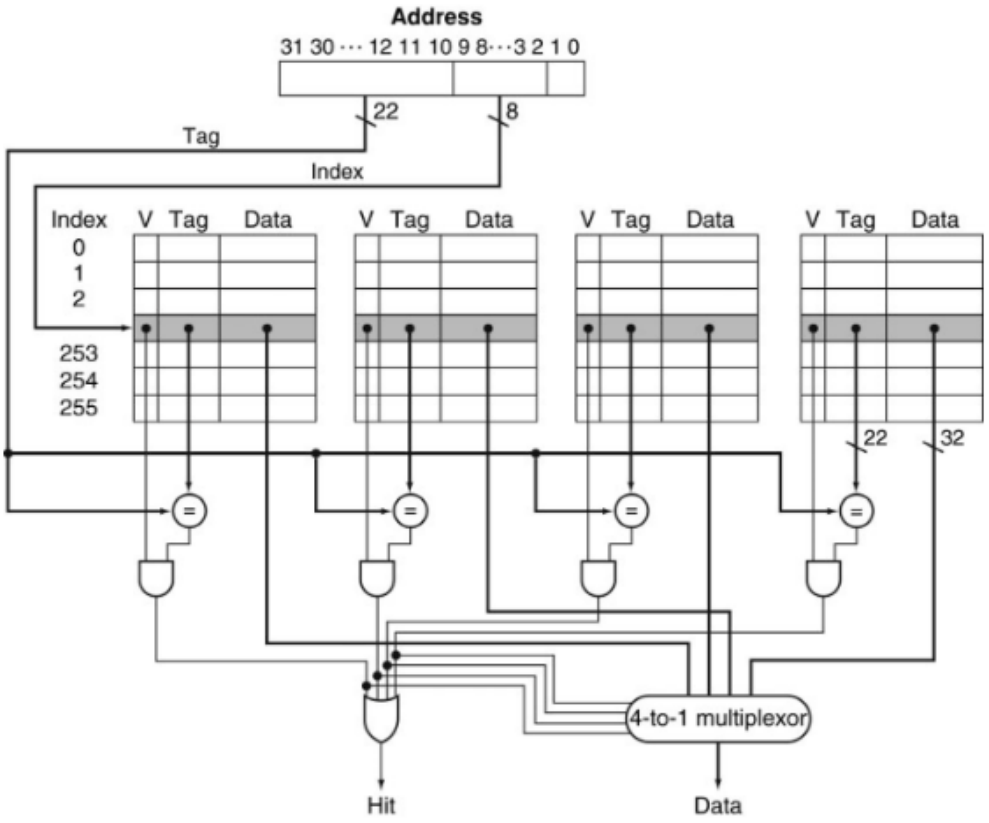
Address
31 30 ··· 12 11 10 9 8···3 2 1 0
22
8
Tag
Index
Index
V
Tag
Data
0
1
2
253
254
255
22
32
=
4-to-1 multiplexor
Hit
Data

<data_cache.v>

```
module data_cache #(
    parameter INDEX_BITS = 8,
    parameter OFFSET_BITS = 2,
    parameter TAG_BITS = 22,
    parameter WAYS = 4
)(
    input clk,
    input rst,
    input [31:0] addr,
    input [31:0] data_in,
    input we,  // Write Enable
    output reg [31:0] data_out,
    output reg hit
);
    // 캐시 메모리
    reg [TAG_BITS-1:0] tag_mem [0:2**INDEX_BITS-1][0:WAYS-1];
    reg valid_mem [0:2**INDEX_BITS-1][0:WAYS-1];
    reg [31:0] data_mem [0:2**INDEX_BITS-1][0:WAYS-1][0:2**OFFSET_BITS-1];
    reg [1:0] lru_mem [0:2**INDEX_BITS-1][0:WAYS-1];

    // 인덱스, 태그, 오프셋 추출
    wire [INDEX_BITS-1:0] index = addr[OFFSET_BITS + INDEX_BITS - 1:OFFSET_BITS];
    wire [TAG_BITS-1:0] tag = addr[31:OFFSET_BITS + INDEX_BITS];
    wire [OFFSET_BITS-1:0] offset = addr[OFFSET_BITS-1:0];

    integer i, j;
    integer lru_index;
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            // 캐시 초기화
            for (i = 0; i < 2**INDEX_BITS; i = i + 1) begin
                for (j = 0; j < WAYS; j = j + 1) begin
                    valid_mem[i][j] = 0;
                end
            end
        end else begin
            // 캐시 조회 및 갱신
            hit = 0;
            for (i = 0; i < WAYS; i = i + 1) begin
                if (valid_mem[index][i] && tag_mem[index][i] == tag) begin
                    hit = 1;
                    data_out = data_mem[index][i][offset];
                    if (we) begin
                        data_mem[index][i][offset] = data_in;
                    end
                    // LRU 업데이트
                    lru_mem[index][i] = 2'b00;
                end else if (valid_mem[index][i]) begin
                    lru_mem[index][i] = lru_mem[index][i] + 1;
                end
            end
            if (!hit && we) begin
                // 캐시 미스 처리 (LRU를 사용한 교체)
                lru_index = 0;
                for (i = 1; i < WAYS; i = i + 1) begin
                    if (lru_mem[index][i] > lru_mem[index][lru_index]) begin
                        lru_index = i;
                    end
                end
                tag_mem[index][lru_index] = tag;
                valid_mem[index][lru_index] = 1;
                data_mem[index][lru_index][offset] = data_in;
                lru_mem[index][lru_index] = 2'b00;
            end
        end
    end
endmodule
```

<instruction_cache.v>

```
module instruction_cache #( //4-way set associative instruction cache
    parameter INDEX_BITS = 8,
    parameter OFFSET_BITS = 2,
    parameter TAG_BITS = 22,
    parameter WAYS = 4
)(
    input clk,
    input rst,
    input [31:0] addr,
    output reg [31:0] instruction_out,
    output reg hit
);

    // 캐시 메모리
    reg [TAG_BITS-1:0] tag_mem [0:2**INDEX_BITS-1][0:WAYS-1];
    reg valid_mem [0:2**INDEX_BITS-1][0:WAYS-1];
    reg [31:0] instruction_mem [0:2**INDEX_BITS-1][0:WAYS-1][0:2**OFFSET_BITS-1];
    reg [1:0] lru_mem [0:2**INDEX_BITS-1][0:WAYS-1];

    // 인덱스, 태그, 오프셋 추출
    wire [INDEX_BITS-1:0] index = addr[OFFSET_BITS + INDEX_BITS - 1:OFFSET_BITS];
    wire [TAG_BITS-1:0] tag = addr[31:OFFSET_BITS + INDEX_BITS];
    wire [OFFSET_BITS-1:0] offset = addr[OFFSET_BITS-1:0];

    integer i, j;
    integer lru_index;
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            // 캐시 초기화
            for (i = 0; i < 2**INDEX_BITS; i = i + 1) begin
                for (j = 0; j < WAYS; j = j + 1) begin
                    valid_mem[i][j] = 0;
                end
            end
        end else begin
            // 캐시 조회
            hit = 0;
            for (i = 0; i < WAYS; i = i + 1) begin
                if (valid_mem[index][i] && tag_mem[index][i] == tag) begin
                    hit = 1;
                    instruction_out = instruction_mem[index][i][offset];
                    // LRU 업데이트
                    lru_mem[index][i] = 2'b00;
                end else if (valid_mem[index][i]) begin
                    lru_mem[index][i] = lru_mem[index][i] + 1;
                end
            end
            if (!hit) begin
                // 캐시 미스 처리 (명령어 캐시는 쓰기 동작이 없으므로 데이터 갱신 로직은 생략)
                lru_index = 0;
                for (i = 1; i < WAYS; i = i + 1) begin
                    if (lru_mem[index][i] > lru_mem[index][lru_index]) begin
                        lru_index = i;
                    end
                end
                tag_mem[index][lru_index] = tag;
                valid_mem[index][lru_index] = 1;
                // 명령어를 외부 메모리로부터 페치하는 부분
                instruction_mem[index][lru_index][offset] = 32'h00000000; // 임시로 0으로 초기화
                lru_mem[index][lru_index] = 2'b00;
            end
        end
    end
endmodule
```