Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Отчёт по лабораторной работе № 4**

Дисциплина: Низкоуровневое программирование

Тема: Раздельная компиляция

Вариант №14

Выполнил студент гр. 3530901/00001 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.И. Пеутина

(подпись)

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.А. Коренев

(подпись)

“ ” 2021 г.

Санкт-Петербург

2021

1. **Описание задачи**
2. На языке C разработать функцию определения наиболее часто встречающегося в массиве значения. Поместить определение функции в отдельный исходный файл, оформить заголовочный файл. Разработать тестовую программу на языке C.
3. 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.
4. 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.
5. **Алгоритм работы**

Для каждого значения находим количество его повторений в массиве с помощью счетчика повторений. Далее сравниваем значение из счетчика с текущим результатом и обновляем результат, если это необходимо.

1. **Реализация программы**

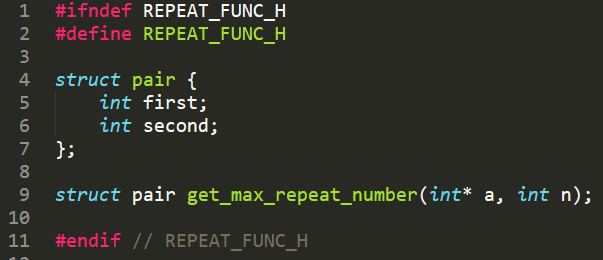


Рис.1 Файл repeat\_func.h

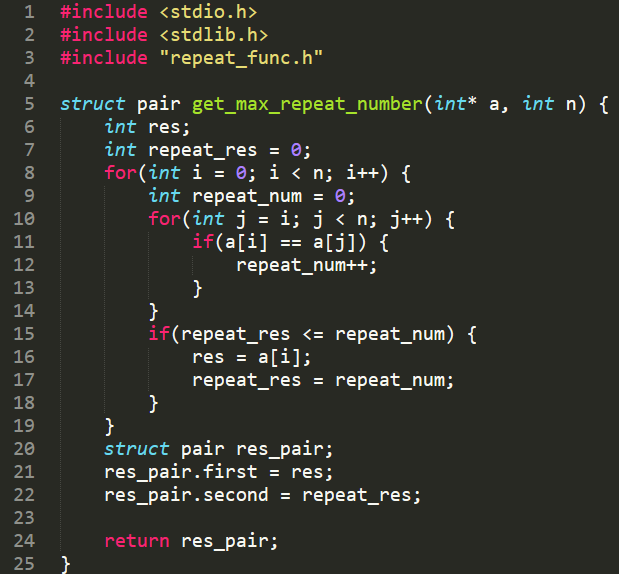


Рис.2 Файл repeat\_func.c

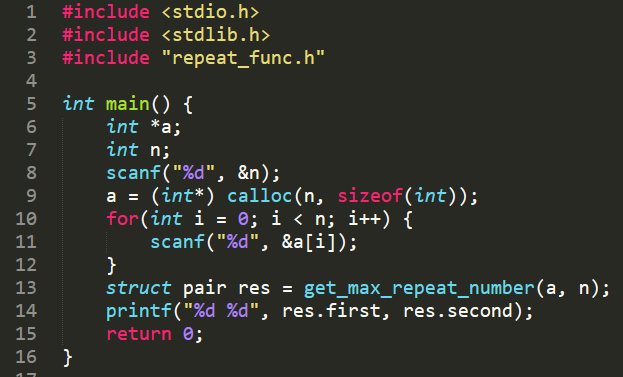


Рис.3 Файл main.c

В файле “repeat\_func.c” реализована функция get\_max\_repeat\_number(), в которую передаётся число n – количество элементов в массиве а и указатель на массив.

Заголовочный файл “repeat\_func.h” содержит в себе определение функции get\_max\_repeat\_number() и структуры pair. В дальнейшем для использования этой функции в другой программе, необходимо организовывать подключение этого заголовочного файла, а также компиляцию исходного файла “ repeat\_func.c”вместе с использующей ее программой.

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

Начнем сборку созданных программ на языке C по шагам. Первым шагом является препроцессирование файлов исходного текста “ get\_max\_repeat\_number.c” и “main.c” в файлы “ get\_max\_repeat\_number.i” и “main.i”.

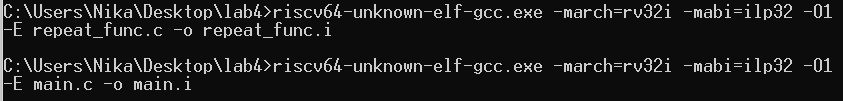


Рис.4 Команды для препроцессирования файлов

Результаты работы препроцессора мало отличаются от исходных версий программ:

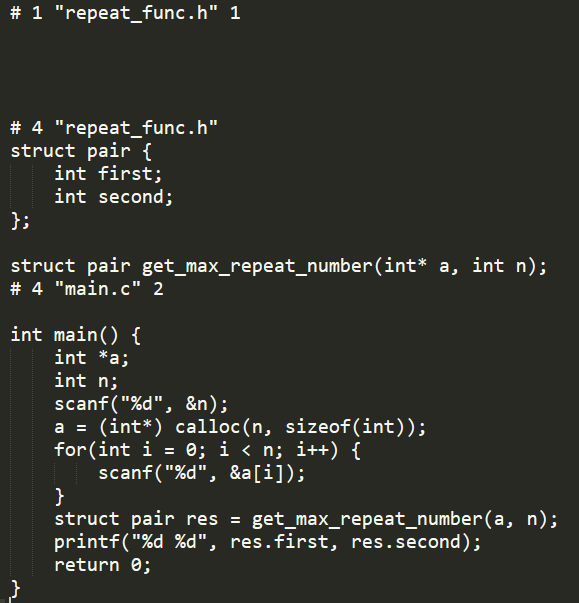


Рис.5 Файл main.i

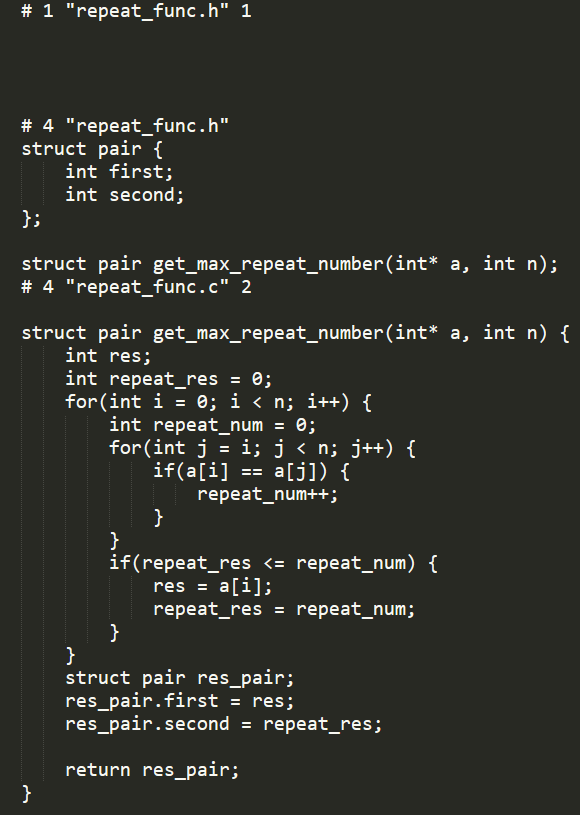


Рис.6 Файл repeat\_func.i

Компиляция — трансляция программы, составленной на исходном языке высокого уровня, в эквивалентную программу на низкоуровневом языке, близком машинному коду (в нашем случае на язык ассемблера RISC-V), выполняемая компилятором.

Для компиляции выполним следующие команды:

1. *riscv64-unknown-elf-gcc.exe -march=rv64i -mabi=lp64 -O1 -S repeat\_func.i -o repeat\_func.s*

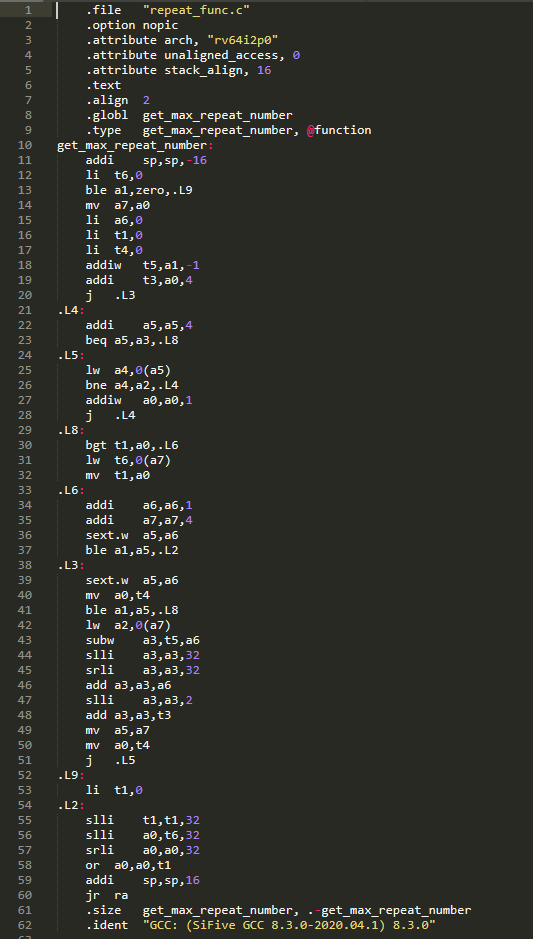


Рис.7 repeat\_func.s

1. *riscv64-unknown-elf-gcc.exe -march=rv64i -mabi=lp64 -O1 -S main.i -o main.s*

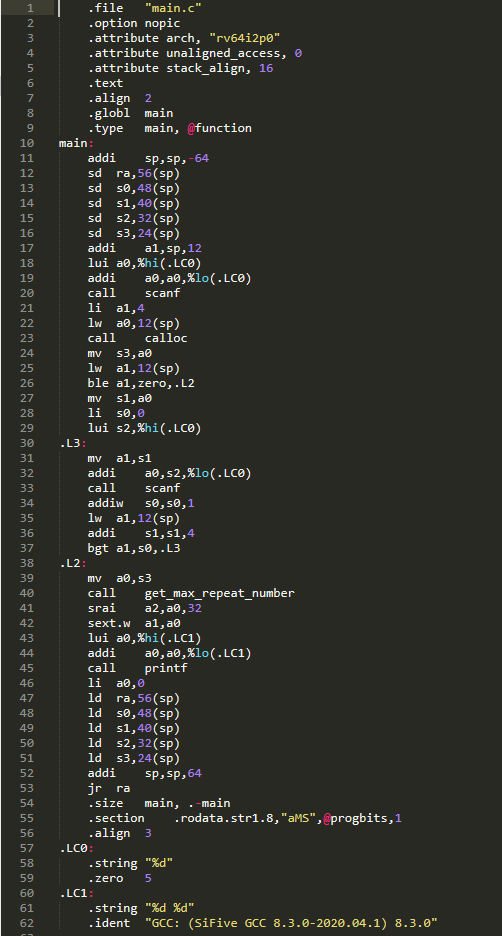


Рис.8 main.s

Произведём ассемблирование полученных после компиляции файлов программы. Для этого выполним следующие команды:





Рис.9 Команды ассемблирования полученных файлов

Для прочтения полученных файлов используем следующие команды:

1. *riscv64-unknown-elf-objdump.exe -h название\_файла.o*
2. *riscv64-unknown-elf-objdump.exe -t название\_файла.o*
3. *riscv64-unknown-elf-objdump.exe -s -j .text название\_файла.o*
4. *riscv64-unknown-elf-objdump.exe -s -j .comment название\_файла.o*
5. *riscv64-unknown-elf-objdump.exe -r название\_файла.o*

На рисунках 7-17 представлены результаты исполнения этих команд

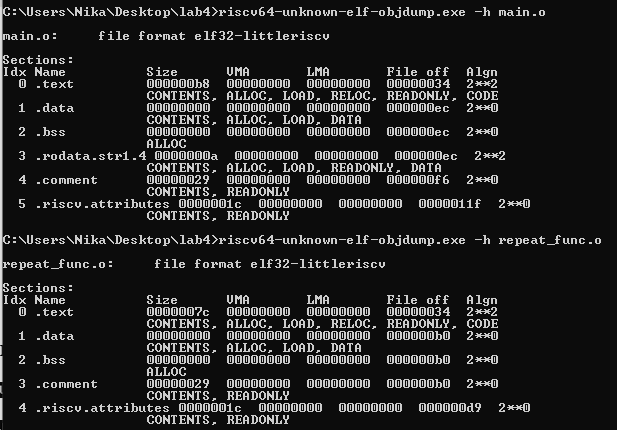


Рис.10 Заголовки секций в main.o

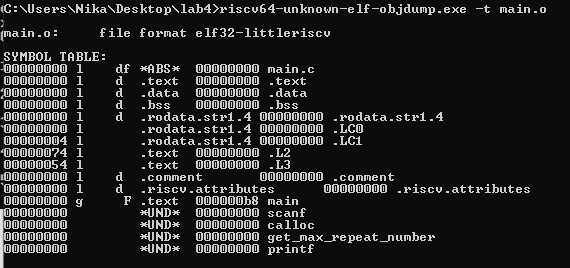
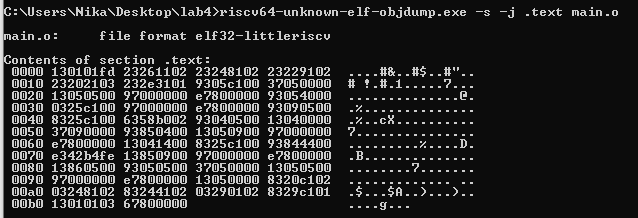


Рис.11 Таблица символов в main.o

 Рис.12 Содержание секции .text в main.o

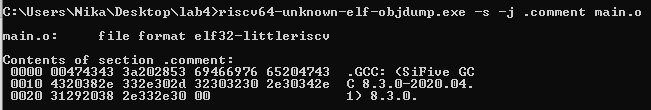


Рис.13 Содержание секции .comment в main.o

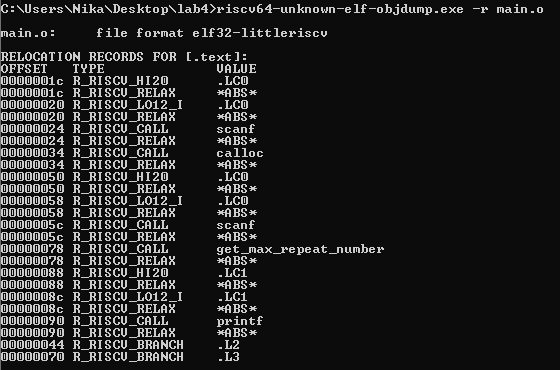


Рис.14 Таблица перемещений для main.o

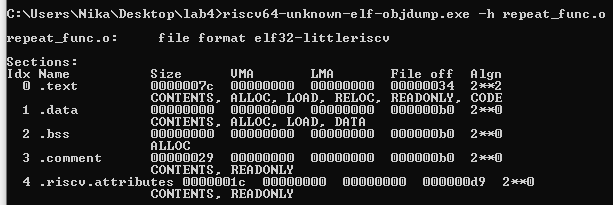


Рис.15 Заголовки секций в repeat\_func.o

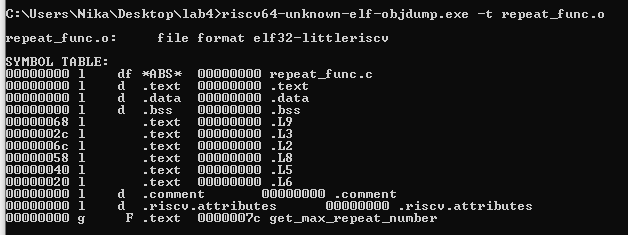


Рис.16 Таблица символов в repeat\_func.o

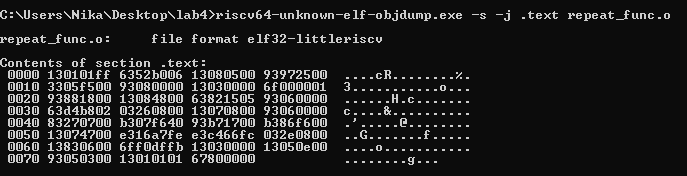


Рис.17 Содержание секции .text в repeat\_func.o

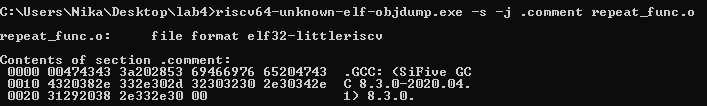


Рис.18 Содержание секции .comment в repeat\_func.o

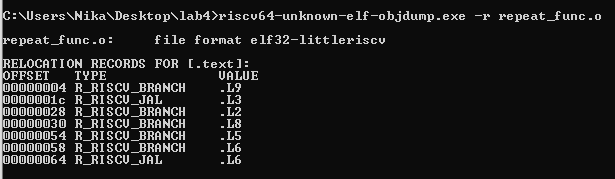


Рис.19 Таблица перемещений для repeat\_func.o

Анализ результатов:

* Изучая таблицу секций, можем определить, какие данные в ней находятся, сколько они запоминают места и в каком коде закодированы;
* Таблица символов содержит один глобальный флаг «g», указывающий на символ типа «функции»;
* Декодировав содержание секции .text мы можем получить код программы, а в .comment значение после .indent в скомпилированном коде;
* Таблица перемещений содержит информацию обо всех неоконченных инструкциях.

Дизассемблируем код с помощью команд:

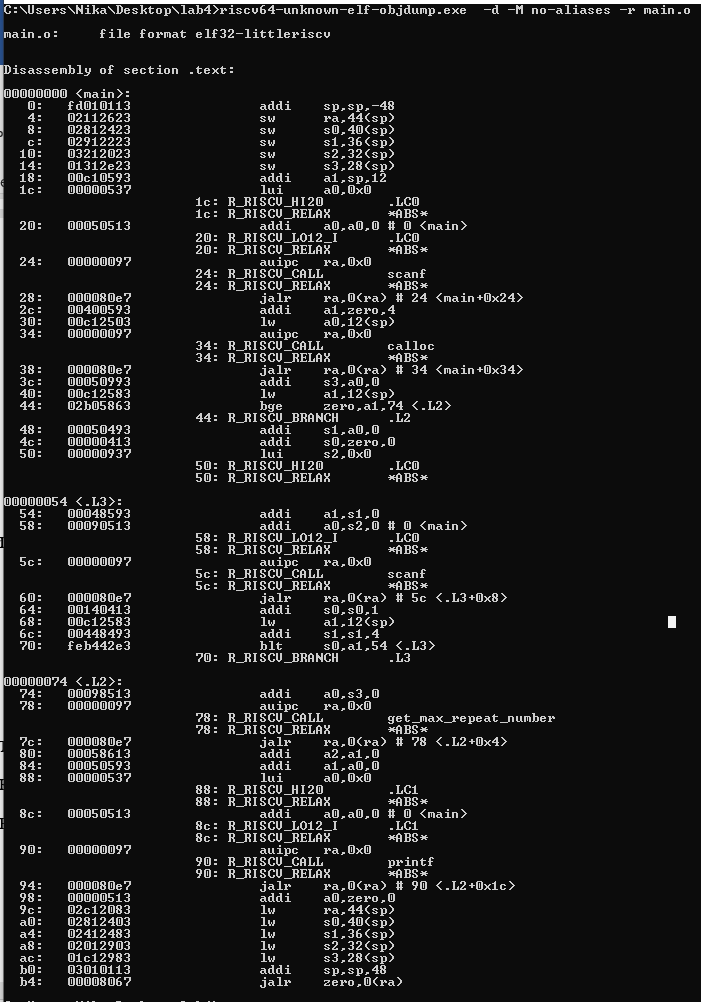


Рис.20 Дизассемблированный код main.o

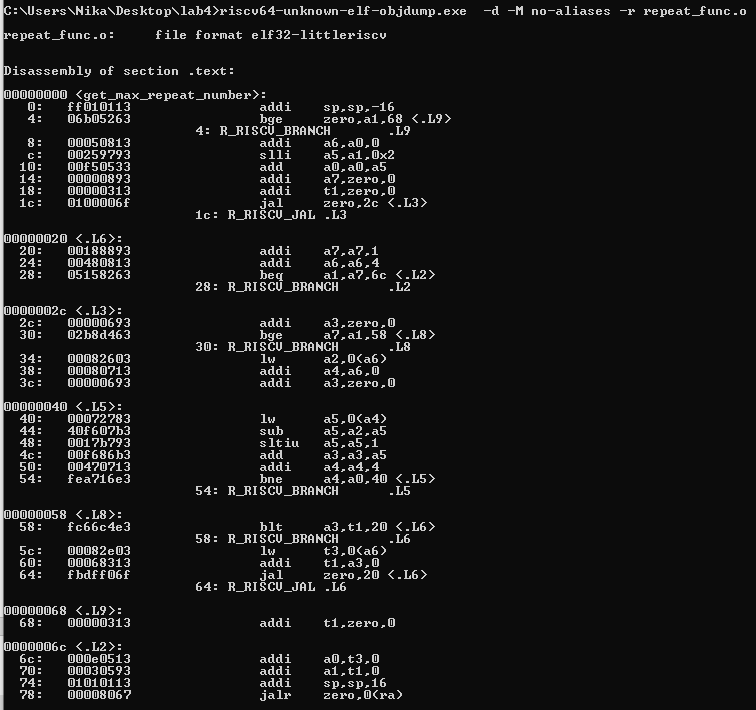


Рис. 21 Дизассемблированный код repeat\_func.o

Далее сделаем компановку.

Редактирование связей (компоновка) – это процесс сборки и объединения различных частей программного кода и данных в единый файл, который может быть загружен в память и запущен на исполнителе.

Скомпонуем программу с помощью команды: *riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -v main.o repeat\_func.o*

Для просмотра получившегося кода воспользуемся следующей командой: riscv64-unknown-elf-objdump.exe –j .text –d –M no-aliases a.out >a.ds

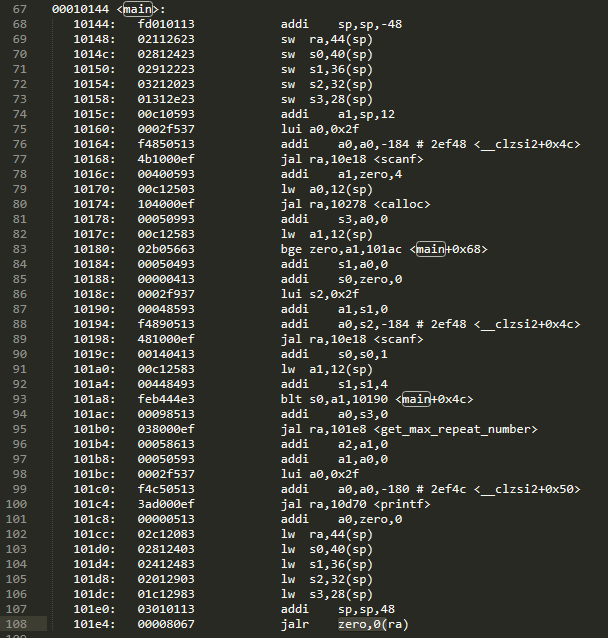


Рис. 22 Фрагмент кода a.ds main

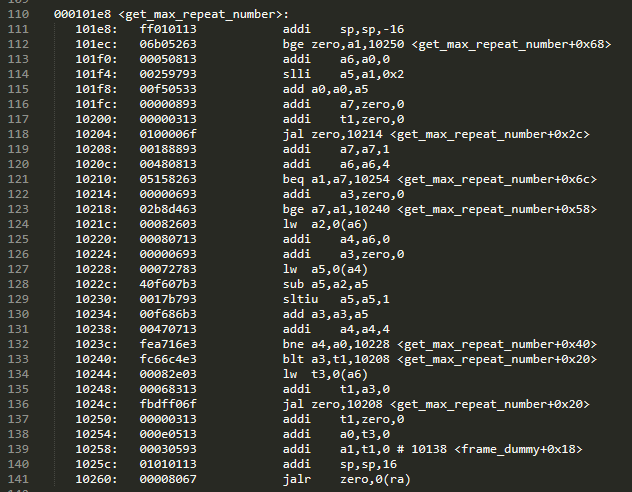


Рис.23 Фрагмент кода из a.ds get\_max\_repeat\_number

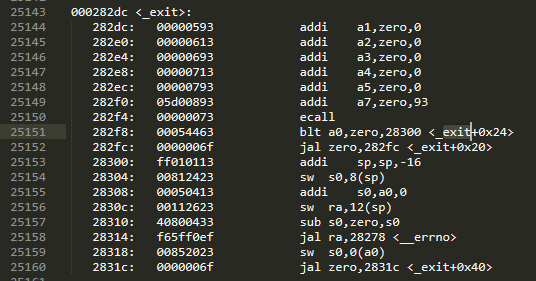


Рис.24 Фрагмент кода из a.ds \_exit

Можем удостовериться в том, что компоновщик все переходы сочетания инструкций auipc и jalr заменил на инструкцию jal и корректным адресом перехода.

Выполнив команду по просмотру таблицы перемещений убедимся, что все необходимые релокации, оптимизации и замены инструкций были успешно проведены компоновщиком (рис. 21).



Рис. 25 Таблица перемещений для a.out.

1. **Процесс создания статической библиотеки, его анализ и создание make-file**

Статическая библиотека – механизм, позволяющий упаковывать связанные объектные модули в единый файл, который может затем быть передан в качестве входа редактору связей.

Получим статическую библиотеку, выполнив следующие последовательности команд:

1. *riscv64-unknown-elf-gcc.exe -march=rv64i -mabi=lp64 -O1 -c StringConverter.i -o StringConverter.o*
2. *riscv64-unknown-elf-ar.exe -rsc staticLib.a StringConverter.o*

Используя получившуюся библиотеку, соберем исполняемый файл программы следующей командой: *riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -O1 --save-temps main.c staticLib.a*

Получим таблицу символов исполняемого файла и убедимся в наличии объектных файлов StringConverter.o с помощью команды: *riscv64-unknown-elf-objdump.exe -t a.out* (рис. 22)

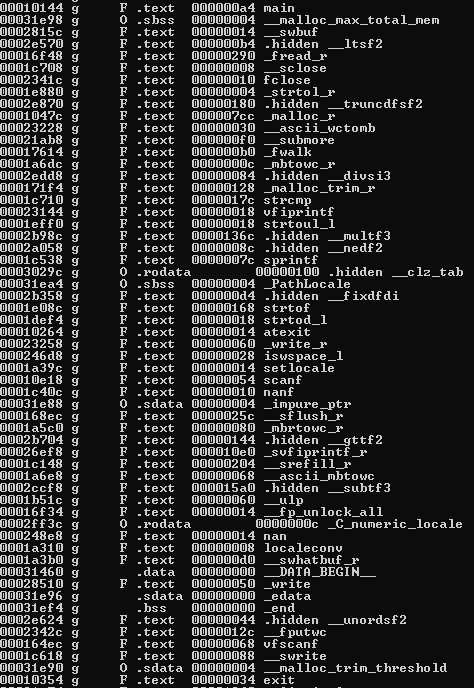


Рис.26 Элементы символьной таблицы a.out

Также создадим make-файл.

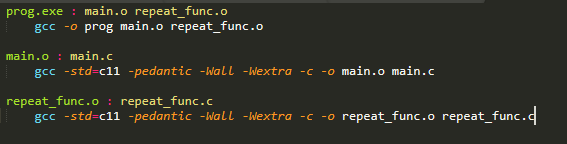


Рис. 27 Make-файл для сборки приложения.



Рис.28 Сборка с помощью Makefile



Рис.29 Тестирование программы

1. **Вывод**

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

Он состоит из:

Препроцессирования: исходного .c файл препроцессируем в .i файл

Компиляции: полученный .iфайл компилируется в файл ассемблера .s

Ассемблирования: файл .s асссемблируется в объектный файл .o

Компоновки: объектный файл .oкомпонуется в исполняемый файл

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

Утилита Make позволяет собирать проекты, состоящие из большого количества файлов, вместо использования PS/SH скриптов, и прописывания файлов вручную.