Санкт-Петербургский политехнический университет Петра Великого

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Отчёт по лабораторной работе№ 4**

**Дисциплина**: Низкоуровневое программирование

**Тема**: Раздельная компиляция

Выполнил студент гр. 3530901/10003 Курочкин С.В

(подпись)

Преподаватель Коренев Д. А. (подпись)

“ ”

Санкт-Петербург

Оглавление

1. ТЗ – 3 стр
2. Метод решения – 3 стр
3. Руководство программисту – 3 стр
4. Реализация программы 1 – 3 стр
5. Работа программы 1 – 4 стр
6. Реализация программы 2 – 6 стр
7. Работа программы 2 – 7 стр

1. Техническое задание

Сделать перестановку в массиве чисел. Сначала элементы с нечетными индексами, потом с четными

2. Метод решения

Метод решения заключается в проходе по всем четным числам и перемещении их в конец массива

3. Ход решения

**Текст программы на языке C:**

main.c

Graphical user interface, text

Description automatically generated

fun.h

A screen shot of a computer

Description automatically generated with low confidence

**Результат работы препроцессора**

main.i

Text

Description automatically generated

fun.i

Text

Description automatically generated

Как видно, результат работы препроцессора не сильно различается с исходным кодом программы

**Результат работы компилятора**

fun.s

.file "fun.h"

.option nopic

.attribute arch, "rv32i2p0"

.attribute unaligned\_access, 0

.attribute stack\_align, 16

main.s

.file "main.c"

.option nopic

.attribute arch, "rv32i2p0"

.attribute unaligned\_access, 0

.attribute stack\_align, 16

.text

.align 2

.globl fun

.type fun, @function

fun:

addi a5,a1,1

srli a6,a5,31

add a6,a6,a5

srai a6,a6,1

ble a1,zero,.L1

mv a7,a0

slli a2,a1,2

addi a0,a0,-4

add a2,a2,a0

li a0,0

addi a1,a1,-1

j .L3

.L6:

addi a0,a0,1

bge a0,a6,.L1

.L3:

slli a5,a0,2

add a5,a5,a7

ble a1,a0,.L6

.L4:

lw a4,0(a5)

lw a3,4(a5)

sw a3,0(a5)

sw a4,4(a5)

addi a5,a5,4

bne a5,a2,.L4

j .L6

.L1:

ret

.size fun, .-fun

.align 2

.globl main

.type main, @function

main:

addi sp,sp,-64

sw ra,60(sp)

lui a5,%hi(.LANCHOR0)

addi a5,a5,%lo(.LANCHOR0)

lw t4,0(a5)

lw t3,4(a5)

lw t1,8(a5)

lw a7,12(a5)

lw a6,16(a5)

lw a0,20(a5)

lw a1,24(a5)

lw a2,28(a5)

lw a3,32(a5)

lw a4,36(a5)

lw a5,40(a5)

sw t4,4(sp)

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 a1,11

addi a0,sp,4

call fun

li a0,0

lw ra,60(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

.word 10

.ident "GCC: (SiFive GCC 8.3.0-2020.04.1) 8.3.0"

Как мы можем видеть, код из fun был вставлен в main. Скорее всего это связано с тем, что реализация функции fun была написана прямо в заголовочном файле fun.h

**Результат работы ассемблера**

Объектный файл не является текстовым, для изучения его содержимого используем утилиту objdump:

**Text

Description automatically generated**

Видно, что файл main.o имеет таблицу перемещений

Выведем все заголовки секций объектных файлов

Graphical user interface, text

Description automatically generated

В файле “main.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

Теперь изучим таблицу символов

Text

Description automatically generated

В таблице находится 2 глобальный символа (флаг g)

**Содержимое секций**

Изучим содержимое секции .text

Graphical user interface

Description automatically generated with medium confidence

A picture containing text

Description automatically generated

Дизассемблированный код практически идентичен сгенерированному (за исключением псевдоинструкций).

Секция .data не содержит данных

Text

Description automatically generated

Секция .bss также не содержит данных

A screenshot of a computer

Description automatically generated with medium confidence

Секция .rodata

Text

Description automatically generated

Секция .comment содержит сведения о компиляторе, который был использован - GCC версии 8.3.0 от SiFive

Text

Description automatically generated

Секция .riscv.attributes содержит информацию об используемой архитектуре команд - RV32I

Text

Description automatically generated

Посмотрим содержимое таблицы перемещений:

Text

Description automatically generated

Таблица перемещений содержит информацию о переходах, ветвлениях, вызовах и оптимизации

**Результат компоновщика**

Результатом работы компоновщика является исполняемый файл a.out

Он тоже является бинарным, будем использовать все тот же objdump

Text

Description automatically generated

Видно, что файл действительно является исполняемым (флаг EXEC\_P).

Также можно увидеть информацию о начальном адресе программы 0x00010090

Изучим секции файла:

Graphical user interface, text

Description automatically generated

Видно, что секций стало больше, по сравнению с объектным файлом, а также увеличились их размеры

Изучим содержимое секции .text

Строк кода стало намного больше, чем в объектном файле, так что здесь я оставлю только самые важные

Disassembly of section .text:

00010090 <\_start>:

10090: 00002197 auipc gp,0x2

10094: df818193 addi gp,gp,-520 # 11e88 <\_\_global\_pointer$>

10098: c3418513 addi a0,gp,-972 # 11abc <\_edata>

1009c: c5018613 addi a2,gp,-944 # 11ad8 <\_\_BSS\_END\_\_>

100a0: 40a60633 sub a2,a2,a0

100a4: 00000593 addi a1,zero,0

100a8: 2c0000ef jal ra,10368 <memset>

100ac: 00000517 auipc a0,0x0

100b0: 1c850513 addi a0,a0,456 # 10274 <\_\_libc\_fini\_array>

100b4: 178000ef jal ra,1022c <atexit>

100b8: 21c000ef jal ra,102d4 <\_\_libc\_init\_array>

100bc: 00012503 lw a0,0(sp)

100c0: 00410593 addi a1,sp,4

100c4: 00000613 addi a2,zero,0

100c8: 0e0000ef jal ra,101a8 <main>

100cc: 1740006f jal zero,10240 <exit>

00010144 <fun>:

10144: 00158793 addi a5,a1,1

10148: 01f7d813 srli a6,a5,0x1f

1014c: 00f80833 add a6,a6,a5

10150: 40185813 srai a6,a6,0x1

10154: 04b05863 bge zero,a1,101a4 <fun+0x60>

10158: 00050893 addi a7,a0,0

1015c: 00259613 slli a2,a1,0x2

10160: ffc50513 addi a0,a0,-4

10164: 00a60633 add a2,a2,a0

10168: 00000513 addi a0,zero,0

1016c: fff58593 addi a1,a1,-1

10170: 00c0006f jal zero,1017c <fun+0x38>

10174: 00150513 addi a0,a0,1

10178: 03055663 bge a0,a6,101a4 <fun+0x60>

1017c: 00251793 slli a5,a0,0x2

10180: 011787b3 add a5,a5,a7

10184: feb558e3 bge a0,a1,10174 <fun+0x30>

10188: 0007a703 lw a4,0(a5)

1018c: 0047a683 lw a3,4(a5)

10190: 00d7a023 sw a3,0(a5)

10194: 00e7a223 sw a4,4(a5)

10198: 00478793 addi a5,a5,4

1019c: fec796e3 bne a5,a2,10188 <fun+0x44>

101a0: fd5ff06f jal zero,10174 <fun+0x30>

101a4: 00008067 jalr zero,0(ra)

000101a8 <main>:

101a8: fc010113 addi sp,sp,-64

101ac: 02112e23 sw ra,60(sp)

101b0: 000107b7 lui a5,0x10

101b4: 64c78793 addi a5,a5,1612 # 1064c <\_\_errno+0xc>

101b8: 0007ae83 lw t4,0(a5)

101bc: 0047ae03 lw t3,4(a5)

101c0: 0087a303 lw t1,8(a5)

101c4: 00c7a883 lw a7,12(a5)

101c8: 0107a803 lw a6,16(a5)

101cc: 0147a503 lw a0,20(a5)

101d0: 0187a583 lw a1,24(a5)

101d4: 01c7a603 lw a2,28(a5)

101d8: 0207a683 lw a3,32(a5)

101dc: 0247a703 lw a4,36(a5)

101e0: 0287a783 lw a5,40(a5)

101e4: 01d12223 sw t4,4(sp)

101e8: 01c12423 sw t3,8(sp)

101ec: 00612623 sw t1,12(sp)

101f0: 01112823 sw a7,16(sp)

101f4: 01012a23 sw a6,20(sp)

101f8: 00a12c23 sw a0,24(sp)

101fc: 00b12e23 sw a1,28(sp)

10200: 02c12023 sw a2,32(sp)

10204: 02d12223 sw a3,36(sp)

10208: 02e12423 sw a4,40(sp)

1020c: 02f12623 sw a5,44(sp)

10210: 00b00593 addi a1,zero,11

10214: 00410513 addi a0,sp,4

10218: f2dff0ef jal ra,10144 <fun>

1021c: 00000513 addi a0,zero,0

10220: 03c12083 lw ra,60(sp)

10224: 04010113 addi sp,sp,64

10228: 00008067 jalr zero,0(ra)

00010240 <exit>:

10240: ff010113 addi sp,sp,-16

10244: 00000593 addi a1,zero,0

10248: 00812423 sw s0,8(sp)

1024c: 00112623 sw ra,12(sp)

10250: 00050413 addi s0,a0,0

10254: 28c000ef jal ra,104e0 <\_\_call\_exitprocs>

10258: c2818793 addi a5,gp,-984 # 11ab0 <\_global\_impure\_ptr>

1025c: 0007a503 lw a0,0(a5)

10260: 03c52783 lw a5,60(a0)

10264: 00078463 beq a5,zero,1026c <exit+0x2c>

10268: 000780e7 jalr ra,0(a5)

1026c: 00040513 addi a0,s0,0

10270: 38c000ef jal ra,105fc <\_exit>

000105fc <\_exit>:

105fc: 00000593 addi a1,zero,0

10600: 00000613 addi a2,zero,0

10604: 00000693 addi a3,zero,0

10608: 00000713 addi a4,zero,0

1060c: 00000793 addi a5,zero,0

10610: 05d00893 addi a7,zero,93

10614: 00000073 ecall

10618: 00054463 blt a0,zero,10620 <\_exit+0x24>

1061c: 0000006f jal zero,1061c <\_exit+0x20>

10620: ff010113 addi sp,sp,-16

10624: 00812423 sw s0,8(sp)

10628: 00050413 addi s0,a0,0

1062c: 00112623 sw ra,12(sp)

10630: 40800433 sub s0,zero,s0

10634: 00c000ef jal ra,10640 <\_\_errno>

10638: 00852023 sw s0,0(a0)

1063c: 0000006f jal zero,1063c <\_exit+0x40>

Видно функции main и fun, а также метки для входа и выхода из программы: start и exit

Секция .comment хранит все те же сведения о GCCверсии 8.3.0 от SiFive:

Text

Description automatically generated

Секция .riscv.attributes по-прежнему содержит информацию об используемой архитектуре команд RV32I:

Text

Description automatically generated

Проанализируем таблицу символов исполняемого файла

Text

Description automatically generated

Text

Description automatically generated

Таблица символов содержит множество дополнительных вхождений, однако в целом определяет все нужные секции, метки и адреса. Функции fun и main так же помечены флагом F, но в отличие от стадии ассемблирования, все они являются определенными и содержатся по корректным адресам для успешного вызова этих функций из других участков программ.

A black screen with white text

Description automatically generated with low confidence

Таблица перемещений оказывается пуста, все необходимые релокации, оптимизации и замены инструкций были успешно проведены компоновщиком.

Итогом сборки программ на языке C по шагам является исполняемый на процессорах архитектуры RISC-V файл, решающий задачу перемещения элементов массива с четными индексами в конец массива

**Разработка статической библиотеки, разработка make-файлов для сборки библиотеки**

Статическая библиотека (static library) является, по сути, архивом (набором, коллекцией) объектных файлов, среди которых компоновщик выбирает «полезные» для данной программы: объектный файл считается «полезным», если в нем определяется еще не разрешенный компоновщиком символ.

Text

Description automatically generated

Поиск библиотеки и сборка программы



На выходе получили файл main

Text

Description automatically generated