Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Лабораторная работа № 2**

Программирование RISC-V

по дисциплине «Низкоуровневое программирование»

Выполнил

студент гр. 3530901/90004

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Остапчук А.С.

(подпись)

Руководитель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Алексюк А.О.

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Санкт-Петербург   
2021

**Задача**

1. Изучить методические материалы, опубликованные на сайте курса.

2. Установить пакет средств разработки “SiFive GNU Embedded Toolchain” для RISC-V.

3. 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.

4. 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 исполняеммом файле.

5. 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**

Вариант: 9 - Расчет биномиальных коэффициентов для данного показателя по треугольнику Паскаля.

**Выполнение работы**

Показатель – переменная parameter

Изначально мы имеем массив из 2 элементов, равных 1. Суть алгоритма заключается в том, чтобы проходить по массиву, складывать значения двух соседних элементов и перезаписывать их и увеличивать массив в размерах. По массиву мы проходим столько раз, сколько необходимо для вычисления коэффициентов для заданного показателя.

Код программы на языке C состоит из трёх файлов - ниже представлены эти файлы:  
Файл main.c  
#include <stdio.h>

#include <stdlib.h>

#include "find\_koeffs.h"

int main()

{

int parameter = 5;

int \*array = (int \*)calloc(parameter + 1, sizeof(int));

array[0] = 1;

array[1] = 1;

find\_koeffs(parameter, array);

for (int i = 0; i <= parameter; i++)

{

printf("%i \n", array[i]);

}

return 0;

}

Файл find\_koeffs.h

#include <stddef.h>

int find\_koeffs(int param, int \*array);

Файл find\_koeffs.c

#include "find\_koeffs.h"

int find\_koeffs(int param, int \*array)

{

int koeff1 = 0;

int koeff2 = 0;

for (int i = 0; i < param; i++)

{

for (int j = 0; j <= i; j++)

{

koeff2 = array[j] + array[j + 1];

if (j != 0)

{

array[j] = koeff1;

}

koeff1 = koeff2;

}

array[i + 1] = 1;

}

return 0;

}

Для сборки программы использовался make, который требовал наличие Makefile в папке с исходными текстами. Был создан Makefile:

# «Фиктивные» цели

.PHONY: all clean

# Объектные файлы

OBJS\_LIB = find\_koeffs.o

OBJS\_APP = main.o

# Файл библиотеки

# Действует соглашение: файл библиотеки <mylib> имеет имя lib<mylib>.a

MYLIBNAME = libfind\_koeffs.a

# Файл программы

MYAPPNAME = myapp.out

# Чтобы достичь цели «all», требуется построить библиотеку

all: $(MYLIBNAME) $(MYAPPNAME)

# Чтобы достичь цели «clean», требуется удалить созданные при сборке файлы

# - объектные файлы

# - файлы зависимостей

# - файл библиотеки

clean:

$(RM) –f $(OBJS\_LIB)

$(RM) –f $(OBJS\_APP)

$(RM) –f $(OBJS\_LIB:.o=.d)

$(RM) –f $(OBJS\_APP:.o=.d)

$(RM) –f $(MYLIBNAME)

$(RM) –f $(MYAPPNAME)

# Параметры компиляции

CFLAGS= -std=c11 –pedantic –Wall –O1

# Включаемые файлы следует искать в каталоге «include»

INCLUDES+= -I ./include

# Make должна искать файлы \*.h в каталогах include и src

vpath %.h include src

# …, а файлы \*.c – в каталоге src

vpath %.c src

# Рецепт построения файла библиотеки из имеющихся объектных файлов

# значением переменной $@ будет имя файла библиотеки

# значением переменной $^ будет список (с пробелом в качестве символа-разделителя) зависимостей

# (с удаленными дубликатами), в данном случае – список объектных файлов

$(MYLIBNAME): $(OBJS\_LIB)

$(AR) –rsc $@ $^

# Рецепт построения объектного файла из исходного текста

# значением переменной $< будет имя первого файла в списке зависимостей, в данном случае – имя исходного файла

%.o: %.c

gcc –MD $(CFLAGS) $(INCLUDES) –c –o $@ $<

# Включаем всю имеющуюся информацию о зависимостях

# Лидирующий «-« означает, что отсутствие файла .d не является ошибкой

-include $(OBJS\_LIB:.o=.d)

$(MYAPPNAME): $(OBJS\_APP) $(MYLIBNAME)

gcc –o $@ $^

-include $(OBJS\_APP:.o=.d)

Сборка программы

Запустим отдельно препроцессор при помощи команды.

gcc -E find\_koeffs.c

Вывод препроцессора:

# 1 "find\_koeffs.c"

# 1 "<built-in>"

# 1 "<command-line>"

# 1 "find\_koeffs.c"

# 1 "find\_koeffs.h" 1

# 1 "c:\\users\\trall\\clionprojects\\riscv64-unknown-elf-gcc-2018.07.0-x86\_64-w64-mingw32\\lib\\gcc\\riscv64-unknown-elf\\8.1.0\\include\\stddef.h" 1 3 4

# 149 "c:\\users\\trall\\clionprojects\\riscv64-unknown-elf-gcc-2018.07.0-x86\_64-w64-mingw32\\lib\\gcc\\riscv64-unknown-elf\\8.1.0\\include\\stddef.h" 3 4

# 149 "c:\\users\\trall\\clionprojects\\riscv64-unknown-elf-gcc-2018.07.0-x86\_64-w64-mingw32\\lib\\gcc\\riscv64-unknown-elf\\8.1.0\\include\\stddef.h" 3 4

typedef long int ptrdiff\_t;

# 216 "c:\\users\\trall\\clionprojects\\riscv64-unknown-elf-gcc-2018.07.0-x86\_64-w64-mingw32\\lib\\gcc\\riscv64-unknown-elf\\8.1.0\\include\\stddef.h" 3 4

typedef long unsigned int size\_t;

# 328 "c:\\users\\trall\\clionprojects\\riscv64-unknown-elf-gcc-2018.07.0-x86\_64-w64-mingw32\\lib\\gcc\\riscv64-unknown-elf\\8.1.0\\include\\stddef.h" 3 4

typedef int wchar\_t;

# 426 "c:\\users\\trall\\clionprojects\\riscv64-unknown-elf-gcc-2018.07.0-x86\_64-w64-mingw32\\lib\\gcc\\riscv64-unknown-elf\\8.1.0\\include\\stddef.h" 3 4

typedef struct {

long long \_\_max\_align\_ll \_\_attribute\_\_((\_\_aligned\_\_(\_\_alignof\_\_(long long))));

long double \_\_max\_align\_ld \_\_attribute\_\_((\_\_aligned\_\_(\_\_alignof\_\_(long double))));

# 437 "c:\\users\\trall\\clionprojects\\riscv64-unknown-elf-gcc-2018.07.0-x86\_64-w64-mingw32\\lib\\gcc\\riscv64-unknown-elf\\8.1.0\\include\\stddef.h" 3 4

} max\_align\_t;

# 2 "find\_koeffs.h" 2

# 2 "find\_koeffs.h"

int find\_koeffs(int param, int \*array);

# 2 "find\_koeffs.c" 2

int find\_koeffs(int param, int \*array)

{

array[0] = 1;

array[1] = 1;

int koeff1 = 0;

int koeff2;

for (int i = 0; i < param; i++)

{

for (int j = 0; j <= i; j++)

{

koeff2 = array[j] + array[j + 1];

if (j != 0)

{

array[j] = koeff1;

}

koeff1 = koeff2;

}

array[i + 1] = 1;

}

return 0;

}

В листинге видно, что препроцессор включил содержимое заголовочных файлов. В свою очередь препроцессор для файла main.с сделает то же самое, однако в связи с использованием нескольких больших заголовочных файлов из стандартных библиотек размер результата работы препроцессора будет намного больше.

Затем при помощи команды gcc -c find\_koeffs.c был получен объектный файл find\_koeffs.o и аналогично при помощи gcc -c main.c был получен объектный файл main.o. После этого при помощи команды «nm» получим список символов для данных объектных файлов.

Результат для find\_koeffs.o:

00000000000000ae t .L2

0000000000000082 t .L3

0000000000000070 t .L4

0000000000000032 t .L5

000000000000002c t .L6

0000000000000000 T find\_koeffs  
Результат для main.o:

000000000000006a t .L2

0000000000000040 t .L3

0000000000000000 r .LC0

U calloc

U find\_koeffs

0000000000000000 T main

U printf

Создадим объектные файлы заново вместе с отладочной информацией при помощи команды  
 gcc -g -c find\_koeffs.c && gcc -g -c main.c

Затем мы сможем рассмотреть результат результат компиляции в виде языка ассемблера. Для этого воспользуемся командами:

objdump -d -S find\_koeffs.o

objdump -d -S main.o

Результат для find\_koeffs.o на языке ассемблера:

find\_koeffs.o: file format elf64-littleriscv

Disassembly of section .text:

0000000000000000 <find\_koeffs>:

#include "find\_koeffs.h"

int find\_koeffs(int param, int \*array)

{

0: 7179 addi sp,sp,-48

2: f422 sd s0,40(sp)

4: 1800 addi s0,sp,48

0000000000000006 <.L0 >:

6: 87aa mv a5,a0

8: fcb43823 sd a1,-48(s0)

c: fcf42e23 sw a5,-36(s0)

0000000000000010 <.L0 >:

array[0] = 1;

10: fd043783 ld a5,-48(s0)

14: 4705 li a4,1

16: c398 sw a4,0(a5)

0000000000000018 <.L0 >:

array[1] = 1;

18: fd043783 ld a5,-48(s0)

1c: 0791 addi a5,a5,4

000000000000001e <.L0 >:

1e: 4705 li a4,1

20: c398 sw a4,0(a5)

0000000000000022 <.L0 >:

int koeff1 = 0;

22: fe042623 sw zero,-20(s0)

0000000000000026 <.L0 >:

int koeff2;

for (int i = 0; i < param; i++)

26: fe042423 sw zero,-24(s0)

000000000000002a <.L0 >:

2a: a051 j ae <.L0 >

000000000000002c <.L0 >:

{

for (int j = 0; j <= i; j++)

2c: fe042223 sw zero,-28(s0)

0000000000000030 <.L0 >:

30: a889 j 82 <.L0 >

0000000000000032 <.L0 >:

{

koeff2 = array[j] + array[j + 1];

32: fe442783 lw a5,-28(s0)

36: 078a slli a5,a5,0x2

38: fd043703 ld a4,-48(s0)

3c: 97ba add a5,a5,a4

3e: 4398 lw a4,0(a5)

0000000000000040 <.L0 >:

40: fe442783 lw a5,-28(s0)

44: 0785 addi a5,a5,1

46: 078a slli a5,a5,0x2

48: fd043683 ld a3,-48(s0)

4c: 97b6 add a5,a5,a3

4e: 439c lw a5,0(a5)

0000000000000050 <.L0 >:

50: 9fb9 addw a5,a5,a4

52: fef42023 sw a5,-32(s0)

0000000000000056 <.L0 >:

if (j != 0)

56: fe442783 lw a5,-28(s0)

5a: 2781 sext.w a5,a5

5c: cb91 beqz a5,70 <.L0 >

000000000000005e <.L0 >:

{

array[j] = koeff1;

5e: fe442783 lw a5,-28(s0)

62: 078a slli a5,a5,0x2

64: fd043703 ld a4,-48(s0)

68: 97ba add a5,a5,a4

000000000000006a <.L0 >:

6a: fec42703 lw a4,-20(s0)

6e: c398 sw a4,0(a5)

0000000000000070 <.L0 >:

}

koeff1 = koeff2;

70: fe042783 lw a5,-32(s0)

74: fef42623 sw a5,-20(s0)

0000000000000078 <.L0 >:

for (int j = 0; j <= i; j++)

78: fe442783 lw a5,-28(s0)

7c: 2785 addiw a5,a5,1

7e: fef42223 sw a5,-28(s0)

0000000000000082 <.L0 >:

82: fe442703 lw a4,-28(s0)

86: fe842783 lw a5,-24(s0)

8a: 2701 sext.w a4,a4

8c: 2781 sext.w a5,a5

8e: fae7d2e3 ble a4,a5,32 <.L0 >

0000000000000092 <.L0 >:

}

array[i + 1] = 1;

92: fe842783 lw a5,-24(s0)

96: 0785 addi a5,a5,1

98: 078a slli a5,a5,0x2

9a: fd043703 ld a4,-48(s0)

9e: 97ba add a5,a5,a4

00000000000000a0 <.L0 >:

a0: 4705 li a4,1

a2: c398 sw a4,0(a5)

00000000000000a4 <.L0 >:

for (int i = 0; i < param; i++)

a4: fe842783 lw a5,-24(s0)

a8: 2785 addiw a5,a5,1

aa: fef42423 sw a5,-24(s0)

00000000000000ae <.L0 >:

ae: fe842703 lw a4,-24(s0)

b2: fdc42783 lw a5,-36(s0)

b6: 2701 sext.w a4,a4

b8: 2781 sext.w a5,a5

ba: f6f749e3 blt a4,a5,2c <.L0 >

00000000000000be <.L0 >:

}

return 0;

be: 4781 li a5,0

00000000000000c0 <.L0 >:

c0: 853e mv a0,a5

c2: 7422 ld s0,40(sp)

00000000000000c4 <.L0 >:

c4: 6145 addi sp,sp,48

c6: 8082 ret  
  
Результат для main.o на языке ассемблера:

main.o: file format elf64-littleriscv

Disassembly of section .text:

0000000000000000 <main>:

#include <stdio.h>

#include <stdlib.h>

#include "find\_koeffs.h"

int main()

{

0: 1101 addi sp,sp,-32

2: ec06 sd ra,24(sp)

4: e822 sd s0,16(sp)

6: 1000 addi s0,sp,32

0000000000000008 <.L0 >:

int parameter = 5;

8: 4795 li a5,5

a: fef42423 sw a5,-24(s0)

000000000000000e <.L0 >:

int \*array = (int \*) calloc(parameter + 1, sizeof(int));

e: fe842783 lw a5,-24(s0)

12: 2785 addiw a5,a5,1

14: 2781 sext.w a5,a5

0000000000000016 <.L0 >:

16: 4591 li a1,4

18: 853e mv a0,a5

1a: 00000097 auipc ra,0x0

1e: 000080e7 jalr ra # 1a <.L0 +0x4>

22: 87aa mv a5,a0

24: fef43023 sd a5,-32(s0)

0000000000000028 <.L0 >:

find\_koeffs(parameter, array);

28: fe842783 lw a5,-24(s0)

2c: fe043583 ld a1,-32(s0)

30: 853e mv a0,a5

32: 00000097 auipc ra,0x0

36: 000080e7 jalr ra # 32 <.L0 +0xa>

000000000000003a <.L0 >:

for (int i = 0; i <= parameter; i++)

3a: fe042623 sw zero,-20(s0)

000000000000003e <.L0 >:

3e: a035 j 6a <.L0 >

0000000000000040 <.L0 >:

{

printf("%i \n", array[i]);

40: fec42783 lw a5,-20(s0)

44: 078a slli a5,a5,0x2

46: fe043703 ld a4,-32(s0)

4a: 97ba add a5,a5,a4

000000000000004c <.L0 >:

4c: 439c lw a5,0(a5)

4e: 85be mv a1,a5

50: 000007b7 lui a5,0x0

54: 00078513 mv a0,a5

58: 00000097 auipc ra,0x0

5c: 000080e7 jalr ra # 58 <.L0 +0xc>

0000000000000060 <.L0 >:

for (int i = 0; i <= parameter; i++)

60: fec42783 lw a5,-20(s0)

64: 2785 addiw a5,a5,1

66: fef42623 sw a5,-20(s0)

000000000000006a <.L0 >:

6a: fec42703 lw a4,-20(s0)

6e: fe842783 lw a5,-24(s0)

72: 2701 sext.w a4,a4

74: 2781 sext.w a5,a5

76: fce7d5e3 ble a4,a5,40 <.L0 >

000000000000007a <.L0 >:

}

return 0;

7a: 4781 li a5,0

000000000000007c <.L0 >:

7c: 853e mv a0,a5

7e: 60e2 ld ra,24(sp)

0000000000000080 <.L0 >:

80: 6442 ld s0,16(sp)

82: 6105 addi sp,sp,32

84: 8082 ret

После этого необходимо запустить линоковщик, который соместит библиотеки и объектные файлы в один исполняемый файл. Для этого выполним команду

gcc main.o find\_koeffs.o -o main

Так же оформим подпрограмму find\_koeffs в качестве статически линкуемой библиотеки при помощи

riscv64-unknown-elf-ar –rsc koeffs.a find\_koeffs.o

**Вывод**

В ходе выполнения лабораторной работы была разработана программа на языке С, выполняющая расчет биномиальных коэффициентов для данного показателя по треугольнику Паскаля, так же была выполнена раздельная компиляция данной программы и создана статическая библиотека.