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. 3530901/10002

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Котов К. А.

(подпись)

Руководитель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Максименко С. Л

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Санкт-Петербург   
2022

# **Задачи**

Реализовать заданную вариантом программу на языке C, содержащую заголовочный файл, файл тестирования и исходный файл алгоритма.

Пошагово собрать программу. Проанализировать выходы предпроцессора, компилятора, составы и содержимое секций таблиц символов и таблиц перемещений, отладочную информацию объектных и исполняемых файлов.

Создать статическую библиотеку разработанной функции, разработав make-файлы. Проанализировать ход сборки библиотеки и программы, созданные файлы зависимостей.

# **Вариант**

Реализовать алгоритм нахождения биномиальных коэффициентов по треугольнику Паскаля.

# **Решение программой на языке C**

Далее представлено содержимое файлов тестирования, алгоритма и заголовочного файла.

Файл main.c – содержит тестирование.

#include <stdlib.h>

#include <stdio.h>

#include "pascal.h"

int main() {

    int pokaz = -1;

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

    pascal(pokaz, array);

    for(int i = 0; i <= pokaz; i++) {

        printf("%d\t", \*(array + i));

    }

    printf("\n");

    pokaz = 0;

    int \*array1 = calloc(pokaz + 1, sizeof(int));

    pascal(pokaz, array1);

    for (size\_t i = 0; i <= pokaz; i++) {

        printf("%d\t", \*(array1 + i));

    }

    printf("\n");

    pokaz = 1;

    int \*array2 = calloc(pokaz + 1, sizeof(int));

    pascal(pokaz, array2);

    for (size\_t i = 0; i <= pokaz; i++) {

        printf("%d\t", \*(array2 + i));

    }

    printf("\n");

    pokaz = 10;

    int \*array3 = calloc(pokaz + 1, sizeof(int));

    pascal(pokaz, array3);

    for (size\_t i = 0; i <= pokaz; i++) {

        printf("%d\t", \*(array3 + i));

    }

    return 0;

}

Файл pascal.h – заголовочный файл

#include <stddef.h>

int pascal(int pokaz, int \*array);

Файл pascal.c – алгоритм

#include "pascal.h"

#include <stdio.h>

int pascal(int pokaz, int \*array) {

    if (pokaz < 0) {

        printf("pokaz must be greater than zero");

        return -1;

    }

    \*array = 1;

    int len = 2;

    int\* ptr = array;

    while (pokaz > 0) {

        for (int i = len - 1; i >= 1; i--) {

            \*(ptr + i) = \*(ptr + i - 1) + \*(ptr + i);

        }

        pokaz--;

        len++;

    }

    return 0;

}

Результат работы программы показан на рис. 1

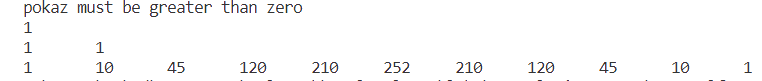


Рис.1 – Результат работы программы

**Раздельная компиляция**

1. **Предпроцессирование.** Предпроцессирование выполняется следующими командами.

>riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -E main.c -o main.i

>riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -E pascal.c -o pascal.i

Результаты работы команд находятся в файлах main.i и pascal.i соответственно.

Файл main.i:

...

# 2 "pascal.h"

int pascal(int pokaz, int \*array);

# 4 "main.c" 2

int main() {

    int pokaz = -1;

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

    pascal(pokaz, array);

    for(int i = 0; i <= pokaz; i++) {

        printf("%d\t", \*(array + i));

    }

    printf("\n");

    pokaz = 0;

    int \*array1 = calloc(pokaz + 1, sizeof(int));

    pascal(pokaz, array1);

    for (size\_t i = 0; i <= pokaz; i++) {

        printf("%d\t", \*(array1 + i));

    }

    printf("\n");

    pokaz = 1;

    int \*array2 = calloc(pokaz + 1, sizeof(int));

    pascal(pokaz, array2);

    for (size\_t i = 0; i <= pokaz; i++) {

        printf("%d\t", \*(array2 + i));

    }

    printf("\n");

    pokaz = 10;

    int \*array3 = calloc(pokaz + 1, sizeof(int));

    pascal(pokaz, array3);

    for (size\_t i = 0; i <= pokaz; i++) {

        printf("%d\t", \*(array3 + i));

    }

    return 0;

}

Файл pascal.i:

...

# 2 "pascal.h"

int pascal(int pokaz, int \*array);

# 3 "pascal.c" 2

int pascal(int pokaz, int \*array) {

    if (pokaz < 0) {

        printf("pokaz must be greater than zero");

        return -1;

    }

    \*array = 1;

    int len = 2;

    int\* ptr = array;

    while (pokaz > 0) {

        for (int i = len - 1; i >= 1; i--) {

            \*(ptr + i) = \*(ptr + i - 1) + \*(ptr + i);

        }

        pokaz--;

        len++;

    }

    return 0;

}

В прикрепленных листингах мы видим, что при предпроцессировании в обработанных файлах появилось определение функции pascal, которое появилось из заголовочного файла pascal.h. Кроме этого полученные файлы содержат много строк кода, которые появились в связи со включением в программу заголовочных файлов stdio.h и stdlib.h, которые позволяют нам использовать дополнительные функции, в нашем случае printf() и calloc().

1. **Компиляция**

Компиляция осуществляется следующими командами:

>riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -O1 -S main.i -o main.s

>riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -O1 -S pascal.i -o pascal.s

Коды соответствующих выходных файлов представлены далее.

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\t"

    .text

    .align  2

    .globl  main

    .type   main, @function

main:

    addi    sp,sp,-16

    sw  ra,12(sp)

    sw  s0,8(sp)

    sw  s1,4(sp)

    sw  s2,0(sp)

    li  a1,4

    li  a0,0

    call    calloc

    mv  a1,a0

    li  a0,-1

    call    pascal

    li  a0,10

    call    putchar

    li  a1,4

    li  a0,1

    call    calloc

    mv  s0,a0

    mv  a1,a0

    li  a0,0

    call    pascal

    lw  a1,0(s0)

    lui s1,%hi(.LC0)

    addi    a0,s1,%lo(.LC0)

    call    printf

    li  a0,10

    call    putchar

    li  a1,4

    li  a0,2

    call    calloc

    mv  s0,a0

    mv  a1,a0

    li  a0,1

    call    pascal

    lw  a1,0(s0)

    addi    a0,s1,%lo(.LC0)

    call    printf

    lw  a1,4(s0)

    addi    a0,s1,%lo(.LC0)

    call    printf

    li  a0,10

    call    putchar

    li  a1,4

    li  a0,11

    call    calloc

    mv  s1,a0

    mv  a1,a0

    li  a0,10

    call    pascal

    mv  s0,s1

    addi    s1,s1,44

    lui s2,%hi(.LC0)

.L2:

    lw  a1,0(s0)

    addi    a0,s2,%lo(.LC0)

    call    printf

    addi    s0,s0,4

    bne s1,s0,.L2

    li  a0,0

    lw  ra,12(sp)

    lw  s0,8(sp)

    lw  s1,4(sp)

    lw  s2,0(sp)

    addi    sp,sp,16

    jr  ra

    .size   main, .-main

    .ident  "GCC: (SiFive GCC 10.1.0-2020.08.2) 10.1.0"

Файл pascal.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\t"

    .text

    .align  2

    .globl  main

    .type   main, @function

main:

    addi    sp,sp,-16

    sw  ra,12(sp)

    sw  s0,8(sp)

    sw  s1,4(sp)

    sw  s2,0(sp)

    li  a1,4

    li  a0,0

    call    calloc

    mv  a1,a0

    li  a0,-1

    call    pascal

    li  a0,10

    call    putchar

    li  a1,4

    li  a0,1

    call    calloc

    mv  s0,a0

    mv  a1,a0

    li  a0,0

    call    pascal

    lw  a1,0(s0)

    lui s1,%hi(.LC0)

    addi    a0,s1,%lo(.LC0)

    call    printf

    li  a0,10

    call    putchar

    li  a1,4

    li  a0,2

    call    calloc

    mv  s0,a0

    mv  a1,a0

    li  a0,1

    call    pascal

    lw  a1,0(s0)

    addi    a0,s1,%lo(.LC0)

    call    printf

    lw  a1,4(s0)

    addi    a0,s1,%lo(.LC0)

    call    printf

    li  a0,10

    call    putchar

    li  a1,4

    li  a0,11

    call    calloc

    mv  s1,a0

    mv  a1,a0

    li  a0,10

    call    pascal

    mv  s0,s1

    addi    s1,s1,44

    lui s2,%hi(.LC0)

.L2:

    lw  a1,0(s0)

    addi    a0,s2,%lo(.LC0)

    call    printf

    addi    s0,s0,4

    bne s1,s0,.L2

    li  a0,0

    lw  ra,12(sp)

    lw  s0,8(sp)

    lw  s1,4(sp)

    lw  s2,0(sp)

    addi    sp,sp,16

    jr  ra

    .size   main, .-main

    .ident  "GCC: (SiFive GCC 10.1.0-2020.08.2) 10.1.0"

1. **Ассемблирование**

Ассемблирование выполняется следующими командами:

>riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -c main.s -o main.o

>riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -c pascal.s -o pascal.o

Файлы main.o и pascal.o содержат в себе машинный код. Для того, чтобы его анализировать вызовем заголовки секций, таблицу символов и таблицу перемещений для main.o.

Используются соответствующие команды:

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

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

riscv64-unknown-elf-objdump -r pascal.o main.o

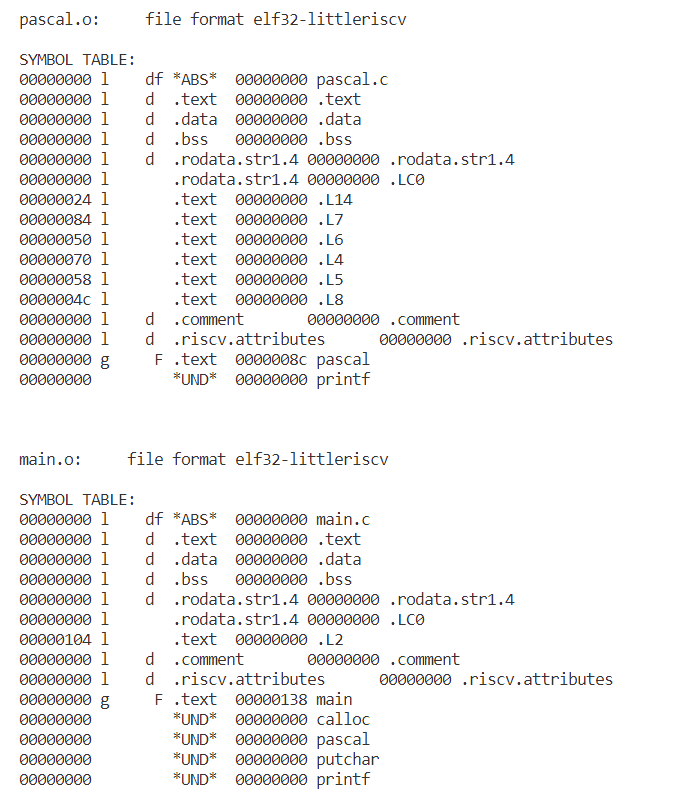


Рис.2 – таблицы символов файлов pascal.o и main.o соответственно

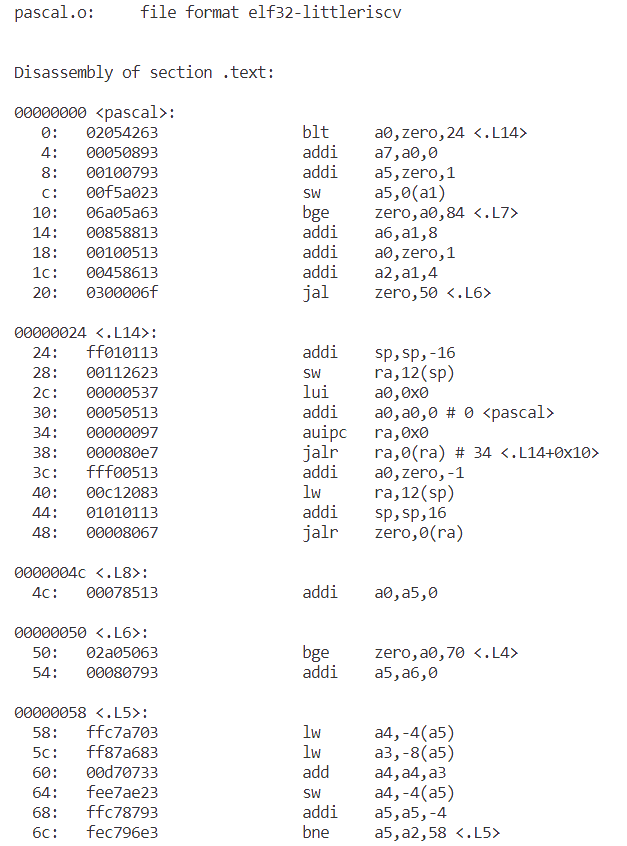


Рис 3.1 – содержимое секции .text файла pascal.o

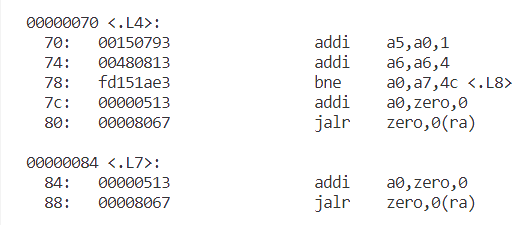


Рис. 3.2 – содержимое секции .text файла pascal.o

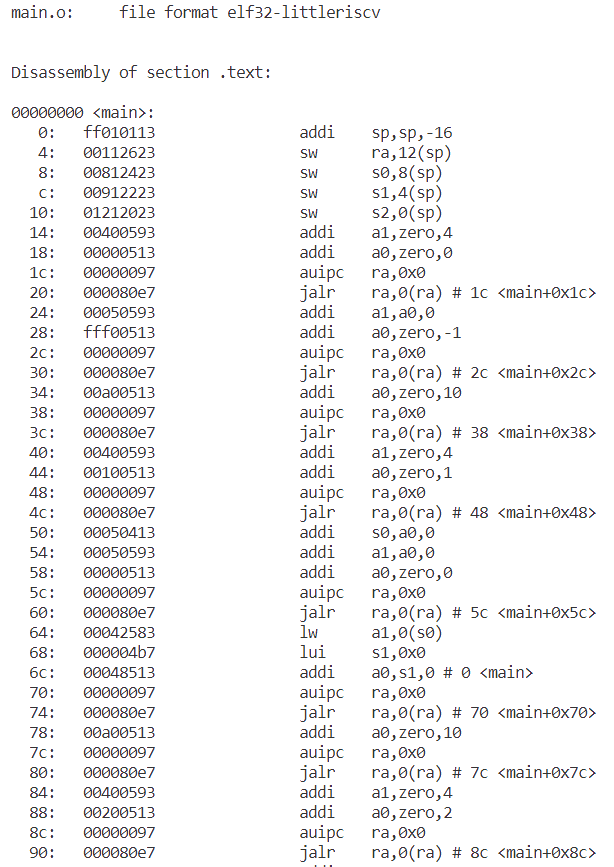


Рис 4.1 – содержимое секции .text файла main.o

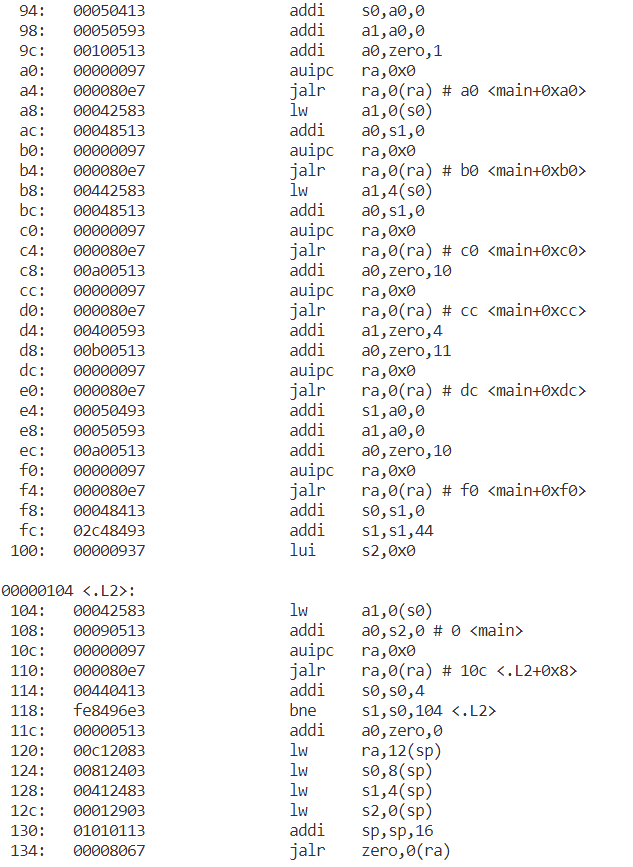


Рис 4.2 – содержимое секции .text файла main.o

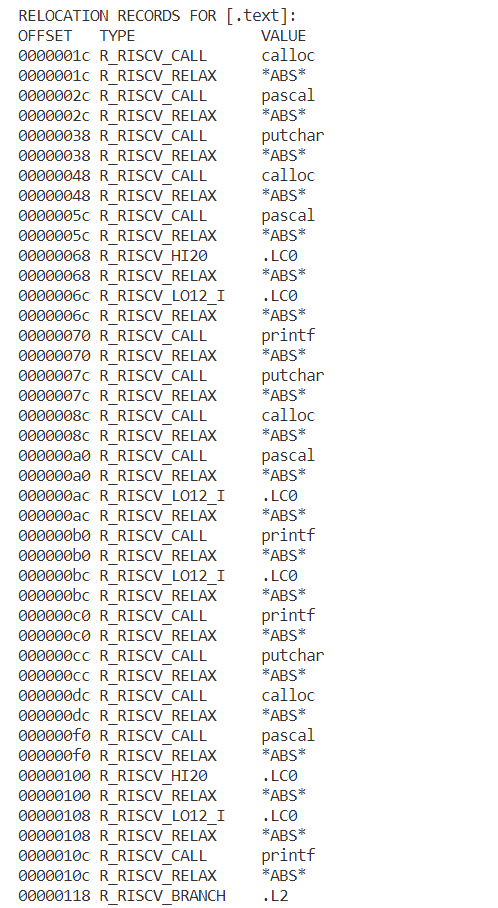


Рис.5 – таблица перемещений для файла main.o

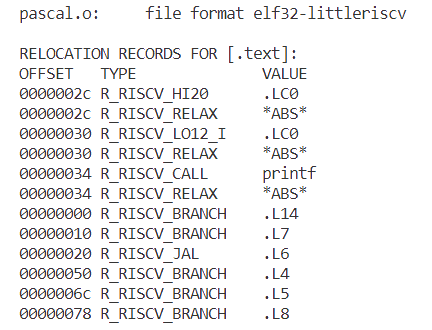


Рис. 6 – таблица перемещений для файла pascal.o

При анализе секции .text файла main.o было замечено, что вызов функции pascal отсутствует, псевдоинструкция call заменена парой инструкций auipc и jarl. Подобное мы можем заметить в ячейках 30 и 34, где как раз и был вызов подпрограммы pascal. Мы можем заметить, что результатом исполнения этой пары инструкций будет переход на адрес 2с, что приводит к зацикливанию программы. Такое поведение ассемблера можно объяснить тем, что он просто не имел возможности определить целевой адрес перехода. Для получения исполняемого кода, следовательно проблема должна быть исправлена компоновщиком, для этого используется таблица перемещений. Для наглядности можно совместить дизассемблирование и вывод таблицы перемещений.

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

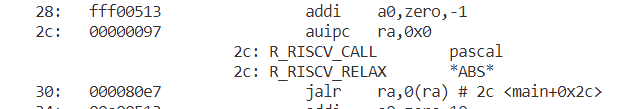


Рис. 7 – часть совмещенного кода секции .text и таблицы перемещений файла main.o

Первая запись таблицы перемещений говорит о том, что надо исправить пару инструкций (тип перемещений R\_RISCV\_CALL) по адресу 2с так, чтобы результат соответствовал вызову подпрограммы pascal. Вторая запись сообщает компоновщику о том, что пара инструкций, обеспечивающая вызов подпрограммы может быть оптимизирована.

1. **Компоновка.**

Компоновка осуществляется следующей командой:

>riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 main.o pascal.o -o main.out

Выведем заголовки секций и таблицу переходов получившегося файла main.out.

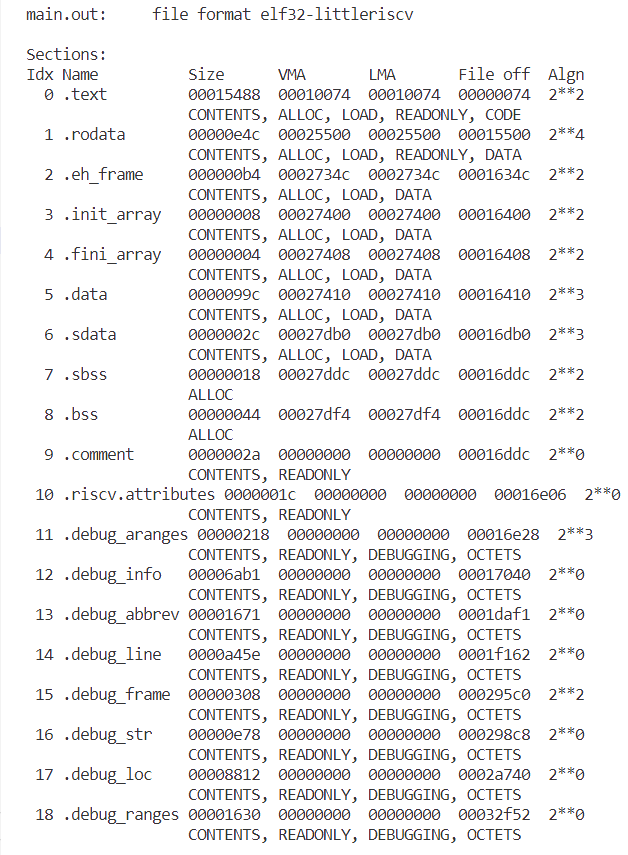


Рис 7 – заголовки секций файла main.out

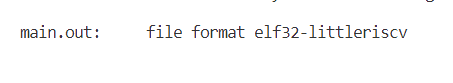


Рис. 8 – таблица перемещений файла main.out

По полученным результатам мы можем заметить, что компоновщик корректно обработал входные объектные файлы, он провел необходимые замены, то есть подключил неопределенные в объектных файлах функции, также он провел необходимые релокации, это мы видим из таблицы перемещений, и оптимизации.

**Создание статической библиотеки.**

Создадим статическую библиотеку с функцией pascal.

>riscv64-unknown-elf-ar -rsc lib.a pascal.o

Рассмотрим список символов полученной библиотеки

>riscv64-unknown-elf-nm lib.a

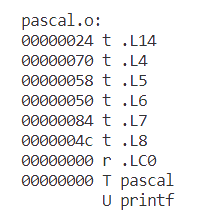


Рис. 9 – таблица символов библиотеки

Как мы можем видеть по символу T функция pascal определена в этой библиотеке, в отличие от printf.

**Создание make файла.**

В процессе выполнения работы мы можем заметить как много команд нужно вводить, чтобы перекомпилировать проект при внесении в него изменений. Для этого были созданы make-файлы, чтобы компилировать проект одной командой.

# Переменные, чтобы не переписывать одно и то же много раз.

COMPILER=riscv64-unknown-elf-gcc

LIB=riscv64-unknown-elf-ar

ARCH=-march=rv32i -mabi=ilp32

#Команды записываются в формате [цель]: [зависимость]

# \t [команда]

# all – цель по умолчанию, перед исполнением команды цели, утилита проходит по зависимостям и выполняет их команды.

all: pascal-risc

pascal-risc: main.o lib.a

    $(COMPILER) $(ARCH) main.o lib.a -o pascal-risc

main.o: main.c

    $(COMPILER) $(ARCH) -c main.c -o main.o

lib.a: pascal.o

    $(LIB) -rsc lib.a pascal.o

pascal.o: pascal.c

    $(COMPILER) $(ARCH) -c pascal.c -o pascal.o

При вызове команды «make -f MakeFile» строки из make-файла транслируются в командную строку, где исполняются. Это освобождает нас от необходимости каждый раз компилировать программу вручную.

**Вывод.**

В ходе работы была рассмотрена пошаговая компиляция программы на  
языке C, отдельная компиляция библиотеки и проанализированы все шаги  
компиляции. Дополнительно, были рассмотрены Make-файлы для  
компиляции кода библиотеки и основной программы.