

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

Отчет по лабораторной работе №4

Дисциплина: Низкоуровневое программирование.

Тема: Раздельная компиляция.

Выполнил

студент гр. 3530901/90003 _____ Бехтольд Ек.В.
(подпись)

Принял

преподаватель _____ Алексюк А.О.
(подпись)

«___» _____ 2021 г.

Санкт-Петербург
2021

ОГЛАВЛЕНИЕ

1. Техническое задание	3
2. Метод решения	3
3. Решение	3
3.1. Анализ выхода препроцессора	8
3.2. Анализ выхода компилятора	9
3.3. Анализ состава и содержимого секций, таблицы символов, таблицы перемещений и отладочной информации, содержащейся в объектных файлах и исполняемом файле	11
3.4. Содержимое таблицы перемещений	16
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
4. Вывод	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

Реализуем программу на языке C:

Листинг 1: файл shift_array.h

```
void shiftArray(int array[], int shift, int length);
```

Листинг 2: файл array_shift.c

```

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;
    }
}

```

Листинг 3: файл main.c

```

#include <stdio.h>
#include "shift_array.h"
#define ARRAY_LENGTH 3

int main() {
    int array[ARRAY_LENGTH] = {1, 2, 3};
    const int shift = 2;
    int length = ARRAY_LENGTH;
    shiftArray(array, shift, length);

    for (int i = 0; i < ARRAY_LENGTH; ++i) {
        printf("%d ", array[i]);
    }
}

```

Начнем сборку созданных программ на языке C по шагам. Первым шагом является препроцессирование файлов исходного текста “shift_array.c” и “main.c” в файлы “shift_array.i” и “main.i”.

```

y$ riscv32-unknown-elf-gcc --save-temps -march=rv32i -mabi=ilp32 -O1 -v ./main.c -c ./shift_array.c >log 2>&1
y$ riscv32-unknown-elf-gcc --save-temps -march=rv32i -mabi=ilp32 -O1 -v ./main.c -c ./shift_array.c >log 2>&1
y$ █
1. Сборка простейшей программы

```

Рис. 1. Сборка программы.

Программа `riscv32-unknown-elf-gcc` является драйвером компилятора `gcc` (compiler driver), в данном случае она запускается со следующими параметрами командной строки (command line arguments):

`--save-temps` – сохранять промежуточные (intermediate, temporary) файлы, создаваемые в процессе сборки;

`-march=rv32i-mabi=ilp32` – целевым является процессор с базовой архитектурой системы команд RV32I;

`-O1` – выполнять простые оптимизации генерируемого кода (мы используем эту опцию в примерах, потому что обычно генерируемый код получается более простым);

`-v` – печатать (в стандартный поток ошибок) выполняемые драйвером команды, а также дополнительную информацию.

В конце команды используется т.н. «перенаправление вывода» (output redirection):

`>log` - вместо печати в консоли (обычно, это означает «на экране») вывод программы направляется в файл с именем “log” (если файл не существует, он создается; если файл существует, его содержимое будет утеряно);

`2>&1` – поток вывода ошибок (2 – стандартный «номер» этого потока) «связывается» с поток вывода («номер» 1), т.е. сообщения об ошибках (и информация, вывод которой вызван использованием флага “-v”) также выводятся в файл “log”.

Листинг 4: файл log

```

Using built-in specs.
COLLECT_GCC=riscv32-unknown-elf-gcc

```

Target: riscv32-unknown-elf

Configured with: /home/katerina/riscv-gnu-toolchain/riscv-gcc/configure --target=riscv32-unknown-elf --prefix=/home/katerina/riscv --disable-shared --disable-threads --enable-languages=c,c++ --with-system-zlib --enable-tls --with-newlib --with-sysroot=/home/katerina/riscv/riscv32-unknown-elf --with-native-system-header-dir=/include --disable-libmudflap --disable-libssp --disable-libquadmath --disable-libgomp --disable-nls --disable-tm-clone-registry --src=../riscv-gcc --disable-multilib --with-abi=ilp32 --with-arch=rv32i --with-tune=rocket 'CFLAGS_FOR_TARGET=-Os -mcmmodel=medlow' 'CXXFLAGS_FOR_TARGET=-Os -mcmmodel=medlow'

Thread model: single

Supported LTO compression algorithms: zlib

gcc version 10.2.0 (GCC)

COLLECT_GCC_OPTIONS='-save-temps' '-march=rv32i' '-mabi=ilp32' '-O1' '-v' '-c' '-mtune=rocket' '-march=rv32i'

/home/katerina/riscv/libexec/gcc/riscv32-unknown-elf/10.2.0/cc1 -E -quiet -v ./main.c -march=rv32i -mabi=ilp32 -mtune=rocket -march=rv32i -O1 -fpch-preprocess -o main.i

ignoring nonexistent directory "/home/katerina/riscv/riscv32-unknown-elf/usr/local/include"

ignoring duplicate directory "/home/katerina/riscv/riscv32-unknown-elf/include"

#include "... " search starts here:

#include <...> search starts here:

/home/katerina/riscv/lib/gcc/riscv32-unknown-elf/10.2.0/include

/home/katerina/riscv/lib/gcc/riscv32-unknown-elf/10.2.0/include-fixed

/home/katerina/riscv/lib/gcc/riscv32-unknown-elf/10.2.0/../../../../riscv32-unknown-elf/include

End of search list.

COLLECT_GCC_OPTIONS='-save-temps' '-march=rv32i' '-mabi=ilp32' '-O1' '-v' '-c' '-mtune=rocket' '-march=rv32i'

/home/katerina/riscv/libexec/gcc/riscv32-unknown-elf/10.2.0/cc1 -fpreprocessed main.i -quiet -dumpbase main.c -march=rv32i -mabi=ilp32 -mtune=rocket -march=rv32i -auxbase main -O1 -version -o main.s

GNU C17 (GCC) version 10.2.0 (riscv32-unknown-elf)

compiled by GNU C version 10.2.0, GMP version 6.2.0, MPFR version 4.1.0, MPC version 1.2.0-rc1, isl version none

GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsize=131072

GNU C17 (GCC) version 10.2.0 (riscv32-unknown-elf)

compiled by GNU C version 10.2.0, GMP version 6.2.0, MPFR version 4.1.0, MPC

```

version 1.2.0-rc1, isl version none

GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsize=131072

Compiler executable checksum: b2aad7ee3dc27f7c0e5c7e63fc736b79

COLLECT_GCC_OPTIONS='-save-temps' '-march=rv32i' '-mabi=ilp32' '-O1' '-v' '-c' '-
mtune=rocket' '-march=rv32i'

/home/katerina/riscv/lib/gcc/riscv32-unknown-elf/10.2.0/../../../../riscv32-unknown-elf/bin/as -v
--traditional-format -march=rv32i -march=rv32i -mabi=ilp32 -o main.o main.s

GNU assembler version 2.35 (riscv32-unknown-elf) using BFD version (GNU Binutils) 2.35

COLLECT_GCC_OPTIONS='-save-temps' '-march=rv32i' '-mabi=ilp32' '-O1' '-v' '-c' '-
mtune=rocket' '-march=rv32i'

/home/katerina/riscv/libexec/gcc/riscv32-unknown-elf/10.2.0/cc1 -E -quiet -v ./shift_array.c -
march=rv32i -mabi=ilp32 -mtune=rocket -march=rv32i -O1 -fpch-preprocess -o shift_array.i
ignoring nonexistent directory "/home/katerina/riscv/riscv32-unknown-elf/usr/local/include"
ignoring duplicate directory "/home/katerina/riscv/riscv32-unknown-elf/include"
#include "... " search starts here:
#include <...> search starts here:

/home/katerina/riscv/lib/gcc/riscv32-unknown-elf/10.2.0/include
/home/katerina/riscv/lib/gcc/riscv32-unknown-elf/10.2.0/include-fixed
/home/katerina/riscv/lib/gcc/riscv32-unknown-elf/10.2.0/../../../../riscv32-unknown-elf/include

End of search list.

COLLECT_GCC_OPTIONS='-save-temps' '-march=rv32i' '-mabi=ilp32' '-O1' '-v' '-c' '-
mtune=rocket' '-march=rv32i'

/home/katerina/riscv/libexec/gcc/riscv32-unknown-elf/10.2.0/cc1 -fpreprocessed shift_array.i -
quiet -dumpbase shift_array.c -march=rv32i -mabi=ilp32 -mtune=rocket -march=rv32i -
auxbase shift_array -O1 -version -o shift_array.s

GNU C17 (GCC) version 10.2.0 (riscv32-unknown-elf)

    compiled by GNU C version 10.2.0, GMP version 6.2.0, MPFR version 4.1.0, MPC
version 1.2.0-rc1, isl version none

GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsize=131072

GNU C17 (GCC) version 10.2.0 (riscv32-unknown-elf)

    compiled by GNU C version 10.2.0, GMP version 6.2.0, MPFR version 4.1.0, MPC
version 1.2.0-rc1, isl version none

GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsize=131072

Compiler executable checksum: b2aad7ee3dc27f7c0e5c7e63fc736b79

```

```
COLLECT_GCC_OPTIONS='-save-temps' '-march=rv32i' '-mabi=ilp32' '-O1' '-v' '-c' '-
mtune=rocket' '-march=rv32i'

/home/katerina/riscv/lib/gcc/riscv32-unknown-elf/10.2.0/../../../../riscv32-unknown-elf/bin/as -v
--traditional-format -march=rv32i -march=rv32i -mabi=ilp32 -o shift_array.o shift_array.s

GNU assembler version 2.35 (riscv32-unknown-elf) using BFD version (GNU Binutils) 2.35

COMPILER_PATH=/home/katerina/riscv/libexec/gcc/riscv32-unknown-elf/10.2.0:/home/
katerina/riscv/libexec/gcc/riscv32-unknown-elf/10.2.0:/home/katerina/riscv/libexec/gcc/
riscv32-unknown-elf:/home/katerina/riscv/lib/gcc/riscv32-unknown-elf/10.2.0:/home/
katerina/riscv/lib/gcc/riscv32-unknown-elf:/home/katerina/riscv/lib/gcc/riscv32-unknown-elf/
10.2.0/../../../../riscv32-unknown-elf/bin/

LIBRARY_PATH=/home/katerina/riscv/lib/gcc/riscv32-unknown-elf/10.2.0:/home/katerina/
riscv/lib/gcc/riscv32-unknown-elf/10.2.0/../../../../riscv32-unknown-elf/lib:/home/katerina/
riscv/riscv32-unknown-elf/lib/

COLLECT_GCC_OPTIONS='-save-temps' '-march=rv32i' '-mabi=ilp32' '-O1' '-v' '-c' '-
mtune=rocket' '-march=rv32i'
```

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 5: файл main.i

```
# 2 "./main.c" 2
# 1 "./shift_array.h" 1
# 1 "./shift_array.h"
void shiftArray(int array[], int shift, int length);
# 3 "./main.c" 2

int main() {
    int array[3] = {1, 2, 3};
    const int shift = 2;
    int length = 3;
    shiftArray(array, shift, length);

    for (int i = 0; i < 3; ++i) {
        printf("%d ", array[i]);
    }
}
```


Аналогично происходит препроцессирование функции:

Листинг 6: файл shift_array.i

```
# 1 "./shift_array.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "./shift_array.c"
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;
    }
}
```

3.2. Анализ выхода компилятора:

Листинг 7: 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 "%d "
.text
.align 2
.globl main
.type main, @function
main:
    addi    sp,sp,-32
    sw      ra,28(sp)
    sw      s0,24(sp)
    li      a5,1
    sw      a5,4(sp)
    li      a5,2
```

```

sw    a5,8(sp)
li    a5,3
sw    a5,12(sp)
li    a2,3
li    a1,2
addi  a0,sp,4
call  shiftArray
lw    a1,4(sp)
lui   s0,%hi(.LC0)
addi  a0,s0,%lo(.LC0)
call  printf
lw    a1,8(sp)
addi  a0,s0,%lo(.LC0)
call  printf
lw    a1,12(sp)
addi  a0,s0,%lo(.LC0)
call  printf
li    a0,0
lw    ra,28(sp)
lw    s0,24(sp)
addi  sp,sp,32
jr    ra
.size  main,.-main
.ident "GCC: (GNU) 10.2.0"

```

Листинг 8: файл shift_array.s

```

.file    "shift_array.c"
.option nopic
.attribute arch, "rv32i2p0"
.attribute unaligned_access, 0
.attribute stack_align, 16
.text
.align 2
.globl shiftArray
.type   shiftArray, @function
shiftArray:
    ble    a1,zero,.L1
    slli   a5,a2,2
    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    a2,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   a1,a6,.L5
.L1:
        ret
.size    shiftArray, .-shiftArray
.ident   "GCC: (GNU) 10.2.0"

```

В программе main выполняется обращение к подпрограмме shift_array (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 shift_array.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 -f main.o
main.o:      file format elf32-littleriscv
architecture: riscv:rv32, flags 0x00000011:
HAS_RELOC, HAS_SYMS
start address 0x00000000

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          00000080  00000000  00000000  00000034  2**2
CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data          00000000  00000000  00000000  000000b4  2**0
CONTENTS, ALLOC, LOAD, DATA
  2 .bss           00000000  00000000  00000000  000000b4  2**0
ALLOC
  3 .rodata.str1.4 00000004  00000000  00000000  000000b4  2**2
CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .comment       00000013  00000000  00000000  000000b8  2**0
CONTENTS, READONLY
  5 .riscv.attributes 0000001c  00000000  00000000  000000cb  2**0
CONTENTS, READONLY
```

Рис.2 Содержимое заголовков секции main.o

В файле имеются следующие секции:

- .text – секция кода;
- .data – секция инициализированных данных;
- .bss – секция данных, инициализированных нулями;
- .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 -f shift_array.o

shift_array.o:  file format elf32-littleriscv
architecture: riscv:rv32, flags 0x00000011:
HAS_RELOC, HAS_SYMS
start address 0x00000000

katerina@pop-os:~/Documents/Programming/Turing/shift_array$ riscv32-unknown-elf-objdump -h shift_array.o

shift_array.o:  file format elf32-littleriscv

Sections:
Idx Name          Size      VMA           LMA           File off  Algn
 0 .text          0000004c  00000000  00000000  00000034  2**2
CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
 1 .data          00000000  00000000  00000000  00000080  2**0
CONTENTS, ALLOC, LOAD, DATA
 2 .bss           00000000  00000000  00000000  00000080  2**0
ALLOC
 3 .comment       00000013  00000000  00000000  00000080  2**0
CONTENTS, READONLY
 4 .riscv.attributes 0000001c  00000000  00000000  00000093  2**0
CONTENTS, READONLY

В файле "main.o" имеются следующие секции:
.text – секция кода, в которой
обусловлено историческими причинами);
.data – секция инициализированных данных;
.bss – секция данных, инициализированных историческими причинами);
.comment – секция данных о версиях
Как можно видеть из колонки "Size" (значения в скобках) – размер секции ".text" – 8 байт, размер секции ".data" – 0 байт, размер секции ".bss" – 0 байт.
Теперь изучим таблицу символов файла:

```

Рис.3. Содержимое заголовков секций shift_array.o

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

```

katerina@pop-os:~/Documents/Programming/Turing/shift_array$ riscv32-unknown-elf-objdump -t main.o shift_array.o

main.o:  file format elf32-littleriscv

SYMBOL TABLE:
00000000 l d f *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 d .rodata.str1.4 00000000 .LC0
00000000 l d .comment 00000000 .comment
00000000 l d .riscv.attributes 00000000 .riscv.attributes
00000000 g F .text 00000080 main
00000000 *UND* 00000000 shiftArray
00000000 *UND* 00000000 printf

shift_array.o:  file format elf32-littleriscv

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

В файле "main.o" имеются следующие секции:
.text – секция кода, в которой содержится код программы;
.data – секция инициализированных данных;
.bss – секция данных, инициализированных историческими причинами);
.comment – секция данных о версиях
Как можно видеть из колонки "Size" (значения в скобках) – размер секции ".text" – 8 байт, размер секции ".data" – 0 байт, размер секции ".bss" – 0 байт.

```

Рис.4. Содержимое таблиц символов объектных файлов

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

UND означает, что символы printf 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 kSearch.o:

```

katerina@pop-os:~/Documents/Programming/Turing/shift_array$ riscv32-unknown-elf-objdump -d -M no-aliases -j .text 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: 00100793      addi    a5,zero,1
10: 00f12223      sw      a5,4(sp)
14: 00200793      addi    a5,zero,2
18: 00f12423      sw      a5,8(sp)
1c: 00300793      addi    a5,zero,3
20: 00f12623      sw      a5,12(sp)
24: 00300613      addi    a2,zero,3
28: 00200593      addi    a1,zero,2
2c: 00410513      addi    a0,sp,4
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: 00812583      lw      a1,8(sp)
50: 00040513      addi    a0,s0,0
54: 00000097      auipc   ra,0x0
58: 000080e7      jalr    ra,0(ra) # 54 <main+0x54>
5c: 00c12583      lw      a1,12(sp)
60: 00040513      addi    a0,s0,0
64: 00000097      auipc   ra,0x0
68: 000080e7      jalr    ra,0(ra) # 64 <main+0x64>
6c: 00000513      addi    a0,zero,0
70: 01c12083      lw      ra,28(sp)
74: 01812403      lw      s0,24(sp)
78: 02010113      addi    sp,sp,32
7c: 00008067      jalr    zero,0(ra)
  
```

Теперь мы можем записать инструкции программы на языке ассемблера.

Перепишем инструкции, используя обозначения регистров, по «RISC-V Assembler and ABI Basics»)

addi a0, zero, 0 ; a0 = 0 – возвращаемое значение
jalr zero, 0(ra) ; возврат из подпрограммы – по регистру ra; записанный вызывающей подпрограммой.

Разумеется, процедура декодирования кодов инструкций является ее реализовывало техническое устройство – процессор), следовательно выполнение ЭВМ:

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

Опция “-d” инициирует процесс дизассемблирования (disassembly) требует использовать в выводе только инструкции системы кода ассемблера). Вывод утилиты:

```

main.o:      file format elf32-littleriscv

Disassembly of section .text:

00000000 <main>:
0: 00000513      addi    a0,zero,0
4: 00008067      jalr    zero,0(ra)
  
```

Секция .comment

Изучим содержимое секции “.comment”:

```

riscv64-unknown-elf-objdump -s -j .comment main.o
  
```

Рис.5. Содержимое секции .text объектного файла main.o

```

katerina@pop-os: ~/Documents/Programming/Turing/shift_array$ riscv32-unknown-elf-objdump -d -M no-aliases -j .text shift_array.o
shift_array.o:      file format elf32-littleriscv

Disassembly of section .text:

00000000 <shiftArray>:
 0: 04b05463      bge     zero,a1,48 <.L1>
 4: 00261793      slli   a5,a2,0x2
 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: 00c35c63      bge    t1,a2,3c <.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: fd059ee3      bne    a1,a6,20 <.L5>

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

```

Рис.6. Содержимое секции .text объектного файла kSearch.o

Результат дизассемблирования shift_array.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:

30:	00000097	auipc ra,0x0
34:	000080e7	jalr ra,0(ra) # 30 <main+0x30>

Результатом выполнения этой пары инструкций станет переход на адрес main+0x30 (0+30=30) - 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:

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

RELOCATION RECORDS FOR [.text]:
OFFSET      TYPE             VALUE
00000030    R_RISCV_CALL             shiftArray
00000030    R_RISCV_RELAX            *ABS*
0000003c    R_RISCV_HI20             .LC0
0000003c    R_RISCV_RELAX            *ABS*
00000040    R_RISCV_L012_I          .LC0
00000040    R_RISCV_RELAX            *ABS*
00000044    R_RISCV_CALL             printf
00000044    R_RISCV_RELAX            *ABS*
00000050    R_RISCV_L012_I          .LC0
00000050    R_RISCV_RELAX            *ABS*
00000054    R_RISCV_CALL             printf
00000054    R_RISCV_RELAX            *ABS*
00000060    R_RISCV_L012_I          .LC0
00000060    R_RISCV_RELAX            *ABS*
00000064    R_RISCV_CALL             printf
00000064    R_RISCV_RELAX            *ABS*
```

Рис.7. Таблица перемещений main.o

```
katerina@pop-os:~/Documents/Programming/Turing/shift_array$ riscv32-unknown-elf-objdump -r shift_array.o
shift_array.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.8. Таблица перемещений shift_array.o

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

Содержимое shift_array.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.


```
katerina@pop-os:~/Documents/Programming/Turing/shift_array$ riscv32-unknown-elf-objdump -d -M no-aliases -r main.o shift_array.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: 00100793      addi    a5,zero,1
10: 00f12223      sw      a5,4(sp)
14: 00200793      addi    a5,zero,2
18: 00f12423      sw      a5,8(sp)
1c: 00300793      addi    a5,zero,3
20: 00f12623      sw      a5,12(sp)
24: 00300613      addi    a2,zero,3
28: 00200593      addi    a1,zero,2
2c: 00410513      addi    a0,sp,4
30: 00000097      auipc   ra,0x0
30: R_RISCV_CALL      shiftArray
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      .LC0
3c: R_RISCV_RELAX      *ABS*
40: 00040513      addi    a0,s0,0 # 0 <main>
40: R_RISCV_L012_I     .LC0
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: 00812583      lw      a1,8(sp)
50: 00040513      addi    a0,s0,0
50: R_RISCV_L012_I     .LC0
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: 00c12583      lw      a1,12(sp)
60: 00040513      addi    a0,s0,0
60: R_RISCV_L012_I     .LC0
```

Таблица перемещений

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

```
riscv64-unknown-elf-objdump -r zero.o main.o
```

Вывод утилиты:

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

RELOCATION RECORDS FOR [.text]:

OFFSET	TYPE	VALUE
00000008	R_RISCV_CALL	zero
00000008	R_RISCV_RELAX	*ABS*

Содержимое "zero.o" не требует модификации, поэтому не содержит записей (relocation entries). В файле же "main.o" имеется две записи, относящиеся к инструкциям, которые мы видели выше, по этому адресу в "main.o" находится первая инструкция. Дизассемблирование и вывод таблицы перемещений можно совместить:

```
riscv64-unknown-elf-objdump -d -M no-aliases -r main.o
```

Вывод утилиты:

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

Disassembly of section .text:

```
00000000 <main>:
0: ff010113      addi    sp,sp,-16
4: 00112e23      sw      ra,12(sp)
8: 00000097      auipc   ra,0x0
8: R_RISCV_CALL      zero
8: R_RISCV_RELAX      *ABS*
```

Рис.9. Таблицы перемещений main.o и shift_array.o

```
60: 00040513      addi    a0,s0,0
60: R_RISCV_L012_I     .LC0
60: R_RISCV_RELAX      *ABS*
64: 00000097      auipc   ra,0x0
64: R_RISCV_CALL      printf
64: R_RISCV_RELAX      *ABS*
```

Рис.9.1. Таблицы перемещений main.o и shift_array.o (продолжение)

```

64: R_RISCV_CALL      printf
64: R_RISCV_RELAX     *ABS*
68: 000080e7          jalr    ra,0(ra) # 64 <main+0x64>
6c: 00000513          addi    a0,zero,0
70: 01c12083          lw      ra,28(sp)
74: 01812403          lw      s0,24(sp)
78: 02010113          addi    sp,sp,32
7c: 00008067          jalr    zero,0(ra)

shift_array.o:      file format elf32-littleriscv

Disassembly of section .text:

00000000 <shiftArray>:
0: 04b05463          bge     zero,a1,48 <.L1>
0: R_RISCV_BRANCH    .L1
4: 00261793          slli    a5,a2,0x2
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: 00c35c63          bge     t1,a2,3c <.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: fd059ee3          bne     a1,a6,20 <.L5>
44: R_RISCV_BRANCH    .L5

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

```

Рис.9.2. Таблицы перемещений main.o и shift_array.o (продолжение)

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 `shift_array`. 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

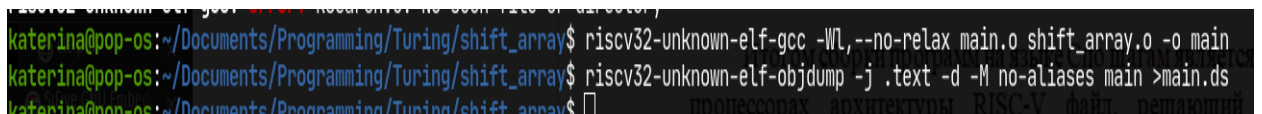


Рис.10. Компоновка

```
riscv32-unknown-elf-gcc -Wl,--no-relax main.o shift_array.o -o main
```

```
riscv32-unknown-elf-objdump -j .text -d -M no-aliases main >main.ds
```

Листинг 9: файл `main.ds` (строки 80-133)

```
00010178 <main>:
 10178:    fe010113      addi    sp,sp,-32
 1017c:    00112e23      sw     ra,28(sp)
 10180:    00812c23      sw     s0,24(sp)
 10184:    00100793      addi    a5,zero,1
 10188:    00f12223      sw     a5,4(sp)
 1018c:    00200793      addi    a5,zero,2
 10190:    00f12423      sw     a5,8(sp)
 10194:    00300793      addi    a5,zero,3
 10198:    00f12623      sw     a5,12(sp)
 1019c:    00300613      addi    a2,zero,3
 101a0:    00200593      addi    a1,zero,2
 101a4:    00410513      addi    a0,sp,4
 101a8:    00000097      auipc   ra,0x0
 101ac:    050080e7      jalr    ra,80(ra) # 101f8 <shiftArray>
 101b0:    00412583      lw     a1,4(sp)
 101b4:    00026437      lui    s0,0x26
 101b8:    e3840513      addi    a0,s0,-456 # 25e38 <__clzsi2+0x4c>
```

101bc:	00000097	auipc	ra,0x0
101c0:	280080e7	jalr	ra,640(ra) # 1043c <printf>
101c4:	00812583	lw	a1,8(sp)
101c8:	e3840513	addi	a0,s0,-456
101cc:	00000097	auipc	ra,0x0
101d0:	270080e7	jalr	ra,624(ra) # 1043c <printf>
101d4:	00c12583	lw	a1,12(sp)
101d8:	e3840513	addi	a0,s0,-456
101dc:	00000097	auipc	ra,0x0
101e0:	260080e7	jalr	ra,608(ra) # 1043c <printf>
101e4:	00000513	addi	a0,zero,0
101e8:	01c12083	lw	ra,28(sp)
101ec:	01812403	lw	s0,24(sp)
101f0:	02010113	addi	sp,sp,32
101f4:	00008067	jalr	zero,0(ra)
000101f8 <shiftArray>:			
101f8:	04b05463	bge	zero,a1,10240 <shiftArray+0x48>
101fc:	00261793	slli	a5,a2,0x2
10200:	ffc78793	addi	a5,a5,-4
10204:	00f50e33	add	t3,a0,a5
10208:	00478793	addi	a5,a5,4
1020c:	00f506b3	add	a3,a0,a5
10210:	00000813	addi	a6,zero,0
10214:	00100313	addi	t1,zero,1
10218:	00052883	lw	a7,0(a0)
1021c:	00c35c63	bge	t1,a2,10234 <shiftArray+0x3c>
10220:	00450793	addi	a5,a0,4
10224:	0007a703	lw	a4,0(a5)
10228:	fee7ae23	sw	a4,-4(a5)
1022c:	00478793	addi	a5,a5,4
10230:	fed79ae3	bne	a5,a3,10224 <shiftArray+0x2c>
10234:	011e2023	sw	a7,0(t3)
10238:	00180813	addi	a6,a6,1
1023c:	fd059ee3	bne	a1,a6,10218 <shiftArray+0x20>
10240:	00008067	jalr	zero,0(ra)

Прежде всего можно видеть, что в результат компоновки попало содержимое обоих объектных файлов – main.o и shift_array.o. Инструкции подпрограммы shift_array начинаются с адреса 101f8₁₆, и пара инструкций auipc+jalr, вызывающих подпрограмму shift_array соответствующим образом откорректированы.

3.6. Анализ отладочной информации

```
katerina@pop-os:~/Documents/Programming/learning/shift_array$ riscv32-unknown-elf-objdump -t -h main
main:      file format elf32-littleriscv
architecture: riscv:rv32, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x00010094

Sections:
Idx Name          Size      VMA           LMA           File off  Algn  Flags
0  .text          00015dc4  00010074  00010074  00000074  2**2  0
CONTENTS, ALLOC, LOAD, READONLY, CODE
1  .rodata        00000dd4  00025e38  00025e38  00015e38  2**3  0
CONTENTS, ALLOC, LOAD, READONLY, DATA
2  .eh_frame      000000b4  00027000  00027000  00017000  2**2  0
CONTENTS, ALLOC, LOAD, DATA
3  .init_array    00000008  000270b4  000270b4  000170b4  2**2  0
CONTENTS, ALLOC, LOAD, DATA
4  .fini_array    00000004  000270bc  000270bc  000170bc  2**2  0
CONTENTS, ALLOC, LOAD, DATA
5  .data          0000099c  000270c0  000270c0  000170c0  2**3  0
CONTENTS, ALLOC, LOAD, DATA
6  .sdata         0000002c  00027a60  00027a60  00017a60  2**3  0
CONTENTS, ALLOC, LOAD, DATA
7  .sbss          00000018  00027a8c  00027a8c  00017a8c  2**2  0
ALLOC
8  .bss           00000044  00027aa4  00027aa4  00017a8c  2**2  0
ALLOC
9  .comment       00000012  00000000  00000000  00017a8c  2**0  0
CONTENTS, READONLY
10 .riscv.attributes 0000001c  00000000  00000000  00017a9e  2**0  0
CONTENTS, READONLY
11 .debug_aranges 00000218  00000000  00000000  00017ac0  2**3  0
CONTENTS, READONLY, DEBUGGING, OCTETS
12 .debug_info    00006a79  00000000  00000000  00017cd8  2**0  0
CONTENTS, READONLY, DEBUGGING, OCTETS
13 .debug_abbrev  00001671  00000000  00000000  0001e751  2**0  0
CONTENTS, READONLY, DEBUGGING, OCTETS
14 .debug_line    0000a41a  00000000  00000000  0001fdc2  2**0  0
CONTENTS, READONLY, DEBUGGING, OCTETS
15 .debug_frame   00000308  00000000  00000000  0002a1dc  2**2  0
CONTENTS, READONLY, DEBUGGING, OCTETS
16 .debug_str     00000dd8  00000000  00000000  0002a4e4  2**0  0
CONTENTS, READONLY, DEBUGGING, OCTETS
17 .debug_loc     00008826  00000000  00000000  0002b2bc  2**0  0
CONTENTS, READONLY, DEBUGGING, OCTETS
18 .debug_ranges  00001630  00000000  00000000  00033ae2  2**0  0
CONTENTS, READONLY, DEBUGGING, OCTETS
```

Рис.11. 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 атрибутах *DW_AT_ranges*;
- *.debug_str* – таблица строк, используемая в *.debug_info*.

3.7. Выделение разработанной функции в статическую библиотеку

```
katerina@pop-os:~/Documents/Programming/Turing/shift_array$ riscv32-unknown-elf-ar -rsc shift_array.a shift_array.o
katerina@pop-os:~/Documents/Programing/Turing/shift_array$ riscv32-unknown-elf-gcc -01 --save-temps main.c shift_array.a -o mainWithLib
riscv32-unknown-elf-gcc: error: unrecognized command-line option '-01'
katerina@pop-os:~/Documents/Programming/Turing/shift_array$ riscv32-unknown-elf-gcc -01 --save-temps main.c shift_array.a -o mainWithLib
katerina@pop-os:~/Documents/Programming/Turing/shift_array$ riscv32-unknown-elf-objdump -t main
main:      file format elf32-littleriscv
```

Рис.12. Выделение разработанной функции в статическую библиотеку.

Листинг 10: таблица символов полученного исполняемого файла.

```
....
0001031c g    F .text    000000dc memset
00010178 g    F .text    00000080 main
....
000101f8 g    F .text    0000004c shiftArray
....
```

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-mn shift_array.a
riscv32-unknown-elf-mn: command not found
katerina@pop-os:~/Documents/Programing/Turing/shift_array$ riscv32-unknown-elf-nm shift_array.a
shift_array.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.13. Список символов библиотеки

Листинг 11: make-файл main

```

build:
    gcc -c shift_array.c
    ar -rcs shift_array.a shift_array.o

    gcc -c main.c
    gcc -o main main.o -L. -l:shift_array.a

clean:
    rm *.o

```

Что происходит в *makefile*:

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

4. Выводы.

В ходе работы исследован процесс сборки проекта на языке С.

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`, 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.