



Topic 3. Cache Design

浙江大学计算机学院

2021年9月

Outline



- Experiment Purpose
- Experiment Task
- Basic Principle
- Operating Procedures
- Checkpoints



Experiment Purpose



- Understand Cache Line.
- Understand the principle of Cache Management Unit (CMU) and State Machine of CMU.
- Master the design methods of CMU.
- Master the design methods of Cache Line.
- master verification methods of Cache Line.



Experiment Task



Design of Cache Line and CMU.

Verify the Cache Line and CMU.

Observe the Waveform of Simulation.



Cache Line



LRU	V	D	Tag	Data



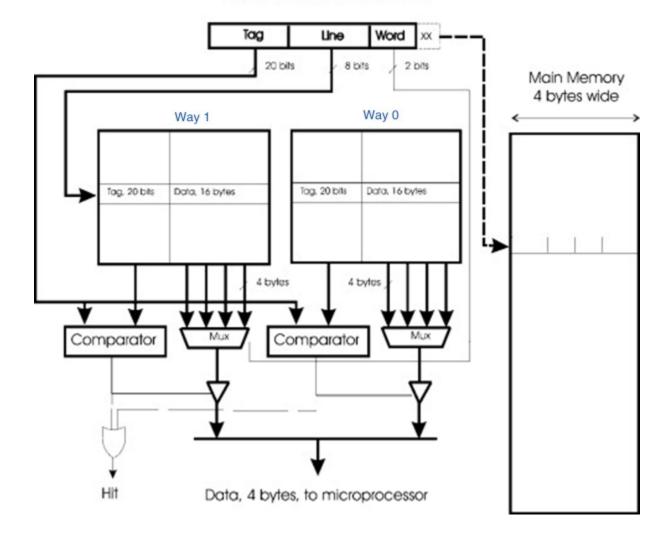
Cache Mode



- 2-way set associative
- Write Back
- Write Allocate



Address from microprocessor





Address



- Bytes of word = 4, WORD_BYTES_WIDTH = 2
- Words of line = 4, LINE_WORDS_WIDTH = 2
- Tag bits = 23
- Address bits = 32
- Ways = 2
- LINE_INDEX_WIDTH = ADDR_BITS TAG_BITS LINE_WORDS_WIDTH WORD_BYTES_WIDTH = ?
- Line Number = ?



Cache Line Memory



```
reg [LINE_NUM-1:0] inner_recent = 0;
```

```
reg [LINE_NUM-1:0] inner_valid = 0;
```

```
reg [LINE_NUM-1:0] inner_dirty = 0;
```

reg [TAG_BITS-1:0] inner_tag [0:LINE_NUM-1];

reg [WORD_BITS-1:0] inner_data [0:LINE_NUM*LINE_WORDS-1];





Input and output Signals of Cache Line

```
module cache (
          input wire clk, // clock
          input wire rst, // reset
          input wire [ADDR_BITS-1:0] addr, // address
          input wire load, // read refreshes recent bit
          input wire store, // set valid to 1 and reset dirty to 0
          input wire edit, // set dirty to 1
          input wire invalid, // reset valid to 0
          input wire [2:0] u b h w, // select signed or not & data width
                 // please refer to definition of LB, LH, LW, LBU, LHU in RV32I Instruction Set
          input wire [31:0] din, // data write in
          output reg hit = 0, // hit or not
          output reg [31:0] dout = 0, // data read out
          output reg valid = 0, // valid bit
          output reg dirty = 0, // dirty bit
          output reg [TAG BITS-1:0] tag = 0 // tag bits
```



Simulation Example(1)



```
// init
32'd10: begin
          load <= 0;
          store <= 1;
          edit <= 0;
          din <= 32'h11111111;
          addr <= 32'h00000004;
end
32'd11: begin
          addr <= 32'h0000000C;
end
32'd12: begin
          addr <= 32'h00000010;
end
32'd13: begin
          addr <= 32'h00000014;
end
```

```
// read miss
32'd14: begin
          load <= 1;
          store <= 0;
          edit <= 0;
          u_b_h_w <= 3'b010;
          din <= 0;
          addr <= 32'h00000020;
end
// read hit
32'd15: begin
          u_b_h_w <= 3'b010;
          addr <= 32'h00000010;
end
```



Simulation Example(2)



```
// write miss
32'd16: begin
          load <= 0;
          store <= 0;
          edit <= 1;
          u_b_h_w <= 3'b010;
          din <= 32'h2222222;
          addr <= 32'h000000024;
end
// write hit
32'd17: begin
          u b h w <= 3'b010;
          addr <= 32'h00000014;
end
```

```
// read line 0 of set 0, set recent bit
           32'd18: begin
                      load <= 1;
                      store <= 0;
                      edit <= 0;
                      u b h w <= 3'b010;
                      din <= 0:
                      addr <= 32'h00000004;
           end
// store to line 1 of set 0 due to line 0 recent
           32'd19: begin
                      load <= 0;
                      store <= 1;
                      edit <= 0;
                      u b h w <= 3'b010;
                      din <= 32'h33333333;
                      addr <= 32'h00000204;
           end
```



Simulation Example(3)

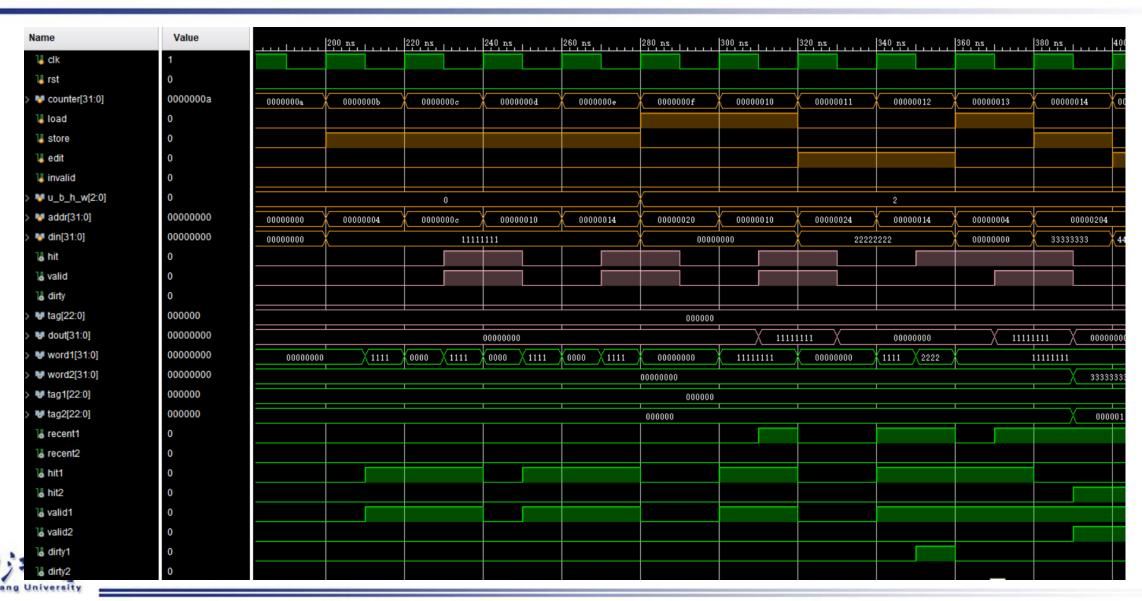


```
// read miss, tag mismatch. output tag (of
// edit line 1 of set 0, set dirty & recent
                                                   line 1), valid and dirty == 1
           32'd20: begin
                                                                                    32'd22: begin
                     load <= 0;
                                                                                               load <= 1;
                     store <= 0;
                                                                                               store <= 0;
                     edit <= 1;
                                                                                               edit <= 0;
                     u b h w <= 3'b010;
                                                                                               u b h w <= 3'b010;
                     din <= 32'h4444444;
                                                                                               din <= 32'h0;
                     addr <= 32'h00000204;
                                                                                               addr <= 32'h00000404;
          end
                                                                                    end
// read line 0 of set 0, set recent bit
                                                                         // auto replace line 1 of set 0
           32'd21: begin
                                                                                    32'd23: begin
                     load <= 1;
                                                                                               load <= 0;
                     store <= 0;
                     edit <= 0:
                                                                                               store <= 1;
                                                                                               edit <= 0;
                     u b h w <= 3'b010;
                                                                                               u_b_h_w <= 3'b010;
                     din <= 0;
                                                                                               din <= 32'h5555555;
                     addr <= 32'h00000004;
                                                                                               addr <= 32'h00000404:
          end
                                                                                    end
```



Simulation Example(1)





Simulation Example(2)



	Name	Value		31	80 ns		400 ns		420 ns		440 ns		460 ns		480 ns	سببا	500 ns	520	ns l	5	40 ns	560 ns	580 1	
	¼ dk	1																						
	¼ rst	0																		_				
	> 🐶 counter[31:0]	0000000a	0000001	: X	00000014		00000015		00000016		00000017		00000018		00000019		0000001a	\mathbb{X}	0000001ъ		0000001c	00000014	0000	
	¼ load	0																						
	¼ store	0																						
	¼ edit	0																						
	¼ invalid	0																						
	> 💖 u_b_h_w[2:0]	0	2																					
	> 😽 addr[31:0]	00000000	00000004 0000				204		00000004		000004		0404						000000	100				
	> 💖 din[31:0]	00000000	00000000 33333333		33	4444444		000000		000		5555555						000000	100					
	1å hit	0																						
	¹& valid	0																						
	¹⊌ dirty	0																						
	> 😻 tag[22:0]	000000	000000					000001			00000		000001			$\overline{}$				000002				
	> W dout[31:0]	00000000	000 \ 11111111 \ 00000000 \ 33333						333		11111	11111111		44444	1444	$\overline{}$	0000				0000			
	> W word1[31:0]	00000000	11111							1111								00000000						
	> W word2[31:0]	00000000	00000000 33333				333		44444		444			5555		0000000								
	> 😻 tag1[22:0]	000000												000000										
	> 😽 tag2[22:0]	000000	000000				000001					X							000002					
	¹⊌ recent1	0																						
	¼ recent2	0																						
	1⊌ hit1	0																						
	1⊌ hit2	0																0						
	¹å valid1	0																						
	¹⊌ valid2	0																						
	¹å dirty1	0																						
10	l⊌ dirty2	0																						



Simulation



- Write Simulation Code yourself
 - **□** cache initialization
 - □ read
 - miss
 - hit
 - **□** write
 - miss
 - hit



Checkpoints



• CP1:

Waveform Simulation of Cache Line.





Thanksl

