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

Отчёт по лабораторной работе № 4

Дисциплина: Низкоуровневое программирование

Тема: раздельная компиляция

Вариант: 17

Выполнил студент гр. 3530901/90002 _____А. И. Юрченко
(подпись)

Принял старший преподаватель _____Д. С. Степанов
(подпись)

“_” _____ 2021 г.

Санкт-Петербург
2021

Формулировка задачи

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

Вариант задания

По варианту номер 17 необходимо реализовать реверс массива. Алгоритм состоит из одного прохода по массиву. За этот проход элемент с индексом [0] меняется местами с элементом с индексом [n-1], элемент с индексом [1] меняется местами с элементом с индексом [n-2] и так далее. Где n – длина массива.

1. Написание программы на языке C

Согласно заданию, была написана программа, выполняющая реверс массива. Функция помещена в отдельный файл.

Листинг 1.1 Функция реверса reverse.c

```c
#include <stdio.h>
#include "reverse.h"

void reverse(unsigned *array, size_t size)
{
    static unsigned temp = 0;

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

Оформлен заголовочный файл.

Листинг 1.2 Заголовочный файл reverse.h

```c
#ifndef REVERSE_H
#define REVERSE_H

void reverse(unsigned *array, size_t size);

#endif
```

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

Листинг 1.3 Файл тестовой программы main.c

```c
#include <stddef.h>
#include <stdio.h>
#include "reverse.h"

static unsigned array[] = {1, 5, 1000, 6, 245, 3, 0};
static const size_t array_length = sizeof(array) / sizeof(array[0]);

int main()
{
    printf("initial: \n");
    for (size_t i = 0; i < array_length; i++) // вывод изначального массива
    {
        printf("%u ", array[i]);
    }

    reverse(array, array_length);

    printf("\nresult: \n");
    for (size_t i = 0; i < array_length; i++) // вывод изменённого массива
    {
        printf("%u ", array[i]);
    }
}
```

Были импортированы стандартные библиотеки «stddef.h» и «stdio.h».

Первая требуется для определения типа size_t, вторая – для вывода в консоль.

Произведём компиляцию программы и посмотрим на результат исполнения:

```
C:\Users\Tuf gaming\Desktop\lab4\lib>main.exe
initial:
1 5 1000 6 245 3 0
result:
0 3 245 6 1000 5 1
```

Рис. 1.1 Вызов программы

Как и ожидалось, программа выполняет реверс изначального массива.

2. Сборка программы “по шагам”

Препроцессирование

Выполним сборку программы по шагам. Для выполнения отдельных шагов мы будем по-прежнему драйвер компилятора (а не обращаться к ассемблеру или компоновщику напрямую), и контролировать его действия.

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

```
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -O1 -v -E main.c -o main.i >log_pre_main.txt 2>&1
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -O1 -v -E reverse.c -o reverse.i >log_pre_reverse.txt 2>&1
```

Разберём параметры запуска.

-march=rv64iac -mabi=lp64	целевым является процессор с базовой архитектурой системы команд RV32I
-O1	выполнять простые оптимизации генерируемого кода
-v	печатать (в стандартный поток ошибок) выполняемые драйвером команды, а также дополнительную информацию.
>	Выводы печати в файлы
-o	Output file
-E	Выполнять обработку файлов только препроцессором
2>&1	поток вывода ошибок (2 – стандартный «номер» этого потока) «связывается» с поток вывода («номер» 1), т.е. сообщения об ошибках (и информация, вывод которой вызван использованием флага “-v”, см.выше) также выводятся в файл

Посмотрим на результаты препроцессирования в файлах main.i и reverse.i.

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

Листинг 2.1 Файл main.i (часть)

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

static unsigned array[] = {1, 5, 1000, 6, 245, 3, 0};
static const size_t array_length = sizeof(array) / sizeof(array[0]);

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

    reverse(array, array_length);

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

Листинг 2.2 Файл reverse.i (часть)

```
# 1 "reverse.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "reverse.c"
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# 4 "reverse.h"
void reverse(unsigned *array, size_t size);
# 3 "reverse.c" 2

void reverse(unsigned *array, size_t size)
{
   static unsigned temp = 0;

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

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

Компиляция

Для компиляции препроцессированных файлов используем следующие команды:

```
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -O1 -v -S -fpreprocessed main.i
-o main.s >log_comp_main.txt 2>&1
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -O1 -v -S -fpreprocessed
reverse.i -o reverse.s >log_comp_reverse.txt 2>&1
```

В результате получаем файлы main.s и reverse.s.

Листинг 2.3 Файл main.s

```
.file  "main.c"
        .option nopic
        .attribute arch, "rv64i2p0_a2p0_c2p0"
        .attribute unaligned_access, 0
        .attribute stack_align, 16
        .text
        .section        .rodata.str1.4,"aMS",@progbits,1
        .align 2
.LC0:
        .string"initial: "
        .align 2
.LC1:
        .string"%u "
        .align 2
.LC2:
        .string"\nresult: "
        .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    a0,%hi(.LC0)
        addi   a0,a0,%lo(.LC0)
        call   puts
        lui    s0,%hi(.LANCHOR0)
        addi   s1,s0,%lo(.LANCHOR0)
        addi   s2,s1,28
        addi   s0,s0,%lo(.LANCHOR0)
```

```
        lui     s3,%hi(.LC1)
.L2:
        lw      a1,0(s0)
        addi    a0,s3,%lo(.LC1)
        call    printf
        addi    s0,s0,4
        bne     s0,s2,.L2
        li      a1,7
        lui     a0,%hi(.LANCHOR0)
        addi    a0,a0,%lo(.LANCHOR0)
        call    reverse
        lui     a0,%hi(.LC2)
        addi    a0,a0,%lo(.LC2)
        call    puts
        lui     s0,%hi(.LC1)
.L3:
        lw      a1,0(s1)
        addi    a0,s0,%lo(.LC1)
        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, 28
array:
        .word 1
        .word 5
        .word 1000
        .word 6
        .word 245
        .word 3
        .word 0
        .ident "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"
```

Листинг 2.4 Файл reverse.s

```
        .file   "reverse.c"
        .option nopic
        .attribute arch, "rv64i2p0_a2p0_c2p0"
        .attribute unaligned_access, 0
        .attribute stack_align, 16
        .text
        .align 2
        .globl reverse
        .type  reverse, @function
reverse:
        addi   a6,a1,-1
        beq    a6,zero,.L1
        slli   a4,a1,2
        add    a4,a0,a4
        li     a5,0
.L3:
        lw     a3,-4(a4)
        lw     a2,0(a0)
        sw     a2,-4(a4)
        sw     a3,0(a0)
        addi   a5,a5,1
        beq    a5,a6,.L1
        addi   a4,a4,-4
        addi   a0,a0,4
        not    a3,a5
        add    a3,a3,a1
        bltu   a5,a3,.L3
.L1:
        ret
        .size  reverse, .-reverse
        .ident "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"
```

Наиболее интересные строчки кода выделены жёлтым цветом.

Мы видим, как реализуется цикл for через инструкции RISC-V. Также заметим, что тестовая программа действительно вызывает reverse через псевдоинструкцию call. Снизу видим метку на наш массив array. В файле reverse.s можно заметить, как программа меняет элементы местами.

Объектный файл

Выполним ассемблирование для получения объектных файлов программы.

```
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -v -c main.s -o main.o
>log_as_main.txt 2>&1
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -v -c reverse.s -o reverse.o
>log_as_reverse.txt 2>&1
```

На выходе получаем файлы “reverse.o” и “main.o”. В отличие от ранее рассмотренных файлов, объектные файлы не является текстовыми, для изучения их содержимого используем утилиту objdump, отображающую содержимое бинарных файлов в текстовом виде:

Листинг 2.5 Хедер файла main.o

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

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

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .text         0000008e  0000000000000000  0000000000000000  00000040  2**1
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         0000001c  0000000000000000  0000000000000000  000000d0  2**3
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  0000000000000000  0000000000000000  000000ec  2**0
                  ALLOC
  3 .rodata.str1.8 00000022  0000000000000000  0000000000000000  000000f0  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .comment      00000031  0000000000000000  0000000000000000  00000112  2**0
                  CONTENTS, READONLY
  5 .riscv.attributes 00000026  0000000000000000  0000000000000000  00000143  2**0
                  CONTENTS, READONLY
```

Листинг 2.6 Хедер файла reverse.o

riscv64-unknown-elf-objdump -h reverse.o

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

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .text         00000032  0000000000000000  0000000000000000  00000040  2**1
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  0000000000000000  0000000000000000  00000072  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  0000000000000000  0000000000000000  00000072  2**0
                  ALLOC
  3 .comment      00000031  0000000000000000  0000000000000000  00000072  2**0
                  CONTENTS, READONLY
  4 .riscv.attributes 00000026  0000000000000000  0000000000000000  000000a3  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 используется 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 elf64-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:   00000537            lui     a0,0x0
  1c:   00050513            addi    a0,a0,0 # 0 <main>
  20:   00000097            auipc   ra,0x0
  24:   000080e7            jalr    ra,0(ra) # 20 <main+0x20>
  28:   00000437            lui     s0,0x0
  2c:   00040493            addi    s1,s0,0 # 0 <main>
  30:   01c48913            addi    s2,s1,28
  34:   00040413            addi    s0,s0,0
  38:   000009b7            lui     s3,0x0

0000003c <.L2>:
  3c:   00042583            lw      a1,0(s0)
  40:   00098513            addi    a0,s3,0 # 0 <main>
  44:   00000097            auipc   ra,0x0
  48:   000080e7            jalr    ra,0(ra) # 44 <.L2+0x8>
  4c:   00440413            addi    s0,s0,4
  50:   ff2416e3            bne     s0,s2,3c <.L2>
  54:   00700593            addi    a1,zero,7
  58:   00000537            lui     a0,0x0
  5c:   00050513            addi    a0,a0,0 # 0 <main>
  60:   00000097            auipc   ra,0x0
  64:   000080e7            jalr    ra,0(ra) # 60 <.L2+0x24>
  68:   00000537            lui     a0,0x0
  6c:   00050513            addi    a0,a0,0 # 0 <main>
  70:   00000097            auipc   ra,0x0
  74:   000080e7            jalr    ra,0(ra) # 70 <.L2+0x34>
```

```
  78:   00000437            lui     s0,0x0

0000007c <.L3>:
  7c:   0004a583            lw      a1,0(s1)
  80:   00040513            addi    a0,s0,0 # 0 <main>
  84:   00000097            auipc   ra,0x0
  88:   000080e7            jalr    ra,0(ra) # 84 <.L3+0x8>
  8c:   00448493            addi    s1,s1,4
  90:   ff2496e3            bne     s1,s2,7c <.L3>
  94:   00000513            addi    a0,zero,0
  98:   01c12083            lw      ra,28(sp)
  9c:   01812403            lw      s0,24(sp)
  a0:   01412483            lw      s1,20(sp)
  a4:   01012903            lw      s2,16(sp)
  a8:   00c12983            lw      s3,12(sp)
  ac:   02010113            addi    sp,sp,32
  b0:   00008067            jalr    zero,0(ra)
```

Можно заметить псеводинстуркцию call, которая здесь записана комбинацей инструкций auipc + jalr. Также наблюдается выход из метода main.

Листинг 2.7 Содержание секции .comment

```
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                                   .
```

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

Рассмотрим таблицу символов:

Листинг 2.8 Таблица символов

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

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

SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 reverse.c
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000000 l    d  .data  0000000000000000 .data
0000000000000000 l    d  .bss   0000000000000000 .bss
0000000000000030 l      .text  0000000000000000 .L1
0000000000000010 l      .text  0000000000000000 .L3
0000000000000000 l    d  .comment       0000000000000000 .comment
0000000000000000 l    d  .riscv.attributes      0000000000000000 .riscv.attributes
0000000000000000 g     F .text  0000000000000032 reverse



main.o:     file format elf64-littleriscv

SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 main.c
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000000 l    d  .data  0000000000000000 .data
0000000000000000 l    d  .bss   0000000000000000 .bss
0000000000000000 l    d  .rodata.str1.8 0000000000000000 .rodata.str1.8
0000000000000000 l      .data  0000000000000000 .LANCHOR0
0000000000000000 l     O .data  000000000000001c array
0000000000000000 l      .rodata.str1.8 0000000000000000 .LC0
0000000000000010 l      .rodata.str1.8 0000000000000000 .LC1
0000000000000018 l      .rodata.str1.8 0000000000000000 .LC2
0000000000000030 l      .text  0000000000000000 .L2
000000000000006a l      .text  0000000000000000 .L3
0000000000000000 l    d  .comment       0000000000000000 .comment
0000000000000000 l    d  .riscv.attributes      0000000000000000 .riscv.attributes
0000000000000000 g     F .text  000000000000008e main
0000000000000000         *UND*  0000000000000000 puts
0000000000000000         *UND*  0000000000000000 printf
0000000000000000         *UND*  0000000000000000 reverse
```

В таблице символов “main.o” имеется интересная запись: символ “reverse” типа “*UND*” (undefined – не определен).Эта запись означает, что символ “reverse” 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 reverse.o main.o

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

RELOCATION RECORDS FOR [.text]:
OFFSET           TYPE                     VALUE
0000000000000004 R_RISCV_BRANCH    .L1
000000000000001e R_RISCV_BRANCH    .L1
000000000000002c R_RISCV_BRANCH    .L3



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     .LANCHOR0
000000000000001c R_RISCV_RELAX    *ABS*
0000000000000020 R_RISCV_LO12_I   .LANCHOR0
0000000000000020 R_RISCV_RELAX    *ABS*
0000000000000028 R_RISCV_LO12_I   .LANCHOR0
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     .LANCHOR0
0000000000000046 R_RISCV_RELAX    *ABS*
000000000000004a R_RISCV_LO12_I   .LANCHOR0
000000000000004a R_RISCV_RELAX    *ABS*
000000000000004e R_RISCV_CALL     reverse
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*
0000000000000040  R_RISCV_BRANCH   .L2
000000000000007a  R_RISCV_BRANCH   .L3
```

В таблице перемещений для main.o наблюдаем вызов метода reverse. 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=rv64iac -mabi=lp64 -v main.o reverse.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
```

```
0000000000010156 <main>:
   10156:   7179                c.addi16sp  sp,-48
   10158:   f406                c.sdsp  ra,40(sp)
   1015a:   f022                c.sdsp  s0,32(sp)
   1015c:   ec26                c.sdsp  s1,24(sp)
   1015e:   e84a                c.sdsp  s2,16(sp)
   10160:   e44e                c.sdsp  s3,8(sp)
   10162:   6575                c.lui   a0,0x1d
   10164:   bd050513            addi    a0,a0,-1072 # 1cbd0 <__clzdi2+0x3a>
   10168:   292000ef            jal     ra,103fa <puts>
   1016c:   0001f437            lui     s0,0x1f
   10170:   b6040493            addi    s1,s0,-1184 # 1eb60 <array>
```

```
   10174:   01c48913            addi   s2,s1,28
   10178:   b6040413            addi   s0,s0,-1184
   1017c:   69f5                c.lui  s3,0x1d
   1017e:   400c                c.lw   a1,0(s0)
   10180:   be098513            addi   a0,s3,-1056 # 1cbe0 <__clzdi2+0x4a>
   10184:   1ca000ef            jal    ra,1034e <printf>
   10188:   0411                c.addi s0,4
   1018a:   ff241ae3            bne    s0,s2,1017e <main+0x28>
   1018e:   459d                c.li   a1,7
   10190:   0001f537            lui    a0,0x1f
   10194:   b6050513            addi   a0,a0,-1184 # 1eb60 <array>
   10198:   030000ef            jal    ra,101c8 <reverse>
   1019c:   6575                c.lui  a0,0x1d
   1019e:   be850513            addi   a0,a0,-1048 # 1cbe8 <__clzdi2+0x52>
   101a2:   258000ef            jal    ra,103fa <puts>
   101a6:   6475                c.lui  s0,0x1d
   101a8:   408c                c.lw   a1,0(s1)
   101aa:   be040513            addi   a0,s0,-1056 # 1cbe0 <__clzdi2+0x4a>
   101ae:   1a0000ef            jal    ra,1034e <printf>
   101b2:   0491                c.addi s1,4
   101b4:   ff249ae3            bne    s1,s2,101a8 <main+0x52>
   101b8:   4501                c.li   a0,0
   101ba:   70a2                c.ldsp ra,40(sp)
   101bc:   7402                c.ldsp s0,32(sp)
   101be:   64e2                c.ldsp s1,24(sp)
   101c0:   6942                c.ldsp s2,16(sp)
   101c2:   69a2                c.ldsp s3,8(sp)
   101c4:   6145                c.addi16sp  sp,48
   101c6:   8082                c.jr   ra

00000000000101c8 <reverse>:
   101c8:   fff58813            addi   a6,a1,-1
   101cc:   02080663            beq    a6,zero,101f8 <reverse+0x30>
   101d0:   00259713            slli   a4,a1,0x2
   101d4:   972a                c.add  a4,a0
   101d6:   4781                c.li   a5,0
   101d8:   ffc72683            lw     a3,-4(a4)
   101dc:   4110                c.lw   a2,0(a0)
   101de:   fec72e23            sw     a2,-4(a4)
   101e2:   c114                c.sw   a3,0(a0)
   101e4:   0785                c.addi a5,1
   101e6:   01078963            beq    a5,a6,101f8 <reverse+0x30>
   101ea:   1771                c.addi a4,-4
   101ec:   0511                c.addi a0,4
   101ee:   fff7c693            xori   a3,a5,-1
   101f2:   96ae                c.add  a3,a1
   101f4:   fed7e2e3            bltu   a5,a3,101d8 <reverse+0x10>
   101f8:    8082                c.jr   ra
```

Адресация для вызовов функций изменилась на абсолютную.

3. Создание статической библиотеки

Выделим функция reverse в отдельную статическую библиотеку. Дляэтого надо получить объектный файл reverse.o и собрать библиотеку.

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

Рассмотрим список символов библиотеки:

Листинг 3.1 Список символов libReverse.a

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

```
reverse.o:
0000000000000030 t .L1
0000000000000010 t .L3
0000000000000000 T reverse
```

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=rv64iac -mabi=lp64 -O1 main.c libReverse.a
-o main.out
```

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
```

```
188  000000000001c894 g     F .text  0000000000000058 .hidden __floatsitf
189  0000000000015b5e g     F .text  00000000000000d6 memmove
190  0000000000012846 g     F .text  0000000000000010 _cleanup
191  0000000000015c38 g     F .text  0000000000000062 _Balloc
192  00000000000101c8 g     F .text  0000000000000032 reverse
193  0000000000019e76 g     F .text  0000000000000006 __errno
194  000000000001b6c4 g     F .text  000000000000005a _conv_stat
```

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

Создание make-файлов

Чтобы автоматизировать процесс сборки библиотеки и приложения напишем make-файлы.

На основе примеров с сайта курса, были написаны следующие файлы:

make_lib
all: lib lib: reverse.o riscv64-unknown-elf-ar -rsc libReverse.a reverse.o $(RM) -f *.o reverse.o: reverse.c riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -c reverse.c -o reverse.o
make_app
all: make -f make_lib riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 main.c libReverse.a -o main $(RM) -f *.o *.a

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

```
21.04.2021  12:01    <DIR>          .
21.04.2021  12:01    <DIR>          ..
17.04.2021  14:56               600 main.c
21.04.2021  12:00               126 make_app
21.04.2021  11:57               304 make_lib
17.04.2021  15:38               361 reverse.c
17.04.2021  15:38                93 reverse.h
               5 файлов          1 484 байт
               2 папок  257 608 749 056 байт свободно

C:\Users\Tuf gaming\Desktop\lab4\lib>make -f make_lib
make[12]: Entering directory '/cygdrive/c/Users/Tuf gaming/Desktop/lab4/lib'
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -c reverse.c -o reverse.o
riscv64-unknown-elf-ar -rsc libReverse.a reverse.o
rm -f -f *.o
make[12]: Leaving directory '/cygdrive/c/Users/Tuf gaming/Desktop/lab4/lib'

C:\Users\Tuf gaming\Desktop\lab4\lib>dir
 Том в устройстве C не имеет метки.
 Серийный номер тома: 0A24-354F

 Содержимое папки C:\Users\Tuf gaming\Desktop\lab4\lib

21.04.2021  12:01    <DIR>          .
21.04.2021  12:01    <DIR>          ..
21.04.2021  12:01             1 832 libReverse.a
17.04.2021  14:56               600 main.c
21.04.2021  12:00               126 make_app
21.04.2021  11:57               304 make_lib
17.04.2021  15:38               361 reverse.c
17.04.2021  15:38                93 reverse.h
               6 файлов          3 316 байт
               2 папок  257 608 744 960 байт свободно
```

```
 Содержимое папки C:\Users\Tuf gaming\Desktop\lab4\lib

21.04.2021  12:02    <DIR>          .
21.04.2021  12:02    <DIR>          ..
17.04.2021  14:56               600 main.c
21.04.2021  12:00               126 make_app
21.04.2021  11:57               304 make_lib
17.04.2021  15:38               361 reverse.c
17.04.2021  15:38                93 reverse.h
               5 файлов          1 484 байт
               2 папок  257 610 924 032 байт свободно

C:\Users\Tuf gaming\Desktop\lab4\lib>make -f make_app
make[12]: Entering directory '/cygdrive/c/Users/Tuf gaming/Desktop/lab4/lib'
make -f make_lib
make[13]: Entering directory '/cygdrive/c/Users/Tuf gaming/Desktop/lab4/lib'
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -c reverse.c -o reverse.o
riscv64-unknown-elf-ar -rsc libReverse.a reverse.o
rm -f -f *.o
make[13]: Leaving directory '/cygdrive/c/Users/Tuf gaming/Desktop/lab4/lib'
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 main.c libReverse.a -o main
rm -f -f *.o *.a
make[12]: Leaving directory '/cygdrive/c/Users/Tuf gaming/Desktop/lab4/lib'

C:\Users\Tuf gaming\Desktop\lab4\lib>dir
 Том в устройстве C не имеет метки.
 Серийный номер тома: 0A24-354F

 Содержимое папки C:\Users\Tuf gaming\Desktop\lab4\lib

21.04.2021  12:02    <DIR>          .
21.04.2021  12:02    <DIR>          ..
21.04.2021  12:02           143 224 main
17.04.2021  14:56               600 main.c
21.04.2021  12:00               126 make_app
21.04.2021  11:57               304 make_lib
17.04.2021  15:38               361 reverse.c
17.04.2021  15:38                93 reverse.h
               6 файлов        144 708 байт
               2 папок  257 610 780 672 байт свободно
```

Рис. 3.1 Выполнение make-файлов

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 с помощью пакета разработки “SiFive GNU Embedded Toolchain” для RISC-V. Были проанализированы выводы препроцессора, компилятора и линковщика последовательно отдельно друг от друга. Была создана своя статически линкуемая библиотека libReverse.a. Были написаны make-файлы для её сборки, а также для сборки тестовой программы с использованием библиотеки.