

# 体系结构Lab4实验报告

*PB16050567 陈炜*

## 实验目标

- 1、实现分支预测BTB
- 2、实现分支预测BHT
- 3、统计分支收益和分支代价。

实验环境和工具

实验环境：Vivado2017.4

实验工具：Vivado自带编辑器和波形仿真工具

## 实验内容和过程

*阶段一：BTB*

**代码：**

**Judge模块：**

我将判断分支预测是否失败的功能从BTB中独立了出来，设为Judge模块。

Judge模块的输入输出如下：

```

input rst,
    input [31:0] EXpc,
    input [31:0] IDpc,          //ID段寄存器的PC, 用于判断预测和实际是否相符
    input [31:0] BrNPC,        //计算出来的跳转地址
    input BranchE,
    input [2:0] BranchTypeE,

    output reg [1:0] BTBflush,          //10的时候, 指示BTB更新设置为有效, 01的时候, 设空cache, EXpc和
    //Aluout不通过本模块传递, 00表示不用改变
    output reg [1:0] PredictMiss        //通知NpcGenerator和Hazard模块

```

其中

Judge模块主要使用BranchE, BranchTypeE, BrNPC, ID段PC, 和EX段PC进行判断。

如果BranchE为真, 表明此次分支真实情况是命中, 此时需要判断ID段寄存器中的PC(预测出来的值)是否和BrNPC相同, 如果相同, 说明预测正确, 否则错误, 需要更新BTB中和当前PC(EX段寄存器中PC值)对应的BTBcache中的值为命中的PC值。

如果BranchE为假, 并且BranchTypeE为真, 说明这是一条不命中的分支指令, 此时需要判断EX段寄存器中的PC+4是否等于ID段寄存器中的PC, 如果相等, 说明预测正确, 否则错误, 需要更新BTB中和当前PC(EX段寄存器中PC值)对应的BTBcache状态为Invalid。

在以上两种情况中, 如果遇到预测错误的情况, 需要通知NPC\_Generator模块去对应的位置取下一个PC值, 还要通知Hazard模块做相应的处理。

如果BranchE和BranchTypeE均为假, 说明这不是一条分支指令, 无需处理。

Judge的判断代码如下:

```

always@(*)
begin
    if(rst)
    begin
        BTBflush<=0;
        PredictMiss<=0;
    end
    else
    begin
        if(BranchE)          //实际命中了
        begin
            if(IDpc==BrNPC)    //预测也命中了
            begin
                BTBflush<=0;
                PredictMiss<=0;
            end
            else
            begin
                BTBflush<=2'b10;          //
                PredictMiss<=2'b10;
            end
        end
        else if(BranchTypeE!=0)    //实际没命中
        begin

```

```

        if(EXpc+3'b100==IDpc)
        begin
            BTBflush<=0;
            PredictMiss<=0;
        end
        else
        begin
            BTBflush<=2'b01;
            PredictMiss<=2'b01;
        end
    end
else          //根本不是跳转指令
begin
    BTBflush<=0;
    PredictMiss<=0;
end
end
end
end

```

## BTB模块:

BTB模块是一个由16路组成的直接映射cache，cache的内容包括，用于比较的tag，用于预测的PC值，用于标识当前块是否有效的标识位。

BTB的输入输出如下:

```

input rst,                //用于初始化
input [1:0] BTBflush,      //在纯BTB中用于判断flush与否
input [31:0] BrNPC,        //新的预测PC，既是跳转PC,BrNPC
input [31:0] EXpc,         //改变了预测值的PC,EXpc
input [31:0] CurrentPC,    //用于提取预测值

output reg [31:0] PrePC,    //预测值
output reg BTBhit          //BTBcache中是否命中

```

BTB中需要使用的地址为:

```

wire [3:0] updateAddr;
wire [26:0] updateTag;
wire [3:0] fetchAddr;
wire [26:0] fetchTag;

assign updateAddr = EXpc[5:2];    //4位地址，更新的映射地址
assign updateTag = EXpc[31:6];    //26位更新tag
assign fetchAddr = CurrentPC[5:2]; //4位地址，IF段寄存器PC映射的地址
assign fetchTag = CurrentPC[31:6];

```

BTB模块中有两个段控制逻辑:

### 1、更新cache内容（时序逻辑）：

根据从Judge中接受的BTBflush信号来在预测失败的时候更新cache的内容。

如果预测不命中而实际命中，更新EXpc对应的

cache块的内容，将其置为valid。

如果预测命中而实际不命中，将EXpc对应cache块的标志位设为0。

```
if(BTBflush==2'b10)      //需要从无效变为有效
begin
    valid[updateAddr]<=1'b1;
    Pretag[updateAddr]<=updateTag;
    PreCache[updateAddr]<=BrNPC;
    miss <= miss+1'b1;
end      //update
else if(BTBflush==2'b01)  //需要从有效变为无效
begin
    valid[updateAddr]=0;
    miss <= miss + 1'b1;
end
```

## 2、为NPC\_Genarator提供预测PC值（组合逻辑）

直接判断当前IF段寄存器的PC对应的映射地址上的valid位是否有效即可

```
if(valid[fetchAddr]&&fetchTag==Pretag[fetchAddr])
begin
    BTBhit<=1'b1;
    PrePC<=PreCache[fetchAddr];
end      //命中
else
begin
    BTBhit<=0;
end
```

## NPC\_Generator:

因为不再考虑Jal和Jalr，所以NPC\_Generator的代码变为：

当PredictMiss为10的时候，说明预测不跳转但是实际上跳转了，这个时候下一条指令应该是BranchTarget。

当PredictMiss为01的时候，说明预测跳转但是实际上不跳转，这个时候下一条指令应该是跳转指令的PC+4，也就是EX段寄存器的PC+4。

当PredictMiss为0的时候，一切正常，只要根据BTBhit信号判断即可。

```
always @(*)
begin
    if(PredictMiss==2'b10)      //这个判断要在其他判断之前，因为如果miss了，其他项中用于判断的PC
    值就不可靠了
        PC_In<=BranchTarget;
    else if(PredictMiss==2'b01)
        PC_In<=Expc+4;
    else if(BTBhit)
        PC_In <= PrePC;
    else
        PC_In <= PCF+4;
end
```

## HazardUnit:

HazardUnit中做的唯一修改就是引入PredictMiss输入，如果发生了PredictMiss,预测错误，FlushEX和ID段。

```
else if(PredictMiss!=0 | JalrE)
    //修改了此处，将BranchE改成了PredictMiss
    {StallF,FlushF,StallD,FlushD,StallE,FlushE,StallM,FlushM,StallW,FlushW} <=
    10'b0001010000;
```

## 实验结果:

### btb.inst:

测试结果:

> miss[7:0]	02				02			
> [6][31:0]	5051				5051			

如图，miss次数为2次，结果为5051。

产生miss的原因:

- 1、初始状态是不命中，所以在第一次分支跳转的时候miss一次。
- 2、循环终止的时候，预测器预测跳转，但是实际上不跳转，产生一次miss。

共计两次。

### bht.inst:

测试结果:

> miss[7:0]	22				22			
> [6][31:0]	451				451			

如图，miss次数为22次，最终结果451。

其中，外循环在进入和退出的时候各产生一次Miss,共2次。

每个内循环也在进入和退出的时候各产生一次Miss,10次内循环产生20次。

共计22次。

## 阶段二: BHT+BTB

## 代码:

### BTB模块:

在BHT+BTB的设计模式中，BTB的功能有所改变，变成一个类似cache的角色。

BTB的主要组成部分还是两个:

1、判断当前PC(IF段寄存器中的PC)是否在BTB中命中：

和仅有BTB构成的分支预测器不同的是，BTBhit信号不是直接传给NPC\_Generator而是传给BHThit做进一步处理。

而地址则还是传给NPC\_Generator。

```
always@(*)          //读取
begin
    if(rst)
        BTBhit<=0;
    else
        begin
            if(valid[fetchAddr]&&fetchTag==Pretag[fetchAddr])
                begin
                    BTBhit<=1'b1;
                    PrePC<=PreCache[fetchAddr];
                end          //命中
            else
                begin
                    BTBhit<=0;
                end
        end
    end          //~rst
end
```

2、根据情况更新BTB中的tag或预测PC：

在BHT+BTB的模式中，预测失败并不会更新BTB，只有当以下两种情况发生的时候会更新BTB：

- 1、出现了一条命中的跳转指令，但是这条指令映射到BTB中的地址的有效位是0.
- 2、出现了一条命中的跳转指令，但是这条指令映射到BTB中的地址的tag和这条指令的tag冲突了。

```
always@(*)          //决定BTB是否flush
begin
    if(rst)
        BTBchange<=0;
    else
        begin
            if(BranchE)          //命中了就需要决定是不是要改BTB的缓存了，冲突或者原本是invalid就改缓存内容
                begin
                    if(valid[updateAddr] == 0 || updateTag!=Pretag[updateAddr])
                        BTBchange<=1'b1;
                    else
                        BTBchange<=0;
                end
            else
                BTBchange<=0;          //与单BTB不同，不命中就不改缓存
        end
    end
end
```

```

if(BTBchange)      //BTBcache需要改变
begin
    valid[updateAddr]<=1'b1;
    Pretag[updateAddr]<=updateTag;
    PreCache[updateAddr]<=BrNPC;
end                //update

```

## BHT模块:

因为BTB的cache有16路，所以BHT的状态机也有16个：

```

parameter SN=0,WN=2'b01,WT=2'b10,ST=2'b11;      //分别是强不命中，弱不命中，弱命中，强命中
reg [1:0] BHT_State [0:15];                      //BHT状态机

```

BHT主要由三个部分构成：

1、根据IF段寄存器输入的PC，给出预测：

当且仅当当前的状态机是命中状态并且BTBhit为1的时候才给出分支跳转命中的信号，否则预测不命中。

```

always@(*)          //根据当前状态决定命中与否
begin
    if(rst)
    begin
        BHThit<=0;
    end
    else
    if(BHT_State[fetchAddr]==2'b01 || BHT_State[fetchAddr]==0)
        BHThit<=0;
    else
    begin
        if(BTBhit)
            BHThit<=1'b1;
        else
            BHThit<=0;
    end
end

```

2、根据实际跳转情况改变状态机：

与预测值和实际值是否相同无关，仅仅根据实际跳转情况改变状态：

```

always@(posedge clk) //根据实际跳转情况改变状态
begin
    if (rst)
    begin
        BHT_State[updateAddr]<=WN;
        miss <=0;
    end
    else
    begin
        if(BranchE)
        begin

```

```

        case(BHT_State[updateAddr])
        SN:    begin
        BHT_State[updateAddr]<=WN;
        miss <= miss+1'b1;
        end
        WN:
        begin
        BHT_State[updateAddr]<=ST;
        miss <= miss +1'b1;
        end
        WT:    BHT_State[updateAddr]<=ST;
        ST:    BHT_State[updateAddr]<=ST;
        endcase
        end
    else
    begin
    if(BranchTypeE!=0)    //不命中
    begin
        case(BHT_State[updateAddr])
        SN:    BHT_State[updateAddr]<=SN;
        WN:    BHT_State[updateAddr]<=SN;
        WT:    begin
            BHT_State[updateAddr]<=SN;
            miss <=miss +1'b1;
            end
        ST:    begin
            BHT_State[updateAddr]<=WT;
            miss <=miss +1'b1;
            end
        endcase
    end
    //else情况是根本不是branch指令，没有操作
    end
    end
end
end

```

3、当预测失败的时候，通知NPC\_Generator和HazzardUnit：

这部分逻辑在三个模块中的工作原理与仅有BTB的时候相同，不再赘述。

```

always@(*)    //如果预测错误，改变状态的同时，还要通知Hazzard和NPC，
begin
    if(rst)
        PredictMiss<=0;
    else
    begin
        if(BranchE)    //实际命中了
        begin
            if(IDpc==BrNPC)    //预测也命中了

            PredictMiss<=0;
            else
            PredictMiss<=2'b10;
            end
        else if(BranchTypeE!=0)    //实际没命中
    end
end

```



```

begin
    if(EXpc+3'b100==IDpc)
        PredictMiss<=0;
    else
        PredictMiss<=2'b01;
    end
else //根本不是跳转指令
    PredictMiss<=0;
end
end

```

## NPC\_Generator模块：

NPC\_Generator的工作原理和仅有BTB的时候类似：

```

always @(*)
begin
    if(PredictMiss==2'b10) //这个判断要在其他判断之前，因为如果miss了，其他项中用于判断的PC
    值就不可靠了
        PC_In<=BranchTarget;
    else if(PredictMiss==2'b01)
        PC_In<=Expc+4;
    else if(BHThit)
        PC_In <= PrePC;
    else
        PC_In <= PCF+4;
end

```

## HazzardUnit:

HazzardUnit同上。

```

else if(PredictMiss!=0 | JalrE)
    //修改了此处，将BranchE改成了PredictMiss
    {StallF,FlushF,StallD,FlushD,StallE,FlushE,StallM,FlushM,StallW,FlushW} <=
    10'b0001010000;

```

## 实验结果：

### btb.inst:

测试结果：

> miss[7:0]	02
> [6][31:0]	5051

miss两次，结果5051。

情况和仅有BTB时完全一样。

### bht.inst:

测试结果：

> 🧠 miss[7:0]	13
> 🧠 [6][31:0]	451

如图，miss13次，最终结果451.

miss的原因是：

1、外层循环一共miss2次。

2、内层循环：

第一次内层循环开始和结束的时候各miss一次，共2次。

之后的每一次循环只在结束的时候miss一次，共9次。

共计13次。

## 实验总结

### 一、问题回答：

1、对于一条分支指令，当动态分支预测命中，实际命中时，比不用分支预测少用几个周期？

如果不使用分支预测而使用默认不跳转的策略的话，一条命中的分支指令将会在其到达EX段时清空IF和ID段的内容，也就是会额外花费两个周期。

2、对于一条分支指令，动态分支预测失败比非跳转指令多用几个周期？

与上面的情况类似的，一条预测失败的分支指令将会在其到达EX段时清空IF和ID段的内容，同样会多花费两个周期。

### 二、主要问题：

1、对BTB的cache理解有误，用组合逻辑实现BTB和BHTcache的更新。

### 三、实验收获：

更好的理解了分支预测器，虽然只实现了一个非常简单的分支预测器，并且实验测试数据也很小，但是对运行时间的改善也是非常明显的。