Project 4: Problem 2

Control hazard에 의한 stall cycle 수 및 CPI 증가값

먼저 프로그램의 총 CPI는 다음과 같이 계산된다.

$$CPI = \frac{Clock\ cycles}{Instruction\ Count} = \frac{7681}{5176} = 1.48396$$

Control hazard에 의한 stall cycle의 계산 logic은 다음과 같다.

1. Branch Instruction이 TAKEN인 경우, 올바른 next PC 값(Branch Target)을 계산해야 한다.
2. 이 계산은 Execution stage에서 이루어진다.
3. 따라서 Branch가 TAKEN 되는 경우, prediction을 하지 않는다면 IF, ID stage에서의 2 cycle stall이 발생한다.
4. 즉, TAKEN된 Branch Instruction 수를 count하여 각 instruction 마다 2 cycle stall을 발생시키므로 2를 곱해주면 control harzard에 의한 stall cycle을 계산할 수 있다.

이러한 logic에 따라 다음과 같이 구현하였다.

```
always @(posedge rst_i or posedge clk) begin
    if     (rst_i)                          HPC_req_TAKEN  <='b0;
    else if(req_branch_taken)         HPC_req_TAKEN  <= HPC_req_TAKEN + 1'b1;
end
```

이때 위의 logic이 valid한지 검증하기 위해서는 두 가지 확인이 필요하다.

첫 번째로 `req_branch_taken`이 정말로 branch의 taken or not taken 결과를 represent하고 있는 변수인지 확인해야한다. 이는 `riscv_pipe_ctrl.v`의 코드를 통해 검증할 수 있었다.

```
268     ,.issue_branch_taken_i(branch_d_exec_request_i)
```

```
512   assign HPC_req_branch_taken    = issue_branch_taken_i;
```

두 번째로 정말로 TAKEN된 branch intruction에 의해 실제로 2 stall cylce stall이 발생함을 검증할 필요가 있다.

- 실제로 다음의 `squash_decode_o`를 중심으로 살펴보면, execution stage에서 squash가 발생함을 알 수 있다.

```
137   assign squash_decode_o     = branch_request_i;
```

```
97  generate
98  if (EXTRA_DECODE_STAGE)
99  begin
100     wire [31:0] fetch_in_instr_w = (fetch_in_fault_page_i | fetch_in_fault_fetch_i) ? 32'b0 : fetch_in_instr_i;
101     reg [66:0]  buffer_q;
102
103     always @(posedge clk_i or posedge rst_i)
104     if (rst_i)
105         buffer_q <= 67'b0;
106     else if (squash_decode_i)
107         buffer_q <= 67'b0;
108     else if (fetch_out_accept_i || !fetch_out_valid_o)
109         buffer_q <= {fetch_in_valid_i, fetch_in_fault_page_i, fetch_in_fault_fetch_i, fetch_in_instr_w, fetch_in_pc_i};
110
111     assign {fetch_out_valid_o,
112             fetch_out_fault_page_o,
113             fetch_out_fault_fetch_o,
114             fetch_out_instr_o,
115             fetch_out_pc_o} = buffer_q;
```

```
671   reg [31:0] cnt_squash;
672   always @(posedge clk_i or posedge rst_i) begin
673       if(rst_i)                  cnt_squash <='b0;
674       else if (squash_decode_w)  cnt_squash <=cnt_squash +1'b1;
675   end
```

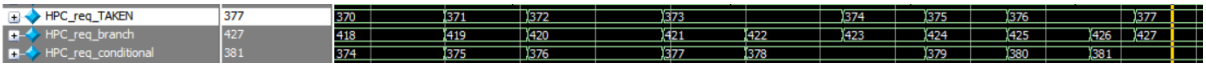
```
298     // Execution stage 2
299     ,.load_e2_o(pipe_load_e2_w)
300     ,.mul_e2_o(pipe_mul_e2_w)
301     ,.rd_e2_o(pipe_rd_e2_w)
302     ,.result_e2_o(pipe_result_e2_w)
303
304     ,.stall_o(pipe_stall_raw_w)
305     ,.squash_e1_e2_o(pipe_squash_e1_e2_w)
306     ,.squash_e1_e2_i(1'b0)
307     ,.squash_wb_i(1'b0)
```

```
186 else if ((issue_valid_i && issue_accept_i) && ~(squash_e1_e2_o || squash_e1_e2_i))
187 begin
188     valid_e1_q                <= 1'b1;
189     ctrl_e1_q[`PCINFO_ALU]    <= ~(issue_lsu_i | issue_csr_i | issue_div_i | issue_mul_i);
190     ctrl_e1_q[`PCINFO_LOAD]   <= issue_lsu_i &  issue_rd_valid_i & ~take_interrupt_i; // TODO: Check
191     ctrl_e1_q[`PCINFO_STORE]  <= issue_lsu_i & ~issue_rd_valid_i & ~take_interrupt_i;
192     ctrl_e1_q[`PCINFO_CSR]    <= issue_csr_i & ~take_interrupt_i;
193     ctrl_e1_q[`PCINFO_DIV]    <= issue_div_i & ~take_interrupt_i;
194     ctrl_e1_q[`PCINFO_MUL]    <= issue_mul_i & ~take_interrupt_i;
195     ctrl_e1_q[`PCINFO_BRANCH] <= issue_branch_i & ~take_interrupt_i;
196     ctrl_e1_q[`PCINFO_RD_VALID] <= issue_rd_valid_i & ~take_interrupt_i;
197     ctrl_e1_q[`PCINFO_INTR]   <= take_interrupt_i;
198     ctrl_e1_q[`PCINFO_COMPLETE] <= 1'b1;
199
200     pc_e1_q         <= issue_pc_i;
201     npc_e1_q        <= issue_branch_taken_i ? issue_branch_target_i : issue_pc_i + 32'd4;
202     opcode_e1_q     <= issue_opcode_i;
203     operand_ra_e1_q <= issue_operand_ra_i;
204     operand_rb_e1_q <= issue_operand_rb_i;
205     exception_e1_q  <= (|issue_exception_i) ? issue_exception_i :
206                        branch_misaligned_w  ? `EXCEPTION_MISALIGNED_FETCH : `EXCEPTION_W'b0;
207 end
```

```
267   // Pipeline flush
268   else if (squash_e1_e2_o || squash_e1_e2_i)
269   begin
270       valid_e2_q      <= 1'b0;
271       ctrl_e2_q       <= `PCINFO_W'b0;
272       csr_wr_e2_q     <= 1'b0;
273       csr_wdata_e2_q  <= 32'b0;
274       pc_e2_q         <= 32'b0;
275       npc_e2_q        <= 32'b0;
276       opcode_e2_q     <= 32'b0;
277       operand_ra_e2_q <= 32'b0;
278       operand_rb_e2_q <= 32'b0;
279       result_e2_q     <= 32'b0;
280       exception_e2_q  <= `EXCEPTION_W'b0;
281   end
```

- Execution stage에서 발생하므로, IF, ID stage 총 2cycle bubble이 발생할 것을 코드를 통해서 직접 확인할 수 있었다.

따라서 위와 같은 logic을 통해 TAKEN된 Branch Instruction 수를 count했고, 그 결과는 총 377개로 확인되었다. 즉 그에 따른 총 stall cycle은 377 x 2 = 754가 된다. 결과에 대한 파형은 다음과 같이 확인할 수 있다.



이에 의한 CPI 증가값을 계산했고, 그 결과는 다음과 같다.

- 총 CPI : 1.4840
- 해당 instruction에 의한 stall이 발생하지 않은 이상적인 경우의 CPI : $\frac{7681-754}{5176} = \frac{6927}{5176} = 1.3383$
- CPI 증가값 : 1.4840 - 1.3383 = 0.1457

Branch History Table

다음과 같은 과정을 거쳐 branch history table을 구현했다.

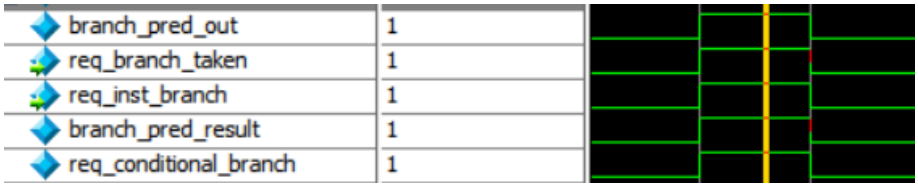
1. 주어진 input, register, wire 변수의 의미 파악.
    - `req_inst_branch`는 현재 instruction의 branch instruction 여부를 나타내며, `req_branch_taken`은 branch instruction의 결과가 TAKEN인지 NOT TAKEN인지를 나타낸다.
    - `req_conditional_branch`의 경우 conditional branch 여부를 나타낸다. `req_inst_branch`는 conditional branch 뿐만 아니라, unconditional branch 또한 포함하고 있다. BHT에서는 conditional branch에 대한 prediction을 수행하므로 `req_branch`에서 conditional branch만 별도로 분리한 변수이다.

```
268   assign req_conditional_branch = (req_inst_branch && req_inst_opcode[6:0] == 7'b110_0011);
269   always @(posedge rst_i or posedge clk) begin
270       if(rst_i)                 req_conditional_branch_r <= 1'b0;
271       else                      req_conditional_branch_r <= req_conditional_branch;
272   end
273
```

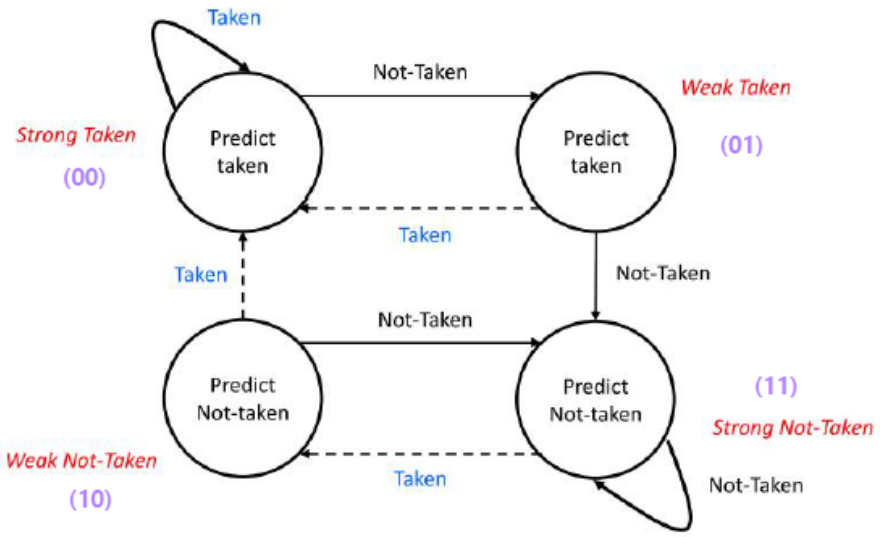
    - `branch_pred_out`과 `branch_pred_result`는 각각 BHT의 prediction 출력과 해당 출력과 실제 taken / not taken 값의 비교를 통해 prediction이 맞았는지 틀렸는지를 나타내는 변수이다.

```
319   assign branch_pred_out = req_conditional_branch && (!HPC_branch_history_table[req_branch_inst_pc[`FETCH_PC_BHT_INDEX]][1]);
320   assign branch_pred_result = req_conditional_branch && (branch_pred_out == req_branch_taken);
```

- 실제로 각각의 값이 나타내는 의미를 파형을 통해 다시 한번 검증하였다. 다음과 같이 예상되는 출력을 보여주고 있는 파형을 확인할 수 있다.



2. 이후 prediction 결과에 따라 BHT 값을 변경해주었다. 이때 주의해야할 점은 현재의 prediction 결과를 바로 BHT에 반영하면, 현재의 `branch_pred_result` 값이 변경되어 결과가 달라지게 되므로, 실제 BHT 값을 변경하는 것은 다음 clock에 수행되어야 한다는 점이다.
   - 따라서 다음과 같이 `req_conditional_branch`, `req_branch_inst_pc[`FETCH_PC_BHT_INDEX]`, `branch_pred_result`를 각각 `req_conditional_branch_r`, `req_branch_history_index`, `branch_pred_result_r`을 사용해 latching하였다.
   - 그리고 해당 latch 값에 따라 BHT 값을 변경하였다. 즉 한 clock 늦게 값을 변경했다. 다음의 조건에 따라 값을 변경했다.



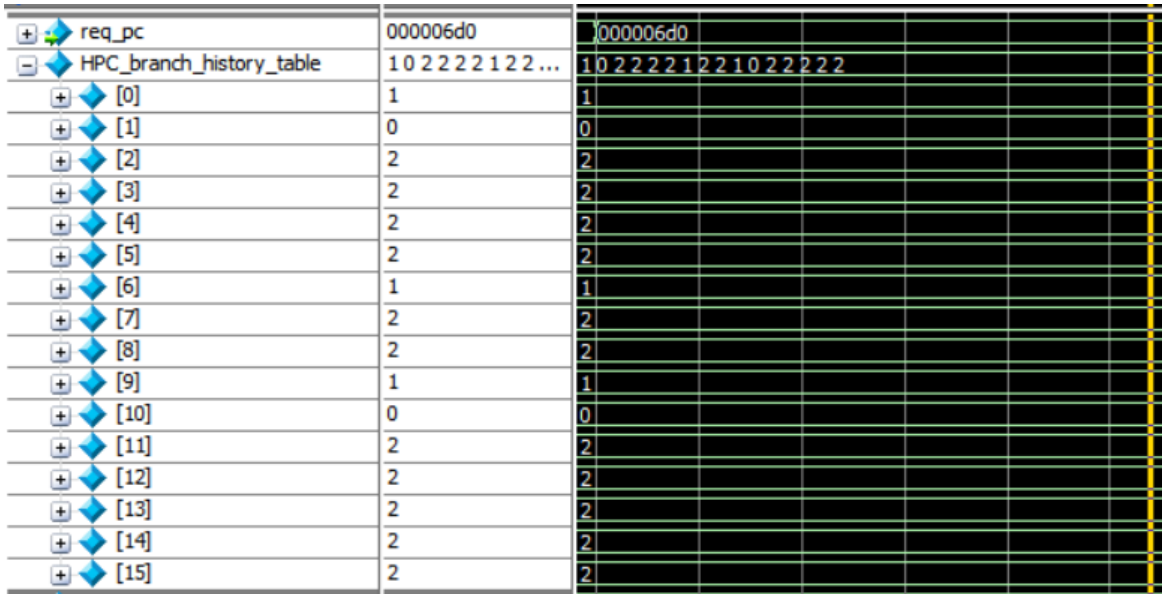
- 해당 코드는 다음과 같다.

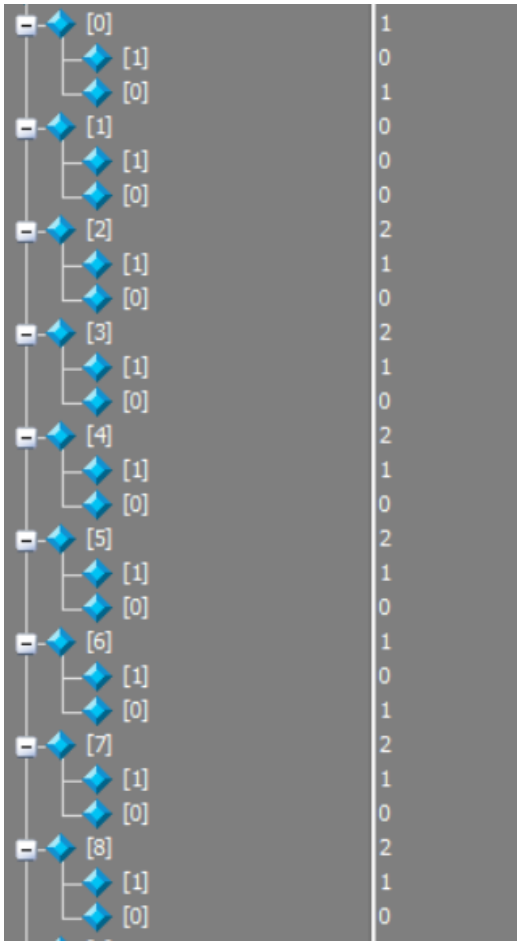
```
assign req_conditional_branch = (req_inst_branch && req_inst_opcode[6:0] == 7'b110_0011);
always @(posedge rst_i or posedge clk) begin
    if(rst_i)                   req_conditional_branch_r <= 1'b0;
    else                        req_conditional_branch_r <= req_conditional_branch;
end

always @(posedge rst_i or posedge clk) begin
    if(rst_i)                   req_branch_history_index <= 'b0;
    else                        req_branch_history_index <= req_branch_inst_pc[`FETCH_PC_BHT_INDEX];
end

always @(posedge rst_i or posedge clk) begin
```

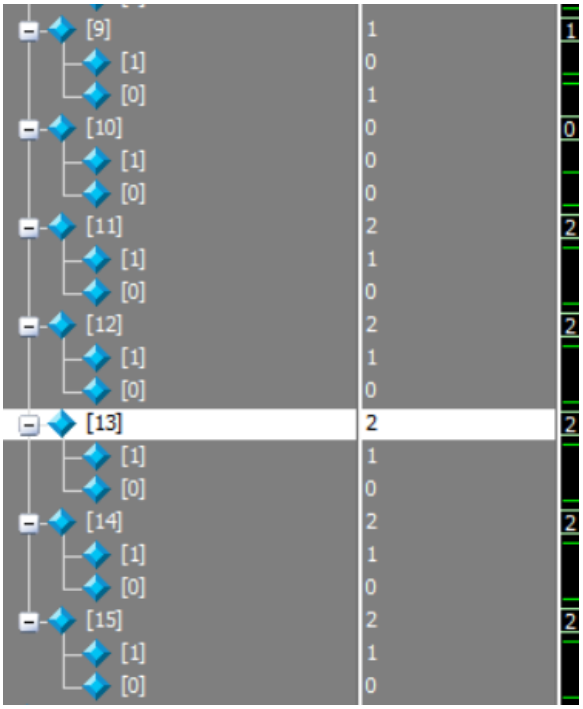
```
    if(rst_i)                   branch_pred_result_r <= 'b0;
    else                        branch_pred_result_r <= branch_pred_result;
end

always @(posedge rst_i or posedge clk) begin
    if (rst_i)
    begin
        HPC_branch_history_table[0] = STATE_WEAK_NOT_TAKEN;
        HPC_branch_history_table[1] = STATE_WEAK_NOT_TAKEN;
        HPC_branch_history_table[2] = STATE_WEAK_NOT_TAKEN;
        HPC_branch_history_table[3] = STATE_WEAK_NOT_TAKEN;
        HPC_branch_history_table[4] = STATE_WEAK_NOT_TAKEN;
        HPC_branch_history_table[5] = STATE_WEAK_NOT_TAKEN;
        HPC_branch_history_table[6] = STATE_WEAK_NOT_TAKEN;
        HPC_branch_history_table[7] = STATE_WEAK_NOT_TAKEN;
        HPC_branch_history_table[8] = STATE_WEAK_NOT_TAKEN;
        HPC_branch_history_table[9] = STATE_WEAK_NOT_TAKEN;
        HPC_branch_history_table[10] = STATE_WEAK_NOT_TAKEN;
        HPC_branch_history_table[11] = STATE_WEAK_NOT_TAKEN;
        HPC_branch_history_table[12] = STATE_WEAK_NOT_TAKEN;
        HPC_branch_history_table[13] = STATE_WEAK_NOT_TAKEN;
        HPC_branch_history_table[14] = STATE_WEAK_NOT_TAKEN;
        HPC_branch_history_table[15] = STATE_WEAK_NOT_TAKEN;
    end
    else if (req_conditional_branch_r)
    begin
        if (branch_pred_result_r)
        begin
            if (HPC_branch_history_table[req_branch_history_index] == STATE_WEAK_NOT_TAKEN)
                HPC_branch_history_table[req_branch_history_index] = STATE_STRONG_NOT_TAKEN;
            else if (HPC_branch_history_table[req_branch_history_index] == STATE_WEAK_TAKEN)
                HPC_branch_history_table[req_branch_history_index] = STATE_STRONG_TAKEN;
        end
        else
        begin
            if (HPC_branch_history_table[req_branch_history_index] == STATE_WEAK_NOT_TAKEN)
                HPC_branch_history_table[req_branch_history_index] = STATE_STRONG_TAKEN;
            else if (HPC_branch_history_table[req_branch_history_index] == STATE_WEAK_TAKEN)
                HPC_branch_history_table[req_branch_history_index] = STATE_STRONG_NOT_TAKEN;
            else if (HPC_branch_history_table[req_branch_history_index] == STATE_STRONG_TAKEN)
                HPC_branch_history_table[req_branch_history_index] = STATE_WEAK_TAKEN;
            else if (HPC_branch_history_table[req_branch_history_index] == STATE_STRONG_NOT_TAKEN)
                HPC_branch_history_table[req_branch_history_index] = STATE_WEAK_NOT_TAKEN;
        end
    end
end
```

위의 과정을 통해 BHT를 구현하였고, 그 결과는 다음과 같았다.

- Fetch PC = 0×0000_06D0 일 때, BHT 의 entry 16 개 모두 보이도록 캡처한 파형 (problem2_2.jpg)







- `branch_prediction_trace.txt` file 을 이용하여 branch prediction hit rate 계산
  - 먼저 hit rate를 계산하기 위해 총 branch instruction 중 conditional branch instruction 수를 count 했다. 해당 코드는 다음와 같다.

```
always @(posedge rst_i or posedge clk) begin
    if     (rst_i)                          HPC_req_conditional  <='b0;
    else if(req_conditional_branch)                 HPC_req_conditional  <= HPC_req_conditional + 1'b1;
end
```

- 그 결과 총 381개의 conditional instruction이 존재함을 확인했다.

- `branch_prediction_trace.txt` 파일에서 branch prediction이 correct한 경우를 count했을 때 다음과 같이 총 325개의 instruction에 대해 correct함을 확인할 수 있었다.

branch_prediction_hit_count - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

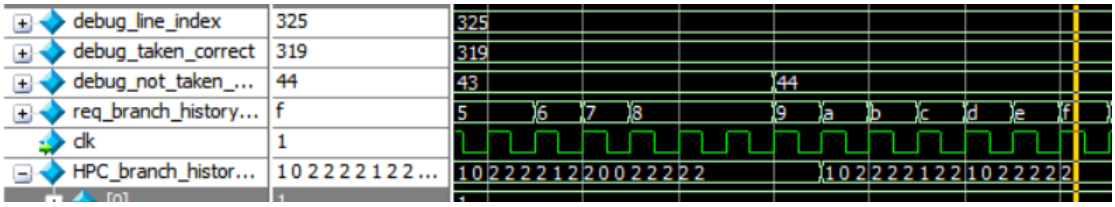
```
308: PC=000001a4,     debug_branch_taken_w=1, debug_branch_pred_out=1
309: PC=00000168,     debug_branch_taken_w=1, debug_branch_pred_out=1
310: PC=000001a4,     debug_branch_taken_w=1, debug_branch_pred_out=1
311: PC=00000168,     debug_branch_taken_w=1, debug_branch_pred_out=1
312: PC=000001a4,     debug_branch_taken_w=1, debug_branch_pred_out=1
313: PC=00000168,     debug_branch_taken_w=1, debug_branch_pred_out=1
314: PC=000001a4,     debug_branch_taken_w=1, debug_branch_pred_out=1
315: PC=00000168,     debug_branch_taken_w=1, debug_branch_pred_out=1
316: PC=000001a4,     debug_branch_taken_w=1, debug_branch_pred_out=1
317: PC=00000168,     debug_branch_taken_w=1, debug_branch_pred_out=1
318: PC=000001a4,     debug_branch_taken_w=1, debug_branch_pred_out=1
318: PC=00000168,     debug_branch_taken_w=0, debug_branch_pred_out=1
319: PC=00000184,     debug_branch_taken_w=1, debug_branch_pred_out=1
320: PC=000001a4,     debug_branch_taken_w=1, debug_branch_pred_out=1
321: PC=00000168,     debug_branch_taken_w=1, debug_branch_pred_out=1
322: PC=000001a4,     debug_branch_taken_w=1, debug_branch_pred_out=1
323: PC=00000168,     debug_branch_taken_w=1, debug_branch_pred_out=1
323: PC=000001a4,     debug_branch_taken_w=0, debug_branch_pred_out=1
324: PC=000000e4,     debug_branch_taken_w=1, debug_branch_pred_out=1
325: PC=000000c4,     debug_branch_taken_w=1, debug_branch_pred_out=1
325: PC=000000e4,     debug_branch_taken_w=0, debug_branch_pred_out=1
```

- 또한 동일한 logic을 다른 변수를 사용하여 구현한 BHT에 대해서도 동일한 결과를 출력함을 확인하여 결과가 valid함을 재차 확인할 수 있었다. 해당 코드는 다음과 같다.

```
always @(posedge rst_i or posedge clk) begin
    if(rst_i)                   branch_taken_r <= 'b0;
    else                        branch_taken_r <= req_branch_taken;
end

always @(posedge rst_i or posedge clk) begin
    if (rst_i)
    begin
        HPC_branch_history_table[0] = STATE_WEAK_NOT_TAKEN;
        HPC_branch_history_table[1] = STATE_WEAK_NOT_TAKEN;
        HPC_branch_history_table[2] = STATE_WEAK_NOT_TAKEN;
```

```
            HPC_branch_history_table[3] = STATE_WEAK_NOT_TAKEN;
            HPC_branch_history_table[4] = STATE_WEAK_NOT_TAKEN;
            HPC_branch_history_table[5] = STATE_WEAK_NOT_TAKEN;
            HPC_branch_history_table[6] = STATE_WEAK_NOT_TAKEN;
            HPC_branch_history_table[7] = STATE_WEAK_NOT_TAKEN;
            HPC_branch_history_table[8] = STATE_WEAK_NOT_TAKEN;
            HPC_branch_history_table[9] = STATE_WEAK_NOT_TAKEN;
            HPC_branch_history_table[10] = STATE_WEAK_NOT_TAKEN;
            HPC_branch_history_table[11] = STATE_WEAK_NOT_TAKEN;
            HPC_branch_history_table[12] = STATE_WEAK_NOT_TAKEN;
            HPC_branch_history_table[13] = STATE_WEAK_NOT_TAKEN;
            HPC_branch_history_table[14] = STATE_WEAK_NOT_TAKEN;
            HPC_branch_history_table[15] = STATE_WEAK_NOT_TAKEN;
        end
        else if (req_conditional_branch_r)
        begin
            if (branch_taken_r)
            begin
                if (HPC_branch_history_table[req_branch_history_index] == STATE_WEAK_NOT_TAKEN)
                    HPC_branch_history_table[req_branch_history_index] = STATE_STRONG_TAKEN;
                else if (HPC_branch_history_table[req_branch_history_index] == STATE_WEAK_TAKEN)
                    HPC_branch_history_table[req_branch_history_index] = STATE_STRONG_TAKEN;
                else if (HPC_branch_history_table[req_branch_history_index] == STATE_STRONG_TAKEN)
                    HPC_branch_history_table[req_branch_history_index] = STATE_STRONG_TAKEN;
                else if (HPC_branch_history_table[req_branch_history_index] == STATE_STRONG_NOT_TAKEN)
                    HPC_branch_history_table[req_branch_history_index] = STATE_WEAK_NOT_TAKEN;
            end
            else
            begin
                if (HPC_branch_history_table[req_branch_history_index] == STATE_WEAK_NOT_TAKEN)
                    HPC_branch_history_table[req_branch_history_index] = STATE_STRONG_NOT_TAKEN;
                else if (HPC_branch_history_table[req_branch_history_index] == STATE_WEAK_TAKEN)
                    HPC_branch_history_table[req_branch_history_index] = STATE_STRONG_NOT_TAKEN;
                else if (HPC_branch_history_table[req_branch_history_index] == STATE_STRONG_TAKEN)
                    HPC_branch_history_table[req_branch_history_index] = STATE_WEAK_TAKEN;
                else if (HPC_branch_history_table[req_branch_history_index] == STATE_STRONG_NOT_TAKEN)
                    HPC_branch_history_table[req_branch_history_index] = STATE_STRONG_NOT_TAKEN;
            end
        end
    end
end
```

- 추가적으로 파형을 직접확인하며, stall이 발생한 경우 한 cycle의 간격을 두고 `req_branch_history_index` 의 값이 적절하게 변화함을 확인하였다.



- 따라서 hit rate는 다음과 같이 계산된다.

$$(hit\ rate) = \frac{prediction\_correct\_count}{conditional\_branch\_instruction\_count} = \frac{325}{381} \approx 0.85301$$

- 상당히 높은 hit rate를 가짐을 확인할 수 있었는데, 이는 사실 testbench에서 실행하는 프로그램이 DFS 알고리즘을 수행하는 프로그램임을 고려하면 충분히 추론가능한 사실이다.
- DFS 알고리즘을 기반으로 그래프를 탐색, 즉 프로그램 특성 상 노드를 방문하는 동작을 반복적으로 수행하기 때문에 history를 통한 naive한 예측으로도 상당히 높은 hit rate를 얻을 수 있을 것이라고 추론할 수 있다.

    - 실제 시뮬레이션 결과를 통해서도 그러한 추론과 부합하는 결과를 얻을 수 있었다.
- 설계한 branch predictor 를 pipeline 실행에 적용한다고 가정하였을 때 향상될 CPI 의 값
    - Branch predictor를 적용하는 경우, 다음의 경우에 성능의 변화가 있을 것이다.
        1. TAKEN Conditional Branch Instruction + Prediction Correct → 2 stall cycle 감소
        2. NOT TAKEN Conditional Branch Instruction + Prediction Wrong → 2 stall cycle 증가
    - 이에 따라 각각의 경우를 카운트하여, 변화된 stall cycle은 다음과 같다.
        - 1번 경우 : 총 319 번 x 2 = 638 stall cycle 감소
        - 2번 경우 : 총 44 번 x 2 = 88 stall cycle 증가
        - 전체 stall cycle 변화 : 754 - 638 + 88 = 204 stall cycle
    - 따라서 설계한 branch predictor를 pipeline 실행에 적용한다고 가정하였을 때, CPI는 다음과 같이 향상될 것이다.
        - 적용 이전의 CPI : 1.4840
        - 적용 시, 향상될 CPI : $\frac{7681-638+88}{5176} = \frac{7131}{5176} = 1.3778$