
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: 14

Выполнил студент гр. 3530901/00002 _____ Чэнь Аосюань
(подпись)

Принял преподаватель _____ Д.С.Степанов
(подпись)

“ _ ” _____ 2021 г.

Санкт-Петербург

2021

остановка задачи

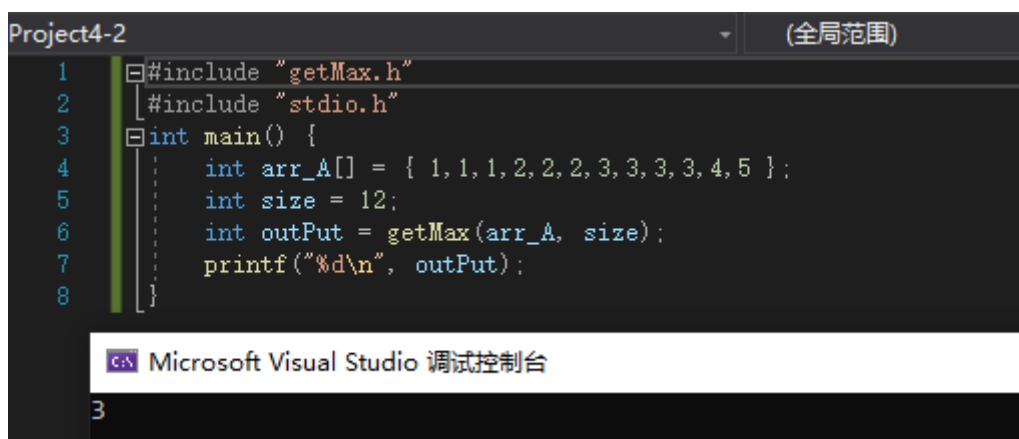
1. Изучить методические материалы, опубликованные на сайте курса.
2. Установить пакет средств разработки “SiFive GNU Embedded Toolchain” для RISC-V.
3. На языке C разработать функцию, реализующую определенную вариантом задания функциональность. Поместить определение функции в отдельный исходный файл, 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 исполняемом 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.

Задача

Определение наиболее часто встречающегося в массиве значения

Программа на язык c

С помощью Visual Studio2019 разработал тествую программу(Рис 1) подпрограмму(Рис 2) и заголовочный файл подпрограммы(Рис 3)

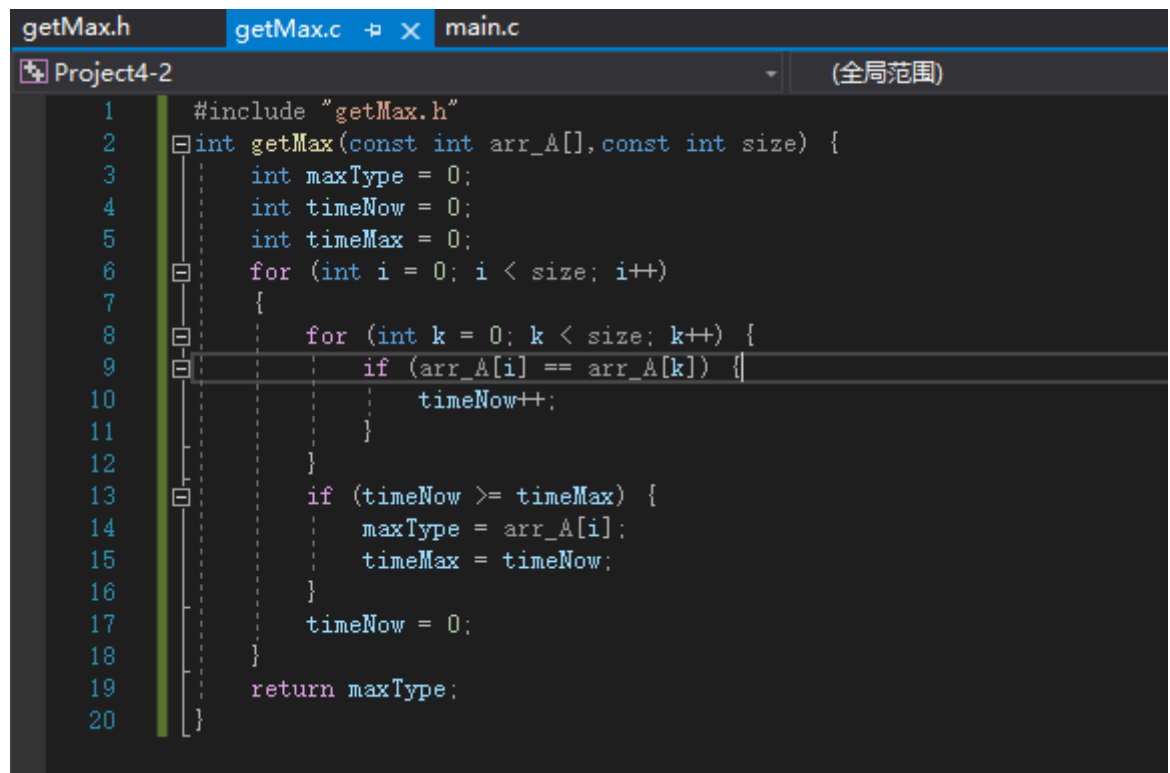


```
Project4-2 (全局范围)
1  #include "getMax.h"
2  #include "stdio.h"
3  int main() {
4      int arr_A[] = { 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 5 };
5      int size = 12;
6      int outPut = getMax(arr_A, size);
7      printf("%d\n", outPut);
8  }
```

Microsoft Visual Studio 调试控制台

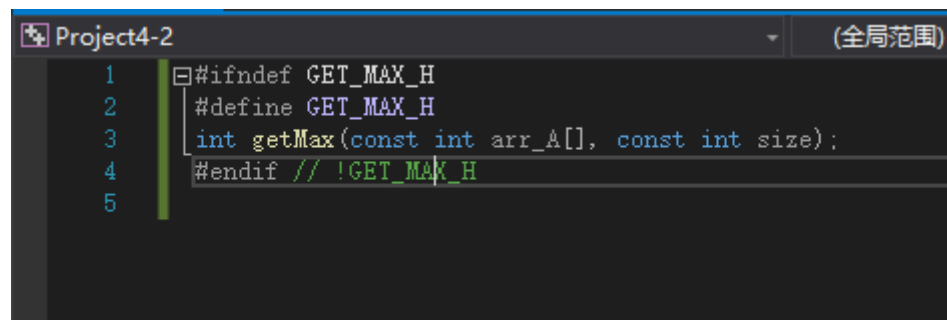
3

(Рис 1 тествая программа и вывод 3)



```
getMax.h  getMax.c  main.c
Project4-2 (全局范围)
1  #include "getMax.h"
2  int getMax(const int arr_A[], const int size) {
3      int maxType = 0;
4      int timeNow = 0;
5      int timeMax = 0;
6      for (int i = 0; i < size; i++)
7      {
8          for (int k = 0; k < size; k++) {
9              if (arr_A[i] == arr_A[k]) {
10                  timeNow++;
11              }
12          }
13          if (timeNow >= timeMax) {
14              maxType = arr_A[i];
15              timeMax = timeNow;
16          }
17          timeNow = 0;
18      }
19      return maxType;
20 }
```

(Рис2 Подпрограмма на язык с)



```
Project4-2 (全局范围)
1  #ifndef GET_MAX_H
2      #define GET_MAX_H
3      int getMax(const int arr_A[], const int size);
4  #endif // !GET_MAX_H
5
```

(Рис 3 Заголовочный файл подпрограммы)

Сборка простейшей программы «по шагам»

Первый шаг – Препроцессирование

riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -O1 -v -E main.c -o main.i

```
# 1 "main.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "main.c"
# 1 "getMax.h" 1

|
int getMax(const int arr_A[], const int size);
# 2 "main.c" 2

# 3 "main.c"
int main() {
    int arr_A[] = { 1,1,1,2,2,2,3,3,3,4,5 };
    int size = 12;
    int outPut = getMax(arr_A, size);
    printf("%d\n", outPut);
}
```

(Рис 3 График main.i)

riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -O1 -v -E getMax.c -o
getMax.i

```
1 # 1 "getMax.c"
2 # 1 "<built-in>"
3 # 1 "<command-line>"
4 # 1 "getMax.c"
5 # 1 "getMax.h" 1
6
7
8 int getMax(const int arr_A[], const int size);
9 # 2 "getMax.c" 2
10 int getMax(const int arr_A[],const int size) {
11     int maxType = 0;
12     int timeNow = 0;
13     int timeMax = 0;
14     for (int i = 0; i < size; i++)
15     {
16         for (int k = 0; k < size; k++) {
17             if (arr_A[i] == arr_A[k]) {
18                 timeNow++;
19             }
20         }
21         if (timeNow >= timeMax) {
22             maxType = arr_A[i];
23             timeMax = timeNow;
24         }
25         timeNow = 0;
26     }
27     return maxType;
28 }
29
```

(Рис 4 График getMax.i)

Видно что в результате препроцессирования . Код мало отличается от исходного кода. Кроме некоторых символов как #1 #2 который показал используются для передачи информации об исходном тексте из препроцессора в компилятор.

Второй шаг – Компиляция

```
1  .file "main.c"
2  .option nopic
3  .attribute arch, "rv64i2p0"
4  .attribute unaligned_access, 0
5  .attribute stack_align, 16
6  .text
7  .align 2
8  .globl main
9  .type main, @function
10 main:
11     addi    sp, sp, -64
12     sd      ra, 56(sp)
13     lui     a5, %hi(. LANCHOR0)
14     addi    a5, a5, %lo(. LANCHOR0)
15     ld      a0, 0(a5)
16     ld      a1, 8(a5)
17     ld      a2, 16(a5)
18     ld      a3, 24(a5)
19     ld      a4, 32(a5)
20     ld      a5, 40(a5)
21     sd      a0, 0(sp)
22     sd      a1, 8(sp)
23     sd      a2, 16(sp)
24     sd      a3, 24(sp)
25     sd      a4, 32(sp)
26     sd      a5, 40(sp)
27     li      a1, 12
28     mv      a0, sp
29     call    getMax
30     mv      a1, a0
31     lui     a0, %hi(. LC1)
32     addi    a0, a0, %lo(. LC1)
33     call    printf
34     li      a0, 0
35     ld      ra, 56(sp)
36     addi    sp, sp, 64
37     jr      ra
38     .size   main, .-main
39     .section .rodata
40     .align 3
41     .set    . LANCHOR0, . + 0
```

(Рис5 график main.s 1-41)

```

42  ▢ .LC0:
43      .word 1
44      .word 1
45      .word 1
46      .word 2
47      .word 2
48      .word 2
49      .word 3
50      .word 3
51      .word 3
52      .word 3
53      .word 4
54      .word 5
55      .section .rodata.strl.8,"aMS",@progbits,1
56      .align 3
57  ▢ .LC1:
58      .string "%d\n"
59      .ident "GCC: (SiFive GCC 8.3.0-2020.04.1) 8.3.0"
60

```

(Рис6 График main.s 42-60)

Комант используемый: riscv64-unknown-elf-gcc.exe -march=rv64i -mabi=lp64 -O1 -S main.i -o main.s

```

1  .file "getMax.c"
2  .option nopic
3  .attribute arch, "rv64i2p0"
4  .attribute unaligned_access, 0
5  .attribute stack_align, 16
6  .text
7  .align 2
8  .globl getMax
9  .type getMax, @function
10 ▢ getMax:
11      ble a1,zero,.L7
12      mv t3,a0
13      addiw a3,a1,-1
14      slli a3,a3,32
15      srli a3,a3,32
16      slli a3,a3,2
17      addi a5,a0,4
18      add a3,a3,a5
19      mv a6,a0
20      li a7,0
21      li a0,0
22      li t1,0
23      j .L3
24 ▢ .L4:
25      addi a5,a5,4
26      beq a5,a3,.L10
27 ▢ .L5:
28      lw a4,0(a5)
29      bne a4,a2,.L4
30      addiw a1,a1,1

```

(Рис7 график getMax.s 1-30)

```

31 | j .L4
32 | .L10:
33 | blt a1,a7,.L6
34 | mv a7,a1
35 | mv a0,a2
36 | .L6:
37 | addi a6,a6,4
38 | beq a6,a3,.L2
39 | .L3:
40 | lw a2,0(a6)
41 | mv a5,t3
42 | mv a1,t1
43 | j .L5
44 | .L7:
45 | li a0,0
46 | .L2:
47 | ret
48 | .size getMax,.-getMax
49 | .ident "GCC: (SiFive GCC 8.3.0-2020.04.1) 8.3.0"
50 |

```

(Рис8 график getMax.s 31-50)

Комант используемый: `riscv64-unknown-elf-gcc.exe -march=rv64i -mabi=lp64 -O1 -S getMax.i -o getMax.s`

Видно что переходил на код для Risc-V и тоже одиноговые функции как в исходных кодах.

Третий шаг – Ассемблирование

Для ассемблировании файла main.s и getMax.s

Комант используемый: `riscv64-unknown-elf-gcc.exe -march=rv64i -mabi=lp64 -c main.s -o main.o`

`riscv64-unknown-elf-gcc.exe -march=rv64i -mabi=lp64 -c getMax.s -o getMax.o`

Для получения заголовки секций

используем: `riscv64-unknown-elf-objdump -h main.o`

```

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .text          00000074  0000000000000000  0000000000000000  00000040  2**2
    CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data          00000000  0000000000000000  0000000000000000  000000b4  2**0
    CONTENTS, ALLOC, LOAD, DATA
  2 .bss           00000000  0000000000000000  0000000000000000  000000b4  2**0
    ALLOC
  3 .rodata        00000030  0000000000000000  0000000000000000  000000b8  2**3
    CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .rodata.str1.8 00000004  0000000000000000  0000000000000000  000000e8  2**3
    CONTENTS, ALLOC, LOAD, READONLY, DATA
  5 .comment       00000029  0000000000000000  0000000000000000  000000ec  2**0
    CONTENTS, READONLY
  6 .riscv.attributes 0000001c  0000000000000000  0000000000000000  00000115  2**0
    CONTENTS, READONLY

```

(Рис 9 заголовки секции main)

Для симполов файла

используем: `riscv64-unknown-elf-objdump -t main.o`

```

main.o:      file format elf64-littleriscv

SYMBOL TABLE:
0000000000000000 1      df *ABS* 0000000000000000 main.c
0000000000000000 1      d  .text 0000000000000000 .text
0000000000000000 1      d  .data 0000000000000000 .data
0000000000000000 1      d  .bss  0000000000000000 .bss
0000000000000000 1      d  .rodata 0000000000000000 .rodata
0000000000000000 1      d  .rodata 0000000000000000 .LANCHOR0
0000000000000000 1      d  .rodata.str1.8 0000000000000000 .rodata.str1.8
0000000000000000 1      d  .rodata.str1.8 0000000000000000 .LC1
0000000000000000 1      d  .comment 0000000000000000 .comment
0000000000000000 1      d  .riscv.attributes 0000000000000000 .riscv.attributes
0000000000000000 g      F  .text 0000000000000074 main
0000000000000000      *UND* 0000000000000000 getMax
0000000000000000      *UND* 0000000000000000 printf

```

(Рис 10 Симполов файла main)

Для содержимоя секции “.text”

используем: `riscv64-unknown-elf-objdump -s -j .text main.o`

```

C:\Users\10367> riscv64-unknown-elf-objdump -s -j .text main.o

main.o:      file format elf64-littleriscv

Contents of section .text:
0000 130101fc 233c1102 b7070000 93870700 ....#<.....
0010 03b50700 83b58700 03b60701 83b68701 .....
0020 03b70702 83b78702 2330a100 2334b100 .....#0..#4..
0030 2338c100 233cd100 2330e102 2334f102 #8..#<..#0..#4..
0040 9305c000 13050100 97000000 e7800000 .....
0050 93050500 37050000 13050500 97000000 ....7.....
0060 e7800000 13050000 83308103 13010104 .....0.....
0070 67800000 g...

C:\Users\10367>

```

(Рис11 Содержание секции main)

Для таблице переменных

используем: riscv64-unknown-elf-objdump -r main.o

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

RELOCATION RECORDS FOR [.text]:
OFFSET      TYPE      VALUE
0000000000000008 R_RISCV_HI20      .LANCHOR0
0000000000000008 R_RISCV_RELAX      *ABS*
000000000000000c R_RISCV_LO12_I     .LANCHOR0
000000000000000c R_RISCV_RELAX      *ABS*
0000000000000048 R_RISCV_CALL       getMax
0000000000000048 R_RISCV_RELAX      *ABS*
0000000000000054 R_RISCV_HI20      .LC1
0000000000000054 R_RISCV_RELAX      *ABS*
0000000000000058 R_RISCV_LO12_I     .LC1
0000000000000058 R_RISCV_RELAX      *ABS*
000000000000005c R_RISCV_CALL       printf
000000000000005c R_RISCV_RELAX      *ABS*
```

(Рис 12 Таблица переменных main)

Для getMax.o

Используем: riscv64-unknown-elf-objdump -h getMax.o

riscv64-unknown-elf-objdump -t getMax.o

riscv64-unknown-elf-objdump -s -j .text getMax.o

riscv64-unknown-elf-objdump -r getMax.o

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

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .text          00000078  0000000000000000  0000000000000000  00000040  2**2
    CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data           00000000  0000000000000000  0000000000000000  000000b8  2**0
    CONTENTS, ALLOC, LOAD, DATA
  2 .bss            00000000  0000000000000000  0000000000000000  000000b8  2**0
    ALLOC
  3 .comment         00000029  0000000000000000  0000000000000000  000000b8  2**0
    CONTENTS, READONLY
  4 .riscv.attributes 0000001c  0000000000000000  0000000000000000  000000e1  2**0
    CONTENTS, READONLY

C:\Users\10367>
```

(Рис 13 Заголовки секции getMax)

```

C:\Users\10367> riscv64-unknown-elf-objdump -t getMax.o

getMax.o:      file format elf64-littleriscv

SYMBOL TABLE:
0000000000000000 l      df *ABS* 0000000000000000 getMax.c
0000000000000000 l      d  .text 0000000000000000 .text
0000000000000000 l      d  .data 0000000000000000 .data
0000000000000000 l      d  .bss 0000000000000000 .bss
0000000000000070 l      .text 0000000000000000 .L7
0000000000000060 l      .text 0000000000000000 .L3
000000000000004c l      .text 0000000000000000 .L10
0000000000000034 l      .text 0000000000000000 .L4
0000000000000058 l      .text 0000000000000000 .L6
0000000000000074 l      .text 0000000000000000 .L2
000000000000003c l      .text 0000000000000000 .L5
0000000000000000 l      d  .comment 0000000000000000 .comment
0000000000000000 l      d  .riscv.attributes 0000000000000000 .riscv.attributes
0000000000000000 g      F  .text 0000000000000078 getMax

```

(Рис 14 Симполов файла getMax)

```

getMax.o:      file format elf64-littleriscv

RELOCATION RECORDS FOR [.text]:
OFFSET          TYPE          VALUE
0000000000000000 R_RISCV_BRANCH  .L7
0000000000000030 R_RISCV_JAL     .L3
0000000000000038 R_RISCV_BRANCH  .L10
0000000000000040 R_RISCV_BRANCH  .L4
0000000000000048 R_RISCV_JAL     .L4
000000000000004c R_RISCV_BRANCH  .L6
000000000000005c R_RISCV_BRANCH  .L2
000000000000006c R_RISCV_JAL     .L5

```

(Рис15 Содержание секции getMax)

```

C:\Users\10367> riscv64-unknown-elf-objdump -s -j .text getMax.o

getMax.o:      file format elf64-littleriscv

Contents of section .text:
 0000 6358b006 130e0500 9b86f5ff 93960602  cX.....
 0010 93d60602 93962600 93074500 b386f600  .....&...E....
 0020 13080500 93080000 13050000 13030000  .....
 0030 6f000003 93874700 638ad700 03a70700  o.....G.C.....
 0040 e31ac7fe 9b851500 6ff0dffe 63c61501  .....O...C...
 0050 93880500 13050600 13084800 630cd800  .....H.C...
 0060 03260800 93070e00 93050300 6ff01ffd  .&.....O...
 0070 13050000 67800000  ....g...

```

(Рис 17 Таблица переменных getMax)

В графике

.text – секция кода, в которой содержатся коды инструкций (название секции

обусловлено историческими причинами);

.data – секция инициализированных данных;

.bss – секция данных, инициализированных нулями (название секции также обусловлено

историческими причинами);

.comment – секция данных о версиях размером 12 байт.

UND = getMax

Сейчас используем riscv64-unknown-elf-objdump -d -M no-aliases -r main.o и riscv64-unknown-elf-objdump -d -M no-aliases -r getMax.o для Дизассемблирования

```

C:\Users\10367> riscv64-unknown-elf-objdump -d -M no-aliases -r main.o
main.o:      file format elf64-littleriscv

Disassembly of section .text:

0000000000000000 <main>:
 0:   fc010113          addi    sp,sp,-64
 4:   02113c23          sd      ra,56(sp)
 8:   000007b7          lui     a5,0x0
                        8: R_RISCV_HI20 .LANCHOR0
                        8: R_RISCV_RELAX *ABS*
 c:   00078793          addi    a5,a5,0 # 0 <main>
                        c: R_RISCV_LO12_I .LANCHOR0
                        c: R_RISCV_RELAX *ABS*
10:   0007b503          ld      a0,0(a5)
14:   0087b583          ld      a1,8(a5)
18:   0107b603          ld      a2,16(a5)
1c:   0187b683          ld      a3,24(a5)
20:   0207b703          ld      a4,32(a5)
24:   0287b783          ld      a5,40(a5)
28:   00a13023          sd      a0,0(sp)
2c:   00b13423          sd      a1,8(sp)
30:   00c13823          sd      a2,16(sp)
34:   00d13c23          sd      a3,24(sp)
38:   02e13023          sd      a4,32(sp)
3c:   02f13423          sd      a5,40(sp)
40:   00c00593          addi    a1,zero,12
44:   00010513          addi    a0,sp,0
48:   00000097          auipc   ra,0x0
                        48: R_RISCV_CALL  getMax
                        48: R_RISCV_RELAX *ABS*
4c:   000080e7          jalr    ra,0(ra) # 48 <main+0x48>
50:   00050593          addi    a1,a0,0
54:   00000537          lui     a0,0x0
                        54: R_RISCV_HI20 .LC1
                        54: R_RISCV_RELAX *ABS*
58:   00050513          addi    a0,a0,0 # 0 <main>
                        58: R_RISCV_LO12_I .LC1
                        58: R_RISCV_RELAX *ABS*
5c:   00000097          auipc   ra,0x0
                        5c: R_RISCV_CALL  printf
                        5c: R_RISCV_RELAX *ABS*
60:   000080e7          jalr    ra,0(ra) # 5c <main+0x5c>
64:   00000513          addi    a0,zero,0
68:   03813083          ld      ra,56(sp)
6c:   04010113          addi    sp,sp,64
70:   00008067          jalr    zero,0(ra)

```

(Рис18 Дизассемблирование main)

```

getMax.o:      file format elf64-littleriscv

Disassembly of section .text:

0000000000000000 <getMax>:
   0:  06b05863                bge      zero,a1,70 <.L7>
                        0: R_RISCV_BRANCH      .L7
   4:  00050e13                addi     t3,a0,0
   8:  fff5869b                addiw    a3,a1,-1
  c:  02069693                slli     a3,a3,0x20
 10:  0206d693                srli     a3,a3,0x20
 14:  00269693                slli     a3,a3,0x2
 18:  00450793                addi     a5,a0,4
 1c:  00f686b3                add      a3,a3,a5
 20:  00050813                addi     a6,a0,0
 24:  00000893                addi     a7,zero,0
 28:  00000513                addi     a0,zero,0
 2c:  00000313                addi     t1,zero,0
 30:  0300006f                jal      zero,60 <.L3>
                        30: R_RISCV_JAL      .L3

0000000000000034 <.L4>:
  34:  00478793                addi     a5,a5,4
  38:  00d78a63                beq      a5,a3,4c <.L10>
                        38: R_RISCV_BRANCH      .L10

000000000000003c <.L5>:
  3c:  0007a703                lw       a4,0(a5)
  40:  fec71ae3                bne      a4,a2,34 <.L4>
                        40: R_RISCV_BRANCH      .L4
  44:  0015859b                addiw    a1,a1,1
  48:  fedff06f                jal      zero,34 <.L4>
                        48: R_RISCV_JAL      .L4

```

(Рис19 Дизассемблирование getMax1-48)

```

48: fedff06f          jal     zero,34 <.L4>
                        48: R_RISCV_JAL .L4

000000000000004c <.L10>:
4c: 0115c663          blt     a1,a7,58 <.L6>
                        4c: R_RISCV_BRANCH .L6

50: 00058893          addi    a7,a1,0
54: 00060513          addi    a0,a2,0

0000000000000058 <.L6>:
58: 00480813          addi    a6,a6,4
5c: 00d80c63          beq     a6,a3,74 <.L2>
                        5c: R_RISCV_BRANCH .L2

0000000000000060 <.L3>:
60: 00082603          lw      a2,0(a6)
64: 000e0793          addi    a5,t3,0
68: 00030593          addi    a1,t1,0
6c: fd1ff06f          jal     zero,3c <.L5>
                        6c: R_RISCV_JAL .L5

0000000000000070 <.L7>:
70: 00000513          addi    a0,zero,0

0000000000000074 <.L2>:
74: 00008067          jalr    zero,0(ra)

```

(Рис20 Дизассемблирование getMax48-74)

Похоже с кодом который получил в прошлом шаге как код для Risc-v

Четвертый шаг – Компоновка

По команде `riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -v main.o getMax.o` и `riscv64-unknown-elf-objdump -j .text -d -M no-aliases a.out >a.ds` получаем `a.out` и `a.ds`

```

0000000000010158 <main>:
10158: fc010113      addi    sp,sp,-64
1015c: 02113c23      sd      ra,56(sp)
10160: 0001c7b7      lui     a5,0x1c
10164: 50078793      addi    a5,a5,1280 # 1c500 <__clzdi2+0x3e>
10168: 0007b503      ld      a0,0(a5)
1016c: 0087b583      ld      a1,8(a5)
10170: 0107b603      ld      a2,16(a5)
10174: 0187b683      ld      a3,24(a5)
10178: 0207b703      ld      a4,32(a5)
1017c: 0287b783      ld      a5,40(a5)
10180: 00a13023      sd      a0,0(sp)
10184: 00b13423      sd      a1,8(sp)
10188: 00c13823      sd      a2,16(sp)
1018c: 00d13c23      sd      a3,24(sp)
10190: 02e13023      sd      a4,32(sp)
10194: 02f13423      sd      a5,40(sp)
10198: 00c00593      addi    a1,zero,12
1019c: 00010513      addi    a0,sp,0
101a0: 024000ef      jal     ra,101c4 <getMax>
101a4: 00050593      addi    a1,a0,0
101a8: 0001c537      lui     a0,0x1c
101ac: 53050513      addi    a0,a0,1328 # 1c530 <__clzdi2+0x6e>
101b0: 228000ef      jal     ra,103d8 <printf>
101b4: 00000513      addi    a0,zero,0
101b8: 03813083      ld      ra,56(sp)
101bc: 04010113      addi    sp,sp,64
101c0: 00008067      jalr    zero,0(ra) # 1011a <__do_global_dtors_aux+0x1c>

00000000000101c4 <getMax>:
101c4: 06b05863      bge     zero,a1,10234 <getMax+0x70>
101c8: 00050e13      addi    t3,a0,0
101cc: fff5869b      addiw   a3,a1,-1
101d0: 02069693      slli    a3,a3,0x20
101d4: 0206d693      srli    a3,a3,0x20
101d8: 00269693      slli    a3,a3,0x2
101dc: 00450793      addi    a5,a0,4
101e0: 00f686b3      add     a3,a3,a5

```

(Рис21 Фрагмент код a.out)

101e0:	00f686b3	add	a3,a3,a5
101e4:	00050813	addi	a6,a0,0
101e8:	00000893	addi	a7,zero,0
101ec:	00000513	addi	a0,zero,0
101f0:	00000313	addi	t1,zero,0
101f4:	0300006f	jal	zero,10224 <getMax+0x60>
101f8:	00478793	addi	a5,a5,4
101fc:	00d78a63	beq	a5,a3,10210 <getMax+0x4c>
10200:	0007a703	lw	a4,0(a5)
10204:	fec71ae3	bne	a4,a2,101f8 <getMax+0x34>
10208:	0015859b	addiw	a1,a1,1
1020c:	fedff06f	jal	zero,101f8 <getMax+0x34>
10210:	0115c663	blt	a1,a7,1021c <getMax+0x58>
10214:	00058893	addi	a7,a1,0
10218:	00060513	addi	a0,a2,0
1021c:	00480813	addi	a6,a6,4
10220:	00d80c63	beq	a6,a3,10238 <getMax+0x74>
10224:	00082603	lw	a2,0(a6)
10228:	000e0793	addi	a5,t3,0
1022c:	00030593	addi	a1,t1,0 # 1014e <frame_dummy+0x16>
10230:	fd1ff06f	jal	zero,10200 <getMax+0x3c>
10234:	00000513	addi	a0,zero,0
10238:	00008067	jalr	zero,0(ra)

000000000001023c <atexit>:

1023c:	85aa	c.mv	a1,a0
1023e:	4681	c.li	a3,0
10240:	4601	c.li	a2,0
10242:	4501	c.li	a0,0
10244:	0060206f	jal	zero,1224a <__register_exitproc>

(Рис22 Фрагмент код a.out)

0000000000010248 <exit>:

10248:	1141	c.addi	sp,-16
1024a:	4581	c.li	a1,0
1024c:	e022	c.sdsp	s0,0(sp)
1024e:	e406	c.sdsp	ra,8(sp)
10250:	842a	c.mv	s0,a0
10252:	070020ef	jal	ra,122c2 <__call_exitprocs>
10256:	74818793	addi	a5,gp,1864 # 1f400 <_global_impure_ptr>
1025a:	6388	c.ld	a0,0(a5)
1025c:	6d3c	c.ld	a5,88(a0)
1025e:	c391	c.beqz	a5,10262 <exit+0x1a>
10260:	9782	c.jalr	a5
10262:	8522	c.mv	a0,s0
10264:	3a30a0ef	jal	ra,1ae06 <_exit>

(Рис23 Фрагмент код a.out)

Создать библиотеки

Добовляеь getMax.o к lib

Используем riscv64-unknown-ef-ar -rsc lib.a getMax.o

И main тоже

Используем riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -O1 --save-temps main.c lib.a

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

```
00000000000010158 g      F .text 0000000000000003e main
0000000000001f400 g      O .sbss 00000000000000008 __malloc_max_total_mem
0000000000001acfe g      F .text 0000000000000000c __swbuf
000000000000163a8 g      F .text 00000000000000008 __sclose
00000000000010198 g      F .text 00000000000000078 getMax
000000000000192cc g      F .text 0000000000000000a fclose
00000000000014e92 g      F .text 00000000000000660 __malloc_r
000000000000190ee g      F .text 00000000000000024 __ascii_wctomb
00000000000012afe g      F .text 0000000000000008c __fwalk
000000000000154f2 g      F .text 0000000000000000a __mbtowc_r
0000000000001c402 g      F .text 00000000000000092 .hidden __divsi3
000000000000127bc g      F .text 000000000000000d8 __malloc_trim_r
000000000000163b0 g      F .text 000000000000000ea strcmp
00000000000019040 g      F .text 00000000000000010 vfiprintf
0000000000001b662 g      F .text 000000000000004fc .hidden __multf3
0000000000001628a g      F .text 0000000000000004c sprintf
0000000000001d180 g      O .rodata 0000000000000100 .hidden __clz_tab
0000000000001f3f8 g      O .sbss 00000000000000008 _PathLocale
00000000000010210 g      F .text 0000000000000000c atexit
00000000000019112 g      F .text 00000000000000040 __write_r
00000000000014d48 g      F .text 0000000000000000c setlocale
0000000000001f3e0 g      O .sdata 00000000000000008 __impure_ptr
00000000000012358 g      F .text 00000000000000196 __sflush_r
0000000000001b516 g      F .text 000000000000000a6 .hidden __gttf2
0000000000001a07a g      F .text 00000000000000b86 __svfiprintf_r
000000000000154fc g      F .text 00000000000000040 __ascii_mbtowc
0000000000001bb5e g      F .text 000000000000005de .hidden __subtbf3
00000000000015dfa g      F .text 00000000000000056 __ulp
000000000000127ac g      F .text 00000000000000010 __fp_unlock_all
00000000000014ce4 g      F .text 00000000000000006 localeconv
00000000000014d54 g      F .text 00000000000000082 __swatbuf_r
0000000000001e2a0 g      .data 00000000000000000 __DATA_BEGIN__
0000000000001af42 g      F .text 00000000000000032 __write
0000000000001f3f8 g      .sdata 00000000000000000 __edata
0000000000001f488 g      .bss 00000000000000000 __end
000000000000192d6 g      F .text 000000000000000c6 __fputwc
00000000000016312 g      F .text 00000000000000054 __swrite
0000000000001f3f0 g      O .sdata 00000000000000008 __malloc_trim_threshold
0000000000001021c g      F .text 00000000000000020 exit
00000000000018376 g      F .text 00000000000000cca __vfiprintf_r
00000000000012b8a g      F .text 00000000000000094 __fwalk_reent
00000000000015cb2 g      F .text 00000000000000148 __mdiff
0000000000001c464 g      F .text 00000000000000028 .hidden __modsi3
00000000000012796 g      F .text 00000000000000002 __sfp_lock_release
00000000000013dd4 g      F .text 00000000000000ece __ldtoa_r
0000000000001ce30 g      O .rodata 0000000000000101 __ctype_
0000000000001aea6 g      F .text 00000000000000032 __read
0000000000001c464 g      F .text 00000000000000028 .hidden __moddi3
0000000000001adda g      F .text 0000000000000002c __exit
```

```
output: main.o getMax.a
mingw32-gcc-9.2.0.exe main.o getMax.a -o output

main.o:main.c
mingw32-gcc-9.2.0.exe -c main.c

getMax.a:getMax.o getMax.h
mingw32-gcc-ar.exe -rsc getMax.a getMax.o

getMax.o:
mingW32-gcc-9.2.0.exe -c getMax.c
clean:
rm -f *.o *.a *.i *.s
```

(Рис24 статический makefile)

Выход лаба

В данной работе исследовали riscv64-unknown-elf-gcc для получения
препроцессирования компиляции асемблирования компоновки и makefile
с кодом с