

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

Отчёт по лабораторной работе № 1

Дисциплина: Низкоуровневое программирование

Тема: программирование для EDSAC

Вариант: 3

Выполнила студентка гр. 3530901/00002 _____ С.Е.
Бельская

(подпись)

Принял преподаватель _____ Д.С.
Степанов

(подпись)

« ____ » _____ 2021 г.

Санкт-Петербург

2021

Задача

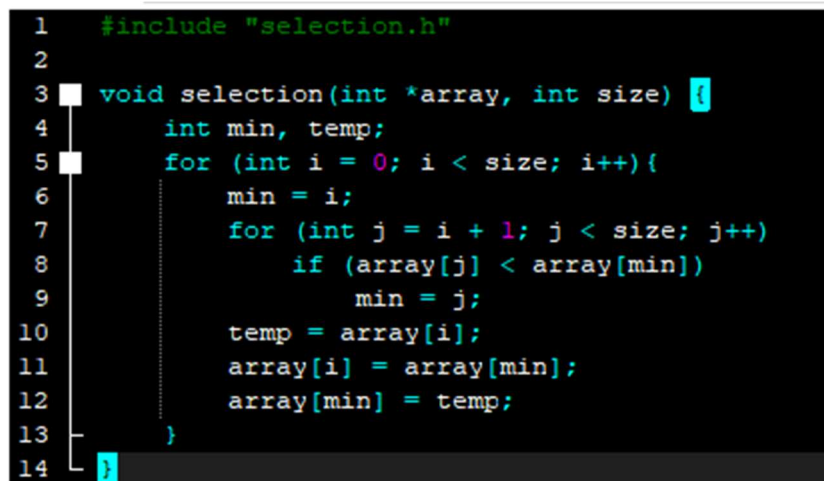
1. Изучить методические материалы, опубликованные на сайте курса.
2. Установить пакет средств разработки “SiFive GNU Embedded Toolchain” для RISC-V.
3. На языке C разработать функцию, реализующую определенную вариантом задания функциональность. Поместить определение функции в отдельный исходный файл,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

Алгоритм сортировки на языке C++

```
#include <iostream>
#include <cstdlib>
using namespace std;

int main ()
{
    int length;
    cin >> length;
    int array[length];
    for (int i = 0; i < length - 1; i++)
        cin >> array[i];
    int temp, min;
    for (int out; out < length - 2; out++)
    {
        min = out;
        for (int in = out + 1; in < length - 1; in++)
            if (array[in] < array[min]) min = in;
        temp = array[out];
        array[out] = array[min];
        array[min] = temp;
    };
    for (int i = 0; i < length - 1; i++)
        cout << array[i] << ' ';
    return 0;
}
```

Текст программы



```
1  #include "selection.h"
2
3  void selection(int *array, int size) {
4      int min, temp;
5      for (int i = 0; i < size; i++){
6          min = i;
7          for (int j = i + 1; j < size; j++)
8              if (array[j] < array[min])
9                  min = j;
10         temp = array[i];
11         array[i] = array[min];
12         array[min] = temp;
13     }
14 }
```

Рис. 1 Основной файл selection.c

```

1  #ifndef SELECTION_H
2  #define SELECTION_H
3
4  extern void selection (int *array, int size);
5
6  #endif

```

Рис. 2 Заголовочный файл selection.h

```

1  #include <stdio.h>
2  #include "selection.h"
3
4  #define size 10
5  int main(void) {
6      int array[size];
7      for (int i = 0; i < size; i++)
8          scanf("%i", array[i]);
9
10     selection(array, size);
11
12     for (int i = 0; i < size; i++)
13         printf("%i ", &array[i]);
14     return 0;
15 }

```

Рис. 3 Тестовая программа main.c

Сборка программы «по шагам»

1. Препроцессирование

Первым шагом является препроцессирование файлов исходного текста (файлов “main.c” и “selection.c”), результаты записываются в файлы “main.i” и “selection.i”:

```
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 selection.c -o
selection.i
```

Параметры:

-march=rv32ic -mabi=ilp32 – целевым является процессор с базовой архитектурой системы команд RV32I;

-O1 – выполнять простые оптимизации генерируемого кода (мы используем эту опцию в примерах, потому что обычно генерируемый код получается более простым);

-E – остановить процесс сборки после препроцессирования;

-o – путь к выходному файлу.

```
# 2 "main.c" 2
# 1 "selection.h" 1

# 4 "selection.h"
extern void selection (int *array, int size);
# 3 "main.c" 2

int main(void) {
    int array[10];
    for (int i = 0; i < 10; i++)
        scanf("%i", array[i]);

    selection(array, 10);

    for (int i = 0; i < 10; i++)
        printf("%i ", &array[i]);
    return 0;
}
```

Рис. 3 Выход препроцессирования (фрагмент из файла main.i)

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

extern void selection (int *array, int size);
# 2 "selection.c" 2

void selection(int *array, int size) {
    int min, temp;
    for (int i = 0; i < size; i++){
        min = i;
        for (int j = i + 1; j < size; j++){
            if (array[j] < array[min])
                min = j;
        }
        temp = array[i];
        array[i] = array[min];
        array[min] = temp;
    }
}
```

Рис. 4 Выход препроцессирования (фрагмент из файла selection.i)

Результат препроцессирования содержится в файле selection.i и main.i.

По причине того, что исходные файлы содержат заголовочные файлы нескольких стандартных библиотек C, результат препроцессирования

отличается от исходных файлов и имеет достаточно много добавочных строк, среди которых и исходные программы. Также можно заметить, что препроцессор включил содержимое файла `selection.h`.

Появившиеся нестандартные директивы, начинающиеся с символа “#”, используются для передачи информации об исходном тексте из препроцессора в компилятор.

2. Компиляция

Далее необходимо выполнить компиляцию файлов “`main.i`” и “`sieve.i`”, сохранив результат – сгенерированный код на языке ассемблера – в файлы “`main.s`” и “`sieve.s`”.

```
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  
sieve.i -o sieve.s
```

Параметры:

-S – остановить процесс сборки после компиляции, не запуская ассемблер;
-fpreprocessed – выполняется компиляция файла, уже отработанного процесса;

Все остальные параметры из прошлого пункта означают тоже самое.

```

| .file "main.c"
  .option nopic
  .attribute arch, "rv32i2p0"
  .attribute unaligned_access, 0
  .attribute stack_align, 16
  .text
  .align      2
  .globl      main
  .type main, @function
main:
  addi sp, sp, -80
  sw    ra, 76(sp)
  sw    s0, 72(sp)
  sw    s1, 68(sp)
  sw    s2, 64(sp)
  sw    s3, 60(sp)
  addi s0, sp, 8
  addi s2, sp, 48
  mv    s1, s0
  lui   s3, %hi(.LC0)
.L2:
  lw    a1, 0(s1)
  addi a0, s3, %lo(.LC0)
  call  scanf
  addi s1, s1, 4
  bne   s1, s2, .L2
  li    a1, 10
  addi a0, sp, 8
  call  selection
  lui   s1, %hi(.LC1)
.L3:
  mv    a1, s0
  addi a0, s1, %lo(.LC1)
  call  printf
  addi s0, s0, 4
  bne   s0, s2, .L3
  li    a0, 0
  lw    ra, 76(sp)
  lw    s0, 72(sp)
  lw    s1, 68(sp)
  lw    s2, 64(sp)
  lw    s3, 60(sp)
  addi sp, sp, 80
  jr    ra
  .size main, .-main
  .section .rodata.str1.4, "aMS", @progbits, 1
  .align      2
.LC0:
  .string    "%i"
  .zero 1
.LC1:
  .string    "%i "
  .ident     "GCC: (SiFive GCC 8.3.0-2020.04.1) 8.3.0"

```

Рис. 5 Выход компилятора (файл main.s)

```

| .file "selection.c"
  .option nopic
  .attribute arch, "rv32i2p0"
  .attribute unaligned_access, 0
  .attribute stack_align, 16
  .text
  .align      2
  .globl      selection
  .type selection, @function
selection:
  ble a1,zero,.L1
  mv a7,a0
  li a2,0
  j .L6
.L3:
  addi a4,a4,1
  addi a3,a3,4
  beq a1,a4,.L8
.L4:
  slli a5,a2,2
  add a5,a0,a5
  lw a6,4(a3)
  lw a5,0(a5)
  bge a6,a5,.L3
  mv a2,a4
  j .L3
.L8:
  lw a5,0(a7)
  slli a2,a2,2
  add a2,a0,a2
  lw a4,0(a2)
  sw a4,0(a7)
  sw a5,0(a2)
  addi a7,a7,4
  mv a2,t1
.L6:
  addi t1,a2,1
  beq a1,t1,.L1
  mv a3,a7
  mv a4,t1
  j .L4
.L1:
  ret
  .size selection, .-selection
  .ident "GCC: (SiFive GCC 8.3.0-2020.04.1) 8.3.0"

```

Рис. 6 Выход компилятора (файл selection.s)

Наибольший интерес представляет файл main.s, так как в нем можно заметить обращение к подпрограмме selection (значение регистра ra, содержащее адрес возврата из main, сохраняется на время вызова в стеке).

3. Ассемблирование

Ассемблирование файлов “main.s” и “selection.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 selection.s -o  
selection.o
```

Параметры:

-c – остановить процесс сборки после ассемблирования.

На выходе получили объектные файлы main.o и selection.o. Они содержат коды инструкций, таблицу символов и таблицу перемещений (relocations). В отличие от ранее рассмотренных файлов, объектный файл не является текстовым, для изучения его содержимого используем утилиту objdump, отображающую содержимое бинарных файлов в текстовом виде.

Как известно, содержательная часть объектного файла разбита на «разделы», называемые обычно секциями (section). Следующая команда обеспечивает отображение заголовков секций файлов “main.o” и “selection.o”:

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

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

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

Sections:
Idx Name          Size      VMA           LMA           File off  Algn
  0 .text          0000008c  00000000  00000000  00000034  2**2
    CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data           00000000  00000000  00000000  000000c0  2**0
    CONTENTS, ALLOC, LOAD, DATA
  2 .bss            00000000  00000000  00000000  000000c0  2**0
    ALLOC
  3 .rodata.str1.4  00000008  00000000  00000000  000000c0  2**2
    CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .comment         00000029  00000000  00000000  000000c8  2**0
    CONTENTS, READONLY
  5 .riscv.attributes 0000001c  00000000  00000000  000000f1  2**0
    CONTENTS, READONLY
```

Рис. 7 Заголовки секций файла main.o

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

| Sections: | | | | | | |
|-----------|--|----------|----------|----------|----------|------|
| Idx | Name | Size | VMA | LMA | File off | Algn |
| 0 | .text | 00000070 | 00000000 | 00000000 | 00000034 | 2**2 |
| | CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE | | | | | |
| 1 | .data | 00000000 | 00000000 | 00000000 | 000000a4 | 2**0 |
| | CONTENTS, ALLOC, LOAD, DATA | | | | | |
| 2 | .bss | 00000000 | 00000000 | 00000000 | 000000a4 | 2**0 |
| | ALLOC | | | | | |
| 3 | .comment | 00000029 | 00000000 | 00000000 | 000000a4 | 2**0 |
| | CONTENTS, READONLY | | | | | |
| 4 | .riscv.attributes | 0000001c | 00000000 | 00000000 | 000000cd | 2**0 |
| | CONTENTS, READONLY | | | | | |

Рис. 8 Заголовки секций файла selection.o

В файлах “main.o” и “selection.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 (название секции также обусловлено историческими причинами);
- .rodata – аналог .data для неизменяемых данных
- .comment – секция данных о версиях размером 12 байт
- .riscv.attributes – информация про RISC-V

Изучим содержимое таблиц символов объектных файлов “main.o” и “selection.o”:

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

```
riscv64-unknown-elf-objdump -t selection.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.str1.4 00000000 .rodata.str1.4
00000000 l      .rodata.str1.4 00000000 .LC0
00000004 l      .rodata.str1.4 00000000 .LC1
00000028 l      .text  00000000 .L2
00000054 l      .text  00000000 .L3
00000000 l      d  .comment 00000000 .comment
00000000 l      d  .riscv.attributes 00000000 .riscv.attributes
00000000 g      F  .text  0000008c main
00000000      *UND*  00000000 scanf
00000000      *UND*  00000000 selection
00000000      *UND*  00000000 printf

```

Рис. 9 Таблица символов файла main.o

```

selection.o:      file format elf32-littleriscv

SYMBOL TABLE:
00000000 l      df *ABS*  00000000 selection.c
00000000 l      d  .text  00000000 .text
00000000 l      d  .data  00000000 .data
00000000 l      d  .bss   00000000 .bss
0000006c l      .text  00000000 .L1
00000058 l      .text  00000000 .L6
00000038 l      .text  00000000 .L8
00000010 l      .text  00000000 .L3
0000001c l      .text  00000000 .L4
00000000 l      d  .comment 00000000 .comment
00000000 l      d  .riscv.attributes 00000000 .riscv.attributes
00000000 g      F  .text  00000070 selection

```

Рис. 10 Таблица символов файла selection.o

В таблице символов main.o имеется запись: символ “selection” типа *UND*. Эта запись означает, что символ “selection” 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 “main.o” и “selection.o”:

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

```

main.o:      file format elf32-littleriscv

Contents of section .text:
0000 130101fb 23261104 23248104 23229104  ....#&...#$.#"...
0010 23202105 232e3103 13048100 13090103  # !.#.1.....
0020 93040400 b7090000 83a50400 13850900  .....
0030 97000000 e7800000 93844400 e39624ff  ....D...$.
0040 9305a000 13058100 97000000 e7800000  .....
0050 b7040000 93050400 13850400 97000000  .....
0060 e7800000 13044400 e31624ff 13050000  ....D...$.....
0070 8320c104 03248104 83244104 03290104  . ...$...$A..)..
0080 8329c103 13010105 67800000  ....).).....g...

selection.o:  file format elf32-littleriscv

Contents of section .text:
0000 6356b006 93080500 13060000 6f00c004  cV.....o...
0010 13071700 93864600 6380e502 93172600  .....F.c.....&.
0020 b307f500 03a84600 83a70700 e352f8fe  .....F.....R..
0030 13060700 6ff0dfffd 83a70800 13162600  ....o.....&.
0040 3306c500 03270600 23a0e800 2320f600  3....'...#...# ..
0050 93884800 13060300 13031600 63886500  ..H.....c.e.
0060 93860800 13070300 6ff05ffb 67800000  ....o._.g...

```

Рис. 11 Вывод утилиты

Процедура декодирования кодов инструкций является «механической» (иначе как бы ее реализовывало техническое устройство – процессор), следовательно, разумно поручить ее выполнение ЭВМ:

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

Опция “-d” инициирует процесс дизассемблирования (disassemble), опция “-M no-aliases” требует использовать в выводе только инструкции системы команд (но не псевдоинструкции ассемблера).


```

main.o:      file format elf32-littleriscv

Disassembly of section .text:

00000000 <main>:
   0:  fb010113      addi    sp,sp,-80
   4:  04112623      sw      ra,76(sp)
   8:  04812423      sw      s0,72(sp)
  c:  04912223      sw      s1,68(sp)
 10:  05212023      sw      s2,64(sp)
 14:  03312e23      sw      s3,60(sp)
 18:  00810413      addi    s0,sp,8
 1c:  03010913      addi    s2,sp,48
 20:  00040493      addi    s1,s0,0
 24:  000009b7      lui     s3,0x0

00000028 <.L2>:
 28:  0004a583      lw      a1,0(s1)
 2c:  00098513      addi    a0,s3,0 # 0 <main>
 30:  00000097      auipc   ra,0x0
 34:  000080e7      jalr    ra,0(ra) # 30 <.L2+0x8>
 38:  00448493      addi    s1,s1,4
 3c:  ff2496e3      bne     s1,s2,28 <.L2>
 40:  00a00593      addi    a1,zero,10
 44:  00810513      addi    a0,sp,8
 48:  00000097      auipc   ra,0x0
 4c:  000080e7      jalr    ra,0(ra) # 48 <.L2+0x20>
 50:  000004b7      lui     s1,0x0

00000054 <.L3>:
 54:  00040593      addi    a1,s0,0
 58:  00048513      addi    a0,s1,0 # 0 <main>
 5c:  00000097      auipc   ra,0x0
 60:  000080e7      jalr    ra,0(ra) # 5c <.L3+0x8>
 64:  00440413      addi    s0,s0,4
 68:  ff2416e3      bne     s0,s2,54 <.L3>
 6c:  00000513      addi    a0,zero,0
 70:  04c12083      lw      ra,76(sp)
 74:  04812403      lw      s0,72(sp)
 78:  04412483      lw      s1,68(sp)
 7c:  04012903      lw      s2,64(sp)
 80:  03c12983      lw      s3,60(sp)
 84:  05010113      addi    sp,sp,80
 88:  00008067      jalr    zero,0(ra)

```

Рис. 12 Вывод утилиты для файла main.o

```

selection.o:      file format elf32-littleriscv

Disassembly of section .text:

00000000 <selection>:
   0:  06b05663          bge     zero,a1,6c <.L1>
   4:  00050893          addi    a7,a0,0
   8:  00000613          addi    a2,zero,0
  c:  04c0006f          jal     zero,58 <.L6>

00000010 <.L3>:
  10:  00170713          addi    a4,a4,1
  14:  00468693          addi    a3,a3,4
  18:  02e58063          beq     a1,a4,38 <.L8>

0000001c <.L4>:
  1c:  00261793          slli    a5,a2,0x2
  20:  00f507b3          add     a5,a0,a5
  24:  0046a803          lw      a6,4(a3)
  28:  0007a783          lw      a5,0(a5)
  2c:  fef852e3          bge     a6,a5,10 <.L3>
  30:  00070613          addi    a2,a4,0
  34:  fddff06f          jal     zero,10 <.L3>

00000038 <.L8>:
  38:  0008a783          lw      a5,0(a7)
  3c:  00261613          slli    a2,a2,0x2
  40:  00c50633          add     a2,a0,a2
  44:  00062703          lw      a4,0(a2)
  48:  00e8a023          sw      a4,0(a7)
  4c:  00f62023          sw      a5,0(a2)
  50:  00488893          addi    a7,a7,4
  54:  00030613          addi    a2,t1,0

00000058 <.L6>:
  58:  00160313          addi    t1,a2,1
  5c:  00658863          beq     a1,t1,6c <.L1>
  60:  00088693          addi    a3,a7,0
  64:  00030713          addi    a4,t1,0
  68:  fb5ff06f          jal     zero,1c <.L4>

0000006c <.L1>:
  6c:  00008067          jalr    zero,0(ra)

```

Рис. 13 Вывод утилиты для файла selection.o

Результат дизассемблирования “selection.o” интереса не представляет, в отличие от результата дизассемблирования “main.o”: сравнивая его с “main.s”, легко понять, что псевдоинструкция вызова подпрограммы “selection”, транслировалась ассемблером в следующую пару инструкций:

```
48: 00000097      auipc    ra,0x0
4c: 000080e7      jalr     ra,0(ra) # 48 <.L2+0x20>
```

Результатом выполнения этой пары инструкций станет переход на адрес 30, то есть заикливанием. Ассемблер не имел возможности определить целевой адрес, поэтому не мог сформировать корректную инструкцию (пару инструкций) передачи управления. В результате была сформирована пара инструкций с некорректными (нулевыми) 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:

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

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

RELOCATION RECORDS FOR [.text]:
OFFSET      TYPE          VALUE
00000024 R_RISCV_HI20      .LC0
00000024 R_RISCV_RELAX     *ABS*
0000002c R_RISCV_LO12_I    .LC0
0000002c R_RISCV_RELAX     *ABS*
00000030 R_RISCV_CALL      scanf
00000030 R_RISCV_RELAX     *ABS*
00000048 R_RISCV_CALL      selection
00000048 R_RISCV_RELAX     *ABS*
00000050 R_RISCV_HI20      .LC1
00000050 R_RISCV_RELAX     *ABS*
00000058 R_RISCV_LO12_I    .LC1
00000058 R_RISCV_RELAX     *ABS*
0000005c R_RISCV_CALL      printf
0000005c R_RISCV_RELAX     *ABS*
0000003c R_RISCV_BRANCH    .L2
00000068 R_RISCV_BRANCH    .L3

selection.o:  file format elf32-littleriscv

RELOCATION RECORDS FOR [.text]:
OFFSET      TYPE          VALUE
00000000 R_RISCV_BRANCH    .L1
0000000c R_RISCV_JAL       .L6
00000018 R_RISCV_BRANCH    .L8
0000002c R_RISCV_BRANCH    .L3
00000034 R_RISCV_JAL       .L3
0000005c R_RISCV_BRANCH    .L1
00000068 R_RISCV_JAL       .L4
```

Рис. 14 Вывод утилиты

В файле “main.o” имеется две записи, относящиеся к адресу 48 (как мы видели выше, по этому адресу в “main.o” находится первая инструкция пары `auipc+jalr`). Дизассемблирование и вывод таблицы перемещений можно совместить:

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


```

main.o:      file format elf32-littleriscv

Disassembly of section .text:

00000000 <main>:
  0:  fb010113      addi    sp,sp,-80
  4:  04112623      sw      ra,76(sp)
  8:  04812423      sw      s0,72(sp)
 c:  04912223      sw      s1,68(sp)
10:  05212023      sw      s2,64(sp)
14:  03312e23      sw      s3,60(sp)
18:  00810413      addi    s0,sp,8
1c:  03010913      addi    s2,sp,48
20:  00040493      addi    s1,s0,0
24:  000009b7      lui     s3,0x0
                        24: R_RISCV_HI20      .LC0
                        24: R_RISCV_RELAX      *ABS*

00000028 <.L2>:
28:  0004a583      lw      a1,0(s1)
2c:  00098513      addi    a0,s3,0 # 0 <main>
                        2c: R_RISCV_LO12_I      .LC0
                        2c: R_RISCV_RELAX      *ABS*
30:  00000097      auipc   ra,0x0
                        30: R_RISCV_CALL      scanf
                        30: R_RISCV_RELAX      *ABS*
34:  000080e7      jalr    ra,0(ra) # 30 <.L2+0x8>
38:  00448493      addi    s1,s1,4
3c:  ff2496e3      bne     s1,s2,28 <.L2>
                        3c: R_RISCV_BRANCH      .L2
40:  00a00593      addi    a1,zero,10
44:  00810513      addi    a0,sp,8
48:  00000097      auipc   ra,0x0
                        48: R_RISCV_CALL      selection
                        48: R_RISCV_RELAX      *ABS*
4c:  000080e7      jalr    ra,0(ra) # 48 <.L2+0x20>
50:  000004b7      lui     s1,0x0
                        50: R_RISCV_HI20      .LC1
                        50: R_RISCV_RELAX      *ABS*

00000054 <.L3>:
54:  00040593      addi    a1,s0,0
58:  00048513      addi    a0,s1,0 # 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 <.L3+0x8>
64:  00440413      addi    s0,s0,4
68:  ff2416e3      bne     s0,s2,54 <.L3>
                        68: R_RISCV_BRANCH      .L3
6c:  00000513      addi    a0,zero,0
70:  04c12083      lw      ra,76(sp)
74:  04812403      lw      s0,72(sp)
78:  04412483      lw      s1,68(sp)
7c:  04012903      lw      s2,64(sp)
80:  03c12983      lw      s3,60(sp)
84:  05010113      addi    sp,sp,80
88:  00008067      jalr    zero,0(ra)

```

Рис. 15 Вывод утилиты

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. Так, в таблице перемещений указано, что по адресу 48 следует исправить пару инструкций (тип перемещения “R_RISCV_CALL”) так, чтобы результат соответствовал вызову подпрограммы “selection”. Типы перемещений специфичны для каждой архитектуры системы команд и обычно определены в ABI (Application Binary Interface).

Следующая запись таблицы перемещений специфична для средств разработки RISC-V. Записи типа “R_RISCV_RELAX” заносятся в таблицу перемещений в дополнение к записям типа “R_RISCV_CALL” (и некоторым другим) и сообщают компоновщику, что пара инструкций, обеспечивающих вызов подпрограммы, может быть оптимизирована.

4. Компоновка

Компоновка программы выполняется по следующей команде:

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

Результатом является исполняемый файл “main” (имя выходного файла указано явно опцией “-o”, если бы она отсутствовала, использовалось имя файла по умолчанию “a.out”).

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

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

```

00010144 <main>:
    10144: fb010113      addi    sp,sp,-80
    10148: 04112623      sw      ra,76(sp)
    1014c: 04812423      sw      s0,72(sp)
    10150: 04912223      sw      s1,68(sp)
    10154: 05212023      sw      s2,64(sp)
    10158: 03312e23      sw      s3,60(sp)
    1015c: 00810413      addi    s0,sp,8
    10160: 03010913      addi    s2,sp,48
    10164: 00040493      addi    s1,s0,0
    10168: 0002f9b7      lui     s3,0x2f
    1016c: 0004a583      lw      a1,0(s1)
    10170: f0898513      addi    a0,s3,-248 # 2ef08
<__clzsi2+0x50>
    10174: 370000ef      jal     ra,104e4 <scanf>
    10178: 00448493      addi    s1,s1,4
    1017c: ff2498e3      bne     s1,s2,1016c <main+0x28>
    10180: 00a00593      addi    a1,zero,10
    10184: 00810513      addi    a0,sp,8
    10188: 03c000ef      jal     ra,101c4 <selection>
    1018c: 0002f4b7      lui     s1,0x2f
    10190: 00040593      addi    a1,s0,0
    10194: f0c48513      addi    a0,s1,-244 # 2ef0c
<__clzsi2+0x54>
    10198: 2f8000ef      jal     ra,10490 <printf>
    1019c: 00440413      addi    s0,s0,4
    101a0: ff2418e3      bne     s0,s2,10190 <main+0x4c>
    101a4: 00000513      addi    a0,zero,0
    101a8: 04c12083      lw      ra,76(sp)
    101ac: 04812403      lw      s0,72(sp)
    101b0: 04412483      lw      s1,68(sp)
    101b4: 04012903      lw      s2,64(sp)
    101b8: 03c12983      lw      s3,60(sp)
    101bc: 05010113      addi    sp,sp,80
    101c0: 00008067      jalr    zero,0(ra)

000101c4 <selection>:
    101c4: 06b05663      bge     zero,a1,10230
<selection+0x6c>
    101c8: 00050893      addi    a7,a0,0

```

Рис. 16 Фрагмент результирующего файла main.ds

Прежде всего можно видеть, что в результат компоновки попало содержимое обоих объектных файлов – “main.o” и “selection.o”.

Инструкции подпрограммы “selection” начинаются с адреса 101c4₁₆, и пара инструкций auipc+jalr, вызывающих подпрограмму “selection” соответствующим образом откорректированы.

В рассматриваемом нами примере точка вызова подпрограммы “selection” и сама подпрограмма находятся очень близко – их разделяет всего

76 байт. Учитывая это, в данном случае нет никакой необходимости использовать пару инструкций `auipc+jalr`, достаточно одной инструкции `jal`, 54-разрядный непосредственный операнд которой позволяет задавать переход в пределах ± 1 МиБ относительно адреса инструкции (значения `pc` в момент ее выполнения).

В общем случае целевой адрес перехода может отличаться от адреса инструкции перехода более чем на 1 МиБ, и поскольку ассемблер не имеет информации о целевом адресе перехода, при использовании псевдоинструкции `call` формируется пара инструкций `auipc+jalr`, использование которой позволяет выполнять переход в пределах ± 2 ГиБ (в любую точку, в случае 32-разрядного адресного пространства).

В отличие от ассемблера, компоновщик имеет всю необходимую информацию об адресах, и может принять решение о возможности использования инструкции `jal` вместо пары `auipc+jalr`. Такая оптимизация реализуется компоновщиком для архитектуры RISC-V и называется “linker relaxation”. Каждая пара инструкций, которая может быть оптимизирована, помечается ассемблером записью о перемещении типа “`R_RISCV_RELAX`”, пример которой мы уже видели.

Создание статистической библиотеки

Статическая библиотека (static library) является, по сути, архивом (набором, коллекцией) объектных файлов.

Поместим `selection.o` в статическую библиотеку `lib`:

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

Параметры:

-r – заменить старые файлы с такими названиями (`selection.o`), если они уже есть в архиве;

-s – записать «index» в архив. Index – это список всех символов, объявленных во включенных в архив объектных файлах, и его присутствие ускоряет линковку;

-c – создать архив, если его еще не было.

Используем статическую библиотеку для сборки программ, для этого напишем make-файл. Makefile – это набор инструкций для программы make, которая помогает собирать программу из файлов в один вызов make.

Общая структура makefile'ов:

Target: dependencies

Action

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

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

lib.a: selection.o selection.h
    mingw32-gcc-ar.exe -rsc lib.a selection.o

output: main.o lib.a
    mingw32-gcc-9.2.0.exe main.o lib.a -o output

clean:
    rm *.o *.a *.exe
```

Рис. 17 Make file

Что происходит в Makefile:

1. Создаём объектный файл main.o из исходного main.c
2. Создаём объектный файл selection.o из исходного selection.c
3. Архивируем объектный файл selection.o (создаём статическую библиотеку lib.a)
4. Компонуем статическую библиотеку lib.a с объектным файлом main.o и получаем исполняемый файл output

Вывод

В ходе выполнения данной лабораторной работы была разработана функция на языке C, реализующая заданную вариантом задания функциональность (сортировка методом выбора). Определение функции было помещено в отдельный исходный файл, также был оформлен заголовочный файл. Была разработана тестовая программа на языке C.

Затем была выполнена сборка программы «по шагам». Проанализированы выход препроцессора и компилятора. Проанализированы состав и содержимое секций, таблицы символов, таблицы перемещений и отладочная информация, содержащаяся в объектных файлах и исполняемом файле.

Разработанная функция была выделена в статическую библиотеку. Разработаны make-файлы для сборки библиотеки и использующей ее тестовой программы. Проанализированы ход сборки библиотеки и программы, созданные файлы зависимостей.