《计算机组成原理与接口技术实验》

实验报告

学 院 名 称：数据科学与计算机学院

学 生 姓 名：黄羽盼

学　　号：14331106

专业（班级）：14 软件工程三（6）班

合 作 者：蒋子涵

时　　间：2016 年 6 月 6 日

成 绩 ：

实验四 ：Cache控制器设计

一. 实验目的

1. 认识和掌握Cache控制器的原理及其设计方法；
2. 掌握Cache控制器的实现方法，代码实现方法。

二. 实验内容

本实验要求实现 Cache（数据 Cache）及其地址变换逻辑（也叫 Cache 控制器），采用直接相联地址变换，CPU 从 Cache 读数据，若读不到，还必须考虑先从主存中读取数据，然后再将数据写到 Cache 中，之后，将数据送往 CPU；其次，CPU 还要向存储器写数据。

说明：CLK 为系统时钟（用于计数器计数控制等操作），CLR 为系统总清零信号（清区表存储器、计数器），WCT 为写 Cache 区表存储器信号，AB31..AB0 为 CPU 访问内存的地址（地址总线），32 位数据为 DB31..DB0（数据总线），RD（为 0，读）为 Cache 的读信号，MWr（为 1，写）为主存的写信号，MRd（为 0，读）为主存的读信号，M（为 1，有效，命中）为 Cache 有效位标志信号（区表存储器标志位），CA17..CA0 为 Cache 地址，MD31..MD0 为主存送 Cache 的数据，D31..D0 为 Cache 送 CPU 数据，LA3...LA0 为块内地址。

三. 实验原理

本实验采用的地址变换是直接相联映象方式，这种变换方式简单而直接，硬件实现很简单，访问速度也比较快，但是块的冲突率比较高。其主要原则是：主存中一块只能映象到 Cache 的一个特定的块中。

假设主存的块号为 B，Cache 的块号为 b，则它们之间的映象关系可以表示为：

$$b = B \bmod C_b$$

其中，C_b 是 Cache 的块容量。设主存的块容量为 M_b，区容量为 M_e，则直接映象方法的关系如图 1 所示。把主存按 Cache 的大小分成区，一般主存容量为 Cache 容量的整数倍，主存每一个分区内的块数与 Cache 的总块数相等。直接映象方式只能把主存各个区中相对块号相同的那些块映象到 Cache 中同一块号的那个特定块中。例如，主存的块 0 只能映象到 Cache 的块 0 中，主存的块 1 只能映象到 Cache 的块 1 中，同样，主存区 1 中的块 C_b（在区 1 中的相对块号是 0），也只能映象到 Cache 的块 0 中，看图 1。根据上面给出的地址映象规则，整个 Cache 地址与主存地址的低位部分是完全相同的。

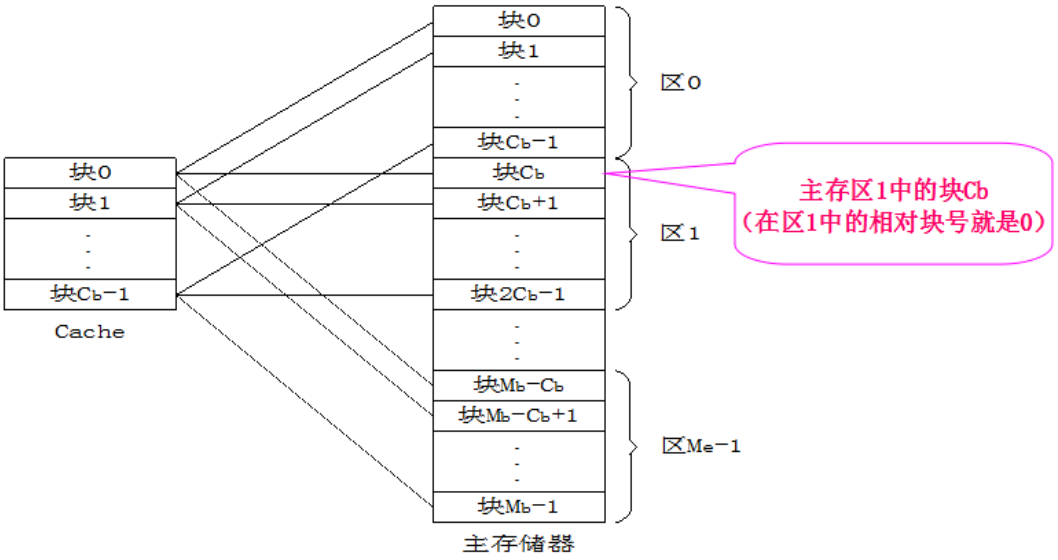


图 1 直接相联映象方式

直接映象方式的地址变换过程如图 2 所示，主存地址分为三个部分：区号 E、块号 B 和块内地址 W；Cache 地址分为两部分：块号 b 和块内地址 w。主存地址中的块号 B 与 Cache 地址中的块号 b 是完全相同的。同样，主存地址中的块内地址 W 与 Cache 地址中的块内地址 w 也是完全相同的，主存地址比 Cache 地址长出来的部分称为区号 E。

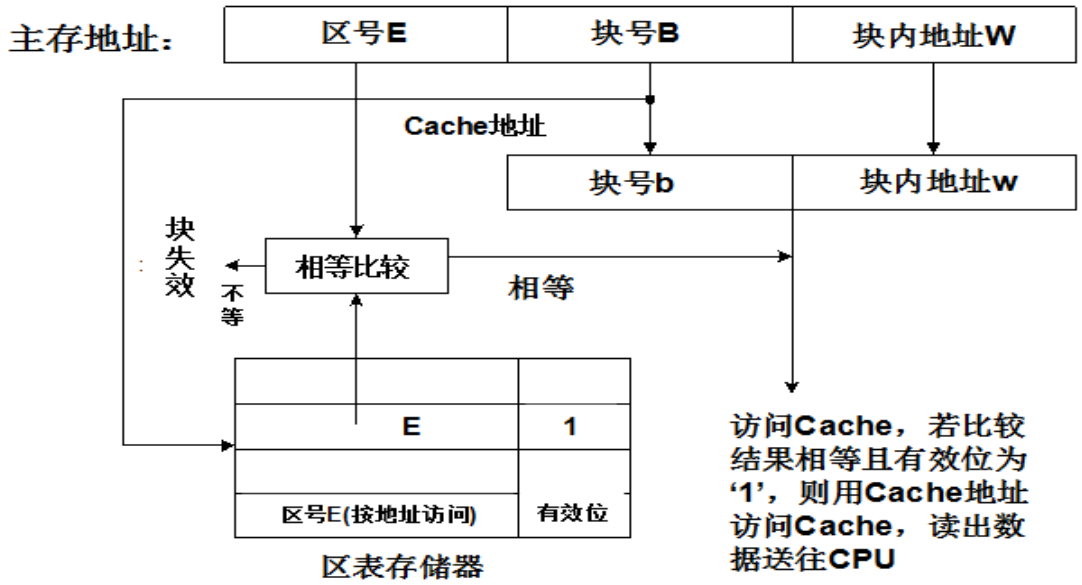


图 2 直接相联地址变换示意图

在程序执行过程中，当要访问 Cache 时，为了实现主存块号到 Cache 块号的变换，需要有一个存放主存区号的小容量存储器（称为区表存储器），这个存储器的容量与 Cache 的块数相等，字长为主存地址中区号 E 的长度，另外再加一个有效位（命中/失效）。

从主存地址到 Cache 地址的变换过程中，首先用主存地址中的块号 B 去访问区表存储器（用块号 B 作为区表存储器的地址，访问它），然后，将读出来的区号与主存地址中的区号 E 进行比较，比较结果相等，有效位为 1，则 Cache 命中，表示要访问的那一块已经装入到 Cache 中了，可以直接用块号及块内地址组成的缓冲地址到缓存 Cache 中取数，把读出来的

数据送往 CPU；如果比较结果不相等，有效位为 1，可以进行替换，如果有效位为 0，可以直接调入所需块。至于比较不相等情况，不论有效位是 1 或 0 均为 Cache 没有命中，或称为 Cache 失效，表示要访问的那个块还没有装入到 Cache 中，这时，要用主存地址去访问主存储器，先把该地址所在的块读到 Cache 中，然后再读取 Cache 中该地址的数据送 CPU。

Cache 和 CPU 以及存储器的关系如下图所示。

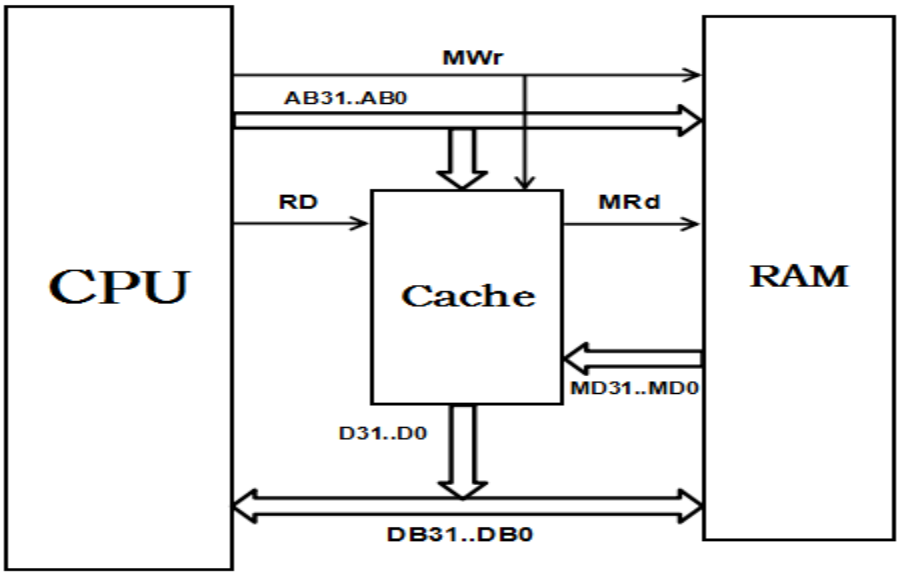


图 3　Cache 的基本框架图

如图 3 所示，32 位主存地址为 AB31..AB0（地址总线），32 位数据为 DB31..DB0（数据总线），RD（为 0，读）为 Cache 的读信号，MWr（为 1，写）为主存的写信号，MRd（为 0，读）为主存的读信号，D31..D0 为 Cache 送往 CPU 的数据信号（**出口处经过一个三态缓冲器然后再输出**），MD31..MD0 为存储器 RAM 送往 Cache 的数据信号。

如图 4 所示，区号 E 取 14 位，块号 B 为 14 位，块内地址为 4 位，这样 Cache 地址就是 18 位了，其中 Cache 块号 b 为 14 位，块内地址 w 为 4 位，所以 Cache 容量为 256KB(2^18)个单元，块号 b 取 14 位，那么 Cache 分为 16KB(2^14)块，块内地址 w 取 4 位，则每块为 16 个单元（每个单元一个字节）。

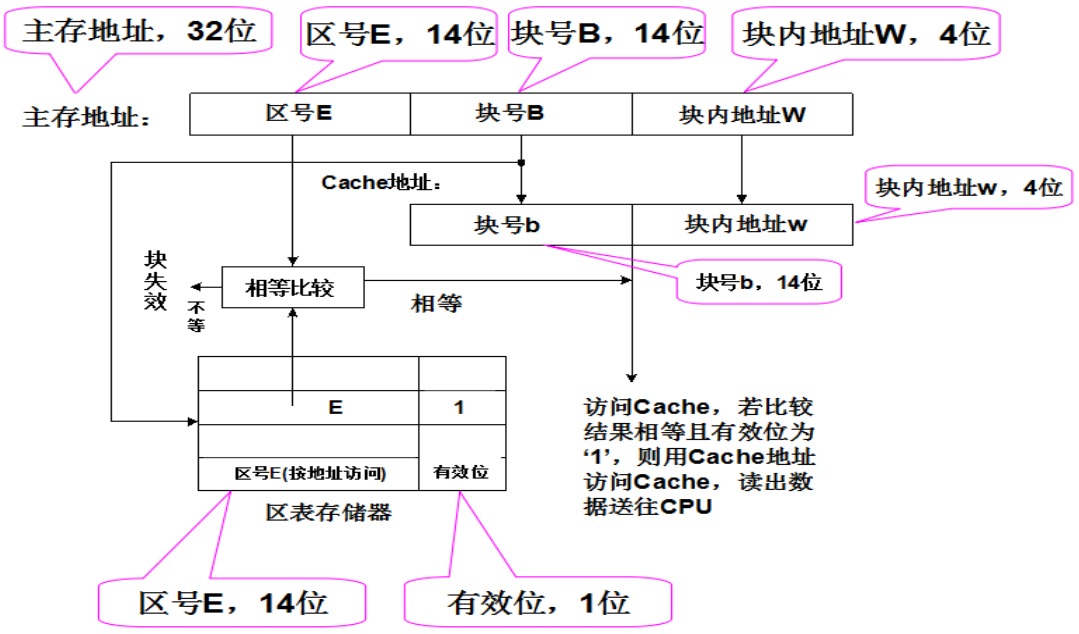


图 4　地址分段划分示意图

实现 Cache 的存储体的方法是先实现**一个 8 位的存储单元**，然后用这个 8 位的存储单元来构成一个 256Kb X 8 位的 Cache（地址 18 位）。

再实现一个 15 位（14+1）的存储单元，然后，用这个 15 位的存储单元来构成一个 16k X 15 位的区表存储器（**地址 14 位与块号 B 相同**），用来存放区号（14 位）和有效位 M(1 位)。在这个部分中，还要实现一个区号 E 比较器，也就是如果主存地址的区号 E 和区表存储器中按块号 B 为地址取出的相应单元中的区号 E 相等，有效位标志为 M，且有效位 M=1 时，则 Cache 命中，否则 Cache 失效，M=0 时表示 Cache 失效。

当 Cache 命中时，就将 Cache 存储体中相应单元的数据送往 CPU，这个过程比较简单。当 Cache 失效时，就将主存中**相应块**中的数据读出写入 Cache 中，这样 Cache 控制器就要产生主存储器的读信号 MRd（为 0，读），由于每个 Cache 块占十六个单元，**按 32 位（4 个字节）为访问存储器单位，那么需要连续访问 4 次主存，读取存储器中该块的数据，即 16 个字节，然后写入 Cache 相应块中，最后再修改区表存储器。至于访问主存的方法，要用到计数器。**写数据时，如果 Cache 中有该地址数据，则修改，然后修改存储器该地址内容（MWr 为 1，写，为主存的写信号）；如果 Cache 中无该地址数据，就直接修改存储器该地址单元内容。**读/写存储器时，要注意互锁情况。**

四. 实验器材

PC机一台，BASYS 2 实验板一块？，Xilinx ISE 开发软件一套。

五. 实验分析与设计

1. 实验步骤

（1） 阅读提供的资料，了解本次实验的内容。

（2） 对实验进行分析设计（见2. 分析设计）

（3） 编写各子模块代码

（4） 编写测试代码（见4.测试单周期CPU）

（5） debug（见心得和体会部分的1.实验过程中遇到的问题及解决的办法）

（6） 反思总结写报告

2. 分析设计

用模块化的思想方法设计。

查阅相关资料（Verilog模块概念和实例化[1]）可知，模块（module）是verilog最基本的概念，是v设计中的基本单元，每个v设计的系统中都由若干module组成。具体来说，模块有以下性质：

1） 模块在语言形式上是以关键词module开始，以关键词endmodule结束的一段程序。

2） 模块的实际意义是代表硬件电路上的逻辑实体。

3） 每个模块都实现特定的功能。

4） 模块的描述方式有行为建模和结构建模之分。

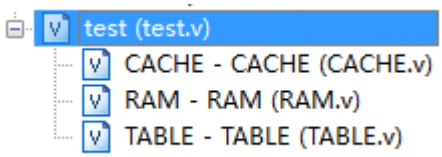
5） 模块之间是并行运行的。

6） 模块是分层的，高层模块通过调用、连接低层模块的实例来实现复杂的功能。

7） 各模块连接完成整个系统需要一个顶层模块（top-module）。

虽然Cache控制器不是一个很复杂的系统，但是划分成对应的功能模块进行设计能使得设计更加简明。因此我按照下面四个步骤进行设计：

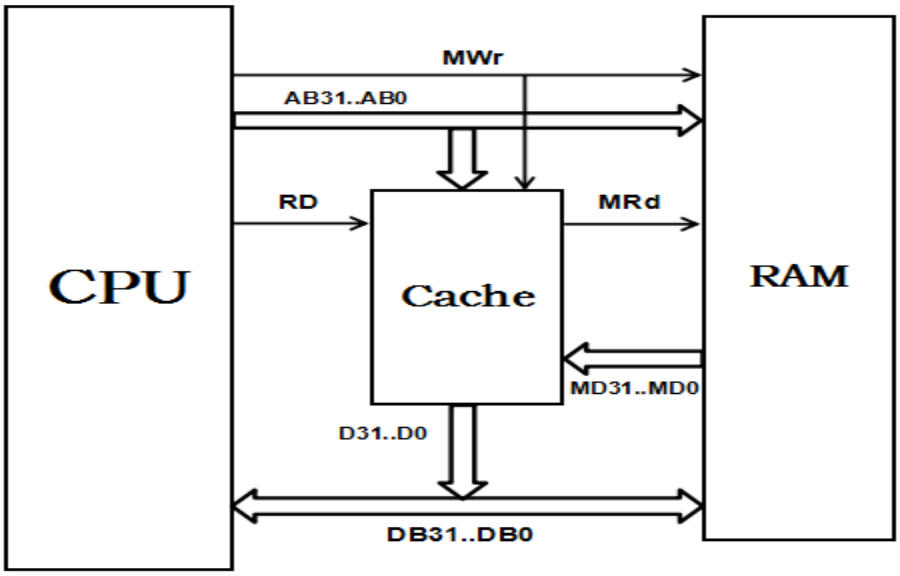
1) 把系统划分成三个子模块CACHE,RAM,TABLE，分别表示缓存、主存和区表存储器。



2) 规定各模块的接口；

各模块接口直接采用数据通路图中的名称，统一规范。

[1] http://jason0214.lofter.com/post/30cbe4_12a8f72



3) 进行子模块设计

a) CACHE的设计

在接收到clr清零信号时初始化；在接收到地址总线或主存的写信号后分情况讨论。

①写信号，命中

修改cache

②写信号，没有命中

没有操作

③读信号，命中

直接读Cache 的数据送CPU

④读信号，没有命中

从存储器读取该地址数据写入Cache，然后送CPU。Cache控制器就要产生主存储器的读信号MRd（为0，读），由于每个Cache块占十六个单元，按32 位（4个字节）为访问存储器单位，那么需要连续访问4次主存，读取存储器中该块的数据，即16个字节，然后写入Cache相应块中，最后再修改区表存储器。至于访问主存的方法，要用到计数器

```
always @ (AB or MWr)
   begin
      #1;
      // 写，没有命中，没有操作
      if (MWr == 1 && hit == 1) // 写，命中
         begin
            cache_memory[AB[17:0]] = DB[31:24];
            cache_memory[AB[17:0]+1] = DB[23:16];
            cache_memory[AB[17:0]+2] = DB[15:8];
            cache_memory[AB[17:0]+3] = DB[7:0];
         end
      else if (RD == 0 && hit == 1) // 读，命中
         begin
            // 当Cache命中时，就将Cache存储体中相应单元的数据送往CPU
            D = {cache_memory[AB[17:0]],cache_memory[AB[17:0]+1],
            cache_memory[AB[17:0]+2],cache_memory[AB[17:0]+3]};
         end
      else if (RD == 0 && hit == 0) // 读，没有命中
         begin
            // 当Cache失效时，就将主存中相应块中的数据读出写入Cache中，
            // 这样Cache控制器就要产生主存储器的读信号MRd（为0，读）
            MRd = 0;
         end
   end
```

b) RAM的设计

在接收到clr清零信号时初始化;在接收到地址总线或主存的写信号后写;在接收到时钟信号后判断是否需要对主存进行读操作。

c) TABLE的设计

在程序执行过程中，当要访问Cache时，为了实现主存块号到Cache块号的变换，需要有一个存放主存区号的小容量存储器（称为区表存储器），这个存储器的容量与Cache的块数相等，字长为主存地址中区号E的长度，另外再加一个有效位（命中/失效）。

```
always @ (AB or DB or MWr)
  begin
     // 主存地址的区号E和区表存储器中按块号B为地址取出的相应单元中的区号E相等，且有效位M=1时，则Cache命中
     if (table_memory[AB[17:4]][14:1] == AB[31:18] && table_memory[AB[17:4]][0] == 1)
        hit = 1;
     // 比较不相等情况，不论有效位是1或0均为Cache没有命中，或称为Cache失效，表示要访问的那个块还没有装入到Cache中
     else hit = 0;
     // 如果没有命中，则设置有效位为0
     if (table_memory[AB[17:4]][14:1] != AB[31:18] && table_memory[AB[17:4]][0] == 1)
        table_memory[AB[17:4]][0]=0;
  end
```

4) 子模块编程（见下文）

5) 连接各模块完成系统设计。

verilog是通过模块调用或称为模块实例化的方式来实现这些子模块与高层模块的连接。调用模块实例的一般形式为：〈模块名〉〈参数列表〉〈实例名〉（〈

端口列表>）；信号端口可通过名称关联即.PortName(port_expr)。要注意的是引用时要严格按照模块定义的端口顺序来连接，不用标明原模块定义时规定的端口名：

Design u_1(u_1的端口1，u_1的端口2，u_1的端口3，u_1的端口……)；

用’.’符号来标明原模块定义时规定的端口名：

```
Design u_2(.(端口1(u_1的端口1),
           .(端口2(u_1的端口2),
           .(端口3(u_1的端口3),
                  ……  );
```

各子模块是通过线连接起的，由于每根线两端连接两个模块，不为单独一个模块所有，我采用为衔接的线另外命名（下图橙色的线）。但控制单元模块输出的十个控制信号（下图蓝色）及输入的Opcode和zero,clk,rst等含义明显，不再单独命名。所有线都写做wire [size - 1:0] wireName;两个子模块的两个端口通过一根线连接，则分别写做.PortName1(port_expr)和.PortName2(port_expr)。由此就可以轻松表示各模块之间的连接关系了。

3. 编写各子模块代码

```
////////////////////////////////////////////////////////////////////////////////
// CACHE模块
module CACHE(
        input wire [31:0] AB, //  地址总线，32位主存地址
        input wire [31:0] DB, //  数据总线
        input wire MWr, //  主存的写1信号
        input wire RD, // Cache的读0信号
        input wire hit,
        input wire clk,
        input wire clr,
        input wire [31:0] MD, //  存储器RAM送往Cache的数据信号
        output reg [31:0] D, // Cache送CPU数据
        output reg MRd, //  主存的读0信号
        output reg WCT); // WCT为写Cache区表存储器信号

        // Cache块号b为14位，块内地址w为4位，所以Cache容量为256KB(2^18)个单元，
        //  块号b取14位，那么Cache分为16KB(2^14)块，块内地址w取4位，
        //  则每块为16个单元（每个单元一个字节）
        //  实现Cache的存储体的方法是先实现一个8位的存储单元，
```

```
        // 然后用这个8位的存储单元来构成一个256Kb X 8位的Cache（地址18位）
        reg [7:0] cache_memory[0:262143]; // 2^18bytes = 256Kbytes = 256*1024b = 262144b
        integer count;
        integer i;
        always @ (negedge clr)
            begin
                count = 0;
                for (i = 0; i < 262144; i = i + 1) cache_memory[i] = 0;
            end
        always @ (AB or MWr)
            begin
                #1;
                if (MWr == 1 && hit == 1) // 写，命中
                    begin
                        cache_memory[AB[17:0]] = DB[31:24];
                        cache_memory[AB[17:0]+1] = DB[23:16];
                        cache_memory[AB[17:0]+2] = DB[15:8];
                        cache_memory[AB[17:0]+3] = DB[7:0];
                    end
                else if (RD == 0 && hit == 1) // 读，命中
                    begin
                        // 当Cache命中时，就将Cache存储体中相应单元的数据送往CPU
                        D = {cache_memory[AB[17:0]],cache_memory[AB[17:0]+1],cache_memory[AB[17:0]+2],cache_memory[AB[17:0]+3]};
                    end
                else if (RD == 0 && hit == 0) // 读，没有命中
                    begin
                        // 当Cache失效时，就将主存中相应块中的数据读出写入Cache中，
                        // 这样Cache控制器就要产生主存储器的读信号MRd（为0，读）
                        MRd = 0;
                    end
            end
        always @ (posedge clk)
            begin
                WCT = 0;
                // Cache失效，表示要访问的那个块还没有装入到Cache中
                if (RD == 0 && hit == 0 && MRd == 0)
                    // 这时，要用主存地址去访问主存储器，先把该地址所在的块读到Cache中
```

```
                    begin
                    // 由于每个Cache块占十六个单元，按32 位（4个字节）
                    // 为访问存储器单位，那么需要连续访问4次主存
                    if (count < 4) // 计数器
                        // 读取存储器中该块的数据，即16个字节，然后写入Cache
相应块中
                        begin
                            #2;
                            cache_memory[{AB[17:4],4'h0}+count*4] = MD[31:24];
                            cache_memory[{AB[17:4],4'h0}+count*4+1] = MD[23:16];
                            cache_memory[{AB[17:4],4'h0}+count*4+2] = MD[15:8];
                            cache_memory[{AB[17:4],4'h0}+count*4+3] = MD[7:0];
                            count = count + 1;
                        end
                    else
                        // 然后再读取Cache中该地址的数据送CPU
                        begin
                            D                                                    =
{cache_memory[AB[17:0]],cache_memory[AB[17:0]+1],cache_memory[AB[17:0]+2],cache_
memory[AB[17:0]+3]};
                            count = 0;
                            MRd = 1;
                            WCT = 1; // 最后再修改区表存储器，发出写Cache区表
存储器信号
                        end
                end
            end

endmodule

////////////////////////////////////////////////////////////////////////////////
// RAM模块
module RAM(
        input wire [31:0] AB, // 地址总线，32位主存地址
        input wire [31:0] DB, // 数据总线
        input wire MWr, // 主存的写1信号
        input wire MRd, // 主存的读0信号
        input wire clk,
        input wire clr,
        output reg [31:0] MD); // 存储器RAM送往cache的数据信号

        reg [7:0] memory[0:1048575];
        integer count;
        integer i;
```

```
        always @ (negedge clr)
            begin
                count = 0;
                for(i = 0; i < 1048576; i = i + 1) memory[i] = 0;
            end
        always @ (AB or MWr)
            begin
                #1;
                if (MWr == 1) //  主存写
                    begin
                        memory[AB] = DB[31:24];
                        memory[AB+1] = DB[23:16];
                        memory[AB+2] = DB[15:8];
                        memory[AB+3] = DB[7:0];
                    end
            end
        always @ (posedge clk)
            begin
                #1;
                if (MRd == 0) //  主存读
                    begin
                        MD                                                                  =
{memory[{AB[31:4],4'h0}+count*4],memory[{AB[31:4],4'h0}+count*4+1],memory[{AB[31:4]
,4'h0}+count*4+2],memory[{AB[31:4],4'h0}+count*4+3]};
                        count = count + 1;
                        if (count == 4) count = 0;
                    end
                else count = 0;
            end


endmodule
//////////////////////////////////////////////////////////////////////////////////
// TABLE模块
module TABLE(
    input wire [31:0] AB,
     input wire [31:0] DB,
     input wire clr,
     input wire WCT, // WCT为写Cache区表存储器信号
     input wire MWr, //  主存的写1信号
     output reg hit);

     reg [14:0] table_memory[0:16383]; // 16k
     integer i;
```

```
    always @ (negedge clr)
        begin
            for (i = 0; i < 16384; i = i + 1) table_memory[i] = 0;
        end
    always @ (posedge WCT)
        begin
            table_memory[AB[17:4]] = {AB[31:18],1'b1};
        end
    always @ (AB or DB or MWr)
        begin
            // 主存地址的区号E和区表存储器中按块号B为地址取出的相应单元中的区号E相等，且有效位M=1时，则Cache命中
            if (table_memory[AB[17:4]][14:1] == AB[31:18] && table_memory[AB[17:4]][0] == 1) hit = 1;
            // 比较不相等情况，不论有效位是1或0均为Cache没有命中，或称为Cache失效，表示要访问的那个块还没有装入到Cache中
            else hit = 0;
            // 如果没有命中，则设置有效位为0
            if (table_memory[AB[17:4]][14:1] != AB[31:18] && table_memory[AB[17:4]][0] == 1) table_memory[AB[17:4]][0]=0;
        end

endmodule
//////////////////////////////////////////////////////////////////////////////////
```

4. 编写测试代码

```
module test;

    // Inputs
    reg clk; // 系统时钟（用于计数器计数控制等操作）
    reg clr; // 系统总清零信号（清区表存储器、计数器）
    reg [31:0] AB; // CPU访问内存的地址（地址总线）
    reg [31:0] DB; // 32位数据为DB31..DB0（数据总线）

    reg MWr; // MWr（为1，写）为主存的写信号
    reg RD; // RD（为0，读）为Cache的读信号
    wire hit; // 是否命中标志
    wire [31:0] MD; // 主存送Cache的数据
    wire [31:0] D; // Cache送CPU数据
    wire MRd; // MRd（为0，读）为主存的读信号
    wire WCT; // 写Cache区表存储器信号

    // Instantiate the Unit Under Test (UUT)
```

```
CACHE CACHE (
        .AB(AB),
        .DB(DB),
        .MWr(MWr),
        .RD(RD),
        .hit(hit),
        .clk(clk),
        .clr(clr),
        .MD(MD),
        .D(D),
        .MRd(MRd),
        .WCT(WCT)
);
RAM RAM (
      .AB(AB),
        .DB(DB),
        .MWr(MWr),
        .MRd(MRd),
        .clk(clk),
        .clr(clr),
        .MD(MD)
);
TABLE TABLE (
        .AB(AB),
        .DB(DB),
        .clr(clr),
        .WCT(WCT),
        .MWr(MWr),
        .hit(hit)
);

// （一）读Cache的情况，
// 1、Cache 中有该地址数据，直接读Cache 的数据送CPU。
// 2、Cache 中无该地址数据，则必须从存储器读取该地址数据写入Cache，然后送CPU。

// （二）修改Cache和存储器
// 如果Cache中有该地址块数据，则修改（没有就没有的改了），同时修改存储器该地址单元数据。


initial begin
        // Initialize Inputs
    clk = 0;
```

```
clr = 1;
// Add stimulus here
#10;
clr = 0;
#10;

// 写数据10（十进制，占四个字节，四个单元）， Cache为空，没有该地址块数据，只修改存储器该地址单元数据
// 存储器中从0x1开始存放，占4个单元
AB = 32'b00000000000000000000000000000001;
DB = 10;
RD = 1;
MWr = 1;
#50;

// 写数据20， Cache为空，没有该地址块数据，只修改存储器该地址单元数据
AB = 32'b00000000000000000000000000010010;
DB = 20;
RD  = 1;
MWr = 1;
#50;

// 写数据30， Cache为空，没有该地址块数据，只修改存储器该地址单元数据
AB = 32'b00000000000001000000000000110000;  // 0x40030
DB = 30;
RD = 1;
MWr = 1;
#50;

// 读数据，Cache为空，无该地址数据，则必须从存储器读取该地址数据写入Cache，然后送CPU
// AB = 32'b00000000000001（14位，区号，在最低位加一位有效位1，得十五位二进制数11，存到区表存储器的根据块号找到的地址单元）00000000000011（14位块号）0000（4位，块内地址）从存储器读取数据30写入cache的第3块的第0-3个单元（即110000地址）
AB = 32'b00000000000001000000000000110000;
RD = 0;
MWr = 0;
#50;

// 读数据，Cache为空，无该地址数据，则必须从存储器读取该地址数据写入Cache，然后送CPU
// AB = 32'b00000000000000（14位，区号，在最低位加一位有效位1，得十五位二进制数1，存到区表存储器的根据块号找到的地址单元）00000000000001（14位块号）0010
```

（4位，块内地址）从存储器读取数据20写入cache的第1块的第2-5个单元（即10010地址）

```
        AB = 32'b00000000000000000000000000010010;
        RD   = 0;
        MWr = 0;
        #50;

        // 写数据40， Cache中有该地址块数据，修改cache，同时修改存储器该地址单元数据。
        AB = 32'b00000000000001000000000000110000;
        DB = 40;
        RD = 1;
        MWr = 1;
        #50;

        // 写数据50， Cache中没有该地址块数据，没有命中，只修改存储器该地址单元数据
        // 块号11和前面相同，但此时Cache中block3已经不再对应地址0x000c0034所属的block，所以区表存储器需要将block3所对应的标志位设为0
        AB = 32'b00000000000011000000000000110100;
        DB = 50;
        MWr = 1;
        RD = 1;
        #50;

        // 读数据，Cache 中无该地址数据，则必须从存储器读取该地址数据写入Cache，然后送CPU。
        AB = 32'b00000000000011000000000000110100;   //read
        MWr = 0;
        RD = 0;
        #50;

        // 读数据，Cache 中有该地址数据，直接读Cache 的数据送CPU。
        AB = 32'b00000000000000000000000000010010;   //read
        RD = 0;
        MWr = 0;
        #50;

        #50;
    end
    always #5 clk=~clk;
endmodule
```

编号	CPU 访问内存的地址 AB	数据总线 DB	描述	区号 E（14）	块号 B(14)	块内地址 W(4)	Hit
1	0x00000001	10	写数据 10（十进制，占四个字节，四个单元）， Cache 为空，没有该地址块数据，只修改存储器该地址单元数据	00000000000000	00000000000000	0001	0
2	0x00000012	20	写数据 20， Cache 为空，没有该地址块数据，只修改存储器该地址单元数据	00000000000000	00000000000001	0010	0
3	0x00040030	30	写数据 30， Cache 为空，没有该地址块数据，只修改存储器该地址单元数据	00000000000001	00000000000011	0000	0
4	0x00040030		读数据 30，Cache 为空，无该地址数据，则必须从存储器读取该地址数据写入 Cache，然后送 CPU	00000000000001	00000000000011	0000	0
5	0x00000012		读数据 20，Cache 为空，无该地址数据，则必须从存储器读取该地址数据写入 Cache，然后送 CPU	00000000000000	00000000000001	0010	0
6	0x00040030	40	写数据 40，覆盖原数据 30. Cache 中有该地址块数据，修改 cache，同时修改存储器该地址单元数据。	00000000000001	00000000000011	0000	1
7	0x000C0034	50	写数据 50， Cache 中没有该地址块数据，没有命中，只修改存储器该地址单元数据	00000000000011	00000000000011	0100	0
8	0x000C0034		读数据 50，Cache 中无该地址数据，则必须从存储器读取该地址数据写入 Cache，然后送 CPU。	00000000000011	00000000000011	0100	0
9	0x00000012		读数据 20，Cache 中有该地址数据，直接读 Cache 的数据送 CPU。	00000000000000	00000000000001	0010	1

5. 实验结果及分析

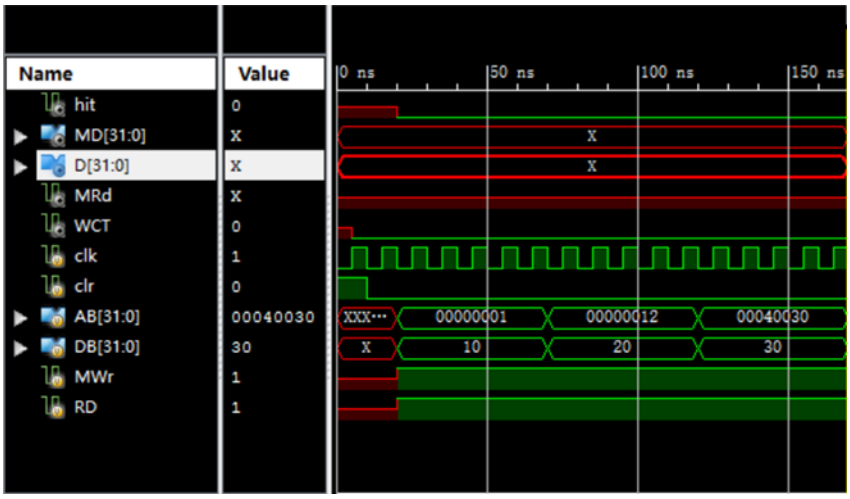
运行Xilinx ISE进行仿真，设置断点，看波形和中间结果。

（1） 前三条指令向主存写数据，Cache都为空，没有相应地址块数据，只修改存储器该地址单元数据。

```
// 写数据10（十进制，占四个字节，四个单元），
// Cache为空，没有该地址块数据，只修改存储器该地址单元数据
// 存储器中从0x1开始存放，占4个单元
AB = 32'b00000000000000000000000000000001;
DB = 10;
RD = 1;
MWr = 1;
#50;

// 写数据20， Cache为空，没有该地址块数据，只修改存储器该地址单元数据
AB = 32'b00000000000000000000000000010010;
DB = 20;
RD  = 1;
MWr = 1;
#50;

// 写数据30， Cache为空，没有该地址块数据，只修改存储器该地址单元数据
AB = 32'b00000000000001000000000000110000;  // 0x40030
DB = 30;
RD = 1;
MWr = 1;
#50;
```



RAM：

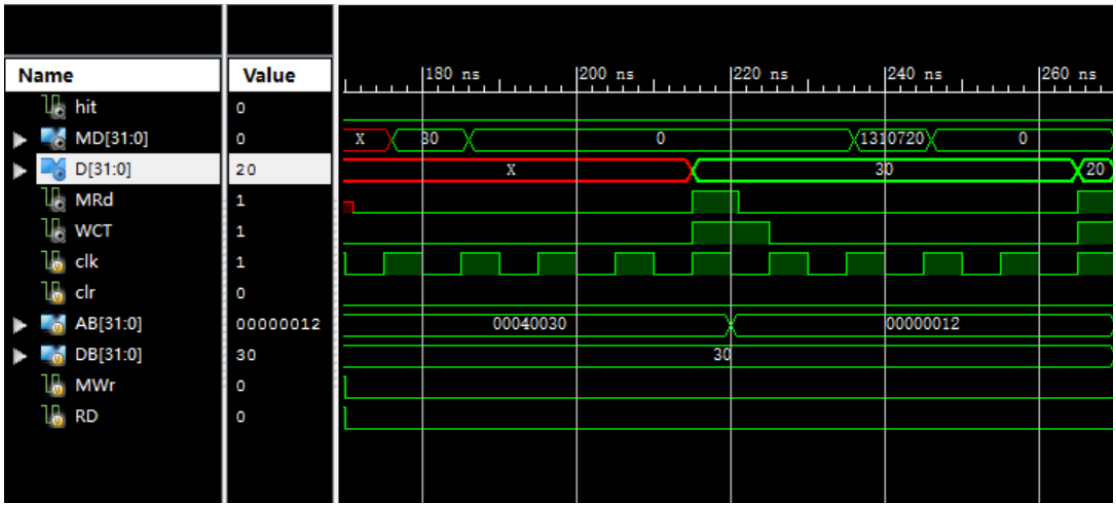
	0	1	2	3	4	5	6	7
0x0	0	0	0	0	10	0	0	0
0x8	0	0	0	0	0	0	0	0
0x10	0	0	0	0	0	20	0	0
0x40030	0	0	0	30	0	0	0	0
0x40038	0	0	0	0	0	0	0	0

CACHE和TABLE为空

（2） 接下来的两条（编号4,5）指令读数据，但Cache为空，无该地址数据，则必须从存储器读取该地址数据写入Cache，然后送CPU

```
// 读数据，Cache为空，无该地址数据，则必须从存储器读取该地址数据写入Cache，然后送CPU
// AB = 32'b00000000000001（14位，区号，在最低位加一位有效位1，得十五位二进制数11，存到区表存储器的根据块号找到的地址单元
// 00000000000011（14位块号）0000（4位，块内地址）从存储器读取数据30写入cache的第3块的第0-3个单元（即110000地址）
AB = 32'b00000000000001000000000000110000;
RD = 0;
MWr = 0;
#50;

// 读数据，Cache为空，无该地址数据，则必须从存储器读取该地址数据写入Cache，然后送CPU
// AB = 32'b00000000000000（14位，区号，在最低位加一位有效位1，得十五位二进制数1，存到区表存储器的根据块号找到的地址单元）
// 00000000000001（14位块号）0010（4位，块内地址）从存储器读取数据20写入cache的第1块的第2-5个单元（即10010地址）
AB = 32'b00000000000000000000000000010010;
RD  = 0;
MWr = 0;
#50;
```



读数据30：

AB = 32’b00000000000001（14位，区号，在最低位加一位有效位1，得十五位二进制数11，存到区表存储器的根据块号找到的地址单元）

00000000000011（14位块号）

0000（4位，块内地址）从存储器读取数据30写入cache的第3块的第0-3个单元（即110000地址）

读数据20：

AB = 32’b00000000000000（14位，区号，在最低位加一位有效位1，得十五位二进制数1，存到区表存储器的根据块号找到的地址单元）

00000000000001（14位块号）0010（4位，块内地址）从存储器读取数据20写入cache的第1块的第2-5个单元（即10010地址）

RAM:

0x40030	0	0	0	30	0	0	0	0
0x40038	0	0	0	0	0	0	0	0

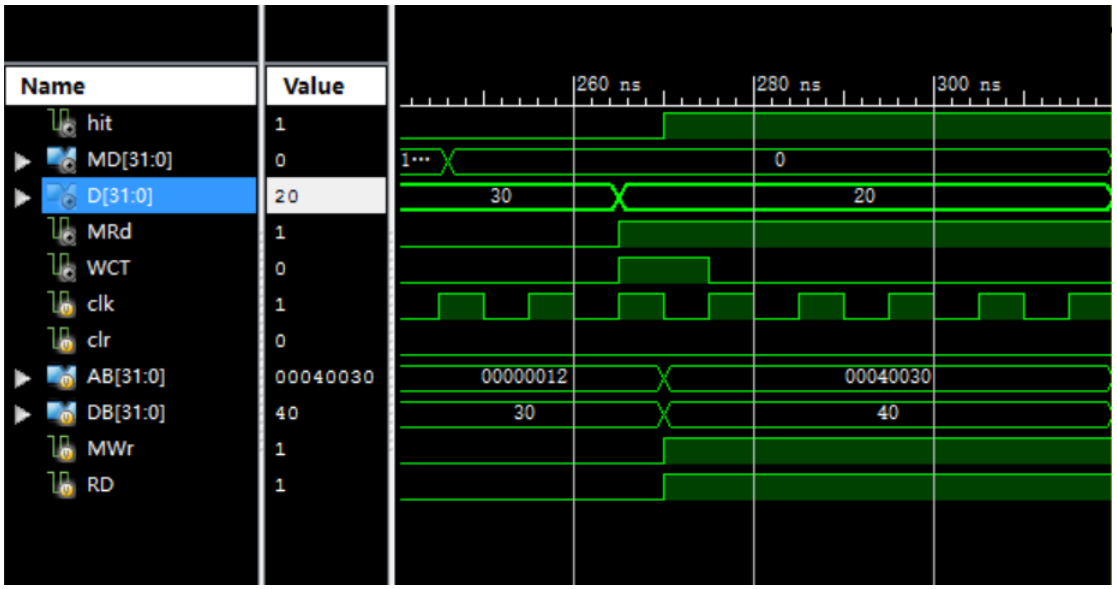
CACHE:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0x0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0
0x20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x30	0	0	0	30	0	0	0	0	0	0	0	0	0	0	0	0

TABLE:

	0	1	2	3
0x0	00000000000000	00000000000001	00000000000000	00000000000011

（3） 第六条指令写数据，地址0x00040030已被写过，因此命中。Cache中有该地址块数据，修改cache，同时修改存储器该地址单元数据。



新数据40覆盖原数据30

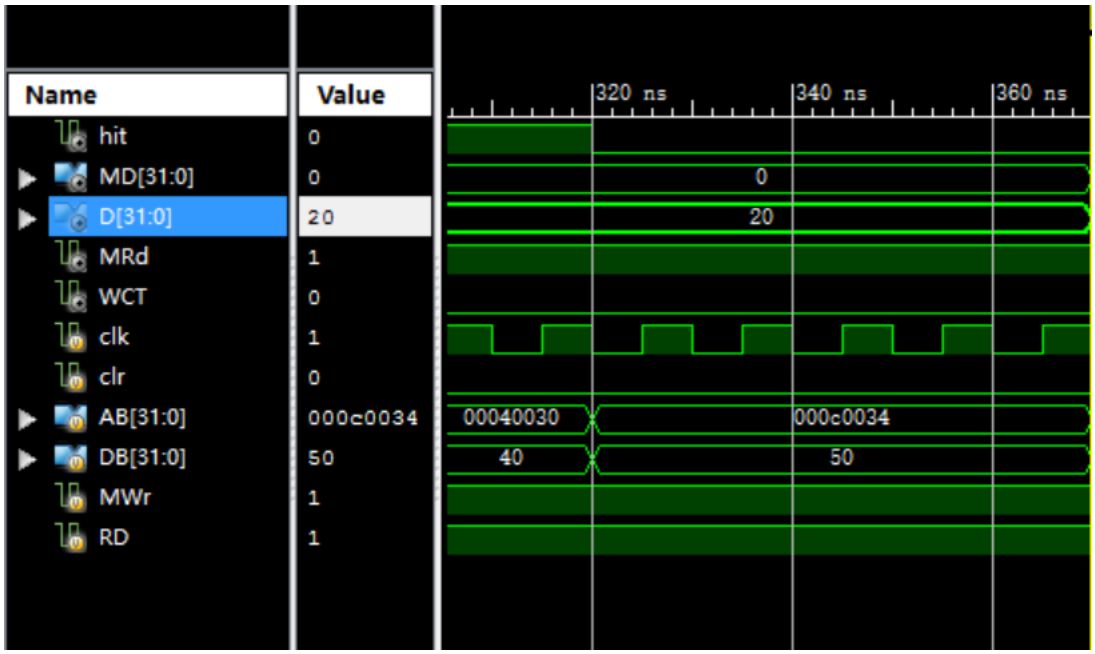
RAM:

0x40030	0	0	0	40	0	0	0	0
0x40038	0	0	0	0	0	0	0	0

CACHE:

0x30	0	0	0	40	0	0	0	0
0x38	0	0	0	0	0	0	0	0

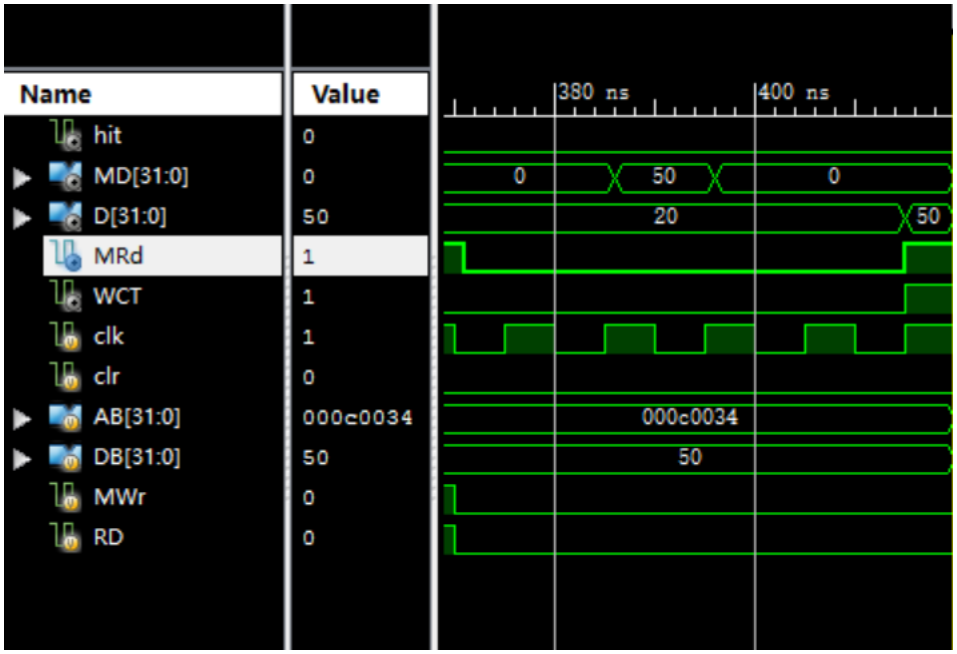
（4） 第七条指令写数据50，地址0x000C0034未被写过，Cache中没有该地址块数据，没有命中，只修改存储器该地址单元数据



RAM:

0xC0030	0	0	0	0	0	0	0	50
0xC0038	0	0	0	0	0	0	0	0

（5） 第八条指令读数据，Cache 中无该地址数据，则必须从存储器读取该地址数据写入Cache，然后送CPU。



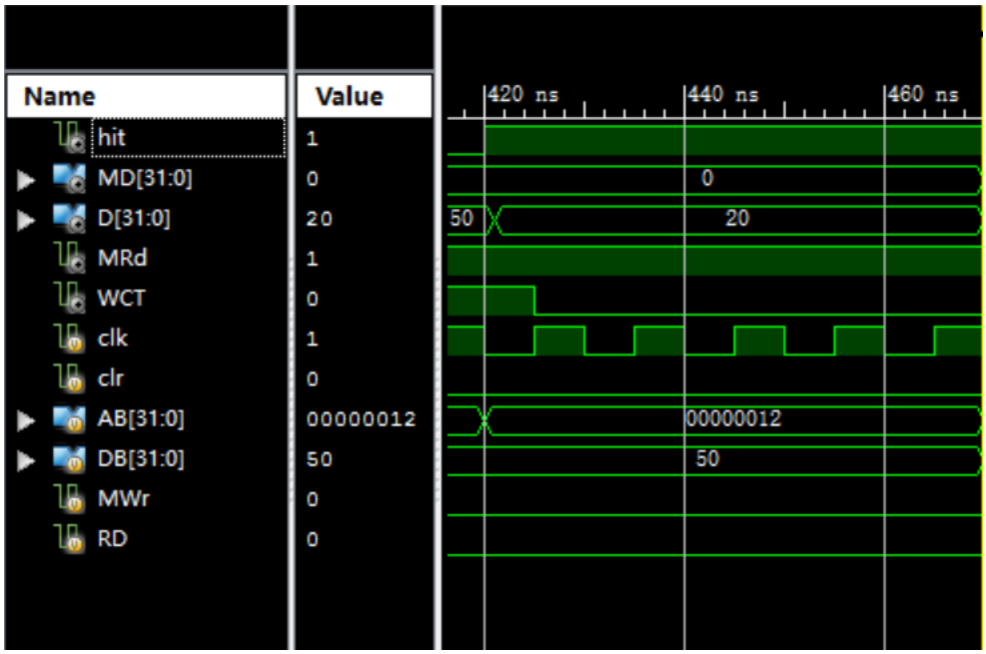
CACHE:

0x30	0	0	0	0	0	0	0	50
0x38	0	0	0	0	0	0	0	0

TABLE:

	0	1	2	3
0x0	000000000000000	000000000000001	000000000000000	000000000000111

（6） 第九条指令读数据，Cache 中有该地址数据，直接读Cache 的数据送CPU。



CACHE:

0x10	0	0	0	0	0	20	0	0
0x18	0	0	0	0	0	0	0	0

通过以上自己编写的测试代码执行的结果，可以判断设计实现的CACHE控制器是正确的。实验结果均和期望一致。

六. 实验心得

1. 实验过程中遇到的问题及解决的办法

控制信号的改变并发出和控制信号的接受并进行判断操作都由时钟信号触发并同时执行，这会导致接收到的控制信号是变化前一刻的旧信号，解决方法是为控制信号的接受加入一个极小的延时#1，达到接受新改变的控制信号的效果。

```
always @ (AB or MWr)
   begin
      #1;
      // 写，没有命中，没有操作
      if (MWr == 1 && hit == 1) // 写
         begin
            cache_memory[AB[17:0]] =
            cache_memory[AB[17:0]+1]
            cache_memory[AB[17:0]+2]
            cache_memory[AB[17:0]+3]
         end
      else if (RD == 0 && hit == 1) /
         begin
```

2. 体会

本次实验是本学期的最后一次计组实验，临近期末，所学渐成体系，渐趋完备，看着实验要求和思考设计方法时感到更易理解和入手。CACHE控制器的设计的知识很容易和课本联系起来，虽说看第一遍时思路不清晰，再认真看了一遍就理解了，脑中也有了大致的框架。这也让我加深了对“书读百遍其义自见”这句话的理解。

理论课上学习了缓存是什么，有什么作用，如何控制等等，也有了大体的印象，但是落实到动手实现就不一样了。需要考虑到许多细节上的设计和实现，是更具有挑战性的。如果没有对理论知识真正地吃透，也是不能完成实验的。

这次的测试方法和以往略有不同，采用了断点测试的方法，这有利于看中间过程存储器的变化，直观且利于操作。

多次试验，老师都给出了详细的设计指导文档和参考资料，非常有助于学生的理解和掌握，很多话直接起到了类似伪代码的作用，只要认真多读多领悟，不难落实到自己的代码实现上。同时我也反思如果老师没有给出详细的指导，而是给出实验要求并给以一定的思路，我能否学习探寻到类似的思想方法。例如CACHE控制器的设计中，对于CACHE模块，需要根据控制信号考虑读、写情况下是否命中，分情况讨论，并作出相应更改。如果没有给出设计图、控制信号的名称和对应功能，我该从何入手？因此我想这学期实验学到的最多的是关于设计思路和方法，例如贯穿几次试验的重要设计方法是模块化设计，控制信号的运用也贯穿始终。如何分析问题、解决问题，这些设计方法给我很多启发。

非常感谢一学期老师和TA的辛勤付出，你们的严格要求和细心检查是对学生更好的督促和推动。