Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Лабораторная Работа №4**

**Дисциплина:** Низкоуровневое программирование

**Тема:** Раздельная компиляция

Выполнил

студент гр. 3530901/90003 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Иванов Т.Р.

(подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Алексюк А.О.

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Санкт-Петербург   
2021

**ОГЛАВЛЕНИЕ**

[ТЕХНИЧЕСКОЕ ЗАДАНИЕ 3](#_Toc58595498)

[МЕТОД РЕШЕНИЯ 3](#_Toc58595499)

ПРОГРАММА НА C [4](#_Toc58595499)

СБОРКА ПРОГРАММЫ «ПО ШАГАМ» [5](#_Toc58595499)

1. Препроцессирование  [5](#_Toc58595499)
2. [Компиляция 6](#_Toc58595499)
3. Ассемблирование [10](#_Toc58595499)
4. [Компоновка 16](#_Toc58595499)

СОЗДАНИЕ СТАТИЧЕСКОЙ БИБЛИОТЕКИ [19](#_Toc58595499)

ВЫВОД 20

**ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

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 её тестовой программы. Проанализировать ход сборки библиотеки и программы, созданные файлы зависимостей.

**Вариант 15:**

В массиве поменять местами максимальный и минимальный элементы

**МЕТОД РЕШЕНИЯ**

Программа последовательно перебирает все элементы массива, сравнивая их с текущими максимальным и минимальным значениями. До работы с циклом как условный максимум и минимум принимается значение первого элемента массива. Дальше идет проверка, если текущий элемент массива меньше минимума, то присваиваем минимуму значение этого элемента и запоминаем его адрес. Аналогично с максимумом, если текущий элемент больше текущего максимума, то присваиваем максимуму значение этого элемента и запоминаем его адрес. После обработки всех элементов массива по адресу минимального элемента записываем значение максимального, а по адресу максимального – значение минимального.

**ПРОГРАММА НА C**

На листинге 1 представлен заголовочный файл *swap\_min\_max.h*, на листинге 2 – основной файл *swap\_min\_max.с*, на листинге 3 – тестовая программа *test.c*.

|  |
| --- |
| #ifndef C\_SWAP\_MIN\_MAX\_H  #define C\_SWAP\_MIN\_MAX\_H  #include <stdio.h>  **void** **swapMinMax**(**int** \*arr, **size\_t** arr\_size);  #endif //C\_SWAP\_MIN\_MAX\_H |

**Листинг 1.** *swap\_min\_max.h*

|  |
| --- |
| #include "swap\_min\_max.h"  **void** **swapMinMax**(**int** \*arr, **size\_t** arr\_size) {  **int** min = arr[**0**];  **int** max = arr[**0**];  **int** minAddress = **0**;  **int** maxAddress = **0**;  **for** (**size\_t** i = **0**; i < arr\_size; i++) {  **if** (arr[i] < min) {  min = arr[i];  minAddress = i;  } **else** **if** (arr[i] > max) {  max = arr[i];  maxAddress = i;  }  }  arr[minAddress] = max;  arr[maxAddress] = min;  } |

**Листинг 2.** *swap\_min\_max.с*

|  |
| --- |
| #include "swap\_min\_max.h"  **static** **void** **printArray**(**const** **int** \*arr, **size\_t** size) {  printf("[");  **for** (**size\_t** i = **0**; i < size; i++) {  printf("%d", arr[i]);  **if** (i != size - **1**) {  printf(", ");  }  }  printf("]**\n**");  }  **int** **main**() {  **int** array[] = {**2**, **4**, **3**, **4**, **10**, **6**, **8**, **7**, **9**, **11**};  **const** **size\_t** array\_length = **sizeof**(array) / **sizeof**(array[**0**]); |

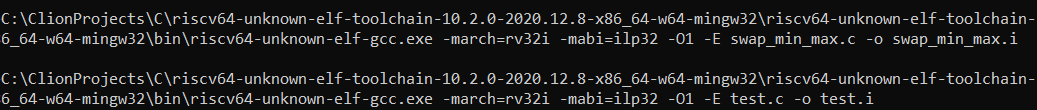
|  |
| --- |
| printf("Initial array: ");  printArray(array, array\_length);  swapMinMax(array, array\_length);  printf("Result array: ");  printArray(array, array\_length);  **return** **0**;  } |

**Листинг 3.** *test.с*

**СБОРКА ПРОГРАММЫ «ПО ШАГАМ»**

**1. Препроцессирование**

Чтобы остановить процесс сборки после препроцессирования, необходимо выполнить следующие команды (рис. 1):

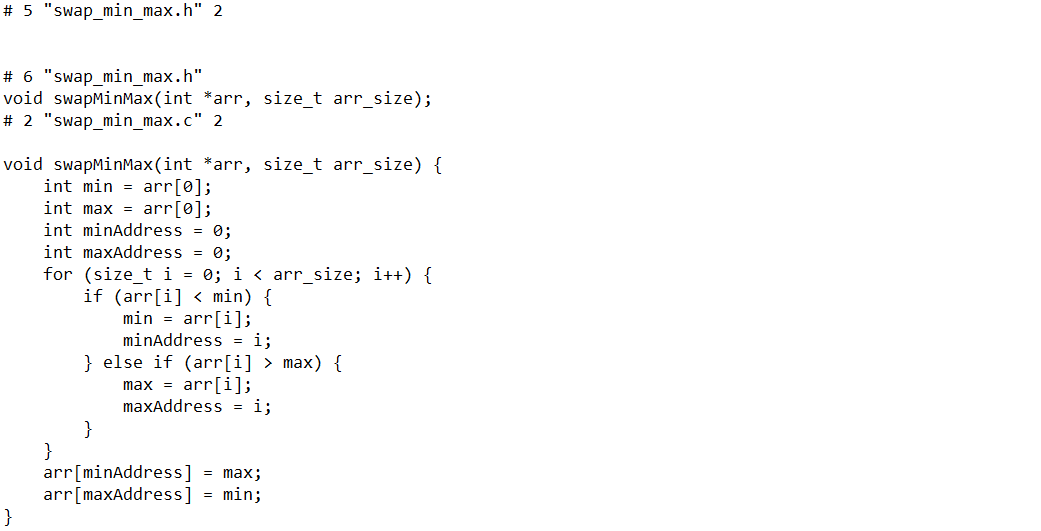


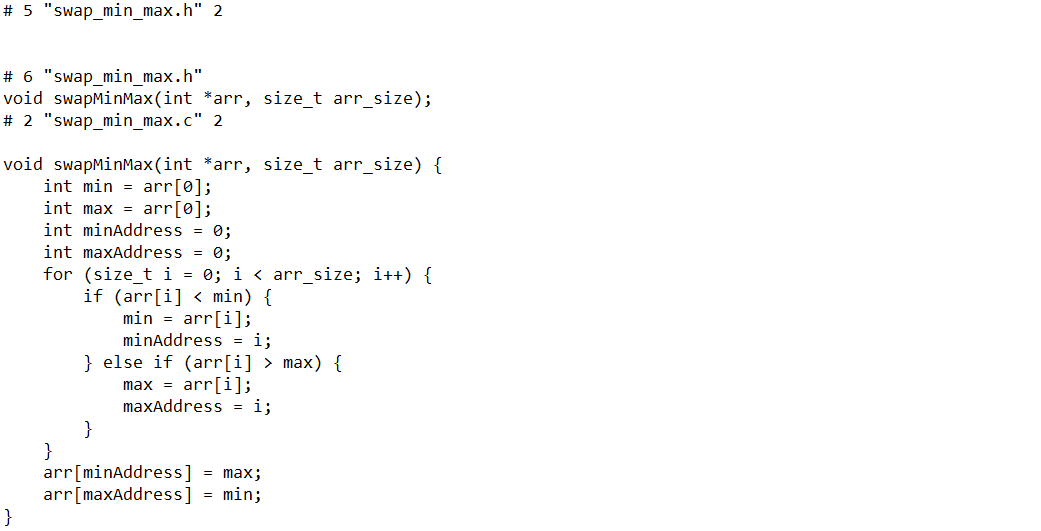
**Рис.1.** Команды для выполнения препроцессирования

Результат препроцессирования был записан в файлы *swap\_min\_max.i* и *test.i.* Из-за того, что была использована директива #include <stdio.h>, эти файлы содержат большое количество дополнительных строк, необходимых для компиляции. Объём файлов составляет 1301 и 1309 строк соответственно. На рисунках 2-3 приведены фрагменты указанных файлов, соответствующие исходным программам:

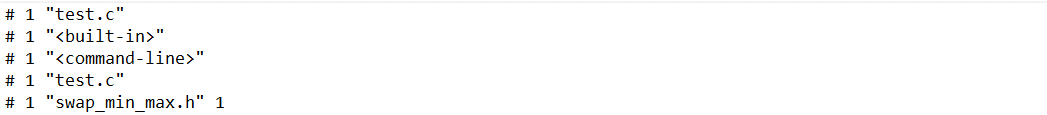


…

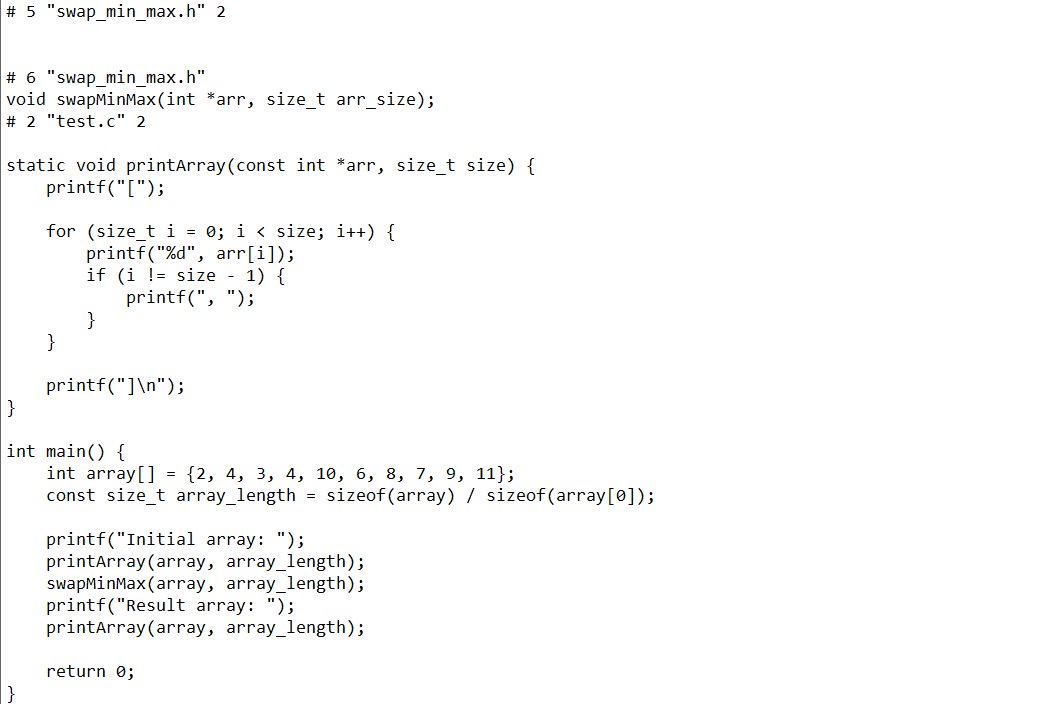




**Рис.2.** *swap\_min\_max.i* (фрагменты)



…



**Рис.3.** *test.i* (фрагменты)

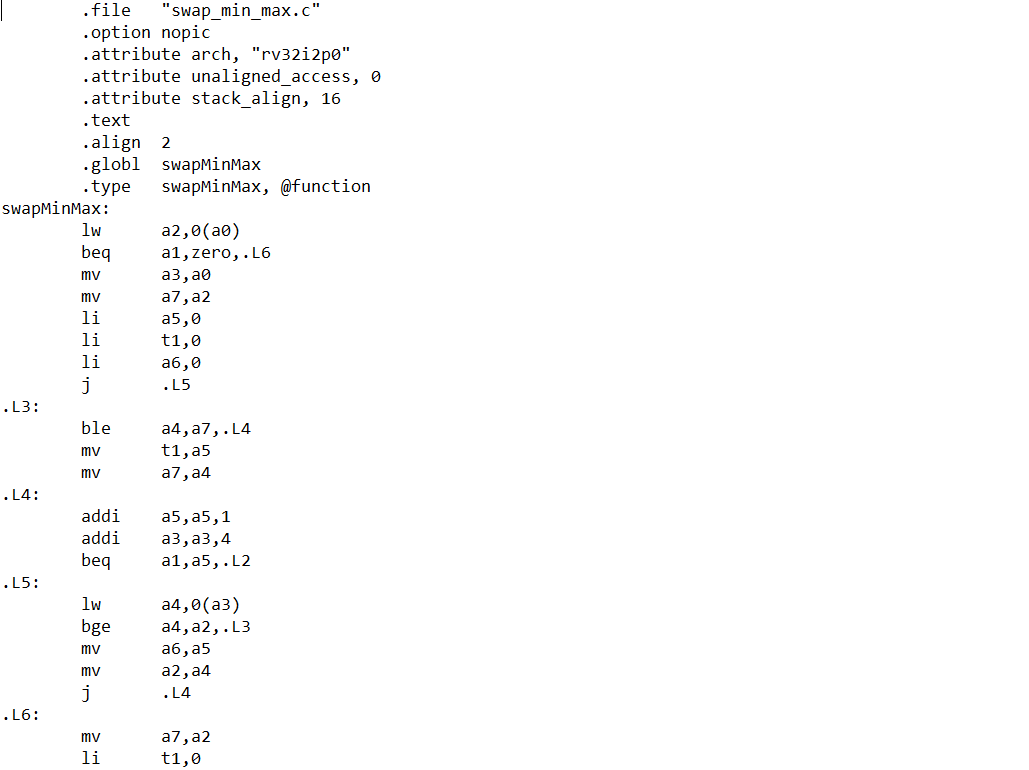
**2. Компиляция**

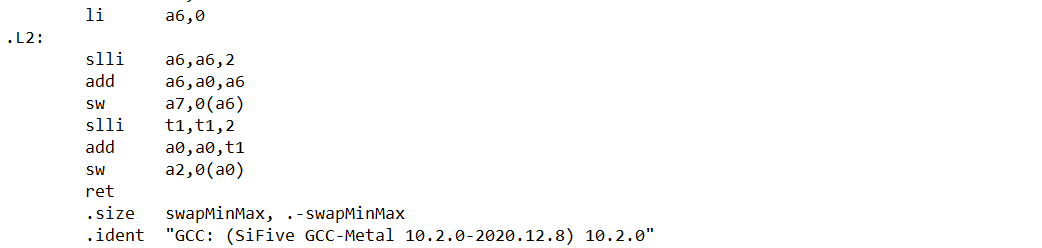
Чтобы остановить процесс сборки после компиляции, необходимо выполнить следующие команды (рис. 4):



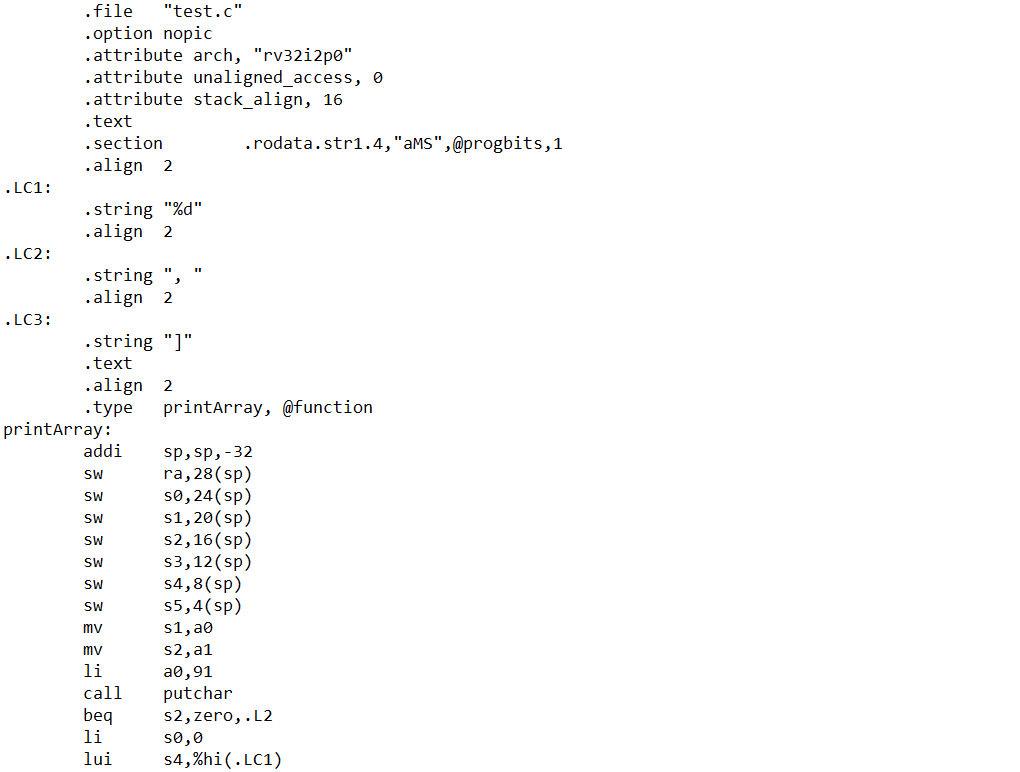
**Рис.4.** Команды для выполнения компиляции

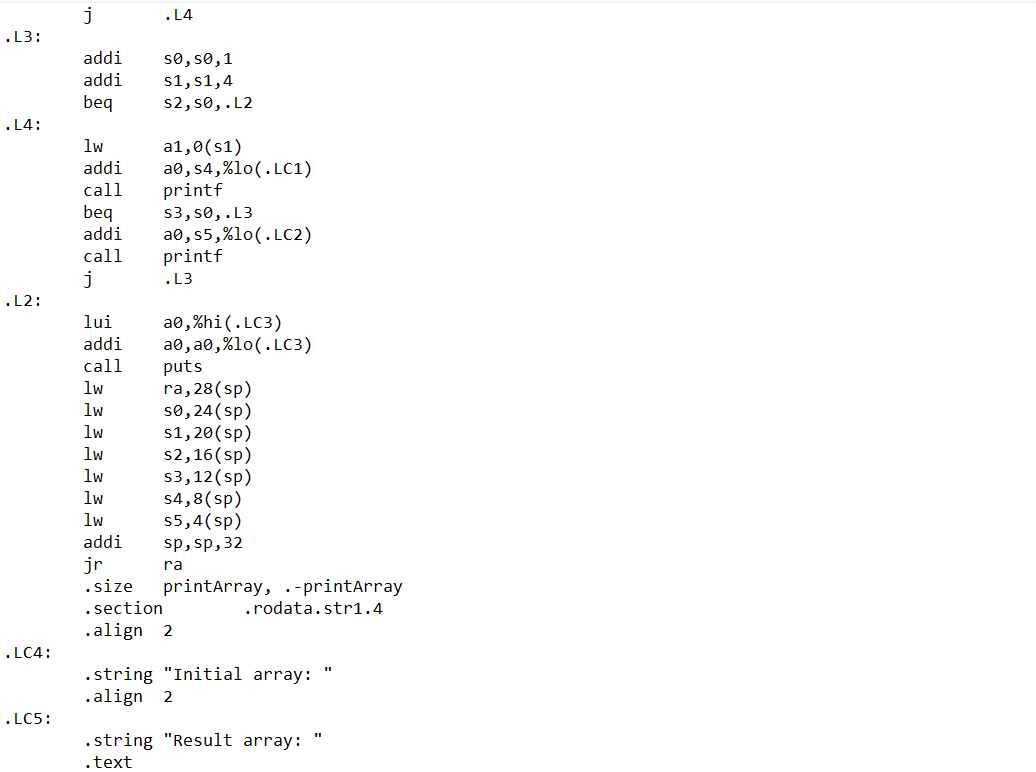
В результате компилятор сформировал файлы *swap\_min\_max.s* и *test.s*, содержащие код на языке ассемблера RISC-V. В *test.s* (рис. 6) можно видеть, как вызывается подпрограмма *swapMinMax*. Текст подпрограммы представлен на рисунке 5:

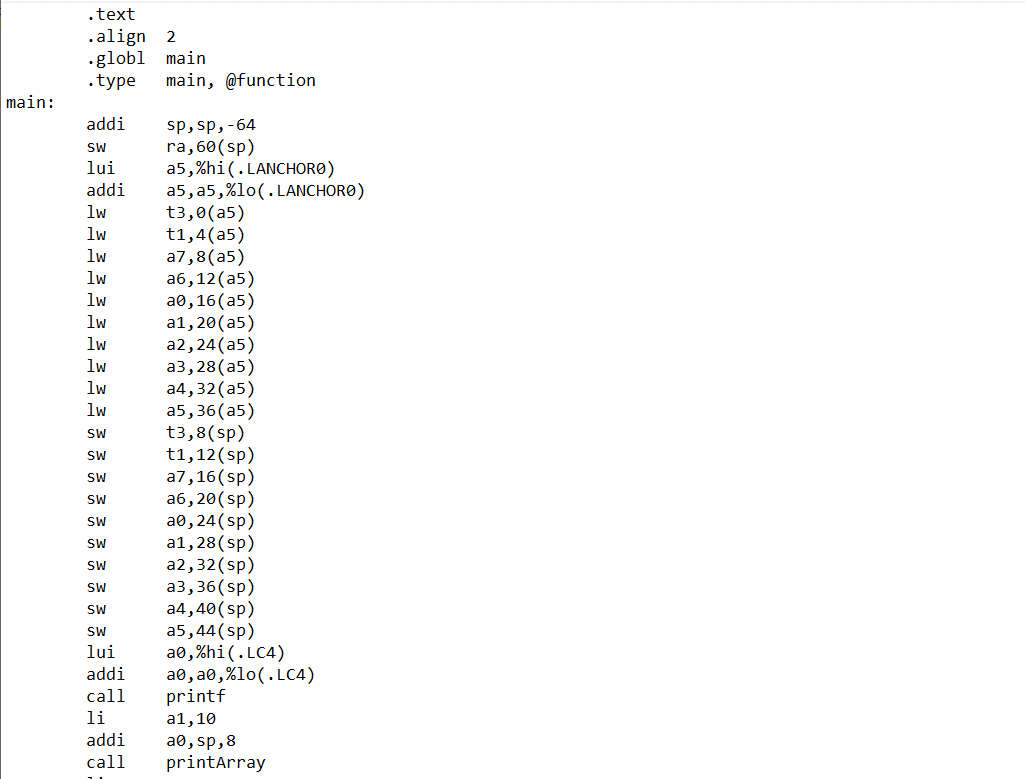
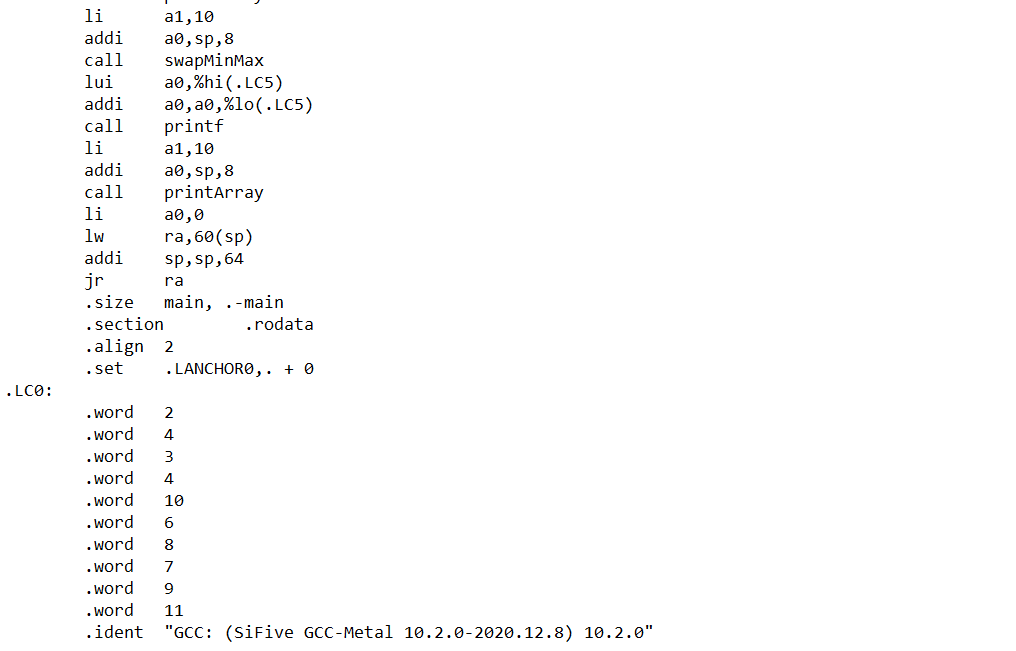




**Рис.5.** *swap\_min\_max.s*



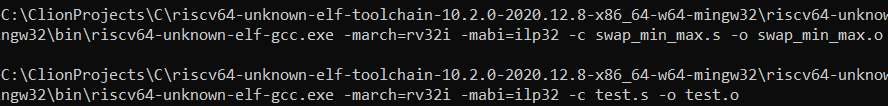


**Рис.6.** *test.s*

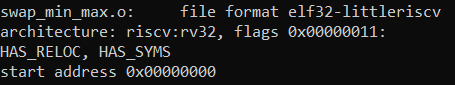
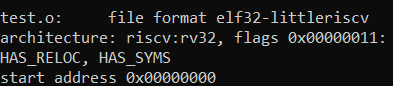
**3. Ассемблирование**

Чтобы остановить процесс сборки после ассемблирования, необходимо выполнить следующие команды (рис. 7):



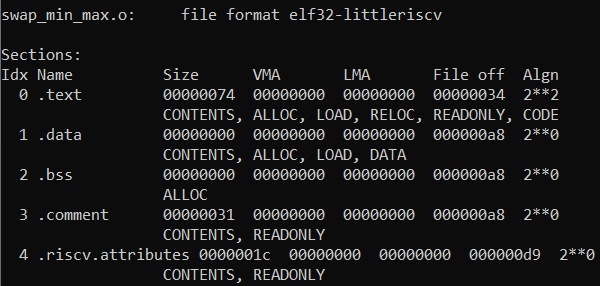
**Рис.7.** Команды для выполнения ассемблирования

Ассемблером сформированы два объектных файла: *swap\_min\_max.o* и *test.o*. Содержимое заголовков файлов представлено на рис. 8–9. Файлы имеют формат ELF, являются объектными файлами 32-разрядной архитектуры RISC-V, содержат символы (флаг HAS\_SYMS) и таблицу перемещений (флаг HAS\_RELOC).

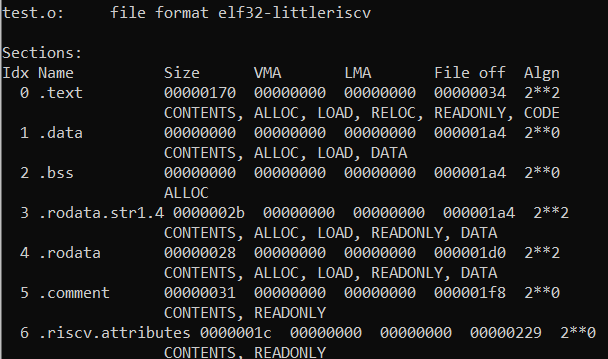
 

**Рис.8.** Заголовок *swap\_min\_max.o* **Рис.9.** Заголовок *test.o*

Заголовки секций файлов представлены на рис. 10–11:

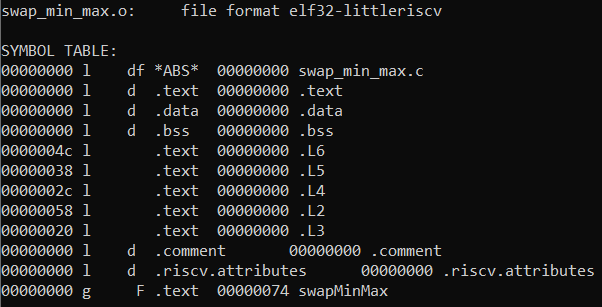


**Рис.10.** Заголовки секций в *swap\_min\_max.o*

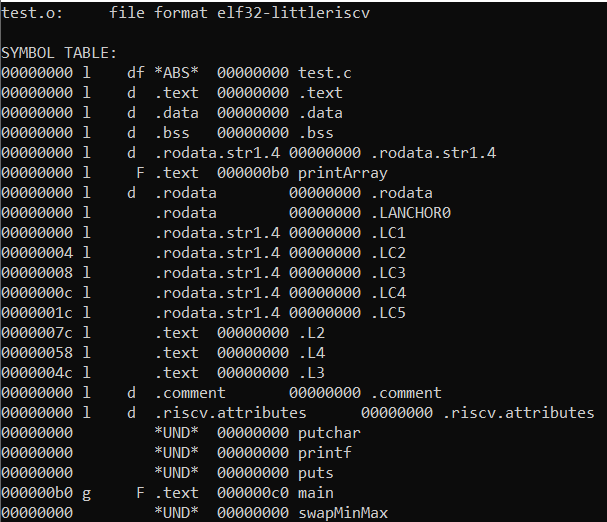


**Рис.11.** Заголовки секций в *test.o*

Таблицы символов представлены на рис. 12–13. Таблица символов *swap\_min\_max.o* содержит одну глобальную (флаг g) функцию (флаг F) — *swapMinMax.* Таблица символов *test.o* содержит одну глобальную функцию — *main* и четыре неопределённых (\*UND\*, использованных, но не определённых в ассемблерном коде) символа — *swapMinMax*, *printf*, *putchar* и *puts*.

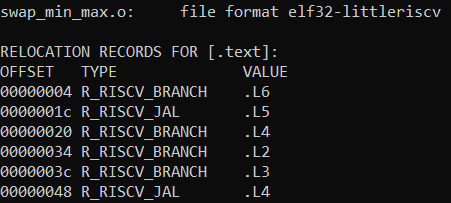


**Рис.12.** Таблица символов *swap\_min\_max.o*

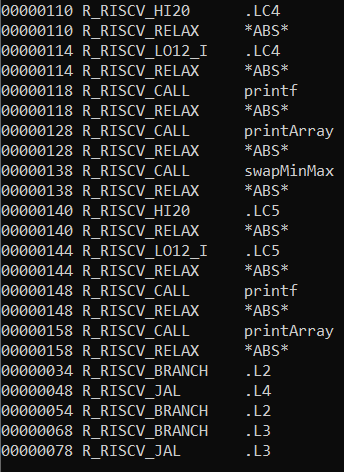
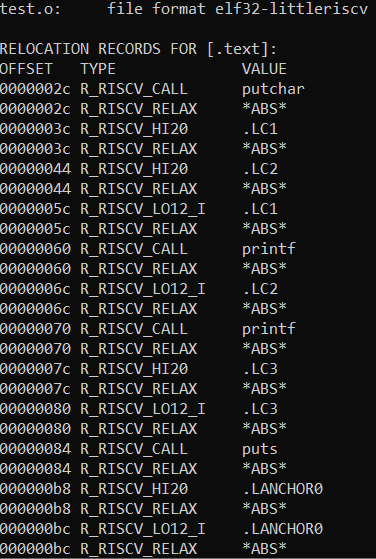


**Рис.13.** Таблица символов *test.o*

Таблицы перемещений представлены на рис. 14–15:

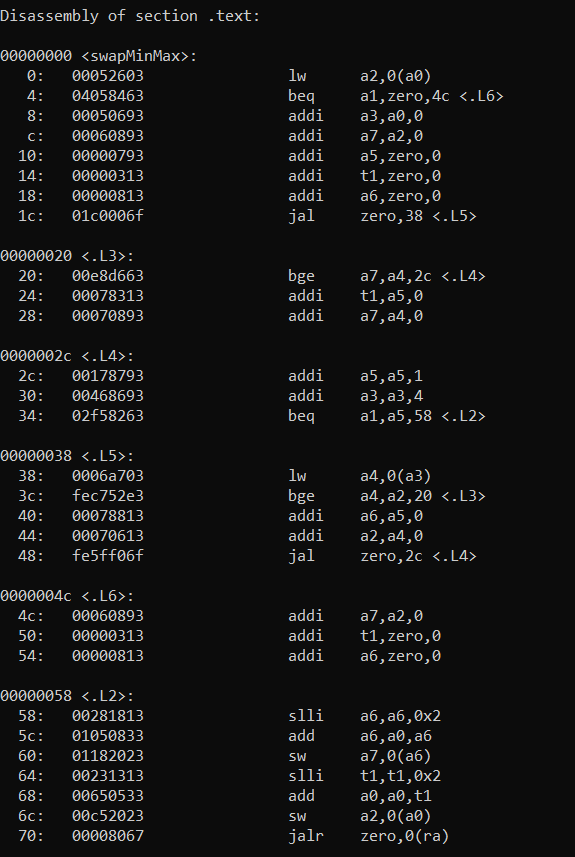


**Рис.14.** Таблица перемещений *swap\_min\_max.o*



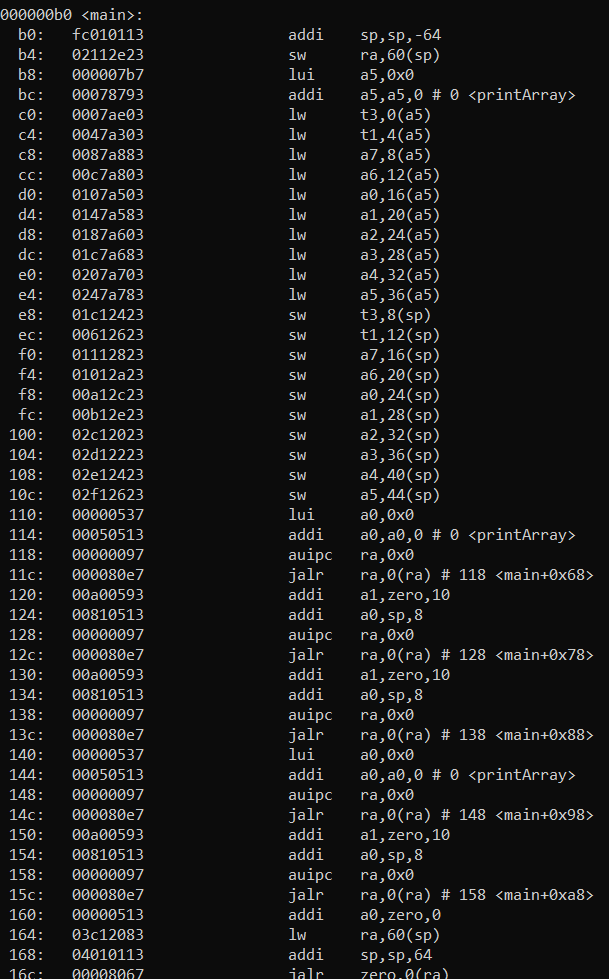
**Рис.15.** Таблица перемещений *test.o*

Дизассемблированное содержимое секции *.text* представлено на рисунках 16‑17. Cодержимое соответствует сформированным компилятором файлам за исключением некоторых инструкций.



**Рис.16.** Содержимое .text в *swap\_min\_max.o*





**Рис.17.** Содержимое .text в *test.o*

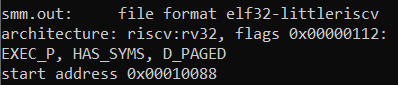
**4.** **Компоновка**

Компоновка выполняется по следующей команде (рис. 18):



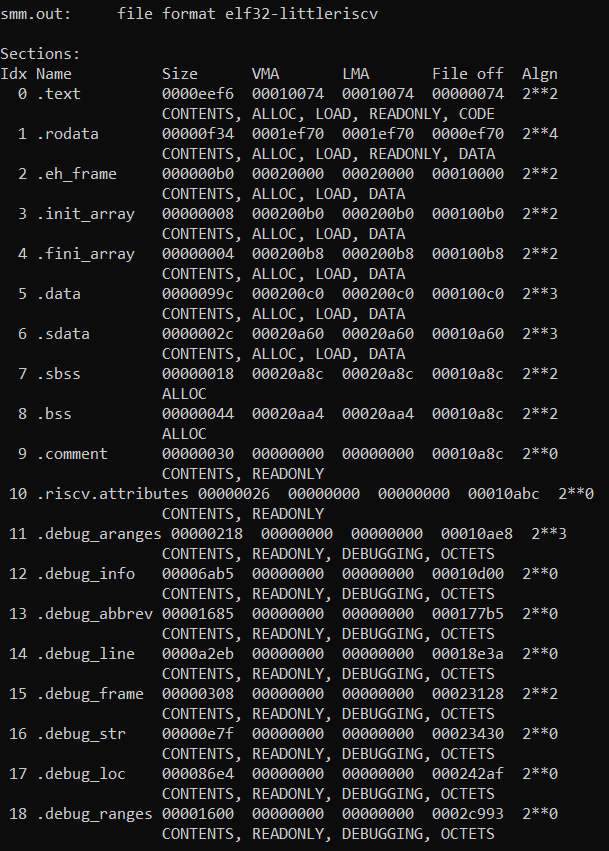
**Рис.18.** Команда для выполнения компоновки

Компоновщик сформировал файл *smm.out*. Содержимое заголовка файла представлено на рис. 19. Флаг EXEC\_P указывает, что файл является исполняемым.



**Рис.19.** Заголовок *smm.out*

Заголовки секций файла представлены на рис. 20. Имеющиеся ранее секции расширены за счёт слияния нескольких объектных файлов. Файл содержит большое количество отладочной информации.



**Рис.20.** Заголовки секций *smm.out*

Дизассемблированное содержимое секции *.text* содержит порядка 20000 строк, поэтому в листинге 4 приведены только фрагмент, связанный с исходной программой.

|  |
| --- |
| ...  00010110 <swapMinMax>:  10110: 00052603 lw a2,0(a0)  10114: 04058463 beq a1,zero,1015c <swapMinMax+0x4c>  10118: 00050693 addi a3,a0,0  1011c: 00060893 addi a7,a2,0  10120: 00000793 addi a5,zero,0  10124: 00000313 addi t1,zero,0  10128: 00000813 addi a6,zero,0  1012c: 01c0006f jal zero,10148 <swapMinMax+0x38>  10130: 00e8d663 bge a7,a4,1013c <swapMinMax+0x2c>  10134: 00078313 addi t1,a5,0  10138: 00070893 addi a7,a4,0  1013c: 00178793 addi a5,a5,1  10140: 00468693 addi a3,a3,4  10144: 02f58263 beq a1,a5,10168 <swapMinMax+0x58>  10148: 0006a703 lw a4,0(a3)  1014c: fec752e3 bge a4,a2,10130 <swapMinMax+0x20>  10150: 00078813 addi a6,a5,0  10154: 00070613 addi a2,a4,0  10158: fe5ff06f jal zero,1013c <swapMinMax+0x2c>  1015c: 00060893 addi a7,a2,0  10160: 00000313 addi t1,zero,0  10164: 00000813 addi a6,zero,0  10168: 00281813 slli a6,a6,0x2  1016c: 01050833 add a6,a0,a6  10170: 01182023 sw a7,0(a6)  10174: 00231313 slli t1,t1,0x2  10178: 00650533 add a0,a0,t1  1017c: 00c52023 sw a2,0(a0)  10180: 00008067 jalr zero,0(ra) # 100de <\_\_do\_global\_dtors\_aux+0x1e>  …  00010224 <main>:  10224: fc010113 addi sp,sp,-64  10228: 02112e23 sw ra,60(sp)  1022c: 0001f7b7 lui a5,0x1f  10230: f9c78793 addi a5,a5,-100 # 1ef9c <\_\_clzsi2+0x6e>  10234: 0007ae03 lw t3,0(a5)  10238: 0047a303 lw t1,4(a5)  1023c: 0087a883 lw a7,8(a5)   1. 10240: 00c7a803 lw a6,12(a5) |
| 10244: 0107a503 lw a0,16(a5)  10248: 0147a583 lw a1,20(a5)  1024c: 0187a603 lw a2,24(a5)  10250: 01c7a683 lw a3,28(a5)  10254: 0207a703 lw a4,32(a5)  10258: 0247a783 lw a5,36(a5)  1025c: 01c12423 sw t3,8(sp)  10260: 00612623 sw t1,12(sp)  10264: 01112823 sw a7,16(sp)  10268: 01012a23 sw a6,20(sp)  1026c: 00a12c23 sw a0,24(sp)  10270: 00b12e23 sw a1,28(sp)  10274: 02c12023 sw a2,32(sp)  10278: 02d12223 sw a3,36(sp)  1027c: 02e12423 sw a4,40(sp)  10280: 02f12623 sw a5,44(sp)  10284: 0001f537 lui a0,0x1f  10288: f7c50513 addi a0,a0,-132 # 1ef7c <\_\_clzsi2+0x4e>  1028c: 194000ef jal ra,10420 <printf>  10290: 00a00593 addi a1,zero,10  10294: 00810513 addi a0,sp,8  10298: eedff0ef jal ra,10184 <printArray>  1029c: 00a00593 addi a1,zero,10  102a0: 00810513 addi a0,sp,8  102a4: e6dff0ef jal ra,10110 <swapMinMax>  102a8: 0001f537 lui a0,0x1f  102ac: f8c50513 addi a0,a0,-116 # 1ef8c <\_\_clzsi2+0x5e>  102b0: 170000ef jal ra,10420 <printf>  102b4: 00a00593 addi a1,zero,10  102b8: 00810513 addi a0,sp,8  102bc: ec9ff0ef jal ra,10184 <printArray>  102c0: 00000513 addi a0,zero,0  102c4: 03c12083 lw ra,60(sp)  102c8: 04010113 addi sp,sp,64  102cc: 00008067 jalr zero,0(ra) |

**Листинг 4.** *smm.ds* (дизассемблированное содержимое *smm.out*, фрагмент)

Таблица символов s*mm.out* содержит *swapMinMax*, *printf*, *putchar* и *puts*, но теперь функции являются определёнными и содержатся по корректным адресам (рис. 21).





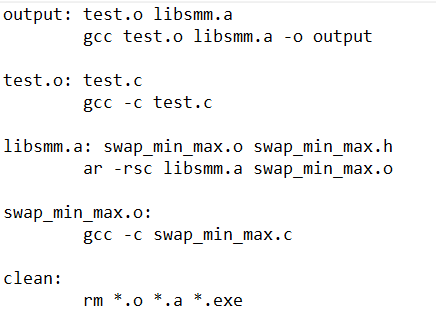




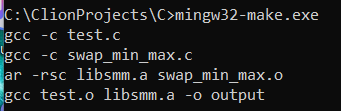
**Рис.21.** Фрагмент таблицы символов *smm.out*

**СОЗДАНИЕ СТАТИЧЕСКОЙ БИБЛИОТЕКИ**

Для получения статической библиотеки используем утилиту *make*. Содержимое make-файла представлено на рисунке 22. Вывод *make* при использовании файла представлен на рис. 23.

****

**Рис.22.** *Makefile*



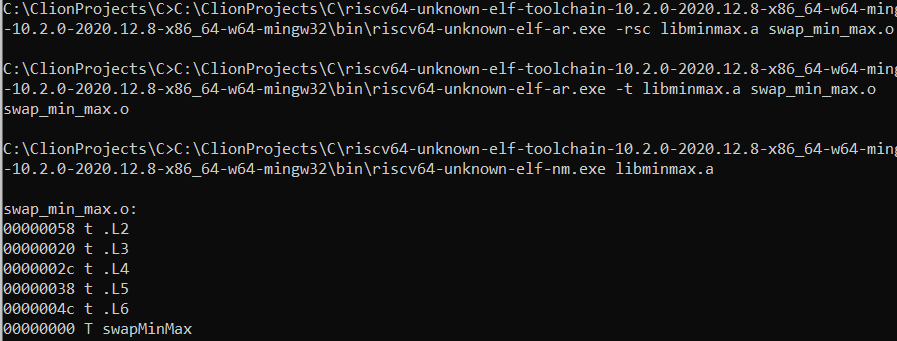
**Рис.23.** Вывод*make*

Проверка содержимого полученного файла архива (рис. 24):



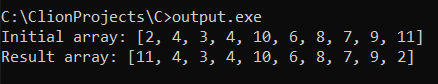
**Рис.24.** Содержимое *libsmm.a*

Список символов библиотеки (рис. 25):



**Рис.25.** Символы *libsmm.a*

Результат запуска тестовой программы соответствует ожидаемому (рис. 26).



**Рис.26.** Запуск программы

**ВЫВОД**

В ходе выполнения лабораторной работы были закреплены знания языка C, ассемблера RISC-V, получены навыки работы с препроцессором, компилятором, ассемблером и компоновщиком пакета GCCи драйвером компилятора riscv64-unknown-elf-gcc. Были изучены особенности каждого этапа пошаговой сборки набора программ, а также инструменты, позволяющие выделить разработанные программы в статическую библиотеку и автоматизировать сборку этой библиотеки.

Была реализована поставленная задача – «в массиве поменять местами максимальный и минимальный элементы», а затем проведена проверка правильности перевода программы решения этой задачи в набор инструкций, выполняемых процессором.