

Отчет по лабораторной работе №4

Дисциплина: Низкоуровневое программирование

Тема: Раздельная компиляция

Работу выполнил: Чевычелов А. А.

Группа: 3530901/10003

Преподаватель: Коренев Д. А.

1. 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 9: Реверс массива целых чисел.

2. Метод решения

Тестовая программа (*main.c*) передает в функцию (*reverse*), находящуюся в отдельном файле (*reverse.c*), массив и его длину. Функция (*reverse*) сначала записывает первый операнд в «буфер», затем первый операнд перезаписывается вторым и далее второй значением из буфера.

3. Программа на языке «C»

3.1. Текст программ, реализующих определенную вариант задания функциональность

Программа состоит из трех файлов: *reverse.h* (файл-заголовок), *main.c* (с тестовой программой), *reverse.c* (с основной программой).

Заголовочный файл *reverse.h* содержит в себе определение функции *reverse*. В дальнейшем для использования этой функции в другой программе необходимо организовывать подключение этого заголовочного файла.

Такая конструкция файла *reverse.h* предотвращает двойное подключение заголовочного файла путём проверки существования этого макроса, который имеет то же самое имя, что и заголовочный файл. Определение макроса *REVERSE_H* происходит, когда заголовочный файл впервые обрабатывается препроцессором. Затем, если этот заголовочный файл вновь подключается,

REVERSE_H уже определен, в результате чего препроцессор пропускает полностью текст этого заголовочного файла.

```
1 #ifndef REVERSE_H //проверка существования макроса
2 #define REVERSE_H //создание макроса
3 void reverse(int *arr, int size);
4 #endif
```

Рисунок 1. reverse.h

```
1 #include "reverse.h"
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 int main() {
6     int arr[7] = {0, 1, 2, 3, 4, 5, 6}; //Входной массив
7     int size = 7; //Размер
8     reverse(arr, size); //Вызов функции
9     for (int i = 0; i < size - 1; i++) { //Печать
10         printf("%d, ", arr[i]); //Вывод
11     }
12     printf("%d", arr[size - 1]); //Вывод
13     return 0;
14 }
```

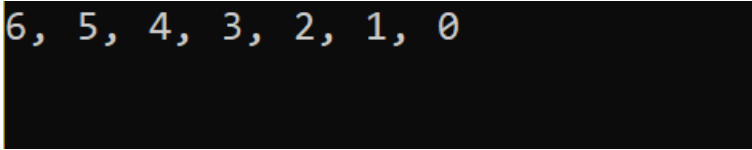
Рисунок 2. main.c

Основная функция (reverse) работает по следующему принципу. На вход подается массив (на деле – указатель на массив, т.к массив нельзя передать как аргумент в функцию) и его размер. Цикл проходит по $size/2$ элементам, для получения первого операнда, а второй операнд вычисляется в цикле. Затем они с помощью буфера меняются местами. (рис. 3)

```
1 #include "reverse.h"
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 //Функция реверса массива чисел
6
7 void reverse(int *arr, int size) {
8
9     for (int i = 0; i <= (size/2); i++) {
10         int buff = arr[i]; //Сохранение во временную ячейку(буфер)
11         int n = size - (i + 1); //(Противоположный эл-т)
12         arr[i] = arr[n]; //Замена
13         arr[n] = buff; //Замена
14     }
15 }
```

Рисунок 2. reverse.c

Результат работы программы верен.

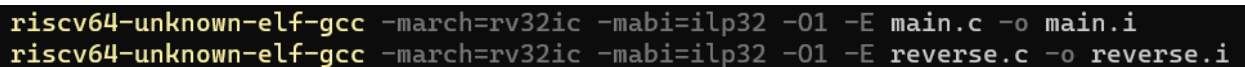


```
6, 5, 4, 3, 2, 1, 0
```

Рисунок 3. Результат работы программы

3.2. Сборка программ «по шагам», анализ промежуточных и результирующих файлов

Препроцессирование выполняется следующими командами:



```
riscv64-unknown-elf-gcc -march=rv32ic -mabi=ilp32 -O1 -E main.c -o main.i  
riscv64-unknown-elf-gcc -march=rv32ic -mabi=ilp32 -O1 -E reverse.c -o reverse.i
```

Драйвер компилятора gcc – riscv64-unknown-elf-gcc запускается с параметрами командной строки «-march=rv32i» и «-mabi=ilp32», указывающих что целевым является процессор с базовой архитектурой системы команд RV32I, а «-O1» – указание выполнять простые оптимизации генерируемого кода; «-E» – указание остановить процесс сборки после препроцессирования.

Результаты содержатся в файлах *main.i* и *reverse.i* (фрагменты):

- **main.i**

```
# 1 "main.c"  
# 1 "<built-in>"  
# 1 "<command-line>"  
# 1 "main.c"  
# 1 "reverse.h" 1  
void reverse(int *arr, int size);  
  
# 2 "main.c" 2  
- - - - -  
# 4 "main.c" 2  
  
# 5 "main.c"  
int main() {  
    int arr[7] = {0, 1, 2, 3, 4, 5, 6};  
    int size = 7;  
    reverse(arr, size);  
    for (int i = 0; i < size - 1; i++) {  
        printf("%d, ", arr[i]);  
    }  
}
```

```

        printf("%d", arr[size - 1]);
        scanf("1");
        return 0;
    }

    • reverse.i
# 1 "reverse.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "reverse.c"
# 1 "reverse.h" 1
void reverse(int *arr, int size);

# 2 "reverse.c" 2
- - - - -
# 4 "reverse.c" 2

# 7 "reverse.c"
void reverse(int *arr, int size) {

    for (int i = 0; i <= (size/2); i++) {
        int buff = arr[i];
        int n = size - (i + 1);
        arr[i] = arr[n];
        arr[n] = buff;
    }
}

```

Символ «#» используются для передачи информации об исходном тексте из препроцессора в компилятор. Препроцессор позволяет включать в текст программы файлы. Так, в файле *reverse.c* первая строка «*#include "reverse.h"*» и результат её обработки представлен на 4-6 строках (вставка содержимого заголовочного файла). Представлены фрагменты кода, так как из-за включения заголовочных файлов функций ввода-вывода (*stdio.h*) и стандартной библиотеки языка C (*stdlib.h*) файлы оказались слишком длинными. Исходный код тестирующей функции *main* после работы препроцессора остался без изменений, как и исходный код функции *reverse*.

3.3. Компиляция

Компиляция в файлы *main.s* и *reverse.s* осуществляется следующими командами:

```
riscv64-unknown-elf-gcc -march=rv32ic -mabi=ilp32 -O1 -S -fpreprocessed reverse.i -o reverse.s  
riscv64-unknown-elf-gcc -march=rv32ic -mabi=ilp32 -O1 -S -fpreprocessed main.i -o main.s
```

Получившийся код на языке ассемблера содержится в файлах *reverse.s* и *main.s*.

- **main.s**

```
.file      "main.c"  
.option nopic  
.attribute arch, "rv32i2p0_c2p0"  
.attribute unaligned_access, 0  
.attribute stack_align, 16  
.text  
.align     1  
.globl     main  
.type      main, @function  
main:  
    addi sp,sp,-48  
    sw   ra,44(sp)  
    sw   s0,40(sp)  
    sw   s1,36(sp)  
    sw   s2,32(sp)  
    lui  a5,%hi(.LANCHOR0)  
    addi a5,a5,%lo(.LANCHOR0)  
    lw   a6,0(a5)  
    lw   a0,4(a5)  
    lw   a1,8(a5)  
    lw   a2,12(a5)  
    lw   a3,16(a5)  
    lw   a4,20(a5)  
    lw   a5,24(a5)  
    sw   a6,4(sp)  
    sw   a0,8(sp)  
    sw   a1,12(sp)  
    sw   a2,16(sp)  
    sw   a3,20(sp)  
    sw   a4,24(sp)  
    sw   a5,28(sp)  
    li   a1,7  
    addi a0,sp,4  
    call reverse  
    addi s0,sp,4  
    addi s2,sp,28  
    lui  s1,%hi(.LC1)  
.L2:
```

```

    lw    a1,0(s0)
    addi a0,s1,%lo(.LC1)
    call printf
    addi s0,s0,4
    bne  s0,s2,.L2
    lw    a1,28(sp)
    lui   a0,%hi(.LC2)
    addi a0,a0,%lo(.LC2)
    call printf
    lui   a0,%hi(.LC3)
    addi a0,a0,%lo(.LC3)
    call scanf
    li    a0,0
    lw    ra,44(sp)
    lw    s0,40(sp)
    lw    s1,36(sp)
    lw    s2,32(sp)
    addi sp,sp,48
    jr    ra
.size    main, .-main
.section .rodata
.align   2
.set .LANCHOR0,. + 0
.LC0:
    .word    0
    .word    1
    .word    2
    .word    3
    .word    4
    .word    5
    .word    6
    .section .rodata.str1.4,"aMS",@progbits,1
    .align   2
.LC1:
    .string  "%d, "
    .zero    3
.LC2:
    .string  "%d"
    .zero    1
.LC3:
    .string  "1"
    .ident   "GCC: (SiFive GCC 8.3.0-2020.04.1) 8.3.0"

```


- **reverse.s**

```
.file      "reverse.c"
.option nopic
.attribute arch, "rv32i2p0_c2p0"
.attribute unaligned_access, 0
.attribute stack_align, 16
.text
.align    1
.globl    reverse
.type     reverse, @function
reverse:
    srli a6,a1,31
    add  a6,a6,a1
    srai a6,a6,1
    li   a5,-1
    blt  a1,a5,.L1
    mv   a5,a0
    slli a1,a1,2
    add  a0,a0,a1
    li   a4,0
.L3:
    lw   a3,0(a5)
    addi a4,a4,1
    lw   a2,-4(a0)
    sw   a2,0(a5)
    sw   a3,-4(a0)
    addi a5,a5,4
    addi a0,a0,-4
    ble  a4,a6,.L3
.L1:
    ret
.size   reverse, .-reverse
.ident  "GCC: (SiFive GCC 8.3.0-2020.04.1) 8.3.0"
```

3.4. Ассемблирование

Ассемблирование файлов *main.s* и *reverse.s* в *main.o* и *reverse.o* соответственно выполняется по следующей команде:

```
riscv64-unknown-elf-gcc -march=rv32ic -mabi=ilp32 -O1 -c main.s -o main.o
riscv64-unknown-elf-gcc -march=rv32ic -mabi=ilp32 -O1 -c reverse.s -o reverse.o
```

Здесь «-с» – указание остановить процесс сборки после ассемблирования.

Объектный файл не является текстовым. Чтобы посмотреть содержимое *main.o* необходимо вызвать команду:

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

Результат её выполнения:

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

Sections:
Idx Name          Size      VMA           LMA           File off  Algn
  0 .text          0000008a  00000000  00000000  00000034  2**1
    CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data          00000000  00000000  00000000  000000be  2**0
    CONTENTS, ALLOC, LOAD, DATA
  2 .bss           00000000  00000000  00000000  000000be  2**0
    ALLOC
  3 .rodata        0000001c  00000000  00000000  000000c0  2**2
    CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .rodata.str1.4 0000000e  00000000  00000000  000000dc  2**2
    CONTENTS, ALLOC, LOAD, READONLY, DATA
  5 .comment        00000029  00000000  00000000  000000ea  2**0
    CONTENTS, READONLY
  6 .riscv.attributes 00000021  00000000  00000000  00000113  2**0
    CONTENTS, READONLY
```

Проведем аналогичную операцию с файлом *reverse.o*:

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

Sections:
Idx Name          Size      VMA           LMA           File off  Algn
  0 .text          00000030  00000000  00000000  00000034  2**1
    CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data          00000000  00000000  00000000  00000064  2**0
    CONTENTS, ALLOC, LOAD, DATA
  2 .bss           00000000  00000000  00000000  00000064  2**0
    ALLOC
  3 .comment        00000029  00000000  00000000  00000064  2**0
    CONTENTS, READONLY
  4 .riscv.attributes 00000021  00000000  00000000  0000008d  2**0
    CONTENTS, READONLY
```

В файлах *main.o* и *reverse.o* имеются следующие секции:

- .text – секция кода, в которой содержатся коды инструкций (название секции обусловлено историческими причинами);
- .data – секция инициализированных данных;
- .bss – секция неинициализированных статических переменных;
- .rodata – аналог .data для неизменяемых данных;
- .comment – секция данных о версиях размером;

`.riscv.attributes` – информация про RISC-V;

Секции `data` объектных файлов – секции инициализированных данных – не содержат данных, размер секций равен нулю.

Секции `bss` объектных файлов – секции данных, инициализированных нулями – таким же образом пусты.

Секция `comment` – секция данных о версиях – и для одного, и для другого файла содержит одни и те же значения – сведения о GCC версии.

Секция `riscv.attributes` обоих объектных файлов содержит одну и ту же информацию об используемой архитектуре команд RV32I.

Изучим таблицы файлов `main.c` и `reverse.c`:

Вызов таблицы для файла `main.o`:

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

Таблица:

```
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      d  .rodata 00000000 .rodata
00000000 l      d  .rodata 00000000 .LANCHOR0
00000000 l      d  .rodata.str1.4 00000000 .rodata.str1.4
00000000 l      d  .rodata.str1.4 00000000 .LC1
00000008 l      d  .rodata.str1.4 00000000 .LC2
0000000c l      d  .rodata.str1.4 00000000 .LC3
00000046 l      d  .text 00000000 .L2
00000000 l      d  .comment 00000000 .comment
00000000 l      d  .riscv.attributes 00000000 .riscv.attributes
00000000 g      F  .text 0000008a main
00000000      *UND* 00000000 reverse
00000000      *UND* 00000000 printf
00000000      *UND* 00000000 scanf
```

Аналогично вызовем таблицу для файла `reverse.o`:

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

SYMBOL TABLE:
00000000 l      df *ABS*  00000000 reverse.c
00000000 l      d  .text 00000000 .text
00000000 l      d  .data 00000000 .data
00000000 l      d  .bss 00000000 .bss
0000002e l      d  .text 00000000 .L1
00000018 l      d  .text 00000000 .L3
00000000 l      d  .comment 00000000 .comment
00000000 l      d  .riscv.attributes 00000000 .riscv.attributes
00000000 g      F  .text 00000030 reverse
```

В каждой таблице только один глобальный (флаг “g”) символ типа «функция» (“F”) – *main* и *reverse* соответственно. (Остальное не обнаружилось и заполнено 0).

Проанализируем секции .text объектных файлов – секций кода, в которых содержатся коды инструкций:

Вызов:

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

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

```
Disassembly of section .text:
```

```
00000000 <main>:
```

0:	7179	c.addi16sp	sp, -48
2:	d606	c.swsp	ra, 44(sp)
4:	d422	c.swsp	s0, 40(sp)
6:	d226	c.swsp	s1, 36(sp)
8:	d04a	c.swsp	s2, 32(sp)
a:	000007b7	lui	a5, 0x0
e:	00078793	addi	a5, a5, 0 # 0 <main>
12:	0007a803	lw	a6, 0(a5)
16:	43c8	c.lw	a0, 4(a5)
18:	478c	c.lw	a1, 8(a5)
1a:	47d0	c.lw	a2, 12(a5)
1c:	4b94	c.lw	a3, 16(a5)
1e:	4bd8	c.lw	a4, 20(a5)
20:	4f9c	c.lw	a5, 24(a5)
22:	c242	c.swsp	a6, 4(sp)
24:	c42a	c.swsp	a0, 8(sp)
26:	c62e	c.swsp	a1, 12(sp)
28:	c832	c.swsp	a2, 16(sp)
2a:	ca36	c.swsp	a3, 20(sp)
2c:	cc3a	c.swsp	a4, 24(sp)
2e:	ce3e	c.swsp	a5, 28(sp)
30:	459d	c.li	a1, 7
32:	0048	c.addi4spn	a0, sp, 4
34:	00000097	auipc	ra, 0x0
38:	000080e7	jalr	ra, 0(ra) # 34 <main+0x34>
3c:	0040	c.addi4spn	s0, sp, 4
3e:	01c10913	addi	s2, sp, 28
42:	000004b7	lui	s1, 0x0

```
00000046 <.L2>:
```

46:	400c	c.lw	a1, 0(s0)
48:	00048513	addi	a0, s1, 0 # 0 <main>
4c:	00000097	auipc	ra, 0x0
50:	000080e7	jalr	ra, 0(ra) # 4c <.L2+0x6>

```

54: 0411          c.addi    s0,4
56: ff2418e3     bne      s0,s2,46 <.L2>
5a: 45f2          c.lwsp    a1,28(sp)
5c: 00000537      lui      a0,0x0
60: 00050513      addi     a0,a0,0 # 0 <main>
64: 00000097      auipc    ra,0x0
68: 000080e7      jalr     ra,0(ra) # 64 <.L2+0x1e>
6c: 00000537      lui      a0,0x0
70: 00050513      addi     a0,a0,0 # 0 <main>
74: 00000097      auipc    ra,0x0
78: 000080e7      jalr     ra,0(ra) # 74 <.L2+0x2e>
7c: 4501          c.li     a0,0
7e: 50b2          c.lwsp    ra,44(sp)
80: 5422          c.lwsp    s0,40(sp)
82: 5492          c.lwsp    s1,36(sp)
84: 5902          c.lwsp    s2,32(sp)
86: 6145          c.addi16sp      sp,48
88: 8082          c.jr     ra

```

Вызов:

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

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

Disassembly of section .text:

00000000 <reverse>:

```

0: 01f5d813      srli     a6,a1,0x1f
4: 982e          c.add    a6,a1
6: 40185813      srai     a6,a6,0x1
a: 57fd          c.li     a5,-1
c: 02f5c163      blt      a1,a5,2e <.L1>
10: 87aa          c.mv     a5,a0
12: 058a          c.slli   a1,0x2
14: 952e          c.add    a0,a1
16: 4701          c.li     a4,0

```

00000018 <.L3>:

```

18: 4394          c.lw     a3,0(a5)
1a: 0705          c.addi   a4,1
1c: ffc52603      lw       a2,-4(a0)
20: c390          c.sw     a2,0(a5)
22: fed52e23      sw       a3,-4(a0)
26: 0791          c.addi   a5,4
28: 1571          c.addi   a0,-4
2a: fee857e3      bge      a6,a4,18 <.L3>

```

0000002e <.L1>:

```

2e: 8082          c.jr     ra

```

Дизассемблированный код (текст программы на языке ассемблера) практически идентичен сгенерированному (за исключением псевдоинструкций).

Аналогично проверим другие секции.

Секции `.data` не содержат данных:

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

Секции `.bss` не содержат данных:

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

Секции `.comment` идентичны для обоих файлов и как указывалось выше содержат информацию о версии GCC от SiFive:

```
Contents of section .comment:
0000 00474343 3a202853 69466976 65204743  .GCC: (SiFive GC
0010 4320382e 332e302d 32303230 2e30342e  C 8.3.0-2020.04.
0020 31292038 2e332e30 00                1) 8.3.0.
```

Секции также идентичны и содержат информацию об используемой архитектуре команд RV32I:

```
Contents of section .riscv.attributes:
0000 41200000 00726973 63760001 16000000  A ...riscv.....
0010 04100572 76333269 3270305f 63327030  ...rv32i2p0_c2p0
0020 00                .
```

Далее проанализируем таблицы перемещений:

```
riscv64-unknown-elf-objdump.exe -r main.o reverse.o
```

Для файла `reverse.o`:

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

RELOCATION RECORDS FOR [.text]:
OFFSET      TYPE          VALUE
0000000c    R_RISCV_BRANCH      .L1
0000002a    R_RISCV_BRANCH      .L3
```

И для файла `main.o`:

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

RELOCATION RECORDS FOR [.text]:
OFFSET      TYPE          VALUE
0000000a R_RISCV_HI20      .LANCHOR0
0000000a R_RISCV_RELAX      *ABS*
0000000e R_RISCV_L012_I     .LANCHOR0
0000000e R_RISCV_RELAX      *ABS*
00000034 R_RISCV_CALL       reverse
00000034 R_RISCV_RELAX      *ABS*
00000042 R_RISCV_HI20      .LC1
00000042 R_RISCV_RELAX      *ABS*
00000048 R_RISCV_L012_I     .LC1
00000048 R_RISCV_RELAX      *ABS*
0000004c R_RISCV_CALL       printf
0000004c R_RISCV_RELAX      *ABS*
0000005c R_RISCV_HI20      .LC2
0000005c R_RISCV_RELAX      *ABS*
00000060 R_RISCV_L012_I     .LC2
00000060 R_RISCV_RELAX      *ABS*
00000064 R_RISCV_CALL       printf
00000064 R_RISCV_RELAX      *ABS*
0000006c R_RISCV_HI20      .LC3
0000006c R_RISCV_RELAX      *ABS*
00000070 R_RISCV_L012_I     .LC3
00000070 R_RISCV_RELAX      *ABS*
00000074 R_RISCV_CALL       scanf
00000074 R_RISCV_RELAX      *ABS*
00000056 R_RISCV_BRANCH     .L2
```

В таблицах перемещений содержится информация о необходимых заменах адресов, которая будет передана компоновщику. Видно, что в файле *main.o* есть информация о необходимости замены для всех внешних функций, а в файле *reverse.o* содержится информация о необходимости подстановки адресов для возвратов в циклах.

3.5. Компоновка

Компоновка:

```
riscv64-unknown-elf-gcc -march=rv32ic -mabi=ilp32 main.o reverse.o -o main
```

Изучим содержимое секции *.text* полученного в результате компоновки программы исполняемого файла, для этого получим файл *main.ds*:

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

Нас интересуют некоторые секции этого файла:

```

- - - - -
00010144 <main>:
    10144: 7179          c.addi16sp    sp,-48
    10146: d606          c.swsp       ra,44(sp)
    10148: d422          c.swsp       s0,40(sp)
    1014a: d226          c.swsp       s1,36(sp)
    1014c: d04a          c.swsp       s2,32(sp)
    1014e: 0002f7b7     lui    a5,0x2f
    10152: eb878793     addi   a5,a5,-328      #      2eeb8
<__clzsi2+0x4c>
    10156: 0007a803     lw      a6,0(a5)
    1015a: 43c8          c.lw   a0,4(a5)
    1015c: 478c          c.lw   a1,8(a5)
    1015e: 47d0          c.lw   a2,12(a5)
    10160: 4b94          c.lw   a3,16(a5)
    10162: 4bd8          c.lw   a4,20(a5)
    10164: 4f9c          c.lw   a5,24(a5)
    10166: c242          c.swsp   a6,4(sp)
    10168: c42a          c.swsp   a0,8(sp)
    1016a: c62e          c.swsp   a1,12(sp)
    1016c: c832          c.swsp   a2,16(sp)
    1016e: ca36          c.swsp   a3,20(sp)
    10170: cc3a          c.swsp   a4,24(sp)
    10172: ce3e          c.swsp   a5,28(sp)
    10174: 459d          c.li    a1,7
    10176: 0048          c.addi4spn  a0,sp,4
    10178: 283d          c.jal    101b6 <reverse>
    1017a: 0040          c.addi4spn  s0,sp,4
    1017c: 01c10913     addi   s2,sp,28
    10180: 0002f4b7     lui    s1,0x2f
    10184: 400c          c.lw   a1,0(s0)
    10186: ed448513     addi   a0,s1,-300      #      2eed4
<__clzsi2+0x68>
    1018a: 2c6d          c.jal    10444 <printf>
    1018c: 0411          c.addi   s0,4
    1018e: ff241be3     bne    s0,s2,10184 <main+0x40>
    10192: 45f2          c.lwsp   a1,28(sp)
    10194: 0002f537     lui    a0,0x2f
    10198: edc50513     addi   a0,a0,-292      #      2eedc
<__clzsi2+0x70>
    1019c: 2465          c.jal    10444 <printf>
    1019e: 0002f537     lui    a0,0x2f

```



```

    101a2: ee050513      addi a0,a0,-288      #      2eee0
<__clzsi2+0x74>
    101a6: 2ccd             c.jal      10498 <scanf>
    101a8: 4501             c.li a0,0
    101aa: 50b2             c.lwsp     ra,44(sp)
    101ac: 5422             c.lwsp     s0,40(sp)
    101ae: 5492             c.lwsp     s1,36(sp)
    101b0: 5902             c.lwsp     s2,32(sp)
    101b2: 6145             c.addi16sp sp,48
    101b4: 8082             c.jr ra

000101b6 <reverse>:
    101b6: 01f5d813      srli a6,a1,0x1f
    101ba: 982e             c.add      a6,a1
    101bc: 40185813      srai a6,a6,0x1
    101c0: 57fd             c.li a5,-1
    101c2: 02f5c163      blt a1,a5,101e4 <reverse+0x2e>
    101c6: 87aa             c.mv a5,a0
    101c8: 058a             c.slli     a1,0x2
    101ca: 952e             c.add      a0,a1
    101cc: 4701             c.li a4,0
    101ce: 4394             c.lw a3,0(a5)
    101d0: 0705             c.addi     a4,1
    101d2: ffc52603      lw a2,-4(a0)
    101d6: c390             c.sw a2,0(a5)
    101d8: fed52e23      sw a3,-4(a0)
    101dc: 0791             c.addi     a5,4
    101de: 1571             c.addi     a0,-4
    101e0: fee857e3      bge a6,a4,101ce <reverse+0x18>
    101e4: 8082             c.jr ra
- - - - -

```

Компоновщик все переходы `auipc+jalr`, заменил на одну инструкцию `jal` и корректным адресом перехода. Например, строка

```
10178: 283d      c.jal      101b6 <reverse>
```

дает переход на

```
101b6: 01f5d813      srli a6,a1,0x1f
```

Проанализируем таблицу перемещений полученного файла (*main*):

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

Таблица перемещений пуста, т. е. все необходимые релокации, оптимизации и замены инструкций были успешно проведены компоновщиком.

3.6 Формирование статической библиотеки

Статическая библиотека, по сути, является архивом (набором, коллекцией) объектных файлов, оперируемых компоновщиком.

Поместим объектный файл *reverse.o* в статическую библиотеку *lib*:

```
riscv64-unknown-elf-ar -rsc lib.a reverse.o
```

```
riscv64-unknown-elf-ar -t lib.a reverse.o
```

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

```
reverse.o:  
0000002e t .L1  
00000018 t .L3  
00000000 T reverse
```

В выводе утилиты *-nm* кодом *T* обозначаются символы, определенные в соответствующем объектном файле. Символ функции *reverse* является основным символом, определяемым в этом объектном файле, остальные символы определяют локальные метки для этого файла.

Посмотрим таблицу символов исполняемого файла:

```
riscv64-unknown-elf-objdump.exe -t a.out
```

```
00000000 l      df *ABS*  00000000 main.c  
00000000 l      df *ABS*  00000000 reverse.c  
00010144 g      F .text  00000072 main  
000101b6 g      F .text  00000030 reverse
```

Можно заметить, что в состав программы вошло содержимое объектных файлов *reverse.o* и *main.o*.

Теперь создадим make-файлы (makefile).

Makefile-1.txt

```
lib.a: reverse.o reverse.h  
    riscv64-unknown-elf-ar -rsc lib.a reverse.o  
  
reverse.o: reverse.c  
    riscv64-unknown-elf-gcc.exe -march=rv32i -mabi=ilp32 -O1 -c  
reverse.c -o reverse.o
```

Makefile-2.txt

```
all:  
    mingw32-make -f Makefile-1.txt  
    riscv64-unknown-elf-gcc -march=rv32ic -mabi=ilp32 -O1 -save-  
temps main.c lib.a -o a.out  
    del *.o, *.i, *.s
```

Запуск make-файла *Makefile-2.txt*:

```
mingw32-make -f Makefile-2.txt
```

```
mingw32-make -f Makefile-1.txt
mingw32-make[1]: Entering directory 'C:/Users/sasha/Projects/lab_04'
riscv64-unknown-elf-gcc.exe -march=rv32i -mabi=ilp32 -O1 -c reverse.c -o reverse.o
riscv64-unknown-elf-ar -rsc lib.a reverse.o
mingw32-make[1]: Leaving directory 'C:/Users/sasha/Projects/lab_04'
riscv64-unknown-elf-gcc -march=rv32ic -mabi=ilp32 -O1 -save-temps main.c lib.a -o a.out
del *.o, *.i, *.s
```

4. Вывод

В ходе выполнения лабораторной работы были получены навыки в сфере языка C, ассемблера RISC-V, получены навыки работы с препроцессором, компилятором, ассемблером и компоновщиком пакета GCC и драйвером компилятора riscv64-unknown-elf-gcc.

Была реализована поставленная задача – «реверс массива чисел».

Была собрана программа «по шагам». Были изучены особенности каждого этапа пошаговой сборки набора программ, а также инструменты, позволяющие выделить разработанные программы в статическую библиотеку и автоматизировать сборку этой библиотеки с помощью make-файлов.