RISC-V LAB

代码参考 y86 顺序模拟器。

1. RISC-V 简介

RISC-V 是一个最近诞生的指令集架构（它诞生于近十年，而大多数其他指令集都诞生与 20 世纪 70-80 年代），也是一个开源的指令集架构。

现代 x86 和 ARM 架构采用了增量 ISA，为了保持架构的向后兼容性使得架构的篇幅已有数千页。而新兴的 RISC-V 架构则具有后发优势，采用模块化的 ISA，其核心是名为 RV32I 的基础 ISA，可以在此基础上根据不同的应用场景扩展其他的模块，实现不同的要求。因此，它的架构篇幅并不大。

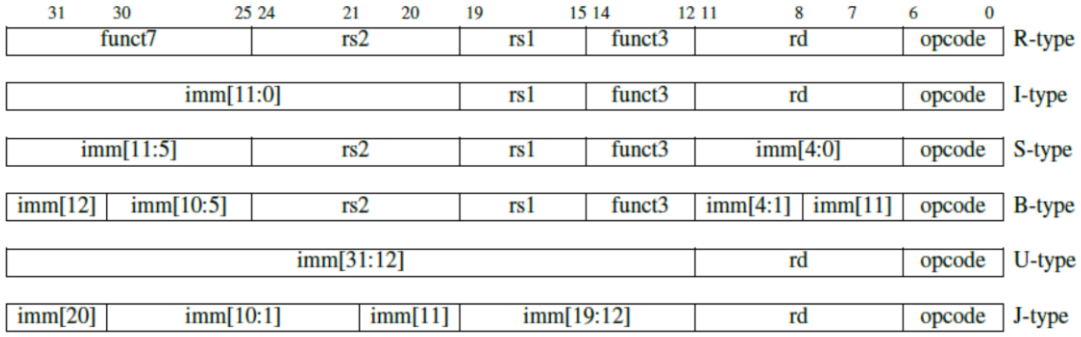
特性	x86或ARM架构	RISC-V
架构篇幅	数千页	少于300页
模块化	不支持	支持模块化可配置的指令子集
可扩展性	不支持	支持可扩展定制指令
指令数目	指令数繁多 不同的架构分支彼此不兼容	一套指令集支持所有架构。基本指令子集仅40余条指令，以此为共有基础，加上其他常用模块子集指令总指令数也仅几十条
易实现性	硬件实现得复杂度高	硬件设计与编译器实现非常简单 1) 仅支持小端格式 2) 存储器访问指令一次只访问一个元素 3) 去除存储器访问指令的地址自增自减模式 4) 规整的指令编码格式 5) 简化的分支跳转指令与静态预测机制 6) 不使用分支延迟槽（Delay Slot） 7) 不使用指令条件码（Conditional Code） 8) 运算指令的结果不产生异常（Exception） 9) 16位的压缩指令有其对应的普通32位指令 10) 不使用零开销硬件循环

X86 或 ARM 架构与 RISC-V 特性对比图

2. RV32I 基础指令集

RV32I 是 RISC-V 固定不变的基础整数指令集，是 RISC-V 的核心内容。其详细内容可以参考 RISC-V-Reader-Chinese-v2p1.pdf 的第二章。

RV32I 指令有六种格式，分别是：用于寄存器-寄存器操作的 R 类型指令，用于短立即数和访存 load 操作的 I 型指令，用于访存 store 操作的 S 型指令，用于条件跳转操作的 B 类型指令，用于长立即数的 U 型指令和用于无条件跳转的 J 型指令。



RV32I 指令格式

RV32I 指令格式有着以下四点优点。第一，RV32I 指令只有六种格式，并且所有的指令都是 32 位长，这简化了指令解码。第二，RISC-V 指令提供三个寄存器操作数，当一个操作天然就需要有三个不同的操作数时，编译器或者汇编程序程序员就可以少使用一条 move（搬运）指令，来保存目的寄存器的值。第三，在 RISC-V 中对于所有指令，要读写的寄存器的标识符总是在同一位置，意味着在解码指令之前，就可以先开始访问寄存器。第四，这些格式的立即数字段总是符号扩展，符号位总是在指令中最高位。

31 25	24 20	19 15	14 12	11 7	6 0	
imm[31:12]				rd	0110111	U lui
imm[31:12]				rd	0010111	U auipc
imm[20|10:1|11|19:12]				rd	1101111	J jal
imm[11:0]		rs1	000	rd	1100111	I jalr
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	B beq
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	B bne
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	B blt
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	B bge
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	B bltu
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	B bgeu
imm[11:0]		rs1	000	rd	0000011	I lb
imm[11:0]		rs1	001	rd	0000011	I lh
imm[11:0]		rs1	010	rd	0000011	I lw
imm[11:0]		rs1	100	rd	0000011	I lbu
imm[11:0]		rs1	101	rd	0000011	I lhu
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	S sb
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	S sh
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	S sw
imm[11:0]		rs1	000	rd	0010011	I addi
imm[11:0]		rs1	010	rd	0010011	I slti
imm[11:0]		rs1	011	rd	0010011	I sltiu
imm[11:0]		rs1	100	rd	0010011	I xori
imm[11:0]		rs1	110	rd	0010011	I ori
imm[11:0]		rs1	111	rd	0010011	I andi
0000000	shamt	rs1	001	rd	0010011	I slli
0000000	shamt	rs1	101	rd	0010011	I srli
0100000	shamt	rs1	101	rd	0010011	I srai
0000000	rs2	rs1	000	rd	0110011	R add
0100000	rs2	rs1	000	rd	0110011	R sub
0000000	rs2	rs1	001	rd	0110011	R sll
0000000	rs2	rs1	010	rd	0110011	R slt
0000000	rs2	rs1	011	rd	0110011	Rsltu
0000000	rs2	rs1	100	rd	0110011	R xor
0000000	rs2	rs1	101	rd	0110011	R srl
0100000	rs2	rs1	101	rd	0110011	R sra
0000000	rs2	rs1	110	rd	0110011	R or
0000000	rs2	rs1	111	rd	0110011	R and
0000 pred	succ	00000	000	00000	0001111	I fence
0000 0000	0000	00000	001	00000	0001111	I fence.i
000000000000		00000	000	00000	1110011	I ecall
000000000001		00000	000	00000	1110011	I ebreak
csr		rs1	001	rd	1110011	I csrrw
csr		rs1	010	rd	1110011	I csrrs
csr		rs1	011	rd	1110011	I csrrc
csr		zimm	101	rd	1110011	I csrrwi
csr		zimm	110	rd	1110011	I cssrrsi
csr		zimm	111	rd	1110011	I csrrci

3. RISC-V LAB 简介

在本实验中，你将了解一个 RISC-V 顺序模拟器的设计与实现，所用到的指令为 RV32I 中的部分指令。

本实验分为三个部分，三个部分内容相似，指在帮助你一点一点的深入了解顺序模拟器的设计与实现。

4. 材料说明

1. 你将获得一个名为 riscvlab.tar 的文件。
2. **输入命令** tar -xvf riscvlab.tar，你将获得一个目录包含以下文件：

   writeup.pdf, RISC-V-Reader-Chinese-v2p1.pdf, riscvlabppt.pdf, ssim-simple.c, sim.h, hcl.c, isa.c, isa.h, testparta.yo, testpartb.yo, testpartc.yo
3. **进入该目录**：cd riscvlab
4. 如果修改完毕，**请输入**：gcc -Wall -O2 -o ssim hcl.c ssim-simple.c isa.c

获得 ssim；**再输入**：./ssim -t *.yo（*.yo 为后缀为.yo 的文件）获得执行结果。

5. 与附录中的输出结果**进行对比**，以求证是否正确。

例如：unix> gcc -Wall -O2 -o ssim hcl.c ssim-simple.c isa.c

unix>./ssim -t testparta.yo

翻到附录比较输出结果是否正确。

5. PART A

添加指令 bne/ble/bge。

你需要修改的地方有两处：

1. lsa.c 文件中的 **instruction_set** 中。
2. ssim-simple 文件中的 **sim_step** 函数中

6. PART B

添加指令 addi/slti/sltiu/xori/ori/andi/slli/srli/srai。

你需要修改的地方有五处:

1. isa.c 文件中的 **instruction_set** 中。
2. isa.h 文件中的 **itype_t** 中
3. **hcl.c** 文件中
4. ssim-simple 文件中的 **sim_step** 函数中有两处

7. PART C

添加指令 lw

你需要修改的地方有五处:

1. isa.c 文件中的 **instruction_set** 中。
2. isa.h 文件中的 **itype_t** 中
3. **hcl.c** 文件中
4. ssim-simple 文件中的 **update_state** 中
5. ssim-simple 文件中的 **sim_step** 函数中

8. 注意事项

1. **需要仔细阅读的内容**包括:

isa.c 文件中的 instruction_set:

获取指令的名字、icode、ifun，可以知道该指令的存在。

isa.h 文件中的 itype_t:

将指令的 icode 改名，便于使用。

hcl.c 文件全部:

用？：模拟硬件的控制，决定 sim_step 函数的运行。

ssim-simple 文件中的 update_state 函数:

更新状态

ssim-simple 文件中的 sim_step 函数:

读取每一条指令时进行的操作。

2. **需要修改的位置**均用

////////////////////////////////////

//PART A:

////////////////////////////////////

括了起来（PART B，PART C 同理)。

可以修改其他位置，只要能实现目标。

3. 其**顺序实现**参考 y86 的顺序实现（CSAPP3e P264 开始)，有许多 y86 的指令实现方式，可以给 risc-v 的指令实现提供一点参考。

4. ifun1 即图中的 funct3，ifun2 即图中 funct7，valc 中存的即 imm

立即数。

5. imm 立即数获取之后再向左平移再向又平移是因为立即数是**符号扩展**的，需要将获取的 32 位的最高位变为符号位。
6. 注意 Part B 部分在获取 imm 立即数时会有一点不同（位移的立即数版本不需要有符号）
7. 注释有很多，不懂的可以在群里提问。
8. 例如：**以 I_R 为例子**。

I_R 是给 icode = 0x33（指令格式为 R）所取的名字。在 isa.h 的 itype_t 中添加。

I_R 有十个功能指令，将其全部填入 isa.c 的 instruction_set 中。

观察一下 I_R，其有 ifun1，ifun2，rs1，rs2，rd。该指令有效，因此将(icode)==(I_R)添加到该有的函数中（参考 hcl.c）。没有 imm，所以不需要 valc。

有寄存器 rs1，rs2，所以需要获取其中的值（srcA，srcB），并将需要计算的值 vala，valb 放入 aluA 和 aluB 中。将计算得到的值 vale 写入寄存器 rd 中（dstE）。

没有需要写入内存中（dstM）。没有需要内存读写（read，write），也不需要内存地址和写入的数据（addr，data）。

新的 pc 就是 valp（pc）（Valp = pc+4）。

9. 附加题

如果对于 risc-v 指令比较熟悉的同学对于上述实验做起来太快，

可以更进一步。

在将顺序模拟器改造成流水线模式之前，我们需要进行简单的调整以适合流水线模式。

将更新 PC 阶段放在一个周期开始时，而不是结束时。具体的结构参照 SEQ+（P288）。

这部分内容不会出题，请有余力和兴趣的同学进行尝试。（也不难的）

10. 附录

1. testparta.yo 正确结果

```
gyc@ubuntu:~/Desktop/risc-v$ gcc -Wall -O2 -o ssim hcl.c ssim-simple.c isa.c
gyc@ubuntu:~/Desktop/risc-v$ ./ssim -t testparta.yo
Risc-v Processor: seq
68 bytes of code read
IF: Fetched lui at 0x0.  rs1=----, rs2=----, rd=x6, Imm = 0x2000
IF: Fetched lui at 0x4.  rs1=----, rs2=----, rd=x7, Imm = 0x1000
IF: Fetched slt at 0x8.  rs1=x7, rs2=x6, rd=a2, Imm = 0x0
IF: Fetched add at 0xc.  rs1=a2, rs2=a2, rd=a3, Imm = 0x0
IF: Fetched bne at 0x10.  rs1=a2, rs2=a3, rd=----, Imm = 0xc
IF: Fetched sll at 0x1c.  rs1=a2, rs2=a3, rd=a4, Imm = 0x0
IF: Fetched blt at 0x20.  rs1=a3, rs2=a4, rd=----, Imm = 0xc
IF: Fetched sub at 0x2c.  rs1=a4, rs2=a2, rd=a5, Imm = 0x0
IF: Fetched bge at 0x30.  rs1=a4, rs2=a5, rd=----, Imm = 0xc
IF: Fetched add at 0x3c.  rs1=a4, rs2=a5, rd=a6, Imm = 0x0
IF: Fetched halt at 0x40.  rs1=----, rs2=----, rd=----, Imm = 0x0
11 instructions executed
Status = HLT
Changed Register State:
x6:     0x00000000      0x00002000
x7:     0x00000000      0x00001000
a2:     0x00000000      0x00000001
a3:     0x00000000      0x00000002
a4:     0x00000000      0x00000004
a5:     0x00000000      0x00000003
a6:     0x00000000      0x00000007
Changed Memory State:
```

2. testpartb.yo 正确结果

```
gyc@ubuntu:~/Desktop/risc-v$ gcc -Wall -O2 -o ssim hcl.c ssim-simple.c isa.c
gyc@ubuntu:~/Desktop/risc-v$ ./ssim -t testpartb.yo
Risc-v Processor: seq
36 bytes of code read
IF: Fetched addi at 0x0.  rs1=x6, rs2=----, rd=x6, Imm = 0x3
IF: Fetched slti at 0x4.  rs1=x6, rs2=----, rd=x7, Imm = 0xffffffff
IF: Fetched sltiu at 0x8.  rs1=x6, rs2=----, rd=a0, Imm = 0xffffffff
IF: Fetched xori at 0xc.  rs1=x6, rs2=----, rd=a1, Imm = 0x4
IF: Fetched ori at 0x10.  rs1=x6, rs2=----, rd=a2, Imm = 0x9
IF: Fetched andi at 0x14.  rs1=x6, rs2=----, rd=a3, Imm = 0x5
IF: Fetched slli at 0x18.  rs1=x6, rs2=----, rd=a4, Imm = 0x1e
IF: Fetched srli at 0x1c.  rs1=a4, rs2=----, rd=a5, Imm = 0x5
IF: Fetched srai at 0x20.  rs1=a4, rs2=----, rd=a6, Imm = 0x5
IF: Fetched halt at 0x24.  rs1=----, rs2=----, rd=----, Imm = 0x0
10 instructions executed
Status = HLT
Changed Register State:
x6:     0x00000000      0x00000003
a0:     0x00000000      0x00000001
a1:     0x00000000      0x00000007
a2:     0x00000000      0x0000000b
a3:     0x00000000      0x00000001
a4:     0x00000000      0xc0000000
a5:     0x00000000      0x06000000
a6:     0x00000000      0xfe000000
Changed Memory State:
```

3. testpartc.yo 正确结果

```
gyc@ubuntu:~/Desktop/risc-v$ gcc -Wall -O2 -o ssim hcl.c ssim-simple.c isa.c
gyc@ubuntu:~/Desktop/risc-v$ ./ssim -t testpartc.yo
Risc-v Processor: seq
44 bytes of code read
IF: Fetched lui at 0x0.  rs1=----, rs2=----, rd=x6, Imm = 0xf000
IF: Fetched lui at 0x4.  rs1=----, rs2=----, rd=x7, Imm = 0x1000
IF: Fetched slt at 0x8.  rs1=x7, rs2=x6, rd=a2, Imm = 0x0
IF: Fetched add at 0xc.  rs1=a2, rs2=a2, rd=a3, Imm = 0x0
IF: Fetched sll at 0x10.  rs1=a2, rs2=a3, rd=a4, Imm = 0x0
IF: Fetched sll at 0x14.  rs1=a4, rs2=a2, rd=a4, Imm = 0x0
IF: Fetched srl at 0x18.  rs1=x6, rs2=a4, rd=a5, Imm = 0x0
IF: Fetched sw at 0x1c.  rs1=a5, rs2=x6, rd=----, Imm = 0x0
Wrote 0xf000 to address 0xf0
IF: Fetched sw at 0x20.  rs1=a5, rs2=x7, rd=----, Imm = 0xfffffffc
Wrote 0x1000 to address 0xec
IF: Fetched lw at 0x24.  rs1=a5, rs2=----, rd=a6, Imm = 0xfffffffc
IF: Fetched halt at 0x28.  rs1=----, rs2=----, rd=----, Imm = 0x0
11 instructions executed
Status = HLT
Changed Register State:
x6:     0x00000000      0x0000f000
x7:     0x00000000      0x00001000
a2:     0x00000000      0x00000001
a3:     0x00000000      0x00000002
a4:     0x00000000      0x00000008
a5:     0x00000000      0x000000f0
a6:     0x00000000      0x00001000
Changed Memory State:
0x00ec: 0x00000000      0x00001000
0x00f0: 0x00000000      0x0000f000
```