Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

Отчёт по лабораторной работе № 3

Дисциплина: Низкоуровневое программирование

Тема: Раздельная компиляция

Вариант: 18

Выполнил студент гр. 3530901/90002 _____Н.С. Якубец

(подпись)

Принял преподаватель _____ Д.С. Степанов

(подпись)

“_____” _____ 2021 г.

Санкт-Петербург

2021

Задание

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.

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.

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 18: Циклический сдвиг массива чисел.

Алгоритм

На каждом шаге своей работы алгоритм замещает элемент el1 на el2. Адрес el2 на каждом шаге считается следующим образом:

el2_addr = (el1_addr + num_of_shifts) % array_length

После замещения в массиве el2 на el1, el2 помещается в регистр, хранящий el1, и, вычисляя адрес нового el2, программа помещает новый el2 в соответствующий регистр. Такие шаги производятся array_length раз, так как каждый элемент изменил свое положение в результате сдвигов и должен быть заменен в исходном массиве.

Программа

Была реализована программа в файлах main.c и reverse.c, циклически сдвигающая указанный массив указанное число раз в указанном направлении. main – основная программа, вызывающая функцию reverse, которая и производит необходимые действия с заданным массивом. Код программы приводится на рис. 1-2.

*main.c reverse.c

```c
#include <stdlib.h>

void reverse(int* array, int size, int moves_num, int direction);

int main() {
    int size = 11;
    int* array = calloc(size, 4);
    for (int i = 0; i < size; i++) array[i] = i;

    reverse(array, size, 3, 1);

    free(array);
    return 0;


}

```

Рис. 1 Исходный код подпрограммы main

reverse.c

```c

void reverse(int* array, int size, int moves_num, int direction) {
    if(size < 2) return;
    moves_num = moves_num % size;
    if (direction != 0) moves_num = size - moves_num;
    int current_el_index = 0;
    int el_to_move;
    int el_to_save;
    int temp_index;
    int i = 0;
    temp_index = (current_el_index + moves_num) % size;
    el_to_move = array[0];
    el_to_save = array[temp_index];
    for (i = 0; i < size; i++) {
        array[temp_index] = el_to_move;
        el_to_move = el_to_save;
        current_el_index = temp_index;
        temp_index = (current_el_index + moves_num) % size;
        el_to_save = array[temp_index];
    }

    return;
}

```

Рис. 2 Исходный код подпрограммы reverse

Компиляция

Для запуска процесса компиляции была выполнена следующая команда в командной строке

```
riscv64-unknown-elf-gcc --save-temps -march=rv32imc -
mabi=ilp32 -O1 -v main.c reverse.c >log 2>&1
```

В результате чего были получены файлы a.out, log, main.o, main.i, main.s reverse.o, reverse.i, reverse.s.

Препроцессор

В результате работы препроцессора были получены два файла: main.i и reverse.i. Содержимое файла reverse не сильно изменилось по сравнению с исходным кодом (рис. 3), лишь добавились специальные директивы, необходимые для передаче информации от препроцессора в компилятор.

Содержимое же файла main.i многократно увеличилось по сравнению с исходным кодом. Это произошло в следствие того, что в файле main.c была подключена библиотека stdlib и заголовок reverse.h.

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

void reverse(int* array, int size, int moves_num, int direction) {
    if(size < 2) return;
    moves_num = moves_num % size;
    if (direction != 0) moves_num = size - moves_num;
    int current_el_index = 0;
    int el_to_move;
    int el_to_save;
    int temp_index;
    int i = 0;
    temp_index = (current_el_index + moves_num) % size;
    el_to_move = array[0];
    el_to_save = array[temp_index];
    for (i = 0; i < size; i++) {
        array[temp_index] = el_to_move;
        el_to_move = el_to_save;
        current_el_index = temp_index;
        temp_index = (current_el_index + moves_num) % size;
        el_to_save = array[temp_index];
    }

    return;
}

```

Рис. 3 Содержимое файла reverse.i

Компилятор

В результате работы компилятора в заданной папке появились файлы main.s и reverse.s в которых находится текст на языке ассемблера, являющийся результатом комплияции исходного кода (рис. 4-5).

Анализируя файл main.s, можно выделить несколько этапов:

Таблица 1 Анализ файла main.s

Строчки файла	Комментарий
11-13	Сохранение адреса возврата и изначального значения s0.

14-15	Загрузка в регистры a0, a1 аргументов функции calloc, использующейся для выделения памяти под массив array.
17-20	Команды по подготовке значений регистров для цикла формирования массива array. В регистр a4 помещается адрес массива array, который функция calloc() поместила в регистр a0. В a5 помещается изначальное значение(0) счетчика циклов. В a1 помещается размер каждого элемента массива в байтах(4). В a3 помещается конечное значение счетчика(11).
22-25	Непосредственно тело цикла. В первой команде значение элемента массива помещается непосредственно в память по указателю, находящемуся в a4. Далее получается значение следующего элемента массива. А в третьей команде увеличивается указатель на массив на 4 байта. Последней командой является условный переход, реализующий повторение цикла в случае если заполнено было меньше 11 элементов массива.
26-30	В регистры a3-a0 помещаются аргументы функции reverse. В a3 – направление сдвига(1). В a2 – число сдвигов. В a1 – длина массива. В a0 – адрес начала массива. Далее производится вызов функции reverse().
31-32	В a0 снова помещается адрес массива, необходимый в качестве аргумента функции free(), и производится вызов функции free().
33-37	Эти функции выполняют восстановление изначальных значений регистров a0, s0 и получение адреса возврата, который помещается в ra. В предпоследней команде освобождается место в стеке. Последняя команда осуществляет безусловный переход по адресу возврата в регистре ra.

main.s

```
    .file   "main.c"
    .option nopic
    .attribute arch, "rv32i2p0_m2p0_c2p0"
    .attribute unaligned_access, 0
    .attribute stack_align, 16
    .text
    .align  1
    .globl  main
    .type   main, @function
main:
    addi    sp,sp,-16
    sw  ra,12(sp)
    sw  s0,8(sp)
    li  a1,4
    li  a0,11
    call    calloc
    mv  s0,a0
    mv  a4,a0
    li  a5,0
    li  a3,11
.L2:
    sw  a5,0(a4)
    addi    a5,a5,1
    addi    a4,a4,4
    bne a5,a3,.L2
    li  a3,1
    li  a2,3
    li  a1,11
    mv  a0,s0
    call    reverse
    mv  a0,s0
    call    free
    li  a0,0
    lw  ra,12(sp)
    lw  s0,8(sp)
    addi    sp,sp,16
    jr  ra
    .size   main, .-main
    .ident  "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"
```

Рис. 4 Содержимое файла main.s

Проанализирую содержимое файла reverse.s:

Таблица 2 Анализа файла reverse.s

Строчки файла	Комментарий

11-12	Проверка размера массива. В случае если массив состоит из 1 элемента или меньше, то производится переход в конец программы.
13	Из заданного количества сдвигов(a2) убираются целые «круги» сдвигов. a7 = a2 % a1 = shift_num % array_length
14-15	Производится проверка направления сдвига, и в зависимости от его значения формируется необходимое число сдвигов.
17	В регистре a5 формируется первоначальное значение temp_index
18	В a6 помещается значение array[0]
19-20	В a4 формируется смещение элемента array[temp_index] в байтах относительно array. После чего в a4 добавляется абсолютный адрес array и формируется абсолютный адрес array[temp_index].
21	В a2 помещается array[temp_index]
22	В a3 помещается начальное значение счетчика итераций цикла – 0.
24-26	Начало основного цикла программы, в котором сначала производится сохранение замещаемого элемента в a6.
27-28	Формируется новый temp_index.
29	В a6 сохраняется замещаемый элемент.
30-32	Формируется абсолютный адрес элемента array[temp_index]. После чего он помещается в a2.
33	Счетчик циклов учеличивается на 1.
34	Условный переход, позволяющий сформировать сам цикл.
36	Возврат из подпрограммы reverse.

reverse.s

```
    .file   "reverse.c"
    .option nopic
    .attribute arch, "rv32i2p0_m2p0_c2p0"
    .attribute unaligned_access, 0
    .attribute stack_align, 16
    .text
    .align  1
    .globl  reverse
    .type   reverse, @function
reverse:
    li  a5,1
    ble a1,a5,.L1
    rem a7,a2,a1
    beq a3,zero,.L3
    sub a7,a1,a7
.L3:
    rem a5,a7,a1
    lw  a6,0(a0)
    slli    a4,a5,2
    add a4,a0,a4
    lw  a2,0(a4)
    li  a3,0
.L4:
    slli    a4,a5,2
    add a4,a0,a4
    sw  a6,0(a4)
    add a5,a7,a5
    rem a5,a5,a1
    mv  a6,a2
    slli    a4,a5,2
    add a4,a0,a4
    lw  a2,0(a4)
    addi    a3,a3,1
    bne a1,a3,.L4
.L1:
    ret
    .size   reverse, .-reverse
    .ident  "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"

```

Рис. 5 Содержимое файла reverse.s

Объектный файл

В результате работы ассемблера были получены объектные файлы main.o и reverse.o. Проанализирую их заголовки, содержимое секций, таблицы символов и таблицы перемещений.

```
1
2  main.o:     file format elf32-littleriscv
3  architecture: riscv:rv32, flags 0x00000011:
4  HAS_RELOC, HAS_SYMS
5  start address 0x00000000
6
7
8  reverse.o:     file format elf32-littleriscv
9  architecture: riscv:rv32, flags 0x00000011:
10 HAS_RELOC, HAS_SYMS
11 start address 0x00000000
12
13
```

Рис. 6 Заголовки объектных файлов main.o и reverse.o

По флагам HAS_SYMS и HAS_RELOC можно понять, что оба файла содержат символы и таблицы перемещений соответственно. Так как объектный файл не должен содержать точки входа, то стартовый адрес равен нулю.

```
1
2  main.o:     file format elf32-littleriscv
3
4  SYMBOL TABLE:
5  00000000 l    df *ABS*  00000000 main.c
6  00000000 l    d  .text  00000000 .text
7  00000000 l    d  .data  00000000 .data
8  00000000 l    d  .bss   00000000 .bss
9  0000001a l       .text  00000000 .L2
10 00000000 l    d  .comment   00000000 .comment
11 00000000 l    d  .riscv.attributes  00000000 .riscv.attributes
12 00000000 g     F .text  00000048 main
13 00000000         *UND*  00000000 calloc
14 00000000         *UND*  00000000 reverse
15 00000000         *UND*  00000000 free
16
17
18
19 reverse.o:     file format elf32-littleriscv
20
21 SYMBOL TABLE:
22 00000000 l    df *ABS*  00000000 reverse.c
23 00000000 l    d  .text  00000000 .text
24 00000000 l    d  .data  00000000 .data
25 00000000 l    d  .bss   00000000 .bss
26 00000042 l       .text  00000000 .L1
27 00000010 l       .text  00000000 .L3
28 00000022 l       .text  00000000 .L4
29 00000000 l    d  .comment   00000000 .comment
30 00000000 l    d  .riscv.attributes  00000000 .riscv.attributes
31 00000000 g     F .text  00000044 reverse
32
33
34
```

Рис. 7 Таблицы символов файлов main.o и reverse.o

В таблице символов main.o можно заметить неопределенные функции calloc, reverse и free, так как компоновка программы еще не производилась. Однако функция main определена и отмечена как глобальная. Таблица символов reverse.o показывает, что компилятор определил единственную функцию reverse как глобальную, так как никаких других функций в коде не использовалось.

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

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000048  00000000  00000000  00000034  2**1
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  0000007c  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  0000007c  2**0
                  ALLOC
  3 .comment      00000031  00000000  00000000  0000007c  2**0
                  CONTENTS, READONLY
  4 .riscv.attributes 00000026  00000000  00000000  000000ad  2**0
                  CONTENTS, READONLY

reverse.o:     file format elf32-littleriscv

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000044  00000000  00000000  00000034  2**1
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  00000078  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  00000078  2**0
                  ALLOC
  3 .comment      00000031  00000000  00000000  00000078  2**0
                  CONTENTS, READONLY
  4 .riscv.attributes 00000026  00000000  00000000  000000a9  2**0
                  CONTENTS, READONLY
```

Рис. 8 Секции файлов main.o и reverse.o

В обеих таблицах можно заметить, что секции .text имеют отличный от нуля размер, тогда как секции .data пустые, так как в коде не определялось никаких отдельно стоящих структур, а все операции выделения памяти производились программно.

```
1
2  main.o:     file format elf32-littleriscv
3
4  RELOCATION RECORDS FOR [.text]:
5  OFFSET   TYPE              VALUE
6  0000000a R_RISCV_CALL      calloc
7  0000000a R_RISCV_RELAX     *ABS*
8  0000002c R_RISCV_CALL      reverse
9  0000002c R_RISCV_RELAX     *ABS*
10 00000036 R_RISCV_CALL      free
11 00000036 R_RISCV_RELAX     *ABS*
12 00000020 R_RISCV_BRANCH    .L2
13
14
15
16 reverse.o:     file format elf32-littleriscv
17
18 RELOCATION RECORDS FOR [.text]:
19 OFFSET   TYPE              VALUE
20 00000002 R_RISCV_BRANCH    .L1
21 0000000a R_RISCV_RVC_BRANCH  .L3
22 0000003e R_RISCV_BRANCH    .L4
23
```

Рис. 9 Таблицы перемещений файлов main.o и reverse.o

В таблице файла main.o можно увидеть записи перемещений для функций calloc, reverse и free, а также их смещения относительно начала файла. Так как в файле reverse.o не было использовано никаких сторонних функций, его таблица перемещений состоит только из меток ветвлений, находящихся внутри этого же файла.

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


Disassembly of section .text:

00000000 <main>:
   0:   1141                    c.addi  sp,-16
   2:   c606                    c.swsp  ra,12(sp)
   4:   c422                    c.swsp  s0,8(sp)
   6:   4591                    c.li    a1,4
   8:   452d                    c.li    a0,11
   a:   00000097            auipc   ra,0x0
   e:   000080e7            jalr    ra,0(ra) # a <main+0xa>
  12:   842a                    c.mv    s0,a0
  14:   872a                    c.mv    a4,a0
  16:   4781                    c.li    a5,0
  18:   46ad                    c.li    a3,11

0000001a <.L2>:
  1a:   c31c                    c.sw    a5,0(a4)
  1c:   0785                    c.addi  a5,1
  1e:   0711                    c.addi  a4,4
  20:   fed79de3            bne a5,a3,1a <.L2>
  24:   4685                    c.li    a3,1
  26:   460d                    c.li    a2,3
  28:   45ad                    c.li    a1,11
  2a:   8522                    c.mv    a0,s0
  2c:   00000097            auipc   ra,0x0
  30:   000080e7            jalr    ra,0(ra) # 2c <.L2+0x12>
  34:   8522                    c.mv    a0,s0
  36:   00000097            auipc   ra,0x0
  3a:   000080e7            jalr    ra,0(ra) # 36 <.L2+0x1c>
  3e:   4501                    c.li    a0,0
  40:   40b2                    c.lwsp  ra,12(sp)
  42:   4422                    c.lwsp  s0,8(sp)
  44:   0141                    c.addi  sp,16
  46:   8082                    c.jr    ra

```

Рис. 10 Дизассемблированный код файла main.o

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


Disassembly of section .text:

00000000 <reverse>:
   0:   4785                    c.li    a5,1
   2:   04b7d063            bge a5,a1,42 <.L1>
   6:   02b668b3            rem a7,a2,a1
   a:   c299                    c.beqz  a3,10 <.L3>
   c:   411588b3            sub a7,a1,a7

00000010 <.L3>:
  10:   02b8e7b3            rem a5,a7,a1
  14:   00052803            lw  a6,0(a0)
  18:   00279713            slli    a4,a5,0x2
  1c:   972a                    c.add   a4,a0
  1e:   4310                    c.lw    a2,0(a4)
  20:   4681                    c.li    a3,0

00000022 <.L4>:
  22:   00279713            slli    a4,a5,0x2
  26:   972a                    c.add   a4,a0
  28:   01072023            sw  a6,0(a4)
  2c:   97c6                    c.add   a5,a7
  2e:   02b7e7b3            rem a5,a5,a1
  32:   8832                    c.mv    a6,a2
  34:   00279713            slli    a4,a5,0x2
  38:   972a                    c.add   a4,a0
  3a:   4310                    c.lw    a2,0(a4)
  3c:   0685                    c.addi  a3,1
  3e:   fed592e3            bne a1,a3,22 <.L4>

00000042 <.L1>:
  42:   8082                    c.jr    ra

```

Рис. 11 Дизассемблированный код файла reverse.o

Просмотр кодов дизассемблированных файлов показывает, что данные коды отличаются от тех, что были получены в результате работы компилятора. Некоторые команды были заменены на формат команд расширения «C» архитектуры risc-v, под которое производилась компиляция. Причиной замены команд стала оптимизация, так как команды, добавляемые расширением «C» занимают не 32 бита, а лишь 16 бит, что делает код более компактным. Тем не менее анализ кода выявляет сходство с кодом, находящимся в файлах main.s reverse.s, что говорит о том, что этот код выполняет те же функции, что и рассмотренные ранее программы.

Компоновщик

Проанализирую исполняемый файл, выданный компоновщиком – a.out.

```
1
2  a.out:     file format elf32-littleriscv
3  architecture: riscv:rv32, flags 0x00000112:
4  EXEC_P, HAS_SYMS, D_PAGED
5  start address 0x00010088
6
```

Рис. 12 Заголовок файла a.out

Видно, что в отличие от объектных файлов в данном исполняемом файле появился ненулевой стартовый адрес. Метка EXEC_P указывает на то, что данный файл является исполняемым, а HAS_SYMS обозначает что файл содержит символы.

```
00011004 l       .init_array    00000000 __preinit_array_end
00011004 l       .init_array    00000000 __init_array_start
00011004 l       .init_array    00000000 __preinit_array_start
00010d1c g     F .text  00000058 cleanup_glue
00011810 g       *ABS*  00000000 __global_pointer$
00010920 g     F .text  00000002 __malloc_unlock
00010148 g     F .text  00000044 reverse
00010238 g     F .text  00000006 __errno
00011860 g     O .sbss  00000004 errno
00011840 g       .sdata 00000000 __SDATA_BEGIN__
000102c4 g     F .text  00000008 malloc
0001185c g     O .sbss  00000004 __malloc_top_pad
00011844 g     O .sdata 00000000 .hidden __dso_handle
00010922 g     F .text  0000002e _sbrk_r
00011858 g     O .sbss  00000004 __malloc_max_sbrked_mem
00011840 g     O .sdata 00000004 _global_impure_ptr
0001025a g     F .text  0000006a __libc_init_array
00010eae g     F .text  00000050 _sbrk
00010a0e g     F .text  00000038 __libc_fini_array
00010d74 g     F .text  000000a2 _reclaim_reent
0001018c g     F .text  0000000a calloc
00010b04 g     F .text  00000218 _free_r
00010950 g     F .text  000000be __call_exitprocs
0001184c g     O .sdata 00000004 __malloc_sbrk_base
00010088 g     F .text  0000003a _start
00010e20 g     F .text  0000006c __register_exitproc
00011884 g     O .bss   00000028 __malloc_current_mallinfo
000118b0 g       .bss   00000000 __BSS_END__
00011438 g     O .data  00000408 __malloc_av_
0001091e g     F .text  00000002 __malloc_lock
00010196 g     F .text  000000a2 _calloc_r
00011854 g       .sbss  00000000 __bss_start
00010876 g     F .text  000000a8 memset
00010112 g     F .text  00000036 main
00011854 g     O .sbss  00000004 __malloc_max_total_mem
000102d6 g     F .text  000005a0 _malloc_r
00010a46 g     F .text  000000be _malloc_trim_r
00010e16 g     F .text  0000000a atexit
00011848 g     O .sdata 00000004 _impure_ptr
00011010 g       .data  00000000 __DATA_BEGIN__
00011854 g       .sdata 00000000 _edata
000118b0 g       .bss   00000000 _end
00011850 g     O .sdata 00000004 __malloc_trim_threshold
0001023e g     F .text  0000001c exit
00010e8c g     F .text  00000022 _exit
000102cc g     F .text  0000000a free
```

Рис. 13 Часть таблицы символов файла a.out

Таблица символов файла a.out значительно увеличилась за счет функций, добавленных библиотекой stdlib.h. Тем не менее в ней все равно сохраняются метки используемых функций calloc, main, reverse, которые по-прежнему являются

глобальными. По адресу 0x10088 можно заметить функцию __start, которая является точкой входа в программу в соответствии со значением стартового адреса из заголовка файла a.out.

```
a.out:     file format elf32-littleriscv

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000e8a  00010074  00010074  00000074  2**1
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .eh_frame     00000004  00011000  00011000  00001000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .init_array   00000008  00011004  00011004  00001004  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  3 .fini_array   00000004  0001100c  0001100c  0000100c  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  4 .data         00000830  00011010  00011010  00001010  2**3
                  CONTENTS, ALLOC, LOAD, DATA
  5 .sdata        00000014  00011840  00011840  00001840  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  6 .sbss         00000014  00011854  00011854  00001854  2**2
                  ALLOC
  7 .bss          00000044  00011868  00011868  00001854  2**2
                  ALLOC
  8 .comment      00000030  00000000  00000000  00001854  2**0
                  CONTENTS, READONLY
  9 .riscv.attributes 00000026  00000000  00000000  00001884  2**0
                  CONTENTS, READONLY
```

Рис. 14 Секции файла a.out

В таблице секций по-прежнему можно заметить секцию .text, размер которой отличен от нуля, так как в исполняемом файле записан необходимый код. Однако в данном файле остальные секции не являются пустыми, в отличие от объектных файлов, что может быть объяснено необходимостью использования каждой из секций в законченной исполняемой программе.

```
00010112 <main>:
   10112:   1141                    c.addi  sp,-16
   10114:   c606                    c.swsp  ra,12(sp)
   10116:   c422                    c.swsp  s0,8(sp)
   10118:   4591                    c.li    a1,4
   1011a:   452d                    c.li    a0,11
   1011c:   2885                    c.jal   1018c <calloc>
   1011e:   842a                    c.mv    s0,a0
   10120:   872a                    c.mv    a4,a0
   10122:   4781                    c.li    a5,0
   10124:   46ad                    c.li    a3,11
   10126:   c31c                    c.sw    a5,0(a4)
   10128:   0785                    c.addi  a5,1
   1012a:   0711                    c.addi  a4,4
   1012c:   fed79de3            bne a5,a3,10126 <main+0x14>
   10130:   4685                    c.li    a3,1
   10132:   460d                    c.li    a2,3
   10134:   45ad                    c.li    a1,11
   10136:   8522                    c.mv    a0,s0
   10138:   2801                    c.jal   10148 <reverse>
   1013a:   8522                    c.mv    a0,s0
   1013c:   2a41                    c.jal   102cc <free>
   1013e:   4501                    c.li    a0,0
   10140:   40b2                    c.lwsp  ra,12(sp)
   10142:   4422                    c.lwsp  s0,8(sp)
   10144:   0141                    c.addi  sp,16
   10146:   8082                    c.jr    ra

00010148 <reverse>:
   10148:   4785                    c.li    a5,1
   1014a:   04b7d063            bge a5,a1,1018a <reverse+0x42>
   1014e:   02b668b3            rem a7,a2,a1
   10152:   c299                    c.beqz  a3,10158 <reverse+0x10>
   10154:   411588b3            sub a7,a1,a7
   10158:   02b8e7b3            rem a5,a7,a1
   1015c:   00052803            lw  a6,0(a0)
   10160:   00279713            slli    a4,a5,0x2
   10164:   972a                    c.add   a4,a0
   10166:   4310                    c.lw    a2,0(a4)
   10168:   4681                    c.li    a3,0
   1016a:   00279713            slli    a4,a5,0x2
   1016e:   972a                    c.add   a4,a0
   10170:   01072023            sw  a6,0(a4)
   10174:   97c6                    c.add   a5,a7
   10176:   02b7e7b3            rem a5,a5,a1
   1017a:   8832                    c.mv    a6,a2
   1017c:   00279713            slli    a4,a5,0x2
   10180:   972a                    c.add   a4,a0
   10182:   4310                    c.lw    a2,0(a4)
   10184:   0685                    c.addi  a3,1
   10186:   fed592e3            bne a1,a3,1016a <reverse+0x22>
   1018a:   8082                    c.jr    ra
```

Рис. 15 Функции main и reverse внутри дизассемблированного файла a.out

Анализ кодов функций main и reverse, находящихся в дизассемблированном файле a.out показывает, что компоновщик, как это и ожидалось, не изменил команды этих функций, за исключением адресов вызовов команд и различных переходов.

Создание библиотеки с использованием make

На языке программы make был разработан makefile, который приводится на рис. 16. Выполнение данного файла программой make приводит к созданию двух целевых файлов executable и reverse_lib.a и двух промежуточных объектных файлов main.o и reverse.o.

```
1
2   .PHONY: all clean clean_trash
3
4   LIBNAME = reverse_lib.a
5
6   EXEC = executable
7
8   OBJS = main.o \
9       reverse.o
10
11  REG_OBJS = main.o
12
13  LIB_OBJS = reverse.o
14
15  all: $(LIBNAME) $(EXEC)
16
17  clean:
18      $(RM) -f $(LIBNAME)
19      $(RM) -f $(OBJS)
20      $(RM) -f $(EXEC)
21      $(RM) -f $(OBJS:.o=.s)
22
23  clean_trash:
24      $(RM) -f $(OBJS)
25      $(RM) -f $(OBJS:.o=.i)
26      $(RM) -f $(OBJS:.o=.s)
27
28  vpath %.h include
29  vpath %.c src
30
31  $(OBJS): $(OBJS:.o=.c)
32      riscv64-unknown-elf-gcc -c -march=rv32imc -mabi=ilp32 --sysroot=include -B src -O1 $^
33
34  $(LIBNAME): $(OBJS)
35      riscv64-unknown-elf-ar -rsc $@ $(LIB_OBJS)
36
37  $(EXEC): $(LIBNAME) $(REG_OBJS)
38      riscv64-unknown-elf-gcc -march=rv32imc -mabi=ilp32 -O1 $(REG_OBJS) $(LIBNAME) -o $(EXEC)
39
```

Рис. 16 Содержимое файла makefile

Анализ объектных файлов main.o и reverse.o показал, что содержимое данных файлов не отличается от соответствующих файлов, рассмотренных в предыдущей части отчета.

Статическая библиотека reverse_lib.a формируется из одного файла reverse.o., что и является причиной того, что эти файлы никак не отличаются по своему содержанию.

На основе статической библиотеки reverse_lib.a и объектного файла main.o компонуется конечный исполняемый файл executable, который и является целью сборки.

Анализ контрольных сумм файлов executable и a.out, полученного при ручной раздельной компиляции показывает, что эти файлы также идентичны(рис. 17).

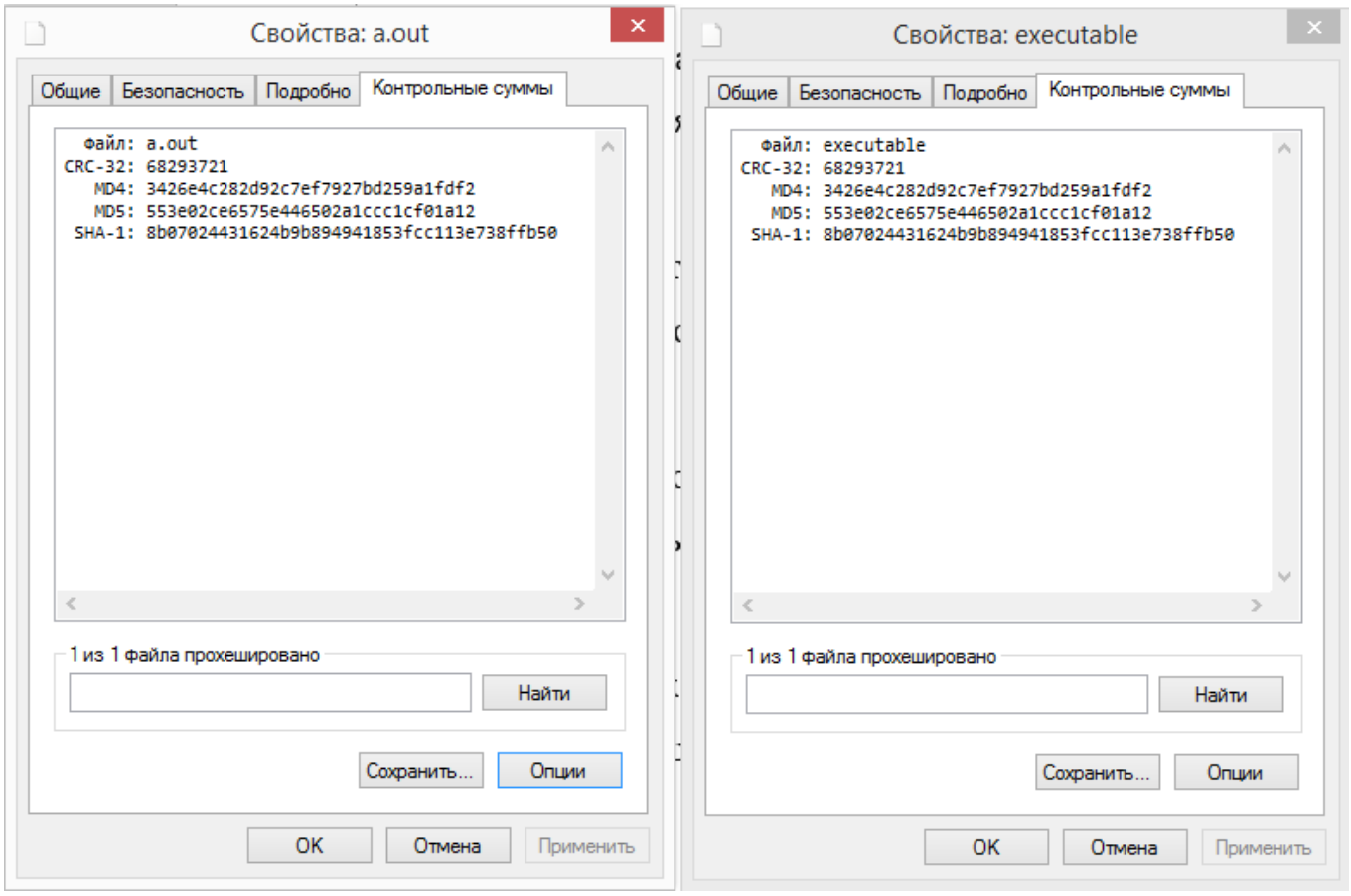

Рис. 17 Контрольные суммы файлов a.out и executable

Вывод

В ходе работы была проведена раздельная компиляция созданных исходных файлов программы. Результат каждого этапа компиляции был изучен и проанализирован. Ожидаемое содержимое объектных файлов, файлов с кодом языка ассемблера, препроцессированных файлов и исполняемых совпало с содержимым полученных файлов. Кроме того, процесс компиляции был успешно произведен с использованием программы make, для которой был написан соответствующий makefile.