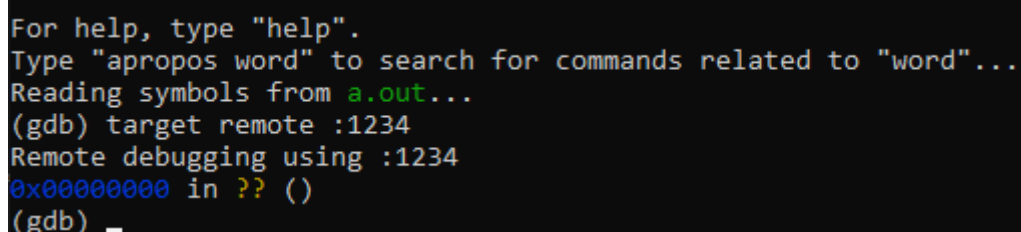
```

Рис. 3.1 Выполнение make-файлов

## 4. Bonus: Запуск выполняемых файлов

Попробуем запустить исполняемые файлы для архитектуры RISCv32 на процессоре x86. Для начала необходимо установить пакет для эмуляции аппаратного обеспечения QEMU. Запустим QEMU с GDB сервером на localhost:1234, а после подключимся к нему. Делается это следующими командами:

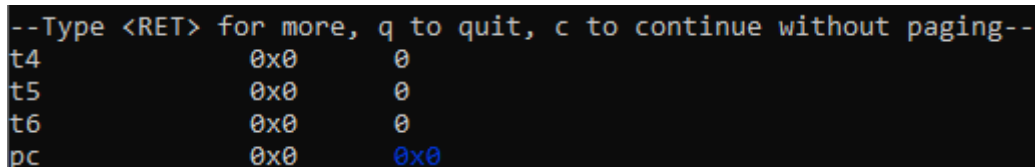
|  |
|--|
| <pre>qemu-system-riscv32 -machine virt -m 128M -gdb tcp::1234 -kernel a.out -bios none</pre> |
| <pre>riscv64-unknown-elf-gdb a.out</pre>   |



```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from a.out...
(gdb) target remote :1234
Remote debugging using :1234
0x00000000 in ?? ()
(gdb) _
```

Рис. 4.1 Результат работы программы

Программа, к сожалению, не сработала. Посмотрим где находится счётчик инструкций.



```
--Type <RET> for more, q to quit, c to continue without paging--
t4          0x0      0
t5          0x0      0
t6          0x0      0
pc          0x0      0x0
```

Рис. 4.2 Значение регистра pc

Можно заметить, что GDB не может разобраться в строке, на которой надо выполнить инструкции и program counter указывает на “нулевой адрес”. Опираясь на данные из интернета можно предположить что дело тут в настройке стека. Для решения проблемы следует выполнить компоновку чуть другим образом.

## Настройка машины

Для начала разберёмся с виртуальной машиной -virt (по умолчанию qemu использует RV32). Сбросим данные о машине в dtb файл и после отредактируем его.

```
qemu-system-riscv32 -machine virt -machine dumpdtb=riscv32-virt.dtb
```

На выходе получаем dtb файл. Он является бинарным, поэтому с помощью утилиты командной строки откроем этот файл как текст.

```
sudo apt-get install -y device-tree-compiler  
dtc -I dtb -O dts -o riscv32-virt.dts riscv32-virt.dtb
```

В открывшемся файле можно увидеть много информации о виртуальной машине. Нам необходима секция связанная с памятью.



```
/dts-v1/;  
{  
    #address-cells = <0x2>;  
    #size-cells = <0x2>;  
    compatible = "riscv-virtio";  
    model = "riscv-virtio,qemu";  
  
    fw-cfg@10100000 {  
        dma-coherent;  
        reg = <0x0 0x10100000 0x0 0x18>;  
        compatible = "qemu,fw-cfg-mmio";  
    };  
  
    chosen {  
        bootargs = [00];  
        stdout-path = "/soc/uart@10000000";  
    };  
  
    memory@80000000 {  
        device_type = "memory";  
        reg = <0x0 0x80000000 0x0 0x80000000>;  
    };  
  
    cpus {
```

Рис 4.3 Секция memory

Теперь мы можем сказать откуда начинается банк памяти. В свойстве reg указано,





```

OUTPUT_ARCH(riscv)
MEMORY
{
    RAM (rwx) : ORIGIN = 0x80000000, LENGTH = 128M
}

```

Рис. 4.5 Добавленные строки

Этой конструкцией мы задали блок памяти с доступом на чтение запись и хранения инструкций. Так же определили его начало и размер, который мы узнали ранее. Далее определяем символ “\_\_stack\_top”.

```

PROVIDE(__stack_top = ORIGIN(RAM) + LENGTH(RAM));

```

-march=rv32i -mabi=ilp32

Теперь перейдём к сборке исполняемого файла:

```

riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -g -ffreestanding -O0 -Wl,--gc-
sections -nostartfiles -Wl,-T,elf32.ld crt0.s main.c bubble.c

```

Тут мы указали свой “crt0.s” который имеет следующее содержание:

Листинг 4.1 crt0.s

```

.section .init, "ax"
.global _start
_start:
    .cfi_startproc
    .cfi_undefined ra
    .option push
    .option norelax
    la gp, __global_pointer$
    .option pop
    la sp, __stack_top
    add s0, sp, zero
    jal zero, main
    .cfi_endproc
.end

```

По итогу pc не может установиться на 0x80000000

```

riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -g -ffreestanding -O0 -Wl,--gc-
sections -nostartfiles -Wl,-T,elf32.ld crt0.s main.c bubble.c

```

```

1
2 a.out:      file format elf32-littleriscv
3
4
5 Disassembly of section .init:
6
7 80000000 <_start>:
8 80000000: 00012197      auipc    gp,0x12
9 80000004: 31818193      addi     gp,gp,792 # 80012318 <__global_pointer$>
10 80000008: 08000117      auipc    sp,0x8000
11 8000000c: ff810113      addi     sp,sp,-8 # 88000000 <__stack_top>
12 80000010: 00010433      add     s0,sp,zero
13 80000014: 0040006f      jal     zero,80000018 <main>
14
15 Disassembly of section .text:
16
17 80000018 <main>:
18 80000018: fe010113      addi     sp,sp,-32
19 8000001c: 00112e23      sw      ra,28(sp)
20 80000020: 00812c23      sw      s0,24(sp)
21 80000024: 02010413      addi     s0,sp,32
22 80000028: fe042623      sw      zero,-20(s0)
23 8000002c: 0380006f      jal     zero,80000064 <main+0x4c>
24 80000030: 800127b7      lui     a5,0x80012
25 80000034: fec42703      lw      a4,-20(s0)
26 80000038: 00271713      slli     a4,a4,0x2
27 8000003c: b1878793      addi     a5,a5,-1256 # 80011b18 <__stack_top+0xf8011b18>
28 80000040: 00f707b3      add     a5,a4,a5
29 80000044: 0007a783      lw      a5,0(a5)
30 80000048: 00078593      addi     a1,a5,0
31 8000004c: 800117b7      lui     a5,0x80011
32 80000050: d5878513      addi     a0,a5,-680 # 80010d58 <__stack_top+0xf8010d58>
33 80000054: 1fc000ef      jal     ra,80000250 <printf>
34 80000058: fec42783      lw      a5,-20(s0)
35 8000005c: 00178793      addi     a5,a5,1
36 80000060: fef42623      sw      a5,-20(s0)
37 80000064: 00600793      addi     a5,zero,6
38 80000068: fec42703      lw      a4,-20(s0)

```

## Вывод

В ходе выполнения лабораторной работы была написана программа на языке C с заданной функциональностью. После была выполнена сборка этой программы по шагам для архитектуры команд RISC-V. Были проанализированы выводы препроцессора, компилятора и линковщика отдельно друг от друга. Была создана своя статически линкуемая библиотека libBubble.a. Были написаны make-файлы для её сборки, а также сборки тестовой программы с использованием библиотеки.