Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Лабораторная работа № 4**

Раздельная компиляция

по дисциплине «Низкоуровневое программирование»

Выполнил

студент гр. 3530901/90004

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шомов М.Ю.

(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.

Согласно условию варианта 19 необходимо реализовать циклический сдвиг элементов массива.

**Реализация**

Код на языке С, реализующий данный функционал представлен файлами:  
Файл main.c  
#include <stdio.h>

#include "shiftArr.h"

int main( )

{

    int shift = 7;

    int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

    shiftArr(shift, a);

    return 0;

}

Файл shiftArr.h

#include <stddef.h>

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

Файл shiftArr.c

#include "shiftArr.h"

int shiftArr(int shift, int \*a)

{

    int n = 10;

    while (shift >= n){

        shift -= n;

    }

    while (shift > 0) {

        int temp = a[9];

        int num = 0;

        int tmp = a[num];

        while (num < n) {

            int tmp2 = a[num+1];

            a[num+1] = tmp;

            tmp = tmp2;

            num++;

        }

        a[0] = temp;

        shift--;

    }

    return 0;

}

**Подготовительный этап. Настройка среды**

Рисунком 1 представлены настройки переменных сред, выполненные перед осуществлением работы.

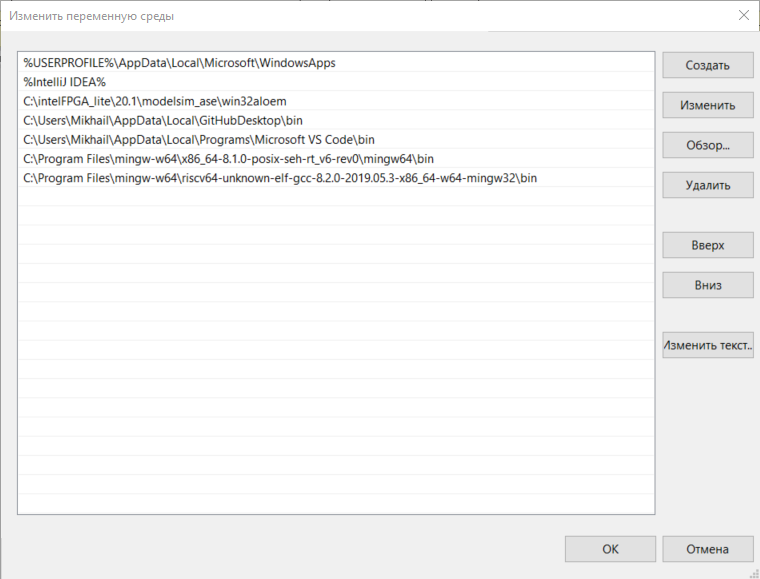


Рисунок 1 Настройки переменных сред

**Сборка программы**

Запуск препроцессора выполнен при помощи команды:

riscv64-unknown-elf-gcc -E -v main.c -o main.i

Содержимое файлов результата препроцессирования main.i

# 1 "main.c"

# 1 "<built-in>"

# 1 "<command-line>"

# 1 "main.c"

# 1 "shiftArr.h" 1

# 1 "c:\\program files\\mingw-w64\\riscv64-unknown-elf-gcc-8.2.0-2019.05.3-x86\_64-w64-mingw32\\lib\\gcc\\riscv64-unknown-elf\\8.2.0\\include\\stddef.h" 1 3 4

# 149 "c:\\program files\\mingw-w64\\riscv64-unknown-elf-gcc-8.2.0-2019.05.3-x86\_64-w64-mingw32\\lib\\gcc\\riscv64-unknown-elf\\8.2.0\\include\\stddef.h" 3 4

# 149 "c:\\program files\\mingw-w64\\riscv64-unknown-elf-gcc-8.2.0-2019.05.3-x86\_64-w64-mingw32\\lib\\gcc\\riscv64-unknown-elf\\8.2.0\\include\\stddef.h" 3 4

typedef long int ptrdiff\_t;

# 216 "c:\\program files\\mingw-w64\\riscv64-unknown-elf-gcc-8.2.0-2019.05.3-x86\_64-w64-mingw32\\lib\\gcc\\riscv64-unknown-elf\\8.2.0\\include\\stddef.h" 3 4

typedef long unsigned int size\_t;

# 328 "c:\\program files\\mingw-w64\\riscv64-unknown-elf-gcc-8.2.0-2019.05.3-x86\_64-w64-mingw32\\lib\\gcc\\riscv64-unknown-elf\\8.2.0\\include\\stddef.h" 3 4

typedef int wchar\_t;

# 426 "c:\\program files\\mingw-w64\\riscv64-unknown-elf-gcc-8.2.0-2019.05.3-x86\_64-w64-mingw32\\lib\\gcc\\riscv64-unknown-elf\\8.2.0\\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:\\program files\\mingw-w64\\riscv64-unknown-elf-gcc-8.2.0-2019.05.3-x86\_64-w64-mingw32\\lib\\gcc\\riscv64-unknown-elf\\8.2.0\\include\\stddef.h" 3 4

} max\_align\_t;

# 2 "shiftArr.h" 2

# 2 "shiftArr.h"

int shiftArr(int shift, int length, int \*a);

# 2 "main.c" 2

int main( )

{

    int shift = 2;

    int a[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

    int length = sizeof(a) / sizeof(int);

    shiftArr(shift, length, a);

    return 0;

}

riscv64-unknown-elf-gcc -E -v shiftArr.c -o shiftArr.i

Содержимое shiftArr.i

# 1 "shiftArr.c"

# 1 "<built-in>"

# 1 "<command-line>"

# 1 "shiftArr.c"

# 1 "shiftArr.h" 1

# 1 "c:\\program files\\mingw-w64\\riscv64-unknown-elf-gcc-8.2.0-2019.05.3-x86\_64-w64-mingw32\\lib\\gcc\\riscv64-unknown-elf\\8.2.0\\include\\stddef.h" 1 3 4

# 149 "c:\\program files\\mingw-w64\\riscv64-unknown-elf-gcc-8.2.0-2019.05.3-x86\_64-w64-mingw32\\lib\\gcc\\riscv64-unknown-elf\\8.2.0\\include\\stddef.h" 3 4

# 149 "c:\\program files\\mingw-w64\\riscv64-unknown-elf-gcc-8.2.0-2019.05.3-x86\_64-w64-mingw32\\lib\\gcc\\riscv64-unknown-elf\\8.2.0\\include\\stddef.h" 3 4

typedef long int ptrdiff\_t;

# 216 "c:\\program files\\mingw-w64\\riscv64-unknown-elf-gcc-8.2.0-2019.05.3-x86\_64-w64-mingw32\\lib\\gcc\\riscv64-unknown-elf\\8.2.0\\include\\stddef.h" 3 4

typedef long unsigned int size\_t;

# 328 "c:\\program files\\mingw-w64\\riscv64-unknown-elf-gcc-8.2.0-2019.05.3-x86\_64-w64-mingw32\\lib\\gcc\\riscv64-unknown-elf\\8.2.0\\include\\stddef.h" 3 4

typedef int wchar\_t;

# 426 "c:\\program files\\mingw-w64\\riscv64-unknown-elf-gcc-8.2.0-2019.05.3-x86\_64-w64-mingw32\\lib\\gcc\\riscv64-unknown-elf\\8.2.0\\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:\\program files\\mingw-w64\\riscv64-unknown-elf-gcc-8.2.0-2019.05.3-x86\_64-w64-mingw32\\lib\\gcc\\riscv64-unknown-elf\\8.2.0\\include\\stddef.h" 3 4

} max\_align\_t;

# 2 "shiftArr.h" 2

# 2 "shiftArr.h"

int shiftArr(int shift, int length, int \*a);

# 2 "shiftArr.c" 2

int shiftArr(int shift, int length, int \*a)

{

    while (shift >= length){

        shift -= length;

    }

    int tmp;

    int tmp2;

    int temp;

    int num;

    while (shift > 0) {

        temp = a[length - 1];

        num = 0;

        tmp = a[num];

        while (num < length) {

            tmp2 = a[num+1];

            a[num+1] = tmp;

            tmp = tmp2;

            num++;

        }

        a[0] = temp;

        shift--;

    }

    return 0;

}

Получение объектного файла для main.c и shiftArr.c:

riscv64-unknown-elf-gcc -g -c main.c && riscv64-unknown-elf-gcc -g -c shiftArr.c

Поскольку объектный файл является бинарным, для его изучения потребуется использовать программу objdump

Вывод команды

riscv64-unknown-elf-objdump-f main.o

*«main.o: file format elf64-littleriscv*

*architecture: riscv:rv64, flags 0x00000011:*

*HAS\_RELOC, HAS\_SYMS*

*start address 0x0000000000000000»*

Как известно, содержательная часть объектного файла разбита на «разделы», называемые обычно секциями. Следующая команда обеспечивает отображение заголовков секций файла “main.o”

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

«*Sections:*

*Idx Name Size VMA LMA File off Algn*

*0 .text 0000006e 0000000000000000 0000000000000000 00000040 2\*\*1*

*CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE*

*1 .data 00000000 0000000000000000 0000000000000000 000000ae 2\*\*0*

*CONTENTS, ALLOC, LOAD, DATA*

*2 .bss 00000000 0000000000000000 0000000000000000 000000ae 2\*\*0*

*ALLOC*

*3 .rodata 00000028 0000000000000000 0000000000000000 000000b0 2\*\*3*

*CONTENTS, ALLOC, LOAD, READONLY, DATA*

*4 .debug\_info 000000ab 0000000000000000 0000000000000000 000000d8 2\*\*0*

*CONTENTS, RELOC, READONLY, DEBUGGING*

*5 .debug\_abbrev 00000078 0000000000000000 0000000000000000 00000183 2\*\*0*

*CONTENTS, READONLY, DEBUGGING*

*6 .debug\_aranges 00000030 0000000000000000 0000000000000000 000001fb 2\*\*0*

*CONTENTS, RELOC, READONLY, DEBUGGING*

*7 .debug\_line 00000067 0000000000000000 0000000000000000 0000022b 2\*\*0*

*CONTENTS, RELOC, READONLY, DEBUGGING*

*8 .debug\_str 00000093 0000000000000000 0000000000000000 00000292 2\*\*0*

*CONTENTS, READONLY, DEBUGGING*

*9 .comment 00000029 0000000000000000 0000000000000000 00000325 2\*\*0*

*CONTENTS, READONLY*

*10 .debug\_frame 00000040 0000000000000000 0000000000000000 00000350 2\*\*3*

*CONTENTS, RELOC, READONLY, DEBUGGING*

*11 .riscv.attributes 00000033 0000000000000000 0000000000000000 00000390 2\*\*0*

*CONTENTS, READONLY»*

В файле “main.o” имеются следующие секции:

* text – секция кода, в которой содержатся коды инструкций (название секции обусловлено историческими причинами);
* data – секция инициализированных данных;
* bss – секция данных, инициализированных нулями(название секции также обусловлено историческими причинами);
* comment – секция данных о версиях размером 12 байт.

Командой objdump с флагом -t получаем список символов для объектных файлов

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

“*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 0000000000000000 .rodata*

*0000000000000000 l .text 0000000000000000 .L0*

*0000000000000000 l .text 0000000000000000 .L0*

*0000000000000008 l .text 0000000000000000 .L0*

*0000000000000008 l .text 0000000000000000 .L0*

*000000000000000e l .text 0000000000000000 .L0*

*0000000000000042 l .text 0000000000000000 .L0*

*0000000000000048 l .text 0000000000000000 .L0*

*0000000000000062 l .text 0000000000000000 .L0*

*0000000000000064 l .text 0000000000000000 .L0*

*0000000000000068 l .text 0000000000000000 .L0*

*000000000000006e l .text 0000000000000000 .L0*

*0000000000000000 l d .debug\_info 0000000000000000 .debug\_info*

*0000000000000000 l d .debug\_abbrev 0000000000000000 .debug\_abbrev*

*0000000000000000 l d .debug\_aranges 0000000000000000 .debug\_aranges*

*0000000000000000 l d .debug\_line 0000000000000000 .debug\_line*

*0000000000000000 l d .debug\_str 0000000000000000 .debug\_str*

*000000000000006e l .text 0000000000000000 .L0*

*0000000000000000 l .debug\_frame 0000000000000000 .L0*

*0000000000000000 l .rodata 0000000000000000 .LC0*

*0000000000000000 l .debug\_abbrev 0000000000000000 .Ldebug\_abbrev0*

*000000000000001e l .debug\_str 0000000000000000 .LASF6*

*0000000000000054 l .debug\_str 0000000000000000 .LASF7*

*0000000000000064 l .debug\_str 0000000000000000 .LASF8*

*0000000000000000 l .text 0000000000000000 .Ltext0*

*000000000000006e l .text 0000000000000000 .Letext0*

*0000000000000000 l .debug\_line 0000000000000000 .Ldebug\_line0*

*000000000000005b l .debug\_str 0000000000000000 .LASF0*

*000000000000000c l .debug\_str 0000000000000000 .LASF1*

*0000000000000085 l .debug\_str 0000000000000000 .LASF2*

*0000000000000000 l .debug\_str 0000000000000000 .LASF3*

*000000000000007a l .debug\_str 0000000000000000 .LASF9*

*0000000000000000 l .text 0000000000000000 .LFB0*

*000000000000006e l .text 0000000000000000 .LFE0*

*000000000000007f l .debug\_str 0000000000000000 .LASF4*

*000000000000004d l .debug\_str 0000000000000000 .LASF5*

*0000000000000000 l .debug\_info 0000000000000000 .Ldebug\_info0*

*0000000000000000 l d .comment 0000000000000000 .comment*

*0000000000000000 l d .debug\_frame 0000000000000000 .debug\_frame*

*0000000000000000 l d .riscv.attributes 0000000000000000 .riscv.attributes*

*0000000000000000 g F .text 000000000000006e main*

*0000000000000000 \*UND\* 0000000000000000 shiftArr*”

\*UND\* значит, что символ использовался в ассемблерном коде, из которого был получен объектный файл, но не был определён, из чего был сделал вывод, что он определяется где-то в другом месте

riscv64-unknown-elf-objdump -t shiftArr.o

“SYMBOL TABLE:

0000000000000000 l df \*ABS\* 0000000000000000 shiftArr.c

0000000000000000 l d .text 0000000000000000 .text

0000000000000000 l d .data 0000000000000000 .data

0000000000000000 l d .bss 0000000000000000 .bss

0000000000000000 l .text 0000000000000000 .L0

0000000000000000 l .text 0000000000000000 .L0

0000000000000006 l .text 0000000000000000 .L0

0000000000000018 l .text 0000000000000000 .L0

0000000000000020 l .text 0000000000000000 .L0

0000000000000022 l .text 0000000000000000 .L0

0000000000000032 l .text 0000000000000000 .L0

0000000000000042 l .text 0000000000000000 .L0

0000000000000044 l .text 0000000000000000 .L0

0000000000000052 l .text 0000000000000000 .L0

0000000000000058 l .text 0000000000000000 .L0

000000000000005c l .text 0000000000000000 .L0

0000000000000068 l .text 0000000000000000 .L0

000000000000006e l .text 0000000000000000 .L0

0000000000000070 l .text 0000000000000000 .L0

000000000000007e l .text 0000000000000000 .L0

0000000000000084 l .text 0000000000000000 .L0

0000000000000092 l .text 0000000000000000 .L0

0000000000000098 l .text 0000000000000000 .L0

00000000000000a0 l .text 0000000000000000 .L0

00000000000000aa l .text 0000000000000000 .L0

00000000000000ba l .text 0000000000000000 .L0

00000000000000c4 l .text 0000000000000000 .L0

00000000000000ce l .text 0000000000000000 .L0

00000000000000d8 l .text 0000000000000000 .L0

00000000000000da l .text 0000000000000000 .L0

00000000000000de l .text 0000000000000000 .L0

00000000000000e2 l .text 0000000000000000 .L0

0000000000000000 l d .debug\_info 0000000000000000 .debug\_info

0000000000000000 l d .debug\_abbrev 0000000000000000 .debug\_abbrev

0000000000000000 l d .debug\_aranges 0000000000000000 .debug\_aranges

0000000000000000 l d .debug\_line 0000000000000000 .debug\_line

0000000000000000 l d .debug\_str 0000000000000000 .debug\_str

00000000000000e2 l .text 0000000000000000 .L0

0000000000000000 l .debug\_frame 0000000000000000 .L0

0000000000000032 l .text 0000000000000000 .L2

0000000000000022 l .text 0000000000000000 .L3

00000000000000ce l .text 0000000000000000 .L4

00000000000000aa l .text 0000000000000000 .L5

0000000000000070 l .text 0000000000000000 .L6

0000000000000044 l .text 0000000000000000 .L7

0000000000000000 l .debug\_abbrev 0000000000000000 .Ldebug\_abbrev0

0000000000000030 l .debug\_str 0000000000000000 .LASF8

0000000000000070 l .debug\_str 0000000000000000 .LASF9

000000000000007b l .debug\_str 0000000000000000 .LASF10

0000000000000000 l .text 0000000000000000 .Ltext0

00000000000000e2 l .text 0000000000000000 .Letext0

0000000000000000 l .debug\_line 0000000000000000 .Ldebug\_line0

0000000000000027 l .debug\_str 0000000000000000 .LASF0

0000000000000015 l .debug\_str 0000000000000000 .LASF1

0000000000000097 l .debug\_str 0000000000000000 .LASF2

0000000000000009 l .debug\_str 0000000000000000 .LASF3

0000000000000000 l .debug\_str 0000000000000000 .LASF11

0000000000000000 l .text 0000000000000000 .LFB0

00000000000000e2 l .text 0000000000000000 .LFE0

0000000000000091 l .debug\_str 0000000000000000 .LASF4

000000000000005f l .debug\_str 0000000000000000 .LASF5

000000000000006b l .debug\_str 0000000000000000 .LASF6

0000000000000066 l .debug\_str 0000000000000000 .LASF7

0000000000000000 l .debug\_info 0000000000000000 .Ldebug\_info0

0000000000000000 l d .comment 0000000000000000 .comment

0000000000000000 l d .debug\_frame 0000000000000000 .debug\_frame

0000000000000000 l d .riscv.attributes 0000000000000000 .riscv.attributes

0000000000000000 g F .text 00000000000000e2 shiftArr”

Результат компиляции на языке ассемблера можно изучить при помощи команды objdump.

riscv64-unknown-elf-objdump -d -S main.o

“main.o: file format elf64-littleriscv

Disassembly of section .text:

0000000000000000 <main>:

#include "shiftArr.h"

int main( )

{

0: 7139 addi sp,sp,-64

2: fc06 sd ra,56(sp)

4: f822 sd s0,48(sp)

6: 0080 addi s0,sp,64

int shift = 7;

8: 479d li a5,7

a: fef42623 sw a5,-20(s0)

int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

e: 000007b7 lui a5,0x0

12: 0007b583 ld a1,0(a5) # 0 <main>

16: 00078713 mv a4,a5

1a: 6710 ld a2,8(a4)

1c: 00078713 mv a4,a5

20: 6b14 ld a3,16(a4)

22: 00078713 mv a4,a5

26: 6f18 ld a4,24(a4)

28: 00078793 mv a5,a5

2c: 739c ld a5,32(a5)

2e: fcb43023 sd a1,-64(s0)

32: fcc43423 sd a2,-56(s0)

36: fcd43823 sd a3,-48(s0)

3a: fce43c23 sd a4,-40(s0)

3e: fef43023 sd a5,-32(s0)

int length = 10;

42: 47a9 li a5,10

44: fef42423 sw a5,-24(s0)

shiftArr(shift, length, a);

48: fc040693 addi a3,s0,-64

4c: fe842703 lw a4,-24(s0)

50: fec42783 lw a5,-20(s0)

54: 8636 mv a2,a3

56: 85ba mv a1,a4

58: 853e mv a0,a5

5a: 00000097 auipc ra,0x0

5e: 000080e7 jalr ra # 5a <main+0x5a>

return 0;

62: 4781 li a5,0

}

64: 853e mv a0,a5

66: 70e2 ld ra,56(sp)

68: 7442 ld s0,48(sp)

6a: 6121 addi sp,sp,64

6c: 8082 ret”

riscv64-unknown-elf-objdump -d -S shiftArr.o

“*shiftArr.o: file format elf64-littleriscv*

*Disassembly of section .text:*

*0000000000000000 <shiftArr>:*

*#include "shiftArr.h"*

*int shiftArr(int shift, int length, int \*a)*

*{*

*0: 7139 addi sp,sp,-64*

*2: fc22 sd s0,56(sp)*

*4: 0080 addi s0,sp,64*

*6: 87aa mv a5,a0*

*8: 872e mv a4,a1*

*a: fcc43023 sd a2,-64(s0)*

*e: fcf42623 sw a5,-52(s0)*

*12: 87ba mv a5,a4*

*14: fcf42423 sw a5,-56(s0)*

*int n = length;*

*18: fc842783 lw a5,-56(s0)*

*1c: fef42223 sw a5,-28(s0)*

*while (shift >= n){*

*20: a809 j 32 <.L2>*

*0000000000000022 <.L3>:*

*shift -= n;*

*22: fcc42703 lw a4,-52(s0)*

*26: fe442783 lw a5,-28(s0)*

*2a: 40f707bb subw a5,a4,a5*

*2e: fcf42623 sw a5,-52(s0)*

*0000000000000032 <.L2>:*

*while (shift >= n){*

*32: fcc42703 lw a4,-52(s0)*

*36: fe442783 lw a5,-28(s0)*

*3a: 2701 sext.w a4,a4*

*3c: 2781 sext.w a5,a5*

*3e: fef752e3 bge a4,a5,22 <.L3>*

*}*

*int tmp;*

*int tmp2;*

*int temp;*

*int num;*

*while (shift > 0) {*

*42: a071 j ce <.L4>*

*0000000000000044 <.L7>:*

*temp = a[length - 1];*

*44: fc842783 lw a5,-56(s0)*

*48: 078a slli a5,a5,0x2*

*4a: 17f1 addi a5,a5,-4*

*4c: fc043703 ld a4,-64(s0)*

*50: 97ba add a5,a5,a4*

*52: 439c lw a5,0(a5)*

*54: fef42023 sw a5,-32(s0)*

*num = 0;*

*58: fe042423 sw zero,-24(s0)*

*tmp = a[num];*

*5c: fe842783 lw a5,-24(s0)*

*60: 078a slli a5,a5,0x2*

*62: fc043703 ld a4,-64(s0)*

*66: 97ba add a5,a5,a4*

*68: 439c lw a5,0(a5)*

*6a: fef42623 sw a5,-20(s0)*

*while (num < n) {*

*6e: a835 j aa <.L5>*

*0000000000000070 <.L6>:*

*tmp2 = a[num+1];*

*70: fe842783 lw a5,-24(s0)*

*74: 0785 addi a5,a5,1*

*76: 078a slli a5,a5,0x2*

*78: fc043703 ld a4,-64(s0)*

*7c: 97ba add a5,a5,a4*

*7e: 439c lw a5,0(a5)*

*80: fcf42e23 sw a5,-36(s0)*

*a[num+1] = tmp;*

*84: fe842783 lw a5,-24(s0)*

*88: 0785 addi a5,a5,1*

*8a: 078a slli a5,a5,0x2*

*8c: fc043703 ld a4,-64(s0)*

*90: 97ba add a5,a5,a4*

*92: fec42703 lw a4,-20(s0)*

*96: c398 sw a4,0(a5)*

*tmp = tmp2;*

*98: fdc42783 lw a5,-36(s0)*

*9c: fef42623 sw a5,-20(s0)*

*num++;*

*a0: fe842783 lw a5,-24(s0)*

*a4: 2785 addiw a5,a5,1*

*a6: fef42423 sw a5,-24(s0)*

*00000000000000aa <.L5>:*

*while (num < n) {*

*aa: fe842703 lw a4,-24(s0)*

*ae: fe442783 lw a5,-28(s0)*

*b2: 2701 sext.w a4,a4*

*b4: 2781 sext.w a5,a5*

*b6: faf74de3 blt a4,a5,70 <.L6>*

*}*

*a[0] = temp;*

*ba: fc043783 ld a5,-64(s0)*

*be: fe042703 lw a4,-32(s0)*

*c2: c398 sw a4,0(a5)*

*shift--;*

*c4: fcc42783 lw a5,-52(s0)*

*c8: 37fd addiw a5,a5,-1*

*ca: fcf42623 sw a5,-52(s0)*

*00000000000000ce <.L4>:*

*while (shift > 0) {*

*ce: fcc42783 lw a5,-52(s0)*

*d2: 2781 sext.w a5,a5*

*d4: f6f048e3 bgtz a5,44 <.L7>*

*}*

*return 0;*

*d8: 4781 li a5,0*

*da: 853e mv a0,a5*

*dc: 7462 ld s0,56(sp)*

*de: 6121 addi sp,sp,64*

*e0: 8082 ret*”

Создаётся статическая линкуемая библиотека:

riscv64-unknown-elf-ar –rsc shiftArr.a shiftArr.o

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

riscv64-unknown-elf-gcc main.o libshiftArr.a -o main

Чтобы автоматизировать процесс сборки создан makefile

all: lib

lib: shiftArr.o

    riscv64-unknown-elf-ar -rsc libshiftArr.a shiftArr.o

shiftArr.o: shiftArr.c

    riscv64-unknown-elf-gcc -g -c shiftArr.c

main.o: main.c

    riscv64-unknown-elf-gcc -g -c main.c

all:

    riscv64-unknown-elf-gcc main.o libshiftArr.a -o main