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

Вариант: 13

Выполнил студент гр. 3530901/90002 _____ Д. Е. Бакин
(подпись)

Принял старший преподаватель _____ Д.С. Степанов
(подпись)

“____” _____ 2021 г.

Санкт-Петербург
2021

Постановка задачи

1. На языке C разработать функцию, реализующую вычисление полиному в точке при помощи схемы Горнера. 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 исполнимом 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.

Программа на языке С:

Схема Горнера, выглядит следующим образом. Задан многочлен:

$$P(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_n x^n, \quad a_i \in \mathbb{R}.$$

Требуется вычислить значение данного многочлена при фиксированном значении $x = x_0$. Представим многочлен $P(x)$ в следующем виде:

$$P(x) = a_0 + x(a_1 + x(a_2 + \cdots x(a_{n-1} + a_n x)\ldots)).$$

Коэффициенты a_0, a_1… a_n задаются в виде массива, где первый элемент — это старший коэффициент a_n, где n – степень полинома.

Напишем данный алгоритм на языке С, поместим функцию вычисления полинома в точке в отдельный файл и оформим заголовочный файл и файл с функцией.

```c
#include <stdio.h>
#include "Horner.h"

int main() {
    double array[] = {55, 128.5, -12, 9};
    double x = 2.1;
    int n = sizeof(array)/8;
    double f = horner(array, n, x);
    int i;
    for(i = 0; i < n; i++)
        printf( _Format: "[%.2f] ", array[i]);
    printf( _Format: "\nx=%f\nf(x)=%f",x,f);
    return 0;
}
```

Рис. 1 Файл тестовой программы main.c.

```c
#include "Horner.h"

double horner(double arr[], int n, double x){
    int i;
    if (n < 2) {
        if (n == 1) return arr[0];
        return 0;
    }
    double fx = arr[0]*x + arr[1];
    for(i=2; i<n; i++) {
        fx *= x;
        fx += arr[i];
    }
    return fx;
}
```

Рис. 2 Файл с функцией вычисление полинома в точке Horner.c.

```c
#ifndef LABA4_HORNER_H
#define LABA4_HORNER_H

double horner(double arr[], int n, double x);

#endif //LABA4_HORNER_H
```

Рис. 3 Заголовочный файл Horner.h.

Сделаем компиляцию программы с помощью MinGW-w64 и посмотрим на результат работы программы.

```
C:\laba4\cmake-build-debug\laba4.exe
[55.00] [128.50] [-12.00] [9.00]
x=2.100000
f(x)=1059.840000
Process finished with exit code 0
```

Рис. 4 Результат работы программы.

Программа выдала число 1059.84, посчитаем значение этого полинома в точке 2.1 и сравним результаты. Значния сошлись, значит программа работает корректно.

Input:

$55\,x^3 + 128.5\,x^2 - 12\,x + 9$ where $x = 2.1$

Result:

1059.84

Сборка программы “по шагам”

Препроцессирование:

Используя пакет разработки “SiFive GNU Embedded Toolchain” выполним препроцессирование файлов, используя следующие команды:

```
riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -O1 -E main.c -o main.i
riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -O1 -E Horner.c -o Horner.i
```

Получили два файла: main.i и Horner.i. В связи с тем, что в файле тестовой программы мы использовали стандартную библиотеку языка С “stdio.h” для вывода на консоль значений массива, результирующий файл препроцессирования получился очень большим.

Листинг 1. Файл main.i (фрагмент):

```
# 1 "main.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "main.c"
-----------------------------------
# 2 "main.c" 2
# 1 "Horner.h" 1
# 4 "Horner.h"
double horner(double arr[], int n, double x);
# 3 "main.c" 2

int main() {
   double array[] = {55, 128.5, -12, 9};
   double x = 2.1;
   int n = sizeof(array)/8;
   double f = horner(array, n, x);
   int i;
   for(i = 0; i < n; i++)
      printf("[%.2f] ", array[i]);
   printf("\nx=%f\nf(x)=%f",x,f);
   return 0;
}
```

Листинг 2. Файл Horner.i:

```
# 1 "Horner.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "Horner.c"
# 1 "Horner.h" 1
double horner(double arr[], int n, double x);
# 2 "Horner.c" 2
double horner(double arr[], int n, double x){
    int i;
    if (n < 2) {
        if (n == 1) return arr[0];
        return 0;
    }
    double fx = arr[0]*x + arr[1];
    for(i=2; i<n; i++) {
        fx *= x;
        fx += arr[i];
    }
    return fx;
}
```

Появившиеся нестандартные директивы, начинающиеся с символа “#”, используются для передачи информации об исходном тексте из препроцессора в компилятор. Например, последняя директива «# 1 “main.c”» информирует компилятор о том, что следующая строка является результатом обработки строки 1 исходного файла “main.c”. Также мы видим, что в данных файлах содержится информация из заголовочного файла.

Компиляция:

Компиляция осуществляется следующими командами:

```
riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -O1 –S main.i -o main.s

riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -O1 -S Horner.i -o Horner.s
```

Получаем файлы на языке ассемблера:

Листинг 3. Файл main.s:

```
        .file   "main.c"
        .option nopic
        .attribute arch, "rv64i2p0_a2p0_c2p0"
        .attribute unaligned_access, 0
        .attribute stack_align, 16
        .text
        .section        .rodata.str1.8,"aMS",@progbits,1
        .align  3
.LC2:
        .string "[%.2f] "
        .align  3
.LC3:
        .string "\nx=%f\nf(x)=%f"
        .text
        .align  1
        .globl  main
        .type   main, @function
main:
        addi    sp,sp,-80
        sd      ra,72(sp)
        sd      s0,64(sp)
        sd      s1,56(sp)
        sd      s2,48(sp)
        sd      s3,40(sp)
        lui     a5,%hi(.LANCHOR0)
        addi    a5,a5,%lo(.LANCHOR0)
        ld      a2,0(a5)
        ld      a3,8(a5)
        ld      a4,16(a5)
        ld      a5,24(a5)
        sd      a2,0(sp)
        sd      a3,8(sp)
        sd      a4,16(sp)
        sd      a5,24(sp)
        lui     a5,%hi(.LC1)
        ld      a2,%lo(.LC1)(a5)
        li      a1,4
        mv      a0,sp
        call    horner //вызов подпрограммы
        mv      s3,a0
        mv      s0,sp
        addi    s2,sp,32
```

```
        lui     s1,%hi(.LC2)
.L2:
        ld      a1,0(s0)
        addi    a0,s1,%lo(.LC2)
        call    printf //печать массива
        addi    s0,s0,8
        bne     s0,s2,.L2//цикл печати массива
        mv      a2,s3
        lui     a5,%hi(.LC1)
        ld      a1,%lo(.LC1)(a5)
        lui     a0,%hi(.LC3)
        addi    a0,a0,%lo(.LC3)
        call    printf//вызов печати в консоль нашего значения в точке x
        li      a0,0
        ld      ra,72(sp)
        ld      s0,64(sp)
        ld      s1,56(sp)
        ld      s2,48(sp)
        ld      s3,40(sp)
        addi    sp,sp,80
        jr      ra
        .size   main, .-main
        .section        .srodata.cst8,"aM",@progbits,8
        .align  3
.LC1:
        .word   -858993459
        .word   1073794252
        .section        .rodata
        .align  3
        .set    .LANCHOR0,. + 0
.LC0:
        .word   0
        .word   1078689792
        .word   0
        .word   1080037376
        .word   0
        .word   -1071120384
        .word   0
        .word   1075970048
        .ident  "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"
```

Листинг 4. Файл Horner.s:

```
        .file   "Horner.c"
        .option nopic
        .attribute arch, "rv64i2p0_a2p0_c2p0"
        .attribute unaligned_access, 0
        .attribute stack_align, 16
        .text
        .globl  __muldf3
        .globl  __adddf3
        .align  1
        .globl  horner
```

```
        .type   horner, @function
horner:
        addi    sp,sp,-48
        sd      ra,40(sp)
        sd      s0,32(sp)
        sd      s1,24(sp)
        sd      s2,16(sp)
        sd      s3,8(sp)
        mv      s3,a0
        mv      s1,a1
        li      a5,1
        ble     a1,a5,.L8//проверка на адекватность входных данных
        mv      s2,a2
        ld      a1,0(a0)
        mv      a0,a2
        call    __muldf3//умножение даблов
        ld      a1,8(s3)
        call    __adddf3//сложение даблов
        mv      a1,a0
        li      a5,2
        ble     s1,a5,.L1
        addi    s0,s3,16
        addiw   s1,s1,-3
        slli    a5,s1,32
        srli    s1,a5,29
        addi    s3,s3,24
        add     s1,s1,s3
.L4:
        mv      a0,s2
        call    __muldf3
        ld      a1,0(s0)
        call    __adddf3
        mv      a1,a0
        addi    s0,s0,8
        bne     s0,s1,.L4//цикл
.L1:
        mv      a0,a1
        ld      ra,40(sp)
        ld      s0,32(sp)
        ld      s1,24(sp)
        ld      s2,16(sp)
        ld      s3,8(sp)
        addi    sp,sp,48
        jr      ra//return
.L8:
        mv      a1,zero
        bne     s1,a5,.L1
        ld      a1,0(a0)
        j       .L1
        .size   horner, .-horner
        .ident  "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"
```

Ассемблирование:

Ассемблирование осуществляется следующими командами:

```
riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -v -c main.s -o main.o
riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -v -c Horner.s -o Horner.o
```

На выходе мы получаем два бинарных файла “main.o” и “ Horner.o”. Для их прочтения используем программу из пакета разработки.

Листинг 5. Заголовки секций файла main.o

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

```
Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .text         00000082  0000000000000000  0000000000000000  00000040  2**1
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  0000000000000000  0000000000000000  000000c2  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  0000000000000000  0000000000000000  000000c2  2**0
                  ALLOC
  3 .rodata.str1.8 00000016  0000000000000000  0000000000000000  000000c8  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .srodata.cst8 00000008  0000000000000000  0000000000000000  000000e0  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  5 .rodata       00000020  0000000000000000  0000000000000000  000000e8  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  6 .comment      00000031  0000000000000000  0000000000000000  00000108  2**0
                  CONTENTS, READONLY
  7 .riscv.attributes 00000026  0000000000000000  0000000000000000  00000139  2**0
                  CONTENTS, READONLY
```

Листинг 6. Заголовки секций файла Horner.o

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

```
Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .text         00000082  0000000000000000  0000000000000000  00000040  2**1
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  0000000000000000  0000000000000000  000000c2  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  0000000000000000  0000000000000000  000000c2  2**0
                  ALLOC
  3 .comment      00000031  0000000000000000  0000000000000000  000000c2  2**0
                  CONTENTS, READONLY
  4 .riscv.attributes 00000026  0000000000000000  0000000000000000  000000f3  2**0
                  CONTENTS, READONLY
```

Секции:

.text - скомпилированный машинный код;

.data - секция инициализированных данных;

.rodata - для неизменяемых данных;

.bss - секция данных, инициализированных нулями;

.comment — информация о версии компилятора;

Произведем декодирование кода, чтобы рассмотреть секцию .text подробнее, с помощью команды:

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

Опция “-d” инициирует процесс дизассемблирования, опция “-M no-aliases” требует использовать в выводе только инструкции системы команд.

Листинг 7. Дизассемблированный файл main.o

```
Disassembly of section .text:
0000000000000000 <main>:
   0:   715d                c.addi16sp      sp,-80
   2:   e486                c.sdsp  ra,72(sp)
   4:   e0a2                c.sdsp  s0,64(sp)
   6:   fc26                c.sdsp  s1,56(sp)
   8:   f84a                c.sdsp  s2,48(sp)
   a:   f44e                c.sdsp  s3,40(sp)
   c:   000007b7            lui     a5,0x0
  10:   00078793            addi    a5,a5,0 # 0 <main>
  14:   6390                c.ld    a2,0(a5)
  16:   6794                c.ld    a3,8(a5)
  18:   6b98                c.ld    a4,16(a5)
  1a:   6f9c                c.ld    a5,24(a5)
  1c:   e032                c.sdsp  a2,0(sp)
  1e:   e436                c.sdsp  a3,8(sp)
  20:   e83a                c.sdsp  a4,16(sp)
  22:   ec3e                c.sdsp  a5,24(sp)
  24:   000007b7            lui     a5,0x0
  28:   0007b603            ld      a2,0(a5) # 0 <main>
  2c:   4591                c.li    a1,4
  2e:   850a                c.mv    a0,sp
  30:   00000097            auipc   ra,0x0
  34:   000080e7            jalr    ra,0(ra) # 30 <main+0x30>
  38:   89aa                c.mv    s3,a0
  3a:   840a                c.mv    s0,sp
  3c:   02010913            addi    s2,sp,32
  40:   000004b7            lui     s1,0x0

0000000000000044 <.L2>:
  44:   600c                c.ld    a1,0(s0)
  46:   00048513            addi    a0,s1,0 # 0 <main>
  4a:   00000097            auipc   ra,0x0
  4e:   000080e7            jalr    ra,0(ra) # 4a <.L2+0x6>
  52:   0421                c.addi  s0,8
  54:   ff2418e3            bne     s0,s2,44 <.L2>
  58:   864e                c.mv    a2,s3
  5a:   000007b7            lui     a5,0x0
  5e:   0007b583            ld      a1,0(a5) # 0 <main>
  62:   00000537            lui     a0,0x0
  66:   00050513            addi    a0,a0,0 # 0 <main>
  6a:   00000097            auipc   ra,0x0
  6e:   000080e7            jalr    ra,0(ra) # 6a <.L2+0x26>
  72:   4501                c.li    a0,0
  74:   60a6                c.ldsp  ra,72(sp)
  76:   6406                c.ldsp  s0,64(sp)
  78:   74e2                c.ldsp  s1,56(sp)
  7a:   7942                c.ldsp  s2,48(sp)
  7c:   79a2                c.ldsp  s3,40(sp)
  7e:   6161                c.addi16sp      sp,80
  80:   8082                c.jr    ra
```

Рассмотрим таблицу символов и таблицу перемещений с помощью команд:

```
riscv64-unknown-elf-objdump -t main.o Horner.o
riscv64-unknown-elf-objdump -r main.o Horner.o
```

Листинг 8. Таблица символов

```
SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 main.c
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000000 l    d  .data  0000000000000000 .data
0000000000000000 l    d  .bss   0000000000000000 .bss
0000000000000000 l    d  .rodata.str1.8 0000000000000000 .rodata.str1.8
0000000000000000 l    d  .srodata.cst8  0000000000000000 .srodata.cst8
0000000000000000 l    d  .rodata        0000000000000000 .rodata
0000000000000000 l      .rodata        0000000000000000 .LANCHOR0
0000000000000000 l      .srodata.cst8  0000000000000000 .LC1
0000000000000000 l      .rodata.str1.8 0000000000000000 .LC2
0000000000000008 l      .rodata.str1.8 0000000000000000 .LC3
0000000000000044 l      .text  0000000000000000 .L2
0000000000000000 l    d  .comment       0000000000000000 .comment
0000000000000000 l    d  .riscv.attributes      0000000000000000 .riscv.attributes
0000000000000000 g     F .text  0000000000000082 main
0000000000000000         *UND*  0000000000000000 horner
0000000000000000         *UND*  0000000000000000 printf

Horner.o:     file format elf64-littleriscv

SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 Horner.c
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000000 l    d  .data  0000000000000000 .data
0000000000000000 l    d  .bss   0000000000000000 .bss
0000000000000076 l      .text  0000000000000000 .L8
0000000000000066 l      .text  0000000000000000 .L1
000000000000004a l      .text  0000000000000000 .L4
0000000000000000 l    d  .comment       0000000000000000 .comment
0000000000000000 l    d  .riscv.attributes      0000000000000000 .riscv.attributes
0000000000000000         *UND*  0000000000000000 __muldf3
0000000000000000         *UND*  0000000000000000 __adddf3
0000000000000000 g     F .text  0000000000000082 horner
```

Видим несколько записей с иентификатором *UND*, это относится к сложению, умножению типа double, а также к вызову printf и horner. Это связанно с тем, что символ определен где-то еще.

Листинг 9. Таблица перемещений

```
main.o:     file format elf64-littleriscv

RELOCATION RECORDS FOR [.text]:
OFFSET           TYPE              VALUE
000000000000000c R_RISCV_HI20      .LANCHOR0
000000000000000c R_RISCV_RELAX     *ABS*
0000000000000010 R_RISCV_LO12_I    .LANCHOR0
0000000000000010 R_RISCV_RELAX     *ABS*
0000000000000024 R_RISCV_HI20      .LC1
0000000000000024 R_RISCV_RELAX     *ABS*
0000000000000028 R_RISCV_LO12_I    .LC1
0000000000000028 R_RISCV_RELAX     *ABS*
0000000000000030 R_RISCV_CALL      horner
0000000000000030 R_RISCV_RELAX     *ABS*
0000000000000040 R_RISCV_HI20      .LC2
0000000000000040 R_RISCV_RELAX     *ABS*
0000000000000046 R_RISCV_LO12_I    .LC2
0000000000000046 R_RISCV_RELAX     *ABS*
000000000000004a R_RISCV_CALL      printf
000000000000004a R_RISCV_RELAX     *ABS*
000000000000005a R_RISCV_HI20      .LC1
000000000000005a R_RISCV_RELAX     *ABS*
000000000000005e R_RISCV_LO12_I    .LC1
000000000000005e R_RISCV_RELAX     *ABS*
0000000000000062 R_RISCV_HI20      .LC3
0000000000000062 R_RISCV_RELAX     *ABS*
0000000000000066 R_RISCV_LO12_I    .LC3
0000000000000066 R_RISCV_RELAX     *ABS*
000000000000006a R_RISCV_CALL      printf
000000000000006a R_RISCV_RELAX     *ABS*
0000000000000054 R_RISCV_BRANCH    .L2


Horner.o:     file format elf64-littleriscv

RELOCATION RECORDS FOR [.text]:
OFFSET           TYPE              VALUE
000000000000001c R_RISCV_CALL      __muldf3
000000000000001c R_RISCV_RELAX     *ABS*
0000000000000028 R_RISCV_CALL      __adddf3
0000000000000028 R_RISCV_RELAX     *ABS*
000000000000004c R_RISCV_CALL      __muldf3
000000000000004c R_RISCV_RELAX     *ABS*
0000000000000056 R_RISCV_CALL      __adddf3
0000000000000056 R_RISCV_RELAX     *ABS*
0000000000000012 R_RISCV_BRANCH    .L8
0000000000000034 R_RISCV_BRANCH    .L1
0000000000000062 R_RISCV_BRANCH    .L4
000000000000007a R_RISCV_BRANCH    .L1
0000000000000080 R_RISCV_RVC_JUMP  .L1
```

Здесь содержится информация обо всех «неоконченных» инструкциях.

Записи типа “R_RISCV_RELAX” заносятся в таблицу перемещений в дополнение к записям типа “R_RISCV_CALL” и сообщают компоновщику, что пара инструкций, обеспечивающих вызов подпрограммы, может быть оптимизирована.

Компиляция

Выполним компоновку следующей командой:

```
riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -v main.o Horner.o
```

Листинг 10. Фрагмент исполняемого файла

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

```
0000000000010156 <main>:
   10156:       715d                    c.addi16sp      sp,-80
   10158:       e486                    c.sdsp  ra,72(sp)
   1015a:       e0a2            c.sdsp  s0,64(sp)
   1015c:       fc26            c.sdsp  s1,56(sp)
   1015e:       f84a            c.sdsp  s2,48(sp)
   10160:       f44e            c.sdsp  s3,40(sp)
   10162:       67f5            c.lui   a5,0x1d
   10164:       eb878793                addi    a5,a5,-328 # 1ceb8 <__moddi3+0x58>
   10168:       6390                    c.ld    a2,0(a5)
   1016a:       6794                    c.ld    a3,8(a5)
   1016c:       6b98                    c.ld    a4,16(a5)
   1016e:       6f9c            c.ld    a5,24(a5)
   10170:       e032                    c.sdsp  a2,0(sp)
   10172:       e436                    c.sdsp  a3,8(sp)
   10174:       e83a            c.sdsp  a4,16(sp)
   10176:       ec3e            c.sdsp  a5,24(sp)
   10178:       7101b603                ld      a2,1808(gp) # 1f120 <__SDATA_BEGIN__>
   1017c:       4591                    c.li    a1,4
   1017e:       850a                    c.mv    a0,sp
   10180:       03e000ef                jal     ra,101be <horner>
   10184:       89aa            c.mv    s3,a0
   10186:       840a                    c.mv    s0,sp
   10188:       02010913                addi    s2,sp,32
   1018c:       64f5            c.lui   s1,0x1d
   1018e:       600c                    c.ld    a1,0(s0)
   10190:       ea048513                addi    a0,s1,-352 # 1cea0 <__moddi3+0x40>
   10194:       79a000ef                jal     ra,1092e <printf>
   10198:       0421                    c.addi  s0,8
   1019a:       ff241ae3                bne     s0,s2,1018e <main+0x38>
   1019e:       864e                    c.mv    a2,s3
   101a0:       7101b583                ld      a1,1808(gp) # 1f120 <__SDATA_BEGIN__>
   101a4:       6575                    c.lui   a0,0x1d
   101a6:       ea850513                addi    a0,a0,-344 # 1cea8 <__moddi3+0x48>
   101aa:       784000ef                jal     ra,1092e <printf>
   101ae:       4501                    c.li    a0,0
```

```
   101b0:       60a6                    c.ldsp  ra,72(sp)
   101b2:       6406                    c.ldsp  s0,64(sp)
   101b4:       74e2                    c.ldsp  s1,56(sp)
   101b6:       7942                    c.ldsp  s2,48(sp)
   101b8:       79a2                    c.ldsp  s3,40(sp)
   101ba:       6161                    c.addi16sp      sp,80
   101bc:       8082                    c.jr    ra
00000000000101be <horner>:
   101be:       7179                    c.addi16sp      sp,-48
   101c0:       f406            c.sdsp  ra,40(sp)
   101c2:       f022            c.sdsp  s0,32(sp)
   101c4:       ec26            c.sdsp  s1,24(sp)
   101c6:       e84a            c.sdsp  s2,16(sp)
   101c8:       e44e            c.sdsp  s3,8(sp)
   101ca:       89aa            c.mv    s3,a0
   101cc:       84ae            c.mv    s1,a1
   101ce:       4785                    c.li    a5,1
   101d0:       04b7da63                bge     a5,a1,10224 <horner+0x66>
   101d4:       8932                    c.mv    s2,a2
   101d6:       610c                    c.ld    a1,0(a0)
   101d8:       8532                    c.mv    a0,a2
   101da:       37c000ef                jal     ra,10556 <__muldf3>
   101de:       0089b583                ld      a1,8(s3)
   101e2:       04e000ef                jal     ra,10230 <__adddf3>
   101e6:       85aa            c.mv    a1,a0
   101e8:       4789                    c.li    a5,2
   101ea:       0297d563                bge     a5,s1,10214 <horner+0x56>
   101ee:       01098413                addi    s0,s3,16
   101f2:       34f5            c.addiw         s1,-3
   101f4:       02049793                slli    a5,s1,0x20
   101f8:       01d7d493                srli    s1,a5,0x1d
   101fc:       09e1                    c.addi  s3,24
   101fe:       94ce            c.add   s1,s3
   10200:       854a                    c.mv    a0,s2
   10202:       354000ef                jal     ra,10556 <__muldf3>
   10206:       600c                    c.ld    a1,0(s0)
   10208:       028000ef                jal     ra,10230 <__adddf3>
   1020c:       85aa            c.mv    a1,a0
   1020e:       0421                    c.addi  s0,8
   10210:       fe9418e3                bne     s0,s1,10200 <horner+0x42>
   10214:       852e                    c.mv    a0,a1
   10216:       70a2                    c.ldsp  ra,40(sp)
   10218:       7402                    c.ldsp  s0,32(sp)
   1021a:       64e2                    c.ldsp  s1,24(sp)
   1021c:       6942                    c.ldsp  s2,16(sp)
   1021e:       69a2                    c.ldsp  s3,8(sp)
   10220:       6145                    c.addi16sp      sp,48
   10222:       8082                    c.jr    ra
   10224:       00000593                addi    a1,zero,0
   10228:       fef496e3                bne     s1,a5,10214 <horner+0x56>
   1022c:       610c                    c.ld    a1,0(a0)
   1022e:       b7dd                    c.j     10214 <horner+0x56>
```

Создание статической библиотеки

Выделим функцию Horner в отдельную статическую библиотеку. Для этого надо получить объектный файл insetion.o и собрать библиотеку.

```
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -O1 -c Horner.c -o Horner.o
riscv64-unknown-elf-ar -rsc libr.a Horner.o
```

Рассмотрим список символов библиотеки:

Листинг 11. Список символов libr.a

```
riscv64-unknown-elf-nm libr.a
```

```
Horner.o:
0000000000000066 t .L1
000000000000004a t .L4
0000000000000076 t .L8
                 U __adddf3
                 U __muldf3
0000000000000000 T horner
```

Кодом “T” обозначаются символы, определенные в объектном файле.

Теперь, имея собранную библиотеку, создадим исполняемый файл тестовой программы *‘main.c”* с помощью следующей команды:

```
riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -O1 main.c libr.a -o main.out
```

Убедимся, что в состав программы вошло содержание объектного файла Horner.o, при помощи таблицы символов исполняемого файла

Листинг 12. Фрагмент списка символов main.out

```
riscv64-unknown-elf-objdump -t main.out >main.ds
```

```
main.out:     file format elf64-littleriscv

SYMBOL TABLE:
--------------------------
0000000000000000 l    df *ABS*    0000000000000000 main.c
0000000000000000 l    df *ABS*    0000000000000000 Horner.c
0000000000000000 l    df *ABS*    0000000000000000 adddf3.c
0000000000000000 l    df *ABS*    0000000000000000 muldf3.c
```

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

Листинг 13. Makefile для создания статической библиотеки “makeLibrary”

```
CC=riscv64-unknown-elf-gcc
AR=riscv64-unknown-elf-ar
CFLAGS=-march=rv64iac -mabi=lp64

all: libr

libr: Horner.o
	$(AR) -rsc libr.a Horner.o
	del -f *.o
Horner.o: Horner.c
	$(CC) $(CFLAGS) -c Horner.c -o Horner.o
```

Листинг 14. Makefile для сборки исполняемого файла “makeApp”

```
TARGET=main.out
CC=riscv64-unknown-elf-gcc
CFLAGS=-march=rv64iac -mabi=lp64

all:
	make -f makeLibrary
	$(CC) $(CFLAGS) main.c libr.a -o $(TARGET)
	del -f *.0 *.a
```

Теперь с помощью GNU make выполним сначала makeLibrary, а затем makeApp, для создания библиотеки.

Посмотрим таблицу символов полученного с помощью makefile исполняемого файла:

Листинг 15. Фрагмент списка символов main.out (makefile).

```
riscv64-unknown-elf-objdump -t main.out >main.ds
```

```
main.out:     file format elf64-littleriscv

SYMBOL TABLE:
--------------------------
0000000000000000 l    df *ABS*      0000000000000000 main.c
0000000000000000 l    df *ABS*      0000000000000000 Horner.c
0000000000000000 l    df *ABS*      0000000000000000 adddf3.c
0000000000000000 l    df *ABS*      0000000000000000 muldf3.c
```

Файлы идентичны.

Вывод

В ходе лабораторной работы была напсана программа вычисления заданного полинома в точке, с помощью схемы Горнера на языке С. Была выполнена компиляци, проверка работоспособности программы. После чего была выполнена раздельная компиляция.