Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа компьютерных технологий и информационных систем

**Отчёт по лабораторной работе № 4**

Дисциплина: Низкоуровневое программирование

Тема: Раздельная компиляция

Выполнил студент гр. 3530901/90006 Вагнер А. А.

(подпись)

Преподаватель Целищева А. А. (подпись)

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

Санкт-Петербург

2023

1. **Развернутая формулировка задачи**

Вариант 4: Сортировка вставкой массива чисел in-place. Реализовать программу на языке C, совершающую обработку заданной строки и сохранением чисел в массив. Выделить заданный функционал в подпрограмму и создать тестовую программу для неё. Провести пошаговую компиляцию данных программ с анализом получающихся файлов. Создать статическую библиотеку, состоящую из тестовой программы и программы с необходимым функционалом, проследив этапы её создания.

1. **Текст программы на языке C**

**Листинг 1.1.** Код программы main

|  |
| --- |
| #include <stdio.h>  #include "ISort.h"  int main() {      int a[10] = {-10, 0, 10, 9, 8, 7, 1, 2, 5, 5};      InsertionSort(10, a);      for (int i = 0; i<10; i++)          printf("%d ", a[i]);      return 0;  } |

**Листинг 1.2.** Код подпрограммы ISort

|  |
| --- |
| void InsertionSort(int n, int \*mass) {      int newElement, location;      for (int i = 1; i < n; i++) {          newElement = mass[i];          location = i - 1;          while(location >= 0 && mass[location] > newElement) {              mass[location+1] = mass[location];              location = location - 1;          }          mass[location+1] = newElement;      }  } |

**Листинг 1.3.** Код заголовка ISort

#ifndef LAB4

#define LAB4

void InsertionSort(int n, int \*mass);

#endif //LAB4

1. **Сборка программы по “шагам” и анализ полученных файлов**
   1. **Препроцессирование**

Для получения результатов работы препроцессора выполним следующие команды:

riscv64-unknown-elf-gcc.exe -march=rv64i -mabi=lp64 -O1 -E main.c -o main.i

riscv64-unknown-elf-gcc.exe -march=rv64i -mabi=lp64 -O1 -E ISort.c -o ISort.i

**Листинг 2.1.** Результат препроцессирования для main.c (фрагмент)

|  |
| --- |
| # 2 "main.c" 2  # 1 "ISort.h" 1  # 3 "ISort.h"  void InsertionSort(int n, int mass[]);  # 3 "main.c" 2  int main() {  int a[10] = {-10, 0, 10, 9, 8, 7, 1, 2, 5, 5};  ISort(10, a);  for (int i = 0; i<10; i++)  printf("%d ", a[i]);  return 0;  } |

**Листинг 2.2.** Результат препроцессирования для ISort.c

|  |
| --- |
| # 1 "ISort.c"  # 1 "<built-in>"  # 1 "<command-line>"  # 1 "ISort.c"  void InsertionSort(int n, int \*mass) {  int newElement, location;  for (int i = 1; i < n; i++) {  newElement = mass[i];  location = i - 1;  while(location >= 0 && mass[location] > newElement) {  mass[location+1] = mass[location];  location = location - 1;  }  mass[location+1] = newElement;  }  } |

Как можно видеть из листингов 2.1 и 2.2, в результате препроцессирования сам код не претерпел каких-либо изменений. Однако перед началом кода в обоих файлов добавился текст технического назначения.

* 1. **Компиляция**

Для выполнения компиляции используем следующие команды:

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

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

**Листинг 3.1.** Результат компиляции для main.i

|  |
| --- |
| .file "main.c"  .option nopic  .attribute arch, "rv64i2p0"  .attribute unaligned\_access, 0  .attribute stack\_align, 16  .text  .section .rodata.str1.8,"aMS",@progbits,1  .align 3  .LC1:  .string "%d "  .text  .align 2  .globl main  .type main, @function  main:  addi sp,sp,-80  sd ra,72(sp)  sd s0,64(sp)  sd s1,56(sp)  sd s2,48(sp)  lui a5,%hi(.LANCHOR0)  addi a5,a5,%lo(.LANCHOR0)  ld a1,0(a5)  ld a2,8(a5)  ld a3,16(a5)  ld a4,24(a5)  ld a5,32(a5)  sd a1,8(sp)  sd a2,16(sp)  sd a3,24(sp)  sd a4,32(sp)  sd a5,40(sp)  addi a1,sp,8  li a0,10  call ISort  addi s0,sp,8  addi s2,sp,48  lui s1,%hi(.LC1)  .L2:  lw a1,0(s0)  addi a0,s1,%lo(.LC1)  call printf  addi s0,s0,4  bne s0,s2,.L2  li a0,0  ld ra,72(sp)  ld s0,64(sp)  ld s1,56(sp)  ld s2,48(sp)  addi sp,sp,80  jr ra  .size main, .-main  .section .rodata  .align 3  .set .LANCHOR0,. + 0  .LC0:  .word -10  .word 0  .word 10  .word 9  .word 8  .word 7  .word 1  .word 2  .word 5  .word 5  .ident "GCC: (GNU) 10.1.0" |

**Листинг 3.2.** Результат препроцессирования для ISort.i

|  |
| --- |
| .file "ISort.c"  .option nopic  .attribute arch, "rv64i2p0"  .attribute unaligned\_access, 0  .attribute stack\_align, 16  .text  .align 2  .globl InsertionSort  .type InsertionSort, @function  InsertionSort:  li a5,1  ble a0,a5,.L1  mv a7,a1  addiw t1,a0,-1  li a6,0  li a0,-1  j .L6  .L4:  addi a4,a4,1  slli a4,a4,2  add a4,a1,a4  sw a2,0(a4)  addiw a6,a6,1  addi a7,a7,4  beq a6,t1,.L1  .L6:  lw a2,4(a7)  sext.w a4,a6  mv a5,a7  blt a6,zero,.L4  .L3:  lw a3,0(a5)  ble a3,a2,.L4  sw a3,4(a5)  addiw a4,a4,-1  addi a5,a5,-4  bne a4,a0,.L3  j .L4  .L1:  ret  .size InsertionSort, .-InsertionSort  .ident "GCC: (GNU) 10.1.0" |

Результатом работы компилятора стал код ассемблера RISC-V для соответствующих программ.

**3.3 Ассемблирование**

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

riscv64-unknown-elf-gcc.exe -march=rv64i -mabi=lp64 -v -c main.s -o main.o

riscv64-unknown-elf-gcc.exe -march=rv64i -mabi=lp64 -v -c ISort.s -o ISort.o

Прочесть полученные в результате ассемблирования файлы также как предыдущие не получится. Для этого используем следующие команды:

riscv64-unknown-elf-objdump.exe -h main.o

riscv64-unknown-elf-objdump.exe -t main.o

riscv64-unknown-elf-objdump.exe -s -j .text main.o

riscv64-unknown-elf-objdump.exe -s -j .comment main.o

riscv64-unknown-elf-objdump.exe -r main.o

**Листинг 4.1.** Заголовки секций в main.o.

|  |
| --- |
| main.o: file format elf64-littleriscv  Sections:  Idx Name Size VMA LMA File off Algn  0 .text 00000094 0000000000000000 0000000000000000 00000040 2\*\*2  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE  1 .data 00000000 0000000000000000 0000000000000000 000000d4 2\*\*0  CONTENTS, ALLOC, LOAD, DATA  2 .bss 00000000 0000000000000000 0000000000000000 000000d4 2\*\*0  ALLOC  3 .rodata.str1.8 00000004 0000000000000000 0000000000000000 000000d8 2\*\*3  CONTENTS, ALLOC, LOAD, READONLY, DATA  4 .rodata 00000028 0000000000000000 0000000000000000 000000e0 2\*\*3  CONTENTS, ALLOC, LOAD, READONLY, DATA  5 .comment 00000013 0000000000000000 0000000000000000 00000108 2\*\*0  CONTENTS, READONLY  6 .riscv.attributes 0000001c 0000000000000000 0000000000000000 0000011b 2\*\*0  CONTENTS, READONLY |

**Листинг 4.2.** Таблица символов в main.o.

|  |
| --- |
| main.o: file format elf64-littleriscv  SYMBOL TABLE:  0000000000000000 l df \*ABS\* 0000000000000000 main.c  0000000000000000 l d .text 0000000000000000 .text  0000000000000000 l d .data 0000000000000000 .data  0000000000000000 l d .bss 0000000000000000 .bss  0000000000000000 l d .rodata.str1.8 0000000000000000 .rodata.str1.8  0000000000000000 l d .rodata 0000000000000000 .rodata  0000000000000000 l .rodata 0000000000000000 .LANCHOR0  0000000000000000 l .rodata.str1.8 0000000000000000 .LC1  0000000000000060 l .text 0000000000000000 .L2  0000000000000000 l d .comment 0000000000000000 .comment  0000000000000000 l d .riscv.attributes 0000000000000000 .riscv.attributes  0000000000000000 g F .text 0000000000000094 main  0000000000000000 \*UND\* 0000000000000000 ISort  0000000000000000 \*UND\* 0000000000000000 printf |

**Листинг 4.3.** Содержание секции .text в main.o.

|  |
| --- |
| main.o: file format elf64-littleriscv  Contents of section .text:  0000 130101fb 23341104 23308104 233c9102 ....#4..#0..#<..  0010 23382103 b7070000 93870700 83b50700 #8!.............  0020 03b68700 83b60701 03b78701 83b70702 ................  0030 2334b100 2338c100 233cd100 2330e102 #4..#8..#<..#0..  0040 2334f102 93058100 1305a000 97000000 #4..............  0050 e7800000 13048100 13090103 b7040000 ................  0060 83250400 13850400 97000000 e7800000 .%..............  0070 13044400 e31624ff 13050000 83308104 ..D...$......0..  0080 03340104 83348103 03390103 13010105 .4...4...9......  0090 67800000 |

**Листинг 4.4.** Содержание секции .comment в main.o.

|  |
| --- |
| main.o: file format elf64-littleriscv  Contents of section .comment:  0000 00474343 3a202847 4e552920 31302e31 .GCC: (GNU) 10.1  0010 2e3000 .0. |

**Листинг 4.5.** Таблица перемещений для main.0.

|  |
| --- |
| main.o: file format elf64-littleriscv  RELOCATION RECORDS FOR [.text]:  OFFSET TYPE VALUE  0000000000000014 R\_RISCV\_HI20 .LANCHOR0  0000000000000014 R\_RISCV\_RELAX \*ABS\*  0000000000000018 R\_RISCV\_LO12\_I .LANCHOR0  0000000000000018 R\_RISCV\_RELAX \*ABS\*  000000000000004c R\_RISCV\_CALL ISort  000000000000004c R\_RISCV\_RELAX \*ABS\*  000000000000005c R\_RISCV\_HI20 .LC1  000000000000005c R\_RISCV\_RELAX \*ABS\*  0000000000000064 R\_RISCV\_LO12\_I .LC1  0000000000000064 R\_RISCV\_RELAX \*ABS\*  0000000000000068 R\_RISCV\_CALL printf  0000000000000068 R\_RISCV\_RELAX \*ABS\*  0000000000000074 R\_RISCV\_BRANCH .L2 |

Аналогичные команды использованы для получения рисунков 13-18, за исключением замены main.o на stringToIntArray.o.

**Листинг 4.6.** Заголовки секций в ISort.o.

|  |
| --- |
| ISort.o: file format elf64-littleriscv  Sections:  Idx Name Size VMA LMA File off Algn  0 .text 00000068 0000000000000000 0000000000000000 00000040 2\*\*2  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE  1 .data 00000000 0000000000000000 0000000000000000 000000a8 2\*\*0  CONTENTS, ALLOC, LOAD, DATA  2 .bss 00000000 0000000000000000 0000000000000000 000000a8 2\*\*0  ALLOC  3 .comment 00000013 0000000000000000 0000000000000000 000000a8 2\*\*0  CONTENTS, READONLY  4 .riscv.attributes 0000001c 0000000000000000 0000000000000000 000000bb 2\*\*0  CONTENTS, READONLY |

**Листинг 4.7.** Таблица символов в ISort.o.

|  |
| --- |
| ISort.o: file format elf64-littleriscv  SYMBOL TABLE:  0000000000000000 l df \*ABS\* 0000000000000000 ISort.c  0000000000000000 l d .text 0000000000000000 .text  0000000000000000 l d .data 0000000000000000 .data  0000000000000000 l d .bss 0000000000000000 .bss  0000000000000064 l .text 0000000000000000 .L1  0000000000000038 l .text 0000000000000000 .L6  000000000000001c l .text 0000000000000000 .L4  0000000000000048 l .text 0000000000000000 .L3  0000000000000000 l d .comment 0000000000000000 .comment  0000000000000000 l d .riscv.attributes 0000000000000000 .riscv.attributes  0000000000000000 g F .text 0000000000000068 InsertionSort |

**Листинг 4.8.** Содержание секции .text в ISort.o.

|  |
| --- |
| ISort.o: file format elf64-littleriscv  Contents of section .text:  0000 93071000 63d0a706 93880500 1b03f5ff ....c...........  0010 13080000 1305f0ff 6f000002 13071700 ........o.......  0020 13172700 3387e500 2320c700 1b081800 ..'.3...# ......  0030 93884800 63086802 03a64800 1b070800 ..H.c.h...H.....  0040 93870800 e34c08fc 83a60700 e358d6fc .....L.......X..  0050 23a2d700 1b07f7ff 9387c7ff e316a7fe #...............  0060 6ff0dffb 67800000 o...g... |

**Листинг 4.9.** Содержание секции .comment в ISort.o.

|  |
| --- |
| Contents of section .comment:  0000 00474343 3a202847 4e552920 31302e31 .GCC: (GNU) 10.1  0010 2e3000 .0. |

**Листинг 4.10.** Таблица перемещений для ISort.o.

|  |
| --- |
| ISort.o: file format elf64-littleriscv  RELOCATION RECORDS FOR [.text]:  OFFSET TYPE VALUE  0000000000000004 R\_RISCV\_BRANCH .L1  0000000000000018 R\_RISCV\_JAL .L6  0000000000000034 R\_RISCV\_BRANCH .L1  0000000000000044 R\_RISCV\_BRANCH .L4  000000000000004c R\_RISCV\_BRANCH .L4  000000000000005c R\_RISCV\_BRANCH .L3  0000000000000060 R\_RISCV\_JAL .L4 |

Анализируя полученные тексты можно сказать, что мы получаем из этой информации. Изучая таблицу секций, можем определить: какие данные в ней находятся, сколько они занимают места и в каком коде закодированы. Таблица символов ожидаемо правильно содержит один глобальный флаг “g”, указывающий на символ типа “функции”. Декодировав содержание секции .text мы можем получить код программы, а .comment значение после .indent в скомпилированном коде. Таблица перемещений содержит информацию о всех неоконченных инструкциях.

Дизассемблируем код с помощью команд:

riscv64-unknown-elf-objdump.exe -d -M no-aliases -r main.o

riscv64-unknown-elf-objdump.exe -d -M no-aliases -r ISort.o

**Листинг 5.1.** Результат дизассемблирования для main.o.

|  |
| --- |
| main.o: file format elf64-littleriscv  Disassembly of section .text:  0000000000000000 <main>:  0: fb010113 addi sp,sp,-80  4: 04113423 sd ra,72(sp)  8: 04813023 sd s0,64(sp)  c: 02913c23 sd s1,56(sp)  10: 03213823 sd s2,48(sp)  14: 000007b7 lui a5,0x0  14: R\_RISCV\_HI20 .LANCHOR0  14: R\_RISCV\_RELAX \*ABS\*  18: 00078793 addi a5,a5,0 # 0 <main>  18: R\_RISCV\_LO12\_I .LANCHOR0  18: R\_RISCV\_RELAX \*ABS\*  1c: 0007b583 ld a1,0(a5)  20: 0087b603 ld a2,8(a5)  24: 0107b683 ld a3,16(a5)  28: 0187b703 ld a4,24(a5)  2c: 0207b783 ld a5,32(a5)  30: 00b13423 sd a1,8(sp)  34: 00c13823 sd a2,16(sp)  38: 00d13c23 sd a3,24(sp)  3c: 02e13023 sd a4,32(sp)  40: 02f13423 sd a5,40(sp)  44: 00810593 addi a1,sp,8  48: 00a00513 addi a0,zero,10  4c: 00000097 auipc ra,0x0  4c: R\_RISCV\_CALL ISort  4c: R\_RISCV\_RELAX \*ABS\*  50: 000080e7 jalr ra,0(ra) # 4c <main+0x4c>  54: 00810413 addi s0,sp,8  58: 03010913 addi s2,sp,48  5c: 000004b7 lui s1,0x0  5c: R\_RISCV\_HI20 .LC1  5c: R\_RISCV\_RELAX \*ABS\*  0000000000000060 <.L2>:  60: 00042583 lw a1,0(s0)  64: 00048513 addi a0,s1,0 # 0 <main>  64: R\_RISCV\_LO12\_I .LC1  64: R\_RISCV\_RELAX \*ABS\*  68: 00000097 auipc ra,0x0  68: R\_RISCV\_CALL printf  68: R\_RISCV\_RELAX \*ABS\*  6c: 000080e7 jalr ra,0(ra) # 68 <.L2+0x8>  70: 00440413 addi s0,s0,4  74: ff2416e3 bne s0,s2,60 <.L2>  74: R\_RISCV\_BRANCH .L2  78: 00000513 addi a0,zero,0  7c: 04813083 ld ra,72(sp)  80: 04013403 ld s0,64(sp)  84: 03813483 ld s1,56(sp)  88: 03013903 ld s2,48(sp)  8c: 05010113 addi sp,sp,80  90: 00008067 jalr zero,0(ra) |

**Листинг 5.2.** Результат дизассемблирования для ISort.o.

|  |
| --- |
| ISort.o: file format elf64-littleriscv  Disassembly of section .text:  0000000000000000 <InsertionSort>:  0: 00100793 addi a5,zero,1  4: 06a7d063 bge a5,a0,64 <.L1>  4: R\_RISCV\_BRANCH .L1  8: 00058893 addi a7,a1,0  c: fff5031b addiw t1,a0,-1  10: 00000813 addi a6,zero,0  14: fff00513 addi a0,zero,-1  18: 0200006f jal zero,38 <.L6>  18: R\_RISCV\_JAL .L6  000000000000001c <.L4>:  1c: 00170713 addi a4,a4,1  20: 00271713 slli a4,a4,0x2  24: 00e58733 add a4,a1,a4  28: 00c72023 sw a2,0(a4)  2c: 0018081b addiw a6,a6,1  30: 00488893 addi a7,a7,4  34: 02680863 beq a6,t1,64 <.L1>  34: R\_RISCV\_BRANCH .L1  0000000000000038 <.L6>:  38: 0048a603 lw a2,4(a7)  3c: 0008071b addiw a4,a6,0  40: 00088793 addi a5,a7,0  44: fc084ce3 blt a6,zero,1c <.L4>  44: R\_RISCV\_BRANCH .L4  0000000000000048 <.L3>:  48: 0007a683 lw a3,0(a5)  4c: fcd658e3 bge a2,a3,1c <.L4>  4c: R\_RISCV\_BRANCH .L4  50: 00d7a223 sw a3,4(a5)  54: fff7071b addiw a4,a4,-1  58: ffc78793 addi a5,a5,-4  5c: fea716e3 bne a4,a0,48 <.L3>  5c: R\_RISCV\_BRANCH .L3  60: fbdff06f jal zero,1c <.L4>  60: R\_RISCV\_JAL .L4  0000000000000064 <.L1>:  64: 00008067 jalr zero,0(ra) |

В листингах можем видеть, что дизассемблированный код практически идентичен коду, полученному на этапе ассемблирования.

**3.4 Компоновка**

Скомпонуем программу с помощью команды:

riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -v main.o ISort.o

Для просмотра получившегося кода воспользуемся следующей командой:

riscv64-unknown-elf-objdump.exe –j .text –d –M no-aliases a.out >a.ds

**Листинг 6.1.** Результат компоновки.

|  |
| --- |
| 00010144 <main>:  10144: fc010113 addi sp,sp,-64  10148: 02112e23 sw ra,60(sp)  1014c: 02812c23 sw s0,56(sp)  10150: 04010413 addi s0,sp,64  10154: 000257b7 lui a5,0x25  10158: 33478793 addi a5,a5,820 # 25334 <\_\_clzsi2+0x50>  1015c: 0007ae03 lw t3,0(a5)  10160: 0047a303 lw t1,4(a5)  10164: 0087a883 lw a7,8(a5)  10168: 00c7a803 lw a6,12(a5)  1016c: 0107a503 lw a0,16(a5)  10170: 0147a583 lw a1,20(a5)  10174: 0187a603 lw a2,24(a5)  10178: 01c7a683 lw a3,28(a5)  1017c: 0207a703 lw a4,32(a5)  10180: 0247a783 lw a5,36(a5)  10184: fdc42223 sw t3,-60(s0)  10188: fc642423 sw t1,-56(s0)  1018c: fd142623 sw a7,-52(s0)  10190: fd042823 sw a6,-48(s0)  10194: fca42a23 sw a0,-44(s0)  10198: fcb42c23 sw a1,-40(s0)  1019c: fcc42e23 sw a2,-36(s0)  101a0: fed42023 sw a3,-32(s0)  101a4: fee42223 sw a4,-28(s0)  101a8: fef42423 sw a5,-24(s0)  101ac: fc440793 addi a5,s0,-60  101b0: 00078593 addi a1,a5,0  101b4: 00a00513 addi a0,zero,10  101b8: 060000ef jal ra,10218 <InsertionSort>  101bc: fe042623 sw zero,-20(s0)  101c0: 0340006f jal zero,101f4 <main+0xb0>  101c4: fec42783 lw a5,-20(s0)  101c8: 00279793 slli a5,a5,0x2  101cc: ff040713 addi a4,s0,-16  101d0: 00f707b3 add a5,a4,a5  101d4: fd47a783 lw a5,-44(a5)  101d8: 00078593 addi a1,a5,0  101dc: 000257b7 lui a5,0x25  101e0: 33078513 addi a0,a5,816 # 25330 <\_\_clzsi2+0x4c>  101e4: 308000ef jal ra,104ec <printf>  101e8: fec42783 lw a5,-20(s0)  101ec: 00178793 addi a5,a5,1  101f0: fef42623 sw a5,-20(s0)  101f4: fec42703 lw a4,-20(s0)  101f8: 00900793 addi a5,zero,9  101fc: fce7d4e3 bge a5,a4,101c4 <main+0x80>  10200: 00000793 addi a5,zero,0  10204: 00078513 addi a0,a5,0  10208: 03c12083 lw ra,60(sp)  1020c: 03812403 lw s0,56(sp)  10210: 04010113 addi sp,sp,64  10214: 00008067 jalr zero,0(ra) |

Компоновщик все переходы сочетания инструкций auipc и jalr заменил на инструкцию jal и корректным адресом перехода.

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

Для получения статической библиотеки выполним следующие последовательности команд:

riscv64-unknown-elf-gcc.exe -march=rv64i -mabi=lp64 -O1 -c ISort.c -o ISort.o

riscv64-unknown-elf-ar.exe -rsc staticLib.a ISort.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 toBinaryDecimal.o с помощью команды:

riscv64-unknown-elf-objdump.exe -t a.out (Рис. 22)

Изображение выглядит как текст

Автоматически созданное описание

Рис. 22. Элементы символьной таблицы a.out.

Также мы можем создать два make-файла: первый для создания статической библиотеки и второй для сборки приложения. Коды данных make-файлов представлены на рисунках 23-24.

Изображение выглядит как текст

Автоматически созданное описание

Рис. 23. Make-файл для сборки статической библиотеки.

Изображение выглядит как текст

Автоматически созданное описание

Рис. 24. Make-файл для сборки приложения

1. **Вывод**

В ходе лабораторной работы была поэтапно проанализирована работа

препроцессора, компилятора и компоновщика пакета GCC и драйвера компилятора riscv64-unknown-elf-gcc на примере программы по представлению целого числа со знаком в памяти устройства. Также была успешно создана статическая библиотека с функционалом обработки строки целых чисел и сохранением их в массив с помощью пакета средств разработки “SiFive GNU Embedded Toolchain”, а также с помощью make-файлов