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: 15

Выполнил студент гр. 3530901/90002 _____З.А. Фрид
(подпись)

Принял преподаватель _____ Д.С. Степанов
(подпись)

“___” _____ 2021 г.

Санкт-Петербург

2021

Постановка 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

Согласно варианту 15, необходимо реализовать слияние двух отсортированных массивов.

Алгоритм

1. Сравнение элемента A(i) и B(j);
2. Если A(i) < B(j) и i меньше длины массива A или j больше или равен длине массива B, то есть все элементы массива бы обработаны, то в результирующий массив печатается A(i), а для следующего прохода в массиве A берется следующий элемент, B(j) остается прежним;
3. Иначе в итоговый массив печатается B(j), а для следующего прохода в массиве B берется следующий элемент, A(i) остается прежним;
4. Цикл 1-3 повторяется до тех пор, пока k, индекс элементов результирующего массива, будет меньше суммы длин массивов A и B.

1. Программа на языке C

Согласно заданию, была написана программа, сливающая два отсортированных массива. Функция помещена в отдельный файл, оформлен заголовочный файл.

Листинг 1.1 Заголовочный файл merge.h

```
#ifndef MERGE_H
#define MERGE_H

#include <stddef.h>

extern unsigned* merge(const unsigned arrayA[], size_t lengthA, const unsigned
arrayB[], size_t lengthB);

#endif //MERGE_H
```

В заголовочном файле определяем функцию слияния двух отсортированных массивов для её использования в тестовой программе.

Листинг 1.2 Файл программы main.c

```
#include <stddef.h>
#include <stdio.h>
#include <malloc.h>
#include "merge.h"

const unsigned arrayA[] = {1, 3, 4, 5, 5, 6, 9, 10};
const unsigned arrayB[] = {2, 3, 5, 7};
const size_t lengthA = sizeof arrayA / sizeof arrayA[0];
const size_t lengthB = sizeof arrayB / sizeof arrayB[0];

int main() {
    for (int i = 0; i < lengthA; i++) {
        printf("%d ", arrayA[i]);
    }
    printf("\n");

    for (int i = 0; i < lengthB; i++) {
        printf("%d ", arrayB[i]);
    }
    printf("\n");

    unsigned *arrayR = merge(arrayA, lengthA, arrayB, lengthB);

    for (int i = 0; i < lengthA + lengthB; i++) {
        printf("%d ", arrayR[i]);
    }

    free(arrayR);
    return 0;
}
```

Были подключена стандартные библиотеки “stddef.h”, которая требуется для определения типа size_t, и “ stdio.h”, которая используется для вывода на консоль. Так же подключена библиотека “malloc.h” для использования функций динамического распределения памяти. В функции main() начинается исполнение. В ней мы печатаем входные массивы, вызываем функцию merge(), и сохраняем результат в массив arrayR, печатаем его, а затем занятую им память следует освободить через функцию free() в конце программы.

Листинг 1.3 Функция слияния двух отсортированных массивов merge.c

```c
#include <malloc.h>
#include <stddef.h>

unsigned* merge(const unsigned arrayA[], const size_t lengthA, const unsigned
arrayB[], const size_t lengthB) {
    const size_t length = lengthA + lengthB;

    unsigned* result = (unsigned *) malloc(sizeof arrayA[0] * length);

    int i = 0, j = 0;
    for (int k = 0; k < length; k++) {
        if (j >= lengthB || (i < lengthA && arrayA[i] < arrayB[j])) {
            result[k] = arrayA[i];
            i++;
        } else {
            result[k] = arrayB[j];
            j++;
        }
    }

    return result;
}
```

Произведём компиляцию программы:

```
C:\Users\Z\CLionProjects\llp4\cmake-build-debug\llp4.exe
1 3 4 5 5 6 9 10
2 3 5 7
1 2 3 3 4 5 5 5 6 7 9 10
```

Рис. 1 Результат работы программы

В первой строке выведен первый отсортированный массив (arrayA), во второй – второй отсортированный массив (arrayB), в третьей – результат слияния двух массивов (arrayR).

2. Сборка программы «по шагам»

Препроцессирование

Выполним препроцессирование файлов с помощью пакета разработки «SiFive GNU Embedded Toolchain» для RISK-V. Для этого необходимо выполнить следующие команды:

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

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

--save-temps	сохранять промежуточные файлы, создаваемые в процессе сборки
-march=rv64iac -mabi=lp64	целевым является процессор с базовой архитектурой системы команд RV64IAC
-O1	выполнять простые оптимизации генерируемого кода
-v	печатать (в стандартный поток ошибок) выполняемые драйвером команды, а также дополнительную информацию
>log	вместо печати в консоли вывод программы направляется в файл с именем “log”
-E	обработка файлов будет выполнятся только препроцессором

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

Листинг 2.1 Файл main.i (часть)

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

# 6 "merge.h"
extern unsigned* merge(const unsigned arrayA[], size_t lengthA, const unsigned
arrayB[], size_t lengthB);
# 6 "main.c" 2

const unsigned arrayA[] = {1, 3, 4, 5, 5, 6, 9, 10};
const unsigned arrayB[] = {2, 3, 5, 7};
const size_t lengthA = sizeof arrayA / sizeof arrayA[0];
const size_t lengthB = sizeof arrayB / sizeof arrayB[0];

int main() {
    for (int i = 0; i < lengthA; i++) {
        printf("%d ", arrayA[i]);
    }
    printf("\n");

    for (int i = 0; i < lengthB; i++) {
        printf("%d ", arrayB[i]);
    }
    printf("\n");

    unsigned *arrayR = merge(arrayA, lengthA, arrayB, lengthB);

    for (int i = 0; i < lengthA + lengthB; i++) {
        printf("%d ", arrayR[i]);
    }

    free(arrayR);
    return 0;
}
```

Листинг 2.2 Файл merge.i (часть)

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

# 4 "merge.c"
unsigned* merge(const unsigned arrayA[], const size_t lengthA, const unsigned
arrayB[], const size_t lengthB) {
    const size_t length = lengthA + lengthB;

    unsigned* result = (unsigned *) malloc(sizeof arrayA[0] * length);

    int i = 0, j = 0;
    for (int k = 0; k < length; k++) {
        if (j >= lengthB || (i < lengthA && arrayA[i] < arrayB[j])) {
            result[k] = arrayA[i];
            i++;
```

```
        } else {
            result[k] = arrayB[j];
            j++;
        }
    }

    return result;
}
```

Появившиеся нестандартные директивы, начинающиеся с символа “#”, используются для передачи информации об исходном тексте из препроцессора в компилятор.

Например, последняя директива «# 1 “main.c”» в файле main.i информирует компилятор о том, что следующая строка является результатом обработки строки 1 исходного файла “main.c”. Аналогично директива «# 1 “merge.c”» в файле merge.i информирует компилятор о том, что следующая строка является результатом обработки строки 1 исходного файла “merge.c”.

Компиляция

Для компиляции препроцессированных файлов используем следующие команды:

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

Ниже приведены файлы – результаты компиляции:

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.str1.8,"aMS",@progbits,1
   .align 3
.LC0:
```

```
   .string    "%d "
   .text
   .align 1
   .globl main
   .type  main, @function
main:
   addi   sp,sp,-48
   sd ra,40(sp)
   sd s0,32(sp)
   sd s1,24(sp)
   sd s2,16(sp)
   sd s3,8(sp)
   lui    s0,%hi(.LANCHOR0)
   addi   s0,s0,%lo(.LANCHOR0)
   addi   s2,s0,32
   lui    s1,%hi(.LC0)
.L2:
   lw a1,0(s0)
   addi   a0,s1,%lo(.LC0)
   call   printf
   addi   s0,s0,4
   bne    s2,s0,.L2
   li a0,10
   call   putchar
   li a1,2
   lui    s0,%hi(.LC0)
   addi   a0,s0,%lo(.LC0)
   call   printf
   li a1,3
   addi   a0,s0,%lo(.LC0)
   call   printf
   li a1,5
   addi   a0,s0,%lo(.LC0)
   call   printf
   li a1,7
   addi   a0,s0,%lo(.LC0)
   call   printf
   li a0,10
   call   putchar
   lui    a0,%hi(.LANCHOR0)
   addi   a2,a0,%lo(.LANCHOR0)
   li a3,4
   addi   a2,a2,32
   li a1,8
   addi   a0,a0,%lo(.LANCHOR0)
   call   merge
   mv s3,a0
   mv s0,a0
   addi   s2,a0,48
   lui    s1,%hi(.LC0)
.L3:
   lw a1,0(s0)
   addi   a0,s1,%lo(.LC0)
   call   printf
   addi   s0,s0,4
   bne    s0,s2,.L3
   mv a0,s3
   call   free
   li a0,0
   ld ra,40(sp)
   ld s0,32(sp)
   ld s1,24(sp)
   ld s2,16(sp)
```

```
   ld s3,8(sp)
   addi   sp,sp,48
   jr ra
   .size  main, .-main
   .globl lengthB
   .globl lengthA
   .globl arrayB
   .globl arrayA
   .section   .rodata
   .align 3
   .set   .LANCHOR0,. + 0
   .type  arrayA, @object
   .size  arrayA, 32
arrayA:
   .word  1
   .word  3
   .word  4
   .word  5
   .word  5
   .word  6
   .word  9
   .word  10
   .type  arrayB, @object
   .size  arrayB, 16
arrayB:
   .word  2
   .word  3
   .word  5
   .word  7
   .section   .srodata,"a"
   .align 3
   .type  lengthB, @object
   .size  lengthB, 8
lengthB:
   .dword 4
   .type  lengthA, @object
   .size  lengthA, 8
lengthA:
   .dword 8
   .ident "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"
```

Листинг 2.4 Файл merge.s

```
   .file  "merge.c"
   .option nopic
   .attribute arch, "rv64i2p0_a2p0_c2p0"
   .attribute unaligned_access, 0
   .attribute stack_align, 16
   .text
   .align 1
   .globl merge
   .type  merge, @function
merge:
   addi   sp,sp,-64
   sd ra,56(sp)
   sd s0,48(sp)
   sd s1,40(sp)
   sd s2,32(sp)
   sd s3,24(sp)
```

```
   sd s4,16(sp)
   sd s5,8(sp)
   mv s1,a0
   mv s2,a1
   mv s3,a2
   mv s0,a3
   add    s5,a1,a3
   slli   s4,s5,2
   mv a0,s4
   call   malloc
   beq    s5,zero,.L1
   mv a5,a0
   add    a3,s4,a0
   li a6,0
   li a7,0
   j  .L6
.L3:
   slli   a4,a7,2
   add    a4,s1,a4
   lw a4,0(a4)
   sw a4,0(a5)
   addiw  a7,a7,1
.L5:
   addi   a5,a5,4
   beq    a5,a3,.L1
.L6:
   mv a4,a6
   bgeu   a6,s0,.L3
   bgeu   a7,s2,.L4
   slli   a1,a7,2
   add    a1,s1,a1
   slli   a2,a6,2
   add    a2,s3,a2
   lw a1,0(a1)
   lw a2,0(a2)
   bltu   a1,a2,.L3
.L4:
   slli   a4,a4,2
   add    a4,s3,a4
   lw a4,0(a4)
   sw a4,0(a5)
   addiw  a6,a6,1
   j  .L5
.L1:
   ld ra,56(sp)
   ld s0,48(sp)
   ld s1,40(sp)
   ld s2,32(sp)
   ld s3,24(sp)
   ld s4,16(sp)
   ld s5,8(sp)
   addi   sp,sp,64
   jr ra
   .size  merge, .-merge
   .ident "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"
```

Объектный файл

Выполним ассемблирование для получения объектных файлов программы. Для этого исполним следующие команды:

```
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -v -c main.s -o
main.o >log_obj_main.txt 2>&1
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -v -c merge.s -o
merge.o >log_obj_merge.txt 2>&1
```

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

Введем команду:

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

Листинг 2.5 Хедер файла main.o

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

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .text         000000d6  0000000000000000  0000000000000000  00000040  2**1
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  0000000000000000  0000000000000000  00000116  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  0000000000000000  0000000000000000  00000116  2**0
                  ALLOC
  3 .rodata.str1.8 00000004  0000000000000000  0000000000000000  00000118  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .rodata       00000030  0000000000000000  0000000000000000  00000120  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  5 .srodata      00000010  0000000000000000  0000000000000000  00000150  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  6 .comment      00000031  0000000000000000  0000000000000000  00000160  2**0
                  CONTENTS, READONLY
  7 .riscv.attributes 00000026  0000000000000000  0000000000000000  00000191  2**0
                  CONTENTS, READONLY
```

Введем команду:

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

Листинг 2.6 Хедер файла merge.o

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

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .text         00000088  0000000000000000  0000000000000000  00000040  2**1
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  0000000000000000  0000000000000000  000000c8  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  0000000000000000  0000000000000000  000000c8  2**0
                  ALLOC
  3 .comment      00000031  0000000000000000  0000000000000000  000000c8  2**0
                  CONTENTS, READONLY
  4 .riscv.attributes 00000026  0000000000000000  0000000000000000  000000f9  2**0
                  CONTENTS, READONLY
```

Вся информация размещается в секциях.

Секция	Назначение
.text	секция кода, в которой содержатся коды инструкций
.data	секция инициализированных данных
.rodata	секция read-only данных
.srodata	это небольшой раздел rodata
.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.

Введем команду:

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

Листинг 2.7 Дизассемблированный файл main.o

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


Disassembly of section .text:

0000000000000000 <main>:
   0:   7179                    c.addi16sp      sp,-48
   2:   f406                    c.sdsp  ra,40(sp)
   4:   f022                    c.sdsp  s0,32(sp)
   6:   ec26                    c.sdsp  s1,24(sp)
   8:   e84a                    c.sdsp  s2,16(sp)
   a:   e44e                    c.sdsp  s3,8(sp)
   c:   00000437                lui     s0,0x0
  10:   00040413                addi    s0,s0,0 # 0 <main>
  14:   02040913                addi    s2,s0,32
  18:   000004b7                lui     s1,0x0

000000000000001c <.L2>:
  1c:   400c                    c.lw    a1,0(s0)
  1e:   00048513                addi    a0,s1,0 # 0 <main>
  22:   00000097                auipc   ra,0x0
  26:   000080e7                jalr    ra,0(ra) # 22 <.L2+0x6>
  2a:   0411                    c.addi  s0,4
  2c:   fe8918e3                bne     s2,s0,1c <.L2>
  30:   4529                    c.li    a0,10
  32:   00000097                auipc   ra,0x0
  36:   000080e7                jalr    ra,0(ra) # 32 <.L2+0x16>
  3a:   4589                    c.li    a1,2
  3c:   00000437                lui     s0,0x0
  40:   00040513                addi    a0,s0,0 # 0 <main>
  44:   00000097                auipc   ra,0x0
  48:   000080e7                jalr    ra,0(ra) # 44 <.L2+0x28>
  4c:   458d                    c.li    a1,3
  4e:   00040513                addi    a0,s0,0
  52:   00000097                auipc   ra,0x0
  56:   000080e7                jalr    ra,0(ra) # 52 <.L2+0x36>
  5a:   4595                    c.li    a1,5
  5c:   00040513                addi    a0,s0,0
  60:   00000097                auipc   ra,0x0
  64:   000080e7                jalr    ra,0(ra) # 60 <.L2+0x44>
  68:   459d                    c.li    a1,7
  6a:   00040513                addi    a0,s0,0
  6e:   00000097                auipc   ra,0x0
  72:   000080e7                jalr    ra,0(ra) # 6e <.L2+0x52>
  76:   4529                    c.li    a0,10
  78:   00000097                auipc   ra,0x0
  7c:   000080e7                jalr    ra,0(ra) # 78 <.L2+0x5c>
  80:   00000537                lui     a0,0x0
  84:   00050613                addi    a2,a0,0 # 0 <main>
  88:   4691                    c.li    a3,4
  8a:   02060613                addi    a2,a2,32
  8e:   45a1                    c.li    a1,8
  90:   00050513                addi    a0,a0,0
  94:   00000097                auipc   ra,0x0
  98:   000080e7                jalr    ra,0(ra) # 94 <.L2+0x78>
  9c:   89aa                    c.mv    s3,a0
  9e:   842a                    c.mv    s0,a0
  a0:   03050913                addi    s2,a0,48
  a4:   000004b7                lui     s1,0x0
```

```
00000000000000a8 <.L3>:
  a8:   400c                    c.lw    a1,0(s0)
  aa:   00048513                addi    a0,s1,0 # 0 <main>
  ae:   00000097                auipc   ra,0x0
  b2:   000080e7                jalr    ra,0(ra) # ae <.L3+0x6>
  b6:   0411                    c.addi  s0,4
  b8:   ff2418e3                bne     s0,s2,a8 <.L3>
  bc:   854e                    c.mv    a0,s3
  be:   00000097                auipc   ra,0x0
  c2:   000080e7                jalr    ra,0(ra) # be <.L3+0x16>
  c6:   4501                    c.li    a0,0
  c8:   70a2                    c.ldsp  ra,40(sp)
  ca:   7402                    c.ldsp  s0,32(sp)
  cc:   64e2                    c.ldsp  s1,24(sp)
  ce:   6942                    c.ldsp  s2,16(sp)
  d0:   69a2                    c.ldsp  s3,8(sp)
  d2:   6145                    c.addi16sp      sp,48
  d4:   8082                    c.jr    ra
```

Рассмотрим таблицу символов, выполнив команду:

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

Листинг 2.8 Таблица символов

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

SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 merge.c
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000000 l    d  .data  0000000000000000 .data
0000000000000000 l    d  .bss   0000000000000000 .bss
0000000000000076 l       .text  0000000000000000 .L1
000000000000004c l       .text  0000000000000000 .L6
000000000000003a l       .text  0000000000000000 .L3
000000000000006a l       .text  0000000000000000 .L4
0000000000000046 l       .text  0000000000000000 .L5
0000000000000000 l    d  .comment       0000000000000000 .comment
0000000000000000 l    d  .riscv.attributes      0000000000000000 .riscv.attributes
0000000000000000 g     F .text  0000000000000088 merge
0000000000000000         *UND*  0000000000000000 malloc



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    d  .rodata        0000000000000000 .rodata
```

```
0000000000000000 l       .rodata        0000000000000000 .LANCHOR0
0000000000000000 l    d  .srodata       0000000000000000 .srodata
0000000000000000 l       .rodata.str1.8 0000000000000000 .LC0
000000000000001c l       .text  0000000000000000 .L2
00000000000000a8 l       .text  0000000000000000 .L3
0000000000000000 l    d  .comment       0000000000000000 .comment
0000000000000000 l    d  .riscv.attributes      0000000000000000 .riscv.attributes
0000000000000000 g     F .text  00000000000000d6 main
0000000000000000         *UND*  0000000000000000 printf
0000000000000000         *UND*  0000000000000000 putchar
0000000000000000         *UND*  0000000000000000 merge
0000000000000000         *UND*  0000000000000000 free
0000000000000000 g     O .srodata       0000000000000008 lengthB
0000000000000008 g     O .srodata       0000000000000008 lengthA
0000000000000020 g     O .rodata        0000000000000010 arrayB
0000000000000000 g     O .rodata        0000000000000020 arrayA
```

В каждой таблице по одному глобальному символу (флаг “g”) типа функция (“F”) - это функции main() и merge().

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:

riscv64-unknown-elf-objdump -r merge.o main.o

Листинг 2.9 Таблица перемещений

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

RELOCATION RECORDS FOR [.text]:
OFFSET           TYPE              VALUE
0000000000000022 R_RISCV_CALL      malloc
0000000000000022 R_RISCV_RELAX     *ABS*
000000000000002a R_RISCV_BRANCH    .L1
0000000000000038 R_RISCV_RVC_JUMP  .L6
0000000000000048 R_RISCV_BRANCH    .L1
000000000000004e R_RISCV_BRANCH    .L3
0000000000000052 R_RISCV_BRANCH    .L4
0000000000000066 R_RISCV_BRANCH    .L3
0000000000000074 R_RISCV_RVC_JUMP  .L5
```

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

RELOCATION RECORDS FOR [.text]:
OFFSET           TYPE              VALUE
000000000000000c R_RISCV_HI20      .LANCHOR0
000000000000000c R_RISCV_RELAX     *ABS*
0000000000000010 R_RISCV_LO12_I    .LANCHOR0
0000000000000010 R_RISCV_RELAX     *ABS*
0000000000000018 R_RISCV_HI20      .LC0
0000000000000018 R_RISCV_RELAX     *ABS*
000000000000001e R_RISCV_LO12_I    .LC0
000000000000001e R_RISCV_RELAX     *ABS*
0000000000000022 R_RISCV_CALL      printf
0000000000000022 R_RISCV_RELAX     *ABS*
0000000000000032 R_RISCV_CALL      putchar
0000000000000032 R_RISCV_RELAX     *ABS*
000000000000003c R_RISCV_HI20      .LC0
000000000000003c R_RISCV_RELAX     *ABS*
0000000000000040 R_RISCV_LO12_I    .LC0
0000000000000040 R_RISCV_RELAX     *ABS*
0000000000000044 R_RISCV_CALL      printf
0000000000000044 R_RISCV_RELAX     *ABS*
000000000000004e R_RISCV_LO12_I    .LC0
000000000000004e R_RISCV_RELAX     *ABS*
0000000000000052 R_RISCV_CALL      printf
0000000000000052 R_RISCV_RELAX     *ABS*
000000000000005c R_RISCV_LO12_I    .LC0
000000000000005c R_RISCV_RELAX     *ABS*
0000000000000060 R_RISCV_CALL      printf
0000000000000060 R_RISCV_RELAX     *ABS*
000000000000006a R_RISCV_LO12_I    .LC0
000000000000006a R_RISCV_RELAX     *ABS*
000000000000006e R_RISCV_CALL      printf
000000000000006e R_RISCV_RELAX     *ABS*
0000000000000078 R_RISCV_CALL      putchar
0000000000000078 R_RISCV_RELAX     *ABS*
0000000000000080 R_RISCV_HI20      .LANCHOR0
0000000000000080 R_RISCV_RELAX     *ABS*
0000000000000084 R_RISCV_LO12_I    .LANCHOR0
0000000000000084 R_RISCV_RELAX     *ABS*
0000000000000090 R_RISCV_LO12_I    .LANCHOR0
0000000000000090 R_RISCV_RELAX     *ABS*
0000000000000094 R_RISCV_CALL      merge
0000000000000094 R_RISCV_RELAX     *ABS*
00000000000000a4 R_RISCV_HI20      .LC0
00000000000000a4 R_RISCV_RELAX     *ABS*
00000000000000aa R_RISCV_LO12_I    .LC0
00000000000000aa R_RISCV_RELAX     *ABS*
00000000000000ae R_RISCV_CALL      printf
00000000000000ae R_RISCV_RELAX     *ABS*
00000000000000be R_RISCV_CALL      free
00000000000000be R_RISCV_RELAX     *ABS*
000000000000002c R_RISCV_BRANCH    .L2
00000000000000b8 R_RISCV_BRANCH    .L3
```

В таблице перемещений для main.o наблюдаем вызов метода merge. 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 командой:

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

В результате выполнения этой команды был получен файл main.out – исполняемый бинарный файл. Рассмотрим его секцию кода с помощью команды:

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

Листинг 2.10 Исполняемый файл (часть)

```
67  0000000000010156 <main>:
68     10156:   7179                    c.addi16sp  sp,-48
69     10158:   f406                    c.sdsp  ra,40(sp)
70     1015a:   f022                    c.sdsp  s0,32(sp)
71     1015c:   ec26                    c.sdsp  s1,24(sp)
72     1015e:   e84a                    c.sdsp  s2,16(sp)
73     10160:   e44e                    c.sdsp  s3,8(sp)
74     10162:   6475                    c.lui   s0,0x1d
75     10164:   cb840413            addi    s0,s0,-840 # 1ccb8 <arrayA>
76     10168:   02040913            addi    s2,s0,32
77     1016c:   64f5                    c.lui   s1,0x1d
78     1016e:   400c                    c.lw    a1,0(s0)
79     10170:   cb048513            addi    a0,s1,-848 # 1ccb0 <__clzdi2+0x36>
80     10174:   09b000ef            jal ra,10a0e <printf>
81     10178:   0411                    c.addi  s0,4
82     1017a:   fe891ae3            bne s2,s0,1016e <main+0x18>
83     1017e:   4529                    c.li    a0,10
84     10180:   0bf000ef            jal ra,10a3e <putchar>
85     10184:   4589                    c.li    a1,2
86     10186:   6475                    c.lui   s0,0x1d
87     10188:   cb040513            addi    a0,s0,-848 # 1ccb0 <__clzdi2+0x36>
88     1018c:   083000ef            jal ra,10a0e <printf>
89     10190:   458d                    c.li    a1,3
90     10192:   cb040513            addi    a0,s0,-848
91     10196:   079000ef            jal ra,10a0e <printf>
92     1019a:   4595                    c.li    a1,5
93     1019c:   cb040513            addi    a0,s0,-848
94     101a0:   06f000ef            jal ra,10a0e <printf>
```

```
   101a4:   459d                    c.li    a1,7
   101a6:   cb040513              addi    a0,s0,-848
   101aa:   065000ef              jal ra,10a0e <printf>
   101ae:   4529                    c.li    a0,10
   101b0:   08f000ef              jal ra,10a3e <putchar>
   101b4:   6575                    c.lui   a0,0x1d
   101b6:   cb850613              addi    a2,a0,-840 # 1ccb8 <arrayA>
   101ba:   4691                    c.li    a3,4
   101bc:   02060613              addi    a2,a2,32
   101c0:   45a1                    c.li    a1,8
   101c2:   cb850513              addi    a0,a0,-840
   101c6:   034000ef              jal ra,101fa <merge>
   101ca:   89aa                    c.mv    s3,a0
   101cc:   842a                    c.mv    s0,a0
   101ce:   03050913              addi    s2,a0,48
   101d2:    64f5                     c.lui   s1,0x1d
   101d4:    400c                     c.lw    a1,0(s0)
   101d6:    cb048513               addi    a0,s1,-848 # 1ccb0 <__clzdi2+0x36>
   101da:    035000ef               jal ra,10a0e <printf>
   101de:    0411                     c.addi  s0,4
   101e0:    ff241ae3               bne s0,s2,101d4 <main+0x7e>
   101e4:    854e                     c.mv    a0,s3
   101e6:    128000ef               jal ra,1030e <free>
   101ea:    4501                     c.li    a0,0
   101ec:    70a2                     c.ldsp  ra,40(sp)
   101ee:    7402                     c.ldsp  s0,32(sp)
   101f0:    64e2                     c.ldsp  s1,24(sp)
   101f2:    6942                     c.ldsp  s2,16(sp)
   101f4:    69a2                     c.ldsp  s3,8(sp)
   101f6:    6145                     c.addi16sp  sp,48
   101f8:    8082                     c.jr    ra

00000000000101fa <merge>:
   101fa:    7139                     c.addi16sp  sp,-64
   101fc:    fc06                     c.sdsp  ra,56(sp)
   101fe:    f822                     c.sdsp  s0,48(sp)
   10200:    f426                     c.sdsp  s1,40(sp)
   10202:    f04a                     c.sdsp  s2,32(sp)
   10204:    ec4e                     c.sdsp  s3,24(sp)
   10206:    e852                     c.sdsp  s4,16(sp)
   10208:    e456                     c.sdsp  s5,8(sp)
   1020a:    84aa                     c.mv    s1,a0
   1020c:    892e                     c.mv    s2,a1
   1020e:    89b2                     c.mv    s3,a2
   10210:    8436                     c.mv    s0,a3
   10212:    00d58ab3               add s5,a1,a3
   10216:    002a9a13               slli    s4,s5,0x2
   1021a:    8552                     c.mv    a0,s4
   1021c:    0ea000ef               jal ra,10306 <malloc>
   10220:    040a8663               beq s5,zero,1026c <merge+0x72>
   10224:    87aa                     c.mv    a5,a0
   10226:    00aa06b3               add a3,s4,a0
   1022a:    4801                     c.li    a6,0
   1022c:    4881                     c.li    a7,0
   1022e:    a811                     c.j 10242 <merge+0x48>
   10230:    00289713               slli    a4,a7,0x2
   10234:    9726                     c.add   a4,s1
   10236:    4318                     c.lw    a4,0(a4)
```

Адресация для вызовов функций изменилась на абсолютную.

3. Создание статической библиотеки

Статическая библиотека (static library) является, по сути, архивом (набо-ром, коллекцией) объектных файлов, среди которых компоновщик выбирает «полезные» для данной программы: объектный файл считается «полезным», ес-ли в нем определяется еще не разрешенный компоновщиком символ.

Выделим функцию merge в отдельную статическую библиотеку. Для этого надо получить объектный файл merge.o и собрать библиотеку.

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

Рассмотрим список символов libMerge.a с помощью команды:

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

Листинг 3.1 Список символов libMerge.a

```
merge.o:
0000000000000076 t .L1
000000000000003a t .L3
000000000000006a t .L4
0000000000000046 t .L5
000000000000004c t .L6
                 U malloc
0000000000000000 T merge
```

В выводе утилиты “nm” кодом “T” обозначаются символы, определенные в соответствующем объектном файле. Единственный внешний символ - malloc - библиотека для выделения памяти под массивы. Символ функции merge является основным символом, определяемым в этом объектном файле.

Используя собранную библиотеку, произведём исполняемый файл тестовой программы с помощью команды:

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

Посмотрим содержимое таблицы символов исполняемого файла с помощью команды и убедимся, что там есть функция merge:

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

Листинг 3.2 Таблица символов main.out

```
34  0000000000000000 l    df *ABS*  0000000000000000 main.c
35  0000000000000000 l    df *ABS*  0000000000000000 merge.c
36  0000000000000000 l    df *ABS*  0000000000000000 exit.c
37  0000000000000000 l    df *ABS*  0000000000000000 impure.c
38  000000000001ec50 l     O .data  0000000000000748 impure_data
```

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

Создание make-файлов

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

Содержимое make_lib файла

```
1  # Чтобы достичь цели "all", требуется построить библиотеку
2  all: lib
3
4  lib: merge.o
5      riscv64-unknown-elf-ar -rsc libMerge.a merge.o
6      $(RM) -f *.s *.o
7
8  merge.o: merge.c
9      riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -c merge.c -o merge.o
```

Содержимое make_app файла

```
1  all:
2      C:\cygwin64\bin\make.exe -f make_lib
3      riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 main.c libMerge.a -o main
4      $(RM) -f *.a *.o
```

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

```
C:\lab>C:\cygwin64\bin\make.exe -f make_lib
riscv64-unknown-elf-ar -rsc libMerge.a merge.o
cmd //C del -f *.o
Microsoft Windows [Version 10.0.18363.1500]
(c) Корпорация Майкрософт (Microsoft Corporation), 2019. Все права защищены.

C:\lab>dir
 Том в устройстве C не имеет метки.
 Серийный номер тома: A475-F44D

 Содержимое папки C:\lab

21.04.2021  12:40    <DIR>          .
21.04.2021  12:40    <DIR>          ..
21.04.2021  12:40             1 838 libMerge.a
20.04.2021  18:40               689 main.c
21.04.2021  11:34               173 make_app
21.04.2021  12:40               293 make_lib
20.04.2021  18:43               584 merge.c
20.04.2021  18:38               186 merge.h
               6 файлов          3 763 байт
               2 папок  135 404 158 976 байт свободно
```

Рис. 2 Выполнение make-файлов (1)

```
C:\lab>C:\cygwin64\bin\make.exe -f make_app
make[2]: Entering directory '/cygdrive/c/lab'
C:\cygwin64\bin\make.exe -f make_lib
make[3]: Entering directory '/cygdrive/c/lab'
riscv64-unknown-elf-ar -rsc libMerge.a merge.o
make[3]: Leaving directory '/cygdrive/c/lab'
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 main.c libMerge.a -o main
make[2]: Leaving directory '/cygdrive/c/lab'

C:\lab>dir
 Том в устройстве C не имеет метки.
 Серийный номер тома: A475-F44D

 Содержимое папки C:\lab

21.04.2021  12:48    <DIR>          .
21.04.2021  12:48    <DIR>          ..
21.04.2021  12:48           143 664 main
20.04.2021  18:40               689 main.c
21.04.2021  12:47               123 make_app
21.04.2021  12:48               272 make_lib
20.04.2021  18:43               584 merge.c
20.04.2021  18:38               186 merge.h
               6 файлов        145 518 байт
               2 папок  131 248 742 400 байт свободно
```

Рис. 3 Выполнение make-файлов (3)

Вывод

В ходе выполнения лабораторной работы была написана программа на языке C с заданной функциональностью (слияние двух отсортированных массивов). 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 libMerge.a. Были написаны make-файлы для её сборки, а также сборки тестовой программы с использованием библиотеки.

Список использованных источников

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