Санкт-Петербургский Политехнический Университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа информационных систем и суперкомпьютерных технологий

Лабораторная работа №4
Дисциплина: «Низкоуровневое программирование».
Тема: «Раздельная компиляция».

Выполнил студенты группы 3530901/90004 _____ Унтила А.А.
(подпись)
Преподаватель _____ Алексюк А.О.
(подпись)

“__” _____ 2021 г.

Санкт-Петербург
2021

Цель работы

1. На языке С разработать функцию, реализующую определенную вариантом задания функциональность. Поместить определение функции в отдельный исходный файл, оформить заголовочный файл. Разработать тестовую программу на языке С.
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.

Программа работы

1. Постановка задачи

Реверс массива чисел – запись разрядов в обратном порядке.

Раздельная компиляция – разбиение проекта на несколько модулей (в данном случае несколько .c-файлов), компилирование всех модулей по отдельности (получение соответствующего кол-ва объектных .o-файлов) и компоновка / линковка в один исполняемый файл.

2. Пакет средств разработки

В лабораторной работе использовались средства разработки "SiFive GNU Embedded Toolchain", доступные для загрузки (для Windows, macOS или Linux) по следующей ссылке: https://www.sifive.com/products/tools/. Данные пакет средств разработки имеет классическую схему процесса сборки программ. Лабораторная работа выполнялась на ОС LinuxMint (Linux-Ubuntu) x64 (Виртуальная машина).

3. Функция, реализующая реверс массива

Листинг 3.1. – Код заголовочного файла ***reverse.h***.

```
void reverseArray(int *, int);
```

Листинг 3.2. – Код файла ***reverse.c***, реализующего реверс массива.

```
#include “reverse.h”

void reverseArray(int *array, int size) { // указатель, размер
     for (int i = 0; i < (size / 2); i++) { // рабочий цикл
          int current = array[i]; // текущий элемент
          array[i] = array[size - i - 1]; // присваивание зеркальных
          array[size - i - 1] = current; //  элементов
     }
}
```

Написанная программа-функция осуществляет реверс массива, для этого в неё передаются 2 входных параметра: указатель на массив (адрес начала массива), размер массива.

4. Тестовая программа

Листинг 4.1. – Код файла ***main.c***, вызывающего ф-цию реверса массива.

```
#include “reverse.h”

int main(void) {
    int a[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }; // массив
    reverseArray(a, (sizeof(a) / sizeof(a[0]))); // вызов ф-ции реверса
    return 0; // возврат
}
```

5. Сбор программы по шагам

Чтобы собрать программу (исполняемый файл) “по шагам”, необходимо понимать, какие шаги необходимо сделать и для чего они нужны.

Шаги сборки программы (теория):

- Препроцессирование – создание ***.i***-файла выхода препроцессора на основе ***.c***-файла. Файл содержит в себе помимо основного кода программы из ***.c*** файла ещё и сведения об используемых сторонних библиотеках, заголовочных файлах и т. д.
- Компиляция – создание (генерация) ***.s***-файла с кодом на языке ассемблера путём компиляции ***.i***-файла.
- Ассемблирование – создание (генерация ассемблером) ***.o*** объектного файла при помощи ***.s***-файла, содержащего код на языке ассемблера.

Важно: ***.o***-файл уже недоступен для чтения обычными текстовыми редакторами, т. к. вовсе не является текстовым, и информацию о нём (например коды инструкций, таблицы символов и перемещений) можно узнать только при помощи специальных консольных команд.

- Компановка – обработка объектных ***.o***-файлов и библиотек – генерирование компановщиком исполняемого (***executable***) файла. Компановка – последний шаг в процессе сборки программы – от ***.c***- к исполняемому файлу.

Полная сборка программы из указанных ***.c***-файлов может быть осуществлён следующей командой:

```
riscv64-unknown-elf-gcc --save-temps -march=rv32i -mabi=ilp32 -
O1 -v main.c >log 2>&1
```

Ключ “--save-temps” сохраняет все промежуточные файлы (***.i***, ***.s***, ***.o***), ключ “-v” запускает поток вывода информации о сборке, дополняющий его ключ “>log 2>&1” просто перенаправляют поток записи информации о сборке из консоли в файл ***log.txt***.

Для более удобного и наглядного практического описания шагов сборки программы будем генерировать все файлы по очереди, останавливая сборщик на определённых этапах.

Шаги сборки программы (практика):

- Прерпоцессирование:

```
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -O1 -v -E
main.c -o main.i >log.pp 2>&1
```

Приведённая команда позволяет получить файл выхода прерпоцесора. Ключ “-O1” отвечает за оптимизацию (упрощение) генерируемого кода. Ключ “-E” останавливает процесс сборки после препроцессирования.

- Компиляция:

```
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -O1 -S
main.i -o main.s
```

С помощью приведённой команды получаем файл с кодом на языке ассемблера. Ключ “-S” останавливает сборку после компиляции.

- Ассемблирование

```
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -c
main.s -o main.o
```

Получаем объектный файл из файла с кодом ассемблера и с помощью “-c” останавливаем сборку после ассемблирования.

Все 3 шага также выполняются и для файла ***reverse.c***, при чём порядок сборки объектных файлов не играет большой роли, просто в конечном исполняемом файле поменяются адреса ***main*** и ***reverse***.

```
# 1 "main.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "main.c"
# 1 "reverse.h" 1
void reverseArray(int *, int);
# 2 "main.c" 2

int main(void) {
    int a[] = { 0, 1, 2, 3, 4, 5, 6,
7, 8, 9, 10 };
    reverseArray(a, (sizeof(a) /
sizeof(a[0])));
    return 0;
}
```

Рис. 5.1.а) Файл ***main.i***.

```
# 1 "reverse.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "reverse.c"
# 1 "reverse.h" 1
void reverseArray(int *, int);
# 2 "reverse.c" 2

void reverseArray(int *array, int
size) {
    for (int i = 0; i < (size / 2);
i++) {
        int current = array[i];
        array[i] = array[size - i -
1];
        array[size - i - 1] =
current;
    }
}
```

Рис. 5.1.б) Файл ***reverse.i***.

.i-файлы содержат код из .c-файлов, а также директивы для информирования компилятора об их исходном тексте (ведь обрабатывать ***.i***-файлы в ***.s*** будет именно компилятор). Видим, что оба файла содержат информацию о заголовочном файле ***reverse.h***.

```
    .file  "main.c"
    .option nopic
    .attribute arch, "rv32i2p0"
    .attribute unaligned_access, 0
    .attribute stack_align, 16
    .text
    .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)
. . . . . . . . . . . . . . .   // заполнение массива
    li     a1,11
    addi   a0,sp,4
    call   reverseArray
    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  10
   .ident "GCC: (SiFive GCC 8.3.0-2019.08.0) 8.3.0"
```

Рис. 5.2. Файл ***main.s***.

Файл ***reverse.s*** также содержит код на языке ассемблера, однако, т. к. в ***reverse.c*** нет импортирования внешних файлов/функций, то в рамках данной лабораторной работы его содержимое не вызывает особого интереса.

Из рис. 5.2. видно, что после заполнения массива, как это и было в ***main.c***, происходит вызов функции ***reverseArray***. Видим знакомый набор инструкций для записи адреса возврата в стек во избежание зацикливания программы. Также важным моментов является то, что ф-ция ***reverseArray*** хоть и вызывается в ***main.s***, однако нигде не описана.

Как было упомянуто ранее, файлы ***.o*** не являются текстовыми, поэтому их содержимое можно посмотреть при помощи следующих консольных команд:

```
riscv64-unknown-elf-objdump -t reverse.o main.o
```

Рис. 5.3.1. Вывод символьных таблиц ***reverse*** и ***main***.

```
reverse.o:     file format elf32-littleriscv

SYMBOL TABLE:
00000000 l    df *ABS*   00000000 reverse.c
00000000 l    d  .text   00000000 .text
00000000 l    d  .data   00000000 .data
00000000 l    d  .bss    00000000 .bss
00000044 l       .text   00000000 .L1
00000024 l       .text   00000000 .L3
00000000 l    d  .comment    00000000 .comment
00000000 l    d  .riscv.attributes00000000 .riscv.attributes
00000000 g     F .text   00000048 reverseArray



main.o:     file format elf32-littleriscv

SYMBOL TABLE:
00000000 l    df *ABS*   00000000 main.c
00000000 l    d  .text   00000000 .text
00000000 l    d  .data   00000000 .data
```

```
00000000 l    d  .bss    00000000 .bss
00000000 l    d  .rodata 00000000 .rodata
00000000 l       .rodata 00000000 .LANCHOR0
00000000 l    d  .comment    00000000 .comment
00000000 l    d  .riscv.attributes 00000000 .riscv.attributes
00000000 g     F .text   00000088 main
00000000         *UND*   00000000 reverseArray
```

Рис. 5.3.2. Символьные таблицы ***reverse*** и ***main***.

На последней строчке рис. 5.3.2 (символьная таблица ***main***) видна запись символа с типом *UND*, что является сокращением от английского “**und**efined” – неопределённый. Как и было сказано ранее при анализе файлов с кодом ассемблера, функция ***reverseArray*** вызывается, однако нигде не описана (не определена), что и подтверждает полученная в символьной таблице запись.

```
riscv64-unknown-elf-objdump –d –M no-aliases –j .text reverse.o main.o
```

Рис. 5.4.1. Вывод инструкций (секция .text) ***reverse*** и ***main***.

```
reverse.o:     file format elf32-littleriscv


Disassembly of section .text:

00000000 <reverseArray>:
   0:     01f5d813            srli a6,a1,0x1f
   4:     00b80833            add  a6,a6,a1
   8:     40185813            srai a6,a6,0x1
   c:     00100793            addi a5,zero,1
  10:     02b7da63            bge  a5,a1,44 <.L1>
  14:     00050793            addi a5,a0,0
  18:     00259593            slli a1,a1,0x2
  1c:     00b50533            add  a0,a0,a1
  20:     00000713            addi a4,zero,0

00000024 <.L3>:
  24:     0007a683            lw   a3,0(a5)
  28:     ffc52603            lw   a2,-4(a0)
  2c:     00c7a023            sw   a2,0(a5)
  30:     fed52e23            sw   a3,-4(a0)
  34:     00170713            addi a4,a4,1
  38:     00478793            addi a5,a5,4
  3c:     ffc50513            addi a0,a0,-4
  40:     ff0742e3            blt  a4,a6,24 <.L3>

00000044 <.L1>:
  44:     00008067            jalr zero,0(ra)

main.o:     file format elf32-littleriscv


Disassembly of section .text:

00000000 <main>:
```

```
   0:     fc010113            addi sp,sp,-64
   4:     02112e23            sw   ra,60(sp)
   8:     000007b7            lui  a5,0x0
   c:     00078793            addi a5,a5,0 # 0 <main>
  10:     0007ae83            lw   t4,0(a5)
  14:     0047ae03            lw   t3,4(a5)
  18:     0087a303            lw   t1,8(a5)
  1c:     00c7a883            lw   a7,12(a5)
  20:     0107a803            lw   a6,16(a5)
  24:     0147a503            lw   a0,20(a5)
  28:     0187a583            lw   a1,24(a5)
  2c:     01c7a603            lw   a2,28(a5)
  30:     0207a683            lw   a3,32(a5)
  34:     0247a703            lw   a4,36(a5)
  38:     0287a783            lw   a5,40(a5)
  3c:     01d12223            sw   t4,4(sp)
  40:     01c12423            sw   t3,8(sp)
  44:     00612623            sw   t1,12(sp)
  48:     01112823            sw   a7,16(sp)
  4c:     01012a23            sw   a6,20(sp)
  50:     00a12c23            sw   a0,24(sp)
  54:     00b12e23            sw   a1,28(sp)
  58:     02c12023            sw   a2,32(sp)
  5c:     02d12223            sw   a3,36(sp)
  60:     02e12423            sw   a4,40(sp)
  64:     02f12623            sw   a5,44(sp)
  68:     00b00593            addi a1,zero,11
  6c:     00410513            addi a0,sp,4
  70:     00000097            auipc     ra,0x0
  74:     000080e7            jalr ra,0(ra) # 70 <main+0x70>
  78:     00000513            addi a0,zero,0
  7c:     03c12083            lw   ra,60(sp)
  80:     04010113            addi sp,sp,64
  84:     00008067            jalr zero,0(ra)
```

Рис. 5.4.2. Инструкции (секция .text) ***reverse*** и ***main***.

Также как и в символьной таблице, в выводе инструкций нас интересует вызов ***reverseArray*** из ***main***. На строчках с индексами 70 и 74 (слева) видно, в какую пару инструкций ассемблер преобразовал вызов функции ***reverseArray***. Ассемблер присвоил адресу перехода адрес строки, из которой функция была вызвана, т. е., очевидно, неправильно определил адрес перехода, однако ассемблеру неоткуда взять адрес вызываемой ф-ции (как было сказано ранее, для него она не определена), этой задачей уже будет заниматься компановщик. Для информирования компановщика о неопределённых инструкциях служит таблица перемещений.

```
riscv64-unknown-elf-objdump -r reverse.o main.o
```

Рис. 5.5.1. Вывод таблиц перемещений ***reverse*** и ***main***.

```
Reverse.o:     file format elf32-littleriscv


Disassembly of section .text:

00000000 <reverseArray>:
   0:     01f5d813            srli a6,a1,0x1f
   4:     00b80833            add  a6,a6,a1
   8:     40185813            srai a6,a6,0x1
   c:     00100793            addi a5,zero,1
  10:     02b7da63            bge  a5,a1,44 <.L1>
          10: R_RISCV_BRANCH  .L1
  14:     00050793            addi a5,a0,0
  18:     00259593            slli a1,a1,0x2
  1c:     00b50533            add  a0,a0,a1
  20:     00000713            addi a4,zero,0

00000024 <.L3>:
  24:     0007a683            lw   a3,0(a5)
  28:     ffc52603            lw   a2,-4(a0)
  2c:     00c7a023            sw   a2,0(a5)
  30:     fed52e23            sw   a3,-4(a0)
  34:     00170713            addi a4,a4,1
  38:     00478793            addi a5,a5,4
  3c:     ffc50513            addi a0,a0,-4
  40:     ff0742e3            blt  a4,a6,24 <.L3>
          40: R_RISCV_BRANCH  .L3

00000044 <.L1>:
  44:     00008067            jalr zero,0(ra)

main.o:     file format elf32-littleriscv


Disassembly of section .text:

00000000 <main>:
   0:     fc010113            addi sp,sp,-64
   4:     02112e23            sw   ra,60(sp)
   8:     000007b7            lui  a5,0x0
          8: R_RISCV_HI20     .LANCHOR0
          8: R_RISCV_RELAX    *ABS*
   c:     00078793            addi a5,a5,0 # 0 <main>
          c: R_RISCV_LO12_I   .LANCHOR0
          c: R_RISCV_RELAX    *ABS*
  10:     0007ae83            lw   t4,0(a5)
  14:     0047ae03            lw   t3,4(a5)
  18:     0087a303            lw   t1,8(a5)
  1c:     00c7a883            lw   a7,12(a5)
  20:     0107a803            lw   a6,16(a5)
  24:     0147a503            lw   a0,20(a5)
  28:     0187a583            lw   a1,24(a5)
  2c:     01c7a603            lw   a2,28(a5)
  30:     0207a683            lw   a3,32(a5)
  34:     0247a703            lw   a4,36(a5)
```

```
38:     0287a783            lw   a5,40(a5)
3c:     01d12223            sw   t4,4(sp)
40:     01c12423            sw   t3,8(sp)
44:     00612623            sw   t1,12(sp)
48:     01112823            sw   a7,16(sp)
4c:     01012a23            sw   a6,20(sp)
50:     00a12c23            sw   a0,24(sp)
54:     00b12e23            sw   a1,28(sp)
58:     02c12023            sw   a2,32(sp)
5c:     02d12223            sw   a3,36(sp)
60:     02e12423            sw   a4,40(sp)
64:     02f12623            sw   a5,44(sp)
68:     00b00593            addi a1,zero,11
6c:     00410513            addi a0,sp,4
70:     00000097            auipc     ra,0x0
        70: R_RISCV_CALL    reverseArray
        70: R_RISCV_RELAX   *ABS*
74:     000080e7            jalr ra,0(ra) # 70 <main+0x70>
78:     00000513            addi a0,zero,0
7c:     03c12083            lw   ra,60(sp)
80:     04010113            addi sp,sp,64
84:     00008067            jalr zero,0(ra)
```

Рис. 5.5.2. Таблицы перемещений ***reverse*** и ***main***.

На рис. 5.5.2 видны инструкции, выделенные верхним регистром и имеющие префикс “R_RISCV” – это и есть неопределённые инструкции, которые передаются компановщику на определение. Непосредственно вызов функции ***reverseArray*** имеет понятную запись “R_RISCV_CALL”, а также дополнительную запись “R_RISCV_RELAX”. Эта запись означает, что вызов подпрограммы в виде инструкций может быть оптимизирован. Более подробно: используемая обычно пара инструкций “auipc+jalr”, обеспечивающая переход на адрес в пределах ± 2 ГиБ, что для нашей небольшой программы является сильно с “запасом”, поэтому ассемблер указывает на возможность использования одной инструкции “jal”, которая обеспечивает переход в пределах ± 1 МиБ, чего в нашем случае оказалось достаточно.

- Компановка:

```
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 main.o reverse.o
-o main
```

Рис. 5.6. Компановка двух объектных файлов в один исполняемый.

Команда на рис. 5.6 эквивалентна команде для полной сборки программы из .c-файлов, указанной в начале пункта 5. Отличие заключается лишь в том, что на рис. 5.6 мы создаём исполняемый файл из уже двух объектных файлов, а также при помощи ключа “`-o`” указываем имя создаваемого файла.

```
riscv64-unknown-elf-objdump –j .text –d –M no-aliases main>main.ds
```

Рис. 5.6.1. Запись инструкций из секции .text исполняемого файла в файл ***main.ds***.

Посмотрим на часть файла main.ds, где осуществляется вызов функции ***reverseArray***.

```
main:     file format elf32-littleriscv


Disassembly of section .text:
. . . . . .
00010144 <main>:
   10144: fc010113            addi sp,sp,-64
   10148: 02112e23            sw   ra,60(sp)
   1014c: 000107b7            lui  a5,0x10
   10150: 63078793            addi a5,a5,1584 # 10630 <__errno+0xc>
   10154: 0007ae83            lw   t4,0(a5)
. . . . . .
   101ac: 00b00593            addi a1,zero,11
   101b0: 00410513            addi a0,sp,4
   101b4: 014000ef            jal  ra,101c8 <reverseArray>
   101b8: 00000513            addi a0,zero,0
   101bc: 03c12083            lw   ra,60(sp)
   101c0: 04010113            addi sp,sp,64
   101c4: 00008067            jalr zero,0(ra)

000101c8 <reverseArray>:
   101c8: 01f5d813            srli a6,a1,0x1f
   101cc: 00b80833            add  a6,a6,a1
   101d0: 40185813            srai a6,a6,0x1
   101d4: 00100793            addi a5,zero,1
   101d8: 02b7da63            bge  a5,a1,1020c <reverseArray+0x44>
. . . . . .
   10208: ff0742e3            blt  a4,a6,101ec <reverseArray+0x24>
   1020c: 00008067            jalr zero,0(ra)
```

Рис. 5.6.2. Фрагмент содержимого файла ***main.ds***.

Из рис. 5.6.2. видим, что ф-ция ***reverseArray*** определена в скомпанованном файле и из ***main*** при вызове происходит переход на известный адрес. Также видно, что произошла оптимизация "**linker relaxation**" (R_RISCV_RELAX) и переход к адресу функции ***reverseArray*** осуществлён при помощи инструкции **jal**.

6. Создание статической библиотеки

Для создания статической библиотеки используется команда "riscv64-unknown-elf-ar -rsc". Результатом её работы является архив, куда помещены все объектные файлы используемых модулей, кроме ***main***-файла.

В нашем случае такой объектный файл всего один. Создадим статическую библиотеку:

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

Рис. 6.1. Создание статической библиотеки.

Далее просмотрим содержимое библиотеки:

```
riscv64-unknown-elf-ar -t lib.a
reverse.o
```

Рис. 6.2. Команда для вывода и вывод содержимого библиотеки (архива ***.a***).

Используем библиотеку для сборки программы (сохраняя файлы, генерируемые на промежуточных шагах):

```
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -O1 --save-temps
main.c lib.a -o main
```

Рис. 6.3. Сборка программы с использованием статической библиотеки.

```
riscv64-unknown-elf-objdump -t main >symtabl
```

Рис. 6.4.1. Команда для записи символьной таблицы ***main*** в файл ***symtabl***.

```
main:     file format elf32-littleriscv

SYMBOL TABLE:
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
00000000 l    df *ABS*  00000000 main.c
00000000 l    df *ABS*  00000000 reverse.c
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
000101c8 g     F .text  00000048 reverseArray
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

Рис. 6.4.2. Фрагменты содержимого файла ***symtabl***.

Из рис. 6.4.2 видно, что компановщик определил использование функции реверса в ***main*** и включил в исполняемый файл все необходимые объектные файлы. Для того, чтобы однозначно в этом убедится, выведем содержимое символьной таблицы объектного файла ***main.o***.

```
riscv64-unknown-elf-objdump -t main.o
main.o:     file format elf32-littleriscv

SYMBOL TABLE:
00000000 l    df *ABS*  00000000 main.c
00000000 l    d  .text  00000000 .text
00000000 l    d  .data  00000000 .data
00000000 l    d  .bss   00000000 .bss
00000000 l    d  .rodata 00000000 .rodata
00000000 l       .rodata 00000000 .LANCHOR0
00000000 l    d  .comment    00000000 .comment
00000000 l    d  .riscv.attributes 00000000 .riscv.attributes
00000000 g     F .text  00000088 main
00000000         *UND*  00000000 reverseArray
```

Рис. 6.5. Символьная таблица файла ***main.o***.

В символьной таблице содержится уже знакомая строка с функцией "`reverseArray`" и её "**und**efined" – неопределённым типом. Посмотрим содержимое символьной таблицы файла reverse.o, расположенного в библиотеке, в котором, как мы знаем, должна быть данная функция определена.

```
riscv64-unknown-elf-objdump -t reverse.o
reverse.o:     file format elf32-littleriscv

SYMBOL TABLE:
00000000 l    df *ABS*  00000000 reverse.c
00000000 l    d  .text  00000000 .text
00000000 l    d  .data  00000000 .data
00000000 l    d  .bss   00000000 .bss
00000044 l       .text  00000000 .L1
00000024 l       .text  00000000 .L3
00000000 l    d  .comment    00000000 .comment
00000000 l    d  .riscv.attributes 00000000 .riscv.attributes
00000000 g     F .text  00000048 reverseArray
```

Рис. 6.6. Символьная таблица файла ***reverse.o***.

Из рис. 6.6 видим, что ф-ция "`reverseArray`" в данной символьной таблице определена – имеет типо "`.text`" вместо "`*UND*`".

Наконец изучим список символов статической библиотеки: объектные файлы и содержащиеся в них ф-ции.

```
riscv64-unknown-elf-nm lib.a
```
```
reverse.o:
00000044 t .L1
00000024 t .L3
00000000 T reverseArray
```
Рис. 6.7. Список символов статической библиотеки ***lib.a***.

Объектный файл ***reverse.o*** содержит ф-цию ***reverseArray*** с меткой “T ”, что означает, что ф-ция определена внутри этого объектного файла, если бы метка была “U”, это бы свидетельствовало о том, что ф-ция не определена в данном объектном файле и является внешней.

7. Создание make-файла

Все действия по сбору библиотеки и сбору программы можно автоматизировать, написав **make**-файлы, запуск которых командой “make -f” выполнит сборку статической библиотеки, а после сборку исполняемого файла.

```
# "Фиктивные" цели
.PHONY: all clean

#Вызываемые приложения
AR = ./riscv64-unknown-elf-ar
CC = ./riscv64-unknown-elf-gcc

# Файл библиотеки
LIBNAME = lib.a

# Параметры компиляции march=rv32i -mabi=ilp32
CFLAGS = -march=rv32i -mabi=ilp32 -O1

# Построение объектного файла из исходного текста
# значением переменной $< будет имя первого файла в списке зависимостей, в данном случае - имя исходного файла
# $@ – имя .o-файла
# $< – имя .c-файла
# Создание .o из .c
.o: .c
	$(CC) $(CFLAGS) -c $< -o $@

# Чтобы достичь цели "all", требуется построить библиотеку
all: $(LIBNAME)

# $^ = (reverse.o)
$(LIBNAME): reverse.o
	$(AR) -rsc $@ $^
```
Рис. 7.1. **make**-файл для сборки статической библиотеки.

```
# "Фиктивные" цели
.PHONY: all clean

# Файлы для сборки исполнимого файла
OBJS= main.c \
     lib.a

#Вызываемые приложения
CC = ./riscv64-unknown-elf-gcc

# Параметры компиляции
CFLAGS = -march=rv32i -mabi=ilp32 -O1

# Чтобы достичь цели "all", требуется собрать исполняемый файл
all: a.out

# Сборка исполняемого файла
a.out: $(OBJS)
     $(CC) $(CFLAGS) $^
```

Рис. 7.2. **make**-файл для сборки программы.

Вывод:

В ходе лабораторной работе исследована тема раздельной компиляции. Изучены основные шаги перехода от кода программы к исполняемому файлу и соответствующие каждому шагу файлы. Осуществлена сборка программы по шагам, отдельно останавливаясь и анализируя каждый этап сборки. Параллельно изучены возможности “SiFive GNU Embedded Toolchain”: различные ключи пакета команд “riscv64-unknown-elf”, позволяющие тем или иным образов оптимизировать сборку программы.

Также были рассмотрены и изучены такие важные элементы, как статические библиотеки, позволяющие более эффективно и быстро собирать программу, используя уже имеющиеся объектные файлы. В конце созданы **make**-файлы, автоматизирующие процессы сборки статической библиотеки и исполняемого файла.