计算机体系结构 实验四 分支预测器

PB18151866 龚小航

【实验目的】

1、实现 BTB（Branch Target Buffer）和 BHT（Branch History Table）两种动态分支预测器
2、体会动态分支预测对流水线性能的影响

【实验环境】

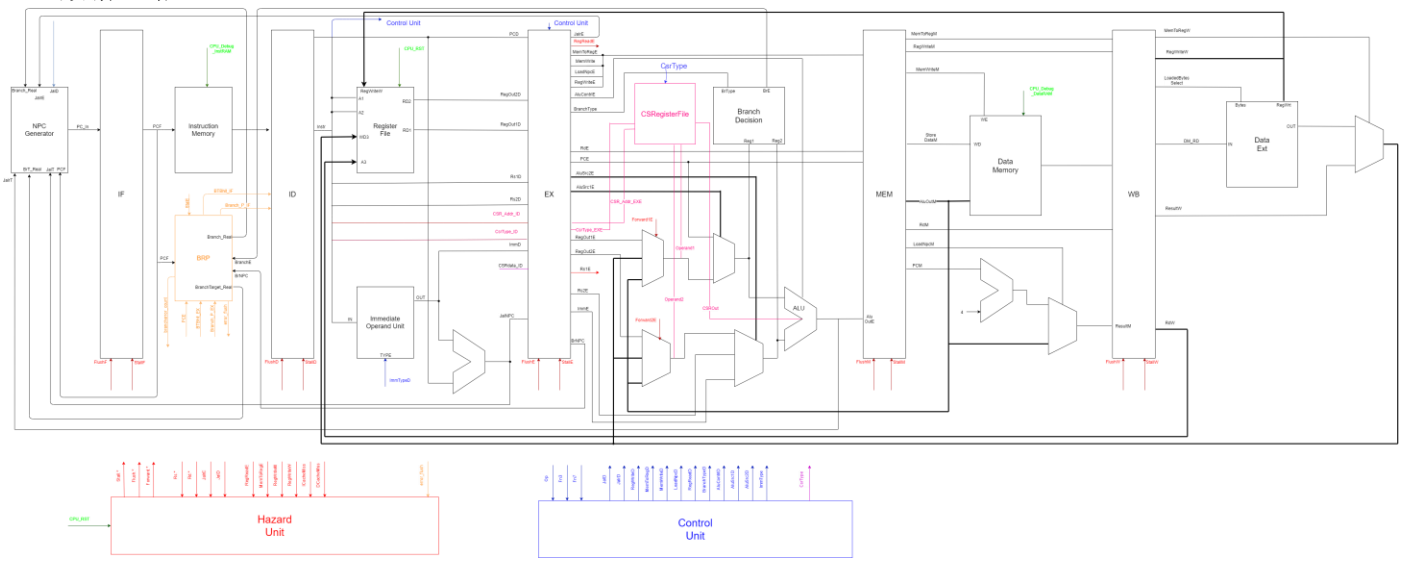
Xilinx Vivado 2019.1

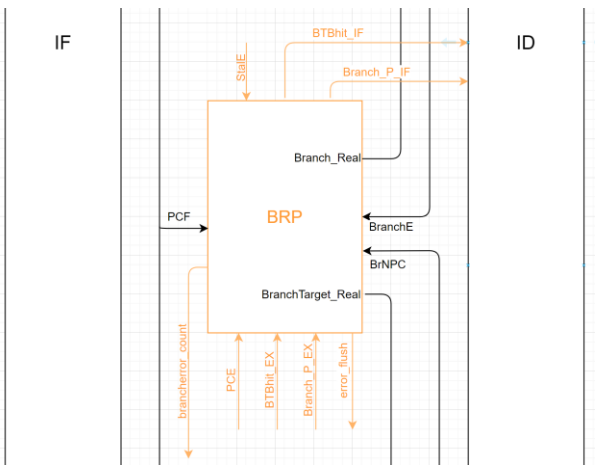
【实验要求】

1、在 Lab3 阶段二的 RV32I Core 基础上，实现 BTB
2、实现 BHT
3、阶段二需要在阶段一的基础上实现，不能仅实现阶段二

【实验过程及具体实现】

数据通路：





一、实现 1 位分支预测器 BTB（Branch Target Buffer）以及 2 位分支预测器 BHT（Branch History Table）

在设计上，只是预测位数不同。为了提高文件的整体性和聚合度，在 BRP 模块文件中使用条件编译选择使用 BTB 或是 BHT。

- 分支预测器的原理如下：

从整体上看来(从外部看)，整个分支预测器整合为一个模块，接受原本分支相关的所有信号，例如 BranchE, BranchTarget, PCE 等，BRP 模块根据这些信号生成真正送入 NPCGenerator 模块的选择地址信号 (Branch_Real, BranchTarget_Real) 以及预测错误时送入 Harzard 模块的 error_flush 信号

从 BRP 模块内部看，每个时钟周期到来时，BRP 需要做两件事：

1、 根据 IF 段得到的信息(PCF)判断当前地址是否对应一条曾经发生过的分支指令，并使用 BTB 或 BHT 的信息来选择下一条指令的地址（通过生成真正控制 NPC 的 Real 信号），同时生成 IF 段的命中标记 hit_IF 信号和 IF 段的预测信息记录信号 Branch_P_IF，它们通过流水线段间寄存器将一路传递到 EX 段。

2、 另一方面，在时钟周期上升沿来临时，BRO 还需要接收 EX 段传回来的 hit_EX 和 Branch_P_EX 信号，以更新 BTB 或 BHT 表。特别的，当 hit_EX = 0 且 BranchE = 1 时，代表某条分支指令第一次发生跳转，它在 IF 段未命中，应该把它的信息加入 BTB 或 BHT（唯一加入信息方法）

- 具体实现如下所示：

1、本地变量宏定义指定根据 PC 的低位寻址的长度。只需要保证 2^BTBSIZE>程序指令数 即可。这里的 buffer 实现类似于直接映射，一条指令地址只能存放在对应的一个位置。这么做的好处是寻址速度快，但同时也带来了空间上的浪费。此处定义 BTBSIZE=12

```
27        localparam BTBSIZE = 12;//PC[13：2] 12位直接映射到BTB表项上
```

BTB 的 buffer：

```
36    reg [ 32 + BTBSIZE : 0 ] Branch_Target_Buffer[ (1<<BTBSIZE) - 1 : 0 ];//PC的低12位直接映射到BTB  [0]是否跳转 [1-32]target，[33-44] PC
```

BHT 的 buffer：

```
111   reg [ 33 + BTBSIZE : 0 ] Branch_Target_Buffer[ (1<<BTBSIZE) - 1 : 0 ];//PC的低12位直接映射  [1:0]是否跳转 [2-33]target，[34-45] PC
```

将 PC 的[13:2]放在 buffer 的高位，然后再放 32 位目标地址，最后根据 BTB 或 BHT 放 1 位或 2 位的预测位。使用时根据预测位值生成预测信号，同时根据 EX 段回传值更新它们。

2、IF 段使用 BTB 或 BHT,同时生成预测错误的清除信号以及真正的跳转信号：（以 BTB 为例）

使用组合逻辑即可实现。

```
always@(*) begin  //IF段使用BTB表
    if( PCF[ BTBSIZE + 1 : 2 ] == Branch_Target_Buffer[ PCF[BTBSIZE + 1 : 2] ][ 33 + BTBSIZE : 34 ] ) begin
        //BTB命中，说明它是分支指令，根据[0]的值选择NPC
        BTBhit_IF = 1'b1;
        if(Branch_Target_Buffer[ PCF[BTBSIZE + 1 : 2] ][1:0] == 2'b11 || Branch_Target_Buffer[ PCF[BTBSIZE + 1 : 2] ][1:0] == 2'b10) begin
            //预测跳转，直接将这些信号传给NPC生成模块将分支地址作为下一个PC
            Branch_Real = 1;
            BranchTarget_Real = Branch_Target_Buffer[ PCF[BTBSIZE + 1 : 2] ][ 33 : 2 ];
            Branch_P_IF = 1;
        end
        else begin //预测不跳转
            Branch_Real = 0;
            Branch_P_IF = 1'b0;
        end
    end
    else begin//BTB未命中
        Branch_Real = 0;
        Branch_P_IF = 1'b0;
        BTBhit_IF = 1'b0;
    end

    error_flush = 0;
    if((BranchE == 1'b1) && (BTBhit_EX == 1'b0)) begin  //增加新的BTB表项,BTB未命中且BranchE==1
        error_flush = 1;          //预测错误，清除前两段段间流水寄存器
        Branch_Real = 1;          //实际跳转
        BranchTarget_Real = brnpc_reg;//实际跳转地址
    end
    if(BTBhit_EX == 1'b1) begin //当前EX段是一条分支指令且早在BTB命中，根据当前BranchE更新BTB表
        if((Branch_P_EX == 1'b0) && (BranchE == 1'b1)) begin
            error_flush = 1;//预测错误，清除前两段段间流水寄存器
            Branch_Real = 1;
            BranchTarget_Real = brnpc_reg;
        end
        if((Branch_P_EX == 1'b1) && (BranchE == 1'b0)) begin
            error_flush = 1;//预测错误，清除前两段段间流水寄存器
            Branch_Real = 1;
            BranchTarget_Real = PCE + 32'h4;
        end
    end
end
```

本质上是根据当前输入的 PCF 在 buffer 里寻找它的跳转记录（不管它实际上是什么指令），并生成真正的决定 NPC 的分支类信号。另一方面，error_flush 信号有效的条件是预测值和实际的分支选择相反，此时清除因为预测而错误读入的 IF, ID 段的指令，控制 Branch_Real, BranchTarget_Real 使 NPCGenerator 的下一个输出为正确地址。

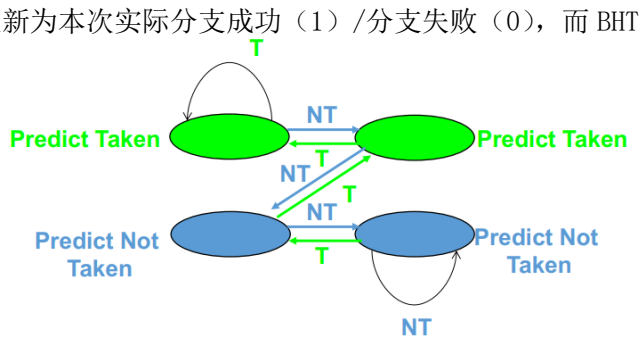
3、接收 EX 段传回的信号并更新 BTB,BHT：（以 BHT 为例）

在每个时钟周期的上升沿进行更新即可。

```
always@(posedge clk) begin //维护BHT表
    if(BranchE && (BTBhit_EX == 1'b0)) begin  //增加新的BTB表项,BTB未命中且BranchE==1
        Branch_Target_Buffer[ PCE[BTBSIZE + 1 : 2] ][ 33 + BTBSIZE : 34 ] = PCE[ BTBSIZE + 1 : 2 ];
        Branch_Target_Buffer[ PCE[BTBSIZE + 1 : 2] ][ 33  : 2 ] = brnpc_reg;
        Branch_Target_Buffer[ PCE[BTBSIZE + 1 : 2] ][1:0] = 2'b11;
    end
end

always@(posedge clk) begin
    if(BTBhit_EX == 1'b1) begin //当前EX段是一条分支指令且早在BTB命中，根据当前BranchE更新BTB表
        if((Branch_P_EX == 1'b0) && (BranchE == 1'b1)) begin
            if(Branch_Target_Buffer[ PCE[BTBSIZE + 1 : 2] ][1:0] == 2'b00) begin
                Branch_Target_Buffer[ PCE[BTBSIZE + 1 : 2] ][1:0] = 2'b01;
            end
            else if(Branch_Target_Buffer[ PCE[BTBSIZE + 1 : 2] ][1:0] == 2'b01)begin
                Branch_Target_Buffer[ PCE[BTBSIZE + 1 : 2] ][1:0] = 2'b10;
            end
        end
        if((Branch_P_EX == 1'b1) && (BranchE == 1'b0)) begin
            if(Branch_Target_Buffer[ PCE[BTBSIZE + 1 : 2] ][1:0] == 2'b10) begin
                Branch_Target_Buffer[ PCE[BTBSIZE + 1 : 2] ][1:0] = 2'b01;
            end
            else if(Branch_Target_Buffer[ PCE[BTBSIZE + 1 : 2] ][1:0] == 2'b11)begin
                Branch_Target_Buffer[ PCE[BTBSIZE + 1 : 2] ][1:0] = 2'b10;
            end
        end
        if((Branch_P_EX == 1'b1) && (BranchE == 1'b1)) begin
            if(Branch_Target_Buffer[ PCE[BTBSIZE + 1 : 2] ][1:0] == 2'b10) begin
                Branch_Target_Buffer[ PCE[BTBSIZE + 1 : 2] ][1:0] = 2'b11;
            end
        end
        if((Branch_P_EX == 1'b0) && (BranchE == 1'b0)) begin
            if(Branch_Target_Buffer[ PCE[BTBSIZE + 1 : 2] ][1:0] == 2'b01) begin
                Branch_Target_Buffer[ PCE[BTBSIZE + 1 : 2] ][1:0] = 2'b00;
            end
        end
    end
end
```

BTB 的跳转逻辑更新为本次实际分支成功（1）/分支失败（0），而 BHT 的更新逻辑如下图所示：



4、在 NPCGenerator 模块中新增跳转优先级设定：

由于增加了预测模块，分支指令有可能在 IF 段发生（预测）也可能在 EX 段发生（预测错误），要对这两种不同的情况设立不同的优先级，以免干扰到 JAL 指令（ID 段执行）的优先级。判断的依据是当前是否为预测错误才发生的跳转，即 error_flush 信号是否有效：

```
always@(*)begin
    if(error_flush) begin
        if(JalrE) begin
            PC_In = JalrTarget;
        end
        else if(BranchE) begin
            PC_In = BranchTarget;
        end
        else if(JalD) begin
            PC_In = JalTarget;
        end
        else begin
            PC_In = PCF + 32'h00000004; //pc+4
        end
    end
    else begin
        if(JalrE) begin
            PC_In = JalrTarget ;
        end
        else if(JalD ) begin
            PC_In = JalTarget ;
        end
        else if(BranchE) begin
            PC_In = BranchTarget;
        end
        else begin
            PC_In = PCF + 32'h00000004; //pc+4
        end
    end

end
```

5、在 Harzard 模块中删除 BranchE 相关的信息，并根据 error_flush 值设立新的 Flush 规则

```
/*else if(BranchE) begin   //分支，EXE段执行分支，清除ID,EX寄存器，IF取同一条指令
    FlushD = 1'b1;  FlushE = 1'b1; FlushM = 1'b1;
end*/
else if(error_flush) begin   //预测失败
     FlushD = 1'b1;  FlushE = 1'b1; FlushM = 1'b1;
end
```

二、统计分支次数与预测成功与失败的次数：

统计过程在 TOP 模块执行。分支总次数可以通过 EX 段的 BranchTypeE 和 StallE 信号共同确定。如下所示：

```
always@(posedge CPU_CLK) begin
    if(CPU_RST) begin
        branch_count <= 0;
    end
    if(StallE == 1'b0 && BranchTypeE != `NOBRANCH) begin
        branch_count <= branch_count + 1;
    end
end
```

另一方面，由于分支预测器在一条指令不在 BTB 或 BHT 中时采用的是默认分支不发生策略，若某条分支语句不在 BTB 或 BHT 中且它本次不跳转此时 BTB 或 BHT 模块完全不知道默认成功预测了一次分支。因此统计 预测成功次数是不直观的，但是统计预测错误次数却很简单。因此正确预测次数采用总值-错误次数的方法求出：

```
initial brancherror_count = 0;
//维护预测错误次数
always@(posedge clk) begin
    if(StallE) begin
        brancherror_count <= brancherror_count;
    end
    else begin
        if(BranchE && (BTBhit_EX == 1'b0))begin//增加新表项时预测错误
            brancherror_count <= brancherror_count + 1;
        end
        if(BTBhit_EX == 1'b1)begin
            if((Branch_P_EX == 1'b0) && (BranchE == 1'b1)) begin  //预测错误
                brancherror_count <= brancherror_count + 1;
            end
            if((Branch_P_EX == 1'b1) && (BranchE == 1'b0)) begin
                brancherror_count <= brancherror_count + 1;
            end
        end
    end
end
```

【实验结果】

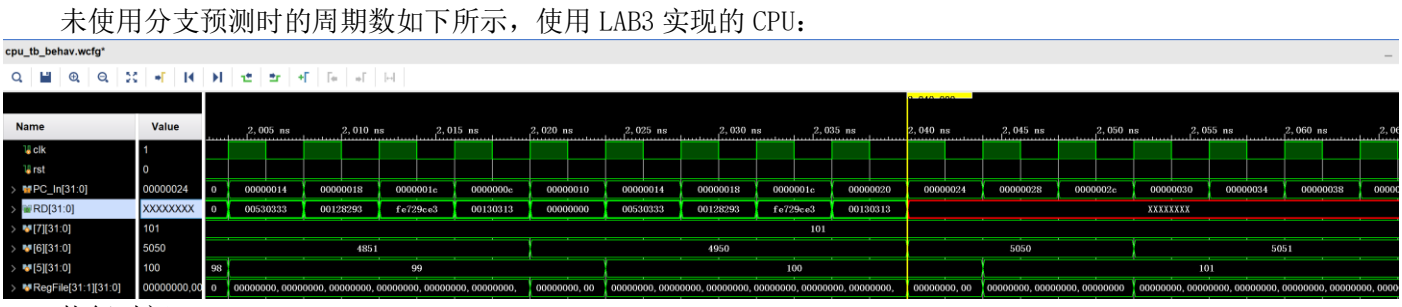
按照四个测试样例分别给出执行结果与分析：(BHT 在新加入条目时状态为 11)

1、btb.S

```
.org 0x0
    .global _start
_start:
    addi t0, zero, 0
            00000293
    addi t1, zero, 0
            00000313
    addi t2, zero, 101
            06500393
for:
    add  t1, t1, t0
            00530333
    addi t0, t0, 1
            00128293
    bne  t0, t2, for
            fe729ce3
    addi t1, t1, 1
            00130313
```

时钟周期统一为 4ns

未使用分支预测时的周期数如下所示，使用 LAB3 实现的 CPU：



执行时间 2040ns

使用 BTB 预测方案时的结果如下所示：

执行时间 1256ns，分支次数 101，正确预测 99 次，错误预测 2 次

使用 BHT 预测方案时的结果如下所示：

执行时间 1256ns，分支次数 101，正确预测 99 次，错误预测 2 次

综上：

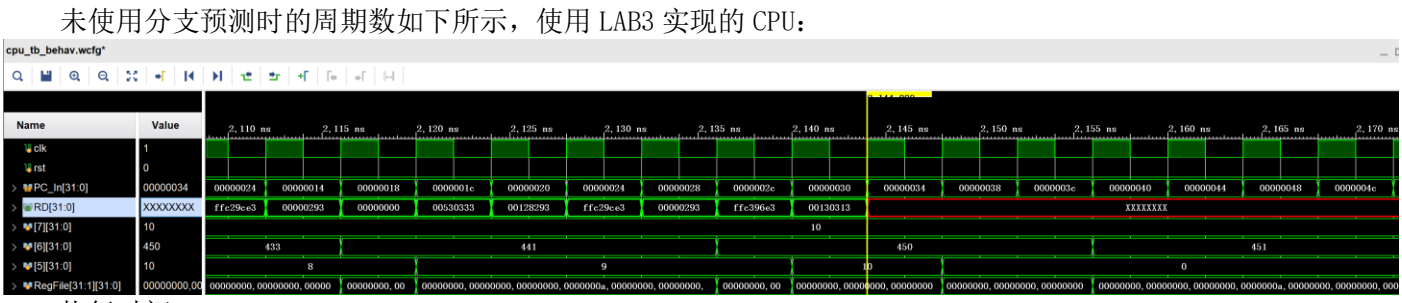
测试用例	预测策略	运行时间(ns)	分支次数	预测成功次数	预测失败次数
btb.S	默认不分支	2040	101	/	/
	BTB	1256		99	2
	BHT	1256		99	2

2、bht.S

```
.org 0x0
    .global _start
_start:
    addi t0, zero, 0
            00000293
    addi t1, zero, 0
            00000313
    addi t2, zero, 0
            00000393
    addi t3, zero, 10
            00a00e13
for_out:
    addi t2, t2, 1
            00138393
for_in:
    add  t1, t1, t0
            00530333
    addi t0, t0, 1
            00128293
    bne  t0, t3, for_in
            ffc29ce3
    addi t0, zero, 0
            00000293
    bne  t2, t3, for_out
            ffc396e3
    addi t1, t1, 1
            00130313
```

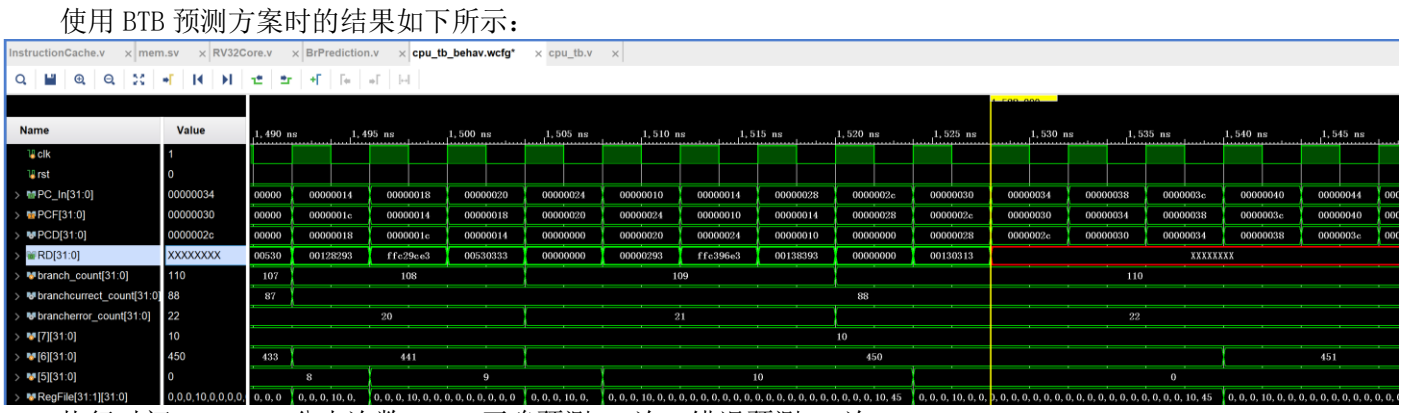
时钟周期统一为 4ns

未使用分支预测时的周期数如下所示，使用 LAB3 实现的 CPU：



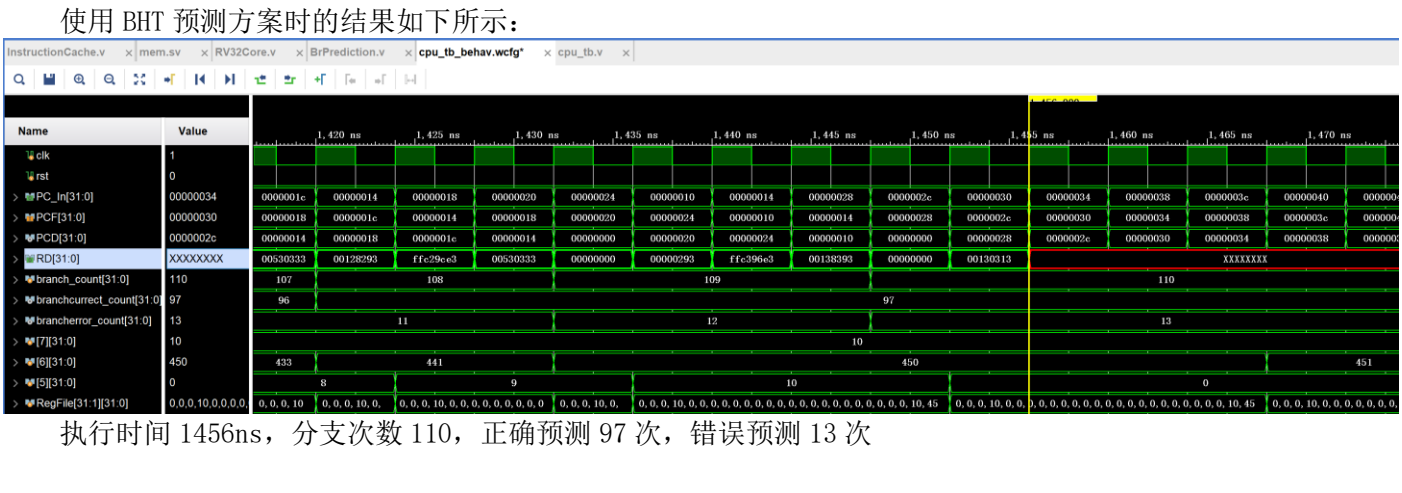
执行时间 2144ns

使用 BTB 预测方案时的结果如下所示：



执行时间 1528ns，分支次数 110，正确预测 88 次，错误预测 22 次

使用 BHT 预测方案时的结果如下所示：



执行时间 1456ns，分支次数 110，正确预测 97 次，错误预测 13 次

综上：

测试用例	预测策略	运行时间(ns)	分支次数	预测成功次数	预测失败次数
bht.S	默认不分支	2144	110	/	/
	BTB	1528		88	22
	BHT	1456		97	13

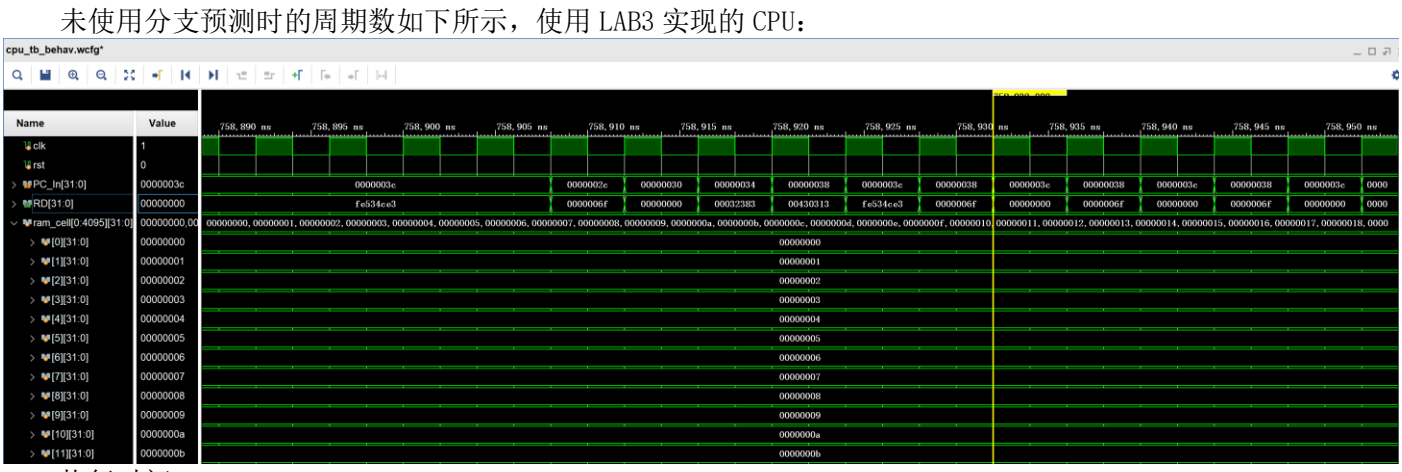
3、256 个数的 QSort：

Cache 参数：

```
cache #(
    .LINE_ADDR_LEN  ( 1              ),
    .SET_ADDR_LEN   ( 1              ),
    .TAG_ADDR_LEN   ( 10             ),
    .WAY_CNT        ( 1              )
) cache_test_instance (
```

时钟周期统一为 4ns

未使用分支预测时的周期数如下所示，使用 LAB3 实现的 CPU：

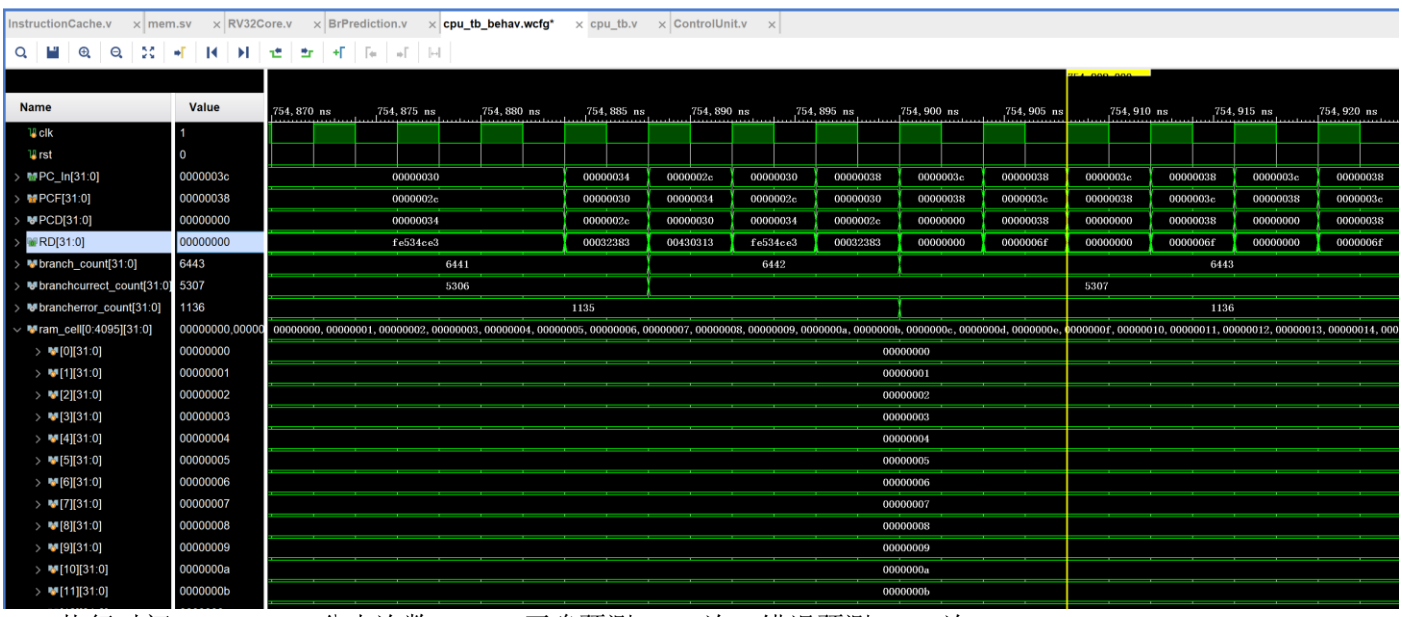


执行时间 758932ns

使用 BTB 预测方案时的结果如下所示：

执行时间 760972ns，分支次数 6443，正确预测 4549 次，错误预测 1894 次

使用 BHT 预测方案时的结果如下所示：



执行时间 754908ns，分支次数 6443，正确预测 5307 次，错误预测 1136 次

综上：

测试用例	预测策略	运行时间(ns)	分支次数	预测成功次数	预测失败次数
QuickSort.S	默认不分支	758932	6443	/	/
	BTB	760972		4549	1894
	BHT	754908		5307	1136

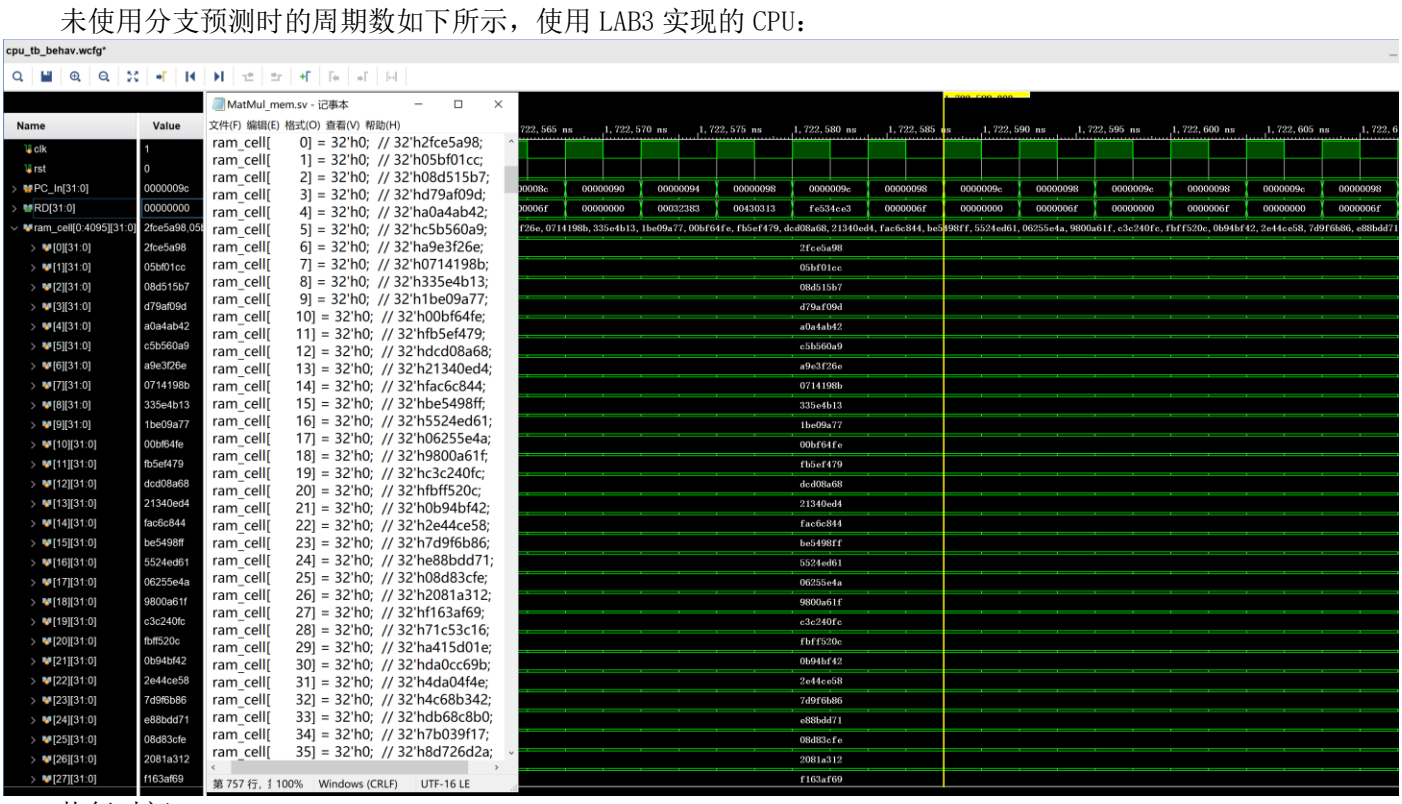
4、矩阵乘法：

cache 参数：

```
cache #(
    .LINE_ADDR_LEN  ( 1             ),
    .SET_ADDR_LEN   ( 1             ),
    .TAG_ADDR_LEN   ( 10            ),
    .WAY_CNT        ( 1             )
) cache_test_instance (
```

时钟周期统一为 4ns

未使用分支预测时的周期数如下所示，使用 LAB3 实现的 CPU：

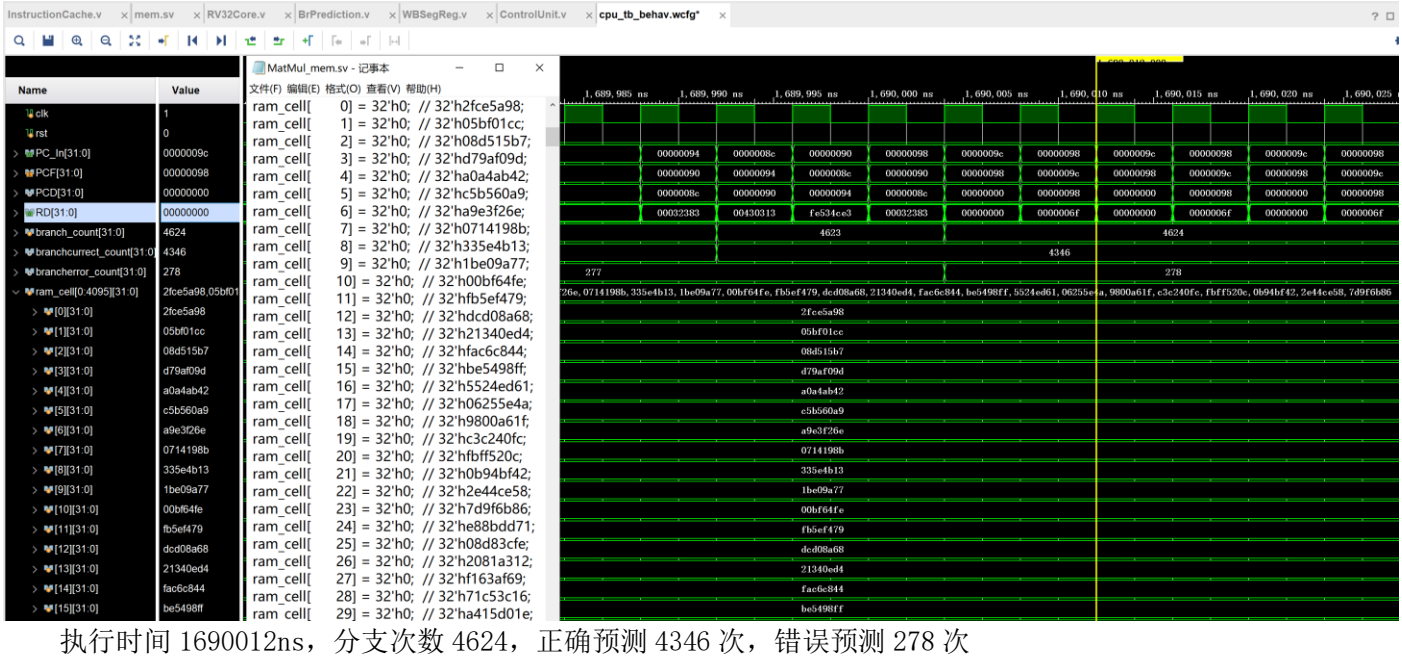


执行时间 1722588ns

使用 BTB 预测方案时的结果如下所示：

执行时间 1692172ns，分支次数 4624，正确预测 4076 次，错误预测 548 次

使用 BHT 预测方案时的结果如下所示：



执行时间 1690012ns，分支次数 4624，正确预测 4346 次，错误预测 278 次

综上：

测试用例	预测策略	运行时间(ns)	分支次数	预测成功次数	预测失败次数
MatMul.S	默认不分支	1722588	4624	/	/
	BTB	1692172		4076	548
	BHT	1690012		4346	278

5、综合分析：

测试用例	预测策略	运行时间(ns)	运行周期	周期差值	分支次数	预测成功次数	预测失败次数
btb.S	默认不分支	2040	510	0	101	/	/
	BTB	1256	314	-196		99	2
	BHT	1256	314	-196		99	2
bht.S	默认不分支	2144	536	0	110	/	/
	BTB	1528	382	-154		88	22
	BHT	1456	364	-172		97	13
QuickSort.S	默认不分支	758932	189733	0	6443	/	/
	BTB	760972	190243	510		4549	1894
	BHT	754908	188727	-1006		5307	1136
MatMul.S	默认不分支	1722588	430647	0	4624	/	/
	BTB	1692172	423043	-7604		4076	548
	BHT	1690012	422503	-8144		4346	278

分支收益和分支代价是由流水线的设计决定的，与动态分支预测的种类无关。对于任何一个样例，预测正确的代价为-2 周期，预测错误对于不预测来说代价为 0.因为不采取分支策略时，遇到分支指令直接令流水线停顿两个周期。

采用分支预测能够在循环较多的程序中取得一定程度的优化效果。从统计数据上来看，由于两位的分支预测器能够容纳表示更多的信息，在通常情况下 2 位预测器能取得比 1 位预测器更好的效果。只有某些特定的极端特例下会出现 1 位预测器效果更好的情况。而采用动态分支预测并不一定能真的带来性能上的提升，某些情况下采用动态分支预测不如使用静态预测。（快排样例的 BTB）

分支预测带来的性能提升并不是决定性能的主要因素。Cache 缺失才是程序运行时间长的主要原因。

对于一般的 for，while 循环来说，第一次分支不论是 BTB 还是 BHT 都会预测错误，而最后退出循环时也会产生一次错误。而两位的预测器在第二次进入循环时将预测正确，1 位预测器就会预测错误。

6、填表：

BTB	BHT(11||10)	REAL	NPC_PRED	Flush	NPC_Real	BTB update
Y	Y	Y	BUF	N	BUF	N
Y	Y	N	BUF	Y	PCE+4	N
Y	N	Y	PCF+4	Y	BUF	N
Y	N	N	PCF+4	N	PCE+4	N
N	Y	Y	PCF+4	Y	BrT	Y
N	Y	N	PCF+4	N	PCE+4	N
N	N	Y	PCF+4	Y	BrT	Y
N	N	N	PCF+4	N	PCE+4	N

【实验总结】

1. BRP 模块决定 NPC 时需要比较其与 JAL 指令的优先级
2. 统计分支数目，预测成败数目时需要考虑 Stall 信号
3. verilog 编程中不能把 wire 信号写进时序逻辑，需要为其分配一个寄存器。原因是 wire 信号的不稳定导致在时序电路中会产生不稳定的结果。
4. 动态分支策略有时不比静态分支预测更好，而且使电路结构更复杂