

## **Отчёт по лабораторной работе № 4**

Дисциплина: Низкоуровневое программирование

Тема: Раздельная компиляция

Вариант: 13

Выполнил студент гр. 3530901/00002 \_\_\_\_\_ А.Д. Чешев  
(подпись)

Принял преподаватель \_\_\_\_\_ Д.С. Степанов  
(подпись)

“ \_\_\_\_ ” \_\_\_\_\_ 2021 г.

Санкт-Петербург

2021

### **Постановка 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.

## 1. Код реализованной программы.

```
#include <stdio.h>
#include "gorn.h"

int main() {
    int x = 4;
    int size = 6;
    int arr[6] = {3, 2, 6, 1, 4, 15};
    printf(_Format: "%d\n", gorn(arr, x, size));
    return 0;
}
```

Рис. 1. Файл main.c.

```
#ifndef LAB4_GORN_H
#define LAB4_GORN_H

int gorn(const int arr[], int x, int size);

#endif //LAB4_GORN_H
```

Рис. 2. Файл gorn.h.

```
#include "gorn.h"

int gorn(const int arr[], int x, int size)
{
    int result = 0;
    for(int i = 0; i < size; i++)
    {
        result = result * x + arr[i];
    }
    return result;
}
```

Рис. 3. Файл gorn.c.

## 2. Препроцессирование

.\riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -O1 -E main.c -o main.i

.\riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -O1 -E gorn.c -o gorn.i

```
# 1 "C:\\Users\\chesh\\CLionProjects\\lab4\\main.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "C:\\Users\\chesh\\CLionProjects\\lab4\\main.c"
# 1 "d:\\riscv64-unknown-elf-toolchain-10.2.0-2020.12.8-x86_64-

# 4 "C:\\Users\\chesh\\CLionProjects\\lab4\\gorn.h"
int gorn(const int arr[], int x, int size);
# 3 "C:\\Users\\chesh\\CLionProjects\\lab4\\main.c" 2

int main() {
    int x = 3;
    int size = 6;
    int arr[6] = {3, 2, 6, 1, 4, 15};
    printf("%d\\n", gorn(arr, x, size));
    return 0;
}
```

Рис. 4. Препроцессирование файла main.c.

```
# 1 "C:\\Users\\chesh\\CLionProjects\\lab4\\gorn.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "C:\\Users\\chesh\\CLionProjects\\lab4\\gorn.c"
# 1 "d:\\riscv64-unknown-elf-toolchain-10.2.0-2020.12.8-x86_64-

# 4 "C:\\Users\\chesh\\CLionProjects\\lab4\\gorn.h"
int gorn(const int arr[], int x, int size);
# 3 "C:\\Users\\chesh\\CLionProjects\\lab4\\gorn.c" 2

int gorn(const int arr[], int x, int size){
    int result = 0;
    for(int i = 0; i < size; i++){
        result = result * x + arr[i];
    }
    return result;
}
```

Рис. 5. Препроцессирование файла gorn.c.

### 3. Компиляция

```
./riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -O1 -v -S -fpreprocessed main.i -o  
main.s
```

```
./riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -O1 -v -S -fpreprocessed gorn.i -o  
gorn.s
```

На данном шаге выполняется компиляция файла “main.i”, уже обработанного препроцессором (опция “-fpreprocessed”), результат работы компилятора – код на языке ассемблера – сохраняется в файле ”main.s”. Опция “-S” – остановка процесса сборки после компиляции.

```
.file "main.c"  
.option nopic  
.attribute arch, "rv64i2p0_a2p0_c2p0"  
.attribute unaligned_access, 0  
.attribute stack_align, 16  
.text  
.section .rodata.str1.8,"aMS",@progbits,1  
.align 3  
.LC1:  
.string "%d\n"  
.text  
.align 1  
.globl main  
.type main, @function  
main:  
addi sp,sp,-48  
sd ra,40(sp)  
lui a5,%hi(.LANCHOR0)  
addi a5,a5,%lo(.LANCHOR0)  
ld a4,0(a5)  
sd a4,8(sp)  
ld a4,8(a5)  
sd a4,16(sp)  
ld a5,16(a5)  
sd a5,24(sp)  
li a2,6  
li a1,3  
addi a0,sp,8  
call gorn  
mv a1,a0  
lui a0,%hi(.LC1)  
addi a0,a0,%lo(.LC1)  
call printf  
li a0,0  
ld ra,40(sp)  
addi sp,sp,48  
jr ra  
.size main,.-main  
.section .rodata  
.align 3  
.set .LANCHOR0,.- + 0  
.LC0:  
.word 3  
.word 2  
.word 6  
.word 1  
.word 4  
.word 15  
.ident "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"
```

Рис. 6. Файл main.s.

```

.file "gorn.c"
.option nopic
.attribute arch, "rv64i2p0_a2p0_c2p0"
.attribute unaligned_access, 0
.attribute stack_align, 16
.text
.globl __muldi3
.align 1
.globl gorn
.type gorn, @function
gorn:
    ble a2,zero,.L4
    addi sp,sp,-32
    sd ra,24(sp)
    sd s0,16(sp)
    sd s1,8(sp)
    sd s2,0(sp)
    mv s2,a1
    mv s0,a0
    addiw s1,a2,-1
    slli a5,s1,32
    srli s1,a5,30
    addi a0,a0,4
    add s1,s1,a0
    li al,0
.L3:
    mv a0,s2
    call __muldi3
    lw al,0(s0)
    addw al,al,a0
    addi s0,s0,4
    bne s0,s1,.L3
    mv a0,al
    ld ra,24(sp)
    ld s0,16(sp)
    ld s1,8(sp)
    ld s2,0(sp)
    addi sp,sp,32
    jr ra
.L4:
    li al,0
    mv a0,al
    ret
.size gorn, .-gorn
.ident "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"

```

Рис. 7. Файл gorn.s.

## 4. Ассемблирование

```
./riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -v -c main.s -o main.o
```

```
./riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -v -c gorn.s -o gorn.o
```

Опция “-с” останавливает процесс сборки после ассемблирования. Используя утилиту objdump, получим содержимое бинарных файлов в текстовом виде.

```
./riscv64-unknown-elf-objdump -f main.o
```

```
C:\Users\chesh\CLionProjects\lab4\an\main.o:      file format elf64-littleriscv
architecture: riscv:rv64, flags 0x00000011:
HAS_RELOC, HAS_SYMS
start address 0x0000000000000000
```

Рис. 8. File (main.o) header.

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

```
C:\Users\chesh\CLionProjects\lab4\an\main.o:      file format elf64-littleriscv

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .text          00000040  0000000000000000  0000000000000000  00000040  2**1
CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data           00000000  0000000000000000  0000000000000000  00000080  2**0
CONTENTS, ALLOC, LOAD, DATA
  2 .bss            00000000  0000000000000000  0000000000000000  00000080  2**0
ALLOC
  3 .rodata.str1.8  00000004  0000000000000000  0000000000000000  00000080  2**3
CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .rodata         00000018  0000000000000000  0000000000000000  00000088  2**3
CONTENTS, ALLOC, LOAD, READONLY, DATA
  5 .comment        00000031  0000000000000000  0000000000000000  000000a0  2**0
CONTENTS, READONLY
  6 .riscv.attributes 00000026  0000000000000000  0000000000000000  000000d1  2**0
CONTENTS, READONLY
```

Рис. 9. Отображение заголовков секций (main.o).

.text (64 байта)– секция кода, в которой содержатся коды инструкций (название секции

обусловлено историческими причинами);

.data (0 байт) – секция инициализированных данных;

.bss (0 байт) – секция данных, инициализированных нулями (название секции также обусловлено историческими причинами);

.comment (49 байт)– секция данных о версиях размером

Изучим таблицу символов файла:

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

```
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.str1.8 0000000000000000 .rodata.str1.8
0000000000000000 1   d  .rodata      0000000000000000 .rodata
0000000000000000 1   d  .rodata      0000000000000000 .LANCHOR0
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  0000000000000040 main
0000000000000000 *UND* 0000000000000000 gorn
0000000000000000 *UND* 0000000000000000 printf
```

Рис. 10. Таблица символов файла main.o.

Изучим содержимое секции “.text”.

./riscv64-unknown-elf-objdump -s -j .text main.o

```
Contents of section .text:
0000 797106f4 b7070000 93870700 98633ae4  yq.....c:..
0010 98673ae8 9c6b3eec 19468d45 28009700  .g:...k>..F.E(...)
0020 0000e780 0000aa85 37050000 13050500  .....7.....
0030 97000000 e7800000 0145a270 45618280  .....E.pEa..
```

Рис. 11. Содержимое секции ".text".



71	79
f4	06

inst[15:13]	-	-	inst[1..0]
011	10001	011110	01
111	10100	000001	10

В соответствии со спецификацией (The RISC-V Instruction Set Manual Volume I: User-Level ISA):

inst[15:13]		000	001	010	011	100	101	110	111	
inst[1:0]										
00	ADDI4SPN	FLD FLD LQ	LW	FLW LD LD	Reserved	FSD FSD SQ	SW	FSW SD SD	RV32 RV64 RV128	
01	ADDI	JAL ADDIW ADDIW	LI	LUI/ADDI16SP	MISC-ALU	J	BEQZ	BNEZ	RV32 RV64 RV128	
10	SLLI	FLDSP FLDSP LQ	LWSP	FLWSP LDSP LDSP	J[AL]R/MV/ADD	FSDSP FSDSP SQ	SWSP	FSWSP SDSP SDSP	RV32 RV64 RV128	

Рис. 12. RVC opcode map.

15	13	12	11	7	6	2	1	0
funct3	imm[5]	rd/rs1	imm[4:0]	op				
3	1	5	5	2				
C.ADDI	nzimm[5]	dest	nzimm[4:0]	C1				
C.ADDIW	imm[5]	dest≠0	imm[4:0]	C1				
C.ADDI16SP	nzimm[9]	2	nzimm[4 6 8:7 5]	C1				

Рис. 13. Формат инструкции C.ADDI16SP.

15	13	12	7	6	2	1	0
funct3	imm	rs2	op				
3	6	5	2				
C.SWSP	offset[5:2 7:6]	src	C2				
C.SDSP	offset[5:3 8:6]	src	C2				
C.SQSP	offset[5:4 9:6]	src	C2				
C.FSWSP	offset[5:2 7:6]	src	C2				
C.FSDSP	offset[5:3 8:6]	src	C2				

Рис. 14. Формат инструкции C.SDSP.

funct3	nzimm[9]	rd/rs1	nzimm[4 6 8:7 5]	op
011	1	00010	11110	01

funct3	offset[5:3 8:6]	rs2	op
111	101000	00001	10

Запишем полученные инструкции на языке ассемблера:

addi16sp x2, x2, nzim[9..4]

sdsp rs2, offset[8..3] (x2)

Перепишем инструкции, используя обозначения регистров, принятые в ABI:

addi16sp sp,-48

sdsp ra, 40(sp)

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

```
Disassembly of section .text:
0000000000000000 <main>:
0: 7179          c.addi16sp      sp,-48
2: f406          c.sdsp         ra,40(sp)
4: 000007b7      lui           a5,0x0
8: 00078793      addi          a5,a5,0 # 0 <main>
c: 6398          c.ld           a4,0(a5)
e: e43a          c.sdsp         a4,8(sp)
10: 6798          c.ld           a4,8(a5)
12: e83a          c.sdsp         a4,16(sp)
14: 6b9c          c.ld           a5,16(a5)
16: ec3e          c.sdsp         a5,24(sp)
18: 4619          c.li           a2,6
1a: 458d          c.li           a1,3
1c: 0028          c.addi4spn     a0,sp,8
1e: 00000097      auipc         ra,0x0
22: 000080e7      jalr          ra,0(ra) # 1e <main+0x1e>
26: 85aa          c.mv           a1,a0
28: 00000537      lui           a0,0x0
2c: 00050513      addi          a0,a0,0 # 0 <main>
30: 00000097      auipc         ra,0x0
34: 000080e7      jalr          ra,0(ra) # 30 <main+0x30>
38: 4501          c.li           a0,0
3a: 70a2          c.ldsp         ra,40(sp)
3c: 6145          c.addi16sp     sp,48
3e: 8082          c.jr           ra
```

Рис. 15. Процесс дизассемблирования.

## 5. Компоновка

```
./riscv64-unknown-elf-gcc -march=rv64iac -mabi=lp64 -v main.o gorn.o -o main.out
```

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

```
C:\Users\chesh\CLionProjects\lab4\an\main.out:      file format  
elf64-littleriscv
```

```
Disassembly of section .text:
```

```
...
```

```
00000000000010156 <main>:
```

```
10156: 7179          c.addi16sp sp,-48  
10158: f406          c.sdsp      ra,40(sp)  
1015a: 67f5          c.lui a5,0x1d  
1015c: b2878793      addi a5,a5,-1240 # 1cb28  
<__clzdi2+0x46>  
10160: 6398          c.ld a4,0(a5)  
10162: e43a          c.sdsp      a4,8(sp)  
10164: 6798          c.ld a4,8(a5)  
10166: e83a          c.sdsp      a4,16(sp)  
10168: 6b9c          c.ld a5,16(a5)  
1016a: ec3e          c.sdsp      a5,24(sp)  
1016c: 4619          c.li a2,6  
1016e: 458d          c.li a1,3  
10170: 0028          c.addi4spn a0,sp,8  
10172: 018000ef      jal ra,1018a <gorn>  
10176: 85aa          c.mv a1,a0  
10178: 6575          c.lui a0,0x1d  
1017a: b2050513      addi a0,a0,-1248 # 1cb20  
<__clzdi2+0x3e>  
1017e: 1c0000ef      jal ra,1033e <printf>  
10182: 4501          c.li a0,0  
10184: 70a2          c.ldsp      ra,40(sp)  
10186: 6145          c.addi16sp sp,48  
10188: 8082          c.jr ra  
...
```

Рис. 16. Секция main в результирующем файле main.out.

```

...
000000000001018a <gorn>:
 1018a: 04c05163      bge zero,a2,101cc <gorn+0x42>
 1018e: 1101          c.addi      sp,-32
 10190: ec06          c.sdsp      ra,24(sp)
 10192: e822          c.sdsp      s0,16(sp)
 10194: e426          c.sdsp      s1,8(sp)
 10196: e04a          c.sdsp      s2,0(sp)
 10198: 892e          c.mv s2,a1
 1019a: 842a          c.mv s0,a0
 1019c: fff6049b      addiws1,a2,-1
 101a0: 02049793      slli a5,s1,0x20
 101a4: 01e7d493      srli s1,a5,0x1e
 101a8: 0511          c.addi      a0,4
 101aa: 94aa          c.adds1,a0
 101ac: 4581          c.li a1,0
 101ae: 854a          c.mv a0,s2
 101b0: 024000ef      jal ra,101d4 <__muldi3>
 101b4: 400c          c.lw a1,0(s0)
 101b6: 9da9          c.addw      a1,a0
 101b8: 0411          c.addi      s0,4
 101ba: fe941ae3      bne s0,s1,101ae <gorn+0x24>
 101be: 852e          c.mv a0,a1
 101c0: 60e2          c.ldsp      ra,24(sp)
 101c2: 6442          c.ldsp      s0,16(sp)
 101c4: 64a2          c.ldsp      s1,8(sp)
 101c6: 6902          c.ldsp      s2,0(sp)
 101c8: 6105          c.addi16sp sp,32
 101ca: 8082          c.jr ra
 101cc: 4581          c.li a1,0
 101ce: 852e          c.mv a0,a1
 101d0: 8082          c.jr ra
...

```

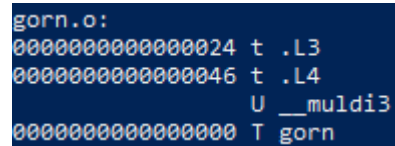
Рис. 17. Секция gorn в результирующем файле main.out.

## 6. Создание статической библиотеки и make-файлов.

`./riscv64-unknown-elf-ar -rsc lib.a gorn.o` – создание библиотеки

`./riscv64-unknown-elf-ar -t lib.a` – содержимое библиотек `lib.a`

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



```
gorn.o:
0000000000000024 t .L3
0000000000000046 t .L4
                U __muldi3
0000000000000000 T gorn
```

Рис. 18. Вывод команды `./riscv64-unknown-elf-nm lib.a`.

Создадим make-файлы для сборки библиотеки и использующей ее тестовой программы.

```
.PHONY: all clean

OBJS= gorn.c

AR = riscv64-unknown-elf-ar.exe
CC = riscv64-unknown-elf-gcc.exe

MYLIBNAME = lib.a

CFLAGS= -O1

INCLUDES+= -I .

vpath %.h .
vpath %.c .

%.o: %.c
    $(CC) -MD $(CFLAGS) $(INCLUDES) -c $< -o $@

all: $(MYLIBNAME)

$(MYLIBNAME): gorn.o
    $(AR) -rsc $@ $^
```

Рис. 19. Make-файл для создания библиотеки `lib.a`.

```

.PHONY: all clean

# файлы для сборки исполнимого файла
OBJS= main.o lib.a

CC = riscv64-unknown-elf-gcc.exe

CFLAGS= -march=rv64iac -mabi=lp64 -O1 --save-temps

INCLUDES+= -I .

vpath %.c .
vpath %.a .

all: a.out

# Сборка исполнимого файла
a.out: $(OBJS)
    $(CC) $(CFLAGS) $(INCLUDES) $^
    del *.i *.d

```

Рис. 20. Make-файл для сборки тестовой программы main.c.

## **Вывод**

В ходе проделанной работы была выполнена раздельная компиляция, изучены принципы создания статической библиотеки, make-файлов.