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

Низкоуровневое программирование

Отчет по лабораторной работе №3

Программирование RISC-V

Работу выполнил:
Аникин Д.А.
Группа:
3530901/90004
Преподаватель:
Алексюк А.О.

Санкт-Петербург
2021

Содержание

1. Формулировка задания

1. На языке C разработать функцию, реализующую определенную вариантом задания функциональность. 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 исполняеммом 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.

Формулировка варианта задания

Интегрирование табличной функции методом трапеций с «длинным» результатом. *Примечание: переполнение разрядной сетки предотвращается пользователем масштабированием параметра шага сетки.*

$$\int_a^b f(x)\,dx = \sum_{i=0}^{n-1} \frac{f(x_i) + f(x_{i+1})}{2}(x_{i+1} - x_i)$$

2. Разработка программы на языке C

Согласно заданию, были разработаны две программы на языке C. Первая программма реализует заданную вариантом функциональность и состоит из файла с исходным кодом (libtrapz.c) и заголовочным файлом (libtrapz.h), в котором объявляется функция, являющаяся частью интерфейса вызова модуля libtrapz. Вторая программа (main.c) предназначена для тестирования работоспособности первой.

Модуль libtrapz

Листинг 1: libtrapz.c

```
1 #include <stddef.h>
2
3 double trapz(double *xdata, double *ydata, size_t len)
4 {
5     double sum;
6
7     for (int i = 0; i < len - 1; i++) {
8         sum += (ydata[i + 1] + ydata[i]) * (xdata[i + 1] - xdata[i]) / 2;
```

```
9       }
10
11      return sum;
12  }
```

Листинг 2: libtrapz.h

```
1 #ifndef _LIBTRAPZ_H_
2 #define _LIBTRAPZ_H_
3
4 double trapz(double *xdata, double *ydata, size_t len);
5
6 #endif /* _LIBTRAPZ_H_ */
```

Модуль app

Листинг 3: main.c

```
1  #include <stddef.h>
2  #include <stdio.h>
3
4  #include "../../include/libtrapz/libtrapz.h"
5
6  int main(void)
7  {
8       double xarray[] = {1.1, 2.28, 4.6, 5, 10.001};
9       double yarray[] = {0.8, 0.16, 0.8, 0.16, 0.8};
10
11      size_t length = sizeof(xarray) / sizeof(*xarray);
12
13      printf("%f\n", trapz(xarray, yarray, length));
14
15      return 0;
16  }
```

3. Сборка программы "по шагам"

Препроцессирование

Первым шагом является препроцессирование файла исходного текста. На данной стадии происходит происходит работа с препроцессорными директивами. Например, препроцессор добавляет хэдеры в код (#**include**), убирает комментирования, заменяет макросы (#**define**) их значениями, выбирает нужные куски кода в соответствии с условиями #**if**, #**ifdef** и #**ifndef**.

Листинг 4: Фрагмент libtrapz.i

```
1 # 1 "/home/dmitri3агрузки//LowLevelProgramming2021/separate/src/libtrapz/libtrapz.c"
2 # 1 "<built-in>"
3 # 1 "<command-line>"
4 # 1 "/home/dmitri3агрузки//LowLevelProgramming2021/separate/src/libtrapz/libtrapz.c"
5 # 1 "/media/dmitri/scdvfbg/riscv64-unknown-elf-toolchain-10.2.0-2020.12.8-x86_64-linux-ubuntu14/lib/gcc/
      ↪ riscv64-unknown-elf/10.2.0/include/stddef.h" 1 3 4
6 # 143 "/media/dmitri/scdvfbg/riscv64-unknown-elf-toolchain-10.2.0-2020.12.8-x86_64-linux-ubuntu14/lib/gcc/
      ↪ riscv64-unknown-elf/10.2.0/include/stddef.h" 3 4Загрузки
23 # 3 "/home/dmitri3агрузки//LowLevelProgramming2021/separate/src/libtrapz/libtrapz.c"
24 double trapz(double *xdata, double *ydata, size_t len)
25 {
26     double sum;
28
28     for (int i = 0; i < len - 1; i++) {
29         sum += (ydata[i + 1] + ydata[i]) * (xdata[i + 1] - xdata[i]) / 2;
30     }
32
32     return sum;
33 }
```

Листинг 5: Фрагмент main.i

```
1 # 1 "/home/dmitri3агрузки//LowLevelProgramming2021/separate/src/app/main.c"
2 # 1 "<built-in>"
3 # 1 "<command-line>"
4 # 1 "/home/dmitri3агрузки//LowLevelProgramming2021/separate/src/app/main.c"
5 # 1 "/media/dmitri/scdvfbg/riscv64-unknown-elf-toolchain-10.2.0-2020.12.8-x86_64-linux-ubuntu14/lib/gcc/
      ↪ riscv64-unknown-elf/10.2.0/include/stddef.h" 1 3 4
6 # 143 "/media/dmitri/scdvfbg/riscv64-unknown-elf-toolchain-10.2.0-2020.12.8-x86_64-linux-ubuntu14/lib/gcc/
      ↪ riscv64-unknown-elf/10.2.0/include/stddef.h" 3 4ЗагрузкиЗагрузкиЗагрузки
1282 # 4 "/home/dmitri3агрузки//LowLevelProgramming2021/separate/src/app/../../include/libtrapz/libtrapz.h"
1283 double trapz(double *xdata, double *ydata, size_t len);
1284 # 5 "/home/dmitri3агрузки//LowLevelProgramming2021/separate/src/app/main.c" 2
1286
1286 int main(void)
1287 {
1288     double xarray[] = {1.1, 2.28, 4.6, 5, 10.001};
1289     double yarray[] = {0.8, 0.16, 0.8, 0.16, 0.8};
1291
1291     size_t length = sizeof(xarray) / sizeof(*xarray);
1293
1293     printf("%f", trapz(xarray, yarray, length));
1295
1295     return 0;
1296 }
```

В новых сгенерированных файлах можно увидеть огромное количество новых строк, в основном "хедеры" **<stddef.h>** и **<stdio.h>**. Появившиеся нестандартные директивы, начинающиеся с символа "#", используются для передачи информации об исходном тексте из препроцессора в компилятор.

Компиляция

На данном шаге компилятор выполняет свою главную задачу — компилирует, то есть преобразует полученный на прошлом шаге код без директив в ассемблерный код. Рассмотрим полученные файлы libtrapz.s и main.s

Листинг 6: libtrapz.s

```
1   .file "libtrapz.c"
2   .option nopic
3   .attribute arch, "rv32i2p0"
4   .attribute unaligned_access, 0
5   .attribute stack_align, 16
```

```
6     .text
7     .globl   __adddf3
8     .globl   __subdf3
9     .globl   __muldf3
10    .align   2
11    .globl   trapz
12    .type trapz, @function
13  trapz:
14    addi  sp,sp,-48
15    sw   ra,44(sp)
16    sw   s0,40(sp)
17    sw   s1,36(sp)
18    sw   s2,32(sp)
19    sw   s3,28(sp)
20    sw   s4,24(sp)
21    sw   s5,20(sp)
22    sw   s6,16(sp)
23    sw   s7,12(sp)
24    sw   s8,8(sp)
25    li   a5,1
26    beq  a2,a5,.L4 # __EXIT_LOOP_CONDITION__
27    mv   s1,a0
28    addi  s0,a1,8
29    slli   a2,a2,3
30    add  s7,a1,a2
31    li   s3,0
32    li   s2,0
33    lui  a5,%hi(.LC0)
34    lw   s4,%lo(.LC0)(a5)
35    lw   s5,%lo(.LC0+4)(a5)
36  .L3: # __COMPUTING_EXPRESSION__
37    lw   a2,-8(s0)
38    lw   a3,-4(s0)
39    lw   a0,0(s0)
40    lw   a1,4(s0)
41    call   __adddf3
42    mv   s8,a0
43    mv   s6,a1
44    lw   a2,0(s1)
45    lw   a3,4(s1)
46    lw   a0,8(s1)
47    lw   a1,12(s1)
48    call   __subdf3
49    mv   a2,a0
50    mv   a3,a1
51    mv   a0,s8
52    mv   a1,s6
53    call   __muldf3
```

```
54    mv   a2,s4
55    mv   a3,s5
56    call   __muldf3
57    mv   a2,a0
58    mv   a3,a1
59    mv   a0,s3
60    mv   a1,s2
61    call   __adddf3
62    mv   s3,a0
63    mv   s2,a1
64    addi   s1,s1,8
65    addi   s0,s0,8
66    bne  s0,s7,.L3
67  .L1: # __RETURN__SUM__
68    mv   a0,s3
69    mv   a1,s2
70    lw   ra,44(sp)
71    lw   s0,40(sp)
72    lw   s1,36(sp)
73    lw   s2,32(sp)
74    lw   s3,28(sp)
75    lw   s4,24(sp)
76    lw   s5,20(sp)
77    lw   s6,16(sp)
78    lw   s7,12(sp)
79    lw   s8,8(sp)
80    addi   sp,sp,48
81    jr   ra
82  .L4: # __EXIT_LOOP__
83    li   s3,0
84    li   s2,0
85    j  .L1
86    .size trapz, .-trapz
87    .section   .srodata.cst8,"aM",@progbits,8
88    .align   3
89  .LC0:
90    .word 0
91    .word 1071644672
92    .ident   "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"
```

Особый интерес представляет форма хранения чисел с плавающей запятой двойной точности: целая и дробная часть числа хранятся раздельно в формате word. Для взаимодействия с ними используются внутренние специальные функции, такие как *__adddf3*. В метке .L3 хранится код для вычисления по формуле. Метка .L1 отвечает за возвращение результата выполнения и выход из программы.

Листинг 7: main.s

```
1   .file "main.c"
2   .option nopic
3   .attribute arch, "rv32i2p0"
4   .attribute unaligned_access, 0
5   .attribute stack_align, 16
6   .text
7   .section  .rodata.str1.4,"aMS",@progbits,1
8   .align  2
9 .LC2:
10  .string "%f"
11  .text
12  .align  2
13  .globl  main
14  .type main, @function
15 main:
16  addi  sp,sp,-96
17  sw  ra,92(sp)
18  lui a5,%hi(.LANCHOR0)
19  addi  a5,a5,%lo(.LANCHOR0)
20  lw  t4,0(a5)
21  lw  t3,4(a5)
22  lw  t1,8(a5)
23  lw  a7,12(a5)
24  lw  a6,16(a5)
25  lw  a0,20(a5)
26  lw  a1,24(a5)
27  lw  a2,28(a5)
28  lw  a3,32(a5)
29  lw  a4,36(a5)
30  sw  t4,40(sp)
31  sw  t3,44(sp)
32  sw  t1,48(sp)
33  sw  a7,52(sp)
34  sw  a6,56(sp)
35  sw  a0,60(sp)
36  sw  a1,64(sp)
37  sw  a2,68(sp)
38  sw  a3,72(sp)
39  sw  a4,76(sp)
40  lw  t3,40(a5)
41  lw  t1,44(a5)
42  lw  a7,48(a5)
43  lw  a6,52(a5)
44  lw  a0,56(a5)
45  lw  a1,60(a5)
46  lw  a2,64(a5)
47  lw  a3,68(a5)
```

```
48    lw   a4,72(a5)
49    lw   a5,76(a5)
50    sw   t3,0(sp)
51    sw   t1,4(sp)
52    sw   a7,8(sp)
53    sw   a6,12(sp)
54    sw   a0,16(sp)
55    sw   a1,20(sp)
56    sw   a2,24(sp)
57    sw   a3,28(sp)
58    sw   a4,32(sp)
59    sw   a5,36(sp)
60    li   a2,5
61    mv   a1,sp
62    addi   a0,sp,40
63    call   trapz
64    mv   a2,a0
65    mv   a3,a1
66    lui  a0,%hi(.LC2)
67    addi   a0,a0,%lo(.LC2)
68    call   printf
69    li   a0,0
70    lw   ra,92(sp)
71    addi   sp,sp,96
72    jr   ra
73    .size  main, .-main
74    .section   .rodata
75    .align   3
76    .set   .LANCHOR0,. + 0
77 .LC0:
78    .word  -1717986918
79    .word  1072798105
80    .word  -1546188227
81    .word  1073888624
82    .word  1717986918
83    .word  1074947686
84    .word  0
85    .word  1075052544
86    .word  309237645
87    .word  1076101251
88 .LC1:
89    .word  -1717986918
90    .word  1072273817
91    .word  1202590843
92    .word  1069841121
93    .word  -1717986918
94    .word  1072273817
95    .word  1202590843
```

```
96    .word 1069841121
97    .word -1717986918
98    .word 1072273817
99    .ident  "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"
```

Видно, что тестовая программа вызывает функцию trapz, аргументы передаются через регистры a0, a1, a2. В метке .LC1 хранится массив Y, а в .LC0 - массив X.

Ассемблирование

Ассемблер преобразовывает ассемблерный код в машинный код, сохраняя его в объектном файле. Сформированный ассемблером объектный файл должен содержать коды инструкций, таблицу символов и таблицу перемещений. В отличие от ранее рассмотренных файлов, объектный файл не является текстовым, для изучения его содержимого используется утилиту objdump, отображающая содержимое бинарных файлов в текстовом виде.

Изучим содержимое таблиц символов объектных файлов.

Листинг 8: Таблица символов libtrapz.o

```
2
2 libtrapz.o:     file format elf32-littleriscv
4
4 SYMBOL TABLE:
5 00000000 l    df *ABS*  00000000 libtrapz.c
6 00000000 l    d  .text  00000000 .text
7 00000000 l    d  .data  00000000 .data
8 00000000 l    d  .bss 00000000 .bss
9 00000000 l    d  .srodata.cst8  00000000 .srodata.cst8
10 00000000 l       .srodata.cst8  00000000 .LC0
11 0000011c l       .text  00000000 .L4
12 00000058 l       .text  00000000 .L3
13 000000e4 l       .text  00000000 .L1
14 00000000 l    d  .comment 00000000 .comment
15 00000000 l    d  .riscv.attributes  00000000 .riscv.attributes
16 00000000         *UND*  00000000 __adddf3
17 00000000         *UND*  00000000 __subdf3
18 00000000         *UND*  00000000 __muldf3
19 00000000 g     F .text  00000128 trapz
```

Листинг 9: Таблица символов main.o

```
2
2 main.o:     file format elf32-littleriscv
4
4 SYMBOL TABLE:
5 00000000 l    df *ABS*  00000000 main.c
6 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  00000000 .rodata
00000000 l       .rodata  00000000 .LANCHOR0
00000000 l       .rodata.str1.4 00000000 .LC2
00000000 l    d  .comment 00000000 .comment
00000000 l    d  .riscv.attributes  00000000 .riscv.attributes
00000000 g     F .text  000000ec main
00000000         *UND*  00000000 trapz
00000000         *UND*  00000000 printf
```

В таблице символов имеется интересные записи: симвоы типа “*UND*” (undefined – не определен). Эти записи означают, что символ 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.

Листинг 10: Инстркци программы

```

main.o:     file format elf32-littleriscv


Disassembly of section .text:

00000000 <main>:
   0: fa010113            addi  sp,sp,-96
   4: 04112e23            sw  ra,92(sp)
   8: 000007b7            lui a5,0x0
   c: 00078793            mv  a5,a5
  10: 0007ae83            lw  t4,0(a5) # 0 <main>
  14: 0047ae03            lw  t3,4(a5)
  18: 0087a303            lw  t1,8(a5)
  1c: 00c7a883            lw  a7,12(a5)
  20: 0107a803            lw  a6,16(a5)
  24: 0147a503            lw  a0,20(a5)
  28: 0187a583            lw  a1,24(a5)
  2c: 01c7a603            lw  a2,28(a5)
  30: 0207a683            lw  a3,32(a5)
  34: 0247a703            lw  a4,36(a5)
  38: 03d12423            sw  t4,40(sp)
  3c: 03c12623            sw  t3,44(sp)
  40: 02612823            sw  t1,48(sp)
  44: 03112a23            sw  a7,52(sp)
  48: 03012c23            sw  a6,56(sp)
  4c: 02a12e23            sw  a0,60(sp)
  50: 04b12023            sw  a1,64(sp)
```

```
  54: 04c12223            sw  a2,68(sp)
  58: 04d12423            sw  a3,72(sp)
  5c: 04e12623            sw  a4,76(sp)
  60: 0287ae03            lw  t3,40(a5)
  64: 02c7a303            lw  t1,44(a5)
  68: 0307a883            lw  a7,48(a5)
  6c: 0347a803            lw  a6,52(a5)
  70: 0387a503            lw  a0,56(a5)
  74: 03c7a583            lw  a1,60(a5)
  78: 0407a603            lw  a2,64(a5)
  7c: 0447a683            lw  a3,68(a5)
  80: 0487a703            lw  a4,72(a5)
  84: 04c7a783            lw  a5,76(a5)
  88: 01c12023            sw  t3,0(sp)
  8c: 00612223            sw  t1,4(sp)
  90: 01112423            sw  a7,8(sp)
  94: 01012623            sw  a6,12(sp)
  98: 00a12823            sw  a0,16(sp)
  9c: 00b12a23            sw  a1,20(sp)
  a0: 00c12c23            sw  a2,24(sp)
  a4: 00d12e23            sw  a3,28(sp)
  a8: 02e12023            sw  a4,32(sp)
  ac: 02f12223            sw  a5,36(sp)
  b0: 00500613            li  a2,5
  b4: 00010593            mv  a1,sp
  b8: 02810513            addi  a0,sp,40
  bc: 00000097            auipc ra,0x0
  c0: 000080e7            jalr  ra # bc <main+0xbc>
  c4: 00050613            mv  a2,a0
  c8: 00058693            mv  a3,a1
  cc: 00000537            lui a0,0x0
  d0: 00050513            mv  a0,a0
  d4: 00000097            auipc ra,0x0
  d8: 000080e7            jalr  ra # d4 <main+0xd4>
  dc: 00000513            li  a0,0
  e0: 05c12083            lw  ra,92(sp)
  e4: 06010113            addi  sp,sp,96
  e8: 00008067            ret

libtrapz.o:     file format elf32-littleriscv


Disassembly of section .text:

00000000 <trapz>:
   0: fd010113            addi  sp,sp,-48
   4: 02112623            sw  ra,44(sp)
   8: 02812423            sw  s0,40(sp)
```

```
77     c: 02912223            sw  s1,36(sp)
78    10: 03212023            sw  s2,32(sp)
79    14: 01312e23            sw  s3,28(sp)
80    18: 01412c23            sw  s4,24(sp)
81    1c: 01512a23            sw  s5,20(sp)
82    20: 01612823            sw  s6,16(sp)
83    24: 01712623            sw  s7,12(sp)
84    28: 01812423            sw  s8,8(sp)
85    2c: 00100793            li  a5,1
86    30: 0ef60663            beq a2,a5,11c <.L4>
87    34: 00050493            mv  s1,a0
88    38: 00858413            addi  s0,a1,8
89    3c: 00361613            slli  a2,a2,0x3
90    40: 00c58bb3            add s7,a1,a2
91    44: 00000993            li  s3,0
92    48: 00000913            li  s2,0
93    4c: 000007b7            lui a5,0x0
94    50: 0007aa03            lw  s4,0(a5) # 0 <trapz>
95    54: 0007aa83            lw  s5,0(a5)
97
97 00000058 <.L3>:
98    58: ff842603            lw  a2,-8(s0)
99    5c: ffc42683            lw  a3,-4(s0)
100   60: 00042503            lw  a0,0(s0)
101   64: 00442583            lw  a1,4(s0)
102   68: 00000097            auipc ra,0x0
103   6c: 000080e7            jalr  ra # 68 <.L3+0x10>
104   70: 00050c13            mv  s8,a0
105   74: 00058b13            mv  s6,a1
106   78: 0004a603            lw  a2,0(s1)
107   7c: 0044a683            lw  a3,4(s1)
108   80: 0084a503            lw  a0,8(s1)
109   84: 00c4a583            lw  a1,12(s1)
110   88: 00000097            auipc ra,0x0
111   8c: 000080e7            jalr  ra # 88 <.L3+0x30>
112   90: 00050613            mv  a2,a0
113   94: 00058693            mv  a3,a1
114   98: 000c0513            mv  a0,s8
115   9c: 000b0593            mv  a1,s6
116   a0: 00000097            auipc ra,0x0
117   a4: 000080e7            jalr  ra # a0 <.L3+0x48>
118   a8: 000a0613            mv  a2,s4
119   ac: 000a8693            mv  a3,s5
120   b0: 00000097            auipc ra,0x0
121   b4: 000080e7            jalr  ra # b0 <.L3+0x58>
122   b8: 00050613            mv  a2,a0
123   bc: 00058693            mv  a3,a1
124   c0: 00098513            mv  a0,s3
```

```
125   c4: 00090593            mv  a1,s2
126   c8: 00000097            auipc ra,0x0
127   cc: 000080e7            jalr  ra # c8 <.L3+0x70>
128   d0: 00050993            mv  s3,a0
129   d4: 00058913            mv  s2,a1
130   d8: 00848493            addi  s1,s1,8
131   dc: 00840413            addi  s0,s0,8
132   e0: f7741ce3            bne s0,s7,58 <.L3>
134
134 000000e4 <.L1>:
135   e4: 00098513            mv  a0,s3
136   e8: 00090593            mv  a1,s2
137   ec: 02c12083            lw  ra,44(sp)
138   f0: 02812403            lw  s0,40(sp)
139   f4: 02412483            lw  s1,36(sp)
140   f8: 02012903            lw  s2,32(sp)
141   fc: 01c12983            lw  s3,28(sp)
142  100: 01812a03            lw  s4,24(sp)
143  104: 01412a83            lw  s5,20(sp)
144  108: 01012b03            lw  s6,16(sp)
145  10c: 00c12b83            lw  s7,12(sp)
146  110: 00812c03            lw  s8,8(sp)
147  114: 03010113            addi  sp,sp,48
148  118: 00008067            ret
150
150 0000011c <.L4>:
151  11c: 00000993            li  s3,0
152  120: 00000913            li  s2,0
153  124: fc1ff06f            j e4 <.L1>
```

Интерес представляет трансформация псевдоинструкции call, а именно странные переходы на предыдущую инструкцию. Поведение ассемблера объясняется очень просто: ассемблер не имел возможности определить целевой адрес перехода, поэтому не мог сформировать корректную инструкцию передачи управления. В результате была сформирована пара инструкций с некорректными значениями непосредственных операндов. Для получения исполняемого кода эта пара инструкций должна быть исправлена компоновщиком.

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

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.

Листинг 11: Таблица перемещений

```
2
2 main.o:     file format elf32-littleriscv
4
4 RELOCATION RECORDS FOR [.text]:
```

```
OFFSET   TYPE              VALUE
00000008 R_RISCV_HI20      .LANCHOR0
00000008 R_RISCV_RELAX     *ABS*
0000000c R_RISCV_LO12_I    .LANCHOR0
0000000c R_RISCV_RELAX     *ABS*
000000bc R_RISCV_CALL      trapz
000000bc R_RISCV_RELAX     *ABS*
000000cc R_RISCV_HI20      .LC2
000000cc R_RISCV_RELAX     *ABS*
000000d0 R_RISCV_LO12_I    .LC2
000000d0 R_RISCV_RELAX     *ABS*
000000d4 R_RISCV_CALL      printf
000000d4 R_RISCV_RELAX     *ABS*



libtrapz.o:     file format elf32-littleriscv

RELOCATION RECORDS FOR [.text]:
OFFSET   TYPE              VALUE
0000004c R_RISCV_HI20      .LC0
0000004c R_RISCV_RELAX     *ABS*
00000050 R_RISCV_LO12_I    .LC0
00000050 R_RISCV_RELAX     *ABS*
00000054 R_RISCV_LO12_I    .LC0+0x00000004
00000054 R_RISCV_RELAX     *ABS*+0x00000004
00000068 R_RISCV_CALL      __adddf3
00000068 R_RISCV_RELAX     *ABS*
00000088 R_RISCV_CALL      __subdf3
00000088 R_RISCV_RELAX     *ABS*
000000a0 R_RISCV_CALL      __muldf3
000000a0 R_RISCV_RELAX     *ABS*
000000b0 R_RISCV_CALL      __muldf3
000000b0 R_RISCV_RELAX     *ABS*
000000c8 R_RISCV_CALL      __adddf3
000000c8 R_RISCV_RELAX     *ABS*
00000030 R_RISCV_BRANCH    .L4
000000e0 R_RISCV_BRANCH    .L3
00000124 R_RISCV_JAL       .L1
```

Так, например, указано, что по адресу 68 следует исправить пару инструкций (тип перемещения "R_RISCV_CALL") так, чтобы результат соответствовал вызову подпрограммы __adddf3.

Компоновка

Компоновщик связывает все объектные файлы и статические библиотеки в единый исполняемый файл, который мы и сможем запустить в дальнейшем. Изучим содержимое

секции “.text” полученного в результате компоновки программы исполняемого файла.

Выходной файл содержит значительно больше строк, чем было до этого (20000!). Это связано с включением стандартной библиотеки при компоновке. Нас интересует лишь небольшой отрывок.

Листинг 12: Фрагмент дизасемблированного файла

```
152    10260: 0ac000ef            jal ra,1030c <__adddf3>
153    10264: 00050c13            mv  s8,a0
154    10268: 00058b13            mv  s6,a1
155    1026c: 0004a603            lw  a2,0(s1)
156    10270: 0044a683            lw  a3,4(s1)
157    10274: 0084a503            lw  a0,8(s1)
158    10278: 00c4a583            lw  a1,12(s1)
159    1027c: 347000ef            jal ra,10dc2 <__subdf3>
196 0001030c <__adddf3>:
197    1030c: 00100837            lui a6,0x100
```

Как видно, адресса были скорректированы компоеновщиком, при этом адрессация изменилась на абсолютную.

4. Создание статической библиотеки

Статическая библиотека является, по сути, архивом объектных файлов, среди которых компоновщик выбирает «полезные» для данной программы:объектный файл считается «полезным», если в нем определяется еще не разрешенный компоновщиком символ.

Рассмотрим список сиволов библиотеки ”libtrapz.a”

Листинг 13: Список символов библиотеки

```
2
2 libtrapz.o:
3          U __adddf3
4 000000e4 t .L1
5 00000058 t .L3
6 0000011c t .L4
7 00000000 r .LC0
8          U __muldf3
9          U __subdf3
10 00000000 T trapz
```

Несложно догадаться, что в выводе утилиты “nm” кодом “T” обозначаются символы, определенные в соответствующем объектном файле, кодом “U” - внешние символы.

Используем статическую библиотеку для сборки программы и изучим таблицу символов исполняемого файла.

Листинг 14: Фрагмент дизасемблированного файла

```
115    101d4: 028000ef            jal ra,101fc <trapz>
```

```
116     101d8: 00050613             mv  a2,a0
117     101dc: 00058693             mv  a3,a1
118     101e0: 00020537             lui a0,0x20
119     101e4: a8050513             addi  a0,a0,-1408 # 1fa80 <__modsi3+0x34>
120     101e8: 396010ef             jal ra,1157e <printf>
121     101ec: 00000513             li  a0,0
122     101f0: 05c12083             lw  ra,92(sp)
123     101f4: 06010113             addi  sp,sp,96
124     101f8: 00008067             ret
126
126 000101fc <trapz>:
127     101fc: fd010113             addi  sp,sp,-48
```

Убеждаемся, что там находится наша функция.

5. Автоматизация процесса сборки

Для автоматизации сборки библиотеки и тестируемой программы был написан Makefile.

Листинг 15: ../Makefile

```
1  CC=riscv64-unknown-elf-gcc
2  AR=riscv64-unknown-elf-ar
3  CFLAGS=-march=rv32iafdc -mabi=ilp32
4  ARFLAGS=-rsc
6
6  SRCDIR=./src
7  BUILDDIR = ./build
9
9  all:
10   make app
12
12 doc:
13   pdflatex doc/report.tex
15
15 clean:
16   rm -f *.aux
17   rm -f *.bbl
18   rm -f *.blg
19   rm -f *.log
20   rm -f *.out
21   rm -f *.pdf
22   rm -f *.toc
23   rm -f *.a
24   rm -f *.o
26
26   rm -rf ${BUILDDIR}
27 lib:
28   mkdir -p build
```

```
30
30    ${CC} ${CFLAGS} ${SRCDIR}/libtrapz/libtrapz.c -o ${BUILDDIR}/libtrapz.o -c
31    ${AR} ${ARFLAGS} libtrapz.a ${BUILDDIR}/libtrapz.o
33
33 app:
34    make lib
35    ${CC} ${CFLAGS} ${SRCDIR}/app/main.c libtrapz.a -o trapz
```

В результате сборки была собрана статическая библиотека libtrapz.a и исполняемый файл тестируемой программы "trapz". Объектные файлы хранятся в дирректории build (Рисунок 5.1).

```
make app
make[1]: Entering directory '/home/dmitri/Загрузки/LowLevelProgramming2021/separate'
make lib
make[2]: Entering directory '/home/dmitri/Загрузки/LowLevelProgramming2021/separate'
mkdir -p build
riscv64-unknown-elf-gcc -march=rv32iafdc -mabi=ilp32 ./src/libtrapz/libtrapz.c -o ./build/libtrapz.o -c
riscv64-unknown-elf-ar -rsc libtrapz.a ./build/libtrapz.o
make[2]: Leaving directory '/home/dmitri/Загрузки/LowLevelProgramming2021/separate'
riscv64-unknown-elf-gcc -march=rv32iafdc -mabi=ilp32 ./src/app/main.c libtrapz.a -o trapz
make[1]: Leaving directory '/home/dmitri/Загрузки/LowLevelProgramming2021/separate'
dmitri@dmitri-Lenovo-G505s:~/Загрузки/LowLevelProgramming2021/separate$ ls
build  doc  include  libtrapz.a  Makefile  src  trapz
dmitri@dmitri-Lenovo-G505s:~/Загрузки/LowLevelProgramming2021/separate$ ls build/
libtrapz.o
```

Рисунок 5.1. Результат сборки программы

6. Проверка работоспособности

Для проверки работоспособности программ, они были собраны тулчейном x86_64. Результат выполнения приведен на рис. 6.1

Тестирующая программа находит значение определенного интергала для заданной табличной функции:

Таблица 6.1

Заданная табличная функция

X	1.1	2.28	4.6	5	10.001
Y	0.8	0.16	0.8	0.16	0.8

```
dmitri@dmitri-Lenovo-G505s:~/Загрузки/LowLevelProgramming2021/separate$ gcc src/{libtrapz/libtrapz.c,app/main.c} -o trapz
dmitri@dmitri-Lenovo-G505s:~/Загрузки/LowLevelProgramming2021/separate$ ./trapz
4.272480
```

Рисунок 6.1. Результат выполнения программы

Проверим верность результата, вызвал аналогичную функцию в среде Matlab

```
>> x = [1.1, 2.28, 4.6, 5, 10.001];
>> y = [0.8, 0.16, 0.8, 0.16, 0.8];
>> trapz(x,y)
ans =  4.2725
```

Рисунок 6.2. Результат функции в среде Matlab

7. Выводы

В ходе выполнения лабораторной работы была разработана программа на языке C, реализующая заданную функциональность. Была осуществлена сборка программы по шагам для ISA RV32I, проанализированы выводы препроцессора, компилятора, компоновщика. Создана статическая библиотека libtrapz.a, на основе которой собрана тестирующая программа. Для автоматизации сборки был разработан Makefile с павилами сборки.