Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Отчёт по лабораторной работе № 3**

Дисциплина: Низкоуровневое программирование

Тема: Раздельная компиляция

Вариант: 18

Выполнил студент гр. 3530901/90002 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Н.С. Якубец

(подпись)

Принял преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.С. Степанов

(подпись)

“ ” 2021 г.

Санкт-Петербург

2021

**Задание**

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.

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.

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 18: Циклический сдвиг массива чисел.

**Алгоритм**

На каждом шаге своей работы алгоритм замещает элемент el1 на el2. Адрес el2 на каждом шаге считается следующим образом:

el2\_addr = (el1\_addr + num\_of\_shifts ) % array\_length

После замещения в массиве el2 на el1, el2 помещается в регистр, хранящий el1, и, вычисляя адрес нового el2, программа помещает новый el2 в соответствующий регистр. Такие шаги производятся array\_length раз, так как каждый элемент изменил свое положение в результате сдвигов и должен быть заменен в исходном массиве.

**Программа**

Была реализована программа в файлах main.c и reverse.c, циклически сдвигающая указанный массив указанное число раз в указанном направлении. main – основная программа, вызывающая функцию reverse, которая и производит необходимые действия с заданным массивом. Код программы приводится на рис. 1-2.

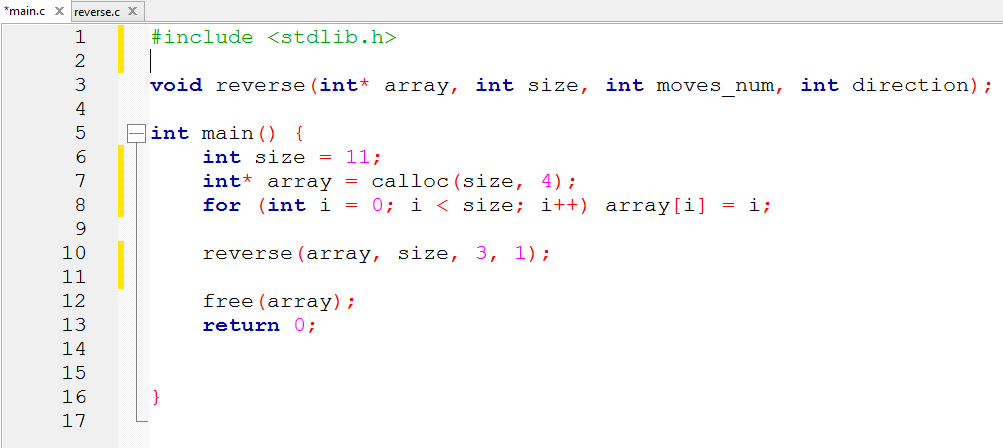


Рис. 1 Исходный код подпрограммы main

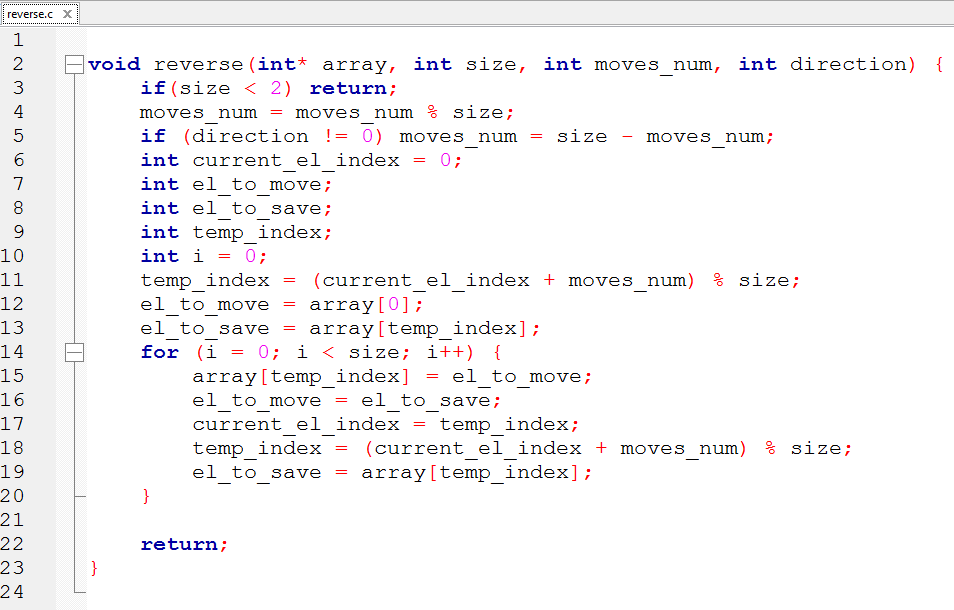


Рис. 2 Исходный код подпрограммы reverse

**Компиляция**

Для запуска процесса компиляции была выполнена следующая команда в командной строке

riscv64-unknown-elf-gcc --save-temps -march=rv32imc -mabi=ilp32 -O1 -v main.c reverse.c >log 2>&1

В результате чего были получены файлы a.out, log, main.o, main.i, main.s reverse.o, reverse.i, reverse.s.

**Препроцессор**

В результате работы препроцессора были получены два файла: main.i и reverse.i. Содержимое файла reverse не сильно изменилось по сравнению с исходным кодом (рис. 3), лишь добавились специальные директивы, необходимые для передаче информации от препроцессора в компилятор.

Содержимое же файла main.i многократно увеличилось по сравнению с исходным кодом. Это произошло в следствие того, что в файле main.c была подключена библиотека stdlib и заголовок reverse.h.

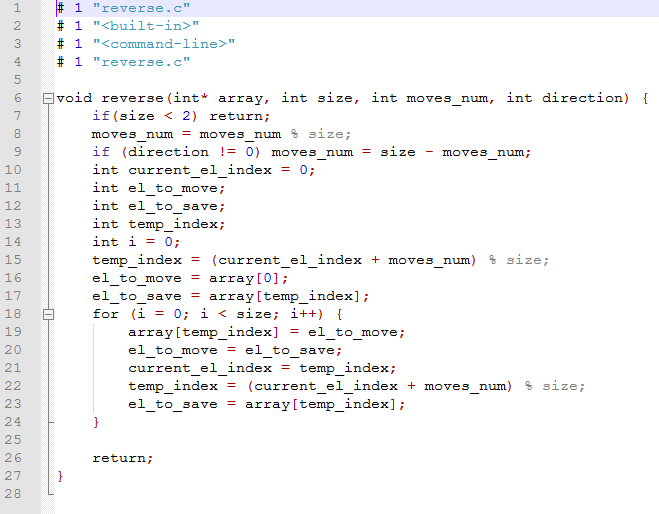


Рис. 3 Содержимое файла reverse.i

**Компилятор**

В результате работы компилятора в заданной папке появились файлы main.s и reverse.s в которых находится текст на языке ассемблера, являющийся результатом комплияции исходного кода (рис. 4-5).

Анализируя файл main.s, можно выделить несколько этапов:

Таблица 1 Анализ файла main.s

|  |  |
| --- | --- |
| Строчки файла | Комментарий |
| 11-13 | Сохранение адреса возврата и изначального значения s0. |
| 14-15 | Загрузка в регистры a0, a1 аргументов функции calloc, использующейся для выделения памяти под массив array. |
| 17-20 | Команды по подготовке значений регистров для цикла формирования массива array. В регистр a4 помещается адрес массива array, который функция calloc() поместила в регистр a0. В a5 помещается изначальное значение(0) счетчика циклов. В a1 помещается размер каждого элемента массива в байтах(4). В a3 помещается конечное значение счетчика(11). |
| 22-25 | Непосредственно тело цикла. В первой команде значение элемента массива помещается непосредственно в память по указателю, находящемуся в a4. Далее получается значение следующего элемента массива. А в третьей команде увеличивается указатель на массив на 4 байта. Последней командой является условный переход, реализующий повторение цикла в случае если заполнено было меньше 11 элементов массива. |
| 26-30 | В регистры a3-a0 помещаются аргументы функции reverse. В a3 – направление сдвига(1). В a2 – число сдвигов. В a1 – длина массива. В a0 – адрес начала массива. Далее производится вызов функции reverse(). |
| 31-32 | В a0 снова помещается адрес массива, необходимый в качестве аргумента функции free(), и производится вызов функции free(). |
| 33-37 | Эти функции выполняют восстановление изначальных значений регистров a0, s0 и получение адреса возврата, который помещается в ra. В предпоследней команде освобождается место в стеке. Последняя команда осуществляет безусловный переход по адресу возврата в регистре ra. |

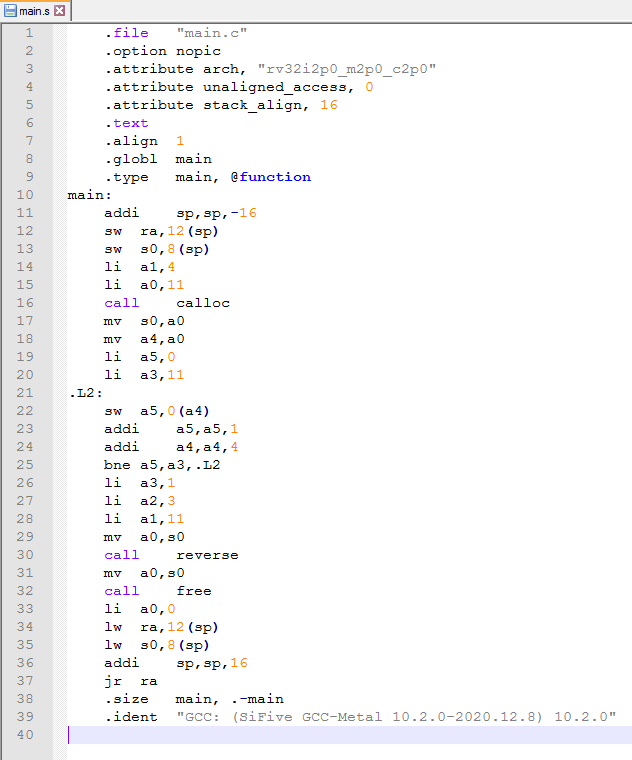


Рис. 4 Содержимое файла main.s

Проанализирую содержимое файла reverse.s:

Таблица 2 Анализа файла reverse.s

|  |  |
| --- | --- |
| Строчки файла | Комментарий |
| 11-12 | Проверка размера массива. В случае если массив состоит из 1 элемента или меньше, то производится переход в конец программы. |
| 13 | Из заданного количества сдвигов(a2) убираются целые «круги» сдвигов. a7 = a2 % a1 = shift\_num % array\_length |
| 14-15 | Производится проверка направления сдвига, и в зависимости от его значения формируется необходимое число сдвигов. |
| 17 | В регистре a5 формируется первоначальное значение temp\_index |
| 18 | В a6 помещается значение array[0] |
| 19-20 | В a4 формируется смещение элемента array[temp\_index] в байтах относительно array. После чего в a4 добавляется абсолютный адрес array и формируется абсолютный адрес array[temp\_index]. |
| 21 | В a2 помещается array[temp\_index] |
| 22 | В a3 помещается начальное значение счетчика итераций цикла – 0. |
| 24-26 | Начало основного цикла программы, в котором сначала производится сохранение замещаемого элемента в a6. |
| 27-28 | Формируется новый temp\_index. |
| 29 | В a6 сохраняется замещаемый элемент. |
| 30-32 | Формируется абсолютный адрес элемента array[temp\_index]. После чего он помещается в a2. |
| 33 | Счетчик циклов учеличивается на 1. |
| 34 | Условный переход, позволяющий сформировать сам цикл. |
| 36 | Возврат из подпрограммы reverse. |

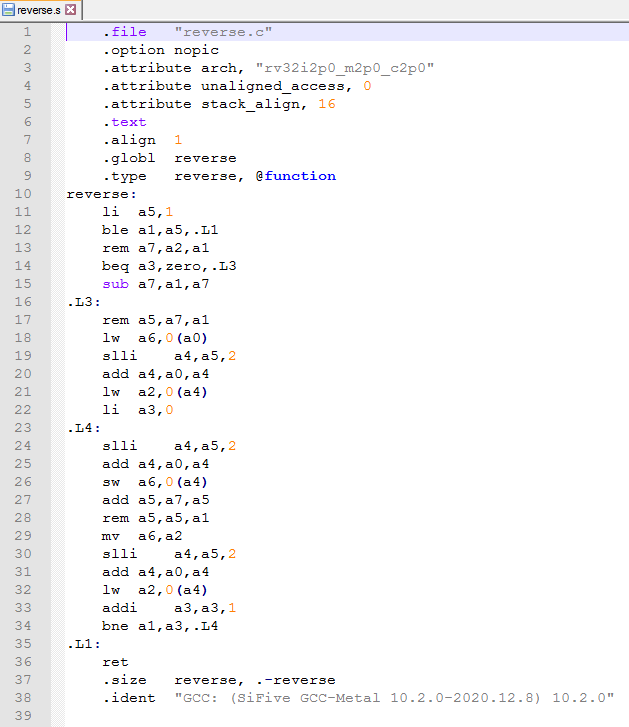


Рис. 5 Содержимое файла reverse.s

**Объектный файл**

В результате работы ассемблера были получены объектные файлы main.o и reverse.o. Проанализирую их заголовки, содержимое секций, таблицы символов и таблицы перемещений.

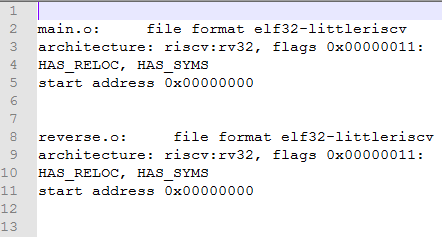


Рис. 6 Заголовки объектных файлов main.o и reverse.o

По флагам HAS\_SYMS и HAS\_RELOC можно понять, что оба файла содержат символы и таблицы перемещений соответственно. Так как объектный файл не должен содержать точки входа, то стартовый адрес равен нулю.

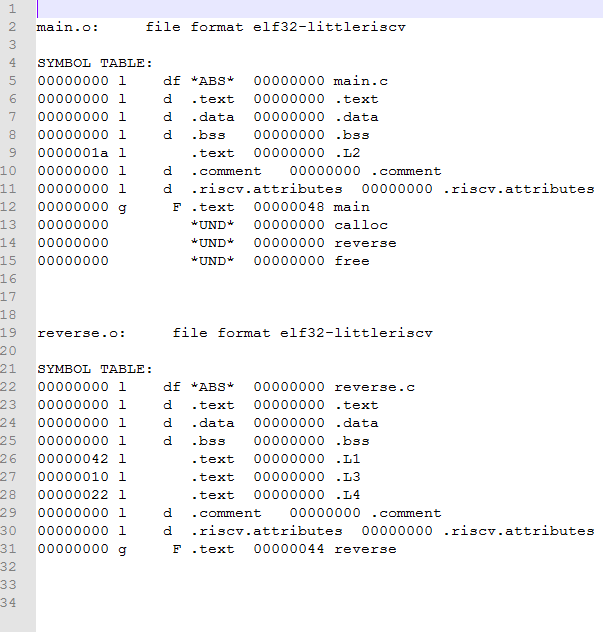


Рис. 7 Таблицы символов файлов main.o и reverse.o

В таблице символов main.o можно заметить неопределенные функции calloc, reverse и free, так как компоновка программы еще не производилась. Однако функция main определена и отмечена как глобальная. Таблица символов reverse.o показывает, что компилятор определил единственную функцию reverse как глобальную, так как никаких других функций в коде не использовалось.

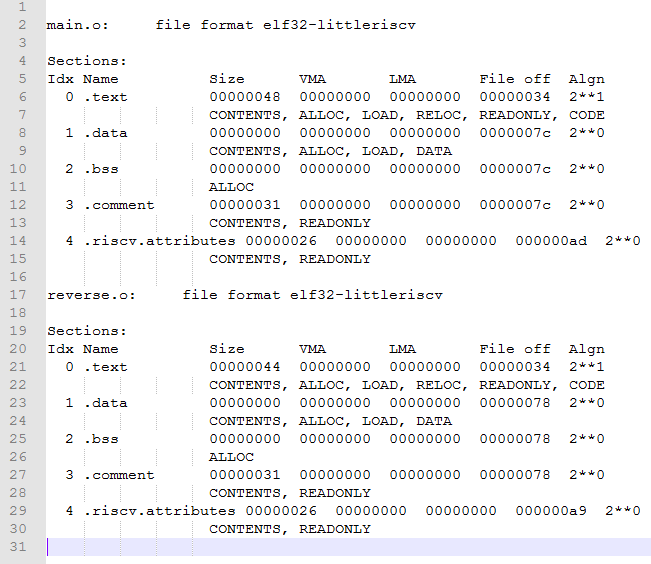


Рис. 8 Секции файлов main.o и reverse.o

В обеих таблицах можно заметить, что секции .text имеют отличный от нуля размер, тогда как секции .data пустые, так как в коде не определялось никаких отдельно стоящих структур, а все операции выделения памяти производились программно.

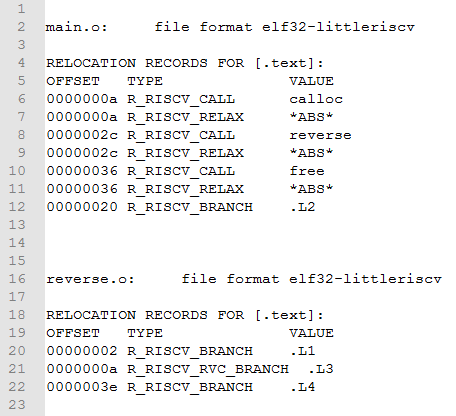


Рис. 9 Таблицы перемещений файлов main.o и reverse.o

В таблице файла main.o можно увидеть записи перемещений для функций calloc, reverse и free, а также их смещения относительно начала файла. Так как в файле reverse.o не было использовано никаких сторонних функций, его таблица перемещений состоит только из меток ветвлений, находящихся внутри этого же файла.

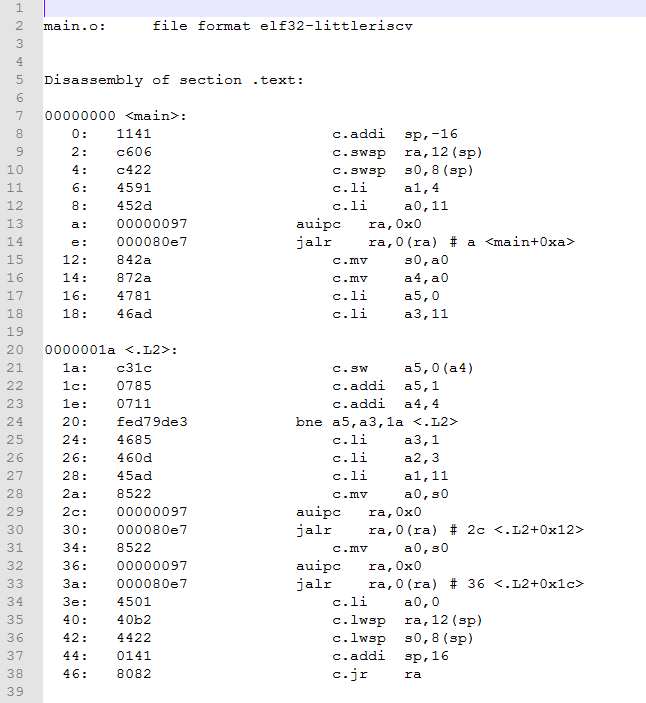


Рис. 10 Дизассемблированный код файла main.o

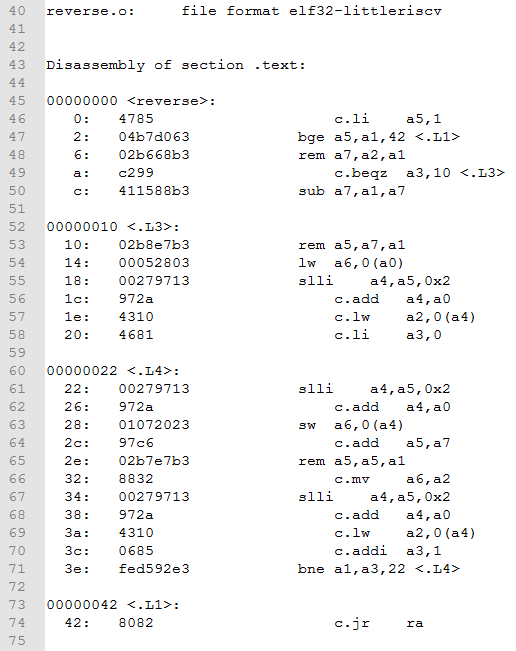


Рис. 11 Дизассемблированный код файла reverse.o

Просмотр кодов дизассемблированных файлов показывает, что данные коды отличаются от тех, что были получены в результате работы компилятора. Некоторые команды были заменены на формат команд расширения «C» архитектуры risc-v, под которое производилась компиляция. Причиной замены команд стала оптимизация, так как команды, добавляемые расширением «C» занимают не 32 бита, а лишь 16 бит, что делает код более компактным. Тем не менее анализ кода выявляет сходство с кодом, находящимся в файлах main.s reverse.s, что говорит о том, что этот код выполняет те же функции, что и рассмотренные ранее программы.

**Компоновщик**

Проанализирую исполняемый файл, выданный компоновщиком – a.out.

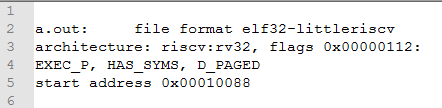


Рис. 12 Заголовок файла a.out

Видно, что в отличие от объектных файлов в данном исполняемом файле появился ненулевой стартовый адрес. Метка EXEC\_P указывает на то, что данный файл является исполняемым, а HAS\_SYMS обозначает что файл содержит символы.

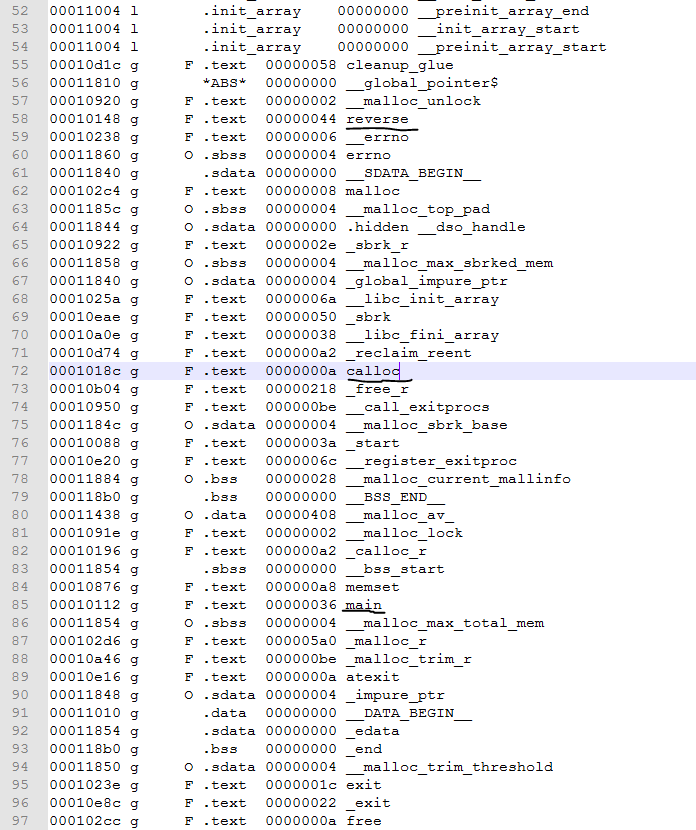


Рис. 13 Часть таблицы символов файла a.out

Таблица символов файла a.out значительно увеличилась за счет функций, добавленных библиотекой stdlib.h. Тем не менее в ней все равно сохраняются метки используемых функций calloc, main, reverse, которые по-прежнему являются глобальными. По адресу 0x10088 можно заметить функцию \_\_start, которая является точкой входа в программу в соответствии со значением стартового адреса из заголовка файла a.out.

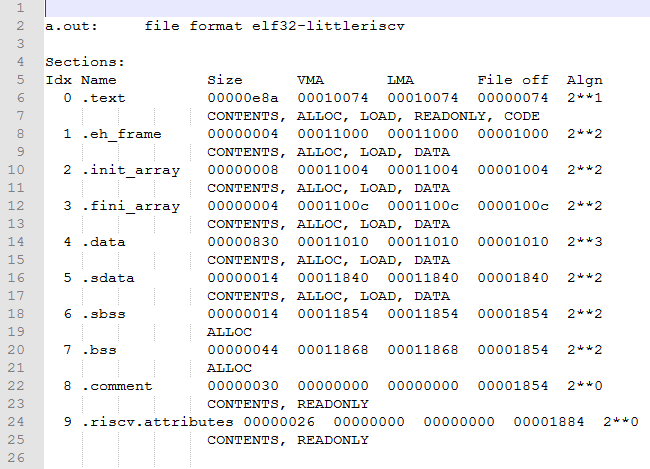


Рис. 14 Секции файла a.out

В таблице секций по-прежнему можно заметить секцию .text, размер которой отличен от нуля, так как в исполняемом файле записан необходимый код. Однако в данном файле остальные секции не являются пустыми, в отличие от объектных файлов, что может быть объяснено необходимостью использования каждой из секций в законченной исполняемой программе.

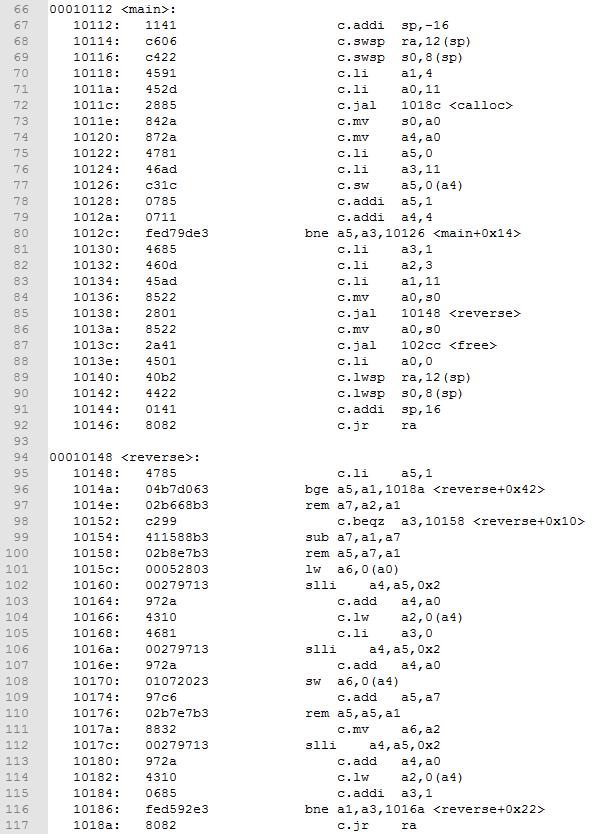


Рис. 15 Функции main и reverse внутри дизассемблированного файла a.out

Анализ кодов функций main и reverse, находящихся в дизассемблированном файле a.out показывает, что компоновщик, как это и ожидалось, не изменил команды этих функций, за исключением адресов вызовов команд и различных переходов.

**Создание библиотеки с использованием make**

На языке программы make был разработан makefile, который приводится на рис. 16. Выполнение данного файла программой make приводит к созданию двух целевых файлов executable и reverse\_lib.a и двух промежуточных объектных файлов main.o и reverse.o.

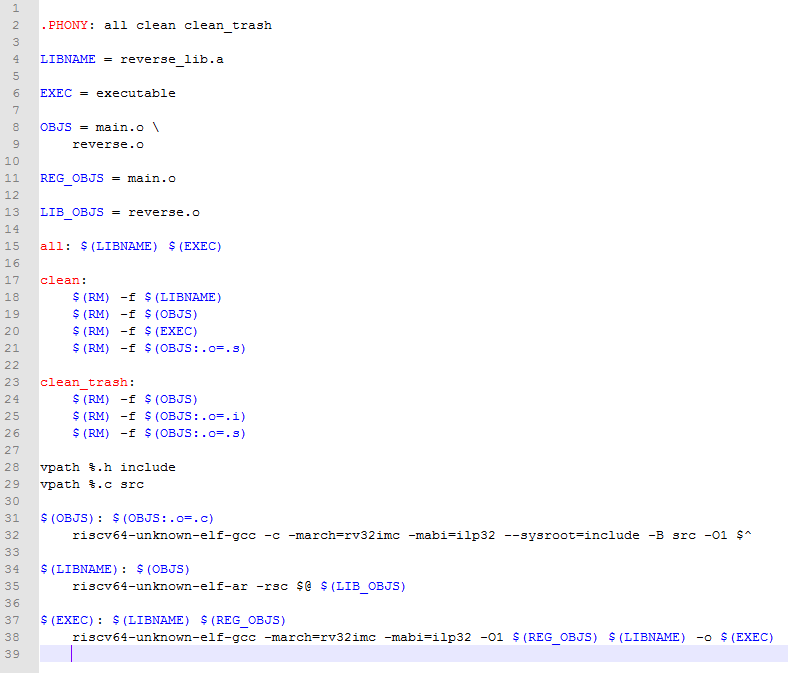


Рис. 16 Содержимое файла makefile

Анализ объектных файлов main.o и reverse.o показал, что содержимое данных файлов не отличается от соответствующих файлов, рассмотренных в предыдущей части отчета.

Статическая библиотека reverse\_lib.a формируется из одного файла reverse.o., что и является причиной того, что эти файлы никак не отличаются по своему содержанию.

На основе статической библиотеки reverse\_lib.a и объектного файла main.o компонуется конечный исполняемый файл executable, который и является целью сборки.

Анализ контрольных сумм файлов executable и a.out, полученного при ручной раздельной компиляции показывает, что эти файлы также идентичны(рис. 17).

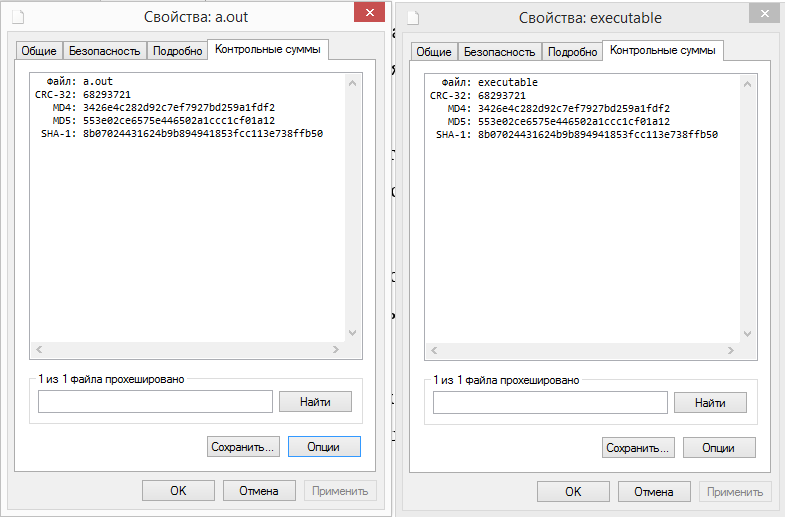


Рис. 17 Контрольные суммы файлов a.out и executable

**Вывод**

В ходе работы была проведена раздельная компиляция созданных исходных файлов программы. Результат каждого этапа компиляции был изучен и проанализирован. Ожидаемое содержимое объектных файлов, файлов с кодом языка ассемблера, препроцессированных файлов и исполняемых совпало с содержимым полученных файлов. Кроме того, процесс компиляции был успешно произведен с использованием программы make, для которой был написан соответствующий makefile.