UNIVERSITY OF NEVADA LAS VEGAS. DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING LABORATORIES.

Class:	CPE300L 1001		Semester:	**Fall 2024**
Points		Document author:	**Alireza Bolourian**	
		Author's email:	**bolouria@unlv.nevada.edu**	
		Document topic:	**Postlab 09**	
Instructor's comments:				

1. Theory of Operation

In this lab, a direct mapped cache is implemented in parallel with two memory banks corresponding to even and odd addresses. The cache uses the instruction address to determine the set and tag for accessing the cache. On a cache miss, instructions are fetched from memory and placed in the cache. Cache is utilized to access frequently used data faster, but space is limited. Eviction refers to replacing older data in cache memory to allow space for newer data.

2. Prelab

Instruction 1: Misses:1
Address 0

V	TAG	Word 1	Word 0
0			
1	0000 0000 0000 0000 0000 0000 000	200300c	20020005

Instruction 2: Misses:1
Address 4

V	TAG	Word 1	Word 0
0			
1	0000 0000 0000 0000 0000 0000 000	200300c	20020005

Instruction 3: Misses: 2
Address 8

V	TAG	Word 1	Word 0
1	0000 0000 0000 0000 0000 0000 000	00e22025	2067fff7
1	0000 0000 0000 0000 0000 0000 000	200300c	20020005

Instruction 4: Misses: 2
Address c

V	TAG	Word 1	Word 0
1	0000 0000 0000 0000 0000 0000 000	00e22025	2067fff7
1	0000 0000 0000 0000 0000 0000 000	200300c	20020005

Instruction 5: Misses: 3
Address 10

V	TAG	Word 1	Word 0
1	0000 0000 0000 0000 0000 0000 000	00e22025	2067fff7
1	0000 0000 0000 0000 0000 0000 001	00642824	00a42820

Instruction 6: Misses: 3
Address 14

V	TAG	Word 1	Word 0
1	0000 0000 0000 0000 0000 0000 000	00e22025	2067fff7
1	0000 0000 0000 0000 0000 0000 001	00a42820	00642824

Instruction 7: Misses: 4
Address 18

V	TAG	Word 1	Word 0
1	0000 0000 0000 0000 0000 0000 001	0064202a	10a7000a
1	0000 0000 0000 0000 0000 0000 001	00a42820	00642824

Instruction 8: **Misses: 4**
Address 1c

V	TAG	Word 1	Word 0
1	0000 0000 0000 0000 0000 0000 001	0064202a	10a7000a
1	0000 0000 0000 0000 0000 0000 001	00a42820	00642824

Instruction 9: **Misses: 5**
Address 20

V	TAG	Word 1	Word 0
1	0000 0000 0000 0000 0000 0000 001	0064202a	10a7000a
1	0000 0000 0000 0000 0000 0000 010	20050000	10800001

Instruction 10: **Misses: 6**
Address 28

V	TAG	Word 1	Word 0
1	0000 0000 0000 0000 0000 0000 010	00853820	00e2202a
1	0000 0000 0000 0000 0000 0000 010	20050000	10800001

Instruction 11: **Misses: 6**
Address 2c

V	TAG	Word 1	Word 0
1	0000 0000 0000 0000 0000 0000 010	00853820	00e2202a
1	0000 0000 0000 0000 0000 0000 010	20050000	10800001

Instruction 12: **Misses: 7**
Address 30

V	TAG	Word 1	Word 0
1	0000 0000 0000 0000 0000 0000 010	00853820	00e2202a
1	0000 0000 0000 0000 0000 0000 100	ac670044	00e23822

Instruction 13: **Misses: 7**
Address 34

V	TAG	Word 1	Word 0
1	0000 0000 0000 0000 0000 0000 010	00853820	00e2202a
1	0000 0000 0000 0000 0000 0000 100	ac670044	00e23822

Instruction 14: **Misses: 8**
Address 38

V	TAG	Word 1	Word 0
1	0000 0000 0000 0000 0000 0000 100	08000011	8c020050
1	0000 0000 0000 0000 0000 0000 100	ac670044	00e23822

Instruction 15: **Misses: 8**
Address 3c

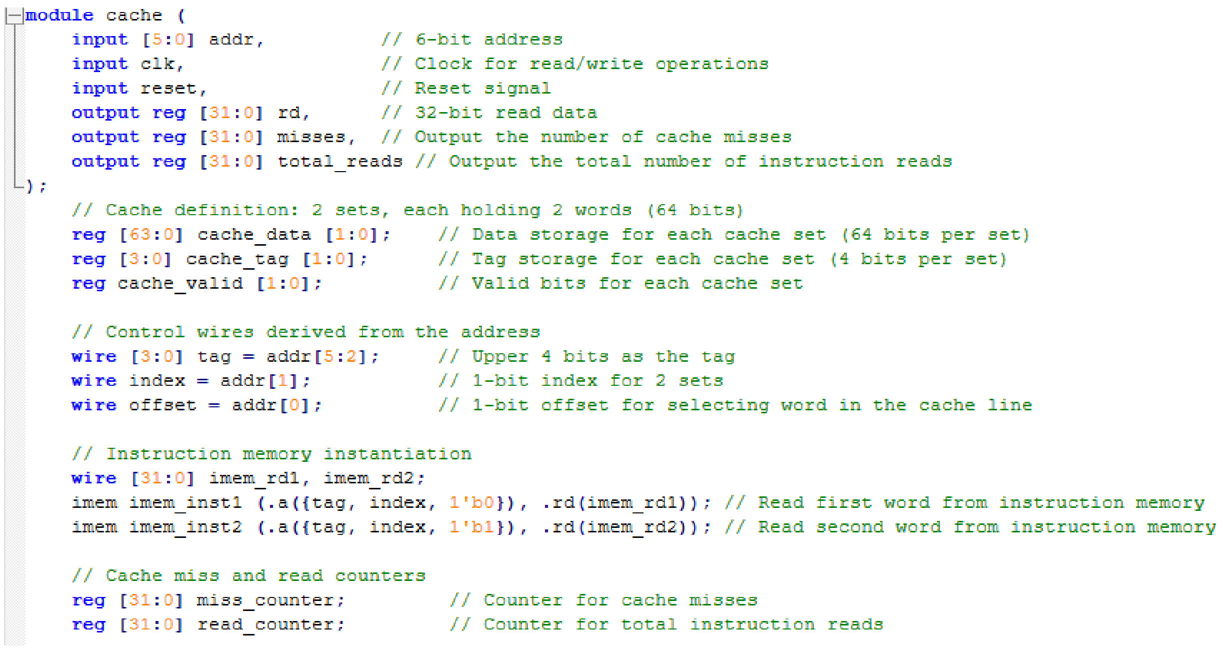
V	TAG	Word 1	Word 0
1	0000 0000 0000 0000 0000 0000 100	08000011	8c020050
1	0000 0000 0000 0000 0000 0000 100	ac670044	00e23822

Instruction 16: **Misses: 9**
Address 44

V	TAG	Word 1	Word 0
1	0000 0000 0000 0000 0000 0000 100	08000011	8c020050
1	0000 0000 0000 0000 0000 0001 000	ac020054	20020001

3. Experimental Results

A new module (cache) was added to the MIPS single stage and the testbench was modified to display the number of misses and total instructions that have been read. The full Verilog code for the single stage MIPS was uploaded separately.

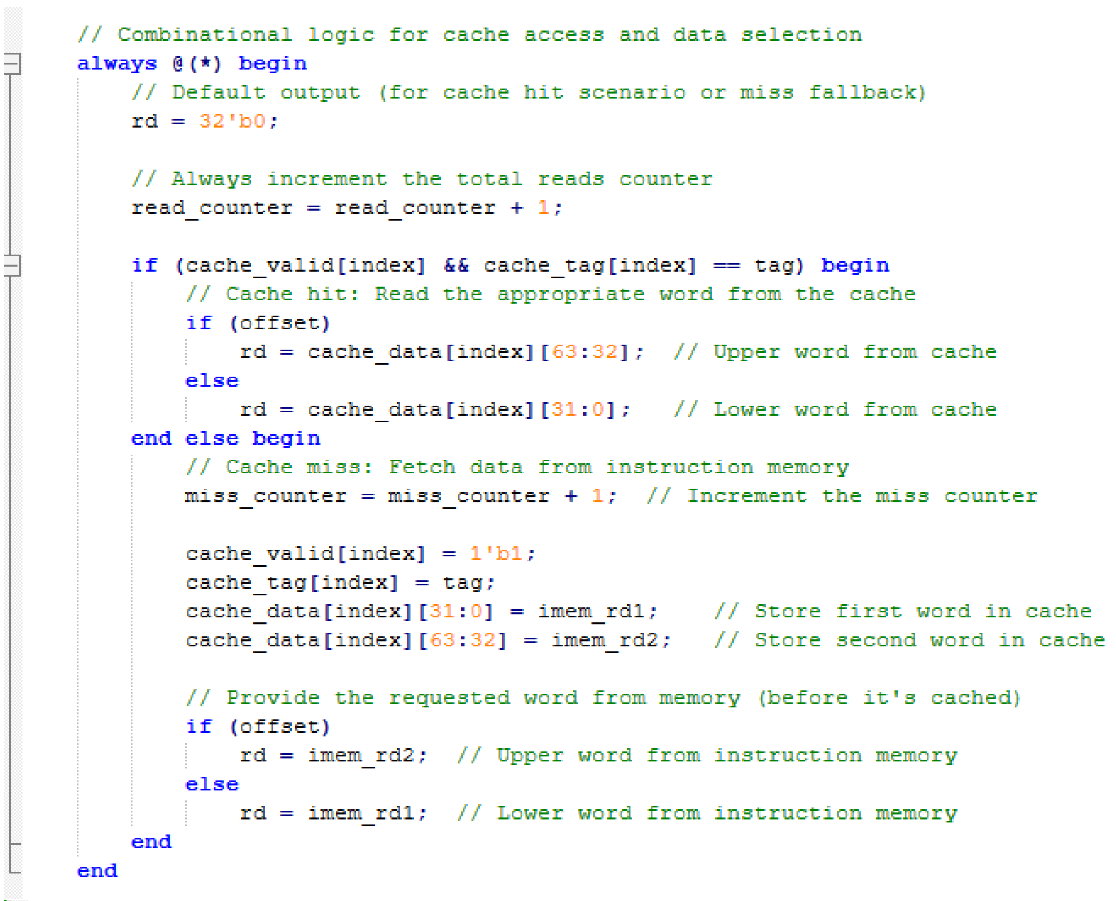


```
module cache (
    input [5:0] addr,          // 6-bit address
    input clk,                 // Clock for read/write operations
    input reset,               // Reset signal
    output reg [31:0] rd,      // 32-bit read data
    output reg [31:0] misses,  // Output the number of cache misses
    output reg [31:0] total_reads // Output the total number of instruction reads
);
    // Cache definition: 2 sets, each holding 2 words (64 bits)
    reg [63:0] cache_data [1:0];    // Data storage for each cache set (64 bits per set)
    reg [3:0] cache_tag [1:0];      // Tag storage for each cache set (4 bits per set)
    reg cache_valid [1:0];          // Valid bits for each cache set

    // Control wires derived from the address
    wire [3:0] tag = addr[5:2];     // Upper 4 bits as the tag
    wire index = addr[1];           // 1-bit index for 2 sets
    wire offset = addr[0];          // 1-bit offset for selecting word in the cache line

    // Instruction memory instantiation
    wire [31:0] imem_rd1, imem_rd2;
    imem imem_inst1 (.a({tag, index, 1'b0}), .rd(imem_rd1)); // Read first word from instruction memory
    imem imem_inst2 (.a({tag, index, 1'b1}), .rd(imem_rd2)); // Read second word from instruction memory

    // Cache miss and read counters
    reg [31:0] miss_counter;         // Counter for cache misses
    reg [31:0] read_counter;         // Counter for total instruction reads
```



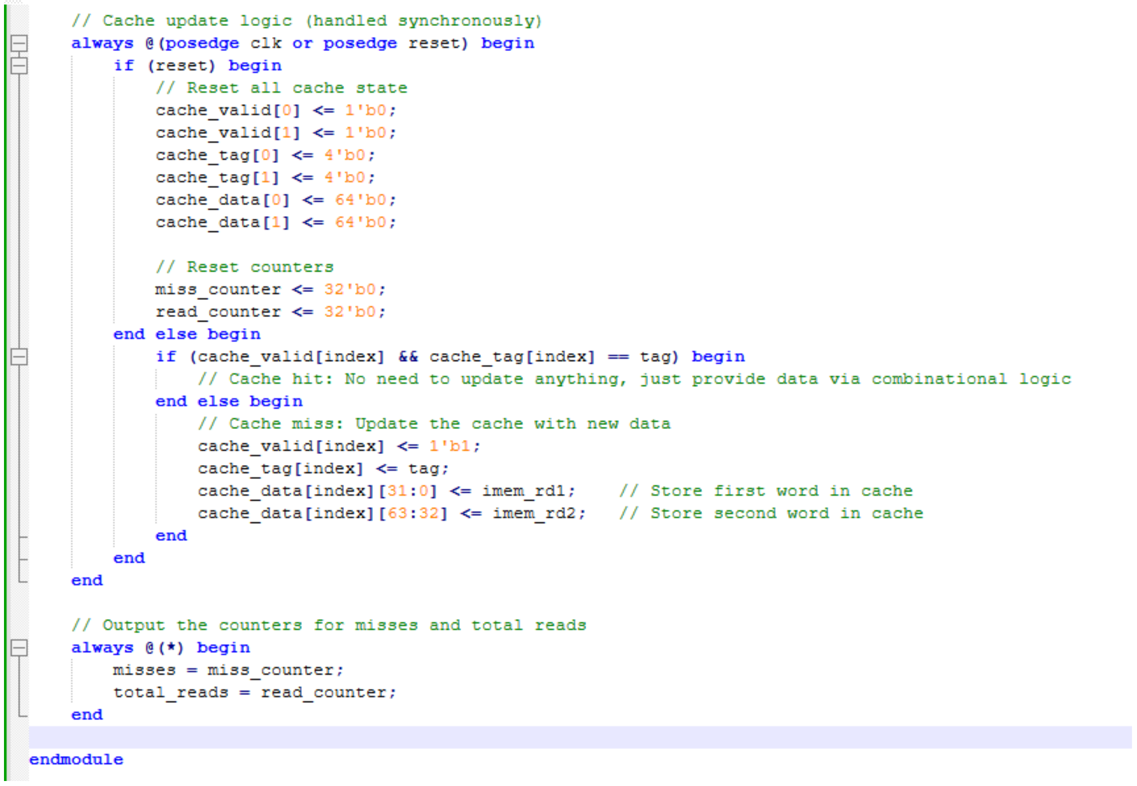
```verilog
// Combinational logic for cache access and data selection
always @(*) begin
    // Default output (for cache hit scenario or miss fallback)
    rd = 32'b0;

    // Always increment the total reads counter
    read_counter = read_counter + 1;

    if (cache_valid[index] && cache_tag[index] == tag) begin
        // Cache hit: Read the appropriate word from the cache
        if (offset)
            rd = cache_data[index][63:32];  // Upper word from cache
        else
            rd = cache_data[index][31:0];   // Lower word from cache
    end else begin
        // Cache miss: Fetch data from instruction memory
        miss_counter = miss_counter + 1;  // Increment the miss counter

        cache_valid[index] = 1'b1;
        cache_tag[index] = tag;
        cache_data[index][31:0] = imem_rd1;    // Store first word in cache
        cache_data[index][63:32] = imem_rd2;   // Store second word in cache

        // Provide the requested word from memory (before it's cached)
        if (offset)
            rd = imem_rd2;  // Upper word from instruction memory
        else
            rd = imem_rd1;  // Lower word from instruction memory
    end
end
```

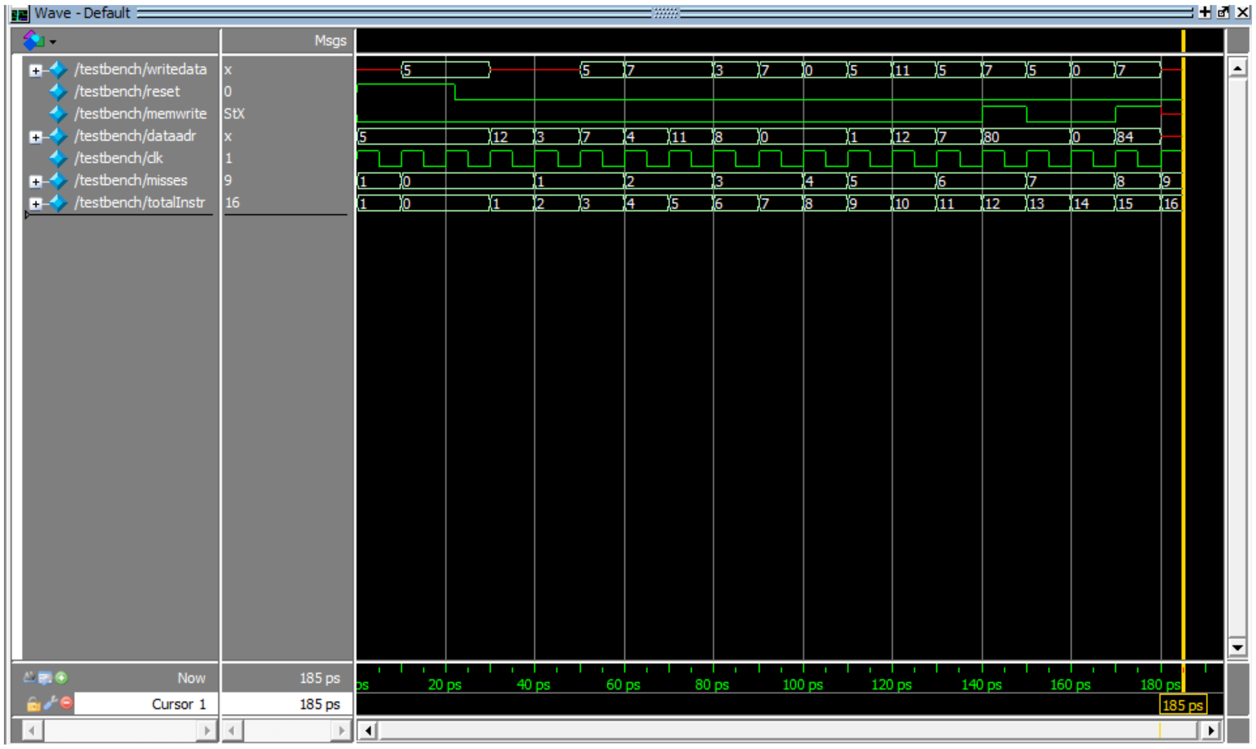


```verilog
    // Cache update logic (handled synchronously)
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            // Reset all cache state
            cache_valid[0] <= 1'b0;
            cache_valid[1] <= 1'b0;
            cache_tag[0] <= 4'b0;
            cache_tag[1] <= 4'b0;
            cache_data[0] <= 64'b0;
            cache_data[1] <= 64'b0;

            // Reset counters
            miss_counter <= 32'b0;
            read_counter <= 32'b0;
        end else begin
            if (cache_valid[index] && cache_tag[index] == tag) begin
                // Cache hit: No need to update anything, just provide data via combinational logic
            end else begin
                // Cache miss: Update the cache with new data
                cache_valid[index] <= 1'b1;
                cache_tag[index] <= tag;
                cache_data[index][31:0] <= imem_rd1;    // Store first word in cache
                cache_data[index][63:32] <= imem_rd2;   // Store second word in cache
            end
        end
    end

    // Output the counters for misses and total reads
    always @(*) begin
        misses = miss_counter;
        total_reads = read_counter;
    end

endmodule
```



The waveform shows the correct results and along with the count of cache misses and the total number of instructions as expected.

4. Questions

1. What is cache memory and why it's used?
Cache memory acts as an intermediary between the processor and main memory. It is smaller than main memory, but it much faster to access data. It is organized into different levels to provide faster access to frequently used data and improve performance.

2. Find the operation speed improvement for a system with and without cache, provide the result.
The average memory access time can be calculated using this formula:
$AMAT = (H \times Lcache) + ((1 - H) \times LRAM)$
H is the hit ratio, Lcache is cache latency time and LRAM is RAM latency
Assuming Lcahce of 1 cycle, LRAM of 100 cycles and Hit ratio of 0.50, then:
AMAT = 50.5 cycles.
The speedup is the ration of RAM latency and AMAT, so
$$Speedup = \frac{LRAM}{AMAT} = 1.980$$
So, the system with cache will perform 2 times better than the system without cache.

5. Conclusions

In this lab, I learned how cache is used to increase the performance of computers. It presents trade off between cost and access time. Cache memory is more expensive and so it is limited in capacity, but it is much faster. Due to this limited space, I realize that cache replacement policies are critical to determine what data are crucial and what data can be replaced and additionally, how the data must be managed in different cache levels.