Lab3 实验报告

PART1 2CYC_HARVARD

PART1 实际上需要完成的是将一个 2 周期的多周期处理器改成一个 3 周期的多周期处理器。因此只需要在原来的基础上添加一个状态即可。

首先需要新定义一个状态 Writeback。

```
typedef enum {Fetch, Execute, Writeback} State deriving (Bits, Eq);
```

图 1 Writeback 状态定义

其次需要添加 Execute 和 Writeback 两个状态之间的状态寄存器，使得第三阶段能够收到第二阶段的运行结果。

```
Reg#(ExecInst) execInst <- mkRegU;
```

图 2 新增状态寄存器

最后是将原本位于 Execute 阶段的程序计数器更新逻辑，寄存器堆写回逻辑独立出来到 Writeback 执行阶段中。过程中需要注意 state 寄存器的更新。

```
rule doExecute(cop.started && state == Execute);
  … …

  if(eInst.iType == Ld)
  begin
    … …
  end

  execInst <= eInst;
  // switch to writeback state
  state <= Writeback;
endrule
```

图 3 Execute 阶段规则

```
rule doWriteback(cop.started && state == Writeback);
  if (isValid(execInst.dst) && validValue(execInst.dst).regType == Normal
    rf.wr(validRegValue(execInst.dst), execInst.data);

  pc <= execInst.brTaken ? execInst.addr : pc + 4;

  cop.wr(execInst.dst, execInst.data);

  // switch back to fetch
  state <= Fetch;
endrule
```

图 4 Writeback 阶段规则

实验结果不便放入报告，见 l3p1.out。

PART2 PCMSG_EPOCH

PART2 需要所修改的二级流水线的原始代码能够编译，但是不能正确执行，因为其中缺少对 J，JR 和 BR 指令的处理。以至于直接编译并模拟会使得处理器忽略跳转指令，顺序执行所有内存中加载的代码。

在实验中需要注意的细节有两处。一个是分支预测器的设计，一个是 epoch 寄存器的作用。

关于分支预测器。其实 lib/AddrPred.bsv 中提供的分支预测器相当简陋，其作用只是缓存分支跳转指令发生跳转时的下一跳地址。对于 J 指令，其预测总是准确的（在 BTB 中有相关信息的情况下）；而对于 BR 指令，只有跳转发生时，才能准确预测。

分支跳转指令是否发生跳转需要在 Execute 阶段才能够确定。分支跳转指令在 Execute 阶段中时，需要判断该指令是否会发生分支跳转，分支预测器是否误判。如果发生跳转，便将本条指令的 PC，下一跳地址等信息发送给 Fetch 阶段，在下一时钟周期中由 Fetch 阶段对分支预测器的 BTB 进行更新。如果预测失败，意味着当前时钟周期中，Fetch 阶段正在获取的指令需要被清除，于是通过旁路向 Fetch 阶段转发预测失败信息和正确的下一跳地址，保证 PC 的正确更新。

关于 epoch 寄存器，其作用实际上是用来清除流水线中错误加载的指令。只有两个 epoch 寄存器中的值相同的时候，Execute 阶段才会执行。之前提到如果预测失败，Fetch 阶段正在获取的指令需要被清除，对应于流水线中具体的操作是将两个 epoch 寄存器反转。那么下一周期中，Execute 阶段用于比较的两个寄存器 fEpoch 和 eEpoch，一个来自上周期，一个来自本周期，值不相同，因此便跳过错误指令的执行。

经过以上分析，便能够进行相应代码的修改了。

```
// Send the branch resolution to fetch stage, irrespective of whether
// it's mispredicted or not
if (eInst.iType == Br || eInst.iType == J || eInst.iType ==Jr)
  execRedirect.enq(Redirect{pc: pc, nextPc: eInst.addr,
    brType: eInst.iType, taken: eInst.brTaken,
    mispredict: eInst.mispredict});
// On a branch mispredict, change the epoch, to throw away wrong
// path instructions
if (eInst.mispredict) eEpoch <= !eEpoch;
```

图 5 跳转分支指令支持

实验结果在 l3p2.out 和 l3p2o0.out 中，l3p2.out 中是开启分支跳转预测器的运行结果，l3p2o0.out 中是关闭分支跳转预测器的测试结果。

从中可以看到，在两级流水线结构中，使用分支预测能够带来部分的性能提升。如果进一步 trace 其执行过程可以发现，其中大部分性能提升得益于 J 指令的预测正确。