分支预测 Lab4 实验报告

PB16060130　　顾健鑫

一、实验目的

本次实验需要在提供的样板 CPU 中实现基于 BTB 和 BHT 的分支预测器。

二、实验环境和工具

本次实验使用的软件环境为 Vivado 2017.4 的版本。在创建工程项目时，直接导入提供好的模块框架文件并在完成代码的编写后，导入测试文件用于测试，并用 Vivado 的波形仿真结果作为结果检验依据。

三、分支预测的具体实现

1. BTB

本次实验中对 CPU 中的 NPC_Generator.v，BranchDecisionMaking.v，HazardUnit.v 做了修改以实现分支预测，首先介绍对于 BranchDecisionMaking.v 的修改。

a) 对于 BranchDecisionMaking 模块，将其的 BranchE 信号做了拓宽处理，变为了两位，分别表示四种情况，2'b00 表示预测分支，实际分支；2'b01 表示预测不分支（cache 命中失败使得预测不分支），实际分支；2'b10 表示预测不分支（命中但当前状态为预测不分支），实际分支；2'b11 表示预测分支，实际未分支。

这四种不同的 BranchE 信号分别对应四种情况，BranchE 信号回传给 NPC_Generator 模块用于状态的更新。

b) 对于 NPC_Generator 模块，添加了如下输入和输出

```
input wire [1:0] BranchFlagsF,
input wire [2:0] BranchIndexF,
input wire [1:0] BranchE,
input wire [2:0] BranchIndexE,
output reg [1:0] BranchFlags,
output reg [2:0] BranchIndex
```

`BranchFlags` 为两位 `E` 和 `F` 表示为哪一个流水段的信号，其 `0` 号位表示是否在 `cache` 命中，`1` 为命中，`0` 为缺失；在命中的情况下，其 `1` 号位表示预测结果是否为跳转，`1` 为预测分支，`0` 为预测不分支。

`BranchFlagsF` 与 `BranchE` 共同决定下一条指令的取指方式，具体的程序逻辑参见下图中的代码以及注释

```
always @(*)
begin
    if(JalrE)
        PC_In <= JalrTarget;
    else if(BranchE == 2'b01 || BranchE == 2'b10)//匹配或命中失败，但分支成功，则使用BranchTarget作为NPC
        PC_In <= BranchTarget;
    else if(BranchE == 2'b11)//命中成功，但分支失败，则使用BranchPC + 4作为NPC
        PC_In <= BranchPC[BranchIndexE] + 4;
    else if(JalD)
        PC_In <= JalTarget;
    else if(BranchFlagsF == 2'b11)
        PC_In <= BranchPredPC[BranchIndexF];
    else
        PC_In <= PCF+4;
end
```

当 BranchE 的信号不是 00 时，表示预测失败，要做相应的回复操作，01 和 10 对应的情

况都需要将 BranchDecisionMaking 模块传回的 BranchTarget 作为跳转地址；当为 11 时，要将在 Ex 段的跳转指令原本的 PC+4 作为 NPC；在以上情况均没有发生时，如果 IF 段的 BranchFlagsF 的值为 11，则表示预测跳转，需要将 cache 中对应的跳转地址作为 NPC。

为了做 Branch 指令与其跳转地址的存储，需要使用 cache，在此次实验中，使用的为大小为可以存储 8 个 branch 条目的基于 FIFO 替换策略的 cache，对于 BTB 中具体涉及到的代码如下，

```
reg [31:0] BranchPC [0:7];
reg [31:0] BranchPredPC [0:7];
reg PredEn[0:7];
reg [2:0] indexNum;
```

IndexNum 为 3 位的寄存器变量，循环加 1 来实现 FIFO 的替换策略，PredEn 则为 8 位的寄存器，每一位的值表示对应的条目的预测结果，BranchPC 和 BranchPredPC 则一一对应的存储分支指令的 PC 和其跳转地址的 PC。

```
output reg [1:0] BranchFlags,
output reg [2:0] BranchIndex
```

上述两个寄存器变量的更新使用组合逻辑，并用 for 循环的方式将当前 PC 和 BranchPC 中的值进行一一比较，命中则将 BranchFlags[0]置 1，在此情况下，若对应的 PredEn 为 1，则将 BranchFlags[1]置 1。并将 BranchIndex 设置为此条目的索引值。这两条信号会随流水段传至 Ex 段并经过判断后，将 BranchE 和 BranchIndex 传回 NPC_Generator 以便于预测失败的恢复操作。
具体的实现代码如下，

```
always @(*)
begin
    if(CpuRst)
    begin
        BranchFlags = 2'b00;
        BranchIndex = 3'b000;
    end
    else
    begin
        BranchFlags = 2'b00;
        BranchIndex = 3'b000;
        for(i = 0; i < 8; i=i+1)
        begin
            if(PC_In == BranchPC[i])
            begin
                BranchFlags[0] = 1'b1;
                if(PredEn[i])
                    BranchFlags[1] = 1'b1;
                BranchIndex = i;
            end
        end
    end
end
```

PredEn 的更新采用时序逻辑实现，因此 NPC_Generator 模块也加入了时钟信号作为输入，分为以下三种情况

i. 命中失败但分支成功：此时应加入相应的 BranchPC 作为新条目；

ii. 命中成功预测不分支但分支成功：此时应依据 BranchIndexE 更新 PredEn 的值（改为 0）；

iii. 命中成功预测分支但分支失败：此时应依据 BranchIndexE 更新 PredEn 的值（改为 1）；

具体的实现代码如下所示

```
always@(posedge clk)
begin
    if(CpuRst)
    begin
        indexNum <= 3'b0;
        for(i = 0; i < 8; i=i+1)
        begin
            BranchPC[i] <= 32'b0;
            BranchPredPC[i] <= 32'b0;
            PredEn[i] <= 1'b0;
        end
    end
    else
    begin
        if(BranchE == 2'b01)//匹配失败，但分支成功，加入新的表项
        begin
            BranchPC[indexNum] <= PCE;
            BranchPredPC[indexNum] <= BranchTarget;
            PredEn[indexNum] <= 1'b1;
            indexNum <= indexNum + 1;
        end
        else if(BranchE == 2'b10)//匹配成功，命中失败，但分支成功，则需将该表项使能
        begin
            PredEn[BranchIndexE] <= 1'b1;
        end
        else if(BranchE == 2'b11)//匹配成功，命中成功，但分支失败，需将该表项置为无效
        begin
            PredEn[BranchIndexE] <= 1'b0;
        end
    end
end
```

c) HazardUnit 模块修改相对简单，即当 BranchE 不是 00 时，要 Flush 相应的流水段，实际情况和 JalrE 需要 Flush 的流水段相同，因此实现如下

```
else if(BranchE != 2'b00 | JalrE)
{StallF,FlushF,StallD,FlushD,StallE,FlushE,StallM,FlushM,StallW,FlushW
} <= 10'b0001010000;
```

2. ***BHT***

BHT 的实现逻辑与 BTB 完全相同，共用同一套结构，仅需做微小的调整。

首先由于状态的增加 PredEn 需要从一位变成两位；其次由于此时预测成功也可能需要更新状态，但 BranchE 取值为 00 不一定表示为预测成功，可能为无事发生的情形，因此要将 BranchFlagsE 也进行回传，并对其 0 号位进行判断是否有命中 cache 以确定是否为预测成功。

在进行预测判定时，需进行微调，根据状态图，当 PredEn[1]为 1 时，预测分支，因此代码实现如下

```
for(i = 0; i < 8; i=i+1)
begin
    if(PC_In == BranchPC[i])
    begin
        BranchFlags[0] = 1'b1;
        if(PredEn[i][1] == 1'b1)
            BranchFlags[1] = 1'b1;
        BranchIndex = i;
    end
end
```

在更新状态时，状态相较于 BTB 有所增加，即不仅仅是预测失败需要进行状态更新，预测成功也需要状态更新，依据上述的实现以及状态转移图，实现如下：在 BranchE 为 00 时，需做额外的判断，剩余三种情况类比 BTB，但需考虑两种不同情况分别进行不同的状态转移。

```
begin
  if(BranchE == 2'b00)
  begin
      if(BranchFlagsE[0] == 1'b1)//是预测成功
      begin
         if(PredEn[BranchIndexE] == 2'b01)
             PredEn[BranchIndexE] <= 2'b00;
         else if(PredEn[BranchIndexE] == 2'b10)
             PredEn[BranchIndexE] <= 2'b11;
         else
             PredEn[BranchIndexE] <= PredEn[BranchIndexE];
      end
  end
  else if(BranchE == 2'b01)//匹配失败，但分支成功，加入新的表项
  begin
      BranchPC[indexNum] <= PCE;
      BranchPredPC[indexNum] <= BranchTarget;
      PredEn[indexNum] <= 2'b10;
      indexNum <= indexNum + 1;
  end
  else if(BranchE == 2'b10)//匹配成功，命中失败，但分支成功，则需将该表项使能
  begin
      if(PredEn[BranchIndexE] == 2'b00)
          PredEn[BranchIndexE] <= 2'b01;
      else if(PredEn[BranchIndexE] == 2'b01)
          PredEn[BranchIndexE] <= 2'b11;
      else
          PredEn[BranchIndexE] <= PredEn[BranchIndexE];
  end
  else if(BranchE == 2'b11)//匹配成功，命中成功，但分支失败，需将该表项置为无
效

  begin
```

```
            if(PredEn[BranchIndexE] == 2'b10)
                PredEn[BranchIndexE] <= 2'b00;
            else if(PredEn[BranchIndexE] == 2'b11)
                PredEn[BranchIndexE] <= 2'b10;
            else
                PredEn[BranchIndexE] <= PredEn[BranchIndexE];
        end
    end
```

四、实验结果

经过仿真测试，本次实验的两种分支预测都可以正常的运行测试代码，并得到正确的运行结果，不会由于加入分支预测而计算出错误的结果。

两种分支预测器分别运行两种测试代码得到的预测成功与失败次数如下：对于 BTB，当加入新表项时，状态设置为 1，即预测分支；对于 BHT，当加入新表项时，状态设置为 10，也是预测分支，具体统计结果如下，

	测试样例	分支总数	预测失败	预测成功	预测成功率	总时长(ns)
BTB	btb	101	2	99	98.02%	1266
	bht	110	22	88	80.00%	1538
BHT	btb	101	2	99	98.02%	1266
	bht	110	13	97	88.18%	1466

五、实验结果分析

通过观察实验统计结果，可以得到以下结论

a. 分支预测不会影响结果的正确性；
b. BHT 的效果会优于 BTB；
c. 分支预测的成功率越高，则程序运行的时间也越短；

对于 BHT 优于 BTB 的结论，是因为 bht 的测试代码中含有两层循环，当内层循环第一次结束后（即一次分支失败），BHT 状态变为 10，BTB 则变为 0,，故在下一次循环到达此处时，BHT 预测分支，但 BTB 预测不分支并在一次预测失败后改为预测会分支。在对于外层循环，两者则应表现一致，即第一次命中失败加入表项后，一直预测成功，直到跳出循环结束程序会有一次预测失败。

六、实验心得与总结

通过本次实验，我实现了 BTB 与 BHT 两种分支预测方式，对其具体的结构以及预测原理有了更加深刻的认知，并通过对运行结果的分析，更加切实的认识到了分支预测带来的性能上的提升。