前面搞好了下面就是找 riscv 测试用例，找了很久，网上只有一些解释类的文章，找不到可以参考的测试文件。
于是决定自己写测试文件。
第一步写了个几行的测试文件，但是之后发觉这么写不知道要写到猴年马月，于是决定写个 java 程序将编译程序转换为指令码。
写了一点觉得这么写下去也不是个办法，要写出来相当地麻烦，要不还是回去手写？
想办法找个完整的测试文件可能会更好。
这个时候，想到了以前一门课大作业留下的一个测试文件，找一找居然被找到了，拿来测试用。
这个测试文件总共包含九个文件，分别是 1testAll.data, 1testAll.instr, 1testAll.txt, 2testAll.data, 2testAll.instr, 2testAll.txt, 3testAll.data, 3testAll.instr, 3testAll.txt。接下来就简单，把这些文件一个个的执行下去就可以了。
可是，执行时遇到错误“[USF-XSim-62] 'elaborate' step failed with error(s) while executing 'E:/2020/project_7/project_7.sim/sim_1/behav/elaborate.bat' script. Please check that the file has the correct 'read/write/execute' permissions and the Tcl console output for any other possible errors or warnings.”这是以往任何一次仿真都没有遇到的错误，这个错误在网上也没搜到怎么解决，最开始我以为是某条指令执行失败，决定找出对应指令，一次次仿真，发现减少指令数目到一定程度后我的 windows 报错一个文件占用内存太多，我发觉可能是我一次写入太多指令导致。
接下来我打算把 3 个 test 文件分解，之前的查找发现指令最多有六百条，下面要做的就是把 test 文件分解成每个 600 条的小文件。这里要注意某些跳到绝对地址的跳转指令。说的就是 jalr 指令。
还好，这些测试文件里面 jalr 指令都是和 auipc 联合出现的，所以可以想怎么分就怎么分，注意不要拆开测试里面的小模块。
然后首先将 1testAll.instr 拆分为了 5 个小模块，每个模块不超过 600 条指令。分别运行，发现都可以很好地运行。
同样的方法，我们把 2testAll.Instr 切分成 6 个小模块。3testAll.instr 切成 6 个小模块放进指令寄存器里面运行。
都可以运行，因为项目已经是开源的了，所以放入指令是一定可以运行的。
接下来试试改写，截图等下放。
先试试 riscv_alu 模块的改写，此处参照了我曾经做过的大作业的 alu 模块。
改写遇到的第一个挑战是 logic 和 wire/reg 的转化，一般来说 input 直接设置为 wire 即可，但是 output 要根据它是用 assign 还是 always 赋值的决定是 wire 还是 reg。
Generate 是 verilog 里面有的，不去管它。
然后就是 always_comb，它约等于 always@（*）
$signed()函数，verilog 里面就有，取有符号数，unsigned 是取无符号数。
V 语言没有++算法，sv 有。
刚才试了一下将 sv+v 混合项目运行，发现运行的时候电脑死机了一下，说我访问了计算机的 0 号地址，现在也不敢在做什么改动，还是试试别的路吧。
目前看来做不到依次改写成一个 verilog 项目，现在尝试别的改写方法。
尝试将这个项目简化成一个只支持 rv32i 的项目。
这几天开始着写论文的草稿了。
改写目标是仅仅实现 rv32i 指令体系，所以第一次改写目标是把 FPU 即浮点操作型给去掉。
仿真的时候发现了个小毛病，这个内核会循环地执行仿真代码，具体情况见下面：

指令地址执行到 0x000610 时候会跳到 0x001b10 地址然后重新跳回 0 地址重新执行所以 instr 文件。

对应的代码是 17500193，代码为 li　gp,373，把 3 号寄存器置为 0x175，

这条指令很好地完成了，那为什么会跳到 0x001b10 呢？

经过计算，下一条指令 j　12af8 被执行的话会刚好跳到 0x001b10

分析这一测试文件的汇编语言：

```
115d4:    00002097            auipc    ra,0x2
115d8:    58008093            addi ra,ra,1408 # 13b54 <sb_tdat>
115dc:    fffff137            lui    sp,0xfffff
115e0:    fa010113            addi sp,sp,-96 # ffffefa0 <__global_pointer$+0xfffeab58>
115e4:    00208123            sb    sp,2(ra)
115e8:    00209f03            lh    t5,2(ra)
115ec:    fffffeb7        lui    t4,0xfffff
115f0:    fa0e8e93            addi t4,t4,-96 # ffffefa0 <__global_pointer$+0xfffeab58>
115f4:    17500193            li    gp,373
115f8:    01df0463            beq  t5,t4,11600 <test_374>
115fc:4fc0106f            j    12af8 <fail>
```

Auipc 将 0x2000 加到 pc 上，存入 1 号寄存器，再将 2 号寄存器设置为 0xfffff000，减去 96，得到 0xffffefa0，接着执行 sb　sp,2(ra)，这时候就找到了问题所在，sb 是内存读取指令，而我没有设置内存。

需要改写 dp_ram.sv

追踪 lsu 模块里面的数据写，找到了对应的数据线，在 dp_ram 里面的 addr_b_int 和 data_addr_dec，找到了数据内存的起始地址为 0x0010 0000，下面把 data 数据也写进去。

不行，写了会卡死

我将 tp_top 里面的 INSTR_RDATA_WIDTH 从 128 改为了 32 位，将 RAM_ADDR_WIDTH 从 22 改为了 14 位。

重新看一下哪里出了问题

lui　sp,0xfffff 之后 2 号寄存器 sp 的值为 0xffff f000，addi　sp,sp,-96 后为 ffffefa0，之后 sb 把 a0 存到地址 11122，紧接着就是 lh　t5,2(ra)，将 11122 取 2 个字节后填充高位写入 t5，就是这步出了错。

不仅仅是这步错了，所有内存读写指令都有错：

换了个测试代码：

```
101a4:    0f700193            li    gp,247
101a8:    00000213            li    tp,0
101ac:    00004097            auipc    ra,0x4
101b0:    9a508093            addi ra,ra,-1627 # 13b51 <tdat2>
101b4:    00108f03            lb    t5,1(ra)
101b8:    000f0313            mv    t1,t5
```

```
101bc:    ff000e93          li    t4,-16
101c0:    01d30463          beq   t1,t4,101c8 <test_247+0x24>
101c4:    1350206f          j     12af8 <fail>
101c8:    00120213          addi  tp,tp,1 # 1 <_start-0x1007f>
101cc:    00200293          li    t0,2
101d0:    fc521ee3          bne   tp,t0,101ac <test_247+0x8>
```

最开始四条语句执行后：

Gp = x3 = ， tp = x4 =， ra = x1 =

（这个过程中发现不能开头有气泡，一旦有气泡就会跳到 boot_addr）

Test1_1~5 全部通过，主要有问题的是有内存访问指令的 test2 和 test3。

1~5 细看也有问题，不能全部通过。

Tb_top 的参数重新改回 parameter INSTR_RDATA_WIDTH = 128,而 boot_addr 改为 0（改为 0 后就好多了，test1 全员通过）

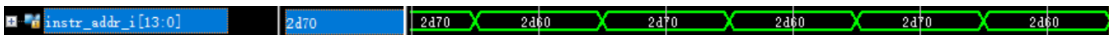
Test2 依旧有错，这个时候我想到了因为地址被我缩小了，那么这时候可不可以不再拆分 test 文件了呢？

成功，所有 test 文件完美运行。

总结一下，其实只要调整 boot_addr 与 RAM_ADDR_WIDTH 就可以了。

之前说到的改写模块，将除了 RV32i 以外的功能的删去，之前已经删了个 FPU 了。

这个时候注意仿真图像中出现的一个问题：



最后的取指令地址会在 2d60 和 2d70 之间横跳，这是为什么呢？

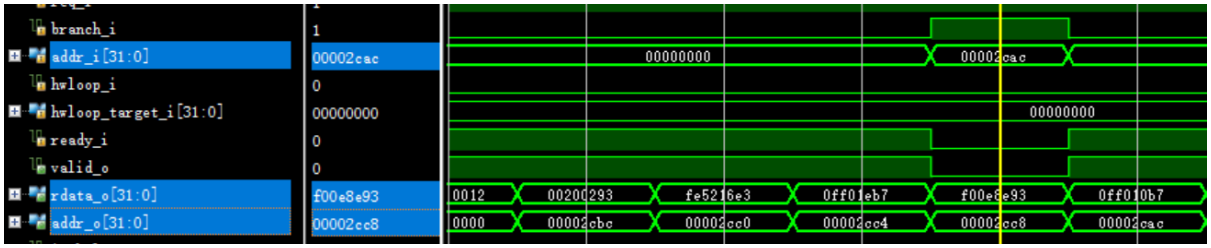
首先推断这个地址就是最后的一条指令的地址，我们找到存储器文件，发现里面的 2d60 转为十进制就是 11616，对应的指令数为 11616/4=2904，看 data 文件：

```
2900  0ff08093
2901  70f0c013
2902  00000e93
2903  34600193
2904  01d01463
2905  00301463
2906  00000a6f
2907  00100193
2908  00000a6f
2909  c0001073
2910  00000000
2911  00000000
2912  00000000
2913  00000000
2914  00000000
2915  00000000
2916  00000000
```

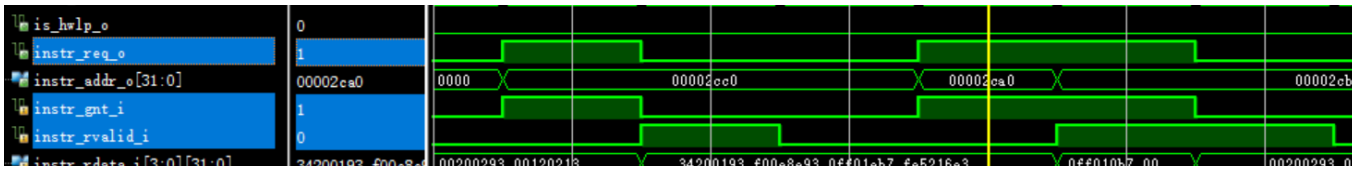
理论上来说应当还有及条指令没有执行呢？这个时候看源代码最后一个测试模块：

之后的 instr 就一直是 00000af6 了。

然后看看这些指令，按照之前说的，最后读取的地址对应的指令是 01d01463，照理说后面应当还有 5 条指令没有执行,那么为什么在波形图里面只有 c0001073 一条指令缺失呢？（为什么一条指令缺失等下再说）
因为之前提到过，RI5CY 最开始是为了使用在多核体系结构 PULP 上面的，所以为了使各个核之间能更好的协同运作，它增加了一个预取指令模块，这个模块会在指令还没执行到的时候将指令存储器里面的指令预取到 cpu 内的一个存储器里面，然后执行指令就直接从这个存储器里面取指令。这个核的 INSTR_RDATA_WIDTH = 128 也就是一次取 128 位数据，即 4 条 32 位指令。
再看最后一个 c0001073 为什么没有执行：
因为这个指令不合法，没有在 riscv32 指令集里面找到对应的指令。
分解看看 1100 0000 0000 0000 0001 0000 0111 0011 对应一条 CSRRW 指令。
预取模块中的 rdata_o 连到 instr 线上，两者之间的同步相差一个周期。Addr_o 是对应的 rdata_o 的地址。
Branch_i 对应的是一位跳转使能，当置为 1 的时候，进入预取指令模块找到 addr_i 对应的地址对应的指令，将其放到下一个出模块的指令里面。
那么如果跳转的指令这个时候已经不再预取指令模块里了呢？之前说过这个模块一次最多可以预取 4 条指令。



当这条指令的地址在预取模块里面找不到的时候，它会向 mem 请求对应的指令：



把 instr_req_o 置为 1 来请求指令，找到指令后 instr_rvalid_i 会返回个 1 表示找到了。
第一处删减，将 riscv_prefetch_buffer 中的有关 hwloop 的第一块给删了，所有相关变量置为 0.同时把置为 HWLP_NONE，意思我们没有 HWLP 命令。
运行成功。
Riscv_prefetch_buffer 不能删了，因为剩下的东西都有用。
Riscv_prefetch_buffer 里面还有一个 riscv_fetch_fifo,这个模块的内容是实现先进先出。这里面最重要的部分是：（DEPTH 为 4）

```
logic [0:DEPTH-1] [31:0]  addr_n,     addr_int,     addr_Q;
logic [0:DEPTH-1] [31:0]  rdata_n,    rdata_int,    rdata_Q;
logic [0:DEPTH-1]         valid_n,    valid_int,    valid_Q;
```

储存了对应的指令的内容，物理地址，。。。
再看 riscv_hwloop_controller，这个模块应当是可以整个删除的，试试看。
3 个 output 始终是 0，删掉。

第二处删减，将 riscv_hwloop_controller 模块删掉，3 个 output 对应的线恒置为 0.为 riscv_if_stage 中 301 行注释部分和下三行 assign。
修改后运行无障碍。
If_stage_i 中这个时候还有很多小的选项在本课题是无意义的，在这里暂时放下，先去改之后的代码。