RISC-V领域加速指令设计与仿真

浙江大学信息与电子工程学院

刘岸林 anlinliu@zju.edu.cn

在本次编程训练中，将从软硬件协同设计的思想出发，利用拓展指令实现特定程序的优化。你的任务是根据拓展指令集，完成仿真器编码、译码与执行过程中的拓展内容，从而得到一个功能完整的指令集仿真器。然后，在此基础上，修改汇编代码添加拓展指令，利用仿真器进行仿真，比较拓展指令减少的指令条目，优化性能。

一、 文件说明

除了本文档外，压缩包还包含以下 12 个文件。

1）*instforms.cpp & instform.hpp*：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 & Hart.hpp*：实现硬件线程的模拟，需要补充部分指令的执行函数；

4）*InstEntry.cpp*：定义指令集中每一条指令对应条目；

5）*InstId.hpp*：定义指令集中每一条指令对应 id；

6）*whisper.zip*：指令集仿真器的完整程序，将你补充完成的 7 个文件替换后可以编译得到仿真器应用；

7）*boost_1_78_0.tar.gz*：编译仿真器文件所需的 boost 库；

8）*sha256*：文件夹中包含了 sha256 算法文件，需修改其中的 sha256opt.s；

9）*test1*：示例 RISC-V 测试程序，完成仿真器设计后用于验证原指令仿真器正确性。

10）*test2*：示例 RISC-V 测试程序，完成仿真器设计后用于验证拓展指令仿真器正确性。

二、 任务 1：仿真器拓展

本节将从拓展指令 cube 出发，引导你实现仿真器的指令拓展。你的任务是完成以下 R-Type 指令的拓展：

0000010	rs2	rs1	000	rd	0110011	cube
0000010	rs2	rs1	001	rd	0110011	rotleft
0000010	rs2	rs1	010	rd	0110011	rotrigt
0000010	rs2	rs1	011	rd	0110011	reverse
0000010	rs2	rs1	100	rd	0110011	notand

cube rd, rs1, rs2: R[rd] = R[rs1]^3

rotleft rd, rs1, rs2: R[rd] = ((R[rs1] << R[rs2]) | (R[rs1] >> (32- R[rs2])))

rotright rd, rs1, rs2: R[rd] = ((R[rs1] >> R[rs2]) | (R[rs1] << (32- R[rs2])))

reverse rd, rs1, rs2: R[rd] = (R[rs1] >> (24 - R[rs2] * 8)) & 0x000000ff;

notand rd, rs1, rs2: R[rd] = ~(R[rs1]) & R[rs2]

1. 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。实验中用到的拓展指令集已在此前文档给出。

R-Type 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。拓展指令 cube 的编码函数为

```
bool
RFormInst::encodeCube(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 = 2;
    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 cube 指令示例，你需要完成其他四条拓展指令的编码函数，除此之外，请不要改变其余部分代码。**
注：instforms.hpp 文件中 84 行起已添加拓展编码函数的声明。

2. 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，cube 指令声明于 InstEntry.cpp 中 3891 行。指令同时需要在 InstId.hpp 中声明，cube 指令声明于 634 行。**你需要参照 cube 及其他指令格式，在 InstEntry.cpp 和 InstId.hpp 中添加其他四条拓展指令内容，注意及时修改 InstId.hpp 中 maxId 的值。**

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。cube 指令的译码已于 decode.cpp 文件 1839 行给出，**你需要参照 cube 及其他指令格式，在 decode.cpp 中添加其他四条拓展指令译码内容。**

3. Hart: 指令执行

Hart 是 Hardware Thread 的缩写，本实验主要关注从 Hart.cpp 文件 979 行开始的内容，即指令的执行过程。下面同样以拓展指令 cube 指令为例：

```
template <typename URV>
inline
void
Hart<URV>::execCube(const DecodedInst* di)
{
  URV v = intRegs_.read(di->op1()) * intRegs_.read(di->op1()) *
          intRegs_.read(di->op1());
  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。执行过程中从 op1 对应寄存器中读出相应内容并立方计算，然后将结果写回 op0 对应寄存器中。

根据这些示例，你需要完成其他四条拓展指令的执行过程，**同时注意左移操作时寄存器中数据的比特位长度应设为 32 位，可使用强制类型转换为 int**。注：Hart.hpp 文件中 1848 行起已添加拓展执行函数的声明。

在完成执行函数后，你需要参照 cube 指令在 Hart.cpp 文件 5442 行开始的 execute()函数中添加拓展指令的 label 以及相应的跳转执行，cube 指令的 label 已于 6007 行给出，执行跳转已于 6171 行给出。

即，你需要在 Hart.cpp 完成其他四条拓展指令的执行过程，添加拓展指令的 label 以及相应的跳转执行。

4. 测试

在完成上述工作后，即修改 instforms.cpp，InstEntry.cpp，InstId.hpp，decode.cpp，Hart.cpp，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。

我们推荐你使用 Linux 系统完成整个实验，如果是 windows 系统，可以使用 windows 的子系统 WSL 模拟 UNIX 环境，然后安装 gcc、make 等必要工具，安装完成后启动 WSL，在该终端下使用与 ubuntu 同样的命令编译模拟器。

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 库已在压缩包给出。解压缩后，运行 bootstrap.sh 脚本会得到 b2 应用程序，再执行 b2 即可。Linux 下执行以下命令即可

- ./bootstrap.sh

- ./b2 install

满足编译条件后，解压 whisper 压缩包，将修改好的 instforms.cpp，InstEntry.cpp，InstId.hpp，decode.cpp，Hart.cpp，instform.hpp，Hart.hpp (instform.hpp，Hart.hpp 已添加函数声明，无需你进行更改) 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。

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测试文件。**交叉编译器的源码可以在 https://github.com/riscv-collab/riscv-gnu-toolchain 下载。**

我们提供了两个已经编译好的简单的测试程序 test1，test2：test1 用于测试原有 RISC-V 指令的执行情况，test2 用于测试拓展指令的执行情况。

可以直接在包含 whisper 应用程序和测试程序的目录中执行

- ./whisper test2

如果对于每条拓展指令终端均打印出对应的“test pass”，说明仿真器正确执行了 RISC-V 程序中的拓展指令。

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 test2

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。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。

三、 任务 2：优化哈希加密

SHA-2，名称来自于安全散列算法 2（英语：Secure Hash Algorithm 2）的缩写，一种密码散列函数算法标准，由美国国家安全局研发，属于 SHA 算法之一。实验中将用到 SHA256 算法，对于任意长度的消息，SHA256 都会产生一个 256bits 长的哈希值。

我们在 SHA256 文件夹中提供了实现 SHA256 算法的 c 文件 sha256.c，汇编文件 sha256.s，可执行文件 sha256。

你的任务是利用拓展指令 rotleft、rotright、reverse、notand，对照 sha256.c 修改 sha256.s 汇编文件，将原来汇编文件中的部分指令等价转换为拓展指令，对 SHA256 算法实现指令级的优化，并将修改后的结果保存到 sha256opt.s 中。实验

重点关注 sha256 算法中的 sha256_transform()，sha256_final()函数。

汇编文件中 R-Type 扩展指令集的格式如下：

- .insn r opcode, func3, func7, rd, rs1, rs2

下以 cube 指令为例，说明如何进行指令等价转换：

原汇编：

```
mul       a4,  a5,  a5
mul       a4,  a4,  a5
```

优化汇编：

```
.insn r 0x33, 0, 2, a4, a5, x0     # a4=a5*a5*a5
```

汇编文件修改完成后，需要使用 RISC-V 交叉编译器得到可执行文件 sha256opt，然后使用仿真器与我们提供给你的可执行文件 sha256 进行对比，仿真器会输出程序执行过程中的指令条目数“Retired XXX instructions in X s”。

在对比过程中，你需要分别输入”Zhejiang University”、”COD”、你的学号，计算其哈希值，在验证你汇编的正确性同时比较优化效果。

最后，你需要按照此前的设计思路，在已提供的五条拓展指令以外，至少自主设计并完成一条拓展指令，同时对仿真器和哈希加密的汇编文件进行修改，从而实现程序的进一步优化。注：自定拓展指令添加函数时需要修改对应的头文件.hpp。你的优化结果将会在一定程度上影响你的最终得分。

四、 提交

只需提交修改后的 instforms.cpp，InstEntry.cpp，InstId.hpp，decode.cpp，Hart.cpp，instform.hpp，Hart.hpp，sha256opt.s，sha256opt 文件，以及一份报告。报告中应说明模拟器的实现思路及测试结果、汇编文件的修改思路、自定拓展指令的设计思路及实现，程序最终的优化结果。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 2022.12.1 课堂上上交一份纸质报告。学在浙大的截止时间是 2022.12.1 日 23:59。

五、 参考文献

[1] 席宇浩, RISC-V 指令集仿真器, 浙江大学信息与电子工程学院, 计算机组成与设计(2021 年)