RISC-V 指令集仿真器

在本次编程训练中，你将会构建一个 RISC-V 指令集仿真器应用，使用 RISC-V 工具链编译得到二进制文件，通过仿真器模拟 RISC-V 程序的运行。你的任务是根据 RV32I 指令集，完成仿真器编码、译码与执行过程中的部分内容，从而得到一个功能完整的指令集仿真器。

1. 文件说明

除了本文档外，压缩包还包含以下六个文件。

1）*instforms.cpp*：根据头文件中不同指令类型的结构，实现了指令对应结构体中的编码函数。

2）*decode.cpp*：实现仿真器的译码函数，根据指令操作码和功能码得到译码结果。

3）*Hart.cpp*：实现硬件线程的模拟，需要补充部分指令的执行函数。

4）*Inst*：文件夹中包含了仿真器中与编码译码过程密切相关的文件，但**无需修改**。

5）*whisper.zip*：指令集仿真器的完整程序，将你补充完成的三个文件替换后可以编译得到仿真器应用。

6）*test*：示例 RISC-V 测试程序，完成仿真器设计后用于验证仿真器正确性。

2. instforms: 指令编码

仿真器在 instforms.hpp 文件中针对不同指令类型构建了用于编码和译码的结构体，你需要在 instforms.cpp 中实现对应的编码函数。编码函数的输入参数是寄存器操作数（和立即数操作数），根据编码规则确定最终的编码。实验中用到的 RV32I 指令集在文档最后给出。下面举一个 add 指令的例子进行说明。

add 是 R 类型指令，R 类型对应的结构体为

```
struct
{
   unsigned opcode   : 7;
   unsigned rd       : 5;
   unsigned funct3   : 3;
   unsigned rs1      : 5;
   unsigned rs2      : 5;
   unsigned funct7   : 7;
```

```
} bits;
```

与编码格式类似，结构体中将 32 位指令分为 6 个部分，包括三个寄存器操作数，操作码和功能码。再看 add 的编码函数

```
bool RFormInst::encodeAdd(unsigned rdv, unsigned rs1v, unsigned rs2v)
{
   if (rdv > 31 or rs1v > 31 or rs2v > 31)
      return false;
   bits.opcode = 0x33;
   bits.rd = rdv & 0x1f;
   bits.funct3 = 0;
   bits.rs1 = rs1v & 0x1f;
   bits.rs2 = rs2v & 0x1f;
   bits.funct7 = 0;
   return true;
}
```

首先判断输入参数是否超出了理论范围，然后为每个结构体内容赋相应的值，操作码和功能码由指令确定，操作数由输入参数确定。根据 add 指令与 addi 指令的示例，**你需要完成六条指令的编码函数：I 类型的 lb 与 slli 指令，B 类型的 beq 指令，S 类型的 sb 指令，U 类型的 lui 指令，J 类型的 jal 指令。除了上述六个函数之外，请不要改变其余部分代码。**

在 I 类型指令中，需要注意 lb 与 slli 对应了不同的结构体，slli 立即数偏移只有 5 比特；在 B 类型与 J 类型指令中，结构体最高位为 int，赋值时需要注意负数需要赋-1 值。

3. decode：指令译码

本部分实现了仿真器的译码功能，实验主要关注从 decode.cpp 文件 1368 行开始的 decode 函数，输入一条指令 inst，需要给出其对于的操作数 op 和指令条目 InstEntry。InstEntry 在 InstEntry.hpp 中定义，指令集中的每一条指令都对应了一个条目。操作数 op 定义在 DecodeInst.hpp 文件中，对于一般的指令，如 add x2, x1, x0，op0 为 x2，op1 为 x1，op2 为 x0，特殊的指令主要是 load 和 store 指令。对于 load 指令，将“load rd, offset(rs1)”映射为“load rd, rs1, offset”，因此 op0 为 rd，op2 为 offset；对于 store 指令，将“store rs2, offset(rs1)”映射为“store rs2, rs1, offset”，因此 op0 为 rs2，op2 为 offset。

下面来看 decode 具体操作。可以将 RISC-V 指令集根据操作码 opcode 的高五位（低两位恒为 11）分为 32 个操作码空间，每个操作码空间对应了同一种类型的指令，在每个操作码空间确定指令对应操作数，并根据功能码进行译码，确定对应的指令条目。**代码中给出了 opcode[6:2]=0 时的译码过程，你需要完成 l5，l8，l13，l24，l25，l27 和 l12 的部分内容，确定操作数 op 值，并返回对应指令条目。**用到的指令集已在文档最后给出。

4. Hart: 指令执行

Hart 是 Hardware Thread 的缩写，本实验主要关注从 Hart.cpp 文件 8175 行开始的内容，即指令的执行过程。下面同样以 add 指令为例：

```
template <typename URV> inline void
Hart<URV>::execAdd(const DecodedInst* di)
{
  URV v = intRegs_.read(di->op1()) + intRegs_.read(di->op2());
  intRegs_.write(di->op0(), v);
}
```

从指令的译码过程中我们知道，op0 对应目的寄存器 rd，op1 和 op2 对应源寄存器 rs1 和 rs2。执行过程中先从 op1 和 op2 对应寄存器中读出相应内容并相加，然后将结果写回 op0 对应寄存器中。

除了 add 指令外，还给出了 addi，beq，blt 指令的示例，根据这些示例，你需要完成 and、andi、slt、sltu、bne、bltu、bge、bgeu、jal、jalr、auipc 指令的执行过程。需要注意的是无符号和有符号数的比较，以及涉及 PC 运算时 pc_和 currPc_变量的不同（提示：pc 值在指令执行前更新）。

5. 测试

完成上述工作后，你可以编译整个仿真器文件，得到应用程序后执行测试程序验证设计的正确性。

为了编译整个仿真器文件，需要 7.2 版本以上 g++编译器，以及 1.67 版本以上的 boost 库，boost 库可以在 boost.org 下载。boost 库下载完成后，运行 bootstrap.sh 脚本会得到 b2 应用程序，再执行 b2（Windows 下为 b2.exe）即可。

满足编译条件后，解压 whisper 压缩包，将修改好的三个文件替换到 whisper 目录中，使用该目录下的 GNUmakefile 进行编译。如果是 Linux 系统，只需在终端中切换到 whisper 目录下，执行

- make BOOST_DIR= $your_path

命令即可（$your_path 填入你安装 boost 库的路径）。执行完毕后会在 whisper 目录下生成 build 子目录，当中包含 whisper 应用程序。

如果是 windows 系统，需要下载 Cygwin 来模拟 UNIX 环境，首先在 https://www.cygwin.com 官网下载软件，然后安装 gcc、make 等必要工具，安装完成后启动 Cygwin Terminal，在该终端下使用与 ubuntu 同样的命令编译 whisper。需要注意的是通过 Cygwin 编译得到的应用程序同样需要在该环境下运行。

得到仿真器的应用程序后，还需要相应的测试文件。你可以编写一个简单的测试程序，然后使用 RISC-V 交叉编译器编译得到机器码。交叉编译器的源码可以在 https://github.com/riscv-collab/riscv-gnu-toolchain 下载。我们提供了一个已经编译好的简单的测试程序 test，程序中主要包含下面的 sim 函数：

```
sim:
   addi  a0,  x0,  1
   slli   t0,   a0,  2
   add   t1,   a0,  t0
   xor   a1,  a0,  t1
   bne   a1,  t0,   error
   jalr   x0,  0(ra)
error:
   addi  a0,  x0,  0
   jalr   x0,  0(ra)
```

当 sim 函数返回 1 时测试通过，返回 0 时测试失败。可以直接在包含 whisper 应用程序和测试程序的目录中执行

- ./whisper test

如果终端打印出“Test Pass”，说明仿真器正确执行了 RISC-V 程序中的指令。

如果你想要逐条指令执行，可以进入 whisper 的交互模式，即输入

- ./whisper --interactive test

在交互模式下，你可以用 peek 命令查看仿真器的寄存器、内存等硬件资源，也可以用 disass 查看反汇编得到的 RISC-V 汇编指令。通过 step 单步执行可以看到每条指令对硬件资源做出的更改。这些命令的格式可以使用 help 查看。

6. 提交要求

只需提交修改后的 instforms.cpp，decode.cpp，Hart.cpp 文件（如有自己设计的测试程序也一起提交），以及一份报告，报告中应说明代码的实现思路以及一些测试结果。**将所有文件打包后以学号加姓名命名，上传到学在浙大，并在 2021.11.25 课堂上上交一份纸质报告。**学在浙大的截止时间是 2021.11.25 日 23:59。

RV32I基础指令集

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
fm | pred | succ		rs1	000	rd	0001111	FENCE
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK