

Computer system III Lab 2 Report

3200105787 张云策

一. 实验目的

- 理解cache在CPU中的作用。
- 了解cache与流水线和内存的交互机制。
- 理解存储层次（Memory Hierarchy）。

二. 实验模块

- Cache:
 - D-cache:

```
1  `timescale 1ns / 1ps
2  ///////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date: 2022/03/29 11:57:25
7  // Design Name:
8  // Module Name: Cache
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 ///////////////////////////////////////////////////////////////////
21
22
23 module DCache(
24     input clk,
25     input rst,
26     input [6:0] debug_cache_index,
27     output reg [31:0] debug_cache_out0,
28     output reg [31:0] debug_cache_out1,
29     output reg [31:0] debug_cache_out2,
30     input [31:0] cache_req_addr,
31     input [31:0] cache_req_data,
32     input cache_req_wen,
33     input cache_req_valid,
34     output reg [31:0] cache_resp_data,
35     output reg cache_resp_stall,
36     output reg [31:0] mem_req_addr,
37     output reg [31:0] mem_req_data,
```

```

38     output reg mem_req_wen,
39     output reg mem_req_valid,
40     input [31:0] mem_resp_data,
41     input mem_resp_valid
42 );
43 reg [31:0] cache_data[0:127];
44 reg [22:0] cache_tag[0:127];
45 reg [6:0] cache_index[0:127];
46 reg [1:0] cache_offset[0:127];
47 reg cache_v[0:127];
48 reg cache_D[0:127];
49 reg hit,miss,read,write;
50 integer i;
51
52 always@(*)begin
53     debug_cache_out0<=cache_data[debug_cache_index];
54     debug_cache_out1<=
{cache_tag[debug_cache_index],cache_index[debug_cache_index
],cache_v[debug_cache_index],cache_D[debug_cache_index]};
55 end
56 always@(negedge clk or posedge rst)begin
57     if(rst==1)begin
58         cache_resp_data<=0;
59         cache_resp_stall<=0;
60         mem_req_addr<=0;
61         mem_req_data<=0;
62         mem_req_wen<=0;
63         mem_req_valid<=1'b0;
64         miss<=0;
65         hit<=0;
66         read<=0;
67         write<=0;
68         for (i = 0; i < 128; i = i + 1) begin
69             cache_data[i]<= 0;
70             cache_tag[i]<= 0;
71             cache_index[i]<=i;
72             cache_v[i]<=0;
73             cache_D[i]<=0;
74             cache_offset[i]<=0;
75         end
76     end
77     else if(cache_req_valid==1'b1)begin
78         if(cache_req_wen==0)begin
79             read<=1;
80             write<=0;
81             if(cache_req_addr==
{cache_tag[cache_req_addr[8:2]],cache_req_addr[8:2],cache_o
ffset[cache_req_addr[8:2]]}&&cache_v[cache_req_addr[8:2]]==
1'b1)begin//hit
82                 hit<=1;
83                 miss<=0;
84
85                 cache_resp_data<=cache_data[cache_req_addr[8:2]];
86                 cache_resp_stall<=0;
87                 mem_req_wen<=0;
88                 mem_req_valid<=1'b0;
89             end
90         else begin

```

```

90         miss<=1;
91         hit<=0;
92
93         if(cache_D[cache_req_addr[8:2]]==1'b1)begin
94             if(mem_resp_valid==1'b1) begin
95                 cache_resp_stall<=1;
96                 mem_req_wen<=0;
97                 mem_req_valid<=1'b1;
98                 mem_req_addr<=cache_req_addr;
99                 cache_D[cache_req_addr[8:2]]
100             <=1'b0;
101             end
102             else begin
103                 cache_resp_stall<=1;
104                 mem_req_addr<=
105                 {cache_tag[cache_req_addr[8:2]],cache_req_addr[8:2],cache_o
106                 fffset[cache_req_addr[8:2]]};
107                 mem_req_data<=cache_data[cache_req_addr[8:2]];
108                 mem_req_wen<=1'b1;
109                 mem_req_valid<=1'b1;
110             end
111             end
112             else begin
113                 if(mem_resp_valid==1'b1) begin
114                     cache_resp_data<=mem_resp_data;
115                     cache_resp_stall<=1'b0;
116                     mem_req_wen<=1'b0;
117                     mem_req_valid<=1'b0;
118                     cache_data[cache_req_addr[8:2]]
119                     <=mem_resp_data;
120                     cache_tag[cache_req_addr[8:2]]
121                     <=cache_req_addr[31:9];
122                     cache_offset[cache_req_addr[8:2]]<=cache_req_addr[1:0];
123                     cache_V[cache_req_addr[8:2]]
124                     <=1'b1;
125                 end
126                 else begin
127                     cache_resp_stall<=1;
128                     mem_req_addr<=cache_req_addr;
129                     mem_req_wen<=1'b0;
130                     mem_req_valid<=1'b1;
131                 end
132             end
133             end
134             else begin
135                 read<=0;
136                 write<=1;
137                 if((cache_req_addr==
138                 {cache_tag[cache_req_addr[8:2]],cache_req_addr[8:2],cache_o
139                 fffset[cache_req_addr[8:2]]}&&cache_V[cache_req_addr[8:2]]==
140                 1'b1)||cache_D[cache_req_addr[8:2]]==1'b0)begin//hit
141                     hit<=1;
142                     miss<=0;
143                     cache_resp_stall<=1'b0;
144                     mem_req_wen<=1'b0;

```

```

136         mem_req_valid<=1'b0;
137         cache_data[cache_req_addr[8:2]]
138         <=cache_req_data;
139         cache_tag[cache_req_addr[8:2]]
140         <=cache_req_addr[31:9];
141         cache_offset[cache_req_addr[8:2]]
142         <=cache_req_addr[1:0];
143         cache_v[cache_req_addr[8:2]]<=1'b1;
144         cache_D[cache_req_addr[8:2]]<=1'b1;
145     end
146     else begin
147         miss<=1;
148         hit<=0;
149         if(mem_resp_valid==1'b1) begin
150             cache_resp_stall<=1'b0;
151             mem_req_wen<=1'b0;
152             mem_req_valid<=1'b0;
153             cache_data[cache_req_addr[8:2]]
154             <=cache_req_data;
155             cache_tag[cache_req_addr[8:2]]
156             <=cache_req_addr[31:9];
157             cache_offset[cache_req_addr[8:2]]
158             <=cache_req_addr[1:0];
159             cache_v[cache_req_addr[8:2]]<=1'b1;
160             cache_D[cache_req_addr[8:2]]<=1'b1;
161         end
162         else begin
163             cache_resp_stall<=1;
164             mem_req_addr<=
165             {cache_tag[cache_req_addr[8:2]],cache_req_addr[8:2],cache_o
166             fffset[cache_req_addr[8:2]]};
167             mem_req_data<=cache_data[cache_req_addr[8:2]];
168             mem_req_wen<=1'b1;
169             mem_req_valid<=1'b1;
170         end
171     end
172 end
173 end
174 end
175 endmodule

```

o I-cache:

```

1  `timescale 1ns / 1ps
2  //////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date: 2022/03/29 12:27:42
7  // Design Name:
8  // Module Name: Cache
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description:

```

```

13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 ///////////////////////////////////////////////////////////////////
21 ///////////////////////////////////////////////////////////////////
22
23 module ICache(
24     input [31:0] cache_req_addr,
25     output reg [31:0] cache_resp_data,
26     output reg [31:0] mem_req_addr,
27     input [31:0] mem_resp_data
28 );
29     reg [31:0] cache_data[0:127];
30     reg [22:0] cache_tag[0:127];
31     reg [6:0] cache_index[0:127];
32     reg [1:0] cache_offset[0:127];
33     reg cache_v[0:127];
34     reg cache_D[0:127];
35     integer i;
36     initial begin
37         cache_resp_data<=0;
38         for (i = 0; i < 128; i = i + 1) begin
39             cache_data[i] <= 0;
40             cache_tag[i] <= 0;
41             cache_index[i]<=i;
42             cache_v[i]<=0;
43             cache_D[i]<=0;
44             cache_offset[i]<=0;
45         end
46     end
47     always@(*)begin
48         if(cache_req_addr==
49 {cache_tag[cache_req_addr[8:2]],cache_req_addr[8:2],cache_of
50 fset[cache_req_addr[8:2]]}&&cache_v[cache_req_addr[8:2]]==1'
51 b1)begin//hit
52
53         cache_resp_data<=cache_data[cache_req_addr[8:2]];
54         end
55         else begin//miss
56             cache_resp_data<=mem_resp_data;
57             cache_data[cache_req_addr[8:2]]<=mem_resp_data;
58             cache_tag[cache_req_addr[8:2]]
59 <=cache_req_addr[31:9];
60             cache_offset[cache_req_addr[8:2]]
61 <=cache_req_addr[1:0];
62             cache_v[cache_req_addr[8:2]]<=1'b1;
63             cache_D[cache_req_addr[8:2]]<=1'b0;
64         end
65     end
66 endmodule

```

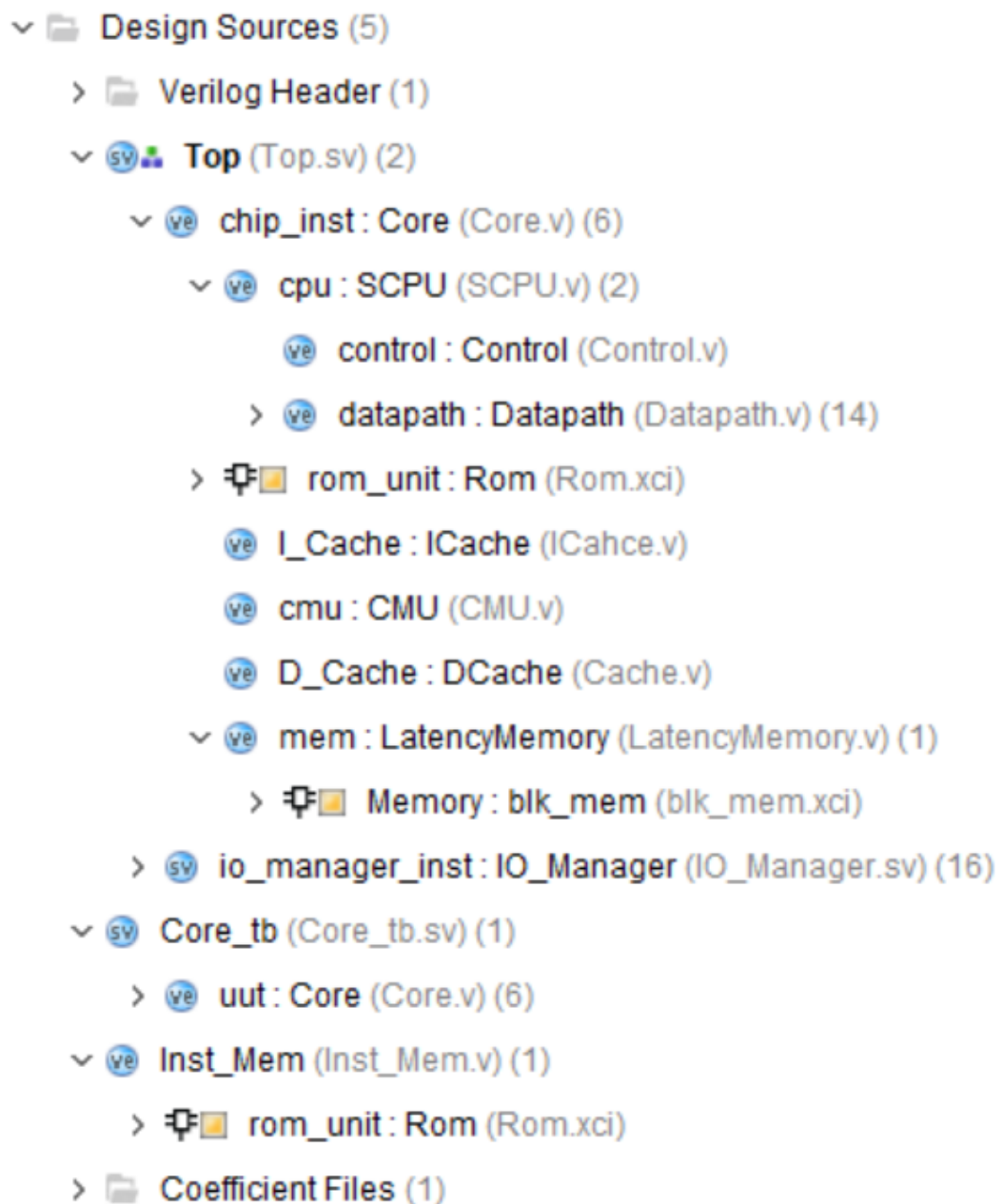
- Cache内部交互设计:

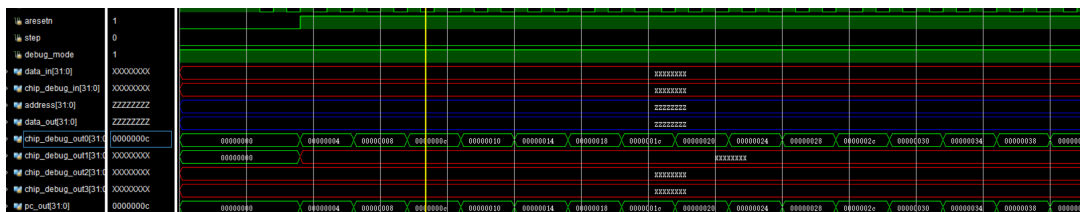
```

1  module CMU(
2      input  [6:0] op_code,
3      output reg cache_req_wen,
4      output reg cache_req_valid
5  );
6      always @(*) begin
7          case(op_code)
8              7'b0100011:begin cache_req_wen <= 1'b1;cache_req_valid
9  <=1'b1; end //SW
10             7'b0000011:begin cache_req_wen <= 1'b0;cache_req_valid
11 <=1'b1; end //LW
12             default:begin cache_req_valid<=1'b0; cache_req_wen <=
13 1'b0;end
14         endcase
15     end
16 endmodule

```

- 仿真:





- 上板:



三. 实验思考题

- 由题意得: Block Memory为 $2^{13} = 8192\text{KB}$, 所以地址位占13; 而Cache中存在 $2^9 = 512$ 个block, 所以index寻址位占9位; 标定tag占4位; 而valid bit和dirty bit各占一位, 故得出:

4bit	9bit	0bit
—tag—	—index—	—offset—
- D-cache:

```

1  Begin:
2      # Reg[1-5]: Used to record data
3      0x00    addi    x1,    x0,    123        # x1 = 123  Data
4      0x04    addi    x2,    x1,    111        # x2 = 234  Data
5      0x08    addi    x3,    x2,    111        # x3 = 345  Data
6      0x0c    addi    x4,    x3,    111        # x4 = 456  Data
7      0x10    addi    x5,    x4,    111        # x5 = 567  Data
8      # Reg[6-10]: Used to store test address
9      0x14    addi    x6,    x0,    0x0
10     0x18    addi    x7,    x0,    0x4
11     0x1c    addi    x8,    x0,    0x94
12     0x20    addi    x9,    x0,    0x98
13     0x24    addi    x10,   x0,    0x404
14     # Reg[20]: Used to record score
15     0x28    addi    x20,   x0,    0          # x20 = 0   以上均与D-
cache功能无关
16     Test_Hit:

```

```

17 0x2c sw x1, 0(x6) # hit 写
18 0x30 sw x2, 0(x7) # hit 写
19 0x34 lw x11, 0(x6) # hit
20 0x38 sw x3, 0(x8) # hit 写
21 0x3c sw x4, 0(x9) # hit 写
22 0x40 lw x13, 0(x8) # hit
23 0x44 lw x12, 0(x7) # hit
24 0x48 lw x14, 0(x9) # hit
25 0x4c sub x11, x11, x13 # 无关
26 0x50 sub x12, x12, x14 # 无关
27 0x54 bne x11, x12, Test_Replacement # 无关
28 0x58 addi x20, x20, 1 # 无关
29 Test_Replacement:
30 0x5c sw x5, 0(x10) # miss, 写缺失, 由dirty-
    bit处理
31 0x60 lw x11, 0(x7) # miss, 读缺失
32 0x64 lw x12, 0(x10) # miss, 读缺失
33 0x68 addi x11, x11, 333 # 无关
34 0x6c bne x11, x12, Test_Dirty_Block # 无关
35 0x70 addi x20, x20, 1 # 无关
36 Test_Dirty_Block:
37 0x74 lw x11, 0(x6) # miss, 读缺失
38 0x78 lw x12, 0(x10) # miss, 读缺失
39 0x7c addi x11, x11, 444 # 无关
40 0x80 bne x11, x12, End # 无关
41 0x84 addi x20, x20, 1 # 无关
42 End:
43 #-----Useless Loop-----
44 0x88 addi x21, x0, 3
45 0x8c bne x21, x20, End
46 #-----Useless Loop-----

```