

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

Выполнил студент гр. 3530901/90003

\_\_\_\_\_  
(подпись) Бехтольд Ек.В.

Преподаватель

\_\_\_\_\_  
(подпись) Алексюк А. О.

“ \_\_\_\_ ” \_\_\_\_\_ 2021 г.

Санкт-Петербург

2021

## Оглавление

1 Техническое задание.....	3
2 Метод решения.....	3
3 Решение.....	3
3.1 Анализ выхода препроцессора:.....	5
3.2 Анализ выхода компилятора:.....	6
3.3 Анализ состава и содержимого секций, таблицы символов, таблицы перемещений и отладочной информации, содержащейся в объектных файлах и исполняемом файле.....	8
3.4 Содержимое таблицы перемещений:.....	14
3.5 Результат компоновки.....	19
3.6 Анализ отладочной информации.....	21
3.7 Выделение разработанной функции в статическую библиотеку.....	22
3.8 Создание и использование полученной статической библиотеки.....	23
Рис.12. Вывод результата.....	24
4 Результаты.....	24

## 1 Техническое задание

- 1 На языке C разработать функцию, реализующую сдвиг в массиве чисел на заданное количество разрядов влево. 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.

## 2 Метод решения

Для реализации данной задачи мы будем брать элемент массива из ячейки  $n$  и записывать его в предыдущую ячейку  $n-1$ , предварительно первый элемент поместив в рабочую переменную. На место последнего элемента положим элемент хранящийся в рабочей переменной. Эта процедура будет продолжаться  $k$ - раз, где  $k$  — количество сдвигов.

## 3 Решение

Напишем программу на языке C:

Листинг 1: файл shiftArray.h

```
#ifndef LOWLEVEL_SHIFTARRAY_H
#define LOWLEVEL_SHIFTARRAY_H
#include <stdio.h>
void shiftArray(int array[], int shift, int length);
#endif // LOWLEVEL_SHIFTARRAY_H
```

```
#include "shiftArray.h"

void shiftArray(int array[], int shift, int length) {
    for (int j = 0; j < shift; ++j) {
        int tmp = array[0];
        for (int i = 0; i < length - 1; ++i) {
            array[i] = array[i + 1];
        }
        array[length - 1] = tmp;
    }
}
```

```
#include "shiftArray.h"

int main() {
    int array[3] = {1, 2, 3};
    const int shift = 2;
    int length = sizeof(array) / sizeof(int);
    printf("Start Array: [");
    for(int i = 0; i < length; i++) {
        printf("%d", array[i]);
        if (i != length - 1) printf(", ");
    }
    printf("]\nshift=%d\n", shift);
    printf("result=%d", shiftArray(array, length, shift));
    printf("End Array: [");
    for(int I = 0; I < length; ++i) {
        printf("%d", array[i]);
    }
    printf("]");
    return 0;
}
```

```
/shift_array$ riscv32-unknown-elf-gcc -O1 -E main.c -o main.i
/shift_array$ riscv32-unknown-elf-gcc -O1 -E shiftArray.c -o shiftArray.i
/shift_array$ riscv32-unknown-elf-gcc -O1 -S shiftArray.i -o shiftArray.s
/shift_array$ riscv32-unknown-elf-gcc -O1 -S main.i -o main.s
/shift_array$ riscv32-unknown-elf-gcc -c main.s -o main.o
/shift_array$ riscv32-unknown-elf-gcc -c shiftArray.s -o shiftArray.o
/shift_array$ riscv32-unknown-elf-gcc main.o shiftArray.o -o main
/shift_array$
```

Рис.1. Сборка программы по этапам

### 3.1 Анализ выхода препроцессора:

Директивы, прописанные в заголовочном файле, определяют вставку стандартной библиотеки ввода-вывода языка C. Пользовательская часть кода практически не меняется:

Листинг 4: файл main.i

```
# 4 "shiftArray.h" 2

# 4 "shiftArray.h"
void shiftArray(int array[], int length, int shift);
# 2 "main.c" 2

int main() {
    int array[3] = {1, 2, 3};
    const int shift = 2;
    int length = sizeof(array)/sizeof(int);
    printf("Start Array: [");
    for(int i = 0; i < length; i++) {
        printf("%d", array[i]);
        if(i != length - 1) printf(", ");
    }
    printf("]\nshift = %d\n", shift);
    shiftArray(array, length, shift);
    printf("End Array: [");
    for (int i = 0; i < length; ++i) {
        printf("%d ", array[i]);
    }
    printf("]");
    return 0;
}
```

Аналогично происходит препроцессирование функции:

Листинг 5: файл shiftArray.i

```
# 4 "shiftArray.h" 2

# 4 "shiftArray.h"
void shiftArray(int array[], int length, int shift);
# 2 "shiftArray.c" 2

void shiftArray(int array[], int length, int shift) {
    for(int j = 0; j < shift; ++j) {
        int tmp = array[0];
        for(int i = 0; i < length - 1; ++i) {
            array[i] = array[i+1];
        }
        array[length - 1] = tmp;
    }
}
```

```
}  
}
```

### 3.2 Анализ выхода компилятора:

Листинг 6: файл main.s

```
.file "main.c"  
.option nopic  
.attribute arch, "rv32i2p0"  
.attribute unaligned_access, 0  
.attribute stack_align, 16  
.text  
.section .rodata.str1.4,"aMS",@progbits,1  
.align 2  
.LC0:  
.string "Start Array: ["  
.align 2  
.LC1:  
.string "%d"  
.align 2  
.LC2:  
.string ", "  
.align 2  
.LC3:  
.string "]\nshift = %d\n"  
.align 2  
.LC4:  
.string "End Array: ["  
.align 2  
.LC5:  
.string "%d "  
.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)  
li a5,1  
sw a5,4(sp)  
li a5,2  
sw a5,8(sp)  
li a5,3  
sw a5,12(sp)  
lui a0,%hi(.LC0)  
addi a0,a0,%lo(.LC0)  
call printf  
lw a1,4(sp)
```

```

lui    s0,%hi(.LC1)
addi   a0,s0,%lo(.LC1)
call   printf
lui    s1,%hi(.LC2)
addi   a0,s1,%lo(.LC2)
call   printf
lw     a1,8(sp)
addi   a0,s0,%lo(.LC1)
call   printf
addi   a0,s1,%lo(.LC2)
call   printf
lw     a1,12(sp)
addi   a0,s0,%lo(.LC1)
call   printf
li     a1,2
lui    a0,%hi(.LC3)
addi   a0,a0,%lo(.LC3)
call   printf
li     a2,2
li     a1,3
addi   a0,sp,4
call   shiftArray
lui    a0,%hi(.LC4)
addi   a0,a0,%lo(.LC4)
call   printf
lw     a1,4(sp)
lui    s0,%hi(.LC5)
addi   a0,s0,%lo(.LC5)
call   printf
lw     a1,8(sp)
addi   a0,s0,%lo(.LC5)
call   printf
lw     a1,12(sp)
addi   a0,s0,%lo(.LC5)
call   printf
li     a0,93
call   putchar
li     a0,0
lw     ra,28(sp)
lw     s0,24(sp)
lw     s1,20(sp)
addi   sp,sp,32
jr     ra
.size   main,.-main
.ident  "GCC: (GNU) 10.2.0"

```

Листинг 7: файл shiftArray.s

```

.file   "shiftArray.c"
.option nopic
.attribute arch, "rv32i2p0"

```

```

.attribute unaligned_access, 0
.attribute stack_align, 16
.text
.align 2
.globl shiftArray
.type shiftArray, @function
shiftArray:
    ble    a2,zero,.L1
    slli   a5,a1,2
    addi   a5,a5,-4
    add    t3,a0,a5
    addi   a5,a5,4
    add    a3,a0,a5
    li     a6,0
    li     t1,1
.L5:
    lw     a7,0(a0)
    ble    a1,t1,.L3
    addi   a5,a0,4
.L4:
    lw     a4,0(a5)
    sw     a4,-4(a5)
    addi   a5,a5,4
    bne    a5,a3,.L4
.L3:
    sw     a7,0(t3)
    addi   a6,a6,1
    bne    a2,a6,.L5
.L1:
    ret
.size    shiftArray, .-shiftArray
.ident   "GCC: (GNU) 10.2.0"

```

В программе `main` выполняется обращение к подпрограмме `shiftArray` (значение регистра `ra`, содержащее адрес возврата из `main`, сохраняется на время вызова в стеке). Следует отметить, что символ `shiftArray` используется в файле `main.s`, но никак не определяется.

### 3.3 Анализ состава и содержимого секций, таблицы символов, таблицы перемещений и отладочной информации, содержащейся в объектных файлах и исполняемом файле

Сформированный ассемблером объектный файл `main.o` и `shiftArray.o` должны содержать коды инструкций, таблицу символов и таблицу перемещений. В отличие от ранее рассмотренных файлов, объектный файл не является



текстовым, для изучения его содержимого используем утилиту objdump, отображающую содержимое бинарных файлов в текстовом виде:

```
katerina@pop-os:~/Documents/Programming/Turing/shift_array$ riscv32-unknown-elf-objdump -h main.o
main.o:      file format elf32-littleriscv

Sections:
Idx Name          Size      VMA           LMA           File off  Algn
  0 .text          00000118  00000000  00000000  00000034  2**2
    CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data          00000000  00000000  00000000  0000014c  2**0
    CONTENTS, ALLOC, LOAD, DATA
  2 .bss           00000000  00000000  00000000  0000014c  2**0
    ALLOC
  3 .rodata.str1.4 0000003c  00000000  00000000  0000014c  2**2
    CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .comment        00000013  00000000  00000000  00000188  2**0
    CONTENTS, READONLY
  5 .riscv.attributes 0000001c  00000000  00000000  0000019b  2**0
    CONTENTS, READONLY
```

Рис.2. Содержимое заголовков секций main.o

В файле имеются следующие секции:

- .text – секция кода;
- .data – секция инициализированных данных;
- .bss – секция данных, инициализированных нулями;
- .rodata – секция неизменяемых данных;
- .rodata.str1.4 –подсекция неизменяемых данных, используется компилятором для хранения дополнительной информации (например, о типе данных) для компоновщика;
- .comment – секция данных о версиях;
- .riscv.attributes – атрибуты для указания определенных свойств функции (в помощь компилятору для проверок и оптимизации кода).

Значения в столбце size приведены в 16-ричной системе счисления.

```
katerina@pop-os:~/Documents/Programming/Turing/shift_array$ riscv32-unknown-elf-objdump -h shiftArray.o
shiftArray.o: file format elf32-littleriscv

Sections:
Idx Name          Size      VMA           LMA           File off  Algn
  0 .text          0000004c  00000000  00000000  00000034  2**2
  1 .data           00000000  00000000  00000000  00000080  2**0
  2 .bss            00000000  00000000  00000000  00000080  2**0
  3 .comment        00000013  00000000  00000000  00000080  2**0
  4 .riscv.attributes 0000001c  00000000  00000000  00000093  2**0
```

Рис.3. Содержимое заголовков секций shiftArray.o

Таблицы символов объектных файлов main.o и shiftArray.o:

```
$ riscv32-unknown-elf-objdump -t main.o shiftArray.o
```

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

SYMBOL TABLE:
00000000 l      df *ABS* 00000000 main.c
00000000 l      d  .text 00000000 .text
00000000 l      d  .data 00000000 .data
00000000 l      d  .bss 00000000 .bss
00000000 l      d  .rodata.str1.4 00000000 .rodata.str1.4
00000000 l      .rodata.str1.4 00000000 .LC0
00000010 l      .rodata.str1.4 00000000 .LC1
00000014 l      .rodata.str1.4 00000000 .LC2
00000018 l      .rodata.str1.4 00000000 .LC3
00000028 l      .rodata.str1.4 00000000 .LC4
00000038 l      .rodata.str1.4 00000000 .LC5
00000000 l      d  .comment 00000000 .comment
00000000 l      d  .riscv.attributes 00000000 .riscv.attributes
00000000 g      F  .text 00000118 main
00000000      *UND* 00000000 printf
00000000      *UND* 00000000 shiftArray
00000000      *UND* 00000000 putchar

shiftArray.o:    file format elf32-littleriscv

SYMBOL TABLE:
00000000 l      df *ABS* 00000000 shiftArray.c
00000000 l      d  .text 00000000 .text
00000000 l      d  .data 00000000 .data
00000000 l      d  .bss 00000000 .bss
00000048 l      .text 00000000 .L1
0000003c l      .text 00000000 .L3
0000002c l      .text 00000000 .L4
00000020 l      .text 00000000 .L5
00000000 l      d  .comment 00000000 .comment
00000000 l      d  .riscv.attributes 00000000 .riscv.attributes
00000000 g      F  .text 0000004c shiftArray
```

Рис.4. Содержимое таблиц символов объектных файлов

Как и ожидалось таблица содержит 2 глобальные (флаг g) функции (флаг F) – main и shiftArray, также три неопределенных (UND) символа.

UND означает, что символы printf, putchar и shiftArray 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.

Изучим содержимое секции .text объектных файлов main.o и shiftArray.o:

Листинг 8: Содержимое секции .text объектного файла main.o и shiftArray.o

```
$ riscv32-unknown-elf-objdump -d -M no-aliases -j .text main.o shiftArray.o
main.o: file format elf32-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:	00100793	addi	a5,zero,1
14:	00f12223	sw	a5,4(sp)
18:	00200793	addi	a5,zero,2
1c:	00f12423	sw	a5,8(sp)
20:	00300793	addi	a5,zero,3
24:	00f12623	sw	a5,12(sp)
28:	00000537	lui	a0,0x0
2c:	00050513	addi	a0,a0,0 # 0 <main>
30:	00000097	auipc	ra,0x0
34:	000080e7	jalr	ra,0(ra) # 30 <main+0x30>
38:	00412583	lw	a1,4(sp)
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 <main+0x44>
4c:	000004b7	lui	s1,0x0
50:	00048513	addi	a0,s1,0 # 0 <main>
54:	00000097	auipc	ra,0x0
58:	000080e7	jalr	ra,0(ra) # 54 <main+0x54>
5c:	00812583	lw	a1,8(sp)
60:	00040513	addi	a0,s0,0
64:	00000097	auipc	ra,0x0
68:	000080e7	jalr	ra,0(ra) # 64 <main+0x64>
6c:	00048513	addi	a0,s1,0
70:	00000097	auipc	ra,0x0

74:	000080e7	jalr	ra,0(ra) # 70 <main+0x70>
78:	00c12583	lw	a1,12(sp)
7c:	00040513	addi	a0,s0,0
80:	00000097	auipc	ra,0x0
84:	000080e7	jalr	ra,0(ra) # 80 <main+0x80>
88:	00200593	addi	a1,zero,2
8c:	00000537	lui	a0,0x0
90:	00050513	addi	a0,a0,0 # 0 <main>
94:	00000097	auipc	ra,0x0
98:	000080e7	jalr	ra,0(ra) # 94 <main+0x94>
9c:	00200613	addi	a2,zero,2
a0:	00300593	addi	a1,zero,3
a4:	00410513	addi	a0,sp,4
a8:	00000097	auipc	ra,0x0
ac:	000080e7	jalr	ra,0(ra) # a8 <main+0xa8>
b0:	00000537	lui	a0,0x0
b4:	00050513	addi	a0,a0,0 # 0 <main>
b8:	00000097	auipc	ra,0x0
bc:	000080e7	jalr	ra,0(ra) # b8 <main+0xb8>
c0:	00412583	lw	a1,4(sp)
c4:	00000437	lui	s0,0x0
c8:	00040513	addi	a0,s0,0 # 0 <main>
cc:	00000097	auipc	ra,0x0
d0:	000080e7	jalr	ra,0(ra) # cc <main+0xcc>
d4:	00812583	lw	a1,8(sp)
d8:	00040513	addi	a0,s0,0
dc:	00000097	auipc	ra,0x0
e0:	000080e7	jalr	ra,0(ra) # dc <main+0xdc>
e4:	00c12583	lw	a1,12(sp)
e8:	00040513	addi	a0,s0,0
ec:	00000097	auipc	ra,0x0
f0:	000080e7	jalr	ra,0(ra) # ec <main+0xec>
f4:	05d00513	addi	a0,zero,93
f8:	00000097	auipc	ra,0x0
fc:	000080e7	jalr	ra,0(ra) # f8 <main+0xf8>
100:	00000513	addi	a0,zero,0
104:	01c12083	lw	ra,28(sp)
108:	01812403	lw	s0,24(sp)
10c:	01412483	lw	s1,20(sp)
110:	02010113	addi	sp,sp,32
114:	00008067	jalr	zero,0(ra)

shiftArray.o: file format elf32-littleriscv

Disassembly of section .text:

00000000 <shiftArray>:

0:	04c05463	bge	zero,a2,48 <.L1>
4:	00259793	slli	a5,a1,0x2
8:	ffc78793	addi	a5,a5,-4
c:	00f50e33	add	t3,a0,a5
10:	00478793	addi	a5,a5,4

14:	00f506b3	add	a3,a0,a5
18:	00000813	addi	a6,zero,0
1c:	00100313	addi	t1,zero,1
00000020 <.L5>:			
20:	00052883	lw	a7,0(a0)
24:	00b35c63	bge	t1,a1,3c <.L3>
28:	00450793	addi	a5,a0,4
0000002c <.L4>:			
2c:	0007a703	lw	a4,0(a5)
30:	fee7ae23	sw	a4,-4(a5)
34:	00478793	addi	a5,a5,4
38:	fed79ae3	bne	a5,a3,2c <.L4>
0000003c <.L3>:			
3c:	011e2023	sw	a7,0(t3)
40:	00180813	addi	a6,a6,1
44:	fd061ee3	bne	a2,a6,20 <.L5>
00000048 <.L1>:			
48:	00008067	jalr	zero,0(ra)

Результат дизассемблирования shiftArray.o интереса не представляет, в отличие от результата дизассемблирования main.o: сравнивая его с main.s, можно понять, что псевдоинструкция вызова подпрограммы shiftArray, транслировалась ассемблером в следующую пару инструкций:

a8:	00000097	auipc	ra,0x0
ac:	000080e7	jalr	ra,0(ra) # a8 <main+0xa8>

Результатом выполнения этой пары инструкций станет переход на адрес main+0xa8 (0+168=168) - произойдет заикливание. Это показано в выводе дизассемблера. Загадочное поведение ассемблера объясняется очень просто: ассемблер не имел возможности определить целевой адрес перехода (кроме того, что этот адрес обозначен символом shiftArray), поэтому не мог сформировать корректную инструкцию (пару инструкций) передачи управления. В результате была сформирована пара инструкций с некорректными (нулевыми) значениями непосредственных операндов. Для получения исполняемого кода эта пара инструкций должна быть исправлена компоновщиком.

### 3.4 Содержимое таблицы перемещений:

```
$ riscv32-unknown-elf-objdump -r main.o shiftArray.o
```

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

```
RELOCATION RECORDS FOR [.text]:
```

OFFSET	TYPE	VALUE
00000028	R_RISCV_HI20	.LC0
00000028	R_RISCV_RELAX	*ABS*
0000002c	R_RISCV_LO12_I	.LC0
0000002c	R_RISCV_RELAX	*ABS*
00000030	R_RISCV_CALL	printf
00000030	R_RISCV_RELAX	*ABS*
0000003c	R_RISCV_HI20	.LC1
0000003c	R_RISCV_RELAX	*ABS*
00000040	R_RISCV_LO12_I	.LC1
00000040	R_RISCV_RELAX	*ABS*
00000044	R_RISCV_CALL	printf
00000044	R_RISCV_RELAX	*ABS*
0000004c	R_RISCV_HI20	.LC2
0000004c	R_RISCV_RELAX	*ABS*
00000050	R_RISCV_LO12_I	.LC2
00000050	R_RISCV_RELAX	*ABS*
00000054	R_RISCV_CALL	printf
00000054	R_RISCV_RELAX	*ABS*
00000060	R_RISCV_LO12_I	.LC1
00000060	R_RISCV_RELAX	*ABS*
00000064	R_RISCV_CALL	printf
00000064	R_RISCV_RELAX	*ABS*
0000006c	R_RISCV_LO12_I	.LC2
0000006c	R_RISCV_RELAX	*ABS*
00000070	R_RISCV_CALL	printf
00000070	R_RISCV_RELAX	*ABS*
0000007c	R_RISCV_LO12_I	.LC1
0000007c	R_RISCV_RELAX	*ABS*
00000080	R_RISCV_CALL	printf
00000080	R_RISCV_RELAX	*ABS*
0000008c	R_RISCV_HI20	.LC3
0000008c	R_RISCV_RELAX	*ABS*
00000090	R_RISCV_LO12_I	.LC3
00000090	R_RISCV_RELAX	*ABS*
00000094	R_RISCV_CALL	printf
00000094	R_RISCV_RELAX	*ABS*
000000a8	R_RISCV_CALL	shiftArray
000000a8	R_RISCV_RELAX	*ABS*
000000b0	R_RISCV_HI20	.LC4
000000b0	R_RISCV_RELAX	*ABS*
000000b4	R_RISCV_LO12_I	.LC4
000000b4	R_RISCV_RELAX	*ABS*
000000b8	R_RISCV_CALL	printf

Рис.5. Таблица перемещений main.o

```
000000e8 R_RISCV_RELAX *ABS*
000000ec R_RISCV_CALL printf
000000ec R_RISCV_RELAX *ABS*
000000f8 R_RISCV_CALL putchar
000000f8 R_RISCV_RELAX *ABS*
```

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

```
RELOCATION RECORDS FOR [.text]:
```

OFFSET	TYPE	VALUE
00000000	R_RISCV_BRANCH	.L1
00000024	R_RISCV_BRANCH	.L3
00000038	R_RISCV_BRANCH	.L4
00000044	R_RISCV_BRANCH	.L5

Рис.6. Таблица перемещений  
shiftArray.o

Информация обо всех «неоконченных» инструкциях передается ассемблером компоновщику посредством таблицы перемещений.

Содержимое shiftArray.o не требует модификации, поэтому не содержит записей о перемещениях (relocation entries). В файле же main.o имеется 7

записей, среди которых есть запись относящаяся к адресу 30 (как мы видели выше, по этому адресу в main.o находится инструкция пары auipc+jalr).

Дизассемблирование и вывод таблицы перемещений можно совместить

#### Листинг 9: Дизассемблирование и вывод таблицы перемещений main.o

```
$ riscv32-unknown-elf-objdump -d -M no-aliases -r main.o
```

```
main.o: file format elf32-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: 00100793      addi   a5,zero,1
14: 00f12223      sw     a5,4(sp)
18: 00200793      addi   a5,zero,2
1c: 00f12423      sw     a5,8(sp)
20: 00300793      addi   a5,zero,3
24: 00f12623      sw     a5,12(sp)
28: 00000537      lui    a0,0x0
      28: R_RISCV_HI20 .LC0
      28: R_RISCV_RELAX *ABS*
2c: 00050513      addi   a0,a0,0 # 0 <main>
      2c: R_RISCV_LO12_I .LC0
      2c: R_RISCV_RELAX *ABS*
30: 00000097      auipc  ra,0x0
      30: R_RISCV_CALL printf
      30: R_RISCV_RELAX *ABS*
34: 000080e7      jalr   ra,0(ra) # 30 <main+0x30>
38: 00412583      lw     a1,4(sp)
3c: 00000437      lui    s0,0x0
      3c: R_RISCV_HI20 .LC1
      3c: R_RISCV_RELAX *ABS*
40: 00040513      addi   a0,s0,0 # 0 <main>
      40: R_RISCV_LO12_I .LC1
      40: R_RISCV_RELAX *ABS*
44: 00000097      auipc  ra,0x0
      44: R_RISCV_CALL printf
      44: R_RISCV_RELAX *ABS*
48: 000080e7      jalr   ra,0(ra) # 44 <main+0x44>
4c: 000004b7      lui    s1,0x0
      4c: R_RISCV_HI20 .LC2
      4c: R_RISCV_RELAX *ABS*
50: 00048513      addi   a0,s1,0 # 0 <main>
      50: R_RISCV_LO12_I .LC2
      50: R_RISCV_RELAX *ABS*
```

```

54: 00000097      auipc ra,0x0
      54: R_RISCV_CALL printf
      54: R_RISCV_RELAX      *ABS*
58: 000080e7      jalr  ra,0(ra) # 54 <main+0x54>
5c: 00812583      lw    a1,8(sp)
60: 00040513      addi  a0,s0,0
      60: R_RISCV_LO12_I      .LC1
      60: R_RISCV_RELAX      *ABS*
64: 00000097      auipc ra,0x0
      64: R_RISCV_CALL printf
      64: R_RISCV_RELAX      *ABS*
68: 000080e7      jalr  ra,0(ra) # 64 <main+0x64>
6c: 00048513      addi  a0,s1,0
      6c: R_RISCV_LO12_I      .LC2
      6c: R_RISCV_RELAX      *ABS*
70: 00000097      auipc ra,0x0
      70: R_RISCV_CALL printf
      70: R_RISCV_RELAX      *ABS*
74: 000080e7      jalr  ra,0(ra) # 70 <main+0x70>
78: 00c12583      lw    a1,12(sp)
7c: 00040513      addi  a0,s0,0
      7c: R_RISCV_LO12_I      .LC1
      7c: R_RISCV_RELAX      *ABS*
80: 00000097      auipc ra,0x0
      80: R_RISCV_CALL printf
      80: R_RISCV_RELAX      *ABS*
84: 000080e7      jalr  ra,0(ra) # 80 <main+0x80>
88: 00200593      addi  a1,zero,2
8c: 00000537      lui   a0,0x0
      8c: R_RISCV_HI20 .LC3
      8c: R_RISCV_RELAX      *ABS*
90: 00050513      addi  a0,a0,0 # 0 <main>
      90: R_RISCV_LO12_I      .LC3
      90: R_RISCV_RELAX      *ABS*
94: 00000097      auipc ra,0x0
      94: R_RISCV_CALL printf
      94: R_RISCV_RELAX      *ABS*
98: 000080e7      jalr  ra,0(ra) # 94 <main+0x94>
9c: 00200613      addi  a2,zero,2
a0: 00300593      addi  a1,zero,3
a4: 00410513      addi  a0,sp,4
a8: 00000097      auipc ra,0x0
      a8: R_RISCV_CALL shiftArray
      a8: R_RISCV_RELAX      *ABS*
ac: 000080e7      jalr  ra,0(ra) # a8 <main+0xa8>
b0: 00000537      lui   a0,0x0
      b0: R_RISCV_HI20 .LC4
      b0: R_RISCV_RELAX      *ABS*
b4: 00050513      addi  a0,a0,0 # 0 <main>
      b4: R_RISCV_LO12_I      .LC4
      b4: R_RISCV_RELAX      *ABS*
b8: 00000097      auipc ra,0x0

```



		b8: R_RISCV_CALL printf
		b8: R_RISCV_RELAX *ABS*
bc:	000080e7	jalr ra,0(ra) # b8 <main+0xb8>
c0:	00412583	lw a1,4(sp)
c4:	00000437	lui s0,0x0
		c4: R_RISCV_HI20 .LC5
		c4: R_RISCV_RELAX *ABS*
c8:	00040513	addi a0,s0,0 # 0 <main>
		c8: R_RISCV_LO12_I .LC5
		c8: R_RISCV_RELAX *ABS*
cc:	00000097	auipc ra,0x0
		cc: R_RISCV_CALL printf
		cc: R_RISCV_RELAX *ABS*
d0:	000080e7	jalr ra,0(ra) # cc <main+0xcc>
d4:	00812583	lw a1,8(sp)
d8:	00040513	addi a0,s0,0
		d8: R_RISCV_LO12_I .LC5
		d8: R_RISCV_RELAX *ABS*
dc:	00000097	auipc ra,0x0
		dc: R_RISCV_CALL printf
		dc: R_RISCV_RELAX *ABS*
e0:	000080e7	jalr ra,0(ra) # dc <main+0xdc>
e4:	00c12583	lw a1,12(sp)
e8:	00040513	addi a0,s0,0
		e8: R_RISCV_LO12_I .LC5
		e8: R_RISCV_RELAX *ABS*
ec:	00000097	auipc ra,0x0
		ec: R_RISCV_CALL printf
		ec: R_RISCV_RELAX *ABS*
f0:	000080e7	jalr ra,0(ra) # ec <main+0xec>
f4:	05d00513	addi a0,zero,93
f8:	00000097	auipc ra,0x0
		f8: R_RISCV_CALL putchar
		f8: R_RISCV_RELAX *ABS*
fc:	000080e7	jalr ra,0(ra) # f8 <main+0xf8>
100:	00000513	addi a0,zero,0
104:	01c12083	lw ra,28(sp)
108:	01812403	lw s0,24(sp)
10c:	01412483	lw s1,20(sp)
110:	02010113	addi sp,sp,32
114:	00008067	jalr zero,0(ra)

```
$ riscv32-unknown-elf-objdump -d -M no-aliases -r shiftArray.o
```

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

Disassembly of section .text:

00000000 <shiftArray>:
  0:  04c05463          bge      zero,a2,48 <.L1>
                   0: R_RISCV_BRANCH      .L1
  4:  00259793          slli     a5,a1,0x2
  8:  ffc78793          addi     a5,a5,-4
  c:  00f50e33          add      t3,a0,a5
 10:  00478793          addi     a5,a5,4
 14:  00f506b3          add      a3,a0,a5
 18:  00000813          addi     a6,zero,0
 1c:  00100313          addi     t1,zero,1

00000020 <.L5>:
 20:  00052883          lw       a7,0(a0)
 24:  00b35c63          bge      t1,a1,3c <.L3>
                   24: R_RISCV_BRANCH      .L3
 28:  00450793          addi     a5,a0,4

0000002c <.L4>:
 2c:  0007a703          lw       a4,0(a5)
 30:  fee7ae23          sw       a4,-4(a5)
 34:  00478793          addi     a5,a5,4
 38:  fed79ae3          bne      a5,a3,2c <.L4>
                   38: R_RISCV_BRANCH      .L4

0000003c <.L3>:
 3c:  011e2023          sw       a7,0(t3)
 40:  00180813          addi     a6,a6,1
 44:  fd061ee3          bne      a2,a6,20 <.L5>
                   44: R_RISCV_BRANCH      .L5

00000048 <.L1>:
 48:  00008067          jalr     zero,0(ra)
```

Рис. 7. Дизассемблирование и вывод таблицы перемещений shiftArray.o

Для того чтобы внести необходимые исправления, требуется знать, что исправить, как исправить и какой символ следует использовать, именно эта информация и содержится в записях о перемещениях. Так, в первой записи таблицы перемещений указано, что по адресу 2a следует исправить пару инструкций (тип перемещения “R\_RISCV\_CALL”) так, чтобы результат соответствовал вызову подпрограммы shiftArray. Типы перемещений специфичны для каждой архитектуры системы команд и обычно определены в ABI (Application Binary Interface).

Вторая запись таблицы перемещений специфична для средств разработки RISC-V. 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.

### 3.5 Результат компоновки

```
riscv64-unknown-elf-gcc -Wl,--no-relax main.o shiftArray.o -o main
```

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

Листинг 10: файл main.ds (строки 80-150)

```
00010178 <main>:
10178:    fe010113      addi    sp,sp,-32
1017c:    00112e23      sw      ra,28(sp)
10180:    00812c23      sw      s0,24(sp)
10184:    00912a23      sw      s1,20(sp)
10188:    00100793      addi    a5,zero,1
1018c:    00f12223      sw      a5,4(sp)
10190:    00200793      addi    a5,zero,2
10194:    00f12423      sw      a5,8(sp)
10198:    00300793      addi    a5,zero,3
1019c:    00f12623      sw      a5,12(sp)
101a0:    00026537      lui     a0,0x26
101a4:    04850513      addi    a0,a0,72 # 26048 <__clzsi2+0x4c>
101a8:    00000097      auipc   ra,0x0
101ac:    32c080e7      jalr    ra,812(ra) # 104d4 <printf>
101b0:    00412583      lw      a1,4(sp)
101b4:    00026437      lui     s0,0x26
101b8:    05840513      addi    a0,s0,88 # 26058 <__clzsi2+0x5c>
101bc:    00000097      auipc   ra,0x0
101c0:    318080e7      jalr    ra,792(ra) # 104d4 <printf>
101c4:    000264b7      lui     s1,0x26
101c8:    05c48513      addi    a0,s1,92 # 2605c <__clzsi2+0x60>
101cc:    00000097      auipc   ra,0x0
101d0:    308080e7      jalr    ra,776(ra) # 104d4 <printf>
101d4:    00812583      lw      a1,8(sp)
101d8:    05840513      addi    a0,s0,88
101dc:    00000097      auipc   ra,0x0
101e0:    2f8080e7      jalr    ra,760(ra) # 104d4 <printf>
101e4:    05c48513      addi    a0,s1,92
101e8:    00000097      auipc   ra,0x0
101ec:    2ec080e7      jalr    ra,748(ra) # 104d4 <printf>
101f0:    00c12583      lw      a1,12(sp)
101f4:    05840513      addi    a0,s0,88
101f8:    00000097      auipc   ra,0x0
101fc:    2dc080e7      jalr    ra,732(ra) # 104d4 <printf>
10200:    00200593      addi    a1,zero,2
10204:    00026537      lui     a0,0x26
```

10208:	06050513	addi	a0,a0,96 # 26060 <__clzsi2+0x64>
1020c:	00000097	auipc	ra,0x0
10210:	2c8080e7	jalr	ra,712(ra) # 104d4 <printf>
10214:	00200613	addi	a2,zero,2
10218:	00300593	addi	a1,zero,3
1021c:	00410513	addi	a0,sp,4
10220:	00000097	auipc	ra,0x0
10224:	070080e7	jalr	ra,112(ra) # 10290 <shiftArray>
10228:	00026537	lui	a0,0x26
1022c:	07050513	addi	a0,a0,112 # 26070 <__clzsi2+0x74>
10230:	00000097	auipc	ra,0x0
10234:	2a4080e7	jalr	ra,676(ra) # 104d4 <printf>
10238:	00412583	lw	a1,4(sp)
1023c:	00026437	lui	s0,0x26
10240:	08040513	addi	a0,s0,128 # 26080 <__clzsi2+0x84>
10244:	00000097	auipc	ra,0x0
10248:	290080e7	jalr	ra,656(ra) # 104d4 <printf>
1024c:	00812583	lw	a1,8(sp)
10250:	08040513	addi	a0,s0,128
10254:	00000097	auipc	ra,0x0
10258:	280080e7	jalr	ra,640(ra) # 104d4 <printf>
1025c:	00c12583	lw	a1,12(sp)
10260:	08040513	addi	a0,s0,128
10264:	00000097	auipc	ra,0x0
10268:	270080e7	jalr	ra,624(ra) # 104d4 <printf>
1026c:	05d00513	addi	a0,zero,93
10270:	00000097	auipc	ra,0x0
10274:	2c4080e7	jalr	ra,708(ra) # 10534 <putchar>
10278:	00000513	addi	a0,zero,0
1027c:	01c12083	lw	ra,28(sp)
10280:	01812403	lw	s0,24(sp)
10284:	01412483	lw	s1,20(sp)
10288:	02010113	addi	sp,sp,32
1028c:	00008067	jalr	zero,0(ra)

Прежде всего можно видеть, что в результат компоновки попало содержимое обоих объектных файлов – main.o и shiftArray.o. Инструкции подпрограммы shiftArray начинаются с адреса 101f8<sub>16</sub>, и пара инструкций auipc+jalr, вызывающих подпрограмму shiftArray соответствующим образом откорректированы.

### 3.6 Анализ отладочной информации

```
$ riscv32-unknown-elf-objdump -f -h main
```

```
main:      file format elf32-littleriscv
architecture: riscv:rv32, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x00010094
```

Idx	Name	Size	VMA	LMA	File off	Algn	
0	.text	00015fd4	00010074	00010074	00000074	2**2	
	CONTENTS, ALLOC, LOAD, READONLY, CODE						
1	.rodata	00000e0c	00026048	00026048	00016048	2**3	
	CONTENTS, ALLOC, LOAD, READONLY, DATA						
2	.eh_frame	000000b4	00027000	00027000	00017000	2**2	
	CONTENTS, ALLOC, LOAD, DATA						
3	.init_array	00000008	000270b4	000270b4	000170b4	2**2	
	CONTENTS, ALLOC, LOAD, DATA						
4	.fini_array	00000004	000270bc	000270bc	000170bc	2**2	
	CONTENTS, ALLOC, LOAD, DATA						
5	.data	0000099c	000270c0	000270c0	000170c0	2**3	
	CONTENTS, ALLOC, LOAD, DATA						
6	.sdata	0000002c	00027a60	00027a60	00017a60	2**3	
	CONTENTS, ALLOC, LOAD, DATA						
7	.sbss	00000018	00027a8c	00027a8c	00017a8c	2**2	
	ALLOC						
8	.bss	00000044	00027aa4	00027aa4	00017a8c	2**2	
	ALLOC						
9	.comment	00000012	00000000	00000000	00017a8c	2**0	
	CONTENTS, READONLY						
10	.riscv.attributes	0000001c	00000000	00000000	00017a9e	2**0	
	CONTENTS, READONLY						
11	.debug_aranges	00000218	00000000	00000000	00017ac0	2**3	
	CONTENTS, READONLY, DEBUGGING, OCTETS						
12	.debug_info	00006a79	00000000	00000000	00017cd8	2**0	
	CONTENTS, READONLY, DEBUGGING, OCTETS						
13	.debug_abbrev	00001671	00000000	00000000	0001e751	2**0	
	CONTENTS, READONLY, DEBUGGING, OCTETS						
14	.debug_line	0000a41a	00000000	00000000	0001fdc2	2**0	
	CONTENTS, READONLY, DEBUGGING, OCTETS						
15	.debug_frame	00000308	00000000	00000000	0002a1dc	2**2	
	CONTENTS, READONLY, DEBUGGING, OCTETS						
16	.debug_str	00000dd8	00000000	00000000	0002a4e4	2**0	
	CONTENTS, READONLY, DEBUGGING, OCTETS						
17	.debug_loc	00008826	00000000	00000000	0002b2bc	2**0	
	CONTENTS, READONLY, DEBUGGING, OCTETS						
18	.debug_ranges	00001630	00000000	00000000	00033ae2	2**0	
	CONTENTS, READONLY, DEBUGGING, OCTETS						

Рис.8. Содержимое файла main

Сформированный исполняемый файл содержит информацию для отладки (в секциях **.debug...**), полную таблицу символов и сведения о версиях средств разработки.

Встреченные разделы DWARF:

- **.debug\_abbrev** – сокращения , используемые в **.debug\_info** разделе;
- **.debug\_aranges** – таблица поиска для сопоставления адресов с единицами компиляции;
- **.debug\_frame** – информация о кадре вызова;
- **.debug\_info** – раздел основной информации DWARF;
- **.debug\_line** – информация о номере строки;
- **.debug\_loc** – списки местоположений, используемые в атрибутах **DW\_AT\_location**;
- **.debug\_ranges** – диапазоны адресов, используемые в трибутах **DW\_AT\_ranges**;
- **.debug\_str** – таблица строк, используемая в **.debug\_info**.

### 3.7 Выделение разработанной функции в статическую библиотеку

```
$ riscv32-unknown-elf-ar -rsc shiftArray.a shiftArray.o
$ riscv32-unknown-elf-gcc -O1 --save-temps main.c shiftArray.a -o mainWithLib
$ riscv32-unknown-elf-objdump -t main
```

Листинг 11: таблица символов полученного исполняемого файла

```
...
00010290 g    F .text 0000004c shiftArray
...
000103b4 g    F .text 000000dc memset
00010178 g    F .text 00000118 main
...
```

Как и следовало ожидать, в состав исполняемого файла вошло содержимое всех объектных файлов, указанных в команде сборки.

### 3.8 Создание и использование полученной статической библиотеки

```
katerina@pop-os:~/Documents/Programming/Turing/shift_array$ riscv32-unknown-elf-nm shiftArray.a
shiftArray.o:
00000048 t .L1
0000003c t .L3
0000002c t .L4
00000020 t .L5
00000000 T shiftArray
katerina@pop-os:~/Documents/Programming/Turing/shift_array$
```

Рис.10. Список символов библиотеки

```
output: main.o shiftArray.a
gcc -o main main.o -L. -l:shiftArray.a
main.o: main.c
gcc -c main.c
shiftArray.a: shiftArray.o shiftArray.h
ar -rsc shiftArray.a shiftArray.o
shiftArray.o:
gcc -c shiftArray.c
clean:
rm *.o
```

Рис.11. Содержимое make-файла Makefile

Что происходит в *makefile*:

- Создаём объектный файл **main.o** из исходного **main.c**;
- Создаём объектный файл **shiftArray.o** из исходного **shiftArray.c**;
- Архивируем объектный файл **shiftArray.o** (создаём статическую библиотеку **shiftArray.a**);
- Компонуем статическую библиотеку **shiftArray.a** с объектным файлом **main.o**, получаем исполняемый файл **main**.

```

katerina@pop-os:~/Documents/Programming/Turing/shift_array$ make
gcc -c main.c
gcc -o main main.o -L. -l:shiftArray.a
katerina@pop-os:~/Documents/Programming/Turing/shift_array$ ./main
Start Array: [1, 2, 3]
shift = 2
End Array: [3 1 2 ]
katerina@pop-os:~/Documents/Programming/Turing/shift_array$

```

Рис.12. Вывод результата.

#### 4 Результаты

В ходе работы исследован процесс сборки проекта на языке СИ.

Он состоит из:

- Препроцессирования исходного <filename>.c в <filename>.i;
- Компиляции полученного <filename>.i в файл ассемблера <filename>.s;
- Ассемблирования <filename>.s в объектный файл <filename>.o;
- Компоновки объектного файла <filename>.o в исполняемый файл.

Также были рассмотрены makefile'ы, которые существенно упрощают процесс сборки.

Вместо того, чтобы поочередно набирать команды в терминале, используется единственная команда make, которая по инструкциям в makefile'е собирает программу в автоматическом режиме.