需要更新的情况

返回地址预测（ras)

需要进行返回地址更新的情况：

- ras miss: 遇到return指令，但l0 btb中不存在该表项或者命中的表项的ras标志位为0，即没有命中L0 BTB。此时更新L0 BTB的对应的表项。这样当下一次该return指令到达时，就可以直接使用L0 BTB中的预测结果进行PC的重定向。
- ras mistaken: 不是return指令，但命中的表项ras标志位为1。即指令包中没有return指令，却发生了L0 BTB中return的命中的情况，`l0_btb_ras_mistaken` 信号为1。此时就将这个表项命中失效，当下一个相同PC的指令包到达L0 BTB时不命中。
- ras mispred: 是return指令，命中的表项ras标志位也为1，但返回地址计算错误

  从L0 BTB读出的跳转目标会传递到IP级，在IP级和从ras模块中栈顶读出的值比较是否相等判断是否预测正确，如果发生预测错误则根据ras模块堆栈中的信息更新L0 BTB。

```
assign l0_btb_ras_update     = (l0_btb_ras_miss
                              || l0_btb_ras_mispred
                              || l0_btb_ras_mistaken);
//ras的miss，mispred，mistaken都归于同一类更新
assign l0_btb_ras_mistaken = ibctrl_ibdp_if_chgflw_vld
                          && ipdp_ibdp_l0_btb_ras
                          && !ibctrl_ibdp_ip_chgflw_vld
                          && ib_data_vld
                          && !(|ibdp_hn_preturn[7:0]);
assign l0_btb_ras_miss     = (ibctrl_ibdp_if_chgflw_vld
                           && !ipdp_ibdp_l0_btb_ras
                           || !ipdp_ibdp_l0_btb_hit)
                          && l0_btb_st_wait
                          && ib_data_vld
                          && (|ibdp_hn_preturn[7:0])
                          && cp0_ifu_ras_en;
assign l0_btb_ras_mispred = ibctrl_ibdp_if_chgflw_vld
                          && ipdp_ibdp_l0_btb_ras
                          && !ipdp_ibdp_l0_btb_ras_pc_hit
                          && l0_btb_st_wait
                          && ib_data_vld
                          && (|ibdp_hn_preturn[7:0])
                          && cp0_ifu_ras_en;
```

分支跳转预测 (br)

表项缺失

根据源码可以看到，筛选出来存入l0 btb的指令PC对应的指令类型有三种，分别为strongly taken的条件跳转指令（预测状态位为11）、绝对跳转指令、以及return指令。

```
assign l0_btb_miss = (!ipdp_ipctrl_l0_btb_vld || ipdp_ipctrl_l0_btb_ras &&
ipdp_ipctrl_l0_btb_vld)
                  && (
                      branch_taken && (bht_data[1:0] == 2'b11) ||
                      branch_ntake
                     )
                  && l0_btb_ipctrl_st_wait
                  && ip_vld
                  && !ip_expt_vld;
```

在IP级给出分支表项缺失的信号后，延迟一拍传递给IB级，作为l0 btb更新的控制信号源。

预测错误br_mispred

预测错误信号可由ibdp与addrgen两个模块给出。

- ibdp：

  ibdp的`br_mispred`信号实际上在**IP级**给出，产生`l0_btb_mispred`信号传递到IB级用于更新L0 BTB，包括两种情况：

  1. bht预测跳转，l0 btb也发送跳转指令，与bht结果相符。但此时命中的表项并不包含在l1 btb中，违背包含关系
  2. bht预测不跳转，但l0 btb发送了跳转指令。

  ```
  assign l0_btb_mispred = ip_if_pcload
                          && !ipdp_ipctrl_l0_btb_ras
                          && !l0_btb_hit_l1_btb
                          && ip_pcload
                          && l0_btb_ipctrl_st_wait
                        || ip_chgflw_mistaken
                          && l0_btb_ipctrl_st_wait;
  ```

  遇到这两类情况时不包含PC的计算，可提前给出预测错误信号，不需等到addrgen中得出PC计算结果后进行更新。

- addrgen：专门针对跳转地址预测错误（包括返回地址预测和分支预测），涉及到预测PC（`branch_pred_result`）和正确PC（`branch_cal_result`）的计算。

  由于涉及到l0 btb预测错误的绝大多数情况在ibdp级会提前得到。因此addrgen主要用于处理l0 btb中表项缺失，而l1 btb预测错误的情况。

  ```
  assign branch_mispred = (branch_pred_result[PC_WIDTH-2:0] !=
  branch_cal_result[PC_WIDTH-2:0]);
  ```

  - 预测PC：低位和高位拼接，在ip级得到结果
    - 低20位来源于BTB预测结果
    - 高位来源于pcgen
  - 正确PC：基地址base+偏移量offset
    - 均来源于预译码，在ip级得到结果
    - 一个指令的偏移量是用21位表示的，而**不同的分支类型要求提取的偏移量位置不同**，因此在`decd_normal`模块分别提取了**五种分支类型**的偏移量信息。

- 若预测PC和正确PC不一致则发生addrgen模块的PC重定向，将指令译码后计算得到的有效地址发送到pcgen模块，刷掉前级流水线的有效信息

其他情况

除了表项缺失和预测错误之外，还有以下几种情况需要更新l0 btb

1. l0 btb命中，但bht结果为弱跳转。根据筛选的原则应将该表项筛出。
2. l0 btb命中，但表项的cnt位为0。L0 BTB触发重定向的条件需要同时满足命中L0 BTB中的表项以及表项对应的cnt位为1，故需要更新。
3. if chgflw valid,but ip doesn't chgflw

l0 btb更新机制

在l0_btb中，新的指令PC对应的预测信息（比如标志、跳转目标、ras位信息等）将传至write preparation模块，在下一个时钟周期写入需要更新的表项中。

```
casez({addrgen_l0_btb_update_vld,
      ibdp_l0_btb_update_vld})
2'b1? : begin
         l0_btb_wen[3:0]           = addrgen_l0_btb_wen[3:0];
         l0_btb_update_vld_bit     = addrgen_l0_btb_update_vld_bit;
         l0_btb_update_cnt_bit     = 1'b0;
         l0_btb_update_ras_bit     = 1'b0;
         l0_btb_update_data[36:0]  = 37'b0;
         end
2'b01 : begin
         l0_btb_wen[3:0]           = ibdp_l0_btb_wen[3:0];
         l0_btb_update_vld_bit     = ibdp_l0_btb_update_vld_bit;
         l0_btb_update_cnt_bit     = ibdp_l0_btb_update_cnt_bit;
         l0_btb_update_ras_bit     = ibdp_l0_btb_update_ras_bit;
         l0_btb_update_data[36:0]  = ibdp_l0_btb_update_data[36:0];
         end
default: begin
         l0_btb_wen[3:0]           = 4'b0;
         l0_btb_update_vld_bit     = 1'b0;
         l0_btb_update_cnt_bit     = 1'b0;
         l0_btb_update_ras_bit     = 1'b0;
         l0_btb_update_data[36:0]  = 37'b0;
         end
endcase
```

可以看到l0 btb的更新数据源来自于两个模块：ibdp和addrgen. 而addrgen更新的优先级更高。

由前文关于需要更新l0 btb的情况分析，addrgen只负责分支预测错误的情况的更新，需要进行PC的运算与比较。由以上源码可知，当addrgen判断预测错误时，会将表项无效（vld位置0），cnt位置零。由于是分支指令，ras位自然为0. 此外，表项中的数据也清零。

由于ibdp涉及到的更新类型较多，此处为其独立设置子目录进行总结。

ibdp

- 有关ras的表项更新（miss, mispred,mistaken)

  mistaken时vld位为0，使l0 btb表项无效化。而miss或mispred的时候vld仍为1，表项仍有效。仍然保持ras位为1。miss的情况和分支预测表项缺失大致相同，但更新的data有区别。高位相同，但icache开启两路路预测，且低20位全复位为0。

- 分支表项缺失

  表项仍然有效。若不为绝对跳转指令，则cnt位置为0。

- 不需计算地址的分支跳转预测错误

  将表项无效化，vld, cnt, ras位全部赋为0，表项中的数据也会清零。与addrgen中预测错误进行的更新处理相同。

- 其他需要更新l0 btb的情况

	vld	cnt
bht为弱跳转	0	0
命中表项但counter为0	1	1
情况三	0	0

```
case({l0_btb_ras_update,
      l0_btb_br_miss,
      l0_btb_br_mispred})

  //ras缺失/预测错误
  3'b100: begin
          l0_btb_wen[3:0]          = 4'b1111;
          l0_btb_update_vld_bit    = !l0_btb_ras_mistaken;
          l0_btb_update_cnt_bit    = 1'b1;
          l0_btb_update_ras_bit    = 1'b1;
          l0_btb_update_data[36:0] =
            {
             ibdp_vpc[14:0],                //entry_tag
             2'b11,                         //entry_way_pred
             20'b0                          //entry_target
            };
          end

  //分支预测表项缺失
  3'b010: begin
          l0_btb_wen[3:0]          = 4'b1111;
          l0_btb_update_vld_bit    = 1'b1;
          l0_btb_update_cnt_bit    = |ibdp_hn_jal[7:0];
          l0_btb_update_ras_bit    = 1'b0;
          l0_btb_update_data[36:0] =
            {ibdp_vpc[14:0],                //entry_tag
             ipdp_ibdp_branch_way_pred[1:0], //entry_way_pred
             ipdp_ibdp_branch_result[19:0]   //entry_target
            };
          end

  //分支预测错误，全部清零
```

```
  3'b001: begin
          l0_btb_wen[3:0]          = 4'b1000;
          l0_btb_update_vld_bit    = 1'b0; //会使entry_vld为0
          l0_btb_update_cnt_bit    = 1'b0;
          l0_btb_update_ras_bit    = 1'b0;
          l0_btb_update_data[36:0] = 37'b0;
          end

  default: begin //br_update
          l0_btb_wen[3:0]          = ipdp_ibdp_l0_btb_wen[3:0];
          l0_btb_update_vld_bit    = ipdp_ibdp_l0_btb_update_vld_bit;
          l0_btb_update_cnt_bit    = ipdp_ibdp_l0_btb_update_cnt_bit;
          l0_btb_update_ras_bit    = 1'b0;
          l0_btb_update_data[36:0] = 37'b0;
          end
```

关于相关信号的补充说明

entry_cnt

表项对应的cnt位主要用于判断命中的表项是否需要进行前端的重定向。**若该表项的cnt位为0，即使命中该L0 BTB表项，在IF级仍不会触发L0 BTB对前端进行重定向。**

```
//Only when Counter == 1,L0 BTB can be hit
assign entry_hit_counter       = entry_hit_flop[0]  & entry0_cnt
                               | entry_hit_flop[1]  & entry1_cnt
                               | entry_hit_flop[2]  & entry2_cnt
                               ...
```

对于某一条条件分支指令，L0 BTB发生miss时，将会清除L0 BTB的cnt位，由于L0 BTB触发重定向的条件需要同时满足命中L0 BTB中的表项以及表项对应的cnt位为1，在条件分支指令对应的表项刚刚更新时将不会触发对前端流水线的重定向。直到L0 BTB的命中信息传递到IP级，在该级发现指令包中存在预测强跳转的条件分支指令，且此时对应L0 BTB命中的表项中的cnt位为0，在该指令包到达IB级下一周期才会将cnt位置1。

之后若命中该条件分支指令对应的表项将会产生重定向信号。之后如果在IP级检测到该条件分支的预测跳转方向由强跳转变为了弱跳转，在该指令包到达IB级下一周期会将该条件分支指令在L0 BTB中对应的表项无效（清零vld位）。

entry_ras

L0 BTB中可以对绝对分支指令，强跳转的条件分支指令，以及return指令进行快速的跳转目标预测。RAS代表该L0 BTB表项对应储存的是否是一条return指令的返回地址，即该L0 BTB表项是RAS表项。

```
//l0_btb
assign entry_hit_ras           = entry_hit_flop[0]  & entry0_ras
                               | entry_hit_flop[1]  & entry1_ras
                               | ...

assign entry_hit_target[PC_WIDTH-2:0] = (entry_hit_ras)
                                      ? ras_pc[PC_WIDTH-2:0]
                                      : {pcgen_l0_btb_if_pc[PC_WIDTH-
2:20],entry_hit_pc[19:0]};
```

写使能信号wen[3:0]

wen[3]: entry_vld的赋值

```
  else if(entry_wen[3] && entry_update_en)
    entry_vld           <= entry_update_vld;
```

wen[2]: entry_cnt的赋值

```
  else if(entry_wen[2] && entry_update_en)
    entry_cnt           <= entry_update_cnt;
```

wen[1]: entry_ras的赋值

```
  else if(entry_wen[1] && entry_update_en)
    entry_ras           <= entry_update_ras;
```

wen[0]: entry_update_data的赋值--> tag, way_pred, target

```
  else if(entry_wen[0] && entry_update_en)
  begin
    entry_tag[14:0]     <= entry_update_data[36:22];
    entry_way_pred[1:0] <= entry_update_data[21:20];
    entry_target[19:0]  <= entry_update_data[19:0];
  end
```

更新机制

表项缺失

发生**未命中L0 BTB**(ras_miss和br_miss)时, 会判断该条件跳转指令或绝对跳转指令**是否存在于BTB内**,

1. 如果存在则发生L0 BTB缺失,
2. 若不存在会首先更新BTB,

   下一次该指令被取出时发生BTB命中而L0 BTB未命中时，才判断L0 BTB缺失，更新L0 BTB。

```
assign l0_btb_br_miss    = l0_btb_br_miss_pre
                        && !ibdp_btb_miss
                        && !l0_btb_ras_update;
```

发生表项缺失时，L0 BTB采用**fifo**的原则更新，决定了下一个时钟周期`entry_fifo[15:0]`是否移位，其信号源来自IB级。

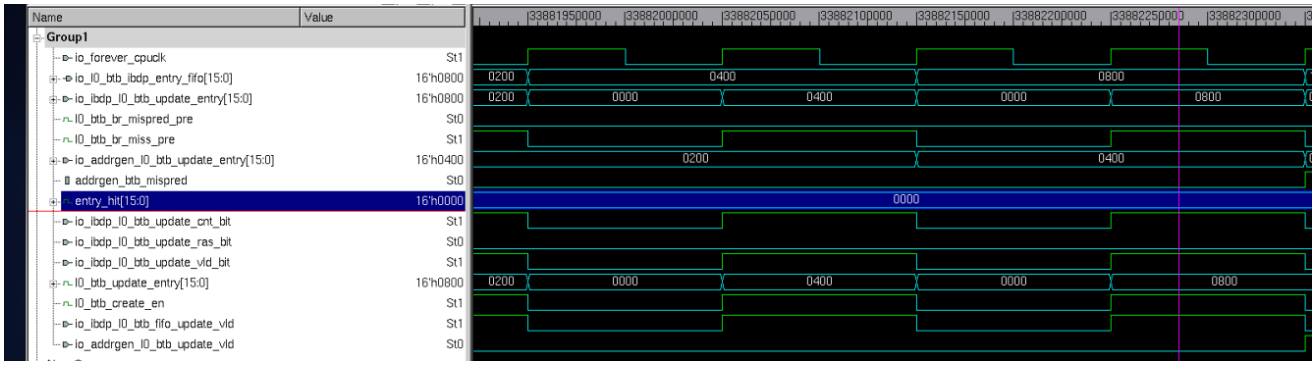
```
//ibdp
assign ibdp_l0_btb_fifo_update_vld   = (l0_btb_ras_miss || l0_btb_br_miss)
                                    && ib_data_vld
                                    && !ipctrl_ibdp_expt_vld
                                    && !ibctrl_ibdp_self_stall;

//l0 btb - fifo
assign l0_btb_create_en = ibdp_l0_btb_fifo_update_vld
```

```
                        && cp0_ifu_btb_en
                        && cp0_ifu_l0btb_en;

always @(posedge l0_btb_create_clk or negedge cpurst_b)
begin
  if(!cpurst_b)
    entry_fifo[15:0] <= 16'b1;
  else if(l0_btb_create_en)
    entry_fifo[15:0] <= {entry_fifo[14:0],entry_fifo[15]};
  else
    entry_fifo[15:0] <= entry_fifo[15:0];
end
```

fifo模块中的`entry_fifo[15:0]`的每一位分别为对应的l0_btb表项的更新使能信号，即最终决定l0 btb将更新哪个表项。从源码中可知复位时`entry_fifo`的值为1，对应更新第0号表项。发生表项缺失时，每次更新时左移一位，更新完第15号表项后下一次更新第0号表项，如此循环。最终代码运行的效果为：发生表项缺失时，l0_btb按顺序更新表项数据。



在波形图中，可以看到entry_fifo为0x0200, 0x0400, 0x0800时都发生表项缺失，此时来自于ibdp表项缺失信号与需要更新的表项相关数据在延迟一拍后得出。

```
//ibdp
//括号中的逻辑是对表项缺失情况进行的判断。对于后一项，由于ras_update信号为ras_miss, ras_mispred,
ras_mistaken相或。为了排除后两种情况而只考虑分支缺失，ras_update信号须为0并另外和br_miss相与。
assign l0_btb_hit_entry[15:0] = (l0_btb_ras_miss || l0_btb_br_miss_pre &&
!l0_btb_ras_update)
                              ? l0_btb_ibdp_entry_fifo[15:0]
                              : ipdp_ibdp_l0_btb_entry_hit[15:0];

//output to addrgen
assign ibdp_addrgen_l0_btb_hit_entry[15:0] = (ibctrl_ibdp_l0_btb_hit)
                                           ? ipdp_ibdp_l0_btb_entry_hit[15:0]
                                           : l0_btb_ibdp_entry_fifo[15:0];

assign l0_btb_update_vld_for_gateclk = (l0_btb_br_miss_pre
                                     || l0_btb_br_mispred_pre
                                     || l0_btb_br_update_pre
                                     || l0_btb_ras_mistaken
                                     || l0_btb_ras_miss
                                     || l0_btb_ras_mispred)
                                   && ib_data_vld;

//addrgen
else if(branch_vld)
addrgen_l0_btb_hit_entry[15:0]          <= l0_btb_hit_entry[15:0];
```

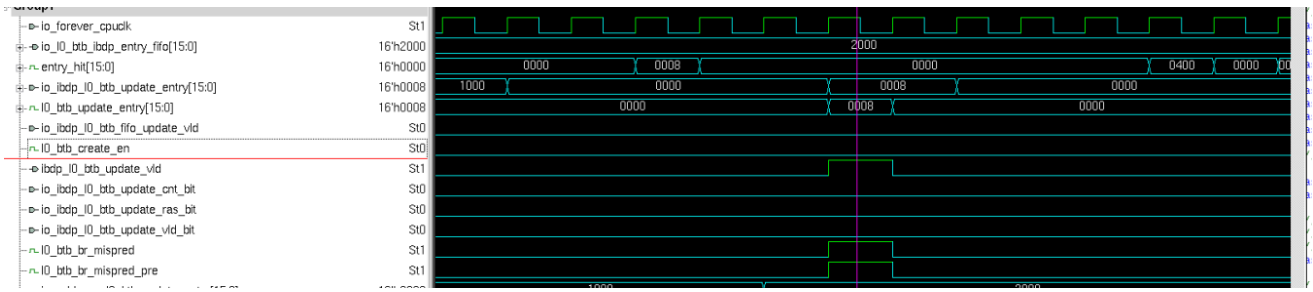
```
assign addrgen_l0_btb_update_entry[15:0] = addrgen_l0_btb_hit_entry[15:0];
```

```
//l0 btb - write preparation
assign l0_btb_update_entry[15:0] =
   {16{addrgen_l0_btb_update_vld}}          & addrgen_l0_btb_update_entry[15:0]
 | {16{l0_btb_update_vld_for_gateclk}}      & ibdp_l0_btb_update_entry[15:0];


//此处以entry0为例，其它15个表项类似。
ct_ifu_l0_btb_entry  x_l0_btb_entry_0 (
    ...
    .entry_update             (l0_btb_update_entry[0]  ),
    ...
);
```

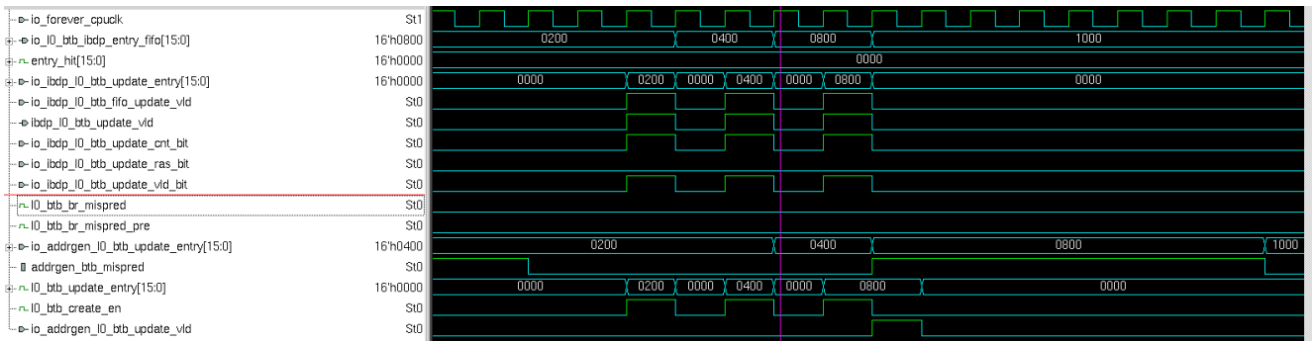
预测错误

由前文对于ibdp和addrgen的entry-fifo表项的分析，ibdp中返回地址预测错误（`ras_mispred` & `ras_mistaken`）和分支预测错误（`br_mispred`）时对应的更新表项为`ipdp_ibdp_l0_btb_entry_hit`，为l0 btb中的命中表项`entry_hit[15:0]`在ifdp和ipdp延迟两拍后得到。



而addrgen判断发生跳转目标预测错误时，由于此时判断的是l1 btb的预测跳转目标（比l0 btb的结果晚一拍得出），故对应l0 btb中需要更新的表项应比ibdp的entry_fifo延迟一拍，即相较于l0 btb中的`entry_fifo`延迟两个周期。

（从波形图中可以看到在`entry_fifo`为0x0800时，l0 btb发生表项缺失，l1 btb发生预测错误）



预测错误时，不会进行fifo移位操作，`entry_fifo`保持不变。直到对应表项更新完成，将从预测错误前的`entry_fifo`表项重新开始，进行后续的表项更新操作。