Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Лабораторная работа № 4**

**Дисциплина**: Низкоуровневое программирование

**Тема:** Раздельная компиляция

Выполнил студент гр. 3530901/10005 Л. С. Алексеев

Преподаватель Коренев Д. А.

«30» ноября 2022 г.

Санкт-Петербург

2022

Оглавление

[**Техническое задание** 3](#_Toc120300519)

[**1. Программа на языке C** 4](#_Toc120300520)

[**2. Сборка программы «по шагам»** 5](#_Toc120300521)

[Препроцессирование 5](#_Toc120300522)

[Компиляция 6](#_Toc120300523)

[Ассемблирование 8](#_Toc120300524)

[Компоновка 11](#_Toc120300525)

[**3. Создание статической библиотеки и make-файлов** 13](#_Toc120300526)

[**Вывод** 16](#_Toc120300527)

**Техническое задание**

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 обмена в массиве элементов попарно. Если 1-е число больше 2-го меняем их местами. Если 3-е число больше 4-го меняем их местами. И т.д. 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 исполнимом файле.

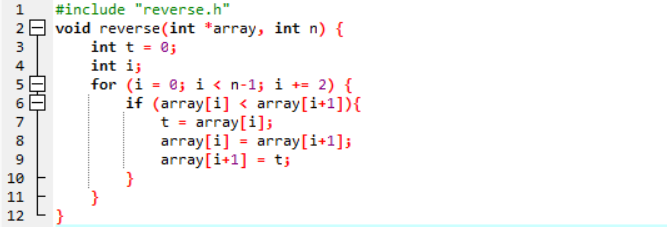
5. Выделить разработанную функцию в статическую библиотеку. Разработать make-файлы для сборки библиотеки и использующей ее тестовой программы. Проанализировать ход сборки библиотеки и программы, созданные файлы зависимостей.

**1. Программа на языке C**

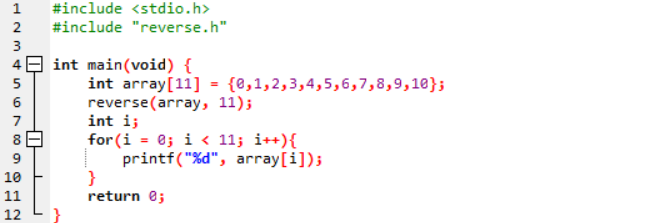
Листинг 1.1. Заголовочный файл reverse.h

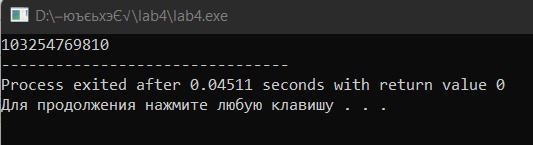


Листинг 1.2. Основной файл reverse.c



Листинг 1.3. Тестовая программа main.c





**2. Сборка программы «по шагам»**

Препроцессирование

Препроцессирование, компиляция и ассемблирование выполняются следующими командами:

riscv64-unknown-elf-gcc --save-temps -march=rv32i -mabi=ilp32 -O1 -v main.c >log 2>&1

riscv64-unknown-elf-gcc --save-temps -march=rv32i -mabi=ilp32 -O1 -v reverse.c >log 2>&1

Результат препроцессирования содержится в файлах main.i и reverse.i*.* По причине того, что main.c содержит заголовочный файл стандартной библиотеки языка С stdio.h, результат препроцессирования этого файла имеет достаточно много добавочных строк.

Листинг 2.1. Файл main.i (фрагмент)

# 1 "C:\\Users\\79006\\lab4\\main.c"

# 1 "<built-in>"

# 1 "<command-line>"

# 1 "C:\\Users\\79006\\lab4\\main.c"

……….

# 2 "C:\\Users\\79006\\lab4\\main.c" 2

# 1 "C:\\Users\\79006\\lab4\\reverse.h" 1

# 3 "C:\\Users\\79006\\lab4\\reverse.h"

void reverse(int \*array, int n);

# 3 "C:\\Users\\79006\\lab4\\main.c" 2

int main(void) {

int array[11] = {0,1,2,3,4,5,6,7,8,9,10};

reverse(array, 11);

int i;

for(i = 0; i < 11; i++){

printf("%d", array[i]);

}

return 0;

}

Листинг 2.2. Файл reverse.i

# 1 "D:\\Документы\\lab4\\reverse.c"

# 1 "<built-in>"

# 1 "<command-line>"

# 1 "D:\\Документы\\lab4\\reverse.c"

# 1 "D:\\Документы\\lab4\\reverse.h" 1

void reverse(int \*array, int n);

# 2 "D:\\Документы\\lab4\\reverse.c" 2

void reverse(int \*array, int n) {

int t = 0;

int i;

for (i = 0; i < n-1; i += 2) {

if (array[i] < array[i+1]){

t = array[i];

array[i] = array[i+1];

array[i+1] = t;

}

}

}

Компиляция

Наибольший интерес представляет файл main.s, так как в нем можно заметить обращение к подпрограмме reverse(значение регистра *ra*, содержащее адрес возврата из main, сохраняется на время вызова в стеке).

Листинг 2.3. 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

.LC1:

.string "%d"

.text

.align 2

.globl main

.type main, @function

main:

addi sp,sp,-64

sw ra,60(sp)

sw s0,56(sp)

sw s1,52(sp)

sw s2,48(sp)

lui a5,%hi(.LANCHOR0)

addi a5,a5,%lo(.LANCHOR0)

lw t4,0(a5)

lw t3,4(a5)

lw t1,8(a5)

lw a7,12(a5)

lw a6,16(a5)

lw a0,20(a5)

lw a1,24(a5)

lw a2,28(a5)

lw a3,32(a5)

lw a4,36(a5)

lw a5,40(a5)

sw t4,4(sp)

sw t3,8(sp)

sw t1,12(sp)

sw a7,16(sp)

sw a6,20(sp)

sw a0,24(sp)

sw a1,28(sp)

sw a2,32(sp)

sw a3,36(sp)

sw a4,40(sp)

sw a5,44(sp)

li a1,11

addi a0,sp,4

call reverse

addi s0,sp,4

addi s2,sp,48

lui s1,%hi(.LC1)

.L2:

lw a1,0(s0)

addi a0,s1,%lo(.LC1)

call printf

addi s0,s0,4

bne s0,s2,.L2

li a0,0

lw ra,60(sp)

lw s0,56(sp)

lw s1,52(sp)

lw s2,48(sp)

addi sp,sp,64

jr ra

.size main, .-main

.section .rodata

.align 2

.set .LANCHOR0,. + 0

.LC0:

.word 0

.word 1

.word 2

.word 3

.word 4

.word 5

.word 6

.word 7

.word 8

.word 9

.word 10

.ident "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"

Листинг 2.4. Файл reverse.s

.file "reverse.c"

.option nopic

.attribute arch, "rv32i2p0"

.attribute unaligned\_access, 0

.attribute stack\_align, 16

.text

.align 2

.globl reverse

.type reverse, @function

reverse:

li a5,1

ble a1,a5,.L1

mv a5,a0

addi a2,a1,-2

srli a2,a2,1

slli a2,a2,3

addi a0,a0,8

add a2,a2,a0

j .L4

.L3:

addi a5,a5,8

beq a5,a2,.L1

.L4:

lw a4,0(a5)

lw a3,4(a5)

bge a4,a3,.L3

sw a3,0(a5)

sw a4,4(a5)

j .L3

.L1:

ret

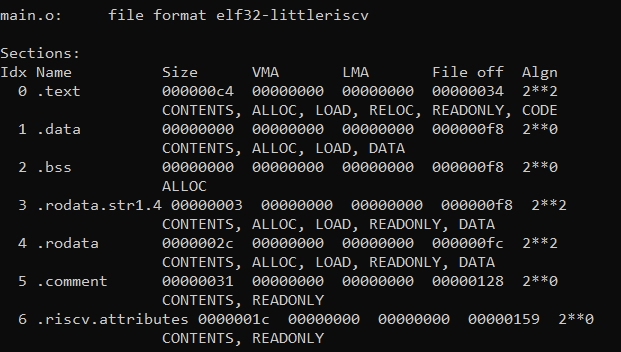
.size reverse, .-reverse

.ident "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"

Ассемблирование

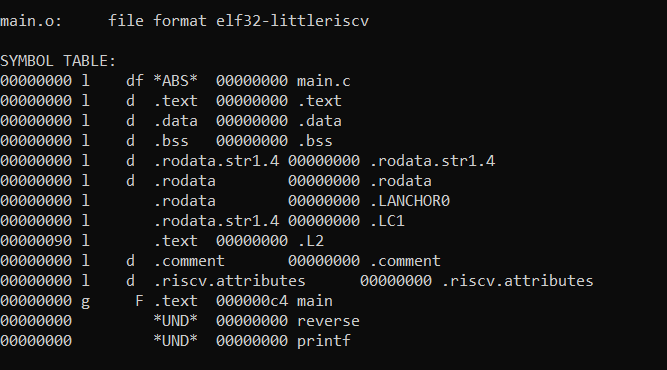
Листинг 2.5. Заголовки секций файла main.o

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



Листинг 2.6. Таблица символов файла main.o

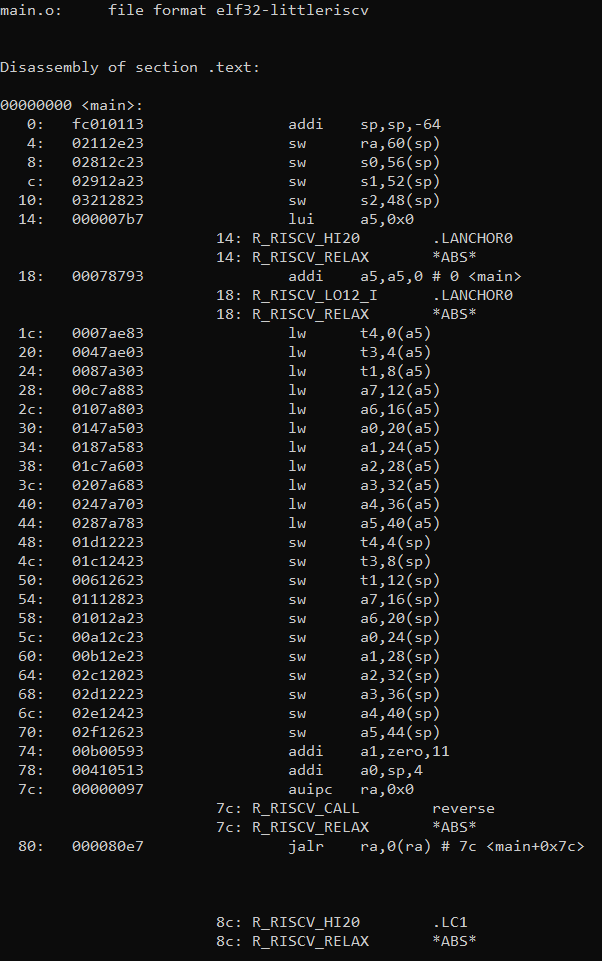
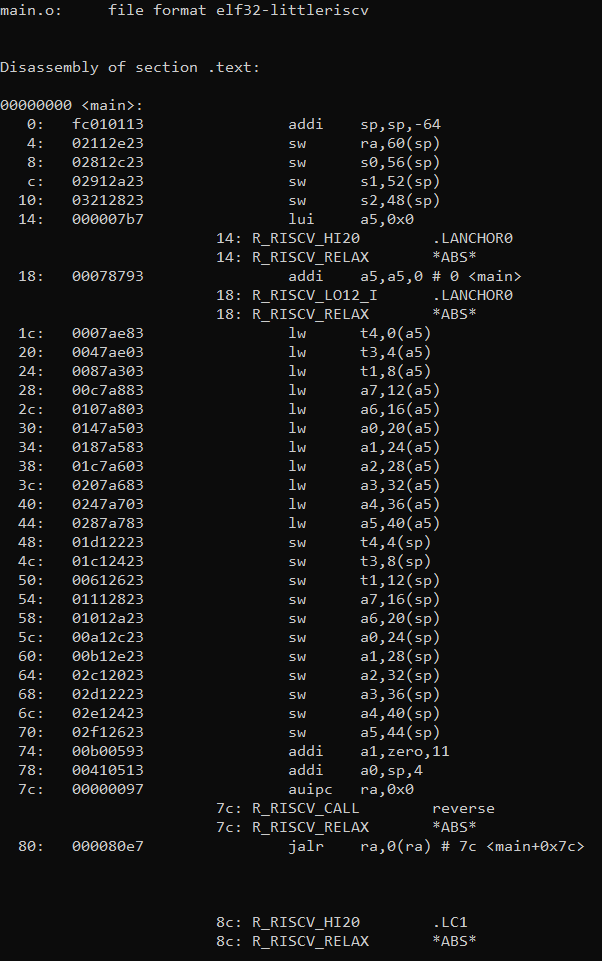
riscv64-unknown-elf-objdump.exe –t main.o

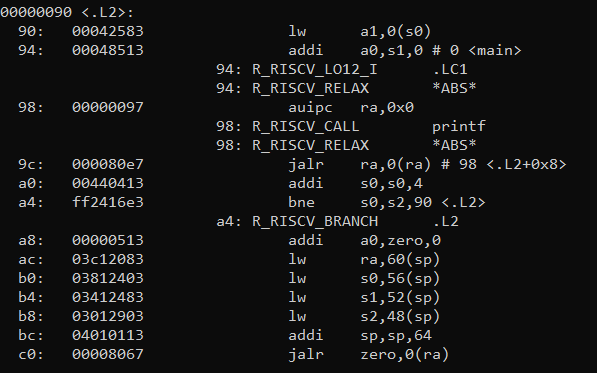


В таблице символов main.o имеется запись: символ “reverse” типа \*UND\*. Эта запись означает, что символ “reverse” использовался в ассемблерном коде, из которого был получен данный объектный файл, но не был определен; ассемблер сделал вывод о том, что символ должен быть определен где-то еще, и отразил это в таблице символов. То же самое относится и к символу “printf”.

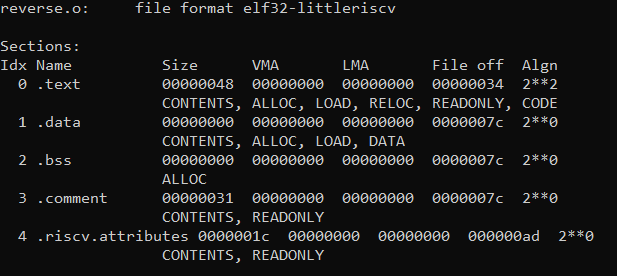
Листинг 2.7. Таблица перемещений файла main.o

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

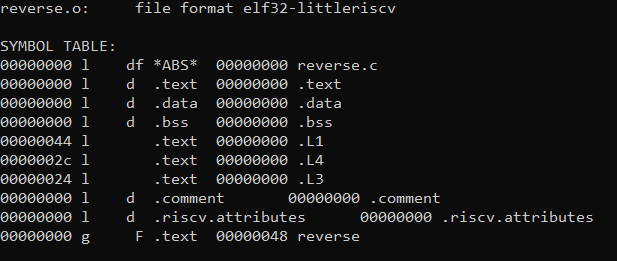




Листинг 2.8. Заголовки секций файла reverse.o



Листинг 2.9. Таблица символов файла reverse.o



Компоновка

Компоновка осуществляется следующей командой:

riscv64-unknown-elf-gcc.exe -v main.o reverse.o

Листинг 2.10. Исполняемый файл a.out (фрагмент)

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

a.out: file format elf32-littleriscv

Disassembly of section .text:

00010074 <register\_fini>:

10074: ffff0797 auipc a5,0xffff0

10078: f8c78793 addi a5,a5,-116 # 0 <register\_fini-0x10074>

1007c: 00078863 beq a5,zero,1008c <register\_fini+0x18>

10080: 00000517 auipc a0,0x0

10084: 21050513 addi a0,a0,528 # 10290 <\_\_libc\_fini\_array>

10088: 1c00006f jal zero,10248 <atexit>

1008c: 00008067 jalr zero,0(ra)

…

00010144 <reverse>:

10144: 00100793 addi a5,zero,1

10148: 04b7d063 bge a5,a1,10188 <reverse+0x44>

1014c: 00050793 addi a5,a0,0

10150: ffe58593 addi a1,a1,-2

10154: 0015d593 srli a1,a1,0x1

10158: 00359593 slli a1,a1,0x3

1015c: 00850513 addi a0,a0,8

10160: 00a58533 add a0,a1,a0

10164: 00c0006f jal zero,10170 <reverse+0x2c>

10168: 00878793 addi a5,a5,8

1016c: 00a78e63 beq a5,a0,10188 <reverse+0x44>

10170: 0007a703 lw a4,0(a5)

10174: 0047a683 lw a3,4(a5)

10178: fed758e3 bge a4,a3,10168 <reverse+0x24>

1017c: 00d7a023 sw a3,0(a5)

10180: 00e7a223 sw a4,4(a5)

10184: fe5ff06f jal zero,10168 <reverse+0x24>

10188: 00008067 jalr zero,0(ra)

0001018c <main>:

1018c: fc010113 addi sp,sp,-64

10190: 02112e23 sw ra,60(sp)

10194: 02812c23 sw s0,56(sp)

10198: 02912a23 sw s1,52(sp)

1019c: 03212823 sw s2,48(sp)

101a0: 000257b7 lui a5,0x25

101a4: 96878793 addi a5,a5,-1688 # 24968 <\_\_clzsi2+0x50>

101a8: 0007ae83 lw t4,0(a5)

101ac: 0047ae03 lw t3,4(a5)

101b0: 0087a303 lw t1,8(a5)

101b4: 00c7a883 lw a7,12(a5)

101b8: 0107a803 lw a6,16(a5)

101bc: 0147a503 lw a0,20(a5)

101c0: 0187a583 lw a1,24(a5)

101c4: 01c7a603 lw a2,28(a5)

101c8: 0207a683 lw a3,32(a5)

101cc: 0247a703 lw a4,36(a5)

101d0: 0287a783 lw a5,40(a5)

101d4: 01d12223 sw t4,4(sp)

101d8: 01c12423 sw t3,8(sp)

101dc: 00612623 sw t1,12(sp)

101e0: 01112823 sw a7,16(sp)

101e4: 01012a23 sw a6,20(sp)

101e8: 00a12c23 sw a0,24(sp)

101ec: 00b12e23 sw a1,28(sp)

101f0: 02c12023 sw a2,32(sp)

101f4: 02d12223 sw a3,36(sp)

101f8: 02e12423 sw a4,40(sp)

101fc: 02f12623 sw a5,44(sp)

10200: 00b00593 addi a1,zero,11

10204: 00410513 addi a0,sp,4

10208: f3dff0ef jal ra,10144 <reverse>

1020c: 00410413 addi s0,sp,4

10210: 03010913 addi s2,sp,48

10214: 000254b7 lui s1,0x25

10218: 00042583 lw a1,0(s0)

1021c: 99448513 addi a0,s1,-1644 # 24994 <\_\_clzsi2+0x7c>

10220: 284000ef jal ra,104a4 <printf>

10224: 00440413 addi s0,s0,4

10228: ff2418e3 bne s0,s2,10218 <main+0x8c>

1022c: 00000513 addi a0,zero,0

10230: 03c12083 lw ra,60(sp)

10234: 03812403 lw s0,56(sp)

10238: 03412483 lw s1,52(sp)

1023c: 03012903 lw s2,48(sp)

10240: 04010113 addi sp,sp,64

10244: 00008067 jalr zero,0(ra)

**3. Создание статической библиотеки и make-файлов**

Сделаем из reverse.c статическую библиотеку revlib, тестовую программу main.c оставим без изменений.

Для создания статической библиотеки получим объектный файл reverse.o.

riscv64-unknown-elf-gcc.exe -O1 -c reverse.c -o reverse.o

Сделаем из получившегося файла библиотеку следующей командой:

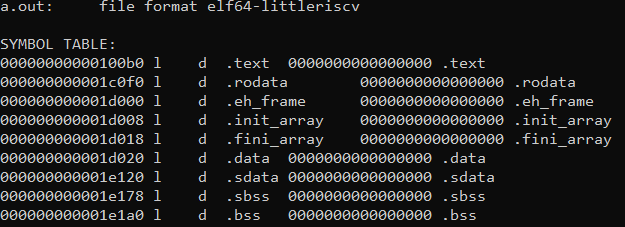
riscv64-unknown-elf-ar.exe -rsc revlib.a reverse.o

Используя получившуюся библиотеку, соберем исполняемый файл программы следующей командой:

riscv64-unknown-elf-gcc.exe -O1 --save-temps main.c revlib.a

Листинг 3.1. Таблица символов исполняемого файла (фрагмент)

riscv64-unknown-elf-objdump.exe -t a.out









Можно заметить, что в состав программы вошло содержимое объектного файла reverse.o.

Процесс выполнения команд выше можно заменить make-файлами, которые произведут создание библиотеки и сборку программы.

Листинг 3.2. Makefile для создания статической библиотеки

# "Фиктивные" цели

.PHONY: all clean

# Исходные файлы, необходимые для сборки библиотеки

OBJS= reverse.c

#Вызываемые приложения

AR = riscv64-unknown-elf-ar.exe

CC = riscv64-unknown-elf-gcc.exe

# Файл библиотеки

MYLIBNAME = revlib.a

# Параметры компиляции

CFLAGS= -O1

# Включаемые файлы следует искать в текущем каталоге

INCLUDES+= -I .

# Make должна искать файлы \*.h и \*.c в текущей директории

vpath %.h .

vpath %.c .

# Построение объектного файла из исходного текста

# $< = %.c

# $@ = %.o

%.o: %.c

$(CC) -MD $(CFLAGS) $(INCLUDES) -c $< -o $@

# Чтобы достичь цели "all", требуется построить библиотеку

all: $(MYLIBNAME)

# $^ = (reverse.o)

$(MYLIBNAME): reverse.o

$(AR) -rsc $@ $^

Листинг 3.3. Makefile для сборки исполняемого файла

# "Фиктивные" цели

.PHONY: all clean

# Файлы для сборки исполнимого файла

OBJS= main.c \

revlib.a

#Вызываемые приложения

CC = riscv64-unknown-elf-gcc.exe

# Параметры компиляции

CFLAGS= -O1 --save-temps

# Включаемые файлы следует искать в текущем каталоге

INCLUDES+= -I .

# Make должна искать файлы \*.c и \*.a в текущей директории

vpath %.c .

vpath %.a .

# Чтобы достичь цели "all", требуется собрать исполнимый файл

all: a.out

# Сборка исполнимого файла и удаление мусора

a.out: $(OBJS)

$(CC) $(CFLAGS) $(INCLUDES) $^

del \*.o \*.i \*.s \*.d

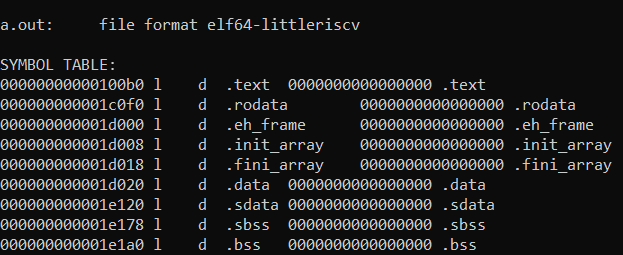
Для запуска Makefile воспользуемся программой mingw32-make.exe.

Листинг 3.4. Запуск Makefile

mingw32-make.exe -f Makefile.win Makefile1.win

Сначала мы запускаем Makefile.win со сборкой библиотеки, а затем Makefile1.win со сборкой исполняемого файла.

Листинг 3.5. Таблица символов исполняемого файла, созданного с помощью Makefile (фрагмент)

****

****

****

****

Видим, созданный исполняемый файл аналогичен тому, что был создан через терминал.

**Вывод**

В ходе выполнения лабораторной работы были закреплены знания языка C, ассемблера RISC-V, получены навыки работы с препроцессором, компилятором, ассемблером и компоновщиком пакета GCC и драйвером компилятора riscv64-unknown-elf-gcc. Были изучены особенности каждого этапа пошаговой сборки набора программ, а также инструменты, позволяющие выделить разработанные программы в статическую библиотеку и автоматизировать сборку этой библиотеки.

Была реализована поставленная задача – «функция обмена в массиве элементов попарно», а затем проведена проверка правильности перевода программы решения этой задачи в набор инструкций, выполняемых процессором.