

Санкт-Петербургский политехнический университет Петра Великого
Кафедра компьютерных систем и программных технологий

Отчёт по лабораторной работе №4

Дисциплина: Низкоуровневое программирование

Тема: Раздельная компиляция

Выполнил студент гр. 3530901/10005

(подпись) Воронов И. В.

Преподаватель

(подпись) Корнев Д. А.

“ ” _____ 2022 г.

Санкт-Петербург

2022

1. Формулировка задачи

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 исполняемом 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.

2. Вариант задания

Расчет заданного члена ряда Фибоначчи.

3. Сборка по шагам

3.1. Исходный код

```
C main.c > ...
1  #include <stdio.h>
2  #include "fib.h"
3  int n = 10;
4
5  int main(void) {
6      int result = fib(n);
7
8      printf("%d", result);
9      return 0;
10 }
```

main.c

```
C fib.c > fib(int)
1  // fib.c
2  int fib(int n) {
3      int first = 0;
4      int second = 1;
5      for(int i = 0; i < n; i++) {
6          second += first;
7          first = second - first;
8      }
9      return first;
10 }
```

fib.c

```
C fib.h > ...
1  int fib(int n);
```

fib.h

В `main.c` содержится основная часть программы, в которой задаётся исходный номер числа `n`, а также из которой вызывается функция `fib`, записанная в отдельный файл `fib.c`, возвращающая число Фибоначчи под номером `n`. Внутри функции `fib` находится цикл, проходящий от 0 до `n-1`, в теле которого сумма переменных `first` и `second` перезаписывает значение `second`, после чего в `first` записывается предыдущее значение `second`, полученное вычитанием `first` из `second`. Итоговое значение `first` возвращается в `main`.

3.2. Преппроцессирование

```
СУПЕР-ПУПЕР КОМАНДА | C:\Users\11opt\Documents\VSCode\LowProg4> riscv64-unknown-elf-gcc.exe -march=rv32i -mabi=ilp32 -O1 -E main.c -o main.i
СУПЕР-ПУПЕР КОМАНДА | C:\Users\11opt\Documents\VSCode\LowProg4> riscv64-unknown-elf-gcc.exe -march=rv32i -mabi=ilp32 -O1 -E fib.c -o fib.i
```

Преппроцессирование

```
1276
1277 # 2 "main.c" 2
1278 # 1 "fib.h" 1
1279
1280 # 1 "fib.h"
1281 int fib(int n);
1282 # 3 "main.c" 2
1283 int n = 10;
1284
1285 int main(void) {
1286     int result = fib(n);
1287
1288     printf("%d", result);
1289     return 0;
1290 }
1291
```

Фрагмент main.i

```
C fib.i > ...
1  # 1 "fib.c"
2  # 1 "<built-in>"
3  # 1 "<command-line>"
4  # 1 "fib.c"
5
6  int fib(int n) {
7      int first = 0;
8      int second = 1;
9      for(int i = 0; i < n; i++) {
10         second += first;
11         first = second - first;
12     }
13     return first;
14 }
```

fib.i

Основа содержания препроцессированных файлов не сильно отличается от исходного кода программ.

3.3. Компиляция

```
СУПЕР-ПУПЕР КОМАНДА | C:\Users\11opt\Documents\VSCode\LowProg4> riscv64-unknown-elf-gcc.exe -march=rv32i -mabi=ilp32 -O1 -S main.i -o main.s
СУПЕР-ПУПЕР КОМАНДА | C:\Users\11opt\Documents\VSCode\LowProg4> riscv64-unknown-elf-gcc.exe -march=rv32i -mabi=ilp32 -O1 -S fib.i -o fib.s
```

Компиляция

```
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  .align 2
8  .globl main
9  .type main, @function
10 main:
11     addi    sp,sp,-16
12     sw      ra,12(sp)
13     lui     a5,%hi(n)
14     lw      a0,%lo(n)(a5)
15     call    fib
16     mv      a1,a0
17     lui     a0,%hi(.LC0)
18     addi    a0,a0,%lo(.LC0)
19     call    printf
20     li      a0,0
21     lw      ra,12(sp)
22     addi    sp,sp,16
23     jr      ra
24     .size   main,.-main
25     .globl n
26     .section .rodata.str1.4,"aMS",@progbits,1
27     .align 2
28 .LC0:
29     .string "%d"
30     .section .sdata,"aw"
31     .align 2
32     .type   n, @object
33     .size   n, 4
34 n:
35     .word   10
36     .ident  "GCC: (SiFive GCC 8.3.0-2020.04.1) 8.3.0"
```

main.s

```

1  .file "fib.c"
2  .option nopic
3  .attribute arch, "rv32i2p0"
4  .attribute unaligned_access, 0
5  .attribute stack_align, 16
6  .text
7  .align 2
8  .globl fib
9  .type fib, @function
10 fib:
11     ble a0,zero,.L4
12     li a4,0
13     li a5,1
14     li a3,0
15     j .L3
16 .L5:
17     mv a5,a2
18 .L3:
19     add a2,a3,a5
20     addi a4,a4,1
21     mv a3,a5
22     bne a0,a4,.L5
23 .L1:
24     mv a0,a5
25     ret
26 .L4:
27     li a5,0
28     j .L1
29 .size fib, .-fib
30 .ident "GCC: (SiFive GCC 8.3.0-2020.04.1) 8.3.0"

```

fib.s

По итогу компиляции получаются файлы на языке ассемблера. В тексте подпрограммы main можно увидеть call вызов подпрограммы fib, при этом символ «fib» никак не определён в файле main.s.

3.4. Ассемблирование

```
СУПЕР-ПУПЕР КОМАНДА | C:\Users\11opt\Documents\VSCode\LowProg4> riscv64-unknown-elf-gcc.exe -march=rv32i -mabi=ilp32 -c main.s -o main.o
СУПЕР-ПУПЕР КОМАНДА | C:\Users\11opt\Documents\VSCode\LowProg4> riscv64-unknown-elf-gcc.exe -march=rv32i -mabi=ilp32 -c fib.s -o fib.o
```

Ассемблирование

```
СУПЕР-ПУПЕР КОМАНДА | C:\Users\11opt\Documents\VSCode\LowProg4> riscv64-unknown-elf-objdump -f main.o fib.o

main.o:      file format elf32-littleriscv
architecture: riscv:rv32, flags 0x00000011:
HAS_RELOC, HAS_SYMS
start address 0x00000000

fib.o:       file format elf32-littleriscv
architecture: riscv:rv32, flags 0x00000011:
HAS_RELOC, HAS_SYMS
start address 0x00000000
```

Содержимое main.o и fib.o

```
СУПЕР-ПУПЕР КОМАНДА | C:\Users\11opt\Documents\VSCode\LowProg4> riscv64-unknown-elf-objdump -t fib.o main.o

fib.o:      file format elf32-littleriscv

SYMBOL TABLE:
00000000 l   df *ABS*  00000000 fib.c
00000000 l   d  .text  00000000 .text
00000000 l   d  .data  00000000 .data
00000000 l   d  .bss   00000000 .bss
00000030 l       .text  00000000 .L4
00000018 l       .text  00000000 .L3
00000014 l       .text  00000000 .L5
00000028 l       .text  00000000 .L1
00000000 l   d  .comment  00000000 .comment
00000000 l   d  .riscv.attributes  00000000 .riscv.attributes
00000000 g   F .text  00000038 fib

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   d  .sdata 00000000 .sdata
00000000 l   d  .rodata.str1.4 00000000 .LC0
00000000 l   d  .comment  00000000 .comment
00000000 l   d  .riscv.attributes  00000000 .riscv.attributes
00000000 g   F .text  0000003c main
00000000 g   O .sdata 00000004 n
00000000 *UND*  00000000 fib
00000000 *UND*  00000000 printf
```

Таблица символов

В таблице символов main.o можно увидеть символы «fib» и «printf» типа *UND*, означающего, что данные методы ещё не были определены, и должны будут определены на следующих стадиях.

```

СУПЕР-ПУПЕР КОМАНДА | C:\Users\11opt\Documents\VSCode\LowProg4> riscv64-unknown-elf-objdump -d -M no-aliases -j .text main.o

main.o:      file format elf32-littleriscv

Disassembly of section .text:

00000000 <main>:
 0: ff010113      addi    sp,sp,-16
 4: 00112623      sw      ra,12(sp)
 8: 000007b7      lui     a5,0x0
 c: 0007a503      lw      a0,0(a5) # 0 <main>
10: 00000097      auipc   ra,0x0
14: 000080e7      jalr    ra,0(ra) # 10 <main+0x10>
18: 00050593      addi    a1,a0,0
1c: 00000537      lui     a0,0x0
20: 00050513      addi    a0,a0,0 # 0 <main>
24: 00000097      auipc   ra,0x0
28: 000080e7      jalr    ra,0(ra) # 24 <main+0x24>
2c: 00000513      addi    a0,zero,0
30: 00c12083      lw      ra,12(sp)
34: 01010113      addi    sp,sp,16
38: 00008067      jalr    zero,0(ra)

```

main.o .text

```

СУПЕР-ПУПЕР КОМАНДА | C:\Users\11opt\Documents\VSCode\LowProg4> riscv64-unknown-elf-objdump -d -M no-aliases -j .text fib.o

fib.o:      file format elf32-littleriscv

Disassembly of section .text:

00000000 <fib>:
 0: 02a05863      bge     zero,a0,30 <.L4>
 4: 00000713      addi    a4,zero,0
 8: 00100793      addi    a5,zero,1
 c: 00000693      addi    a3,zero,0
10: 0080006f      jal     zero,18 <.L3>

00000014 <.L5>:
14: 00060793      addi    a5,a2,0

00000018 <.L3>:
18: 00f68633      add     a2,a3,a5
1c: 00170713      addi    a4,a4,1
20: 00078693      addi    a3,a5,0
24: fee518e3      bne     a0,a4,14 <.L5>

00000028 <.L1>:
28: 00078513      addi    a0,a5,0
2c: 00008067      jalr    zero,0(ra)

00000030 <.L4>:
30: 00000793      addi    a5,zero,0
34: ff5ff06f      jal     zero,28 <.L1>

```

fib.o .text

Можно заметить, что в дизассемблированном файле, в отличие от файла «main.i», псевдоинструкции «call» была заменена на сочетание инструкций «auipc» и «jalr». Здесь также можно увидеть, что вызовы «call» ещё не были конкретно определены и будут дополнены компоновщиком.

```

СУПЕР-ПУПЕР КОМАНДА | C:\Users\11opt\Documents\VSCode\LowProg4> riscv64-unknown-elf-objdump -d -M no-aliases -r main.o
main.o:      file format elf32-littleriscv

Disassembly of section .text:

00000000 <main>:
 0: ff010113      addi    sp,sp,-16
 4: 00112623      sw      ra,12(sp)
 8: 000007b7      lui     a5,0x0
      8: R_RISCV_HI20 n
      8: R_RISCV_RELAX *ABS*
 c: 0007a503      lw      a0,0(a5) # 0 <main>
      c: R_RISCV_LO12_I n
      c: R_RISCV_RELAX *ABS*
10: 00000097      auipc   ra,0x0
      10: R_RISCV_CALL fib
      10: R_RISCV_RELAX *ABS*
14: 000080e7      jalr    ra,0(ra) # 10 <main+0x10>
18: 00050593      addi    a1,a0,0
1c: 00000537      lui     a0,0x0
      1c: R_RISCV_HI20 .LC0
      1c: R_RISCV_RELAX *ABS*
20: 00050513      addi    a0,a0,0 # 0 <main>
      20: R_RISCV_LO12_I .LC0
      20: R_RISCV_RELAX *ABS*
24: 00000097      auipc   ra,0x0
      24: R_RISCV_CALL printf
      24: R_RISCV_RELAX *ABS*
28: 000080e7      jalr    ra,0(ra) # 24 <main+0x24>
2c: 00000513      addi    a0,zero,0
30: 00c12083      lw      ra,12(sp)
34: 01010113      addi    sp,sp,16
38: 00008067      jalr    zero,0(ra)

```

Таблица перемещений main.o, совмещённая с дизассемблированный КОДОМ.

Таблица перемещений также сообщает нам о том, что вызовы подпрограмм ещё не были определены.

3.5. Компоновка

```
78
79 00010174 <main>:
80   10174: ff010113      addi sp,sp,-16
81   10178: 00112623      sw ra,12(sp)
82   1017c: 000287b7      lui a5,0x28
83   10180: c887a503      lw a0,-888(a5) # 27c88 <n>
84   10184: 00000097      auipc ra,0x0
85   10188: 02c080e7      jalr ra,44(ra) # 101b0 <fib>
86   1018c: 00050593      addi a1,a0,0
87   10190: 00025537      lui a0,0x25
88   10194: 52050513      addi a0,a0,1312 # 25520 <__clzsi2+0x50>
89   10198: 00000097      auipc ra,0x0
90   1019c: 2c0080e7      jalr ra,704(ra) # 10458 <printf>
91   101a0: 00000513      addi a0,zero,0
92   101a4: 00c12083      lw ra,12(sp)
93   101a8: 01010113      addi sp,sp,16
94   101ac: 00008067      jalr zero,0(ra)
95
96 000101b0 <fib>:
97   101b0: 02a05863      bge zero,a0,101e0 <fib+0x30>
98   101b4: 00000713      addi a4,zero,0
99   101b8: 00100793      addi a5,zero,1
100  101bc: 00000693      addi a3,zero,0
101  101c0: 0080006f      jal zero,101c8 <fib+0x18>
102  101c4: 00060793      addi a5,a2,0
103  101c8: 00f68633      add a2,a3,a5
104  101cc: 00170713      addi a4,a4,1
105  101d0: 00078693      addi a3,a5,0
106  101d4: fee518e3      bne a0,a4,101c4 <fib+0x14>
107  101d8: 00078513      addi a0,a5,0
108  101dc: 00008067      jalr zero,0(ra)
109  101e0: 00000793      addi a5,zero,0
110  101e4: ff5ff06f      jal zero,101d8 <fib+0x28>
111
```

Скомпонованный файл `main` с отредактированными вызовами методов.

Можно увидеть, что компоновщик отредактировал инструкции вызова так, в результате чего «`jalr`» осуществит корректные переходы к адресам подпрограмм «`fib`» и «`printf`».

4. Сборка с помощью Make-файла

```
M Makefile
1  output: main.o fiblib.a
2      mingw32-gcc-9.2.0.exe main.o fiblib.a -o output
3
4  main.o: main.c
5      mingw32-gcc-9.2.0.exe -c main.c
6
7  fiblib.a: fib.o fib.h
8      mingw32-gcc-ar.exe -rsc fiblib.a fib.o
9
10 fib.o:
11     mingw32-gcc-9.2.0.exe -c fib.c
12
13 clean:
14     del *.o *.a *.exe
```

Makefile

```
СУПЕР-ПУПЕР КОМАНДА | C:\Users\11opt\Documents\VSCode\LowProg4> mingw32-make.exe
mingw32-gcc-9.2.0.exe -c main.c
mingw32-gcc-9.2.0.exe -c fib.c
mingw32-gcc-ar.exe -rsc fiblib.a fib.o
mingw32-gcc-9.2.0.exe main.o fiblib.a -o output

СУПЕР-ПУПЕР КОМАНДА | C:\Users\11opt\Documents\VSCode\LowProg4> output.exe
55
```

Сборка библиотеки с исполняемым файлом и результат работы для
 $n = 10$

Содержание Makefile:

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

Вывод

В данной лабораторной работе мы познакомились с процессом сборки проекта на языке C.

Он состоит из:

1. Препроцессирования: исходного .c файл препроцессируем в .i файл
2. Компиляции: полученный .i файл компилируется в файл ассемблера .s
3. Ассемблирования: файл .s ассемблируется в объектный файл .o
4. Компоновки: объектный файл .o компоуется в исполняемый файл

Также мы ознакомились в makefile'ами, которые упрощают процесс сборки.