浙江大学

本科实验报告

课程名称：计算机体系结构

姓　　名：汪辉

学　　院：计算机科学与技术学院

系：

专　　业：计算机科学与技术

学　　号：3190105609

指导教师：陈文智

2021 年 11 月 2 日

浙江大学实验报告

课程名称：计算机体系结构 实验类型：综合

实验项目名称：Cache Design

学生姓名：汪辉 专业：计算机科学与技术 学号：3190105609

同组学生姓名：王嘉豪 指导老师：陈文智

实验地点：曹西 301 实验日期：2021 年 11 月 2 日

一、 实验目的和要求

1. 理解 Cache Line。
2. 理解缓存管理单元 (CMU)和 CMU 状态机的原理。
3. 掌握 CMU 的设计方法。
4. 掌握 Cache Line 的设计方法。
5. 掌握检验 Cache Line 的方法。

二、 实验内容和原理

1. 实验内容

   1.1. 设计 Cache Line 和 CMU

   1.2. 合适 Cache Line 和 CMU

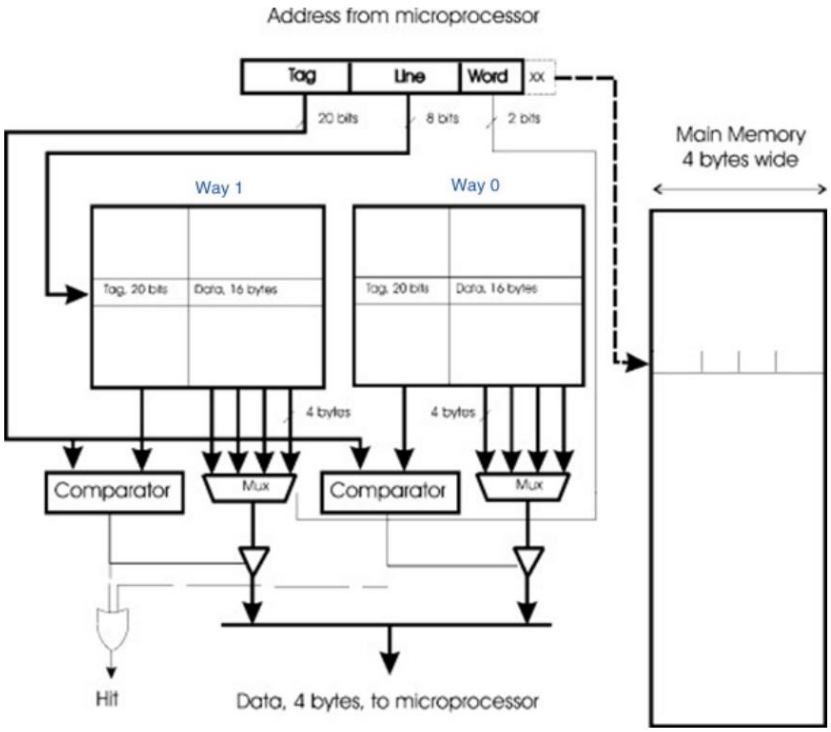
   1.3. 观察仿真波形

2. 实验原理

   2.1. Cache Line

LRU	V	D	Tag	Data

2.2. Cache Mode



三、 实验过程和数据记录

本实验的主要工作是编写 cache.v，以下对相关代码进行分析。代码开头给出的地址结构比较重要。

```
// | ----------- address 32 ----------- |
// | 31   9 | 8     4 | 3    2 | 1    0 |
// | tag 23 | index 5 | word 2 | byte 2 |
```

1. 从地址中截取部分作为 tag 和 index 的值

```
    assign addr_tag = addr[ADDR_BITS-1:ADDR_BITS-TAG_BITS];
    assign addr_index = addr[ADDR_BITS-TAG_BITS-1:BLOCK_WIDTH];
```

2. 补全地址信号

```
assign addr_element1 = {addr_index, 1'b0};  // the 1st block id in set
assign addr_element2 = {addr_index, 1'b1};  // the 2nd block id in set
assign addr_word1 = {addr_element1, addr[ELEMENT_WORDS_WIDTH+WORD_BYTES_WIDTH-1:WORD_BYTES_WIDTH]};
assign addr_word2 = {addr_element2, addr[ELEMENT_WORDS_WIDTH+WORD_BYTES_WIDTH-1:WORD_BYTES_WIDTH]};
```

element 地址是 set 地址加上一位 way 选择，word 地址是 element 地址加上 word 的偏移量。

3. 补全数据信号

```
    assign word1 = inner_data[addr_word1];
    assign word2 = inner_data[addr_word2];
    assign half_word1 = addr[1] ? word1[31:16] : word1[15:0];
    assign half_word2 = addr[1] ? word2[31:16] : word2[15:0];
    assign byte1 = addr[1] ?
```

```
                    addr[0] ? word1[31:24] : word1[23:16] :
                    addr[0] ? word1[15:8] :  word1[7:0]   ;
    assign byte2 = addr[1] ?
                    addr[0] ? word2[31:24] : word2[23:16] :
                    addr[0] ? word2[15:8] :  word2[7:0]   ;
```

word 的数据是根据地址从数据区直接获取。half_word 的数据是在 word 中选取高位或者低位，由 addr[1]决定。byte 是在 half_word 的基础上由 addr[0]决定是高位还是低位。

4. 补充 element 状态和 tag

```
    assign recent1 = inner_recent[addr_element1];
    assign recent2 = inner_recent[addr_element2];
    assign valid1 = inner_valid[addr_element1];
    assign valid2 = inner_valid[addr_element2];
    assign dirty1 = inner_dirty[addr_element1];
    assign dirty2 = inner_dirty[addr_element2];
    assign tag1 = inner_tag[addr_element1];
    assign tag2 = inner_tag[addr_element2];
```

根据 element 地址，从状态数组中找到对应的状态值，从 tag 数组中找到对应的 tag 值。

5. 命中

```
    assign hit1 = valid1 & (tag1 == addr_tag);  // if 1st block hit
    assign hit2 = valid2 & (tag2 == addr_tag);  // if 2nd block hit
```

当 tag 一致且为有效状态时说明命中。

6. 部分状态

```
    valid <= (!recent1 & valid1) | (!recent2 & valid2); // recent is the
    dirty <= (!recent1 & dirty1) | (!recent2 & dirty2);
    tag <= ({23{~recent1}} & tag1) | ({23{~recent2}} & tag2);
    hit <= hit1 | hit2;
```

当一个 element 近期没有使用过且状态有效时，则状态有效。当一个 element 近期没有使用过且 dirty 为 1 时，dirty 赋为 1。最近未使用过的 element 的 tag 即为 tag。任意一个 element 命中时说明命中。

7. load 时命中

```
else if (hit2) begin
    dout <=
        u_b_h_w[1] ? word2 :
        u_b_h_w[0] ? {u_b_h_w[2] ? 16'b0 : {16{half_word2[15]}},
half_word2} :
        {u_b_h_w[2] ? 24'b0 : {24{byte2[7]}}, byte2};
```

```
    inner_recent[addr_element2] <= 1'b1;
    inner_recent[addr_element1] <= 1'b0;
end
```

若 u_b_h_w[1]为 1，则数据为字；若其为 0，且 u_b_h_w[0]=1&& u_b_h_w[2]=1，则数据为 half_word 且无符号拓展，若 u_b_h_w[0]=1&& u_b_h_w[2]=0，则数据为 half_word 且有符号拓展，若 u_b_h_w[0]=0&& u_b_h_w[2]=1，数据为 byte 且无符号拓展，若 u_b_h_w[0]=0&& u_b_h_w[2]=0，数据为 byte 且有符号拓展。

8. edit 时命中

```
    else if (hit2) begin
        //need to fill in
    inner_data[addr_word2] <=
        u_b_h_w[1] ?        // word?
            din
        :
            u_b_h_w[0] ?    // half word?
                addr[1] ?       // upper / lower?
                    {din[15:0], word2[15:0]}
                :
                    {word2[31:16], din[15:0]}
            :   // byte
                addr[1] ?
                    addr[0] ?
                        {din[7:0], word2[23:0]}   // 11
                    :
                        {word2[31:24], din[7:0], word2[15:0]} // 10
                :
                    addr[0] ?
                        {word2[31:16], din[7:0], word2[7:0]}   // 01
                    :
                        {word2[31:8], din[7:0]} // 00
    ;
    inner_dirty[addr_element2] <= 1'b1;
    inner_recent[addr_element2] <= 1'b1;
    inner_recent[addr_element1] <= 1'b0;
end
```

命中时，写入数据由 u_b_h_w[]、addr[]、din 决定。

9. store 时发现最近使用过

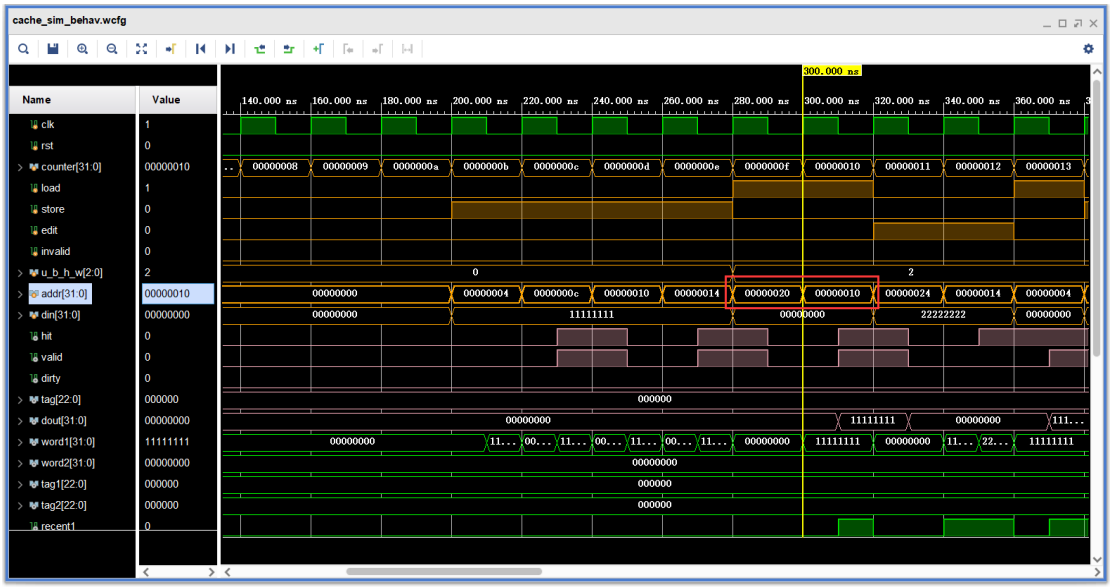
```
else begin
    inner_data[addr_word1] <= din;
    inner_valid[addr_element1] <= 1'b1;
    inner_dirty[addr_element1] <= 1'b0;
```

```
    inner_tag[addr_element1] <= addr_tag;
end
```

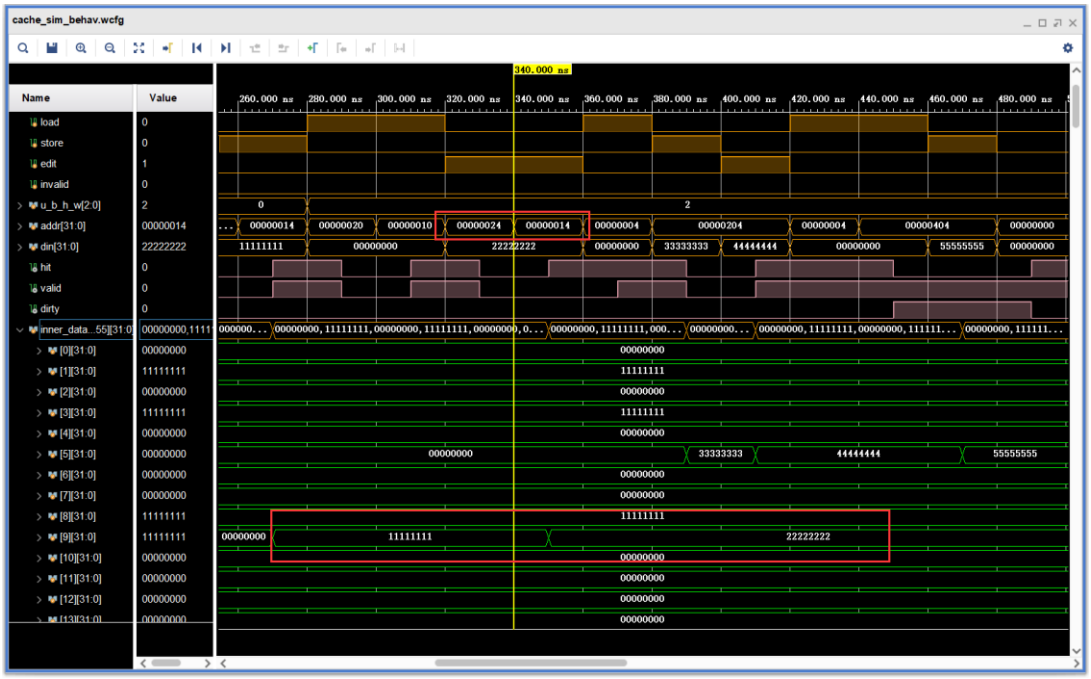
若 element1 最近使用过，则将数据写入 element2 的数据区，将其有效位赋 1，脏位赋 0，tag 改为输入的 tag。

四、 实验结果分析

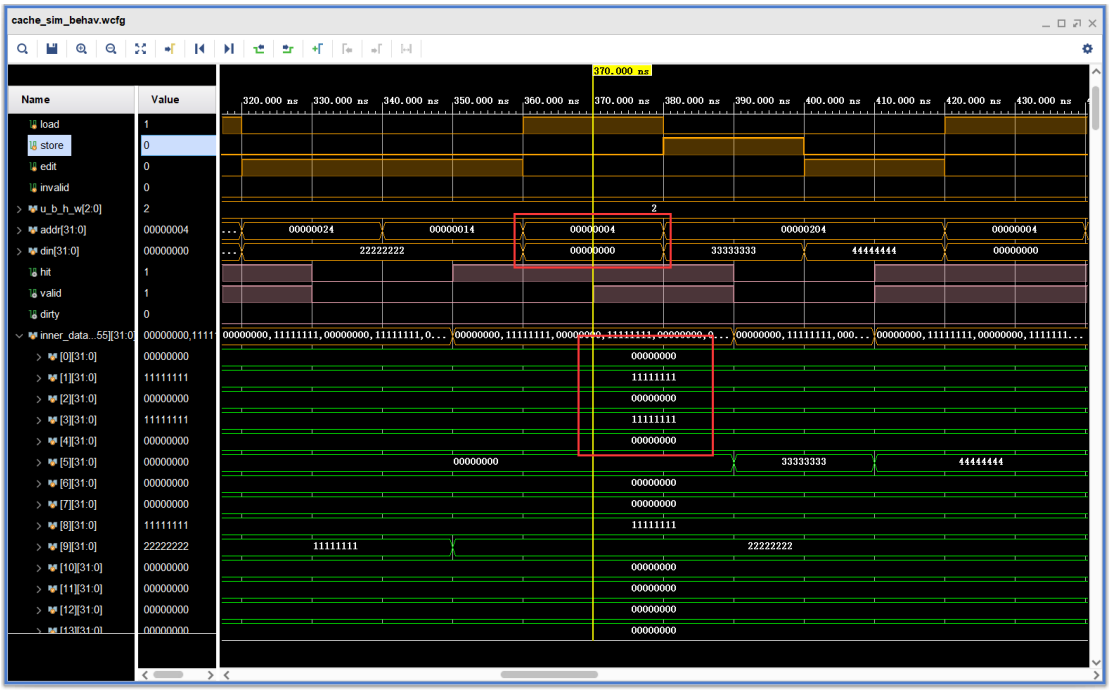
1. read miss and hit



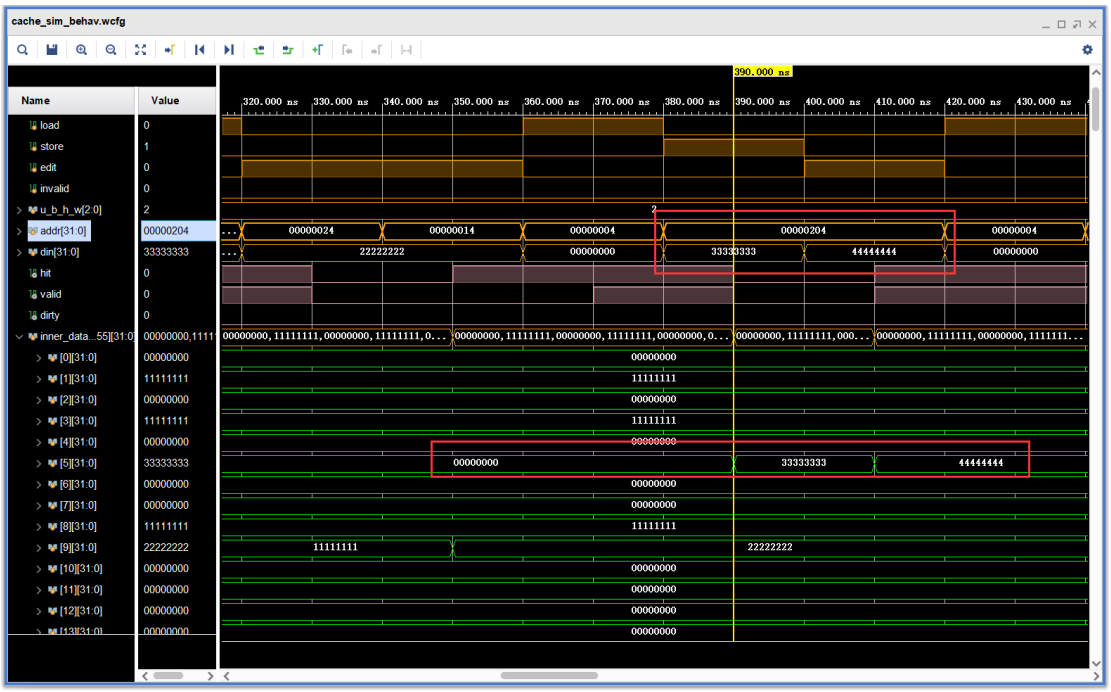
2. write miss and hit



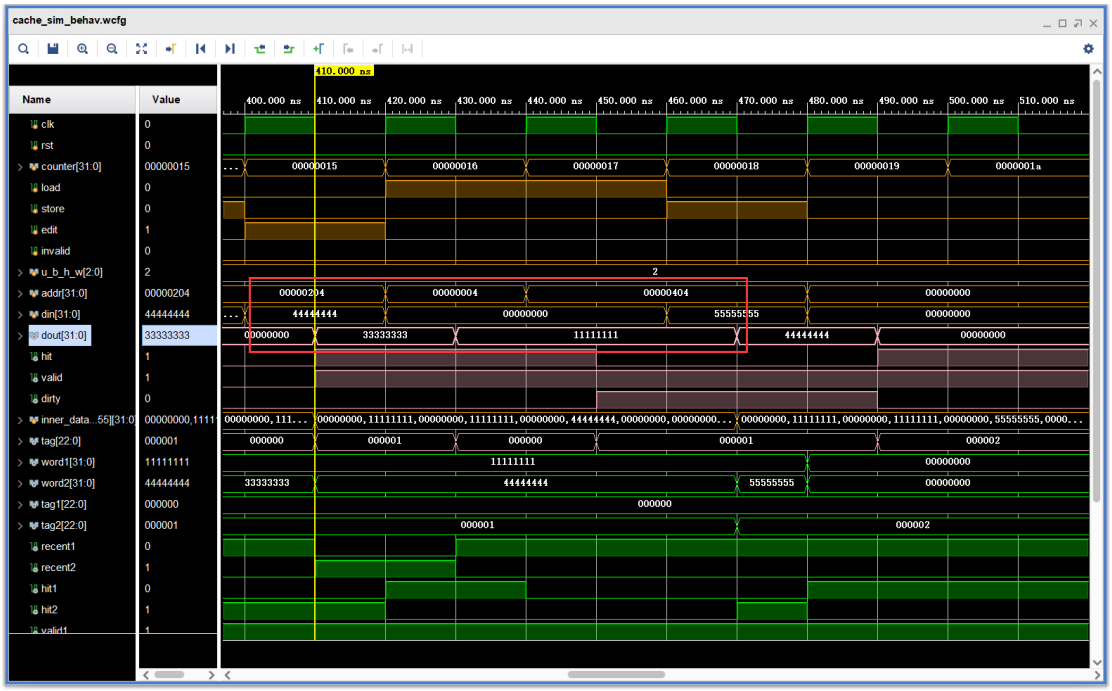
3. read line 0 of set 0, set recent bit



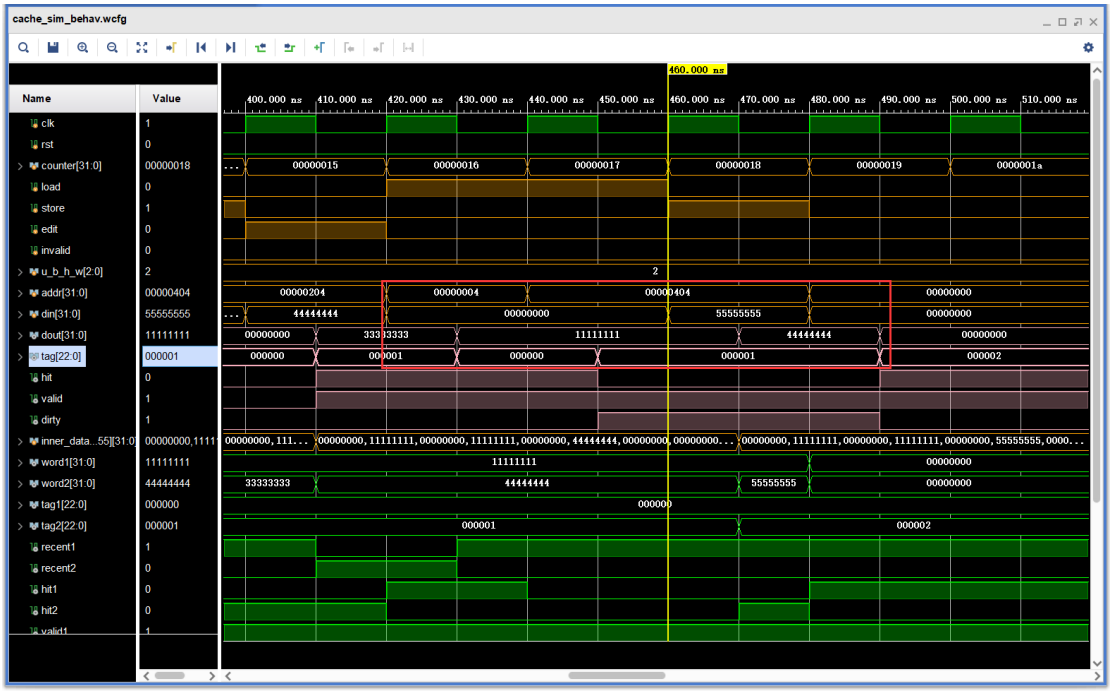
4. store to line 1 of set 0 due to line 0 recent



5. edit line 1 of set 0, set dirty & recent



6. tag mismatch. output tag, valid and dirty == 1



五、 讨论与心得

本实验是 Cache 的设计，但从实验过程来说，只需要根据老师给出的一半代码进行对应的一点变换即可，但是我们还是需要搞清楚其中的原理，由于实验结构比较简单，所以很多问题都可以直接通过仿真找到原因，直接进行修改，联系课内学到的知识，能够比较容易解决。本次实验的难点在于对于 read miss、write miss 等各种情况的判断，需要根据每种情况，联系课内知识，给出对应的操作。