2024 Spring CSED311 Lab 4-2 Report

Team ID: 67735
20220312 박준혁, 20220871 홍지우

명예서약 (Honor Code)

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

I completed this programming task without the improper help of others.

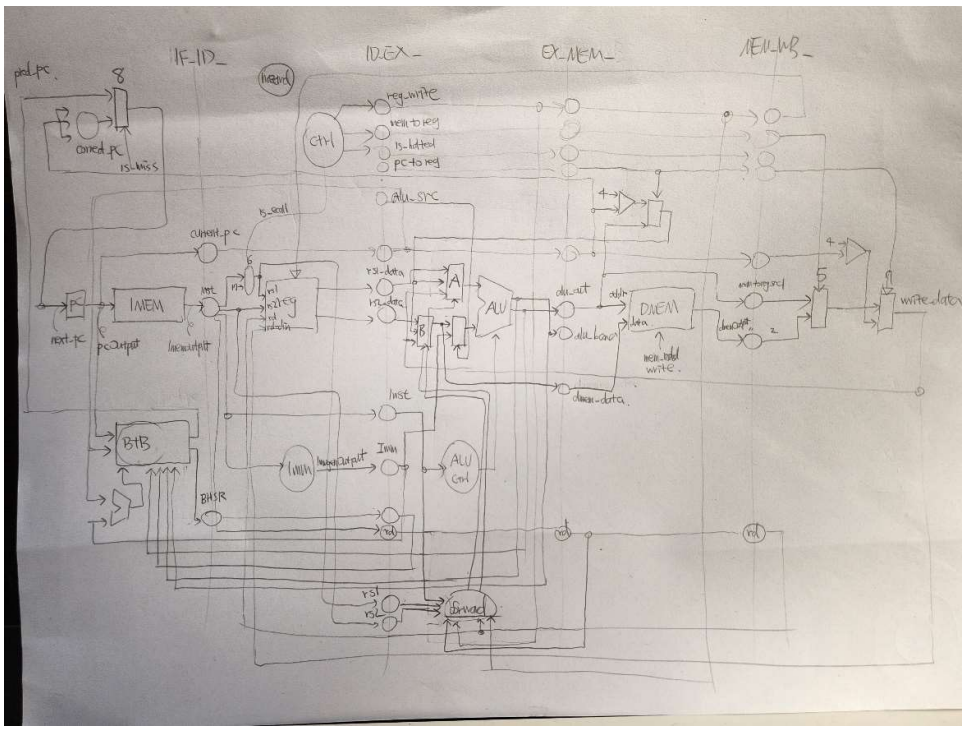
1 Introduction

1.1 Lab 4-1의 Pipelined CPU에 Control Flow Instruction이 전부 돌아가도록 제작한다. Branch Prediction이 추가되었다.

2 Design

2.1 설계 내용.

전체 모습은 다음과 같다.



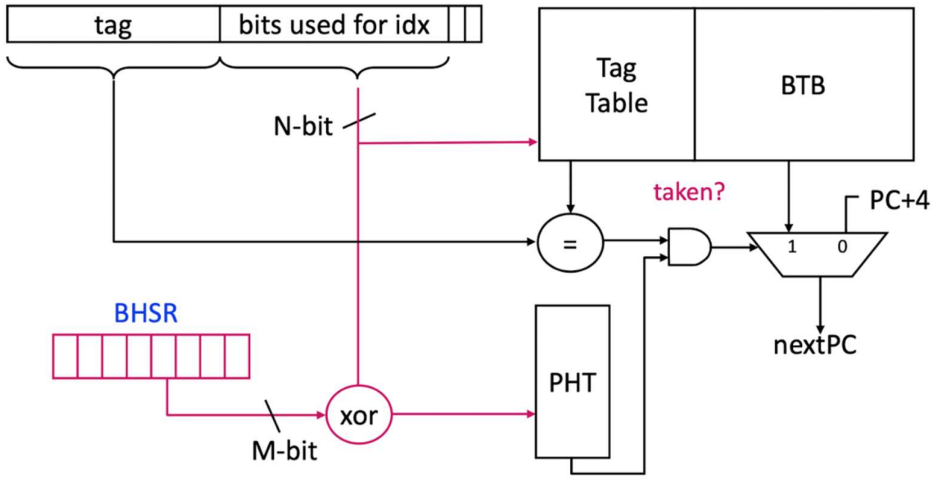
그림에는 없으나

hazard detection, ecall forwarding unit, prediction 확인하는 회로까지 있다.

BTB는 Gshare 기반으로 구현되어 있다.

2.2 Gshare Branch Predictor

아래와 같이 작동한다.



3 Implementation

3.1 Branch Predictor Implementation

3.1.1 wire, regs

```
// tag table + btb + pht
integer idx;
reg [31:0] tag_table [0:31];
reg [31:0] btb       [0:31]; // 32 entry btb
reg [1:0]  pht       [0:31]; // 2-bit prediction

// input query
wire [31:0] query_tag;
wire [4:0]  query_idx;
assign query_tag = pc[31:0];
assign query_idx = pc[6:2] ^ BHSR;
// real pc
wire [4:0]  real_pc_idx;
wire [31:0] real_pc_tag;
assign real_pc_tag = real_pc[31:0];
assign real_pc_idx = real_pc[6:2] ^ real_pc_BHSR;

wire taken;
assign taken = (branch & alu_bcond) | is_jal | is_jalr;

reg [4:0] BHSR_tmp;
assign BHSR = (BHSR_tmp << 1) + {4'b0, taken};

reg [31:0] dest; // temporary wire for convenience
```

BHSR의 정의를 보면

한칸씩 shift하고, 빈 자리에 새로운 taken 값을 순서대로 넣는 식이다.

3.1.2 Initialization

```
if (reset) begin
    for (idx = 0; idx <= 31; idx = idx + 1) begin
        btb[idx] <= 0; // empty btb
        tag_table[idx] <= -1; // invalid tag
        pht[idx] <= 2'b00;
        BHSR_tmp<=0;
    end
end
```

3.1.3 Setting Prediction values, updating tables

instruction에 따라 테이블에 넣을 값을 바꾸어 주고

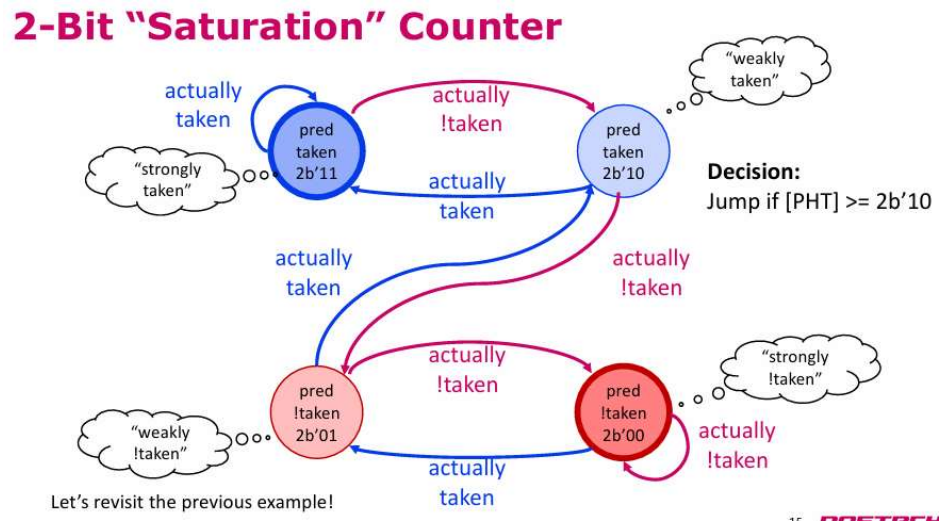
real_pc (branch 계산 후 얻은 실제 주소) 값을 이용해 테이블을 업데이트한다.

```
else begin
    if (is_jal | branch) begin // destination = pc + imm
        dest <= pc_plus_imm;
    end else if (is_jalr) begin // destination = reg + imm
        dest <= reg_plus_imm;
    end

    if ((is_jal | branch| is_jalr ) &&
        (real_pc_tag != tag_table[real_pc_idx] | dest != btb[real_pc_idx])) begin
        tag_table[real_pc_idx] <= real_pc_tag;
        btb[real_pc_idx] <= dest;
    end
```

3.1.4 2-bit Counter

```
// 3. pht : 2-bit prediction
if (branch | is_jal | is_jalr) begin
    if (taken) begin
        case (pht[real_pc_idx])
            2'b00: pht[real_pc_idx] <= 2'b01;
            2'b01: pht[real_pc_idx] <= 2'b10;
            2'b10: pht[real_pc_idx] <= 2'b11;
            2'b11: pht[real_pc_idx] <= 2'b11;
        endcase
    end else begin // not taken
        case (pht[real_pc_idx])
            2'b00: pht[real_pc_idx] <= 2'b00;
            2'b01: pht[real_pc_idx] <= 2'b00;
            2'b10: pht[real_pc_idx] <= 2'b01;
            2'b11: pht[real_pc_idx] <= 2'b10;
        endcase
    end
end
```



saturation counter이다.

3.1.5 Setting Output Value: pred_pc

bxx, jump 연산이면서 테이블에 있다면 예측한 값을 내보내고 아니라면 pc+4를 내보낸다

```
// 4. finally check "taken?"
always @(*) begin
    if ( (branch | is_jal | is_jalr)
        && (query_tag == tag_table[query_idx])
        && (pht[query_idx] >= 2'b10)) begin
        pred_pc = btb[query_idx];
    end else begin
        pred_pc = pc + 4;
    end
end
```

4 Discussion

4.1 어려웠던 부분은 다음과 같다

4.1.1 Flush가 일어나는 조건 설정(misprediction)

4.1.2 Flush가 일어나고 나서의 behavior

4.1.3 포워딩, hazard, prediction이 충돌하지 않아야 함

5 Conclusion

5.1 결과: 모든 테스트 케이스 통과

```
### SIMULATING ###
TEST END
SIM TIME : 72
TOTAL CYCLE : 35 (Answer : 36)
FINAL REGISTER OUTPUT
 0 00000000 (Answer : 00000000)
 1 00000000 (Answer : 00000000)
 2 00002ffc (Answer : 00002ffc)
 3 00000000 (Answer : 00000000)
 4 00000000 (Answer : 00000000)
 5 00000000 (Answer : 00000000)
 6 00000000 (Answer : 00000000)
 7 00000000 (Answer : 00000000)
 8 00000000 (Answer : 00000000)
 9 00000000 (Answer : 00000000)
10 00000013 (Answer : 00000013)
11 00000003 (Answer : 00000003)
12 fffffff7 (Answer : fffffff7)
13 00000037 (Answer : 00000037)
14 00000013 (Answer : 00000013)
15 00000026 (Answer : 00000026)
16 0000001e (Answer : 0000001e)
17 0000000a (Answer : 0000000a)
18 00000000 (Answer : 00000000)
19 00000000 (Answer : 00000000)
20 00000000 (Answer : 00000000)
21 00000000 (Answer : 00000000)
22 00000000 (Answer : 00000000)
23 00000000 (Answer : 00000000)
24 00000000 (Answer : 00000000)
25 00000000 (Answer : 00000000)
26 00000000 (Answer : 00000000)
27 00000000 (Answer : 00000000)
28 00000000 (Answer : 00000000)
29 00000000 (Answer : 00000000)
30 00000000 (Answer : 00000000)
31 00000000 (Answer : 00000000)
Correct output : 32/32
```

basic_mem cycle: 35 / 36

```
### SIMULATING ###
TEST END
SIM TIME : 88
TOTAL CYCLE : 43 (Answer : 44)
FINAL REGISTER OUTPUT
 0 00000000 (Answer : 00000000)
 1 00000000 (Answer : 00000000)
 2 00002ffc (Answer : 00002ffc)
 3 00000000 (Answer : 00000000)
 4 00000000 (Answer : 00000000)
 5 00000000 (Answer : 00000000)
 6 00000000 (Answer : 00000000)
 7 00000000 (Answer : 00000000)
 8 00000000 (Answer : 00000000)
 9 00000000 (Answer : 00000000)
10 00000000 (Answer : 00000000)
11 00000000 (Answer : 00000000)
12 00000000 (Answer : 00000000)
13 00000000 (Answer : 00000000)
14 0000000a (Answer : 0000000a)
15 00000028 (Answer : 00000028)
16 00000000 (Answer : 00000000)
17 0000000a (Answer : 0000000a)
18 00000000 (Answer : 00000000)
19 00000000 (Answer : 00000000)
20 00000000 (Answer : 00000000)
21 00000000 (Answer : 00000000)
22 00000000 (Answer : 00000000)
23 00000000 (Answer : 00000000)
24 00000000 (Answer : 00000000)
25 00000000 (Answer : 00000000)
26 00000000 (Answer : 00000000)
27 00000000 (Answer : 00000000)
28 00000000 (Answer : 00000000)
29 00000000 (Answer : 00000000)
30 00000000 (Answer : 00000000)
31 00000000 (Answer : 00000000)
Correct output : 32/32
```

ifelse_mem cycle: 43 / 44

```
### SIMULATING ###
TEST END
SIM TIME : 96
TOTAL CYCLE : 47 (Answer : 48)
FINAL REGISTER OUTPUT
 0 00000000 (Answer : 00000000)
 1 00000000 (Answer : 00000000)
 2 00002ffc (Answer : 00002ffc)
 3 00000000 (Answer : 00000000)
 4 00000000 (Answer : 00000000)
 5 00000000 (Answer : 00000000)
 6 00000000 (Answer : 00000000)
 7 00000000 (Answer : 00000000)
 8 00000000 (Answer : 00000000)
 9 00000000 (Answer : 00000000)
10 0000000a (Answer : 0000000a)
11 0000003f (Answer : 0000003f)
12 fffffff1 (Answer : fffffff1)
13 0000002f (Answer : 0000002f)
14 0000000e (Answer : 0000000e)
15 00000021 (Answer : 00000021)
16 0000000a (Answer : 0000000a)
17 0000000a (Answer : 0000000a)
18 00000000 (Answer : 00000000)
19 00000000 (Answer : 00000000)
20 00000000 (Answer : 00000000)
21 00000000 (Answer : 00000000)
22 00000000 (Answer : 00000000)
23 00000000 (Answer : 00000000)
24 00000000 (Answer : 00000000)
25 00000000 (Answer : 00000000)
26 00000000 (Answer : 00000000)
27 00000000 (Answer : 00000000)
28 00000000 (Answer : 00000000)
29 00000000 (Answer : 00000000)
30 00000000 (Answer : 00000000)
31 00000000 (Answer : 00000000)
Correct output : 32/32
```

non-controlflow_mem 47/44

```
### SIMULATING ###
TEST END
SIM TIME : 670
TOTAL CYCLE : 334 (Answer : 323)
FINAL REGISTER OUTPUT
 0 00000000 (Answer : 00000000)
 1 00000000 (Answer : 00000000)
 2 00002ffc (Answer : 00002ffc)
 3 00000000 (Answer : 00000000)
 4 00000000 (Answer : 00000000)
 5 00000000 (Answer : 00000000)
 6 00000000 (Answer : 00000000)
 7 00000000 (Answer : 00000000)
 8 00000000 (Answer : 00000000)
 9 00000000 (Answer : 00000000)
10 00000000 (Answer : 00000000)
11 00000000 (Answer : 00000000)
12 00000000 (Answer : 00000000)
13 00000000 (Answer : 00000000)
14 0000000a (Answer : 0000000a)
15 00000009 (Answer : 00000009)
16 0000005a (Answer : 0000005a)
17 0000000a (Answer : 0000000a)
18 00000000 (Answer : 00000000)
19 00000000 (Answer : 00000000)
20 00000000 (Answer : 00000000)
21 00000000 (Answer : 00000000)
22 00000000 (Answer : 00000000)
23 00000000 (Answer : 00000000)
24 00000000 (Answer : 00000000)
25 00000000 (Answer : 00000000)
26 00000000 (Answer : 00000000)
27 00000000 (Answer : 00000000)
28 00000000 (Answer : 00000000)
29 00000000 (Answer : 00000000)
30 00000000 (Answer : 00000000)
31 00000000 (Answer : 00000000)
Correct output : 32/32
```

loop_mem 334/323

```
### SIMULATING ###
TEST END
SIM TIME : 2622
TOTAL CYCLE : 1310 (Answer : 1188)
FINAL REGISTER OUTPUT
 0 00000000 (Answer : 00000000)
 1 00000000 (Answer : 00000000)
 2 00002ffc (Answer : 00002ffc)
 3 00000000 (Answer : 00000000)
 4 00000000 (Answer : 00000000)
 5 00000000 (Answer : 00000000)
 6 00000000 (Answer : 00000000)
 7 00000000 (Answer : 00000000)
 8 00000000 (Answer : 00000000)
 9 00000000 (Answer : 00000000)
10 0000000d (Answer : 0000000d)
11 00000000 (Answer : 00000000)
12 00000000 (Answer : 00000000)
13 00000000 (Answer : 00000000)
14 00000001 (Answer : 00000001)
15 0000000d (Answer : 0000000d)
16 00000015 (Answer : 00000015)
17 0000000a (Answer : 0000000a)
18 00000000 (Answer : 00000000)
19 00000000 (Answer : 00000000)
20 00000000 (Answer : 00000000)
21 00000022 (Answer : 00000022)
22 00000000 (Answer : 00000000)
23 00000037 (Answer : 00000037)
24 00000059 (Answer : 00000059)
25 00000000 (Answer : 00000000)
26 00000000 (Answer : 00000000)
27 00000000 (Answer : 00000000)
28 00000000 (Answer : 00000000)
29 00000000 (Answer : 00000000)
30 00000000 (Answer : 00000000)
31 00000000 (Answer : 00000000)
Correct output : 32/32
```

recursive_mem 1310/1188