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 A. А.

Группа: 3530901/10003

Преподаватель: Коренев Д. А.

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 и 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.

**Вариант 9:** Реверс массива целых чисел.

1. **Метод решения**

Тестовая программа (*main.c*) передает в функцию (reverse), находящуюся в отдельном файле (*reverse.c*), массив и его длину. Функция (reverse) сначала записывает первый операнд в «буфер», затем первый операнд перезаписывается вторым и далее второй значением из буфера.

1. **Программа на языке «C»**

**3.1.  Текст программ, реализующих определенную вариантом задания функциональность**

Программа состоит из трех файлов: *reverse.h* (файл-заголовок), *main.c* (с тестовой программой), *reverse.c* (с основной программой).

Заголовочный файл *reverse.h* содержит в себе определение функции reverse. В дальнейшем для использования этой функции в другой программе необходимо организовывать подключение этого заголовочного файла.

Такая конструкция файла *reverse.h* предотвращает двойное подключение заголовочного файла путём проверки существования этого макроса, который имеет то же самое имя, что и заголовочный файл. Определение макроса REVERSE\_H происходит, когда заголовочный файл впервые обрабатывается препроцессором. Затем, если этот заголовочный файл вновь подключается, REVERSE\_H уже определен, в результате чего препроцессор пропускает полностью текст этого заголовочного файла.

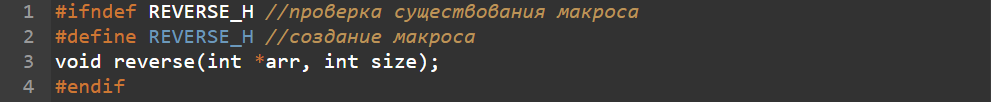


Рисунок 1. reverse.h

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2. main.c

Основная функция (reverse) работает по следующему принципу. На вход подается массив (на деле – указатель на массив, т.к массив нельзя передать как аргумент в функцию) и его размер. Цикл проходит по size/2 элементам, для получения первого операнда, а второй операнд вычисляется в цикле. Затем они с помощью буфера меняются местами. (рис. 3)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2. reverse.c

Результат работы программы верен.

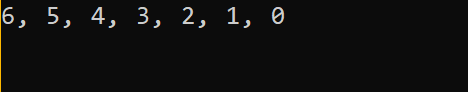


Рисунок 3. Результат работы программы

**3.2. Сборка программ «по шагам», анализ промежуточных и результирующих файлов**

Препроцессирование выполняется следующими командами:



Драйвер компилятора gcc – riscv64-unknown-elf-gcc запускается с параметрами командной строки «-march=rv32i» и «-mabi=ilp32», указывающих что целевым является процессор с базовой архитектурой системы команд RV32I, а «-O1» – указание выполнять простые оптимизации генерируемого кода; «-E» – указание остановить процесс сборки после препроцессирования.

Результаты содержатся в файлах *main.i* и *reverse.i* (фрагменты):

* **main.i**

# 1 "main.c"

# 1 "<built-in>"

# 1 "<command-line>"

# 1 "main.c"

# 1 "reverse.h" 1

void reverse(int \*arr, int size);

# 2 "main.c" 2

- - - - - - - - - - - - - - - - - - - -

# 4 "main.c" 2

# 5 "main.c"

int main() {

int arr[7] = {0, 1, 2, 3, 4, 5, 6};

int size = 7;

reverse(arr, size);

for (int i = 0; i < size - 1; i++) {

printf("%d, ", arr[i]);

}

printf("%d", arr[size - 1]);

scanf("1");

return 0;

}

* **reverse.i**

# 1 "reverse.c"

# 1 "<built-in>"

# 1 "<command-line>"

# 1 "reverse.c"

# 1 "reverse.h" 1

void reverse(int \*arr, int size);

# 2 "reverse.c" 2

- - - - - - - - - - - - - - - - - - - -

# 4 "reverse.c" 2

# 7 "reverse.c"

void reverse(int \*arr, int size) {

for (int i = 0; i <= (size/2); i++) {

int buff = arr[i];

int n = size - (i + 1);

arr[i] = arr[n];

arr[n] = buff;

}

}

Символ «#» используются для передачи информации об исходном тексте из препроцессора в компилятор. Препроцессор позволяет включать в текст программы файлы. Так, в файле *reverse.c* первая строка «*#include "reverse.h"*» и результат её обработки представлен на 4-6 строках (вставка содержимого заголовочного файла). Представлены фрагменты кода, так как из-за включения заголовочных файлов функций ввода-вывода (*stdio.h*) и стандартной библиотеки языка C (*stdlib.h*) файлы оказались слишком длинными. Исходный код тестирующей функции main после работы препроцессора остался без изменений, как и исходный код функции reverse.

**3.3. Компиляция**

Компиляция в файлы *main.s* и *reverse.s* осуществляется следующими командами:



Получившийся код на языке ассемблера содержится в файлах *reverse.s* и *main.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,-48

sw ra,44(sp)

sw s0,40(sp)

sw s1,36(sp)

sw s2,32(sp)

lui a5,%hi(.LANCHOR0)

addi a5,a5,%lo(.LANCHOR0)

lw a6,0(a5)

lw a0,4(a5)

lw a1,8(a5)

lw a2,12(a5)

lw a3,16(a5)

lw a4,20(a5)

lw a5,24(a5)

sw a6,4(sp)

sw a0,8(sp)

sw a1,12(sp)

sw a2,16(sp)

sw a3,20(sp)

sw a4,24(sp)

sw a5,28(sp)

li a1,7

addi a0,sp,4

call reverse

addi s0,sp,4

addi s2,sp,28

lui s1,%hi(.LC1)

.L2:

lw a1,0(s0)

addi a0,s1,%lo(.LC1)

call printf

addi s0,s0,4

bne s0,s2,.L2

lw a1,28(sp)

lui a0,%hi(.LC2)

addi a0,a0,%lo(.LC2)

call printf

lui a0,%hi(.LC3)

addi a0,a0,%lo(.LC3)

call scanf

li a0,0

lw ra,44(sp)

lw s0,40(sp)

lw s1,36(sp)

lw s2,32(sp)

addi sp,sp,48

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

.section .rodata.str1.4,"aMS",@progbits,1

.align 2

.LC1:

.string "%d, "

.zero 3

.LC2:

.string "%d"

.zero 1

.LC3:

.string "1"

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

* **reverse.s**

.file "reverse.c"

.option nopic

.attribute arch, "rv32i2p0\_c2p0"

.attribute unaligned\_access, 0

.attribute stack\_align, 16

.text

.align 1

.globl reverse

.type reverse, @function

reverse:

srli a6,a1,31

add a6,a6,a1

srai a6,a6,1

li a5,-1

blt a1,a5,.L1

mv a5,a0

slli a1,a1,2

add a0,a0,a1

li a4,0

.L3:

lw a3,0(a5)

addi a4,a4,1

lw a2,-4(a0)

sw a2,0(a5)

sw a3,-4(a0)

addi a5,a5,4

addi a0,a0,-4

ble a4,a6,.L3

.L1:

ret

.size reverse, .-reverse

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

**3.4. Ассемблирование**

Ассемблирование файлов *main.s* и *reverse.s* в *main.o* и *reverse.o* соответственно выполняется по следующей команде:

****

Здесь «-c» – указание остановить процесс сборки после ассемблирования.

Объектный файл не является текстовым. Чтобы посмотреть содержимое *main.o* необходимо вызвать команду:

****

Результат её выполнения:

Изображение выглядит как текст

Автоматически созданное описание

Проведем аналогичную операцию с файлом *reverse.o:*

Изображение выглядит как текст

Автоматически созданное описание

В файлах *main.o* и *reverse.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;

.riscv.attributes – информация про RISC-V;

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.

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

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

Изучим таблицы файлов *main.c* и *reverse.c:*

Вызов таблицы для файла *main.o:*



Таблица:

Изображение выглядит как текст

Автоматически созданное описание

Аналогично вызовем таблицу для файла *reverse.o*:

Изображение выглядит как текст

Автоматически созданное описание

В каждой таблице только одни глобальный (флаг “g”) символ типа «функция» (“F”) – *main* и *reverse* соответственно. (Остальное не обнаружилось и заполнено 0).

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

Вызов:



Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Вызов:



Изображение выглядит как текст

Автоматически созданное описание

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

Аналогично проверим другие секции.

Секции .data не содержат данных:





Секции .bss не содержат данных:





Секции .comment идентичны для обоих файлов и как указывалось выше содержат информацию о версии GCC от SiFive:

Изображение выглядит как текст

Автоматически созданное описание

Секции также идентичны и содержат информацию об используемой архитектуре команд RV32I:

Изображение выглядит как текст

Автоматически созданное описание

Далее проанализируем таблицы перемещений:



Для файла *reverse.o*:

Изображение выглядит как текст

Автоматически созданное описание

И для файла *main.o*:

Изображение выглядит как стол

Автоматически созданное описание

В таблицах перемещений содержится информация о необходимых заменах адресов, которая будет передана компоновщику. Видно, что в файле *main.o* есть информация о необходимости замены для всех внешних функций, а в файле *reverse.o* содержится информация о необходимости подстановки адресов для возвратов в циклах.

**3.5. Компоновка**

Компоновка:



Изучим содержимое секции .text полученного в результате компоновки программы исполняемого файла, для этого получим файл *main.ds*:

****

Нас интересуют некоторые секции этого файла:

- - - - - - - - - - - - - - - - - - - - - - - - - - - - -

00010144 <main>:

10144: 7179 c.addi16sp sp,-48

10146: d606 c.swsp ra,44(sp)

10148: d422 c.swsp s0,40(sp)

1014a: d226 c.swsp s1,36(sp)

1014c: d04a c.swsp s2,32(sp)

1014e: 0002f7b7 lui a5,0x2f

10152: eb878793 addi a5,a5,-328 # 2eeb8 <\_\_clzsi2+0x4c>

10156: 0007a803 lw a6,0(a5)

1015a: 43c8 c.lw a0,4(a5)

1015c: 478c c.lw a1,8(a5)

1015e: 47d0 c.lw a2,12(a5)

10160: 4b94 c.lw a3,16(a5)

10162: 4bd8 c.lw a4,20(a5)

10164: 4f9c c.lw a5,24(a5)

10166: c242 c.swsp a6,4(sp)

10168: c42a c.swsp a0,8(sp)

1016a: c62e c.swsp a1,12(sp)

1016c: c832 c.swsp a2,16(sp)

1016e: ca36 c.swsp a3,20(sp)

10170: cc3a c.swsp a4,24(sp)

10172: ce3e c.swsp a5,28(sp)

10174: 459d c.li a1,7

10176: 0048 c.addi4spn a0,sp,4

10178: 283d c.jal 101b6 <reverse>

1017a: 0040 c.addi4spn s0,sp,4

1017c: 01c10913 addi s2,sp,28

10180: 0002f4b7 lui s1,0x2f

10184: 400c c.lw a1,0(s0)

10186: ed448513 addi a0,s1,-300 # 2eed4 <\_\_clzsi2+0x68>

1018a: 2c6d c.jal 10444 <printf>

1018c: 0411 c.addi s0,4

1018e: ff241be3 bne s0,s2,10184 <main+0x40>

10192: 45f2 c.lwsp a1,28(sp)

10194: 0002f537 lui a0,0x2f

10198: edc50513 addi a0,a0,-292 # 2eedc <\_\_clzsi2+0x70>

1019c: 2465 c.jal 10444 <printf>

1019e: 0002f537 lui a0,0x2f

101a2: ee050513 addi a0,a0,-288 # 2eee0 <\_\_clzsi2+0x74>

101a6: 2ccd c.jal 10498 <scanf>

101a8: 4501 c.li a0,0

101aa: 50b2 c.lwsp ra,44(sp)

101ac: 5422 c.lwsp s0,40(sp)

101ae: 5492 c.lwsp s1,36(sp)

101b0: 5902 c.lwsp s2,32(sp)

101b2: 6145 c.addi16sp sp,48

101b4: 8082 c.jr ra

000101b6 <reverse>:

101b6: 01f5d813 srli a6,a1,0x1f

101ba: 982e c.add a6,a1

101bc: 40185813 srai a6,a6,0x1

101c0: 57fd c.li a5,-1

101c2: 02f5c163 blt a1,a5,101e4 <reverse+0x2e>

101c6: 87aa c.mv a5,a0

101c8: 058a c.slli a1,0x2

101ca: 952e c.add a0,a1

101cc: 4701 c.li a4,0

101ce: 4394 c.lw a3,0(a5)

101d0: 0705 c.addi a4,1

101d2: ffc52603 lw a2,-4(a0)

101d6: c390 c.sw a2,0(a5)

101d8: fed52e23 sw a3,-4(a0)

101dc: 0791 c.addi a5,4

101de: 1571 c.addi a0,-4

101e0: fee857e3 bge a6,a4,101ce <reverse+0x18>

101e4: 8082 c.jr ra

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Компоновщик все переходы auipc+jalr, заменил на одну инструкцию jal и корректным адресом перехода. Например, строка

10178: 283d c.jal 101b6 <reverse>

дает переход на

101b6: 01f5d813 srli a6,a1,0x1f

Проанализируем таблицу перемещений полученного файла (*main*):



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

**3.6 Формирование статической библиотеки**

Статическая библиотека, по сути, является архивом (набором, коллекцией) объектных файлов, оперируемых компоновщиком.

Поместим объектный файл *reverse.o* в статическую библиотеку lib:









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

Посмотрим таблицу символов исполняемого файла:









Можно заметить, что в состав программы вошло содержимое объектных файлов *reverse.o* и *main.o*.

Теперь создадим make-файлы (makefile).

*Makefile-1.txt*

lib.a: reverse.o reverse.h

riscv64-unknown-elf-ar -rsc lib.a reverse.o

reverse.o: reverse.c

riscv64-unknown-elf-gcc.exe -march=rv32i -mabi=ilp32 -O1 -c reverse.c -o reverse.o

*Makefile-2.txt*

all:

mingw32-make -f Makefile-1.txt

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

del \*.o, \*.i, \*s

Запуск make-файла *Makefile-2.txt*:



Изображение выглядит как текст

Автоматически созданное описание

1. **Вывод**

В ходе выполнения лабораторной работы были получены навыки в сфере языка C, ассемблера RISC-V, получены навыки работы с препроцессором, компилятором, ассемблером и компоновщиком пакета GCC и драйвером компилятора riscv64-unknown-elf-gcc.

Была реализована поставленная задача – «реверс массива чисел».

Была собрана программа «по шагам». Были изучены особенности каждого этапа пошаговой сборки набора программ, а также инструменты, позволяющие выделить разработанные программы в статическую библиотеку и автоматизировать сборку этой библиотеки с помощью make-файлов.