Lab6 综合实验

PB17111623 范睿

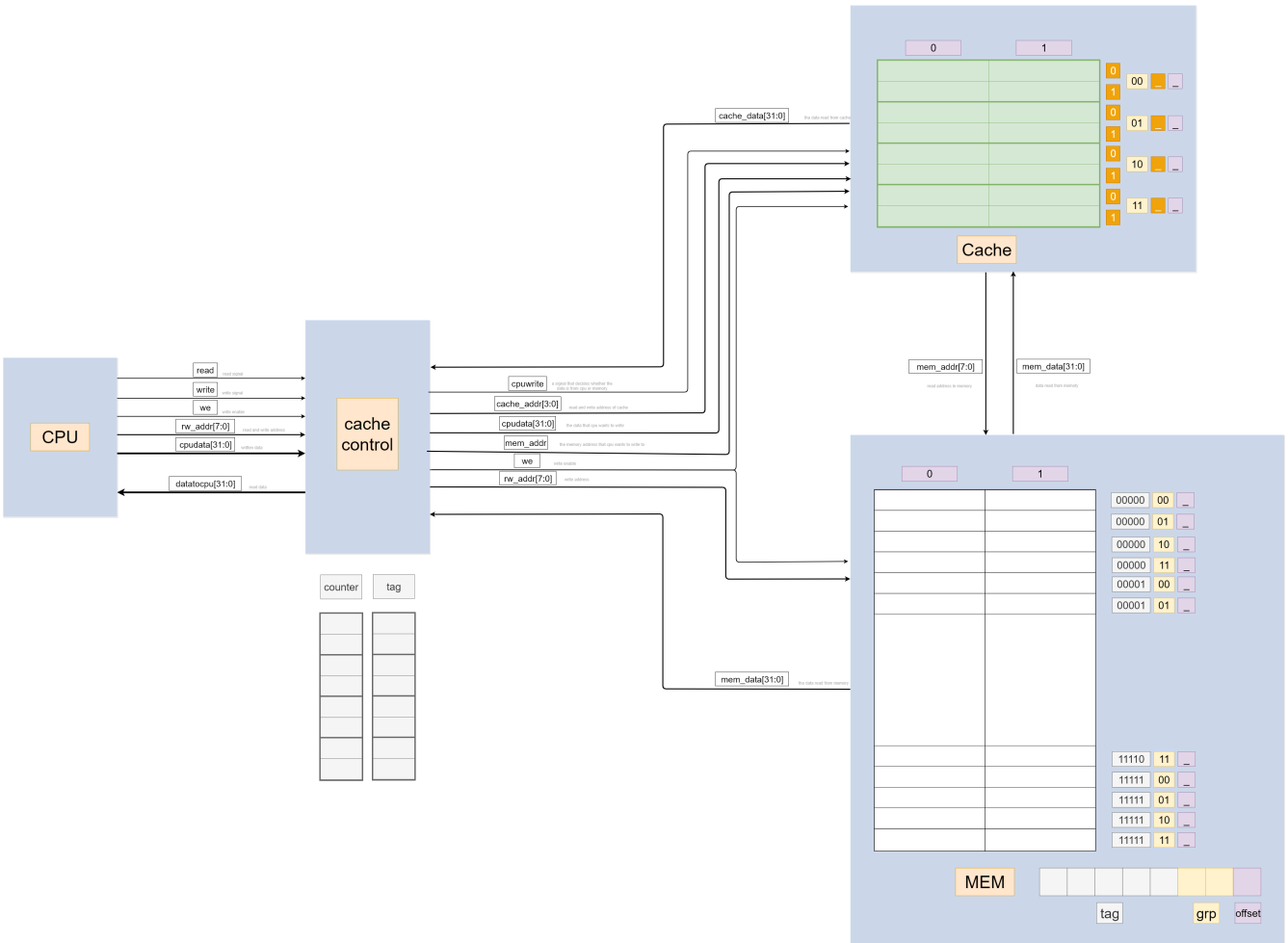
实现内容

- **CPU改进：** 增加Cache
- **连接外设：** VGA

逻辑设计

Cache

Cache结构

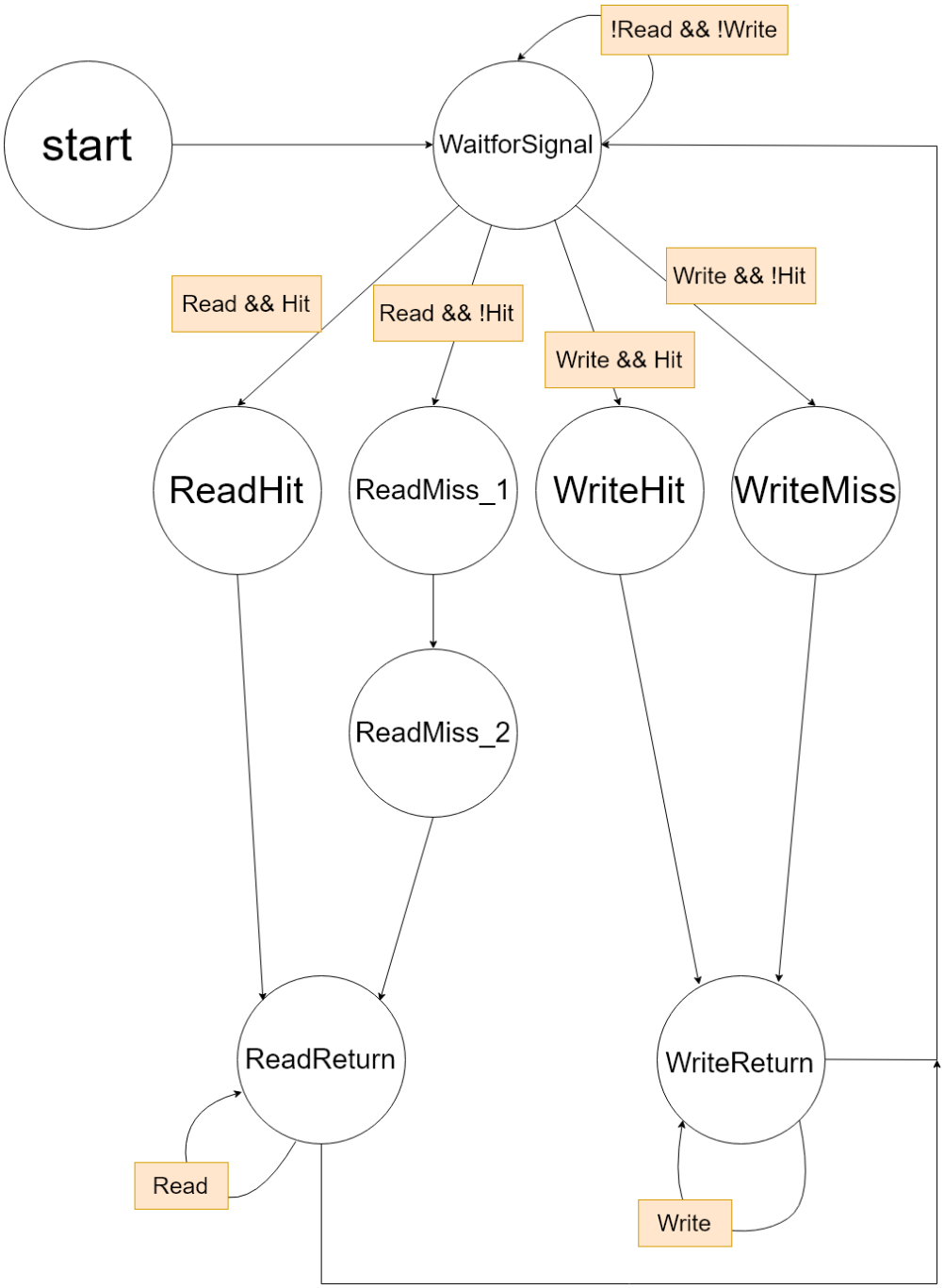


Cache结构如上图所示。

- **CPU：** 为lab5中的CPU，没有任何改动
- **Cache Control：** Cache控制器，根据cpu传进来的信号做出响应，控制下一级模块的行为
- **Cache：** 真正存储数据的Cache模块

- **MEM：** 内存

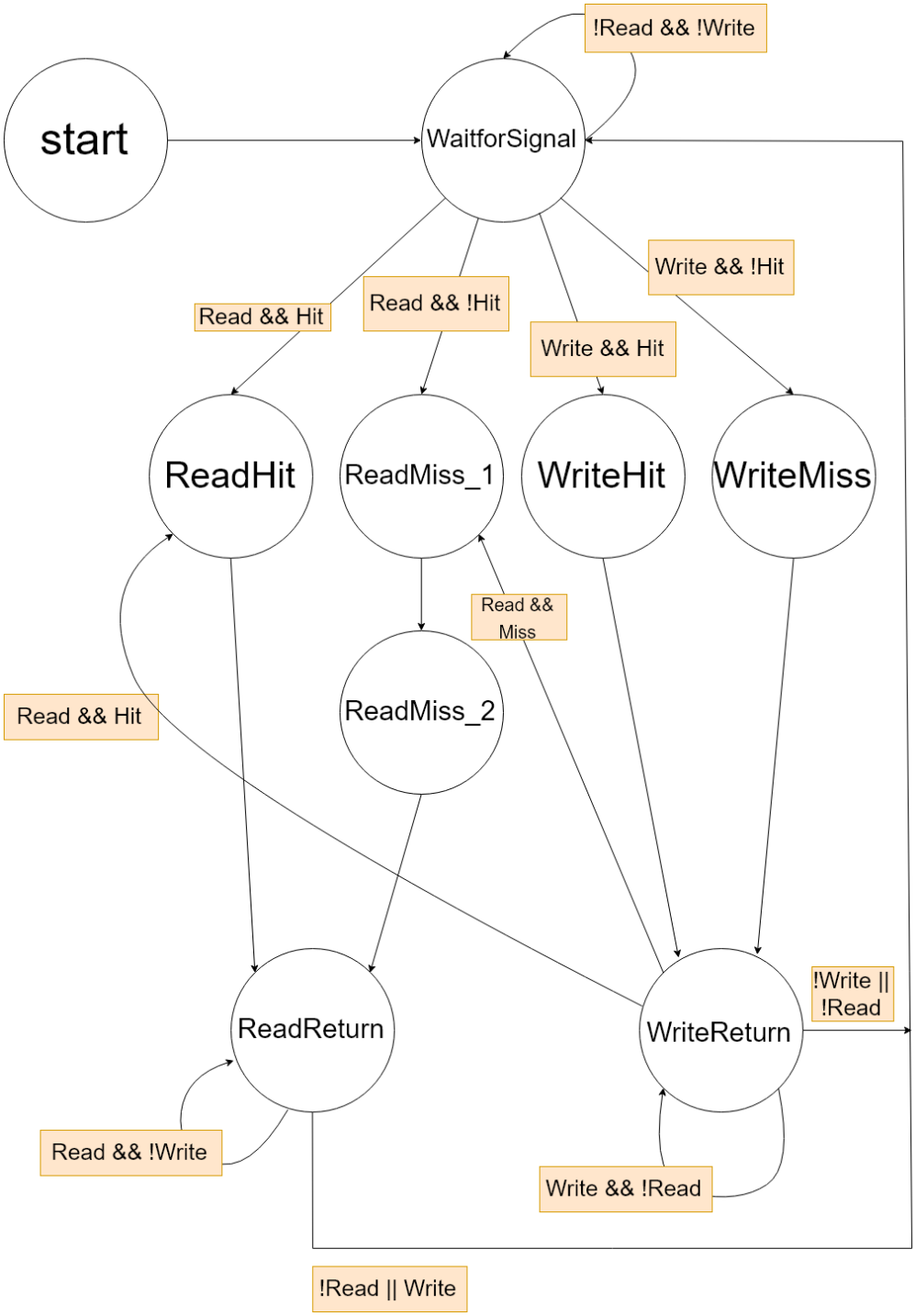
Cache状态机



说明1：ReadMiss有两个状态，是因为一个块中有8个字节，对应两个内存地址的位置，所以如果读失效需要复制两个数据过来，一个状态读进来一个，所以有两个状态。

说明2：设置了两个返回等待状态ReadReturn和WriteReturn，是为了防止有可能发生的循环读写（因为cache的时钟比cpu的快，所以有可能读或写完回到WaitforSignal时读或写信号仍然有效，这样会继续进入读或写状态，可能会发生错误）。

说明3：这个状态机中没有充分考虑读写谁优先级更高。在WaitforSignal中若读写同时有效会先进入写的状态，但是如果在WriteReturn等待时读信号突然有效，就无法读。之所以不考虑优先级问题是因为在lab5的MIPSCPU中不会出现读写信号同时有效的问题，所以无需考虑。**但是**当我在完成vga的代码时，发现我的设计有可能会使读信号在写信号仍有效期间突然有效，而且迅速失效，使得无法读取数据。因此我在vga的代码中稍微更改了我的状态机如下：



在WriteReturn状态中，若Read有效，根据是否命中去到Read的状态；若Read不有效但Write仍有效，在WriteReturn状态等待；若Read和Write都失效，回到WaitforSignal。

在ReadReturn中，若Write有效或Read失效，立即回到WaitforSignal，否在ReadReturn中等待。

写策略：写直达 - 不按写分配：

- 命中：写cache写主存
- 失效：只写主存

替换策略：LRU

- 被访问的块计数器**置0**，其他块的计数器**增加1**，替换时选择计数值**最大**的块

VGA：实现写内存功能

功能介绍

我想实现的功能是在屏幕上显示一行可编辑的代码，可以根据上下左右键确定此条代码的opcode（我只实现了ADD,LW,SW）和源寄存器，目标寄存器，和立即数。按下确定键可以将此条已编辑好的代码写进内存，同时也显示在屏幕上。

显示区域

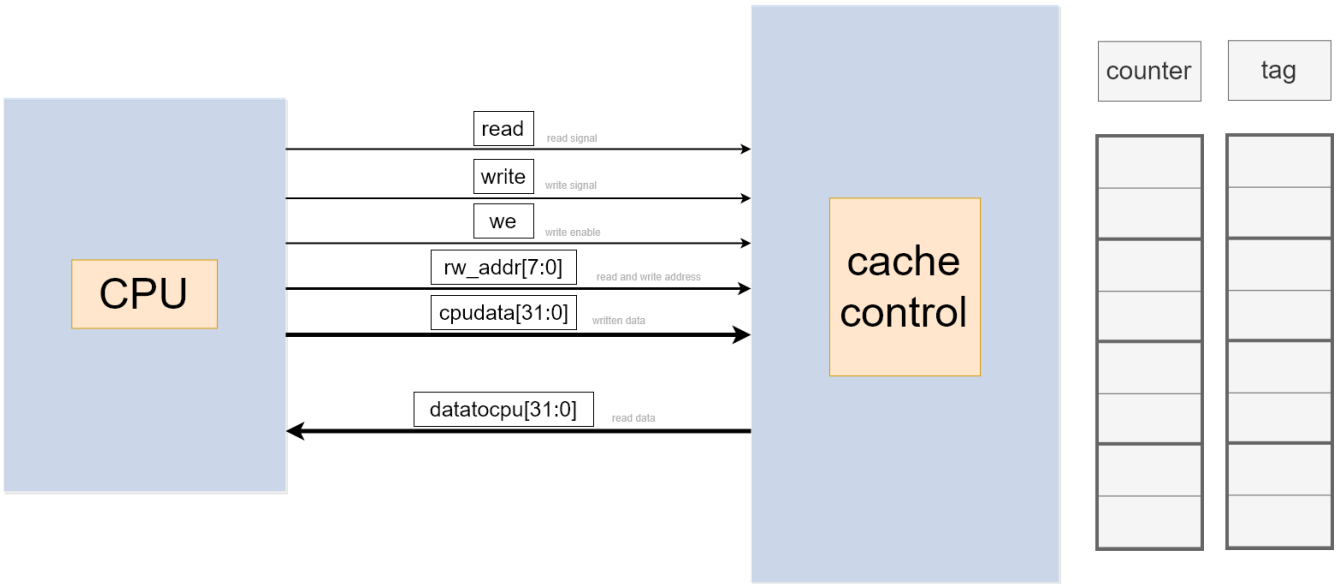
显示区域为一个80x160大小的区域。其中每行每列以8像素为单位进行分割，每8x8大小的方框中显示一个字符。所以此区域中最多可以显示10*20个字符。

字符显示

开辟存储字模的模块，其中存放着需要用到的字符在8x8的区域中的排列方式。每扫描到一个像素，读取这个像素所在的8x8的区域内应该显示什么字符，在从字模模块中读取该字符的排列方式；再根据当前扫描的位置计算x方向和y方向的偏移量，确定当前像素应该是什么颜色。

数据通路

Cache Control



CPU传输给Cache控制器的信号和数据：

- read：读信号
- write：写信号
- rw_addr：读写地址
- cpudata：写数据

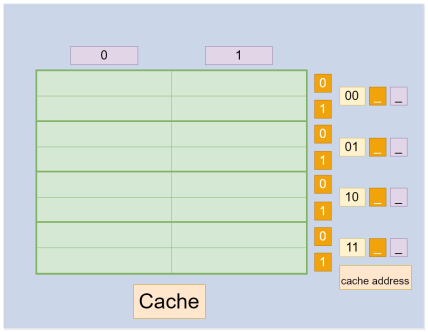
Cache控制器传输给CPU的数据：

- datatocpu：从Cache中读取到的数据
- （写操作Cache控制器不需要传信号给CPU）

Cache Control的两个寄存器：

- tag：大小为8*6，8个行分别对应Cache中8个块；每个6位数据中，最高位记录此块是否被分配（0未分配，1被分配），低五位记录此块数据来源的内存地址的高五位
- counter，8行同样对应Cache中8个块，没行记录此块未被使用次数，在LRU替换时参考

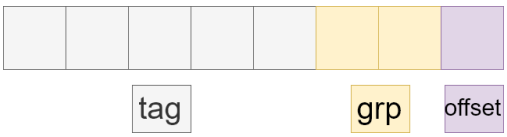
Cache



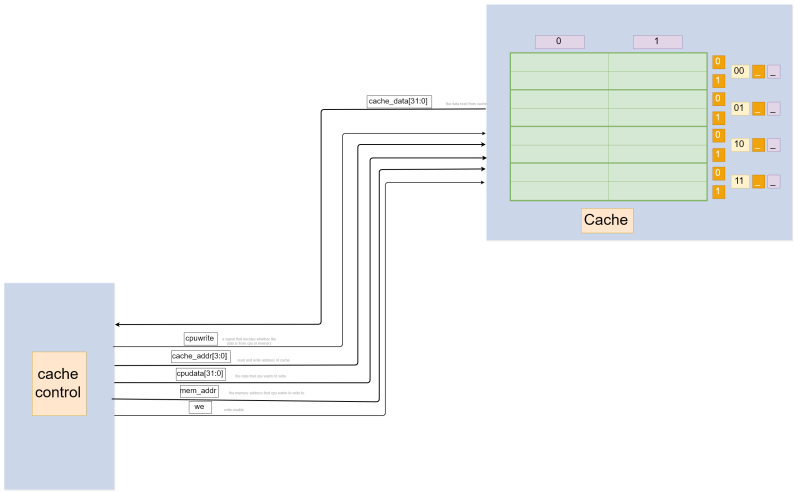
Cache结构如上图所示。

我实现的Cache以组相连的形式与内存对应。

一个Cache中有4组，每组中有2块，每块占8字节。



一个内存地址如上图，占8位。其中第1，2位为组号（黄色），与Cache地址在高两位（黄色）相同，表示此地址映射到Cache中哪个组。内存地址中高五位存放在Cache Control中的tag里。至于存放在tag中的哪个位置，需要根据tag中的内容判断。如果同一组里的第一个块没有被分配，则放在第一个块中；若被分配了且第二个块没被分配，则放在第二个块中；若都被分配了，则利用**LRU**替换算法将它替换进去。内存地址中的最后一位是offset，表示此内存地址中的数据放在块中的哪个位置。一次读取以块为单位。



Cache Control与Cache之间的数据通路如上图所示。

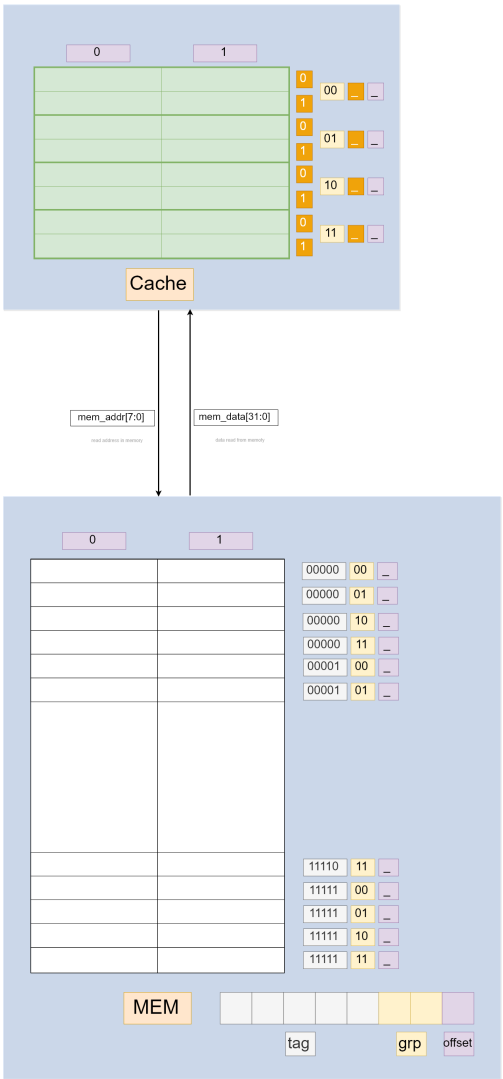
Cache Control传输给Cache的数据与信号：

- cpuwrite：写的数据是否来源于CPU（1为来源于cpu，0位来源于内存）
- cache_addr：读写的cache地址
- cpudata：cpu写的数据
- mem_addr：读写的内存地址（若此地址的数据不在cache中时使用）

- we：写使能

cache传输给Cache Control的数据：

- cache_data：cache中读到的数据

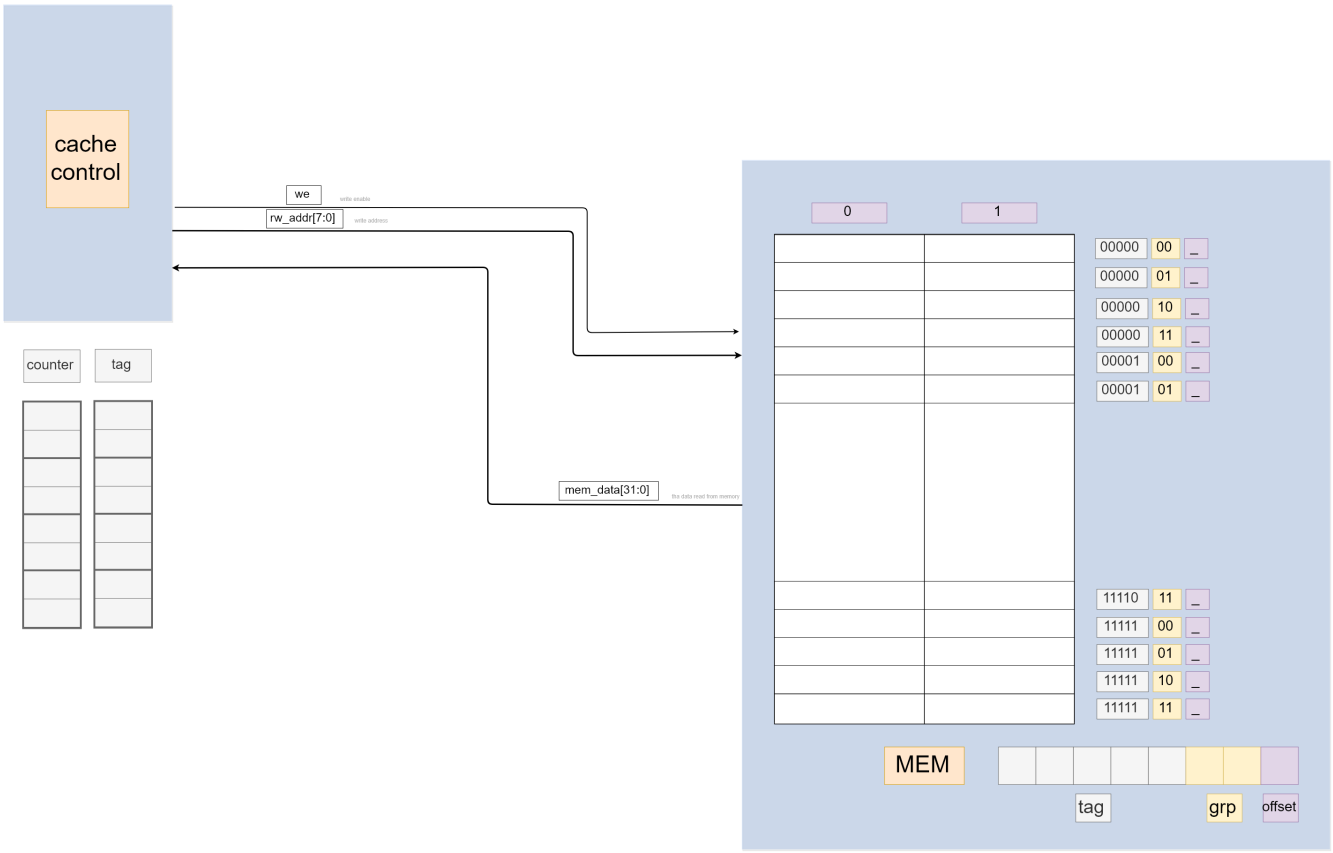


Cache与内存的数据通路如上图所示。

当发现当前访问的内存地址的数据不在Cache中时，Cache直接从内存中读取mem_addr处的数据放到Cache指定位置中。

MEM

内存仍为256*32的大小。



MEM与Cache Control的数据通路如上图所示。

Cache Control传输给MEM的数据和信号：

- we：写使能（当写的地址不在cache中，直接写内存，不写Cache）
- rw_addr：读写地址

MEM给Cache Control的数据：

- mem_data：读到的数据（读失效时，先直接从内存中读取数据，同时将数据写给cache）

核心代码

Cache替换算法：LRU

```
if(cache_is_full)begin//替换策略：LRU
    if(counter[{rw_addr[2:1],1'b0}] >= counter[{rw_addr[2:1],1'b1}])begin
        available_bit = 0;
    end
    else available_bit = 1;
end
```

被替换的块要么是组中的第一块，要么是第二块，将第一块和第二块的未使用次数作比较，选择大的那个作为替换块。

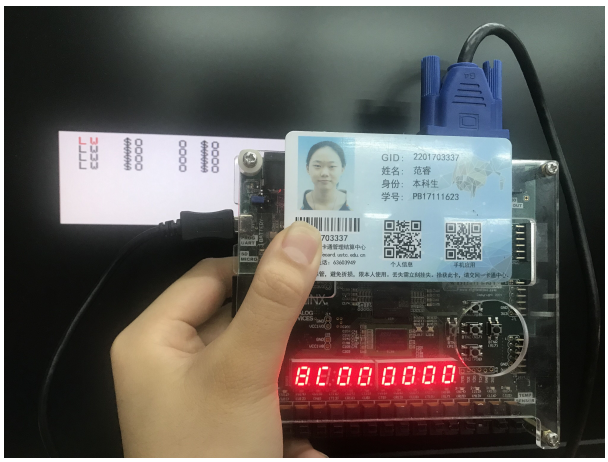
仿真

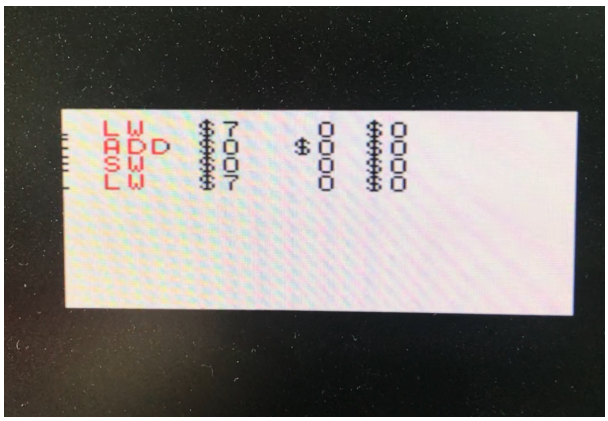
Cache

可以看counter的替换过程，比较直观。我给的仿真是读一段连续的地址，所以counter的某一位是0会持续两个周期。

下载

VGA





第一行是编辑行，后面的行是已经确定的行。

第一张照片是最终结果，但是我只显示除了ADD命令，所以我加了第二张调试过程中的结果，还有些地方没有改的比较干净，但是显示出了LW和SW。

实验总结

在本次实验中，我实现了CPU的Cache改进和VGA显示字母的功能。调试过程真的很崩溃，但是觉得代码写的更加熟练，而且拥有了可以把脑子里想的东西实际实现的能力。其中我仍然有一个很困惑的问题，虽然我已经解决了，但我仍然不知道为什么要这样解决。就是我在创建字模的时候，需要把这个字的排列在一个8*8的空间里写出来，如果这个字符过于在这个8x8的空间中过于靠右，本该显示在最左边的东西会跑到最右边，但是放在左边就没问题。另外，我在前面说的第二个cache的状态图是我在写vga之后才发现的问题然后改动的。因为vga的像素时钟非常快，所以有可能出现在cpu的取指时间（read有效）时，我vga去写内存（摁下确定是写内存），这样就根本写不进去，所以我就意识到读写优先级的问题，然后对我的状态机进行了改动。我认为这次的综合实验对我是一个很大的锻炼。让我对cache的理解更加深刻，对vga的显示更加熟练。

这两学期fpga的编程可算（划掉）结束了，崩溃是真的崩溃过，但是the only way out fire is through it，我很庆幸我没有放弃，也要谢谢所有帮过我debug的老师和同学们，没有他们，我真的，会感到十分无助。

代码

```
module cache_control(
    input clk,
    input rst,
    input read,
    input write,
    input [7:0]rw_addr,
    input [31:0]cpudata,
    input [7:0]mem_addr_ddu,

    output reg [31:0]datatocpu,
    output [31:0]mem_data_ddu
    );

    parameter start = 4'b000;
    parameter WaitforSignal = 4'b001;
    parameter ReadHit = 4'b010;
    parameter ReadMiss_1 = 4'b011;
    parameter ReadMiss_2 = 4'b110;
    parameter WriteHit = 4'b100;
    parameter WriteMiss = 4'b101;
    parameter ReadReturn = 4'b111;
    parameter WriteReturn = 4'b1000;

    reg [3:0]current_state;
    reg [3:0]next_state;

    always@(posedge clk or posedge rst)begin
        if(rst)current_state <= start;
        else current_state <= next_state;
    end

    reg [5:0] tag [0:7];//{分配位，tag}
```

```
    wire [7:0] counter [0:7];
    reg [7:0] counter_next [0:7];
    wire [1:0]group_number;//组号
    wire offset;
    wire [4:0]current_tag;
    reg [3:0]cache_addr;
    wire hit,hit0,hit1;
    wire [31:0]cache_data;
    reg cache_we;
    reg mem_we;
    reg cpuwrite;
    wire [31:0]mem_data;
    wire cache_is_full;
    reg available_bit;
    reg [7:0]cache_read_mem_addr;

    assign hit = hit0 || hit1;

    assign current_tag = rw_addr[7:3];
    assign group_number = rw_addr[2:1];
    assign offset = rw_addr[0];
    assign hit0 = (tag[{group_number,1'b0}] == {1'b1, current_tag});
    assign hit1 = (tag[{group_number,1'b1}] == {1'b1, current_tag});

    always@(posedge clk)begin
        case(current_state)
            start,WaitforSignal,WriteMiss,ReadMiss_2:begin
                counter_next[0] = counter[0];
                counter_next[1] = counter[1];
                counter_next[2] = counter[2];
                counter_next[3] = counter[3];
                counter_next[4] = counter[4];
                counter_next[5] = counter[5];
                counter_next[6] = counter[6];
                counter_next[7] = counter[7];
            end
            ReadHit,WriteHit:begin
                counter_next[0] = counter[0] + 1;
                counter_next[1] = counter[1] + 1;
                counter_next[2] = counter[2] + 1;
                counter_next[3] = counter[3] + 1;
                counter_next[4] = counter[4] + 1;
                counter_next[5] = counter[5] + 1;
                counter_next[6] = counter[6] + 1;
                counter_next[7] = counter[7] + 1;
                if(hit0) counter_next[{group_number,1'b0}] = 0;
                else if(hit1) counter_next[{group_number,1'b1}] = 0;
            end
            ReadMiss_1:begin
                counter_next[0] = counter[0] + 1;
                counter_next[1] = counter[1] + 1;
                counter_next[2] = counter[2] + 1;
                counter_next[3] = counter[3] + 1;
```

```
                counter_next[4] = counter[4] + 1;
                counter_next[5] = counter[5] + 1;
                counter_next[6] = counter[6] + 1;
                counter_next[7] = counter[7] + 1;
                counter_next[{group_number,available_bit}] = 0;
            end
        endcase
    end

    assign counter[0] =  (current_state == start) ? 8'b0 : counter_next[0];
    assign counter[1] =  (current_state == start) ? 8'b0 : counter_next[1];
    assign counter[2] =  (current_state == start) ? 8'b0 : counter_next[2];
    assign counter[3] =  (current_state == start) ? 8'b0 : counter_next[3];
    assign counter[4] =  (current_state == start) ? 8'b0 : counter_next[4];
    assign counter[5] =  (current_state == start) ? 8'b0 : counter_next[5];
    assign counter[6] =  (current_state == start) ? 8'b0 : counter_next[6];
    assign counter[7] =  (current_state == start) ? 8'b0 : counter_next[7];

    always@(*)begin
        case(current_state)
            start: next_state = WaitforSignal;
            WaitforSignal:begin
                if(!read && !write)next_state = WaitforSignal;
                else if(write)begin
                    if(hit)next_state = WriteHit;
                    else next_state = WriteMiss;
                end
                else if(read)begin
                    if(hit)next_state = ReadHit;//hit还没查
                    else next_state = ReadMiss_1;
                end
                else next_state = start;
            end
            ReadHit:next_state = ReadReturn;
            ReadMiss_1:next_state = ReadMiss_2;
            WriteHit:next_state = WriteReturn;
            WriteMiss:next_state = WriteReturn;
            ReadMiss_2:next_state = ReadReturn;
            ReadReturn:begin
                if(read && !write) next_state = ReadReturn;
                else next_state = WaitforSignal;
            end
            WriteReturn:begin
                if(read) begin//write完了，若要read，去read
                    if(hit) next_state = ReadHit;
                    else next_state = ReadMiss_1;
                end
                else if(!write) next_state = WaitforSignal;
                else next_state = WriteReturn;
            end
        endcase
    end

```

```
    always@(*)begin
        case(current_state)
            start:begin
                tag[0] = 0;tag[1] = 0;tag[2] = 0;tag[3] = 0;
                tag[4] = 0;tag[5] = 0;tag[6] = 0;tag[7] = 0;
            end
            WaitforSignal:begin
                cache_we = 0;
                mem_we = 0;
                cpuwrite = 0;
            end
            ReadHit:begin//算cache_addr;cache读到的data给cpu;计算访问次数
                if(hit0) cache_addr = {group_number,1'b0,offset};
                else if(hit1) cache_addr = {group_number,1'b1,offset};
                datatocpu = cache_data;
            end
            ReadMiss_1:begin//cache满了：更新策略；cache没满：放进cache里，mem读到的data给
cpu
                cache_we = 1;
                cpuwrite = 0;
                cache_addr = {group_number,available_bit,1'b0};//块中第0位
                cache_read_mem_addr = {rw_addr[7:1],1'b0};
                datatocpu = mem_data;

                if(cache_is_full)begin//替换策略：LRU
                    if(counter[{rw_addr[2:1],1'b0}] >=
counter[{rw_addr[2:1],1'b1}])begin
                        available_bit = 0;
                    end
                    else available_bit = 1;
                    end
                    else begin
                    if(tag[{group_number,1'b0}][5]==0)begin//组中第一块可用
                        available_bit = 0;
                    end
                    else if(tag[{group_number,1'b1}][5]==0)begin//组中第二块可用
                        available_bit = 1;
                    end
                end

            end
            ReadMiss_2:begin
                cache_we = 1;
                cpuwrite = 0;
                cache_addr = {group_number,available_bit,1'b1};//块中第1位
                cache_read_mem_addr = {rw_addr[7:1],1'b1};
                tag[{group_number,available_bit}] = {1'b1,current_tag};
            end
            WriteHit:begin//写直达：写cache，写内存
                cache_we = 1;
                cpuwrite = 1;
                mem_we = 1;
```

```
                cache_addr = hit0 ? {rw_addr[2:1], 1'b0, offset} : {rw_addr[2:1], 1'b1, offset};
                if(hit0)begin
                    tag[{group_number,1'b0}] = {1'b1, current_tag};
                end
                else if(hit1)begin
                    tag[{group_number,1'b1}] = {1'b1, current_tag};
                end
            end
            WriteMiss:begin//写回：只写mem
                mem_we = 1;
            end
            ReadReturn,WriteReturn:begin
                cache_we = 0;
                mem_we = 0;
                cpuwrite = 0;
            end
        endcase
    end
    //若本组内tag的最高位都是1，则cache满
    assign cache_is_full = (tag[{group_number,1'b0}][5]) && (tag[{group_number,1'b1}][5]);

    cache_memory_module cache_dut(
        .clk(clk),
        .rst(rst),
        .cache_addr(cache_addr),
        .cache_data(cache_data),
        .cpuwrite(cpuwrite),
        .cpudata(cpudata),
        .mem_addr(cache_read_mem_addr),
        .we(cache_we)
        );

    dist_mem_gen_0 cache_control_mem(.a(rw_addr),.d(cpudata),.dpra(mem_addr_ddu),.we(mem_we),.spo(mem_data),.dpo(mem_data_ddu),.clk(clk));
endmodule
```

```
module cache_memory_module(
    input clk,
    input rst,
    input [3:0]cache_addr,//读写地址
    input we,//写使能
    input [7:0]mem_addr,//cache读mem的地址
    input cpuwrite,//写的数据是否来源于cpu
    input [31:0]cpudata,//cpu写的数据

    output [31:0]cache_data//cache中读到的数据
    );

    reg [31:0]cache [0:15];
    wire [31:0]mem_data;//调mem
```

```
15
16      always@(posedge clk or posedge rst)begin
17          if(rst)begin
18              cache[0]<=0;cache[1]<=0;cache[2]<=0;cache[3]<=0;
19              cache[4]<=0;cache[5]<=0;cache[6]<=0;cache[7]<=0;
20              cache[8]<=0;cache[9]<=0;cache[10]<=0;cache[11]<=0;
21              cache[12]<=0;cache[13]<=0;cache[14]<=0;cache[15]<=0;
22          end
23          else begin
24              if(we)begin
25                  if(cpuwrite)cache[cache_addr] = cpudata;//writehit
26                  else cache[cache_addr] = mem_data;//readmiss
27              end
28          end
29      end
30      assign cache_data = cache[cache_addr];
31      dist_mem_gen_0 mem_cache(.a(mem_addr),
  .clk(clk),.spo(mem_data),.d(0),.dpra(mem_addr),.dpo(),.we(0));
32
33  endmodule
```

```
1   module display_word(
2       input clk,
3       input rst,
4       input [11:0]x_cnt,
5       input [11:0]y_cnt,
6       input vga_de,
7       input up,down,left,right,
8       input enter,
9
10      output reg [3:0] vga_r, //屏幕显示 (r)
11      output [3:0] vga_g, //屏幕显示 (g)
12      output [3:0] vga_b,  //屏幕显示 (b)
13      output reg [31:0]instruction
14      );
15
16      parameter OP = 2'b00;
17      parameter RD = 2'b01;
18      parameter RS = 2'b10;
19      parameter RT = 2'b11;
20
21      parameter ADD = 3'b000;
22      parameter LW = 3'b001;
23      parameter SW = 3'b010;
24
25      parameter SPACE = 6'b11_1110;
26      parameter A = 6'b00_1010;
27      parameter D = 6'b00_1101;
28      parameter DOLLAR = 6'b11_1111;
29      parameter T = 6'b01_1101;
30      parameter ZERO = 6'b00_0000;
31      parameter S = 6'b01_1100;
32      parameter ONE = 6'b00_0001;
```

```
    parameter TWO = 6'b00_0010;
    parameter L = 6'b01_0101;
    parameter W = 6'b10_0000;
    parameter THREE = 6'b00_0011;
    parameter FOUR = 6'b00_0100;
    parameter FIVE = 6'b00_0101;
    parameter SIX = 6'b00_0110;
    parameter SEVEN = 6'b00_0111;
    parameter EIGHT =  6'b00_1000;
    parameter NINE = 6'b00_1001;



    reg [5:0]Line0 [0:19];
    reg [5:0]Line1 [0:19];
    reg [5:0]Line2 [0:19];
    reg [5:0]Line3 [0:19];
    reg [5:0]Line4 [0:19];
    reg [5:0]Line5 [0:19];
    reg [5:0]Line6 [0:19];
    reg [5:0]Line7 [0:19];
    reg [5:0]Line8 [0:19];
    reg [5:0]Line9 [0:19];
    reg [1:0]choose;
    reg [2:0]op;
    reg [4:0]rd;
    reg [4:0]rs;
    reg [4:0]rt;
    wire [4:0]rd_up_next,rd_down_next;
    wire [4:0]rs_up_next,rs_down_next;
    wire [4:0]rt_up_next,rt_down_next;
    wire [11:0]LINE0_67_UP,LINE0_1011_UP,LINE0_1314_UP;
    wire [11:0]LINE0_67_DOWN,LINE0_1011_DOWN,LINE0_1314_DOWN;
    reg [3:0]current_line;

    always@(*)begin
        case(op)
            ADD:instruction = {6'b000000,rs,rt,rd,5'b00000,6'b100000};
            LW: instruction = {6'b100011,rt,rd,11'b0,rs};
            SW: instruction = {6'b101011,rt,rd,11'b0,rs};
            default:;
        endcase
    end

    always@(posedge rst,posedge up,posedge down)begin
        if(rst)begin
            op <= ADD;
            rd <= 0;
            rs <= 0;
            rt <= 0;
            Line0[0]<=6'b11_1110;Line0[1]<=A;/*a*/Line0[2]<=D;/*d*/Line0[3]
<=D;/*d*/Line0[4]<=6'b11_1110;
```

```
            Line0[5]<=DOLLAR;/*$*/Line0[6]<=ZERO;Line0[7]<=SPACE;Line0[8]
<=SPACE;Line0[9]<=DOLLAR;/*$*/
            Line0[10]<=ZERO;/*0*/Line0[11]<=SPACE;/*0*/Line0[12]<=DOLLAR;Line0[13]
<=ZERO;/*0*/Line0[14]<=6'b11_1110;
            Line0[15]<=6'b11_1110;Line0[16]<=6'b11_1110;Line0[17]
<=6'b11_1110;Line0[18]<=6'b11_1110;Line0[19]<=6'b11_1110;
        end
        else if(up)begin
            if(choose == OP)begin
                case(op)
                        ADD:begin
                            Line0[1] <= S;Line0[2] <= W;Line0[3] <= SPACE;Line0[9] <=
SPACE;
                            op <= SW;
                        end
                        LW:begin
                            Line0[1] <= A;Line0[2] <= D;Line0[3] <= D;Line0[9] <=
DOLLAR;
                            op <= ADD;
                        end
                        SW: begin
                            Line0[1] <= L;
                            op <= LW;
                        end
                        default:;
                    endcase
            end
            else if(choose == RD)begin
                rd = rd_up_next;
                {Line0[6],Line0[7]}=LINE0_67_UP;
            end
            else if(choose == RS)begin
                rs = rs_up_next;
                {Line0[10],Line0[11]}=LINE0_1011_UP;
            end
            else if(choose == RT)begin
                rt = rt_up_next;
               {Line0[13],Line0[14]}=LINE0_1314_UP;
            end
            else;
        end
        else if(down)begin
            if(choose == OP)begin
                case(op)
                    ADD:begin
                        Line0[1] <= L;Line0[2] <= W;Line0[3] <= SPACE;Line0[9] <=
SPACE;
                        op <= LW;
                    end
                    LW:begin
                        Line0[1] <= S;
                        op <= SW;
                    end
```

```
                    SW: begin
                        Line0[1] <= A;Line0[2] <= D;Line0[3] <= D;Line0[9] <= DOLLAR;
                        op <= ADD;
                    end
                    default:;
                endcase
            end
            else if(choose == RD)begin
                rd = rd_down_next;
                {Line0[6],Line0[7]}=LINE0_67_DOWN;
            end
            else if(choose == RS)begin
                rs = rs_down_next;
               {Line0[10],Line0[11]}=LINE0_1011_DOWN;
            end
            else if(choose == RT) begin
                rt = rt_down_next;
                {Line0[13],Line0[14]}=LINE0_1314_DOWN;
            end
            else;
        end
        else;
    end

    assign rd_up_next = rd-1;
    assign rs_up_next = rs-1;
    assign rt_up_next = rt-1;
    assign rd_down_next = rd+1;
    assign rs_down_next = rs+1;
    assign rt_down_next = rt+1;

    word_dictionary dic_dut0(.data(rd_up_next), .display_figure(LINE0_67_UP));
    word_dictionary dic_dut1(.data(rs_up_next), .display_figure(LINE0_1011_UP));
    word_dictionary dic_dut2(.data(rt_up_next), .display_figure(LINE0_1314_UP));

    word_dictionary dic_dut3(.data(rd_down_next), .display_figure(LINE0_67_DOWN));
    word_dictionary dic_dut4(.data(rs_down_next), .display_figure(LINE0_1011_DOWN));
    word_dictionary dic_dut5(.data(rt_down_next), .display_figure(LINE0_1314_DOWN));

    always@(posedge rst or posedge left or posedge right)begin
        if(rst) choose <= OP;
        else if(left)begin
            case(choose)
                OP:choose <= RT;
                RD:choose <= OP;
                RS:choose <= RD;
                RT:choose <= RS;
                default:;
            endcase
        end
        else if(right)begin
            case(choose)
                OP:choose <= RD;
```

```
                RD:choose <= RS;
                RS:choose <= RT;
                RT:choose <= OP;
                default:;
            endcase
        end
        else ;
    end

    wire [2:0]x_r,y_r;
    wire [7:0]x_grp,y_grp;
    reg [5:0]current_data;
    wire [7:0]p [0:7];
    reg enable;

    assign {x_grp,x_r} = x_cnt - 11'd5;
    assign {y_grp,y_r} = y_cnt - 11'd5;

    always@(posedge clk)begin
        case(y_grp)
            0:current_data <= Line0[x_grp];
            1:current_data <= Line1[x_grp];
            2:current_data <= Line2[x_grp];
            3:current_data <= Line3[x_grp];
            4:current_data <= Line4[x_grp];
            5:current_data <= Line5[x_grp];
            6:current_data <= Line6[x_grp];
            7:current_data <= Line7[x_grp];
            8:current_data <= Line8[x_grp];
            9:current_data <= Line9[x_grp];
            default:;
        endcase
    end

    always@(posedge clk)begin
        enable = p[y_r][x_r];
    end

    always@(*)begin
        if(vga_de)begin
            if(enable && (y_grp>=0 && y_grp <= 9) && (x_grp >=0 && x_grp <= 19))begin
                case(choose)
                    OP:begin
                        if((x_grp == 8'd1 || x_grp == 8'd2 || x_grp == 8'd3) && y_grp
== 8'd0) vga_r = 4'b1111;
                        else vga_r = 4'b0;
                    end
                    RD:begin
                        if((x_grp == 8'd5 || x_grp == 8'd6 || x_grp == 8'd7) && y_grp
== 8'd0) vga_r = 4'b1111;
                        else vga_r = 4'b0;
                    end
                    RS:begin
```

```
                        if((x_grp == 8'd9 || x_grp == 8'd10 || x_grp == 8'd11) &&
y_grp == 8'd0) vga_r = 4'b1111;
                        else vga_r = 4'b0;
                    end
                    RT:begin
                        if((x_grp == 8'd13 || x_grp == 8'd14) && y_grp == 8'd0) vga_r
= 4'b1111;
                        else vga_r = 4'b0;
                    end
                endcase
            end
            else vga_r = 4'b1111;
        end
        else vga_r = 4'b0;
    end

    assign vga_g = vga_de ? ((enable && (y_grp>=0 && y_grp <= 9) && (x_grp >=0 &&
x_grp <= 19)) ? 4'b0 : 4'b1111) : 4'b0;
    assign vga_b = vga_de ? ((enable && (y_grp>=0 && y_grp <= 9) && (x_grp >=0 &&
x_grp <= 19)) ? 4'b0 : 4'b1111) : 4'b0;

    cur_data cur_data_dut(
        .clk(clk),
        .rst(rst),
        .data(current_data),
        .col0(p[0]),
        .col1(p[1]),
        .col2(p[2]),
        .col3(p[3]),
        .col4(p[4]),
        .col5(p[5]),
        .col6(p[6]),
        .col7(p[7])
         );

     always@(posedge rst or posedge enter)begin
        if(rst)begin
        current_line <= 0;
        Line1[0]<=6'b11_1110;Line1[1]<=6'b11_1110;Line1[2]<=6'b11_1110;Line1[3]
<=6'b11_1110;Line1[4]<=6'b11_1110;
        Line1[5]<=6'b11_1110;Line1[6]<=6'b11_1110;Line1[7]<=6'b11_1110;Line1[8]
<=6'b11_1110;Line1[9]<=6'b11_1110;
        Line1[10]<=6'b11_1110;Line1[11]<=6'b11_1110;Line1[12]<=6'b11_1110;Line1[13]
<=6'b11_1110;Line1[14]<=6'b11_1110;
        Line1[15]<=6'b11_1110;Line1[16]<=6'b11_1110;Line1[17]<=6'b11_1110;Line1[18]
<=6'b11_1110;Line1[19]<=6'b11_1110;

        Line2[0]<=6'b11_1110;Line2[1]<=6'b11_1110;Line2[2]<=6'b11_1110;Line2[3]
<=6'b11_1110;Line2[4]<=6'b11_1110;
        Line2[5]<=6'b11_1110;Line2[6]<=6'b11_1110;Line2[7]<=6'b11_1110;Line2[8]
<=6'b11_1110;Line2[9]<=6'b11_1110;
        Line2[10]<=6'b11_1110;Line2[11]<=6'b11_1110;Line2[12]<=6'b11_1110;Line2[13]
<=6'b11_1110;Line2[14]<=6'b11_1110;
```

```
        Line2[15]<=6'b11_1110;Line2[16]<=6'b11_1110;Line2[17]<=6'b11_1110;Line2[18]<=6'b11_1110;Line2[19]<=6'b11_1110;

        Line3[0]<=6'b11_1110;Line3[1]<=6'b11_1110;Line3[2]<=6'b11_1110;Line3[3]<=6'b11_1110;Line3[4]<=6'b11_1110;
        Line3[5]<=6'b11_1110;Line3[6]<=6'b11_1110;Line3[7]<=6'b11_1110;Line3[8]<=6'b11_1110;Line3[9]<=6'b11_1110;
        Line3[10]<=6'b11_1110;Line3[11]<=6'b11_1110;Line3[12]<=6'b11_1110;Line3[13]<=6'b11_1110;Line3[14]<=6'b11_1110;
        Line3[15]<=6'b11_1110;Line3[16]<=6'b11_1110;Line3[17]<=6'b11_1110;Line3[18]<=6'b11_1110;Line3[19]<=6'b11_1110;

        Line4[0]<=6'b11_1110;Line4[1]<=6'b11_1110;Line4[2]<=6'b11_1110;Line4[3]<=6'b11_1110;Line4[4]<=6'b11_1110;
        Line4[5]<=6'b11_1110;Line4[6]<=6'b11_1110;Line4[7]<=6'b11_1110;Line4[8]<=6'b11_1110;Line4[9]<=6'b11_1110;
        Line4[10]<=6'b11_1110;Line4[11]<=6'b11_1110;Line4[12]<=6'b11_1110;Line4[13]<=6'b11_1110;Line4[14]<=6'b11_1110;
        Line4[15]<=6'b11_1110;Line4[16]<=6'b11_1110;Line4[17]<=6'b11_1110;Line4[18]<=6'b11_1110;Line4[19]<=6'b11_1110;

        Line5[0]<=6'b11_1110;Line5[1]<=6'b11_1110;Line5[2]<=6'b11_1110;Line5[3]<=6'b11_1110;Line5[4]<=6'b11_1110;
        Line5[5]<=6'b11_1110;Line5[6]<=6'b11_1110;Line5[7]<=6'b11_1110;Line5[8]<=6'b11_1110;Line5[9]<=6'b11_1110;
        Line5[10]<=6'b11_1110;Line5[11]<=6'b11_1110;Line5[12]<=6'b11_1110;Line5[13]<=6'b11_1110;Line5[14]<=6'b11_1110;
        Line5[15]<=6'b11_1110;Line5[16]<=6'b11_1110;Line5[17]<=6'b11_1110;Line5[18]<=6'b11_1110;Line5[19]<=6'b11_1110;

        Line6[0]<=6'b11_1110;Line6[1]<=6'b11_1110;Line6[2]<=6'b11_1110;Line6[3]<=6'b11_1110;Line6[4]<=6'b11_1110;
        Line6[5]<=6'b11_1110;Line6[6]<=6'b11_1110;Line6[7]<=6'b11_1110;Line6[8]<=6'b11_1110;Line6[9]<=6'b11_1110;
        Line6[10]<=6'b11_1110;Line6[11]<=6'b11_1110;Line6[12]<=6'b11_1110;Line6[13]<=6'b11_1110;Line6[14]<=6'b11_1110;
        Line6[15]<=6'b11_1110;Line6[16]<=6'b11_1110;Line6[17]<=6'b11_1110;Line6[18]<=6'b11_1110;Line6[19]<=6'b11_1110;

        Line7[0]<=6'b11_1110;Line7[1]<=6'b11_1110;Line7[2]<=6'b11_1110;Line7[3]<=6'b11_1110;Line7[4]<=6'b11_1110;
        Line7[5]<=6'b11_1110;Line7[6]<=6'b11_1110;Line7[7]<=6'b11_1110;Line7[8]<=6'b11_1110;Line7[9]<=6'b11_1110;
        Line7[10]<=6'b11_1110;Line7[11]<=6'b11_1110;Line7[12]<=6'b11_1110;Line7[13]<=6'b11_1110;Line7[14]<=6'b11_1110;
        Line7[15]<=6'b11_1110;Line7[16]<=6'b11_1110;Line7[17]<=6'b11_1110;Line7[18]<=6'b11_1110;Line7[19]<=6'b11_1110;

        Line8[0]<=6'b11_1110;Line8[1]<=6'b11_1110;Line8[2]<=6'b11_1110;Line8[3]<=6'b11_1110;Line8[4]<=6'b11_1110;
        Line8[5]<=6'b11_1110;Line8[6]<=6'b11_1110;Line8[7]<=6'b11_1110;Line8[8]<=6'b11_1110;Line8[9]<=6'b11_1110;
```

```
        Line8[10]<=6'b11_1110;Line8[11]<=6'b11_1110;Line8[12]<=6'b11_1110;Line8[13]
<=6'b11_1110;Line8[14]<=6'b11_1110;
        Line8[15]<=6'b11_1110;Line8[16]<=6'b11_1110;Line8[17]<=6'b11_1110;Line8[18]
<=6'b11_1110;Line8[19]<=6'b11_1110;

        Line9[0]<=6'b11_1110;Line9[1]<=6'b11_1110;Line9[2]<=6'b11_1110;Line9[3]
<=6'b11_1110;Line9[4]<=6'b11_1110;
        Line9[5]<=6'b11_1110;Line9[6]<=6'b11_1110;Line9[7]<=6'b11_1110;Line9[8]
<=6'b11_1110;Line9[9]<=6'b11_1110;
        Line9[10]<=6'b11_1110;Line9[11]<=6'b11_1110;Line9[12]<=6'b11_1110;Line9[13]
<=6'b11_1110;Line9[14]<=6'b11_1110;
        Line9[15]<=6'b11_1110;Line9[16]<=6'b11_1110;Line9[17]<=6'b11_1110;Line9[18]
<=6'b11_1110;Line9[19]<=6'b11_1110;
        end
        else if(enter)begin
            current_line <= current_line+1;
            case(current_line)
                1:begin
                    Line1[1] <= Line0[1];Line1[2] <= Line0[2];Line1[3] <= Line0[3];
                    Line1[5] <= Line0[5];Line1[6] <= Line0[6];Line1[7] <= Line0[7];
                    Line1[9] <= Line0[9];Line1[10] <= Line0[10];Line1[11] <=
Line0[11];
                    Line1[12] <= Line0[12];Line1[13] <= Line0[13];Line1[14] <=
Line0[14];
                end
                2:begin
                    Line2[1] <= Line0[1];Line2[2] <= Line0[2];Line2[3] <= Line0[3];
                    Line2[5] <= Line0[5];Line2[6] <= Line0[6];Line2[7] <= Line0[7];
                    Line2[9] <= Line0[9];Line2[10] <= Line0[10];Line2[11] <=
Line0[11];
                    Line2[12] <= Line0[12];Line2[13] <= Line0[13];Line2[14] <=
Line0[14];
                end
                3:begin
                    Line3[1] <= Line0[1];Line3[2] <= Line0[2];Line3[3] <= Line0[3];
                    Line3[5] <= Line0[5];Line3[6] <= Line0[6];Line3[7] <= Line0[7];
                    Line3[9] <= Line0[9];Line3[10] <= Line0[10];Line3[11] <=
Line0[11];
                    Line3[12] <= Line0[12];Line3[13] <= Line0[13];Line3[14] <=
Line0[14];
                end
                4:begin
                    Line4[1] <= Line0[1];Line4[2] <= Line0[2];Line4[3] <= Line0[3];
                    Line4[5] <= Line0[5];Line4[6] <= Line0[6];Line4[7] <= Line0[7];
                    Line4[9] <= Line0[9];Line4[10] <= Line0[10];Line4[11] <=
Line0[11];
                    Line4[12] <= Line0[12];Line4[13] <= Line0[13];Line4[14] <=
Line0[14];
                end
                5:begin
                    Line5[1] <= Line0[1];Line5[2] <= Line0[2];Line5[3] <= Line0[3];
                    Line5[5] <= Line0[5];Line5[6] <= Line0[6];Line5[7] <= Line0[7];
```

```
                    Line5[9] <= Line0[9];Line5[10] <= Line0[10];Line5[11] <=
Line0[11];
                    Line5[12] <= Line0[12];Line5[13] <= Line0[13];Line5[14] <=
Line0[14];
                end
                6:begin
                    Line6[1] <= Line0[1];Line6[2] <= Line0[2];Line6[3] <= Line0[3];
                    Line6[5] <= Line0[5];Line6[6] <= Line0[6];Line6[7] <= Line0[7];
                    Line6[9] <= Line0[9];Line6[10] <= Line0[10];Line6[11] <=
Line0[11];
                    Line6[12] <= Line0[12];Line6[13] <= Line0[13];Line6[14] <=
Line0[14];
                end
                7:begin
                    Line7[1] <= Line0[1];Line7[2] <= Line0[2];Line7[3] <= Line0[3];
                    Line7[5] <= Line0[5];Line7[6] <= Line0[6];Line7[7] <= Line0[7];
                    Line7[9] <= Line0[9];Line7[10] <= Line0[10];Line7[11] <=
Line0[11];
                    Line7[12] <= Line0[12];Line7[13] <= Line0[13];Line7[14] <=
Line0[14];
                end
                8:begin
                    Line8[1] <= Line0[1];Line8[2] <= Line0[2];Line8[3] <= Line0[3];
                    Line8[5] <= Line0[5];Line8[6] <= Line0[6];Line8[7] <= Line0[7];
                    Line8[9] <= Line0[9];Line8[10] <= Line0[10];Line8[11] <=
Line0[11];
                    Line8[12] <= Line0[12];Line8[13] <= Line0[13];Line8[14] <=
Line0[14];
                end
                9:begin
                    Line9[1] <= Line0[1];Line9[2] <= Line0[2];Line9[3] <= Line0[3];
                    Line9[5] <= Line0[5];Line9[6] <= Line0[6];Line9[7] <= Line0[7];
                    Line9[9] <= Line0[9];Line9[10] <= Line0[10];Line9[11] <=
Line0[11];
                    Line9[12] <= Line0[12];Line9[13] <= Line0[13];Line9[14] <=
Line0[14];
                end
                default:;
            endcase
        end
        else;
    end
endmodule
```