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: 2

Выполнил студент группы 3530901/00002 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.Г. Антонов

(подпись)

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

(подпись)

«\_\_» \_\_\_\_\_\_\_\_\_\_ 2021г.

Санкт-Петербург

2021

**Задание:**

1. Изучить методические материалы, опубликованные на сайте курса.

2. Установить пакет средств разработки “SiFive GNU Embedded Toolchain” для RISC-V.

3. 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.

4. 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.

5. 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.

Задание по варианту: Формирование в памяти десятичного представления дробного числа (-1;1).

1. **Программа на языке С**

Изображение выглядит как текст

Автоматически созданное описание

**Рис 1.1** Заголовочный файл form.h

Изображение выглядит как текст

Автоматически созданное описание

**Рис 1.2** Основной файл form.c

Изображение выглядит как текст

Автоматически созданное описание

**Рис 1.3** Тестовая программа main.c

Изображение выглядит как текст

Автоматически созданное описание

**Рис 1.4** Результат выполнения программы

Как видно из рис 1.4, первая цифра это знак числа. Далее идут числа после запятой.

1. **Сборка программы «по шагам»**

**Препроцессирование**

Препроцессирование выполняется следующими командами:

riscv64-unknown-elf-gcc --save-temps -march=rv32imac -mabi=ilp32 -O1 -E main.c -o main.i

riscv64-unknown-elf-gcc --save-temps -march=rv32imac -mabi=ilp32 -O1 -E form.c -o form.i

Результат препроцессирования содержится в файлах main.i и form.i*.* По причине того, что main.c содержит заголовочный файл стандартной библиотеки языка С stdio.h, результат препроцессирования этого файла имеет достаточно много добавочных строк.

Рис 2.1 Файл main.i (фрагмент)

|  |
| --- |
| # 1 "main.c" # 1 "<built-in>" # 1 "<command-line>" # 1 "main.c"  ………….  # 2 "main.c" 2 # 1 "form.h" 1        # 7 "form.h" void form (int num, int array[],int len); # 3 "main.c" 2  int main() { double x=0.32; int y=0; double num = x; float a = (int)num; while(a!=num){  num\*=10;  a=(int)num;  y+=1; } int array[y+1];  form((int)num, array,y);  for(int i=0;i<y+1;i++){  printf("%d",array[i]);  putchar('\n');  }  return 0; } |

Рис 2.2 Файл form.i

|  |
| --- |
| # 1 "form.c" # 1 "<built-in>" # 1 "<command-line>" # 1 "form.c" # 1 "form.h" 1       void form (int num, int array[],int len); # 2 "form.c" 2  void form(int num, int array[],int len){  if (num<0){  array[0]=1;  num=0-num;  }else{  array[0]=0;  }  for(int i = len; i>0;i--){  int d=num%10;  array[i]=d;  num=num/10;  } } |

**Компиляция**

Компиляция выполняется следующими командами:

riscv64-unknown-elf-gcc -march=rv32ic -mabi=ilp32 -O1 -S -fpreprocessed main.i -o main.s

riscv64-unknown-elf-gcc -march=rv32ic -mabi=ilp32 -O1 -S -fpreprocessed form.i -o form.s

Наибольший интерес представляет файл main.s, так как в нем можно заметить обращение к подпрограмме bSort(значение регистра *ra*, содержащее адрес возврата из main, сохраняется на время вызова в стеке).

Рис 2.3 Файл main.s

|  |
| --- |
| .file "main.c"  .option nopic  .attribute arch, "rv32i2p0\_c2p0"  .attribute unaligned\_access, 0  .attribute stack\_align, 16  .text  .align 1  .globl main  .type main, @function main:  addi sp,sp,-32  sw ra,28(sp)  sw s0,24(sp)  sw s1,20(sp)  sw s2,16(sp)  sw s3,12(sp)  addi s0,sp,32  addi sp,sp,-16  mv s2,sp  li a2,2  mv a1,s2  li a0,32  call form  mv s1,s2  addi s2,s2,12  lui s3,%hi(.LC0) .L2:  lw a1,0(s1)  addi a0,s3,%lo(.LC0)  call printf  li a0,10  call putchar  addi s1,s1,4  bne s2,s1,.L2  li a0,0  addi sp,s0,-32  lw ra,28(sp)  lw s0,24(sp)  lw s1,20(sp)  lw s2,16(sp)  lw s3,12(sp)  addi sp,sp,32  jr ra  .size main, .-main  .section .rodata.str1.4,"aMS",@progbits,1  .align 2 .LC0:  .string "%d"  .ident "GCC: (SiFive GCC 8.3.0-2020.04.1) 8.3.0" |

Рис. 2.4 Файл form.s

|  |
| --- |
| .file "form.c"  .option nopic  .attribute arch, "rv32i2p0\_c2p0"  .attribute unaligned\_access, 0  .attribute stack\_align, 16  .text  .globl \_\_modsi3  .globl \_\_divsi3  .align 1  .globl form  .type form, @function form:  addi sp,sp,-16  sw ra,12(sp)  sw s0,8(sp)  sw s1,4(sp)  sw s2,0(sp)  mv s1,a0  li a5,0  blt a0,zero,.L8 .L2:  sw a5,0(a1)  ble a2,zero,.L1  slli a2,a2,2  add s0,a1,a2  mv s2,a1 .L4:  li a1,10  mv a0,s1  call \_\_modsi3  sw a0,0(s0)  li a1,10  mv a0,s1  call \_\_divsi3  mv s1,a0  addi s0,s0,-4  bne s0,s2,.L4 .L1:  lw ra,12(sp)  lw s0,8(sp)  lw s1,4(sp)  lw s2,0(sp)  addi sp,sp,16  jr ra .L8:  neg s1,a0  li a5,1  j .L2  .size form, .-form  .ident "GCC: (SiFive GCC 8.3.0-2020.04.1) 8.3.0" |

**Ассемблирование**

Ассемблирование осуществляется следующими командами:

riscv64-unknown-elf-gcc--save-temps -march=rv32imac -mabi=ilp32 -v -c main.s -o main.o

riscv64-unknown-elf-gcc --save-temps -march=rv32imac -mabi=ilp32 -v -c form.s -o form.o

Рис. 2.5 Заголовки секции файла main.o

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

Изображение выглядит как текст

Автоматически созданное описание

Рис. 2.6 Таблица символов файла main.o

|  |
| --- |
| riscv64-unknown-elf-objdump -t main.o –t |

Изображение выглядит как текст

Автоматически созданное описание

В таблице символов main.o имеется запись: символ “form” типа \*UND\*. Эта запись означает, что символ “formt” использовался в ассемблерном коде, из которого был получен данный объектный файл, но не был определен; ассемблер сделал вывод о том, что символ должен быть определен где-то еще, и отразил это в таблице символов. То же самое относится и к символу “printf”.

Рис 2.7 Таблица перемещений файла мain.o

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

|  |
| --- |
| main.o: file format elf32-littleriscv  Disassembly of section .text:  00000000 <main>:  0: 1101 c.addi sp,-32  2: ce06 c.swsp ra,28(sp)  4: cc22 c.swsp s0,24(sp)  6: ca26 c.swsp s1,20(sp)  8: c84a c.swsp s2,16(sp)  a: c64e c.swsp s3,12(sp)  c: 1000 c.addi4spn s0,sp,32  e: 1141 c.addi sp,-16  10: 890a c.mv s2,sp  2a: 408c c.lw a1,0(s1)  2c: 00098513 addi a0,s3,0 # 0 <main>  2c: R\_RISCV\_LO12\_I .LC0  2c: R\_RISCV\_RELAX \*ABS\*  30: 00000097 auipc ra,0x0  30: R\_RISCV\_CALL printf  30: R\_RISCV\_RELAX \*ABS\*  34: 000080e7 jalr ra,0(ra) # 30 <.L2+0x6>  38: 4529 c.li a0,10  3a: 00000097 auipc ra,0x0  3a: R\_RISCV\_CALL putchar  3a: R\_RISCV\_RELAX \*ABS\*  3e: 000080e7 jalr ra,0(ra) # 3a <.L2+0x10>  42: 0491 c.addi s1,4  44: fe9913e3 bne s2,s1,2a <.L2>  44: R\_RISCV\_BRANCH .L2  48: 4501 c.li a0,0  4a: fe040113 addi sp,s0,-32  4e: 40f2 c.lwsp ra,28(sp)  50: 4462 c.lwsp s0,24(sp)  52: 44d2 c.lwsp s1,20(sp)  54: 4942 c.lwsp s2,16(sp)  56: 49b2 c.lwsp s3,12(sp)  58: 6105 c.addi16sp sp,32  5a: 8082 c.jr ra |

Рис. 2.8 Заголовки секции файла form.o

|  |
| --- |
| riscv64-unknown-elf-objdump -h form.o |

Изображение выглядит как текст

Автоматически созданное описание

Рис. 2.9 Таблица символов form.o

|  |
| --- |
| riscv64-unknown-elf-objdump -t form.o -t |

Изображение выглядит как текст

Автоматически созданное описание

**Компоновка**

Компоновка осуществляется следующей командой:

|  |
| --- |
| riscv64-unknown-elf-gcc --save-temps -march=rv32imac -mabi=ilp32 -v main.o form.o |

Рис. 2.10 Исполняемый файл a.out (фрагмент)

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

|  |
| --- |
| a.out: file format elf32-littleriscv   Disassembly of section .text:  ………..  00010110 <main>:  10110: 1101 c.addi sp,-32  10112: ce06 c.swsp ra,28(sp)  10114: cc22 c.swsp s0,24(sp)  10116: ca26 c.swsp s1,20(sp)  10118: c84a c.swsp s2,16(sp)  1011a: c64e c.swsp s3,12(sp)  1011c: 1000 c.addi4spn s0,sp,32  1011e: 1141 c.addi sp,-16  10120: 890a c.mv s2,sp  10122: 4609 c.li a2,2  10124: 85ca c.mv a1,s2  10126: 02000513 addi a0,zero,32  1012a: 203d c.jal 10158 <form>  1012c: 84ca c.mv s1,s2  1012e: 0931 c.addi s2,12  10130: 69f9 c.lui s3,0x1e  10132: 408c c.lw a1,0(s1)  10134: 13098513 addi a0,s3,304 # 1e130 <\_\_clzsi2+0x3c>  10138: 22dd c.jal 1031e <printf>  1013a: 4529 c.li a0,10  1013c: 2c19 c.jal 10352 <putchar>  1013e: 0491 c.addi s1,4  10140: fe9919e3 bne s2,s1,10132 <main+0x22>  10144: 4501 c.li a0,0  10146: fe040113 addi sp,s0,-32  1014a: 40f2 c.lwsp ra,28(sp)  1014c: 4462 c.lwsp s0,24(sp)  1014e: 44d2 c.lwsp s1,20(sp)  10150: 4942 c.lwsp s2,16(sp)  10152: 49b2 c.lwsp s3,12(sp)  10154: 6105 c.addi16sp sp,32  10156: 8082 c.jr ra  00010158 <form>:  10158: 4781 c.li a5,0  1015a: 02054163 blt a0,zero,1017c <form+0x24>  1015e: c19c c.sw a5,0(a1)  10160: 00c05d63 bge zero,a2,1017a <form+0x22>  10164: 060a c.slli a2,0x2  10166: 962e c.add a2,a1  10168: 47a9 c.li a5,10  1016a: 02f56733 rem a4,a0,a5  1016e: c218 c.sw a4,0(a2)  10170: 02f54533 div a0,a0,a5  10174: 1671 c.addi a2,-4  10176: feb61ae3 bne a2,a1,1016a <form+0x12>  1017a: 8082 c.jr ra  1017c: 40a00533 sub a0,zero,a0  10180: 4785 c.li a5,1  10182: bff1 c.j 1015e <form+0x6> |

1. **Создание статической библиотеки и make-файлов**

Объединим form.c и use.c в статическую библиотеку formLib, тестовую программу main.c оставим без изменений.

Для создания статической библиотеки получим объектные файлы всех используемых программ: form.o и use.o.

|  |
| --- |
| riscv64-unknown-elf-gcc.exe -march=rv64i -mabi=ilp64 -O1 -c form.c -o form.o  riscv64-unknown-elf-gcc.exe -march=rv64i -mabi=ilp64 -O1 -c use.c -o use.o |

Объединим получившиеся файлы в одну библиотеку следующей командой:

|  |
| --- |
| riscv64-unknown-elf-ar.exe -rsc formLib.a form.o use.o |

Используя получившуюся библиотеку, соберем исполняемый файл программы следующей командой:

|  |
| --- |
| riscv64-unknown-elf-gcc.exe -march=rv64i -mabi=ilp64 -O1 --save-temps main.c formLib.a |

Процесс выполнения команд выше можно заменить make-файлами, которые произведут создание библиотеки и сборку программы.

Makefile-1

|  |
| --- |
| # "Фиктивные" цели  .PHONY: all  # Исходные файлы, необходимые для сборки библиотеки  OBJS= form.c \  use.c  #Вызываемые приложения  AR = riscv64-unknown-elf-ar  CC = riscv64-unknown-elf-gcc  # Файл библиотеки  MYLIBNAME = newFormLib.a  # Параметры компиляции  CFLAGS= -O1  # Включаемые файлы следует искать в текущем каталоге  INCLUDES+= -I .  # Make должна искать файлы \*.h и \*.c в текущей директории  vpath %.h .  vpath %.c .  # Построение объектного файла из исходного текста  # $< = %.c  # $@ = %.o  %.o: %.c  $(CC) -MD $(CFLAGS) $(INCLUDES) -c $< -o $@  # Чтобы достичь цели "all", требуется построить библиотеку  all: $(MYLIBNAME)  # $^ = (form.o, use.o)  $(MYLIBNAME): form.o use.o  $(AR) -rsc $@ $^ |

Makefile-2

|  |
| --- |
| Содержание make-файла для создания статической библиотеки.  # "Фиктивные" цели  .PHONY: all  # Файлы для сборки исполнимого файла  OBJS= main.c \  newformLib.a  #Вызываемые приложения  CC = riscv64-unknown-elf-gcc.exe  # Параметры компиляции  CFLAGS= --save-temps  # Включаемые файлы следует искать в текущем каталоге  INCLUDES+= -I .  # Make должна искать файлы \*.c и \*.a в текущей директории  vpath %.c .  vpath %.a .  # Чтобы достичь цели "all", требуется собрать исполнимый файл  all: a.out  # Сборка исполнимого файла и удаление мусора  a.out: $(OBJS)  $(CC) $(CFLAGS) $(INCLUDES) $^  del \*.o \*.i \*.s \*.d |

1. **Вывод**

В ходе лабораторной работы изучена пошаговая компиляция программы на языке C. На этапе препроцессирования <filename>.c получили <filename>.i. На этапе компиляции мы из полученного файла с расширением «.i» получаем файл на «.s»– код на языке ассемблера. При ассемблировании мы из файла <filename>.i получаем объектный файл с расширением «.o». На последнем этапе (компоновке) из объектного файла получаем исполняемый.

Также была создана статическая библиотека и произведена сборка программы с помощью Makefile. С помощью этих файлов и инструкций,

содержащихся в них, мы можем собрать программу в автоматическом режиме.