САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И ТЕХНОЛОГИЙ
ВЫСШАЯ ШКОЛА ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ И СУПЕРКОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №4

Дисциплина: Низкоуровневое программирование
Тема: Раздельная компиляция

Работу выполнил: Чевычелов Д. А.
Группа: 3530901/10003
Преподаватель: Коренев Д. А.

Санкт-Петербург
2022

Оглавление

1. **ТЗ**

Установить пакет средств разработки “SiFive GNU Embedded Toolchain” для RISC-V.

Поместить определение функции в отдельный исходный файл, оформить заголовочный файл. Разработать тестовую программу на языке С.

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.

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.

Вариант: найти сумму всех элементов массива. Если сумма меньше 50 – увеличить значения всех элементов на 7.

2. **Метод решения**

Тестовая программа передает в функцию, находящуюся в отдельном файле, массив, его длину, максимальную сумму для условия задачи, а также значение, на которое будут увеличены элементы. Функция sumOf сначала проходит по всем элементам массива и считает их сумму, а затем, в зависимости от выполнения услвия задачи, входит в новый цикл, где добавляет к каждому элементу заданной значение и запичывает обратно.

3. **Программа на языке С**

Программа состоит из трех файлов main.c (тестовая программа), sumOf.c (основная функция) и sumOf.h (заголовок).

```c
#include "sumOf.h"
#include <stdio.h>
int main() {
    int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    int size = 10;
    int maxSum = 50;
    int toSum = 7;
    //int r[10];
    sumOf(a, size, maxSum, toSum);
    for (int i = 0; i < 10; i++)
        printf("%d ", a[i]);
    scanf("1");
    return 0;
}
```

Рисунок 1 main.c

```c
#include <stdio.h>
#include "sumOf.h"
void sumOf(int *a, int size, int maxSum, int toSum) {
    int i;
    int sum = 0;
    for (i=0; i < size; i++) {
        sum += a[i];
    }
    printf("%d\n", sum);
    if (sum < maxSum) {
        for (i=0; i < size; i++) {
            a[i] = a[i] + toSum;
        }
    }
}

```

Рисунок 2 sumOf.c

```c
#ifndef SUMOF_H
#define SUMOF_H
void sumOf(int *a, int size, int maxSum, int toSum);
#endif
```

Рисунок 3 sumOf.h

Результат работы программы:

```
45
7 8 9 10 11 12 13 14 15 16
```

4. Компиляция и сборка

4.1 Препроцессирование

Препроцессирование выполняется следующими командами:

```
riscv64-unknown-elf-gcc -march=rv32ic -mabi=ilp32 -O1 -E main.c -o main.i
riscv64-unknown-elf-gcc -march=rv32ic -mabi=ilp32 -O1 -E sumOf.c -o sumOf.i
```

Результат препроцессирования содержится в файлах main.i и sumOf.i. :

main.i:

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

 # 3 "main.c" 2

 # 3 "main.c"
int main() {
** int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};**
** int size = 10;**

```
    int maxSum = 50;
    int toSum = 7;

    sumOf(a, size, maxSum, toSum);
    for (int i = 0; i < 10; i++)
        printf("%d ", a[i]);
    scanf("1");
    return 0;
}
```

sumOf.i:

```
# 1 "sumOf.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "sumOf.c"
----------------
# 2 "sumOf.c" 2
# 1 "sumOf.h" 1



# 3 "sumOf.h"
void sumOf(int *a, int size, int maxSum, int toSum);
# 3 "sumOf.c" 2
void sumOf(int *a, int size, int maxSum, int toSum) {
    int i;
    int sum = 0;
    for (i=0; i < size; i++) {
        sum += a[i];
    }
    printf("%d\n", sum);
    if (sum < maxSum) {
        for (i=0; i < size; i++) {
            a[i] = a[i] + toSum;
        }
    }
}
```

4.2 Компиляция

Компиляция осуществляется следующими командами:

```
riscv64-unknown-elf-gcc -march=rv32ic -mabi=ilp32 -O1 -S -fpreprocessed main.i -o main.s
riscv64-unknown-elf-gcc -march=rv32ic -mabi=ilp32 -O1 -S -fpreprocessed sumOf.i -o sumOf.s
```

Результат препроцессирования содержится в файлах main.s и sumOf.s:

main.s:

```
    .file   "main.c"
    .option nopic
    .attribute arch, "rv32i2p0_c2p0"
```

```
    .attribute unaligned_access, 0
    .attribute stack_align, 16
    .text
    .align  1
    .globl  main
    .type   main, @function
main:
    addi    sp,sp,-64
    sw  ra,60(sp)
    sw  s0,56(sp)
    sw  s1,52(sp)
    sw  s2,48(sp)
    lui a5,%hi(.LANCHOR0)
    addi    a5,a5,%lo(.LANCHOR0)
    lw  t3,0(a5)
    lw  t1,4(a5)
    lw  a7,8(a5)
    lw  a6,12(a5)
    lw  a0,16(a5)
    lw  a1,20(a5)
    lw  a2,24(a5)
    lw  a3,28(a5)
    lw  a4,32(a5)
    lw  a5,36(a5)
    sw  t3,8(sp)
    sw  t1,12(sp)
    sw  a7,16(sp)
    sw  a6,20(sp)
    sw  a0,24(sp)
    sw  a1,28(sp)
    sw  a2,32(sp)
    sw  a3,36(sp)
    sw  a4,40(sp)
    sw  a5,44(sp)
    li  a3,7
    li  a2,50
    li  a1,10
    addi    a0,sp,8
    call    sumOf
    addi    s0,sp,8
    addi    s2,sp,48
    lui s1,%hi(.LC1)
.L2:
    lw  a1,0(s0)
    addi    a0,s1,%lo(.LC1)
    call    printf
    addi    s0,s0,4
    bne s0,s2,.L2
    lui a0,%hi(.LC2)
    addi    a0,a0,%lo(.LC2)
    call    scanf
    li  a0,0
```

```
    lw  ra,60(sp)
    lw  s0,56(sp)
    lw  s1,52(sp)
    lw  s2,48(sp)
    addi    sp,sp,64
    jr  ra
    .size   main, .-main
    .section    .rodata
    .align  2
    .set    .LANCHOR0,. + 0
.LC0:
    .word   0
    .word   1
    .word   2
    .word   3
    .word   4
    .word   5
    .word   6
    .word   7
    .word   8
    .word   9
    .section    .rodata.str1.4,"aMS",@progbits,1
    .align  2
.LC1:
    .string "%d "
.LC2:
    .string "1"
    .ident  "GCC: (SiFive GCC 8.3.0-2020.04.1) 8.3.0"

sumOf.s:

    .file   "sumOf.c"
    .option nopic
    .attribute arch, "rv32i2p0_c2p0"
    .attribute unaligned_access, 0
    .attribute stack_align, 16
    .text
    .align  1
    .globl  sumOf
    .type   sumOf, @function
sumOf:
    addi    sp,sp,-32
    sw  ra,28(sp)
    sw  s0,24(sp)
    sw  s1,20(sp)
    sw  s2,16(sp)
    sw  s3,12(sp)
    sw  s4,8(sp)
    ble a1,zero,.L2
    mv  s0,a0
    mv  s2,a1
    mv  s4,a2
```

```
   mv  s3,a3
   mv  a5,a0
   slli    a3,a1,2
   add a3,a3,a0
   li  s1,0
.L3:
   lw  a4,0(a5)
   add s1,s1,a4
   addi    a5,a5,4
   bne a5,a3,.L3
   mv  a1,s1
   lui a0,%hi(.LC0)
   addi    a0,a0,%lo(.LC0)
   call    printf
   bge s1,s4,.L1
   mv  a0,s0
   li  a4,0
.L6:
   lw  a5,0(a0)
   add a5,a5,s3
   sw  a5,0(a0)
   addi    a4,a4,1
   addi    a0,a0,4
   bgt s2,a4,.L6
.L1:
   lw  ra,28(sp)
   lw  s0,24(sp)
   lw  s1,20(sp)
   lw  s2,16(sp)
   lw  s3,12(sp)
   lw  s4,8(sp)
   addi    sp,sp,32
   jr  ra
.L2:
   li  a1,0
   lui a0,%hi(.LC0)
   addi    a0,a0,%lo(.LC0)
   call    printf
   j   .L1
   .size   sumOf, .-sumOf
   .section    .rodata.str1.4,"aMS",@progbits,1
   .align  2
.LC0:
   .string "%d\n"
   .ident  "GCC: (SiFive GCC 8.3.0-2020.04.1) 8.3.0"
```

4.3 Ассемблирование

Ассемблирование файлов main.s и sumOf.s выполняется по следующей команде:

```
riscv64-unknown-elf-gcc -march=rv32ic -mabi=ilp32 -O1 -c main.s -o main.o
riscv64-unknown-elf-gcc -march=rv32ic -mabi=ilp32 -O1 -c sumOf.s -o sumOf.o
```

Чтобы посмотреть содержимое main.o и sumOf.o необходимо вызвать комманду:

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

Результат ее выполнения - заголовки секций файла main.o:

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000090  00000000  00000000  00000034  2**1
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  000000c4  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000c4  2**0
                  ALLOC
  3 .rodata       00000028  00000000  00000000  000000c4  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .rodata.str1.4 00000006  00000000  00000000  000000ec  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  5 .comment      00000029  00000000  00000000  000000f2  2**0
                  CONTENTS, READONLY
  6 .riscv.attributes 00000021  00000000  00000000  0000011b  2**0
                  CONTENTS, READONLY
```

Заголовки секций файла sumOf.o:

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000007a  00000000  00000000  00000034  2**1
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  000000ae  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000ae  2**0
                  ALLOC
  3 .rodata.str1.4 00000004  00000000  00000000  000000b0  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .comment      00000029  00000000  00000000  000000b4  2**0
                  CONTENTS, READONLY
  5 .riscv.attributes 00000021  00000000  00000000  000000dd  2**0
                  CONTENTS, READONLY
```

В файлах “main.o” и “ sumOf.o” имеются следующие секции:
.text – секция кода, в которой содержатся коды инструкций (название секции обусловлено историческими причинами);
.data – секция инициализированных данных;
.bss – секция неинициализированных статических переменных;
.rodata – аналог .data для неизменяемых данных
.comment – секция данных о версиях размером 12 байт
.riscv.attributes – информация про RISC-V

Секция riscv.attributes обоих объектных файлов содержит одну и ту же информацию об используемой архитектуре команд RV32I.

Секция comment – секция данных о версиях – и для одного, и для другого файла содержит одни и те же значения – сведения о GCC версии.

Секции data объектных файлов – секции инициализированных данных – не содержат данных, размер секций равен нулю.

Секции bss объектных файлов – секции данных, инициализированных нулями – таким же образом пусты.

Теперь изучим таблицу символов файла main.o:

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

```
main.o:     file format elf32-littleriscv

SYMBOL TABLE:
00000000 l    df *ABS*  00000000 main.c
00000000 l    d  .text  00000000 .text
00000000 l    d  .data  00000000 .data
00000000 l    d  .bss   00000000 .bss
00000000 l    d  .rodata        00000000 .rodata
00000000 l       .rodata        00000000 .LANCHOR0
00000000 l    d  .rodata.str1.4 00000000 .rodata.str1.4
00000000 l       .rodata.str1.4 00000000 .LC1
00000004 l       .rodata.str1.4 00000000 .LC2
0000005e l       .text  00000000 .L2
00000000 l    d  .comment       00000000 .comment
00000000 l    d  .riscv.attributes      00000000 .riscv.attributes
00000000 g     F .text  00000090 main
00000000         *UND*  00000000 sumOf
00000000         *UND*  00000000 printf
00000000         *UND*  00000000 scanf
```

В таблице символов main.o имеется запись: символ “sumOf” типа *UND*. Эта запись означает, что символ “sumOf” использовался в ассемблерном коде, из которого был получен данный объектный файл, но не был определен; ассемблер сделал вывод о том, что символ должен быть определен где-то еще, и отразил это в таблице символов. То же самое относится и к символу “printf” и “scanf”.
Как и следовало ожидать, таблица содержит один глобальный (флаг “g”) символ типа «функция» (“F”) – символ “main”.

Теперь изучим таблицу символов файла sumOf.o:

```
sumOf.o:     file format elf32-littleriscv

SYMBOL TABLE:
00000000 l    df *ABS*  00000000 sumOf.c
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       .rodata.str1.4 00000000 .LC0
00000066 l       .text  00000000 .L2
00000024 l       .text  00000000 .L3
00000056 l       .text  00000000 .L1
00000048 l       .text  00000000 .L6
00000000 l    d  .comment       00000000 .comment
00000000 l    d  .riscv.attributes      00000000 .riscv.attributes
00000000 g     F .text  0000007a sumOf
00000000         *UND*  00000000 printf
```

Как и следовало ожидать, таблица содержит один глобальный (флаг “g”) символ типа «функция» (“F”) – символ “sumOf”.

Проанализируем таблицы перемещений:

```
riscv64-unknown-elf-objdump -r main.o sumOf.o
```

Для файла main.o:

```
main.o:     file format elf32-littleriscv

RELOCATION RECORDS FOR [.text]:
OFFSET   TYPE              VALUE
0000000a R_RISCV_HI20      .LANCHOR0
0000000a R_RISCV_RELAX     *ABS*
0000000e R_RISCV_LO12_I    .LANCHOR0
0000000e R_RISCV_RELAX     *ABS*
0000004c R_RISCV_CALL      sumOf
0000004c R_RISCV_RELAX     *ABS*
0000005a R_RISCV_HI20      .LC1
0000005a R_RISCV_RELAX     *ABS*
00000060 R_RISCV_LO12_I    .LC1
00000060 R_RISCV_RELAX     *ABS*
00000064 R_RISCV_CALL      printf
00000064 R_RISCV_RELAX     *ABS*
00000072 R_RISCV_HI20      .LC2
00000072 R_RISCV_RELAX     *ABS*
00000076 R_RISCV_LO12_I    .LC2
00000076 R_RISCV_RELAX     *ABS*
0000007a R_RISCV_CALL      scanf
0000007a R_RISCV_RELAX     *ABS*
0000006e R_RISCV_BRANCH    .L2
```

Для файла sumOf.o:

```
sumOf.o:     file format elf32-littleriscv

RELOCATION RECORDS FOR [.text]:
OFFSET   TYPE              VALUE
00000030 R_RISCV_HI20      .LC0
00000030 R_RISCV_RELAX     *ABS*
00000034 R_RISCV_LO12_I    .LC0
00000034 R_RISCV_RELAX     *ABS*
00000038 R_RISCV_CALL      printf
00000038 R_RISCV_RELAX     *ABS*
00000068 R_RISCV_HI20      .LC0
00000068 R_RISCV_RELAX     *ABS*
0000006c R_RISCV_LO12_I    .LC0
0000006c R_RISCV_RELAX     *ABS*
00000070 R_RISCV_CALL      printf
00000070 R_RISCV_RELAX     *ABS*
0000000e R_RISCV_BRANCH    .L2
0000002a R_RISCV_BRANCH    .L3
00000040 R_RISCV_BRANCH    .L1
00000052 R_RISCV_BRANCH    .L6
00000078 R_RISCV_RVC_JUMP  .L1
```

В файле main.o есть информация о необходимости замены для всех внешних функций, а в файле sumOf.o содержится информация о необходимости подстановки адресов для возвратов в циклах.

Проанализируем секции .text объектных файлов – секций кода, в которых содержатся коды инструкций:

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

Для файла main.o:

```
main.o:     file format elf32-littleriscv


Disassembly of section .text:

00000000 <main>:
   0:   7139                    c.addi16sp      sp,-64
   2:   de06                    c.swsp  ra,60(sp)
   4:   dc22                    c.swsp  s0,56(sp)
   6:   da26                    c.swsp  s1,52(sp)
   8:   d84a                    c.swsp  s2,48(sp)
   a:   000007b7                lui     a5,0x0
   e:   00078793                addi    a5,a5,0 # 0 <main>
  12:   0007ae03                lw      t3,0(a5)
  16:   0047a303                lw      t1,4(a5)
  1a:   0087a883                lw      a7,8(a5)
  1e:   00c7a803                lw      a6,12(a5)
  22:   4b88                    c.lw    a0,16(a5)
```

```
  24:   4bcc                    c.lw    a1,20(a5)
  26:   4f90                    c.lw    a2,24(a5)
  28:   4fd4                    c.lw    a3,28(a5)
  2a:   5398                    c.lw    a4,32(a5)
  2c:   53dc                    c.lw    a5,36(a5)
  2e:   c472                    c.swsp  t3,8(sp)
  30:   c61a                    c.swsp  t1,12(sp)
  32:   c846                    c.swsp  a7,16(sp)
  34:   ca42                    c.swsp  a6,20(sp)
  36:   cc2a                    c.swsp  a0,24(sp)
  38:   ce2e                    c.swsp  a1,28(sp)
  3a:   d032                    c.swsp  a2,32(sp)
  3c:   d236                    c.swsp  a3,36(sp)
  3e:   d43a                    c.swsp  a4,40(sp)
  40:   d63e                    c.swsp  a5,44(sp)
  42:   469d                    c.li    a3,7
  44:   03200613                addi    a2,zero,50
  48:   45a9                    c.li    a1,10
  4a:   0028                    c.addi4spn      a0,sp,8
  4c:   00000097                auipc   ra,0x0
  50:   000080e7                jalr    ra,0(ra) # 4c <main+0x4c>
  54:   0020                    c.addi4spn      s0,sp,8
  56:   03010913                addi    s2,sp,48
  5a:   000004b7                lui     s1,0x0

0000005e <.L2>:
  5e:   400c                    c.lw    a1,0(s0)
  60:   00048513                addi    a0,s1,0 # 0 <main>
  64:   00000097                auipc   ra,0x0
  68:   000080e7                jalr    ra,0(ra) # 64 <.L2+0x6>
  6c:   0411                    c.addi  s0,4
  6e:   ff2418e3                bne     s0,s2,5e <.L2>
  72:   00000537                lui     a0,0x0
  76:   00050513                addi    a0,a0,0 # 0 <main>
  7a:   00000097                auipc   ra,0x0
  7e:   000080e7                jalr    ra,0(ra) # 7a <.L2+0x1c>
  82:   4501                    c.li    a0,0
  84:   50f2                    c.lwsp  ra,60(sp)
  86:   5462                    c.lwsp  s0,56(sp)
  88:   54d2                    c.lwsp  s1,52(sp)
  8a:   5942                    c.lwsp  s2,48(sp)
  8c:   6121                    c.addi16sp      sp,64
  8e:   8082                    c.jr    ra
```

Для файла sumOf.o:

```
sumOf.o:     file format elf32-littleriscv


Disassembly of section .text:

00000000 <sumOf>:
   0:   1101                    c.addi  sp,-32
   2:   ce06                    c.swsp  ra,28(sp)
   4:   cc22                    c.swsp  s0,24(sp)
   6:   ca26                    c.swsp  s1,20(sp)
   8:   c84a                    c.swsp  s2,16(sp)
   a:   c64e                    c.swsp  s3,12(sp)
   c:   c452                    c.swsp  s4,8(sp)
   e:   04b05c63                bge     zero,a1,66 <.L2>
  12:   842a                    c.mv    s0,a0
  14:   892e                    c.mv    s2,a1
  16:   8a32                    c.mv    s4,a2
  18:   89b6                    c.mv    s3,a3
  1a:   87aa                    c.mv    a5,a0
  1c:   00259693                slli    a3,a1,0x2
  20:   96aa                    c.add   a3,a0
  22:   4481                    c.li    s1,0

00000024 <.L3>:
  24:   4398                    c.lw    a4,0(a5)
  26:   94ba                    c.add   s1,a4
  28:   0791                    c.addi  a5,4
  2a:   fed79de3                bne     a5,a3,24 <.L3>
  2e:   85a6                    c.mv    a1,s1
  30:   00000537                lui     a0,0x0
  34:   00050513                addi    a0,a0,0 # 0 <sumOf>
  38:   00000097                auipc   ra,0x0
  3c:   000080e7                jalr    ra,0(ra) # 38 <.L3+0x14>
  40:   0144db63                bge     s1,s4,56 <.L1>
  44:   8522                    c.mv    a0,s0
  46:   4701                    c.li    a4,0

00000048 <.L6>:
  48:   411c                    c.lw    a5,0(a0)
  4a:   97ce                    c.add   a5,s3
  4c:   c11c                    c.sw    a5,0(a0)
  4e:   0705                    c.addi  a4,1
  50:   0511                    c.addi  a0,4
  52:   ff274be3                blt     a4,s2,48 <.L6>
```

```
00000056 <.L1>:
  56:   40f2                    c.lwsp  ra,28(sp)
  58:   4462                    c.lwsp  s0,24(sp)
  5a:   44d2                    c.lwsp  s1,20(sp)
  5c:   4942                    c.lwsp  s2,16(sp)
  5e:   49b2                    c.lwsp  s3,12(sp)
  60:   4a22                    c.lwsp  s4,8(sp)
  62:   6105                    c.addi16sp      sp,32
  64:   8082                    c.jr    ra

00000066 <.L2>:
  66:   4581                    c.li    a1,0
  68:   00000537                lui     a0,0x0
  6c:   00050513                addi    a0,a0,0 # 0 <sumOf>
  70:   00000097                auipc   ra,0x0
  74:   000080e7                jalr    ra,0(ra) # 70 <.L2+0xa>
  78:   bff9                    c.j     56 <.L1>
```

Дизассемблированный код (текст программы на языке ассемблера) практически идентичен сгенерированному (за исключением псевдоинструкций).

4.4 Компоновка

Компоновка осуществляется следующей командой:

```
riscv64-unknown-elf-gcc -march=rv32ic -mabi=ilp32 main.o sumOf.o -o main
```

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

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

Нас интересует только небольшой фрагмент результирующего файла “main.ds”:

00010144 <**main**>:

```
  10144:        7139                    c.addi16sp      sp,-64
  10146:        de06                    c.swsp ra,60(sp)
  10148:        dc22                    c.swsp s0,56(sp)
  1014a:        da26                    c.swsp s1,52(sp)
  1014c:        d84a                    c.swsp s2,48(sp)
  1014e:        0002f7b7                lui     a5,0x2f
  10152:        f0078793                addi    a5,a5,-256 # 2ef00 <__clzsi2+0x4c>
  10156:        0007ae03                lw      t3,0(a5)
  1015a:        0047a303                lw      t1,4(a5)
  1015e:        0087a883                lw      a7,8(a5)
  10162:        00c7a803                lw      a6,12(a5)
  10166:        4b88                    c.lw    a0,16(a5)
  10168:        4bcc            c.lw    a1,20(a5)
  1016a:        4f90            c.lw    a2,24(a5)
  1016c:        4fd4            c.lw    a3,28(a5)
  1016e:        5398                    c.lw    a4,32(a5)
  10170:        53dc                    c.lw    a5,36(a5)
```

```
  10172:	c472                	c.swsp t3,8(sp)
  10174:	c61a            	c.swsp t1,12(sp)
  10176:	c846                	c.swsp a7,16(sp)
  10178:	ca42            	c.swsp a6,20(sp)
  1017a:	cc2a            	c.swsp a0,24(sp)
  1017c:	ce2e            	c.swsp a1,28(sp)
  1017e:	d032                	c.swsp a2,32(sp)
  10180:	d236                	c.swsp a3,36(sp)
  10182:	d43a                	c.swsp a4,40(sp)
  10184:	d63e                	c.swsp a5,44(sp)
  10186:	469d                	c.li	a3,7
  10188:	03200613            	addi	a2,zero,50
  1018c:	45a9                	c.li	a1,10
  1018e:	0028                	c.addi4spn	a0,sp,8
  10190:	280d                	c.jal	101c2 <sumOf>
  10192:	0020                	c.addi4spn	s0,sp,8
  10194:	03010913            	addi	s2,sp,48
  10198:	0002f4b7            	lui	s1,0x2f
  1019c:	400c                	c.lw	a1,0(s0)
  1019e:	f2848513            	addi	a0,s1,-216 # 2ef28 <__clzsi2+0x74>
  101a2:	24ed                	c.jal	1048c <printf>
  101a4:	0411                	c.addi	s0,4
  101a6:	ff241be3            	bne	s0,s2,1019c <main+0x58>
  101aa:	0002f537            	lui	a0,0x2f
  101ae:	f2c50513            	addi	a0,a0,-212 # 2ef2c <__clzsi2+0x78>
  101b2:	263d                	c.jal	104e0 <scanf>
  101b4:	4501                	c.li	a0,0
  101b6:	50f2            	c.lwsp	ra,60(sp)
  101b8:	5462                	c.lwsp	s0,56(sp)
  101ba:	54d2                	c.lwsp	s1,52(sp)
  101bc:	5942                	c.lwsp	s2,48(sp)
  101be:	6121                	c.addi16sp	sp,64
  101c0:	8082                	c.jr	ra

000101c2 <sumOf>:
  101c2:	1101                	c.addi	sp,-32
  101c4:	ce06            	c.swsp ra,28(sp)
  101c6:	cc22            	c.swsp s0,24(sp)
  101c8:	ca26            	c.swsp s1,20(sp)
  101ca:	c84a            	c.swsp s2,16(sp)
  101cc:	c64e            	c.swsp s3,12(sp)
  101ce:	c452                	c.swsp s4,8(sp)
  101d0:	04b05963            	bge	zero,a1,10222 <sumOf+0x60>
  101d4:	842a                	c.mv	s0,a0
  101d6:	892e                	c.mv	s2,a1
  101d8:	8a32                	c.mv	s4,a2
  101da:	89b6                	c.mv	s3,a3
  101dc:	87aa            	c.mv	a5,a0
  101de:	00259693            	slli	a3,a1,0x2
  101e2:	96aa            	c.add	a3,a0
  101e4:	4481                	c.li	s1,0
  101e6:	4398                	c.lw	a4,0(a5)
```

```
101e8:	94ba                	c.add	s1,a4
101ea:	0791                	c.addi	a5,4
101ec:	fed79de3            	bne	a5,a3,101e6 <sumOf+0x24>
101f0:	85a6                	c.mv	a1,s1
101f2:	0002f537            	lui	a0,0x2f
101f6:	f3050513            	addi	a0,a0,-208 # 2ef30 <__clzsi2+0x7c>
101fa:	2c49                	c.jal	1048c <printf>
101fc:	0144db63            	bge	s1,s4,10212 <sumOf+0x50>
10200:	8522                	c.mv	a0,s0
10202:	4701                	c.li	a4,0
10204:	411c                	c.lw	a5,0(a0)
10206:	97ce                	c.add	a5,s3
10208:	c11c                	c.sw	a5,0(a0)
1020a:	0705                	c.addi	a4,1
1020c:	0511                	c.addi	a0,4
1020e:	ff274be3            	blt	a4,s2,10204 <sumOf+0x42>
10212:	40f2                	c.lwsp	ra,28(sp)
10214:	4462                	c.lwsp	s0,24(sp)
10216:	44d2                	c.lwsp	s1,20(sp)
10218:	4942                	c.lwsp	s2,16(sp)
1021a:	49b2                	c.lwsp	s3,12(sp)
1021c:	4a22                	c.lwsp	s4,8(sp)
1021e:	6105                	c.addi16sp	sp,32
10220:	8082                	c.jr	ra
10222:	4581                	c.li	a1,0
10224:	0002f537            	lui	a0,0x2f
10228:	f3050513            	addi	a0,a0,-208 # 2ef30 <__clzsi2+0x7c>
1022c:	2485                	c.jal	1048c <printf>
1022e:	b7d5                	c.j	10212 <sumOf+0x50>
```

5. **Создание статической библиотеки:**

Статическая библиотека является архивом (набором, коллекцией) объектных файлов.

Поместим sumOf.o в статическую библиотеку lib:

```
riscv64-unknown-elf-ar -rsc lib.a sumOf.o
```

Параметры:
-r – заменить старые файлы с такими названиями, если они уже есть в архиве;
-s – записать «index» в архив. Index – это список всех символов, объявленных во включенных в архив объектных файлах;
-c – создать архив.
Теперь мы можем вызывать библиотку:

```
riscv64-unknown-elf-gcc -march=rv32ic -mabi=ilp32 -O1 --save-temps main.c lib.a -o a.out
```

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

sumOf.o:
00000056 t .L1
00000066 t .L2
00000024 t .L3
00000048 t .L6
00000000 r .LC0
         U printf
00000000 T sumOf
```

В выводе утилиты -nm кодом T обозначаются символы, определенные в соответствующем объектном файле. Символ функции sumOf является основным символом, определяемым в этом объектном файле, остальные символы определяют локальные метки для этого файла.

Создадим make-файлы:
Это такие файлы, в которых заранее прописаны инструкции, необходимы, например, для создания библиотеки, сборки программы и т.д.
Make-файлы, произведут создание библиотеки и сборку программы:

Makefile1:

```
lib.a: sumOf.o sumOf.h
     riscv64-unknown-elf-ar -rsc lib.a sumOf.o

sumOf.o: sumOf.c
     riscv64-unknown-elf-gcc.exe -march=rv32i -mabi=ilp32 -O1 -c sumOf.c -o sumOf.o
```

Makefile2:

```
all:
     mingw32-make -f Makefile-1
     riscv64-unknown-elf-gcc -march=rv32ic -mabi=ilp32 -O1 --save-temps main.c lib.a -o a.out
     del *.o, *.i, *.s
```

Makefile1 создает библиотеку с sumOf.o, а Makefile2 вызывает Makefile1, а затем создает файл a.out с ответом, далее следует удаление файлов, созданных при работе.
Затем мы можем вызвать make-file:

```
mingw32-make -f Makefile-2
```

После вызова этого make-file сначала будет создан обьектный файл sumOf.o, затем он будет добавлен в библиотку и будет создан файл a.out.