|  |  |  |
| --- | --- | --- |
| Лабораторная Работа №3 | М3100 | 2022 |
| ISA | Ерёмин Владимир Ильич | |
|  |

**Цель работы:** знакомство с архитектурой набора команд RISC-V.

**Инструментарий и требования к работе:**C++20, MSVC *(19.34.31937.0, x64)* и CMake *(3.24.202208181-MSVC\_2).*

**Описание**: изучить систему кодирования команд RISC-V и структуру Elf файла. Решить предложенную задачу. Оформить отчет.

**Задача:** необходимо написать программу-транслятор (дизассемблер), с помощью которой можно преобразовывать машинный код в текст программы на языке ассемблера.

# 

# Система кодирования команд RISC-V

## Основная информация.

Систему кодирования команд описывает так называемая **ISA** (Instruction set architecture). В общем ISA задает поддерживаемые инструкции, типы данных, регистры. Описывает управление памятью, а также фундаментальный функционал. А также модель ввода/вывода..RISС-V — это, семейство связанных между собой ISA. На данных момент оно состоит из 4-основных ISA. Каждая из которых задает ширину регистров, размер адресного пространства и количество регистров, соответственно. Из них мы будем рассматривать **RV32I**.

**RV32I** имеет: 32-битное адресное пространство, 32-битные инструкции и 32-битные целочисленные регистры.

**RISC-V** была спроектирована с расчетом на расширяемость и специфику конкретного окружения. Так, каждая из базовых ISA может быть расширена с одним или более *“расширением набора инструкций”.* Причем, стандарт задает и специфику конкретной инструкции, это сделано для того, чтобы основные инструкции не были перекрыты расширениями. Мы рассматриваем стандартную **ISA - I**, и расширение **M -** для поддержки целочисленного деления и умножения.

Система также задает и модель памяти: в данном случае **RV32I** имеет зацикленное байт-адресное адресное пространство размером байт. Также задаются стандартные типы данных *(эти типы встретятся еще при описании elf-файла):* word (4 байта), halfword (2 байта), doubleword (8 байт) и quadword (16 байт).

Хоть мы требуем здесь 32-битных инструкции, в действительности размер инструкций в разных ISA может отличаться. Это характеризуется префиксом соответствующей команды. Сейчас, официально задокументированы только 16 и 32-битные инструкции.

Основные ISA RISC-V могут иметь как little-endian, так и big-endian модели памяти. Сами инструкции записаны последовательно так, чтобы при последовательном считывании по 32-х бит - команда не оказалась считанной частично.

## Регистры.

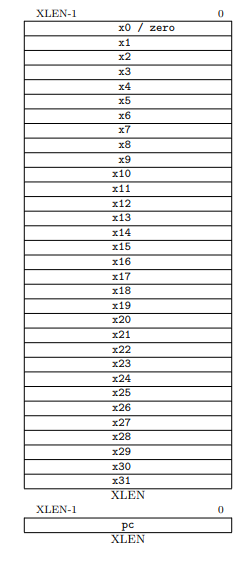


Рисунок 1 - Базовое состояние регистров.

Продолжая описывать ISA, мы переходим к регистрам. Аналогично предыдущим параметрам, количество регистров у RV32I также 32. В нулевом регистре **(zero)** все биты установлены в 0. Дополнительно имеется дополнительный регистр **pc** - содержащий адрес исполняемой инструкции.

## Формат инструкций.

Все инструкции описываются через 6 возможных типов. Основными считаются 4 (R/I/S/U), с дополнительными 2 (B/J) - они незначительно отличаются, поэтому названия иногда даже совмещают (SB/UJ).

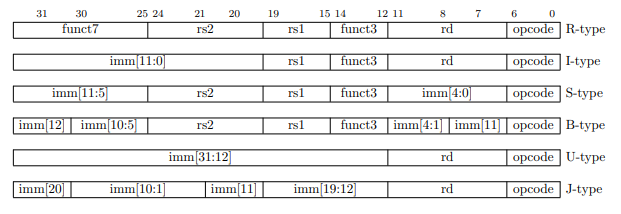


Рисунок 2 - Виды инструкций RISC-V ISA.

Можно заметить, что первые 7 бит *(opcode)* статичны независимо от вида инструкции. Верно! Именно по ним и можно различать вид, чтобы затем более детально ее разобрать. Также стоит заметить, что соответствующие поля *(rs1/rs2/rd) и (funct3/funct7)* остаются на одном месте независимо от вида.

Кстати, эти поля отвечают за регистры источника **(RS1/2 - Register Source 1/2)** и регистр назначения **(RD - Register Destination)**. А “*funct3/7*” отвечают за идентификацию инструкции.

**Imm** часто остается разбитым на кусочки, но в действительности — это одно дополнительное значение для инструкции: константа, адрес, офсет.

## 

## Пример.

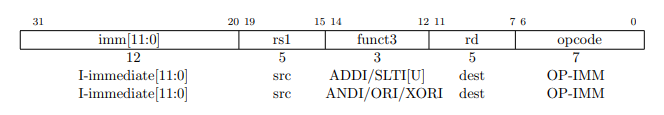


Рисунок 3 - Пример инструкции вида I.

Здесь исходя из *opcode* - мы понимаем вид. Основываясь на поле *funct3 -* мы узнаем инструкцию. Данная инструкции производят соответствующую операцию над регистром *rs1* и значением *imm,* и возвращают результат в *rd.*

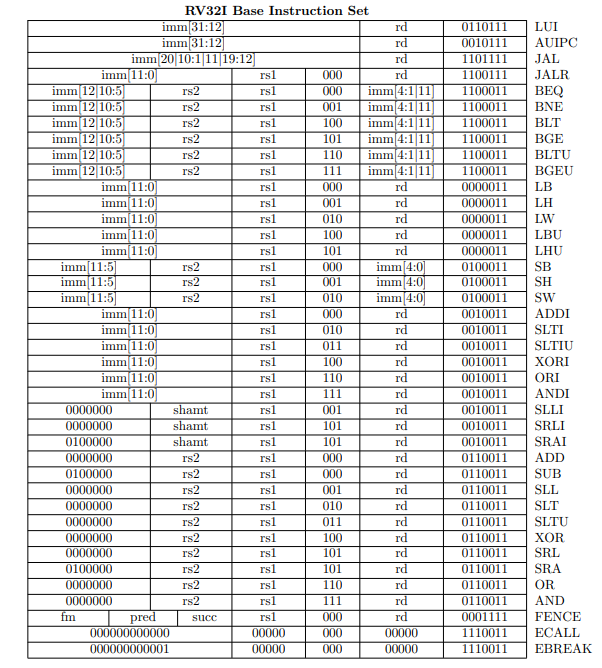


Рисунок 4 - Список используемых инструкций (1/2).

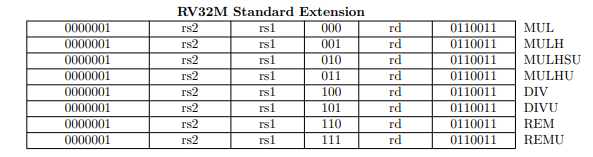


Рисунок 5 - Список используемых инструкций (2/2).

Имея информацию про каждую конкретную инструкцию, можно сопоставить любой набор из 32-битов любой команде. Используя битовые маски и уникальное для каждой команды значение, мы прогоняем рассматриваемую команду через маску и сравниваем с конкретным значением, чтобы определить вид команды.

# 

# Структура elf файла.

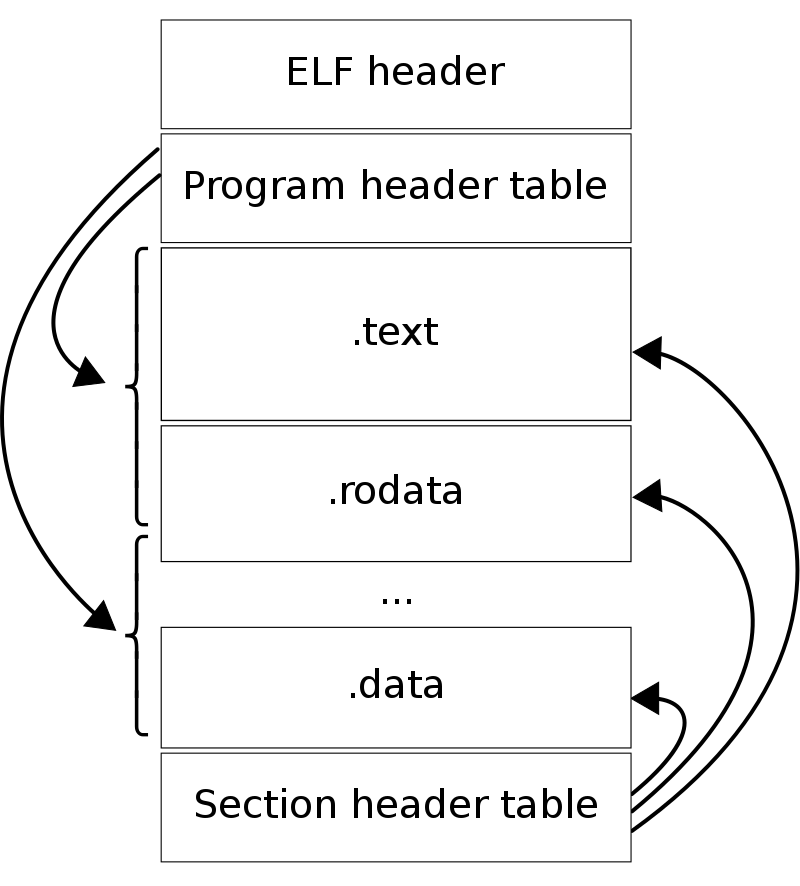


Рисунок 6 - Структура Elf файла.

Elf файл состоит из:

Заголовок файла (ELF Header) имеет фиксированное расположение в начале файла и содержит общее описание структуры файла и его основные характеристики, такие как: тип, версия формата, архитектура процессора, виртуальный адрес точки входа, размеры и смещения остальных частей файла.

Таблица заголовков программы - содержит заголовки, каждый из которых описывает отдельный сегмент программы и его атрибуты либо другую информацию, необходимую операционной системе для подготовки программы к исполнению. (но в данном случае мы ее будем игнорировать).

Таблица заголовков секций содержит атрибуты секций файла. Данная таблица необходима только компоновщику, исполняемые файлы в наличии этой таблицы не нуждаются (ELF загрузчик её игнорирует). Предоставленную в таблице заголовков секций информацию компоновщик использует для оптимального размещения данных секций по сегментам при сборке файла с учётом их атрибутов. Секции, которые мы обрабатываем — это: .text - подряд записанные инструкции. .symtab - таблица со статическими символами elf файла. .strtab - имена вхождений .symtab. И таблица с именами секций, чтобы иметь возможность их распознать.

# Описание работы написанного кода.

*disasm*

Данный файл содержит констаты (маски, названия регистров и уникальные биты для каждой команды). С их помощью производится анализ каждой отдельной команды, а затем такая информация, как: имя, регистры и формат помещаются в массив в таком-же порядке.

/elf

Содержит магические байты, объявления функций для парсинга. Функции парсинга, пытаются разбить работу на секции, а затем собрать все в одну структуру которую в дальнейшем выведет основная программа.

/main

Для тестов – не иммет смысла в ходе данной работы.

/rv3

Исполняемый файл. Принимает аргументы, запускает парсеры. После получения структуры elf выводит ее. Чтобы расставить метки – делает двойной проход. Поддерживат std::map для контроля адресного положения при выводе.

# Результат работы написанной программы.

*/test/test\_elf.disasm*

|  |
| --- |
| 1. SYMBOL TABLE 3. 00000000 l \*UND\* 00000000 \*UND\* 4. 00010074 l d .text 00000000 .text 5. 00011124 l d .bss 00000000 .bss 6. 00000000 l d .comment 00000000 .comment 7. 00000000 l d .riscv.attributes 00000000 .riscv.attributes 8. 00000000 l df \*ABS\* 00000000 test.c 9. 00011924 g \*ABS\* 00000000 \_\_global\_pointer$ 10. 000118f4 g O .bss 00000320 b 11. 00011124 g .text 00000000 \_\_SDATA\_BEGIN\_\_ 12. 000100ac g F .text 00000078 mmul 13. 00000000 g \*UND\* 00000000 \_start 14. 00011124 g O .bss 00000640 c 15. 00011c14 g .bss 00000000 \_\_BSS\_END\_\_ 16. 00011124 g .bss 00000000 \_\_bss\_start 17. 00010074 g F .text 0000001c main 18. 00011124 g .text 00000000 \_\_DATA\_BEGIN\_\_ 19. 00011124 g .text 00000000 \_edata 20. 00011c14 g .bss 00000000 \_end 21. 00011764 g O .bss 00000190 a 23. Disassembly of section .text: 25. 00010074 <main>: 26. 10074: ff010113 addi sp,sp,-16 27. 10078: 00112623 sw ra,12(sp) 28. 1007c: 030000ef jal ra,0x100ac <mmul> 29. 10080: 00c12083 lw ra,12(sp) 30. 10084: 00000513 li a0,0 31. 10088: 01010113 addi sp,sp,16 32. 1008c: 00008067 ret 33. 10090: 00000013 nop 34. 10094: 00100137 lui sp,0x100 35. 10098: fddff0ef jal ra,0x10074 <main> 36. 1009c: 00050593 mv a1,a0 37. 100a0: 00a00893 li a7,10 38. 100a4: 0ff0000f fence 39. 100a8: 00000073 ecall 41. 000100ac <mmul>: 42. 100ac: 00011f37 lui t5,0x11 43. 100b0: 124f0513 addi a0,t5,292 44. 100b4: 65450513 addi a0,a0,1620 45. 100b8: 124f0f13 addi t5,t5,292 46. 100bc: e4018293 addi t0,gp,-448 47. 100c0: fd018f93 addi t6,gp,-48 48. 100c4: 02800e93 li t4,40 49. 100c8: fec50e13 addi t3,a0,-20, L2 50. 100cc: 000f0313 mv t1,t5 51. 100d0: 000f8893 mv a7,t6 52. 100d4: 00000813 li a6,0 53. 100d8: 00088693 mv a3,a7, L1 54. 100dc: 000e0793 mv a5,t3 55. 100e0: 00000613 li a2,0 56. 100e4: 00078703 lb a4,0(a5), L0 57. 100e8: 00069583 lh a1,0(a3) 58. 100ec: 00178793 addi a5,a5,1 59. 100f0: 02868693 addi a3,a3,40 60. 100f4: 02b70733 mul a4,a4,a1 61. 100f8: 00e60633 add a2,a2,a4 62. 100fc: fea794e3 bne a5,a0,0x100e4 <L3> 63. 10100: 00c32023 sw a2,0(t1) 64. 10104: 00280813 addi a6,a6,2 65. 10108: 00430313 addi t1,t1,4 66. 1010c: 00288893 addi a7,a7,2 67. 10110: fdd814e3 bne a6,t4,0x100d8 <L3> 68. 10114: 050f0f13 addi t5,t5,80 69. 10118: 01478513 addi a0,a5,20 70. 1011c: fa5f16e3 bne t5,t0,0x100c8 <L3> 71. 10120: 00008067 ret |

# Список источников.

# Листинг кода.

*/include/disasm.h*

|  |
| --- |
| 1. *#pragma once* 3. *#include <array>* 4. *#include <bitset>* 6. *namespace elf {* 7. *constexpr uint32\_t MATCH\_BEQ = 0x63;* 8. *constexpr uint32\_t MASK\_BEQ = 0x707f;* 9. *constexpr uint32\_t MATCH\_BNE = 0x1063;* 10. *constexpr uint32\_t MASK\_BNE = 0x707f;* 11. *constexpr uint32\_t MATCH\_BLT = 0x4063;* 12. *constexpr uint32\_t MASK\_BLT = 0x707f;* 13. *constexpr uint32\_t MATCH\_BGE = 0x5063;* 14. *constexpr uint32\_t MASK\_BGE = 0x707f;* 15. *constexpr uint32\_t MATCH\_BLTU = 0x6063;* 16. *constexpr uint32\_t MASK\_BLTU = 0x707f;* 17. *constexpr uint32\_t MATCH\_BGEU = 0x7063;* 18. *constexpr uint32\_t MASK\_BGEU = 0x707f;* 19. *constexpr uint32\_t MATCH\_JALR = 0x67;* 20. *constexpr uint32\_t MASK\_JALR = 0x707f;* 21. *constexpr uint32\_t MATCH\_JAL = 0x6f;* 22. *constexpr uint32\_t MASK\_JAL = 0x7f;* 23. *constexpr uint32\_t MATCH\_LUI = 0x37;* 24. *constexpr uint32\_t MASK\_LUI = 0x7f;* 25. *constexpr uint32\_t MATCH\_AUIPC = 0x17;* 26. *constexpr uint32\_t MASK\_AUIPC = 0x7f;* 27. *constexpr uint32\_t MATCH\_ADDI = 0x13;* 28. *constexpr uint32\_t MASK\_ADDI = 0x707f;* 29. *constexpr uint32\_t MATCH\_SLLI = 0x1013;* 30. *constexpr uint32\_t MASK\_SLLI = 0xfc00707f;* 31. *constexpr uint32\_t MATCH\_SLTI = 0x2013;* 32. *constexpr uint32\_t MASK\_SLTI = 0x707f;* 33. *constexpr uint32\_t MATCH\_SLTIU = 0x3013;* 34. *constexpr uint32\_t MASK\_SLTIU = 0x707f;* 35. *constexpr uint32\_t MATCH\_XORI = 0x4013;* 36. *constexpr uint32\_t MASK\_XORI = 0x707f;* 37. *constexpr uint32\_t MATCH\_SRLI = 0x5013;* 38. *constexpr uint32\_t MASK\_SRLI = 0xfc00707f;* 39. *constexpr uint32\_t MATCH\_SRAI = 0x40005013;* 40. *constexpr uint32\_t MASK\_SRAI = 0xfc00707f;* 41. *constexpr uint32\_t MATCH\_ORI = 0x6013;* 42. *constexpr uint32\_t MASK\_ORI = 0x707f;* 43. *constexpr uint32\_t MATCH\_ANDI = 0x7013;* 44. *constexpr uint32\_t MASK\_ANDI = 0x707f;* 45. *constexpr uint32\_t MATCH\_ADD = 0x33;* 46. *constexpr uint32\_t MASK\_ADD = 0xfe00707f;* 47. *constexpr uint32\_t MATCH\_SUB = 0x40000033;* 48. *constexpr uint32\_t MASK\_SUB = 0xfe00707f;* 49. *constexpr uint32\_t MATCH\_SLL = 0x1033;* 50. *constexpr uint32\_t MASK\_SLL = 0xfe00707f;* 51. *constexpr uint32\_t MATCH\_SLT = 0x2033;* 52. *constexpr uint32\_t MASK\_SLT = 0xfe00707f;* 53. *constexpr uint32\_t MATCH\_SLTU = 0x3033;* 54. *constexpr uint32\_t MASK\_SLTU = 0xfe00707f;* 55. *constexpr uint32\_t MATCH\_XOR = 0x4033;* 56. *constexpr uint32\_t MASK\_XOR = 0xfe00707f;* 57. *constexpr uint32\_t MATCH\_SRL = 0x5033;* 58. *constexpr uint32\_t MASK\_SRL = 0xfe00707f;* 59. *constexpr uint32\_t MATCH\_SRA = 0x40005033;* 60. *constexpr uint32\_t MASK\_SRA = 0xfe00707f;* 61. *constexpr uint32\_t MATCH\_OR = 0x6033;* 62. *constexpr uint32\_t MASK\_OR = 0xfe00707f;* 63. *constexpr uint32\_t MATCH\_AND = 0x7033;* 64. *constexpr uint32\_t MASK\_AND = 0xfe00707f;* 65. *constexpr uint32\_t MATCH\_ADDIW = 0x1b;* 66. *constexpr uint32\_t MASK\_ADDIW = 0x707f;* 67. *constexpr uint32\_t MATCH\_SLLIW = 0x101b;* 68. *constexpr uint32\_t MASK\_SLLIW = 0xfe00707f;* 69. *constexpr uint32\_t MATCH\_SRLIW = 0x501b;* 70. *constexpr uint32\_t MASK\_SRLIW = 0xfe00707f;* 71. *constexpr uint32\_t MATCH\_SRAIW = 0x4000501b;* 72. *constexpr uint32\_t MASK\_SRAIW = 0xfe00707f;* 73. *constexpr uint32\_t MATCH\_ADDW = 0x3b;* 74. *constexpr uint32\_t MASK\_ADDW = 0xfe00707f;* 75. *constexpr uint32\_t MATCH\_SUBW = 0x4000003b;* 76. *constexpr uint32\_t MASK\_SUBW = 0xfe00707f;* 77. *constexpr uint32\_t MATCH\_SLLW = 0x103b;* 78. *constexpr uint32\_t MASK\_SLLW = 0xfe00707f;* 79. *constexpr uint32\_t MATCH\_SRLW = 0x503b;* 80. *constexpr uint32\_t MASK\_SRLW = 0xfe00707f;* 81. *constexpr uint32\_t MATCH\_SRAW = 0x4000503b;* 82. *constexpr uint32\_t MASK\_SRAW = 0xfe00707f;* 83. *constexpr uint32\_t MATCH\_LB = 0x3;* 84. *constexpr uint32\_t MASK\_LB = 0x707f;* 85. *constexpr uint32\_t MATCH\_LH = 0x1003;* 86. *constexpr uint32\_t MASK\_LH = 0x707f;* 87. *constexpr uint32\_t MATCH\_LW = 0x2003;* 88. *constexpr uint32\_t MASK\_LW = 0x707f;* 89. *constexpr uint32\_t MATCH\_LD = 0x3003;* 90. *constexpr uint32\_t MASK\_LD = 0x707f;* 91. *constexpr uint32\_t MATCH\_LBU = 0x4003;* 92. *constexpr uint32\_t MASK\_LBU = 0x707f;* 93. *constexpr uint32\_t MATCH\_LHU = 0x5003;* 94. *constexpr uint32\_t MASK\_LHU = 0x707f;* 95. *constexpr uint32\_t MATCH\_LWU = 0x6003;* 96. *constexpr uint32\_t MASK\_LWU = 0x707f;* 97. *constexpr uint32\_t MATCH\_SB = 0x23;* 98. *constexpr uint32\_t MASK\_SB = 0x707f;* 99. *constexpr uint32\_t MATCH\_SH = 0x1023;* 100. *constexpr uint32\_t MASK\_SH = 0x707f;* 101. *constexpr uint32\_t MATCH\_SW = 0x2023;* 102. *constexpr uint32\_t MASK\_SW = 0x707f;* 103. *constexpr uint32\_t MATCH\_SD = 0x3023;* 104. *constexpr uint32\_t MASK\_SD = 0x707f;* 105. *constexpr uint32\_t MATCH\_FENCE = 0xf;* 106. *constexpr uint32\_t MASK\_FENCE = 0x707f;* 107. *constexpr uint32\_t MATCH\_FENCE\_I = 0x100f;* 108. *constexpr uint32\_t MASK\_FENCE\_I = 0x707f;* 109. *constexpr uint32\_t MATCH\_ECALL = 0x73;* 110. *constexpr uint32\_t MASK\_ECALL = 0xffffffff;* 111. *constexpr uint32\_t MATCH\_EBREAK = 0x100073;* 112. *constexpr uint32\_t MASK\_EBREAK = 0xffffffff;* 114. *constexpr uint32\_t MATCH\_MUL = 0x2000033;* 115. *constexpr uint32\_t MASK\_MUL = 0xfe00707f;* 116. *constexpr uint32\_t MATCH\_MULH = 0x2001033;* 117. *constexpr uint32\_t MASK\_MULH = 0xfe00707f;* 118. *constexpr uint32\_t MATCH\_MULHSU = 0x2002033;* 119. *constexpr uint32\_t MASK\_MULHSU = 0xfe00707f;* 120. *constexpr uint32\_t MATCH\_MULHU = 0x2003033;* 121. *constexpr uint32\_t MASK\_MULHU = 0xfe00707f;* 122. *constexpr uint32\_t MATCH\_DIV = 0x2004033;* 123. *constexpr uint32\_t MASK\_DIV = 0xfe00707f;* 124. *constexpr uint32\_t MATCH\_DIVU = 0x2005033;* 125. *constexpr uint32\_t MASK\_DIVU = 0xfe00707f;* 126. *constexpr uint32\_t MATCH\_REM = 0x2006033;* 127. *constexpr uint32\_t MASK\_REM = 0xfe00707f;* 128. *constexpr uint32\_t MATCH\_REMU = 0x2007033;* 129. *constexpr uint32\_t MASK\_REMU = 0xfe00707f;* 131. *const std::array<std::string, 32> REGISTERS{* 132. *"zero", "ra", "sp", "gp", "tp", "t0", "t1", "t2", "s0", "s1", "a0",* 133. *"a1", "a2", "a3", "a4", "a5", "a6", "a7", "s2", "s3", "s4", "s5",* 134. *"s6", "s7", "s8", "s9", "s10", "s11", "t3", "t4", "t5", "t6"* 135. *};* 137. *using RawInst = uint32\_t;* 139. *struct Inst {* 140. *std::string name;* 141. *std::string rs1;* 142. *std::string rs2;* 143. *std::string rd;* 144. *int32\_t imm;* 145. *std::string fmt;* 146. *};* 148. *uint32\_t GetCsrImm(const RawInst& inst);* 149. *uint32\_t GetRD(const RawInst& inst);* 150. *uint32\_t GetRS1(const RawInst& inst);* 151. *uint32\_t GetRS2(const RawInst& inst);* 153. *inline uint32\_t Bextr(* 154. *uint32\_t src, uint32\_t start,* 155. *uint32\_t len* 156. *);* 158. *inline int32\_t Shamt(uint32\_t value);* 160. *inline uint32\_t ImmSign(uint32\_t value);* 162. *inline int32\_t GetBImm(uint32\_t value);* 164. *inline uint32\_t GetIImmUnsigned(uint32\_t value);* 166. *inline int32\_t GetIImm(uint32\_t value);* 168. *inline int32\_t GetSImm(uint32\_t value);* 170. *inline int32\_t GetJImm(uint32\_t value);* 172. *inline Inst Add(const RawInst& inst);* 174. *inline Inst Addi(const RawInst& inst);* 176. *inline Inst And(const RawInst& inst);* 178. *inline Inst Andi(const RawInst& inst);* 180. *struct InstructionU {* 181. *int opcode : 7;* 182. *int rd : 5;* 183. *int data : 20;* 184. *};* 186. *inline Inst Auipc(const RawInst& inst);* 188. *inline Inst Beq(const RawInst& inst, int32\_t pc);* 190. *inline Inst Bge(const RawInst& inst, int32\_t pc);* 192. *inline Inst Bgeu(const RawInst& inst, int32\_t pc);* 194. *inline Inst Blt(const RawInst& inst, int32\_t pc);* 196. *inline Inst Bltu(const RawInst& inst, int32\_t pc);* 198. *inline Inst Bne(const RawInst& inst, int32\_t pc);* 200. *inline Inst Fence(const RawInst& inst);* 202. *inline Inst Fencei(const RawInst& inst);* 204. *inline Inst Jal(const RawInst& inst, int32\_t pc);* 206. *inline Inst Jalr(const RawInst& inst);* 208. *inline Inst Lb(const RawInst& inst);* 210. *inline Inst Lbu(const RawInst& inst);* 212. *inline Inst Lh(const RawInst& inst);* 214. *inline Inst Lhu(const RawInst& inst);* 216. *inline Inst Lui(const RawInst& inst);* 218. *inline Inst Lw(const RawInst& inst);* 220. *inline Inst Or(const RawInst& inst);* 222. *struct InstructionI {* 223. *int opcode : 7;* 224. *int rd : 5;* 225. *int func : 3;* 226. *int rs1 : 5;* 227. *int imm : 12;* 228. *};* 230. *inline Inst Ori(const RawInst& inst);* 232. *inline Inst Sb(const RawInst& inst);* 234. *inline Inst Sh(const RawInst& inst);* 236. *inline Inst Sll(const RawInst& inst);* 238. *struct InstructionIShift {* 239. *int opcode : 7;* 240. *int rd : 5;* 241. *int func : 3;* 242. *int rs1 : 5;* 243. *int shamt : 5;* 244. *int imm : 7;* 245. *};* 247. *inline Inst Slli(const RawInst& inst);* 249. *inline Inst Slt(const RawInst& inst);* 251. *inline Inst Sltu(const RawInst& inst);* 253. *inline Inst Slti(const RawInst& inst);* 255. *inline Inst Sltiu(const RawInst& inst);* 257. *inline Inst Sra(const RawInst& inst);* 259. *inline Inst Srai(const RawInst& inst);* 261. *inline Inst Srl(const RawInst& inst);* 263. *inline Inst Srli(const RawInst& inst);* 265. *inline Inst Sub(const RawInst& inst);* 267. *inline Inst Sw(const RawInst& inst);* 269. *inline Inst Xor(const RawInst& inst);* 271. *inline Inst Xori(const RawInst& inst);* 273. *inline Inst Mret(const RawInst& inst);* 275. *inline Inst Sret(const RawInst& inst);* 277. *inline Inst Uret(const RawInst& inst);* 279. *inline Inst SfenceVma(const RawInst& inst);* 281. *inline Inst Wfi(const RawInst& inst);* 283. *inline Inst ECall(const RawInst& inst);* 285. *inline Inst EBreak(const RawInst& inst);* 287. *inline Inst Mul(const RawInst& inst);* 289. *inline Inst Mulh(const RawInst& inst);* 291. *inline Inst Mulhu(const RawInst& inst);* 293. *inline Inst Mulhsu(const RawInst& inst);* 295. *inline Inst Rem(const RawInst& inst);* 297. *inline Inst Remu(const RawInst& inst);* 299. *inline Inst OpDiv(const RawInst& inst);* 301. *inline Inst Divu(const RawInst& inst);* 303. *Inst DiInst(const RawInst& inst);* 304. *} // namespace elf* |

*/include/elf.h*

|  |
| --- |
| 1. *#pragma once* 3. *#include <cstdint>* 4. *#include <filesystem>* 6. *#include "disasm.h"* 8. *namespace fs = std::filesystem;* 10. *namespace elf {* 11. *constexpr char MAGIC[4] = {0x7f, 'E', 'L', 'F'};* 13. *using Elf32\_Half = uint16\_t;* 15. *using Elf32\_Word = uint32\_t;* 16. *using Elf32\_Sword = int32\_t;* 18. *using Elf32\_Xword = uint64\_t;* 19. *using Elf32\_Sxword = int64\_t;* 21. *using Elf32\_Addr = int32\_t;* 23. *using Elf32\_Off = uint32\_t;* 25. *using Elf32\_Section = uint16\_t;* 27. *using Elf32\_Versym = Elf32\_Half;* 29. *constexpr uint32\_t EI\_NIDENT = 16;* 31. *struct Elf32\_Ehdr {* 32. *unsigned char e\_ident[EI\_NIDENT]; // Magic number and other info* 33. *Elf32\_Half e\_type; // Object file type* 34. *Elf32\_Half e\_machine; // Architecture* 35. *Elf32\_Word e\_version; // Object file version* 36. *Elf32\_Addr e\_entry; // Entry point virtual address* 37. *Elf32\_Off e\_phoff; // Program header table file offset* 38. *Elf32\_Off e\_shoff; // Section header table file offset* 39. *Elf32\_Word e\_flags; // Processor-specific flags* 40. *Elf32\_Half e\_ehsize; // ELF header size in bytes* 41. *Elf32\_Half e\_phentsize; // Program header table entry size* 42. *Elf32\_Half e\_phnum; // Program header table entry count* 43. *Elf32\_Half e\_shentsize; // Section header table entry size* 44. *Elf32\_Half e\_shnum; // Section header table entry count* 45. *Elf32\_Half e\_shstrndx; // Section header string table index* 46. *};* 48. *struct Elf32\_Shdr {* 49. *Elf32\_Word sh\_name; // Section name (string tbl index)* 50. *Elf32\_Word sh\_type; // Section type* 51. *Elf32\_Word sh\_flags; // Section flags* 52. *Elf32\_Addr sh\_addr; // Section virtual addr at execution* 53. *Elf32\_Off sh\_offset; // Section file offset* 54. *Elf32\_Word sh\_size; // Section size in bytes* 55. *Elf32\_Word sh\_link; // Link to another section* 56. *Elf32\_Word sh\_info; // Additional section information* 57. *Elf32\_Word sh\_addralign; // Section alignment* 58. *Elf32\_Word sh\_entsize; // Entry size if section holds table* 59. *};* 61. *struct Elf32\_Sym {* 62. *Elf32\_Word st\_name;* 63. *Elf32\_Addr st\_value;* 64. *Elf32\_Word st\_size;* 65. *unsigned char st\_info;* 66. *unsigned char st\_other;* 67. *Elf32\_Half st\_shndx;* 68. *};* 70. *struct Elf {* 71. *Elf32\_Ehdr header;* 72. *std::vector<Elf32\_Shdr> sections;* 73. *std::vector<Elf32\_Sym> symtab;* 74. *std::vector<char> shstr;* 75. *std::vector<char> strtab;* 76. *std::vector<std::pair<RawInst, Inst>> text;* 77. *};* 79. *bool Validate(const fs::path& path);* 81. *Elf Parse(const fs::path& path);* 83. *Elf32\_Ehdr ParseEhdr(std::istream& in);* 84. *std::vector<Elf32\_Shdr> ParseShdr(std::istream& in, Elf32\_Off e\_shoff, Elf32\_Half e\_shnum);* 85. *std::vector<Elf32\_Sym> ParseSym(std::istream& in, const Elf32\_Shdr& shdr);* 87. *std::vector<char> ParseShstr(std::istream& in, const Elf32\_Shdr& shstr);* 88. *std::vector<char> ParseStrtab(std::istream& in, const Elf32\_Shdr& strtab);* 89. *std::vector<std::pair<RawInst, Inst>> ParseText(std::istream& in, const Elf32\_Shdr& text);* 90. *};* |

*/lib/disasm.cpp*

|  |
| --- |
| 1. *#include <disasm.h>* 3. *#include <stdexcept>* 5. *uint32\_t elf::GetCsrImm(const RawInst& inst) {* 6. *return (inst >> 15) & 0x1f;* 7. *}* 9. *uint32\_t elf::GetRD(const RawInst& inst) {* 10. *return (inst >> 7) & 0x1f;* 11. *}* 13. *uint32\_t elf::GetRS1(const RawInst& inst) {* 14. *return (inst >> 15) & 0x1f;* 15. *}* 17. *uint32\_t elf::GetRS2(const RawInst& inst) {* 18. *return (inst >> 20) & 0x1f;* 19. *}* 21. *uint32\_t elf::Bextr(* 22. *const uint32\_t src, const uint32\_t start,* 23. *const uint32\_t len* 24. *) {* 25. *return (src >> start) & ((1 << len) - 1);* 26. *}* 28. *int32\_t elf::Shamt(const uint32\_t value) {* 29. *return Bextr(value, 20, 6);* 30. *}* 32. *uint32\_t elf::ImmSign(const uint32\_t value) {* 33. *const int sign = Bextr(value, 31, 1);* 34. *return sign == 1 ? static\_cast<uint32\_t>(-1) : 0;* 35. *}* 37. *int32\_t elf::GetBImm(const uint32\_t value) {* 38. *return (Bextr(value, 8, 4) << 1) + (Bextr(value, 25, 6) << 5) +* 39. *(Bextr(value, 7, 1) << 11) + (ImmSign(value) << 12);* 40. *}* 42. *uint32\_t elf::GetIImmUnsigned(const uint32\_t value) {* 43. *return value >> 20;* 44. *}* 46. *int32\_t elf::GetIImm(const uint32\_t value) {* 47. *int val = Bextr(value, 20, 12);* 48. *int sign = Bextr(value, 31, 1) == 1 ? -1 : 1;* 50. *if (sign == -1)* 51. *val |= (0xfffff000);* 52. *return val;* 53. *}* 55. *int32\_t elf::GetSImm(const uint32\_t value) {* 56. *int sign = Bextr(value, 31, 1) == 1 ? -1 : 1;* 57. *int val = Bextr(value, 7, 5) + (Bextr(value, 25, 7) << 5);* 59. *if (sign == -1)* 60. *val |= (0xfffff000);* 61. *return val;* 62. *}* 64. *int32\_t elf::GetJImm(const uint32\_t value) {* 65. *int sign = Bextr(value, 31, 1) == 1 ? -1 : 1;* 66. *int val = (Bextr(value, 21, 10) << 1) + (Bextr(value, 20, 1) << 11) +* 67. *(Bextr(value, 12, 8) << 12) + (Bextr(value, 31, 1) << 20);* 69. *if (sign == -1)* 70. *val |= (0xfff00000);* 71. *return val;* 72. *}* 74. *elf::Inst elf::Add(const RawInst& inst) {* 75. *return {* 76. *"add", REGISTERS[GetRS1(inst)], REGISTERS[GetRS2(inst)],* 77. *REGISTERS[GetRD(inst)], 0, "add****\t****%R,%1S,%2S"* 78. *};* 79. *}* 81. *elf::Inst elf::Addi(const RawInst& inst) {* 82. *const int32\_t imm = GetIImm(inst);* 83. *if (GetRS1(inst) == 0) {* 84. *if (GetRD(inst) == 0 && imm == 0) {* 85. *return {"nop", "", "", "", 0, "nop"};* 86. *}* 88. *return {"li", "", "", REGISTERS[GetRD(inst)], imm, "li****\t****%R,%I"};* 89. *}* 91. *if (imm == 0) {* 92. *return {"mv", REGISTERS[GetRS1(inst)], "", REGISTERS[GetRD(inst)], 0, "mv****\t****%R,%1S"};* 93. *}* 95. *return {"addi", REGISTERS[GetRS1(inst)], "", REGISTERS[GetRD(inst)], imm, "addi****\t****%R,%1S,%I"};* 96. *}* 98. *elf::Inst elf::And(const RawInst& inst) {* 99. *return {* 100. *"and", REGISTERS[GetRS1(inst)], REGISTERS[GetRS2(inst)],* 101. *REGISTERS[GetRD(inst)], 0, "and****\t****%R,%1S,%2S"* 102. *};* 103. *}* 105. *elf::Inst elf::Andi(const RawInst& inst) {* 106. *const int32\_t imm = GetIImm(inst);* 108. *return {"andi", REGISTERS[GetRS1(inst)], "", REGISTERS[GetRD(inst)], imm, "andi****\t****%R,0x%X"};* 109. *}* 111. *elf::Inst elf::Auipc(const RawInst& inst) {* 112. *const auto\* in = reinterpret\_cast<const InstructionU\*>(&inst);* 113. *return {* 114. *"auipc", "", "", REGISTERS[GetRD(inst)],* 115. *(in->data << 12 >> 12) & 0xfffff, "auipc****\t****%R,0x%X"* 116. *};* 117. *}* 119. *elf::Inst elf::Beq(const RawInst& inst, const int32\_t pc) {* 120. *if (GetRS2(inst) == 0) {* 121. *return {* 122. *"begz", REGISTERS[GetRS1(inst)], "", "", GetBImm(inst) + pc, "beqz****\t****%1S,0x%X"* 124. *};* 125. *}* 127. *return {* 128. *"beq", REGISTERS[GetRS1(inst)], REGISTERS[GetRS2(inst)], "",* 129. *GetBImm(inst) + pc, "beqz****\t****%1S,%2S,0x%X"* 130. *};* 131. *}* 133. *elf::Inst elf::Bge(const RawInst& inst, const int32\_t pc) {* 134. *if (GetRS2(inst) == 0) {* 135. *return {"bgez", REGISTERS[GetRS1(inst)], "", "", GetBImm(inst) + pc, "bgez****\t****%1S,0x%X"};* 136. *}* 137. *if (GetRS1(inst) == 0) {* 138. *return {"blez", "", REGISTERS[GetRS2(inst)], "", GetBImm(inst) + pc, "blez****\t****%2S,0x%X"};* 139. *}* 140. *return {* 141. *"bge", REGISTERS[GetRS1(inst)], REGISTERS[GetRS2(inst)], "",* 142. *GetBImm(inst) + pc, "bge****\t****%1S,%2S,0x%X"* 143. *};* 144. *}* 146. *elf::Inst elf::Bgeu(const RawInst& inst, const int32\_t pc) {* 147. *return {* 148. *"bgeu", REGISTERS[GetRS1(inst)], REGISTERS[GetRS2(inst)], "",* 149. *GetBImm(inst) + pc, "bgeu****\t****%1S,%2S,0x%X"* 150. *};* 151. *}* 153. *elf::Inst elf::Blt(const RawInst& inst, const int32\_t pc) {* 154. *if (GetRS2(inst) == 0) {* 155. *return {"bltz", REGISTERS[GetRS1(inst)], "", "", GetBImm(inst) + pc, "bltz****\t****%1S,0x%X"};* 156. *}* 158. *if (GetRS1(inst) == 0) {* 159. *return {"bgtz", "", REGISTERS[GetRS2(inst)], "", GetBImm(inst) + pc, "bgtz****\t****%2S,0x%X"};* 160. *}* 162. *return {* 163. *"blt", REGISTERS[GetRS1(inst)], REGISTERS[GetRS2(inst)], "",* 164. *GetBImm(inst) + pc, "blt****\t****%1S,%2S,0x%X"* 165. *};* 166. *}* 168. *elf::Inst elf::Bltu(const RawInst& inst, const int32\_t pc) {* 169. *return {* 170. *"bltu", REGISTERS[GetRS1(inst)], REGISTERS[GetRS2(inst)], "",* 171. *GetBImm(inst) + pc, "bltu****\t****%1S,%2S,0x%X"* 172. *};* 173. *}* 175. *elf::Inst elf::Bne(const RawInst& inst, const int32\_t pc) {* 176. *if (GetRS2(inst) == 0) {* 177. *return {"bnez", REGISTERS[GetRS1(inst)], "", "", GetBImm(inst) + pc, "bnez****\t****%1S,0x%X"};* 178. *}* 180. *return {* 181. *"bne", REGISTERS[GetRS1(inst)], REGISTERS[GetRS2(inst)], "",* 182. *GetBImm(inst) + pc, "bne****\t****%1S,%2S,0x%0X"* 183. *};* 184. *}* 186. *elf::Inst elf::Fence(const RawInst& inst) {* 187. *return {"fence", "", "", "", 0, "fence"};* 188. *}* 190. *elf::Inst elf::Fencei(const RawInst& inst) {* 191. *return {"fence.i", "", "", "", 0, "fence.i"};* 192. *}* 194. *elf::Inst elf::Jal(const RawInst& inst, const int32\_t pc) {* 195. *const int32\_t offset = GetJImm(inst);* 196. *const int32\_t rd = GetRD(inst);* 197. *if (rd == 0) {* 198. *return {"j", "", "", "", offset + pc, "j****\t****0x%0X"};* 199. *}* 201. *return {"jal", "", "", REGISTERS[rd], offset + pc, "jal****\t****%R,0x%0X"};* 202. *}* 204. *elf::Inst elf::Jalr(const RawInst& inst) {* 205. *const int32\_t offset = GetIImm(inst);* 206. *if (offset == 0 && GetRD(inst) == 0 && GetRS1(inst) == 1) {* 207. *return {"ret", "", "", "", 0, "ret"};* 208. *}* 210. *if (offset == 0 && GetRD(inst) == 1) {* 211. *return {"jalr", REGISTERS[GetRS1(inst)], "", "", 0, "jalr****\t****%1S"};* 212. *}* 213. *if (offset == 0 && GetRD(inst) == 0) {* 214. *return {"jr", REGISTERS[GetRS1(inst)], "", "", 0, "jr****\t****%1S"};* 215. *}* 216. *if (GetRD(inst) == GetRS1(inst)) {* 217. *return {"jalr", REGISTERS[GetRS1(inst)], "", "", offset, "jalr****\t****%I(%1S)"};* 218. *}* 220. *return {"jalr", REGISTERS[GetRS1(inst)], "", REGISTERS[GetRD(inst)], offset, "jalr****\t****%R,%1S,%I"};* 221. *}* 223. *elf::Inst elf::Lb(const RawInst& inst) {* 224. *const int32\_t offset = GetIImm(inst);* 226. *return {"lb", REGISTERS[GetRS1(inst)], "", REGISTERS[GetRD(inst)], offset, "lb****\t****%R,%I(%1S)"};* 227. *}* 229. *elf::Inst elf::Lbu(const RawInst& inst) {* 230. *const int32\_t offset = GetIImm(inst);* 232. *return {"lbu", REGISTERS[GetRS1(inst)], "", REGISTERS[GetRD(inst)], offset, "lbu****\t****%R,%I(%1S)"};* 233. *}* 235. *elf::Inst elf::Lh(const RawInst& inst) {* 236. *const int32\_t offset = GetIImm(inst);* 238. *return {"lh", REGISTERS[GetRS1(inst)], "", REGISTERS[GetRD(inst)], offset, "lh****\t****%R,%I(%1S)"};* 239. *}* 241. *elf::Inst elf::Lhu(const RawInst& inst) {* 242. *const int32\_t offset = GetIImm(inst);* 244. *return {"lhu", REGISTERS[GetRS1(inst)], "", REGISTERS[GetRD(inst)], offset, "lhu****\t****%R,%I(%1S)"};* 245. *}* 247. *elf::Inst elf::Lui(const RawInst& inst) {* 248. *const auto\* in = reinterpret\_cast<const InstructionU\*>(&inst);* 249. *return {* 250. *"lui", "", "", REGISTERS[GetRD(inst)],* 251. *in->data << 12 >> 12 & 0xfffff, "lui****\t****%R,0x%X"* 252. *};* 253. *}* 255. *elf::Inst elf::Lw(const RawInst& inst) {* 256. *const int32\_t offset = GetIImm(inst);* 258. *return {"lw", REGISTERS[GetRS1(inst)], "", REGISTERS[GetRD(inst)], offset, "lw****\t****%R,%I(%1S)"};* 259. *}* 261. *elf::Inst elf::Or(const RawInst& inst) {* 262. *return {* 263. *"or", REGISTERS[GetRS1(inst)], REGISTERS[GetRS2(inst)],* 264. *REGISTERS[GetRD(inst)], 0, "or****\t****%R,%1S,%2S"* 265. *};* 266. *}* 268. *elf::Inst elf::Ori(const RawInst& inst) {* 269. *const auto\* in = reinterpret\_cast<const InstructionI\*>(&inst);* 271. *return {"ori", REGISTERS[GetRS1(inst)], "", REGISTERS[GetRD(inst)], in->imm, "ori****\t****%R,%1S,%I"};* 272. *}* 274. *elf::Inst elf::Sb(const RawInst& inst) {* 275. *const int32\_t offset = GetSImm(inst);* 277. *return {"sb", REGISTERS[GetRS1(inst)], REGISTERS[GetRS2(inst)], "", offset, "sb****\t****%2S,%I(%1S)"};* 278. *}* 280. *elf::Inst elf::Sh(const RawInst& inst) {* 281. *const int32\_t offset = GetSImm(inst);* 283. *return {"sh", REGISTERS[GetRS1(inst)], REGISTERS[GetRS2(inst)], "", offset, "sh****\t****%2S,%I(%1S)"};* 284. *}* 286. *elf::Inst elf::Sll(const RawInst& inst) {* 287. *return {* 288. *"sll", REGISTERS[GetRS1(inst)], REGISTERS[GetRS2(inst)],* 289. *REGISTERS[GetRD(inst)], 0, "sll****\t****%R,%1S,%2S"* 290. *};* 291. *}* 293. *elf::Inst elf::Slli(const RawInst& inst) {* 294. *const auto\* in = reinterpret\_cast<const InstructionIShift\*>(&inst);* 296. *return {* 297. *"slli", REGISTERS[GetRS1(inst)], "", REGISTERS[GetRD(inst)],* 298. *in->shamt & 0x1F, "slli****\t****%R,%1S,0x%X"* 299. *};* 300. *}* 302. *elf::Inst elf::Slt(const RawInst& inst) {* 303. *if (GetRS1(inst) == 0) {* 304. *return {"sgtz", "", REGISTERS[GetRS2(inst)], REGISTERS[GetRD(inst)], 0, "sgtz****\t****%R,%2S"};* 305. *}* 307. *return {* 308. *"slt", REGISTERS[GetRS1(inst)], REGISTERS[GetRS2(inst)],* 309. *REGISTERS[GetRD(inst)], 0, "slt****\t****%R,%1S,%2S"* 310. *};* 311. *}* 313. *elf::Inst elf::Sltu(const RawInst& inst) {* 314. *if (GetRS1(inst) == 0) {* 315. *return {"snez", "", REGISTERS[GetRS2(inst)], REGISTERS[GetRD(inst)], 0, "snez****\t****%R,%2S"};* 316. *}* 317. *return {* 318. *"sltu", REGISTERS[GetRS1(inst)], REGISTERS[GetRS2(inst)],* 319. *REGISTERS[GetRD(inst)], 0, "sltu****\t****%R,%1S,%2S"* 320. *};* 321. *}* 323. *elf::Inst elf::Slti(const RawInst& inst) {* 324. *const auto\* in = reinterpret\_cast<const InstructionI\*>(&inst);* 326. *return {"slti", REGISTERS[GetRS1(inst)], "", REGISTERS[GetRD(inst)], in->imm, "slti****\t****%R,%1S,%I"};* 327. *}* 329. *elf::Inst elf::Sltiu(const RawInst& inst) {* 330. *const auto\* in = reinterpret\_cast<const InstructionI\*>(&inst);* 331. *if (in->imm == 1) {* 332. *return {"seqz", REGISTERS[GetRS1(inst)], "", REGISTERS[GetRD(inst)], 0, "seqz****\t****%R,%1S"};* 333. *}* 335. *return {* 336. *"sltiu", REGISTERS[GetRS1(inst)], "", REGISTERS[GetRD(inst)],* 337. *in->imm, "sltiu****\t****%R,%1S,%I"* 338. *};* 339. *}* 341. *elf::Inst elf::Sra(const RawInst& inst) {* 342. *return {* 343. *"sra", REGISTERS[GetRS1(inst)], REGISTERS[GetRS2(inst)],* 344. *REGISTERS[GetRD(inst)], 0, "sra****\t****%R,%1S,%2S"* 345. *};* 346. *}* 348. *elf::Inst elf::Srai(const RawInst& inst) {* 349. *const auto\* in = reinterpret\_cast<const InstructionIShift\*>(&inst);* 351. *return {* 352. *"srai", REGISTERS[GetRS1(inst)], "", REGISTERS[GetRD(inst)],* 353. *in->shamt & 0x1F, "srai****\t****%R,%1S,0x%X"* 354. *};* 355. *}* 357. *elf::Inst elf::Srl(const RawInst& inst) {* 358. *return {* 359. *"srl", REGISTERS[GetRS1(inst)], REGISTERS[GetRS2(inst)],* 360. *REGISTERS[GetRD(inst)], 0, "srl****\t****%R,%1S,%2S"* 361. *};* 362. *}* 364. *elf::Inst elf::Srli(const RawInst& inst) {* 365. *const auto\* in = reinterpret\_cast<const InstructionIShift\*>(&inst);* 367. *return {* 368. *"srli", REGISTERS[GetRS1(inst)], "", REGISTERS[GetRD(inst)],* 369. *in->shamt & 0x1F, "srli****\t****%R,%1S,0x%X"* 370. *};* 371. *}* 373. *elf::Inst elf::Sub(const RawInst& inst) {* 374. *if (GetRS1(inst) == 0) {* 375. *return {"neg", "", REGISTERS[GetRS2(inst)], REGISTERS[GetRD(inst)], 0, "neg****\t****%R,%2S"};* 376. *}* 378. *return {* 379. *"sub", REGISTERS[GetRS1(inst)], REGISTERS[GetRS2(inst)],* 380. *REGISTERS[GetRD(inst)], 0, "sub****\t****%R,%1S,%2S"* 381. *};* 382. *}* 384. *elf::Inst elf::Sw(const RawInst& inst) {* 385. *int32\_t offset = GetSImm(inst);* 387. *return {"sw", REGISTERS[GetRS1(inst)], REGISTERS[GetRS2(inst)], "", offset, "sw****\t****%2S,%I(%1S)"};* 388. *}* 390. *elf::Inst elf::Xor(const RawInst& inst) {* 391. *return {* 392. *"xor", REGISTERS[GetRS1(inst)], REGISTERS[GetRS2(inst)],* 393. *REGISTERS[GetRD(inst)], 0, "xor****\t****%R,%1S,%2S"* 394. *};* 395. *}* 397. *elf::Inst elf::Xori(const RawInst& inst) {* 398. *const auto\* in = reinterpret\_cast<const InstructionI\*>(&inst);* 399. *if (in->imm == -1) {* 400. *return {"not", REGISTERS[GetRS1(inst)], "", REGISTERS[GetRD(inst)], 0, "not****\t****%R,%1S"};* 401. *}* 403. *return {"xori", REGISTERS[GetRS1(inst)], "", REGISTERS[GetRD(inst)], in->imm, "xori****\t****%R,%1S,%I"};* 404. *}* 406. *elf::Inst elf::Mret(const RawInst& inst) {* 407. *return {"mret", "", "", "", 0, "mret"};* 408. *}* 410. *elf::Inst elf::Sret(const RawInst& inst) {* 411. *return {"sret", "", "", "", 0, "sret"};* 412. *}* 414. *elf::Inst elf::Uret(const RawInst& inst) {* 415. *return {"uret", "", "", "", 0, "uret"};* 416. *}* 418. *elf::Inst elf::SfenceVma(const RawInst& inst) {* 419. *return {"sfence.vma", "", "", "", 0, "sfence.vma"};* 420. *}* 422. *elf::Inst elf::Wfi(const RawInst& inst) {* 423. *return {"wfi", "", "", "", 0, "wfi"};* 424. *}* 426. *elf::Inst elf::ECall(const RawInst& inst) {* 427. *return {* 428. *"ecall",* 429. *"",* 430. *"",* 431. *"",* 432. *0,* 433. *"ecall"* 434. *};* 435. *}* 437. *elf::Inst elf::EBreak(const RawInst& inst) {* 438. *return {* 439. *"ebreak",* 440. *"",* 441. *"",* 442. *"",* 443. *0,* 444. *"ebreak"* 445. *};* 446. *}* 448. *elf::Inst elf::Mul(const RawInst& inst) {* 449. *return {* 450. *"mul",* 451. *REGISTERS[GetRS1(inst)],* 452. *REGISTERS[GetRS2(inst)],* 453. *REGISTERS[GetRD(inst)],* 454. *0,* 455. *"mul****\t****%R,%1S,%2S"* 456. *};* 457. *}* 459. *elf::Inst elf::Mulh(const RawInst& inst) {* 460. *return {* 461. *"mulh",* 462. *REGISTERS[GetRS1(inst)],* 463. *REGISTERS[GetRS2(inst)],* 464. *REGISTERS[GetRD(inst)],* 465. *0,* 466. *"mulh****\t****%R,%1S,%2S"* 467. *};* 468. *}* 470. *elf::Inst elf::Mulhu(const RawInst& inst) {* 471. *return {* 472. *"mulhu",* 473. *REGISTERS[GetRS1(inst)],* 474. *REGISTERS[GetRS2(inst)],* 475. *REGISTERS[GetRD(inst)],* 476. *0,* 477. *"mulhu****\t****%R,%1S,%2S"* 478. *};* 479. *}* 481. *elf::Inst elf::Mulhsu(const RawInst& inst) {* 482. *return {* 483. *"mulhsu",* 484. *REGISTERS[GetRS1(inst)],* 485. *REGISTERS[GetRS2(inst)],* 486. *REGISTERS[GetRD(inst)],* 487. *0,* 488. *"mulhsu****\t****%R,%1S,%2S"* 489. *};* 490. *}* 492. *elf::Inst elf::Rem(const RawInst& inst) {* 493. *return {* 494. *"rem",* 495. *REGISTERS[GetRS1(inst)],* 496. *REGISTERS[GetRS2(inst)],* 497. *REGISTERS[GetRD(inst)],* 498. *0,* 499. *"rem****\t****%R,%1S,%2S"* 500. *};* 501. *}* 503. *elf::Inst elf::Remu(const RawInst& inst) {* 504. *return {* 505. *"remu",* 506. *REGISTERS[GetRS1(inst)],* 507. *REGISTERS[GetRS2(inst)],* 508. *REGISTERS[GetRD(inst)],* 509. *0,* 510. *"remu****\t****%R,%1S,%2S"* 511. *};* 512. *}* 514. *elf::Inst elf::OpDiv(const RawInst& inst) {* 515. *return {* 516. *"div",* 517. *REGISTERS[GetRS1(inst)],* 518. *REGISTERS[GetRS2(inst)],* 519. *REGISTERS[GetRD(inst)],* 520. *0,* 521. *"div****\t****%R,%1S,%2S"* 522. *};* 523. *}* 525. *elf::Inst elf::Divu(const RawInst& inst) {* 526. *return {* 527. *"divu",* 528. *REGISTERS[GetRS1(inst)],* 529. *REGISTERS[GetRS2(inst)],* 530. *REGISTERS[GetRD(inst)],* 531. *0,* 532. *"divu****\t****%R,%1S,%2S"* 533. *};* 534. *}* 536. *elf::Inst elf::DiInst(const RawInst& inst) {* 537. *if ((inst & MASK\_BEQ) == MATCH\_BEQ) {* 538. *return Beq(inst, 0);* 539. *}* 540. *if ((inst & MASK\_BNE) == MATCH\_BNE) {* 541. *return Bne(inst, 0);* 542. *}* 543. *if ((inst & MASK\_BLT) == MATCH\_BLT) {* 544. *return Blt(inst, 0);* 545. *}* 546. *if ((inst & MASK\_BGE) == MATCH\_BGE) {* 547. *return Bge(inst, 0);* 548. *}* 549. *if ((inst & MASK\_BLTU) == MATCH\_BLTU) {* 550. *return Bltu(inst, 0);* 551. *}* 552. *if ((inst & MASK\_BGEU) == MATCH\_BGEU) {* 553. *return Bgeu(inst, 0);* 554. *}* 555. *if ((inst & MASK\_JALR) == MATCH\_JALR) {* 556. *return Jalr(inst);* 557. *}* 558. *if ((inst & MASK\_JAL) == MATCH\_JAL) {* 559. *return Jal(inst, 0);* 560. *}* 561. *if ((inst & MASK\_LUI) == MATCH\_LUI) {* 562. *return Lui(inst);* 563. *}* 564. *if ((inst & MASK\_AUIPC) == MATCH\_AUIPC) {* 565. *return Auipc(inst);* 566. *}* 567. *if ((inst & MASK\_ADDI) == MATCH\_ADDI) {* 568. *return Addi(inst);* 569. *}* 570. *if ((inst & MASK\_SLLI) == MATCH\_SLLI) {* 571. *return Slli(inst);* 572. *}* 573. *if ((inst & MASK\_SLTI) == MATCH\_SLTI) {* 574. *return Slti(inst);* 575. *}* 576. *if ((inst & MASK\_SLTIU) == MATCH\_SLTIU) {* 577. *return Sltu(inst);* 578. *}* 579. *if ((inst & MASK\_XORI) == MATCH\_XORI) {* 580. *return Xori(inst);* 581. *}* 582. *if ((inst & MASK\_SRLI) == MATCH\_SRLI) {* 583. *return Srli(inst);* 584. *}* 585. *if ((inst & MASK\_SRAI) == MATCH\_SRAI) {* 586. *return Srai(inst);* 587. *}* 588. *if ((inst & MASK\_ORI) == MATCH\_ORI) {* 589. *return Ori(inst);* 590. *}* 591. *if ((inst & MASK\_ANDI) == MATCH\_ANDI) {* 592. *return Andi(inst);* 593. *}* 594. *if ((inst & MASK\_ADD) == MATCH\_ADD) {* 595. *return Add(inst);* 596. *}* 597. *if ((inst & MASK\_SUB) == MATCH\_SUB) {* 598. *return Sub(inst);* 599. *}* 600. *if ((inst & MASK\_SLL) == MATCH\_SLL) {* 601. *return Sll(inst);* 602. *}* 603. *if ((inst & MASK\_SLT) == MATCH\_SLT) {* 604. *return Slt(inst);* 605. *}* 606. *if ((inst & MASK\_SLTU) == MATCH\_SLTU) {* 607. *return Sltu(inst);* 608. *}* 609. *if ((inst & MASK\_XOR) == MATCH\_XOR) {* 610. *return Xor(inst);* 611. *}* 612. *if ((inst & MASK\_SRL) == MATCH\_SRL) {* 613. *return Srl(inst);* 614. *}* 615. *if ((inst & MASK\_SRA) == MATCH\_SRA) {* 616. *return Sra(inst);* 617. *}* 618. *if ((inst & MASK\_OR) == MATCH\_OR) {* 619. *return Or(inst);* 620. *}* 621. *if ((inst & MASK\_AND) == MATCH\_AND) {* 622. *return And(inst);* 623. *}* 624. *if ((inst & MASK\_ADDIW) == MATCH\_ADDIW) {* 625. *return Addi(inst);* 626. *}* 627. *if ((inst & MASK\_SLLIW) == MATCH\_SLLIW) {* 628. *return Sll(inst);* 629. *}* 630. *if ((inst & MASK\_SRLIW) == MATCH\_SRLIW) {* 631. *return Srli(inst);* 632. *}* 633. *if ((inst & MASK\_SRAIW) == MATCH\_SRAIW) {* 634. *return Srai(inst);* 635. *}* 636. *if ((inst & MASK\_ADDW) == MATCH\_ADDW) {* 637. *return Add(inst);* 638. *}* 639. *if ((inst & MASK\_SUBW) == MATCH\_SUBW) {* 640. *return Sub(inst);* 641. *}* 642. *if ((inst & MASK\_SLLW) == MATCH\_SLLW) {* 643. *return Sll(inst);* 644. *}* 645. *if ((inst & MASK\_SRLW) == MATCH\_SRLW) {* 646. *return Srl(inst);* 647. *}* 648. *if ((inst & MASK\_SRAW) == MATCH\_SRAW) {* 649. *return Sra(inst);* 650. *}* 651. *if ((inst & MASK\_LB) == MATCH\_LB) {* 652. *return Lb(inst);* 653. *}* 654. *if ((inst & MASK\_LH) == MATCH\_LH) {* 655. *return Lh(inst);* 656. *}* 657. *if ((inst & MASK\_LW) == MATCH\_LW) {* 658. *return Lw(inst);* 659. *}* 661. */\*if ((inst & MASK\_LD) == MATCH\_LD) {* 662. *return func(inst);* 663. *}\*/* 665. *if ((inst & MASK\_LBU) == MATCH\_LBU) {* 666. *return Lbu(inst);* 667. *}* 669. *if ((inst & MASK\_LHU) == MATCH\_LHU) {* 670. *return Lhu(inst);* 671. *}* 673. */\*if ((inst & MASK\_LWU) == MATCH\_LWU) {* 674. *return func(inst);* 675. *}\*/* 677. *if ((inst & MASK\_SB) == MATCH\_SB) {* 678. *return Sb(inst);* 679. *}* 680. *if ((inst & MASK\_SH) == MATCH\_SH) {* 681. *return Sh(inst);* 682. *}* 683. *if ((inst & MASK\_SW) == MATCH\_SW) {* 684. *return Sw(inst);* 685. *}* 687. */\*if ((inst & MASK\_SD) == MATCH\_SD) {* 688. *return func(inst);* 689. *}\*/* 691. *if ((inst & MASK\_FENCE) == MATCH\_FENCE) {* 692. *return Fence(inst);* 693. *}* 694. *if ((inst & MASK\_FENCE\_I) == MATCH\_FENCE\_I) {* 695. *return Fencei(inst);* 696. *}* 697. *if ((inst & MASK\_ECALL) == MATCH\_ECALL) {* 698. *return ECall(inst);* 699. *}* 700. *if ((inst & MASK\_EBREAK) == MATCH\_EBREAK) {* 701. *return EBreak(inst);* 702. *}* 704. *if ((inst & MASK\_MUL) == MATCH\_MUL) {* 705. *return Mul(inst);* 706. *}* 707. *if ((inst & MASK\_MULH) == MATCH\_MULH) {* 708. *return Mulh(inst);* 709. *}* 710. *if ((inst & MASK\_MULHSU) == MATCH\_MULHSU) {* 711. *return Mulhsu(inst);* 712. *}* 713. *if ((inst & MASK\_MULHU) == MATCH\_MULHU) {* 714. *return Mulhu(inst);* 715. *}* 716. *if ((inst & MASK\_DIV) == MATCH\_DIV) {* 717. *return OpDiv(inst);* 718. *}* 719. *if ((inst & MASK\_DIVU) == MATCH\_DIVU) {* 720. *return Divu(inst);* 721. *}* 722. *if ((inst & MASK\_REM) == MATCH\_REM) {* 723. *return Rem(inst);* 724. *}* 725. *if ((inst & MASK\_REMU) == MATCH\_REMU) {* 726. *return Remu(inst);* 727. *}* 729. *throw std::invalid\_argument("elf::DiInst error: Passed instruction cannot be recognized.");* 730. *}* |

*/lib/elf.cpp*

|  |
| --- |
| 1. *#include <array>* 2. *#include <elf.h>* 3. *#include <fstream>* 5. *#include <disasm.h>* 7. *bool elf::Validate(const fs::path& path) {* 8. *const std::string error = "elf::Validate error: ";* 10. *constexpr char kElfClass = 1;* 11. *constexpr Elf32\_Half kEMachine = 243;* 13. *const Elf elf = Parse(path);* 15. *if (memcmp(MAGIC, elf.header.e\_ident, sizeof MAGIC) != 0) {* 16. *throw std::invalid\_argument(error + "Magic number is not compatible.");* 17. *}* 19. *if (elf.header.e\_ident[4] != kElfClass) {* 20. *throw std::invalid\_argument(error + "Elf file should be '32-bit object'.");* 21. *}* 23. *if (elf.header.e\_machine != kEMachine) {* 24. *throw std::invalid\_argument(error + "Elf file is not 'RISC-V ELF' compatible.");* 25. *}* 27. *return is\_regular\_file(path);* 28. *}* 30. *elf::Elf elf::Parse(const fs::path& path) {* 31. *std::ifstream in{path, std::ios::in | std::ios::binary};* 32. *in.exceptions(std::ios::failbit);* 34. *Elf result;* 36. *Elf32\_Ehdr header = ParseEhdr(in);* 37. *std::vector<Elf32\_Shdr> sections = ParseShdr(in, header.e\_shoff, header.e\_shnum);* 38. *std::vector<char> shstr = ParseShstr(in, sections[header.e\_shstrndx]);* 39. *std::vector<Elf32\_Sym> symtab;* 40. *std::vector<char> strtab;* 41. *std::vector<std::pair<RawInst, Inst>> text;* 43. *for (const auto& shdr : sections) {* 44. *if (std::string(shstr.data() + shdr.sh\_name) == ".symtab") {* 45. *symtab = ParseSym(in, shdr);* 46. *}* 48. *if (std::string(shstr.data() + shdr.sh\_name) == ".strtab") {* 49. *strtab = ParseStrtab(in, shdr);* 50. *}* 52. *if (std::string(shstr.data() + shdr.sh\_name) == ".text") {* 53. *text = ParseText(in, shdr);* 54. *}* 55. *}* 57. *return {header, sections, symtab, shstr, strtab, text};* 58. *}* 60. *elf::Elf32\_Ehdr elf::ParseEhdr(std::istream& in) {* 61. *std::array<char, sizeof(Elf32\_Ehdr)> buffer{};* 63. *in.read(buffer.data(), buffer.size());* 65. *const Elf32\_Ehdr& result = \*reinterpret\_cast<Elf32\_Ehdr\*>(buffer.data());* 67. *return result;* 68. *}* 70. *std::vector<elf::Elf32\_Shdr> elf::ParseShdr(std::istream& in, const Elf32\_Off e\_shoff, const Elf32\_Half e\_shnum) {* 71. *in.seekg(e\_shoff);* 73. *std::vector<Elf32\_Shdr> result;* 74. *result.reserve(e\_shnum);* 76. *std::array<char, sizeof(Elf32\_Shdr)> buffer{};* 78. *for (Elf32\_Half i = 0; i < e\_shnum; ++i) {* 79. *in.read(buffer.data(), buffer.size());* 81. *result.push\_back(\*reinterpret\_cast<Elf32\_Shdr\*>(buffer.data()));* 82. *}* 84. *return result;* 85. *}* 87. *std::vector<elf::Elf32\_Sym> elf::ParseSym(std::istream& in, const Elf32\_Shdr& shdr) {* 88. *in.seekg(shdr.sh\_offset);* 90. *std::vector<Elf32\_Sym> result;* 92. *std::array<char, sizeof(Elf32\_Sym)> buffer{};* 94. *for (Elf32\_Word i = 0; i < shdr.sh\_size / sizeof(Elf32\_Sym); ++i) {* 95. *in.read(buffer.data(), buffer.size());* 97. *result.push\_back(\*reinterpret\_cast<Elf32\_Sym\*>(buffer.data()));* 98. *}* 100. *return result;* 101. *}* 103. *std::vector<char> elf::ParseShstr(std::istream& in, const Elf32\_Shdr& shstr) {* 104. *in.seekg(shstr.sh\_offset);* 106. *std::vector<char> result(shstr.sh\_size);* 108. *in.read(result.data(), shstr.sh\_size);* 110. *return {result};* 111. *}* 113. *std::vector<char> elf::ParseStrtab(std::istream& in, const Elf32\_Shdr& strtab) {* 114. *in.seekg(strtab.sh\_offset);* 116. *std::vector<char> result(strtab.sh\_size);* 118. *in.read(result.data(), strtab.sh\_size);* 120. *return {result};* 121. *}* 123. *std::vector<std::pair<elf::RawInst, elf::Inst>> elf::ParseText(std::istream& in, const Elf32\_Shdr& text) {* 124. *in.seekg(text.sh\_offset);* 125. *RawInst inst = 0;* 127. *std::vector<std::pair<RawInst, Inst>> result;* 128. *result.reserve(text.sh\_size / sizeof(inst));* 130. *for (Elf32\_Word i = 0; i < result.capacity(); ++i) {* 131. *in.read(reinterpret\_cast<char\*>(&inst), sizeof(inst));* 133. *result.emplace\_back(inst, DiInst(inst));* 134. *}* 136. *return result;* 137. *}* 138. *#include <array>* 139. *#include <elf.h>* 140. *#include <fstream>* 142. *#include <disasm.h>* 144. *bool elf::Validate(const fs::path& path) {* 145. *const std::string error = "elf::Validate error: ";* 147. *constexpr char kElfClass = 1;* 148. *constexpr Elf32\_Half kEMachine = 243;* 150. *const Elf elf = Parse(path);* 152. *if (memcmp(MAGIC, elf.header.e\_ident, sizeof MAGIC) != 0) {* 153. *throw std::invalid\_argument(error + "Magic number is not compatible.");* 154. *}* 156. *if (elf.header.e\_ident[4] != kElfClass) {* 157. *throw std::invalid\_argument(error + "Elf file should be '32-bit object'.");* 158. *}* 160. *if (elf.header.e\_machine != kEMachine) {* 161. *throw std::invalid\_argument(error + "Elf file is not 'RISC-V ELF' compatible.");* 162. *}* 164. *return is\_regular\_file(path);* 165. *}* 167. *elf::Elf elf::Parse(const fs::path& path) {* 168. *std::ifstream in{path, std::ios::in | std::ios::binary};* 169. *in.exceptions(std::ios::failbit);* 171. *Elf result;* 173. *Elf32\_Ehdr header = ParseEhdr(in);* 174. *std::vector<Elf32\_Shdr> sections = ParseShdr(in, header.e\_shoff, header.e\_shnum);* 175. *std::vector<char> shstr = ParseShstr(in, sections[header.e\_shstrndx]);* 176. *std::vector<Elf32\_Sym> symtab;* 177. *std::vector<char> strtab;* 178. *std::vector<std::pair<RawInst, Inst>> text;* 180. *for (const auto& shdr : sections) {* 181. *if (std::string(shstr.data() + shdr.sh\_name) == ".symtab") {* 182. *symtab = ParseSym(in, shdr);* 183. *}* 185. *if (std::string(shstr.data() + shdr.sh\_name) == ".strtab") {* 186. *strtab = ParseStrtab(in, shdr);* 187. *}* 189. *if (std::string(shstr.data() + shdr.sh\_name) == ".text") {* 190. *text = ParseText(in, shdr);* 191. *}* 192. *}* 194. *return {header, sections, symtab, shstr, strtab, text};* 195. *}* 197. *elf::Elf32\_Ehdr elf::ParseEhdr(std::istream& in) {* 198. *std::array<char, sizeof(Elf32\_Ehdr)> buffer{};* 200. *in.read(buffer.data(), buffer.size());* 202. *const Elf32\_Ehdr& result = \*reinterpret\_cast<Elf32\_Ehdr\*>(buffer.data());* 204. *return result;* 205. *}* 207. *std::vector<elf::Elf32\_Shdr> elf::ParseShdr(std::istream& in, const Elf32\_Off e\_shoff, const Elf32\_Half e\_shnum) {* 208. *in.seekg(e\_shoff);* 210. *std::vector<Elf32\_Shdr> result;* 211. *result.reserve(e\_shnum);* 213. *std::array<char, sizeof(Elf32\_Shdr)> buffer{};* 215. *for (Elf32\_Half i = 0; i < e\_shnum; ++i) {* 216. *in.read(buffer.data(), buffer.size());* 218. *result.push\_back(\*reinterpret\_cast<Elf32\_Shdr\*>(buffer.data()));* 219. *}* 221. *return result;* 222. *}* 224. *std::vector<elf::Elf32\_Sym> elf::ParseSym(std::istream& in, const Elf32\_Shdr& shdr) {* 225. *in.seekg(shdr.sh\_offset);* 227. *std::vector<Elf32\_Sym> result;* 229. *std::array<char, sizeof(Elf32\_Sym)> buffer{};* 231. *for (Elf32\_Word i = 0; i < shdr.sh\_size / sizeof(Elf32\_Sym); ++i) {* 232. *in.read(buffer.data(), buffer.size());* 234. *result.push\_back(\*reinterpret\_cast<Elf32\_Sym\*>(buffer.data()));* 235. *}* 237. *return result;* 238. *}* 240. *std::vector<char> elf::ParseShstr(std::istream& in, const Elf32\_Shdr& shstr) {* 241. *in.seekg(shstr.sh\_offset);* 243. *std::vector<char> result(shstr.sh\_size);* 245. *in.read(result.data(), shstr.sh\_size);* 247. *return {result};* 248. *}* 250. *std::vector<char> elf::ParseStrtab(std::istream& in, const Elf32\_Shdr& strtab) {* 251. *in.seekg(strtab.sh\_offset);* 253. *std::vector<char> result(strtab.sh\_size);* 255. *in.read(result.data(), strtab.sh\_size);* 257. *return {result};* 258. *}* 260. *std::vector<std::pair<elf::RawInst, elf::Inst>> elf::ParseText(std::istream& in, const Elf32\_Shdr& text) {* 261. *in.seekg(text.sh\_offset);* 262. *RawInst inst = 0;* 264. *std::vector<std::pair<RawInst, Inst>> result;* 265. *result.reserve(text.sh\_size / sizeof(inst));* 267. *for (Elf32\_Word i = 0; i < result.capacity(); ++i) {* 268. *in.read(reinterpret\_cast<char\*>(&inst), sizeof(inst));* 270. *result.emplace\_back(inst, DiInst(inst));* 271. *}* 273. *return result;* 274. *}* |

*/bin/rv3.cpp*

|  |
| --- |
| 1. *#include "rv3.h"* 3. *#include <fstream>* 4. *#include <map>* 6. *std::pair<fs::path, fs::path> ParseArguments(const int argc, char\*\* argv) {* 7. *if (argc != 3) {* 8. *throw std::runtime\_error("rv3 error: Wrong amount of arguments (should be 3): " + std::to\_string(argc));* 9. *}* 11. *std::pair<fs::path, fs::path> result = {argv[1], argv[2]};* 13. *if (!elf::Validate(result.first)) {* 14. *throw std::invalid\_argument("rv3 error: Invalid argument was passed: " + result.first.string());* 15. *}* 17. *return result;* 18. *}* 20. *std::string ToHex(const int32\_t value, const bool padding = true) {* 21. *std::stringstream ss;* 23. *ss << std::hex << value;* 25. *std::string hex = ss.str();* 27. *return padding ? std::string(8 - ss.str().size(), '0') + ss.str() : ss.str();* 28. *}* 30. *std::string BindType(const unsigned char info) {* 31. *constexpr auto bind = [](const unsigned char info) {* 32. *switch (info >> 4) {* 33. *case 0:* 34. *return 'l';* 35. *case 1:* 36. *return 'g';* 37. *case 2:* 38. *return 'w';* 39. *case 13:* 40. *return 'L';* 41. *case 15:* 42. *return 'H';* 43. *default:* 44. *return ' ';* 45. *}* 46. *};* 47. *constexpr auto type = [](const unsigned char info) -> std::string {* 48. *switch (info & 0xf) {* 49. *case 0:* 50. *return " ";* 51. *case 1:* 52. *return " O";* 53. *case 2:* 54. *return " F";* 55. *case 3:* 56. *return "d ";* 57. *case 4:* 58. *return "df";* 59. *case 13:* 60. *return "LO";* 61. *case 15:* 62. *return "HI";* 63. *default:* 64. *return " ";* 65. *}* 66. *};* 68. *std::stringstream ss;* 70. *ss << bind(info) << std::string(4, ' ') << type(info);* 72. *return ss.str();* 73. *}* 75. *std::string Visibility(const unsigned char other) {* 76. *switch (other & 0x3) {* 77. *case 0:* 78. *return "Default";* 79. *case 1:* 80. *return "Internal";* 81. *case 2:* 82. *return "Hidden";* 83. *case 3:* 84. *return "Protected";* 85. *default:* 86. *return "";* 87. *}* 88. *}* 90. *std::string Association(const elf::Elf& elf, const elf::Elf32\_Half st\_shndx) {* 91. *if (st\_shndx >= elf.sections.size() || st\_shndx < 0) {* 92. *return "\*ABS\*";* 93. *}* 95. *std::string result{elf.shstr.data() + elf.sections[st\_shndx].sh\_name};* 97. *return result.empty() ? "\*UND\*" : result;* 98. *}* 100. *std::string StringifyIter(* 101. *const elf::Inst& inst, const elf::Elf32\_Addr addr, const std::map<elf::Elf32\_Addr, std::string>& symtab,* 102. *std::map<elf::Elf32\_Addr, std::string>& marks* 103. *) {* 104. *std::string result = inst.fmt;* 106. *if (const size\_t pos = result.find("%R"); pos != std::string::npos) {* 107. *result.replace(pos, 2, inst.rd);* 108. *}* 110. *if (const size\_t pos = result.find("%1S"); pos != std::string::npos) {* 111. *result.replace(pos, 3, inst.rs1);* 112. *}* 114. *if (const size\_t pos = result.find("%2S"); pos != std::string::npos) {* 115. *result.replace(pos, 3, inst.rs2);* 116. *}* 118. *if (const size\_t pos = result.find("%I"); pos != std::string::npos) {* 119. *result.replace(pos, 2, std::to\_string(inst.imm));* 120. *}* 122. *if (const size\_t pos = result.find("%0X"); pos != std::string::npos) {* 123. *std::stringstream ss;* 125. *ss << std::hex << addr + inst.imm;* 127. *std::string associate;* 128. *if (symtab.contains(addr + inst.imm)) {* 129. *associate = symtab.at(addr + inst.imm);* 130. *}* 131. *else {* 132. *associate = "L" + std::to\_string(marks.size());* 133. *marks[addr + inst.imm] = associate;* 134. *}* 136. *result.replace(pos, 3, ss.str() + " <" + associate + '>');* 137. *}* 139. *if (const size\_t pos = result.find("%X"); pos != std::string::npos) {* 140. *std::stringstream ss;* 142. *ss << std::hex << inst.imm;* 144. *result.replace(pos, 2, ss.str());* 145. *}* 147. *return result;* 148. *}* 150. *void SymtabOut(std::ostream& out, const elf::Elf& elf) {* 151. *out << "SYMBOL TABLE****\n\n****";* 153. *for (const auto& sym : elf.symtab) {* 154. *out << ToHex(sym.st\_value) << ' ';* 155. *out << BindType(sym.st\_info) << ' ';* 156. *out << Association(elf, sym.st\_shndx) << ' ';* 157. *out << ToHex(sym.st\_size) << ' ';* 158. *out << (sym.st\_name == 0 ? Association(elf, sym.st\_shndx) : std::string{elf.strtab.data() + sym.st\_name}) <<* 159. *'****\n****';* 160. *}* 162. *out << std::endl;* 163. *}* 165. *void DisAsmOut(std::ostream& out, const elf::Elf& elf) {* 166. *out << "Disassembly of section .text:****\n****";* 168. *std::map<elf::Elf32\_Addr, std::string> functions;* 169. *std::map<elf::Elf32\_Addr, std::string> symtab;* 170. *std::map<elf::Elf32\_Addr, std::string> marks;* 172. *for (const auto& sym : elf.symtab) {* 173. *if (BindType(sym.st\_info).back() == 'F') {* 174. *functions[sym.st\_value] = std::string{elf.strtab.data() + sym.st\_name};* 175. *}* 177. *symtab[sym.st\_value] = std::string{elf.strtab.data() + sym.st\_name};* 178. *}* 180. *auto addr\_iter = functions.begin();* 181. *elf::Elf32\_Addr addr = addr\_iter->first;* 183. *for (const auto& inst : elf.text) {* 184. *StringifyIter(inst.second, addr, symtab, marks);* 186. *addr += sizeof(elf::RawInst);* 187. *}* 189. *addr = addr\_iter->first;* 191. *for (const auto& inst : elf.text) {* 192. *if (addr\_iter != functions.end() && addr\_iter->first == addr) {* 193. *out << '****\n****';* 194. *out << ToHex(addr\_iter->first) << " <" << addr\_iter->second << ">:****\n****";* 196. *++addr\_iter;* 197. *}* 199. *out << '****\t****' << ToHex(addr, false) << ":****\t****" << ToHex(inst.first) << "****\t\t****" << StringifyIter(* 200. *inst.second, addr, symtab, marks* 201. *);* 203. *if (marks.contains(addr)) {* 204. *out << ", " << marks[addr];* 205. *}* 207. *out << '****\n****';* 209. *addr += sizeof(elf::RawInst);* 210. *}* 212. *out << std::endl;* 213. *}* 215. *void Print(const elf::Elf& data, const fs::path& path) {* 216. *std::ofstream out{path, std::ios::out};* 217. *out.exceptions(std::ios::failbit);* 219. *SymtabOut(out, data);* 220. *DisAsmOut(out, data);* 221. *}* 223. *int main(int argc, char\*\* argv) {* 224. *std::pair<fs::path, fs::path> args;* 226. *try {* 227. *args = ParseArguments(argc, argv);* 228. *}* 229. *catch (std::exception& err) {* 230. *std::cout << err.what();* 231. *}* 233. *auto data = elf::Parse(args.first);* 235. *try {* 236. *Print(data, args.second);* 237. *}* 238. *catch (std::exception& err) {* 239. *std::cout << err.what();* 240. *}* 242. *return 0;* 243. *}* |